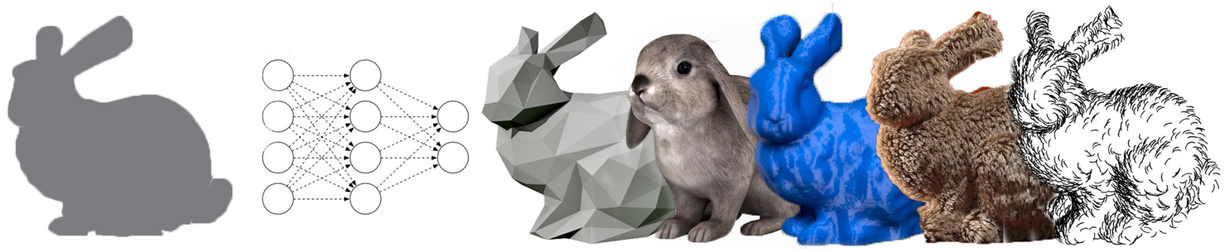
SIGGRAPH ASIA 2018 TOKYO



CreativeAI

Deep Learning for Graphics

Niloy Mitra UCL

Iasonas Kokkinos UCL

Paul Guerrero UCL

Nils Thuerey TUM

Tobias Ritschel UCL

University College London

Technische Universität München

http://geometry.cs.ucl.ac.uk/creativeai/

People

Niloy Mitra

People

Niloy Mitra

Iasonas Kokkinos

People

Niloy Mitra

Iasonas Kokkinos

Paul Guerrero

People

Niloy Mitra

Iasonas Kokkinos

Paul Guerrero

Nils Thuerey

People

Niloy Mitra

Iasonas Kokkinos

Paul Guerrero

Nils Thuerey

Tobias Ritschel

People

Niloy Mitra

Iasonas Kokkinos

Paul Guerrero

Nils Thuerey

Tobias Ritschel

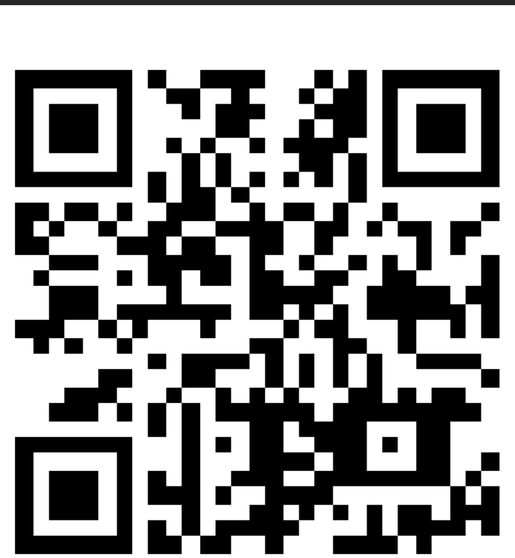
Timetable

			Niloy	Paul	Nils
Theory + Basics	Introduction	2:15 pm	X	X	X
	Machine Learning Basics	~ 2:25 pm	X		
	Neural Network Basics	~ 2:55 pm			X
	Feature Visualization	~ 3:25 pm		X	
	Alternatives to Direct Supervision	~ 3:35 pm		X	
	15 min. break				
State of the Art	Image Domains	4:15 pm		X	
	3D Domains	~ 4:45 pm	X		
	Motion and Physics	~ 5:15 pm			X
	Discussion	~ 5:45 pm	X	X	X

Code Examples

PCA/SVD basis
Linear Regression
Polynomial Regression
Stochastic Gradient Descent vs. Gradient Descent
Multi-layer Perceptron
Edge Filter ‘Network’
Convolutional Network
Filter Visualization
Weight Initialization Strategies
Colorization Network
Autoencoder
Variational Autoencoder
Generative Adversarial Network

http://geometry.cs.ucl.ac.uk/creativeai/



Course Objectives

Course Objectives

- Provide an overview of the popular **ML algorithms** used in CG

Course Objectives

- Provide an overview of the popular **ML algorithms** used in CG

- Provide a quick overview of **theory** and **CG applications**
  - Many extra slides in the course notes + example code

Course Objectives

- Provide an overview of the popular **ML algorithms** used in CG

- Provide a quick overview of **theory** and **CG applications**
  - Many extra slides in the course notes + example code

- Progress in the last 3-5 years has been dramatic
  - We have organized them to help newcomers
  - Discuss the main **challenges and opportunities** specific to CG

Two-way Communication

Two-way Communication

- Our aim is to convey what we found to be relevant so far

- You are invited/encouraged to give feedback

Two-way Communication

- Our aim is to convey what we found to be relevant so far

- You are invited/encouraged to give feedback
  - Speakup. Please send us your criticism/comments/suggestions

Two-way Communication

- Our aim is to convey what we found to be relevant so far

- You are invited/encouraged to give feedback
  - Speakup. Please send us your criticism/comments/suggestions
  - Ask questions, please!

- **Thanks to many people who helped so far with slides/comments**

Representations in CG

Representations in CG

- Images (e.g., pixel grid)

- Volume (e.g., voxel grid)

Representations in CG

- Images (e.g., pixel grid)
- Volume (e.g., voxel grid)
- Meshes (e.g., vertices/edges/faces)

Representations in CG

- Images (e.g., pixel grid)
- Volume (e.g., voxel grid)
- Meshes (e.g., vertices/edges/faces)
- Animation (e.g., skeletal positions over time; cloth dynamics over time)

Representations in CG

- Images (e.g., pixel grid)
- Volume (e.g., voxel grid)
- Meshes (e.g., vertices/edges/faces)
- Animation (e.g., skeletal positions over time; cloth dynamics over time)
- Pointclouds (e.g., point arrays)

Representations in CG

- Images (e.g., pixel grid)
- Volume (e.g., voxel grid)
- Meshes (e.g., vertices/edges/faces)
- Animation (e.g., skeletal positions over time; cloth dynamics over time)
- Pointclouds (e.g., point arrays)
- Physics simulations (e.g., fluid flow over space/time, object-body interaction)

Problems in Computer Graphics

- Feature detection (image features, point features) $\mathbb{R}^{m \times m} \rightarrow \mathbb{Z}$
- Denoising, Smoothing, etc. $\mathbb{R}^{m \times m} \rightarrow \mathbb{R}^{m \times m}$
- Embedding, Distance computation $\mathbb{R}^{m \times m, m \times m} \rightarrow \mathbb{R}^{d}$
- Rendering $\mathbb{R}^{m \times m} \rightarrow \mathbb{R}^{m \times m}$
- Animation $\mathbb{R}^{3m \times t} \rightarrow \mathbb{R}^{3m}$
- Physical simulation $\mathbb{R}^{3m \times t} \rightarrow \mathbb{R}^{3m}$
- Generative models $\mathbb{R}^{d} \rightarrow \mathbb{R}^{m \times m}$

Problems in Computer Graphics

- Feature detection (image features, point features) $\mathbb{R}^{m \times m} \rightarrow \mathbb{Z}$
- Denoising, Smoothing, etc. $\mathbb{R}^{m \times m} \rightarrow \mathbb{R}^{m \times m}$
- Embedding, Distance computation $\mathbb{R}^{m \times m, m \times m} \rightarrow \mathbb{R}^{d}$

analysis

- Rendering $\mathbb{R}^{m \times m} \rightarrow \mathbb{R}^{m \times m}$
- Animation $\mathbb{R}^{3m \times t} \rightarrow \mathbb{R}^{3m}$
- Physical simulation $\mathbb{R}^{3m \times t} \rightarrow \mathbb{R}^{3m}$
- Generative models $\mathbb{R}^{d} \rightarrow \mathbb{R}^{m \times m}$

Problems in Computer Graphics

- Feature detection (image features, point features) $\mathbb{R}^{m \times m} \rightarrow \mathbb{Z}$
- Denoising, Smoothing, etc. $\mathbb{R}^{m \times m} \rightarrow \mathbb{R}^{m \times m}$
- Embedding, Distance computation $\mathbb{R}^{m \times m, m \times m} \rightarrow \mathbb{R}^{d}$

analysis

- Rendering $\mathbb{R}^{m \times m} \rightarrow \mathbb{R}^{m \times m}$
- Animation $\mathbb{R}^{3m \times t} \rightarrow \mathbb{R}^{3m}$
- Physical simulation $\mathbb{R}^{3m \times t} \rightarrow \mathbb{R}^{3m}$
- Generative models $\mathbb{R}^{d} \rightarrow \mathbb{R}^{m \times m}$

synthesis

Goal: Learn a Parametric Function

$$f_\theta : \mathbb{X} \longrightarrow \mathbb{Y}$$

θ: function parameters, $\mathbb{X}$: source domain $\mathbb{Y}$: target domain
these are learned

Goal: Learn a Parametric Function

$$f_\theta : \mathbb{X} \longrightarrow \mathbb{Y}$$

θ: function parameters, $\mathbb{X}$: source domain $\mathbb{Y}$: target domain
these are learned

Examples:

Goal: Learn a Parametric Function

$$f_\theta : \mathbb{X} \longrightarrow \mathbb{Y}$$

θ: function parameters, $\mathbb{X}$: source domain $\mathbb{Y}$: target domain
these are learned

Examples:

Image Classification: $f_\theta : \mathbb{R}^{w \times h \times c} \longrightarrow \{0, 1, \ldots, k-1\}$

$w \times h \times c$: image dimensions k: class count

Goal: Learn a Parametric Function

$$f_\theta : \mathbb{X} \longrightarrow \mathbb{Y}$$

θ: function parameters, $\mathbb{X}$: source domain $\mathbb{Y}$: target domain
these are learned

Examples:

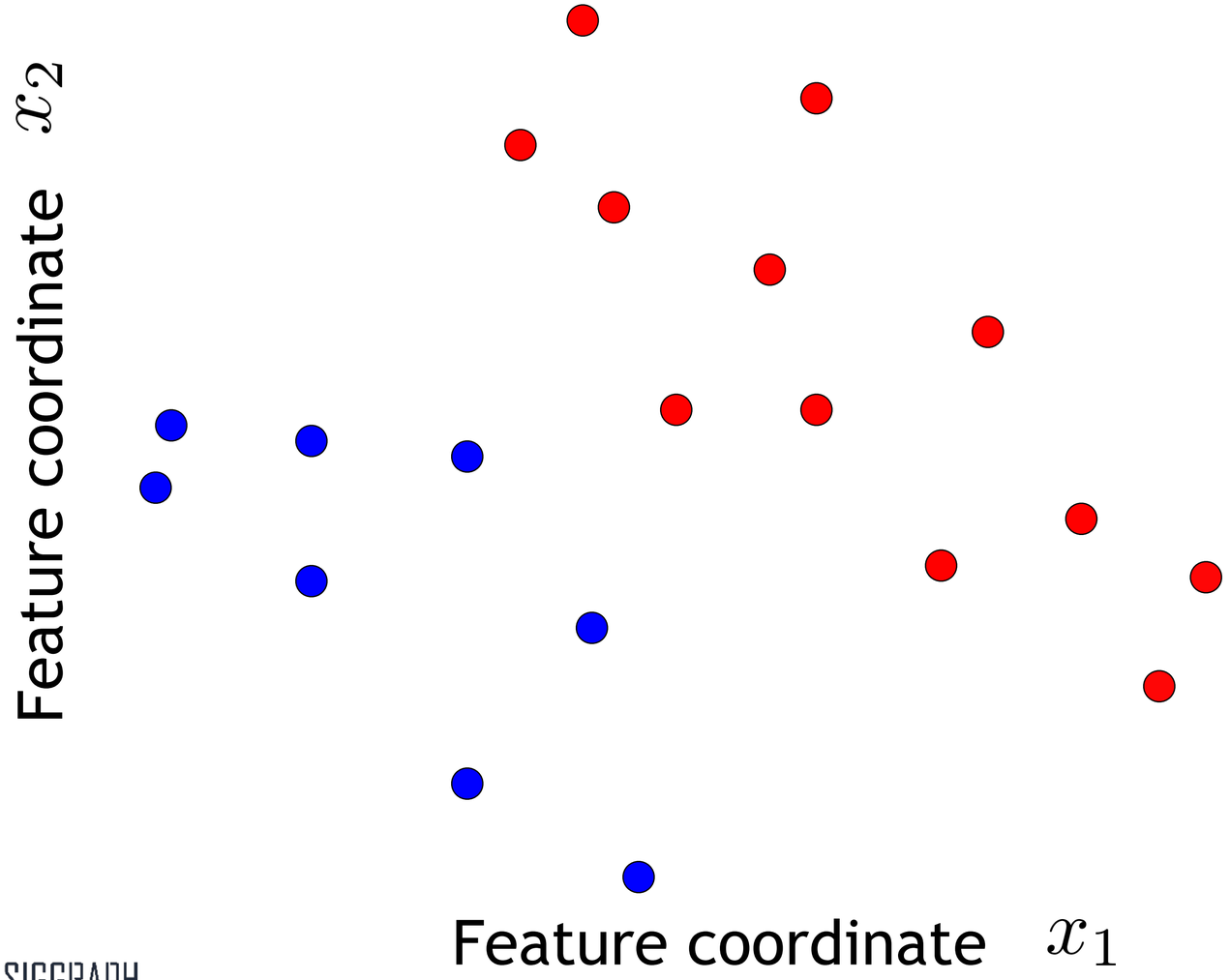
Image Classification: $f_\theta : \mathbb{R}^{w \times h \times c} \longrightarrow \{0, 1, \ldots, k-1\}$

$w \times h \times c$: image dimensions k: class count

Image Synthesis: $f_\theta : \mathbb{R}^{n} \longrightarrow \mathbb{R}^{w \times h \times c}$

n : latent variable count $w \times h \times c$: image dimensions

Machine Learning 101: Linear Classifier

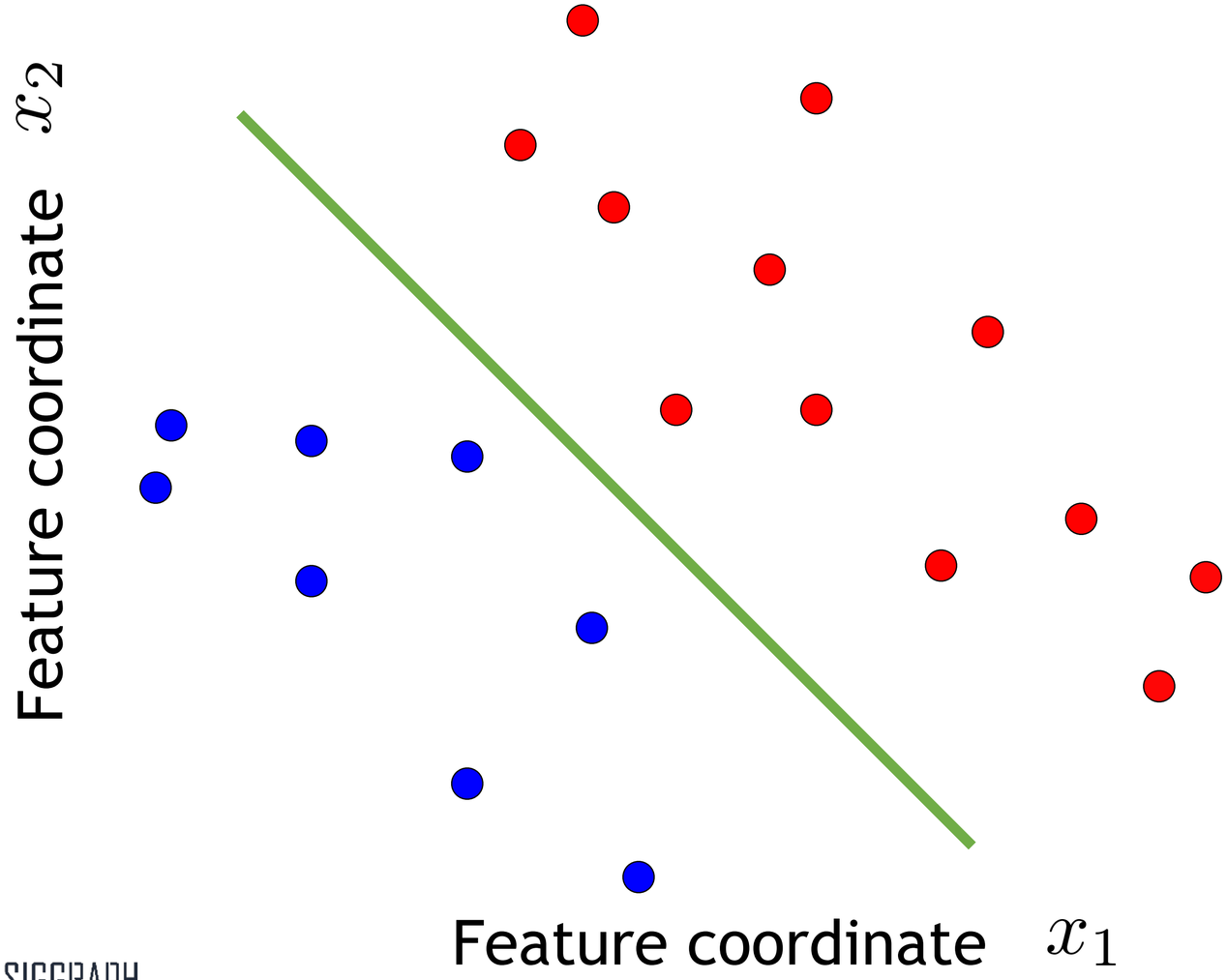


$$f_\theta : \mathbb{R}^n \longrightarrow \{0, 1\}$$

Each data point has a class label:

$$y^i = \begin{cases} 1 & (\bullet) \\ 0 & (\bullet) \end{cases}$$

Machine Learning 101: Linear Classifier

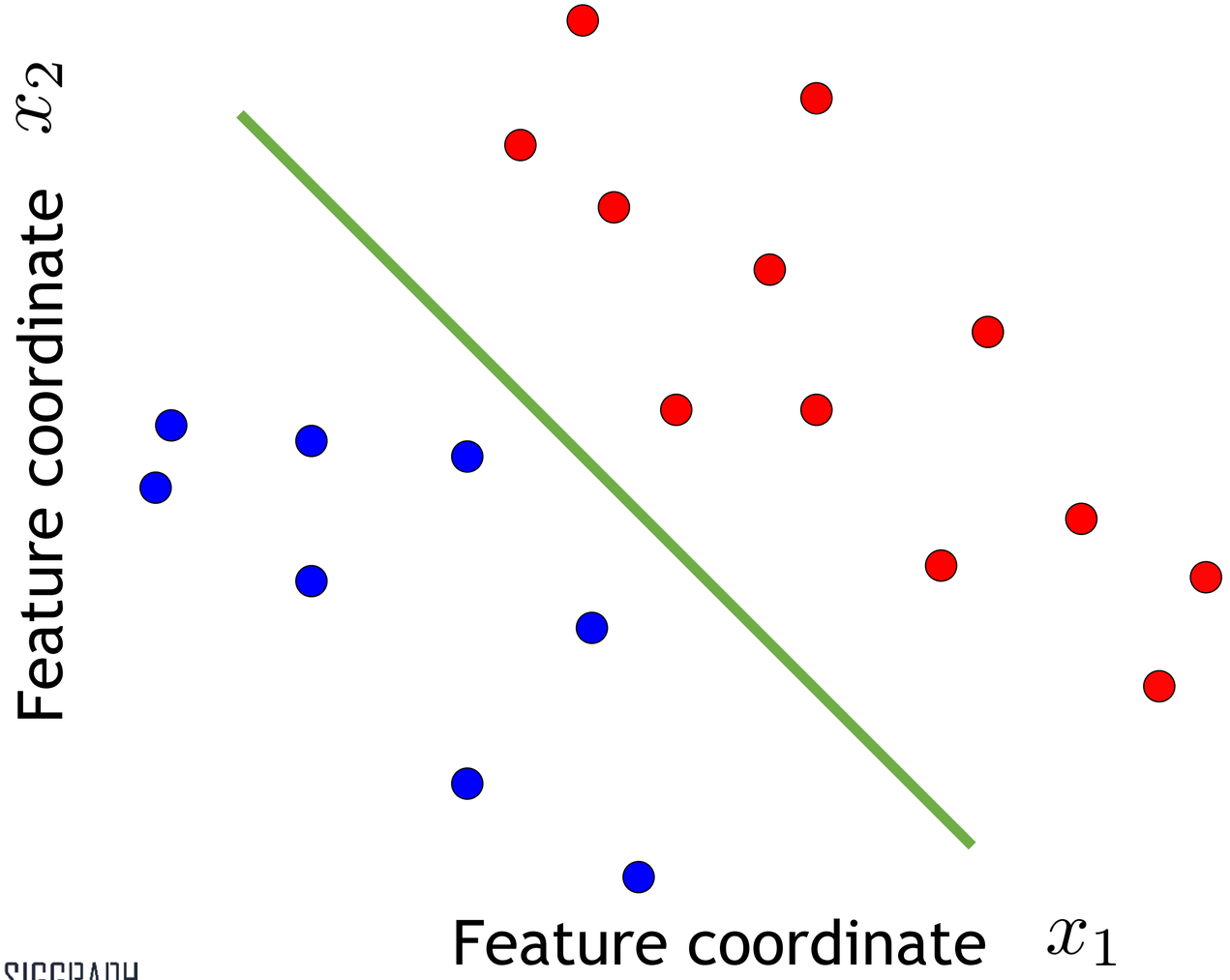


$$f_\theta : \mathbb{R}^n \longrightarrow \{0, 1\}$$

Each data point has a class label:

$$y^i = \begin{cases} 1 & (\bullet) \\ 0 & (\bullet) \end{cases}$$

Machine Learning 101: Linear Classifier



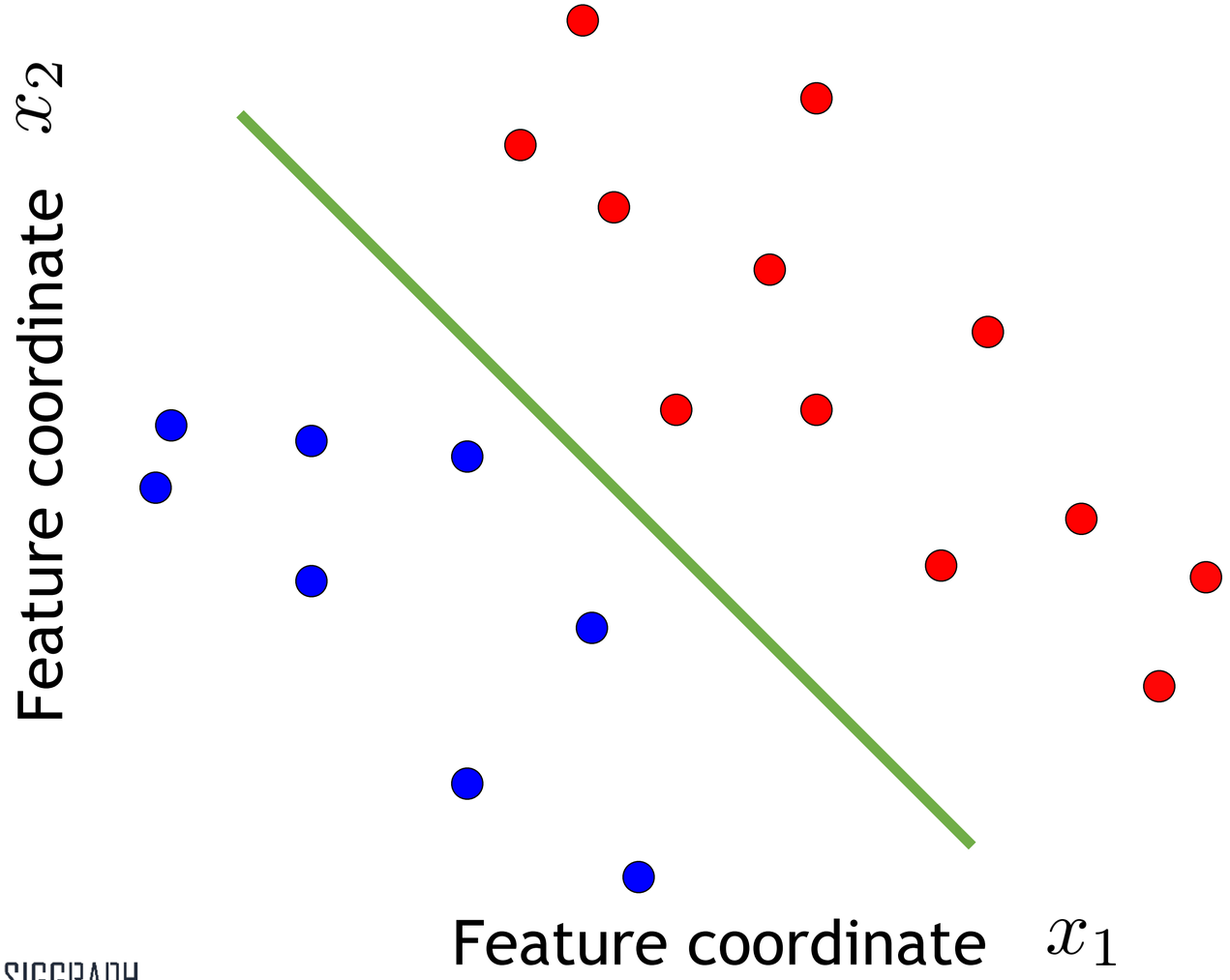
$$f_\theta : \mathbb{R}^n \longrightarrow \{0, 1\}$$

$$f_\theta(x) = \begin{cases} 1 & \text{if } wx + b \geq 0 \\ 0 & \text{if } wx + b < 0 \end{cases}$$

Each data point has a class label:

$$y^i = \begin{cases} 1 & (\bullet) \\ 0 & (\bullet) \end{cases}$$

Machine Learning 101: Linear Classifier



$$f_\theta : \mathbb{R}^n \longrightarrow \{0, 1\}$$

$$f_\theta(x) = \begin{cases} 1 & \text{if } wx + b \geq 0 \\ 0 & \text{if } wx + b < 0 \end{cases}$$

$$\theta = \{w, b\}$$

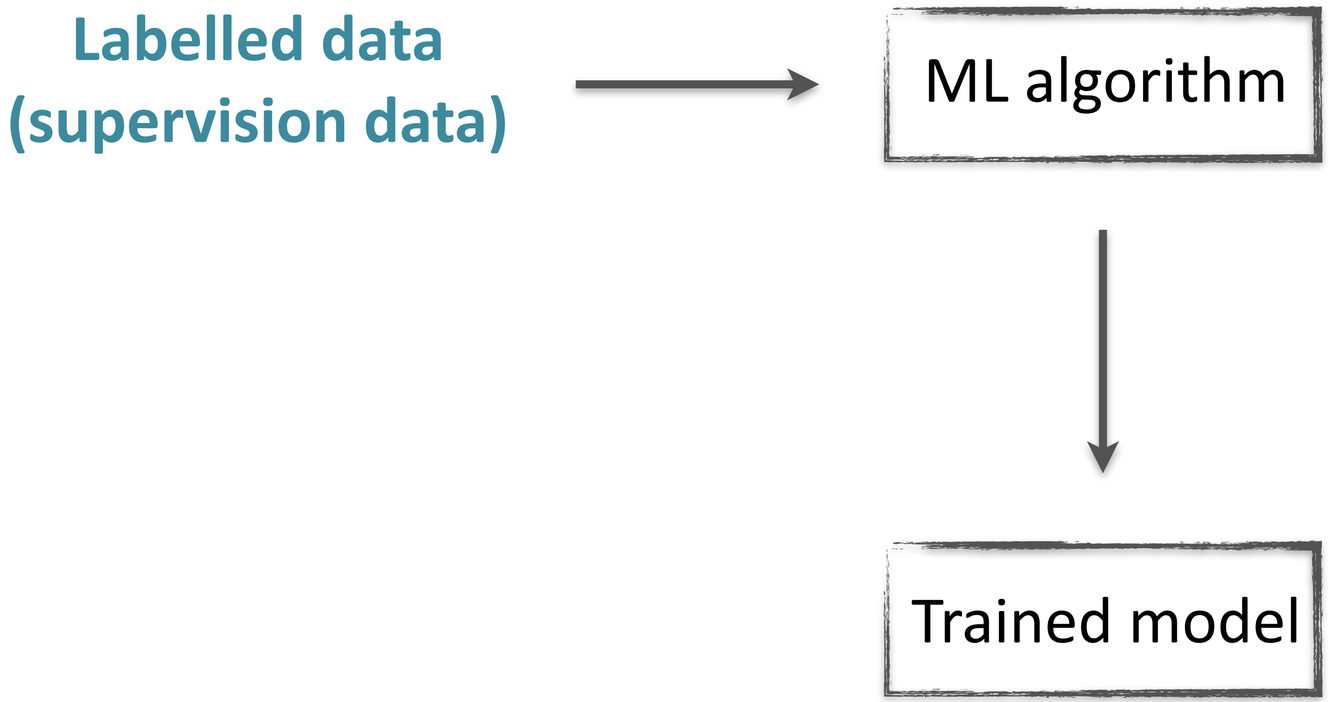
Each data point has a class label:

$$y^i = \begin{cases} 1 & (\bullet) \\ 0 & (\bullet) \end{cases}$$

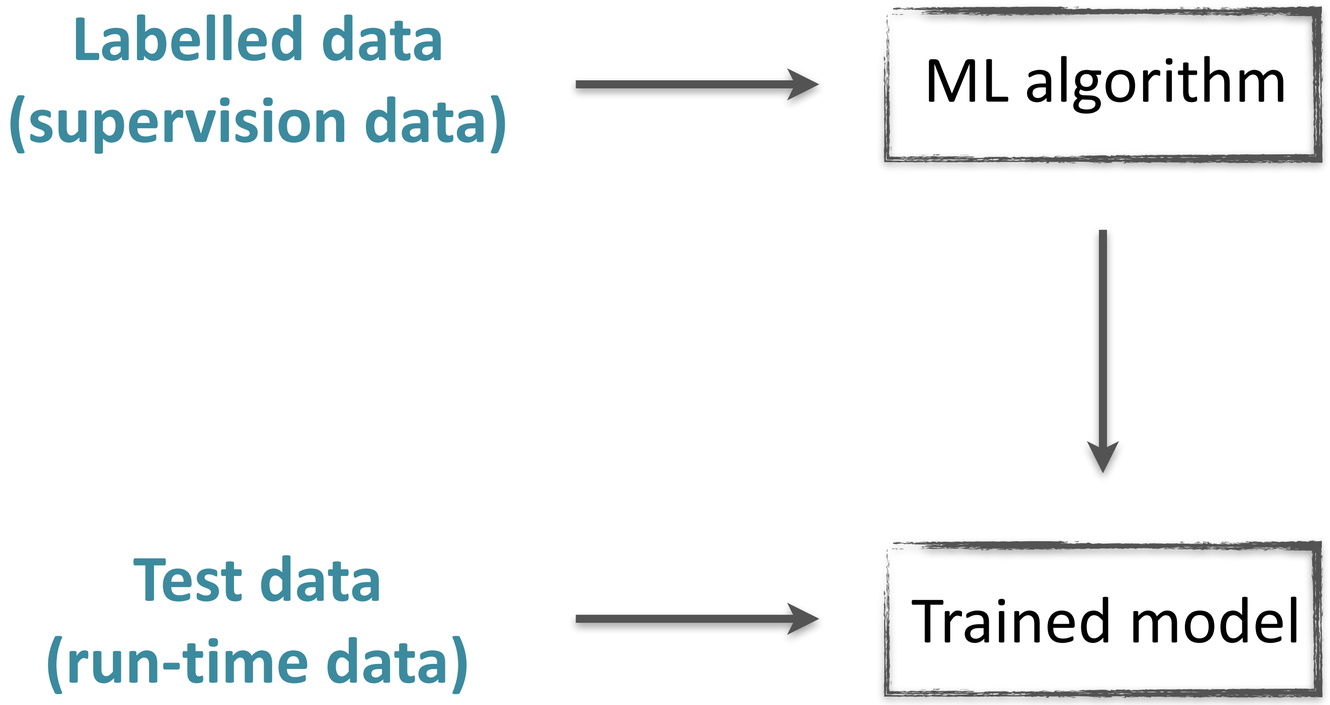
Data-driven Algorithms (Supervised)

Labelled data
(supervision data)

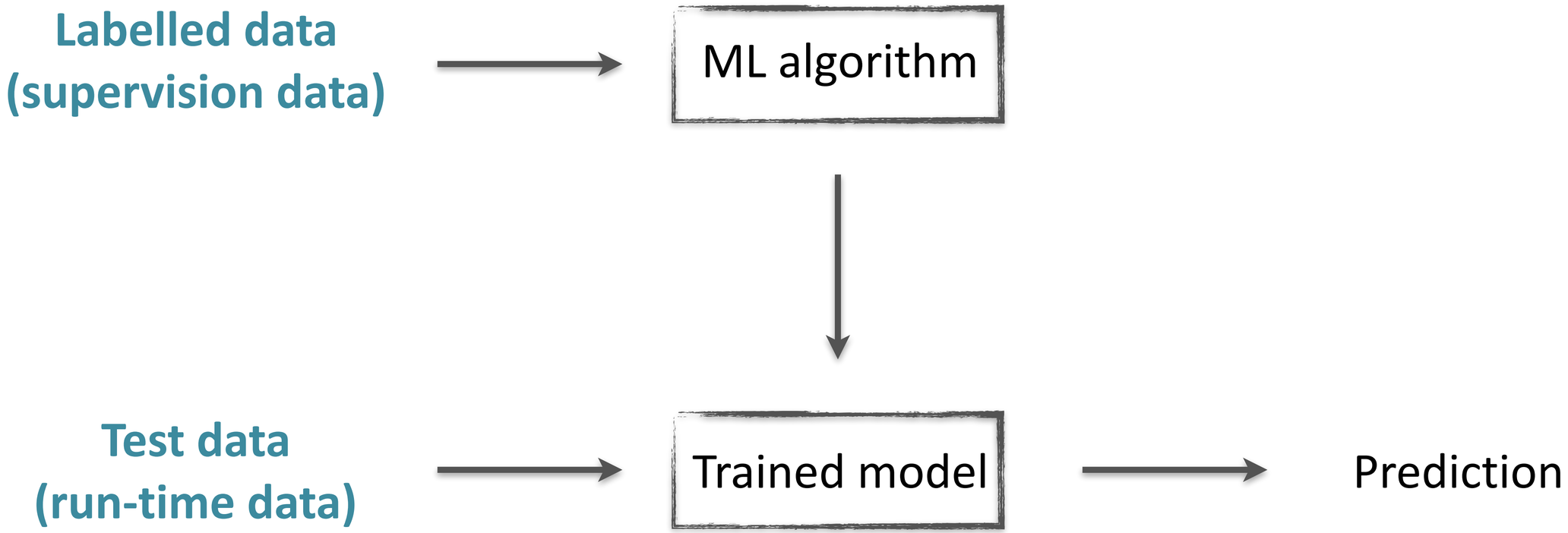
Data-driven Algorithms (Supervised)



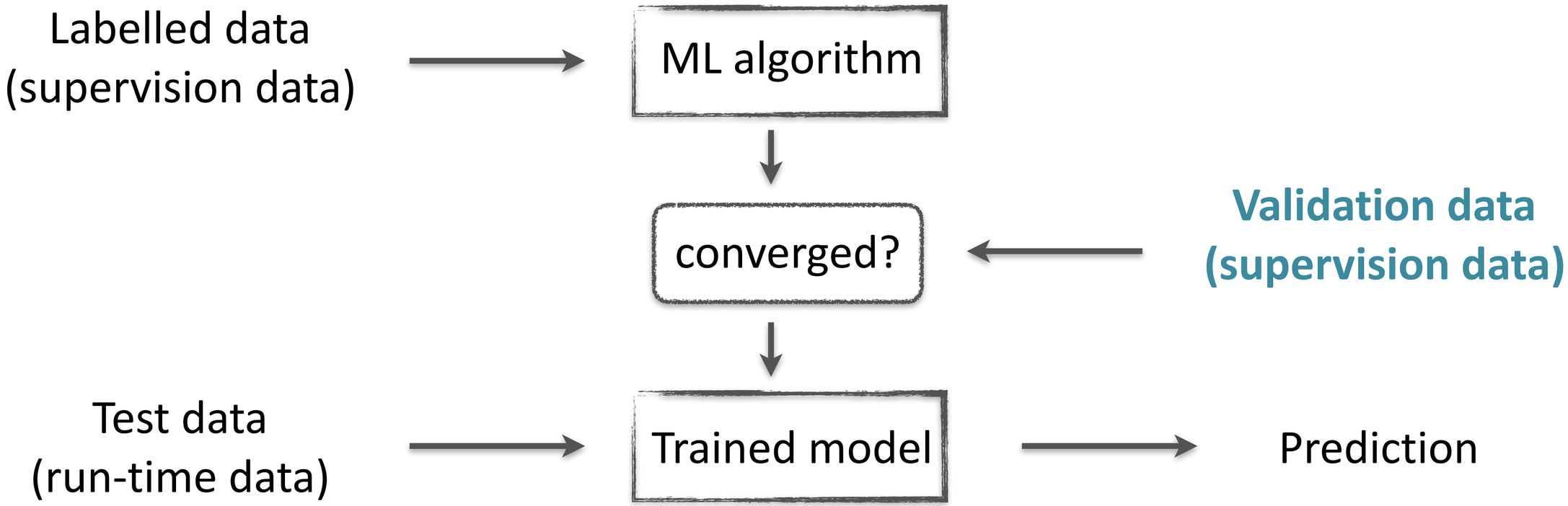
Data-driven Algorithms (Supervised)



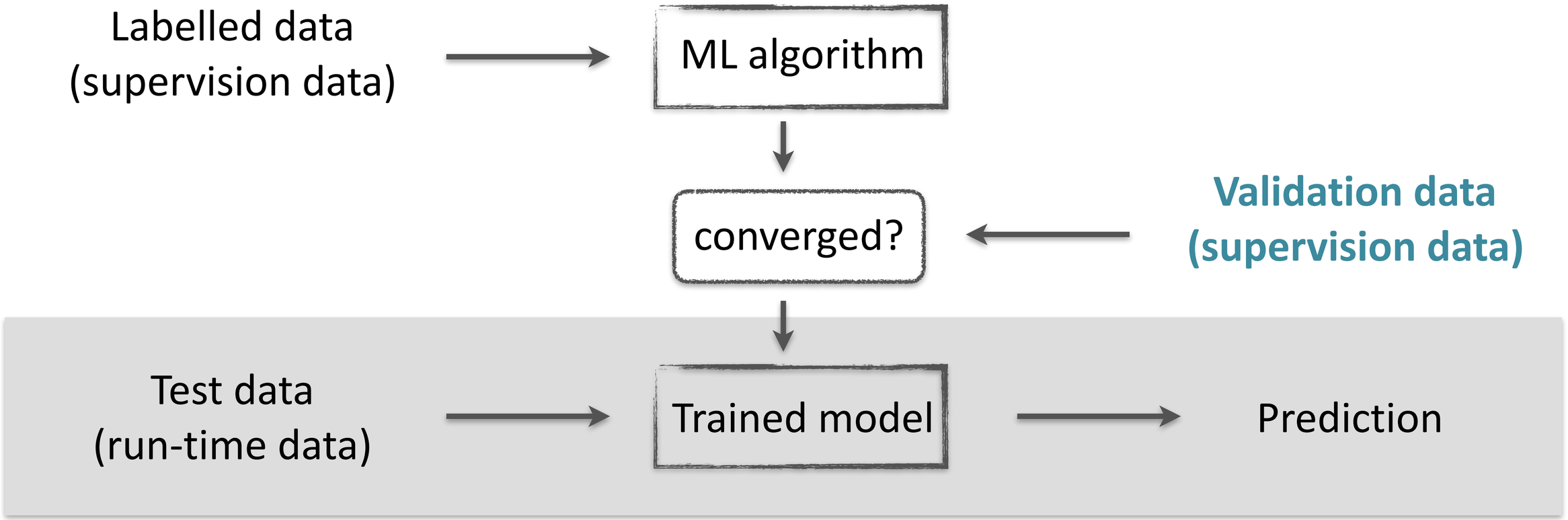
Data-driven Algorithms (Supervised)



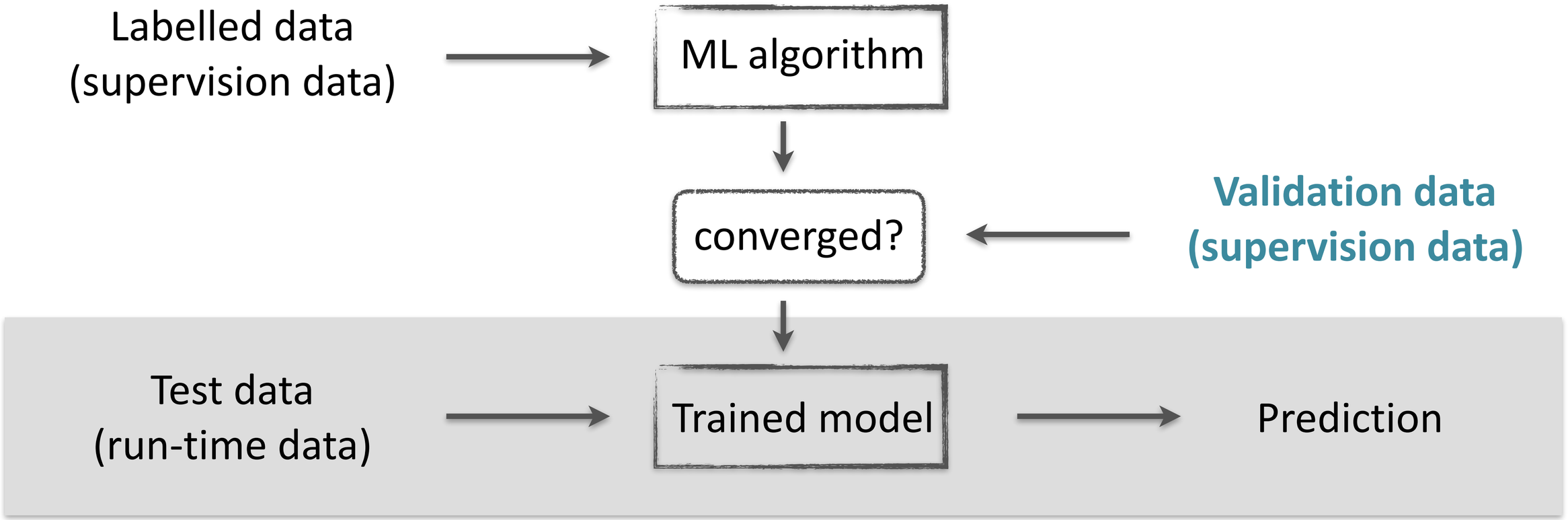
Data-driven Algorithms (Supervised)



Data-driven Algorithms (Supervised)

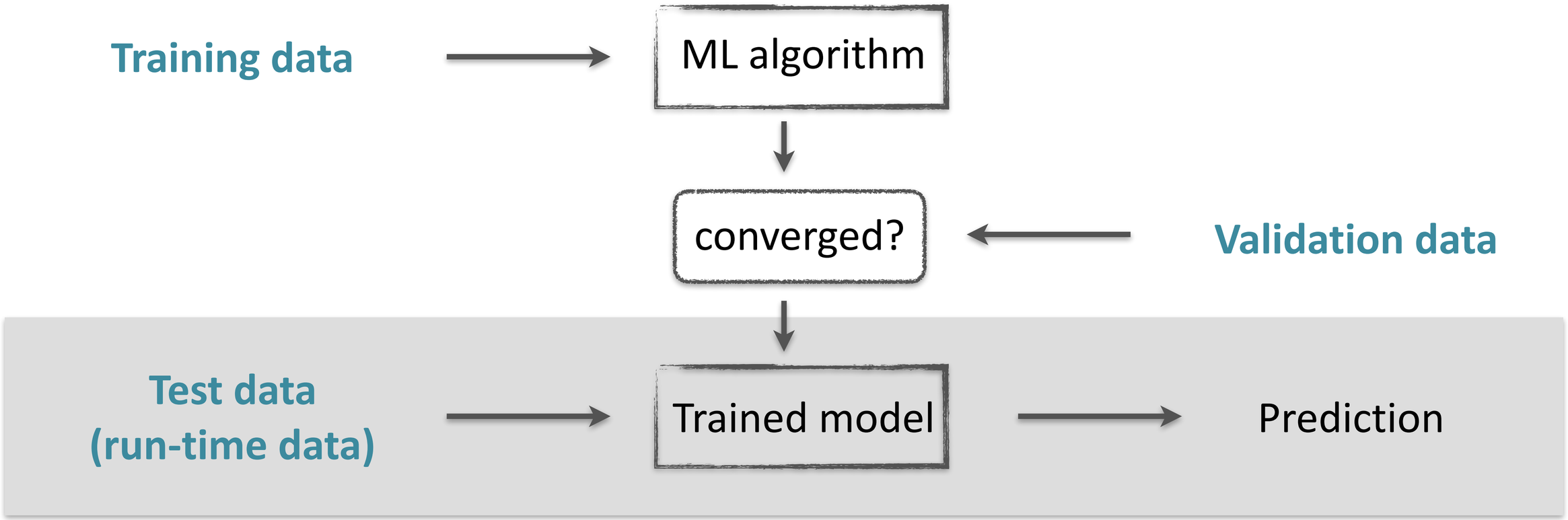


Data-driven Algorithms (Supervised)



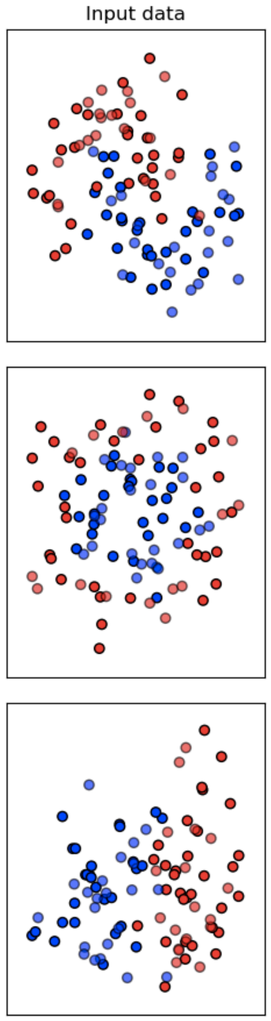
Implementation Practice: Training: 70%; Validation: 15%; Test 15%

Data-driven Algorithms (Unsupervised)



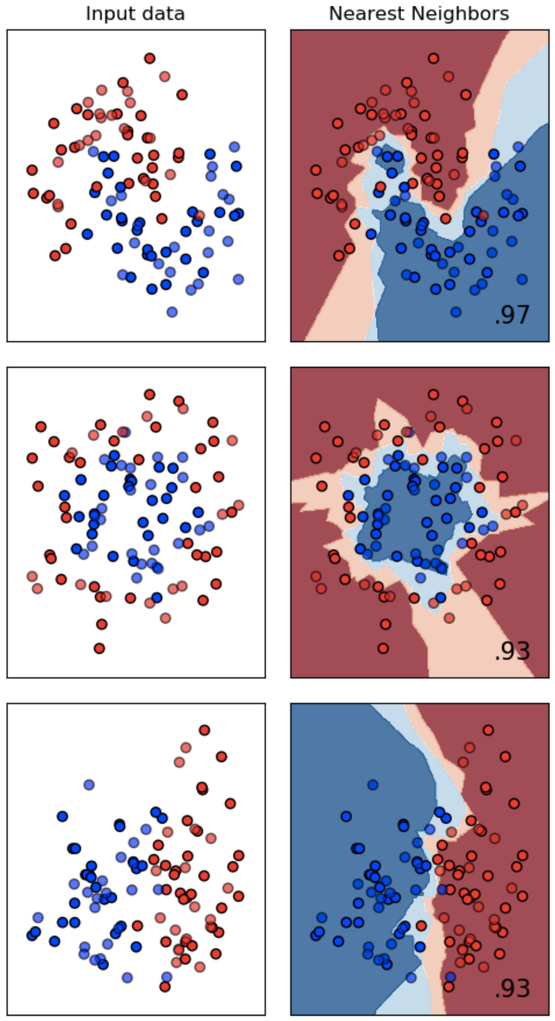
Implementation Practice: Training: 70%; Validation: 15%; Test 15%

Various ML Approaches (Supervised approaches)



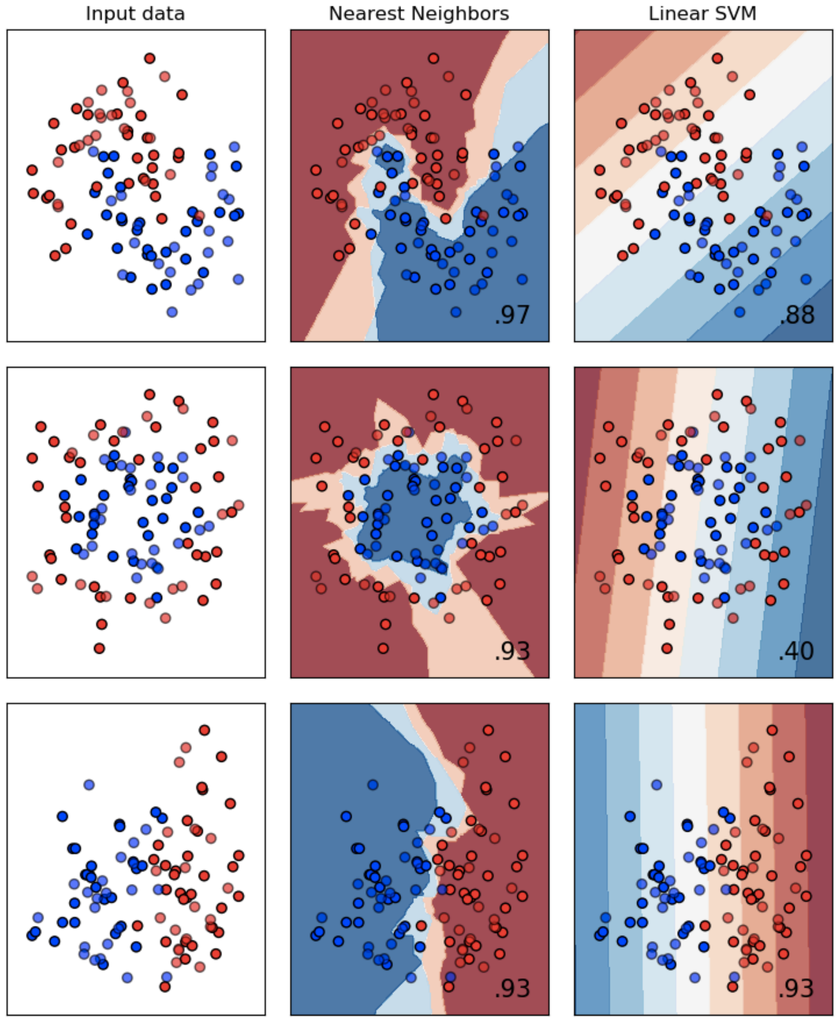
http://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html

Various ML Approaches (Supervised approaches)



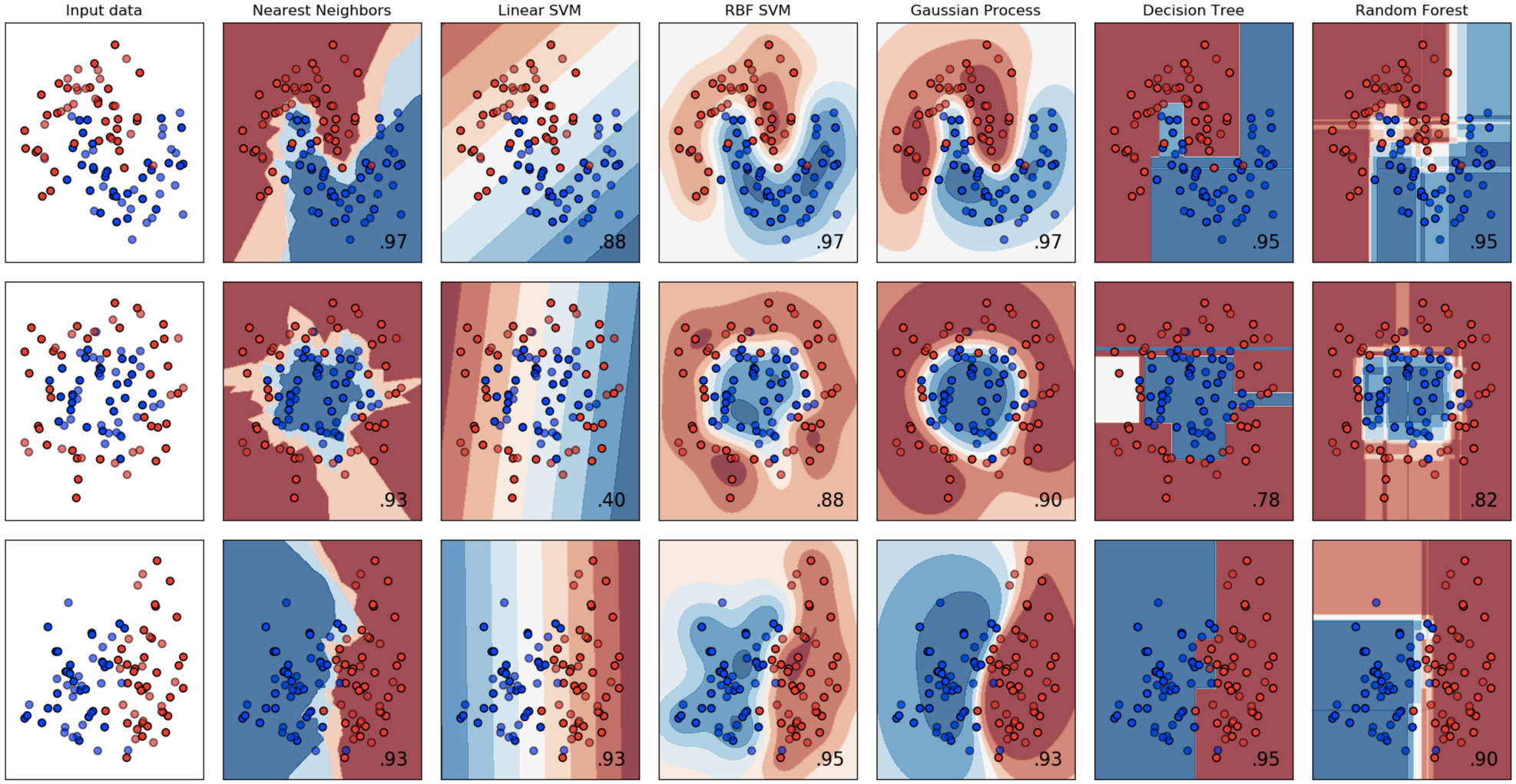
http://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html

Various ML Approaches (Supervised approaches)



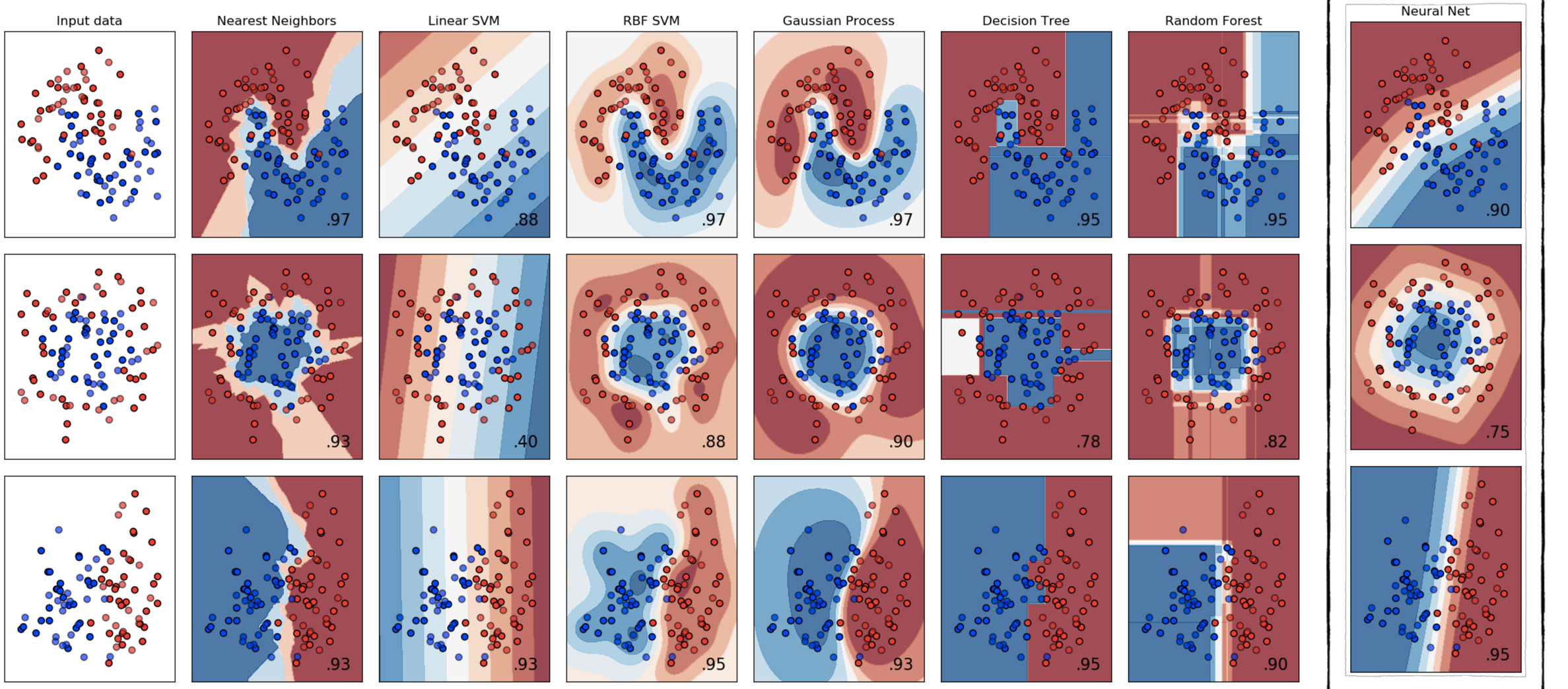
http://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html

Various ML Approaches (Supervised approaches)



http://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html

Various ML Approaches (Supervised approaches)



http://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html

Rise of Learning

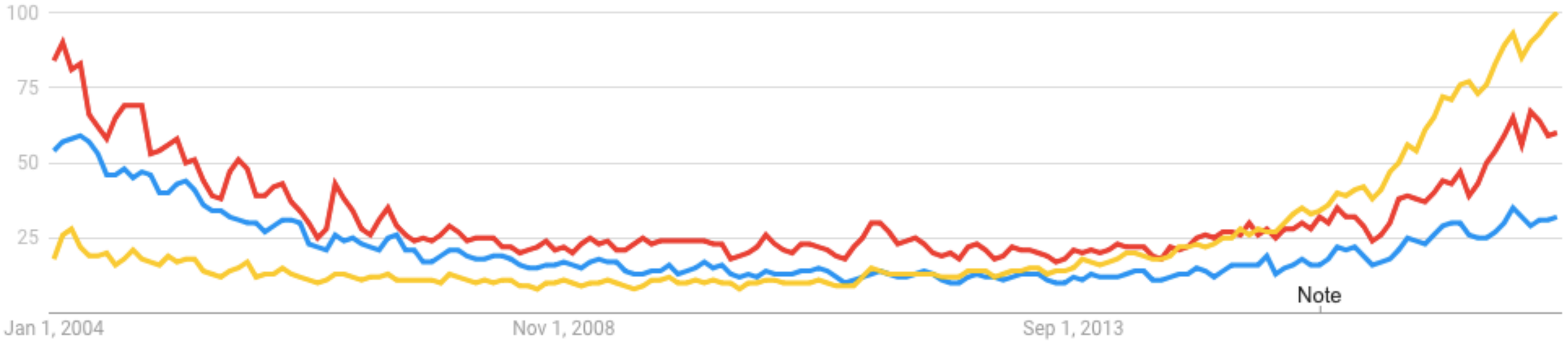
- 1958: Perceptron
- 1974: Backpropagation
- 1981: Hubel & Wiesel wins Nobel prize for ‘visual system’
- 1990s: SVM era
- 1998: CNN used for handwriting analysis
- **2012: AlexNet wins ImageNet**

Rise of Machine Learning

neural network

artificial intelligence

machine learning



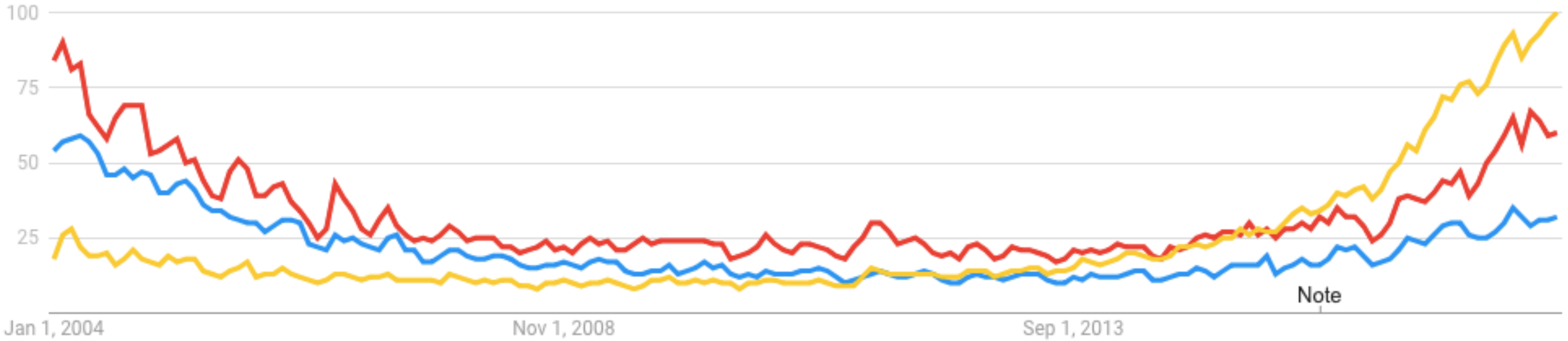
Rise of Machine Learning

neural network

artificial intelligence

machine learning

NN

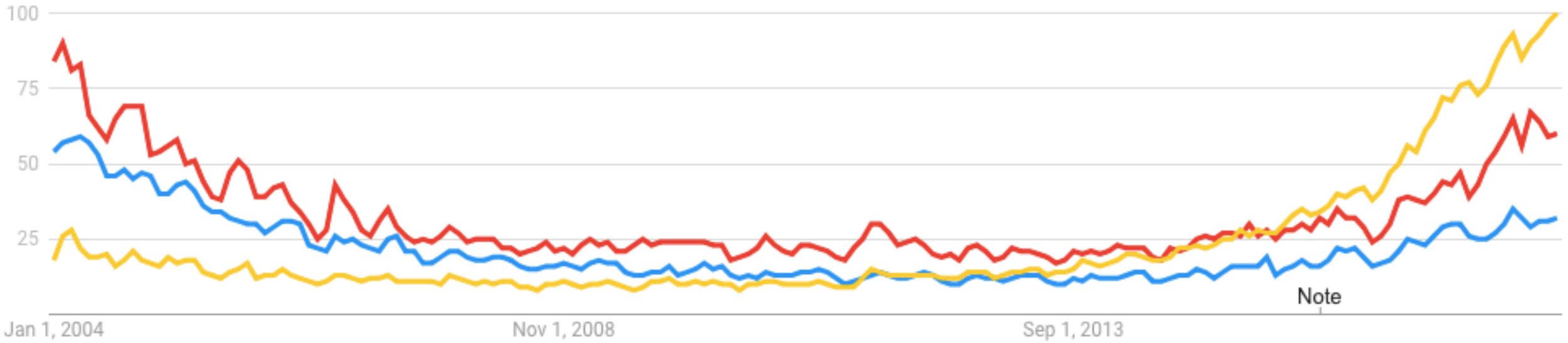


Rise of Machine Learning

neural network

artificial intelligence

machine learning

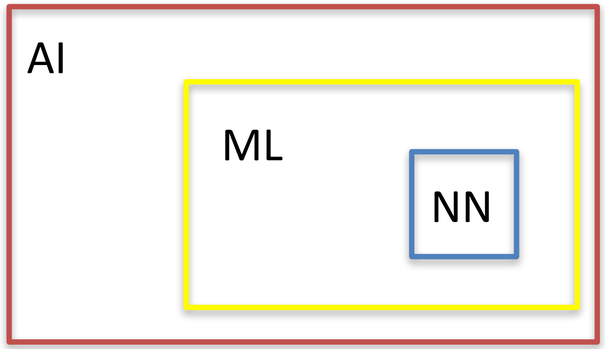


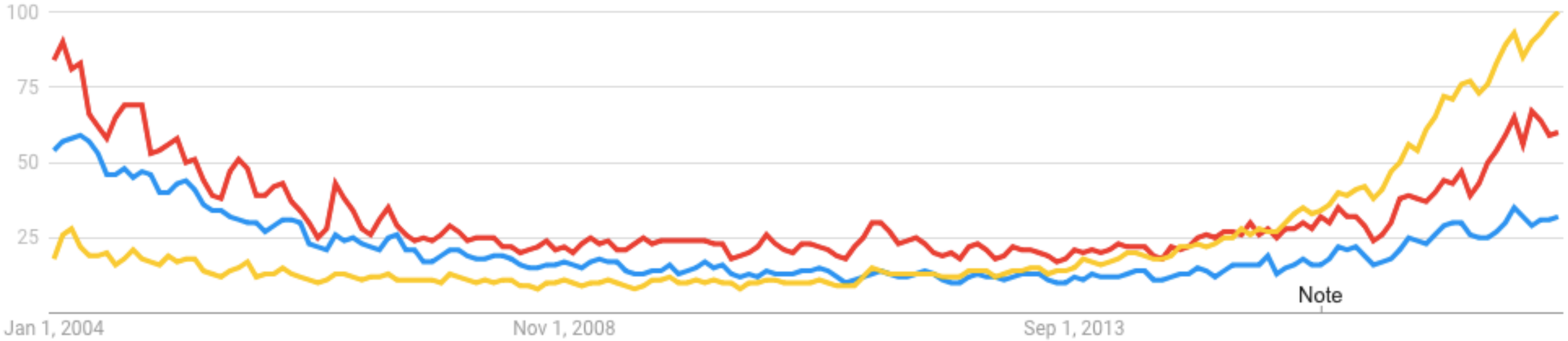
Rise of Machine Learning

neural network

artificial intelligence

machine learning

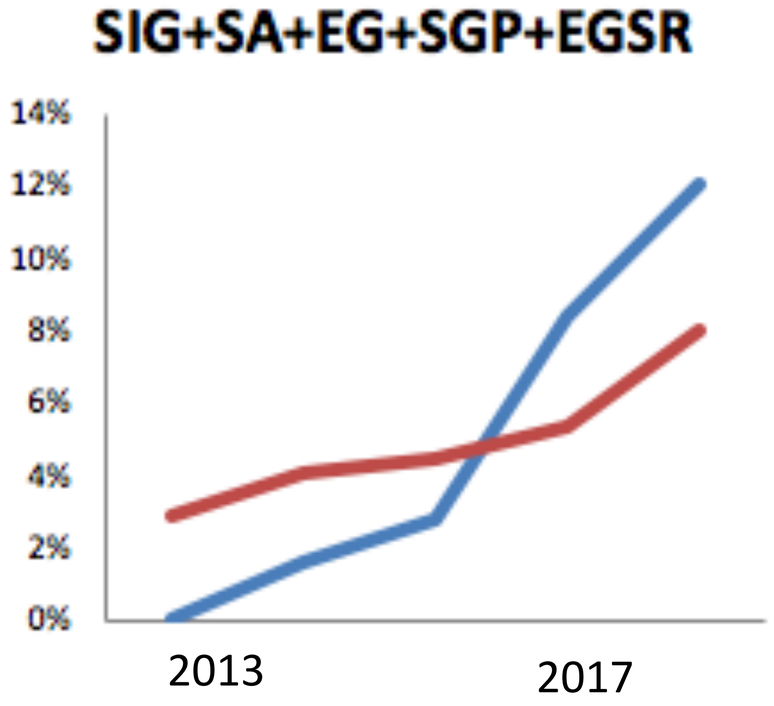


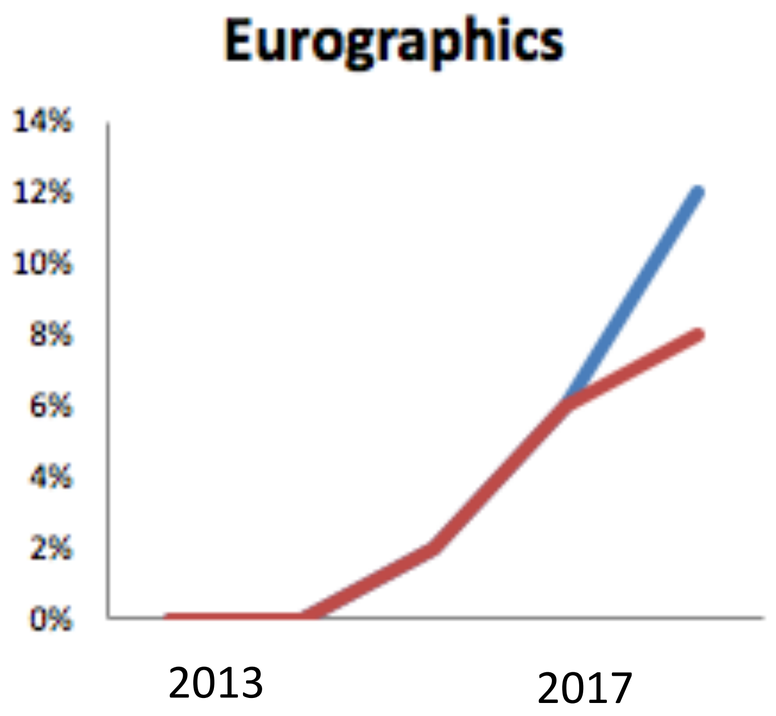


Rise of Machine Learning (in Graphics)

machine learning

neural network





What is Special about CG?

What is Special about CG?

1. **Image Processing** (image translation tasks)

What is Special about CG?

1. **Image Processing** (image translation tasks)

2. Many sources of input data — **model building**
   (e.g., images, scanners, motion capture)

What is Special about CG?

1. **Image Processing** (image translation tasks)

2. Many sources of input data — **model building**
   (e.g., images, scanners, motion capture)

3. Many sources of **synthetic data** — can serve as supervision data
   (e.g., rendering, animation)

What is Special about CG?

1. **Image Processing** (image translation tasks)

2. Many sources of input data — **model building**
   (e.g., images, scanners, motion capture)

3. Many sources of **synthetic data** — can serve as supervision data
   (e.g., rendering, animation)

4. Many problems in **generative models**

Main Challenges and Scope for Innovation

Main Challenges and Scope for Innovation

1. **Representation:** How is the data organised and structured?

Main Challenges and Scope for Innovation

1. **Representation:** How is the data organised and structured?

2. **Training data:** Is it synthetic or real, or mixed?

Main Challenges and Scope for Innovation

1. **Representation:** How is the data organised and structured?

2. **Training data:** Is it synthetic or real, or mixed?

3. **User control:** End-to-end or in small steps?

Main Challenges and Scope for Innovation

1. **Representation:** How is the data organised and structured?

2. **Training data:** Is it synthetic or real, or mixed?

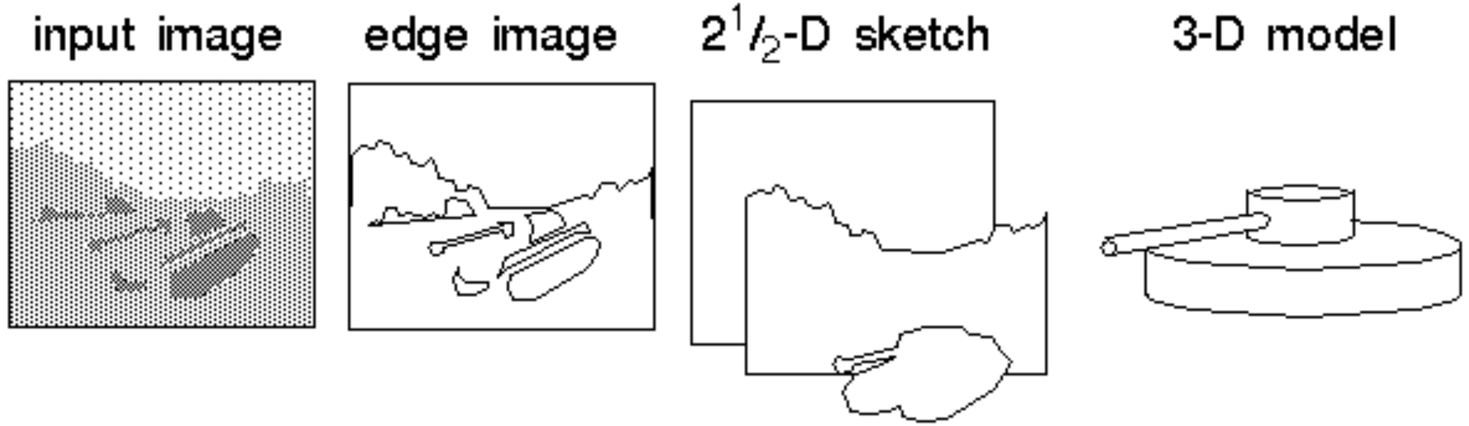
3. **User control:** End-to-end or in small steps?

4. **Loss functions:** Hand-crafted or learned from data?

End-to-end: Learned **Features**

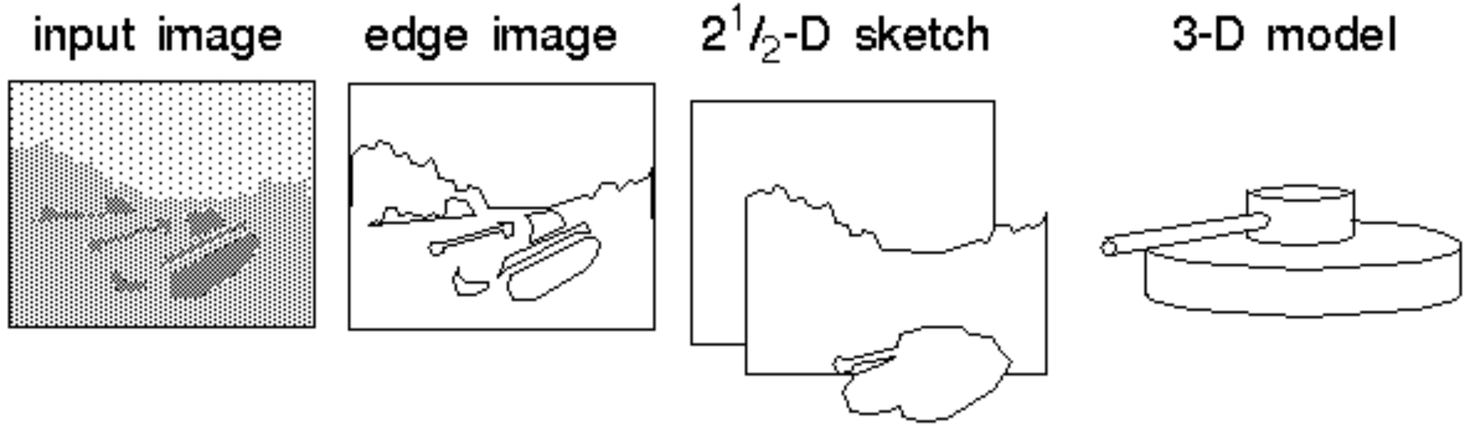
End-to-end: Learned **Features**

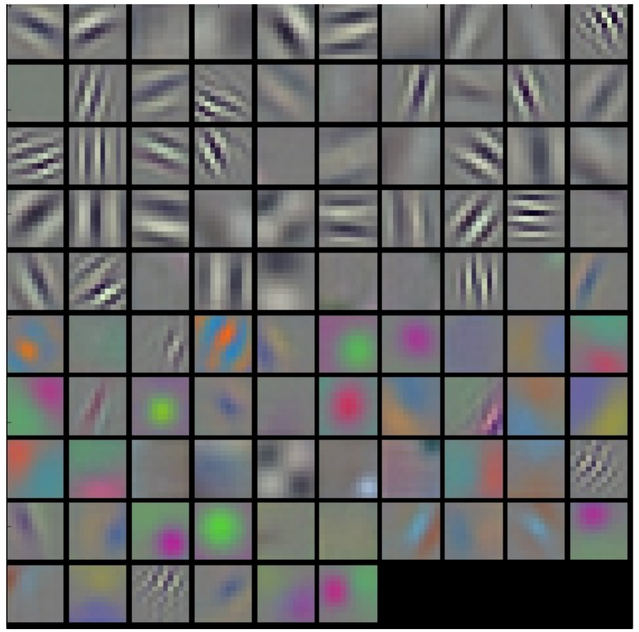
- *Old days*
  - Handcrafted feature extraction, e.g., edges or corners (hand-crafted)
  - Mostly with linear models (PCA, etc.)



End-to-end: Learned **Features**

- *Old days*
  - Handcrafted feature extraction, e.g., edges or corners (hand-crafted)
  - Mostly with linear models (PCA, etc.)
- *Now*
  - End-to-end
  - Move away from hand-crafted representations





End-to-end: Learned **Loss**

End-to-end: Learned Loss

- *Old days*
  - Evaluation came after
  - It was a bit optional
    - You might still have a good algorithm without a good way of quantifying it
    - Evaluation helped publishing

End-to-end: Learned Loss

- *Old days*
  - Evaluation came after
  - It was a bit optional
    - You might still have a good algorithm without a good way of quantifying it
    - Evaluation helped publishing
- *Now*
  - It is essential and build-in
  - If the loss is not good, the result is not good
  - (Extensive) Evaluation happens automatically

End-to-end: Learned Loss

- *Old days*
  - Evaluation came after
  - It was a bit optional
    - You might still have a good algorithm without a good way of quantifying it
    - Evaluation helped publishing
- *Now*
  - It is essential and build-in
  - If the loss is not good, the result is not good
  - (Extensive) Evaluation happens automatically

- While still much is left to do, this makes graphics much more reproducible

End-to-end: Real/Generated **Data**

End-to-end: Real/Generated Data

- *Old days*
  - Test with some toy examples
  - Deploy on real stuff
  - Maybe collect some performance data later

End-to-end: Real/Generated Data

- *Old days*
  - Test with some toy examples
  - Deploy on real stuff
  - Maybe collect some performance data later

- *Now*
  - Test and deploy need to be as identical **(in distribution)**
  - Need to collect data first
  - No two steps

Examples in Graphics

Geometry

Image
manipulation

Animation

Rendering

Examples in Graphics

Geometry

Procedural modelling

Mesh segmentation

Learning deformations

Colorization

Sketch simplification

Image manipulation

Animation

BRDF estimation

Boxification

Fluid

Real-time rendering

Animation

Rendering

Denoising

Facial animation

PCD processing

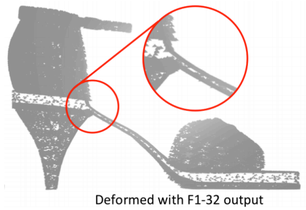
Examples in Graphics

Sketch simplification

Colorization

Procedural modelling

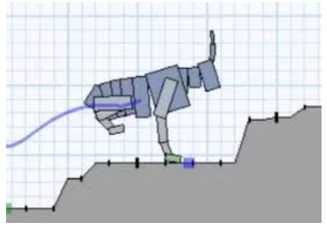
Mesh segmentation



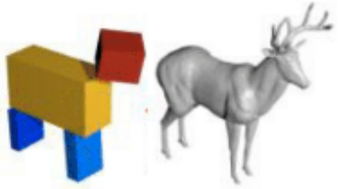
Learning deformations

BRDF estimation

Fluid



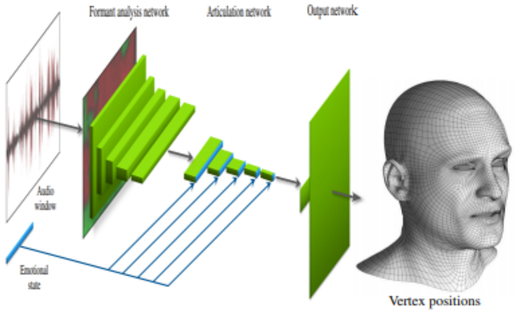
Animation



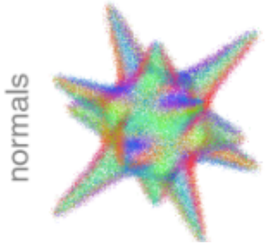
Boxification

Real-time rendering

Denoising

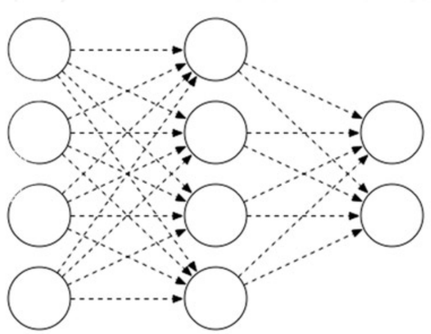


Facial animation



PCD processing

Course Information (slides/code/comments)



http://geometry.cs.ucl.ac.uk/creativeai/

CreativeAI

Machine Learning Basics

Niloy Mitra
UCL

Iasonas Kokkinos
UCL

Paul Guerrero
UCL

Nils Thuerey
TUM

Tobias Ritschel
UCL

Timetable

			Niloy	Paul	Nils
Theory + Basics	Introduction	2:15 pm	X	X	X
	Machine Learning Basics	~ 2:25 pm	X		
	Neural Network Basics	~ 2:55 pm			X
	Feature Visualization	~ 3:25 pm		X	
	Alternatives to Direct Supervision	~ 3:35 pm		X	
	15 min. break				
State of the Art	Image Domains	4:15 pm		X	
	3D Domains	~ 4:45 pm	X		
	Motion and Physics	~ 5:15 pm			X
	Discussion	~ 5:45 pm	X	X	X

Machine Learning

Machine Learning

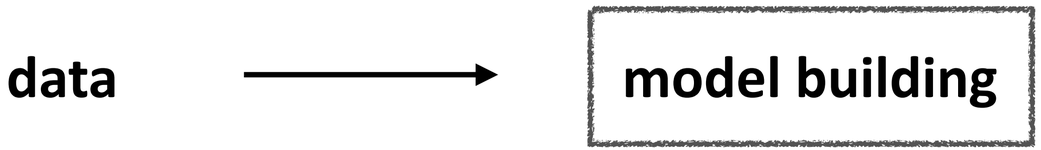
Machine learning is a field of **computer science** that uses statistical techniques to give computer systems the ability to ***learn*** (i.e., progressively improve performance on a specific task) with **data**, without being explicitly programmed.

'ML' coined by Arthur Samuel, 1959.

Machine Learning

Machine learning is a field of **computer science** that uses statistical techniques to give computer systems the ability to ***learn*** (i.e., progressively improve performance on a specific task) with **data**, without being explicitly programmed.

'ML' coined by Arthur Samuel, 1959.



Machine Learning

Machine learning is a field of **computer science** that uses statistical techniques to give computer systems the ability to ***learn*** (i.e., progressively improve performance on a specific task) with **data**, without being explicitly programmed.

'ML' coined by Arthur Samuel, 1959.

Machine Learning Variants

- **Supervised**
  - Classification
  - Regression
  - Data consolidation
- **Unsupervised**
  - Clustering
  - Dimensionality Reduction
- **Weakly supervised/semi-supervised**

  Some data supervised, some unsupervised
- **Reinforcement learning**

  Supervision: sparse reward for a sequence of decisions

Machine Learning Variants

- **Supervised**
  - **Classification**
  - Regression
  - Data consolidation
- **Unsupervised**
  - Clustering
  - Dimensionality Reduction
- **Weakly supervised/semi-supervised**

  Some data supervised, some unsupervised
- **Reinforcement learning**

  Supervision: sparse reward for a sequence of decisions

Classification Examples

- Digit Recognition

Classification Examples

- Digit Recognition

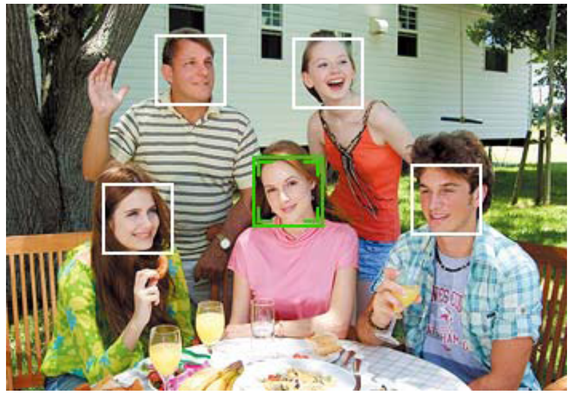
- Spam Detection

Classification Examples

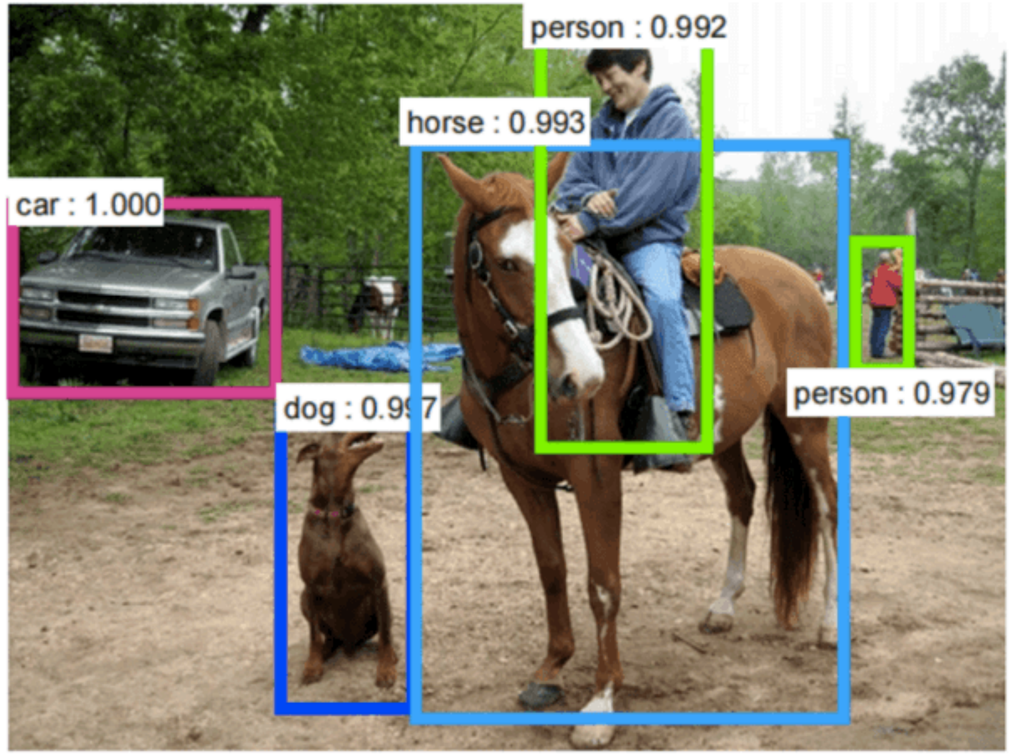
- Digit Recognition

- Spam Detection

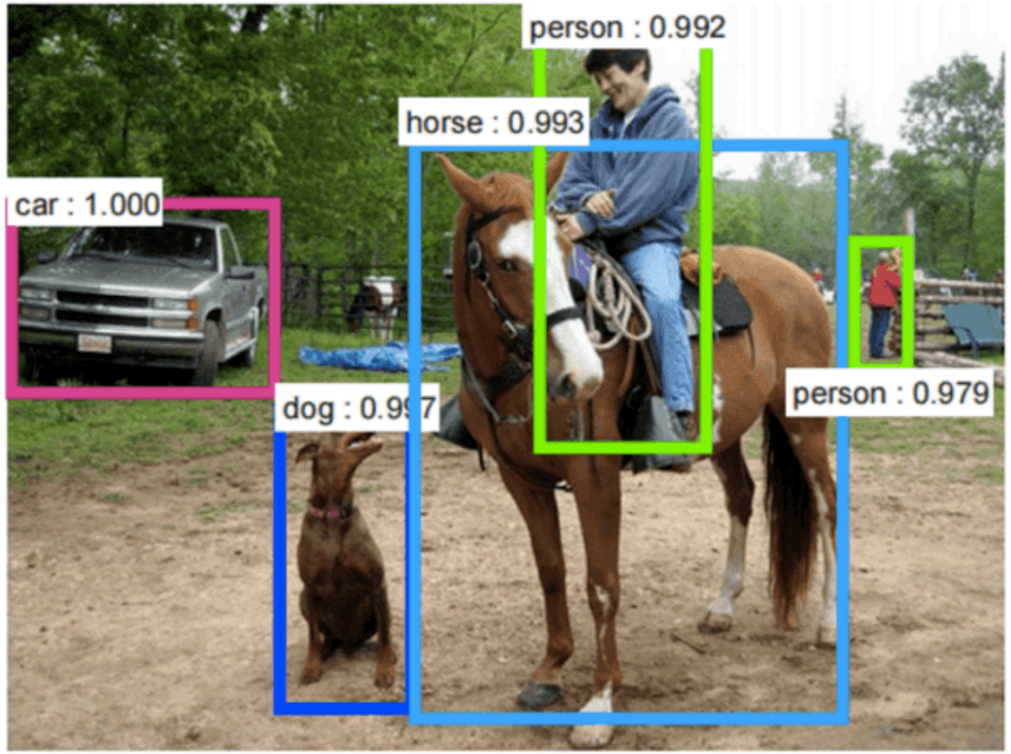
- Face detection

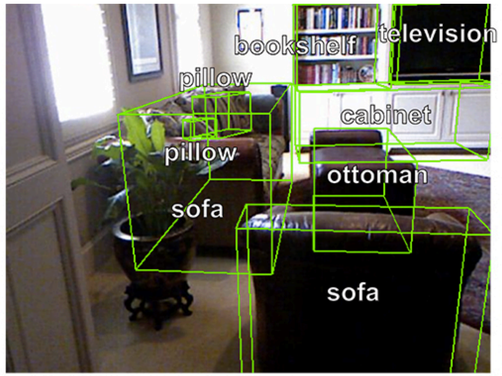


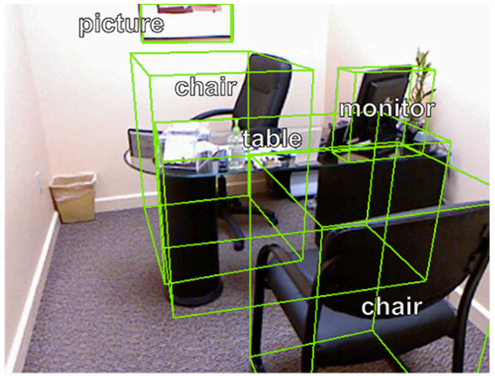
Segmentation + Classification in Real Images

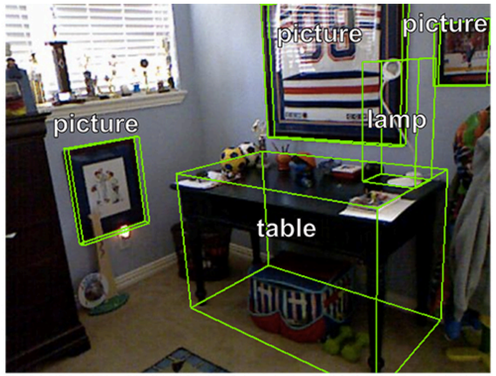


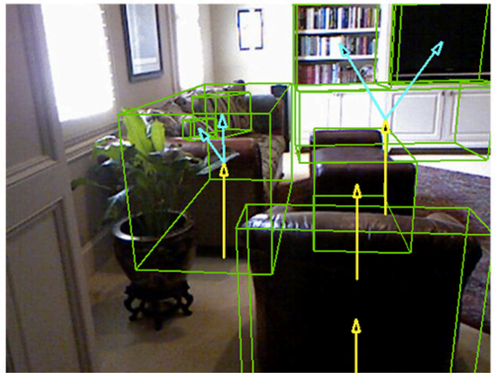
Segmentation + Classification in Real Images

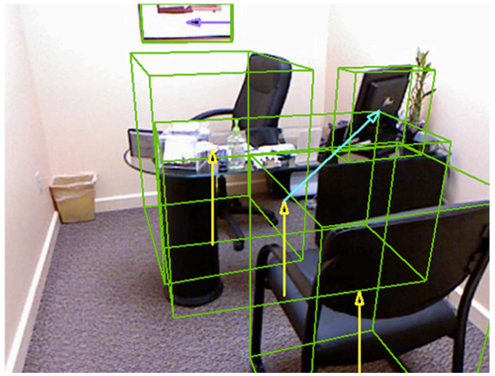


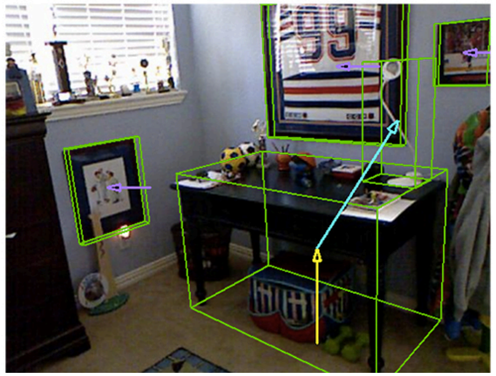




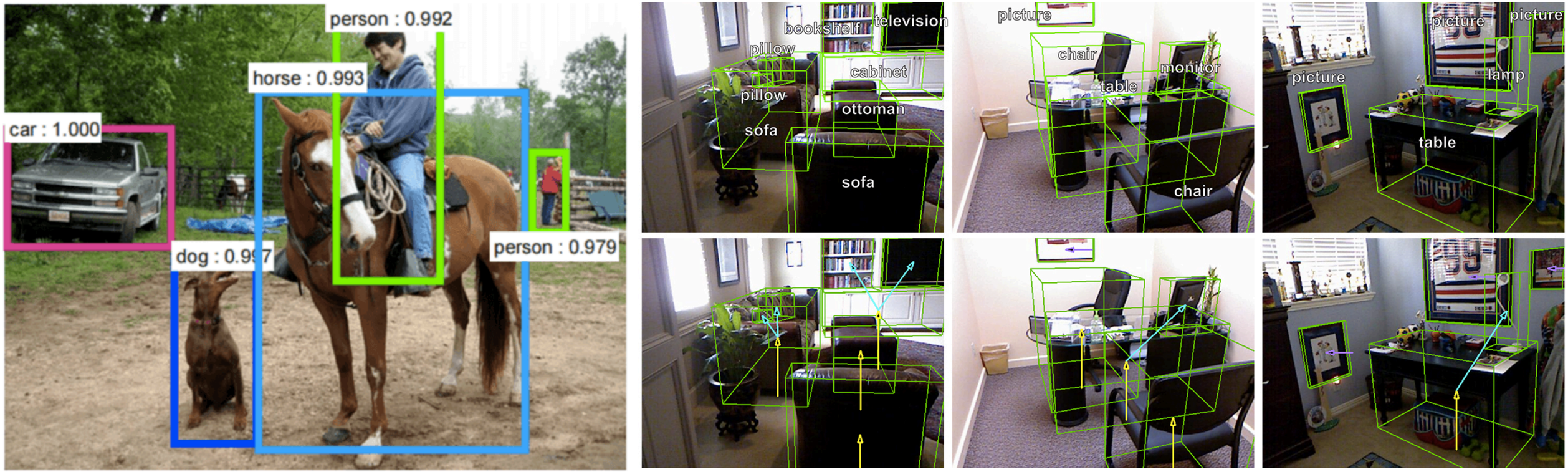








Segmentation + Classification in Real Images



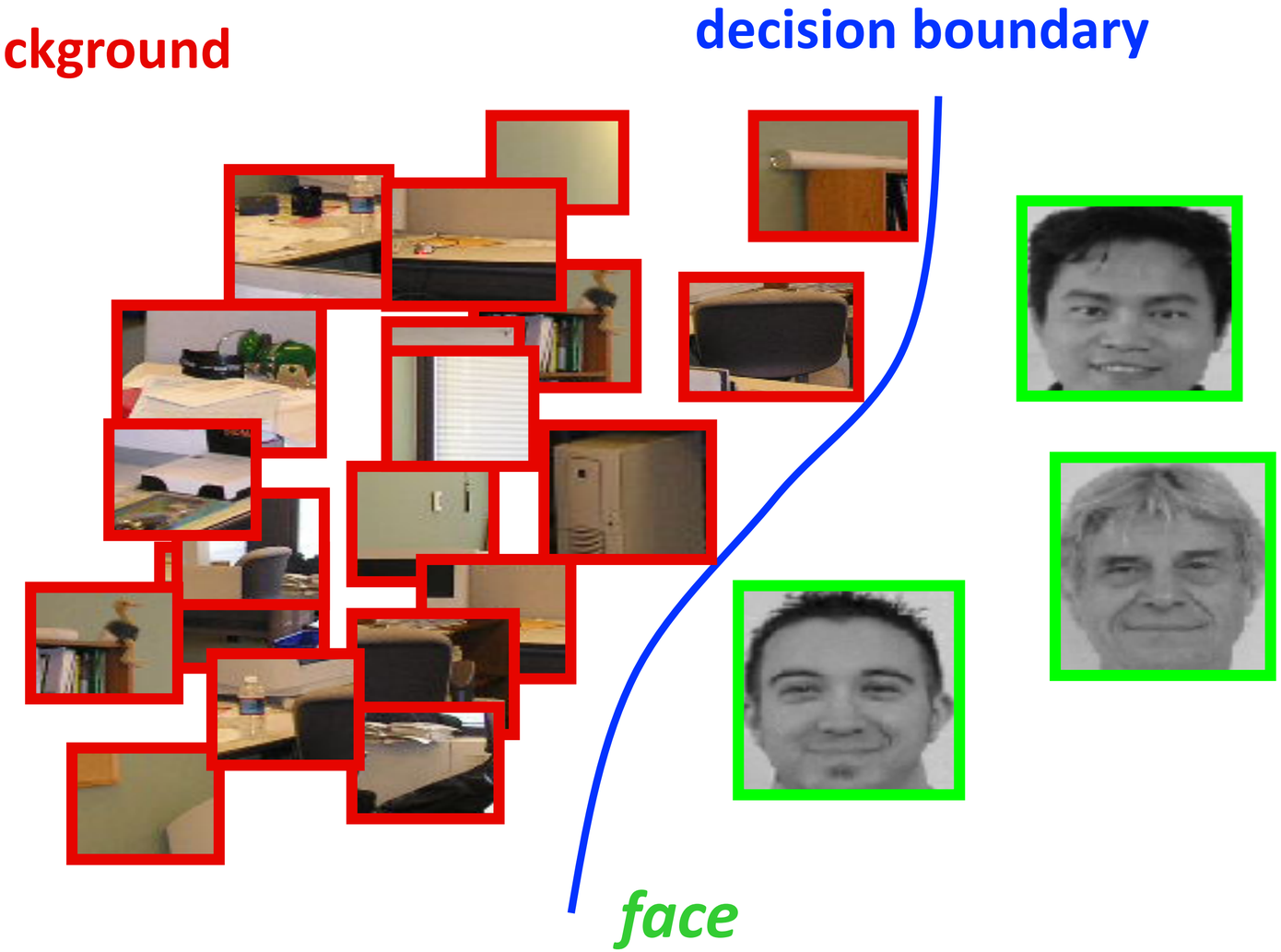
Evaluation measures: Confusion matrix, ROC curve, precision, recall, etc.

`Faceness’ Function: Classifier

background

face

`Faceness’ Function: Classifier



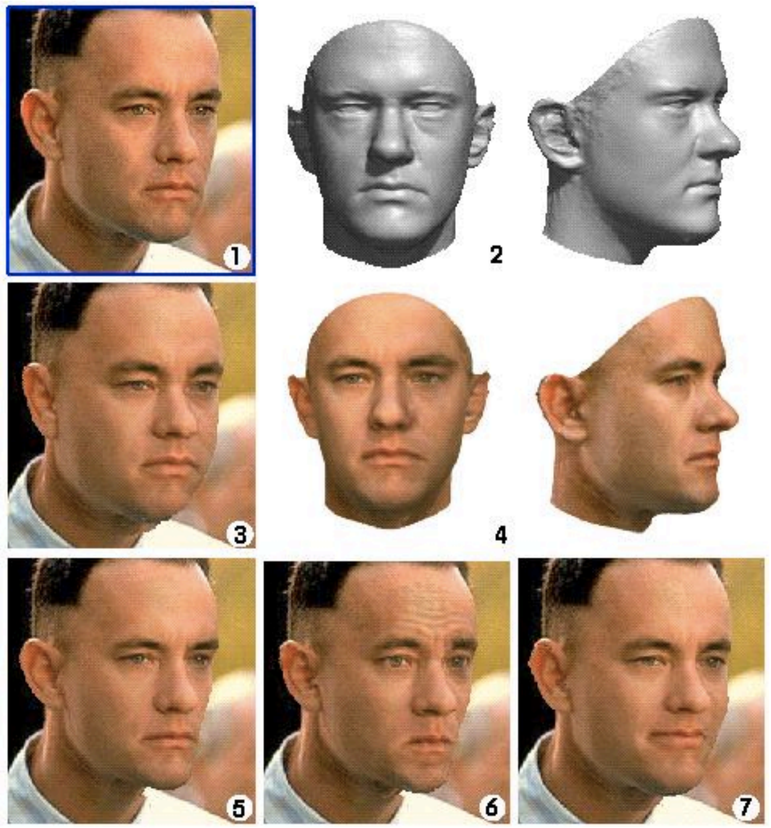
Machine Learning Variants

- **Supervised**
  - Classification
  - **Regression**
  - Data consolidation
- **Unsupervised**
  - Clustering
  - Dimensionality Reduction
- **Weakly supervised/semi-supervised**

  Some data supervised, some unsupervised
- **Reinforcement learning**

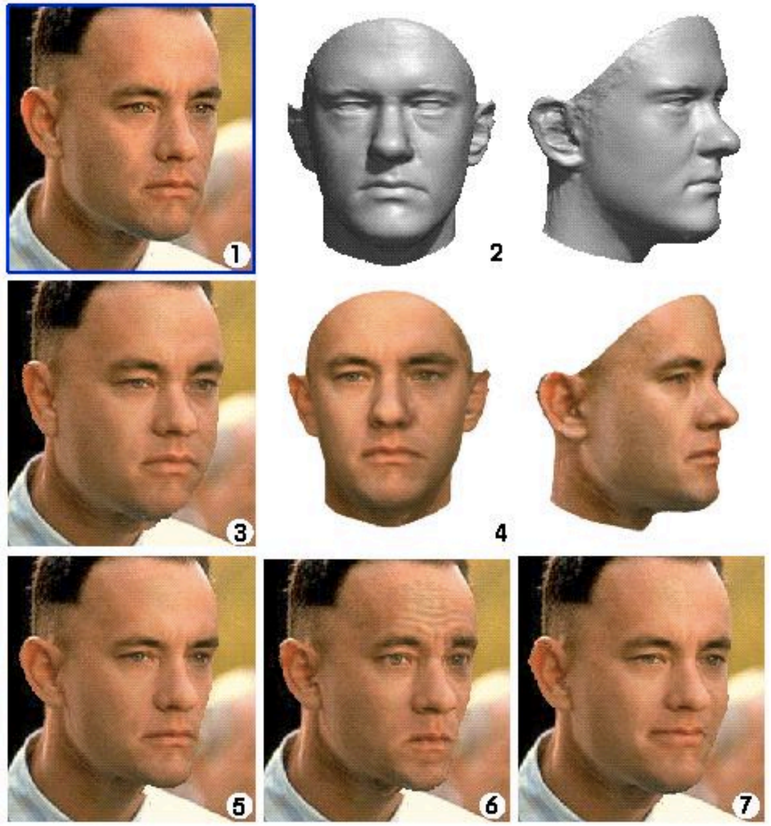
  Supervision: sparse reward for a sequence of decisions

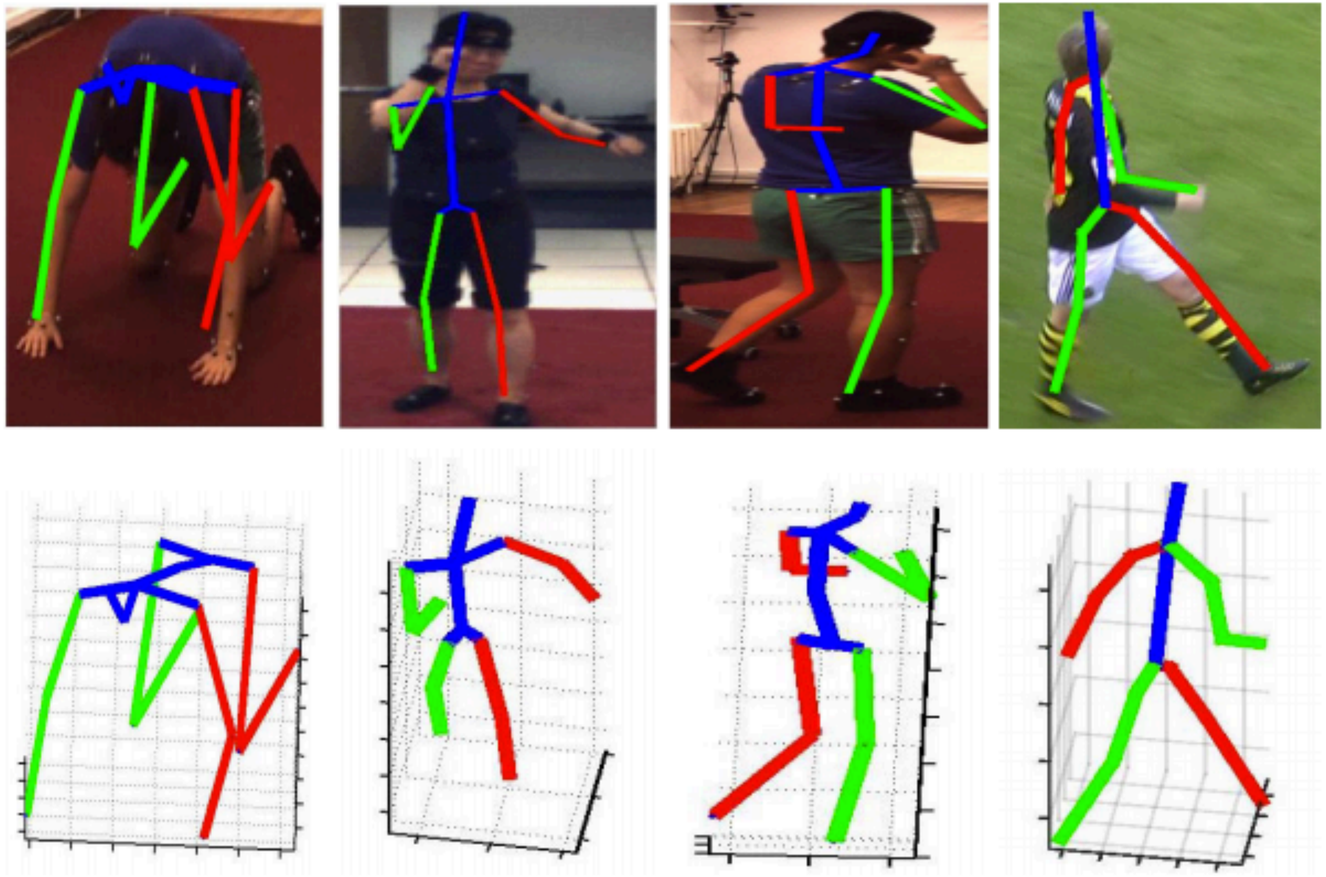
Human Face/Pose Estimation



[Blanz and Vetter, Siggraph, 1999]

Human Face/Pose Estimation

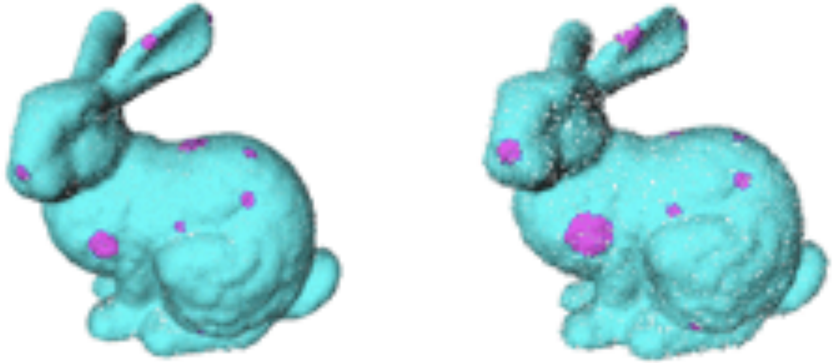


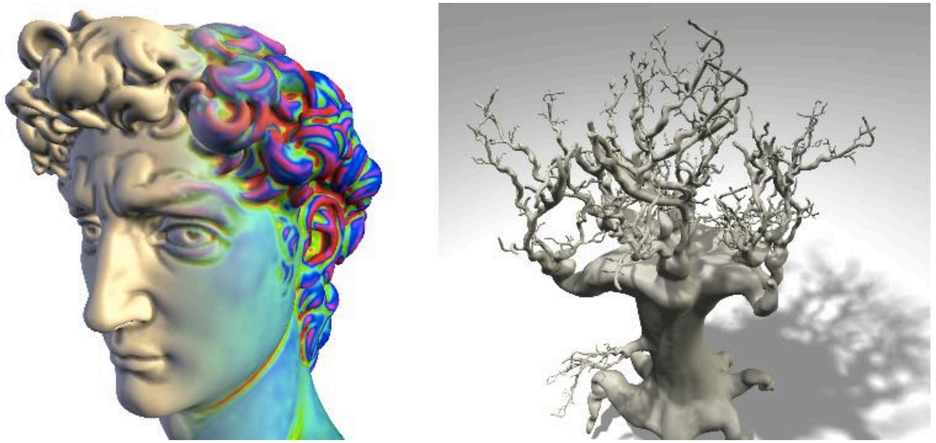


[Blanz and Vetter, Siggraph, 1999]

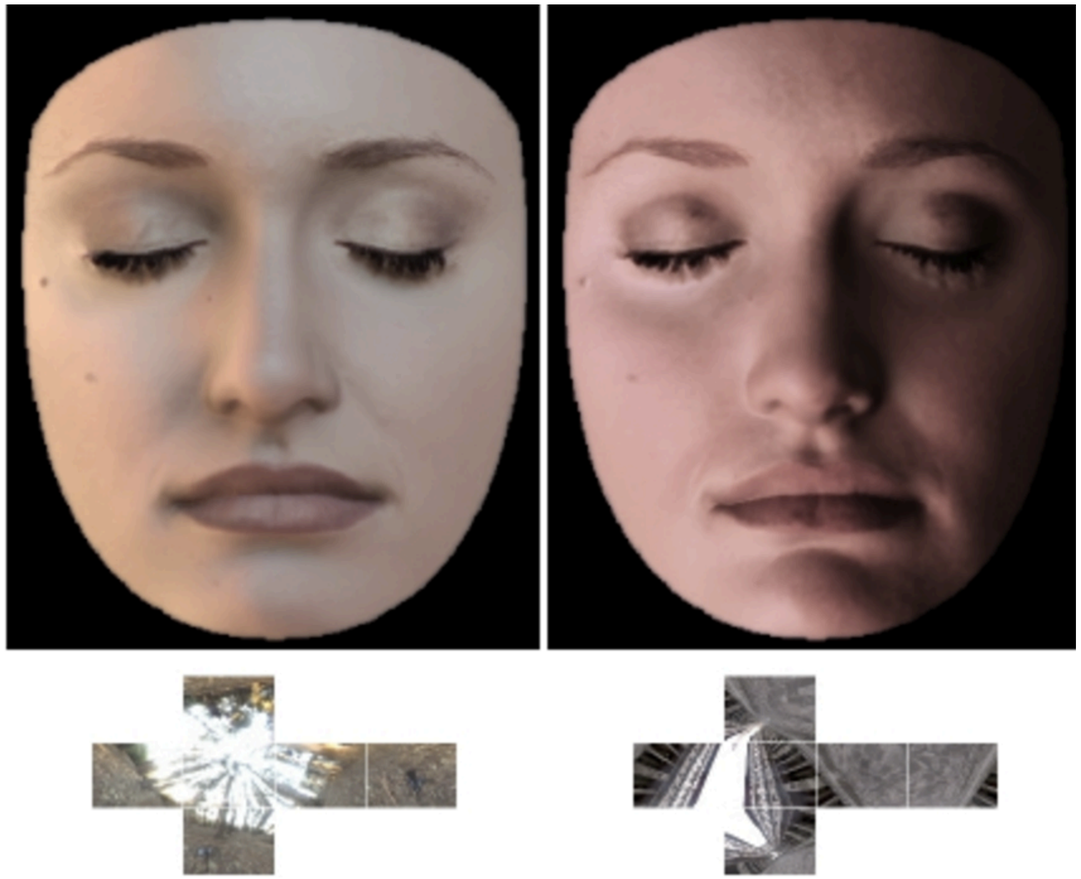
Regression: Model Estimation

[Mitra et al. SoCG, 2003]





[Guennebaud et al., Siggraph, 2007]



[Zwicker et al., EGSR, 2005]

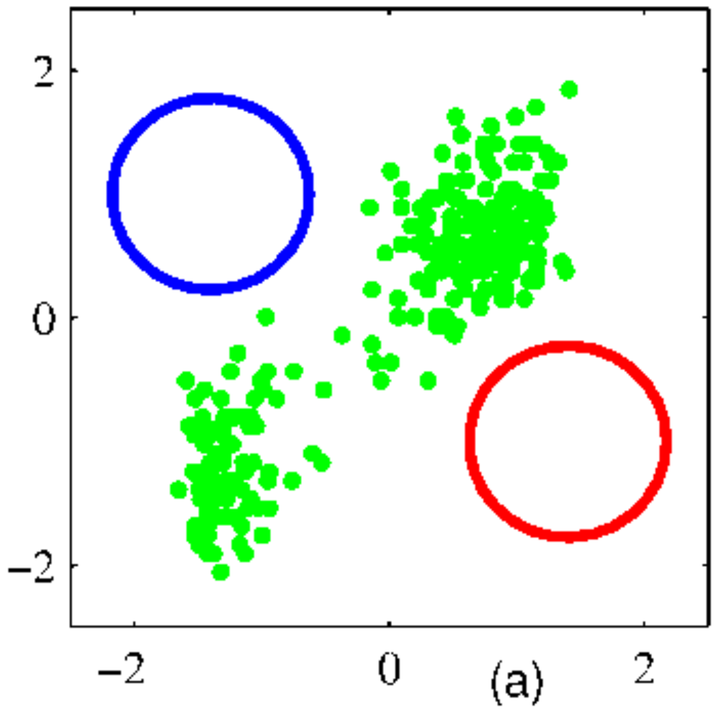
Machine Learning Variants

- **Supervised**
  - Classification
  - Regression
  - Data consolidation
- **Unsupervised**
  - **Clustering**
  - Dimensionality Reduction
- **Weakly supervised/semi-supervised**

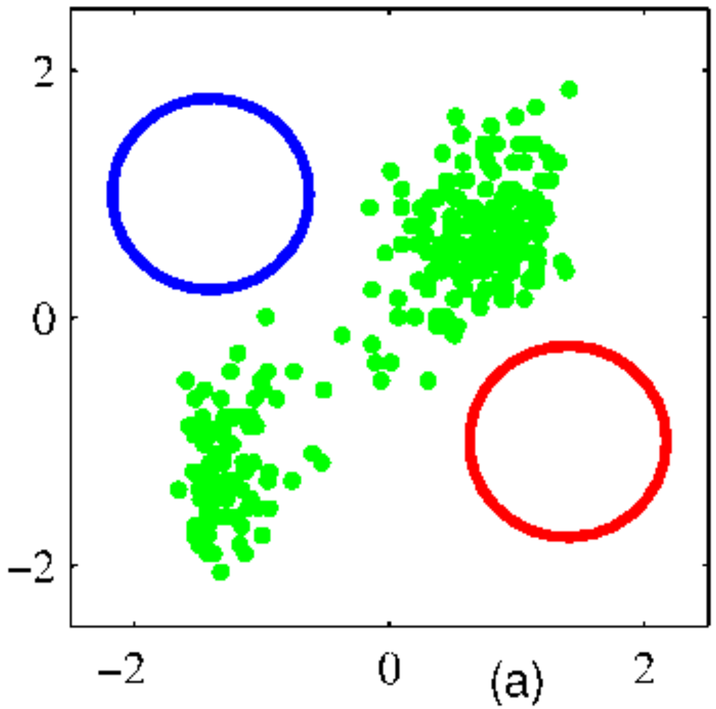
  Some data supervised, some unsupervised
- **Reinforcement learning**

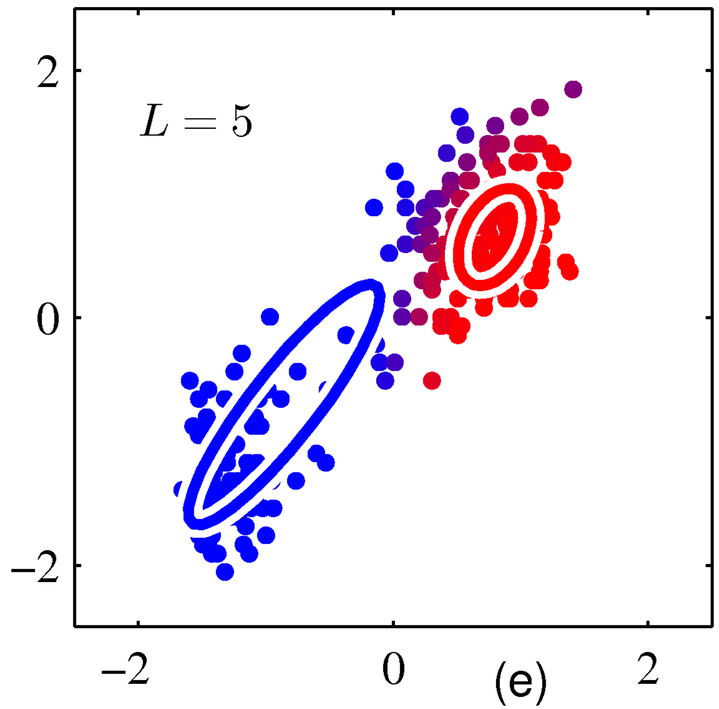
  Supervision: sparse reward for a sequence of decisions

Clustering: Group Points According to X

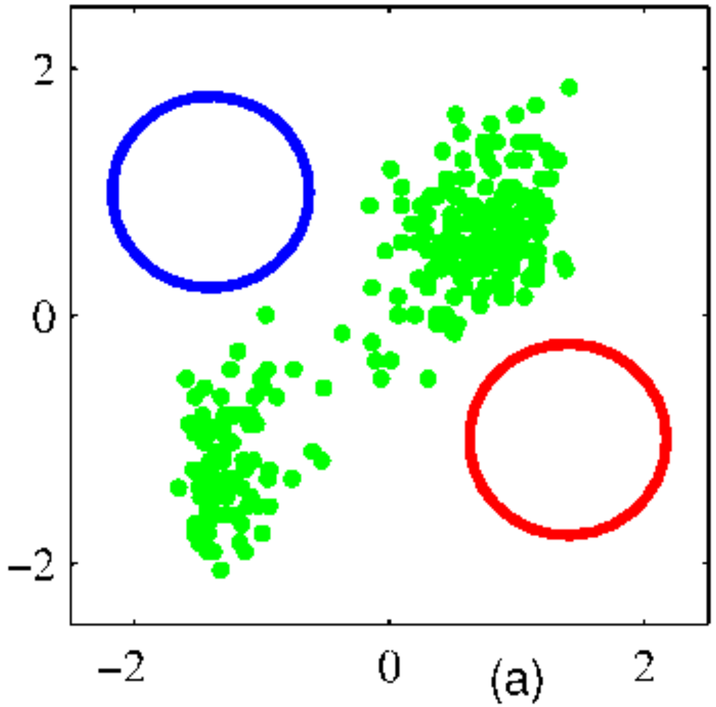


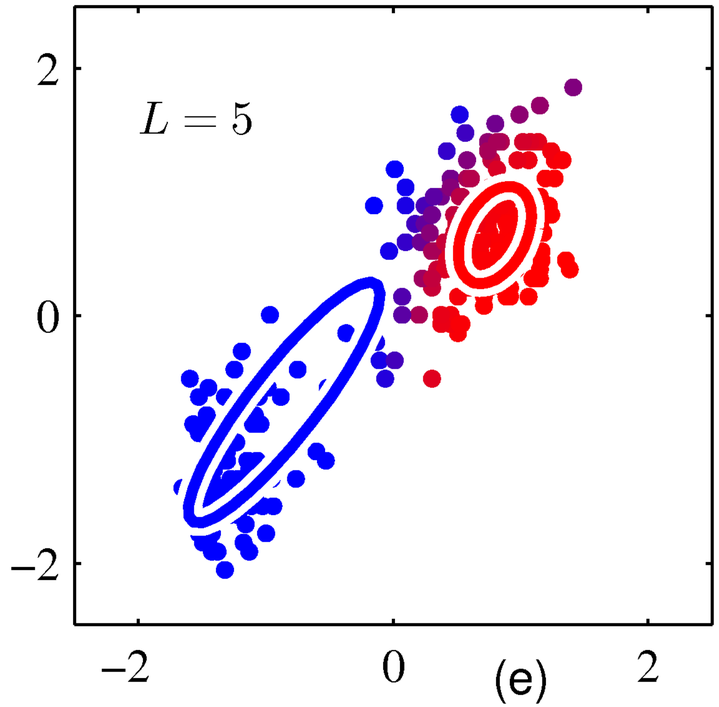
Clustering: Group Points According to X

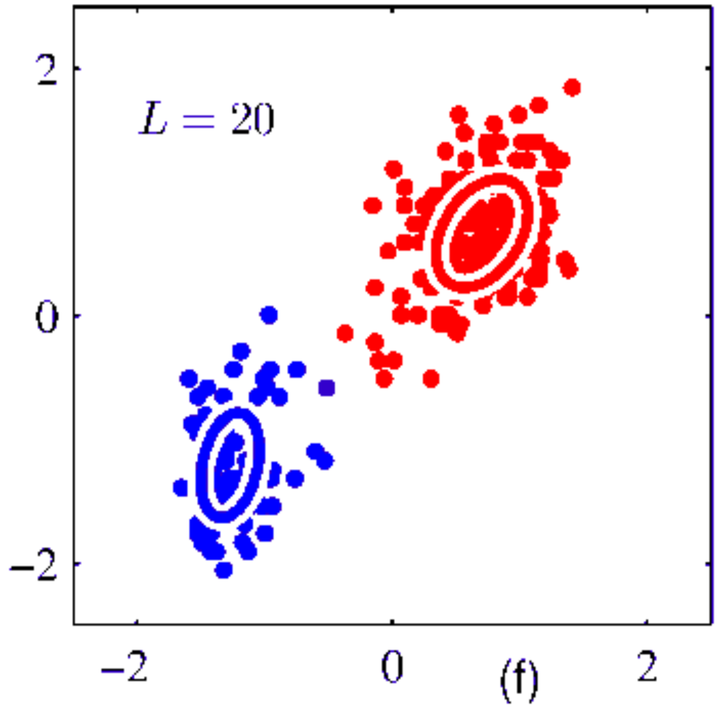




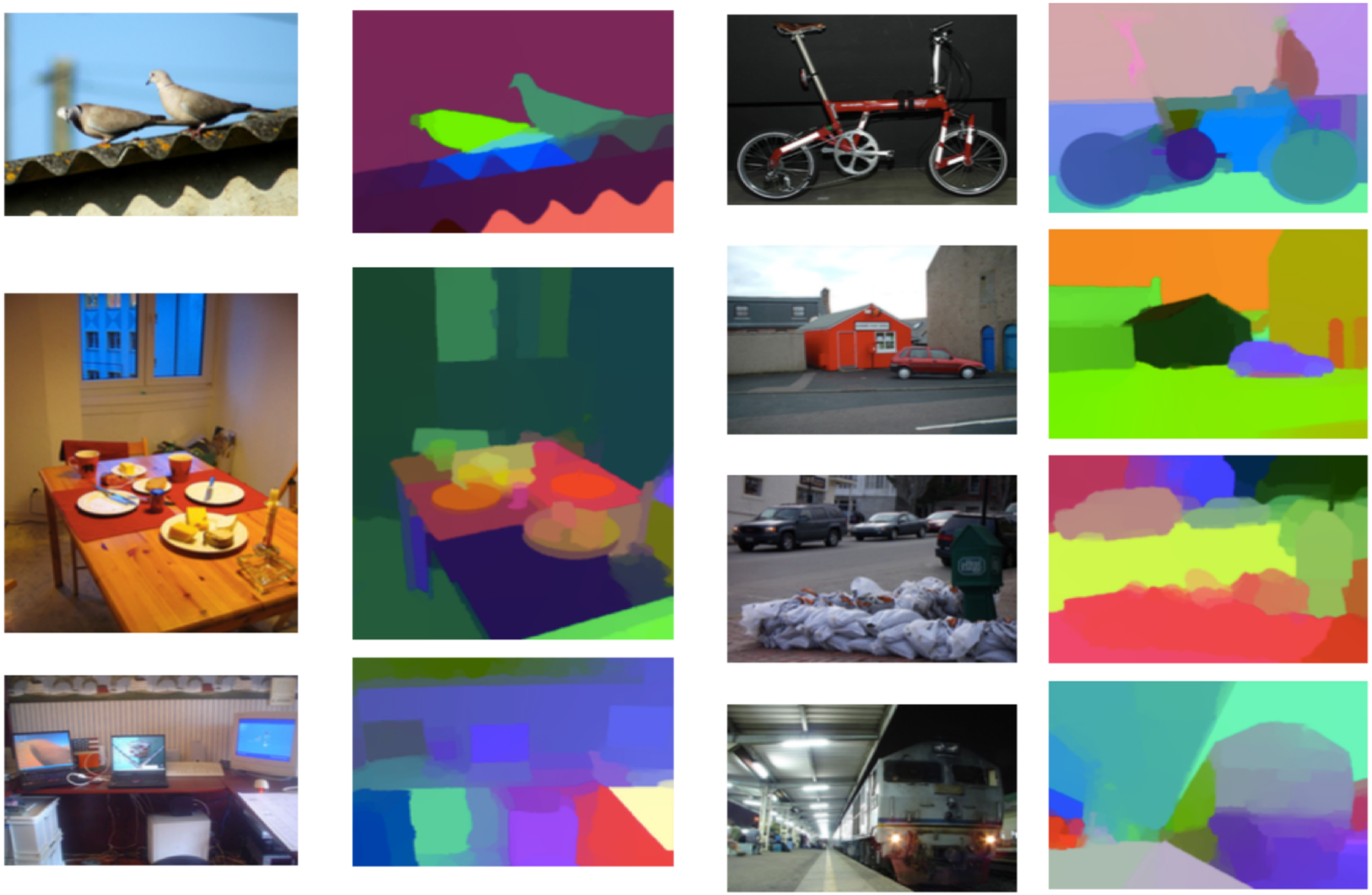
Clustering: Group Points According to X





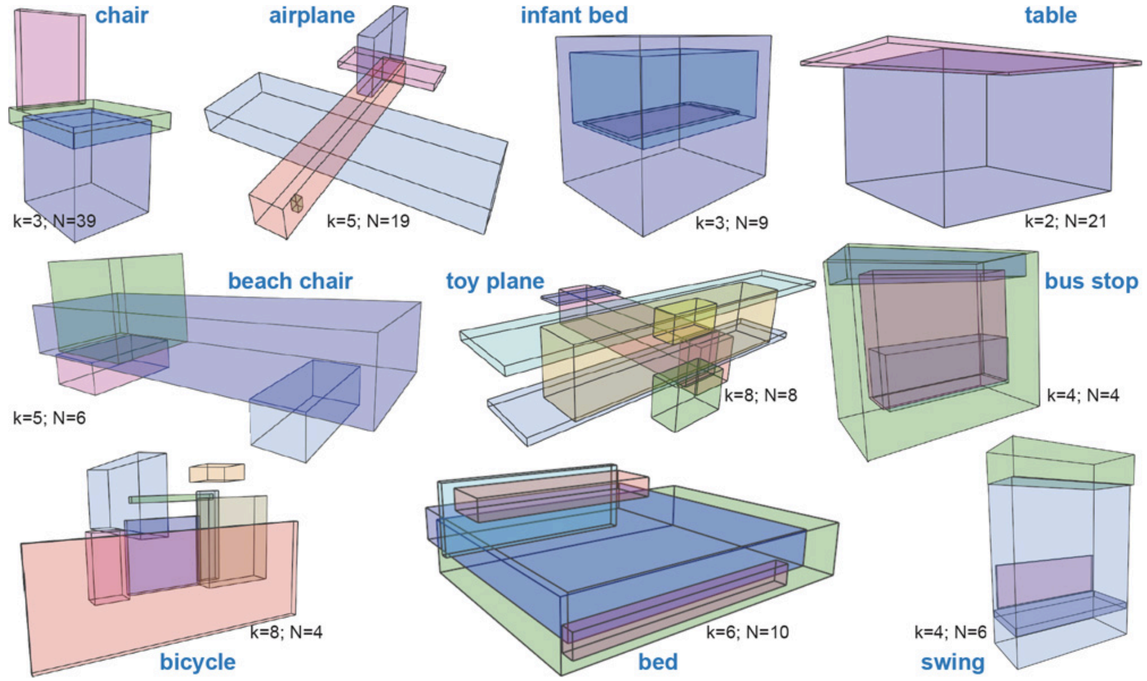


Clustering Examples: Image Segmentation using NCuts



Clustering Examples

[Chu et al., TVCG, 2009]



[Zheng et al., Eurographics, 2014]

Machine Learning Variants

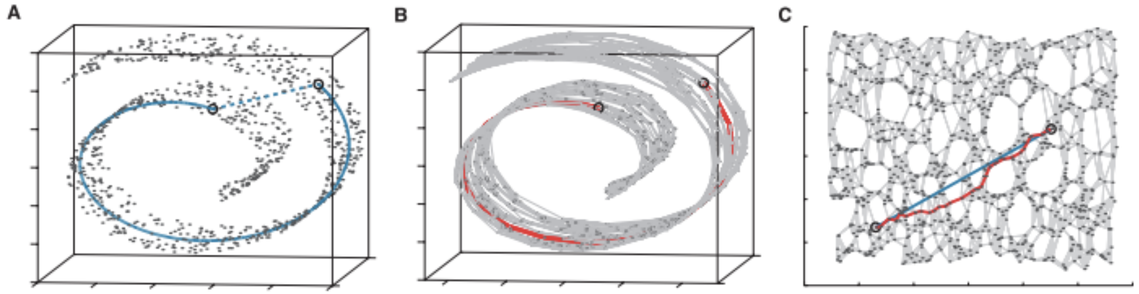
- **Supervised**
  - Classification
  - Regression
  - Data consolidation
- **Unsupervised**
  - Clustering
  - **Dimensionality Reduction**
- **Weakly supervised/semi-supervised**

  Some data supervised, some unsupervised
- **Reinforcement learning**

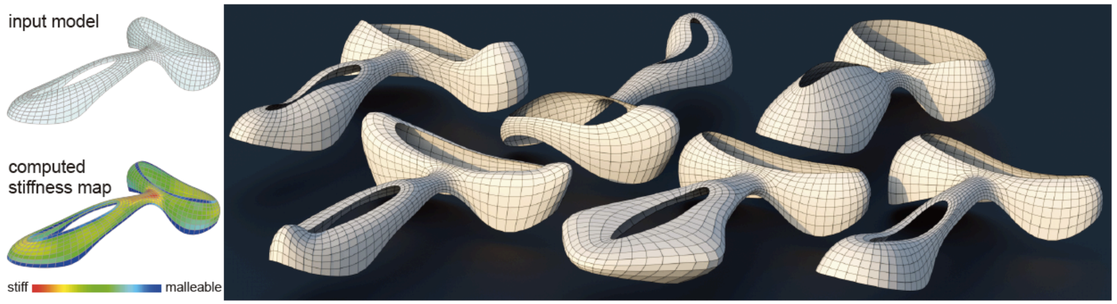
  Supervision: sparse reward for a sequence of decisions

Dimensionality Reduction (Manifold Learning)

Isomap

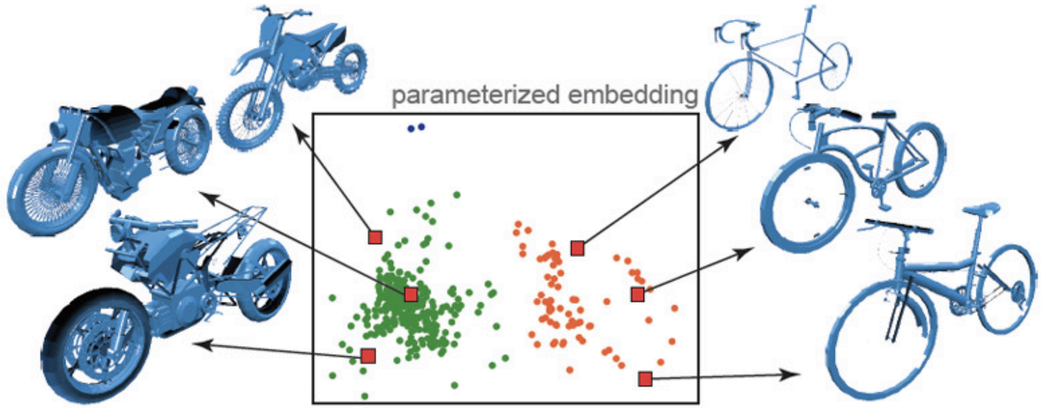


[Tenenbaum et al., Science, 2000]



[Yang et al., TOG, 2011]

Face Manifold



[Averkiou et al., Eurographics, 2014]

Example of Nonlinear Manifold: Faces

$\mathbf{x}_1$

$\mathbf{x}_2$

Example of Nonlinear Manifold: Faces

$\mathbf{x}_1$

$\frac{1}{2}(\mathbf{x}_1 + \mathbf{x}_2)$

$\mathbf{x}_2$

Example of Nonlinear Manifold: Faces

X

$\mathbf{x}_1$

$\frac{1}{2}(\mathbf{x}_1 + \mathbf{x}_2)$

$\mathbf{x}_2$

Moving Along Learned Face Manifold

Trajectory along the “male” dimension

[Lample et. al. Fader Networks, NIPS 2017]

Moving Along Learned Face Manifold

Trajectory along the “male” dimension

Trajectory along the “young” dimension

[Lample et. al. Fader Networks, NIPS 2017]

Notations: Eigenvectors and Eigenvalues

- All eigenvalues of symmetric matrices are real.
- Any real symmetric nxn matrix has a set of **n** mutually orthogonal eigenvectors.

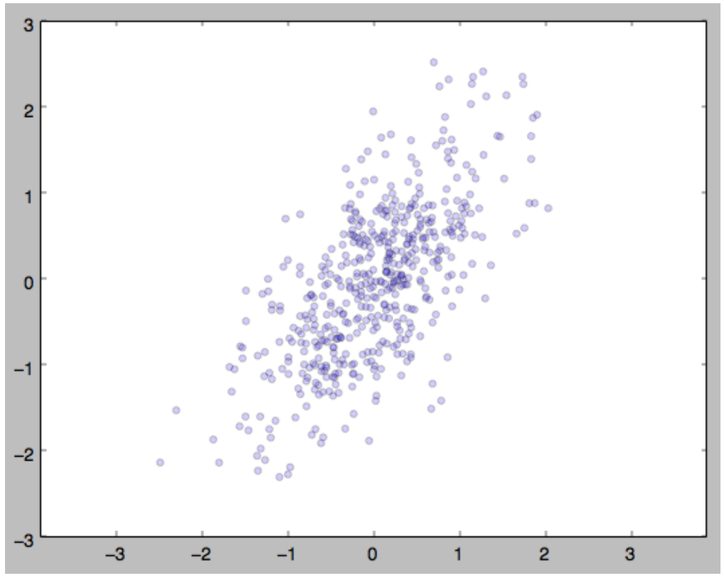
$$\mathbf{y} = \mathbf{A}\mathbf{x}$$

$$\mathbf{A}\mathbf{e}_i = \lambda_i \mathbf{e}_i$$

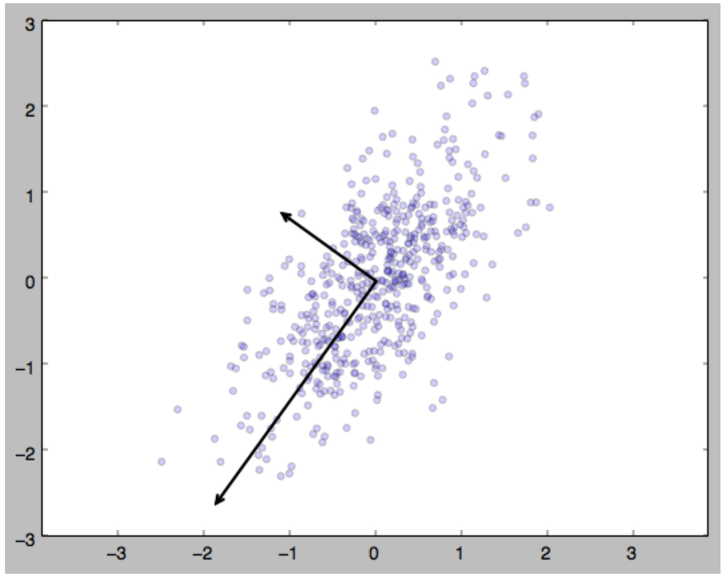
$$\mathbf{T} = [\mathbf{v}_1\ \mathbf{v}_2 \dots]$$

$$\mathbf{T}^{-1} A \mathbf{T} = \mathrm{diag}(\lambda_1, \lambda_2, \dots)$$

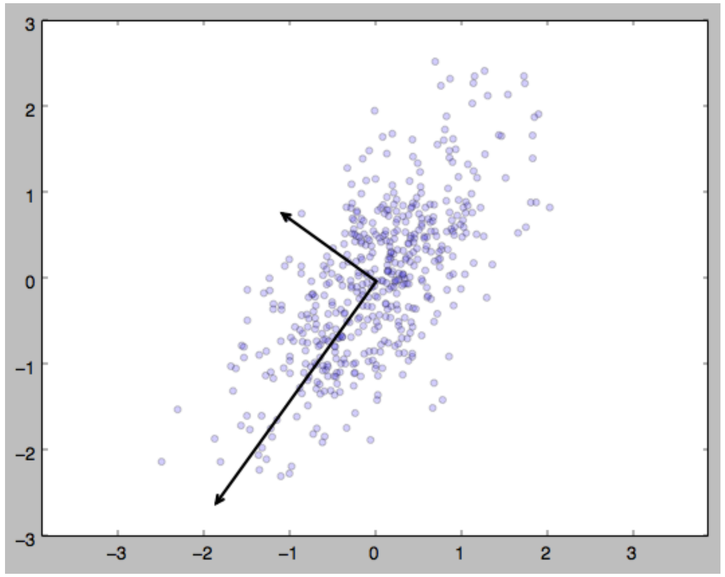
Code Example



Code Example

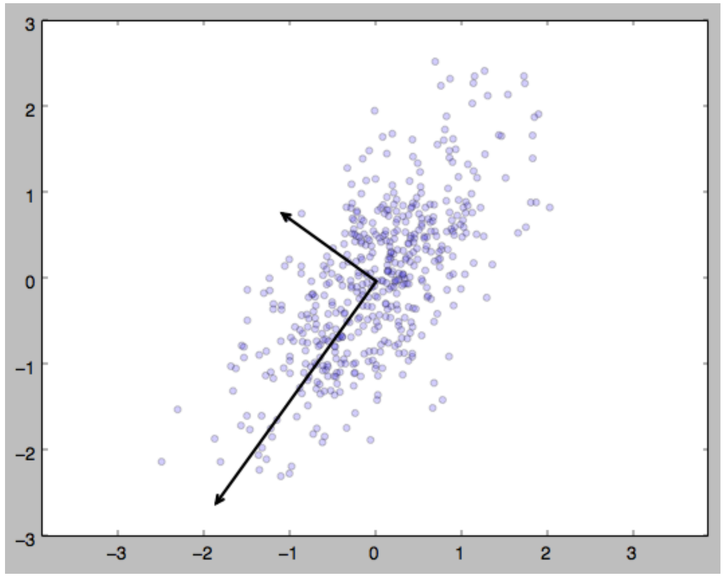


Code Example



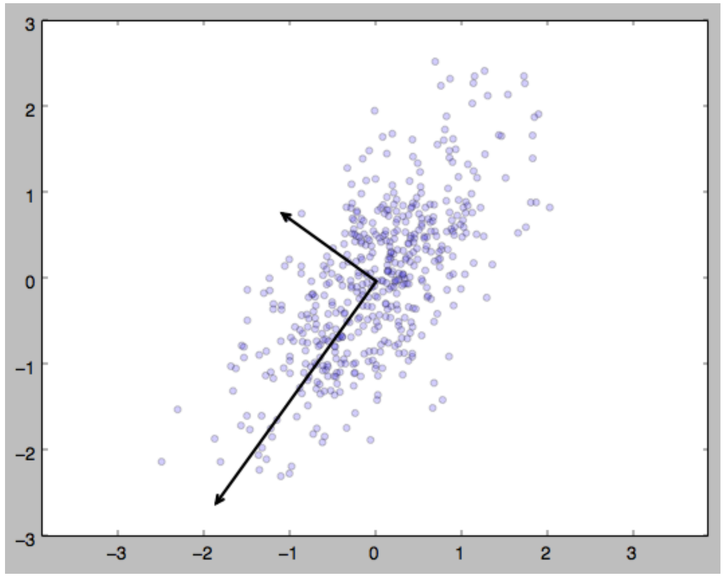

```
rng = np.random.RandomState(10)
X = np.dot(rng.rand(2, 2), rng.randn(2, 500)).T

mean_vec = np.mean(X, axis=0)
cov_mat = (X - mean_vec).T.dot((X - mean_vec)) / (X.shape[0]-1)
eig_vals, eig_vecs = np.linalg.eig(cov_mat)
```

Code Example



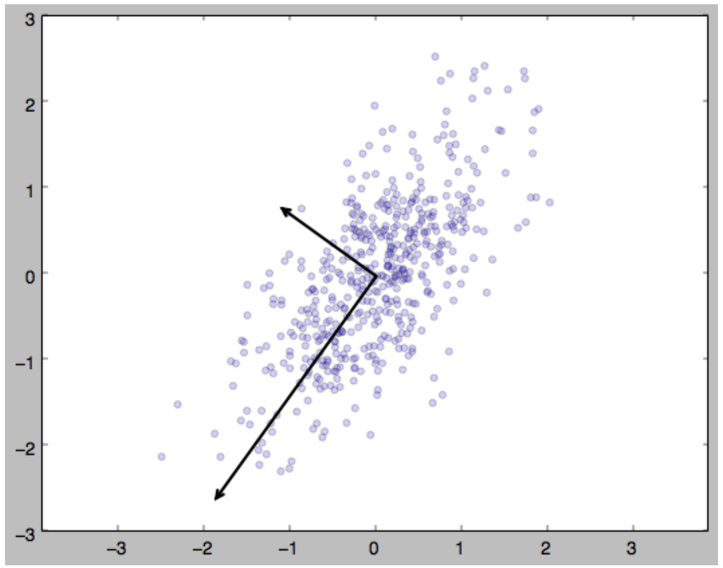

```
rng = np.random.RandomState(10)
X = np.dot(rng.rand(2, 2), rng.randn(2, 500)).T

mean_vec = np.mean(X, axis=0)
cov_mat = (X - mean_vec).T.dot((X - mean_vec)) / (X.shape[0]-1)
eig_vals, eig_vecs = np.linalg.eig(cov_mat)
```

Code Example



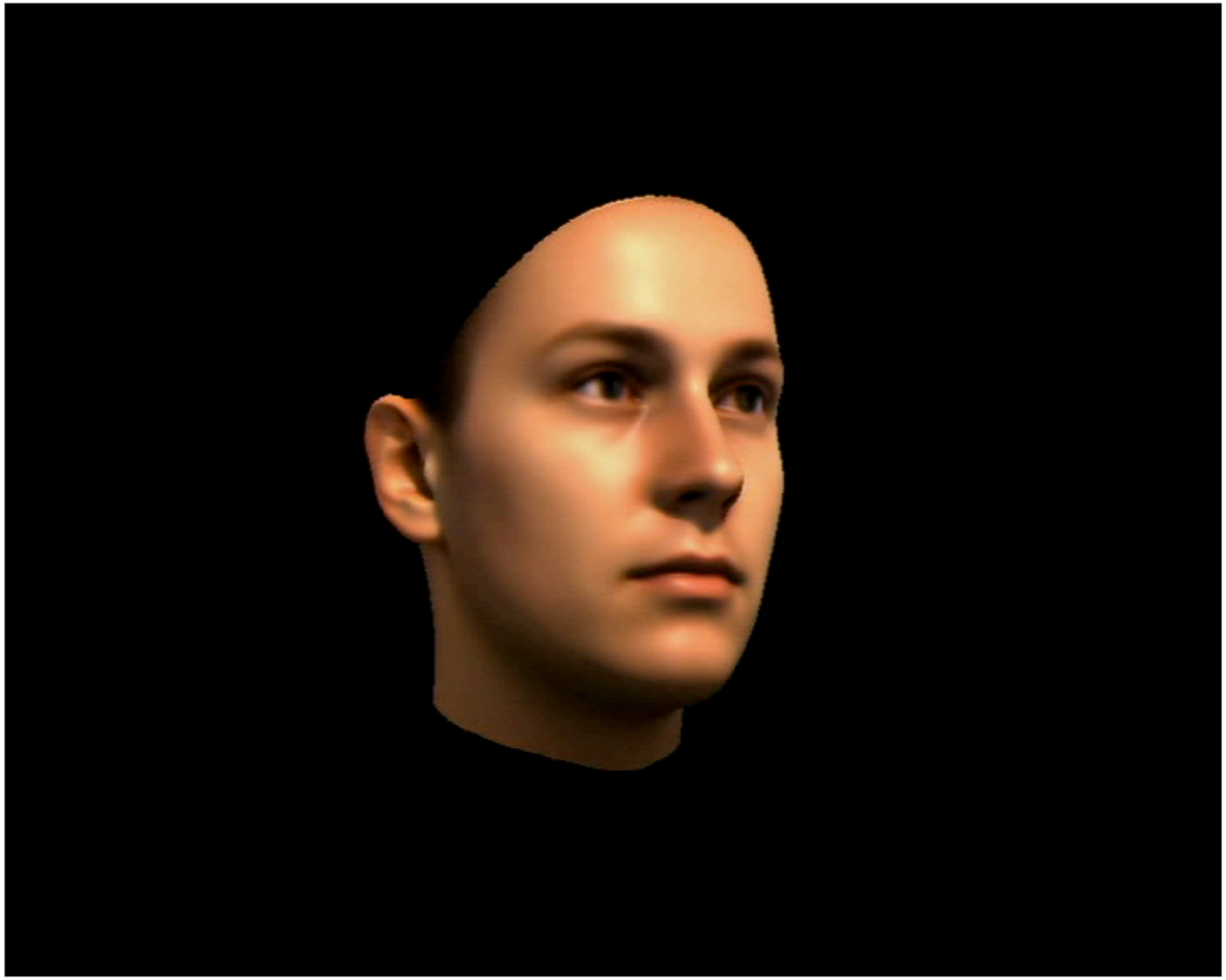

```
rng = np.random.RandomState(10)
X = np.dot(rng.rand(2, 2), rng.randn(2, 500)).T

mean_vec = np.mean(X, axis=0)
cov_mat = (X - mean_vec).T.dot((X - mean_vec)) / (X.shape[0]-1)
eig_vals, eig_vecs = np.linalg.eig(cov_mat)
```

Code Example

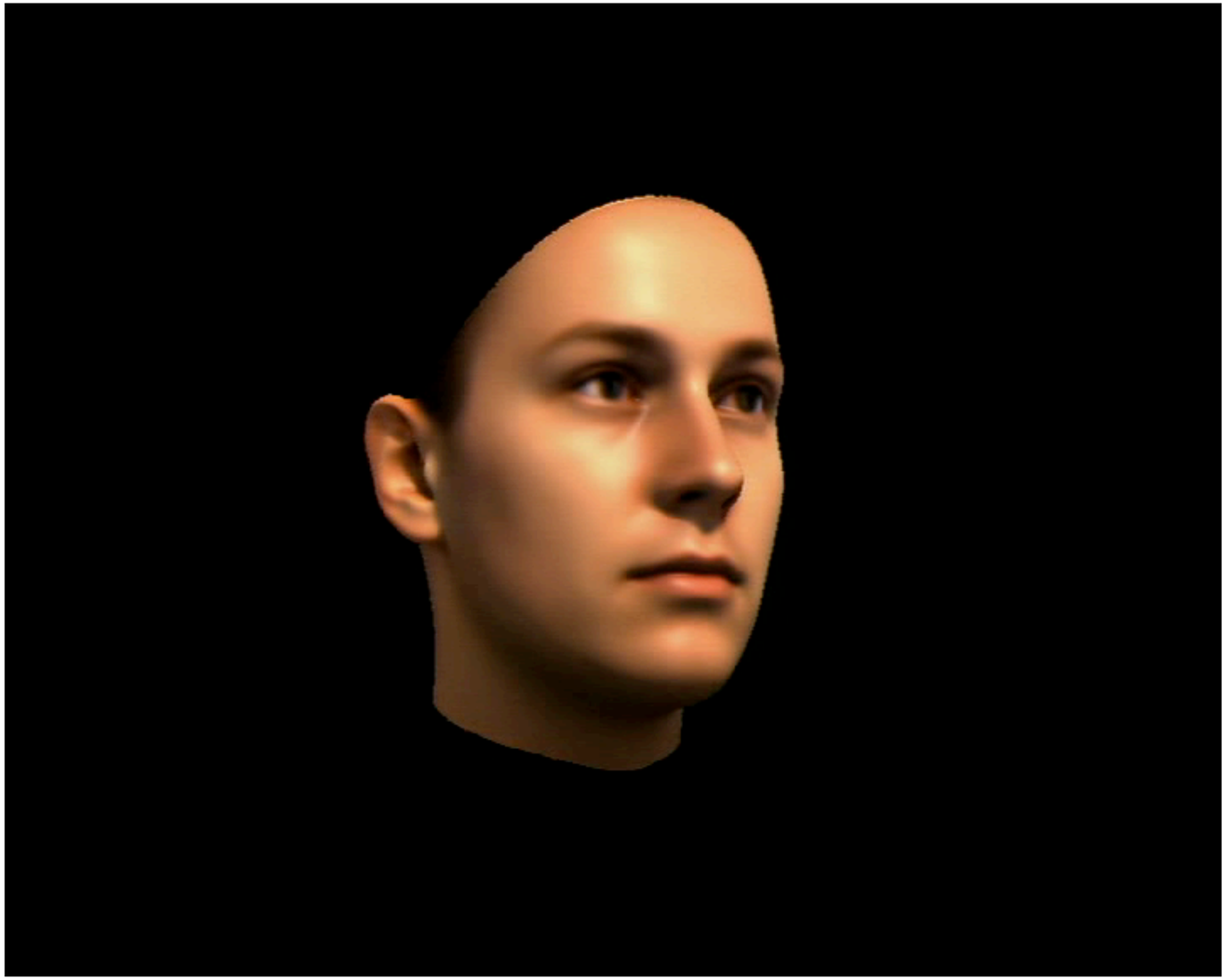



```
mean_vec = np.mean(X, axis=0)
cov_mat = (X - mean_vec).T.dot((X - mean_vec)) / (X.shape[0]-1)
matU, sigma, matV = np.linalg.svd(cov_mat)
```

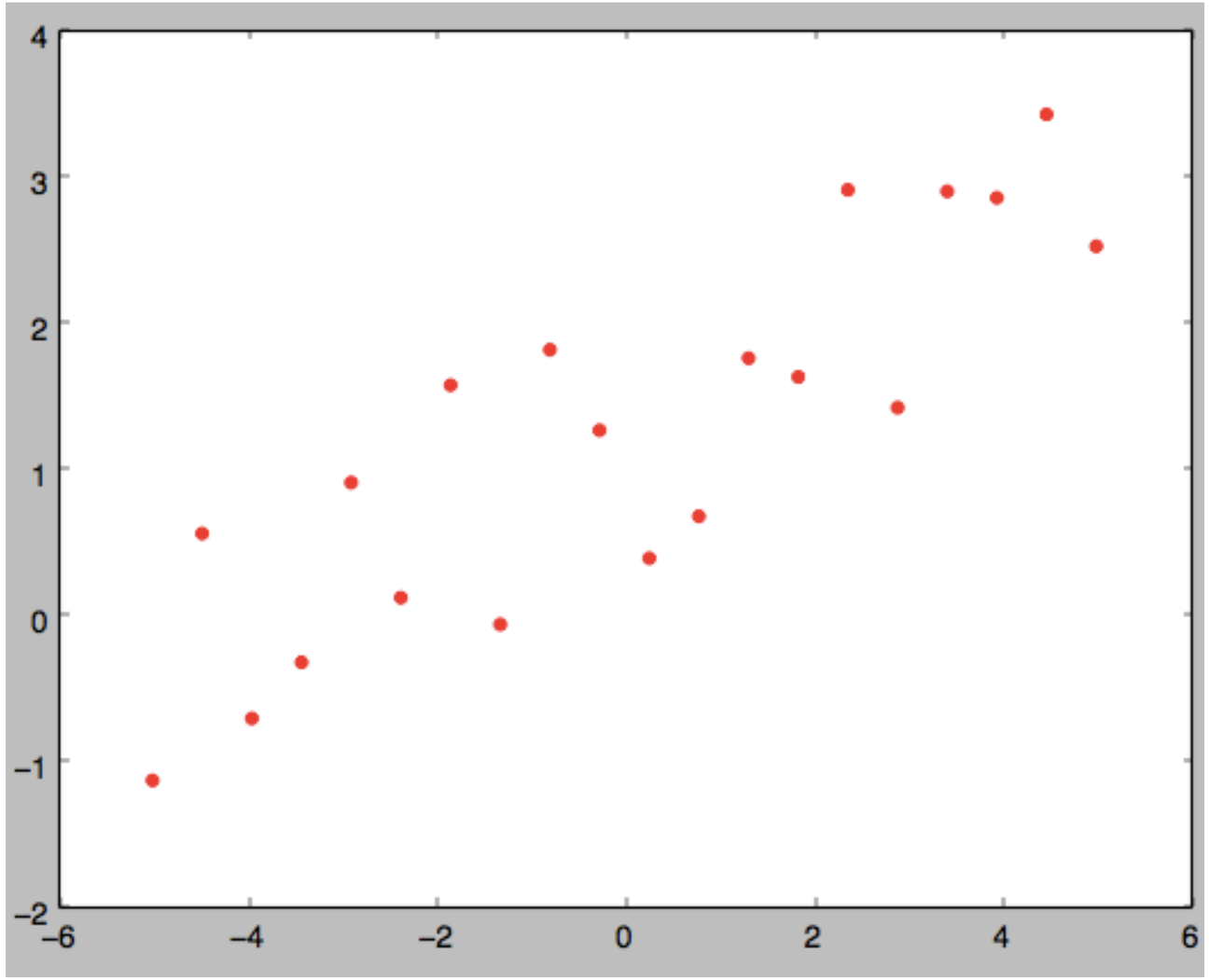
Morphable Faces



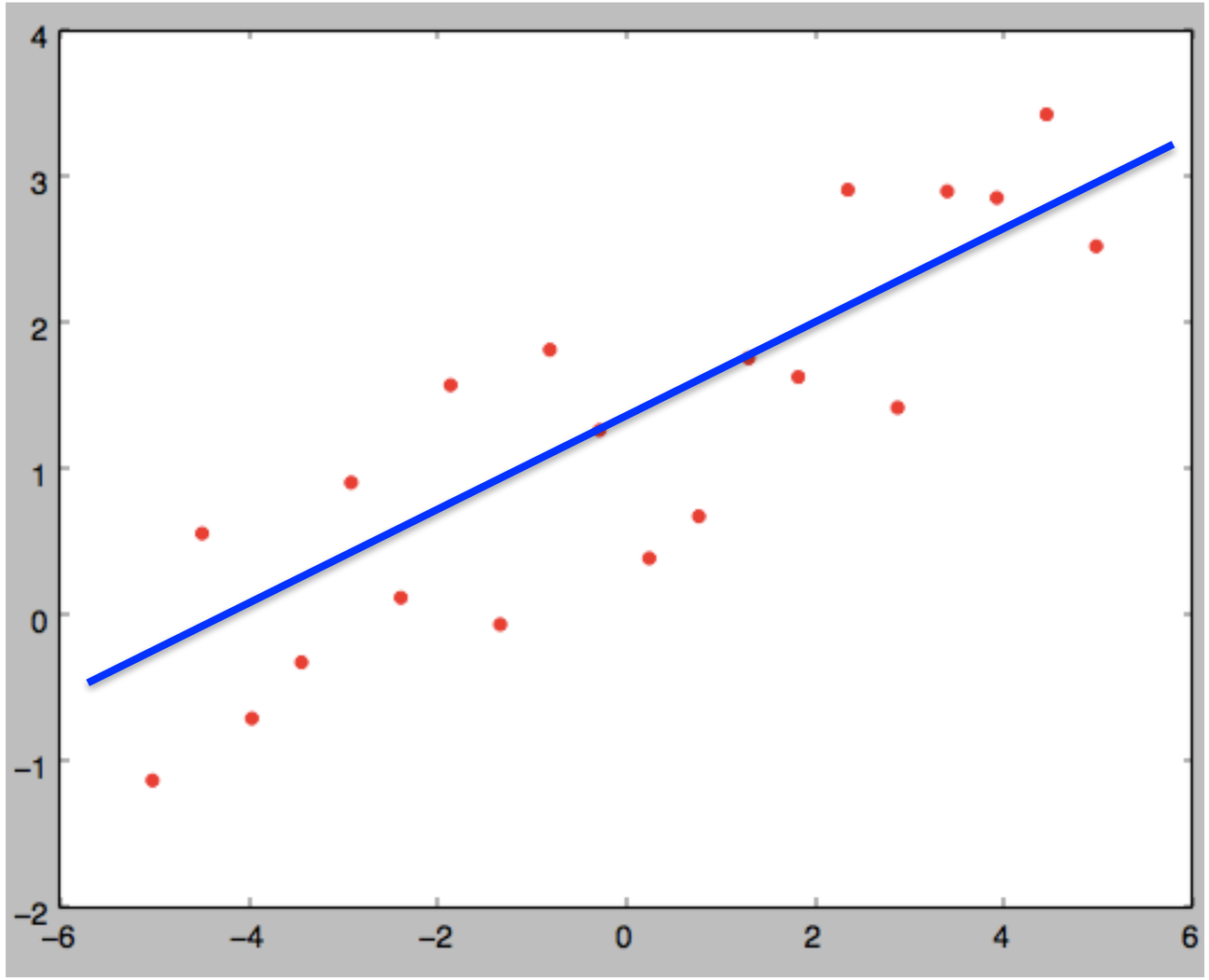
Morphable Faces



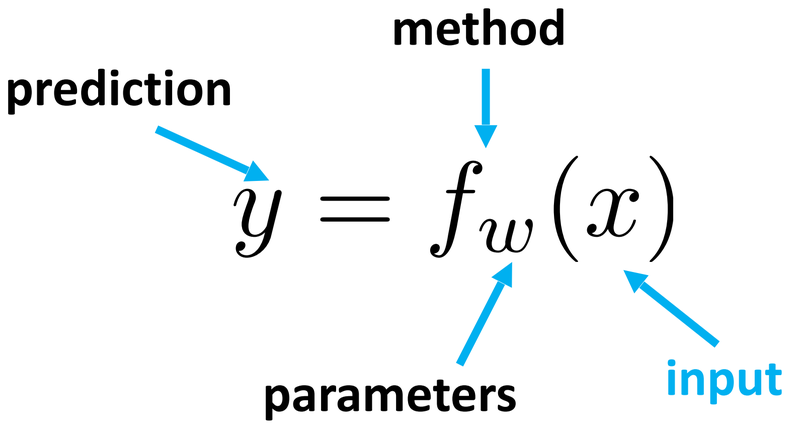
Regression: Continuous Output



Regression: Continuous Output



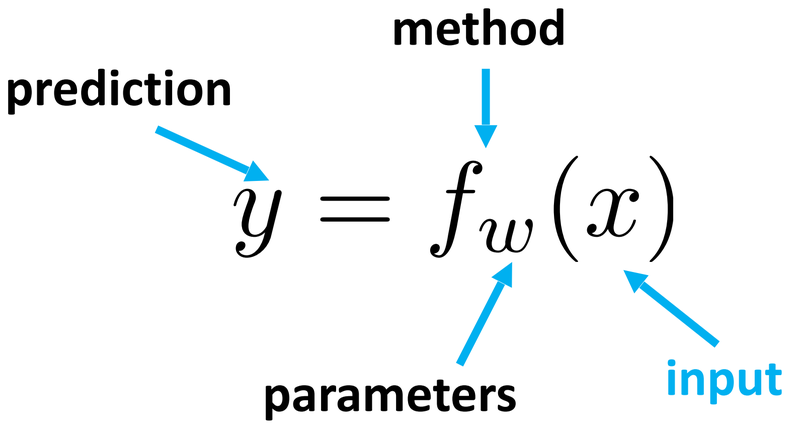
Learning a Function



Calculus $x \in \mathbb{R}$

Vector calculus $\mathbf{x} \in \mathbb{R}^d$

Learning a Function

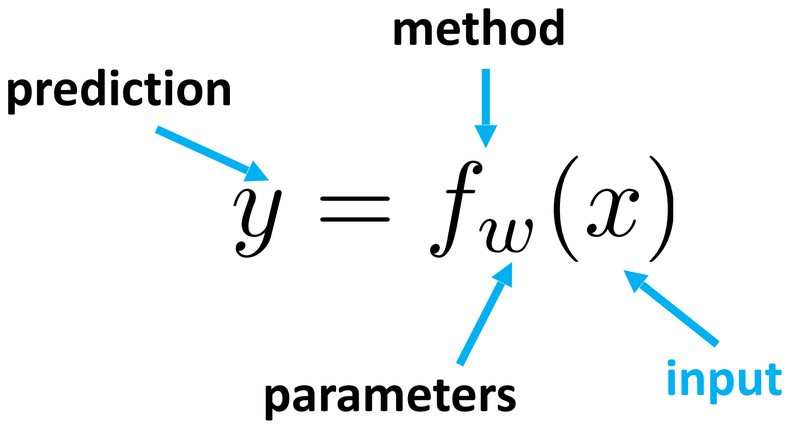


Calculus $x \in \mathbb{R}$

Vector calculus $\mathbf{x} \in \mathbb{R}^d$

Machine learning: can work also for discrete inputs, strings, images, meshes, animations, …

Learning a Function



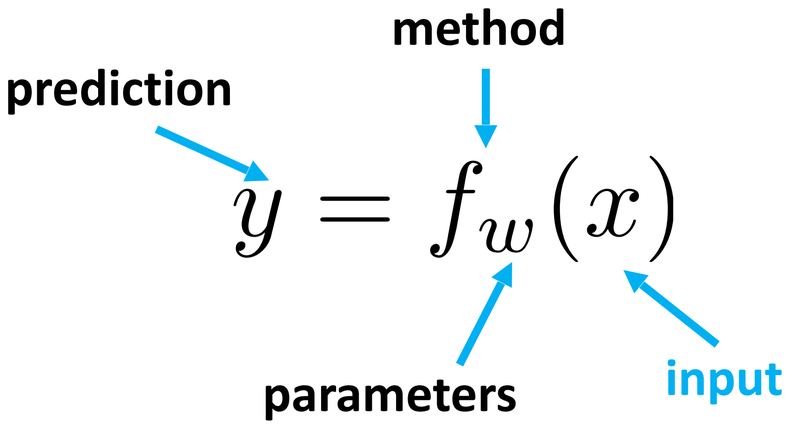
Calculus $x \in \mathbb{R}$

Vector calculus $\mathbf{x} \in \mathbb{R}^d$

Classification: $y \in \{0, 1\}$

Machine learning: can work also for discrete inputs, strings, images, meshes, animations, …

Learning a Function



Calculus $x \in \mathbb{R}$

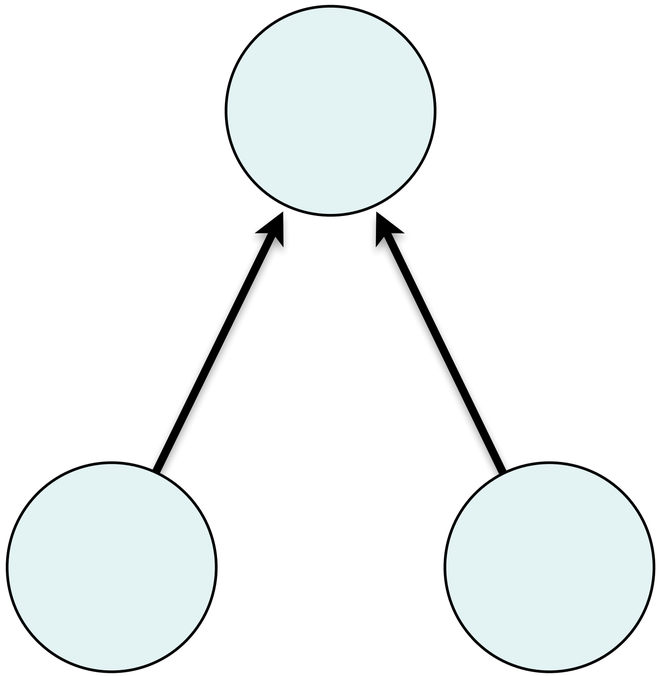
Vector calculus $\mathbf{x} \in \mathbb{R}^d$

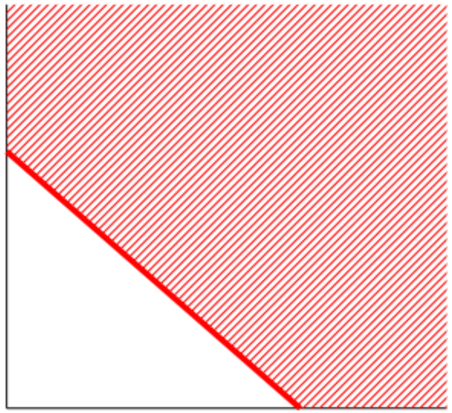
Classification: $y \in \{0, 1\}$

Regression: $y \in \mathbb{R}$

Machine learning: can work also for discrete inputs, strings, images, meshes, animations, …

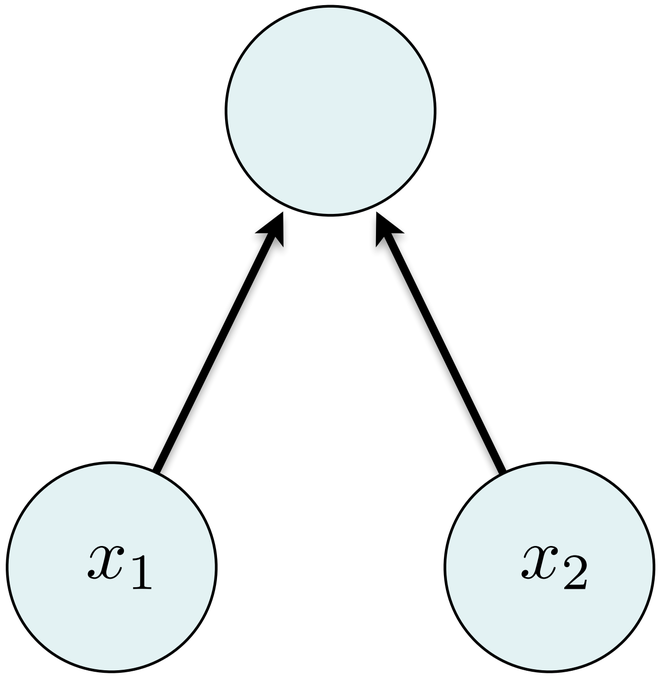
Learning a Linear Separator/Classifier

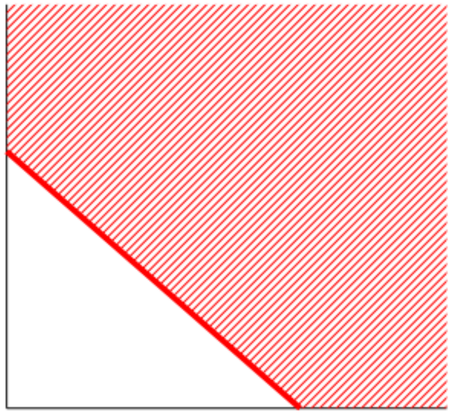




separating hyperplane

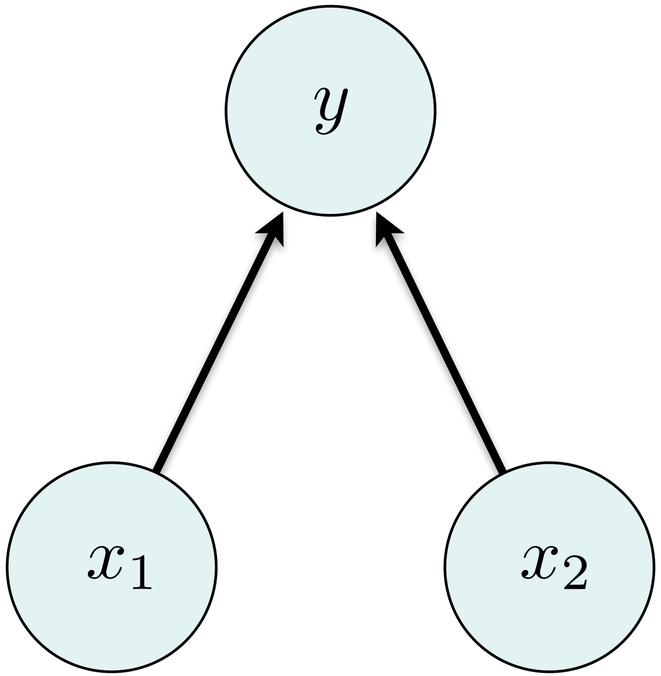
Learning a Linear Separator/Classifier

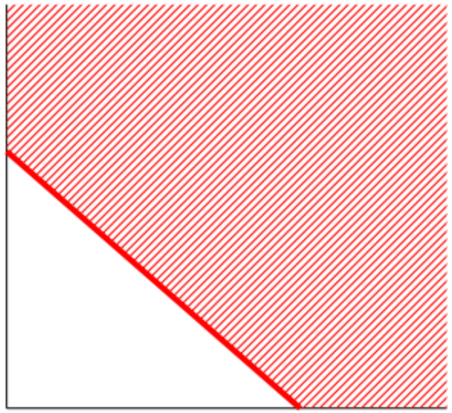




separating hyperplane

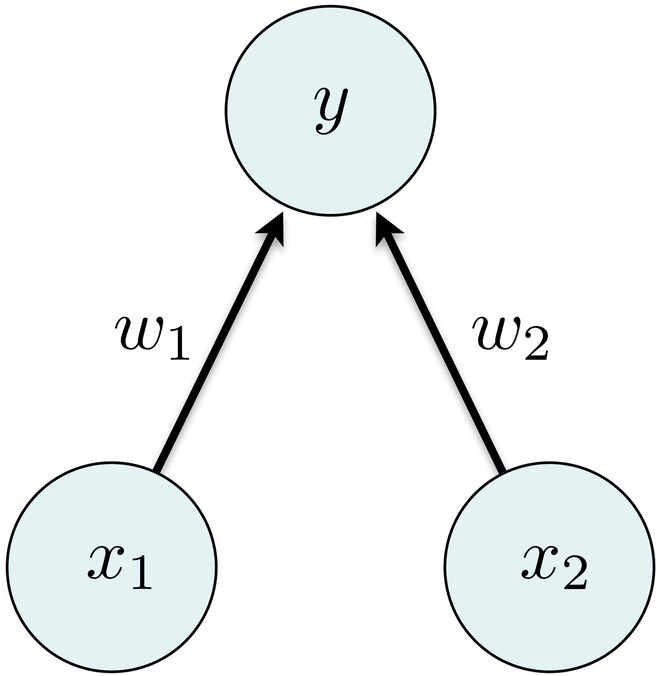
Learning a Linear Separator/Classifier

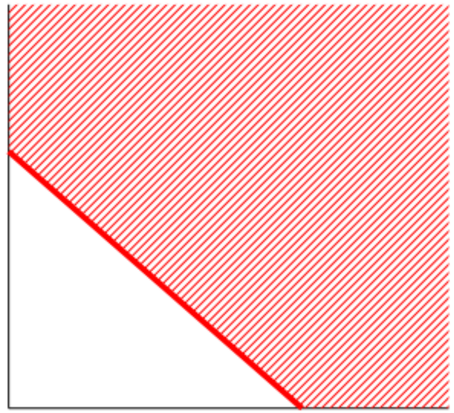




separating hyperplane

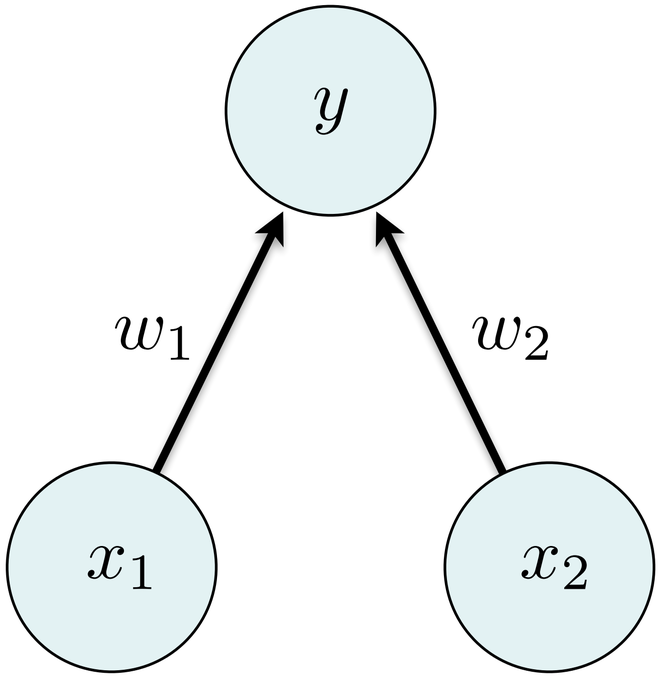
Learning a Linear Separator/Classifier

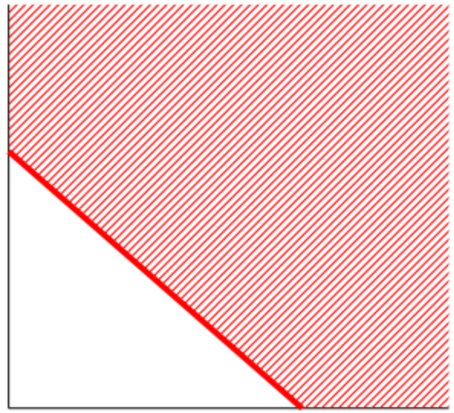




separating hyperplane

Learning a Linear Separator/Classifier

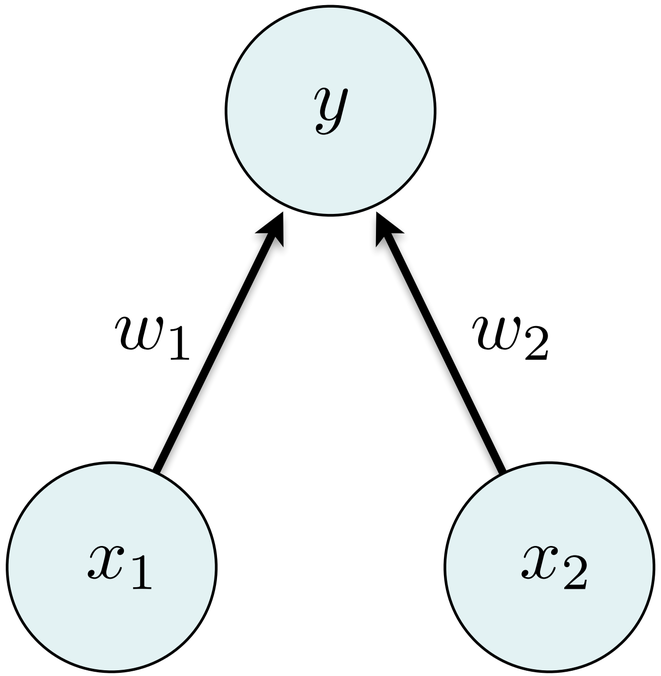


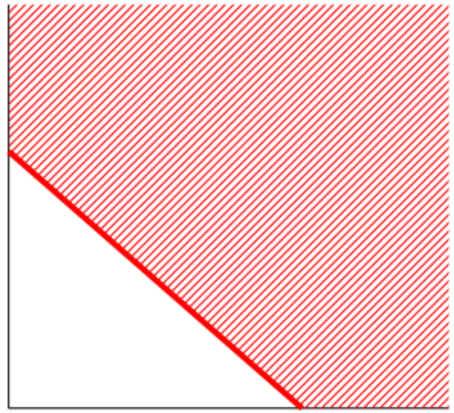


separating hyperplane

$$y = f(w_1 x_1 + w_2 x_2)$$

Learning a Linear Separator/Classifier

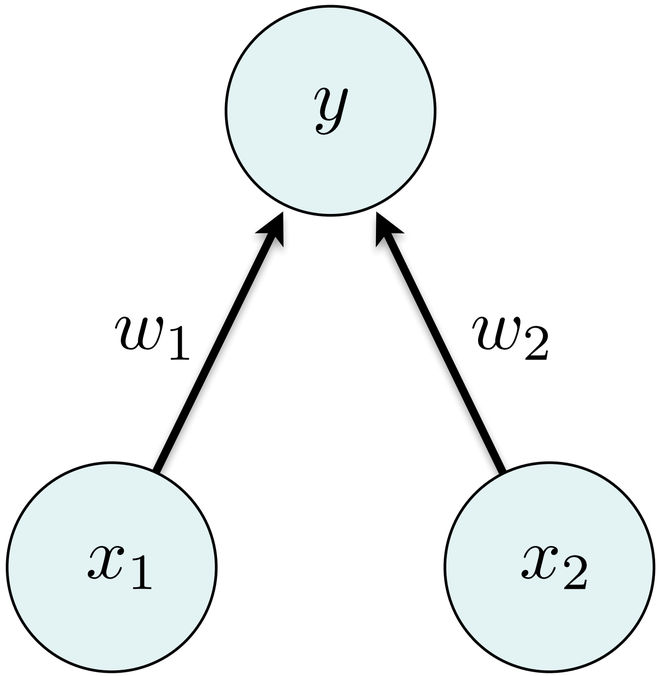


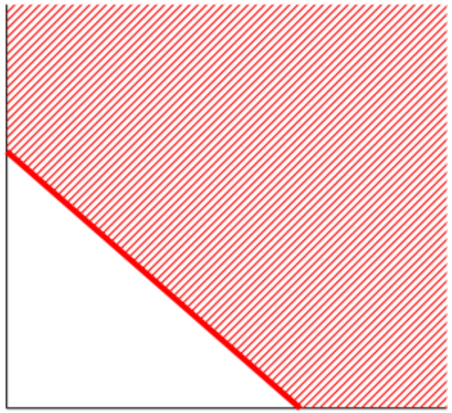


separating hyperplane

$$y = f(w_1 x_1 + w_2 x_2) = \mathcal{H}(w_1 x_1 + w_2 x_2)$$

Learning a Linear Separator/Classifier



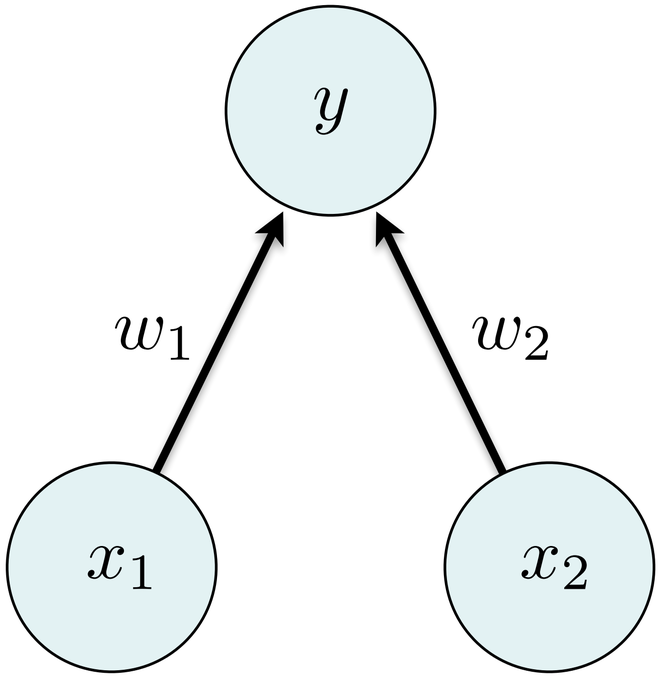


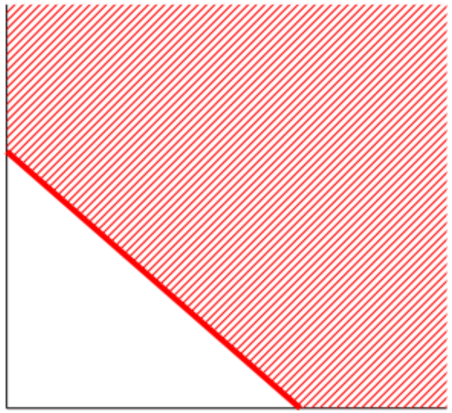
separating hyperplane

fixed non-linearity

$$y = f(w_1 x_1 + w_2 x_2) = \mathcal{H}(w_1 x_1 + w_2 x_2)$$

Learning a Linear Separator/Classifier





separating hyperplane

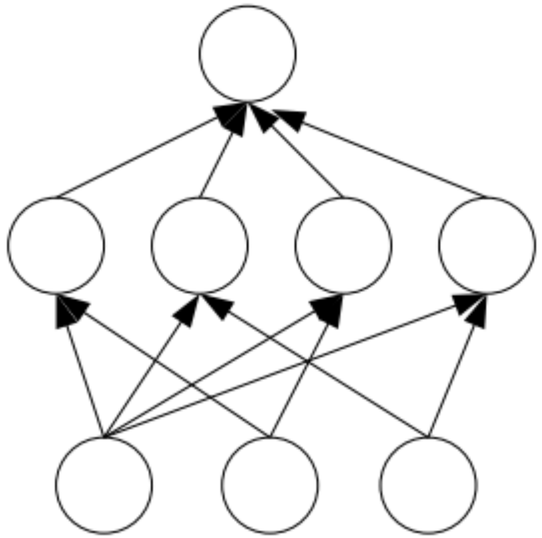
fixed non-linearity

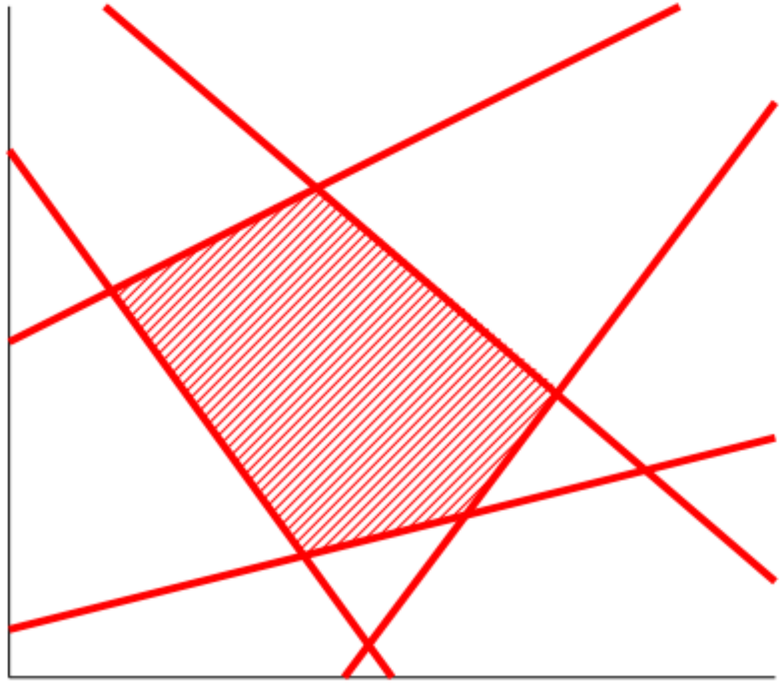
learned

$$y = f(w_1 x_1 + w_2 x_2) = \mathcal{H}(w_1 x_1 + w_2 x_2)$$

Combining Simple Functions/Classifiers

2 layers of trainable weights

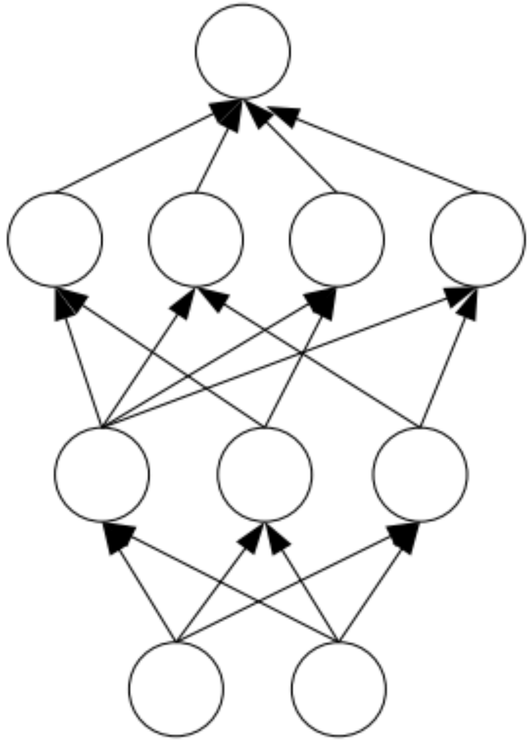


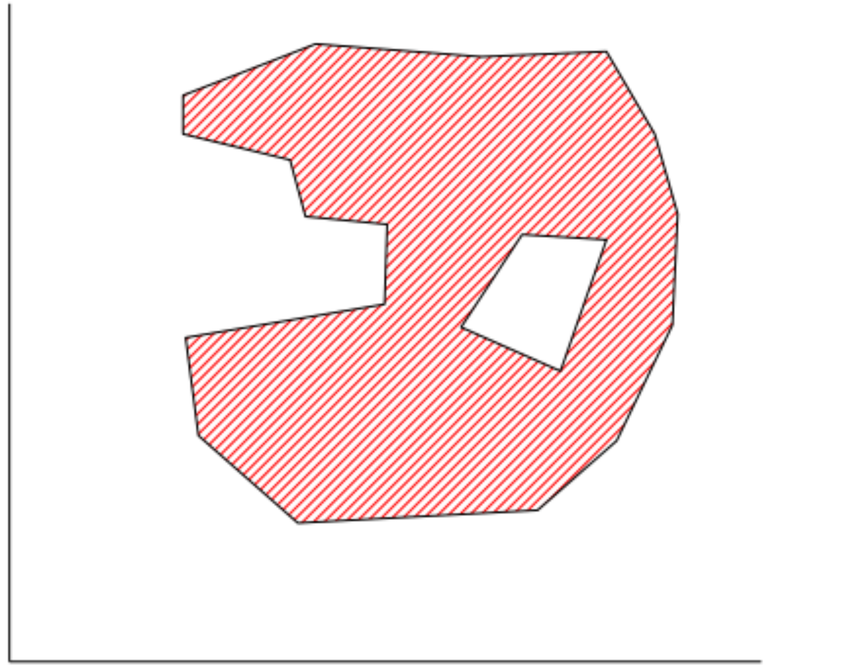


convex polygon region

Combining Simple Functions/Classifiers

3 layers of trainable weights





composition of polygons: convex regions

Regression

1. Least Squares fitting

2. Nonlinear error function and gradient descent

3. Perceptron training (simple neural network)

Regression

1. **Least Squares fitting**

2. Nonlinear error function and gradient descent

3. Perceptron training (simple neural network)

Assumption: Linear Function

$$y = f_{\mathbf{w}}(\mathbf{x}) = f(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T \mathbf{x}$$

