Introduction to Neural Networks

Data-X Spring 2019

Tanya Piplani

What is a Neural Network?

•Like other machine learning methods that we saw earlier in the class , it is a technique to:

***map features** to labels or some dependent continuous value.*

or

•***compute the function** that relates features to labels or some dependent continuous value.*

What is a Neural Network?

•Like other machine learning methods that we saw earlier in the class , it is a technique to:

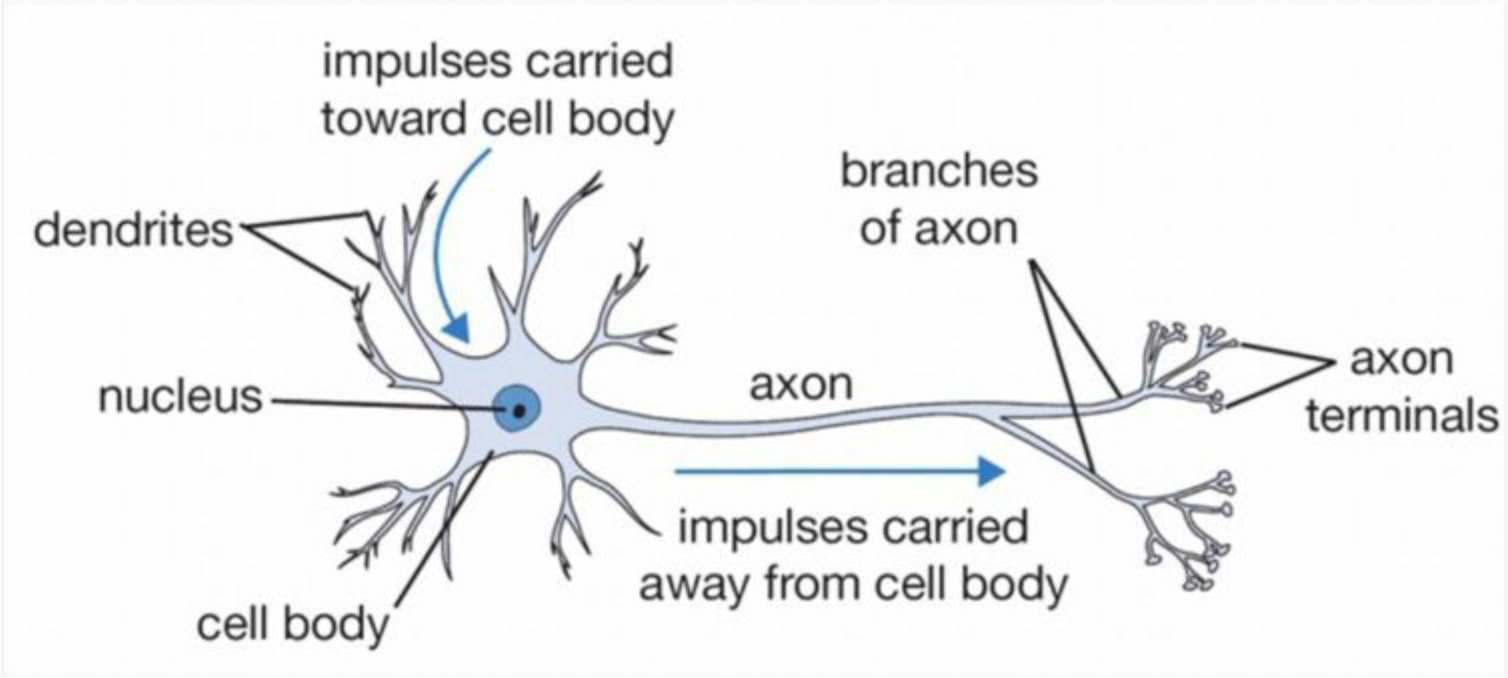
map features *to labels or some dependent continuous value.*

or

$$f(y|x)$$

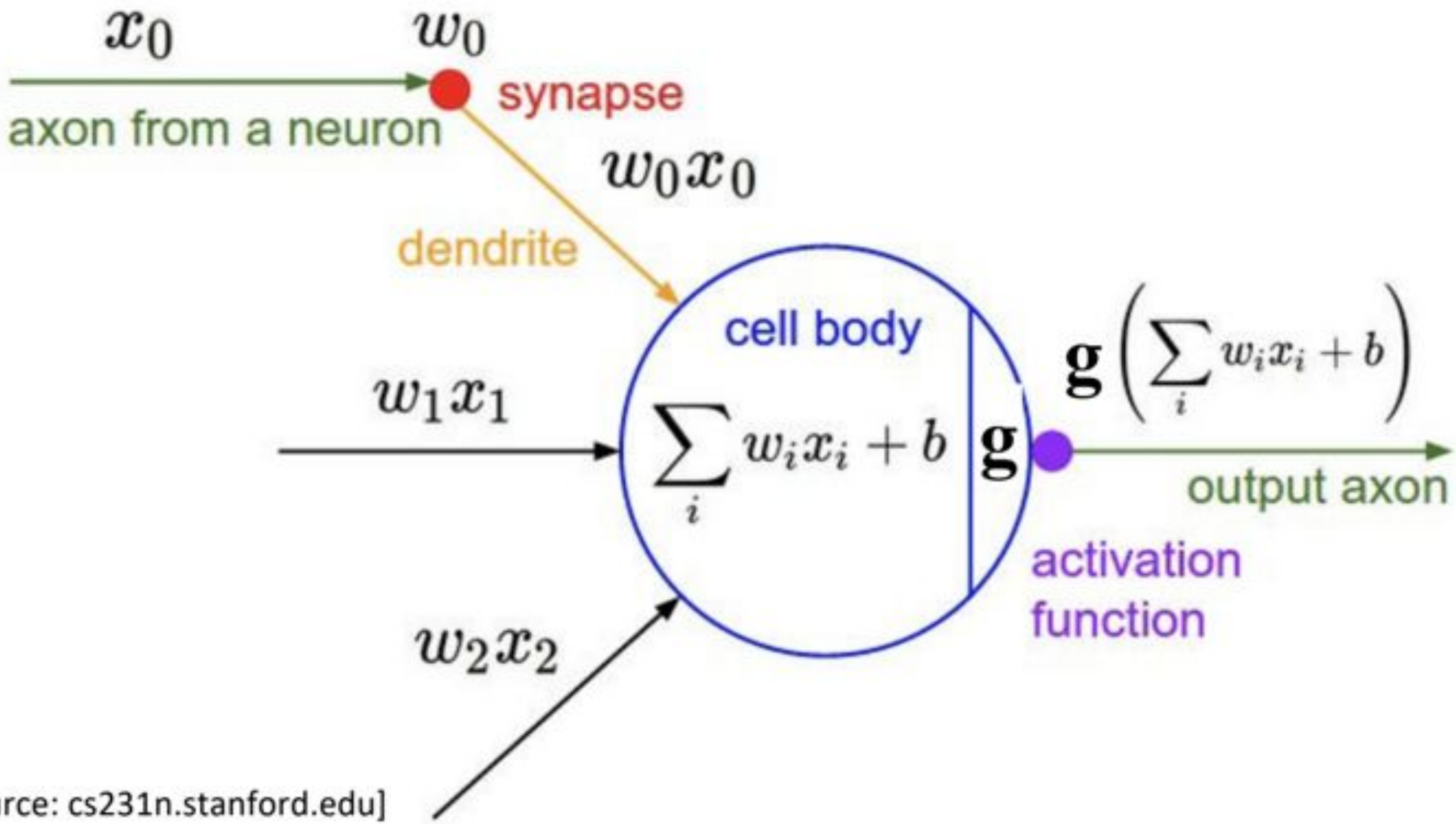
•***compute the function*** *that relates features to labels or some dependent continuous value.*

Single (Biological) Neuron



[image source: cs231n.stanford.edu]

Single (Artificial) Neuron



[image source: cs231n.stanford.edu]

Types of Learning

- Supervised Learning
- Unsupervised Learning
- Reinforcement Learning

- [also Transfer Learning, Imitation Learning , Meta Learning etc …]

Supervised Learning: X → Y

- Ex 1: Image Recognition
  - X = pixel values
  - Y = one hot vector encoding category

$$\rightarrow \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

x **y**

$$\rightarrow \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

x **y**

[figure sources: https://en.wiktionary.org/wiki/cat; https://www.guidedogs.org/]

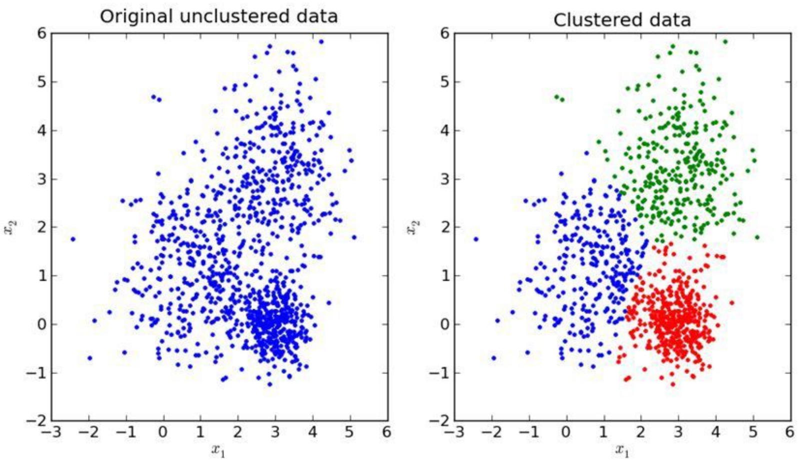
Unsupervised Learning

Unsupervised learning is a type of machine

Learning algorithm used to draw inferences

from datasets consisting of input data

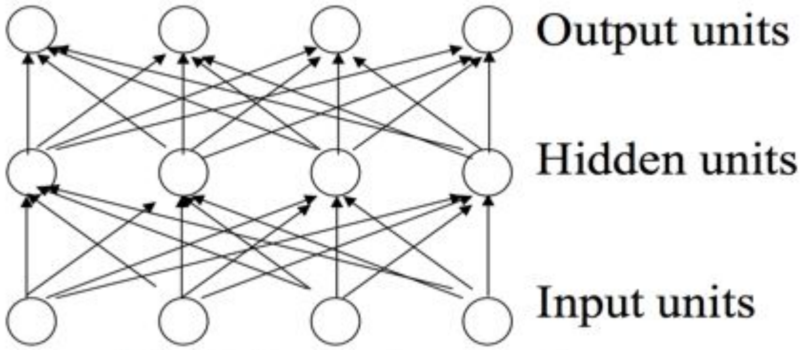
without labeled responses.



