

# Introduction to Deep Learning

January 28, 2019



Follow me on [LinkedIn](#) for more:  
Steve Nouri  
<https://www.linkedin.com/in/stevenouri/>



## 'Deep Voice' Software Can Clone Anyone's Voice With Just 3.7 Seconds of Audio

Using snippets of voices, Baidu's 'Deep Voice' can generate new speech, accents, and tones.



### 'Creative' AlphaZero leads way for chess computers and, maybe, science

Former chess world champion Garry Kasparov likes what he sees of computer that could be used to find cures for diseases



### Stock Predictions Based On AI: Is the Market Truly Predictable?



### Google's DeepMind aces protein folding

By Robert F. Service | Dec. 6, 2018, 12:05 PM

## t with DEEPMIND E STARCRAFT TRIUMPH FO

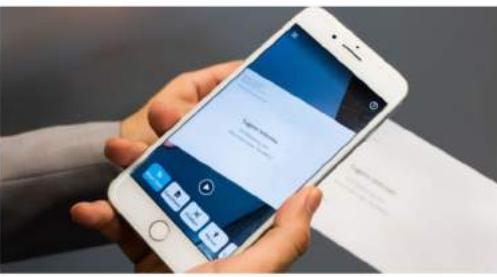
### How an A.I. 'Cat-and-Mouse Game' Generates Believable Fake Photos

By CADE METZ and KEITH COLLINS | JAN 2, 2018



To create the final image in this set, the system generated 10 million revisions over 18 days.

### Let There Be Sight: How Deep Learning Is Helping the Blind 'See'



### Technology outpacing security measures

| Facial Recognition | Features and Interviews



### Neural networks everywhere

New chip reduces neural networks' power consumption by up to 95 percent, making them practical for battery-powered devices.



### Researchers introduce a deep learning method that converts mono audio recordings into 3D sounds using video scenes

By Natasja Mekhora | December 26, 2018, 11:21 AM

### AI beats docs in cancer spotting

A new study provides a fresh example of machine learning as an important diagnostic tool. Paul Biegler reports.



### The faces show how far AI image generation has advanced in just four years

Those on the right aren't real; they're the product of machine learning



### Automation And Algorithms: De-Risking Manufacturing With Artificial Intelligence

Sarah Goehrke Contributor @  
Manufacturing  
I focus on the industrialization of additive manufacturing.

TWEET THIS

The two key applications of AI in manufacturing are pricing and manufacturability feedback

### AI Can Help In Predicting Cryptocurrency Value

By Erik Berndsen | Last updated Jan 21, 2019



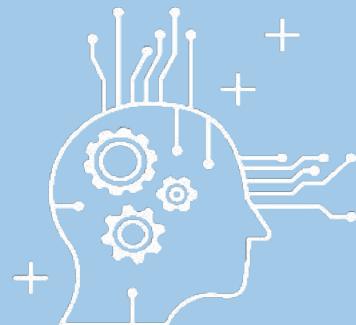
parent company Alphabet, is  
ascent self-driving technology

Complex of bacteria-infecting viral proteins modeled in CASP 13. The complex contains 10 proteins that were modeled individually. PROTEIN DATA BANK

# What is Deep Learning?

## ARTIFICIAL INTELLIGENCE

Any technique that enables computers to mimic human behavior



## MACHINE LEARNING

Ability to learn without explicitly being programmed



## DEEP LEARNING

Extract patterns from data using neural networks



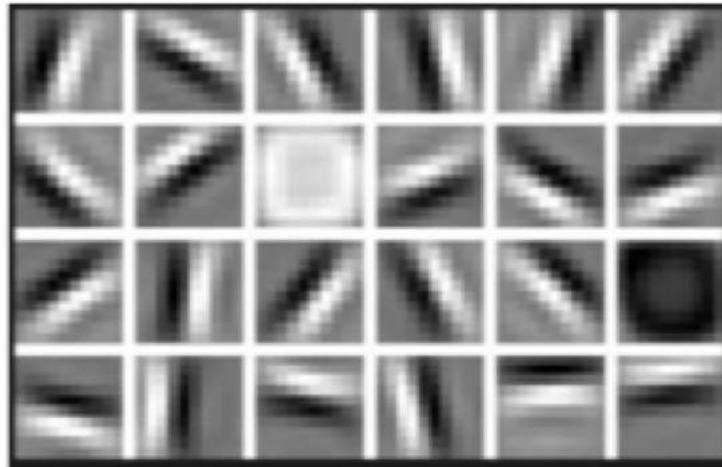
# Why Deep Learning and Why Now?

# Why Deep Learning?

Hand engineered features are time consuming, brittle and not scalable in practice

Can we learn the **underlying features** directly from data?

**Low Level Features**



Lines & Edges

**Mid Level Features**



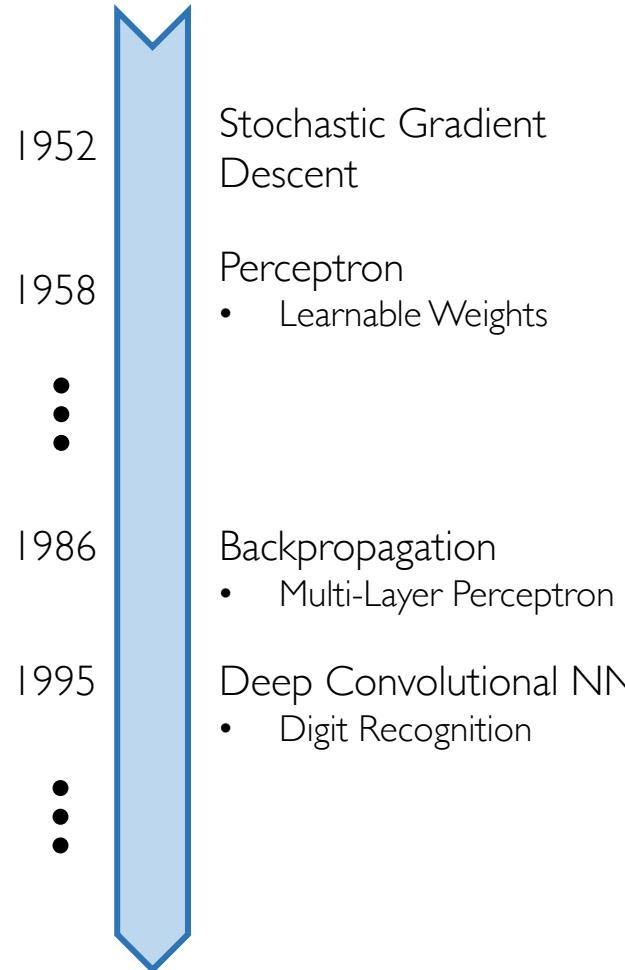
Eyes & Nose & Ears

**High Level Features**



Facial Structure

# Why Now?



Neural Networks date back decades, so why the resurgence?

## I. Big Data

- Larger Datasets
- Easier Collection & Storage



## 2. Hardware

- Graphics Processing Units (GPUs)
- Massively Parallelizable



## 3. Software

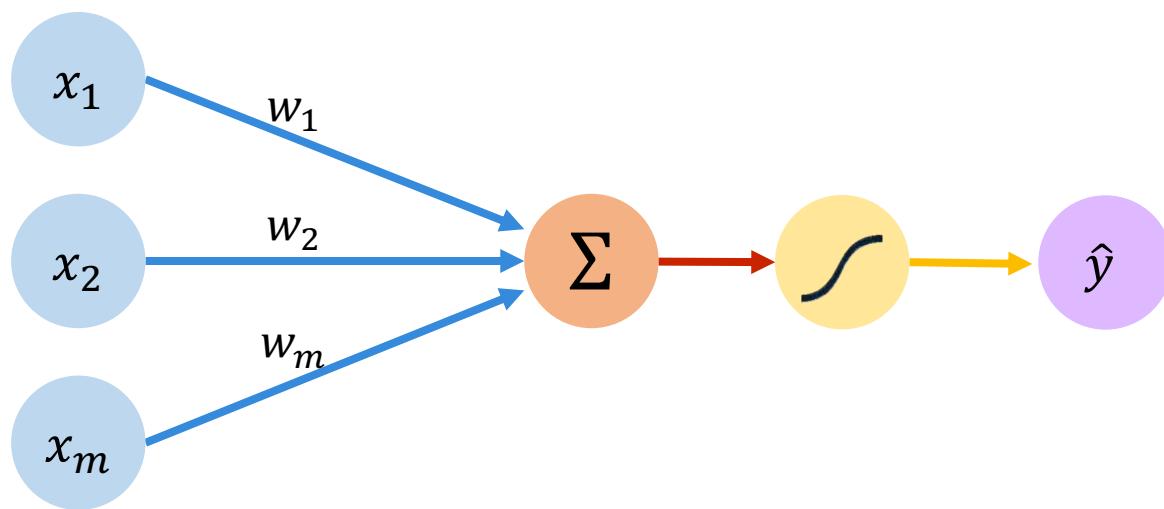
- Improved Techniques
- New Models
- Toolboxes



# The Perceptron

## The structural building block of deep learning

# The Perceptron: Forward Propagation



Inputs    Weights    Sum    Non-Linearity    Output

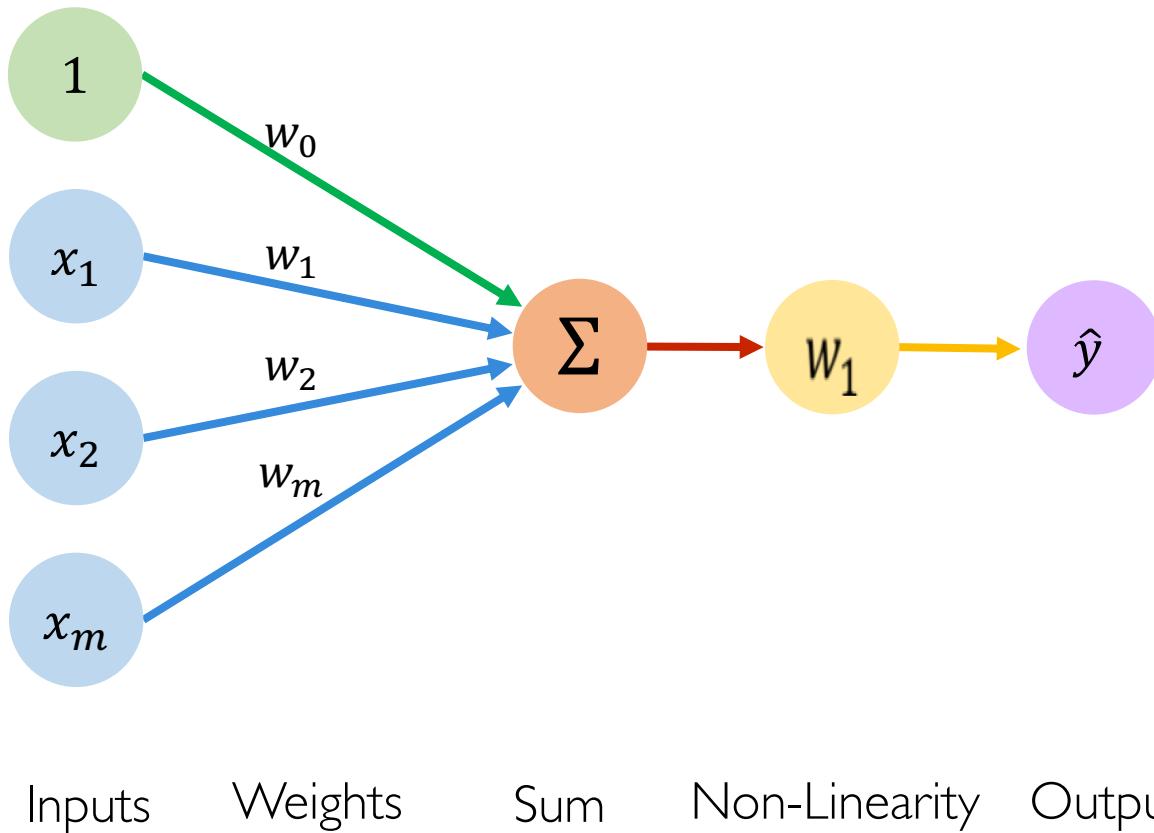
Linear combination  
of inputs

Output

$$\hat{y} = g \left( \sum_{i=1}^m x_i w_i \right)$$

Non-linear  
activation function

# The Perceptron: Forward Propagation



Linear combination of inputs

Output

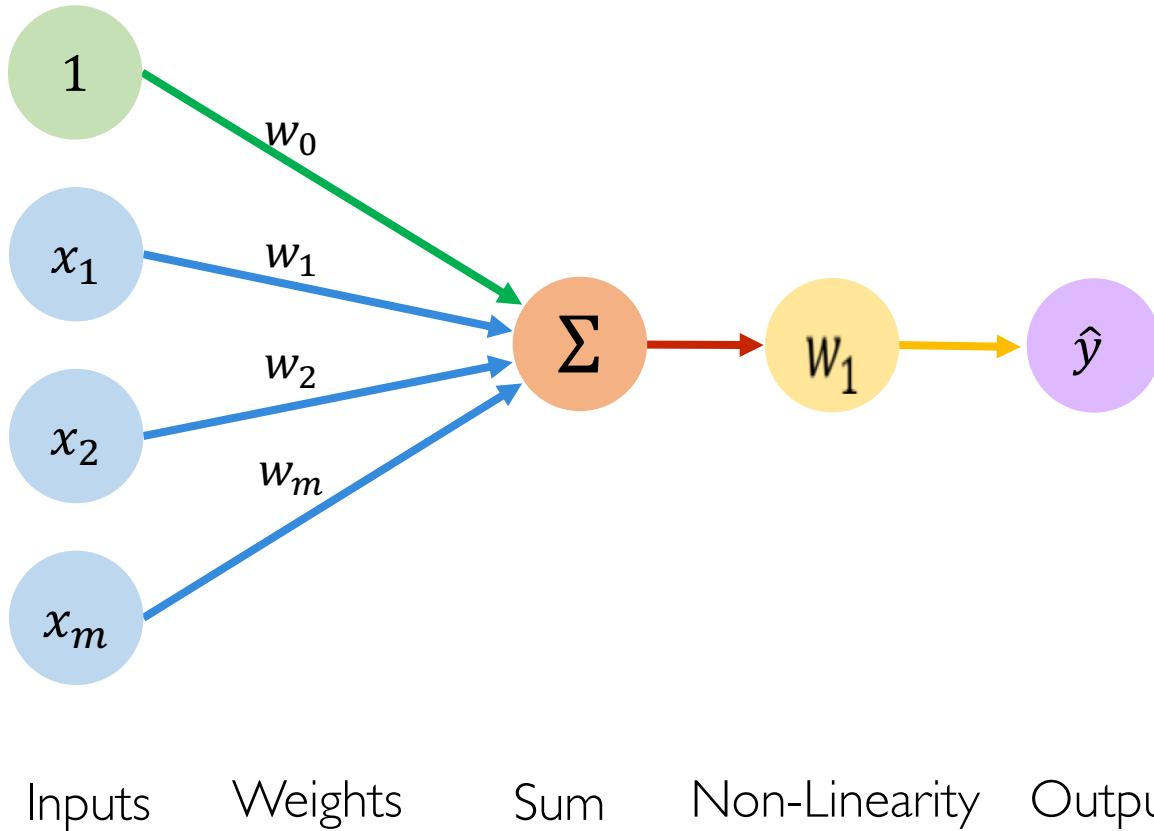
$\hat{y} = g \left( w_0 + \sum_{i=1}^m x_i w_i \right)$

Non-linear activation function

Bias

Diagram illustrating the mathematical formula for the perceptron's output. The output  $\hat{y}$  is the result of applying a non-linear activation function  $g$  to the linear combination of inputs and weights. The linear combination is calculated as  $w_0 + \sum_{i=1}^m x_i w_i$ . The term  $w_0$  is labeled as the bias, and the term  $\sum_{i=1}^m x_i w_i$  is labeled as the linear combination of inputs.

# The Perceptron: Forward Propagation

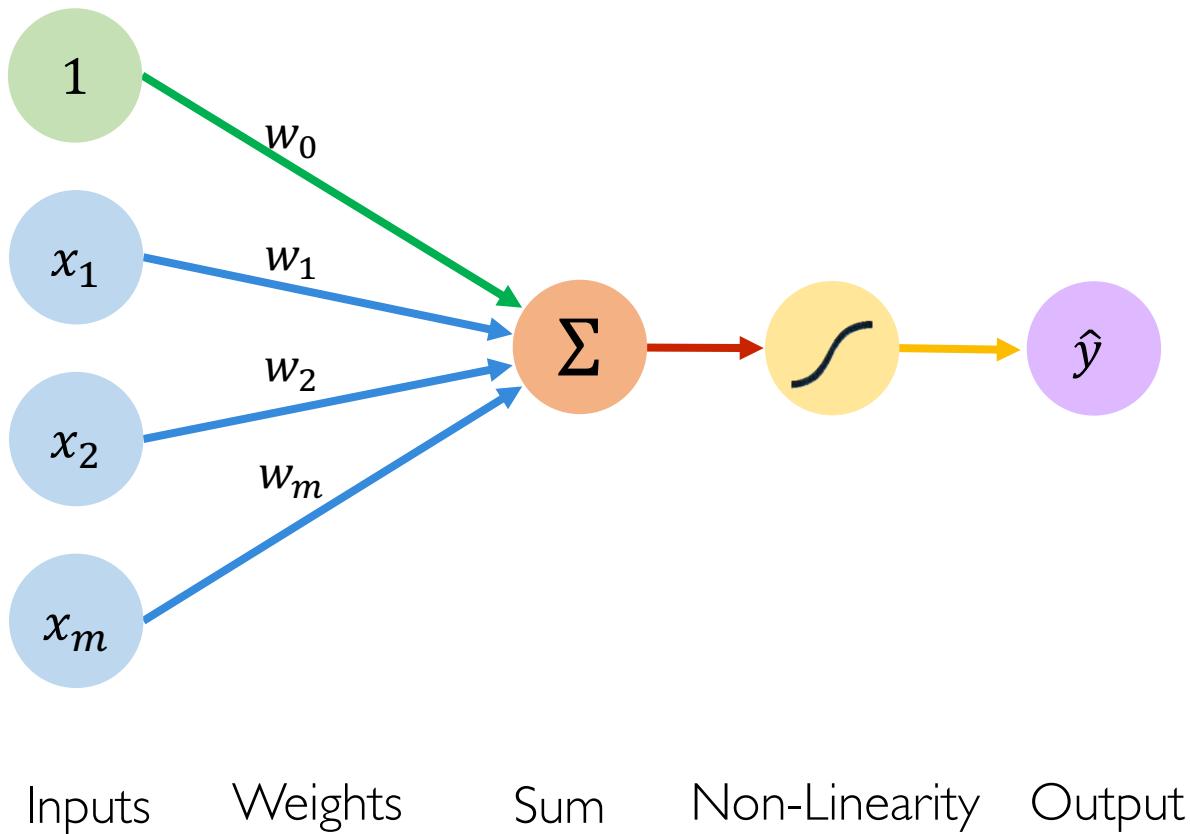


$$\hat{y} = g \left( w_0 + \sum_{i=1}^m x_i w_i \right)$$

$$\hat{y} = g ( w_0 + \mathbf{X}^T \mathbf{W} )$$

where:  $\mathbf{X} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$  and  $\mathbf{W} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}$

# The Perceptron: Forward Propagation

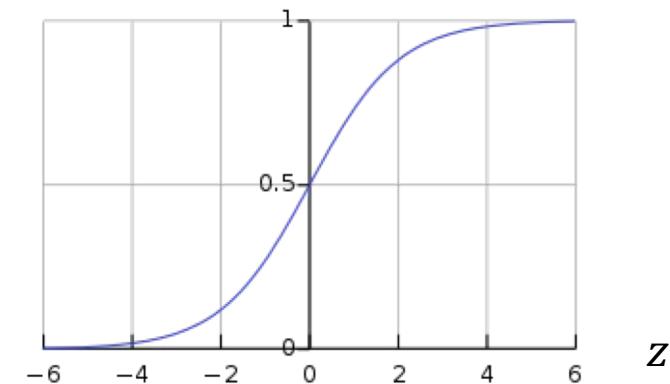


## Activation Functions

$$\hat{y} = g(w_0 + X^T W)$$

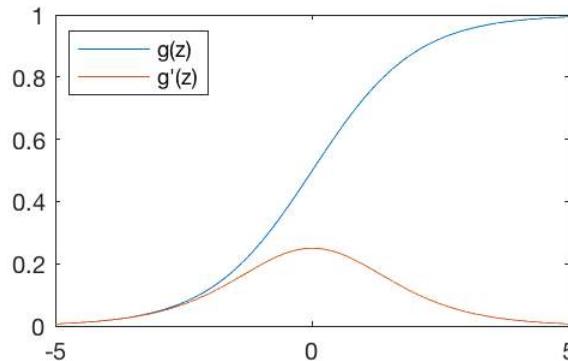
- Example: sigmoid function

$$g(z) = \sigma(z) = \frac{1}{1 + e^{-z}}$$



# Common Activation Functions

Sigmoid Function

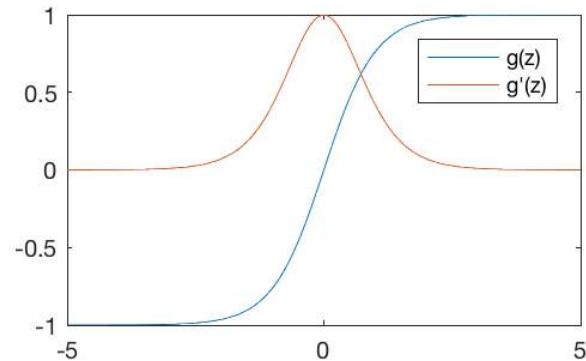


$$g(z) = \frac{1}{1 + e^{-z}}$$

$$g'(z) = g(z)(1 - g(z))$$

 `tf.nn.sigmoid(z)`

Hyperbolic Tangent

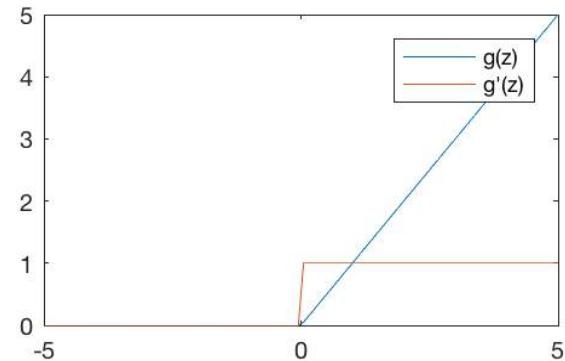


$$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

$$g'(z) = 1 - g(z)^2$$

 `tf.nn.tanh(z)`

Rectified Linear Unit (ReLU)



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

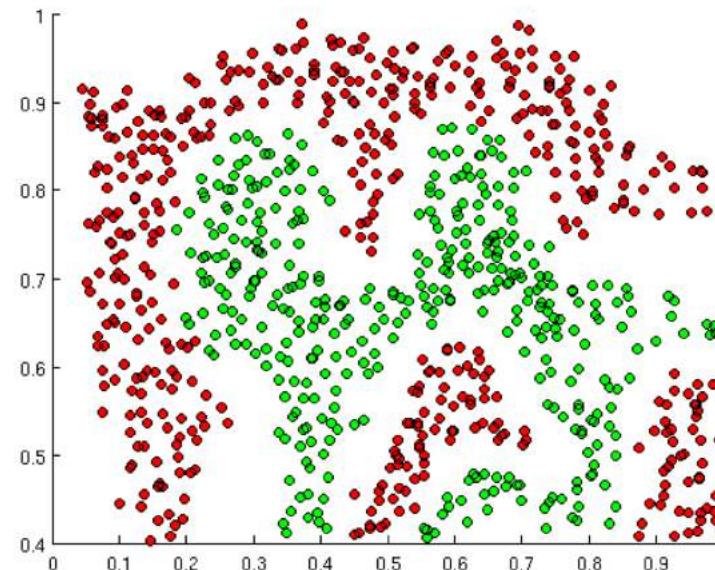
$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

 `tf.nn.relu(z)`

NOTE: All activation functions are non-linear

# Importance of Activation Functions

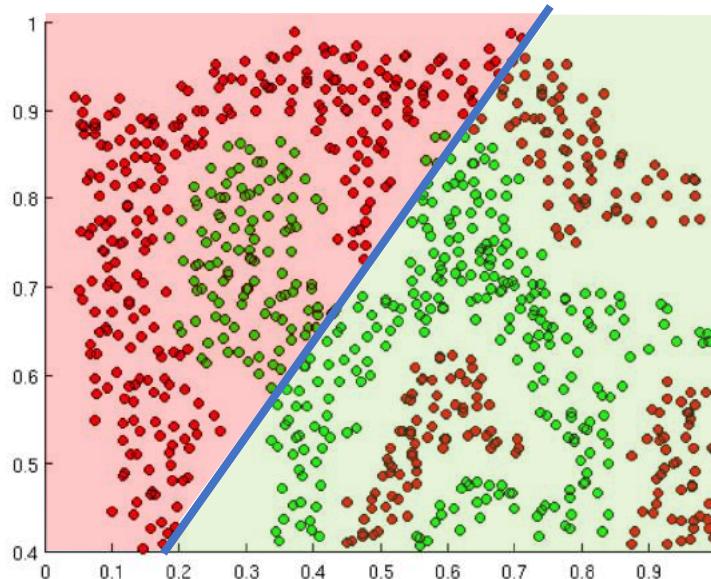
The purpose of activation functions is to **introduce non-linearities** into the network



What if we wanted to build a Neural Network to  
distinguish green vs red points?

# Importance of Activation Functions

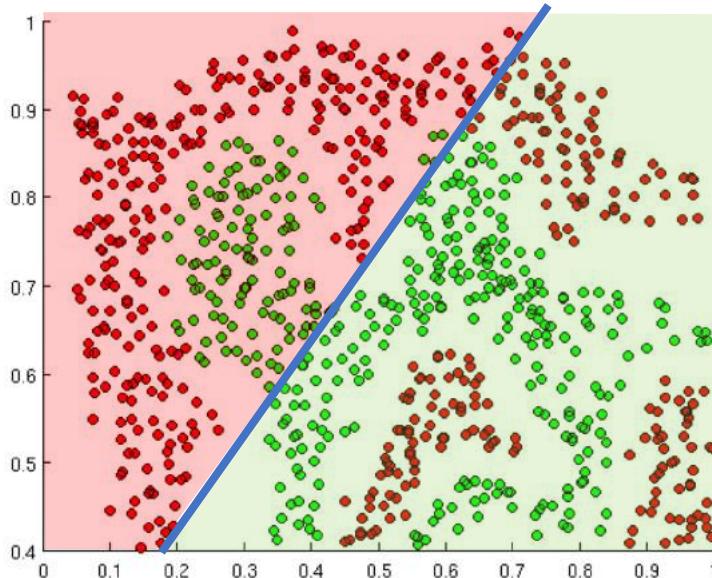
The purpose of activation functions is to **introduce non-linearities** into the network



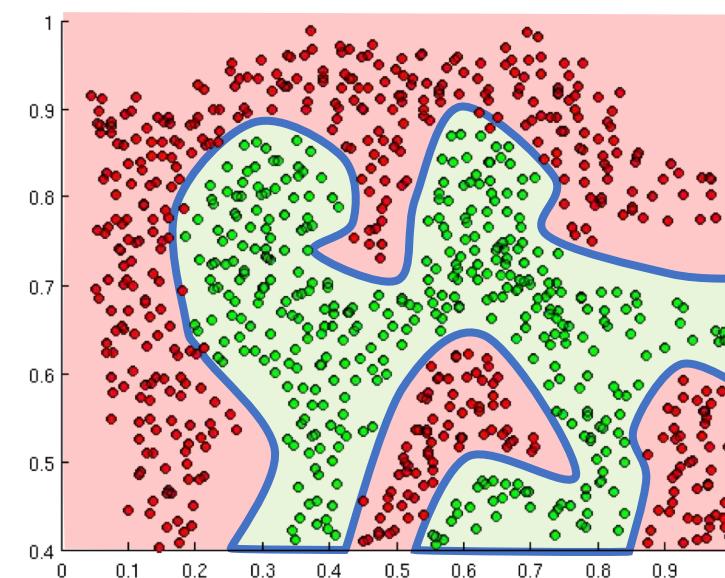
Linear Activation functions produce linear decisions no matter the network size

# Importance of Activation Functions

The purpose of activation functions is to **introduce non-linearities** into the network

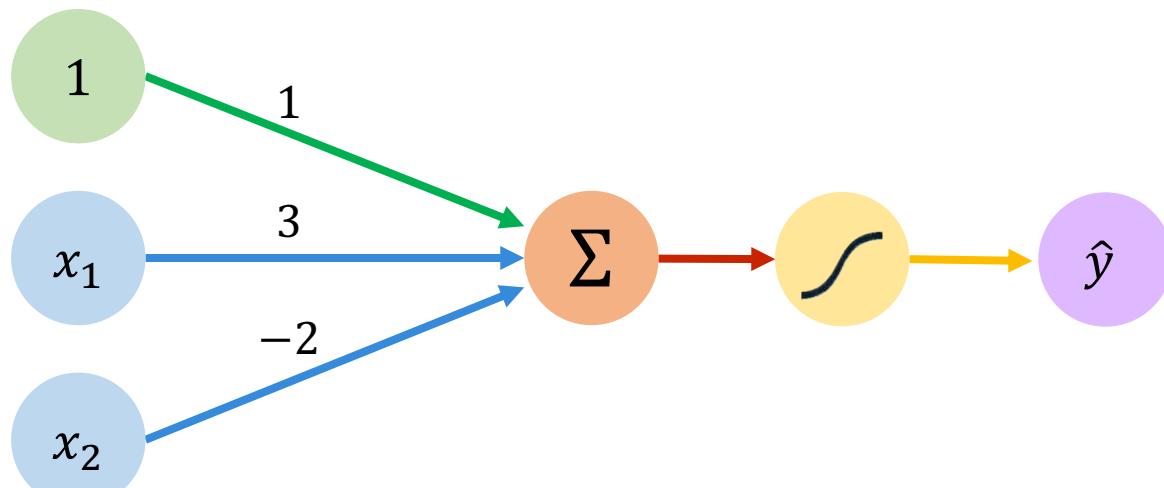


Linear Activation functions produce linear decisions no matter the network size



Non-linearities allow us to approximate arbitrarily complex functions

# The Perceptron: Example

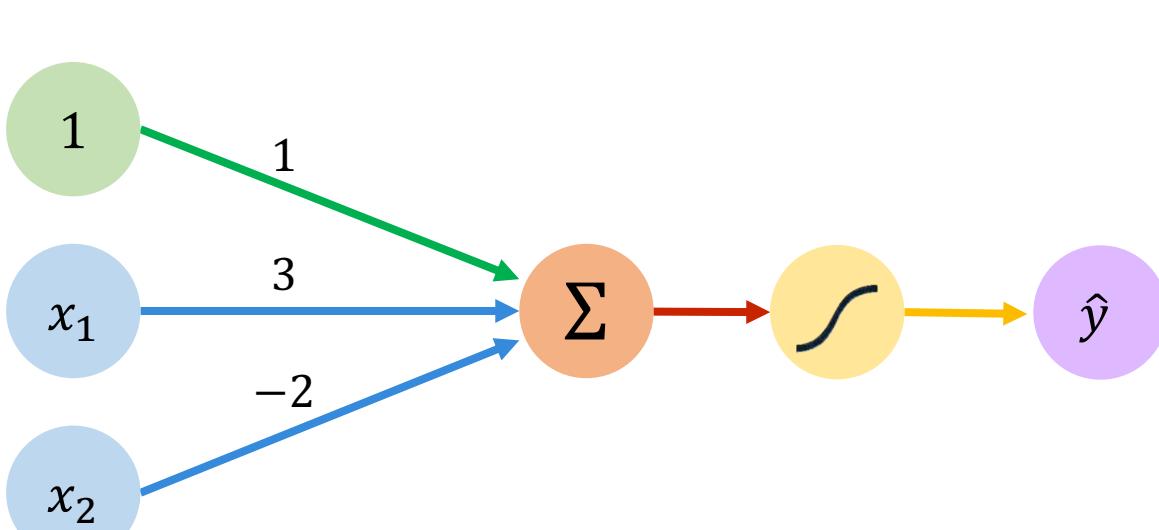


We have:  $w_0 = 1$  and  $\mathbf{W} = \begin{bmatrix} 3 \\ -2 \end{bmatrix}$

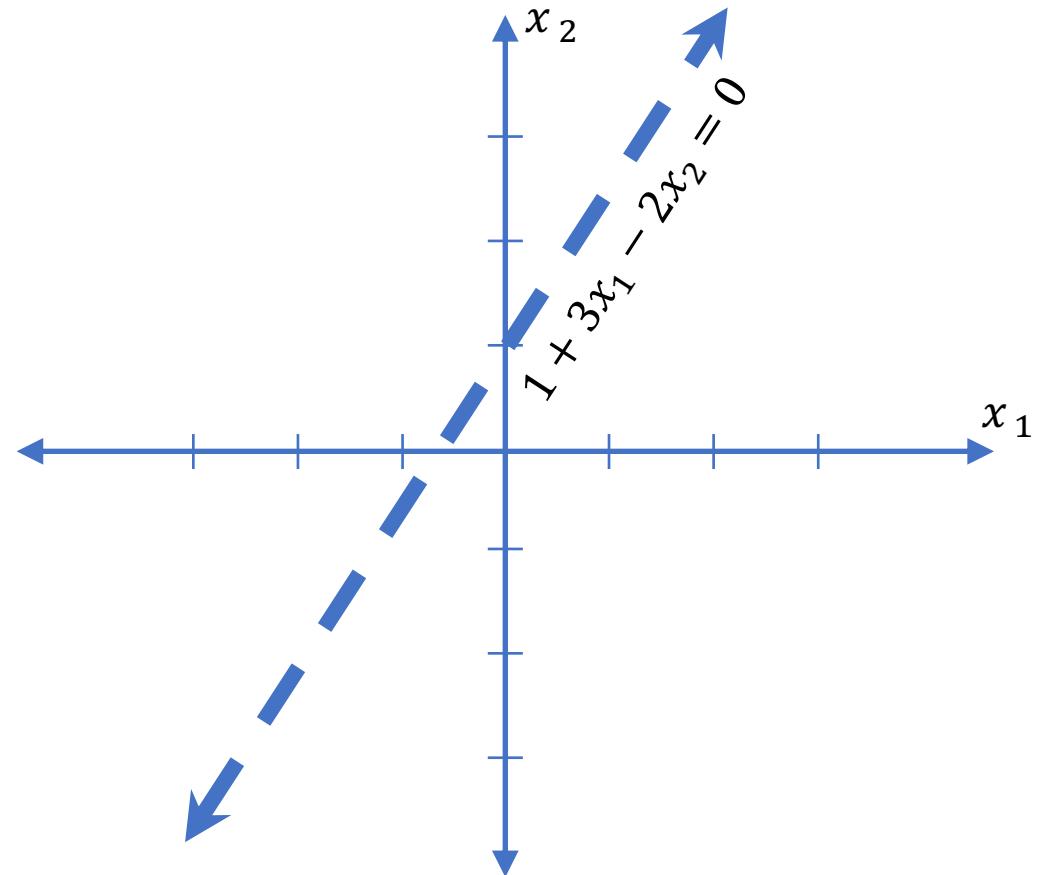
$$\begin{aligned}\hat{y} &= g(w_0 + \mathbf{X}^T \mathbf{W}) \\ &= g\left(1 + \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T \begin{bmatrix} 3 \\ -2 \end{bmatrix}\right) \\ \hat{y} &= g(1 + 3x_1 - 2x_2)\end{aligned}$$

This is just a line in 2D!

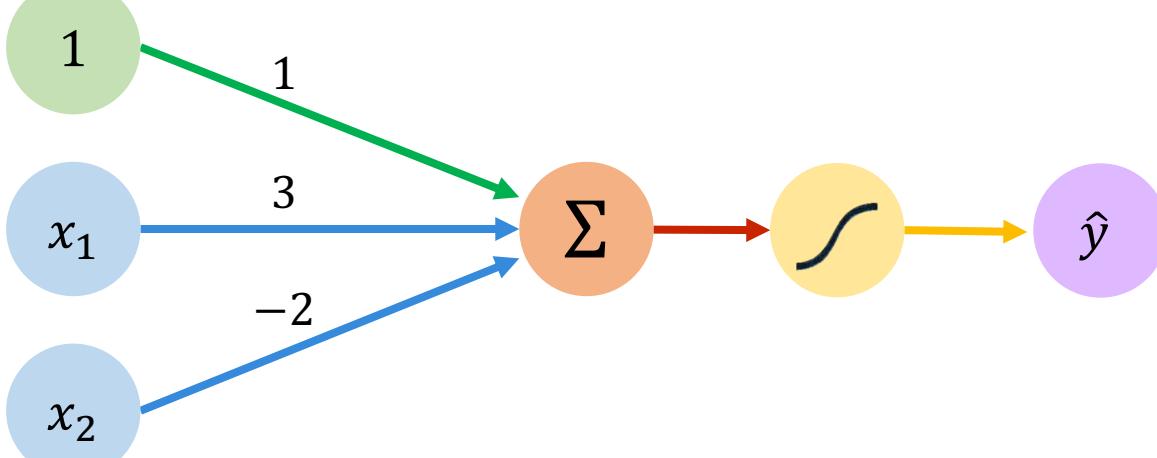
# The Perceptron: Example



$$\hat{y} = g(1 + 3x_1 - 2x_2)$$



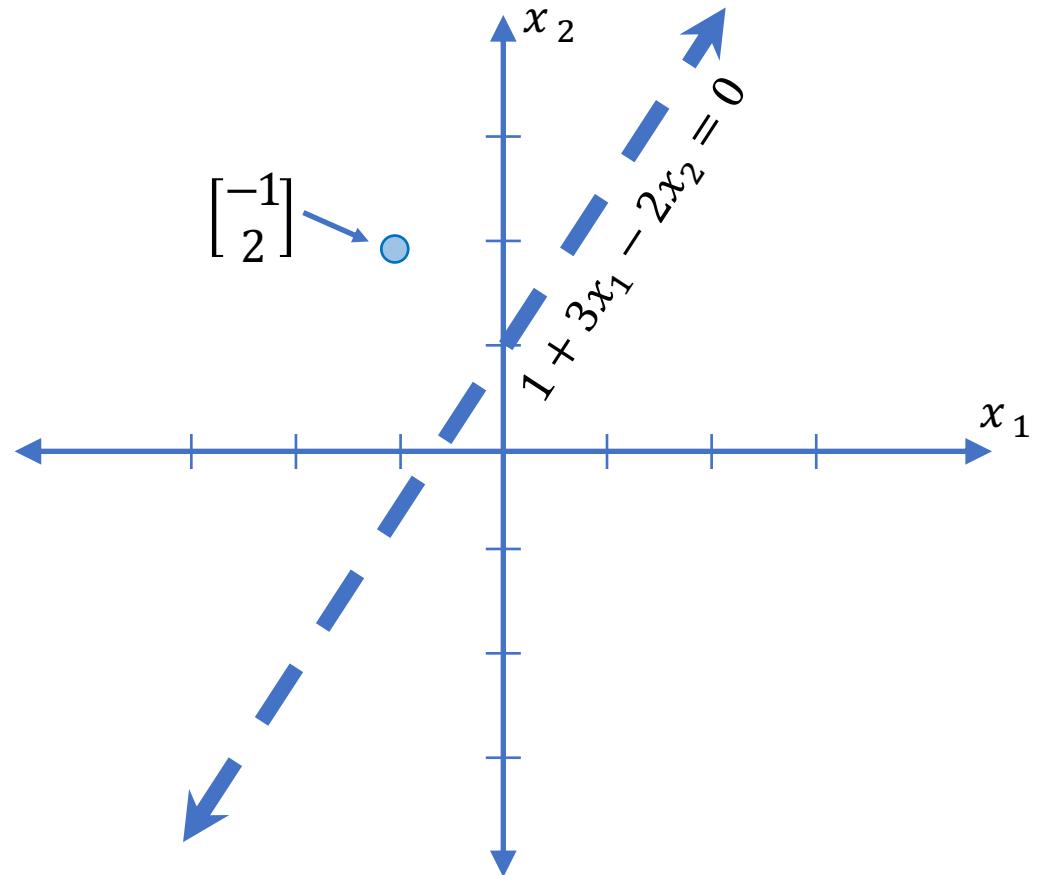
# The Perceptron: Example



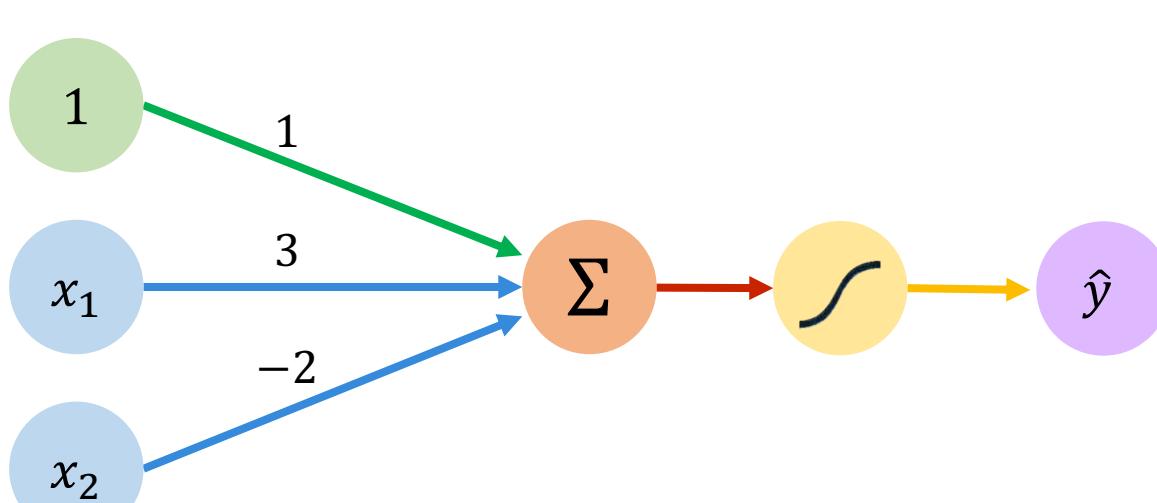
Assume we have input:  $\mathbf{x} = \begin{bmatrix} -1 \\ 2 \end{bmatrix}$

$$\begin{aligned}\hat{y} &= g(1 + (3 * -1) - (2 * 2)) \\ &= g(-6) \approx 0.002\end{aligned}$$

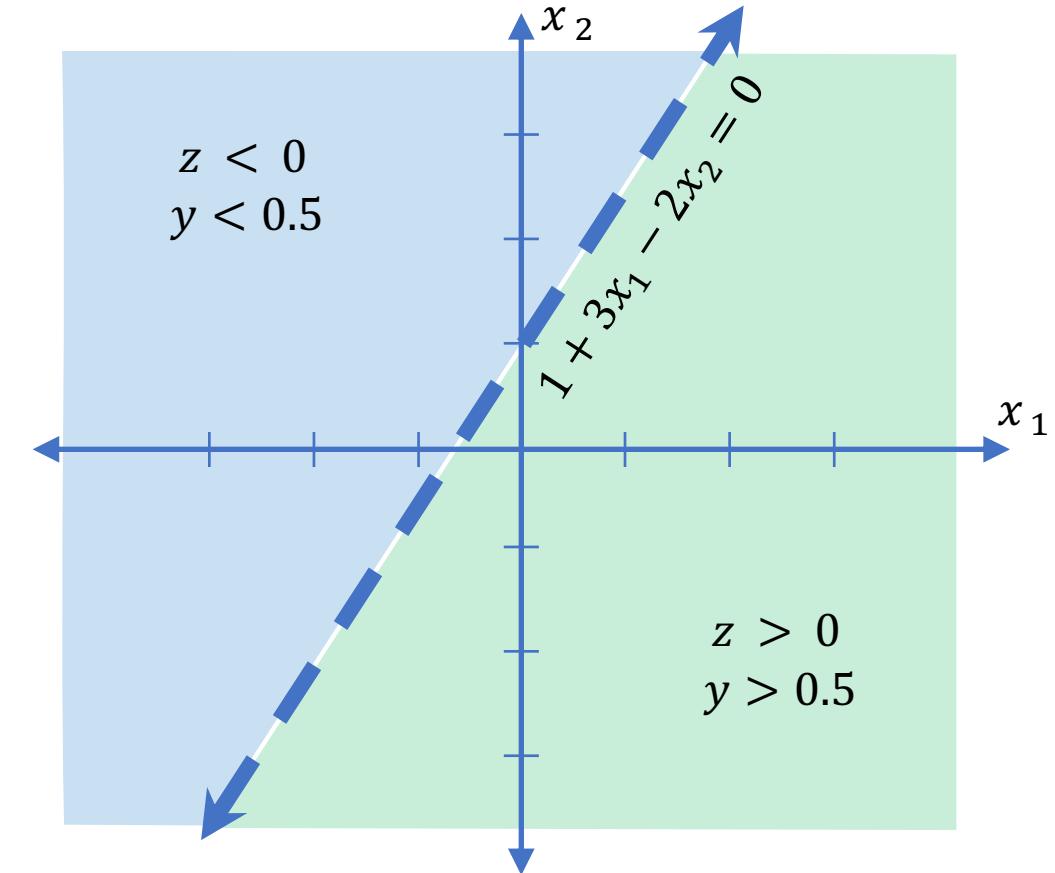
$$\hat{y} = g(1 + 3x_1 - 2x_2)$$



# The Perceptron: Example

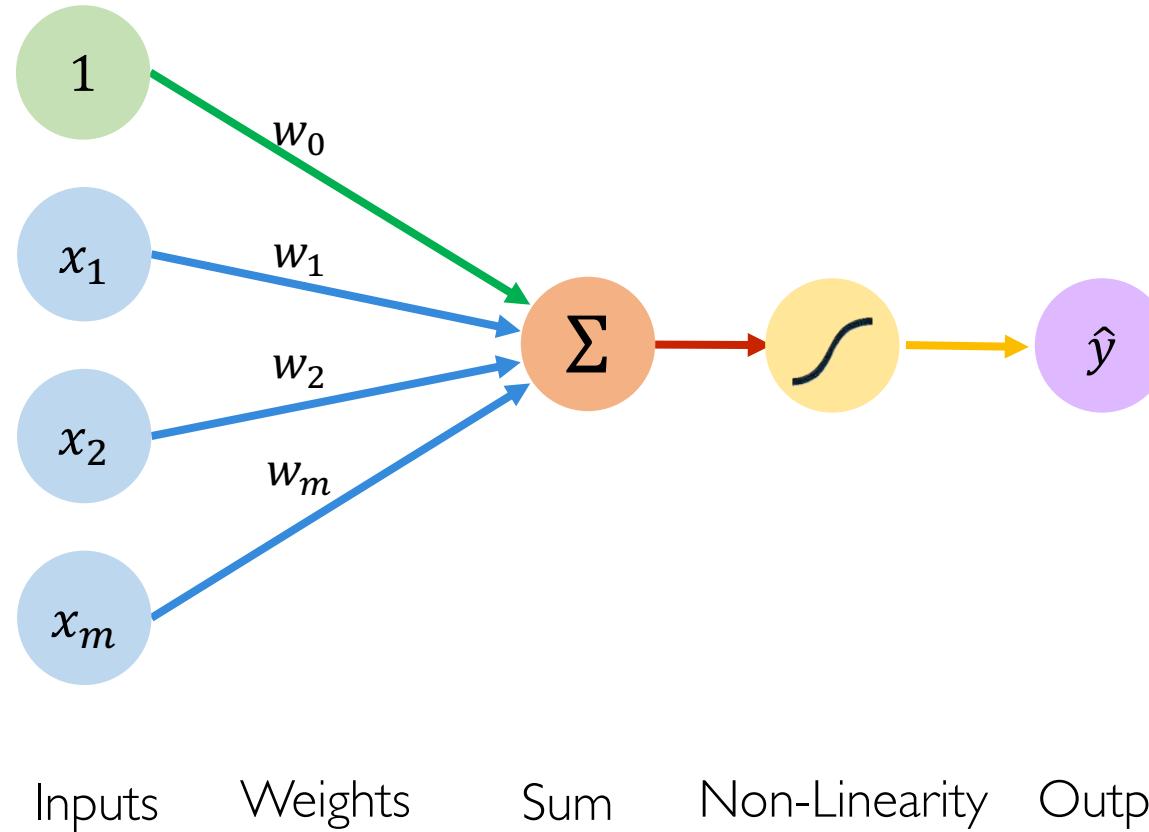


$$\hat{y} = g(1 + 3x_1 - 2x_2)$$

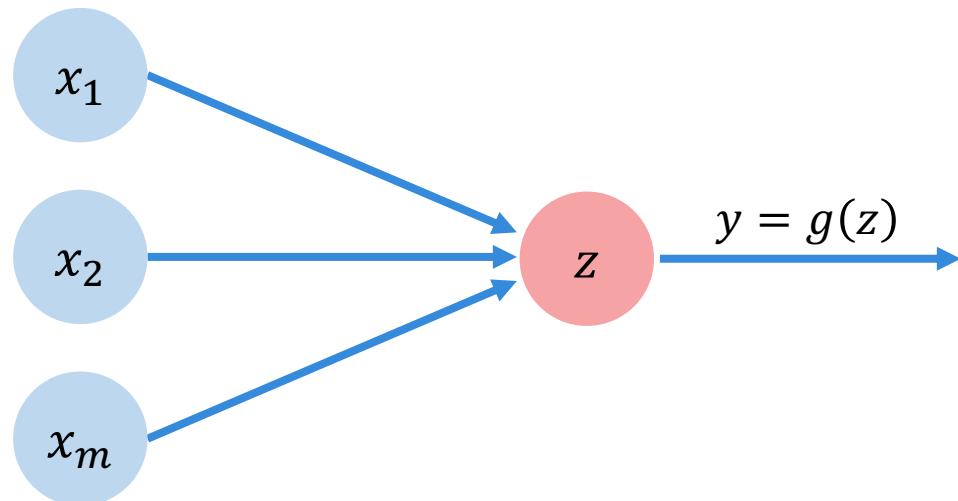


# Building Neural Networks with Perceptrons

# The Perceptron: Simplified

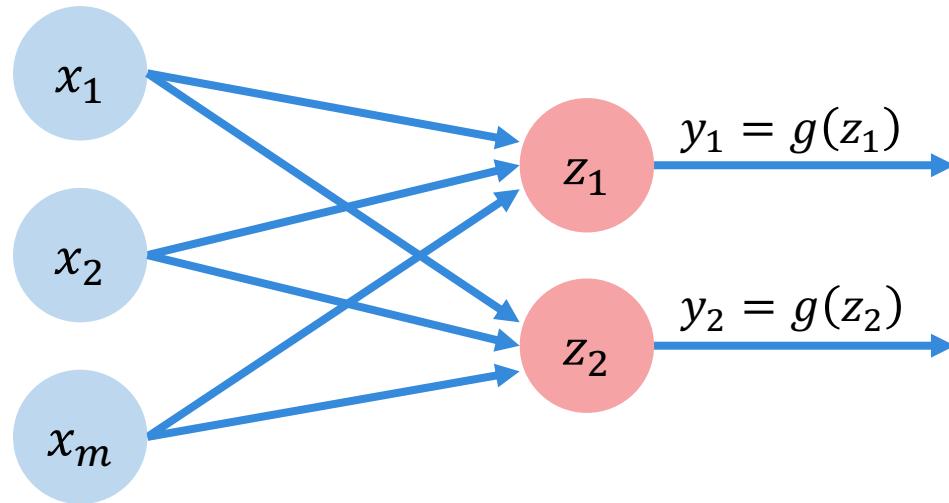


# The Perceptron: Simplified



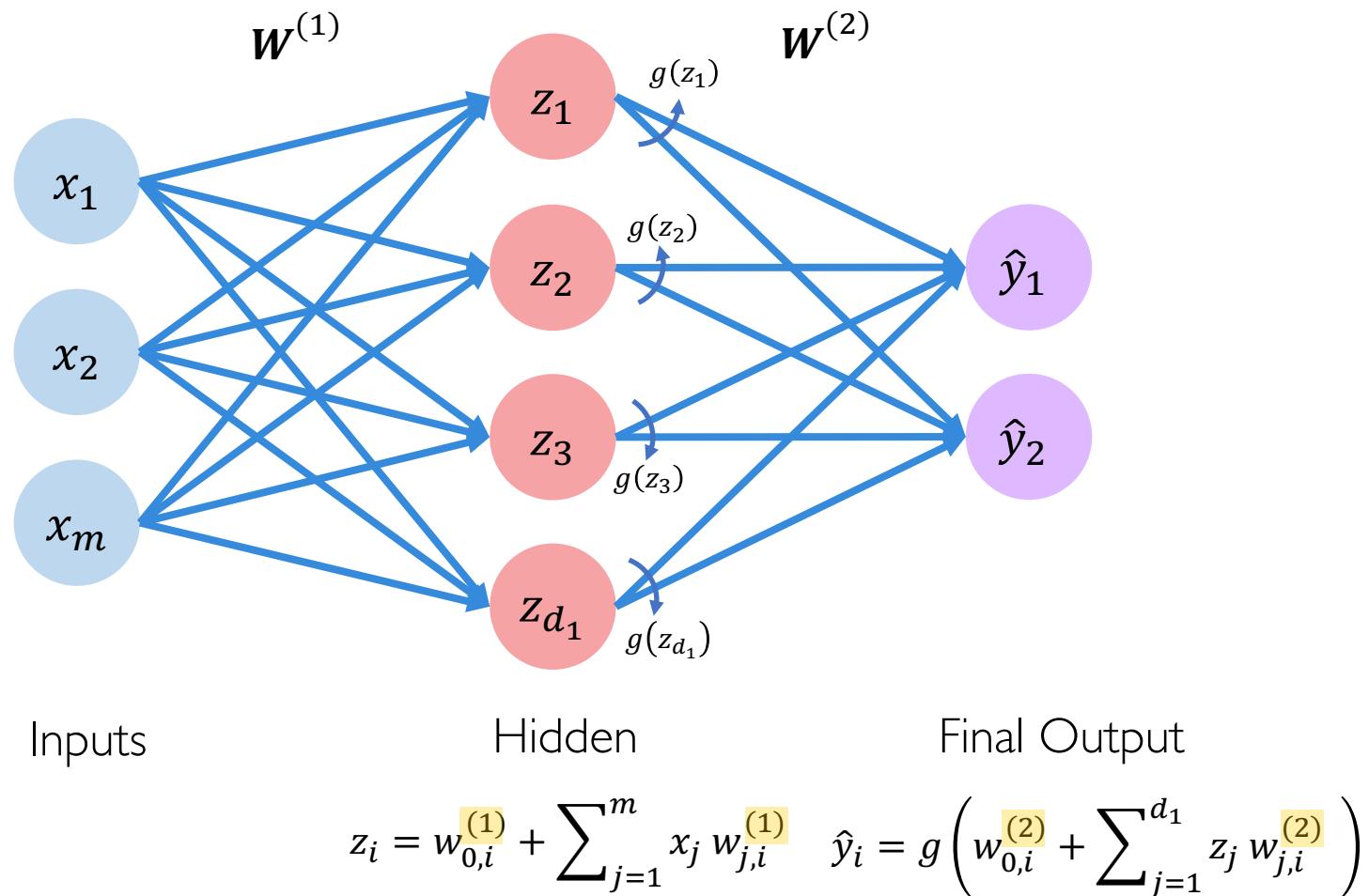
$$z = w_0 + \sum_{j=1}^m x_j w_j$$

# Multi Output Perceptron

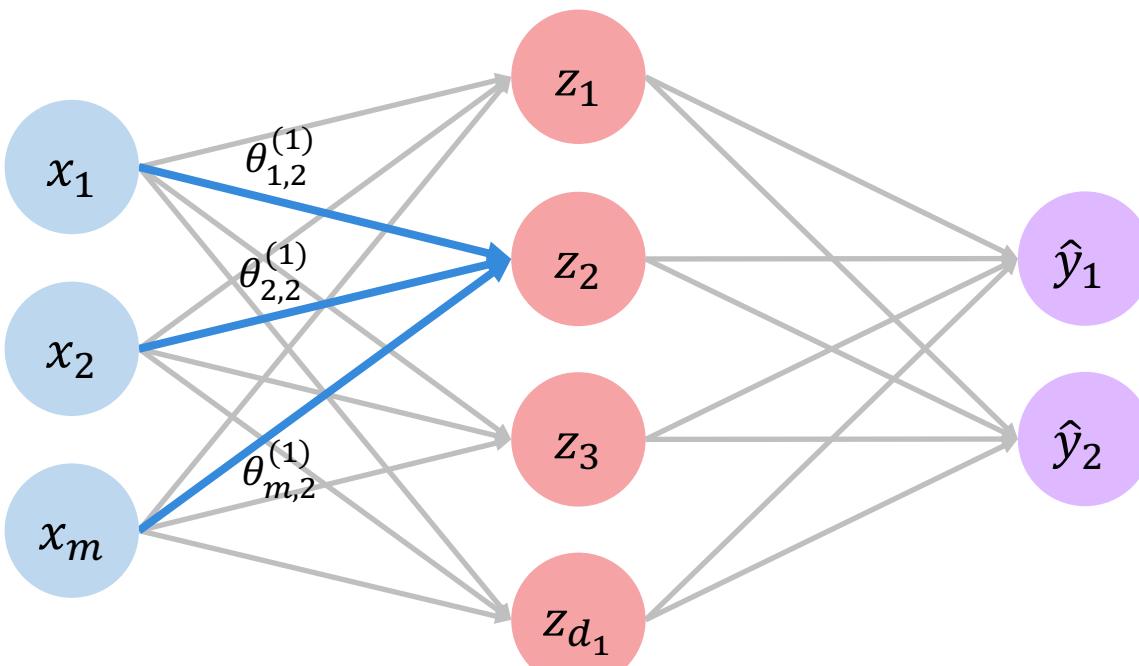


$$z_i = w_{0,i} + \sum_{j=1}^m x_j w_{j,i}$$

# Single Layer Neural Network

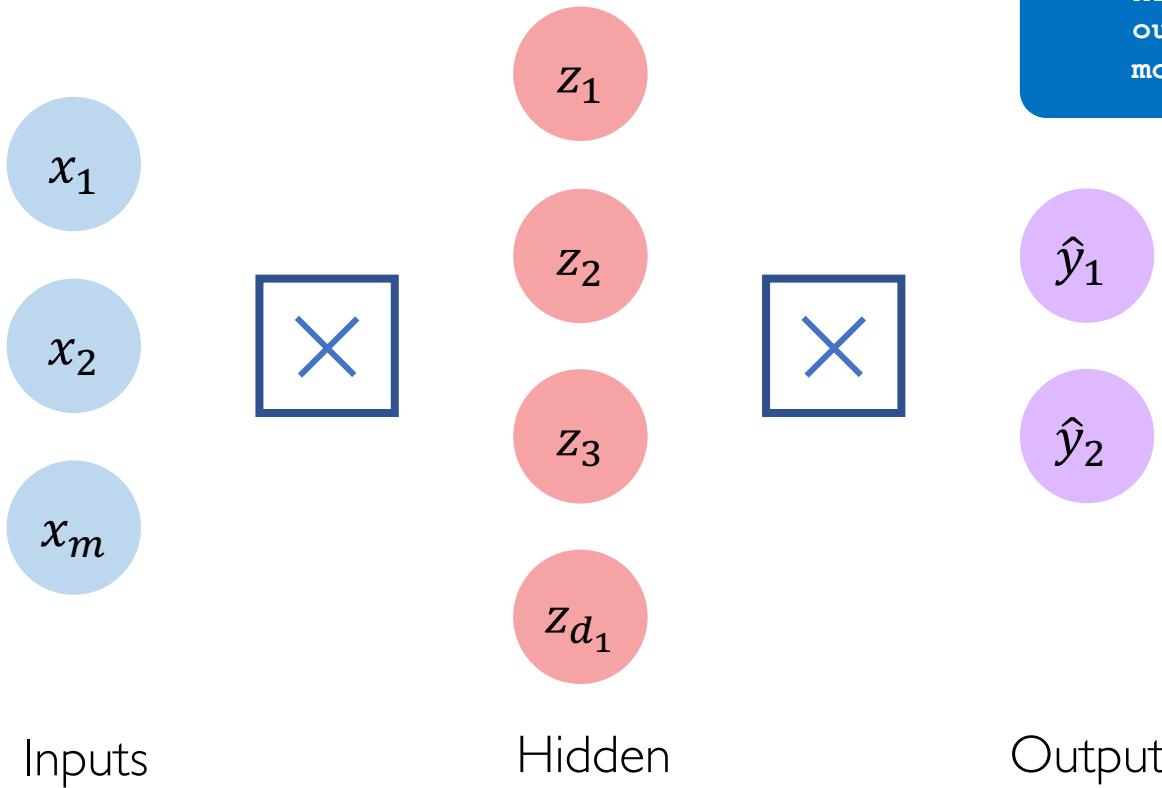


# Single Layer Neural Network



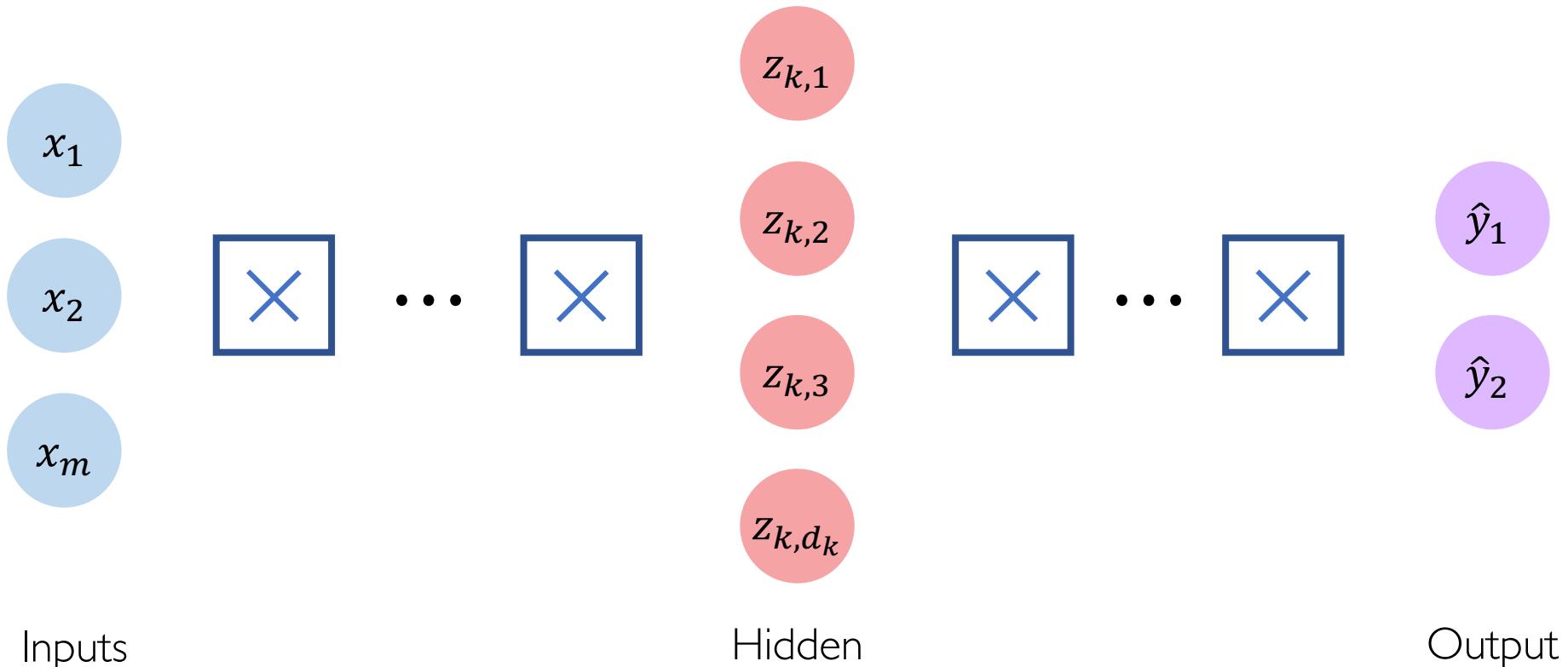
$$\begin{aligned} z_2 &= w_{0,2}^{(1)} + \sum_{j=1}^m x_j w_{j,2}^{(1)} \\ &= w_{0,2}^{(1)} + x_1 w_{1,2}^{(1)} + x_2 w_{2,2}^{(1)} + x_m w_{m,2}^{(1)} \end{aligned}$$

# Multi Output Perceptron



```
from tf.keras.layers import *
inputs = Inputs(m)
hidden = Dense(d1)(inputs)
outputs = Dense(2)(hidden)
model = Model(inputs, outputs)
```

# Deep Neural Network



$$z_{k,i} = w_{0,i}^{(k)} + \sum_{j=1}^{d_{k-1}} g(z_{k-1,j}) w_{j,i}^{(k)}$$

# Applying Neural Networks

# Example Problem

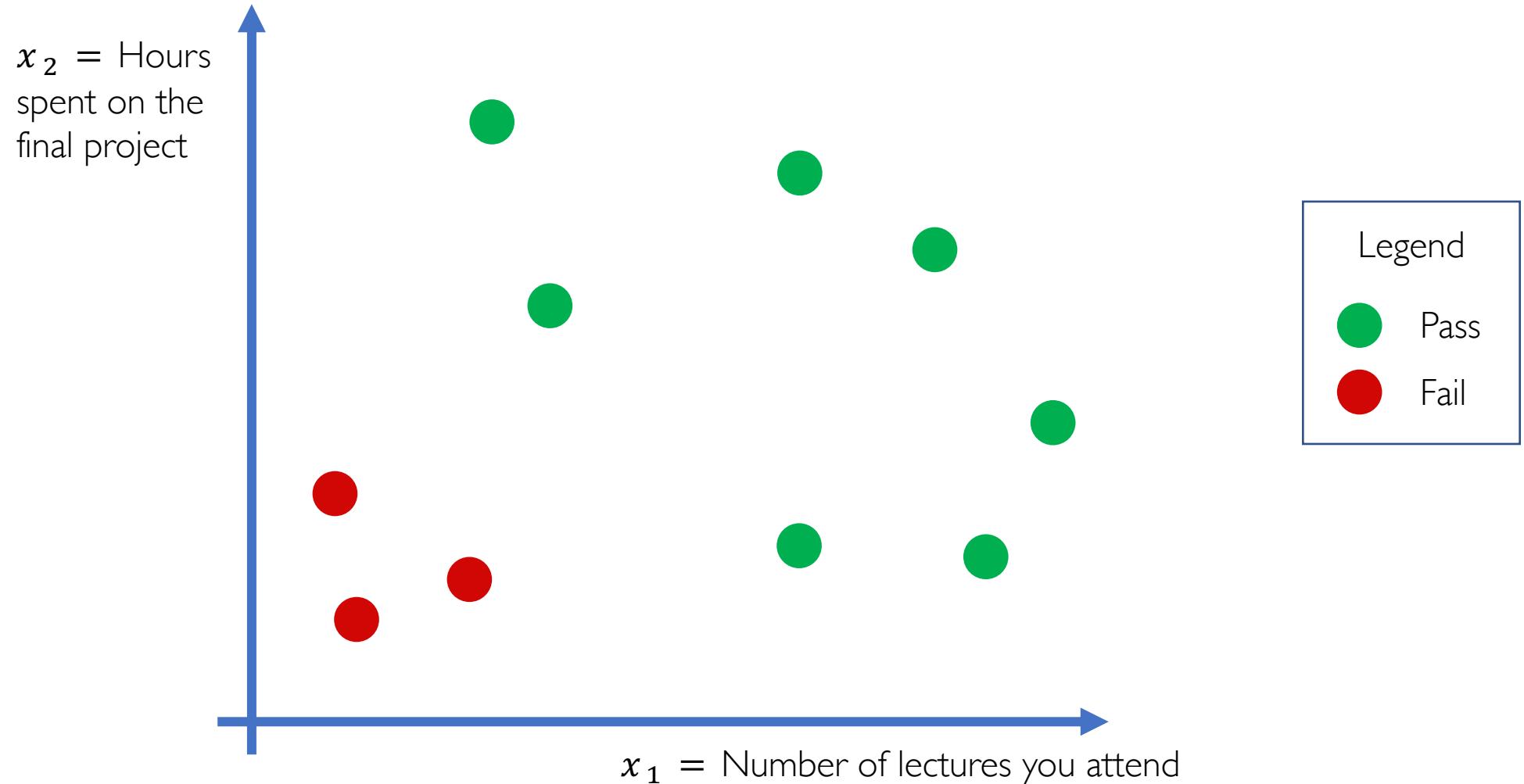
Will I pass this class?

Let's start with a simple two feature model

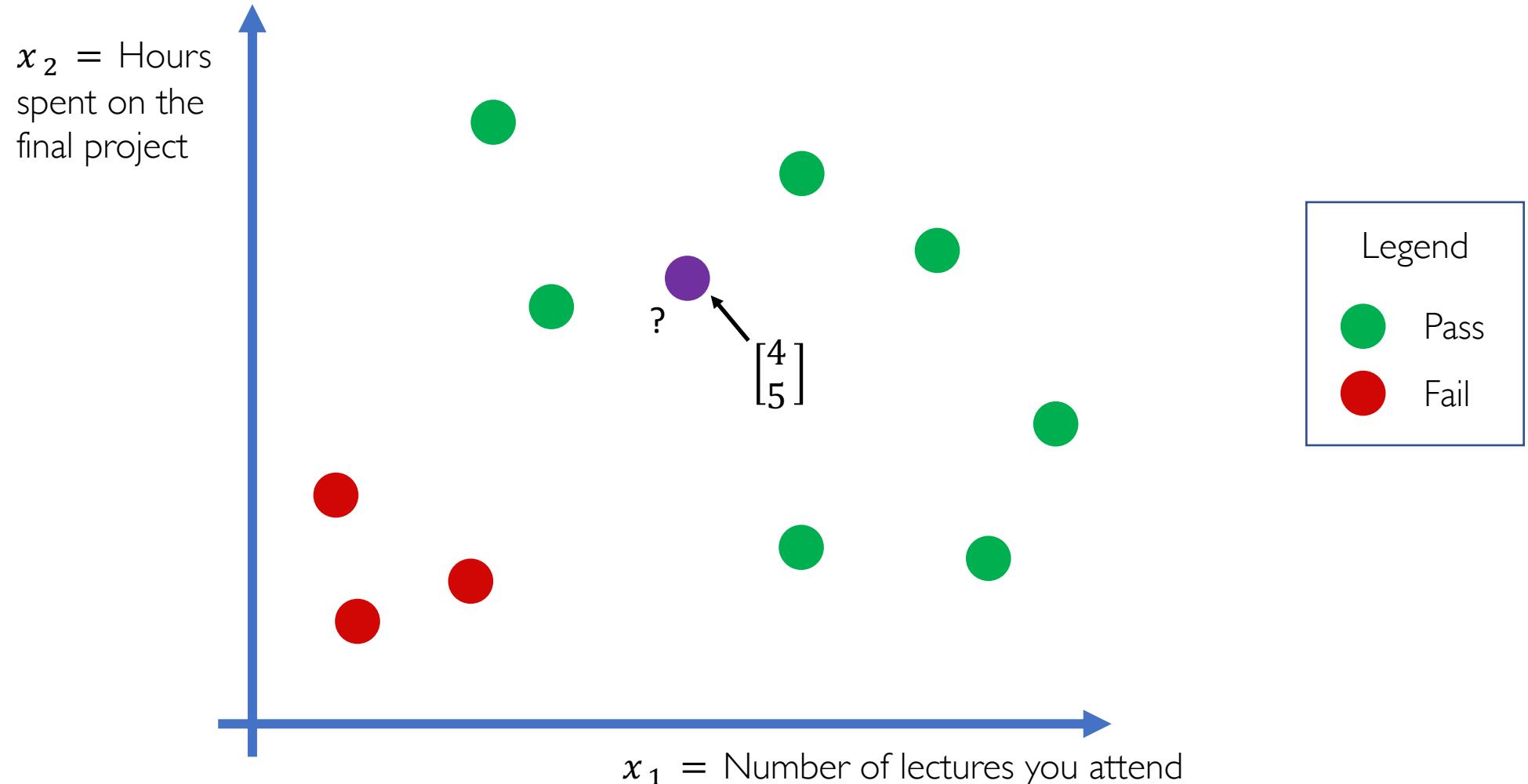
$x_1$  = Number of lectures you attend

$x_2$  = Hours spent on the final project

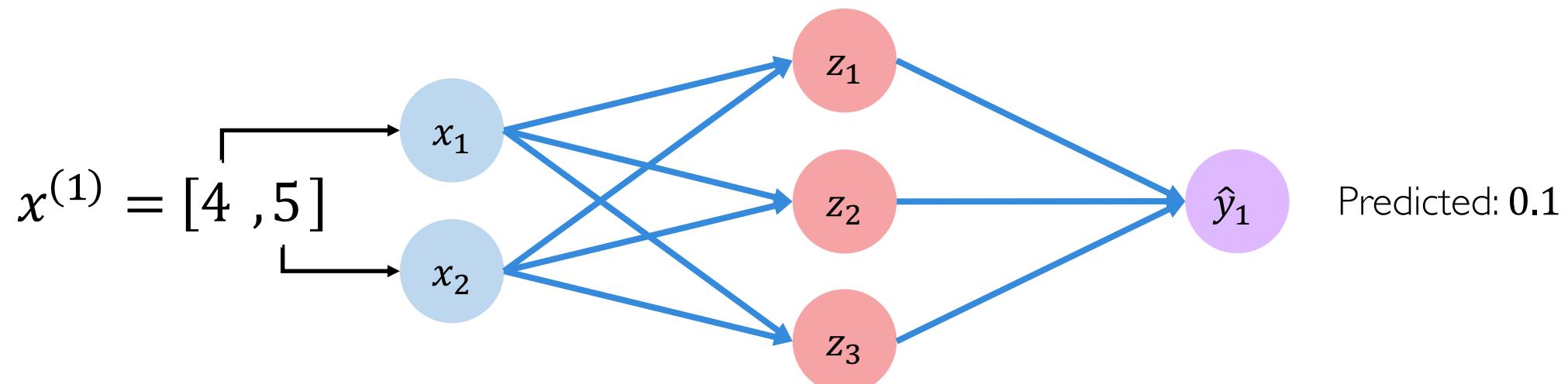
# Example Problem: Will I pass this class?



