# CSE 152: Computer Vision

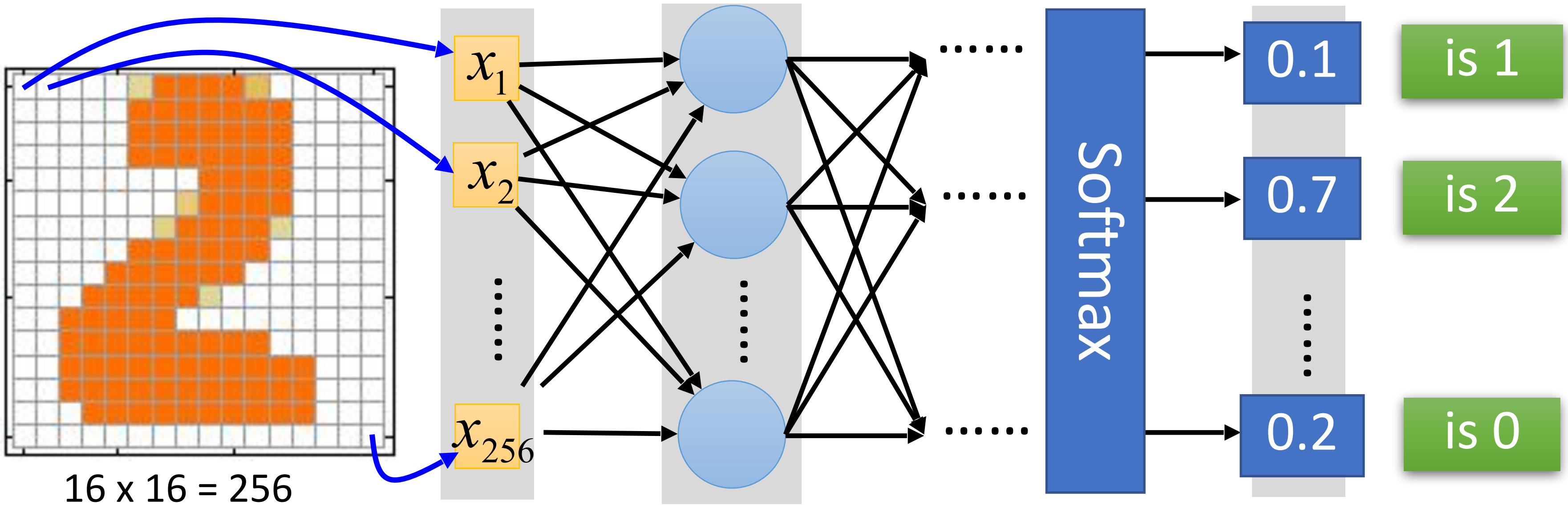
Hao Su

## Lecture 8: Statistical and Optimization Perspectives of Deep Learning

# Optimization of Neural Network

# How to set network parameters

$$\theta = \{W_1, b_1, \ldots, W_n, b_n\}$$

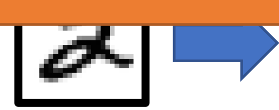



Ink → 1

No ink → 0

Set the network parameters $\theta$ such that ……

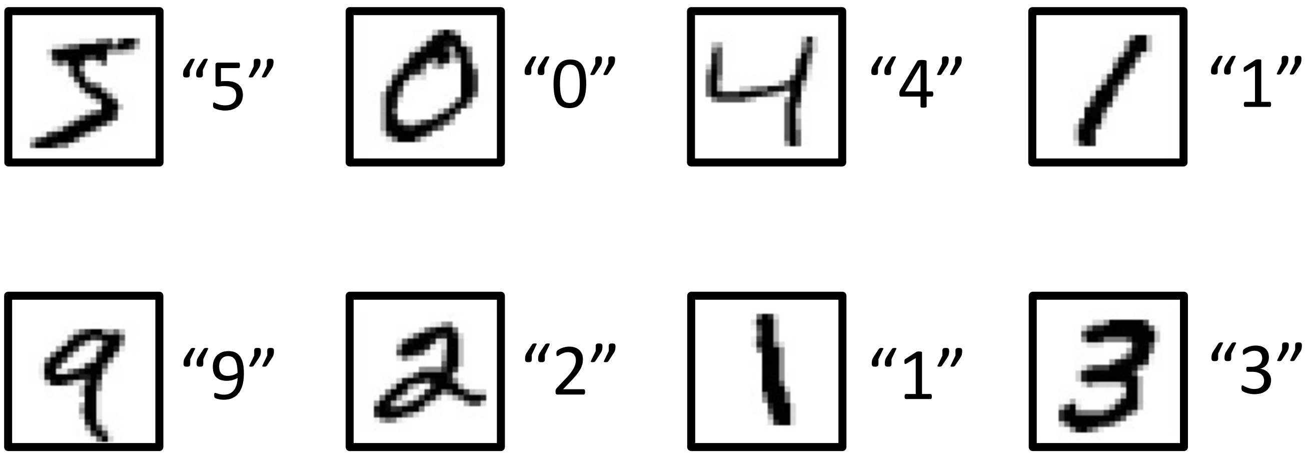
Input ... m value

Input:  $y_2$ has the maximum value

How to let the neural network achieve this

# Training Data

- Preparing training data: images and their labels

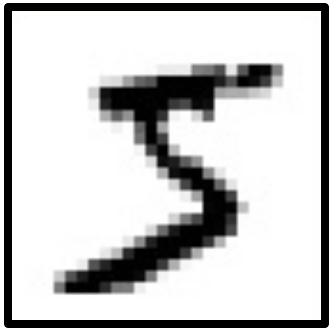



Using the training data to find the network parameters.

# Formalize

- Preparing training data: images and their labels

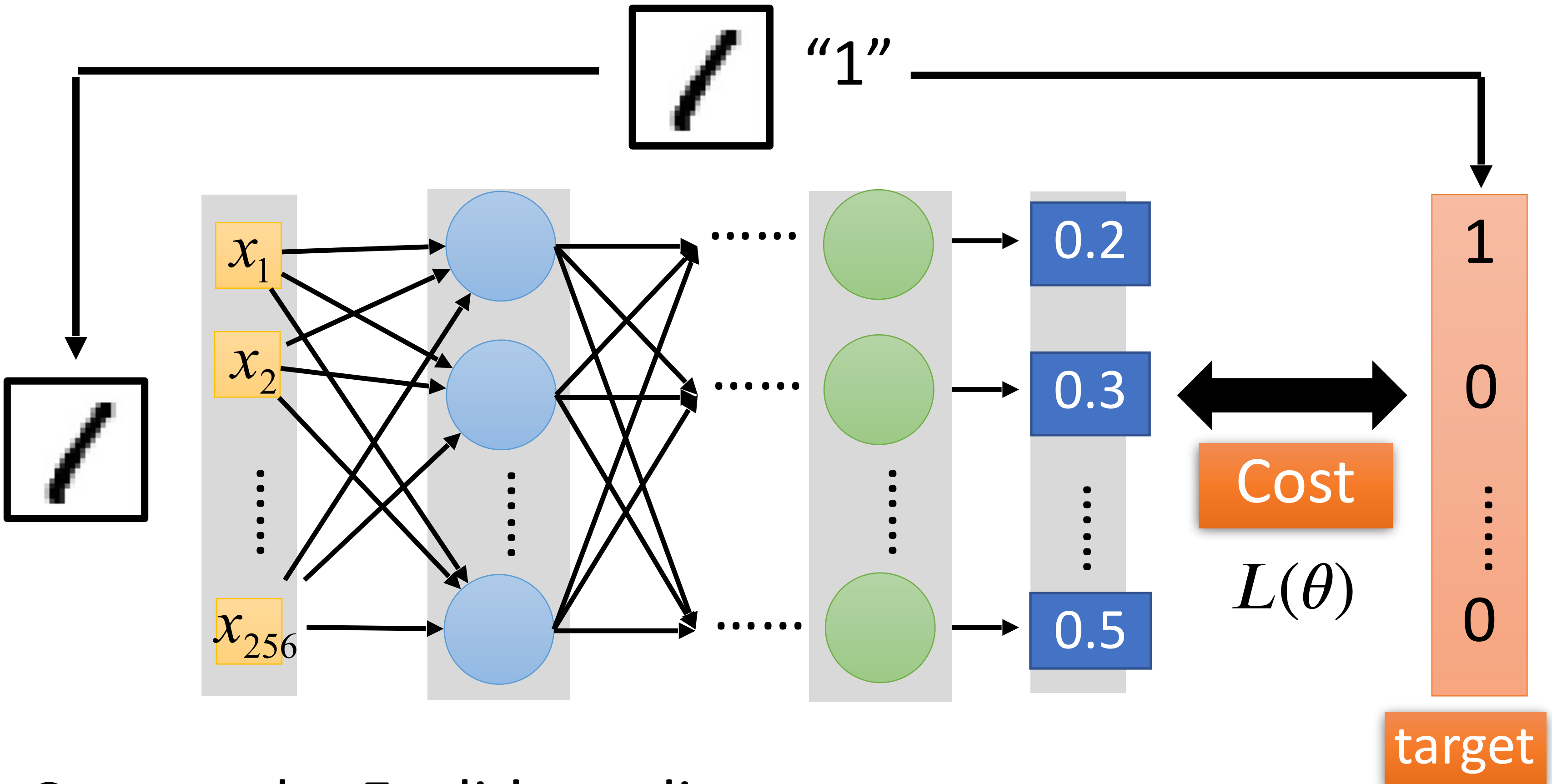
$x:$



$$y_0^{gt} = 0,\ \ldots,\ y_4^{gt} = 0,\ y_5^{gt} = 1,\ y_6^{gt} = 0,\ \ldots,\ y_9^{gt} = 0$$

# Cost

Given a set of network parameters $\theta$, each example has a cost value.




Cost can be Euclidean distance or cross entropy of the network output and target

# Soft-Entropy Loss

The predicted score of groundtruth label category is larger than other categories:

$$y_{label} > y_j \text{ for any } j \neq label$$

How to set up a loss for this goal?