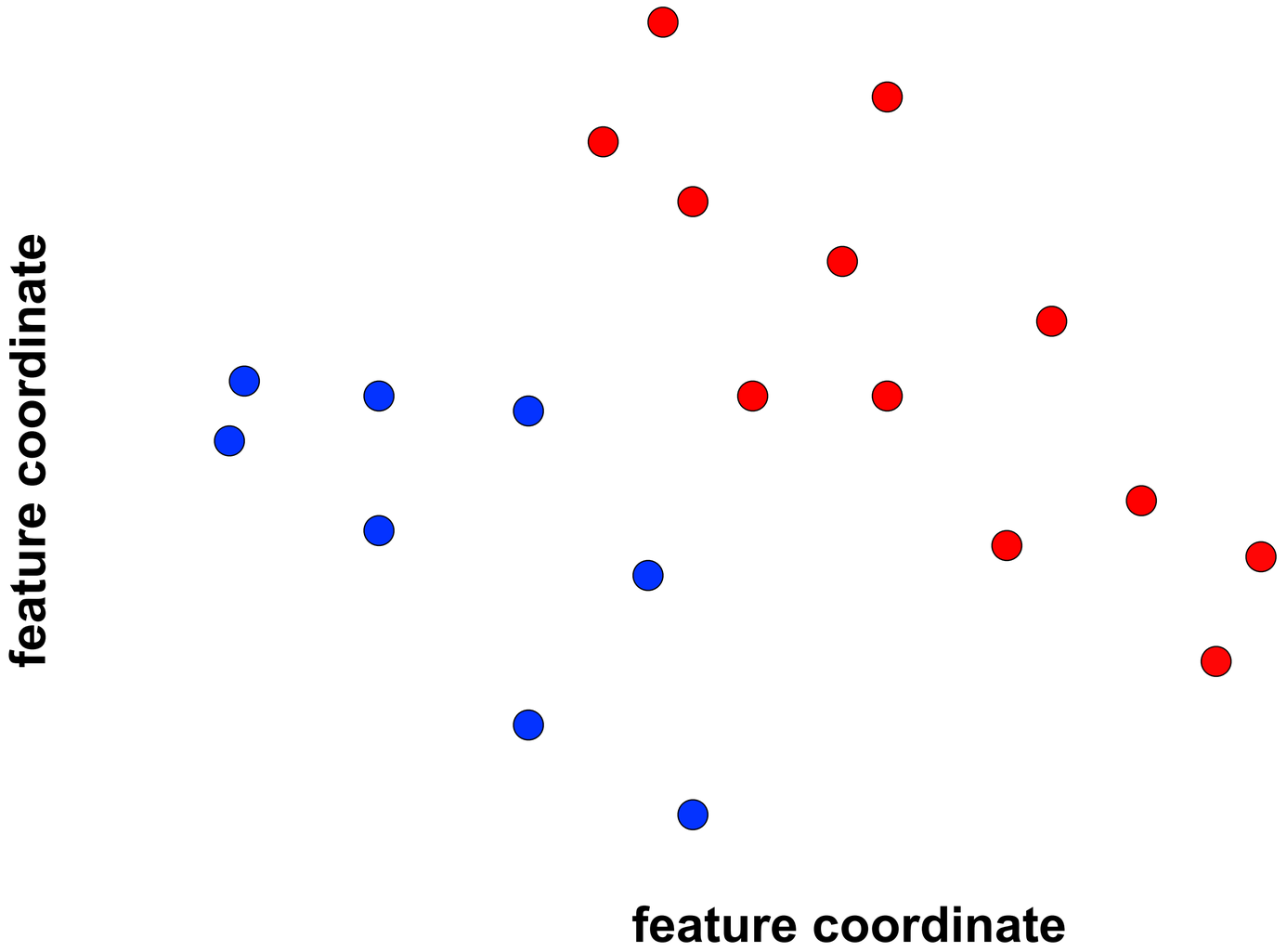
Assumption: Linear Function

$$y = f_{\mathbf{w}}(\mathbf{x}) = f(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T \mathbf{x}$$

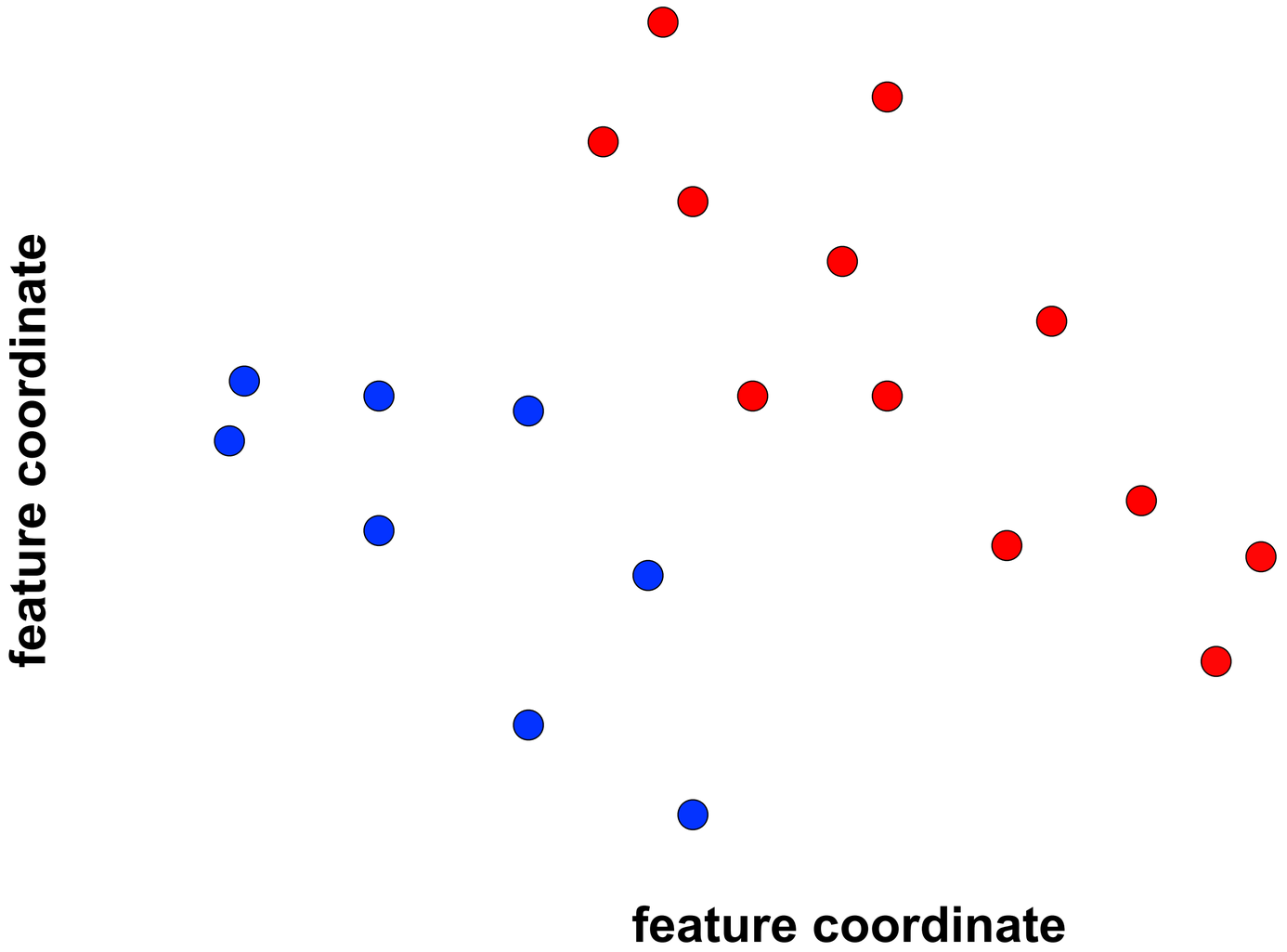
$$\mathbf{w}^T \mathbf{x} = \langle \mathbf{w}, \mathbf{x} \rangle = \sum_{d=1}^{D} \mathbf{w}_d \mathbf{x}_d$$

$$\mathbf{x} \in \mathbb{R}^D, \mathbf{w} \in \mathbb{R}^D$$

Reminder: Linear Classifier



Reminder: Linear Classifier

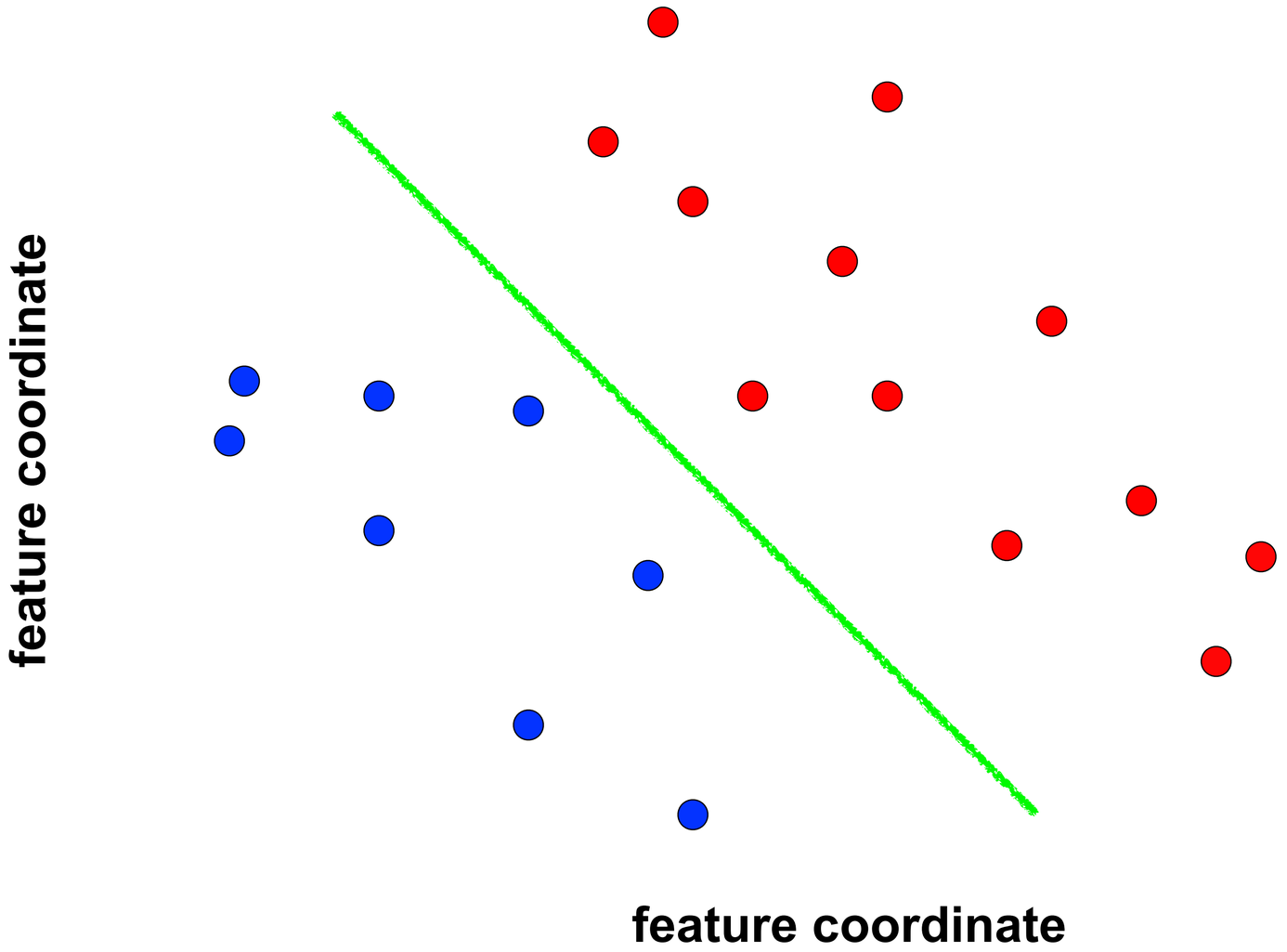


supervised setting

labelled input

$$y_t = \begin{cases} +1 & \bullet \\ -1 & \bullet \end{cases}$$

Reminder: Linear Classifier

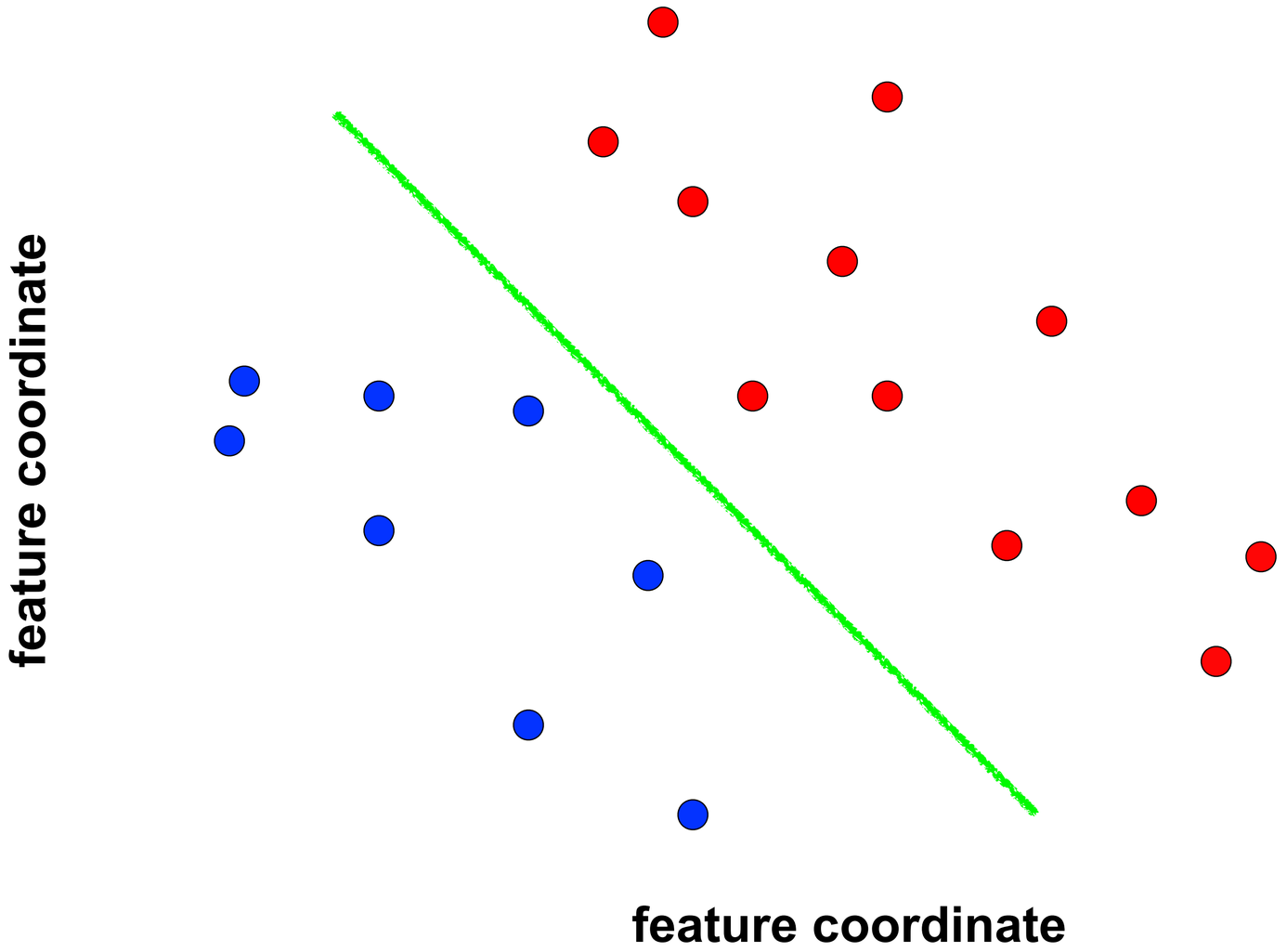


supervised setting

labelled input

$$y_t = \begin{cases} +1 & \bullet \\ -1 & \bullet \end{cases}$$

Reminder: Linear Classifier



$\mathbf{x}_i$ positive: $\mathbf{x}_i \cdot \mathbf{w} \geq 0$

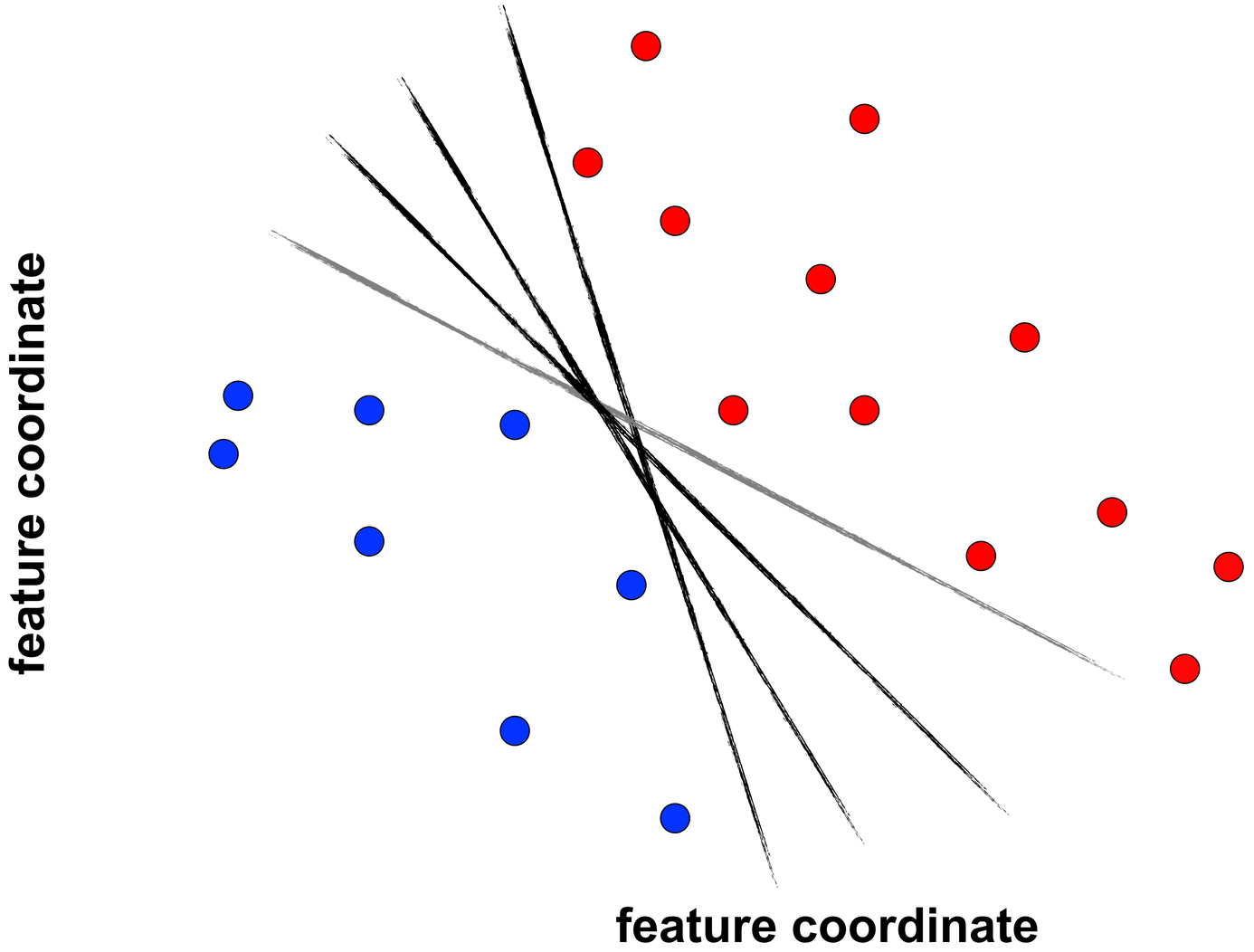
$\mathbf{x}_i$ negative: $\mathbf{x}_i \cdot \mathbf{w} < 0$

supervised setting

labelled input

$$y_t = \begin{cases} +1 & \bullet \\ -1 & \bullet \end{cases}$$

Which Line to Pick?



$\mathbf{x}_i$ positive: $\mathbf{x}_i \cdot \mathbf{w} \geq 0$

$\mathbf{x}_i$ negative: $\mathbf{x}_i \cdot \mathbf{w} < 0$

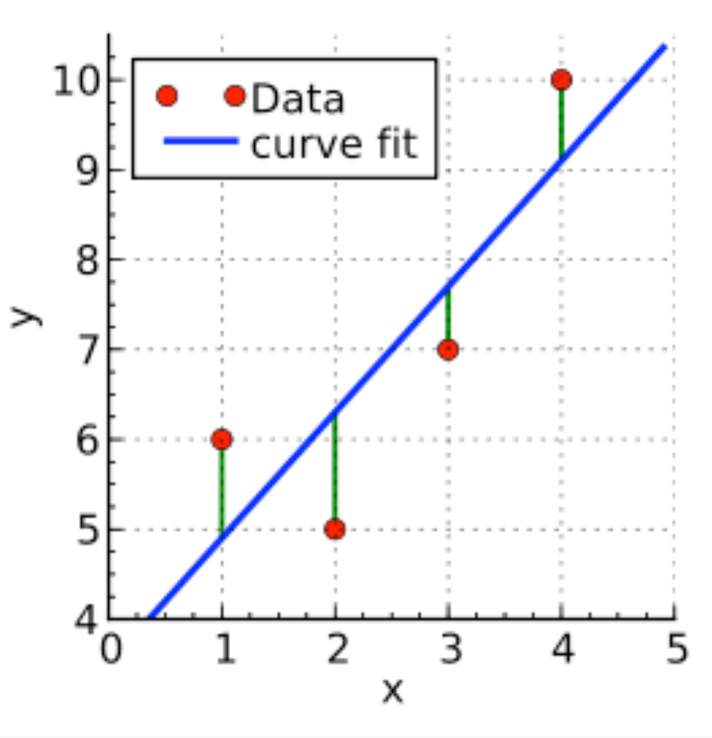
supervised setting

labelled input

$$y_t = \begin{cases} +1 & \bullet \\ -1 & \bullet \end{cases}$$

Sum of Square Errors *(MSE without the mean)*

$$y^i = \mathbf{w}^T \mathbf{x}^i + \epsilon^i$$

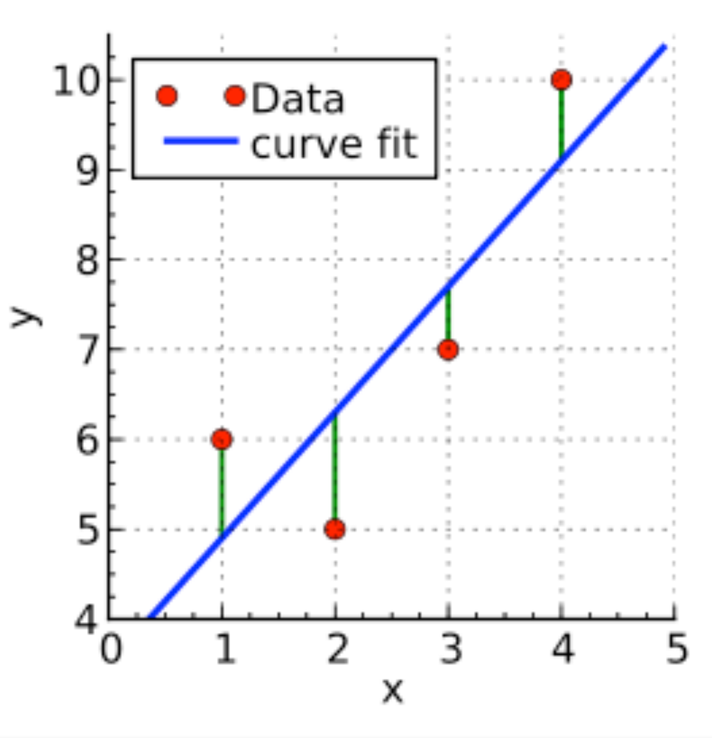


Sum of Square Errors *(MSE without the mean)*

$$y^i = \mathbf{w}^T \mathbf{x}^i + \epsilon^i$$

Loss function: sum of squared errors

$$L(\mathbf{w}) = \sum_{i=1}^{N} (\epsilon^i)^2$$



Sum of Square Errors *(MSE without the mean)*

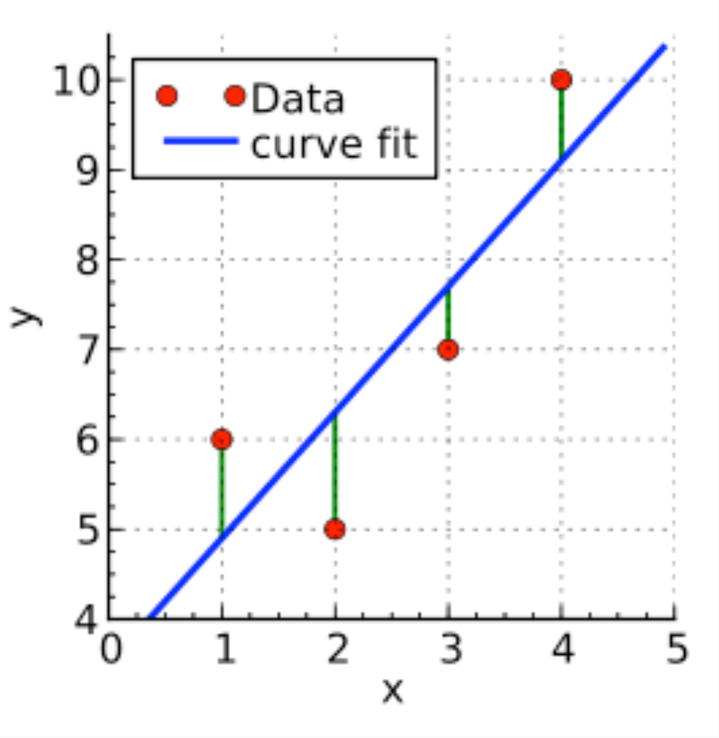
$$y^i = \mathbf{w}^T \mathbf{x}^i + \epsilon^i$$

Loss function: sum of squared errors

$$L(\mathbf{w}) = \sum_{i=1}^{N} (\epsilon^i)^2$$

In two variables:

$$L(w_0, w_1) = \sum_{i=1}^{N} \left[y^i - \left(w_0 x_0^i + w_1 x_1^i \right) \right]^2$$



Sum of Square Errors *(MSE without the mean)*

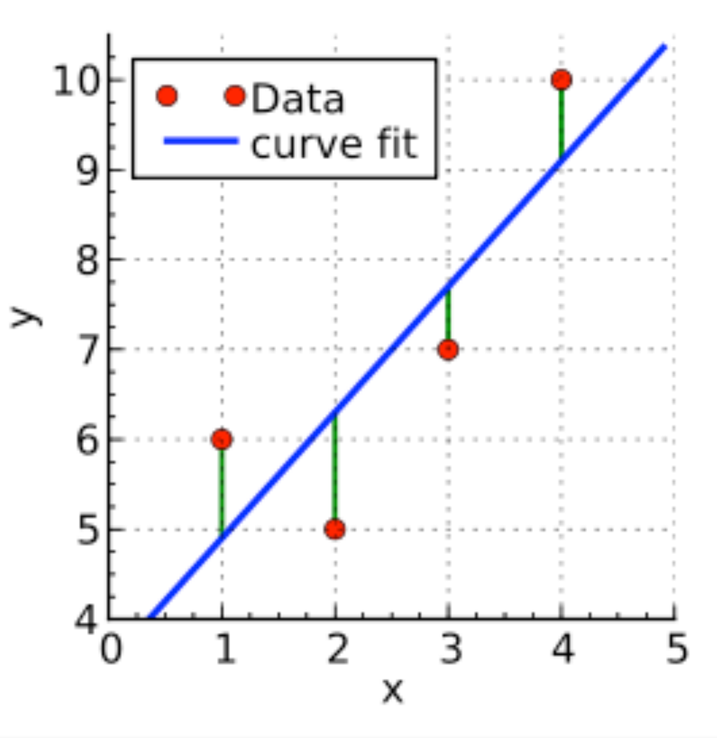
$$y^i = \mathbf{w}^T \mathbf{x}^i + \epsilon^i$$

Loss function: sum of squared errors

$$L(\mathbf{w}) = \sum_{i=1}^{N} (\epsilon^i)^2$$

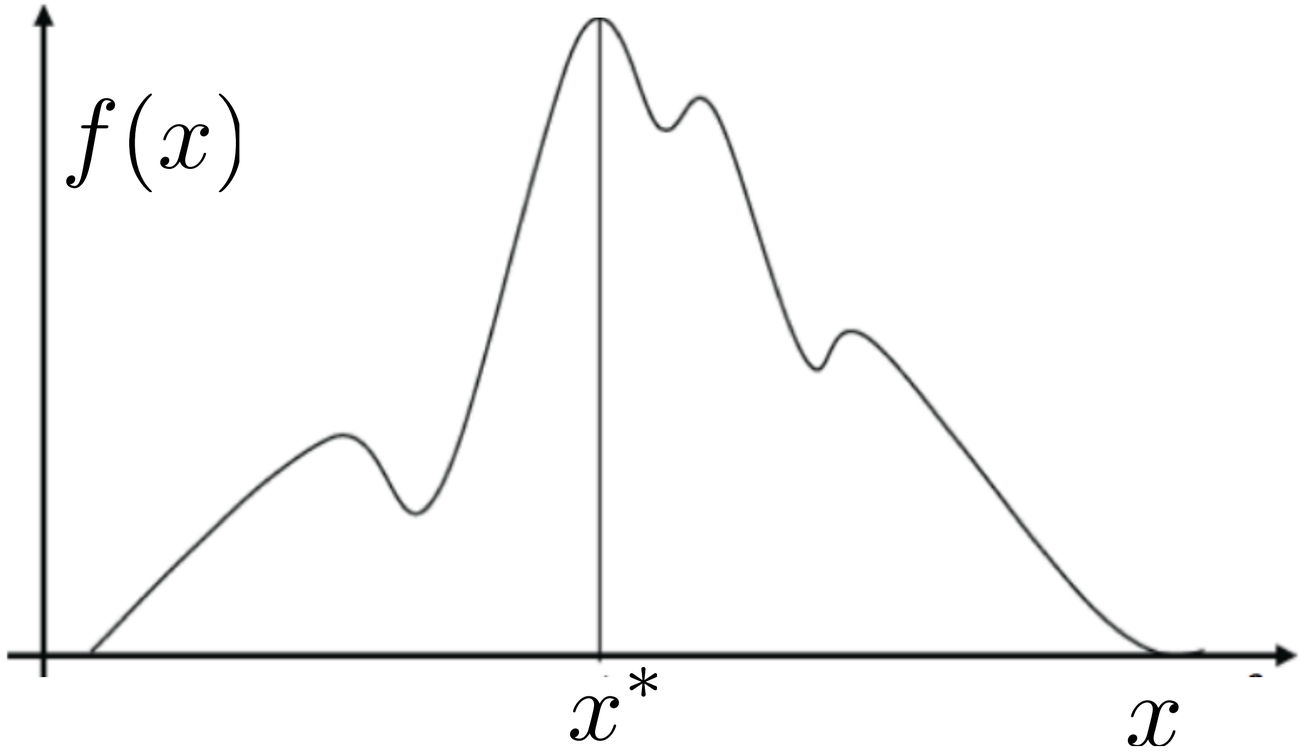
In two variables:

$$L(w_0, w_1) = \sum_{i=1}^{N} \left[y^i - \left(w_0 x_0^i + w_1 x_1^i \right) \right]^2$$

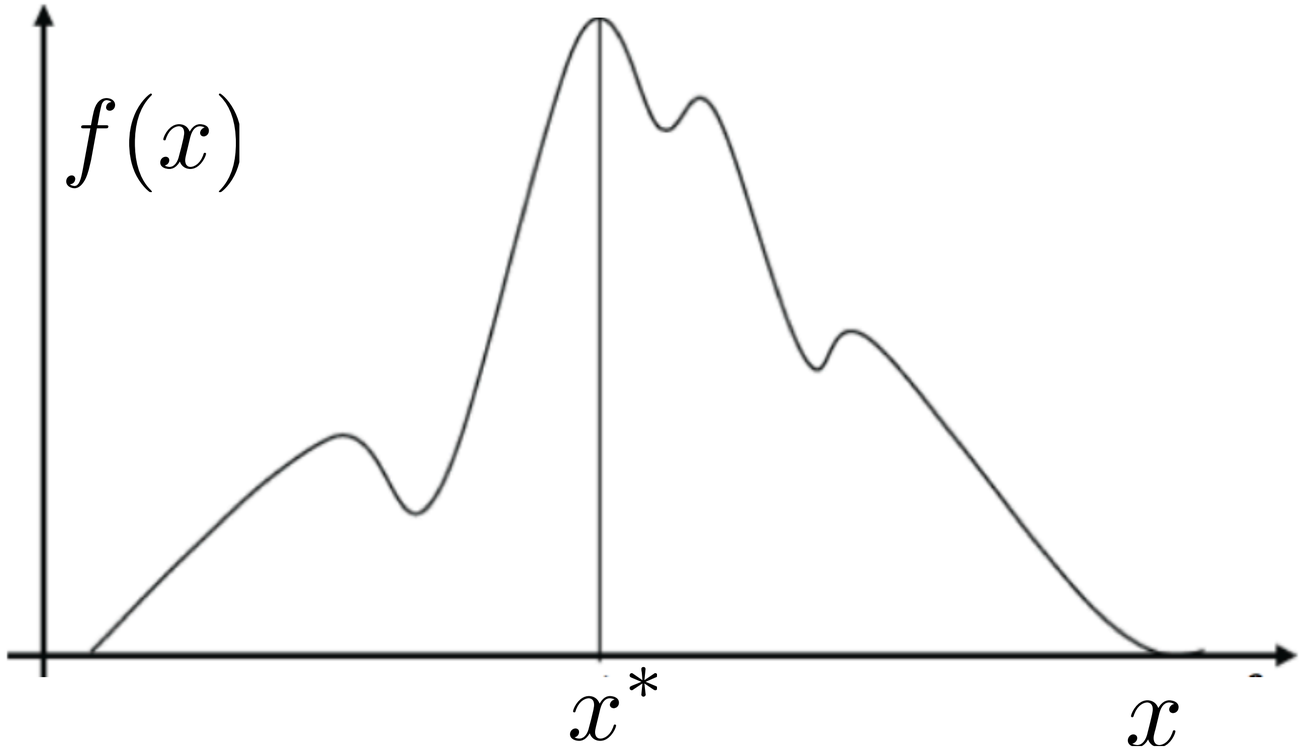


*Question: what is the best (or least bad) value of **w**?*

Calculus 101

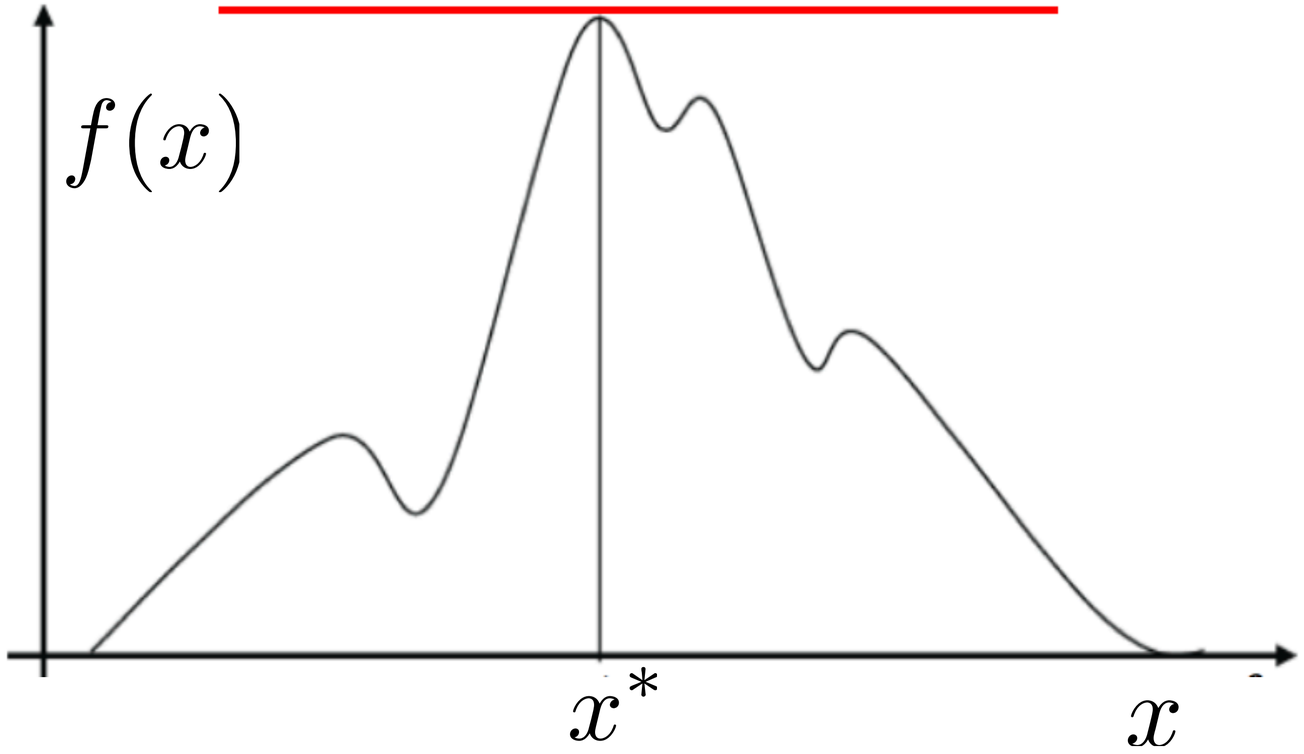


Calculus 101



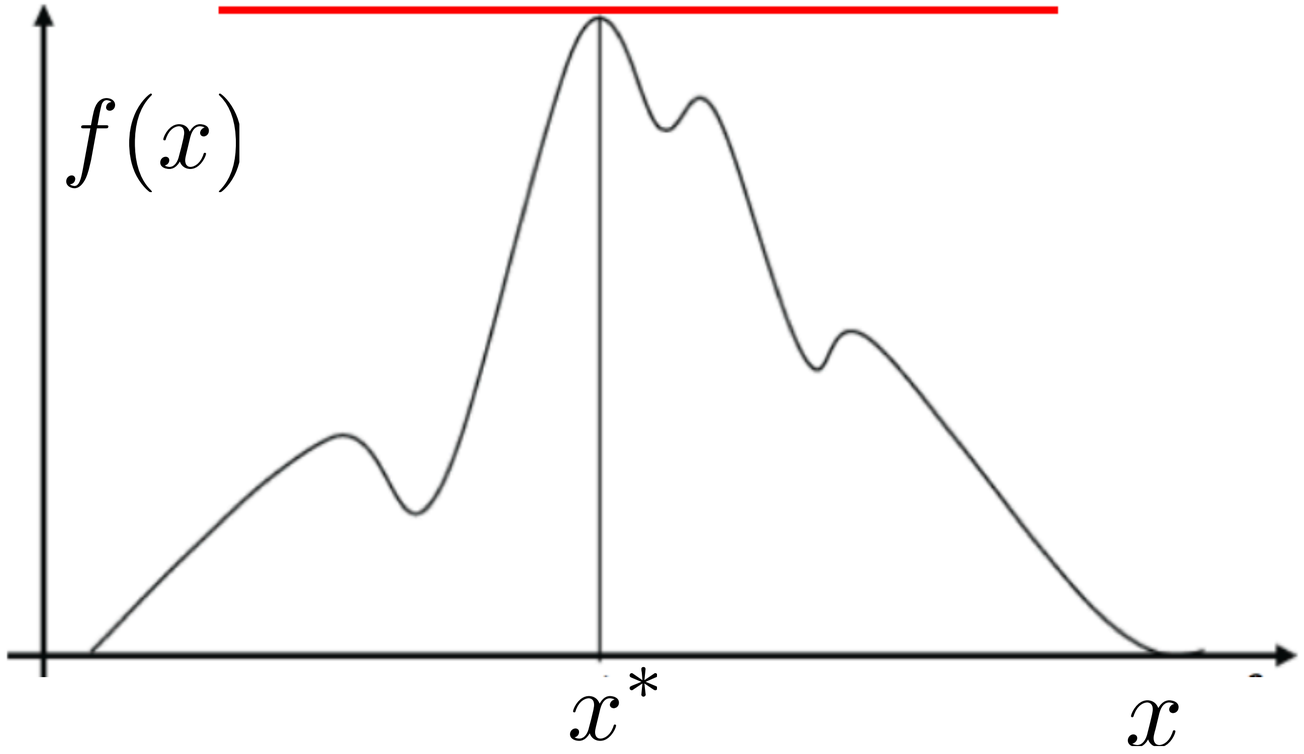
$$x^* = \text{argmax}_x f(x)$$

Local Extrema Condition



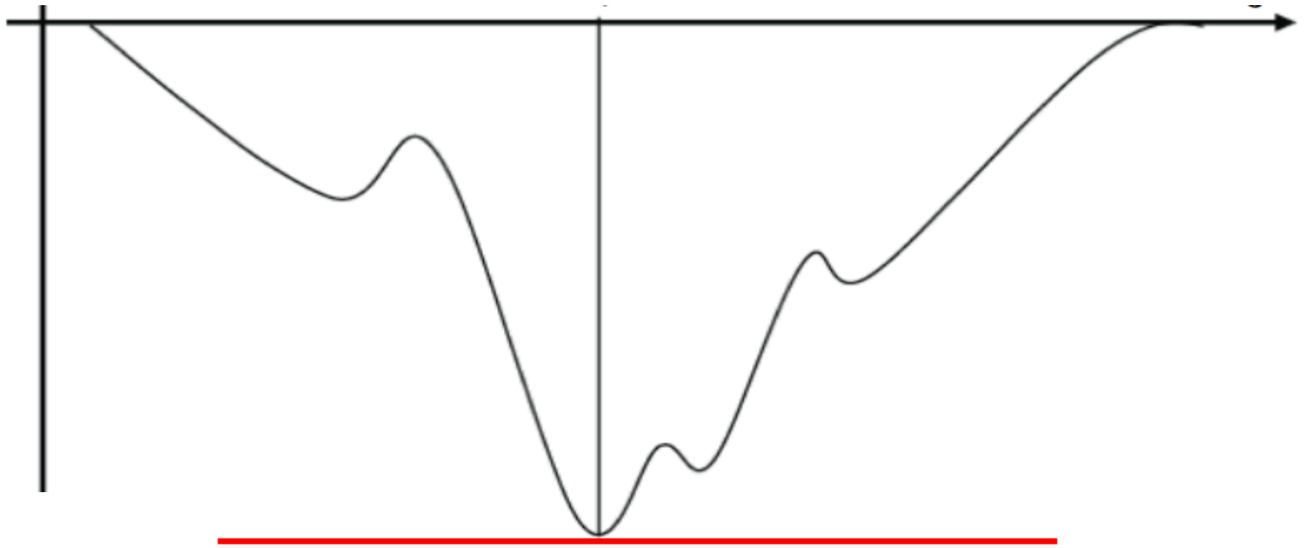
$$x^* = \text{argmax}_x f(x)$$

Local Extrema Condition



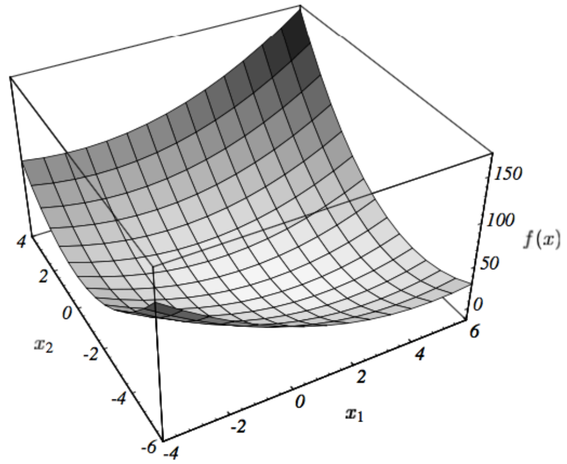
$$x^* = \text{argmax}_x f(x) \quad \rightarrow \quad f'(x^*) = 0$$

Local Extrema Condition



$$x^* = \text{argmax}_x f(x) \quad \rightarrow \quad f'(x^*) = 0$$

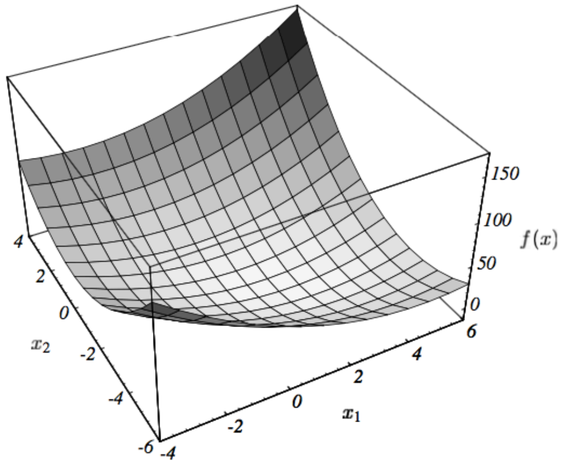
Vector Calculus 101

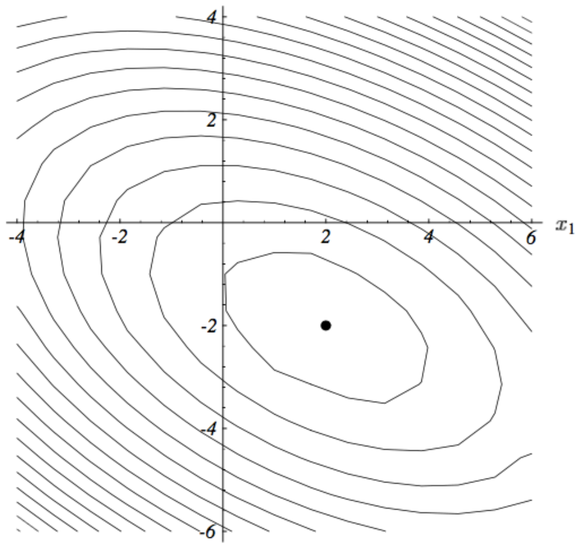


$$f(\mathbf{x})$$

2D function graph

Vector Calculus 101





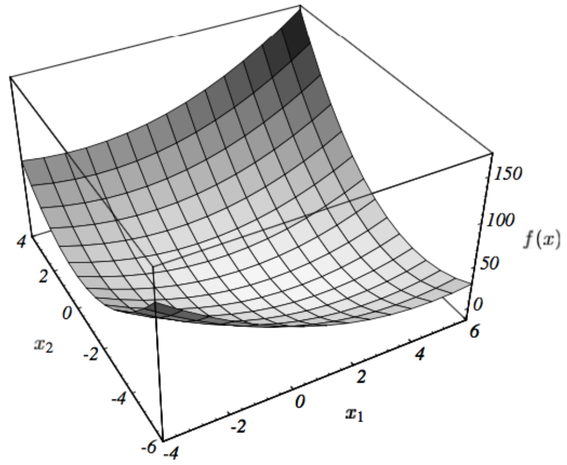
$$f(\mathbf{x})$$

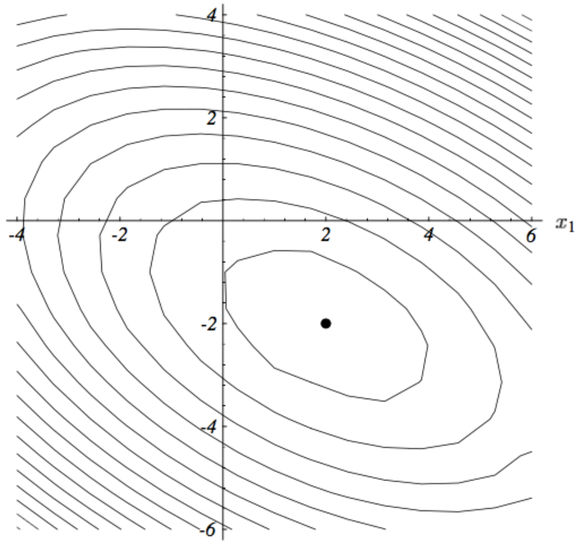
$$f(\mathbf{x}) = c$$

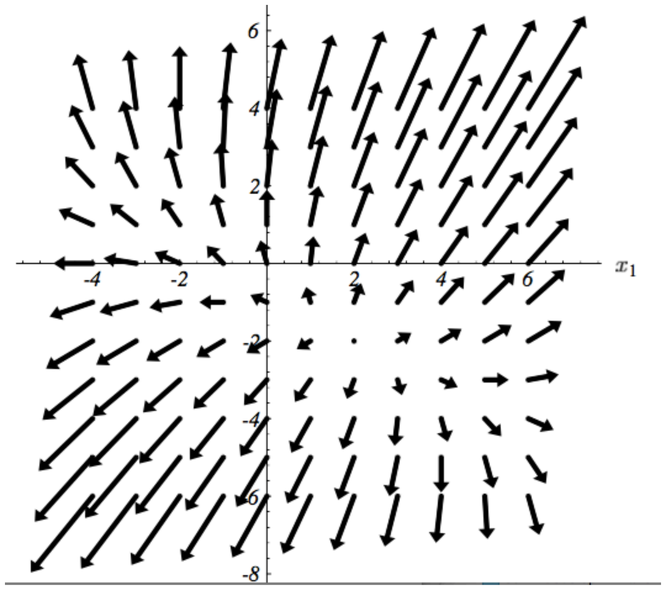
2D function graph

isocontours

Vector Calculus 101







$f(\mathbf{x})$

$f(\mathbf{x}) = c$

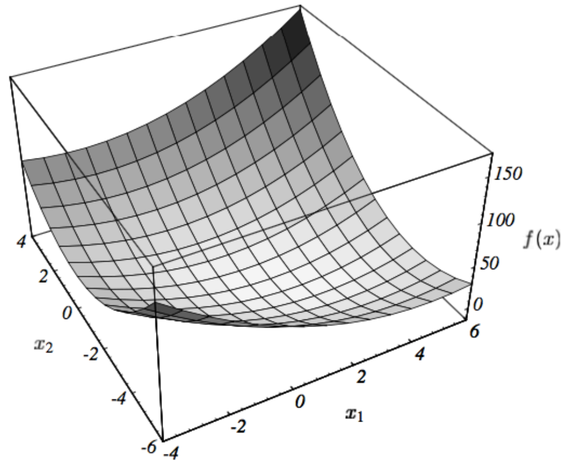
$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

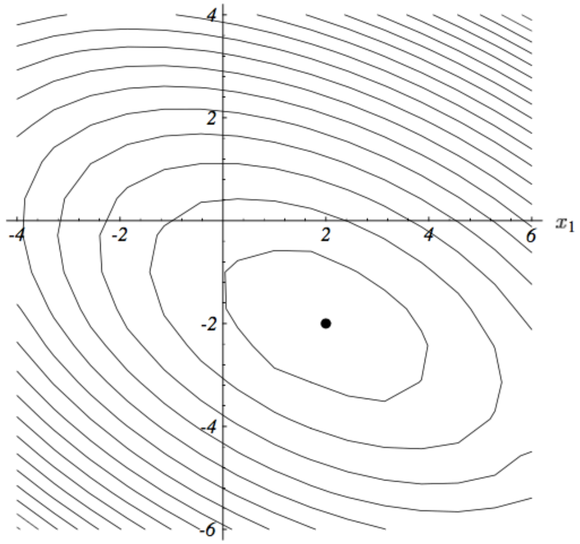
2D function graph

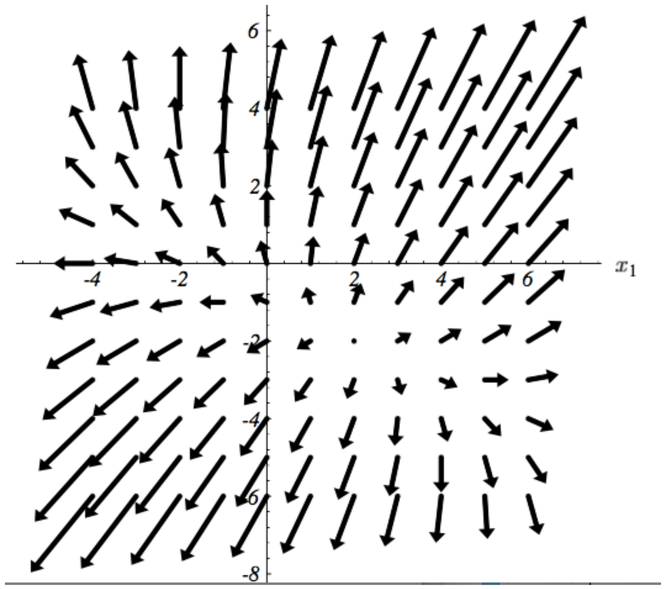
isocontours

gradient field

Vector Calculus 101







$$f(\mathbf{x})$$

$$f(\mathbf{x}) = c$$

$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

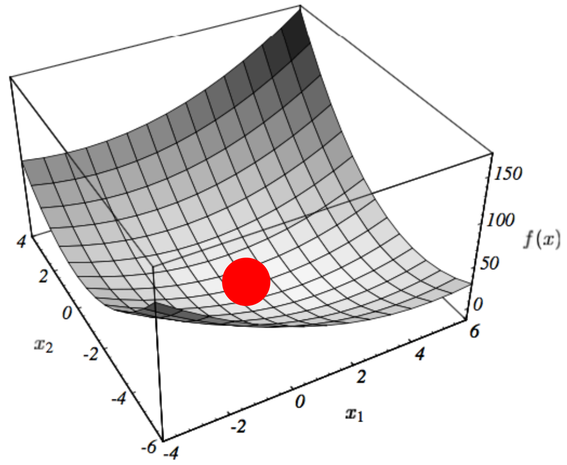
2D function graph

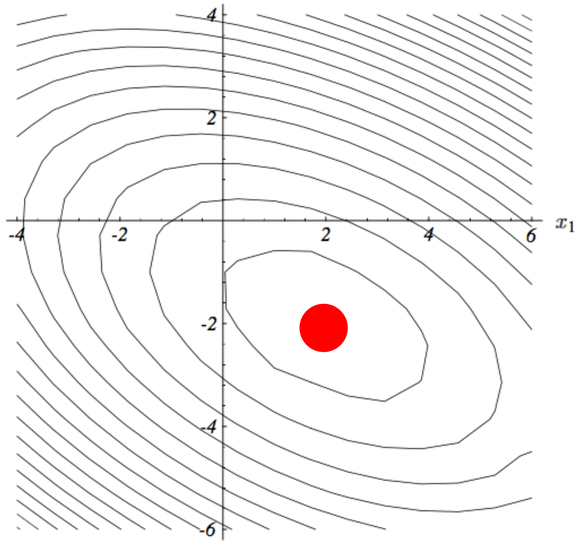
isocontours

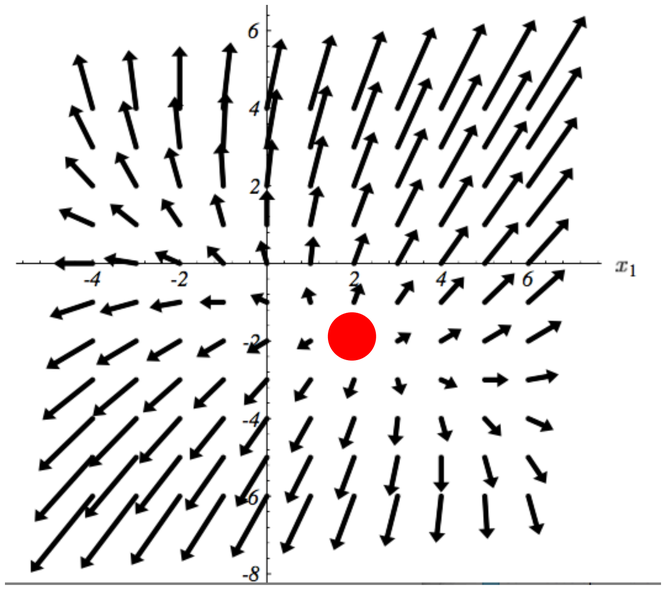
gradient field

- at minimum of function: $\nabla f(\mathbf{x}) = \mathbf{0}$

Vector Calculus 101







$f(\mathbf{x})$

$f(\mathbf{x}) = c$

$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

2D function graph

isocontours

gradient field

- at minimum of function: $\nabla f(\mathbf{x}) = \mathbf{0}$

LS Solution for Linear Regression

$$\mathbf{y} = \mathbf{X}\mathbf{w} + \boldsymbol{\epsilon}$$

$$L(\mathbf{w}) = \boldsymbol{\epsilon}^T \boldsymbol{\epsilon}$$

$$\mathbf{X} = \begin{bmatrix} (\mathbf{x}^1)^T \\ \hline (\mathbf{x}^2)^T \\ \hline \vdots \\ \hline (\mathbf{x}^N)^T \end{bmatrix}$$

LS Solution for Linear Regression

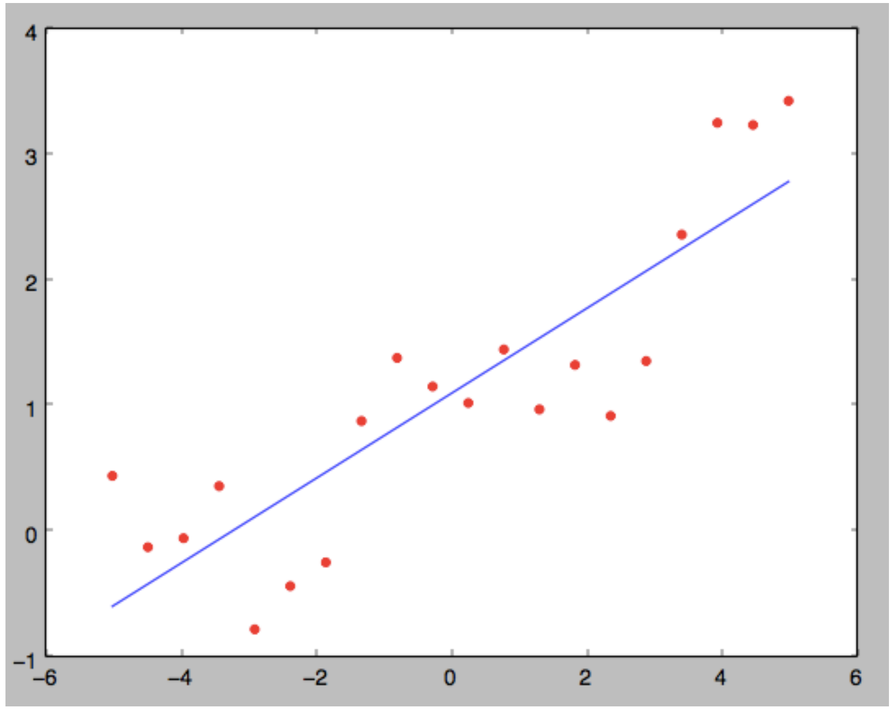
$$\mathbf{y} = \mathbf{X}\mathbf{w} + \boldsymbol{\epsilon}$$

$$L(\mathbf{w}) = \boldsymbol{\epsilon}^T \boldsymbol{\epsilon}$$

$$\mathbf{X} = \begin{bmatrix} \frac{(\mathbf{x}^1)^T}{(\mathbf{x}^2)^T} \\ \hline \vdots \\ \hline (\mathbf{x}^N)^T \end{bmatrix}$$

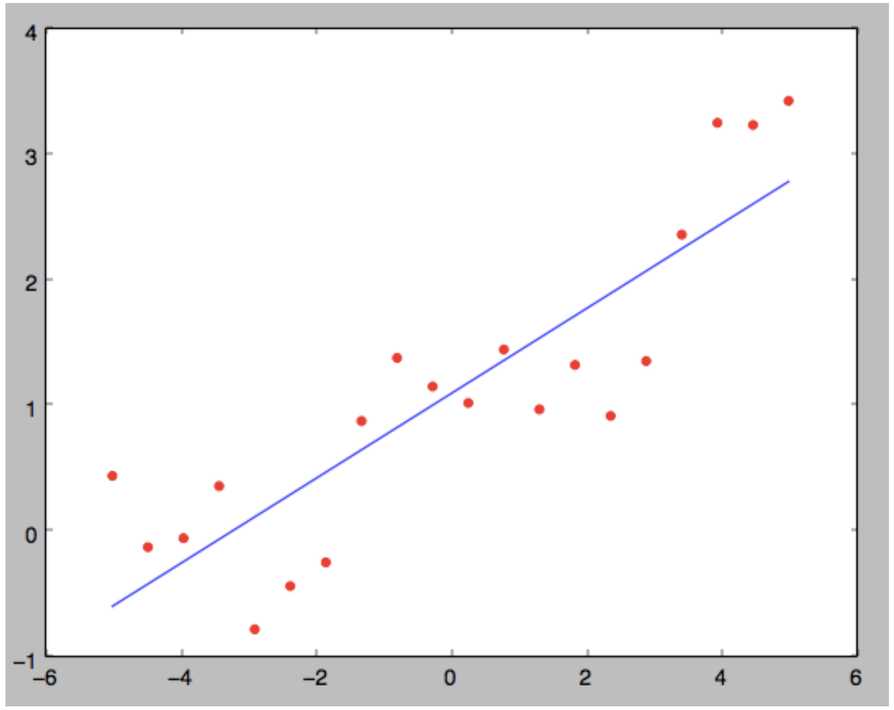
$$\mathbf{w}^* \leftarrow (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

Code Example



$$\mathbf{w}^* \leftarrow (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

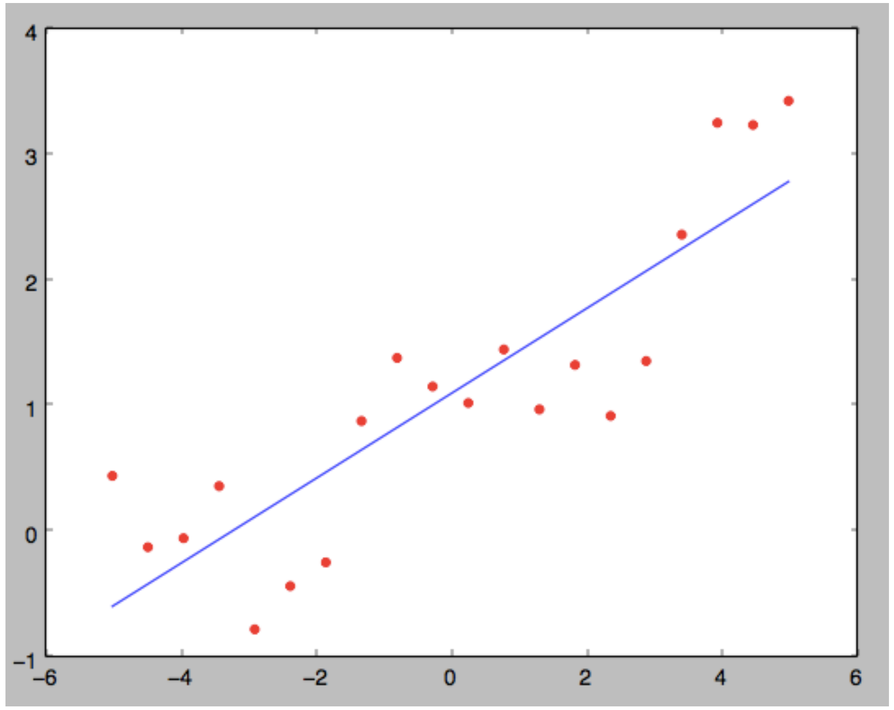
Code Example




```python
import numpy as np
from numpy import array
from numpy import matmul
from numpy.linalg import inv
from numpy.random import rand
from matplotlib import pyplot

# generate data on a line perturbed with some noise
noise_margin= 2
w = rand(2,1) # w[0] is random constant term (offset from origin), w[1] is random linear term (slope)
x  = np.linspace(-5,5,20)
y = w[0] + w[1]*x + noise_margin*rand(len(x))

# create the design matrix: the x data, and add a column of ones for the constant term
X = np.column_stack( [np.ones([len(x), 1]), x.reshape(-1, 1)] )

# These are the normal equations in matrix form: w = (X' X)^-1 X' y
w_est = matmul(inv(matmul(X.transpose(),X)),X.transpose()).dot(y)

# For ridge regression, use regularizer
#weight = 0.01
#w_est = matmul(inv(matmul(X.transpose(),X) + weight*np.identity(2)),X.transpose()).dot(y)


# evaluate the x values in the fitted model to get estimated y values
y_est = w_est[0] + w_est[1]*x

# visualize the fitted model
pyplot.scatter(x, y, color='red')
pyplot.plot(x, y_est, color='blue')
pyplot.show()
```

$$\mathbf{w}^* \leftarrow (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

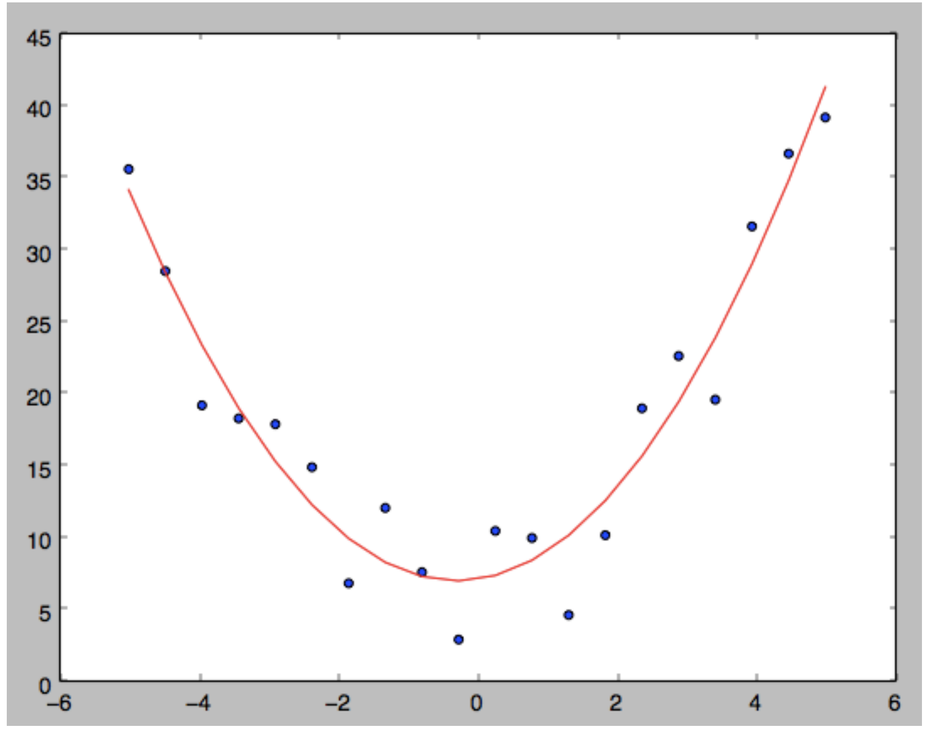
Code Example




```python
import numpy as np
from numpy import array
from numpy import matmul
from numpy.linalg import inv
from numpy.random import rand
from matplotlib import pyplot

# generate data on a line perturbed with some noise
noise_margin= 2
w = rand(2,1) # w[0] is random constant term (offset from origin), w[1] is random linear term (slope)
x  = np.linspace(-5,5,20)
y = w[0] + w[1]*x + noise_margin*rand(len(x))

# create the design matrix: the x data, and add a column of ones for the constant term
X = np.column_stack( [np.ones([len(x), 1]), x.reshape(-1, 1)] )

# These are the normal equations in matrix form: w = (X' X)^-1 X' y
w_est = matmul(inv(matmul(X.transpose(),X)),X.transpose()).dot(y)

# For ridge regression, use regularizer
#weight = 0.01
#w_est = matmul(inv(matmul(X.transpose(),X) + weight*np.identity(2)),X.transpose()).dot(y)


# evaluate the x values in the fitted model to get estimated y values
y_est = w_est[0] + w_est[1]*x

# visualize the fitted model
pyplot.scatter(x, y, color='red')
pyplot.plot(x, y_est, color='blue')
pyplot.show()
```

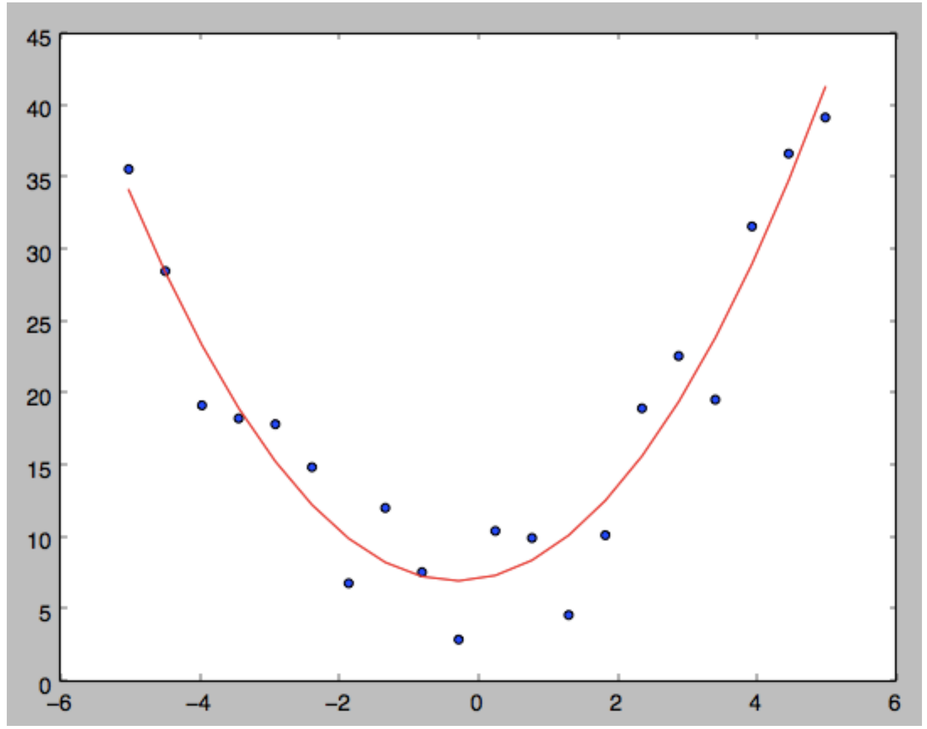
$$\mathbf{w}^* \leftarrow (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

Code Example




```
import numpy as np
from numpy import array
from numpy import matmul
from numpy.linalg import inv
from numpy.random import rand
from matplotlib import pyplot

# generate data on a line perturbed with some noise
noise_margin= 3
w = 2*rand(3,1) # w[0] is random constant term (offset from origin), w[1] is random linear term, w[2] is random quadratic term
x  = np.linspace(-5,5,20)
y = w[0] + w[1]*x + w[2]*x**2 + noise_margin*rand(len(x))

# create the design matrix: the x data, and add a column of ones for the constant term
X = np.column_stack( [np.ones([len(x), 1]), x.reshape(-1, 1), (x**2).reshape(-1, 1)] )

# These are the normal equations in matrix form: w = (X' X)^-1 X' y
w_est = matmul(inv(matmul(X.transpose(),X)),X.transpose()).dot(y)


# evaluate the x values in the fitted model to get estimated y values
y_est = w_est[0] + w_est[1]*x + w_est[2]*x**2

# visualize the fitted model
pyplot.scatter(x, y)
pyplot.plot(x, y_est, color='red')
pyplot.show()
```

Code Example



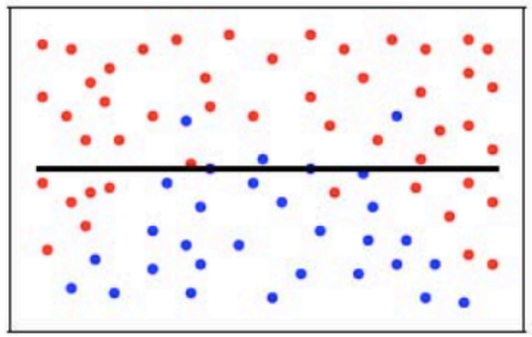

```
import numpy as np
from numpy import array
from numpy import matmul
from numpy.linalg import inv
from numpy.random import rand
from matplotlib import pyplot

# generate data on a line perturbed with some noise
noise_margin= 3
w = 2*rand(3,1) # w[0] is random constant term (offset from origin), w[1] is random linear term, w[2] is random quadratic term
x  = np.linspace(-5,5,20)
y = w[0] + w[1]*x + w[2]*x**2 + noise_margin*rand(len(x))

# create the design matrix: the x data, and add a column of ones for the constant term
X = np.column_stack( [np.ones([len(x), 1]), x.reshape(-1, 1), (x**2).reshape(-1, 1)] )

# These are the normal equations in matrix form: w = (X' X)^-1 X' y
w_est = matmul(inv(matmul(X.transpose(),X)),X.transpose()).dot(y)


# evaluate the x values in the fitted model to get estimated y values
y_est = w_est[0] + w_est[1]*x + w_est[2]*x**2

# visualize the fitted model
pyplot.scatter(x, y)
pyplot.plot(x, y_est, color='red')
pyplot.show()
```

$$\mathbf{w}^* = (\mathbf{\Phi}^T\mathbf{\Phi})^{-1}\mathbf{\Phi}\mathbf{y}$$

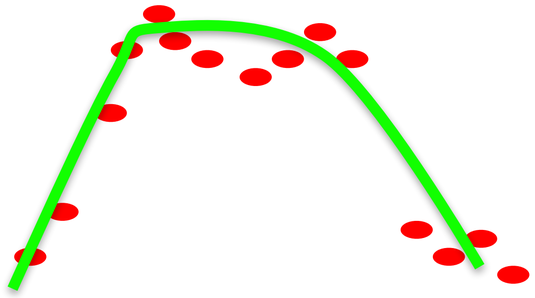
Hyperparameter: Underfitting vs. Overfitting

classification

Underfitting



regression

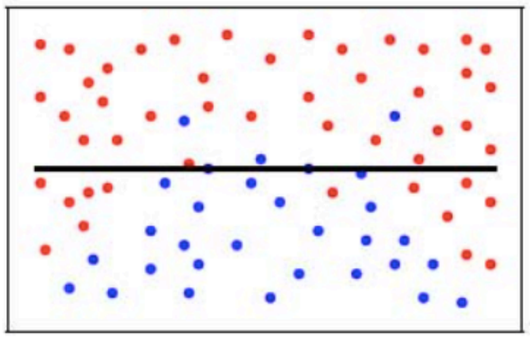


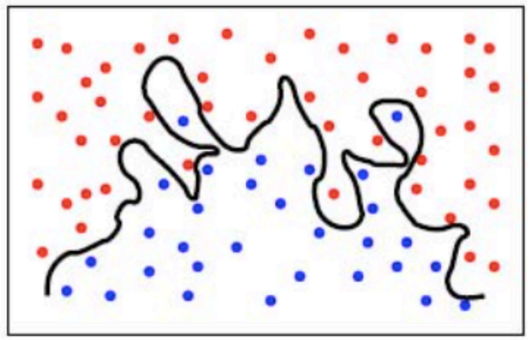
Hyperparameter: Underfitting vs. Overfitting

Underfitting

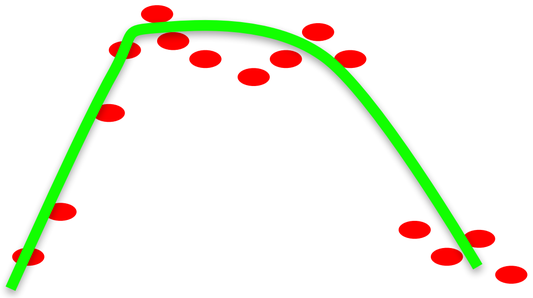
Overfitting

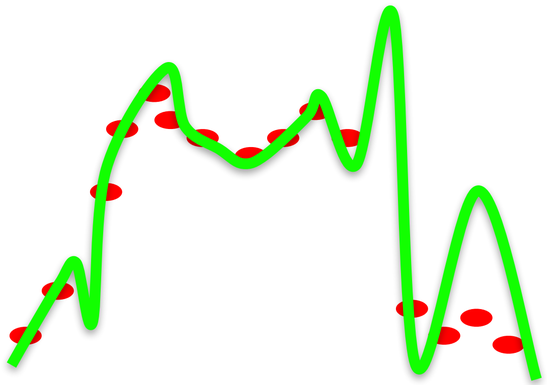
classification



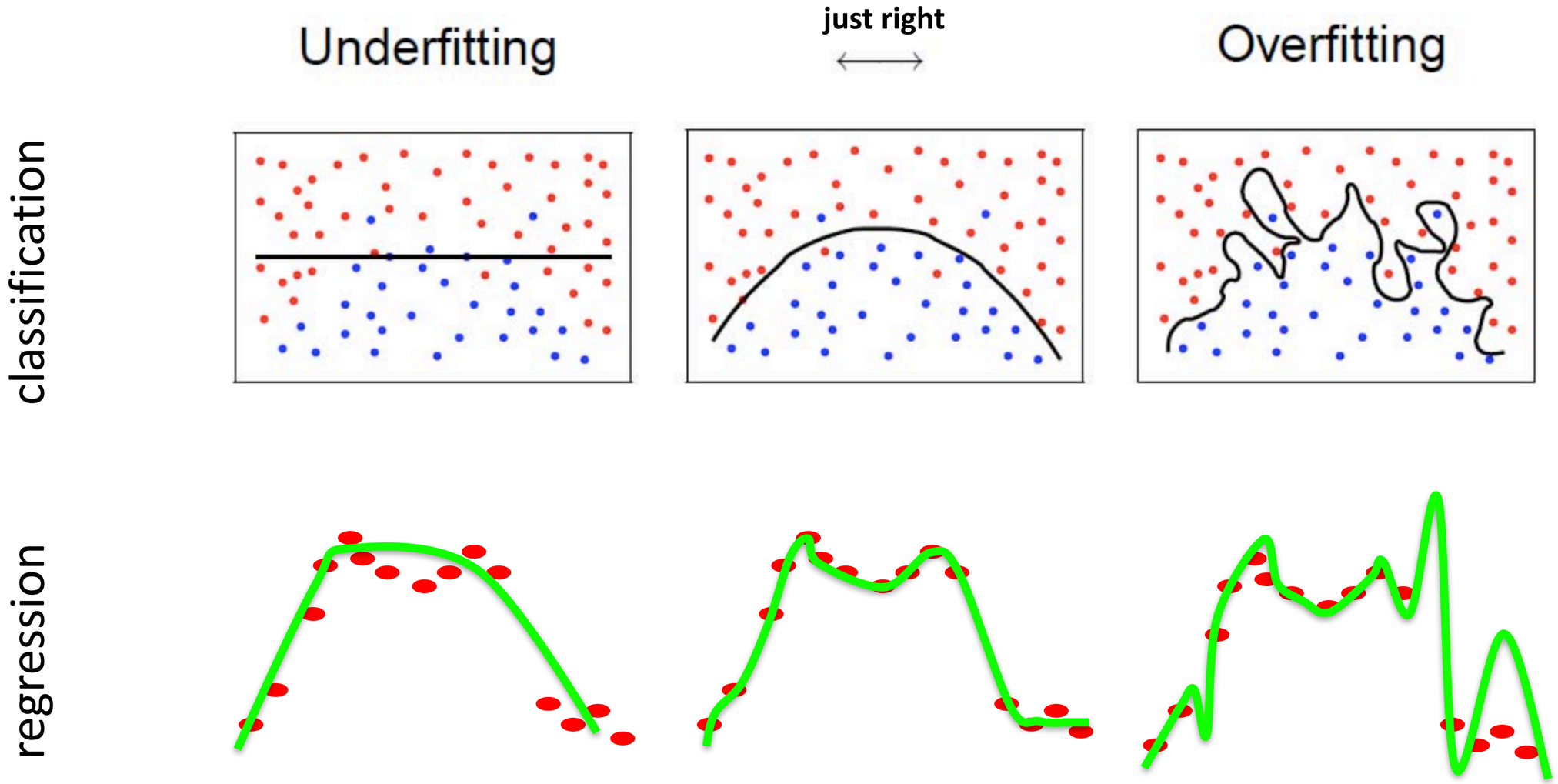


regression



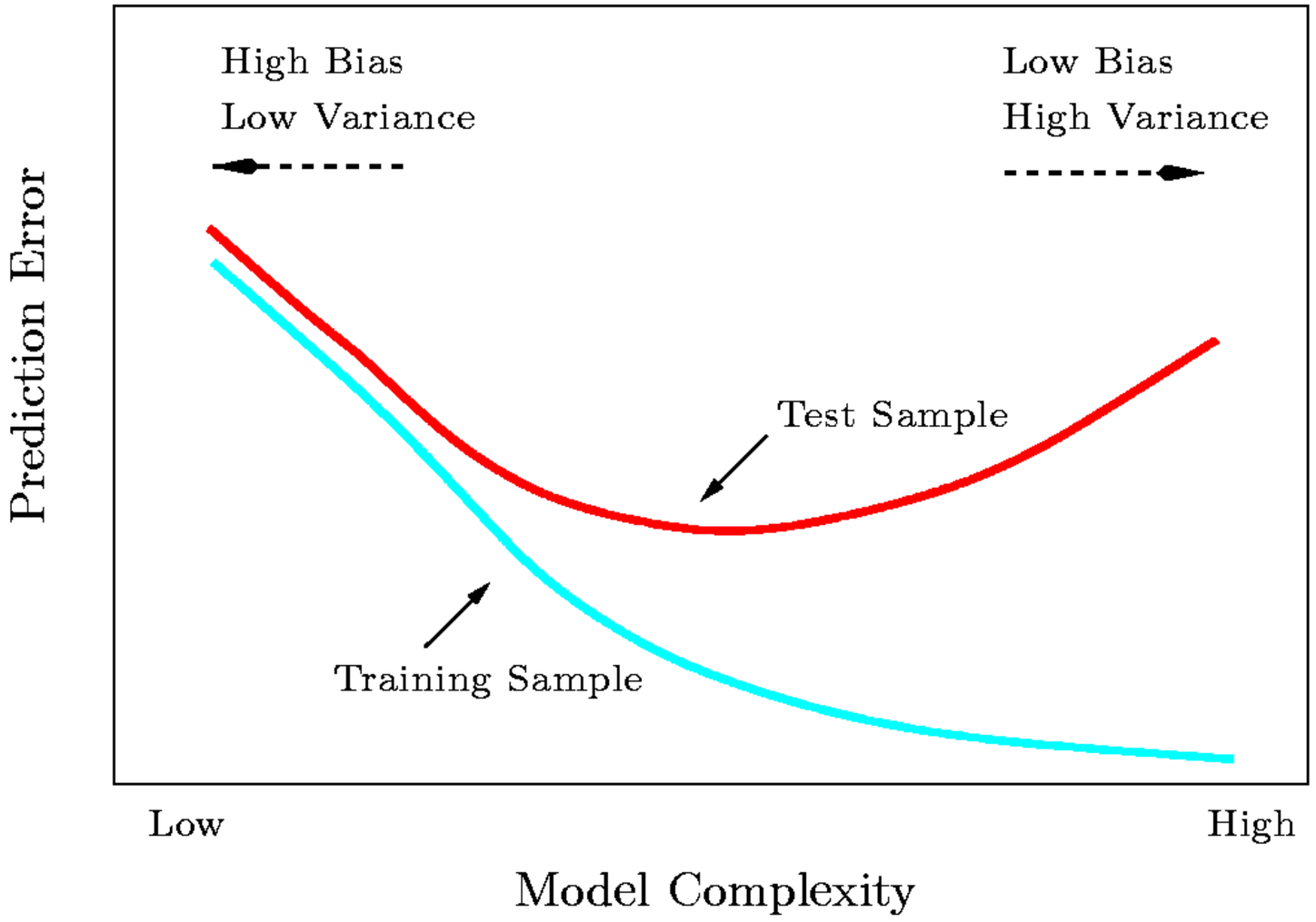


Hyperparameter: Underfitting vs. Overfitting

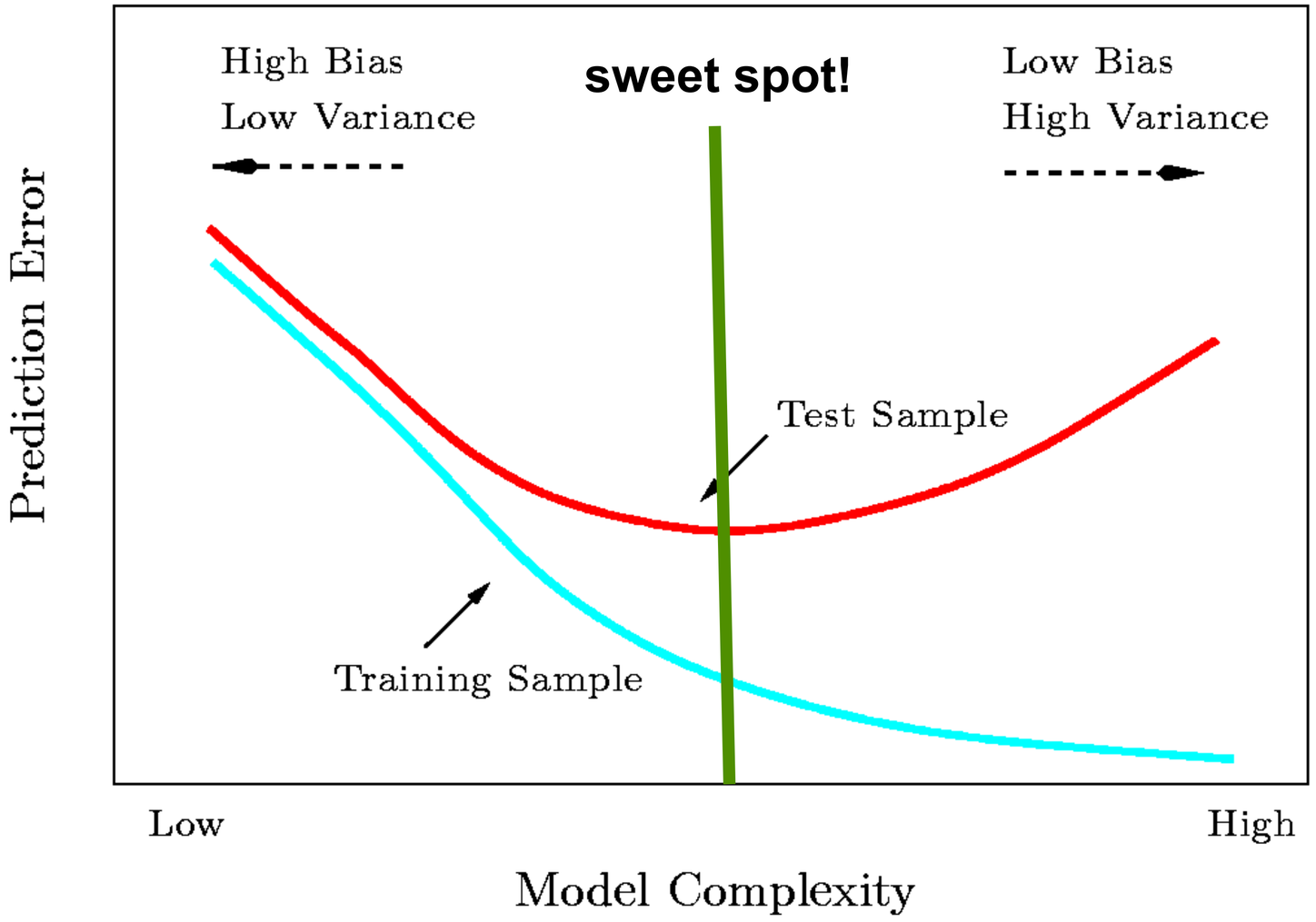


Hyperparameter Tuning

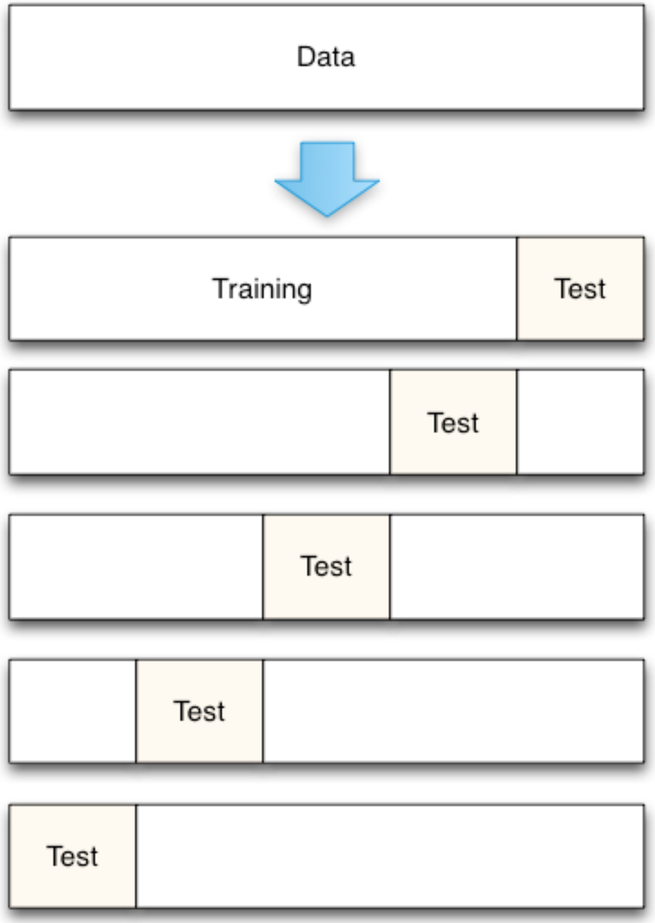
Hyperparameter Tuning



Hyperparameter Tuning

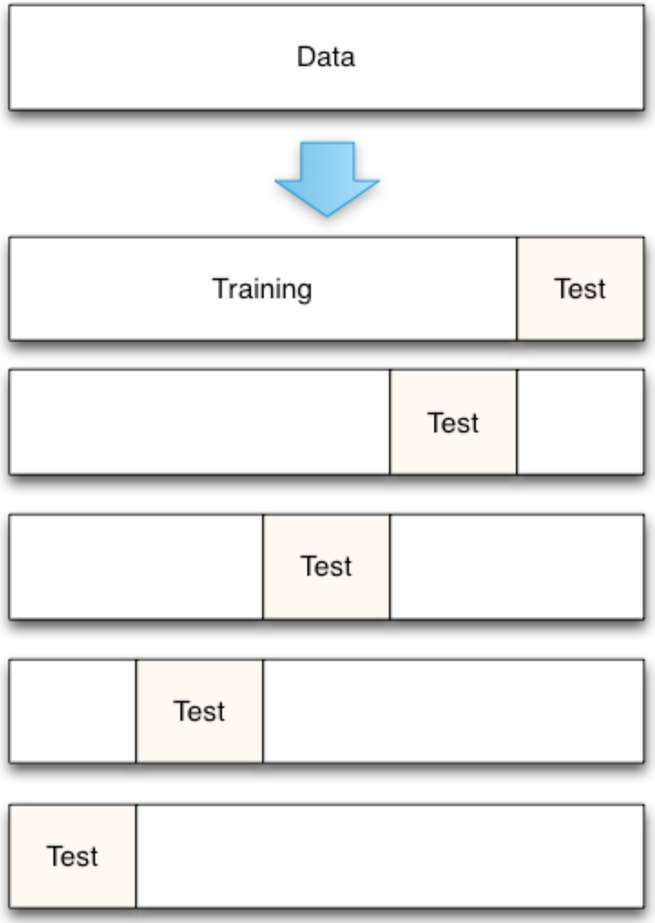


Selecting λ with Cross-validation



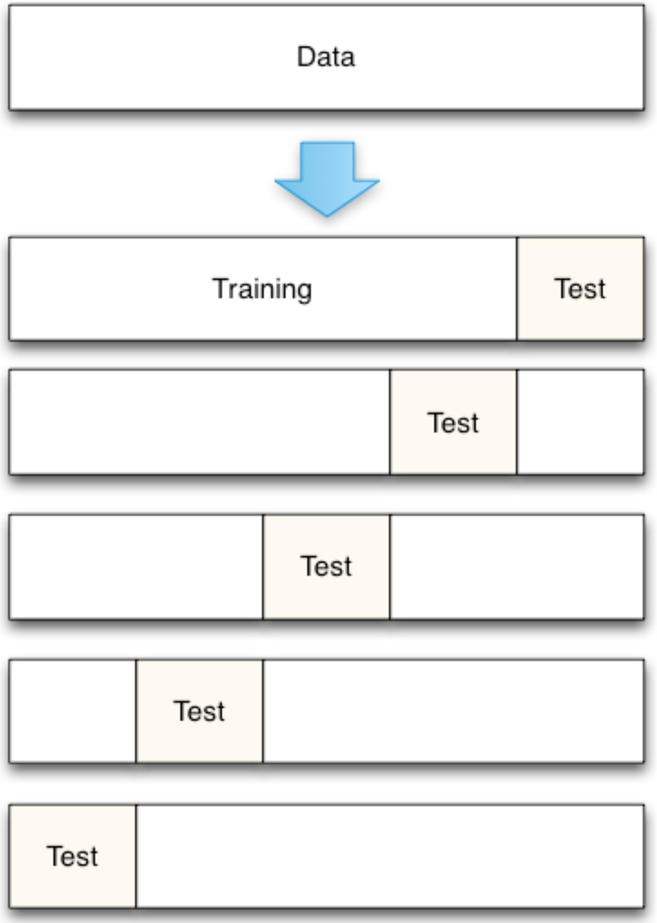
Selecting λ with Cross-validation

- Cross validation technique
  - Exclude part of the training data from parameter estimation
  - Use them only to predict the test error



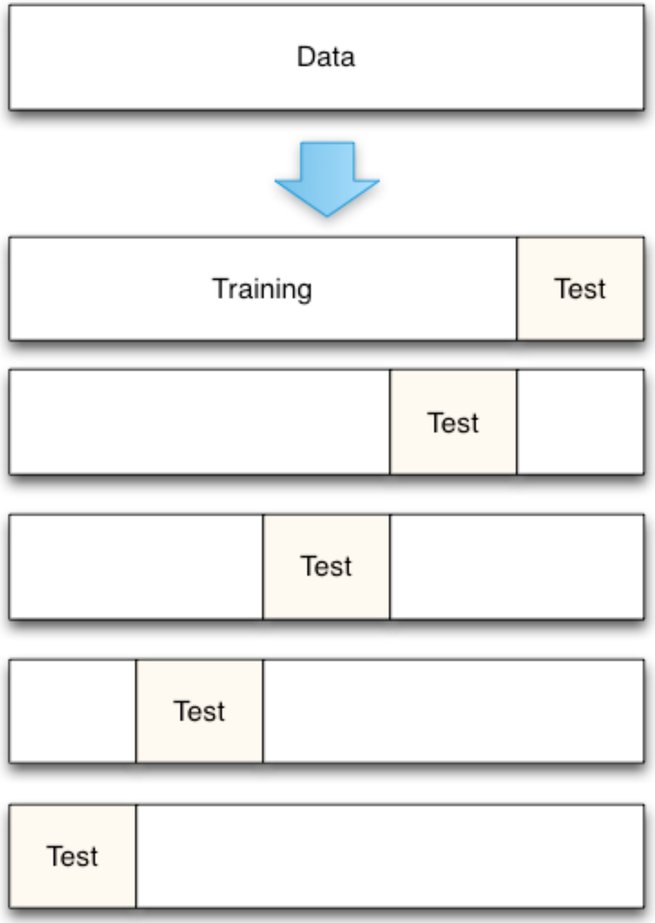
Selecting λ with Cross-validation

- Cross validation technique
  - Exclude part of the training data from parameter estimation
  - Use them only to predict the test error
- K-fold cross validation:
  - K splits, average K errors



Selecting λ with Cross-validation

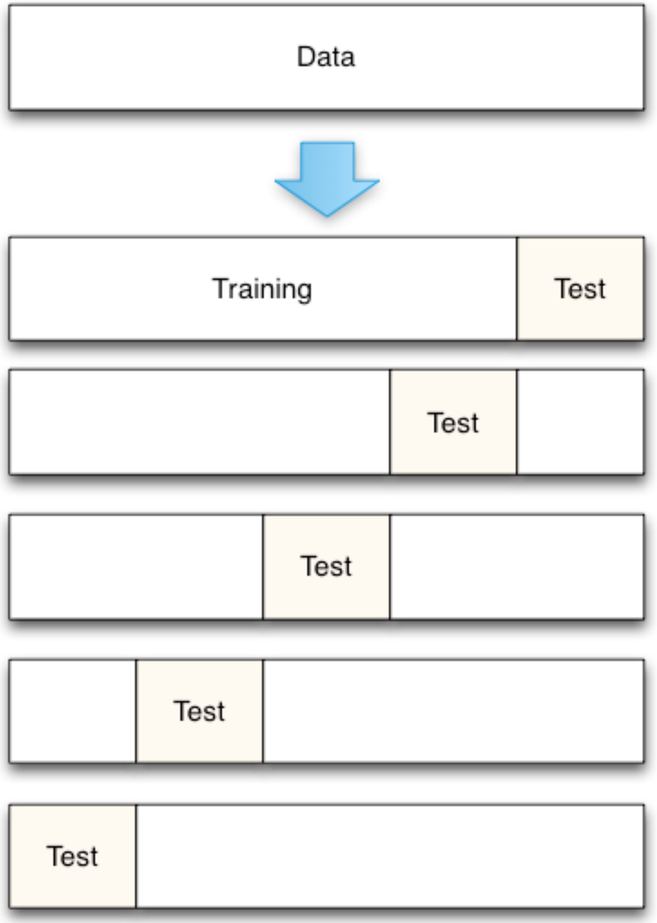
- Cross validation technique
  - Exclude part of the training data from parameter estimation
  - Use them only to predict the test error
- K-fold cross validation:
  - K splits, average K errors
- Use cross-validation for different values of λ
  - pick value that minimizes cross-validation error



Selecting λ with Cross-validation

- Cross validation technique
  - Exclude part of the training data from parameter estimation
  - Use them only to predict the test error
- K-fold cross validation:
  - K splits, average K errors
- Use cross-validation for different values of λ
  - pick value that minimizes cross-validation error

Least glorious, most effective of all methods.



Regression

1. Least Squares fitting

2. **Nonlinear error function and gradient descent**

3. Perceptron training (simple neural network)

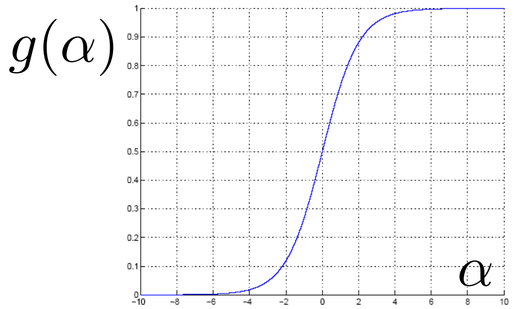
Extension #1: Logistic Regression

Using squashing (sigmoidal) function for robustness

Extension #1: Logistic Regression

Using squashing (sigmoidal) function for robustness

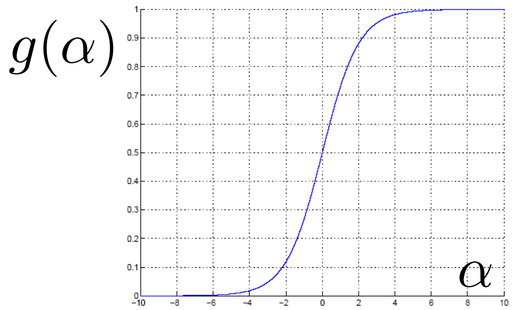
$$g(\alpha) = \frac{1}{1 + \exp(-\alpha)}$$

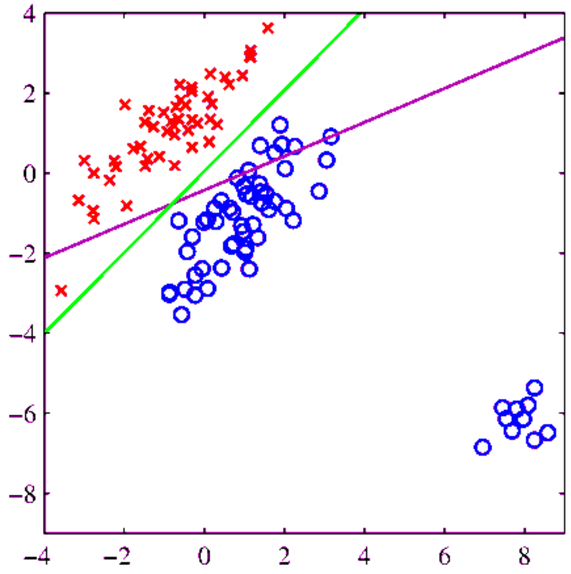


Extension #1: Logistic Regression

Using squashing (sigmoidal) function for robustness

$$g(\alpha) = \frac{1}{1 + \exp(-\alpha)}$$

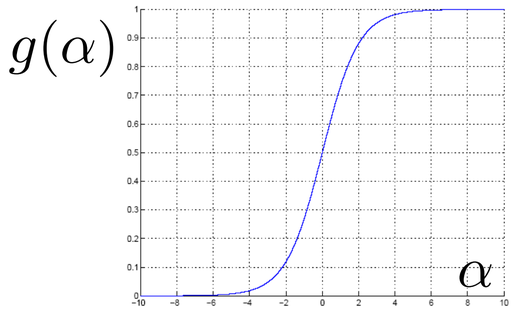


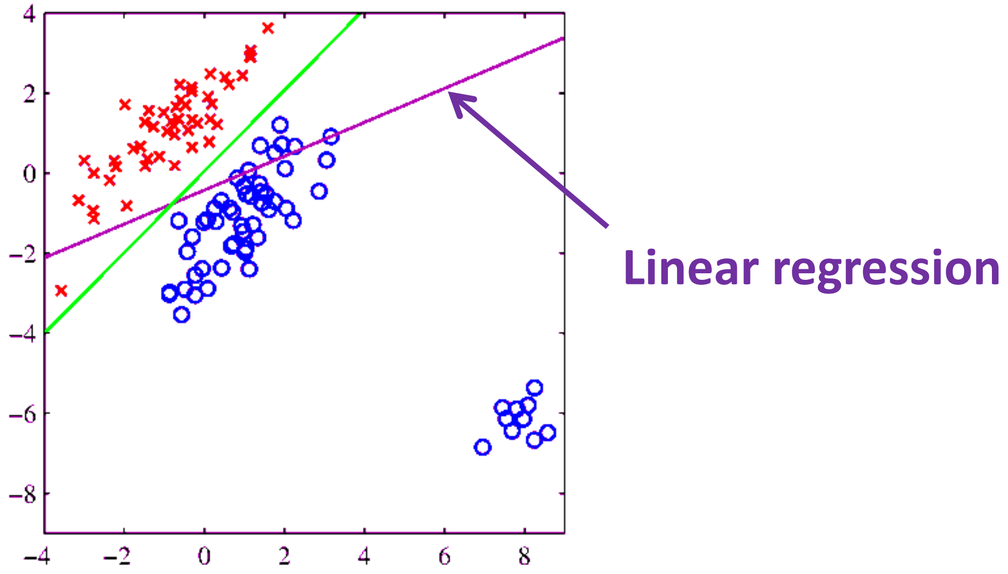


Extension #1: Logistic Regression

Using squashing (sigmoidal) function for robustness

$$g(\alpha) = \frac{1}{1 + \exp(-\alpha)}$$

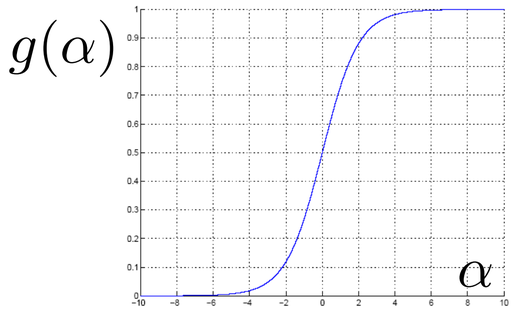


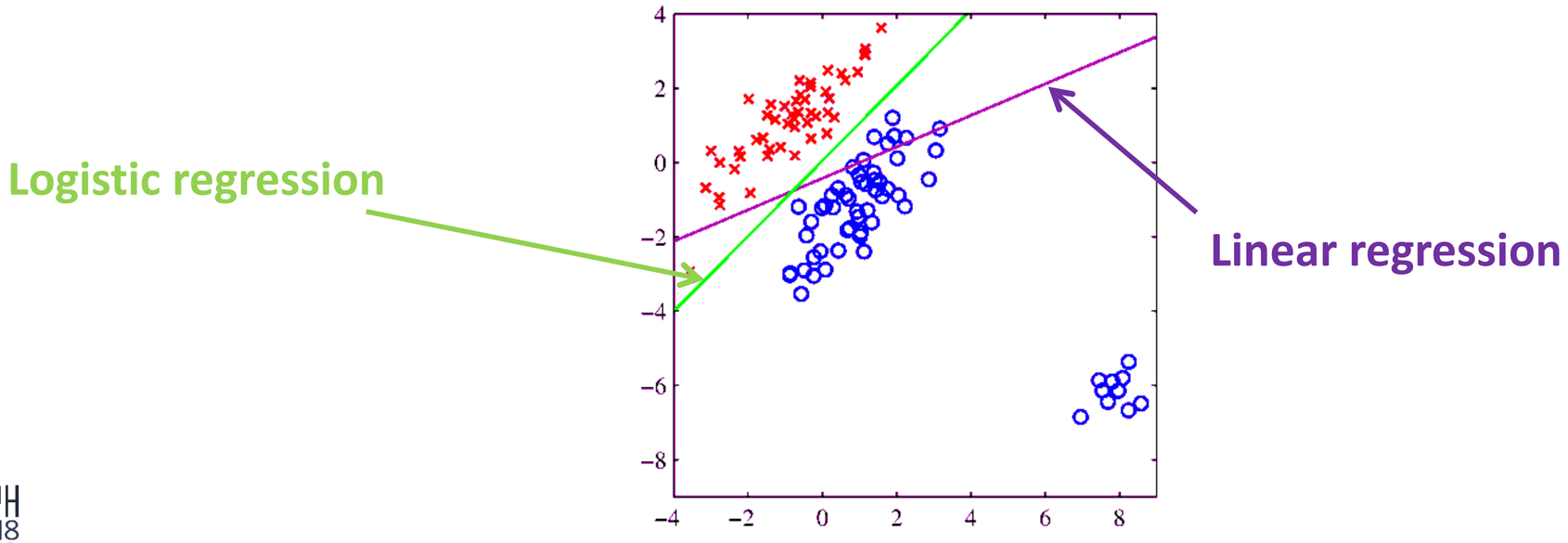


Extension #1: Logistic Regression

Using squashing (sigmoidal) function for robustness

$$g(\alpha) = \frac{1}{1 + \exp(-\alpha)}$$





Extension #2: Handling Multiple (2+) Classes

Extension #2: Handling Multiple (2+) Classes

C classes: one-of-c coding (or **one-hot encoding**)

4 classes, i-th sample is in 3rd class:

$$\mathbf{y}^i = (0, 0, 1, 0)$$

Extension #2: Handling Multiple (2+) Classes

C classes: one-of-c coding (or **one-hot encoding**)

4 classes, i-th sample is in 3rd class:
$\mathbf{y}^i = (0, 0, 1, 0)$

Matrix notation:

$$\mathbf{Y} = \begin{bmatrix} \mathbf{y}^1 \\ \hline \vdots \\ \hline \mathbf{y}^N \end{bmatrix} = \left[\begin{array}{c|c|c} \mathbf{y}_1 & \dots & \mathbf{y}_C \end{array} \right] \quad \text{where} \quad \mathbf{y}_c = \begin{bmatrix} y_c^1 \\ \vdots \\ y_c^N \end{bmatrix}$$

$$\mathbf{W} = \left[\begin{array}{c|c|c} \mathbf{w}_1 & \dots & \mathbf{w}_C \end{array} \right]$$

Loss function:

$$L(\mathbf{W}) = \sum_{c=1}^{C} (\mathbf{y}_c - \mathbf{X}\mathbf{w}_c)^T (\mathbf{y}_c - \mathbf{X}\mathbf{w}_c)$$

Extension #2: Handling Multiple (2+) Classes

C classes: one-of-c coding (or **one-hot encoding**)

4 classes, i-th sample is in 3rd class:
$\mathbf{y}^i = (0, 0, 1, 0)$

Matrix notation:

$$\mathbf{Y} = \begin{bmatrix} \mathbf{y}^1 \\ \hline \vdots \\ \hline \mathbf{y}^N \end{bmatrix} = \left[\begin{array}{c|c|c} \mathbf{y}_1 & \dots & \mathbf{y}_C \end{array} \right] \quad \text{where} \quad \mathbf{y}_c = \begin{bmatrix} y_c^1 \\ \vdots \\ y_c^N \end{bmatrix}$$

$$\mathbf{W} = \left[\begin{array}{c|c|c} \mathbf{w}_1 & \dots & \mathbf{w}_C \end{array} \right]$$

Loss function:

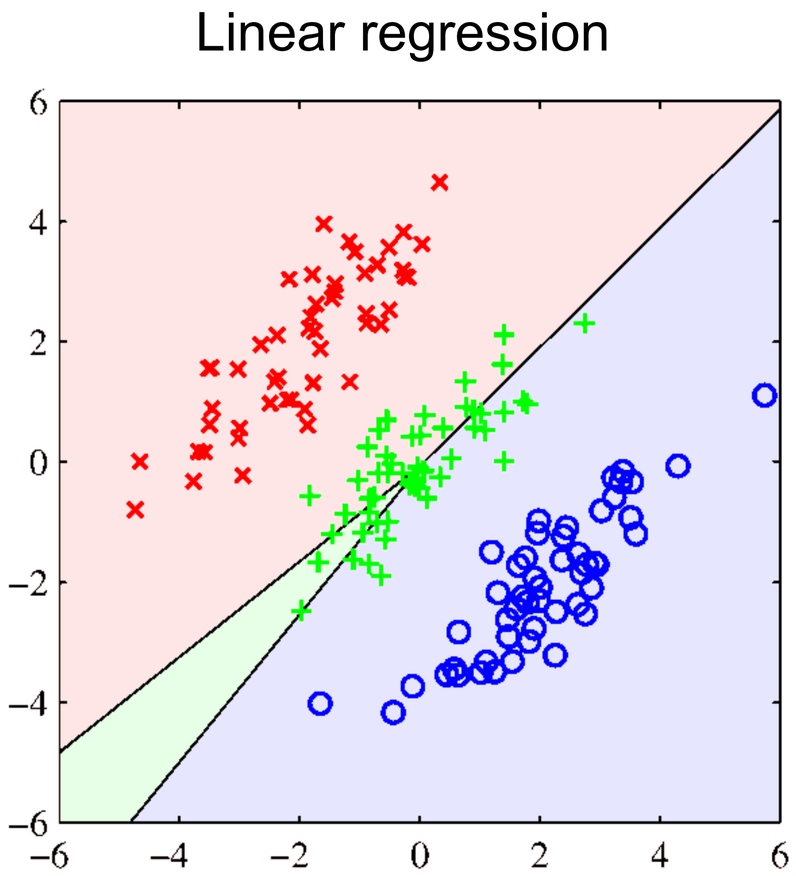
$$L(\mathbf{W}) = \sum_{c=1}^{C} (\mathbf{y}_c - \mathbf{X}\mathbf{w}_c)^T (\mathbf{y}_c - \mathbf{X}\mathbf{w}_c)$$

Least squares fit (decouples per class):

$$\mathbf{w}_c^* = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}_c$$

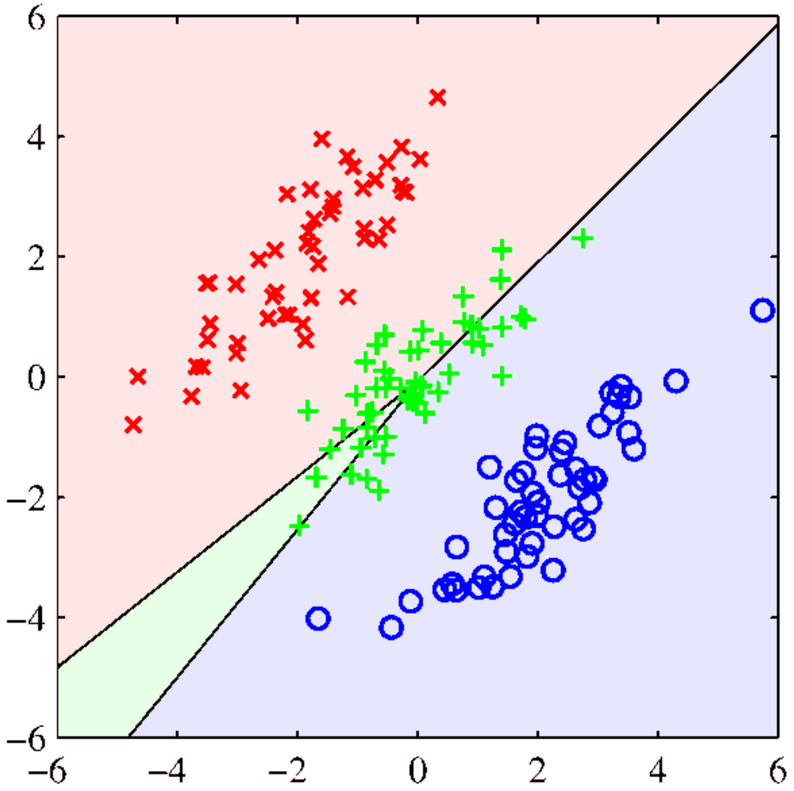
Logistic vs Linear Regression, n>2 classes

Logistic vs Linear Regression, n>2 classes

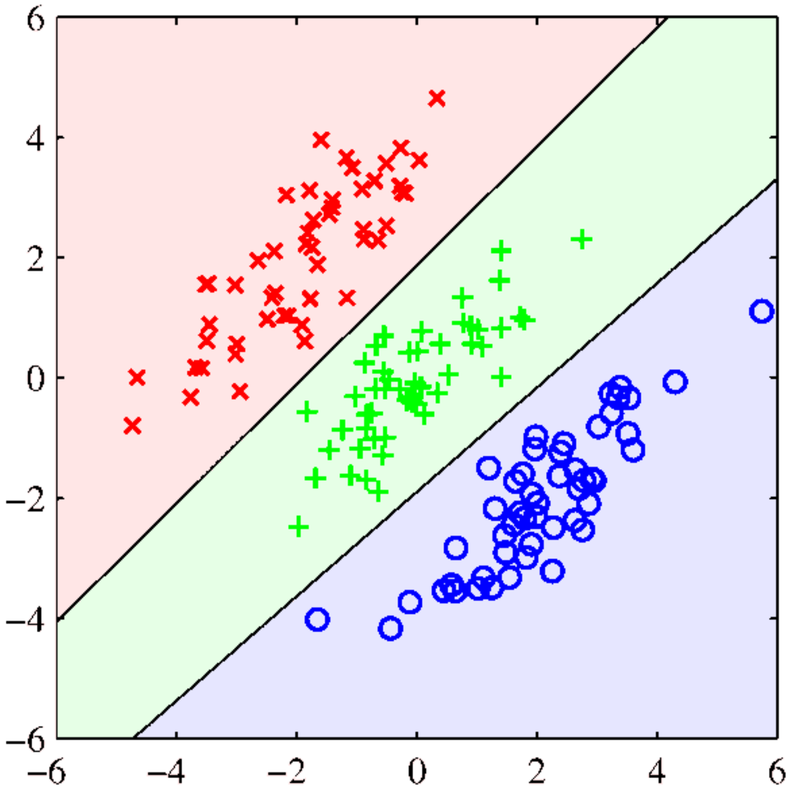


Logistic vs Linear Regression, n>2 classes

Linear regression

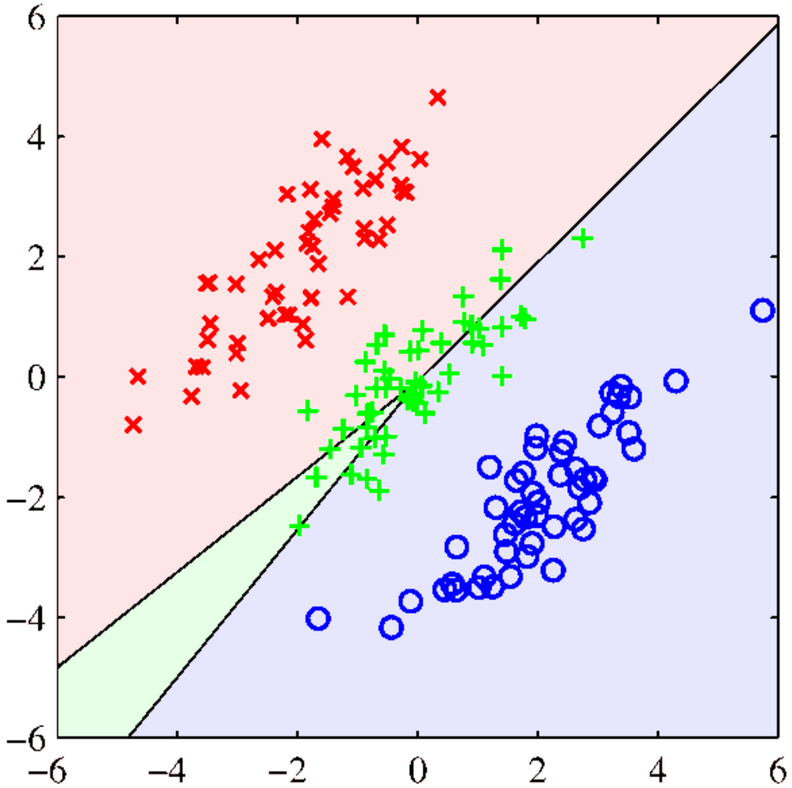


Logistic regression

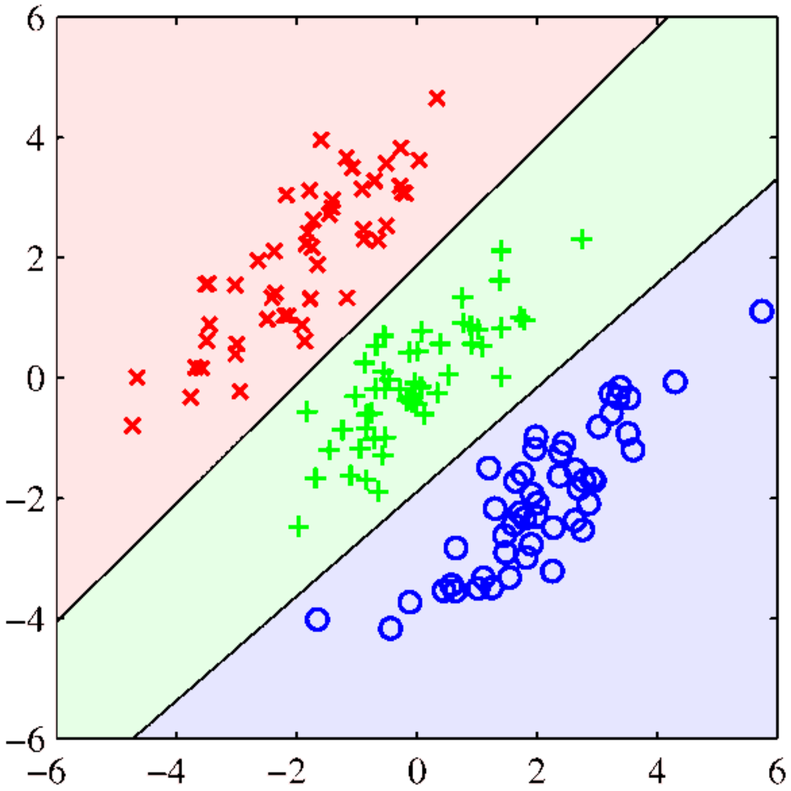


Logistic vs Linear Regression, n>2 classes

Linear regression



Logistic regression



Logistic regression does not exhibit the masking problem

Gradient of Cross-entropy Loss

$$L(\mathbf{w}) = -\sum_{i=1}^{N} y^i \log g(\mathbf{w}^T \mathbf{x}^i) + (1 - y^i) \log(1 - g(\mathbf{w}^T \mathbf{x}^i))$$

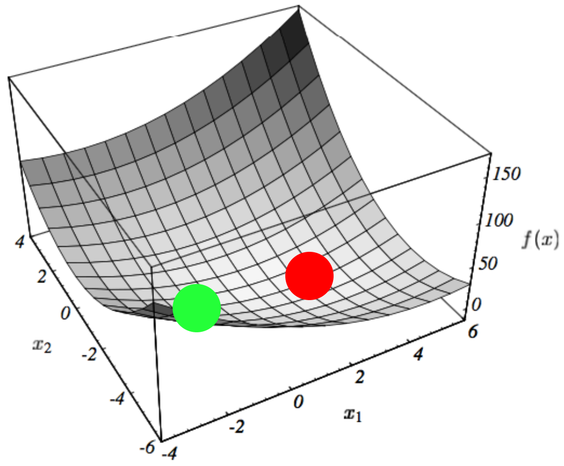
Gradient of Cross-entropy Loss

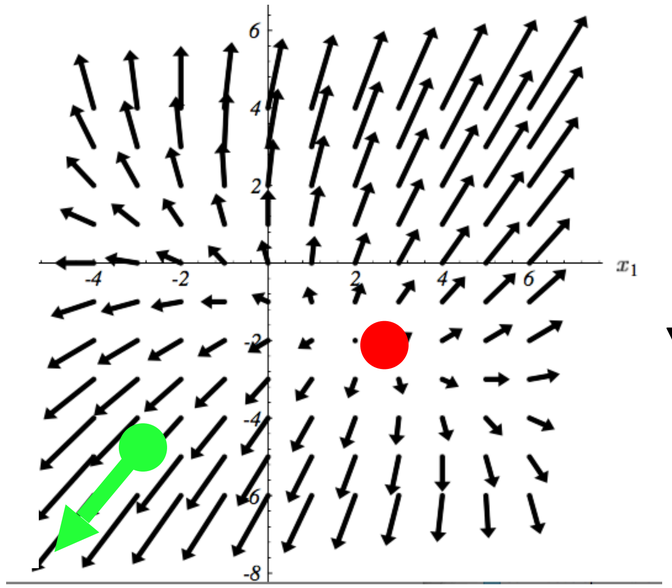
$$L(\mathbf{w}) = -\sum_{i=1}^{N} y^i \log g(\mathbf{w}^T \mathbf{x}^i) + (1 - y^i) \log(1 - g(\mathbf{w}^T \mathbf{x}^i))$$

$$\nabla L(\mathbf{w}^*) = \mathbf{0}$$

nonlinear system of equations!!

Gradient Descent Minimization

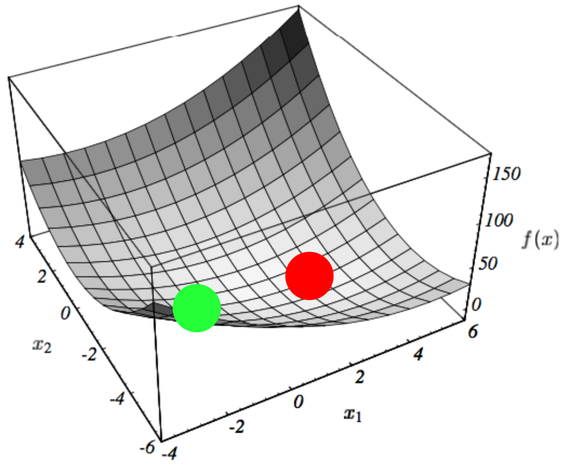


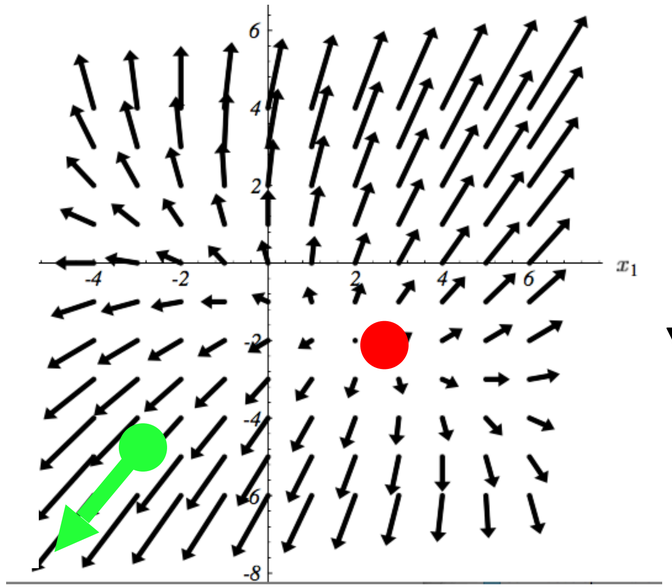


$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

gradient at any point gives direction of fastest increase

Gradient Descent Minimization



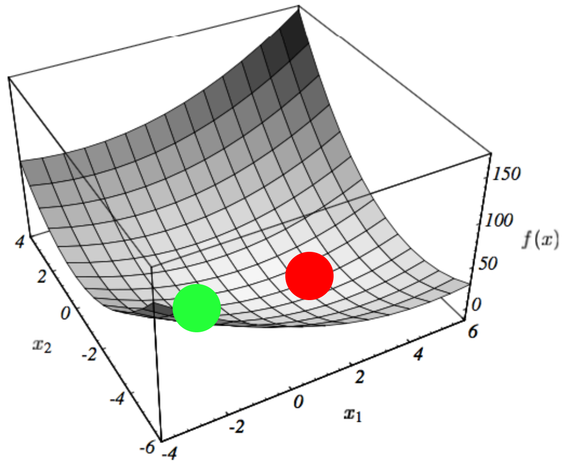


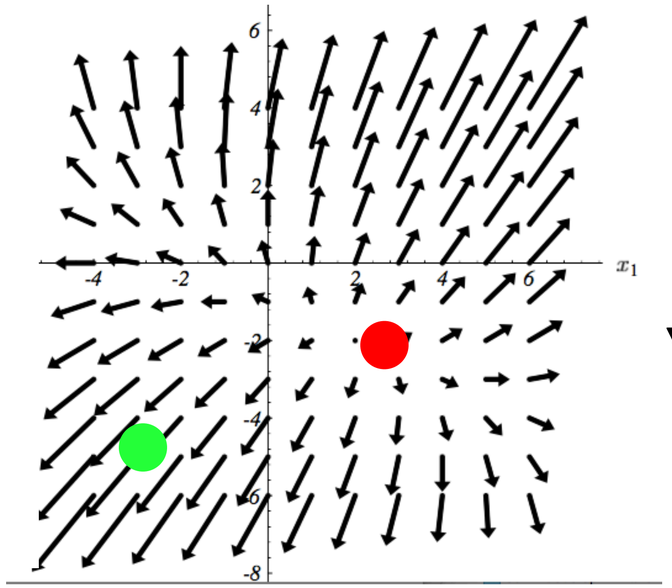
$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

gradient at any point gives direction of fastest increase

Idea: start at a point and move in the direction opposite to the gradient

Gradient Descent Minimization



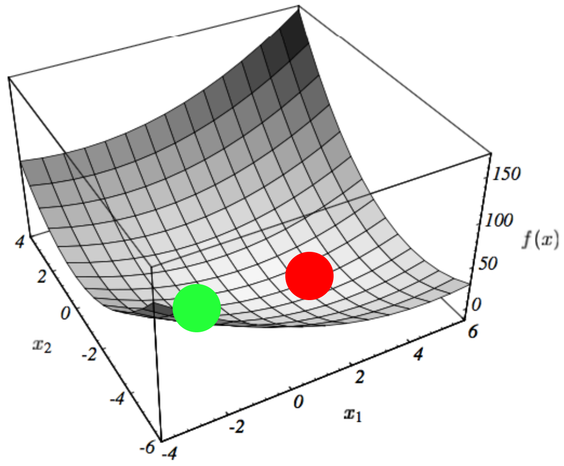


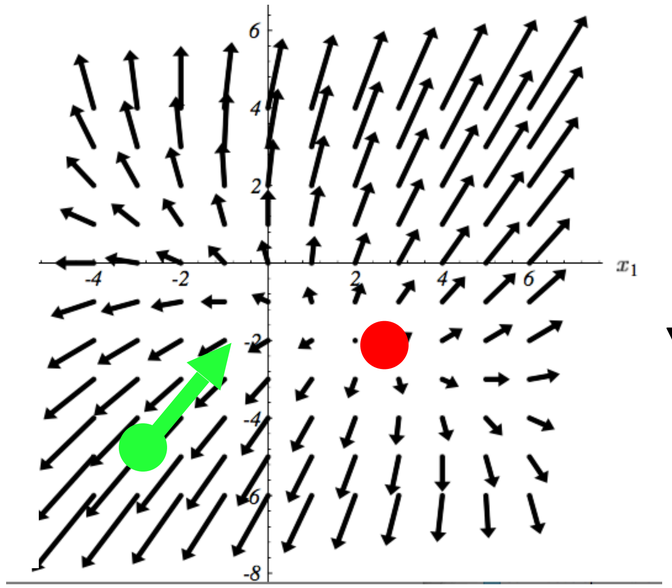
$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

gradient at any point gives direction of fastest increase

Idea: start at a point and move in the direction opposite to the gradient

Gradient Descent Minimization



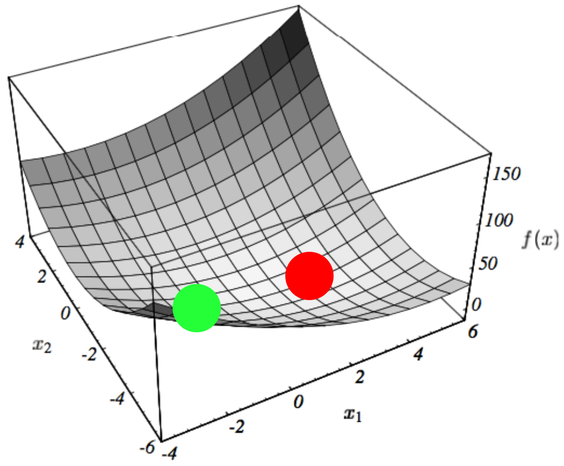


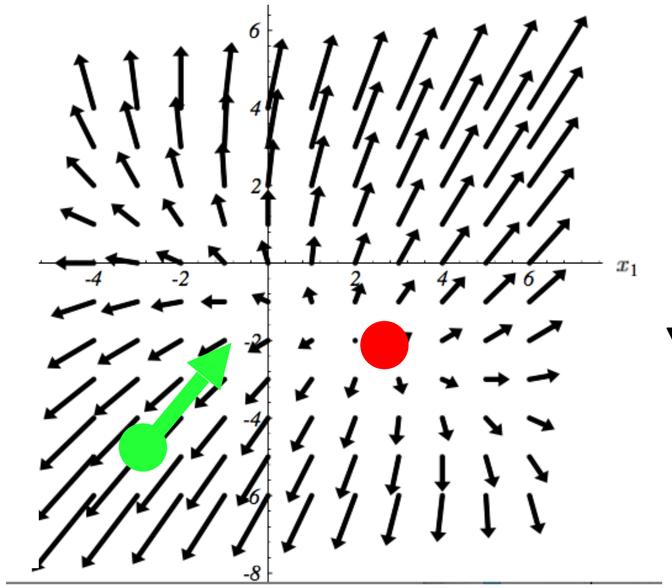
$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

gradient at any point gives direction of fastest increase

Idea: start at a point and move in the direction opposite to the gradient

Gradient Descent Minimization





$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

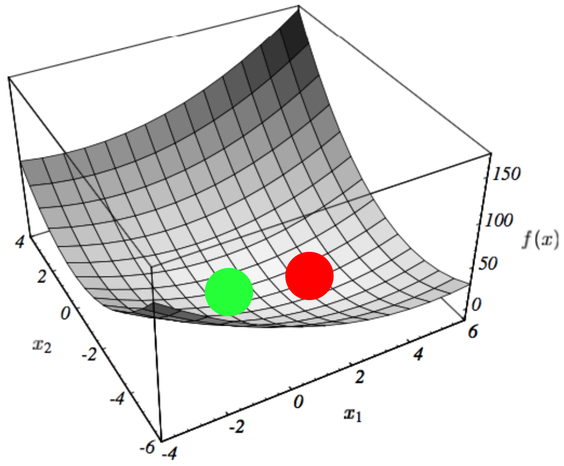
gradient at any point gives direction of fastest increase

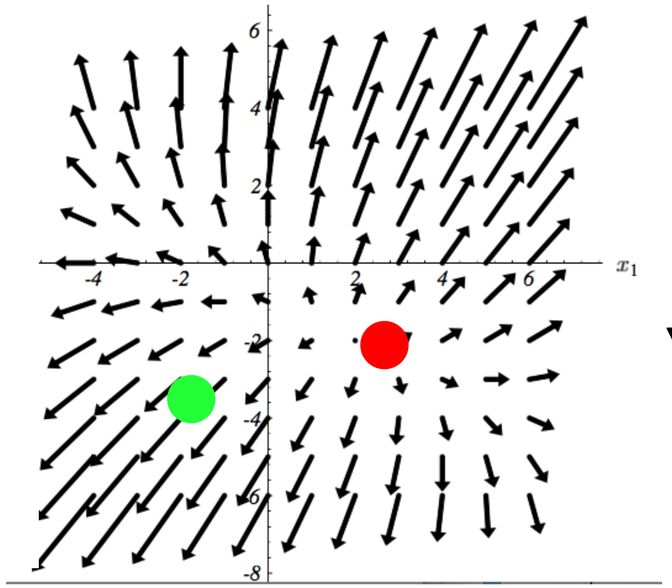
Idea: start at a point and move in the direction opposite to the gradient

Initialize: $\mathbf{x}_0$

Update: $\mathbf{x}_{i+1} = \mathbf{x}_i - \alpha \nabla f(\mathbf{x}_i)$ **i=0**

Gradient Descent Minimization

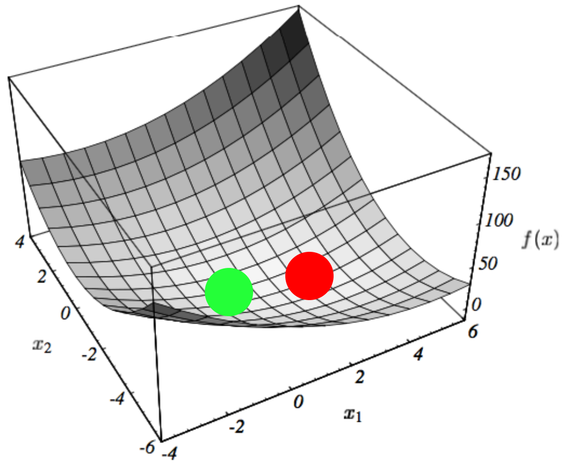


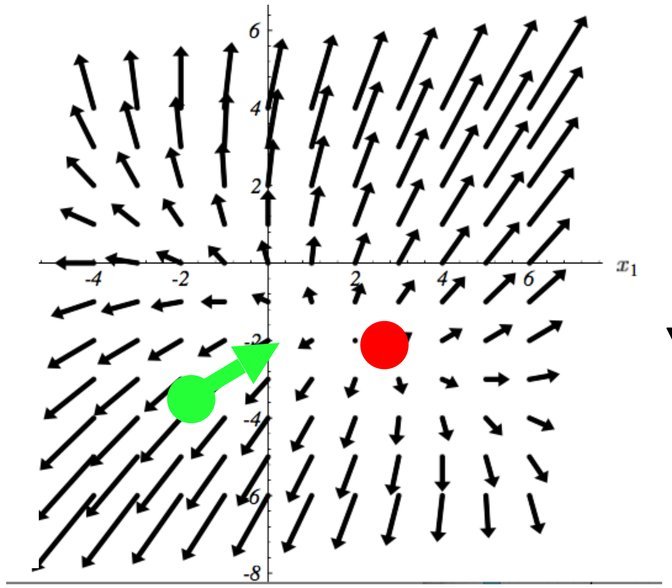


$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

Update: $$\mathbf{x}_{i+1} = \mathbf{x}_i - \alpha \nabla f(\mathbf{x}_i)$$ **i=1**

Gradient Descent Minimization

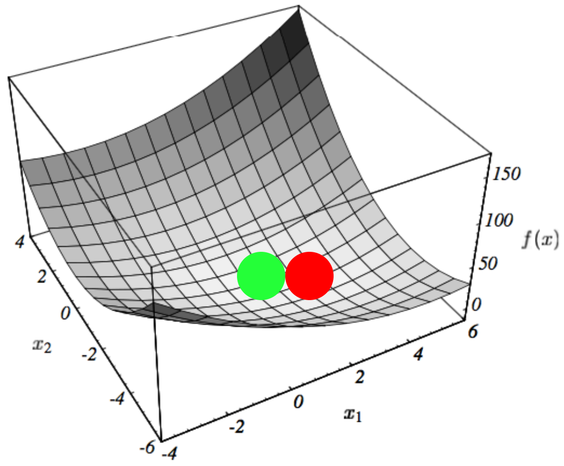


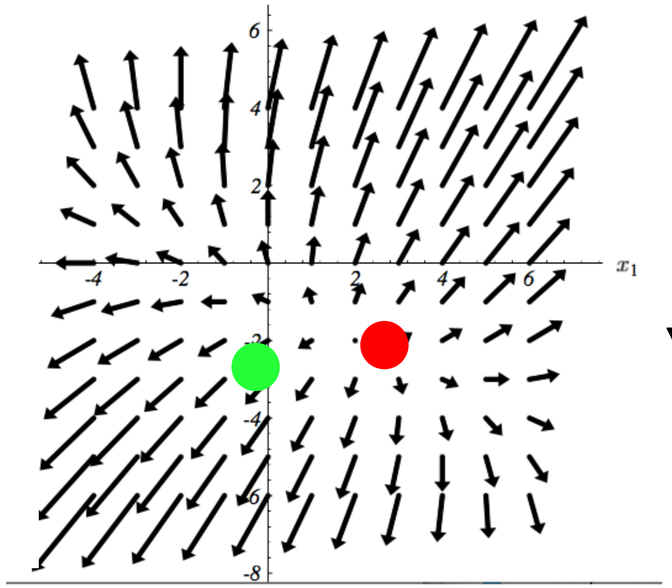


$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

Update: $$\mathbf{x}_{i+1} = \mathbf{x}_i - \alpha \nabla f(\mathbf{x}_i)$$ **i=1**

Gradient Descent Minimization

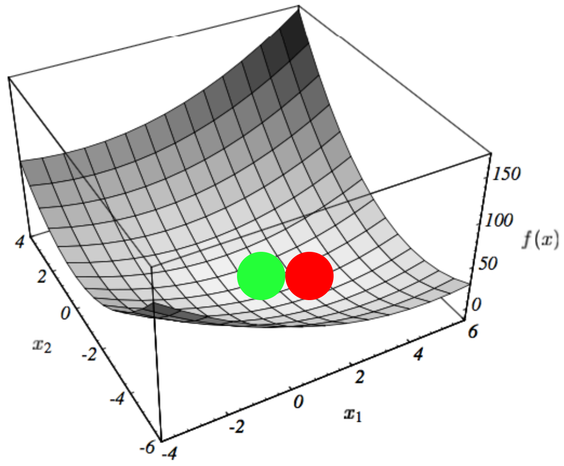


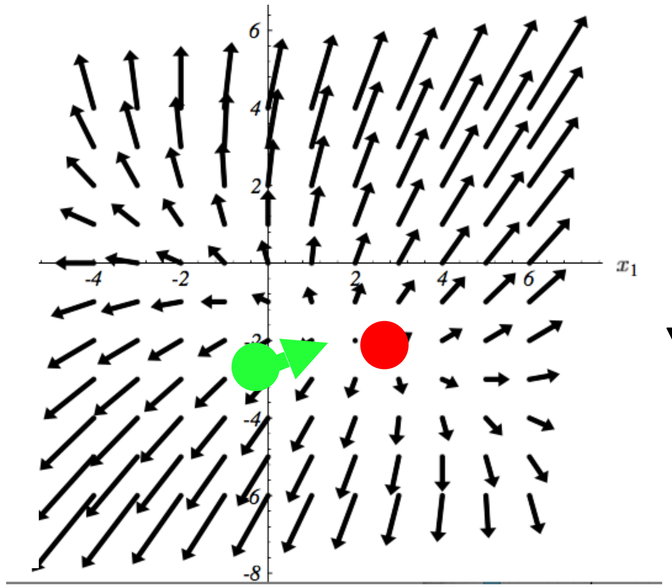


$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

Update: $$\mathbf{x}_{i+1} = \mathbf{x}_i - \alpha \nabla f(\mathbf{x}_i)$$ **i=2**

Gradient Descent Minimization

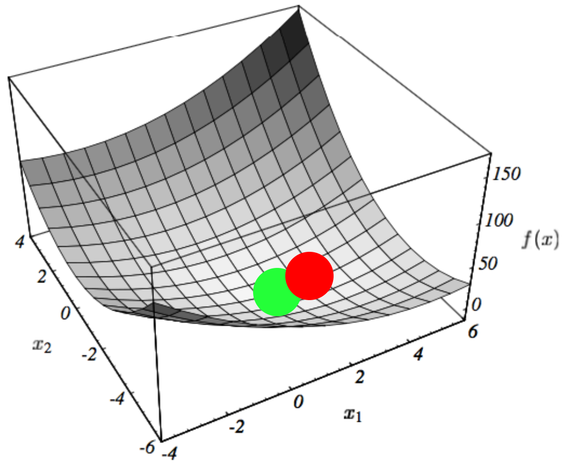


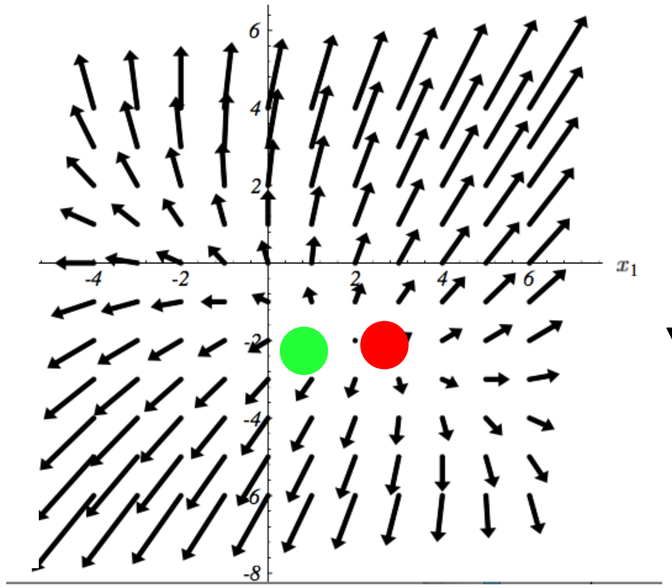


$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

Update: $$\mathbf{x}_{i+1} = \mathbf{x}_i - \alpha \nabla f(\mathbf{x}_i)$$ **i=2**

Gradient Descent Minimization





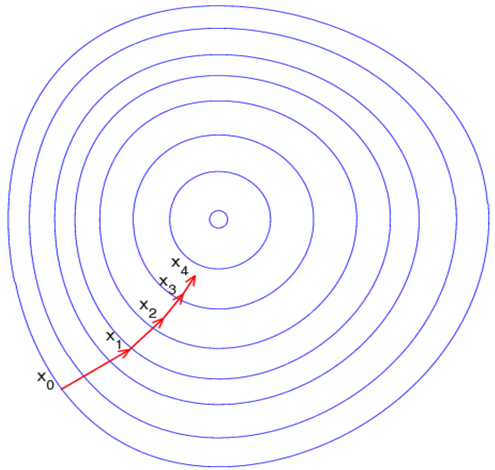
$$\nabla f(\mathbf{x}) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$$

Update: $$\mathbf{x}_{i+1} = \mathbf{x}_i - \alpha \nabla f(\mathbf{x}_i)$$ **i=3**

Gradient Descent Minimization

Initialize: $\mathbf{x}_0$

Update: $\mathbf{x}_{i+1} = \mathbf{x}_i - \alpha \nabla f(\mathbf{x}_i)$

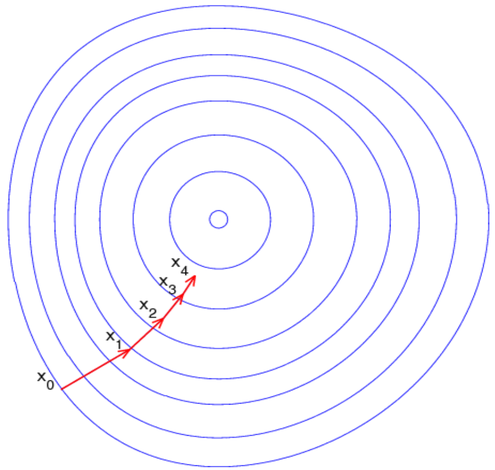


Gradient Descent Minimization

Initialize: $\mathbf{x}_0$

Update: $\mathbf{x}_{i+1} = \mathbf{x}_i - \alpha \nabla f(\mathbf{x}_i)$

We can always make it converge for a convex function.

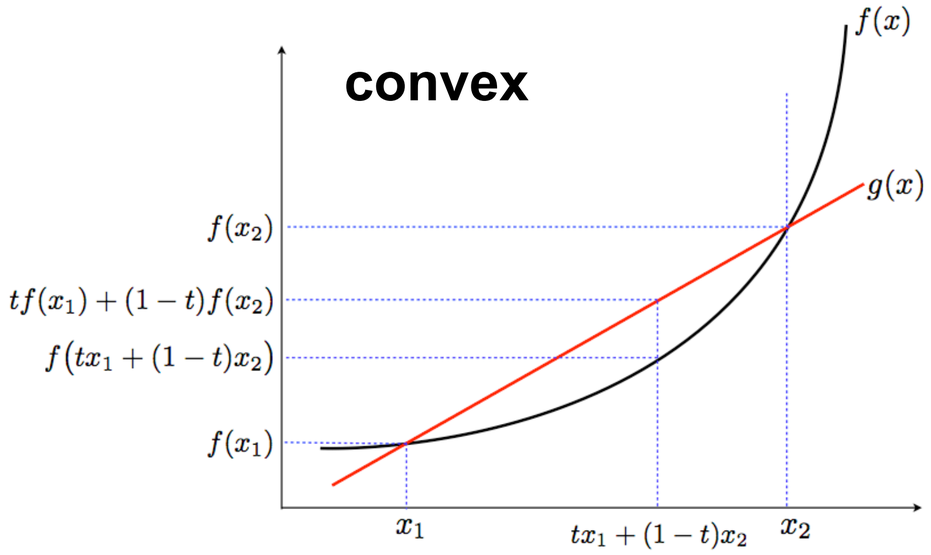


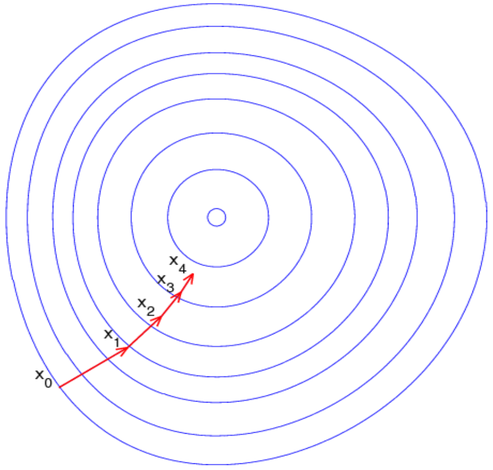
Gradient Descent Minimization

Initialize: $\mathbf{x}_0$

Update: $\mathbf{x}_{i+1} = \mathbf{x}_i - \alpha \nabla f(\mathbf{x}_i)$

We can always make it converge for a convex function.



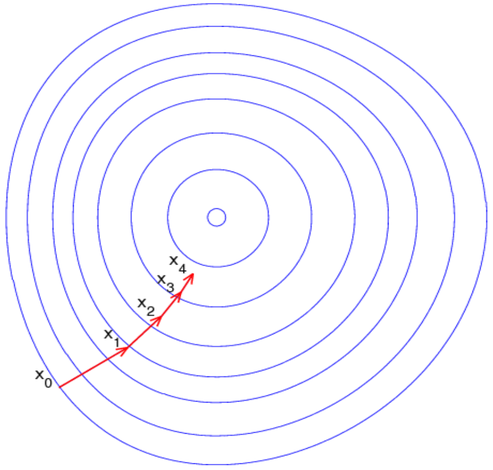


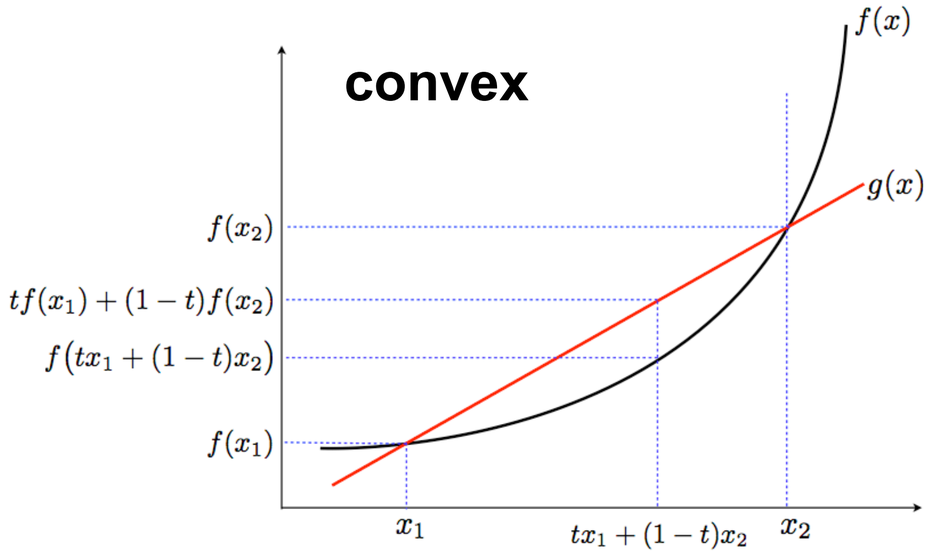
Gradient Descent Minimization

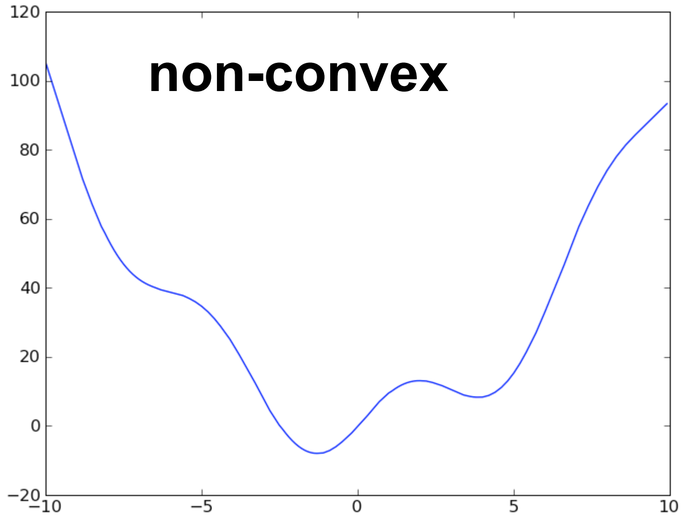
Initialize: $\mathbf{x}_0$

Update: $\mathbf{x}_{i+1} = \mathbf{x}_i - \alpha \nabla f(\mathbf{x}_i)$

We can always make it converge for a convex function.







