Neural Networks

Spring 2018

Review

- Last week:
  - History of Modern Neural Networks and “Deep Learning”
  - Single Layer Neural Network: Perceptron
  - Single Layer Neural Network: Adaline
  - Variants of Gradient Descent
- Assignments (Canvas):
  - Problem set due yesterday
  - New lab assignment out
- Questions?

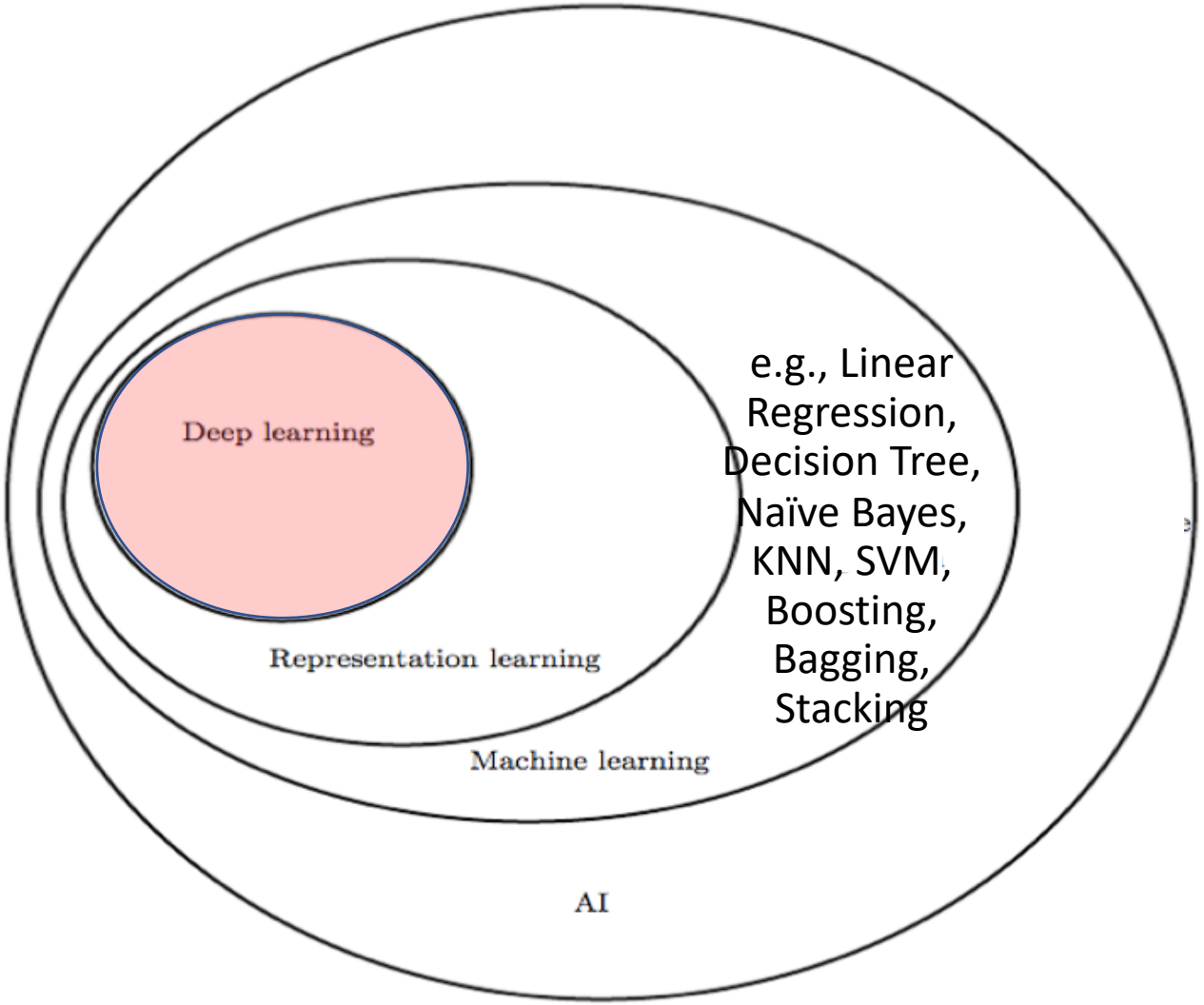
Today’s Topics

- Revisiting History of Modern Neural Networks and “Deep Learning”
- Neural Network Architecture
- Training a Neural Network
- Lab

Today's Topics

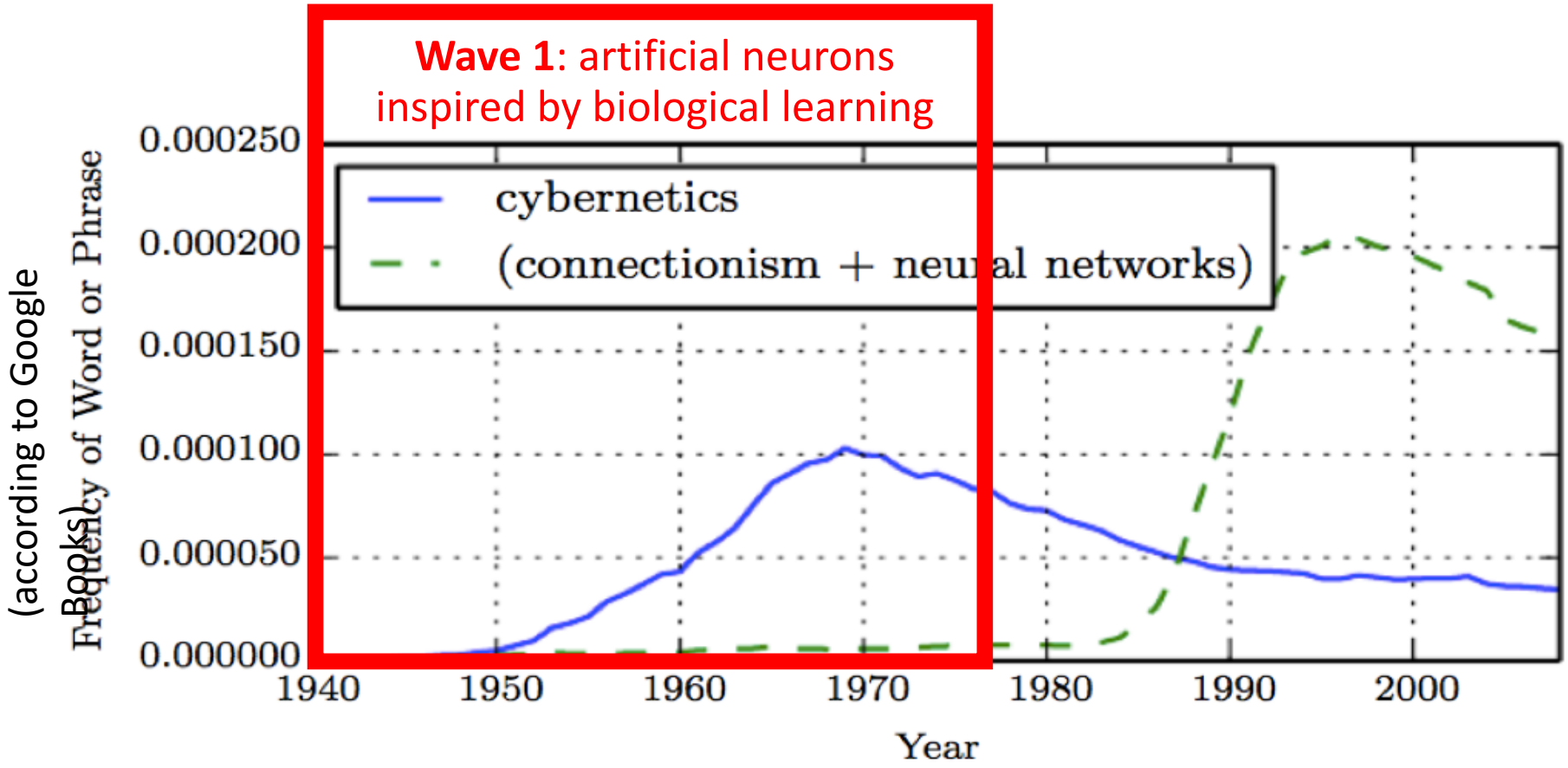
- Revisiting History of Modern Neural Networks and "Deep Learning"
- Neural Network Architecture
- Training a Neural Network
- Lab

Current Class Focus: Deep Learning



Ian Goodfellow, Yoshua Bengio, and Aaron Courville; Deep Learning, 2016.

Last Week: Wave 1 of "Deep Learning"

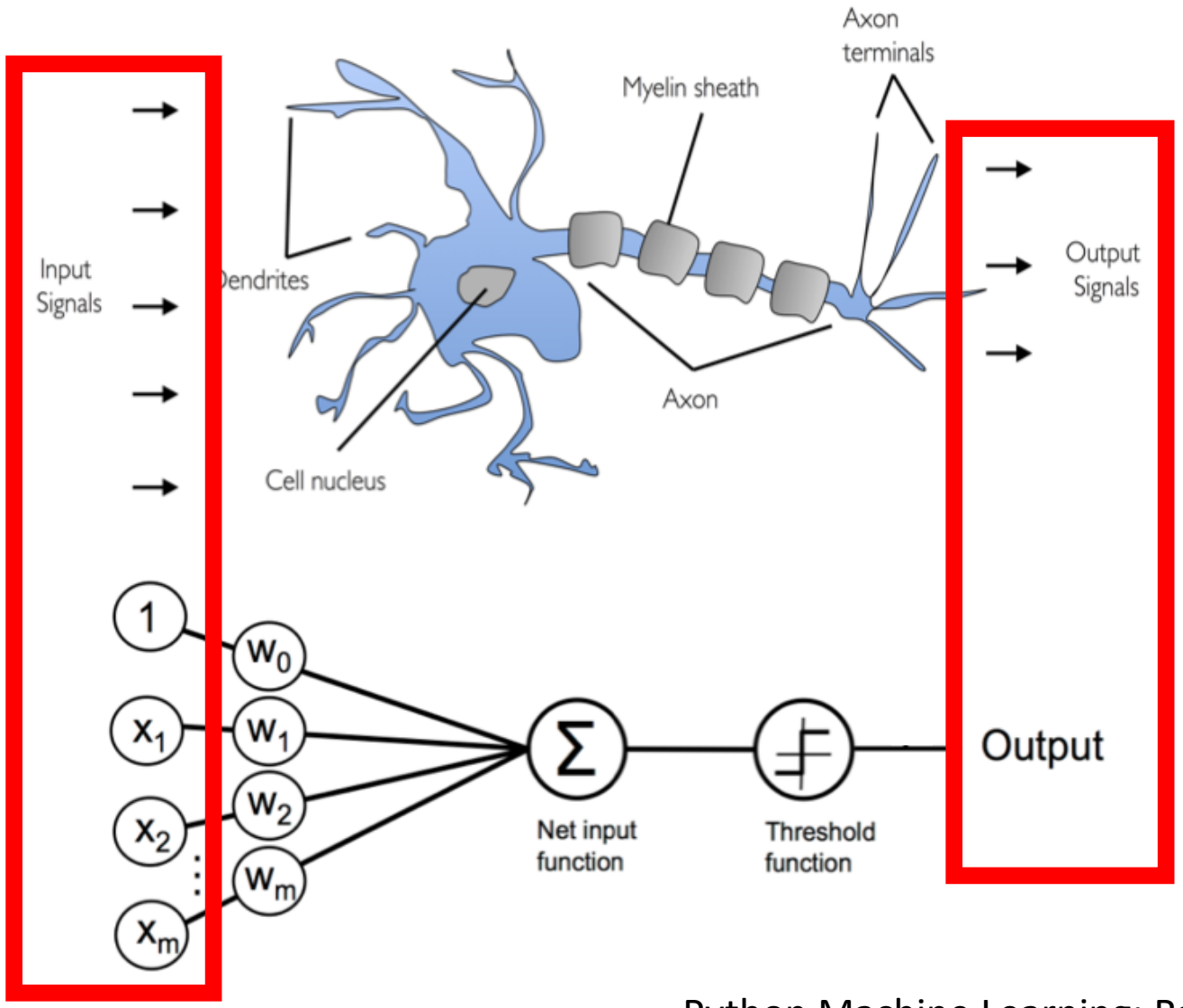


Ian Goodfellow, Yoshua Bengio, and Aaron Courville; Deep Learning, 2016.

Inspiration

Biological Neuron:

Artificial Neurons
(e.g., Perceptron):



Python Machine Learning; Raschka & Mirjalili

https://github.com/rasbt/python-machine-learning-book-2nd-edition/blob/master/code/ch02/ch02.ipynb

AI Birth Followed by First "AI Winter"

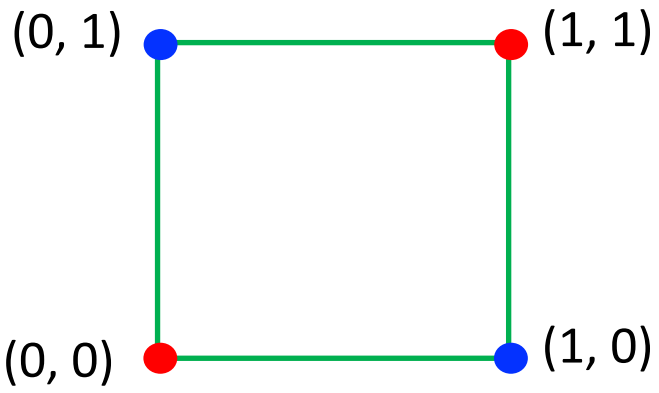
1957: Frank Rosenblatt, inventor of perceptron, predicts:

> The perceptron will be "the embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence."

1969: an AI founder, Marvin Minsky, publishes a book called "Perceptrons" to discuss its limitations:

> e.g., how to separate 1s from 0s? (XOR)
>
> Perceptron can only handle linear functions! How can a machine be "conscious" when it can't do XOR?

INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0



1974–80: criticisms led to first "AI Winter",
a time period of reduced funding and research

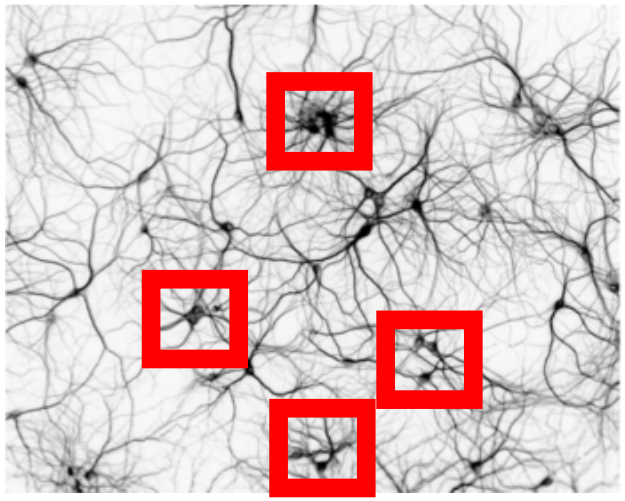
https://people.eecs.berkeley.edu/~jrs/189/lec/17.pdf

How to Overcome Limitation of Linear Model?

