



NEURAL NETWORKS

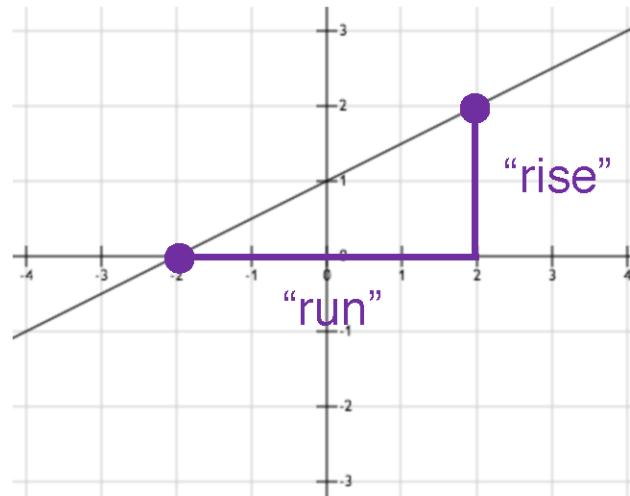
Ravi Kiran
CVIT, IIIT Hyderabad

15.02.2019

Rate of Change

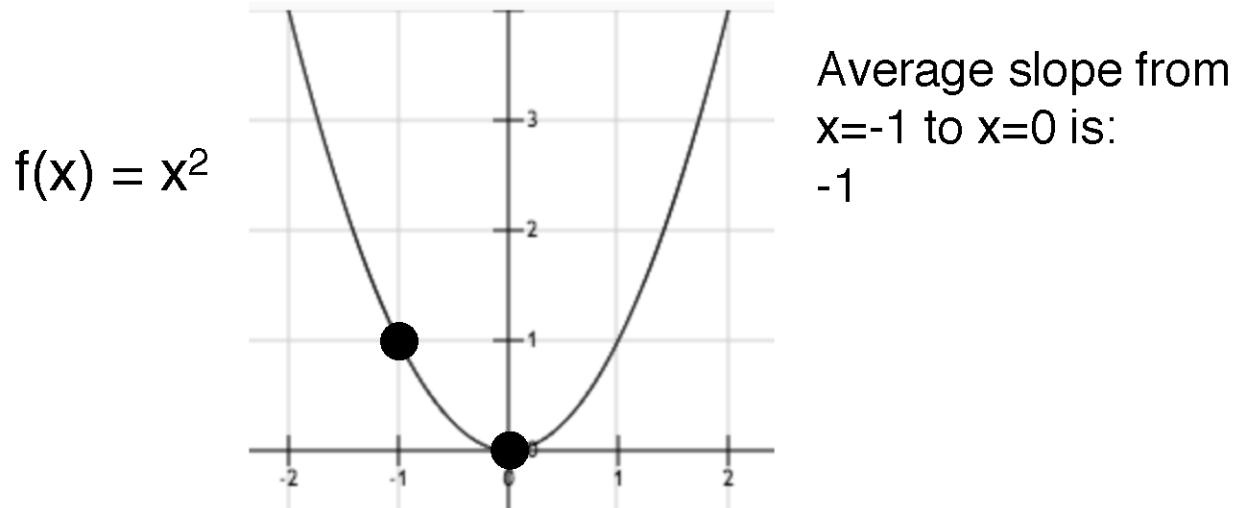
The slope of a line is also called the **rate of change** of the line.

$$y = \frac{1}{2}x + 1$$



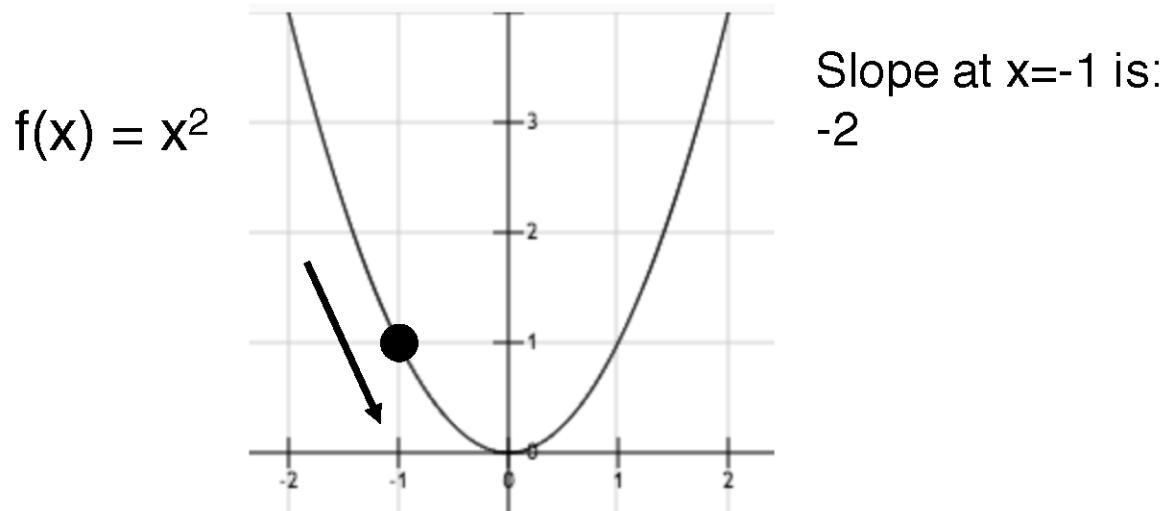
Rate of Change

For nonlinear functions, the “rise over run” formula gives you the average rate of change between two points



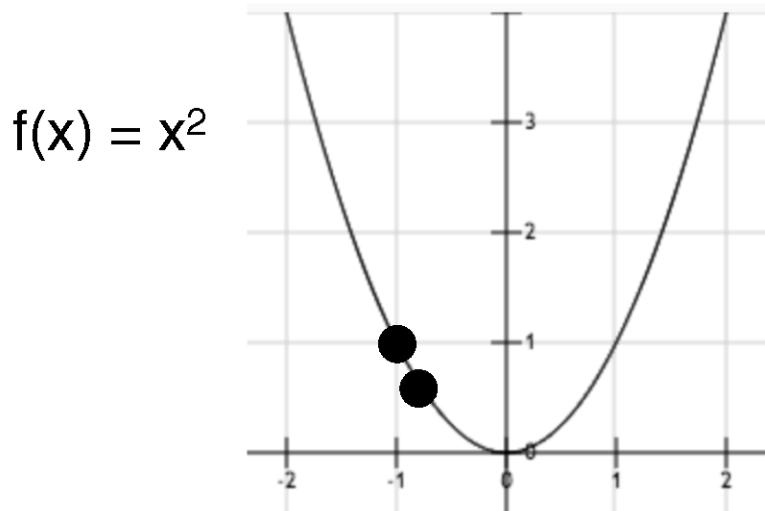
Rate of Change

There is also a concept of rate of change at individual points (rather than two points)



