Neural Networks

Data Mining & Analytics

Prof. Zach Pardos

INFO254/154: Spring ’19

Neural Networks: Terminology

edge, weight, coefficient

node, layer, activation function, loss

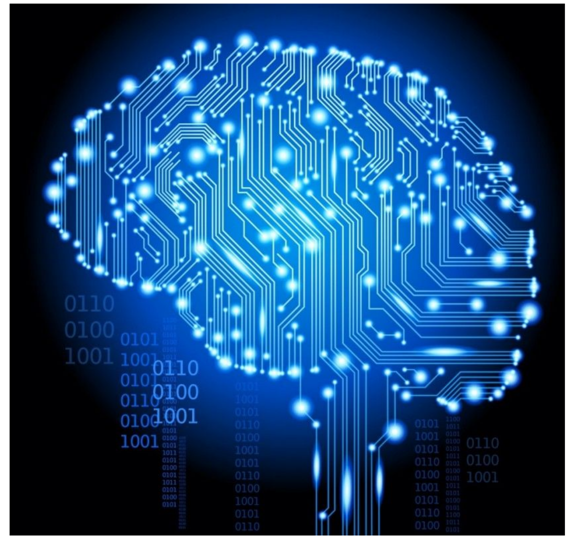
Neural Networks: Terminology

edge, <u>weight</u>, coefficient

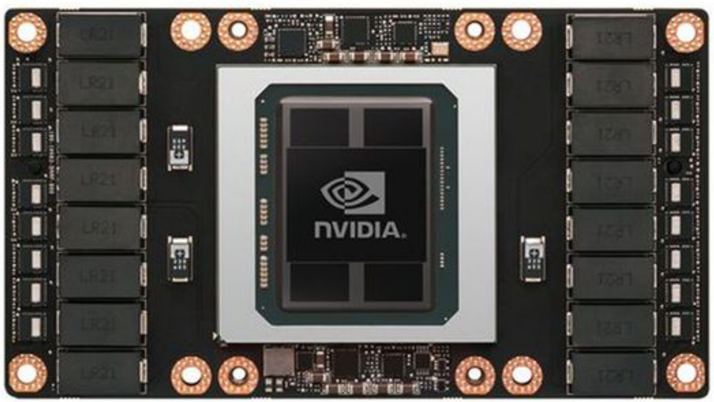
node, layer, activation function, loss

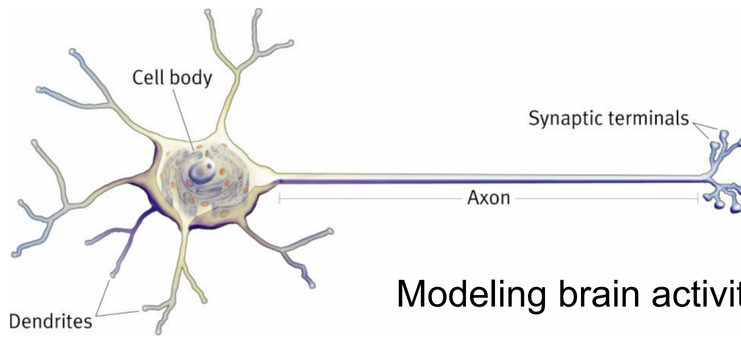
Neural Networks: Abstract levels

Model of the mind



Hardware / Software optimization

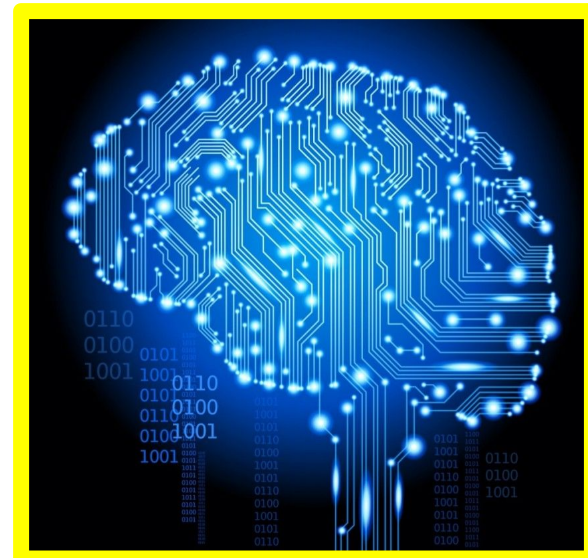




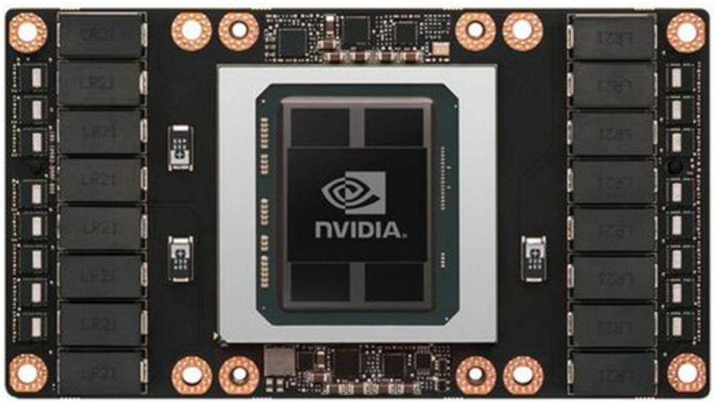
Modeling brain activity

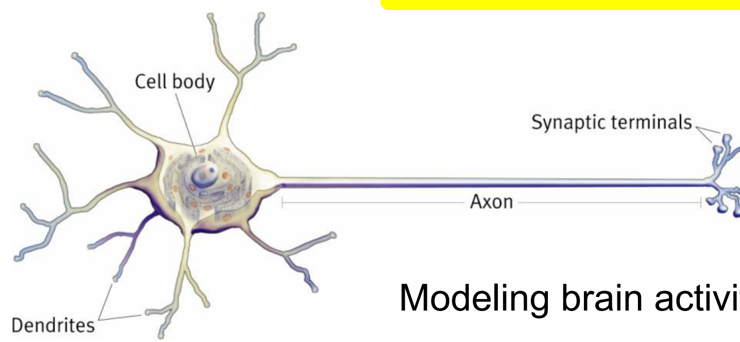
Neural Networks: Abstract levels

Model of the mind



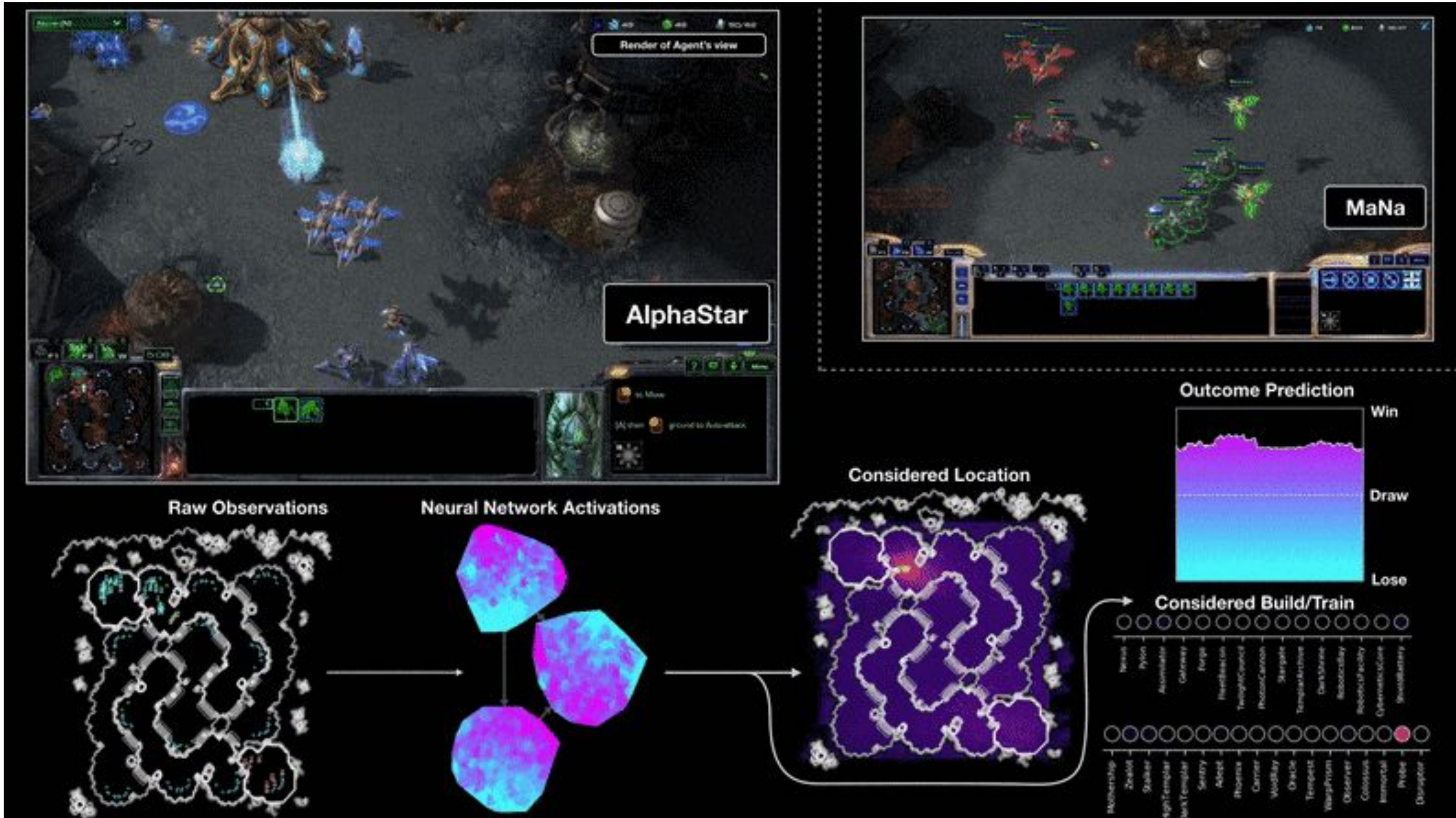
