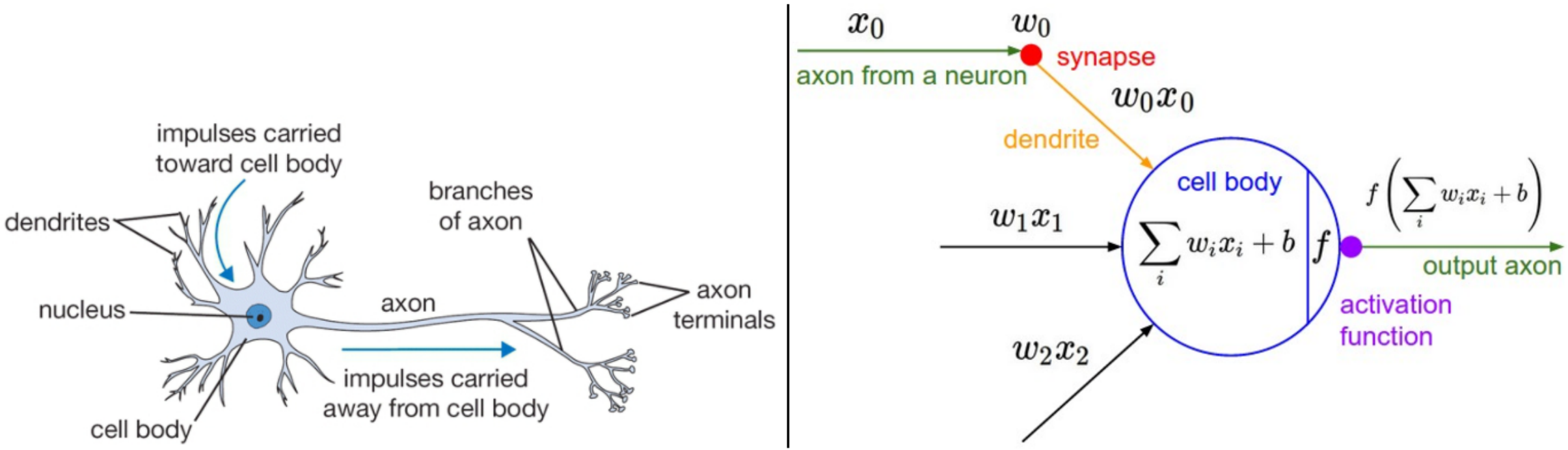
Neural Network

Neuron, Activation Function and more…

Credits: Stanford CS231n Convolutional Neural Networks for Visual Recognition



A cartoon drawing of a biological neuron (left) and its mathematical model (right).

Forward-propagating Single Neuron

```
class Neuron(object):
  # ...
  def forward(self, inputs):
    """ assume inputs and weights are 1-D numpy arrays and bias is a number """
    cell_body_sum = np.sum(inputs * self.weights) + self.bias
    firing_rate = 1.0 / (1.0 + math.exp(-cell_body_sum)) # sigmoid activation function
    return firing_rate
```

- Each neuron performs a dot product with the input and weights
- Adds the bias
- Applies the non-linearity (or activation function)

Sigmod

$\sigma(x)=1/(1+e^{-x})$

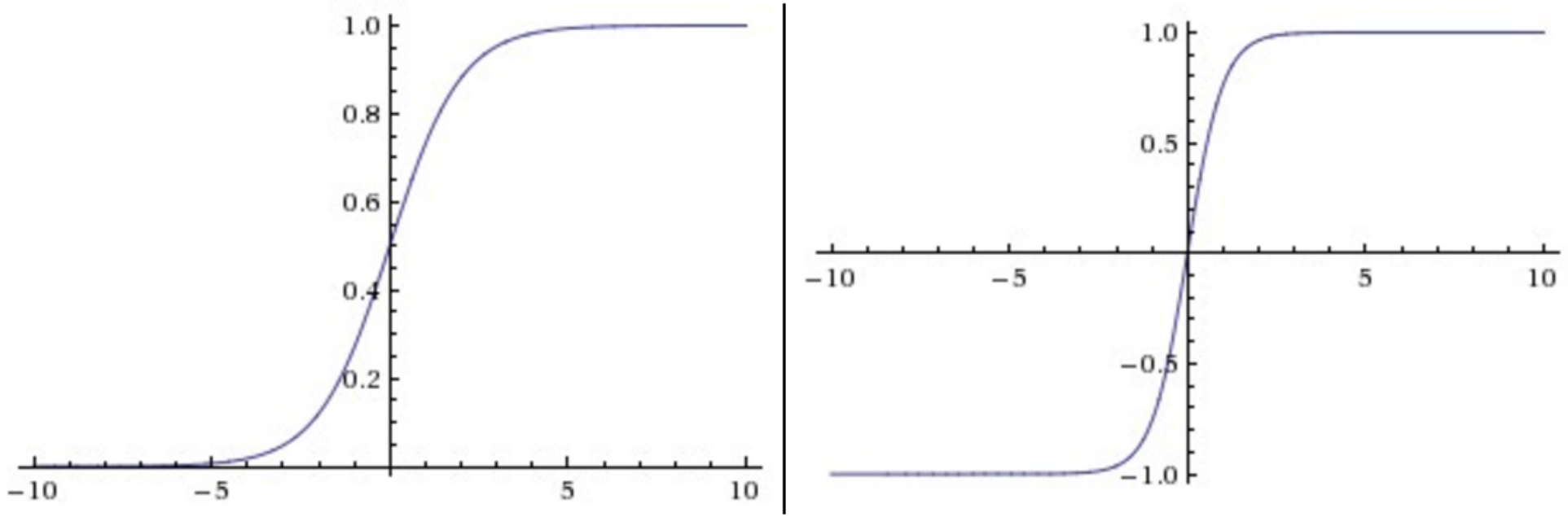
Using a Single Neuron as a Linear Classifier

- A neuron can have an activation of 1 or 0 (depending on which linear regions of the input space)

- With an appropriate loss function on the neuron's output, a single neuron can be a linear classifier.

- Example - **Binary SVM Classifier** – Attach a max-margin hinge loss to the output of the neuron and train it to be a binary SVM

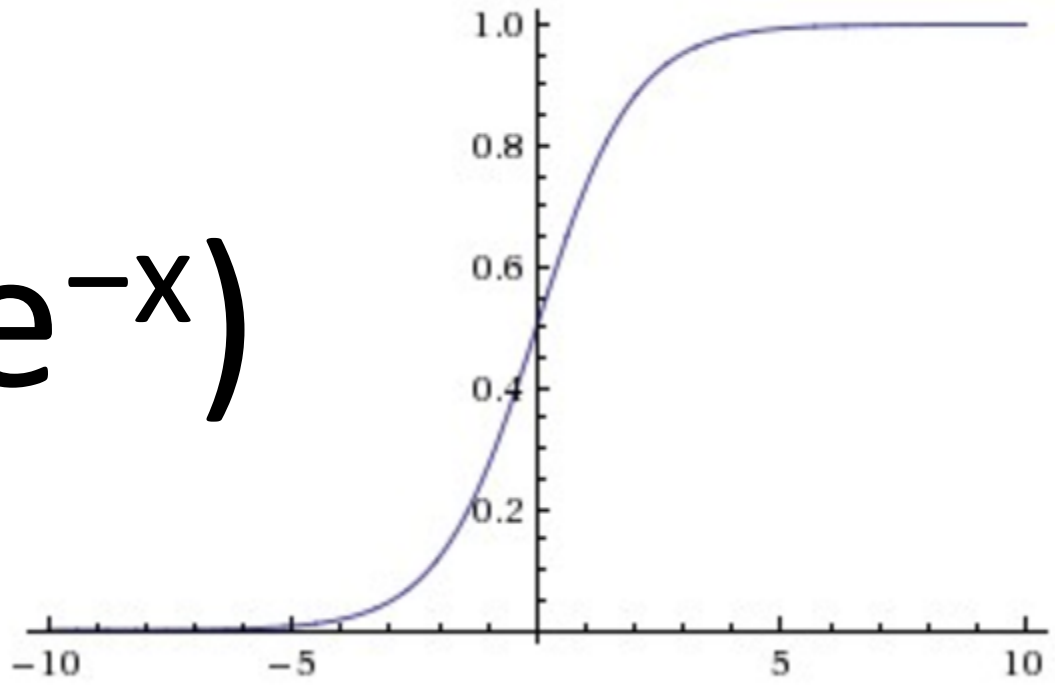
Common activation functions



Left: Sigmoid non-linearity squashes real numbers to range between [0,1] **Right:** The tanh non-linearity squashes real numbers to range between [-1,1].

Activation Function - Sigmoid

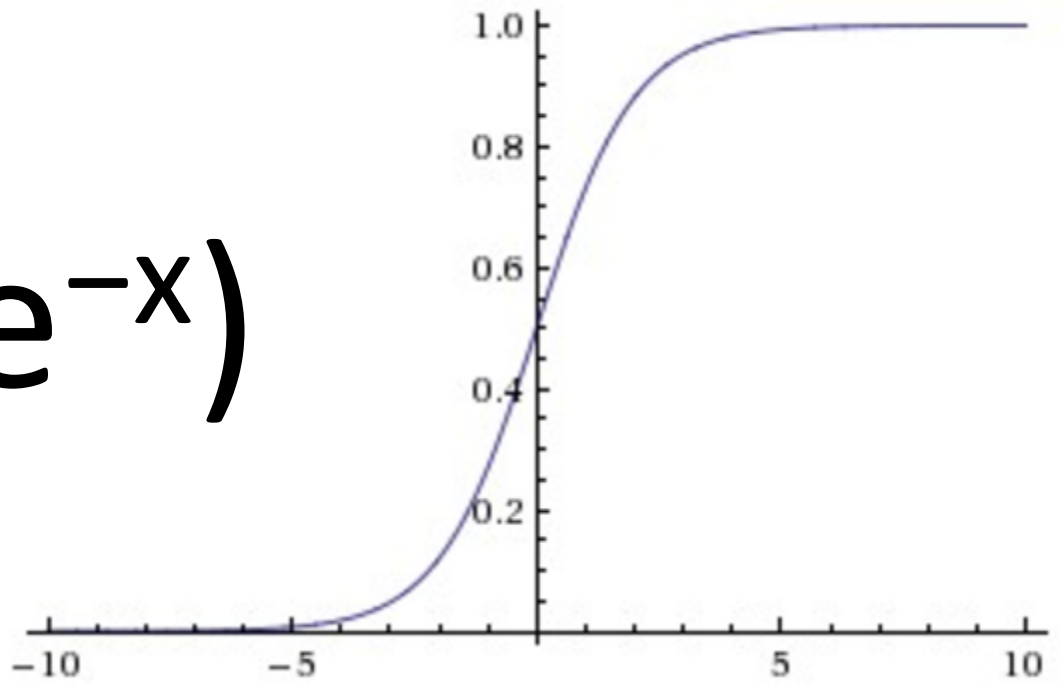
$$\sigma(x)=1/(1+e^{-x})$$



- “Squashes” value into range of 0 and 1
- Large negative numbers become 0
- Large positive Numbers become 1

Activation Function - Sigmoid

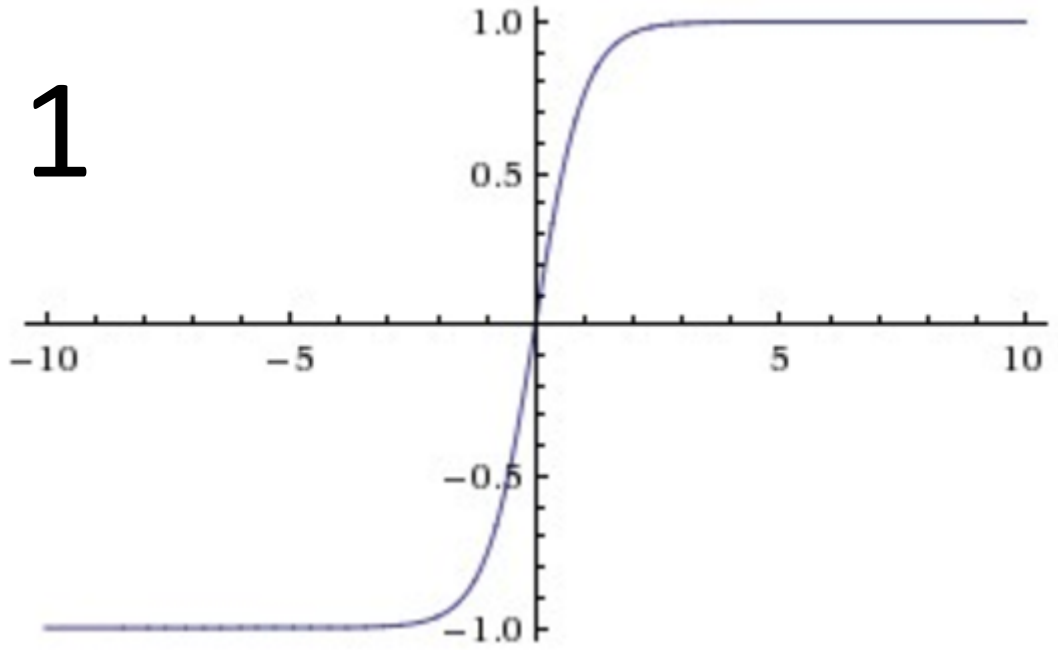
$$\sigma(x)=1/(1+e^{-x})$$



- Commonly used in Neural Networks
- Non-linearity of sigmod has caused issues:
  - Saturate and kill gradient
  - Sigmod outputs are not zero centered

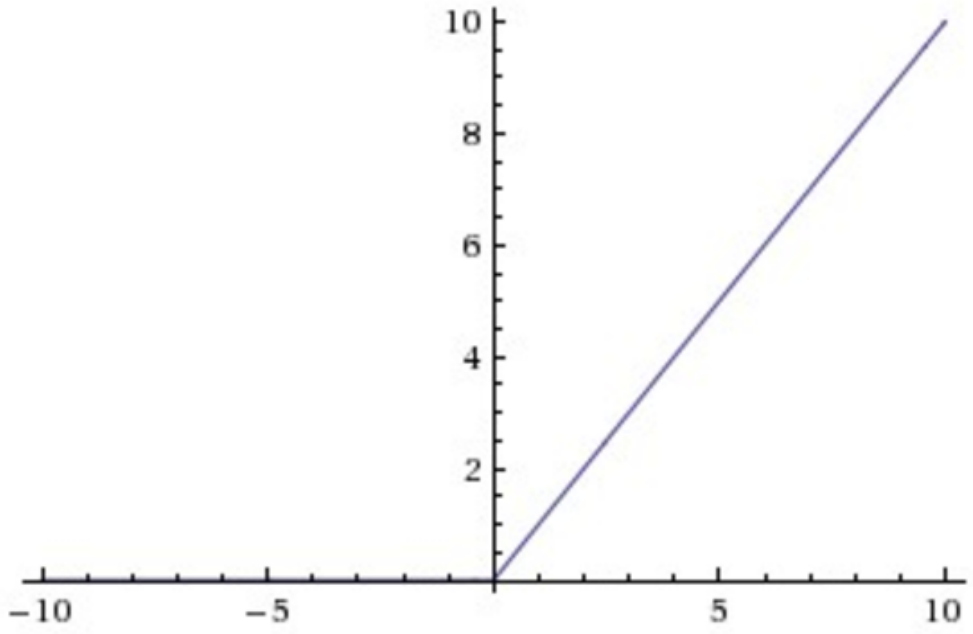
Activation Function - tanh

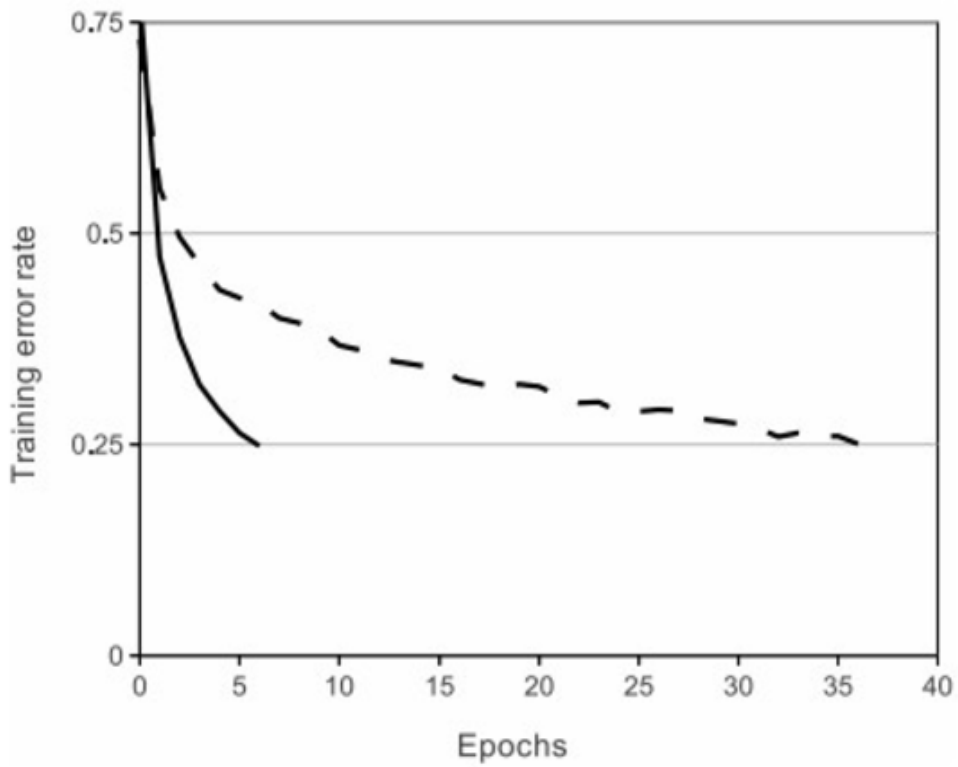
$$\tanh(x) = 2\sigma(2x) - 1$$



- “Squashes” value to [-1, 1]
- Activations saturate
- Output is zero-centered
- Usually preferred to sigmod

Activation Function - ReLU $f(x)=\max(0,x)$





Left: Rectified Linear Unit (ReLU) activation function, which is zero when x < 0 and then linear with slope 1 when x > 0. **Right:** A plot from Krizhevsky et al. (pdf) paper indicating the 6x improvement in convergence with the ReLU unit compared to the tanh unit.