# Soft-Entropy Loss

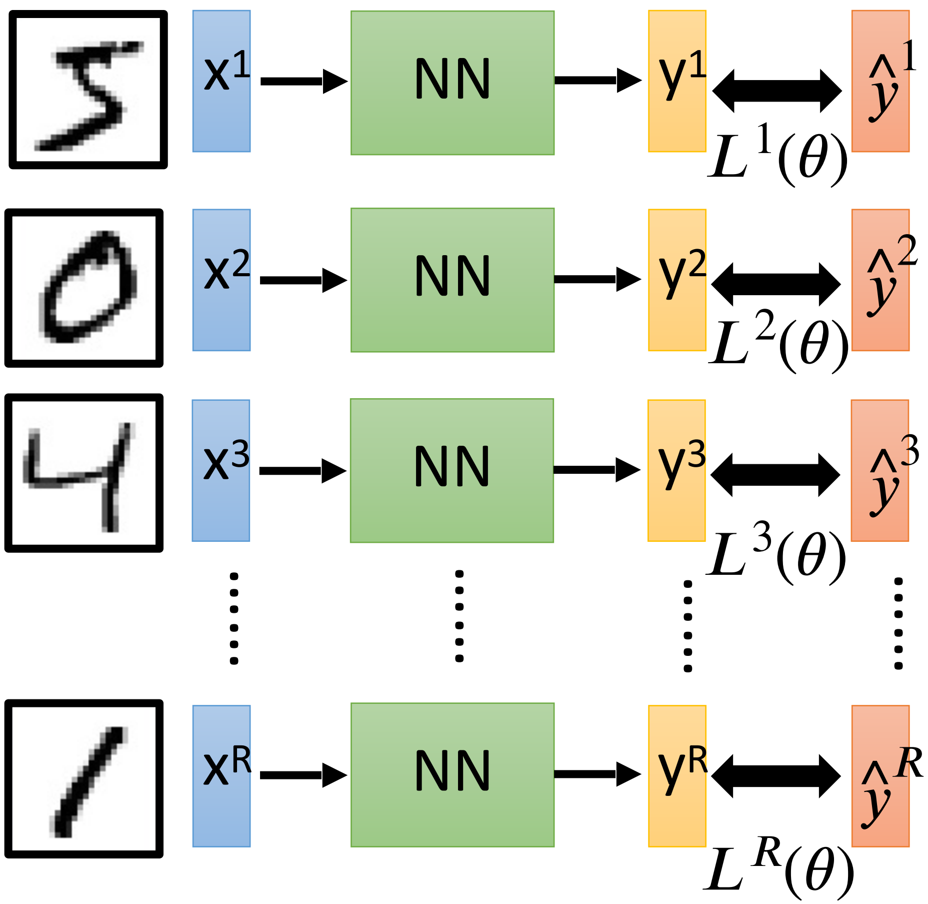
Let $$y_{label}(x;\theta) = \frac{e^{f(x;\theta)_{label}}}{\sum_j e^{f(x;\theta)_j}}$$

We minimize the loss

$$L(\theta) = -\log\ y_{label}(\theta)$$

# Total Cost

For all training data …




Total Cost:

$$C(\theta) = \sum_{r=1}^{R} L^r(\theta)$$

How bad the network parameters $\theta$ is on this task

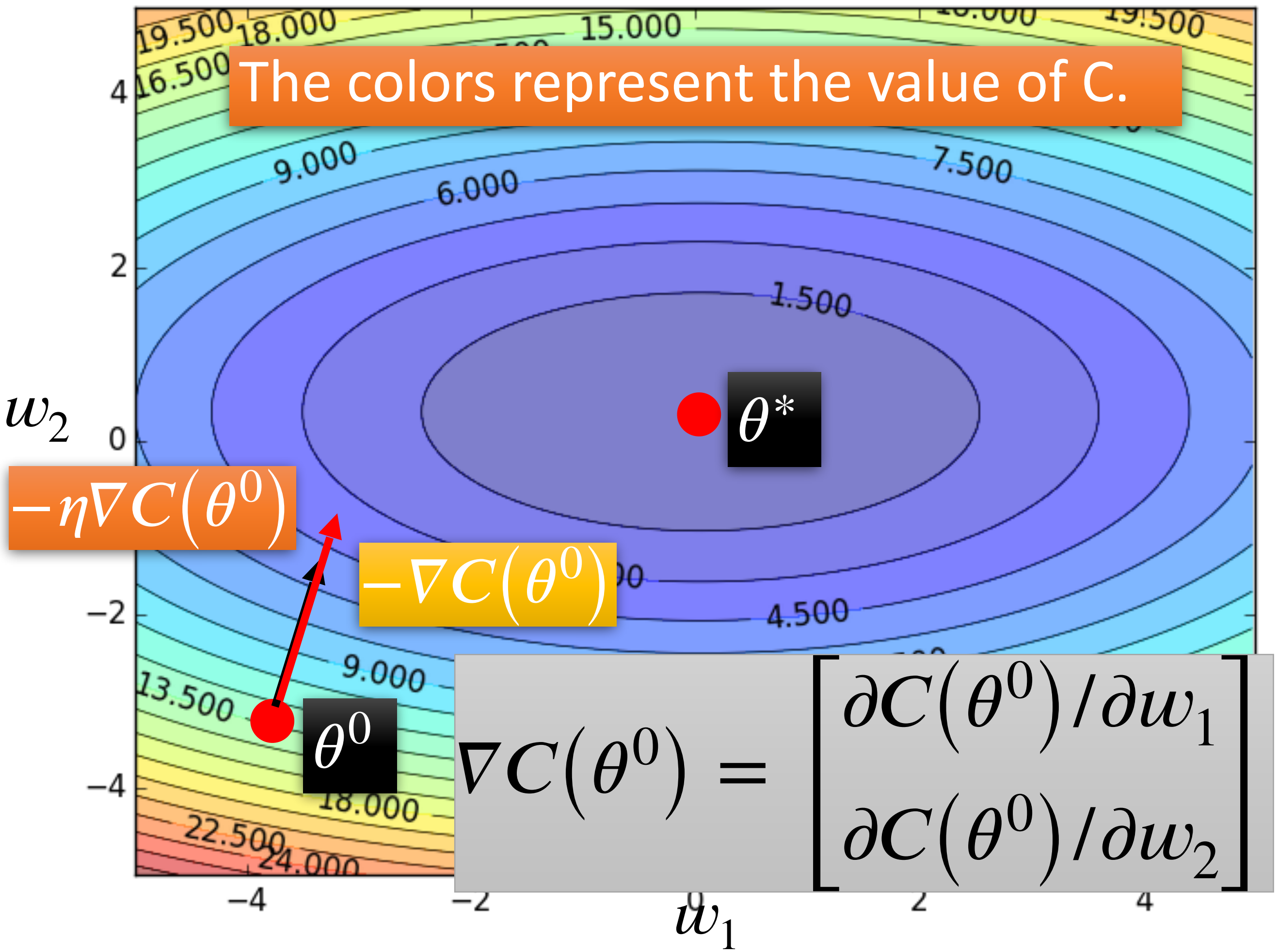
Find the network parameters $\theta^*$ that minimize this value

# Gradient Descent

Assume there are only two parameters $w_1$ and $w_2$ in a network.

$$\theta = \{w_1, w_2\}$$

Error Surface




Randomly pick a starting point $\theta^0$

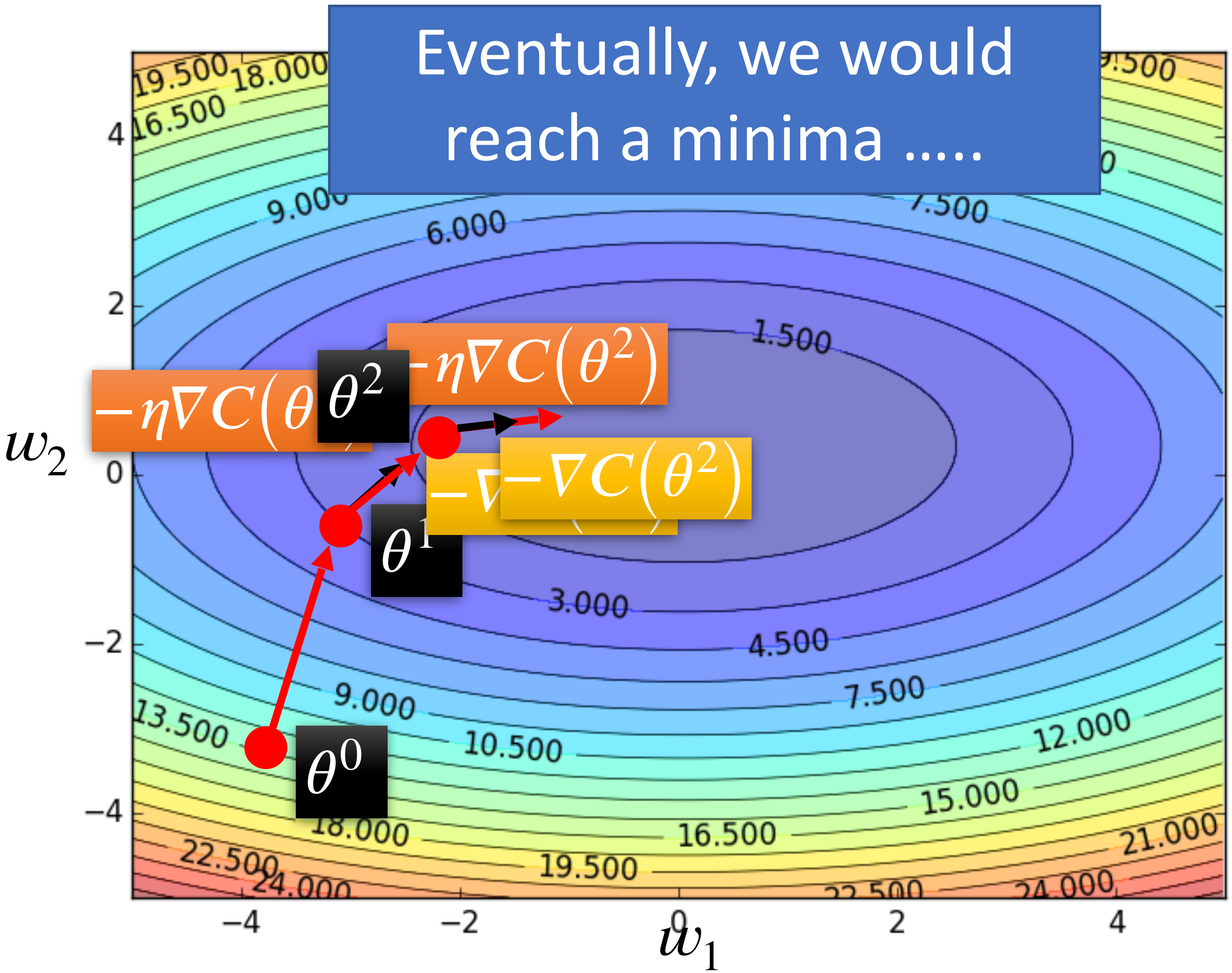
Compute the negative gradient at $\theta^0$