- e.g. linear regression: perform non-linear transformation of input features
- e.g., SVM: apply kernel trick to project data into a higher dimensional space
- e.g., neural networks

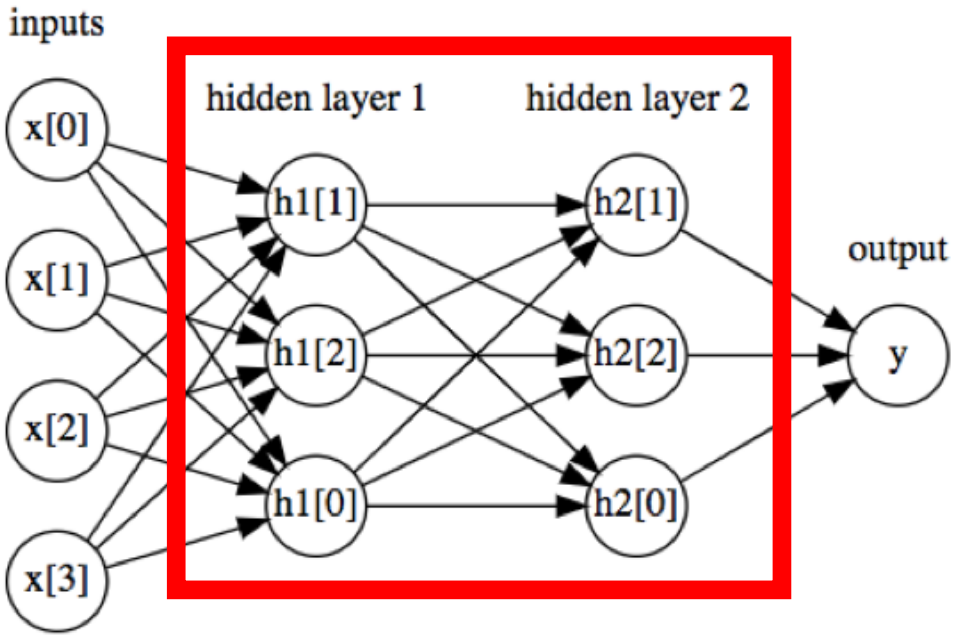
Inspiration: Neurons are “Connected”

Biological Neural Network:



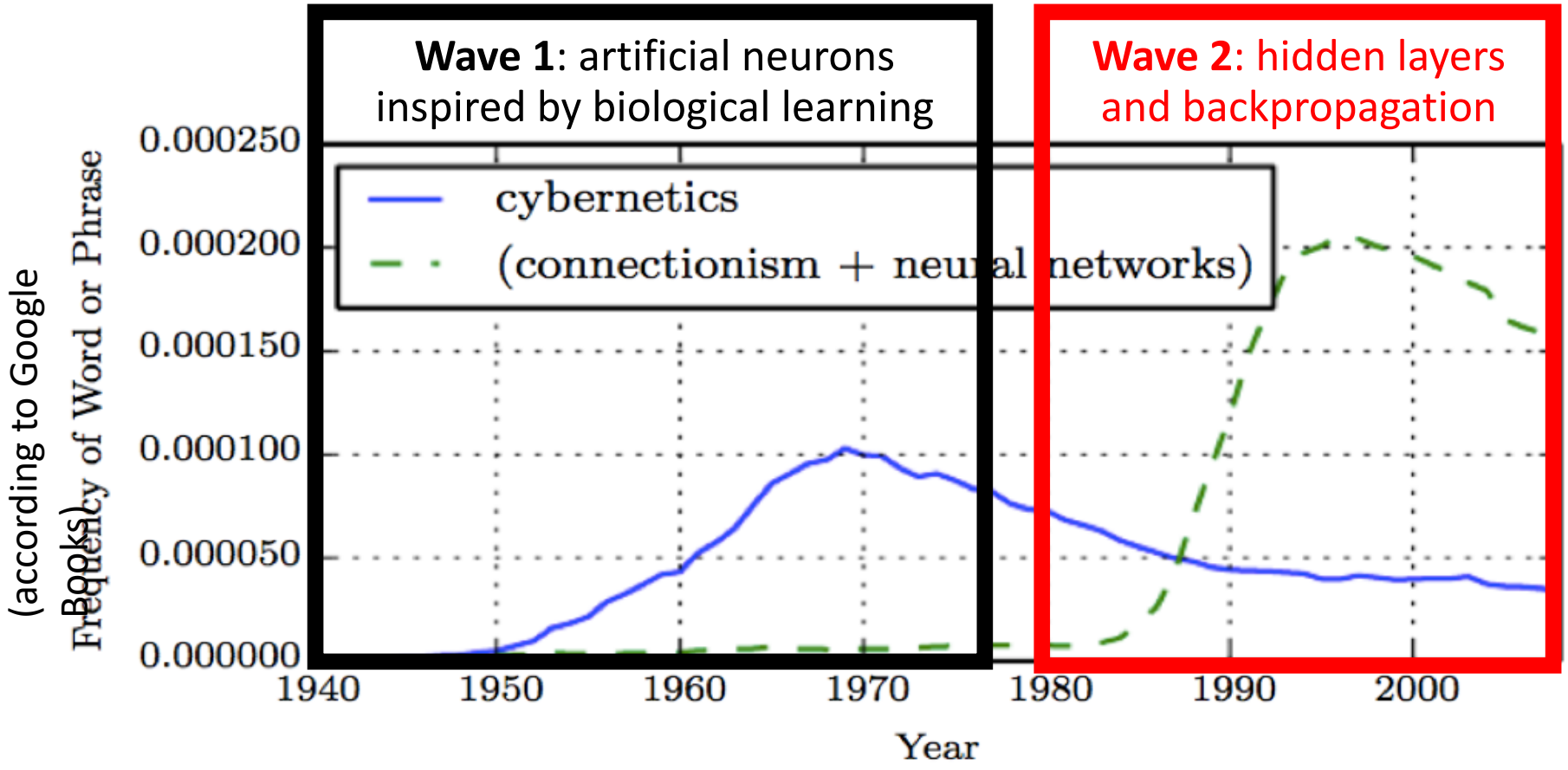
http://www.rzagabe.com/2014/11/03/an-introduction-to-artificial-neural-networks.html

Artificial Neural Network:



https://github.com/amueller/introduction_to_ml_with_python/blob/master/02-supervised-learning.ipynb

Today's Topic: Neural Networks

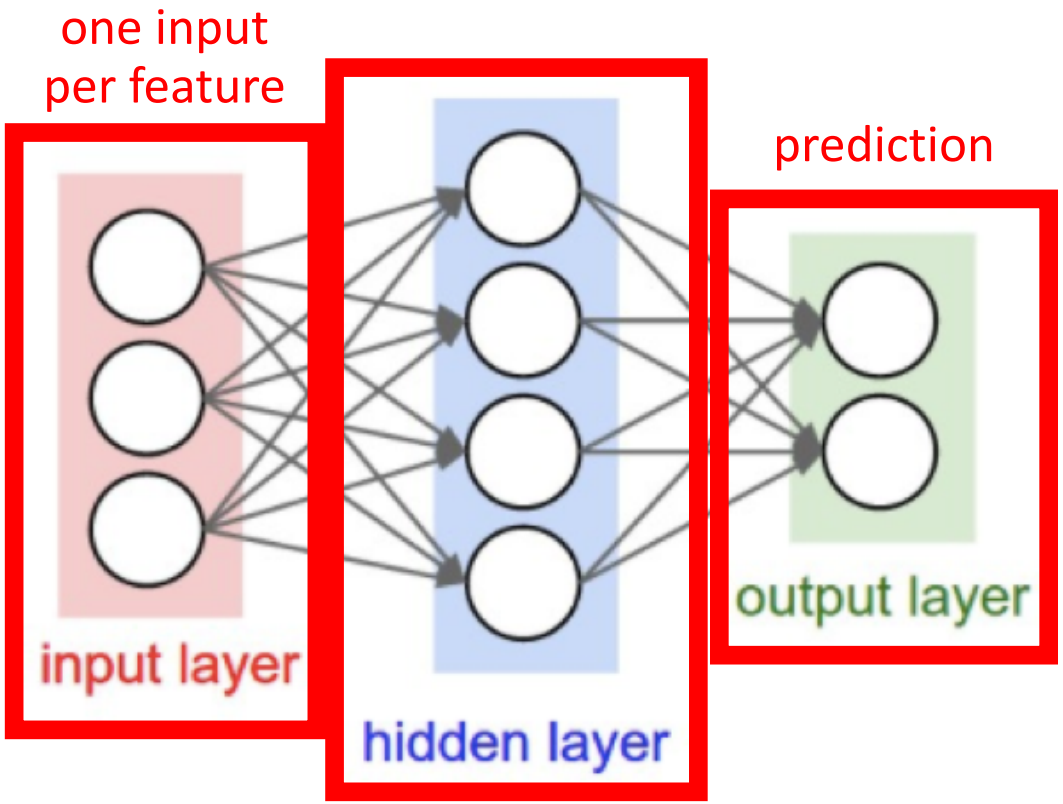


Ian Goodfellow, Yoshua Bengio, and Aaron Courville; Deep Learning, 2016.

Today's Topics

- Revisiting History of Modern Neural Networks and "Deep Learning"
- Neural Network Architecture
- Training Neural Network
- Lab

Neural Network

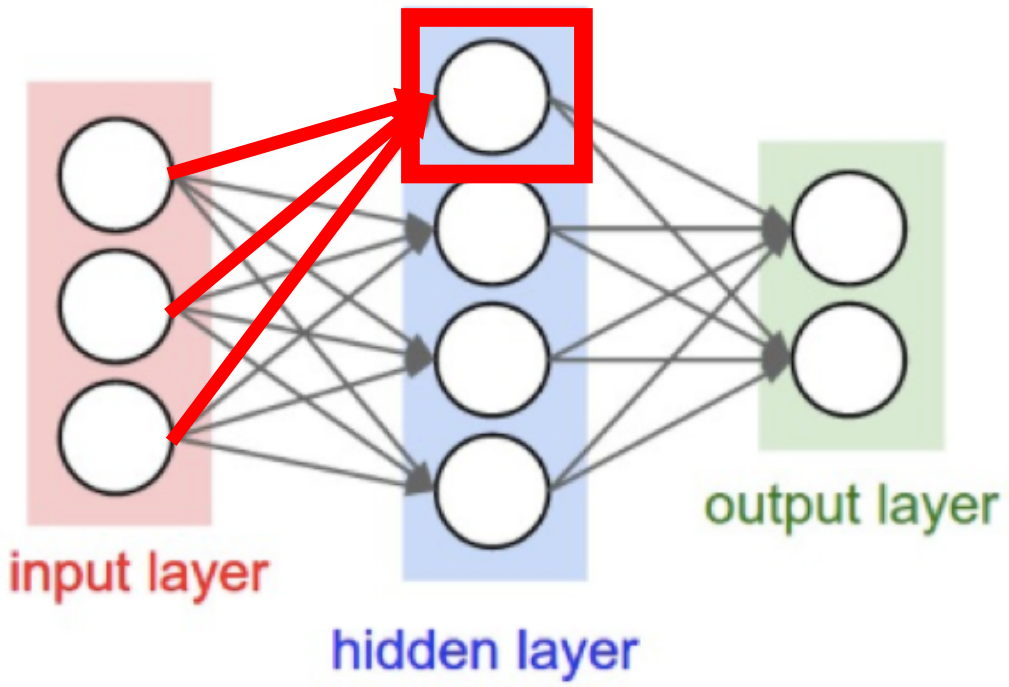


"hidden layer" uses outputs of units (i.e., neurons) and provides them as inputs to other units (i.e., neurons)

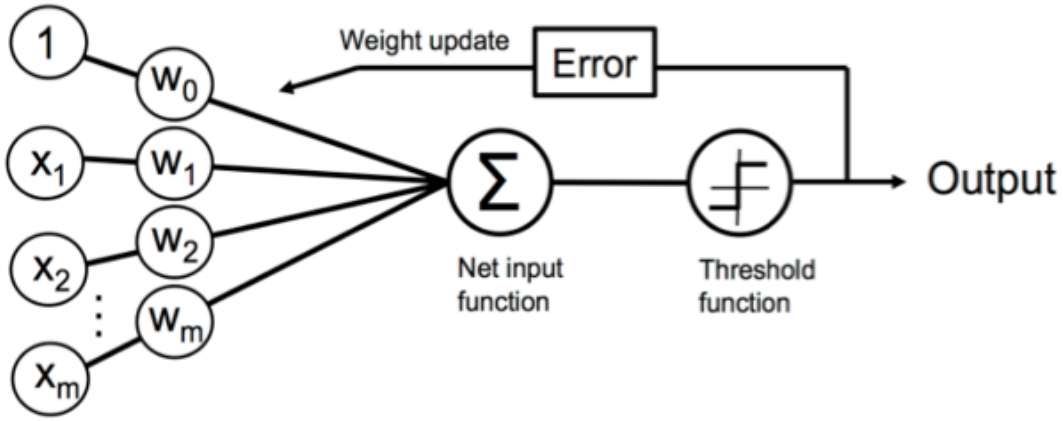
- Also called "multilayer perceptron"
- This is a 2-layer neural network (number of hidden layers plus output layer; exclude input layer when counting)

http://cs231n.github.io/neural-networks-1/

Neural Network



- How does this relate to a perceptron?

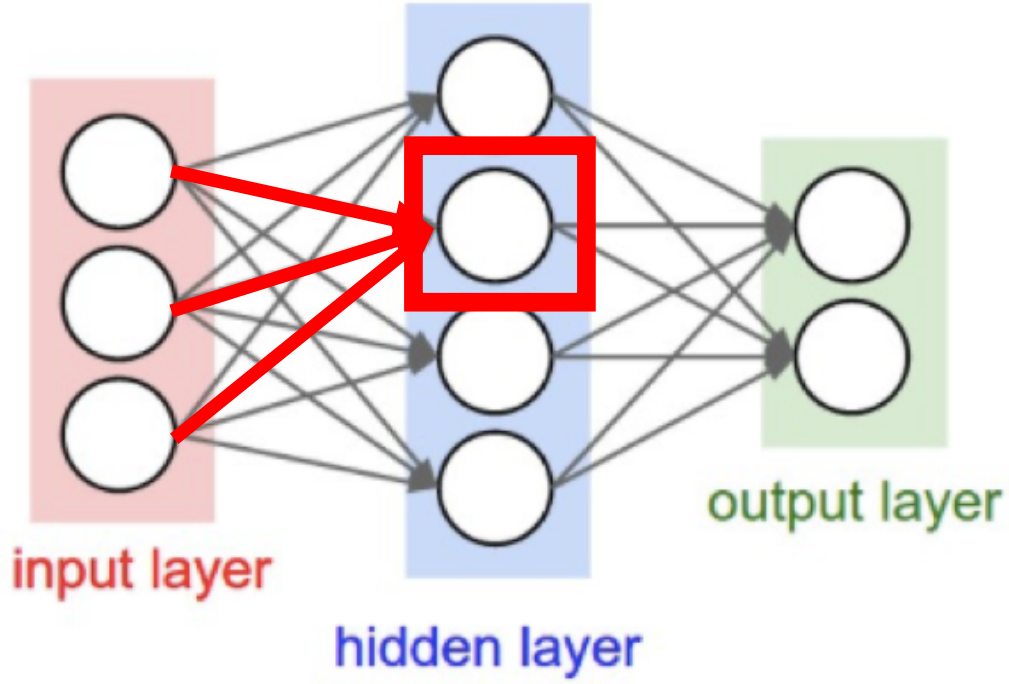


- Unit: takes as input a weighted sum and applies a non-linear (activation) function

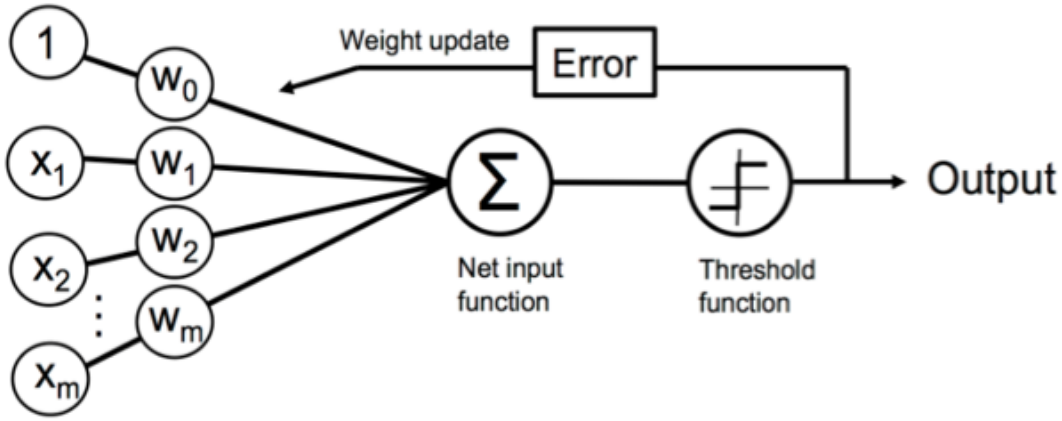
Python Machine Learning; Raschka & Mirjalili

http://cs231n.github.io/neural-networks-1/

Neural Network



- How does this relate to a perceptron?