Rate of Change

The slope at a point is called the **derivative** at that point



$$f(x) = x^2$$

Intuition:
Measure the slope
between two points
that are really close
together

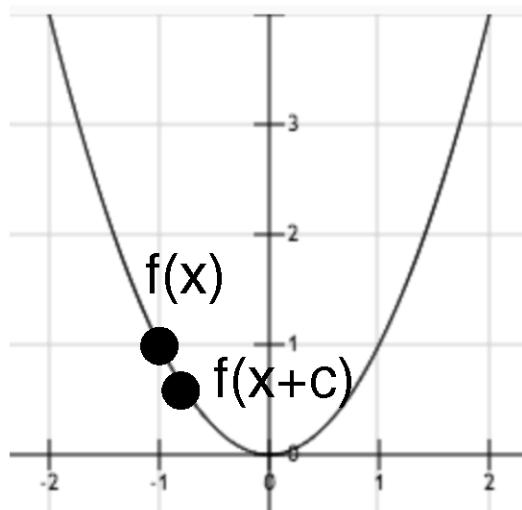
Rate of Change

The slope at a point is called the **derivative** at that point

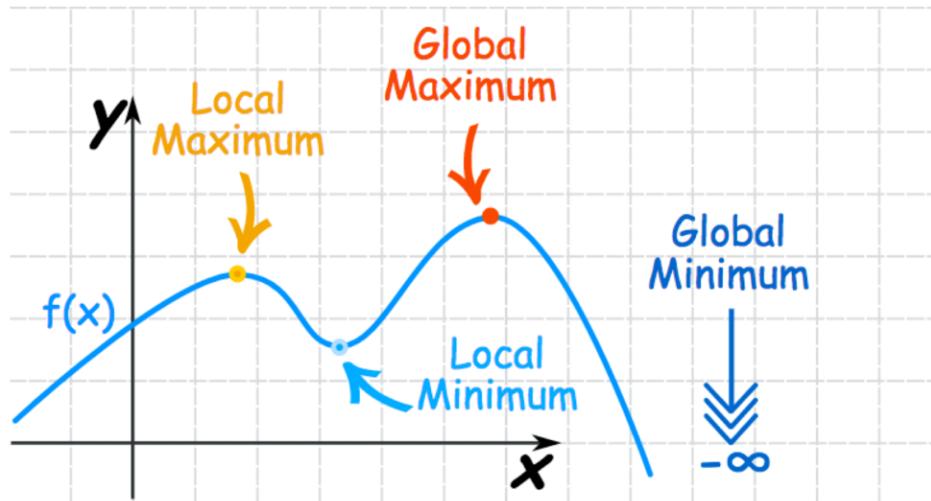
Intuition: Measure the slope between two points that are really close together

$$\frac{f(x + c) - f(x)}{c}$$

Limit as c goes to zero



Maxima and Minima



From: <https://www.mathsisfun.com/algebra/functions-maxima-minima.html>

Derivatives

The derivative of $f(x) = x^2$ is $2x$

Other ways of writing this:

$$f'(x) = 2x$$

$$\frac{d}{dx} [x^2] = 2x$$

$$\frac{df}{dx} = 2x$$

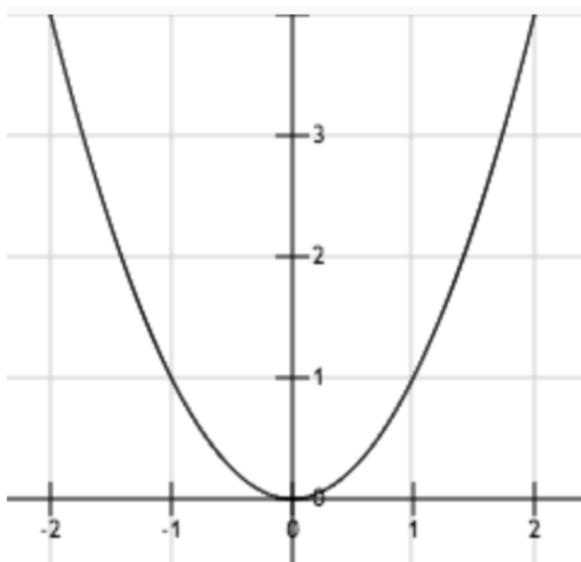
The derivative is also a function! It depends on the value of x .

- The rate of change is different at different points

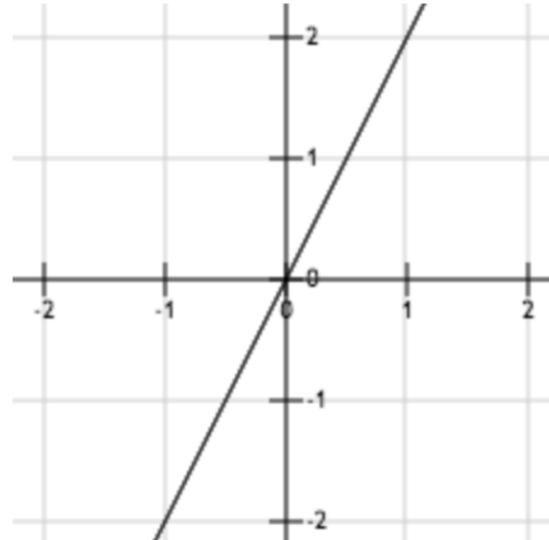
Derivatives

The derivative of $f(x) = x^2$ is $2x$

$f(x)$



$f'(x)$



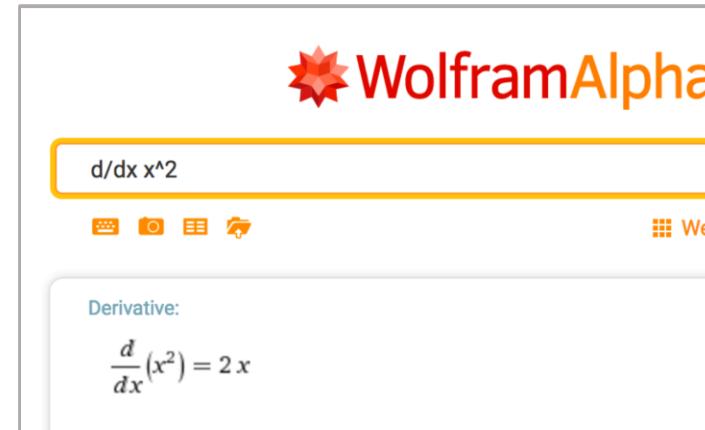
Derivatives

How to calculate a derivative?