$\Rightarrow -\nabla C(\theta^0)$

Times the learning rate $\eta$

$\Rightarrow -\eta\nabla C(\theta^0)$

# Gradient Descent




Randomly pick a starting point $\theta^0$

Compute the negative gradient at $\theta^0$

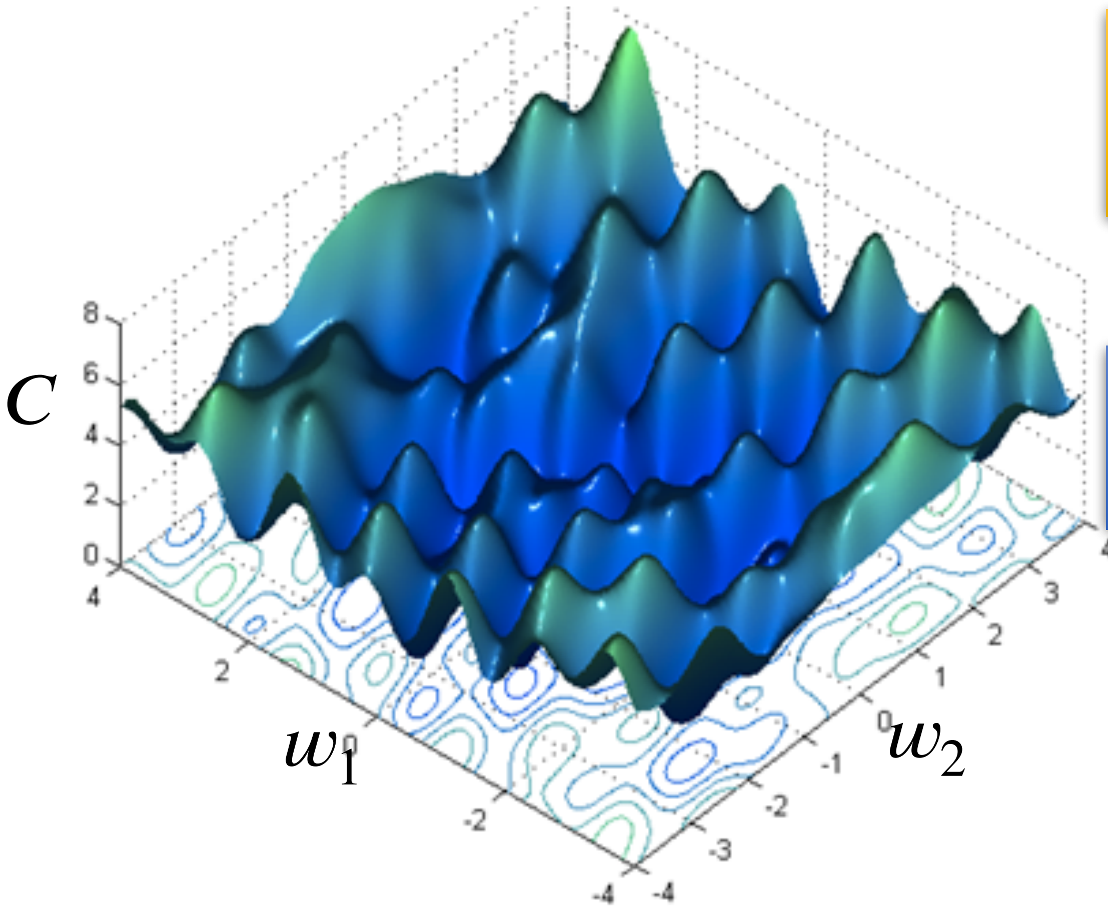
$\Rightarrow -\nabla C(\theta^0)$

Times the learning rate $\eta$

$\Rightarrow -\eta\nabla C(\theta^0)$

# Local Minima

- Gradient descent never guarantee global minima

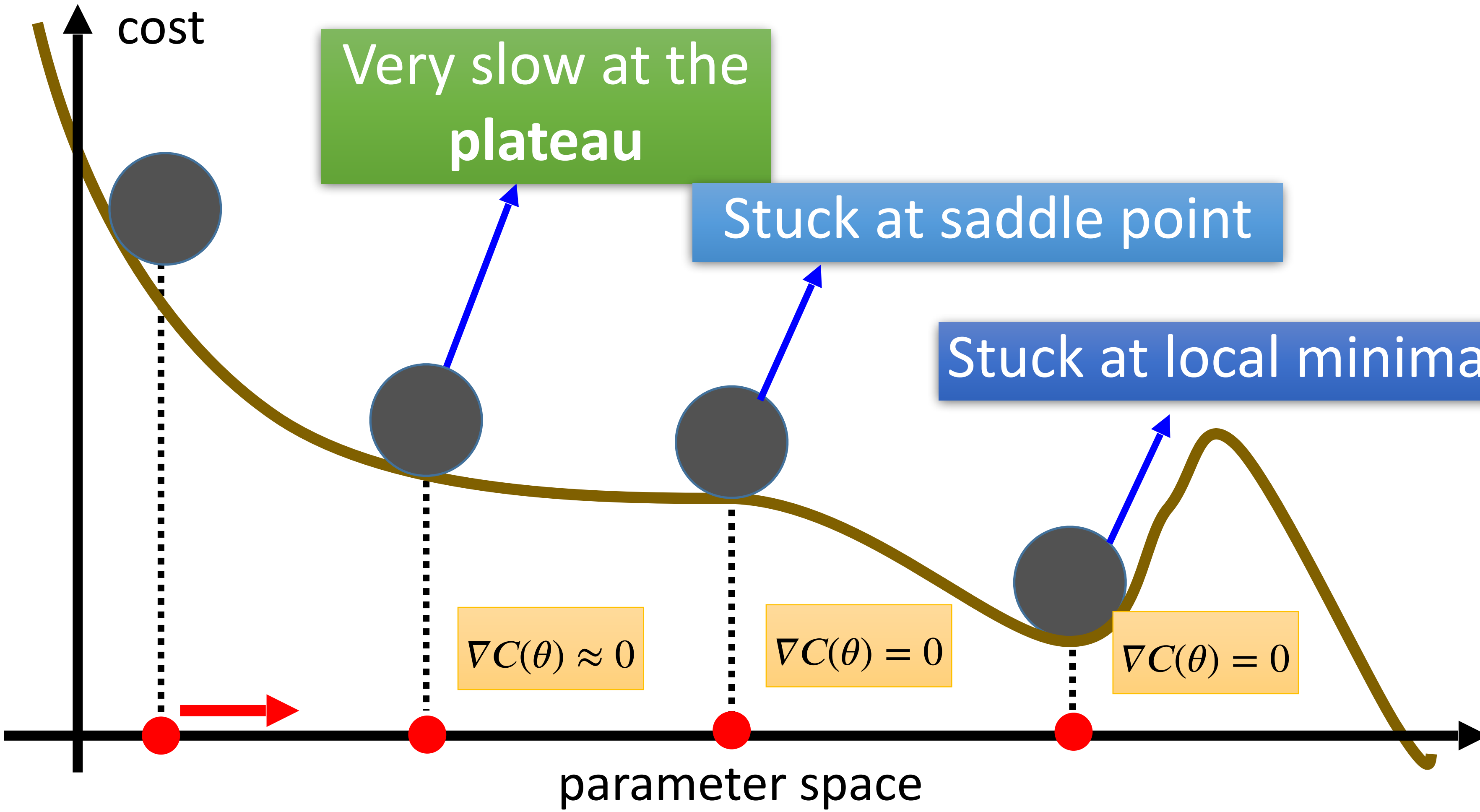



Different initial point $\theta^0$

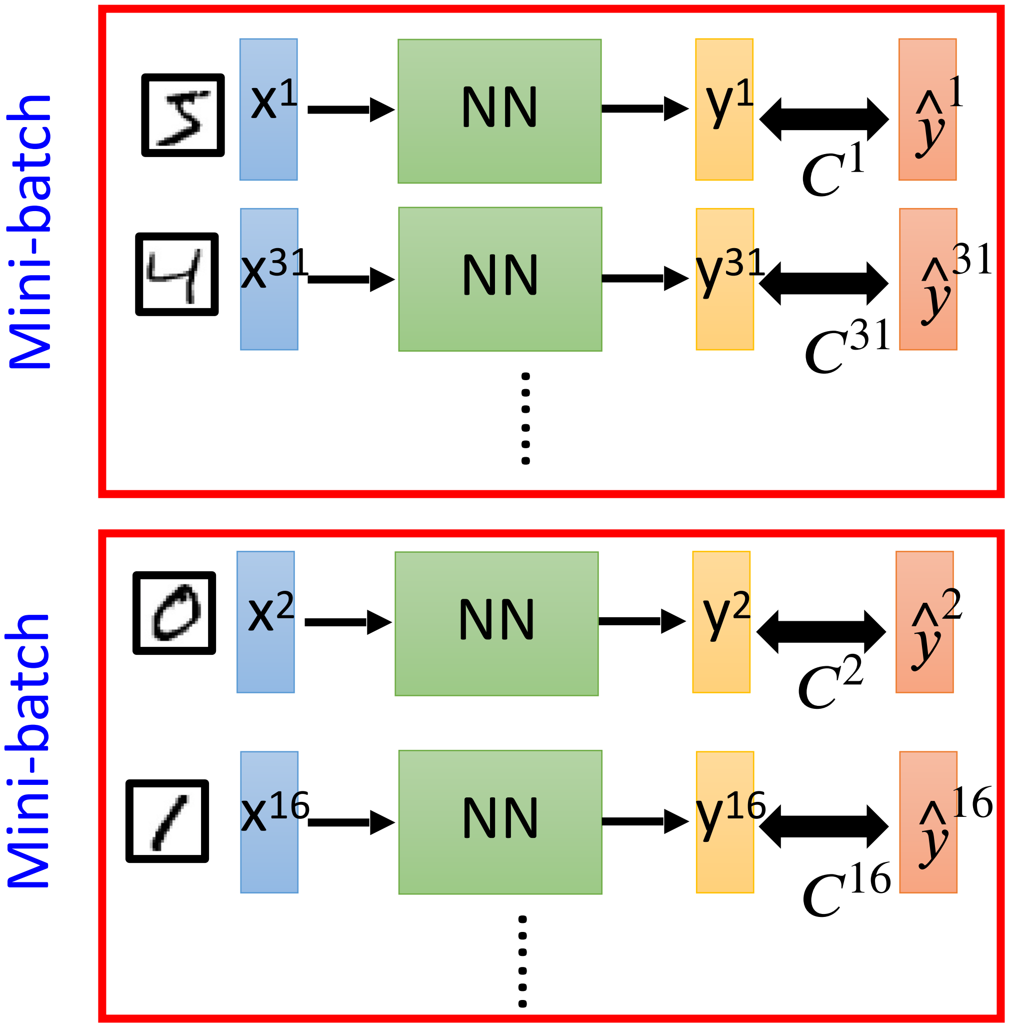
Reach different minima, so different results

Who is Afraid of Non-Convex Loss Functions?
http://videolectures.net/eml07_lecun_wia/

# Besides local minima ……



# Stochastic Gradient Descent (SGD)




- Randomly initialize $\theta^0$
- Pick the 1st batch

$$C = C^1 + C^{31} + \cdots$$

$$\theta^1 \leftarrow \theta^0 - \eta \nabla C(\theta^0)$$

- Pick the 2nd batch

$$C = C^2 + C^{16} + \cdots$$

$$\theta^2 \leftarrow \theta^1 - \eta \nabla C(\theta^1)$$

⋮

- Until all mini-batches have been picked

one epoch

Repeat the above process

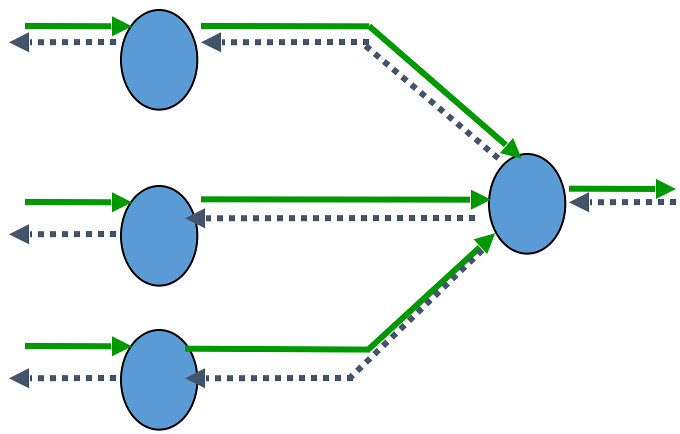
# Backpropagation

- A network can have millions of parameters.
  - Backpropagation is the way to compute the gradients efficiently (not today)
  - Ref: http://speech.ee.ntu.edu.tw/~tlkagk/courses/MLDS_2015_2/Lecture/DNN%20backprop.ecm.mp4/index.html
- Many toolkits can compute the gradients automatically

Ref: http://speech.ee.ntu.edu.tw/~tlkagk/courses/MLDS_2015_2/Lecture/Theano%20DNN.ecm.mp4/index.html

# Back Propagation

- Back-propagation training algorithm



*Network activation*
*Forward Step*

*Error propagation*
*Backward Step*

- Backprop adjusts the weights of the NN in order to minimize the network total error.