Activation Function – Leaky ReLU

- Solves the “dying ReLU” problem
- Instead of the function being zero when x < 0, a leaky ReLU will have a small negative slope (e.g. 0.01).

$$f(x)=1(x<0)(\alpha x) + 1(x>=0)(x)$$

where α is a small constant.

Activation Function - ReLU

- **Pros**
  - Accelerate the convergence of stochastic gradient descent (Compared with sigmoid/tanh)
  - Simpler to implement

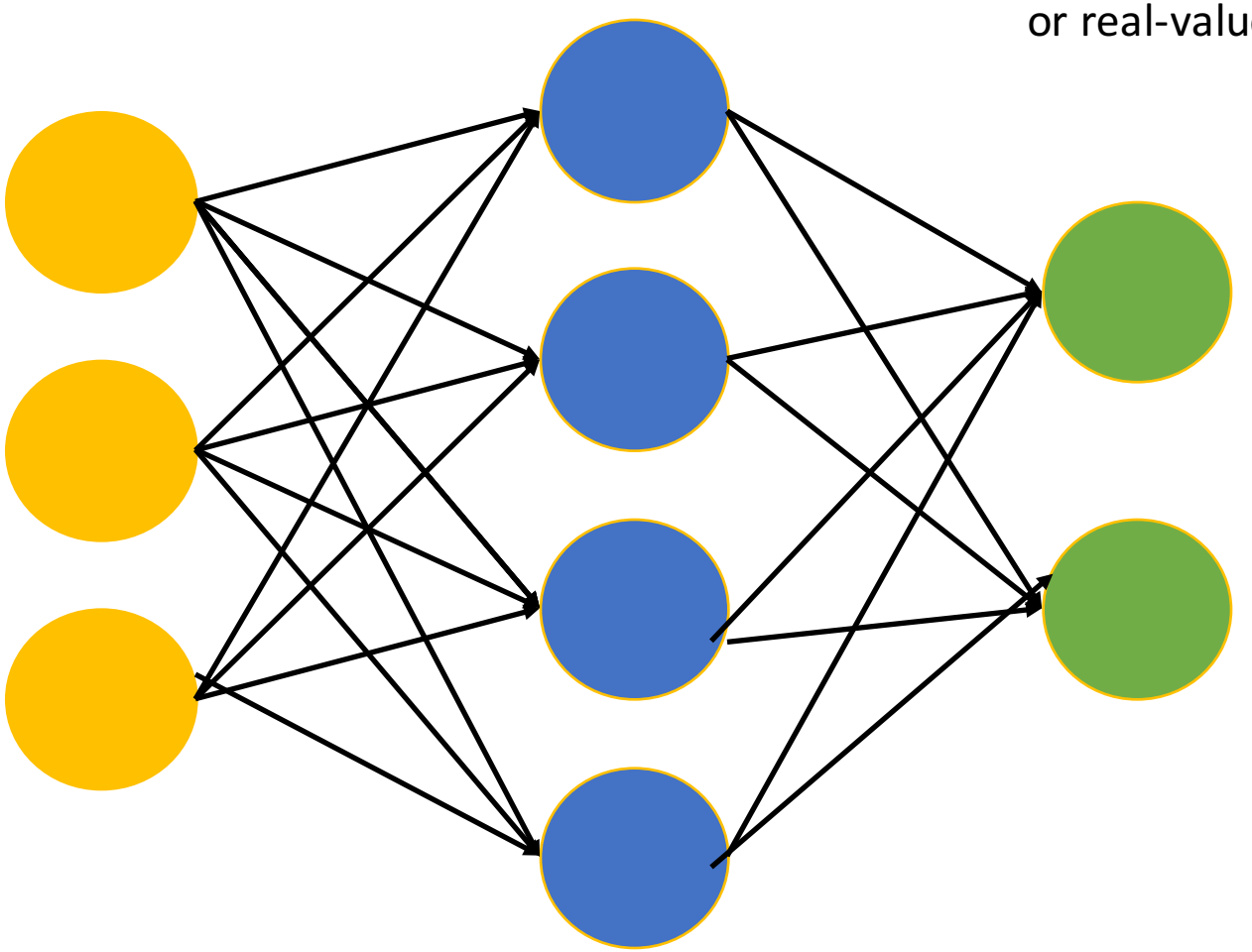
- **Cons**
  - Fragile during training and can “die”.
    Large gradient flowing through a ReLU neuron can cause the weights to update, such that the neuron will never activate on any data point again. (i.e. always zero)

    Example – as much as 40% of the network can be “dead” (i.e. neurons that never activate for the entire training set) if the learning rate is set too high

Neural Networks Architecture

Neural Networks

Fully-connected Networks (FCN)



Input Layer

Hidden Layer

Output layer

Output layer:
Represent class scores (classification) or real-value (regression)

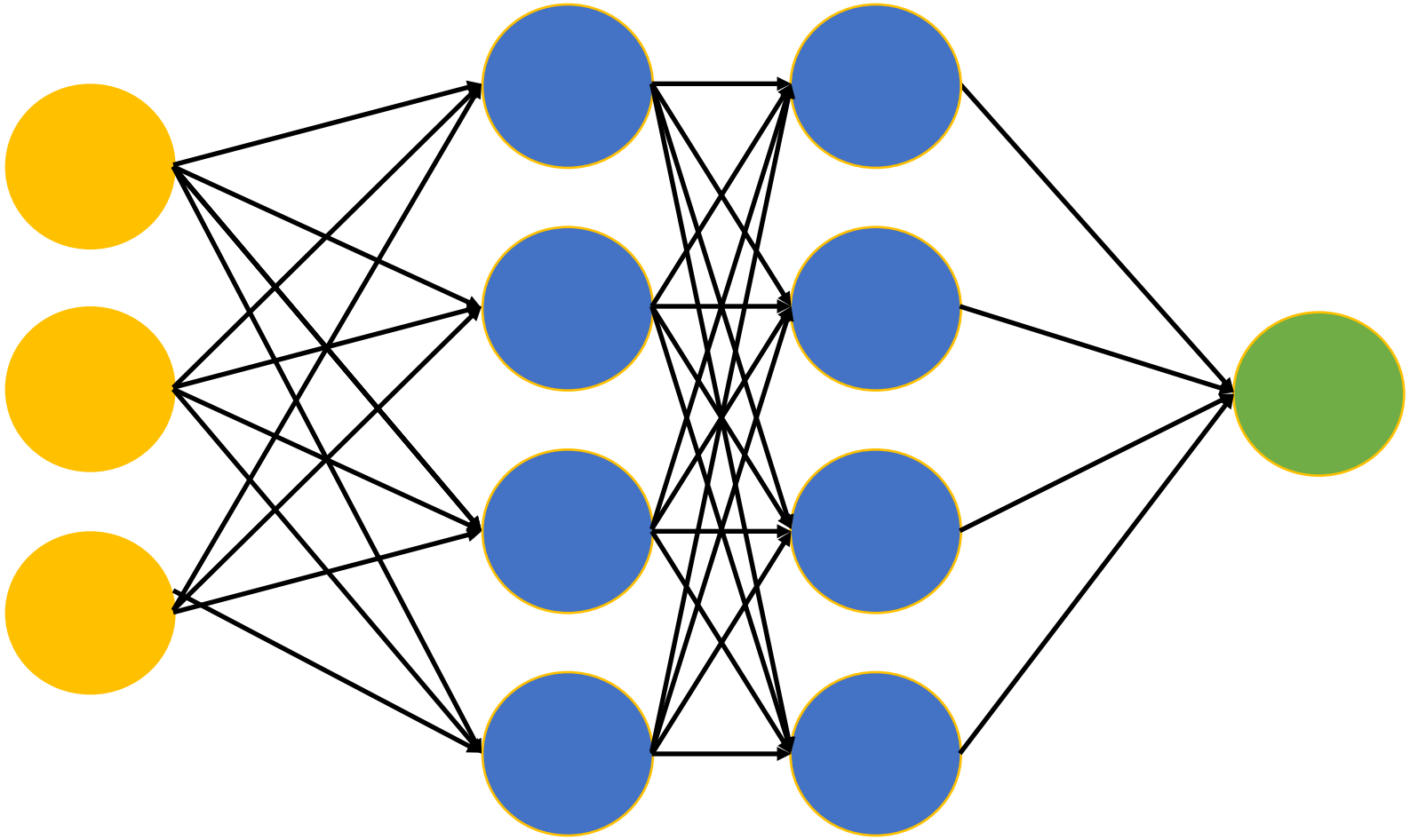
Number of Neurons: 6

Number of weights: 20
(3x4 + 4x2)

Bias: 6 (4+2)

Total :26 learnable parameters

Exercise - Neural Networks



Input Layer

Hidden Layers

Output layer

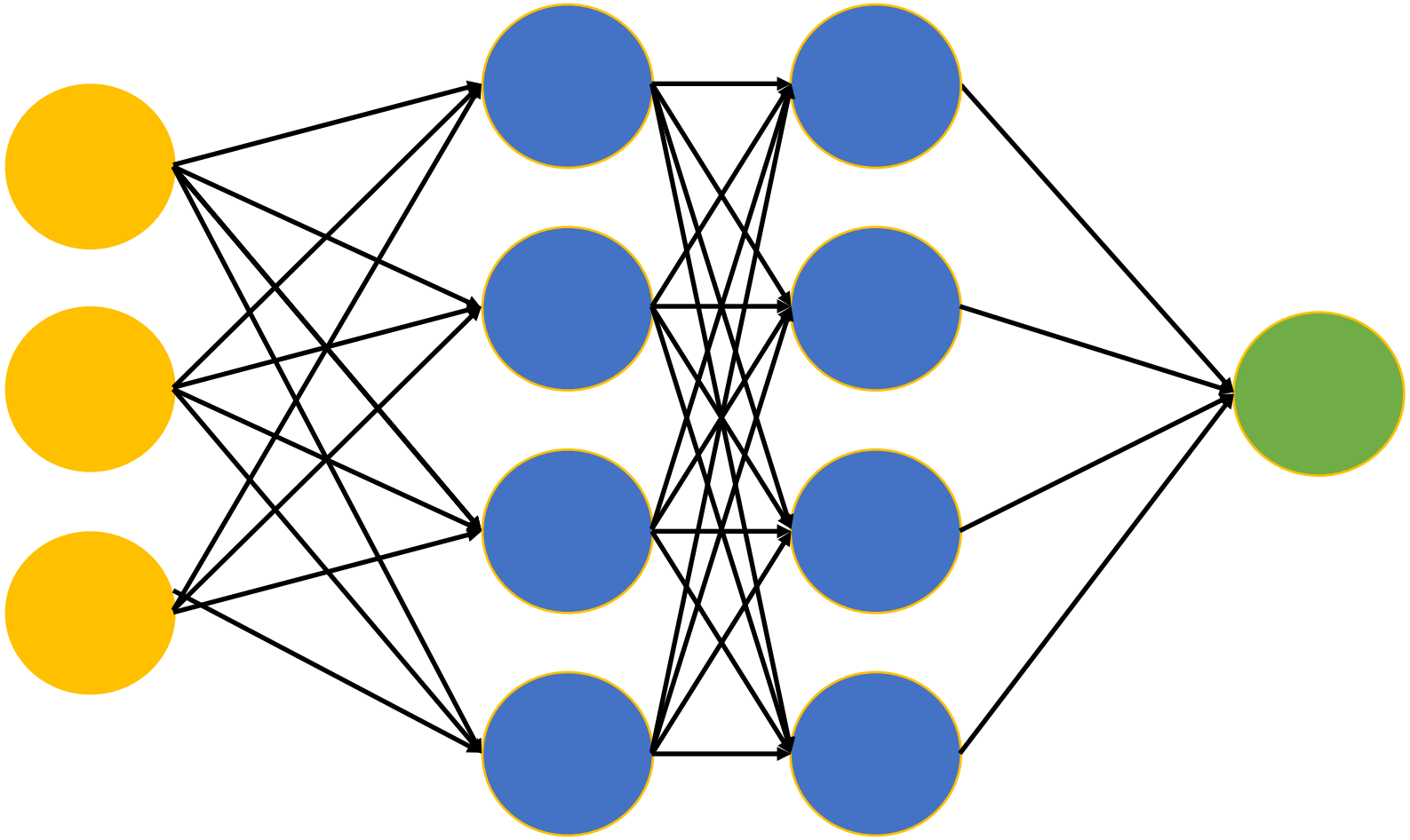
Number of Neurons:

Number of weights:

Bias:

Learnable parameters:

Exercise - Neural Networks



Input Layer

Hidden Layers

Output layer

Number of Neurons: 9

Number of weights: 32
(3x4 + 4x4 + 4x1)

Bias: 9

Learnable parameters: 41

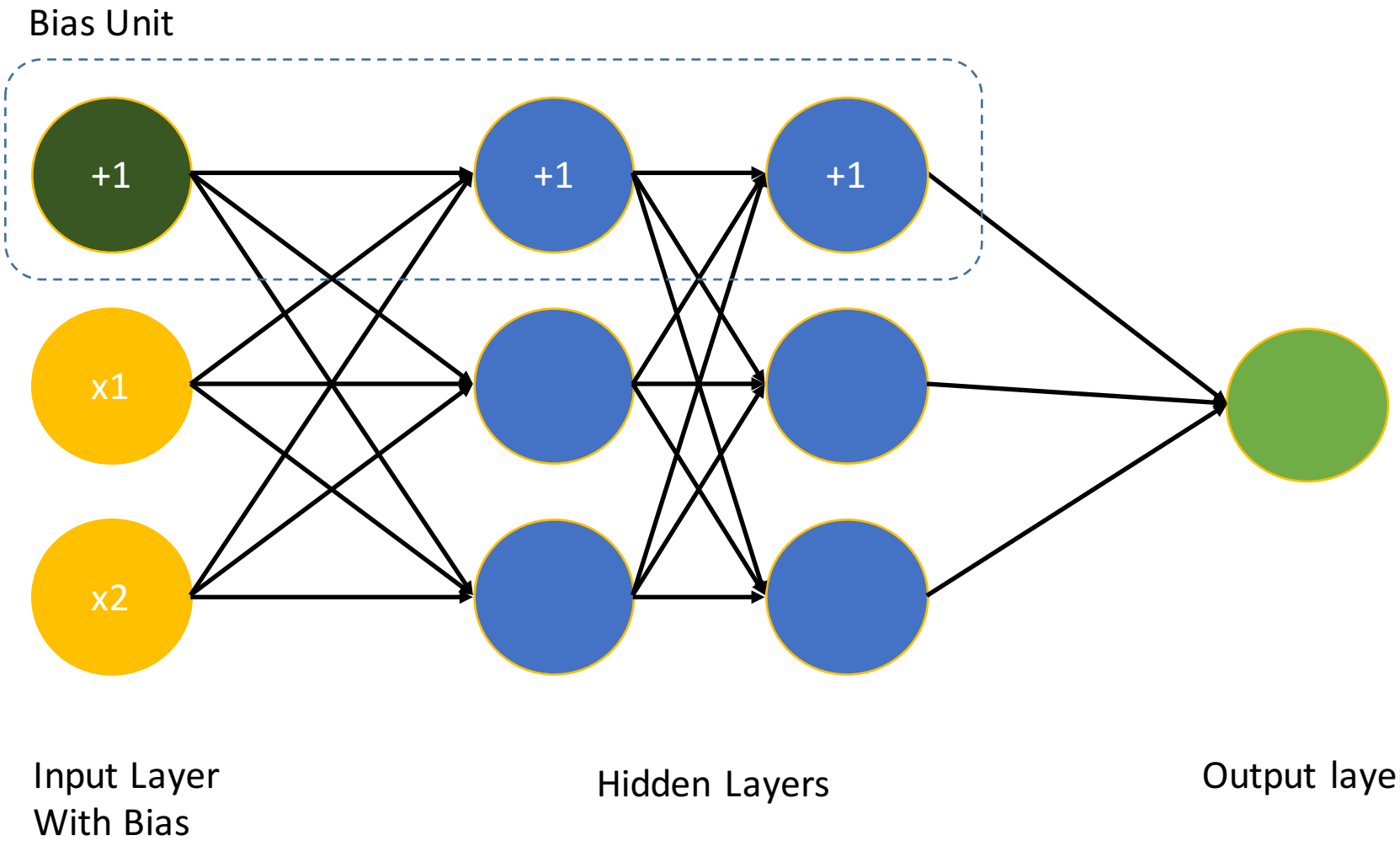
Feed Forward Computation

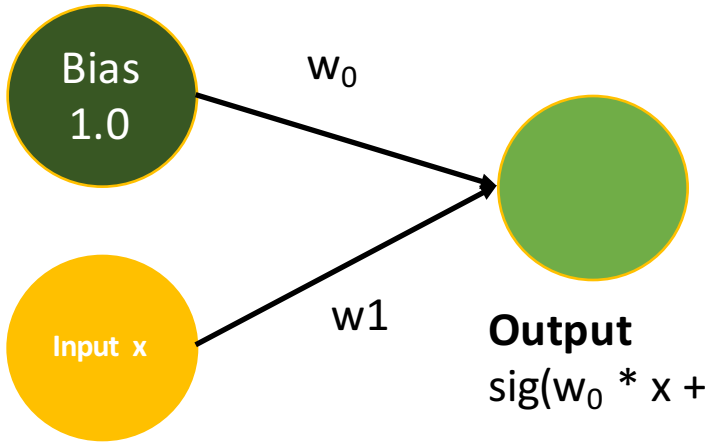
```
# forward-pass of a 3-layer neural network:
f = lambda x: 1.0/(1.0 + np.exp(-x)) # activation function (use sigmoid)
x = np.random.randn(3, 1) # random input vector of three numbers (3x1)
h1 = f(np.dot(W1, x) + b1) # calculate first hidden layer activations (4x1)
h2 = f(np.dot(W2, h1) + b2) # calculate second hidden layer activations (4x1)
out = np.dot(W3, h2) + b3 # output neuron (1x1)
```