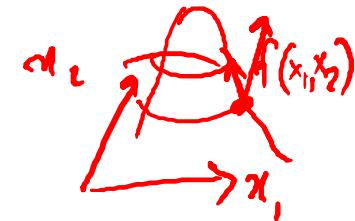
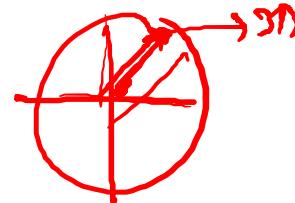
- Not going to do it in this class.

Some software can do it for you.

- [Wolfram Alpha](#)



Derivatives



What if a function has multiple arguments?

$$\text{Ex: } f(x_1, x_2) = 3x_1 + 5x_2$$

$$\underline{df/dx_1} = 3 + 5x_2 \quad \text{The derivative "with respect to" } x_1$$

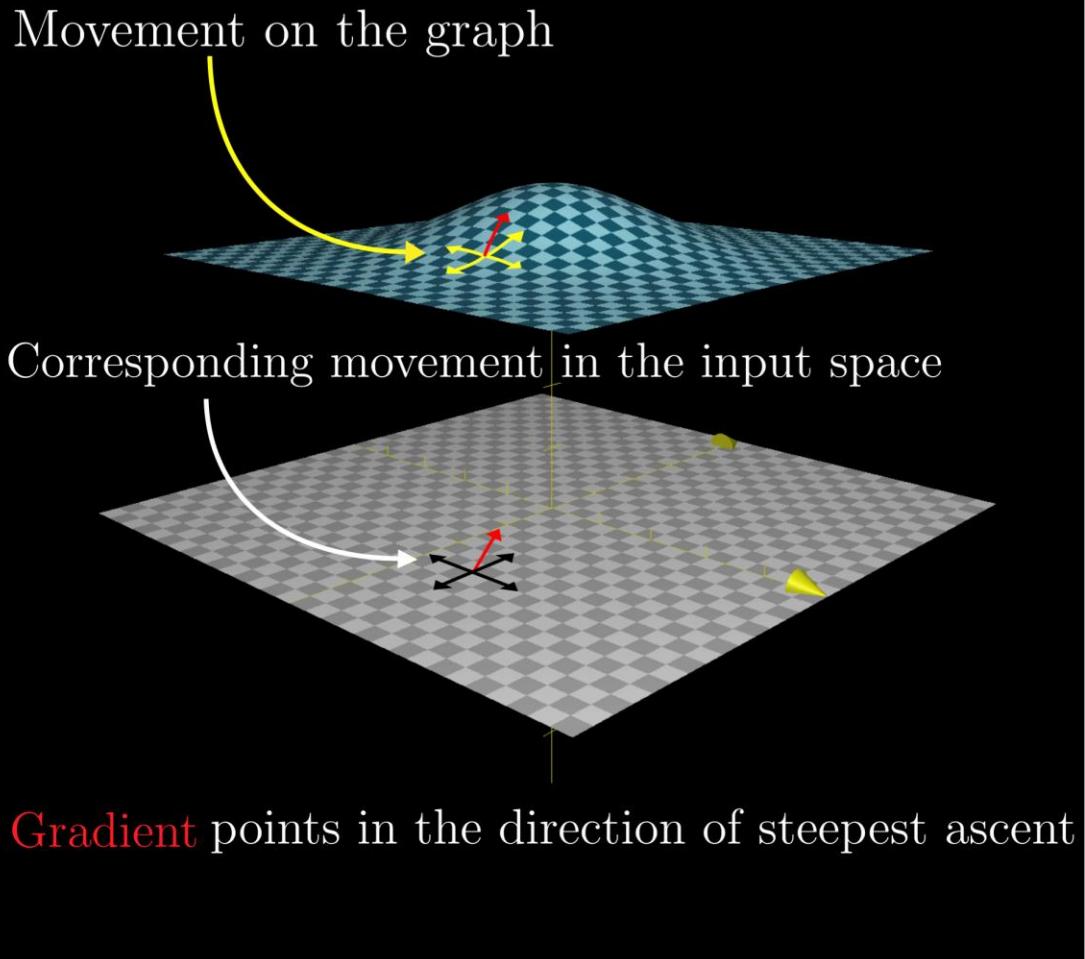
$$\underline{df/dx_2} = 3x_1 + 5 \quad \text{The derivative "with respect to" } x_2$$

These two functions are called **partial derivatives**.

The vector of all partial derivatives for a function f is called the **gradient** of the function:

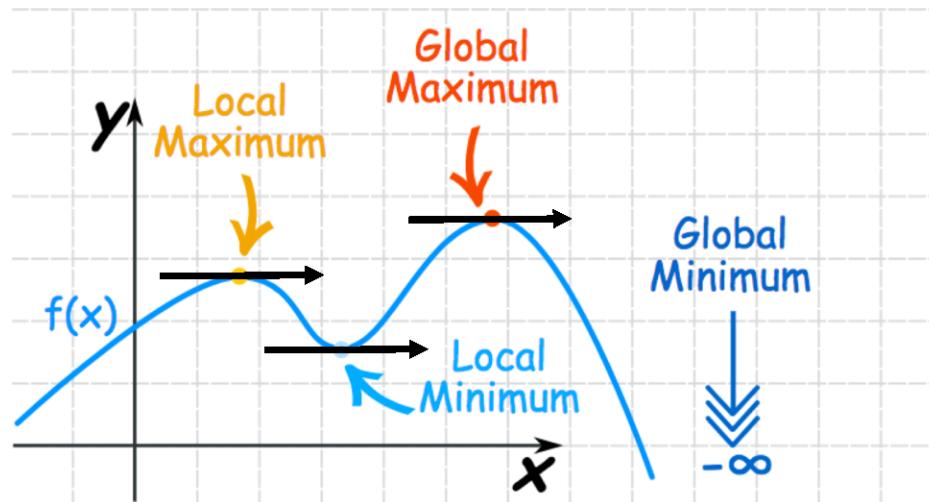
$$\nabla f(x_1, x_2) = \langle df/dx_1, df/dx_2 \rangle \quad [0.2 \quad 0.7]$$

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \\ \vdots \end{bmatrix}$$



Finding Minima

The derivative is *zero* at any local maximum or minimum.