Hardware / Software optimization





Modeling brain activity

Contemporary applications



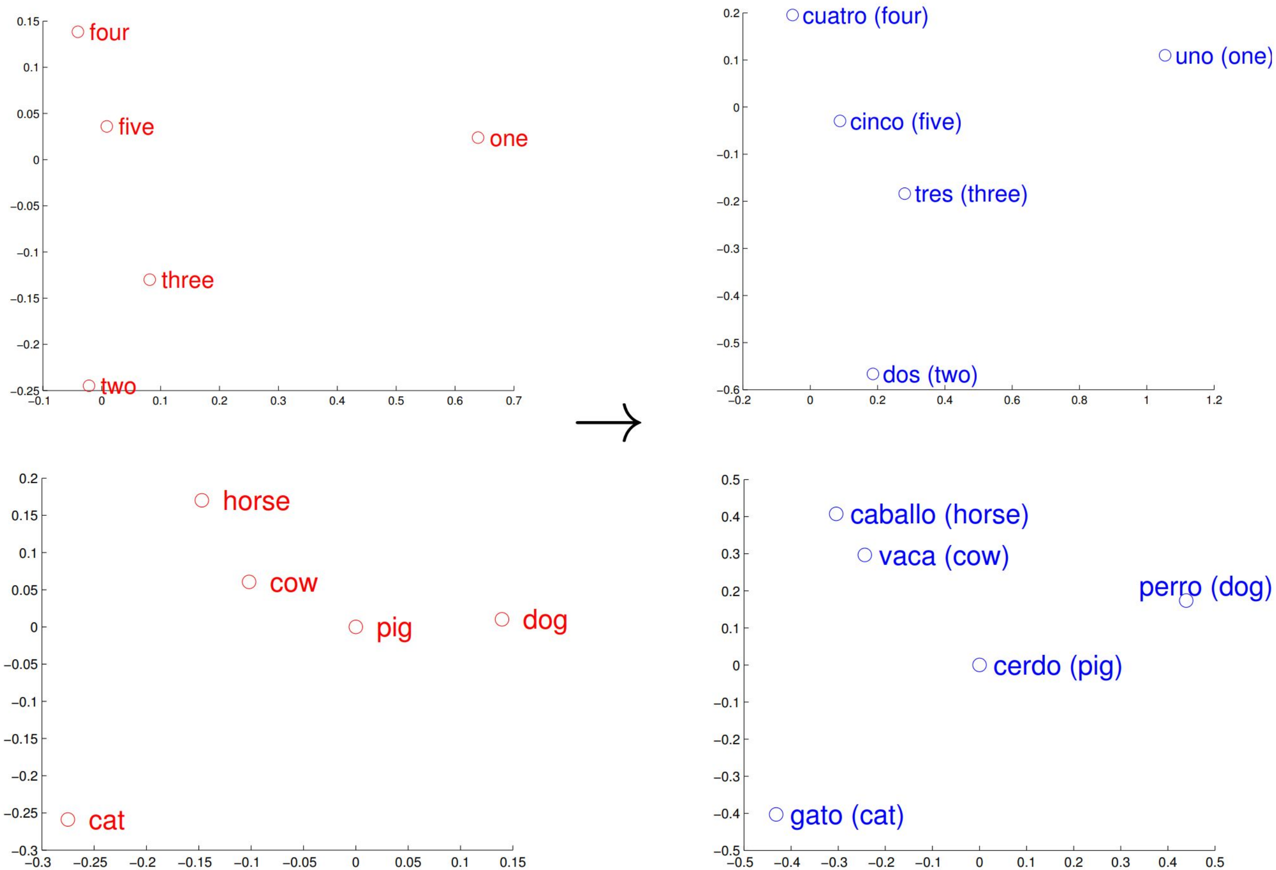
Long Short-Term Memory (a type of neural network) + Reinforcement Learning:
https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/

Contemporary applications

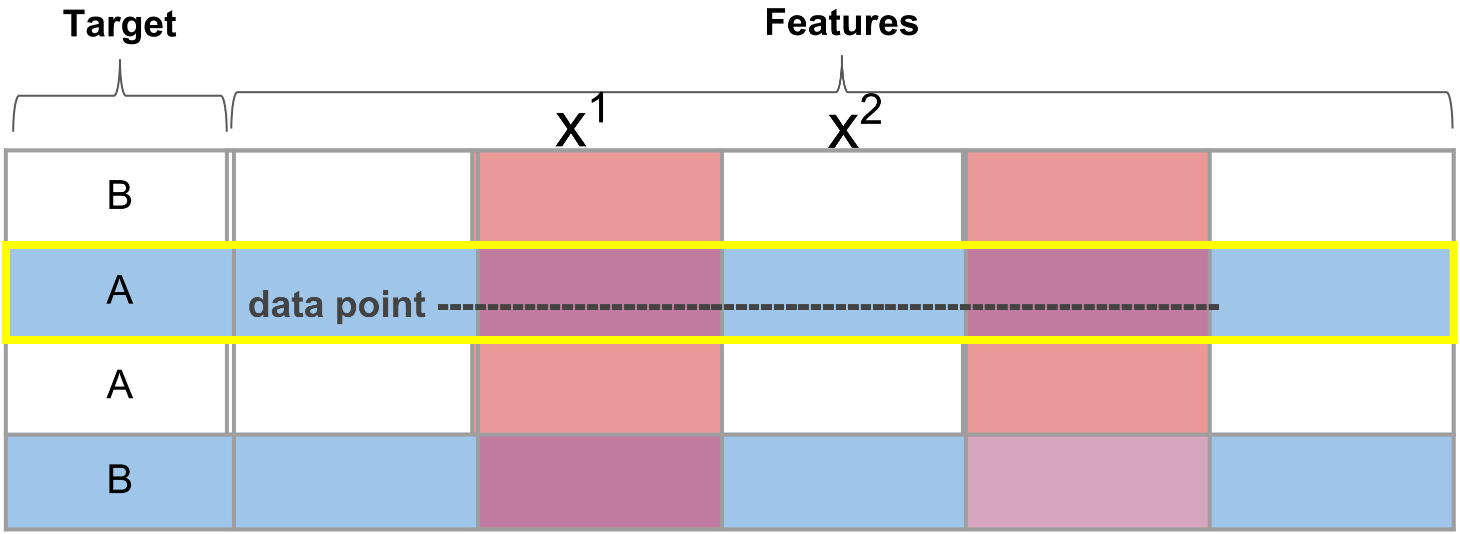
 

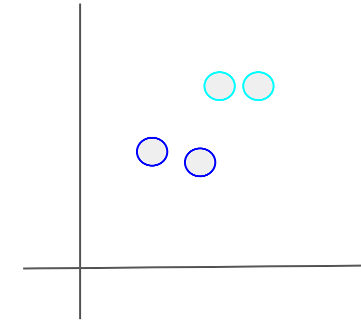
Generative Adversarial Networks https://thispersondoesnotexist.com/

Contemporary applications

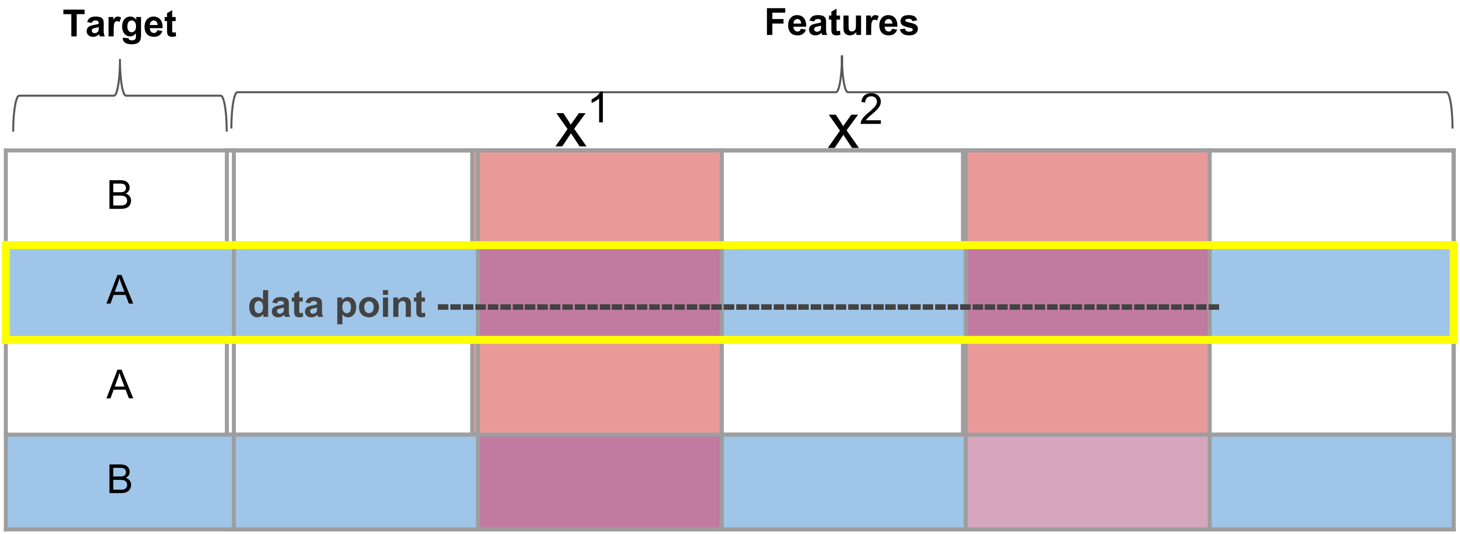


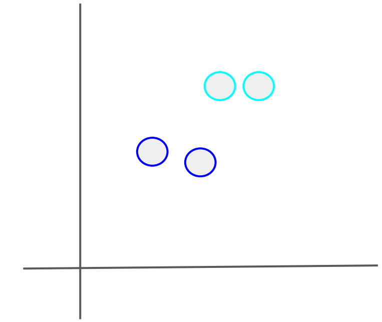
Machine Translation of words (word2vec): https://arxiv.org/abs/1309.4168
Phrase Translation (sequence2sequence):
https://blog.keras.io/a-ten-minute-introduction-to-sequence-to-sequence-learning-in-keras.html