- *Repeated matrix multiplications interwoven with activation function*
- Neural Networks are organized into layers
  This enables simple and efficient to evaluate Neural Networks using matrix vector operations

Forward and Backward Propagation

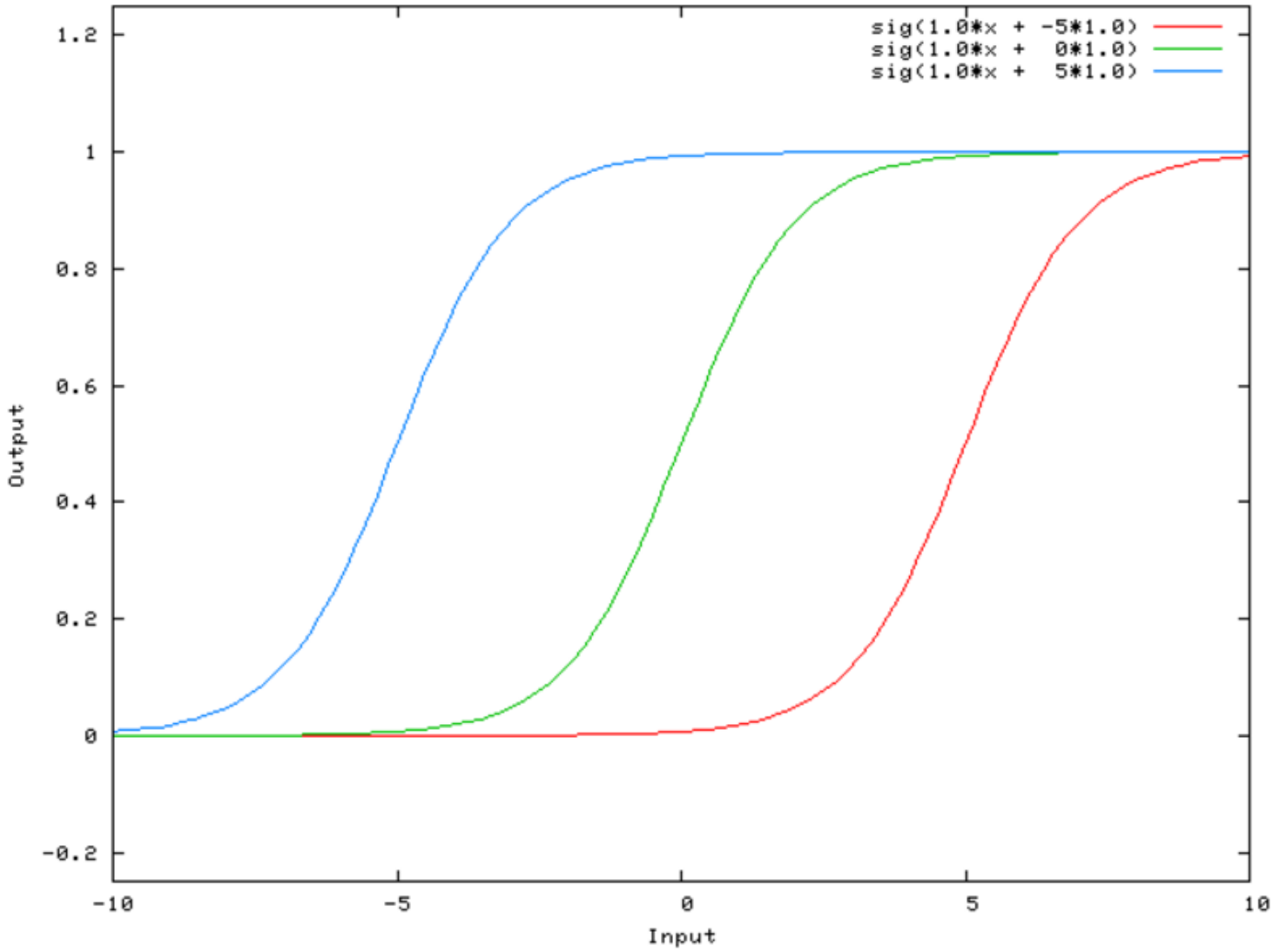
Credits: Andrew Ng – Backpropagation Intuition