Finding Minima

The derivative is *zero* at any local maximum or minimum.

One way to find a minimum: set $f'(x)=0$ and solve for x .

- For most functions, there isn't a way to solve this.
- Instead: algorithmically search different values of x until you find one that results in a gradient near 0.

Finding Minima

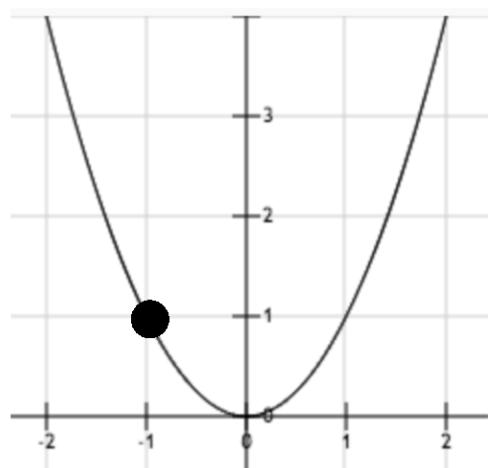
If the derivative is positive, the function is **increasing**.

- Don't move in that direction, because you'll be moving away from a trough.

If the derivative is negative, the function is **decreasing**.

- Keep going, since you're getting closer to a trough

Finding Minima



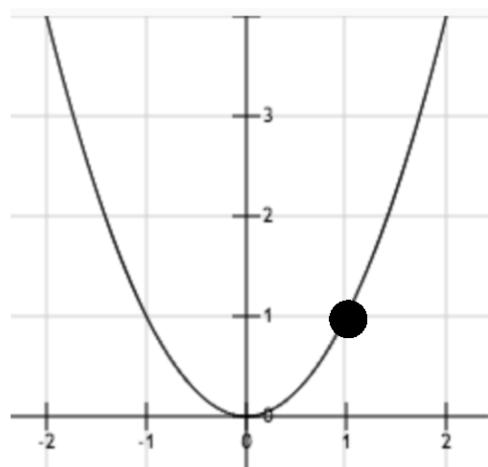
$$f'(-1) = -2$$

At $x=-1$, the function is decreasing as x gets larger. This is what we want, so let's make x larger.

Increase x by the size of the gradient:

$$-1 + 2 = 1$$

Finding Minima



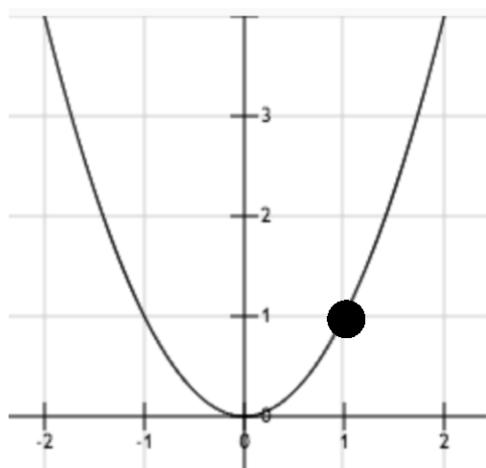
$$f'(-1) = -2$$

At $x=-1$, the function is decreasing as x gets larger. This is what we want, so let's make x larger.

Increase x by the size of the gradient:

$$-1 + 2 = 1$$

Finding Minima

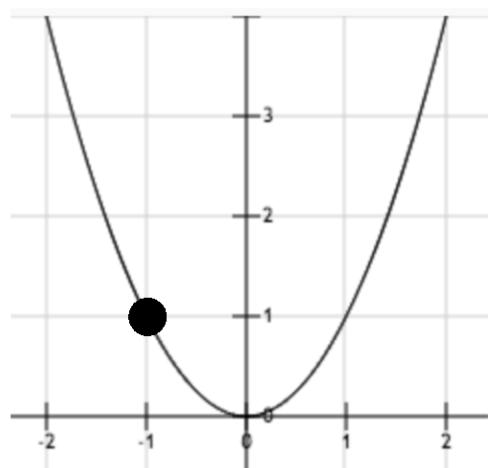


$$f'(1) = 2$$

At $x=1$, the function is increasing as x gets larger. This is not what we want, so let's make x smaller. Decrease x by the size of the gradient:

$$1 - 2 = -1$$

Finding Minima

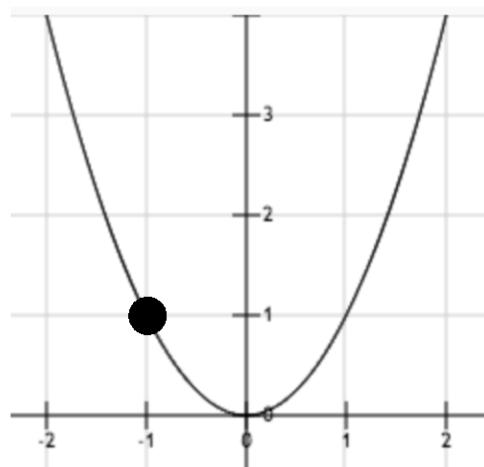


$$f'(1) = 2$$

At $x=1$, the function is increasing as x gets larger. This is not what we want, so let's make x smaller. Decrease x by the size of the gradient:

$$1 - 2 = -1$$

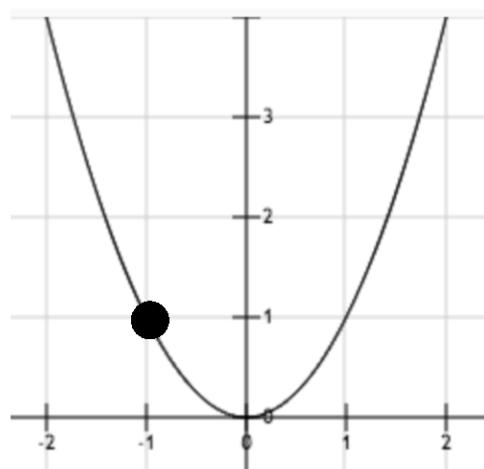
Finding Minima



We will keep jumping between the same two points this way.