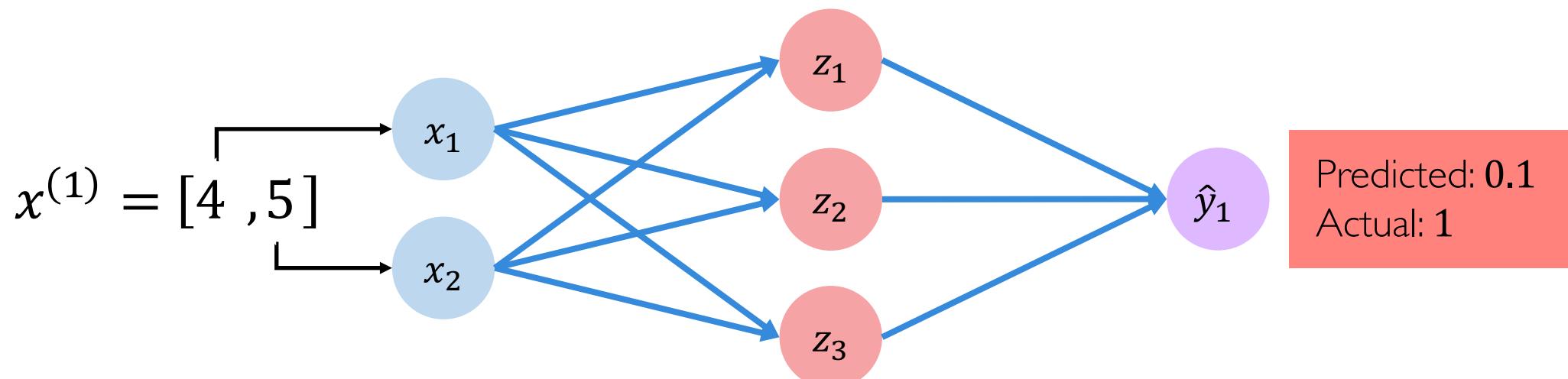
# Example Problem: Will I pass this class?



# Example Problem: Will I pass this class?

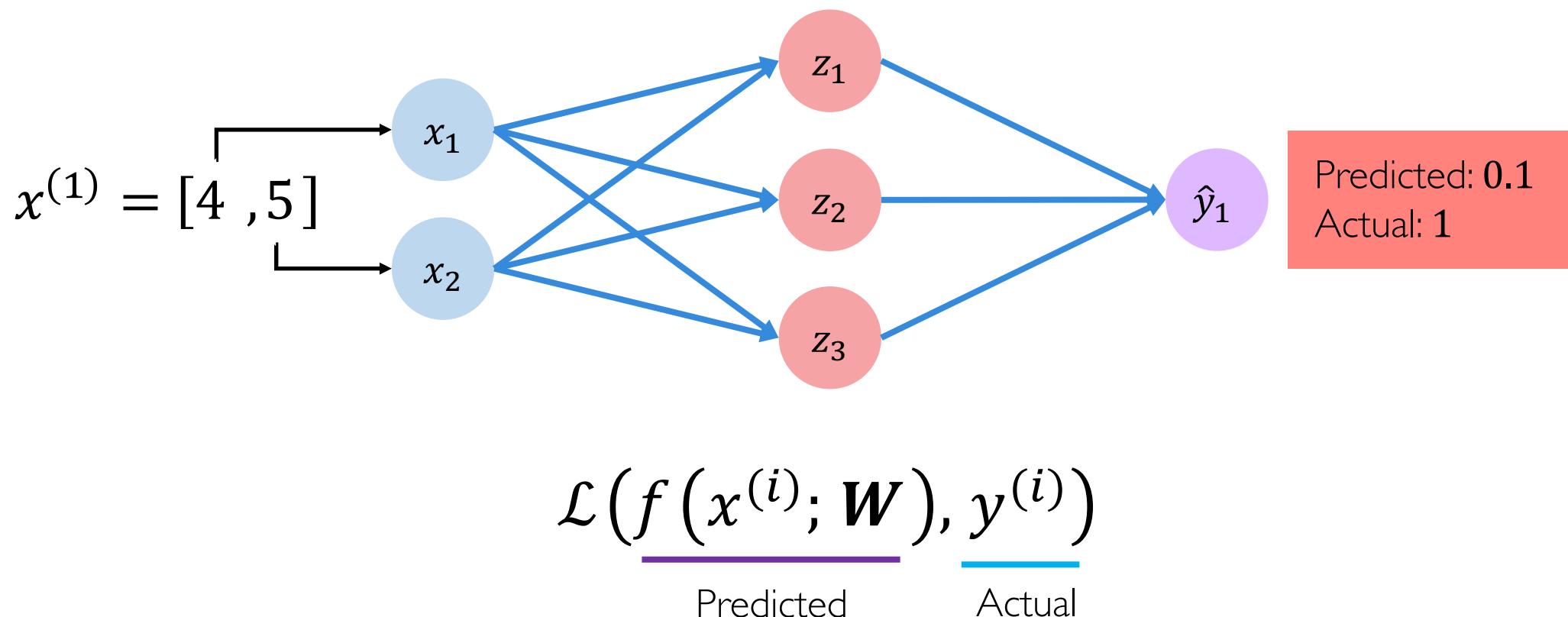


# Example Problem: Will I pass this class?



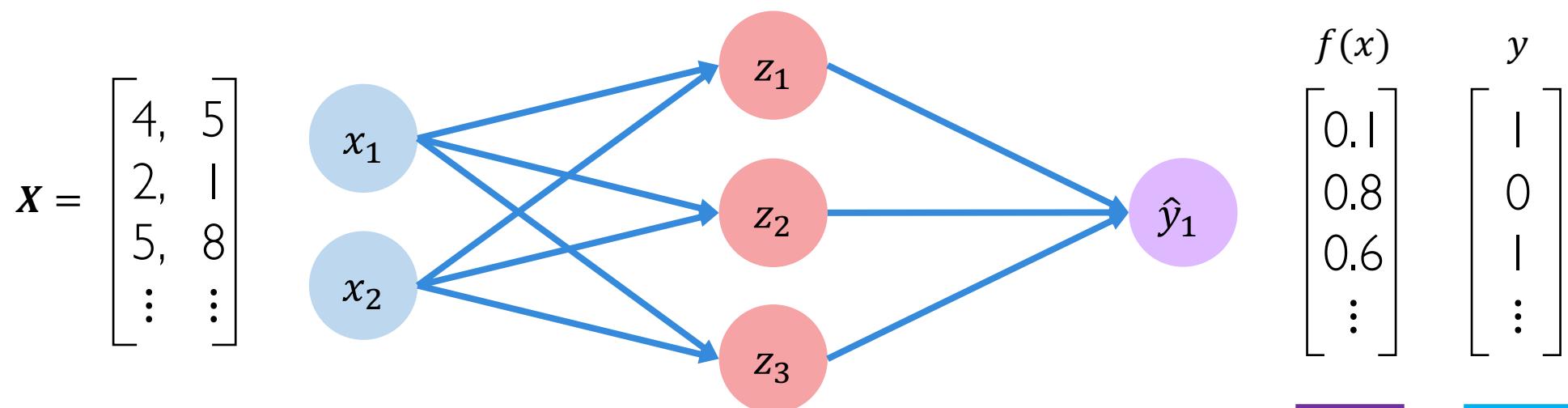
# Quantifying Loss

The *loss* of our network measures the cost incurred from incorrect predictions



# Empirical Loss

The **empirical loss** measures the total loss over our entire dataset



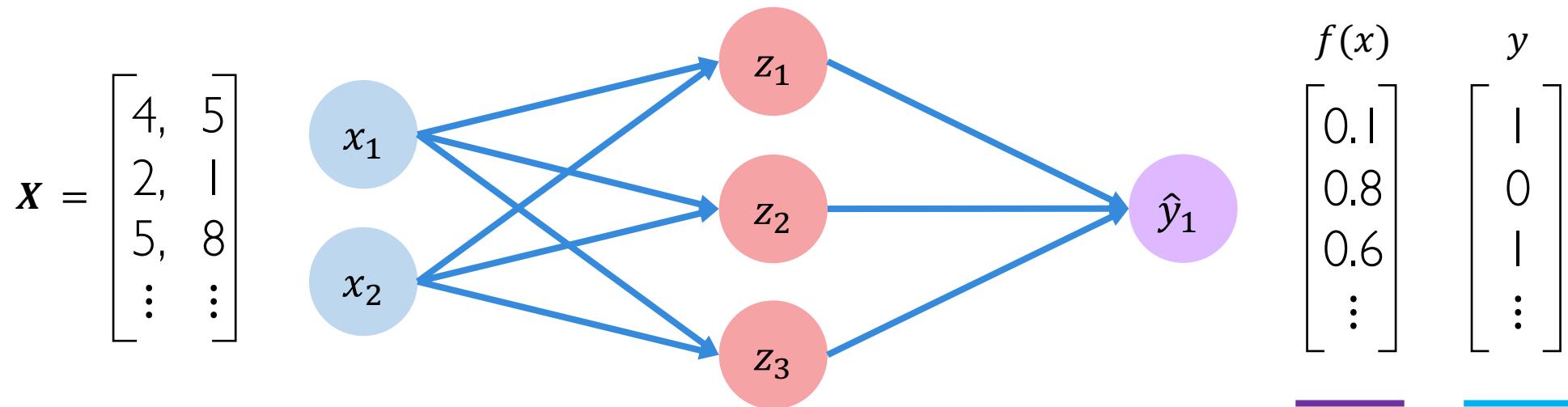
- Also known as:
- Objective function
  - Cost function
  - Empirical Risk

$J(W) = \frac{1}{n} \sum_{i=1}^n \mathcal{L}(f(x^{(i)}; W), y^{(i)})$

Predicted                      Actual

# Binary Cross Entropy Loss

Cross entropy loss can be used with models that output a probability between 0 and 1



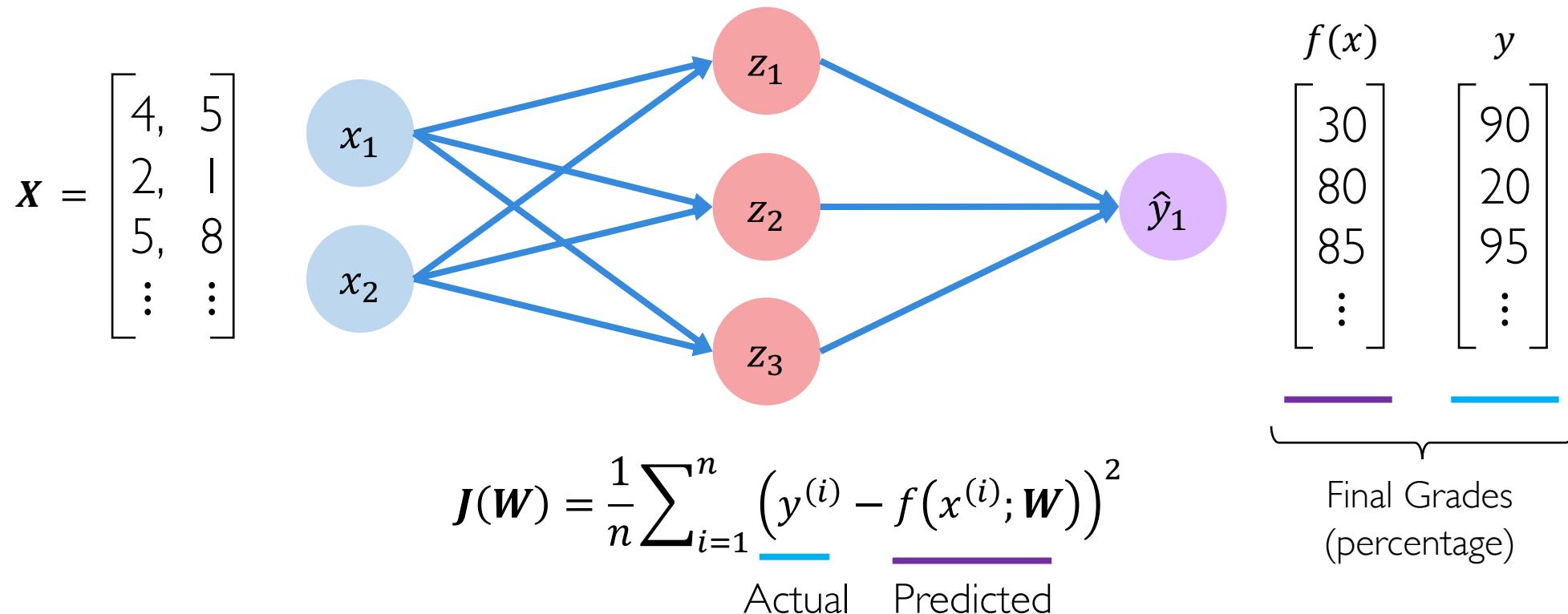
$$J(\mathbf{W}) = \frac{1}{n} \sum_{i=1}^n \underbrace{y^{(i)}}_{\text{Actual}} \underbrace{\log(f(x^{(i)}; \mathbf{W}))}_{\text{Predicted}} + (1 - \underbrace{y^{(i)}}_{\text{Actual}}) \underbrace{\log(1 - f(x^{(i)}; \mathbf{W}))}_{\text{Predicted}}$$



```
loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(model.y, model.pred))
```

# Mean Squared Error Loss

**Mean squared error loss** can be used with regression models that output continuous real numbers



```
loss = tf.reduce_mean( tf.square(tf.subtract(model.y, model.pred)) )
```

# Training Neural Networks

# Loss Optimization

We want to find the network weights that **achieve the lowest loss**

$$\mathbf{W}^* = \operatorname{argmin}_{\mathbf{W}} \frac{1}{n} \sum_{i=1}^n \mathcal{L}(f(x^{(i)}; \mathbf{W}), y^{(i)})$$

$$\mathbf{W}^* = \operatorname{argmin}_{\mathbf{W}} J(\mathbf{W})$$

# Loss Optimization

We want to find the network weights that **achieve the lowest loss**

$$\mathbf{W}^* = \operatorname{argmin}_{\mathbf{W}} \frac{1}{n} \sum_{i=1}^n \mathcal{L}(f(x^{(i)}; \mathbf{W}), y^{(i)})$$

$$\mathbf{W}^* = \operatorname{argmin}_{\mathbf{W}} J(\mathbf{W})$$

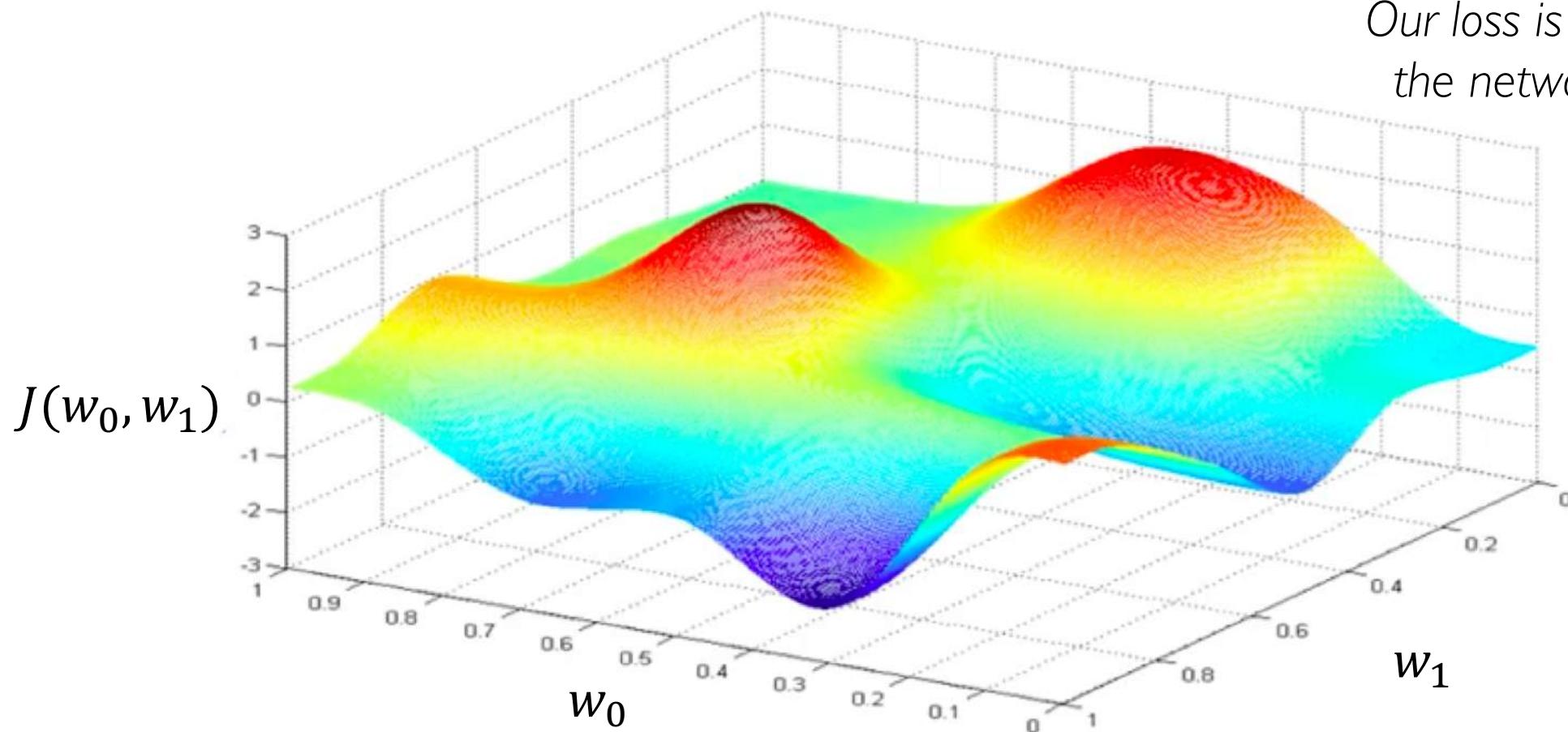


Remember:

$$\mathbf{W} = \{\mathbf{W}^{(0)}, \mathbf{W}^{(1)}, \dots\}$$

# Loss Optimization

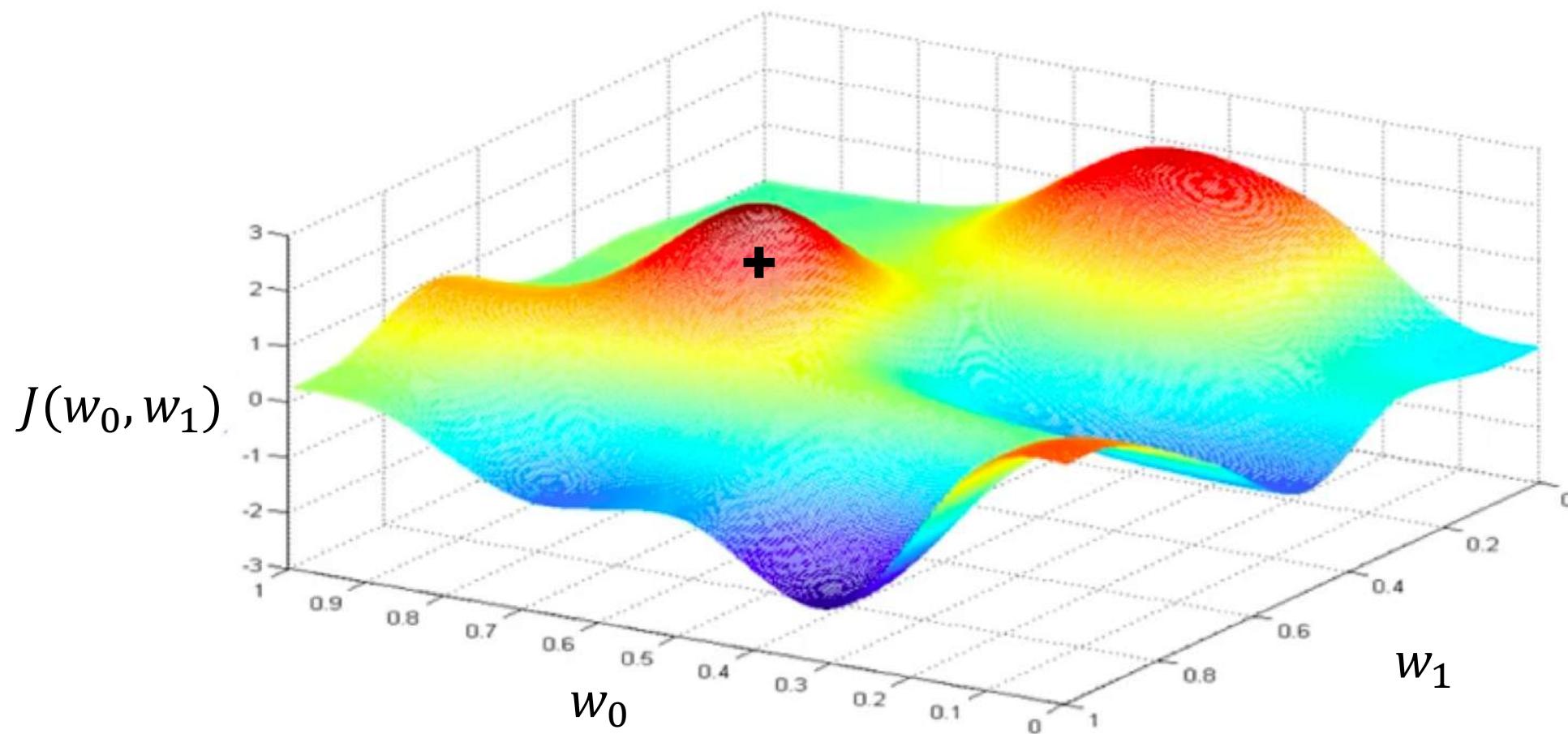
$$\mathbf{W}^* = \underset{\mathbf{W}}{\operatorname{argmin}} J(\mathbf{W})$$



Remember:  
Our loss is a function of  
the network weights!

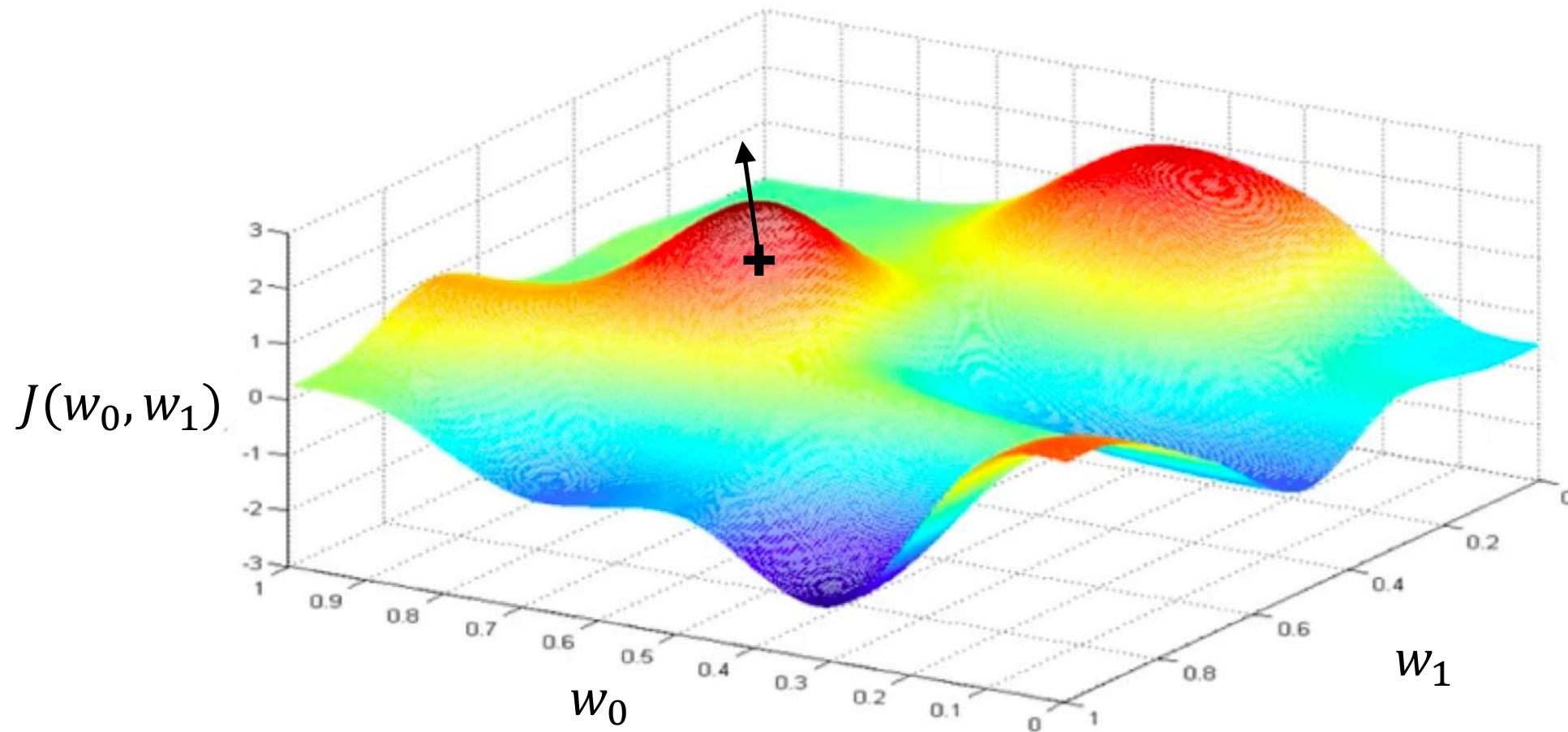
# Loss Optimization

Randomly pick an initial  $(w_0, w_1)$



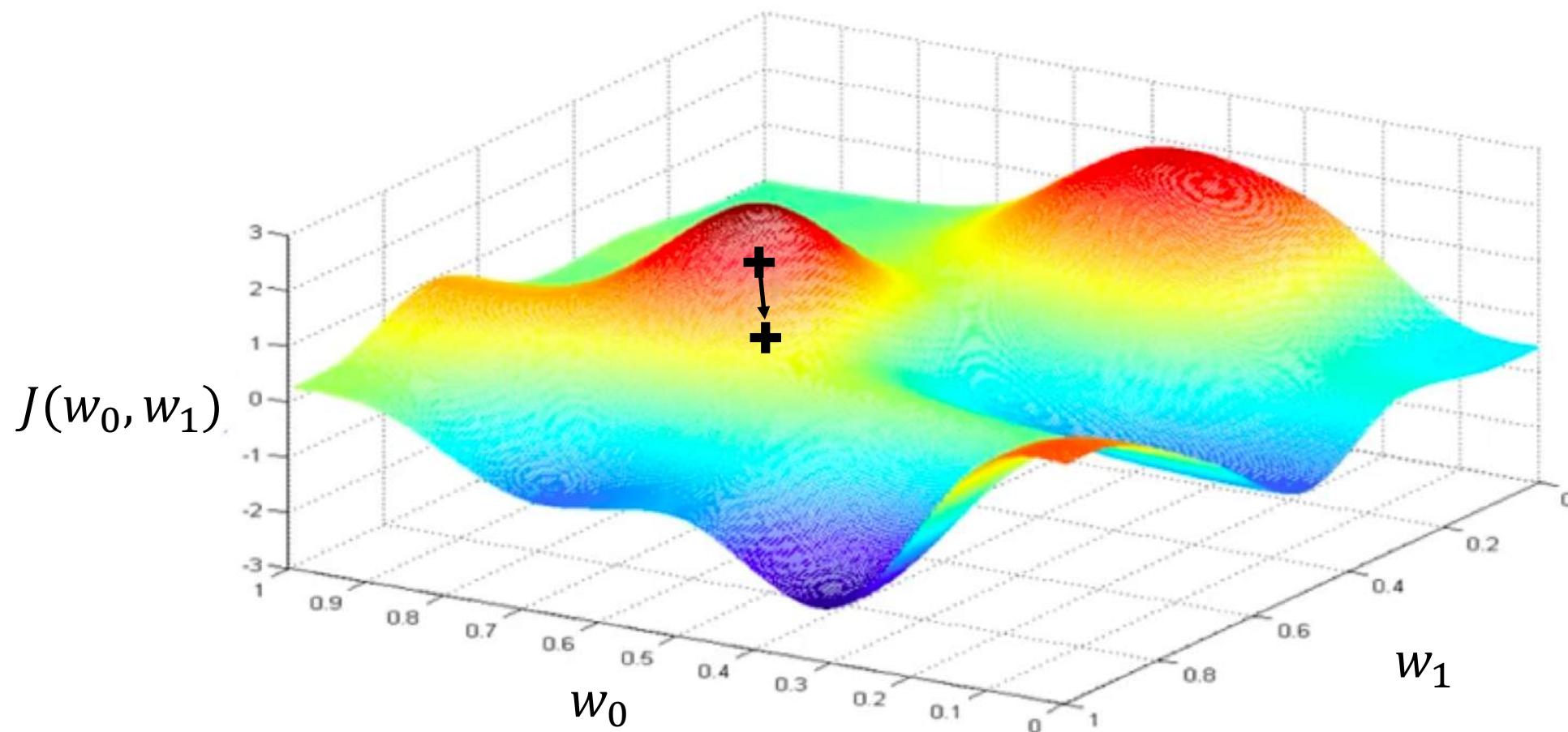
# Loss Optimization

Compute gradient,  $\frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$



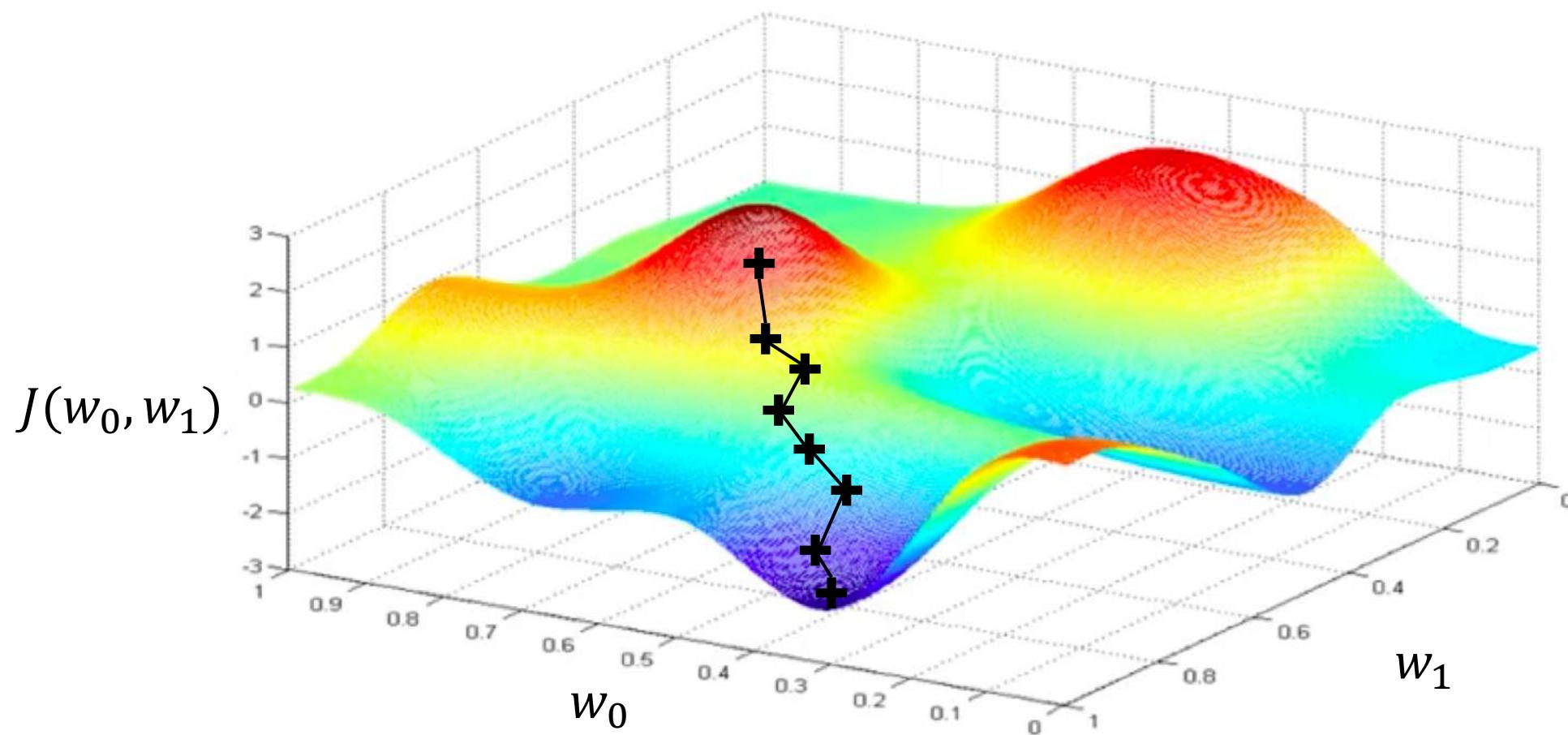
# Loss Optimization

Take small step in opposite direction of gradient



# Gradient Descent

Repeat until convergence



# Gradient Descent

## Algorithm

1. Initialize weights randomly  $\sim \mathcal{N}(0, \sigma^2)$

 weights = tf.random\_normal(shape, stddev=sigma)

2. Loop until convergence:

3. Compute gradient,  $\frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$

 grads = tf.gradients(ys=loss, xs=weights)

4. Update weights,  $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$

 weights\_new = weights.assign(weights - lr \* grads)

5. Return weights

# Gradient Descent

## Algorithm

1. Initialize weights randomly  $\sim \mathcal{N}(0, \sigma^2)$

 weights = tf.random\_normal(shape, stddev=sigma)

2. Loop until convergence:

3. Compute gradient,  $\frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$

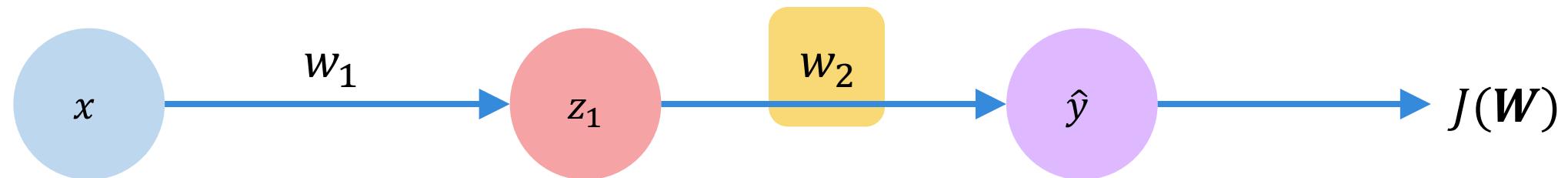
 grads = tf.gradients(ys=loss, xs=weights)

4. Update weights,  $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$

 weights\_new = weights.assign(weights - lr \* grads)

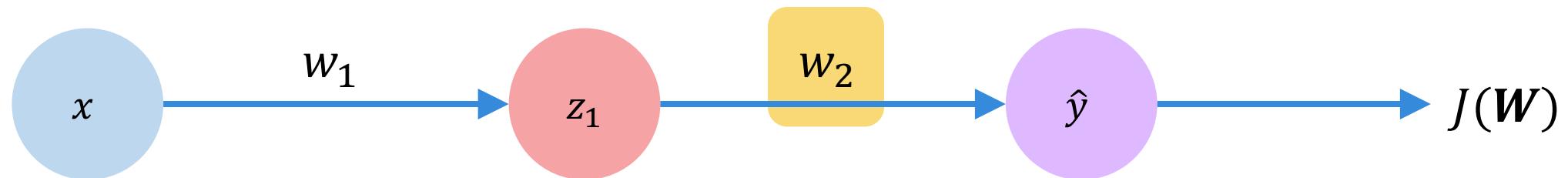
5. Return weights

# Computing Gradients: Backpropagation



How does a small change in one weight (ex.  $w_2$ ) affect the final loss  $J(\mathbf{W})$ ?

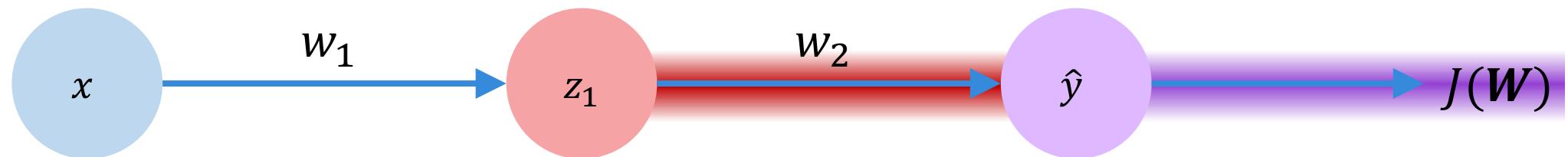
# Computing Gradients: Backpropagation



$$\frac{\partial J(\mathbf{W})}{\partial w_2} =$$

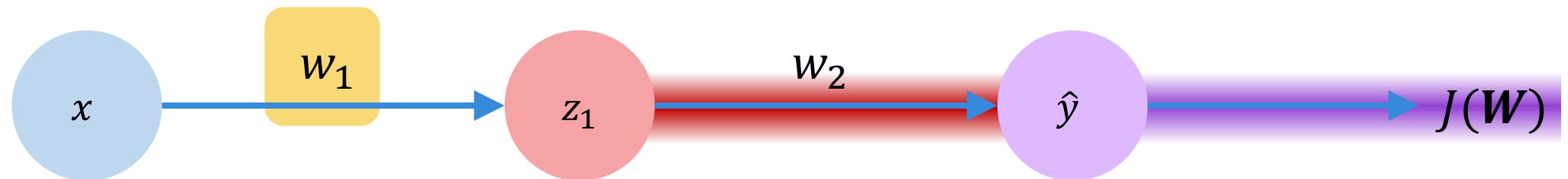
Let's use the chain rule!

# Computing Gradients: Backpropagation



$$\frac{\partial J(\mathbf{W})}{\partial w_2} = \underline{\frac{\partial J(\mathbf{W})}{\partial \hat{y}}} * \underline{\frac{\partial \hat{y}}{\partial w_2}}$$

# Computing Gradients: Backpropagation

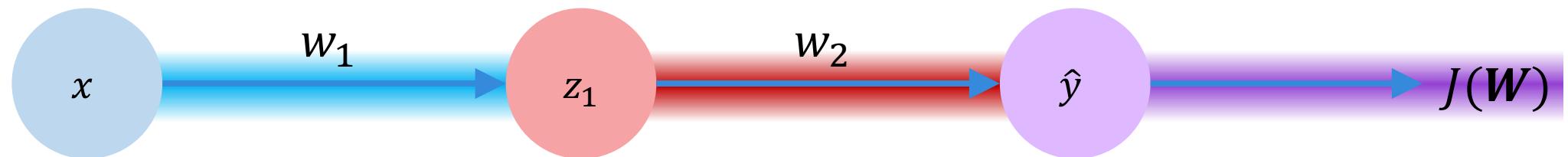


$$\frac{\partial J(\mathbf{W})}{\partial w_1} = \frac{\partial J(\mathbf{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_1}$$

Apply chain rule!

Apply chain rule!

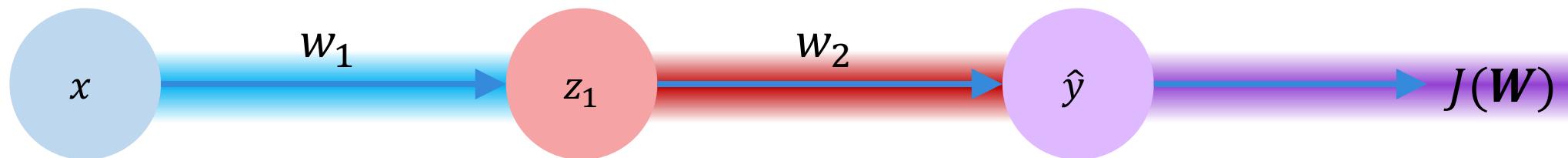
# Computing Gradients: Backpropagation



$$\frac{\partial J(\mathbf{W})}{\partial w_1} = \underline{\frac{\partial J(\mathbf{W})}{\partial \hat{y}}} * \underline{\frac{\partial \hat{y}}{\partial z_1}} * \underline{\frac{\partial z_1}{\partial w_1}}$$



# Computing Gradients: Backpropagation

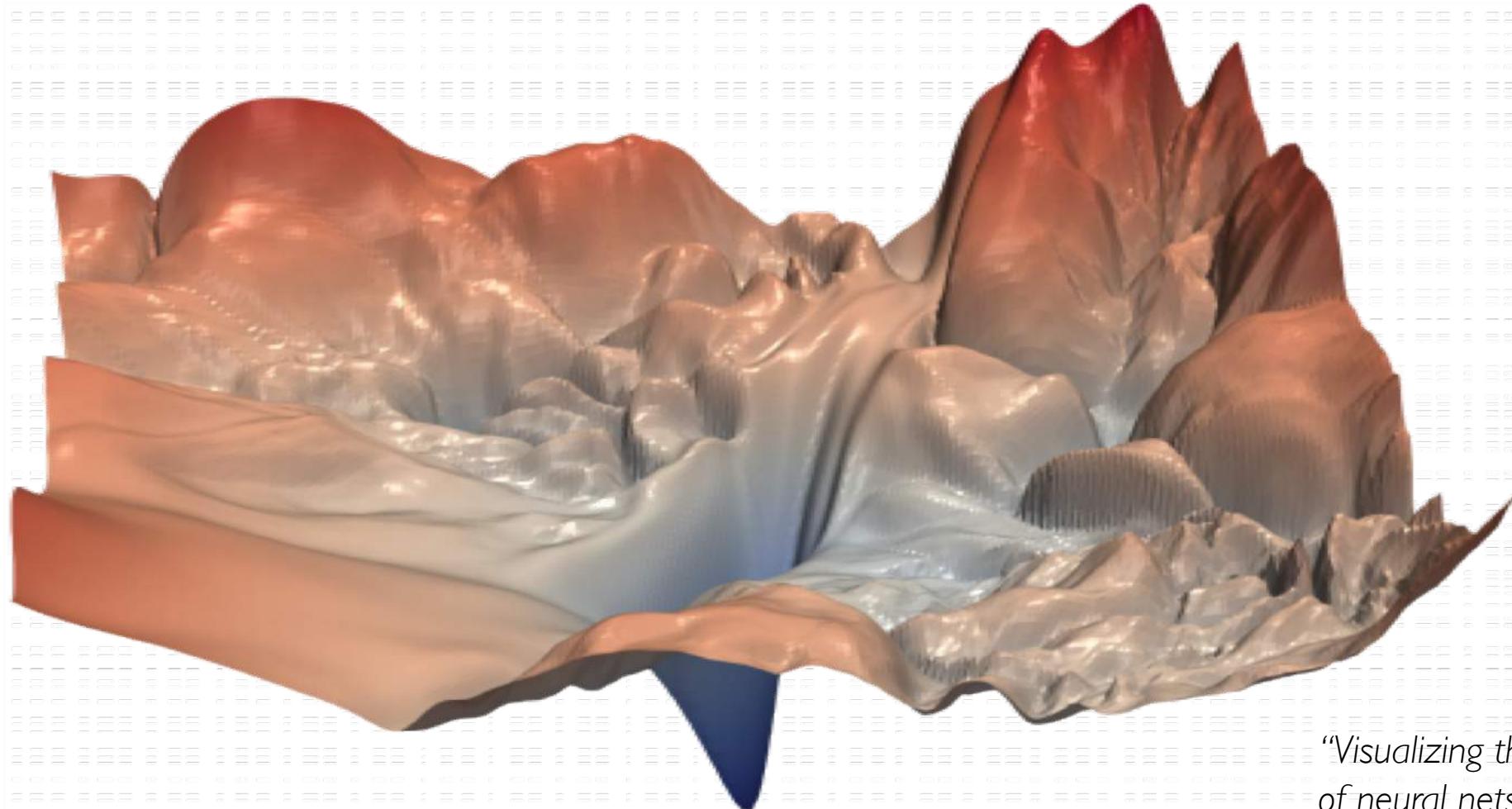


$$\frac{\partial J(\mathbf{W})}{\partial w_1} = \underline{\frac{\partial J(\mathbf{W})}{\partial \hat{y}}} * \underline{\frac{\partial \hat{y}}{\partial z_1}} * \underline{\frac{\partial z_1}{\partial w_1}}$$

Repeat this for **every weight in the network** using gradients from later layers

# Neural Networks in Practice: Optimization

# Training Neural Networks is Difficult



*“Visualizing the loss landscape  
of neural nets”. Dec 2017.*

# Loss Functions Can Be Difficult to Optimize

**Remember:**

Optimization through gradient descent

$$\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$$

# Loss Functions Can Be Difficult to Optimize

**Remember:**

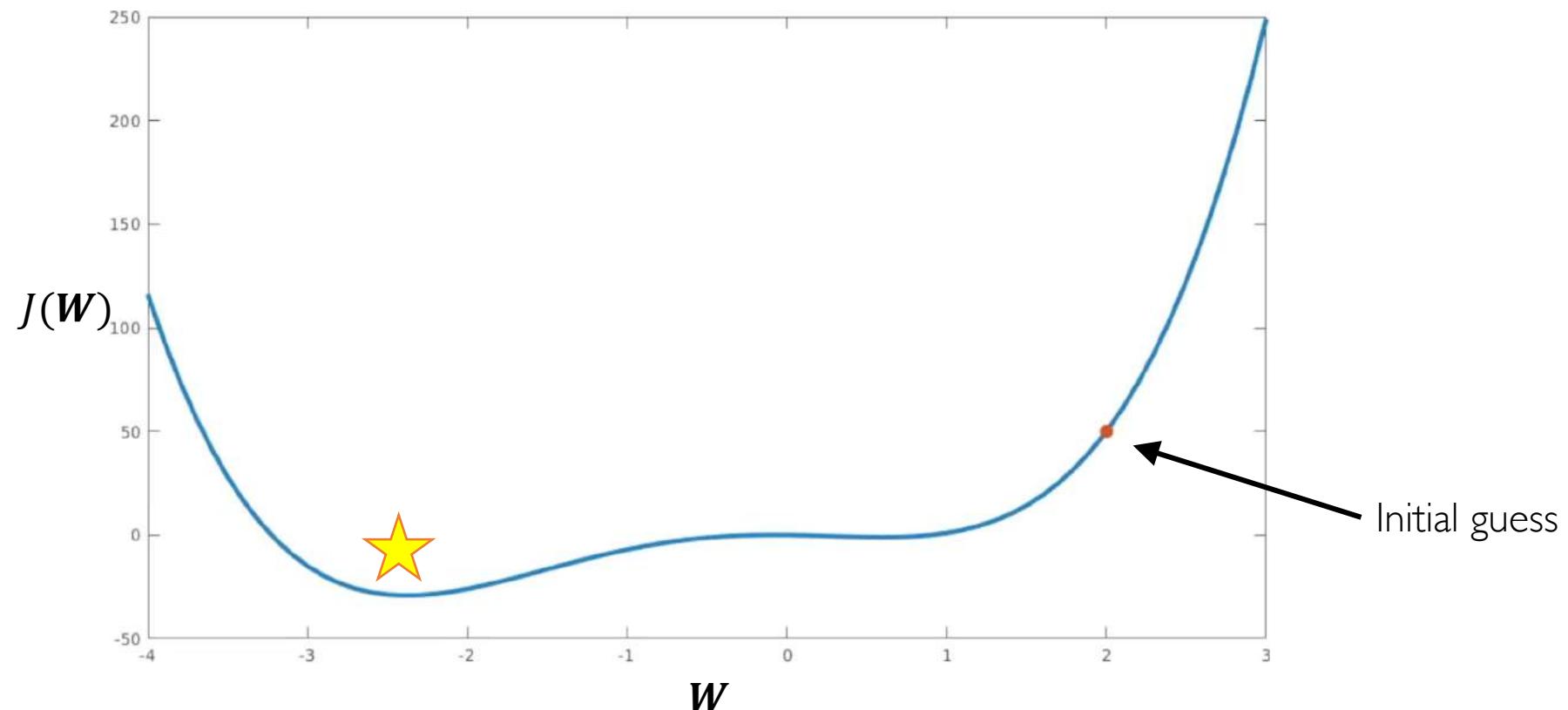
Optimization through gradient descent

$$\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$$

How can we set the  
learning rate?

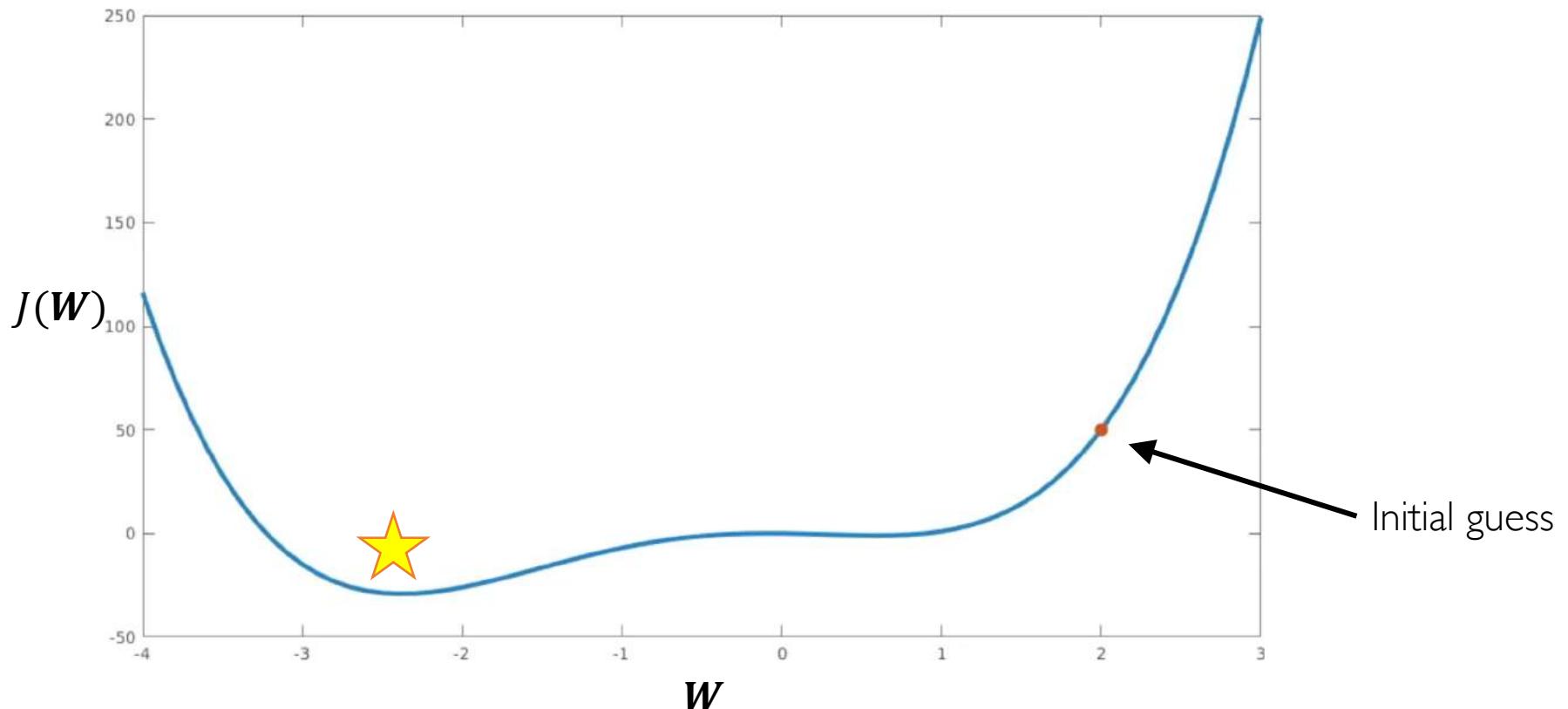
# Setting the Learning Rate

*Small learning rate* converges slowly and gets stuck in false local minima



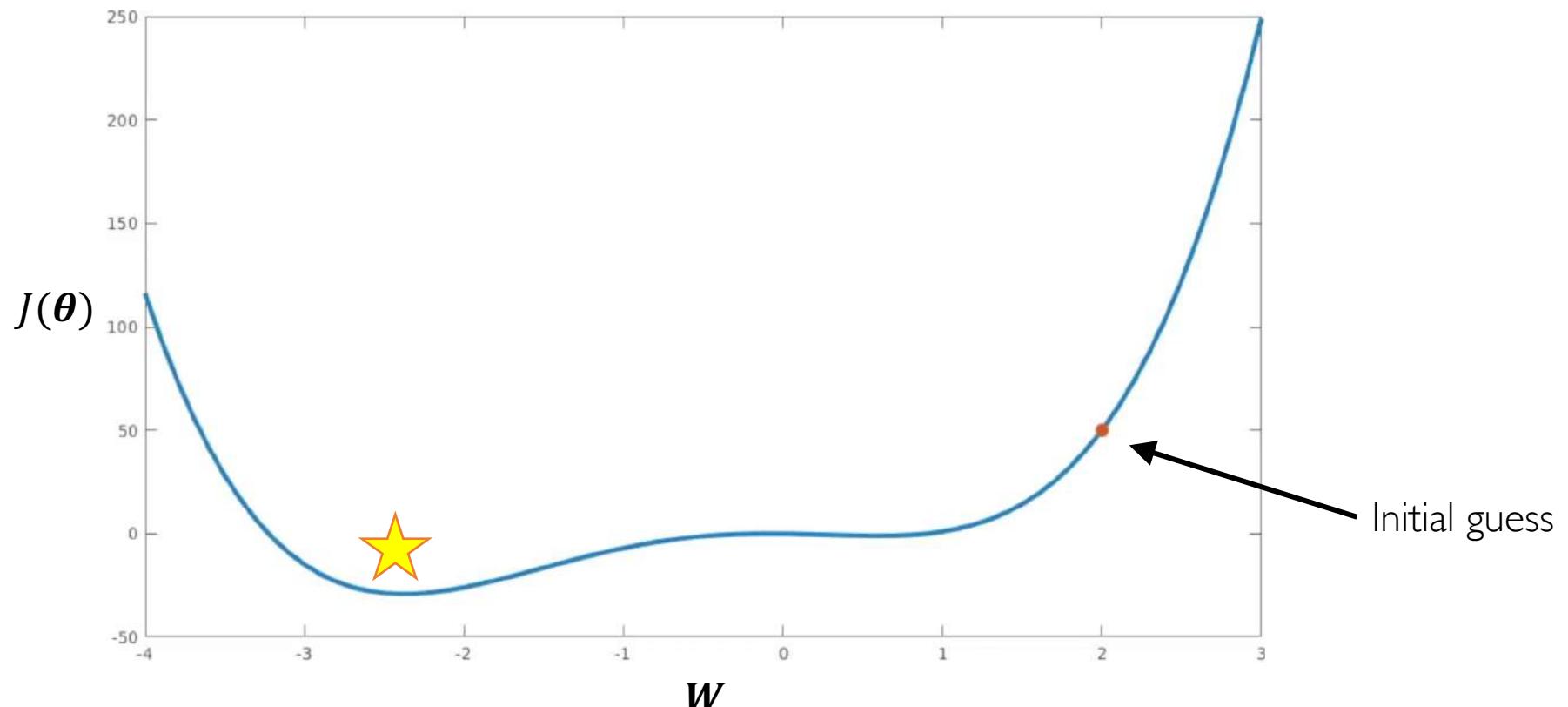
# Setting the Learning Rate

*Large learning rates* overshoot, become unstable and diverge



# Setting the Learning Rate

*Stable learning rates* converge smoothly and avoid local minima



# How to deal with this?

## Idea I:

Try lots of different learning rates and see what works “just right”

# How to deal with this?

## Idea 1:

Try lots of different learning rates and see what works “just right”

## Idea 2:

Do something smarter!

Design an adaptive learning rate that “adapts” to the landscape

# Adaptive Learning Rates

- Learning rates are no longer fixed
- Can be made larger or smaller depending on:
  - how large gradient is
  - how fast learning is happening
  - size of particular weights
  - etc...

# Adaptive Learning Rate Algorithms

- Momentum
- Adagrad
- Adadelta
- Adam
- RMSProp



`tf.train.MomentumOptimizer`



`tf.train.AdagradOptimizer`



`tf.train.AdadeltaOptimizer`



`tf.train.AdamOptimizer`



`tf.train.RMSPropOptimizer`

Qian et al. "On the momentum term in gradient descent learning algorithms." 1999.

Duchi et al. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization." 2011.

Zeiler et al. "ADADELTA: An Adaptive Learning Rate Method." 2012.

Kingma et al. "Adam: A Method for Stochastic Optimization." 2014.

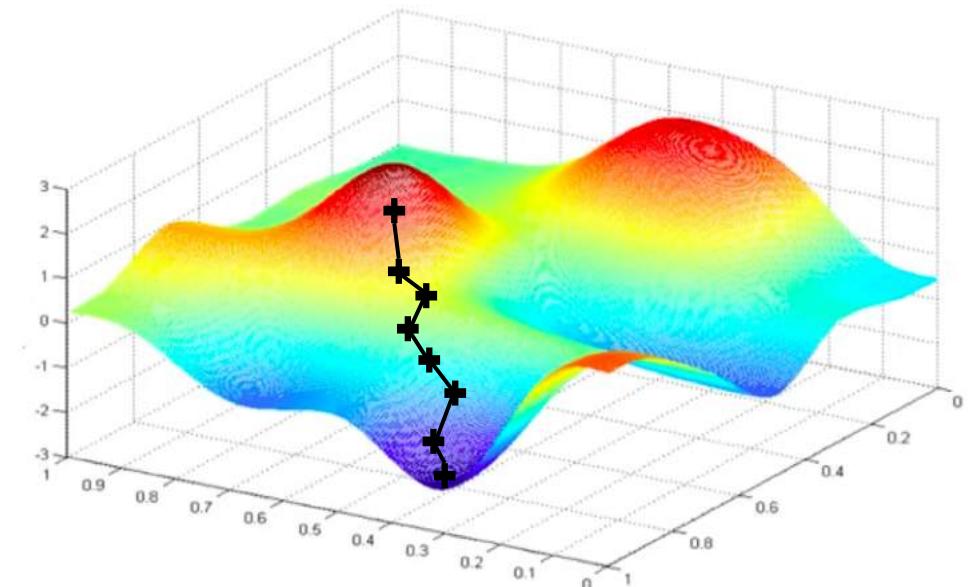
Additional details: <http://ruder.io/optimizing-gradient-descent/>

# Neural Networks in Practice: Mini-batches

# Gradient Descent

## Algorithm

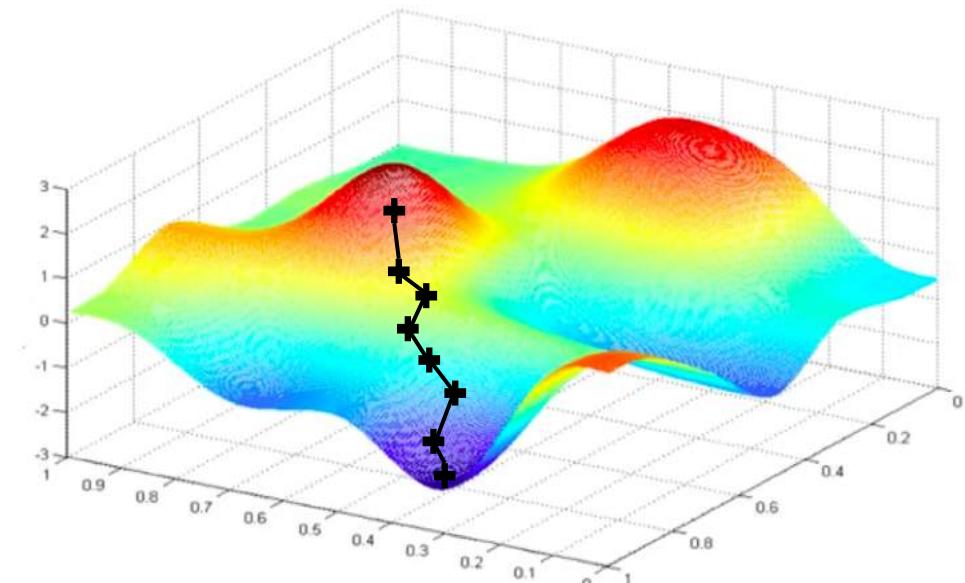
1. Initialize weights randomly  $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient,  $\frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$
4. Update weights,  $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$
5. Return weights



# Gradient Descent

## Algorithm

1. Initialize weights randomly  $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient,  $\frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$
4. Update weights,  $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$
5. Return weights

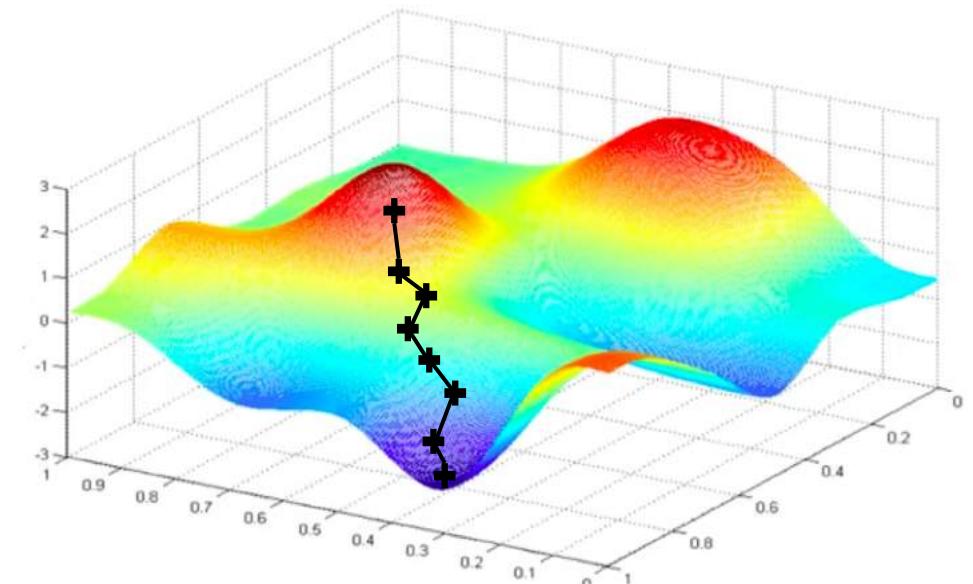


Can be very  
computational to  
compute!

# Stochastic Gradient Descent

## Algorithm

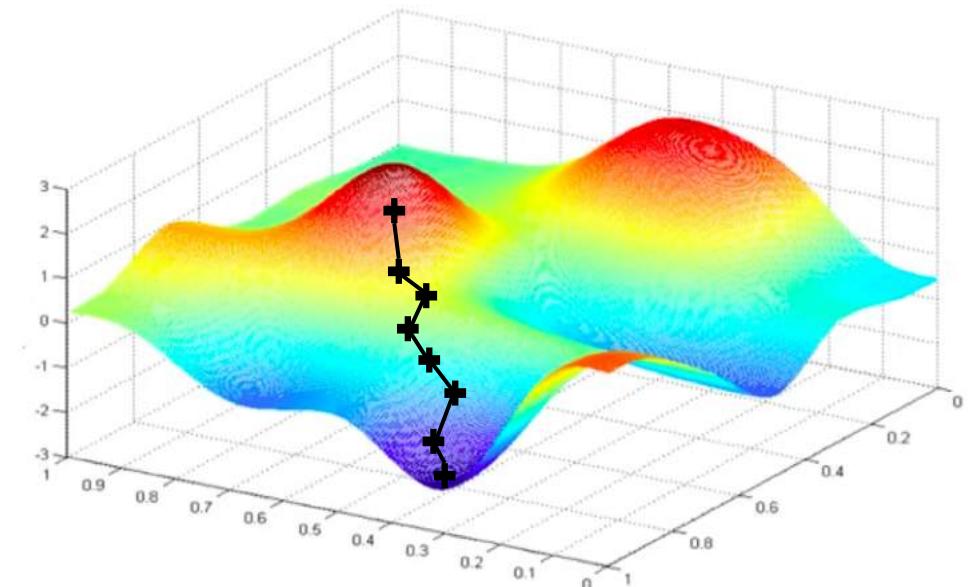
1. Initialize weights randomly  $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Pick single data point  $i$
4. Compute gradient,  $\frac{\partial J_i(\mathbf{W})}{\partial \mathbf{W}}$
5. Update weights,  $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$
6. Return weights



# Stochastic Gradient Descent

## Algorithm

1. Initialize weights randomly  $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Pick single data point  $i$
4. Compute gradient,  $\frac{\partial J_i(\mathbf{W})}{\partial \mathbf{W}}$
5. Update weights,  $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$
6. Return weights

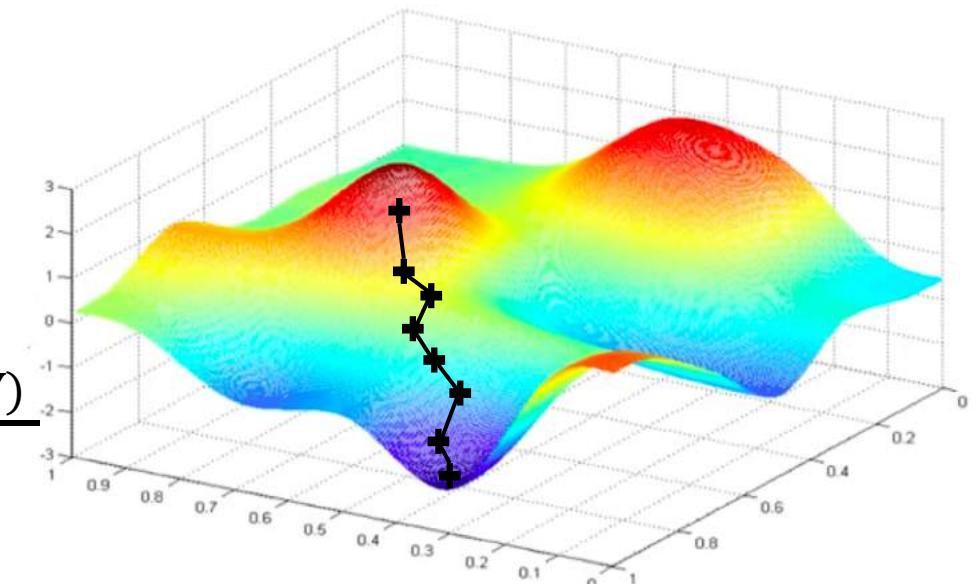


Easy to compute but  
**very noisy**  
(stochastic)!

# Stochastic Gradient Descent

## Algorithm

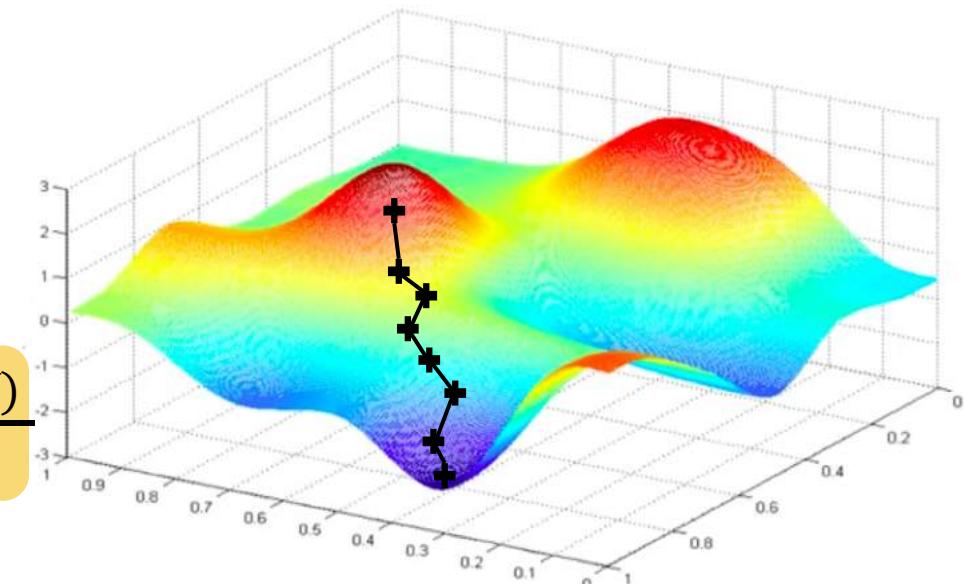
1. Initialize weights randomly  $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Pick batch of  $B$  data points
4. Compute gradient,  $\frac{\partial J(\mathbf{W})}{\partial \mathbf{W}} = \frac{1}{B} \sum_{k=1}^B \frac{\partial J_k(\mathbf{W})}{\partial \mathbf{W}}$
5. Update weights,  $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$
6. Return weights



# Stochastic Gradient Descent

## Algorithm

1. Initialize weights randomly  $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Pick batch of  $B$  data points
4. Compute gradient, 
$$\frac{\partial J(\mathbf{W})}{\partial \mathbf{W}} = \frac{1}{B} \sum_{k=1}^B \frac{\partial J_k(\mathbf{W})}{\partial \mathbf{W}}$$
5. Update weights,  $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$
6. Return weights



Fast to compute and a much better estimate of the true gradient!