XOR Problem

XOR Problem

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

XOR Problem

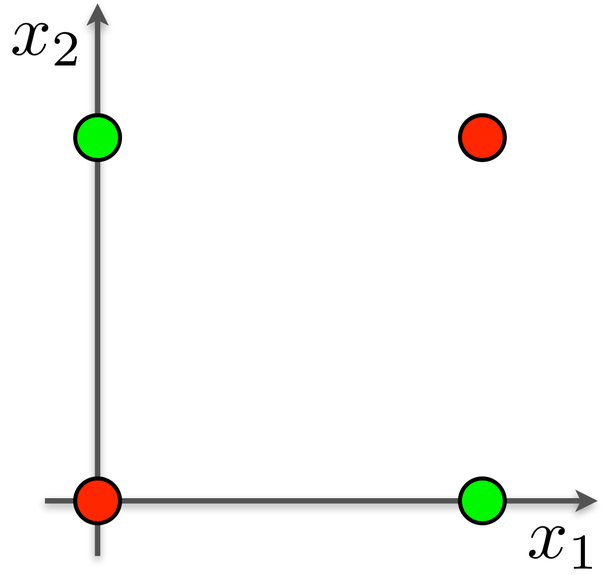
$$y = f(x_1, x_2)$$

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

XOR Problem

$$y = f(x_1, x_2)$$

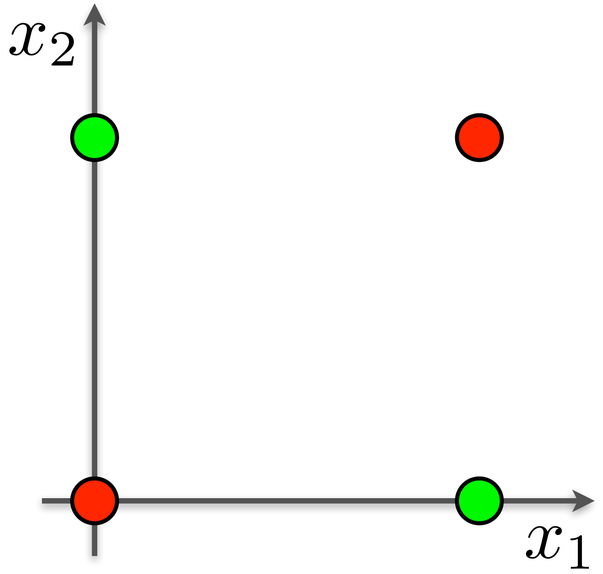
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0



XOR Problem

$$y = f(x_1, x_2)$$

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

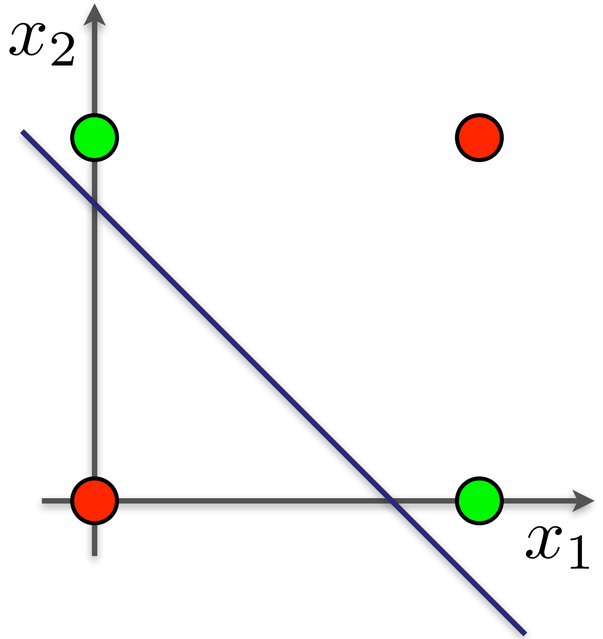


$$y = f(w_0, w_1, w_2) = \mathcal{H}(w_0 + w_1 x_1 + w_2 x_2)$$

XOR Problem

$$y = f(x_1, x_2)$$

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

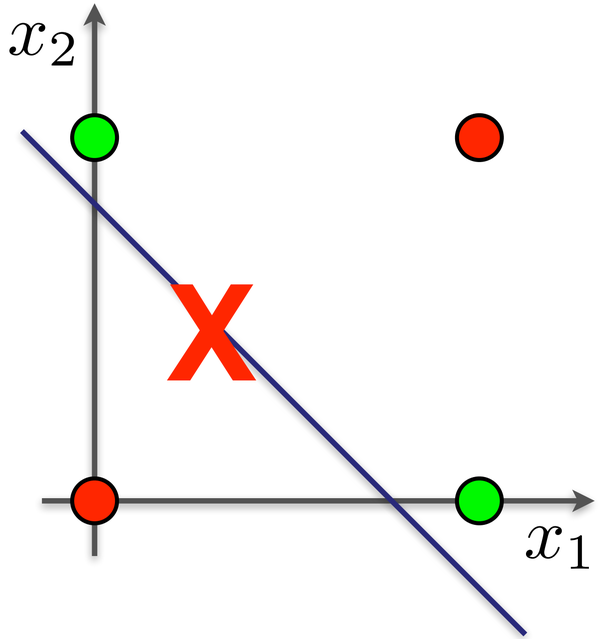


$$y = f(w_0, w_1, w_2) = \mathcal{H}(w_0 + w_1 x_1 + w_2 x_2)$$

XOR Problem

$$y = f(x_1, x_2)$$

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0



$$y = f(w_0, w_1, w_2) = \mathcal{H}(w_0 + w_1 x_1 + w_2 x_2)$$

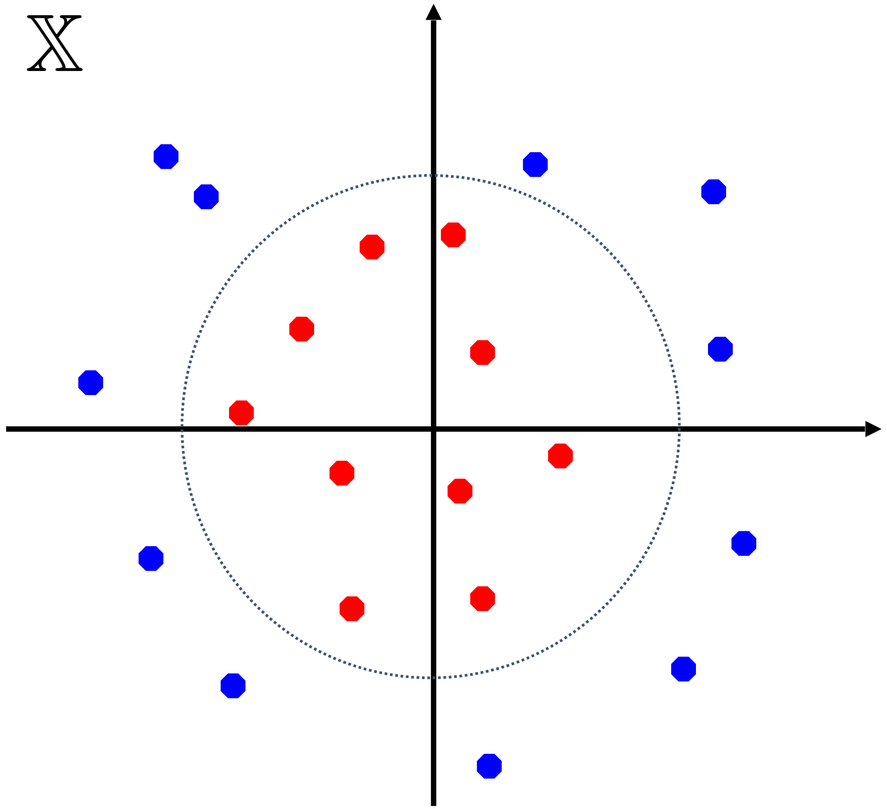
Regression

1. Least Squares fitting

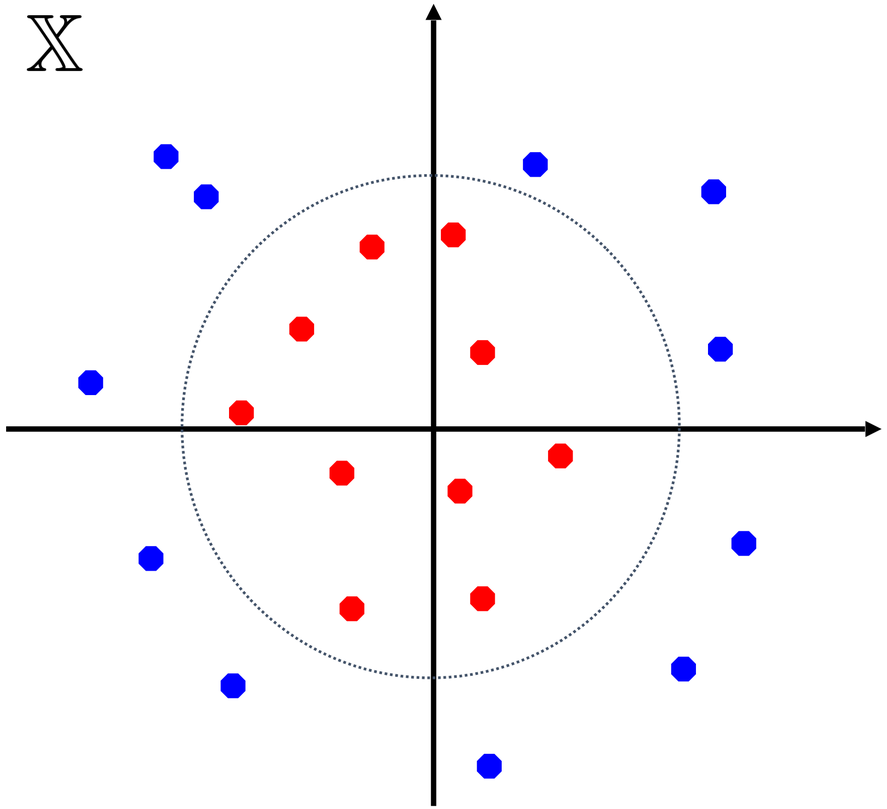
2. Nonlinear error function and gradient descent

3. Perceptron training (simple neural network)

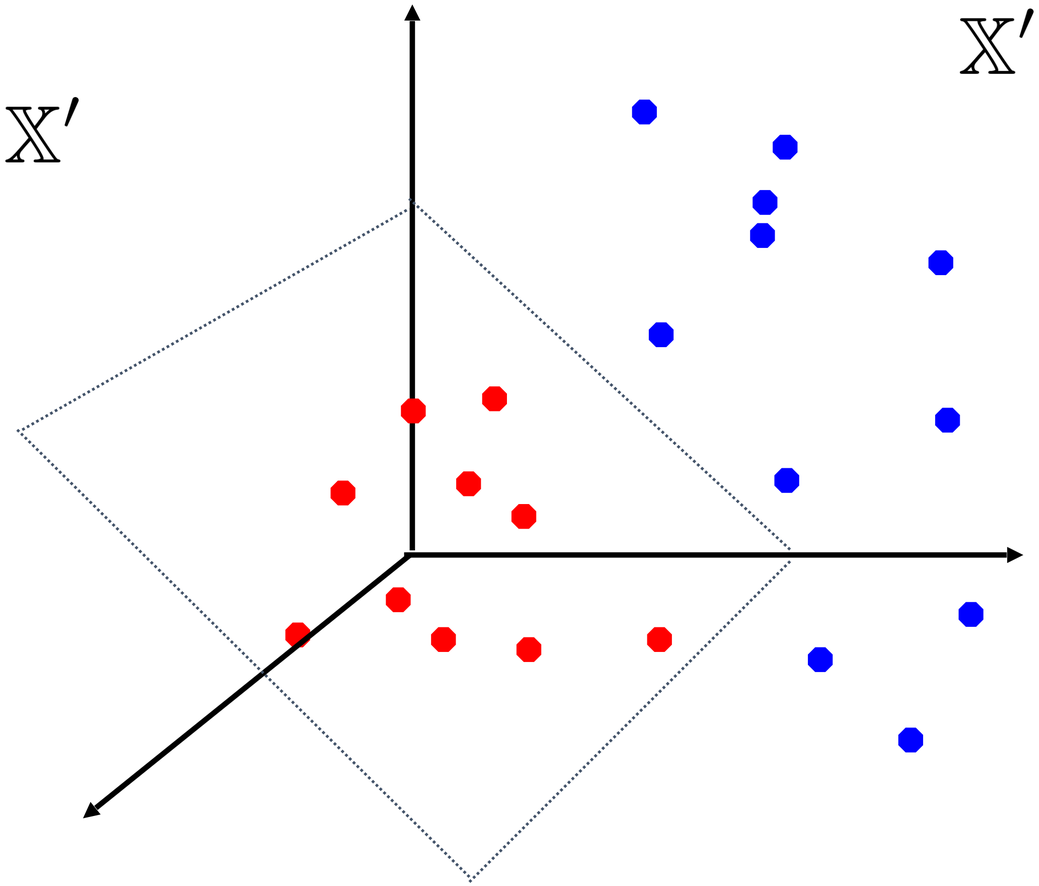
Lifting to Higher Dimensions



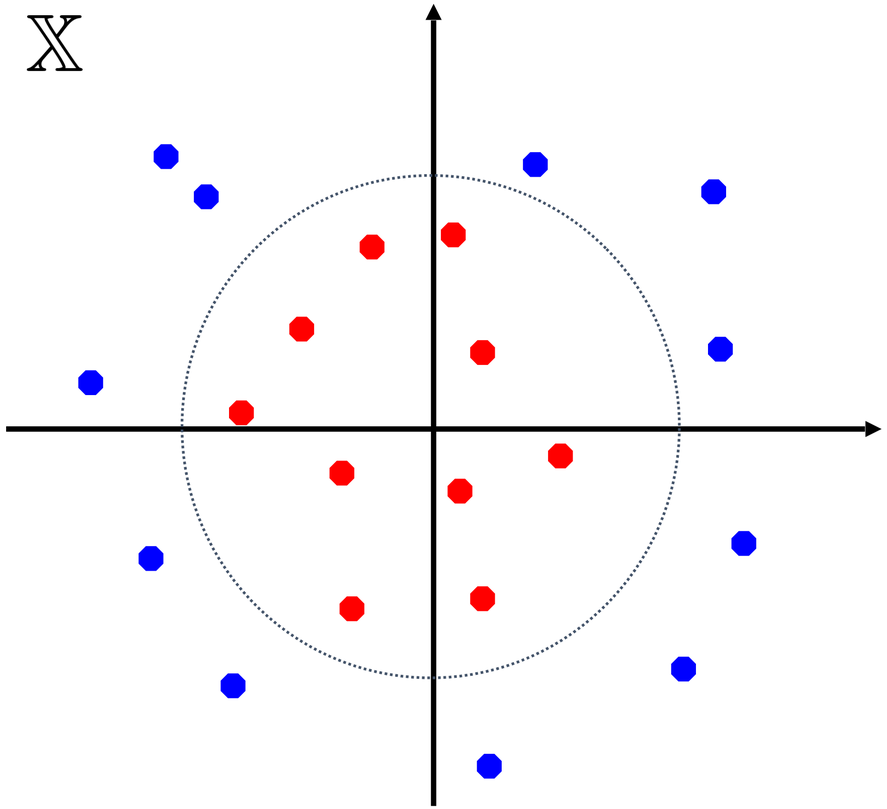
Lifting to Higher Dimensions



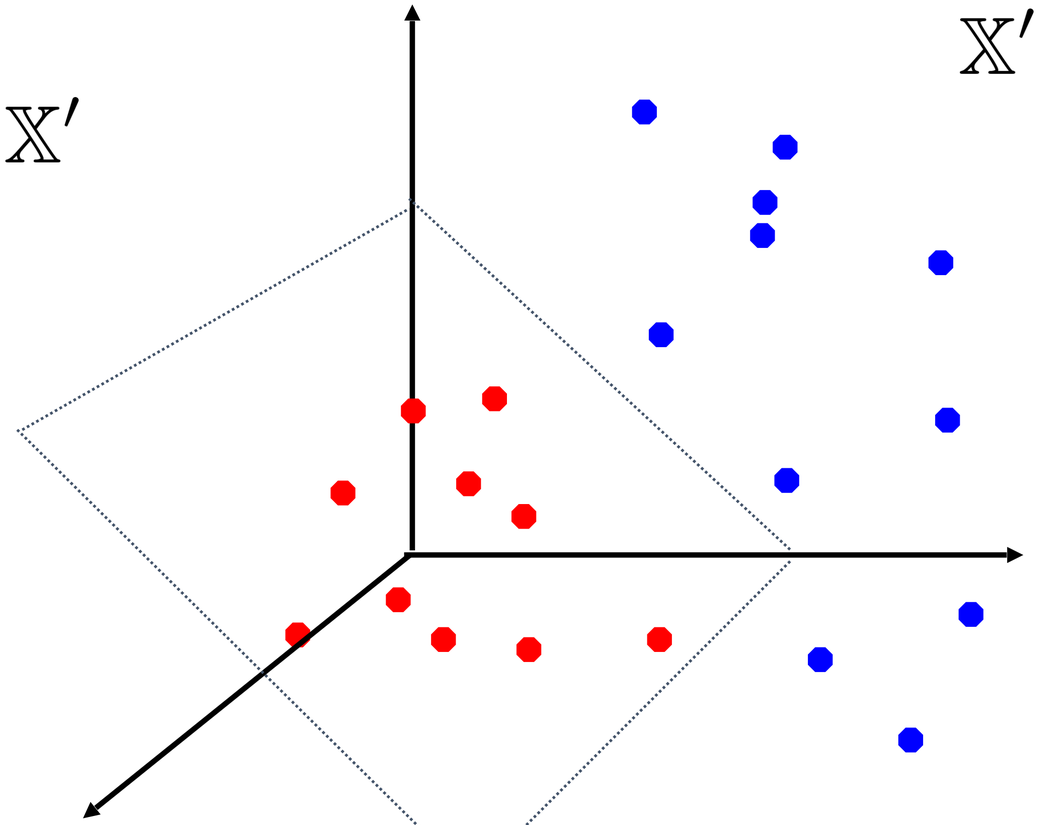
$$g : \mathbb{X} \longrightarrow \mathbb{X}'$$



Lifting to Higher Dimensions



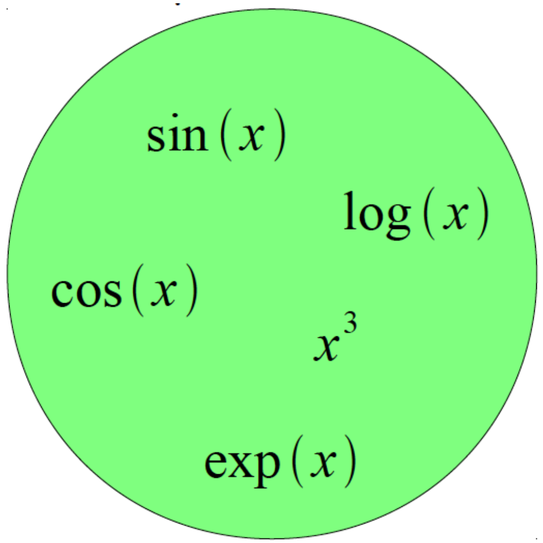
$$g : \mathbb{X} \longrightarrow \mathbb{X}'$$



$$f_\theta(x) = \begin{cases} 1 & \text{if } w\ g(x) + b \geq 0 \\ 0 & \text{if } w\ g(x) + b < 0 \end{cases}$$

Building A Complicated Function

Given a library of simple functions



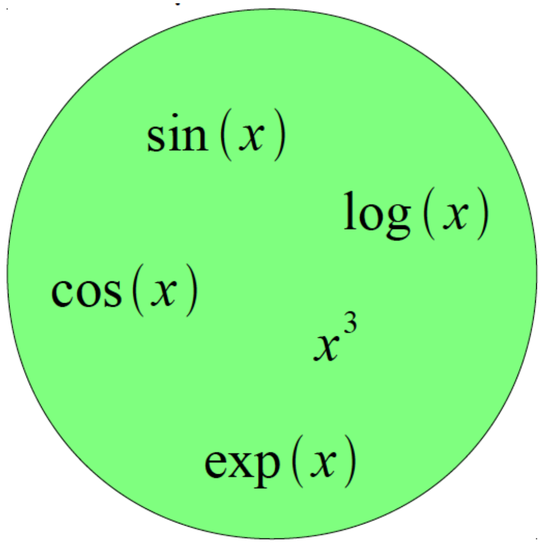
Compose into a

complicated function

Slide Credit: Marc'Aurelio Ranzato, Yann LeCun

Building A Complicated Function

Given a library of simple functions



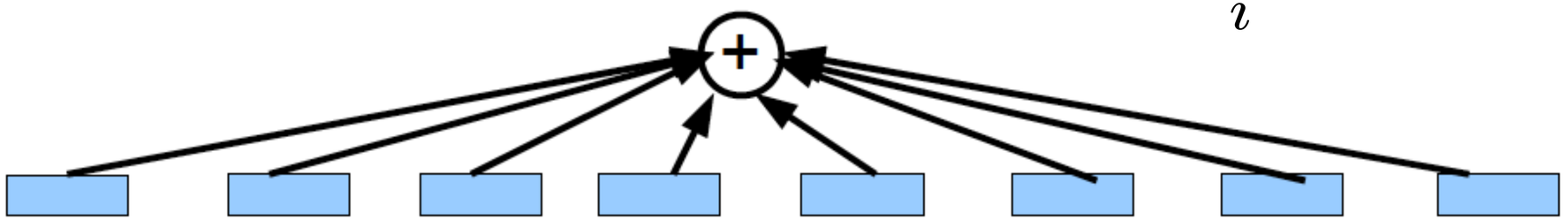
Compose into a

complicated function

Idea 1: Linear Combinations

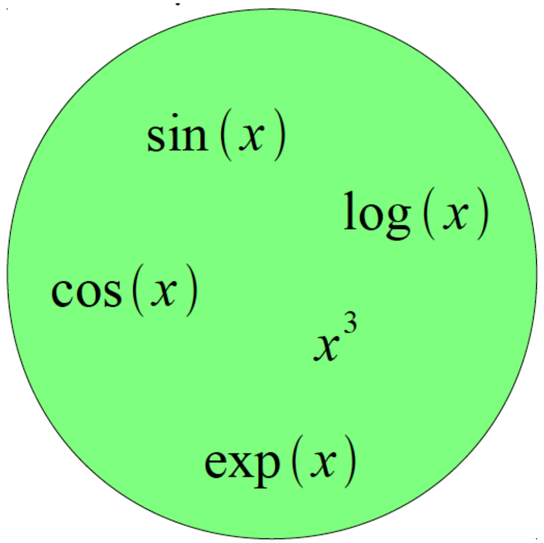
- Boosting
- Kernels
- …

$$f(x) = \sum_i \alpha_i g_i(x)$$



Building A Complicated Function

Given a library of simple functions



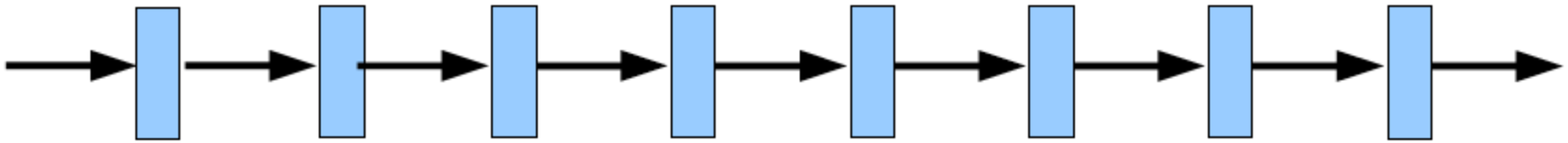
Compose into a

complicated function

Idea 2: Compositions

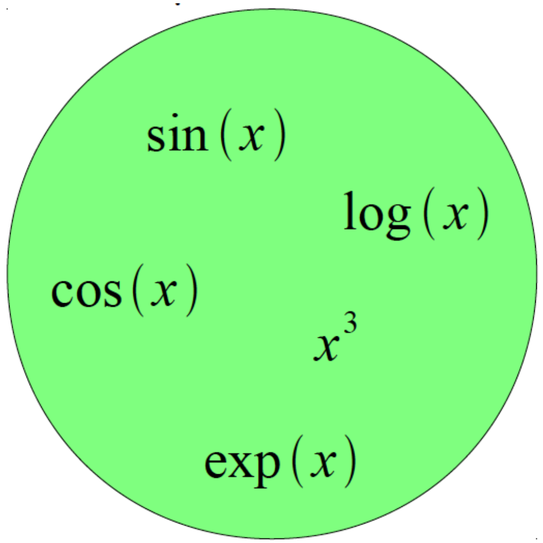
- Decision Trees
- Deep Learning

$$f(x) = g_1(g_2(\ldots(g_n(x)\ldots))$$



Building A Complicated Function

Given a library of simple functions



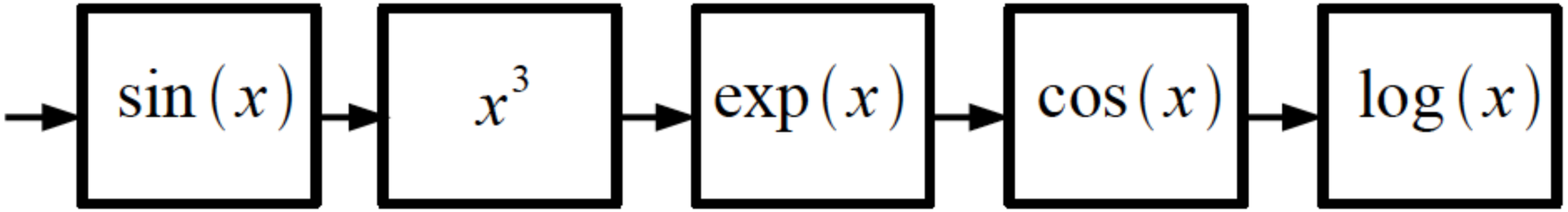
Compose into a

complicated function

Idea 2: Compositions

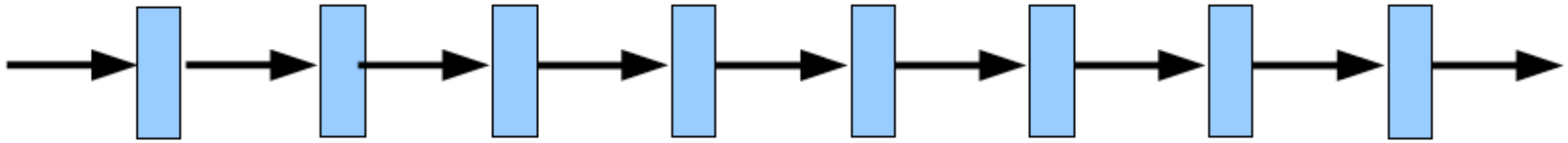
- Decision Trees
- Deep Learning

$$f(x) = \log(\cos(\exp(\sin^3(x))))$$



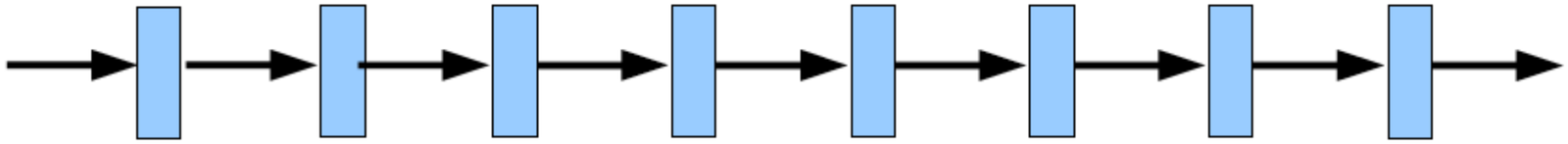
‘Neuron’: Cascade of Linear and Nonlinear Function

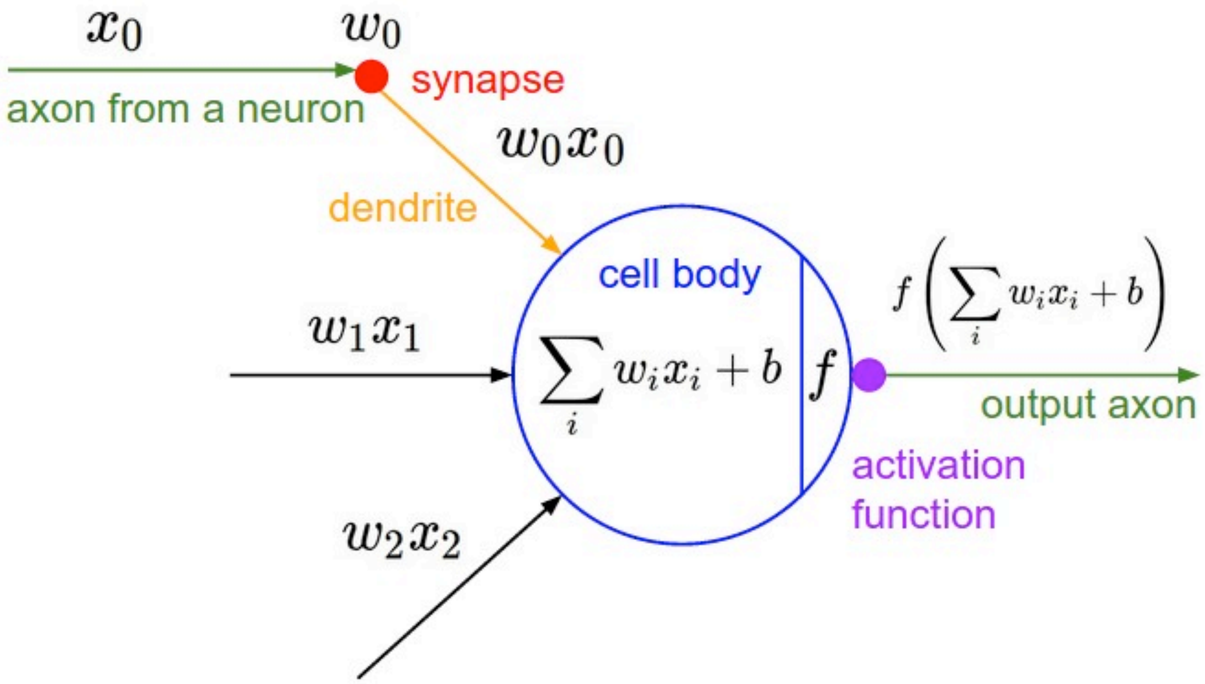
basic building block



‘Neuron’: Cascade of Linear and Nonlinear Function

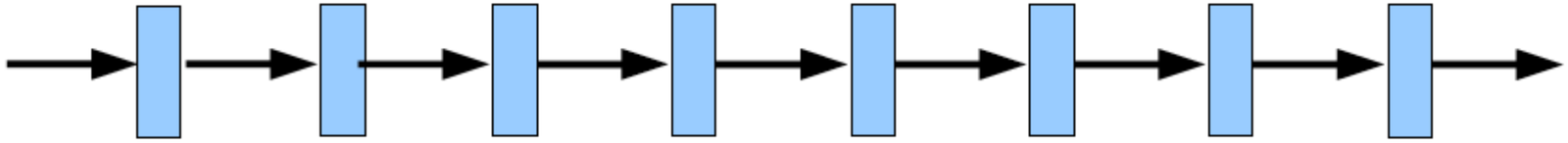
basic building block

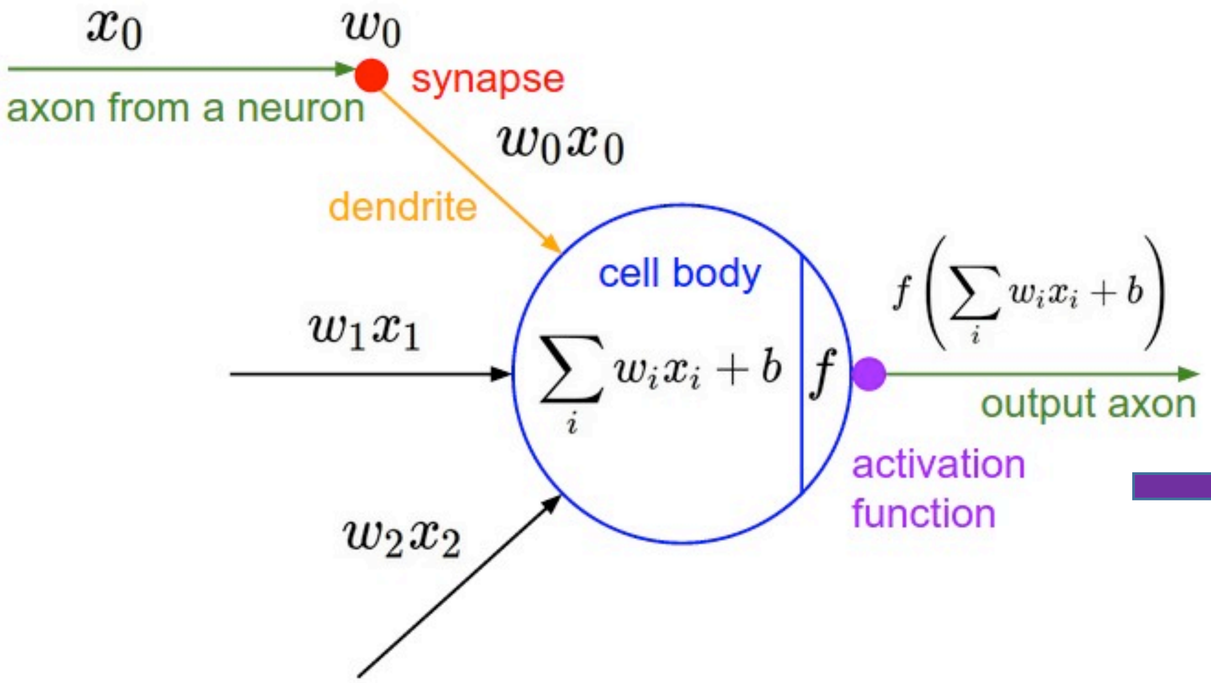




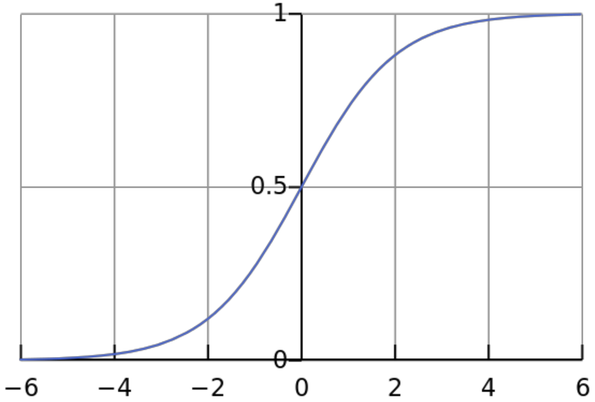
‘Neuron’: Cascade of Linear and Nonlinear Function

basic building block



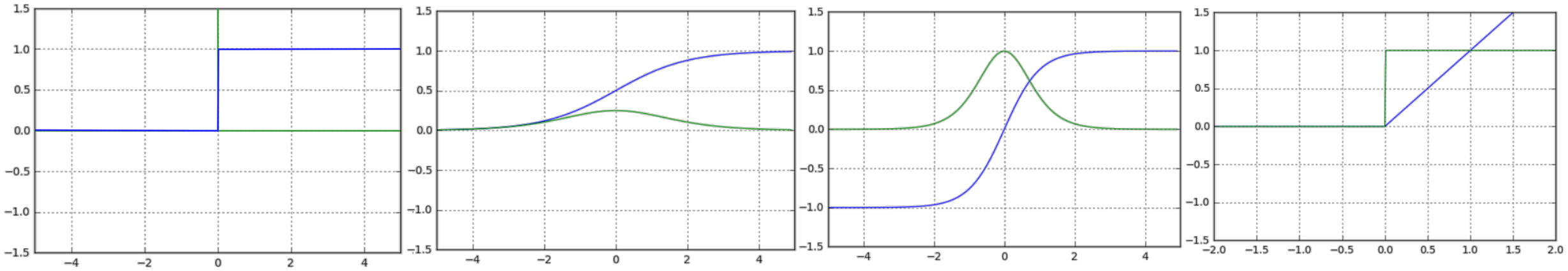


Sigmoidal activation



TOKYO

Activation Functions



Step ("perceptron")

$$g(a) = \begin{cases} 0 & a < 0 \\ 1 & a \geq 0 \end{cases}$$

Sigmoidal ("logistic")

$$g(a) = \frac{1}{1 + \exp(-a)}$$

Hyperbolic tangent

$$g(a) = \frac{\exp(a) - \exp(-a)}{\exp(a) + \exp(-a)}$$

Rectified Linear Unit (RELU)

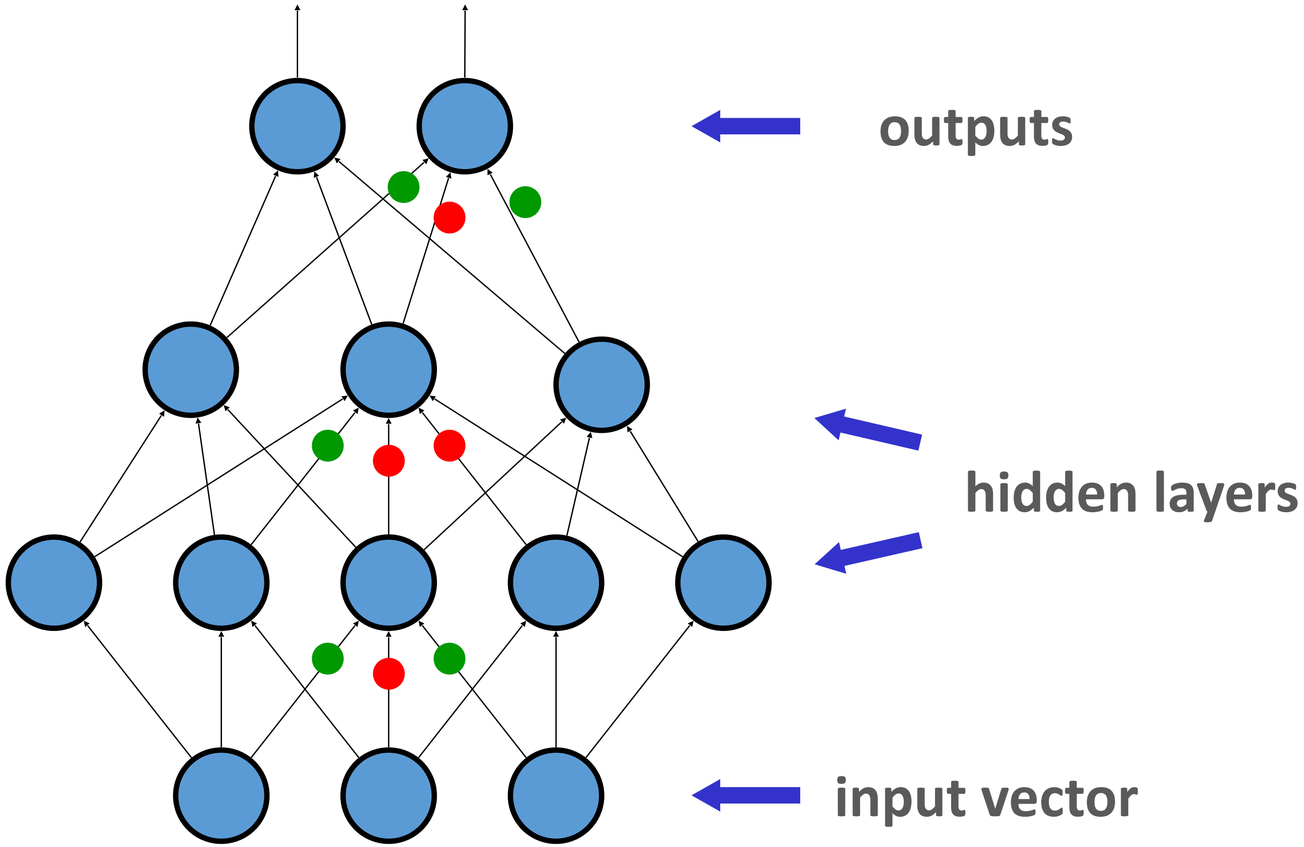
$$g(a) = \max(0, a)$$

SIGGRAPH ASIA 2018 TOKYO

Image Credit: Olivier Grisel and Charles Ollion

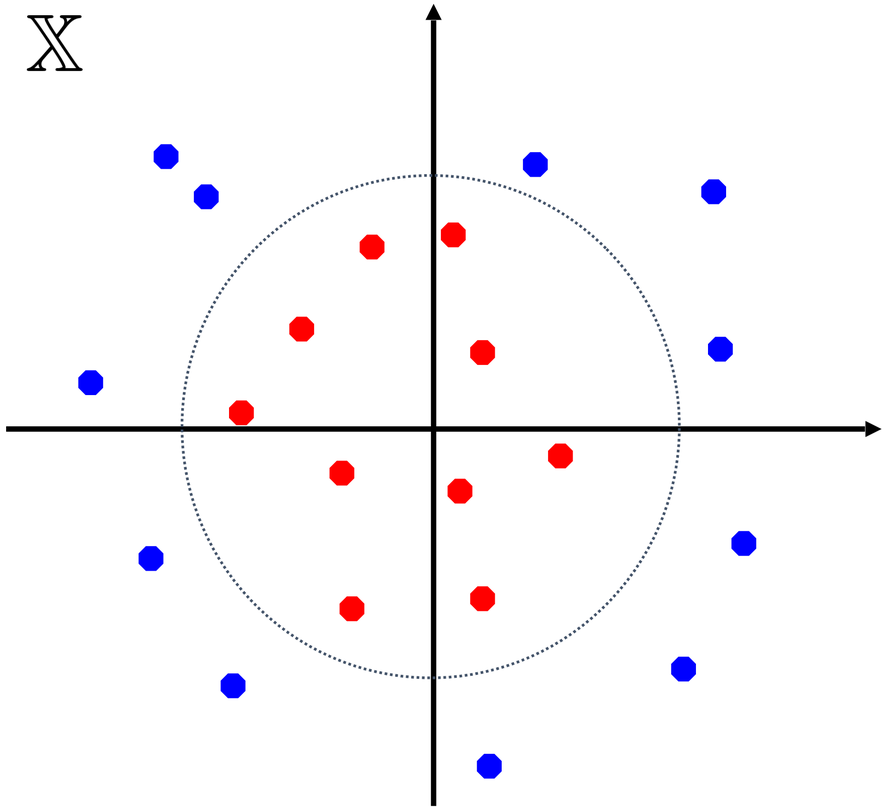
Multi-Layer Perceptrons (~1985)

$$u_i = g\left(\sum_{k\in\mathcal{N}(i)} w_{k,i}\, g\left(\sum_{m\in\mathcal{N}(k)} w_{m,k}u_m + b_k\right) + b_i\right)$$

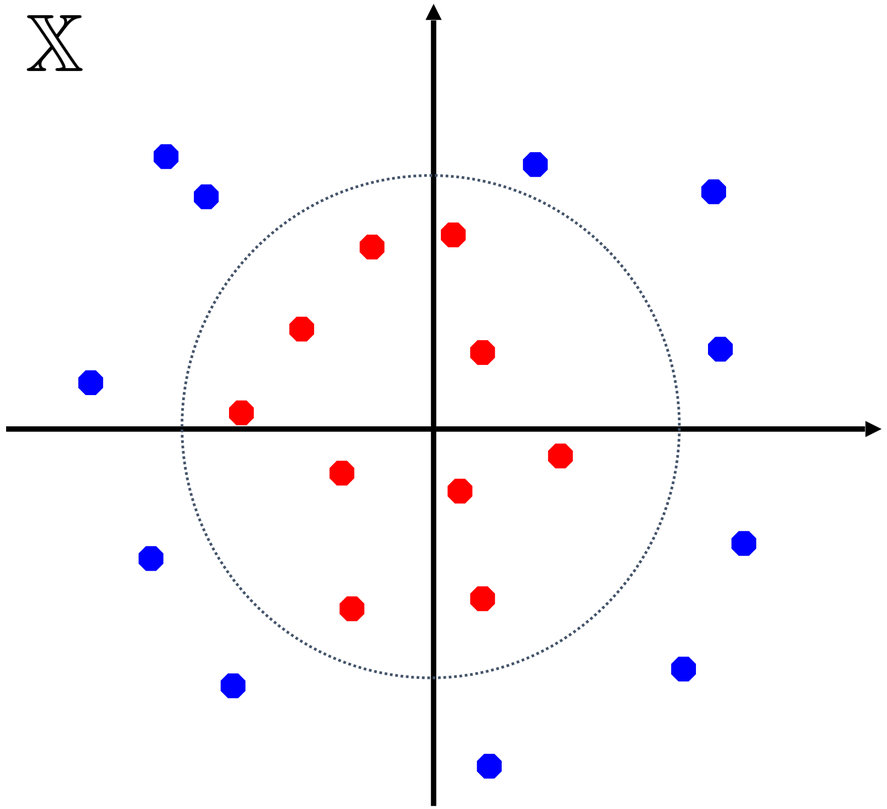


Slide credit: G. Hinton

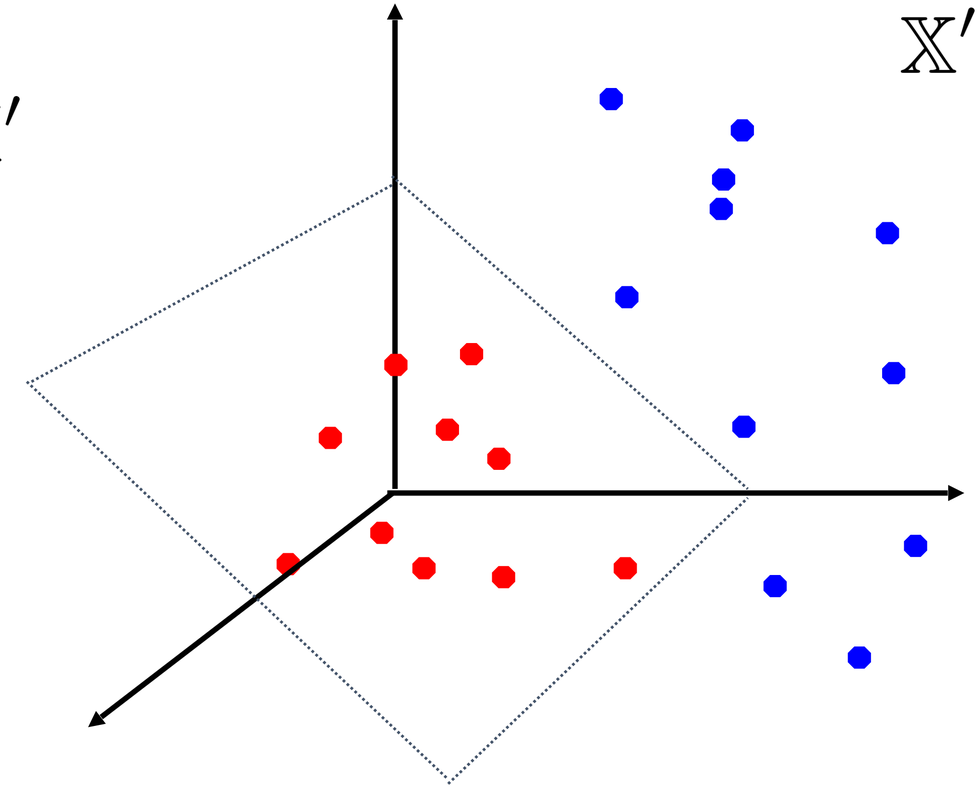
Reminder: Non-linear Decision Boundaries



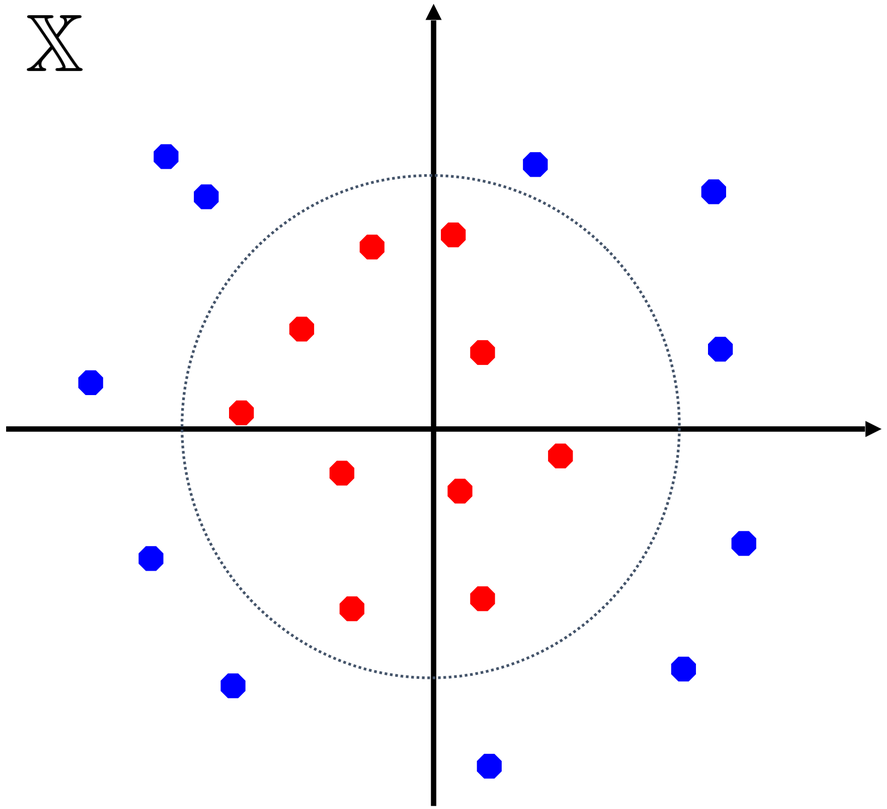
Reminder: Non-linear Decision Boundaries



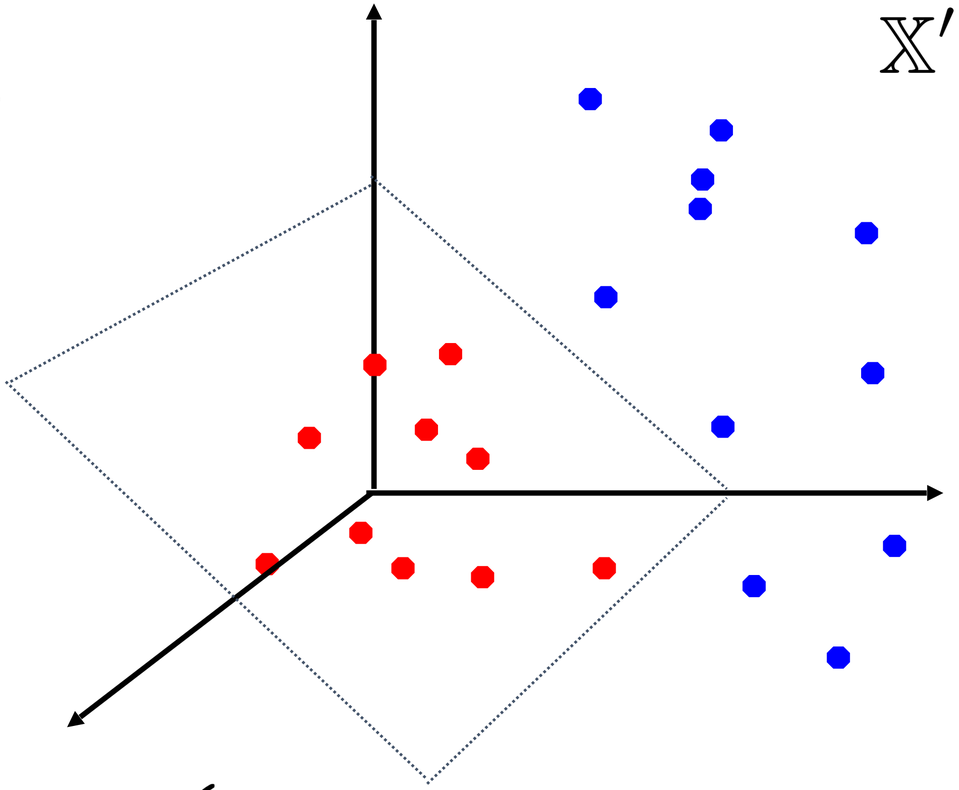
$$g : \mathbb{X} \longrightarrow \mathbb{X}'$$



Reminder: Non-linear Decision Boundaries



$$g : \mathbb{X} \longrightarrow \mathbb{X}'$$

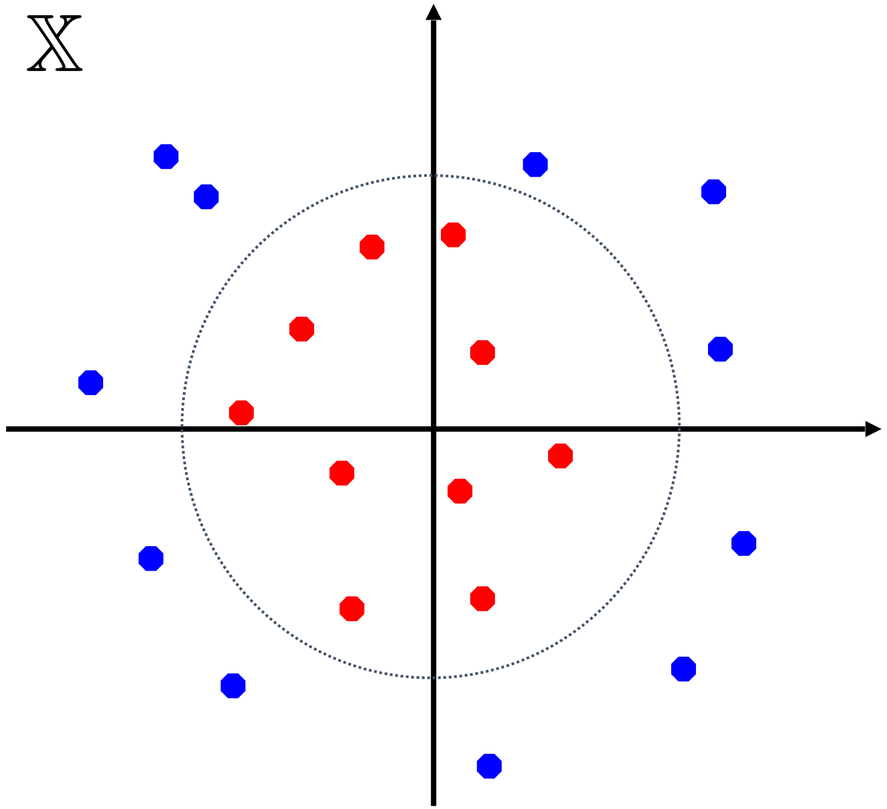


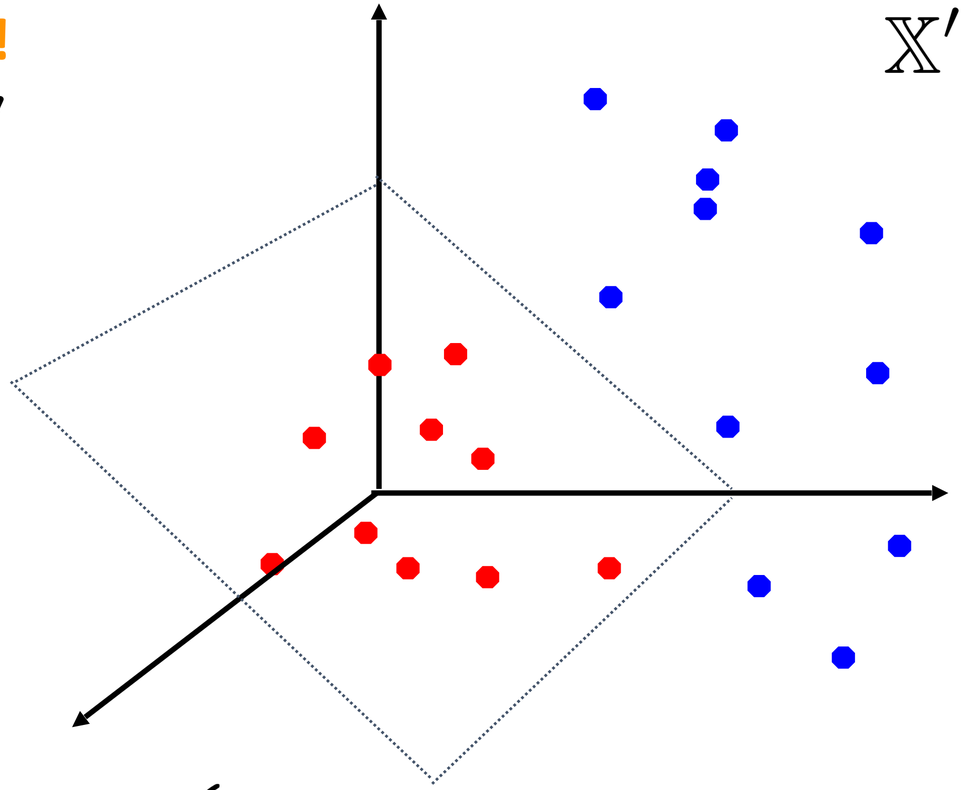
$$f_\theta(x) = \begin{cases} 1 & \text{if } w\ g(x) + b \geq 0 \\ 0 & \text{if } w\ g(x) + b < 0 \end{cases}$$

Reminder: Non-linear Decision Boundaries

This is what the hidden layers should be doing!

$$g : \mathbb{X} \longrightarrow \mathbb{X}'$$

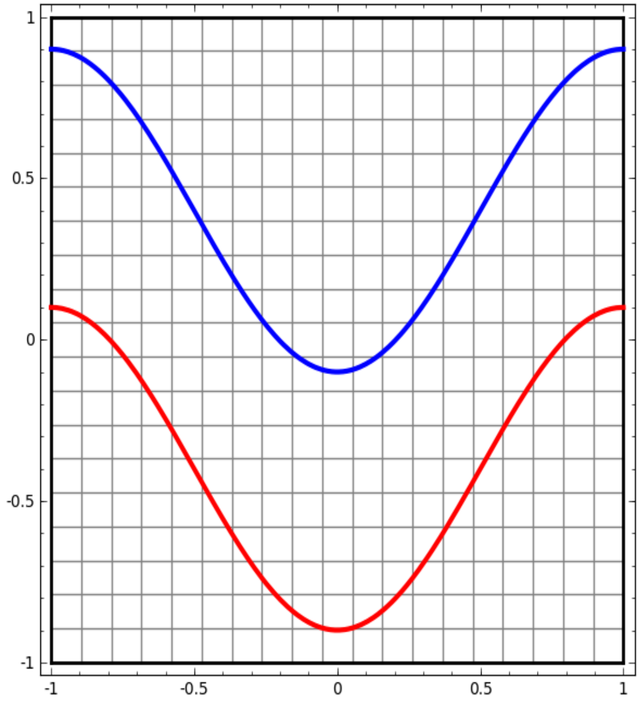




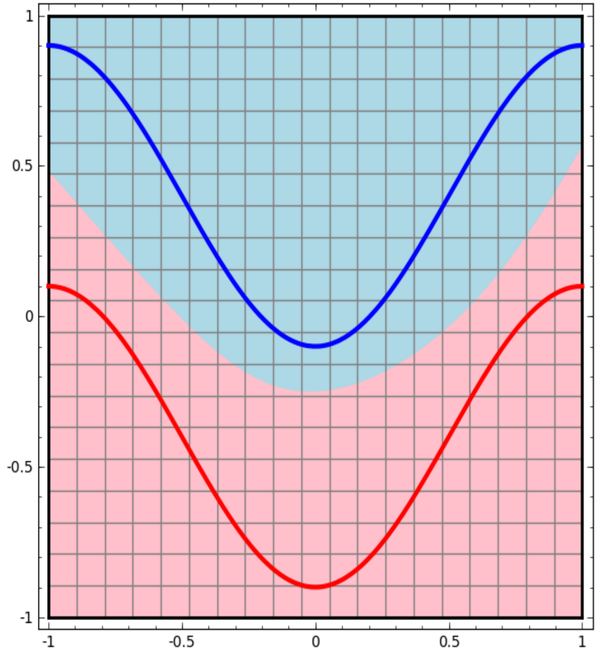
$$f_\theta(x) = \begin{cases} 1 & \text{if } w\ g(x) + b \geq 0 \\ 0 & \text{if } w\ g(x) + b < 0 \end{cases}$$

From Non-separable to Linearly Separable

Non-linearly separable data



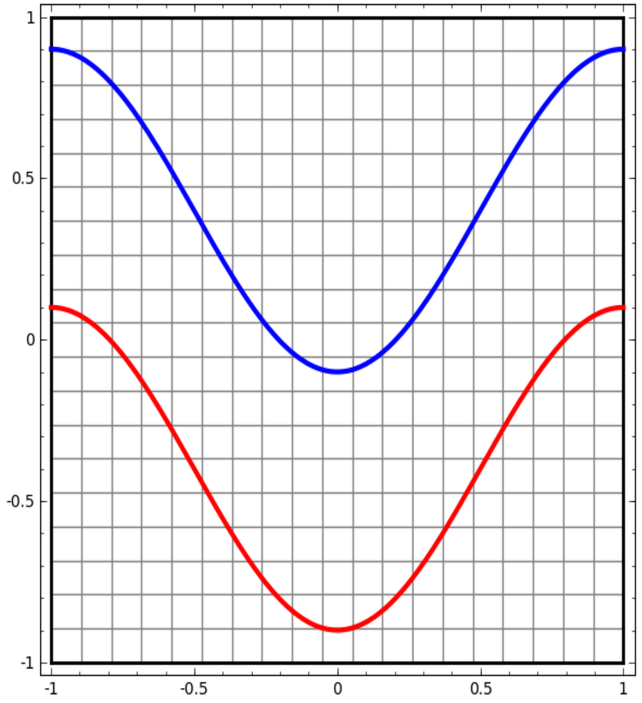
Decision function



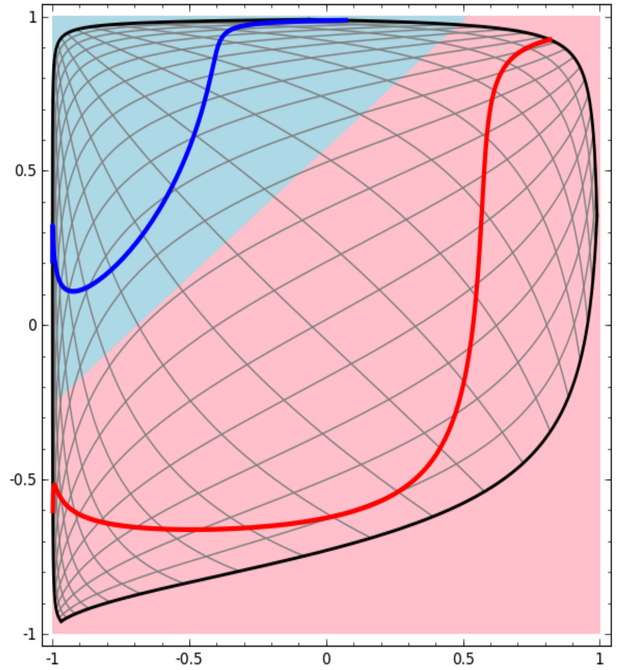
http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

From Non-separable to Linearly Separable

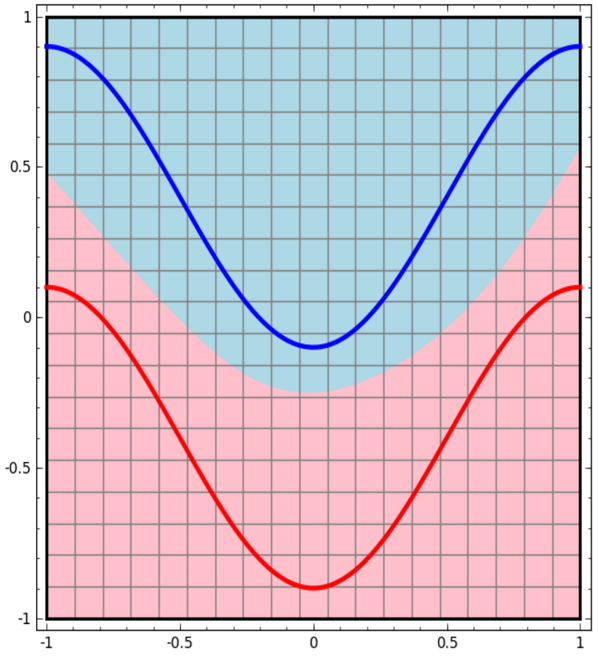
Non-linearly separable data



Data mapped to learned space



Decision function



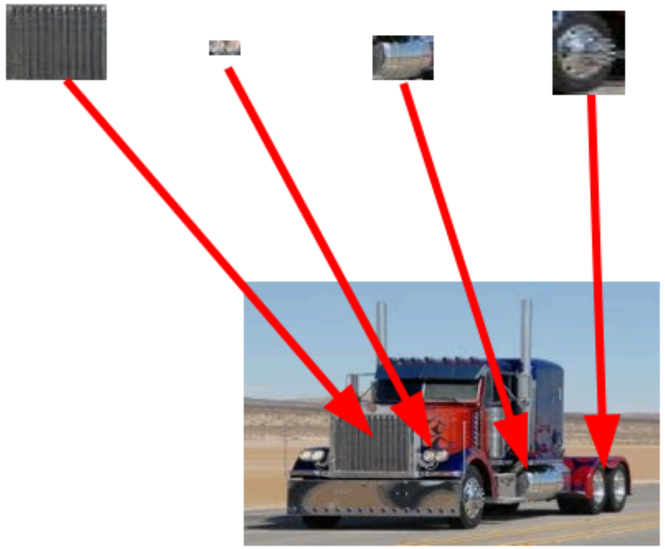
http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

Hidden Layers: What do They Do?

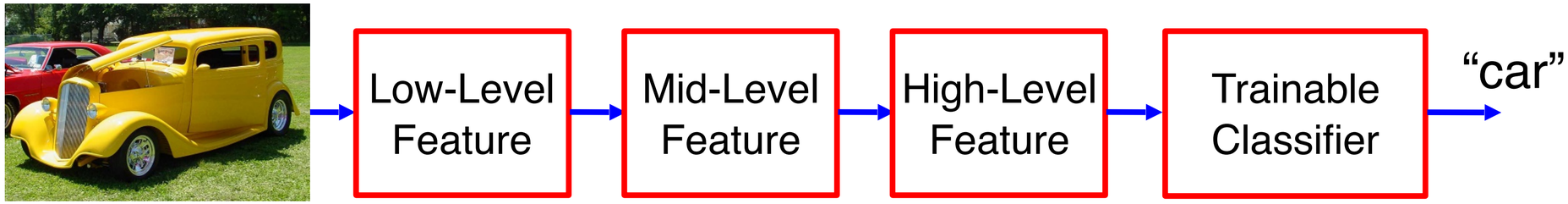
Intuition: learn “dictionary” for objects

“Distributed representation”:
represent (and classify) objects by mixing & mashing reusable parts

[0 0 **1** 0 0 0 0 **1** 0 0 **1** **1** 0 0 **1** 0 ...] truck feature



Deep Learning ~ Hierarchical Composition

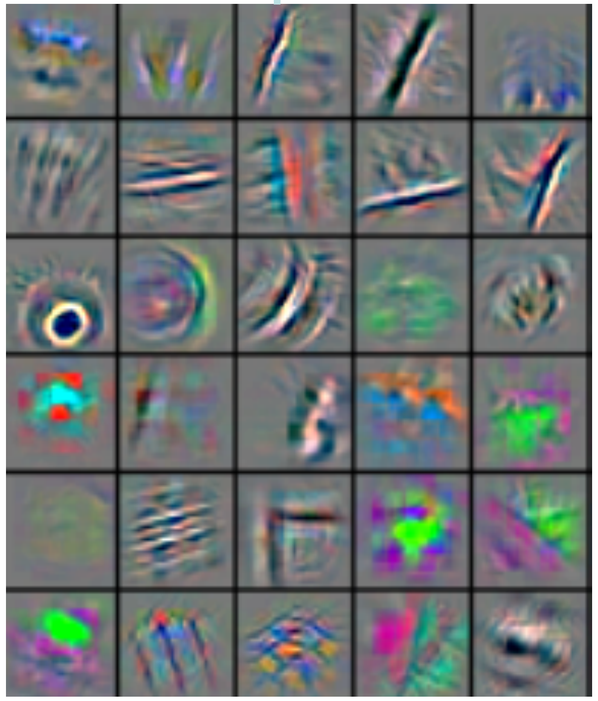


Deep Learning ~ Hierarchical Composition

Low-Level Feature → Mid-Level Feature → High-Level Feature → Trainable Classifier → “car”

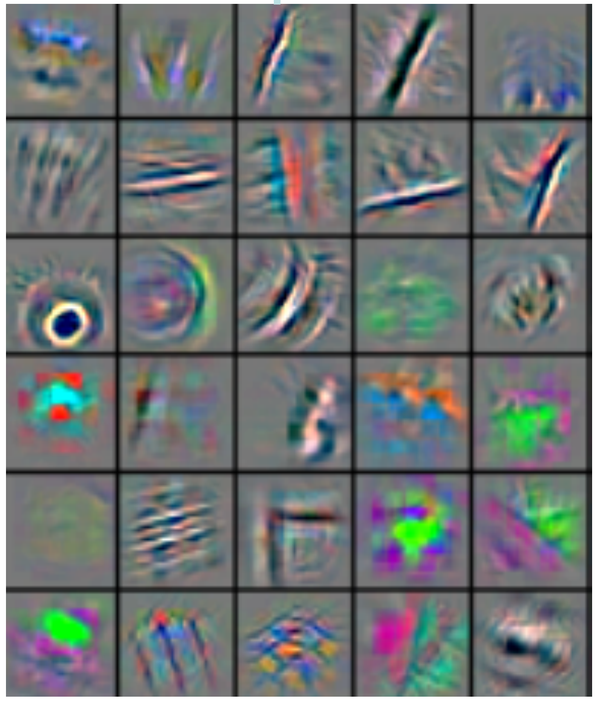
Deep Learning ~ Hierarchical Composition

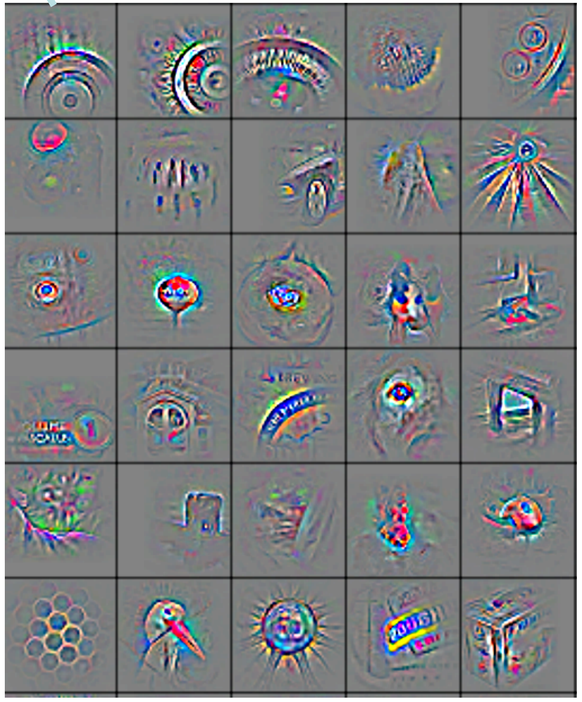
Low-Level Feature → Mid-Level Feature → High-Level Feature → Trainable Classifier → “car”



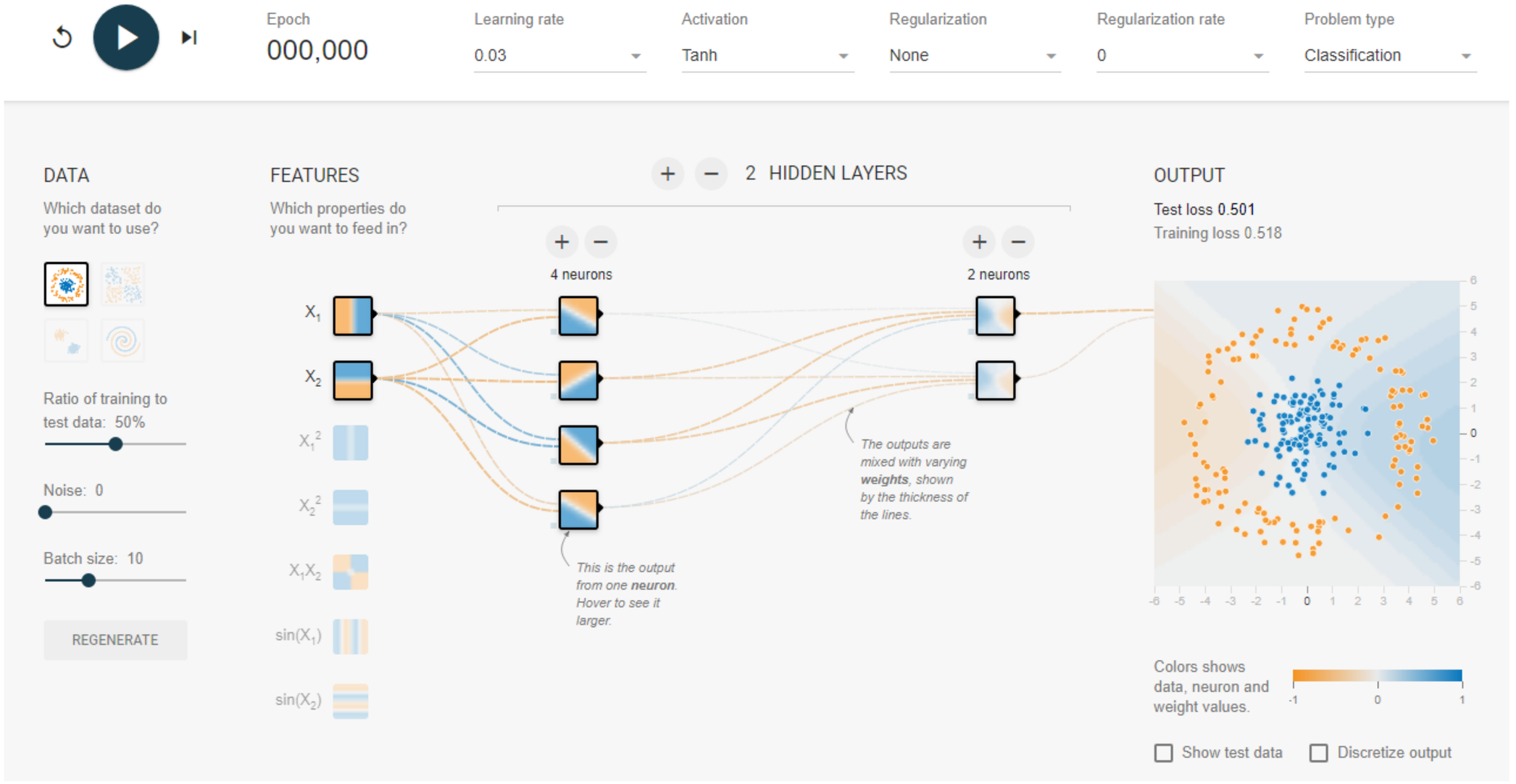
Deep Learning ~ Hierarchical Composition

Low-Level Feature → Mid-Level Feature → High-Level Feature → Trainable Classifier → “car”





MLP Demo: playground.tensorflow.org



Neural Network Training: Old and New Tricks

- Old
  - **Backpropagation algorithm**
  - Stochastic gradient, momentum, weight decay

- New
  - Dropout
  - Relu
  - Batch Norm(alization), GroupNorm, Spectral Normalization
  - Res(idual) Net(work)

Training Goal

Our network implements a parametric function:

$$f_\theta : \mathbb{X} \longrightarrow \mathbb{Y} \qquad \hat{y} = f(x; \theta)$$

Training Goal

Our network implements a parametric function:

$$f_\theta : \mathbb{X} \longrightarrow \mathbb{Y} \qquad \hat{y} = f(x; \theta)$$

During training, we search for parameters that minimize a loss:

$$\min_\theta L_f(\theta)$$

Training Goal

Our network implements a parametric function:

$$f_\theta : \mathbb{X} \longrightarrow \mathbb{Y} \qquad \hat{y} = f(x;\theta)$$

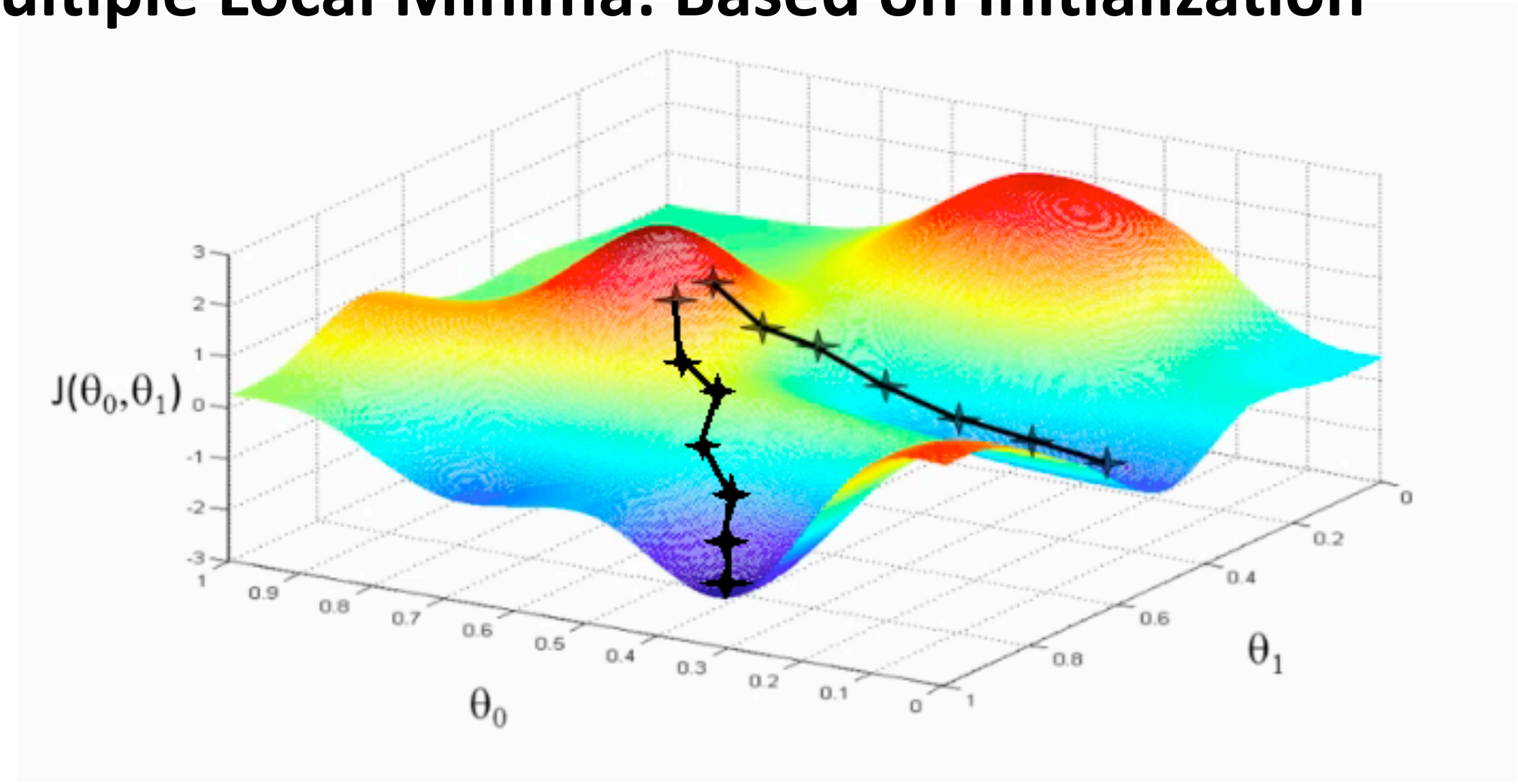
During training, we search for parameters that minimize a loss:

$$\min_\theta L_f(\theta)$$

Example: L2 regression loss given target (x^i, y^i) pairs :

$$L_f(\theta) = \sum_i \| f(x^i;\theta) - y^i \|_2^2$$

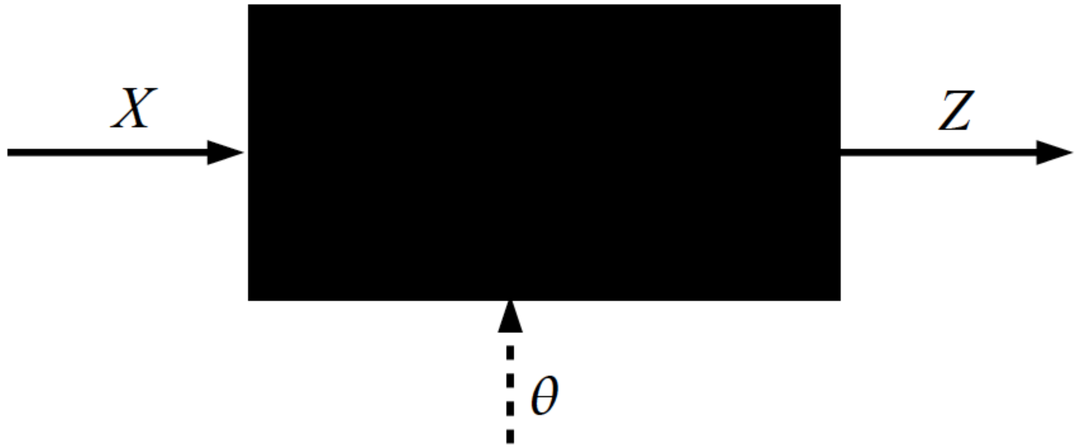
Multiple Local Minima: Based on Initialization



empirically all are almost equally good

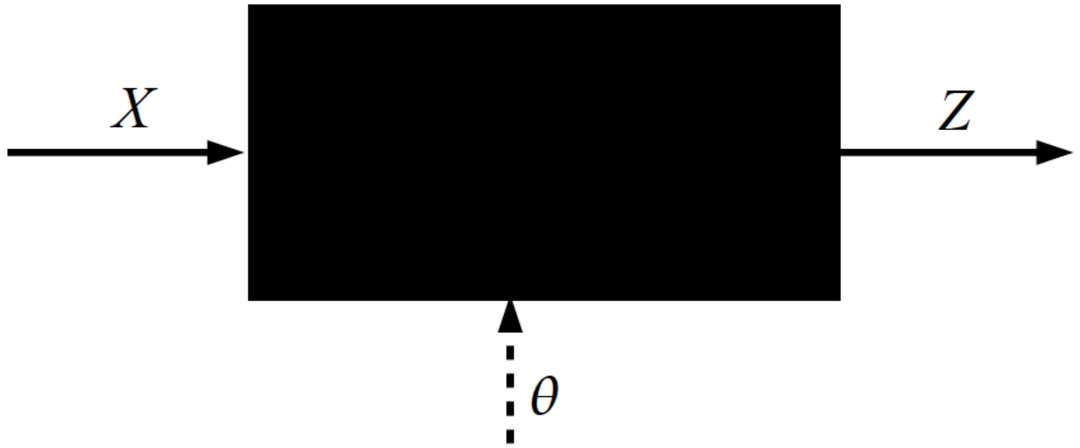
All You Need is Gradients

Forward

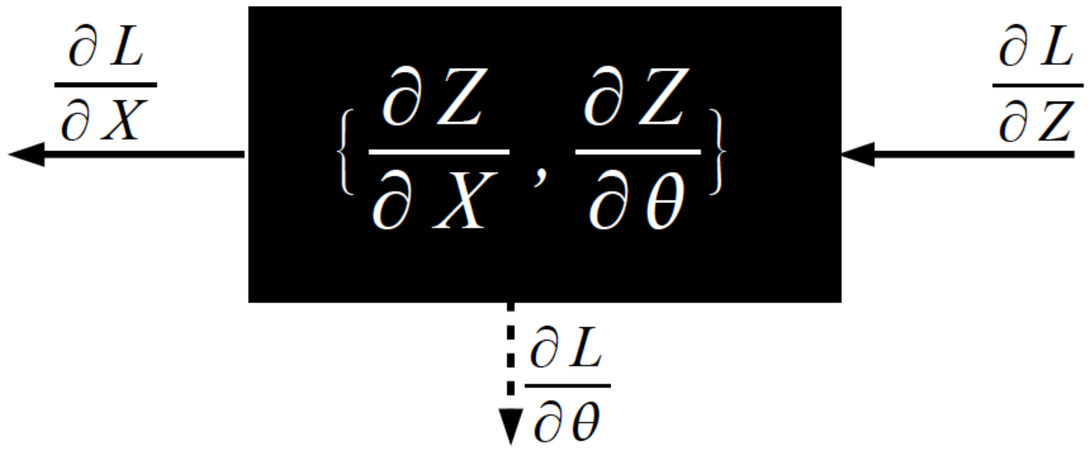


All You Need is Gradients

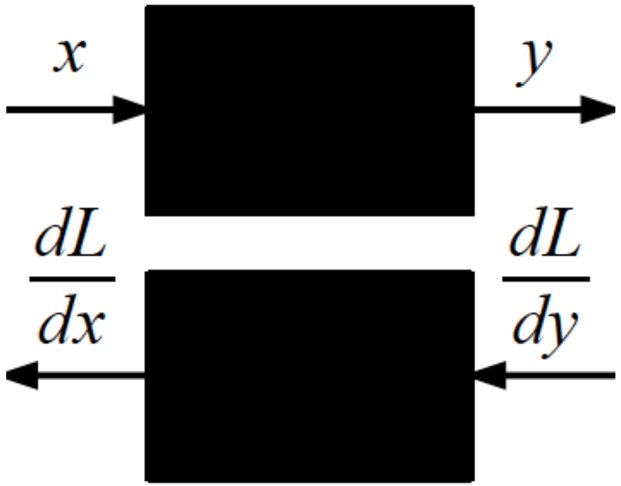
Forward



Backward

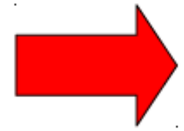


Chain Rule



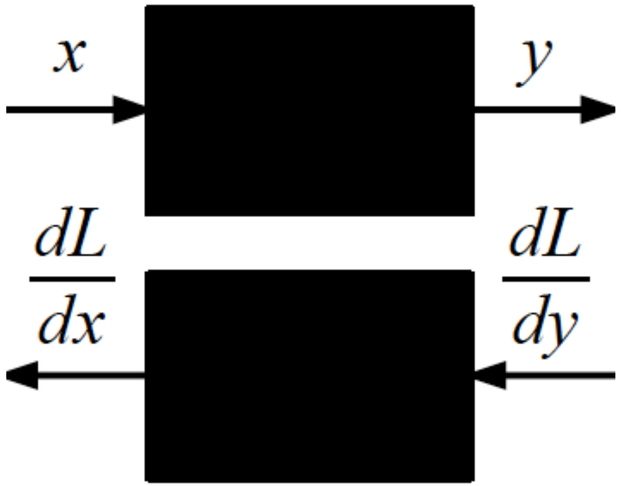
Given $y(x)$ and dL/dy,

What is dL/dx ?



Slide Credit: Marc'Aurelio Ranzato, Yann LeCun

Chain Rule



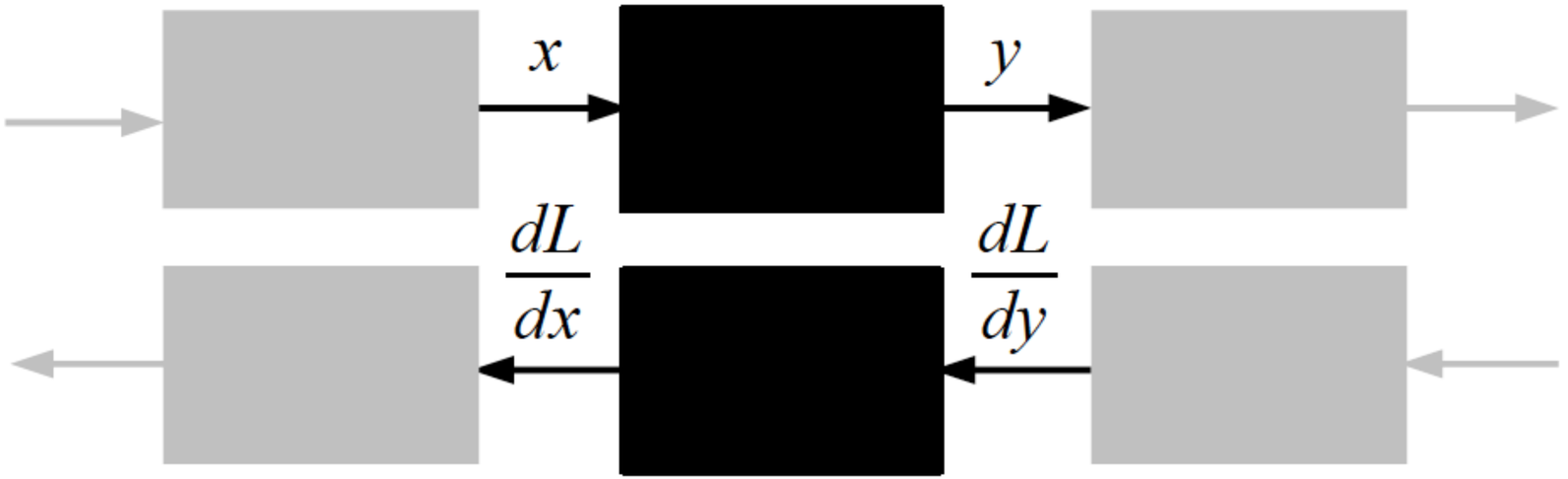
Given $y(x)$ and dL/dy,

What is dL/dx ?

$$\frac{dL}{dx} = \frac{dL}{dy} \cdot \frac{dy}{dx}$$

Slide Credit: Marc'Aurelio Ranzato, Yann LeCun

‘Another Brick in the Wall’

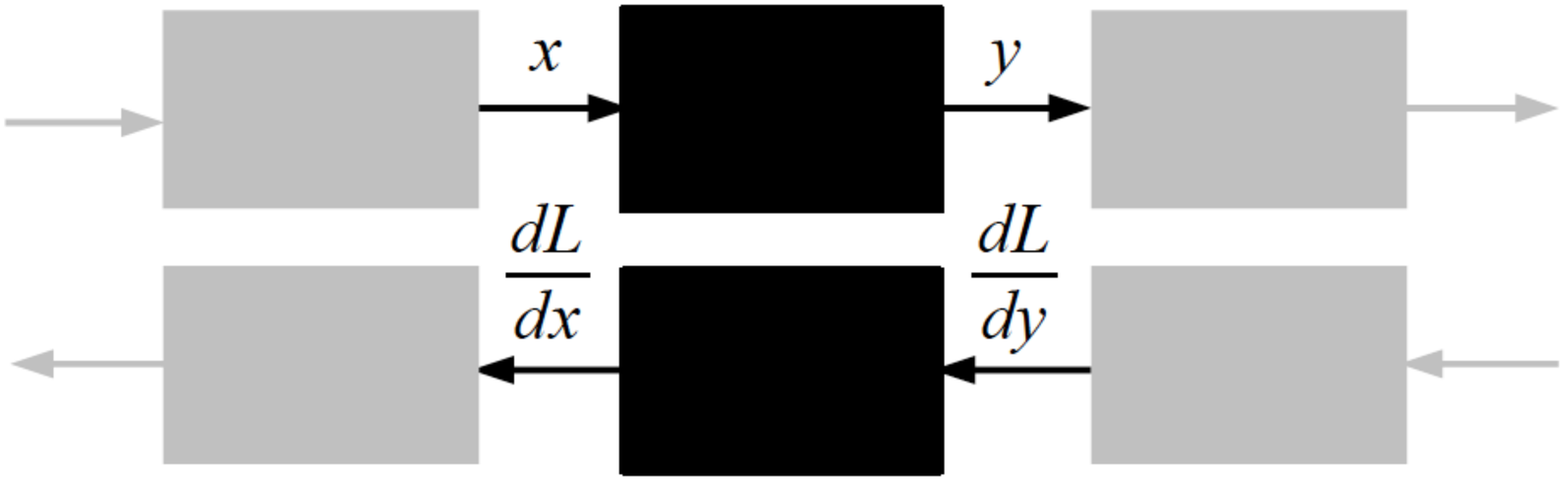


Given $y(x)$ and dL/dy,

What is dL/dx ?

$$\frac{dL}{dx} = \frac{dL}{dy} \cdot \frac{dy}{dx}$$

‘Another Brick in the Wall’

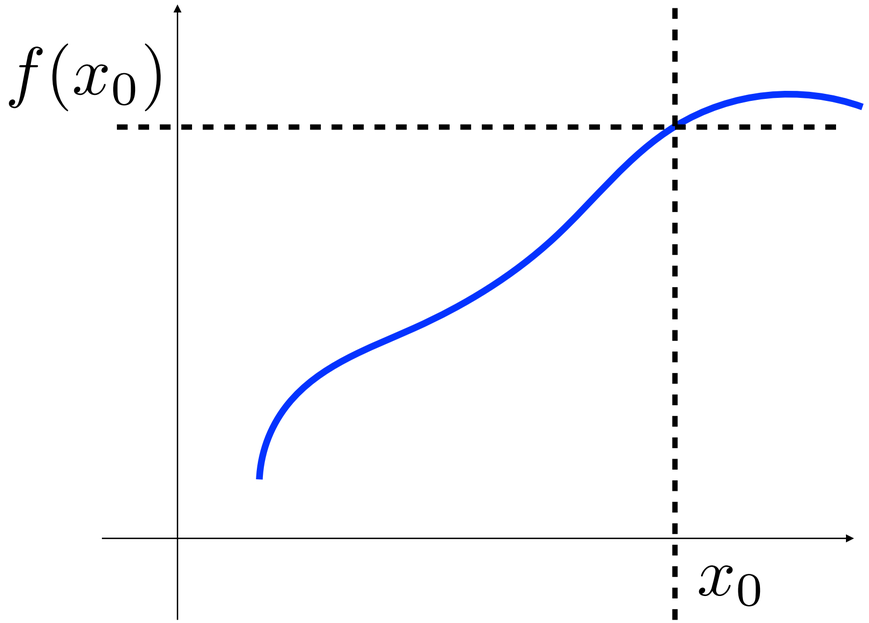


Given $y(x)$ and dL/dy,

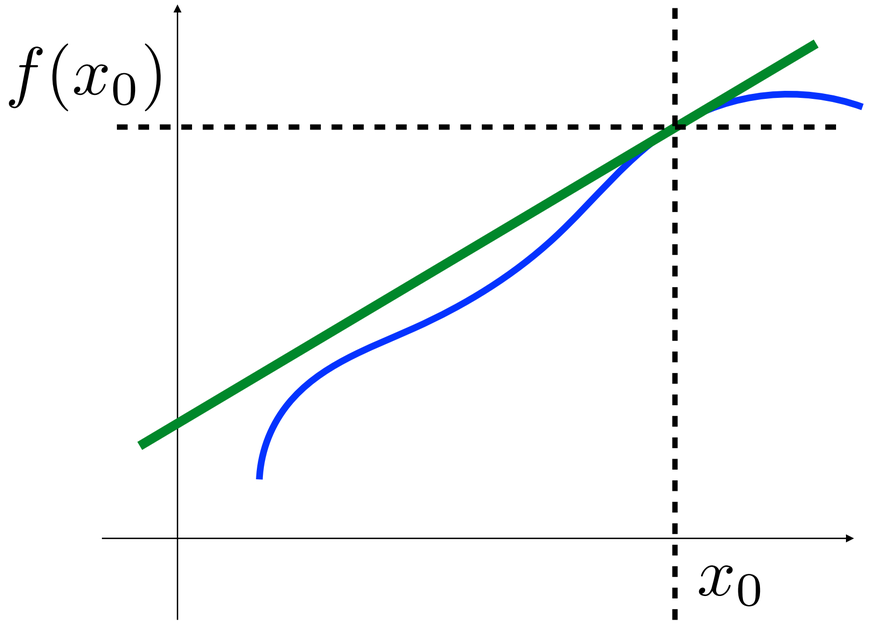
What is dL/dx ?

$$\frac{dL}{dx} = \frac{dL}{dy} \cdot \frac{dy}{dx}$$

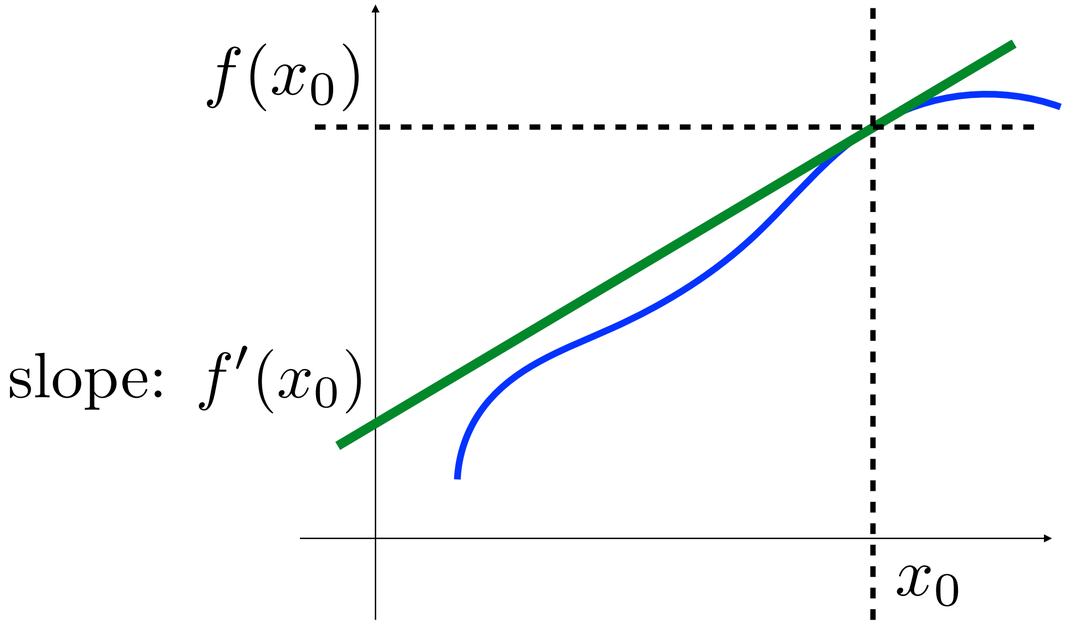
Differentiation (chain rule recap)



Differentiation (chain rule recap)

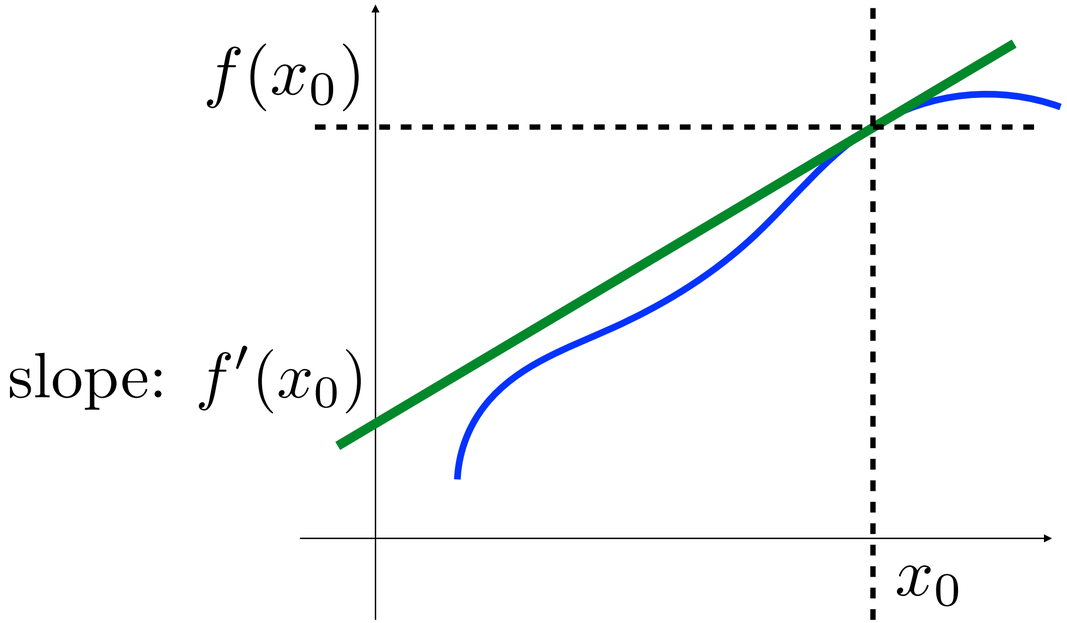


Differentiation (chain rule recap)



Differentiation (chain rule recap)

$$z = f \circ g(x) = f(g(x))$$

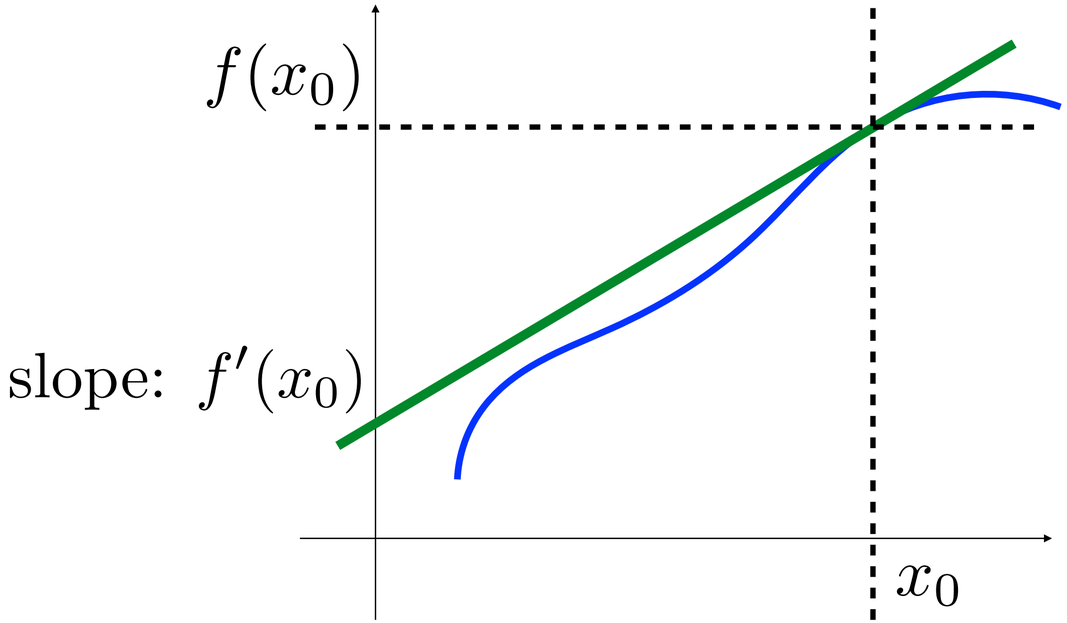


Differentiation (chain rule recap)

$$z = f(y)$$

$$y = g(x)$$

$$z = f \circ g(x) = f(g(x))$$



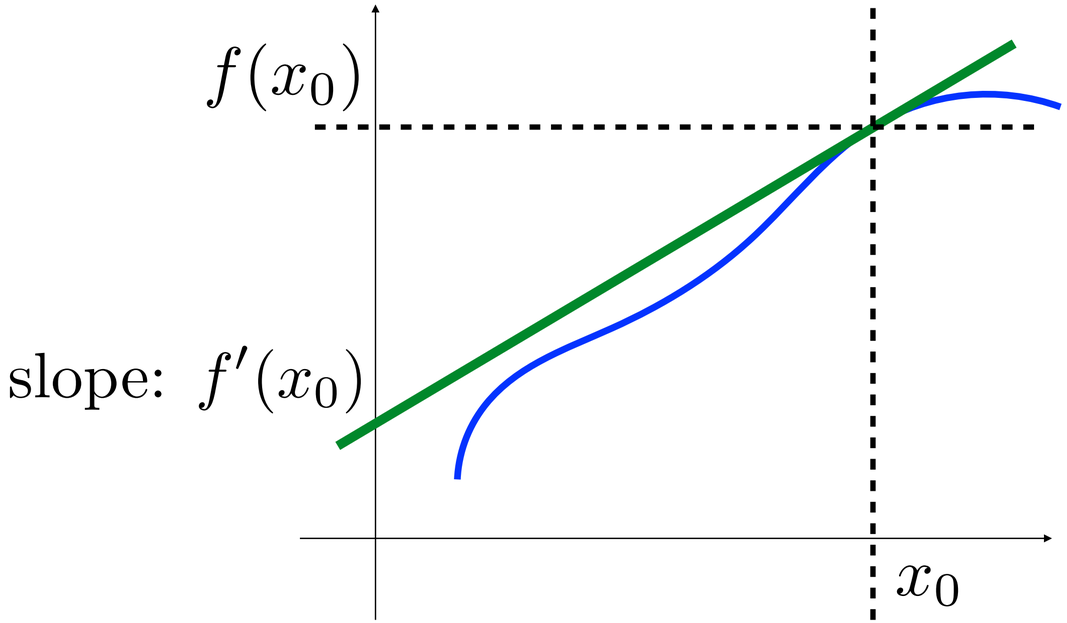
Differentiation (chain rule recap)

$$z = f(y)$$

$$y = g(x)$$

$$z = f \circ g(x) = f(g(x))$$

$$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}$$



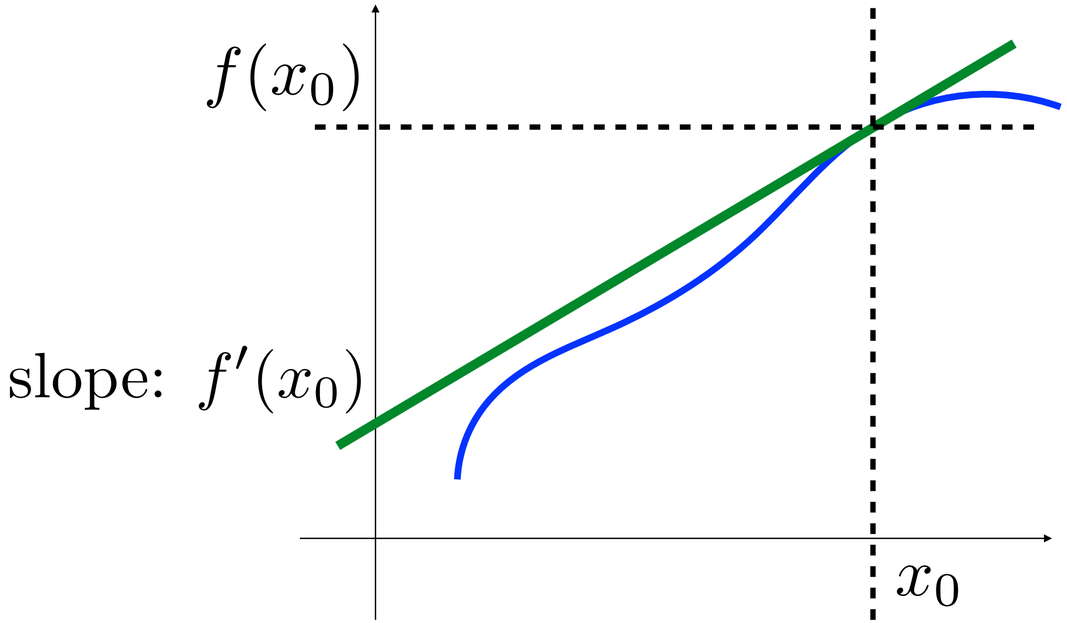
Differentiation (chain rule recap)

$$z = f(y)$$

$$y = g(x)$$

$$z = f \circ g(x) = f(g(x))$$

$$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx} = f'(y)g'(x)$$



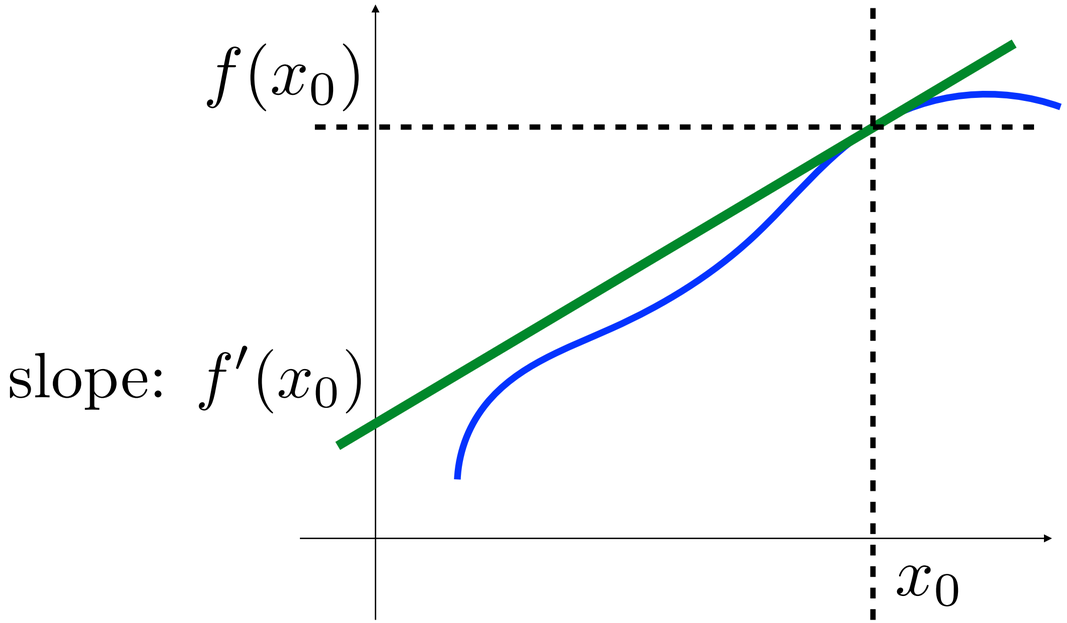
Differentiation (chain rule recap)

$$z = f(y)$$

$$y = g(x)$$

$$z = f \circ g(x) = f(g(x))$$

$$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx} = f'(y)g'(x) = f'(g(x))g'(x)$$



Differentiation (chain rule recap)

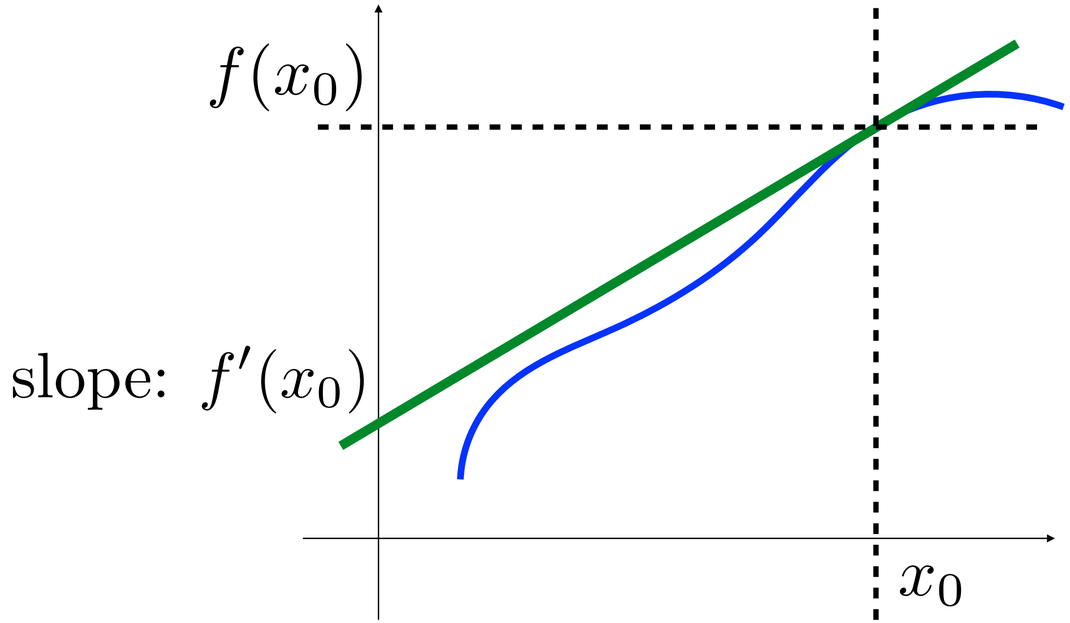
$$z = f(y)$$

$$y = g(x)$$

$$z = f \circ g(x) = f(g(x))$$

$$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx} = f'(y)g'(x) = f'(g(x))g'(x)$$

$z = \sin(5x)$



Differentiation (chain rule recap)

$$z = f(y)$$

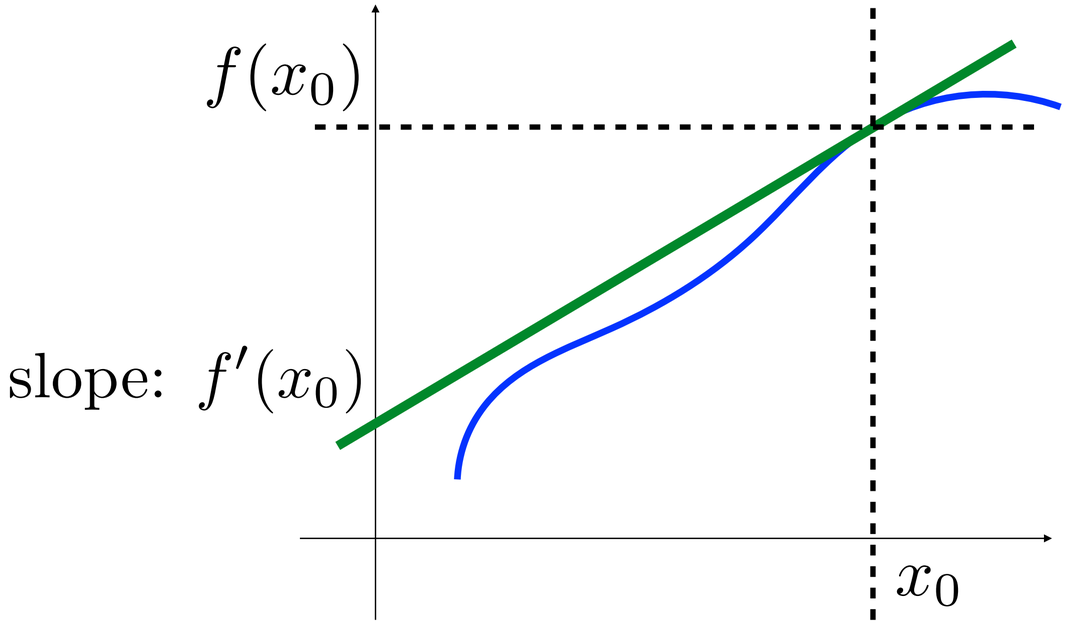
$$y = g(x)$$

$$z = f \circ g(x) = f(g(x))$$

$$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx} = f'(y)g'(x) = f'(g(x))g'(x)$$

$$z = \sin(5x)$$

$$= \frac{d\sin(5x)}{d(5x)} \frac{d(5x)}{dx}$$



Differentiation (chain rule recap)

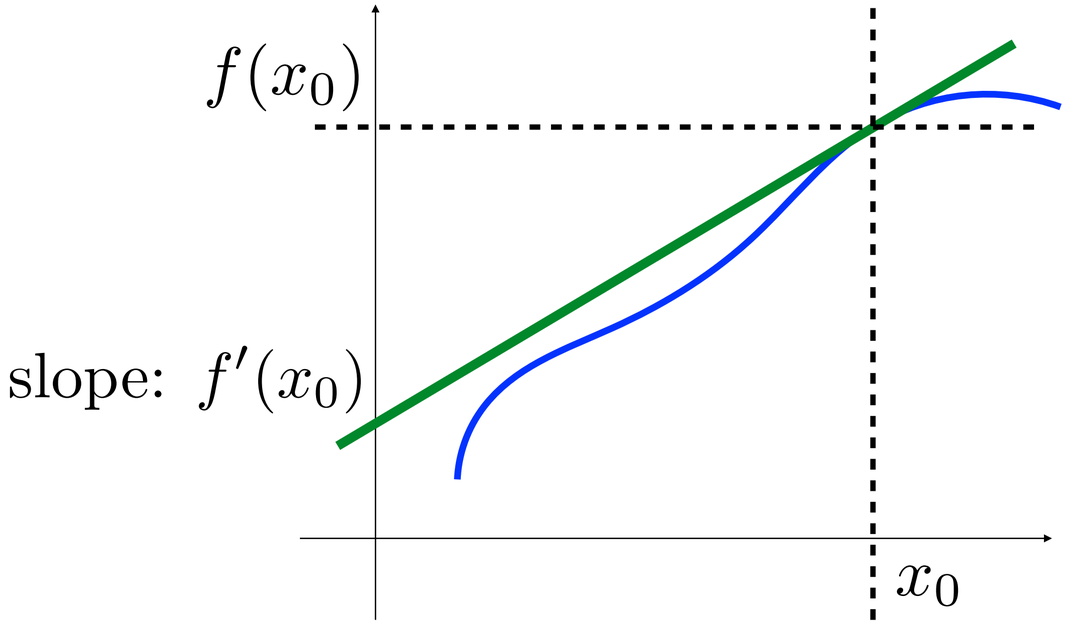
$$z = f(y)$$

$$y = g(x)$$

$$z = f \circ g(x) = f(g(x))$$

$$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx} = f'(y)g'(x) = f'(g(x))g'(x)$$

$$\begin{aligned} z &= \sin(5x) \\ &= \frac{d\sin(5x)}{d(5x)} \frac{d(5x)}{dx} \\ &= 5\cos(5x) \end{aligned}$$



Toy Example: Single Sigmoidal Unit

$$f(w, x) = \frac{1}{1 + \exp(-(w_0 x_0 + w_1 x_1 + w_2))}$$

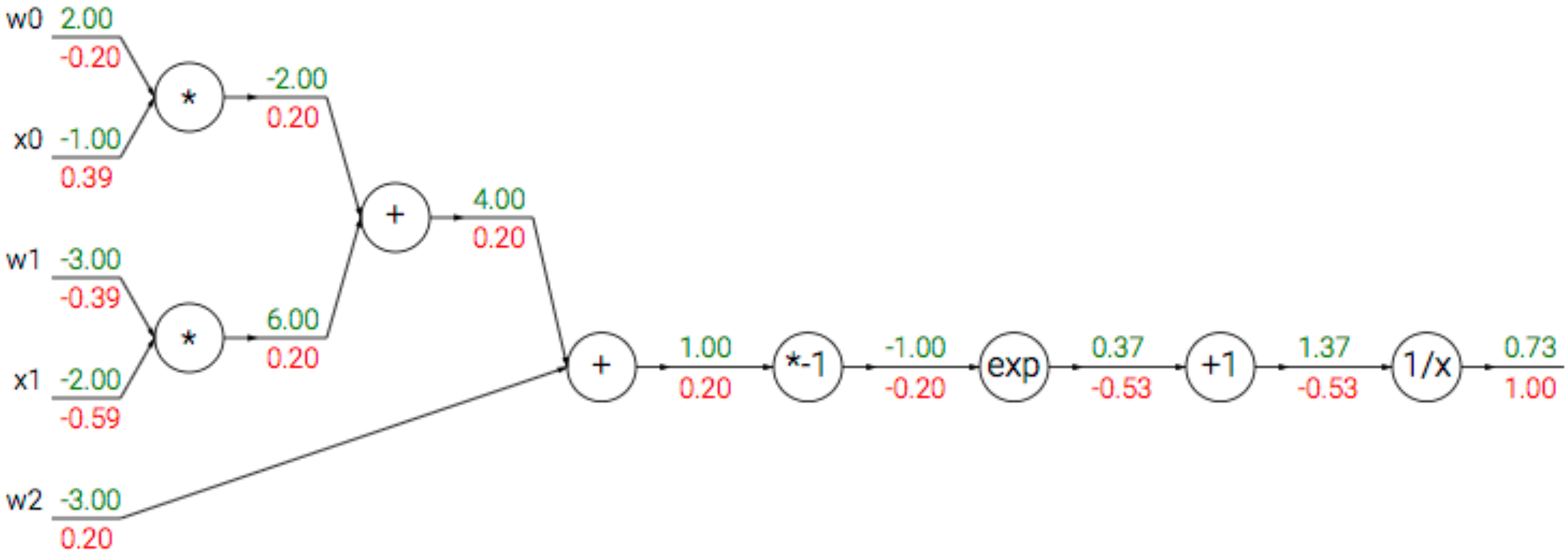
Toy Example: Single Sigmoidal Unit

$$f(w,x) = \frac{1}{1+\exp(-(w_0x_0 + w_1x_1 + w_2))}$$

Composition of differentiable blocks:

$$\begin{aligned}
f(x) = \frac{1}{x} \quad &\rightarrow \quad f'(x) = -\frac{1}{x^2} \\
f_c(x) = c + x \quad &\rightarrow \quad f'(x) = 1 \\
f(x) = e^x \quad &\rightarrow \quad f'(x) = e^x \\
f_a(x) = ax \quad &\rightarrow \quad f'(x) = a
\end{aligned}$$

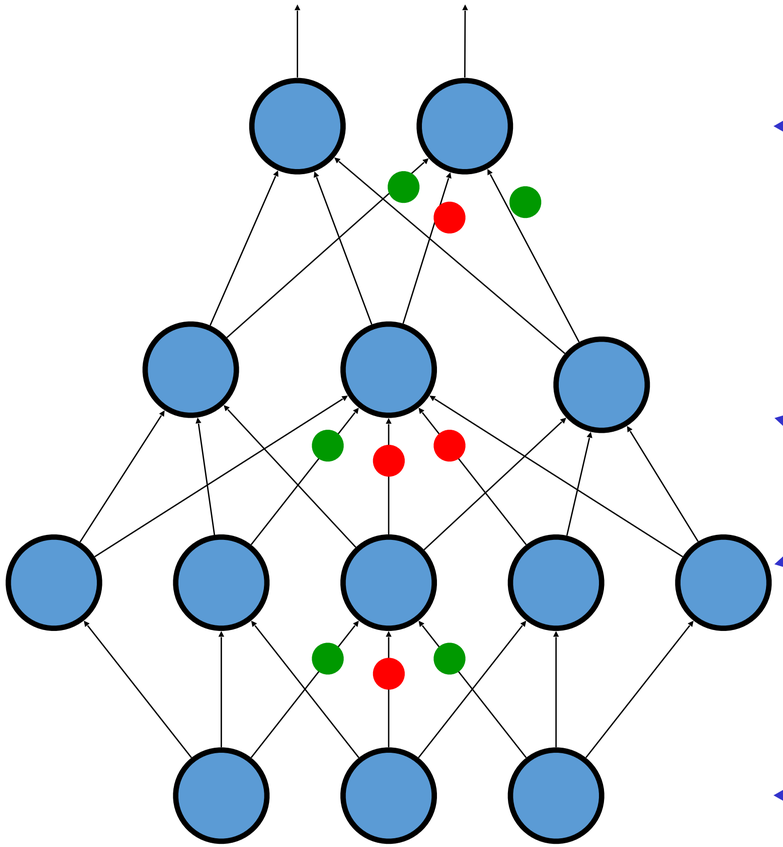
Computation Graph and Automatic Differentiation



Slide Credit: Justin Johnson

Multi-Layer Perceptrons (~1985)

$$u_i = g\left(\sum_{k\in\mathcal{N}(i)} w_{k,i}\, g\left(\sum_{m\in\mathcal{N}(k)} w_{m,k}u_m + b_k\right) + b_i\right)$$



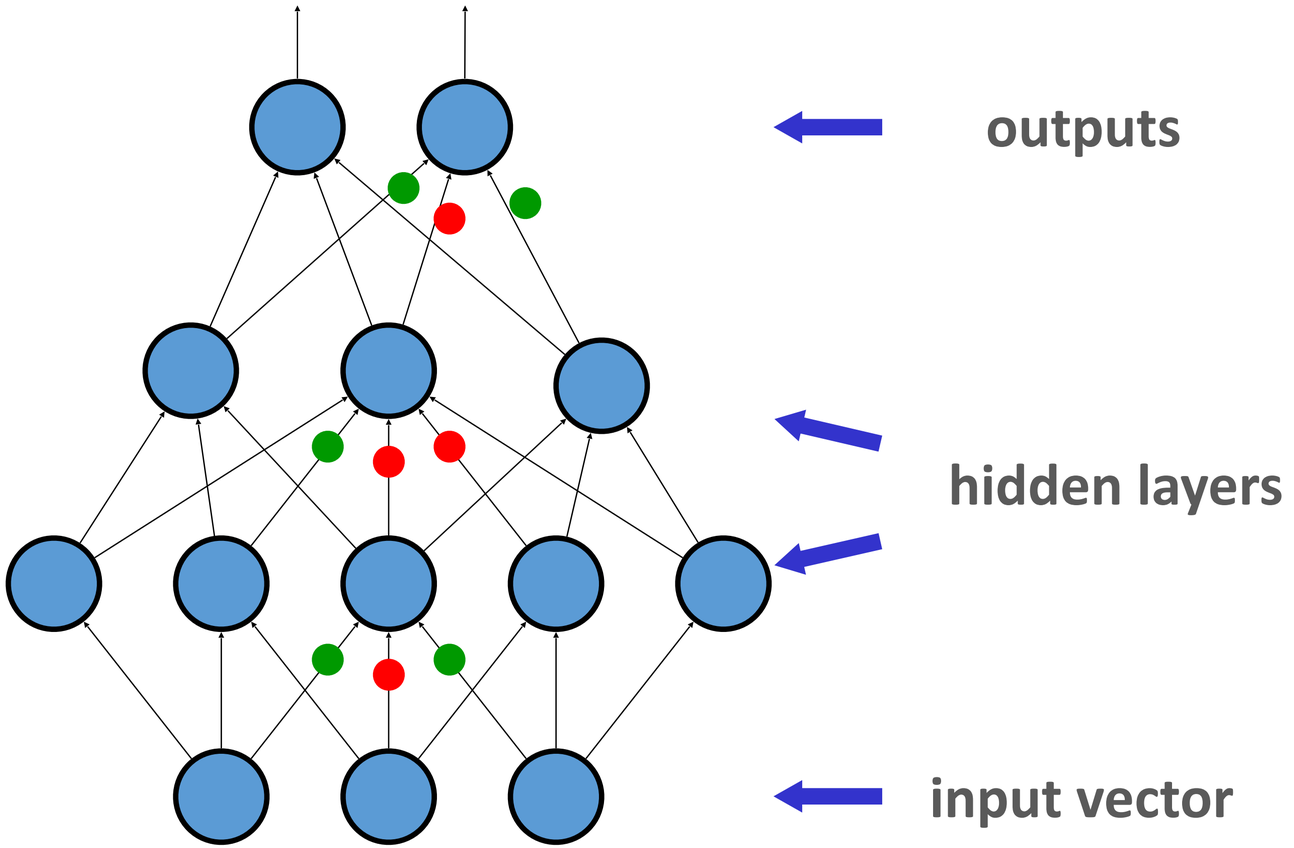
outputs

hidden layers

input vector

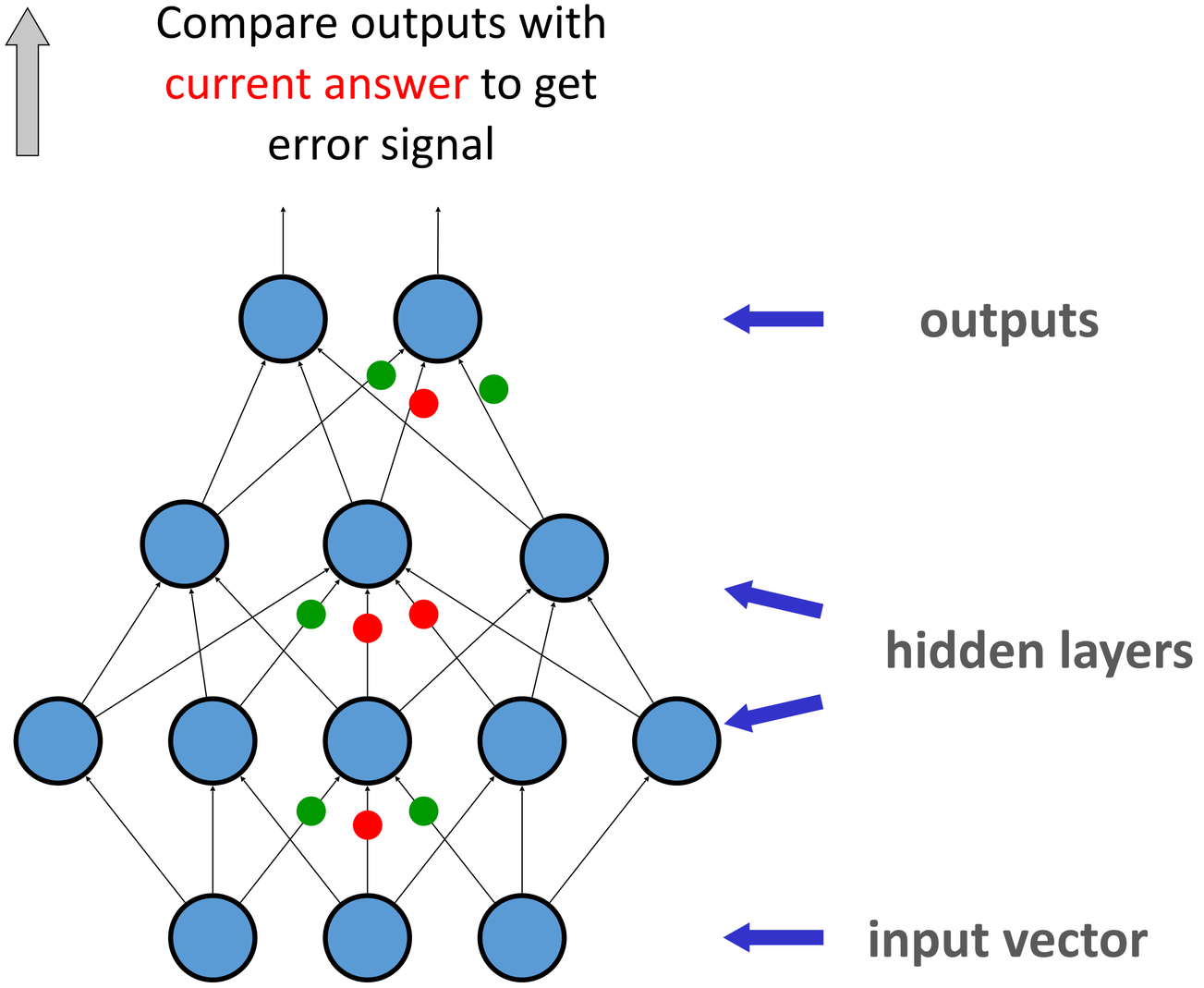
Slide credit: G. Hinton

Multi-Layer Perceptrons (~1985)



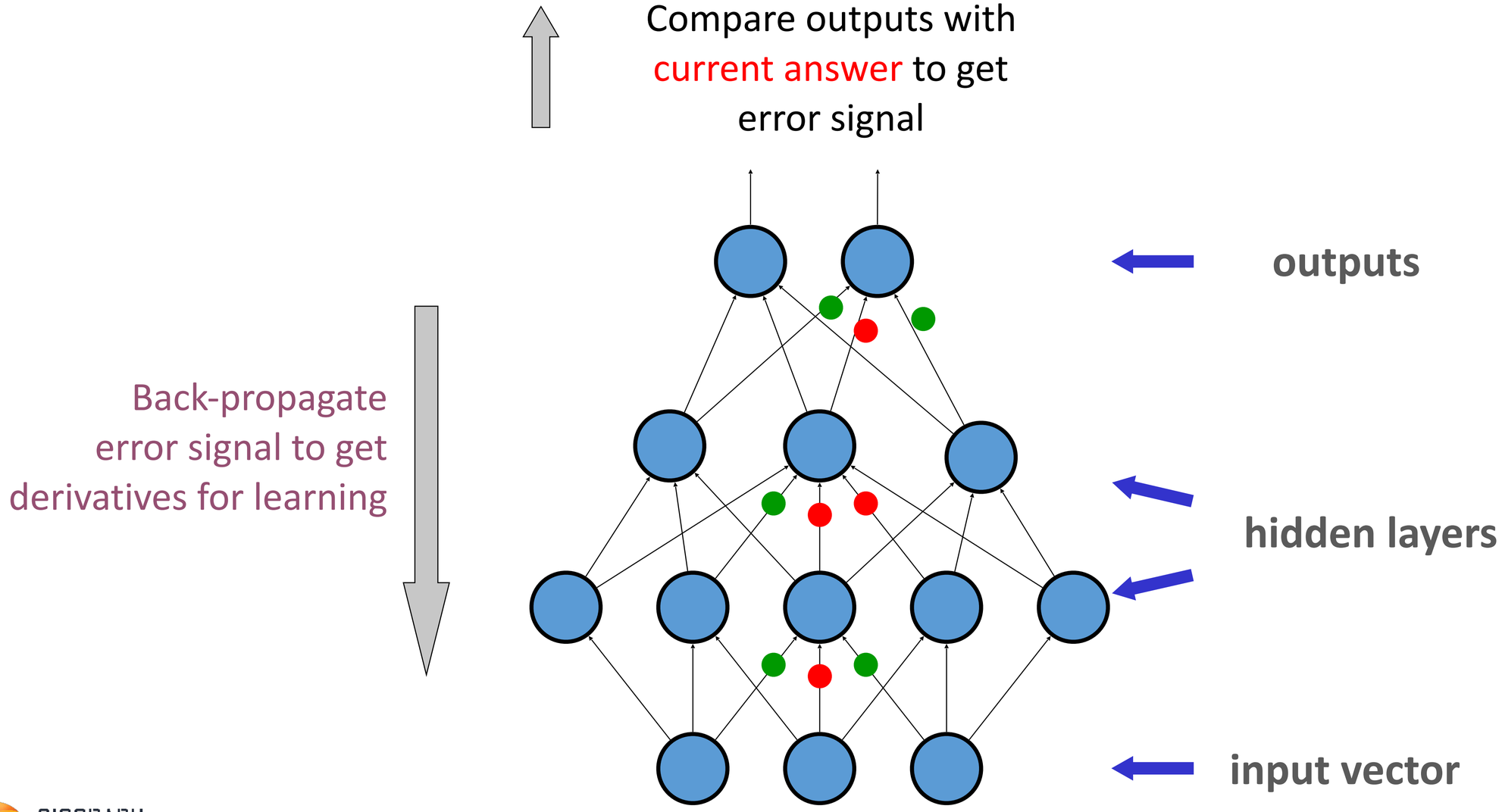
Slide credit: G. Hinton

Multi-Layer Perceptrons (~1985)



Slide credit: G. Hinton

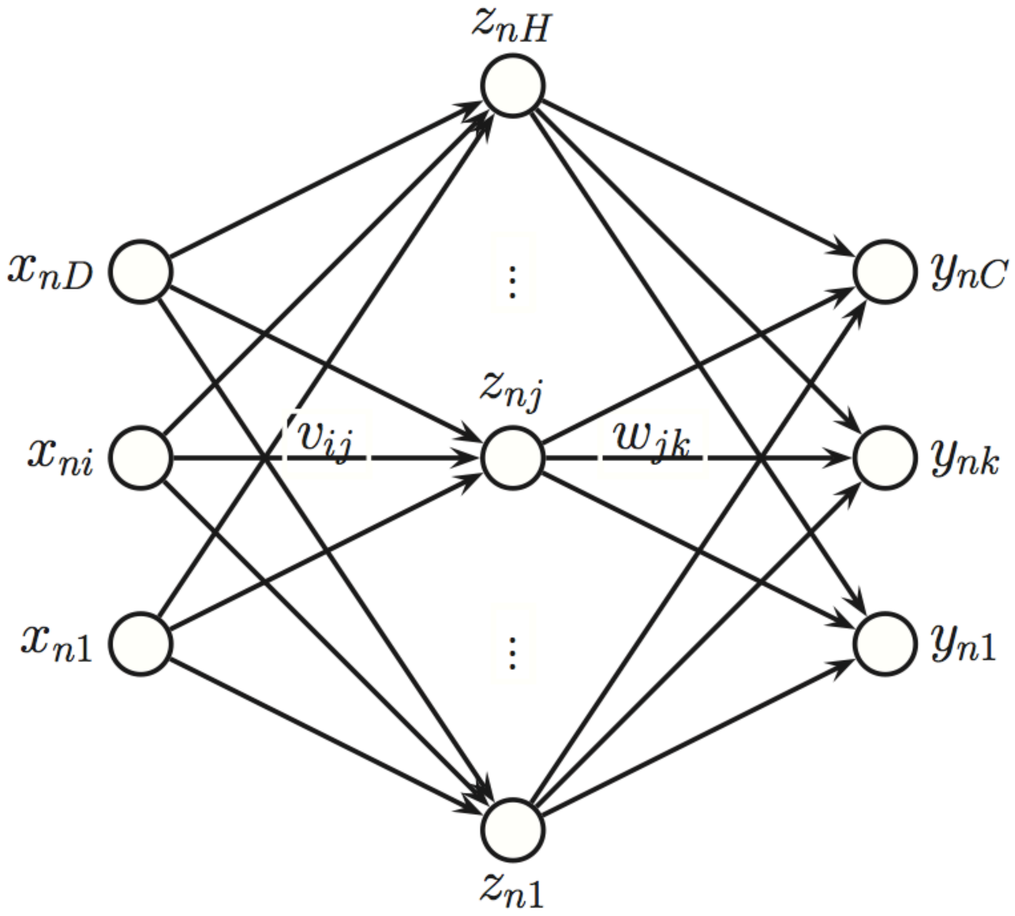
Multi-Layer Perceptrons (~1985)



Slide credit: G. Hinton

A Neural Network for Multi-way Classification

$$\mathbf{x}_n \xrightarrow{\mathbf{V}} \mathbf{a}_n \xrightarrow{g} \mathbf{z}_n \xrightarrow{\mathbf{W}} \mathbf{b}_n \xrightarrow{h} \hat{\mathbf{y}}_n$$



Inputs

Outputs

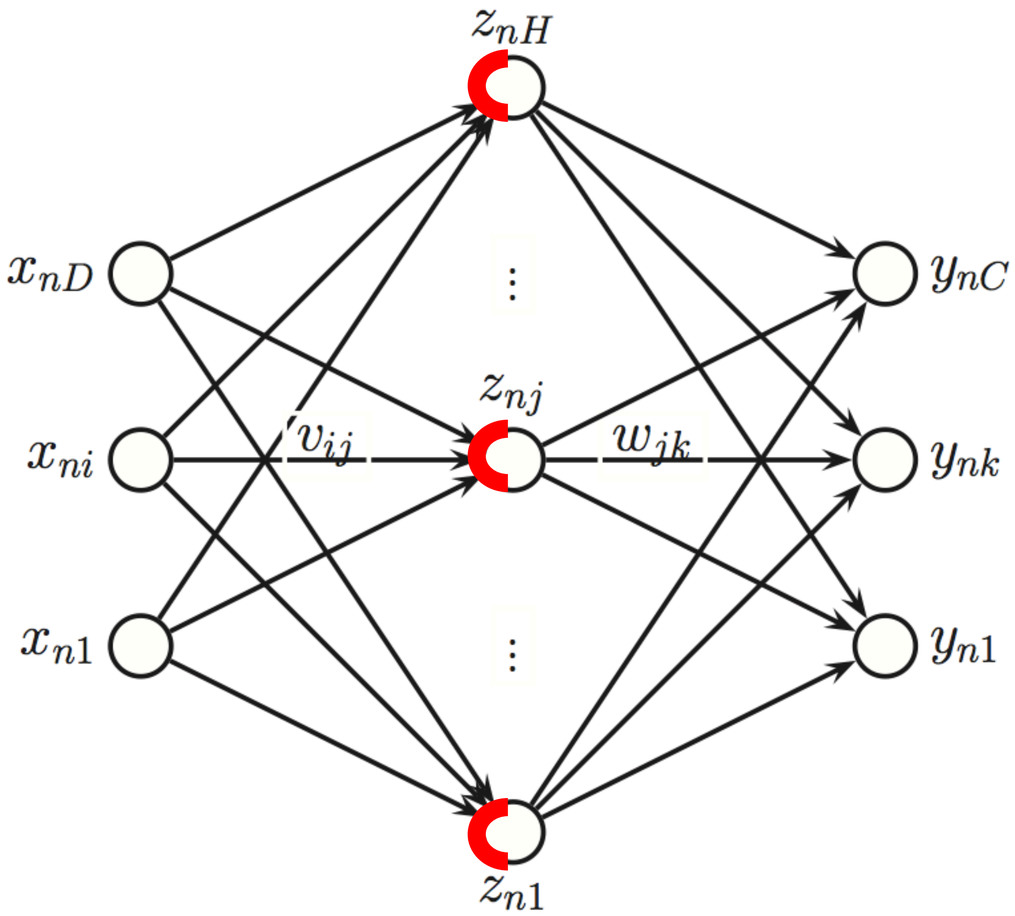
Hidden layer

Parameters:

$$\theta = \{\mathbf{v}, \mathbf{w}\}$$

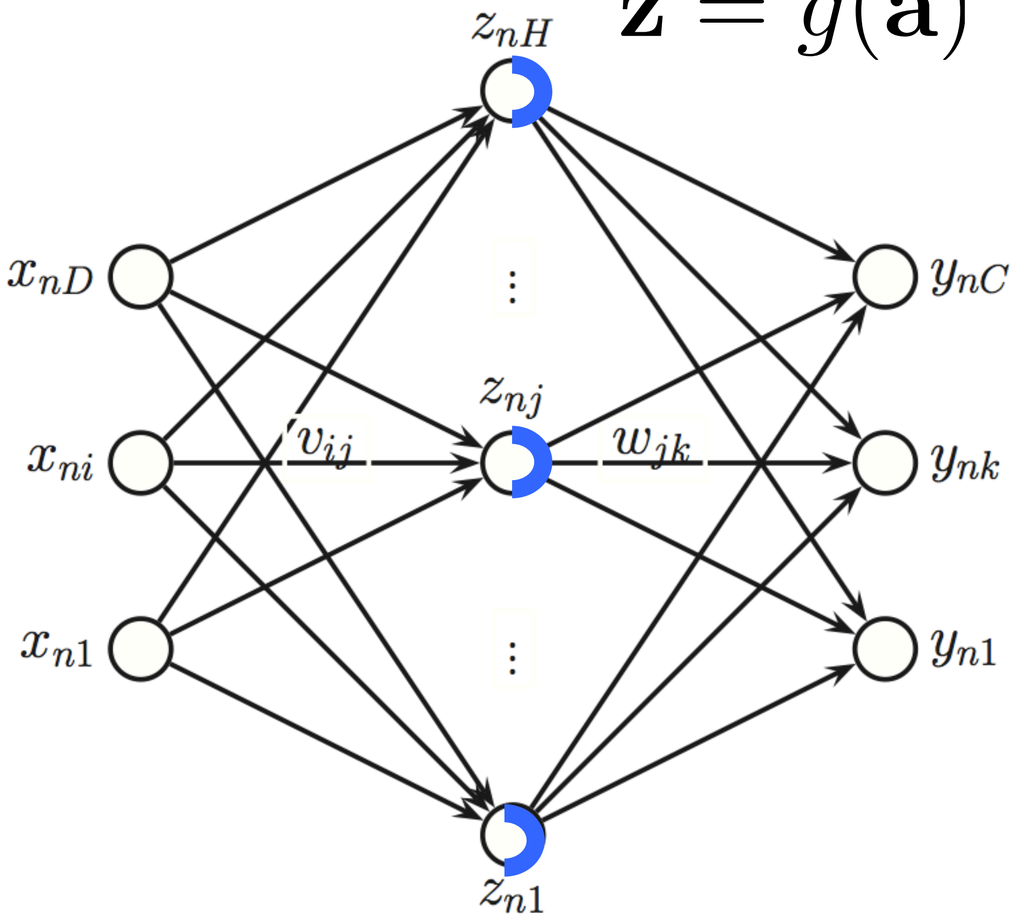
A Neural Network in Forward Mode

$$\mathbf{a} = \mathbf{V}\mathbf{x}$$

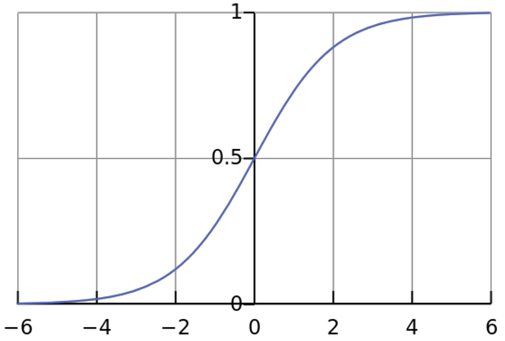


A Neural Network in Forward Mode

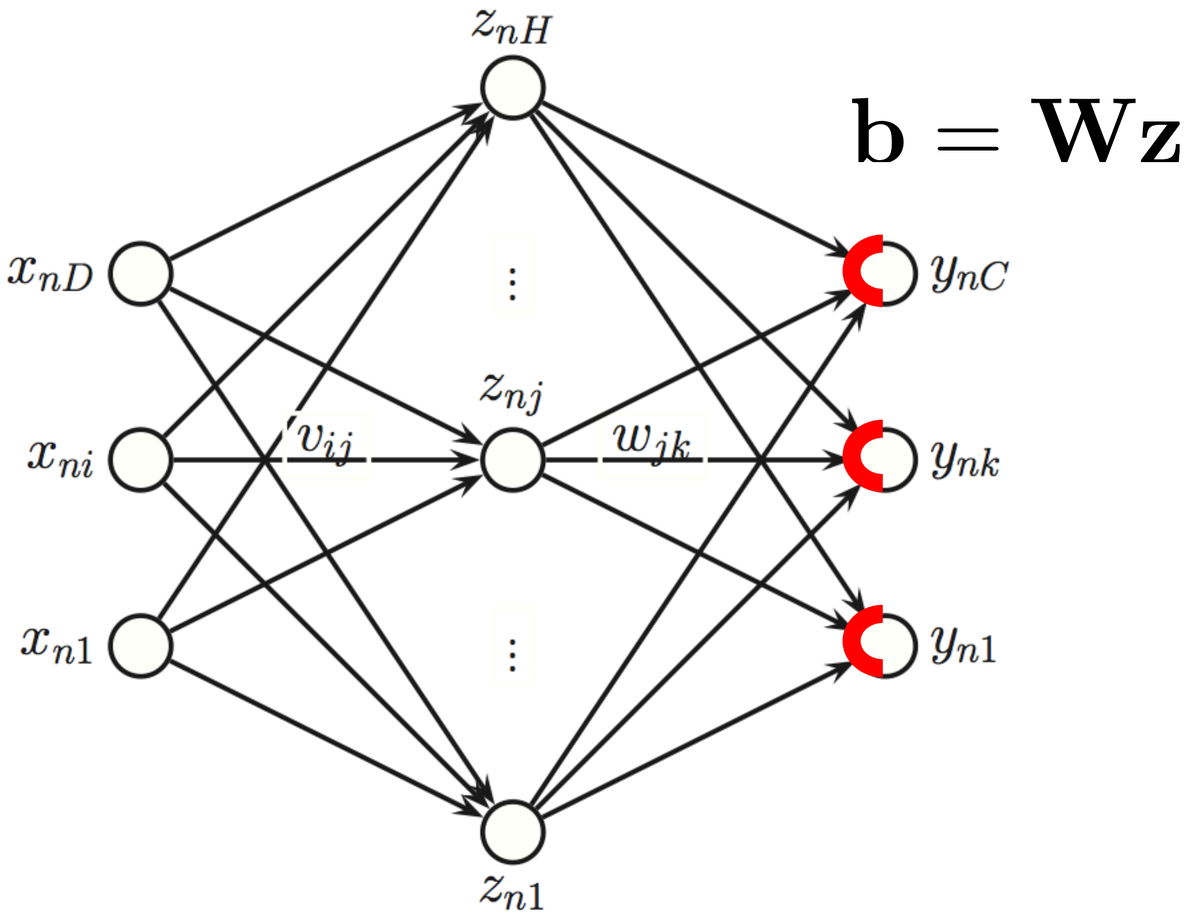
$$\mathbf{z} = g(\mathbf{a})$$



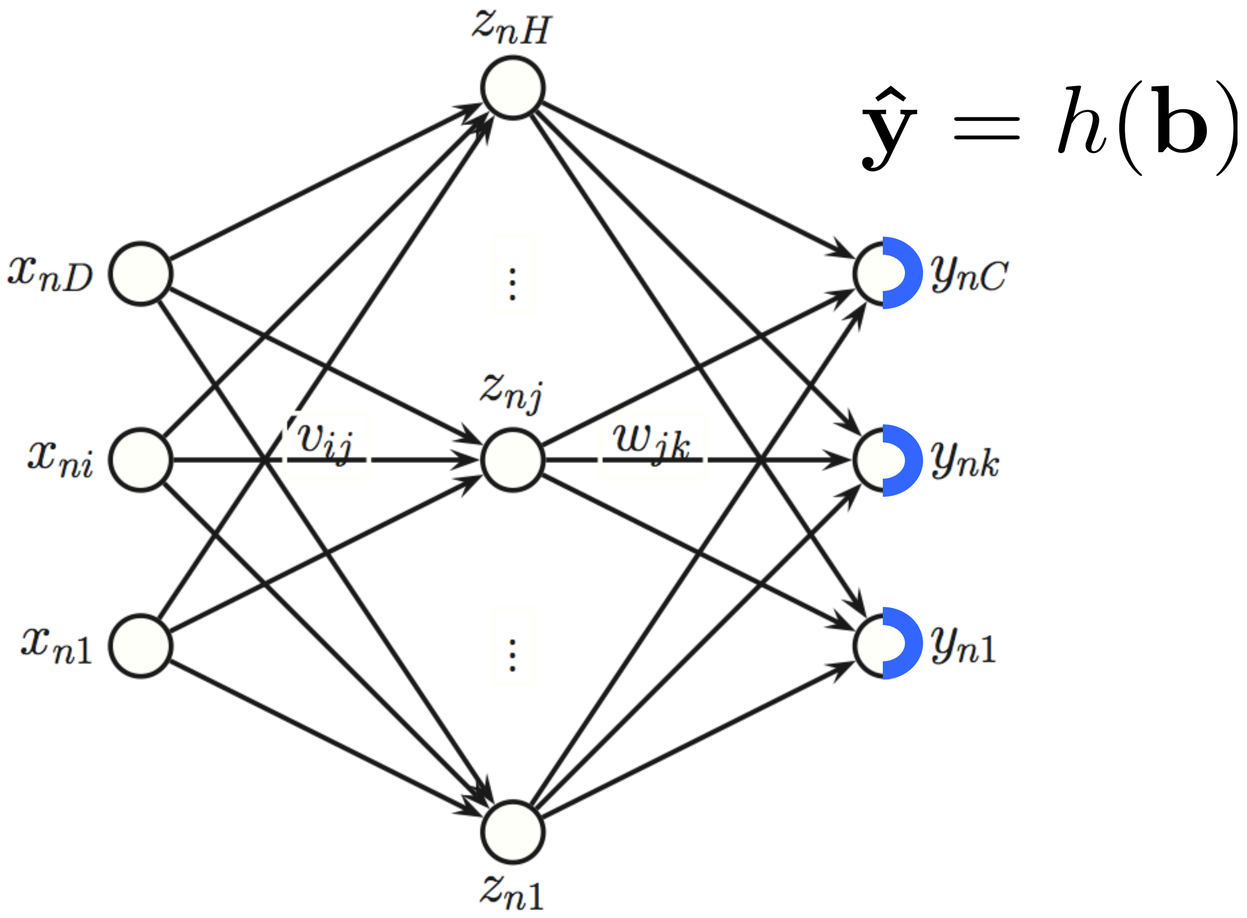
$$z_k = \frac{1}{1 + \exp(-a_k)}$$



A Neural Network in Forward Mode

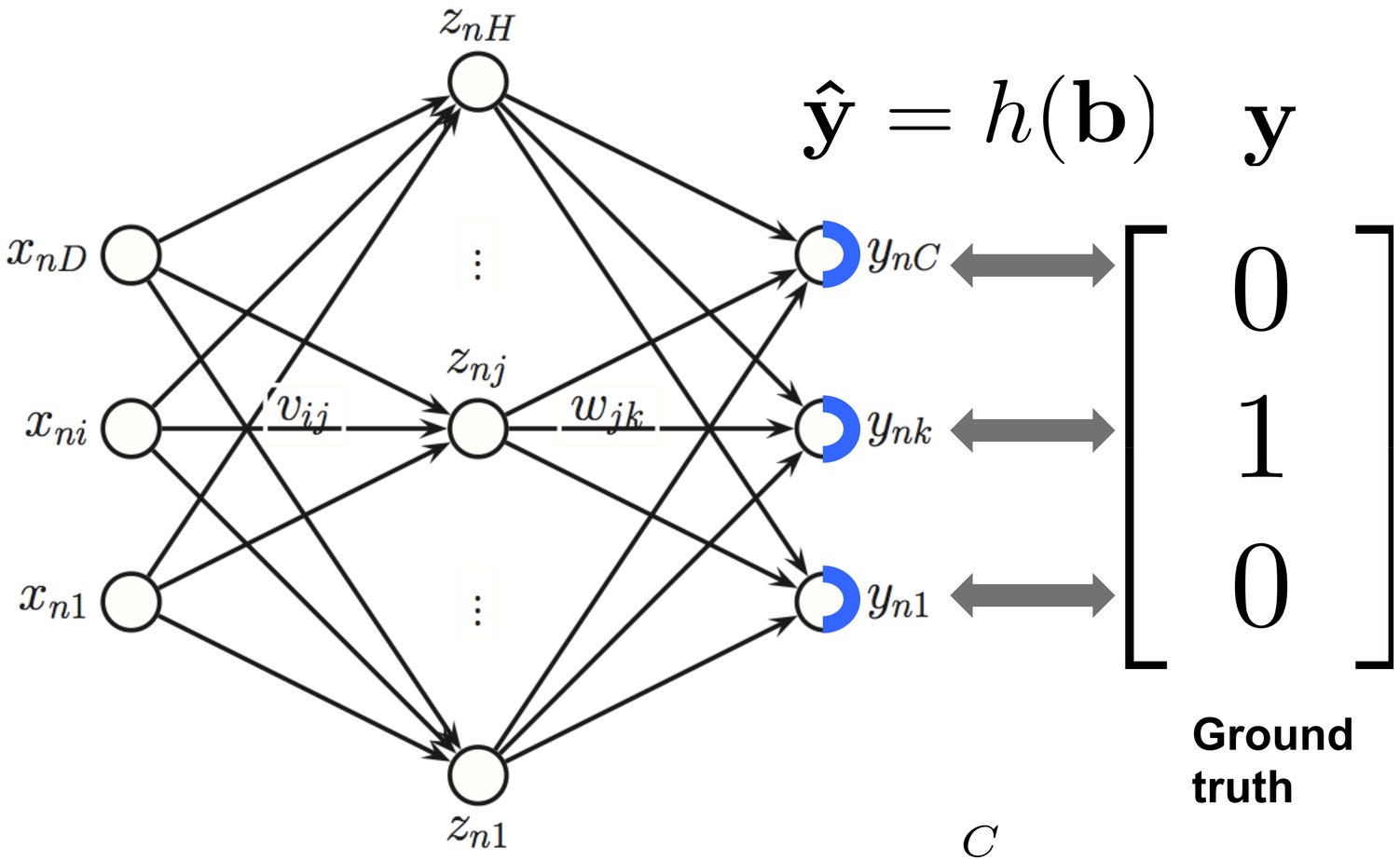


A Neural Network in Forward Mode



Objective for Linear Regression

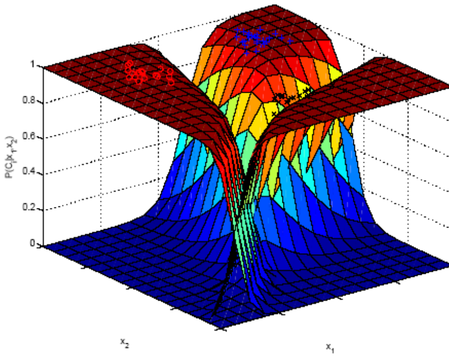
$$h(\mathbf{b}) = \mathbf{b}$$



L2 loss

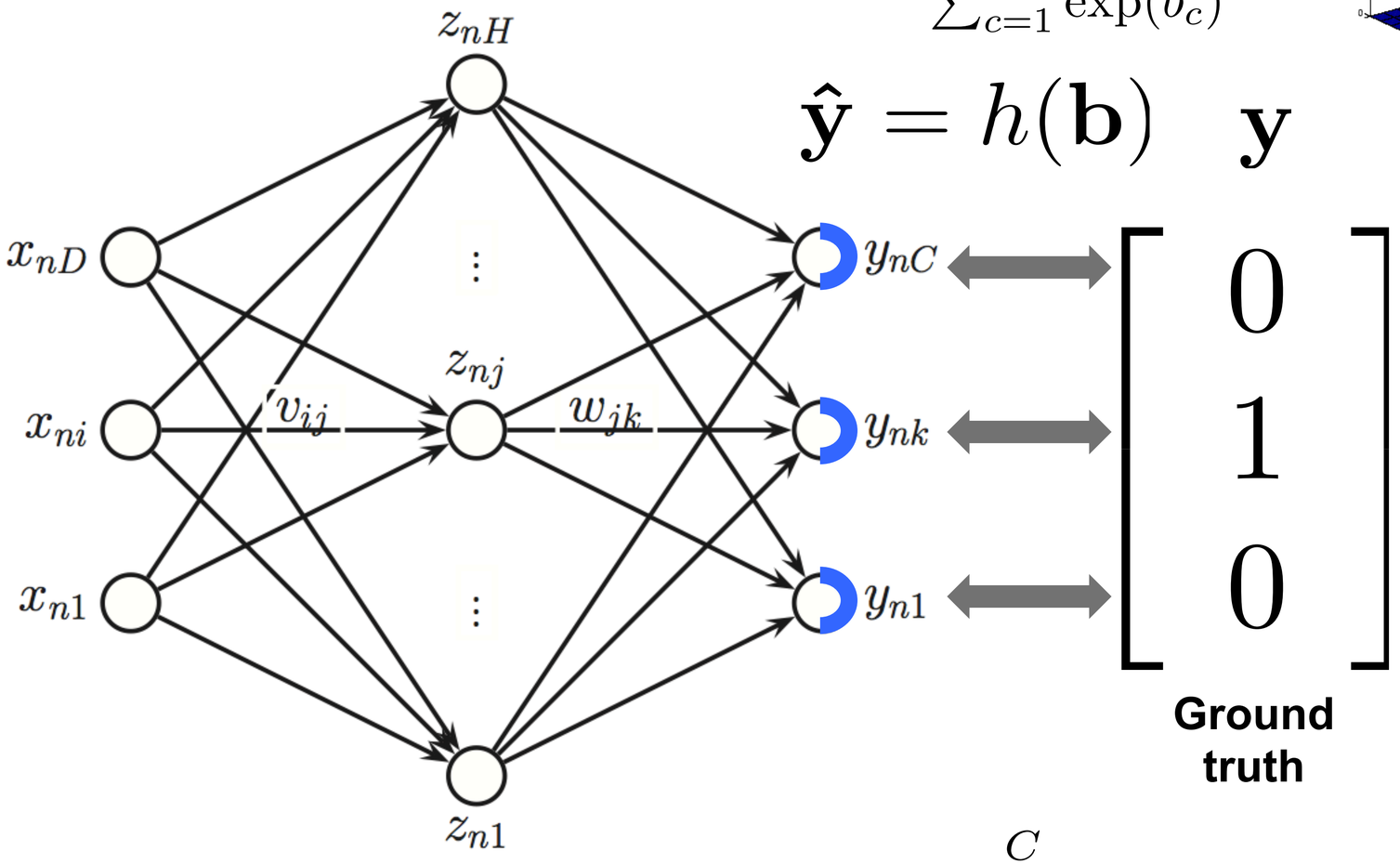
$$l(\hat{\mathbf{y}}, \mathbf{y}) = \sum_{c=1}^{C} (\mathbf{y}_c - \hat{\mathbf{y}}_c)^2$$

Objective for Multi-class Classification



Softmax unit $\hat{y}_k = \frac{\exp(b_k)}{\sum_{c=1}^{C} \exp(b_c)}$

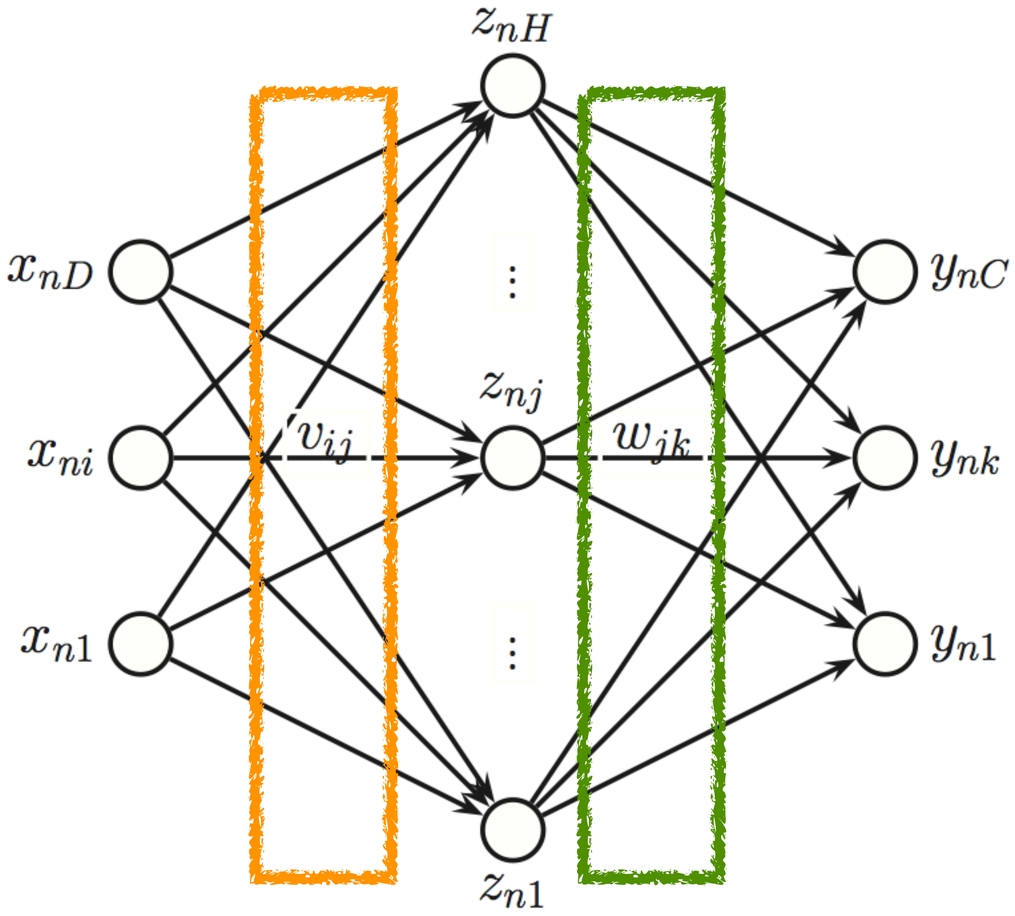
$\hat{\mathbf{y}} = h(\mathbf{b})$ $\mathbf{y}$



'Cross-entropy' loss $l(\hat{\mathbf{y}}, \mathbf{y}) = \sum_{c=1}^{C} \mathbf{y}_c \log(\hat{\mathbf{y}}_c)$

A Neural Network for Multi-way Classification

$$\mathbf{x}_n \xrightarrow{\mathbf{V}} \mathbf{a}_n \xrightarrow{g} \mathbf{z}_n \xrightarrow{\mathbf{W}} \mathbf{b}_n \xrightarrow{h} \hat{\mathbf{y}}_n$$



Inputs

Outputs

Hidden layer

Parameters:

$$\theta = \{\mathbf{v}, \mathbf{w}\}$$

Neural Network in Forward Mode: Recap

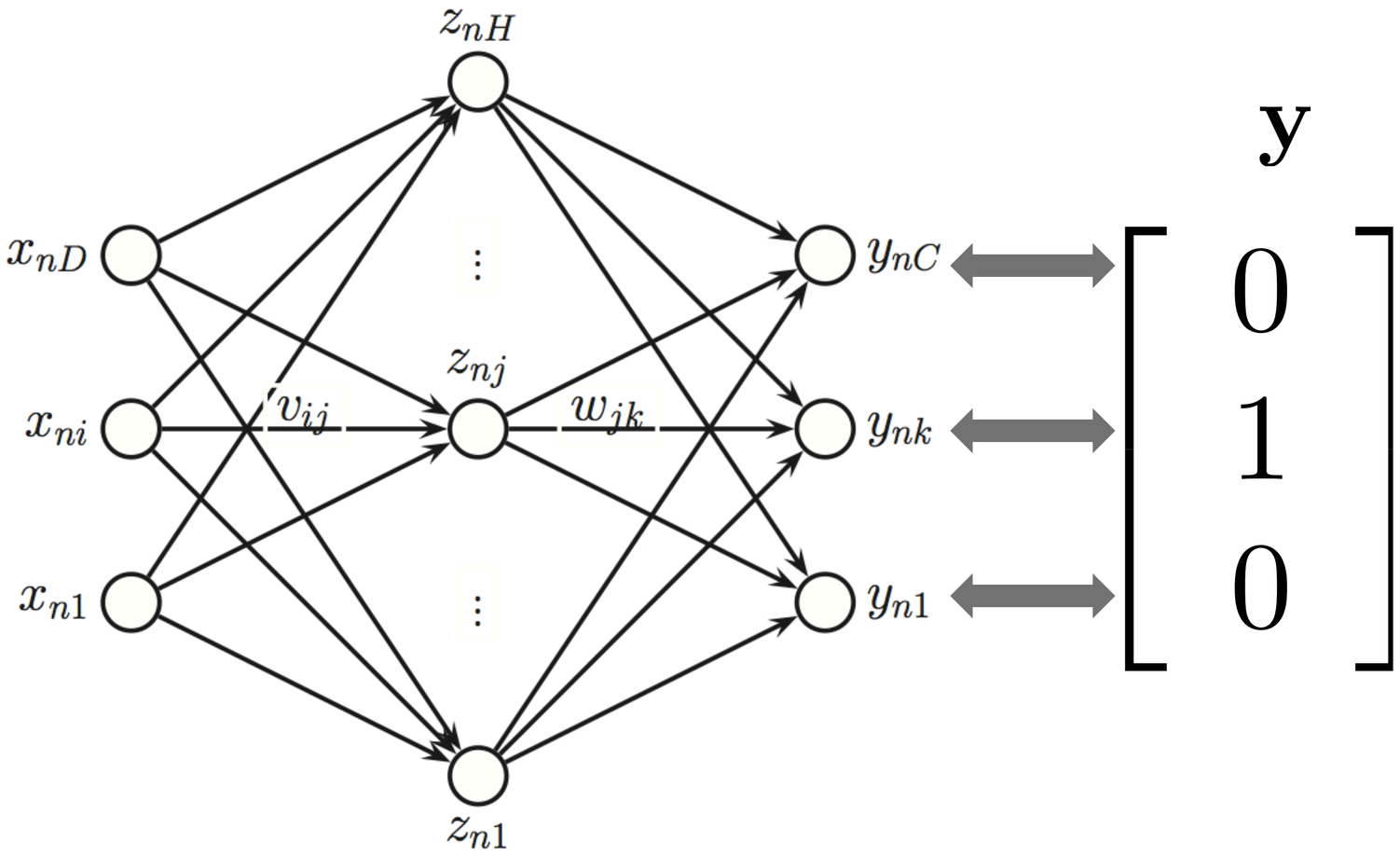
Network output: $\hat{\mathbf{y}} = f(\mathbf{x}; \mathbf{v}, \mathbf{w})$

Loss (prediction error): $l(\hat{\mathbf{y}}, \mathbf{y})$

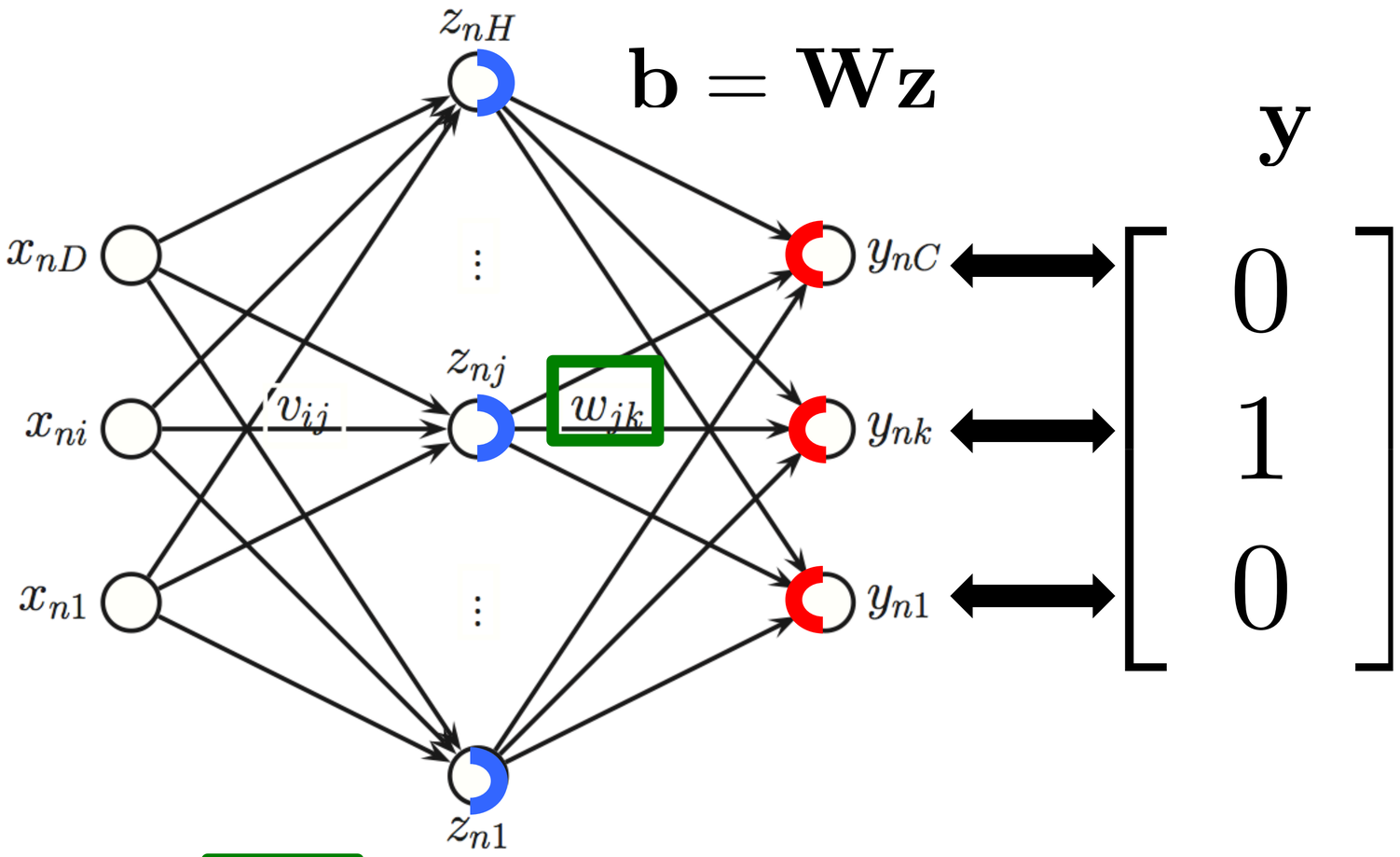
What we need to compute for gradient descent: $\frac{\partial l(\hat{\mathbf{y}}, \mathbf{y})}{\partial \mathbf{v}_i}$

$\frac{\partial l(\hat{\mathbf{y}}, \mathbf{y})}{\partial \mathbf{w}_j}$

A Neural Network in Backward Mode

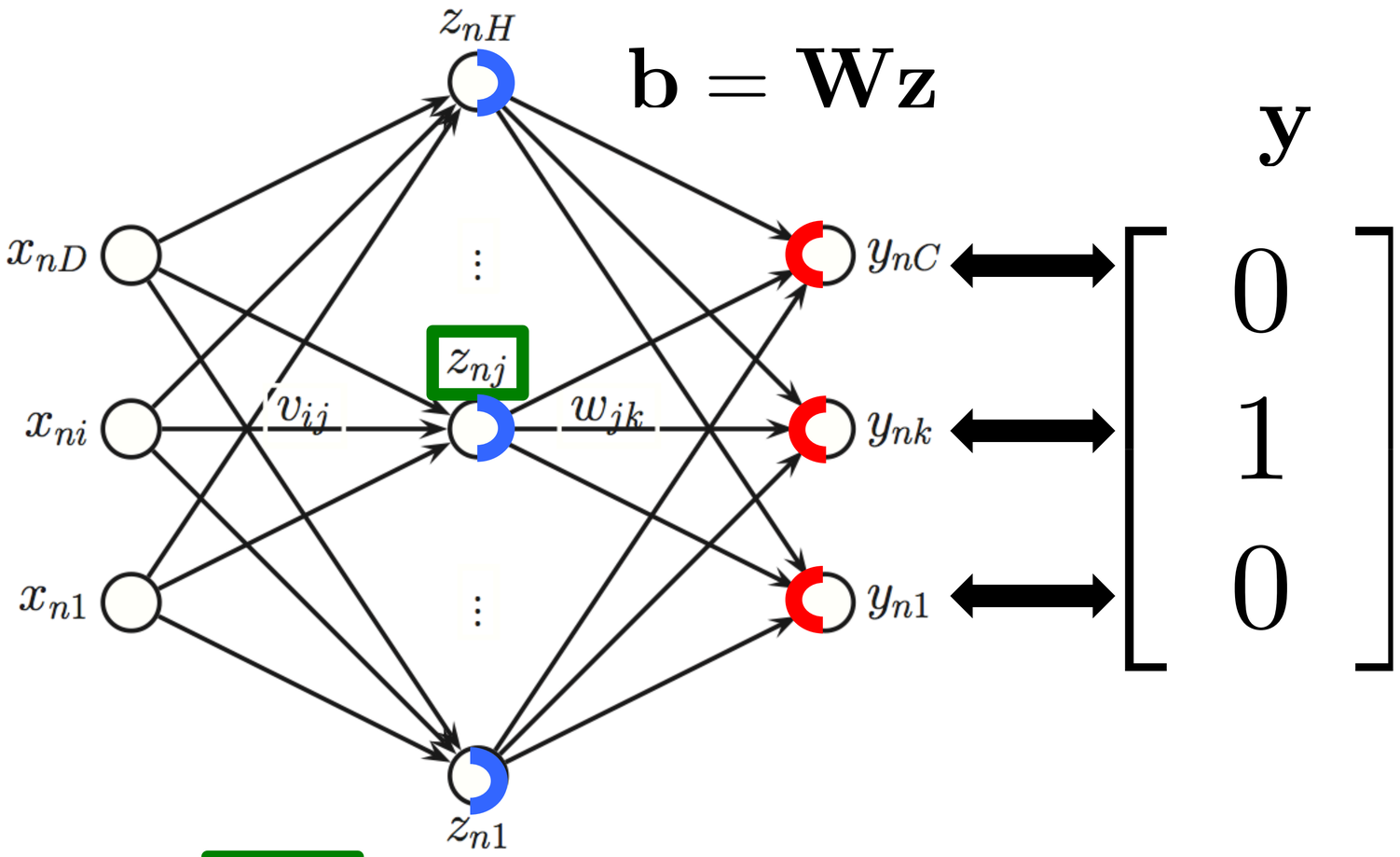


A Neural Network in Backward Mode



This we want $\boxed{\frac{\partial l}{\partial w_{jk}}} = ?$

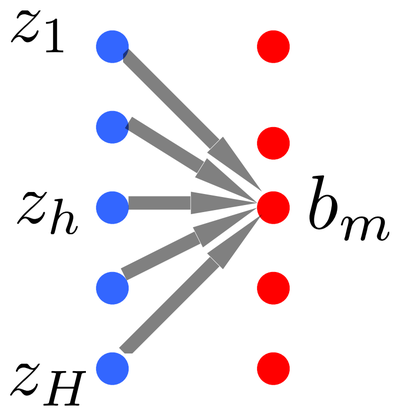
A Neural Network in Backward Mode



This we want $\boxed{\frac{\partial l}{\partial z_j}} = ?$

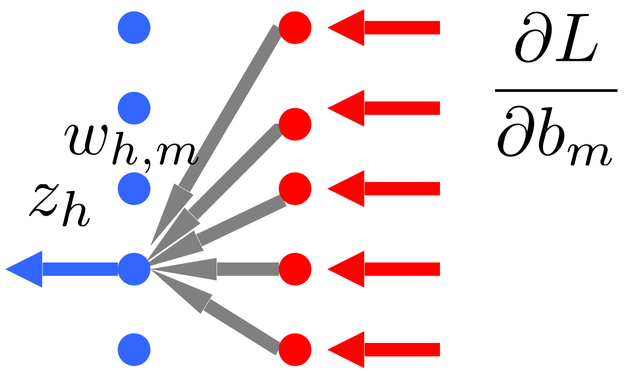
Linear Layer in Forward Mode: All For One

$$b_m = \sum_{h=1}^{H} z_h w_{h,m}$$



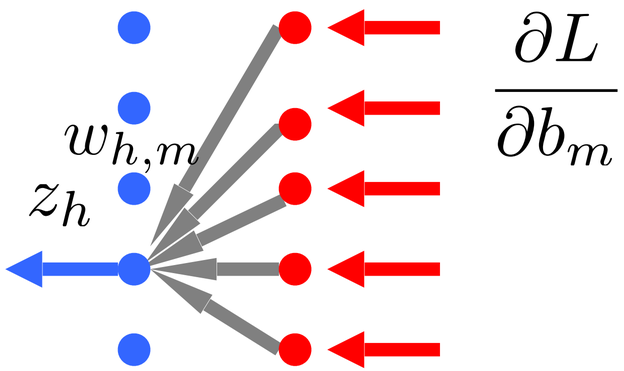
Linear Layer in Backward Mode: All For One

$$b_m = \sum_{h=1}^{H} z_h w_{h,m}$$



Linear Layer in Backward Mode: All For One

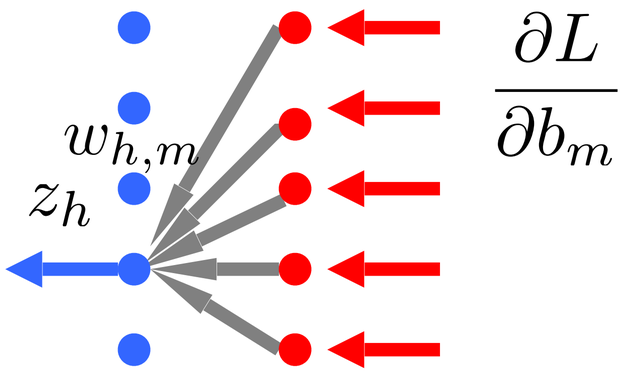
$$b_m = \sum_{h=1}^{H} z_h w_{h,m}$$



$$\frac{\partial L}{\partial z_h} = \sum_{c=1}^{C} \frac{\partial L}{\partial b_c} \cdot \frac{\partial b_c}{\partial z_h}$$

Linear Layer in Backward Mode: All For One

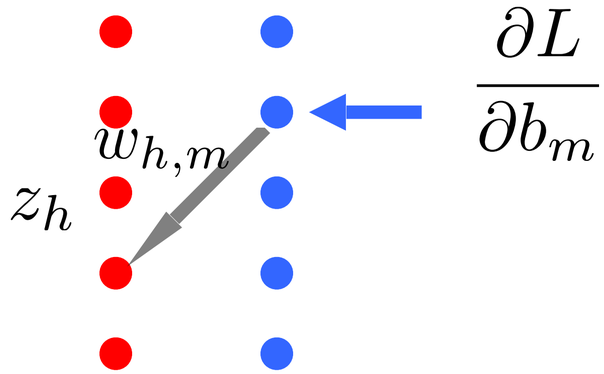
$$b_m = \sum_{h=1}^{H} z_h w_{h,m}$$



$$\frac{\partial L}{\partial z_h} = \sum_{c=1}^{C} \frac{\partial L}{\partial b_c} \cdot \frac{\partial b_c}{\partial z_h} = \sum_{c=1}^{C} \frac{\partial L}{\partial b_c} w_{h,c}$$

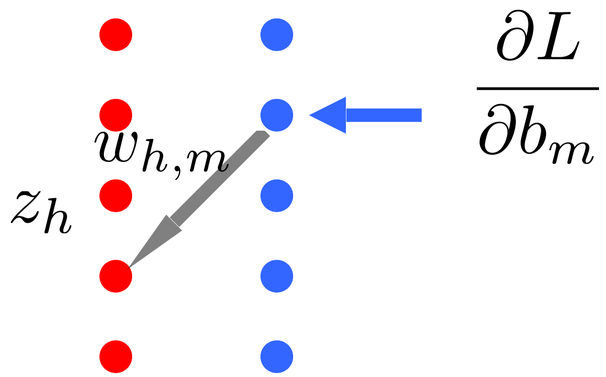
Linear Layer Parameters in Backward: 1-to-1

$$b_m = \sum_{h=1}^{H} z_h w_{h,m}$$



Linear Layer Parameters in Backward: 1-to-1

$$b_m = \sum_{h=1}^{H} z_h w_{h,m}$$

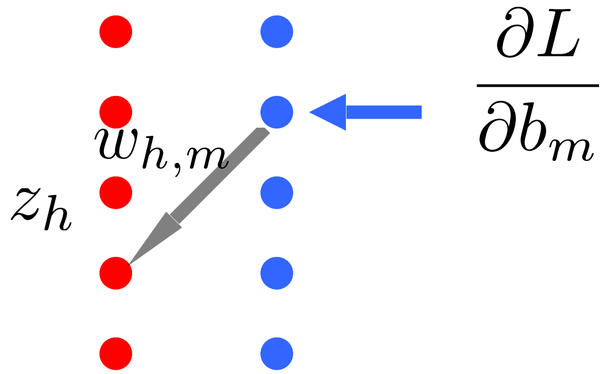


$$\frac{\partial L}{\partial b_m}$$

$$\frac{\partial L}{\partial w_{h,m}} = \sum_{c=1}^{C} \frac{\partial L}{\partial b_c} \cdot \frac{\partial b_c}{\partial w_{h,m}}$$

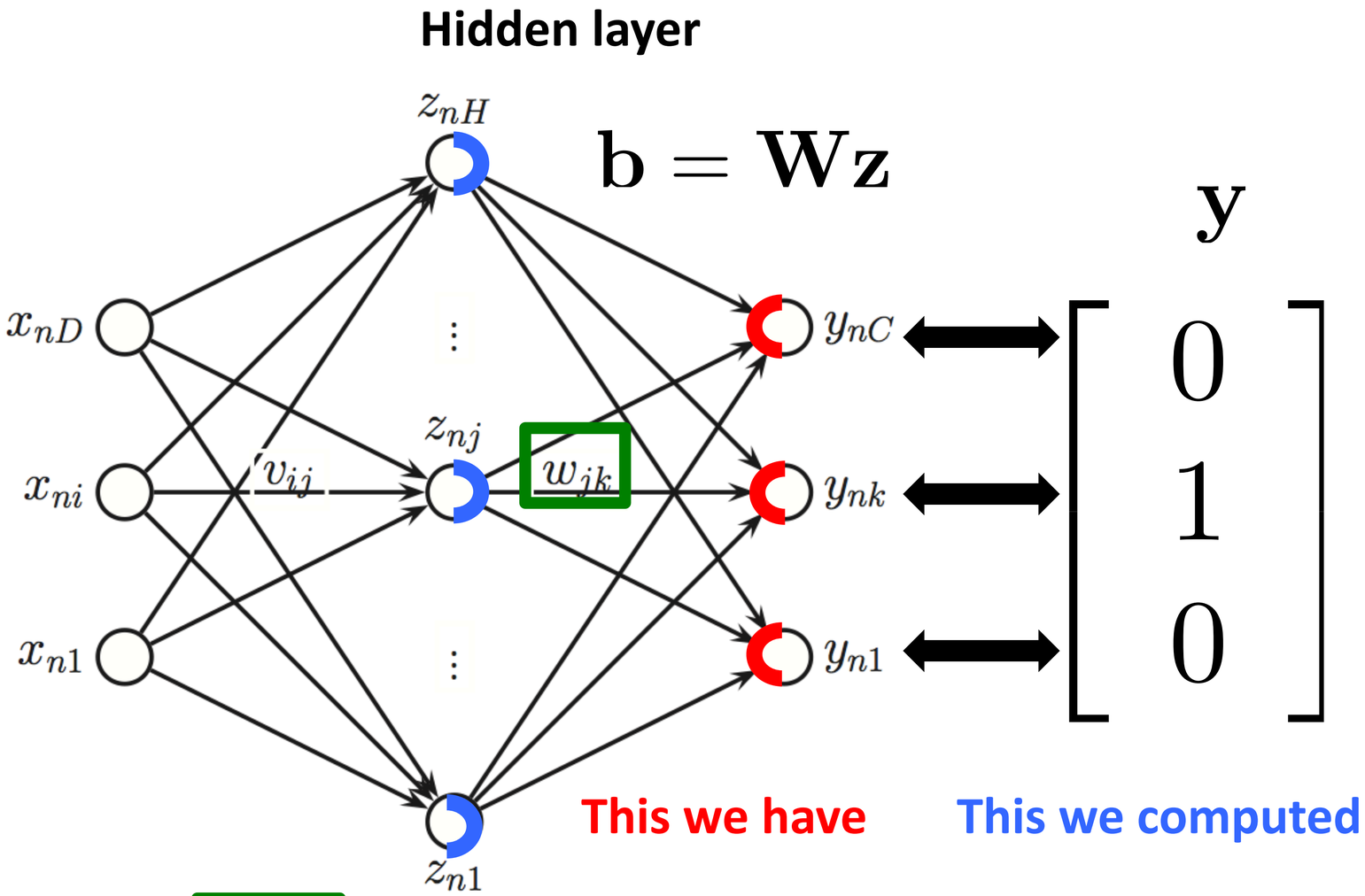
Linear Layer Parameters in Backward: 1-to-1

$$b_m = \sum_{h=1}^{H} z_h w_{h,m}$$



$$\frac{\partial L}{\partial w_{h,m}} = \sum_{c=1}^{C} \frac{\partial L}{\partial b_c} \cdot \frac{\partial b_c}{\partial w_{h,m}} = \frac{\partial L}{\partial b_m} z_h$$

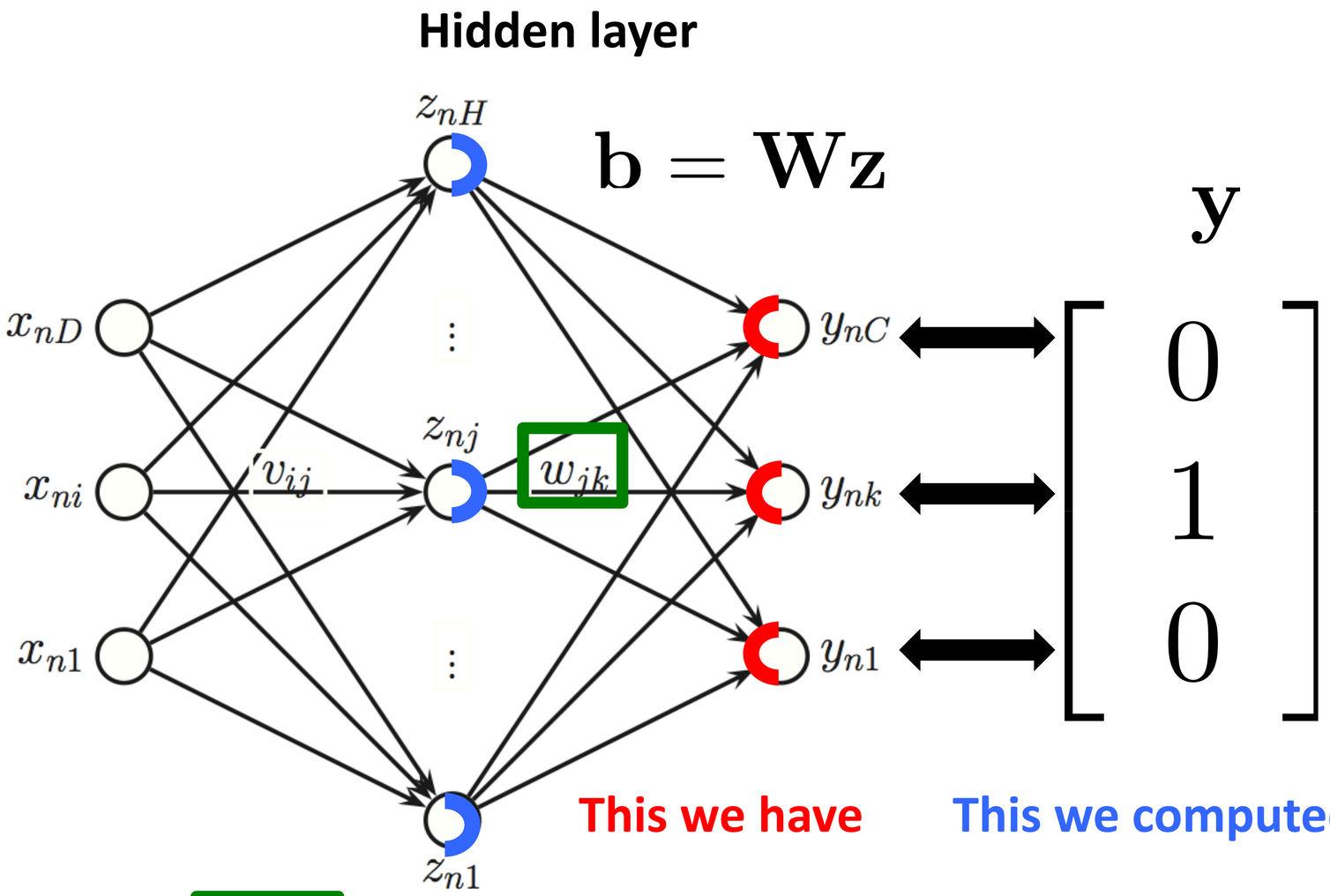
A Neural Network in Backward Mode



This we want

$$\boxed{\frac{\partial l}{\partial w_{jk}}} = \sum_m \boxed{\frac{\partial l}{\partial b_m}} \boxed{\frac{\partial b_m}{\partial w_{jk}}}$$

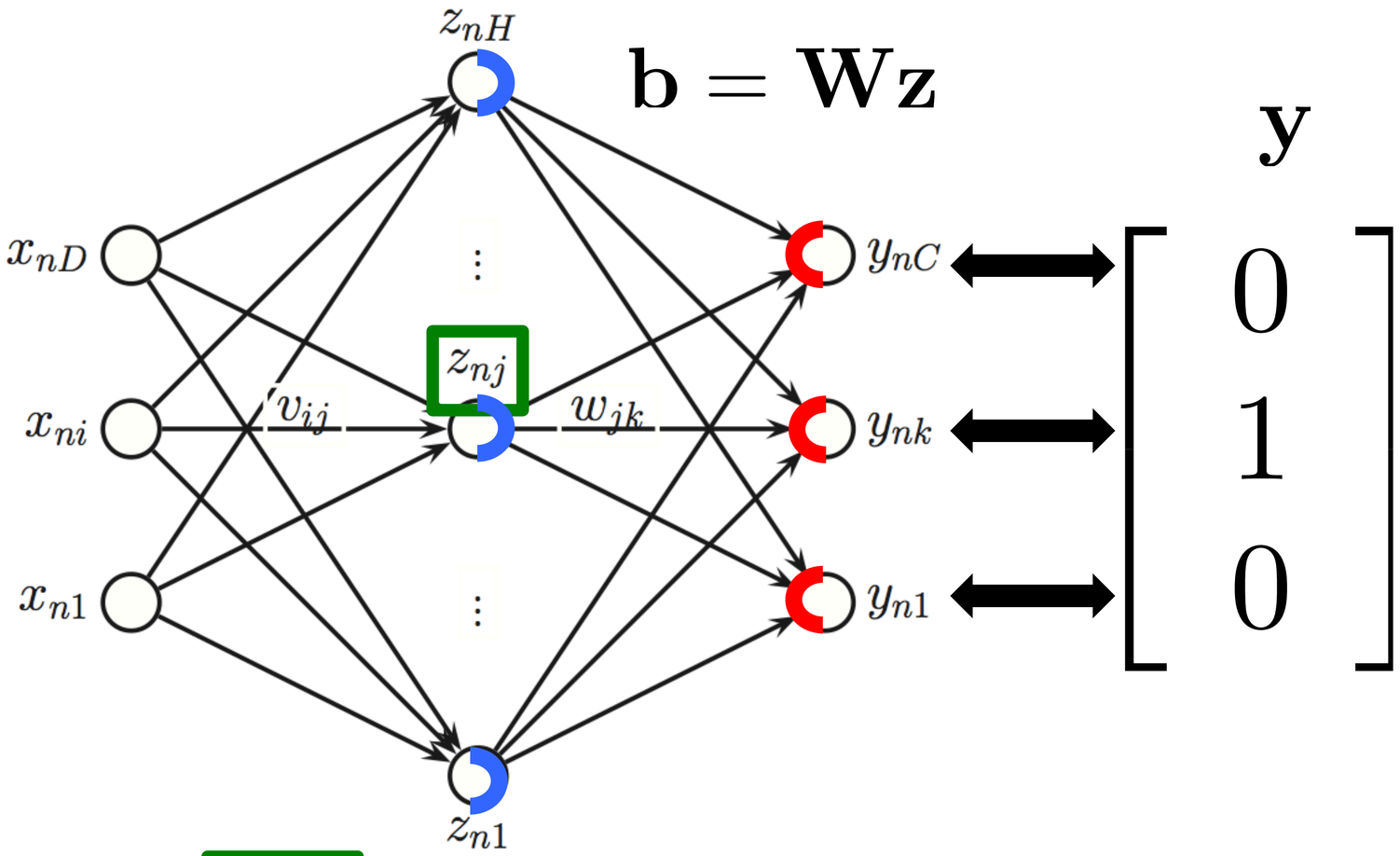
A Neural Network in Backward Mode



This we want

$$\frac{\partial l}{\partial w_{jk}} = \sum_m \frac{\partial l}{\partial b_m} \frac{\partial b_m}{\partial w_{jk}} = \frac{\partial l}{\partial b_m} z_j$$

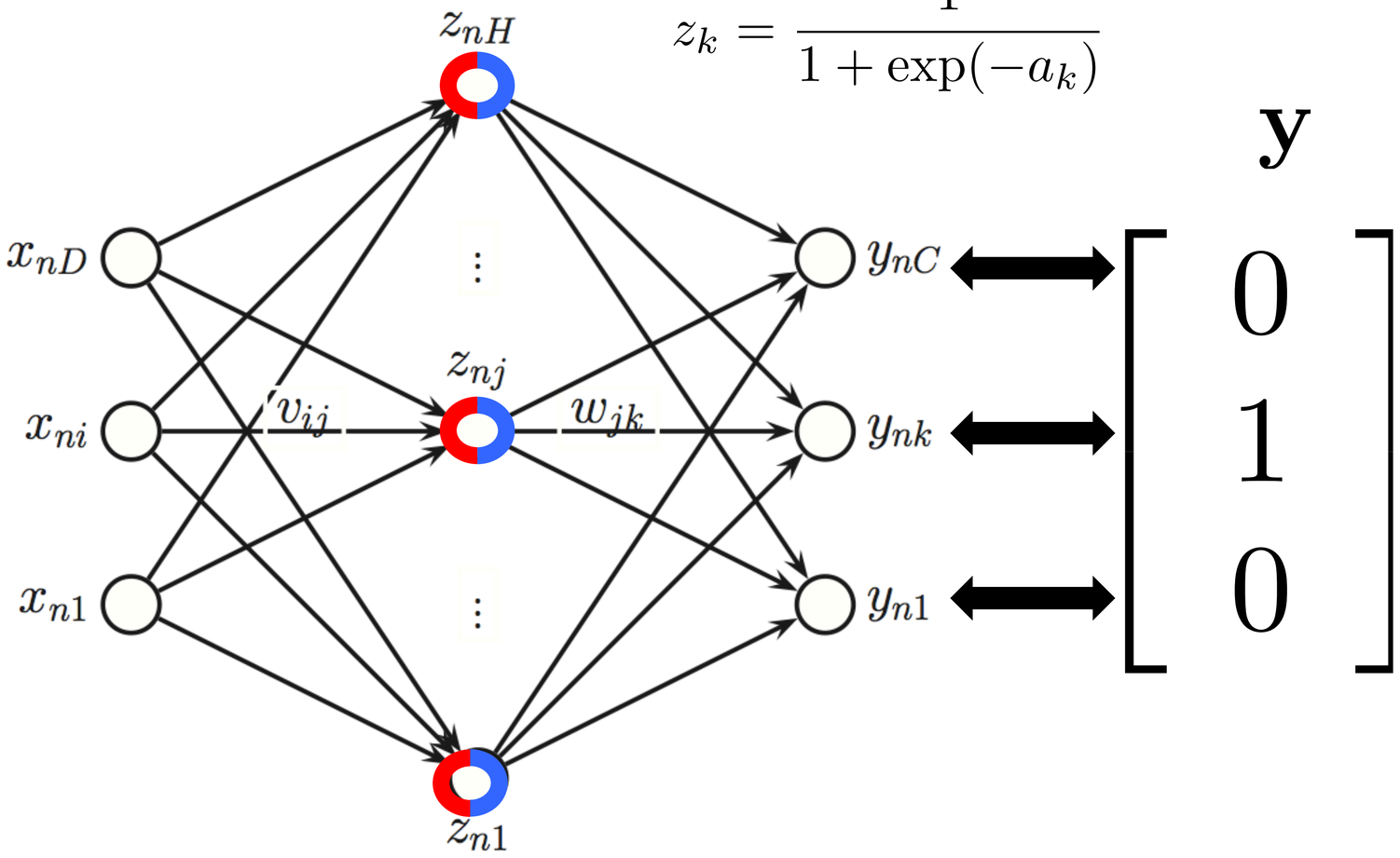
A Neural Network in Backward Mode



$$\frac{\partial l}{\partial z_j} = \sum_m \frac{\partial l}{\partial b_m} \frac{\partial b_m}{\partial z_j} = \sum_m \frac{\partial l}{\partial b_m} w_{j,m}$$

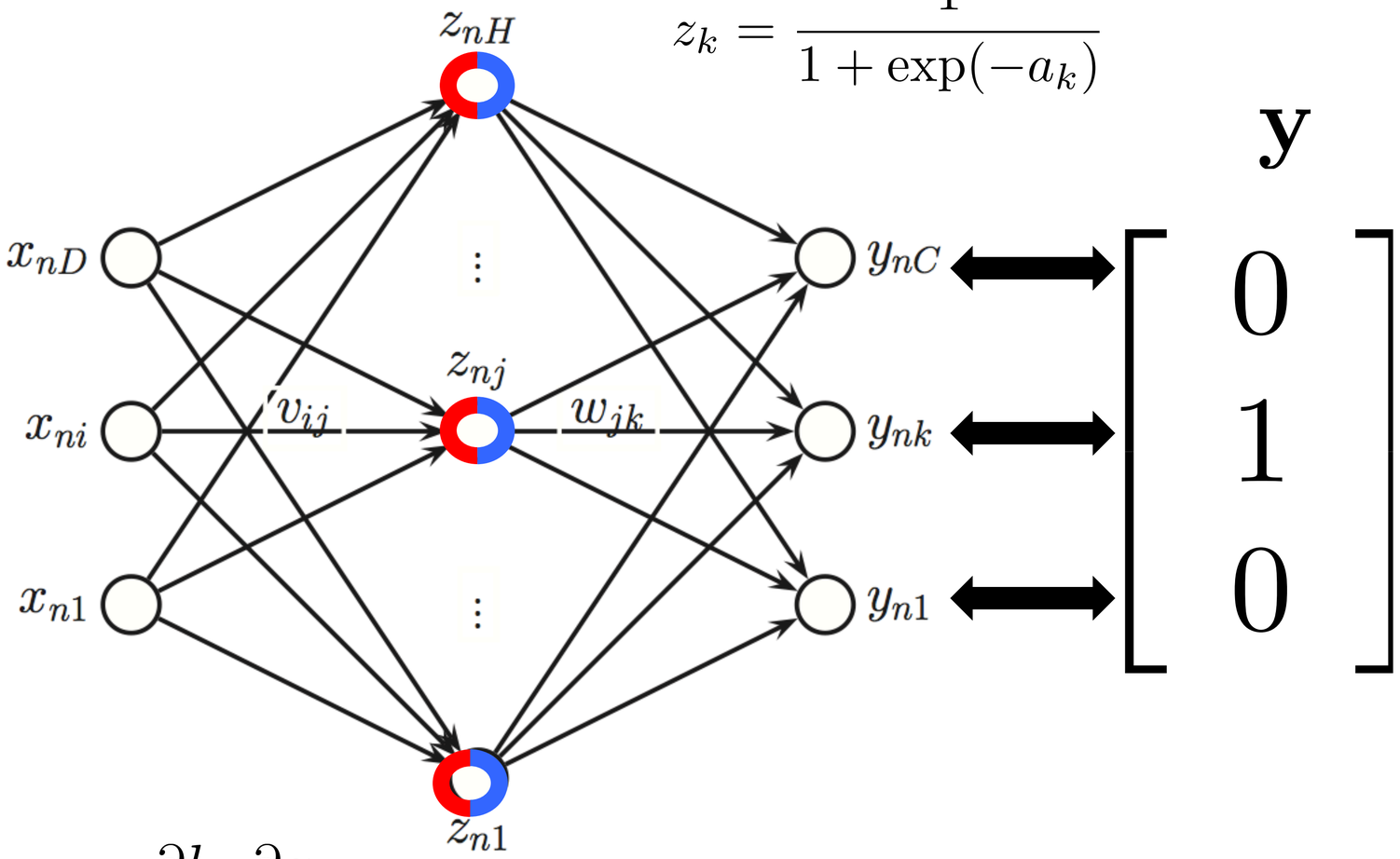
A Neural Network in Backward Mode

$$z_k = \frac{1}{1 + \exp(-a_k)}$$



A Neural Network in Backward Mode

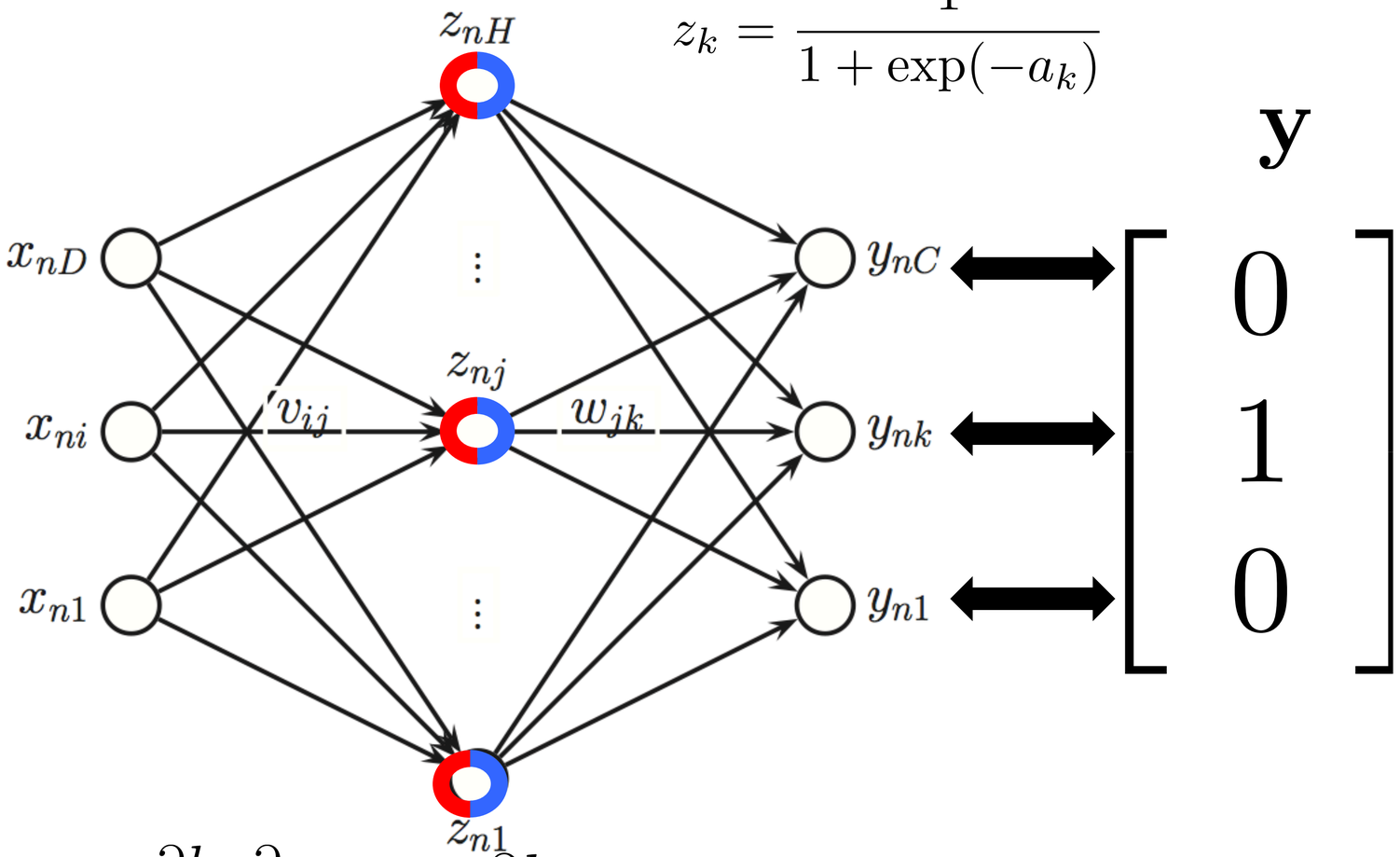
$$z_k = \frac{1}{1 + \exp(-a_k)}$$



$$\frac{\partial l}{\partial a_k} = \sum_m \frac{\partial l}{\partial z_m} \frac{\partial z_m}{\partial a_k}$$

A Neural Network in Backward Mode

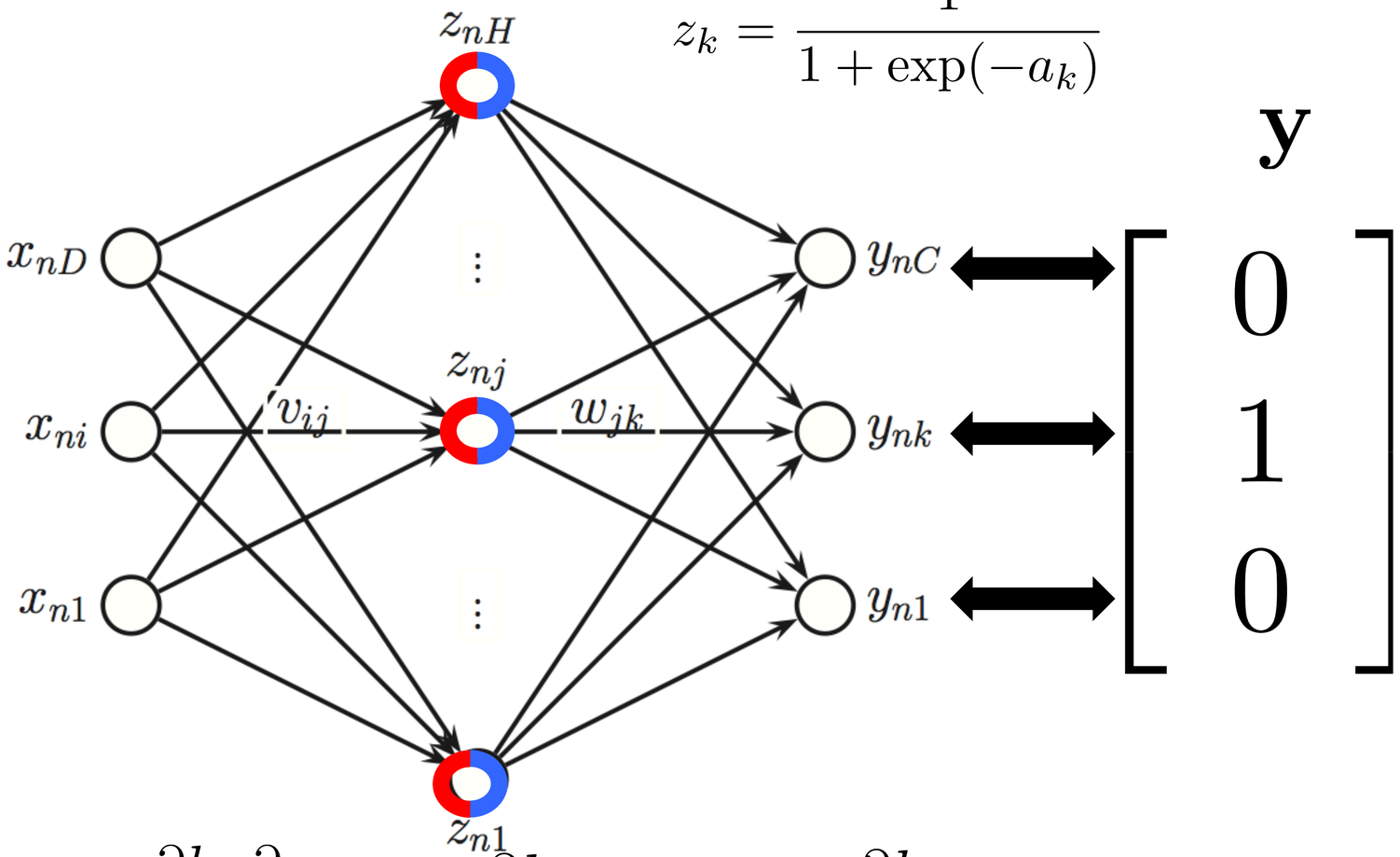
$$z_k = \frac{1}{1 + \exp(-a_k)}$$



$$\frac{\partial l}{\partial a_k} = \sum_m \frac{\partial l}{\partial z_m} \frac{\partial z_m}{\partial a_k} = \frac{\partial l}{\partial z_k} g'(a_k)$$

A Neural Network in Backward Mode

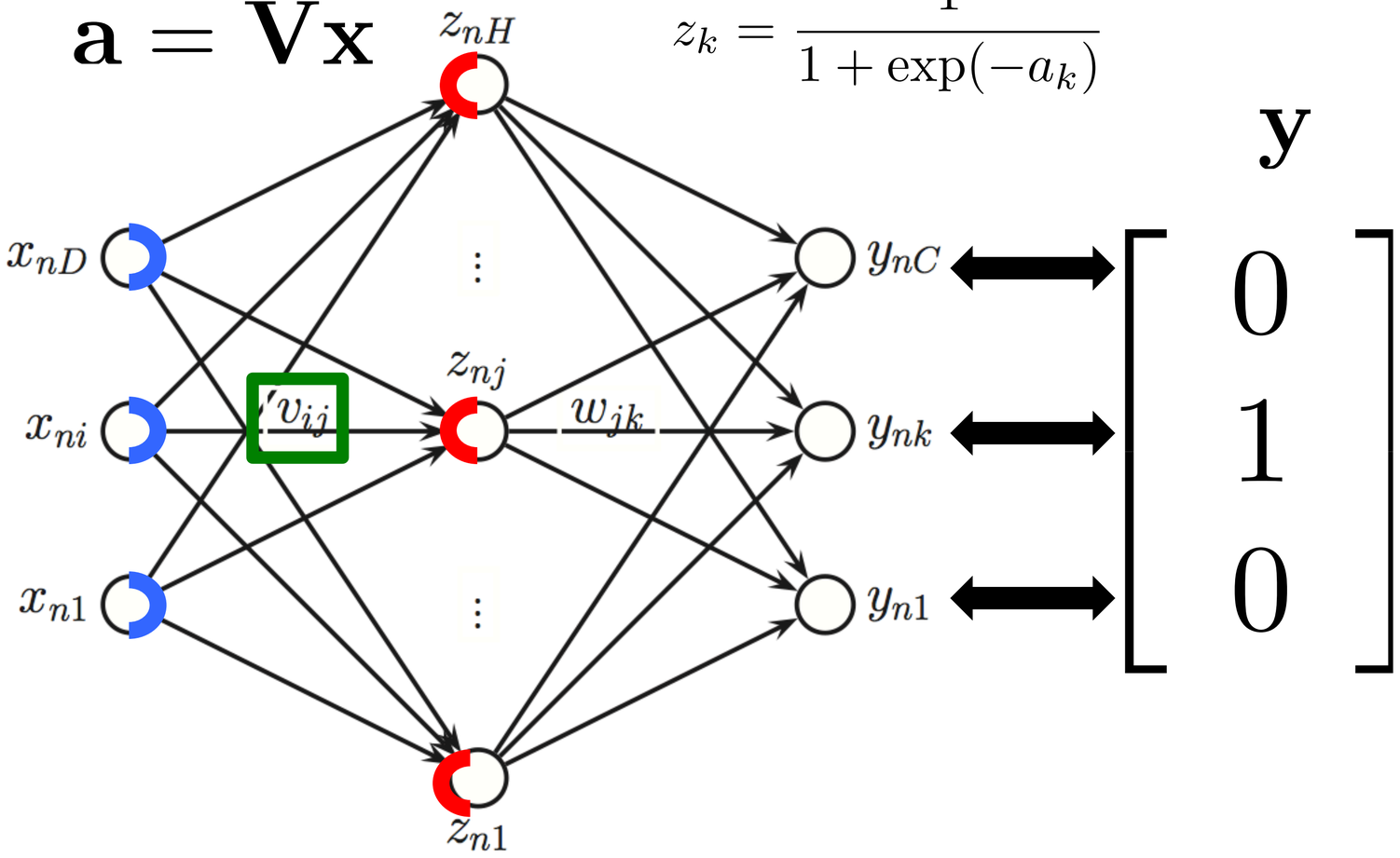
$$z_k = \frac{1}{1 + \exp(-a_k)}$$



$$\frac{\partial l}{\partial a_k} = \sum_m \frac{\partial l}{\partial z_m}\frac{\partial z_m}{\partial a_k} = \frac{\partial l}{\partial z_k} g'(a_k) = \frac{\partial l}{\partial z_k} g(a_k)(1 - g(a_k))$$

A Neural Network in Backward Mode

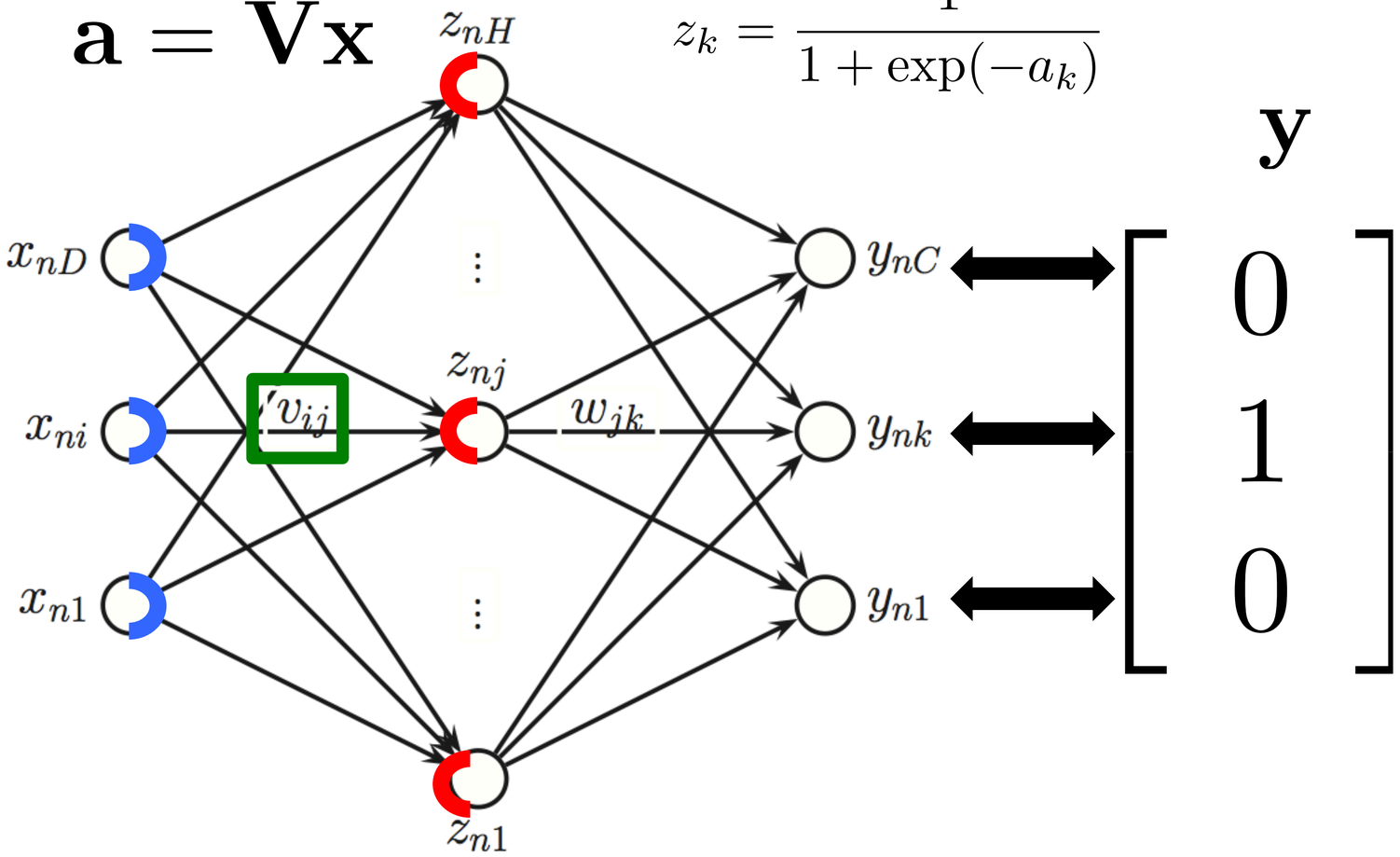
$$\mathbf{a} = \mathbf{V}\mathbf{x} \qquad z_k = \frac{1}{1 + \exp(-a_k)}$$



A Neural Network in Backward Mode

$$\mathbf{a} = \mathbf{V}\mathbf{x}$$

$$z_k = \frac{1}{1 + \exp(-a_k)}$$

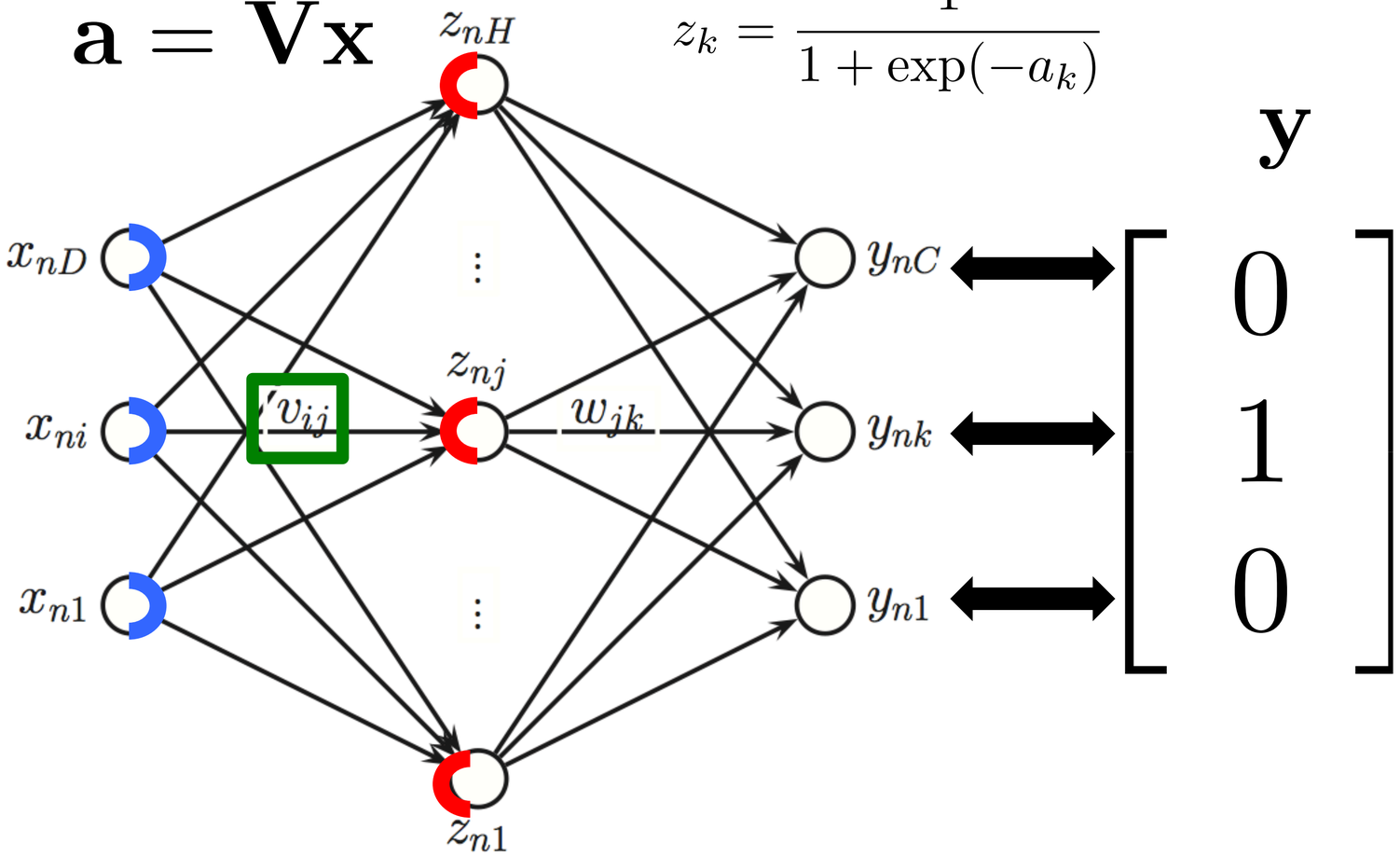


$$\frac{\partial l}{\partial v_{ij}} = \sum_k \frac{\partial l}{\partial a_k} \frac{\partial a_k}{\partial v_{ij}}$$

A Neural Network in Backward Mode

$$\mathbf{a} = \mathbf{V}\mathbf{x}$$

$$z_k = \frac{1}{1 + \exp(-a_k)}$$



$$\frac{\partial l}{\partial v_{ij}} = \sum_k \frac{\partial l}{\partial a_k} \frac{\partial a_k}{\partial v_{ij}} = \frac{\partial l}{\partial a_j} x_i$$

Neural Network Training: Old and New Tricks

- Old
  - Backpropagation algorithm
  - **Stochastic gradient, momentum, weight decay**

- New
  - Dropout
  - Relu
  - Batch Norm(alization), GroupNorm, Spectral Normalization
  - Res(idual) Net(work)

Training Objective for N training samples

$$L(\mathbf{W}) = \frac{1}{N}\sum_{i=1}^{N} l(\mathbf{y}^i, \hat{\mathbf{y}}^i) + \sum_l \lambda_l \sum_{k,m} (\mathbf{W}_{k,m}^l)^2$$

Per-sample loss

Per-layer regularization

Training Objective for N training samples

$$L(\mathbf{W}) = \frac{1}{N}\sum_{i=1}^{N} l(\mathbf{y}^i, \hat{\mathbf{y}}^i) + \sum_l \lambda_l \sum_{k,m} (\mathbf{W}_{k,m}^l)^2$$

Per-sample loss

Per-layer regularization

Gradient descent: $\mathbf{W}_{t+1} = \mathbf{W}_t - \epsilon \nabla_{\mathbf{W}} L(\mathbf{W}_t)$

Training Objective for N training samples

$$L(\mathbf{W}) = \frac{1}{N}\sum_{i=1}^{N} l(\mathbf{y}^i, \hat{\mathbf{y}}^i) + \sum_l \lambda_l \sum_{k,m} (\mathbf{W}_{k,m}^l)^2$$

Per-sample loss

Per-layer regularization

Gradient descent: $\mathbf{W}_{t+1} = \mathbf{W}_t - \epsilon \nabla_{\mathbf{W}} L(\mathbf{W}_t)$

(l,k,m) element of gradient vector:

Training Objective for N training samples

$$L(\mathbf{W}) = \frac{1}{N}\sum_{i=1}^{N} l(\mathbf{y}^i, \hat{\mathbf{y}}^i) + \sum_l \lambda_l \sum_{k,m} (\mathbf{W}_{k,m}^l)^2$$

Per-sample loss

Per-layer regularization

Gradient descent: $\mathbf{W}_{t+1} = \mathbf{W}_t - \epsilon \nabla_{\mathbf{W}} L(\mathbf{W}_t)$

(l,k,m) element of gradient vector:

$$\frac{\partial L}{\partial \mathbf{W}_{k,m}^l} = \frac{1}{N}\sum_{i=1}^{N} \frac{\partial l(\mathbf{y}^i, \hat{\mathbf{y}}^i)}{\partial \mathbf{W}_{k,m}^l} + 2\lambda_l \mathbf{W}_{k,m}^l$$

Training Objective for N training samples

$$L(\mathbf{W}) = \frac{1}{N}\sum_{i=1}^{N} l(\mathbf{y}^i, \hat{\mathbf{y}}^i) + \sum_l \lambda_l \sum_{k,m} (\mathbf{W}_{k,m}^l)^2$$

Per-sample loss

Per-layer regularization

Gradient descent: $\mathbf{W}_{t+1} = \mathbf{W}_t - \epsilon \nabla_{\mathbf{W}} L(\mathbf{W}_t)$

(l,k,m) element of gradient vector:

$$\frac{\partial L}{\partial \mathbf{W}_{k,m}^l} = \frac{1}{N}\sum_{i=1}^{N} \frac{\partial l(\mathbf{y}^i, \hat{\mathbf{y}}^i)}{\partial \mathbf{W}_{k,m}^l} + 2\lambda_l \mathbf{W}_{k,m}^l$$

Back-prop for i-th example

Training Objective for N training samples

$$L(\mathbf{W}) = \frac{1}{N}\sum_{i=1}^{N} l(\mathbf{y}^i, \hat{\mathbf{y}}^i) + \sum_l \lambda_l \sum_{k,m} (\mathbf{W}_{k,m}^l)^2$$

Per-sample loss

Per-layer regularization

Gradient descent: $\mathbf{W}_{t+1} = \mathbf{W}_t - \epsilon \nabla_{\mathbf{W}} L(\mathbf{W}_t)$

(l,k,m) element of gradient vector:

$$\frac{\partial L}{\partial \mathbf{W}_{k,m}^l} = \frac{1}{N}\sum_{i=1}^{N} \frac{\partial l(\mathbf{y}^i, \hat{\mathbf{y}}^i)}{\partial \mathbf{W}_{k,m}^l} + 2\lambda_l \mathbf{W}_{k,m}^l$$

Back-prop for i-th example

If N=10^6 , to update **W** **once** need to back-prop 10^6 times!