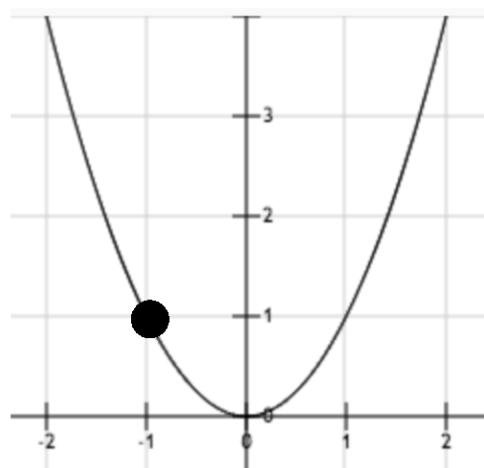
We can fix this by using a learning rate or step size.

Finding Minima



$$f'(-1) = -2$$
$$x + \eta =$$

Finding Minima

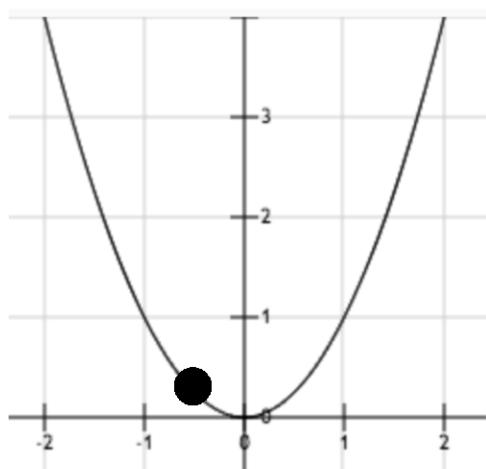


$$f'(-1) = -2$$

$$x += 2\eta =$$

Let's use $\eta = 0.25$.

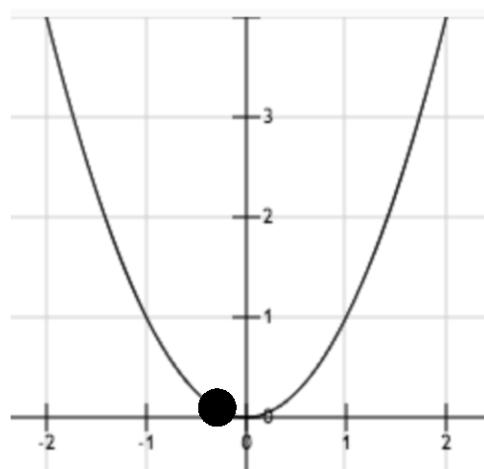
Finding Minima



$$f'(-1) = -2$$

$$x = -1 + 2(.25) = -0.5$$

Finding Minima



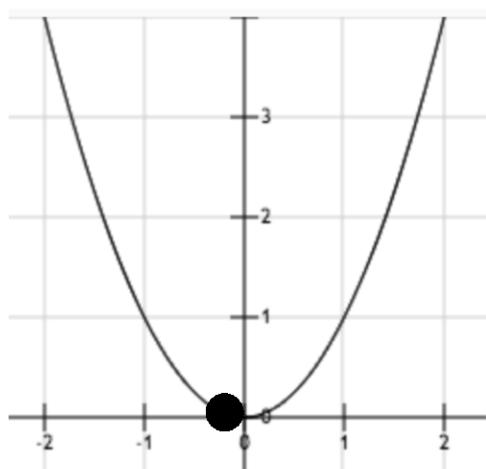
$$f'(-1) = -2$$

$$x = -1 + 2(.25) = -0.5$$

$$f'(-0.5) = -1$$

$$x = -0.5 + 1(.25) = -0.25$$

Finding Minima



$$f'(-1) = -2$$

$$x = -1 + 2(.25) = -0.5$$

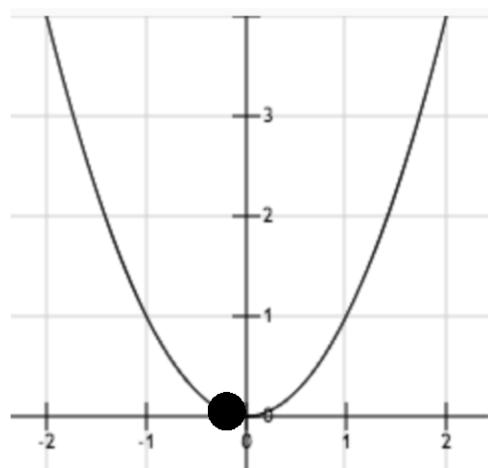
$$f'(-0.5) = -1$$

$$x = -0.5 + 1(.25) = -0.25$$

$$f'(-0.25) = -0.5$$

$$x = -0.25 + 0.5(.25) = -0.125$$

Finding Minima



$$f'(-1) = -2$$

$$x = -1 + 2(.25) = -0.5$$

$$f'(-0.5) = -1$$

$$x = -0.5 + 1(.25) = -0.25$$

$$f'(-0.25) = -0.5$$

$$x = -0.25 + 0.5(.25) = -0.125$$

Eventually we'll reach $x=0$.

Gradient Descent

1. Initialize the parameters \mathbf{w} to some guess
(usually all zeros, or random values)
2. Update the parameters:
$$\mathbf{w} = \mathbf{w} - \eta \nabla L(\mathbf{w})$$
3. Update the learning rate η
4. Repeat steps 2-3 until $\nabla L(\mathbf{w})$ is close to zero.

Gradient Descent

Gradient descent is guaranteed to eventually find a *local* minimum if:

- the learning rate is decreased appropriately;
- a finite local minimum exists (i.e., the function doesn't keep decreasing forever).

Gradient Ascent

What if we want to find a local *maximum*?

Same idea, but the update rule moves the parameters in the opposite direction:

$$\mathbf{w} = \mathbf{w} + \eta \nabla L(\mathbf{w})$$

Learning Rate

In order to guarantee that the algorithm will converge, the learning rate should decrease over time. Here is a general formula.

At iteration t :

$$\eta_t = c_1 / (t^a + c_2),$$

where $0.5 < a < 2$

$$c_1 > 0$$

$$c_2 \geq 0$$

Stopping Criteria

For most functions, you probably won't get the gradient to be exactly equal to **0** in a reasonable amount of time.