Bias Unit
+1
+1
+1
x1
x2
Input Layer
With Bias
Hidden Layers
Output layer



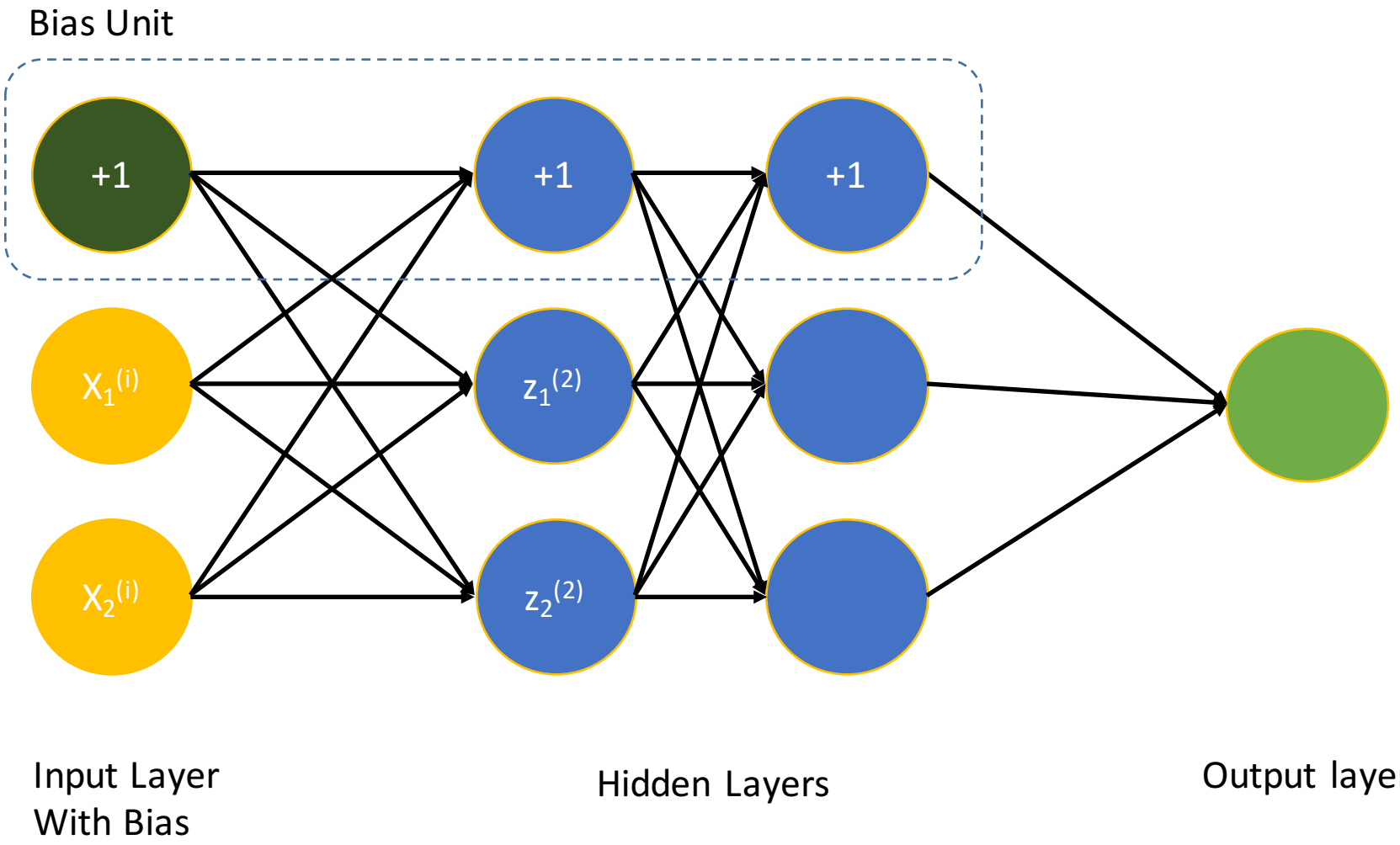
Output

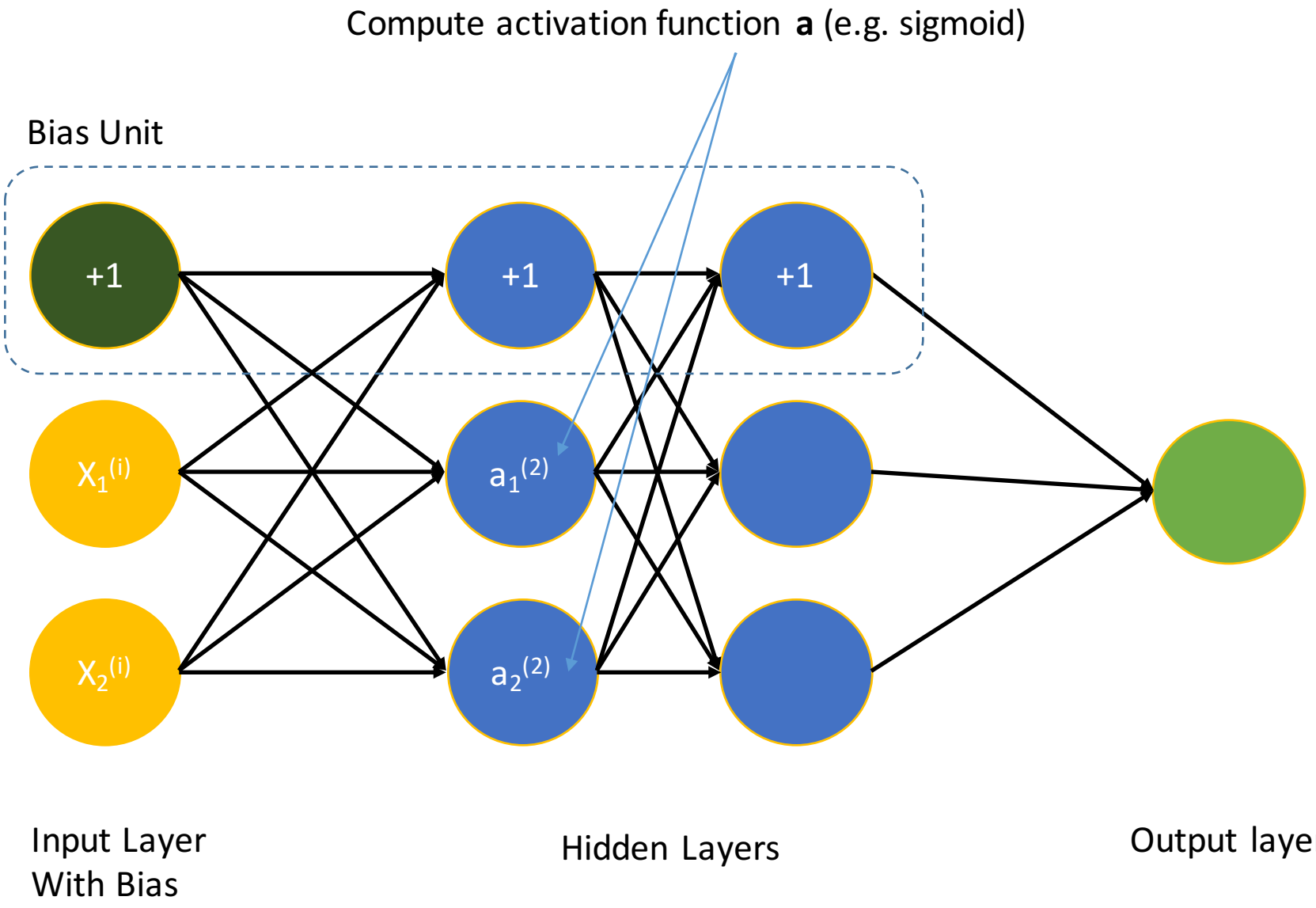
$sig(w_0 * x + w_1 * 1.0)$

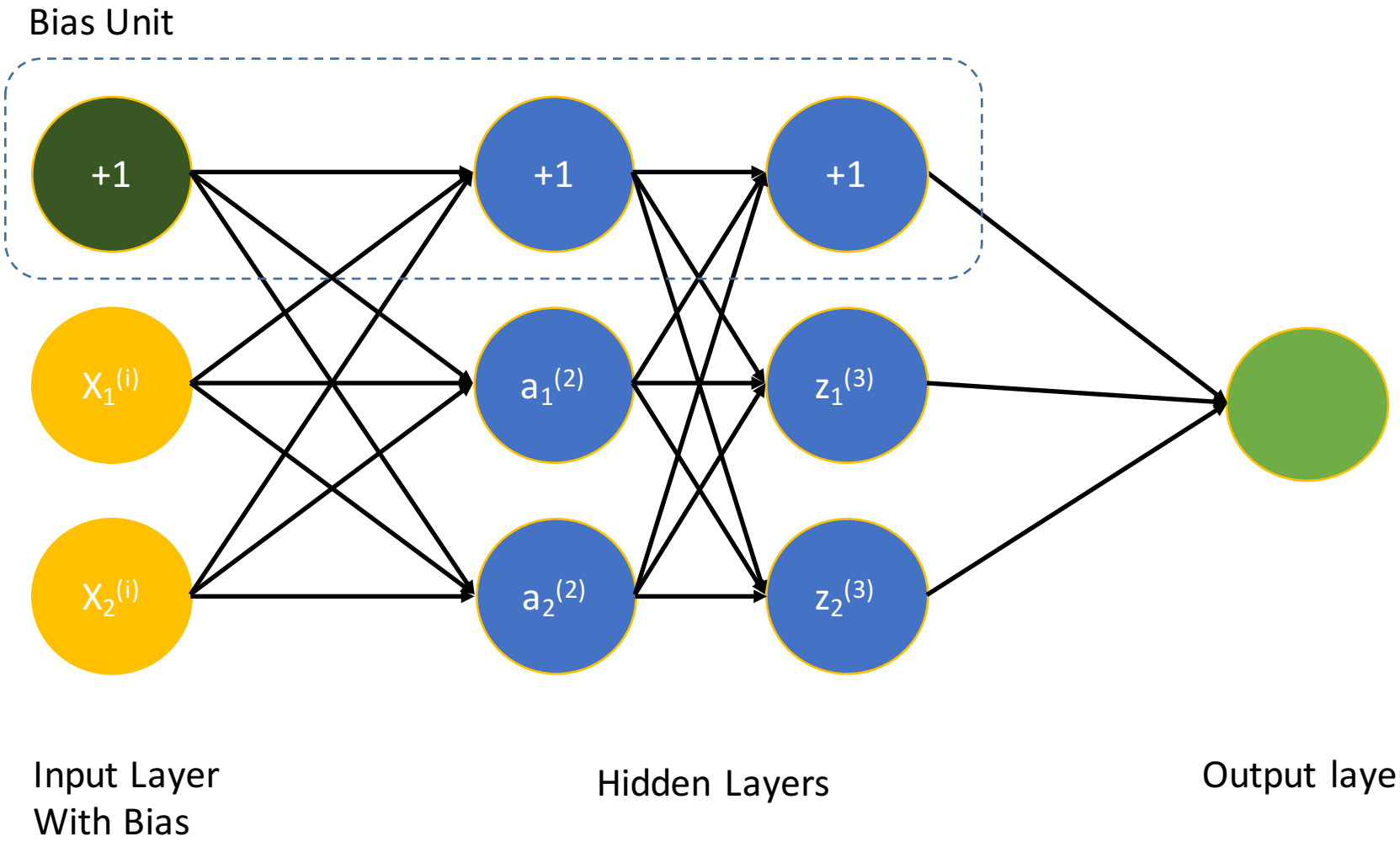


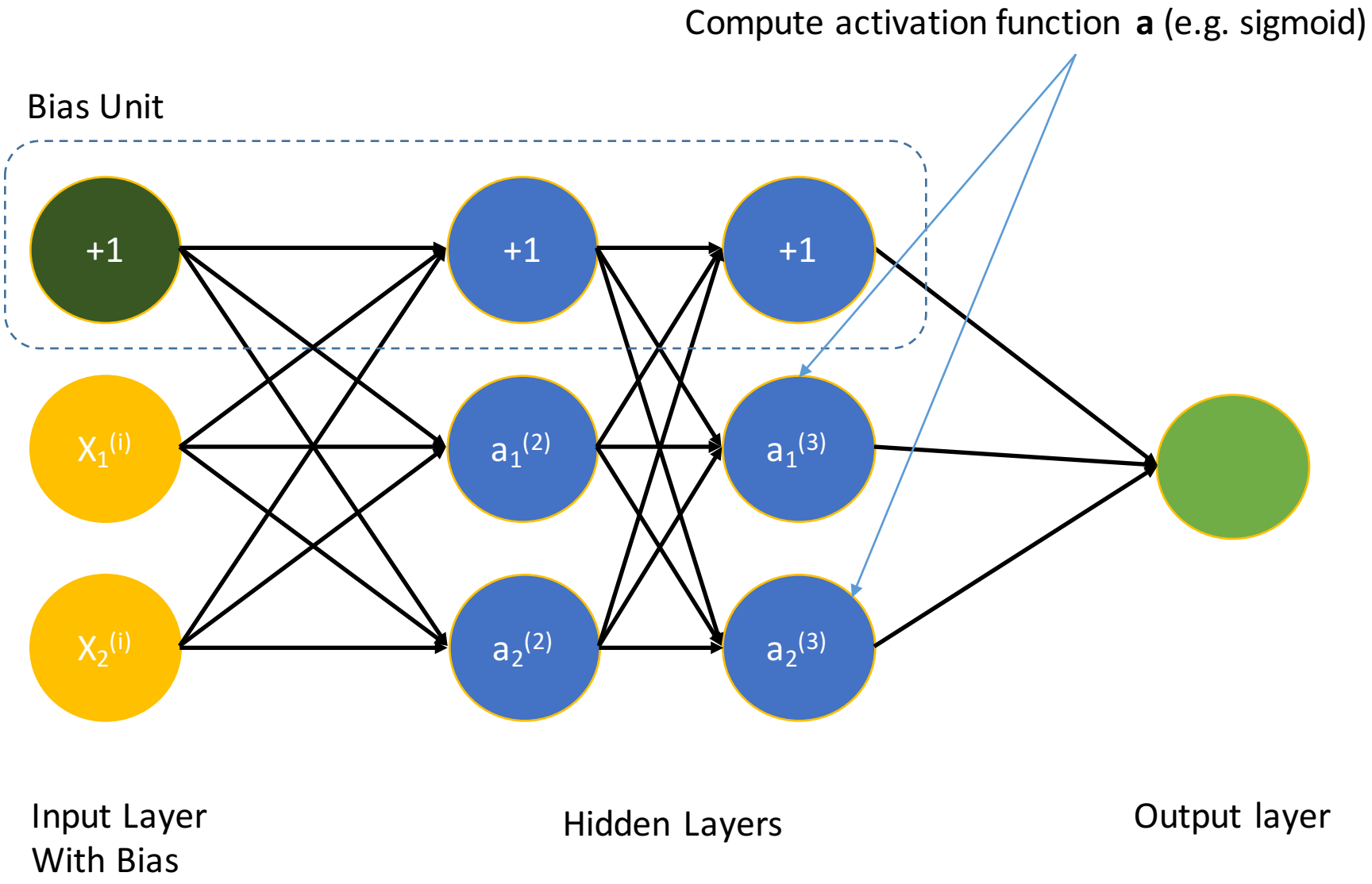
Feed inputs

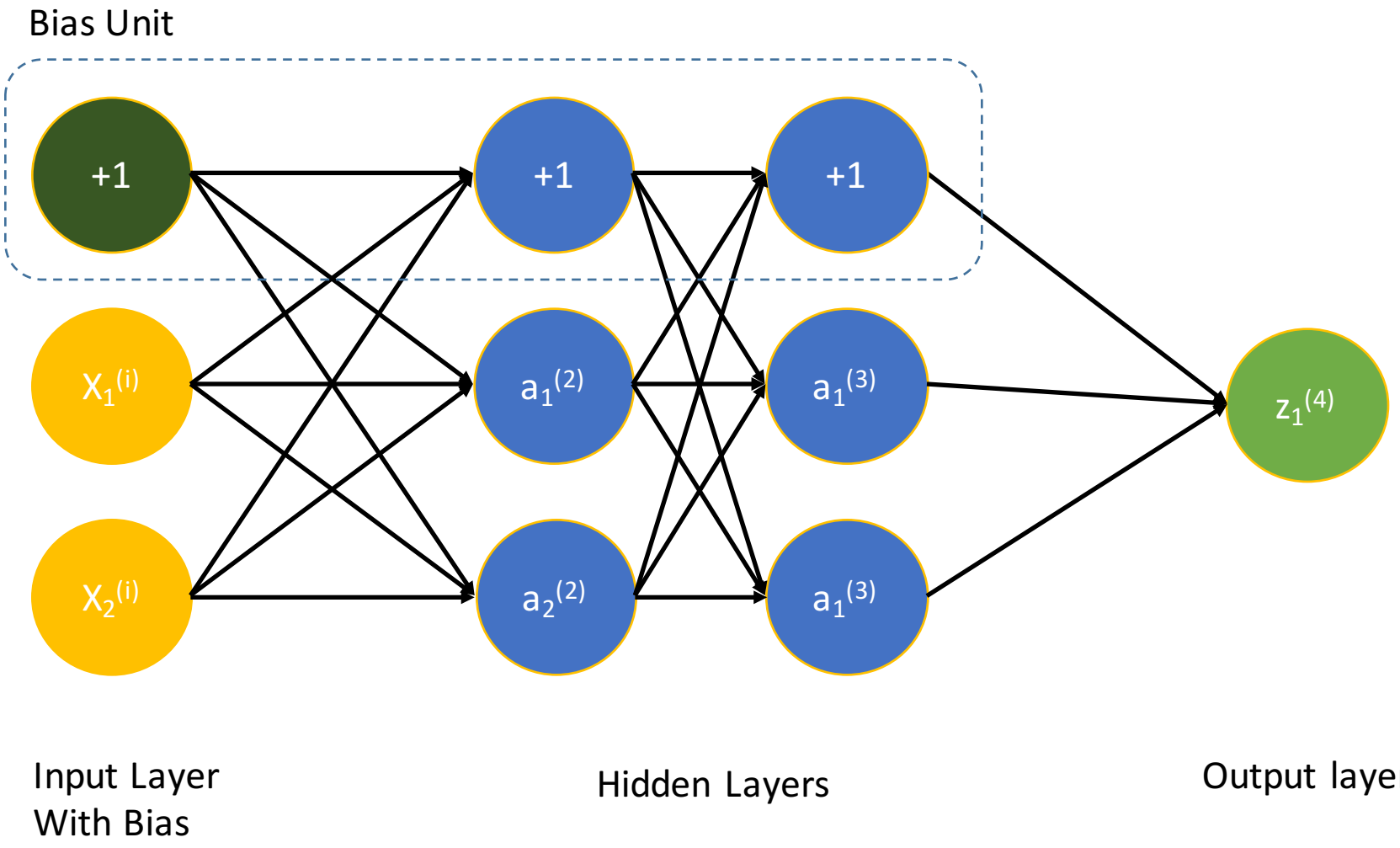
$(x^{(i)}, y^{(i)})$

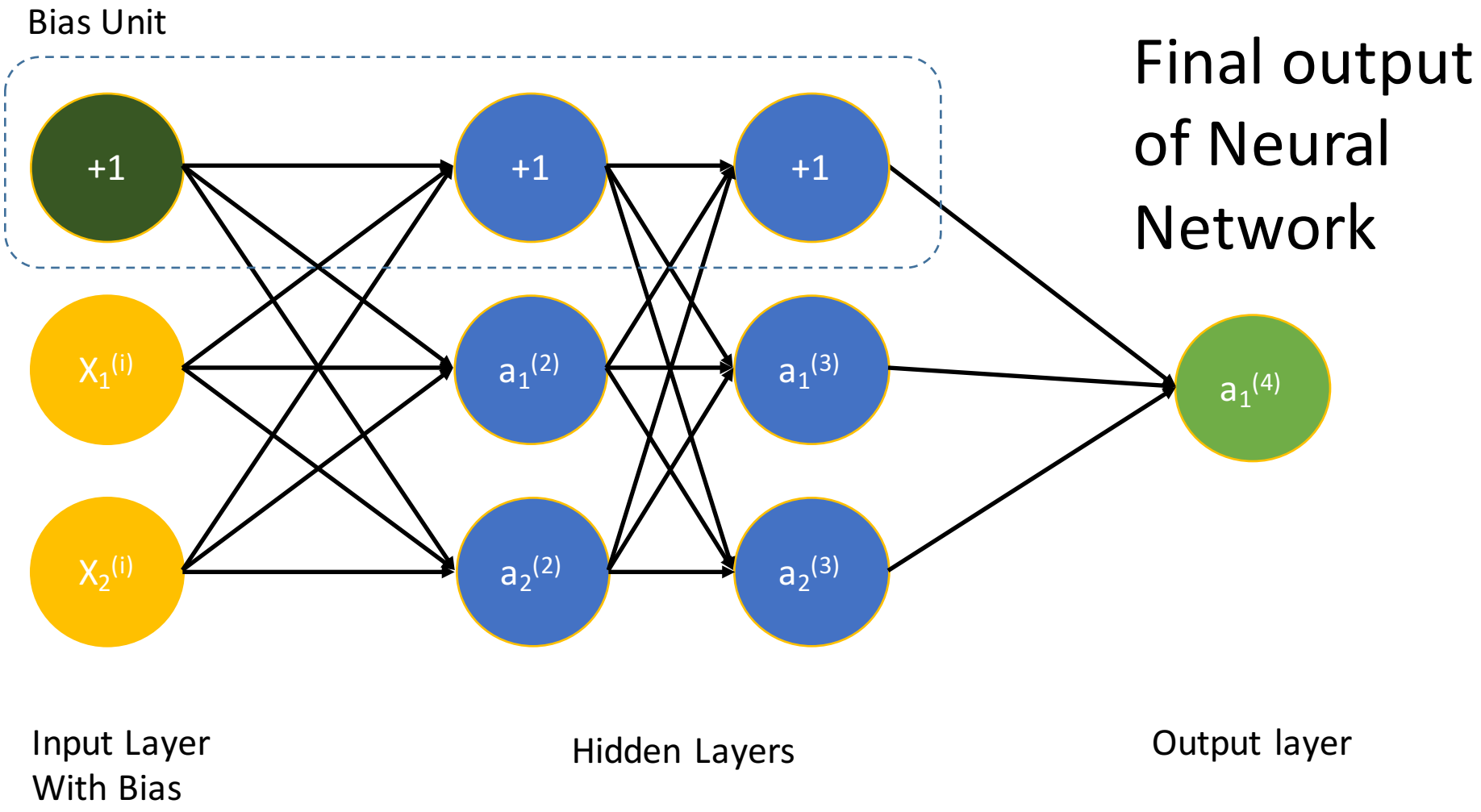


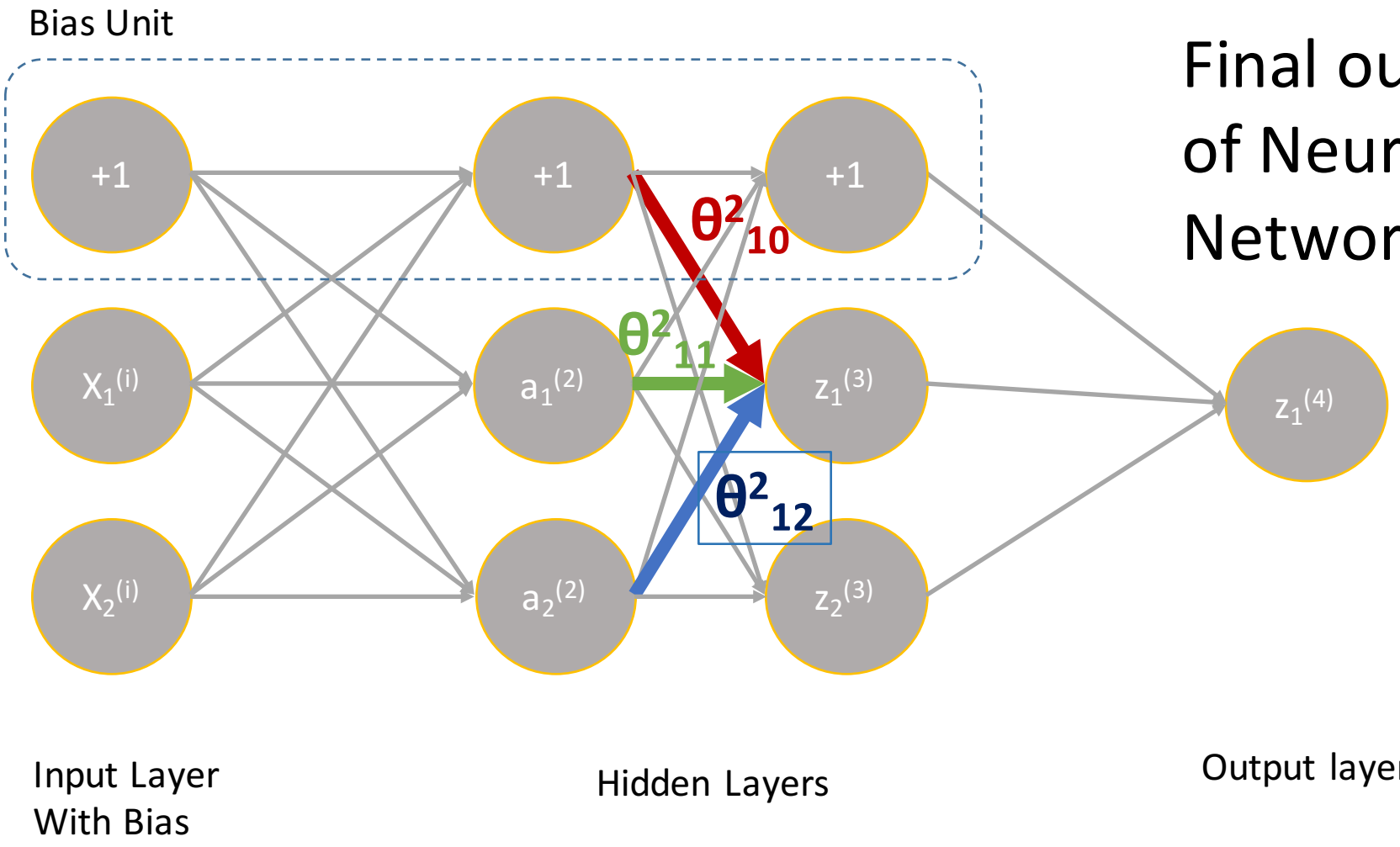

Compute activation function **a** (e.g. sigmoid)
Bias Unit
+1
+1
+1
$X_1^{(i)}$
$X_2^{(i)}$
$a_1^{(2)}$
$a_2^{(2)}$
Input Layer
With Bias
Hidden Layers
Output layer


Bias Unit
+1
+1
+1
$X_1^{(i)}$
$a_1^{(2)}$
$z_1^{(3)}$
$X_2^{(i)}$
$a_2^{(2)}$
$z_2^{(3)}$
Input Layer
With Bias
Hidden Layers
Output layer


Compute activation function **a** (e.g. sigmoid)
Bias Unit
+1
+1
+1
$X_1^{(i)}$
$X_2^{(i)}$
$a_1^{(2)}$
$a_2^{(2)}$
$a_1^{(3)}$
$a_2^{(3)}$
Input Layer
With Bias
Hidden Layers
Output layer