Regularization in SGD: Weight Decay

Gradient: **Batch:** [1..N]

$$\frac{\partial L}{\partial \mathbf{W}^l_{k,m}} = \frac{1}{N}\sum_{i=1}^{N}\frac{\partial l(\mathbf{y}^i, \hat{\mathbf{y}}^i)}{\partial \mathbf{W}^l_{k,m}} + 2\lambda_l \mathbf{W}^l_{k,m}$$

Regularization in SGD: Weight Decay

Gradient: **Batch:** [1..N]

$$\frac{\partial L}{\partial \mathbf{W}^l_{k,m}} = \frac{1}{N}\sum_{i=1}^{N}\frac{\partial l(\mathbf{y}^i, \hat{\mathbf{y}}^i)}{\partial \mathbf{W}^l_{k,m}} + 2\lambda_l \mathbf{W}^l_{k,m}$$

Noisy **('Stochastic') Gradient**:

Minibatch: B elements

b(1), b(2),…, b(B): ***randomly*** sampled from [1,N]

Regularization in SGD: Weight Decay

Gradient: **Batch:** [1..N]

$$\frac{\partial L}{\partial \mathbf{W}^l_{k,m}} = \frac{1}{N}\sum_{i=1}^{N}\frac{\partial l(\mathbf{y}^i, \hat{\mathbf{y}}^i)}{\partial \mathbf{W}^l_{k,m}} + 2\lambda_l \mathbf{W}^l_{k,m}$$

Noisy **('Stochastic') Gradient**:

Minibatch: B elements

b(1), b(2),…, b(B): ***randomly*** sampled from [1,N]

$$\frac{\partial L}{\partial \mathbf{W}^l_{k,m}} \simeq \frac{1}{B}\sum_{i=1}^{B}\frac{\partial l(\mathbf{y}^{b(i)}, \hat{\mathbf{y}}^{b(i)})}{\partial \mathbf{W}^l_{k,m}} + 2\lambda_l \mathbf{W}^l_{k,m}$$

Back-prop on minibatch

Regularization in SGD: Weight Decay

Gradient: **Batch:** [1..N]

$$\frac{\partial L}{\partial \mathbf{W}^l_{k,m}} = \frac{1}{N}\sum_{i=1}^{N}\frac{\partial l(\mathbf{y}^i, \hat{\mathbf{y}}^i)}{\partial \mathbf{W}^l_{k,m}} + 2\lambda_l \mathbf{W}^l_{k,m}$$

Noisy **('Stochastic') Gradient**:

Minibatch: B elements

b(1), b(2),…, b(B): ***randomly*** sampled from [1,N]

$$\frac{\partial L}{\partial \mathbf{W}^l_{k,m}} \simeq \frac{1}{B}\sum_{i=1}^{B}\frac{\partial l(\mathbf{y}^{b(i)}, \hat{\mathbf{y}}^{b(i)})}{\partial \mathbf{W}^l_{k,m}} + 2\lambda_l \mathbf{W}^l_{k,m}$$

Back-prop on minibatch

Weight decay

Regularization in SGD: Weight Decay

Gradient: **Batch:** [1..N]

$$\frac{\partial L}{\partial \mathbf{W}^l_{k,m}} = \frac{1}{N}\sum_{i=1}^{N}\frac{\partial l(\mathbf{y}^i, \hat{\mathbf{y}}^i)}{\partial \mathbf{W}^l_{k,m}} + 2\lambda_l \mathbf{W}^l_{k,m}$$

Noisy **('Stochastic') Gradient**:

Minibatch: B elements

b(1), b(2),…, b(B): ***randomly*** sampled from [1,N]

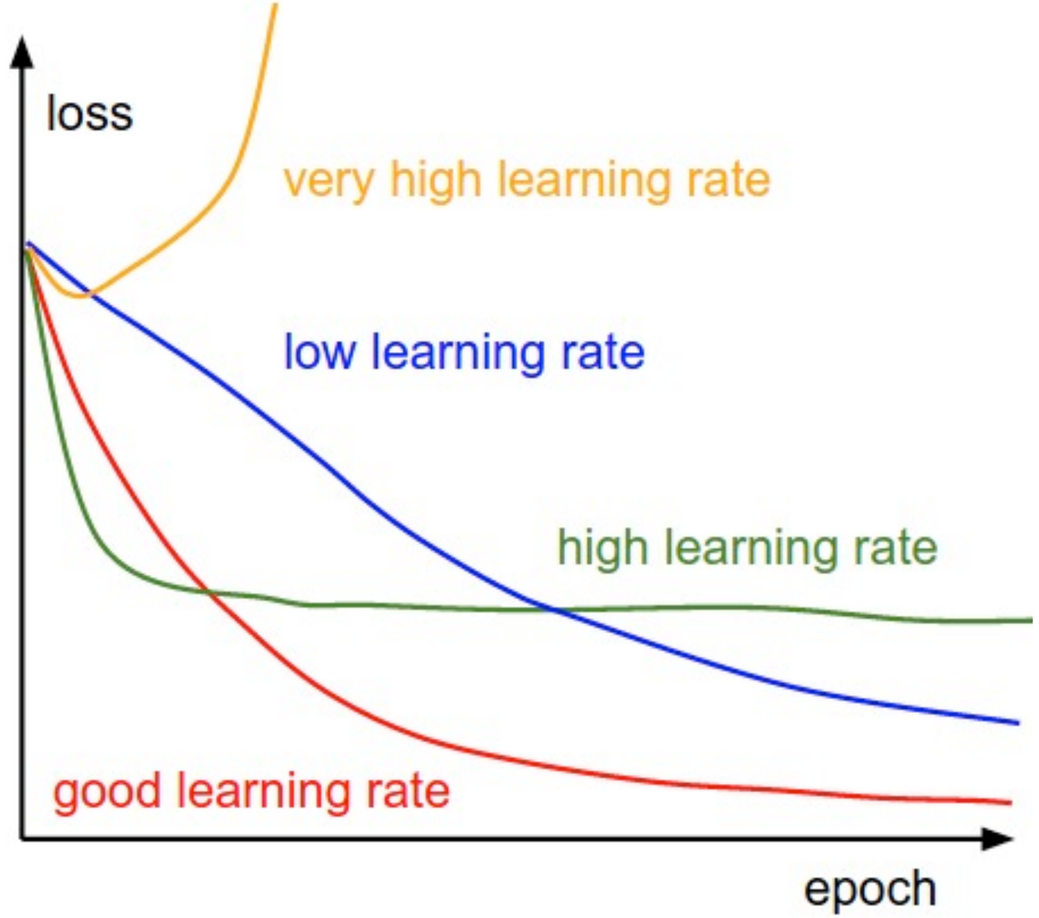
$$\frac{\partial L}{\partial \mathbf{W}^l_{k,m}} \simeq \frac{1}{B}\sum_{i=1}^{B}\frac{\partial l(\mathbf{y}^{b(i)}, \hat{\mathbf{y}}^{b(i)})}{\partial \mathbf{W}^l_{k,m}} + 2\lambda_l \mathbf{W}^l_{k,m}$$

Back-prop on minibatch

Weight decay

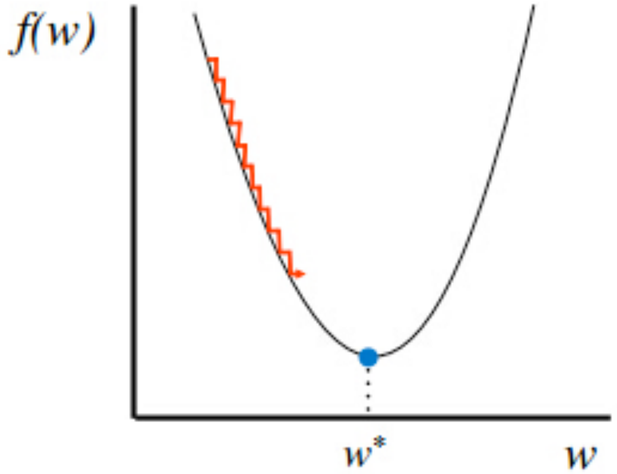
Epoch: N samples, N/B batches

Learning Rate

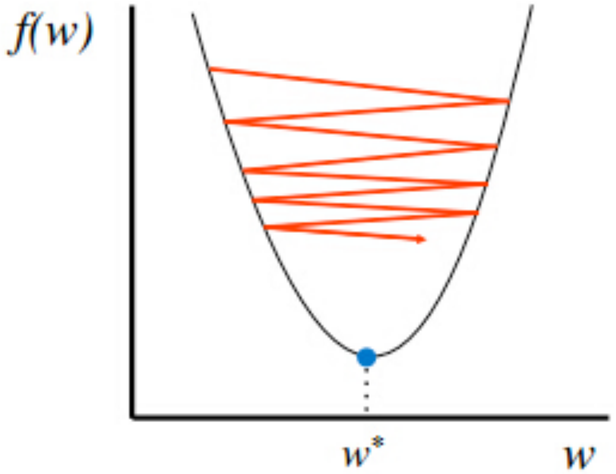


$$\mathbf{W}_{t+1} = \mathbf{W}_t - \epsilon \nabla_{\mathbf{W}} L(\mathbf{W}_t)$$

(S)GD with Adaptable Stepsize

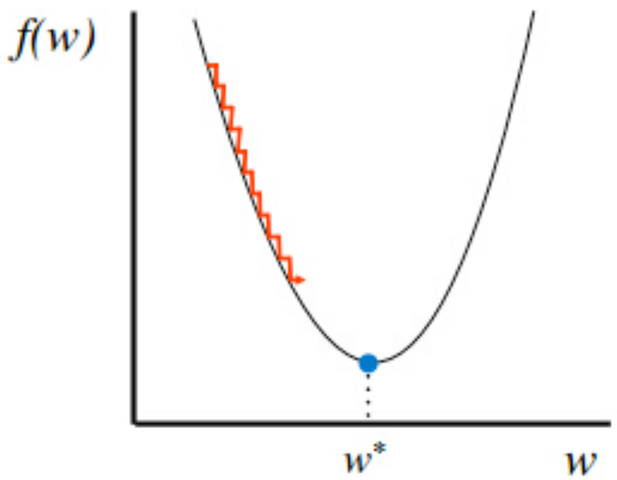


Too small: converge very slowly

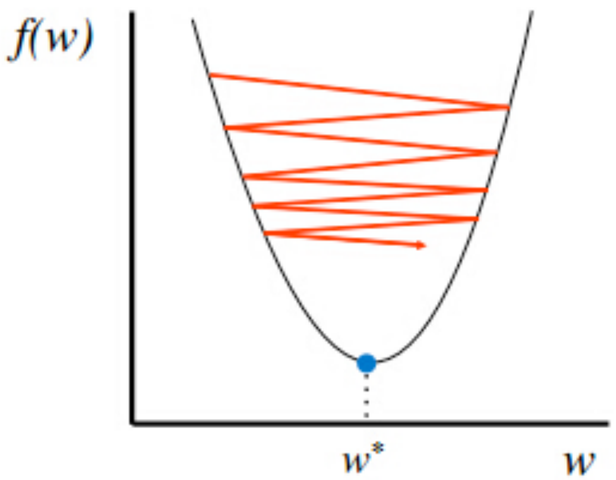


Too big: overshoot and even diverge

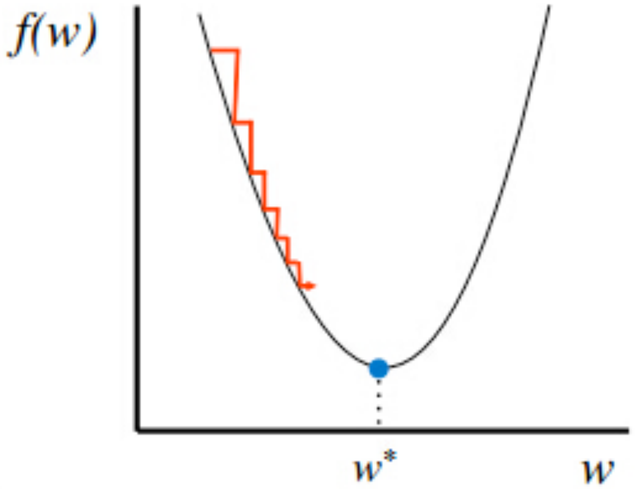
(S)GD with Adaptable Stepsize



Too small: converge very slowly

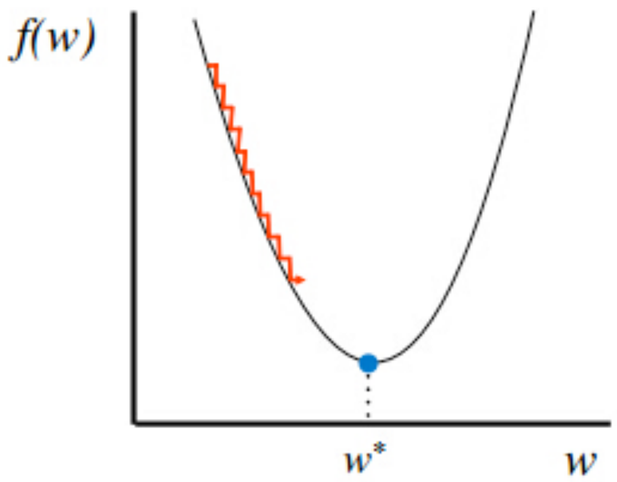


Too big: overshoot and even diverge

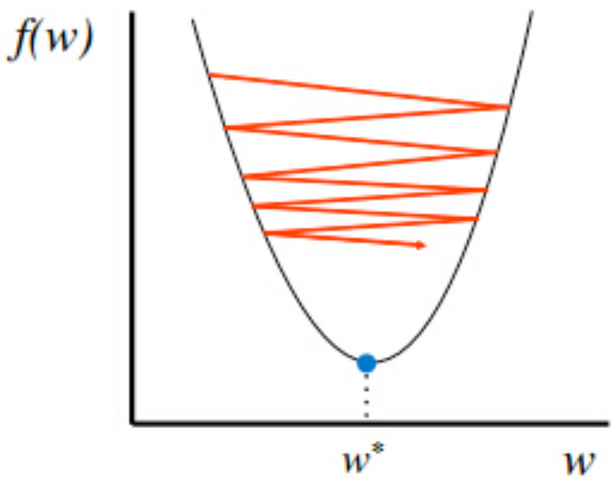


Reduce size over time

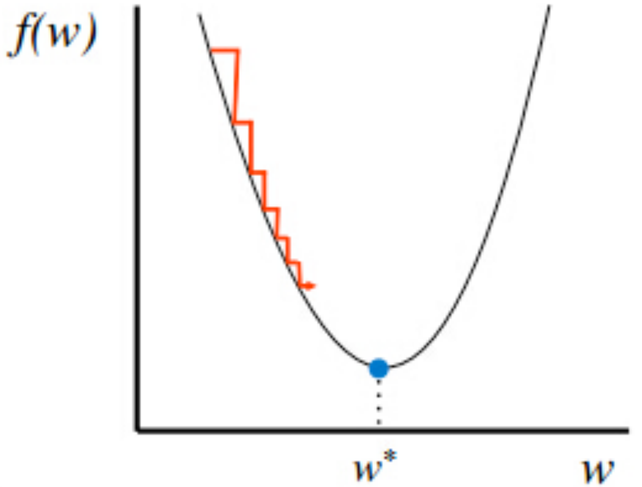
(S)GD with Adaptable Stepsize



Too small: converge very slowly



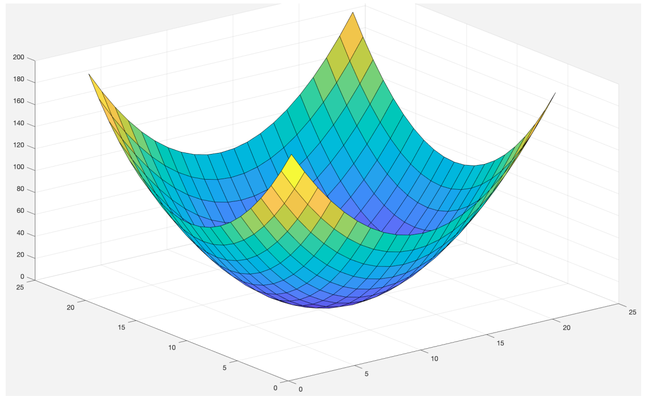
Too big: overshoot and even diverge

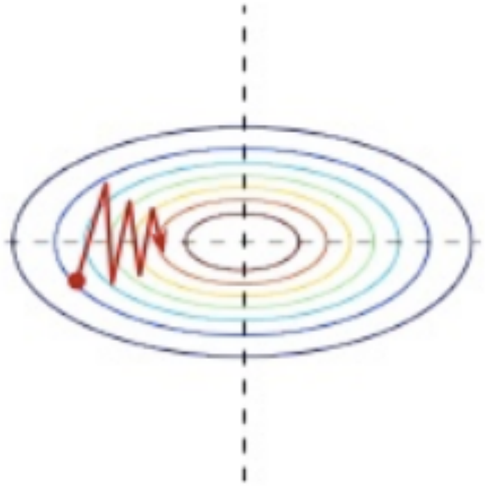


Reduce size over time

e.g. $\epsilon_t = \frac{c}{t}$

(S)GD with Momentum

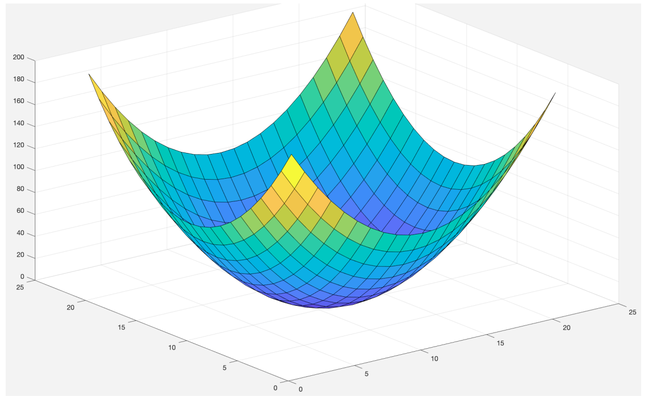


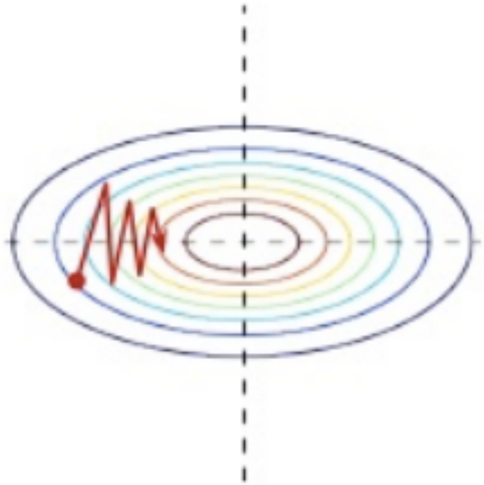


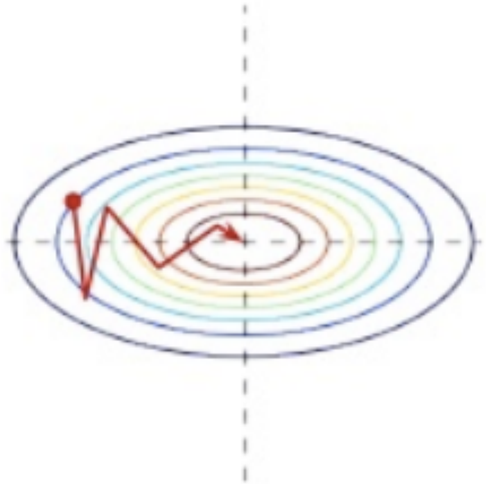
Main idea: retain long-term trend of updates, drop oscillations

$$\mathbf{W}_{t+1} = \mathbf{W}_t - \epsilon_t \nabla_{\mathbf{W}} L(\mathbf{W}_t)$$

(S)GD with Momentum



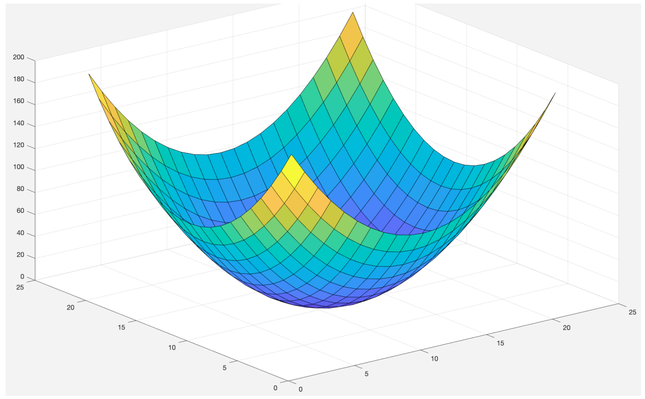


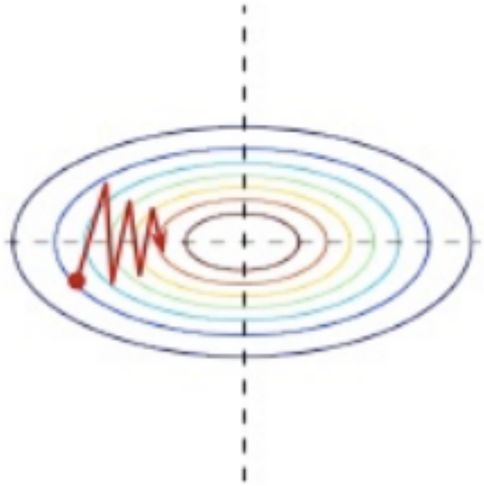


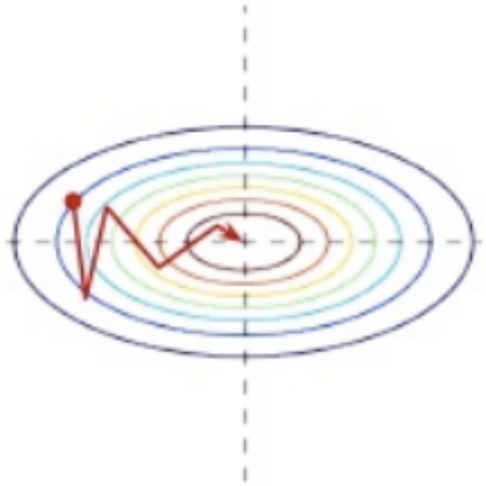
Main idea: retain long-term trend of updates, drop oscillations

$$\mathbf{W}_{t+1} = \mathbf{W}_t - \epsilon_t \nabla_{\mathbf{W}} L(\mathbf{W}_t)$$

(S)GD with Momentum



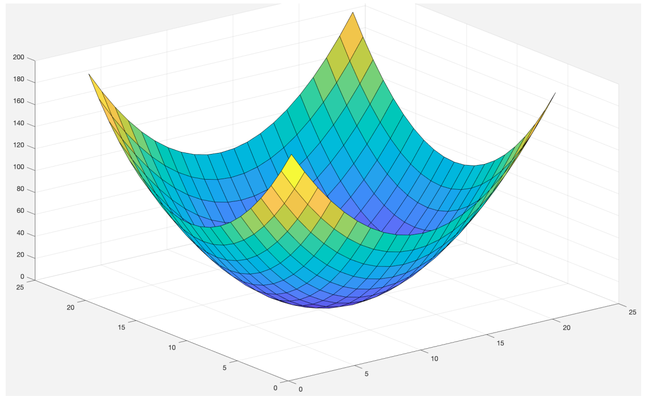


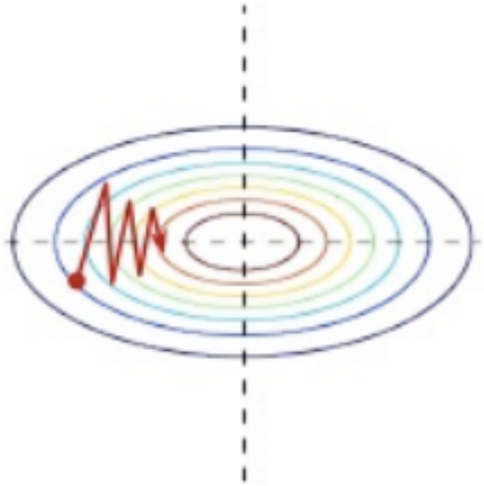


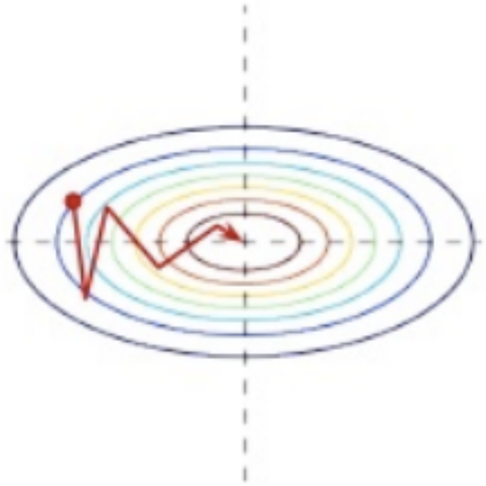
Main idea: retain long-term trend of updates, drop oscillations

(S)GD $$\mathbf{W}_{t+1} = \mathbf{W}_t - \epsilon_t \nabla_{\mathbf{W}} L(\mathbf{W}_t)$$

(S)GD with Momentum







Main idea: retain long-term trend of updates, drop oscillations

(S)GD $$\mathbf{W}_{t+1} = \mathbf{W}_t - \epsilon_t \nabla_{\mathbf{W}} L(\mathbf{W}_t)$$

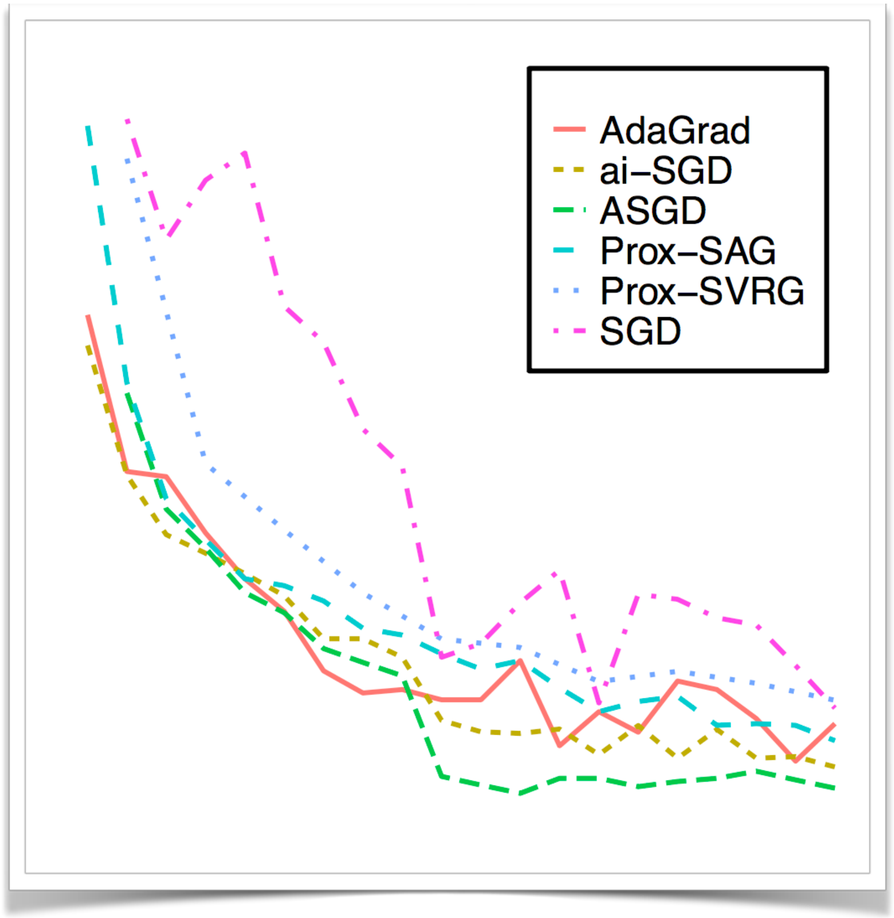
(S)GD + momentum

$$\mathbf{V}_{t+1} = \mu \mathbf{V}_t + (1-\mu) \nabla_{\mathbf{W}} L(\mathbf{W}_t)$$

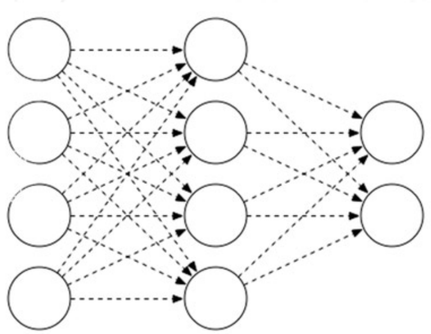
$$\mathbf{W}_{t+1} = \mathbf{W}_t - \epsilon_t \mathbf{V}_{t+1}$$

Step-size Selection & Optimizers

- Nesterov’s Accelerated Gradient (NAG)
- R-prop
- AdaGrad
- RMSProp
- AdaDelta
- Adam
- …



Course Information (slides/code/comments)



http://geometry.cs.ucl.ac.uk/creativeai/

CreativeAI: Deep Learning for Graphics

NN Tricks & Architectures

Niloy Mitra
UCL

Iasonas Kokkinos
UCL/Facebook

Paul Guerrero
UCL

Nils Thuerey
TUM

Tobias Ritschel
UCL

facebook
Artificial Intelligence Research

Technische Universität München

Neural Network Training: Old & New Tricks

Old: (80's)

Stochastic Gradient Descent, Momentum, "weight decay"

New: (last 5-6 years)

Dropout

ReLUs

Batch Normalization

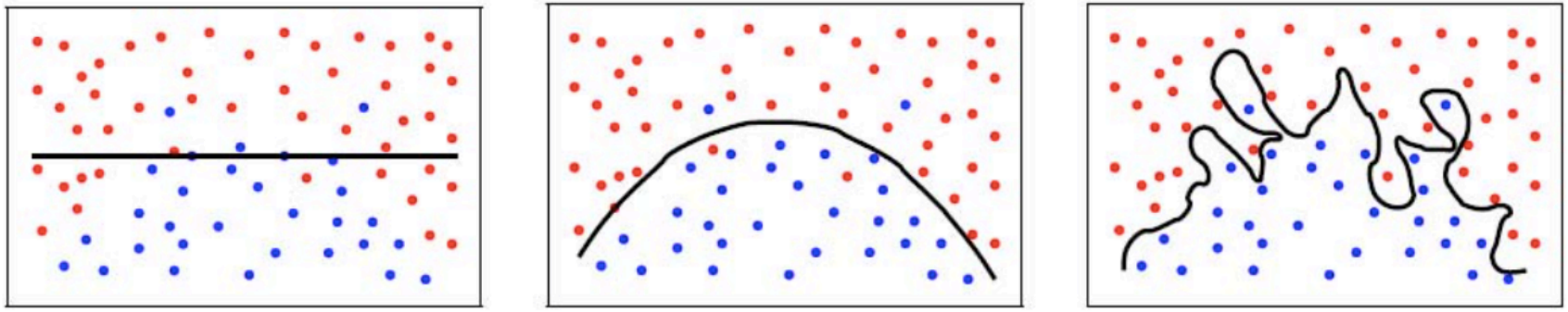
Reminder: Overfitting, in images

Classification

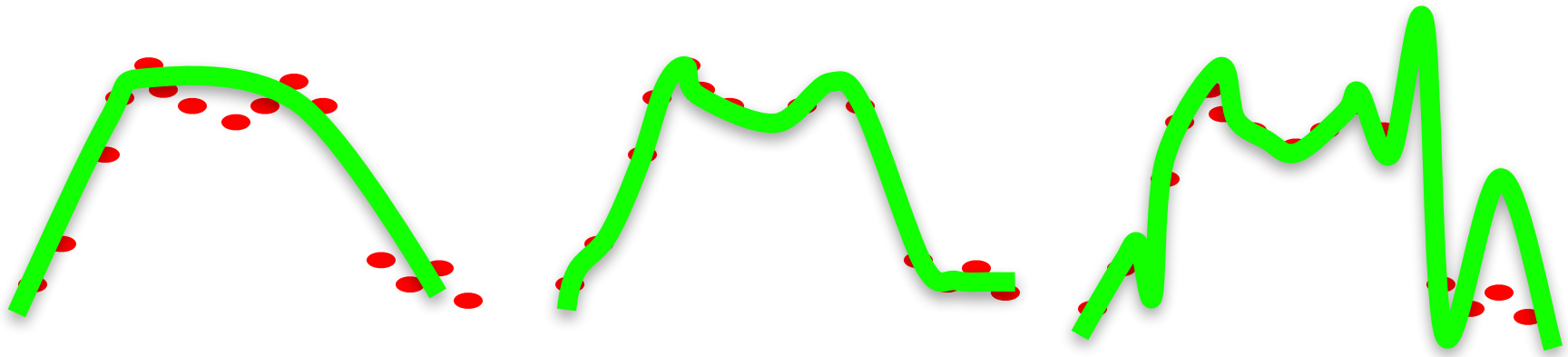
Underfitting

just right ⟷

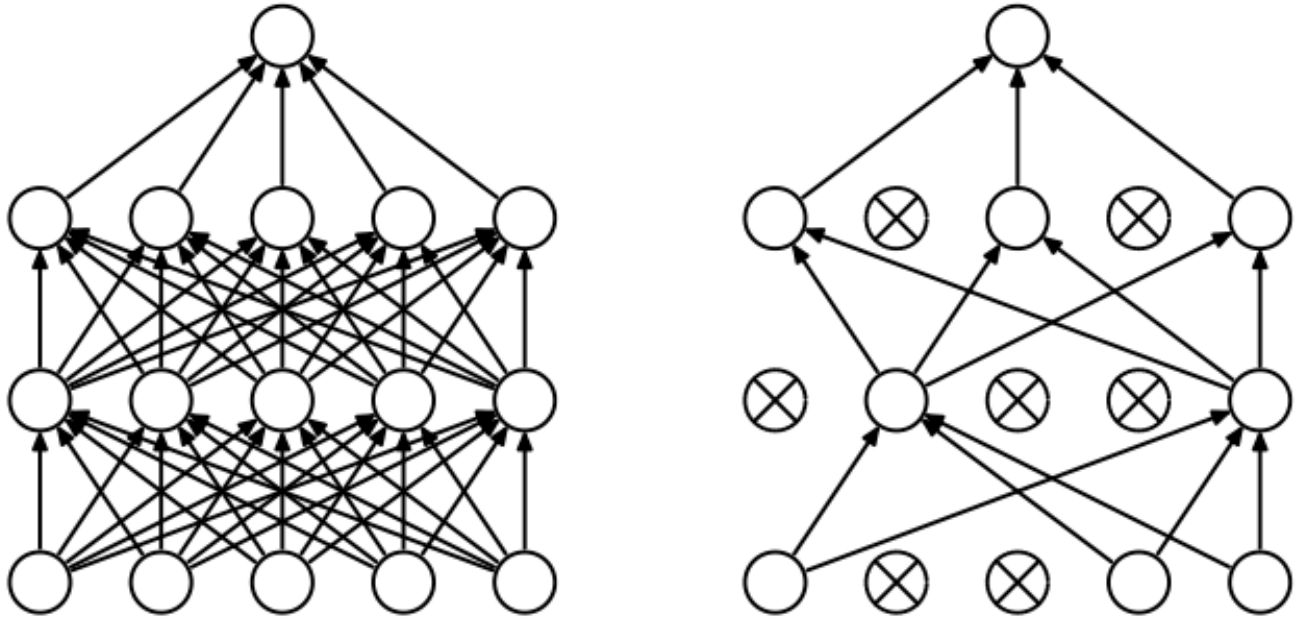
Overfitting



Regression



Dropout



(a) Standard Neural Net

(b) After applying dropout.

Each sample is processed by a ‘decimated’ neural net

Decimated nets: distinct classifiers

But: they should all do the same job

Dropout Performance

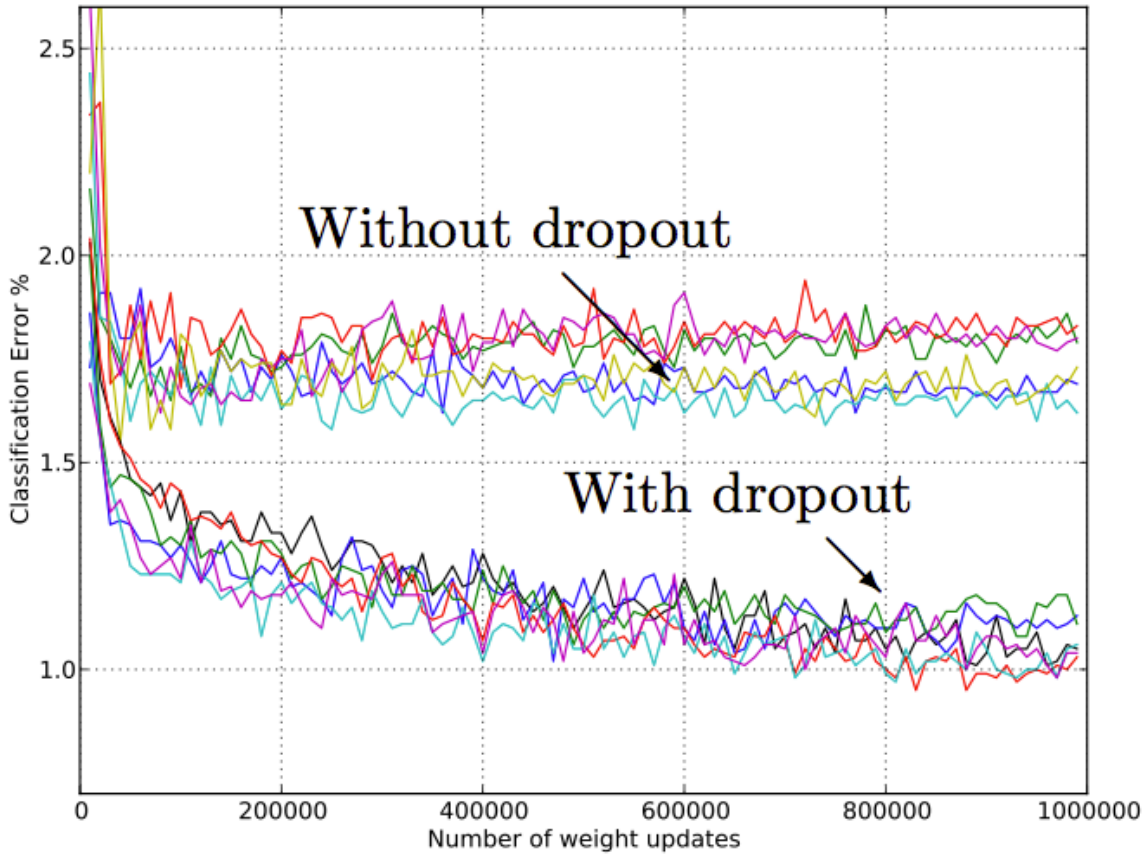


Figure 4: Test error for different architectures with and without dropout. The networks have 2 to 4 hidden layers each with 1024 to 2048 units.

Neural Network Training: Old & New Tricks

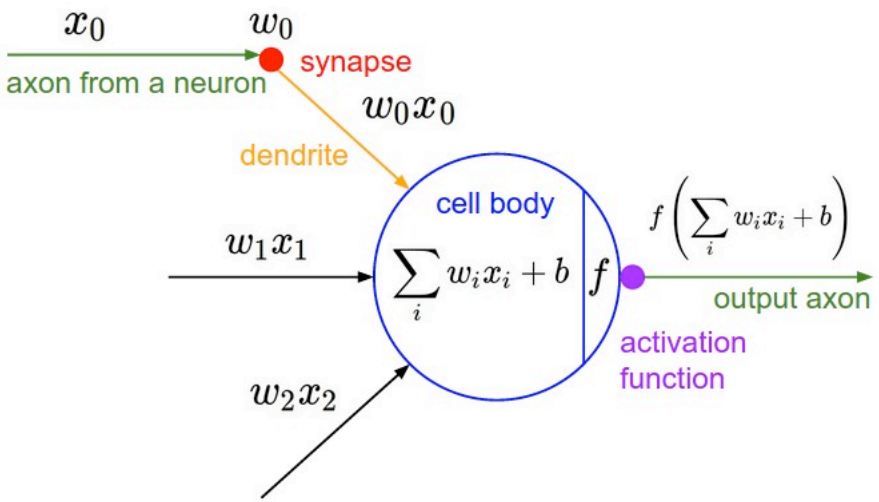
Old: (80's)

- Stochastic Gradient Descent, Momentum, "weight decay"

New: (last 5-6 years)

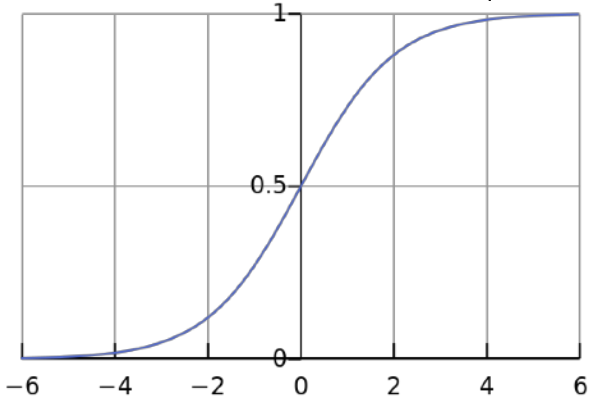
- Dropout
- ReLUs
- Batch Normalization

‘Neuron’: Cascade of Linear and Nonlinear Function



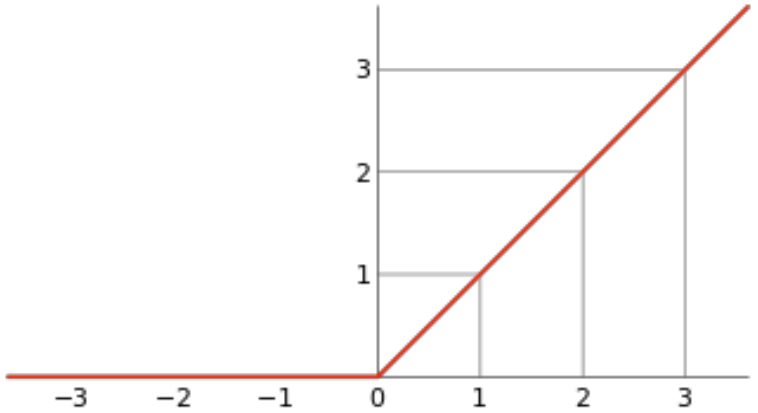
Sigmoidal (“logistic”)

$$g(a) = \frac{1}{1 + \exp(-a)}$$



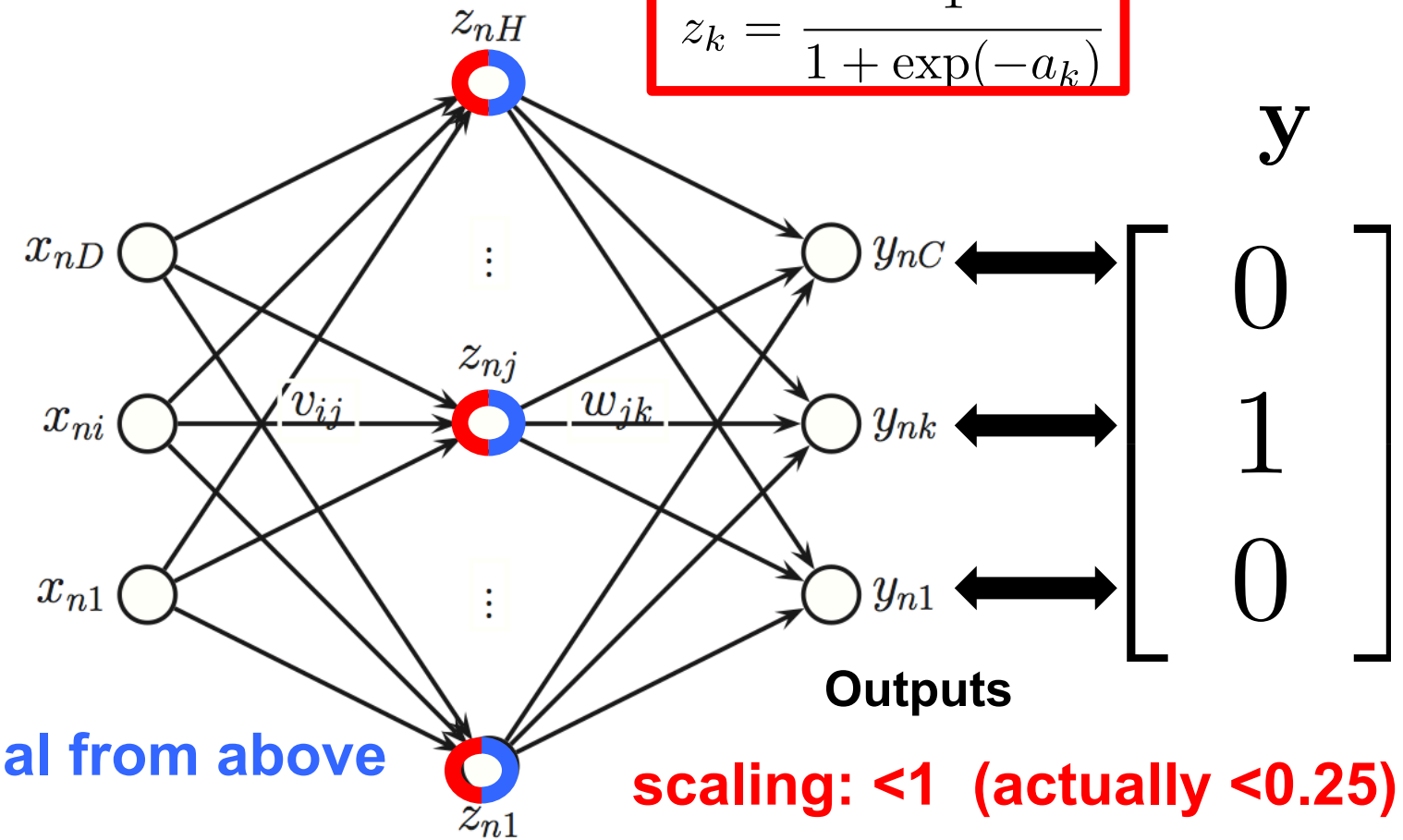
Rectified Linear Unit (RELU)

$$g(a) = \max(0, a)$$



Reminder: a network in backward mode

$$z_k = \frac{1}{1 + \exp(-a_k)}$$



Gradient signal from above

scaling: <1 (actually <0.25)

$$\frac{\partial l}{\partial a_k} = \sum_m \frac{\partial l}{\partial z_m}\frac{\partial z_m}{\partial a_k} = \frac{\partial l}{\partial z_k} g'(a_k) = \frac{\partial l}{\partial z_k} g(a_k)(1 - g(a_k))$$

Vanishing Gradients Problem

Gradient signal from above

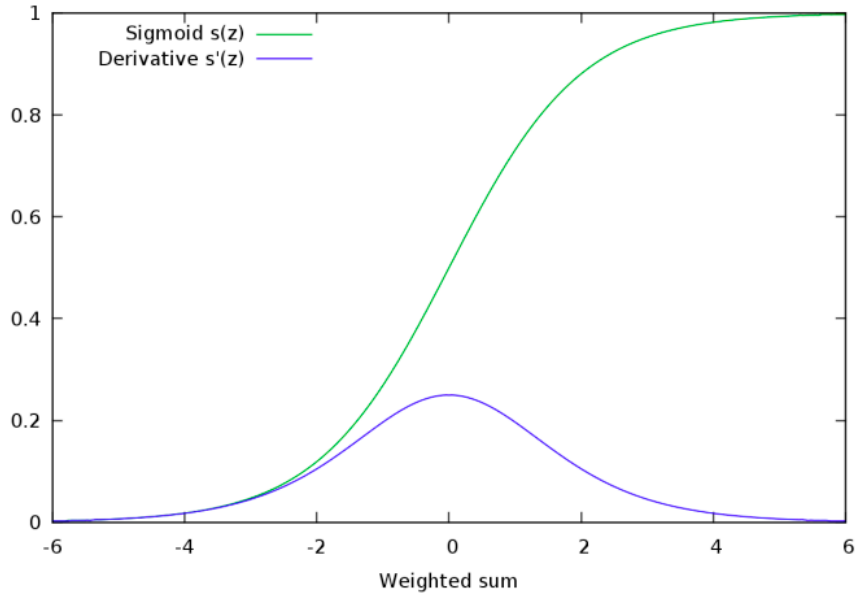
scaling: <1 (actually <0.25)

$$\frac{\partial l}{\partial a_k} = \sum_m \frac{\partial l}{\partial z_m}\frac{\partial z_m}{\partial a_k} = \frac{\partial l}{\partial z_k} g'(a_k) = \frac{\partial l}{\partial z_k} g(a_k)(1 - g(a_k))$$

Do this 10 times: updates in the first layers get minimal

Top layer knows what to do, lower layers “don’t get it”

Sigmoidal Unit: Signal is not getting through!

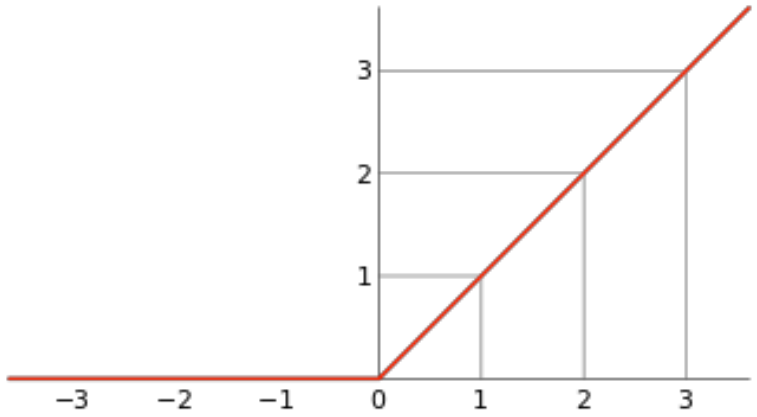


Vanishing Gradients Problem: ReLU Solves It

Gradient signal from above ⊙ **Scaling: {0,1}**

$$\frac{\partial l}{\partial a_k} = \sum_m \frac{\partial l}{\partial z_m}\frac{\partial z_m}{\partial a_k} = \frac{\partial l}{\partial z_k} g'(a_k)$$

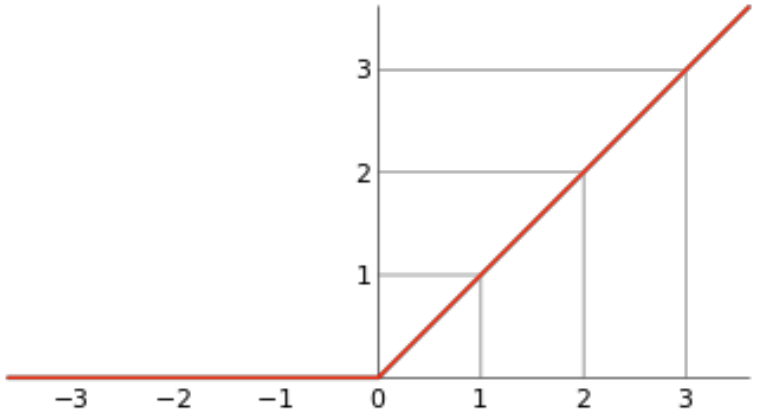
$$g(a) = \max(0, a)$$



$$g'(a) = \begin{cases} 1 & a > 0 \\ 0 & a < 0 \end{cases}$$

Activation Functions: ReLU & Co

$$g(a) = \max(0, a)$$



$$g'(a) = \begin{cases} 1 & a > 0 \\ 0 & a < 0 \end{cases}$$

Great! But… no gradient for negative half-space.

Lots of follow up work: LeakyReLU, eLU, etc.

Can improve results, but typically fine-tuning only.

Neural Network Training: Old & New Tricks

Old: (80's)

- Stochastic Gradient Descent, Momentum, "weight decay"

New: (last 5-6 years)

- Dropout
- ReLUs
- Batch Normalization

External Covariate Shift: your input changes

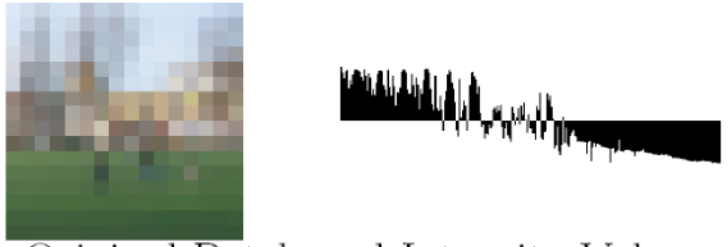
10 am

2pm

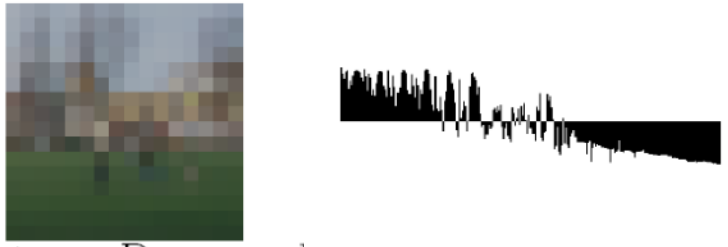
7pm

“Whitening”: Set Mean = 0, Variance = 1

Photometric transformation: $I \rightarrow a\,I + b$



Original Patch and Intensity Values



Brightness Decreased

Contrast increased,

- Make each patch have zero mean:

$$\mu = \frac{1}{N}\sum_{x,y} I(x,y)$$

$$Z(x,y) = I(x,y) - \mu$$

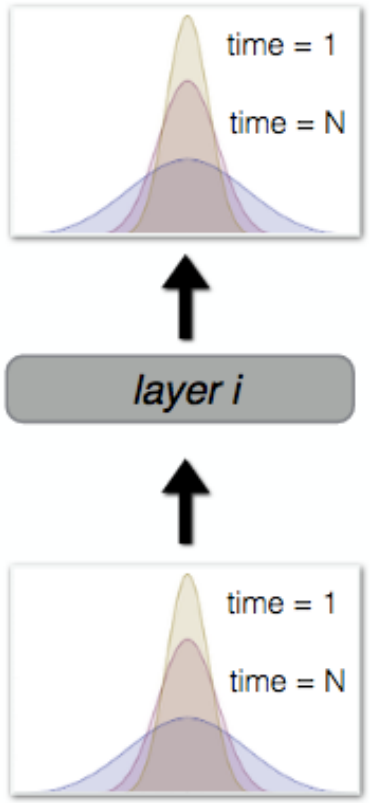
- Then make it have unit variance:

$$\sigma^2 = \frac{1}{N}\sum_{x,y} Z(x,y)^2$$

$$ZN(x,y) = \frac{Z(x,y)}{\sigma}$$

Internal Covariate Shift

Neural network activations during training: moving target



Batch Normalization

Whiten-as-you-go:

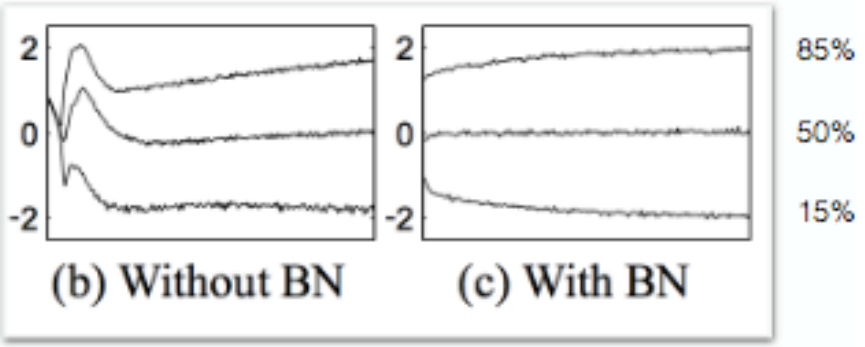
- Normalize the activations in each layer within a mini-batch.
- Learn the mean and variance (γ, β) of each layer as parameters

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

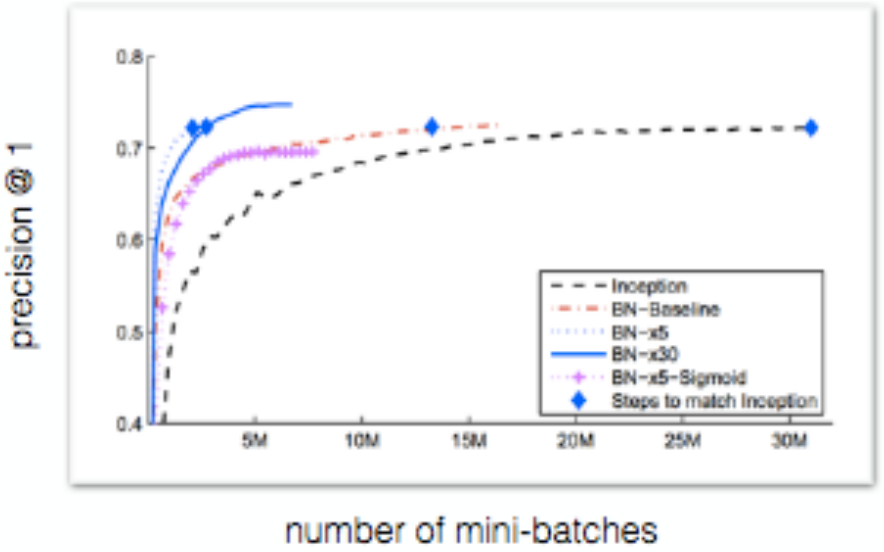
$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$



Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift
S Ioffe and C Szegedy (2015)

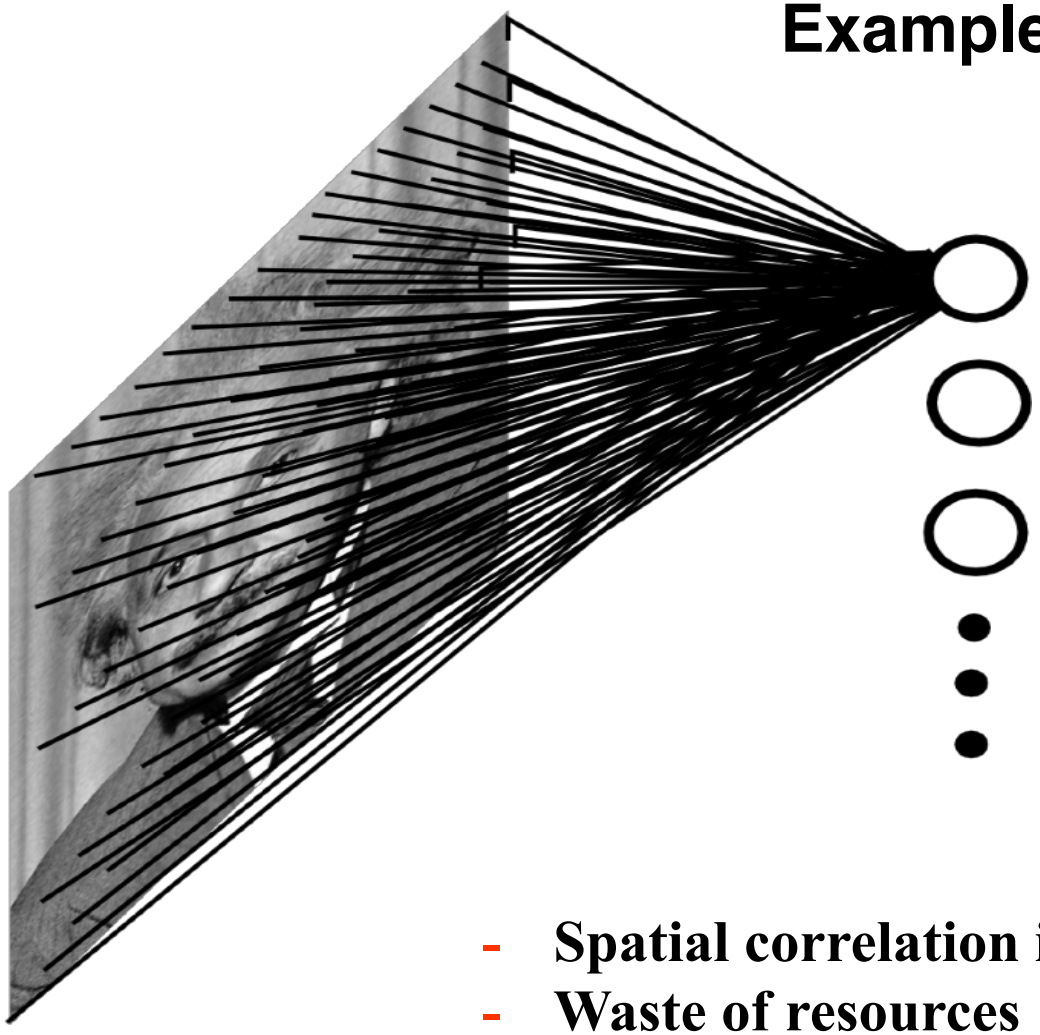
Batch Normalization: Used in all current systems

- Multi-layer CNN's train faster with fewer data samples (15x).
- Employ faster learning rates and less network regularizations.
- Achieves state of the art results on ImageNet.



Convolutional Neural Networks

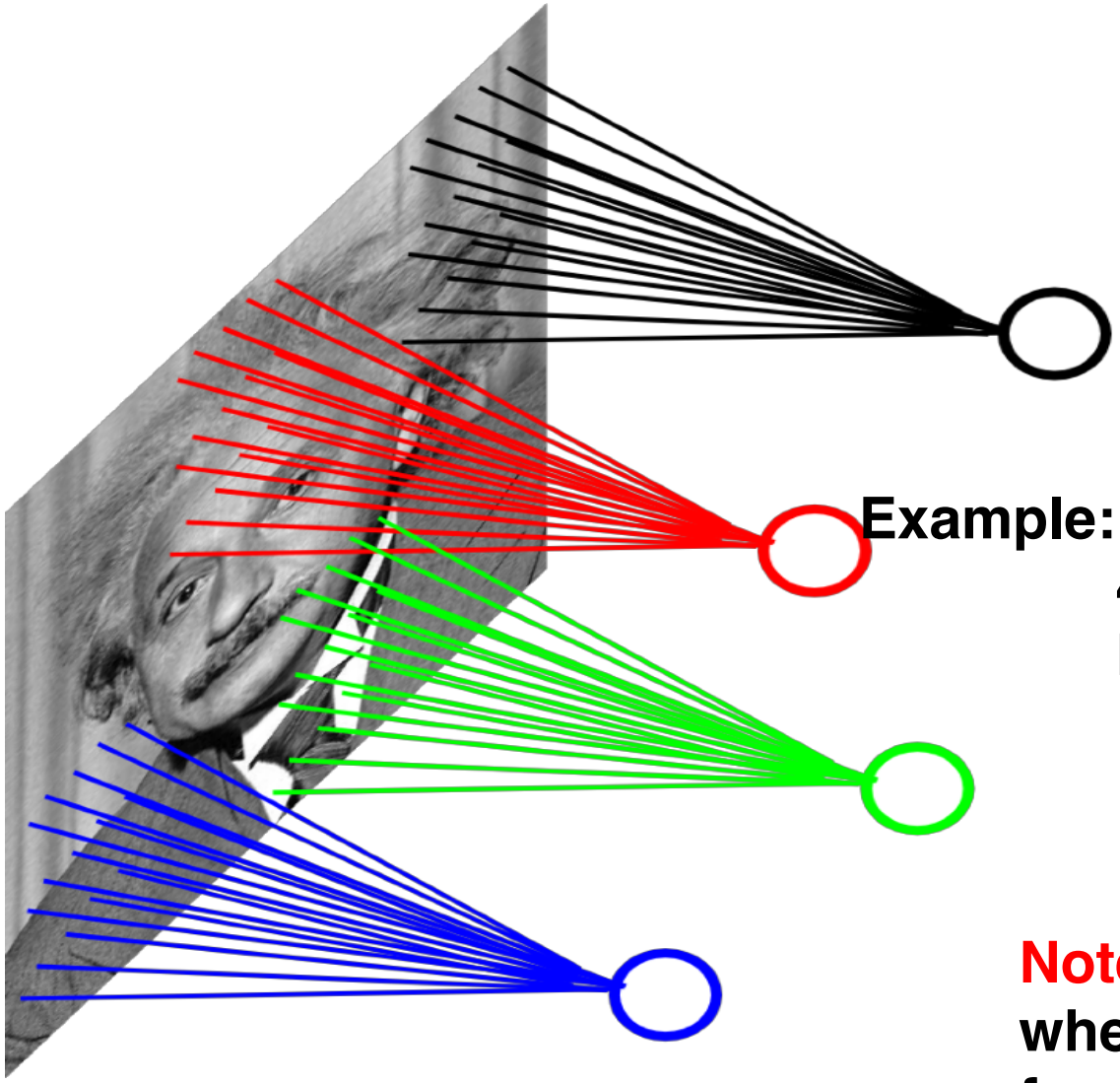
Fully-connected Layer



Example: 200x200 image
40K hidden units
~1.6B parameters!!!

- **Spatial correlation is local**
- **Waste of resources**
- **We don't have enough training samples anyway…**

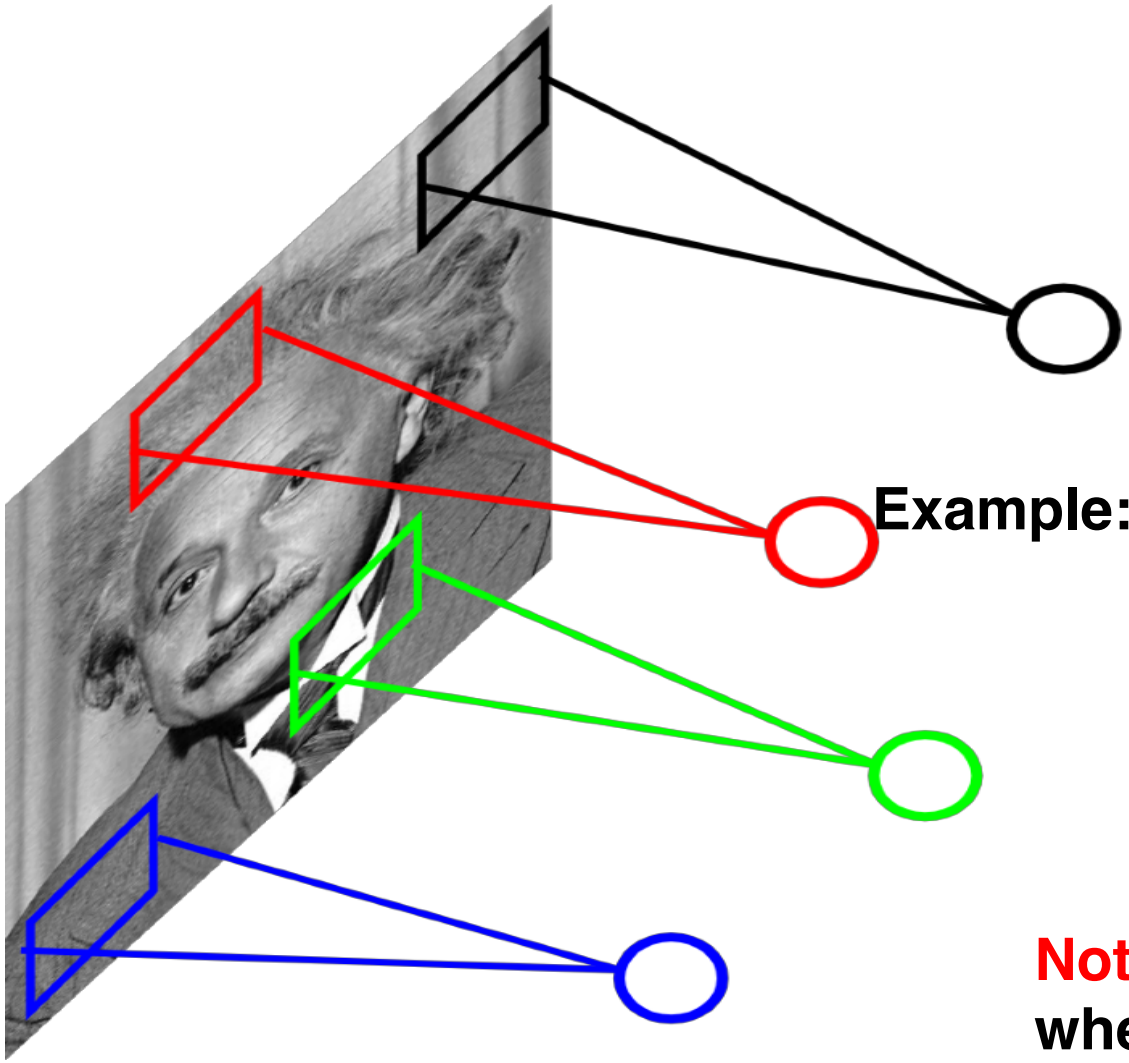
Locally-connected Layer



Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters

Note: This parameterization is good when input image is registered (e.g., face recognition).

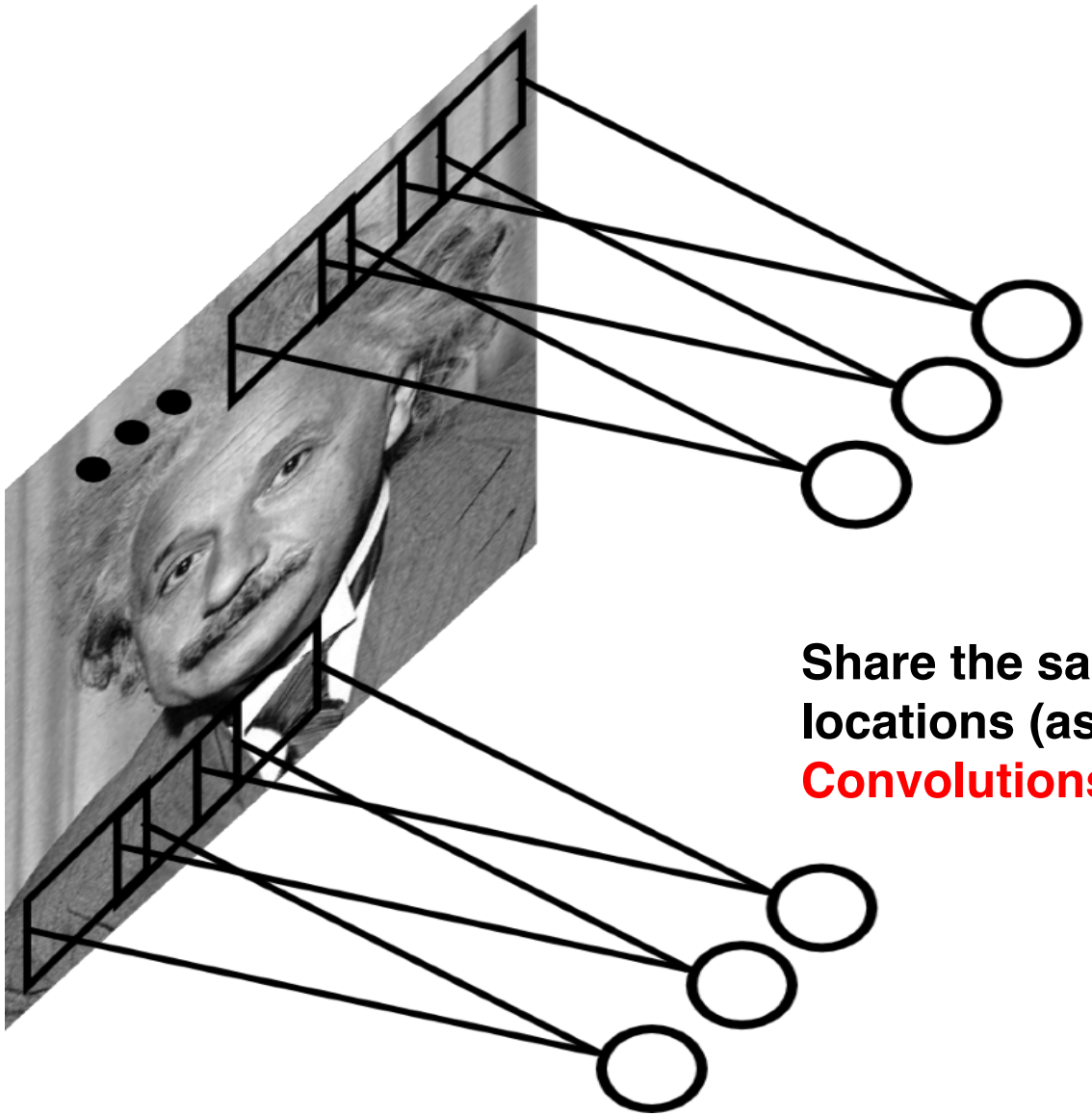
Locally-connected Layer



Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters

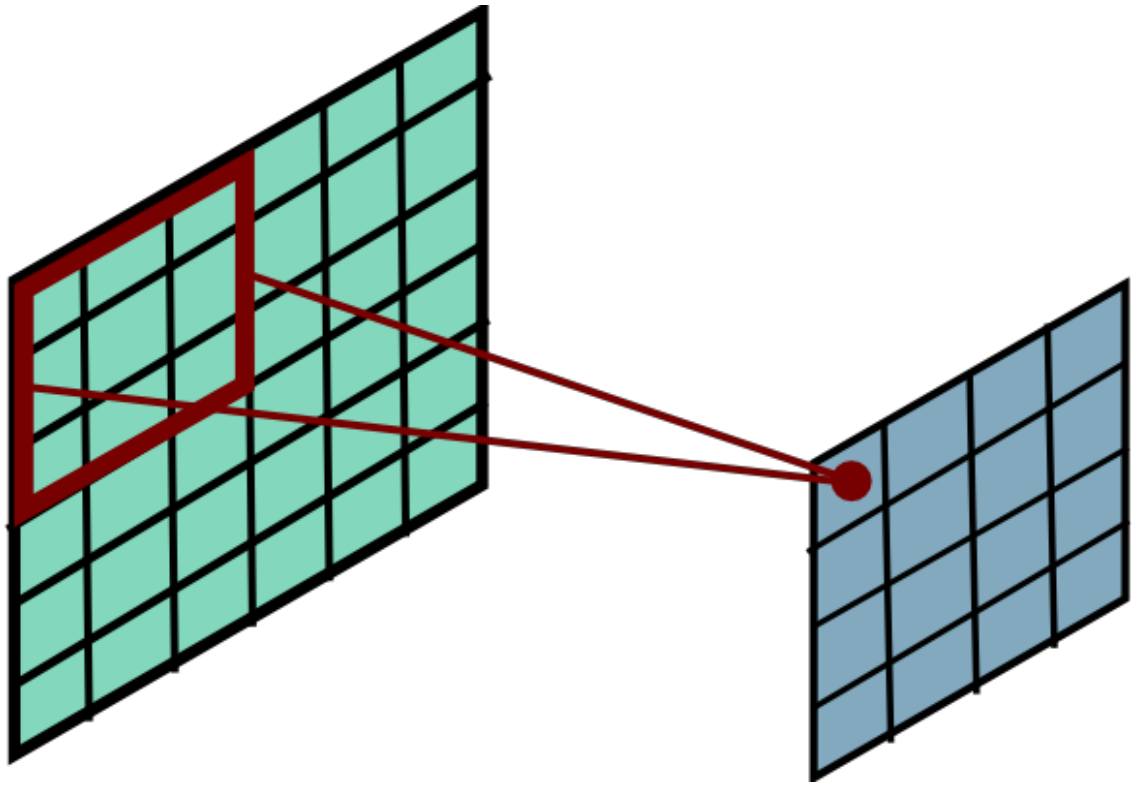
Note: This parameterization is good when input image is registered (e.g., face recognition).

Convolutional Layer

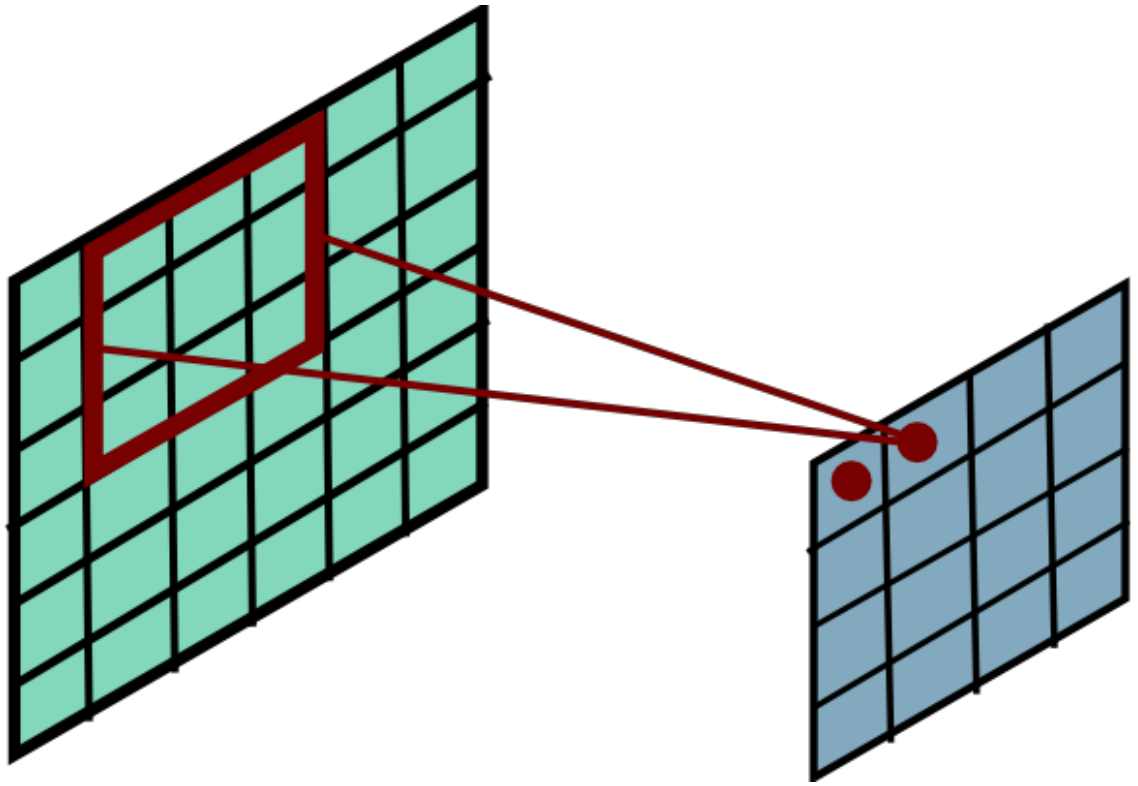


Share the same parameters across different locations (assuming input is stationary):
Convolutions with learned kernels

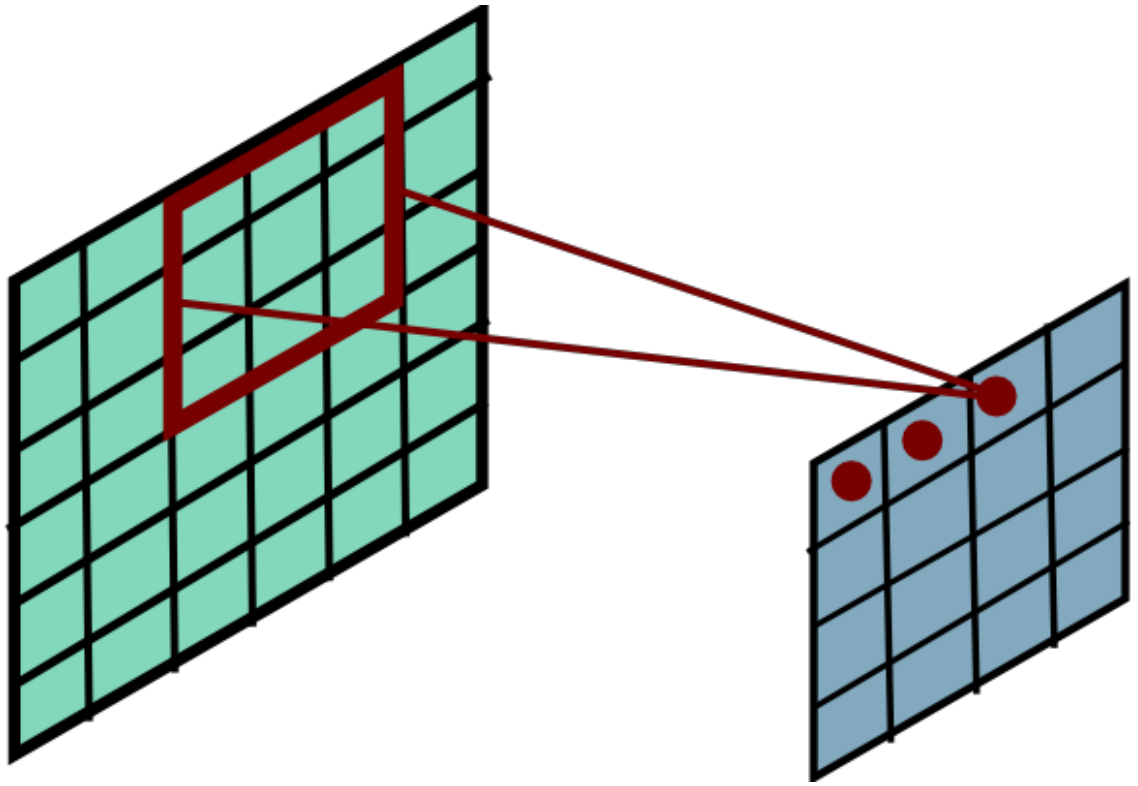
Convolutional Layer



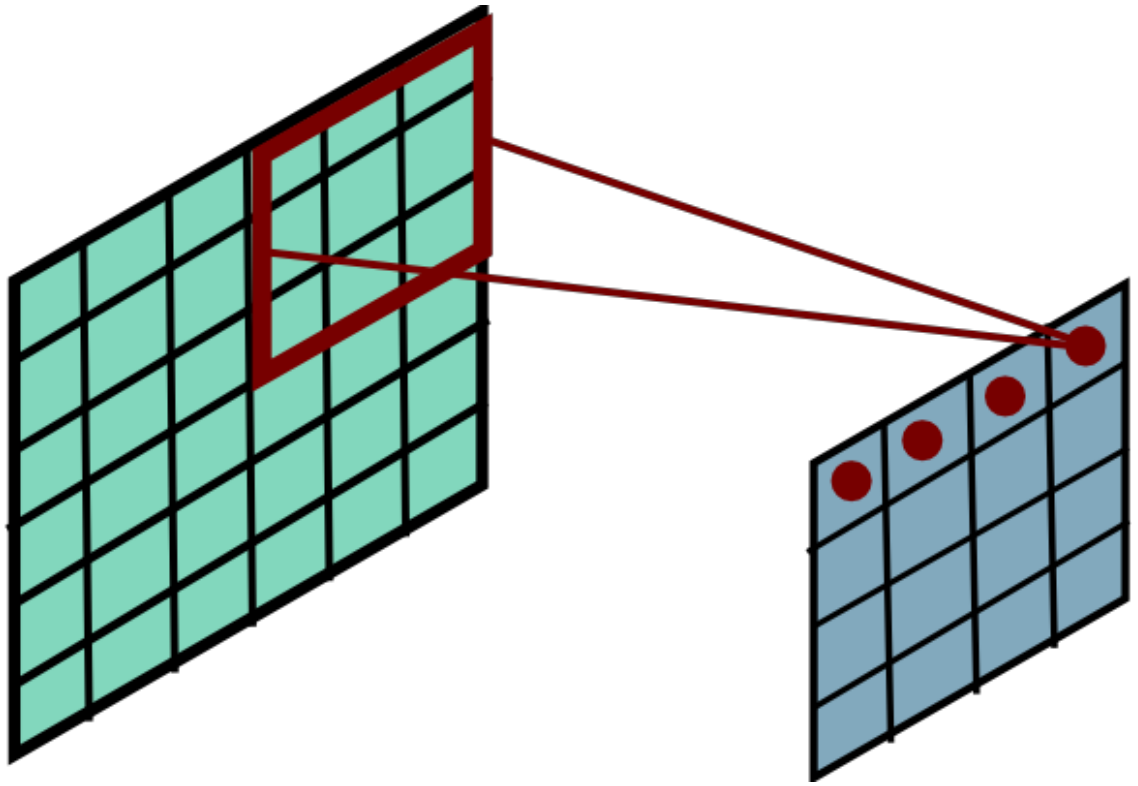
Convolutional Layer



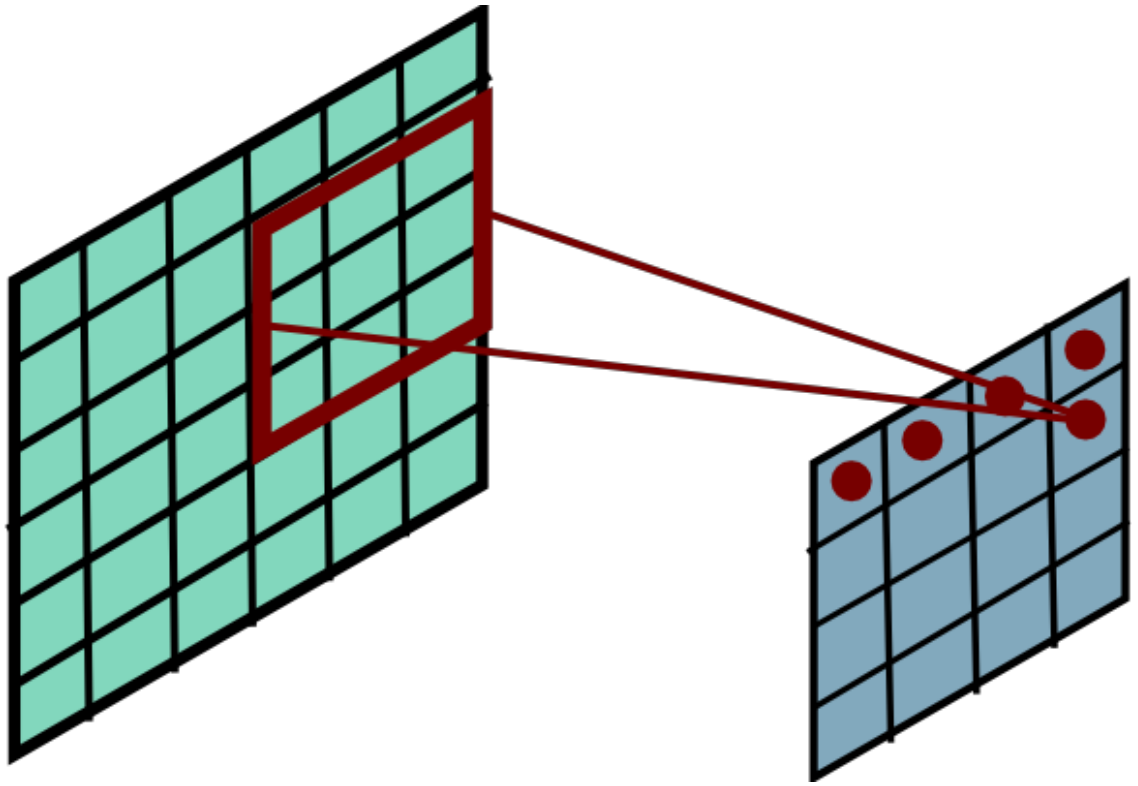
Convolutional Layer



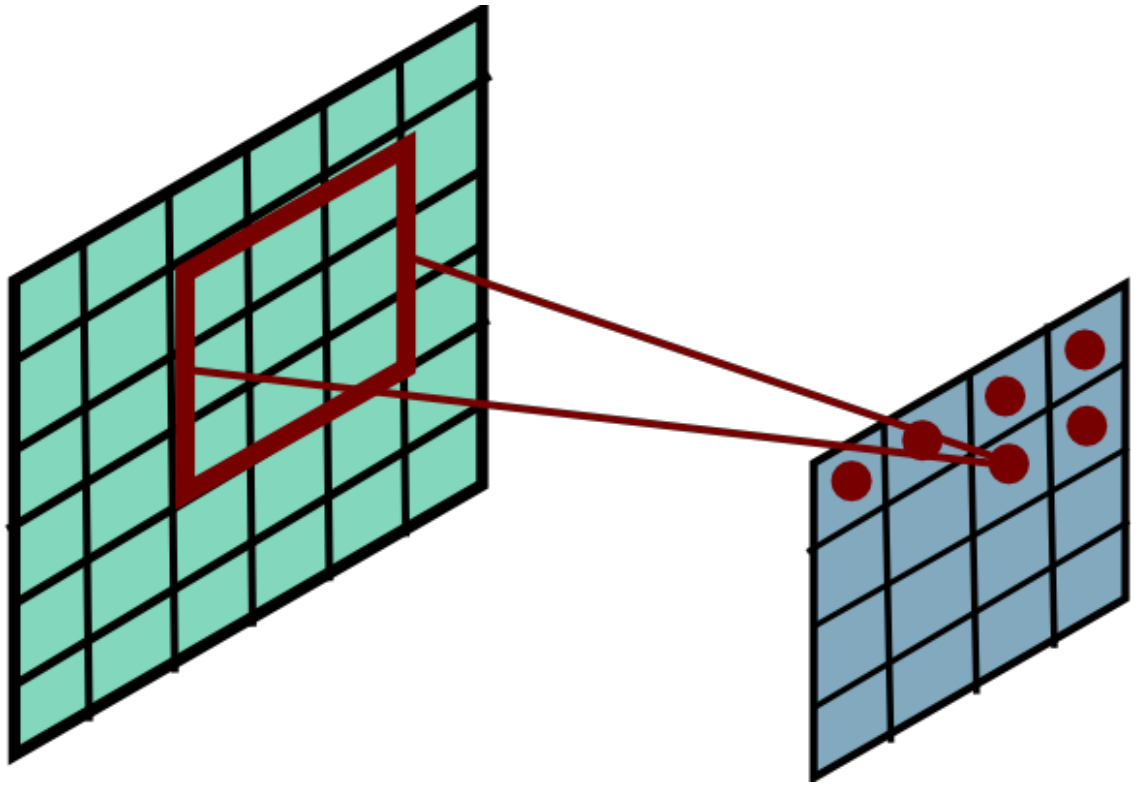
Convolutional Layer



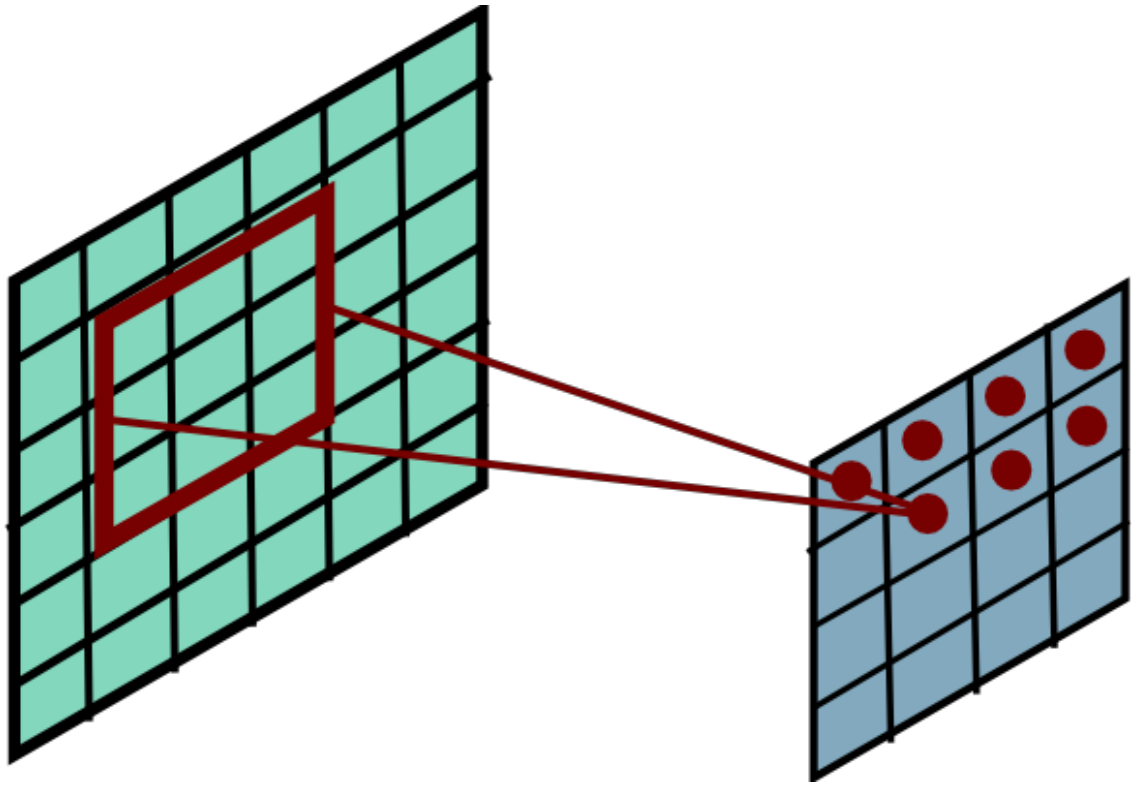
Convolutional Layer



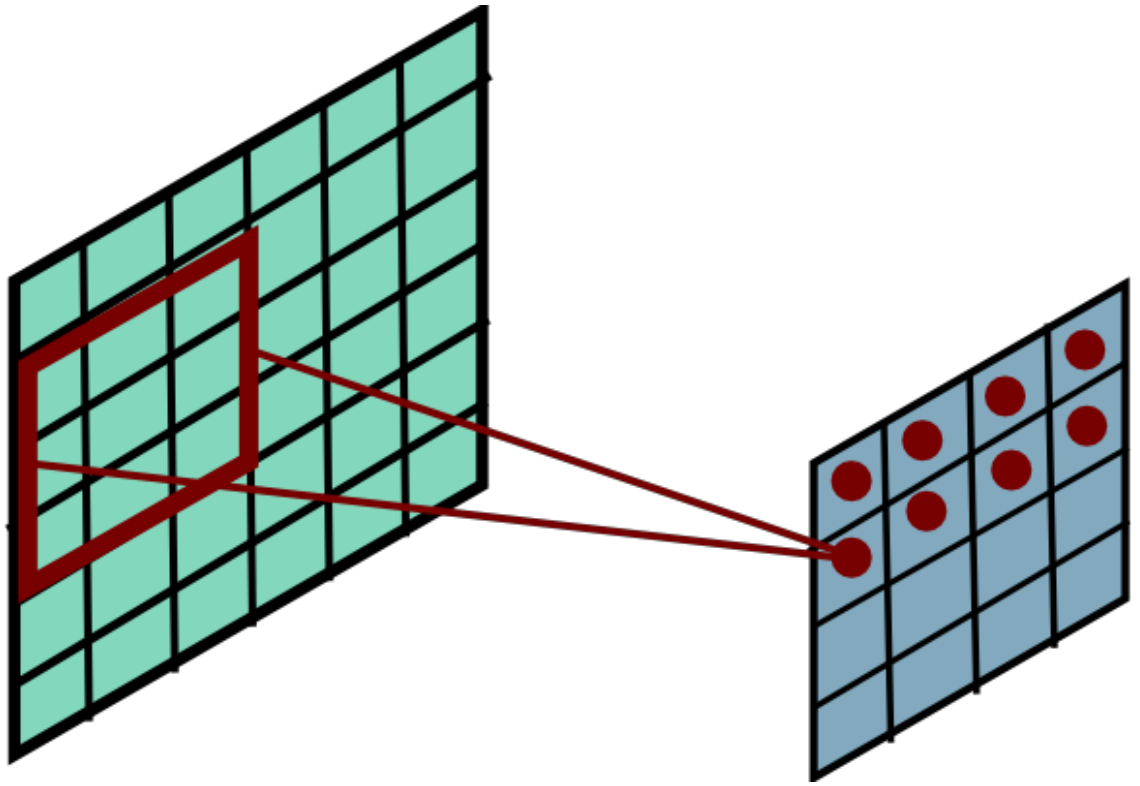
Convolutional Layer



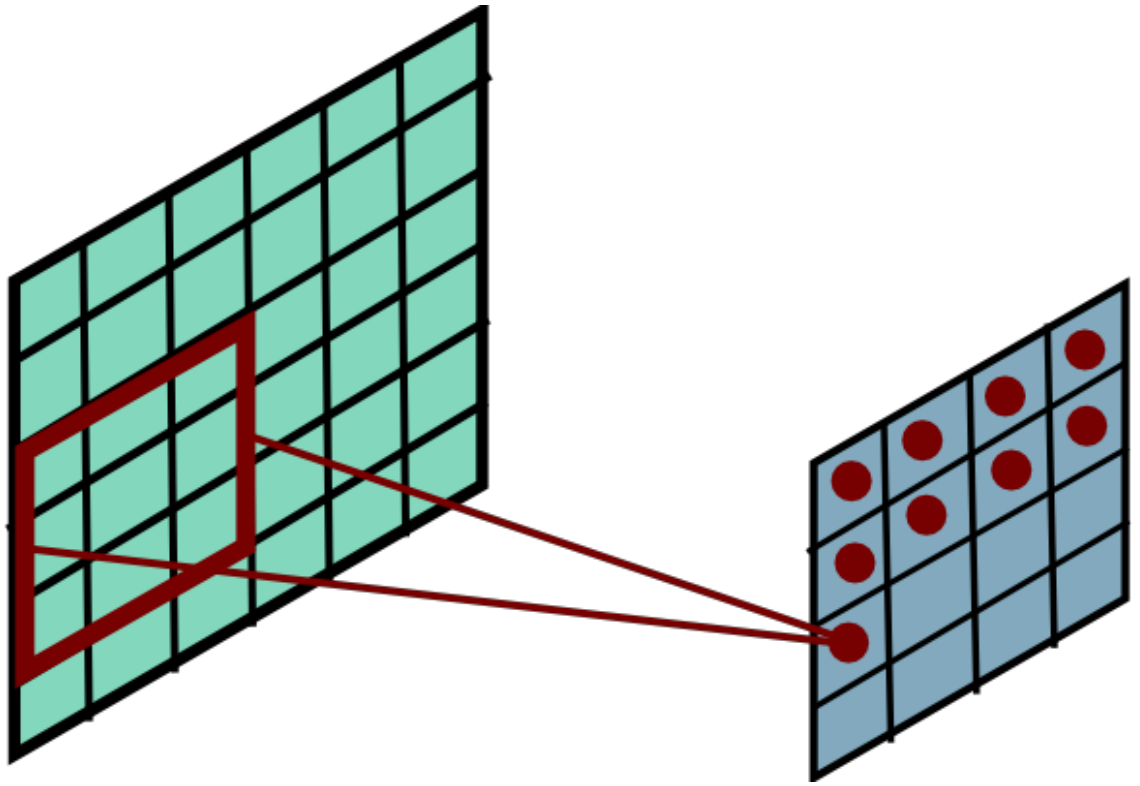
Convolutional Layer



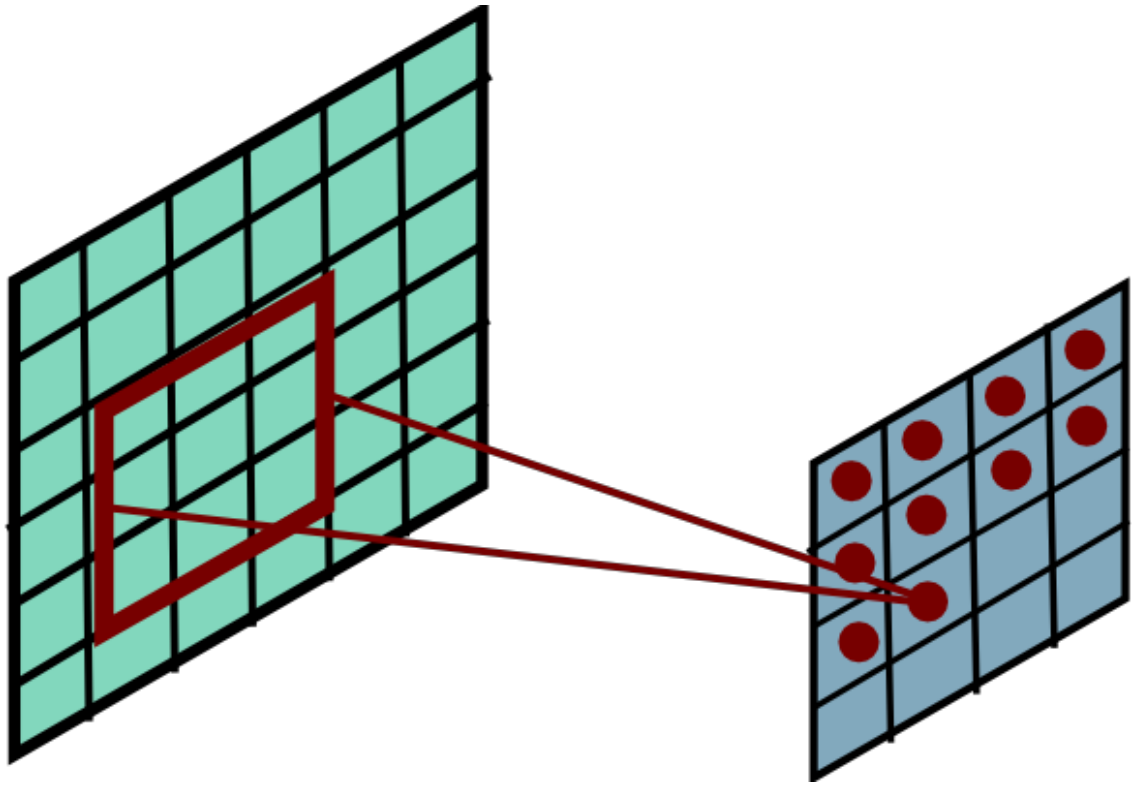
Convolutional Layer



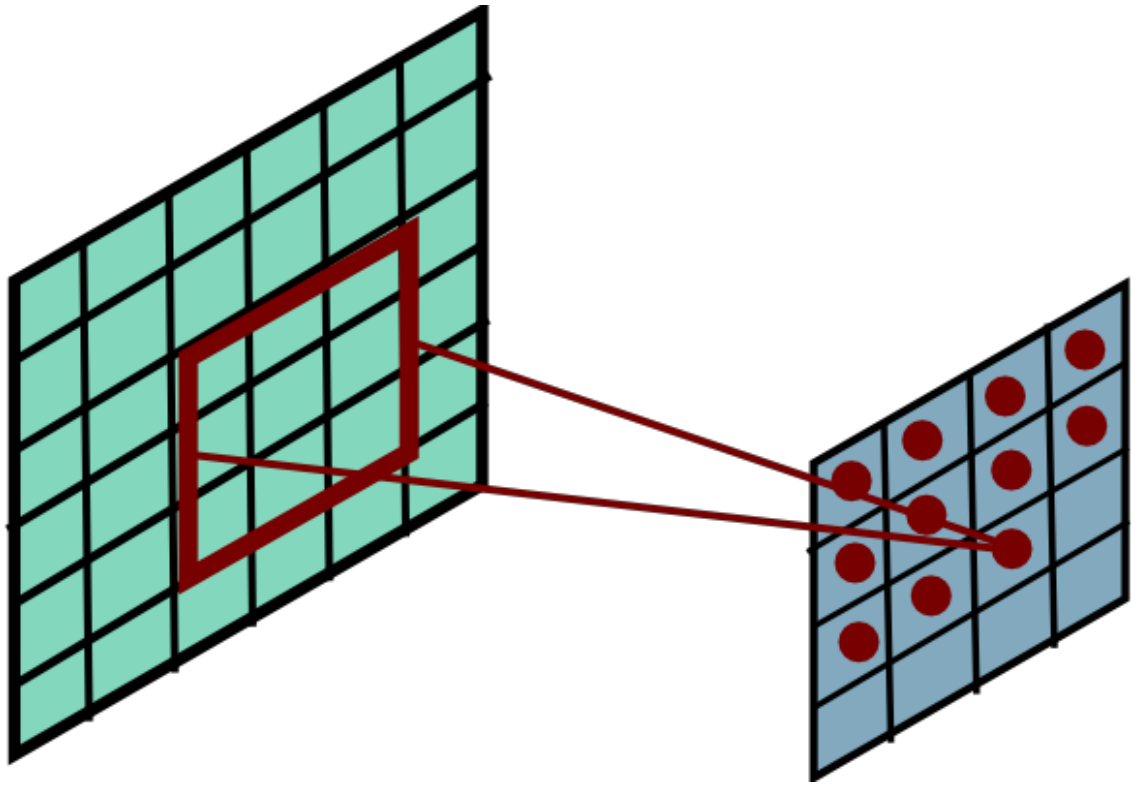
Convolutional Layer



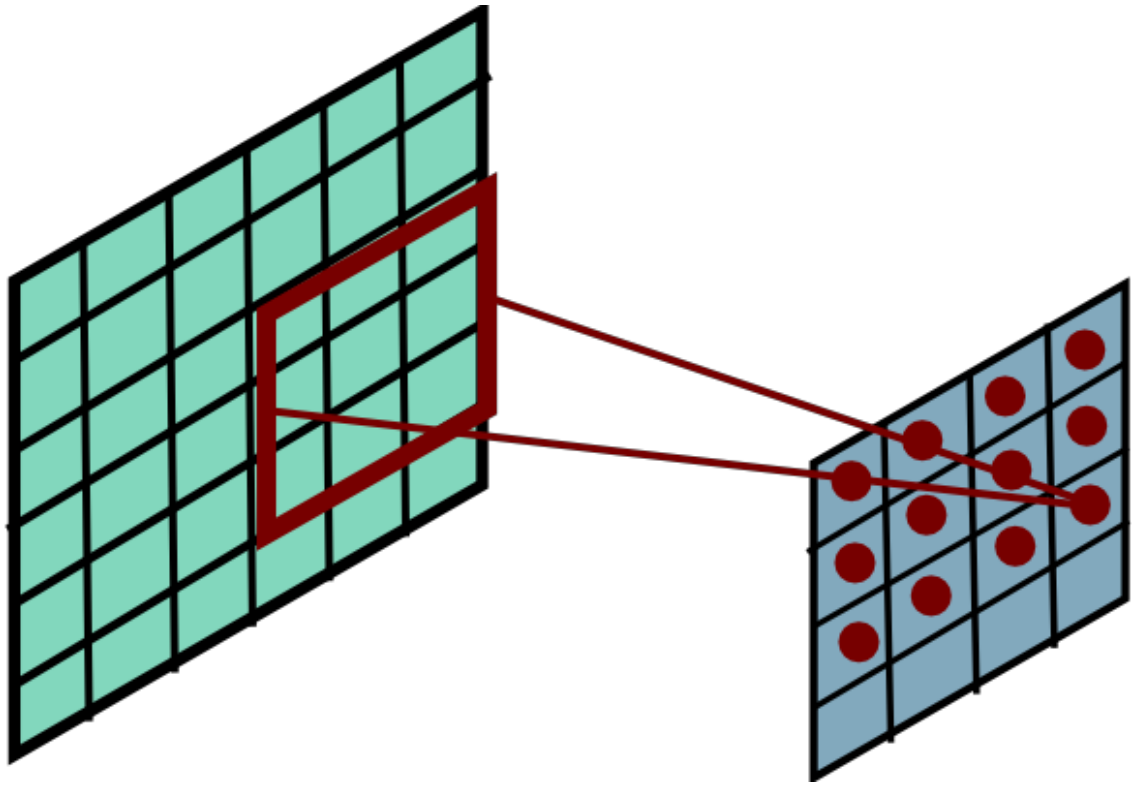
Convolutional Layer



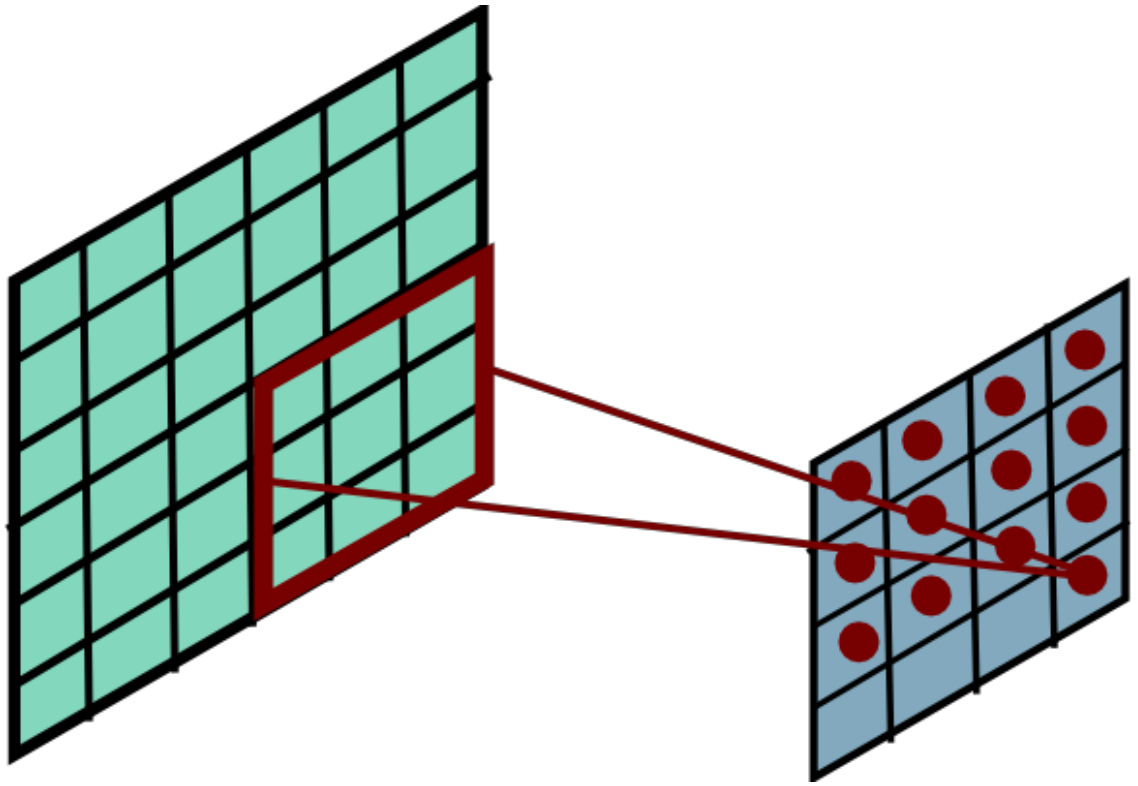
Convolutional Layer



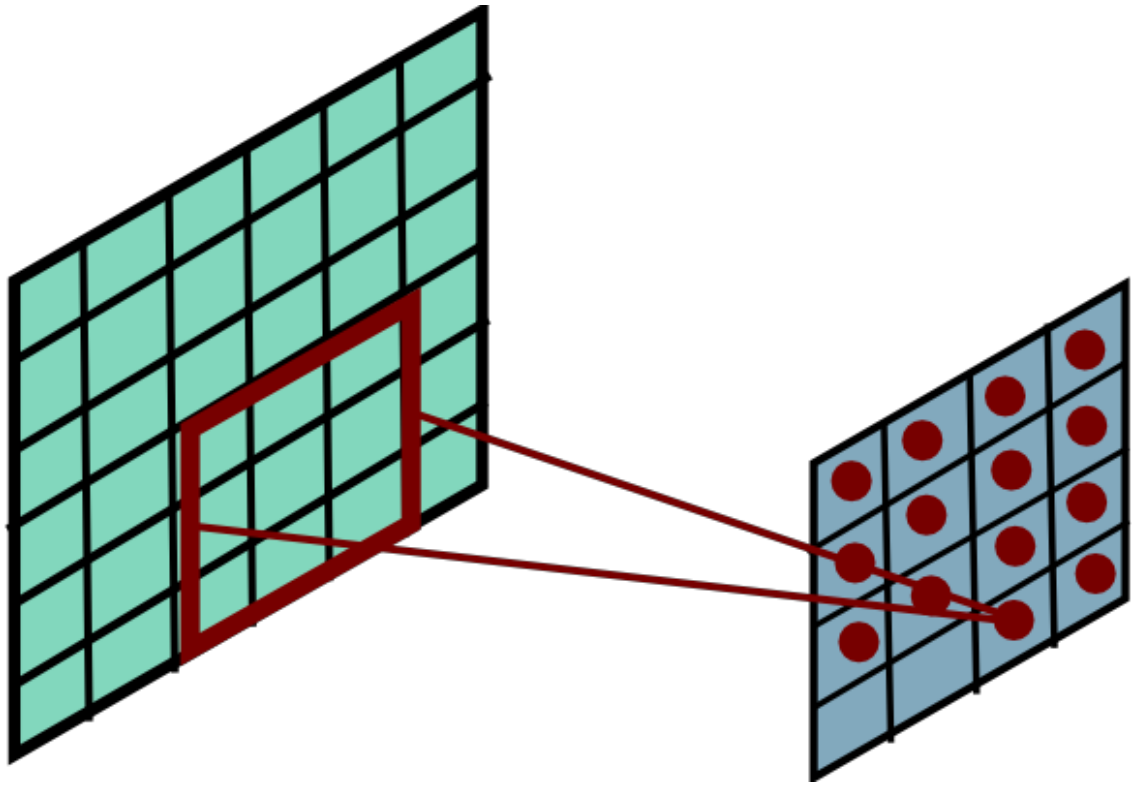
Convolutional Layer



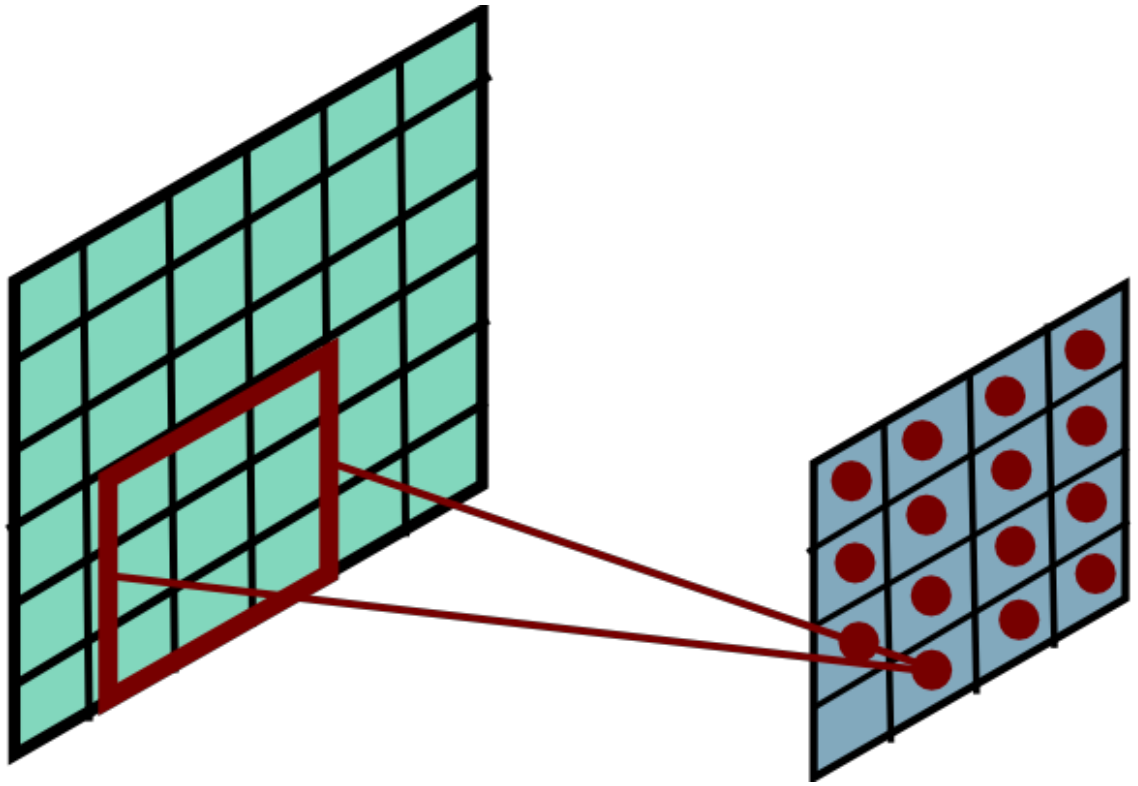
Convolutional Layer



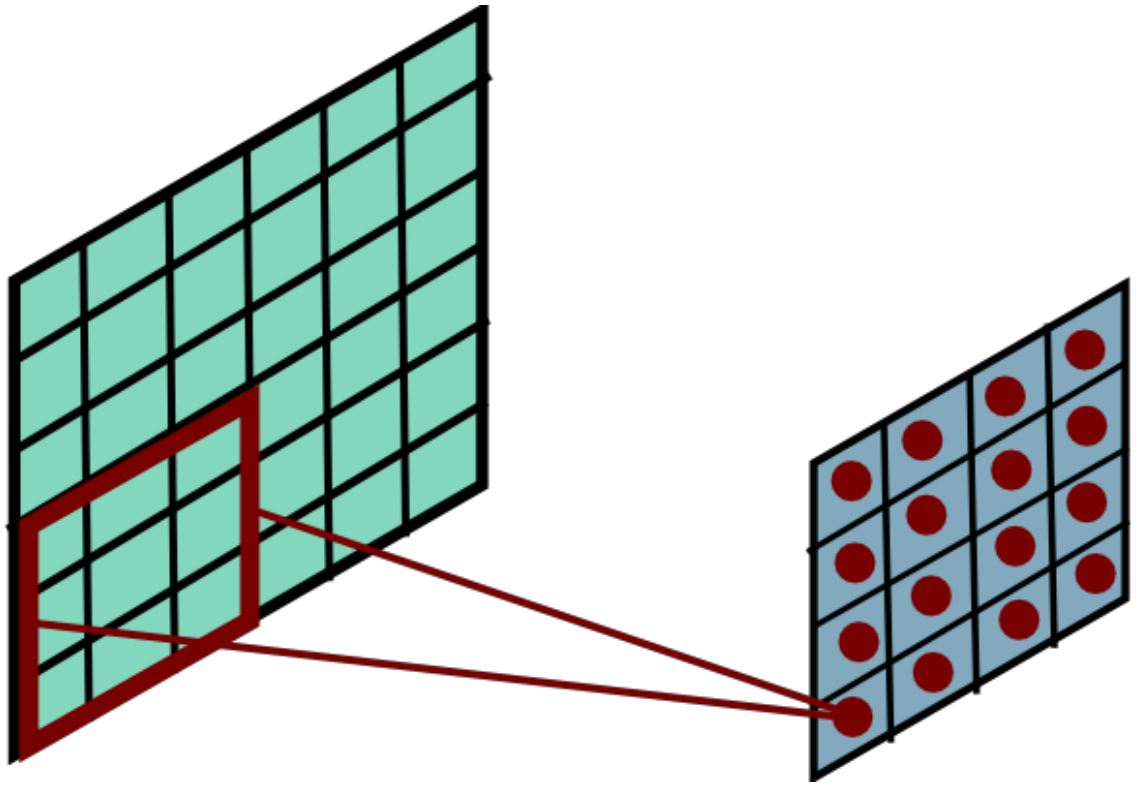
Convolutional Layer



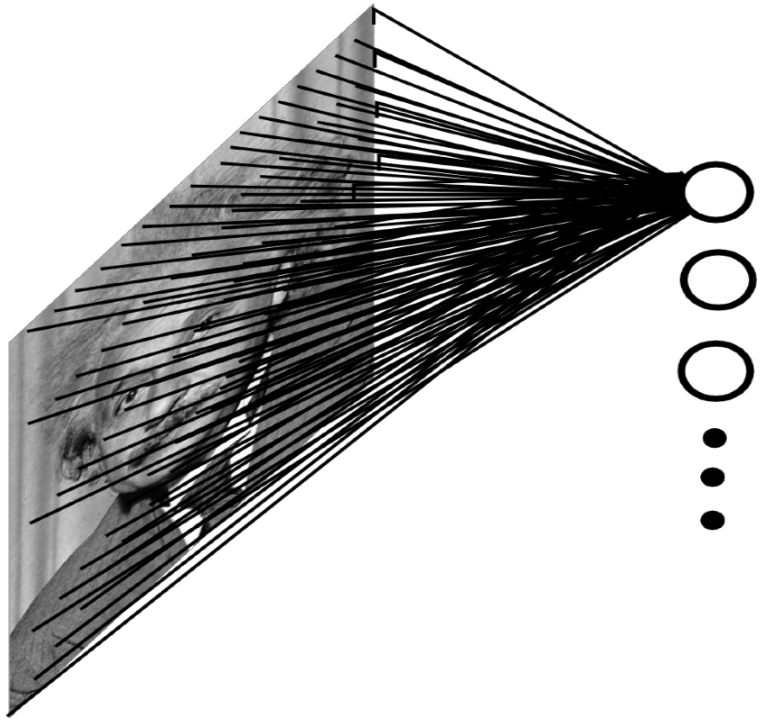
Convolutional Layer



Convolutional Layer



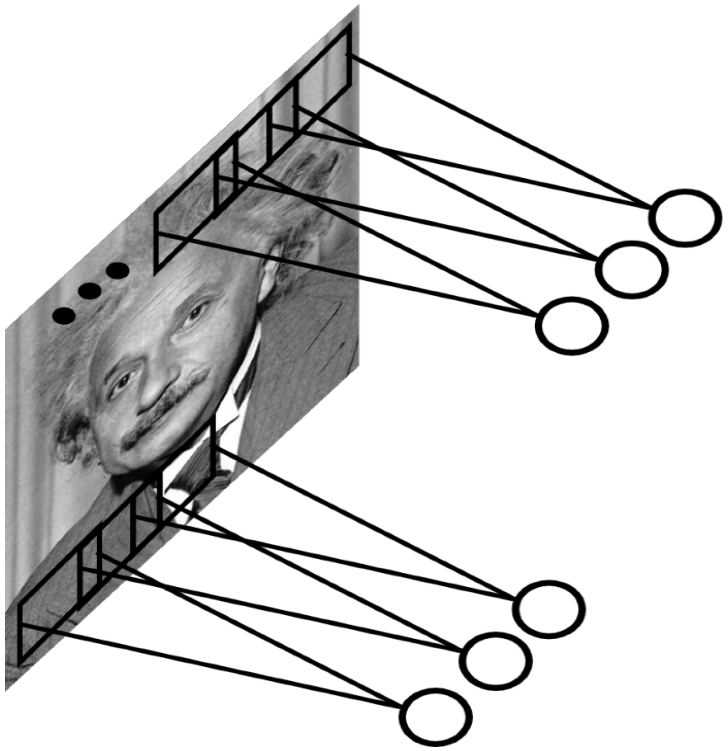
Fully-connected layer



#of parameters: K^2

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ \vdots \\ y_K \end{bmatrix} = \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} & w_{1,4} & \dots & w_{1,K} \\ w_{2,1} & w_{2,2} & w_{2,3} & w_{2,4} & \dots & w_{2,K} \\ w_{3,1} & w_{3,2} & w_{3,3} & w_{3,4} & \dots & w_{3,K} \\ w_{4,1} & w_{4,2} & w_{4,3} & w_{4,4} & \dots & w_{4,K} \\ & & \vdots & & & \\ w_{K,1} & w_{K,2} & w_{K,3} & w_{K,4} & \dots & w_{K,K} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \vdots \\ x_K \end{bmatrix}$$

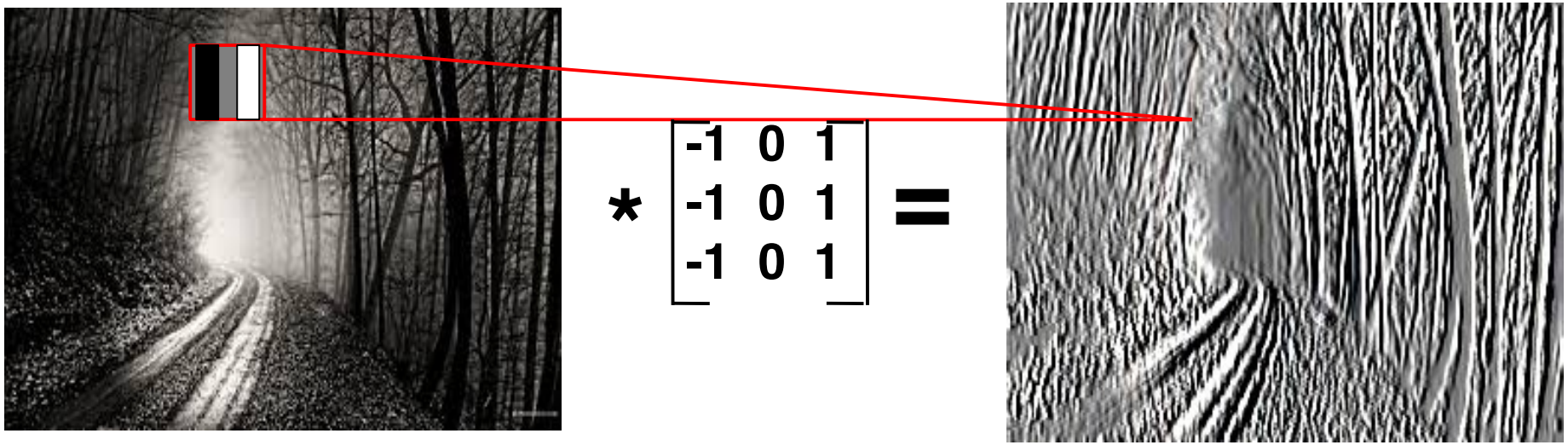
Convolutional layer



#of parameters: size of window

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ \vdots \\ y_K \end{bmatrix} = \begin{bmatrix} w_0 & w_1 & w_2 & 0 & \dots & 0 \\ 0 & w_0 & w_1 & w_2 & \dots & 0 \\ 0 & 0 & w_0 & w_1 & \dots & 0 \\ 0 & 0 & 0 & w_0 & \dots & 0 \\ & & \vdots & & & \\ 0 & 0 & 0 & 0 & \dots & w_0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \vdots \\ x_K \end{bmatrix}$$

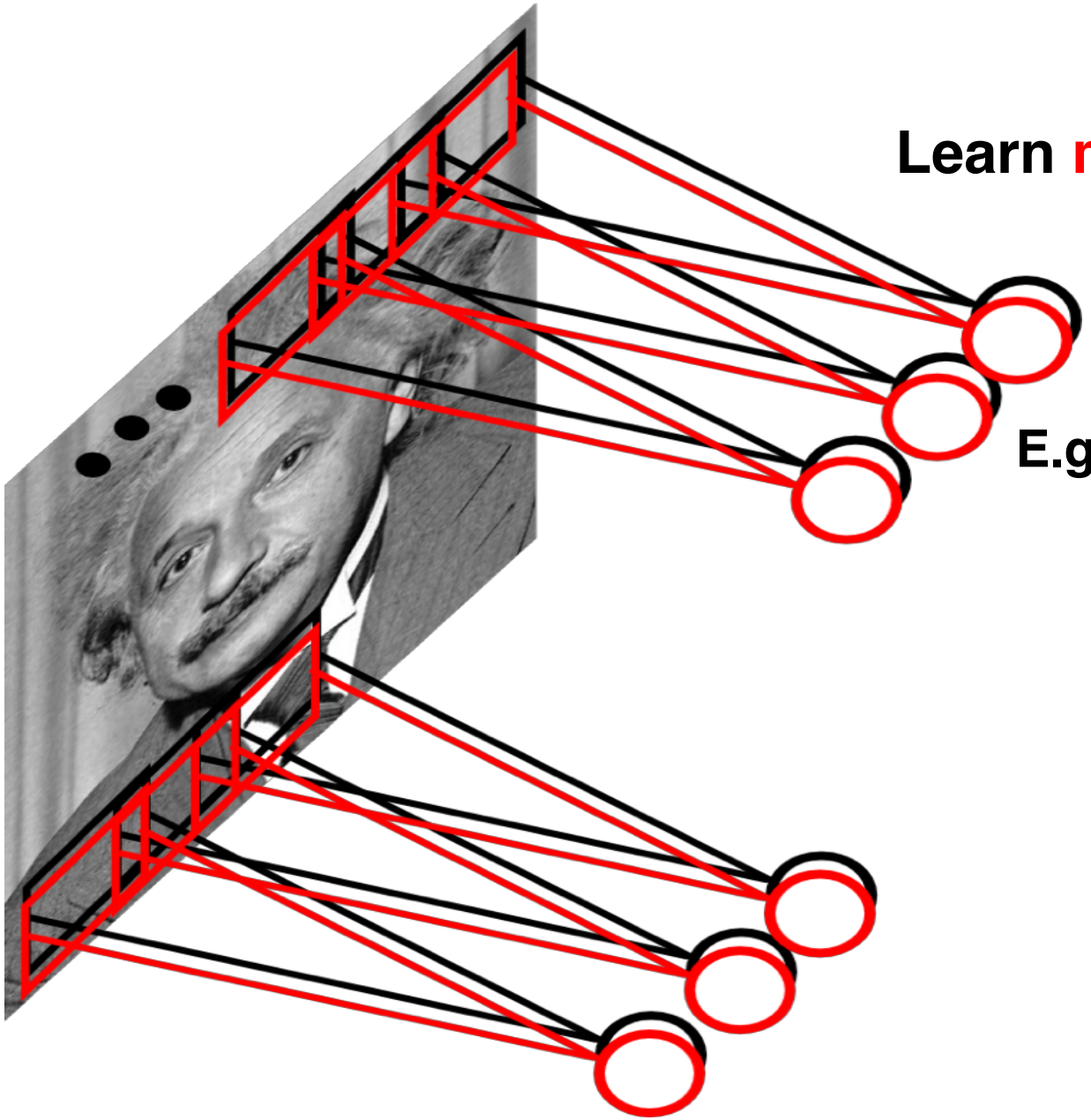
Convolutional layer



Code example

Learning an edge filter

Convolutional layer



Learn multiple filters.

E.g.: 200x200 image
100 Filters
Filter size: 10x10
10K parameters

Convolutional layer with ReLU activation

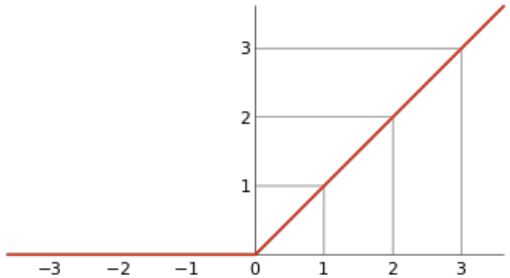
$$h_i^n = \max\left\{0, \sum_{j=1}^{\#\text{input channels}} h_j^{n-1} * w_{ij}^n\right\}$$

output feature map

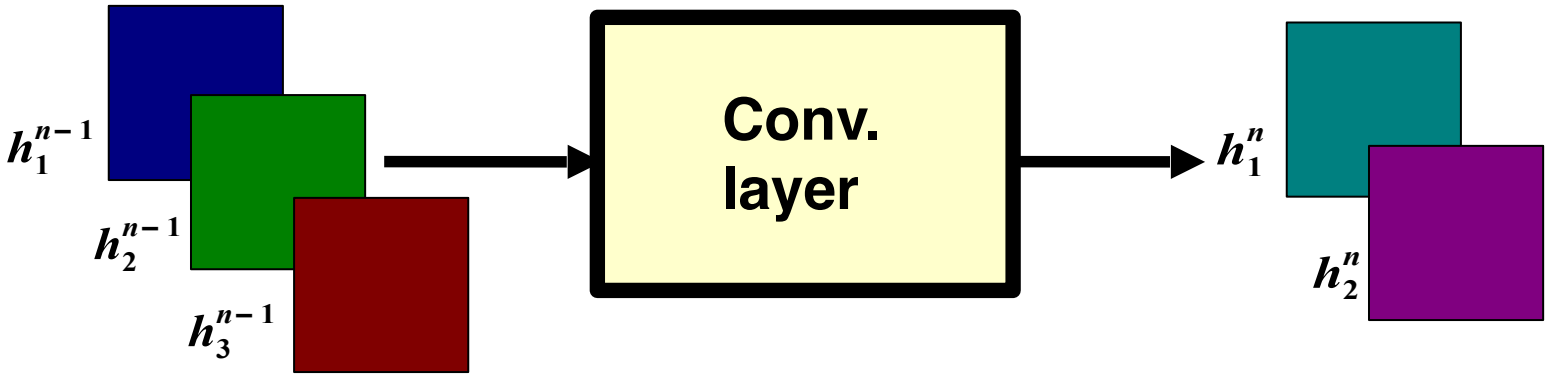
ReLU

input feature map

kernel



Activation



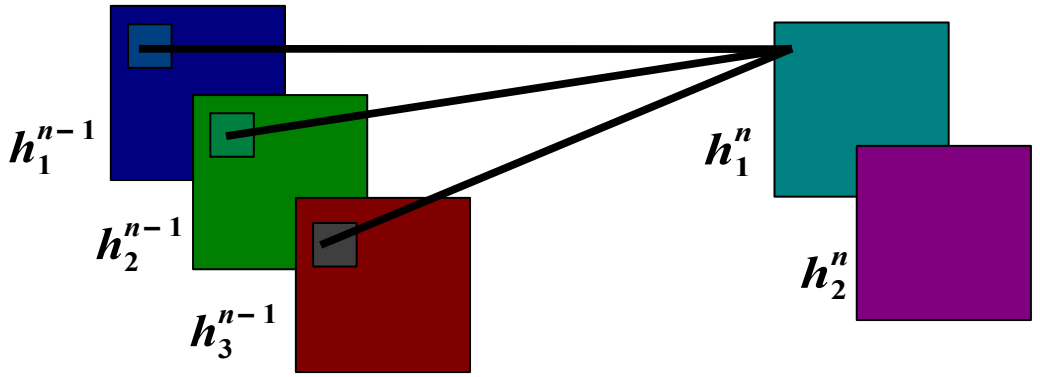
Convolutional layer with ReLU activation

$$h_i^n = \max\left\{0, \sum_{j=1}^{\#\text{input channels}} h_j^{n-1} * w_{ij}^n\right\}$$

output feature map

input feature map

kernel



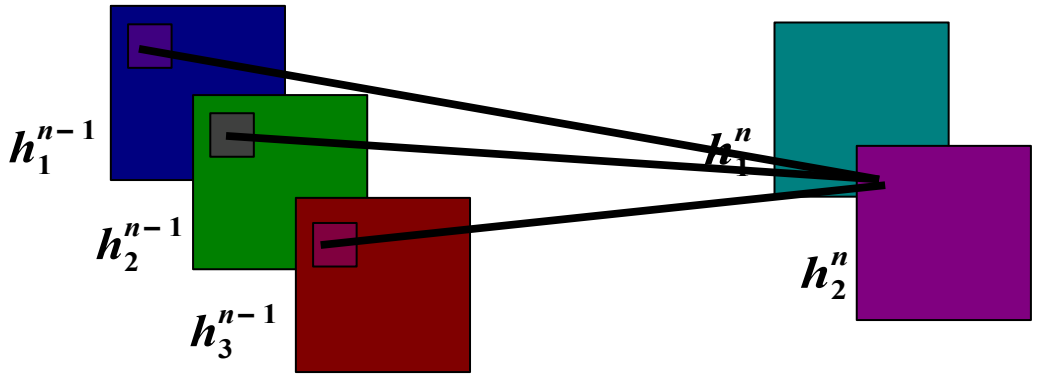
Convolutional layer with ReLU activation

$$h_i^n = \max\left\{0, \sum_{j=1}^{\#\text{input channels}} h_j^{n-1} * w_{ij}^n\right\}$$

output feature map

input feature map

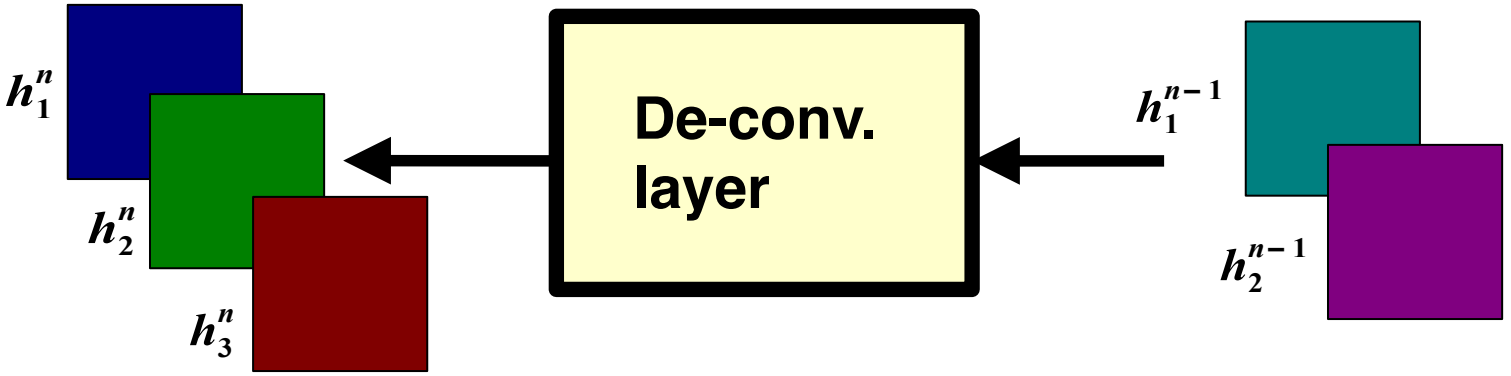
kernel



De-convolutional layer with ReLU activation

$$h_i^n = \max\left\{0, \sum_{j=1}^{\#\text{input channels}} h_j^{n-1} * w_{ij}^n\right\}$$

Still holds, same structure

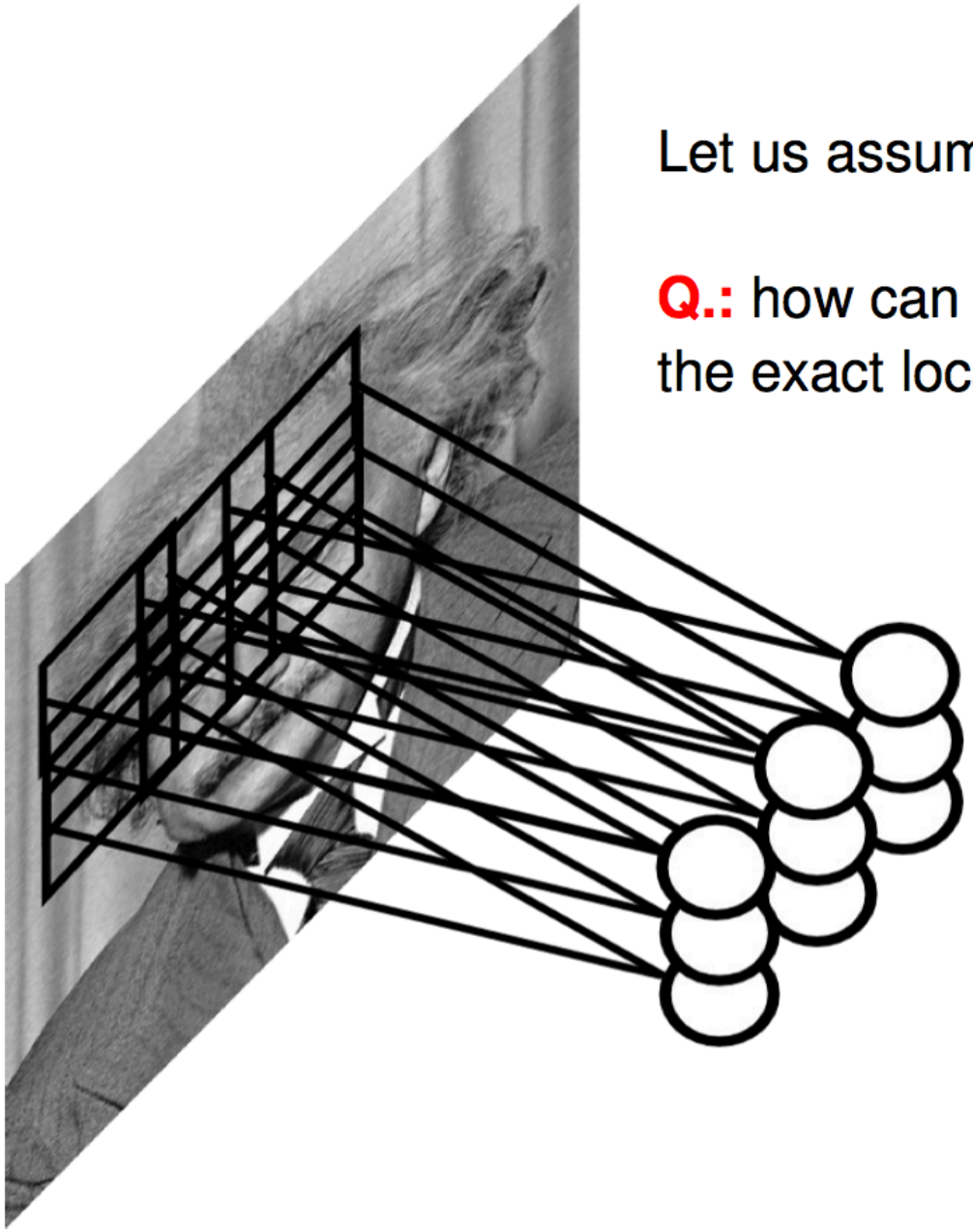


No real inverse - but convolutions can easily go the other way

"De-convolution" or "Transposed convolution"

Also a convolution with transposed weight tensor

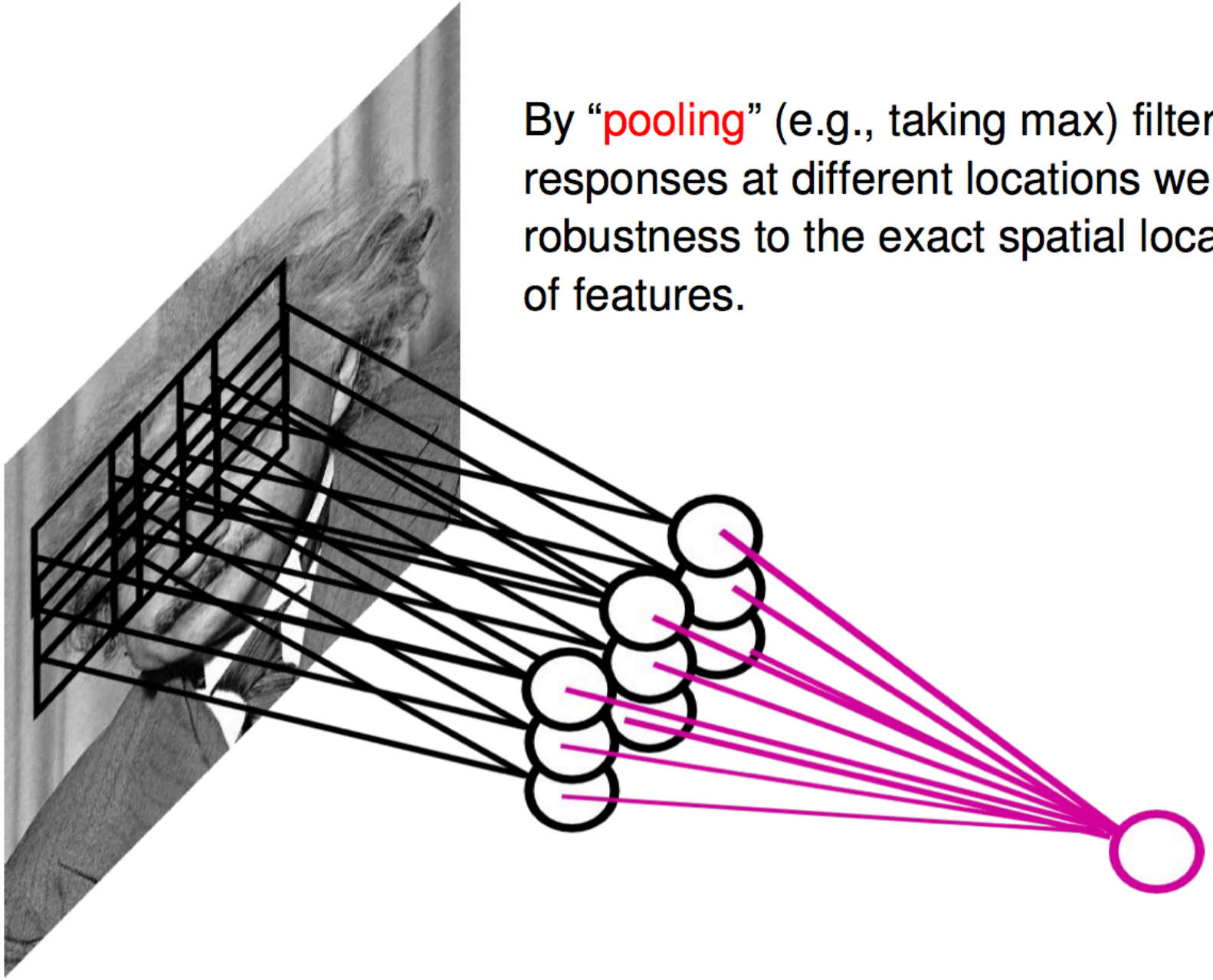
Pooling layer



Let us assume filter is an “eye” detector.

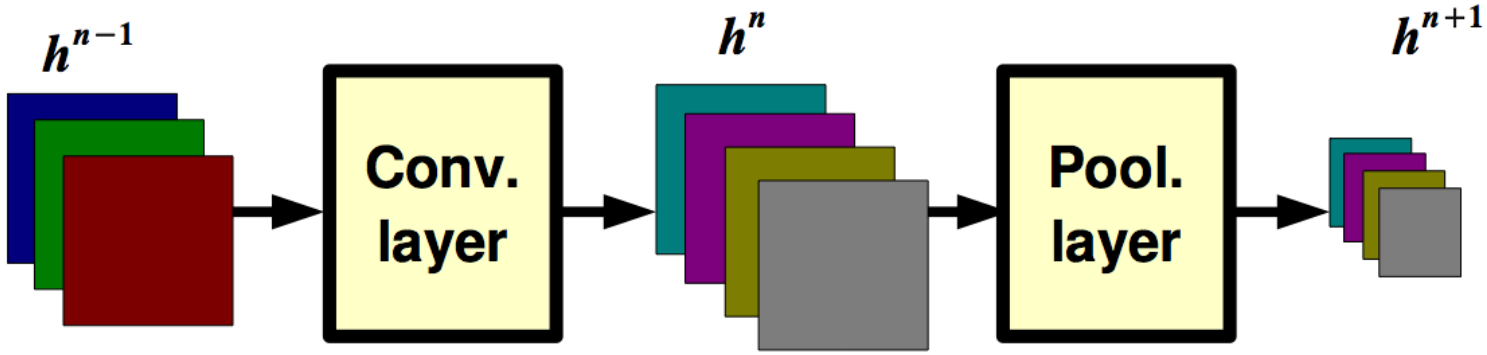
Q.: how can we make the detection robust to the exact location of the eye?

Pooling layer

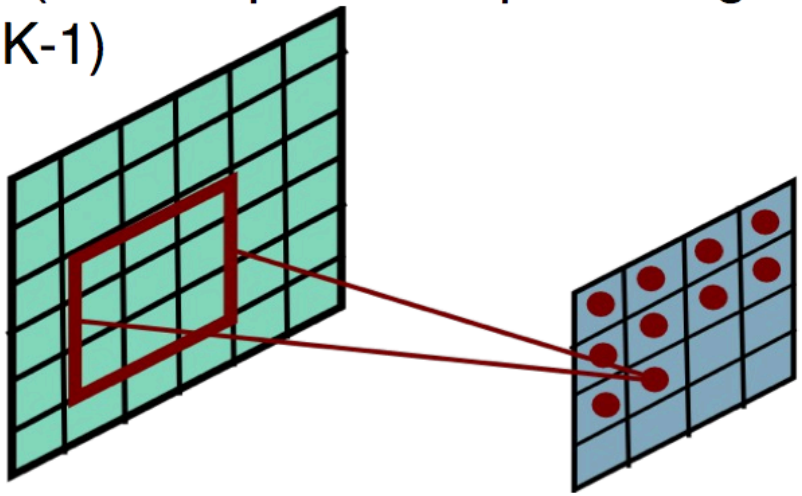


By “pooling” (e.g., taking max) filter responses at different locations we gain robustness to the exact spatial location of features.