Once the gradient is sufficiently close to **0**, stop trying to minimize further.

How do we measure how close a gradient is to **0**?

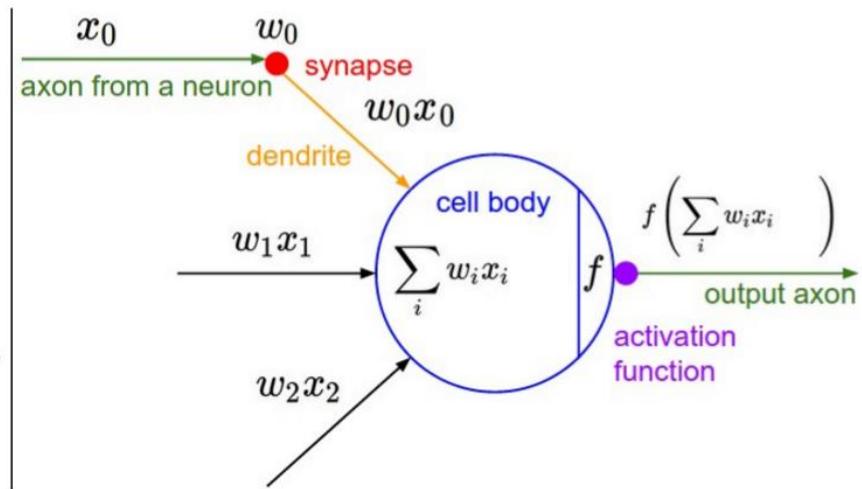
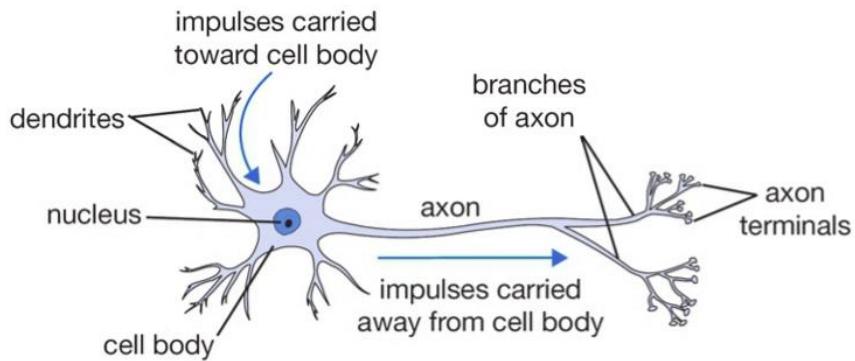
Stopping Criteria

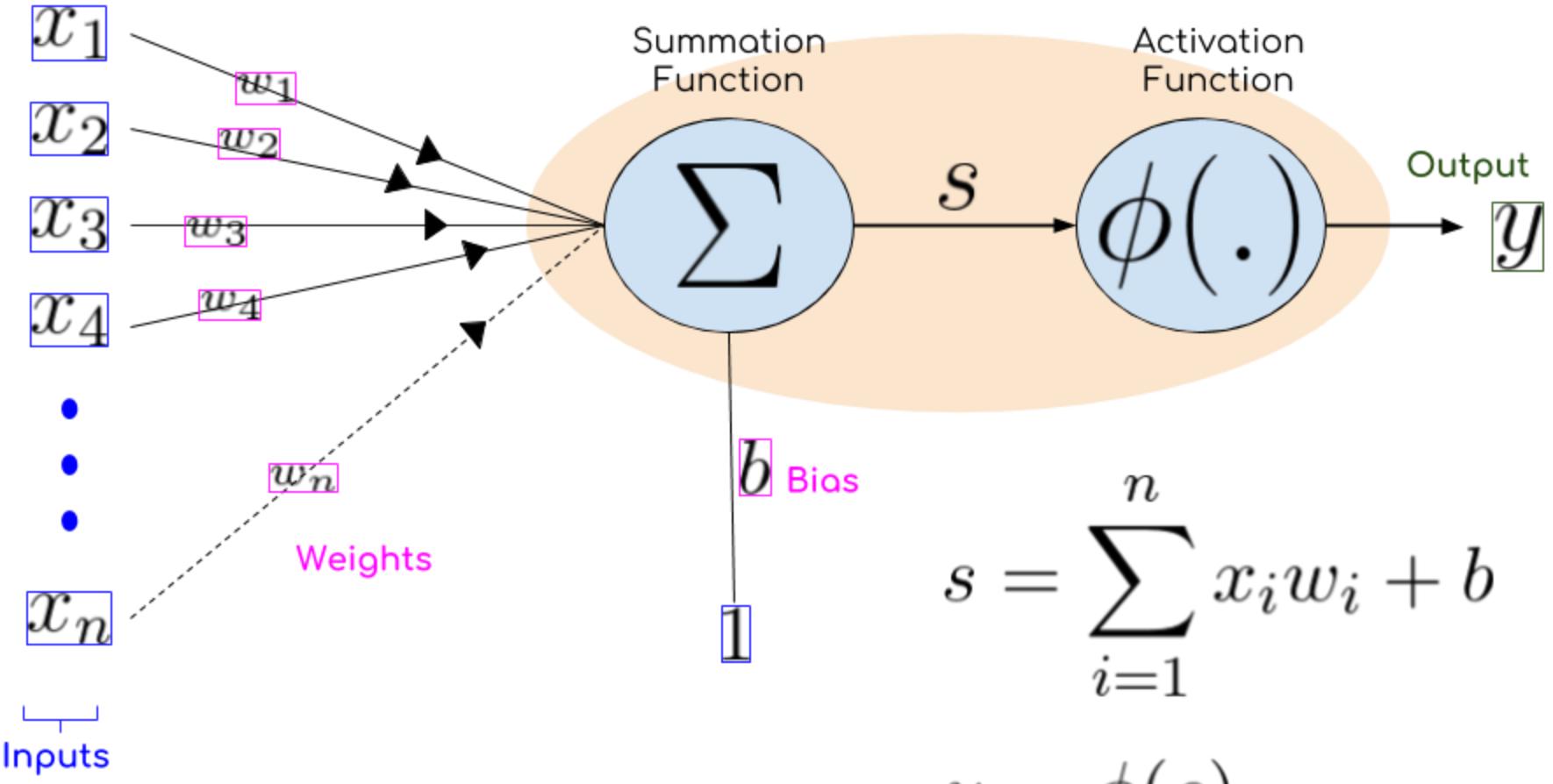
Stop when the norm of the gradient is below some threshold, θ :

$$\|\nabla L(\mathbf{w})\| < \theta$$

Common values of θ are around .01, but if it is taking too long, you can make the threshold larger.