Neural Networks

- Origins: Algorithms that try to mimic the brain.
- Very widely used in 80s and early 90s; popularity diminished in late 90s.
- Recent resurgence: State-of-the-art technique for many applications
- Artificial neural networks are not nearly as complex or intricate as the actual brain structure

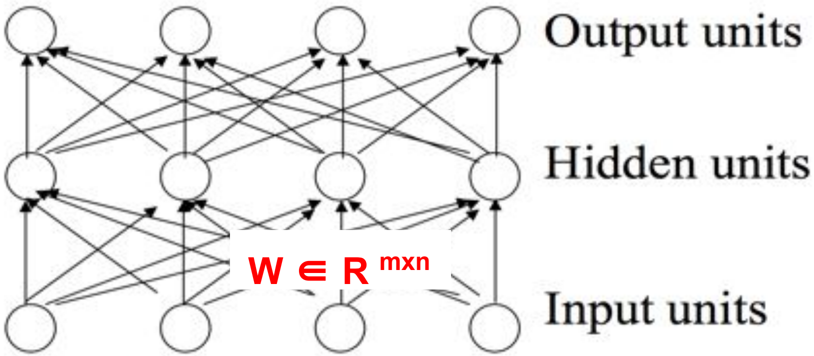
Neural networks



Layered feed-forward network

- Neural networks are made up of **nodes** or **units**, connected by **links**
- Each link has an associated **weight** and **activation level**
- Each node has an **input function** (typically summing over weighted inputs), an **activation function**, and an **output**

Neural networks

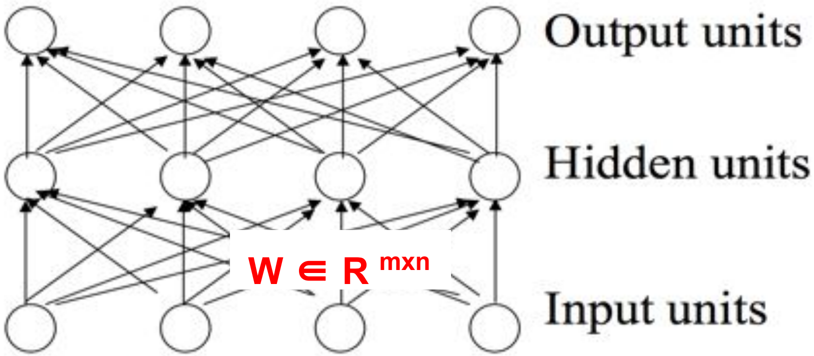


Layered feed-forward network

- Neural networks are made up of **nodes** or **units**, connected by **links**
- Each link has an associated **weight** and **activation level**
- Each node has an **input function** (typically summing over weighted inputs), an **activation function**, and an **output**

Based on slide by T. Finin, M. desJardins, L Getoor, R. Par

Neural networks



Layered feed-forward network

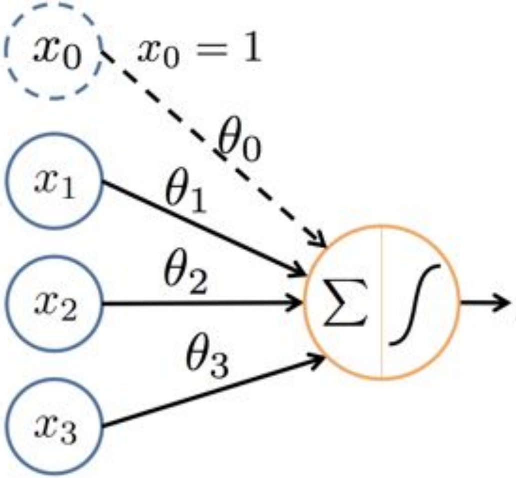
Fully Connected !!
Why ?

- Neural networks are made up of **nodes** or **units**, connected by **links**
- Each link has an associated **weight** and **activation level**
- Each node has an **input function** (typically summing over weighted inputs), an **activation function**, and an **output**

Based on slide by T. Finin, M. desJardins, L Getoor, R. Par

Neuron Model: Logistic Unit

“bias unit”



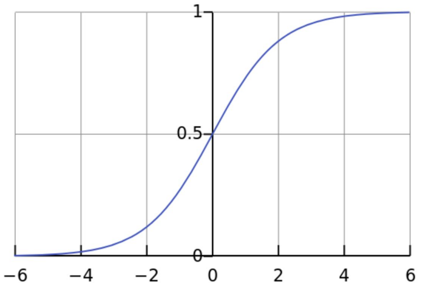
$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = g\left(\boldsymbol{\theta}^\mathsf{T}\mathbf{x}\right) = \frac{1}{1 + e^{-\boldsymbol{\theta}^\mathsf{T}\mathbf{x}}}$$

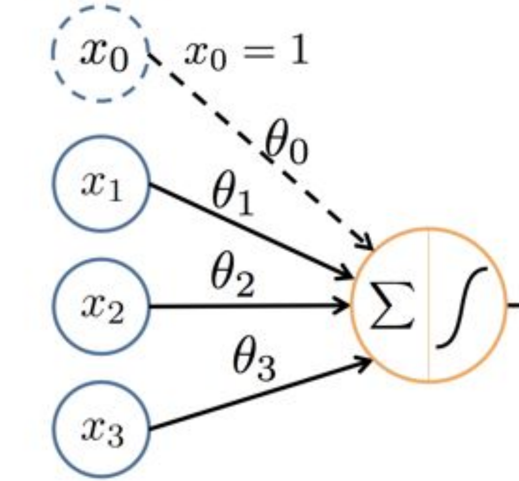
Sigmoid (logistic) activation function: $g(z) = \dfrac{1}{1 + e^{-z}}$

Sigmoid Function

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



Neuron Model: Logistic Unit



$$h_{\boldsymbol{\theta}}(\mathbf{x}) = g\left(\boldsymbol{\theta}^\mathsf{T}\mathbf{x}\right) = \frac{1}{1 + e^{-\boldsymbol{\theta}^\mathsf{T}\mathbf{x}}}$$

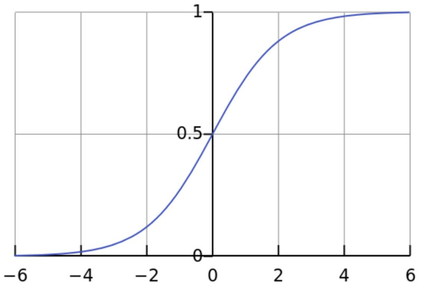
$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

Why have non-linearity???

Sigmoid (logistic) activation function: $g(z) = \dfrac{1}{1 + e^{-z}}$

Sigmoid Function

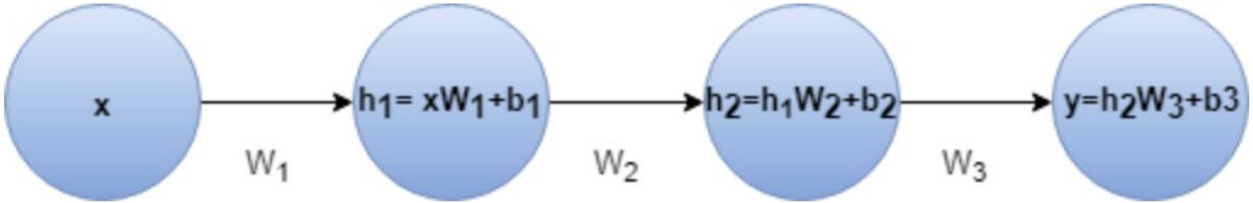
$$A = \frac{1}{1+e^{-x}}$$



Based on slide by Andrew Ng

A feed-forward neural network with linear activation and any number of hidden layers is equivalent to just a linear neural neural network with no hidden layer. For example let us consider the neural network in figure with two hidden layers and no activation-

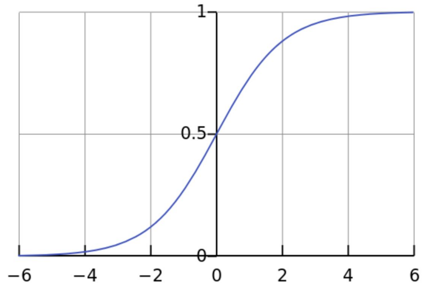
Why have non-linearity???