Bias Unit
+1
+1
+1
$X_1^{(i)}$
$a_1^{(2)}$
$a_1^{(3)}$
$z_1^{(4)}$
$X_2^{(i)}$
$a_2^{(2)}$
$a_1^{(3)}$
Input Layer
With Bias
Hidden Layers
Output layer


Bias Unit
+1
+1
+1
$X_1^{(i)}$
$X_2^{(i)}$
$a_1^{(2)}$
$a_2^{(2)}$
$a_1^{(3)}$
$a_2^{(3)}$
$a_1^{(4)}$
Final output of Neural Network
Input Layer With Bias
Hidden Layers
Output layer



Final output of Neural Network

Forward Propagation: $Z_1^{(3)} = \theta^2_{10} \times 1 + \theta^2_{11} \times a_1^{(2)} + \theta^2_{12} \times a_2^{(2)}$

Backpropagation Intuition

Cost Function for 1 Output Unit

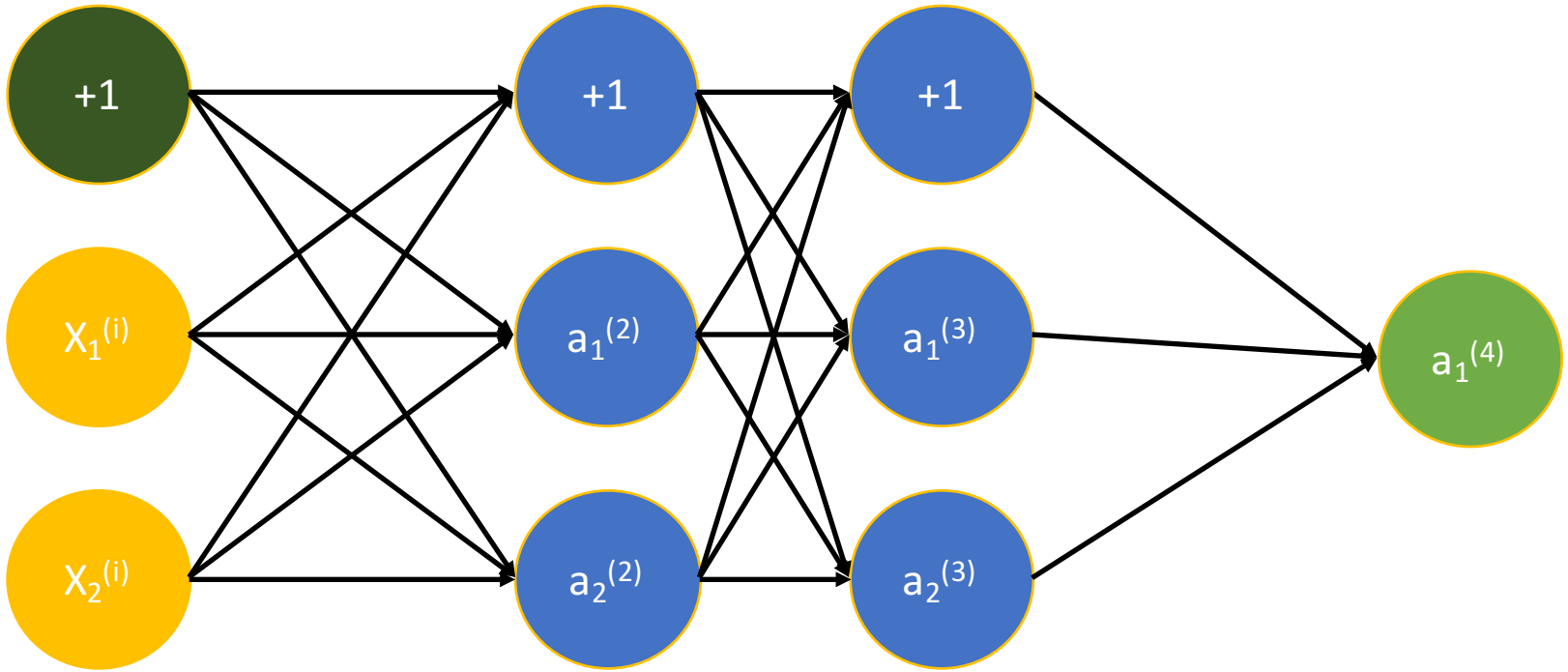
$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log(h_\Theta(x^{(i)})) + (1 - y^{(i)}) \log(1 - (h_\Theta(x^{(i)})))\right]$$
$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{ji}^{(l)})^2$$

cost(i)

- Cost associated with training example ($x^{(i)}$, $y^{(i)}$)
- Similar to squared error (squared difference between what the neural network output vs actual value)

Backpropagation Intuition

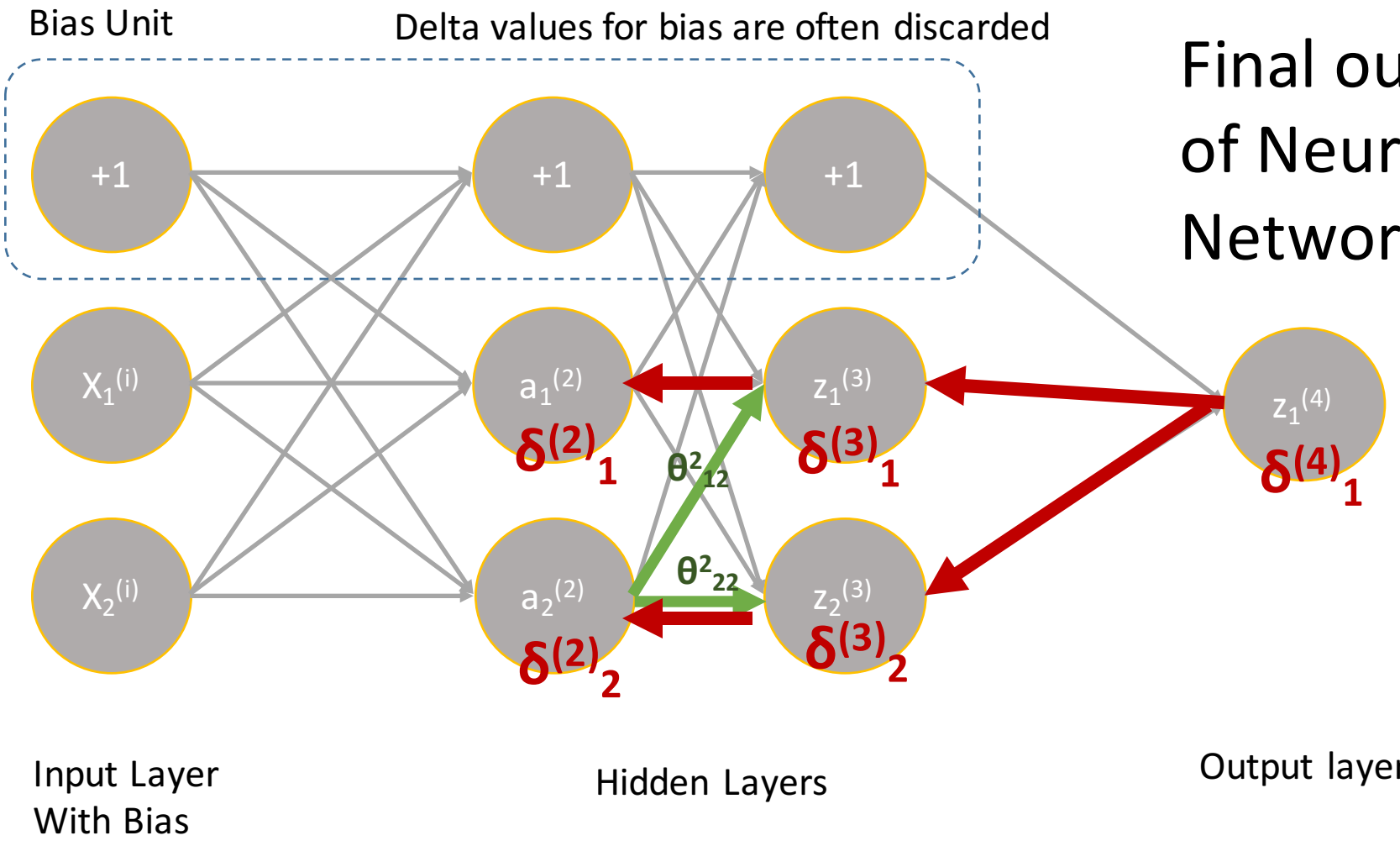
$\delta_j^{(l)}$ = "error" of cost for $a_j^{(l)}$ (unit j in layer l).



Input Layer
With Bias

Hidden Layers

Output layer

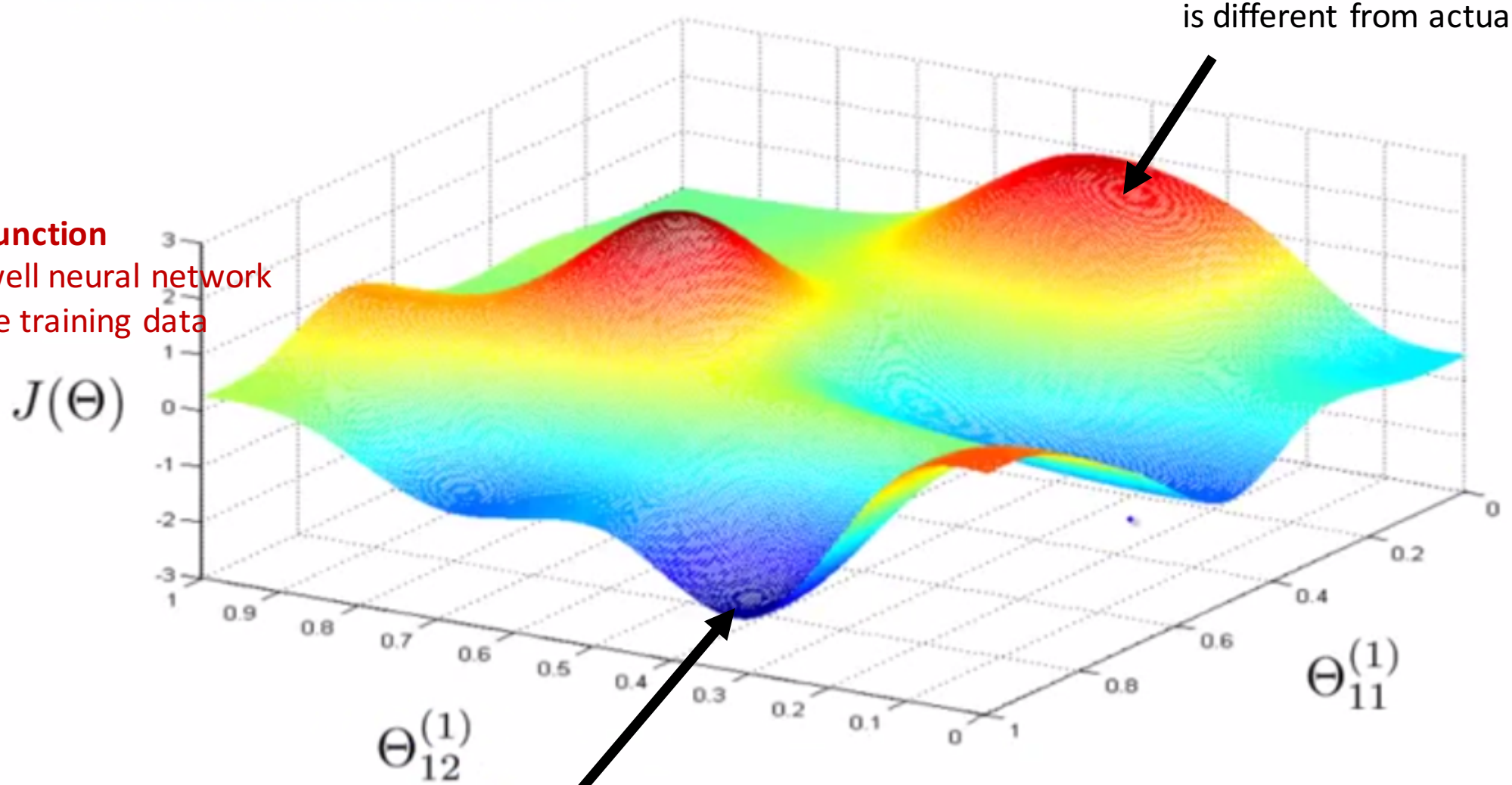


Backward propagation: $\delta_2^{(2)} = \theta^2_{12}\,\delta^{(3)}_1 + \theta^2_{22}\,\delta^{(3)}_2$

Gradient Descent

Not fitting training set well
Neural network output values that
is different from actual value