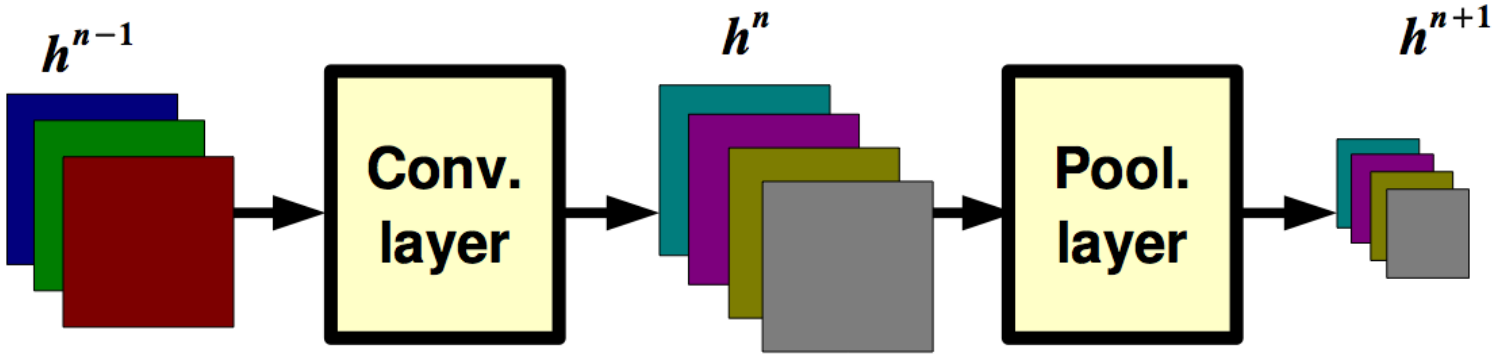
Pooling layer: receptive field size



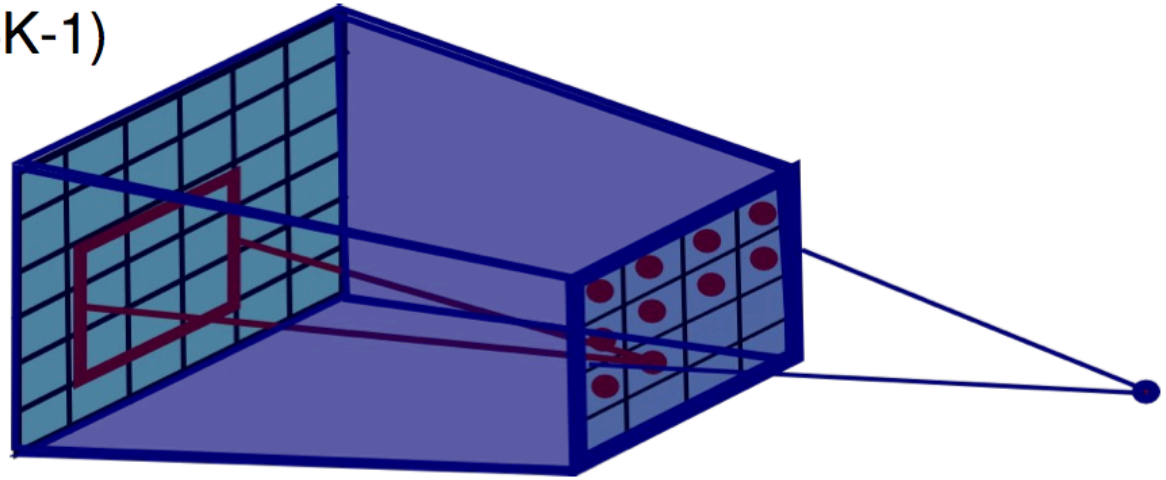
If convolutional filters have size KxK and stride 1, and pooling layer has pools of size PxP, then each unit in the pooling layer depends upon a patch (at the input of the preceding conv. layer) of size: (P+K-1)x(P+K-1)



Pooling layer: receptive field size



If convolutional filters have size KxK and stride 1, and pooling layer has pools of size PxP, then each unit in the pooling layer depends upon a patch (at the input of the preceding conv. layer) of size: (P+K-1)x(P+K-1)



Receptive field

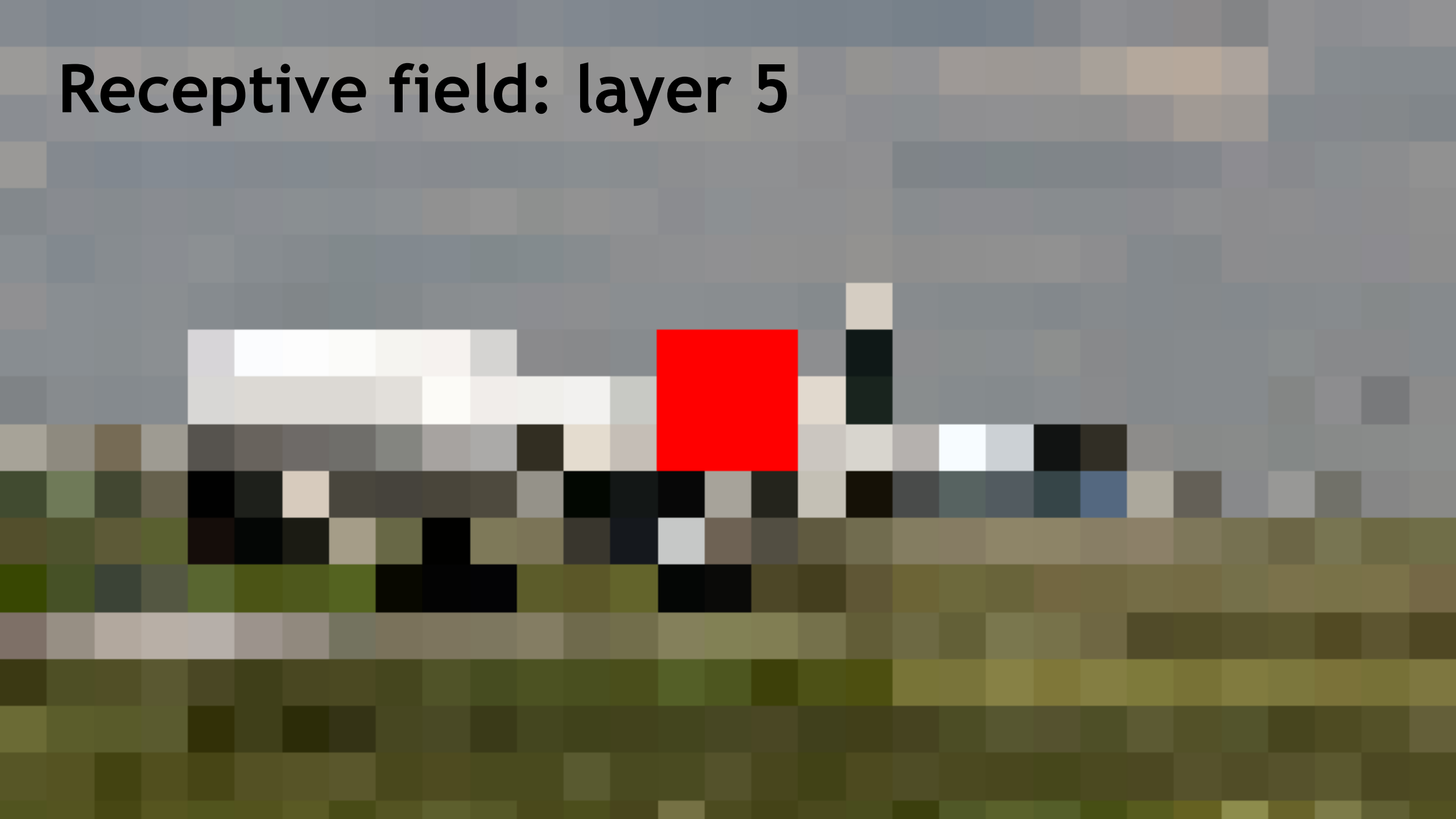
Receptive field: layer 1

Receptive field: layer 2

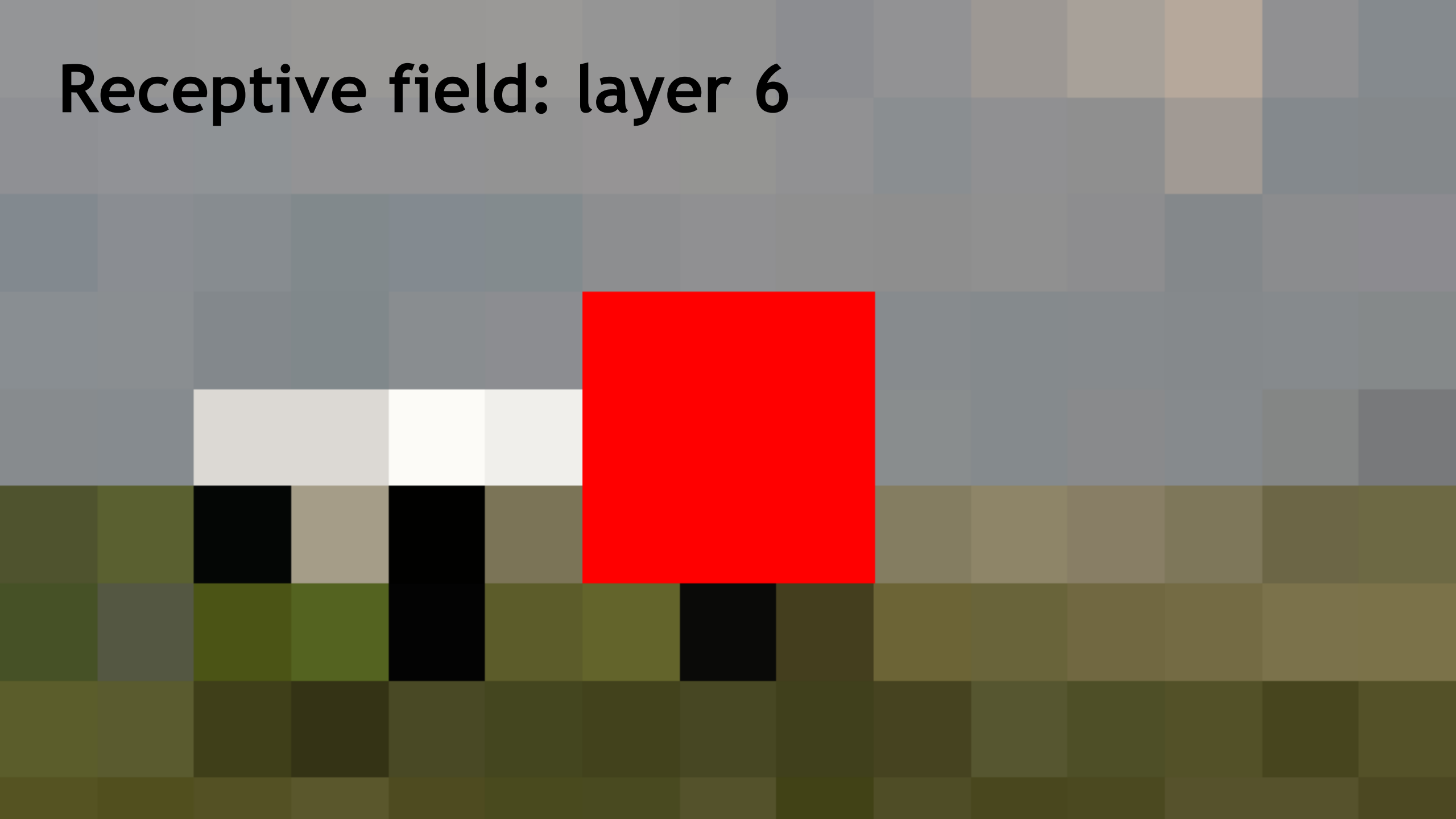
Receptive field: layer 3

Receptive field: layer 4

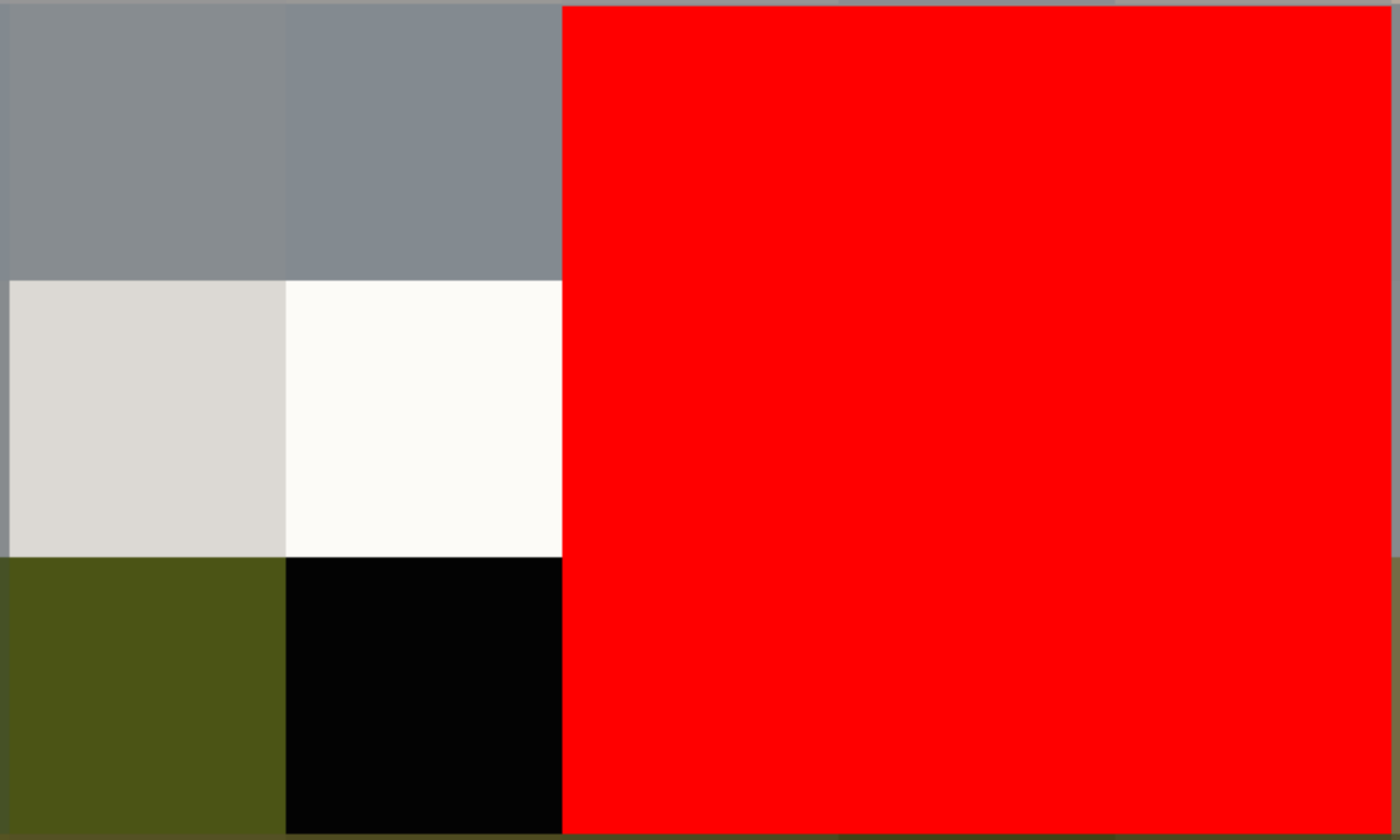
Receptive field: layer 5



Receptive field: layer 6



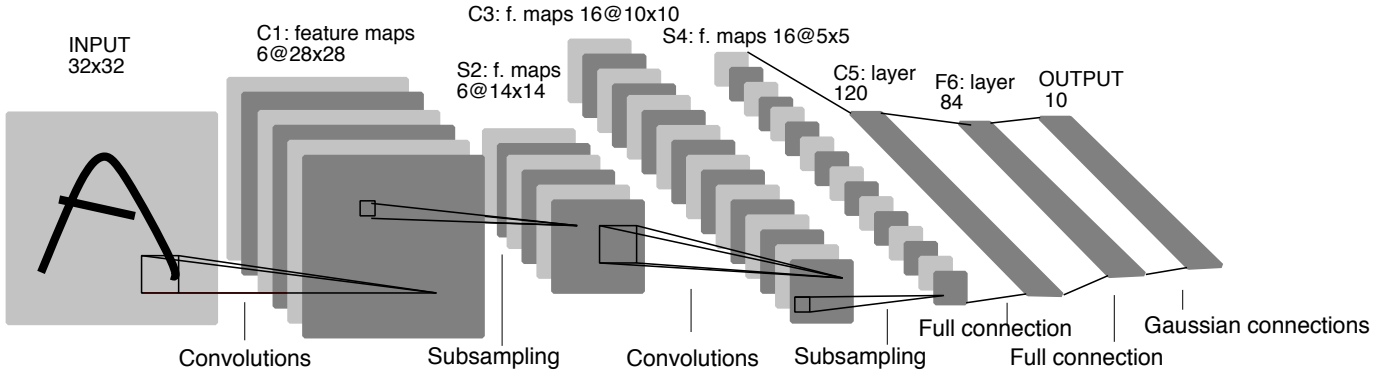
Receptive field: layer 7



Receptive field: layer 8

Modern Architectures

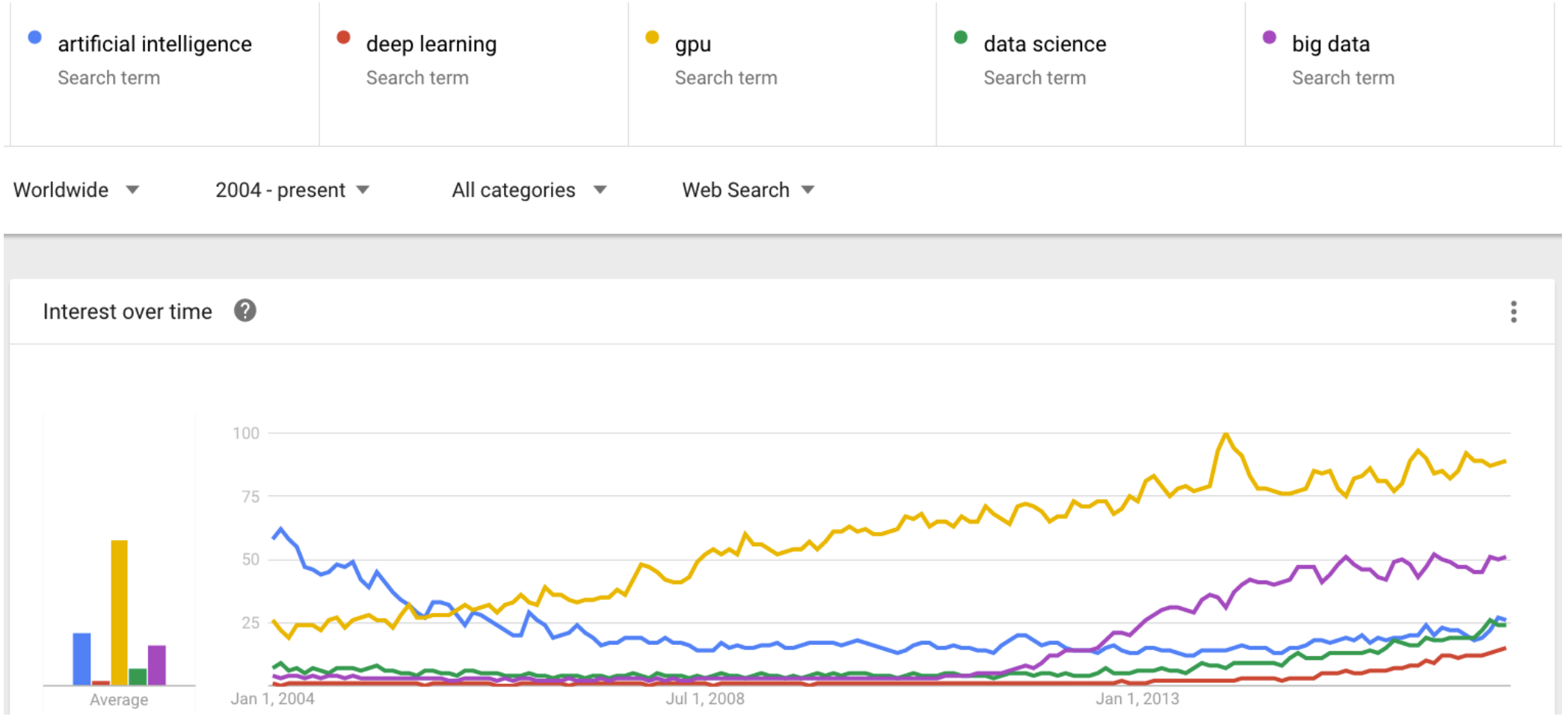
CNNs, late 1980's: LeNet



https://www.youtube.com/watch?v=FwFduRA_L6Q

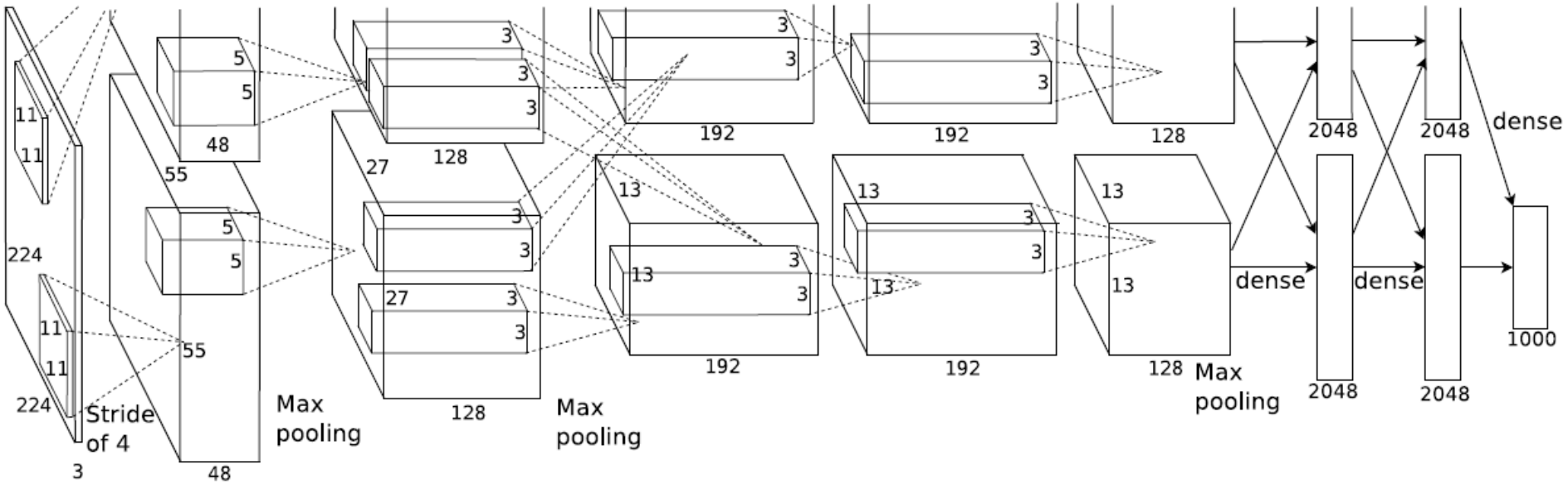
Gradient-based learning applied to document recognition,
Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, 1998.

What happened in between?



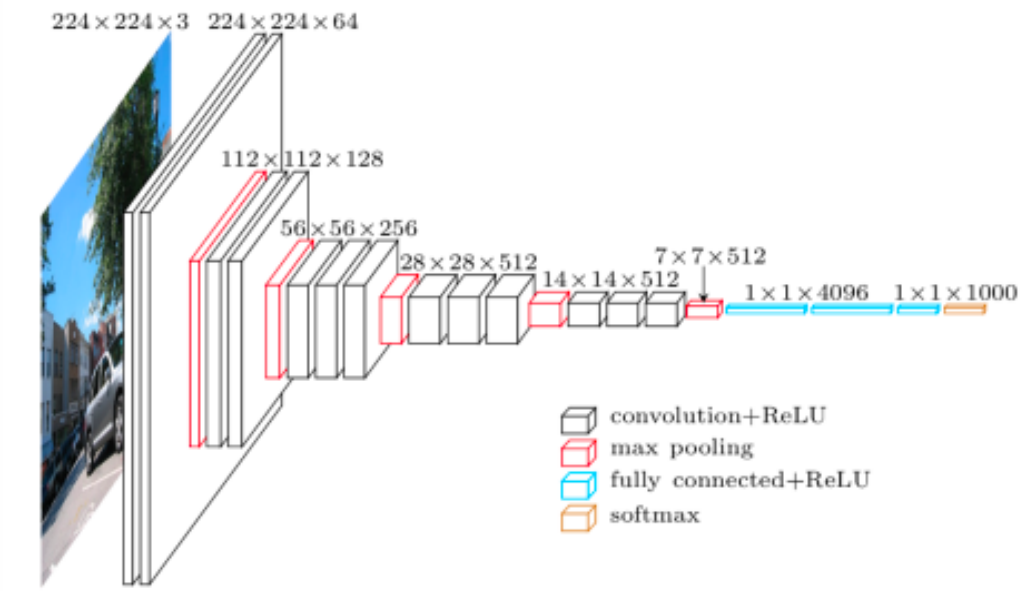
deep learning = neural networks (+ big data + GPUs) **+ a few more recent tricks!**

CNNs, 2012



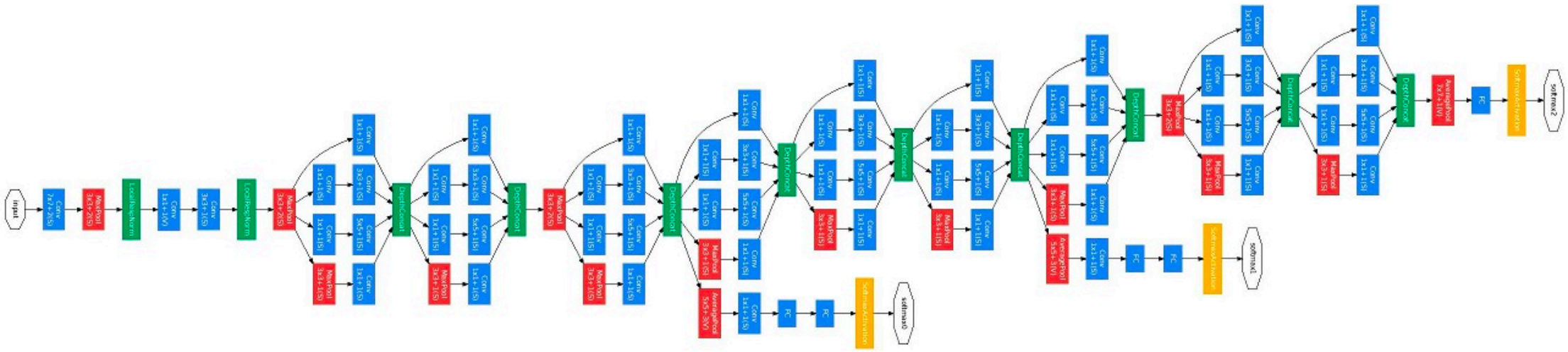
AlexNet
Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton: ImageNet classification with deep convolutional neural networks. Commun. ACM 60(6): 84-90 (2017)

CNNs, 2014: VGG



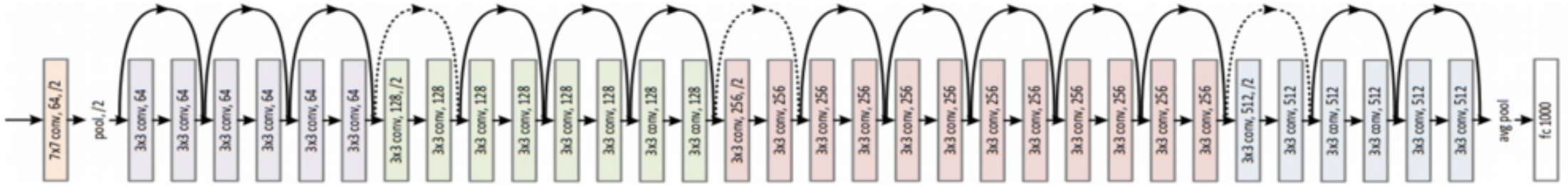
VGG
Karen Simonyan, Andrew Zisserman (=Visual Geometry Group)
Very Deep Convolutional Networks for Large-Scale Image Recognition, arxiv, 2014.

CNNs, 2014: GoogLeNet



Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich
Going Deeper with Convolutions, CVPR 2015

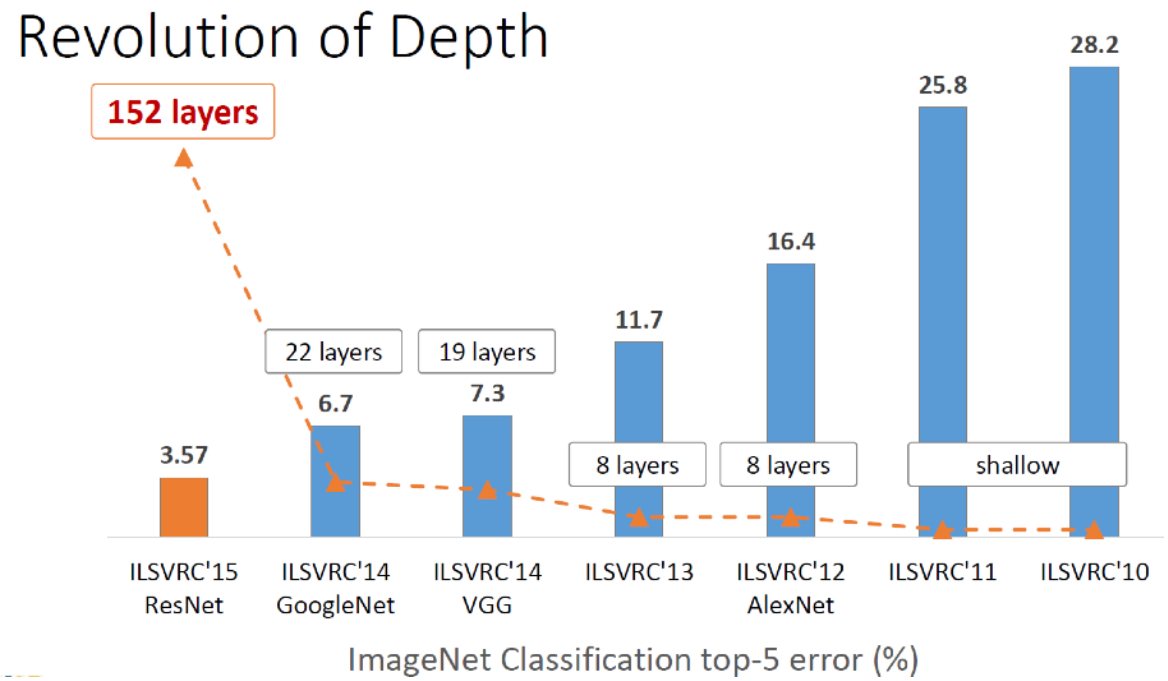
CNNs, 2015: ResNet



ResNet
Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun,
Deep Residual Learning for Image Recognition, CVPR 2016.

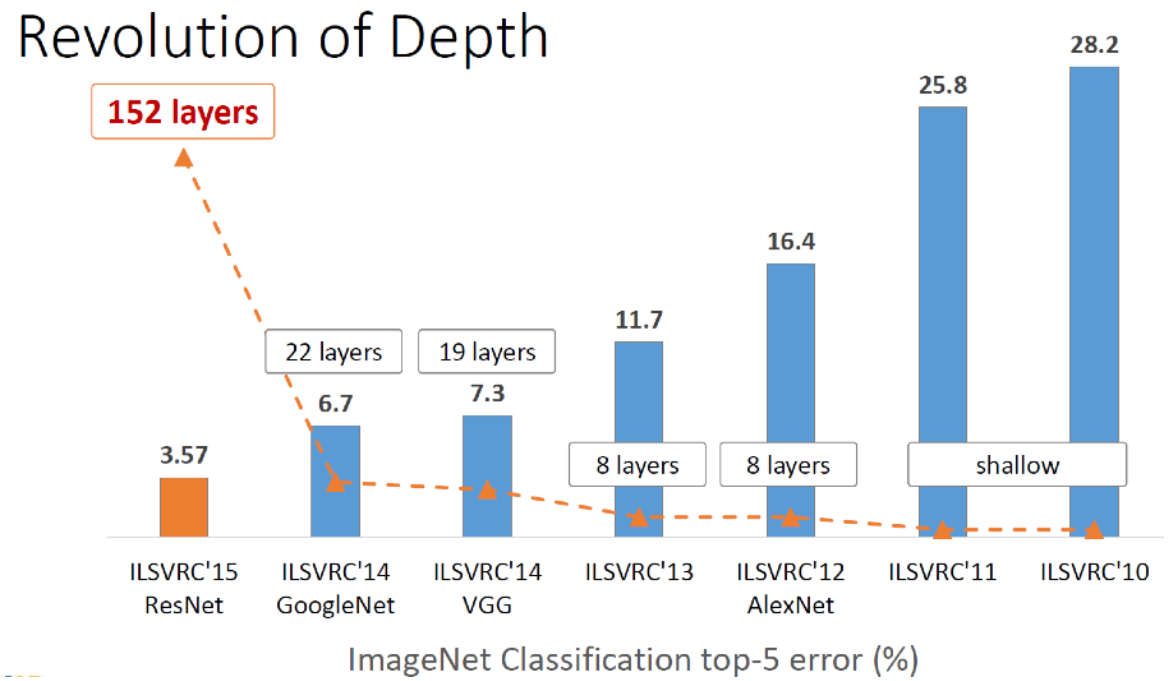
The Deeper, the Better

- Deeper networks can cover more complex problems
  - Increasingly large receptive field size & rich patterns



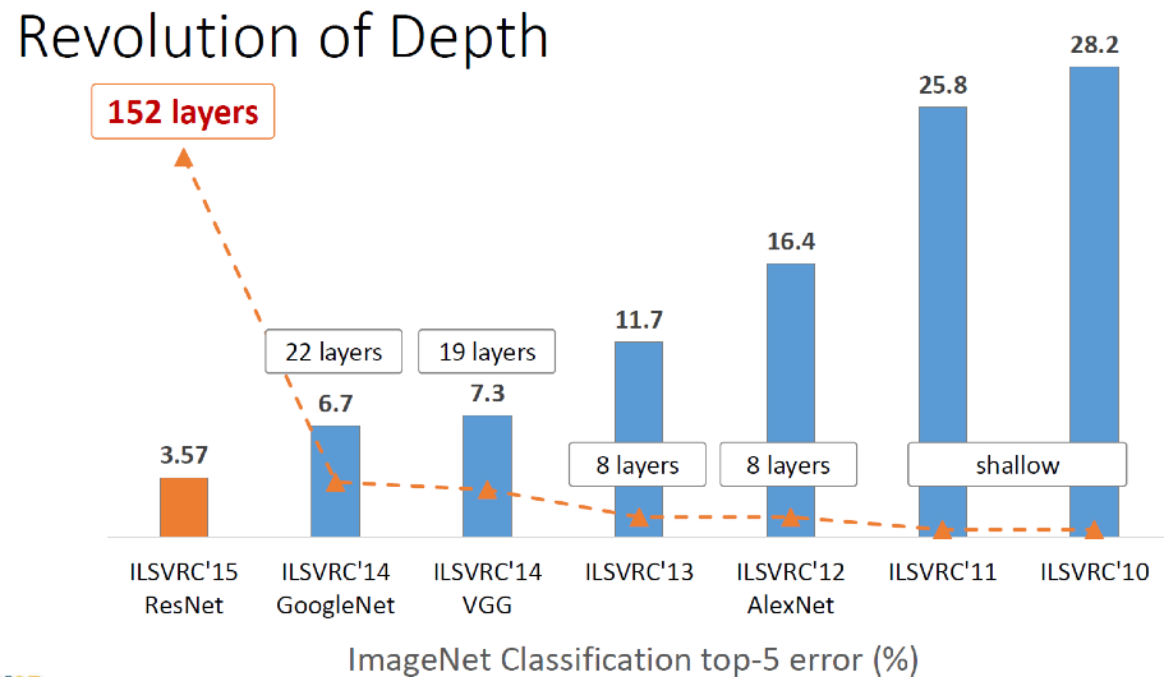
Going Deeper

- From 2 to 10: 2010-2012
  - ReLUs
  - Dropout
  - …



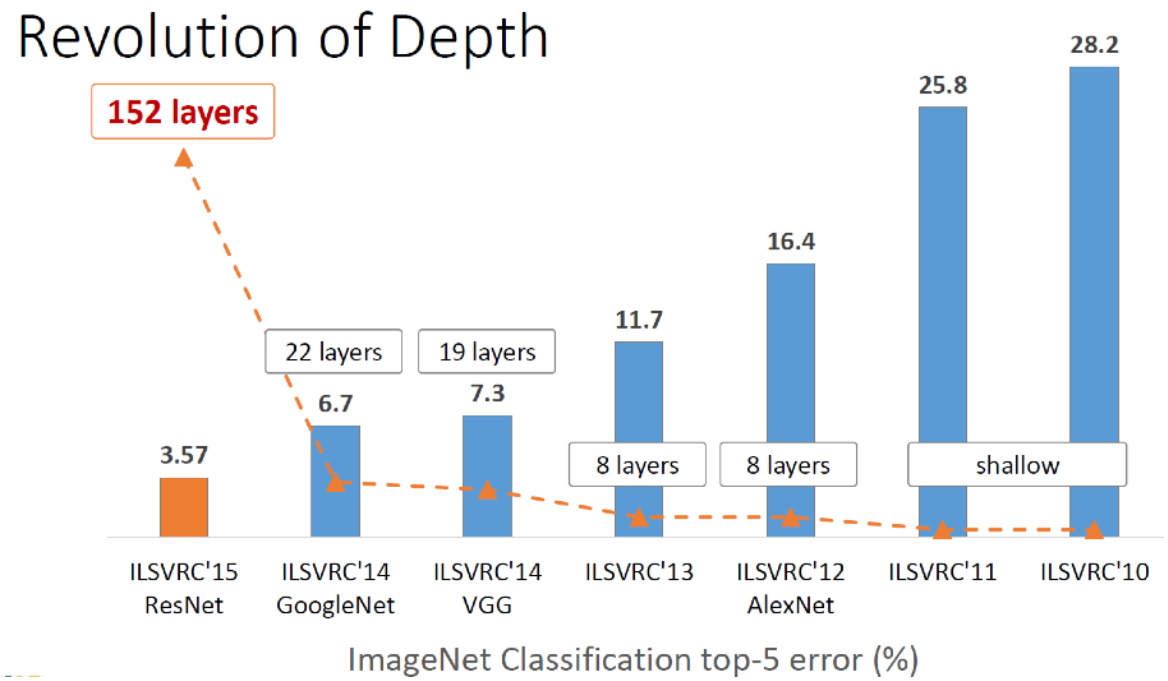
Going Deeper

- From 10 to 20: 2015
  - Batch Normalization



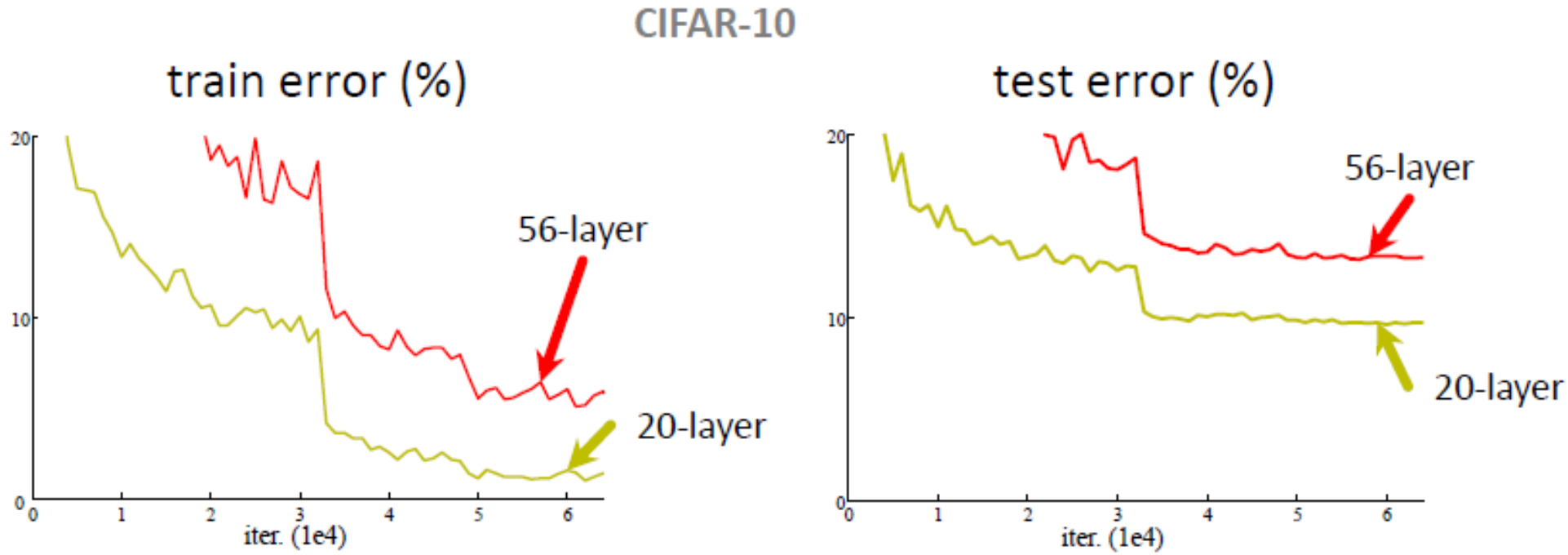
Going Deeper

- From 20 to 100/1000
  - Residual networks



Plain Network: Deeper is not necessarily better

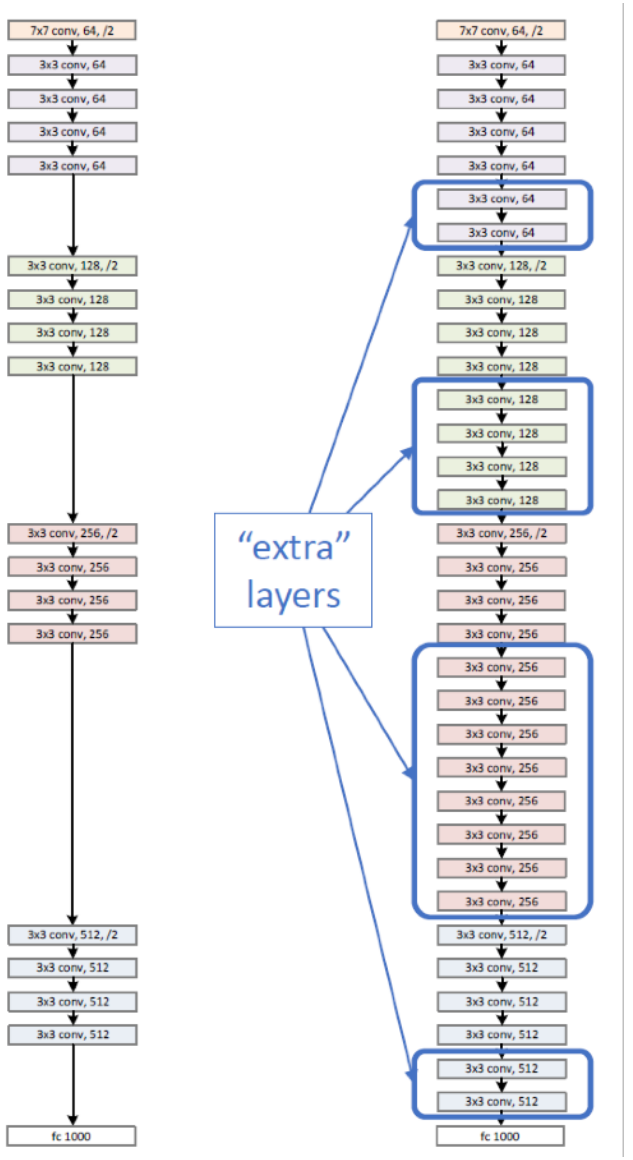
- Plain nets: stacking 3x3 conv layers
- 56-layer net has higher training error and test error than 20-layer net



Residual Network

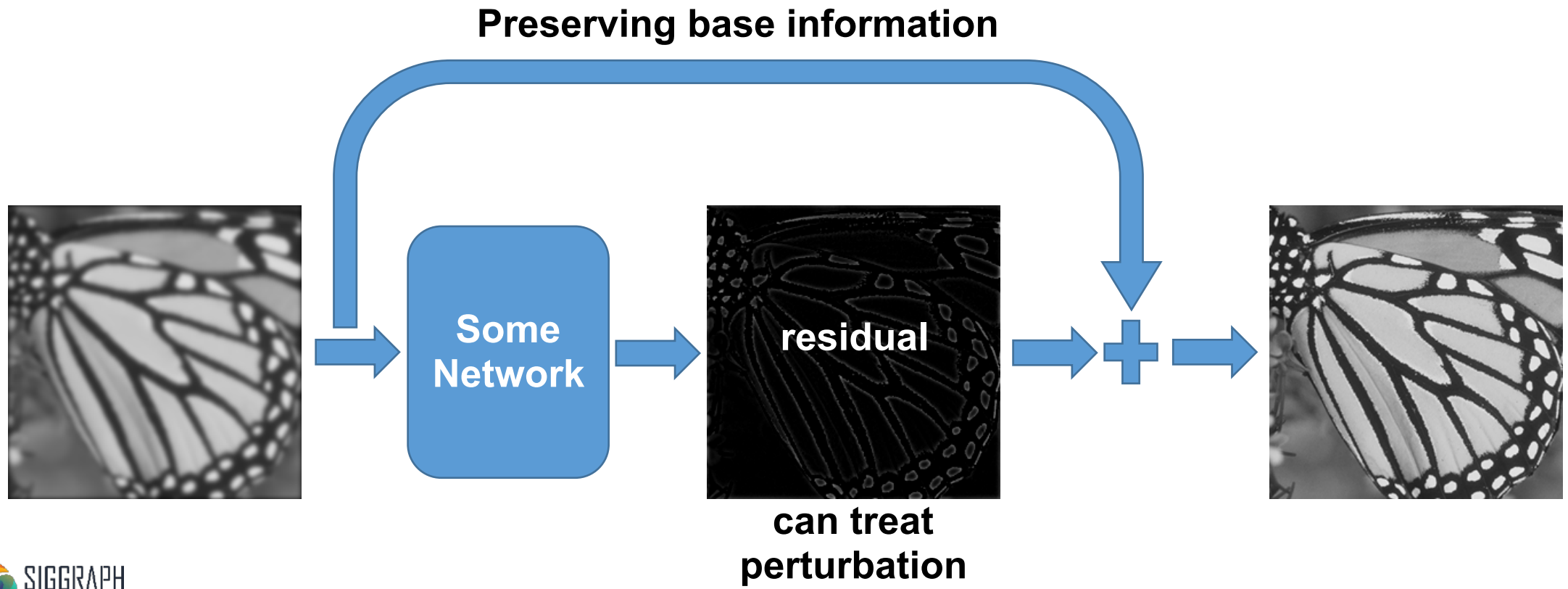
Naïve solution

- If extra layers are an identity mapping, then training errors can not increase



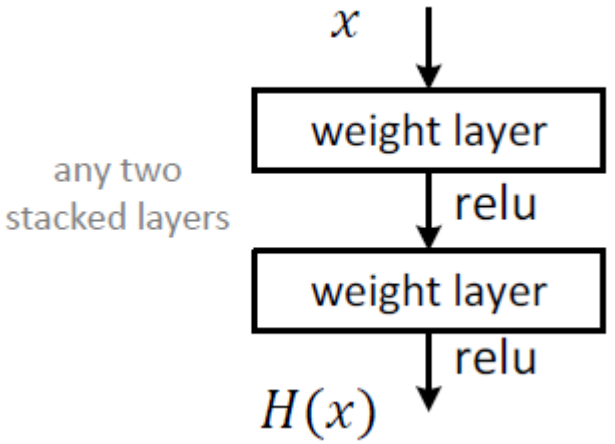
Residual Modelling: Basic idea in image processing

- Goal: estimate update between an original image and a changed image



Residual Network

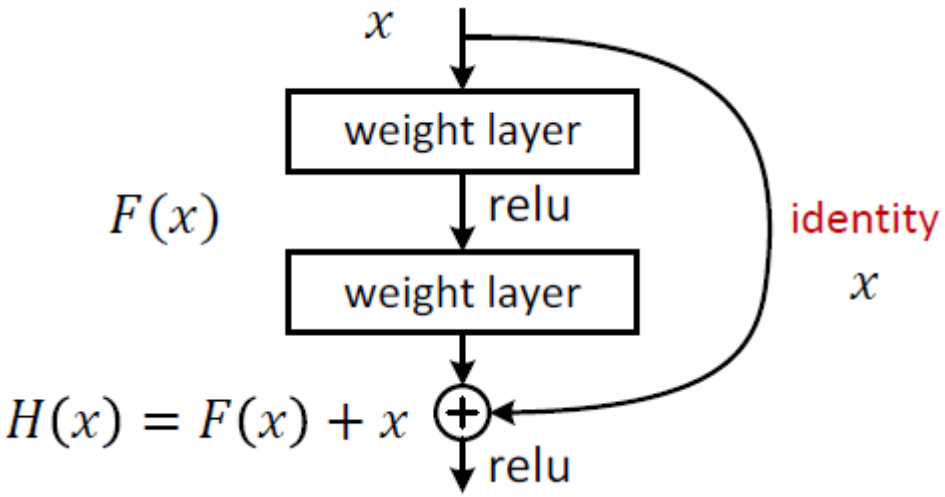
- Plain block
  - Difficult to make identity mapping because of multiple non-linear layers



Residual Network

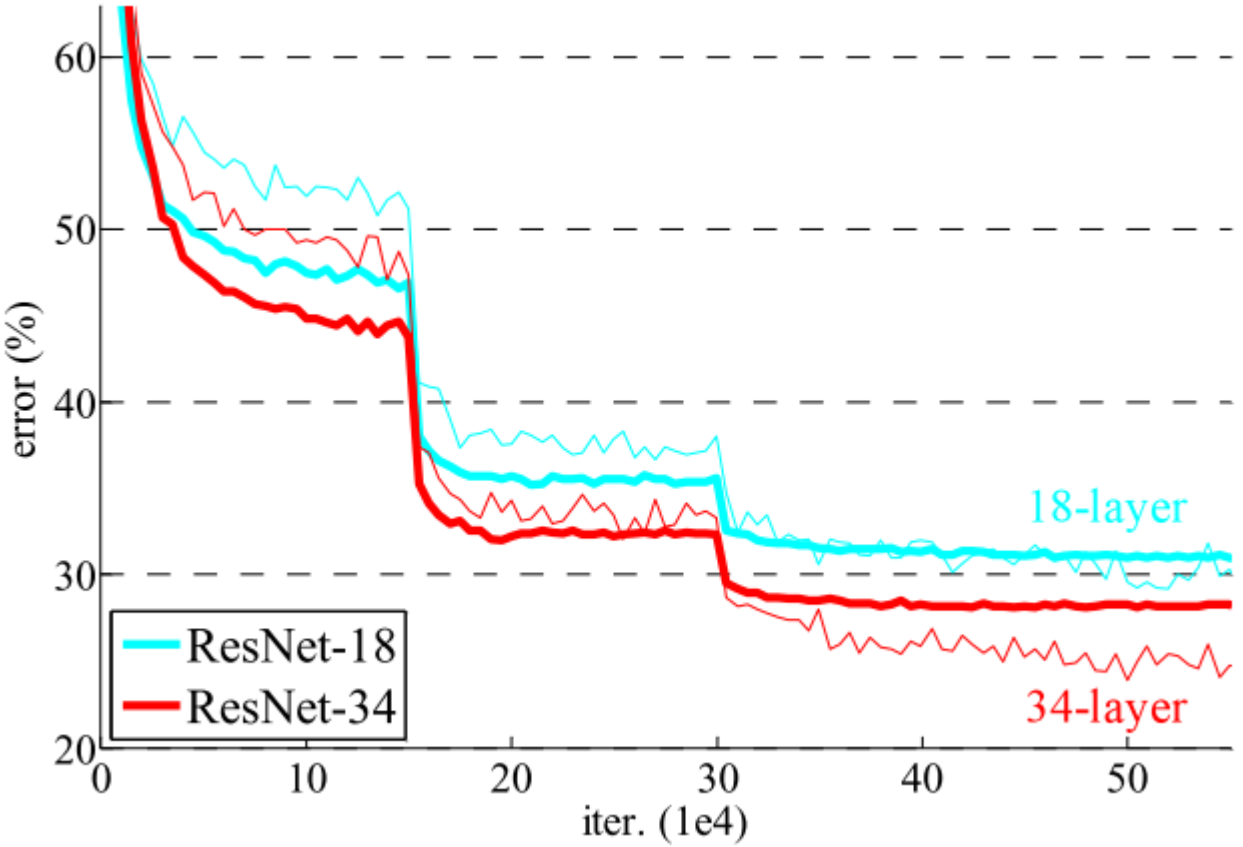
- Residual block
  - If identity were optimal, easy to set weights as 0
  - If optimal mapping is closer to identity, easier to find small fluctuations

Appropriate for treating perturbation a
keeping a base information

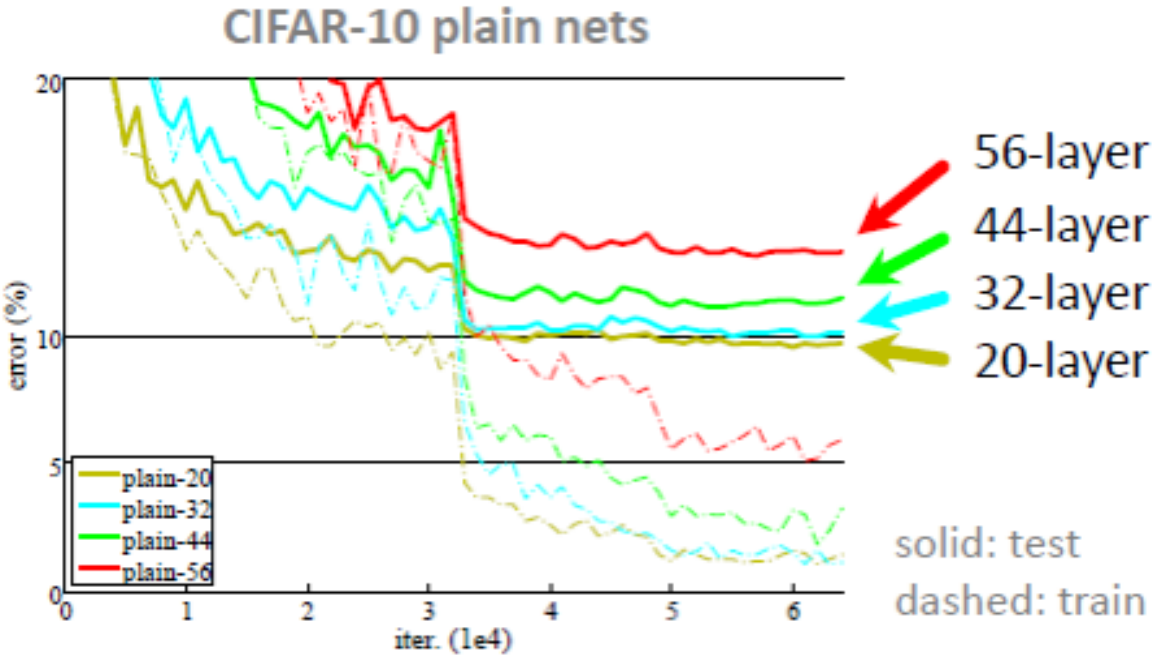


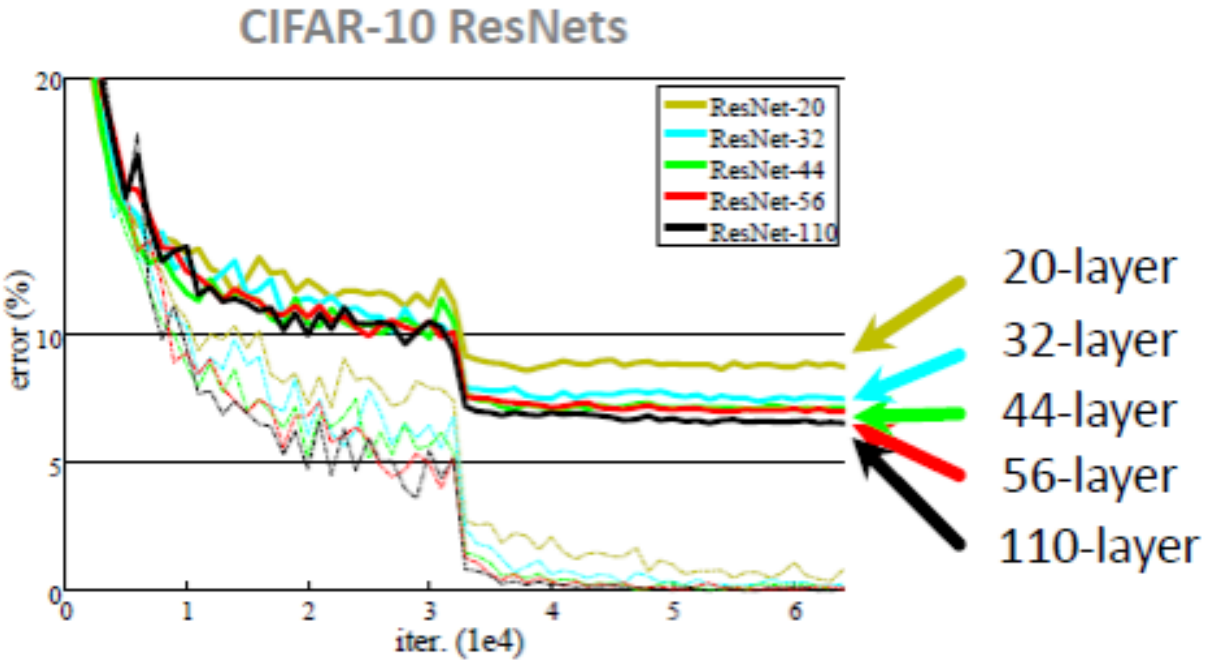
Residual Network: Deeper is better

- Deeper ResNets have lower training error



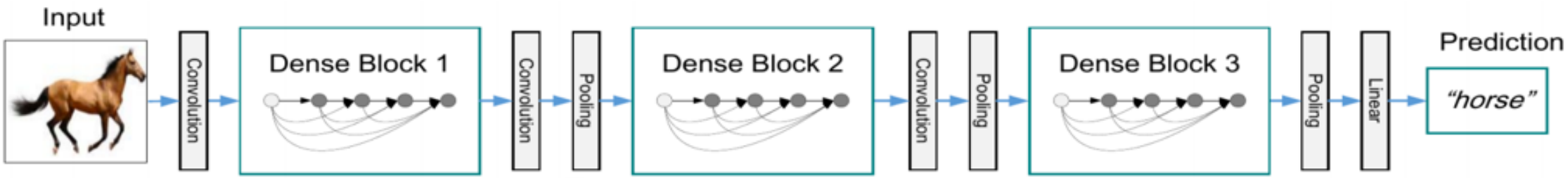
Residual Network: Deeper is better



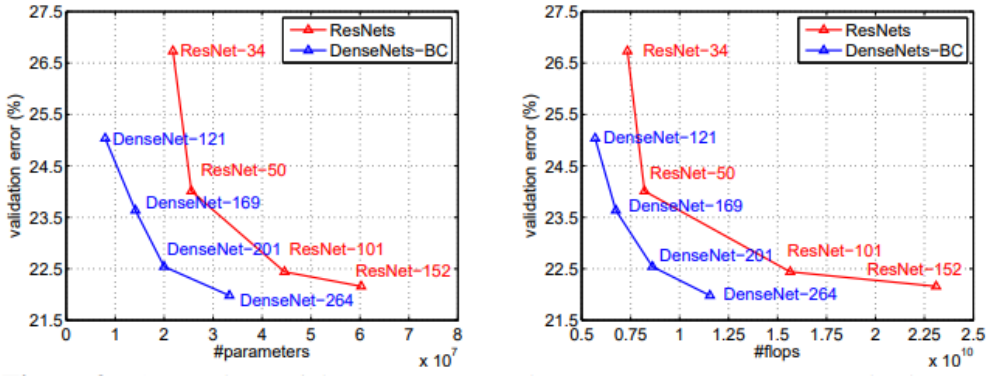


CNNs, 2017: DenseNet

Densely Connected Convolutional Networks, CVPR 2017
Gao Huang, Zhuang Liu, Laurens van der Maaten, Kilian Q. Weinberger



Recently proposed, better performance/parameter ratio



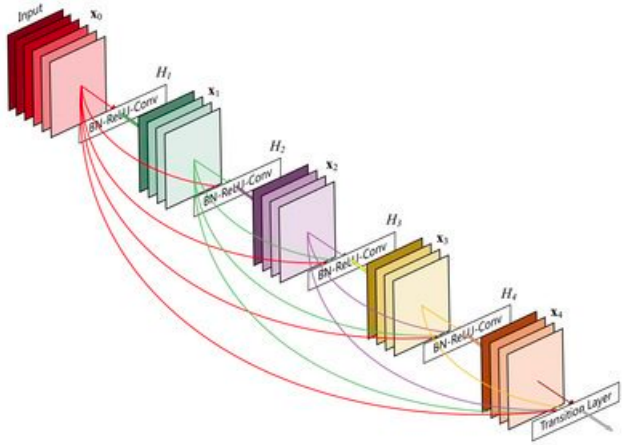
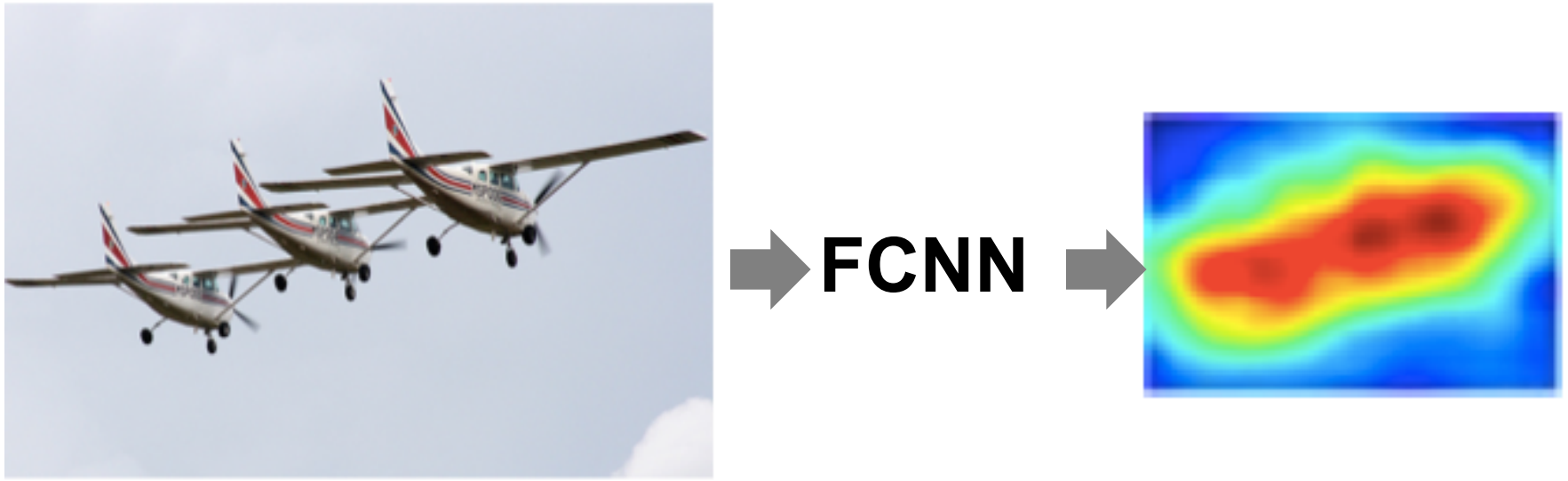


Image-to-Image

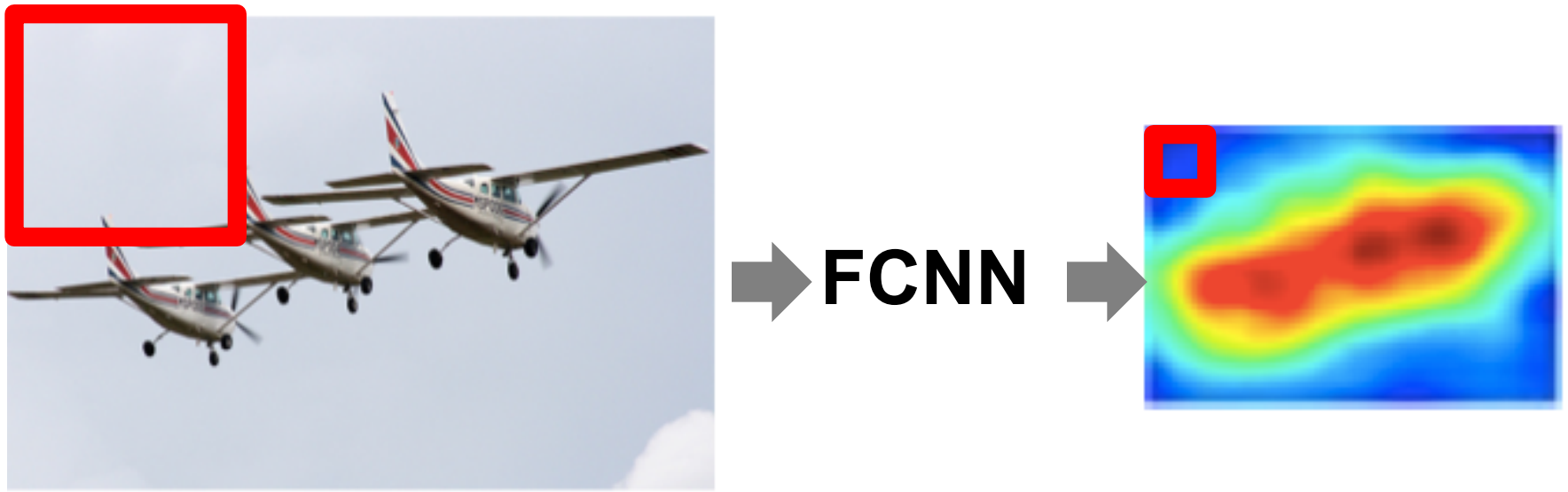
Image-to-image

- So far we mapped an image image to a number or label
- In graphics, output often is “richer”:
  - An image
  - A volume
  - A 3D mesh
  - …
- Note: “*image*” just placeholder name here for any Eulerian data
- **Architectures**
  - Encoder-Decoder
  - Skip connections

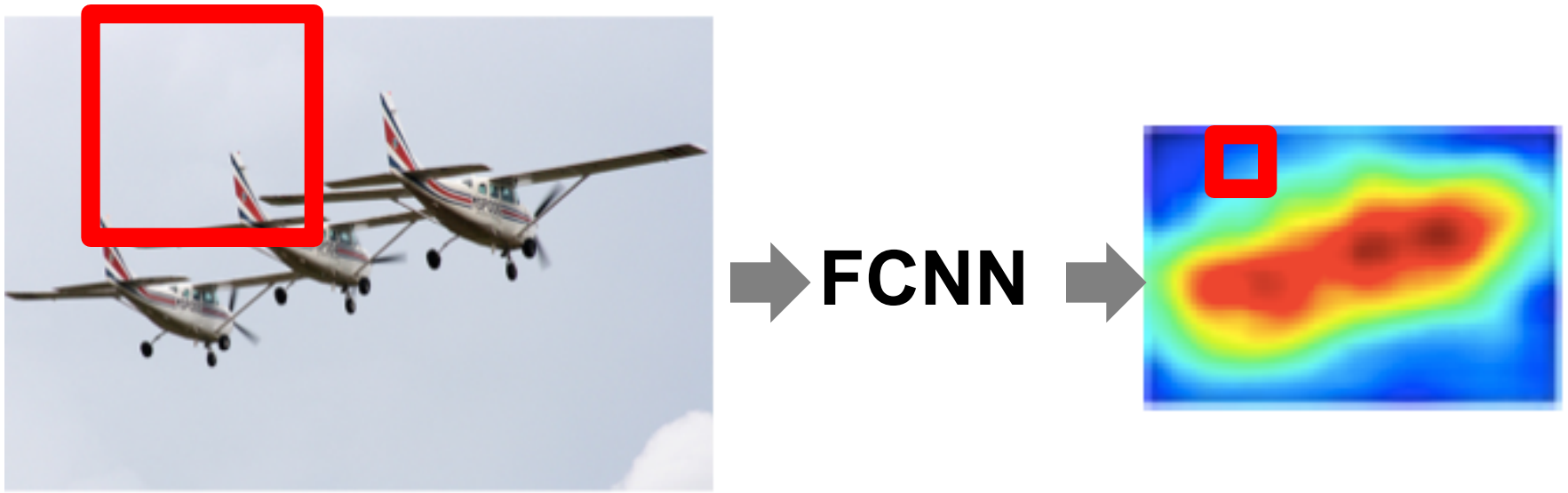
Fully-convolutional Neural Networks



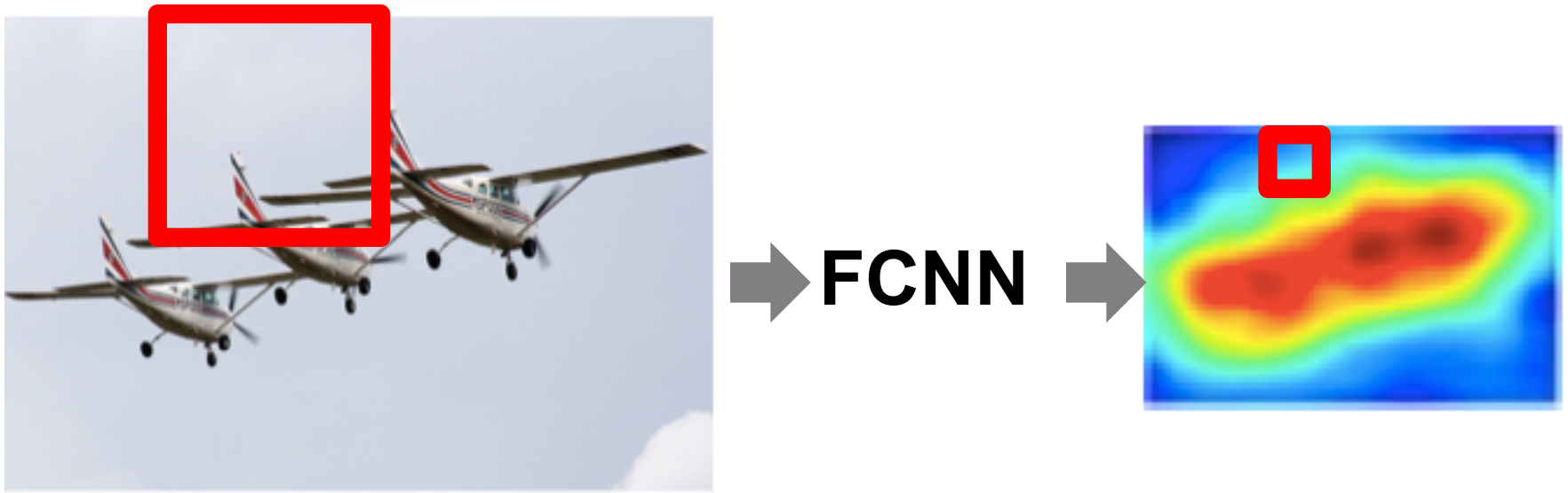
Fully-convolutional Neural Networks



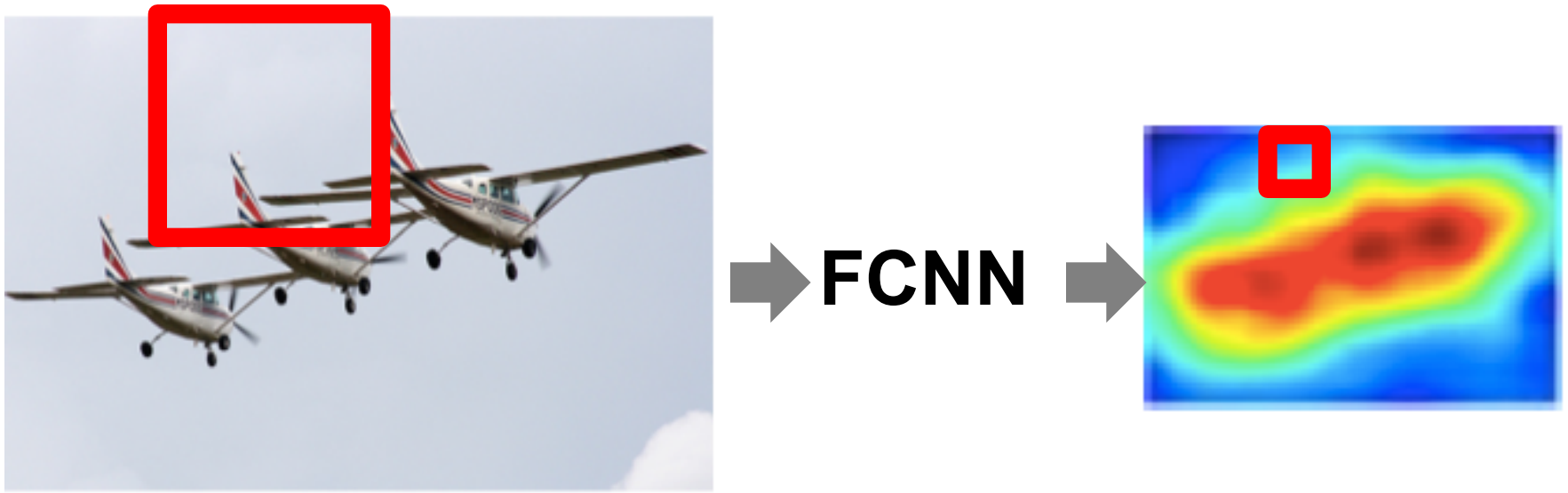
Fully-convolutional Neural Networks



Fully-convolutional Neural Networks



Fully-convolutional Neural Networks

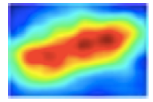


Flexible - works with varying input sizes

Fully Convolutional Neural Networks in Practice

32-fold decimation
224x224 to 7x7

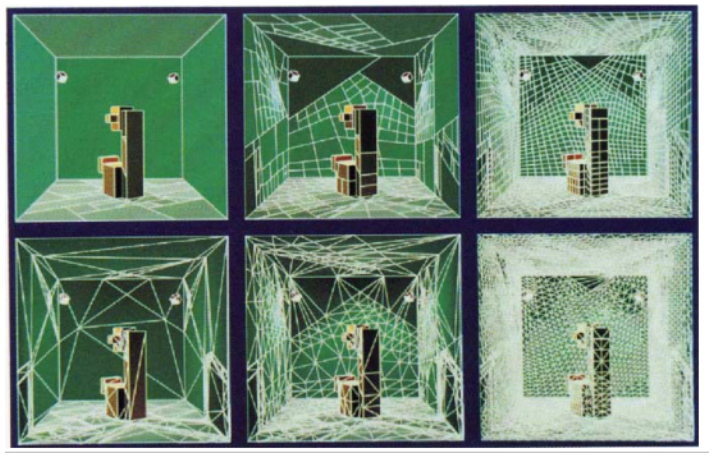
FCNN

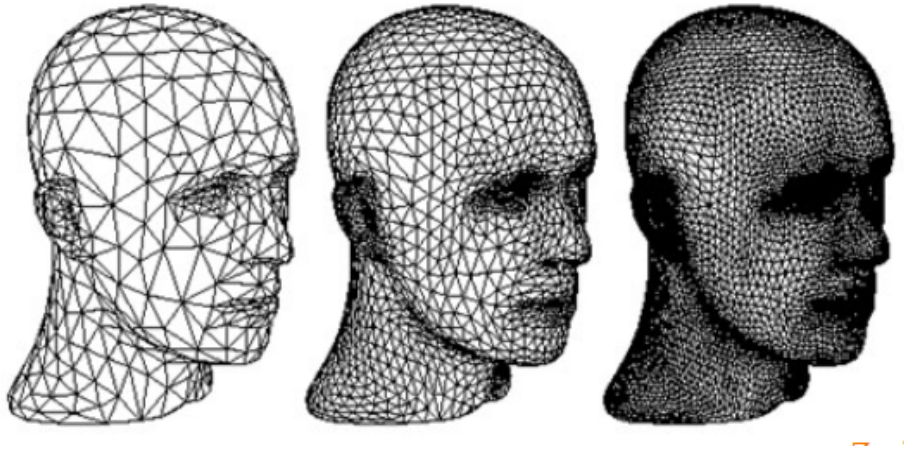


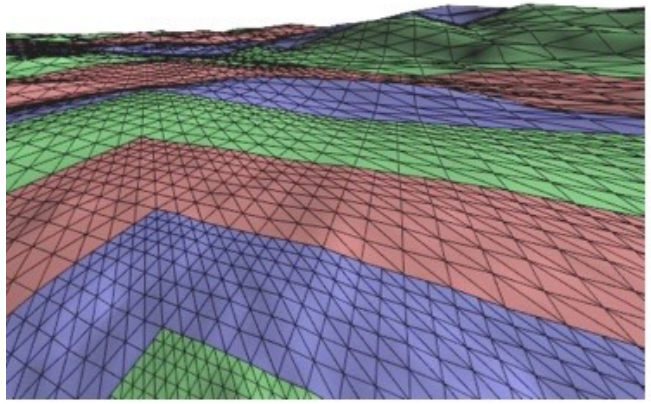
Flexible - works with varying input sizes

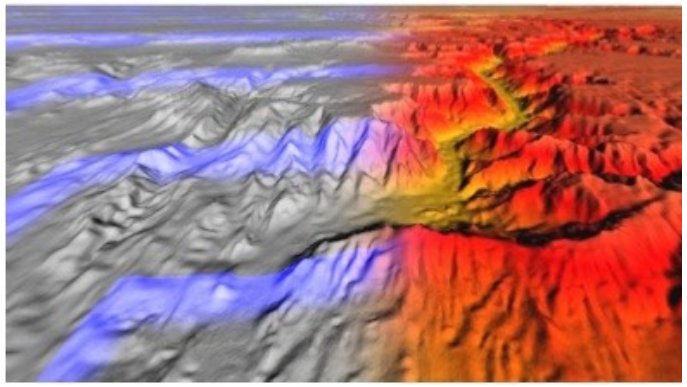
Typically reduces input by fixed factor

Graphics: Multiresolution

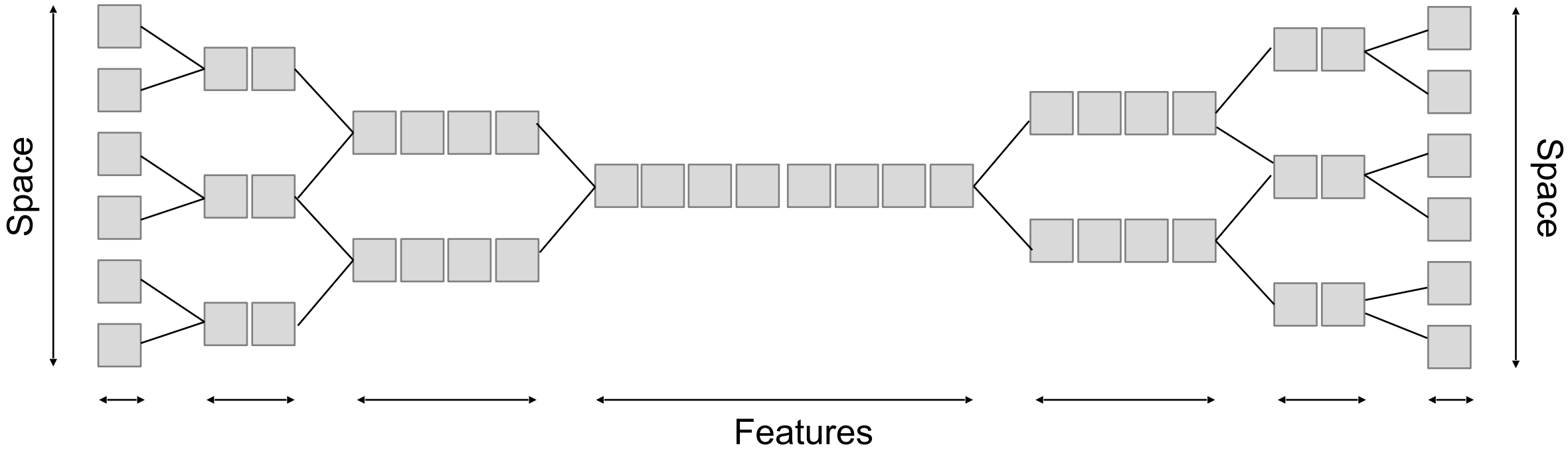








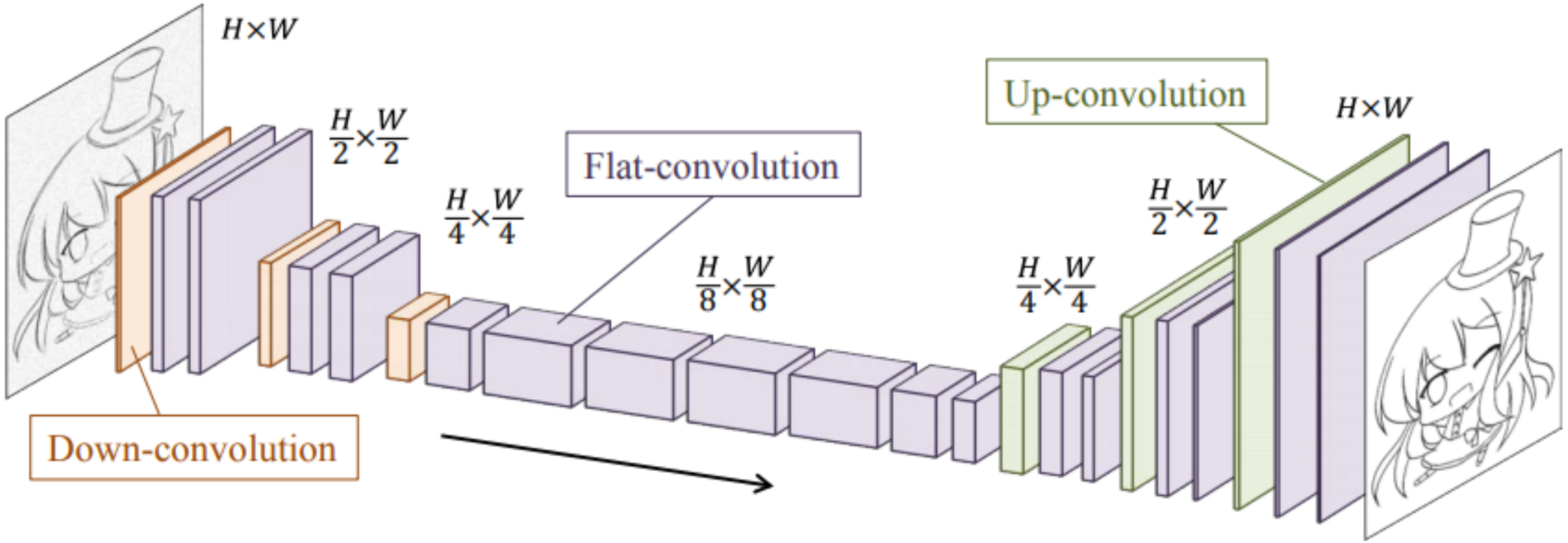
Encoder-Decoder



Interpretation

- Encoder: turns data set (e.g. image) into vector
- This vector is a very compact and abstract “code”
- Lives in the “latent space” of the neural network
- Decoder: turns code back into image

Encoder-Decoder



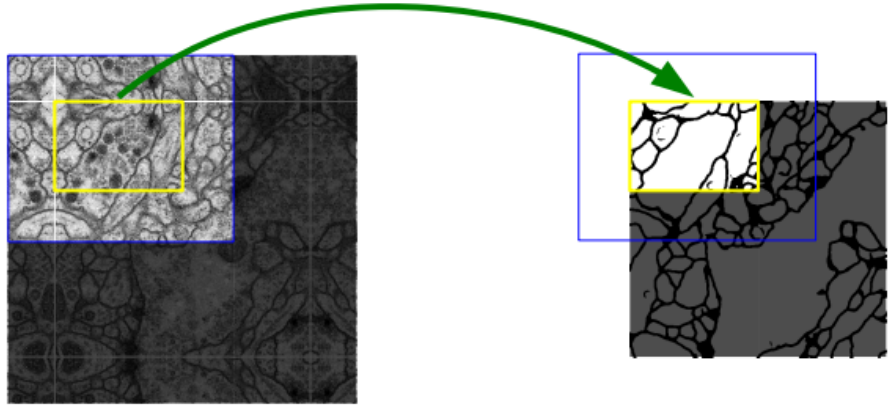
Learning to simplify. Simo-Serra et al. 2016

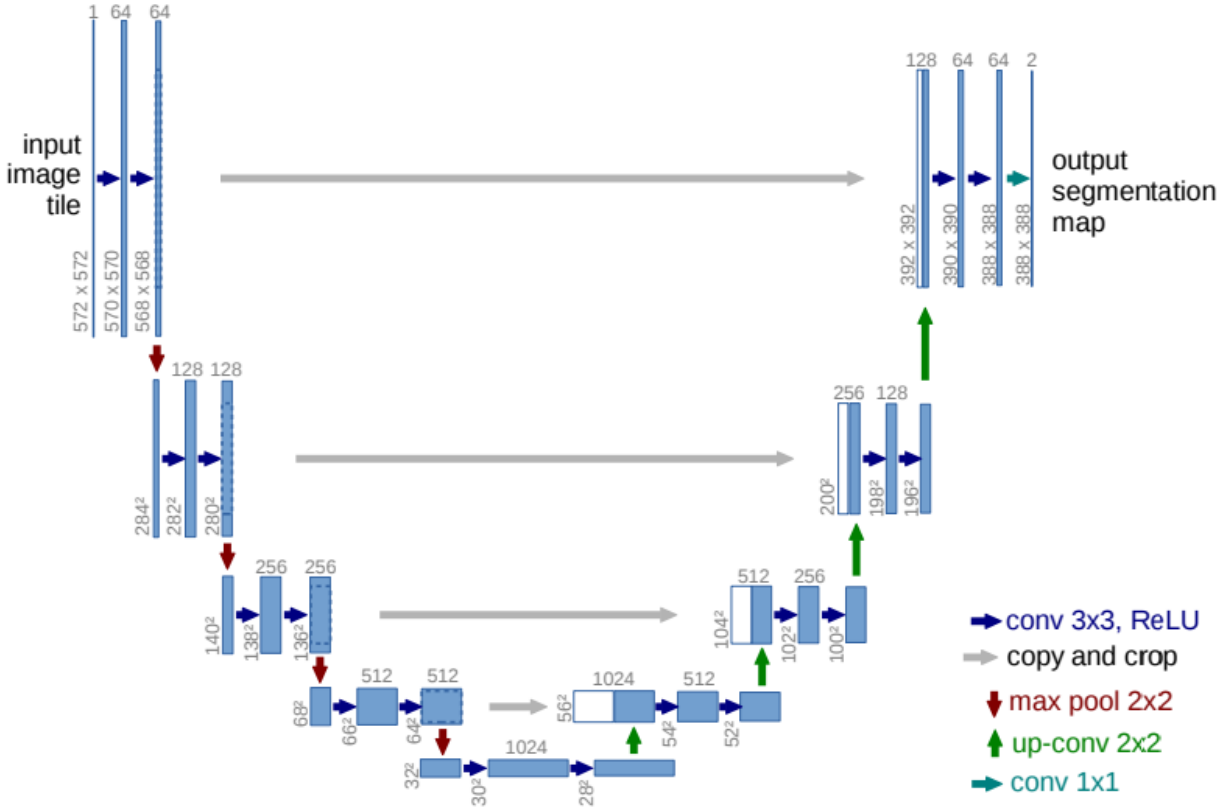
Up-sampling

- We saw
  - … how to keep resolution
  - … how to reduce it with pooling
- But how to increase it again?
- Options
  - Interpolation
  - Padding (insert zeros)
  - Transpose convolutions

Encoder-decoder + Skip connections

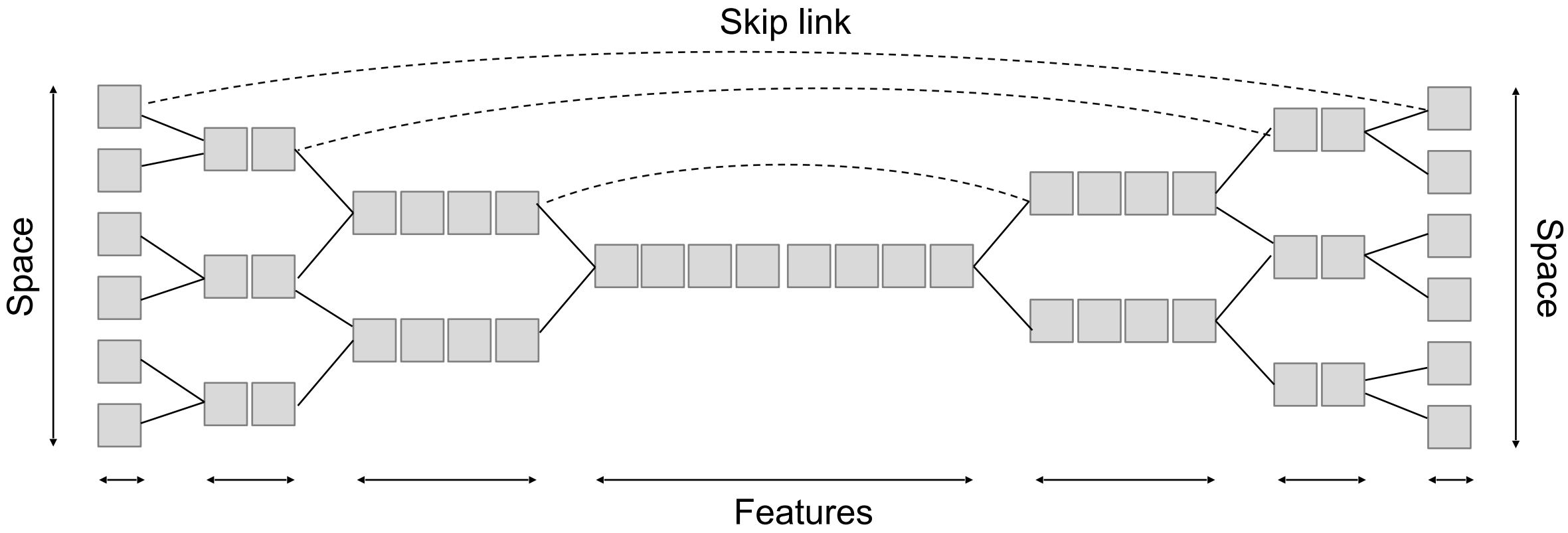
- 1st: Reduce resolutions as before
- 2nd: Increase resolution
- Transposed convolutions
- Preserves information
- But cannot be split into en- and decoder anymore





U-Net: Convolutional Networks for Biomedical Image Segmentatio. Ronneberger et al. 2015

Encoder-decoder with skip connections



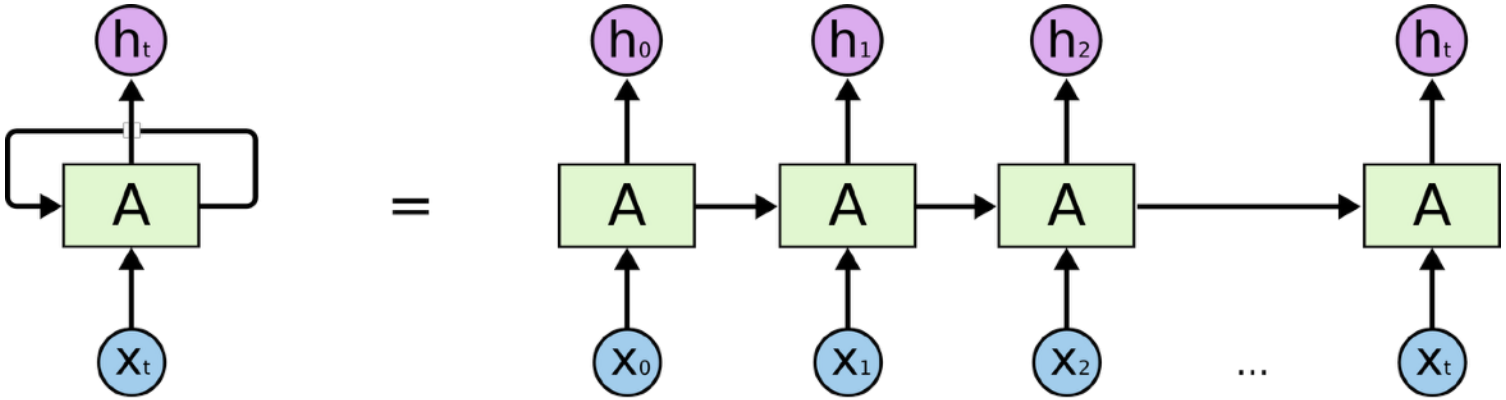
Interpretation

- Turns image into vector
- Turns vector back into image
- At every step of increasing the resolution, check back with the input to preserve details
- Familiar trick to graphics people
  - (Haar) wavelet
  - Residual coding
  - Pyramidal schemes (Laplacian pyramid, etc.)

Recurrent Neural Networks

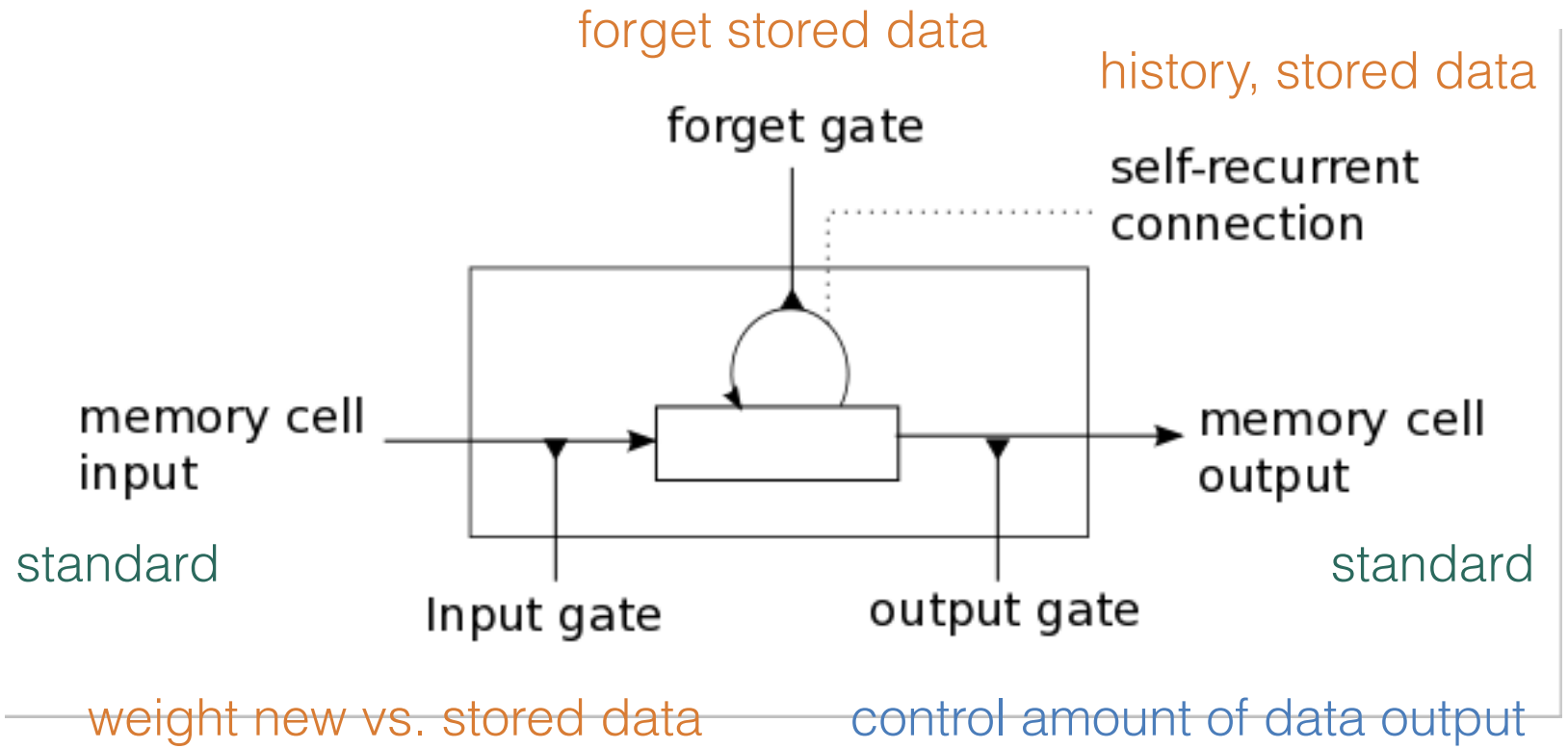
Recurrent Neural Networks

- Time dependent problems: repeated evaluations with internal “state”
- State x_t at time t, depends on previous times
- Recurrent Neural Networks (RNNs)
- Specialized back-prop possible: Back-propagation through time (BPTT)
- Unrolled:



Common Building Block: LSTM Units

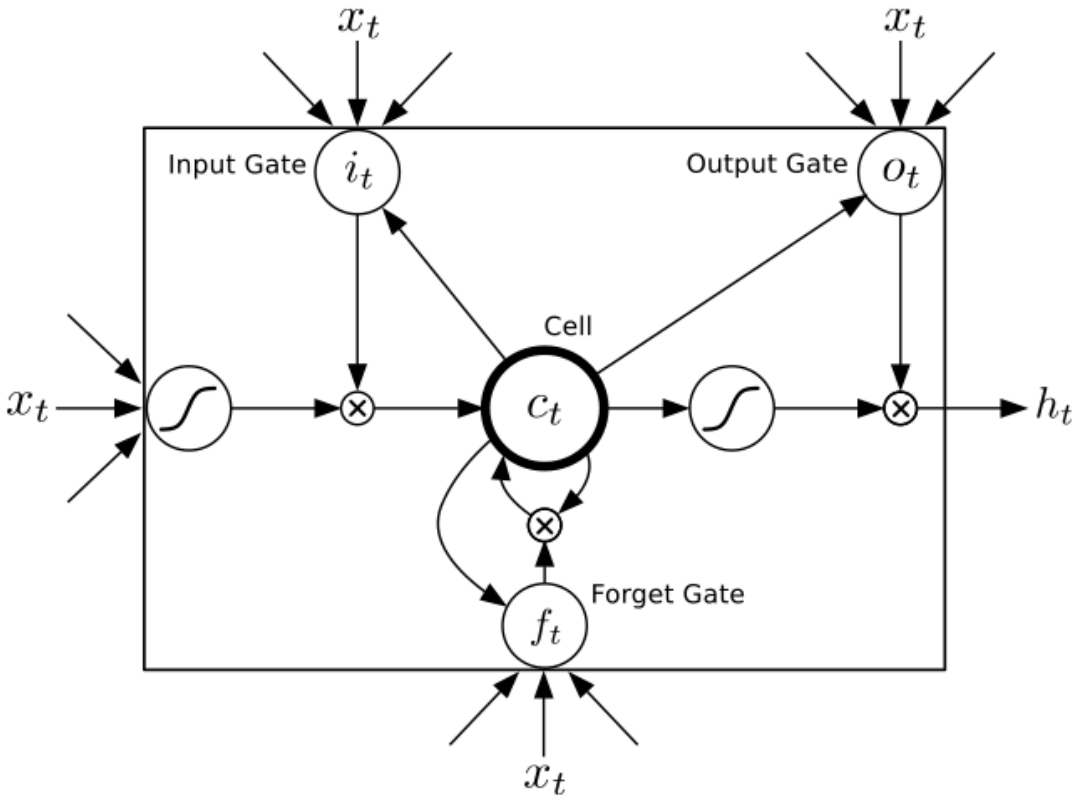
- *Long short term memory* (LSTM) networks
- Three internal states: input, output, forget



Common Building Block: LSTM Units

- *Long short term memory* (LSTM) networks
- In equation form:

$$
\begin{aligned}
i_t &= \sigma\left(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i\right) \\
f_t &= \sigma\left(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f\right) \\
c_t &= f_t c_{t-1} + i_t \tanh\left(W_{xc}x_t + W_{hc}h_{t-1} + b_c\right) \\
o_t &= \sigma\left(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o\right) \\
h_t &= o_t \tanh(c_t)
\end{aligned}
$$



[Sutskever et al., “Sequence to Sequence Learning with Neural Networks”, 2014]

Recurrent Neural Networks

- LSTM networks powerful tool for sequences over time
- Alternatives:
  - Gated Recurrent Units (GRUs)
  - Time convolutional networks (TCNs)
  - …

[Chung et al., "Empirical evaluation of gated recurrent neural networks on sequence modeling",2014]

[Bai et al., "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling", 2018]

Deep Learning Frameworks

Main frameworks

(Python, C++, Java)

(Python, backends support other languages)

(Python)

(C++, Python, Matlab)

Currently less frequently used

(Python)

(Python)

(Python, C++)

(Python, C++, C#)

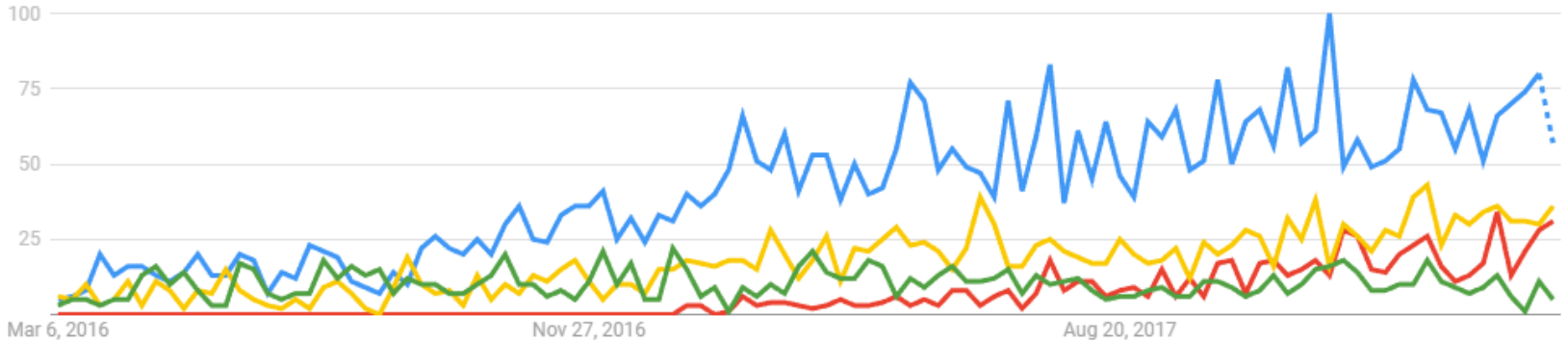
(Matlab)

(Python, Java, Scala)

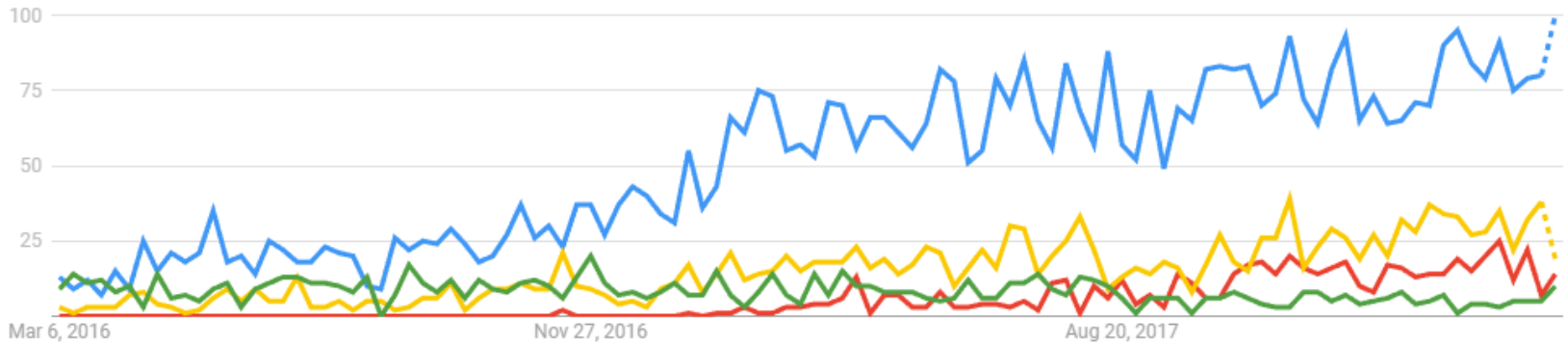
(Python, C++, and others)

Popularity

Google Trends for search terms: “[name] github”



Google Trends for search terms: “[name] tutorial”



TensorFlow™
Keras
PYTORCH
Caffe

SIGGRAPH ASIA 2018 TOKYO

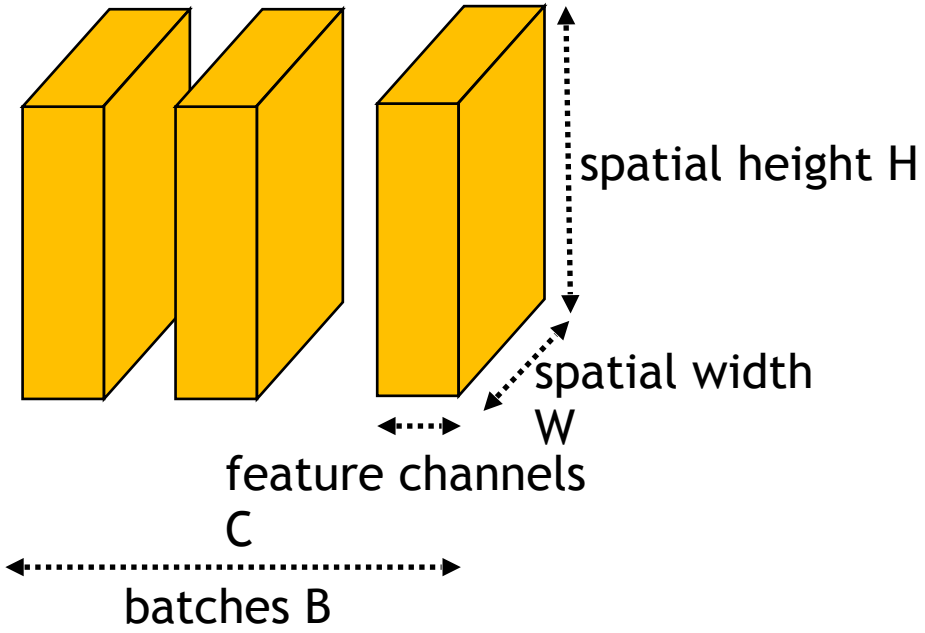
Typical Training Steps

```
for i = 1 .. max_iterations

    input, ground_truth = load_minibatch(data, i)

    output = network_evaluate(input, parameters)

    loss = compute_loss(output, ground_truth)

    # gradients of loss with respect to parameters
    gradients = network_backpropagate(loss, parameters)

    parameters = optimizer_step(parameters, gradients)
```

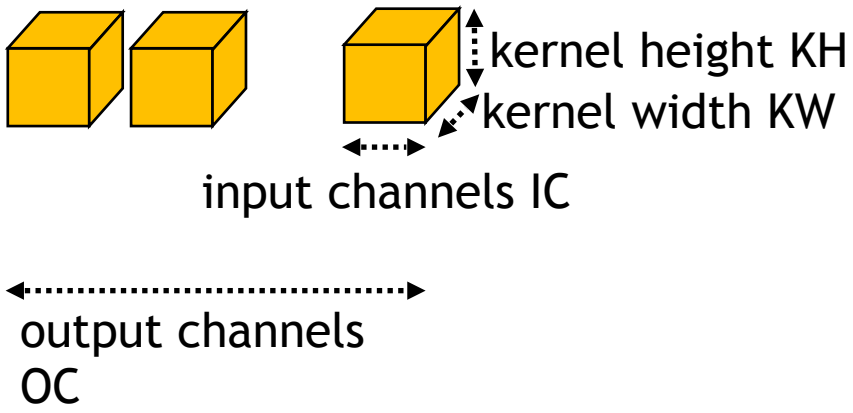
Tensors

- Frameworks typically represent data as tensors
- Examples:

4D input data: B x C x H x W



4D convolution kernel: OC x IC x KH x KW



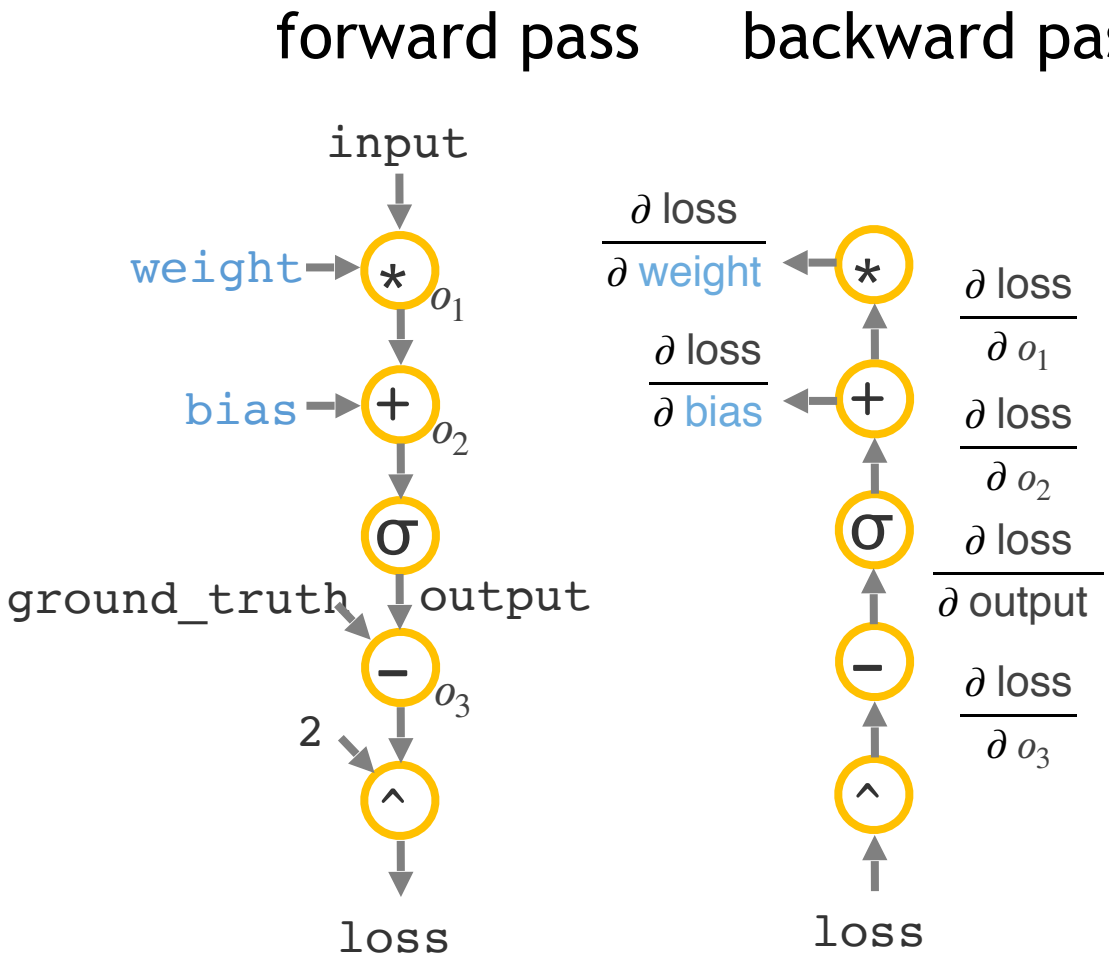
What Does a Deep Learning Framework Do?

- Tensor math
- Common network operations/layers
- Gradients of common operations
- Backpropagation
- Optimizers
- GPU implementations of the above
- usually: data loading, network parameter saving/loading
- sometimes: distributed computing

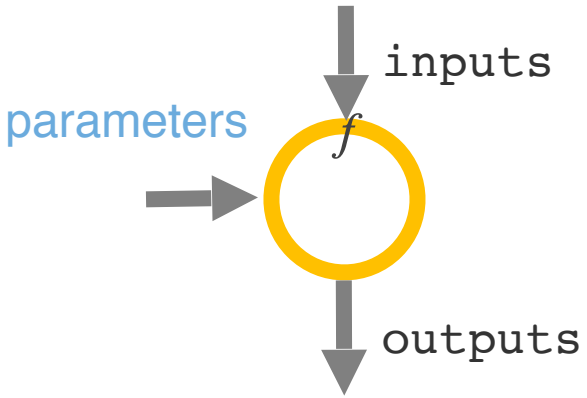
Automatic Differentiation & the Computation Graph

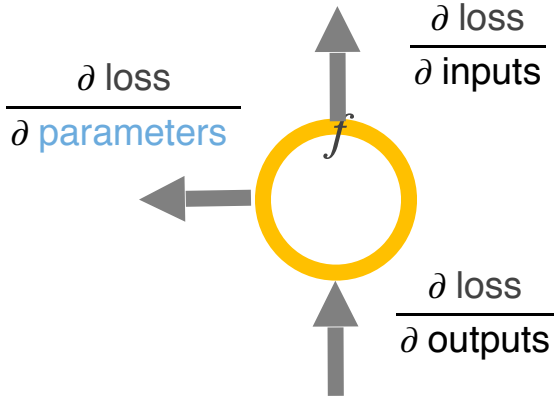
```
parameters = (weight, bias)

output = σ(weight * input + bias)

loss = (output - ground_truth)^2

# gradients of loss with respect to parameters
gradients = backpropagate(loss, parameters)
```

Since loss is a scalar, the gradients are the same size as the parameters



Automatic Differentiation & the Computation Graph






```
outputs = forward(inputs, )
```

```
,    = backward()
```

Static vs Dynamic Computation Graphs

- Static analysis allows optimizations and distributing workload
- Dynamic graphs make data-driven control flow easier
- In static graphs, the graph is usually defined in a separate ‘language’
- Static graphs have less support for debugging

define once,
evaluate during training

Static

```
x = Variable()
loss = if_node(x < parameter[0],
    x + parameter[0],
    x - parameter[1])

for i = 1 .. max_iterations
    x = data()
    run(loss)
    backpropagate(loss, parameters)
```

define implicitly by running operations,
a new graph is created in each evaluation

Dynamic

```
for i = 1 .. max_iterations
    x = data()
    if x < parameter[0]
        loss = x + parameter[0]
    else
        loss = x – parameter[1]
    backpropagate(loss, parameters)
```

Tensorflow

- Currently the largest community
- Static graphs (dynamic graphs are in development: Eager Execution)
- Good support for deployment
- Good support for distributed computing
- Typically slower than the other three main frameworks on a single GPU

PyTorch

- Fast growing community
- Dynamic graphs
- Distributed computing is in development (some support is already available)
- Intuitive code, easy to debug and good for experimenting with less traditional architectures due to dynamic graphs
- Very Fast

Keras

- A high-level interface for various backends (Tensorflow, CNTK, Theano)
- Intuitive high-level code
- Focus on optimizing time from idea to code
- Static graphs

Caffe

- Created earlier than Tensorflow, PyTorch or Keras
- Less flexible and less general than the other three frameworks
- Static graphs
- Legacy - to be replaced by Caffe2: focus is on performance and deployment
  - Facebook's platform for Detectron (Mask-RCNN, DensePose, …)

Converting Between Frameworks

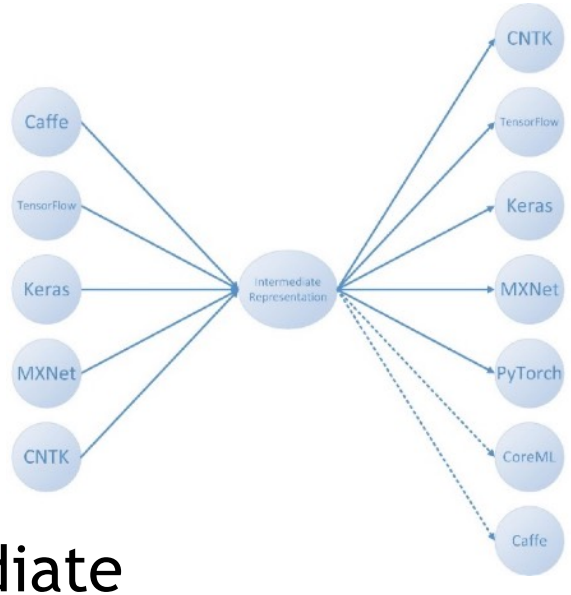
- Example: develop in one framework, deploy in another
- Currently: a large range of converters, but no clear standard
- Standardized model formats are in development

from https://github.com/ysh329/deep-learning-model-convertor

convertor	tensorflow	pytorch	keras	caffe	caffe2	CNTK	chainer	mxnet
tensorflow	-	pytorch-tf/ MMdnn	model-converters/ nn_toolsconvert-to-tensorflow/MMdnn	MMdnn/ nn_tools	None	crosstalk/MMdnn	None	MMdnn
pytorch	pytorch2keras (over Keras)	-	Pytorch2keras/ nn-transfer	Pytorch2caffe/ pytorch-caffe-darknet-convert	onnx-caffe2	ONNX	None	None
keras	nn_tools /convert-to-tensorflow/ keras_to_tensorflow/ keras_to_tensorflow/ MMdnn	MMdnn/ nn-transfer	-	MMdnnnn_tools	None	MMdnn	None	MMdnn
caffe	MMdnn/nn_tools/ caffe-tensorflow	MMdnn/ pytorch-caffe-darknet-convert/ pytorch-resnet	caffe_weight_converter / caffe2keras/nn_tools/ kerascaffe2keras/ Deep_Learning_Model_ Converter/MMdnn	-	CaffeToCaffe2	crosstalkcaffe/ CaffeConverterMMdnn	None	mxnet/tools/ caffe_converter/ ResNet_caffe2mxnet/ MMdnn
caffe2	None	ONNX	None	None	-	ONNX	None	None
CNTK	MMdnn	ONNX MMdnn	MMdnn	MMdnn	ONNX	-	None	MMdnn
chainer	None	chainer2pytorch	None	None	None	None	-	None
mxnet	MMdnn	MMdnn	MMdnn	MMdnn/MXNet2Caffe/ Mxnet2Caffe	None	MMdnn	None	-

- Standard format for models
- Native support in development for Pytorch, Caffe2, Chainer, CNTK, and MxNet
- Converter in development for Tensorflow

MMdnn



- Converters available for several frameworks
- Common intermediate representation, but no clear standard

Thank you!

http://geometry.cs.ucl.ac.uk/creativeai/

CreativeAI: Deep Learning for Graphics

Feature Visualization

Niloy Mitra
UCL

Iasonas Kokkinos
UCL

Paul Guerrero
UCL

Nils Thuerey
TU Munich

Tobias Ritschel
UCL

Timetable

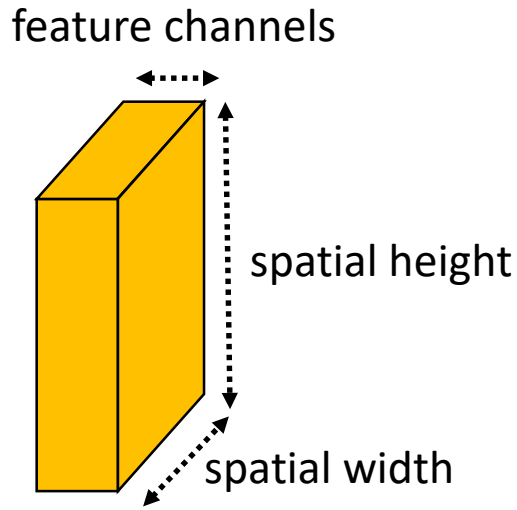
			Niloy	Paul	Nils
Theory and Basics	Introduction	2:15 pm	X	X	X
	Machine Learning Basics	~ 2:25 pm	X		
	Neural Network Basics	~ 2:55 pm			X
	Feature Visualization	~ 3:25 pm		X	
	Alternatives to Direct Supervision	~ 3:35 pm		X	
	15 min. break				
State of the Art	Image Domains	4:15 pm		X	
	3D Domains	~ 4:45 pm	X		
	Motion and Physics	~ 5:15 pm			X
	Discussion	~ 5:45 pm	X	X	X

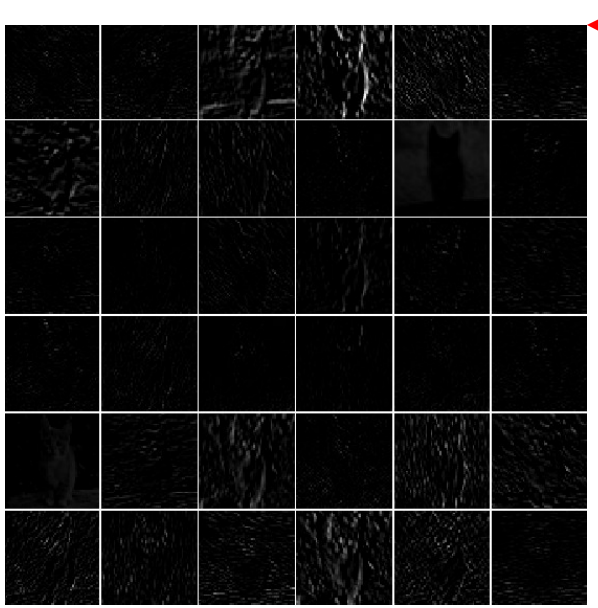
What to Visualize

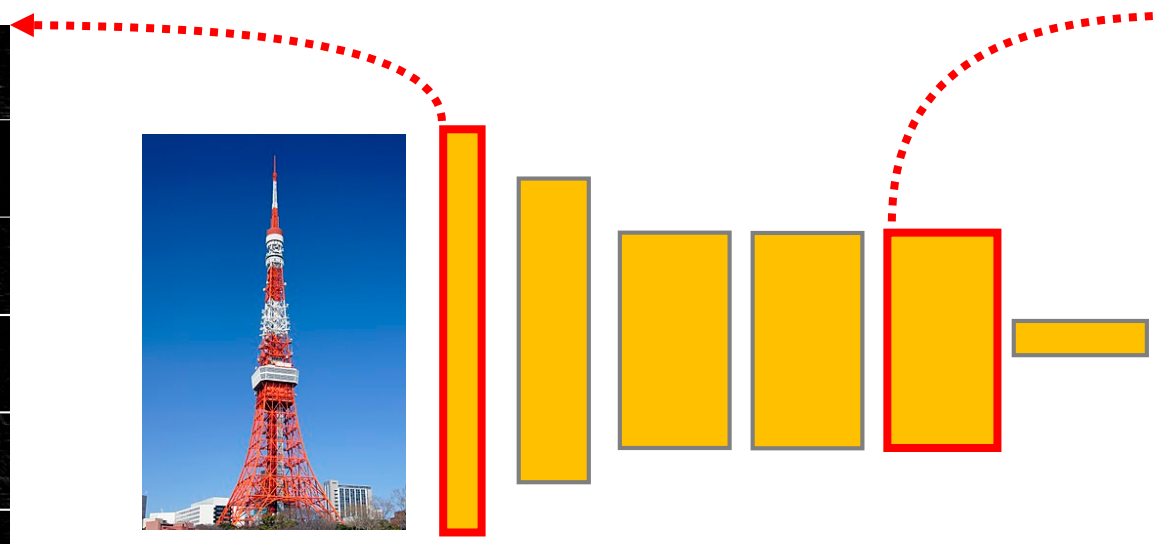
- Features (activations)
- Weights (filter kernels in a CNN)
- Attribution: input parts that contribute to a given activation
- Inputs that maximally activate some class probabilities or features
- Inputs that maximize the error (adversarial examples)

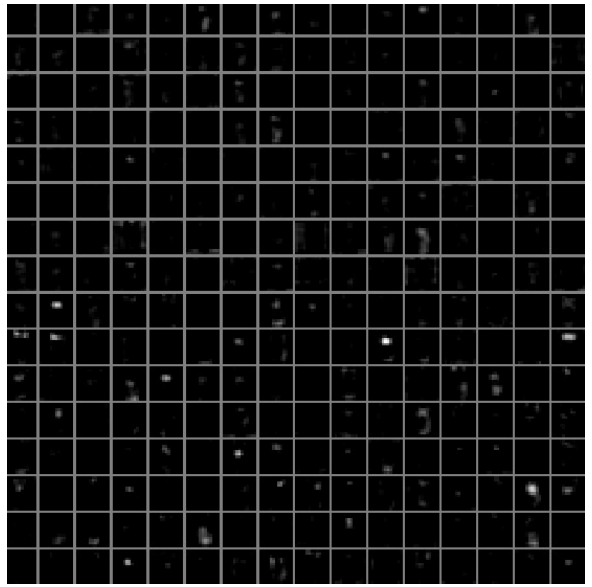
Feature Samples

- In good training, features are usually sparse
- Can find “dead” features that never activate







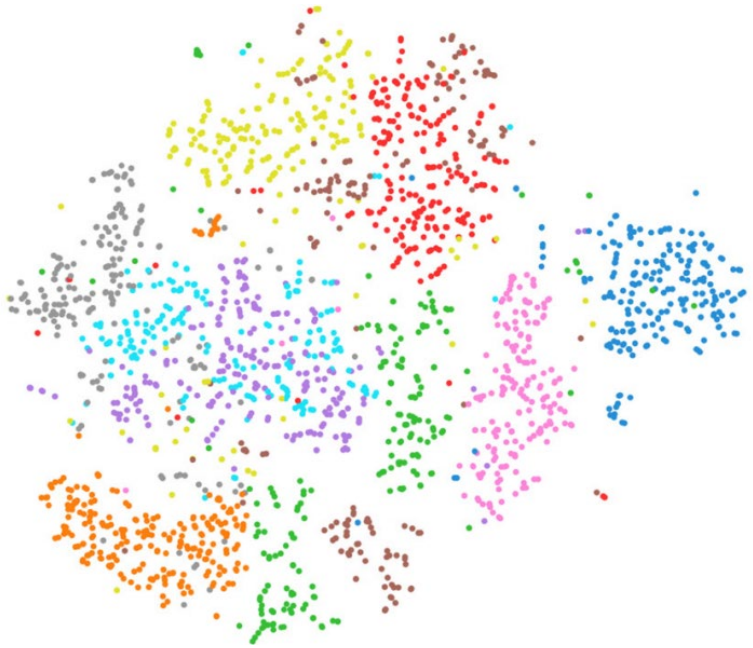


Images from: http://cs231n.github.io/understanding-cnn/

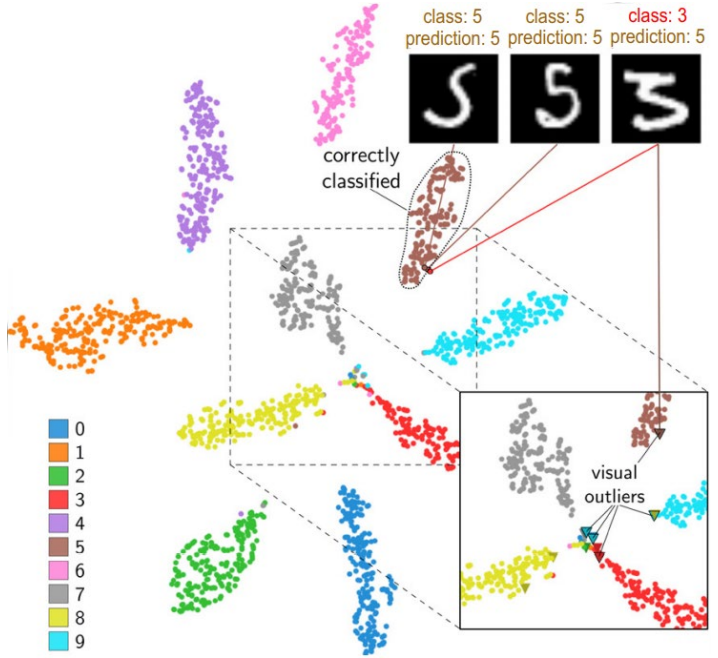
Feature Distribution using t-SNE

- Low-dimensional embedding of the features for visualization

t-SNE embedding of image features in a CNN layer



before training



after training

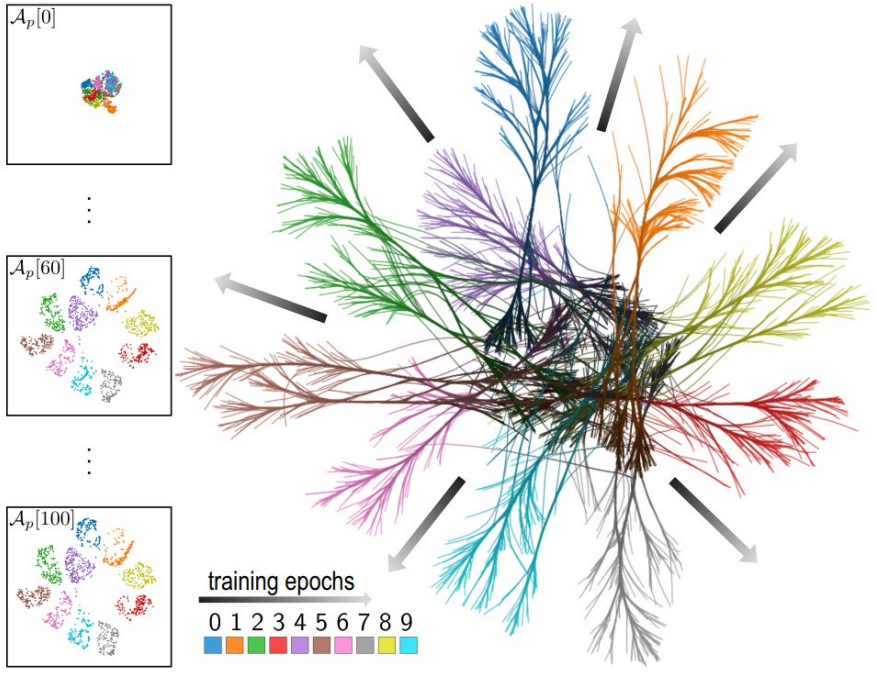
t-SNE embedding of MNIST (images of digits) features in a CNN layer, colored by class

Images from: https://cs.stanford.edu/people/karpathy/cnnembed/ and
Rauber et al. *Visualizing the Hidden Activity of Artificial Neural Networks*. TVCG 2017

Feature Distribution using t-SNE

- Low-dimensional embedding of the features for visualization

t-SNE embedding of image features in a CNN layer



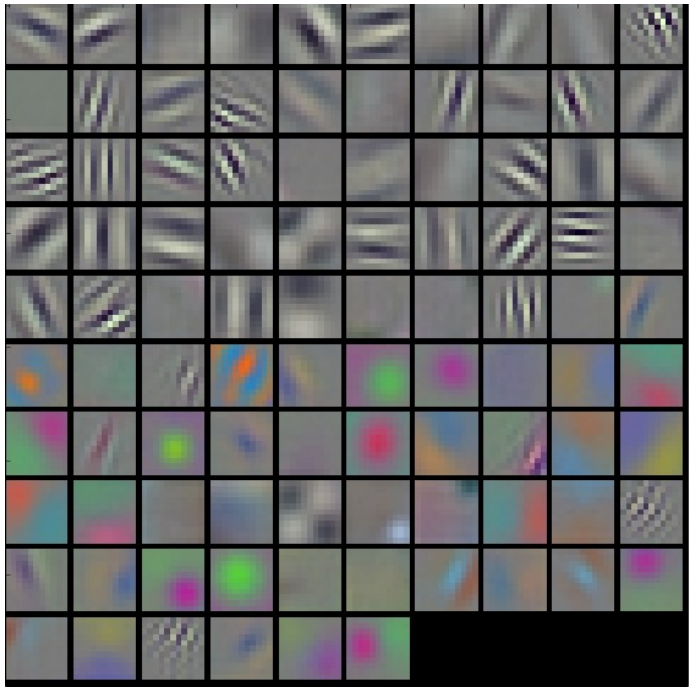
evolution during training

t-SNE embedding of MNIST (images of digits) features in a CNN layer, colored by class

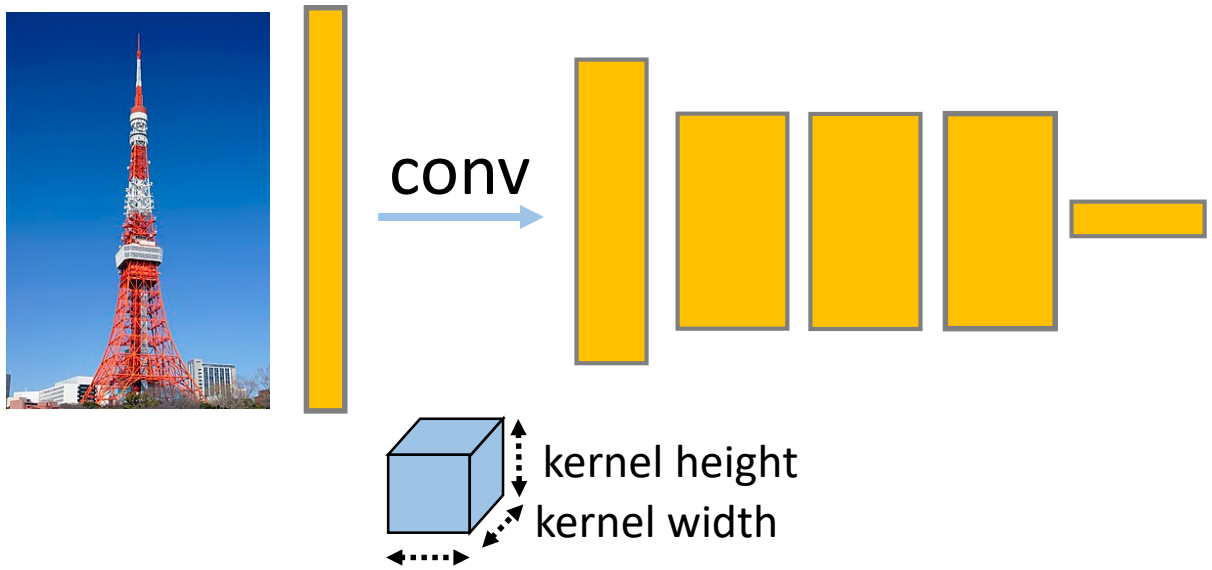
Images from: https://cs.stanford.edu/people/karpathy/cnnembed/ and
Rauber et al. *Visualizing the Hidden Activity of Artificial Neural Networks*. TVCG 2017

Weights (Filter Kernels)

- Useful for CNN kernels, not useful for fully connected layers
- Kernels are typically smooth and diverse after a successful training



first layer filters of AlexNet



Images from: http://cs231n.github.io/understanding-cnn/

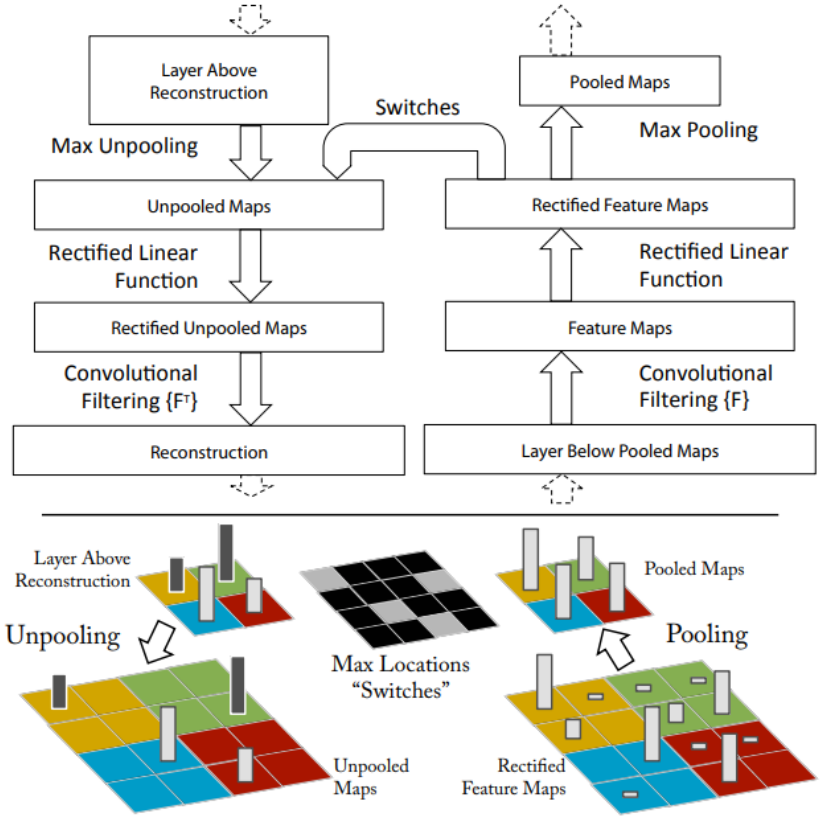
Code Examples

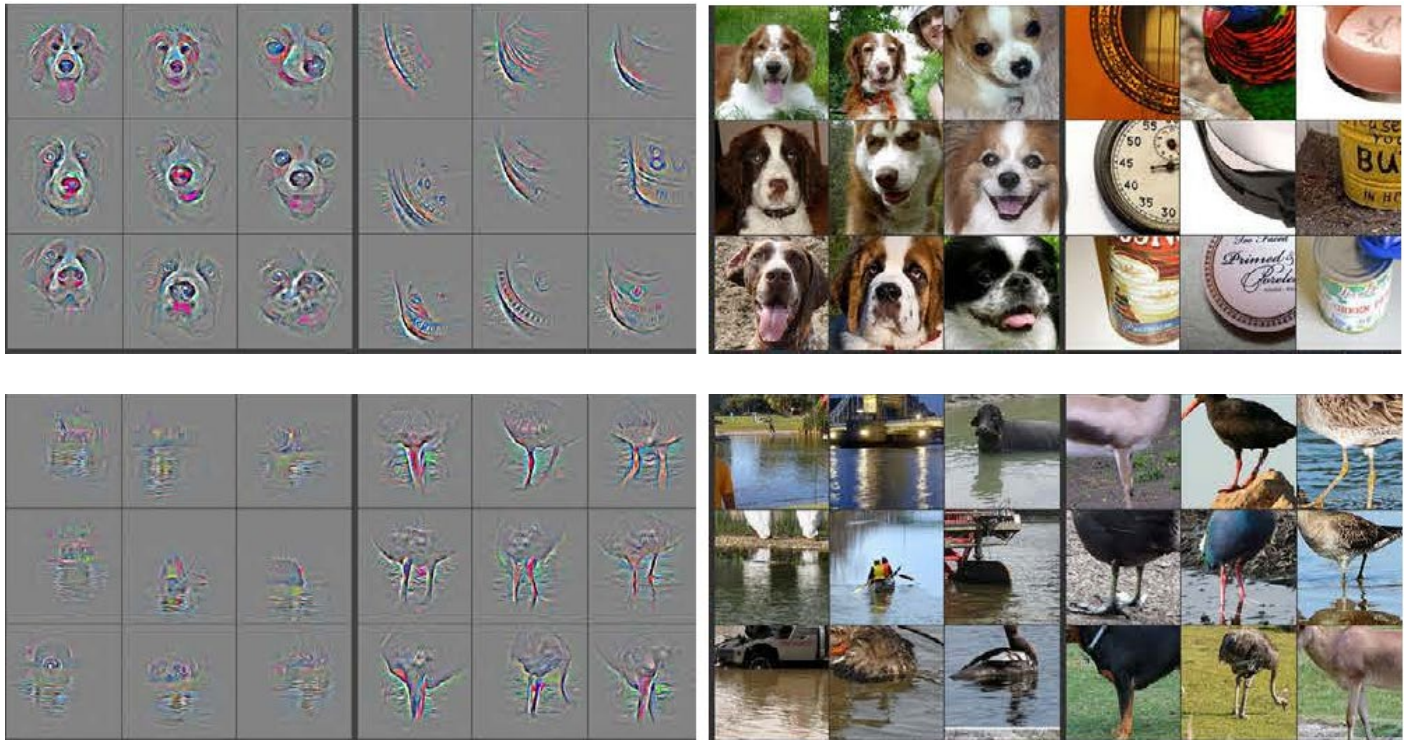
Filter Visualization

http://geometry.cs.ucl.ac.uk/creativeai

Attribution by Approximate Inversion

- Reconstruct Input from a given feature channel
- What information does the feature channel focus on?

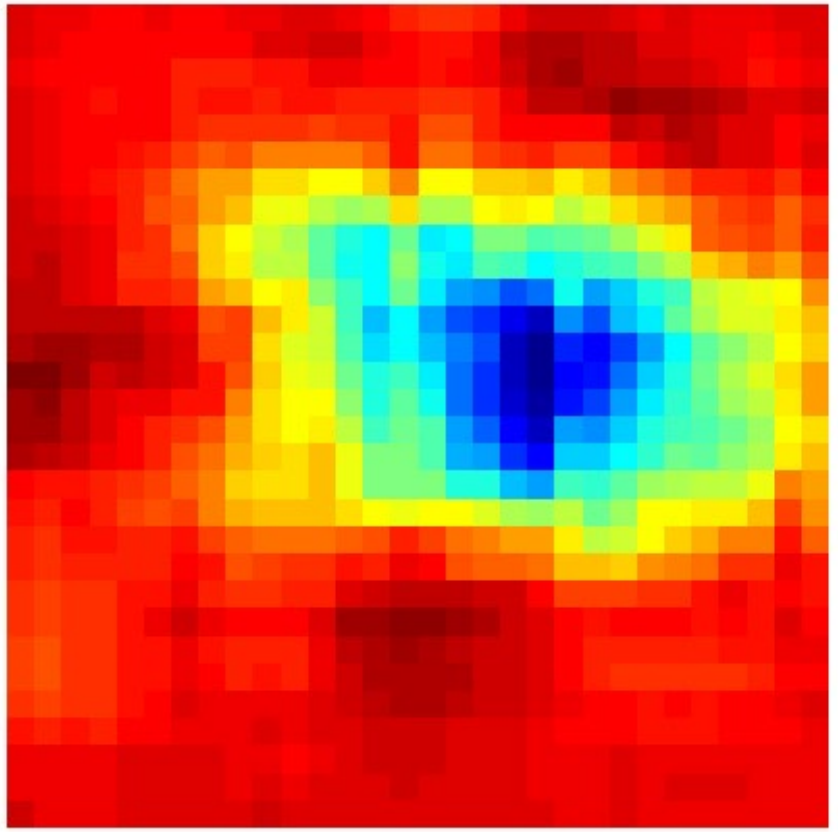




Zeiler and Fergus, *Visualizing and Understanding Convolutional Networks*, ECCV 2014

Perturbation-based Attribution

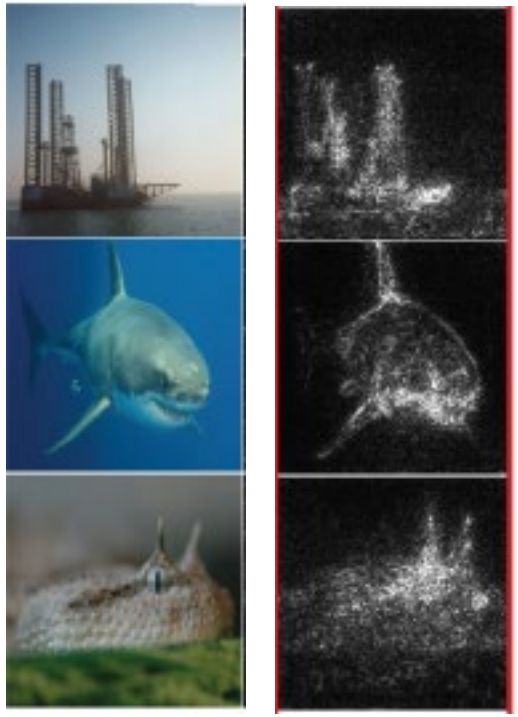
Probability for correct classification when centering the box at each pixel.

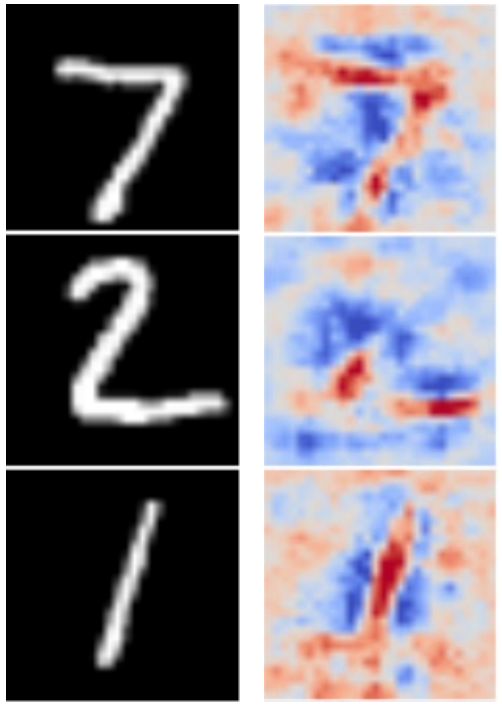


Zeiler and Fergus, *Visualizing and Understanding Convolutional Networks*, ECCV 2014

Gradient-based Attribution

- Derivative of class probability w.r.t input pixels
- Which parts of the input is the class probability sensitive to?



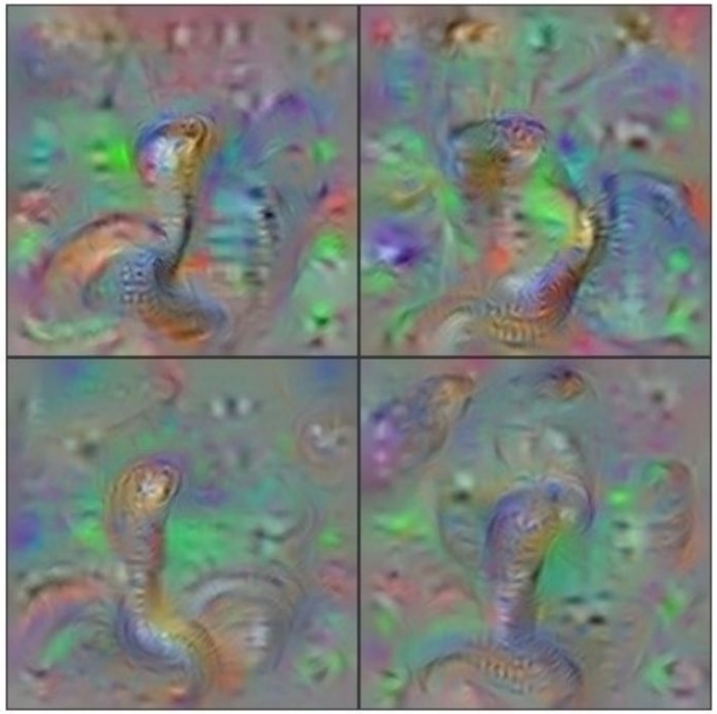


Smilkov et al., *SmoothGrad: removing noise by adding noise*, arXiv 2017

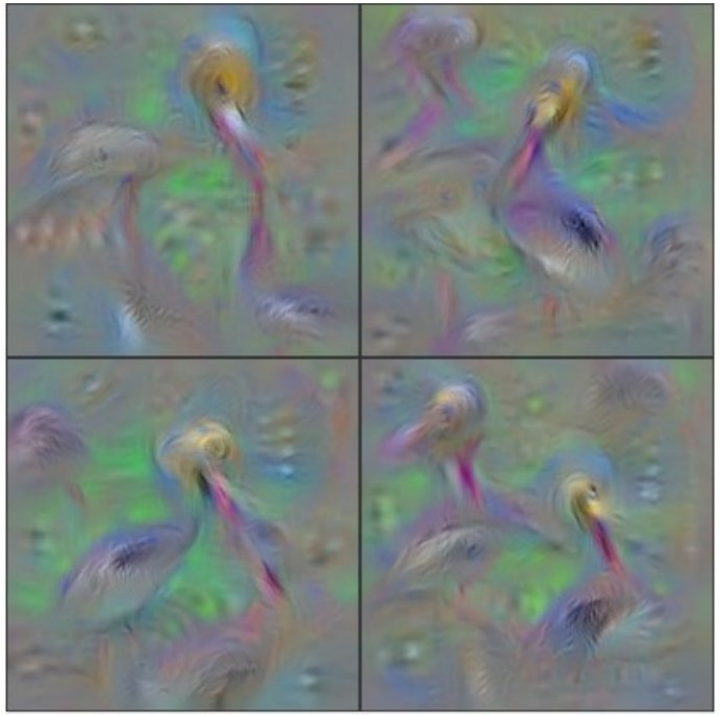
Inputs that Maximize Feature Response

Local maxima of the response for class:

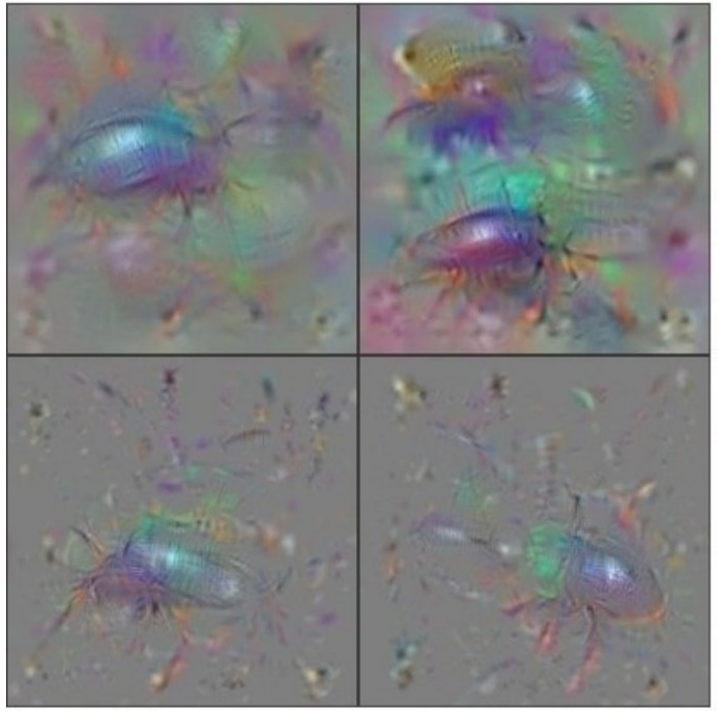
Indian Cobra



Pelican



Ground Beetle



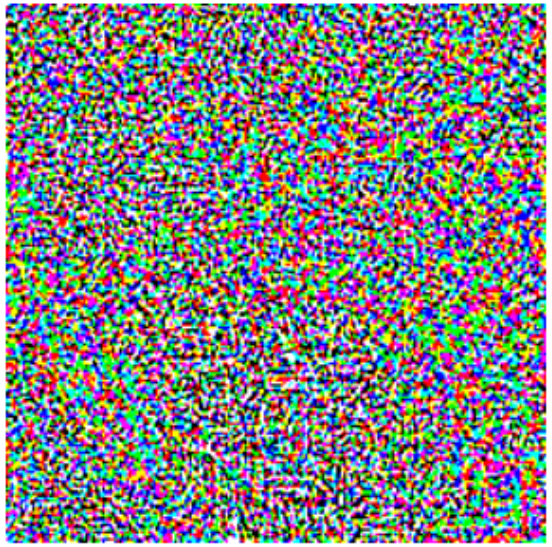
Images from: Yosinski et al. *Understanding Neural Networks Through Deep Visualization*. ICML 2015

Inputs that Maximize the Error

$$\max_{\delta \in \Delta} \mathcal{L}(x + \delta, y; \theta) \qquad \Delta = \{\delta \in \mathbb{R}^d \mid \|\delta\|_p \leq \varepsilon\}$$

x

$+ .007 \times$



δ

$=$

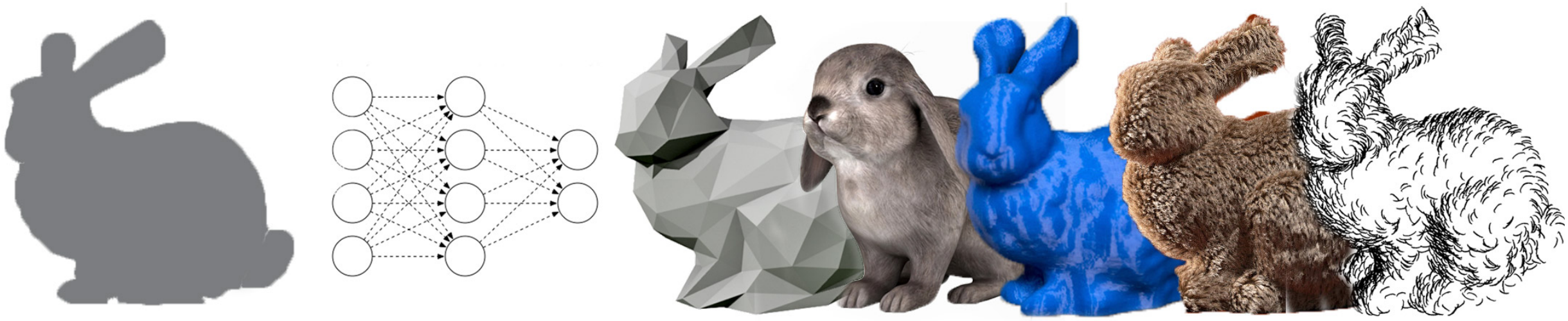
$x + \delta$

"Panda" 55.7% conf.

"Gibbon" 99.3% conf.

Images from: Goodfellow et al. *Explaining and Harnessing Adversarial Examples*. ICLR 2015

Course Information (slides/code/comments)



http://geometry.cs.ucl.ac.uk/creativeai/

CreativeAI: Deep Learning for Graphics

Alternatives to Direct Supervision

Niloy Mitra
UCL

Iasonas Kokkinos
UCL

Paul Guerrero
UCL

Nils Thuerey
TUM

Tobias Ritschel
UCL

Timetable

			Niloy	Paul	Nils
Theory and Basics	Introduction	2:15 pm	X	X	X
	Machine Learning Basics	~ 2:25 pm	X		
	Neural Network Basics	~ 2:55 pm			X
	Feature Visualization	~ 3:25 pm		X	
	Alternatives to Direct Supervision	~ 3:35 pm		X	
	15 min. break				
State of the Art	Image Domains	4:15 pm		X	
	3D Domains	~ 4:45 pm	X		
	Motion and Physics	~ 5:15 pm			X
	Discussion	~ 5:45 pm	X	X	X

Unsupervised Learning

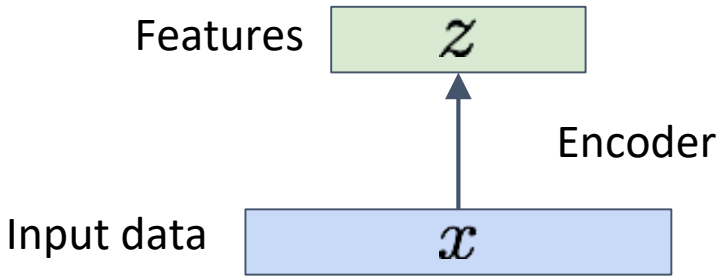
- There is no direct ground truth for the quantity of interest

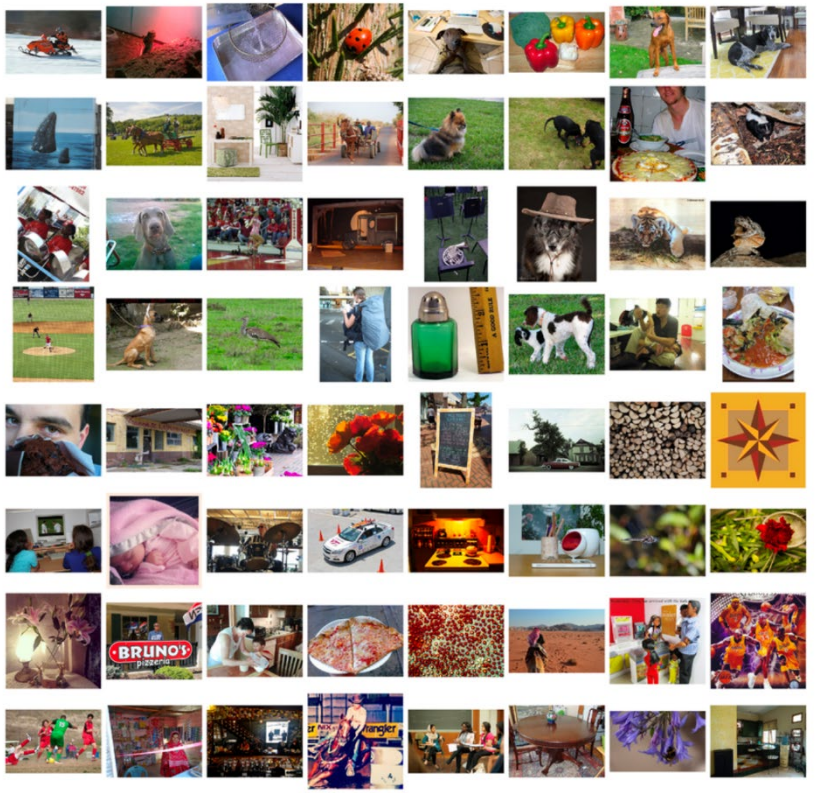
- Autoencoders
- Variational Autoencoders (VAEs)
- Generative Adversarial Networks (GANs)

Autoencoders

Goal: Meaningful features that capture the main factors of variation in the dataset

- These are good for classification, clustering, exploration, generation, …
- We have no ground truth for them



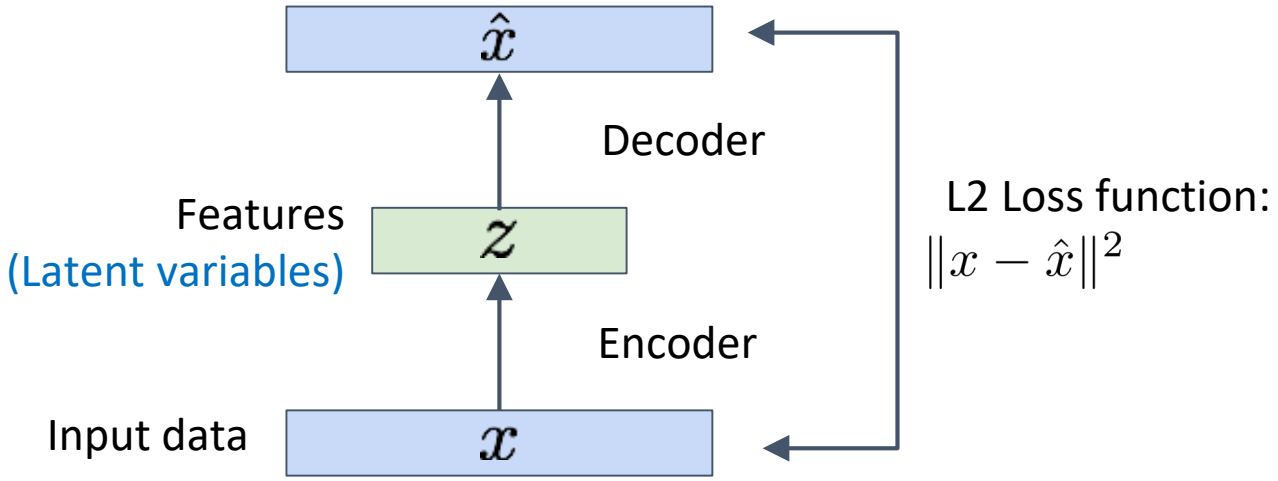


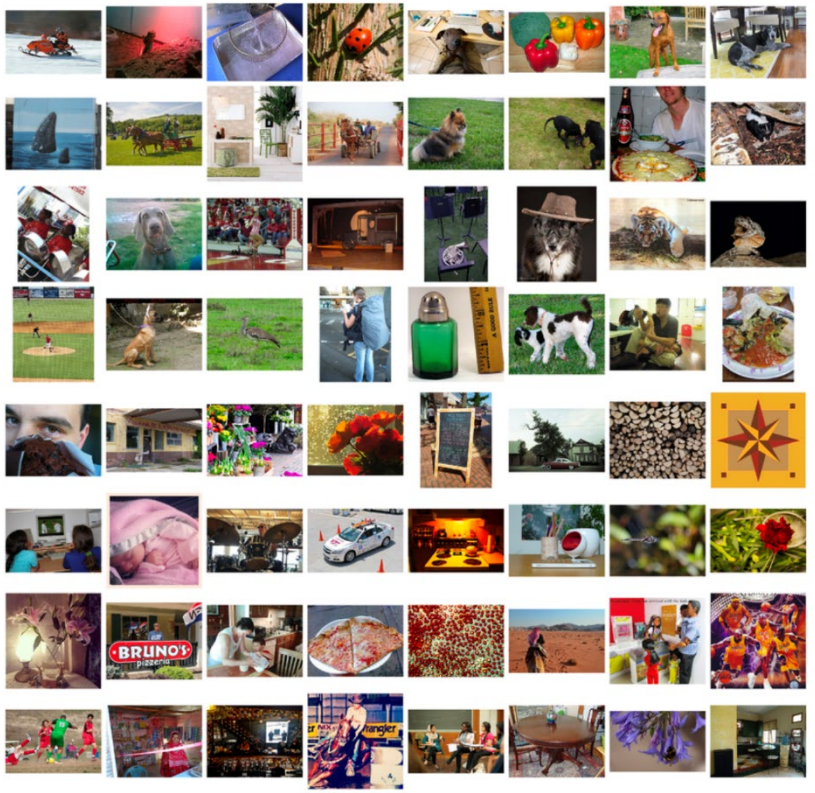
Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Autoencoders

Goal: Meaningful features that capture the main factors of variation

Features that can be used to reconstruct the image



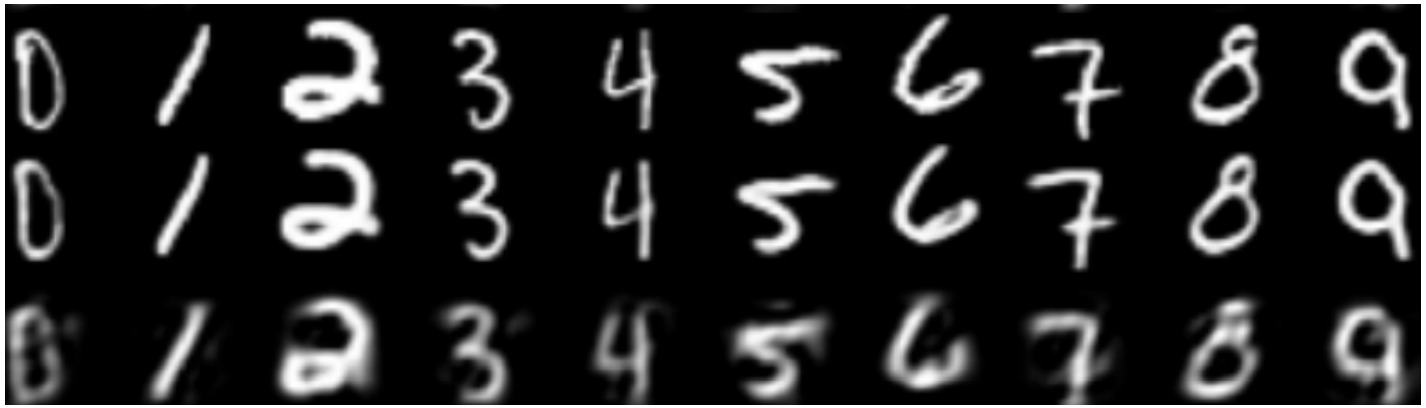


Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Autoencoders

Linear Transformation for Encoder and Decoder give result close to PCA

Deeper networks give better reconstructions, since basis can be non-linear



Original

Autoencoder

PCA

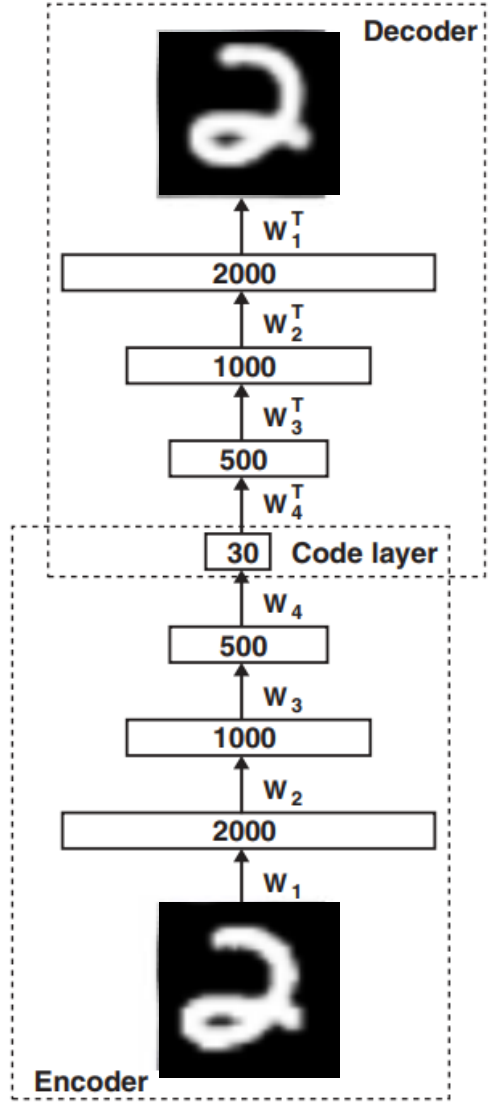
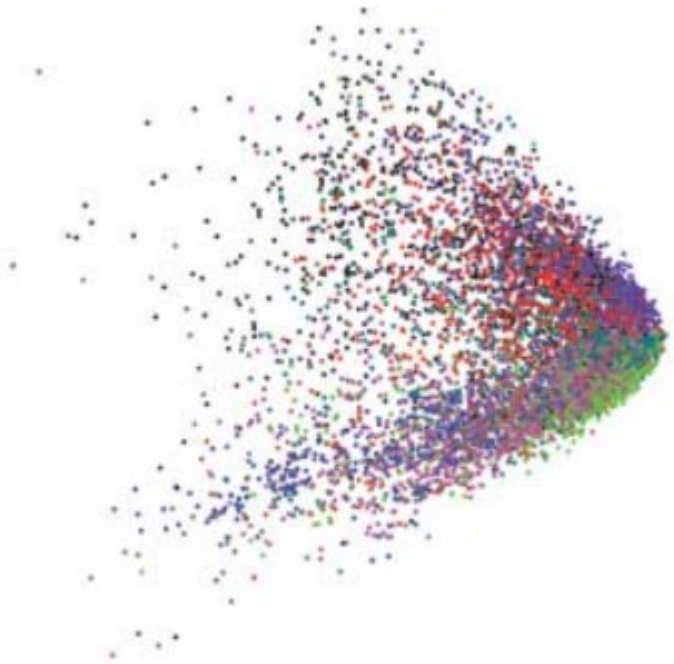


Image Credit: Reducing the Dimensionality of Data with Neural Networks, . Hinton and Salakhutdinov

Example: Document Word Prob. → 2D Code

PCA-based



Autoencoder

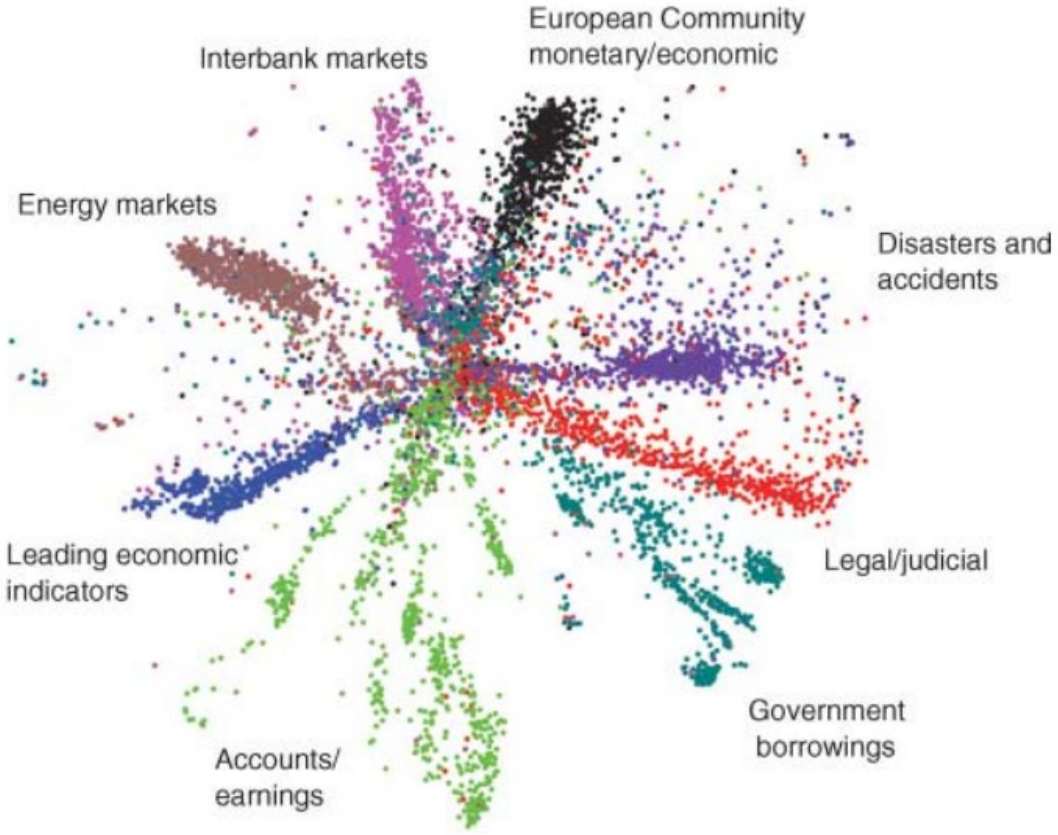
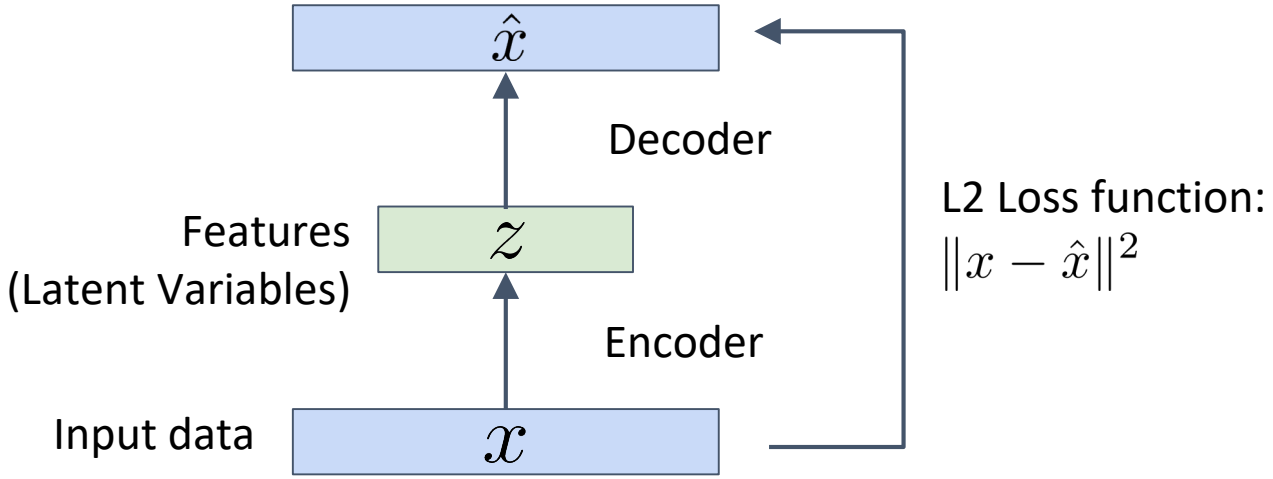


Image Credit: Reducing the Dimensionality of Data with Neural Networks, Hinton and Salakhutdinov

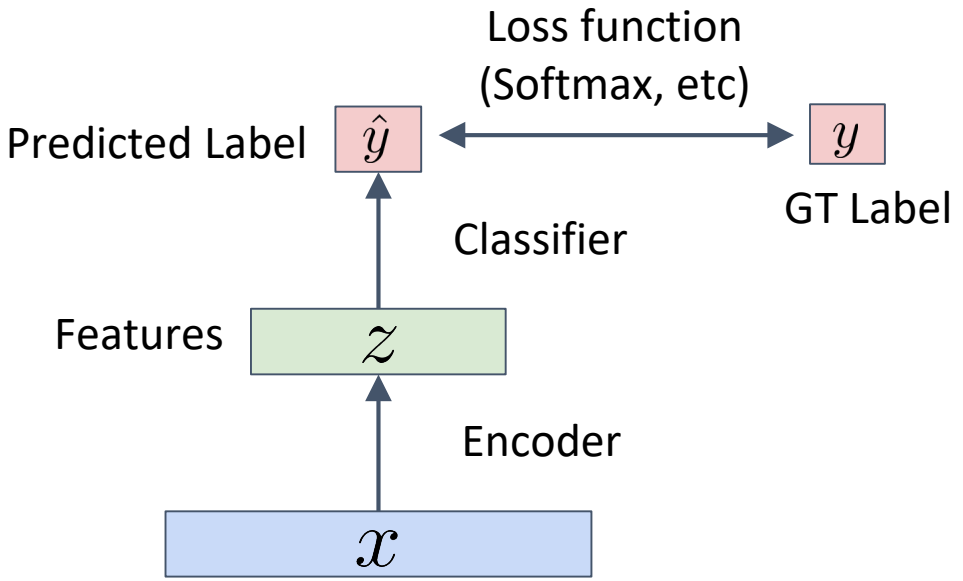
Example: Semi-Supervised Classification

- Many images, but few ground truth labels

start unsupervised
train autoencoder on many images



supervised fine-tuning
train classification network on labeled images



Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n

Code example

Autoencoder

geometry.cs.ucl.ac.uk/creativeai

Generative Models

- Assumption: the dataset are samples from an unknown distribution $p_{\text{data}}(x)$
- Goal: create a new sample from $p_{\text{data}}(x)$ that is not in the dataset

Dataset

Generated

Image credit: *Progressive Growing of GANs for Improved Quality, Stability, and Variation,* Karras et al.

Generative Models

- Assumption: the dataset are samples from an unknown distribution $p_{\text{data}}(x)$
- Goal: create a new sample from $p_{\text{data}}(x)$ that is not in the dataset

Dataset

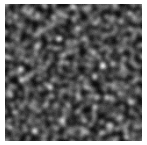
Generated

Image credit: *Progressive Growing of GANs for Improved Quality, Stability, and Variation,* Karras et al.

Generative Models

$$p_\theta(x) \approx p_{\text{data}}(x)$$

Generator with parameters θ



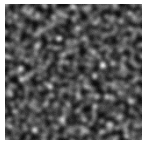
$$p(z)$$

known and
easy to sample from

Generative Models

$p_\theta(x) \approx p_{\text{data}}(x)$

Generator with parameters θ



$p(z)$

known and easy to sample from

How to measure similarity of $p_\theta(x)$ and $p_{\text{data}}(x)$?

1) Likelihood of data in $p_\theta(x)$

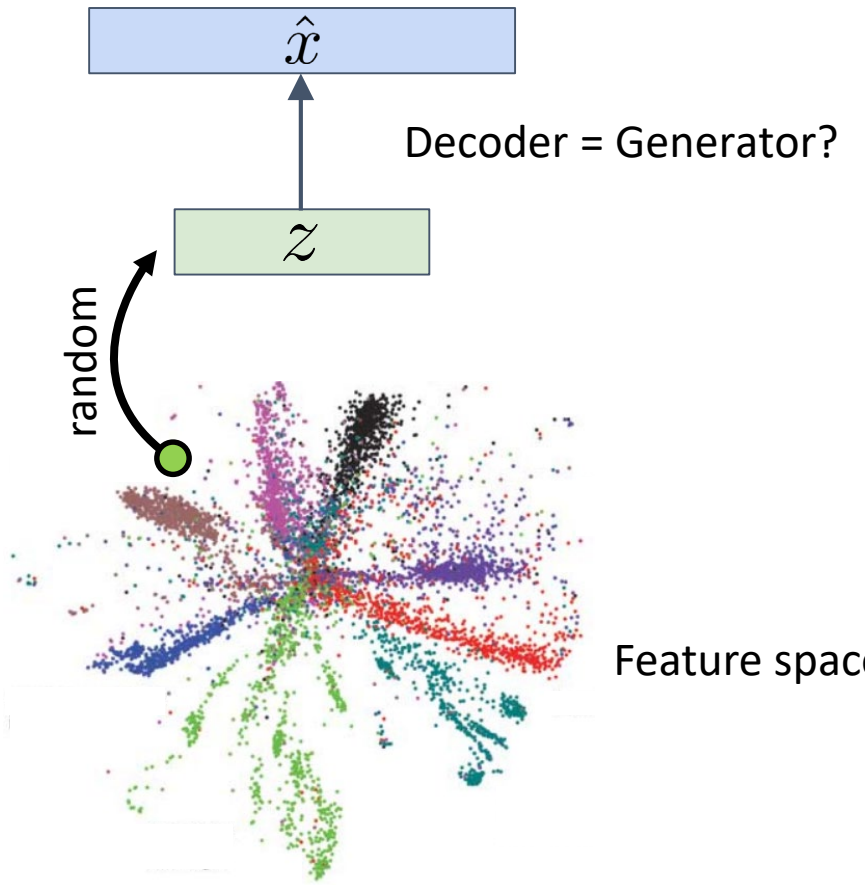
Variational Autoencoders (VAEs)

2) Adversarial game:

Discriminator distinguishes $p_\theta(x)$ and $p_{\text{data}}(x)$ vs *Generator* makes it hard to distinguish

Generative Adversarial Networks (GANs)

Autoencoders as Generative Models?