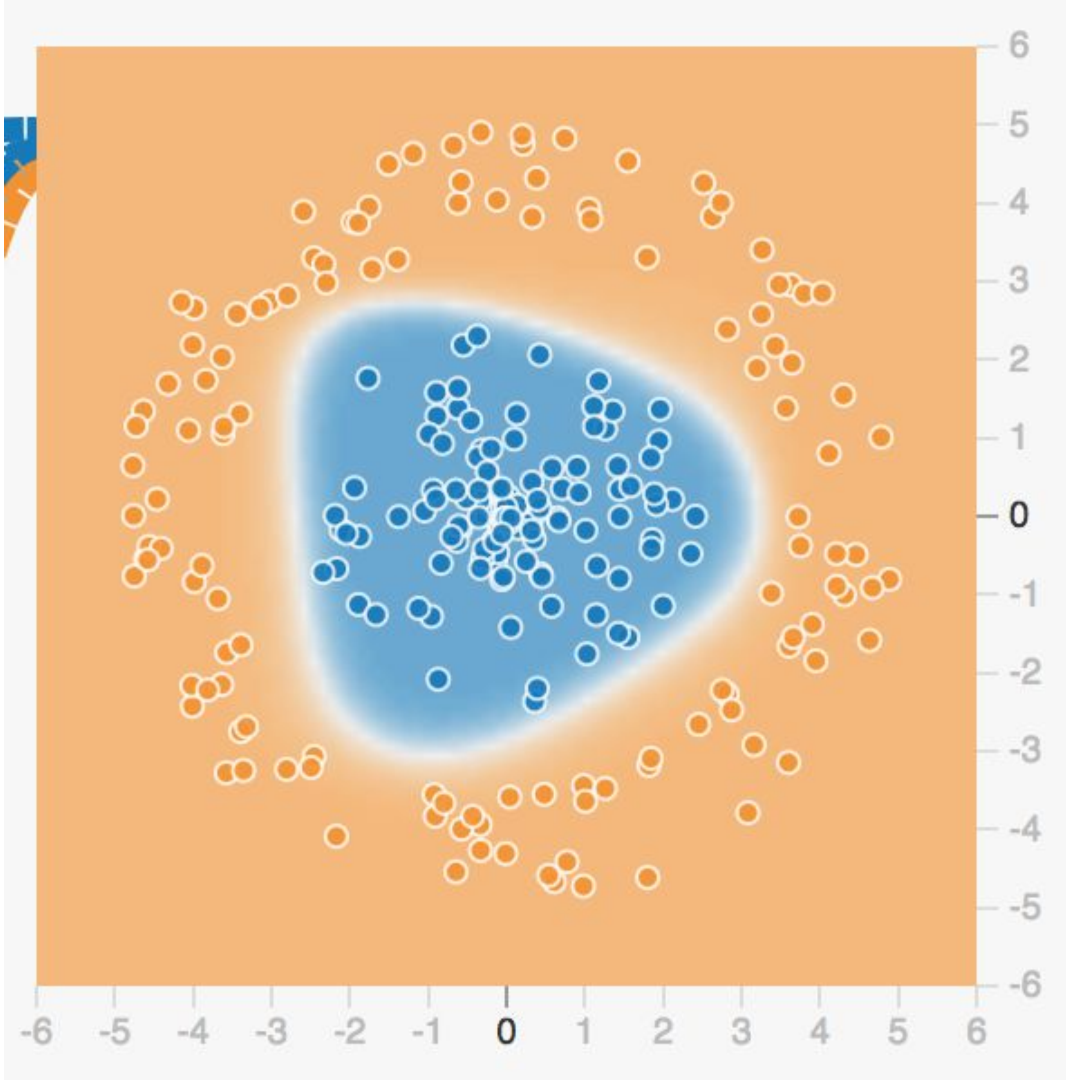



```
y = h2 * W3 + b3
  = (h1 * W2 + b2) * W3 + b3
  = h1 * W2 * W3 + b2 * W3 + b3
  = (x * W1 + b1) * W2 * W3 + b2 * W3 + b3
  = x * W1 * W2 * W3 + b1 * W2 * W3 + b2 * W3 + b3
  = x * W' + b'
```

Sigmoid Function

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

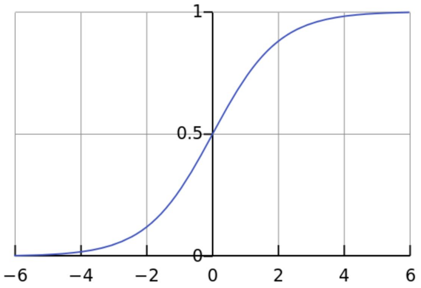




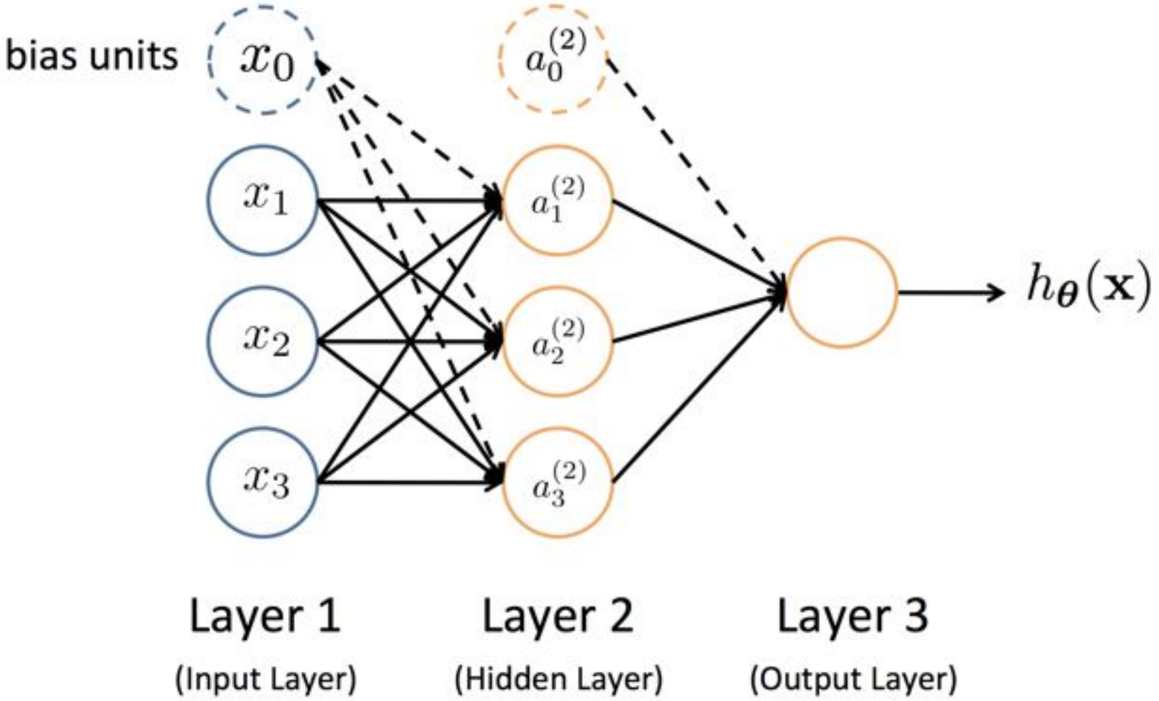
Why have non-linearity???

Sigmoid Function

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



Neural Network



Slide by Andrew Ng

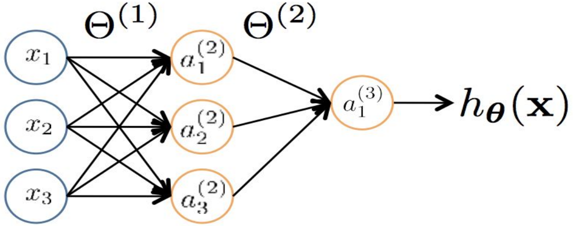
Feed-Forward Process

- Input layer units are set by some exterior function (think of these as **sensors**), which causes their output links to be **activated** at the specified level

- Working forward through the network, the **input function** of each unit is applied to compute the input value
  - Usually this is just the weighted sum of the activation on the links feeding into this node

- The **activation function** transforms this input function into a final value
  - Typically this is a **nonlinear** function, often a **sigmoid** function corresponding to the "threshold" of that node

Neural Network



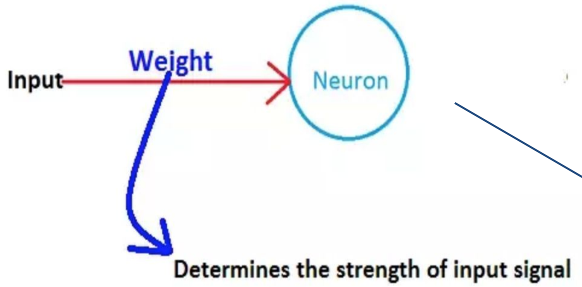
$a_i^{(j)} =$ "activation" of unit i in layer j

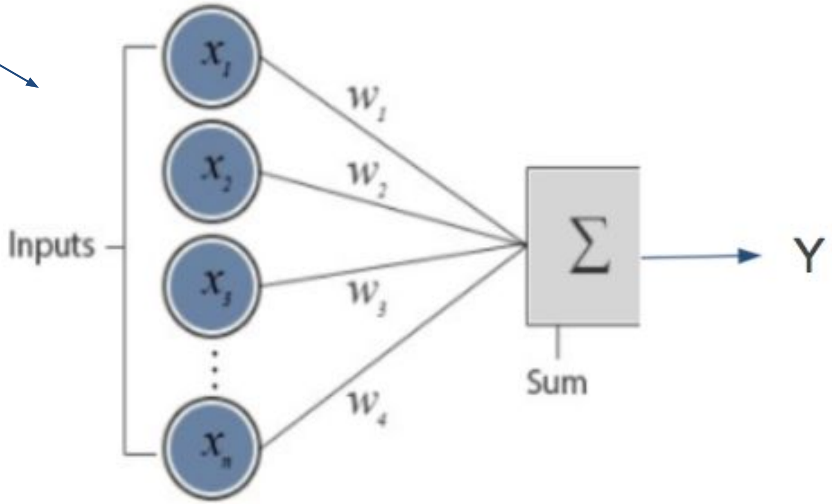
$\Theta^{(j)} =$ weight matrix controlling function mapping from layer j to layer j + 1

$$a_1^{(2)} = g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3)$$
$$a_2^{(2)} = g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3)$$
$$a_3^{(2)} = g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3)$$
$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)})$$

If network has s_j units in layer j *and* s_{j+1} units in layer $j+1$, then $\Theta^{(j)}$ has dimension $s_{j+1} \times (s_j+1)$.