Cost Function
How well neural network
Fits the training data



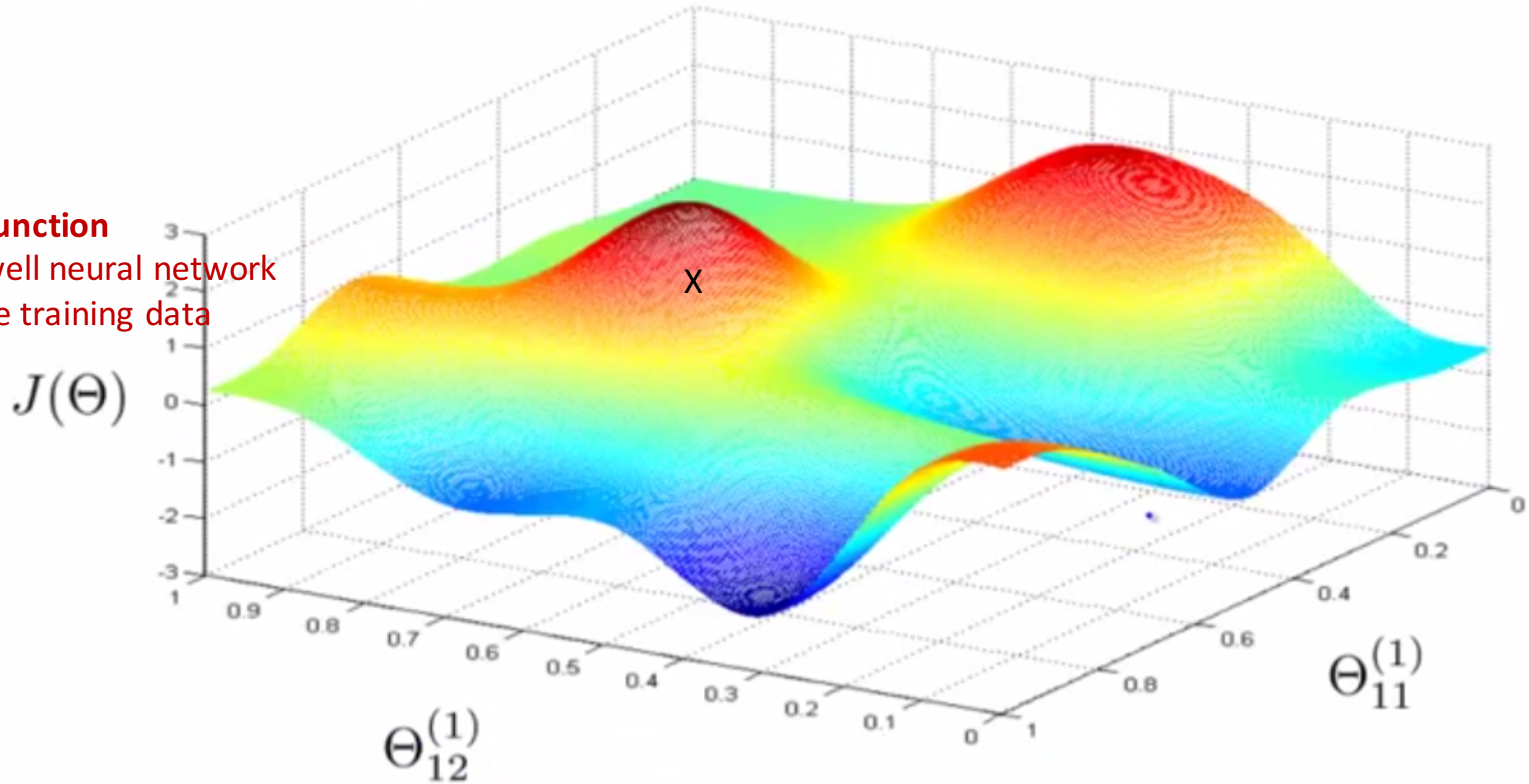
Parameters of neural networks

Low cost function
Neural network output values that
Is similar to actual value

Cost Function

How well neural network

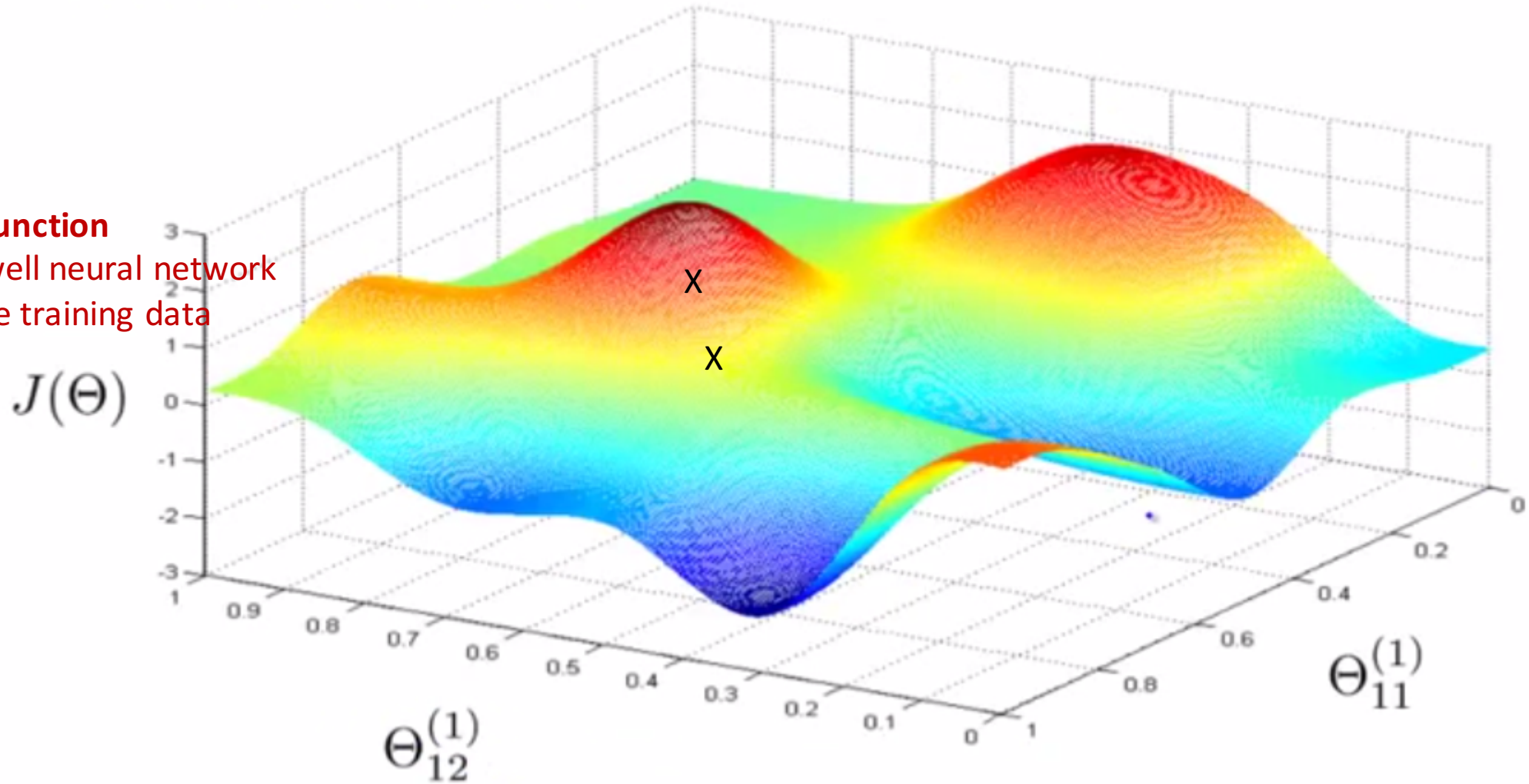
Fits the training data



Parameters of neural networks

Cost Function

How well neural network
Fits the training data

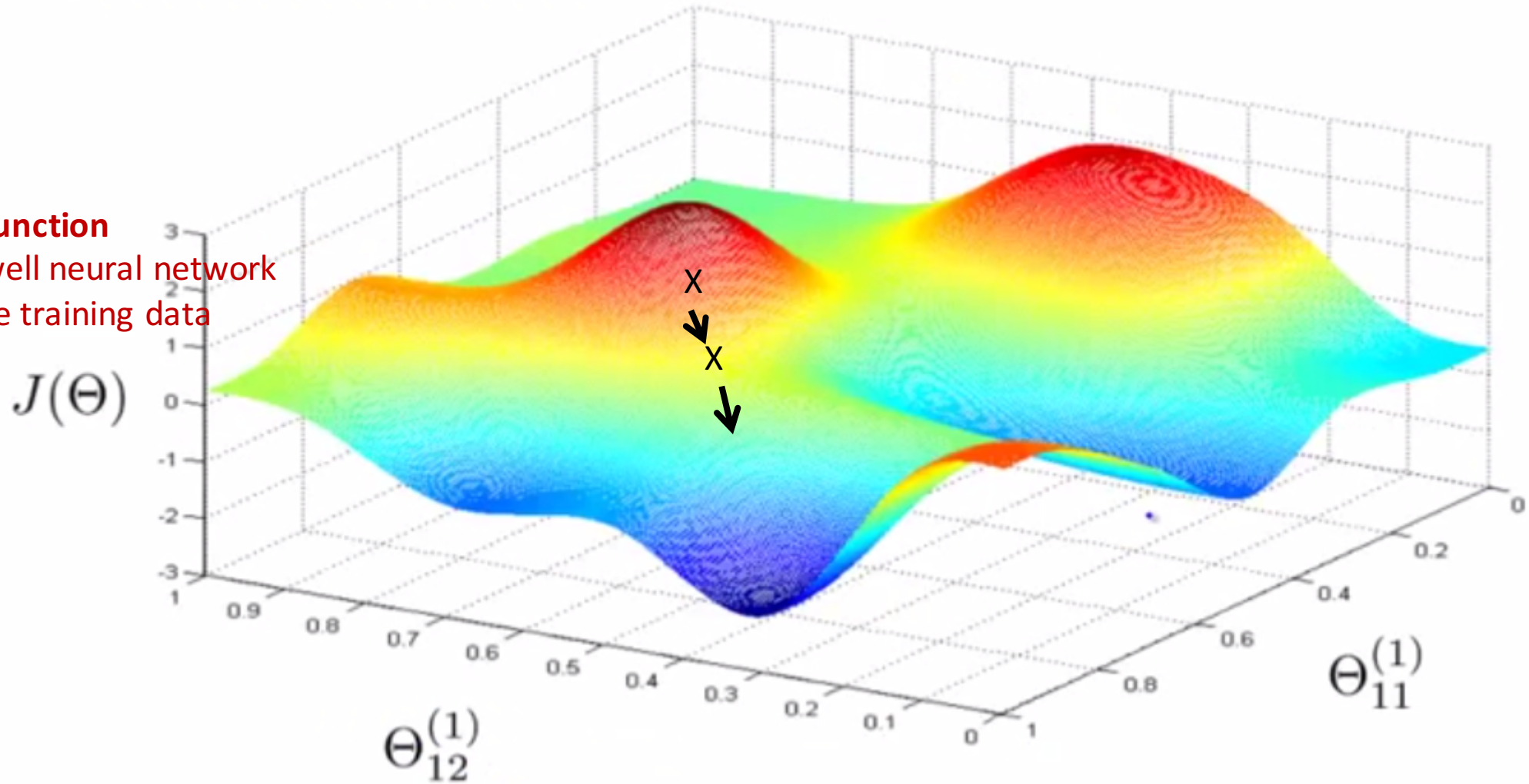


Parameters of neural networks

Cost Function

How well neural network

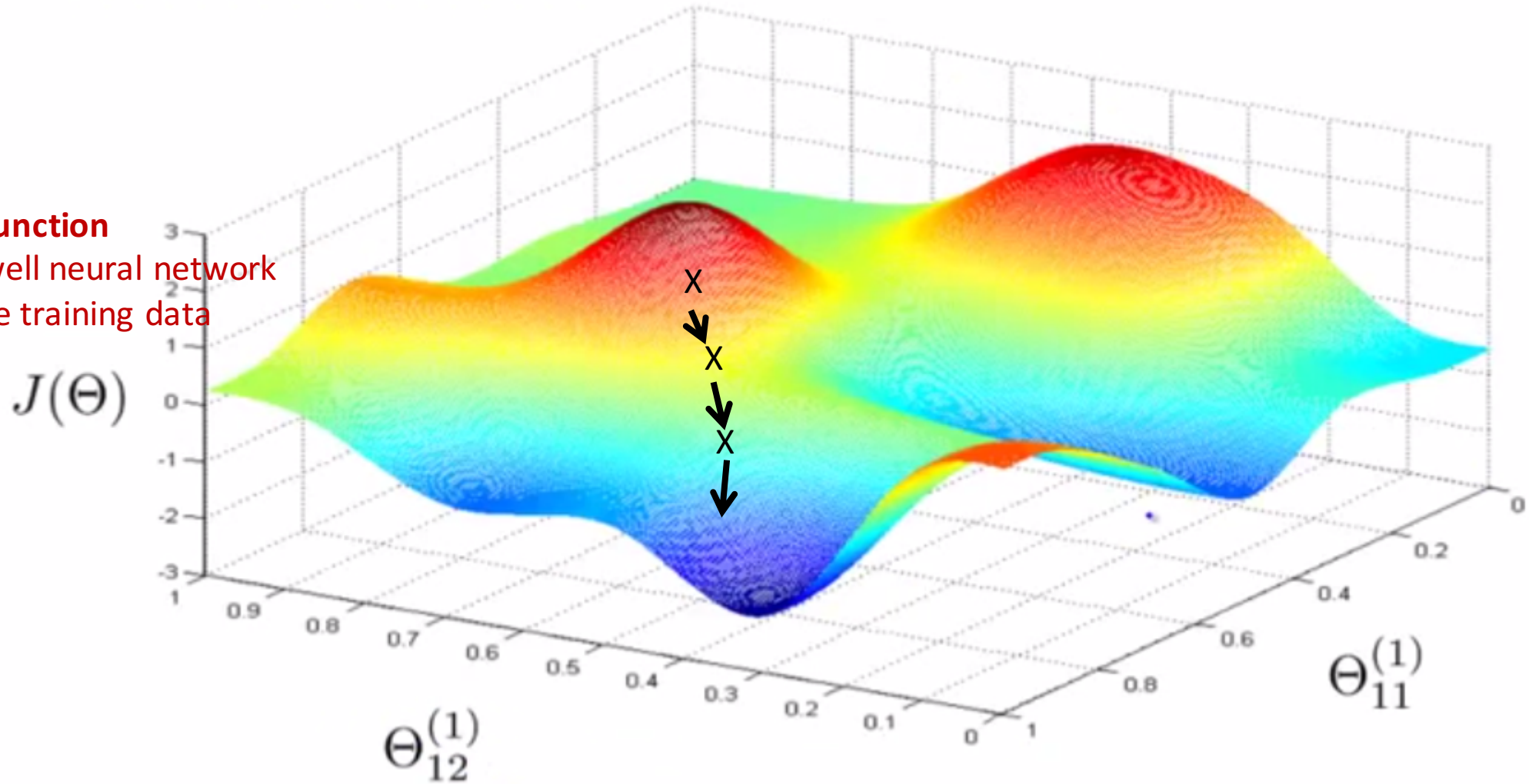
Fits the training data



Parameters of neural networks

Cost Function

How well neural network
Fits the training data



Parameters of neural networks

How many hidden layers?

Increasing layers

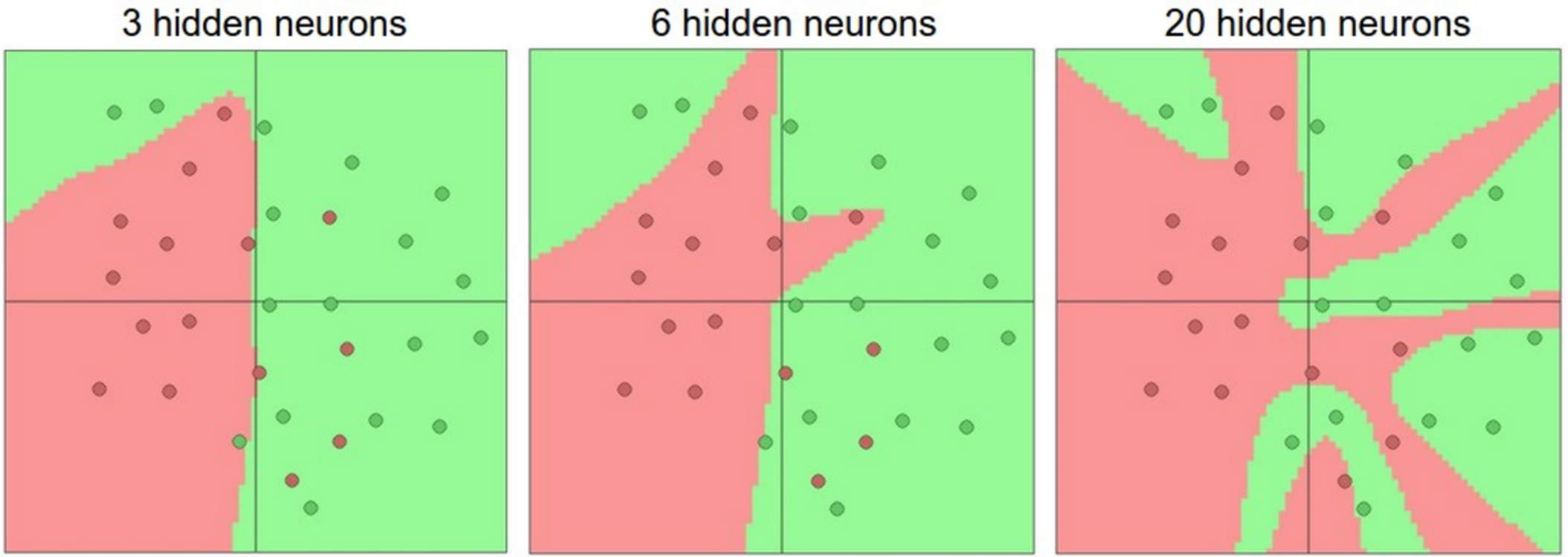
- In practice, 3-layer neural network will perform better than 2-layer
- Going deeper (4,5,6) does not help
- For Convolution Neural Network, depth (>= 10) is better.

	Architecture	# Param.	# Hidden units	Err.
DNN	2000-2000 + dropout	~10M	4k	57.8%
SNN-30k	128c-p-1200L-30k + dropout input&hidden	~70M	~190k	21.8%
single-layer feature extraction	4000c-p followed by SVM	~125M	~3.7B	18.4%
CNN[11] (no augmentation)	64c-p-64c-p-64c-p-16lc + dropout on lc	~10k	~110k	15.6%
CNN[21] (no augmentation)	64c-p-64c-p-128c-p-fc + dropout on fc and stochastic pooling	~56k	~120k	15.13%
teacher CNN (no augmentation)	128c-p-128c-p-128c-p-1000fc + dropout on fc and stochastic pooling	~35k	~210k	**12.0%**
ECNN (no augmentation)	ensemble of 4 CNNs	~140k	~840k	**11.0%**
SNN-CNN-MIMIC-30k trained on a single CNN	64c-p-1200L-30k with no regularization	~54M	~110k	**15.4%**
SNN-CNN-MIMIC-30k trained on a single CNN	128c-p-1200L-30k with no regularization	~70M	~190k	**15.1%**
SNN-ECNN-MIMIC-30k trained on ensemble	128c-p-1200L-30k with no regularization	~70M	~190k	**14.2%**

Table 2: Comparison of shallow and deep models: classification error rate on CIFAR-10. Key: c, convolution layer; p, pooling layer; lc, locally connected layer; fc, fully connected layer

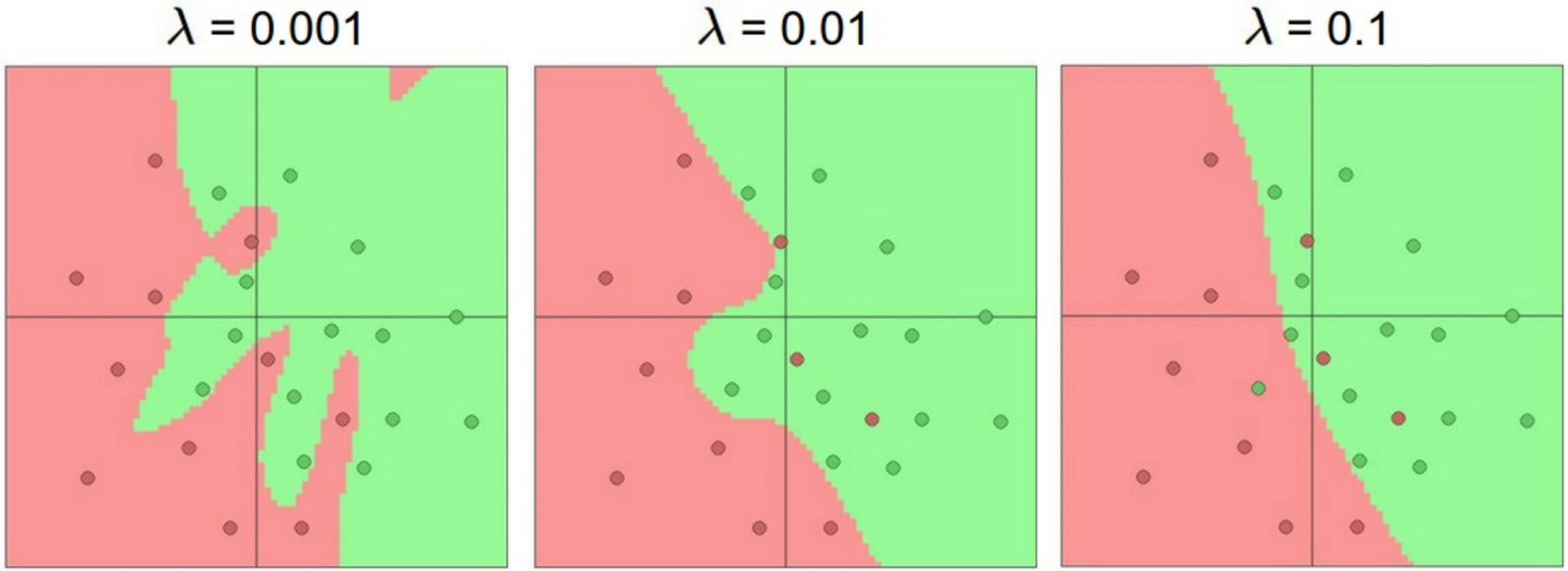
NIPS 2014 - Do Deep Nets Really Need to be Deep?, Lei Jimmy Ba, Rich Caruana

Layers and Size



Larger Neural Networks can represent more complicated functions. The data are shown as circles colored by their class, and the decision regions by a trained neural network are shown underneath. You can play with these examples in this ConvNetsJS demo.