# Mini-batches while training

## More accurate estimation of gradient

Smoother convergence

Allows for larger learning rates

# Mini-batches while training

**More accurate estimation of gradient**

Smoother convergence

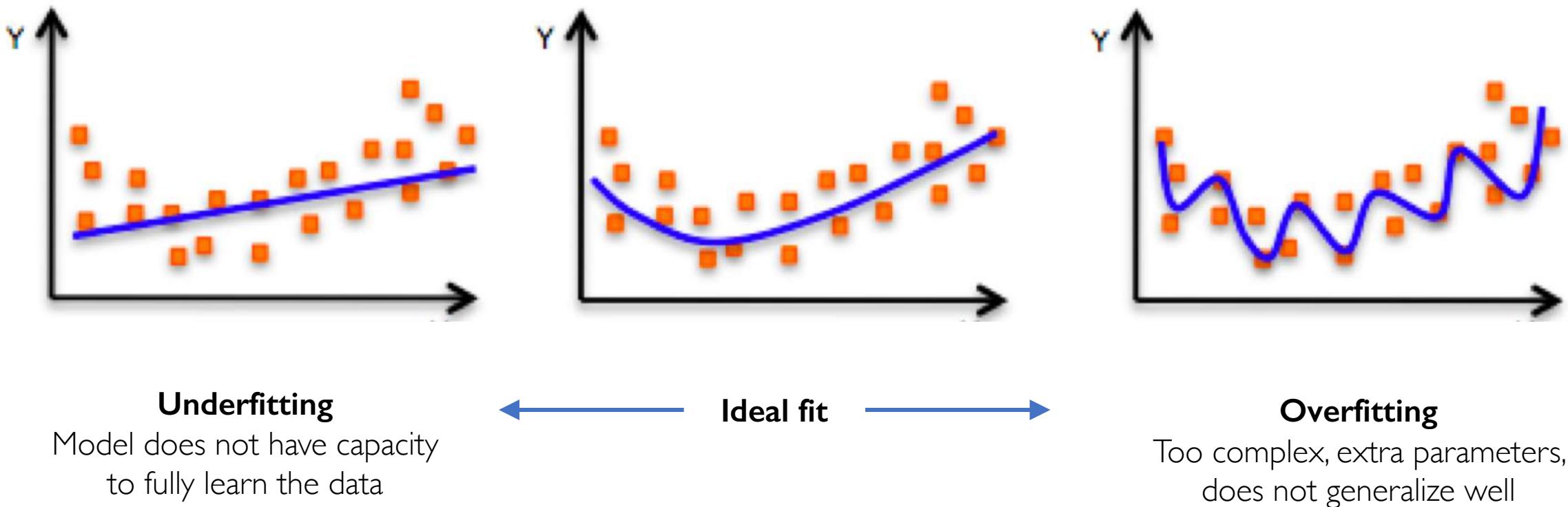
Allows for larger learning rates

**Mini-batches lead to fast training!**

Can parallelize computation + achieve significant speed increases on GPU's

# Neural Networks in Practice: Overfitting

# The Problem of Overfitting



# Regularization

## **What is it?**

*Technique that constrains our optimization problem to discourage complex models*

# Regularization

## *What is it?*

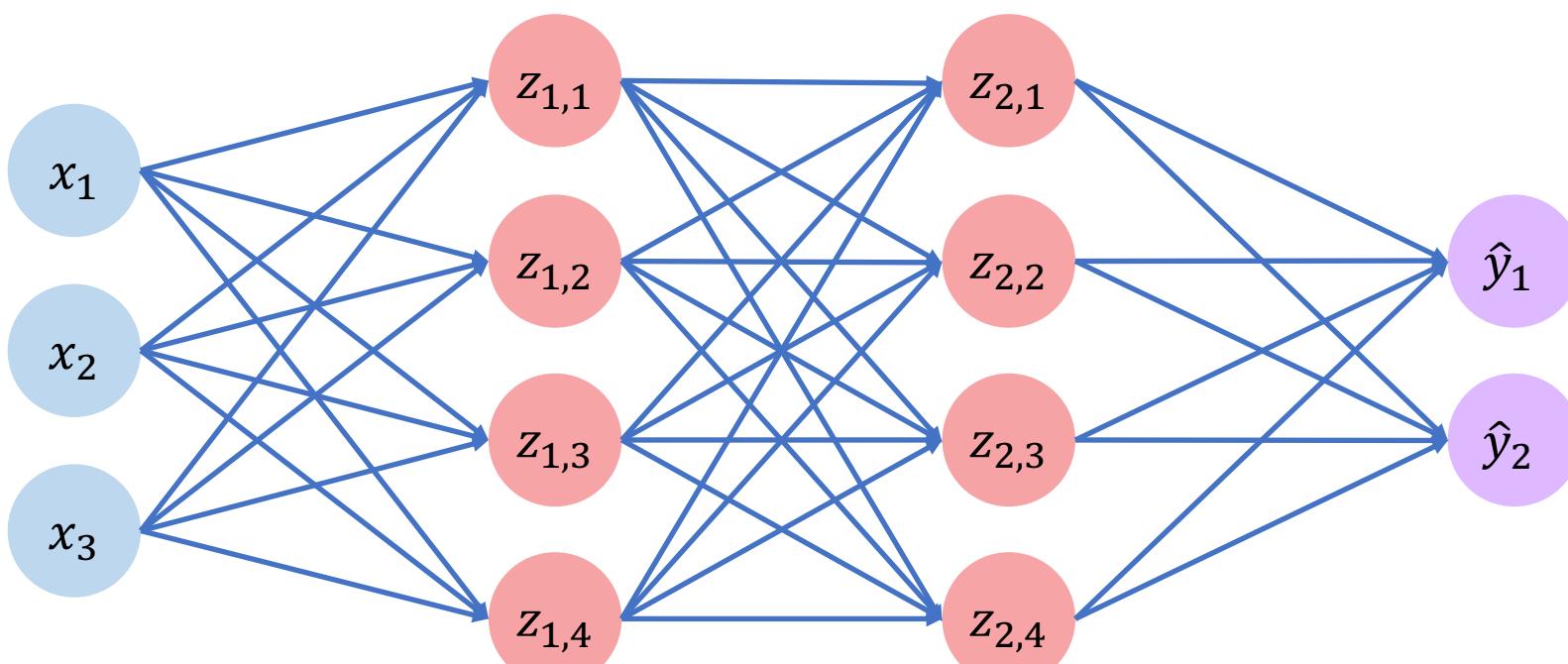
*Technique that constrains our optimization problem to discourage complex models*

## **Why do we need it?**

*Improve generalization of our model on unseen data*

# Regularization I: Dropout

- During training, randomly set some activations to 0

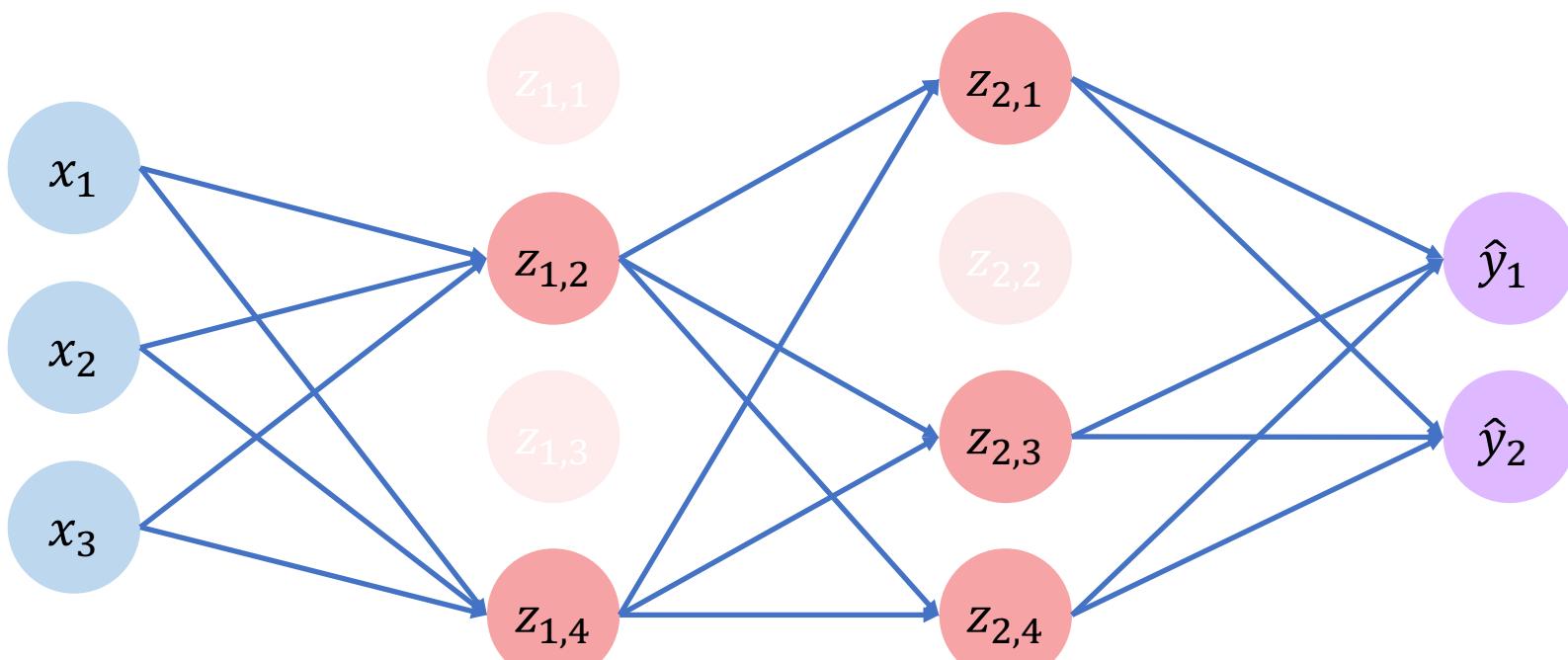


# Regularization I: Dropout

- During training, randomly set some activations to 0
  - Typically ‘drop’ 50% of activations in layer
  - Forces network to not rely on any 1 node



tf.keras.layers.Dropout (p=0.5)

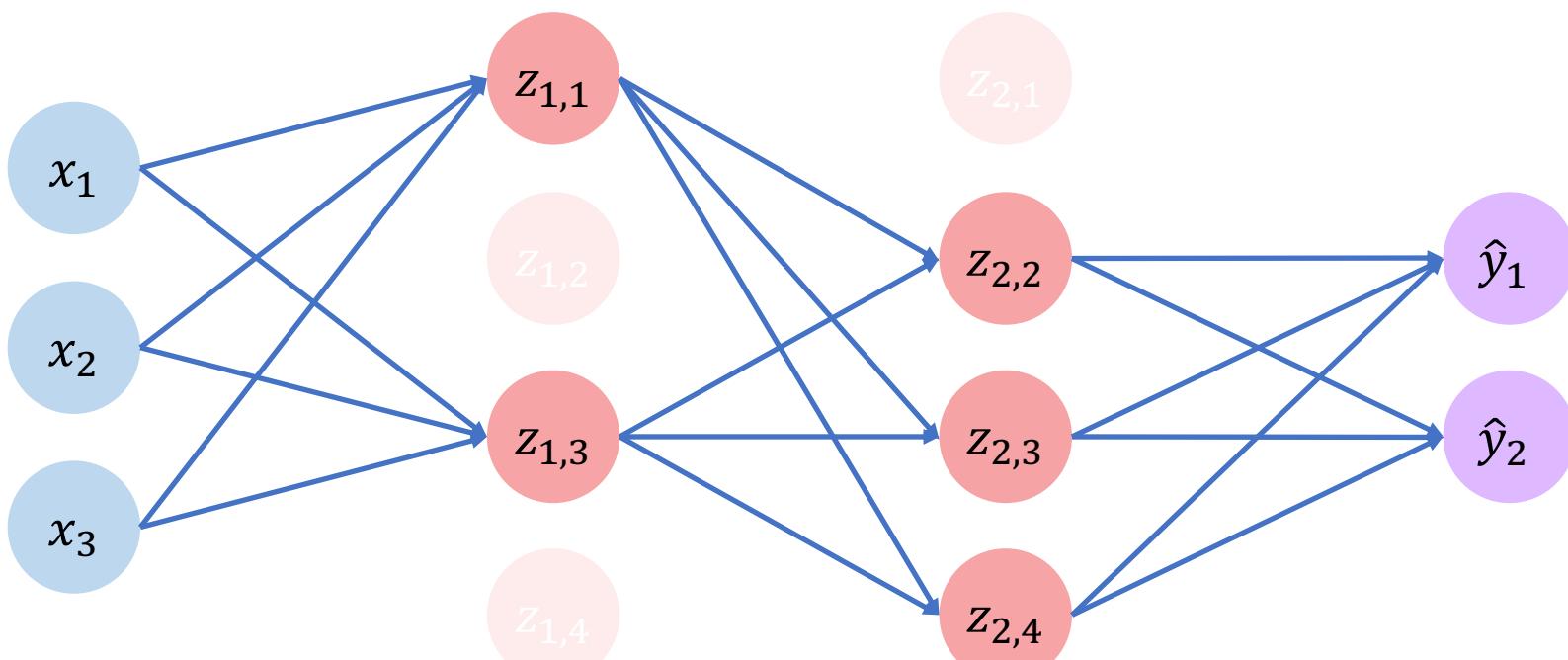


# Regularization I: Dropout

- During training, randomly set some activations to 0
  - Typically ‘drop’ 50% of activations in layer
  - Forces network to not rely on any 1 node



`tf.keras.layers.Dropout (p=0.5)`



# Regularization 2: Early Stopping

- Stop training before we have a chance to overfit



# Regularization 2: Early Stopping

- Stop training before we have a chance to overfit



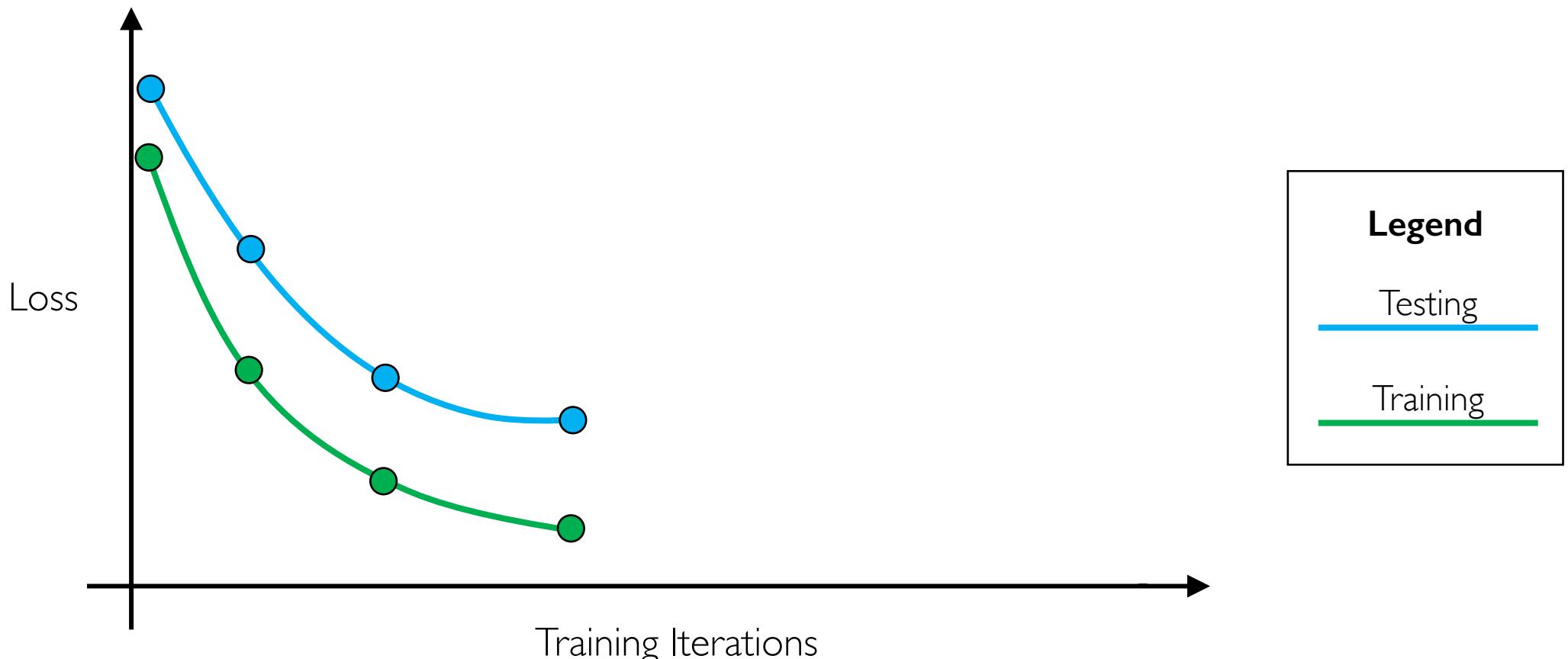
# Regularization 2: Early Stopping

- Stop training before we have a chance to overfit



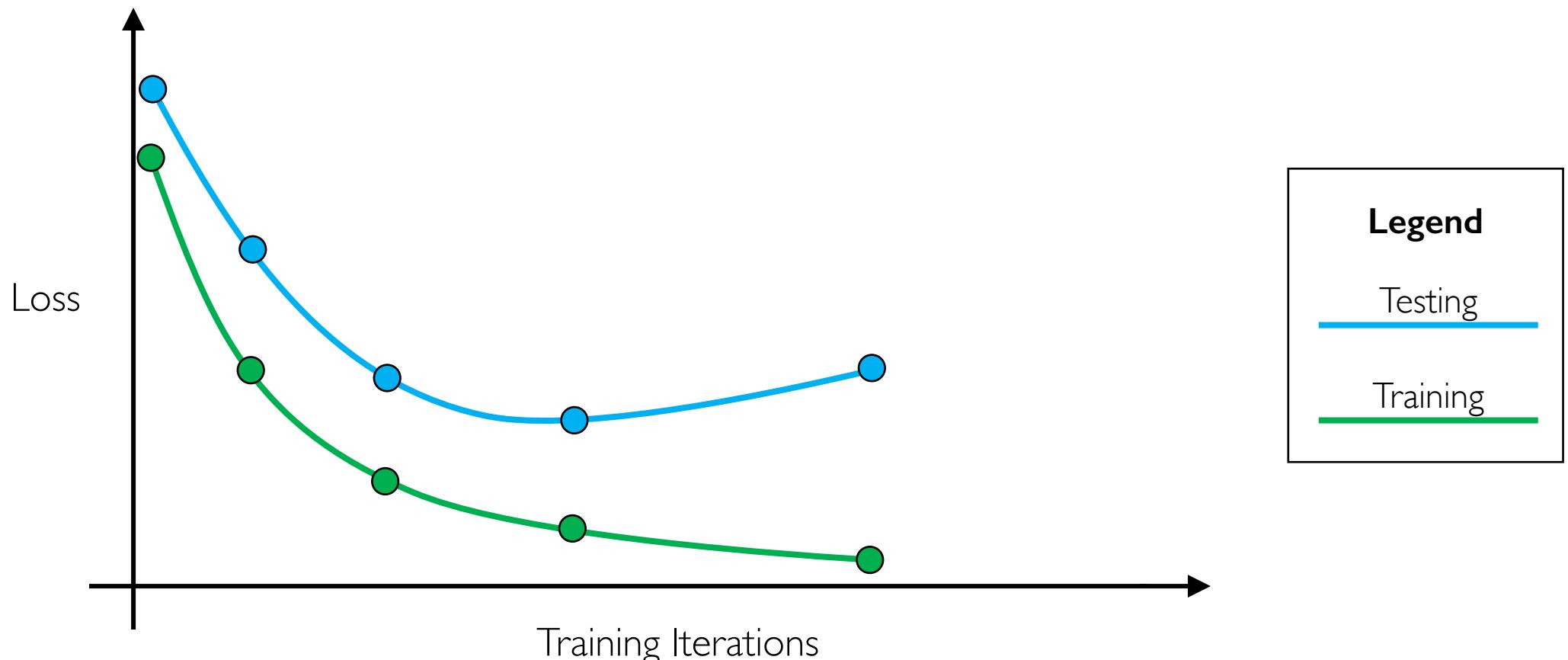
# Regularization 2: Early Stopping

- Stop training before we have a chance to overfit



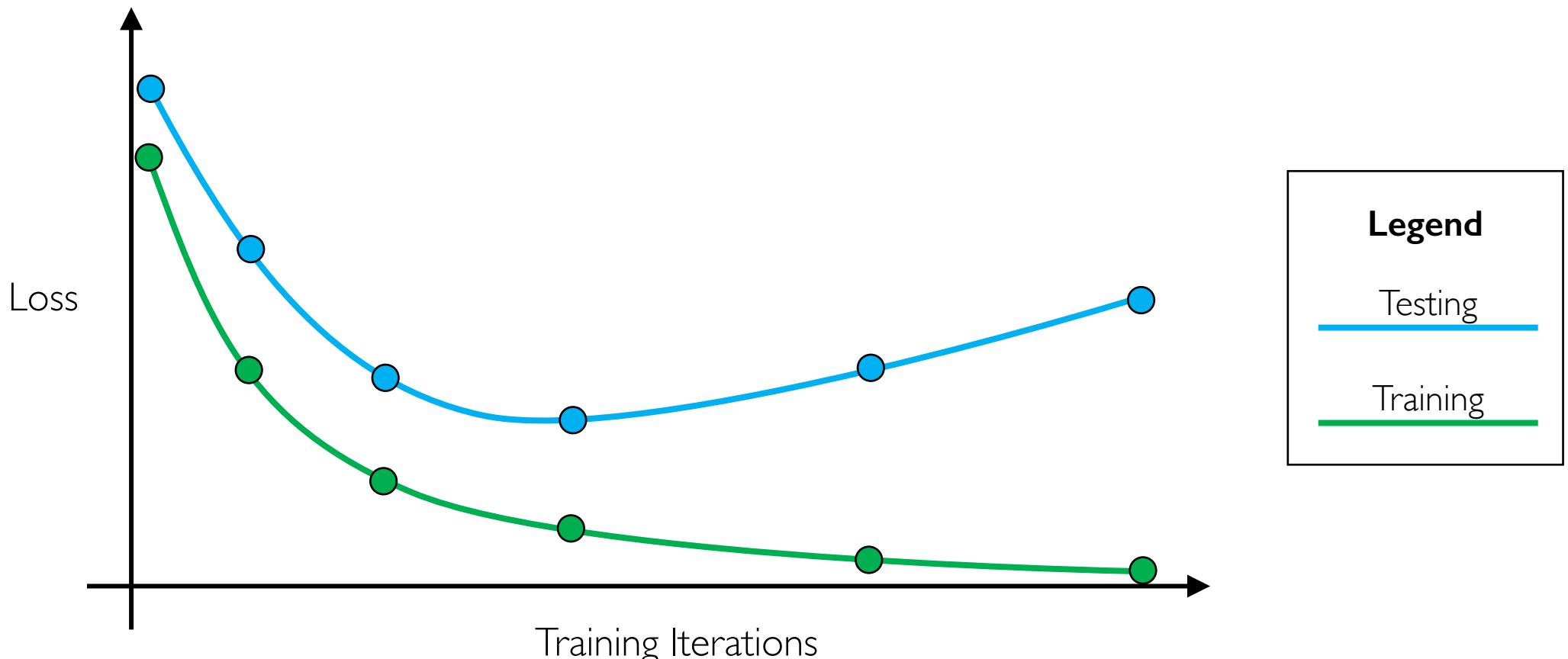
# Regularization 2: Early Stopping

- Stop training before we have a chance to overfit



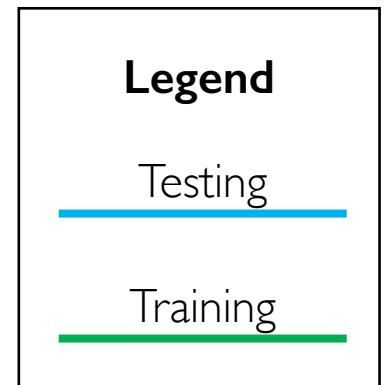
# Regularization 2: Early Stopping

- Stop training before we have a chance to overfit



# Regularization 2: Early Stopping

- Stop training before we have a chance to overfit



# Regularization 2: Early Stopping

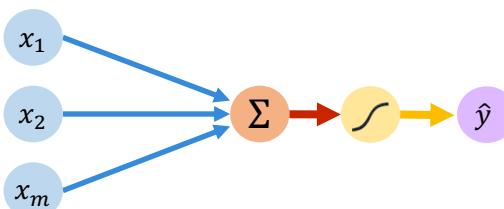
- Stop training before we have a chance to overfit



# Core Foundation Review

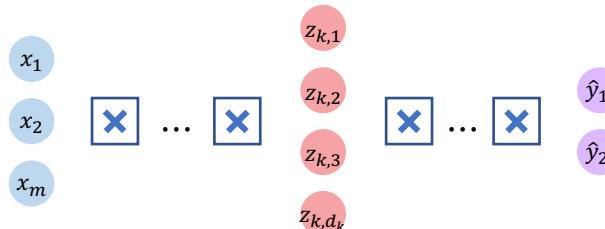
## The Perceptron

- Structural building blocks
- Nonlinear activation functions



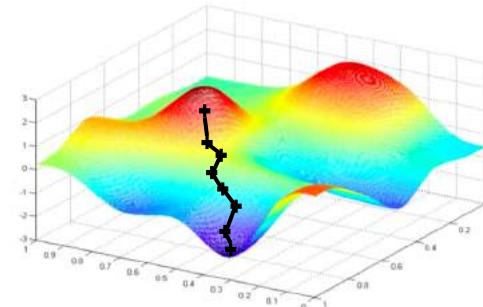
## Neural Networks

- Stacking Perceptrons to form neural networks
- Optimization through backpropagation



## Training in Practice

- Adaptive learning
- Batching
- Regularization



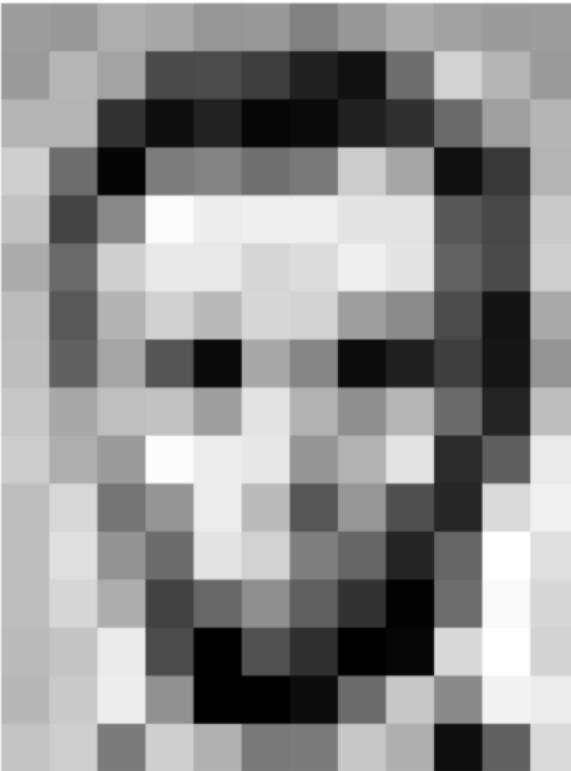


# What Computers “See”

# Images are Numbers

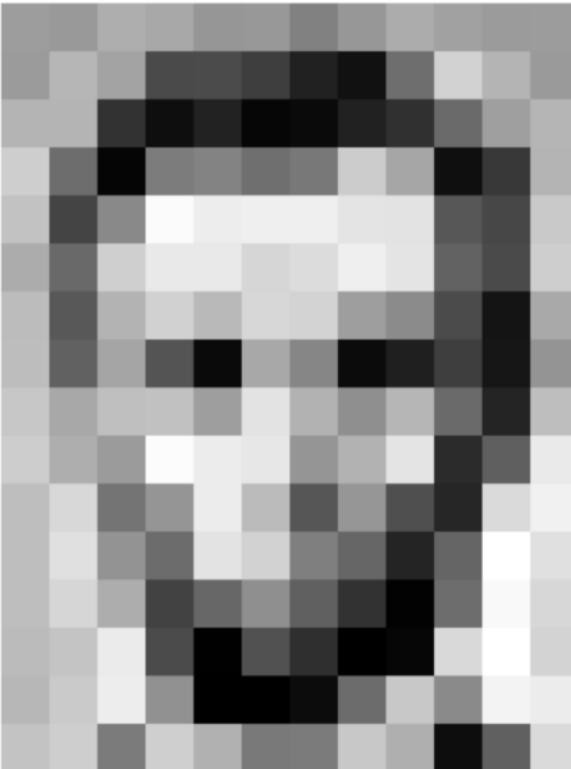


# Images are Numbers



157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	84	6	10	33	48	105	159	181
206	109	5	124	191	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	105	36	190
205	174	155	252	236	251	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	234	147	108	227	210	127	102	35	101	255	224
190	214	173	66	103	143	95	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

# Images are Numbers



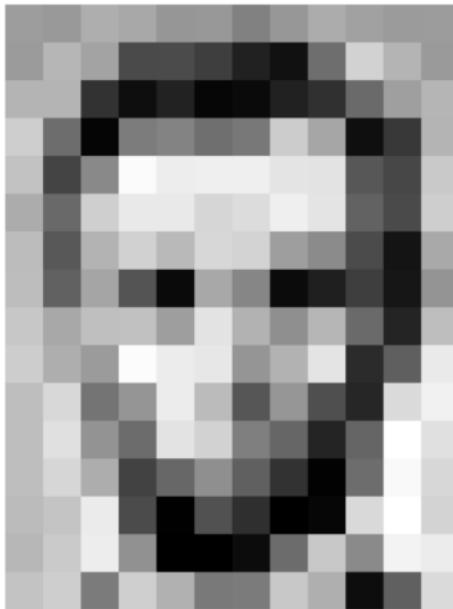
157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	84	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	105	36	190
205	174	155	252	236	251	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	35	101	255	224
190	214	173	66	103	143	95	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

What the computer sees

157	153	174	168	150	152	129	151	172	161	155	156
156	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	35	101	255	224
190	214	173	66	103	143	95	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

An image is just a matrix of numbers [0,255]!  
i.e., 1080x1080x3 for an RGB image

# Tasks in Computer Vision



Input Image



157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	106	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

Pixel Representation

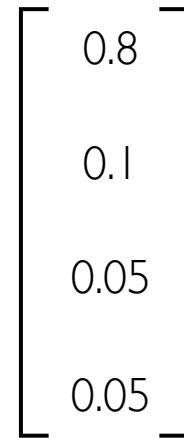
classification

Lincoln

Washington

Jefferson

Obama



- **Regression:** output variable takes continuous value
- **Classification:** output variable takes class label. Can produce probability of belonging to a particular class

# High Level Feature Detection

Let's identify key features in each image category



Nose,  
Eyes,  
Mouth



Wheels,  
License Plate,  
Headlights



Door,  
Windows,  
Steps

# Manual Feature Extraction

Domain knowledge

Define features

Detect features  
to classify

Problems?

# Manual Feature Extraction

Domain knowledge

Define features

Detect features  
to classify

Viewpoint variation



Scale variation



Deformation



Occlusion



Illumination conditions



Background clutter



Intra-class variation



# Manual Feature Extraction

Domain knowledge

Define features

Detect features  
to classify

Viewpoint variation



Illumination conditions



Scale variation



Deformation



Background clutter



Occlusion



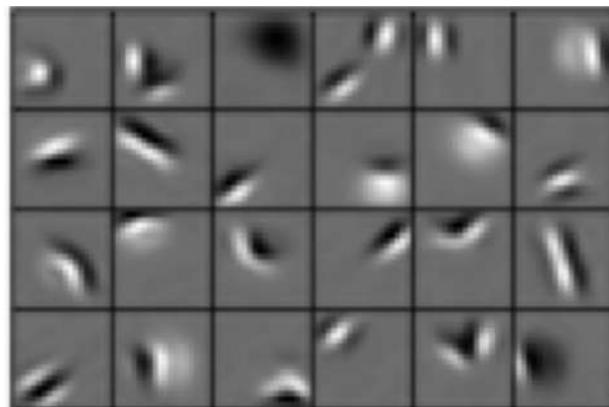
Intra-class variation



# Learning Feature Representations

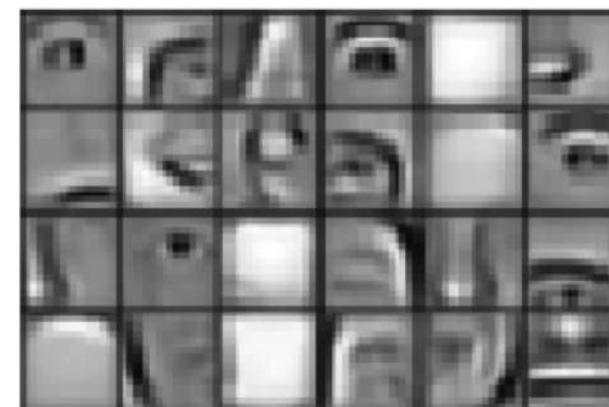
Can we learn a **hierarchy of features** directly from the data instead of hand engineering?

Low level features



Edges, dark spots

Mid level features



Eyes, ears, nose

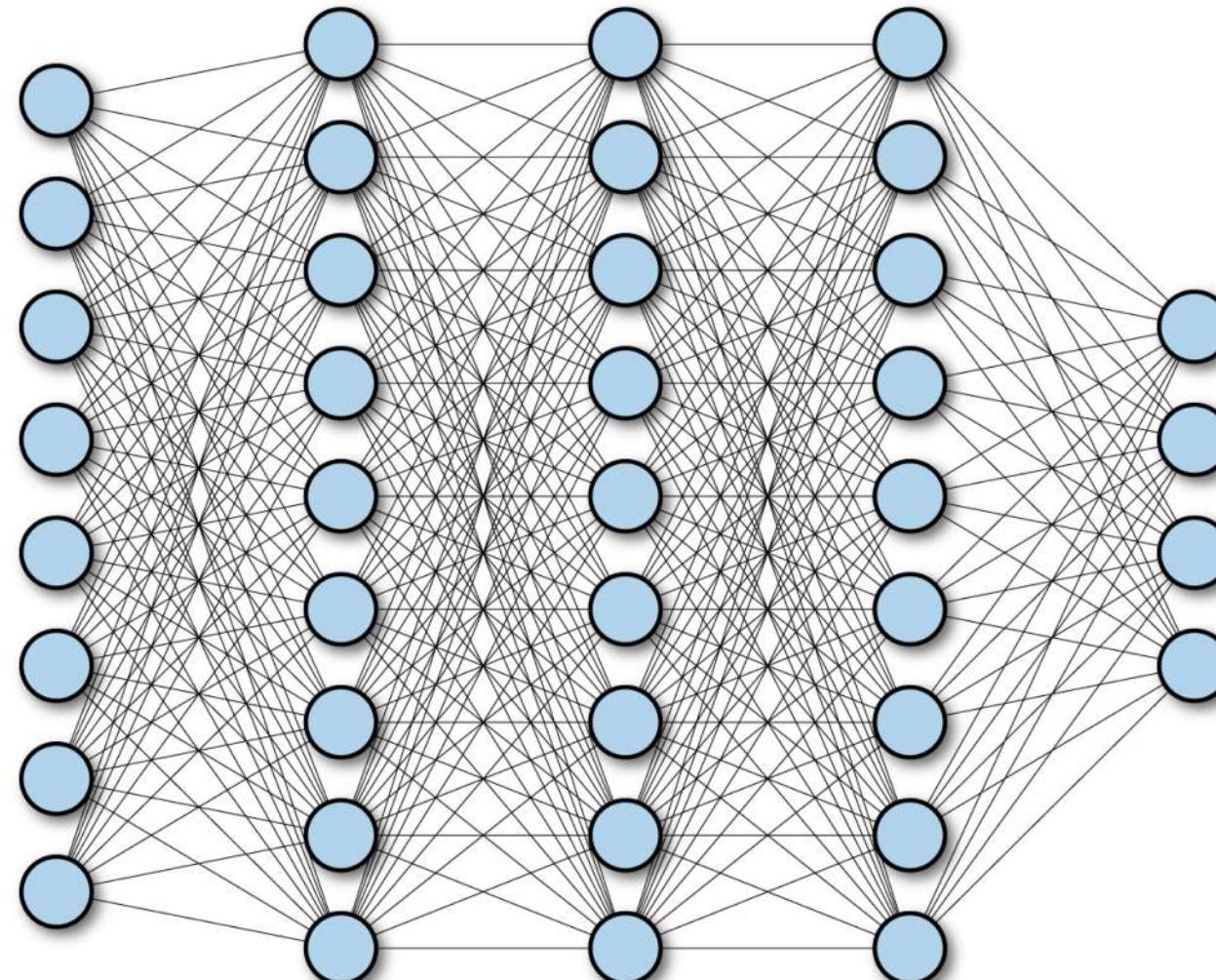
High level features



Facial structure

# Learning Visual Features

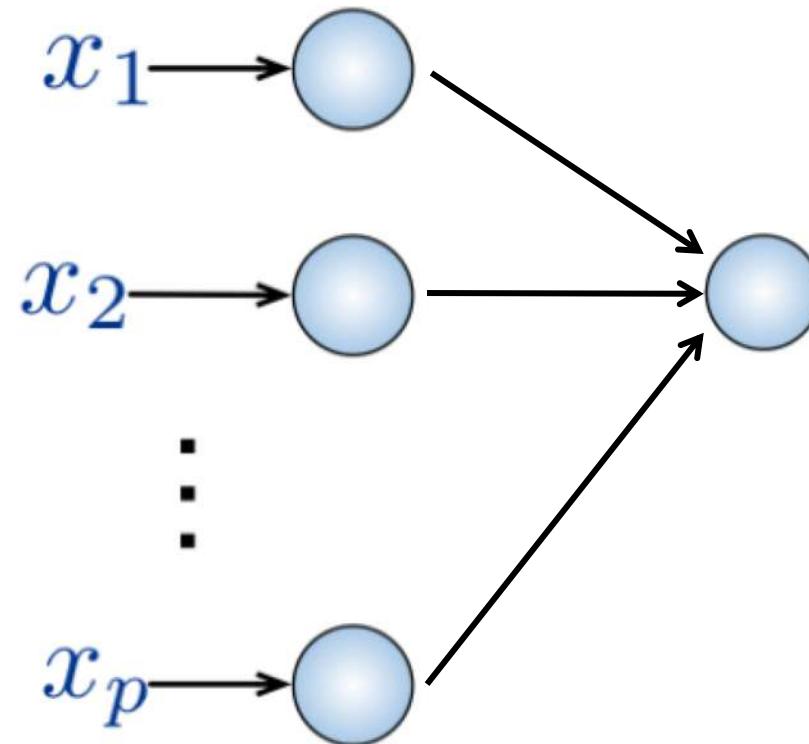
# Fully Connected Neural Network



# Fully Connected Neural Network

## Input:

- 2D image
- Vector of pixel values



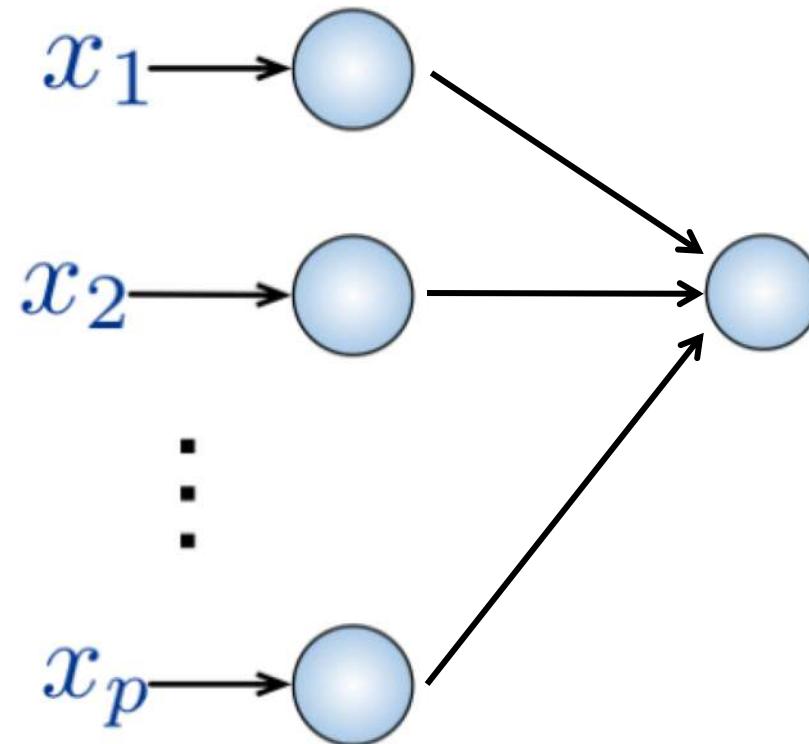
## Fully Connected:

- Connect neuron in hidden layer to all neurons in input layer
- No spatial information!
- And many, many parameters!

# Fully Connected Neural Network

## Input:

- 2D image
- Vector of pixel values



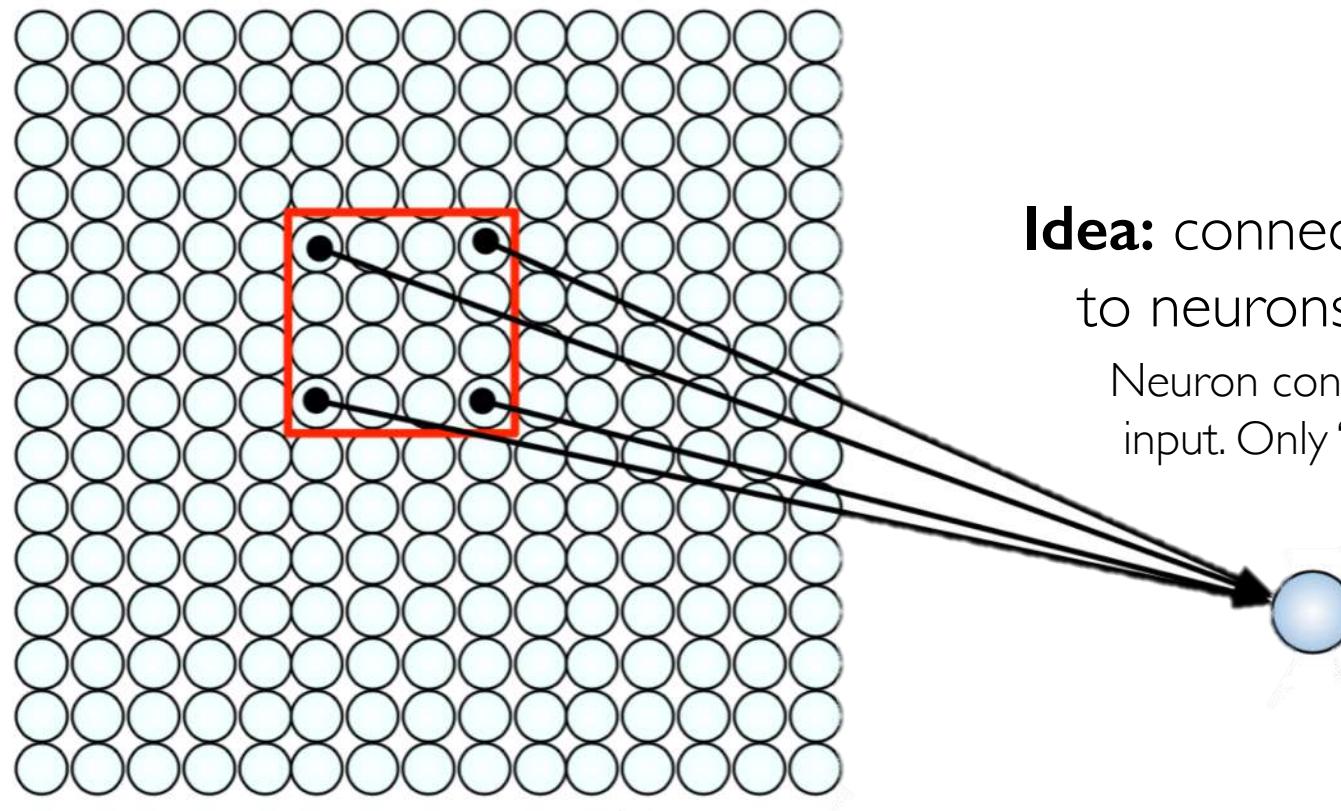
## Fully Connected:

- Connect neuron in hidden layer to all neurons in input layer
- No spatial information!
- And many, many parameters!

How can we use **spatial structure** in the input to inform the architecture of the network?

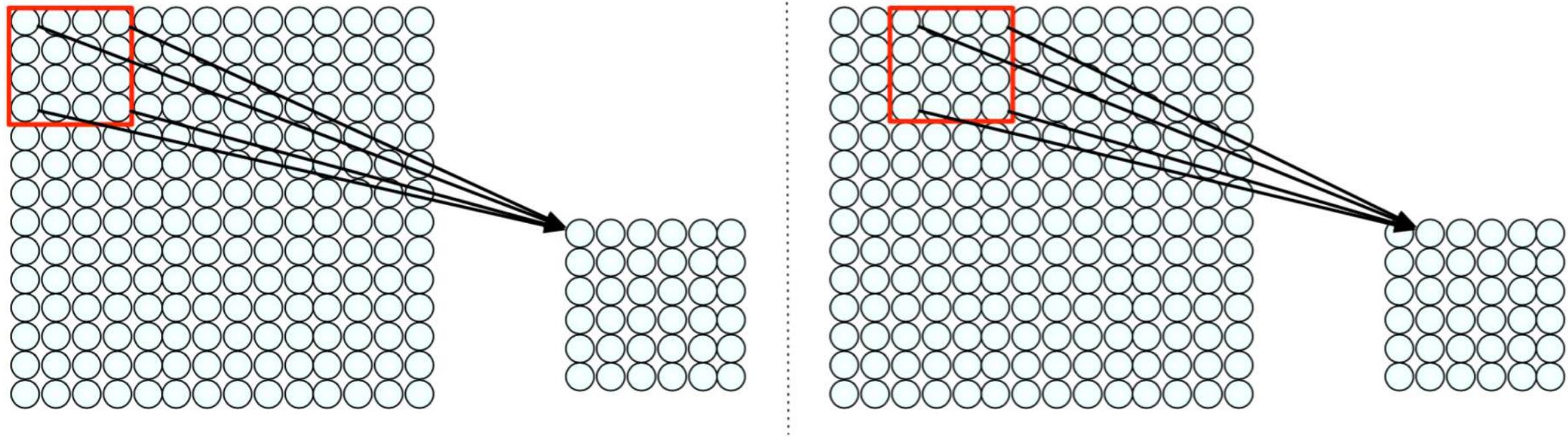
# Using Spatial Structure

**Input:** 2D image.  
Array of pixel values



**Idea:** connect patches of input  
to neurons in hidden layer.  
Neuron connected to region of  
input. Only “sees” these values.

# Using Spatial Structure



Connect patch in input layer to a single neuron in subsequent layer.

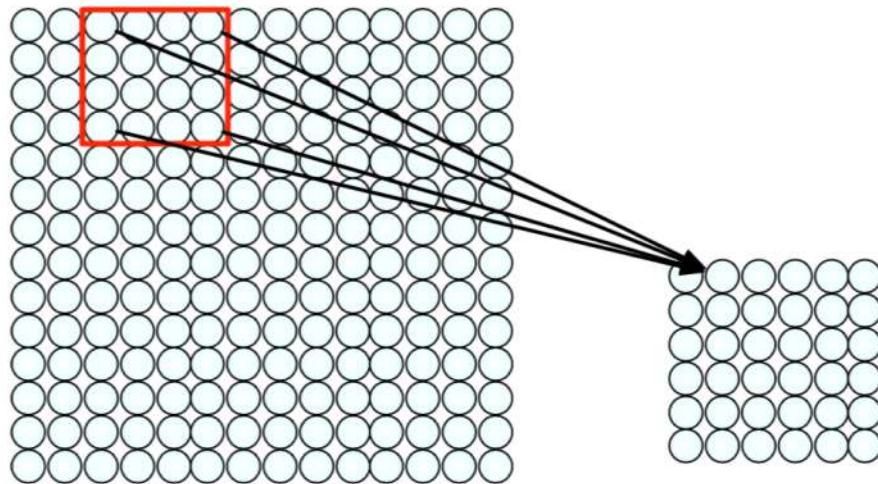
Use a sliding window to define connections.

*How can we **weight** the patch to detect particular features?*

# Applying Filters to Extract Features

- I) Apply a set of weights – a filter – to extract **local features**
- 2) Use **multiple filters** to extract different features
- 3) Spatially **share** parameters of each filter  
(features that matter in one part of the input should matter elsewhere)

# Feature Extraction with Convolution



- Filter of size  $4 \times 4$  : 16 different weights
- Apply this same filter to  $4 \times 4$  patches in input
- Shift by 2 pixels for next patch

This “patchy” operation is **convolution**

- 1) Apply a set of weights – a filter – to extract **local features**
- 2) Use **multiple filters** to extract different features
- 3) **Spatially share** parameters of each filter

# Feature Extraction and Convolution

## A Case Study

# X or X?

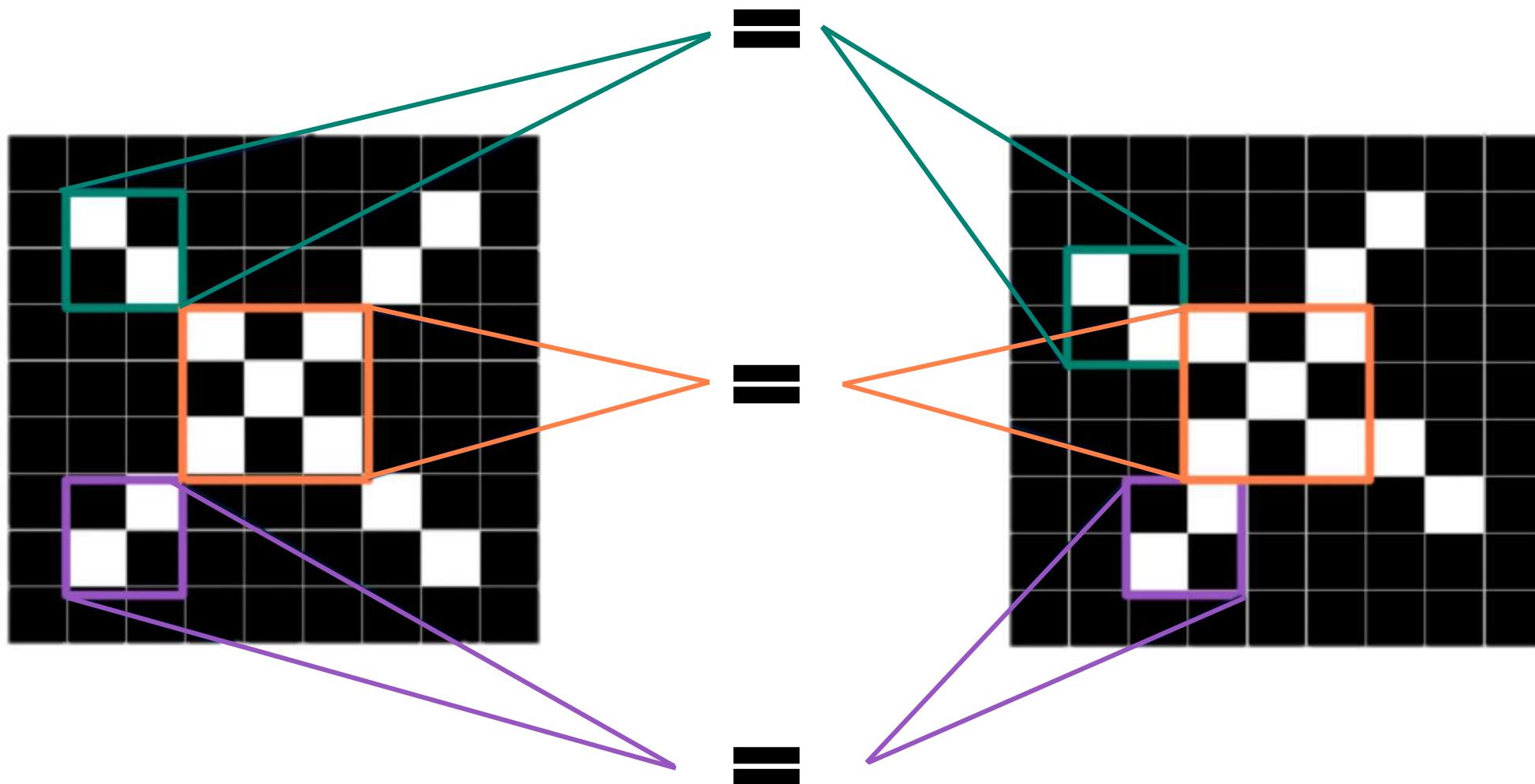


-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	1	-1
-1	1	-1	-1	-1	-1	1	-1	-1
-1	-1	1	1	-1	-1	1	-1	-1
-1	-1	-1	-1	1	1	-1	1	-1
-1	-1	-1	-1	-1	1	-1	-1	-1
-1	-1	-1	1	-1	-1	1	1	-1
-1	-1	-1	1	-1	-1	-1	-1	1
-1	-1	-1	-1	-1	-1	-1	-1	-1

Image is represented as matrix of pixel values... and computers are literal!  
We want to be able to classify an X as an X even if it's shifted, shrunk, rotated, deformed.

# Features of X



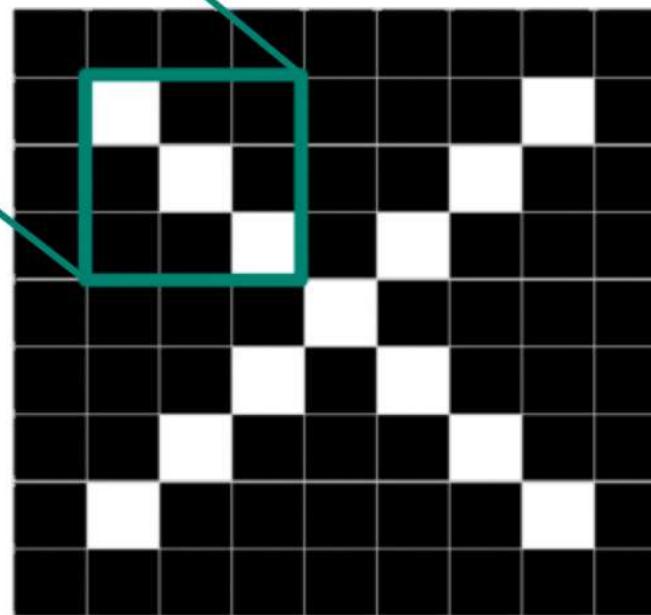
# Filters to Detect X Features

filters

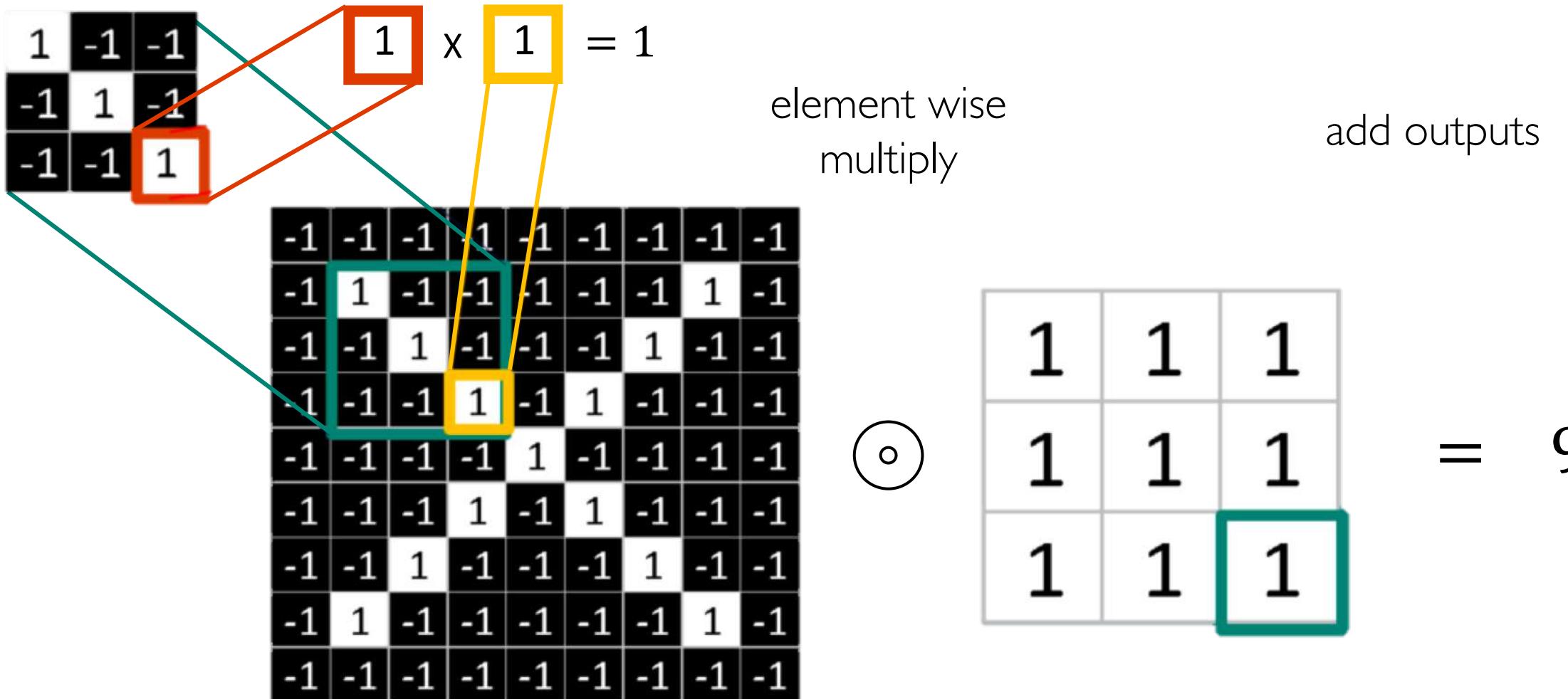
$$\begin{bmatrix} 1 & -1 & -1 \\ -1 & 1 & -1 \\ -1 & -1 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & -1 & 1 \\ -1 & 1 & -1 \\ 1 & -1 & 1 \end{bmatrix}$$

$$\begin{bmatrix} -1 & -1 & 1 \\ -1 & 1 & -1 \\ 1 & -1 & -1 \end{bmatrix}$$



# The Convolution Operation

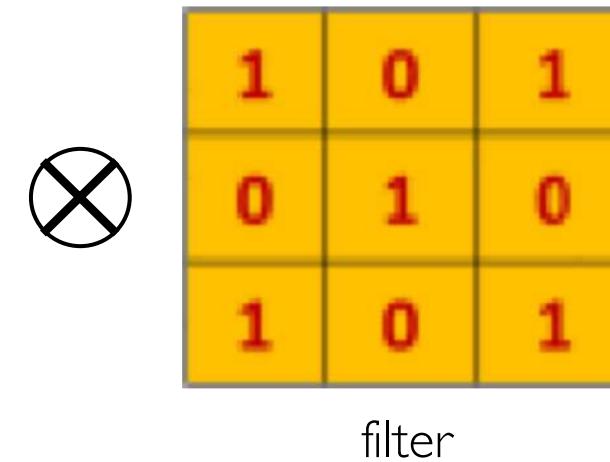


# The Convolution Operation

Suppose we want to compute the convolution of a 5x5 image and a 3x3 filter:

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

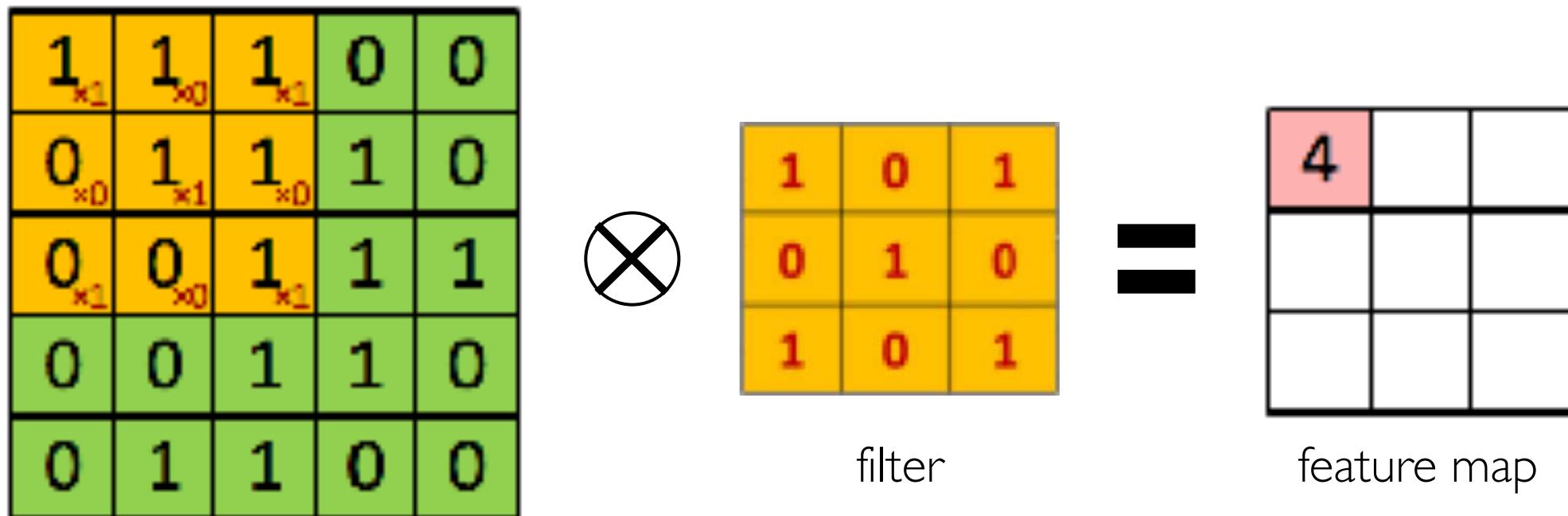
image



We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs...