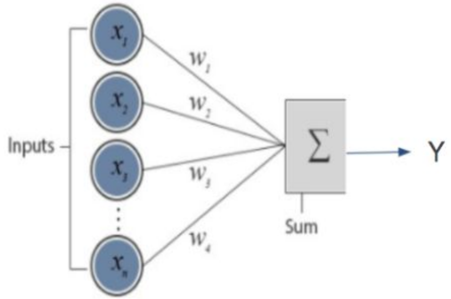
Background





$Y = x1*w1 + x2*w2 + x3*w3 + \cdots + xn*wn$ **--linear regression**

Example:

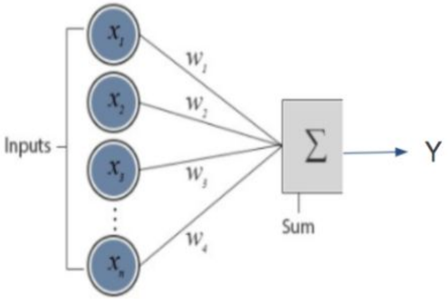


Y = x1∗w1 + x2∗w2 + x3∗w3 +·····+ xn∗wn **--linear regression**

For sample 1:				
x	6	5	3	1
w	**0.3**	**0.2**	**-0.5**	**0**
Y = ?				

For sample 2:				
x	20	5	3	1
w	**0.3**	**0.2**	**-0.5**	**0**
Y = ?				

Example:

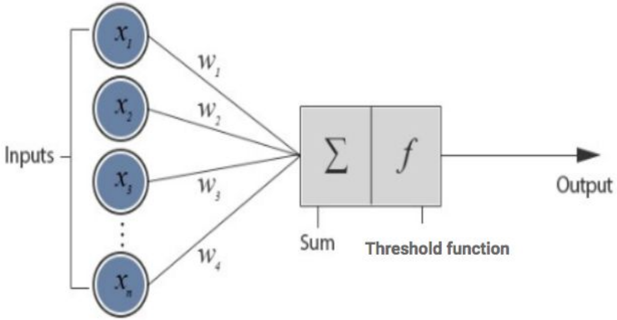


Y = x1*w1 + x2*w2 + x3*w3 +·····+ xn*wn **--linear regression**

For sample 1:				
x	6	5	3	1
w	**0.3**	**0.2**	**-0.5**	**0**
Y = sum(x * w) =**1.3**				

For sample 2:				
x	20	5	3	1
w	**0.3**	**0.2**	**-0.5**	**0**
Y = sum(x * w) = 5.5				

Lets us apply a **threshold function** on the output:

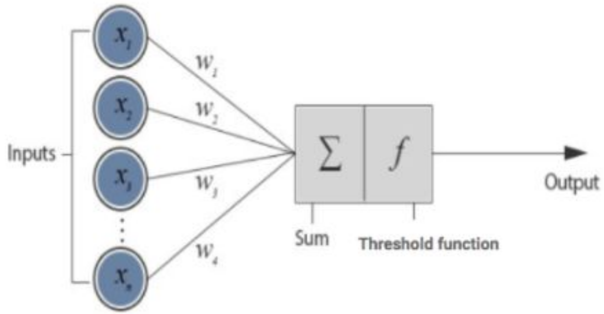


$$f(t) = \begin{cases} t & \textit{if } t < 3 \\ 0 & \textit{otherwise} \end{cases}$$

For sample 1:				
x	6	5	3	1
w	**0.3**	**0.2**	**-0.5**	**0**
Y = f(sum(x * w)) = f(1.3)= 1.3				

For sample 2:				
x	20	5	3	1
w	**0.3**	**0.2**	**-0.5**	**0**
Y = f(sum(x * w))= f(5.5)=0				

*Now, if we apply a **logistic/sigmoid function** on the output it will squeeze all the output between 0 and 1 :*



Y = Sigmoid(x1∗w1 + x2∗w2 + .. + xn∗wn) **--logistic regression**

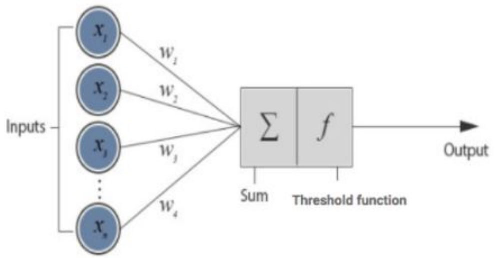
Logistic/sigmoid function

$$S(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}.$$

For sample 1:				
x	6	5	3	1
w	**0.3**	**0.2**	**-0.5**	**0**
Y = σ(sum(x * w)) = **σ(1.3) = 0.78**				

For sample 2:				
x	20	5	3	1
w	**0.3**	**0.2**	**-0.5**	**0**
Y = σ(sum(x * w)) = **σ(5.5) = 0.99**				

*Now, if we apply a **logistic/sigmoid function** on the output it will squeeze all the output between 0 and 1 :*



Y = Sigmoid(x1∗w1 + x2∗w2 + .. + xn∗wn) **--logistic regression**

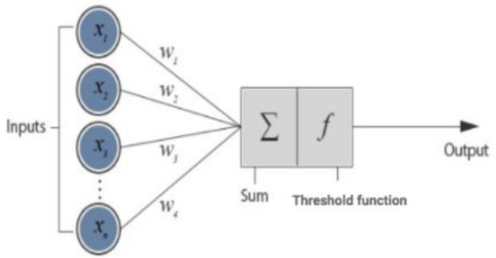
Logistic/sigmoid function

$$S(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}.$$

For sample 1:				
x	6	5	3	1
w	**0.3**	**0.2**	**-0.5**	**0**
Y = σ(sum(x * w)) = **σ(1.3) =** **0.78**				

For sample 2:				
x	20	5	3	1
w	**0.3**	**0.2**	**-0.5**	**0**
Y = σ(sum(x * w)) = **σ(5.5) =** **0.99**				

*Now, if we apply a threshold on the **logistic/sigmoid** output it will set the final output as 0 or 1 :*



Logistic/sigmoid function

$$S(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}.$$

$$f(t) = \begin{cases} 1 & \text{if } t > 0.6 \\ \mathbf{0} & \textit{otherwise} \end{cases}$$

For sample 1:				
x	6	5	3	1
w	**0.3**	**0.2**	**-0.5**	**0**
Y = **f** (**σ**(sum(x * w)))= f(**σ**(1.3)) = f (0.78) =1				

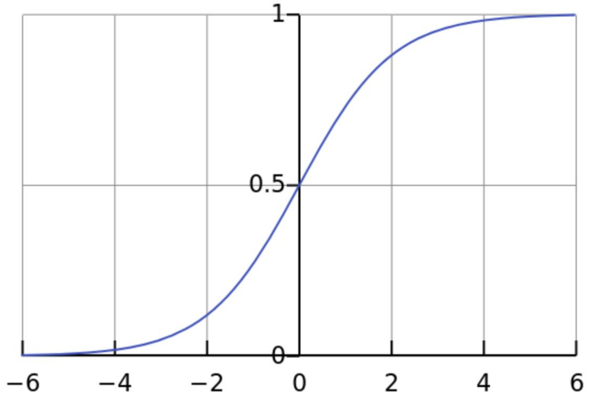
For sample 2:				
x	20	5	3	1
w	**0.3**	**0.2**	**-0.5**	**0**
Y = **f**(**σ**(sum(x * w))) = f (**σ**(5.5)) = f (0.99)=1				

Activation functions

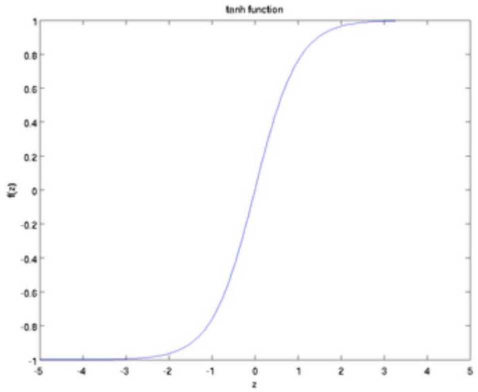
We use activation functions in neurons to induce nonlinearity in the neural nets so that it can learn complex functions.