Regularization helps to reduce overfitting



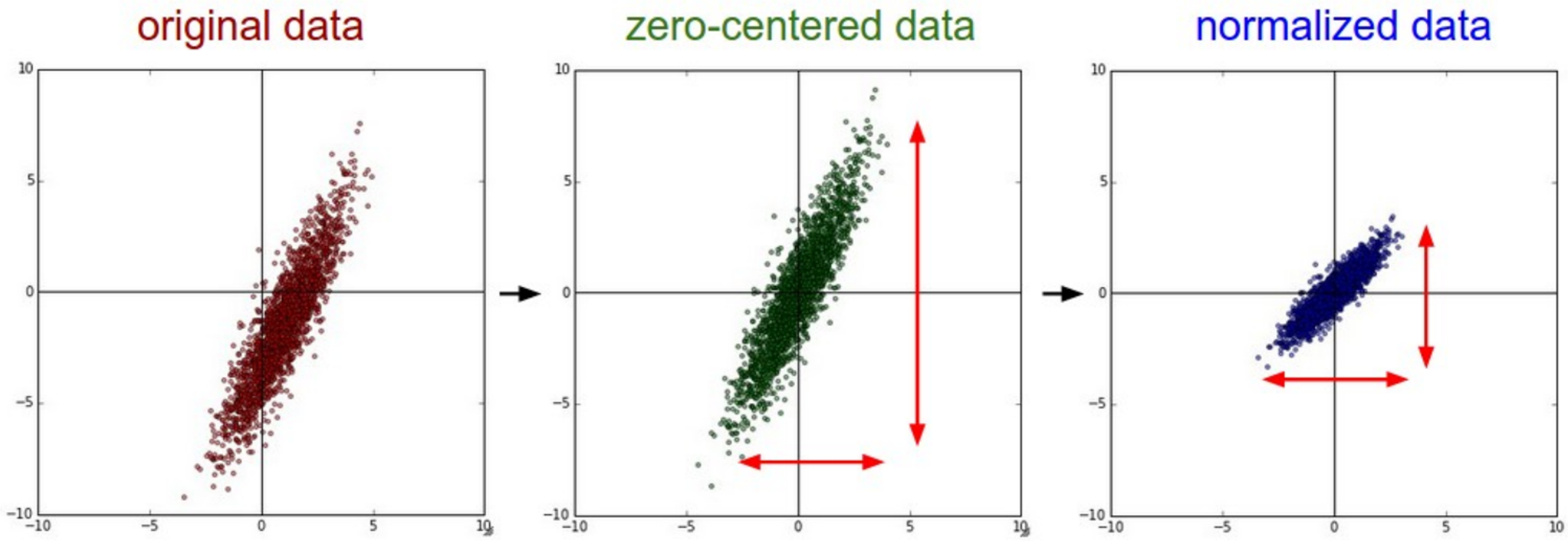
The effects of regularization strength: Each neural network above has 20 hidden neurons, but changing the regularization strength makes its final decision regions smoother with a higher regularization. You can play with these examples in this ConvNetsJS demo.

Data Preprocessing

Different Methods for Data Preprocessing

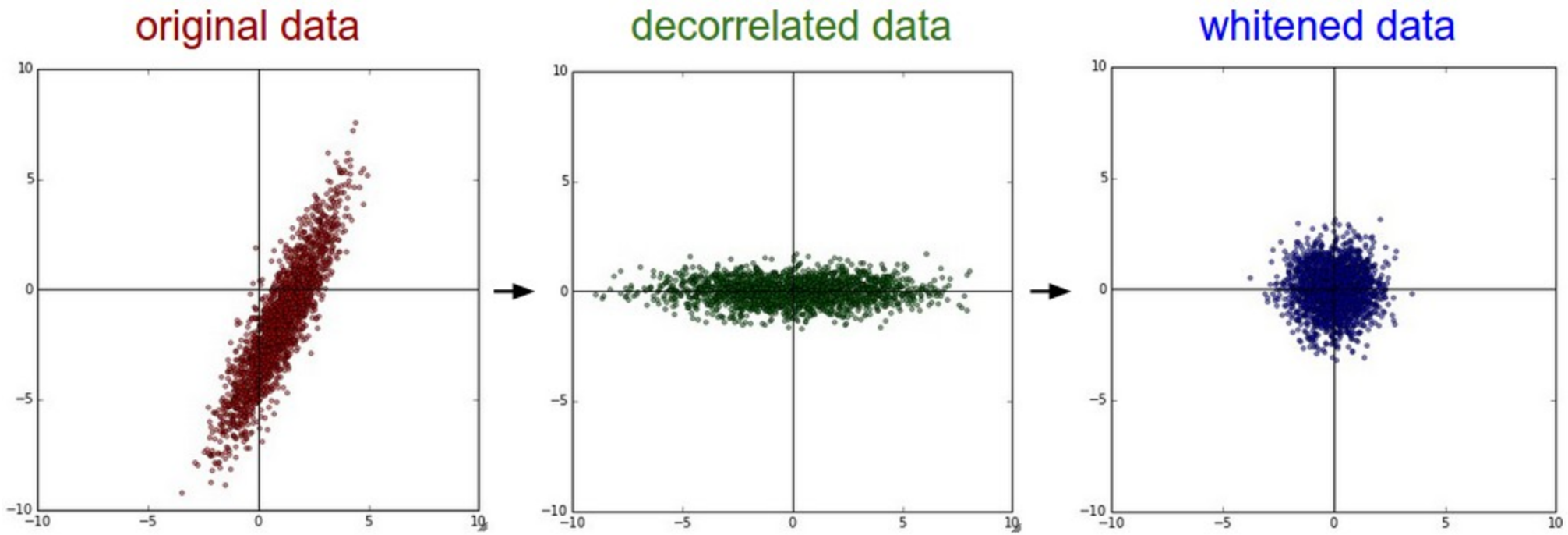
- Mean subtraction
- Normalization
- PCA and Whitening

Effects of Data Preprocessing



Common data preprocessing pipeline. **Left**: Original toy, 2-dimensional input data. **Middle**: The data is zero-centered by subtracting the mean in each dimension. The data cloud is now centered around the origin. **Right**: Each dimension is additionally scaled by its standard deviation. The red lines indicate the extent of the data - they are of unequal length in the middle, but of equal length on the right.

PCA and Whitening



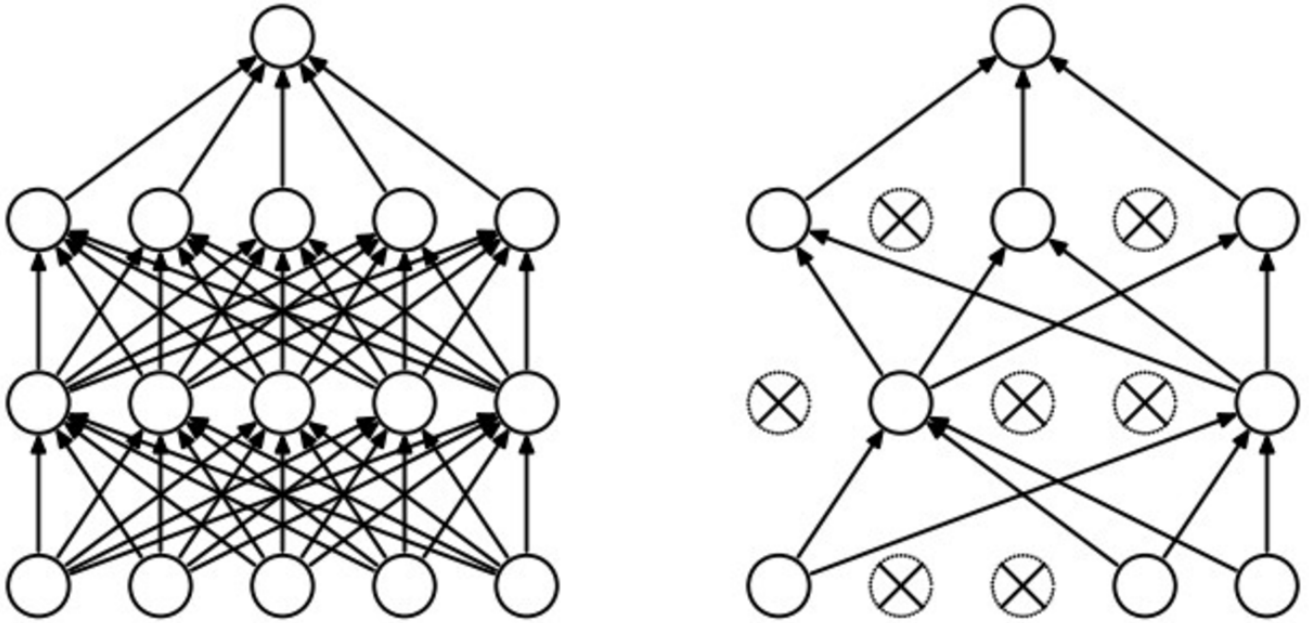
PCA / Whitening. **Left**: Original toy, 2-dimensional input data. **Middle**: After performing PCA. The data is centered at zero and then rotated into the eigenbasis of the data covariance matrix. This decorrelates the data (the covariance matrix becomes diagonal). **Right**: Each dimension is additionally scaled by the eigenvalues, transforming the data covariance matrix into the identity matrix. Geometrically, this corresponds to stretching and squeezing the data into an isotropic gaussian blob.

Managing Overfitting

Controlling Neural Network's Capacity

- Reduce overfitting by
  - L2 Regularization
  - L1 Regularization
  - Max norm constraints
  - Dropout

Dropout

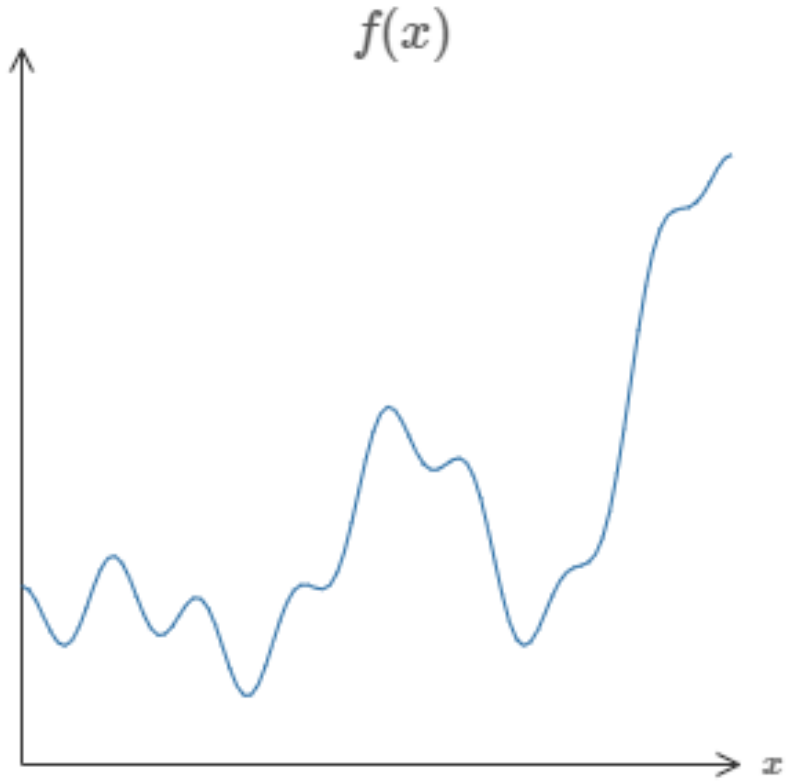


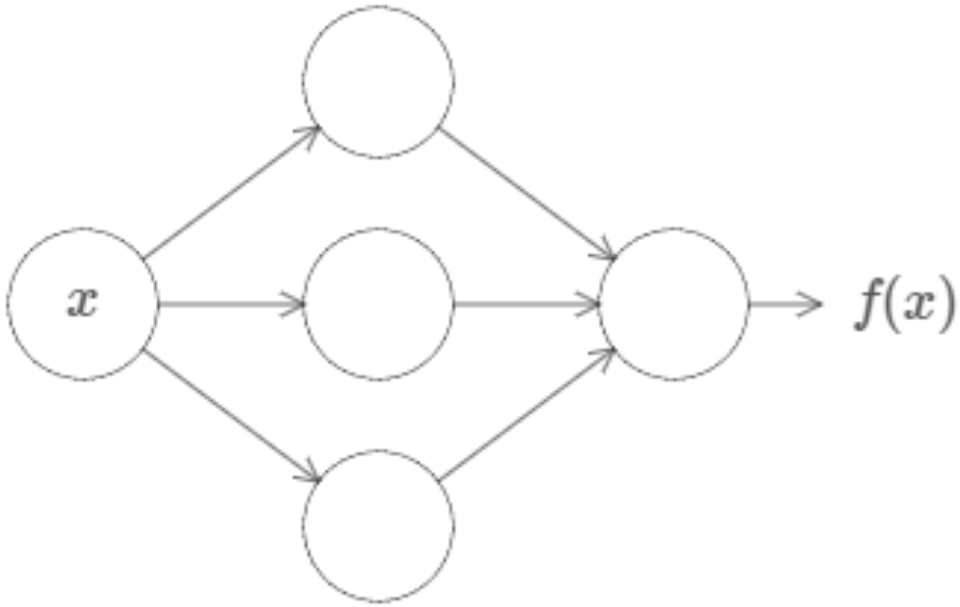
(a) Standard Neural Net (b) After applying dropout.

Figure taken from the Dropout paper that illustrates the idea. During training, Dropout can be interpreted as sampling a Neural Network within the full Neural Network, and only updating the parameters of the sampled network based on the input data. (However, the exponential number of possible sampled networks are not independent because they share the parameters.) During testing there is no dropout applied, with the interpretation of evaluating an averaged prediction across the exponentially-sized ensemble of all sub-networks (more about ensembles in the next section).