BASIC BUILDING BLOCK :: A NEURON

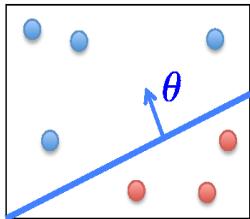




Linear Classifiers

- **Linear classifiers:** represent decision boundary by hyperplane

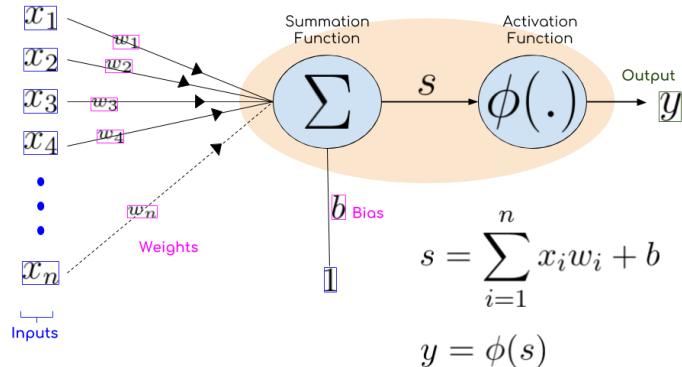
$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_d \end{bmatrix} \quad x^\top = [1 \quad x_1 \quad \dots \quad x_d]$$



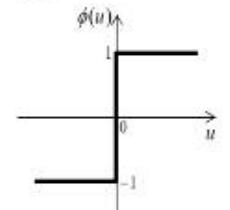
$$h(x) = \text{sign}(\theta^\top x) \text{ where } \text{sign}(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ -1 & \text{if } z < 0 \end{cases}$$

– Note that: $\theta^\top x > 0 \implies y = +1$

$\theta^\top x < 0 \implies y = -1$



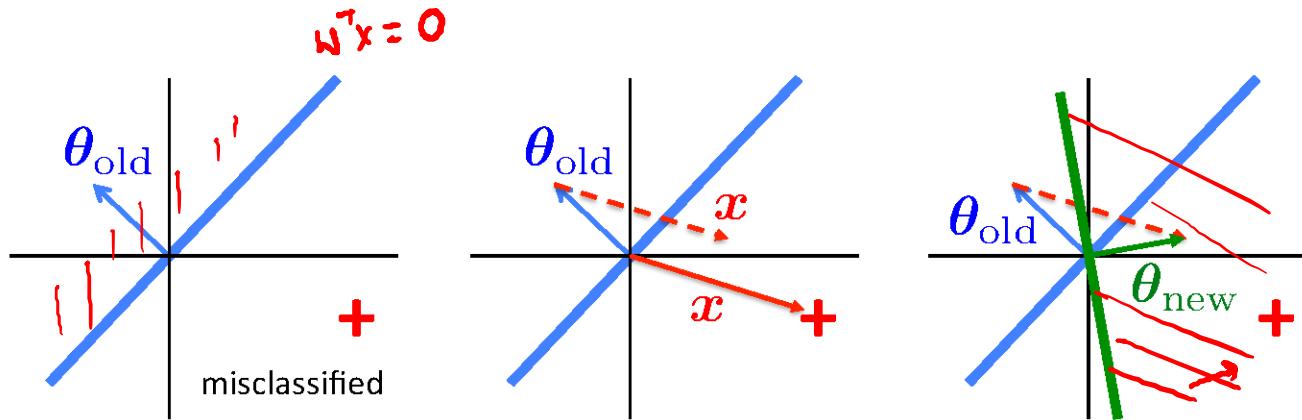
sign function



$$\phi_{\text{sign}}(u) = \begin{cases} 1 & \text{if } u \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

Perceptron Rule: If $\mathbf{x}^{(i)}$ is misclassified, do $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + y^{(i)} \mathbf{x}^{(i)}$

Perceptron Rule: If $\mathbf{x}^{(i)}$ is misclassified, do $\theta \leftarrow \theta + y^{(i)}\mathbf{x}^{(i)}$

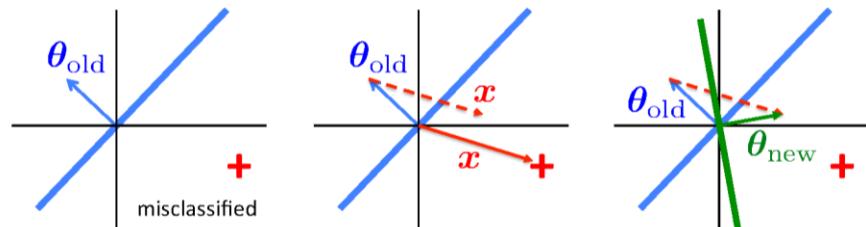


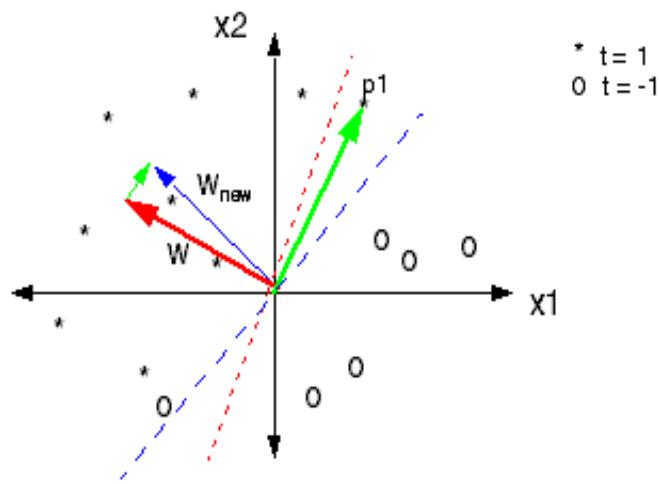
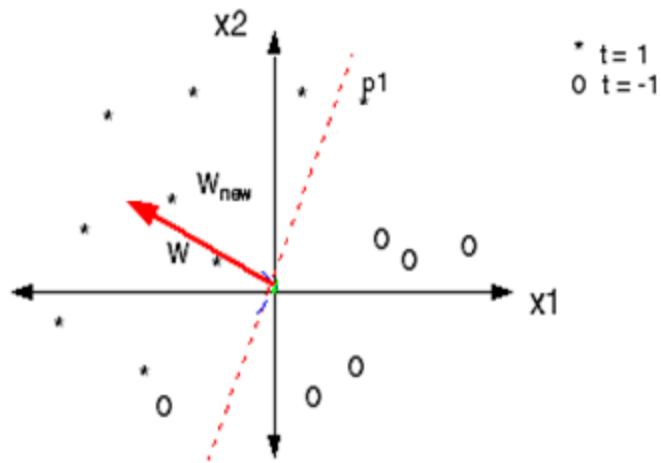
Why the Perceptron Update Works