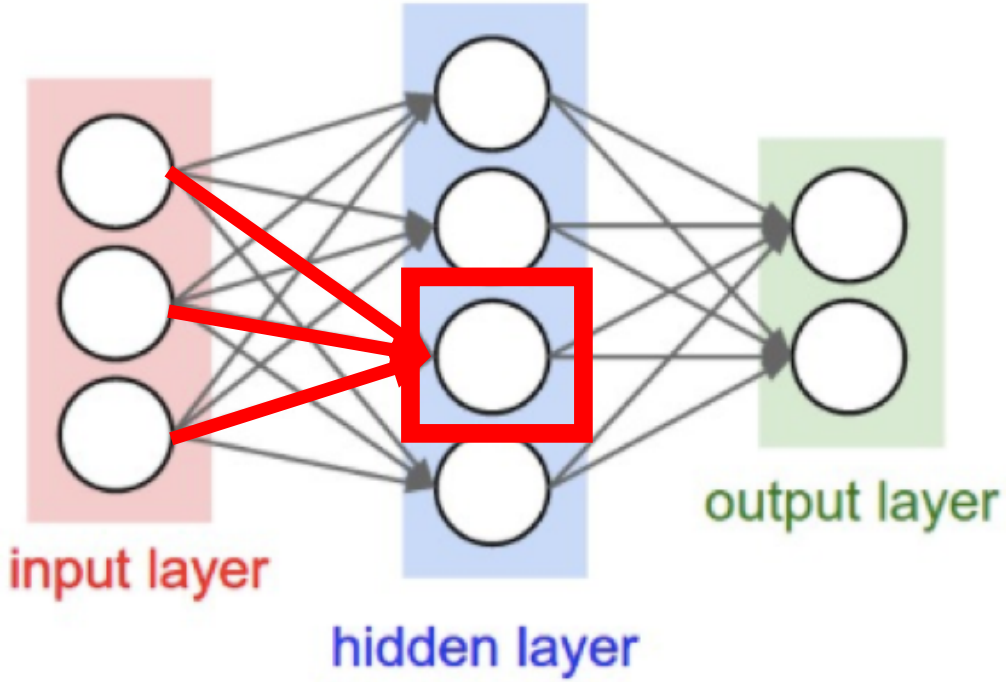


- Unit: takes as input a weighted sum and applies a non-linear (activation) function

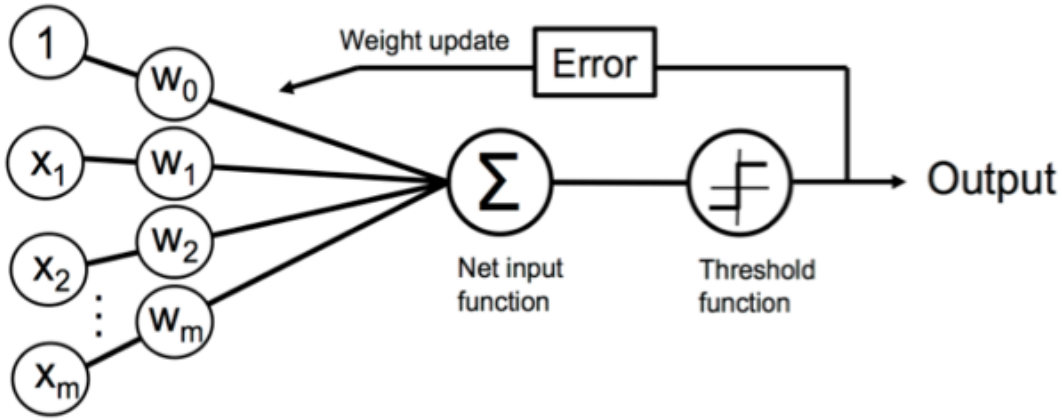
Python Machine Learning; Raschka & Mirjalili

http://cs231n.github.io/neural-networks-1/

Neural Network



- How does this relate to a perceptron?

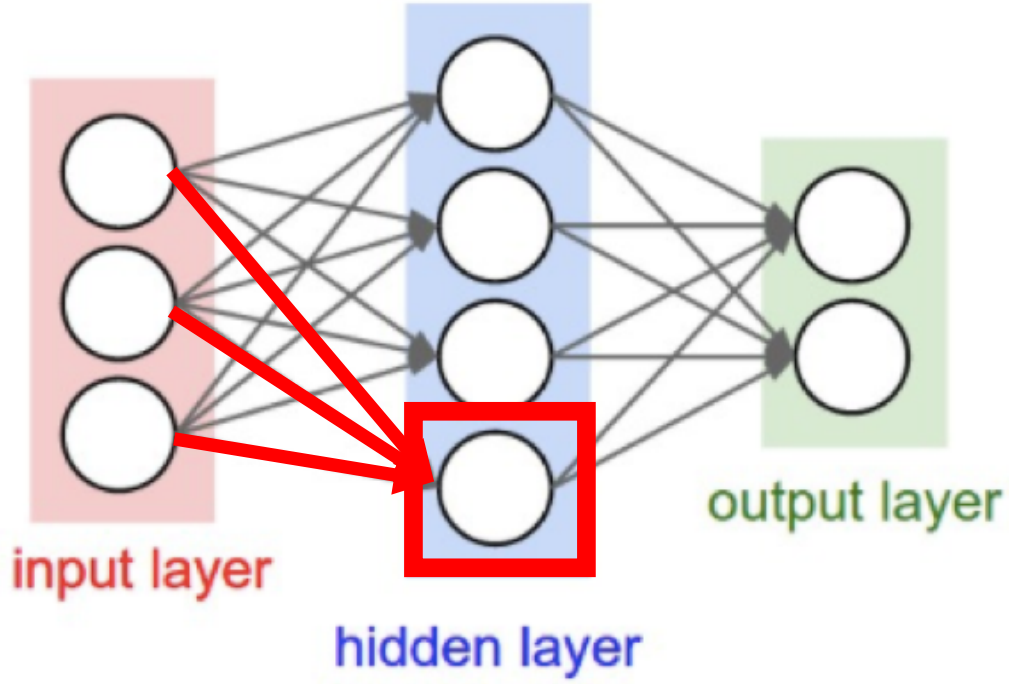


- Unit: takes as input a weighted sum and applies a non-linear (activation) function

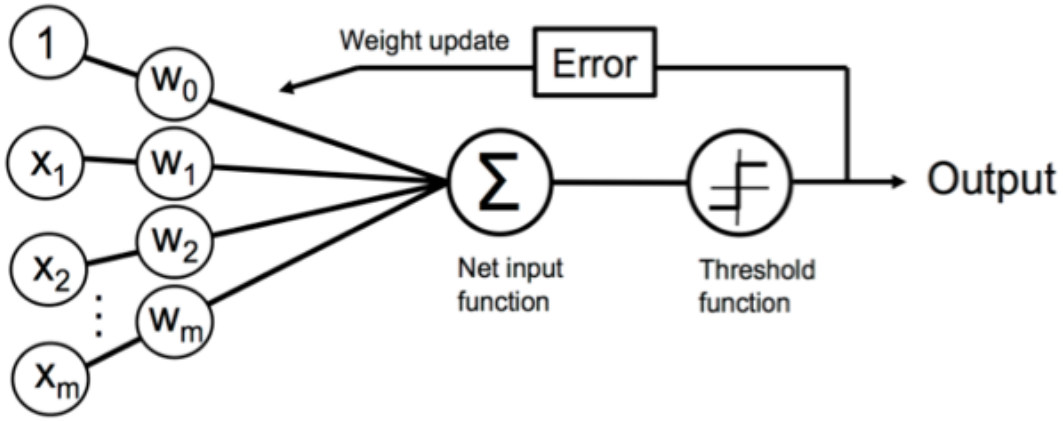
Python Machine Learning; Raschka & Mirjalili

http://cs231n.github.io/neural-networks-1/

Neural Network



- How does this relate to a perceptron?

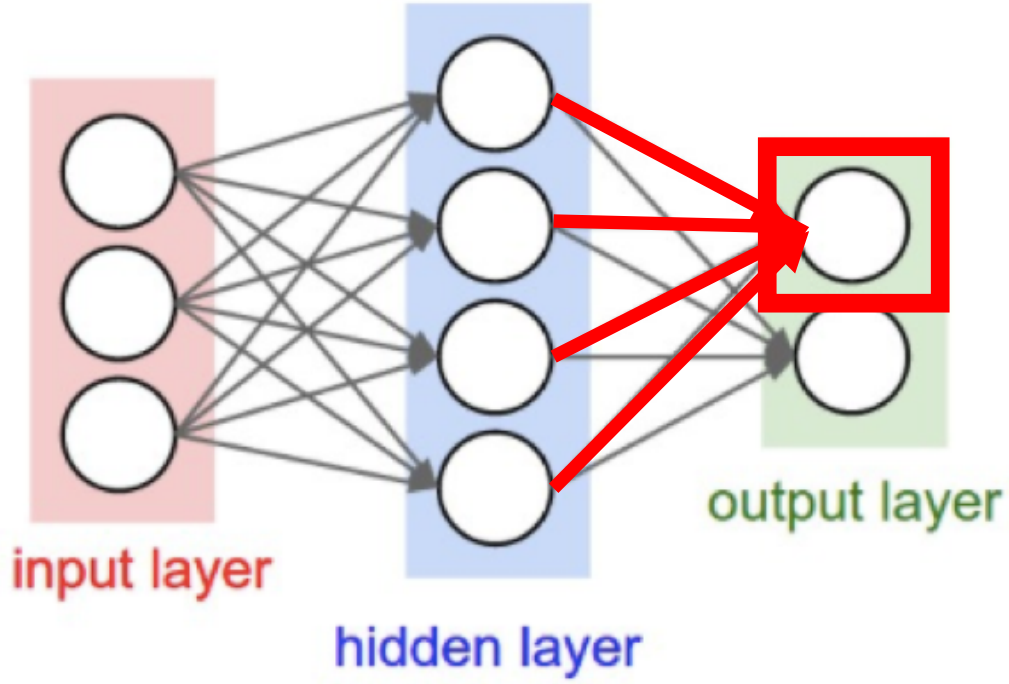


- Unit: takes as input a weighted sum and applies a non-linear (activation) function

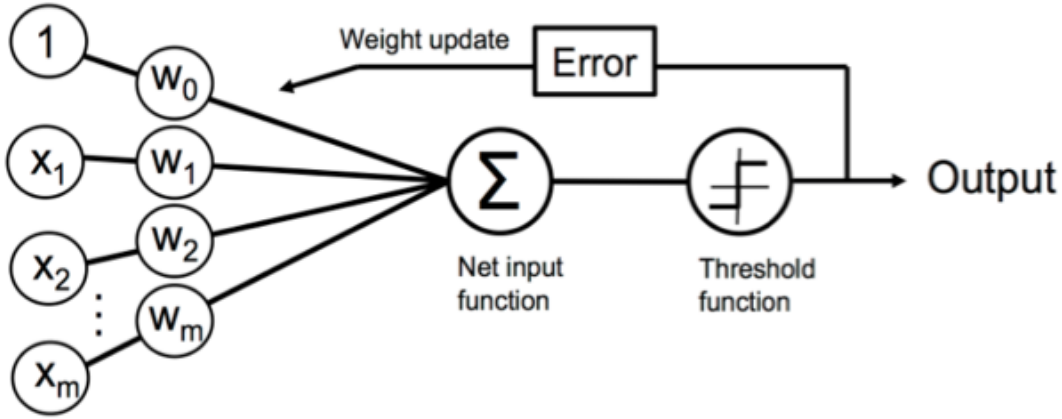
Python Machine Learning; Raschka & Mirjalili

http://cs231n.github.io/neural-networks-1/

Neural Network



- How does this relate to a perceptron?