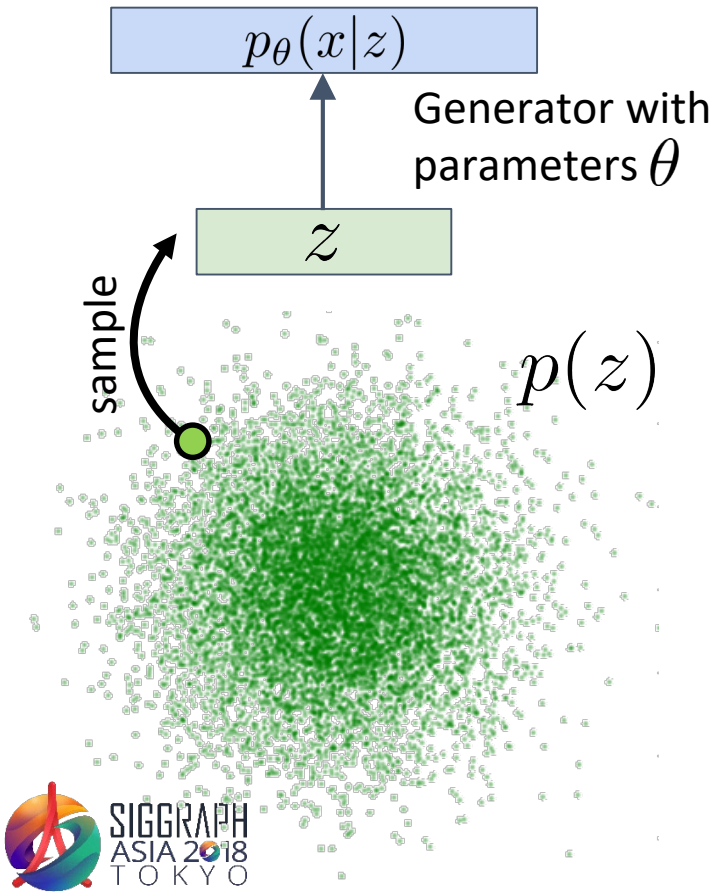


Feature space / latent space

- A trained decoder transforms some features z to approximate samples from $p_{\text{data}}(x)$
- What happens if we pick a random z?
- We do not know the distribution $p(z)$ of features that decode to likely samples

Image Credit: *Reducing the Dimensionality of Data with Neural Networks*, Hinton and Salakhutdinov

Variational Autoencoders (VAEs)



- Pick a parametric distribution $p(z)$ for features
- The generator maps $p(z)$ to an image distribution $p_\theta(x)$ (where θ are parameters)

$$p_\theta(x) = \int p_\theta(x|z)\ p(z)\ dz$$

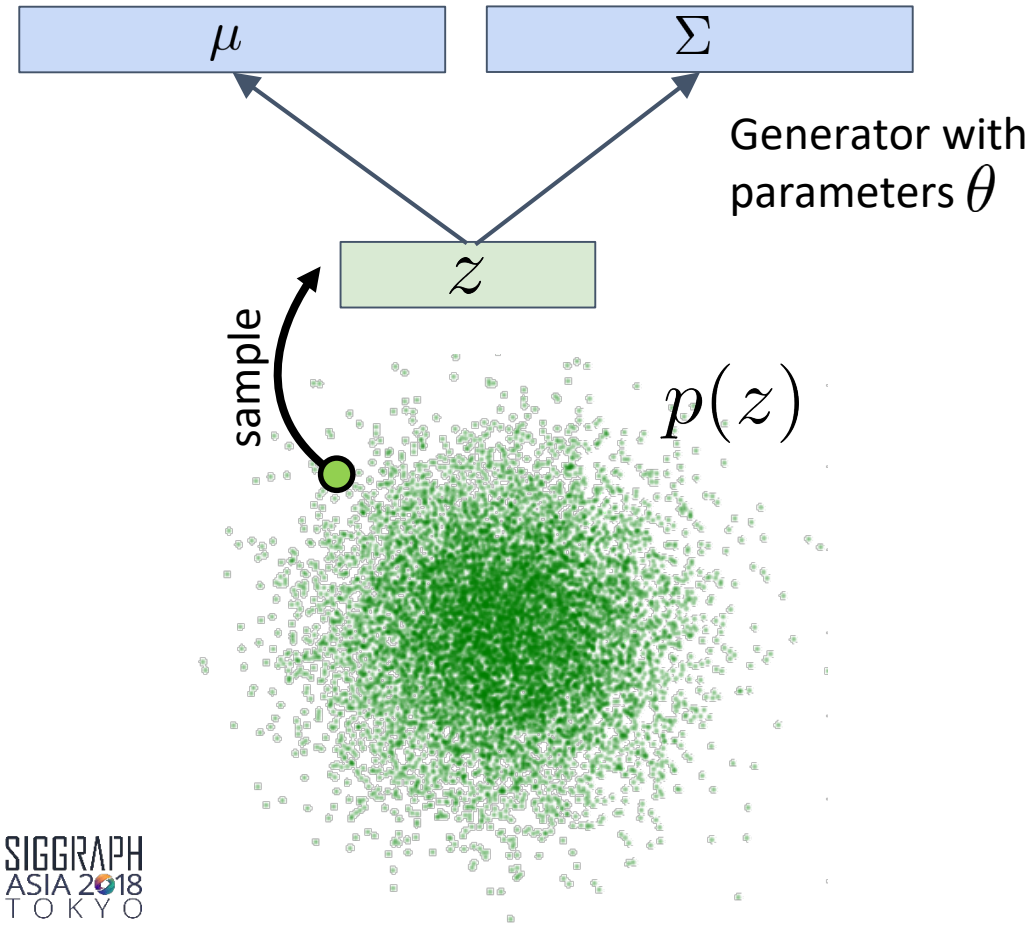
- Train the generator to maximize the likelihood of the data in $p_\theta(x)$:

$$\max_\theta \sum_{x_i \in \text{data}} \log p_\theta(x_i)$$

Outputting a Distribution

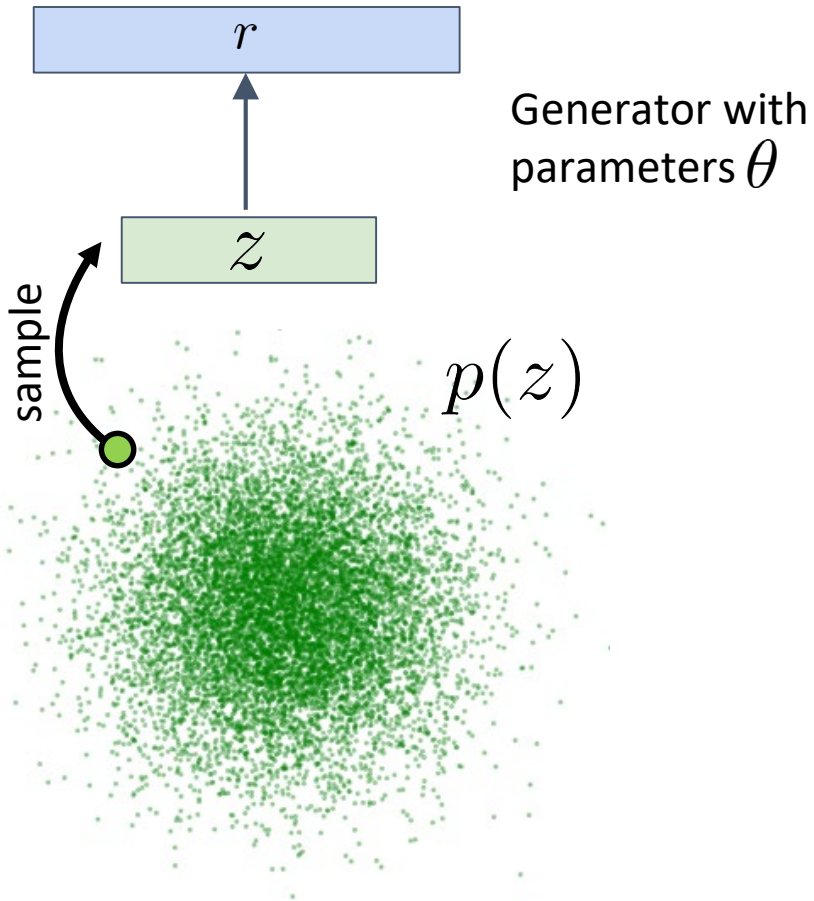
Normal distribution

$$p_\theta(x|z) = N(x; \mu(z), \Sigma(z))$$

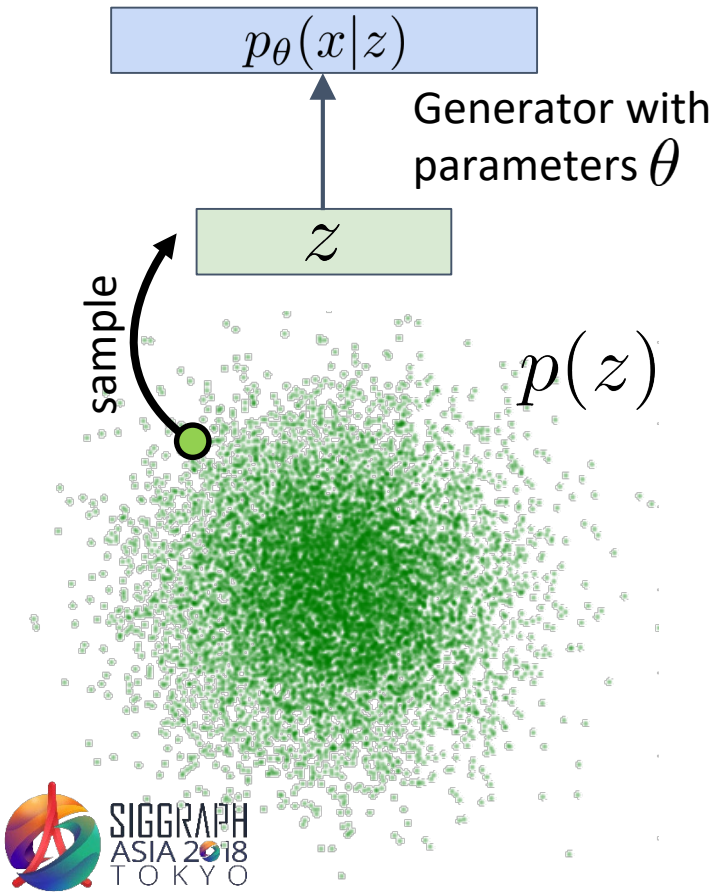


Bernoulli distribution

$$p_\theta(x|z) = Bern(x; r(z))$$



Variational Autoencoders (VAEs)



- Pick a parametric distribution $p(z)$ for features
- The generator maps $p(z)$ to an image distribution $p_\theta(x)$ (where θ are parameters)

$$p_\theta(x) = \int p_\theta(x|z)\ p(z)\ dz$$

- Train the generator to maximize the likelihood of the data in $p_\theta(x)$:

$$\max_\theta \sum_{x_i \in \text{data}} \log p_\theta(x_i)$$

Variational Autoencoders (VAEs): Naïve Sampling (Monte-Carlo)

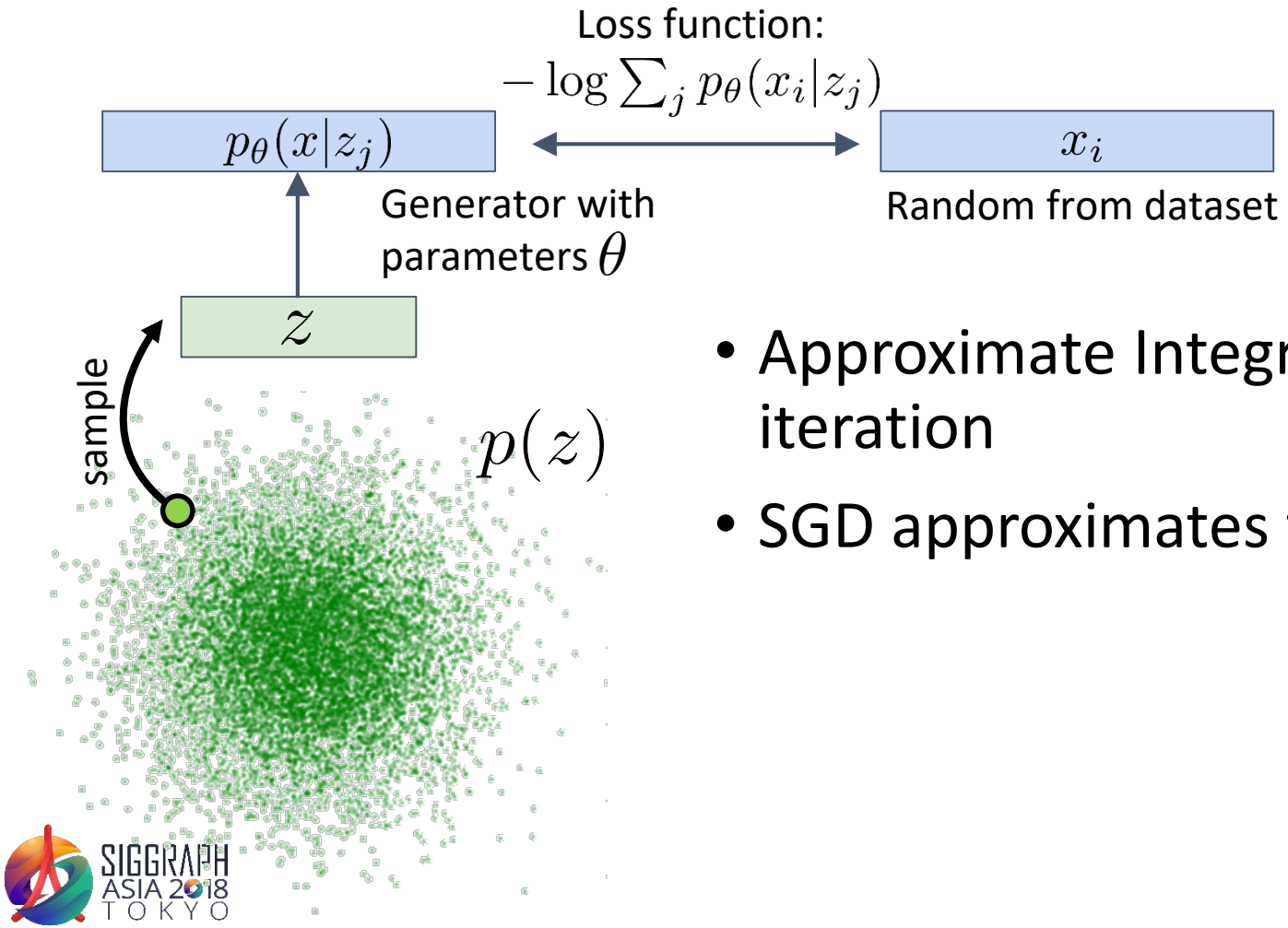
Maximum likelihood of data in generated distribution:

$$\theta^* = \arg\max_\theta \sum_{x_i \in \text{data}} \log \int p_\theta(x_i|z)\ p(z)\ dz$$

$$\theta^* \approx \arg\max_\theta \mathbb{E}_{x_i \sim p_{\text{data}}(x)} \log \mathbb{E}_{z \sim p(z)}\ p_\theta(x_i|z)$$

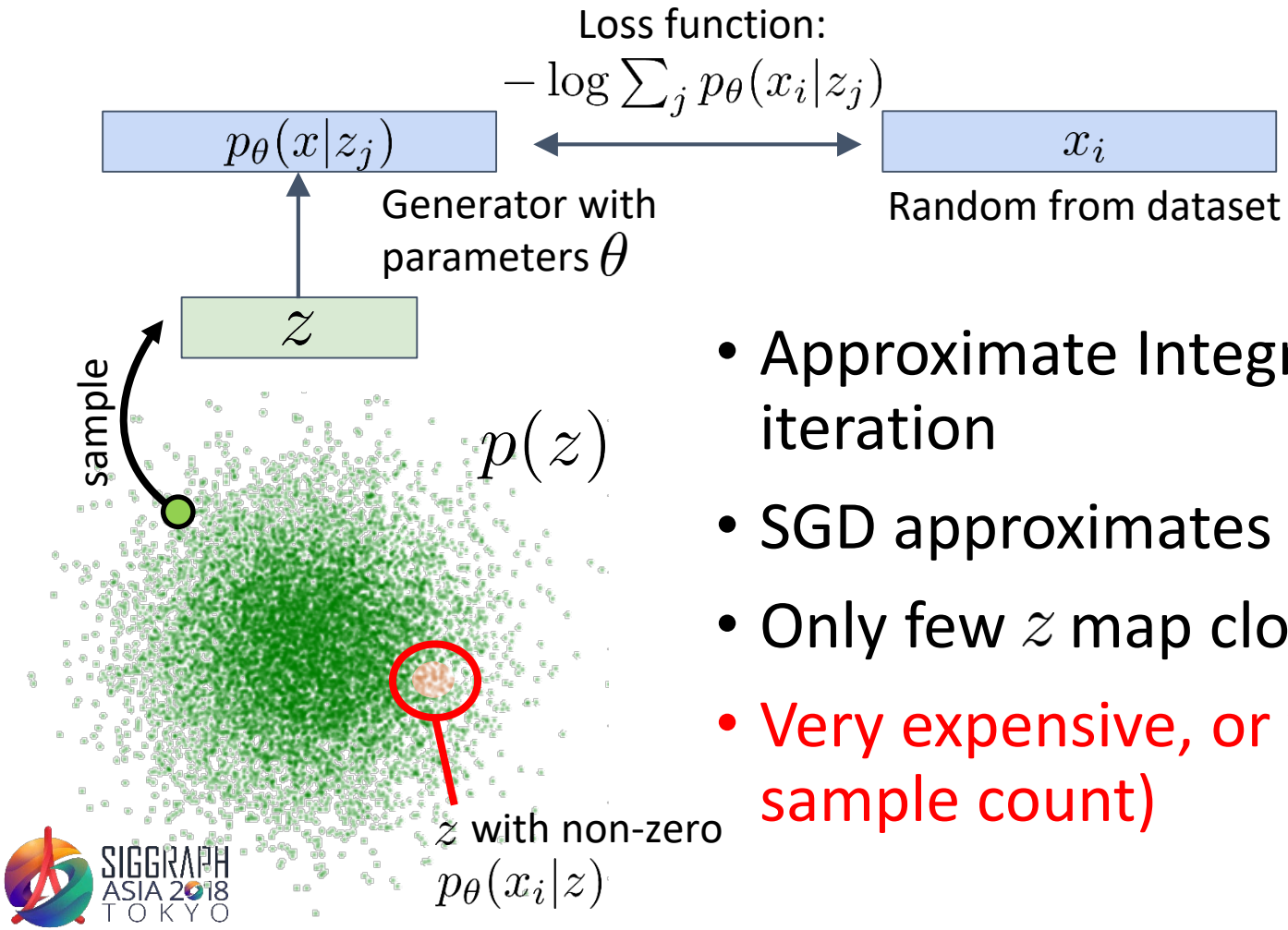
- Approximate Integral with Monte-Carlo in each iteration
- SGD approximates the sum over data

Variational Autoencoders (VAEs): Naïve Sampling (Monte-Carlo)



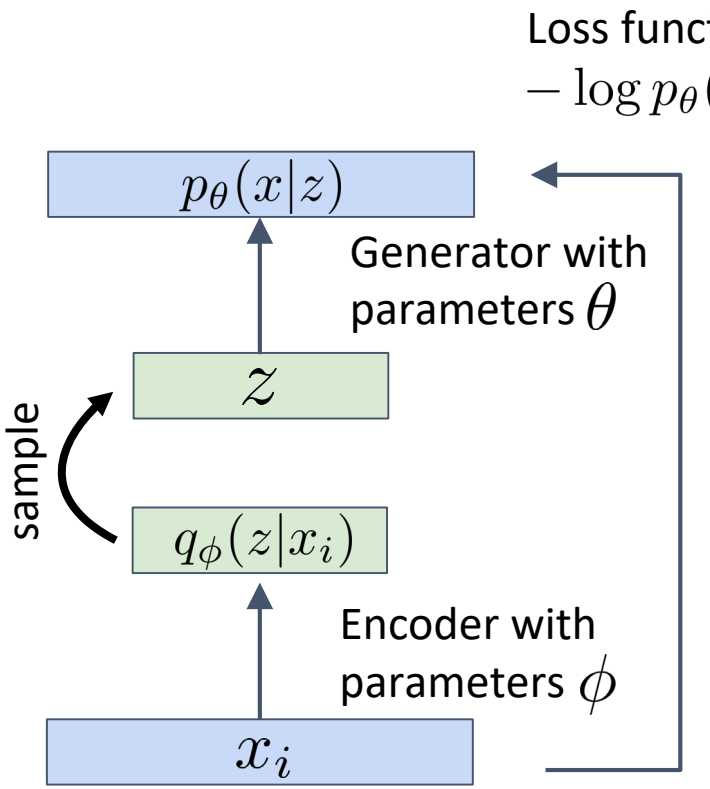
- Approximate Integral with Monte-Carlo in each iteration
- SGD approximates the expectancy over data

Variational Autoencoders (VAEs): Naïve Sampling (Monte-Carlo)



- Approximate Integral with Monte-Carlo in each iteration
- SGD approximates the expectancy over data
- Only few z map close to a given x_i
- Very expensive, or very inaccurate (depending on sample count)

Variational Autoencoders (VAEs): The Encoder



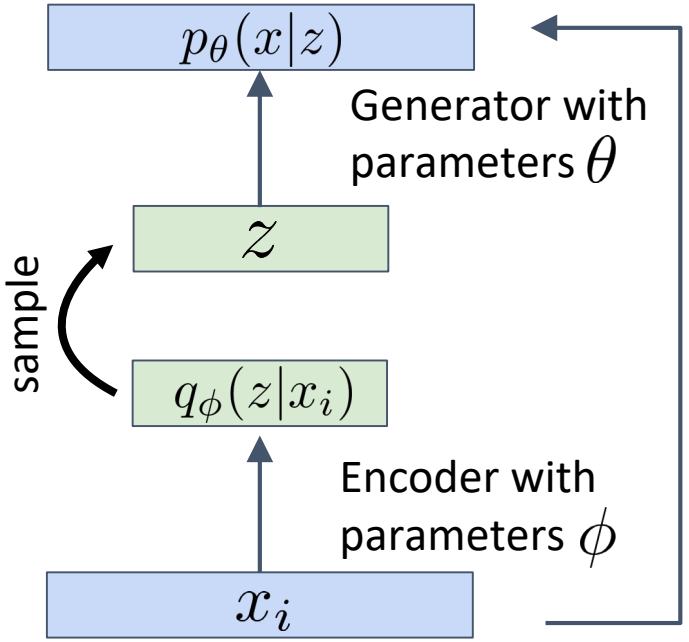
$$p_\theta(x) = \int p_\theta(x|z)\ p(z)\ dz$$

- During training, another network can learn a distribution of good z for a given x_i
- $q_\phi(z|x_i)$ should be much smaller than $p(z)$
- A single sample is good enough

Variational Autoencoders (VAEs): The Encoder

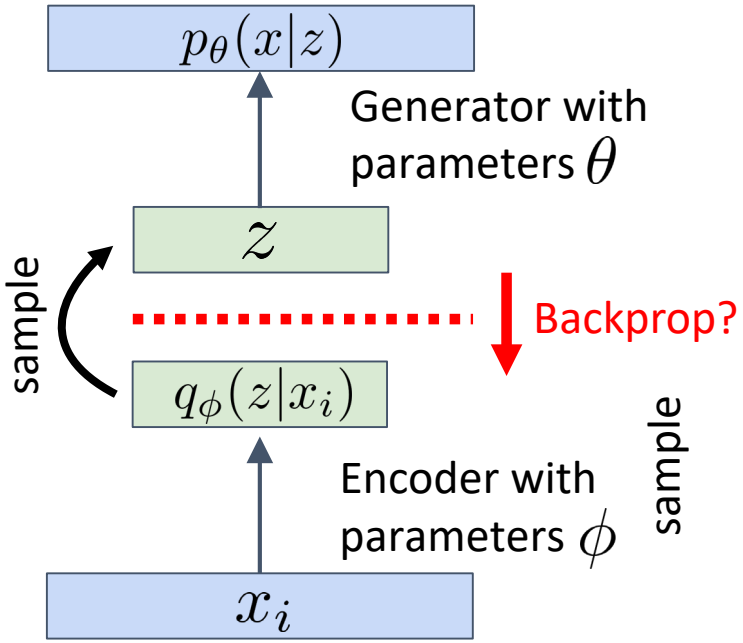
Loss function:

$$-\log p_\theta(x_i|z) + KL(\ q_\phi(z|x_i)\ \|\ p(z)\ )$$



- Can we still easily sample a new z?
- Need to make sure $q_\phi(z|x_i)$ approximates $p(z)$
- Regularize with KL-divergence
- Negative loss can be shown to be a lower bound for the likelihood, and equivalent if $q_\phi(z|x) = p_\theta(z|x)$

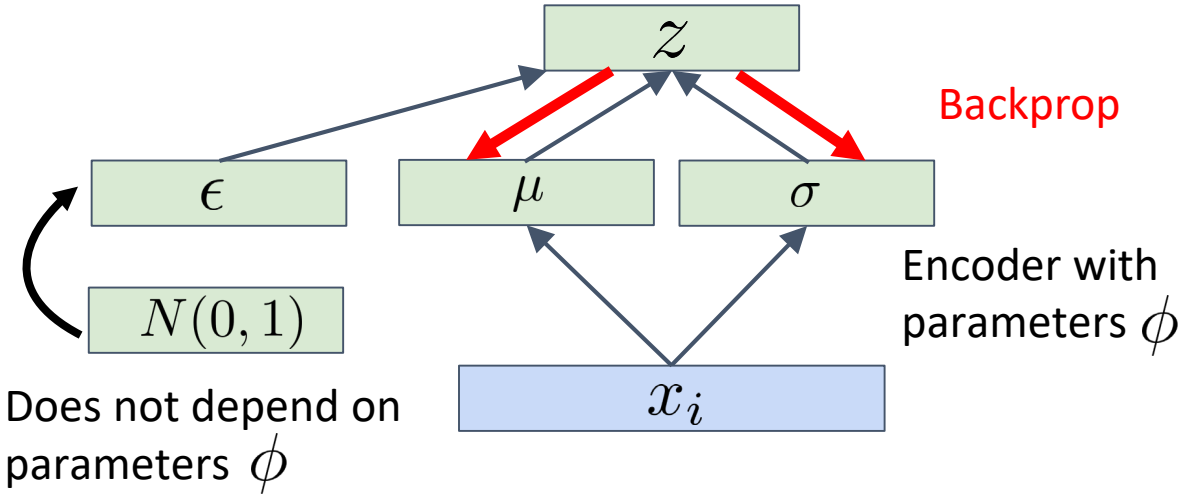
Reparameterization Trick



Example when $q_\phi(z|x_i) = N(z; \mu(x_i), \sigma(x_i))$:

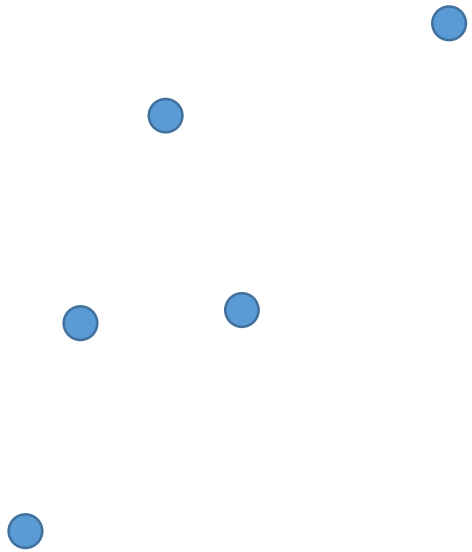
$z = \sigma + \mu \cdot \epsilon$, where $\epsilon \sim N(0,1)$

$$\frac{\partial z}{\partial \phi} = \frac{\partial \mu}{\partial \phi} + \frac{\partial \sigma}{\partial \phi} \cdot \epsilon$$

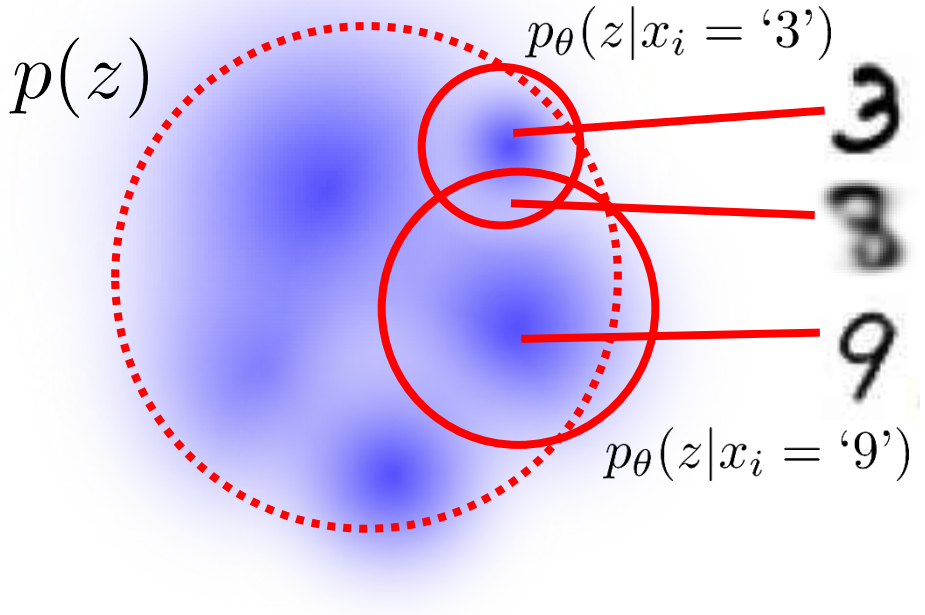


Feature Space of Autoencoders vs. VAEs

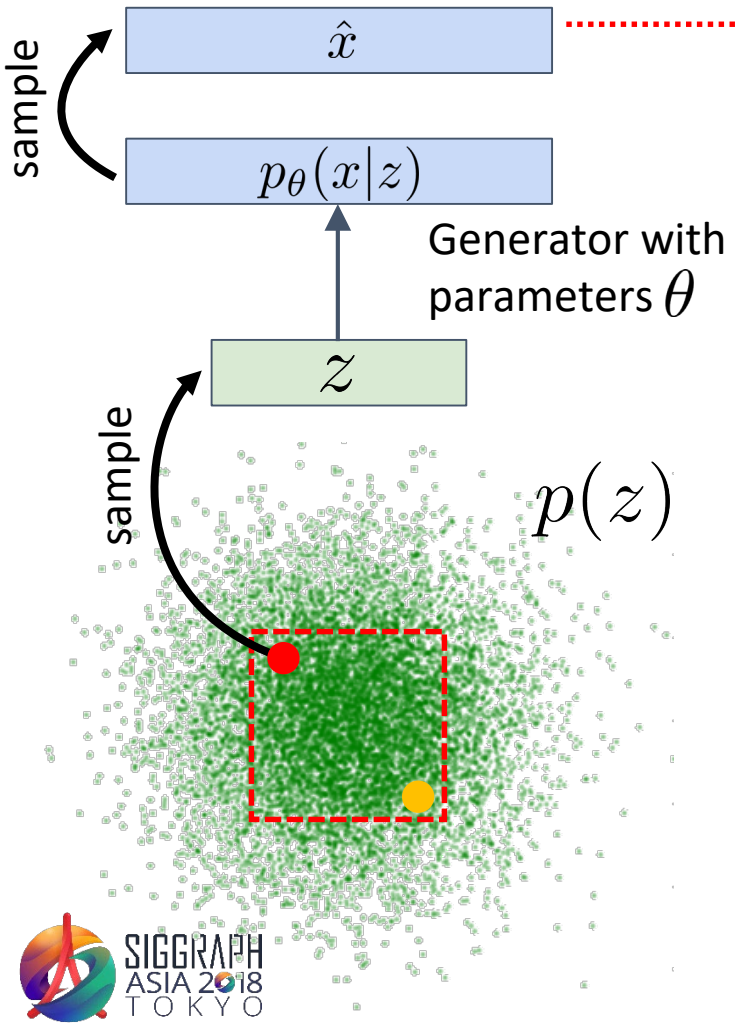
Autoencoder



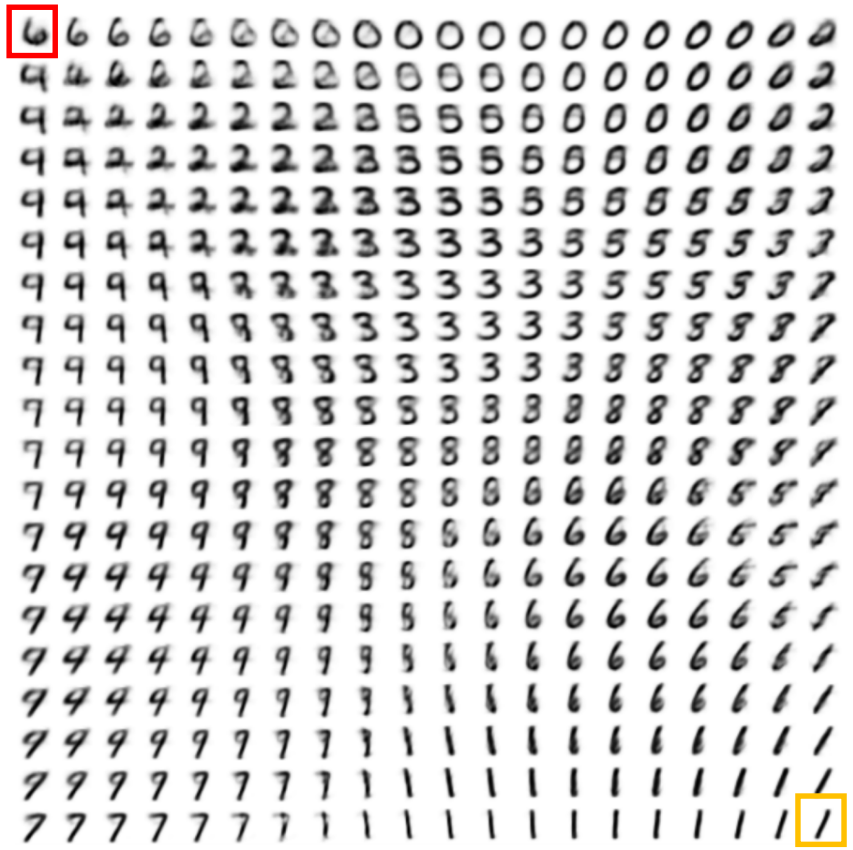
VAE



Generating Data



MNIST



Frey Faces

Image Credit: *Auto-Encoding Variational Bayes*, Kingma and Welling

Demos

VAE on MNIST

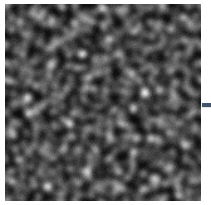
https://www.siarez.com/projects/variational-autoencoder

Code example

Variational Autoencoder

geometry.cs.ucl.ac.uk/creativeai

Generative Adversarial Networks



Player 1: generator
Scores if discriminator can't distinguish output from real image

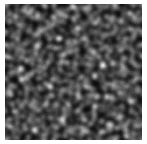
from dataset

Player 2: discriminator → real/fake
Scores if it can distinguish between real and fake

Generative Models

$p_\theta(x) \approx p_{\text{data}}(x)$

Generator with parameters θ



$p(z)$

known and easy to sample from

How to measure similarity of $p_\theta(x)$ and $p_{\text{data}}(x)$?

1) Likelihood of data in $p_\theta(x)$

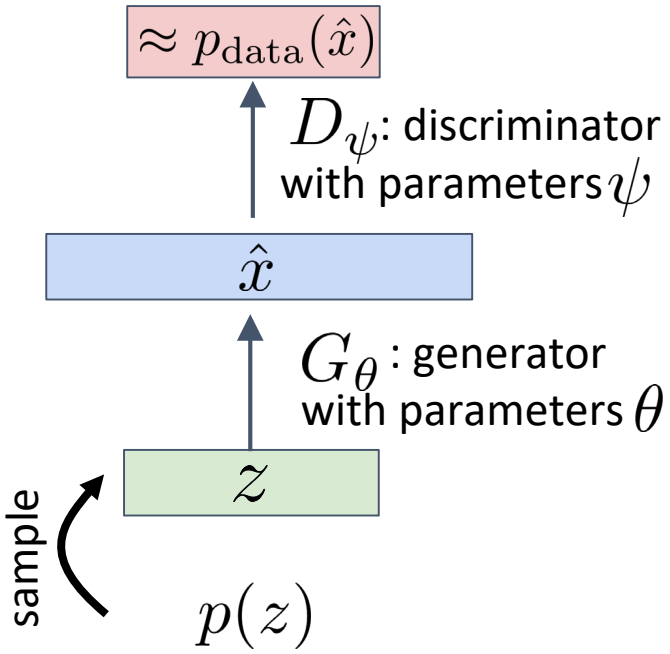
Variational Autoencoders (VAEs)

2) Adversarial game:

Discriminator distinguishes $p_\theta(x)$ and $p_{\text{data}}(x)$ vs *Generator* makes it hard to distinguish

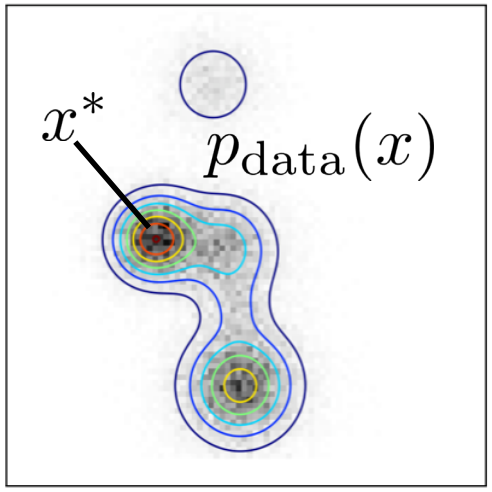
Generative Adversarial Networks (GANs)

Why Adversarial?



- If discriminator approximates $p_{\text{data}}(x)$:
- x^* at maximum of $p_{\text{data}}(x)$ has lowest loss
- Optimal $p_\theta(x)$ has single mode at x^*, small variance

$D_\psi \approx p_{\text{data}}(\hat{x})$



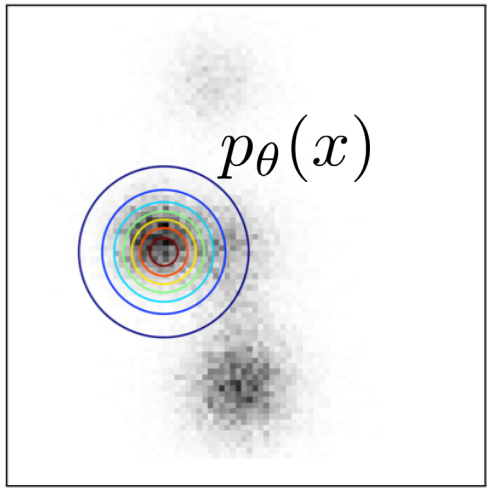
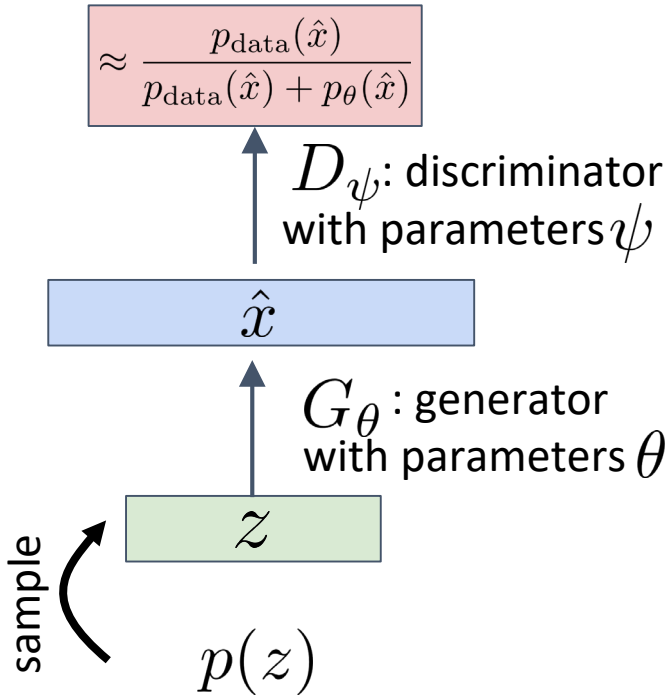


Image Credit: *How (not) to Train your Generative Model: Scheduled Sampling, Likelihood, Adversary?*, Ferenc Huszár

Why Adversarial?



- For GANs, the discriminator instead approximates:

$$\frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_\theta(x)}$$

→ depends on the generator

$D_\psi \approx p_{\text{data}}(\hat{x})$ $D_\psi \approx \frac{p_{\text{data}}(x)}{p_{\text{data}}(x)+p_\theta(x)}$

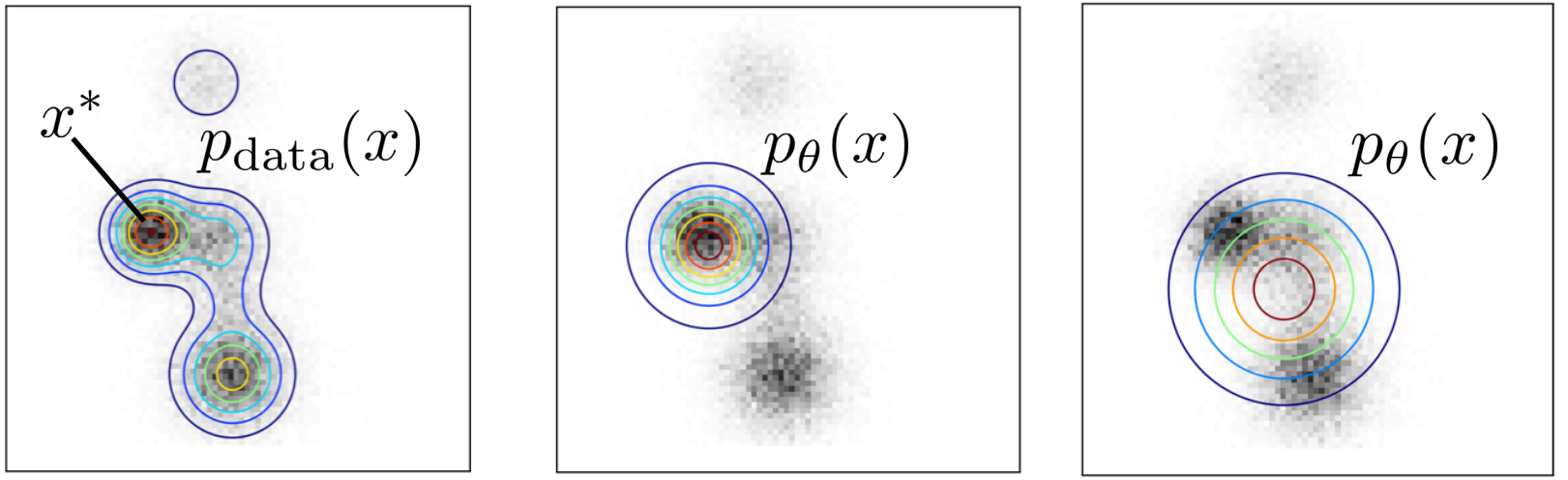
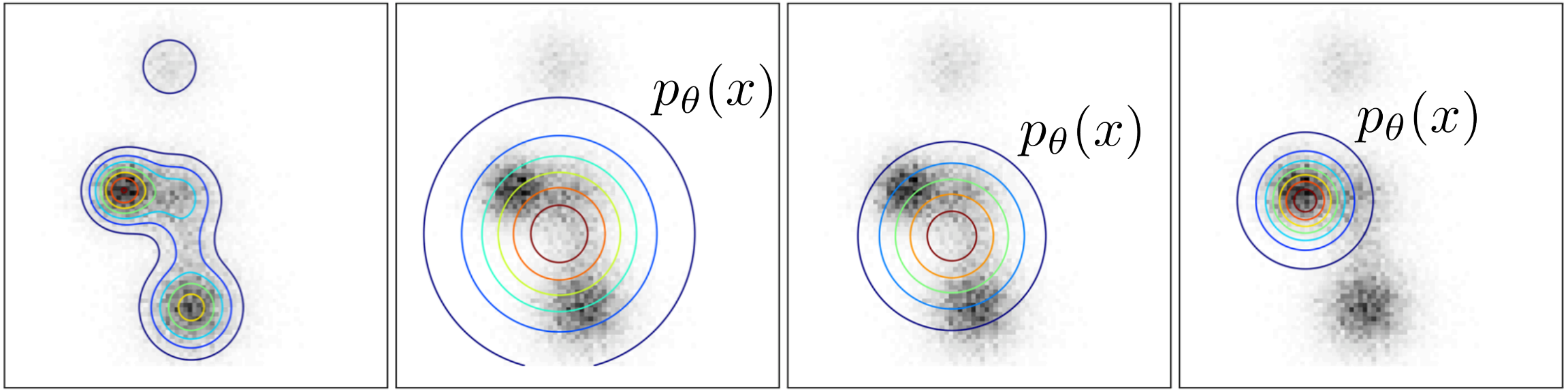


Image Credit: *How (not) to Train your Generative Model: Scheduled Sampling, Likelihood, Adversary?*, Ferenc Huszár

Why Adversarial?



$p_{\text{data}}(x)$

VAEs:
Maximize likelihood of **data samples** in $p_\theta(x)$

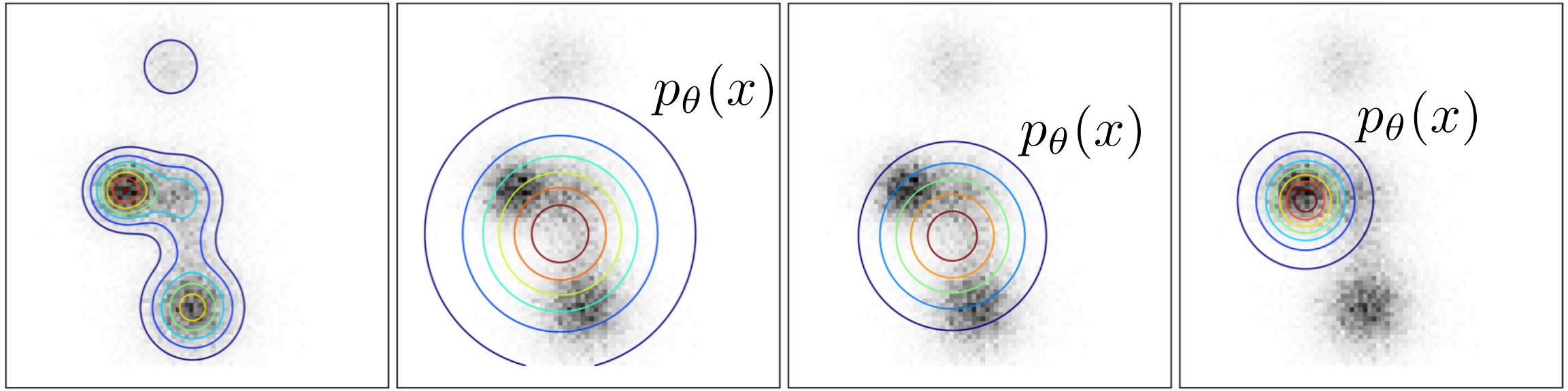
GANs:
Adversarial game

Maximize likelihood of **generator samples** in approximate $p_{\text{data}}(x)$

Image Credit: *How (not) to Train your Generative Model: Scheduled Sampling, Likelihood, Adversary?*, Ferenc Huszár

Why Adversarial?

$\approx KL(p_{\text{data}} \parallel p_\theta)$ $\approx JS(p_{\text{data}} \parallel p_\theta)$ $\approx KL(p_\theta \parallel p_{\text{data}})$



$p_{\text{data}}(x)$

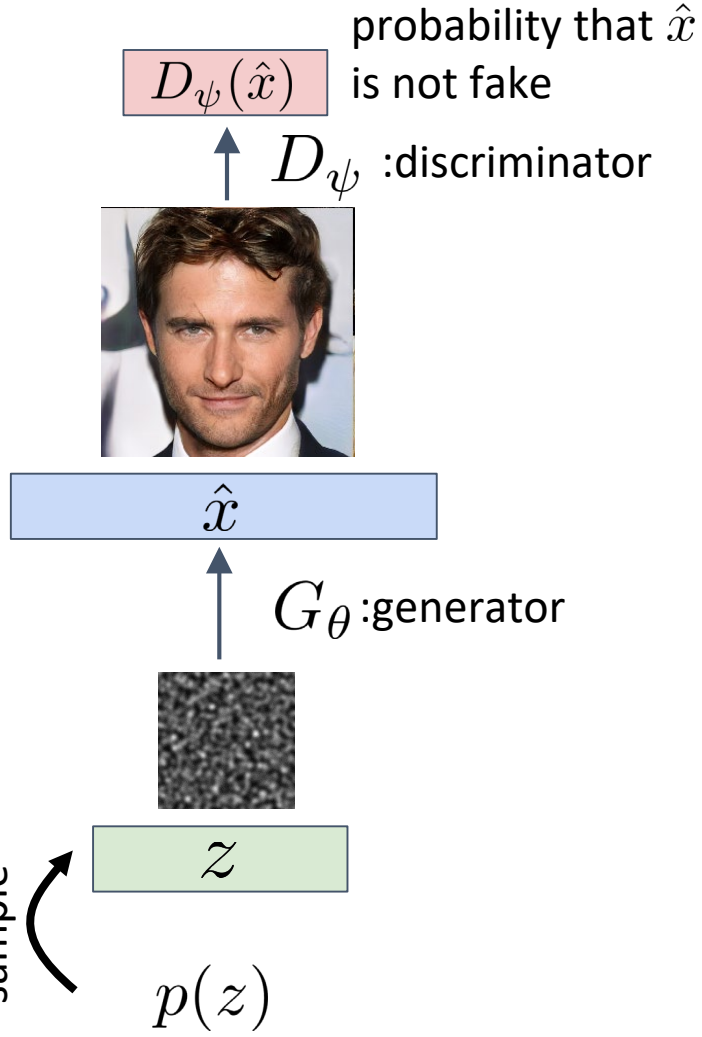
VAEs:
Maximize likelihood of **data samples** in $p_\theta(x)$

GANs:
Adversarial game

Maximize likelihood of **generator samples** in approximate $p_{\text{data}}(x)$

Image Credit: *How (not) to Train your Generative Model: Scheduled Sampling, Likelihood, Adversary?*, Ferenc Huszár

GAN Objective



fake/real classification loss (BCE):

$$L(\theta, \psi) = -0.5\ \mathbb{E}_{x \sim p_{\text{data}}}\ \log D_\psi(x) - 0.5\ \mathbb{E}_{x \sim p_\theta}\ \log(1 - D_\psi(x))$$

Discriminator objective:

$$\min_\psi L(\theta, \psi)$$

Generator objective:

$$\max_\theta L(\theta, \psi)$$

Non-saturating Heuristic

$$L(\theta, \psi) = -0.5\ \mathbb{E}_{x \sim p_{\text{data}}}\ \log D_\psi(x) \\ -0.5\ \mathbb{E}_{x \sim p_\theta}\ \log(1 - D_\psi(x))$$

Generator loss is negative binary cross-entropy:

$$L_G(\theta, \psi) = 0.5\ \mathbb{E}_{x \sim p_\theta}\ \log(1 - D_\psi(x))$$ poor convergence

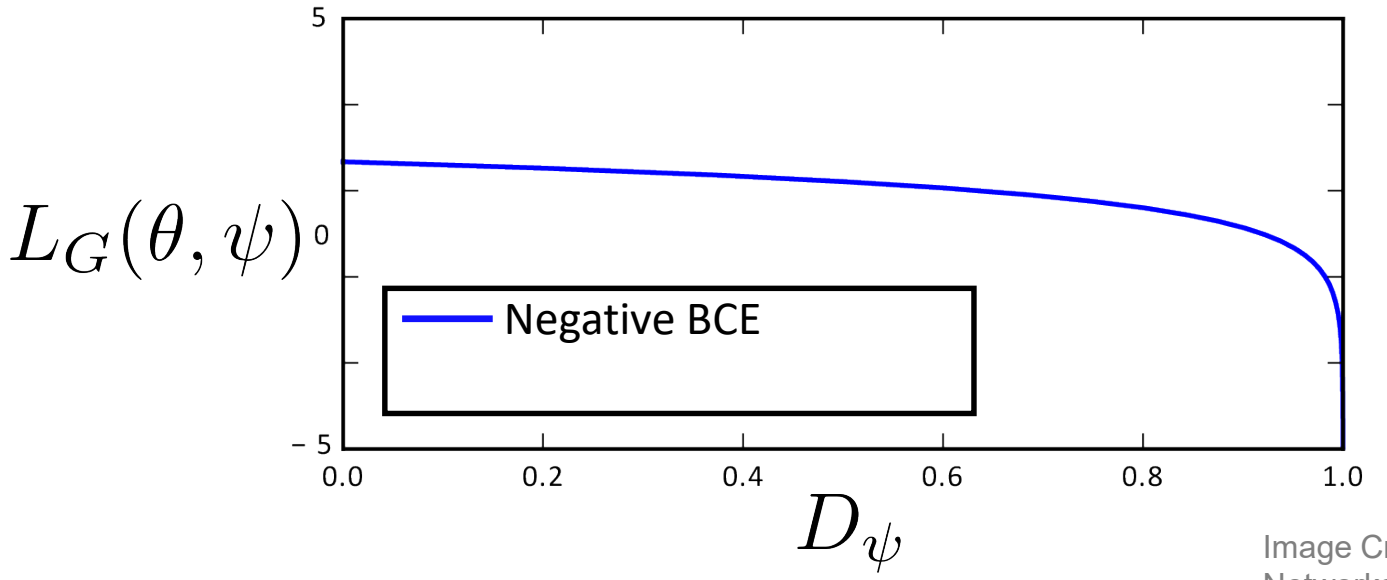


Image Credit: NIPS 2016 Tutorial: Generative Adversarial Networks, Ian Goodfellow

Non-saturating Heuristic

Generator loss is negative binary cross-entropy:

$$L_G(\theta, \psi) = 0.5\ \mathbb{E}_{x \sim p_\theta}\ \log(1 - D_\psi(x))$$ poor convergence

Flip target class instead of flipping the sign for generator loss:

$$L_G(\theta, \psi) = -0.5\ \mathbb{E}_{x \sim p_\theta}\ \log D_\psi(x)$$ good convergence – like BCE

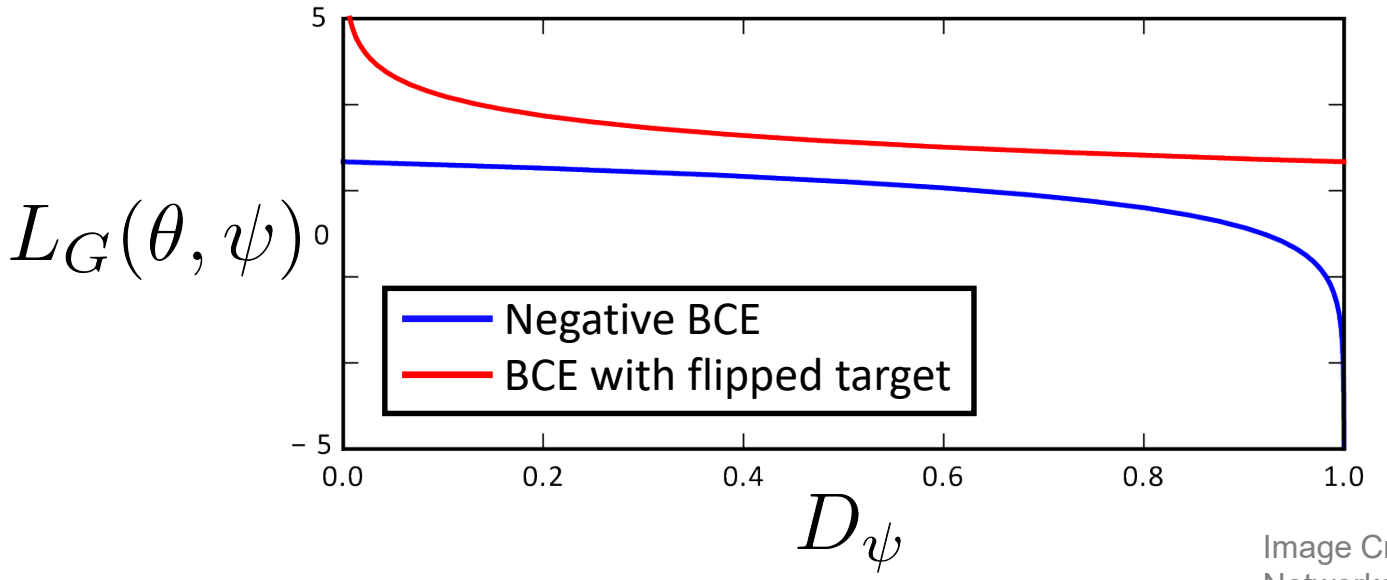


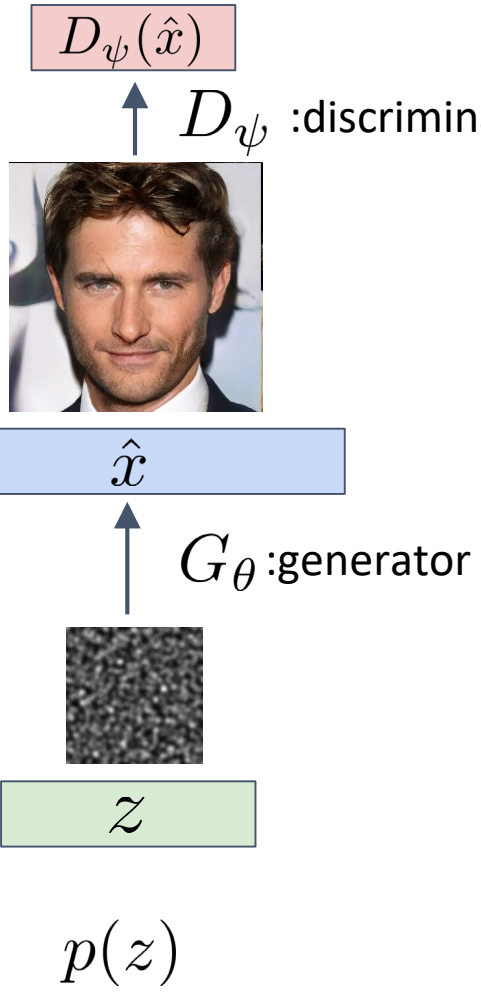
Image Credit: NIPS 2016 Tutorial: Generative Adversarial Networks, Ian Goodfellow

GAN Training

Generator training

Loss:

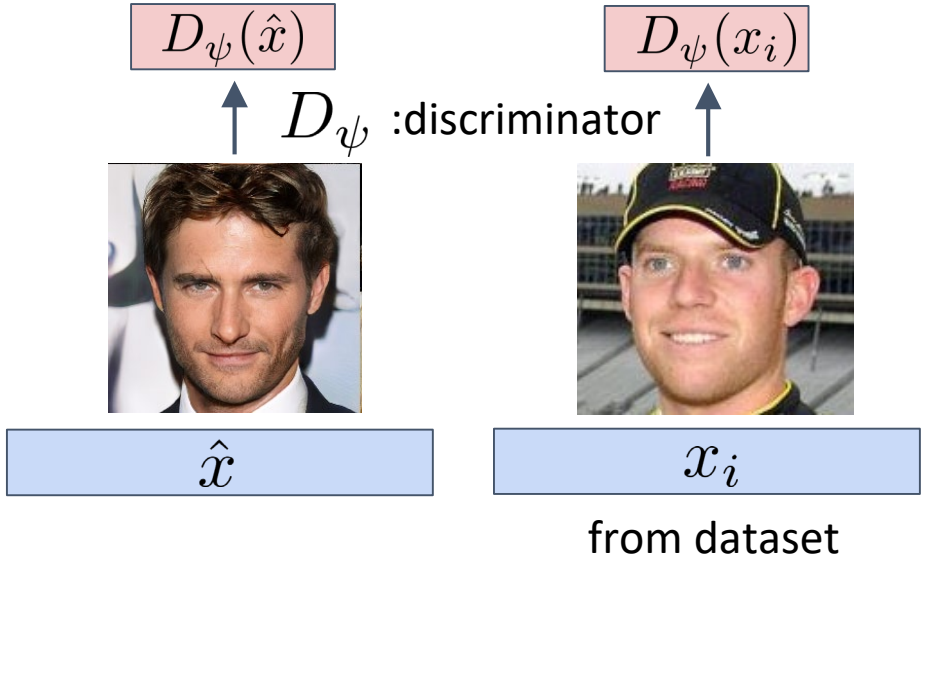
$$L_G(\theta, \psi) = -\log D_\psi(\hat{x})$$



Discriminator training

Loss:

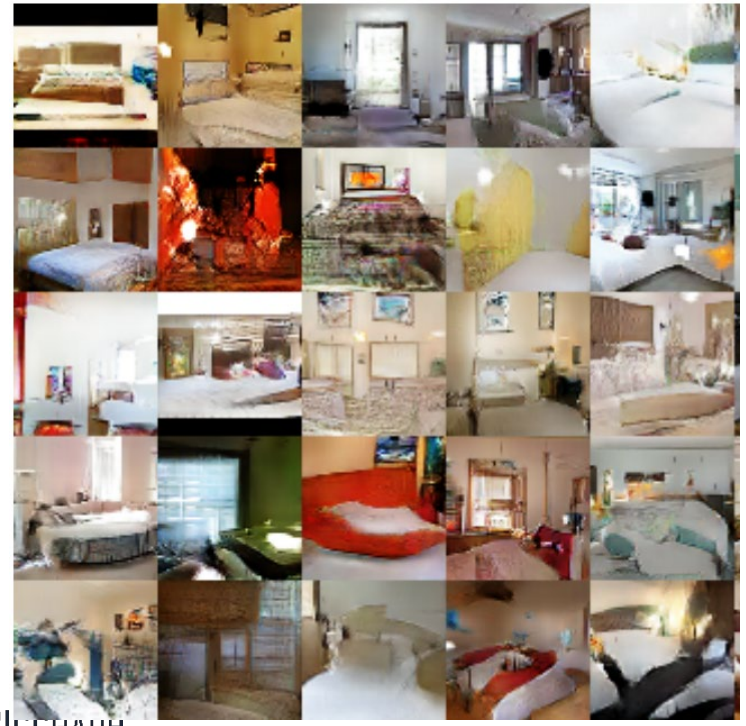
$$L_D(\theta, \psi) = -0.5 \log(1 - D_\psi(\hat{x})) - 0.5 \log D_\psi(x_i)$$



Interleave in each training step

DCGAN

- First paper to successfully use CNNs with GANs
- Due to using novel components (at that time) like batch norm., ReLUs, etc.



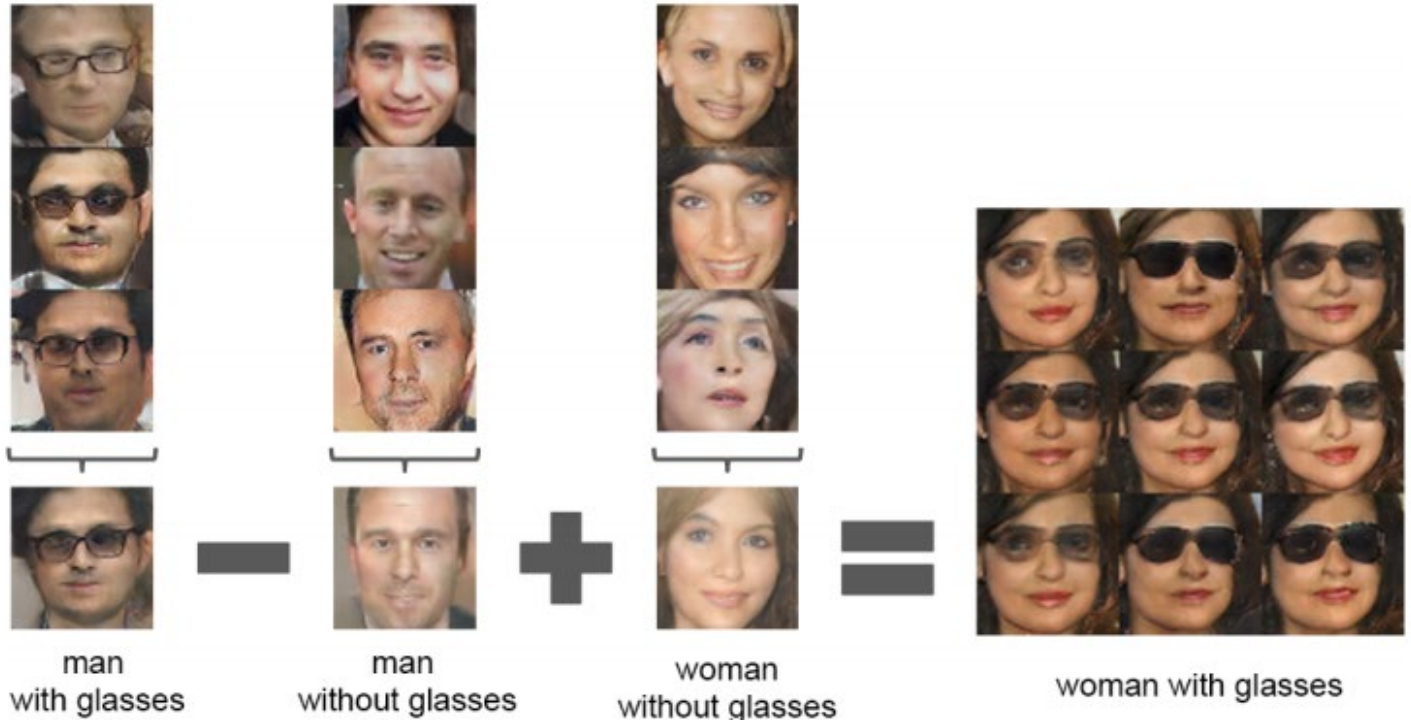


Image Credit: *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*, Radford et al.

Code example

Generative Adversarial Network

geometry.cs.ucl.ac.uk/creativeai

Conditional GANs (CGANs)

- ≈ learn a mapping between images from example pairs
- Approximate sampling from a conditional distribution $p_{\text{data}}(x \mid c)$

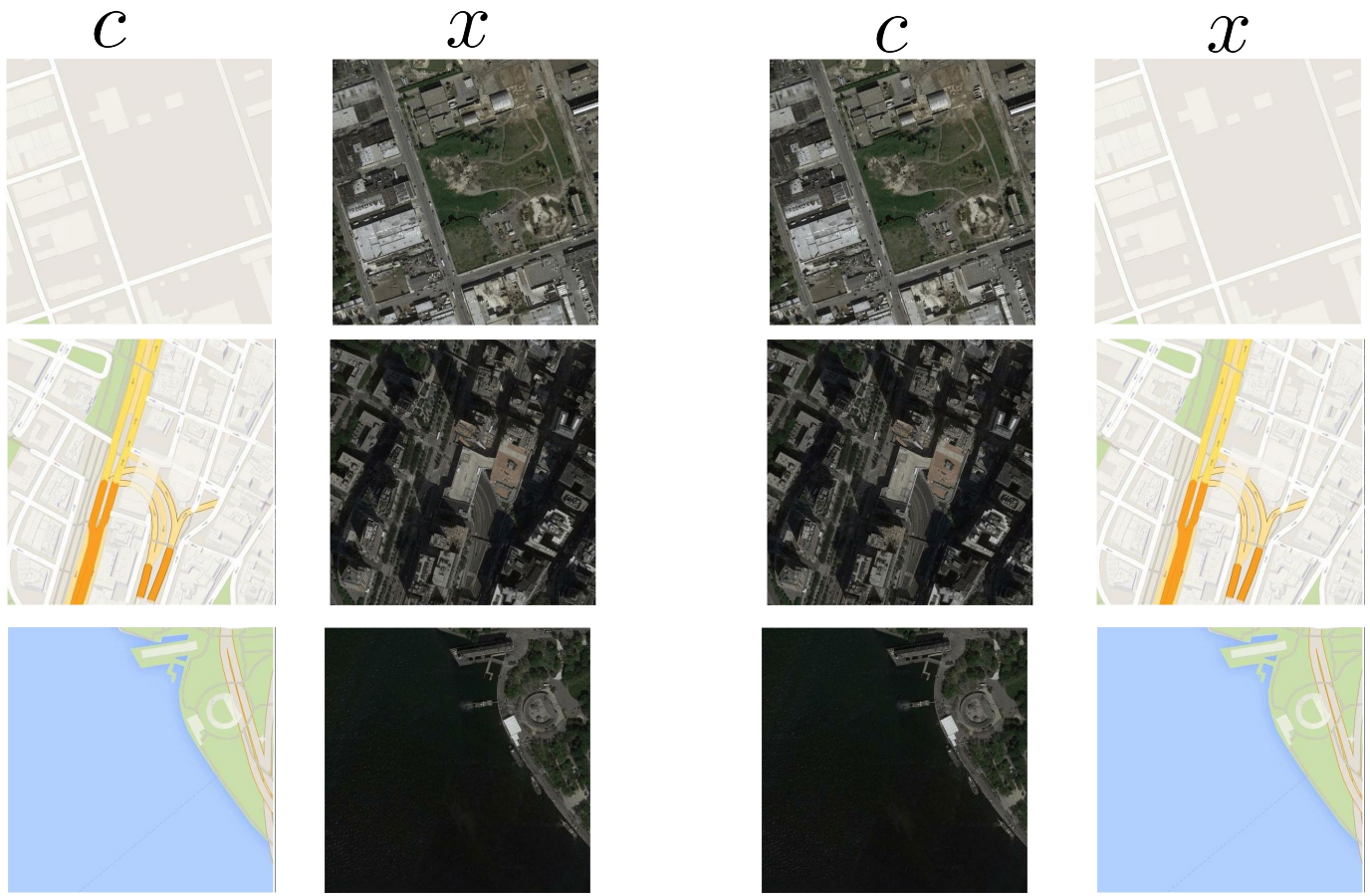


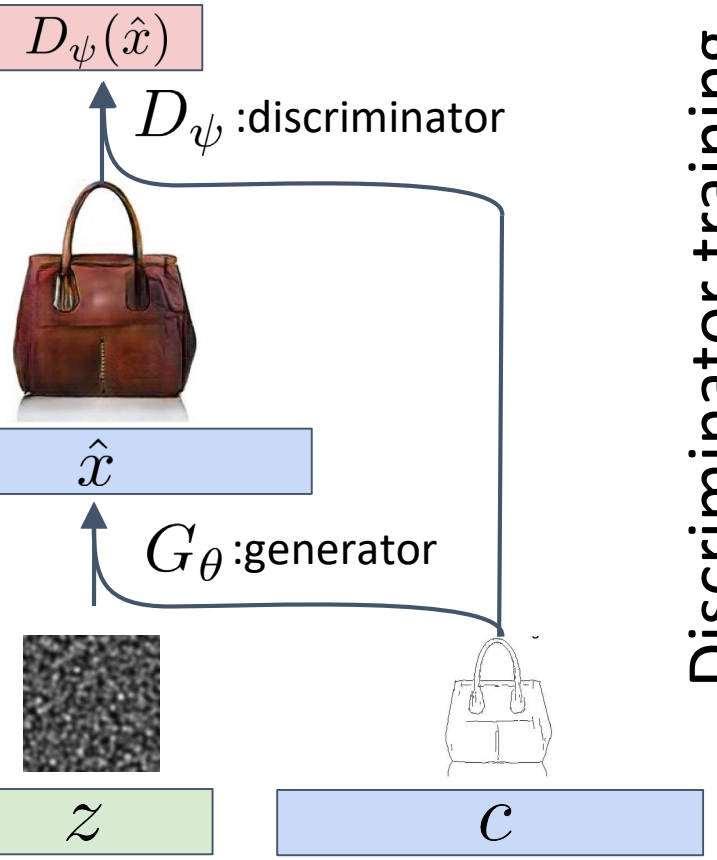
Image Credit: *Image-to-Image Translation with Conditional Adversarial Nets*, Isola et al.

Conditional GANs

Loss:

$$L_G(\theta, \psi) = -\log D_\psi(\hat{x}, c)$$

Generator training



Loss:

$$L_D(\theta, \psi) = -0.5 \log(1 - D_\psi(\hat{x}, c)) - 0.5 \log D_\psi(x_i, c)$$

Discriminator training

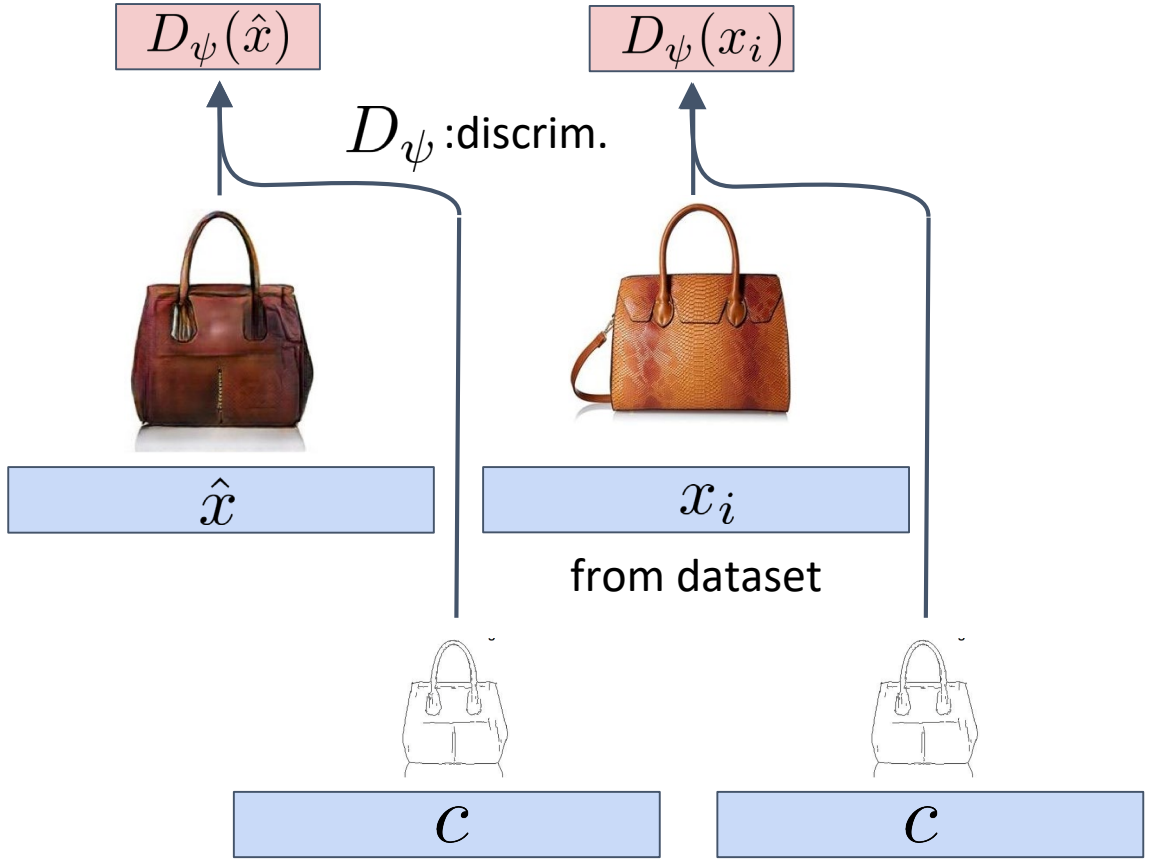


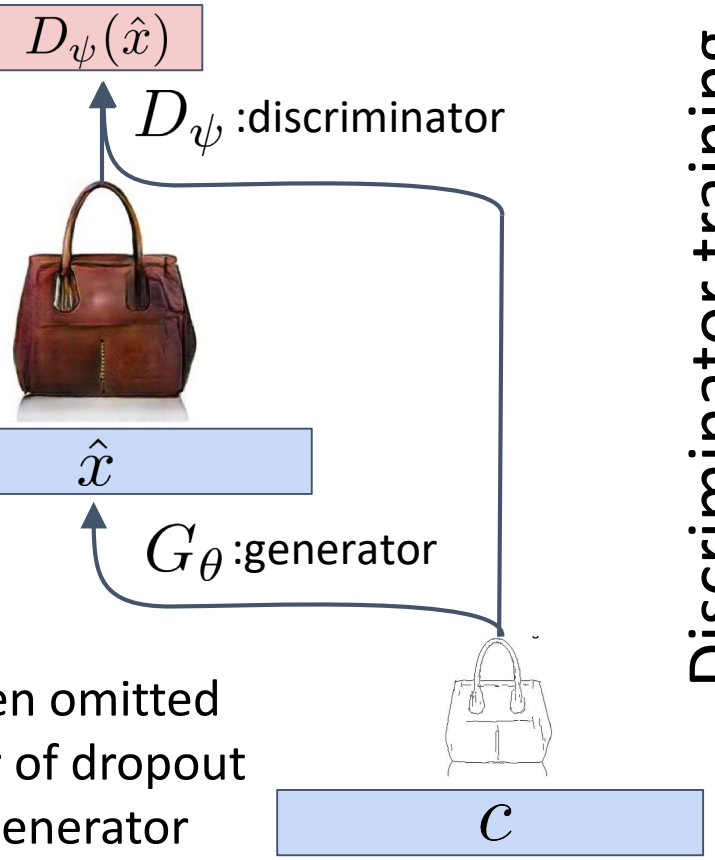
Image Credit: *Image-to-Image Translation with Conditional Adversarial Nets*, Isola et al.

Conditional GANs: Low Variation per Condition

Loss:

$$L_G(\theta, \psi) = -\log D_\psi(\hat{x}, c)$$

Generator training



z is often omitted in favor of dropout in the generator

Loss:

$$L_D(\theta, \psi) = -0.5 \log(1 - D_\psi(\hat{x}, c)) - 0.5 \log D_\psi(x_i, c)$$

Discriminator training

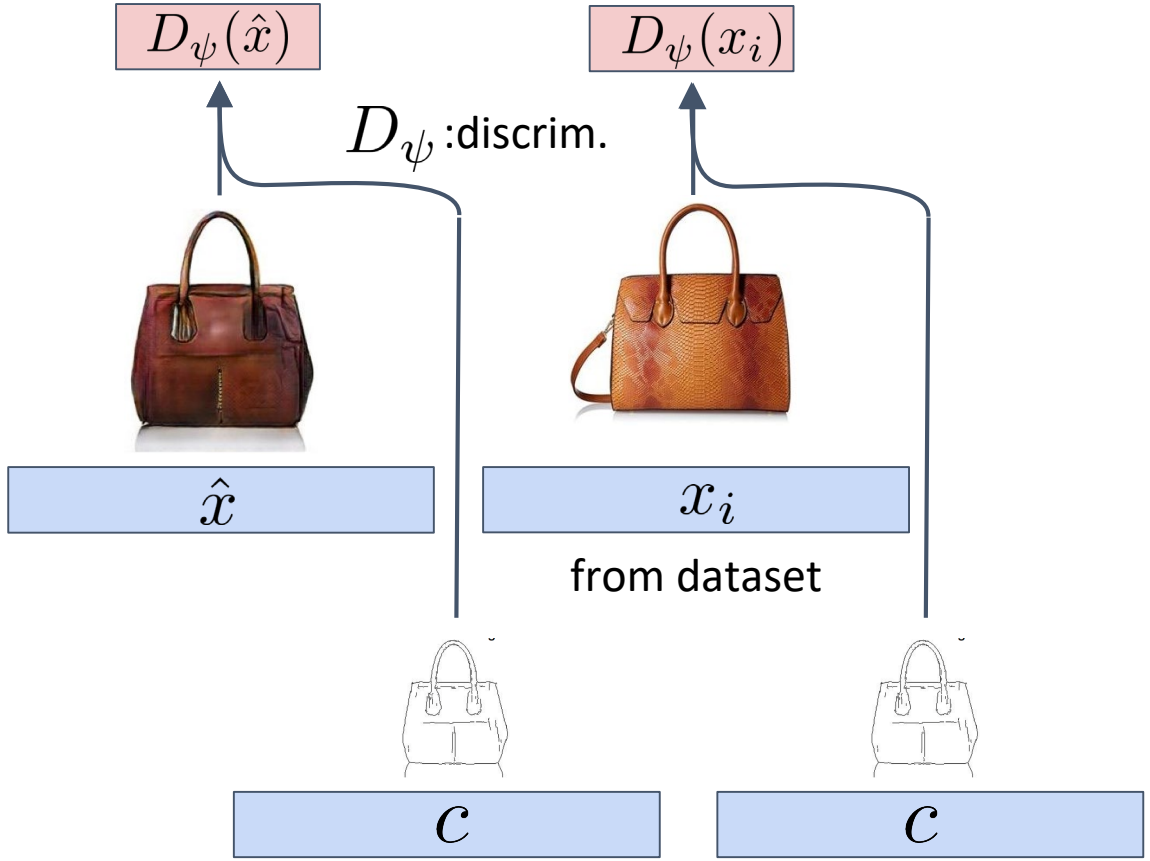


Image Credit: *Image-to-Image Translation with Conditional Adversarial Nets*, Isola et al.

Demos

CGAN

https://affinelayer.com/pixsrv/index.html

Unstable Training

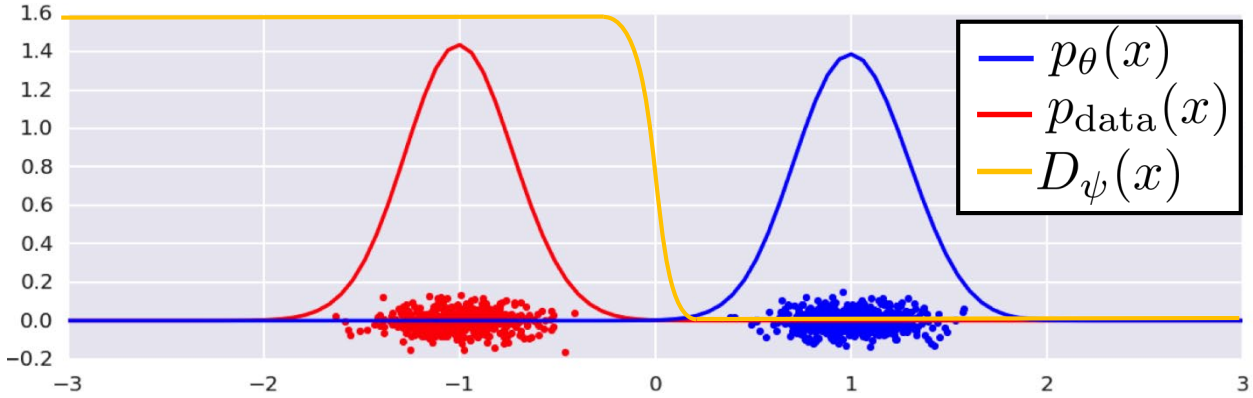
GAN training can be unstable

Three current research problems (may be related):

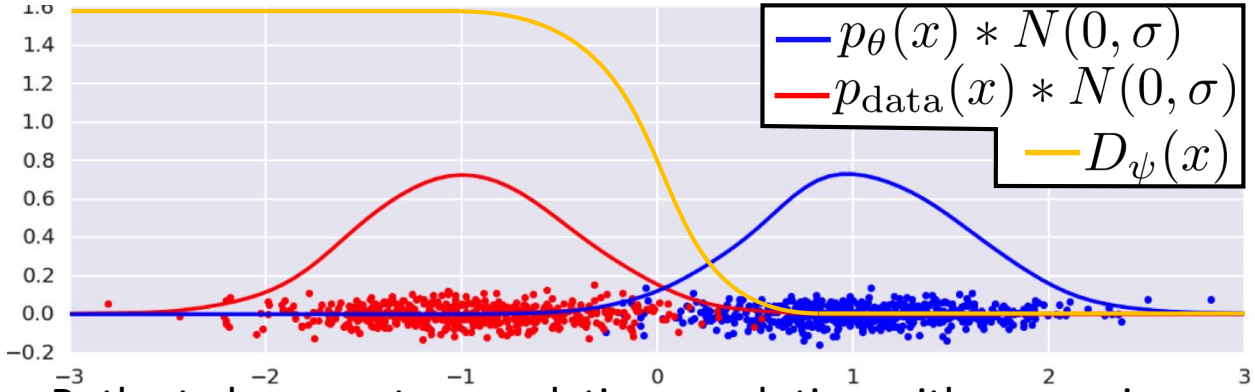
- Reaching a Nash equilibrium (the gradient for both L_G and L_D is 0)
- p_θ and p_{data} initially don't overlap
- Mode Collapse

Generator and Data Distribution Don’t Overlap

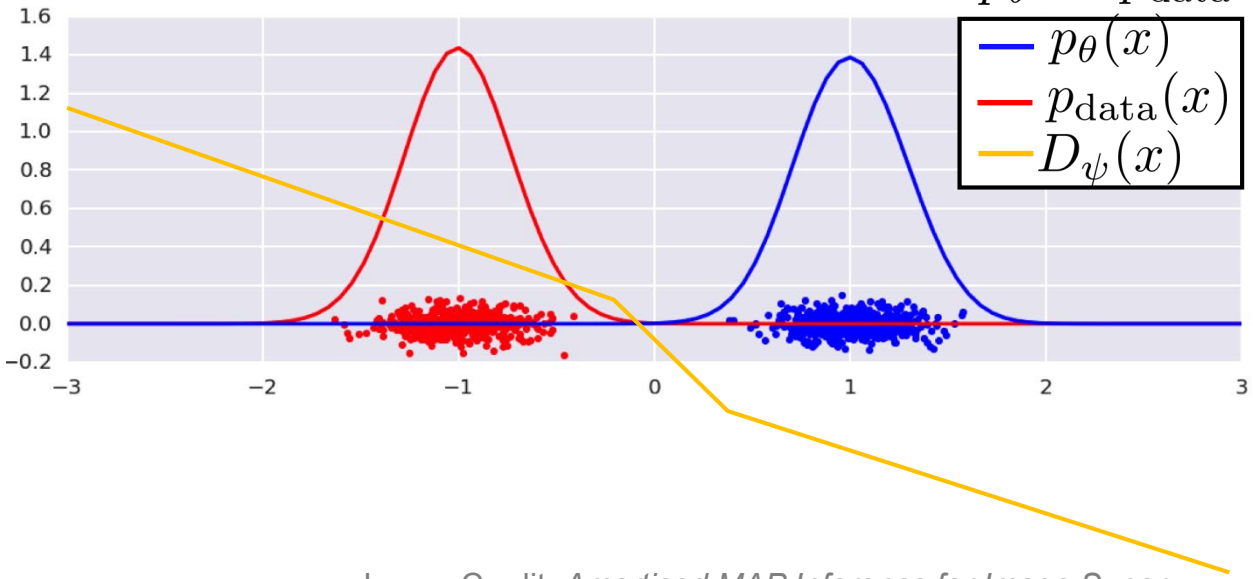
Standard



Instance noise: adding noise to generated and real images



Wasserstein GANs: EMD as distance between p_θ and p_{data}



Roth et al. suggest an analytic convolution with a gaussian:

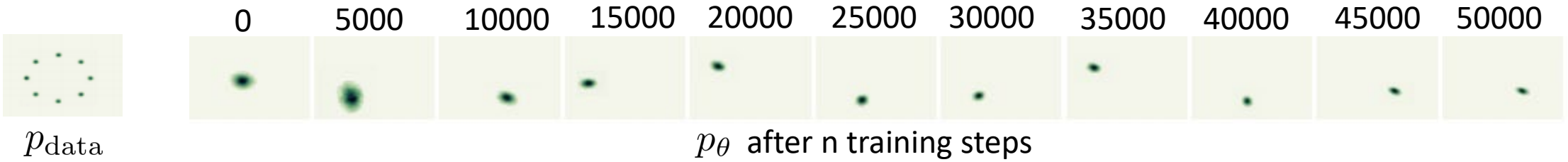
Stabilizing Training of Generative Adversarial Networks through Regularization, Roth et al. 2017

Image Credit: *Amortised MAP Inference for Image Super-resolution*, Sønderby et al.

Mode Collapse

Optimal $D_\psi(x)$: $\dfrac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_\theta(x)}$

p_θ only covers one or a few modes of p_{data}



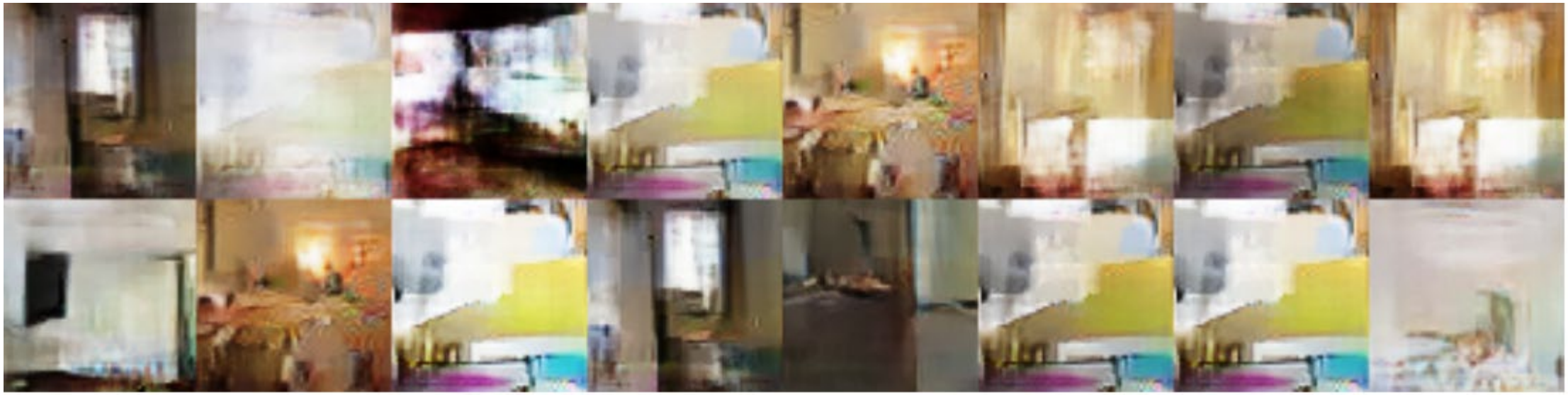


Image Credit: *Wasserstein GAN,* Arjovsky et al.
Unrolled Generative Adversarial Networks, Metz et al.

Mode Collapse

Solution attempts:

- Minibatch comparisons: Discriminator can compare instances in a minibatch (*Improved Techniques for Training GANs*, Salimans et al.)
- Unrolled GANs: Take k steps with the discriminator in each iteration, and backpropagate through all of them to update the generator

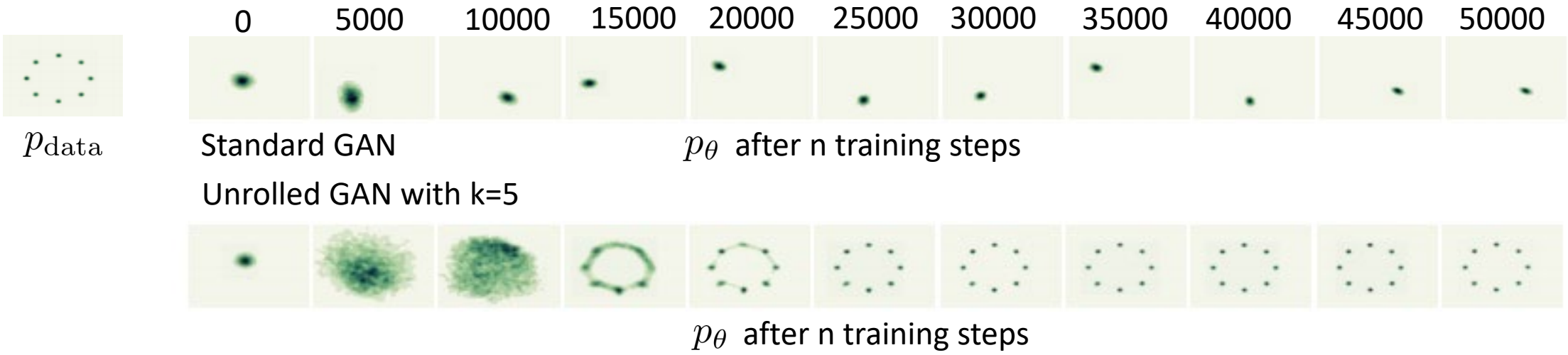
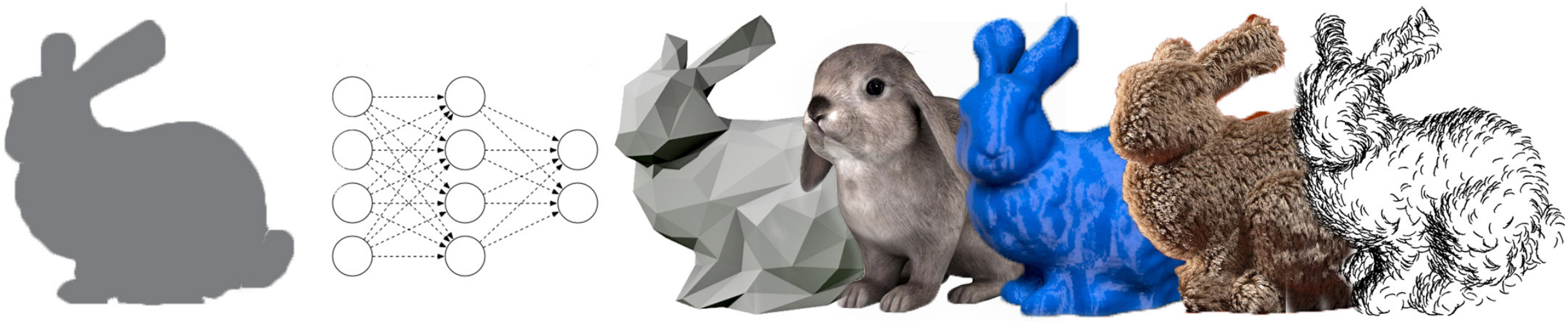


Image Credit: *Wasserstein GAN,* Arjovsky et al.
Unrolled Generative Adversarial Networks, Metz et al.

Summary

- Autoencoders
  - Can infer useful latent representation for a dataset
  - Bad generators
- VAEs
  - Can infer a useful latent representation for a dataset
  - Better generators due to latent space regularization
  - Lower quality reconstructions and generated samples (usually blurry)
- GANs
  - Can not find a latent representation for a given sample (no encoder)
  - Usually better generators than VAEs
  - Currently unstable training (active research)

Course Information (slides/code/comments)



http://geometry.cs.ucl.ac.uk/creativeai/

SIGGRAPH ASIA 2018 TOKYO

CreativeAI: Deep Learning for Graphics

Image Domains

Niloy Mitra
UCL

Iasonas Kokkinos
UCL

Paul Guerrero
UCL

Nils Thuerey
TUM

Tobias Ritschel
UCL

Timetable

			Niloy	Paul	Nils
Theory and Basics	Introduction	2:15 pm	X	X	X
	Machine Learning Basics	~ 2:25 pm	X		
	Neural Network Basics	~ 2:55 pm			X
	Feature Visualization	~ 3:25 pm		X	
	Alternatives to Direct Supervision	~ 3:35 pm		X	
	15 min. break				
State of the Art	Image Domains	4:15 pm		X	
	3D Domains	~ 4:45 pm	X		
	Motion and Physics	~ 5:15 pm			X
	Discussion	~ 5:45 pm	X	X	X

Overview

Examples of deep learning techniques that are commonly used in the image domain:

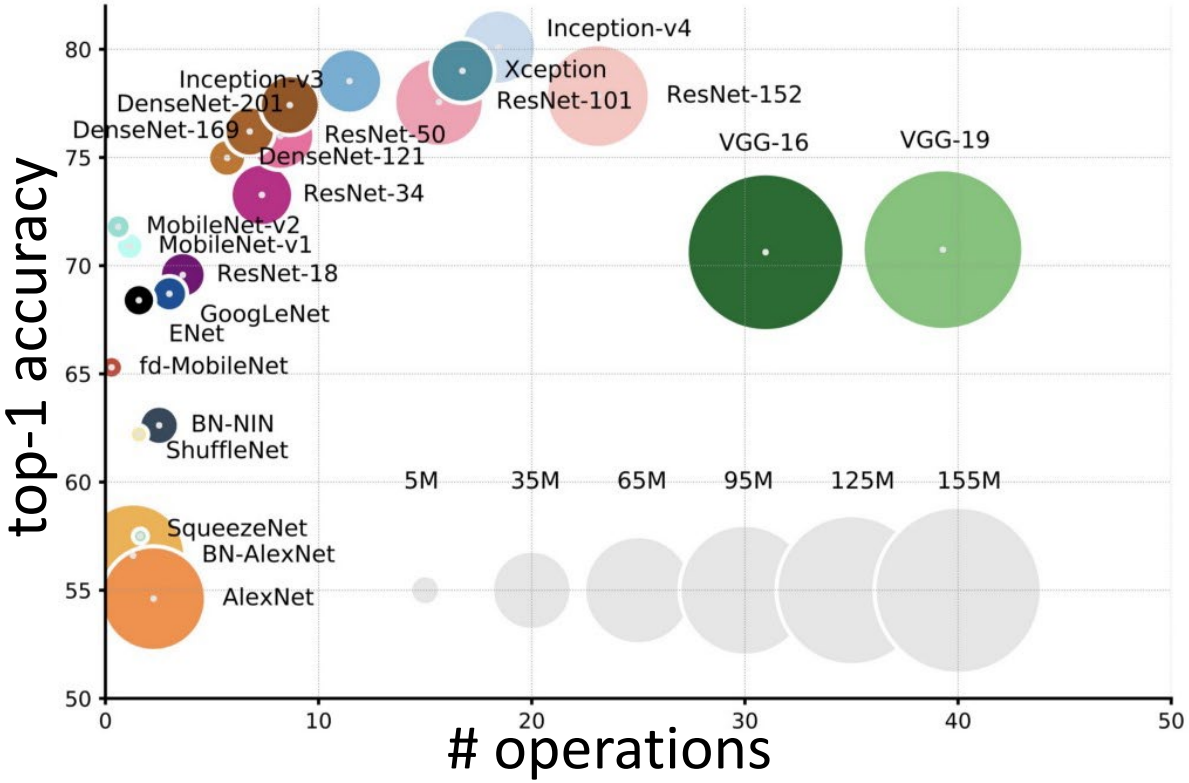
- Common Architecture Elements
  (Dilated Convolution, Grouped Convolutions)
- Deep Features
  (Autoencoders, Transfer Learning, One-shot Learning, Style Transfer)
- Adversarial Image Generation
  (GANs, CGANs)
- Interesting Trends
  (Attention, “Gray Box” Learning)

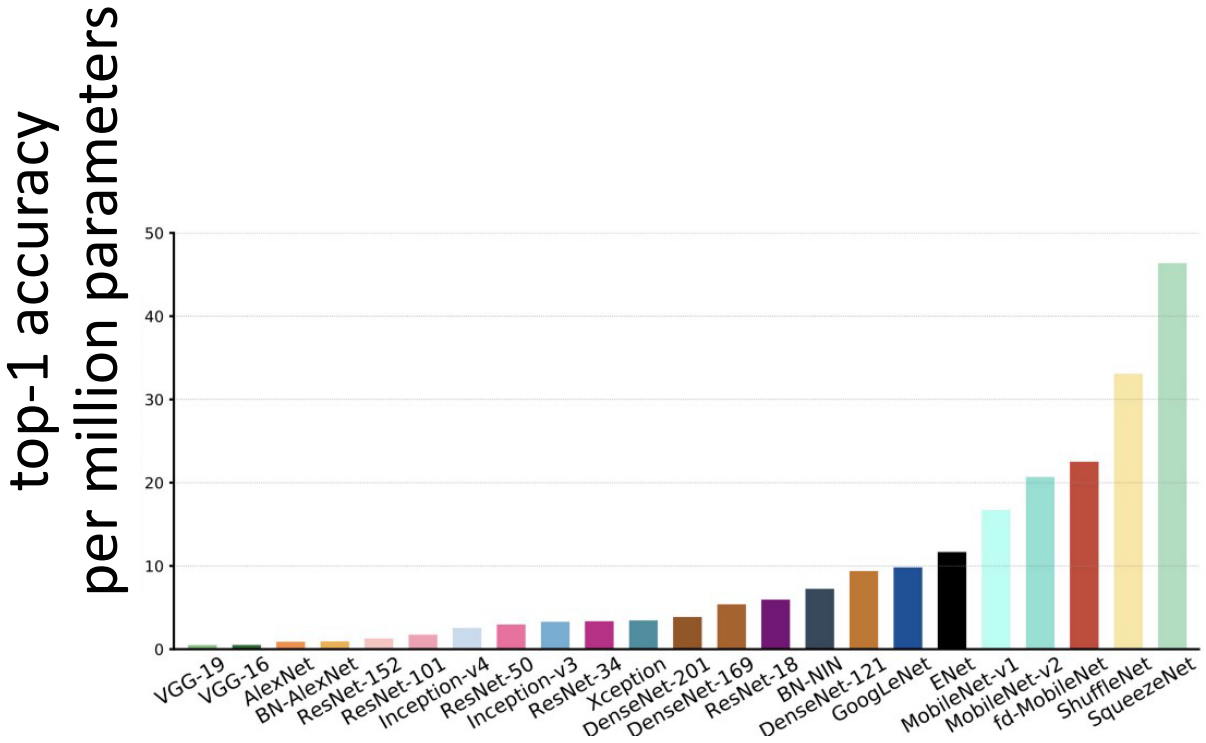
Common Architecture Elements

Classification, Segmentation, Detection

ImageNet classification performance

(for up-to-date top-performers see leaderboards of datasets like ImageNet or COCO)





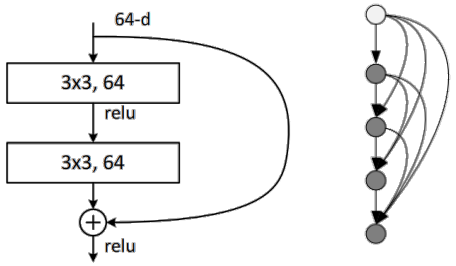
Images from: Canziani et al., *An Analysis of Deep Neural Network Models for Practical Applications*, arXiv 2017

Blog: https://towardsdatascience.com/neural-network-architectures-156e5bad51ba

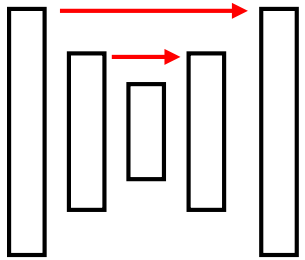
Architecture Elements

Some notable architecture elements shared by many successful architectures:

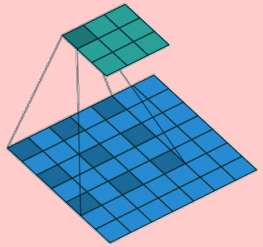
Residual Blocks
and Dense Blocks



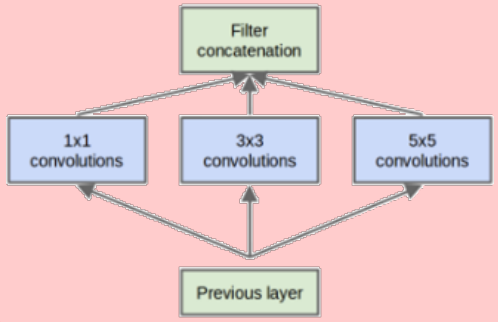
Skip Connections
(UNet)



Dilated
Convolutions



Grouped
Convolutions



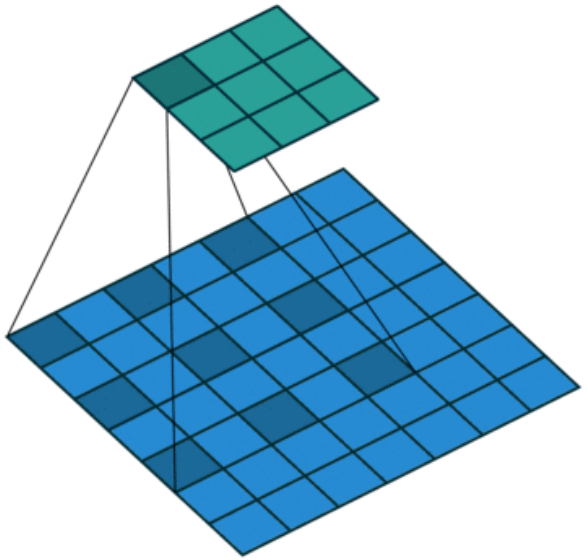
Attention
(Spatial and over Channels)

Dilated (Atrous) Convolutions

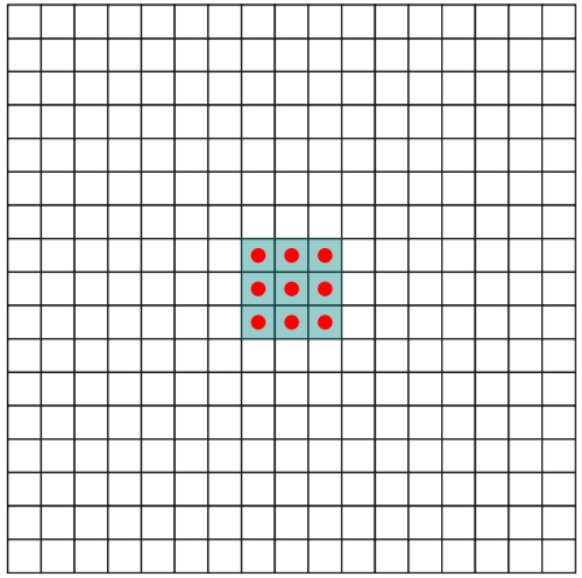
Problem: increasing the receptive field costs a lots of parameters.

Idea: spread out the samples used in each convolution.

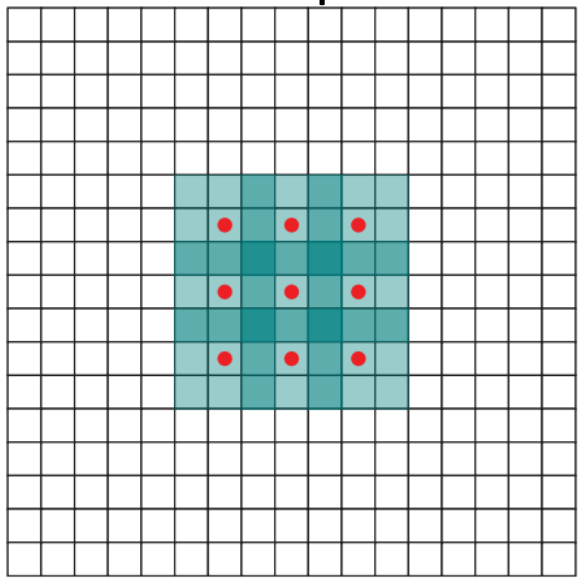
dilated convolution



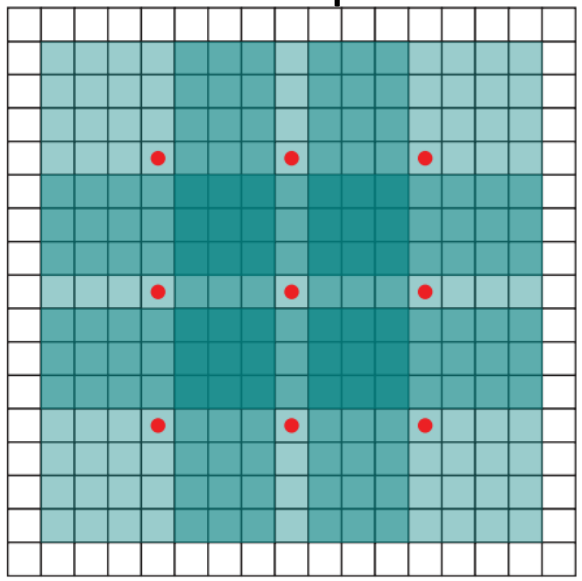
1st layer: not dilated
3x3 recep. field



2nd layer: 1-dilated
7x7 recep. field



3rd layer: 2-dilated
15x15 recep. field



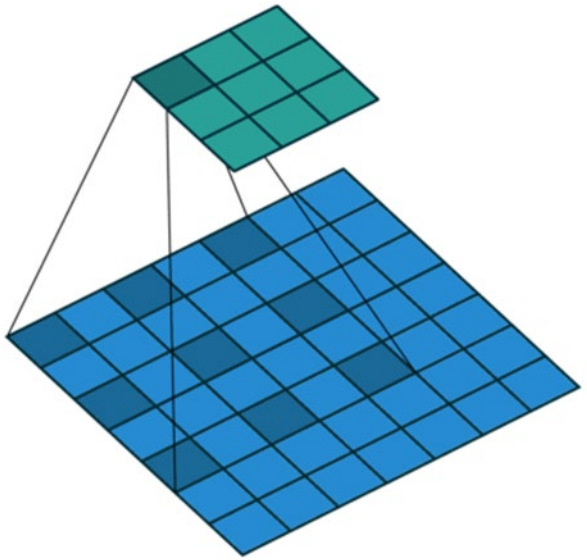
Images from: Dumoulin and Visin, *A guide to convolution arithmetic for deep learning*, arXiv 2016
Yu and Koltun, *Multi-scale Context Aggregation by Dilated Convolutions*, ICLR 2016

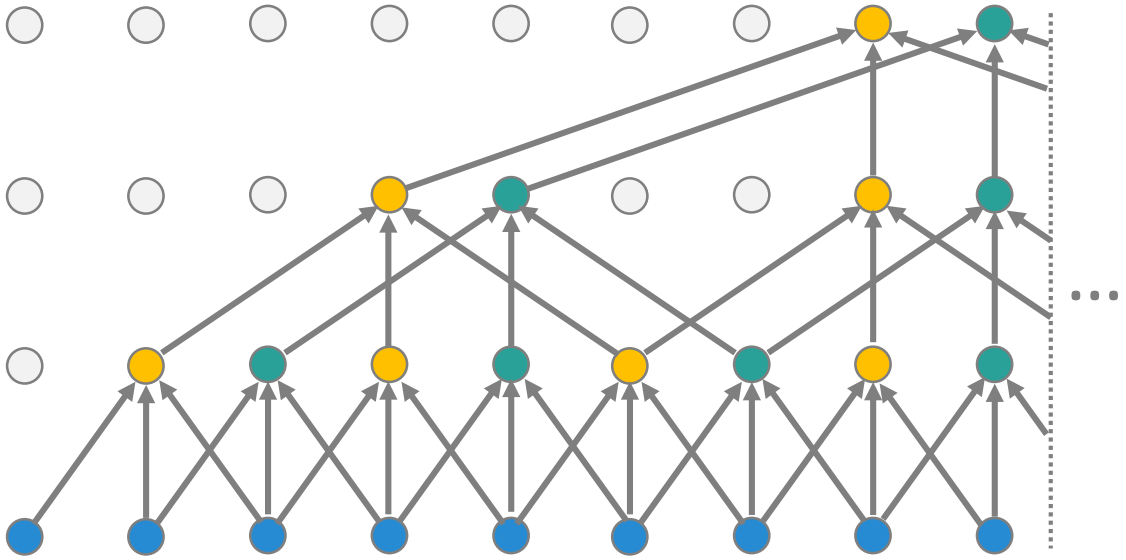
Dilated (Atrous) Convolutions

Problem: increasing the receptive field costs a lots of parameters.

Idea: spread out the samples used for a convolution.

dilated convolution





3rd layer: 2-dilated
15x15 recep. field

2nd layer: 1-dilated
7x7 recep. field

1st layer: not dilated
3x3 recep. field

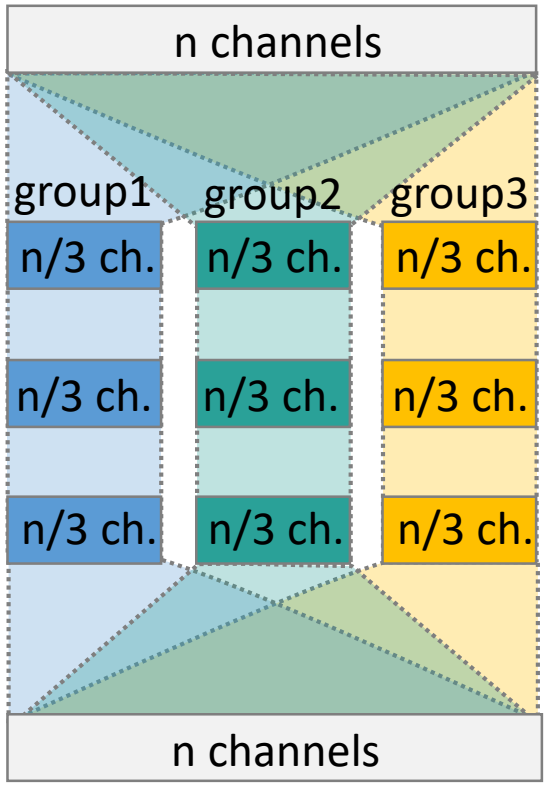
Input image

Dumoulin and Visin, *A guide to convolution arithmetic for deep learning*, arXiv 2016

Grouped Convolutions (Inception Modules)

Problem: conv. parameters grow quadratically in the number of channels

Idea: split channels into groups, remove connections between different groups



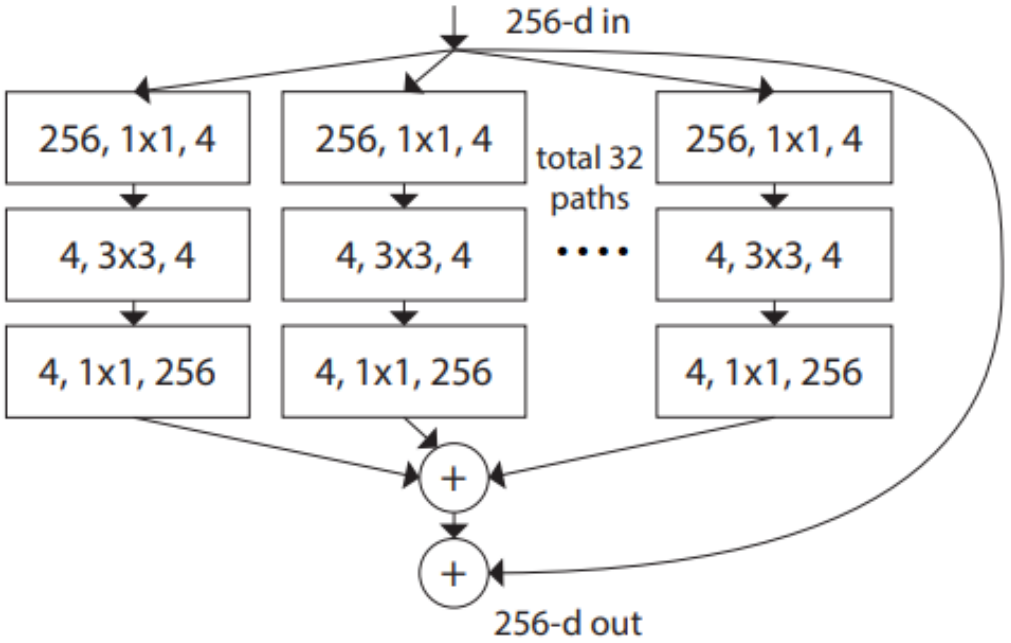
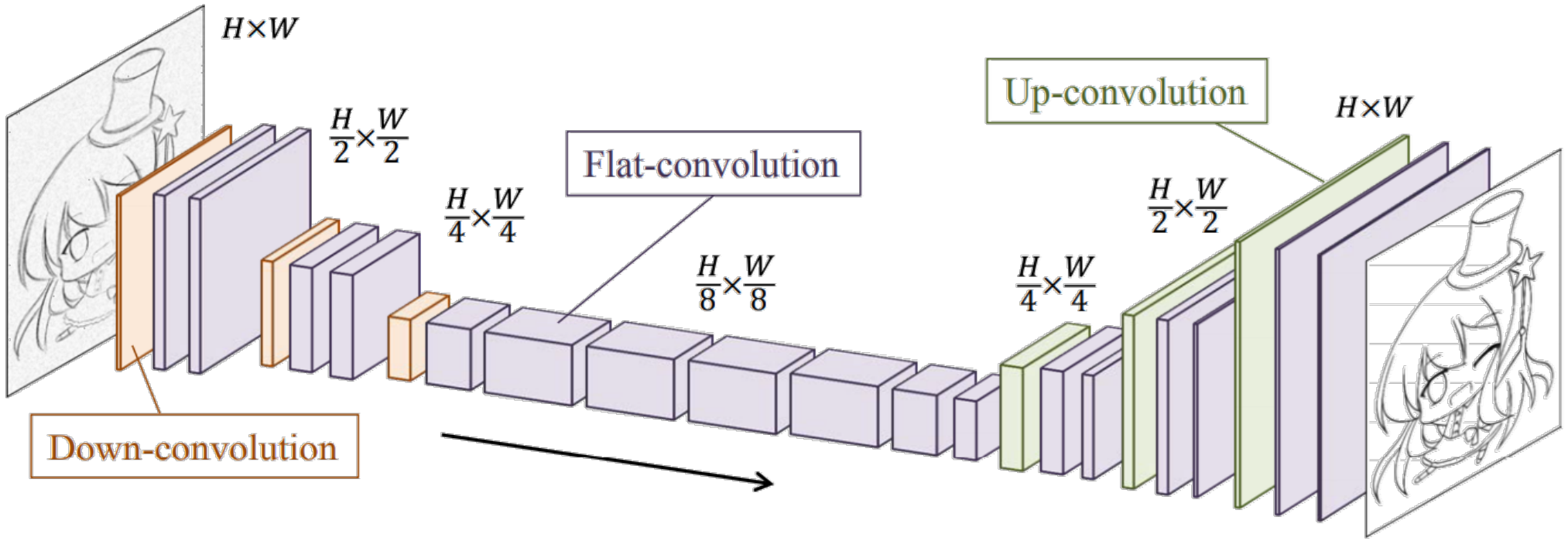


Image from: Xie et al., *Aggregated Residual Transformations for Deep Neural Networks*, CVPR 2017

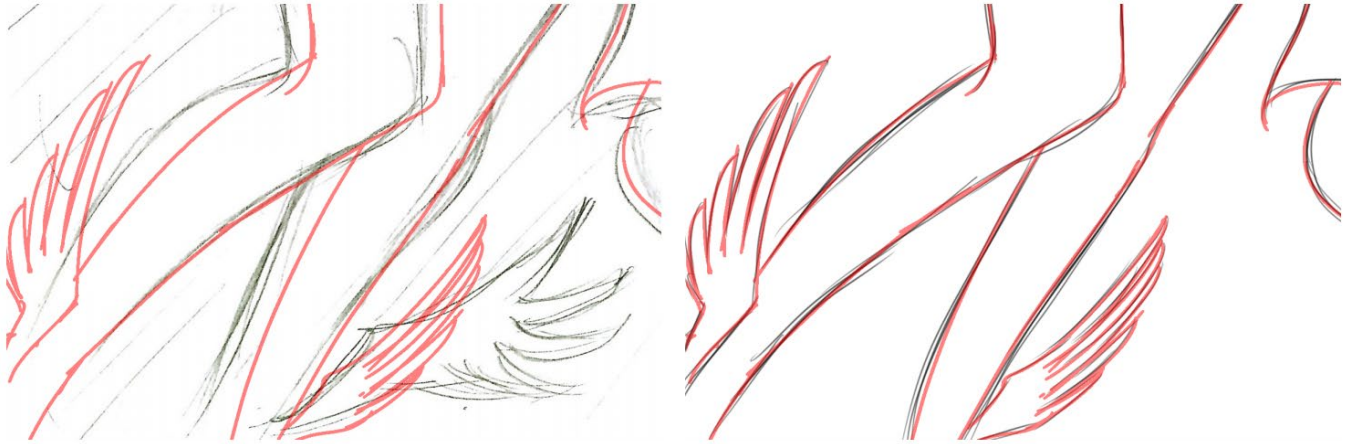
Example: Sketch Simplification



Learning to Simplify: Fully Convolutional Networks for Rough Sketch Cleanup, Simo-Serra et al.

Example: Sketch Simplification

- Loss for thin edges saturates easily
- Authors take extra steps to align input and ground truth edges



Pencil: input
Red: ground truth

Learning to Simplify: Fully Convolutional Networks for Rough Sketch Cleanup, **Simo-Serra et al.**

Image Decomposition

- A selection of methods:
- *Direct Instrinsics*, Narihira et al., 2015
- *Learning Data-driven Reflectance Priors for Intrinsic Image Decomposition,* Zhou et al., 2015
- *Decomposing Single Images for Layered Photo Retouching*, Innamorati et al. 2017

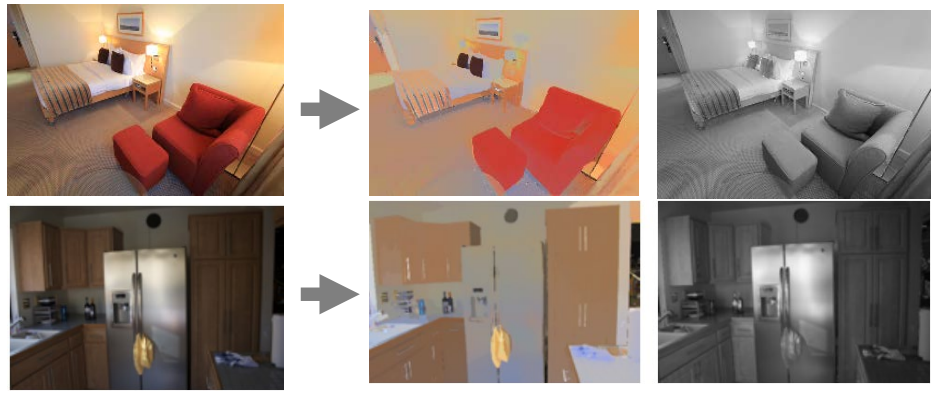
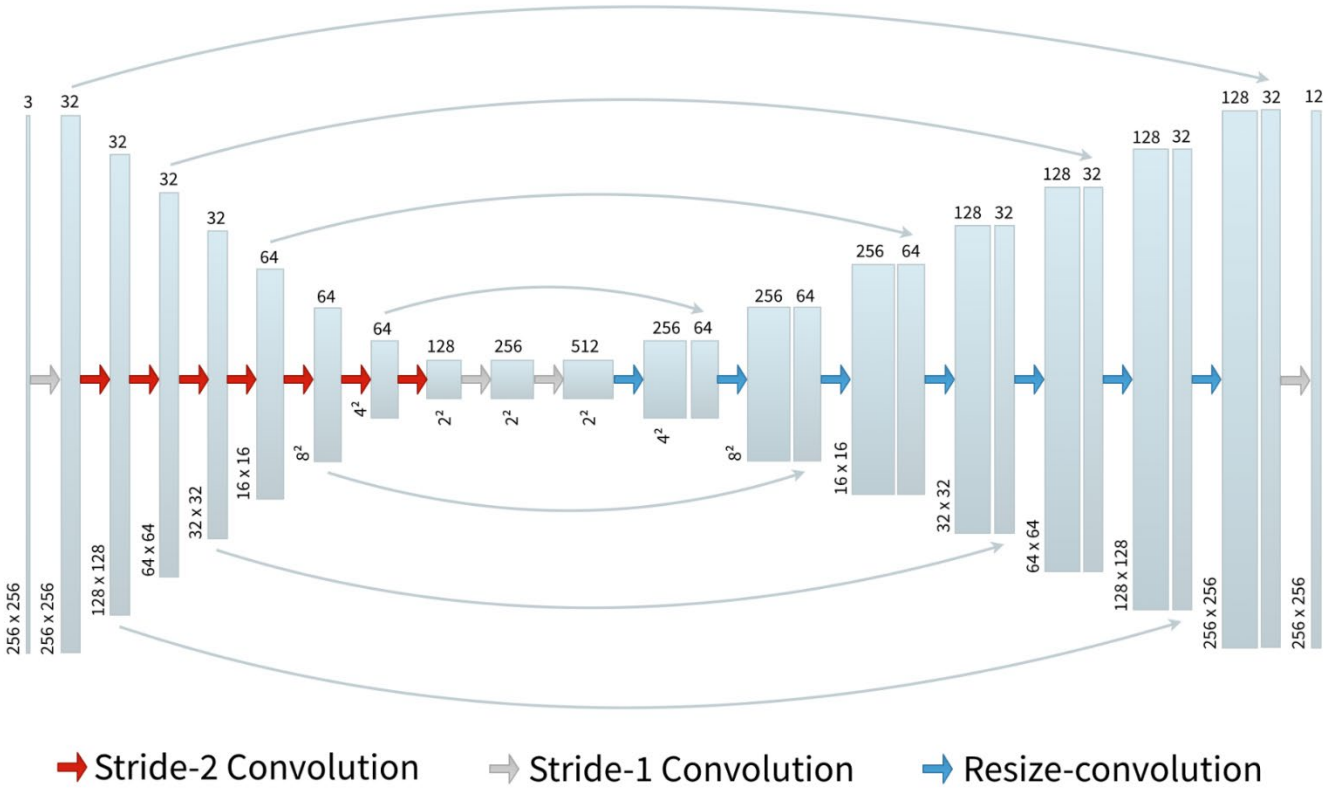


Image Decomposition: Decomposing *Single Images for Layered Photo Retouching*

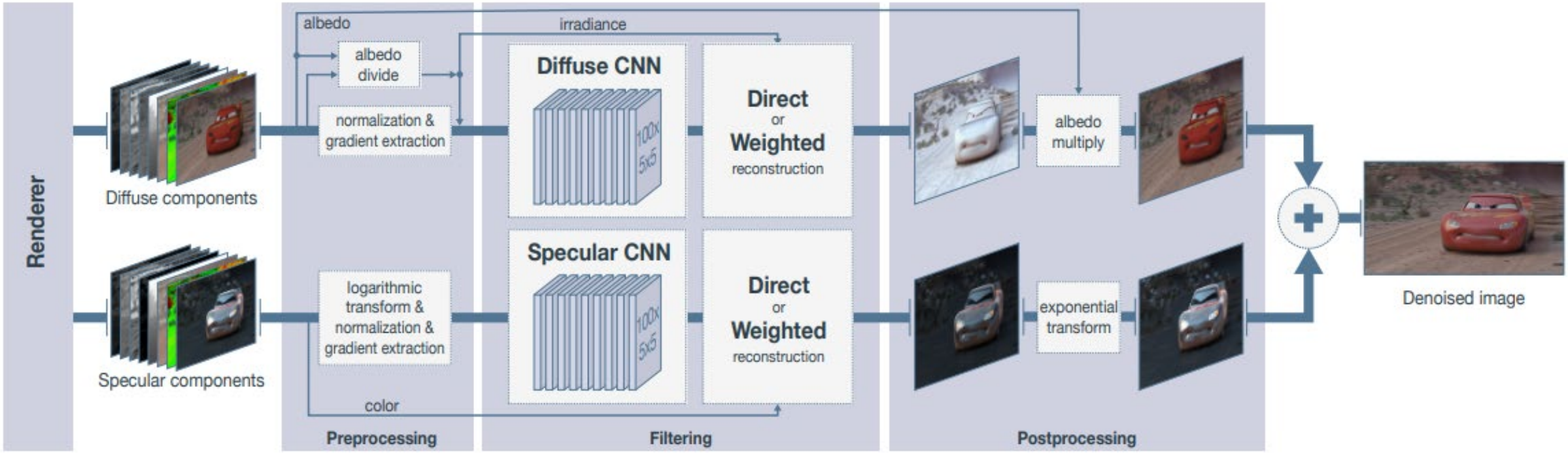


Albedo

Irradiance

Specular

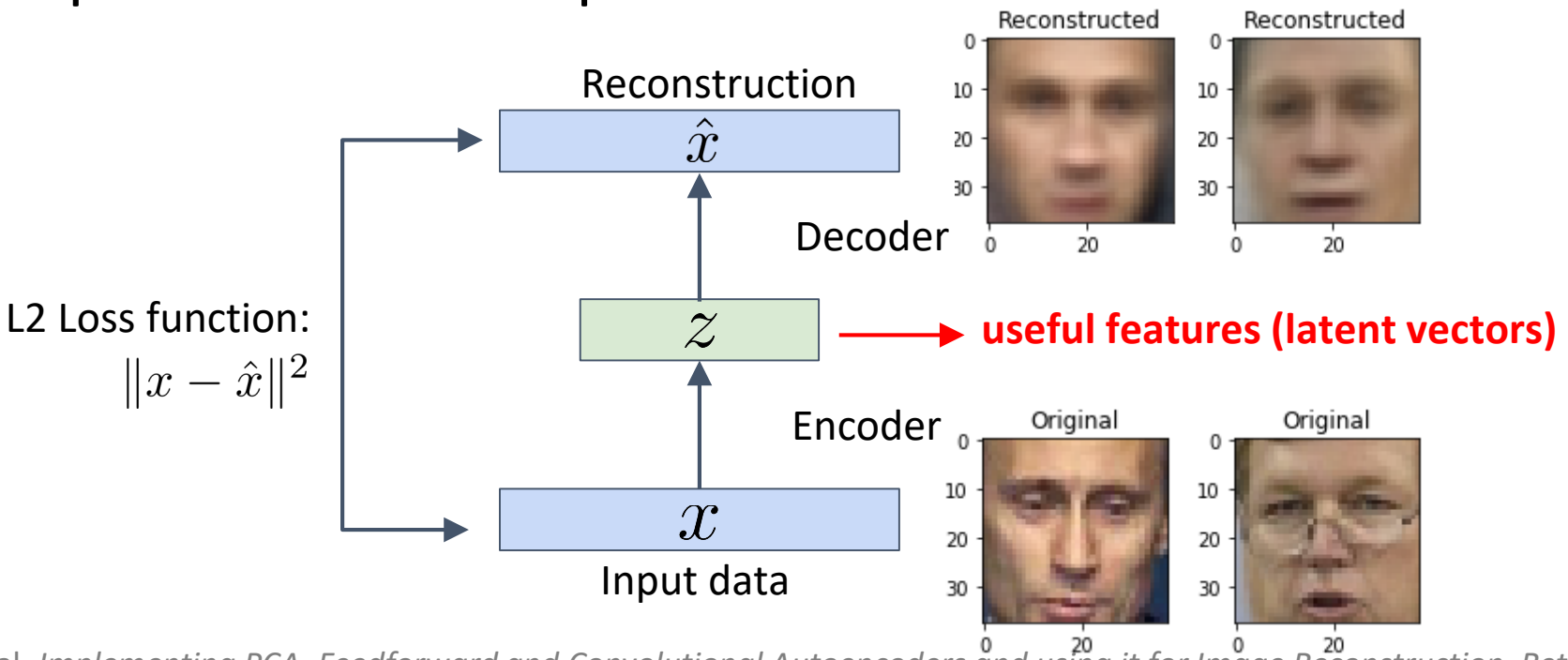
Example Application: Denoising



Deep Features

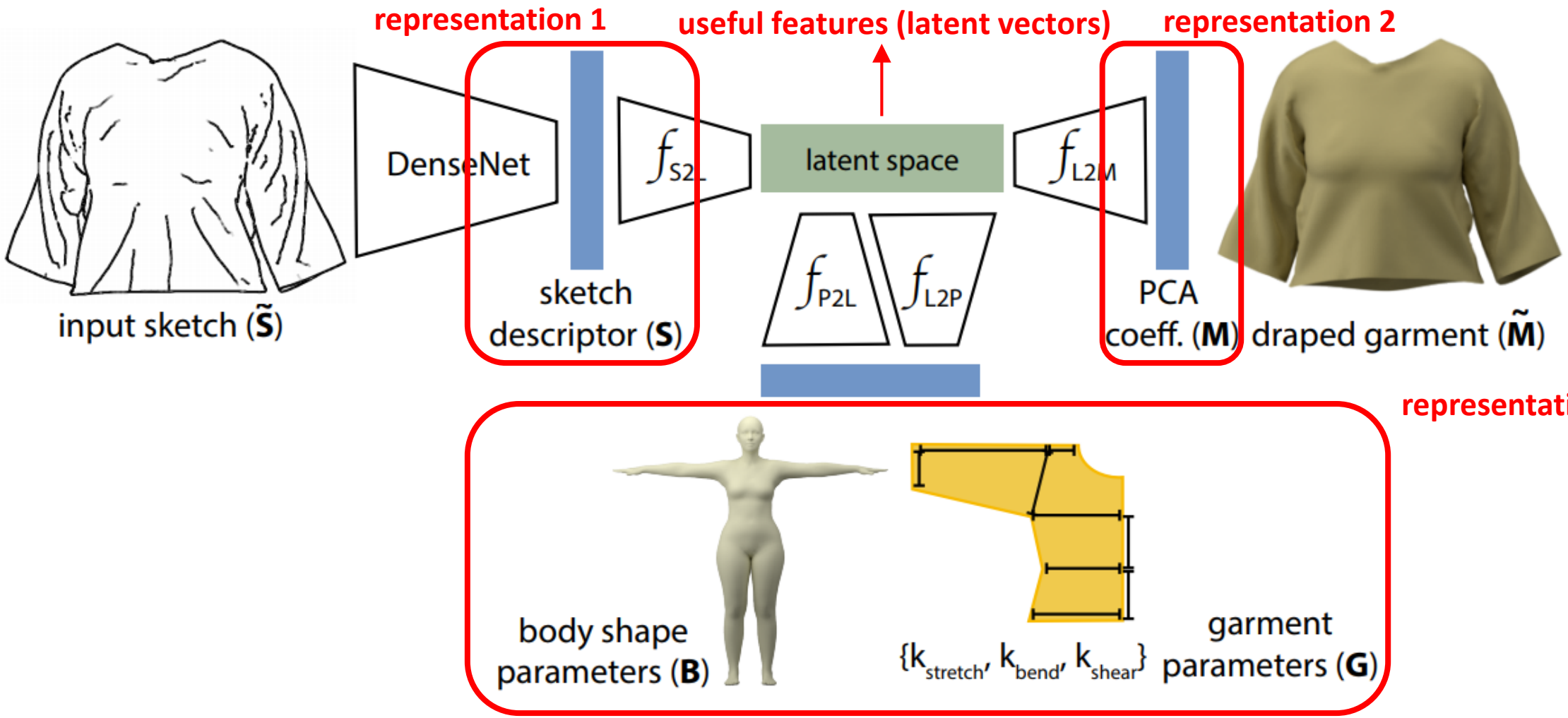
Autoencoders

- Features learned by deep networks are useful for a large range of tasks.
- An autoencoder is a simple way to obtain these features.
- Does not require additional supervision.



Manash Kumar Mandal, *Implementing PCA, Feedforward and Convolutional Autoencoders and using it for Image Reconstruction, Retrieval & Compression*, https://blog.manash.me/

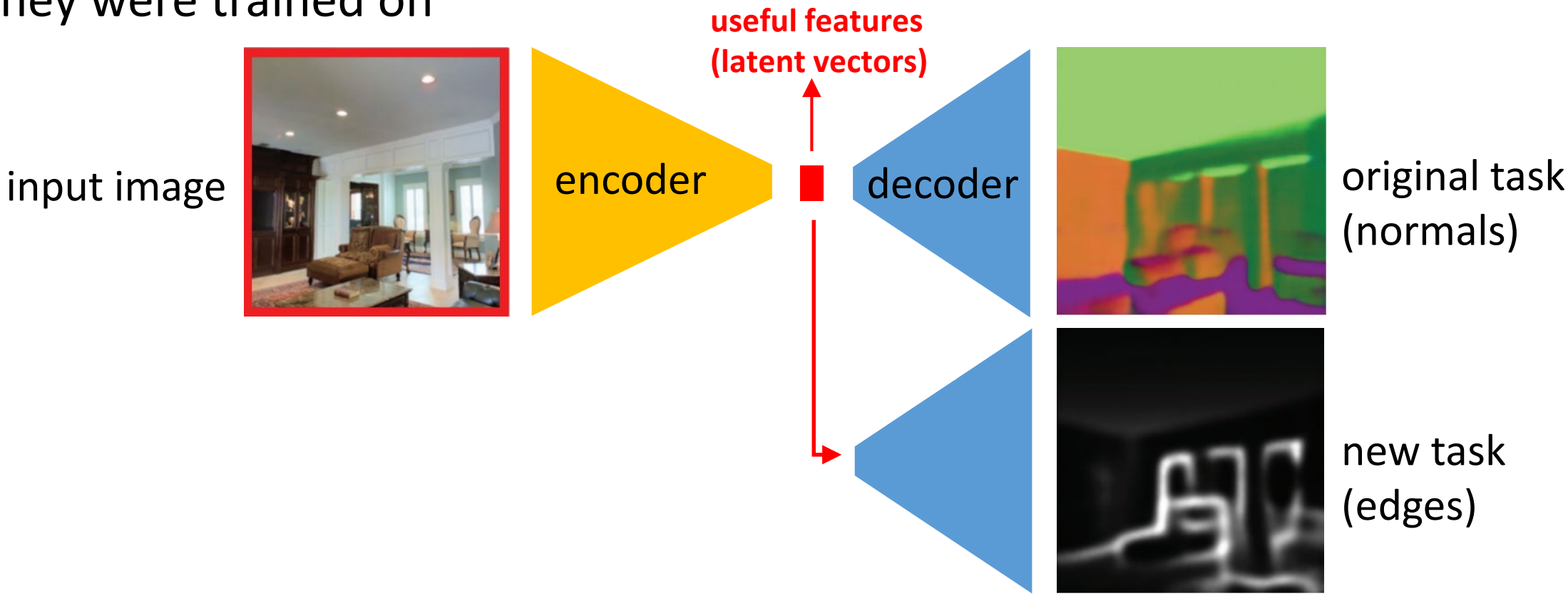
Shared Feature Space: Interactive Garments



Wang et al., *Learning a Shared Shape Space for Multimodal Garment Design*, Siggraph Asia 2018

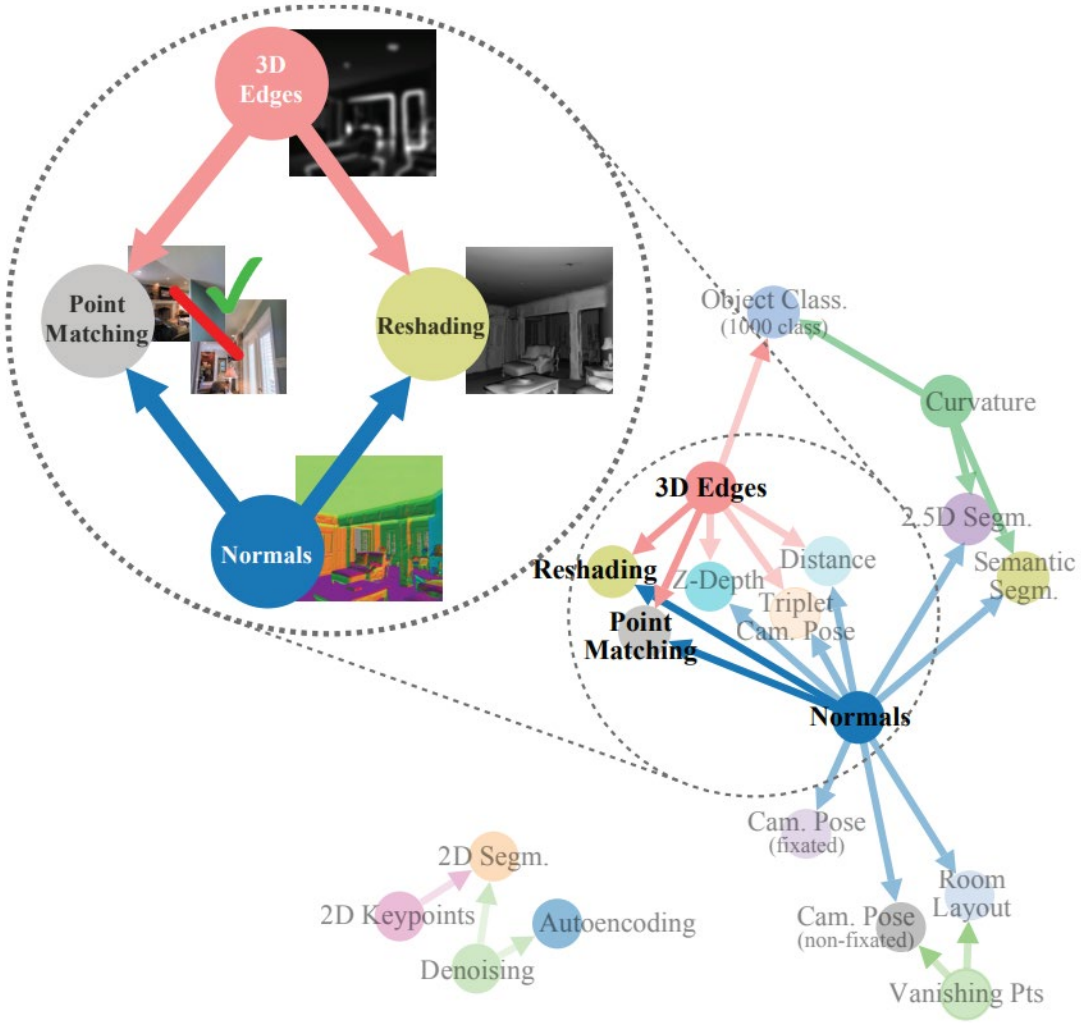
Transfer Learning

Features extracted by well-trained CNNs often generalize beyond the task they were trained on

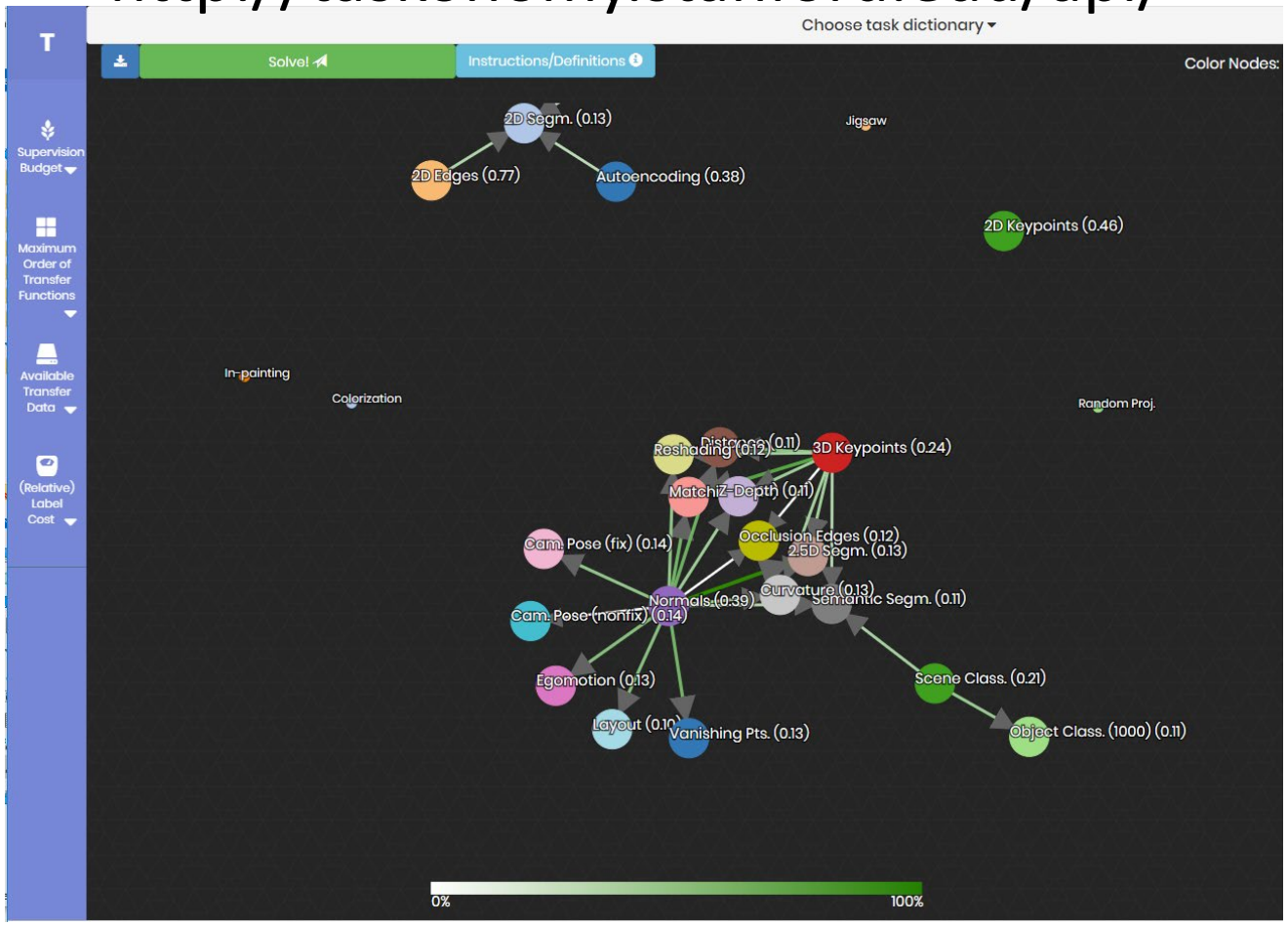


Images from: Zamir et al., *Taskonomy: Disentangling Task Transfer Learning*, CVPR 2018

Taxonomy of Tasks: Taskonomy

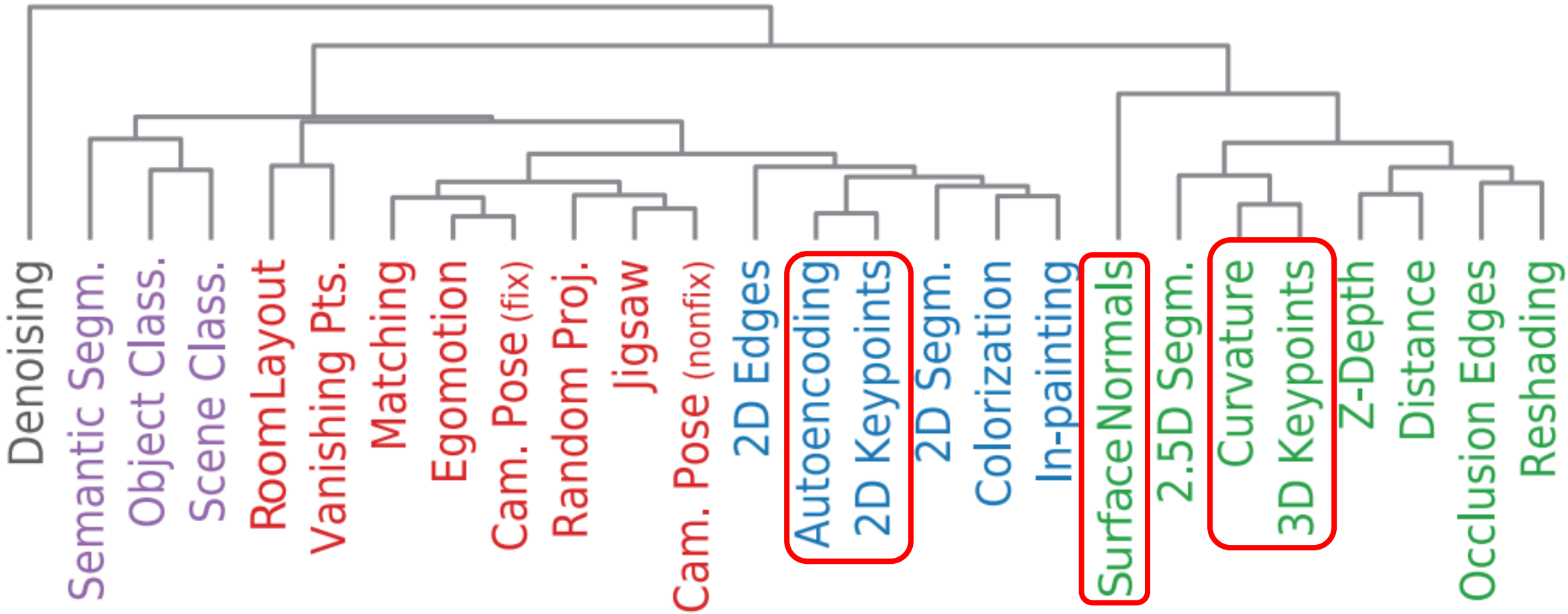


http://taskonomy.stanford.edu/api/



Images from: Zamir et al., *Taskonomy: Disentangling Task Transfer Learning*, CVPR 2018

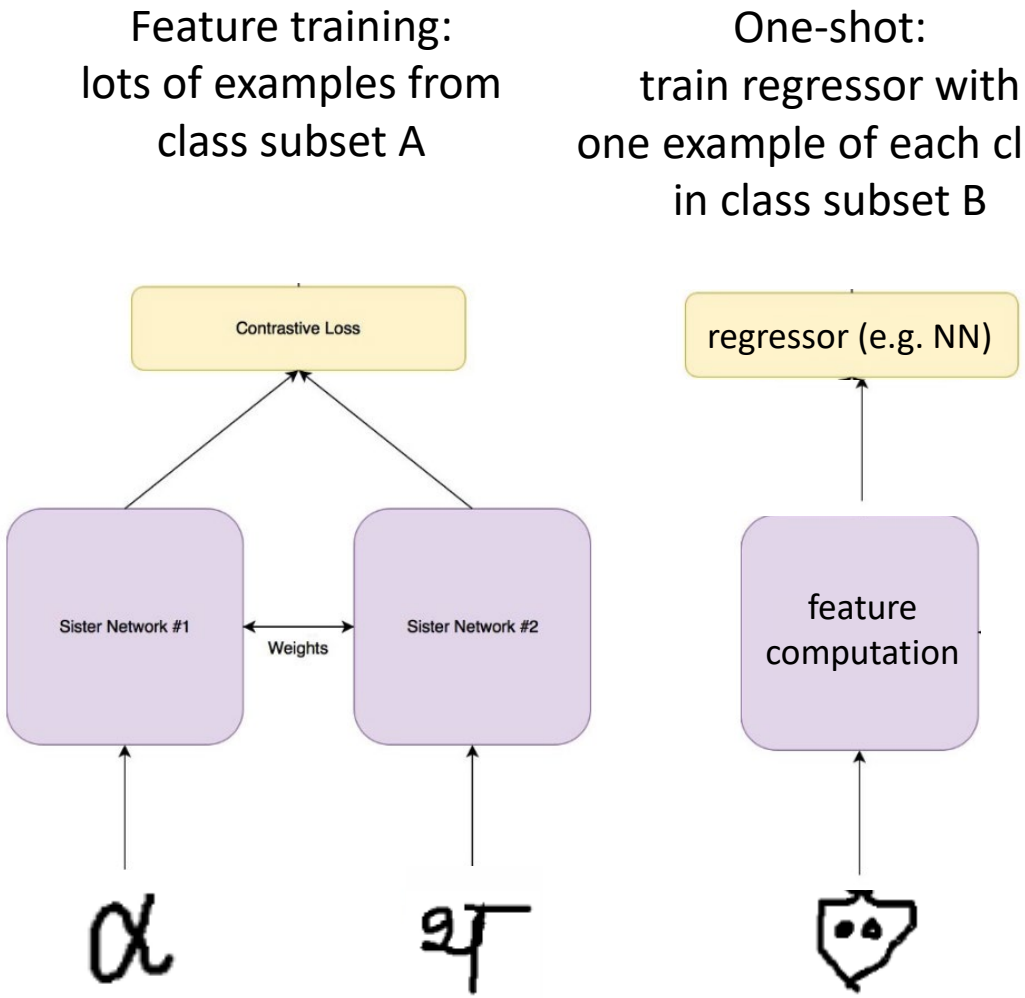
Taxonomy of Tasks: Taskonomy



Images from: Zamir et al., *Taskonomy: Disentangling Task Transfer Learning*, CVPR 2018

Few-shot, One-shot Learning

- With a good feature space, tasks become easier
- In classification, for example, nearest neighbors might already be good enough
- Often trained with a Siamese network, to optimize the metric in feature space



https://hackernoon.com/one-shot-learning-with-siamese-networks-in-pytorch-8ddaab10340e

Style Transfer

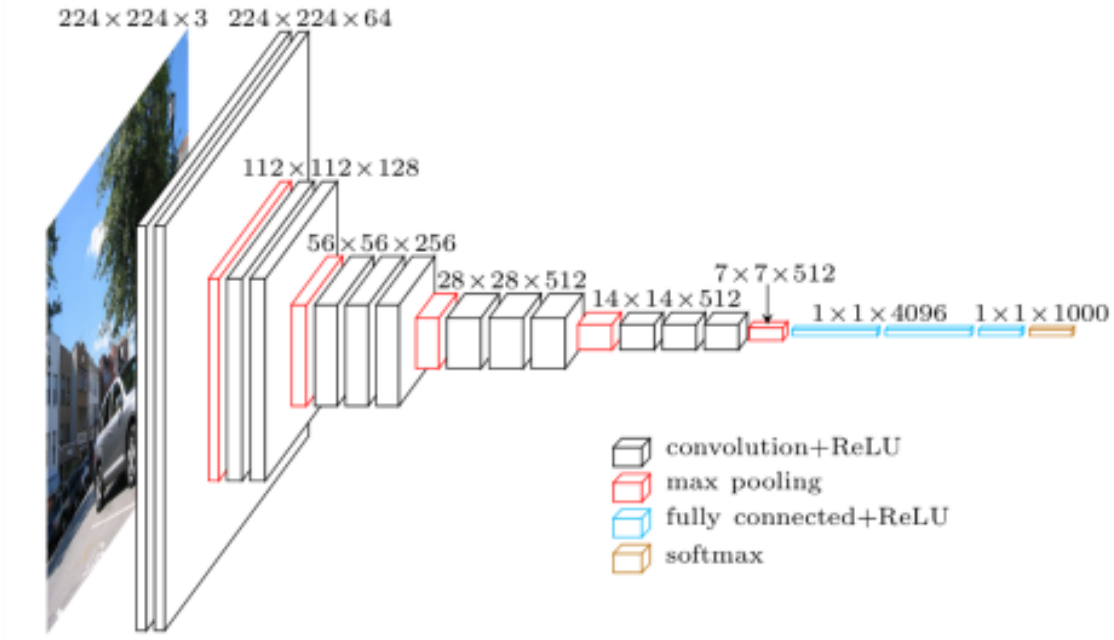
- Combine content from image A with style from image B

Images from: Gatys et al., *Image Style Transfer using Convolutional Neural Networks*, CVPR 2016

What is Style and Content?

Remember that features in a CNN often generalize well.

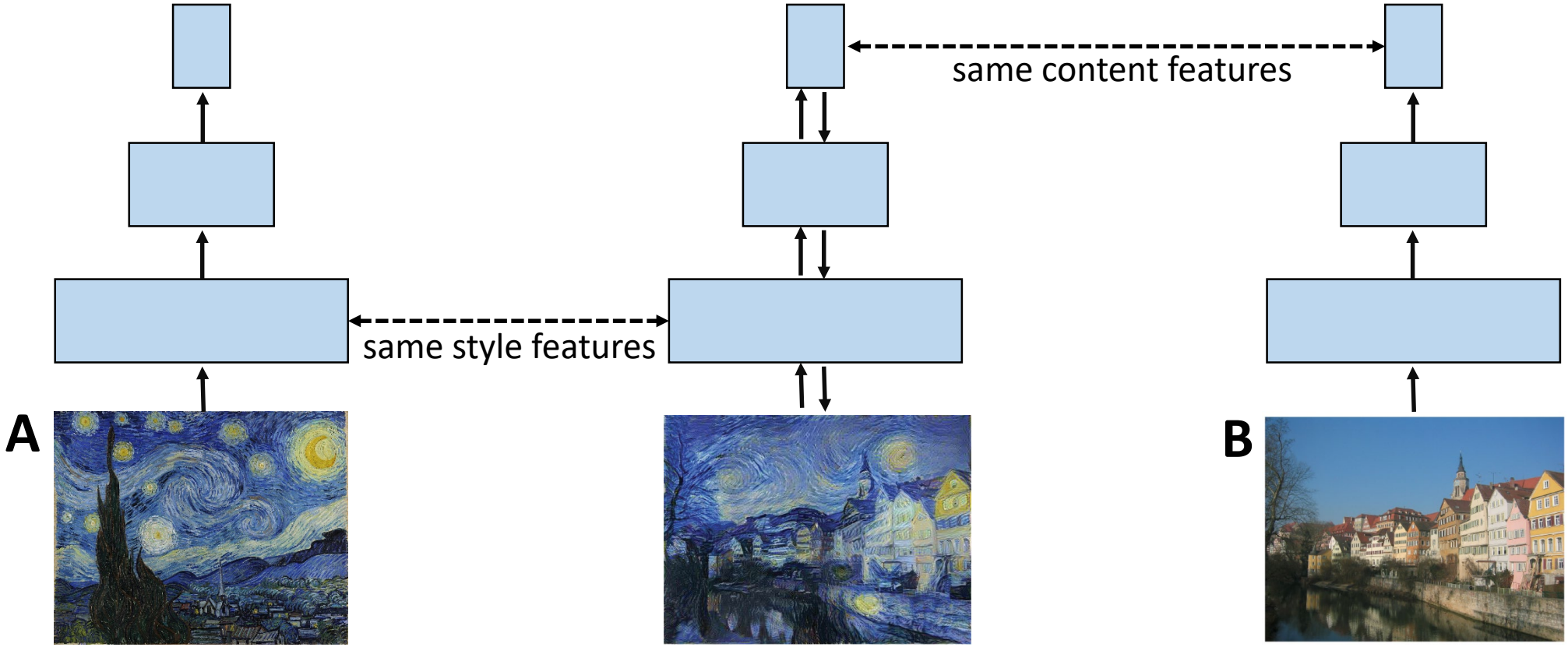
Define style and content using the layers of a CNN (VGG19 for example):



shallow layers describe style ⟷ deeper layers describe content

Optimize for Style A and Content B

same pre-trained networks, fix weights



optimize to have same style/content features

Style Transfer: Follow-Ups

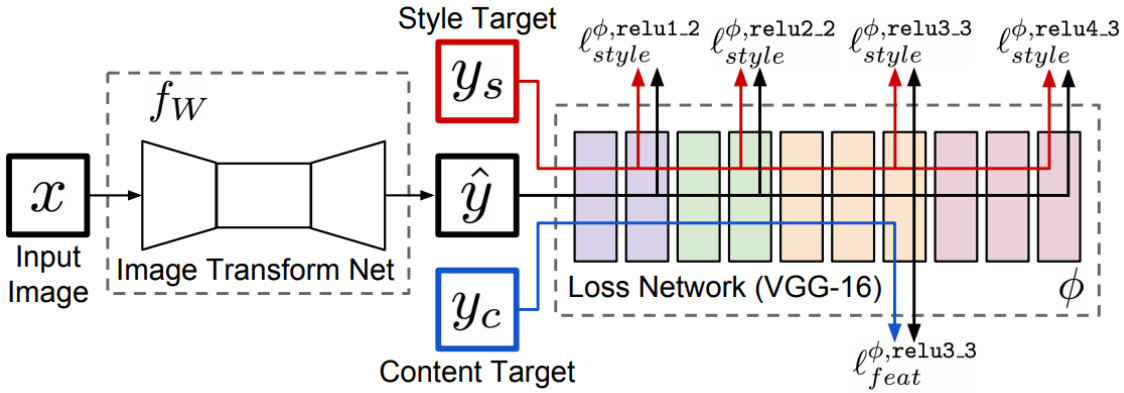
more control over the result

(a) Content

(b) Style I

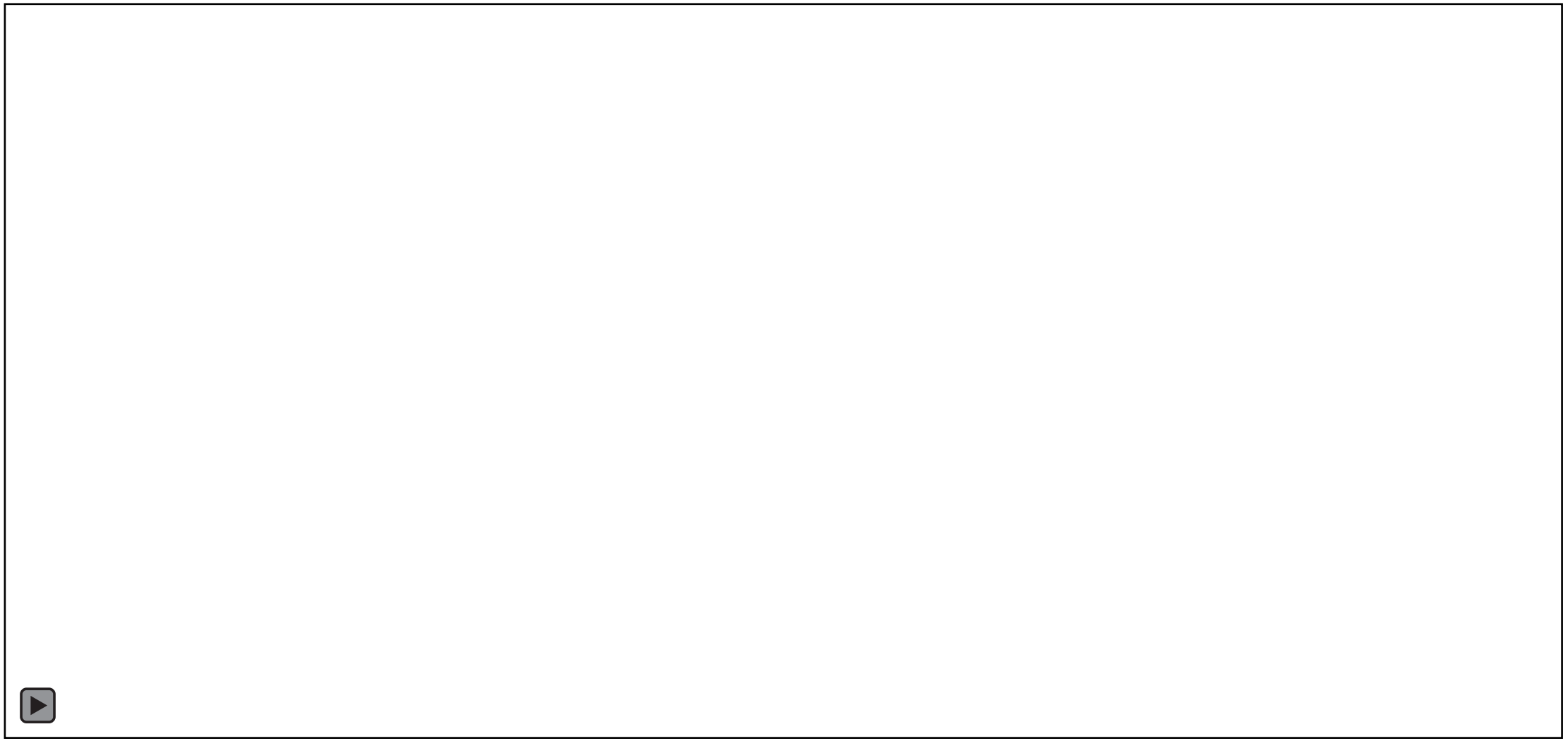
(c) Style II

feed-forward networks



Images from: Gatys, et al., *Controlling Perceptual Factors in Neural Style Transfer*, CVPR 2017
Johnson et al., *Perceptual Losses for Real-Time Style Transfer and Super-Resolution*, ECCV 2016

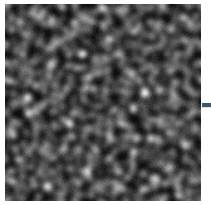
Style Transfer for Videos



Ruder et al., *Artistic Style Transfer for Videos*, German Conference on Pattern Recognition 2016

Adversarial Image Generation

Generative Adversarial Networks



Player 1: generator
Scores if discriminator can't distinguish output from real image

from dataset

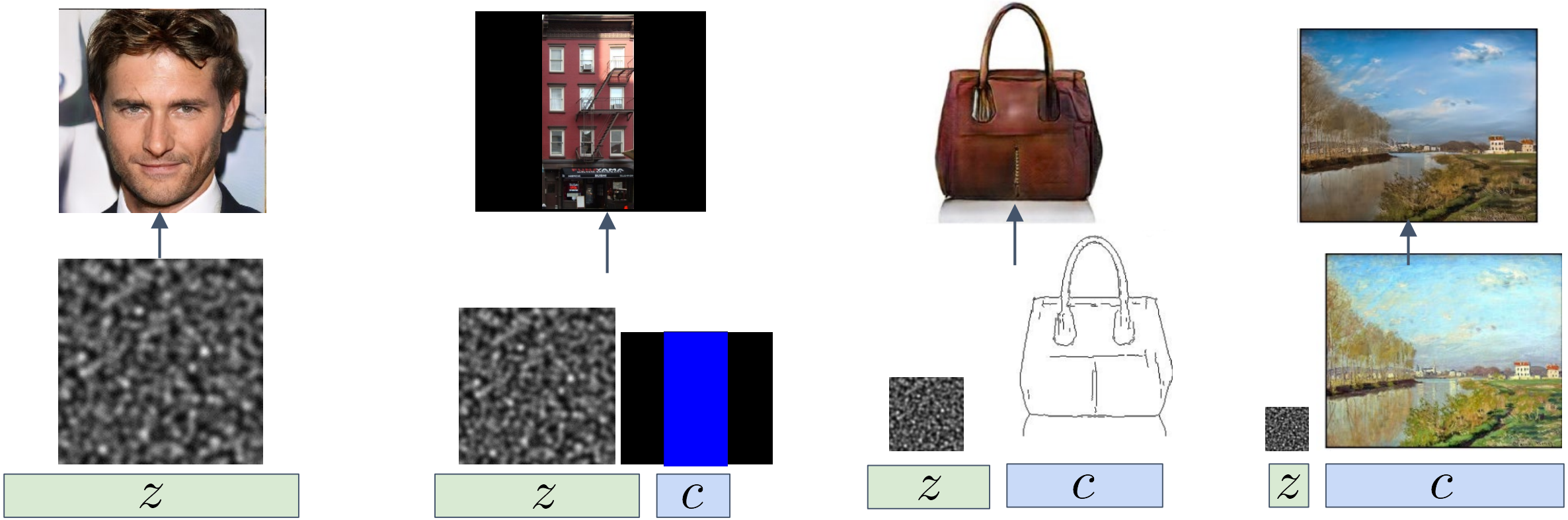
Player 2: discriminator → real/fake
Scores if it can distinguish between real and fake

GANs to CGANs (Conditional GANs)

GAN

CGAN

increasingly determined by the condition



GAN

Karras et al., *Progressive Growing of GANs for Improved Quality, Stability, and Variation*, ICLR 2018

Kelly and Guerrero et al., *FrankenGAN: Guided Detail Synthesis for Building Mass Models using Style-Synchonized GANs*, Siggraph Asia 2018

Isola et al., *Image-to-Image Translation with Conditional Adversarial Nets*, CVPR 2017

Image Credit: Zhu et al. , *Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks* , ICCV 2017

Image-to-image Translation

- ≈ learn a mapping between images from example pairs
- Approximate sampling from a conditional distribution $p_{\text{data}}(x \mid c)$

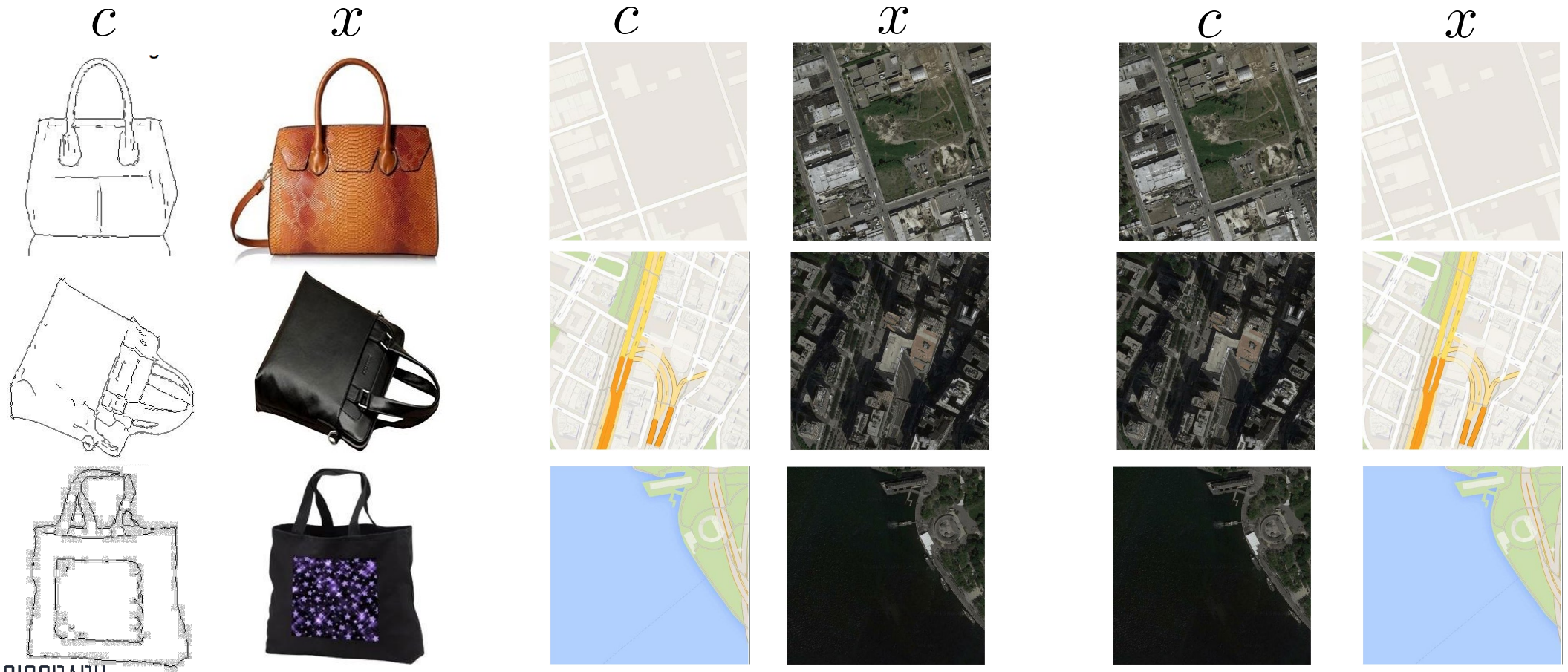
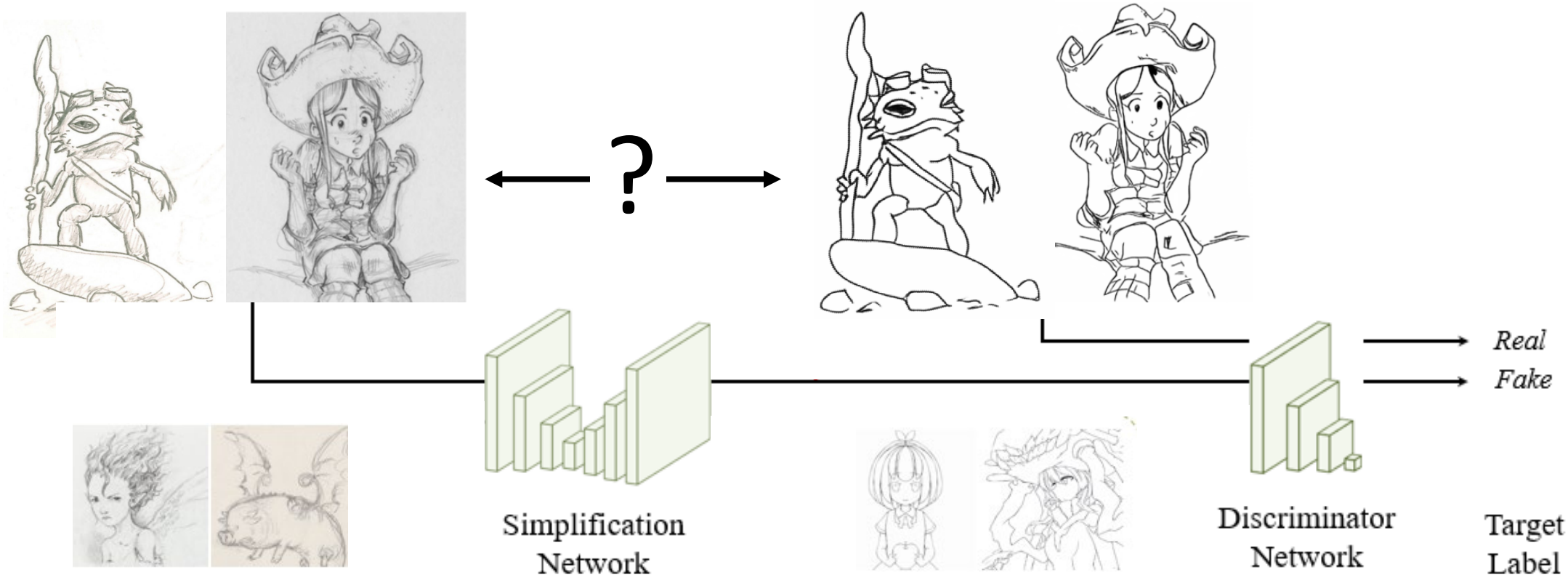


Image Credit: *Image-to-Image Translation with Conditional Adversarial Nets*, Isola et al.

Adversarial Loss vs. Manual Loss

Problem: A good loss function is often hard to find

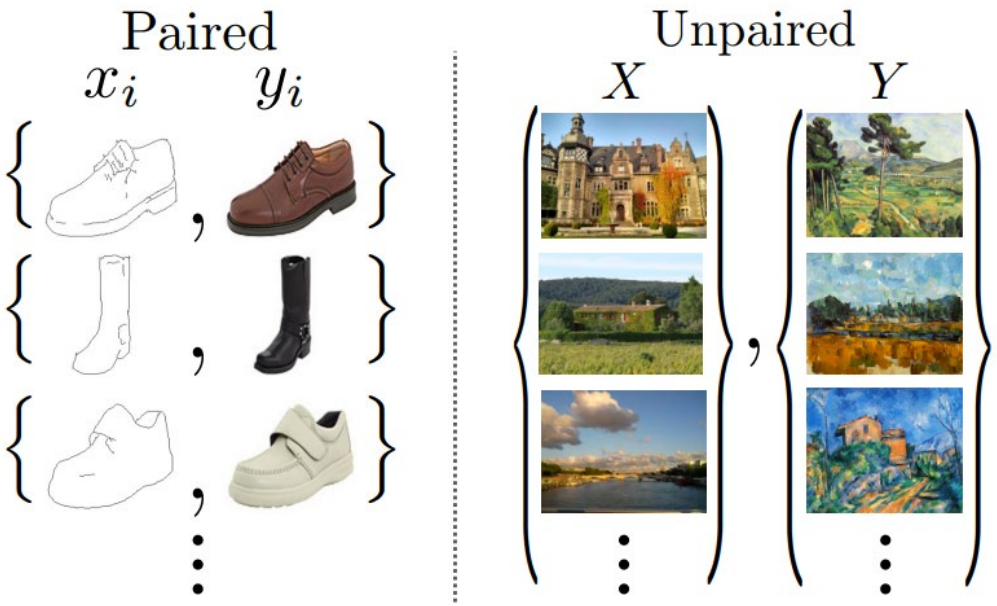
Idea: Train a network to discriminate between network output and ground truth



Images from: Simo-Serra, Iizuka and Ishikawa, *Mastering Sketching*, Siggraph 2018

CycleGANs

- Less supervision than CGANs: mapping between unpaired datasets
- Two GANs + cycle consistency



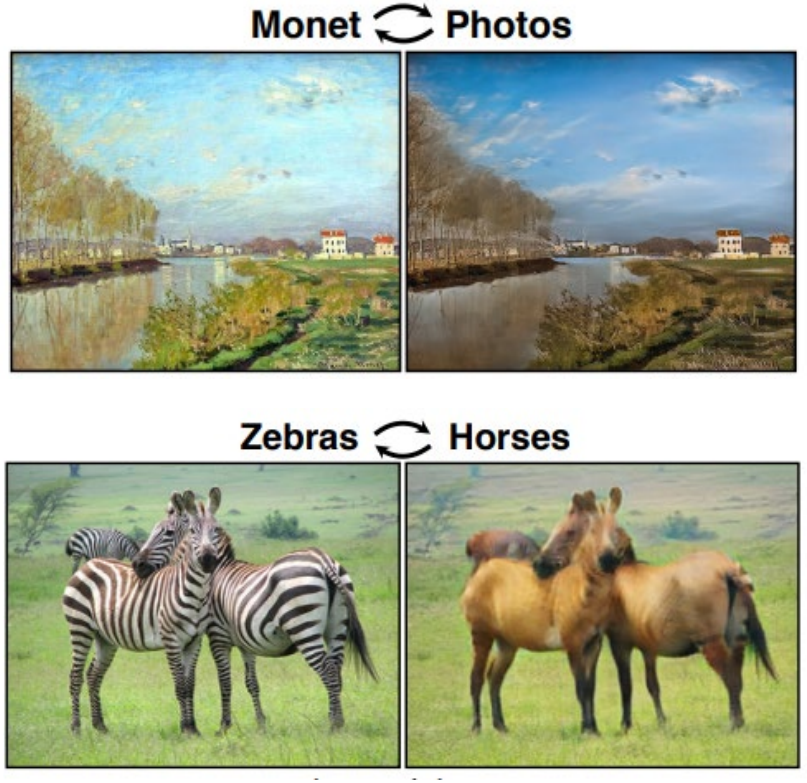


Image Credit: *Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks*, Zhu et al.

CycleGAN: Two GANs …

- Not conditional, so this alone does not constrain generator input and output to match

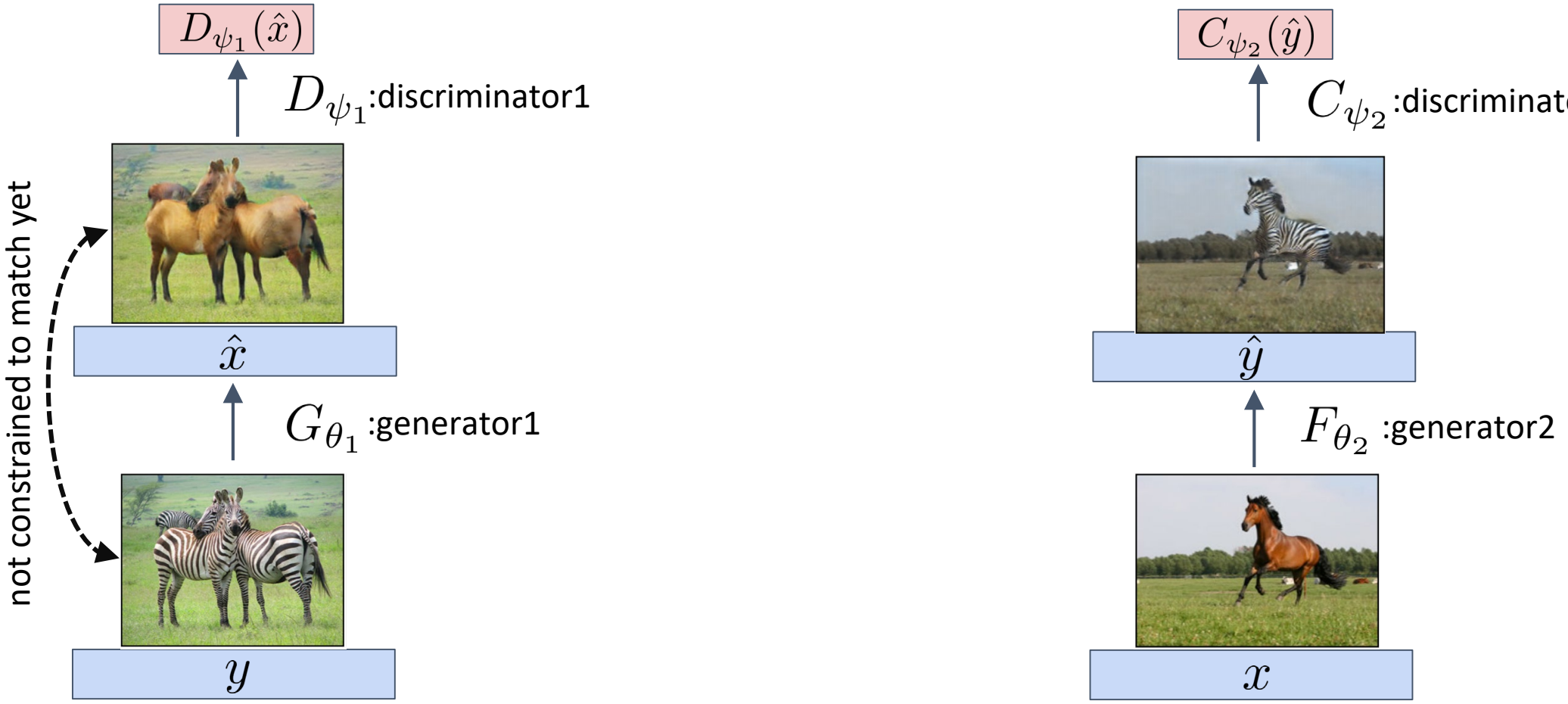


Image Credit: *Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks*, Zhu et al.

CycleGAN: … and Cycle Consistency

$\tilde{y}$

F_{θ_2} :generator2

$\hat{x}$

G_{θ_1} :generator1

y

L1 Loss function:
$\|y - \tilde{y}\|_1$

$\tilde{x}$

G_{θ_1} :generator1

$\hat{y}$

F_{θ_2} :generator2

x

L1 Loss function:
$\|x - \tilde{x}\|_1$

Image Credit: *Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks*, Zhu et al.