# Why Deep?

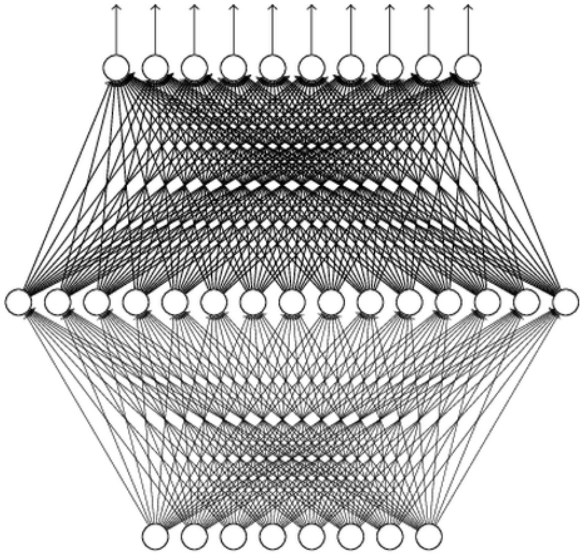
# Universality Theorem

Any continuous function f

$$f : R^N \rightarrow R^{\mathrm{M}}$$

Can be realized by a network with one hidden layer

(given **enough** hidden neurons)



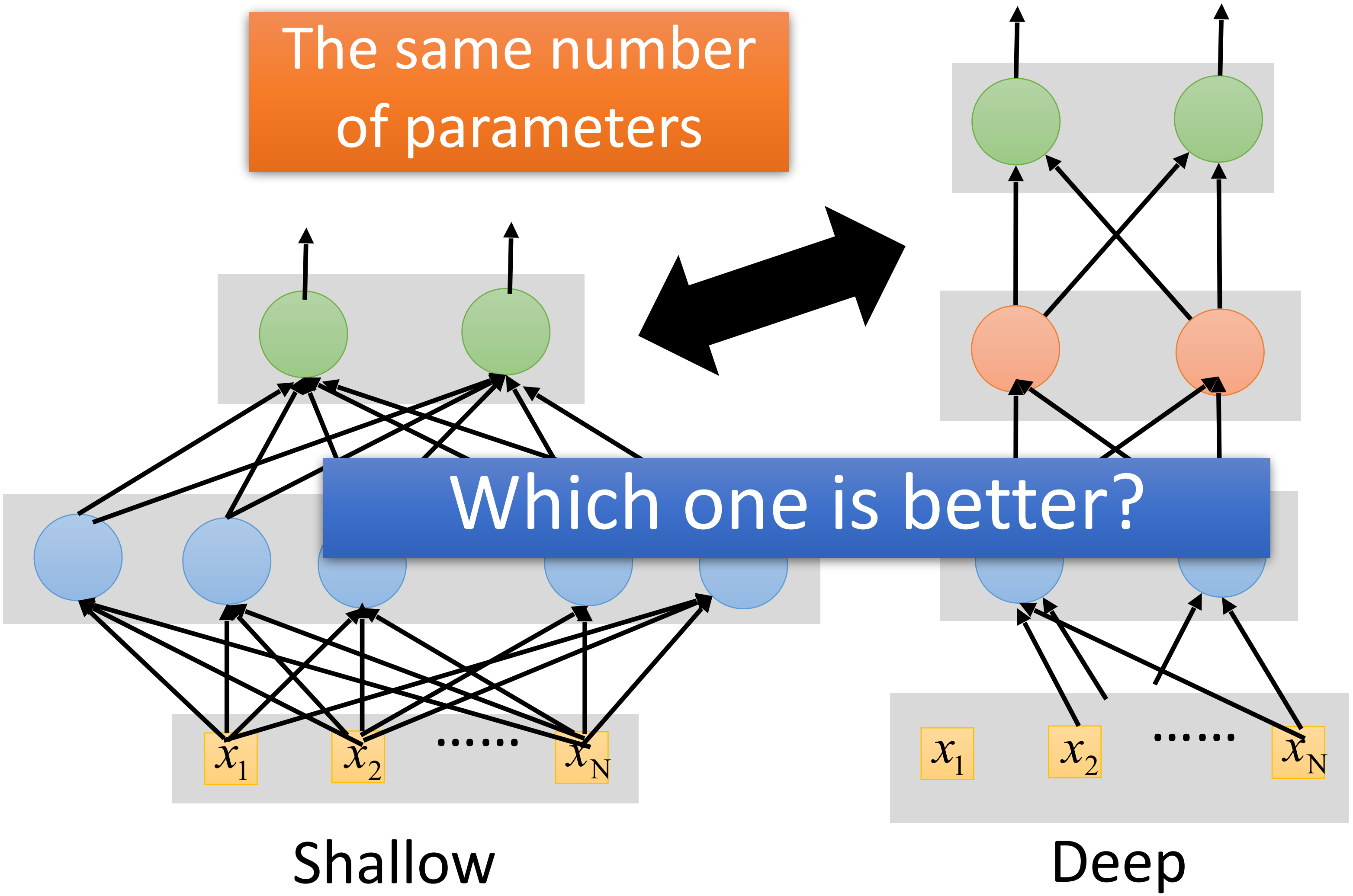
Reference for the reason: http://neuralnetworksanddeeplearning.com/chap4.html

# The Unreasonable Effectiveness of Gradient Descent

- While the loss function for neural networks is highly non-convex, empirically (and theoretically), we can show that, with many hidden neurons, the value of local minima are almost as **small** as the global minimum

Then why “Deep” neural network not “Fat” neural network?

# Fat + Short v.s. Thin + Tall



# Fat + Short v.s. Thin + Tall

"Why deep" is a very "deep" question!

No simple answer yet, even no fully convincing answer yet!

# Statistical View of Machine Learning

We start from understanding some simple classifiers,
to draw inspiration for understanding neural networks!

# Example Task: Image Classification



(assume given set of discrete labels)
{dog, cat, truck, plane, ...}

→ cat

# An image classifier

```
def classify_image(image):
  # Some magic here?
  return class_label
```

Unlike e.g. sorting a list of numbers,

**no obvious way** to hard-code the algorithm for recognizing a cat, or other classes.

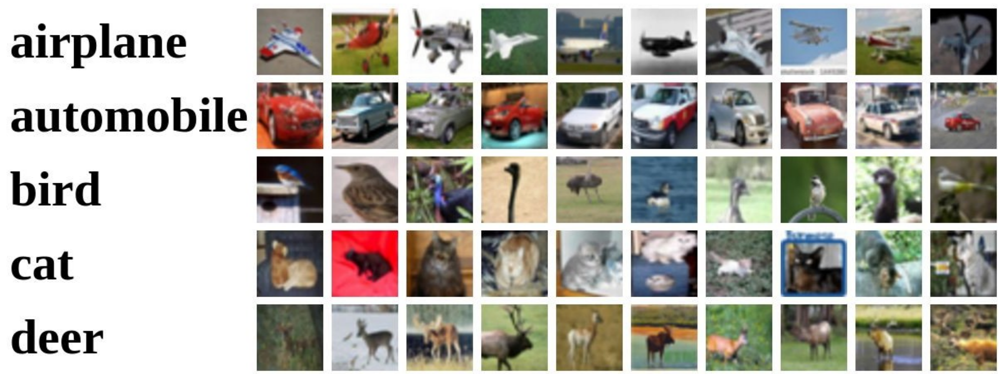
# Machine Learning: Data-Driven Approach

1. Collect a dataset of images and labels
2. Use Machine Learning to train a classifier
3. Evaluate the classifier on new images

**Example training set**

```python
def train(images, labels):
  # Machine learning!
  return model
```

```python
def predict(model, test_images):
  # Use model to predict labels
  return test_labels
```



# First classifier: Nearest Neighbor

```
def train(images, labels):
  # Machine learning!
  return model
```

→ Memorize all data and labels

```
def predict(model, test_images):
  # Use model to predict labels
  return test_labels
```

→ Predict the label of the most similar training image

# Example Dataset: CIFAR10

**10** classes
**50,000** training images
**10,000** testing images


Alex Krizhevsky, "Learning Multiple Layers of Features from Tiny Images", Technical Report, 2009.

# Example Dataset: CIFAR10

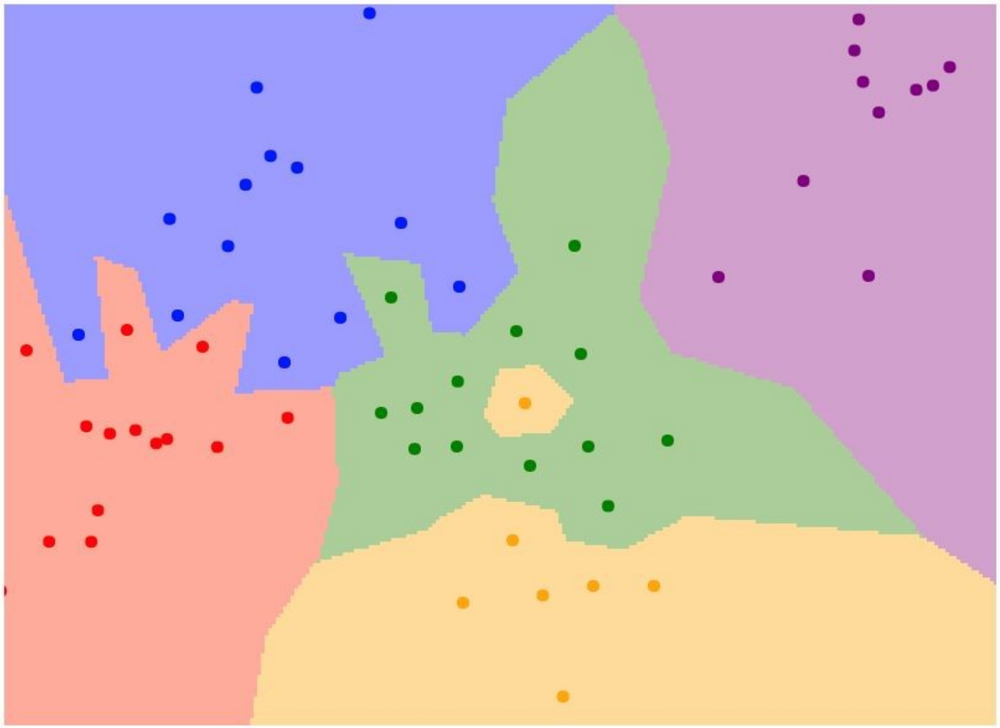
**10** classes
**50,000** training images
**10,000** testing images


Test images and nearest neighbors

Alex Krizhevsky, “Learning Multiple Layers of Features from Tiny Images”, Technical Report, 2009.