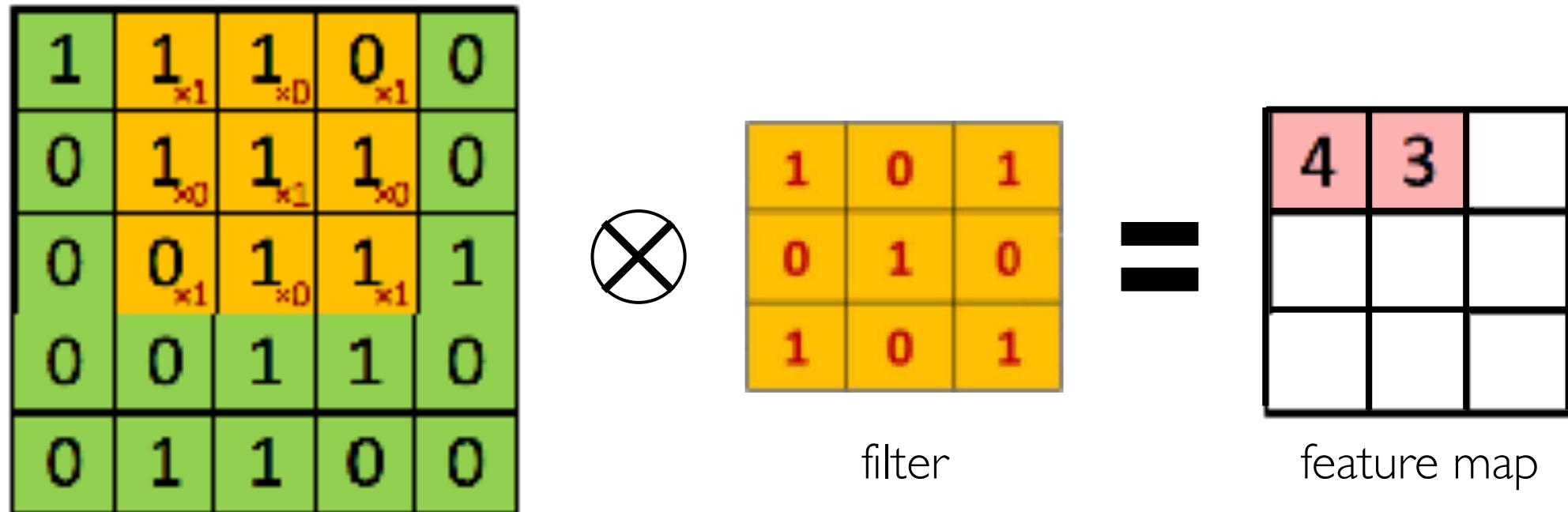
# The Convolution Operation

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



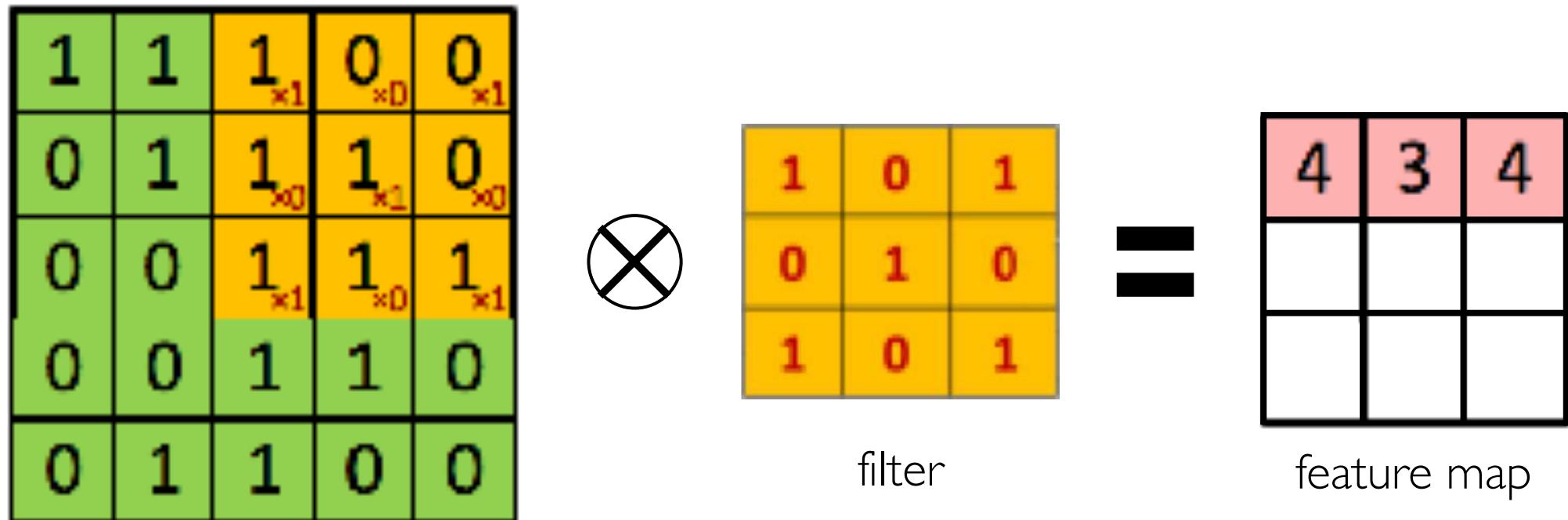
# The Convolution Operation

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



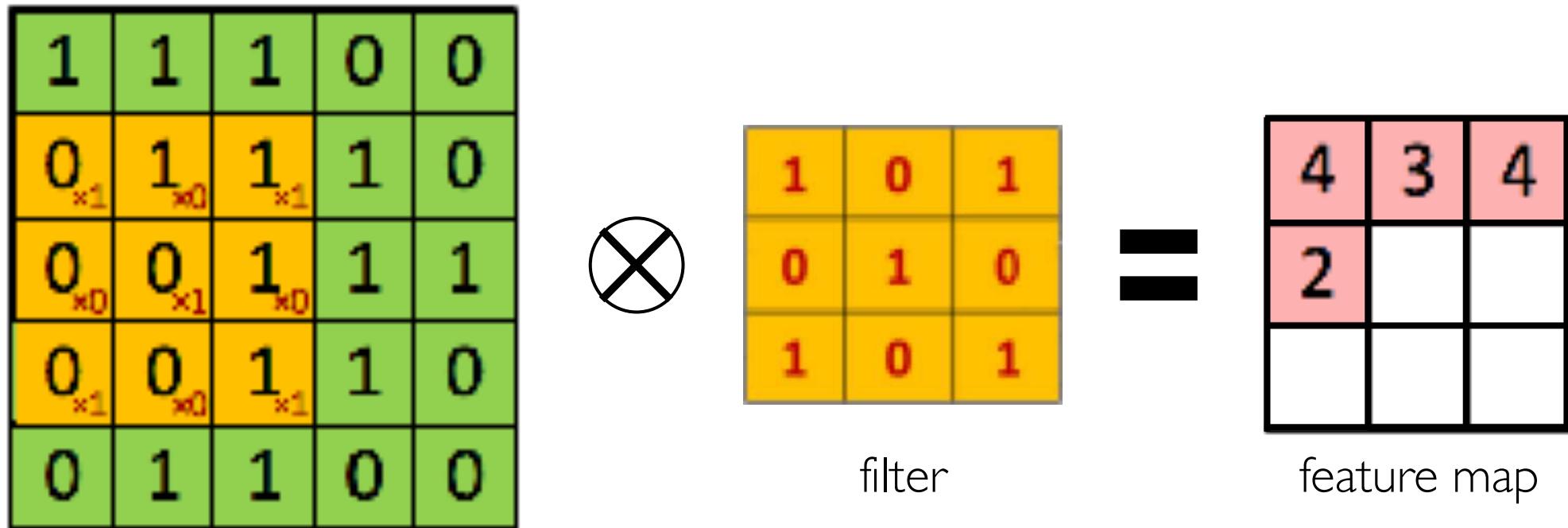
# The Convolution Operation

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



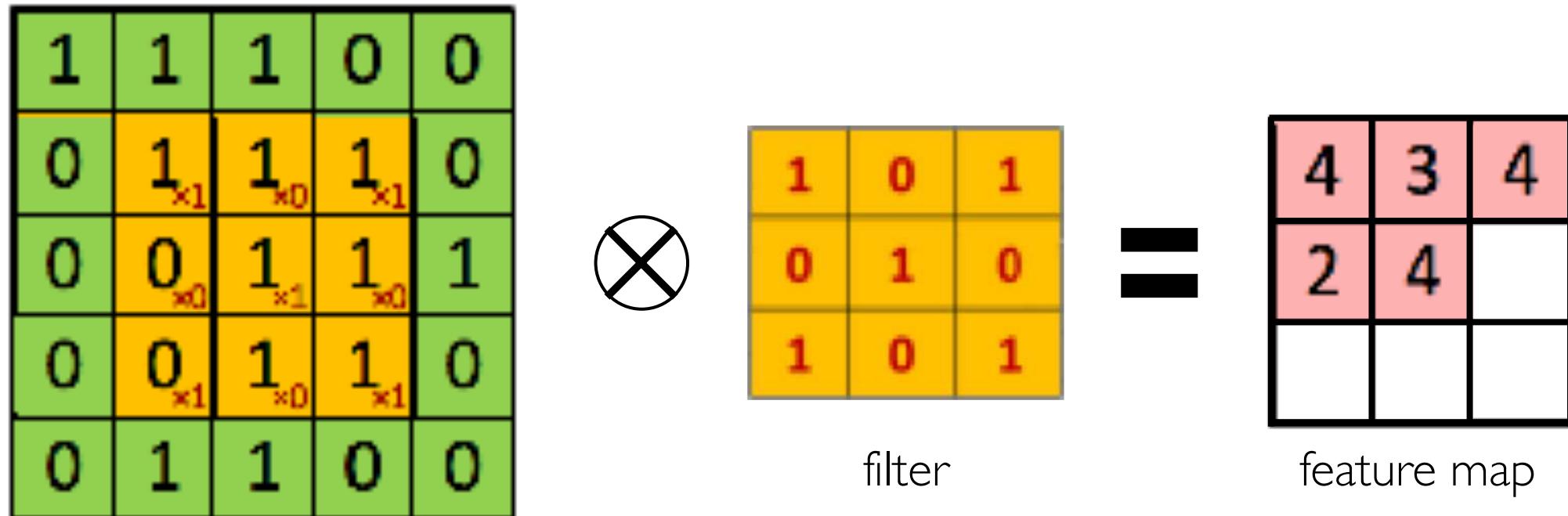
# The Convolution Operation

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



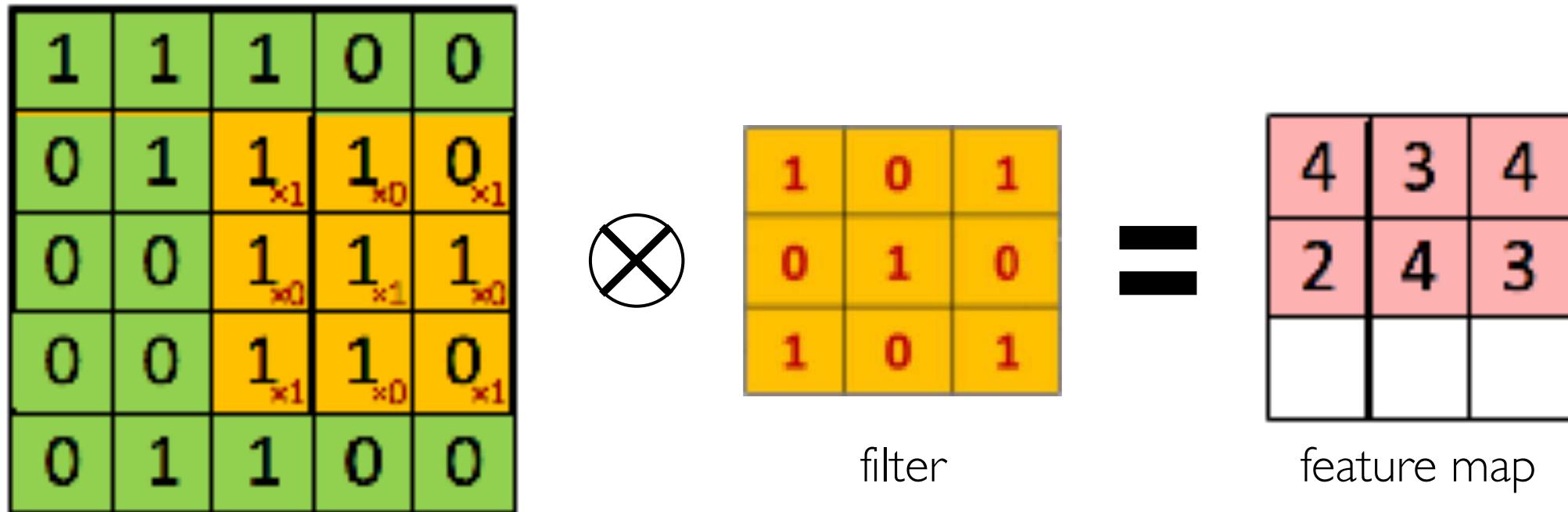
# The Convolution Operation

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



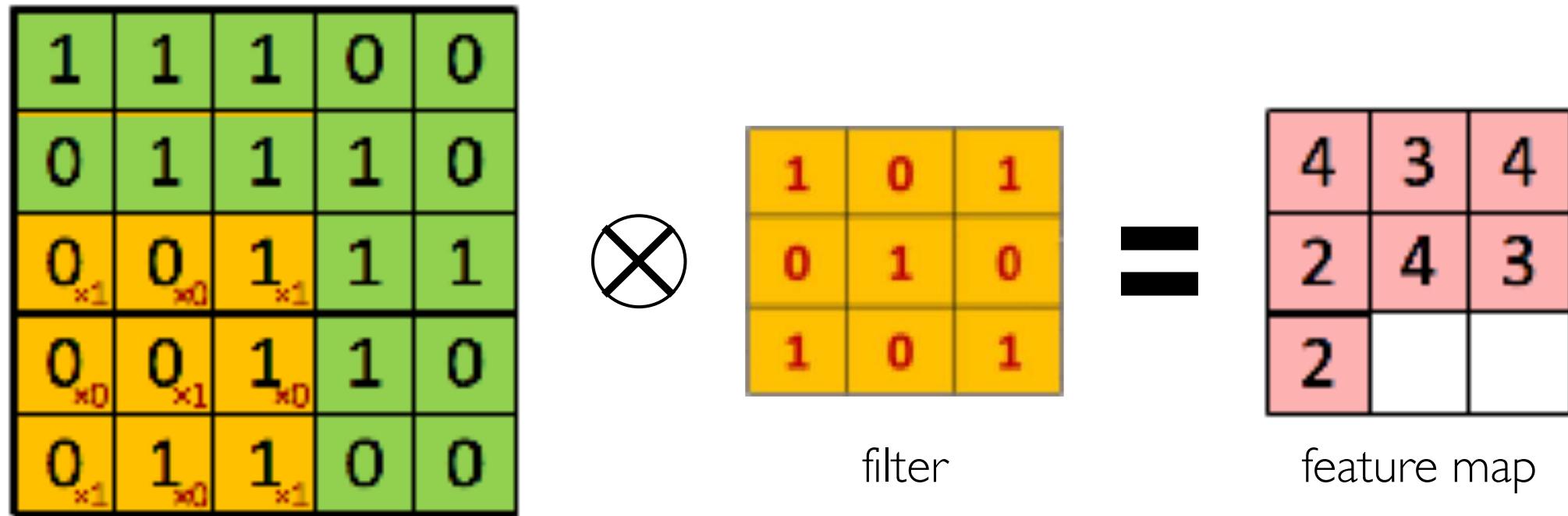
# The Convolution Operation

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



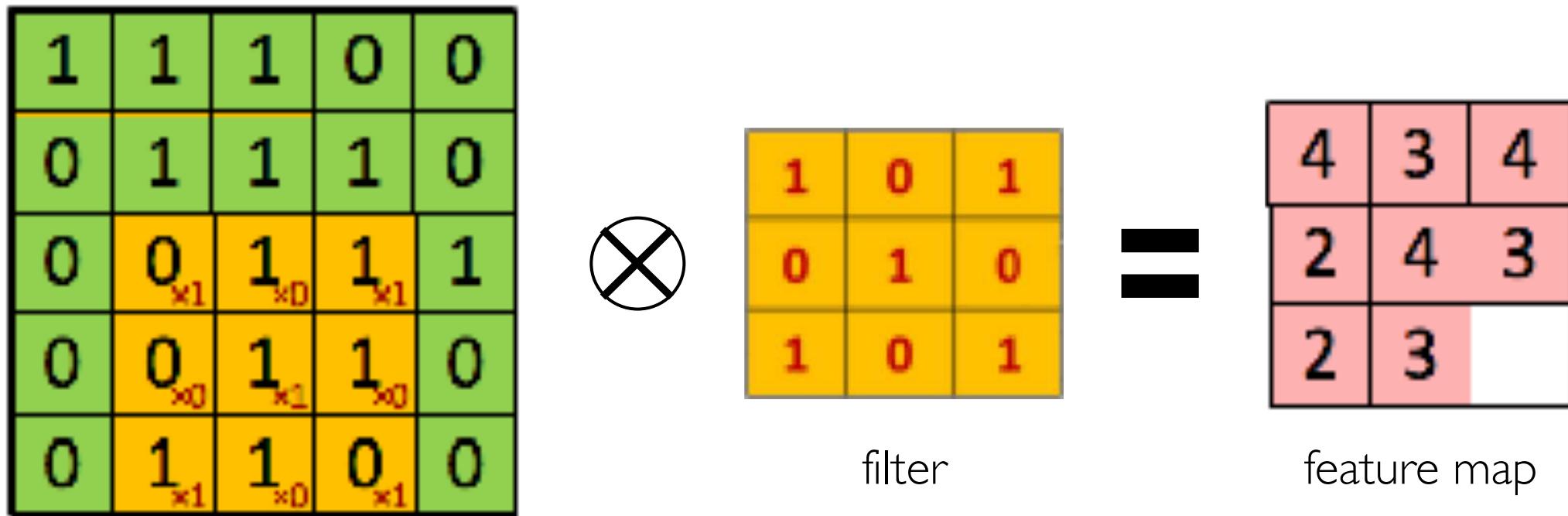
# The Convolution Operation

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



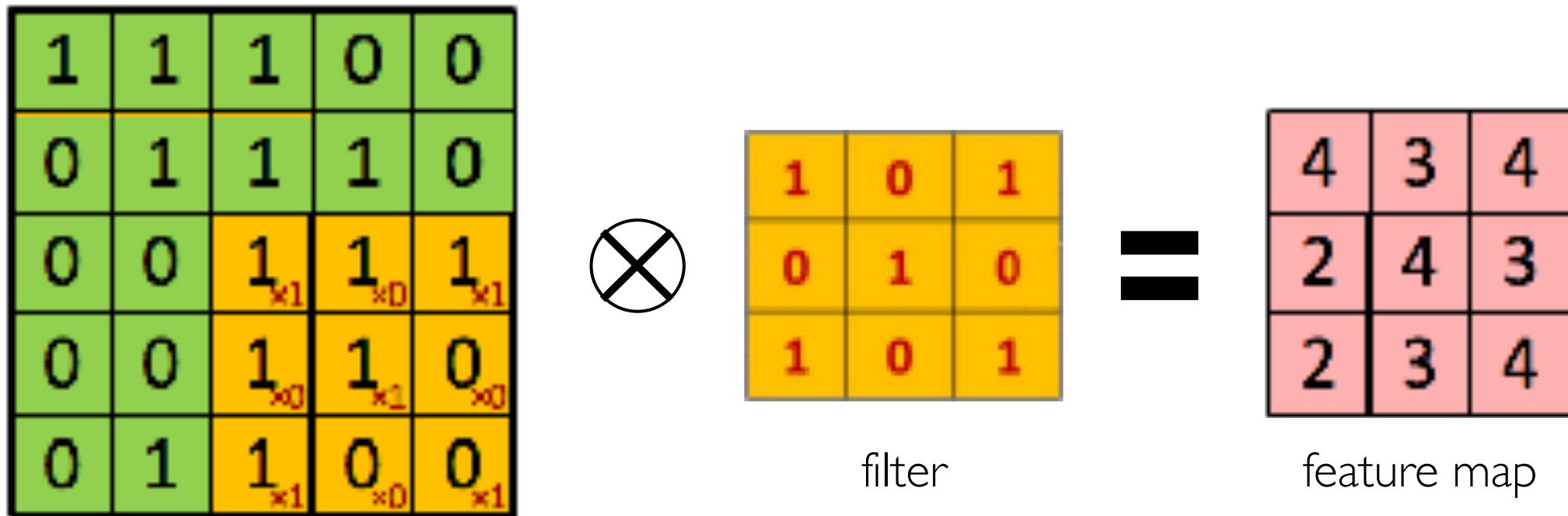
# The Convolution Operation

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



# The Convolution Operation

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



# Producing Feature Maps



Original



Sharpen

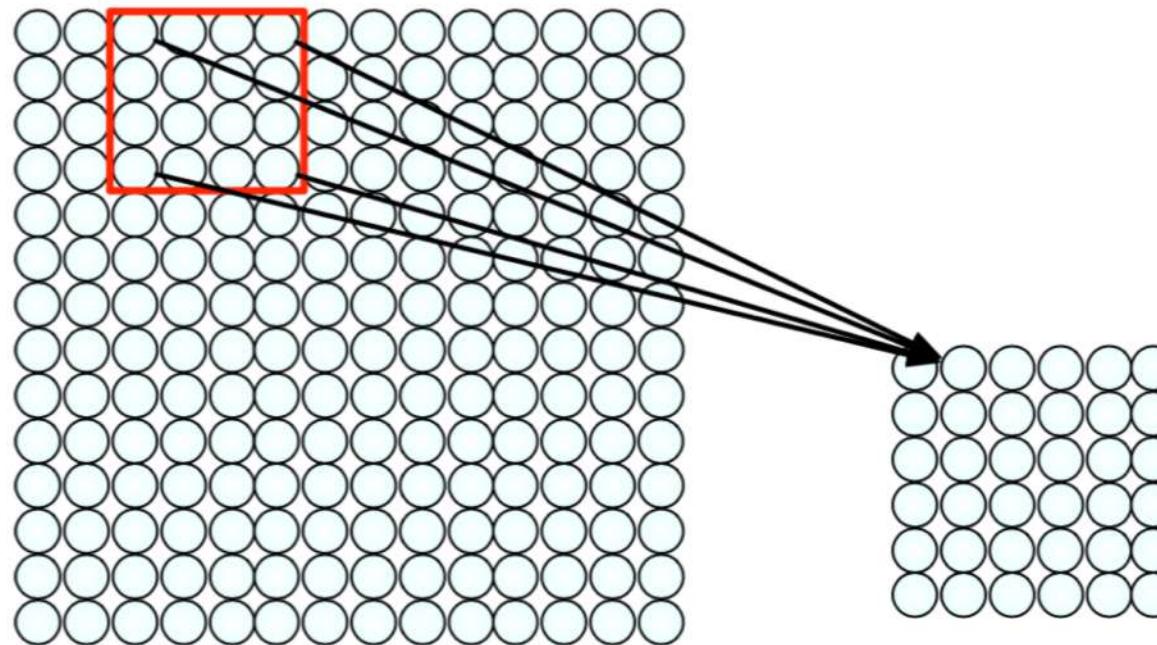


Edge Detect



“Strong” Edge  
Detect

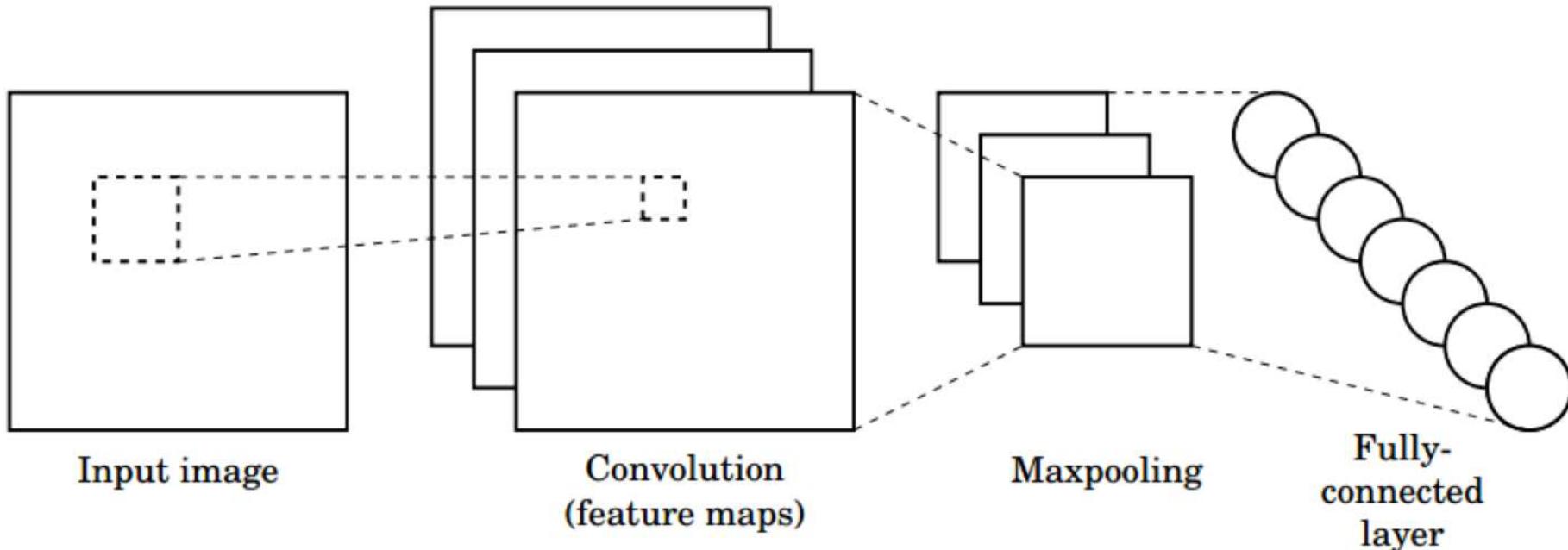
# Feature Extraction with Convolution



- 1) Apply a set of weights – a filter – to extract **local features**
- 2) Use **multiple filters** to extract different features
- 3) **Spatially share** parameters of each filter

# Convolutional Neural Networks (CNNs)

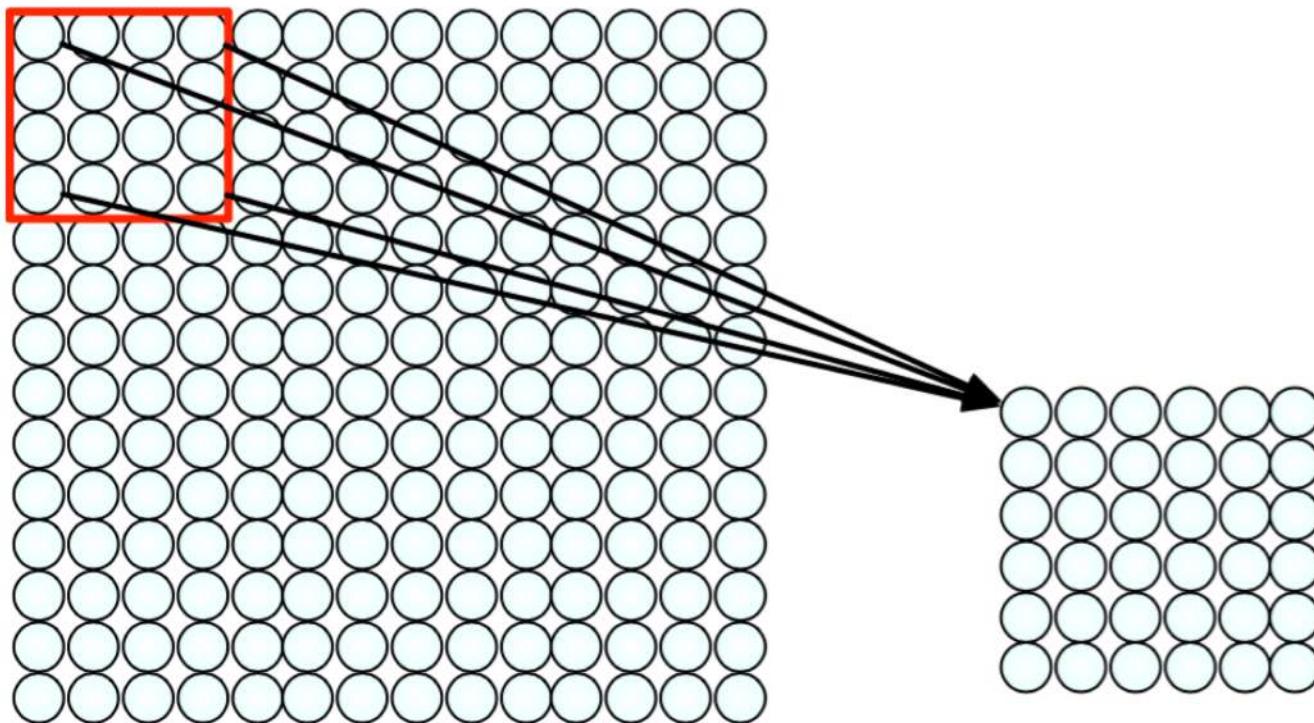
# CNNs for Classification



- 1. Convolution:** Apply filters with learned weights to generate feature maps.
- 2. Non-linearity:** Often ReLU.
- 3. Pooling:** Downsampling operation on each feature map.

**Train model with image data.**  
**Learn weights of filters in convolutional layers.**

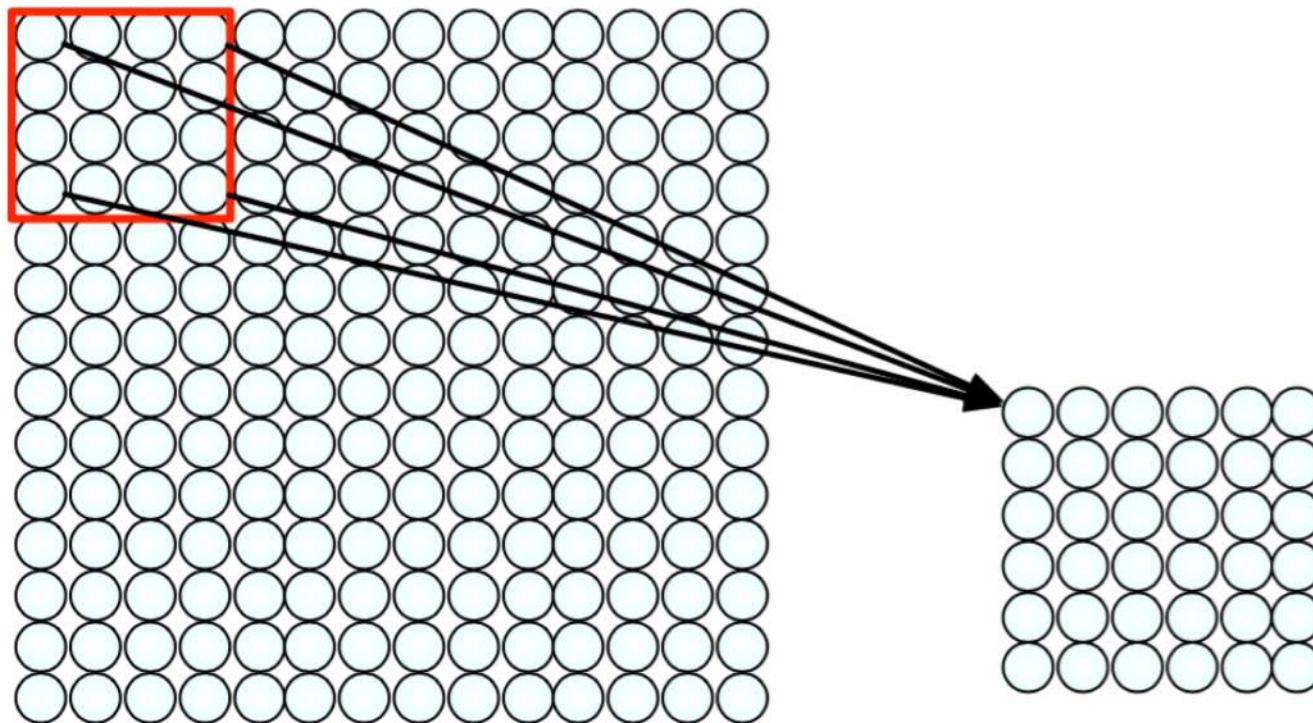
# Convolutional Layers: Local Connectivity



**For a neuron in hidden layer:**

- Take inputs from patch
- Compute weighted sum
- Apply bias

# Convolutional Layers: Local Connectivity



4x4 filter: matrix  
of weights  $w_{ij}$

$$\sum_{i=1}^4 \sum_{j=1}^4 w_{ij} x_{i+p,j+q} + b$$

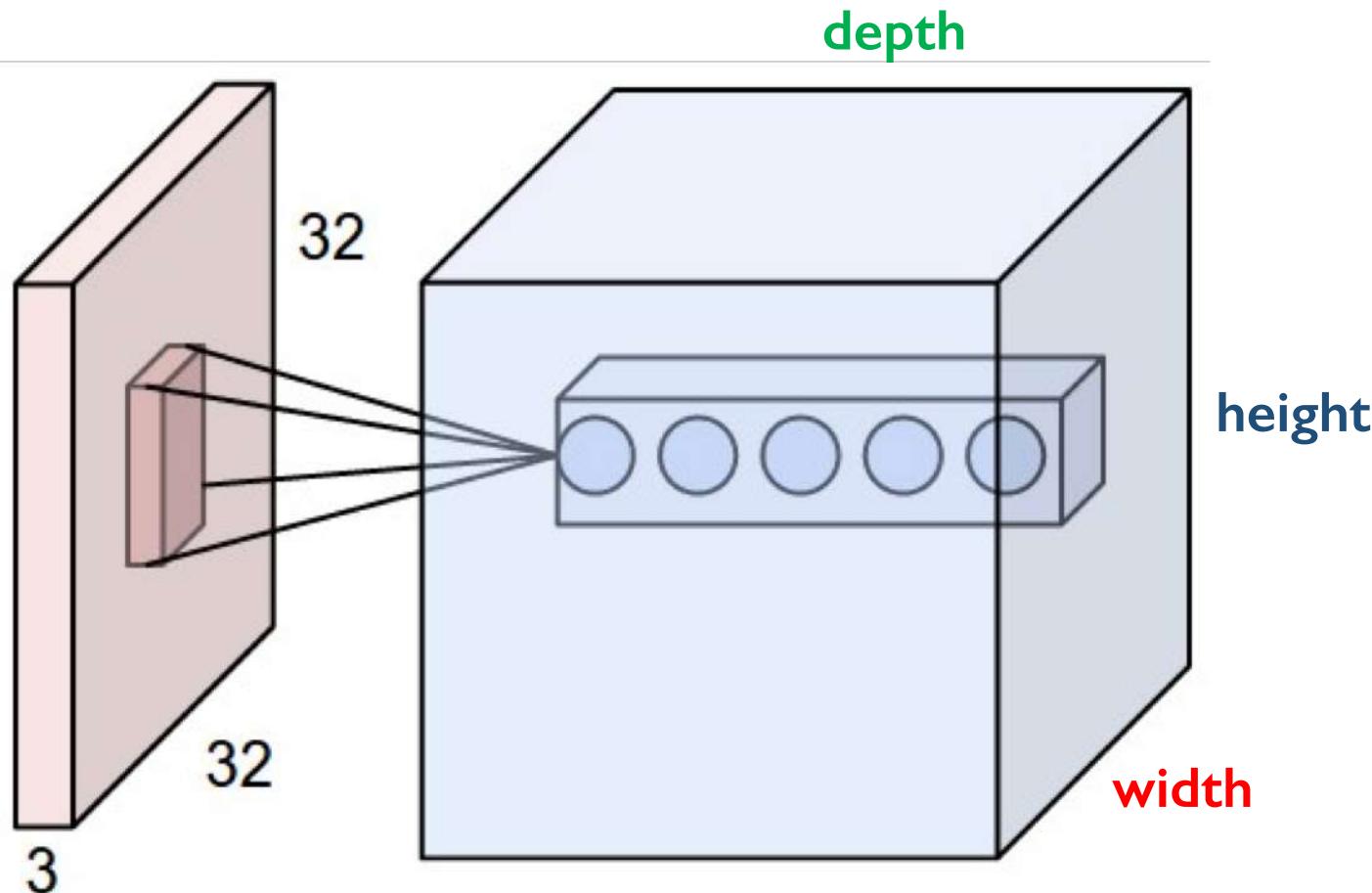
for neuron  $(p,q)$  in hidden layer

**For a neuron in hidden layer:**

- Take inputs from patch
- Compute weighted sum
- Apply bias

- 1) applying a window of weights
- 2) computing linear combinations
- 3) activating with non-linear function

# CNNs: Spatial Arrangement of Output Volume



**Layer Dimensions:**

$$h \times w \times d$$

where h and w are spatial dimensions  
d (depth) = number of filters

**Stride:**

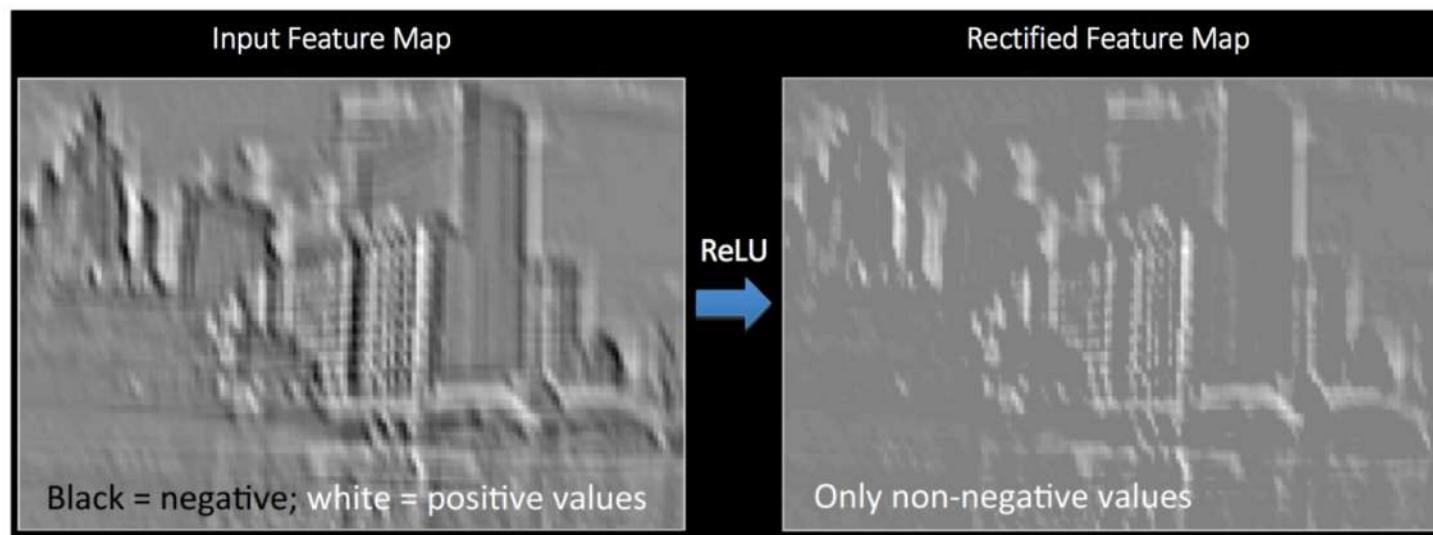
Filter step size

**Receptive Field:**

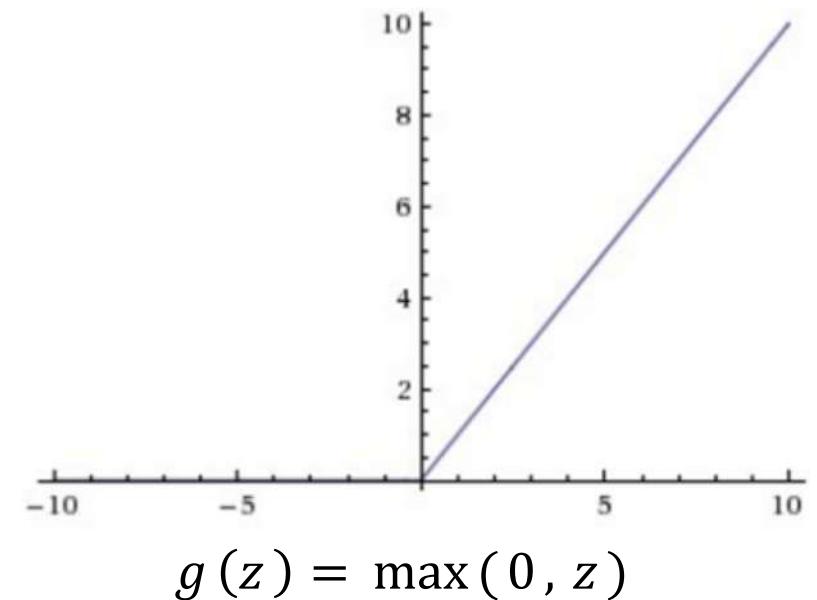
Locations in input image that  
a node is path connected to

# Introducing Non-Linearity

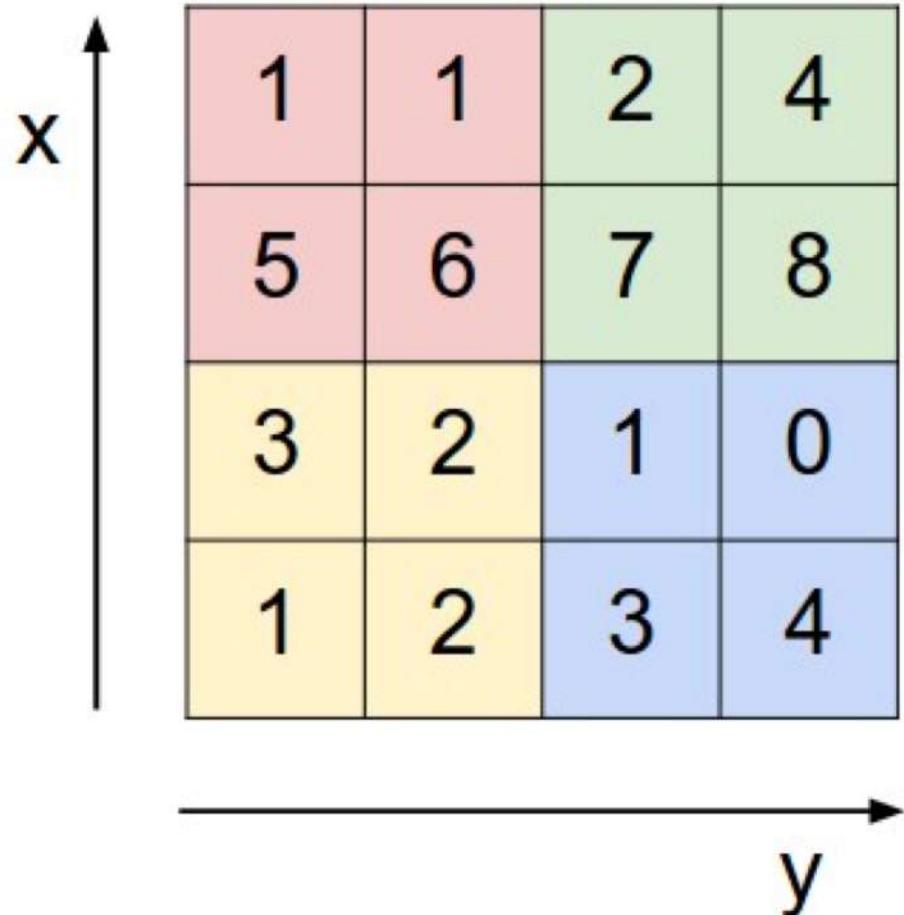
- Apply after every convolution operation (i.e., after convolutional layers)
- ReLU: pixel-by-pixel operation that replaces all negative values by zero. **Non-linear operation!**



Rectified Linear Unit (ReLU)



# Pooling



max pool with 2x2 filters  
and stride 2

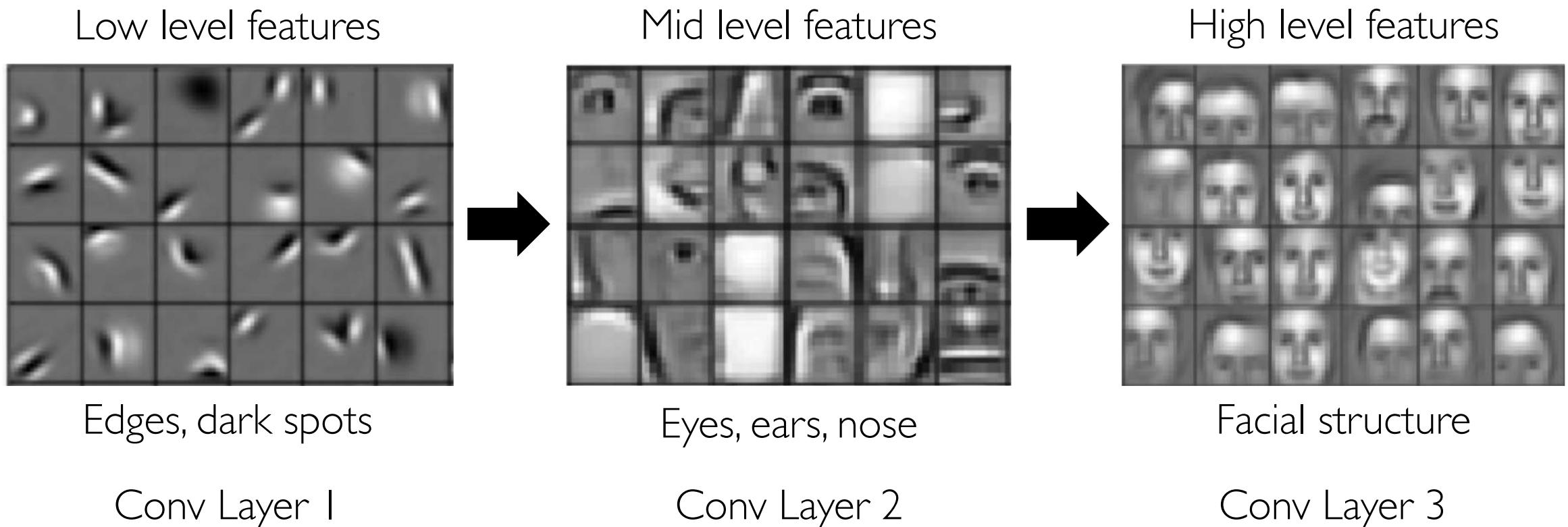


6	8
3	4

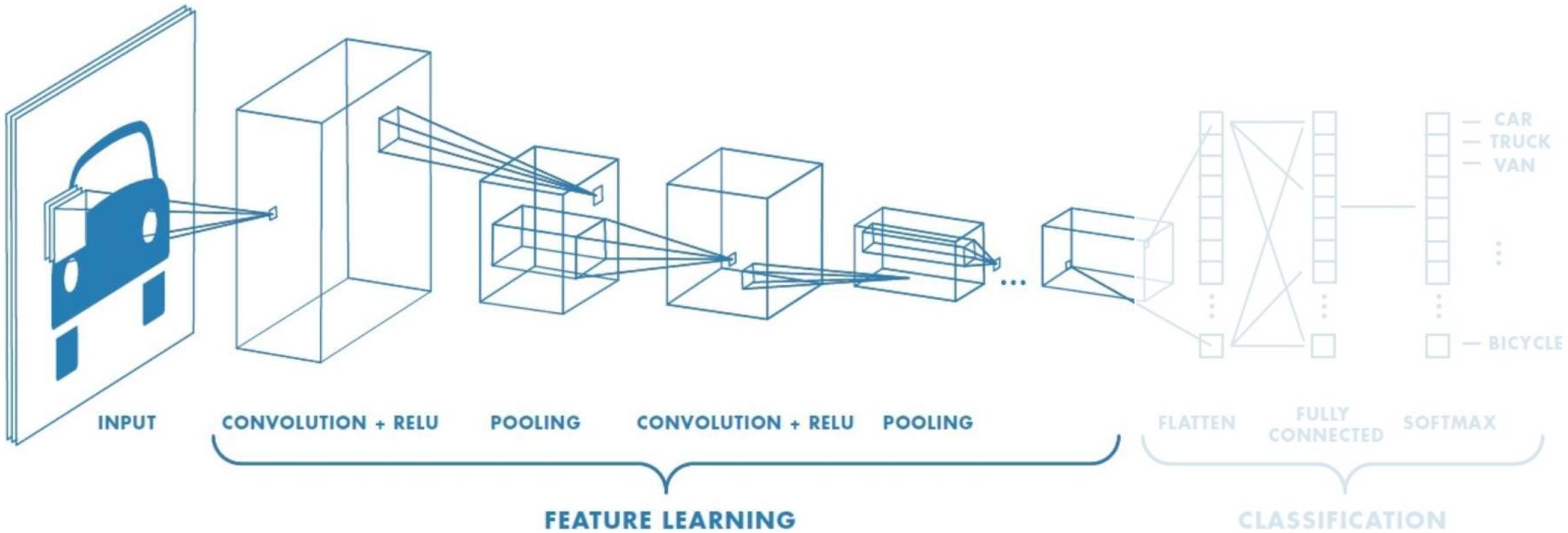
- 1) Reduced dimensionality
- 2) Spatial invariance

How else can we downsample and preserve spatial invariance?

# Representation Learning in Deep CNNs

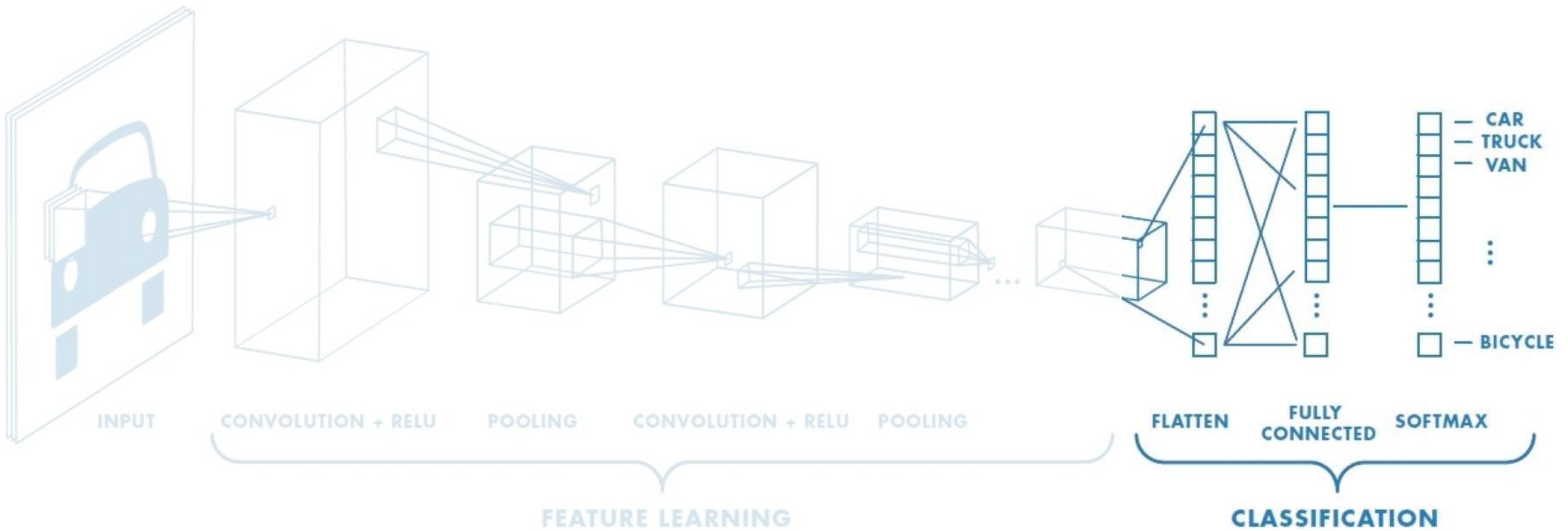


# CNNs for Classification: Feature Learning



1. Learn features in input image through **convolution**
2. Introduce **non-linearity** through activation function (real-world data is non-linear!)
3. Reduce dimensionality and preserve spatial invariance with **pooling**

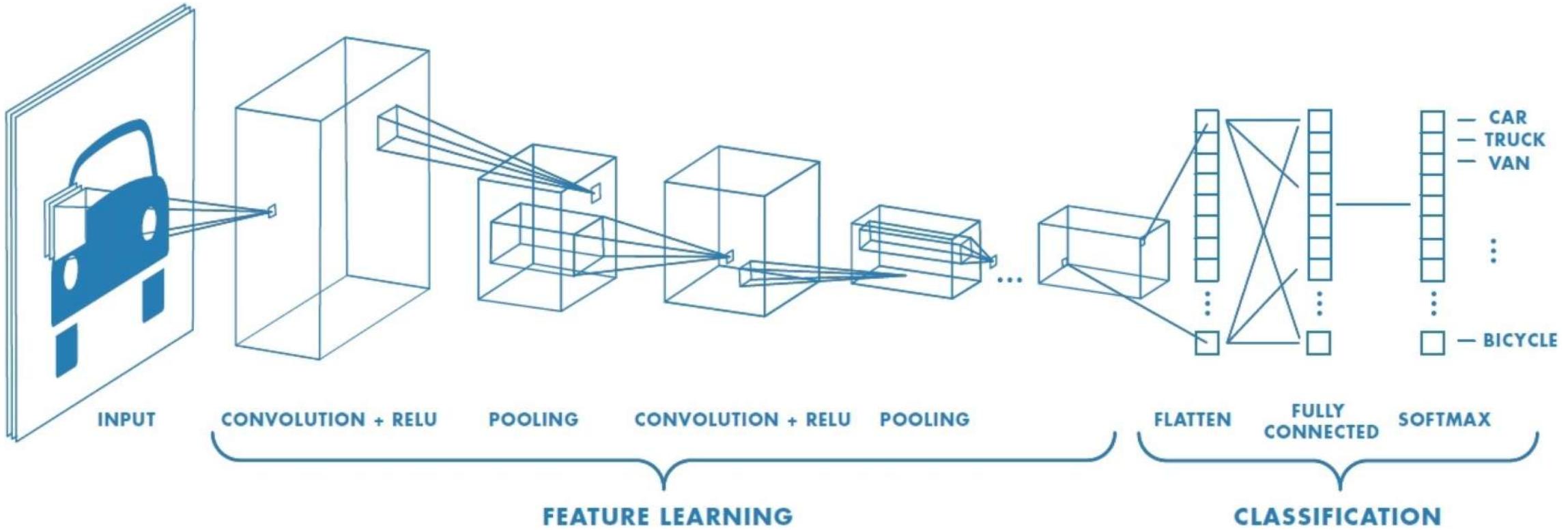
# CNNs for Classification: Class Probabilities



- CONV and POOL layers output high-level features of input
- Fully connected layer uses these features for classifying input image
- Express output as **probability** of image belonging to a particular class

$$\text{softmax}(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}}$$

# CNNs: Training with Backpropagation



Learn weights for convolutional filters and fully connected layers

Backpropagation: cross-entropy loss

$$J(\theta) = \sum_i y^{(i)} \log(\hat{y}^{(i)})$$

# CNNs for Classification: ImageNet

# ImageNet Dataset

Dataset of over 14 million images across 21,841 categories

*“Elongated crescent-shaped yellow fruit with soft sweet flesh”*



1409 pictures of bananas.

# ImageNet Challenge



## ImageNet Large Scale Visual Recognition Challenges



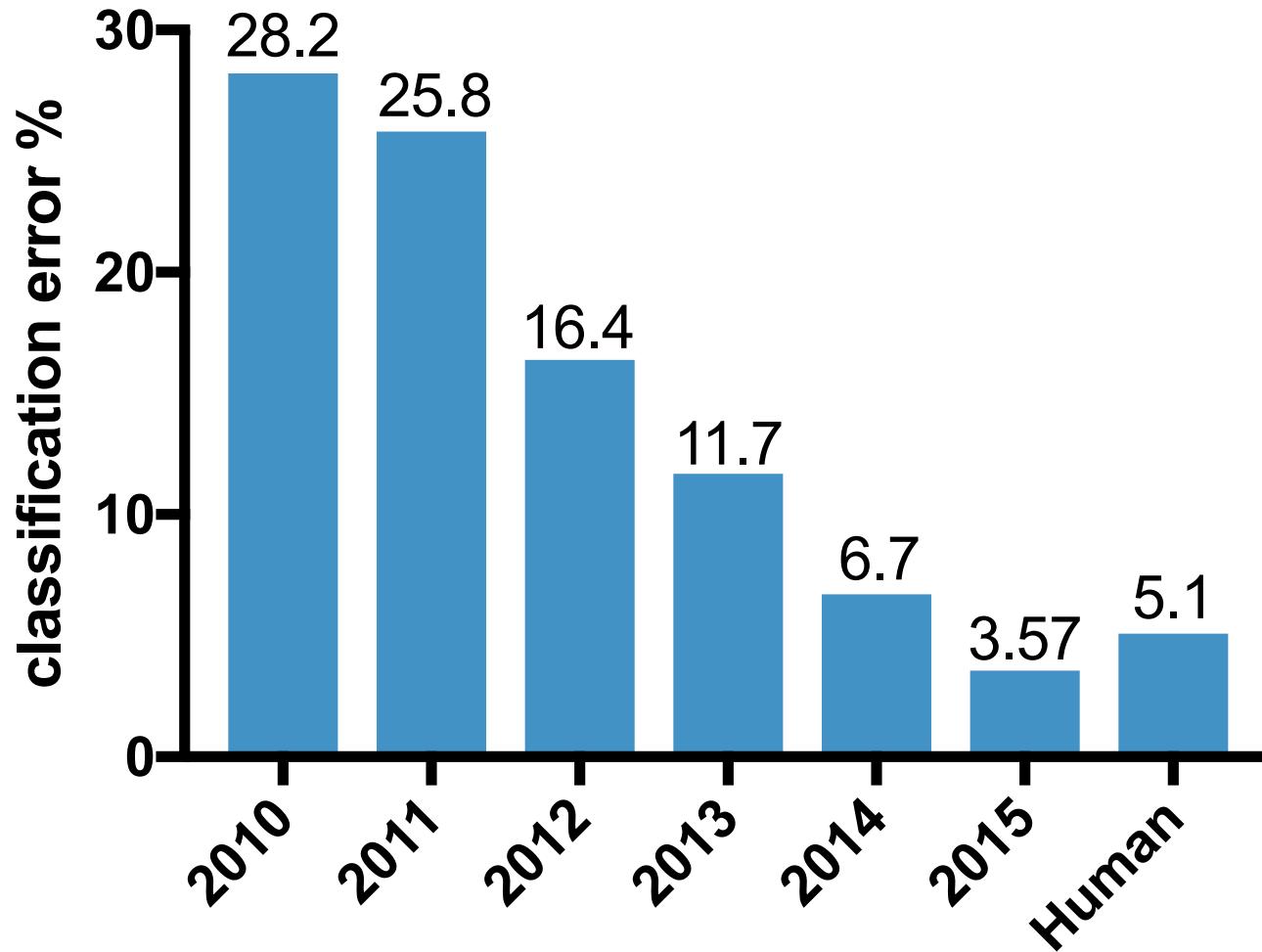
**Classification task:** produce a list of object categories present in image. 1000 categories.

“Top 5 error”: rate at which the model does not output correct label in top 5 predictions

Other tasks include:

single-object localization, object detection from video/image, scene classification, scene parsing

# ImageNet Challenge: Classification Task



**2012: AlexNet. First CNN to win.**

- 8 layers, 61 million parameters

**2013: ZFNet**

- 8 layers, more filters

**2014: VGG**

- 19 layers

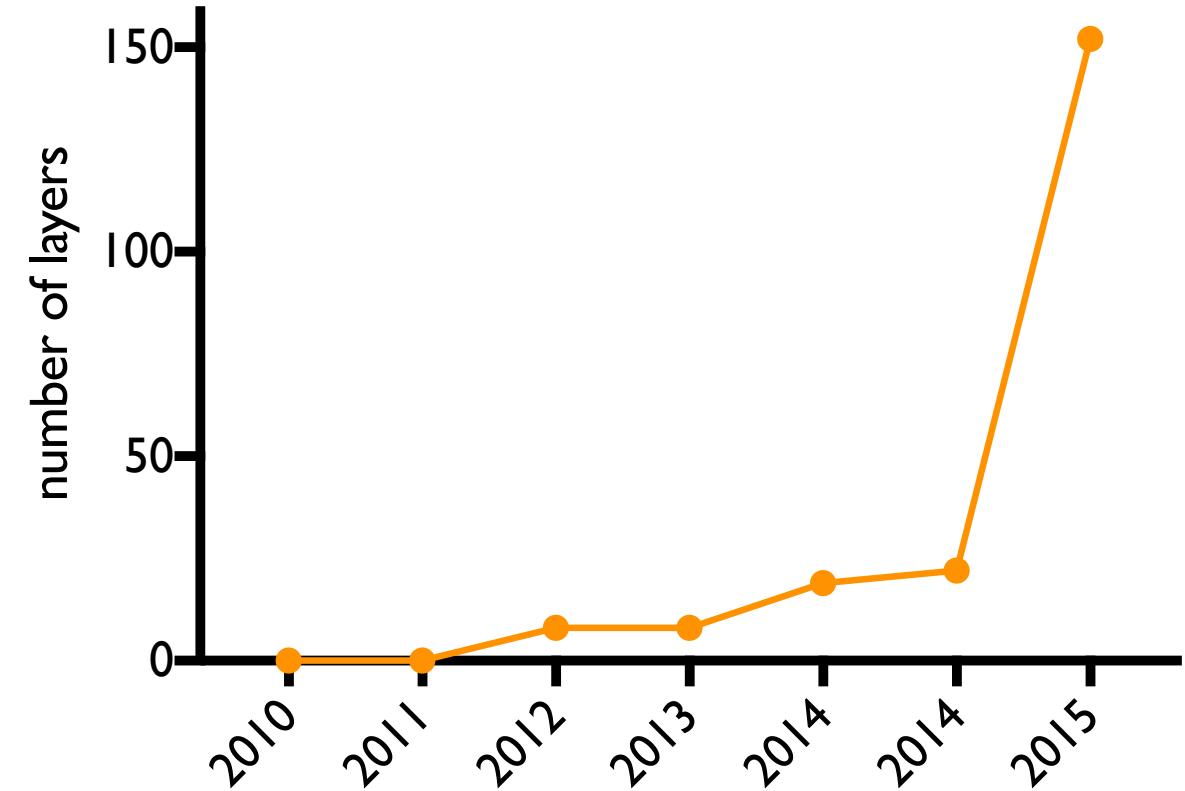
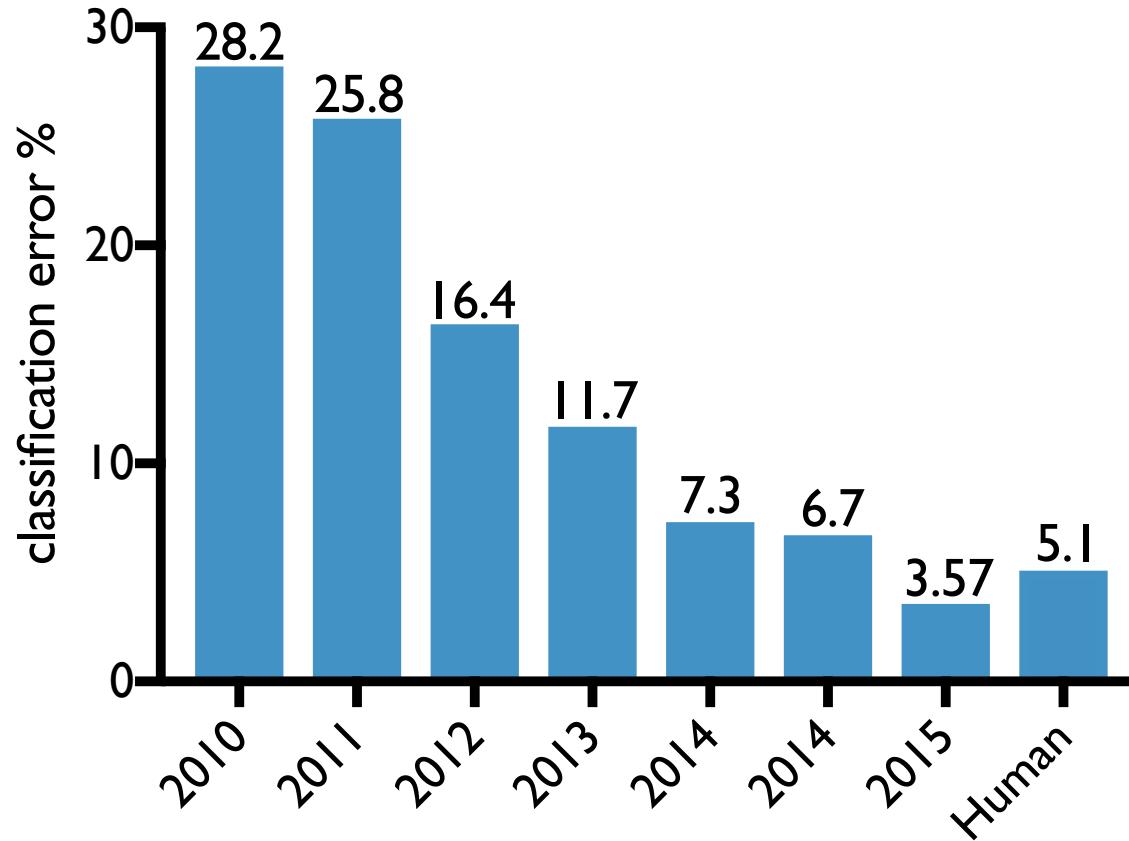
**2014: GoogLeNet**

- “Inception” modules
- 22 layers, 5 million parameters

**2015: ResNet**

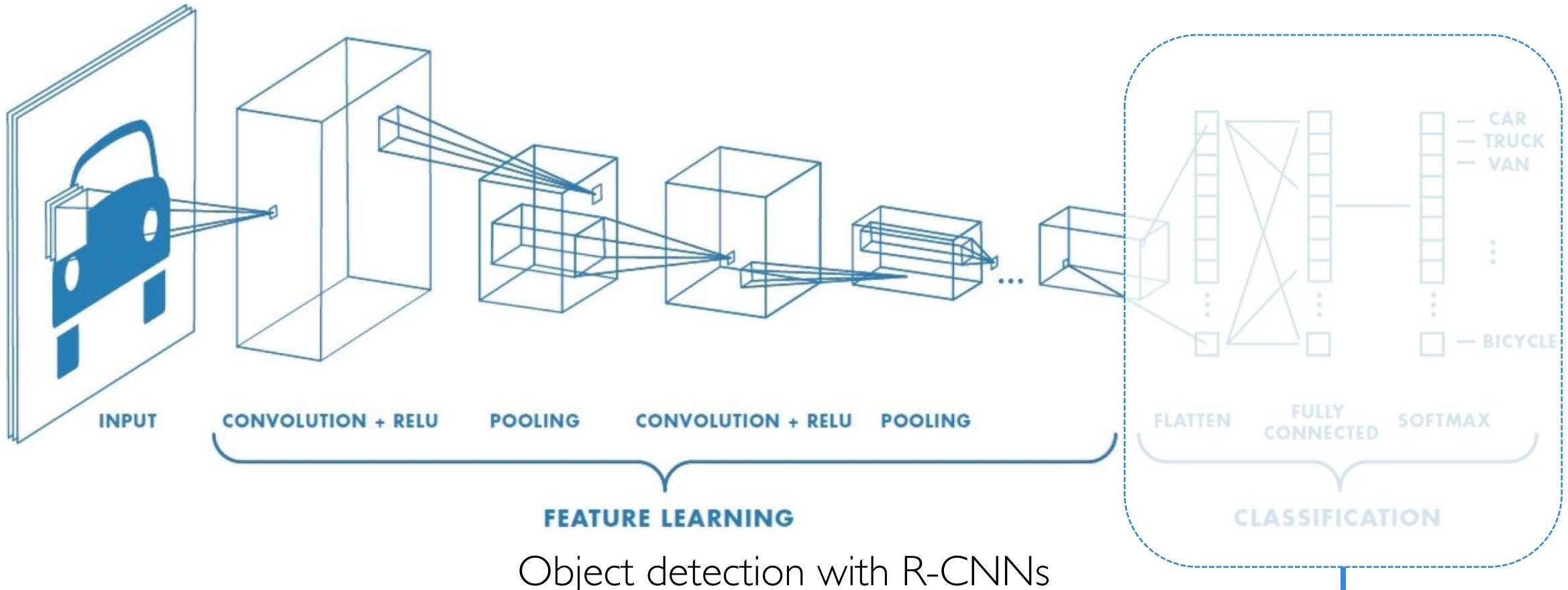
- 152 layers

# ImageNet Challenge: Classification Task



# An Architecture for Many Applications

# An Architecture for Many Applications



Object detection with R-CNNs  
Segmentation with fully convolutional networks  
Image captioning with RNNs

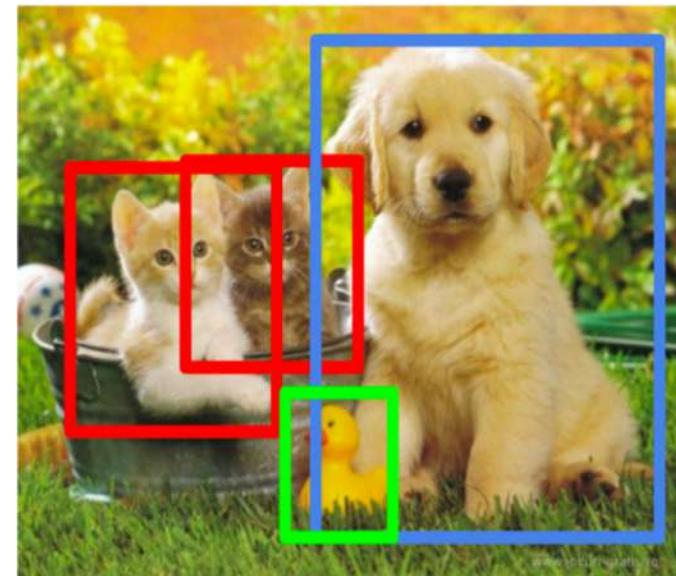
# Beyond Classification

Semantic Segmentation



CAT

Object Detection



CAT, DOG, DUCK

Image Captioning



The cat is in the grass.

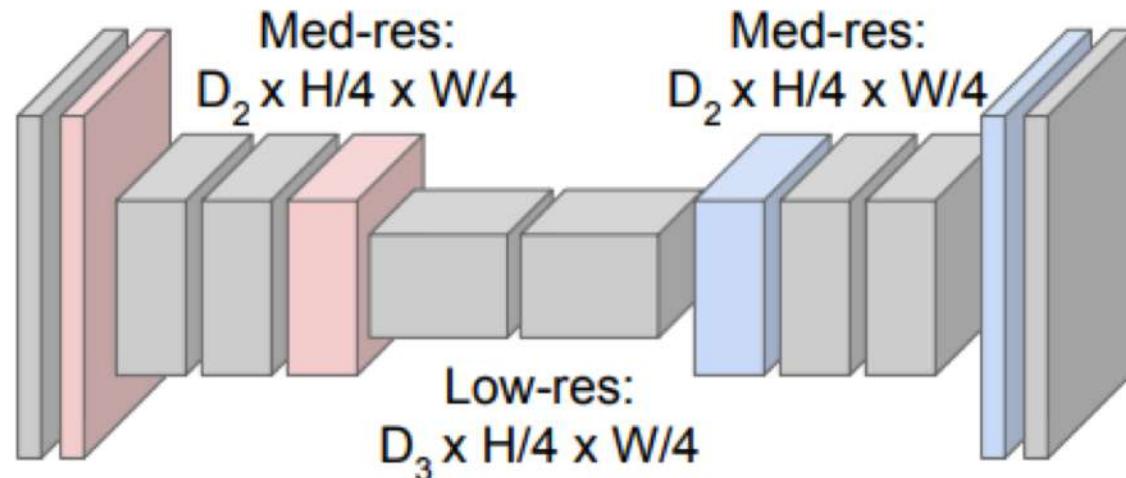
# Semantic Segmentation: FCNs

FCN: Fully Convolutional Network.

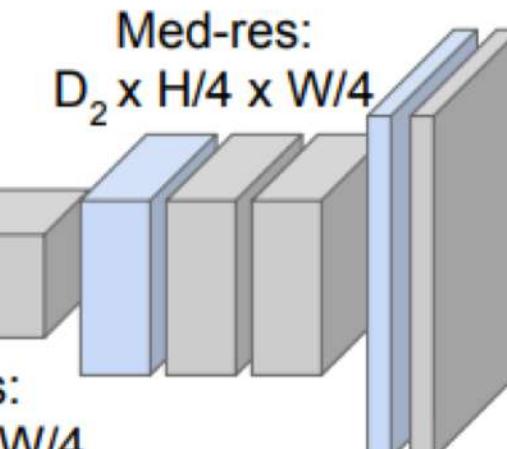
Network designed with all convolutional layers,  
with **downsampling** and **upsampling** operations



Input:  
 $3 \times H \times W$



High-res:  
 $D_1 \times H/2 \times W/2$

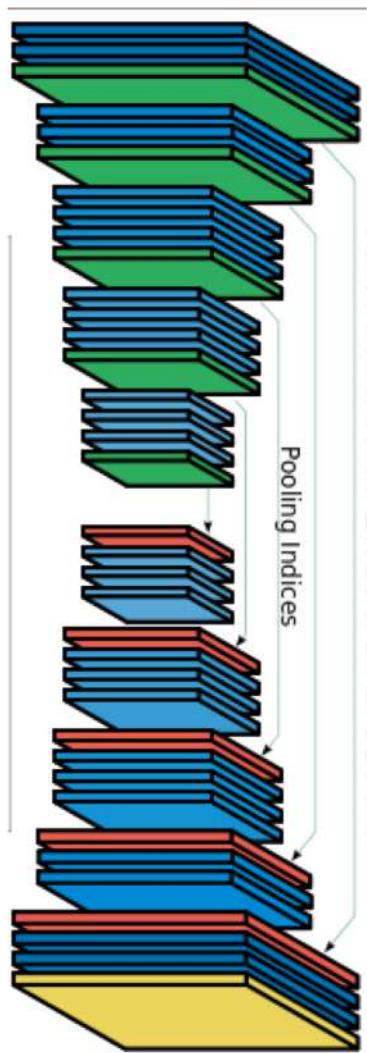


High-res:  
 $D_1 \times H/2 \times W/2$

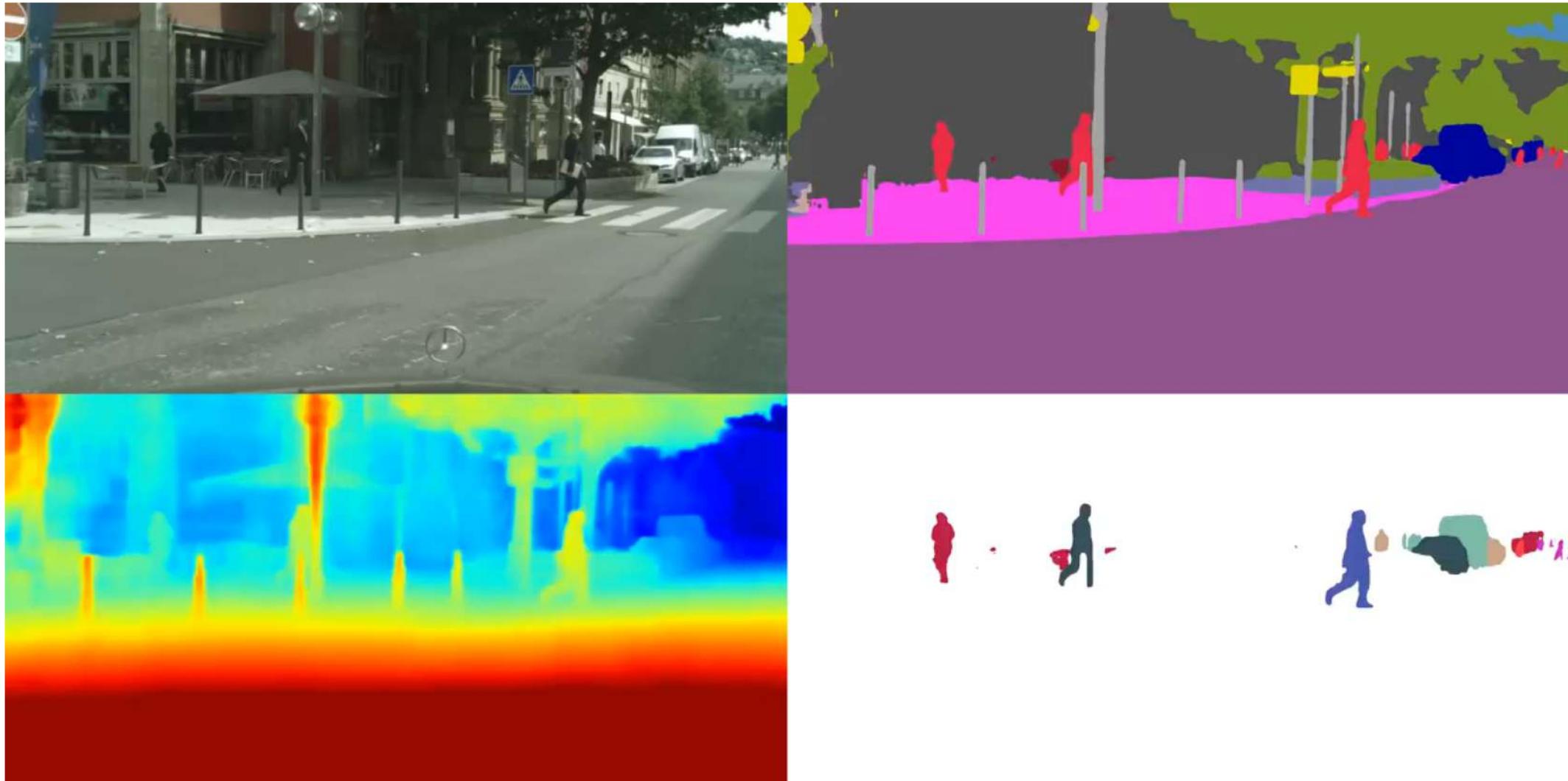


Predictions:  
 $H \times W$

# Driving Scene Segmentation

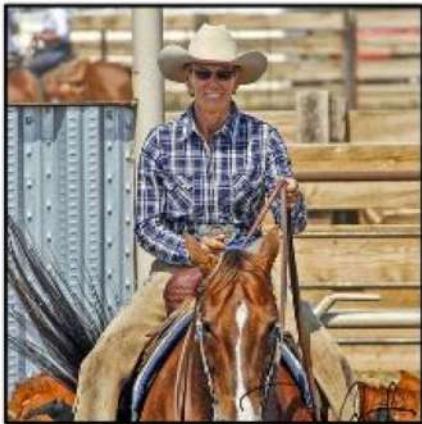


# Driving Scene Segmentation

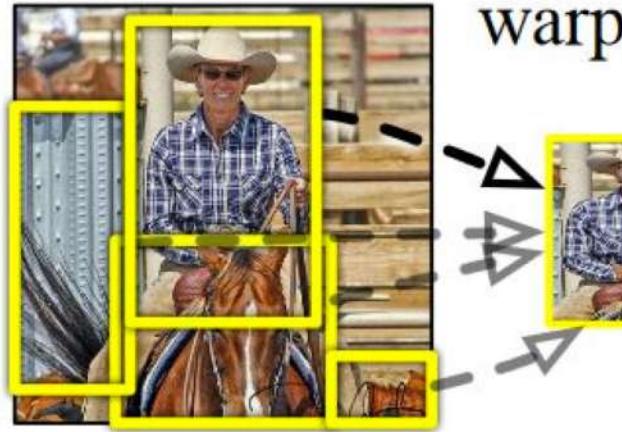


# Object Detection with R-CNNs

R-CNN: Find regions that we think have objects. Use CNN to classify.