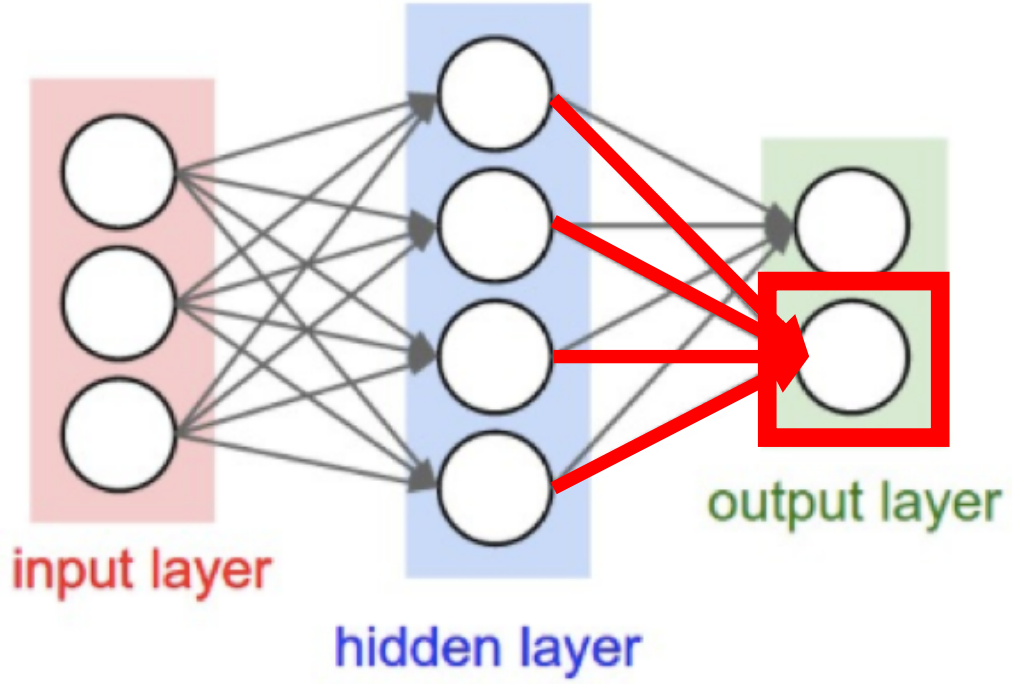


- Unit: takes as input a weighted sum and applies a non-linear (activation) function

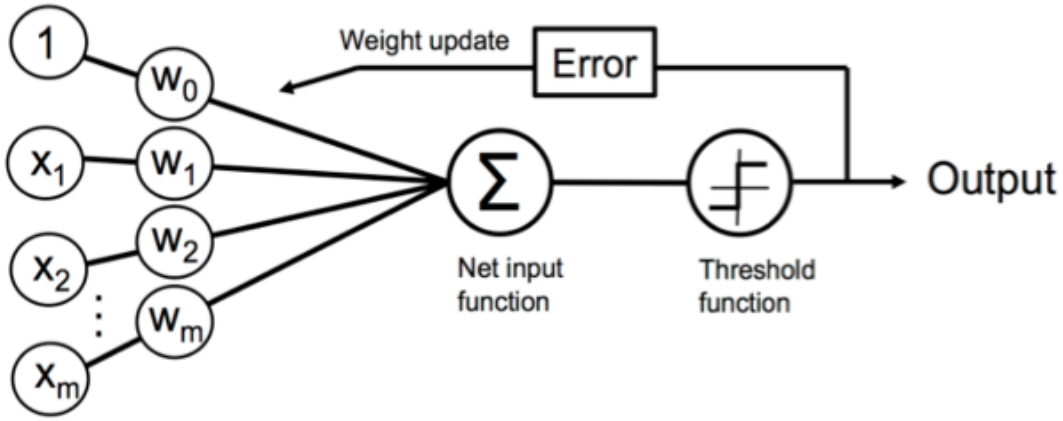
Python Machine Learning; Raschka & Mirjalili

http://cs231n.github.io/neural-networks-1/

Neural Network



- How does this relate to a perceptron?

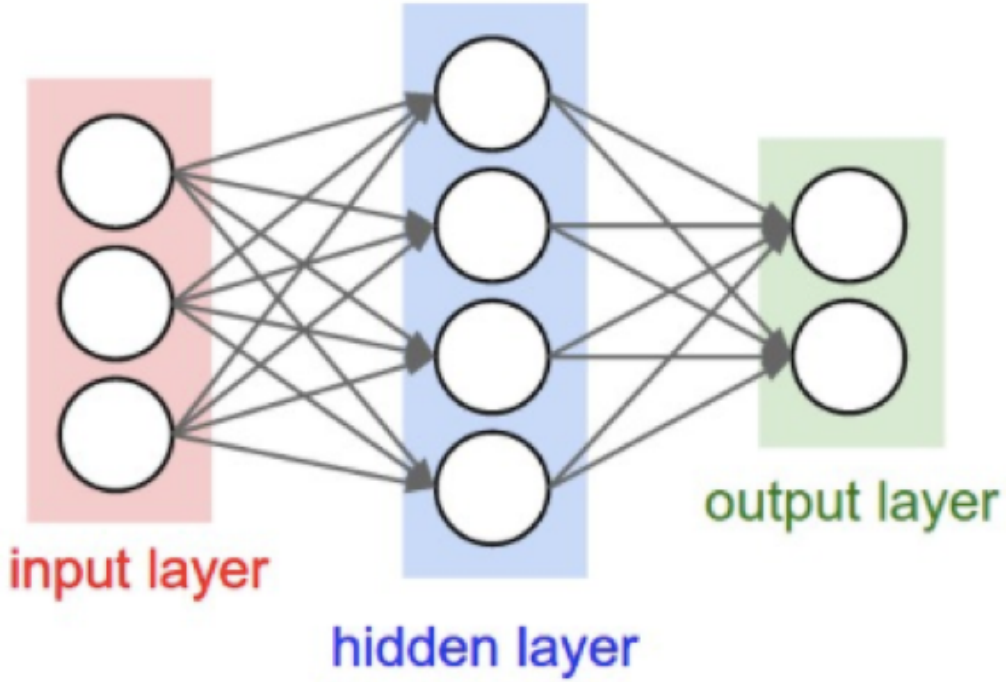


- Unit: takes as input a weighted sum and applies a non-linear (activation) function

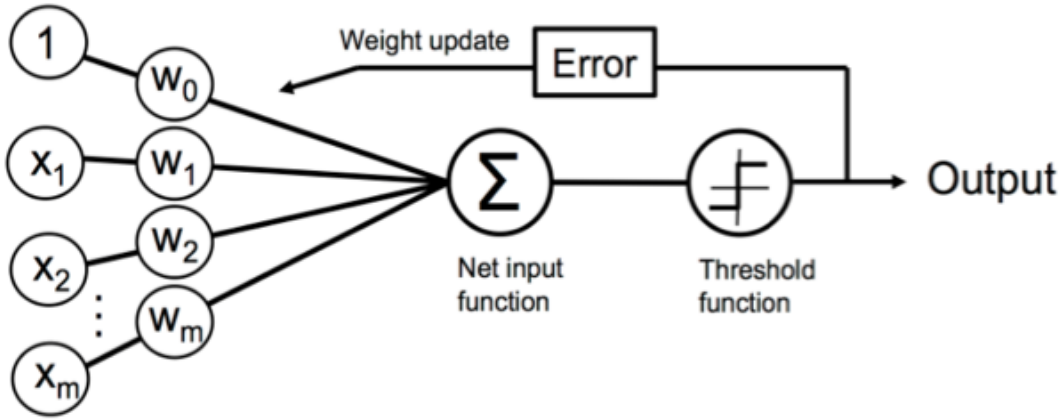
Python Machine Learning; Raschka & Mirjalili

http://cs231n.github.io/neural-networks-1/

Neural Network



- How does this relate to a perceptron?

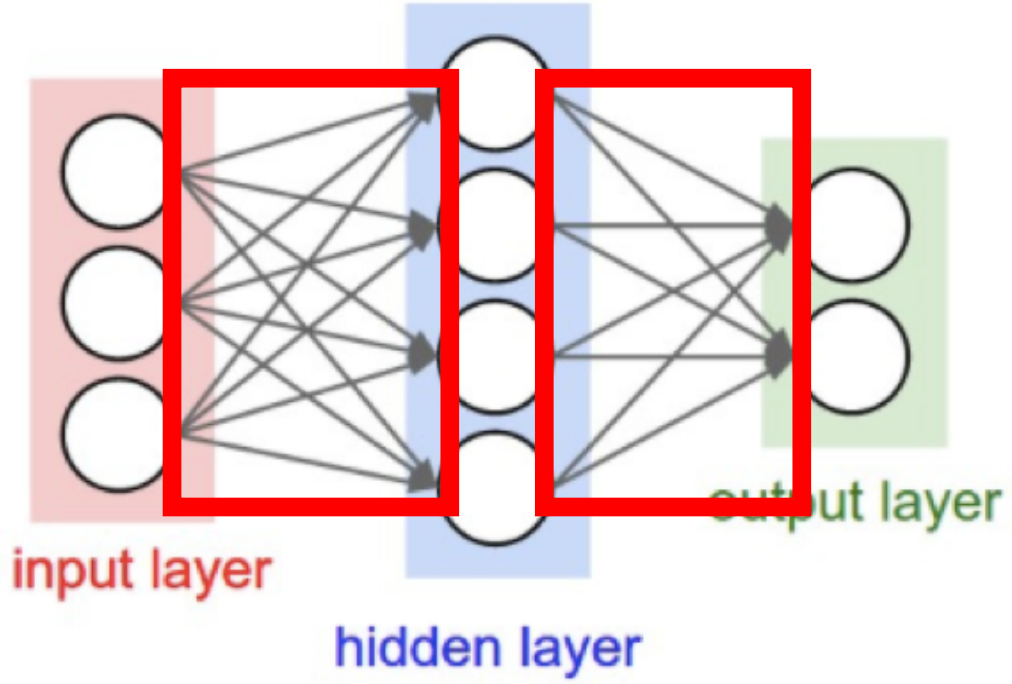


- **Training goal: learn model weights**

Python Machine Learning; Raschka & Mirjalili

http://cs231n.github.io/neural-networks-1/

Neural Network

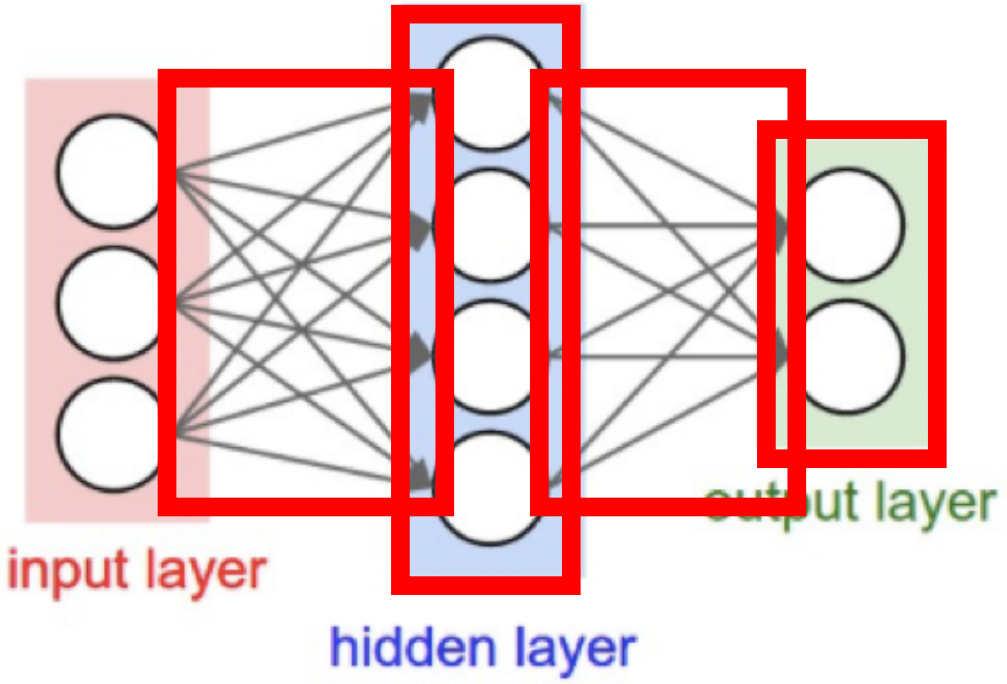


How many weights are in this model?

- Input to Hidden Layer:
  - 3x4 = 12
- Hidden Layer to Output Layer
  - 4x2 = 8
- Total:
  - 12 + 8 = 20

http://cs231n.github.io/neural-networks-1/

Neural Network

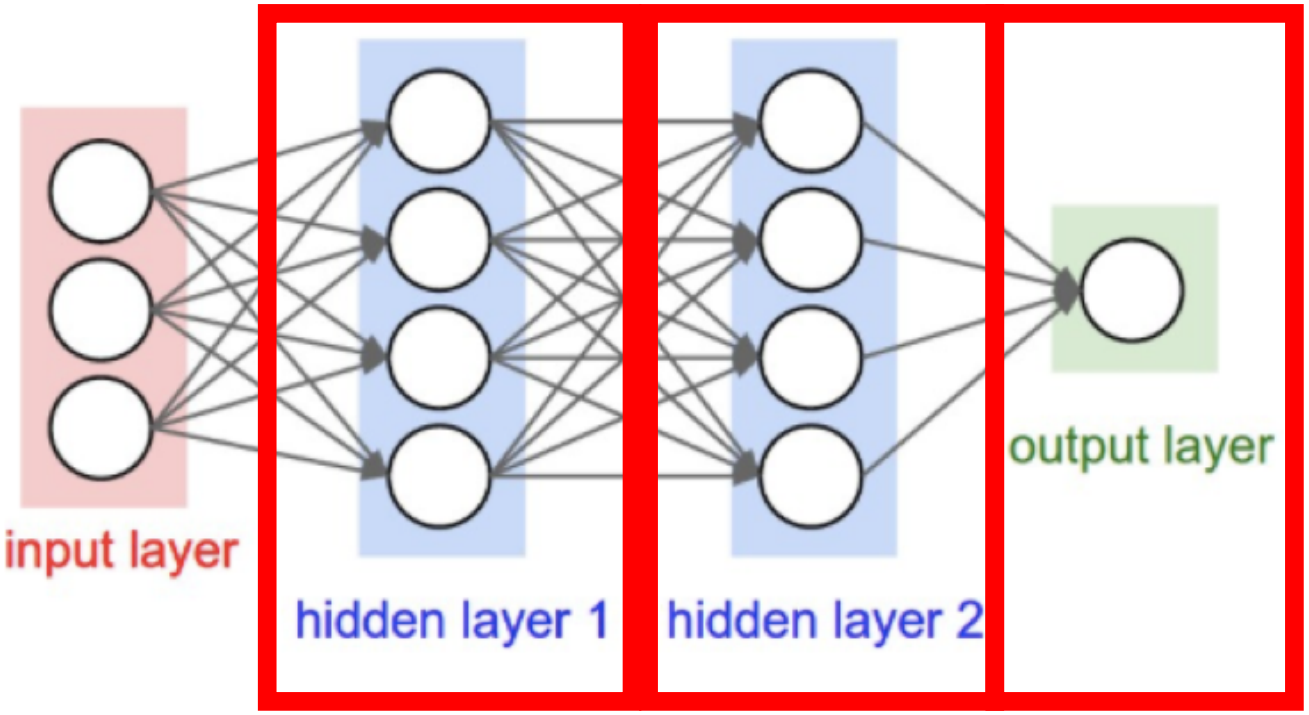


How many parameters are there to learn?

- Number of weights:
  - 20
- Number of biases:
  - 4 + 2 = 6
- Total:
  - 26

http://cs231n.github.io/neural-networks-1/

Neural Network

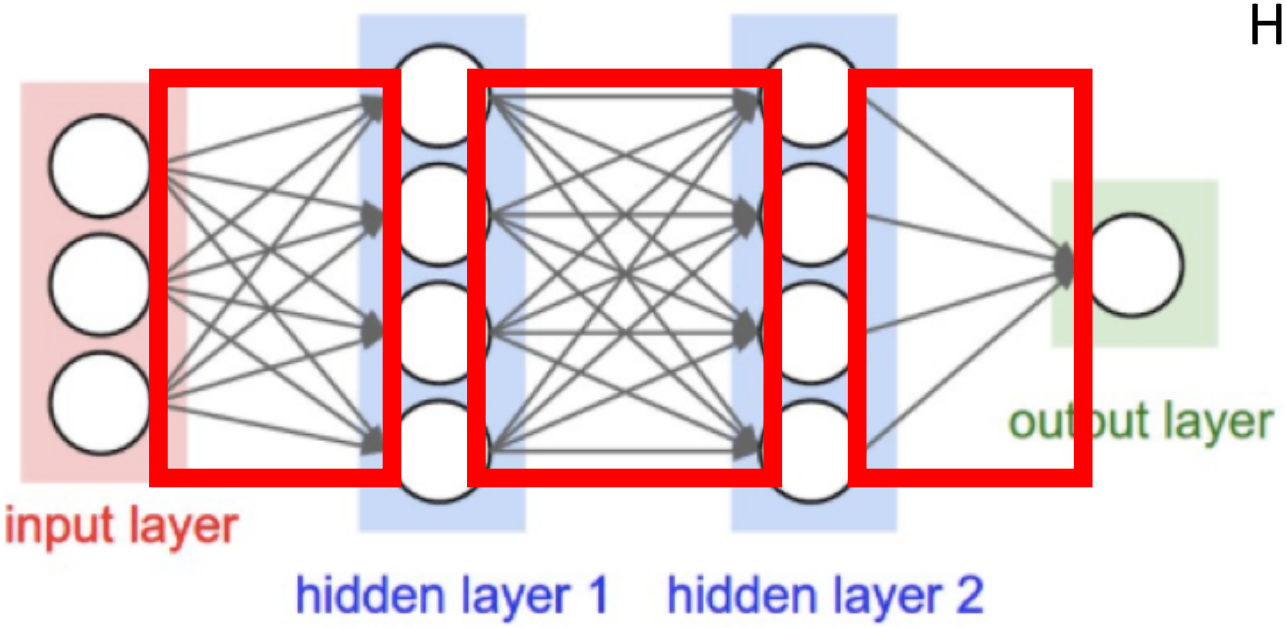


How many layers are in this network?