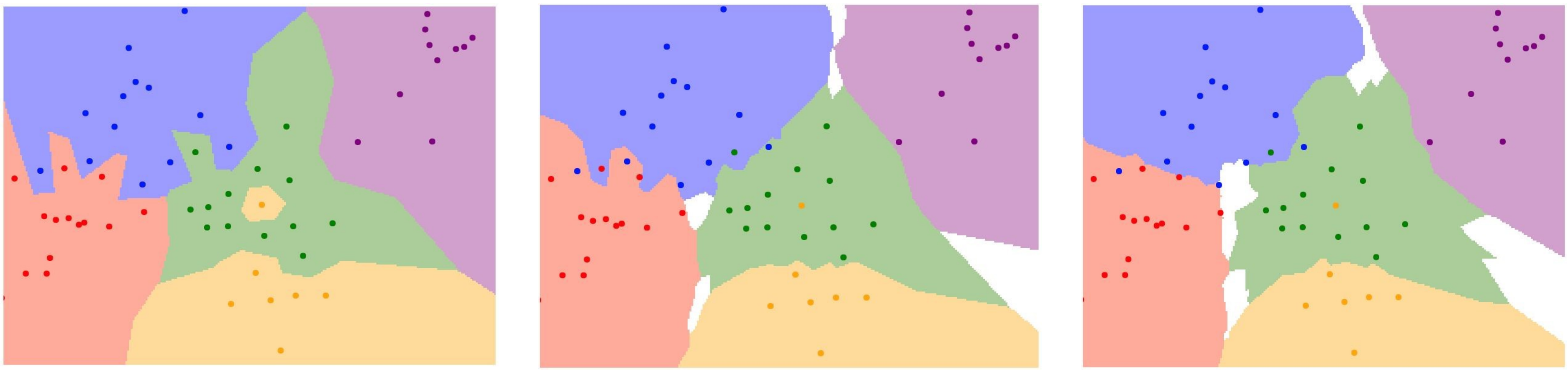
# What does Nearest Neighbor look like?

# What does Nearest Neighbor look like?



# K-Nearest Neighbors

Instead of copying label from nearest neighbor,
take **majority vote** from K closest points

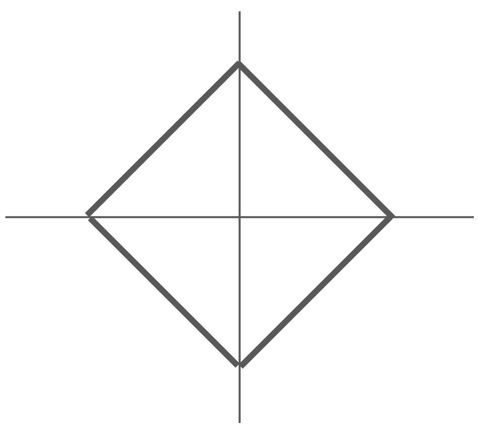


K = 1 K = 3 K = 5

# K-Nearest Neighbors: Distance Metric

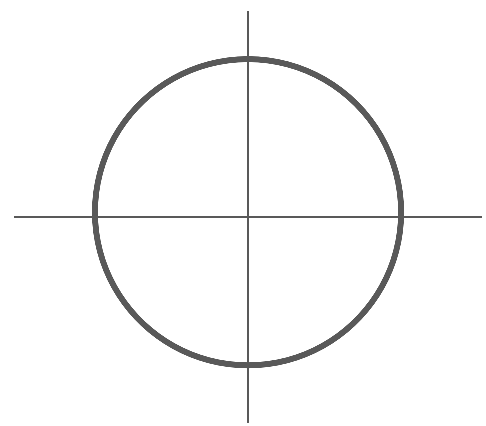
L1 (Manhattan) distance

$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$



L2 (Euclidean) distance

$$d_2(I_1, I_2) = \sqrt{\sum_p \left(I_1^p - I_2^p\right)^2}$$



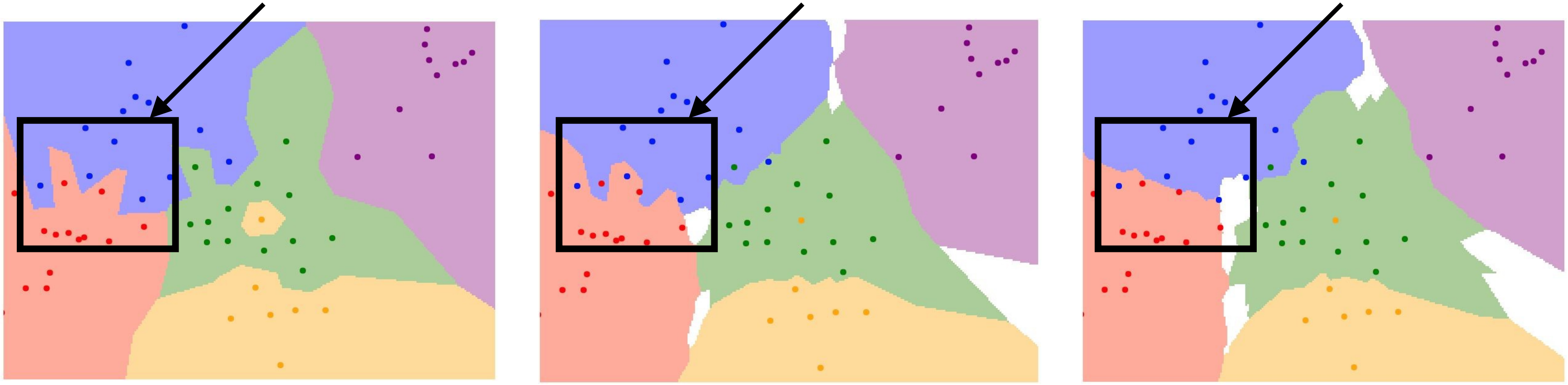
# Hyperparameters

- What is the best value of k to use?  What is the best distance to use?

- These are hyperparameters: choices about the algorithm that we set rather than learn

- The deep v.s. fat choice for neural networks is similarly a choice of "algorithms"

# K-Nearest Neighbors



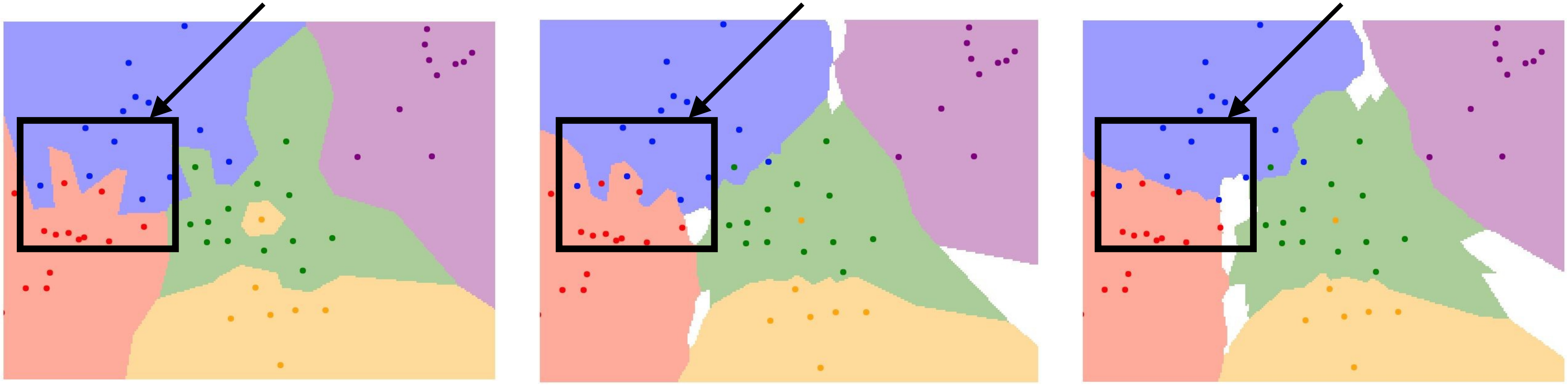
K = 1 K = 3 K = 5

Observations:

Small K (e.g., K=1): every sample matters, sophisticated boundary

Large K (e.g., K=5): voting finds the consensus in the neighborhood, simpler boundary

# K-Nearest Neighbors



K = 1 K = 3 K = 5

Observations:

Small K (e.g., K=1): every sample matters, sophisticated boundary, **high model complexity**

Large K (e.g., K=5): voting finds the consensus in the neighborhood, simpler boundary, **low model complexity**

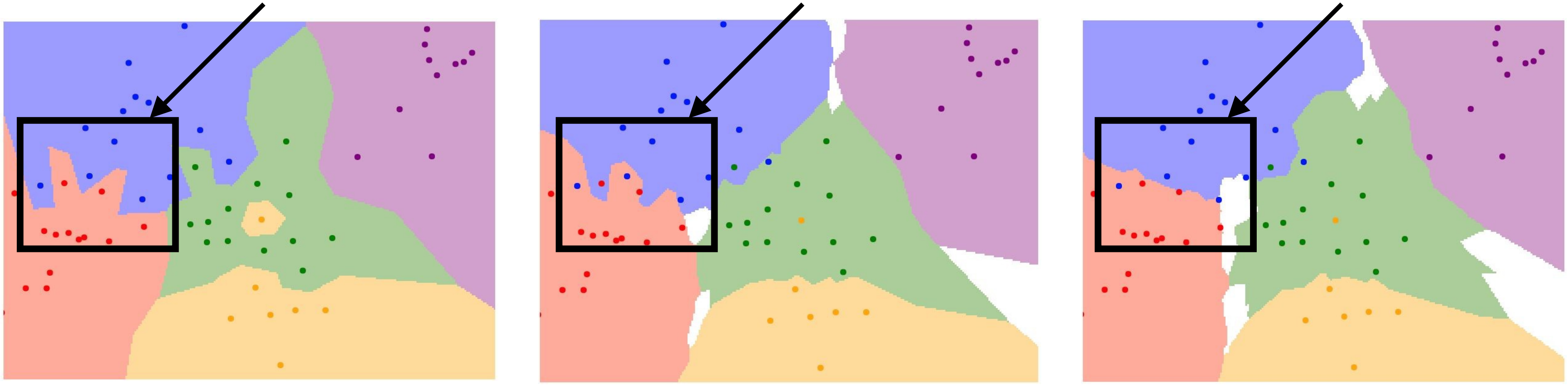
# Bias and Variance

- Bias – error caused because the model lacks the ability to represent the (complex) concept

- Variance – error caused because the learning algorithm overreacts to small changes (noise) in the training data

TotalLoss = Bias + Variance (+ noise)

# K-Nearest Neighbors



K = 1 K = 3 K = 5

**Which one has higher bias? higher variance?**

- Bias – error caused because the model lacks the ability to represent the (complex) concept
- Variance – error caused because the learning algorithm overreacts to small changes (noise) in the training data

# The Power of a Model Building Process

**Weaker Modeling Process**
**( higher bias )**

- Simple Model (e.g. linear, large K in KNN)
- Small Feature Set (e.g. few neurons)
- Constrained Search (e.g. few iterations of gradient descent)

**More Powerful Modeling Process**
**(higher variance)**

- Complex Model (e.g. networks, small K in KNN)
- Large Feature Set (e.g. many neurons)
- Unconstrained Search (e.g. exhaustive search)