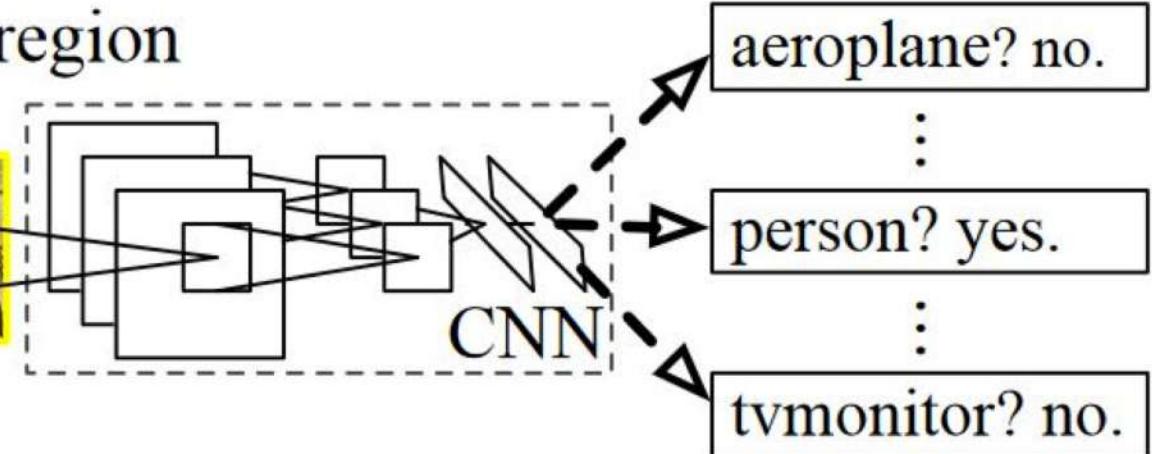


1. Input  
image



2. Extract region  
proposals (~2k)

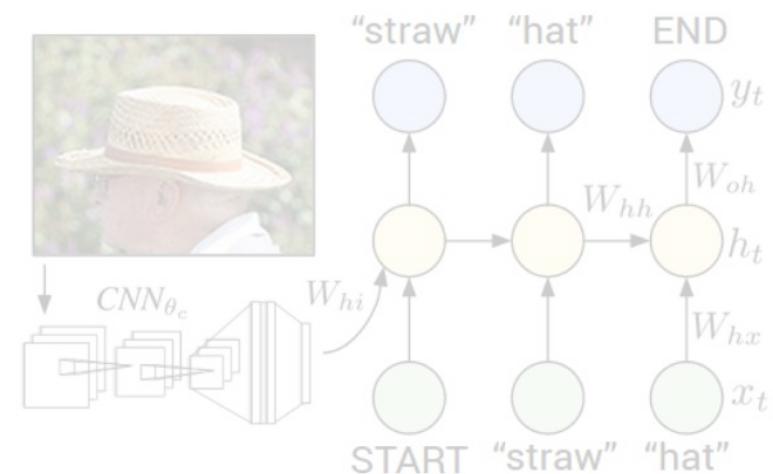
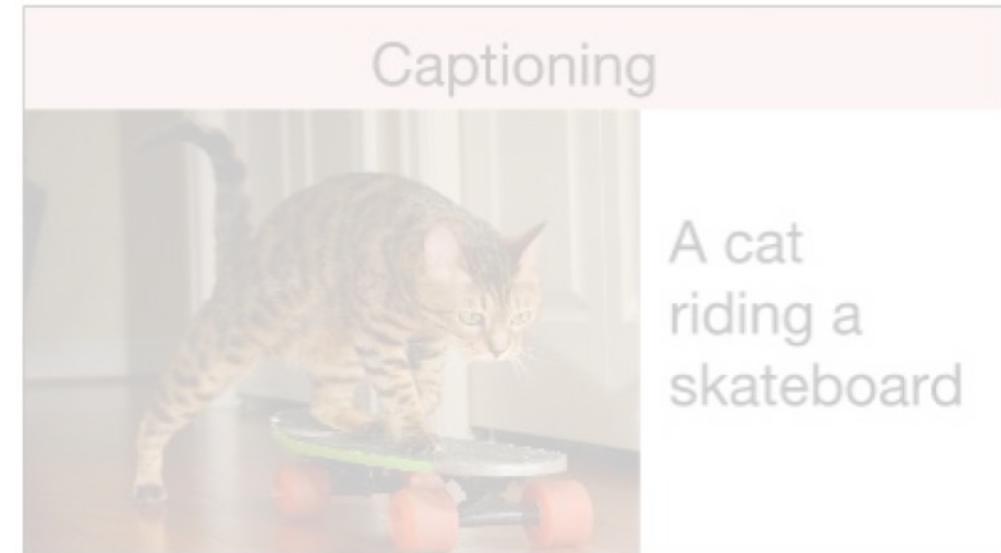
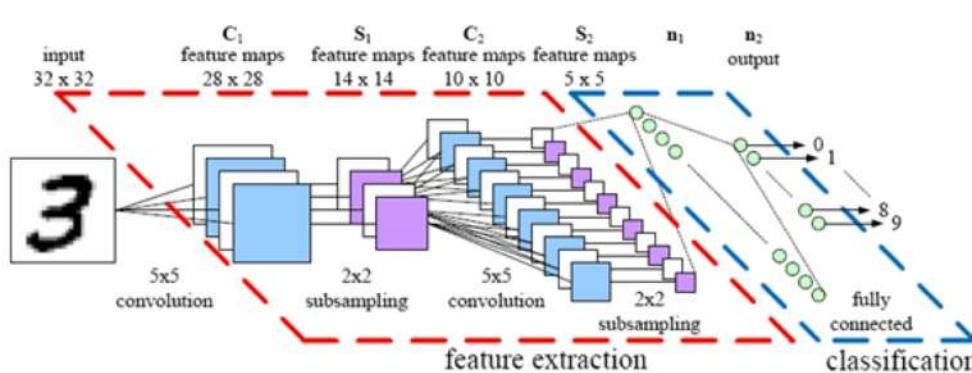
warped region



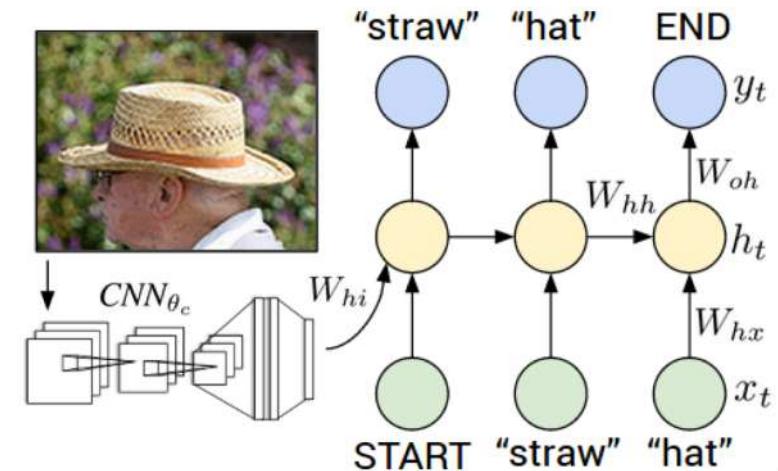
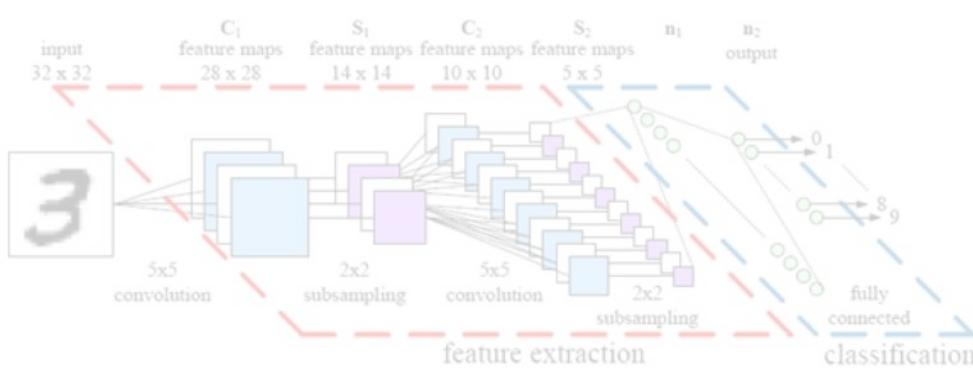
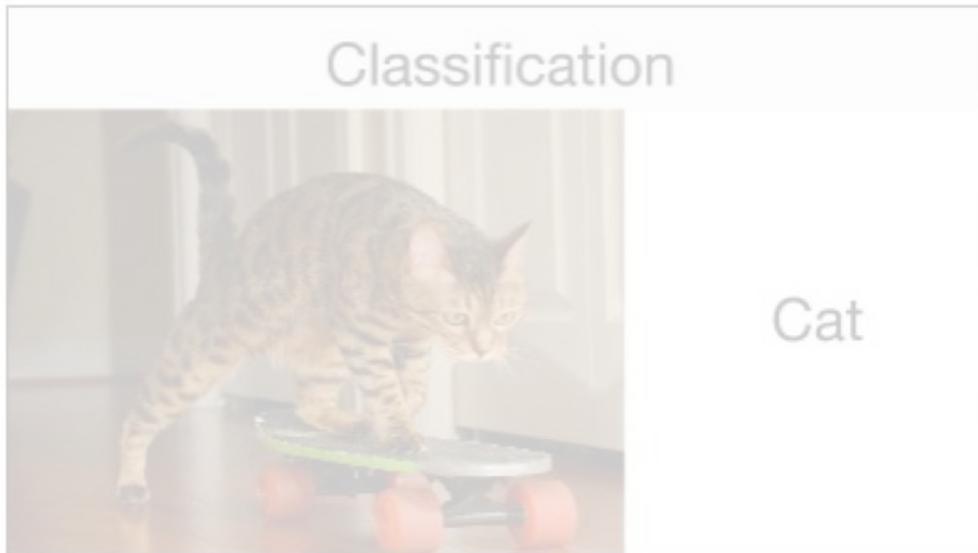
3. Compute  
CNN features

4. Classify  
regions

# Image Captioning using RNNs

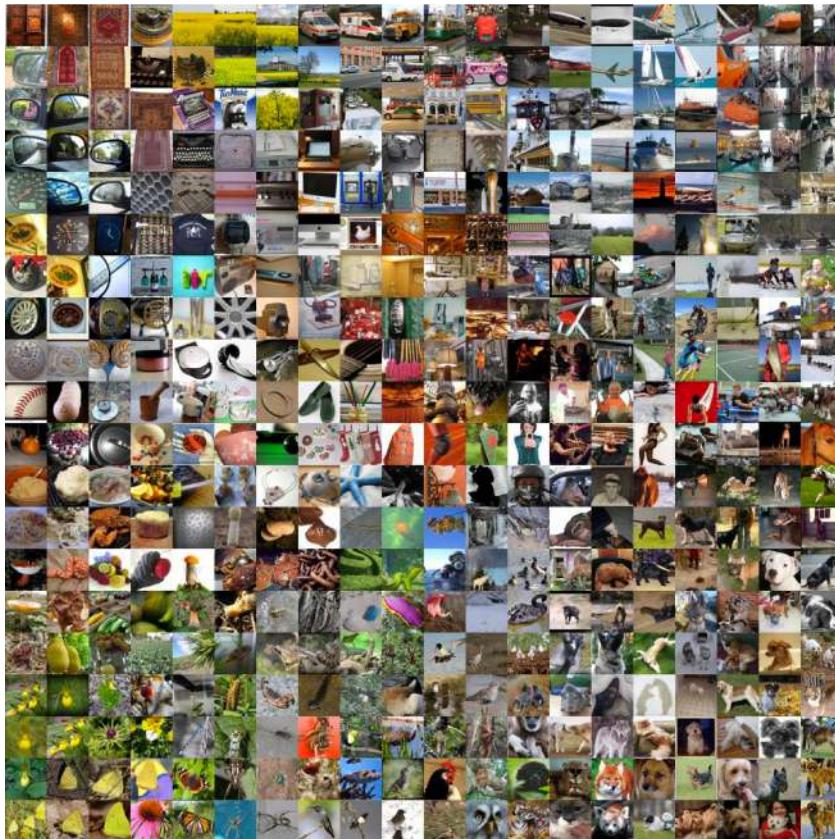


# Image Captioning using RNNs



# Deep Learning for Computer Vision: Impact and Summary

# Data, Data, Data



ImageNet:  
22K categories. 14M images.

Airplane

3	4	2	1	9	5	6	2	1	8
8	9	1	2	5	0	0	6	6	4
6	7	0	1	6	3	6	3	7	0
3	7	7	9	4	6	6	1	8	2
2	9	3	4	3	9	8	7	2	5
1	5	9	8	3	6	5	7	2	3
9	3	1	9	1	5	8	0	8	4
5	6	2	6	8	5	8	8	9	9
3	7	7	0	9	4	8	5	4	3
7	9	6	4	1	0	6	9	2	3

Automobile

Bird

Cat

Deer

Dog

Frog

Horse

Ship

Truck

CIFAR-10

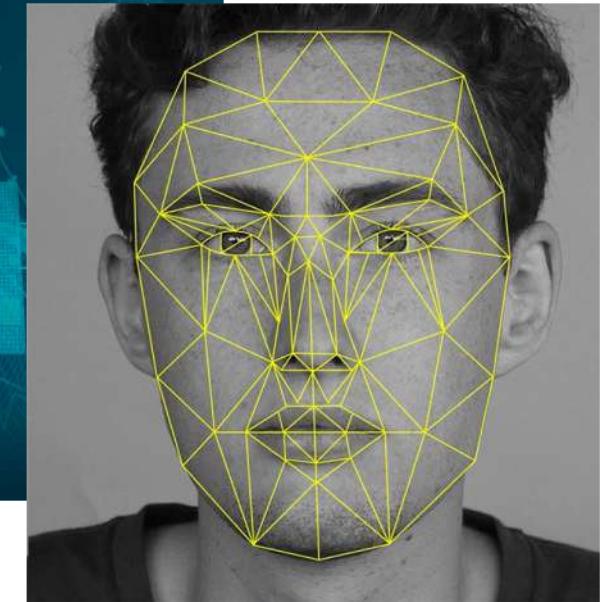
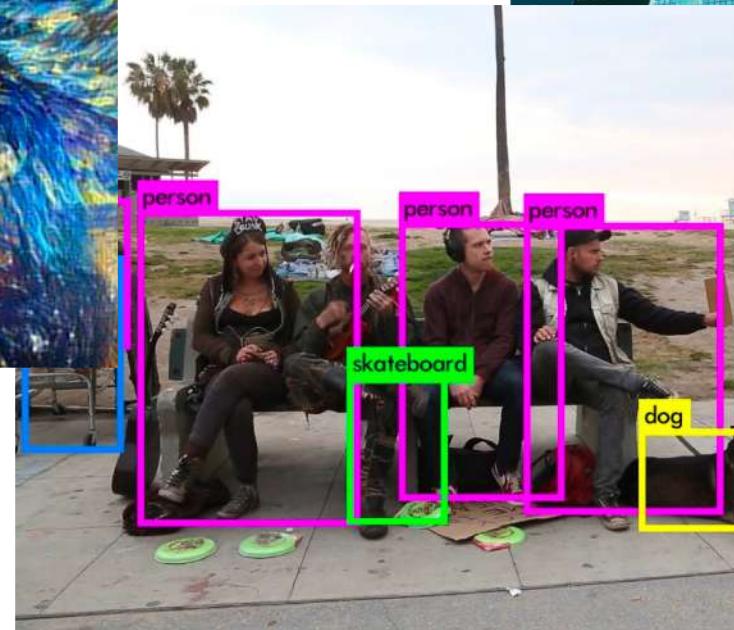
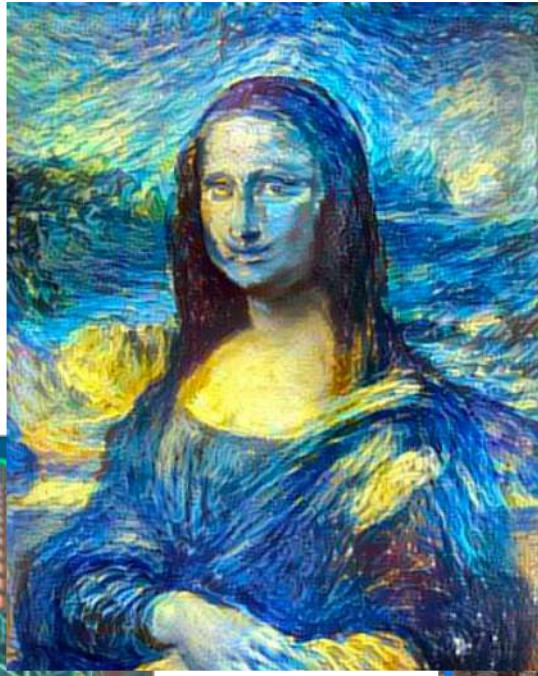
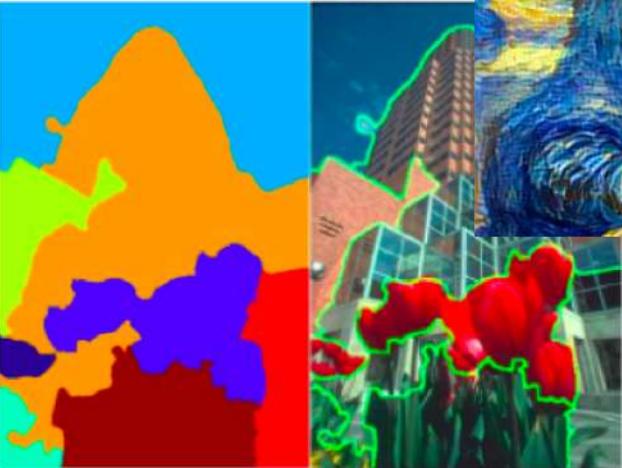
MNIST: handwritten digits

---

places   
THE SCENE RECOGNITION DATABASE

places: natural scenes

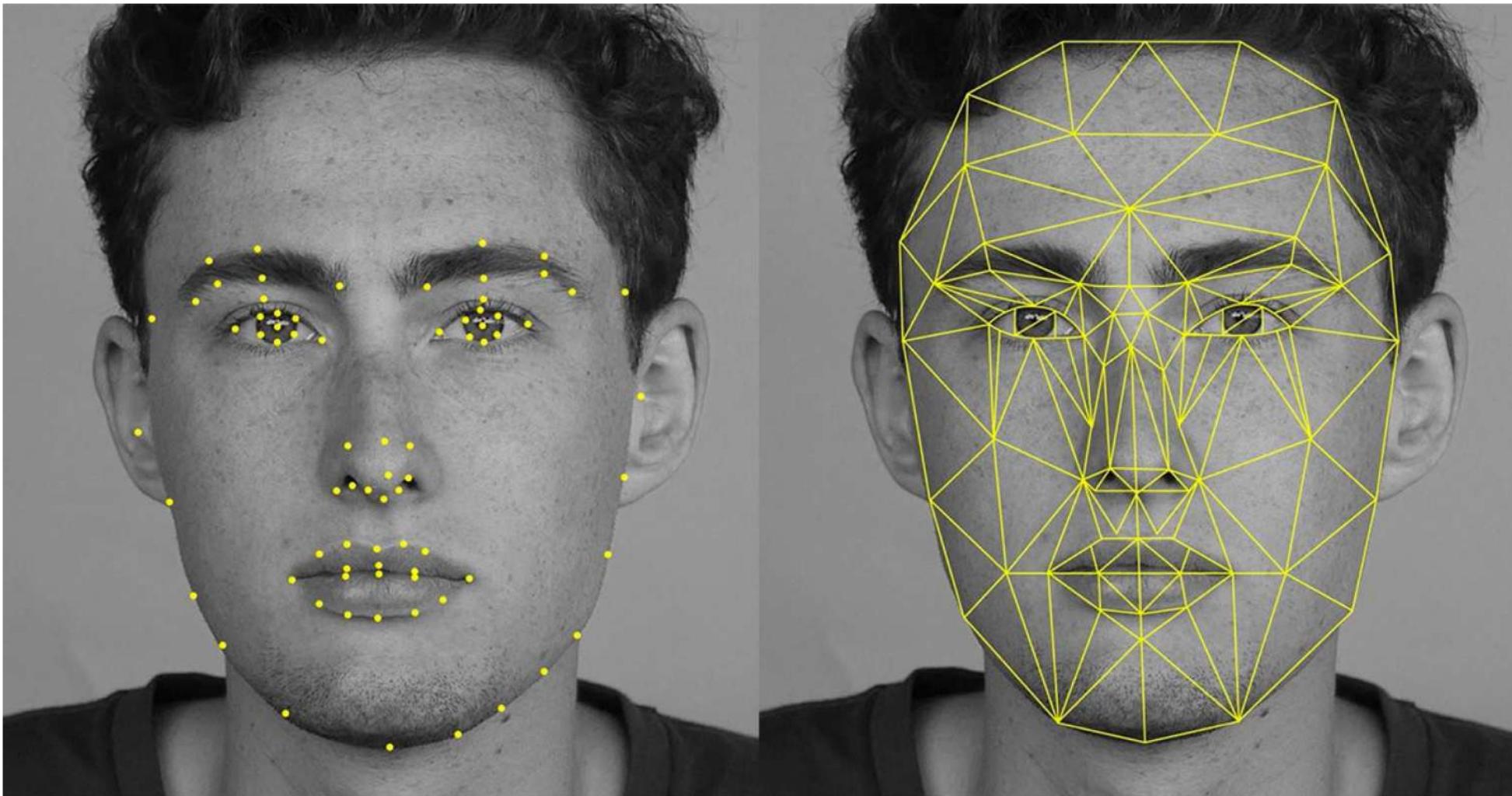
# Deep Learning for Computer Vision: Impact



# Impact: Face Detection



6.SI91 Lab!



# Impact: Self-Driving Cars

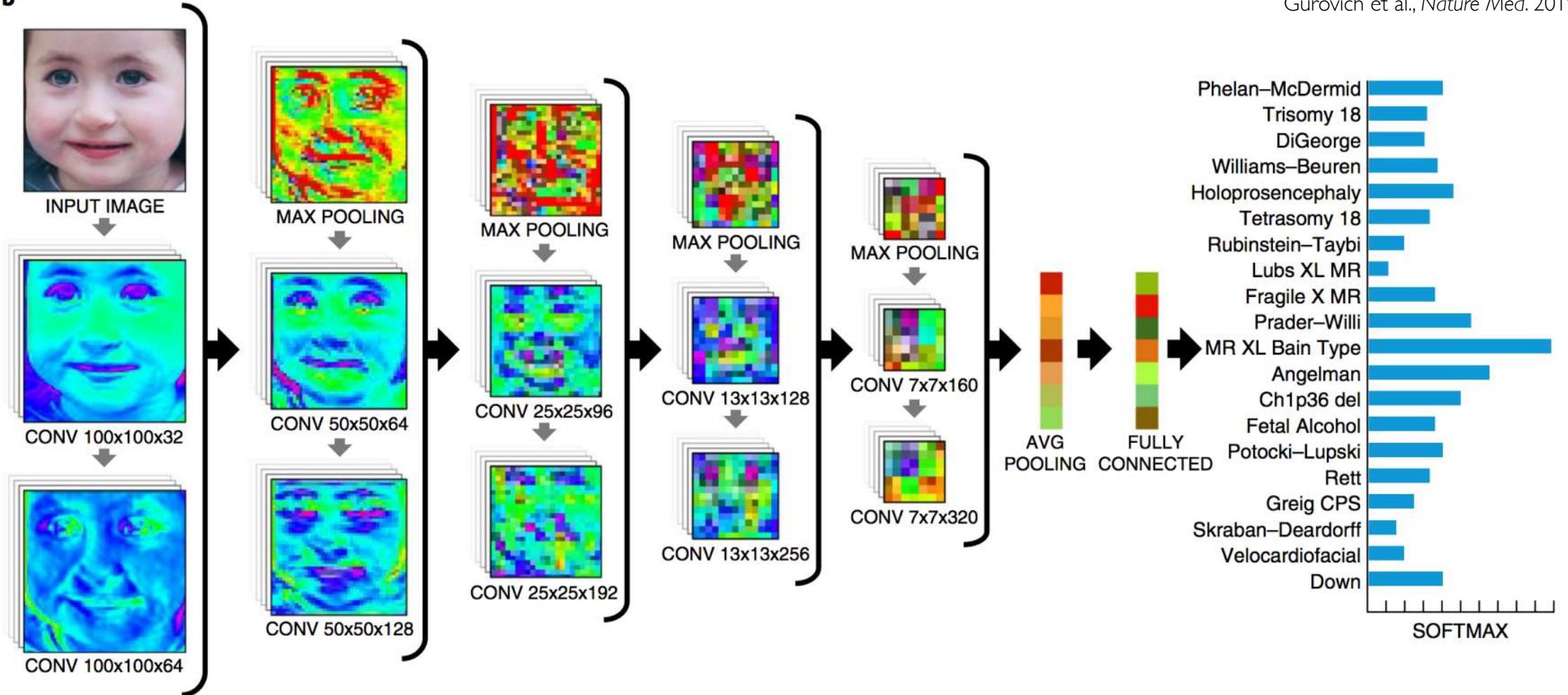


# Impact: Healthcare

Identifying facial phenotypes of genetic disorders using deep learning

Gurovich et al., Nature Med. 2019

b



# Deep Learning for Computer Vision: Summary

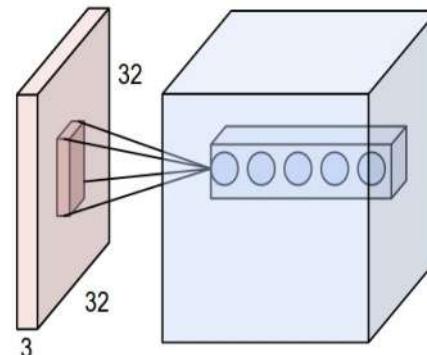
## Foundations

- Why computer vision?
- Representing images
- Convolutions for feature extraction



## CNNs

- CNN architecture
- Application to classification
- ImageNet



## Applications

- Segmentation, object detection, image captioning
- Visualization



# Which face is fake?



# Supervised vs unsupervised learning

## Supervised Learning

**Data:**  $(x, y)$

$x$  is data,  $y$  is label

**Goal:** Learn function to map  
 $x \rightarrow y$

**Examples:** Classification,  
regression, object detection,  
semantic segmentation, etc.

## Unsupervised Learning

**Data:**  $x$

$x$  is data, no labels!

**Goal:** Learn some *hidden* or  
*underlying structure* of the data

**Examples:** Clustering, feature or  
dimensionality reduction, etc.

# Supervised vs unsupervised learning

## Supervised Learning

**Data:**  $(x, y)$

$x$  is data,  $y$  is label

**Goal:** Learn function to map  
 $x \rightarrow y$

**Examples:** Classification,  
regression, object detection,  
semantic segmentation, etc.

## Unsupervised Learning

**Data:**  $x$

$x$  is data, no labels!

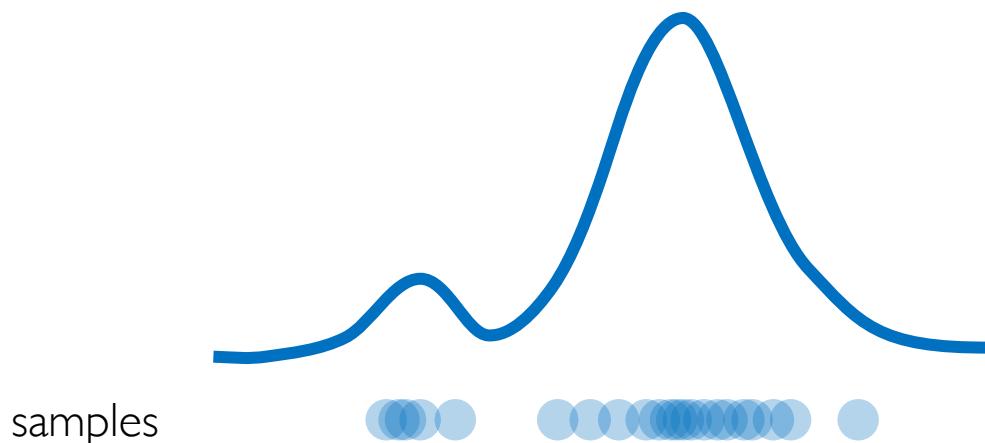
**Goal:** Learn some *hidden* or  
*underlying structure* of the data

**Examples:** Clustering, feature or  
dimensionality reduction, etc.

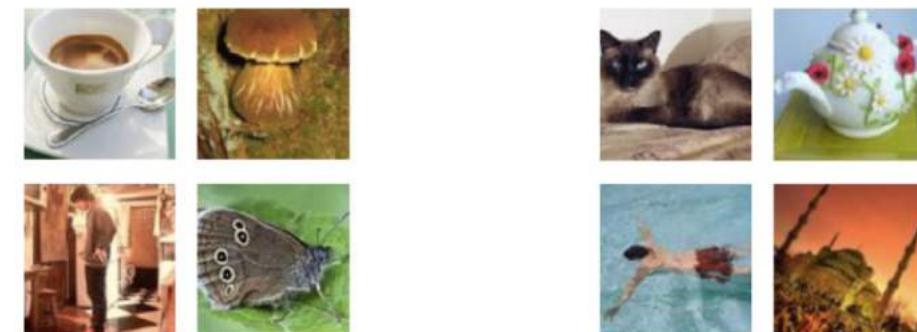
# Generative modeling

**Goal:** Take as input training samples from some distribution and learn a model that represents that distribution

Density Estimation



Sample Generation



Input samples

Training data  $\sim P_{data}(x)$

Generated samples

Generated  $\sim P_{model}(x)$

How can we learn  $P_{model}(x)$  similar to  $P_{data}(x)$ ?

# Why generative models? Debiasing

Capable of uncovering **underlying latent variables** in a dataset



Homogeneous skin color, pose

VS

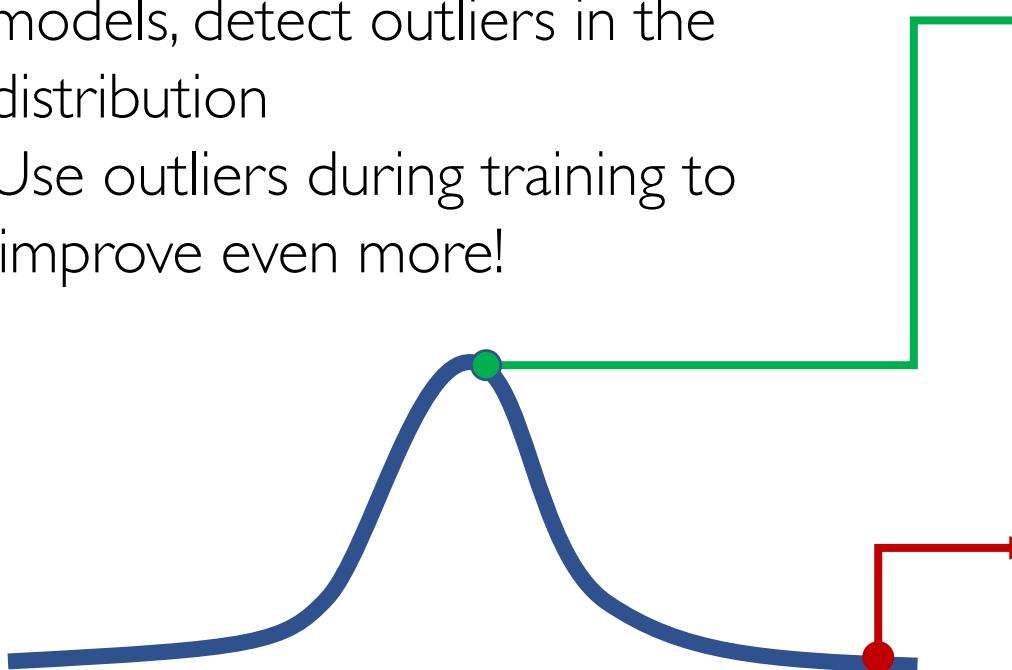


Diverse skin color, pose, illumination?

How can we use latent distributions to create fair and representative datasets?

# Why generative models? Outlier detection

- **Problem:** How can we detect when we encounter something new or rare?
- **Strategy:** Leverage generative models, detect outliers in the distribution
- Use outliers during training to improve even more!



## 95% of Driving Data:

- (1) sunny, (2) highway, (3) straight road



Detect outliers to avoid unpredictable behavior when training



Edge Cases



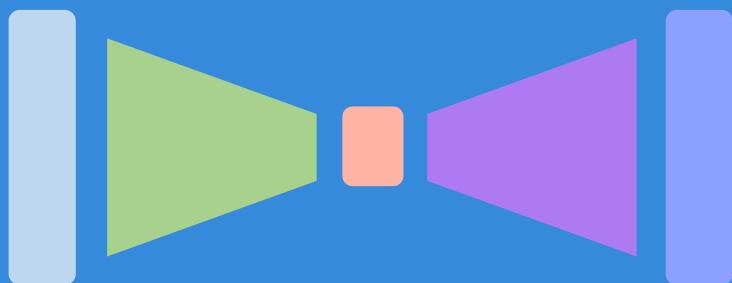
Harsh Weather



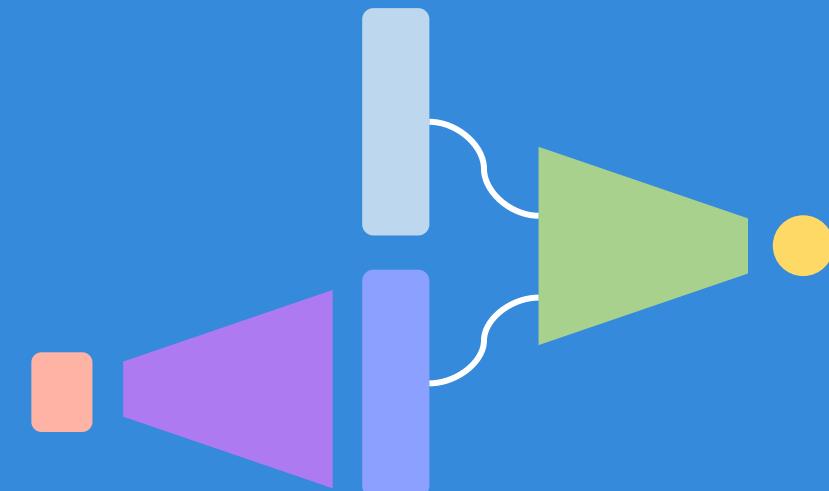
Pedestrians

# Latent variable models

Autoencoders and Variational  
Autoencoders (VAEs)



Generative Adversarial  
Networks (GANs)



# What is a latent variable?



*Myth of the Cave*

# What is a latent variable?

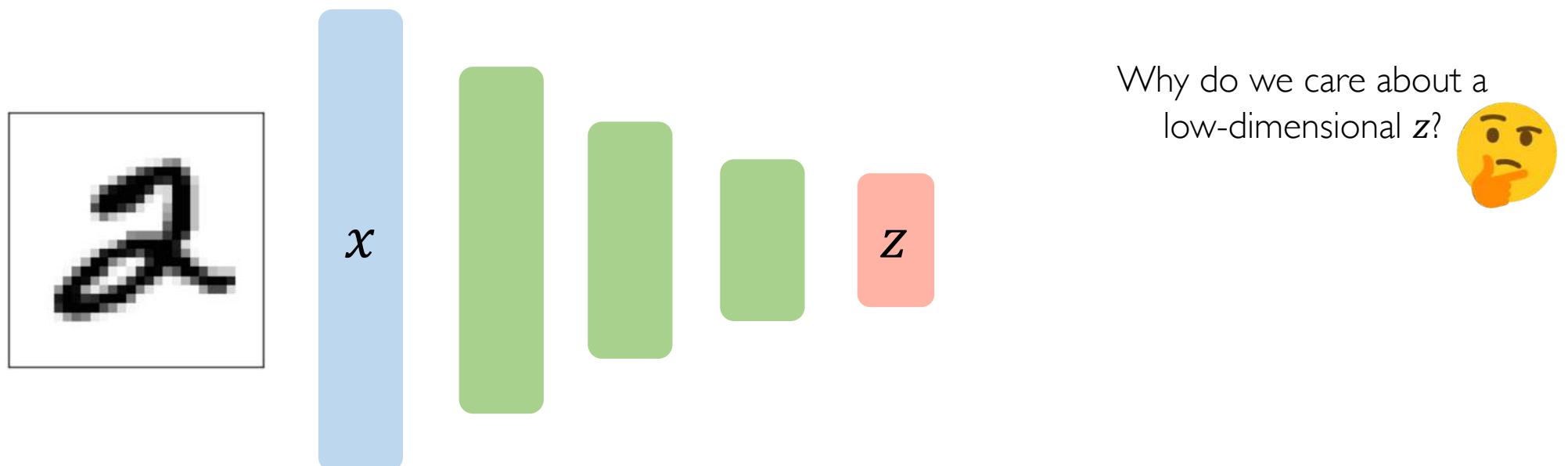


Can we learn the **true explanatory factors**, e.g. latent variables, from only observed data?

# Autoencoders

# Autoencoders: background

Unsupervised approach for learning a **lower-dimensional** feature representation from unlabeled training data

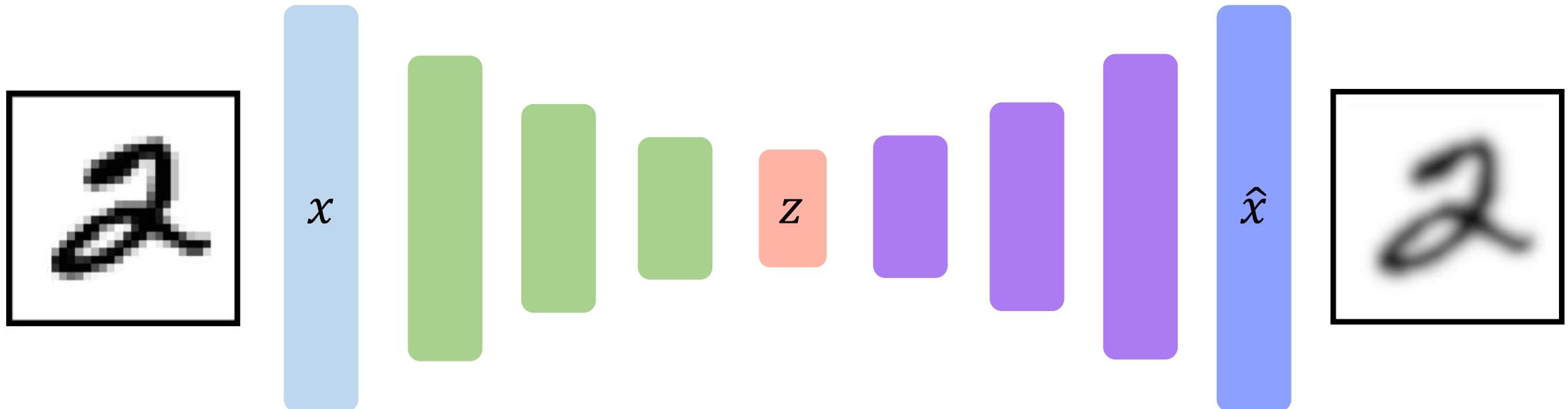


“Encoder” learns mapping from the data,  $x$ , to a low-dimensional latent space,  $z$

# Autoencoders: background

How can we learn this latent space?

Train the model to use these features to **reconstruct the original data**

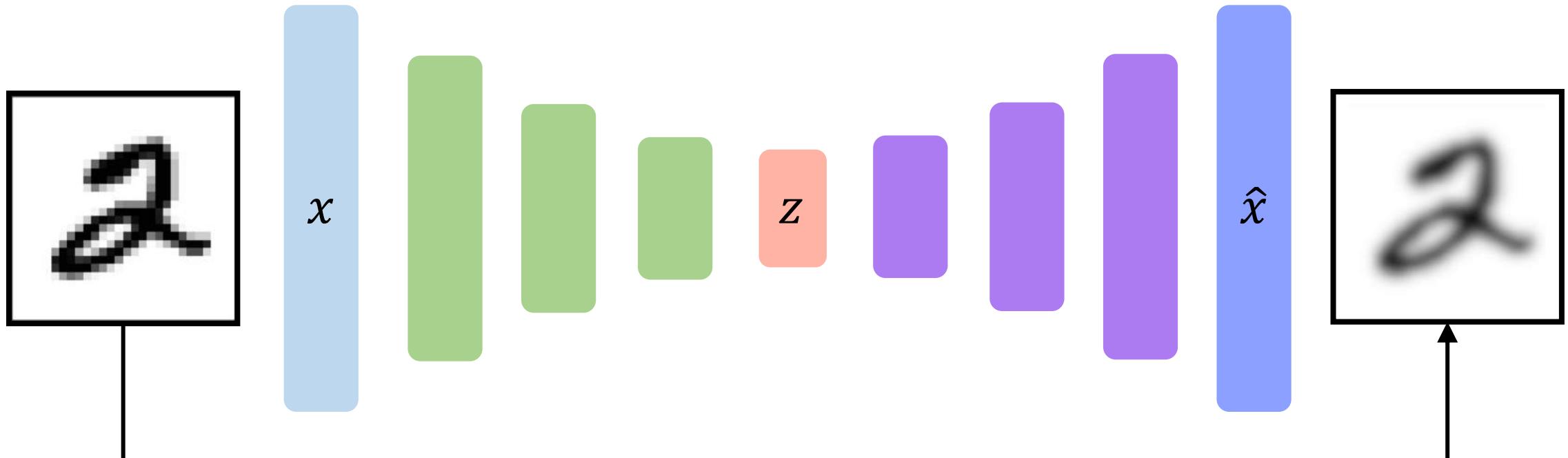


“Decoder” learns mapping back from latent,  $z$ , to a reconstructed observation,  $\hat{x}$

# Autoencoders: background

How can we learn this latent space?

Train the model to use these features to **reconstruct the original data**



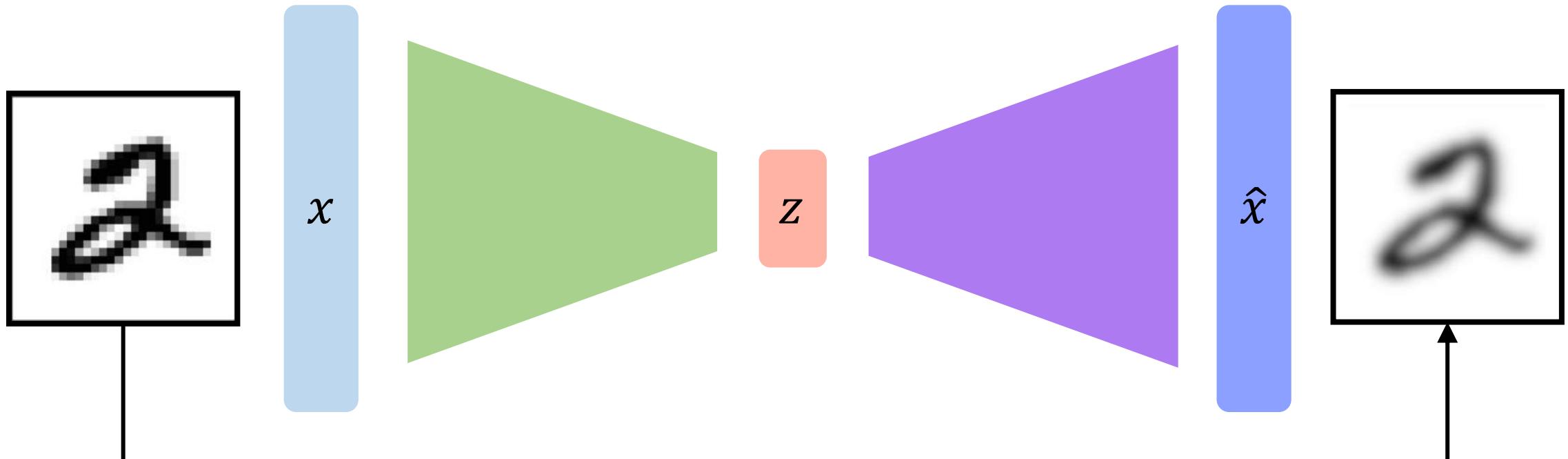
$$\mathcal{L}(x, \hat{x}) = \|x - \hat{x}\|^2$$

Loss function doesn't  
use any labels!!

# Autoencoders: background

How can we learn this latent space?

Train the model to use these features to **reconstruct the original data**



$$\mathcal{L}(x, \hat{x}) = \|x - \hat{x}\|^2$$

Loss function doesn't  
use any labels!!

# Dimensionality of latent space → reconstruction quality

Autoencoding is a form of compression!

Smaller latent space will force a larger training bottleneck

2D latent space



5D latent space



Ground Truth



# Autoencoders for representation learning

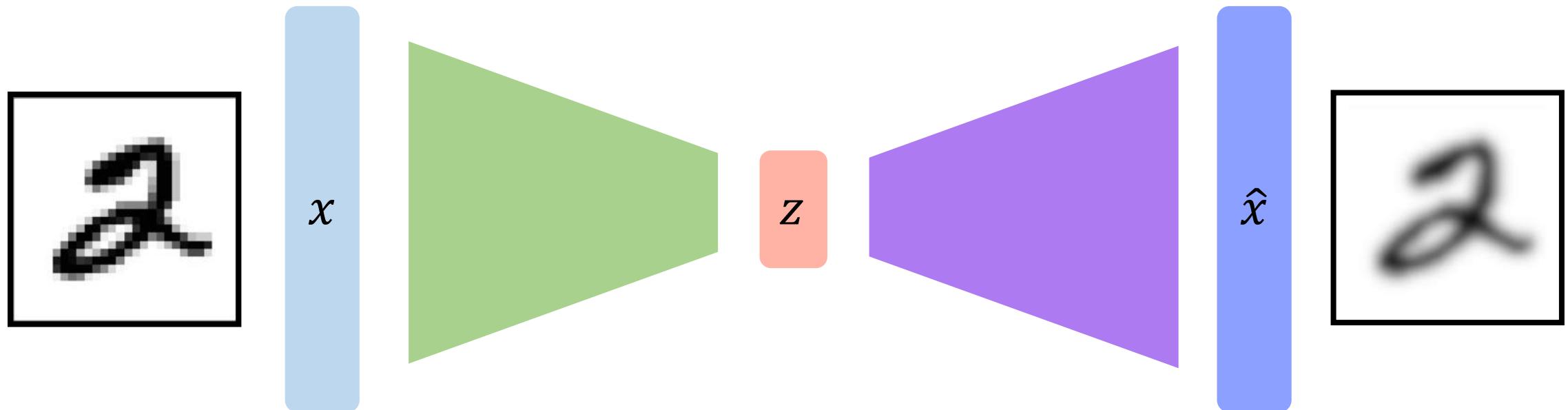
**Bottleneck hidden layer** forces network to learn a compressed latent representation

**Reconstruction loss** forces the latent representation to capture (or encode) as much “information” about the data as possible

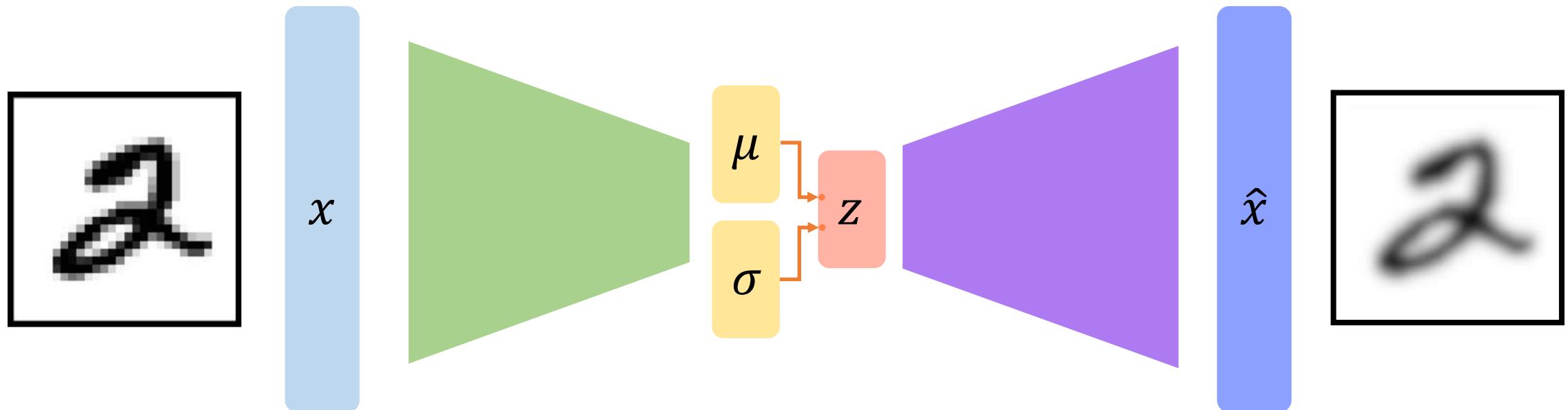
**Autoencoding** = **A**uto**m**atically **e**ncoding data

# Variational Autoencoders (VAEs)

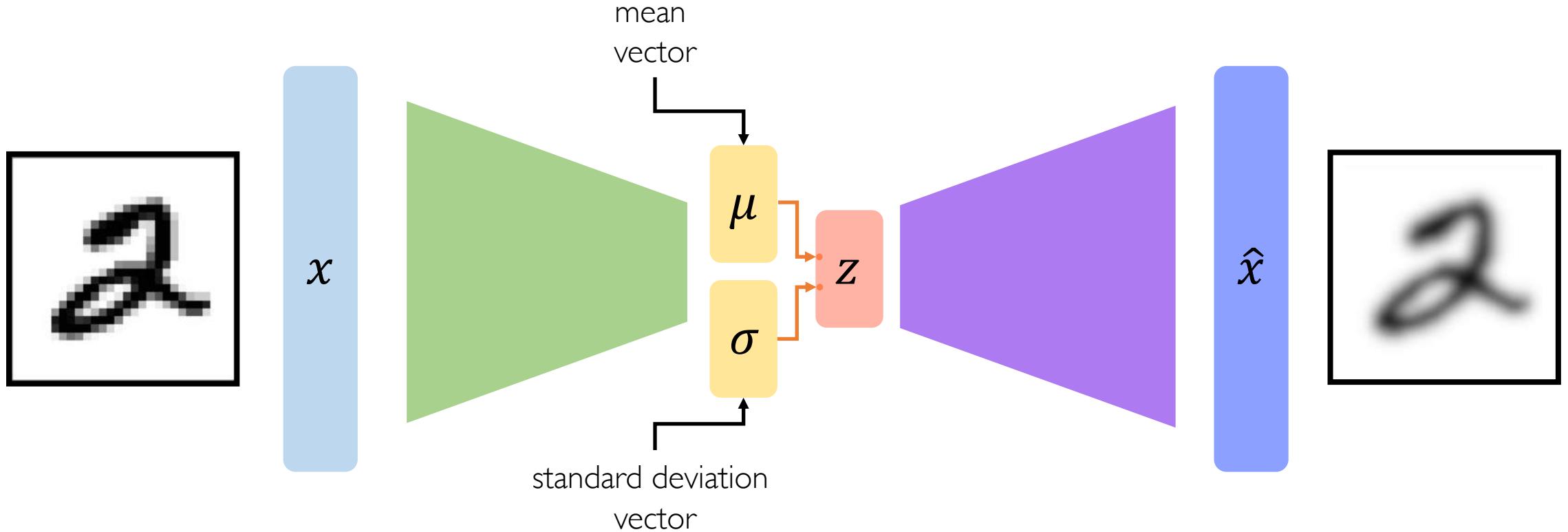
# VAEs: key difference with traditional autoencoder



# VAEs: key difference with traditional autoencoder



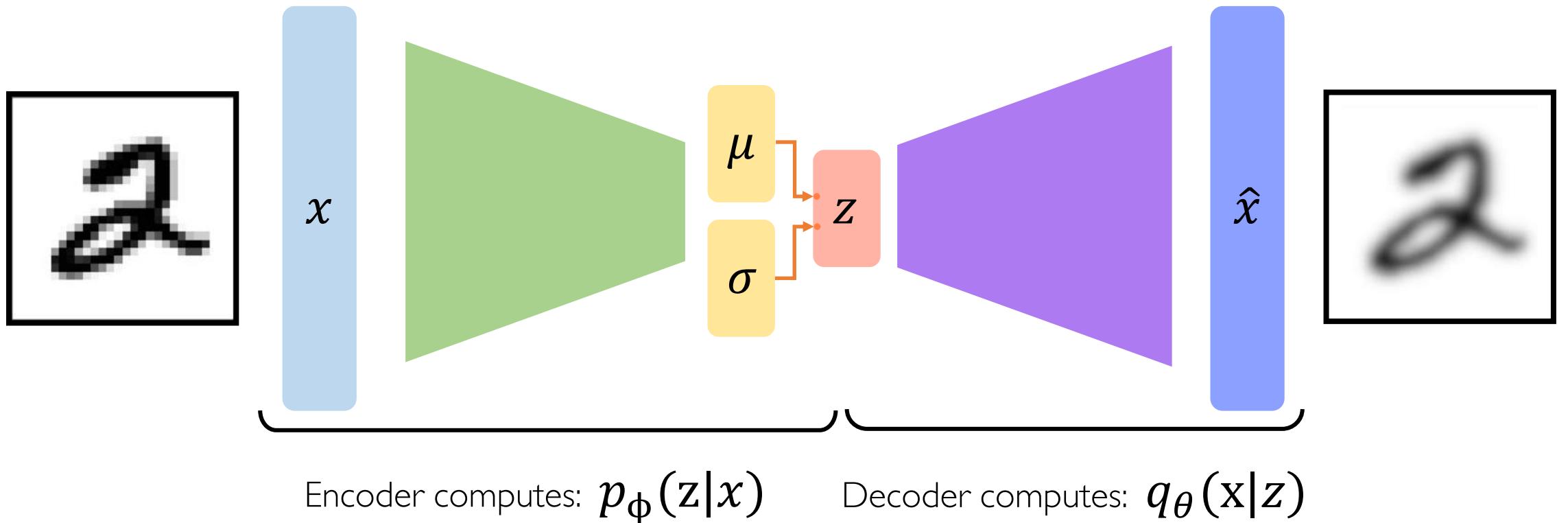
# VAEs: key difference with traditional autoencoder



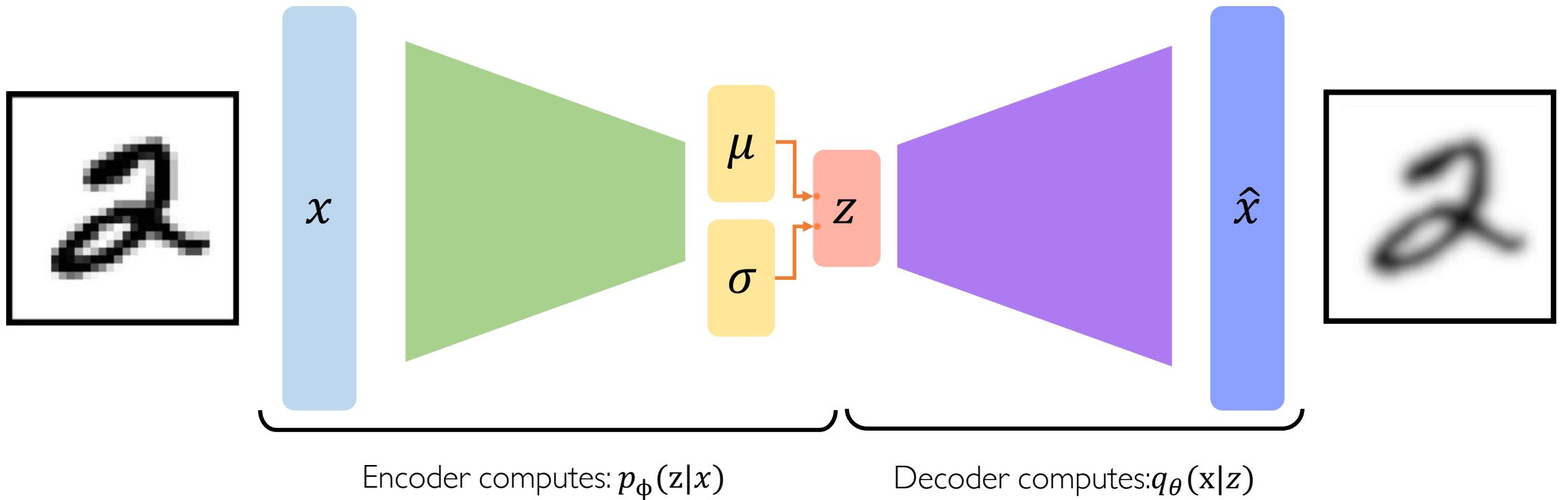
**Variational autoencoders are a probabilistic twist on autoencoders!**

Sample from the mean and standard dev. to compute latent sample

# VAE optimization

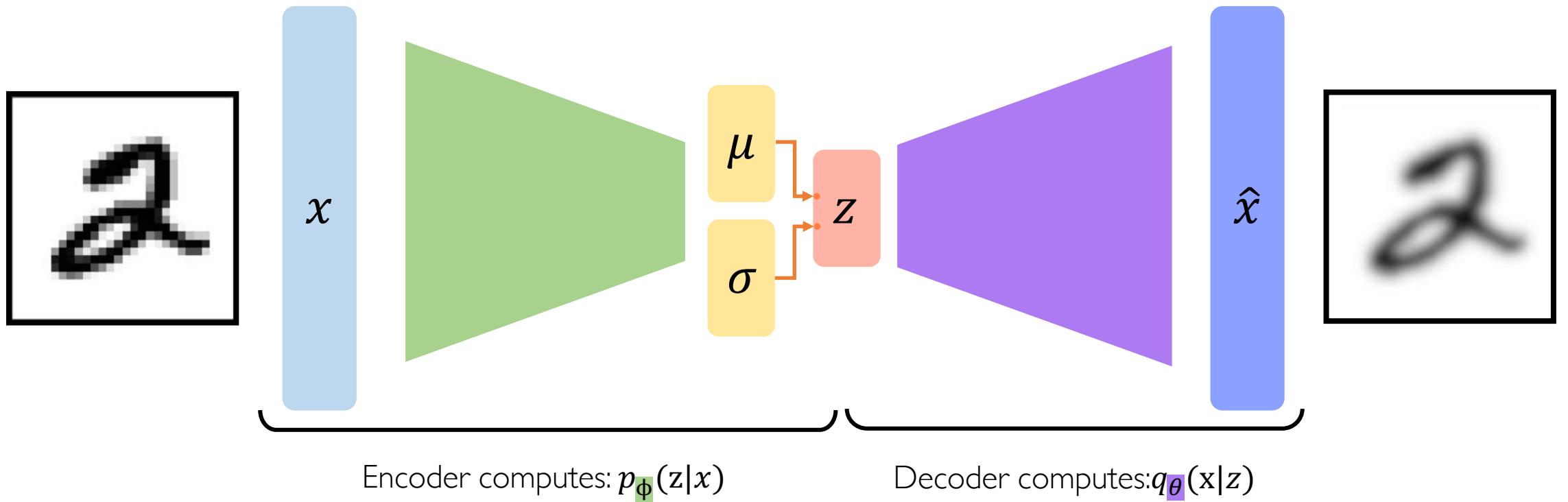


# VAE optimization



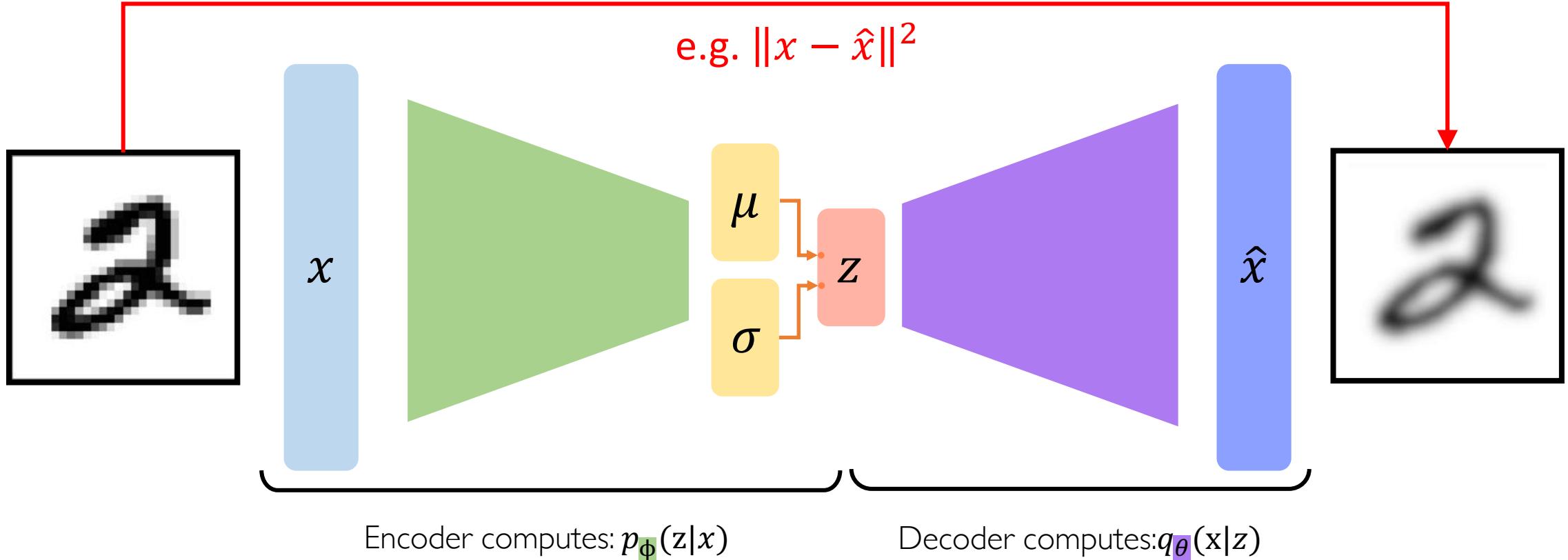
$$\mathcal{L}(\phi, \theta) = (\text{reconstruction loss}) + (\text{regularization term})$$

# VAE optimization



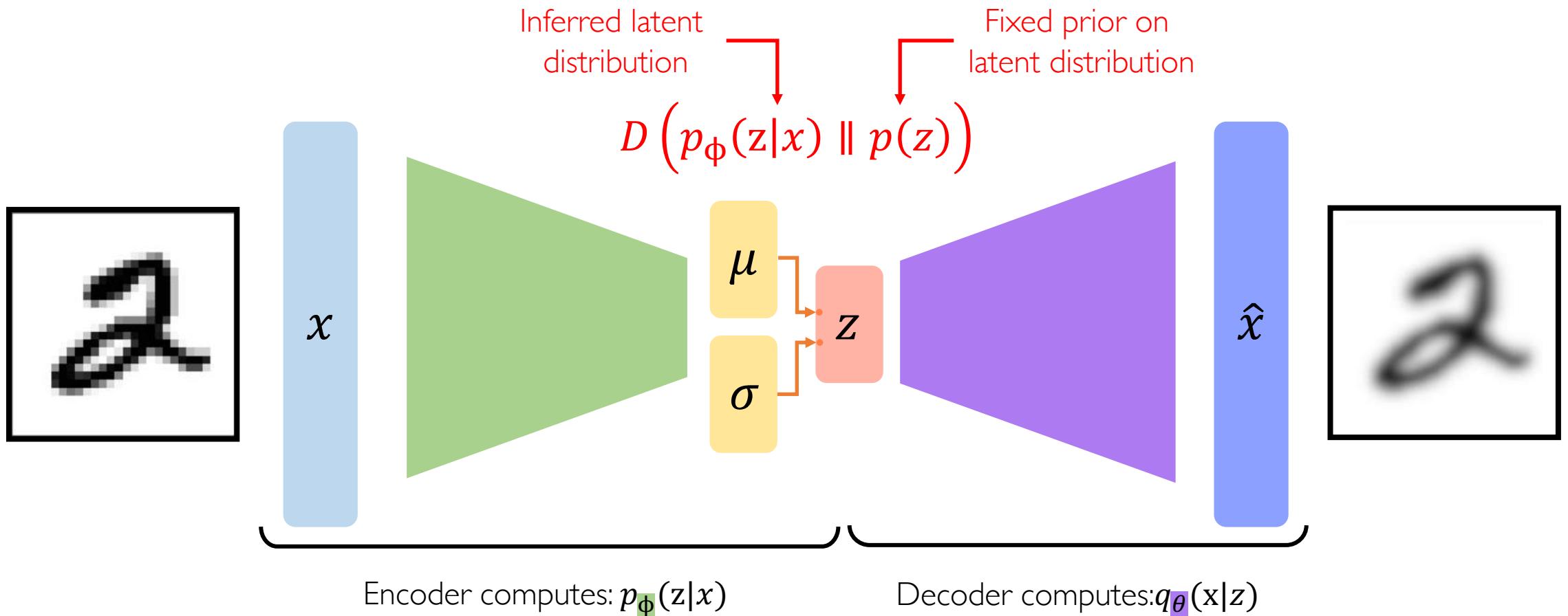
$$\mathcal{L}(\phi, \theta, x) = (\text{reconstruction loss}) + (\text{regularization term})$$

# VAE optimization



$$\mathcal{L}(\phi, \theta, x) = \text{(reconstruction loss)} + \text{(regularization term)}$$

# VAE optimization

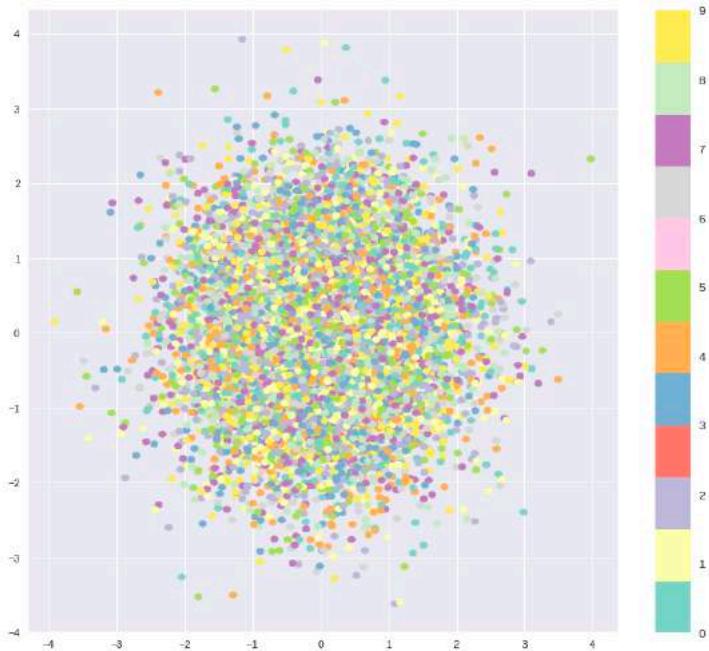


$$\mathcal{L}(\phi, \theta, x) = \text{(reconstruction loss)} + \boxed{\text{(regularization term)}}$$

# Priors on the latent distribution

$$D(p_{\phi}(z|x) \parallel p(z))$$

Inferred latent distribution      ↑      Fixed prior on latent distribution



**Common choice of prior:**

$$p(z) = \mathcal{N}(\mu = 0, \sigma^2 = 1)$$

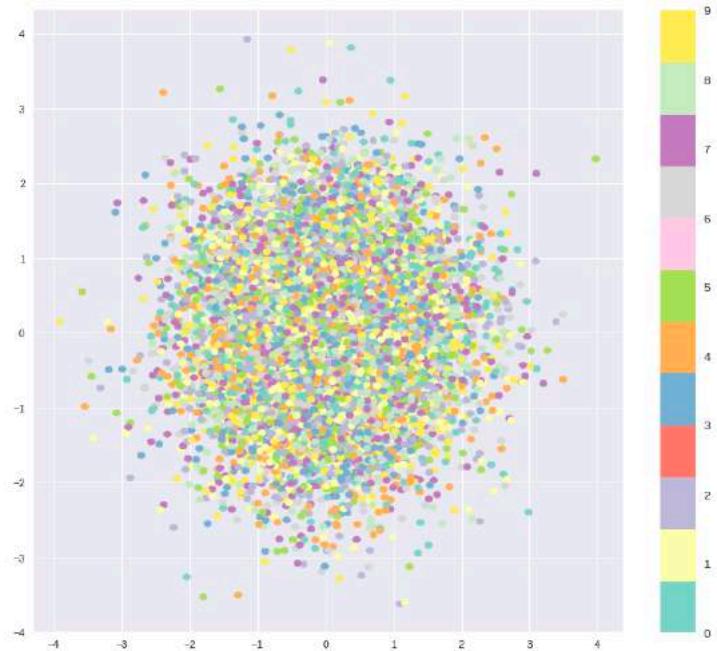
- Encourages encodings to distribute encodings evenly around the center of the latent space
- Penalize the network when it tries to “cheat” by clustering points in specific regions (ie. memorizing the data)

# Priors on the latent distribution

$$D(p_{\phi}(z|x) \parallel p(z))$$

$$= -\frac{1}{2} \sum_{j=0}^{k-1} (\sigma_j + \mu_j^2 - 1 - \log \sigma_j)$$

KL-divergence between  
the two distributions

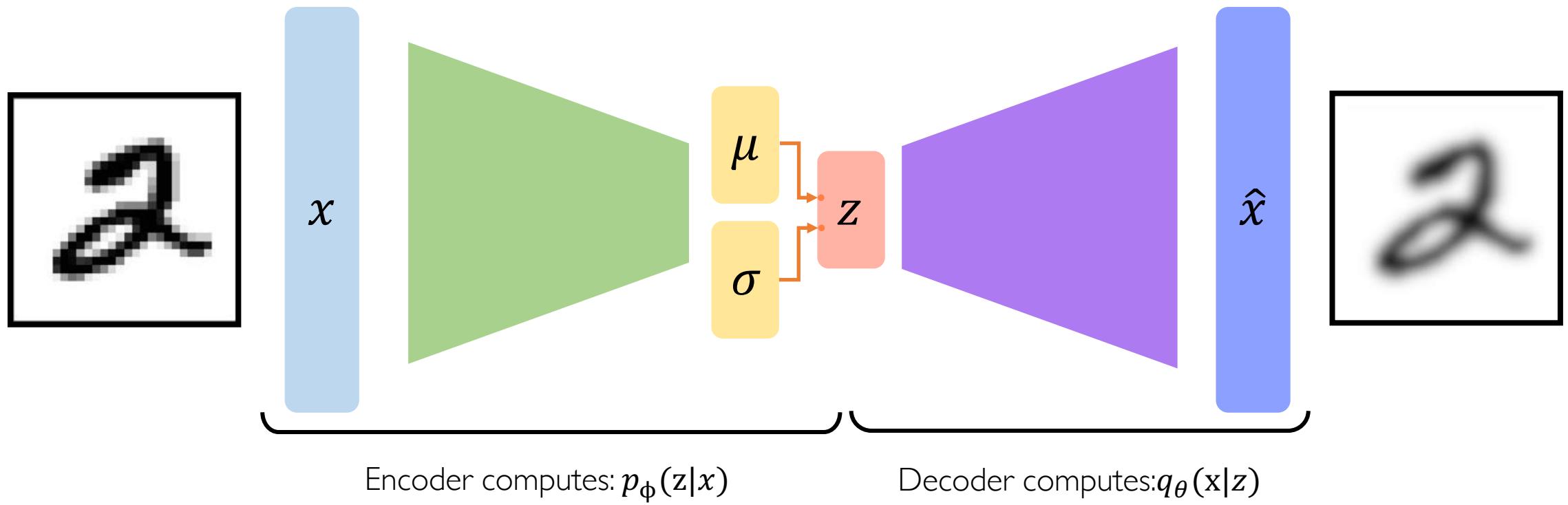


**Common choice of prior:**

$$p(z) = \mathcal{N}(\mu = 0, \sigma^2 = 1)$$

- Encourages encodings to distribute encodings evenly around the center of the latent space
- Penalize the network when it tries to “cheat” by clustering points in specific regions (ie. memorizing the data)

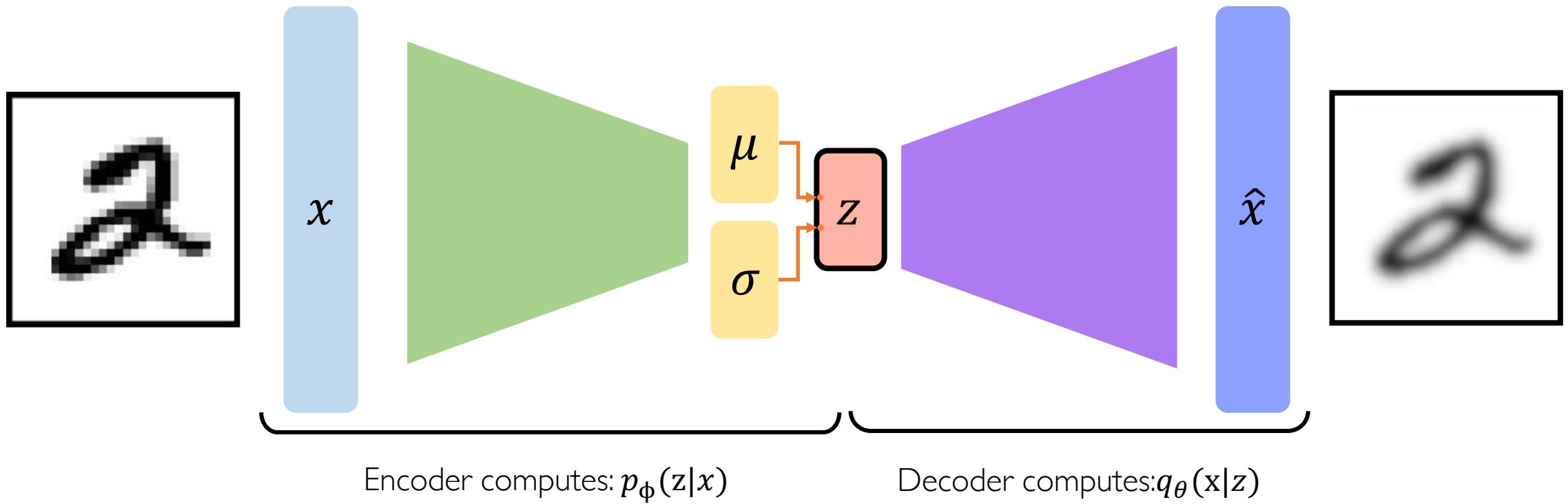
# VAEs computation graph



$$\mathcal{L}(\phi, \theta, x) = (\text{reconstruction loss}) + (\text{regularization term})$$