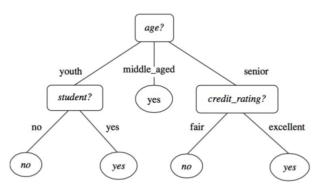
Classification: $X \rightarrow Y$ (the target)



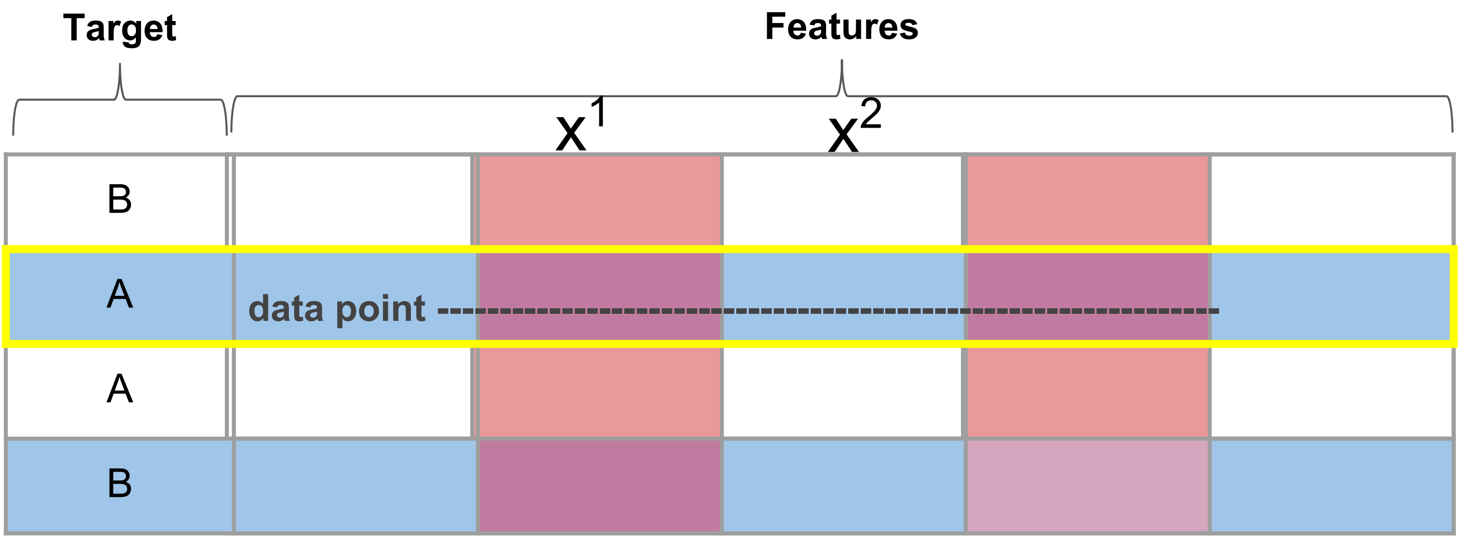


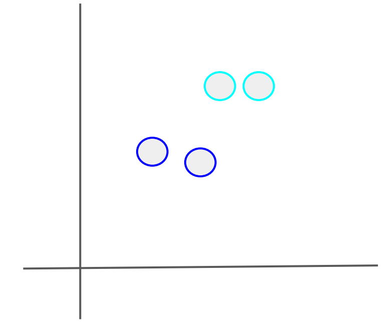
Classification: $X \rightarrow Y$ (the target)

Decision Tree



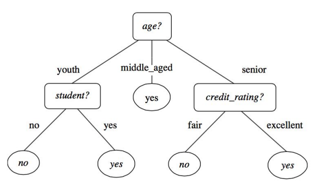
Learns a series of if-then-else rules to apply to X to get Y





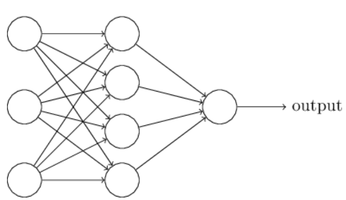
Classification: $X \rightarrow Y$ (the target)

Decision Tree

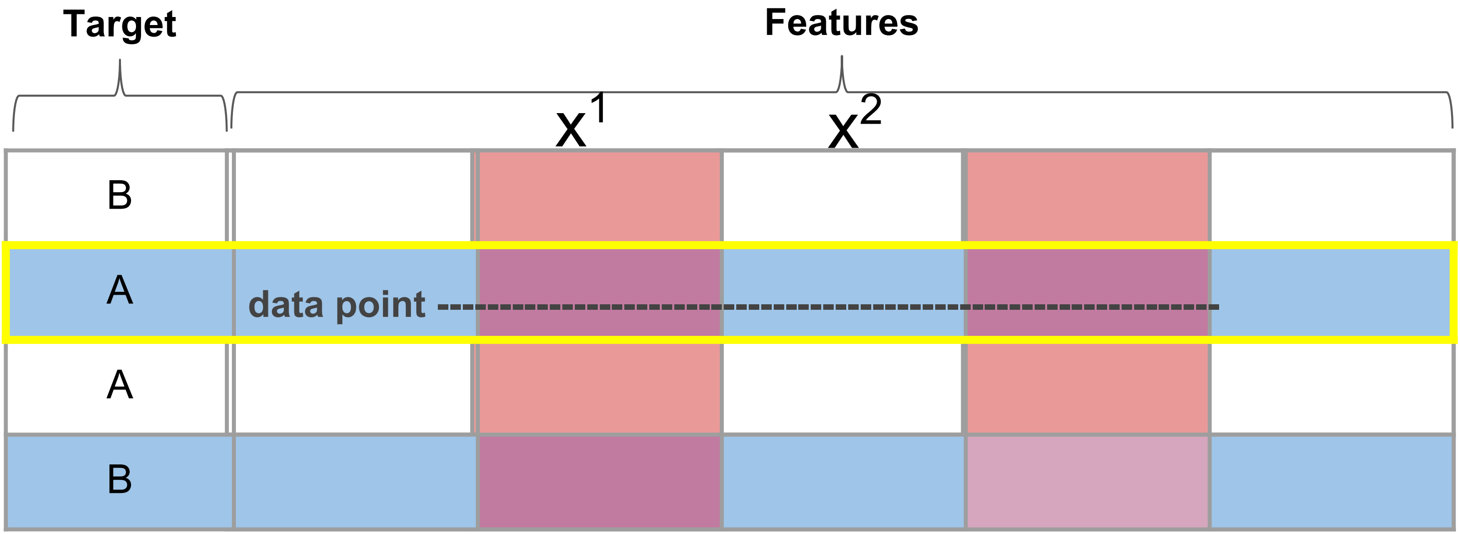


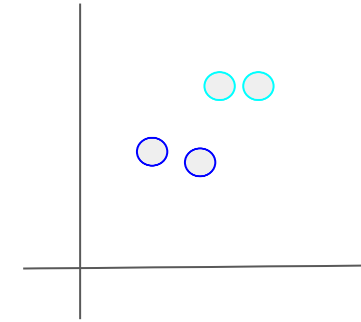
Learns a series of if-then-else rules to apply to X to get Y

Neural Network



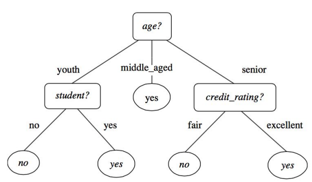
Learns a series of matrix multiplications to apply to X (an embedding) to get Y





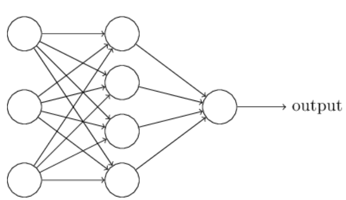
Classification: $X \rightarrow Y$ (the target)

Decision Tree



Learns a series of if-then-else rules to apply to X to get Y

Neural Network



Learns a series of matrix multiplications to apply to X (an embedding) to get Y

Loosely similar concepts

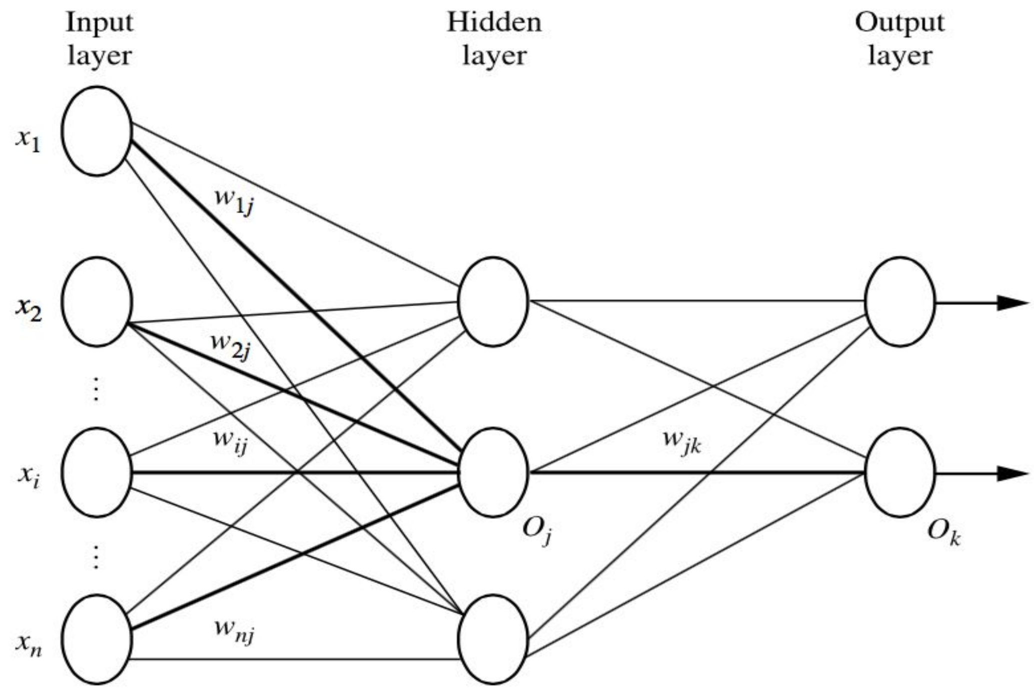
Decision Trees	Neural Networks
Impurity	Loss
Depth	Layers
Rules	Weights