# Overfitting v.s. Underfitting

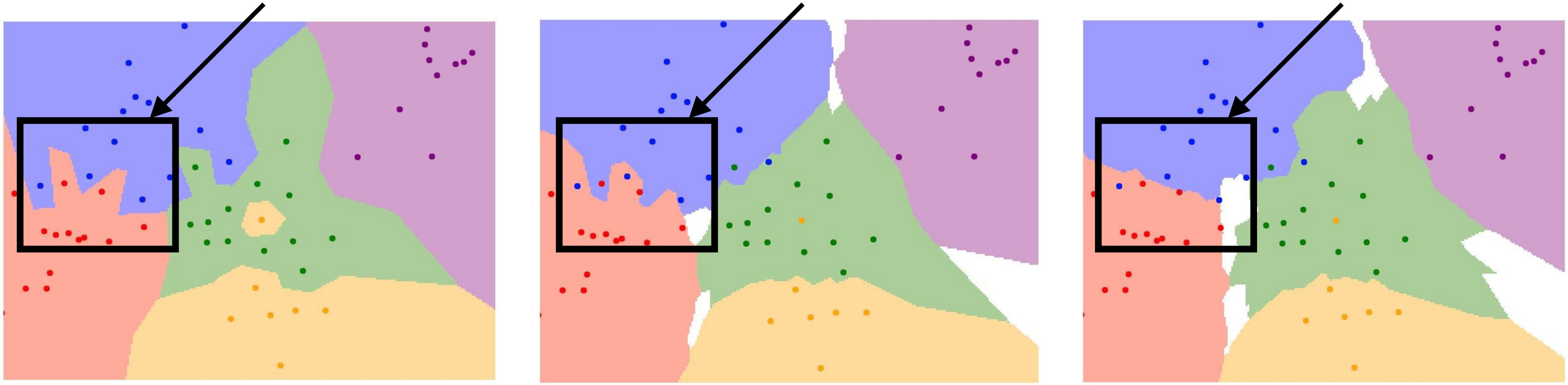
**Overfitting**

- Fitting the data too well
  - Features are noisy / uncorrelated to concept
  - Modeling process very sensitive (powerful)
  - Too much search

**Underfitting**

- Learning too little of the true concept
  - Features don’t capture concept
  - Too much bias in model
  - Too little search to fit model

# K-Nearest Neighbors



K = 1 K = 3 K = 5

**Which one tends to overfit? to underfit?**

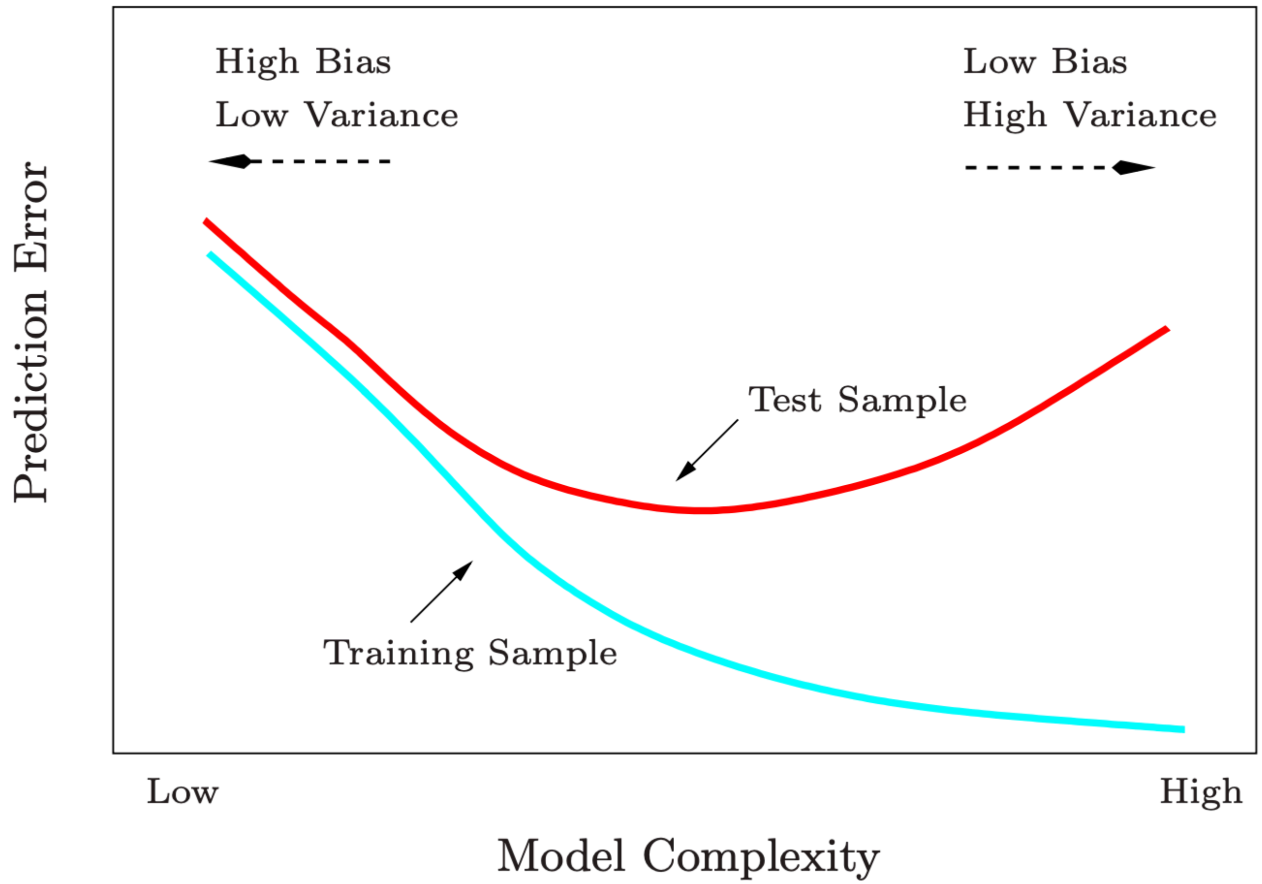



**FIGURE 2.11.** *Test and training error as a function of model complexity.*

*Credit: Elements of Statistical Learning, Second edition*

# Summary of Overfitting and Underfitting

- Bias / Variance tradeoff a primary challenge in machine learning
- Internalize: More powerful modeling is not always better
- Learn to identify overfitting and underfitting
- Tuning parameters & interpreting output correctly is key

# Back to Neural Networks

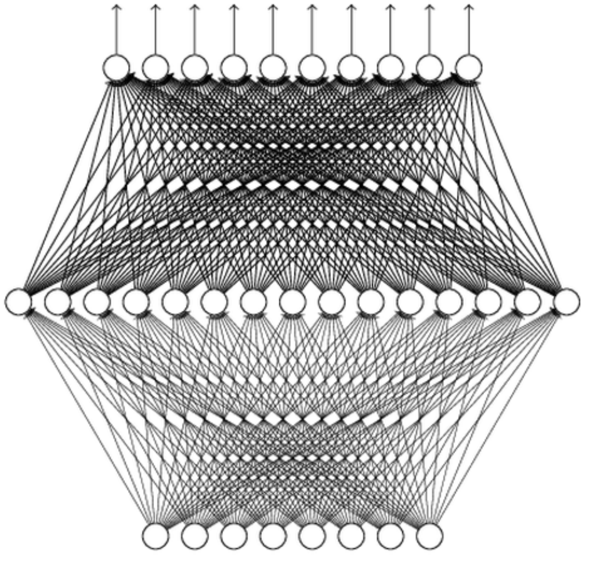
# Recap: Universality Theorem

Any continuous function f

$$f : R^N \rightarrow R^{\mathrm{M}}$$

Can be realized by a network with one hidden layer

(given **enough** hidden neurons)



Reference for the reason: http://neuralnetworksanddeeplearning.com/chap4.html

# Universality is Not Enough

- Neural network has very high capacity (millions of parameters)

- By our basic knowledge of bias-variance tradeoff, so many parameters should imply very low bias, and very high variance. The test loss may not be small.

- Many efforts of deep learning are about mitigating overfitting!