Sigmoid Function

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



Tanh function



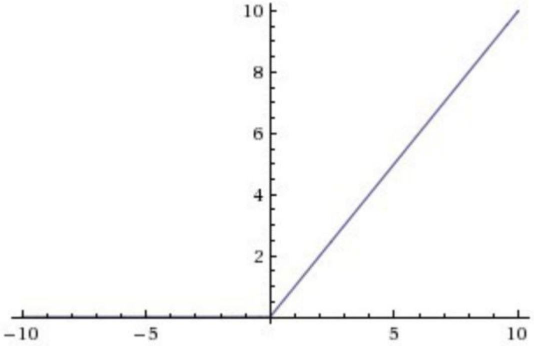
$$f(x) = tanh(x) = \frac{2}{1+e^{-2x}} - 1$$

Problem -

- Towards either end of the sigmoid/tanh function, the Y values tend to respond very less to changes in X.
- The gradient at that region is going to be small. It gives rise to a problem of “vanishing gradients”

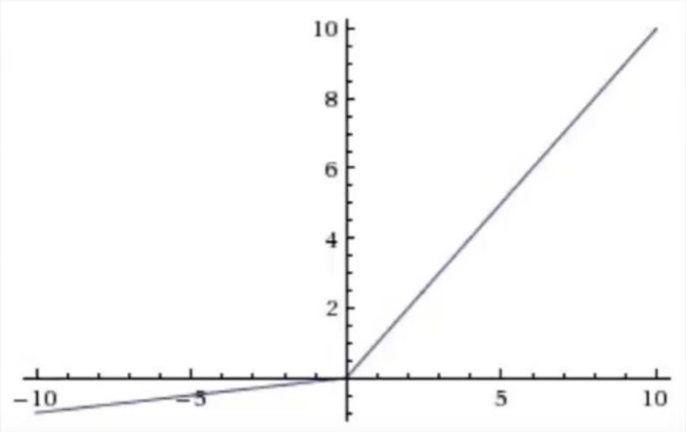
ReLU

$f(x) = \max(0, x)$



Leaky ReLU

$$f(x) = \begin{cases} 0.01x & \text{for } x < 0 \\ x & \text{for } x => 0 \end{cases}$$



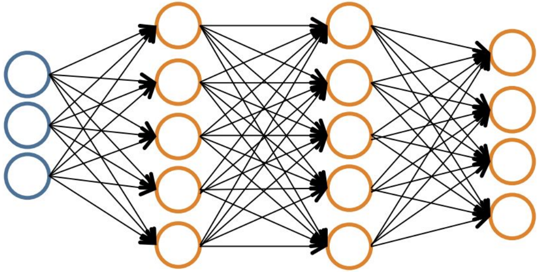
Multiple Output Units: One-vs-Rest

Pedestrian

Car

Motorcycle

Truck



$$h_\Theta(\mathbf{x}) \in \mathbb{R}^K$$

We want:

$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$
when pedestrian

$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$
when car

$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$
when motorcycle

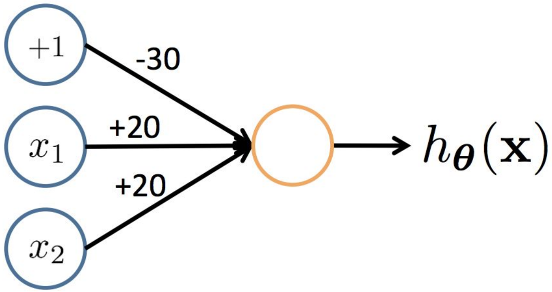
$$h_\Theta(\mathbf{x}) \approx \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$
when truck

Representing Boolean Functions

Simple example: AND

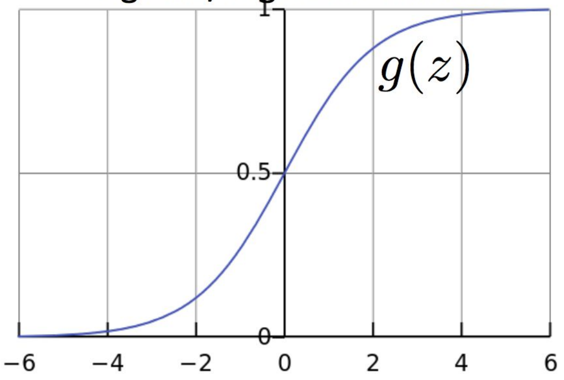
$x_1, x_2 \in \{0, 1\}$

$y = x_1 \text{ AND } x_2$



$$\mathrm{h}_{\Theta}(\mathbf{x}) = g(-30 + 20x_1 + 20x_2)$$

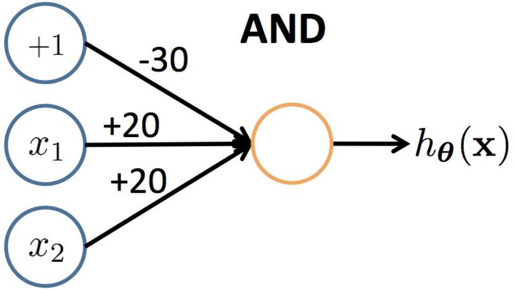
Logistic / Sigmoid Function

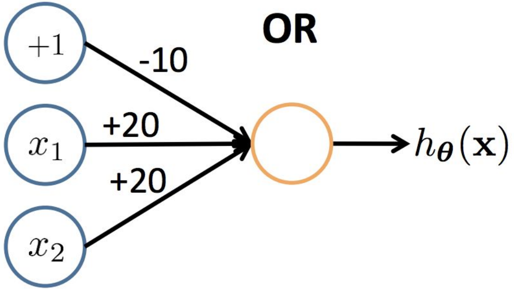


x_1	x_2	$\mathrm{h}_{\Theta}(\mathbf{x})$
0	0	$g(-30) \approx 0$
0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(10) \approx 1$

Based on slide and example by Andrew Ng

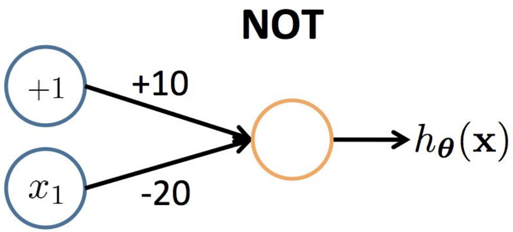
Representing Boolean Functions





A OR B

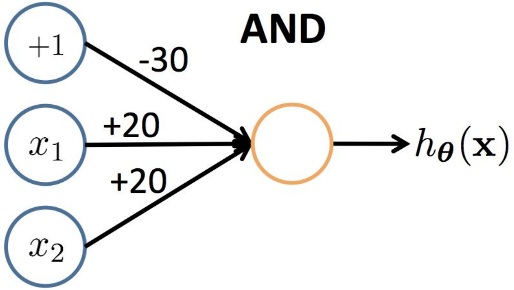
Input		Output
A	B	Y=A+B
0	0	0
0	1	1
1	0	1
1	1	1

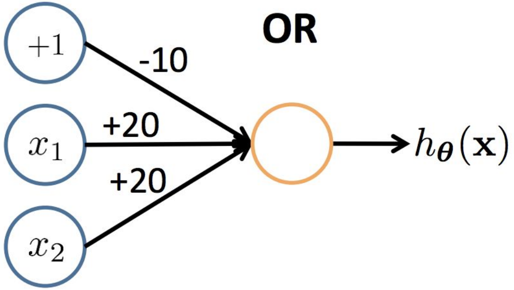


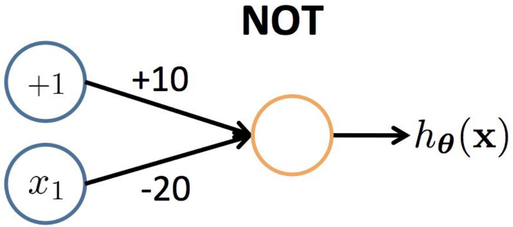
(NOT x_1) AND (NOT x_2)

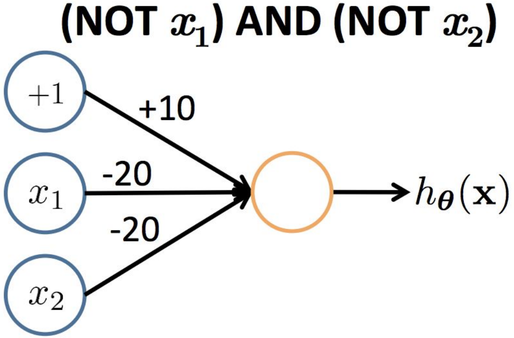
????