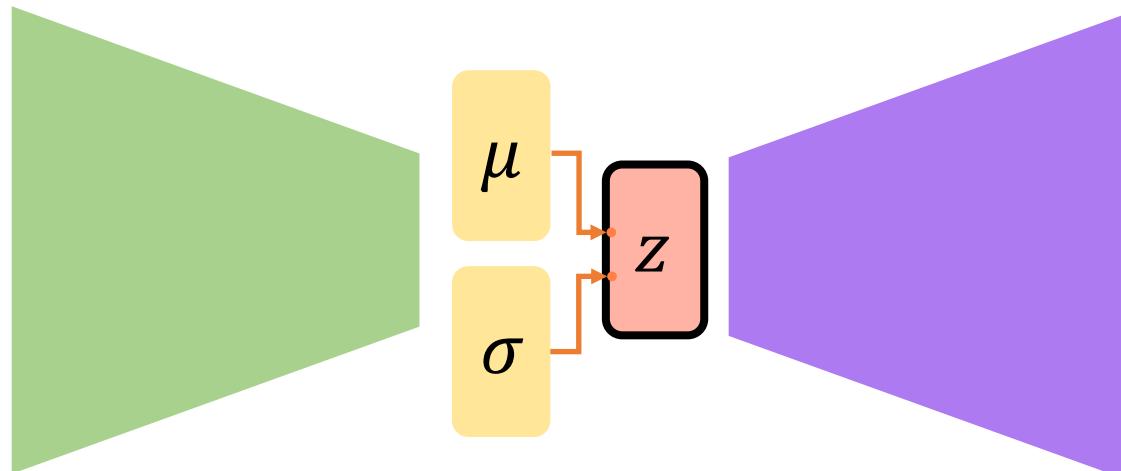
# VAEs computation graph

**Problem:** We cannot backpropagate gradients through sampling layers!



$$\mathcal{L}(\phi, \theta, x) = (\text{reconstruction loss}) + (\text{regularization term})$$

# Reparametrizing the sampling layer



**Key Idea:**

$$z \sim \mathcal{N}(\mu, \sigma^2)$$

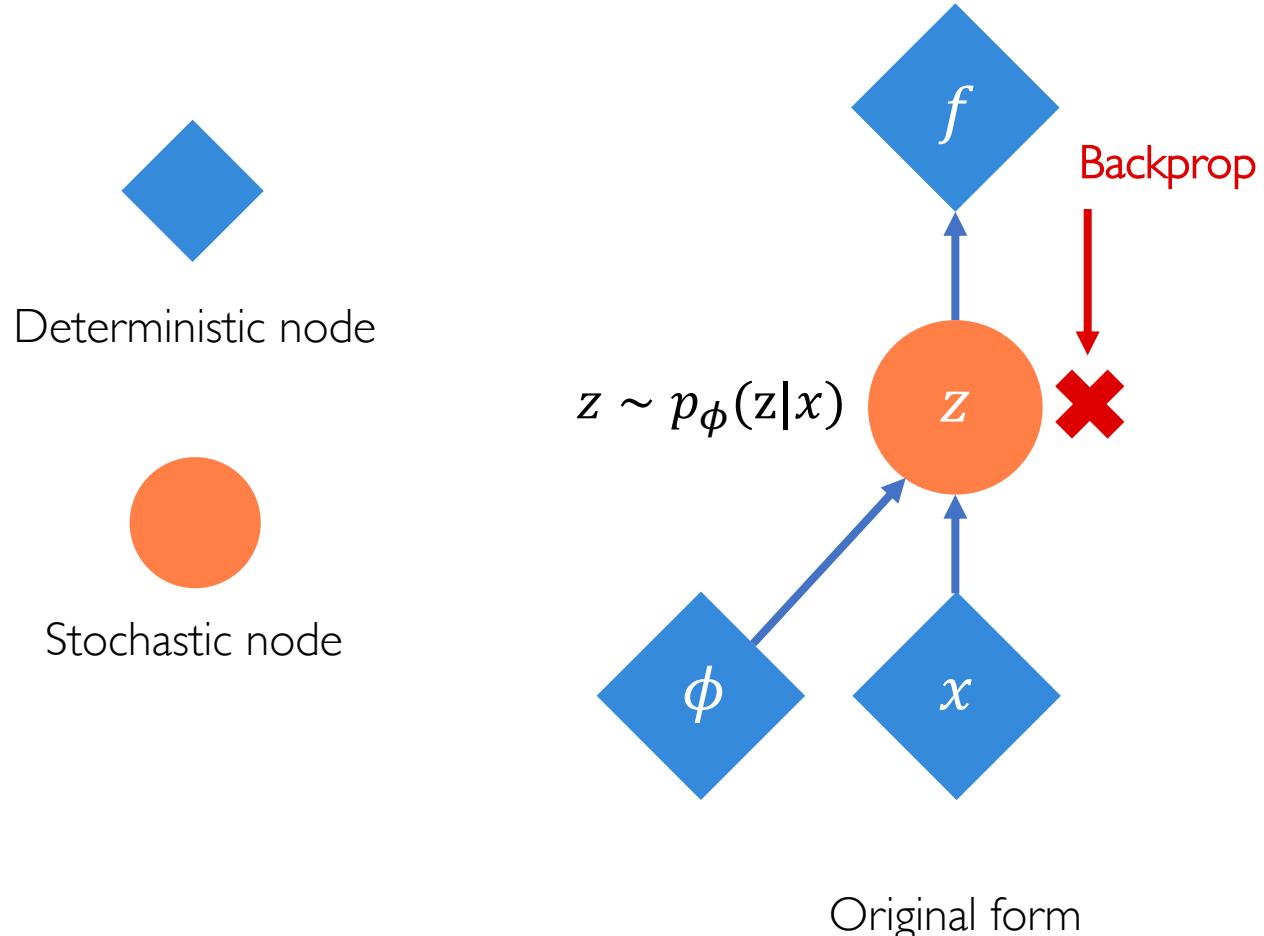
Consider the sampled latent vector as a sum of

- a fixed  $\mu$  vector,
- and fixed  $\sigma$  vector, scaled by random constants drawn from the prior distribution

$$\Rightarrow z = \mu + \sigma \odot \varepsilon$$

where  $\varepsilon \sim \mathcal{N}(0,1)$

# Reparametrizing the sampling layer



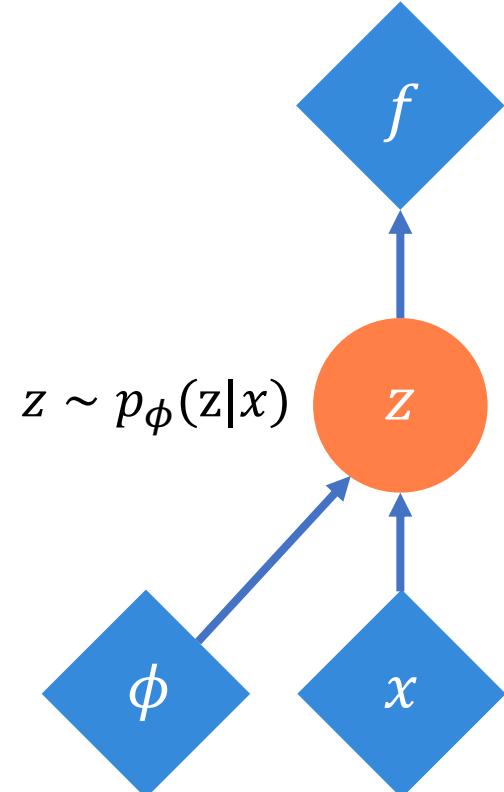
# Reparametrizing the sampling layer



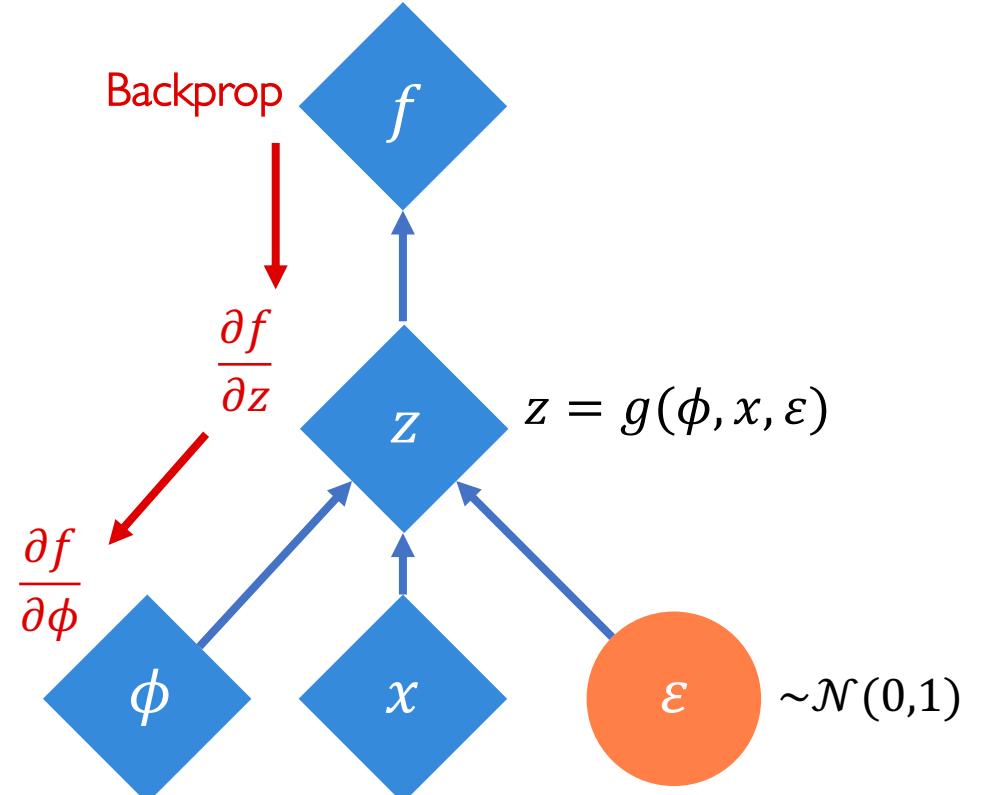
Deterministic node



Stochastic node



Original form



Reparametrized form

# VAEs: Latent perturbation

Slowly increase or decrease a **single latent variable**

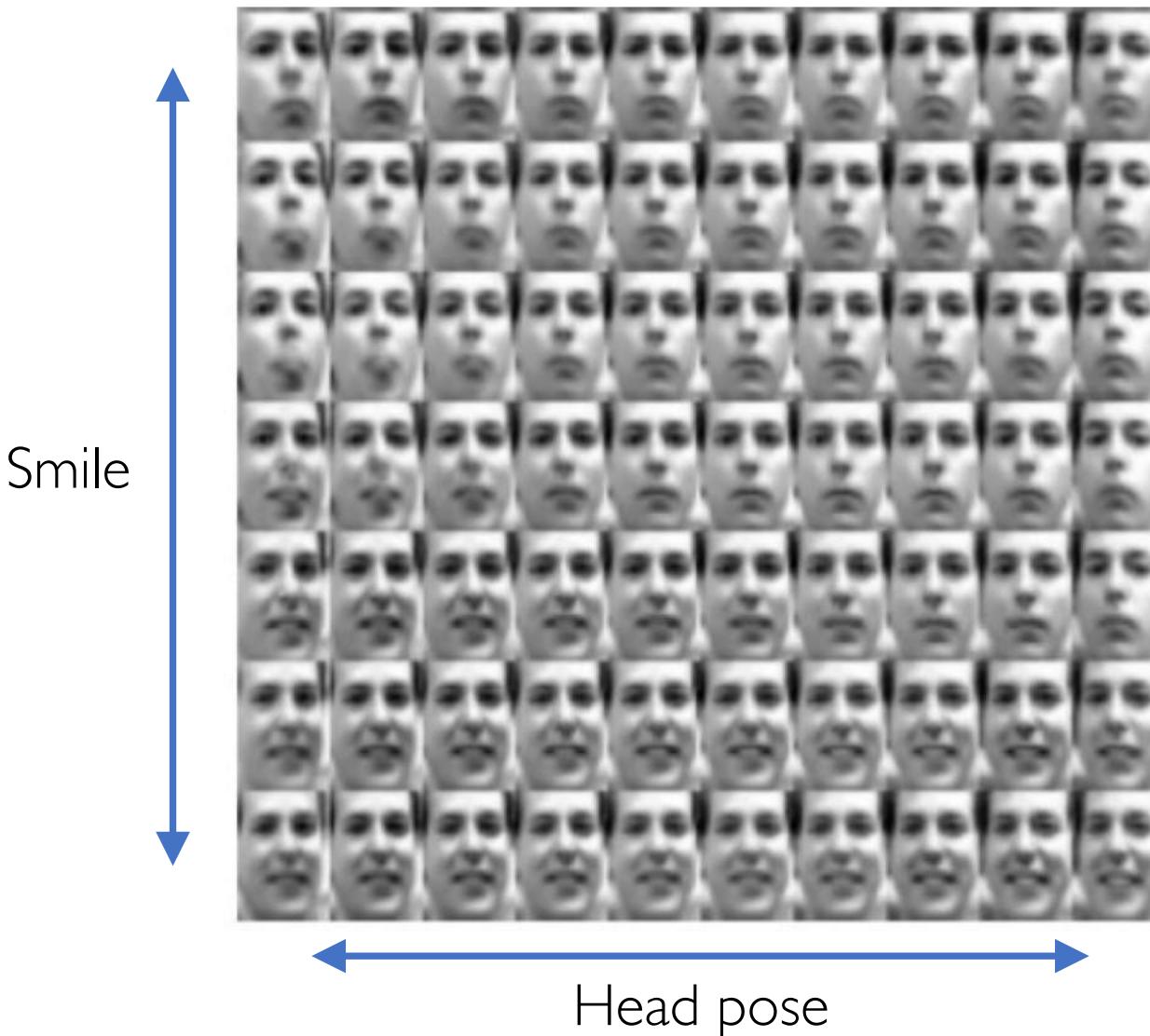
Keep all other variables fixed



Head pose

Different dimensions of  $z$  encodes **different interpretable latent features**

# VAEs: Latent perturbation

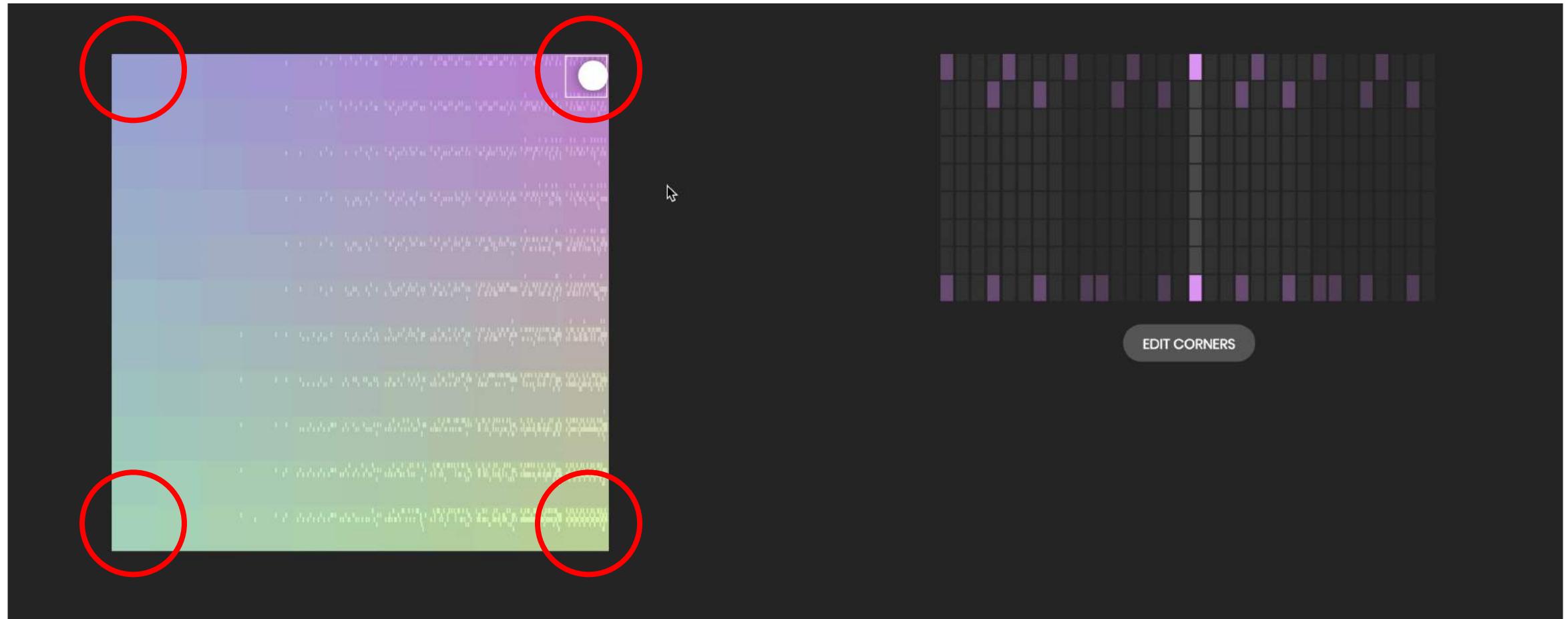


Ideally, we want latent variables that are uncorrelated with each other

Enforce diagonal prior on the latent variables to encourage independence

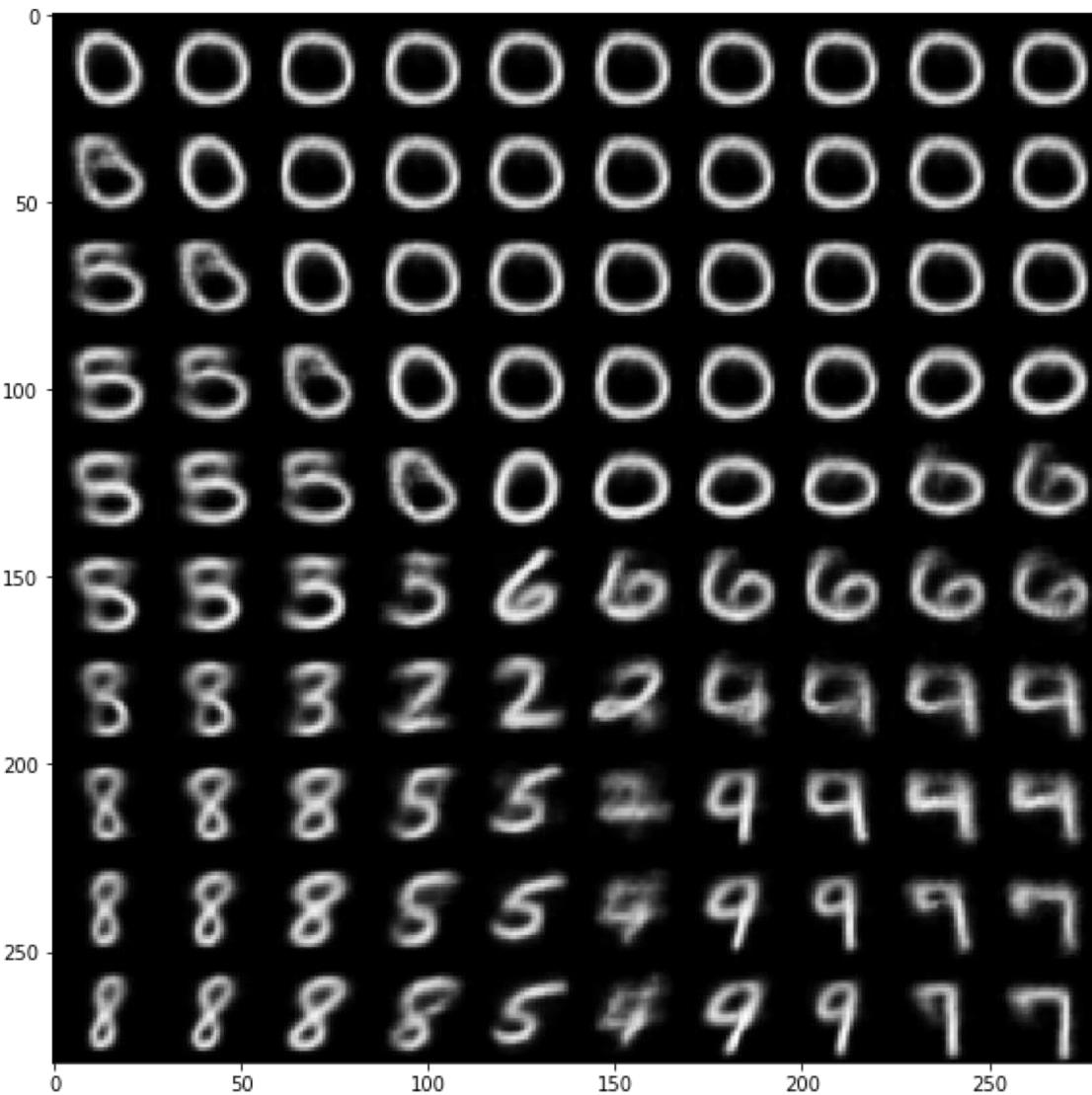
**Disentanglement**

# VAEs: Latent perturbation



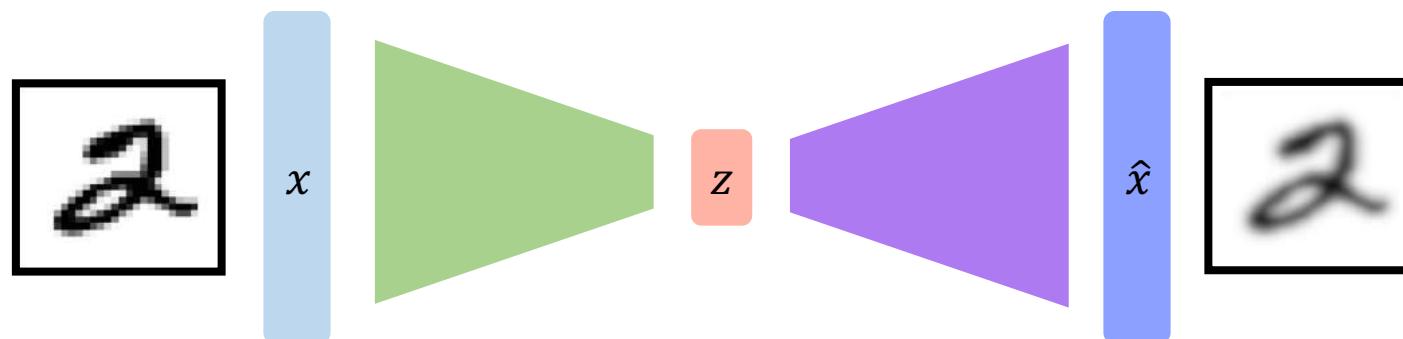
Google BeatBlender

# VAEs: Latent perturbation



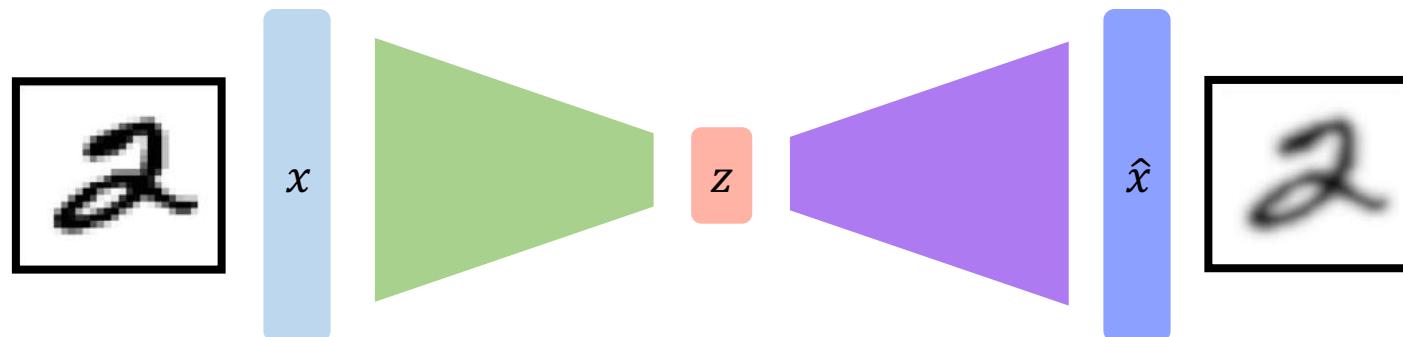
# VAE summary

- I. Compress representation of world to something we can use to learn



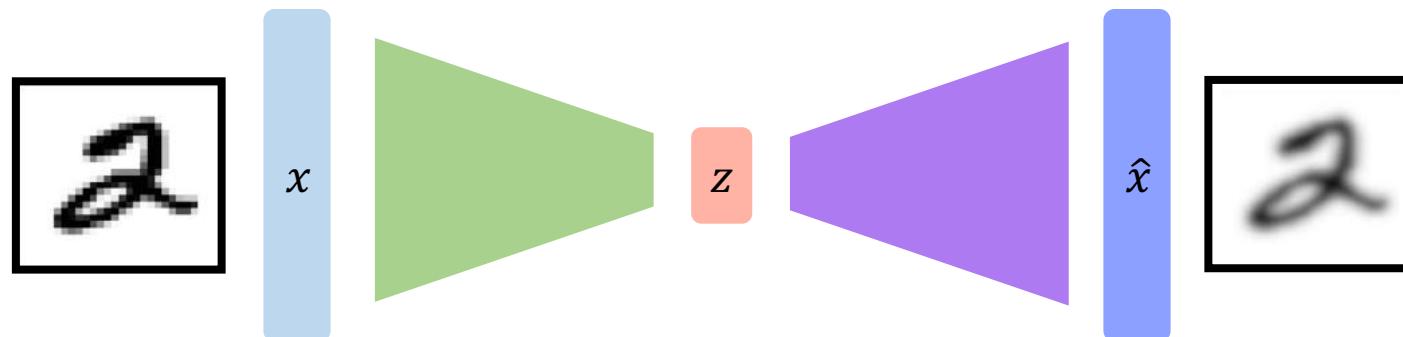
# VAE summary

1. Compress representation of world to something we can use to learn
2. Reconstruction allows for unsupervised learning (no labels!)



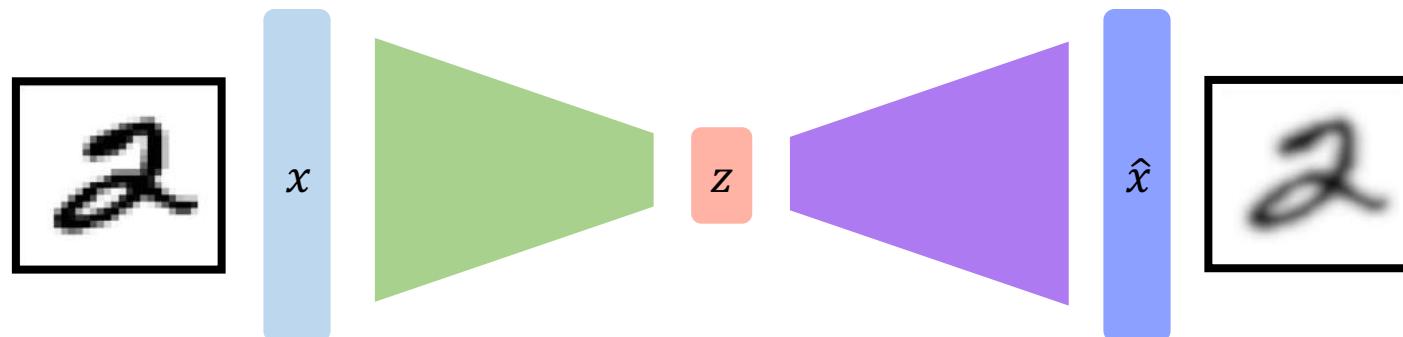
# VAE summary

1. Compress representation of world to something we can use to learn
2. Reconstruction allows for unsupervised learning (no labels!)
3. Reparameterization trick to train end-to-end



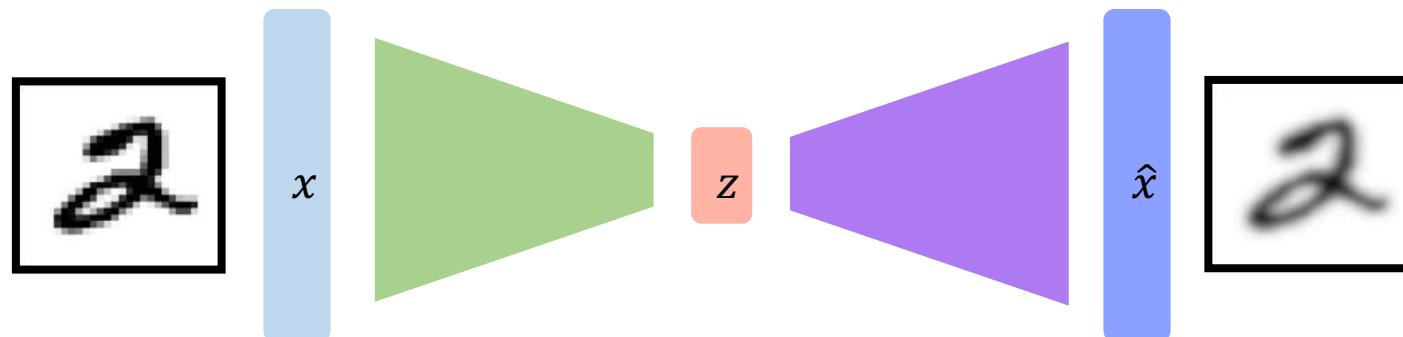
# VAE summary

1. Compress representation of world to something we can use to learn
2. Reconstruction allows for unsupervised learning (no labels!)
3. Reparameterization trick to train end-to-end
4. Interpret hidden latent variables using perturbation



# VAE summary

1. Compress representation of world to something we can use to learn
2. Reconstruction allows for unsupervised learning (no labels!)
3. Reparameterization trick to train end-to-end
4. Interpret hidden latent variables using perturbation
5. Generating new examples



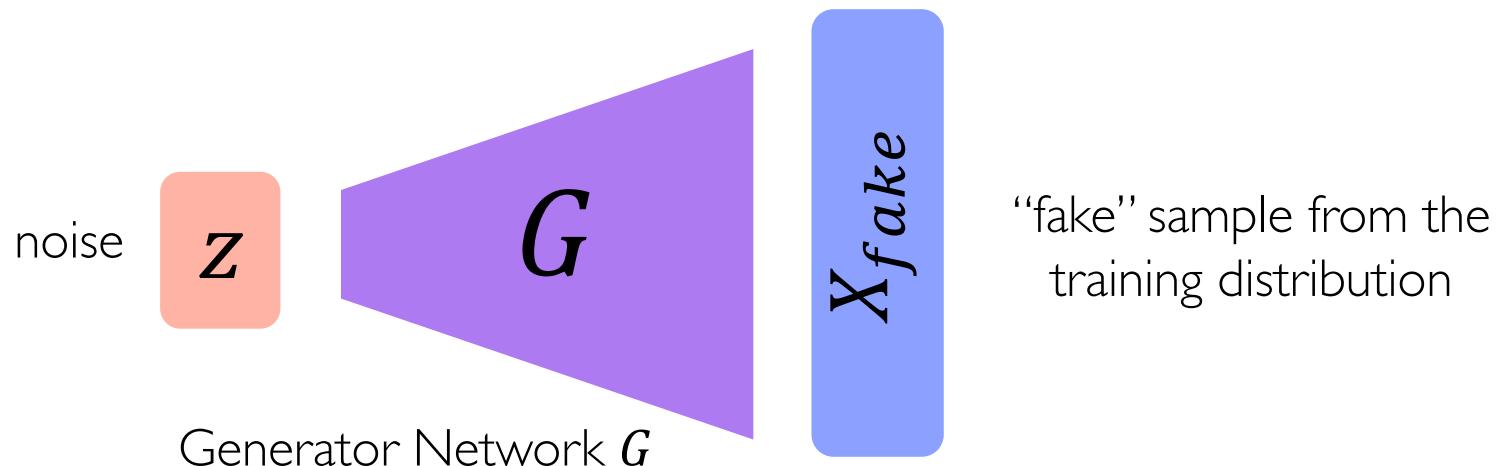
# Generative Adversarial Networks (GANs)

# What if we just want to sample?

**Idea:** don't explicitly model density, and instead just sample to generate new instances.

**Problem:** want to sample from complex distribution – can't do this directly!

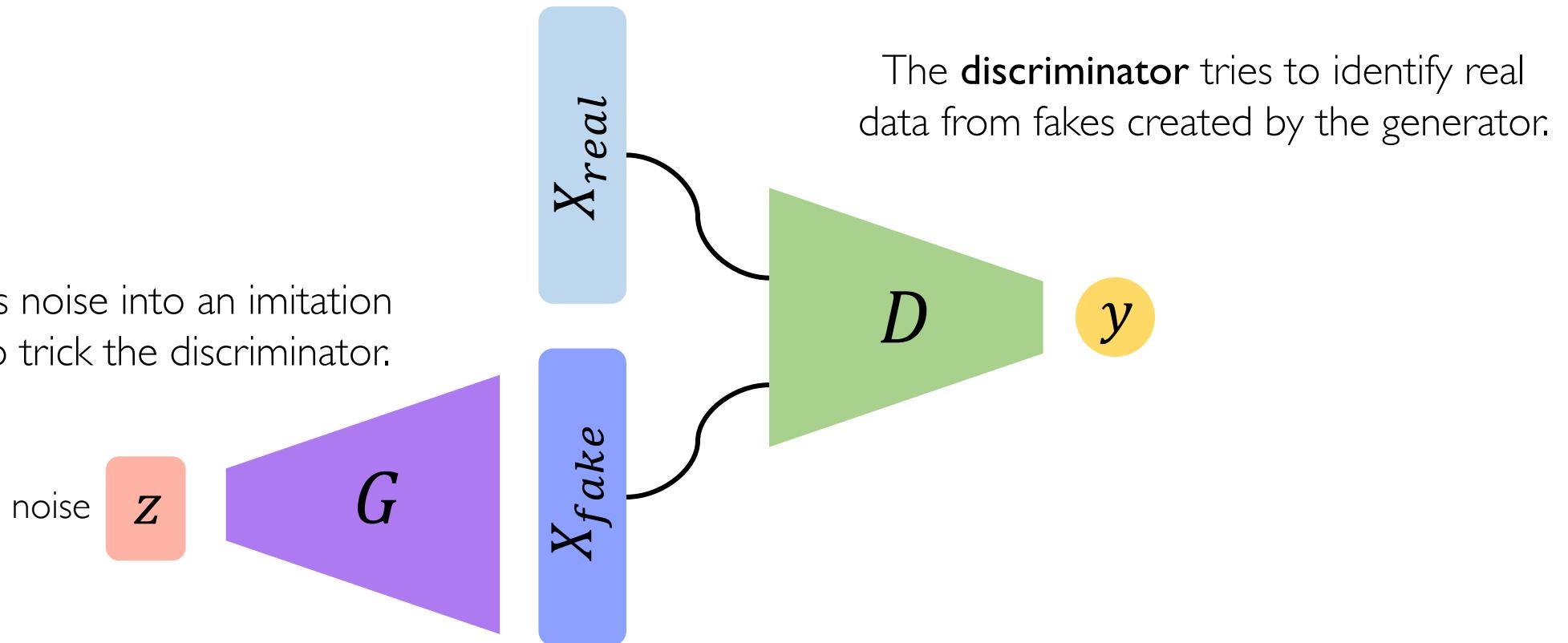
**Solution:** sample from something simple (noise), learn a transformation to the training distribution.



# Generative Adversarial Networks (GANs)

Generative Adversarial Networks (GANs) are a way to make a generative model by having two neural networks compete with each other.

The **generator** turns noise into an imitation of the data to try to trick the discriminator.