# Address Overfitting for NN

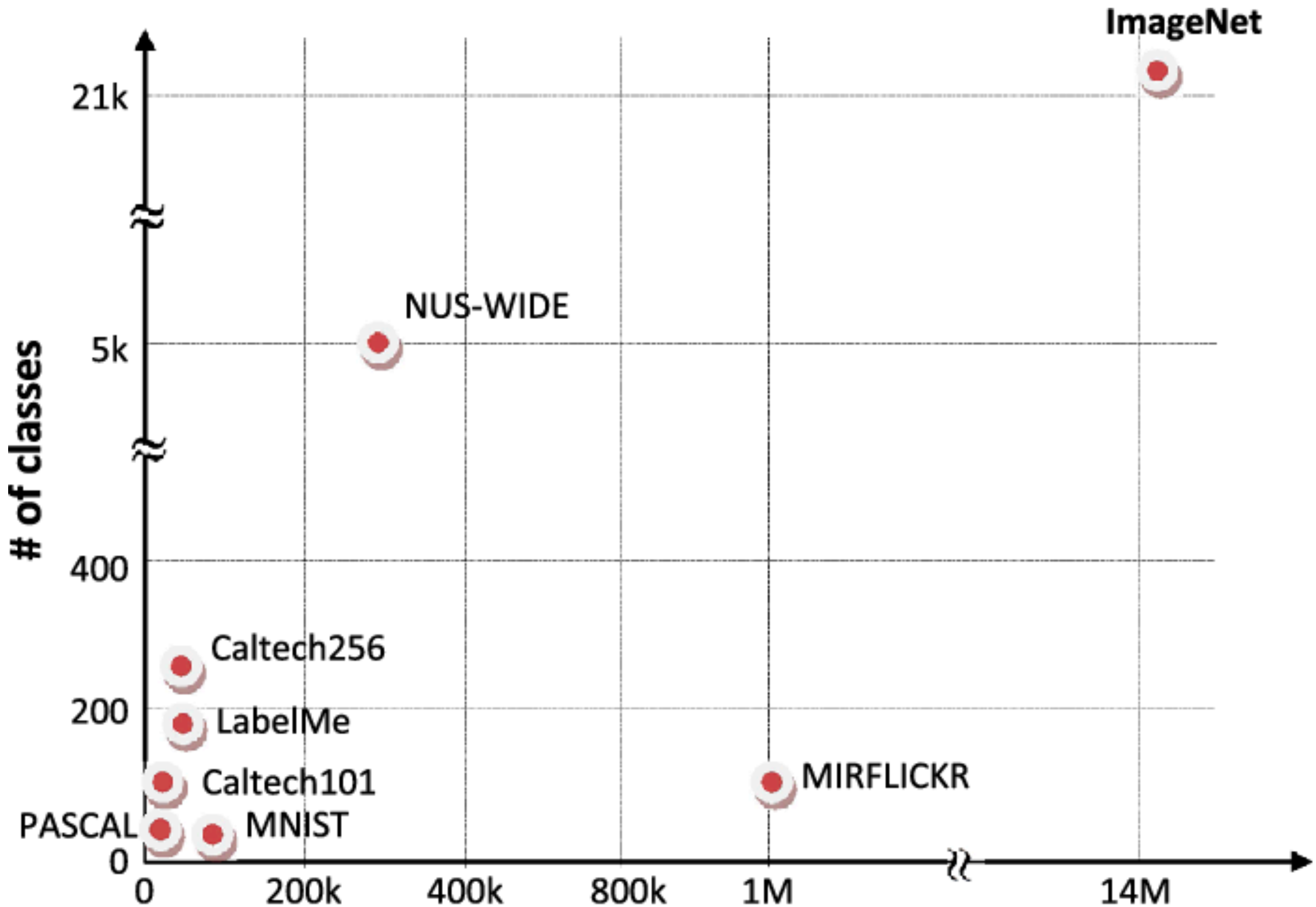
- Use larger training data set

- Design better network architecture

# Address Overfitting for NN

- **Use larger training data set**

- Design better network architecture



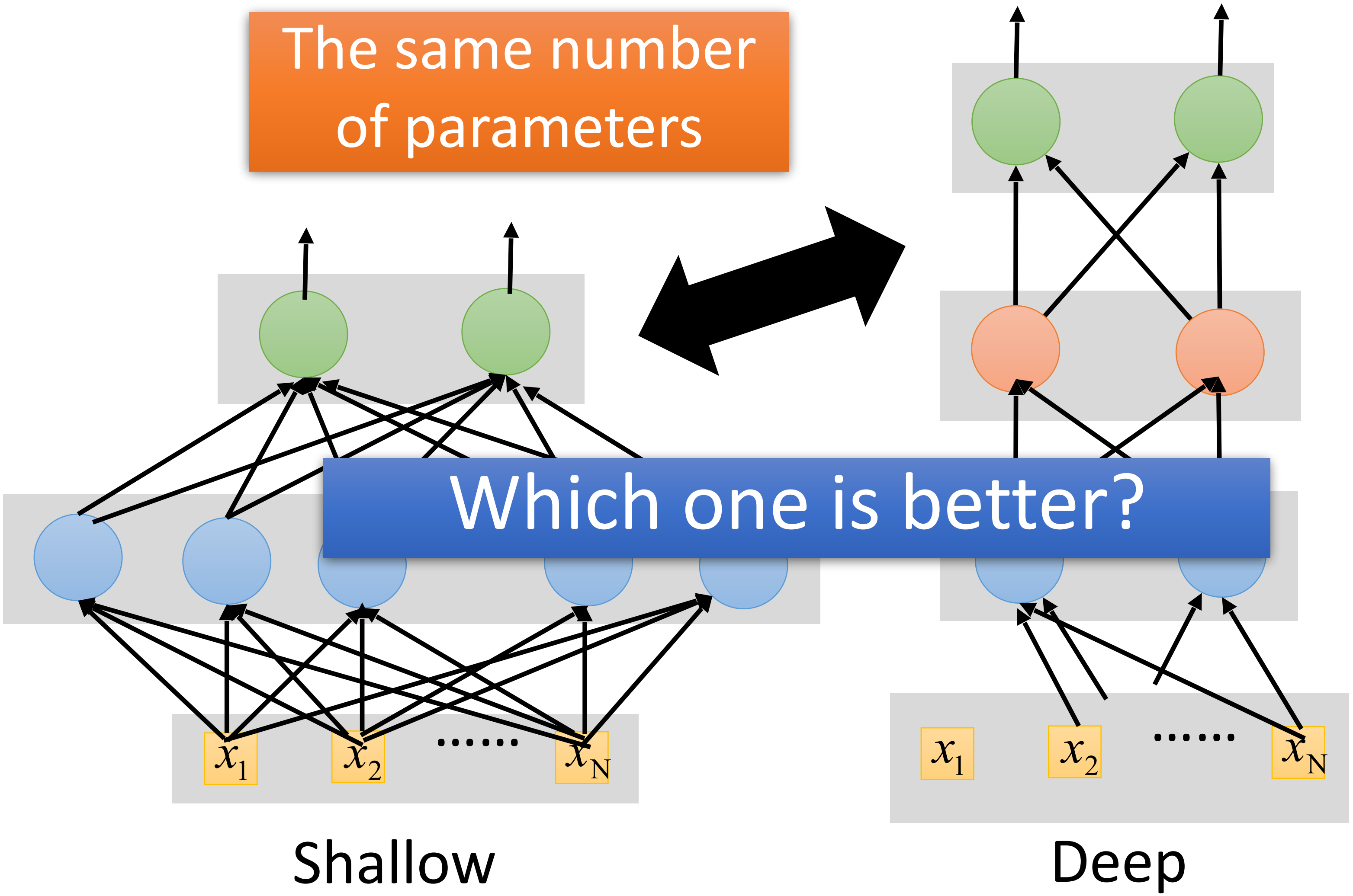

ImageNet Large Scale Visual Recognition Challenge
Russakovsky, Deng, Su, et al. IJCV 2015

# Address Overfitting for NN

- **Design better network architecture**

- Use larger training data set

# Fat + Short v.s. Thin + Tall



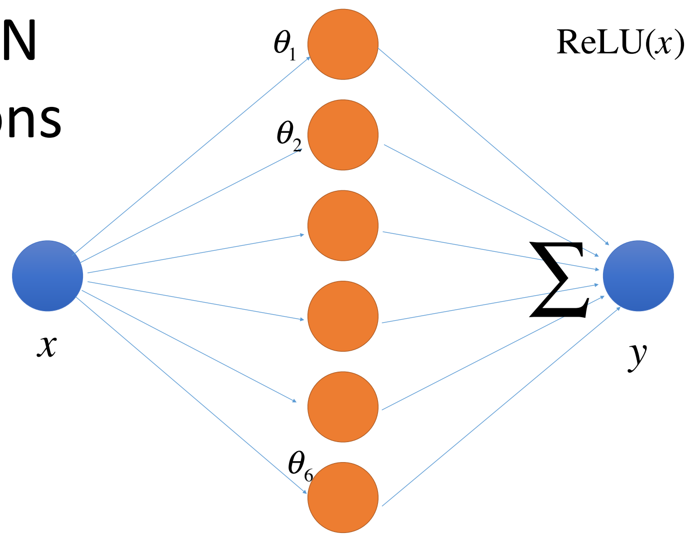
# The Intuition behind Deep

- To achieve the same representation power, we can use fewer neurons with a deeper architecture

- Fewer neurons risk less for overfitting (lacking rigor for this argument)

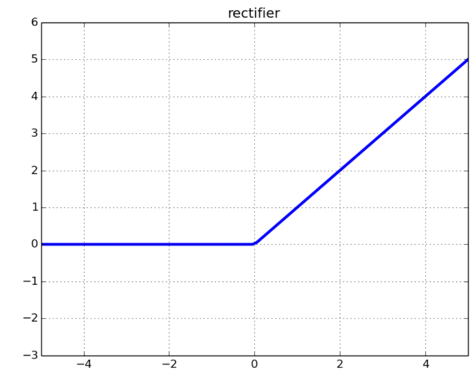
Fat + Short NN

With 6 neurons



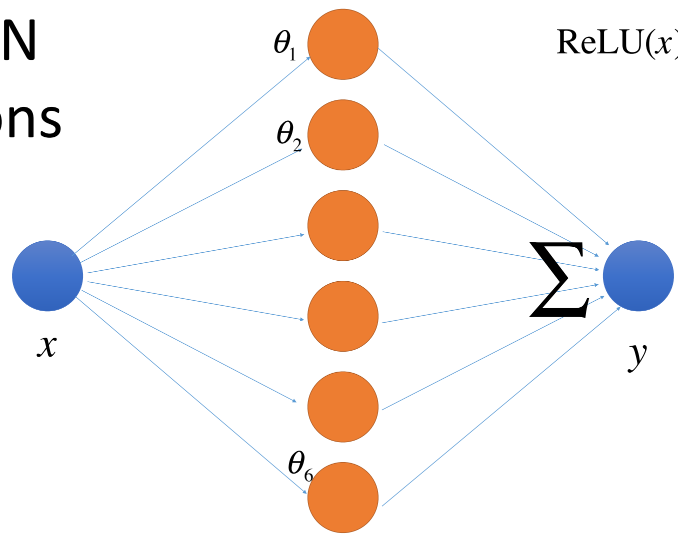

$a_i = \mathrm{ReLU}(x - \theta_i)$

$$\mathrm{ReLU}(x) = \begin{cases} 0 \text{ } if \text{ } x < 0 \\ x \text{ } if \text{ } x >= 0 \end{cases}$$

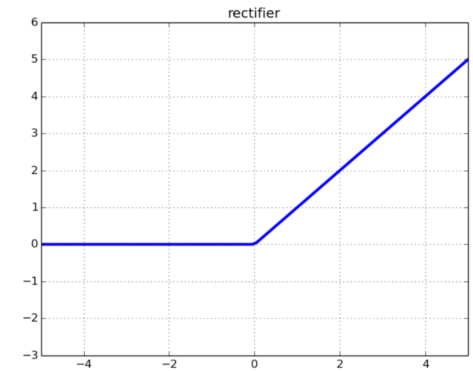



Assume $w_i = 1$ for all neurons, just learn the bias term $b_i$

# Fat + Short NN
# With 6 neurons

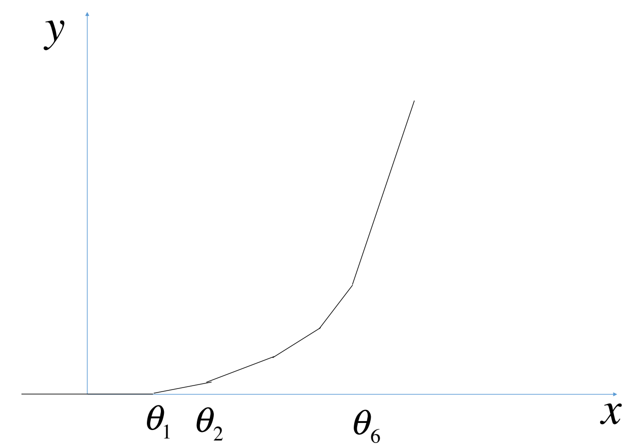



$$\mathrm{ReLU}(x) = \begin{cases} 0 \; if \;\; x < 0 \\ x \; if \;\; x >= 0 \end{cases}$$