- 3 (number of hidden layers plus output layer; exclude input layer when counting)

http://cs231n.github.io/neural-networks-1/

Neural Network

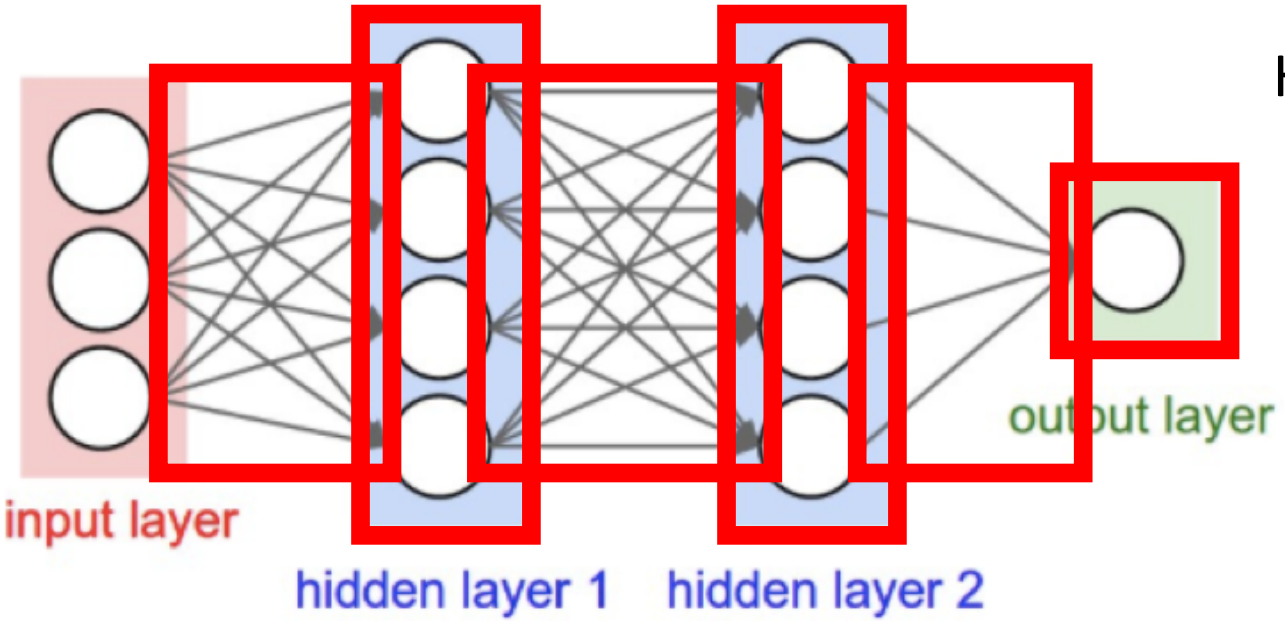


How many weights are in this model?

- Input to Hidden Layer 1:
  - 3x4 = 12
- Input to Hidden Layer 2:
  - 4x4 = 16
- Hidden Layer 2 to Output Layer
  - 4x1 = 4
- Total:
  - 12 + 16 + 4 = 32

http://cs231n.github.io/neural-networks-1/

Neural Network

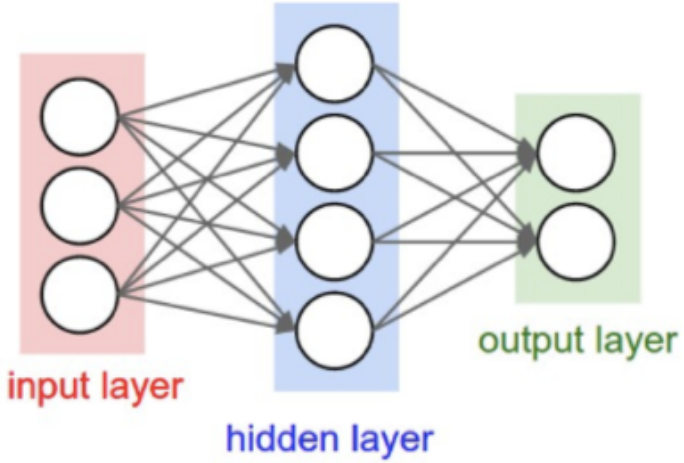


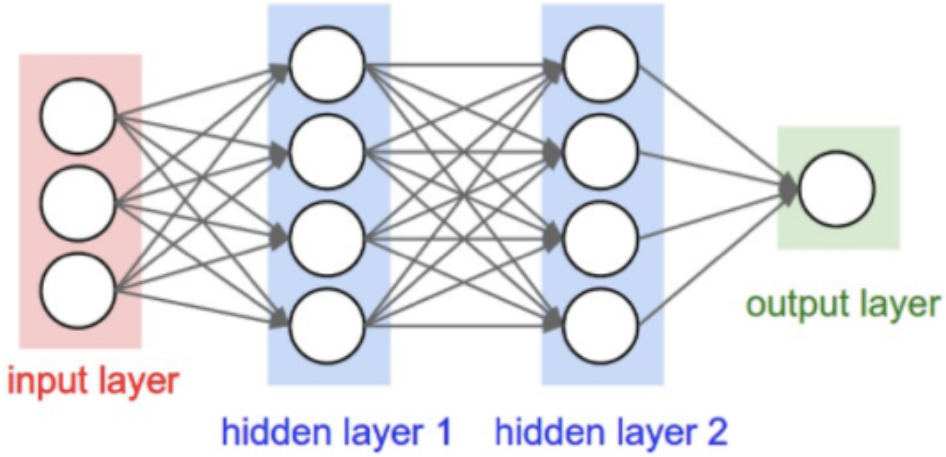
How many parameters are there to learn?

- Number of weights:
  - 32
- Number of biases:
  - 4 + 4 + 1 = 9
- Total
  - 41

http://cs231n.github.io/neural-networks-1/

Fully Connected, Feed Forward Neural Networks

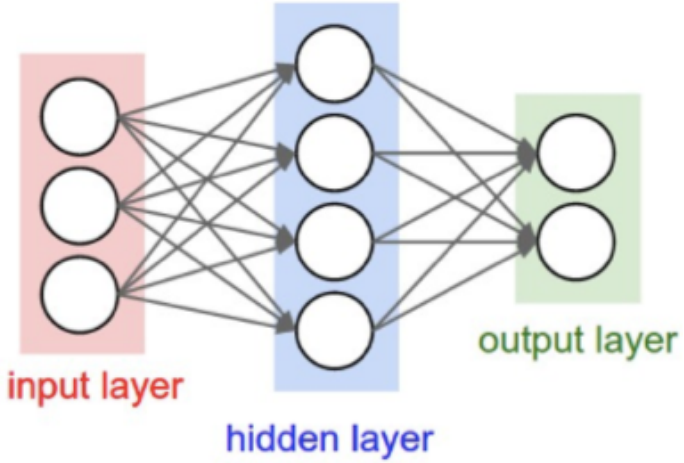


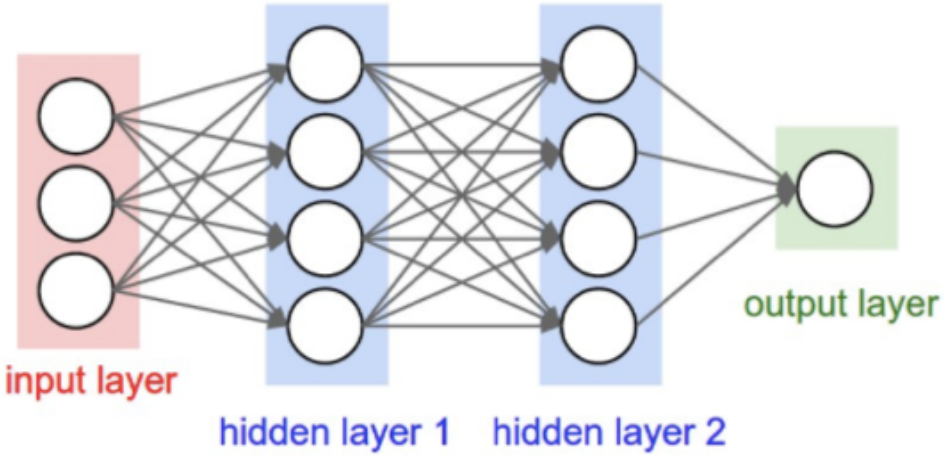


- What does it mean for a model to be fully connected?
  - Each unit provides input to each unit in the next layer
- What does it mean for a model to be feed forward?
  - Each layer serves as input to the next layer with no loops

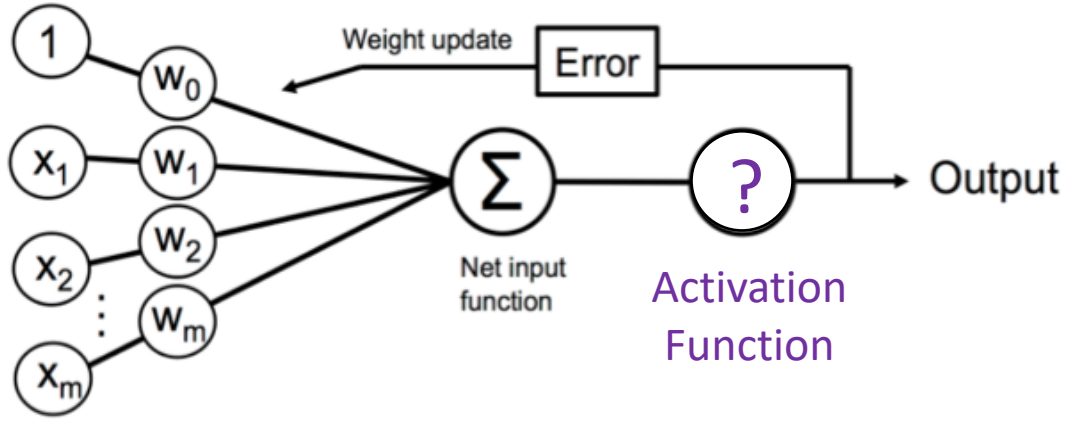
http://cs231n.github.io/neural-networks-1/

Neural Networks: Activation Functions





Each unit takes as input a weighted sum and applies a non-linear (activation) function

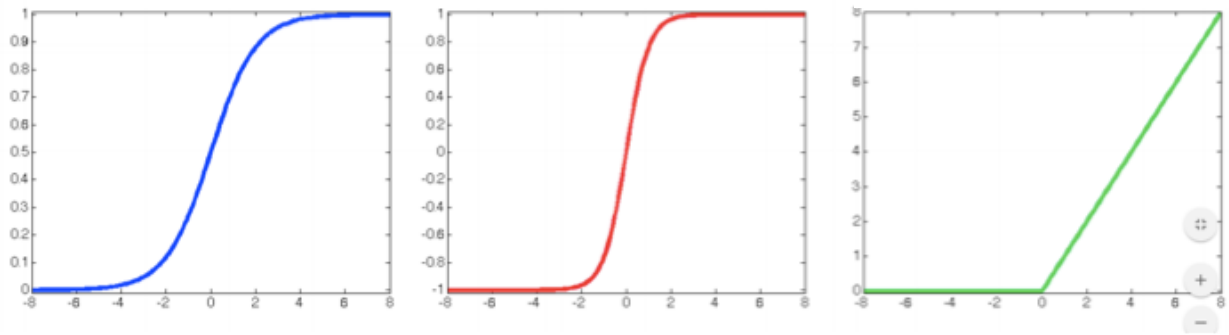


Python Machine Learning; Raschka & Mirjalili

http://cs231n.github.io/neural-networks-1/

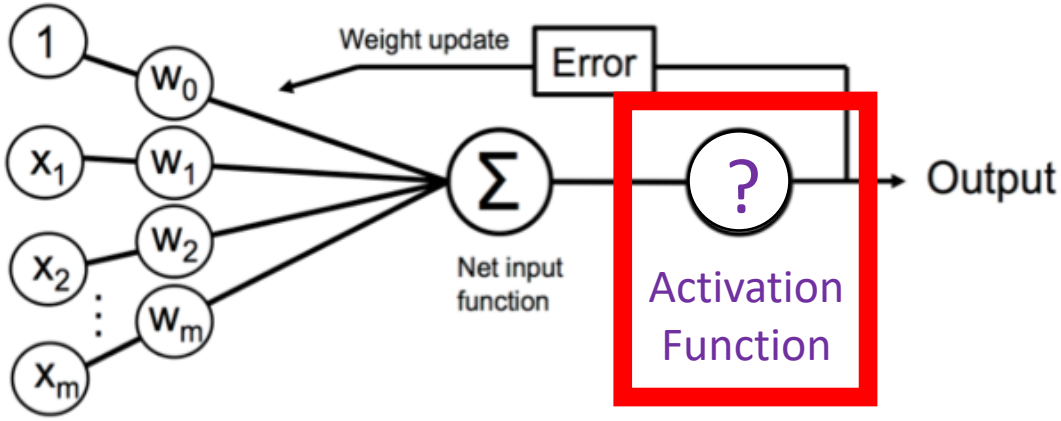
Neural Networks: Activation Functions

Sigmoid Tanh ReLU



name	function
Sigmoid	$\sigma(z) = \frac{1}{1+\exp(-z)}$
Tanh	$\tanh(z) = \frac{\exp(z)-\exp(-z)}{\exp(z)+\exp(-z)}$
ReLU	$\mathrm{ReLU}(z) = \max(0, z)$

Each unit takes as input a weighted sum and applies a non-linear (activation) function