# Intuition behind GANs

**Generator** starts from noise to try to create an imitation of the data.

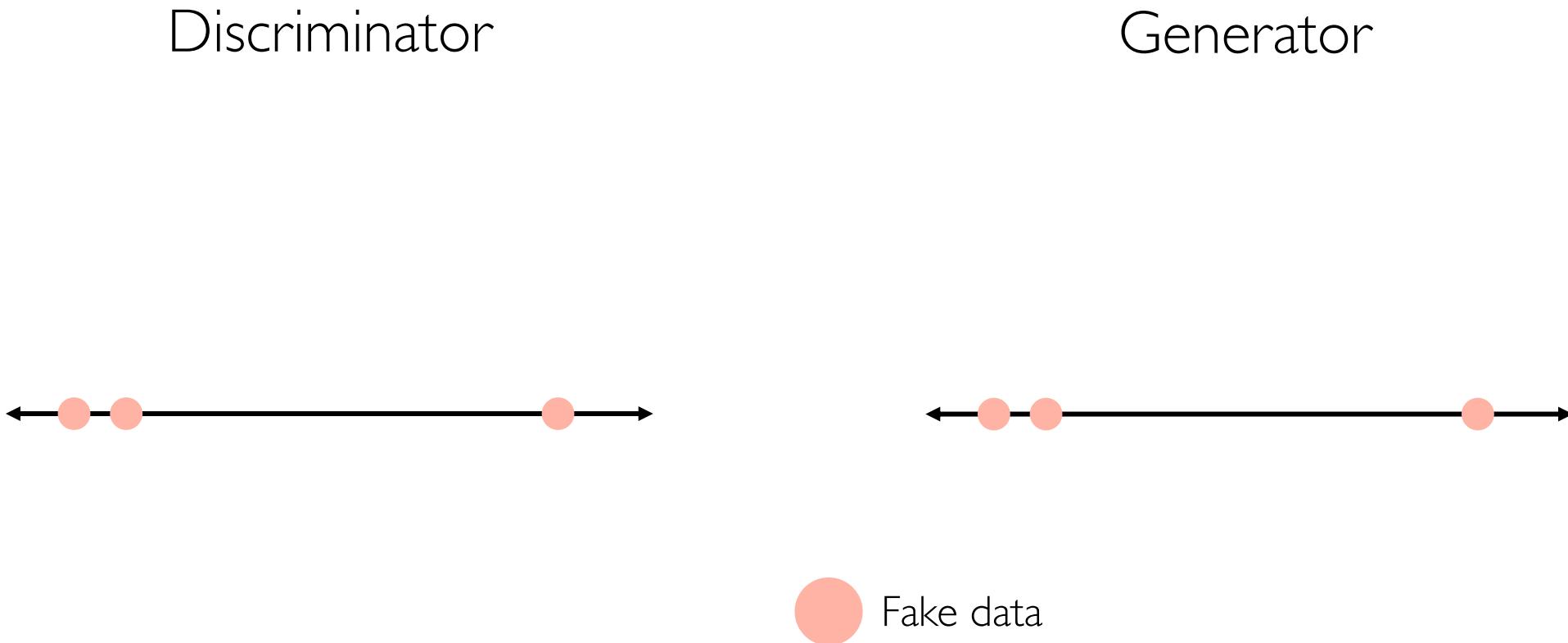
Generator



Fake data

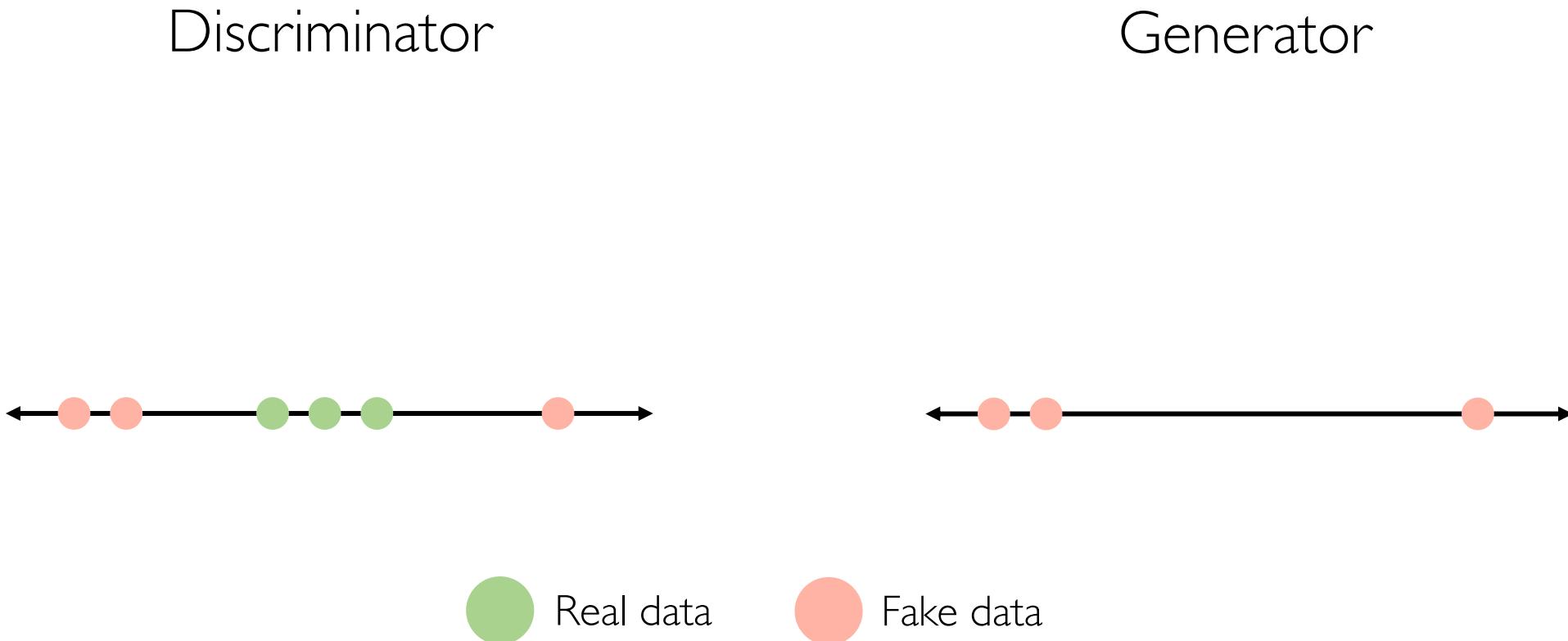
# Intuition behind GANs

**Discriminator** looks at both real data and fake data created by the generator.



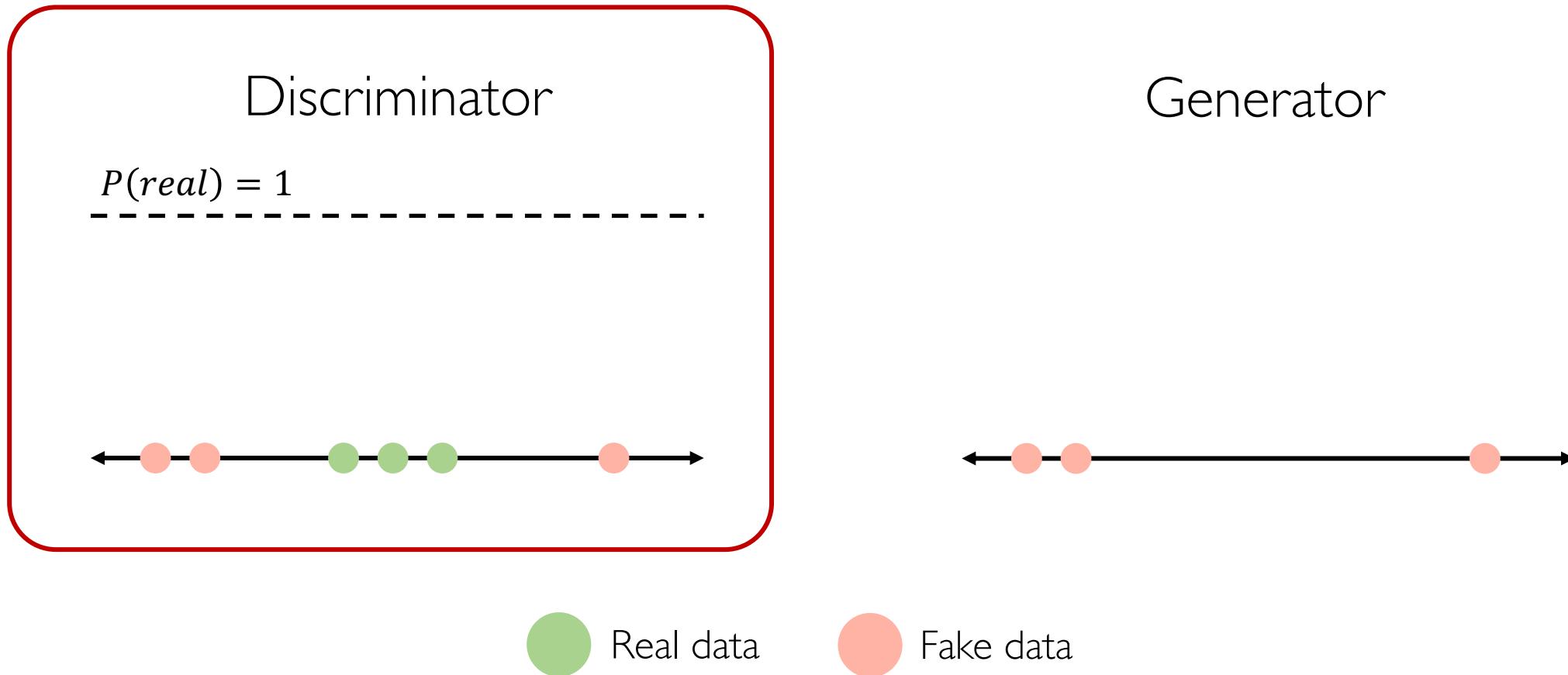
# Intuition behind GANs

**Discriminator** looks at both real data and fake data created by the generator.



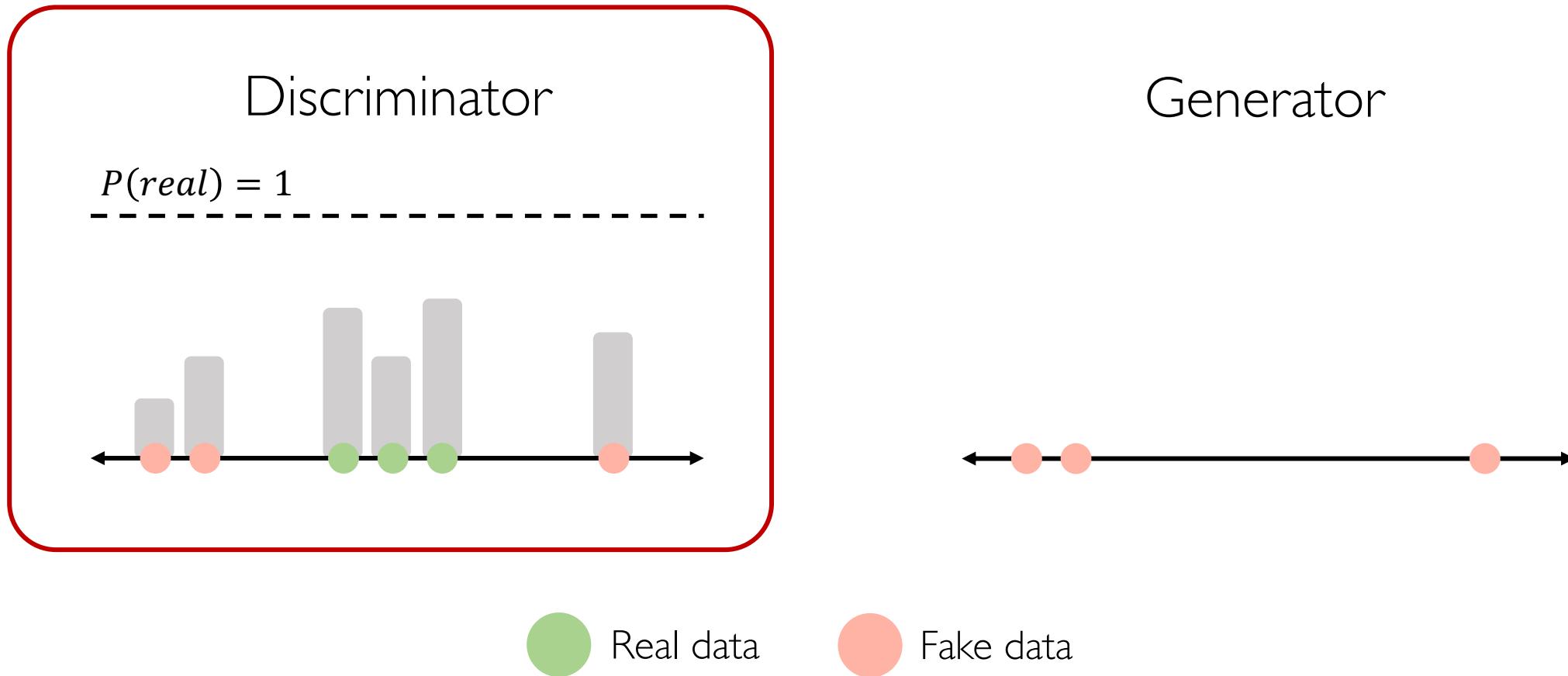
# Intuition behind GANs

**Discriminator** tries to predict what's real and what's fake.



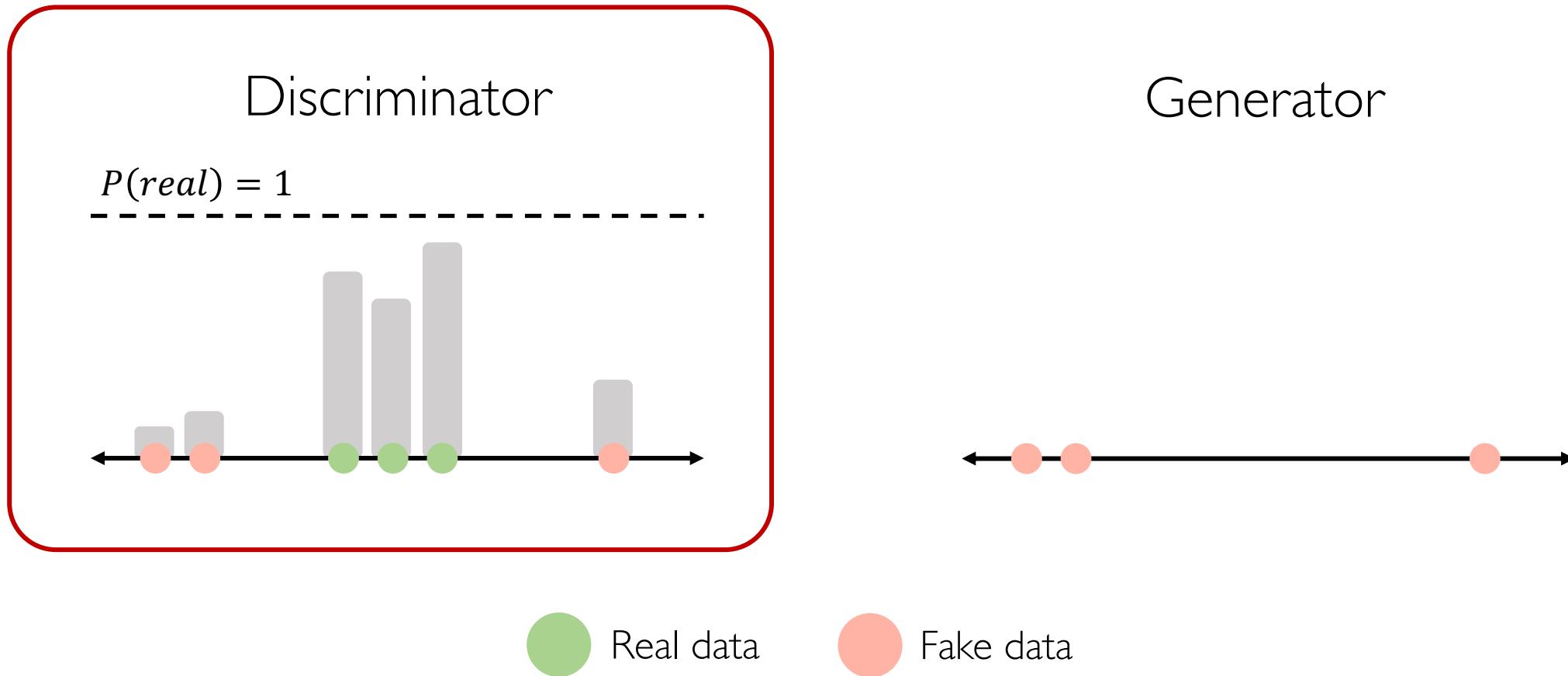
# Intuition behind GANs

**Discriminator** tries to predict what's real and what's fake.



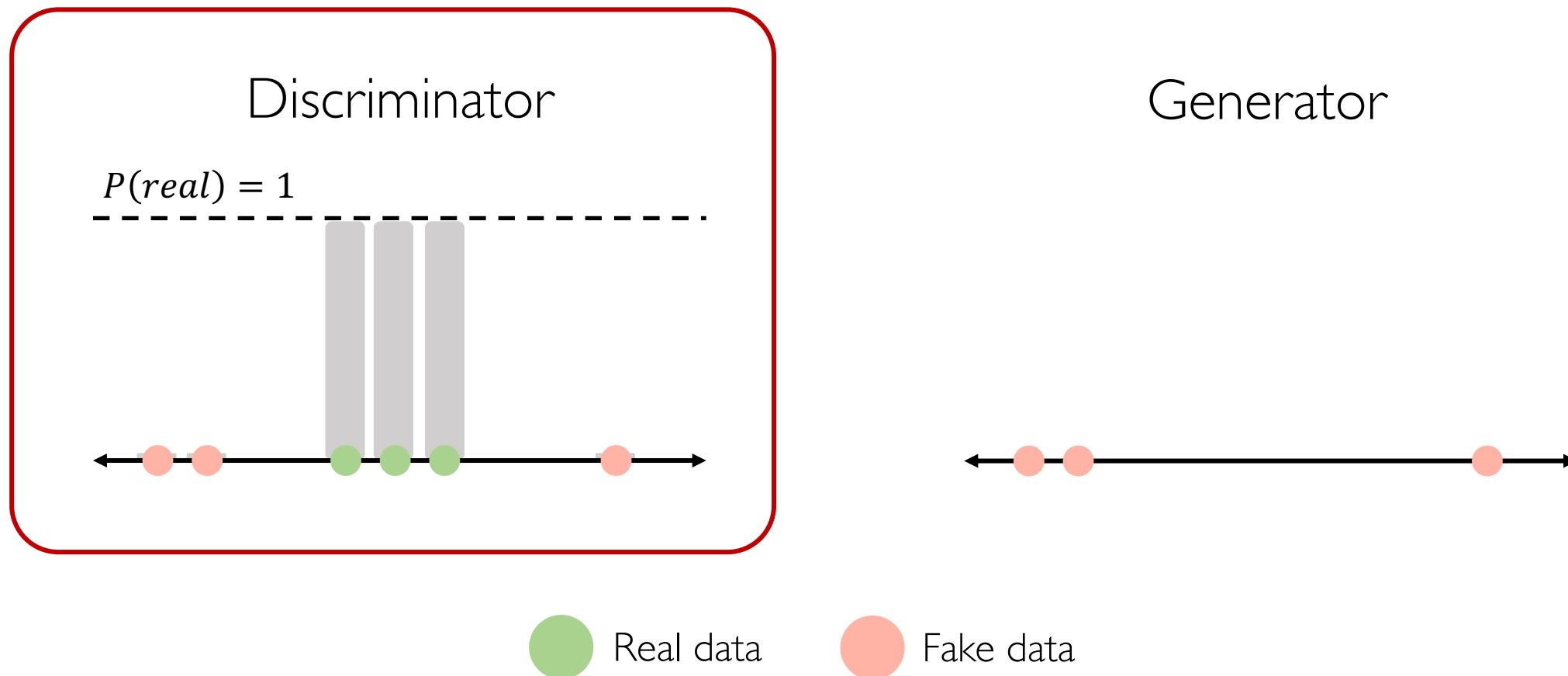
# Intuition behind GANs

**Discriminator** tries to predict what's real and what's fake.



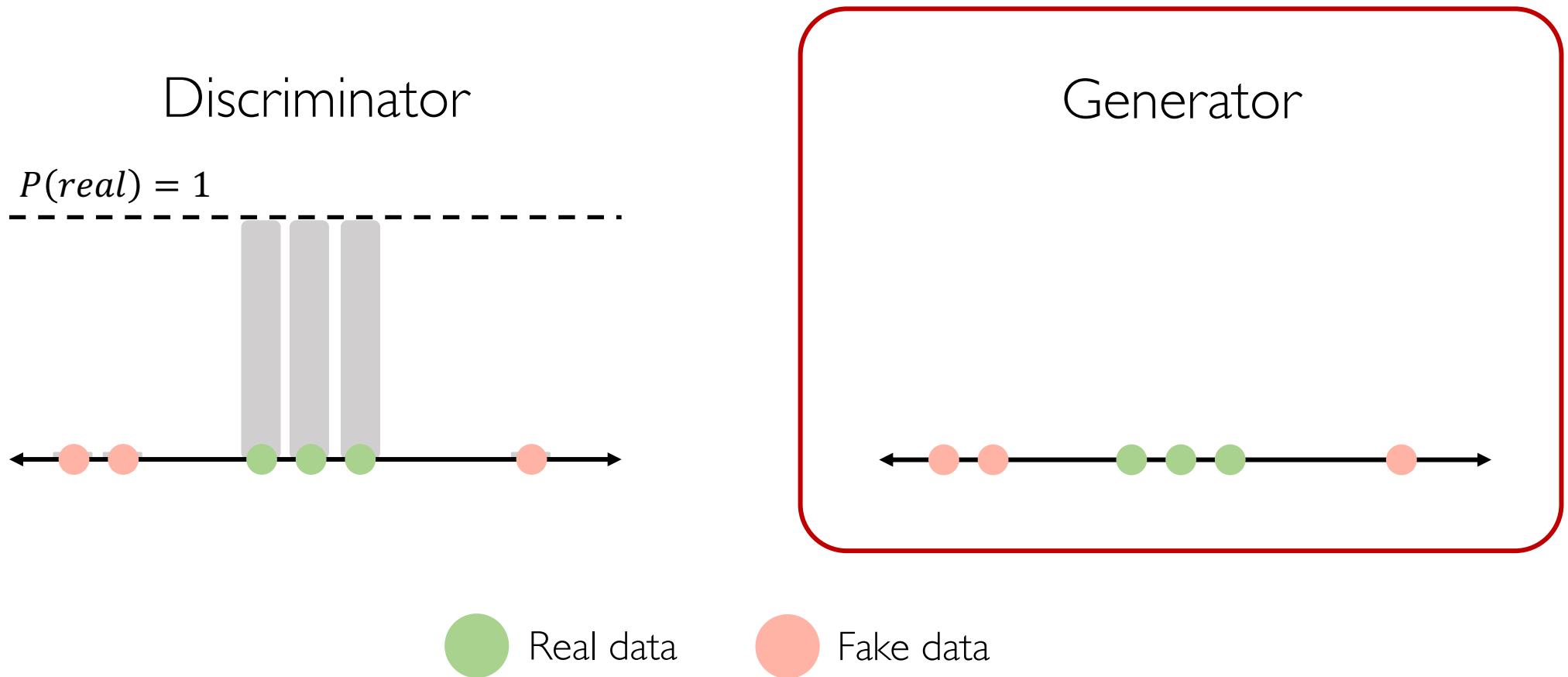
# Intuition behind GANs

**Discriminator** tries to predict what's real and what's fake.



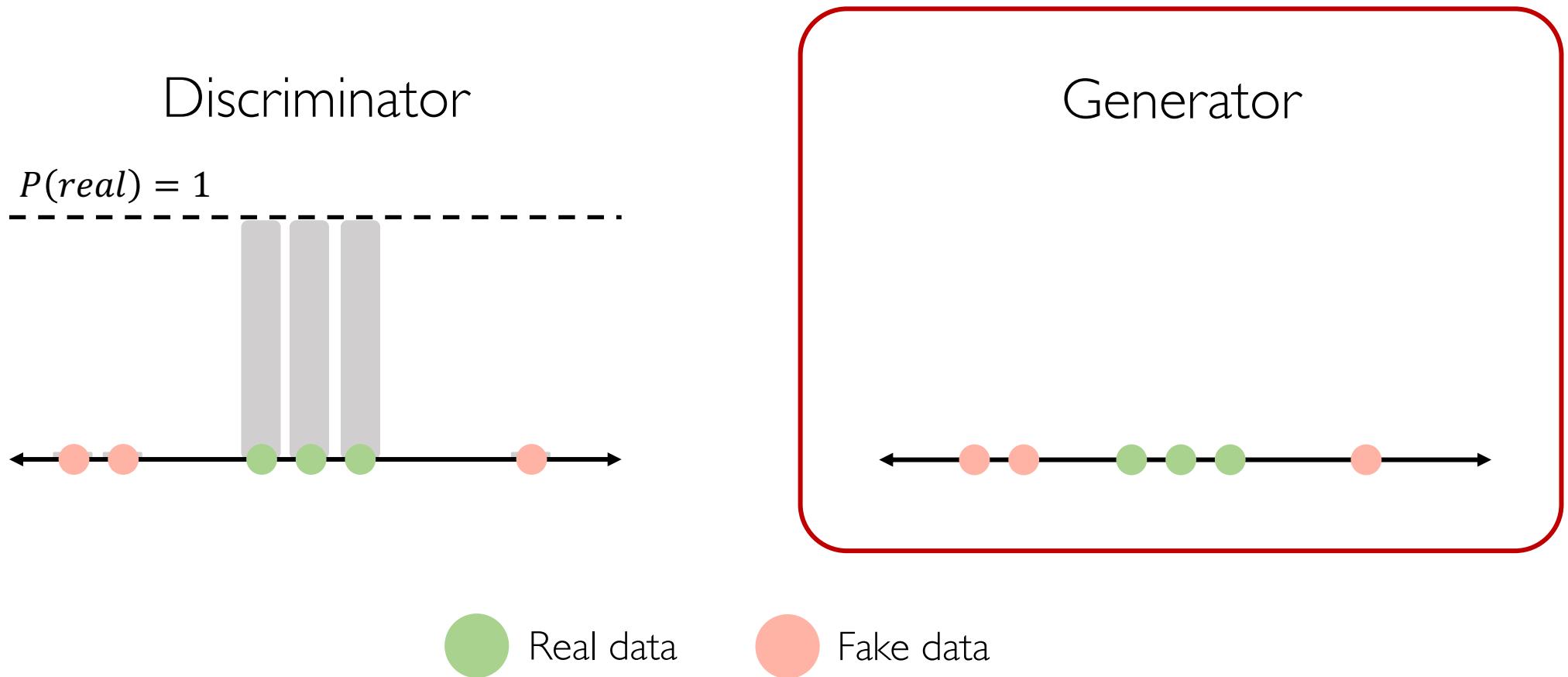
# Intuition behind GANs

**Generator** tries to improve its imitation of the data.



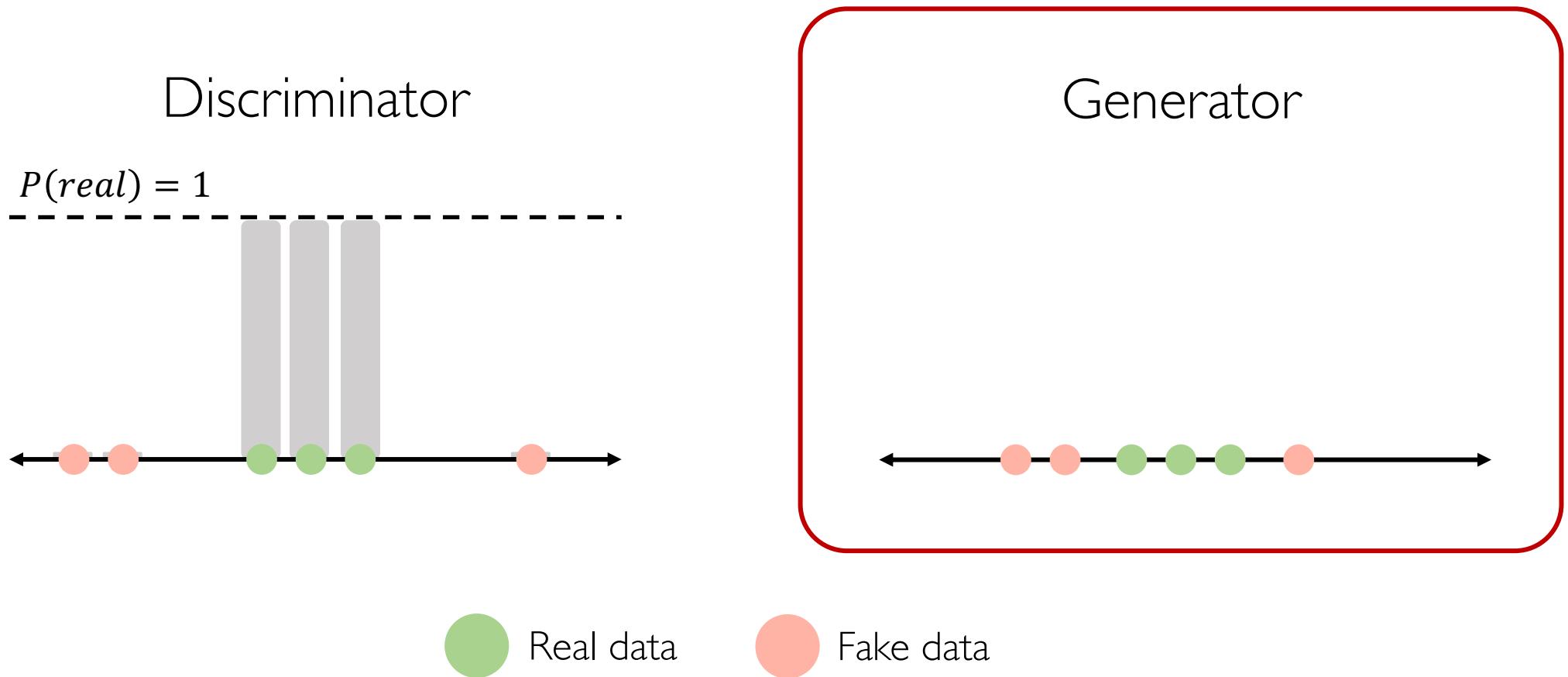
# Intuition behind GANs

**Generator** tries to improve its imitation of the data.



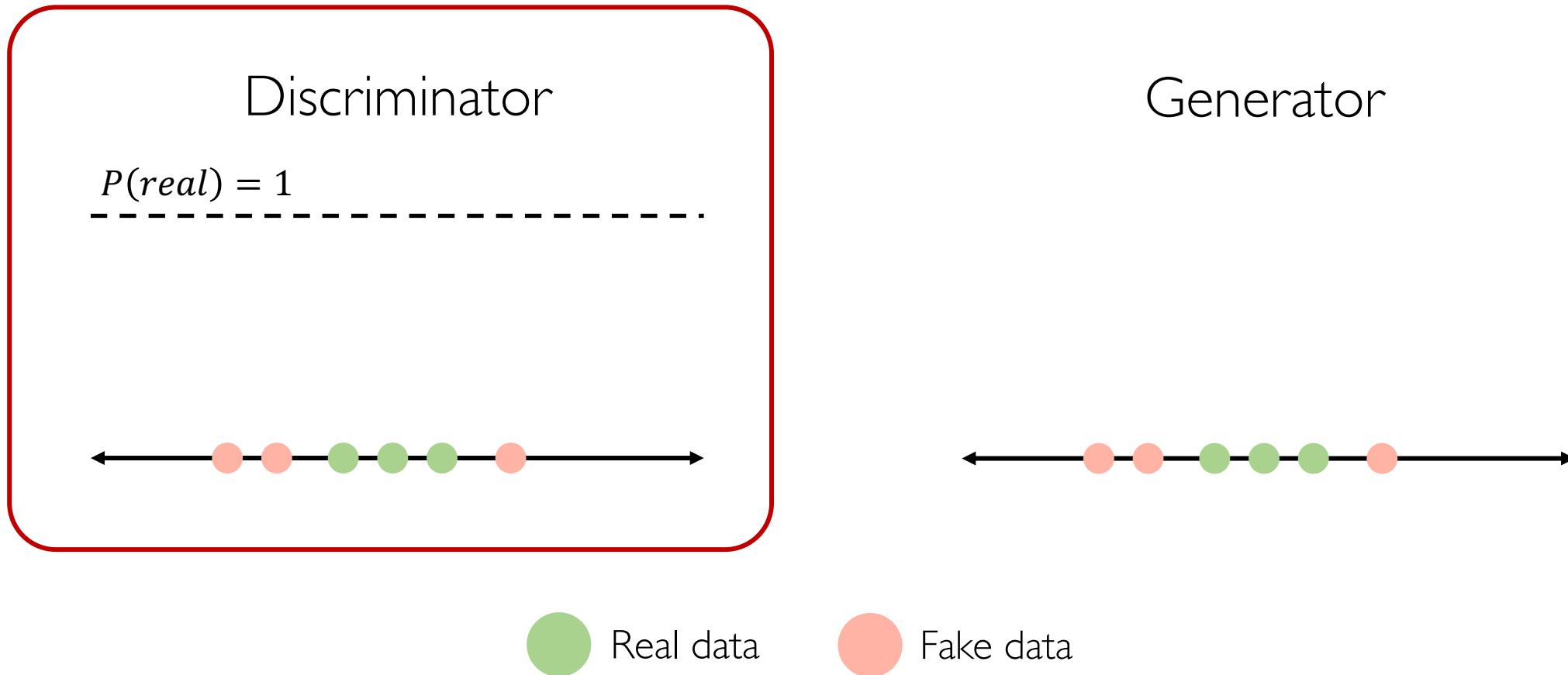
# Intuition behind GANs

**Generator** tries to improve its imitation of the data.



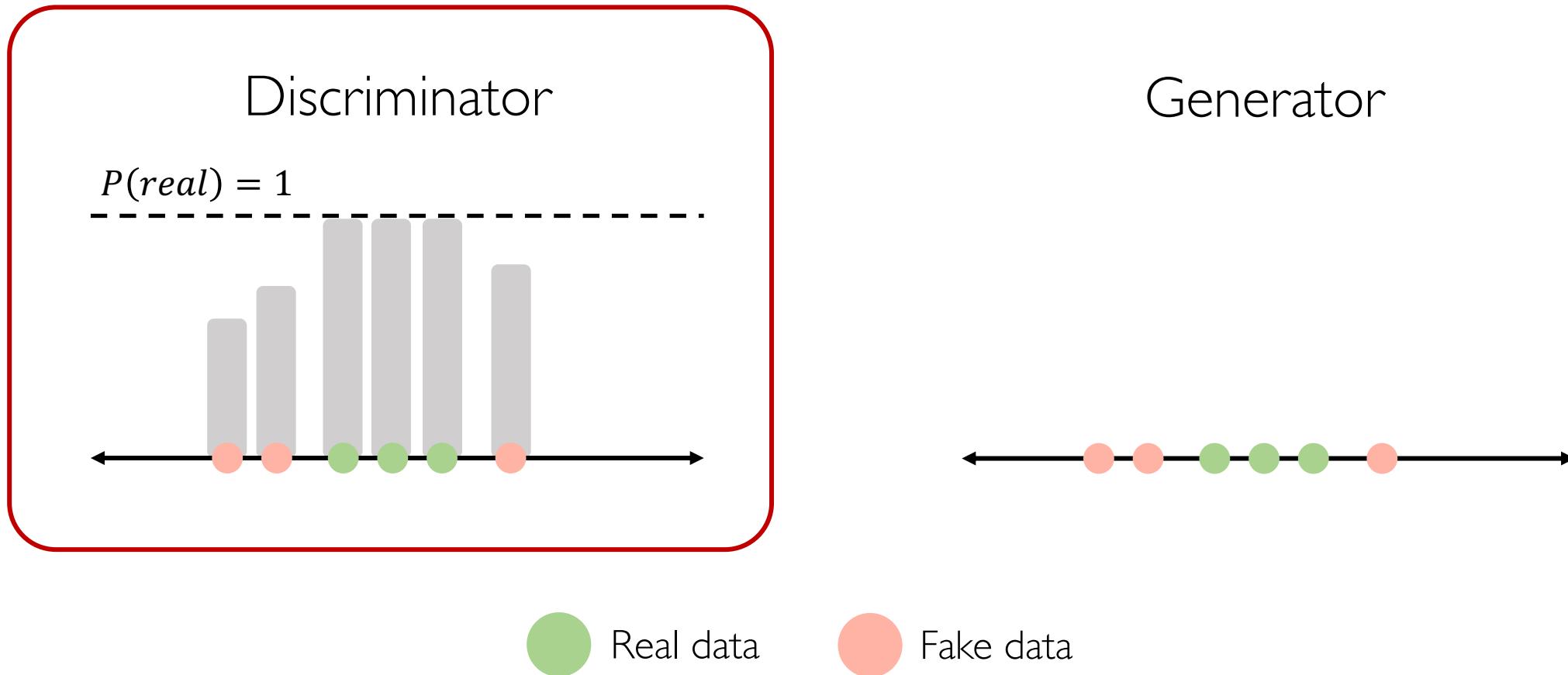
# Intuition behind GANs

**Discriminator** tries to predict what's real and what's fake.



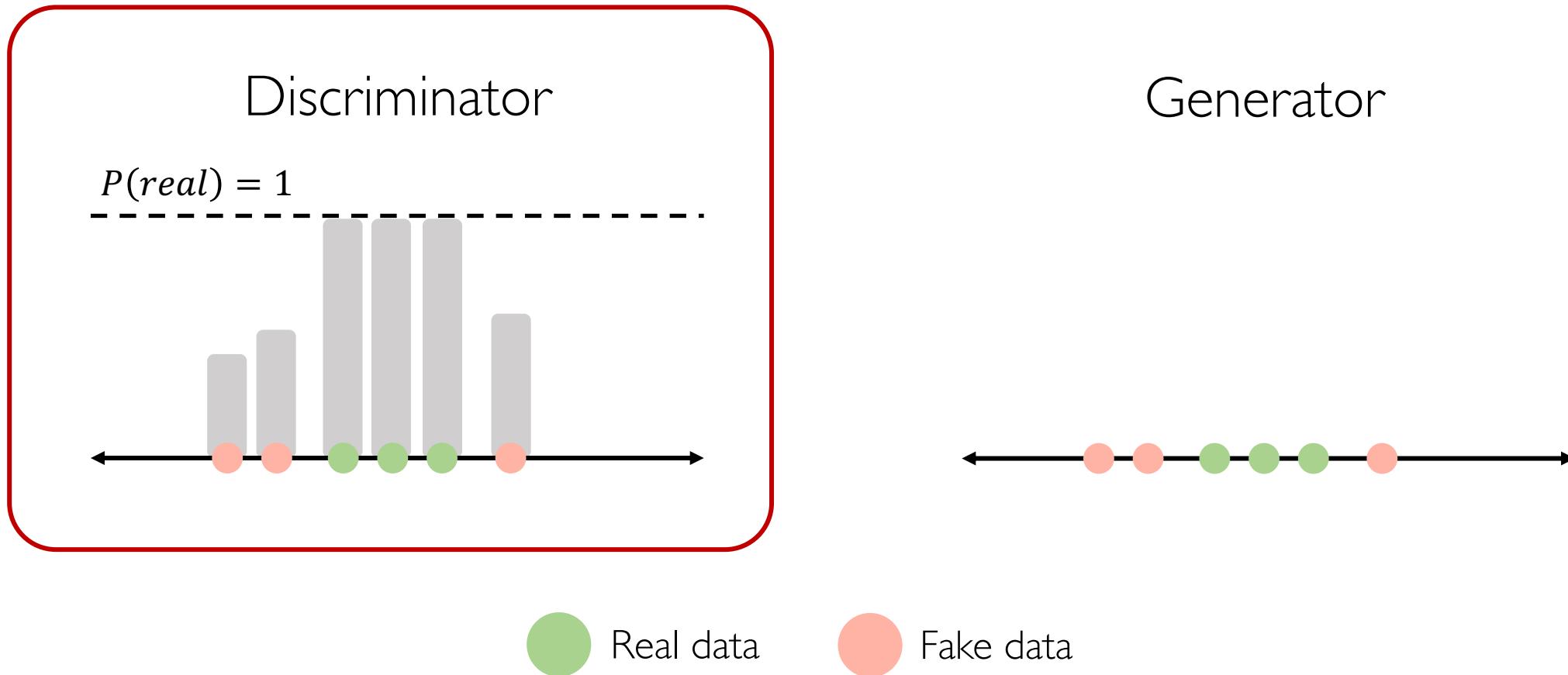
# Intuition behind GANs

**Discriminator** tries to predict what's real and what's fake.



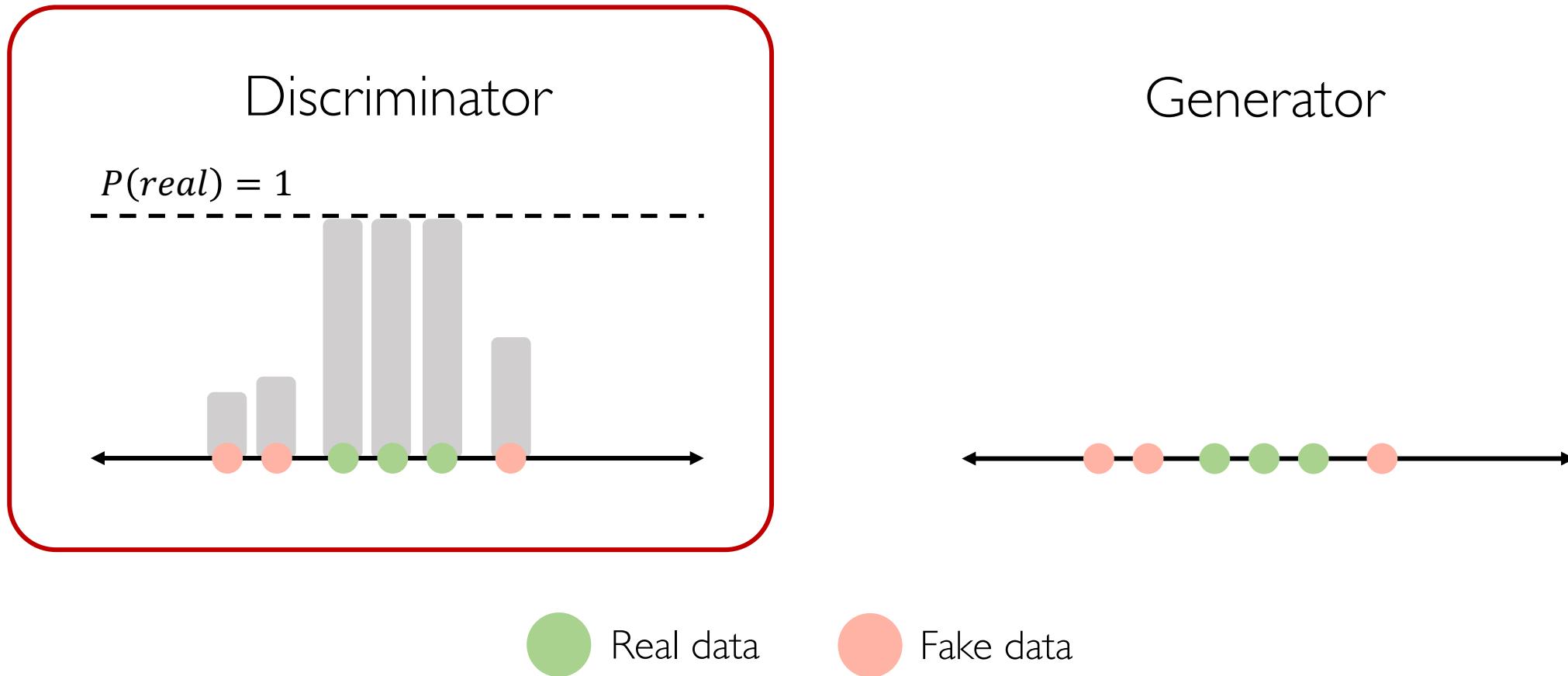
# Intuition behind GANs

**Discriminator** tries to predict what's real and what's fake.



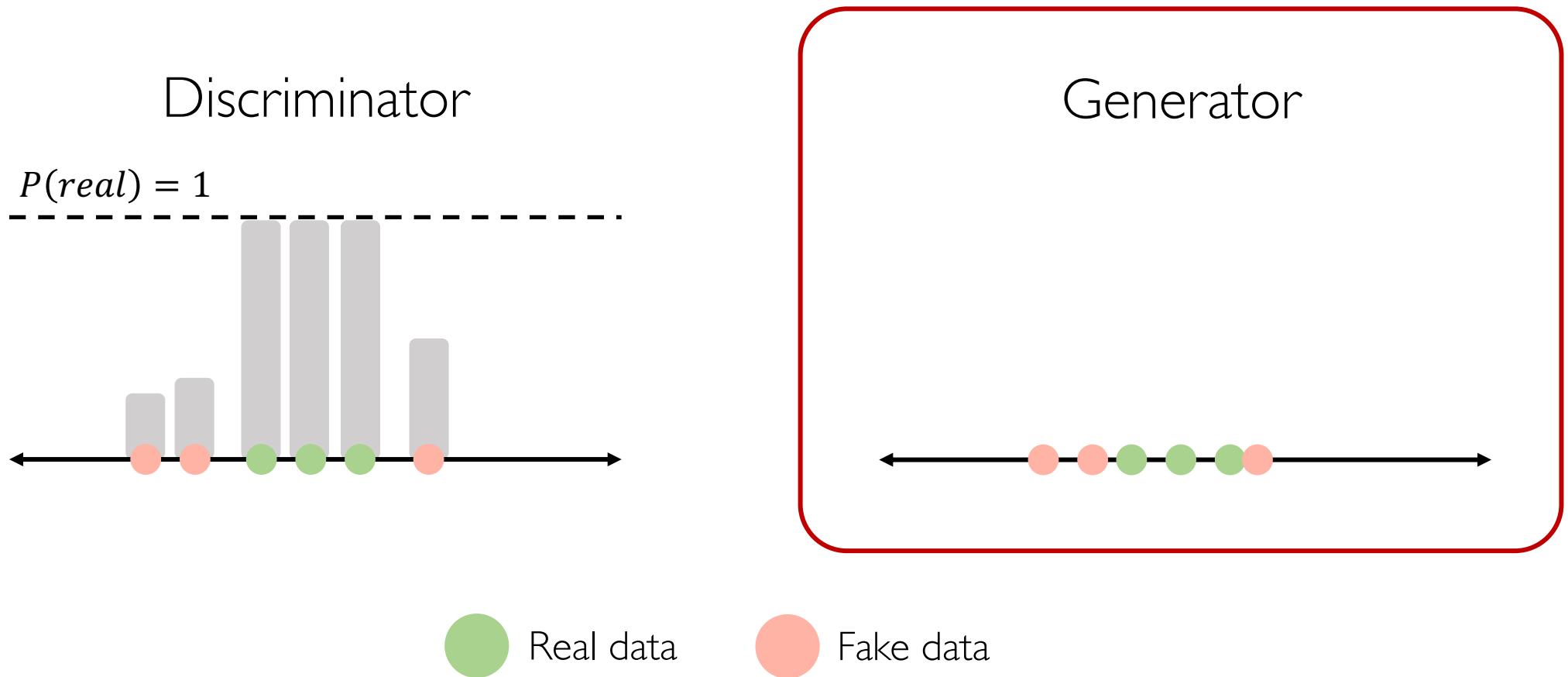
# Intuition behind GANs

**Discriminator** tries to predict what's real and what's fake.



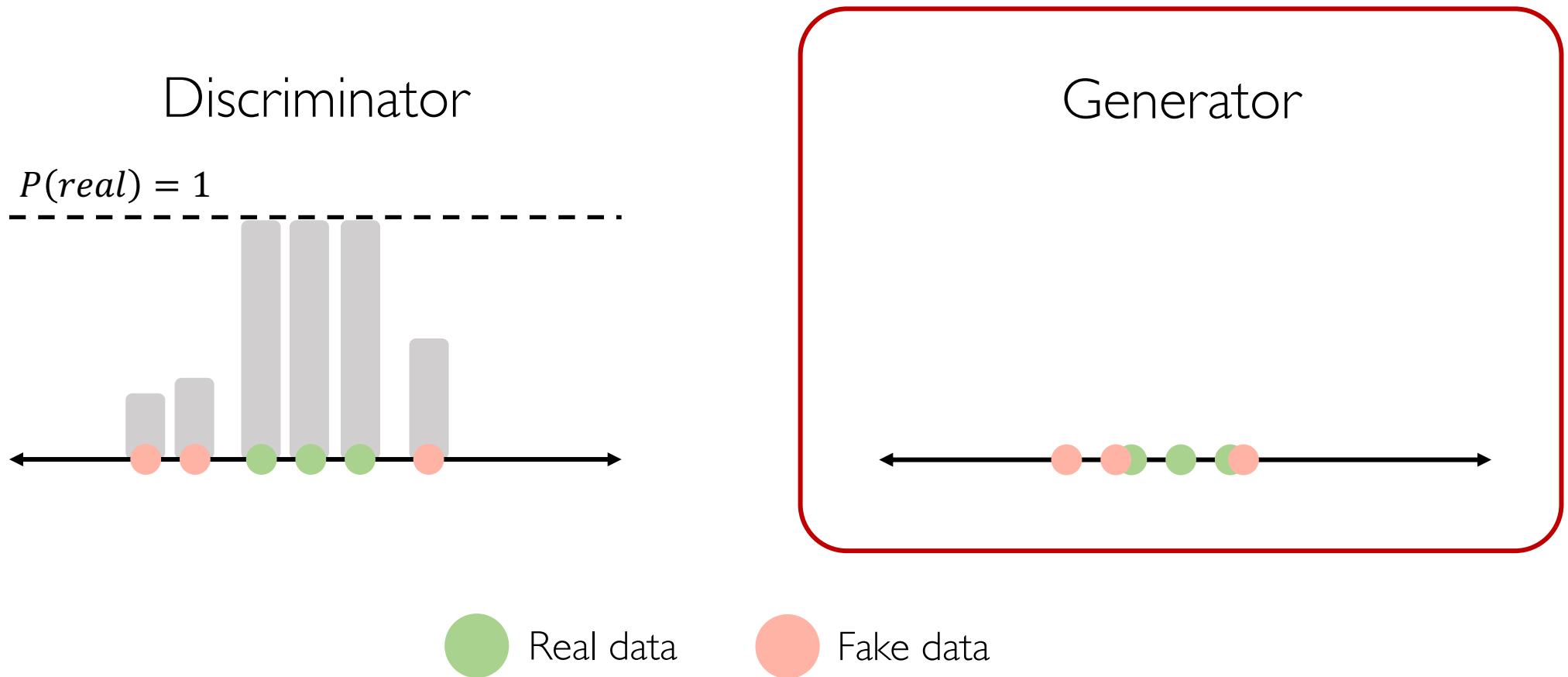
# Intuition behind GANs

**Generator** tries to improve its imitation of the data.



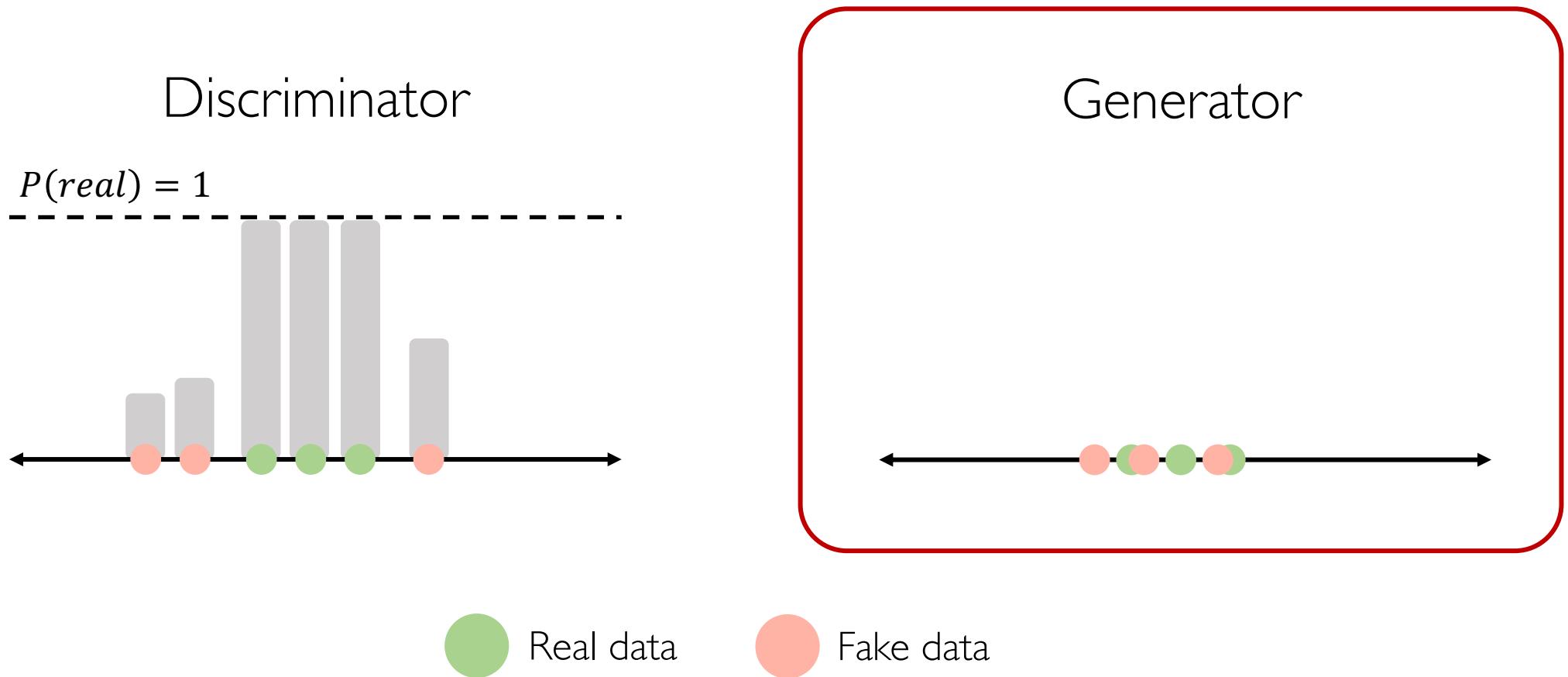
# Intuition behind GANs

**Generator** tries to improve its imitation of the data.



# Intuition behind GANs

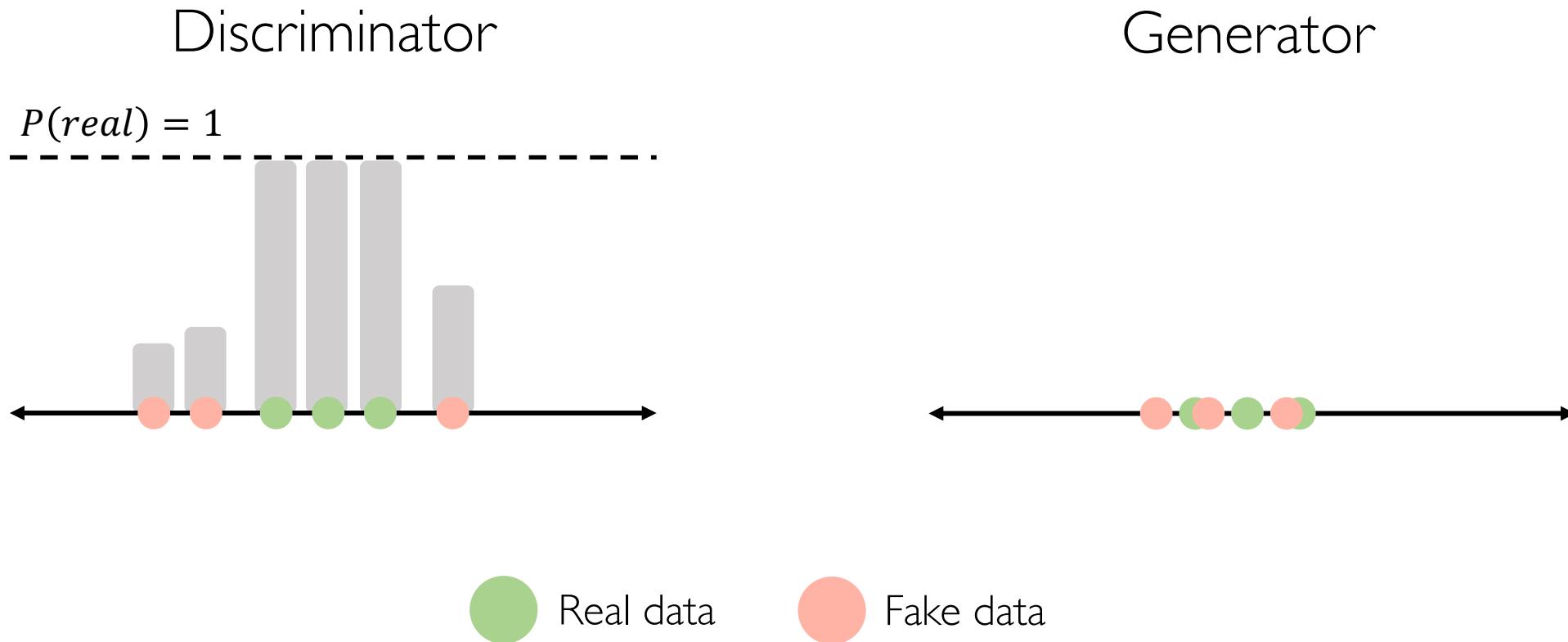
**Generator** tries to improve its imitation of the data.



# Intuition behind GANs

**Discriminator** tries to identify real data from fakes created by the generator.

**Generator** tries to create imitations of data to trick the discriminator.



# Training GANs

**Discriminator** tries to identify real data from fakes created by the generator.

**Generator** tries to create imitations of data to trick the discriminator.

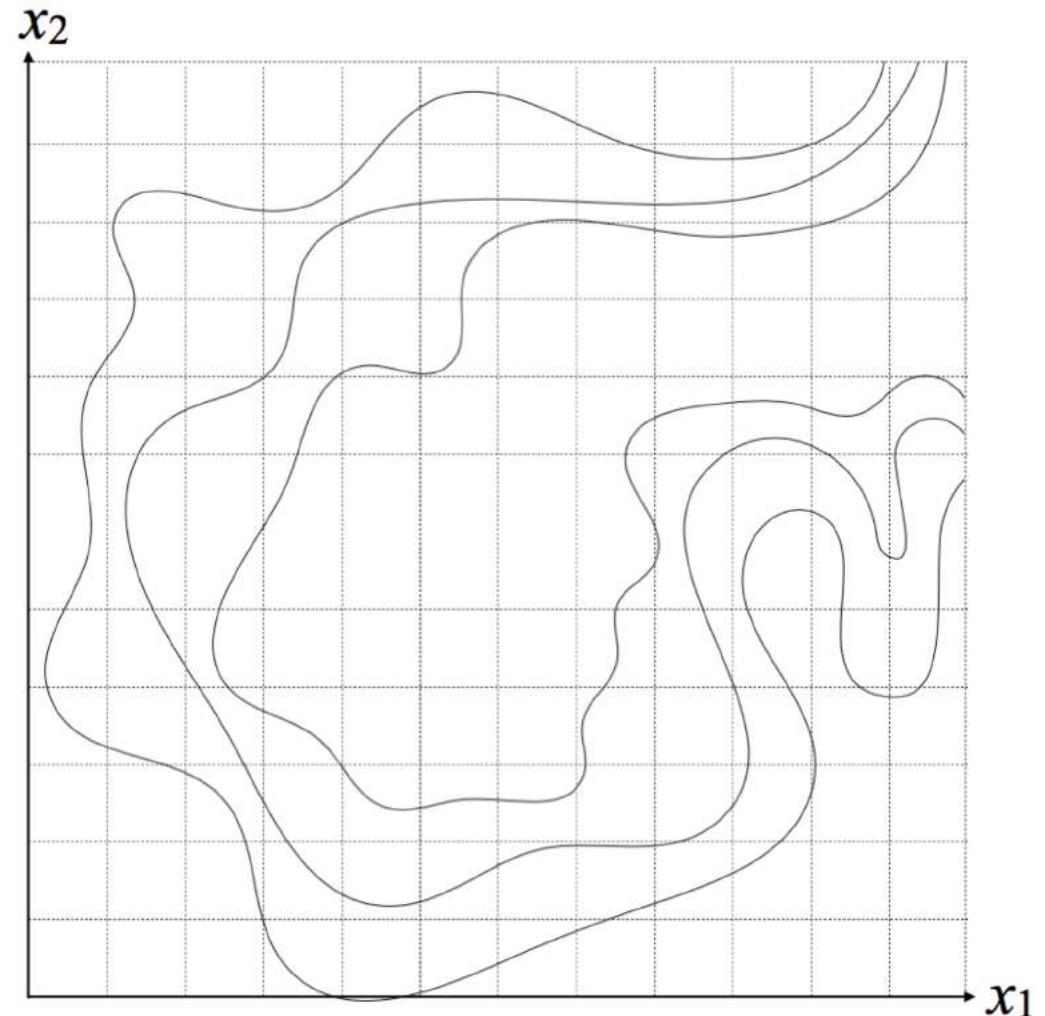
Train GAN jointly via **minimax** game:

$$\min_{\theta_g} \max_{\theta_d} \left[ \mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log \left( 1 - D_{\theta_d} (G_{\theta_g}(z)) \right) \right]$$

**Discriminator** wants to maximize objective s.t.  $D(x)$  close to 1,  $D(G(z))$  close to 0.

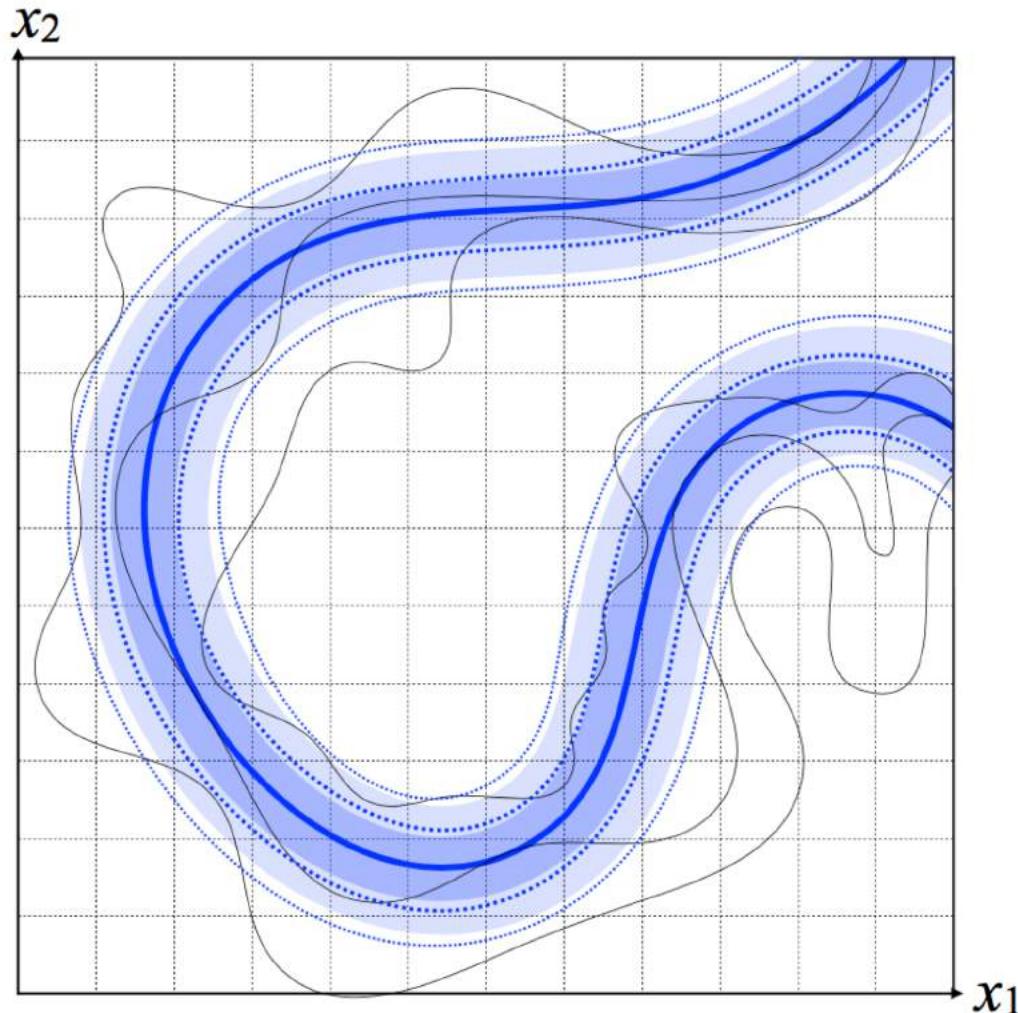
**Generator** wants to minimize objective s.t.  $D(G(z))$  close to 1.

# Why GANs?

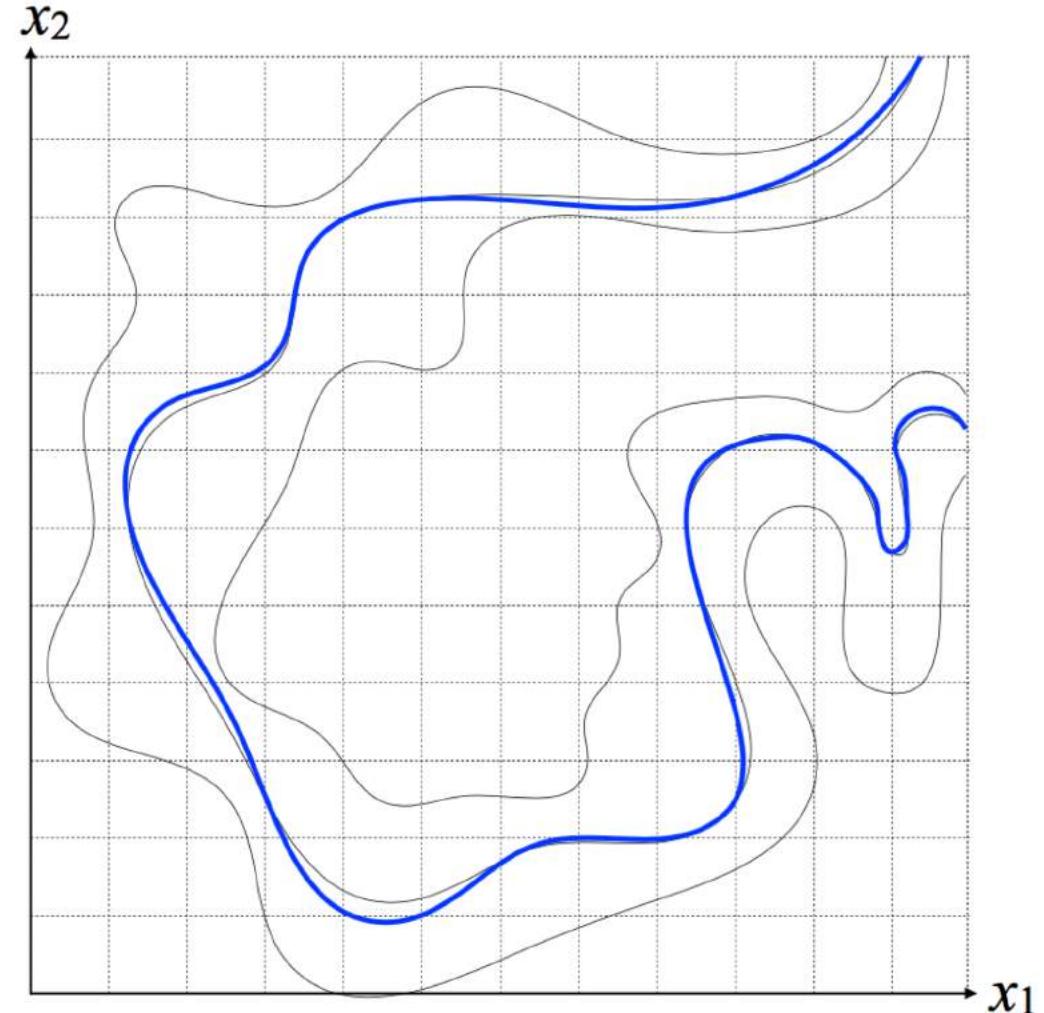


A. Courville, 6S191 2018.

# Why GANs?



more traditional max-likelihood approach

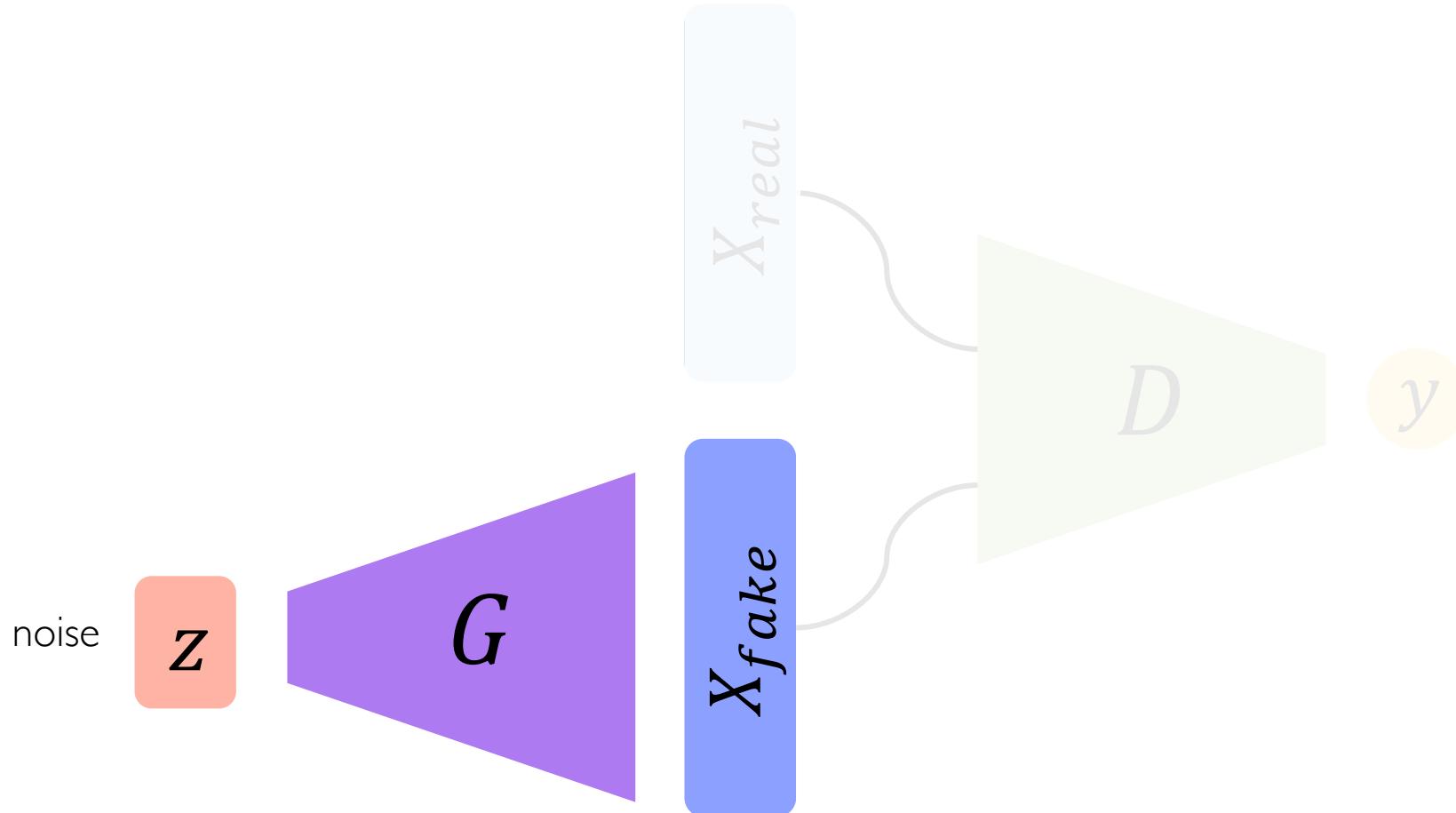


GAN

A. Courville, 6S191 2018.

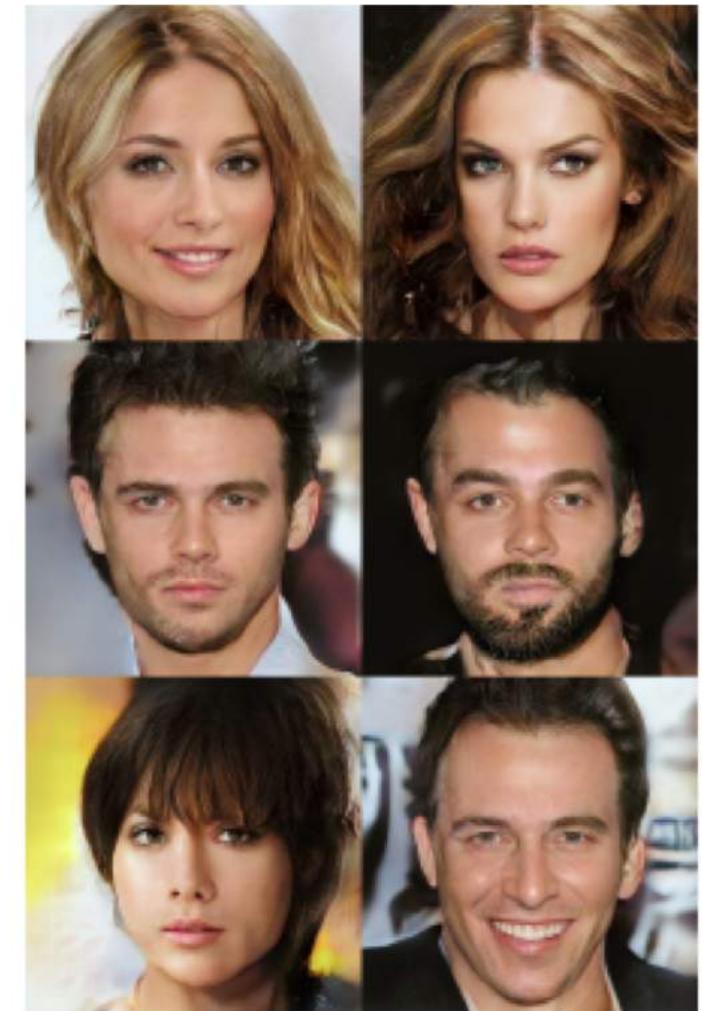
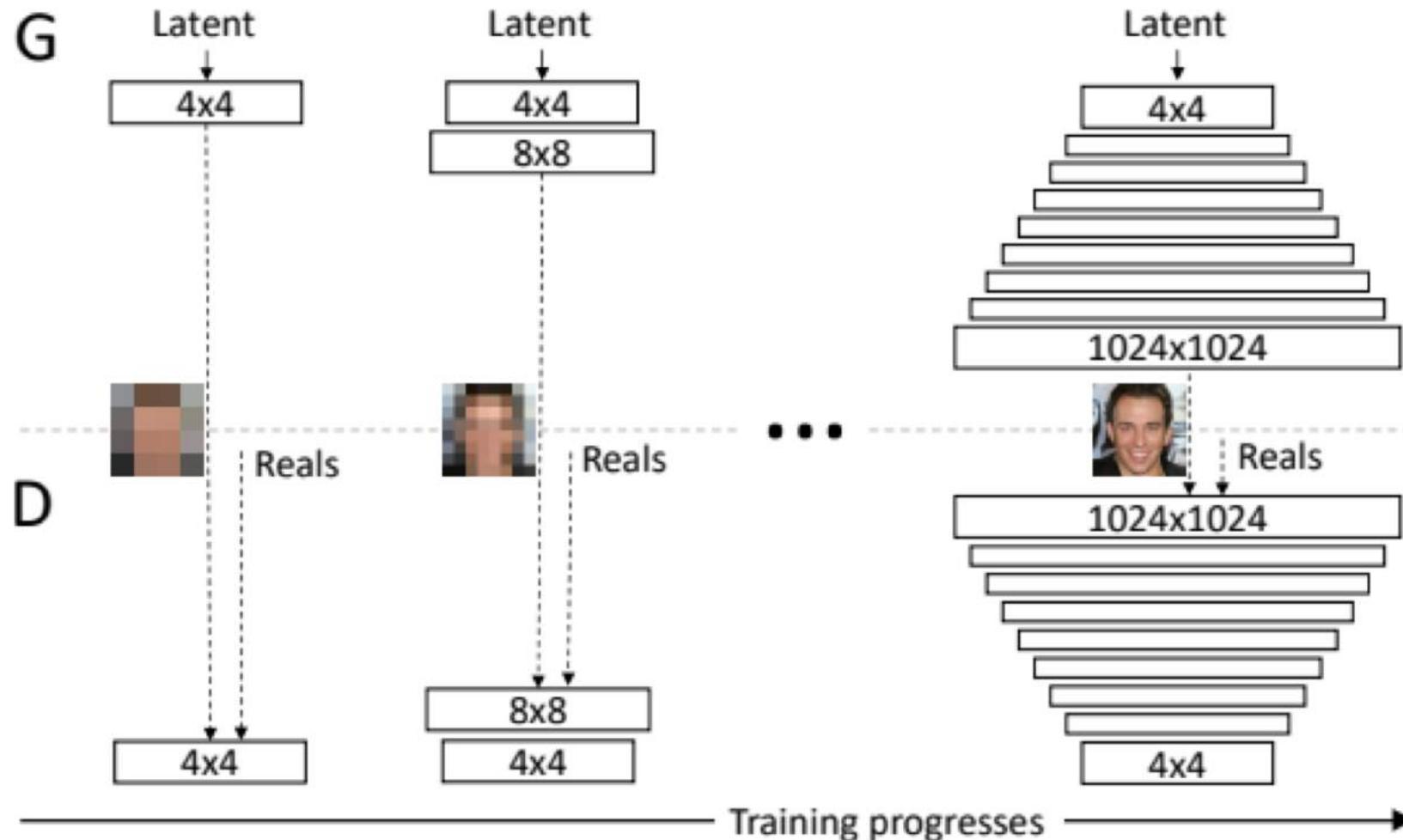
# Generating new data with GANs

After training, use generator network to create **new data** that's never been seen before.



# GANs: Recent Advances

# Progressive growing of GANs (NVIDIA)



Karras et al., ICLR 2018.

# Progressive growing of GANs: results



Karras et al., ICLR 2018.

# Style-based generator: results



Karras et al., Arxiv 2018.

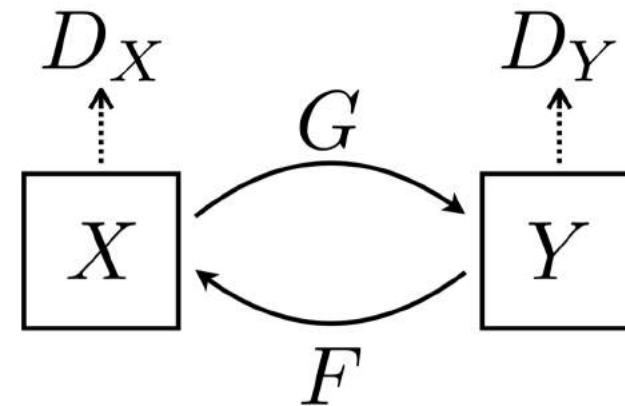
# Style-based transfer: results



Karras et al., Arxiv 2018.

# CycleGAN: domain transformation

CycleGAN learns transformations across domains with unpaired data.

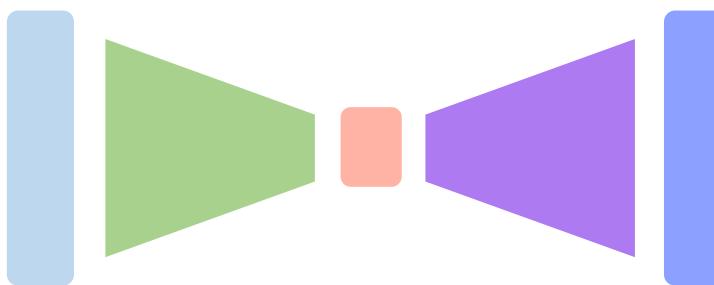


Zhu et al., ICCV 2017.

# Deep Generative Modeling: Summary

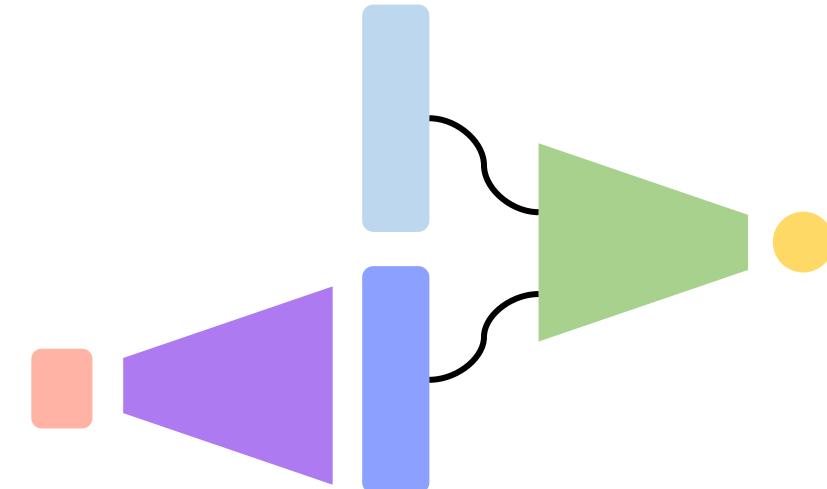
## Autoencoders and Variational Autoencoders (VAEs)

Learn **lower-dimensional** latent space and **sample** to generate input reconstructions



## Generative Adversarial Networks (GANs)

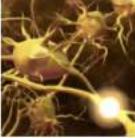
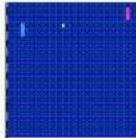
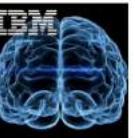
Competing **generator** and **discriminator** networks



# T-shirts! Today!



# Course Schedule

Session	Part 1	Part 2	Lab
1	 Introduction to Deep Learning [Slides] [Video] <i>coming soon!</i>	 Deep Sequence Modeling [Slides] [Video] <i>coming soon!</i>	 Intro to TensorFlow, Music Generation with RNNs [Code] <i>coming soon!</i>
2	 Deep Computer Vision [Slides] [Video] <i>coming soon!</i>	 Deep Generative Models [Slides] [Video] <i>coming soon!</i>	 De-biasing Facial Recognition Systems [Code] <i>coming soon!</i>
3	 Deep Reinforcement Learning [Slides] [Video] <i>coming soon!</i>	 Limitations and New Frontiers [Slides] [Video] <i>coming soon!</i>	 Model-Free Reinforcement Learning [Code] <i>coming soon!</i>
4	 Data Visualization for Machine Learning [Info] [Slides] [Video] <i>coming soon!</i>	 Biologically Inspired Learning [Info] [Slides] [Video] <i>coming soon!</i>	 Work time for paper reviews/project proposals
5	 Learning and Perception [Info] [Slides] [Video] <i>coming soon!</i>	 Final Project Presentations	 Judging and Awards Ceremony

# Final Class Project

## Option I: Proposal Presentation

- Present a novel deep learning research idea or application
- Groups of 1 welcome
- Listeners welcome
- Groups of 2 to 4 to be eligible for prizes, incl. 1 for-credit student
- 3 minutes
- Proposal instructions:

**goo.gl/JGJ5E7**

- Judged by a panel of industry judges
- Top winners are awarded:



3x NVIDIA RTX 2080 Ti  
MSRP: \$4000



4x Google Home  
MSRP: \$400

# Final Class Project

## Option I: Proposal Presentation

- Present a novel deep learning research idea or application
- Groups of 1 welcome
- Listeners welcome
- Groups of 2 to 4 to be eligible for prizes, incl. 1 for-credit student
- 3 minutes
- Proposal instructions:  
**[goo.gl/JGJ5E7](https://goo.gl/JGJ5E7)**

## Proposal Logistics

- $\geq 1$  for-credit student to be eligible for prizes
- Prepare slides on Google Slides
- **Group submit by today 10pm:**  
**[goo.gl/rV6rLK](https://goo.gl/rV6rLK)**
- In class project work: **Thu, Jan 31**
- **Slide submit by Thu 11:59 pm:**  
**[goo.gl/7smL8w](https://goo.gl/7smL8w)**
- Presentations on **Friday, Feb 1**

# Final Class Project

## Option 1: Proposal Presentation

- Present a novel deep learning research idea or application
- Groups of 1 welcome
- Listeners welcome
- Groups of 2 to 4 to be eligible for prizes, incl. 1 for-credit student
- 3 minutes
- Proposal instructions:  
[goo.gl/JGJ5E7](http://goo.gl/JGJ5E7)

## Option 2: Write a 1-page review of a deep learning paper

- Grade is based on clarity of writing and technical communication of main ideas
- Due **Friday 1:00pm** (before lecture)

# Thursday: Visualization in ML + Biologically Inspired Learning



**Fernanda Viegas,**  
**Co-Director Google PAIR**

Data Visualization for  
Machine Learning

**Google**



**Dmitry Krotov,**  
**MIT-IBM Watson AI Lab**

Biologically Inspired Deep  
Learning

**IBM Research**

# Friday: Learning and Perception + Project Proposals + Awards + Pizza



**Jan Kautz,**  
**VP of Research**  
Learning and Perception



**Project Proposals!**

**Judging and Awards!**

**Pizza Celebration!**

So far in 6.SI91...

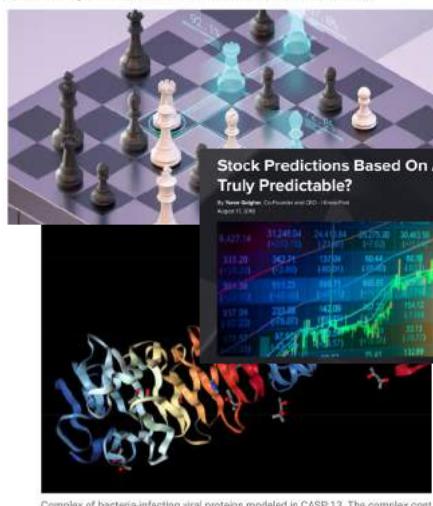
## 'Deep Voice' Software Can Clone Anyone's Voice With Just 3.7 Seconds of Audio

Using snippets of voices, Baidu's 'Deep Voice' can generate new speech, accents, and tones.



### 'Creative' AlphaZero leads way for chess computers and, maybe, science

Former chess world champion Garry Kasparov likes what he sees of computer that could be used to find cures for diseases



### Google's DeepMind aces protein folding

By Robert F. Service | Dec. 6, 2018, 12:05 PM

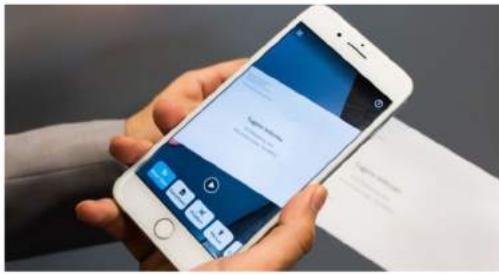
# The Rise of Deep Learning

AI

Nv  
pr  
DEAN T

## DEEPMIND E STARCRAFT TRIUMPH FO

Let There Be Sight: How Deep Learning Is Helping the Blind 'See'



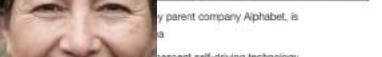
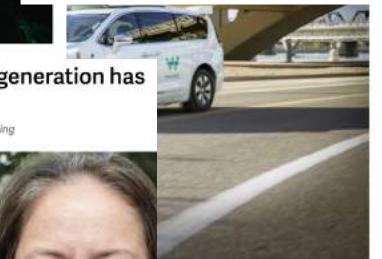
## Technology outpacing security

| Facial Recognition | Features and Interviews



## AI beats docs in cancer spottin

A new study provides a fresh example of machine learning as an important diagnostic tool. Paul Biegler reports.



My parent company Alphabet, is

## How an A.I. 'Cat-and-Mouse Game' Generates Believable Fake Photos

By CADE METZ and KEITH COLLINS | JAN 2, 2018



## Neural networks everywhere

New chip reduces neural networks' power consumption by up to 95 percent, making them practical for battery-powered devices.

Deep L

Wed, 01/10/2018 - 8:00am | 1 Comment | by Kenny Walter - Digital Reporter - @RandDMagazine

## After Millions of Trials, These Simulated Humans Learned to Do Perfect Backflips and Cartwheels

George Durkin

47,919,115 views • Posted on Al

28.3K 50 19 f t e

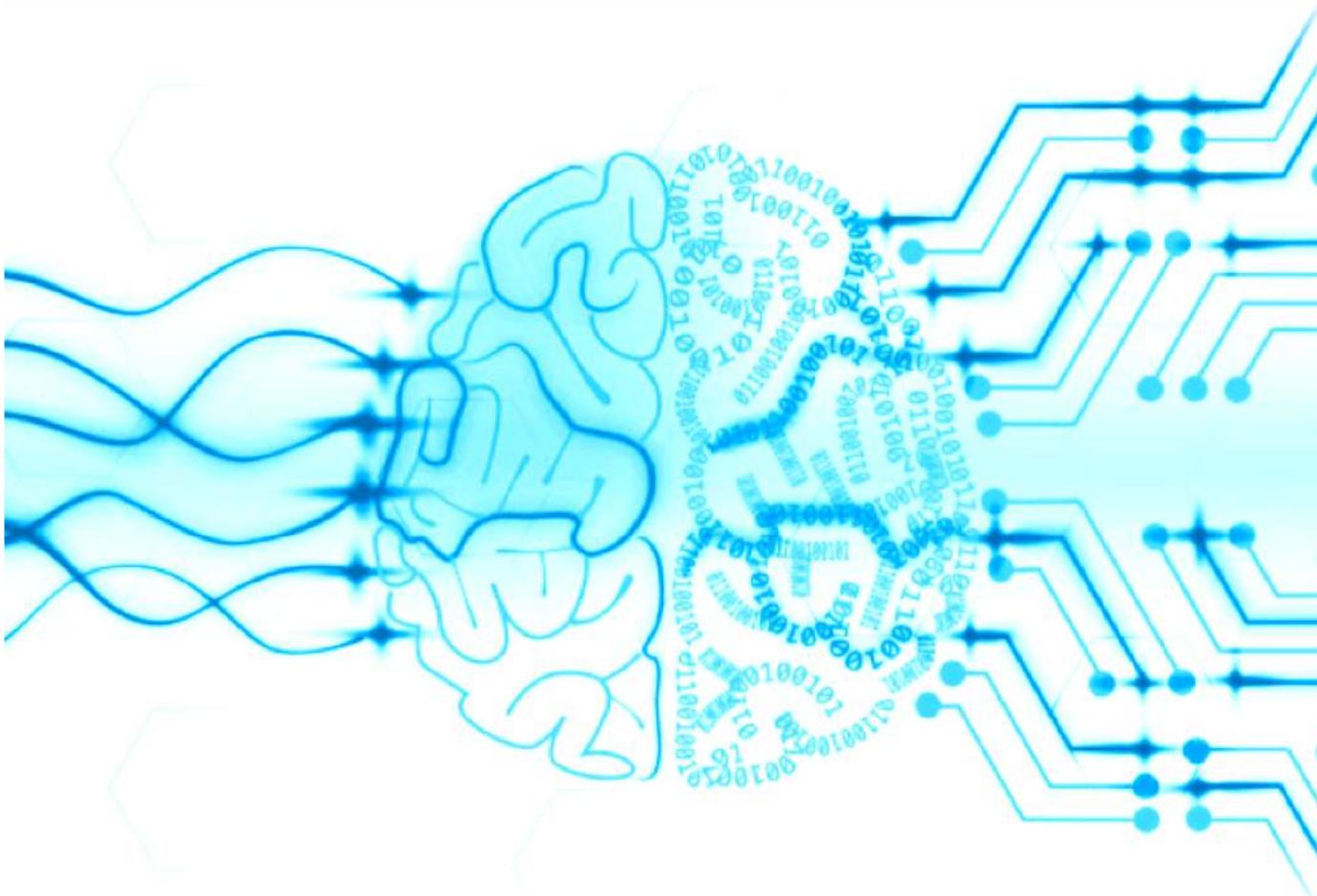
28.3K 50 19 f t e</p

# So far in 6.S191...

## Data

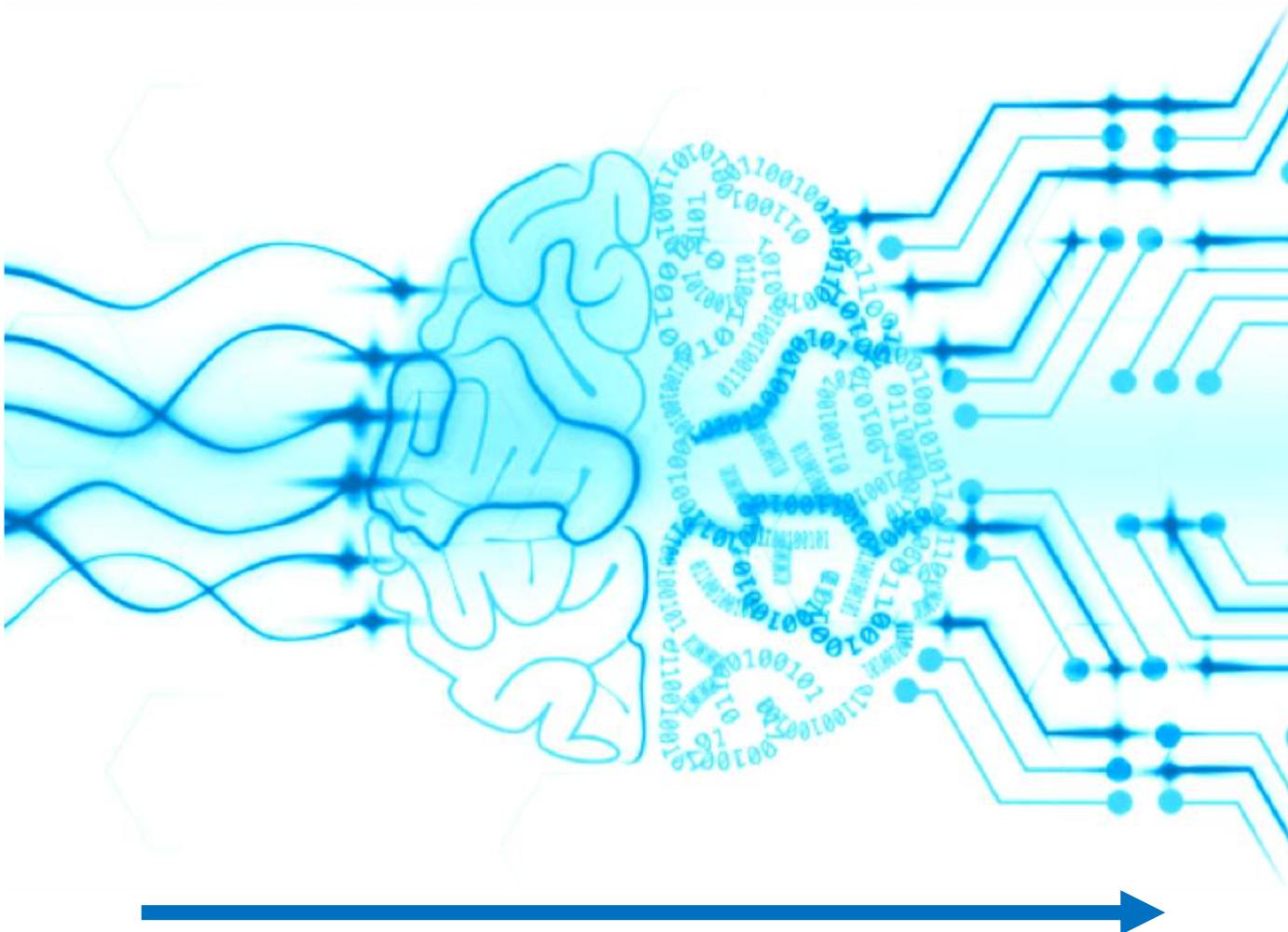
- Signals
- Images
- Sensors

...



# So far in 6.S191...

- Data**
- Signals
  - Images
  - Sensors
- ...



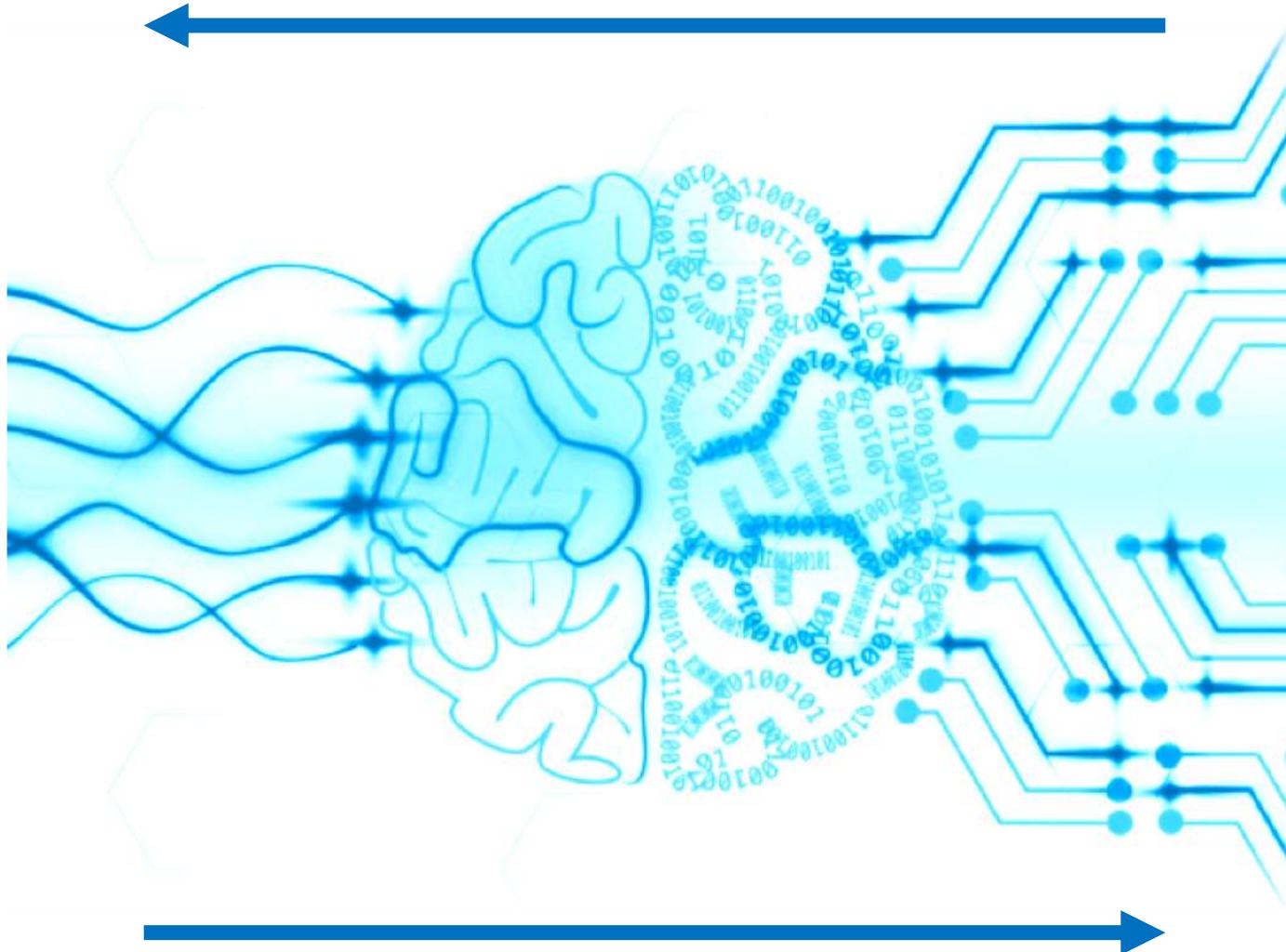
- Decision**
- Prediction
  - Detection
  - Action
- ...

# So far in 6.S191...

## Data

- Signals
- Images
- Sensors

...



## Decision

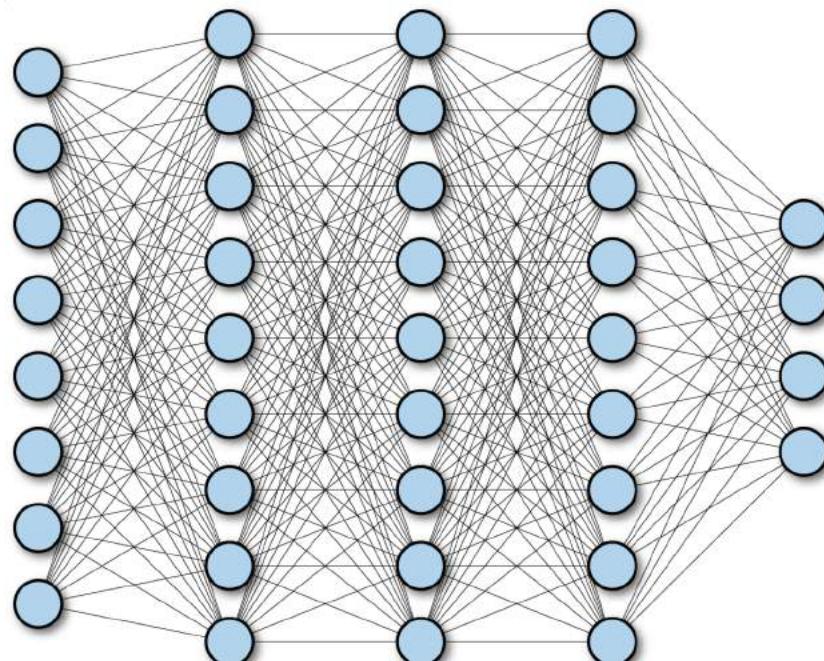
- Prediction
- Detection
- Action

...

# Power of Neural Nets

## Universal Approximation Theorem

A feedforward network with a single layer is sufficient to approximate, to an arbitrary precision, any continuous function.



Hornik et al. Neural Networks. (1989)

# Power of Neural Nets

## Universal Approximation Theorem

A feedforward network with a single layer is sufficient to approximate, to an arbitrary precision, any continuous function.