$$a_i = \mathrm{ReLU}(x - \theta_i)$$

$$y = \sum_i \mathrm{ReLU}(\mathrm{x}\text{-}\theta_i)$$

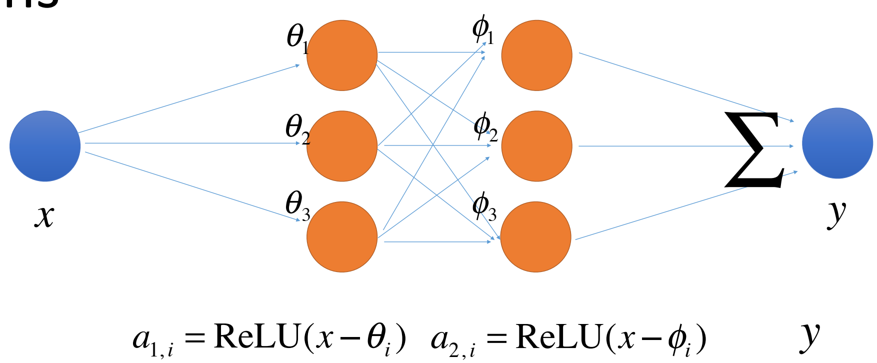



piece-wise linear, 6 knots

Assume $w_i = 1$ for all neurons, just learn the bias term $b_i$

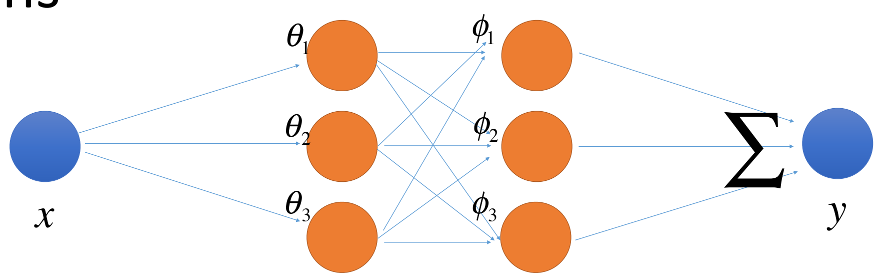
Thin + Tall NN

With 6 neurons

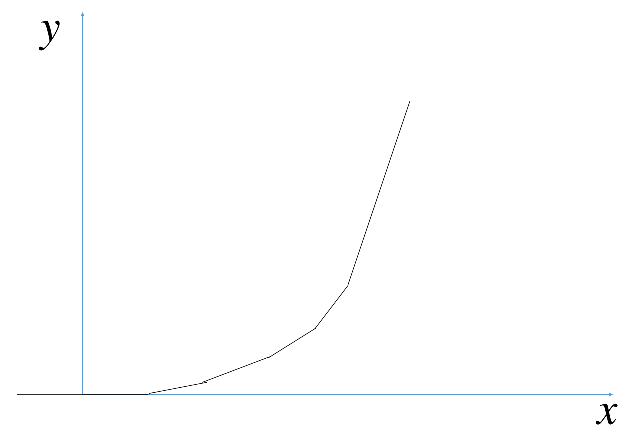



Assume $w_i = 1$ for all neurons, just learn the bias term $b_i$

Thin + Tall NN
With 6 neurons




$a_{1,i} = \mathrm{ReLU}(x - \theta_i)$ $a_{2,i} = \mathrm{ReLU}(x - \phi_i)$

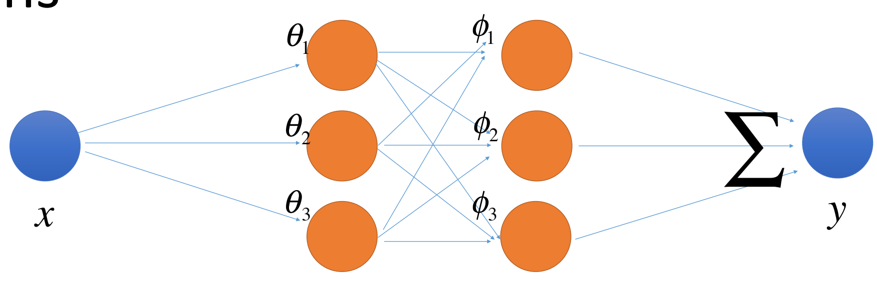



$$y = \sum_{1 \le i \le 3} \mathrm{ReLU}([\sum_{1 \le j \le 3} \mathrm{ReLU}(\mathrm{x}\text{-}\theta_j)]\text{-}\phi_i)$$

piece-wise linear, can have **9 knots**!

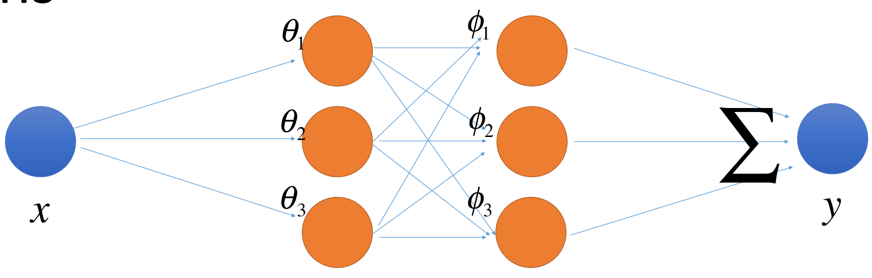
Assume $w_i = 1$ for all neurons, just learn the bias term $b_i$

Thin + Tall NN
With 6 neurons




Interpretation I: With the same number of neurons, create combinatorial data flow

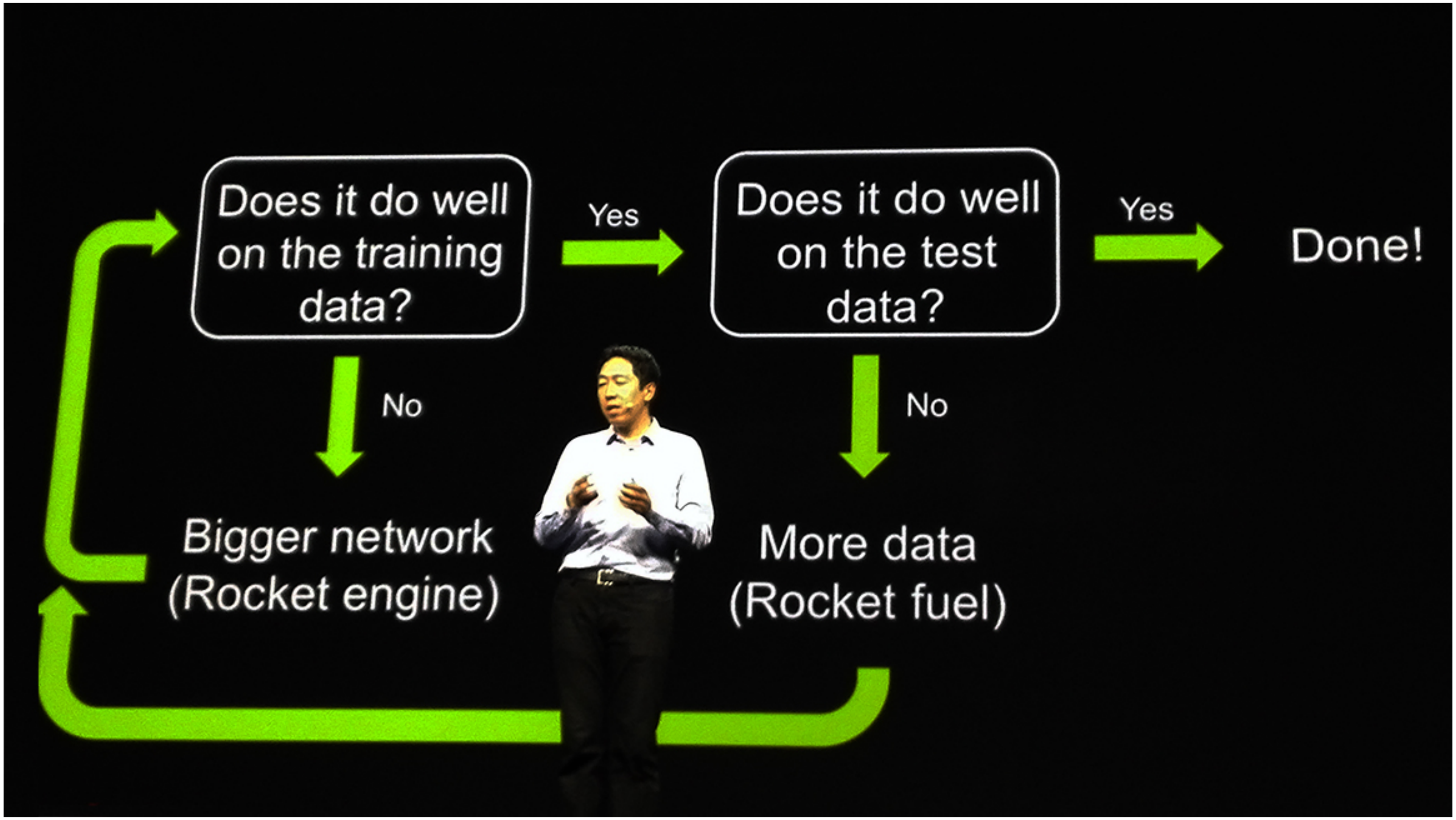
Thin + Tall NN
With 6 neurons




Interpretation I: With the same number of neurons, create combinatorial data flow

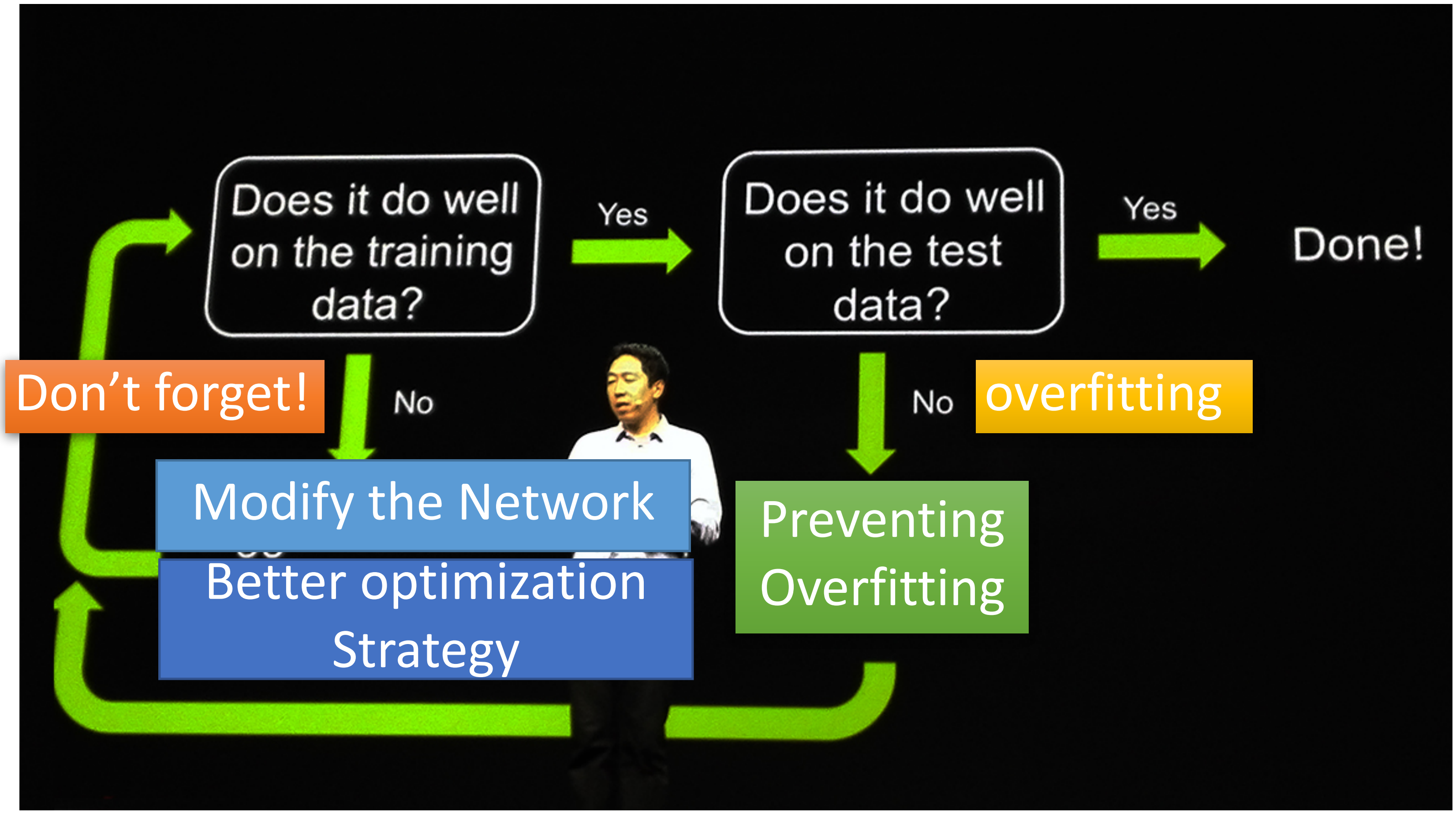
Interpretation II: Abstract data progressively (edge-part-object)

# Recipe for Learning




http://www.gizmodo.com.au/2015/04/the-basic-recipe-for-machine-learning-explained-in-a-single-powerpoint-slide/

# Recipe for Learning




http://www.gizmodo.com.au/2015/04/the-basic-recipe-for-machine-learning-explained-in-a-single-powerpoint-slide/

# Next lecture:

A big step-forward to reduce parameters of networks:

## Convolutional Neural Network