Neural Networks - Overview

- Weaknesses
  - Long training times
  - Large hyper parameter
    - # hidden layers, # hidden nodes in each layer, learning rate, max epochs, noise injection, validation set stopping criterion
  - Interpretability (area of research)
- Strengths
  - Turing complete
  - High tolerance for noise
  - Recent progress in GPU computing has sped up training considerably (~10-100x vs CPU)
  - Dimensionality reduction techniques (t-sne) have opened up possibilities for interpretability

Neural Networks - Anatomy

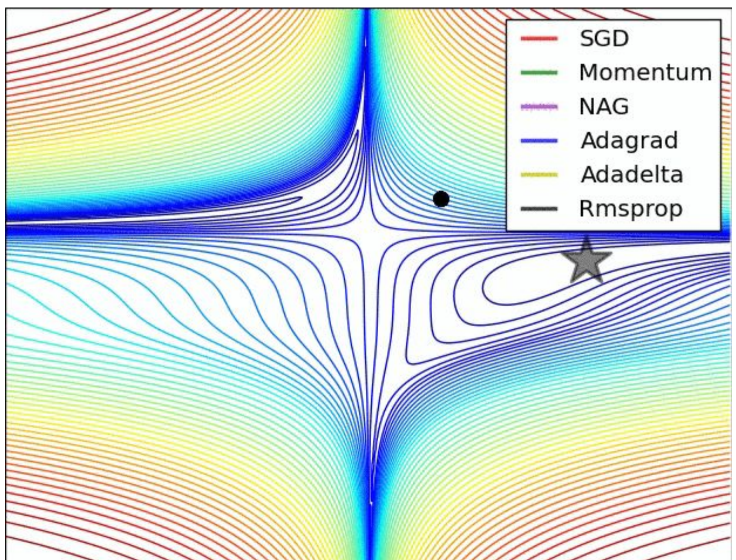
- Backpropagation
- Input layer
- Hidden layer
- Output layer
- *'n'* layer network
- "fully connected"
- Activations



Han, Kamber, Pei (2011), Sec 9.2.1

Neural Networks - Anatomy

- Backpropagation
  - Algorithm for iteratively improving the prediction of the model by updating the model's weights based on prediction error (loss)
  - Typically variants on Stochastic Gradient Descent (SGD)



An overview of gradient descent optimization algorithms (Rudder, blog)

Neural Networks - Anatomy

- Backpropagation
- Input layer
  - A vector of numeric features
    - Categoricals are typically converted to one-hot
    - Numeric is typically normalized
  - Number of nodes in the output layer = # of classes

Neural Networks - Anatomy

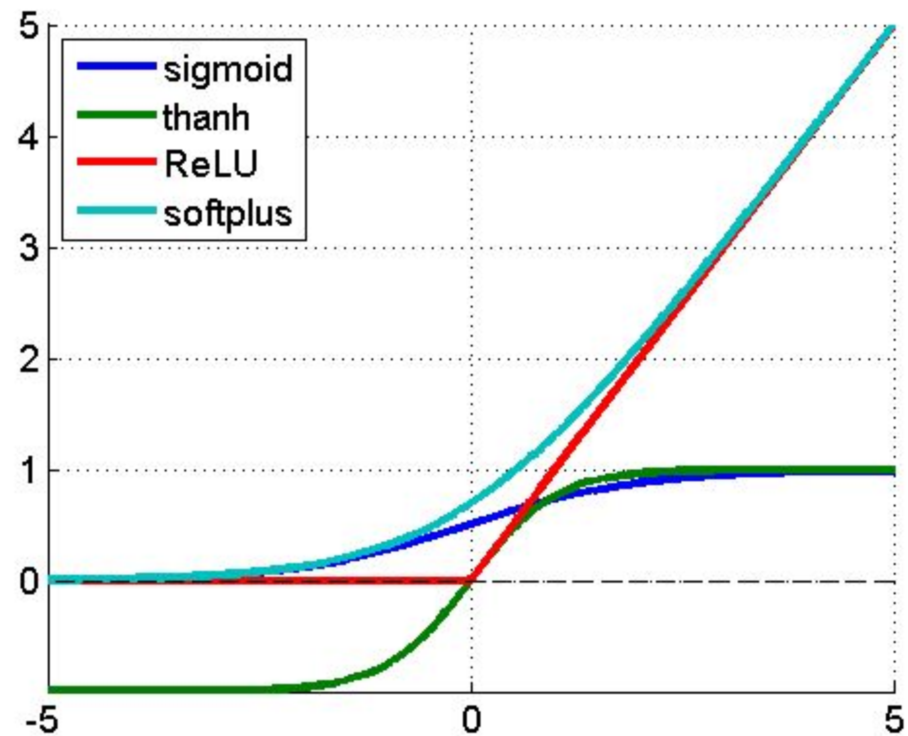
- Backpropagation
- Input layer
- Hidden layer
  - All-connected to the previous and next layers’ nodes
  - Has a user defined number of nodes
  - There can be multiple hidden layers stacked on top of one another
  - Nodes in the hidden layer have an activation function
  - Different hidden layers can have different numbers of nodes and activations

Neural Networks - Anatomy

- Backpropagation
- Input layer
- Hidden layer
- Output layer
  - The number of nodes corresponds to the number of classes being predicted
    - Except for binary classes, which are represented with a single node
  - Nodes in the output layer also have an activation function
  - A single output node can be used for numeric prediction (regression)
  - The activation function should be compatible with the output type

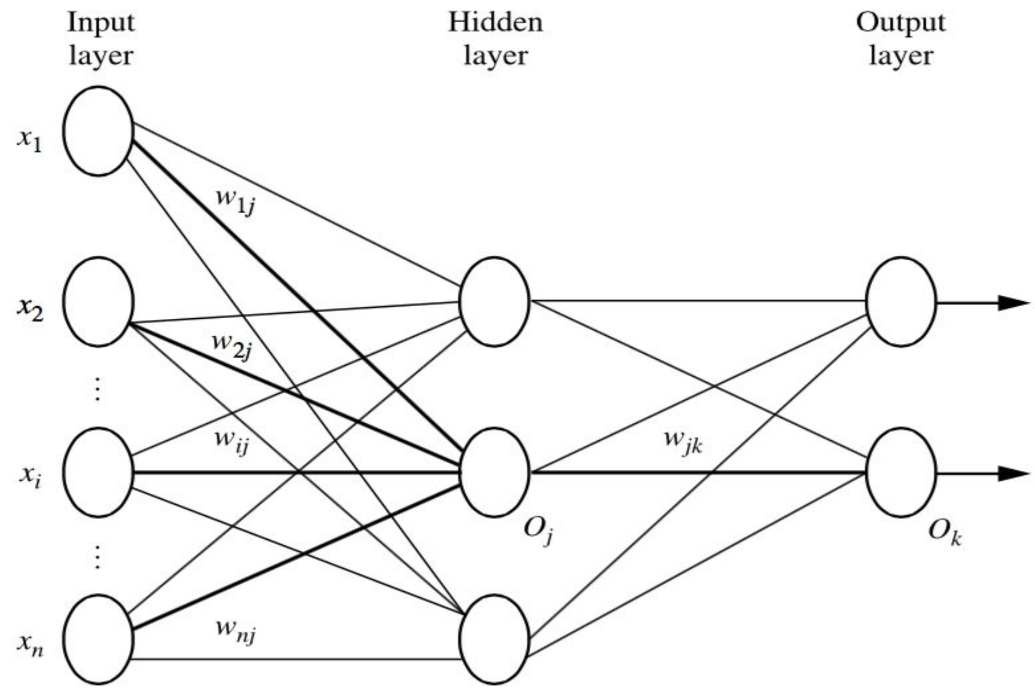
Neural Networks - Anatomy

- Backpropagation
- Input layer
- Hidden layer
- Output layer
- Activations



Neural Networks - Anatomy

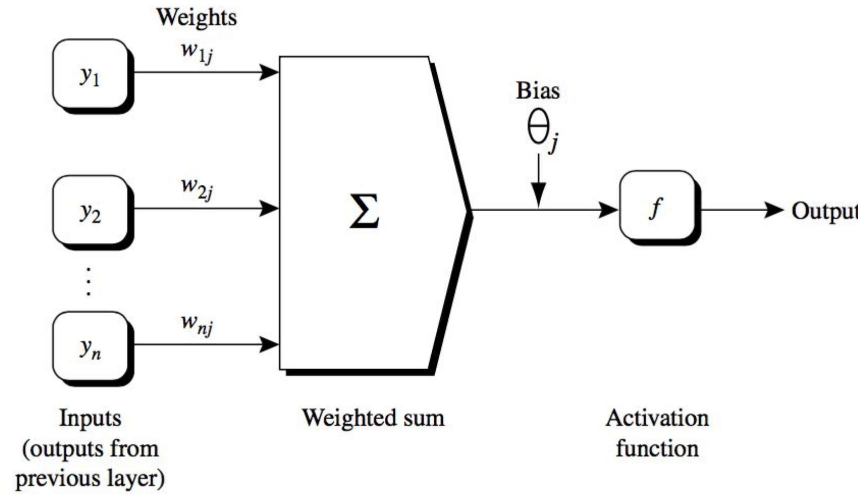
- Backpropagation
- Input layer
- Hidden layer
- Output layer
- *‘n’* layer network
    - n = # layers - the input layer
- “All (or fully) connected”
    - In a typical Multi-layer feed-forward network, all nodes in the previously layer are connected to all nodes in the subsequent later