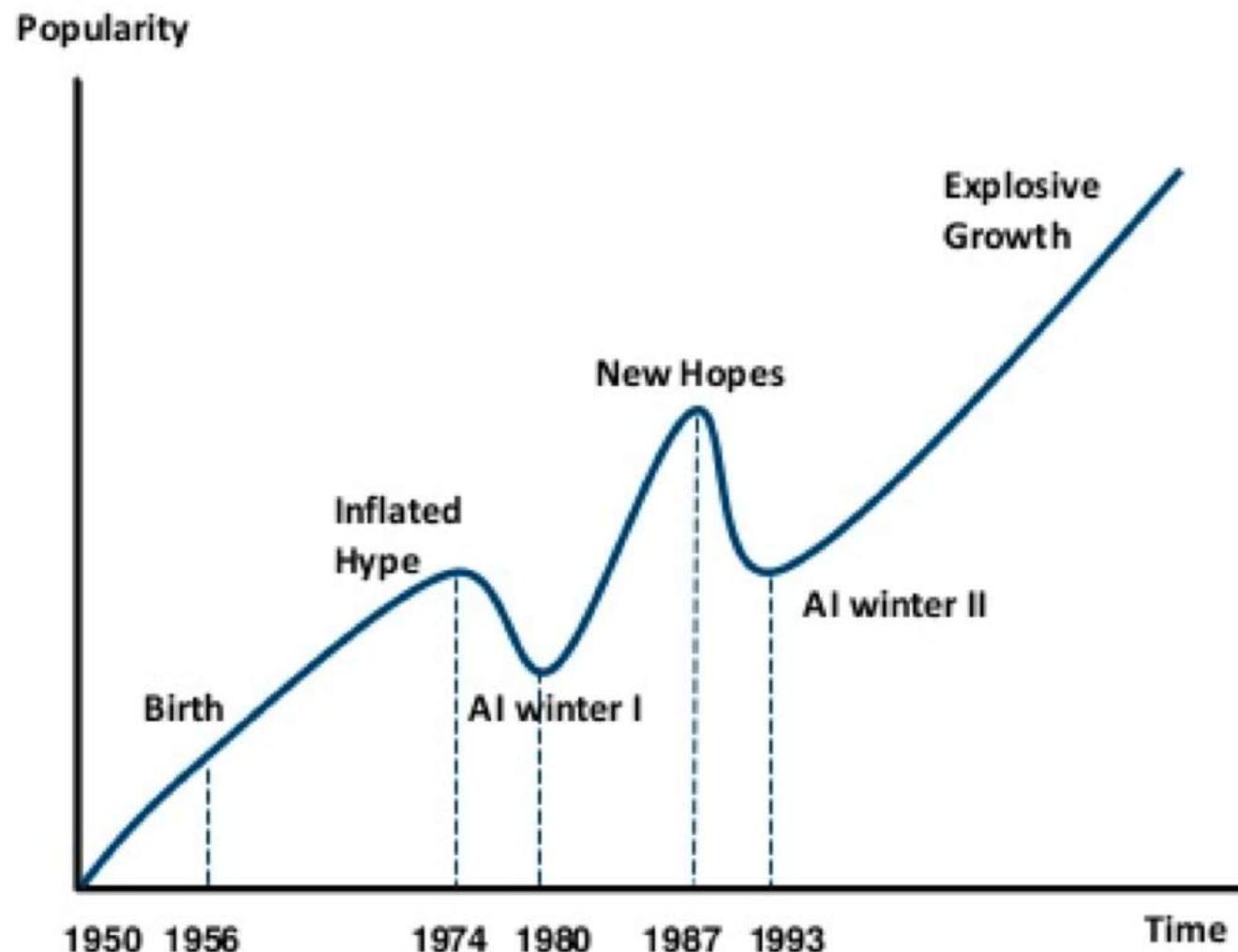
### Caveats:

The number of hidden units may be infeasibly large

The resulting model may not generalize

Hornik et al. Neural Networks. (1989)

# Artificial Intelligence “Hype”: Historical Perspective



# Limitations

# Rethinking Generalization

“Understanding Deep Neural Networks Requires Rethinking Generalization”



dog



banana



dog



tree

Zhang et al. ICLR. (2017)

# Rethinking Generalization

“Understanding Deep Neural Networks Requires Rethinking Generalization”



dog



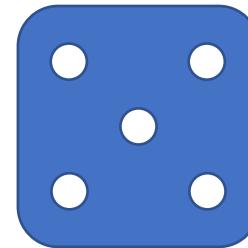
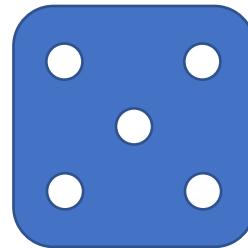
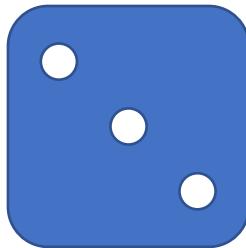
banana



dog



tree



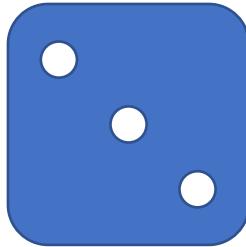
Zhang et al. ICLR. (2017)

# Rethinking Generalization

“Understanding Deep Neural Networks Requires Rethinking Generalization”



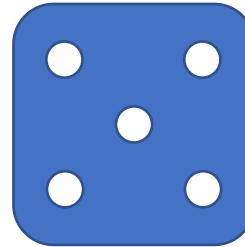
dog



banana



banana



dog



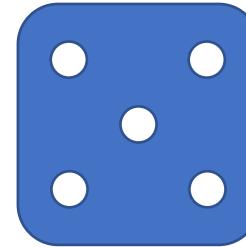
dog



tree



tree



dog

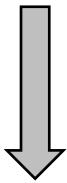
Zhang et al. ICLR. (2017)

# Rethinking Generalization

“Understanding Deep Neural Networks Requires Rethinking Generalization”



~~dog~~



banana

~~banana~~



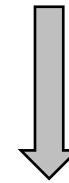
dog

~~dog~~



tree

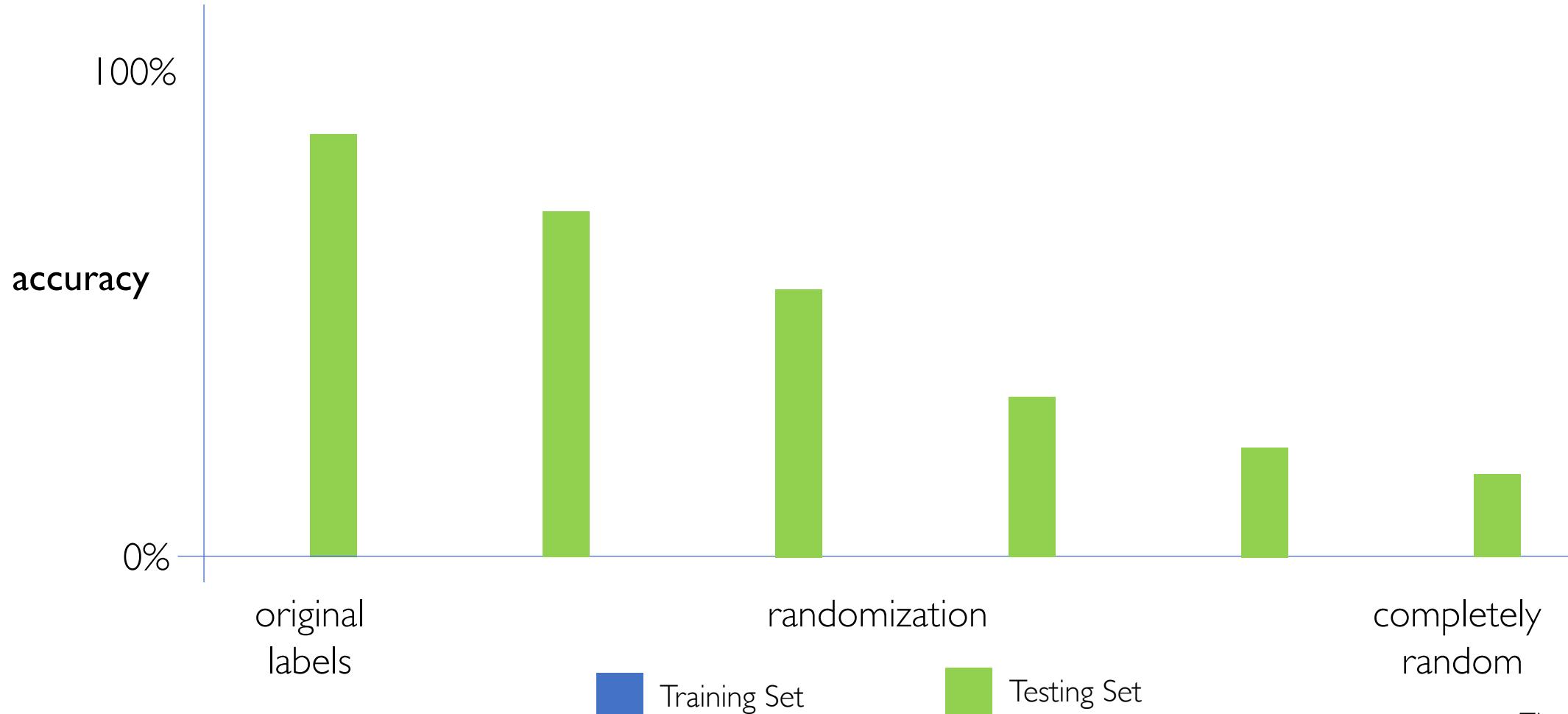
~~tree~~



dog

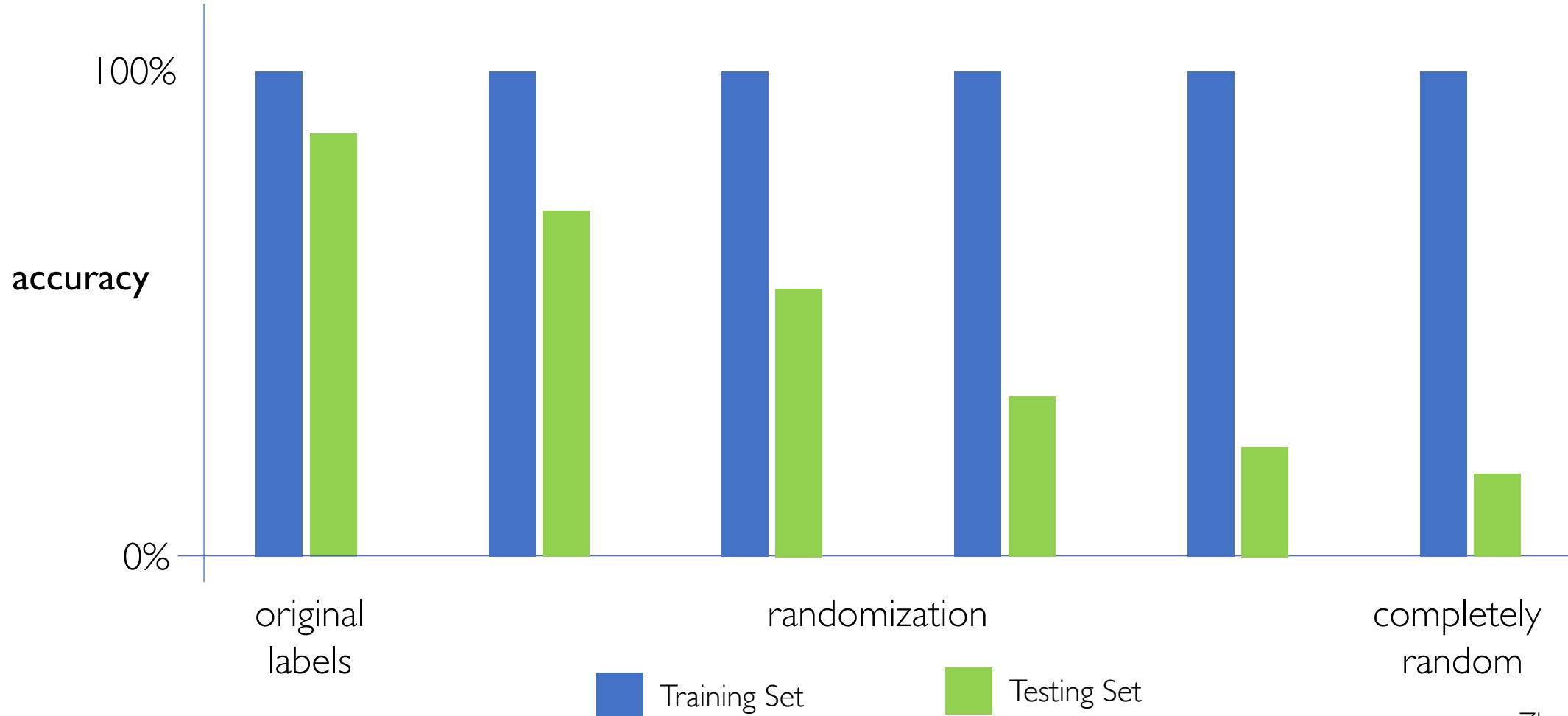
Zhang et al. ICLR. (2017)

# Capacity of Deep Neural Networks



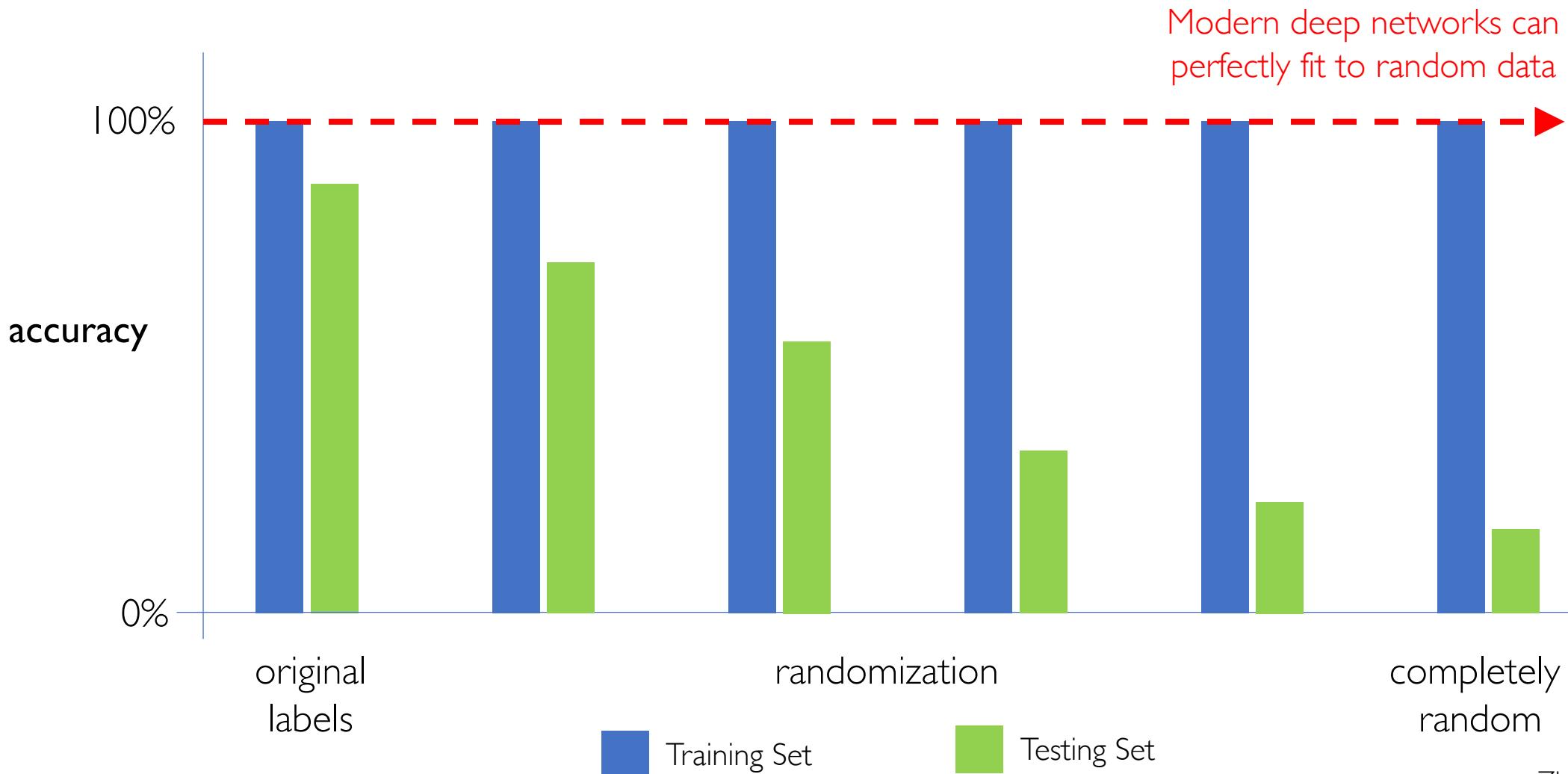
Zhang et al. ICLR. (2017)

# Capacity of Deep Neural Networks



Zhang et al. ICLR. (2017)

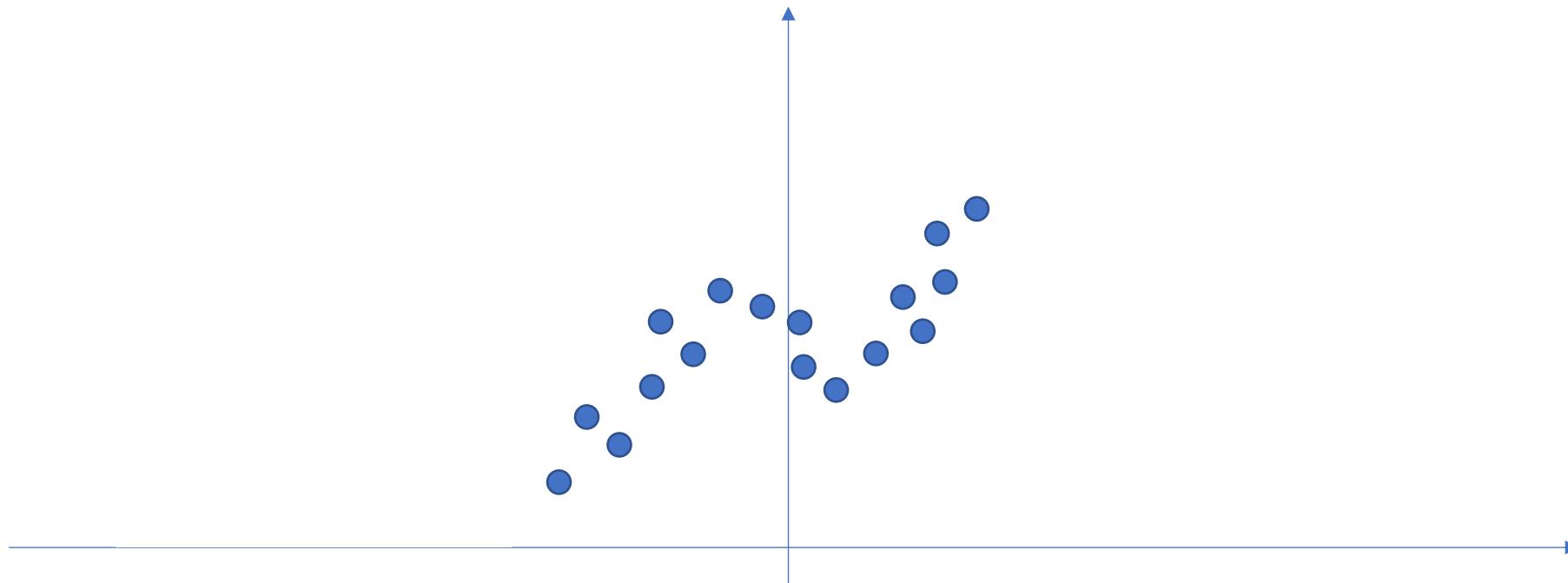
# Capacity of Deep Neural Networks



Zhang et al. ICLR. (2017)

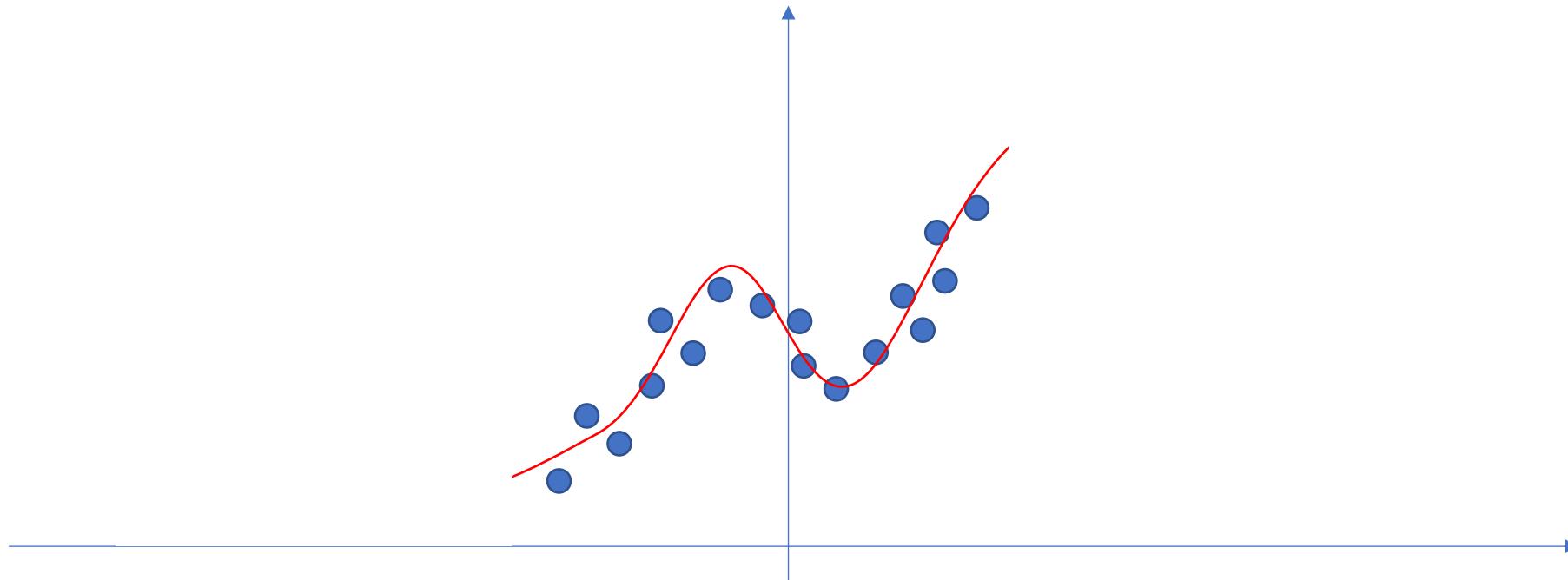
# Neural Networks as Function Approximators

Neural networks are **excellent** function approximators



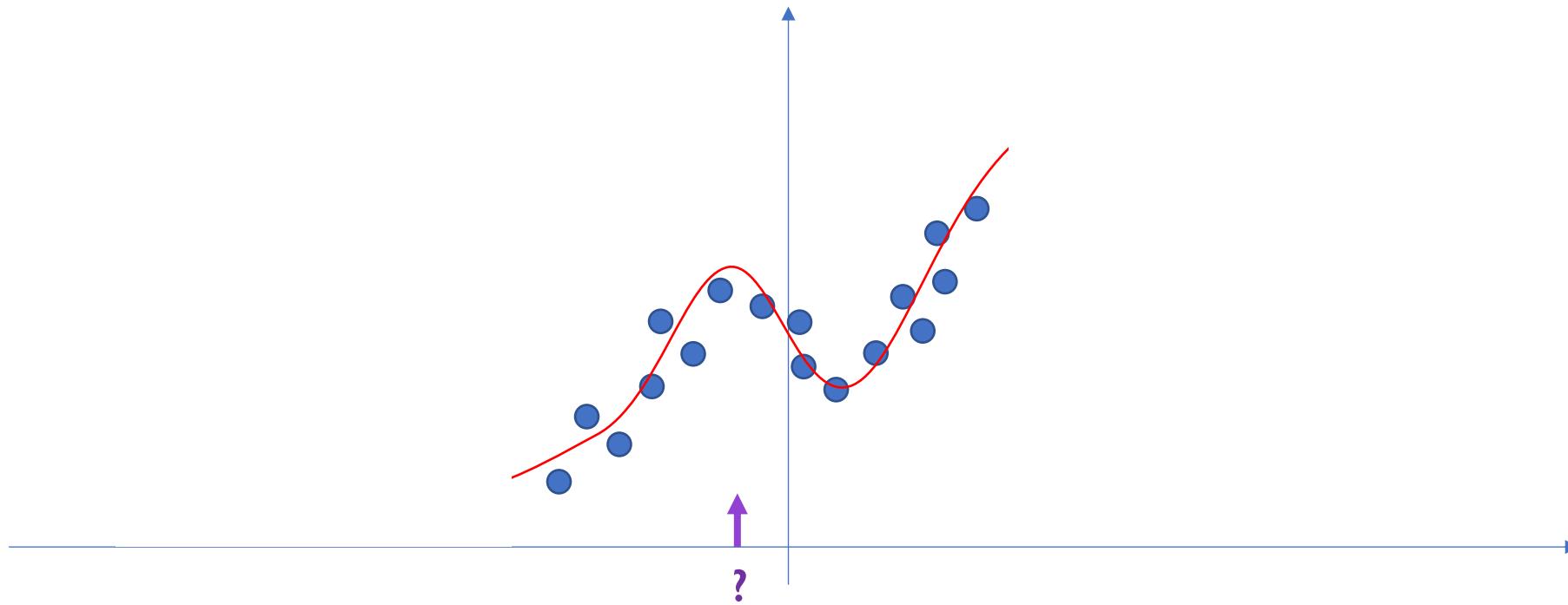
# Neural Networks as Function Approximators

Neural networks are **excellent** function approximators



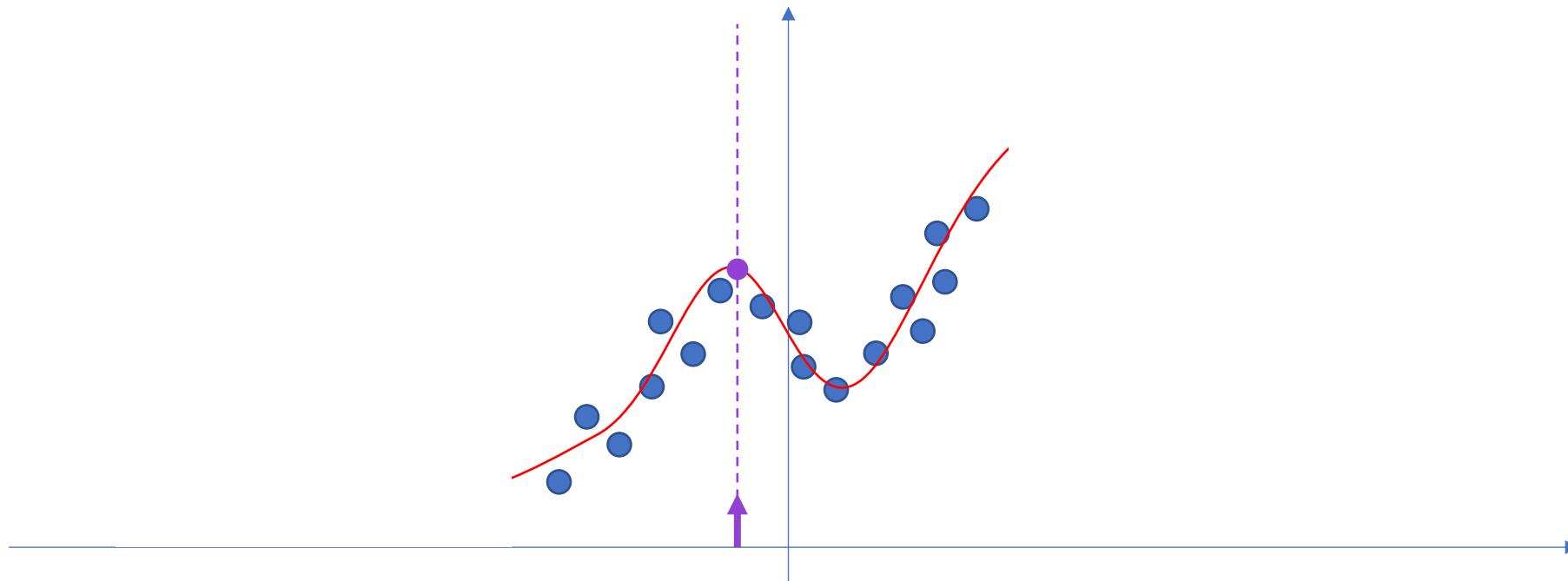
# Neural Networks as Function Approximators

Neural networks are **excellent** function approximators



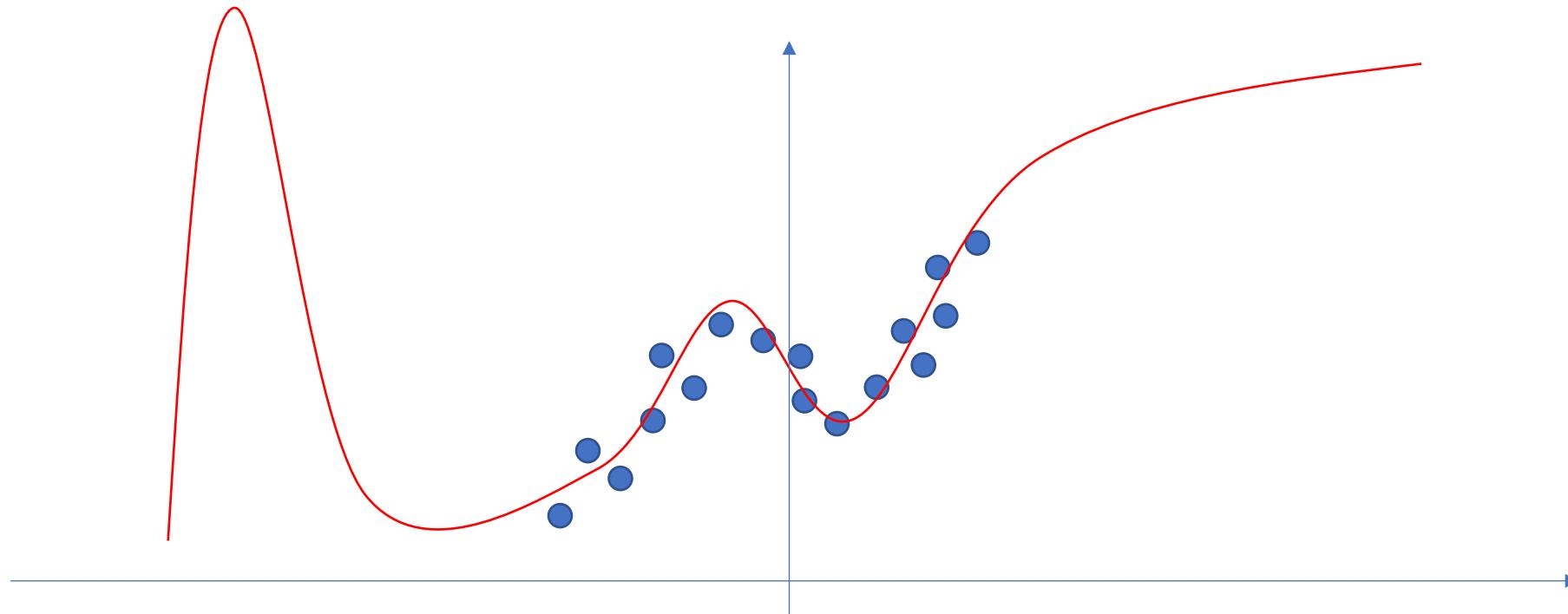
# Neural Networks as Function Approximators

Neural networks are **excellent** function approximators



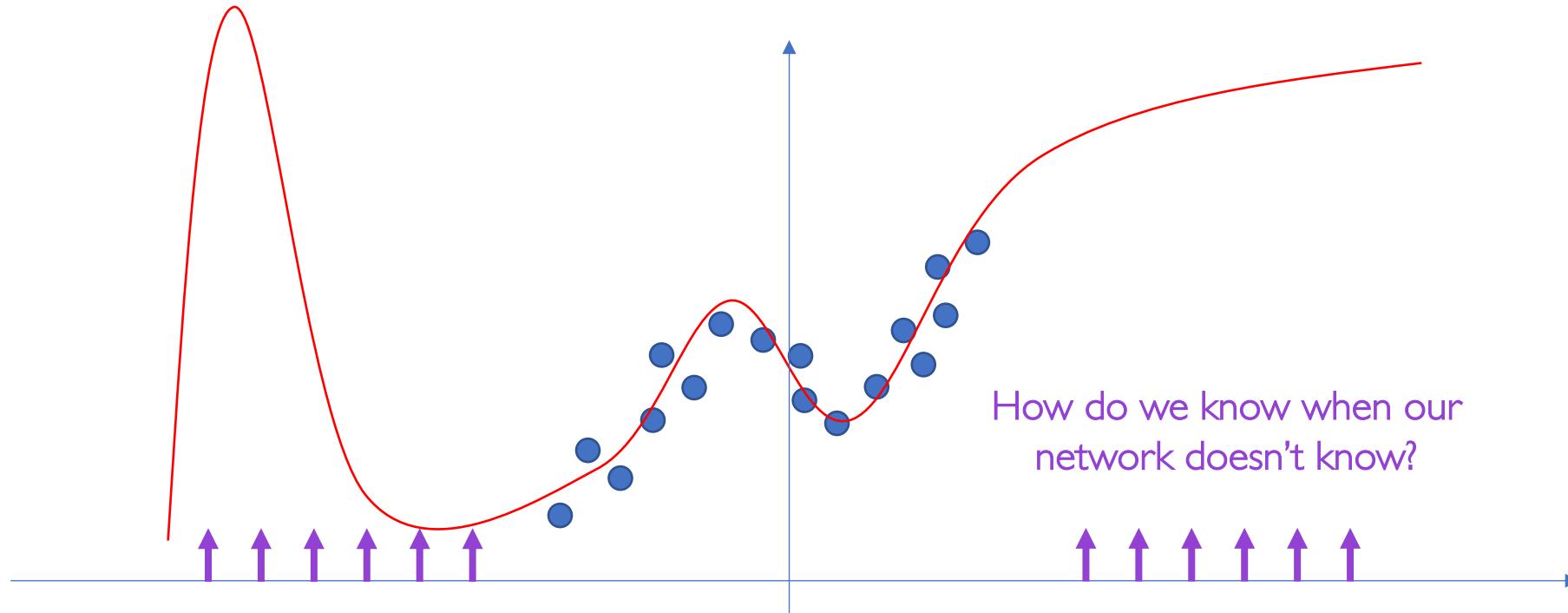
# Neural Networks as Function Approximators

Neural networks are **excellent** function approximators

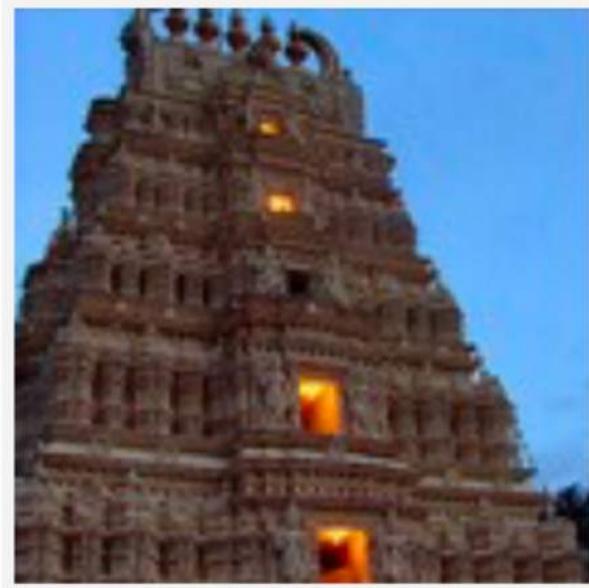


# Neural Networks as Function Approximators

Neural networks are **excellent** function approximators  
...when they have training data

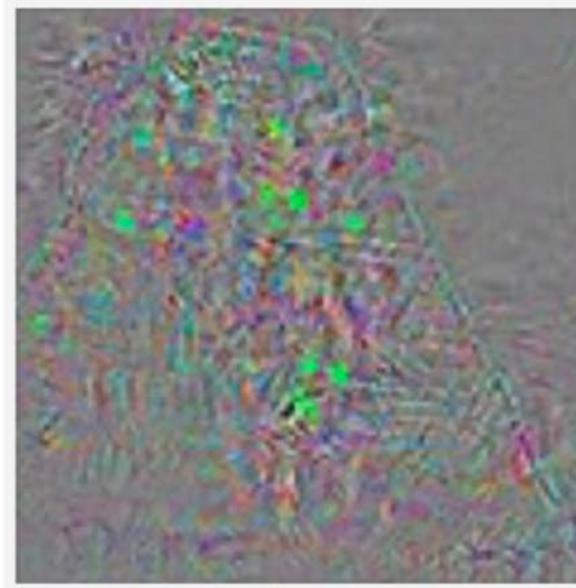


# Adversarial Attacks on Neural Networks

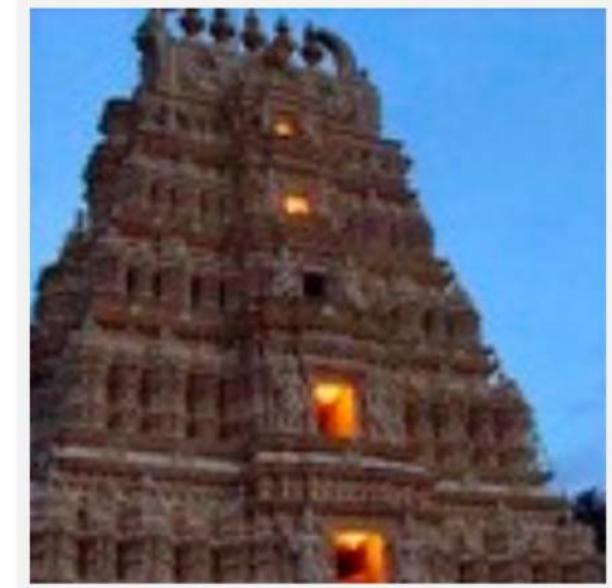


**Original image**

Temple (97%)



**Perturbations**



**Adversarial example**

Ostrich (98%)

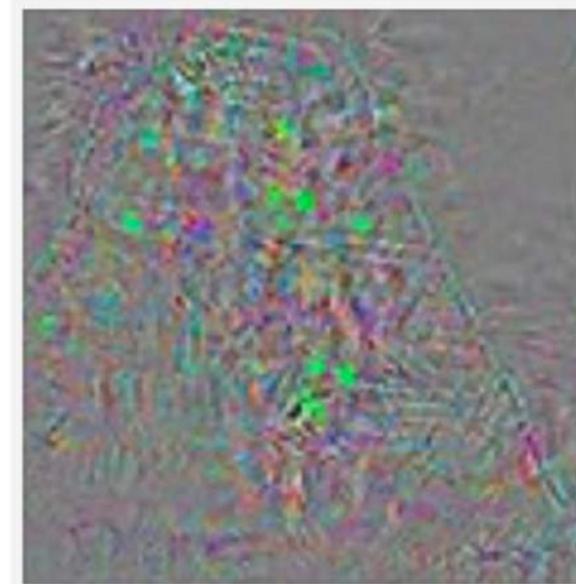
Despois. "Adversarial examples and their implications" (2017).

# Adversarial Attacks on Neural Networks

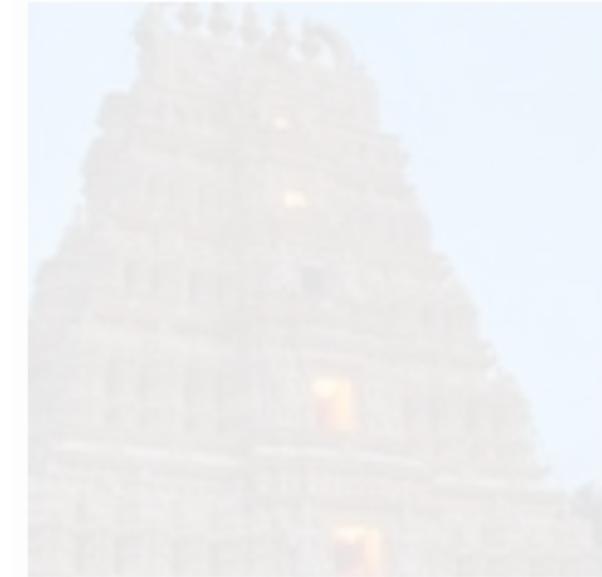


Original image

Temple (97%)



Perturbations



Adversarial example

Ostrich (98%)

# Adversarial Attacks on Neural Networks

## Remember:

We train our networks with gradient descent

$$\theta \leftarrow \theta - \eta \frac{\partial J(\theta, x, y)}{\partial \theta}$$

“How does a small change in weights decrease our loss”

# Adversarial Attacks on Neural Networks

## Remember:

We train our networks with gradient descent

$$\theta \leftarrow \theta - \eta \frac{\partial J(\theta, x, y)}{\partial \theta}$$

“How does a small change in weights decrease our loss”

# Adversarial Attacks on Neural Networks

## Remember:

We train our networks with gradient descent

$$\theta \leftarrow \theta - \eta \frac{\partial J(\theta, x, y)}{\partial \theta}$$

Fix your image  $x$ ,  
and true label  $y$

“How does a small change in weights decrease our loss”

# Adversarial Attacks on Neural Networks

## Adversarial Image:

Modify image to increase error

$$x \leftarrow x + \eta \frac{\partial J(\theta, x, y)}{\partial x}$$

*“How does a small change in the input increase our loss”*

Goodfellow et al. NIPS (2014)

# Adversarial Attacks on Neural Networks

## Adversarial Image:

Modify image to increase error

$$x \leftarrow x + \eta \frac{\partial J(\theta, x, y)}{\partial x}$$

“How does a small change in the input increase our loss”

# Adversarial Attacks on Neural Networks

## Adversarial Image:

Modify image to increase error

$$x \leftarrow x + \eta \frac{\partial J(\theta, x, y)}{\partial x}$$

Fix your weights  $\theta$ ,  
and true label  $y$

“How does a small change in the input increase our loss”

# Synthesizing Robust Adversarial Examples



■ classified as turtle   ■ classified as rifle  
■ classified as other

Athalye et al. ICML. (2018)

# Neural Network Limitations...

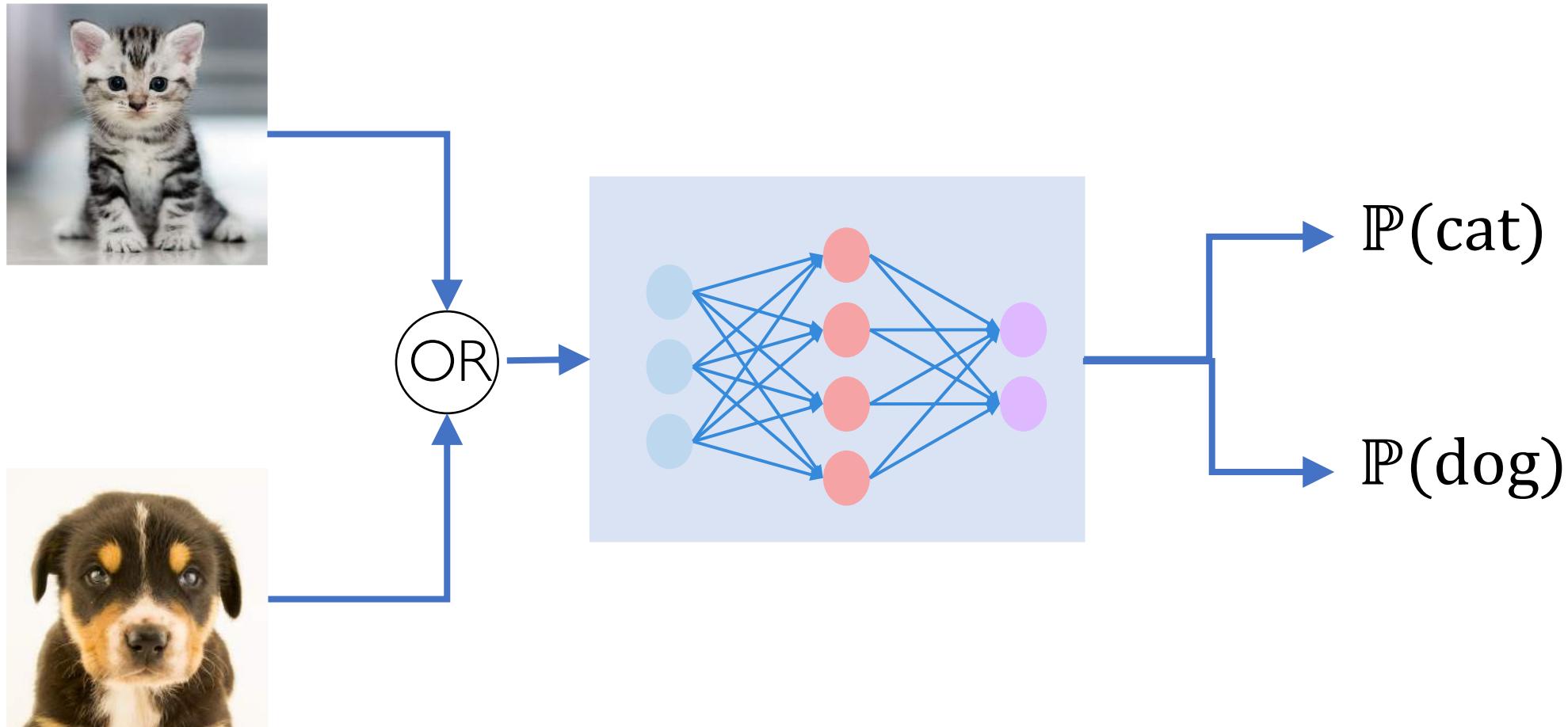
- Very **data hungry** (eg. often millions of examples)
- **Computationally intensive** to train and deploy (tractably requires GPUs)
- Easily fooled by **adversarial examples**
- Can be subject to **algorithmic bias**
- Poor at **representing uncertainty** (how do you know what the model knows?)
- Uninterpretable **black boxes**, difficult to trust
- **Finicky to optimize**: non-convex, choice of architecture, learning parameters
- Often require **expert knowledge** to design, fine tune architectures

# Neural Network Limitations...

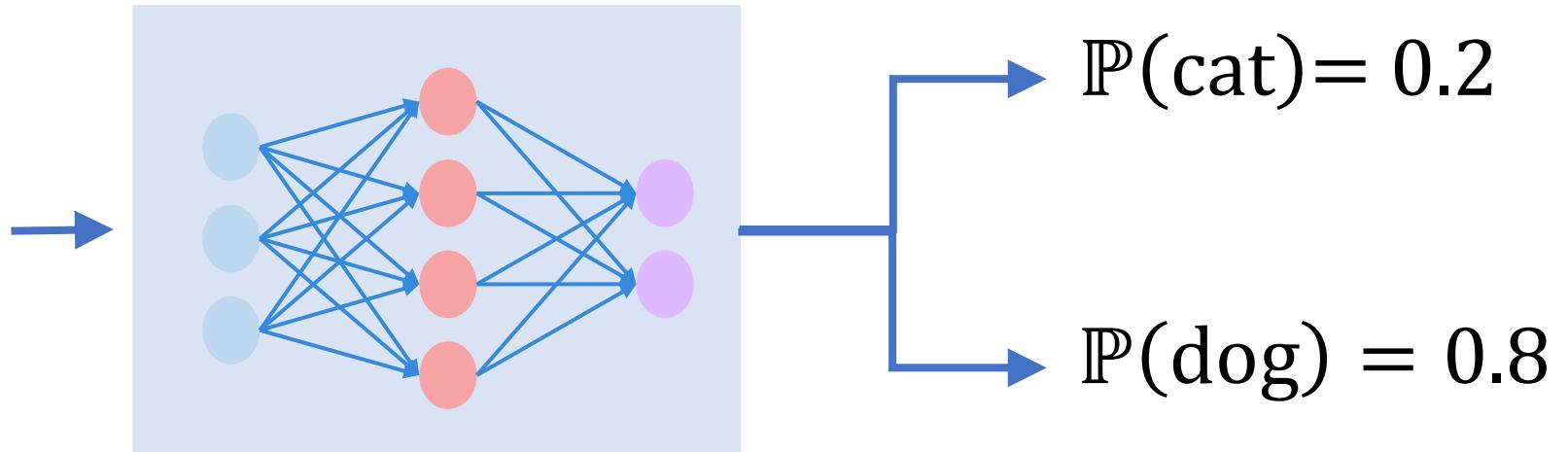
- Very **data hungry** (eg. often millions of examples)
- **Computationally intensive** to train and deploy (tractably requires GPUs)
- Easily fooled by **adversarial examples**
- Can be subject to **algorithmic bias**
- Poor at **representing uncertainty** (how do you know what the model knows?)
- Uninterpretable **black boxes**, difficult to trust
- **Finicky to optimize**: non-convex, choice of architecture, learning parameters
- Often require **expert knowledge** to design, fine tune architectures

# New Frontiers I: Bayesian Deep Learning

# Why Care About Uncertainty?



# Why Care About Uncertainty?



Remember:  $\mathbb{P}(\text{cat}) + \mathbb{P}(\text{dog}) = 1$

# Bayesian Deep Learning for Uncertainty

Network tries to learn output,  $\mathbf{Y}$ , directly from raw data,  $\mathbf{X}$

Find mapping,  $f$ , parameterized by weights  $\boldsymbol{\theta}$  such that

$$\min \mathcal{L}(\mathbf{Y}, f(\mathbf{X}; \boldsymbol{\theta}))$$

Bayesian neural networks aim to learn a posterior over weights,

$$\mathbb{P}(\boldsymbol{\theta}|\mathbf{X}, \mathbf{Y}):$$

$$\mathbb{P}(\boldsymbol{\theta}|\mathbf{X}, \mathbf{Y}) = \frac{\mathbb{P}(\mathbf{Y}|\mathbf{X}, \boldsymbol{\theta})\mathbb{P}(\boldsymbol{\theta})}{\mathbb{P}(\mathbf{Y}|\mathbf{X})}$$

# Bayesian Deep Learning for Uncertainty

Network tries to learn output,  $\mathbf{Y}$ , directly from raw data,  $\mathbf{X}$

Find mapping,  $f$ , parameterized by weights  $\boldsymbol{\theta}$  such that

$$\min \mathcal{L}(\mathbf{Y}, f(\mathbf{X}; \boldsymbol{\theta}))$$

Bayesian neural networks aim to learn a posterior over weights,

$$\mathbb{P}(\boldsymbol{\theta}|\mathbf{X}, \mathbf{Y}):$$

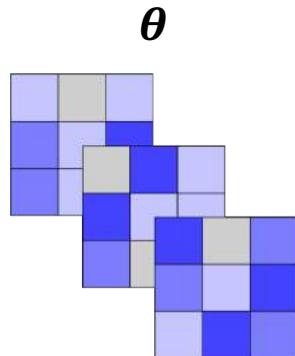
Intractable! 
$$\mathbb{P}(\boldsymbol{\theta}|\mathbf{X}, \mathbf{Y}) = \frac{\mathbb{P}(\mathbf{Y}|\mathbf{X}, \boldsymbol{\theta})\mathbb{P}(\boldsymbol{\theta})}{\mathbb{P}(\mathbf{Y}|\mathbf{X})}$$

# Elementwise Dropout for Uncertainty

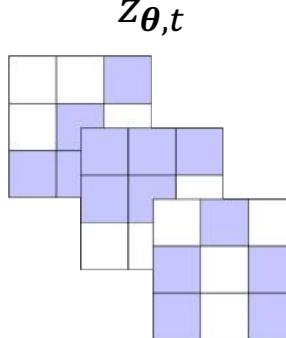
Evaluate  $T$  stochastic forward passes through the network  $\{\boldsymbol{\theta}_t\}_{t=1}^T$

Dropout as a form of stochastic sampling  $z_{w,t} \sim \text{Bernoulli}(p) \quad \forall w \in \boldsymbol{\theta}$

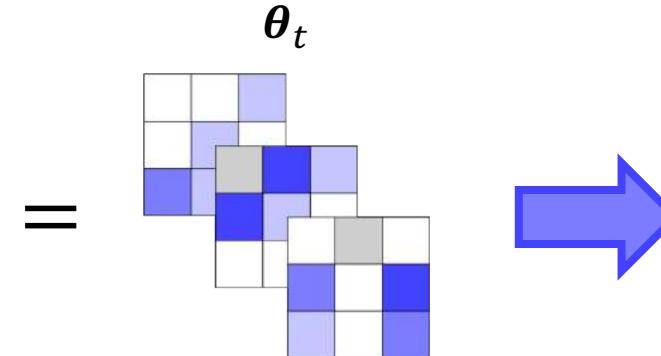
Unregularized Kernel



Bernoulli Dropout



Stochastic Sampled



$$\mathbb{E}(\hat{Y}|X) = \frac{1}{T} \sum_{t=1}^T f(X|\boldsymbol{\theta}_t)$$

$$Var(\hat{Y}|X) = \frac{1}{T} \sum_{t=1}^T f(X)^2 - \mathbb{E}(\hat{Y}|X)^2$$

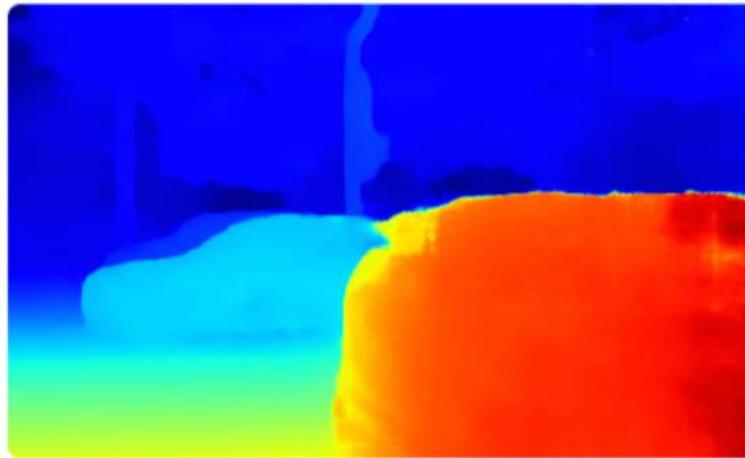
Gal and Ghahramani, ICML, 2016.

Amini, Soleimany, et al., NIPS Workshop on Bayesian Deep Learning, 2017.

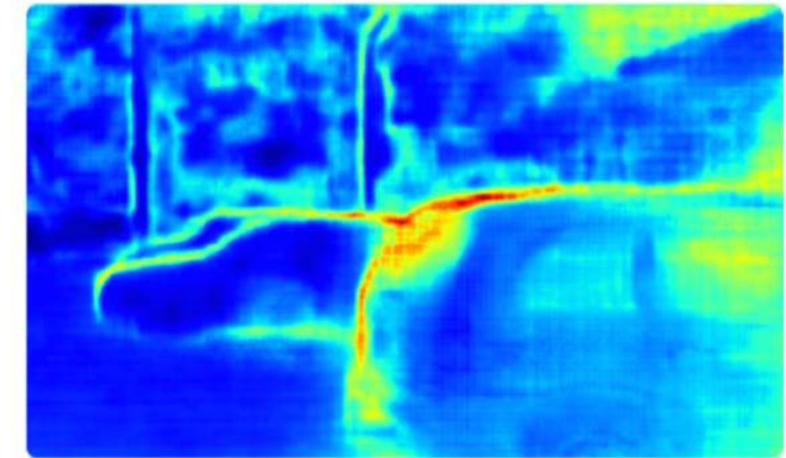
# Model Uncertainty Application



Input image



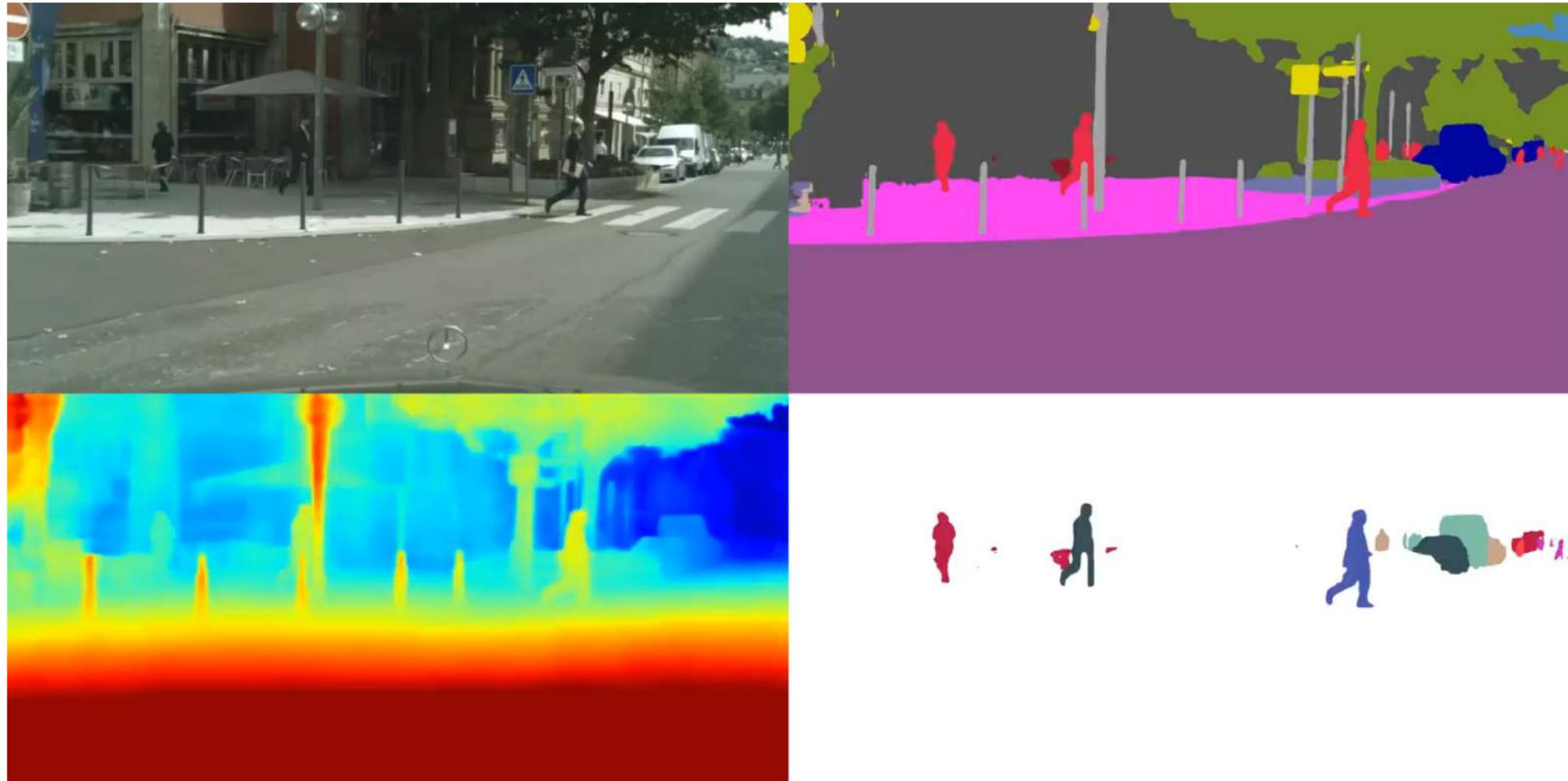
Predicted Depth



Model Uncertainty

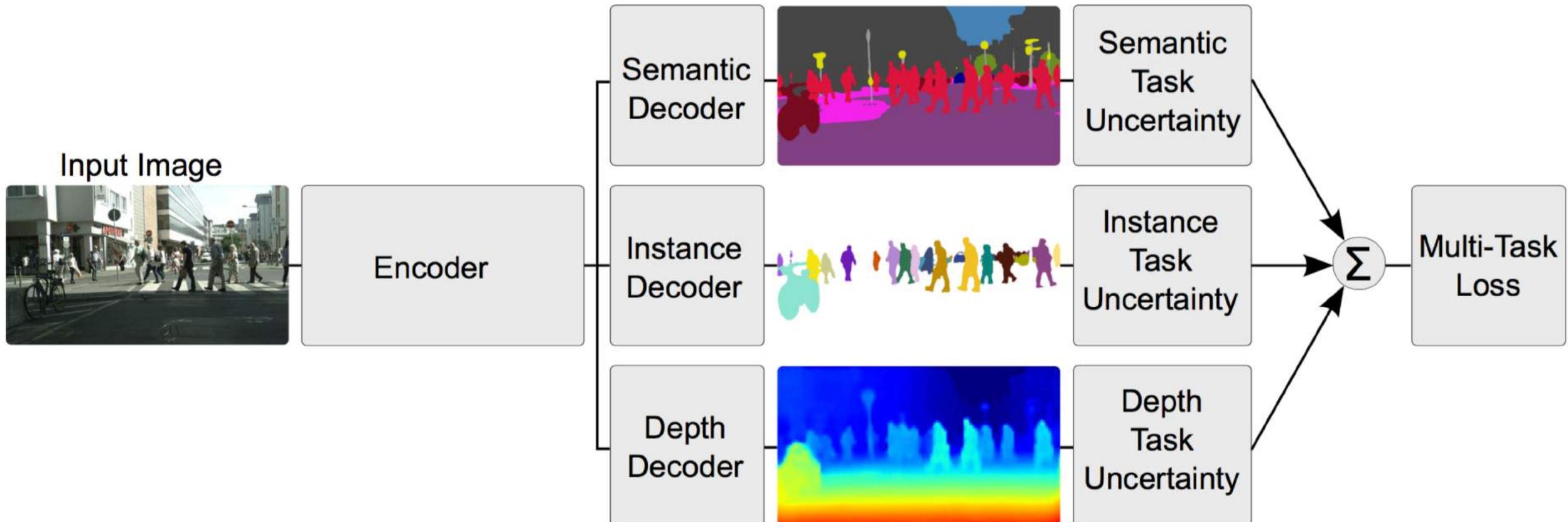
Kendall, Gal, NIPS, 2017.

# Multi-Task Learning Using Uncertainty



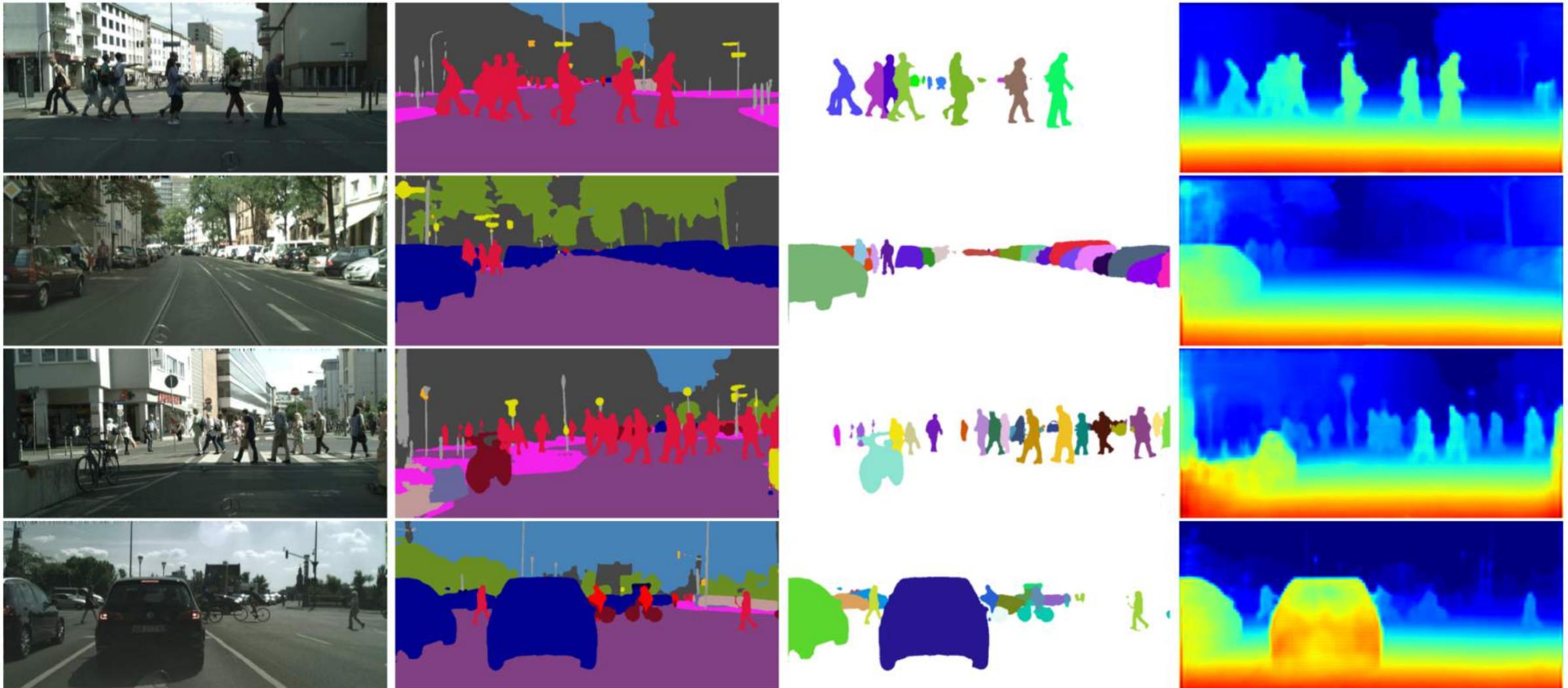
Kendall, et al., CVPR, 2018.

# Multi-Task Learning Using Uncertainty



Kendall, et al., CVPR, 2018.

# Multi-Task Learning Using Uncertainty

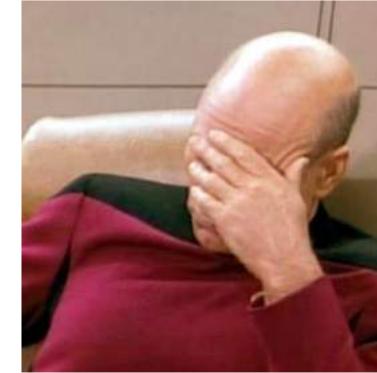
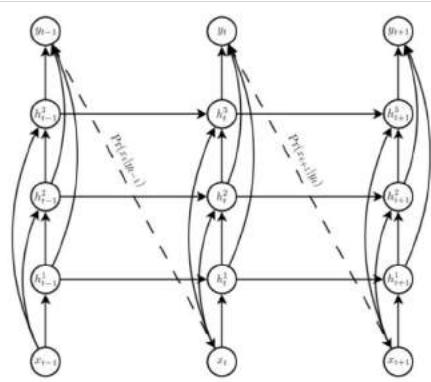


Kendall, et al., CVPR, 2018.

# New Frontiers II: Learning to Learn

# Motivation: Learning to Learn

Standard deep neural networks are optimized for **a single task**



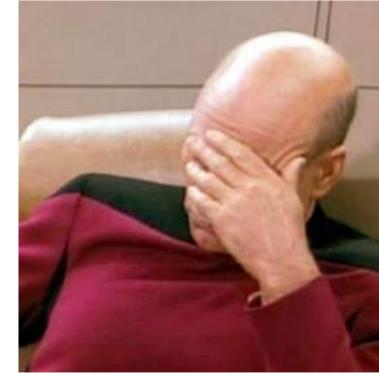
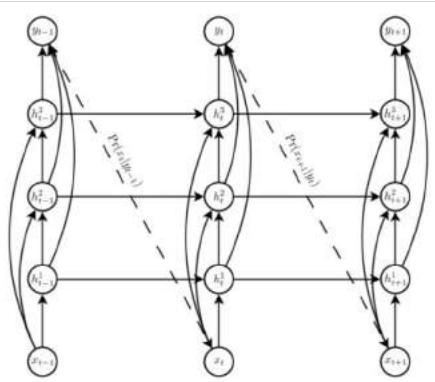
Complexity of models increases

Greater need for specialized engineers

Often require **expert knowledge** to build an architecture for a given task

# Motivation: Learning to Learn

Standard deep neural networks are optimized for **a single task**



Complexity of models increases

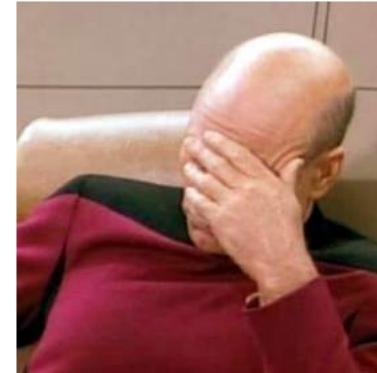
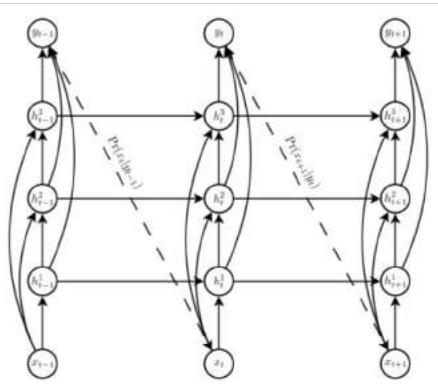
Greater need for specialized engineers

Often require **expert knowledge** to build an architecture for a given task

Build a learning algorithm that **learns which model** to use to solve a given problem

# Motivation: Learning to Learn

Standard deep neural networks are optimized for **a single task**



Complexity of models increases

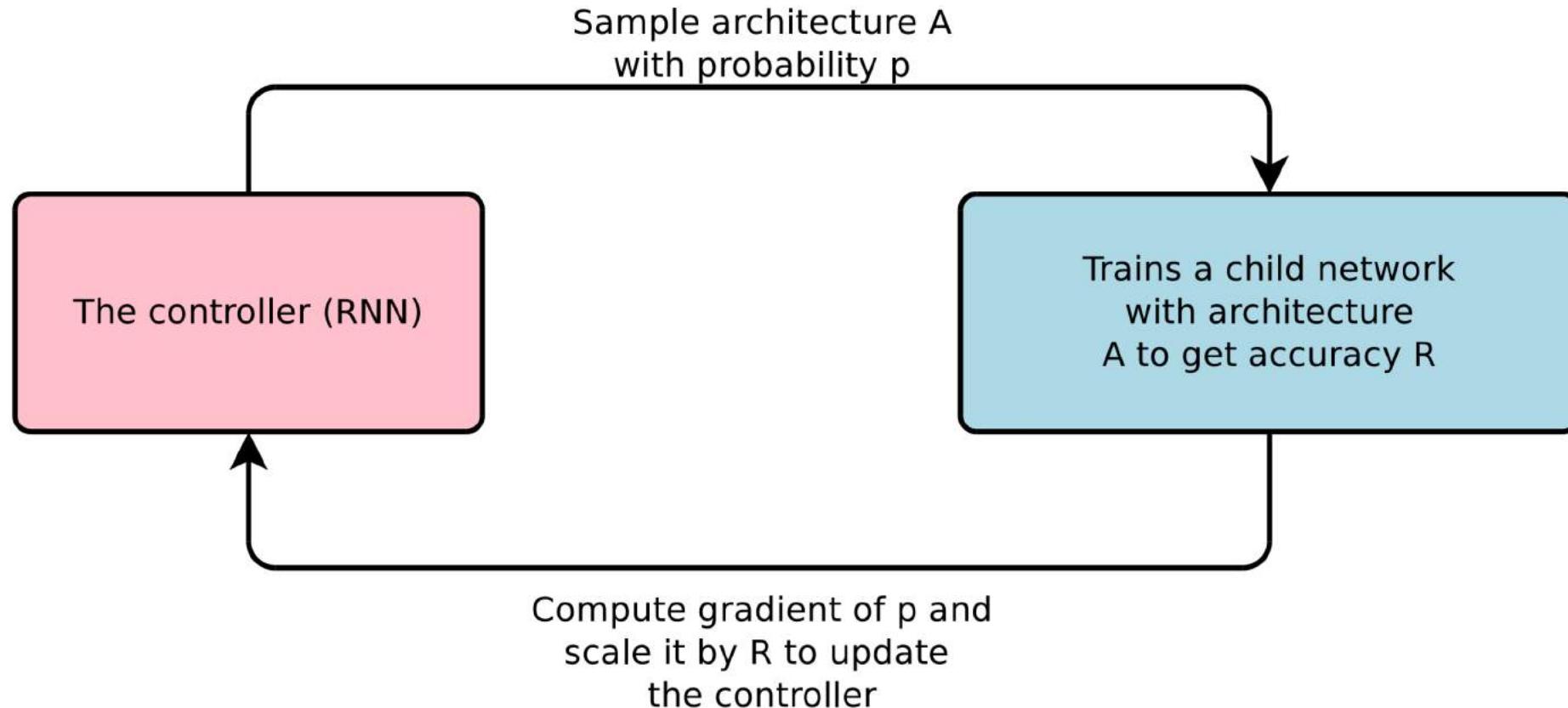
Greater need for specialized engineers

Often require **expert knowledge** to build an architecture for a given task

Build a learning algorithm that **learns which model** to use to solve a given problem

## AutoML

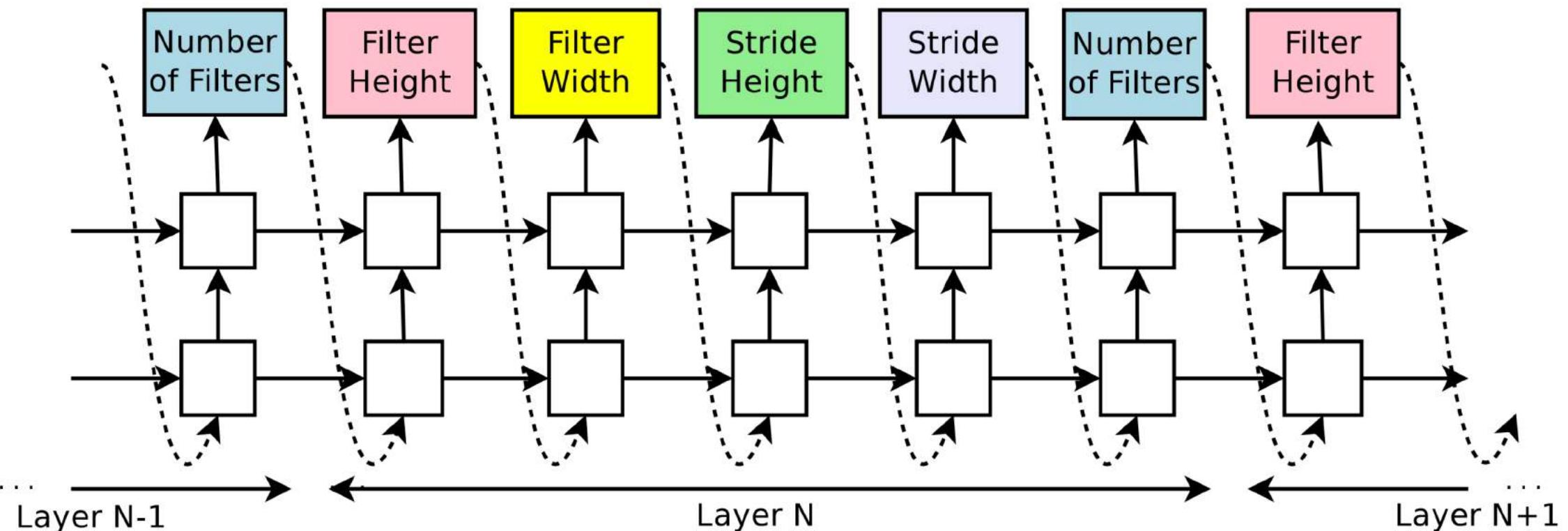
# AutoML: Learning to Learn



Zoph and Le, ICLR 2017.

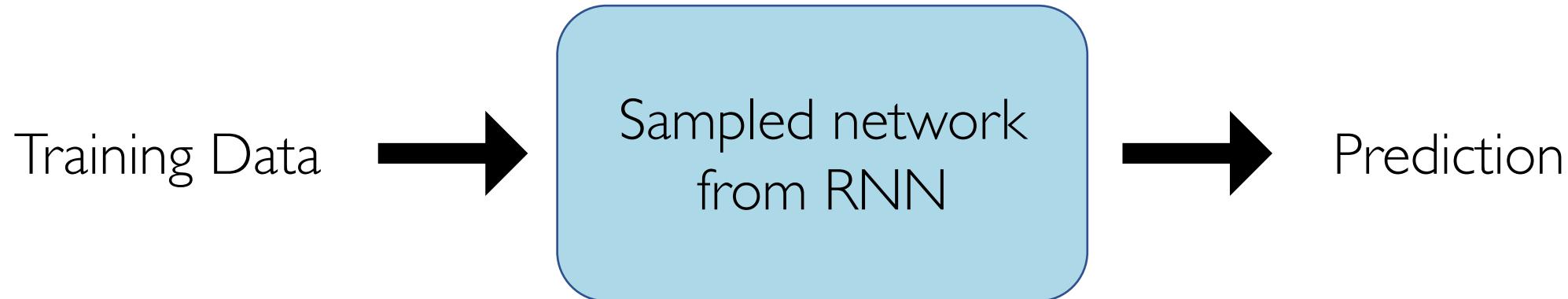
# AutoML: Model Controller

At each step, the model samples a brand new network



Zoph and Le, ICLR 2017.

# AutoML: The Child Network



Compute final accuracy on this dataset.  
Update RNN controller based on the accuracy of the child network after training.

Zoph and Le, ICLR 2017.

# AutoML on the Cloud



## AutoML Vision<sup>BETA</sup>

Start with as little as a few dozen photographic samples, and Cloud AutoML will do the rest.



## AutoML Natural Language<sup>BETA</sup>

Automatically predict text categories through either single or multi-label classification.



## AutoML Translation<sup>BETA</sup>

Upload translated language pairs to train your own custom model.

Google Cloud.



# AutoML Spawns a Powerful Idea

- Design an AI algorithm that can build new models capable of solving a task
- Reduces the need for experienced engineers to design the networks
- Makes deep learning more accessible to the public

Connection to  
Artificial General Intelligence:  
**the ability to intelligently  
reason about how we learn**