Representing Boolean Functions



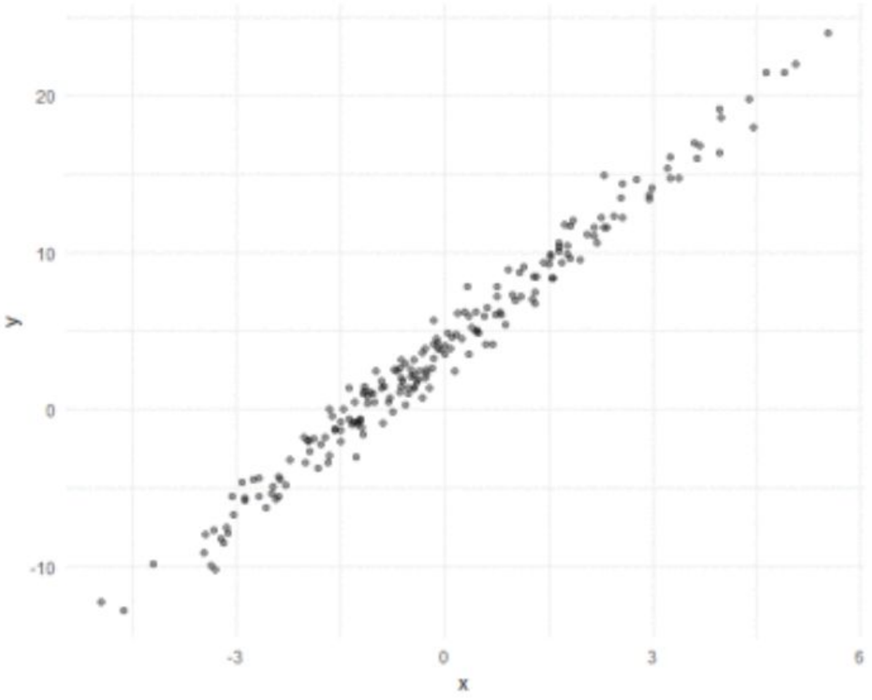




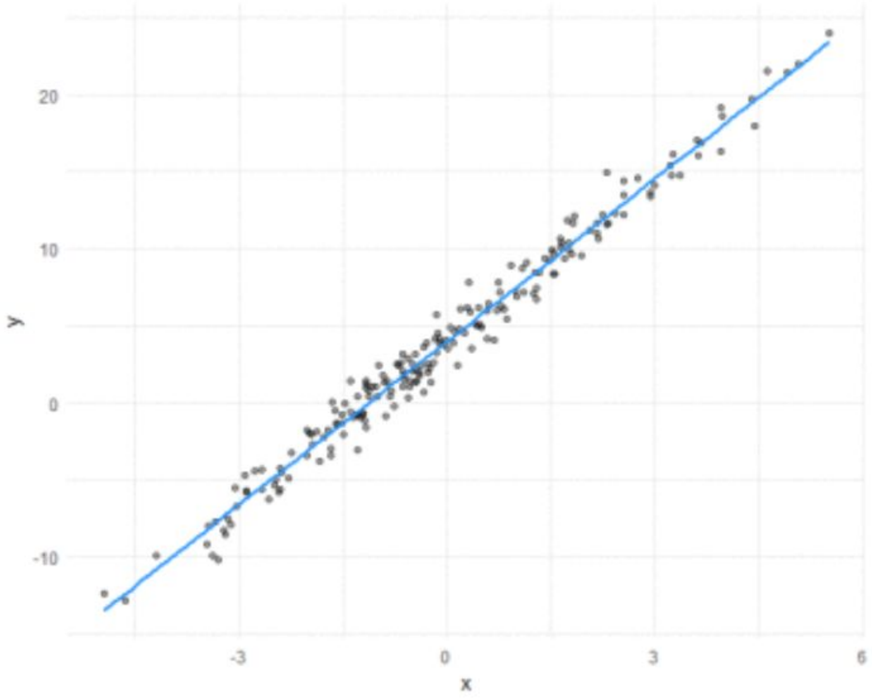


INPUTS		OUTPUT
X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	0

Neural Network Learning



20
10
0
-10
y
-3
0
3
6
x



20
10
0
-10
y
-3
0
3
6
x

What does the machine learn?

One question that people often have when getting started in ML is:

"What does the machine (i.e. the statistical model) actually learn?"

This will vary from model to model, but in simple terms the model learns a function f such that $f(X)$ maps to y. Put differently, the model learns how to take X (i.e. features, or, more traditionally, independent variable(s)) in orde to predict y (the target, response or more traditionally the dependent variable).

In the case of the simple linear regression (***y* ~ b0 + b1** * ***X*** where X is one column/variable) the model "learns" (read: estimates) two parameters;

- b0: the bias (or more traditionally the intercept); and,
- b1: the slope

Learning parameters: Cost functions

Remember that in ML, the focus is on **learning from data**. **cost function—it helps the learner to correct / change behaviour to minimize mistakes.**

In ML, cost functions are used to estimate how badly models are performing. Put simply, ***a cost function is a measure of how wrong the model is in terms of its ability to estimate the relationship between X and y.***

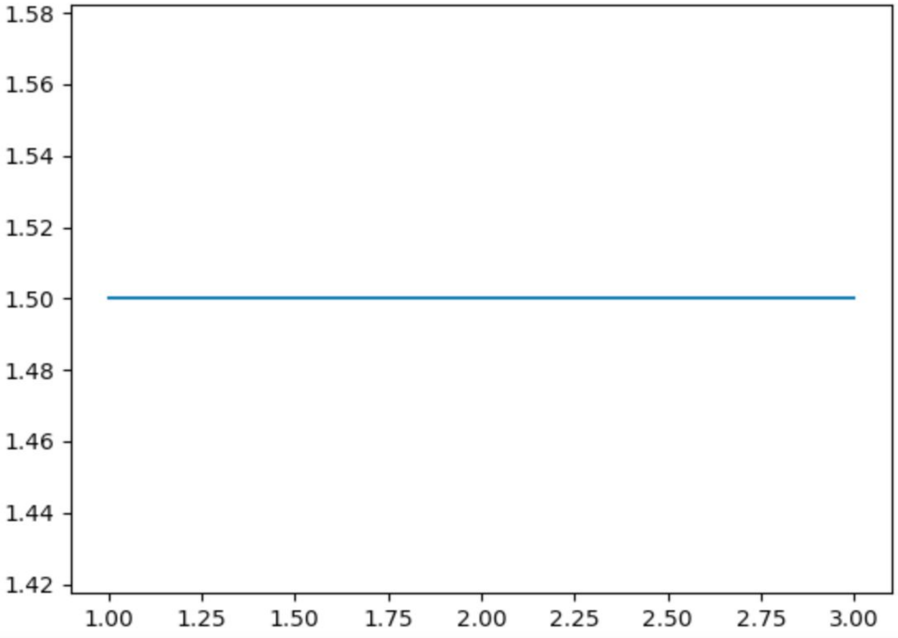
Example

$$h_\theta(x) = \theta_0 + \theta_1 x$$

The theta values are the *parameters*.

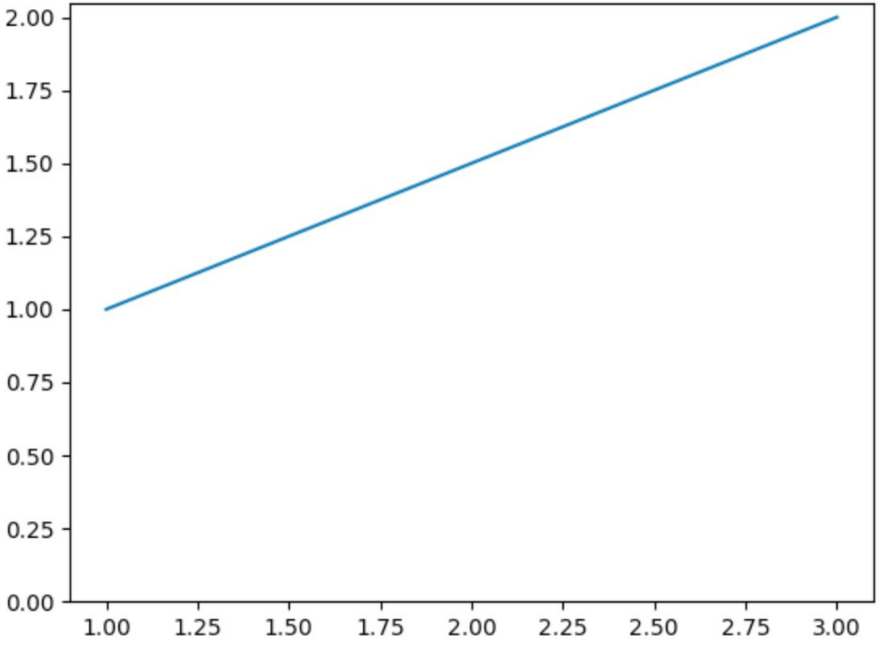
$$\theta_0 = 1.5$$
$$\theta_1 = 0$$

This yields h(x) = 1.5 + 0x. 0x means no slope, and y will always be the constant 1.5. This looks like:

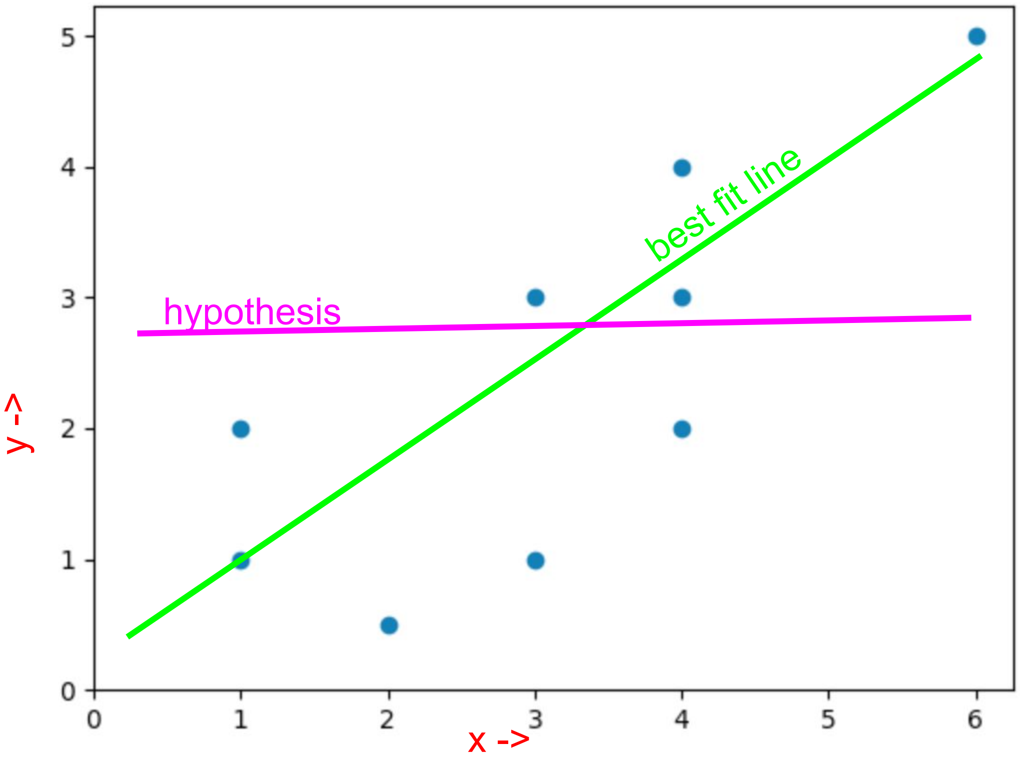


How about

$$\theta_0 = 1$$
$$\theta_1 = 0.5$$



```
x = [1, 1, 2, 3, 4, 3, 4, 6, 4]
y = [2, 1, 0.5, 1, 3, 3, 2, 5, 4]
```



You want to match your hypothesis to your best fit line

```
x = [1, 1, 2, 3, 4, 3, 4, 6, 4]
y = [2, 1, 0.5, 1, 3, 3, 2, 5, 4]
```

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2.$$

Mean Squared Error

```
x = [1, 1, 2, 3, 4, 3, 4, 6, 4]
y = [2, 1, 0.5, 1, 3, 3, 2, 5, 4]
```

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2.$$

$$= \tfrac{1}{m}\textstyle\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

Backpropagation

Given the cost function how do you update parameters?