The Conditional Distribution in CGANs

A

$p(B|A)$

(a) Input night image

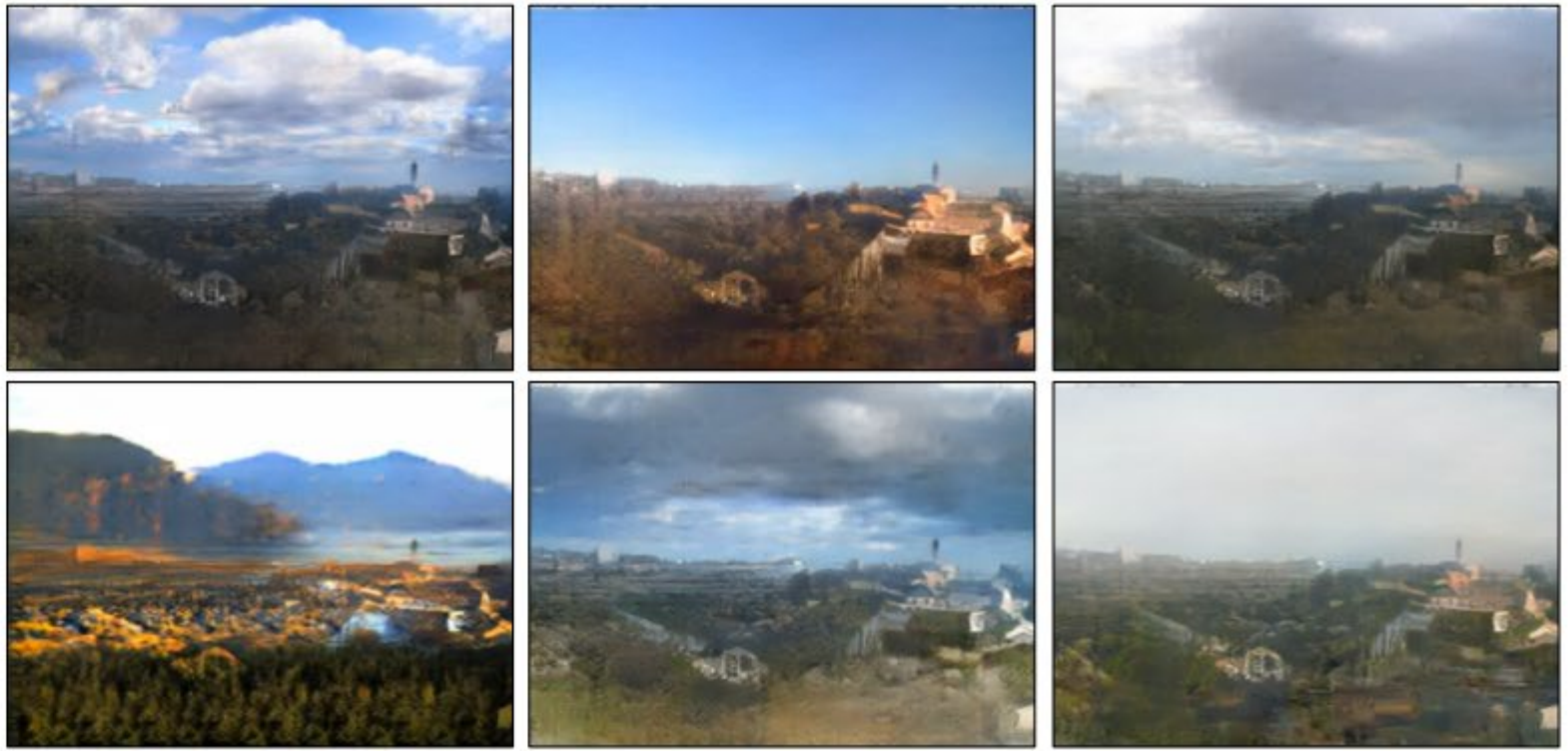
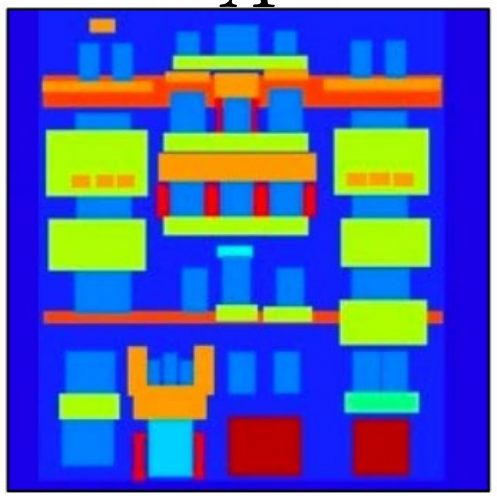


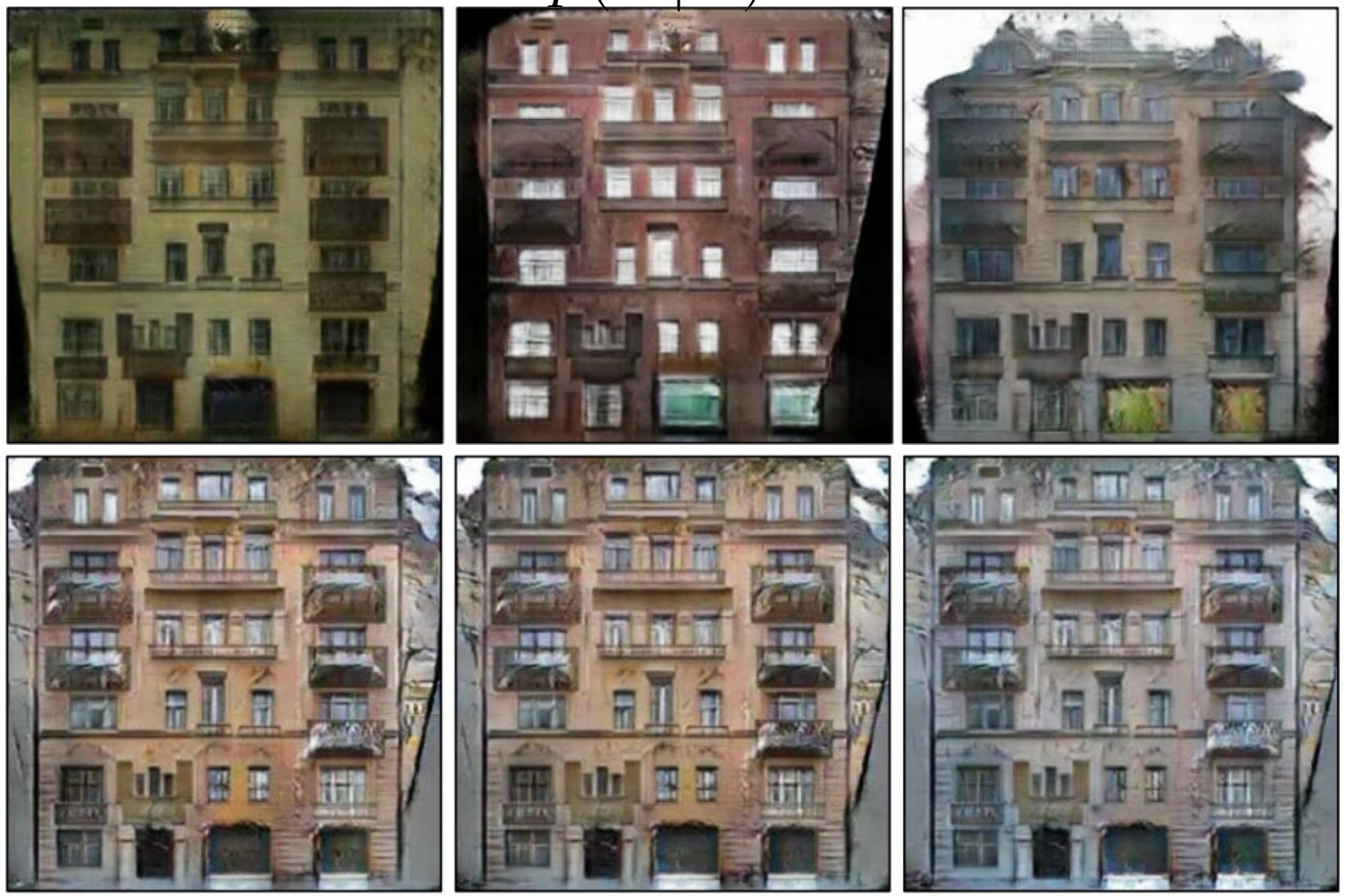
Image from: Zhu et al., *Toward Multimodal Image-to-Image Translation*, NIPS 2017

The Conditional Distribution in CGANs

A



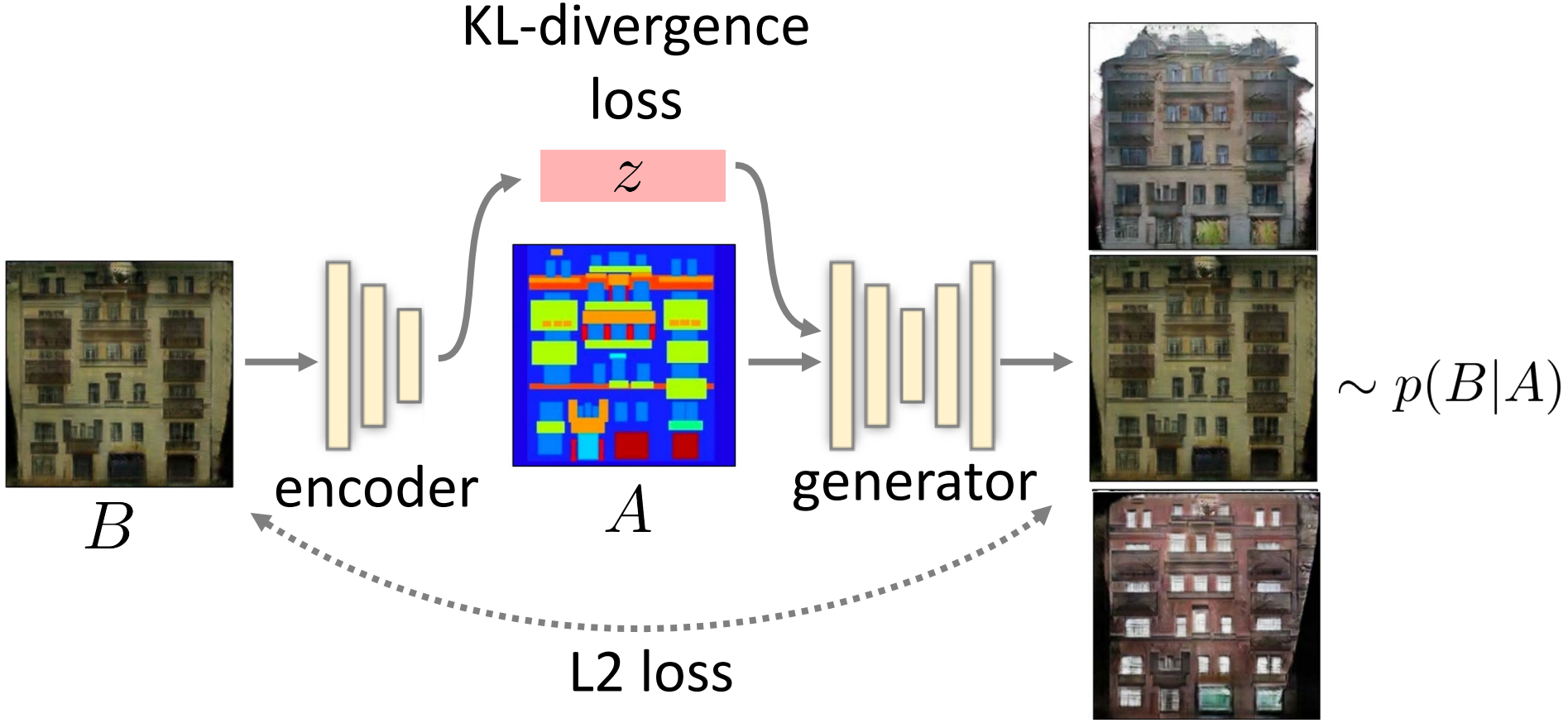
$p(B|A)$



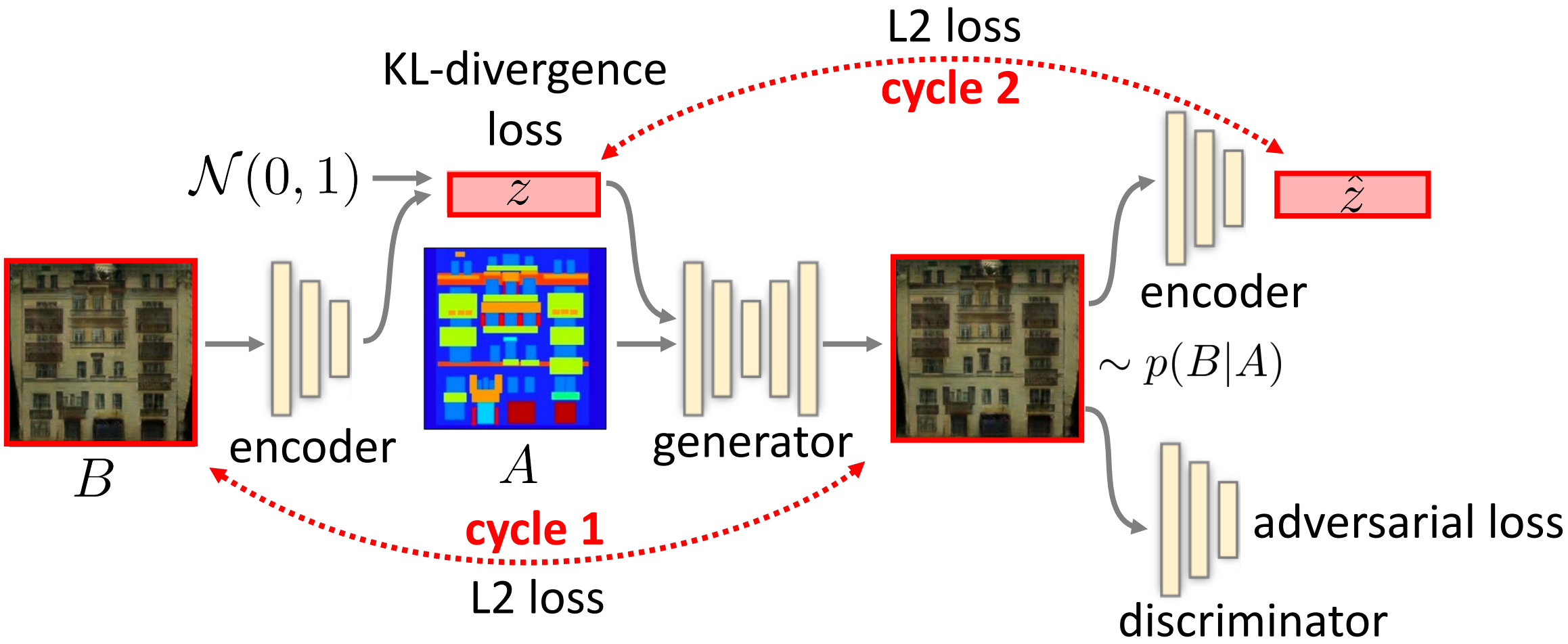
Pix2Pix

Zhu et al., *Toward Multimodal Image-to-Image Translation*, NIPS 2017

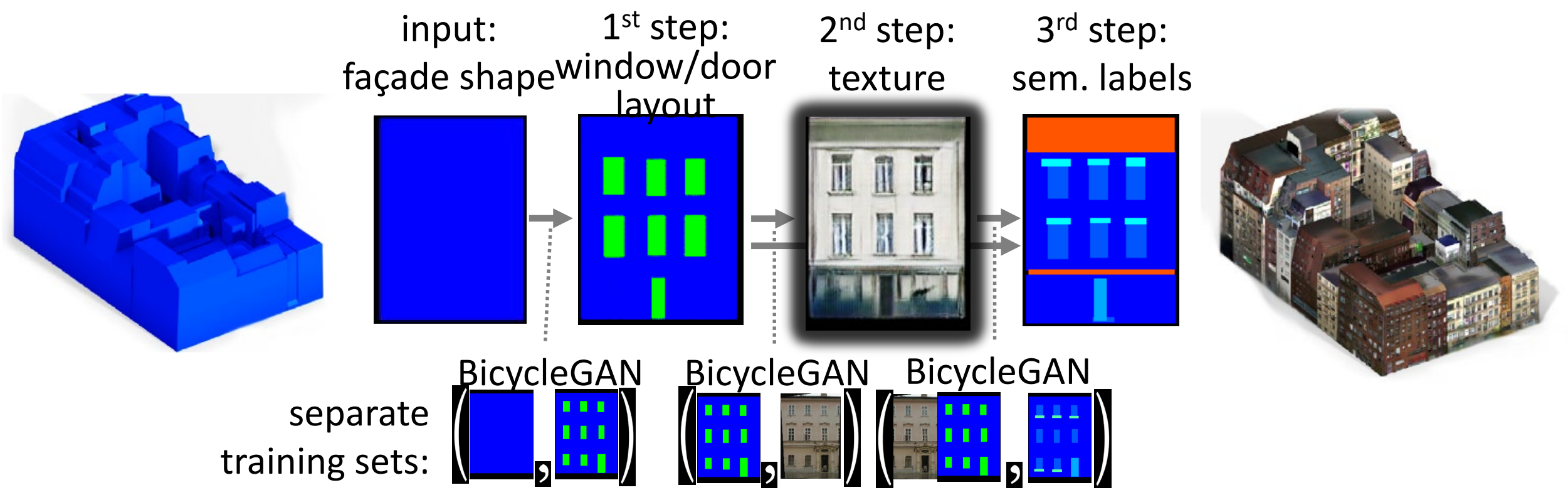
BicycleGAN



BicycleGAN

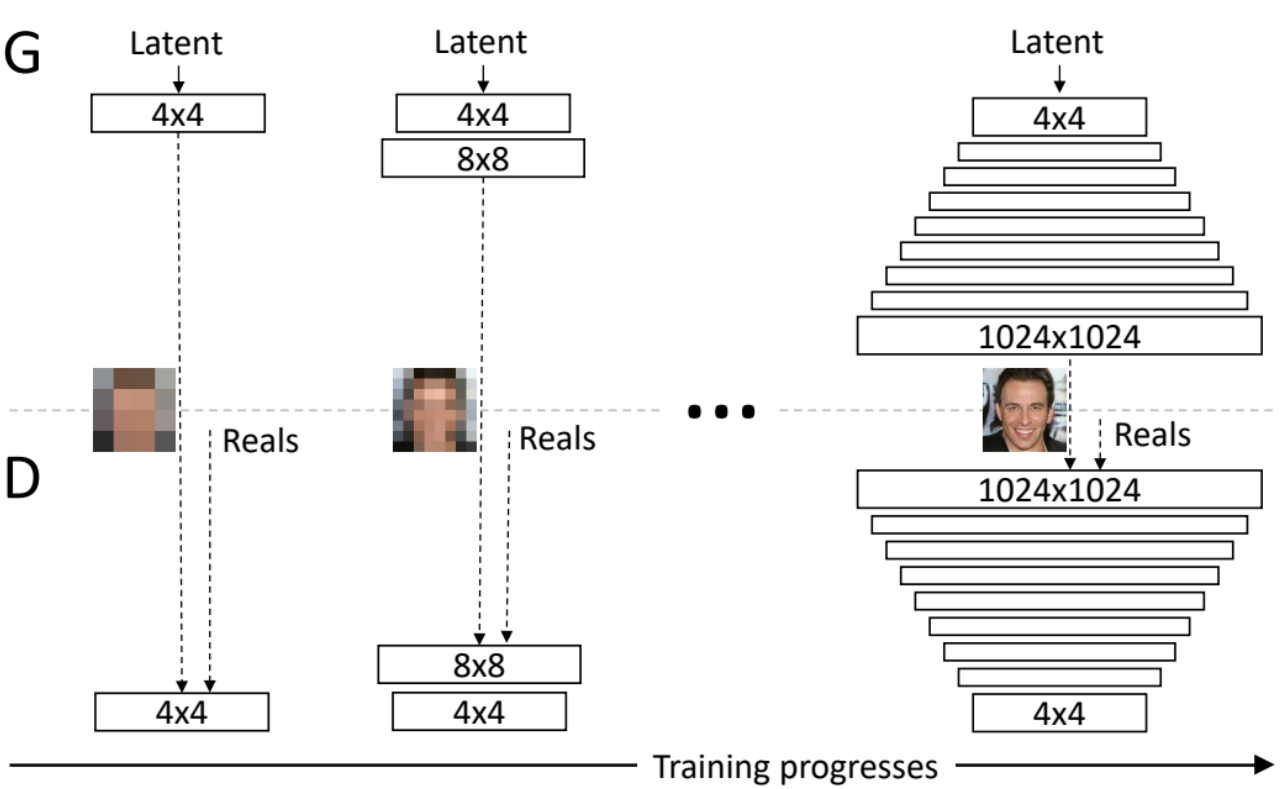

L2 loss
cycle 2
KL-divergence loss
$\mathcal{N}(0,1)$
z
$\hat{z}$
encoder
$\sim p(B|A)$
B
encoder
A
generator
adversarial loss
cycle 1
L2 loss
discriminator

FrankenGAN


input:
façade shape
1st step:
window/door
layout
2nd step:
texture
3rd step:
sem. labels
BicycleGAN
BicycleGAN
BicycleGAN
separate
training sets:

Progressive GAN

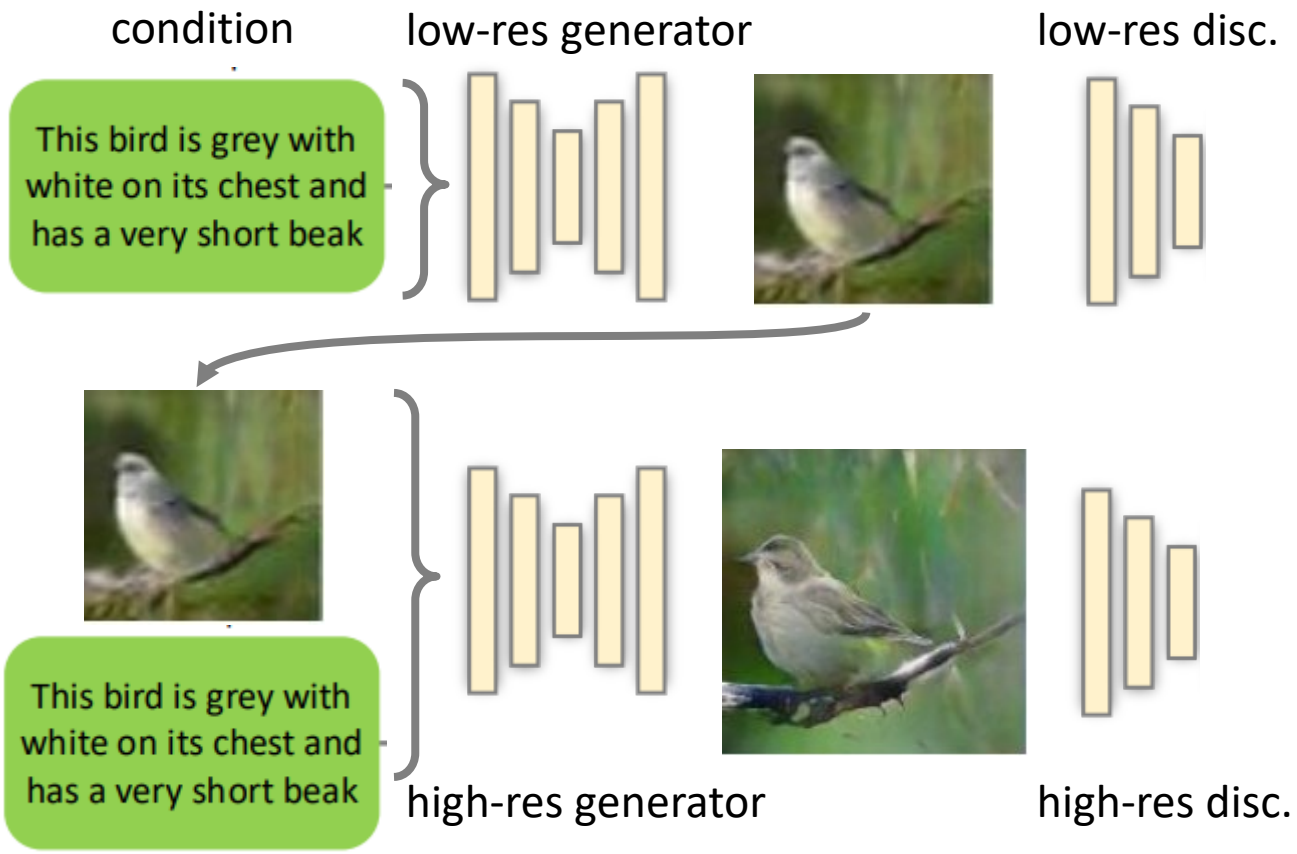
- Resolution is increased progressively during training
- Also other tricks like using minibatch statistics and normalizing feature vectors



Karras et al., *Progressive Growing of GANs for Improved Quality, Stability, and Variation*, ICLR 2018

StackGAN

Condition does not have to be an image



This flower has white petals with a yellow tip and a yellow pistil

A large bird has large thighs and large wings that have white wingbars

Disentanglement

z

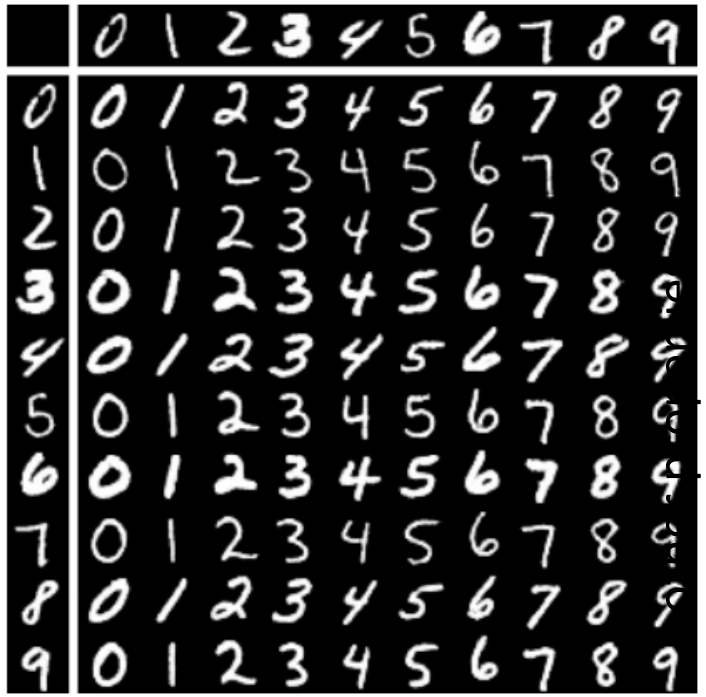
Entangled: different properties may be mixed up over all dimensions

z_a z_b $\cdots$

Disentangled: different properties are in different dimensions

specified property: number

other properties



specified property: character

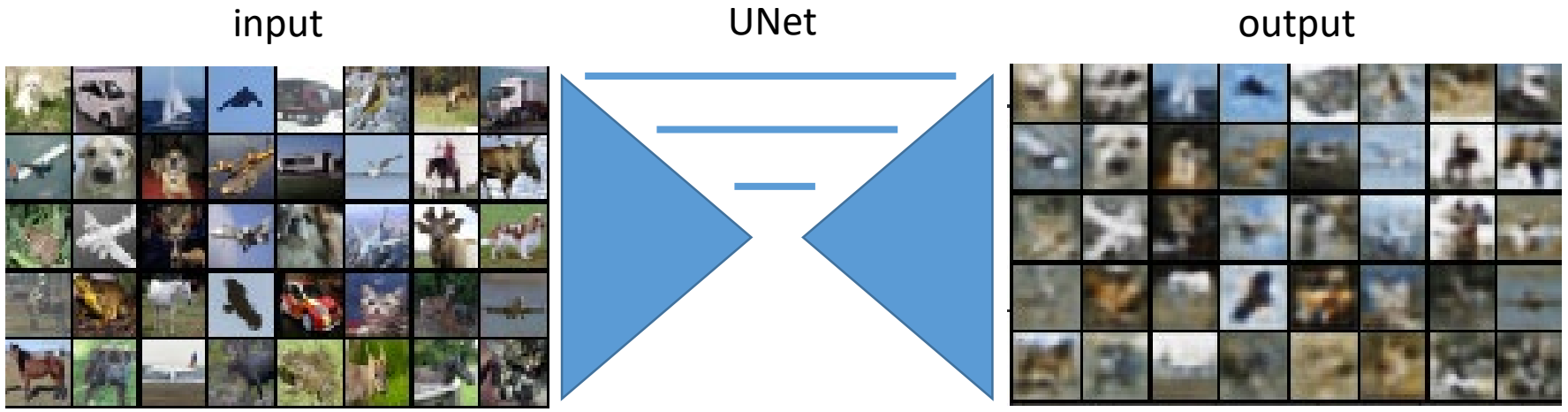
other properties

Mathieu et al., *Disentangling factors of variation in deep representations using adversarial training*, NIPS 2016

Attention and Gray Box Learning

Attention in Deep Learning

target: horizontal mirroring

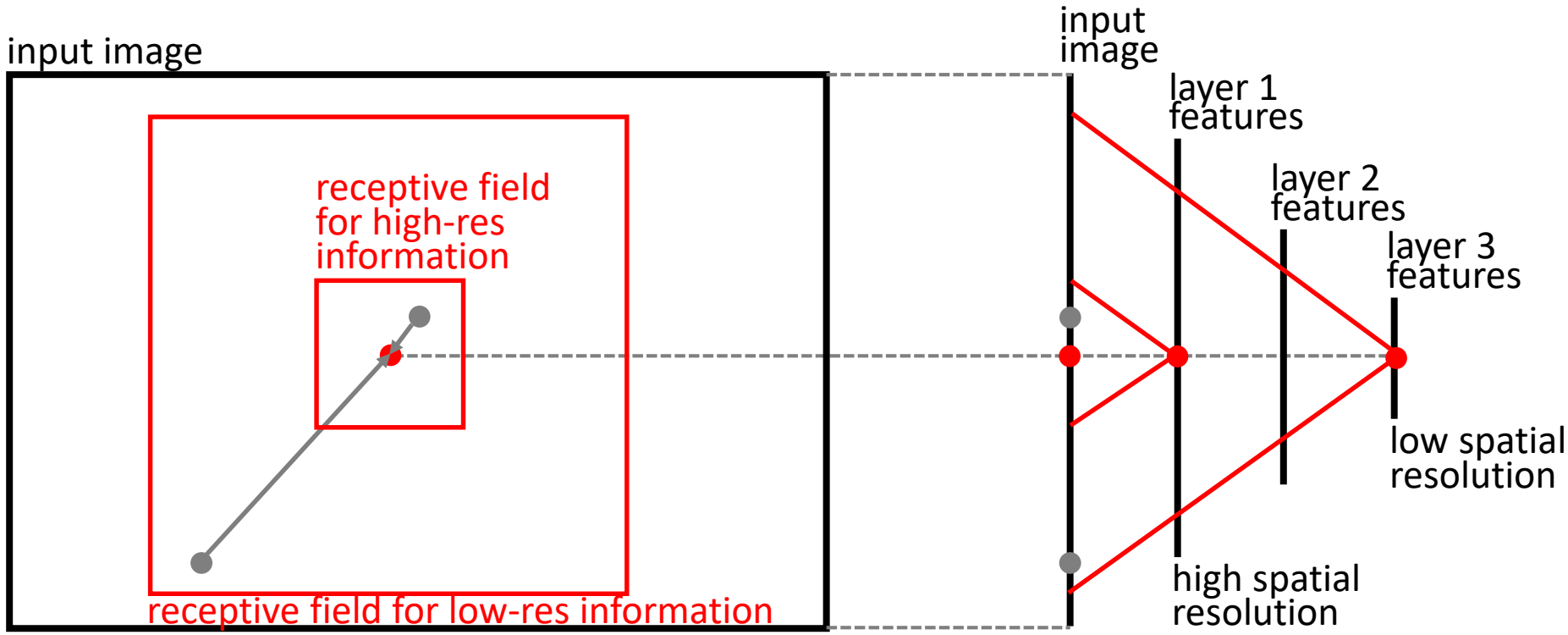


Why is this hard for the network?

1) Locality of convolutions
2) Driven only by data from shallower layers (no semantics)

Attention in Deep Learning

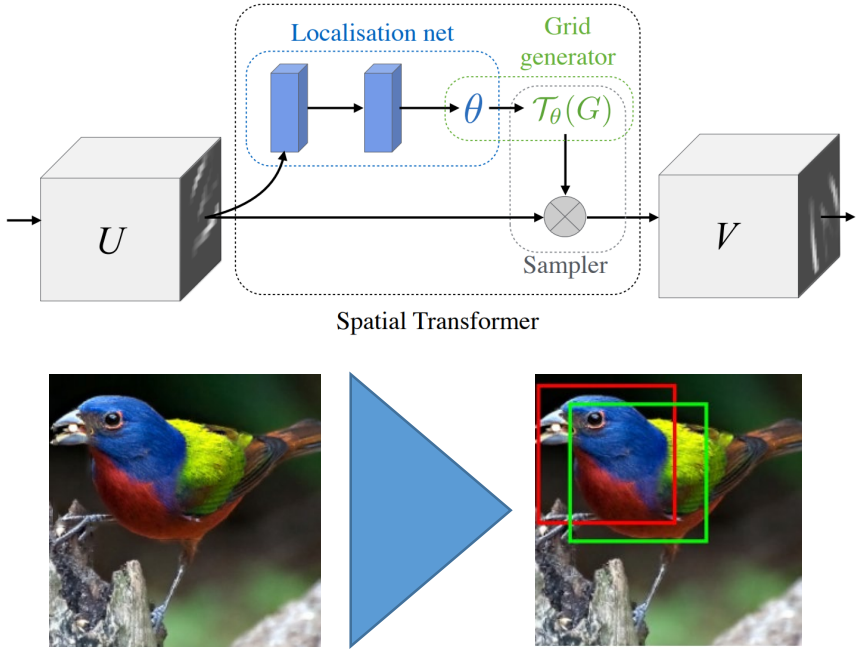
Problem: architecture constrains information flow. For example, in a typical CNN, at a given image location (red), information about other image locations (grey) is available in a resolution that depends on the spatial distance.



Attention Based on Semantics

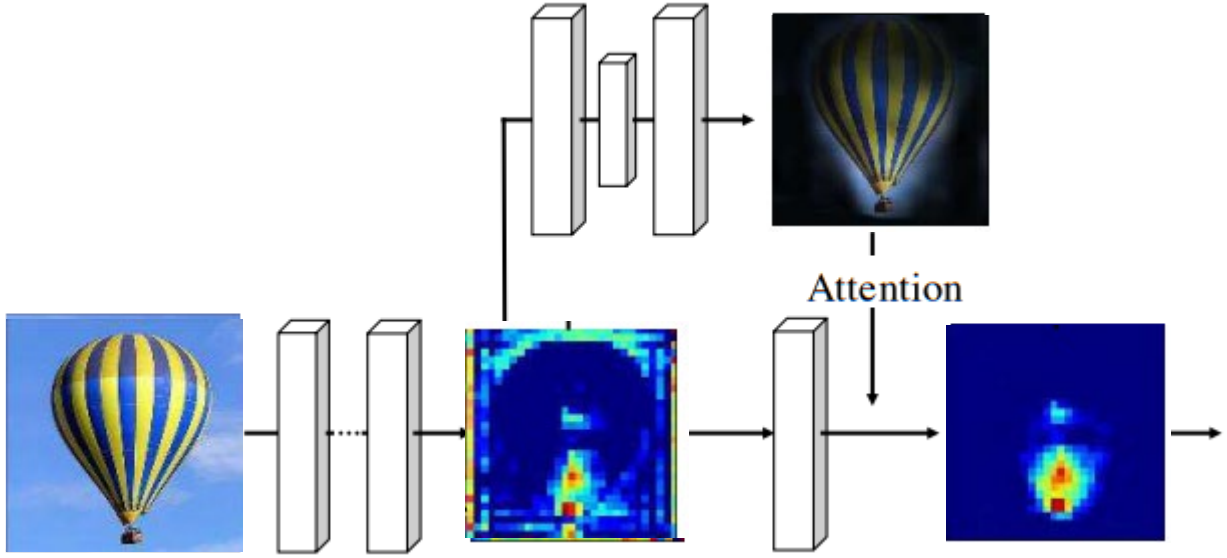
Idea: use higher-level semantics to select relevant information

Spatial Transformer Networks



Jaderberg et al., *Spatial Transformer Networks*, NIPS 2015

Residual Attention Network for Image Classification



Wang et al., *Residual Attention Network for Image Classification*, CVPR 2017

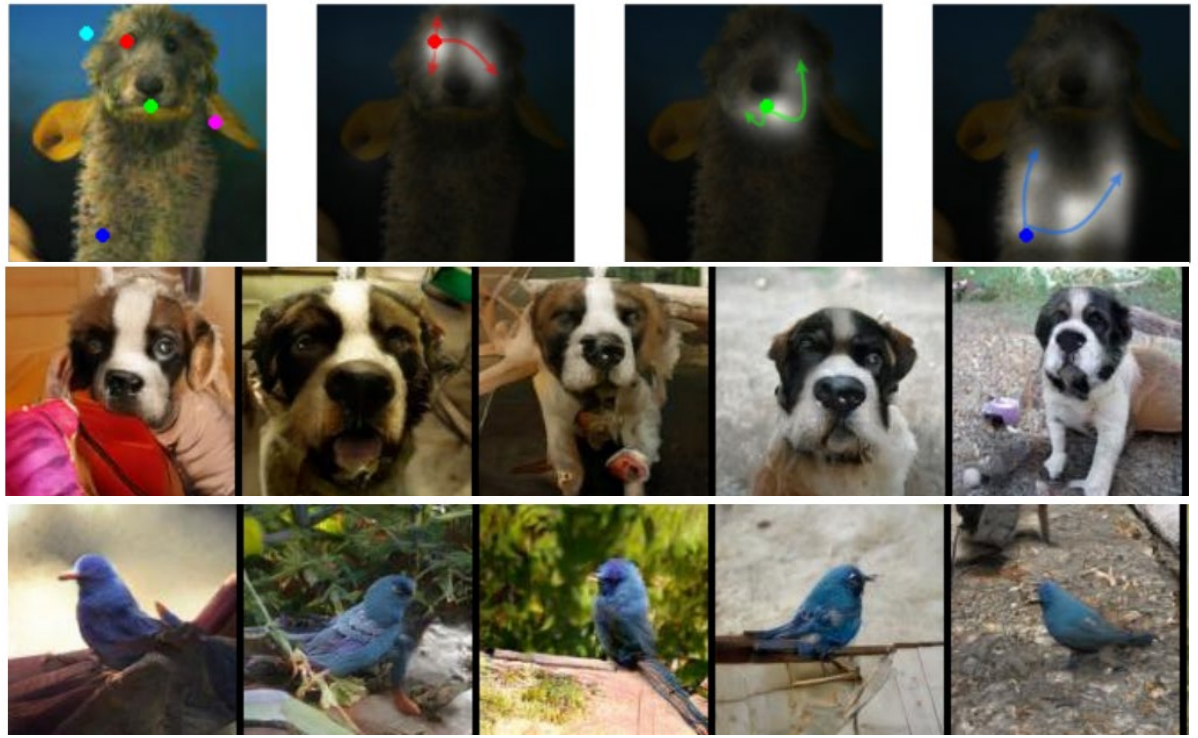
Attention to Distant Details

Idea: gather information from distant details based on their features

Non-local Neural Networks

Wang et al., *Non-local Neural Networks*, CVPR 2018

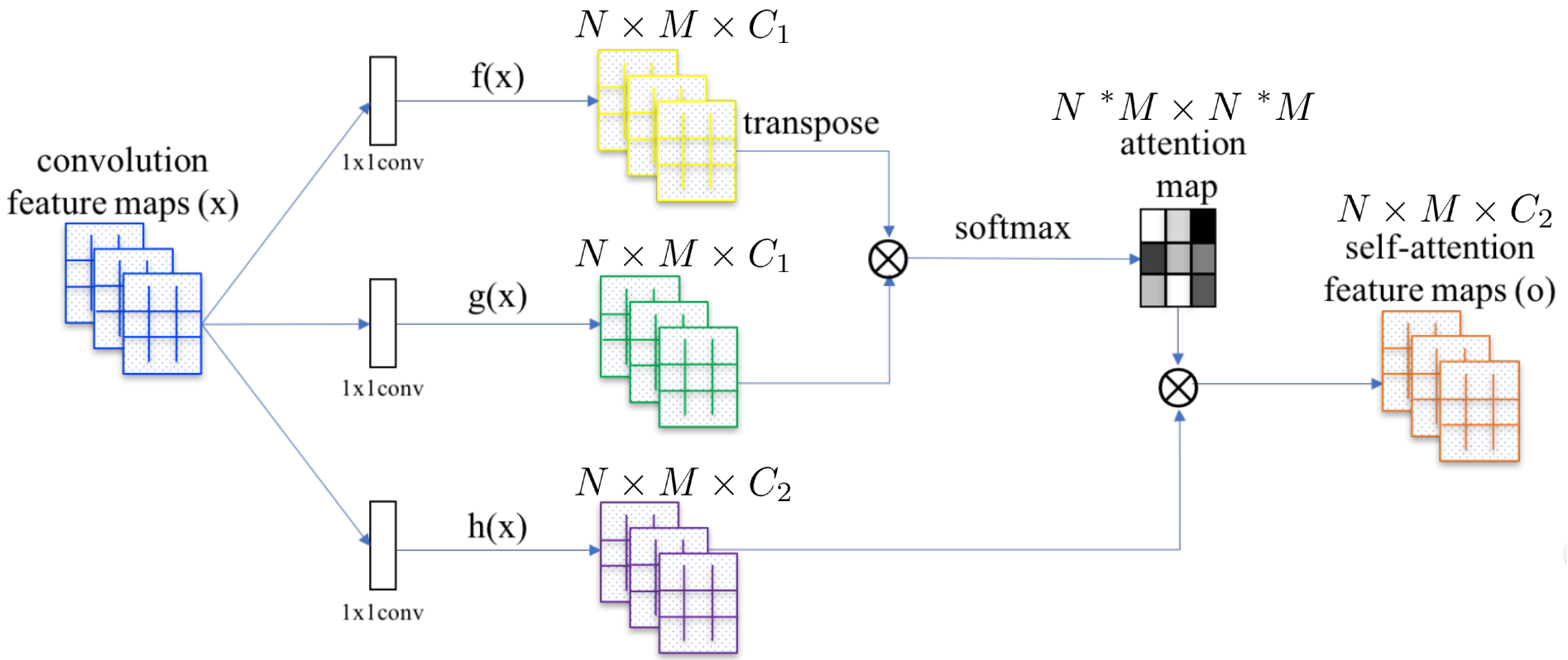
Attention GAN



Zhang et al., *Self-Attention Generative Adversarial Networks*, CVPR 2018

Attention to Distant Details

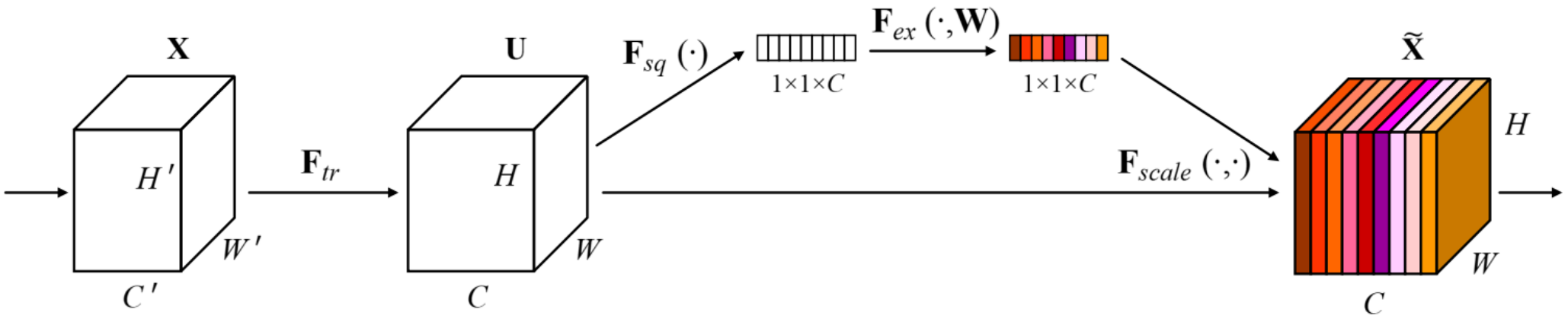
Idea: gather information from distant details based on their features



Zhang et al., *Self-Attention Generative Adversarial Networks*, CVPR 2018

Squeeze and Excitation: Attention over Channels

Idea: weigh (emphasize and suppress) channels based on global information

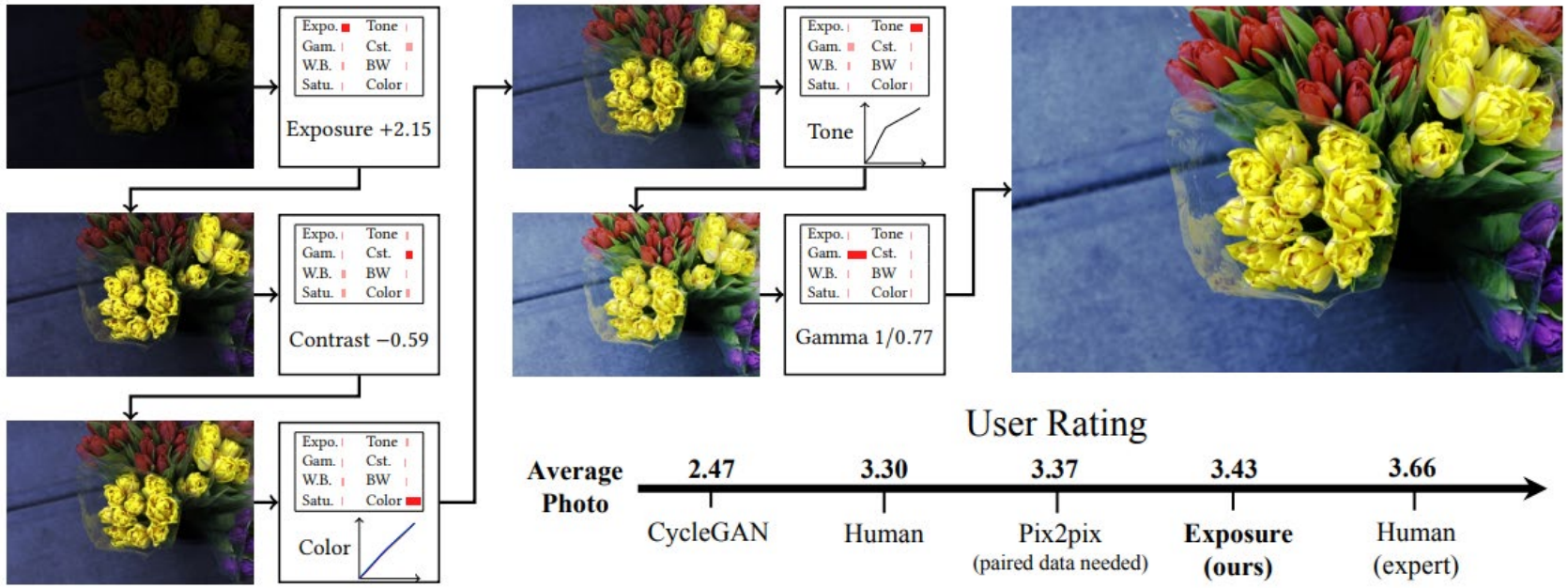


Hu et al., *Squeeze-and-Excitation Networks*, CVPR 2018

Gray Box Learning

Problem: Most networks are black boxes.

Idea: Regress parameters for a small set of well-known operations.

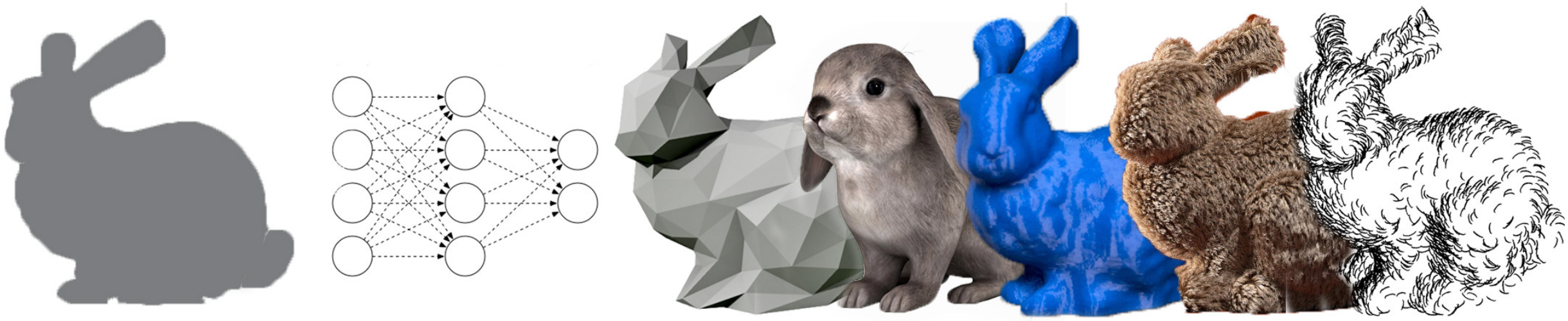


Hu et al., *Exposure: A White-Box Photo Post-Processing Framework*, Siggraph 2018

Summary

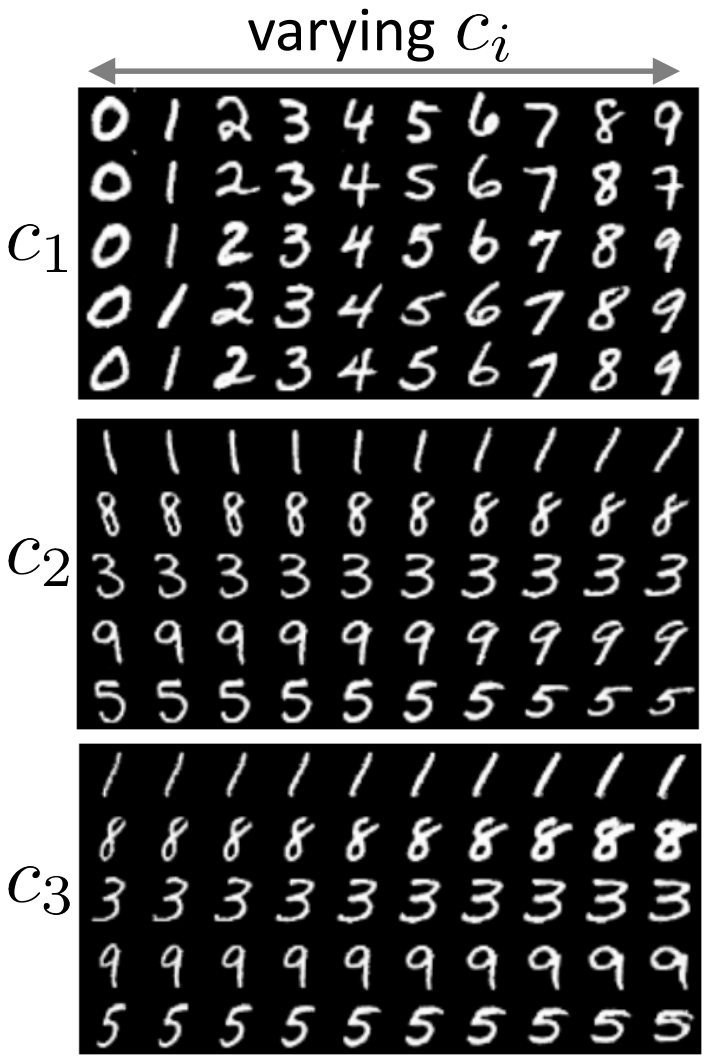
- Common Architecture Elements
  (Dilated Convolution, Grouped Convolutions)
- Deep Features
  (Autoencoders, Transfer Learning, One-shot Learning, Style Transfer)
- Adversarial Image Generation
  (GANs, CGANs)
- Interesting Trends
  (Attention, “Gray Box” Learning)

Course Information (slides/code/comments)



http://geometry.cs.ucl.ac.uk/creativeai/

InfoGAN



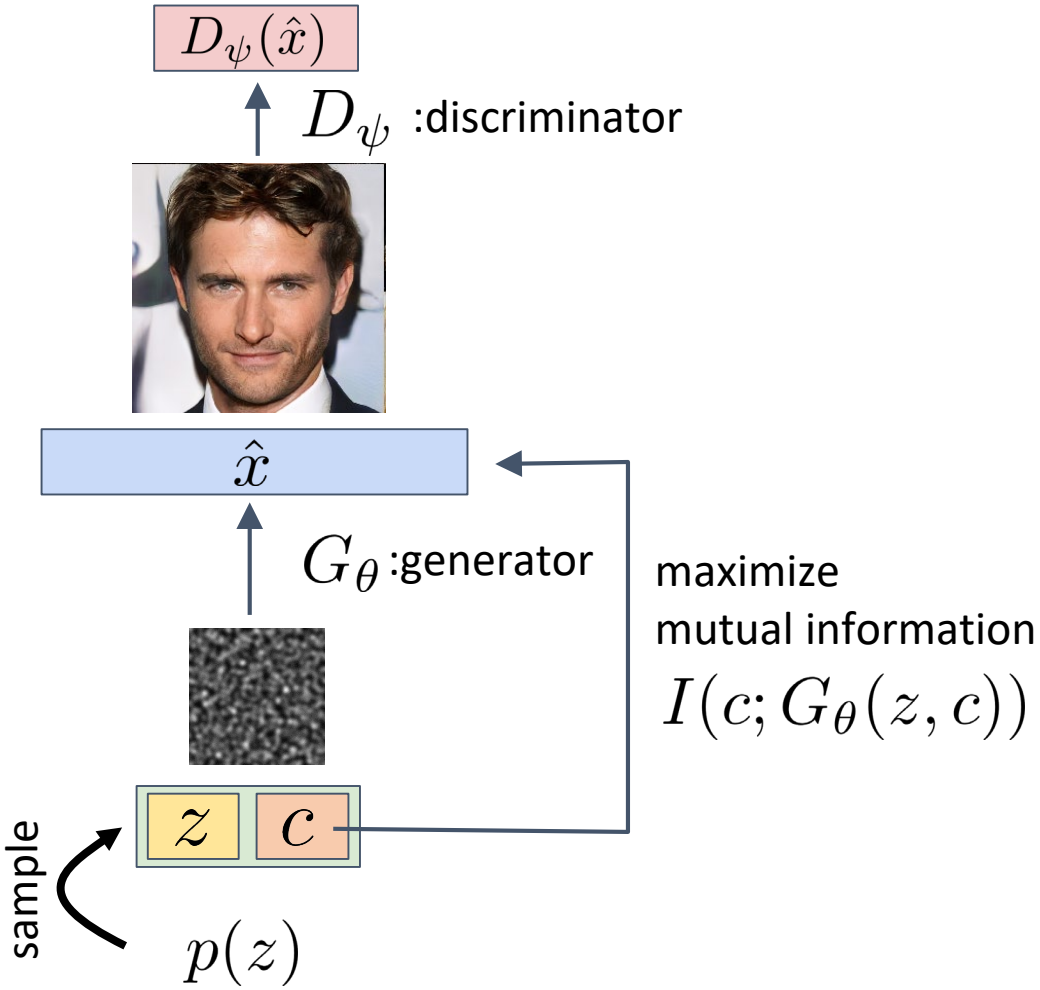


Image Credit: *InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets*, Chen et al.

CreativeAI

3D (Geometric) Domain

Niloy Mitra
UCL

Iasonas Kokkinos
UCL

Paul Guerrero
UCL

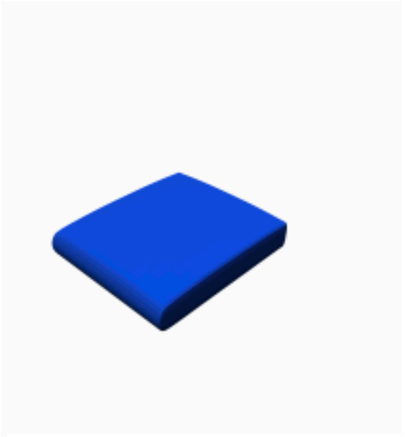
Nils Thuerey
TUM

Tobias Ritschel
UCL

Timetable

			Niloy	Paul	Nils
Theory + Basics	Introduction	2:15 pm	X	X	X
	Machine Learning Basics	~ 2:25 pm	X		
	Neural Network Basics	~ 2:55 pm			X
	Feature Visualization	~ 3:25 pm		X	
	Alternatives to Direct Supervision	~ 3:35 pm		X	
	15 min. break				
State of the Art	Image Domains	4:15 pm		X	
	3D Domains	~ 4:40 pm	X		
	Motion and Physics	~ 5:05 pm			X
	Discussion	~ 5:30 pm	X	X	X

Application #1: 3D Modeling



Deep neural network predicts the **next best part** to add and its **position** to enable non-expert users to create novel shapes.

[Sung et al. 2017]

Application #1: 3D Modeling

Deep neural network predicts the **next best part** to add and its **position** to enable non-expert users to create novel shapes.

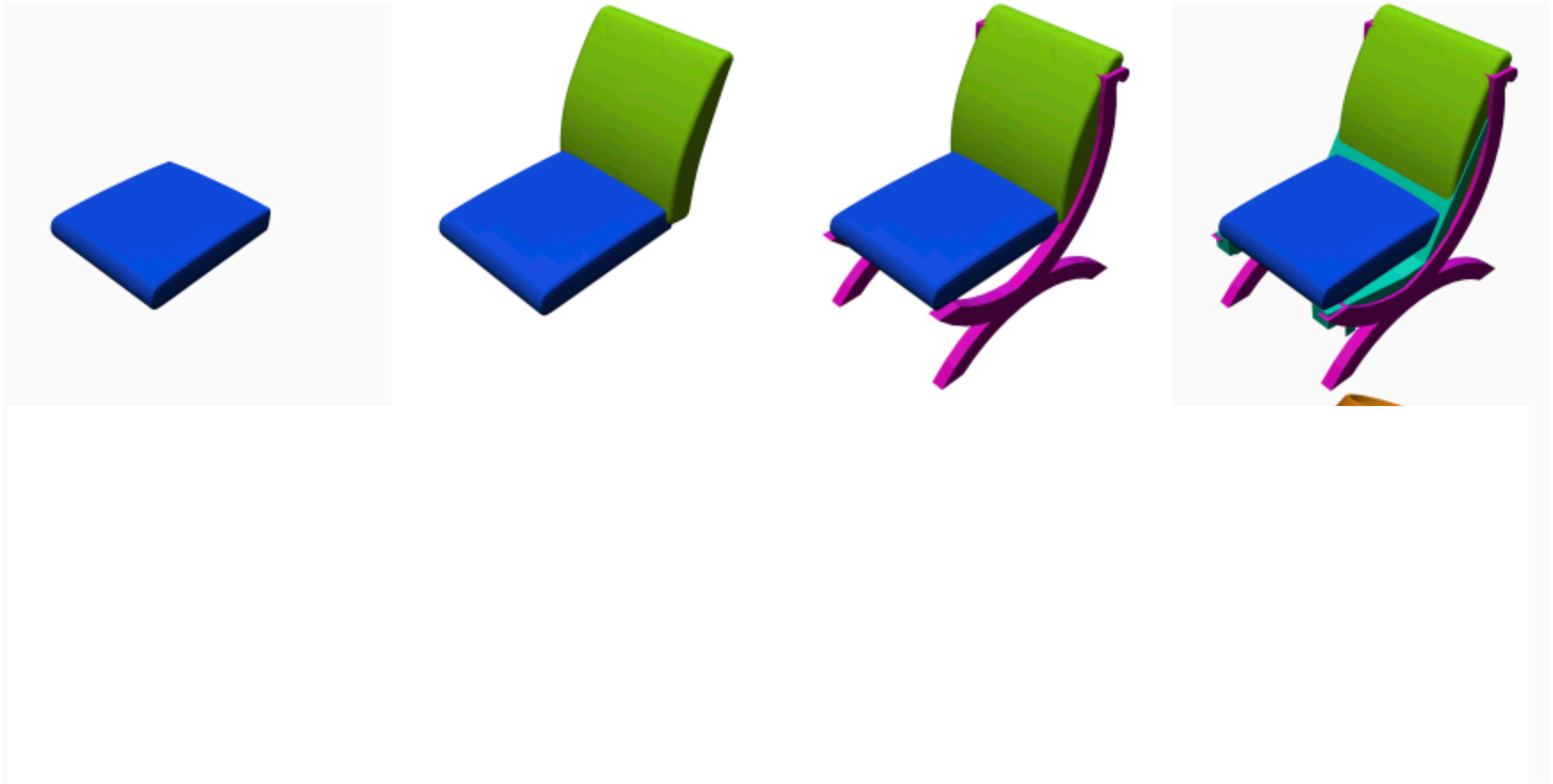
[Sung et al. 2017]

Application #1: 3D Modeling

Deep neural network predicts the **next best part** to add and its **position** to enable non-expert users to create novel shapes.

[Sung et al. 2017]

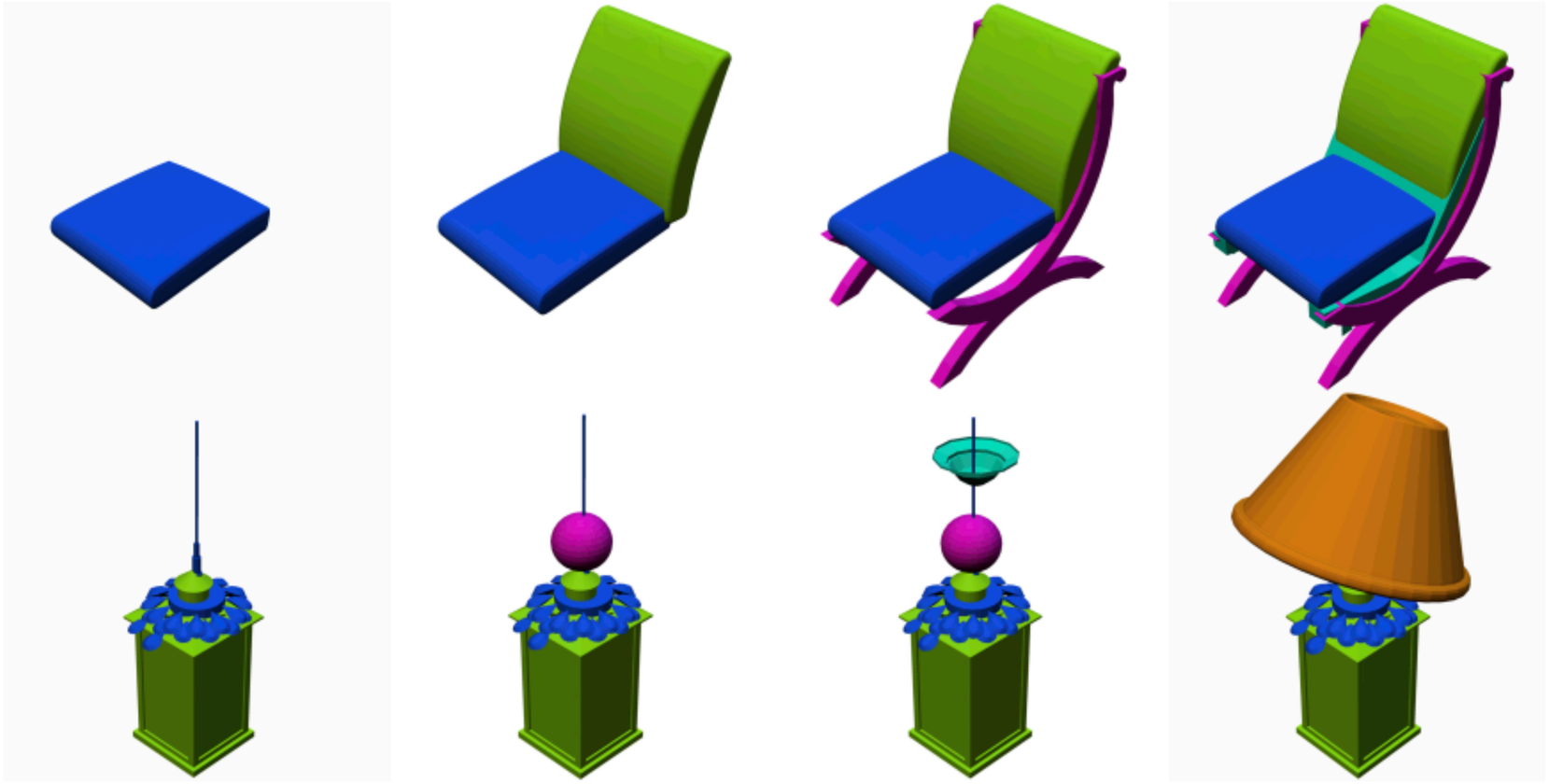
Application #1: 3D Modeling



Deep neural network predicts the **next best part** to add and its **position** to enable non-expert users to create novel shapes.

[Sung et al. 2017]

Application #1: 3D Modeling



Deep neural network predicts the **next best part** to add and its **position** to enable non-expert users to create novel shapes.

[Sung et al. 2017]

Application #2: Image Understanding

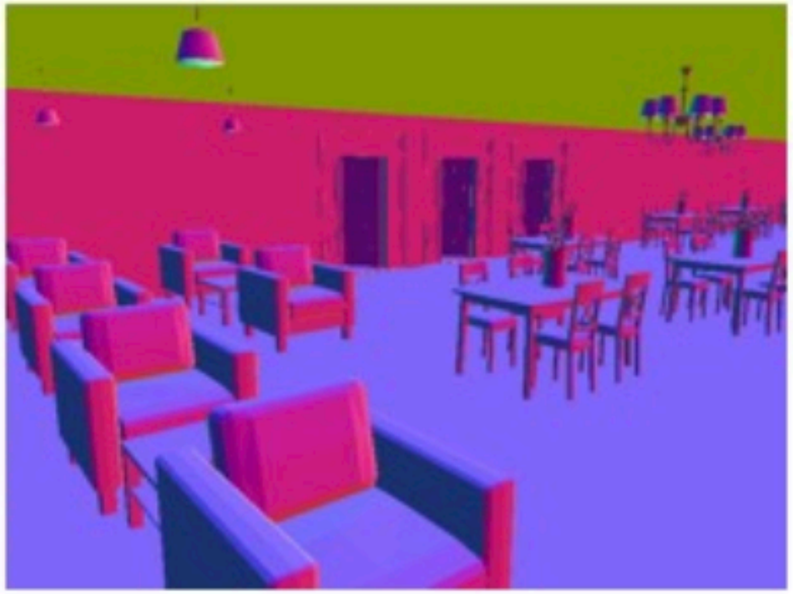
understanding 3D shapes can benefit image understanding

[Zhang et al. 2017]

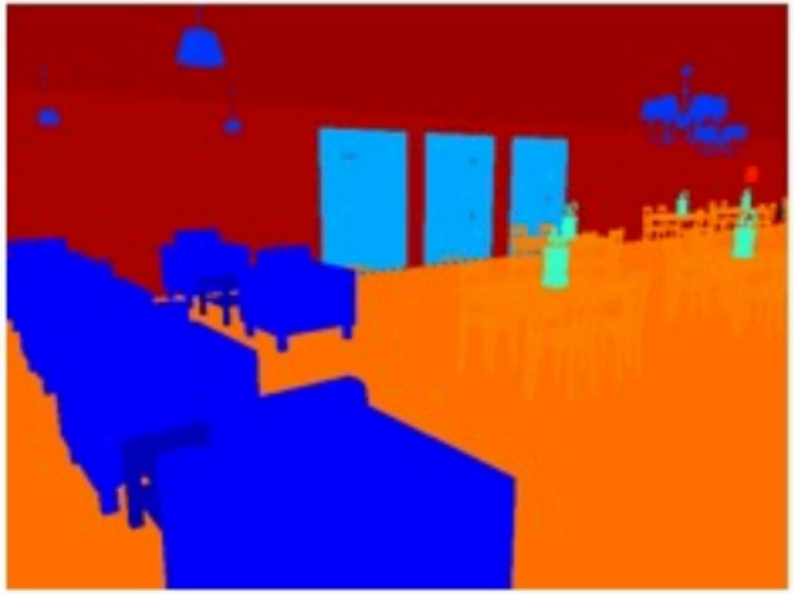
Application #2: Image Understanding

understanding 3D shapes can benefit image understanding

Physically based Rendering



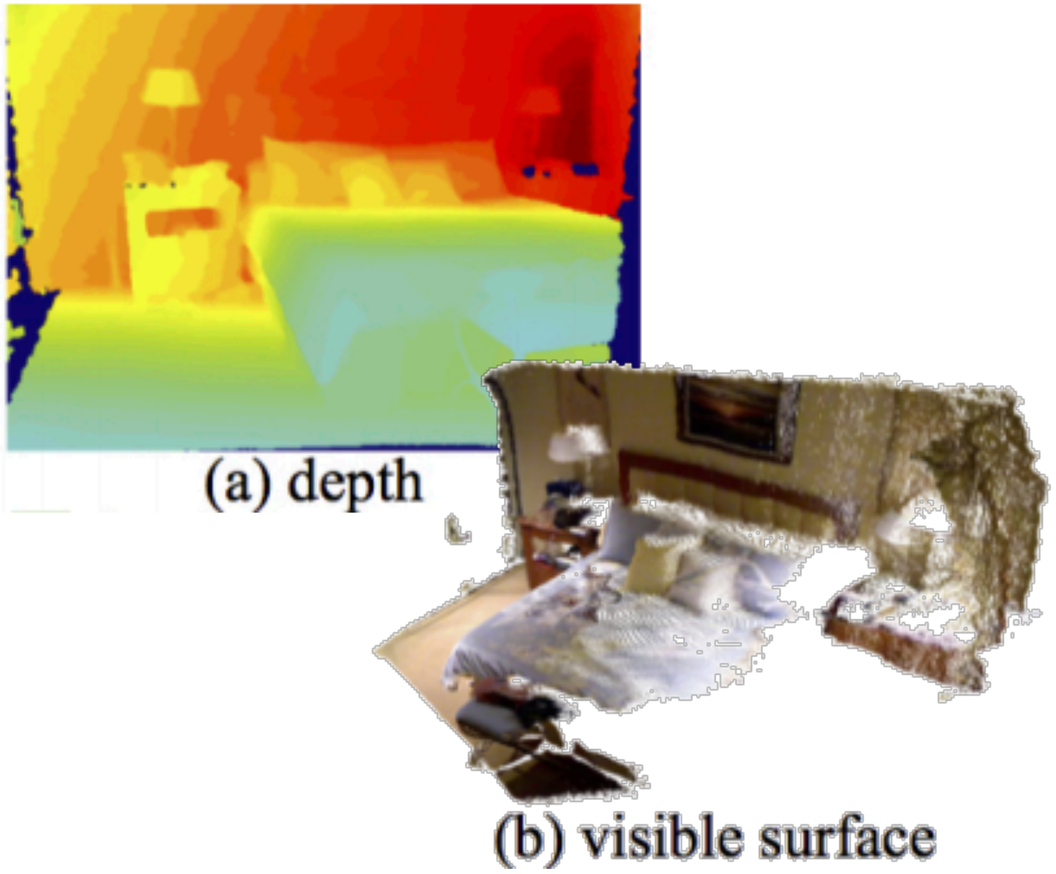
Surface Normal

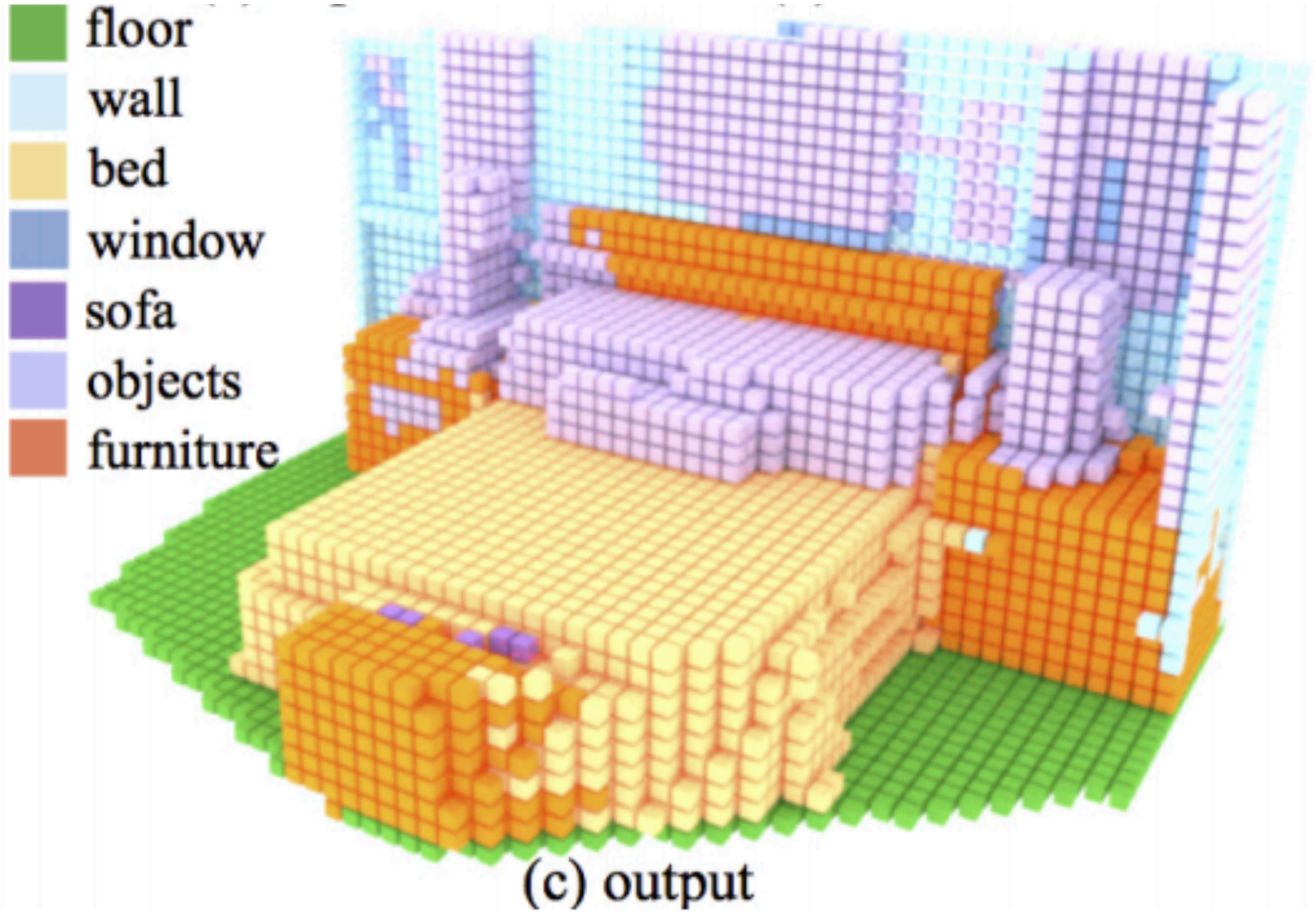


Semantic Segmentation

[Zhang et al. 2017]

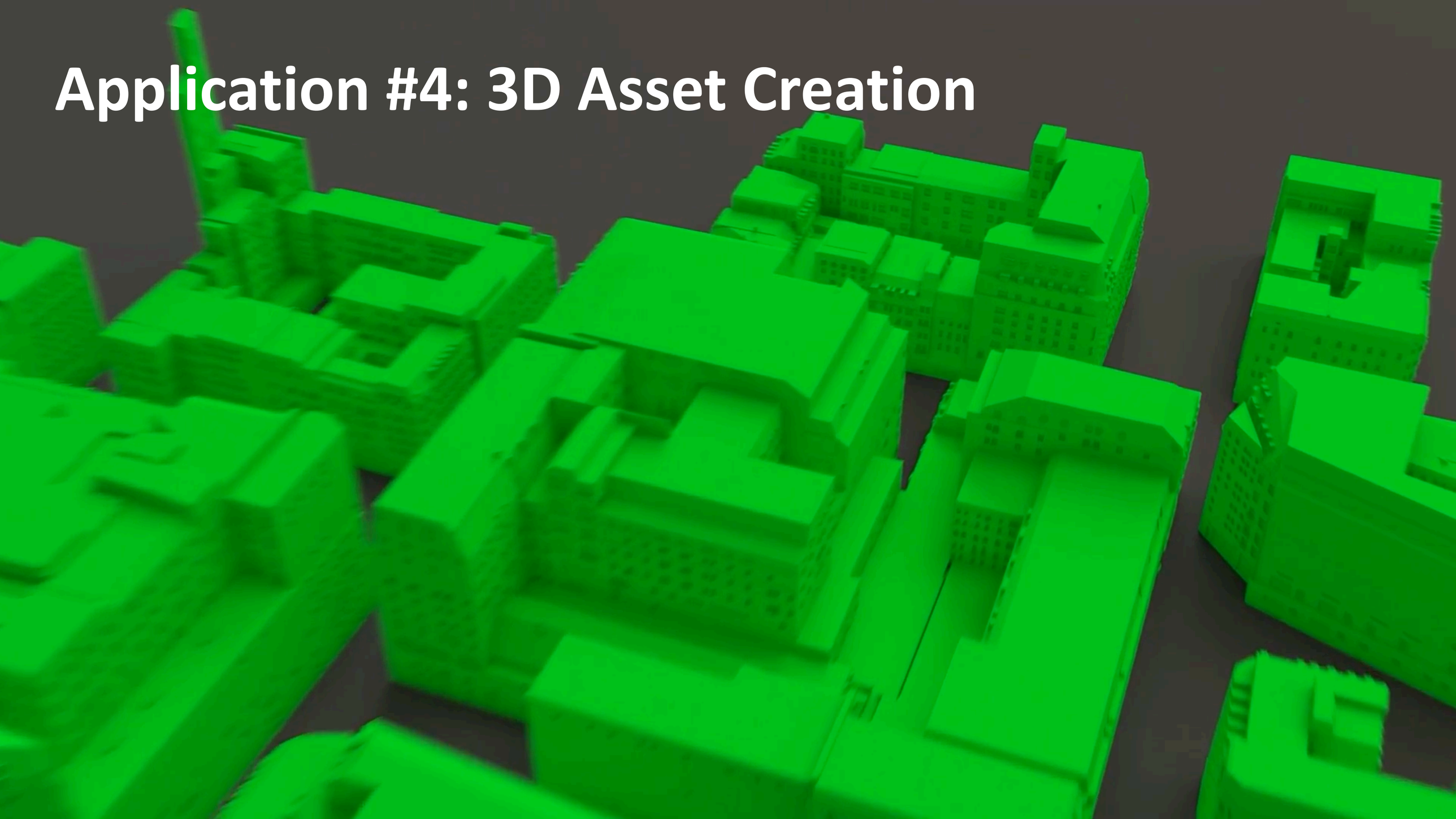
Application #3: Semantic Scene Understanding



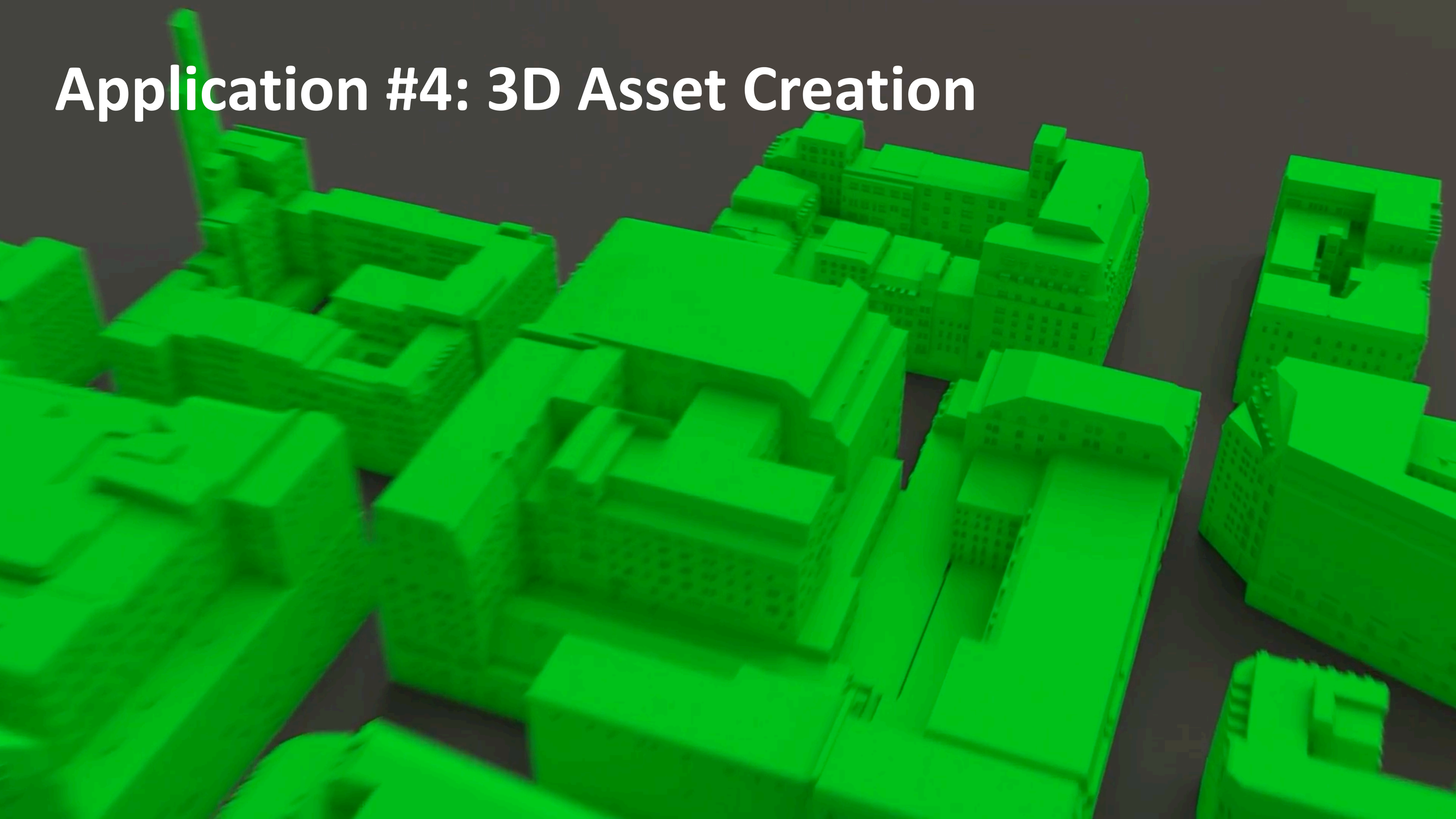


[Song et al. 2017]

Application #4: 3D Asset Creation



Application #4: 3D Asset Creation



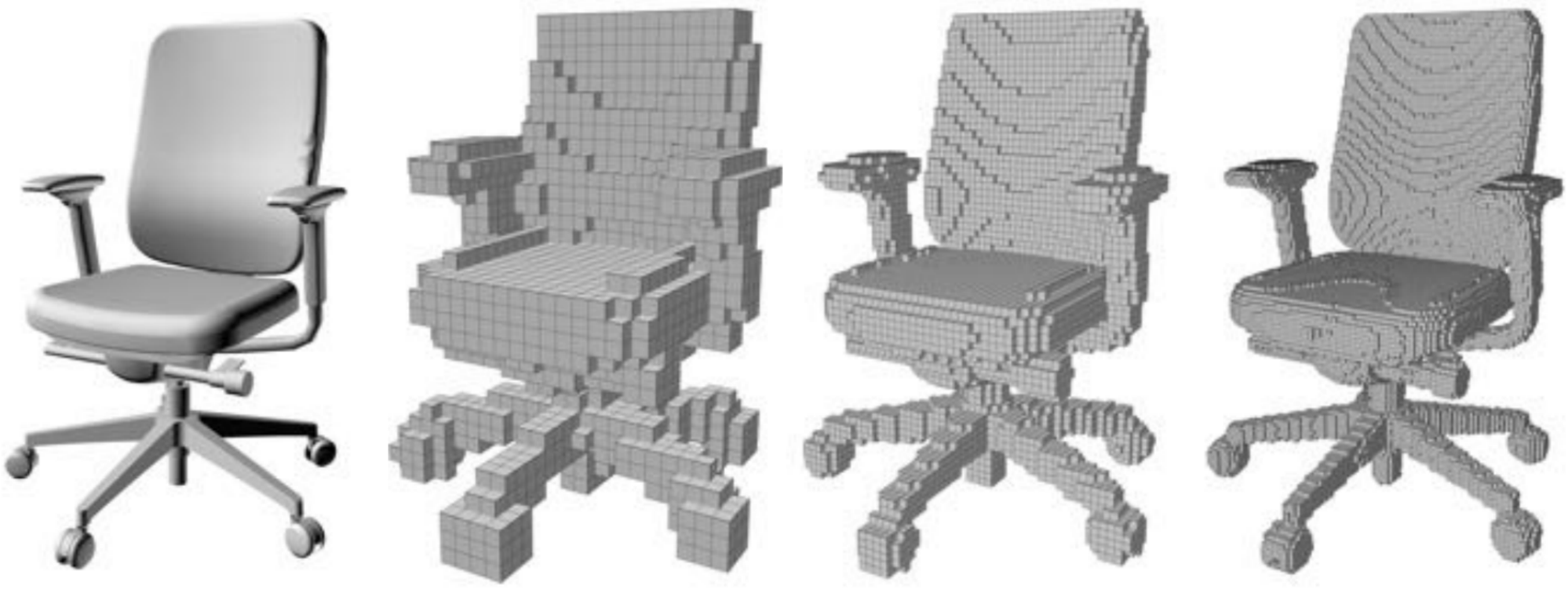
What’s Different in 3D?

What's Different in 3D?

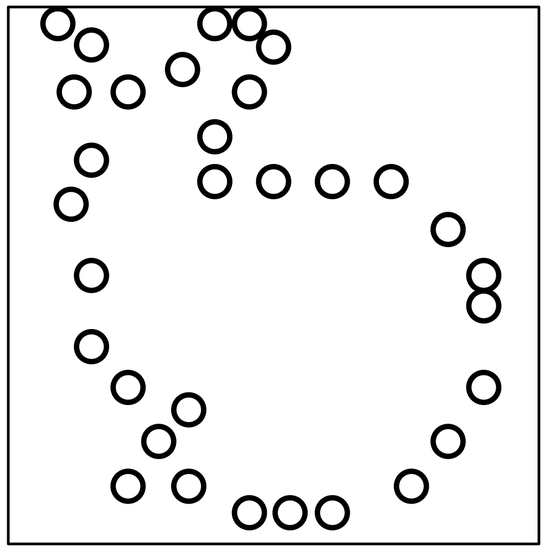
- Number of Voxels grows as $O(n^3)$ versus occupied **surface** $O(n^2)$

What's Different in 3D?

- Number of Voxels grows as $O(n^3)$ versus occupied **surface** $O(n^2)$

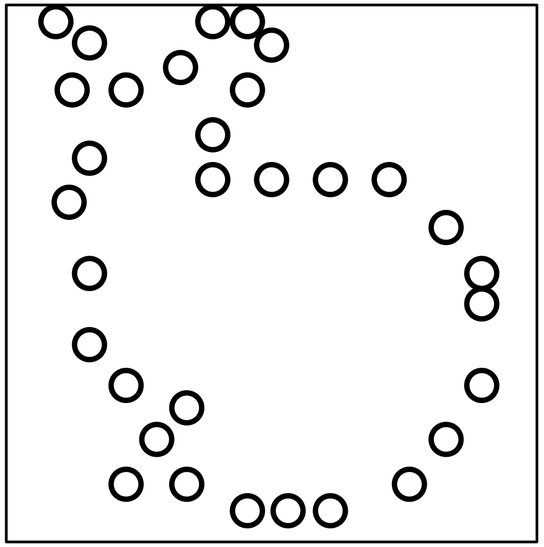


Data Representation .. Many Possibilities!

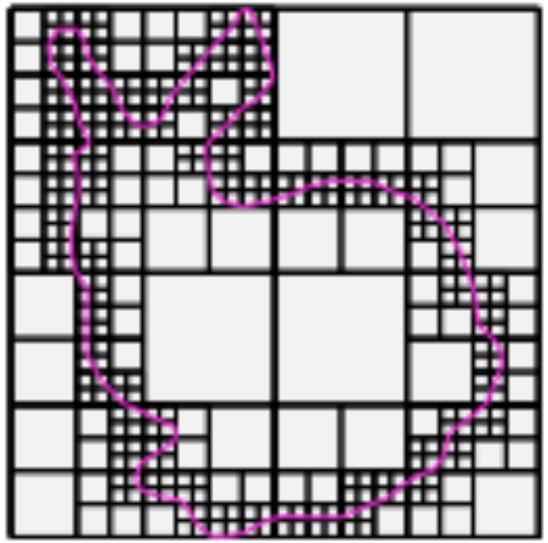


points

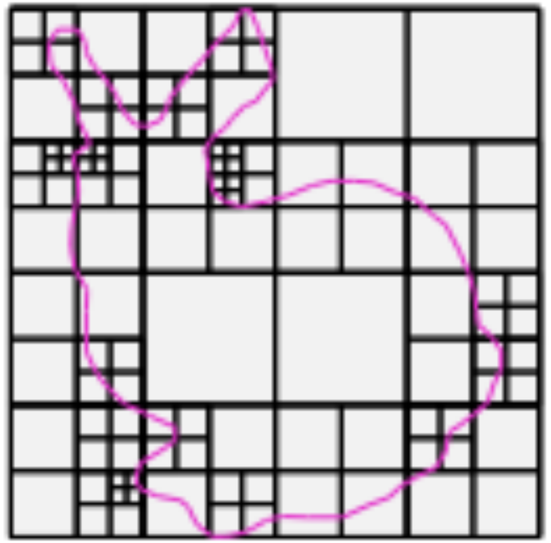
Data Representation .. Many Possibilities!



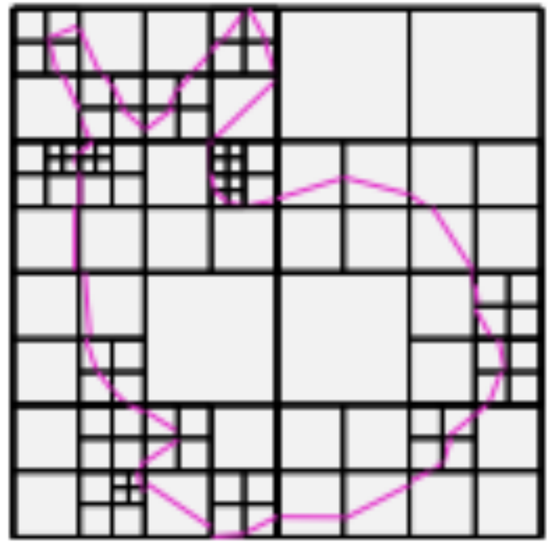
points



voxels



cells



patches

Challenges

Challenges

1. Representation

Challenges

1. Representation

2. **Neighborhood** information
   - who are the neighbouring elements
   - how are the elements ordered

Challenges

1. Representation

2. **Neighborhood** information
   - who are the neighbouring elements
   - how are the elements ordered

3. **Extrinsic** versus **intrinsic** representation

Challenges

1. Representation

2. **Neighborhood** information
   - who are the neighbouring elements
   - how are the elements ordered

3. **Extrinsic** versus **intrinsic** representation

4. Simplicity versus memory/runtime tradeoff

Representation for 3D

- Image-based
- Volumetric
- Surface-based
- Point-based

Representation for 3D

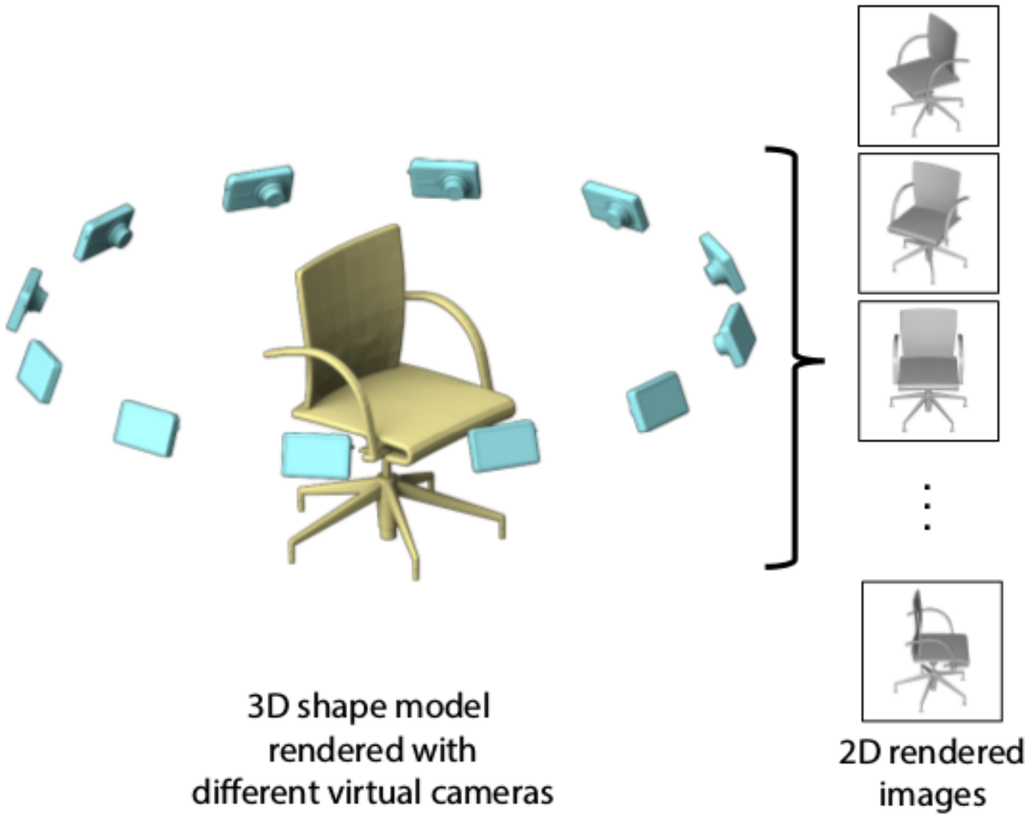
- **Image-based**
- Volumetric
- Surface-based
- Point-based

Representation for 3D: Multi-view CNN

3D shape model
rendered with
different virtual cameras

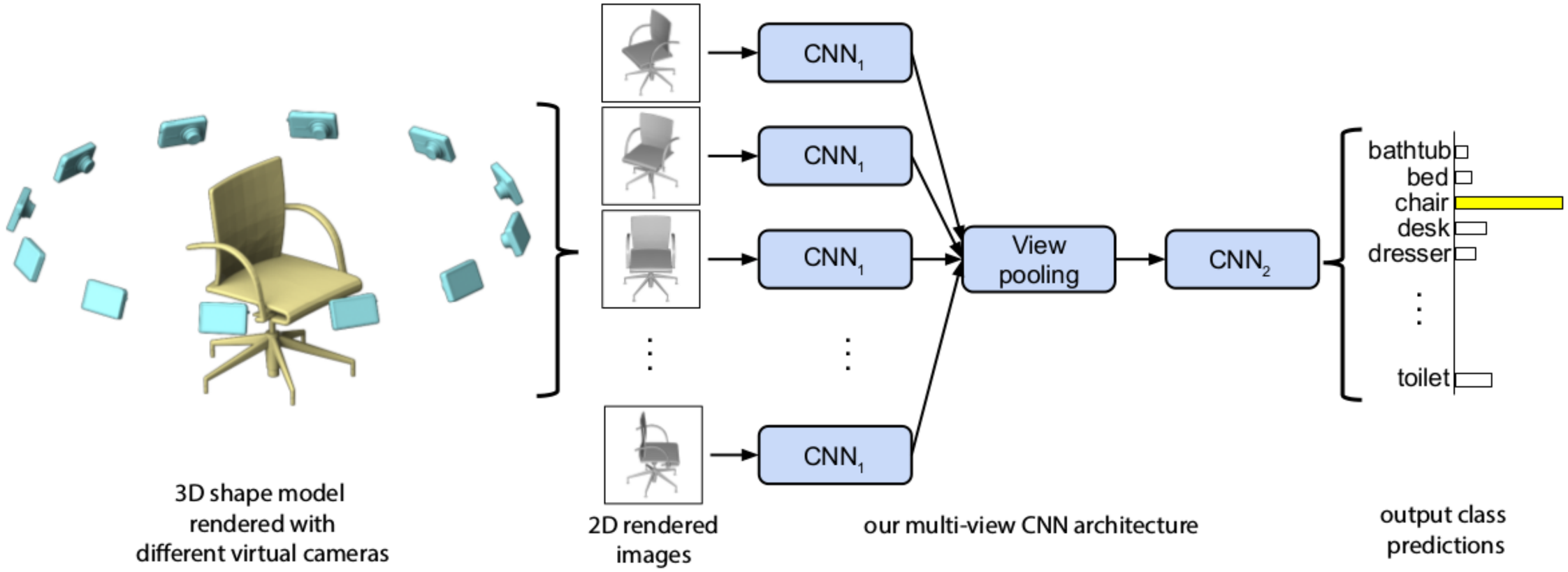
[Kalogerakis et al. 2015]

Representation for 3D: Multi-view CNN



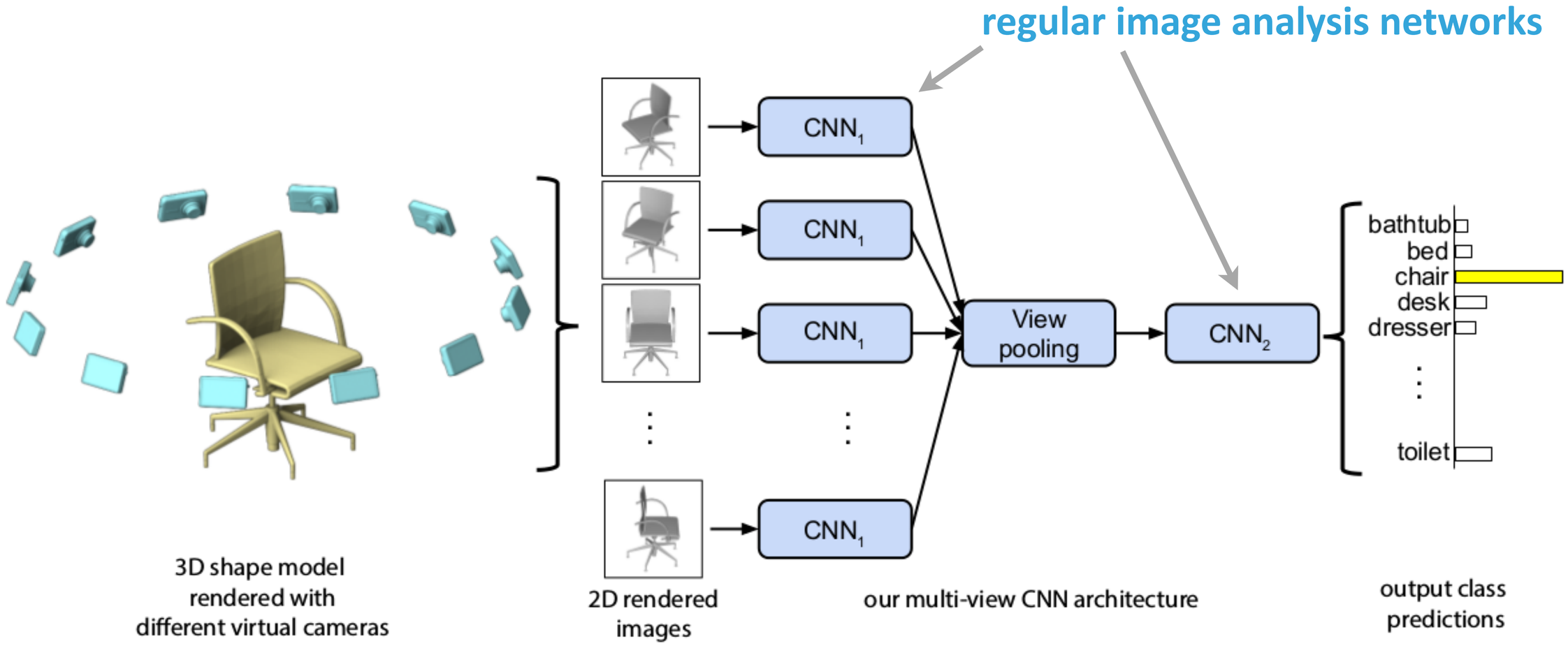
[Kalogerakis et al. 2015]

Representation for 3D: Multi-view CNN



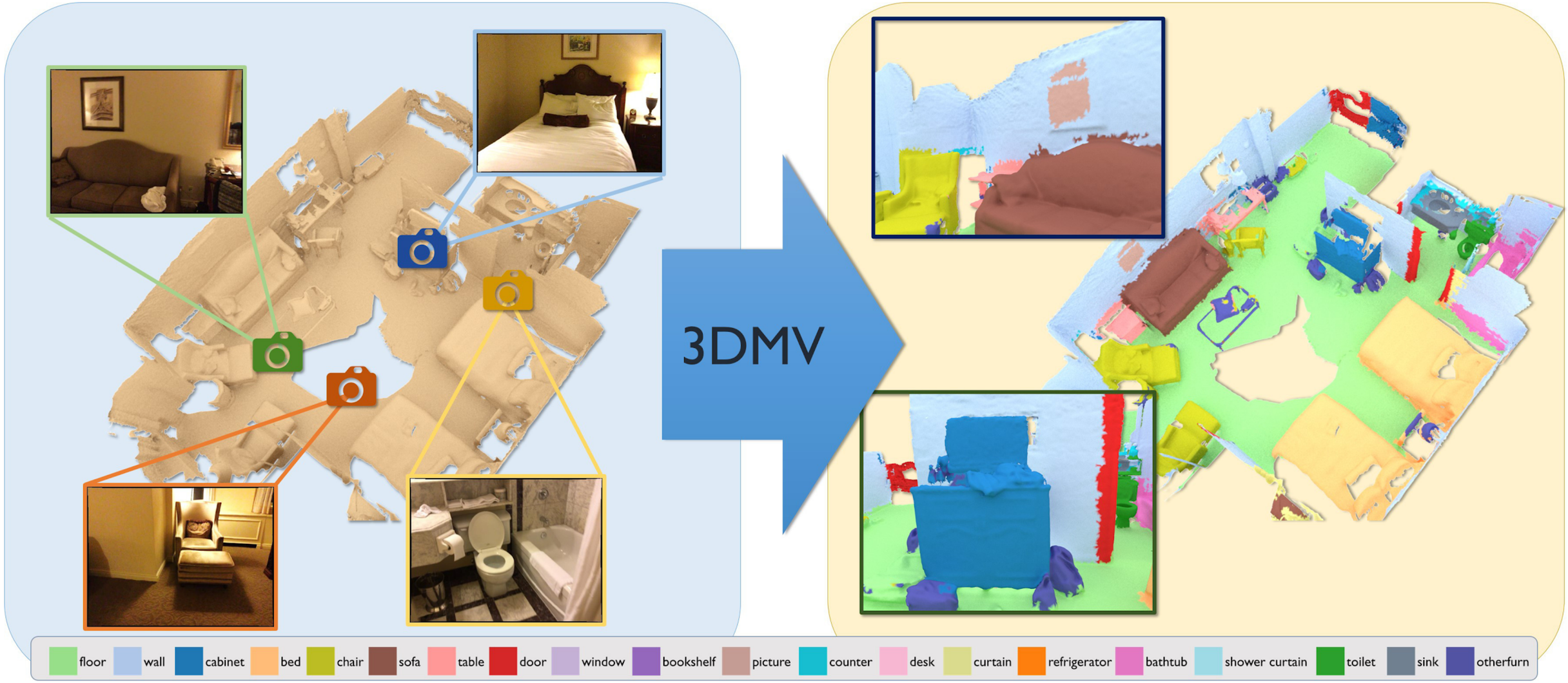
[Kalogerakis et al. 2015]

Representation for 3D: Multi-view CNN

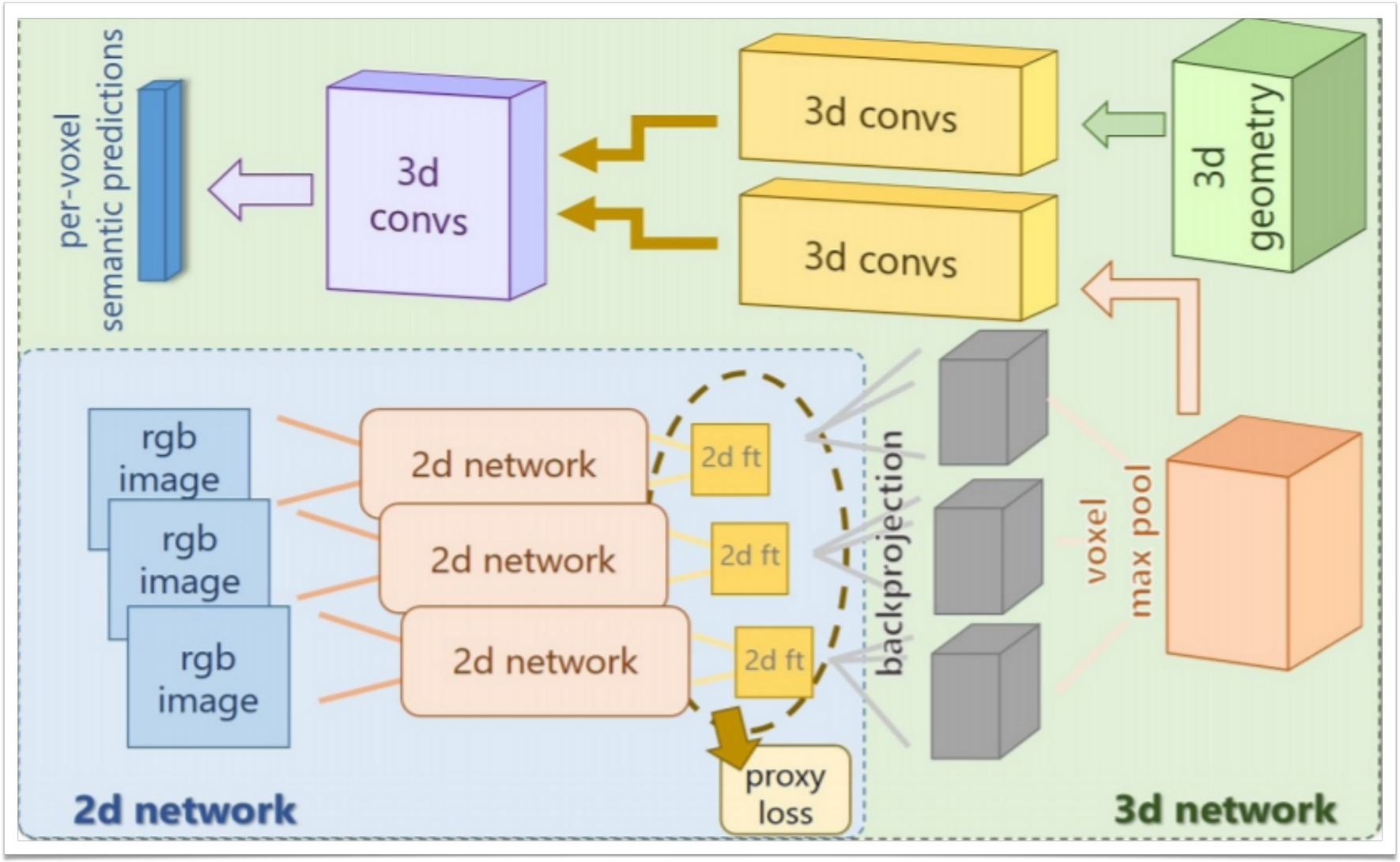


[Kalogerakis et al. 2015]

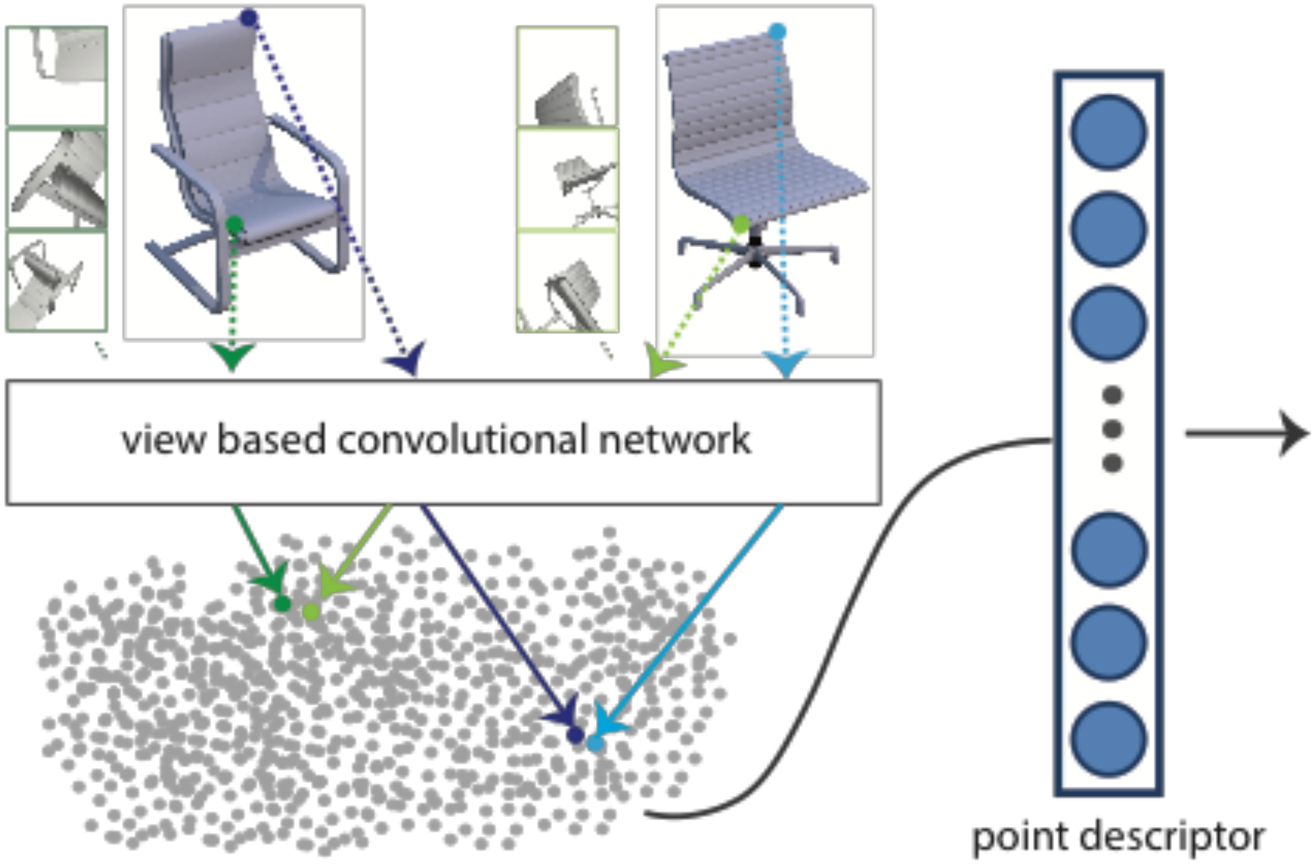
3DMV: Joint 3D-Multi-View Prediction for 3D Semantic Scene Segmentation



Integrating View Information

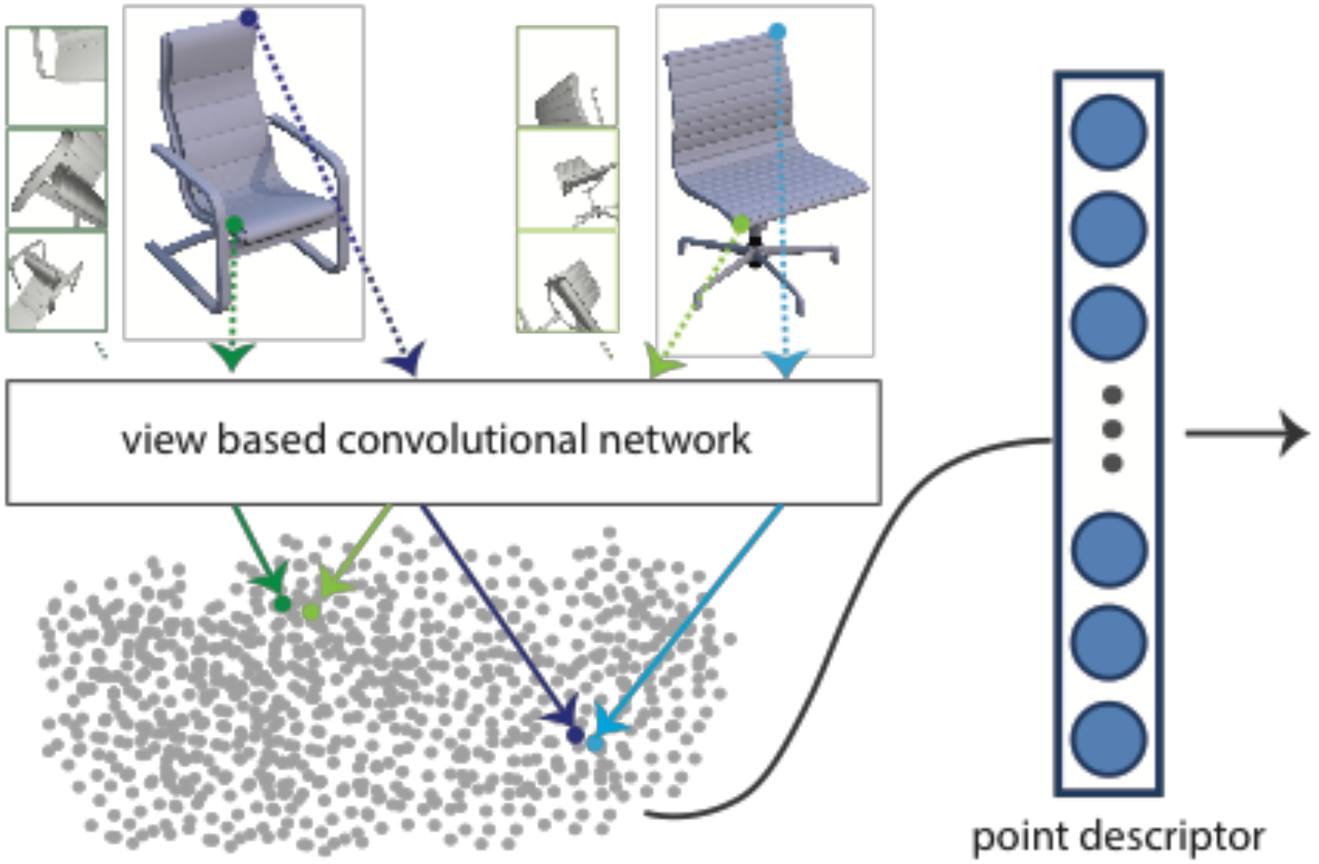


Representation for 3D: Local Multi-view CNN



[Huang et al. 2018]

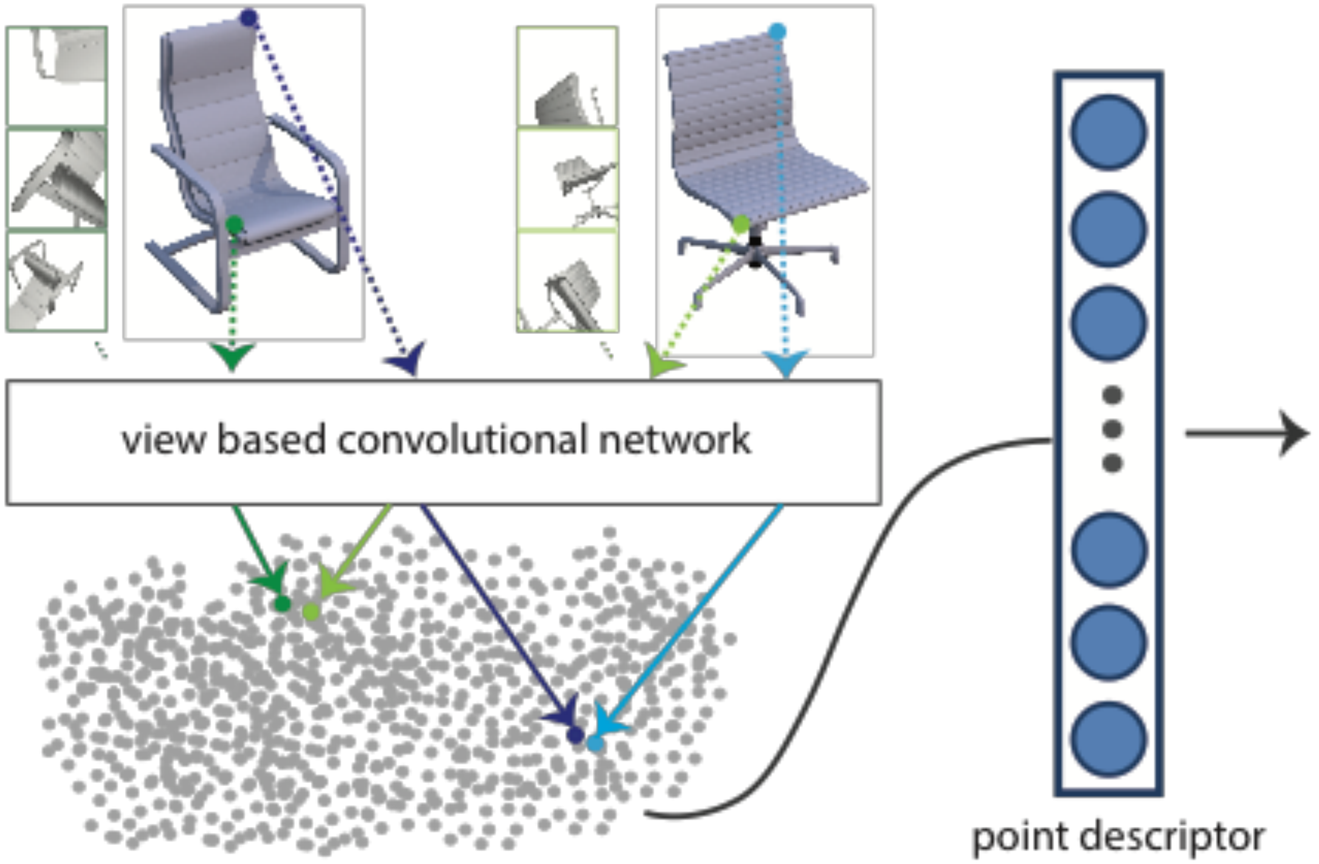
Representation for 3D: Local Multi-view CNN



Segmentation
Correspondence
Feature matching
Predicting semantic functions

[Huang et al. 2018]

Representation for 3D: Local Multi-view CNN

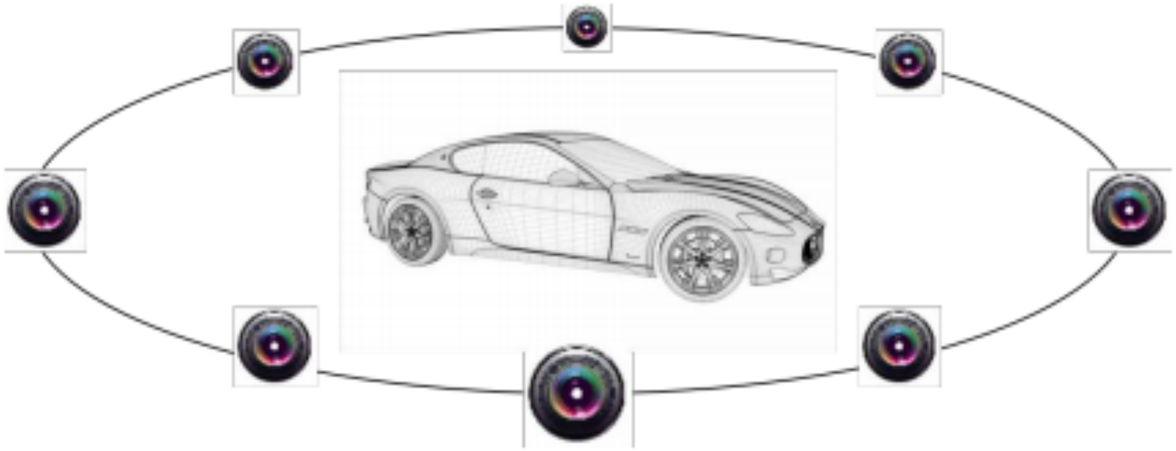


Segmentation
Correspondence
Feature matching
Predicting semantic functions

localized renderings for point-wise features

[Huang et al. 2018]

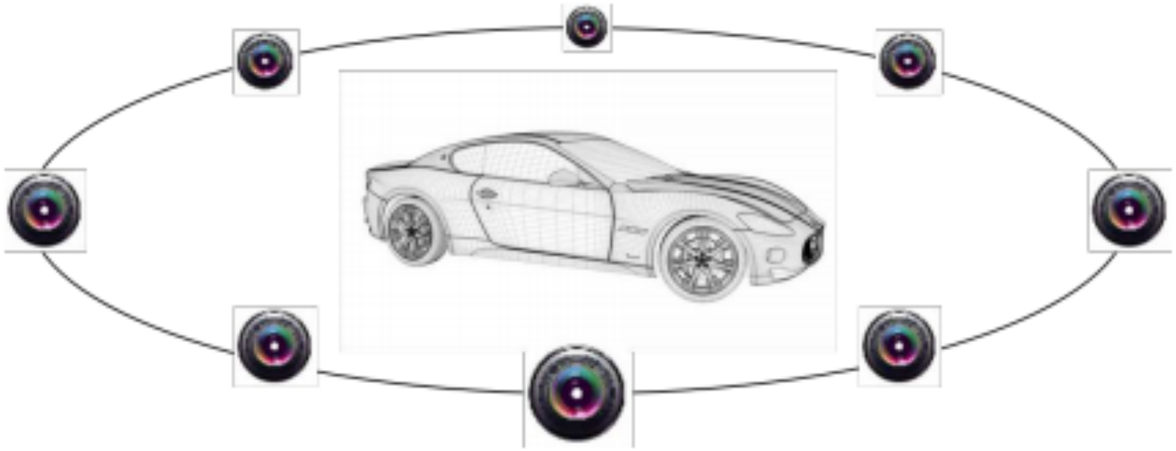
Tangent Convolutions



loses information due to occlusion

[Tatarchenko et al. 2018]

Tangent Convolutions

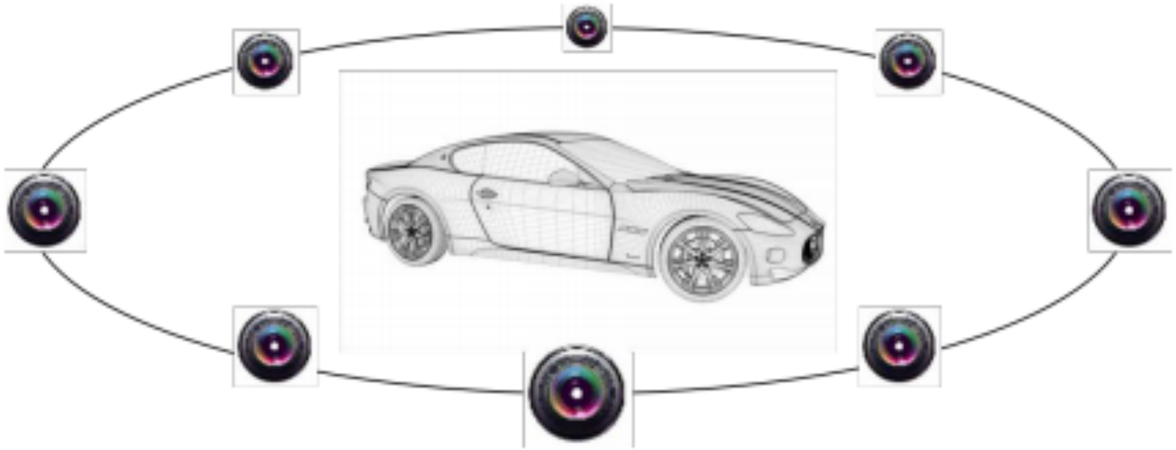


loses information due to occlusion

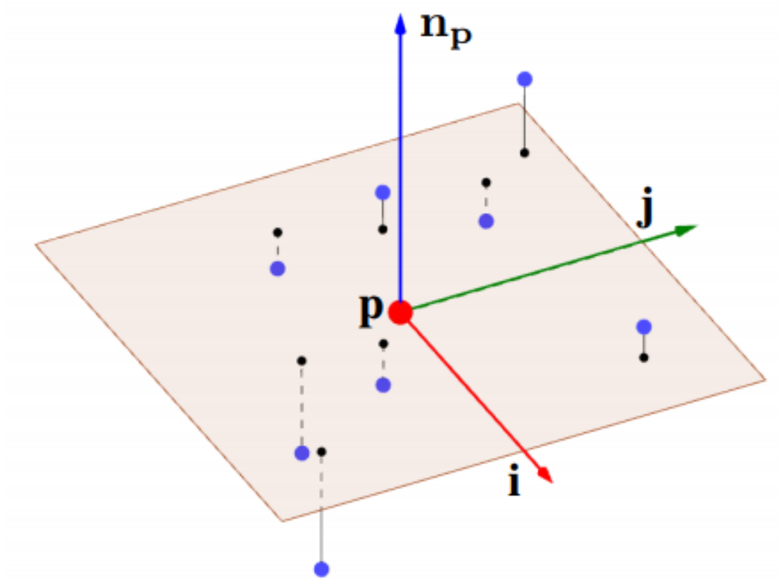
project to local patches
(contrast with PCPNet construction)

[Tatarchenko et al. 2018]

Tangent Convolutions



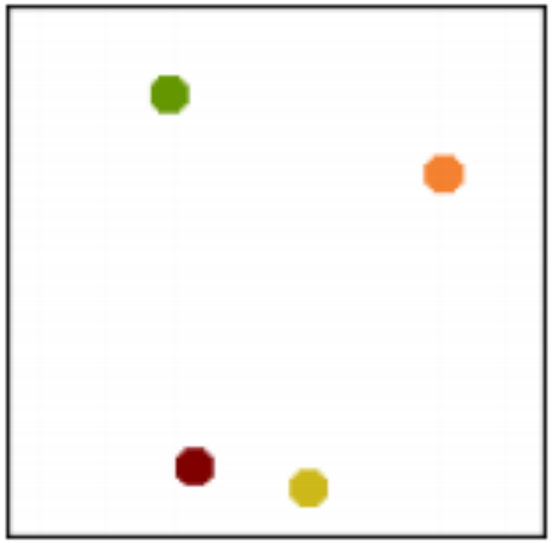
loses information due to occlusion



project to local patches
(contrast with PCPNet construction)

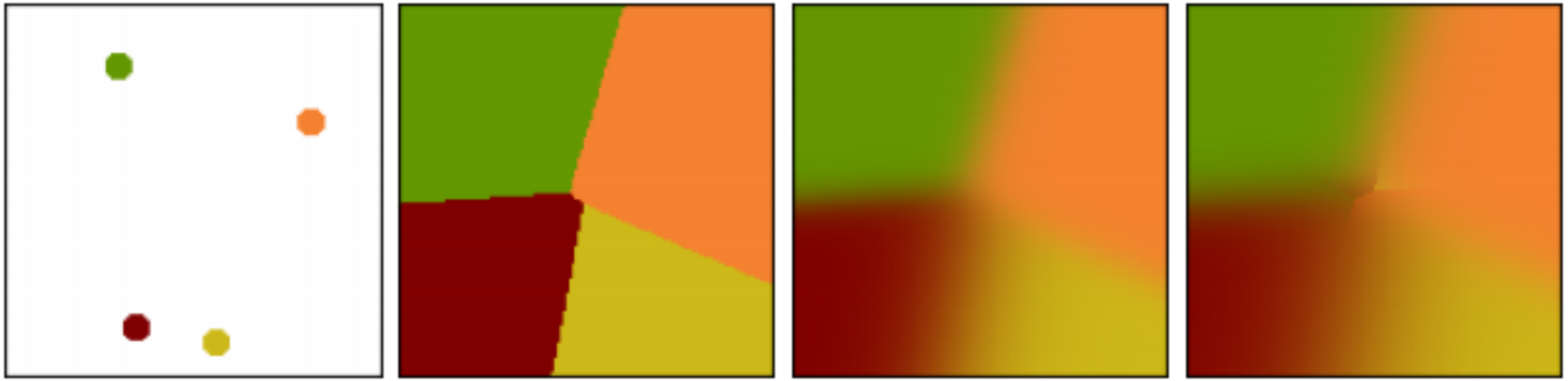
[Tatarchenko et al. 2018]

Dealing with Sparse Points



(a)

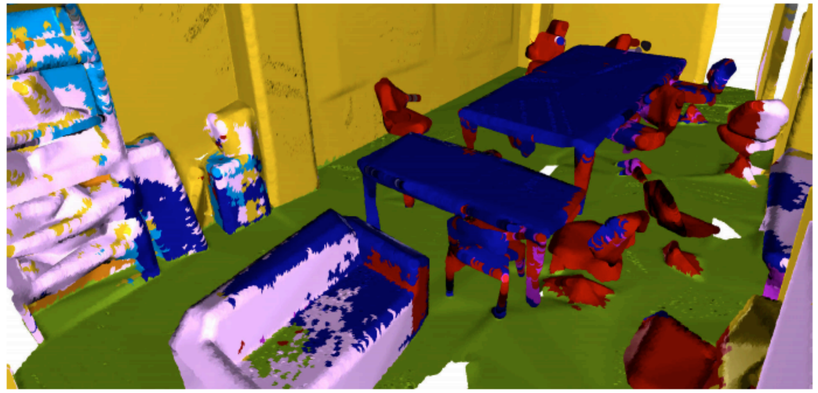
Dealing with Sparse Points



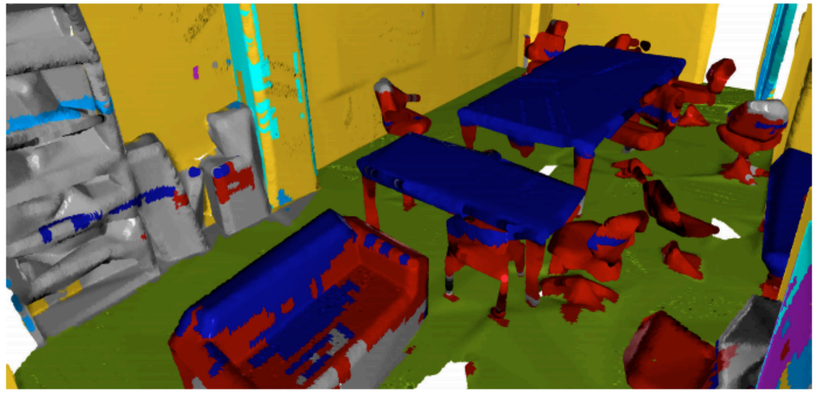
(a) (b) (c) (d)

Improved Performance

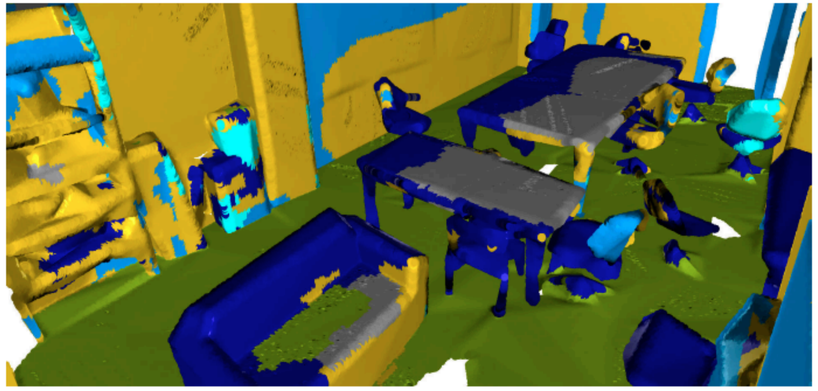
Color



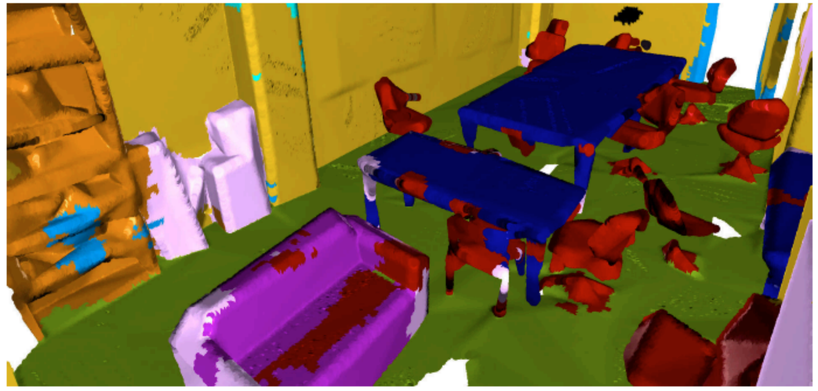
PointNet [39]



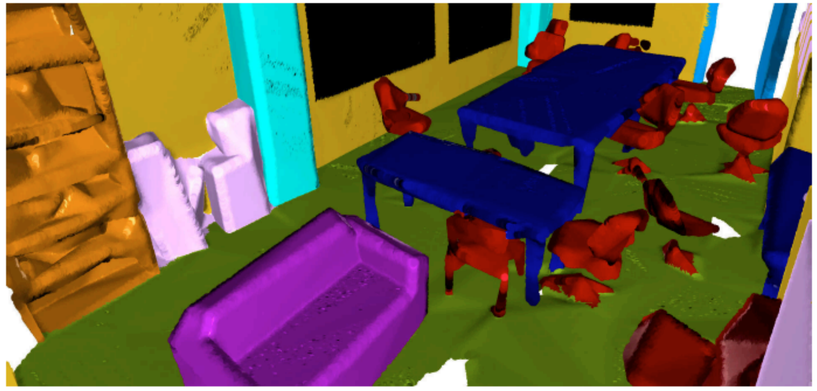
ScanNet [10]



OctNet [43]



Ours (DHNRGB)



Ground truth

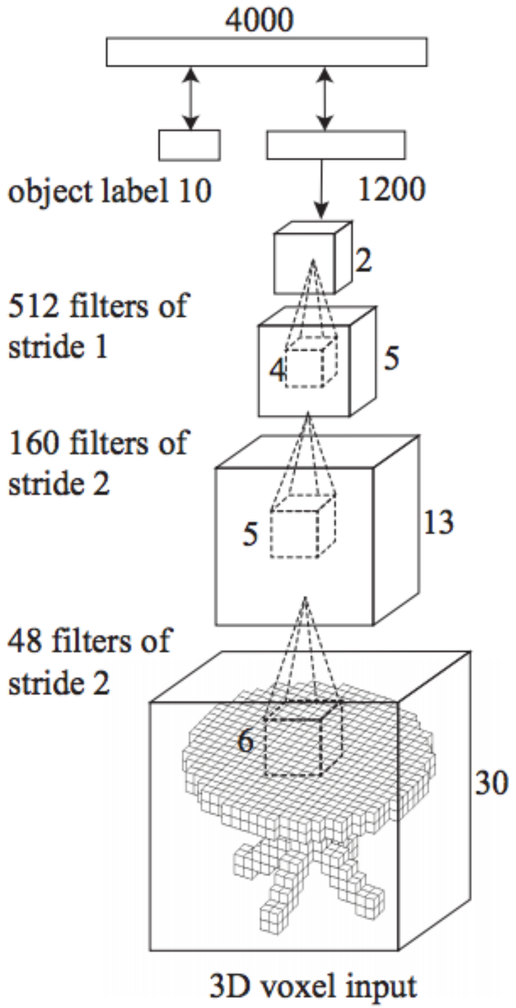
Representation for 3D

- **Image-based**
  - **PROS**: directly use image networks, good performance
  - **CONS**: rendering is slow and memory-heavy, not very geometric
- Volumetric
- Point-based
- Surface-based

Representation for 3D

- Image-based
- **Volumetric**
- Surface-based
- Point-based

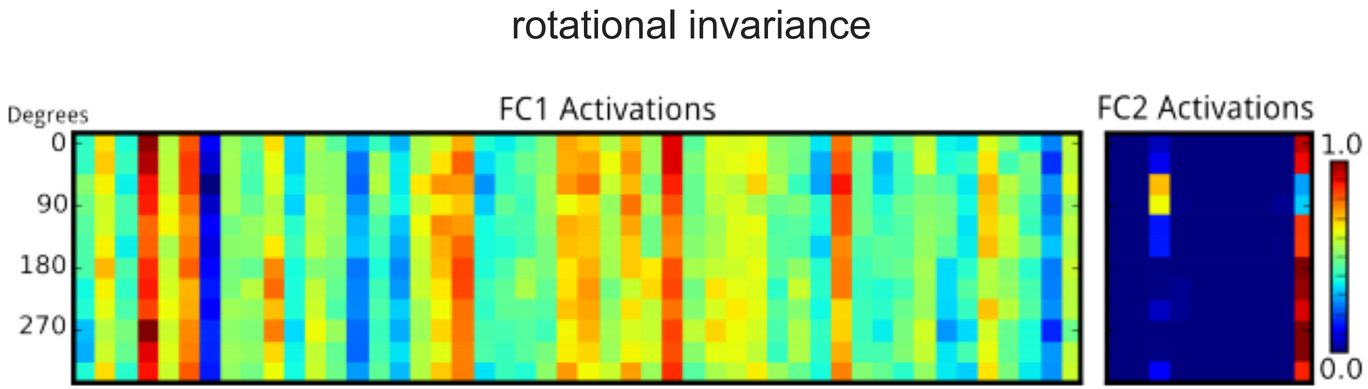
3D CNNs : Direct Approach

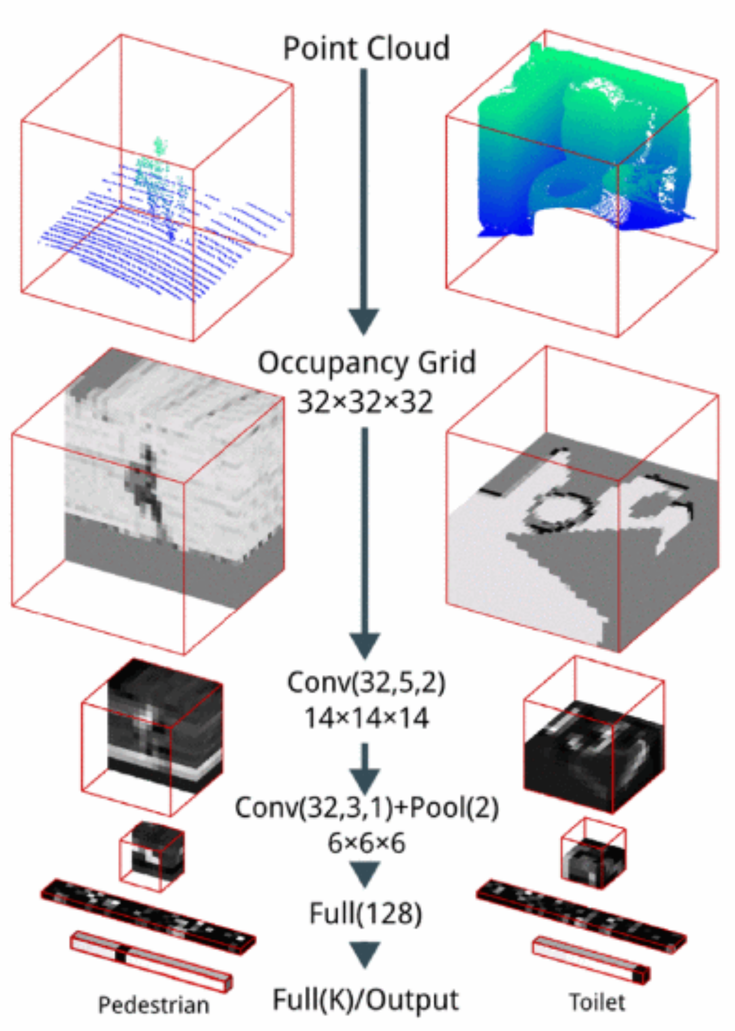


[Xiao et al. 2014]

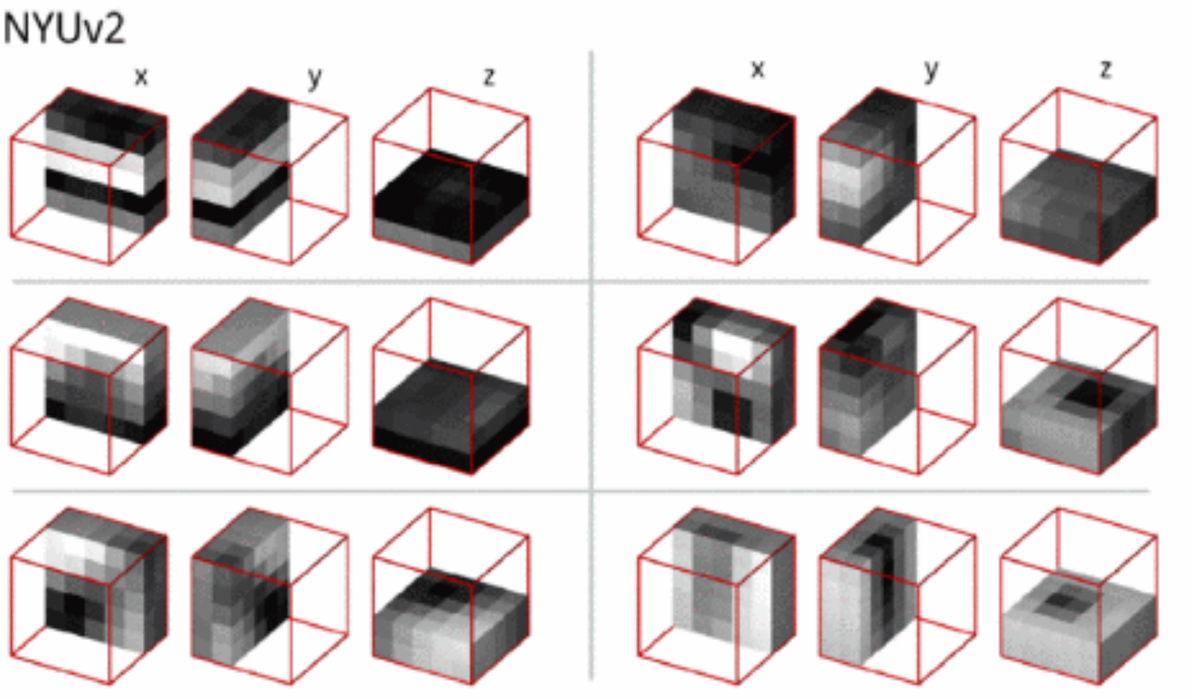
VoxNet [Maturana et al. 15]

- Binary occupancy, density grid, etc.

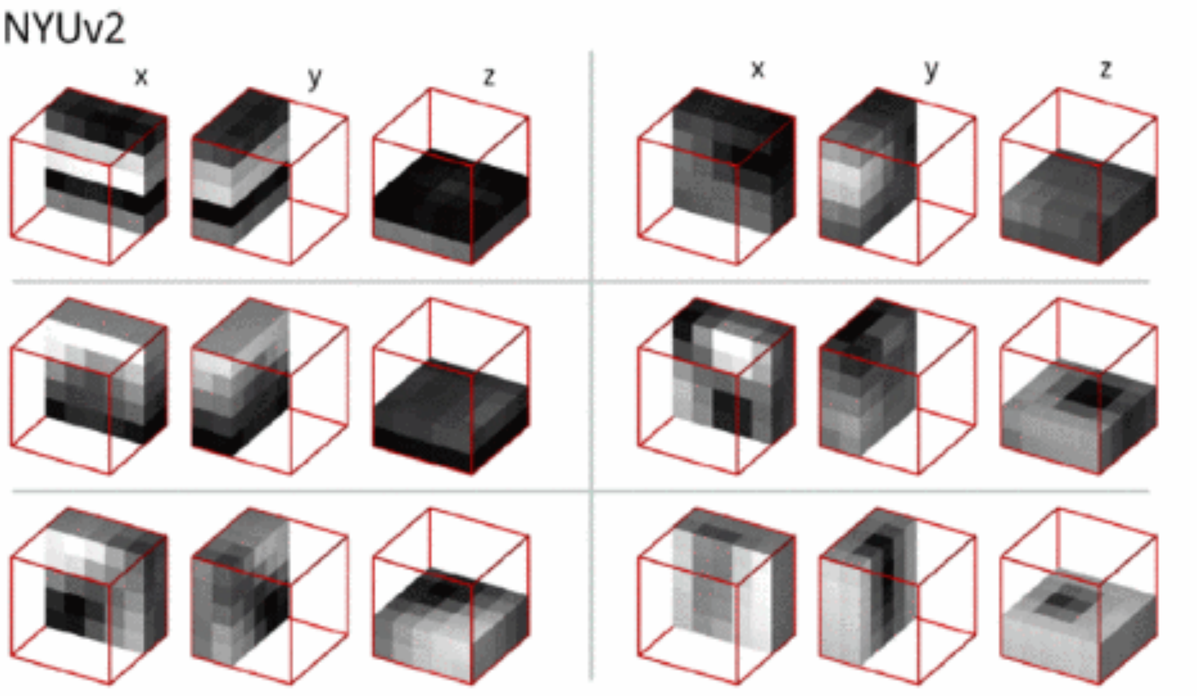


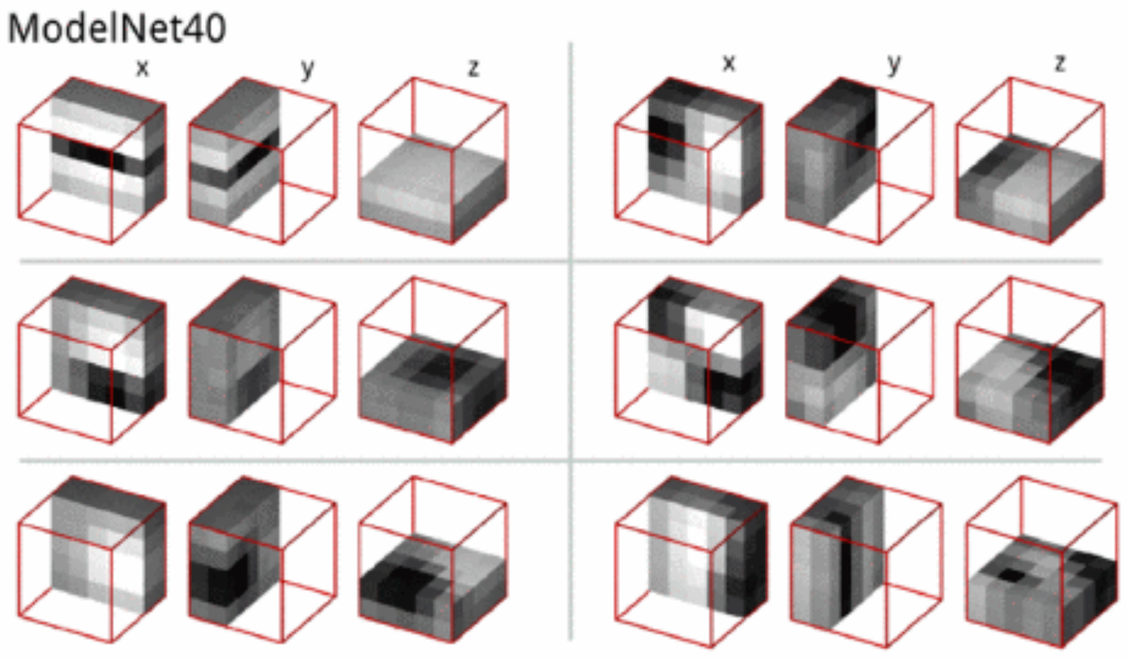


Visualization of First Level Filters

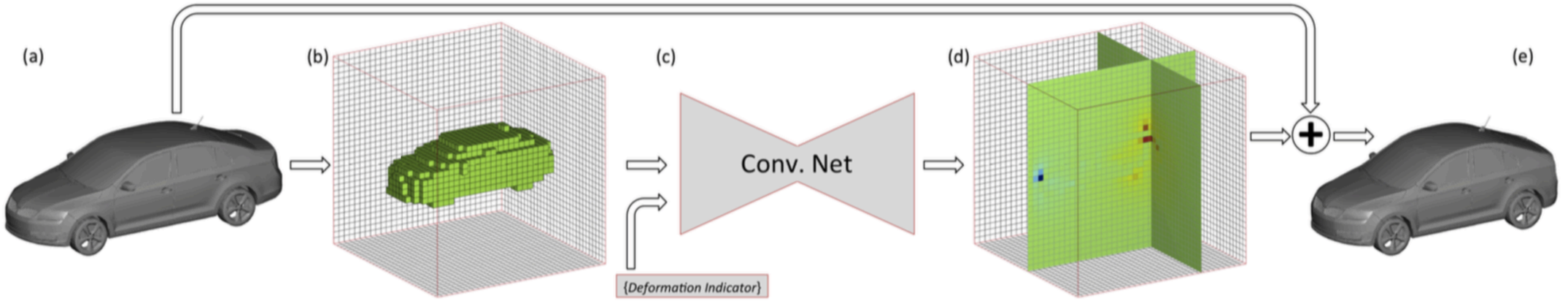


Visualization of First Level Filters



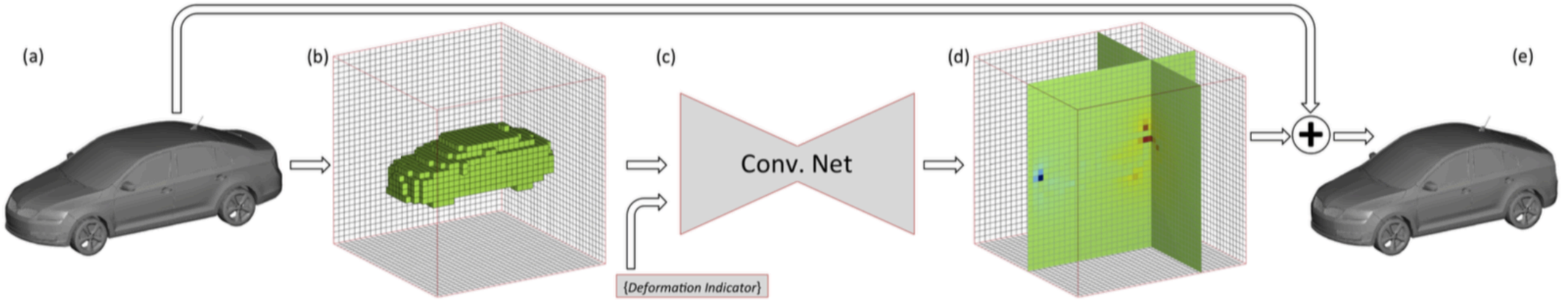


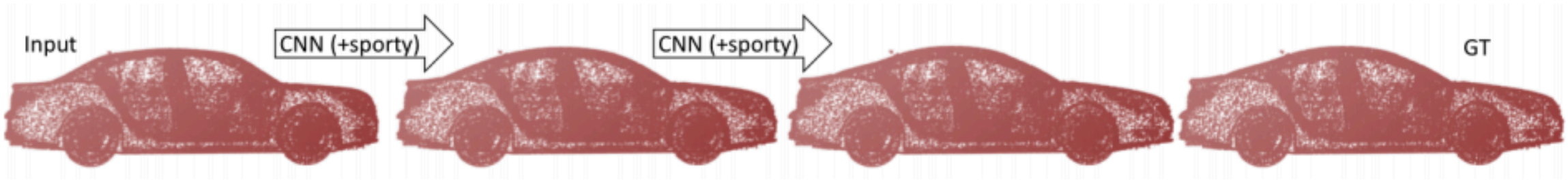
Representation for 3D: Volumetric Deformation



[Yumer and Mitra 2016]

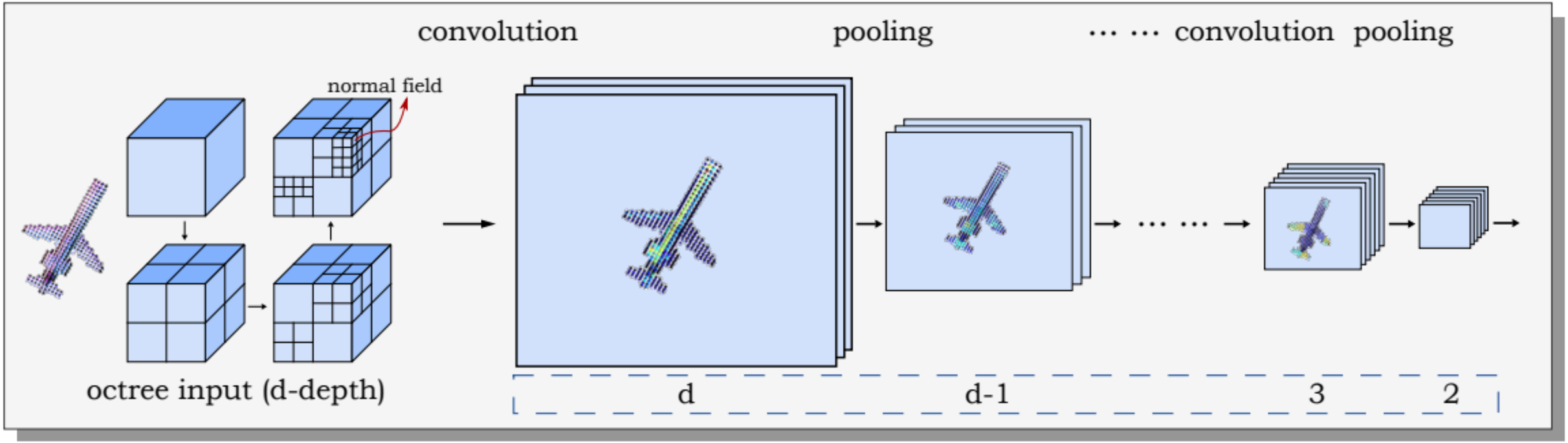
Representation for 3D: Volumetric Deformation





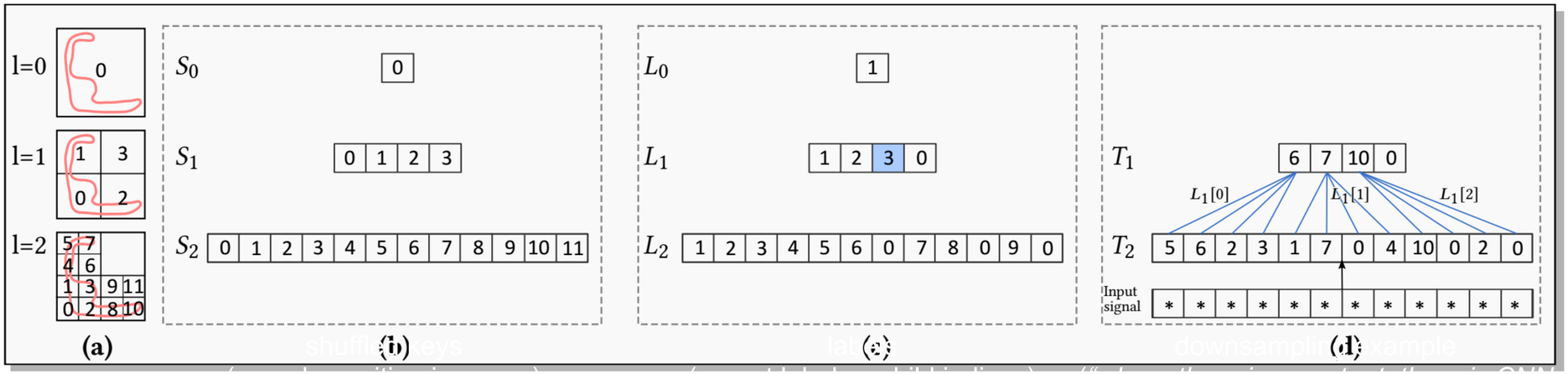
[Yumer and Mitra 2016]

Efficient Volumetric Datastructures

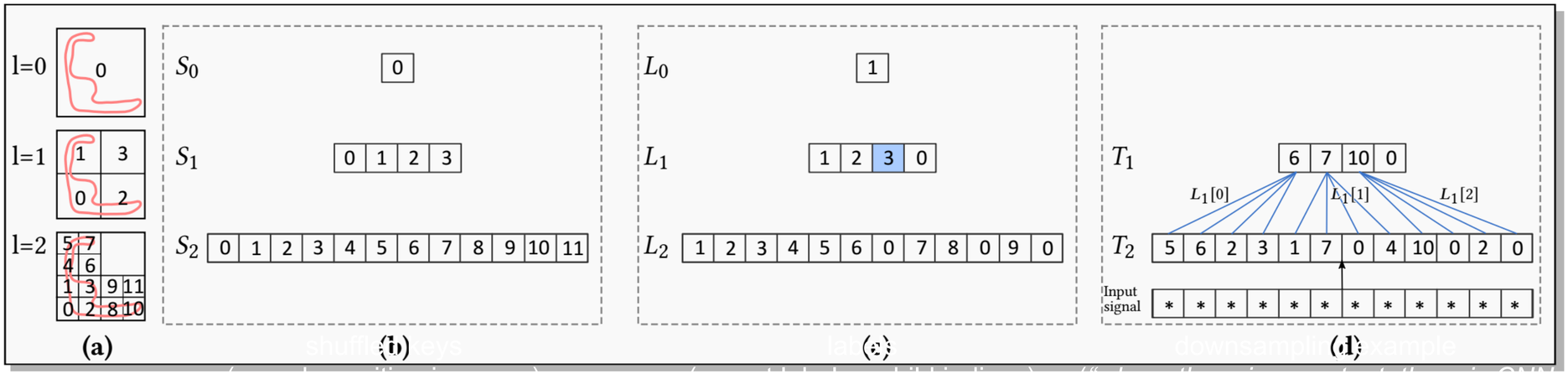


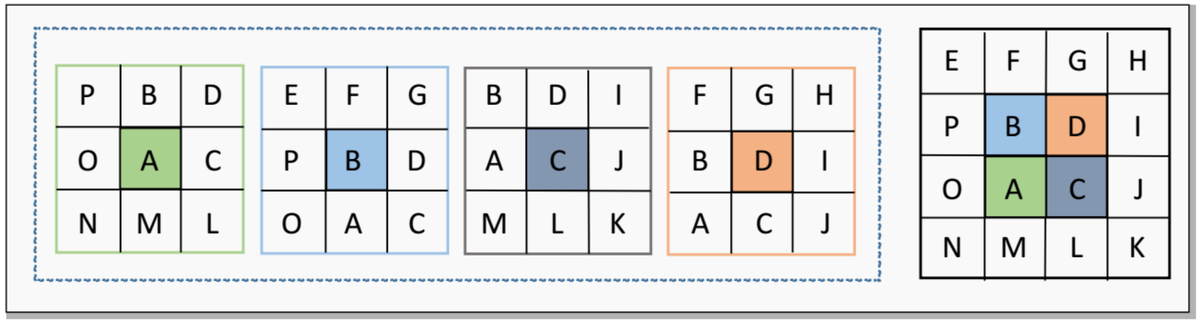
[Wang et al. 2017]

Data Structure and CNN Operations

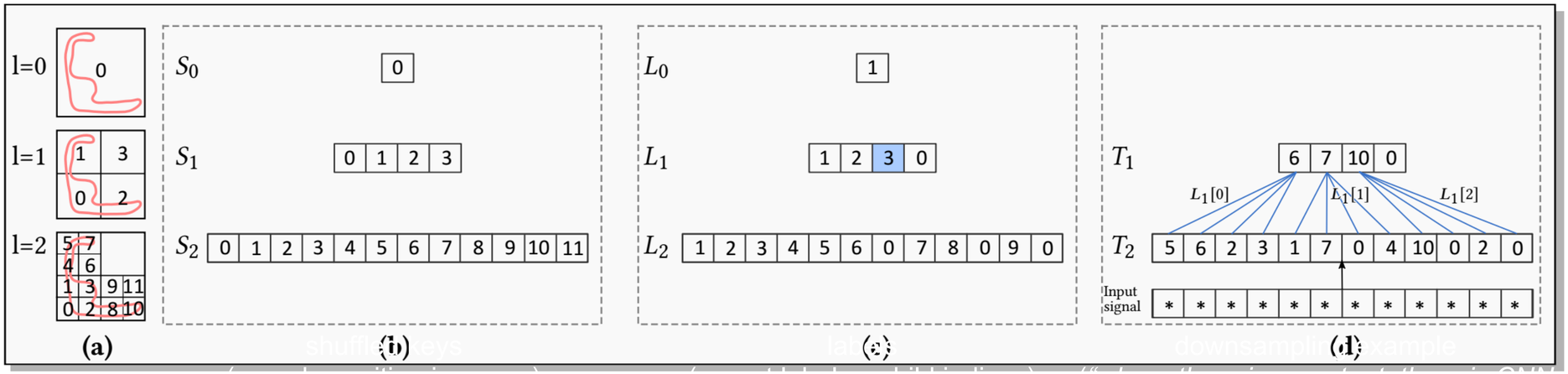


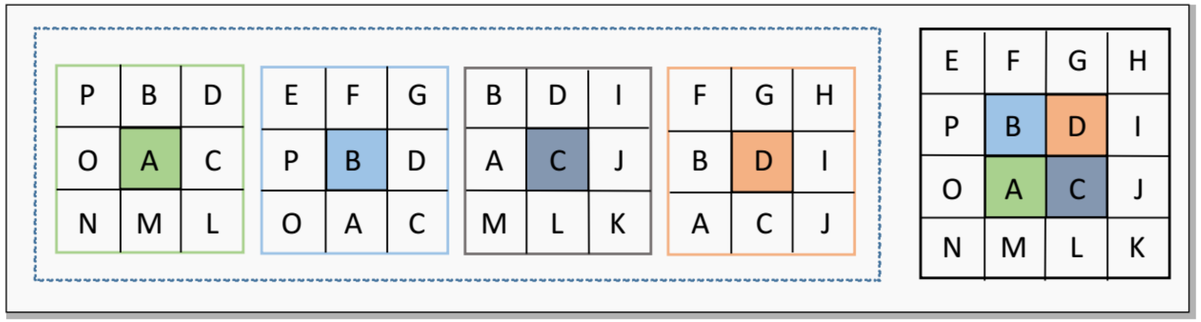
Data Structure and CNN Operations

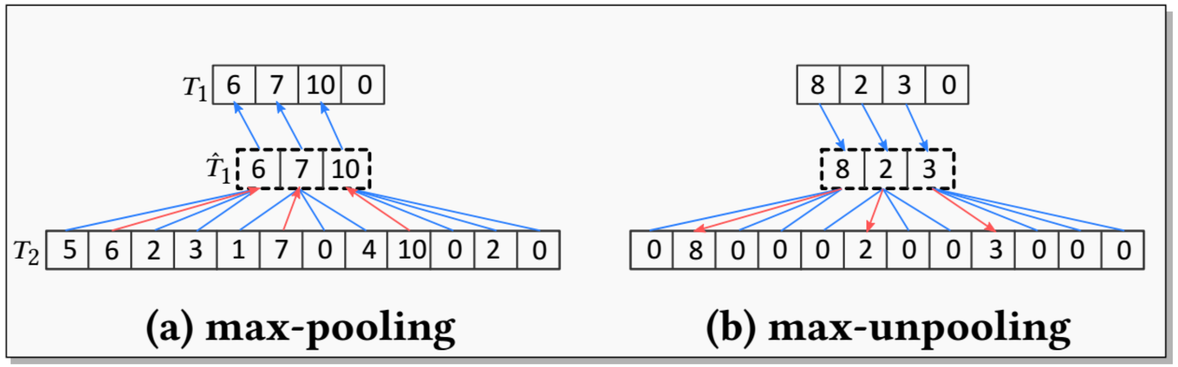




Data Structure and CNN Operations

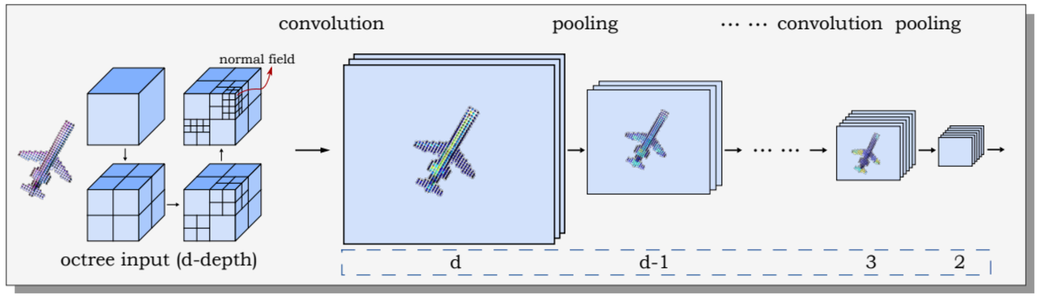






Efficient Volumetric Datastructures

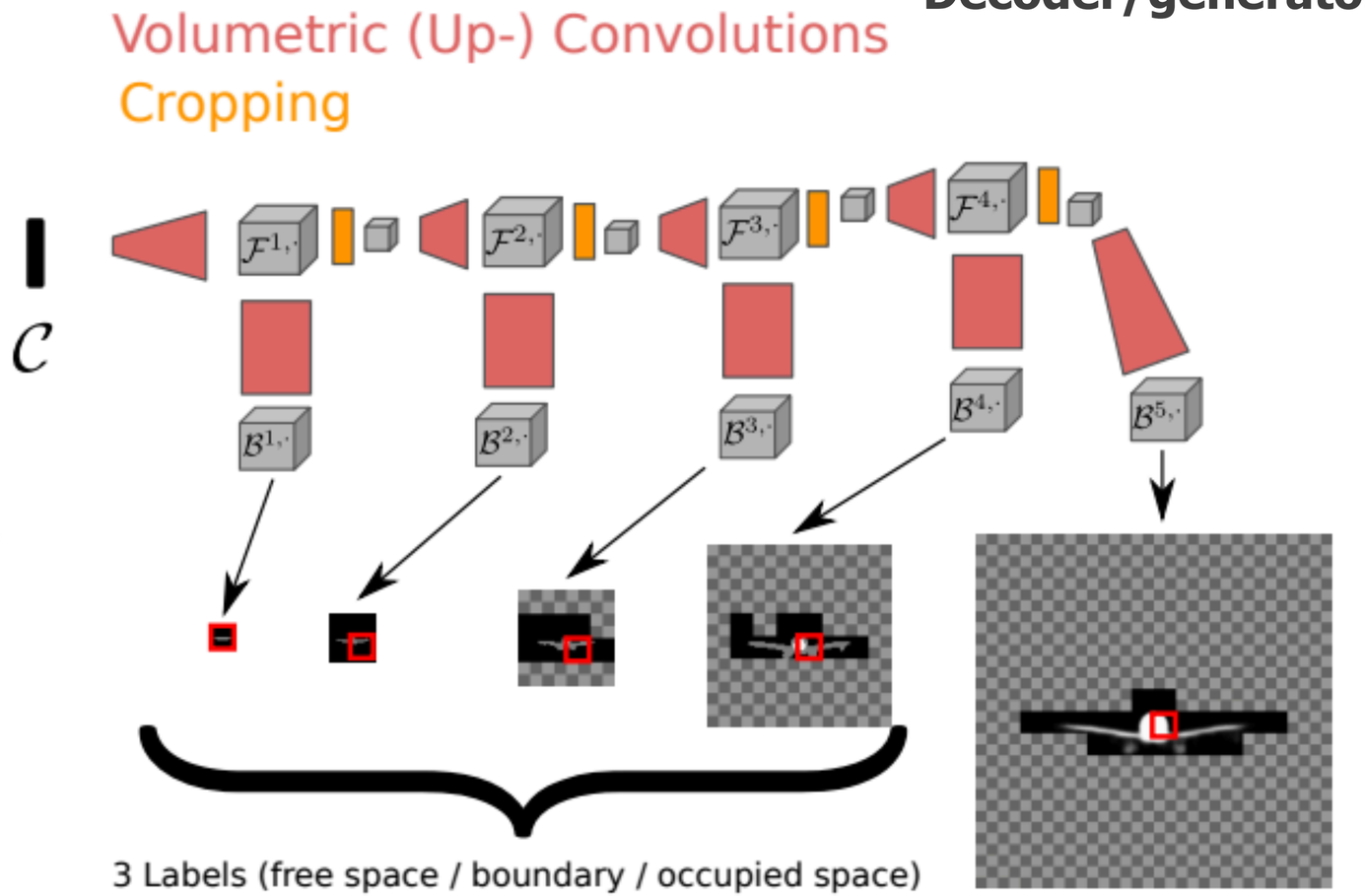
Encoder



Wang et al. 2017

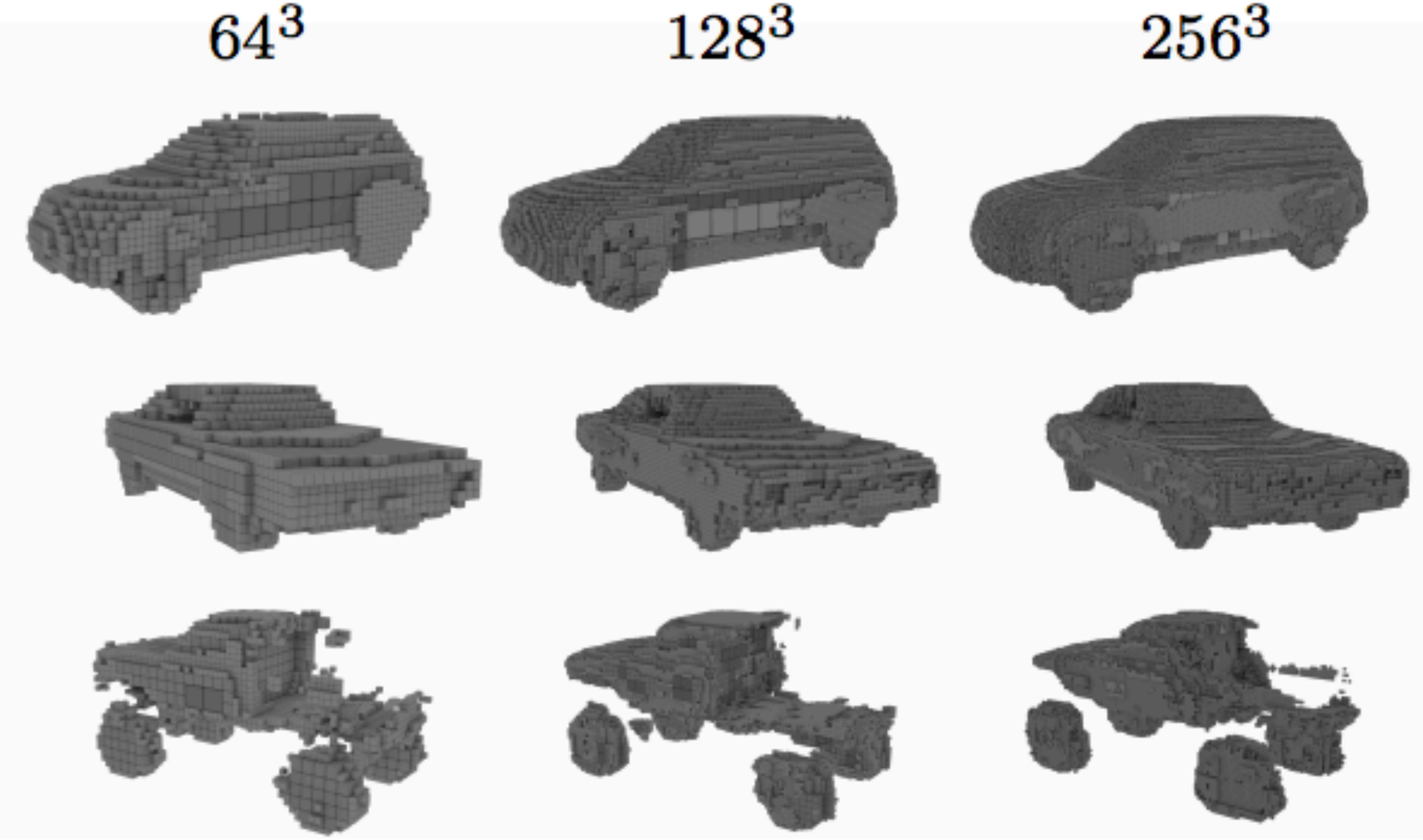
only generate non-empty voxels

Decoder/generator



[Hane et al. 2018]

Efficient Volumetric Datastructures



[Hane et al. 2018]

Lower Memory Footprint

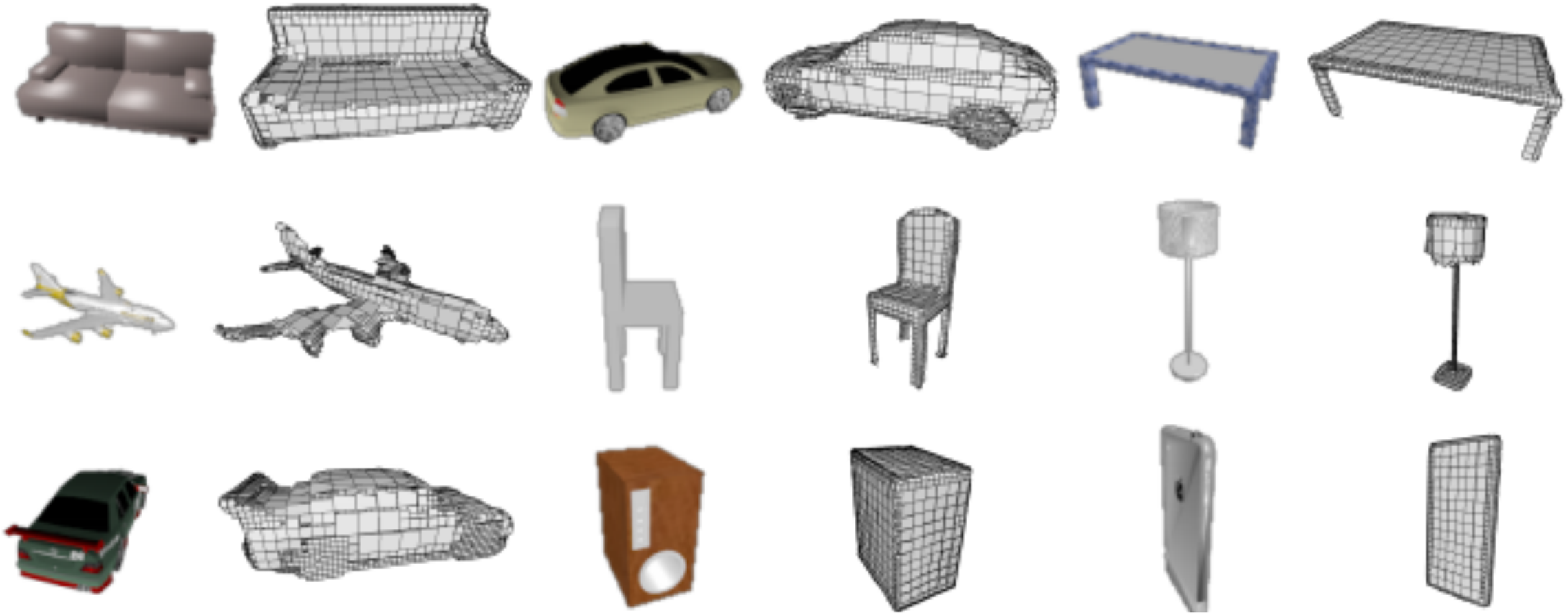
Method	16^3	32^3	64^3	128^3	256^3
O-CNN	0.32GB	0.58GB	1.1GB	2.7GB	6.4GB
full voxel+binary	0.23GB	0.71GB	3.7GB	Out of memory	Out of memory
full voxel+normal	0.27GB	1.20GB	4.3GB	Out of memory	Out of memory

Table 3. Comparisons on GPU-memory consumption. The batch size is 32.

Method	16^3	32^3	64^3	128^3	256^3
O-CNN	17ms	33ms	90ms	327ms	1265ms
full voxel+binary	59ms	425ms	1648ms	-	-
full voxel+normal	75ms	510ms	4654ms	-	-

Table 4. Timings of one backward and forward operation in milliseconds. The batch size is 32.