Han, Kamber, Pei (2011), Sec 9.2.1

The Perceptron (node)

1. Weight Initialization
   - Weights
   - bias
2. Forward propagation of Input
   - $I_j = \sum_i w_{ij} O_i + \theta_j$, θ_j is the **bias** of the unit.
   - $O_j = \frac{1}{1+e^{-I_j}}$. Squashing function



Han, Kamber, Pei (2011), Sec 9.2.3

Feed-forward neural network

Logistic regression

Input = one-hot of sock color (3 values)
Output = are shoes black? (binary)

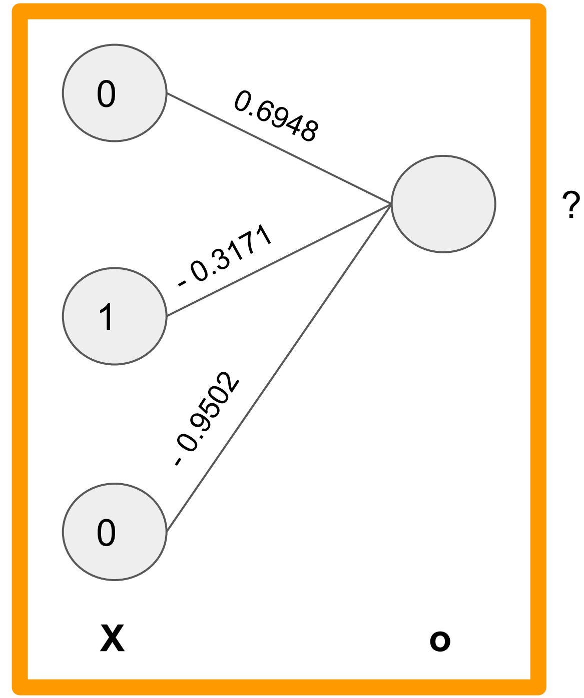
$X =$

0	1	0

$W_{xo} =$

0.6948
- 0.3171
- 0.9502

$O = X^*W_{xo} =$?



No bias, no squashing (activation) function

Feed-forward neural network

Logistic regression

Input = one-hot of sock color (3 values)
Output = are shoes black? (binary)

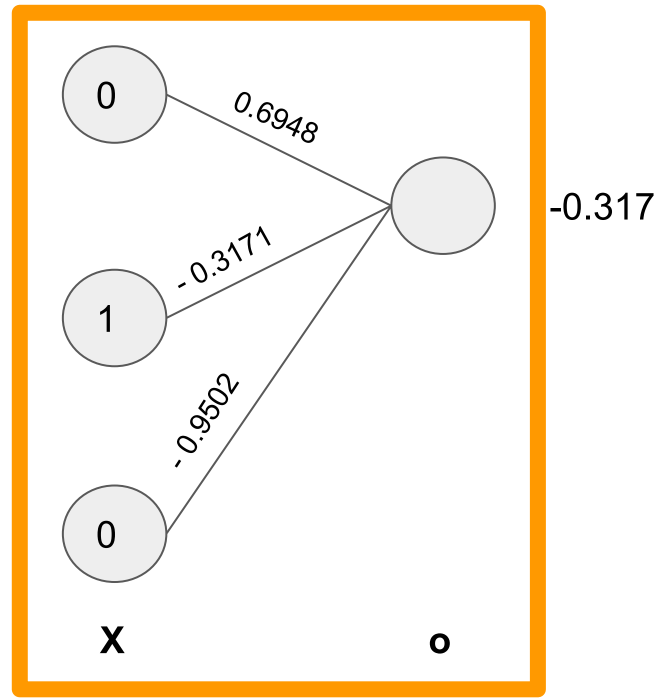
$X =$

0	1	0

$W_{xo} =$

0.6948
- 0.3171
- 0.9502

$O = X^*W_{xo} =$ -0.3171



What needs to be added to complete this logistic model?

No bias, no squashing (activation) function

Feed-forward neural network

Logistic regression

Input = one-hot of sock color (3 values)
Output = are shoes black? (binary)

$X =$

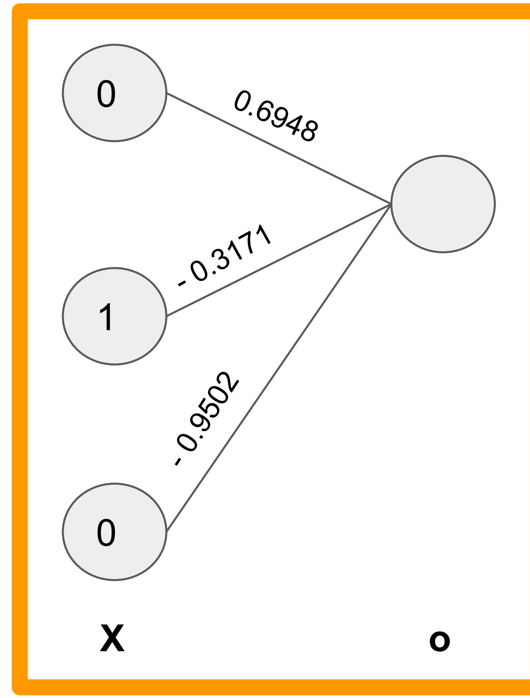
0	1	0

$W_{xo} =$

0.6948
- 0.3171
- 0.9502

$O = X^*W_{xo} =$ -0.3171

sigmoid(O) = ?



$$\frac{1}{1+e^{-(-0.3171)}}$$

Add a sigmoid activation to "squash" input to an output domain of 0 to 1

Sigmoid = $\frac{1}{1+e^{-x}}$

No bias

Feed-forward neural network

Logistic regression

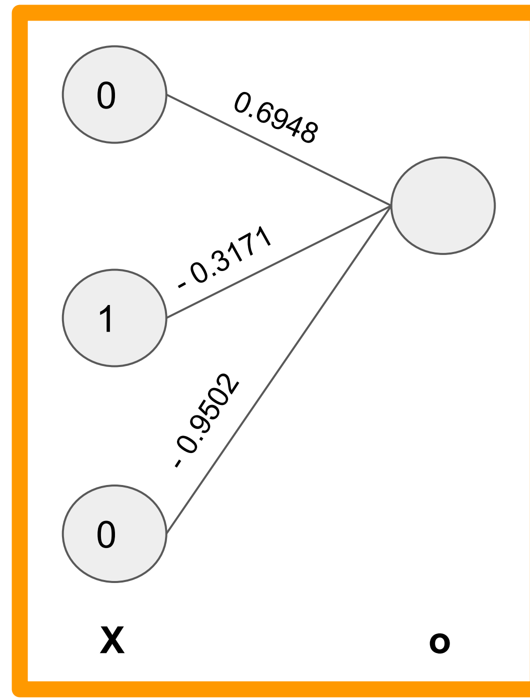
Input = one-hot of sock color (3 values)
Output = are shoes black? (binary)

$$X = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}$$

$$W_{xo} = \begin{bmatrix} 0.6948 \\ -0.3171 \\ -0.9502 \end{bmatrix}$$

$$O = X^*W_{xo} = \boxed{-0.3171}$$

$$\text{sigmoid}(O) = \boxed{0.4214}$$



$$\frac{1}{1+e^{-(-0.3171)}}$$

Add a sigmoid activation to "squash" input to an output domain of 0 to 1

Sigmoid = $\frac{1}{1+e^{-x}}$

No bias

Feed-forward neural network

Logistic regression

Input = one-hot of sock color (3 values)
Output = are shoes black? (binary)

$X =$

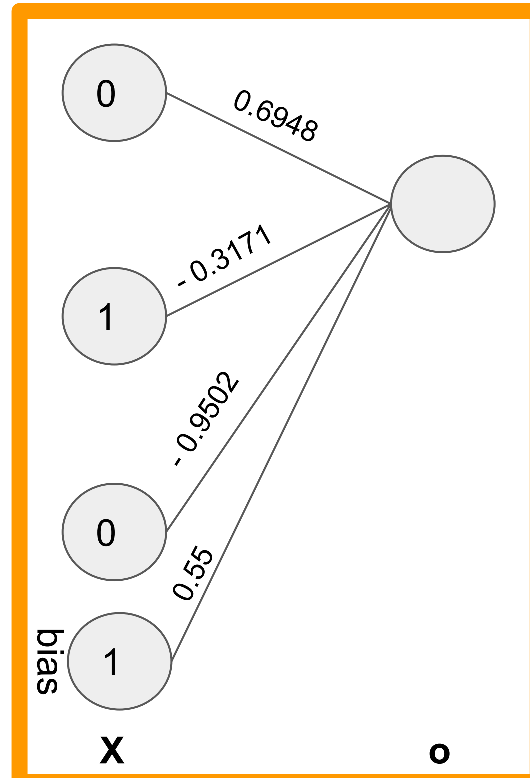
0	1	0

$W_{xo} =$

0.6948
- 0.3171
- 0.9502

$O = X^*W_{xo} =$ -0.3171 + bias

sigmoid(O) = 0.5580



Add a bias is like adding a constant "hot" feature to your instances

Feed-forward neural network

Logistic regression

Input = one-hot of sock color (3 values)
Output = are shoes black? (binary)