1. **Calculate the partial derivative** $\frac{\partial}{\partial \theta_j} J(\theta)$ for all j

2. Form the **update rule** for every parameter:

$$\theta_{j,iter+1} := \theta_{j,iter} - \alpha \frac{\partial}{\partial \theta_j} J(\theta) = \theta_{j,iter} - \alpha/m \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

$$J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

Repeat {

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

(for every $j = 0, \ldots, n$)

}

Motivation: loss minimization

Optimization problem:

$$\min_{\mathbf{V},\mathbf{w}} \mathsf{TrainLoss}(\mathbf{V},\mathbf{w})$$

$$\mathsf{TrainLoss}(\mathbf{V},\mathbf{w}) = \frac{1}{|\mathcal{D}_{\mathsf{train}}|} \sum_{(x,y)\in\mathcal{D}_{\mathsf{train}}} \mathsf{Loss}(x,y,\mathbf{V},\mathbf{w})$$

$$\mathsf{Loss}(x,y,\mathbf{V},\mathbf{w}) = (y - f_{\mathbf{V},\mathbf{w}}(x))^2$$

$$f_{\mathbf{V},\mathbf{w}}(x) = \sum_{j=1}^{k} w_j \sigma(\mathbf{v}_j \cdot \phi(x))$$

Goal: compute gradient

$$\nabla_{\mathbf{V},\mathbf{w}} \mathsf{TrainLoss}(\mathbf{V},\mathbf{w})$$

Source - Stanford CS221 Liang

Approach

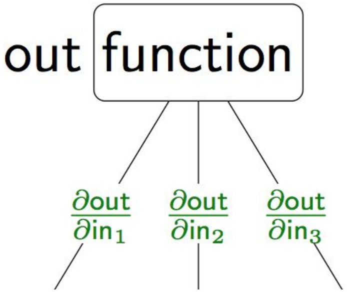
Mathematically: just grind through the chain rule

Next: visualize the computation using a computation graph

Advantages:

- Avoid long equations
- Reveal structure of computations (modularity, efficiency, dependencies)

Functions as boxes



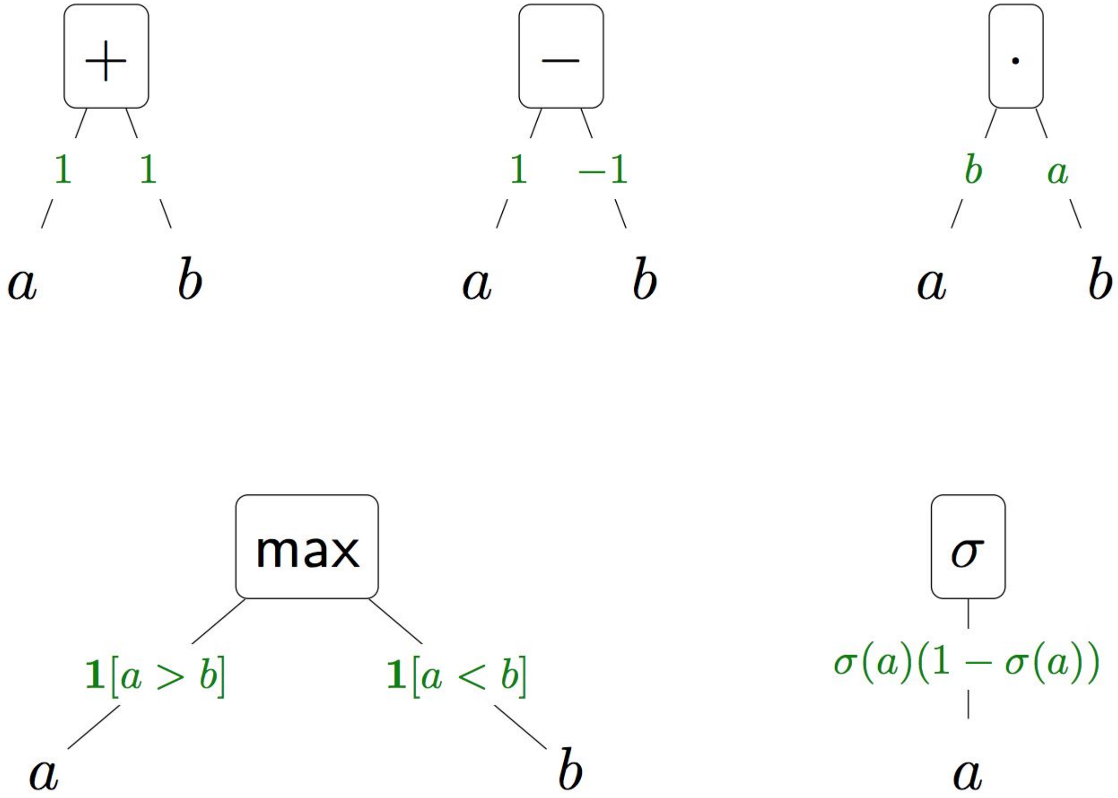
$$2\text{in}_1 + \text{in}_2 * \text{in}_3 = \text{out}$$

Partial derivatives (gradients): how much does the output change if an input changes?

Example:

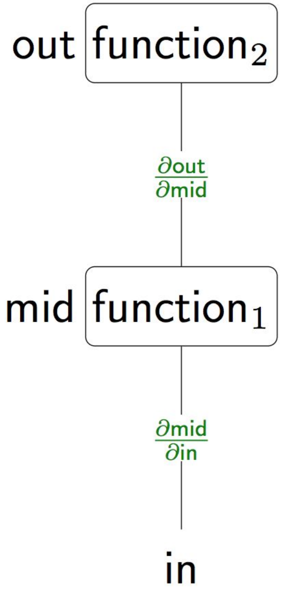
$$2\text{in}_1 + (\text{in}_2 + \epsilon)\text{in}_3 = \text{out} + \text{in}_3\epsilon$$

Basic building blocks



Source - Stanford CS221 Liang

Composing functions

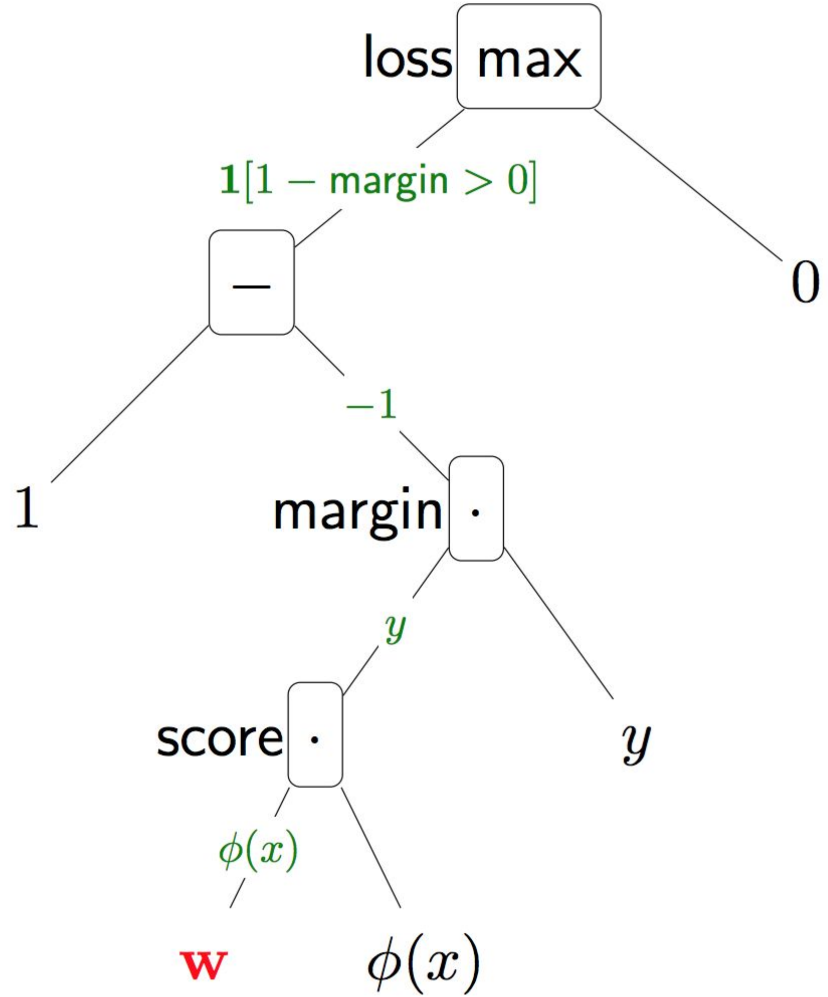


Chain rule:

$$\frac{\partial \text{out}}{\partial \text{in}} = \frac{\partial \text{out}}{\partial \text{mid}} \frac{\partial \text{mid}}{\partial \text{in}}$$