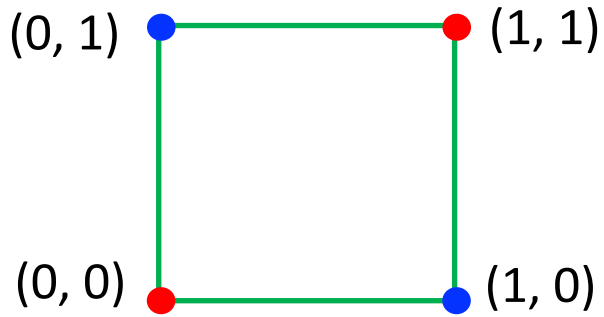


Python Machine Learning; Raschka & Mirjalili

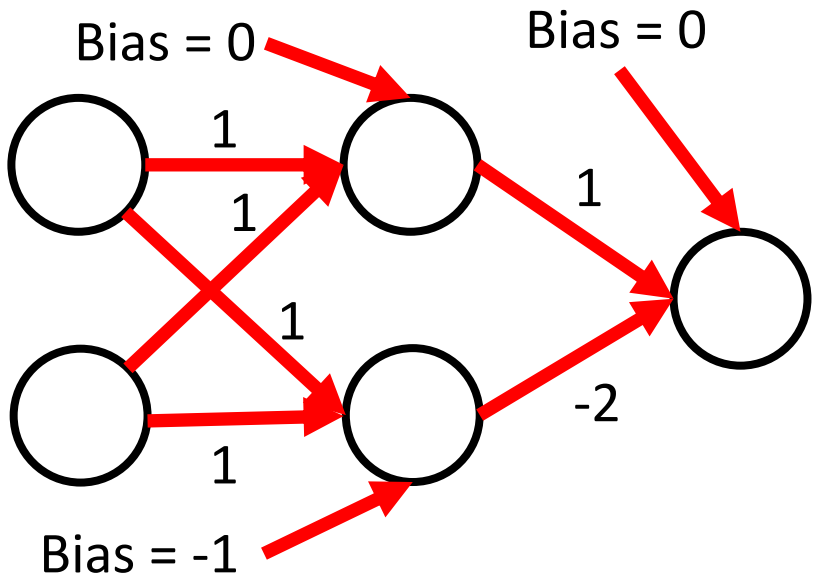
http://www.cs.utoronto.ca/~fidler/teaching/2015/slides/CSC411/10_nn1.pdf

Revisiting XOR problem

- Non-linear function: separate 1s from 0s:



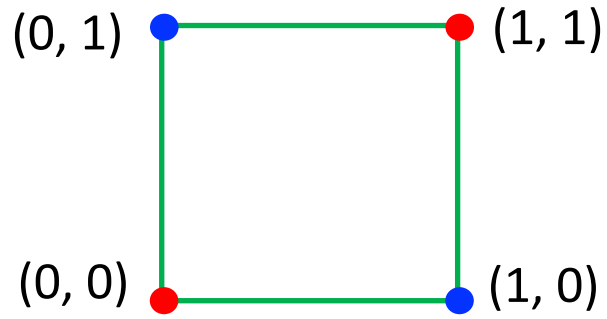
- Approach: Use ReLU activation function ($\mathrm{ReLU}(z) = \max(0, z)$) with this model:



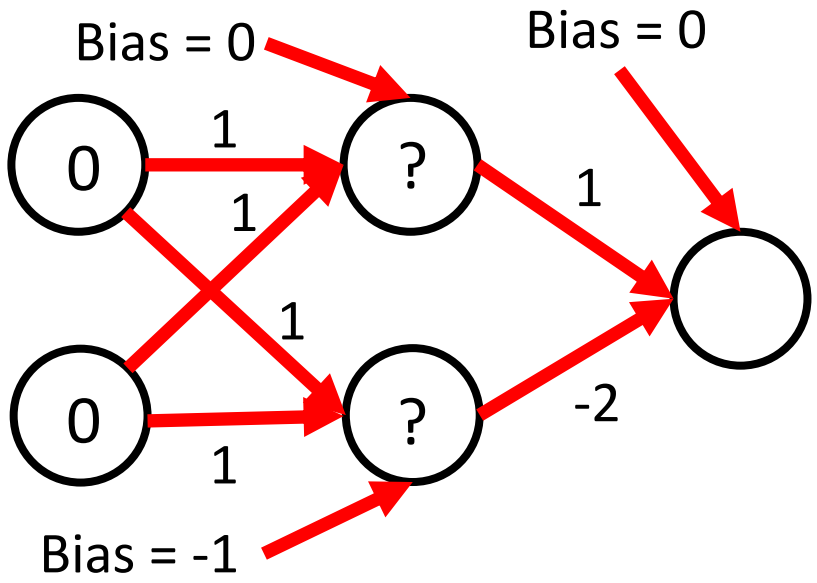
INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Revisiting XOR problem

- Non-linear function: separate 1s from 0s:



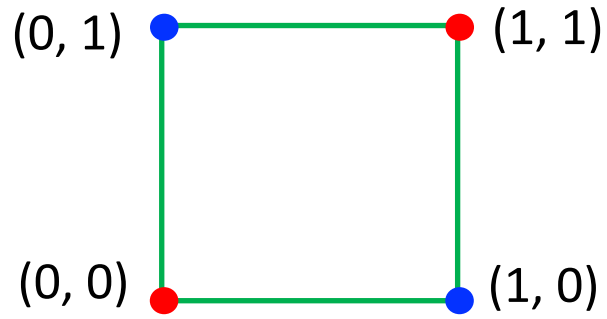
- Approach: Use ReLU activation function ($\mathrm{ReLU}(z) = \max(0, z)$) with this model:



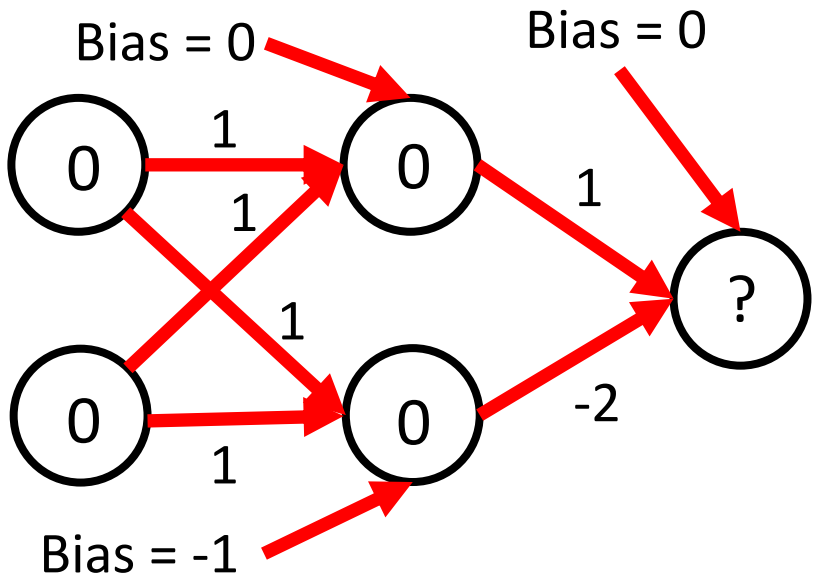
INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Revisiting XOR problem

- Non-linear function: separate 1s from 0s:



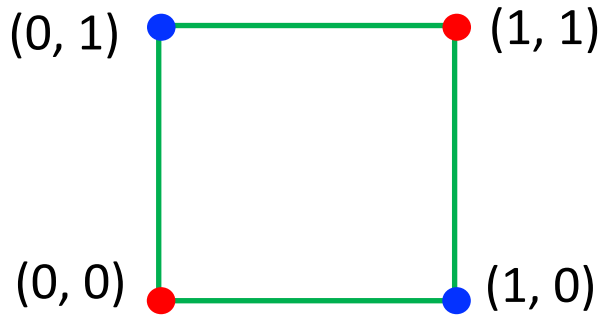
- Approach: Use ReLU activation function ($\mathrm{ReLU}(z) = \max(0, z)$) with this model:



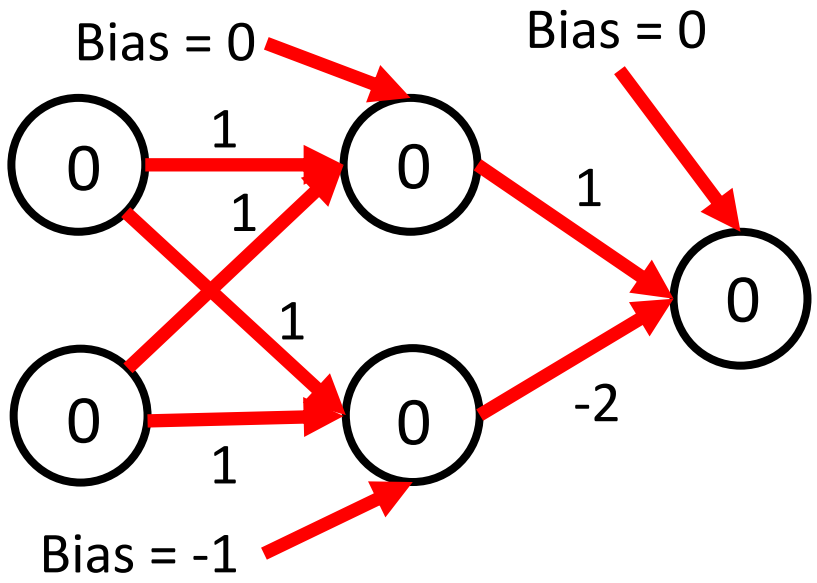
INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Revisiting XOR problem

- Non-linear function: separate 1s from 0s:



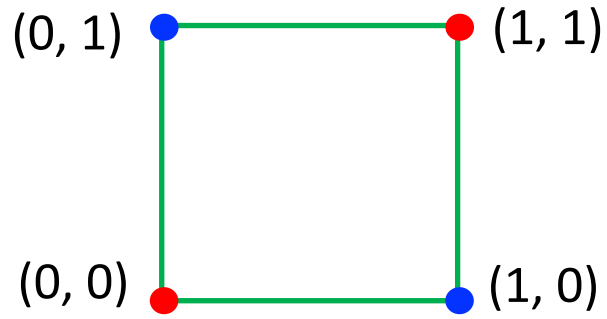
- Approach: Use ReLU activation function ($\mathrm{ReLU}(z) = \max(0, z)$) with this model:



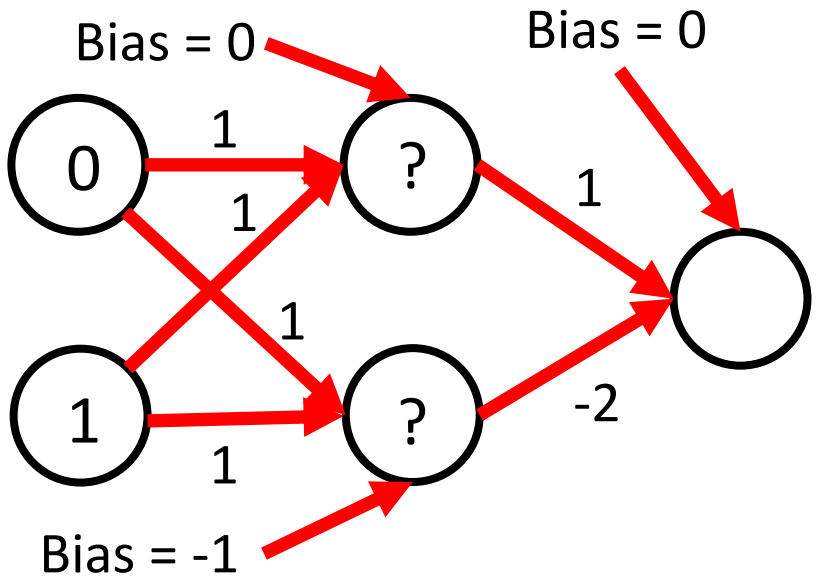
INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Revisiting XOR problem

- Non-linear function: separate 1s from 0s:



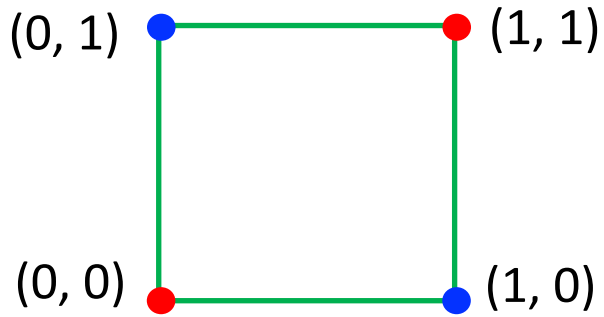
- Approach: Use ReLU activation function ($\mathrm{ReLU}(z) = \max(0, z)$) with this model:



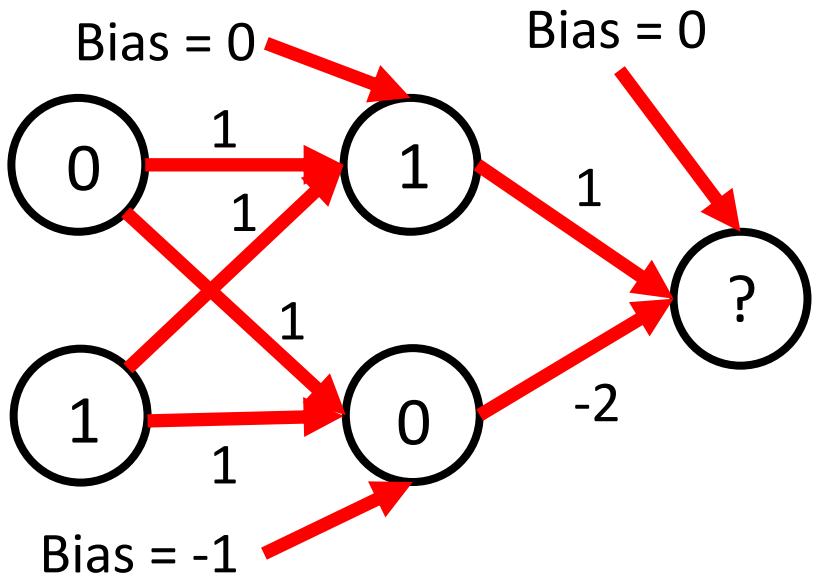
INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Revisiting XOR problem

- Non-linear function: separate 1s from 0s:



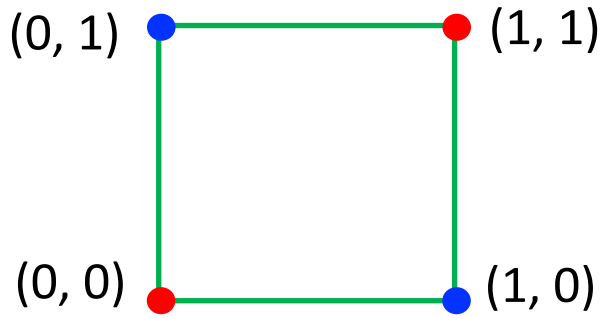
- Approach: Use ReLU activation function ($\mathrm{ReLU}(z) = \max(0, z)$) with this model:



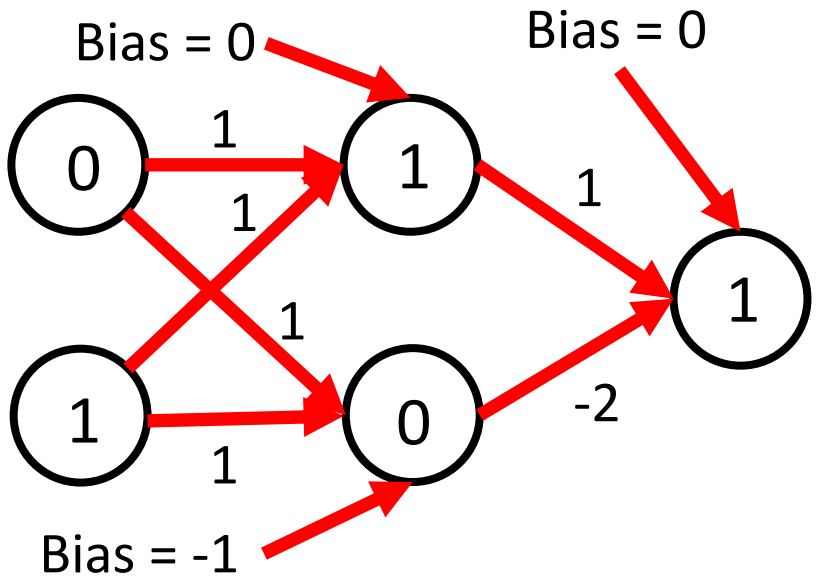
INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Revisiting XOR problem