Universality of Neural Networks

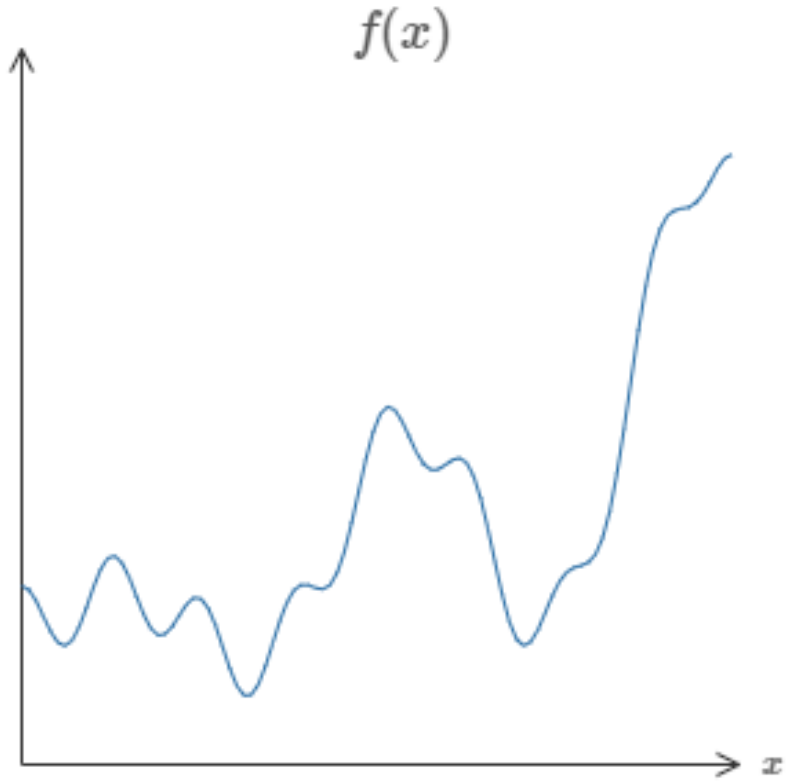
Neural Nets can Compute Any Function

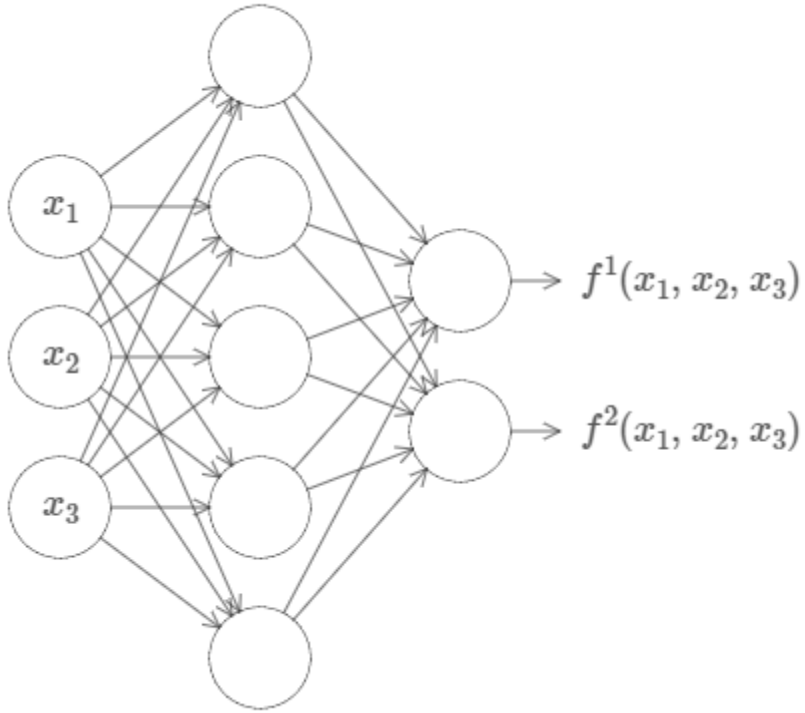




Credits: Visual Proff that neural nets can compute any function
http://neuralnetworksanddeeplearning.com/chap4.html

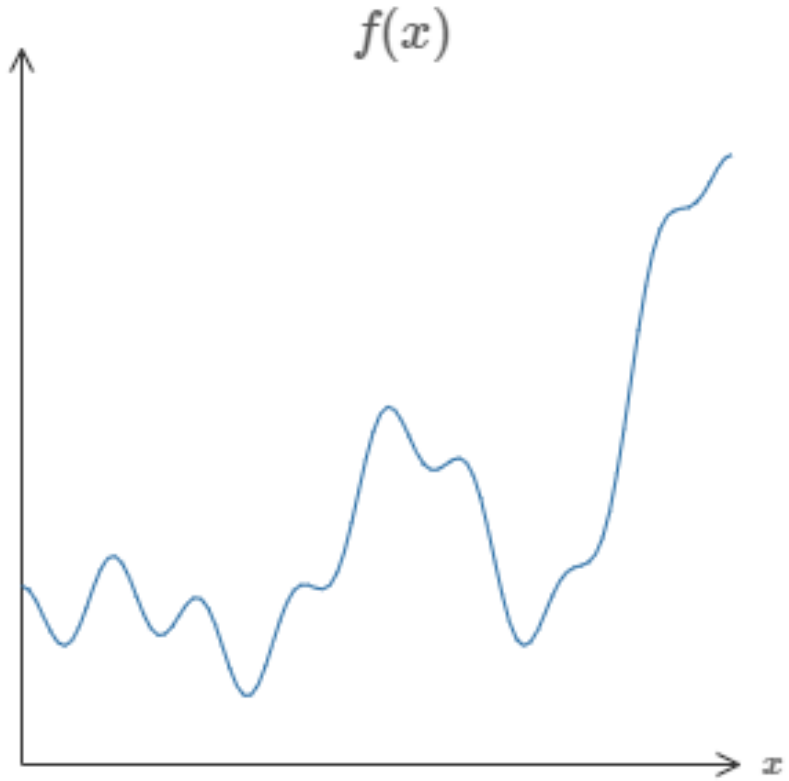
Neural Nets can Compute Any Function



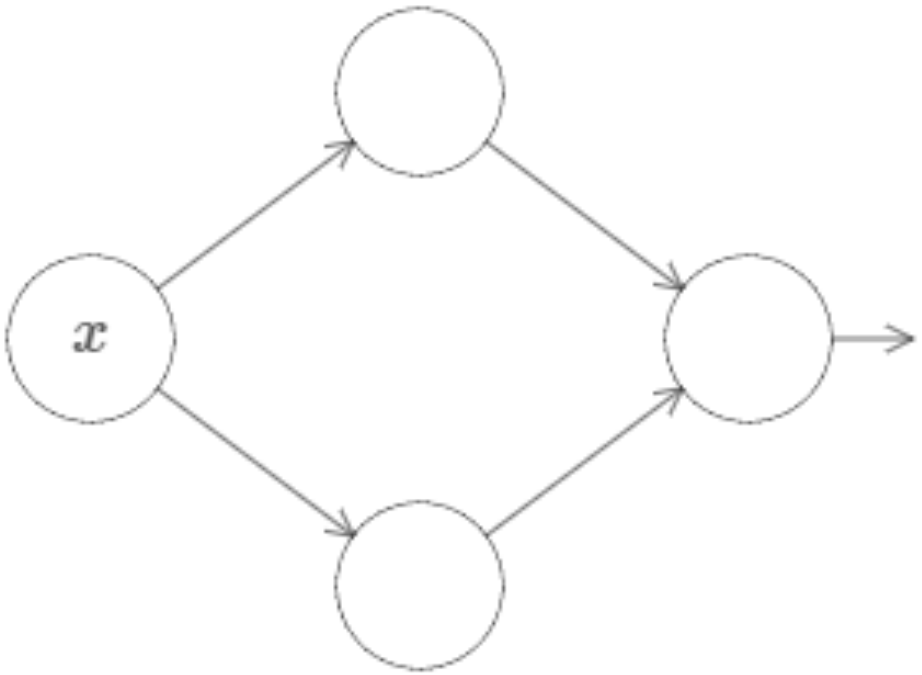


Credits: Visual Proff that neural nets can compute any function
http://neuralnetworksanddeeplearning.com/chap4.html

Universality – 1 input, 1 output

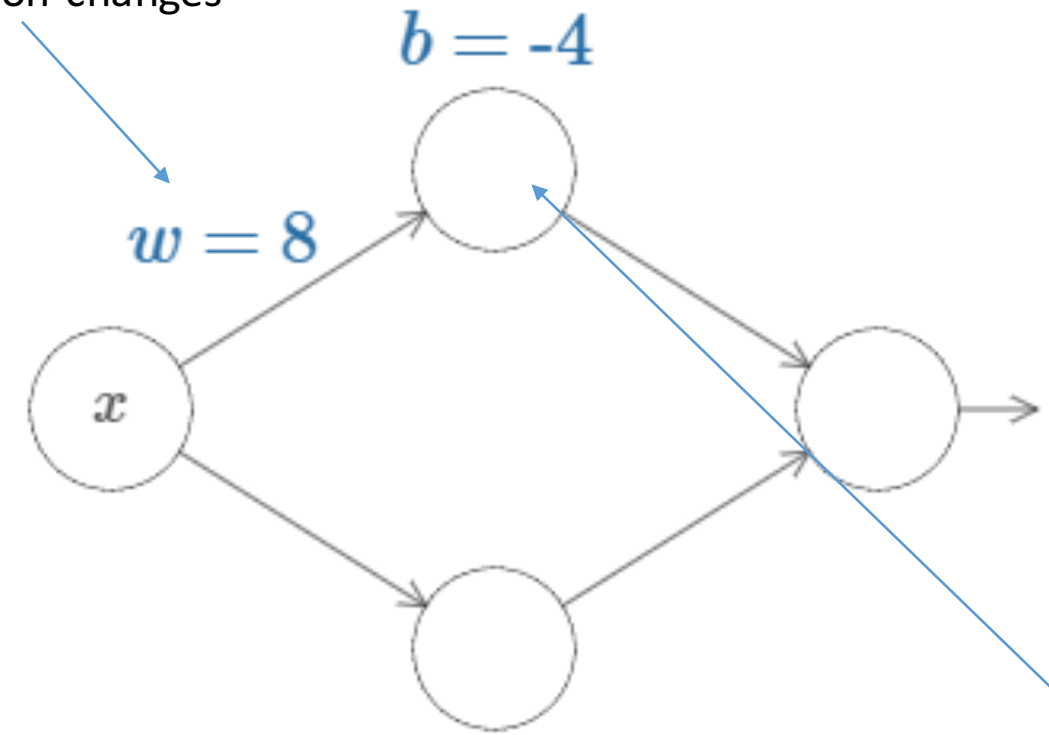


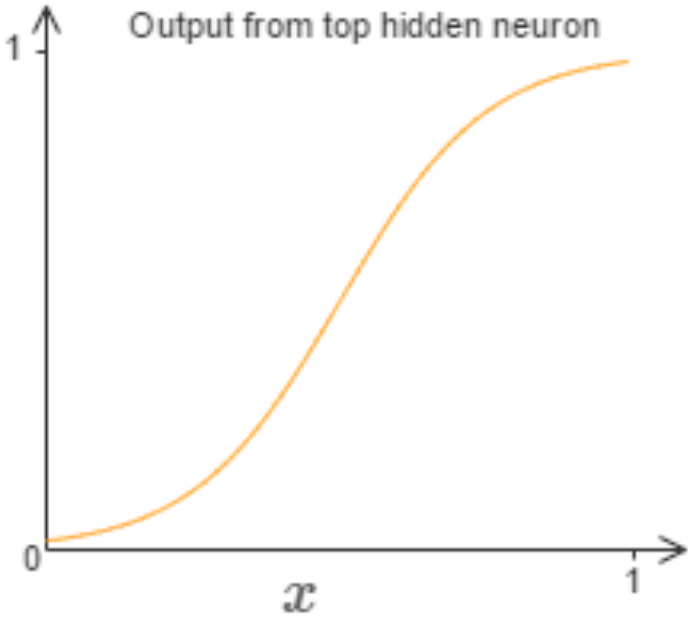
Universality – 1 input, 1 output



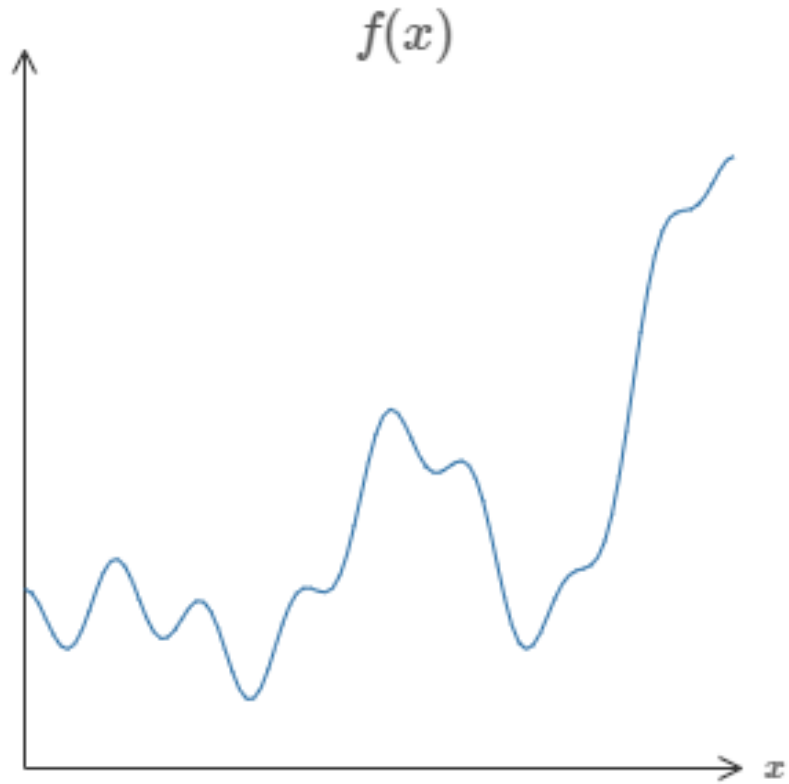
Universality – 1 input, 1 output

Try increasing w
Function computed by the top hidden neuron changes

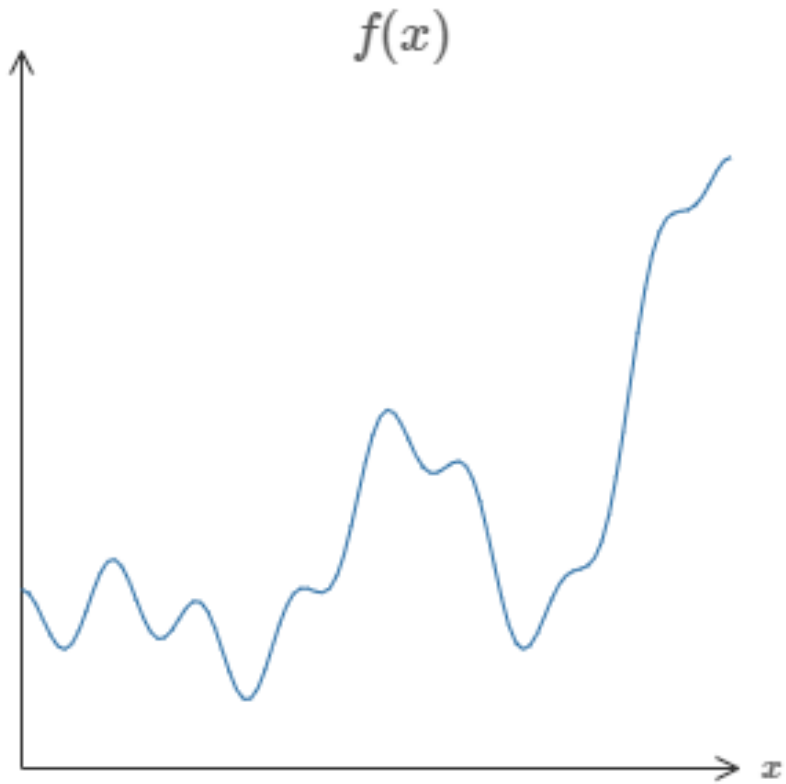


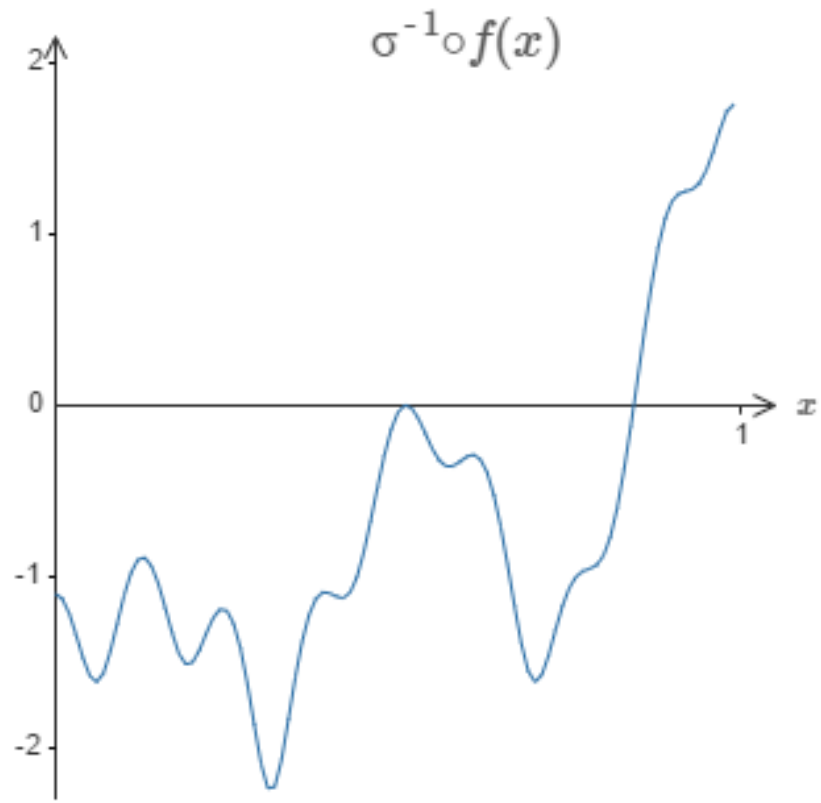


σ(wx+b),
where σ(z)≡1/(1+e^{-z})



$$f(x) = 0.2 + 0.4x^2 + 0.3x\sin(15x) + 0.05\cos(50x),$$





Design a neural network whose hidden layer has a weighted output given by **σ−1∘f(x)**, where σ^{-1} is just the inverse of the σ function.

Weighted output of the hidden layer to be the inverse function. This will be a good approximate to f(x)*