Example

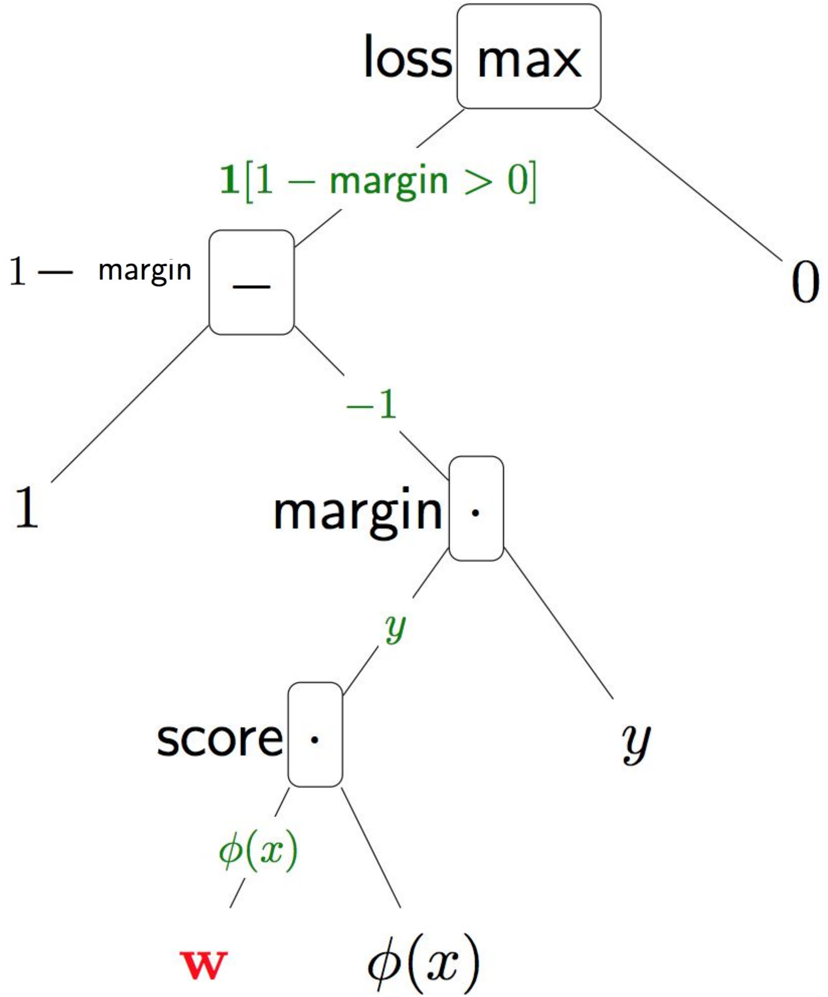
$$\text{Loss}(x, y, \mathbf{w}) = \max\{1 - \mathbf{w} \cdot \phi(x) y, 0\}$$



Source - Stanford CS221 Liang

Example

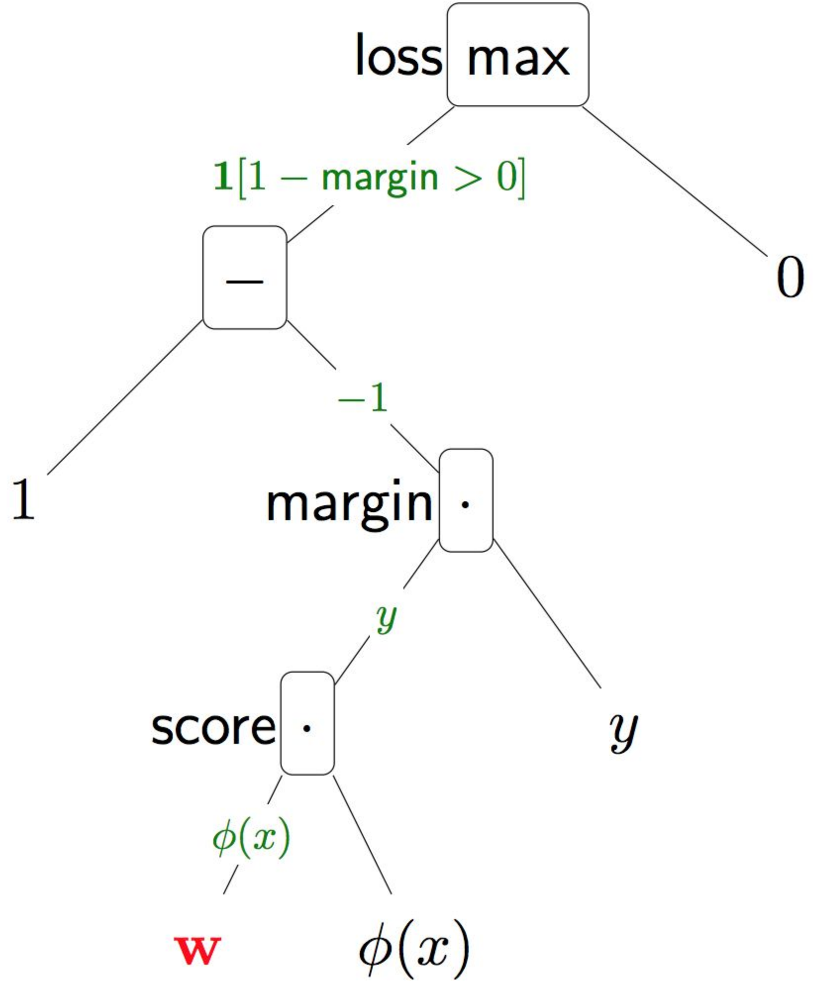
$$\text{Loss}(x, y, \mathbf{w}) = \max\{1 - \mathbf{w} \cdot \phi(x)y, 0\}$$



Source - Stanford CS221 Liang

Example

$$\text{Loss}(x, y, \mathbf{w}) = \max\{1 - \mathbf{w} \cdot \phi(x)y, 0\}$$



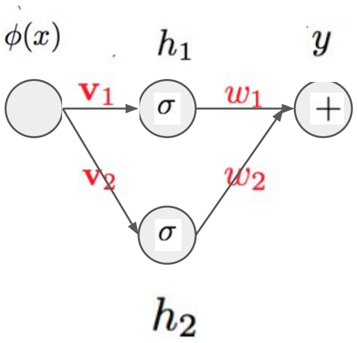
Gradient: multiply the edges

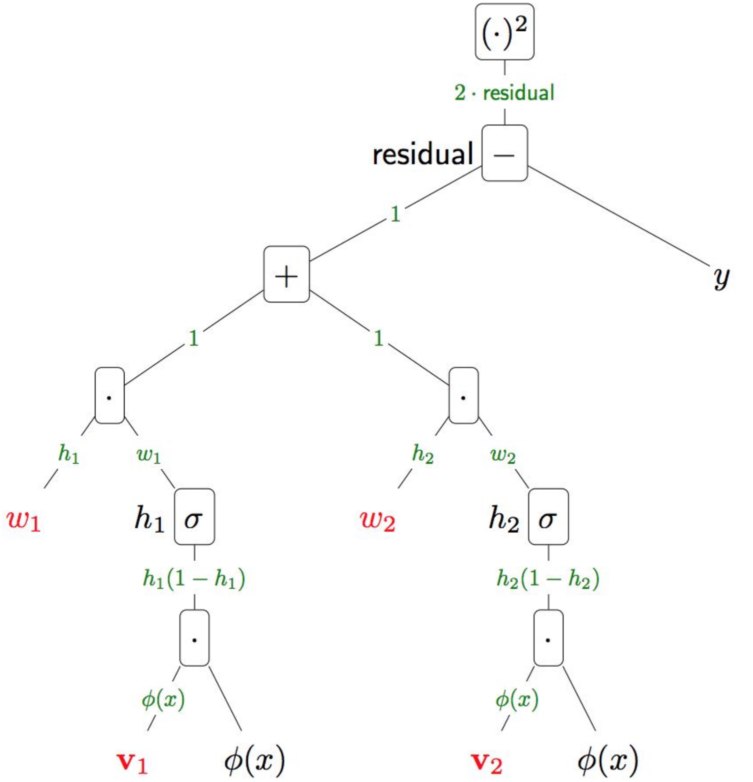
$$-\mathbf{1}[\text{margin} < 1]\phi(x)y$$

Neural Network

$$\text{Loss}(x, y, \mathbf{w}) = \left(\sum_{j=1}^{k} w_j \sigma(\mathbf{v}_j \cdot \phi(x)) - y\right)^2$$

- 2 layer Neural Network





Source - Stanford CS221 Liang

Other Types of Optimizers

Apart from gradient descent there are other optimizers widely used-

- **Adagrad** - Adagrad adapts the learning rate specifically to individual features: that means that some of the weights in your dataset will have different learning rates than others.
- **RMSProp** - RMSProp is Root Mean Square Propagation. It was devised by Geoffrey Hinton. RMSProp tries to resolve Adagrad's radically diminishing learning rates by using a moving average of the squared gradient.
- **Adam** - Adam stands for adaptive moment estimation, and is another way of using past gradients to calculate current gradients. A combination of RMSProp and Adagrad. Widely used in Computer Vision tasks.

Training a Neural Network

1. Pick a network architecture

   # of input units = # of features

   # of output units = # of classes

2. Randomly initialize weights
3. Implement forward propagation to get h(x) for any x
4. Compute cost function
5. Use gradient descent/optimizer with backprop to fit the network

Resources-

https://www.youtube.com/watch?v=aircAruvnKk

https://www.youtube.com/watch?v=Ilg3gGewQ5U

https://towardsdatascience.com/activation-functions-and-its-types-which-is-better-a9a5310cc8f

http://neuralnetworksanddeeplearning.com/

https://playground.tensorflow.org

Stanford CS229 course