$X =$

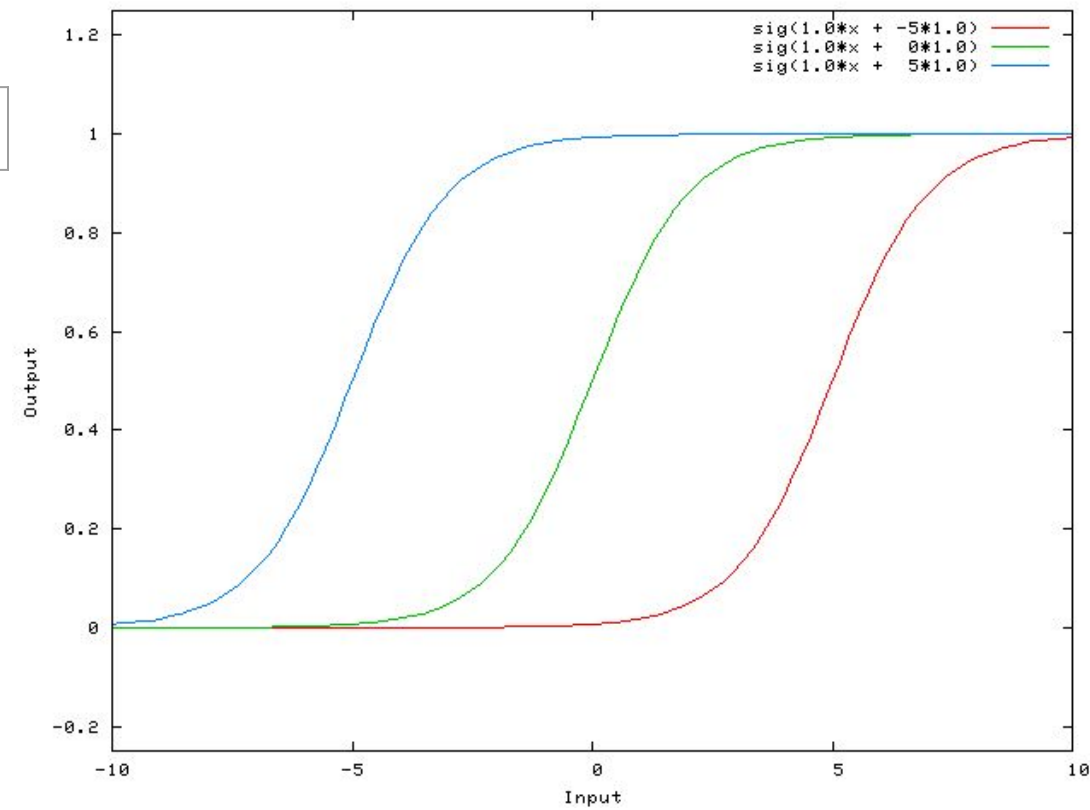
0	1	0

$W_{xo} =$

0.6948
- 0.3171
- 0.9502

$O = X*W_{xo} =$ -0.3171 + bias

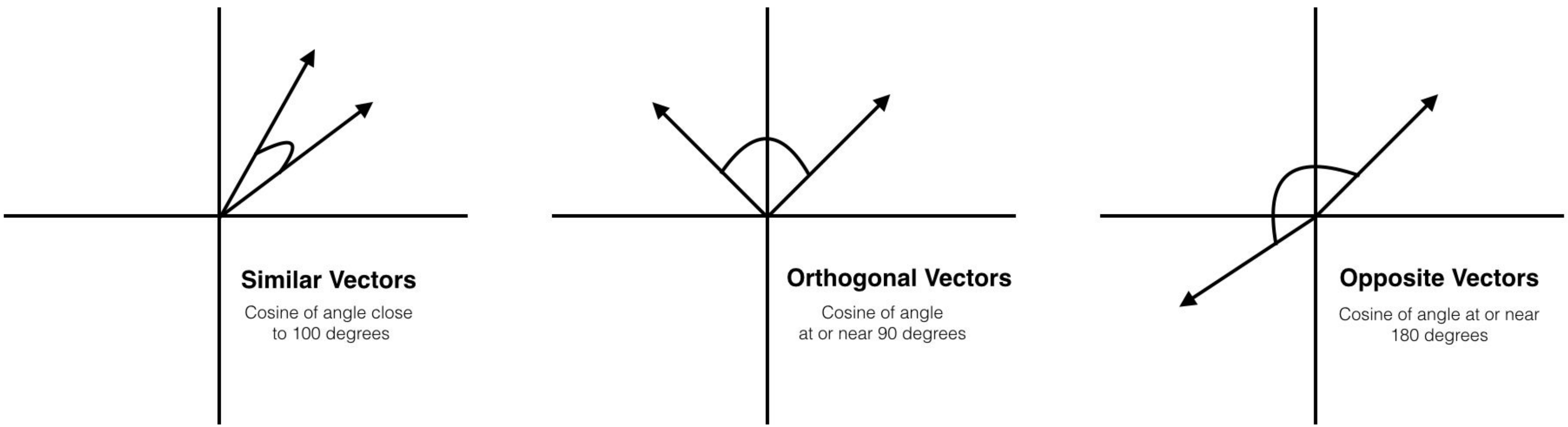
sigmoid(O) = 0.5580



Add a bias is like adding a constant "hot" feature to your instances

(In a logistic regression, bias is the y-intercept)

Relevance to Lin. Alg.



Feed-forward neural network

Logistic regression

Input = one-hot of sock color (3 values)
Output = are shoes black? (binary)

$X =$

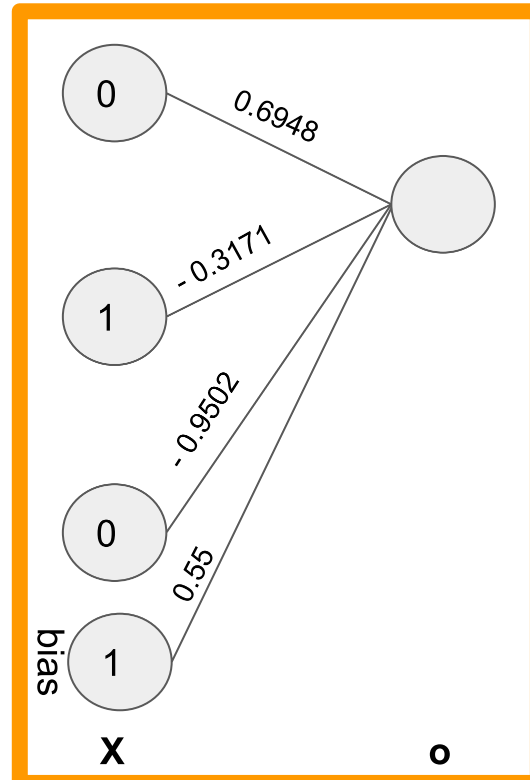
0	1	0

$W_{xo} =$

0.6948
- 0.3171
- 0.9502

$O = X{*}W_{xo} =$ -0.3171 + bias

sigmoid(O) = 0.5580



What about generalizing to classification problem with more than two classes?

Feed-forward neural network

(Multinomial) Logistic regression

Input = one-hot of sock color (3 values)
Output = are shoes black? (binary)

X =

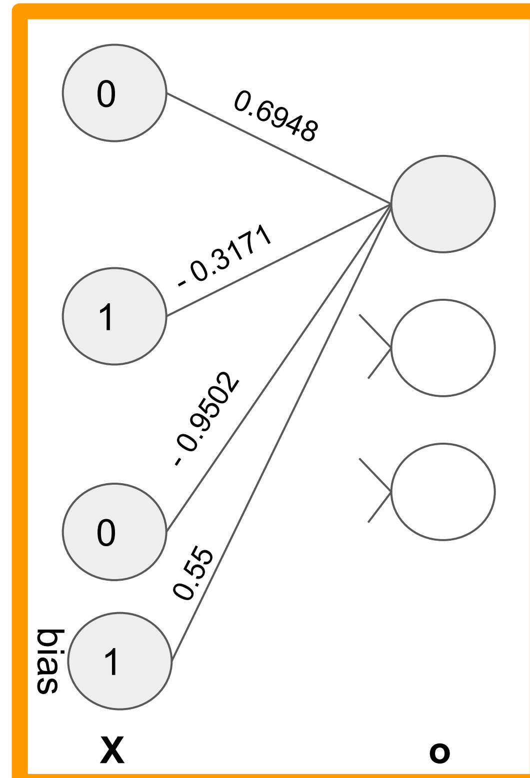
0	1	0

W_{xo} =

0.6948	...	...
- 0.3171	...	...
- 0.9502	...	...

O = $X*W_{xo}$ =

argmax(softmax(O)) = Class prediction



$$y_i = \frac{e^{z_i}}{\sum_{j \in Classes} e^{z_j}}$$

Softmax Activation

Generalizes to multi-class (multinomial logistic) with an output node per class + softmax activation

Feed-forward neural network

Multilayer Perceptron

Input size = 3
Hidden size = 2

$X =$

0	1	0

$W_{xh} =$

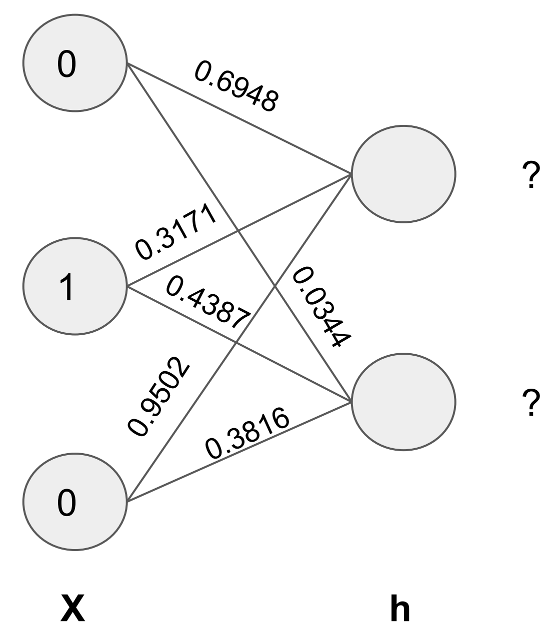
0.6948	0.0344
0.3171	0.4387
0.9502	0.3816

input dimensionality

output dimensionality

$h = X*W_{xh} =$

?	?