- Non-linear function: separate 1s from 0s:



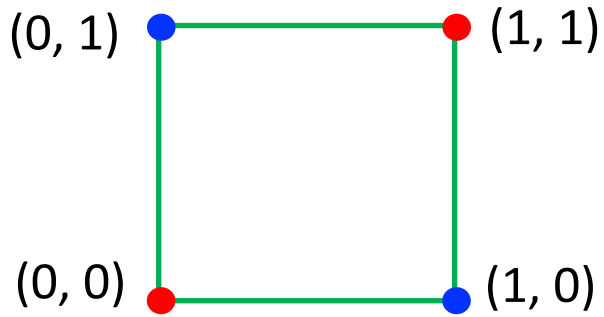
- Approach: Use ReLU activation function ($\mathrm{ReLU}(z) = \max(0, z)$) with this model:



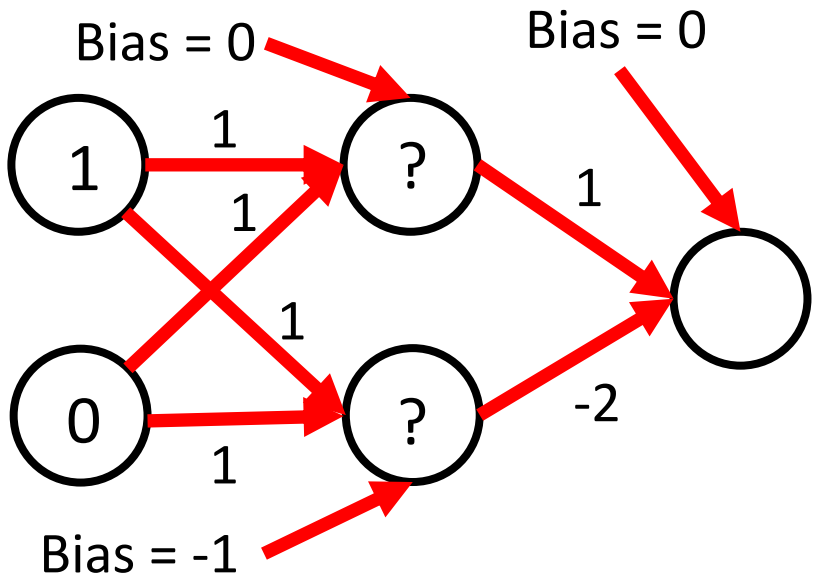
INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Revisiting XOR problem

- Non-linear function: separate 1s from 0s:



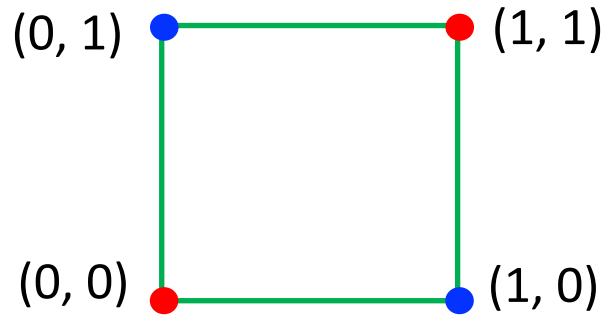
- Approach: Use ReLU activation function ($\mathrm{ReLU}(z) = \max(0, z)$) with this model:



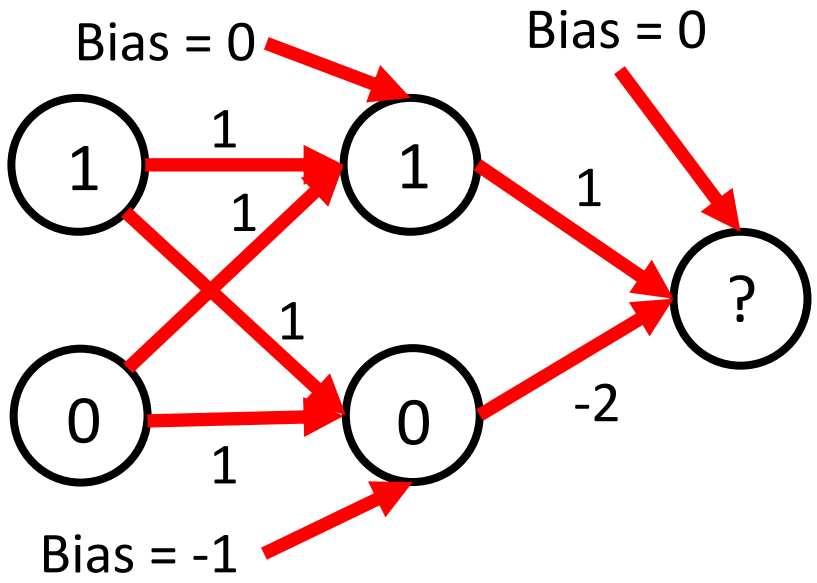
INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Revisiting XOR problem

- Non-linear function: separate 1s from 0s:



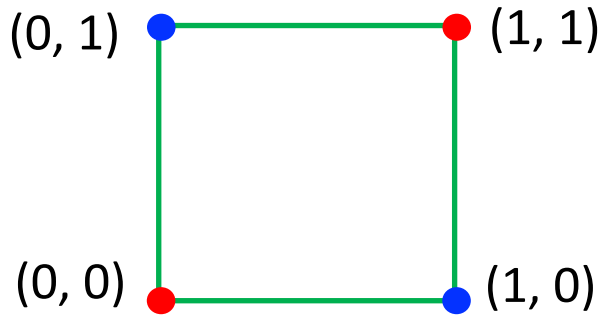
- Approach: Use ReLU activation function ($\mathrm{ReLU}(z) = \max(0, z)$) with this model:



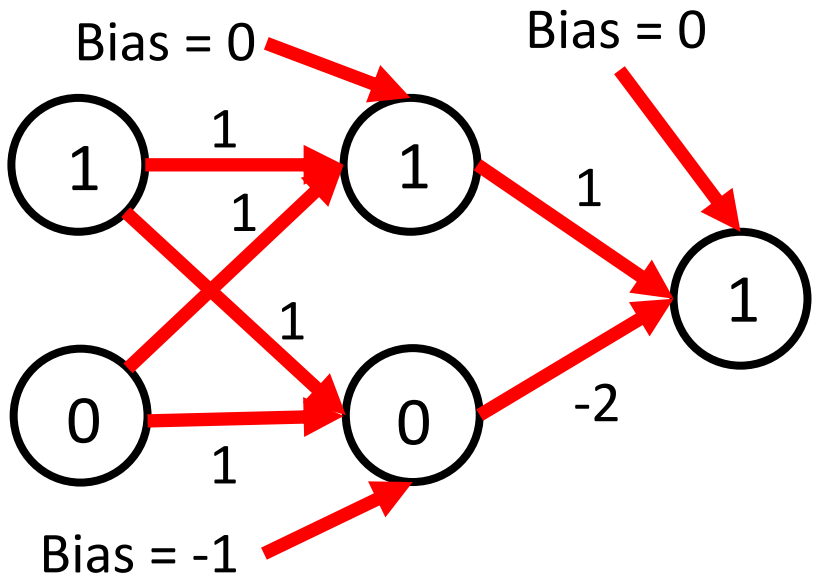
INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Revisiting XOR problem

- Non-linear function: separate 1s from 0s:



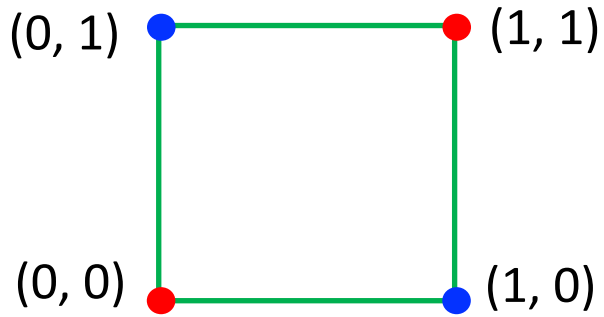
- Approach: Use ReLU activation function ($\mathrm{ReLU}(z) = \max(0, z)$) with this model:



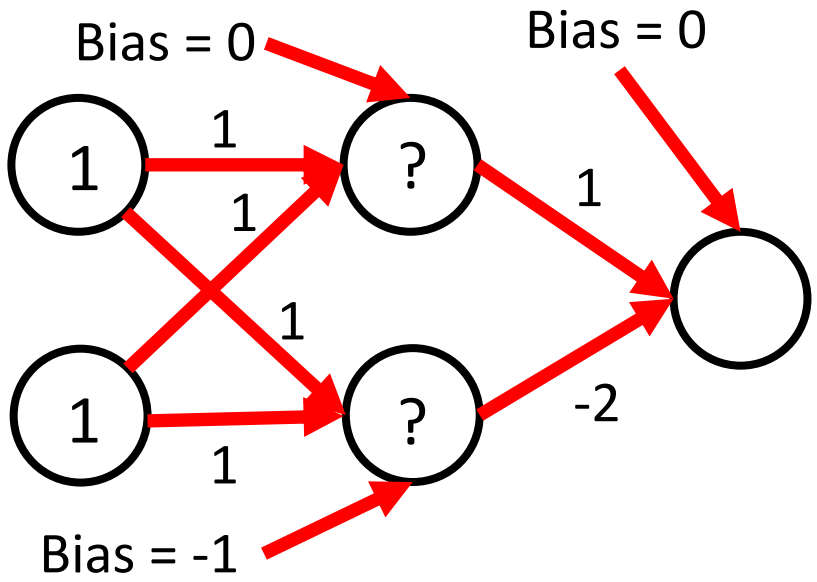
INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Revisiting XOR problem

- Non-linear function: separate 1s from 0s:



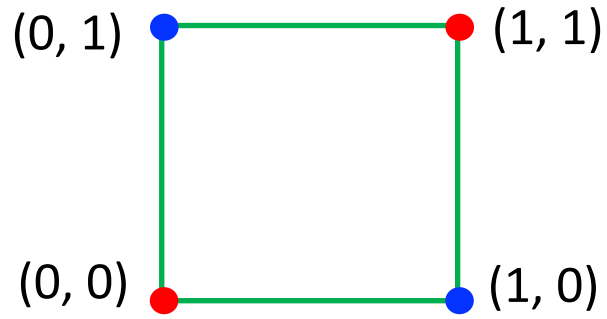
- Approach: Use ReLU activation function ($\mathrm{ReLU}(z) = \max(0, z)$) with this model:



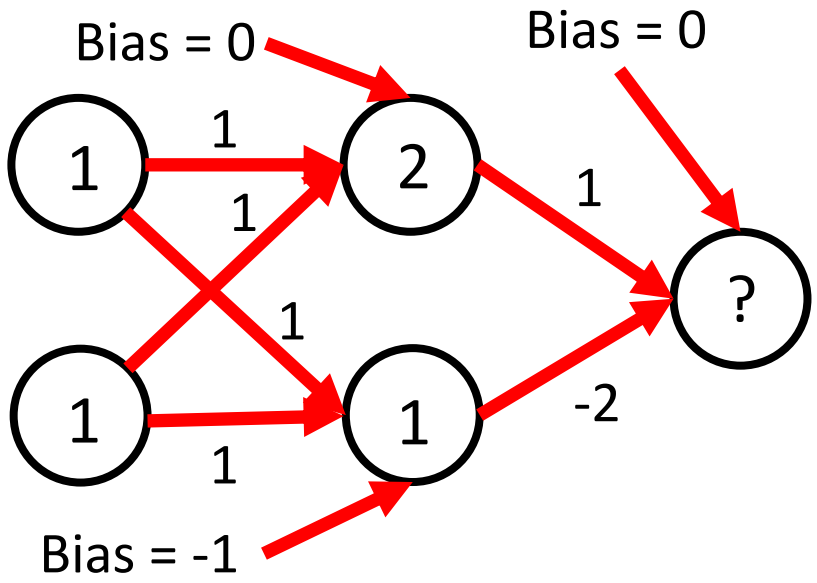
INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Revisiting XOR problem

- Non-linear function: separate 1s from 0s:



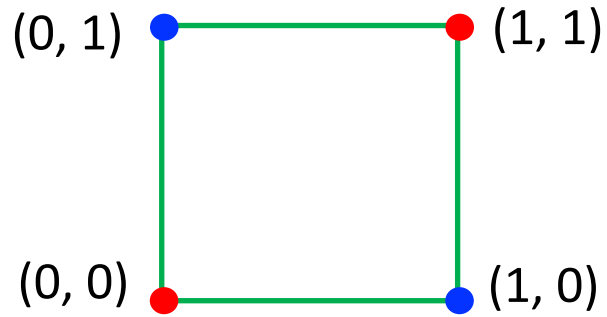
- Approach: Use ReLU activation function ($\mathrm{ReLU}(z) = \max(0, z)$) with this model:



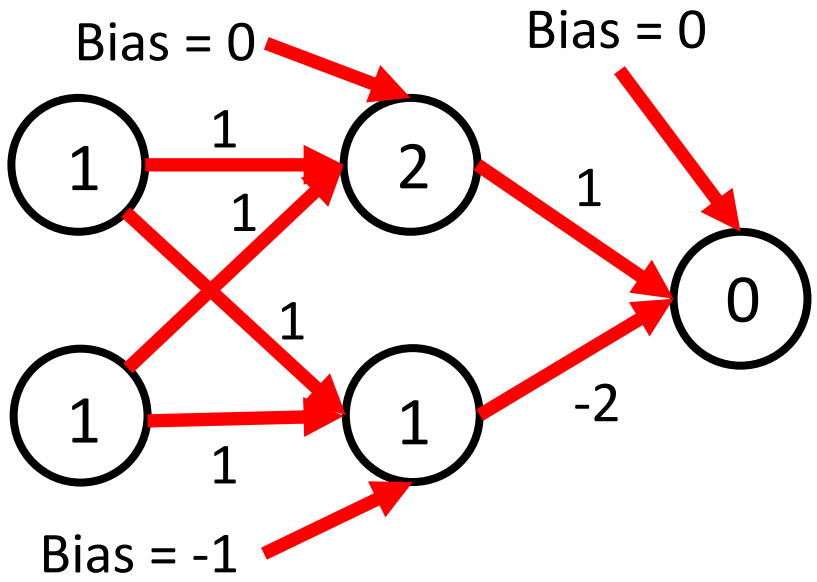
INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Revisiting XOR problem

- Non-linear function: separate 1s from 0s:



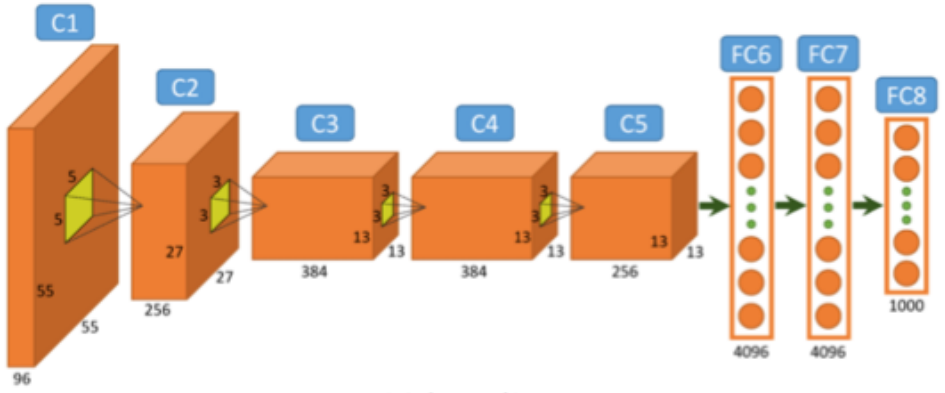
- Approach: Use ReLU activation function ($\mathrm{ReLU}(z) = \max(0, z)$) with this model:



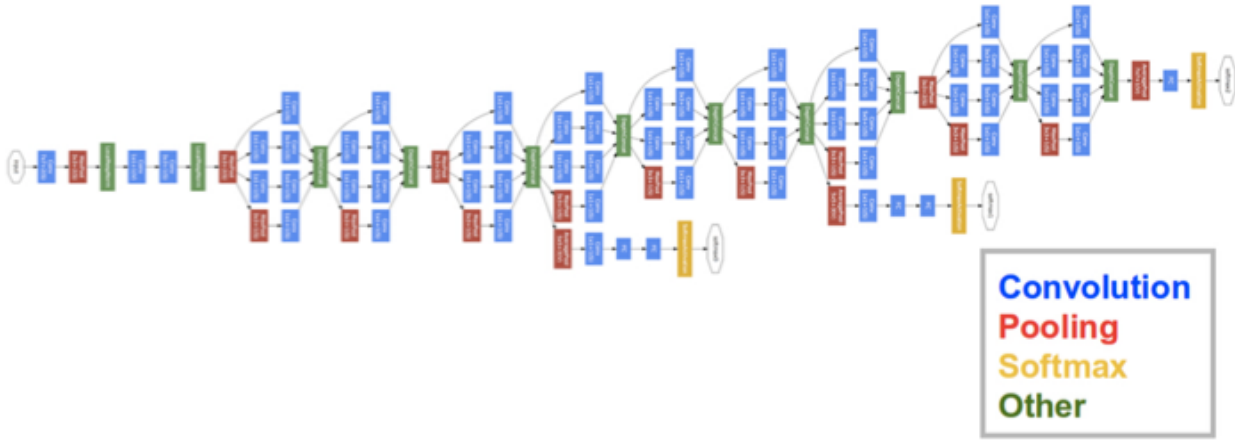
INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Modern Trend: “Deeper” Neural Networks Are Modeling Even More Complex Problems

AlexNet (2012)



GoogLeNet (2014)



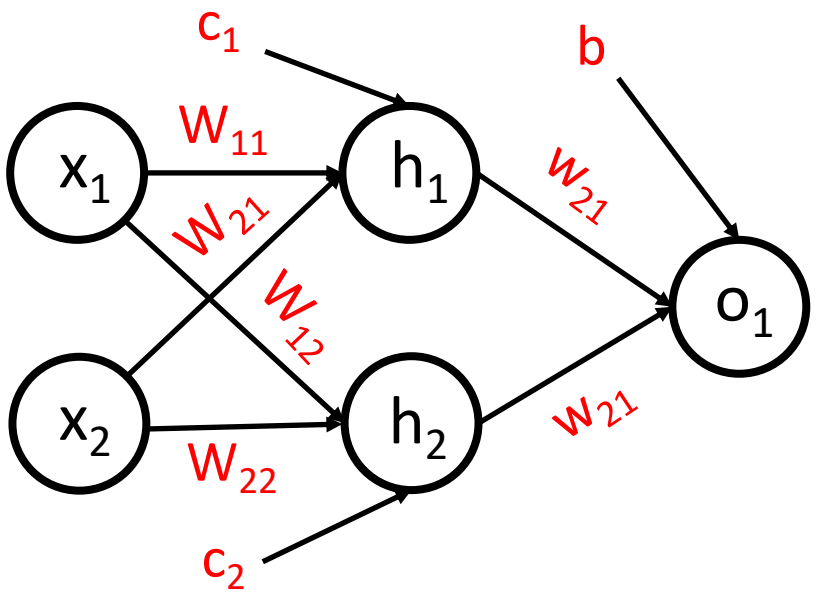
Hundreds of layers, millions of neurons, and millions of model parameters to learn… more on this in future classes!

Today's Topics

- Revisiting History of Modern Neural Networks and "Deep Learning"
- Neural Network Architecture
- Training a Neural Network
- Lab

Neural Network: What to Learn?

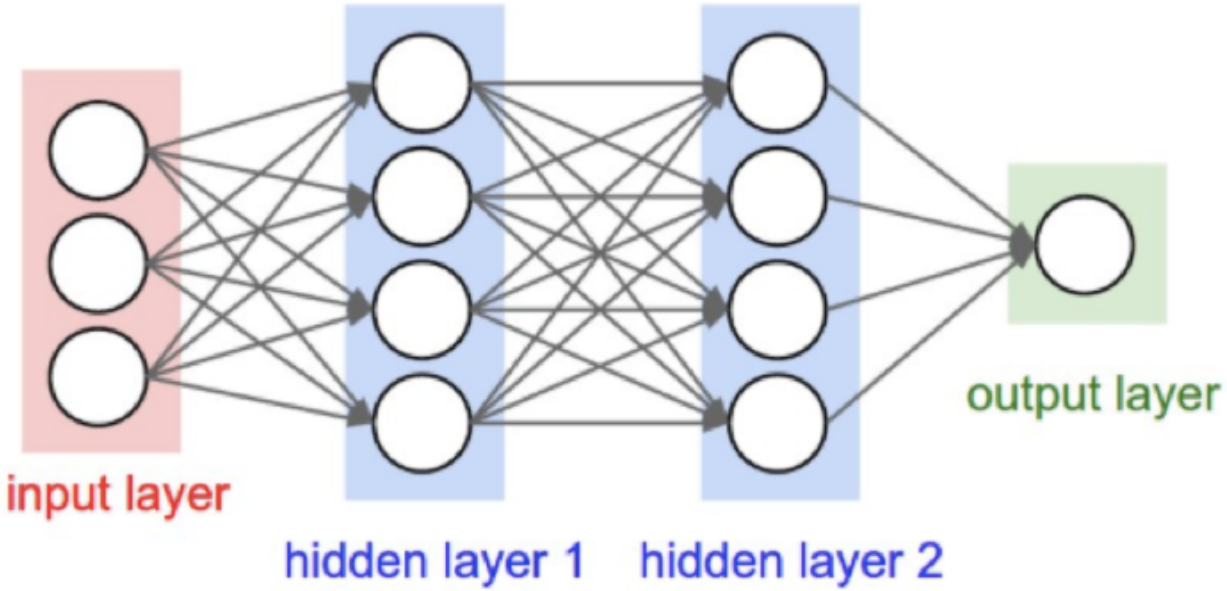
- Learn:
  - weights connecting units
  - bias for each unit
- e.g., 2 layer neural network:



- What does a low weight mean?
  - A connection does not contribute much, if at all, to the network
- What does a high weight mean?
  - A connection contributes much to the network

Neural Network: What to Learn?

- Algorithm decides how to use each layer to produce the output
- For this reason, layers are called “hidden”… training data does not decide the output for each layer



http://cs231n.github.io/neural-networks-1/

Neural Network: Learn Model Parameters

- Repeat until stopping criterion met:
  1. **Forward pass**: propagate training data through network to make prediction
  2. **Backward pass**: using predicted output, calculate gradients backward
  3. Update each weight using calculated gradients

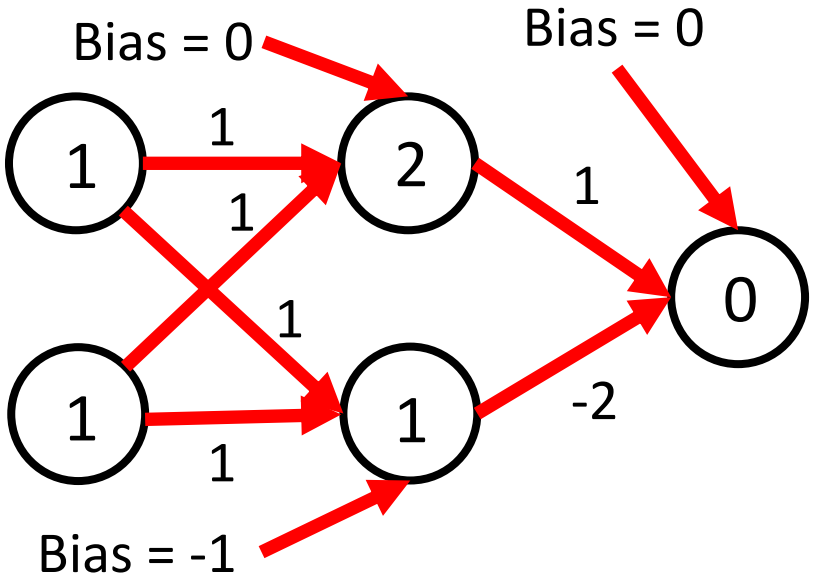
Neural Network: Stopping criteria

- Repeat until stopping criterion met:
  1. **Forward pass**: propagate training data through network to make prediction
  2. **Backward pass**: using predicted output, calculate gradients backward
  3. Update each weight using calculated gradients

- What stopping criterion to use when training?
  - Weight changes are incredibly small
  - Percentage of misclassified example is below some threshold
  - Finished a pre-specified number of epochs
  - …

Neural Network: Forward Pass

- Repeat until stopping criterion met:
  1. **Forward pass**: propagate training data through network to make prediction
  2. **Backward pass**: using predicted output, calculate gradients backward
  3. Update each weight using calculated gradients
- Plug input values into network and feed it forward;
  - e.g.,



INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Neural Network: Learn Model Parameters

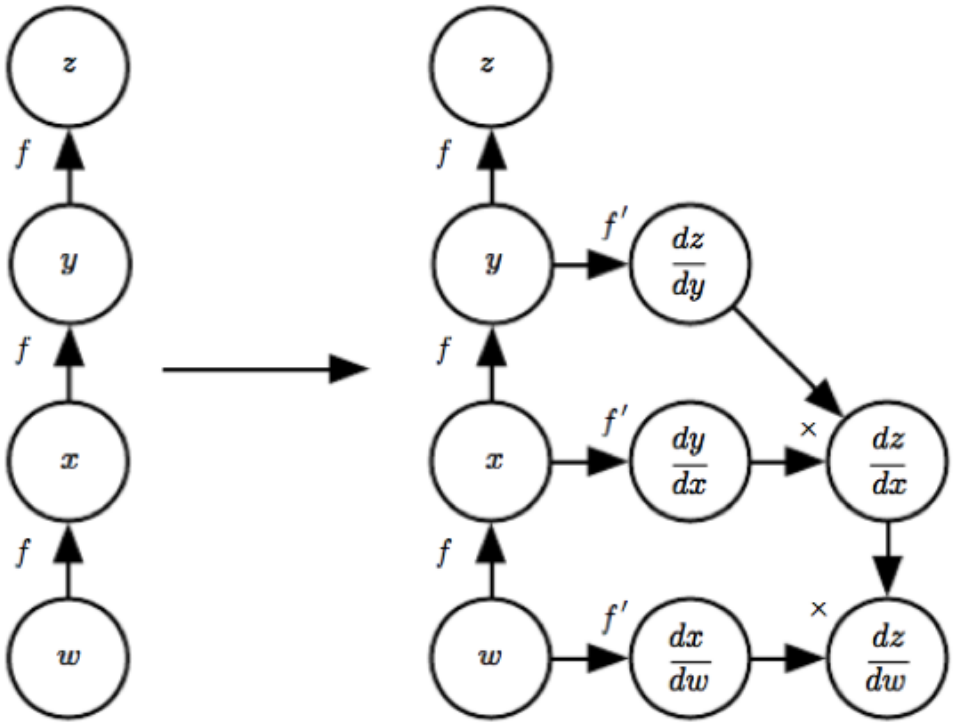
- Repeat until stopping criterion met:
  1. **Forward pass**: propagate training data through network to make prediction
  2. **Backward pass**: using predicted output, calculate gradients backward
  3. Update each weight using calculated gradients

- Key challenge: How to compute gradient for a multilayer network?
  - Automatic differentiation: field of study on computing derivatives algorithmically
  - **Backpropagation**: this technique, which originated in the automatic differentiation field, is widely used today; it is a special case of a broader class of techniques called “reverse mode accumulation”

History: http://people.idsia.ch/~juergen/who-invented-backpropagation.html

Neural Network Training: Backpropagation

- Idea: chain: $x = f(w), y = f(x), z = f(y)$



$$\begin{aligned}
&\frac{\partial z}{\partial w} \\
=&\frac{\partial z}{\partial y}\frac{\partial y}{\partial x}\frac{\partial x}{\partial w} \\
=&f'(y)f'(x)f'(w) \\
=&f'(f(f(w)))f'(f(w))f'(w).
\end{aligned}$$

- More to follow on this next week…

Ian Goodfellow, Yoshua Bengio, and Aaron Courville; Deep Learning, 2016.

Neural Network Training: Backpropagation

- But briefly…
- 1986: Successful results using backpropagation to train neural networks instigated next wave of excitement about neural networks
  - Challenge overcome was how to train a non-convex/non-smooth error surface with respect to parameters (single layer networks like Adaline have smooth error surfaces)
- Backpropagation: computes the chain rule, with a specific order of operations that is highly efficient
  - Goes through each layer in reverse to measure the error contribution from each connection (reverse pass)
- Required Calculus Background:
  - Chain rule: computes derivative of a complex, nested function, such as f(g(x))
  - Can extend chain rule for arbitrarily long function compositions
  - https://sebastianraschka.com/pdf/books/dlb/appendix_d_calculus.pdf

D. Rulhart, G. Hinton, and R. Williams, Learning Internal Representations by Error Propagation, 1986.

Neural Network: Learn Model Parameters

- Repeat until stopping criterion met:
  1. **Forward pass**: propagate training data through network to make prediction
  2. **Backward pass**: using predicted output, calculate gradients backward
  3. Update each weight using calculated gradients

- After computing the gradients, we can update the weights by taking an opposite step towards the gradient for each layer

$$W^{(l)} = W^{(l)} - \eta \Delta^{(l)}$$

Rule of thumb: set learning rate to 1/t where t is number of iterations through the training set so far

Implementation Details

- Data should have mean of zero and unit variance (i.e., standardize input)
- Weights should be initialized randomly to avoid any symmetries that the Gradient Descent algorithm would be unable to break
  - e.g., if all weights are set to 0, then all units will output 0 and the error gradient will be the same for all neurons in a given hidden layer; consequently gradient descent would update all weights in the same way in each layer;

Today's Topics

- Revisiting History of Modern Neural Networks and "Deep Learning"
- Neural Network Architecture
- Training a Neural Network
- Lab