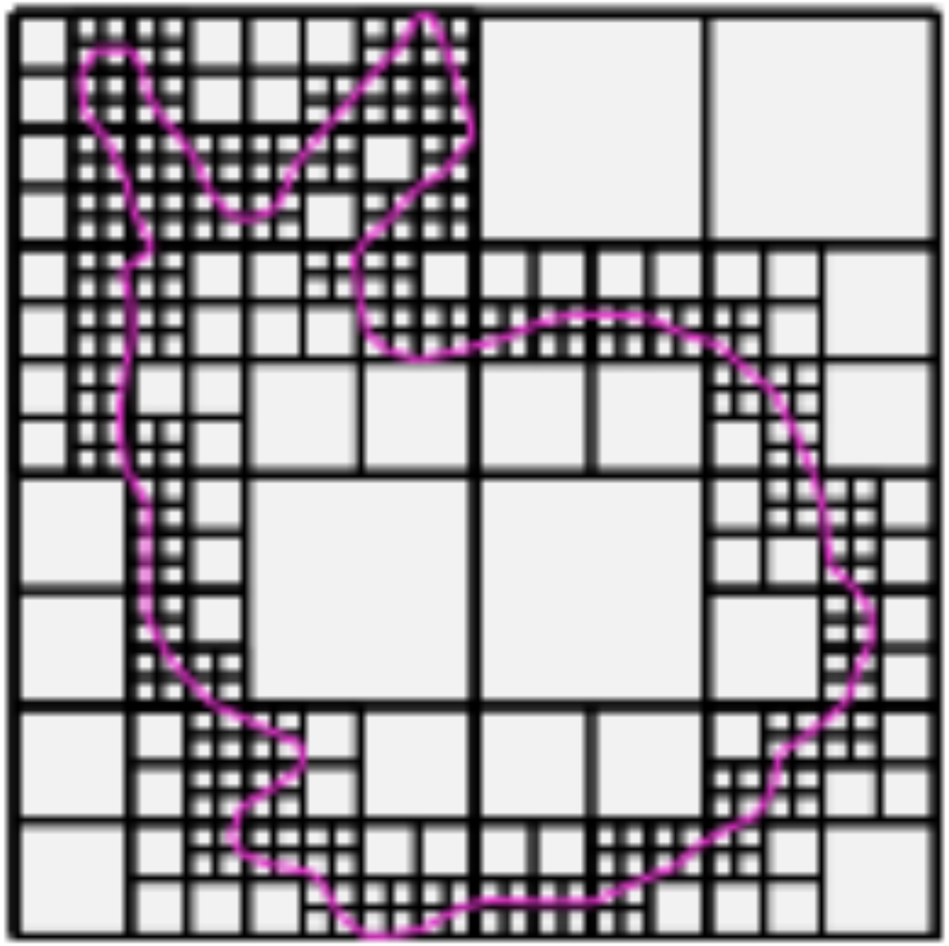
Adaptive O-CNN



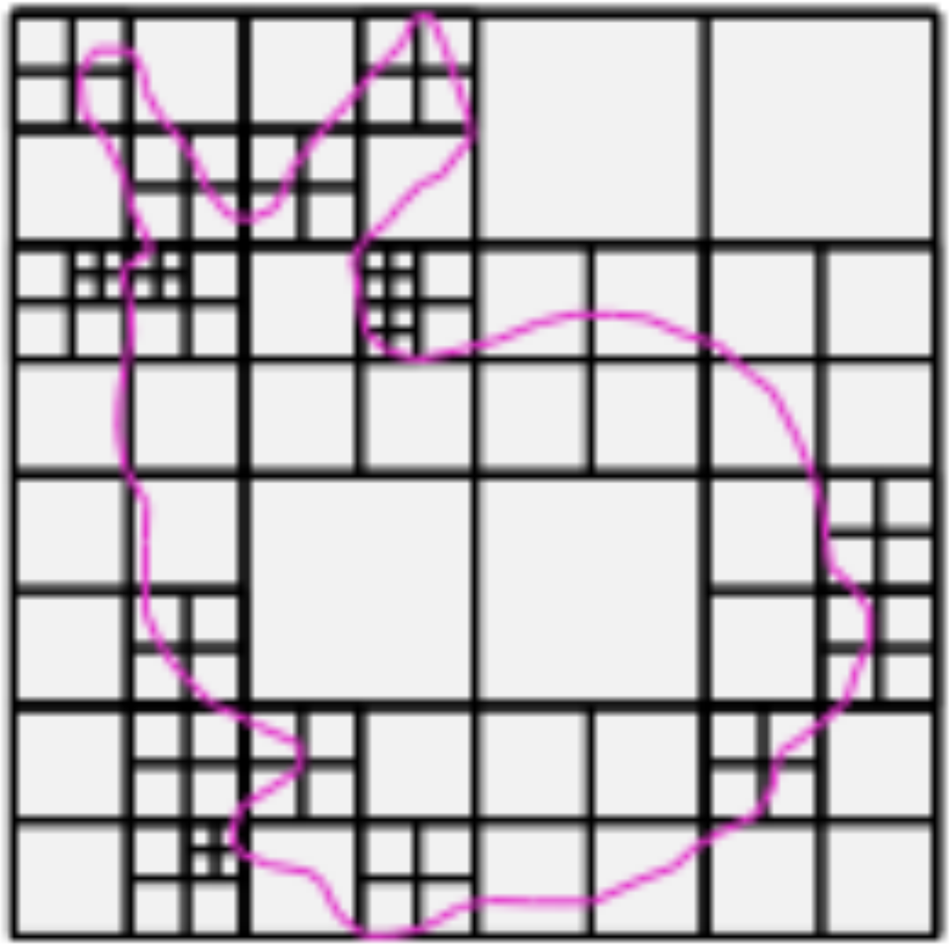
[Wang et al. 2018]

image to planar patch-based shapes

First-order Patches



OCNN

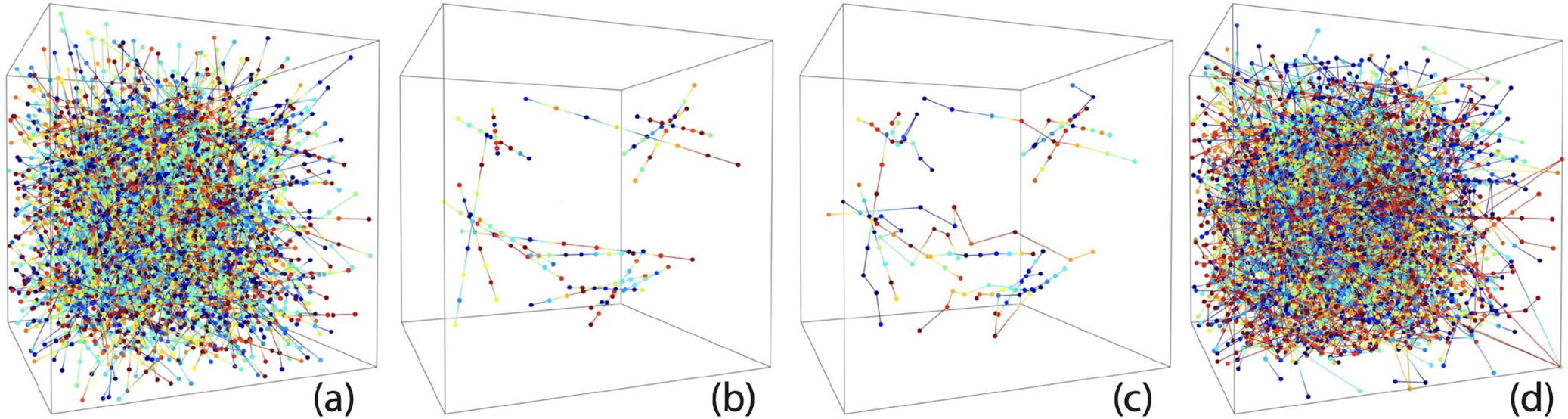


Adaptive OCNN

Field Probing Neural Networks for 3D Data

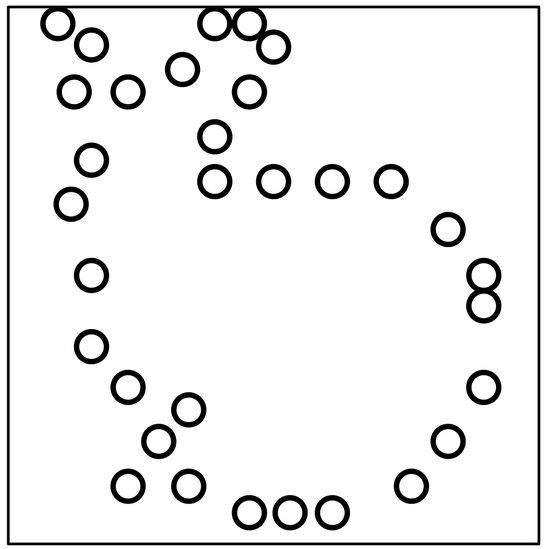
[Li et al. 2016]

Field Probing Neural Networks for 3D Data

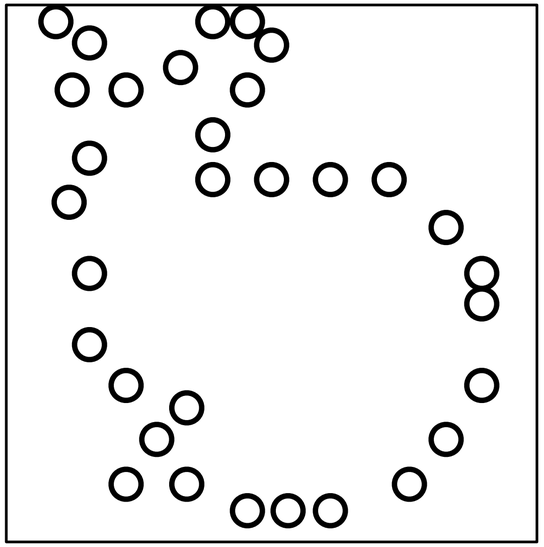


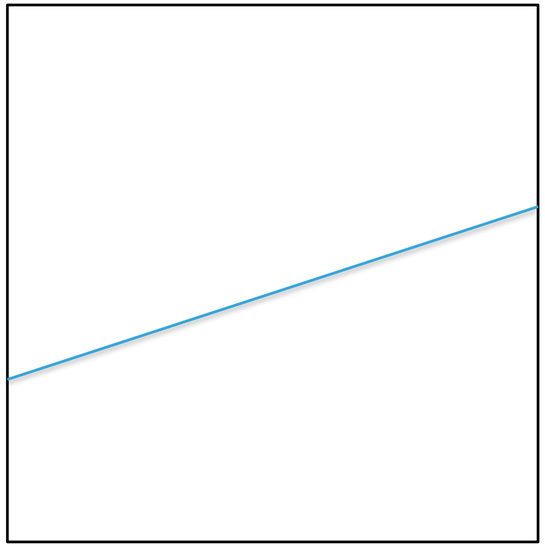
[Li et al. 2016]

Spatial Probes

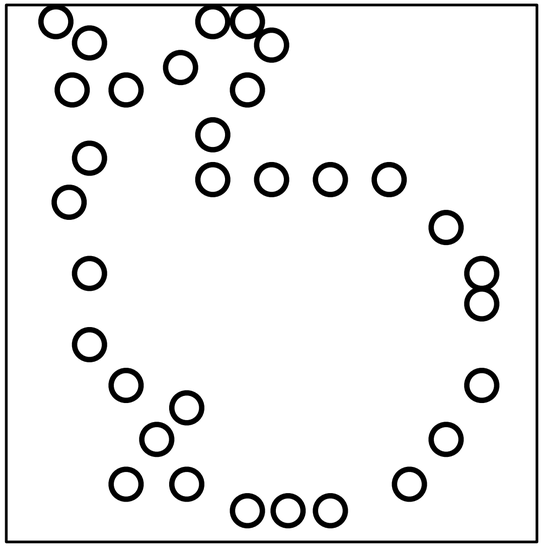


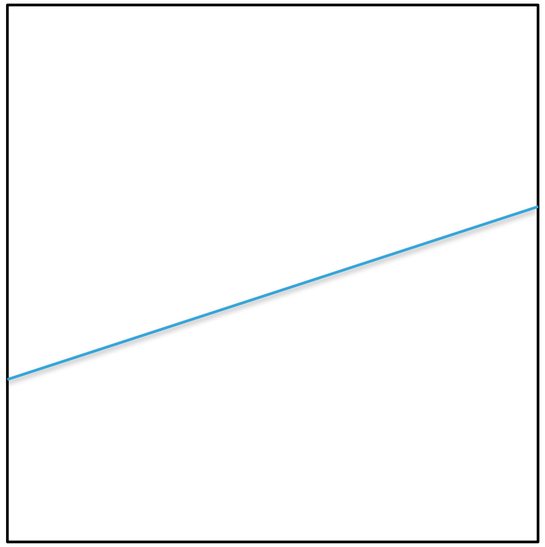
Spatial Probes

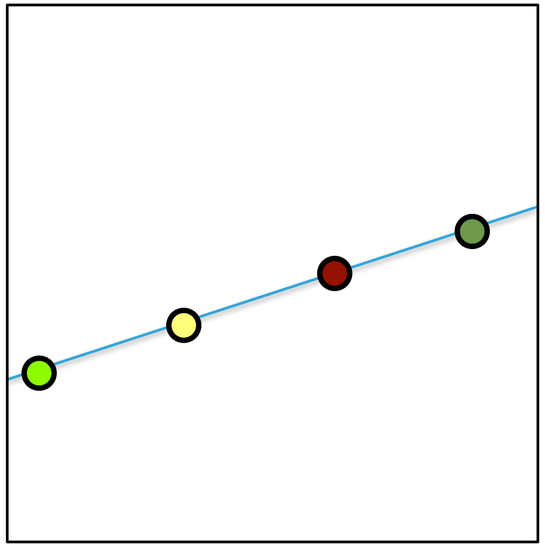




Spatial Probes

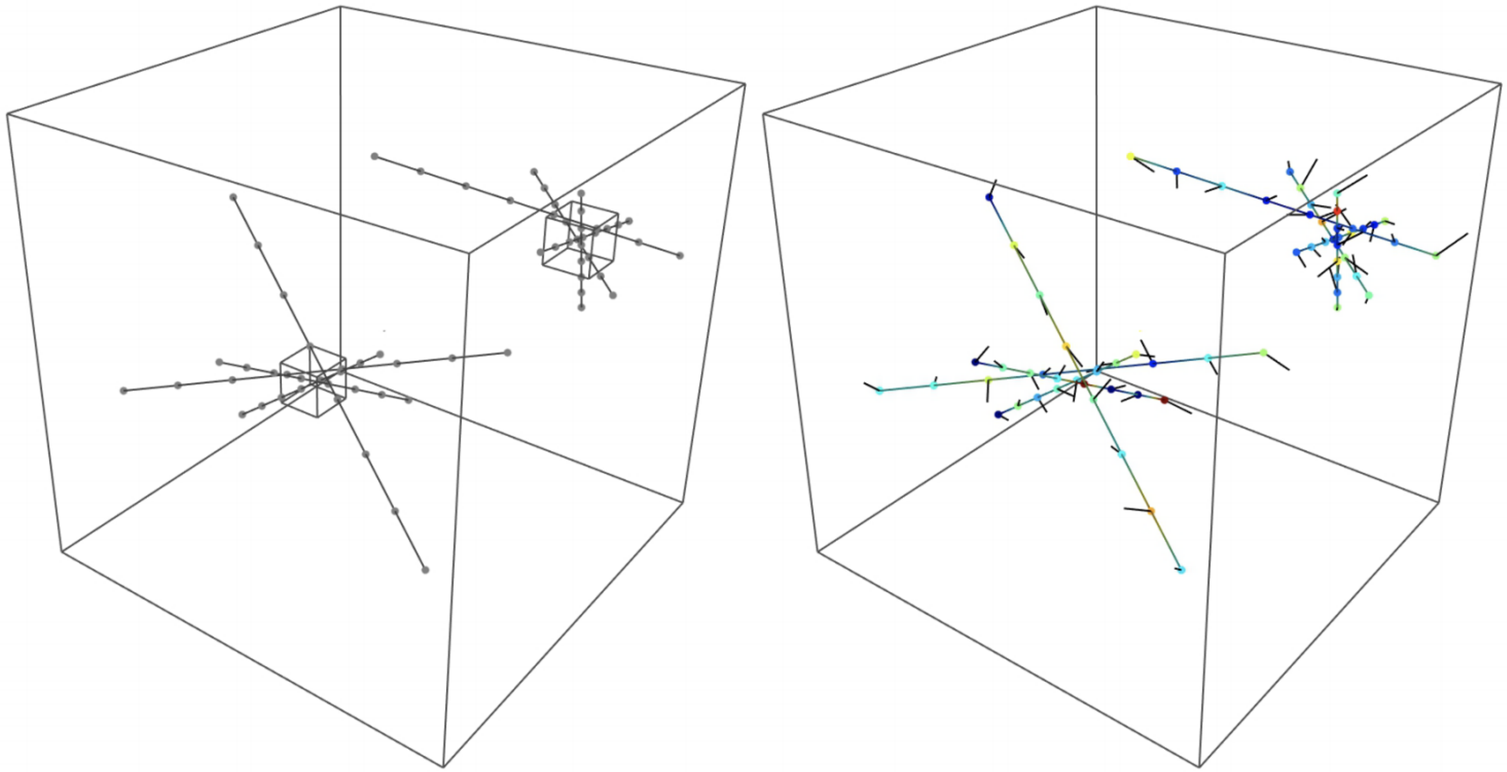




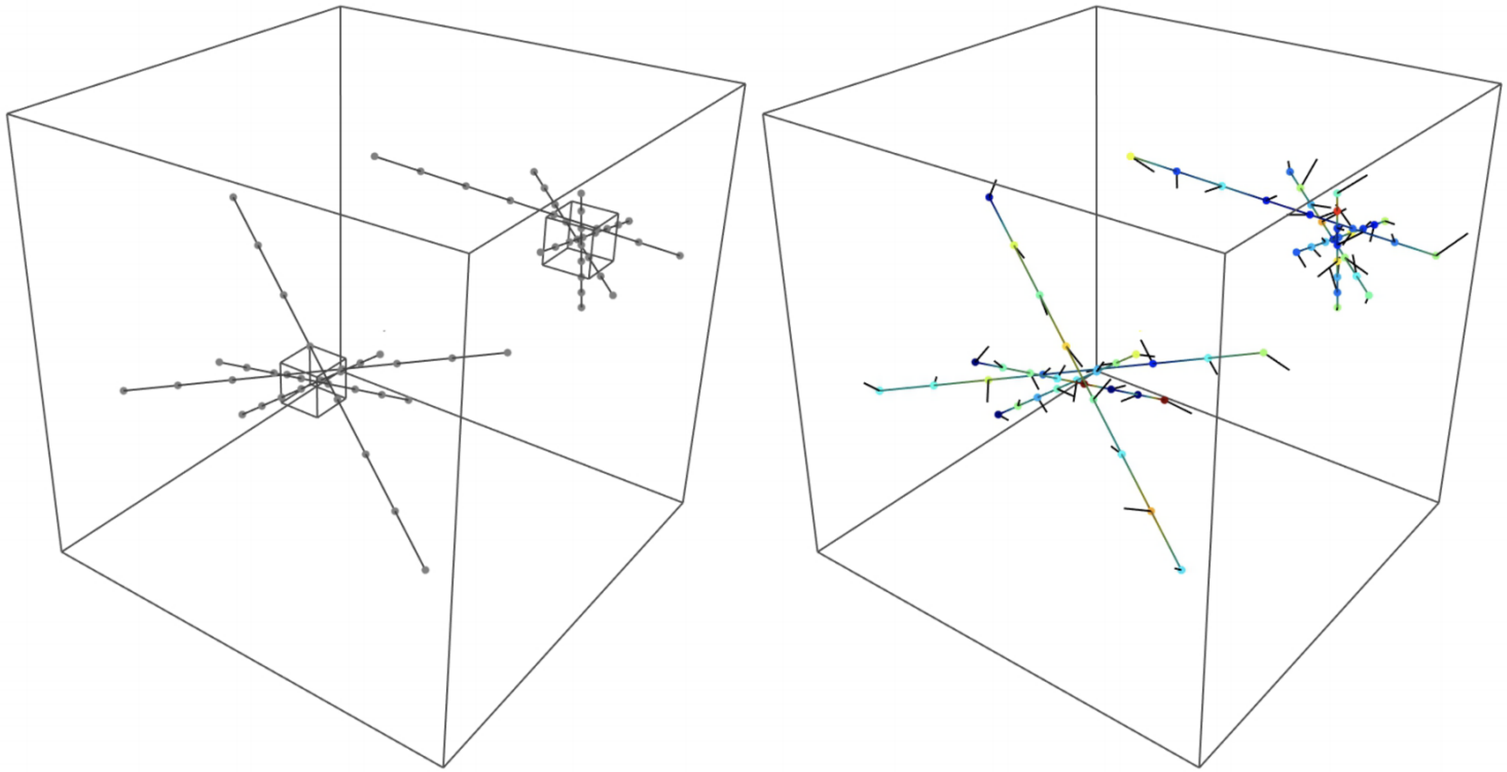


Method Details

Method Details

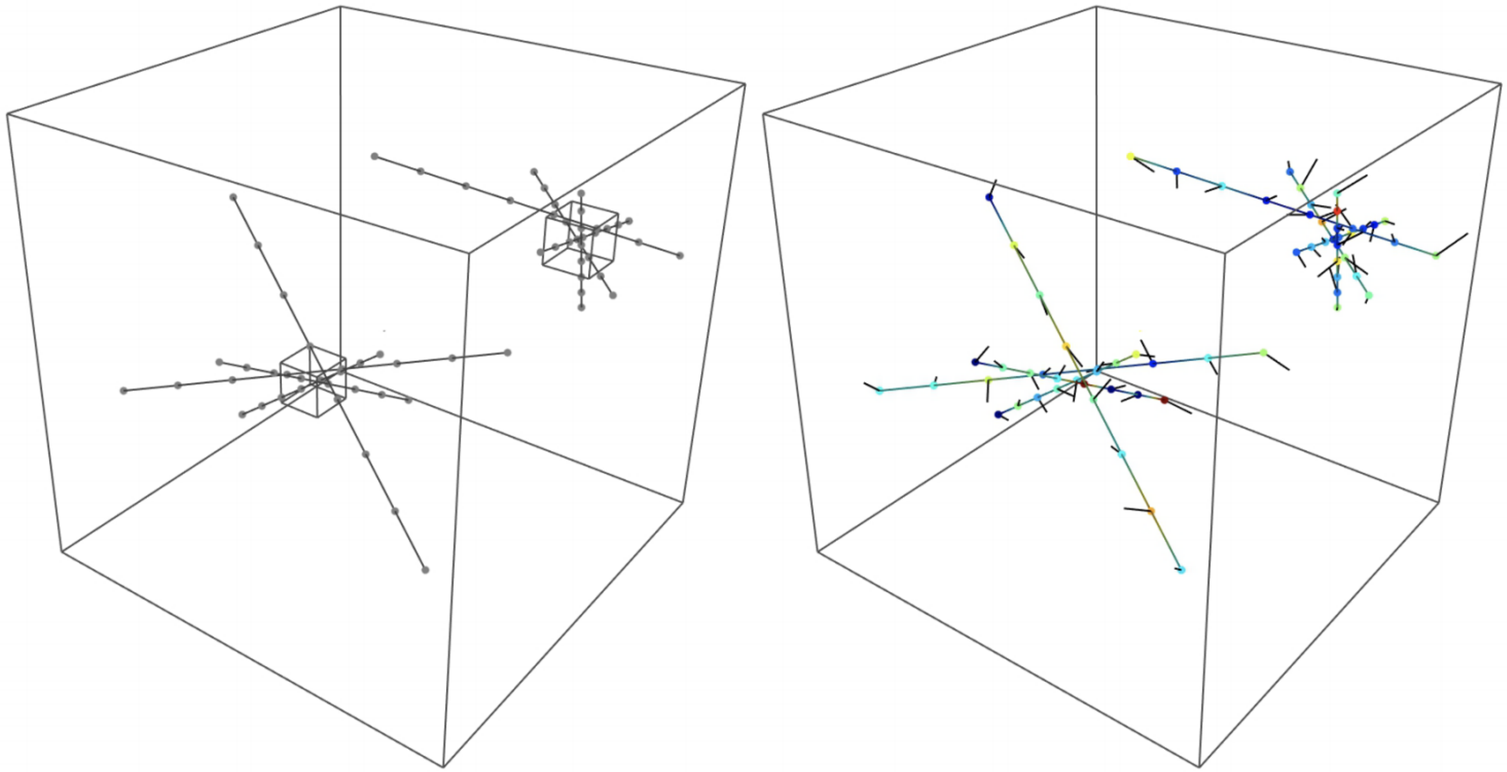


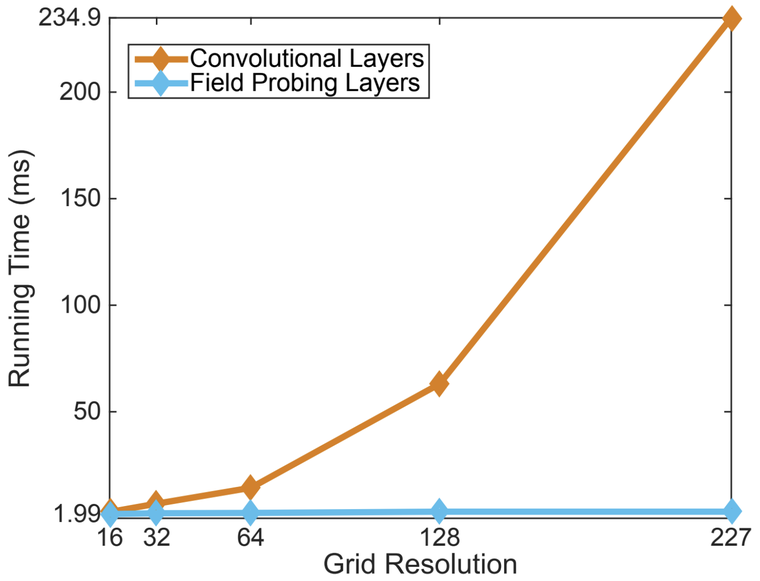
Method Details



$$v_c = v(\{p_{c,i,j}\}, \{w_{c,i,j}\}) = \sum_{\substack{i=1,\ldots,N \\ j=1,\ldots,T}} p_{c,i,j} \times w_{c,i,j}$$

Method Details





$$v_c = v(\{p_{c,i,j}\}, \{w_{c,i,j}\}) = \sum_{\substack{i=1,\ldots,N \\ j=1,\ldots,T}} p_{c,i,j} \times w_{c,i,j}$$

Representation for 3D

- Image-based

- Volumetric
  - **PROS**: adaptations of image networks
  - **CONS**: special layers for hierarchical datastructures, still too coarse

- Surface-based

- Point-based

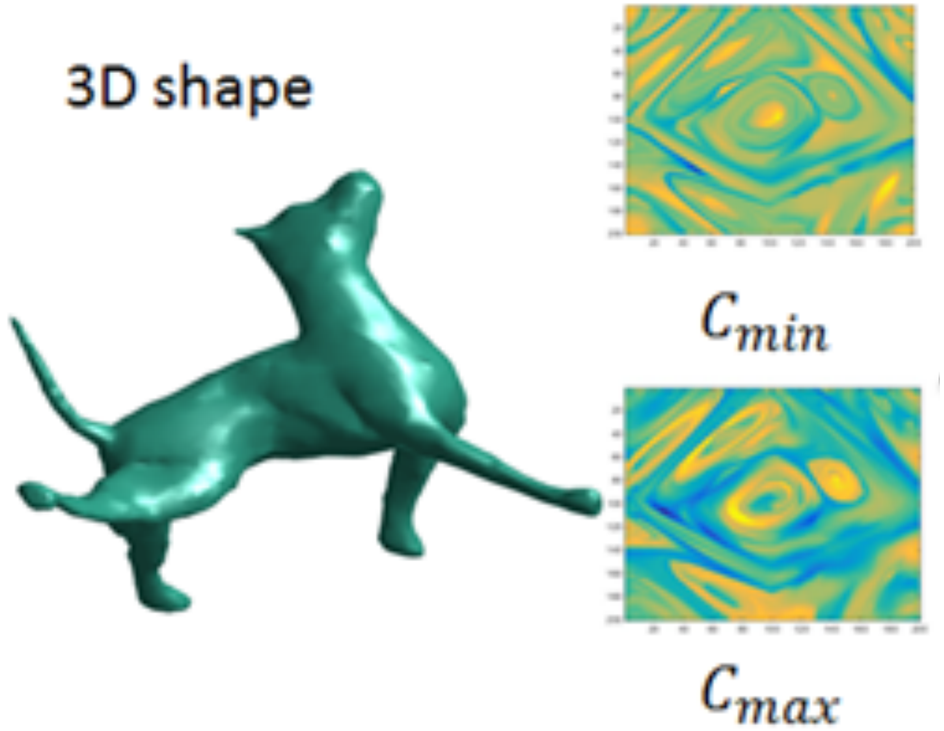
Representation for 3D

- Image-based
- Volumetric
- **Surface-based**
- Point-based

Local/Global Parameterizations

Local/Global Parameterizations

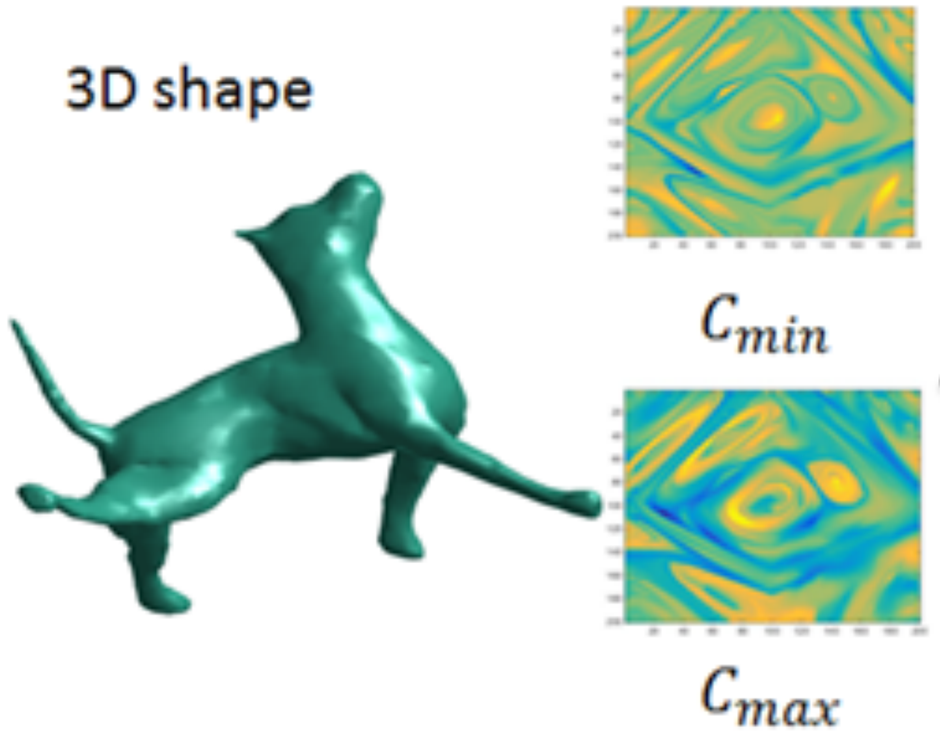
Geometry Image



[Sinha et al. 2016]

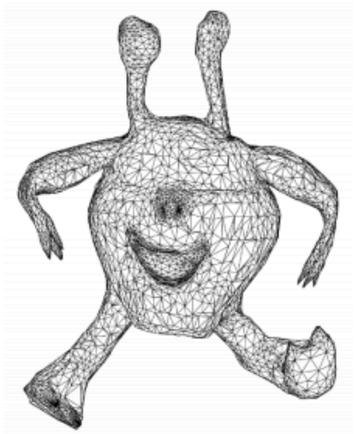
Local/Global Parameterizations

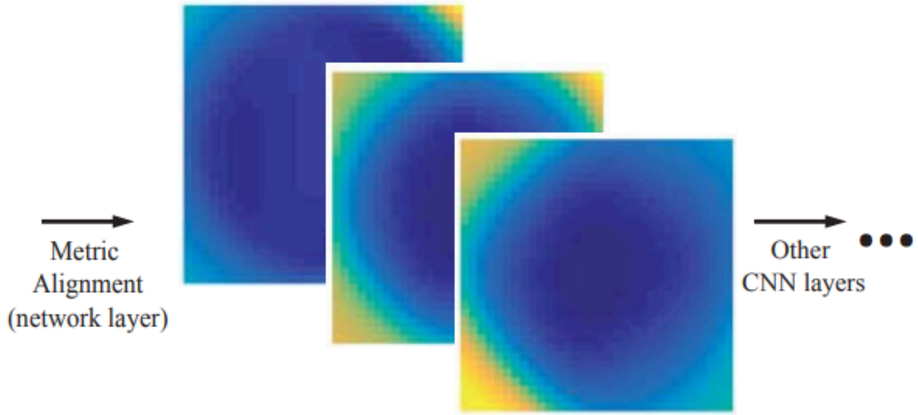
Geometry Image



[Sinha et al. 2016]

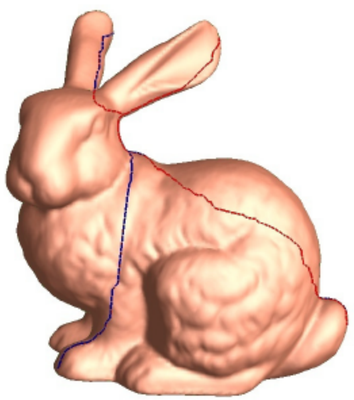
Metric Alignment (GWCNN)



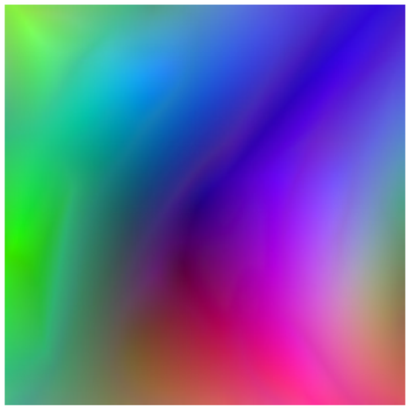


[Ezuz et al. 2017]

Shape Surfaces using Geometry Images

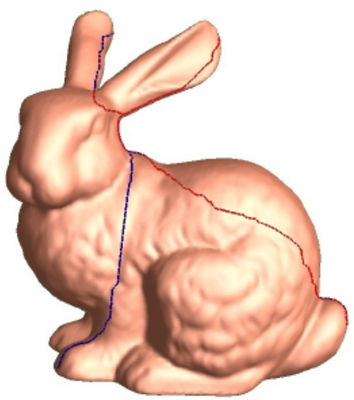


(a) Original mesh with cut
70K faces; genus 0

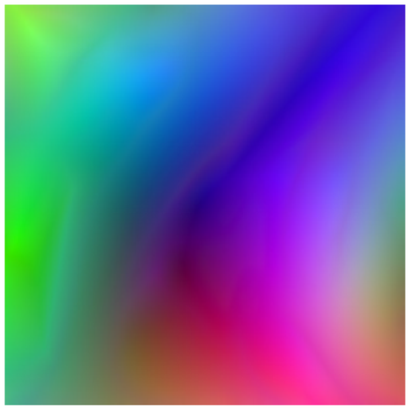


(b) Geometry image 257×257
(b∗) Compr. to 1.5KB (not shown)

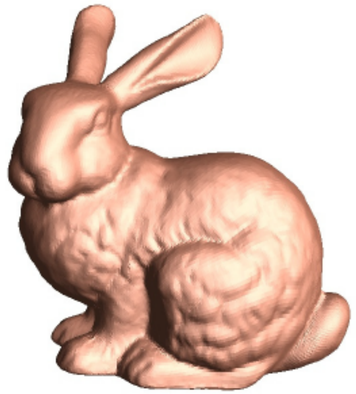
Shape Surfaces using Geometry Images



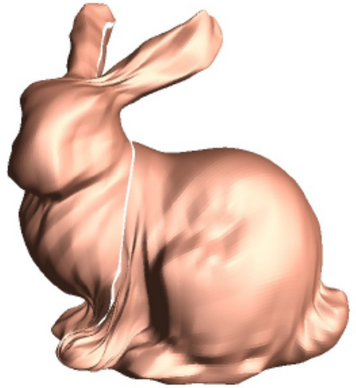
(a) Original mesh with cut
70K faces; genus 0



(b) Geometry image 257×257
(b∗) Compr. to 1.5KB (not shown)

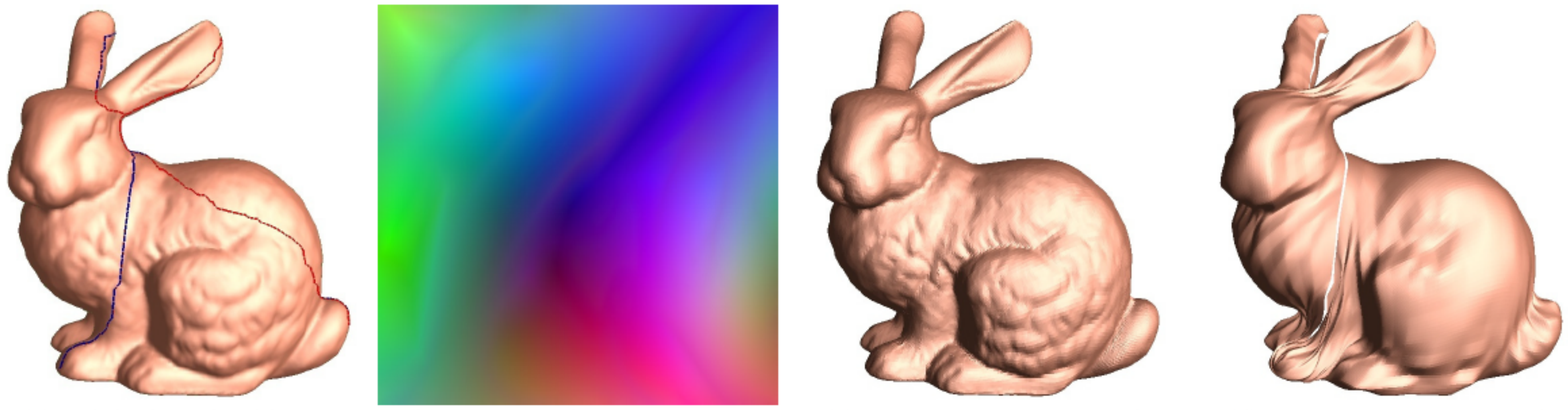


(c) Geometry reconstructed
entirely from b



(d) Geometry reconstructed
entirely from b∗

Shape Surfaces using Geometry Images

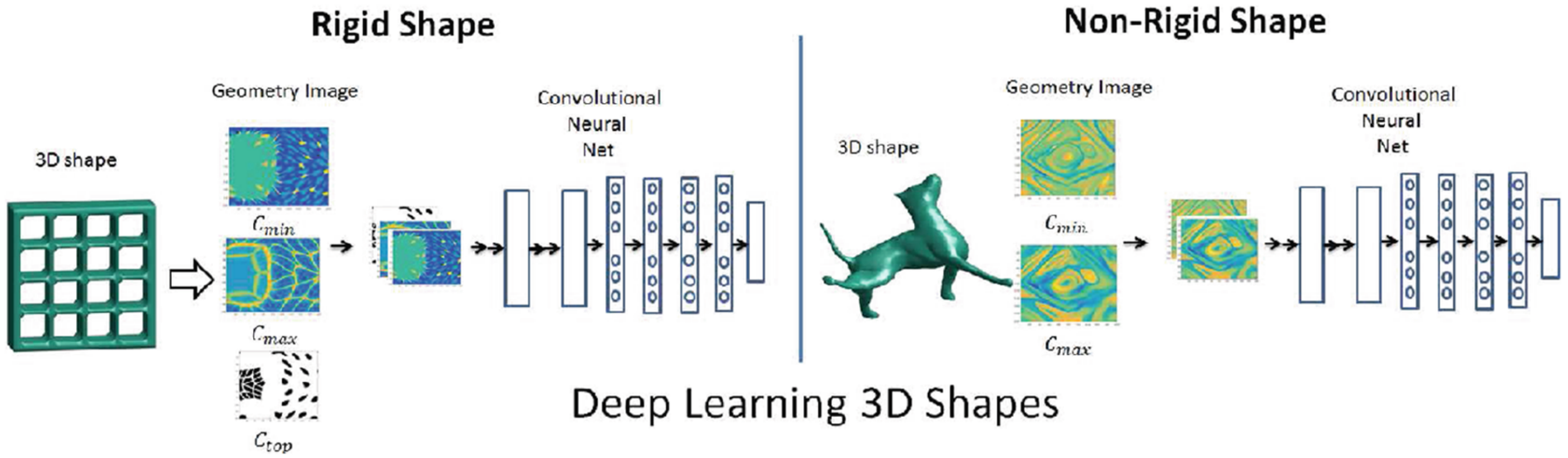


(a) Original mesh with cut 70K faces; genus 0

(b) Geometry image 257×257 (b∗) Compr. to 1.5KB (not shown)

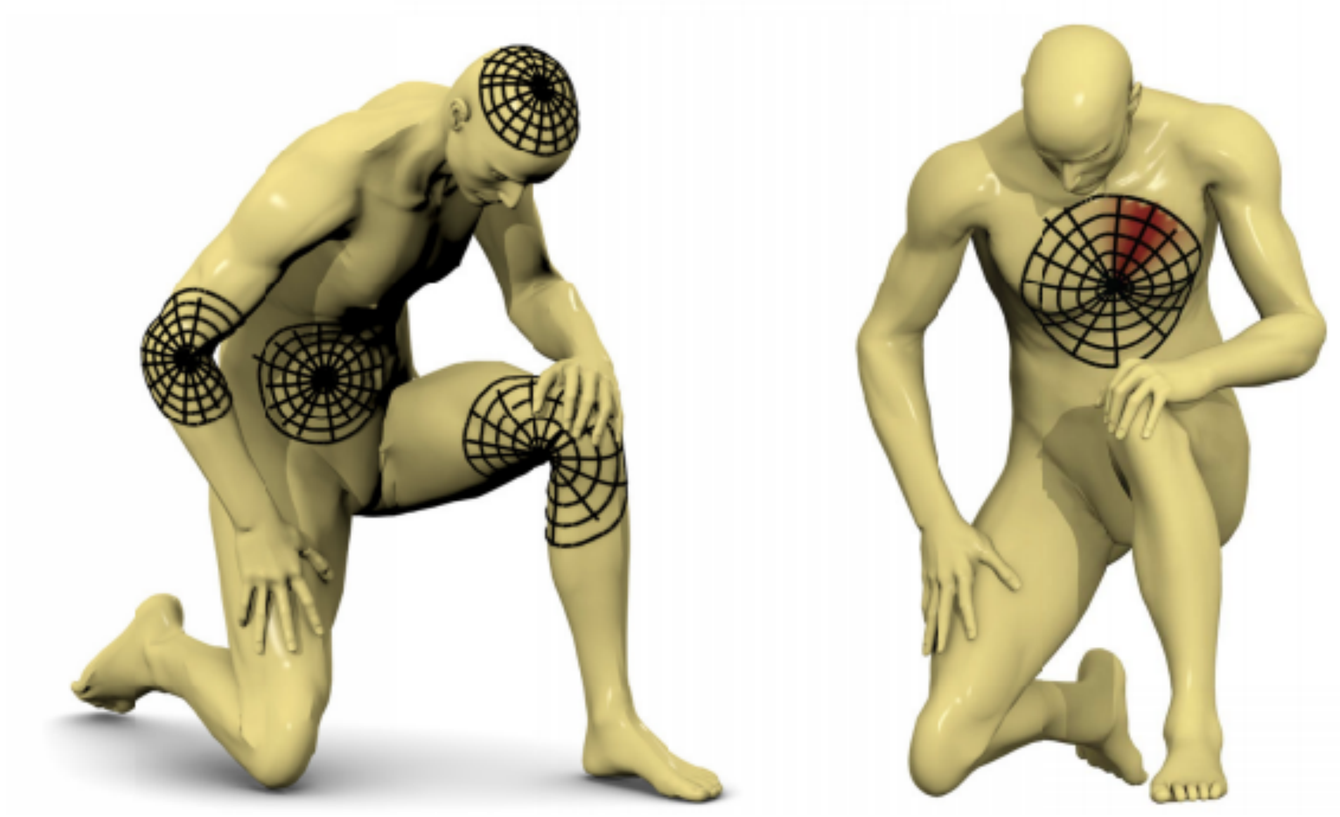
(c) Geometry reconstructed entirely from b

(d) Geometry reconstructed entirely from b∗



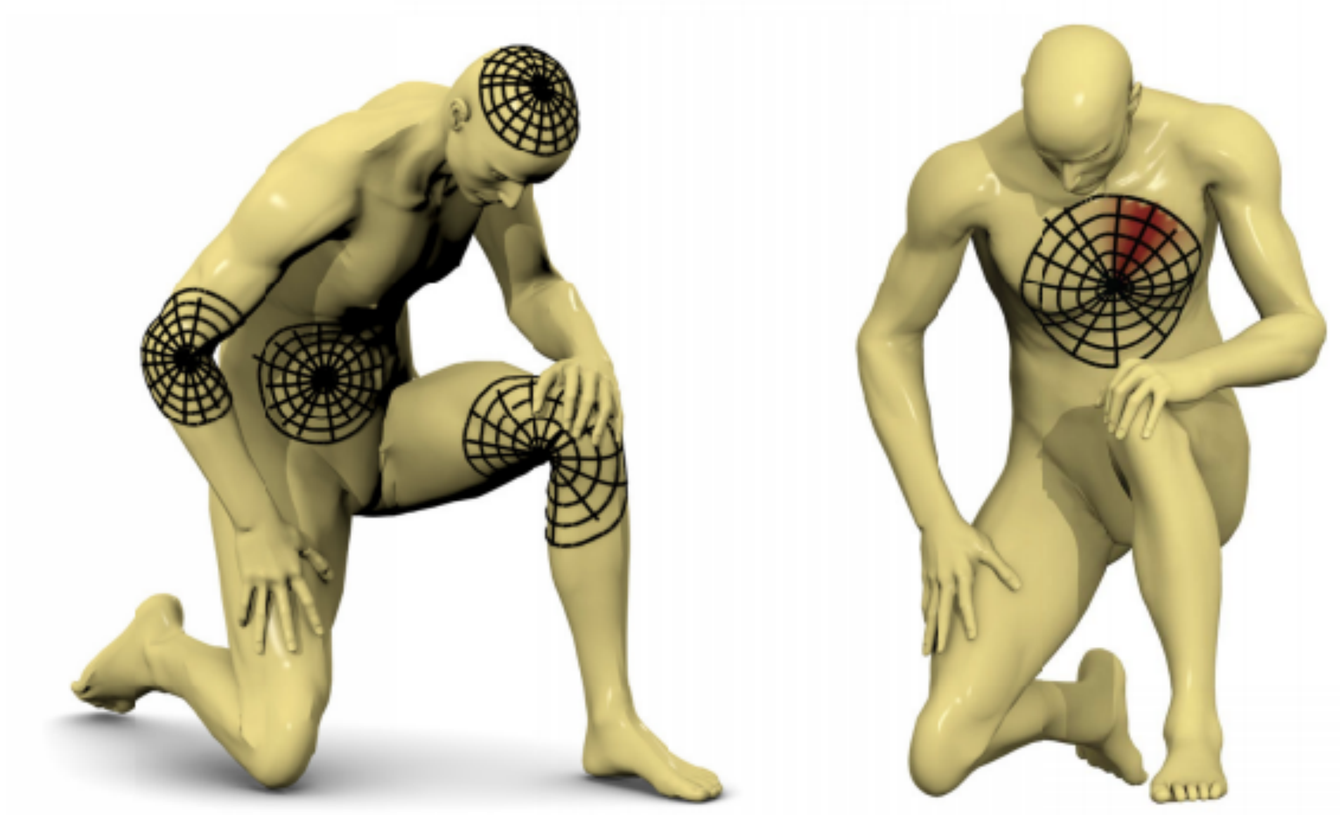
Deep Learning 3D Shapes

Using Geodesic Patches: GCNN



[Masci et al. 2015]

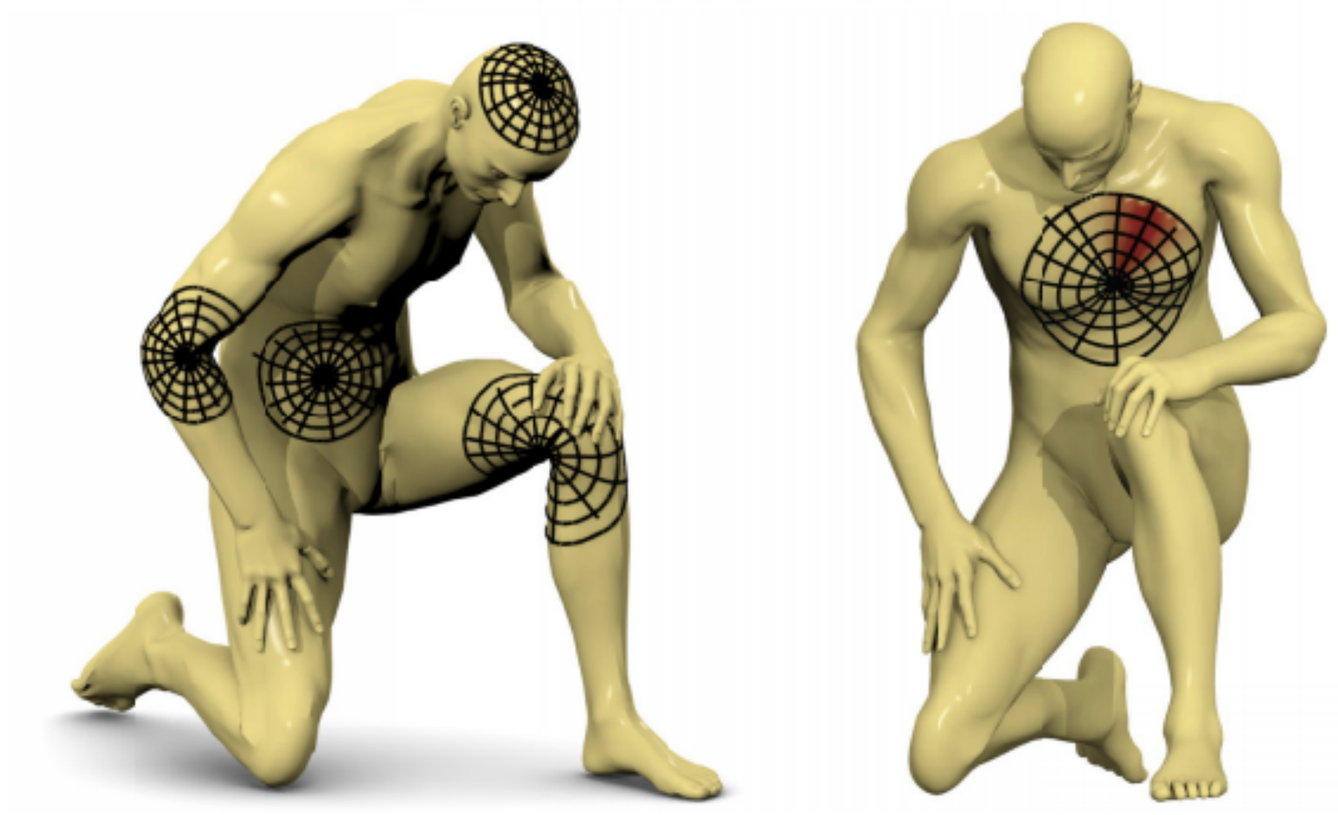
Using Geodesic Patches: GCNN

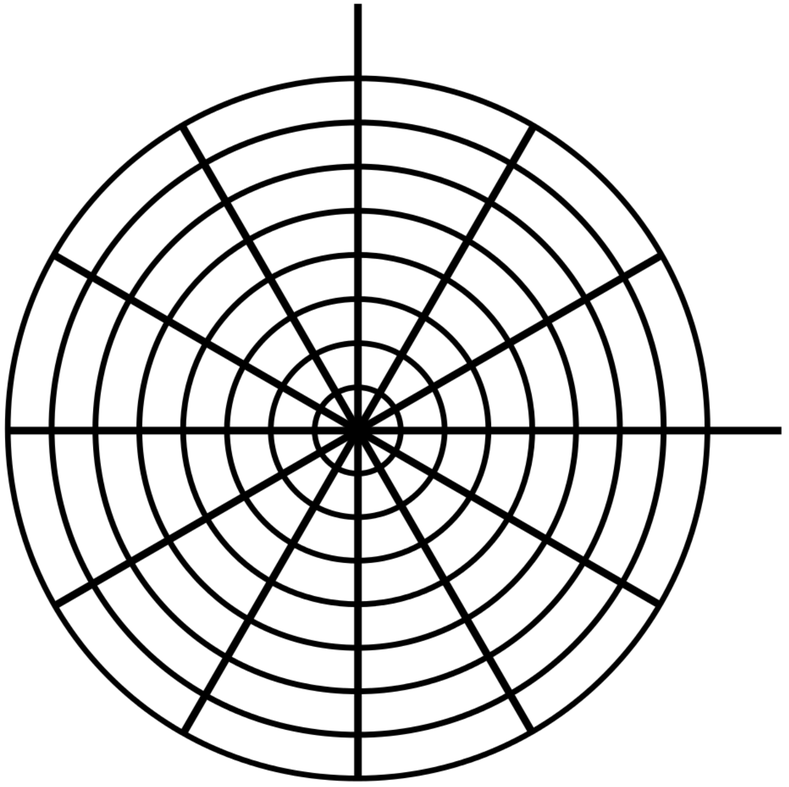


$$(f \star a)(x) := \sum_{\theta,r} a(\theta + \Delta\theta, r)(D(x)f)(r,\theta)$$

[Masci et al. 2015]

Using Geodesic Patches: GCNN

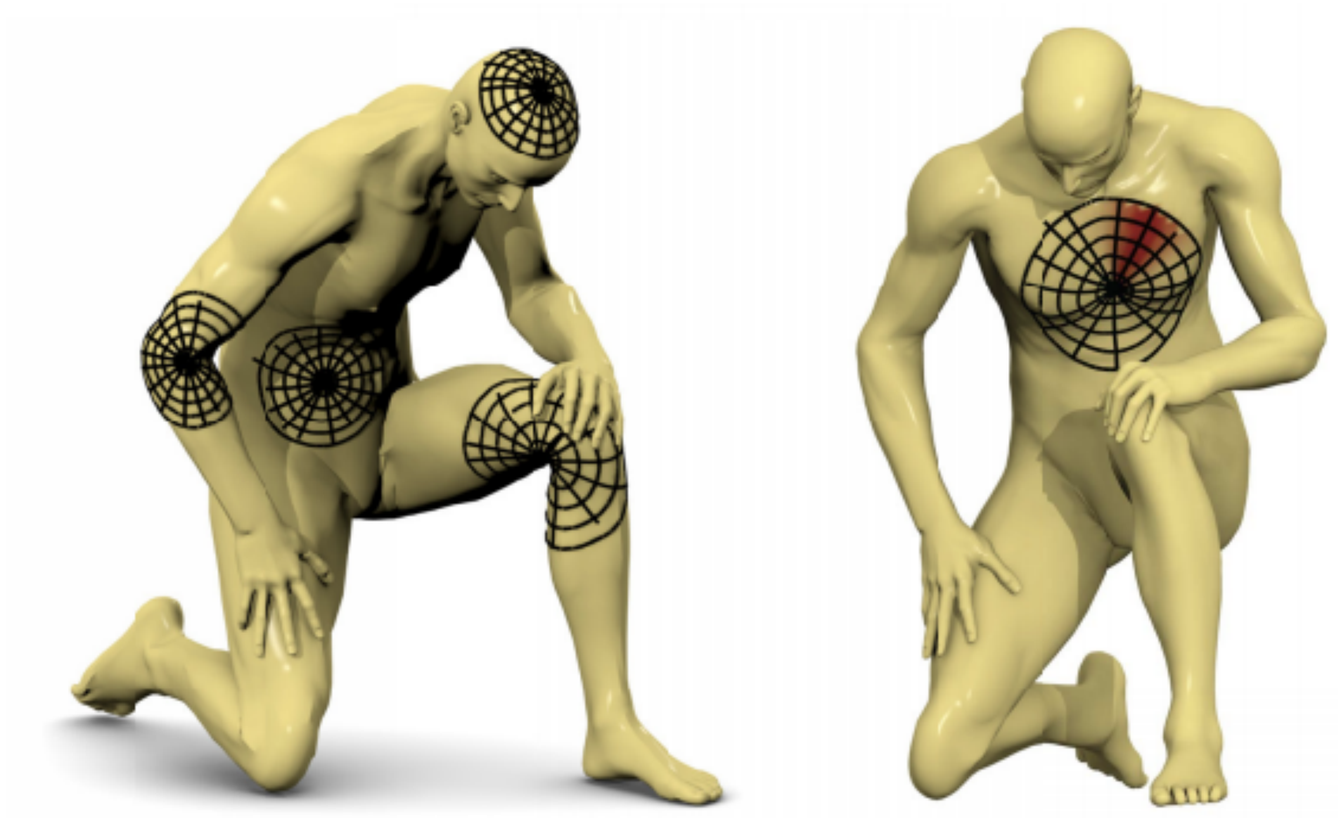


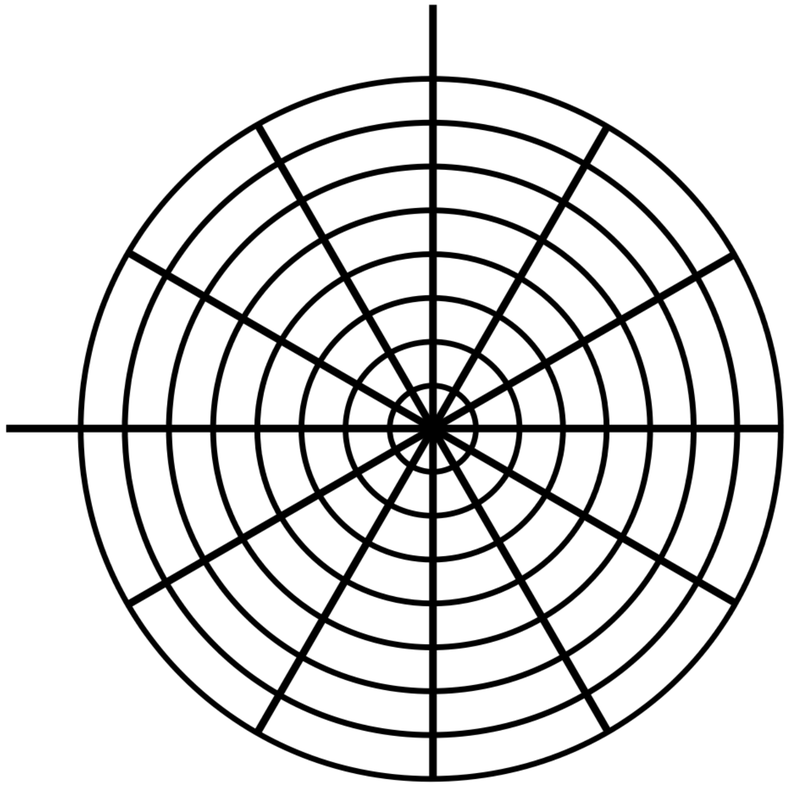


$$(f \star a)(x) := \sum_{\theta,r} a(\theta + \Delta\theta, r)(D(x)f)(r,\theta)$$

[Masci et al. 2015]

Using Geodesic Patches: GCNN

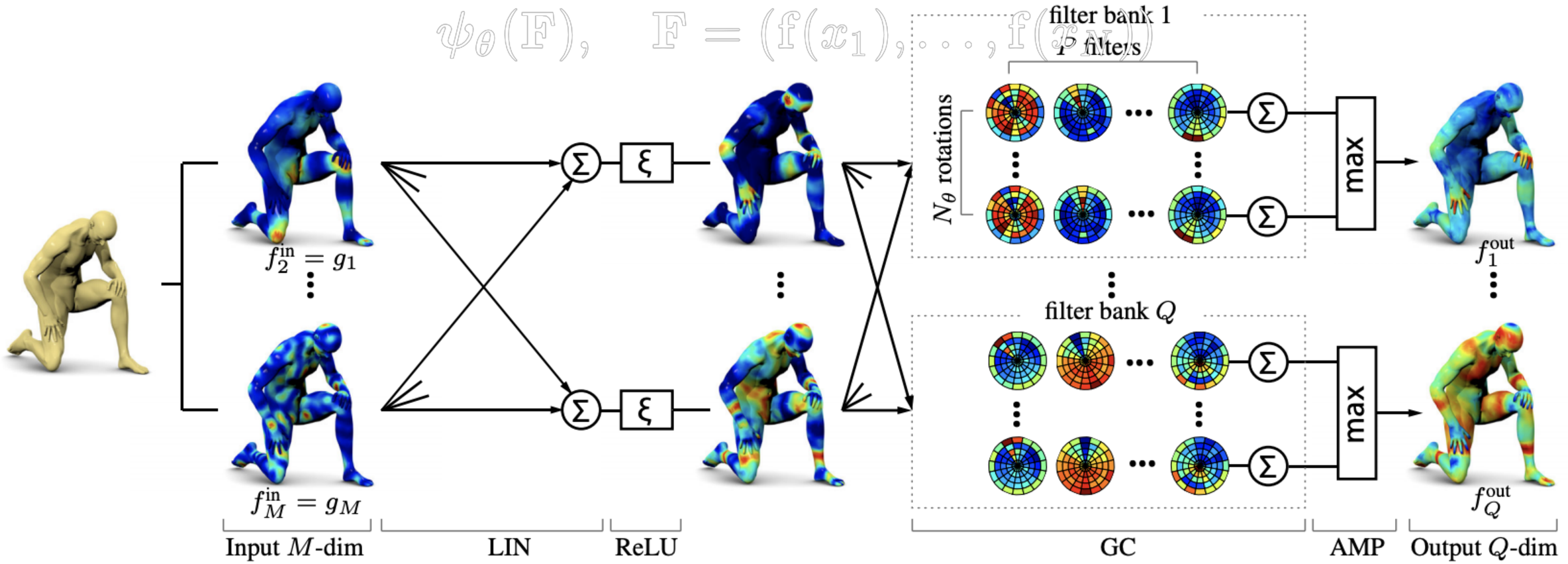




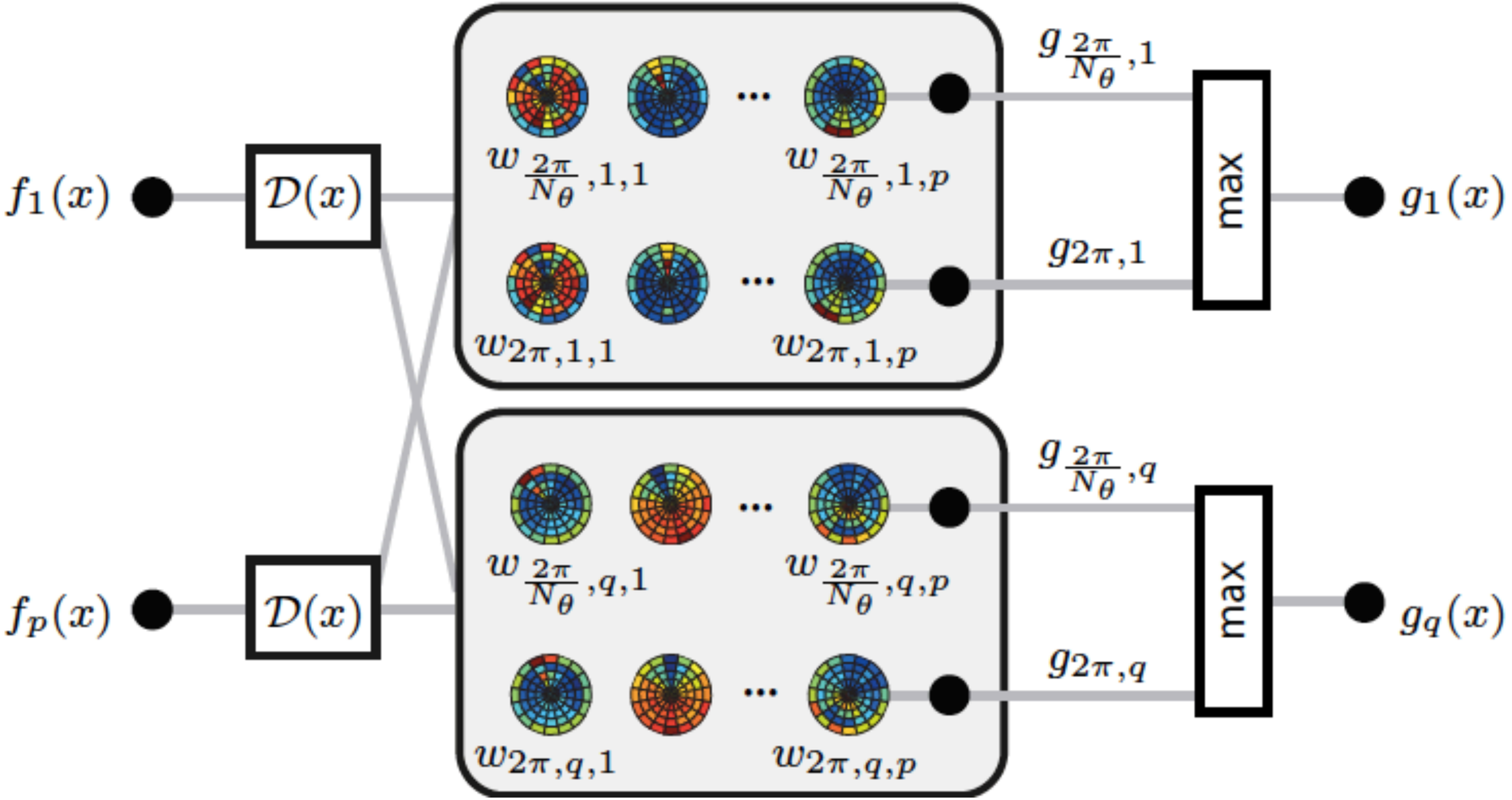
$$(f \star a)(x) := \sum_{\theta,r} a(\theta + \Delta\theta, r)(D(x)f)(r,\theta)$$

[Masci et al. 2015]

GCNN Architecture



Handling Rotational Ambiguity



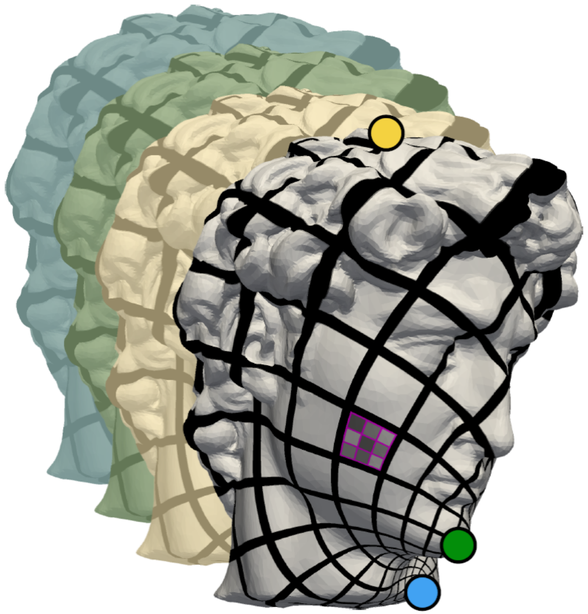
Parameterization for Surface Analysis

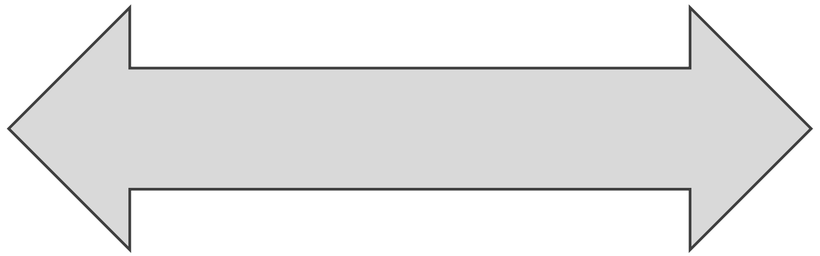
map 3D surface to 2D domain

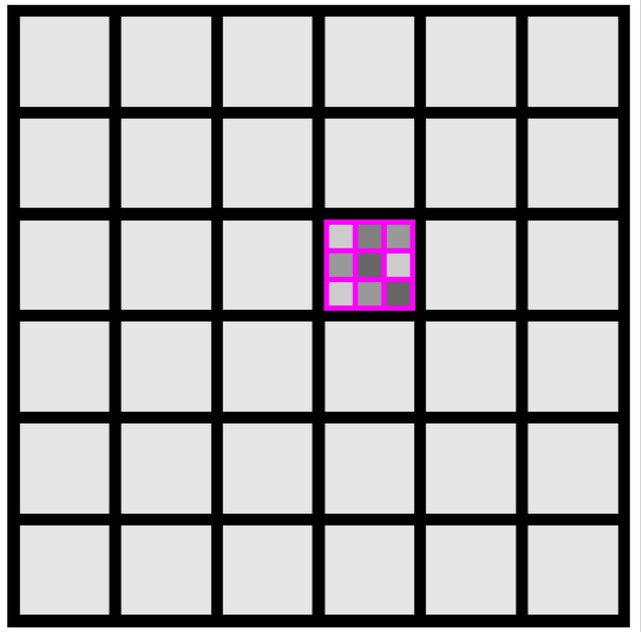
[Maron et al. 2017]

Parameterization for Surface Analysis

map 3D surface to 2D domain

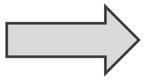


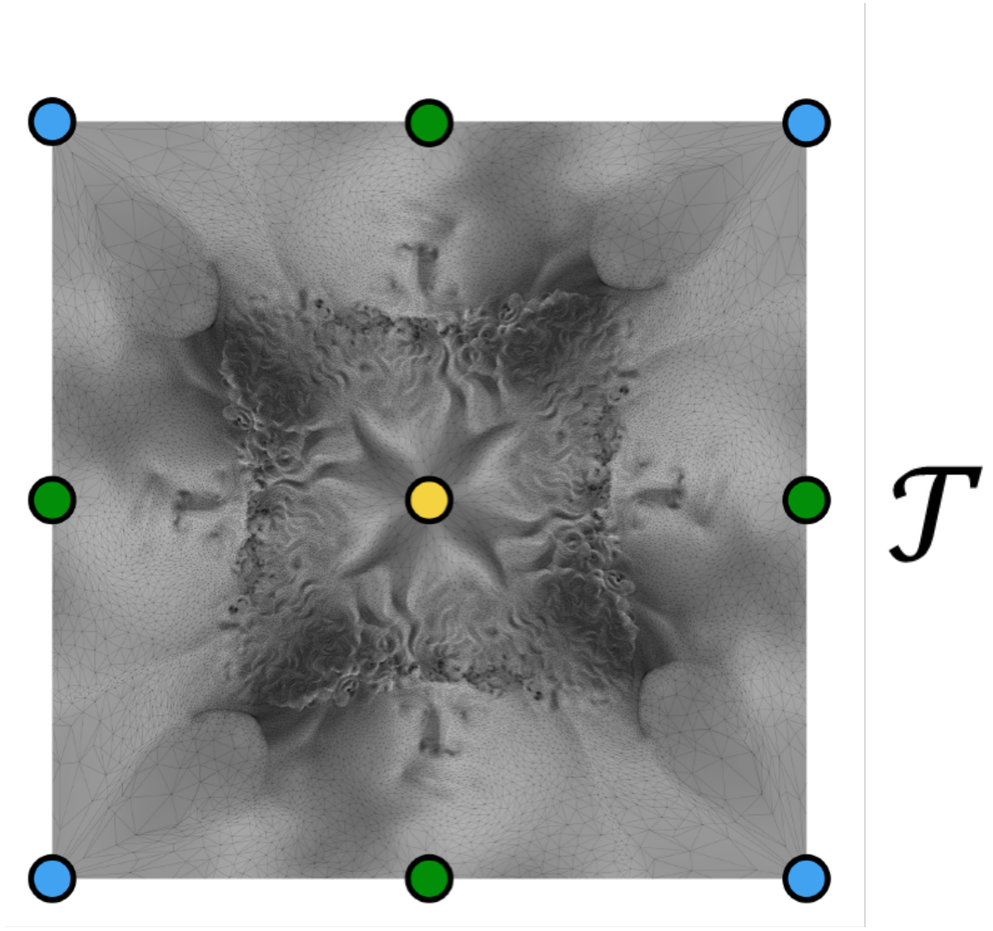




[Maron et al. 2017]

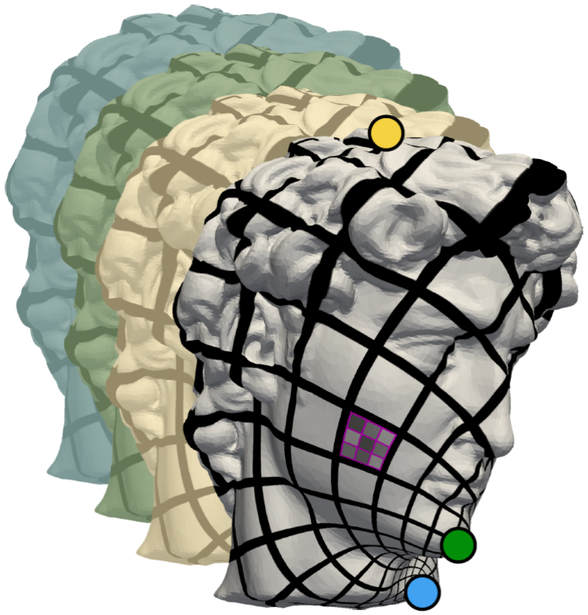
Parameterization for Surface Analysis

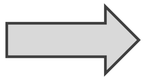


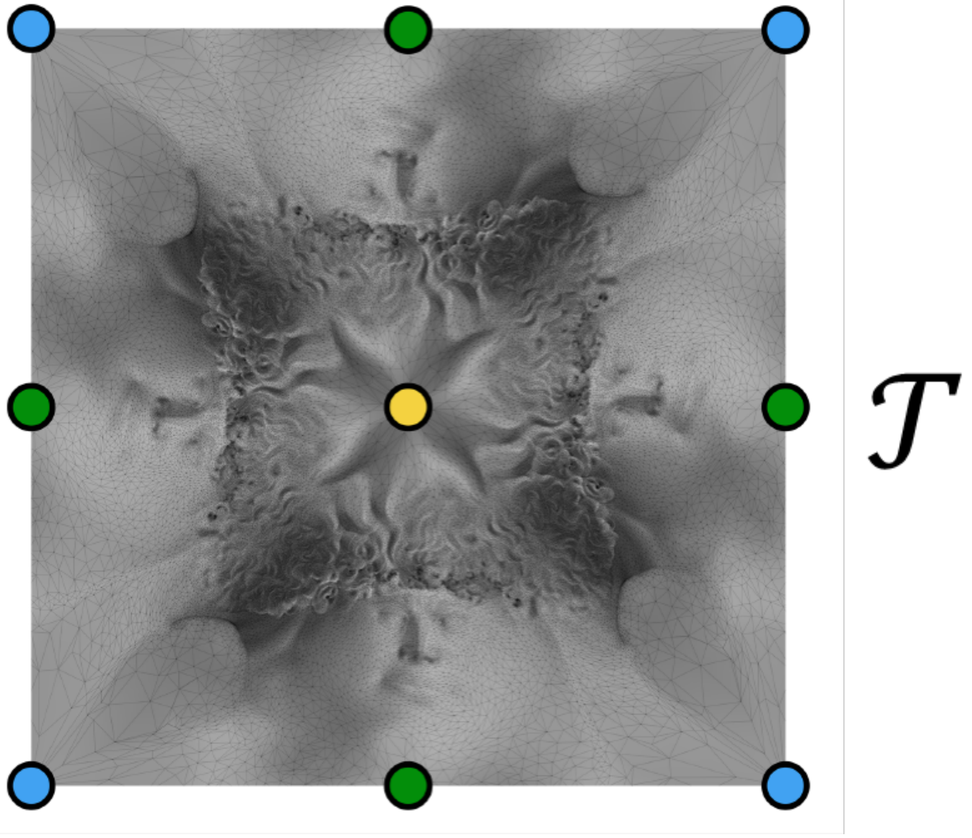


[Maron et al. 2017]

Parameterization for Surface Analysis







$\mathcal{T}$

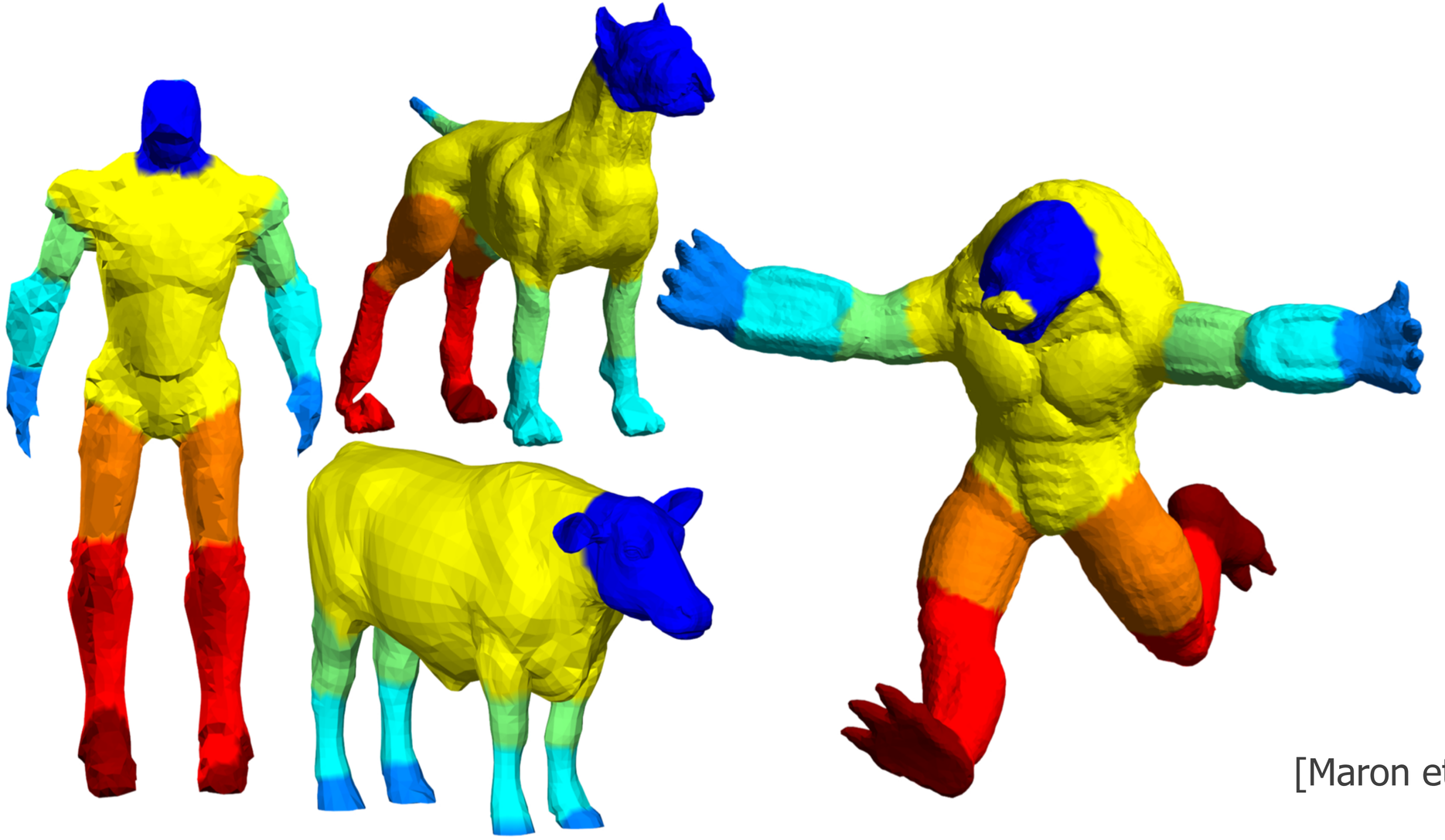
[Maron et al. 2017]

Parameterization for Surface Analysis

- Map 3D surface to 2D domain
  - One such mapping: **flat torus** (seamless => translation-invariant)
  - Many mappings exists: sample a few and average result
  - Which functions to map?
    XYZ, normals, curvature, …

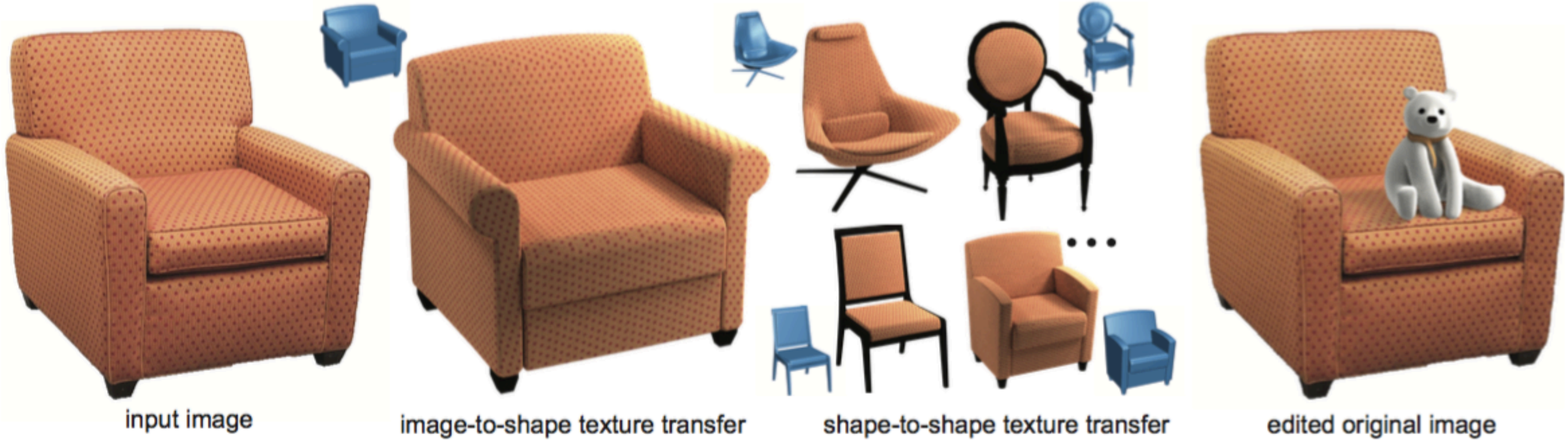
[Maron et al. 2017]

Parameterization for Surface Analysis



[Maron et al. 2017]

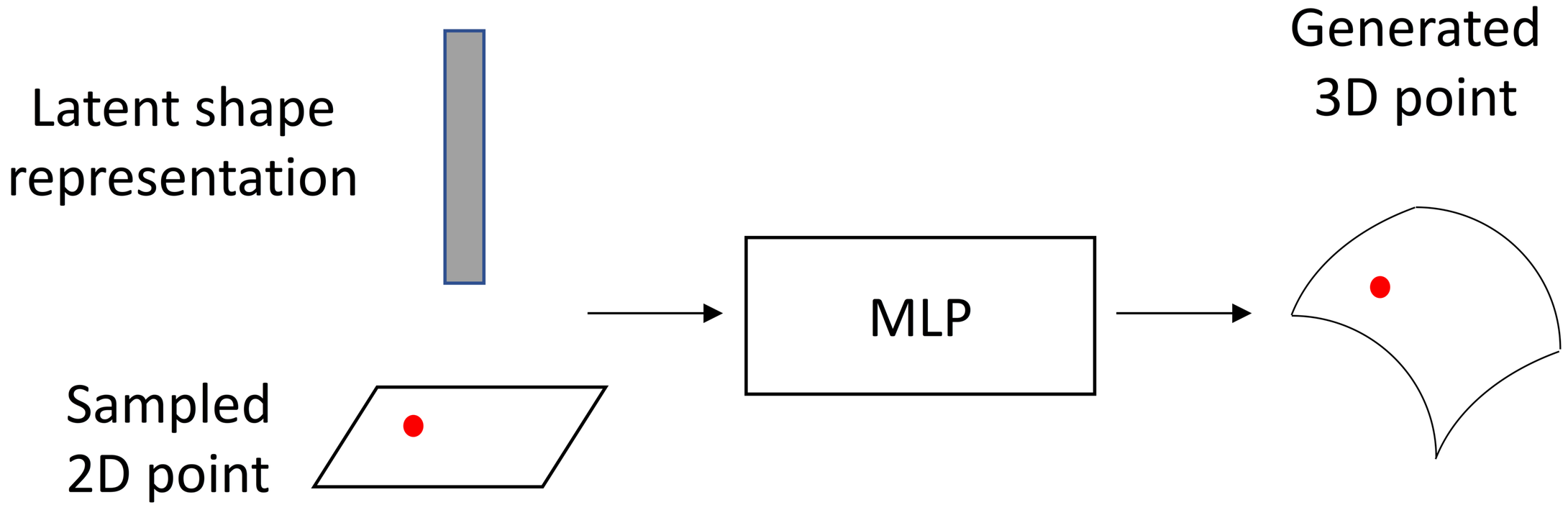
Texture Transfer (Parameterization + Alignment)



[Wang et al. 2016]

AtlasNet for Surface Generation

condition decoded points on 2D patches

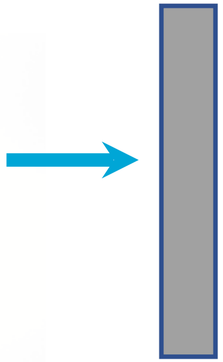


[Groueix et al. 2018]

AtlasNet for Surface Generation

2D Image

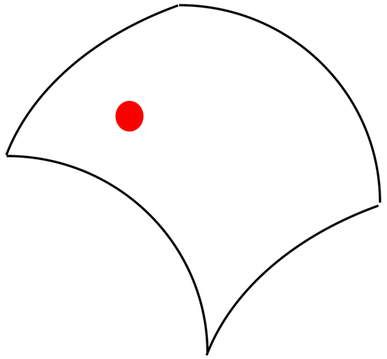
3D Point Cloud



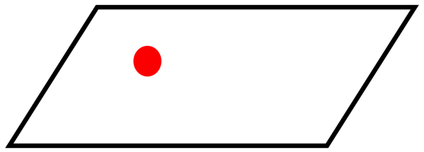
Latent representation can be inferred from images or point clouds

Generated 3D point

MLP



Sampled 2D point



[Groueix et al. 2018]

AtlasNet for Surface Generation

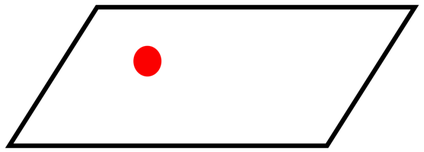
Quad Mesh is generated by mapping a regular grid in 2D domain to 3D points

2D Image

3D Point Cloud

Latent representation can be inferred from images or point clouds

Sampled 2D point



MLP

[Groueix et al. 2018]

AtlasNet for Surface Generation

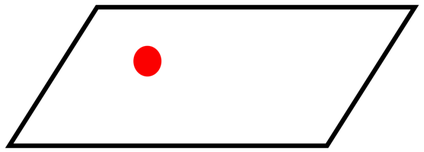
texture coordinates come for free!!

2D Image

3D Point Cloud

Latent representation can be inferred from images or point clouds

Sampled 2D point



MLP

Representation for 3D

- Image-based
- Volumetric
- Surface-based
  - **PROS**: parameterize + image networks (instrinsic representation)
  - **CONS**: suffers from parameterisation artefacts (local versus global distortion), requires good quality mesh
- Point-based

Representation for 3D

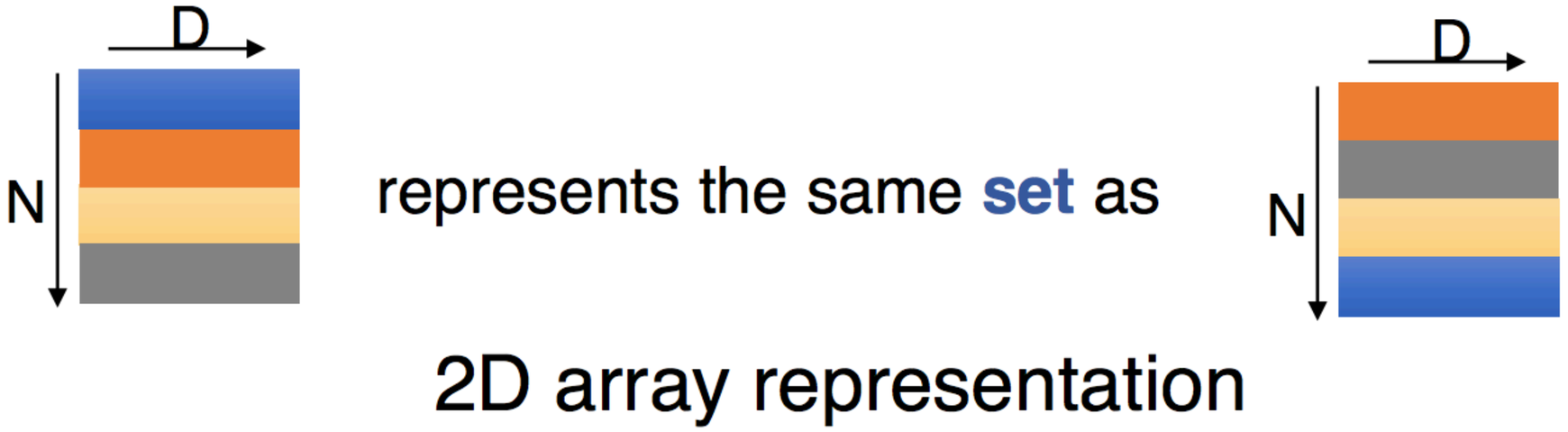
- Image-based
- Volumetric
- Surface-based
- **Point-based**

Representation for 3D: Point-based

- Common representation: native representation
- Easy to obtain from meshes, depth scans, laser scans

In Original Representation

- Common representation
- Easy to obtain from meshes, depth scans, laser scans
- **Unstructured** (e.g., any permutation of points gives same shape!)



[Qi et al. 2017]

PointNet for Point Cloud Analysis

$$f(x_1, x_2, \ldots, x_n) \equiv f(x_{\pi_1}, x_{\pi_2}, \ldots, x_{\pi_n}), \quad x_i \in \mathbb{R}^D$$

permutation-invariant functions

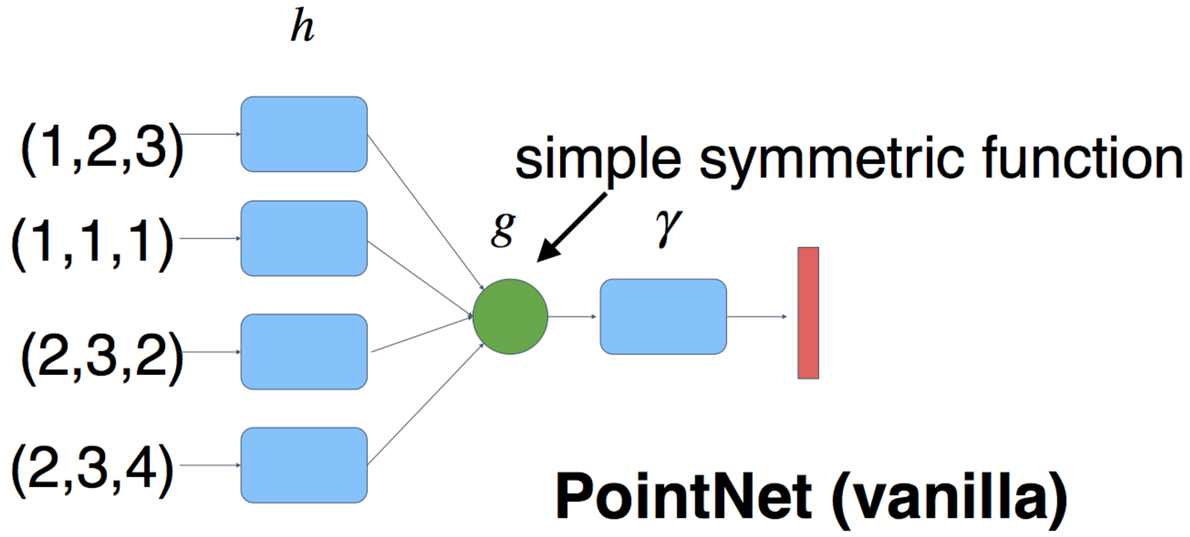
[Qi et al. 2017]

PointNet for Point Cloud Analysis

Use **MLPs** (h) and **max-pooling** (g) as simple symmetric functions

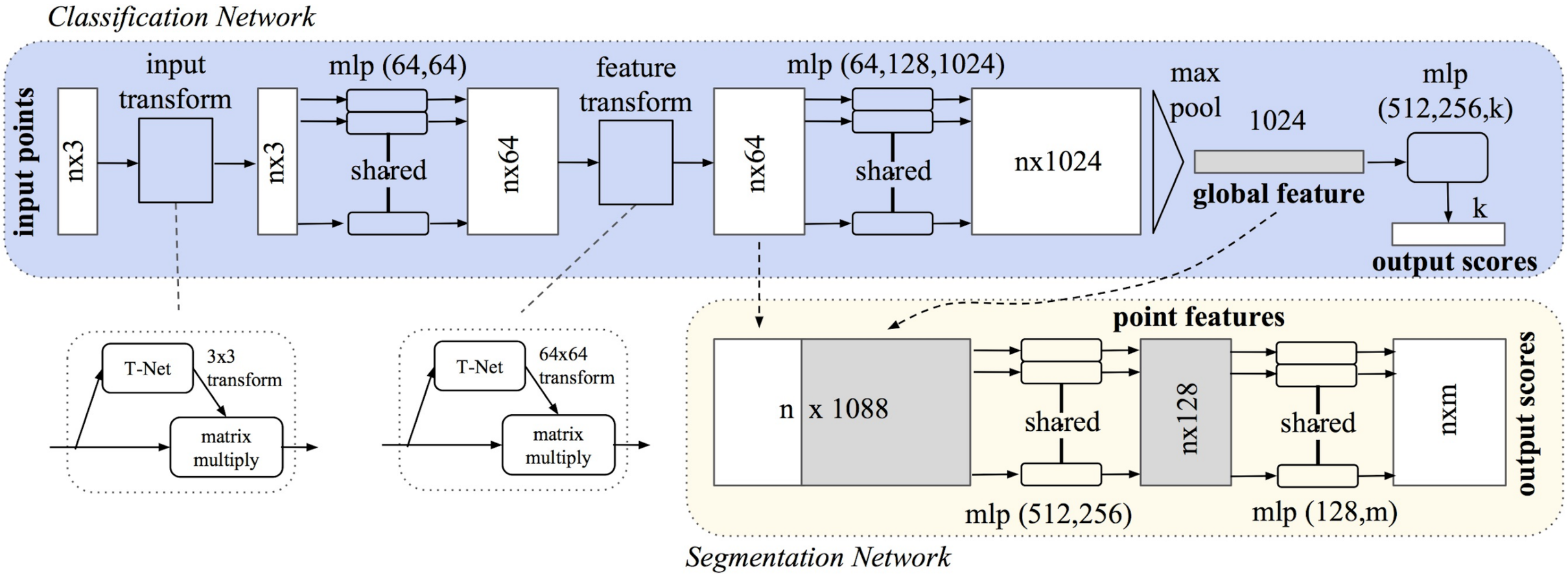
$$f(x_1, x_2, \ldots, x_n) \equiv f(x_{\pi_1}, x_{\pi_2}, \ldots, x_{\pi_n}), \quad x_i \in \mathbb{R}^D$$

$$f(x_1, x_2, \ldots, x_n) = \gamma \circ g(h(x_1), \ldots, h(x_n))$$



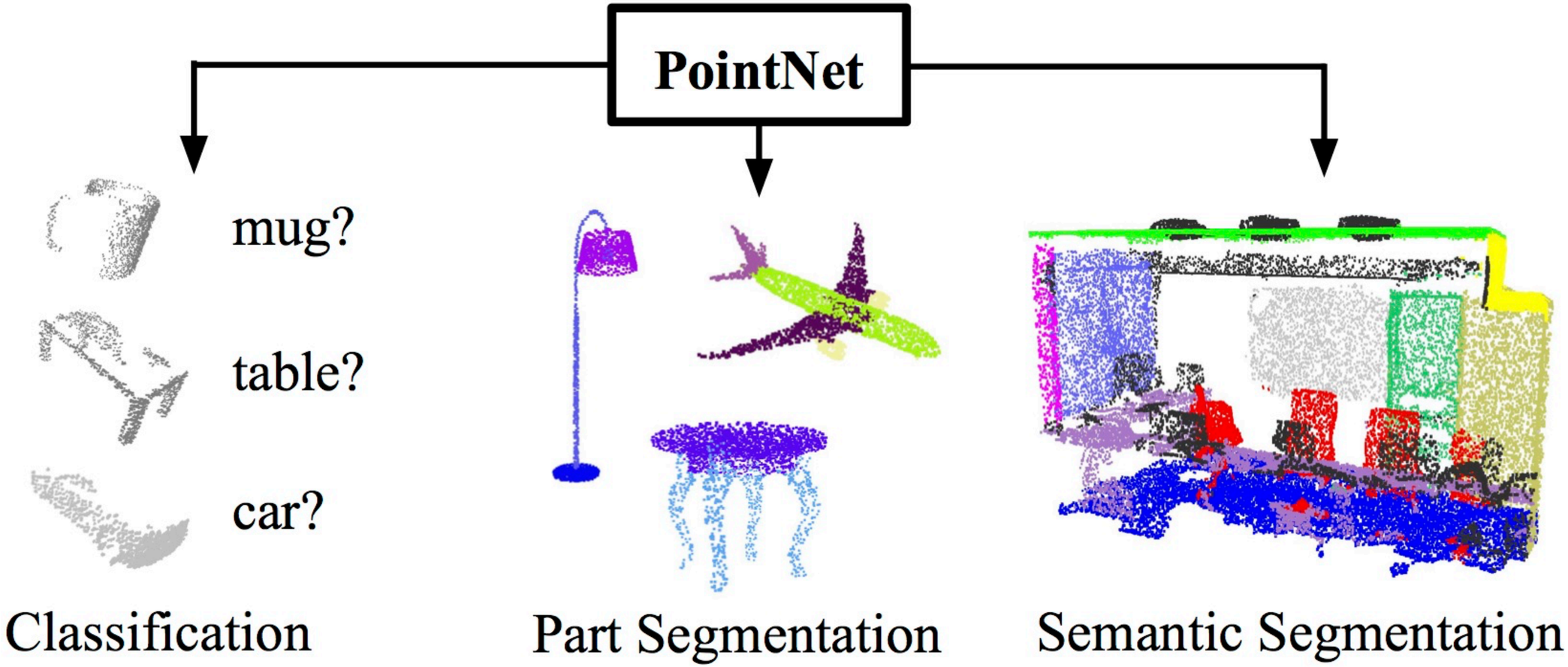
[Qi et al. 2017]

PointNet Architecture

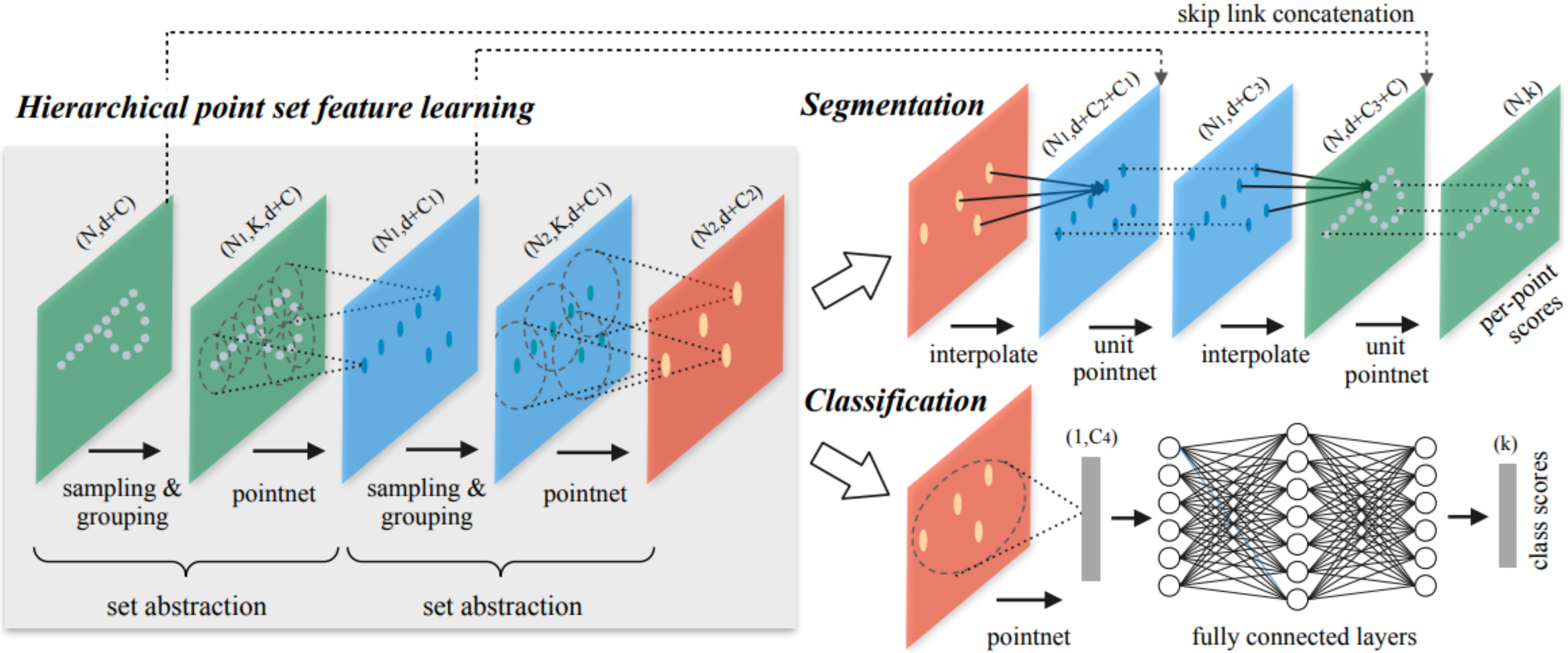


[Qi et al. 2017]

PointNet for Point Cloud Analysis

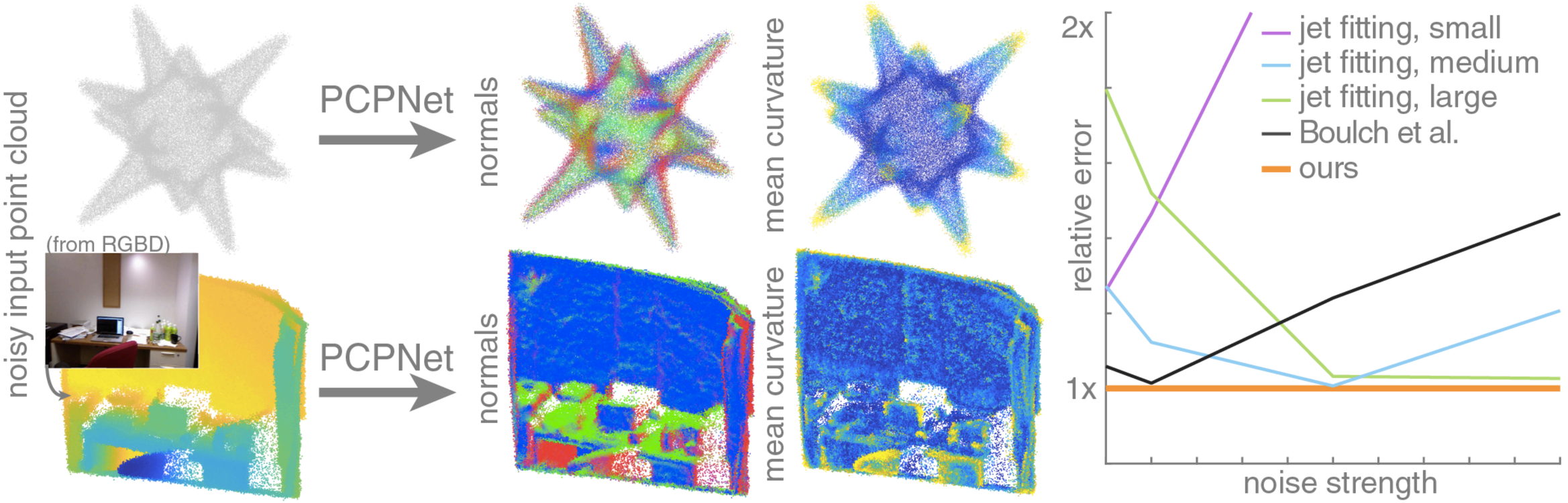


PointNet++



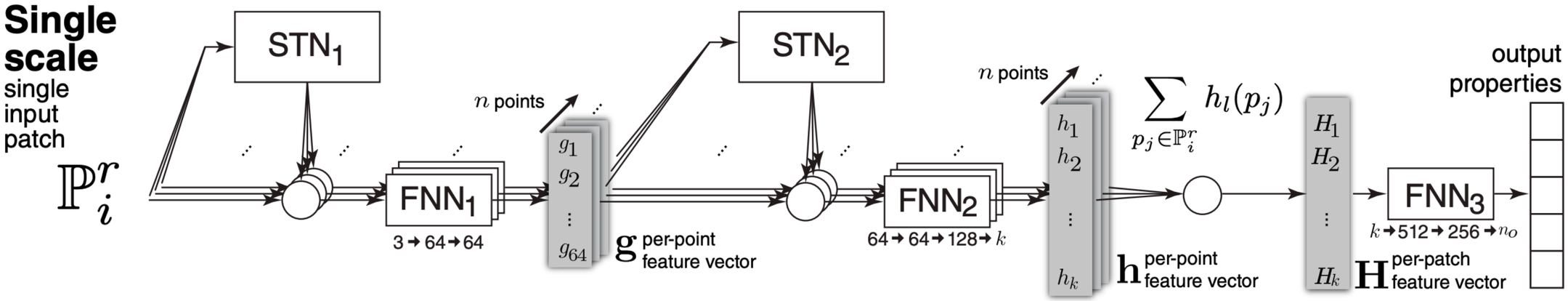
[Qi et al. 2018]

PCPNet for Local Point Cloud Analysis

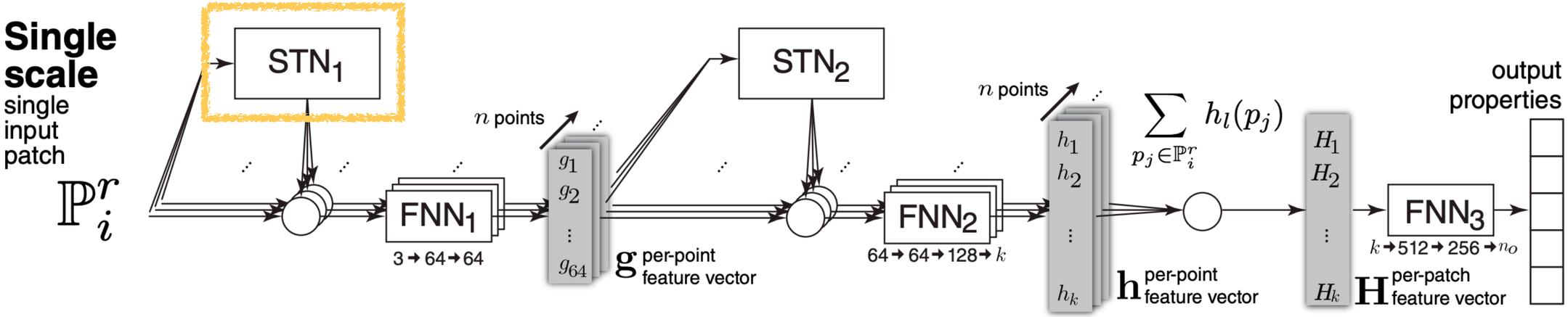


[Guerrero et al. 2018]

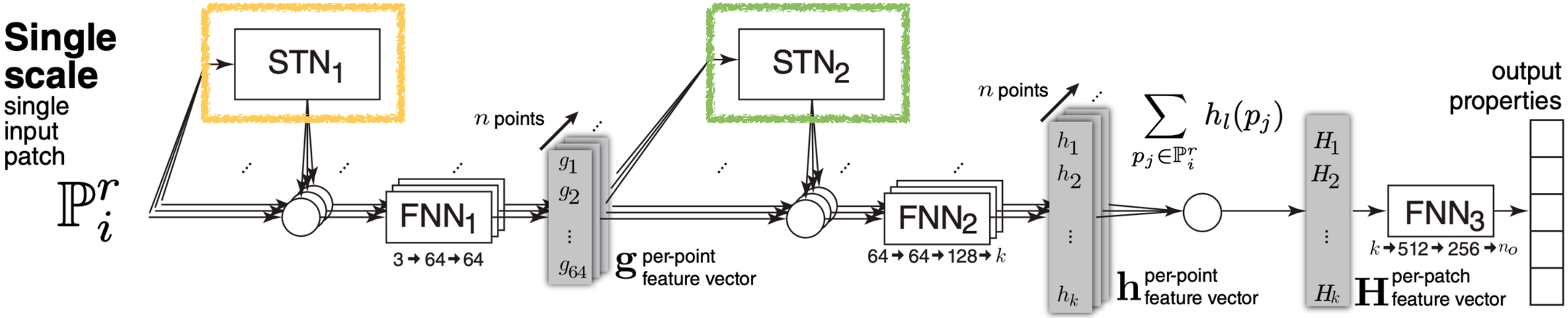
PCPNet Architecture



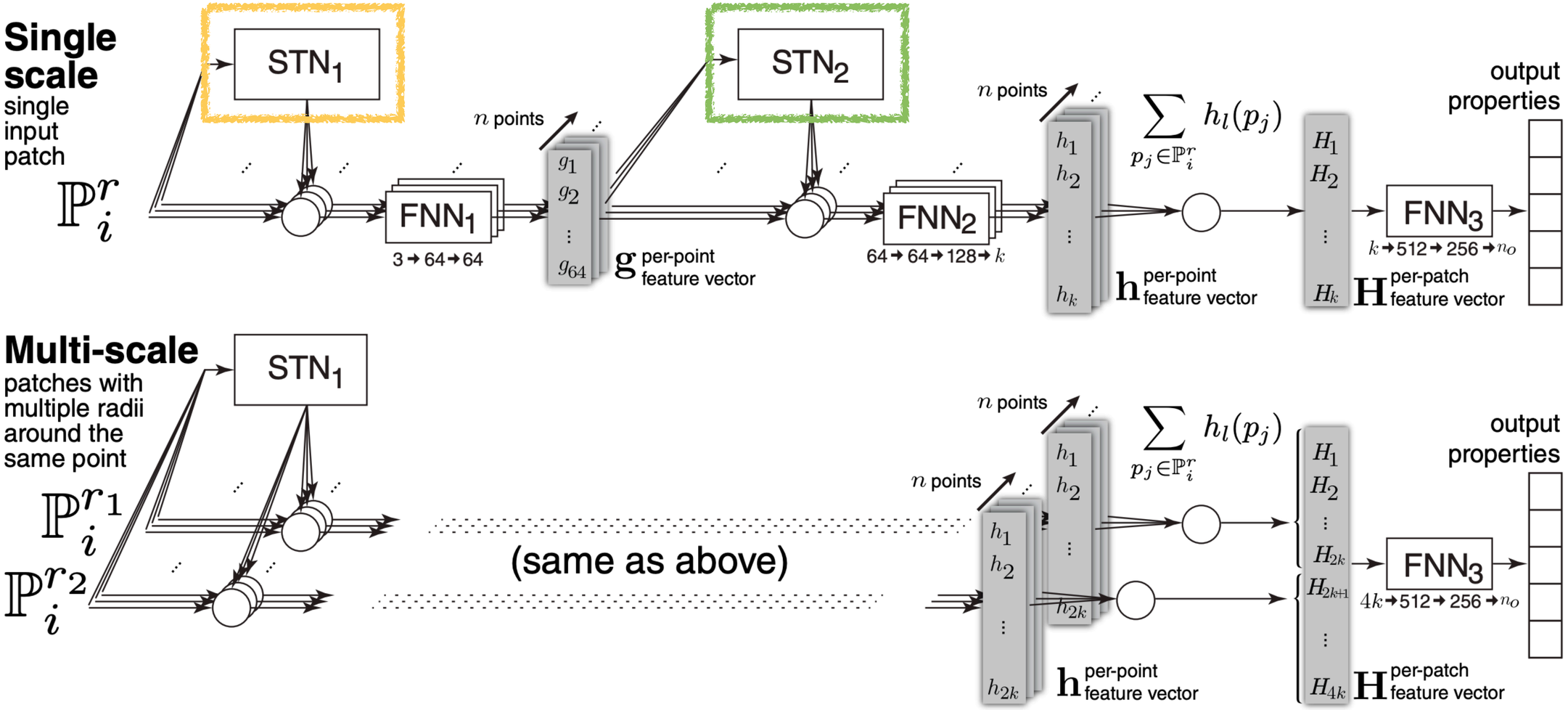
PCPNet Architecture



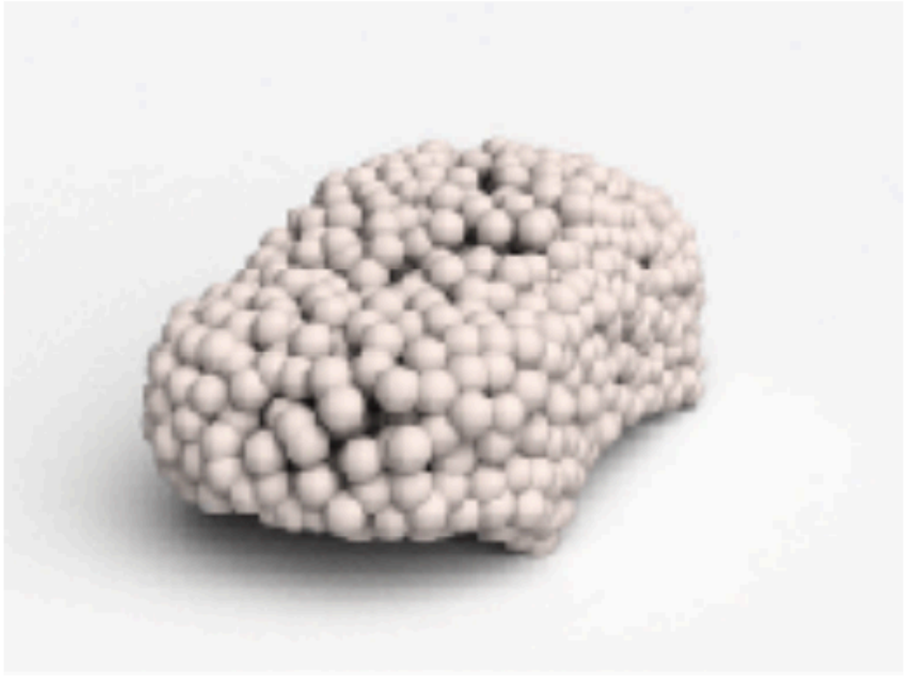
PCPNet Architecture



PCPNet Architecture

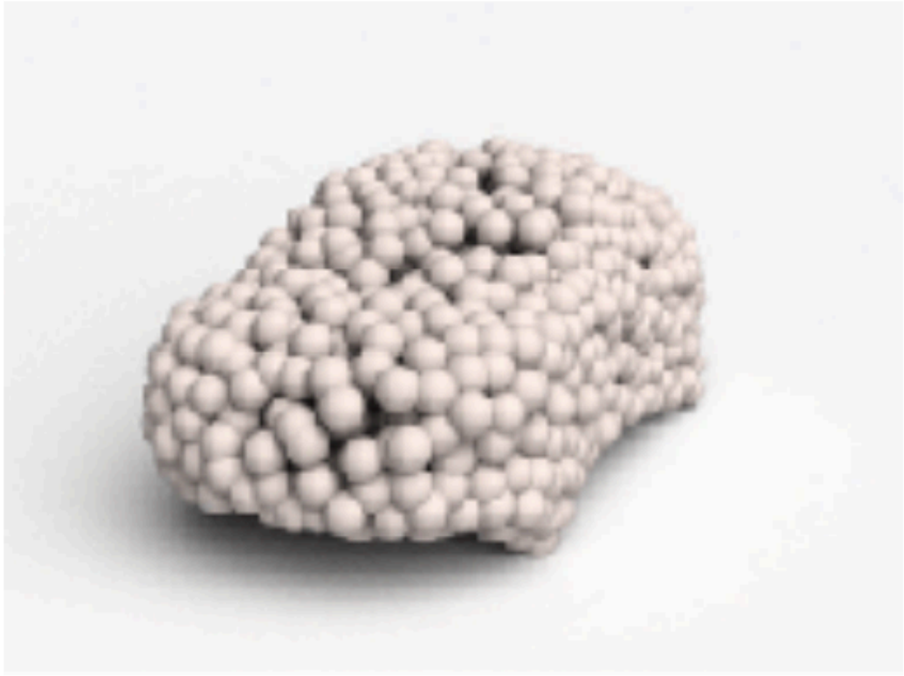


PointNet for Point Cloud Synthesis



[Su et al. 2017]

PointNet for Point Cloud Synthesis

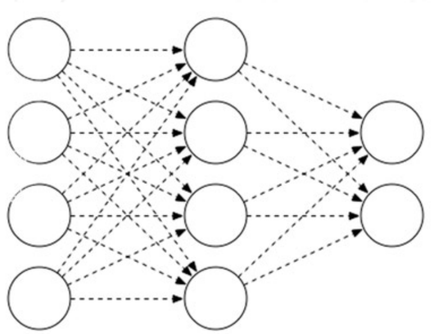


Earth Mover Distance as loss function

$$d_{EMD}(S_1, S_2) = \min_{\phi: S_1 \to S_2} \sum_{x \in S_1} \|x - \phi(x)\|_2$$

[Su et al. 2017]

Course Information (slides/code/comments)



http://geometry.cs.ucl.ac.uk/creativeai/

CreativeAI: Deep Learning for Graphics

Motion & Physics

Niloy Mitra
UCL

Iasonas Kokkinos
UCL/Facebook

Paul Guerrero
UCL

Nils Thuerey
TUM

Tobias Ritschel
UCL

facebook
Artificial Intelligence Research

Technische Universität München

Computer Animation

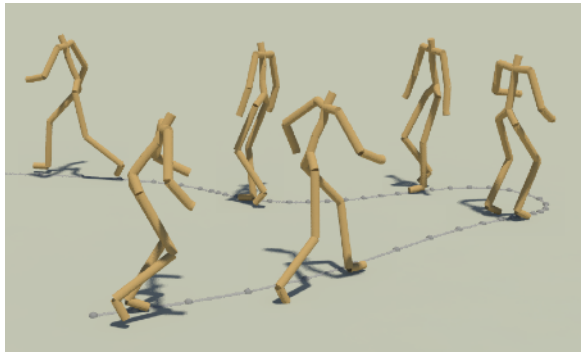
- Feature detection (image features, point features) $\mathbb{R}^{m \times m} \rightarrow \mathbb{Z}$
- Denoising, Smooth $\mathbb{R}^{m \times m} \rightarrow \mathbb{R}^{m \times m}$
- Embedding, Distar $\mathbb{R}^{m \times m, m \times m} \rightarrow \mathbb{R}^{d}$
- Rendering $\mathbb{R}^{m \times m} \rightarrow \mathbb{R}^{m \times m}$
- Animation $\mathbb{R}^{3m \times t} \rightarrow \mathbb{R}^{3m}$
- Physical simulation $\mathbb{R}^{3m \times t} \rightarrow \mathbb{R}^{3m}$
- Generative models $\mathbb{R}^{d} \rightarrow \mathbb{R}^{m \times m}$

- Motion over time
- Loads of data, expensive
- Relationships between spatial and temporal changes

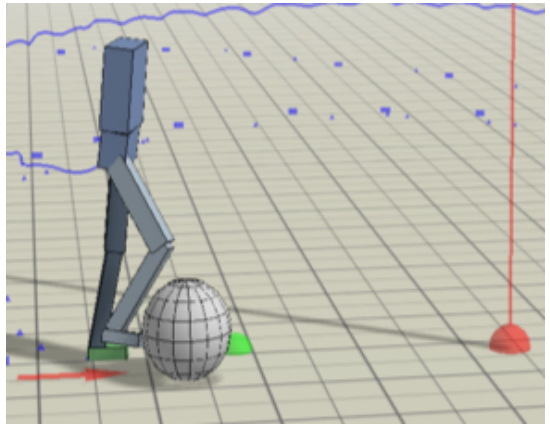
Character Animation

- Learn controllers for character rigs
- Powerful and natural
- Beyond the scope of this course…

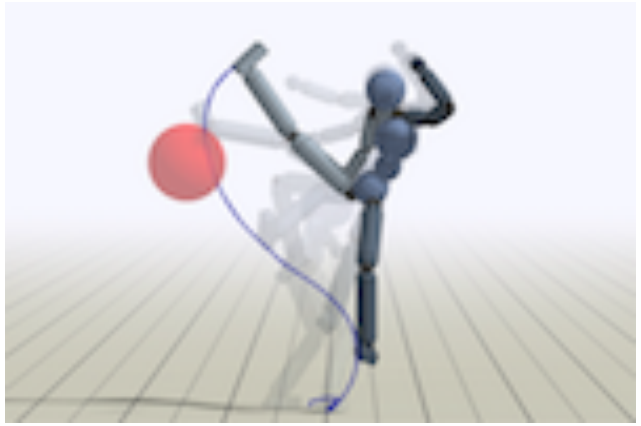
[Mode-Adaptive Neural Networks for Quadruped Motion Control, SIGGRAPH 2018]



[A Deep Learning Framework for Character Motion Synthesis and Editing, SIGGRAPH 2016]



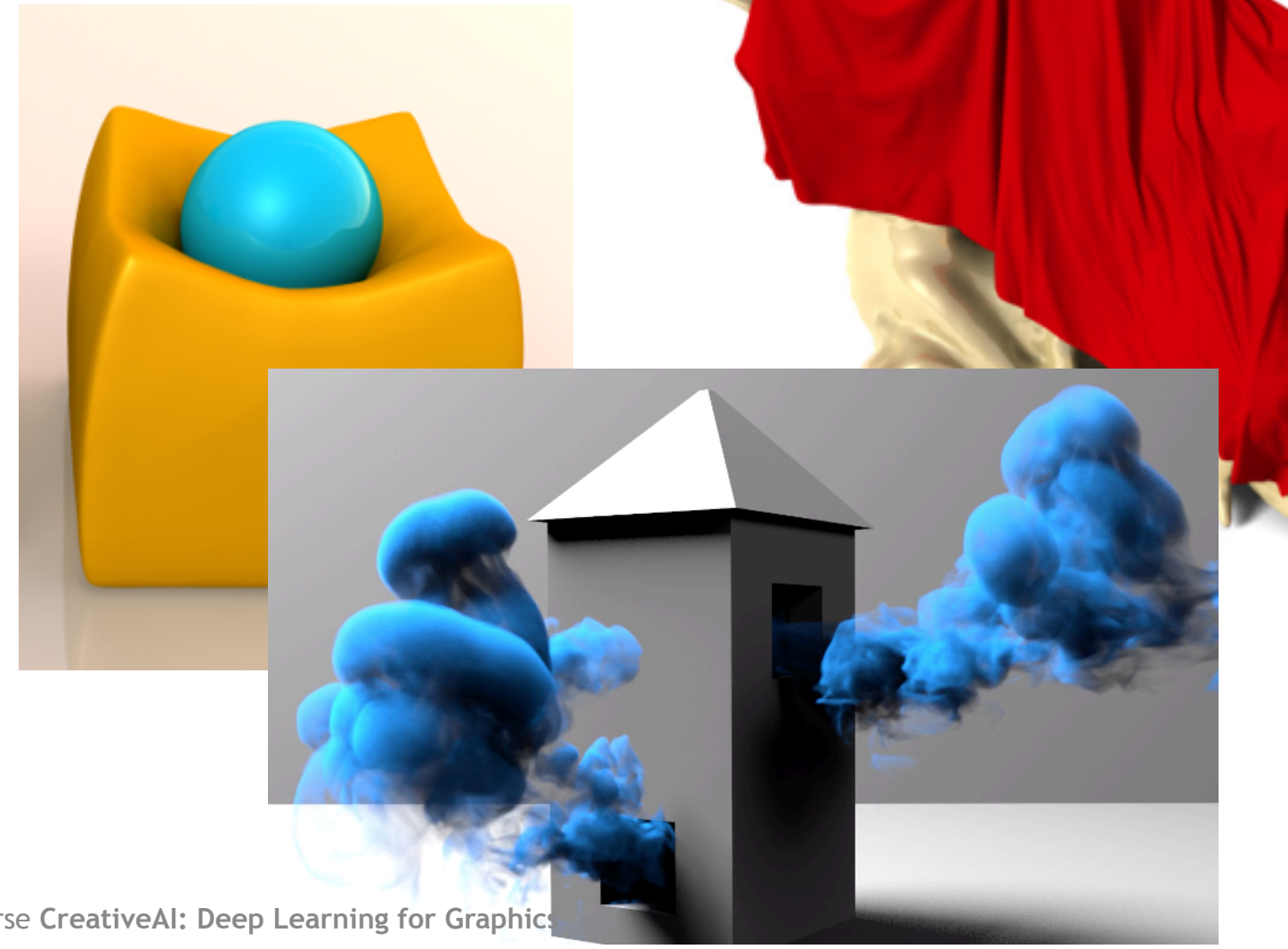
[DeepLoco: Dynamic Locomotion Skills Using Hierarchical Deep Reinforcement Learning, SIGGRAPH 2017]



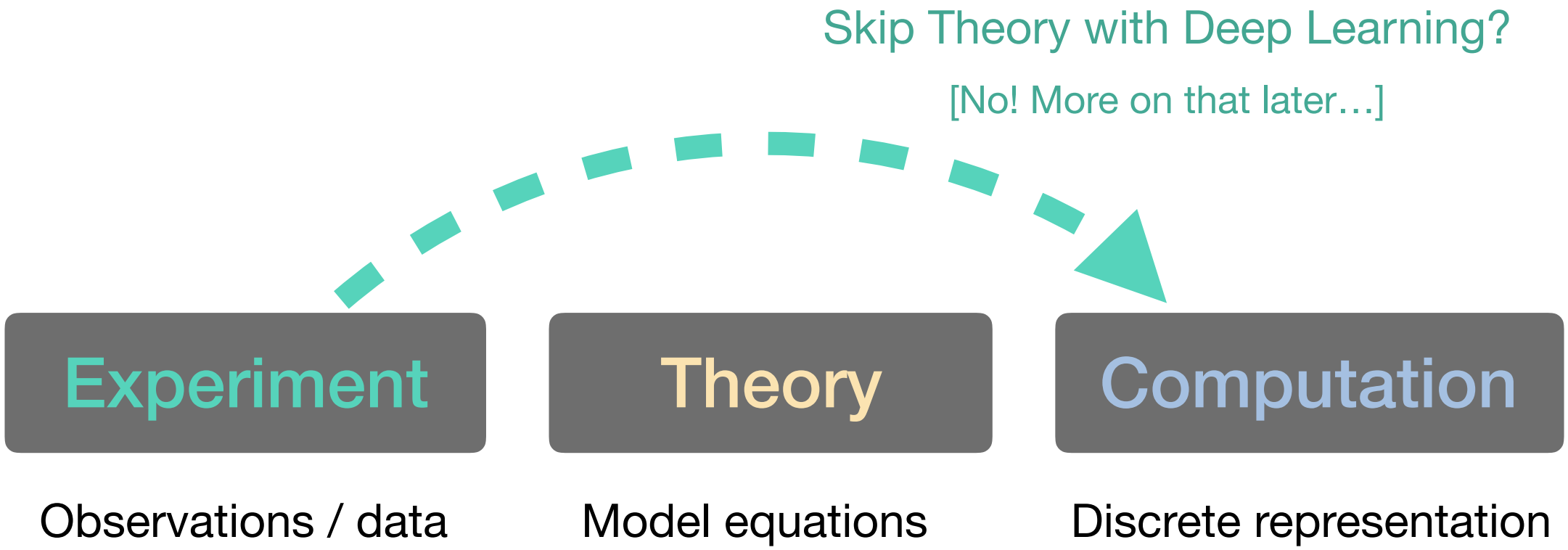
[DeepMimic: Example-Guided Deep Reinforcement Learning of Physics-Based Character Skills, SIGGRAPH 2018]

Physics-Based Animation

- Leverage *physical models*
- Examples:
  - Rigid bodies
  - Cloth
  - Deformable objects
  - Fluids



Physics-Based Animation



Physics-Based Animation

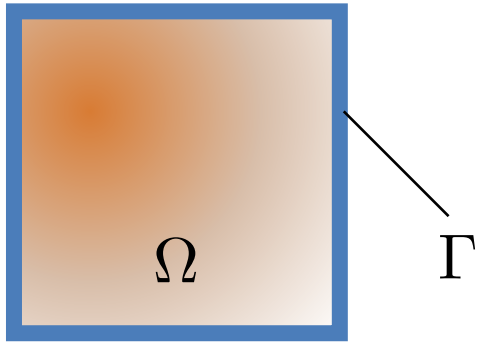
- Better goal: support solving suitable physical models
- Nature = Partial Differential Equations (PDEs)
- Hence we are aiming for solving PDEs with deep learning (DL)
- Requirement: "regularity" of the targeted function

"Bypass the solving of evolution equations when these equations conceptually exist but are not available or known in closed form." [Kevrekidis et al.]

Partial Differential Equations

- Typical problem formulation: unknown function $u(x_1, ..., x_n)$
- PDE of the general form:

$$f\left(x_1, ..., x_n; \frac{\partial u}{\partial x_1}, ..., \frac{\partial u}{\partial x_n}; \frac{\partial^2 u}{\partial^2 x_1}, \frac{\partial^2 u}{\partial x_1 \partial x_2}, ...\right) = 0$$



- Solve in domain Ω , with boundary conditions on boundary Γ
- Traditionally: discretize & solve numerically. Here: also discretize, but solve with DL…

Methodology 1

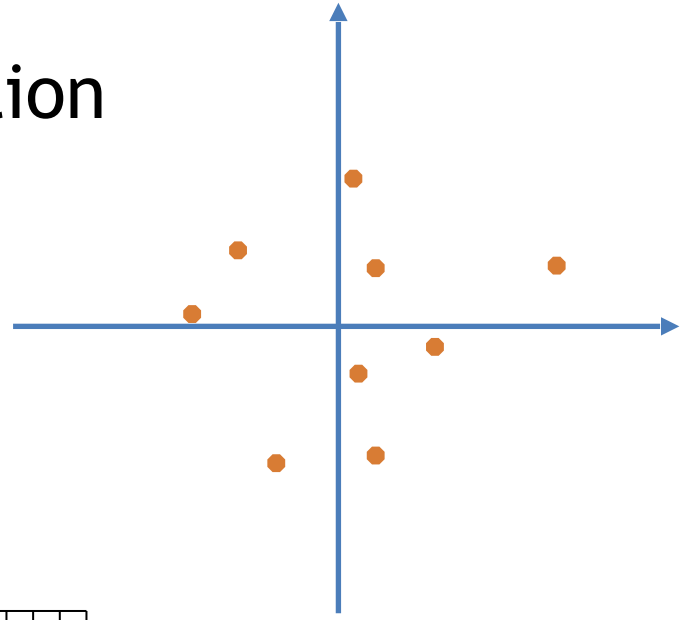
- Viewpoints: *holistic* or *partial*

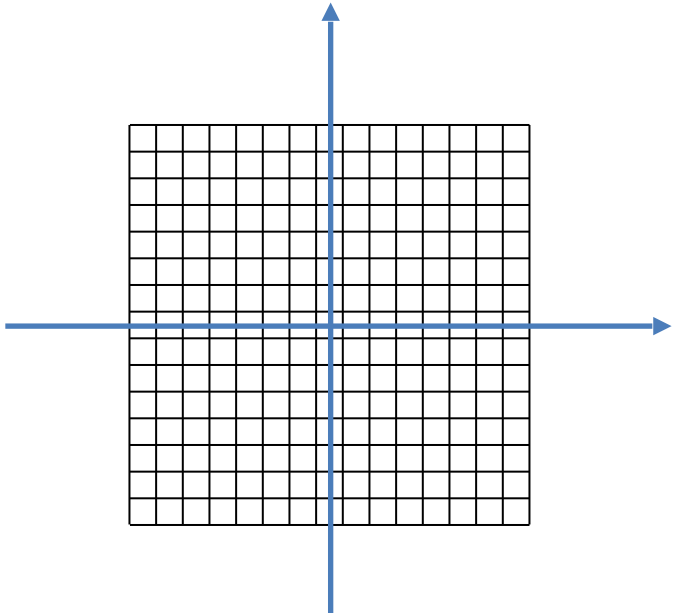
  [partial also meaning “coarse graining” or “sub-grid / up-res”]

- Influences complexity and non-linearity of solution space
- Trade off computation vs accuracy:
  - Target most costly parts of solving
  - Often at the expense of accuracy

Methodology 2

- Consider dimensionality & structure of discretization
- Small & unstructured
  - Fully connected NNs only choice
  - Only if necessary…
- Large & structured
  - Employ convolutional NNs
  - Usually well suited



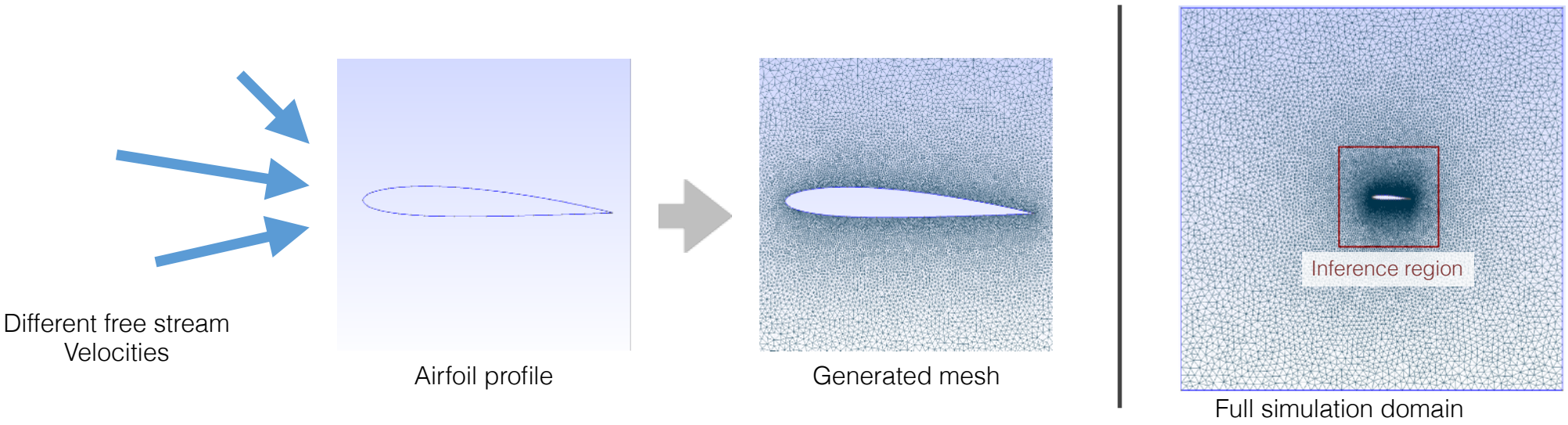


Solving PDEs with DL

- Practical example: *airfoil flow*
  - Given boundary conditions solve stationary flow problem on grid
  - Fully replace traditional solver
  - 2D data, no time dimension
  - I.e., holistic approach with structured data

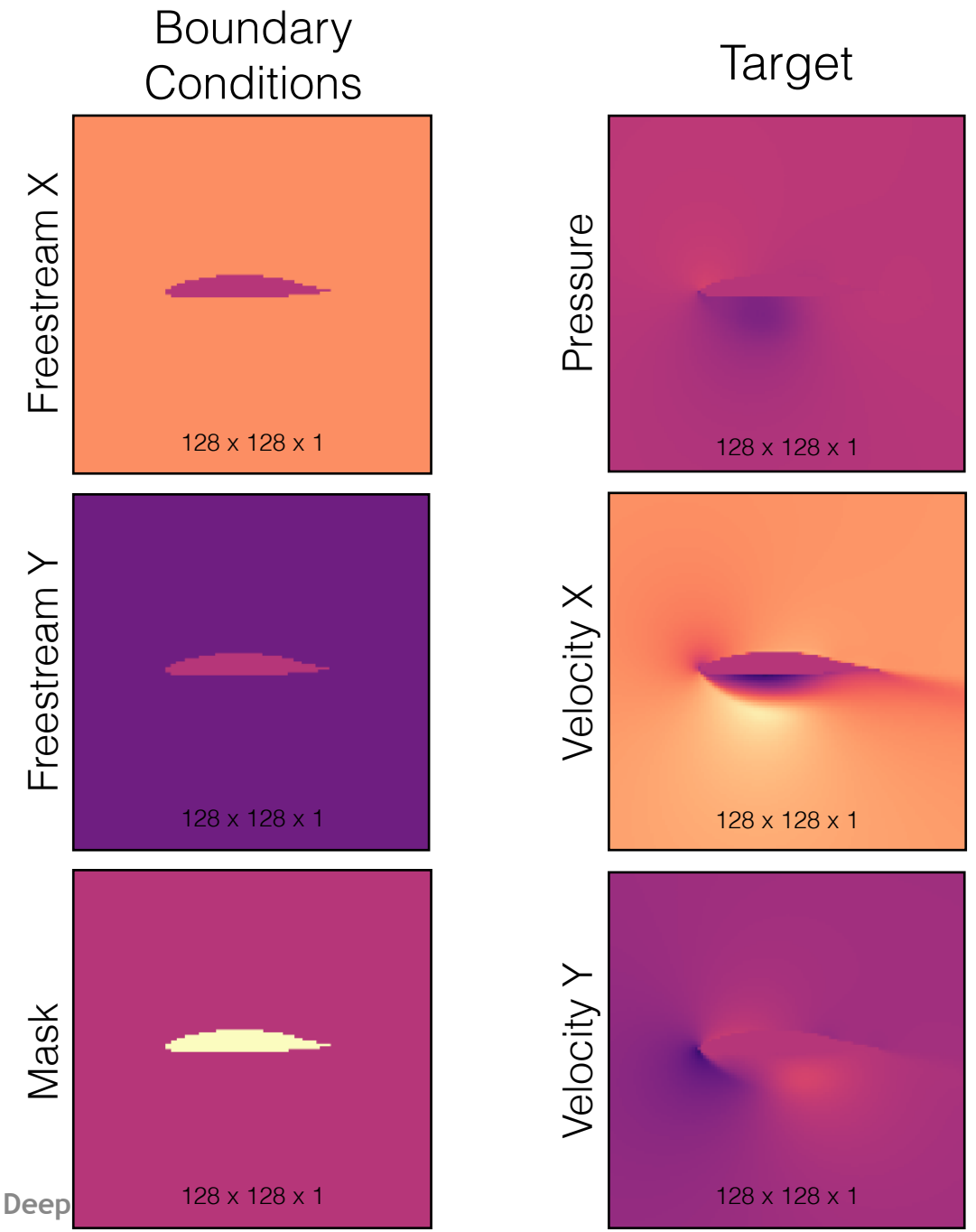
Solving PDEs with DL

- Data generation
- Large number of pairs: input (BCs) - targets (solutions)



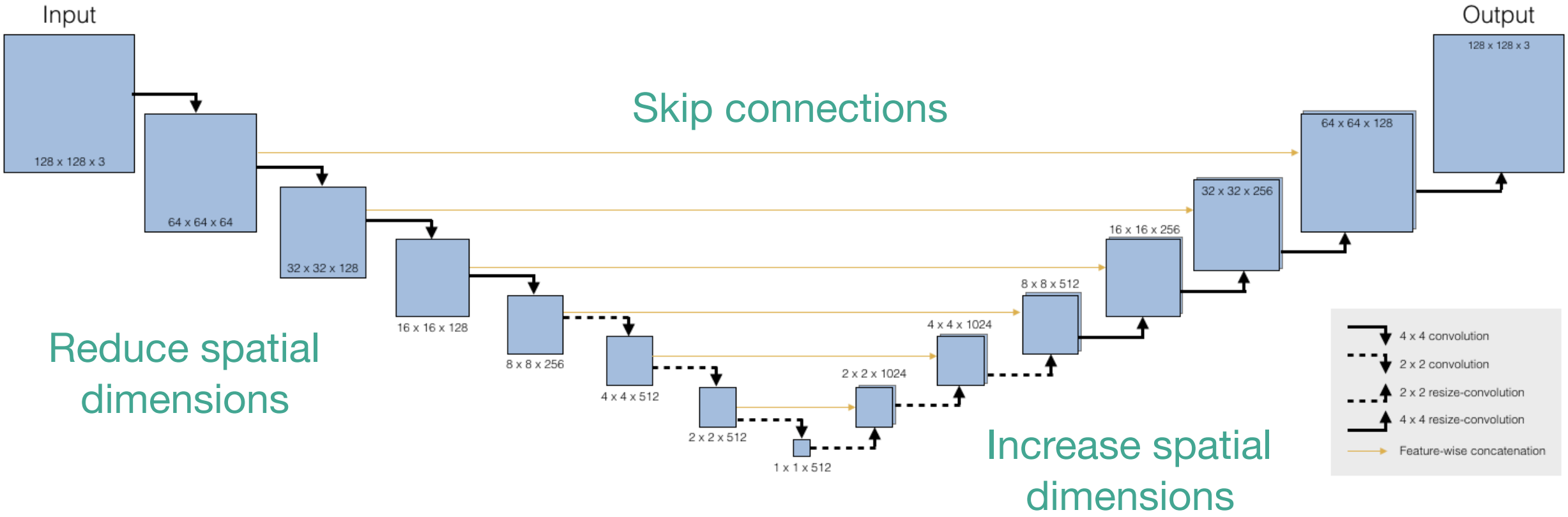
Solving PDEs with DL

- Data generation
- Example pair
- Note - boundary conditions (i.e. input fields) are typically constant
- Rasterized airfoil shape present in all three input fields



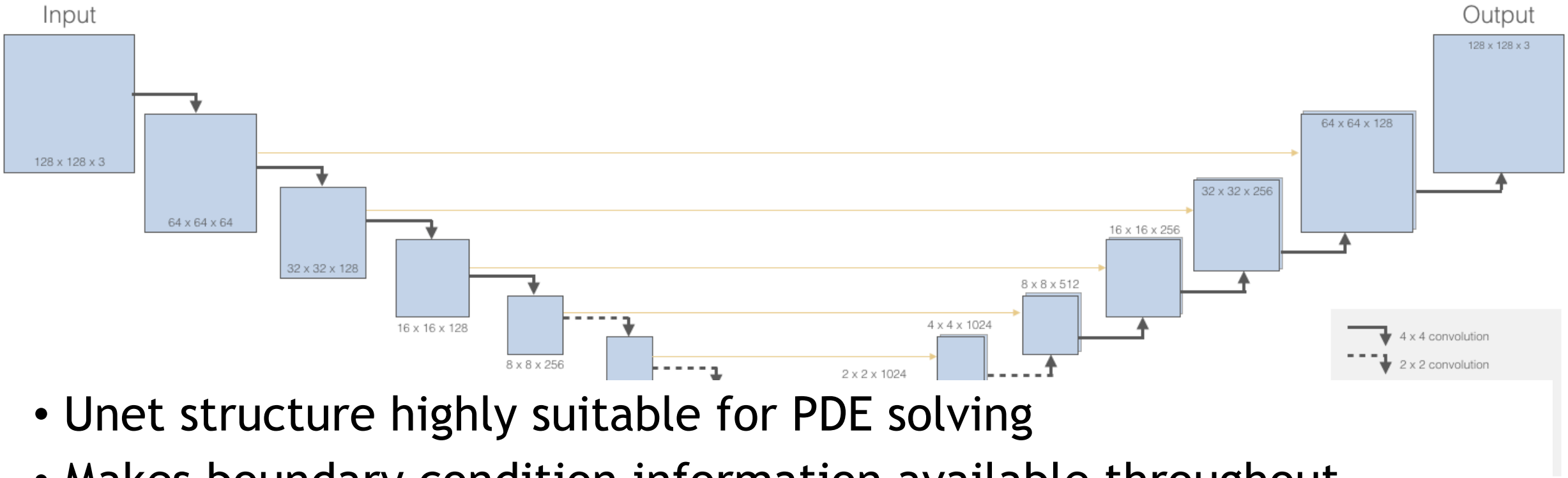
Solving PDEs with DL

- U-net NN architecture



Solving PDEs with DL

- U-net NN architecture



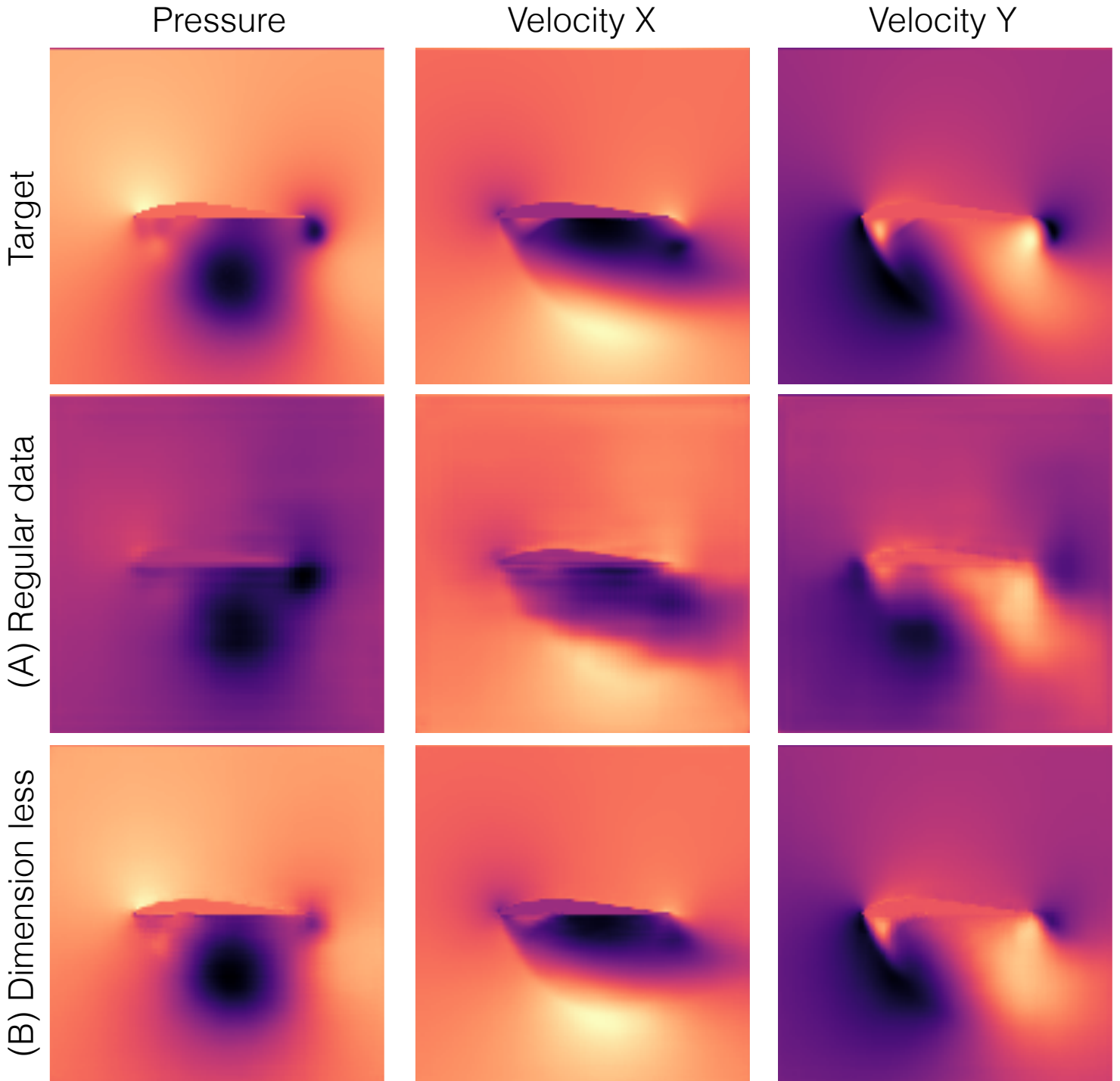
- Unet structure highly suitable for PDE solving
- Makes boundary condition information available throughout
- Crucial for inference of solution

Solving PDEs with DL

- Training: 80.000 iterations with ADAM optimizer
- Convolutions with enough data - no dropout necessary
- Learning rate decay stabilizes models

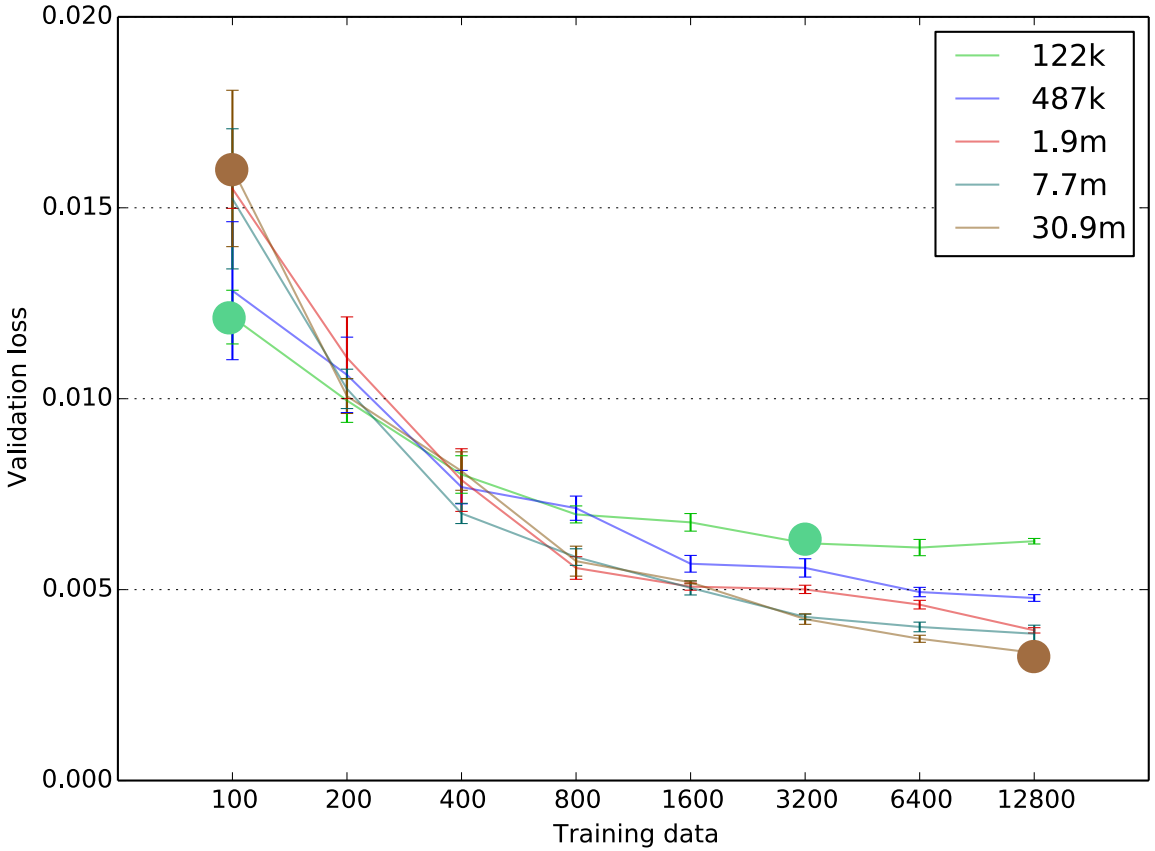
Results

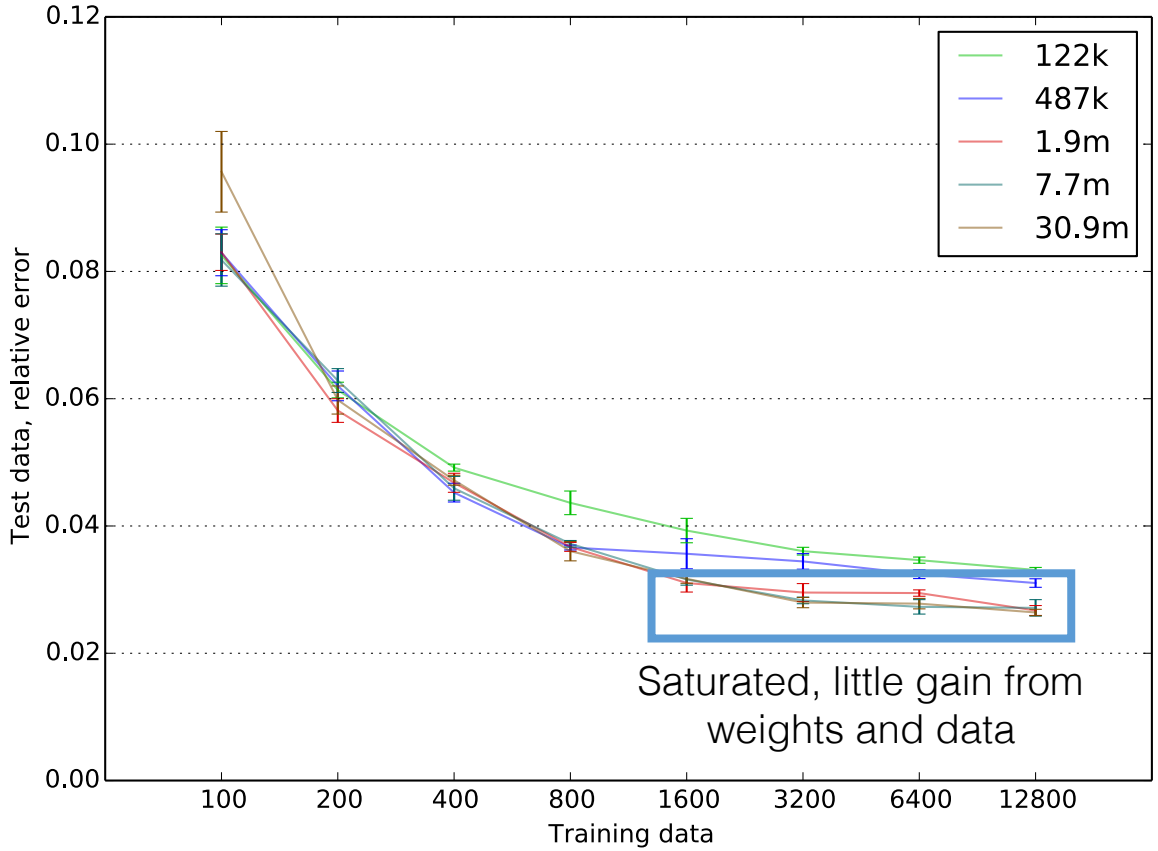
- Use knowledge about physics to simplify space of solutions: make quantities dimension- less
- Significant gains in inference accuracy



Solving PDEs with DL

- Validation and test accuracy for different model sizes





Code example

Solving PDEs with DL

Solving PDEs with DL

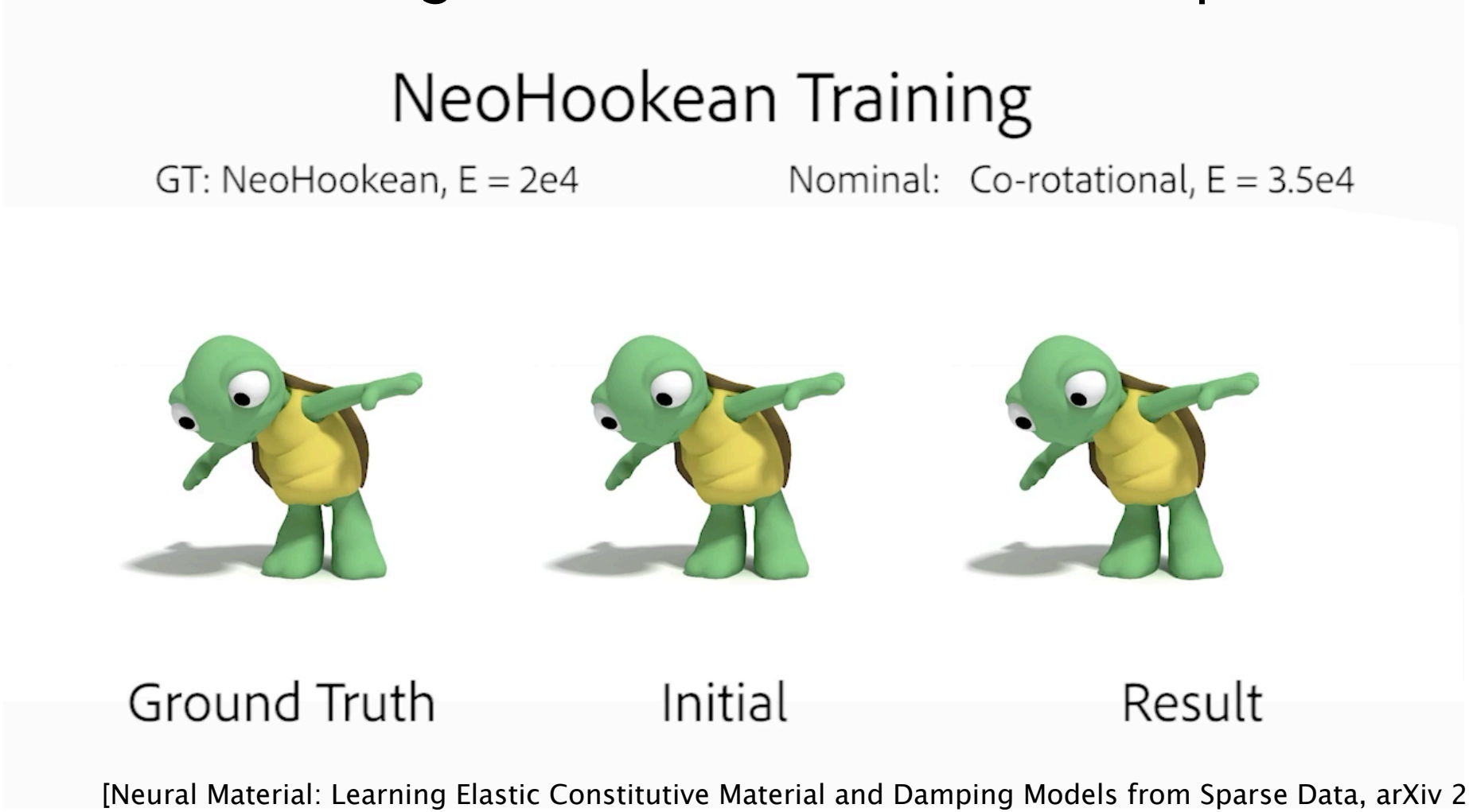
- Source code and training data available
- Requirements: numpy / pytorch , *OpenFOAM* for data generation
- Details at:
  https://github.com/thunil/Deep-Flow-Prediction and
  http://geometry.cs.ucl.ac.uk/creativeai/

Additional Examples

- Elasticity: material models
- Fluids: up-res algorithm & dimensionality reduction
- By no means exhaustive…

Neural Material - Elasticity

- Learn correction of regular FEM simulation for complex materials



[Neural Material: Learning Elastic Constitutive Material and Damping Models from Sparse Data, arXiv 2018]

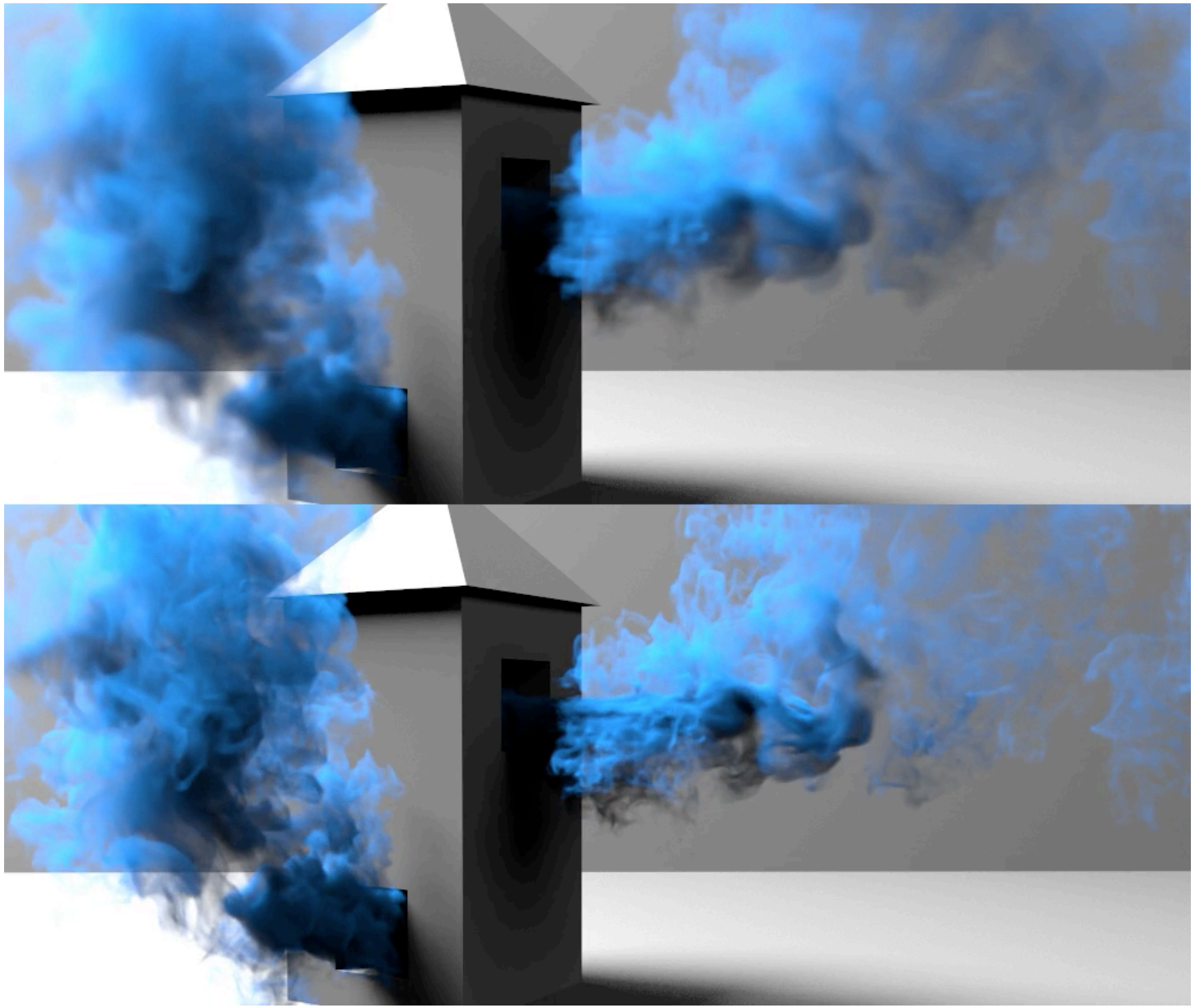
Neural Material - Elasticity

- Learn correction of regular FEM simulation for complex materials
- “Partial” approach
- Numerical simulation with flexible NN for material behavior

[Neural Material: Learning Elastic Constitutive Material and Damping Models from Sparse Data, arXiv 2018]

Temporal Data

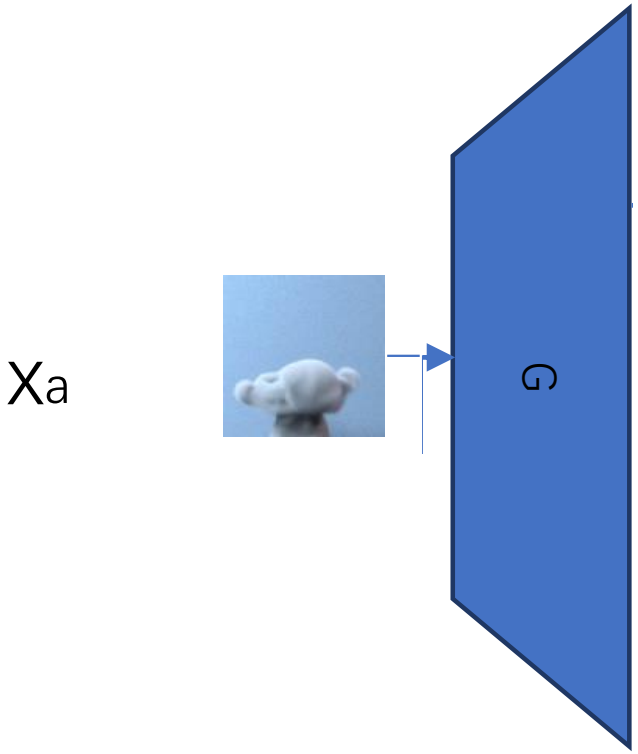
- tempoGAN: 3D GAN with temporal coherence



[tempoGAN: A Temporally Coherent, Volumetric GAN for Super-resolution Fluid Flow , SIGGRAPH 2018]

Temporal Data

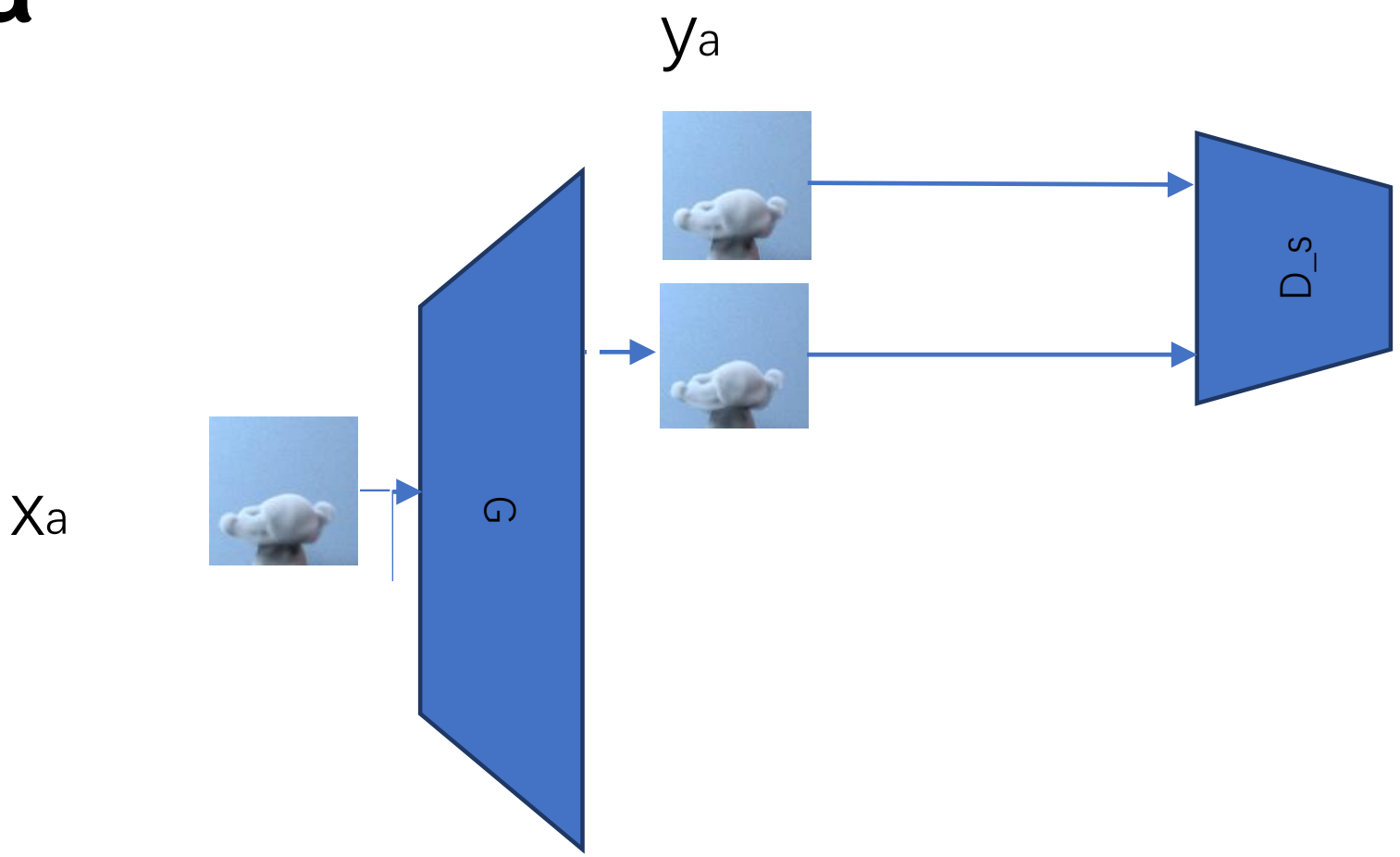
- tempoGAN: 3D GAN with temporal coherence



[tempoGAN: A Temporally Coherent, Volumetric GAN for Super-resolution Fluid Flow , SIGGRAPH 2018]

Temporal Data

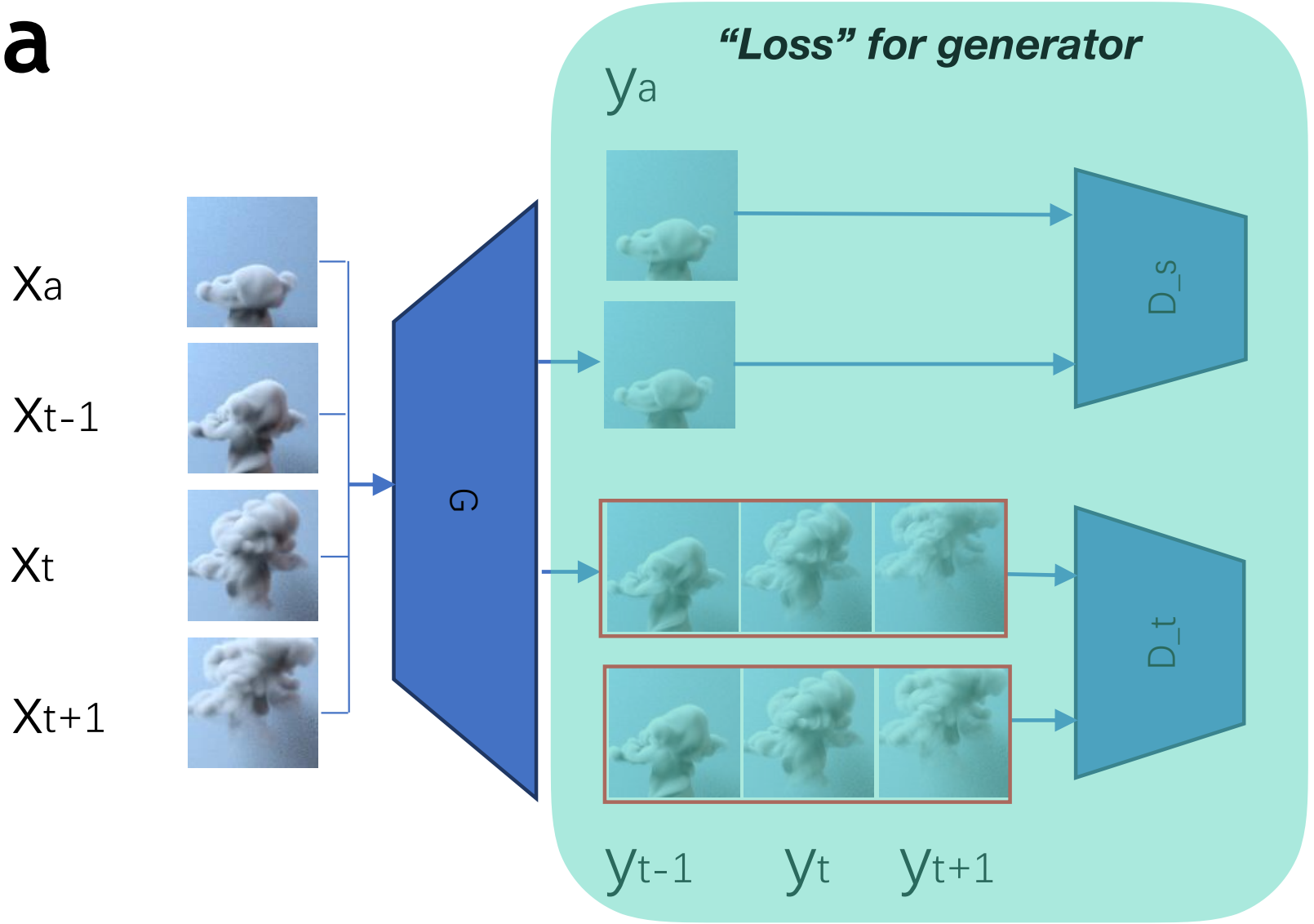
- tempoGAN: 3D GAN with temporal coherence



[tempoGAN: A Temporally Coherent, Volumetric GAN for Super-resolution Fluid Flow , SIGGRAPH 2018]

Temporal Data

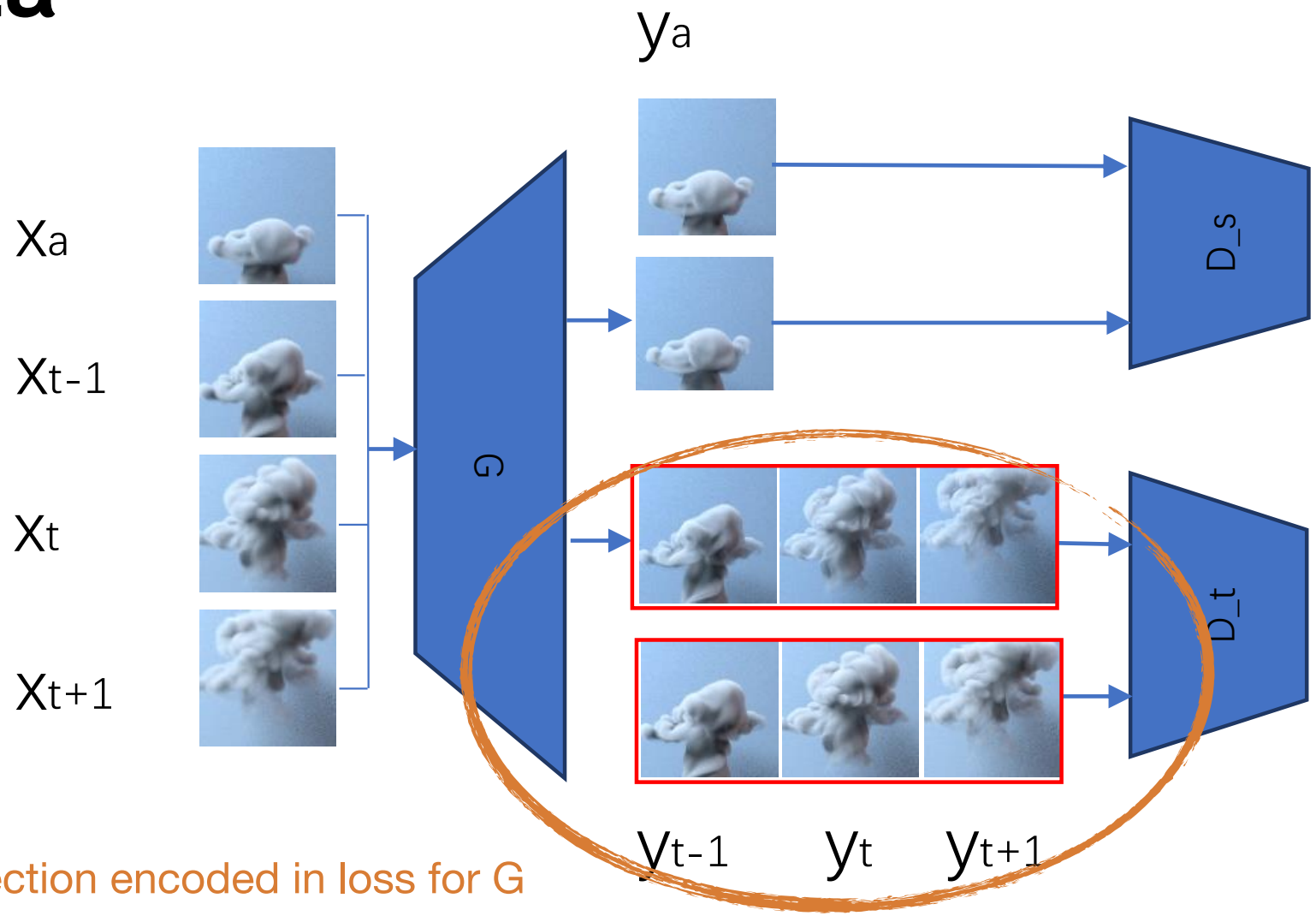
- tempoGAN: 3D GAN with temporal coherence



[tempoGAN: A Temporally Coherent, Volumetric GAN for Super-resolution Fluid Flow , SIGGRAPH 2018]

Temporal Data

- tempoGAN: 3D GAN with temporal coherence



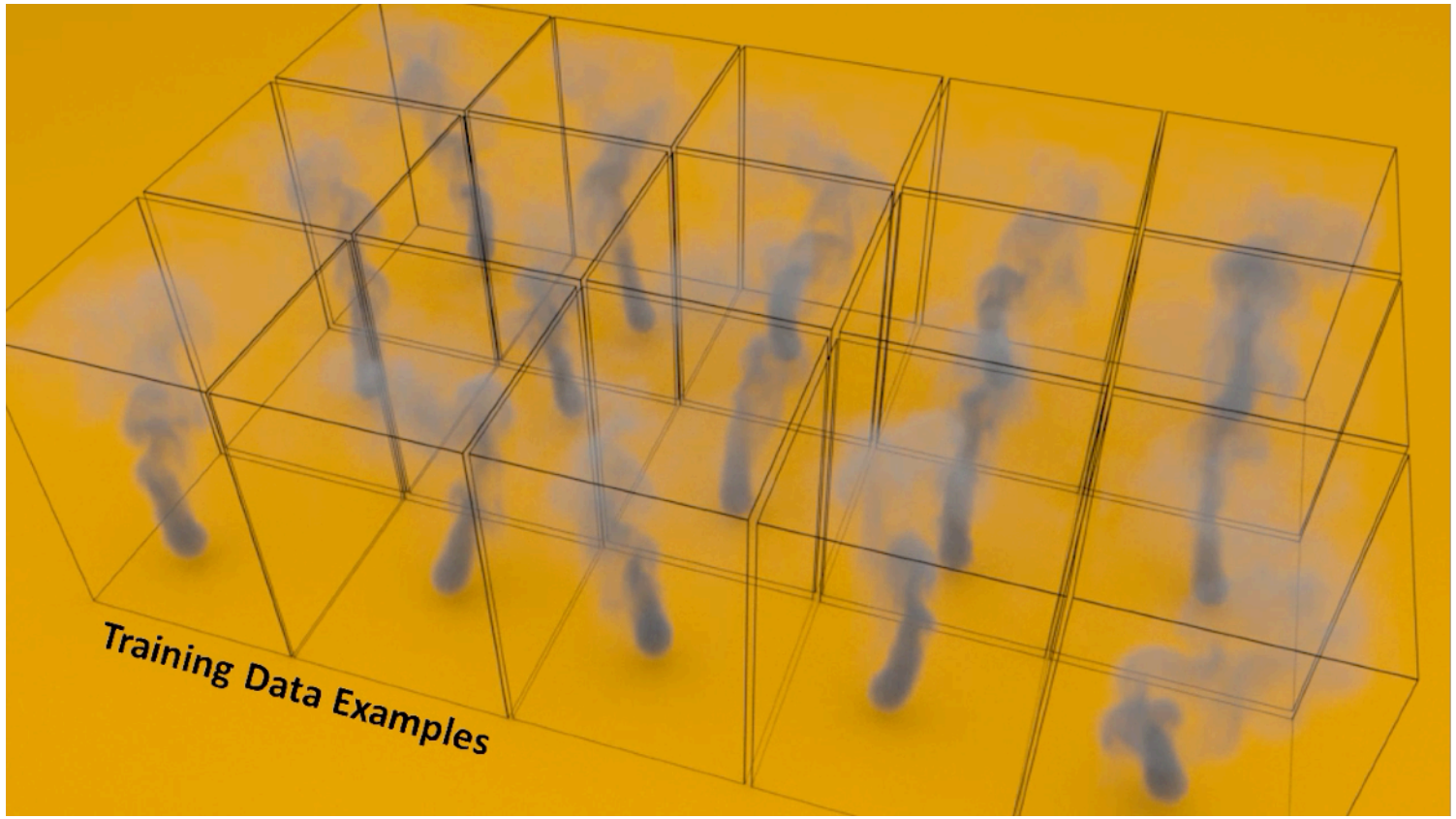
[tempoGAN: A Temporally Coherent, Volumetric GAN for Super-resolution Fluid Flow , SIGGRAPH 2018]

Temporal Data

[tempoGAN: A Temporally Coherent, Volumetric GAN for Super-resolution Fluid Flow , SIGGRAPH 2018]

Latent Spaces

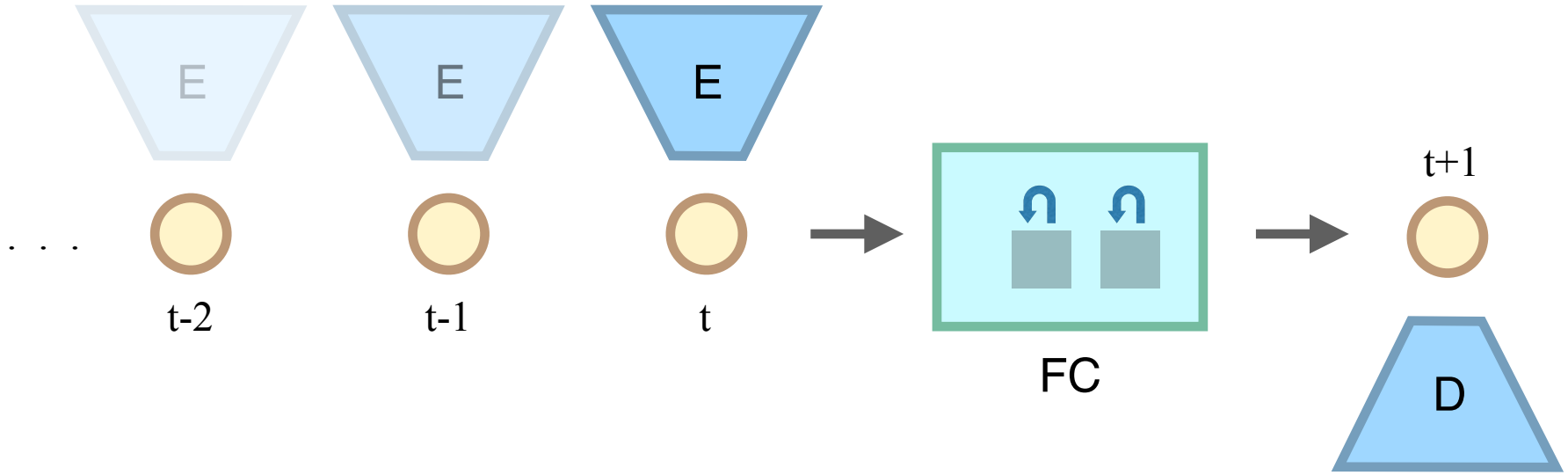
- Learn flexible reduced representation for physics problems



[Deep Fluids: A Generative Network for Parameterized Fluid Simulations, arXiv 2018]
[Latent-space Physics: Towards Learning the Temporal Evolution of Fluid Flow, arXiv 2018]

Latent Spaces

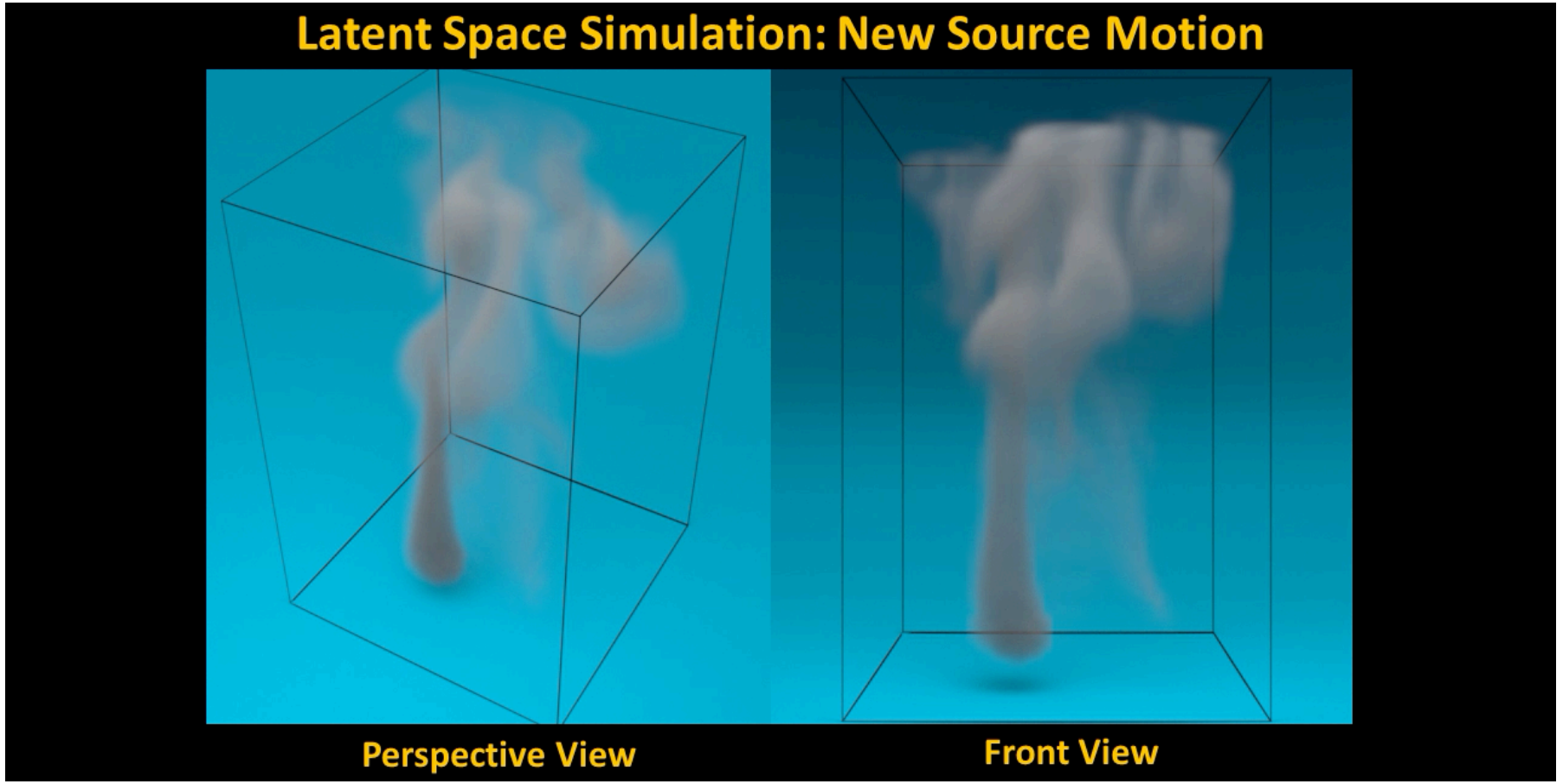
- Learn flexible reduced representation for physics problems
  - Employ Encoder part (E) of Autoencoder network to reduce dimensions
  - Predict future state in latent space with FC network
  - Use Decoder (D) of Autoencoder to retrieve volume data



[Deep Fluids: A Generative Network for Parameterized Fluid Simulations, arXiv 2018]
[Latent-space Physics: Towards Learning the Temporal Evolution of Fluid Flow, arXiv 2018]

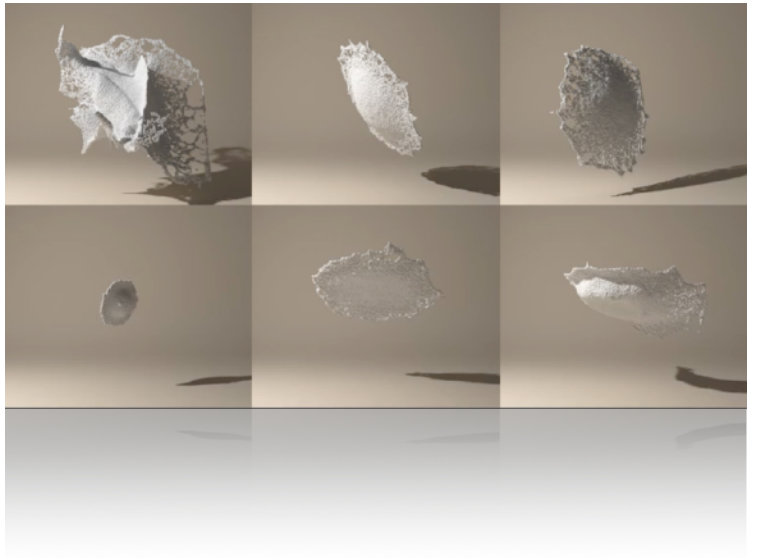
Latent Spaces

- Learn flexible reduced representation for physics problems



[Deep Fluids: A Generative Network for Parameterized Fluid Simulations, arXiv 2018]
[Latent-space Physics: Towards Learning the Temporal Evolution of Fluid Flow, arXiv 2018]

Summary



- Checklist for solving PDEs with DL:
  - ✓ Model? (Typically given)
  - ✓ Data? Can enough training data be generated?
  - ✓ Which NN Architecture?
  - ✓ Fine tuning: learning rate, number of layers & features?
  - ✓ Hyper-parameters, activation functions etc.?

Summary

- Approach PDE solving with DL like solving with traditional numerical methods:
  - Find closest example in literature
  - Reproduce & test
  - Then vary, adjust, refine …

Thank you!

http://geometry.cs.ucl.ac.uk/creativeai/