No bias, no squashing (activation) function

Feed-forward neural network

Input size = 3
Output size = 2

$X =$

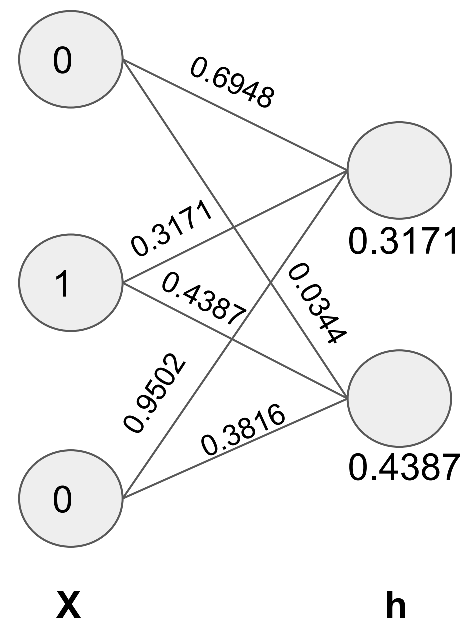
0	1	0

$W_{xh} =$

0.6948	0.0344
0.3171	0.4387
0.9502	0.3816

$h =$

0.3171	0.4387



No bias, no squashing (activation) function

Feed-forward neural network

Input size = 3
Output size = 2

$X =$

0	1	0

$W_{xh} =$

0.6948	0.0344
0.3171	0.4387
0.9502	0.3816

$h =$

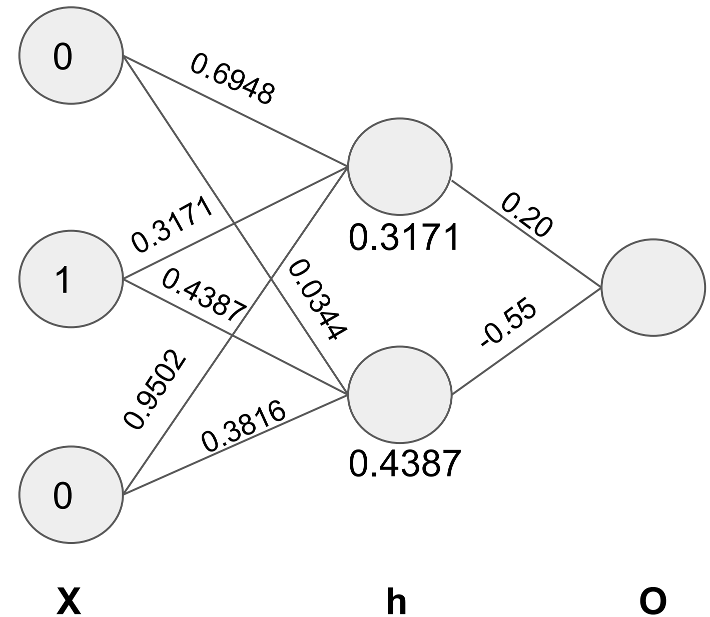
0.3171	0.4387

$W_{ho} =$

0.20
-0.55

$O = h{*}W_{ho} = -0.1779$

No bias, no squashing (activation) function



Feed-forward neural network

Input size = 3
Output size = 2

$X =$

0	1	0

$W_{xh} =$

0.6948	0.0344
0.3171	0.4387
0.9502	0.3816

$h =$

0.3171	0.4387

$W_{ho} =$

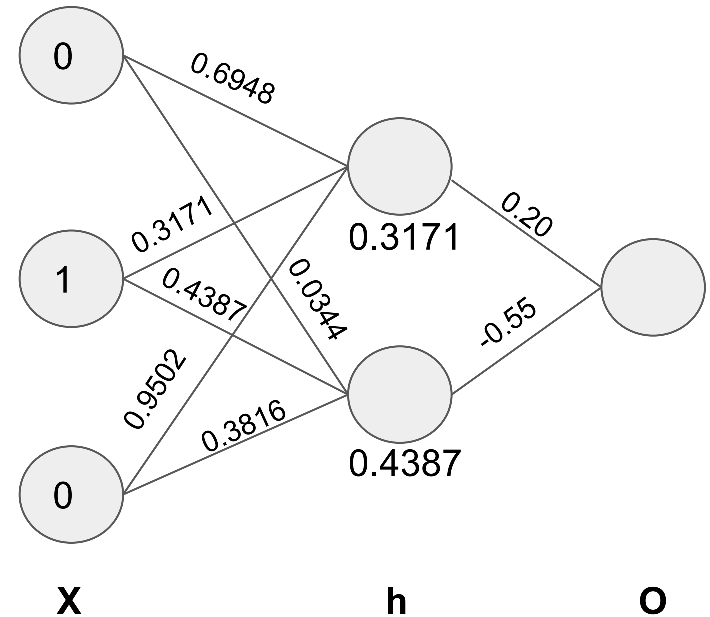
0.20
-0.55

$$O = h^*W_{ho} = -0.1779$$

sigmoid(O) =

0.4556

No bias



Feed-forward neural network

Input size = 3
Output size = 2

$X =$

0	1	0

$W_{xh} =$

0.6948	0.0344
0.3171	0.4387
0.9502	0.3816

$h =$

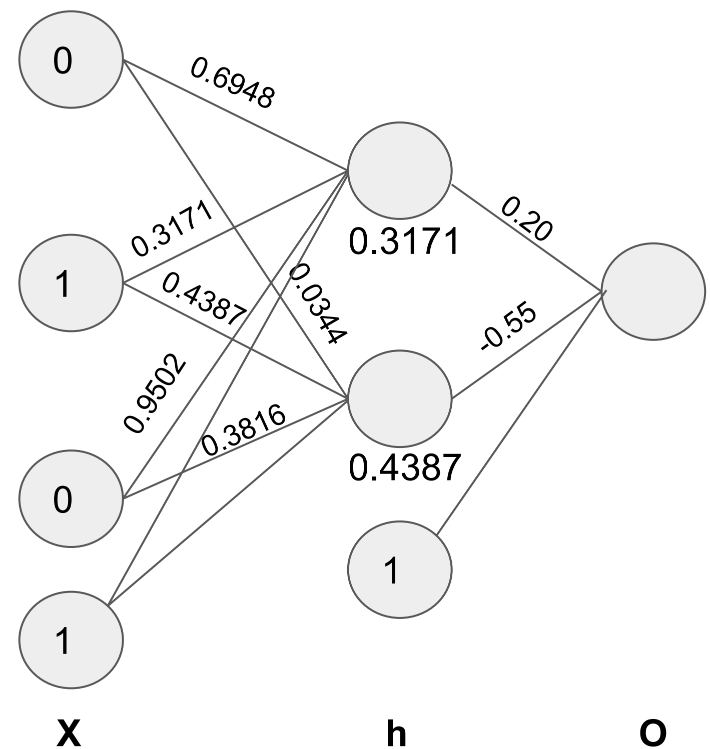
$0.3171 + h_{bias[0]}$	$0.4387 + h_{bias[1]}$

$W_{ho} =$

0.20
-0.55

$$O = h*W_{ho} + O_{bias}$$

$\text{sigmoid}(O) =$



Example Keras implementation

Of a multilayer perceptron

```python
from keras.layers import Input, Dense
from keras.models import Model

# This returns a tensor
inputs = Input(shape=(784,))

# a layer instance is callable on a tensor, and returns a tensor
x = Dense(64, activation='relu')(inputs)
x = Dense(64, activation='relu')(x)
predictions = Dense(10, activation='softmax')(x)

# This creates a model that includes
# the Input layer and three Dense layers
model = Model(inputs=inputs, outputs=predictions)
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.fit(data, labels)  # starts training
```

Use of Keras is optional for the upcoming lab - scikit's neural network package is acceptable and simpler

Exercise

1. Can you find weight values that minimize the error for this instance?
2. Do these values work well for a 2nd instance: X = [1 0 0], Y = 1?
3. What modifications (if necessary) were needed to accommodate both instances (minimize the average of their errors)?

X =

0	1	0

Y = 0

W_{xh} =

0.6948	0.0344
0.3171	0.4387
0.9502	0.3816

h =

0.3171	0.4387

W_{ho} =

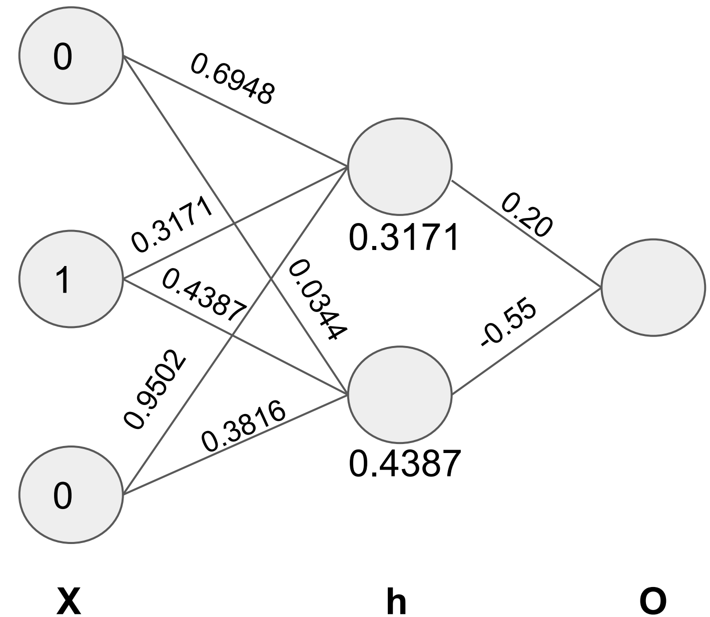
0.20
-0.55

$O = h^*W_{ho} = -0.1779$

sigmoid(O) =

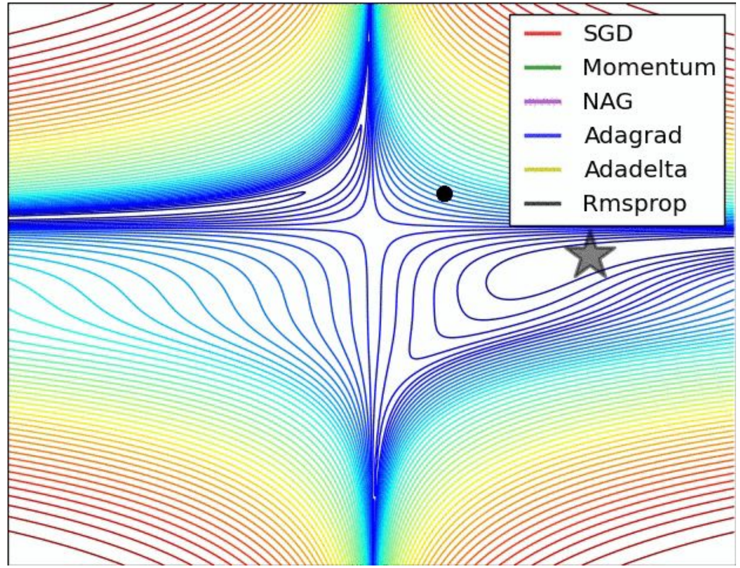
0.4556

Error = |Sigmoid(O) - Y|



$$\text{Sigmoid} = \frac{1}{1+e^{-x}}$$

SGD



Backpropagation using Stochastic Gradient Descent
(for efficient weight value search)

Covered Thursday

Additional Neural Network details

- Batch size
- Epoch
- Stopping criteria
- Rules of thumb in network design
- Tractability
- Applications (ImageNet, word2vec, AlphaGo)
- GPU acceleration
- Neural network software frameworks

Data Mining & Analytics

Prof. Zach Pardos

INFO254/154: Spring ’19

The remainder of the slides in this deck will be covered on Thursday (lab tutorial)

Process - continued

3. Backpropagation of error

- $$Err_j = O_j(1 - O_j)(T_j - O_j)$$
    - Error in a single output unit
    - Oj - actual output of unit j
    - Tj - known target value of the training tuple
- $$Err_j = O_j(1 - O_j)\sum_k Err_k w_{jk}$$
    - Error of a hidden unit is the weighted sum of the errors connected to it
- Weights and biases are updated.
    - *(l)* is the learning rate

$$\Delta w_{ij} = (l) Err_j O_i. \qquad \Delta\theta_j = (l) Err_j.$$

$$w_{ij} = w_{ij} + \Delta w_{ij}. \qquad \theta_j = \theta_j + \Delta\theta_j.$$

Han, Kamber, Pei (2011), Sec 9.2.3

Algorithm - Neural Networks

Algorithm: Backpropagation. Neural network learning for classification or numeric prediction, using the backpropagation algorithm.

Input:

- D, a data set consisting of the training tuples and their associated target values;
- l, the learning rate;
- *network*, a multilayer feed-forward network.

Output: A trained neural network.

Method:

(1) Initialize all weights and biases in *network*;
(2) **while** terminating condition is not satisfied {
(3)   **for** each training tuple $\boldsymbol{X}$ in D {
(4)   // Propagate the inputs forward:
(5)   **for** each input layer unit j {
(6) $O_j = I_j$; // output of an input unit is its actual input value
(7)   **for** each hidden or output layer unit j {
(8) $I_j = \sum_i w_{ij} O_i + \theta_j$; //compute the net input of unit j with respect to the previous layer, i
(9) $O_j = \frac{1}{1+e^{-I_j}}$; } // compute the output of each unit j

Han, Kamber, Pei (2011), Sec 9.2.2

Algorithm - Neural Networks (continued)

// Backpropagate the errors:
for each unit j in the output layer
    $Err_j = O_j(1 - O_j)(T_j - O_j)$; // compute the error
for each unit j in the hidden layers, from the last to the first hidden layer
    $Err_j = O_j(1 - O_j)\sum_k Err_k w_{jk}$; // compute the error with respect to the next higher layer, k
for each weight w_{ij} in *network* {
    $\Delta w_{ij} = (l) Err_j O_i$; // weight increment
    $w_{ij} = w_{ij} + \Delta w_{ij}$; } // weight update
for each bias θ_j in *network* {
    $\Delta \theta_j = (l) Err_j$; // bias increment
    $\theta_j = \theta_j + \Delta \theta_j$; } // bias update
} }

Han, Kamber, Pei (2011), Sec 9.2.2

Terminating Condition

Terminating condition: Training stops when

- All Δw_{ij} in the previous epoch are so small as to be below some specified threshold, or
- The percentage of tuples misclassified in the previous epoch is below some threshold, or
- A prespecified number of epochs has expired.

- Accuracy no longer is increasing on a held out validation set

Han, Kamber, Pei (2011), Sec 9.2.3

Example

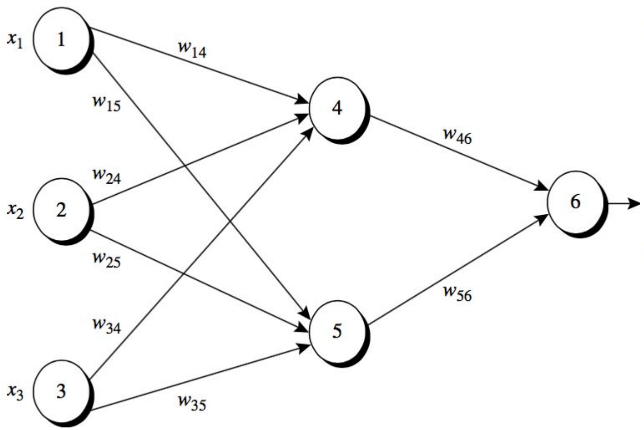


Table 9.1 Initial Input, Weight, and Bias Values

x_1	x_2	x_3	w_{14}	w_{15}	w_{24}	w_{25}	w_{34}	w_{35}	w_{46}	w_{56}	θ_4	θ_5	θ_6
1	0	1	0.2	−0.3	0.4	0.1	−0.5	0.2	−0.3	−0.2	−0.4	0.2	0.1

Table 9.2 Net Input and Output Calculations

Unit, j	*Net Input, I_j*	*Output, O_j*
4	$0.2 + 0 - 0.5 - 0.4 = -0.7$	$1/(1 + e^{0.7}) = 0.332$
5	$-0.3 + 0 + 0.2 + 0.2 = 0.1$	$1/(1 + e^{-0.1}) = 0.525$
6	$(-0.3)(0.332) - (0.2)(0.525) + 0.1 = -0.105$	$1/(1 + e^{0.105}) = 0.474$

Han, Kamber, Pei (2011), Sec 9.2.3

Example- continued

Table 9.3 Calculation of the Error at Each Node

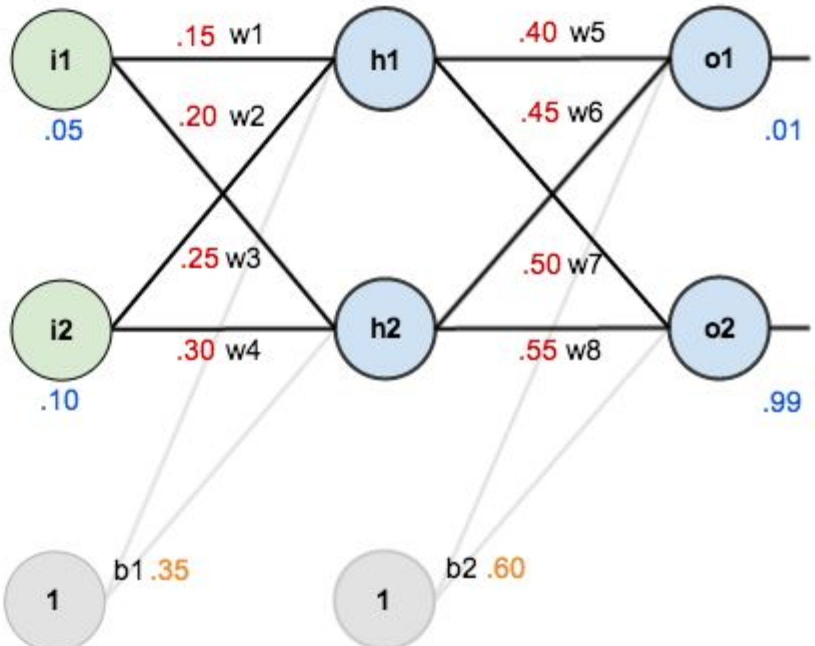
Unit, j	Err_j
6	$(0.474)(1-0.474)(1-0.474) = 0.1311$
5	$(0.525)(1-0.525)(0.1311)(-0.2) = -0.0065$
4	$(0.332)(1-0.332)(0.1311)(-0.3) = -0.0087$

Table 9.4 Calculations for Weight and Bias Updating

Weight or Bias	*New Value*
w_{46}	$-0.3 + (0.9)(0.1311)(0.332) = -0.261$
w_{56}	$-0.2 + (0.9)(0.1311)(0.525) = -0.138$
w_{14}	$0.2 + (0.9)(-0.0087)(1) = 0.192$
w_{15}	$-0.3 + (0.9)(-0.0065)(1) = -0.306$
w_{24}	$0.4 + (0.9)(-0.0087)(0) = 0.4$
w_{25}	$0.1 + (0.9)(-0.0065)(0) = 0.1$
w_{34}	$-0.5 + (0.9)(-0.0087)(1) = -0.508$
w_{35}	$0.2 + (0.9)(-0.0065)(1) = 0.194$
θ_6	$0.1 + (0.9)(0.1311) = 0.218$
θ_5	$0.2 + (0.9)(-0.0065) = 0.194$
θ_4	$-0.4 + (0.9)(-0.0087) = -0.408$

Han, Kamber, Pei (2011), Sec 9.2.3

Further Backprop Example



<u>Calculate</u>

1. The outputs of all nodes on the first feed-forward pass
2. The errors at each hidden and output node
3. The updated weight values
4. The outputs of all nodes on the second feed-forward pass

NOTE: skip bias updates to save time

Guide [here]