- Consider the misclassified example ($y = +1$)
 - Perceptron wrongly thinks that $\theta_{\text{old}}^T x < 0$
- Update:
$$\theta_{\text{new}} = \theta_{\text{old}} + yx = \theta_{\text{old}} + x \quad (\text{since } y = +1)$$
- Note that

$$\begin{aligned}\theta_{\text{new}}^T x &= (\theta_{\text{old}} + x)^T x \\ &= \theta_{\text{old}}^T x + \underbrace{x^T x}_{\|x\|_2^2 > 0}\end{aligned}$$

- Therefore, $\theta_{\text{new}}^T x$ is less negative than $\theta_{\text{old}}^T x$
 - So, we are making ourselves more correct on this example!





Perceptron Rule: If $\mathbf{x}^{(i)}$ is misclassified, do $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + y^{(i)} \mathbf{x}^{(i)}$

The Perceptron Cost Function

- Prediction is correct if $y^{(i)} \mathbf{x}^{(i)} \boldsymbol{\theta} > 0$

Perceptron Rule: If $\mathbf{x}^{(i)}$ is misclassified, do $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + y^{(i)} \mathbf{x}^{(i)}$

The Perceptron Cost Function

- Prediction is correct if $y^{(i)} \mathbf{x}^{(i)} \boldsymbol{\theta} > 0$

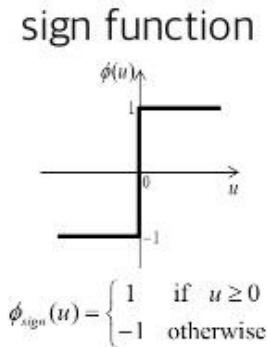
-

$0/1$ loss

$$J_{0/1}(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^n \ell(sign(x^{(i)} \boldsymbol{\theta}), y^{(i)})$$

C.G.T

where $\ell()$ is 0 if the prediction is correct, 1 otherwise



Perceptron Rule: If $\mathbf{x}^{(i)}$ is misclassified, do $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + y^{(i)} \mathbf{x}^{(i)}$

The Perceptron Cost Function

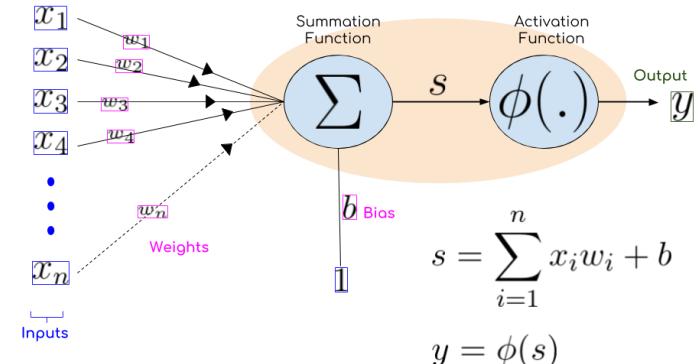
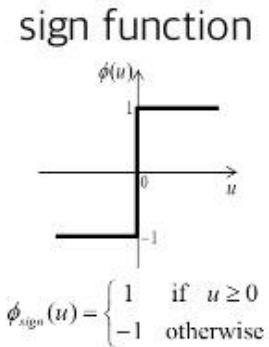
- Prediction is correct if $y^{(i)} \mathbf{x}^{(i)} \boldsymbol{\theta} > 0$

- 0/1 loss

$$J_{0/1}(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^n \ell(\text{sign}(\mathbf{x}^{(i)} \boldsymbol{\theta}), y^{(i)})$$

where $\ell()$ is 0 if the prediction is correct, 1 otherwise

$$\begin{aligned} J(\boldsymbol{\theta}) & \quad \frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \\ & \quad (f + g)' = f' + g' \\ & \quad \frac{\partial \ell(\text{sign}(\mathbf{x}^{(i)} \boldsymbol{\theta}, y^{(i)}))}{\partial \boldsymbol{\theta}} \end{aligned}$$



Perceptron Rule: If $\mathbf{x}^{(i)}$ is misclassified, do $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + y^{(i)} \mathbf{x}^{(i)}$

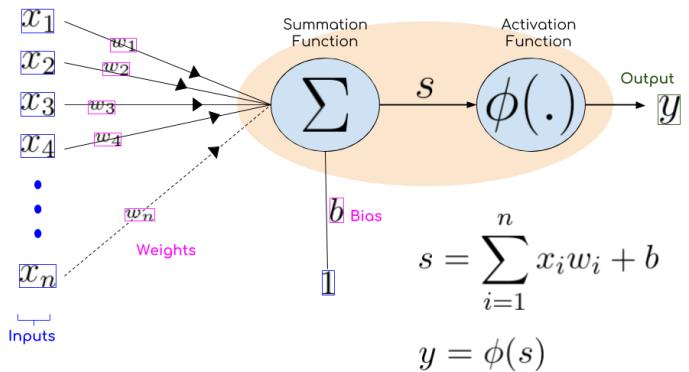
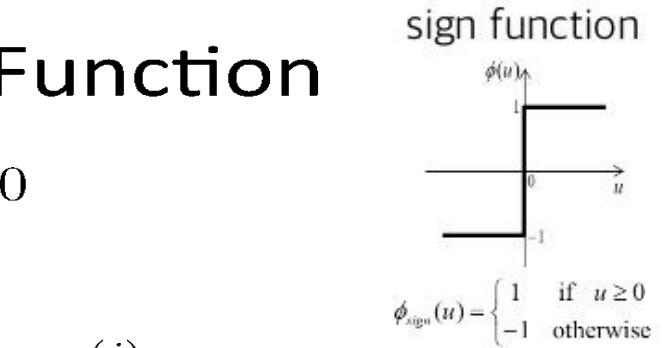
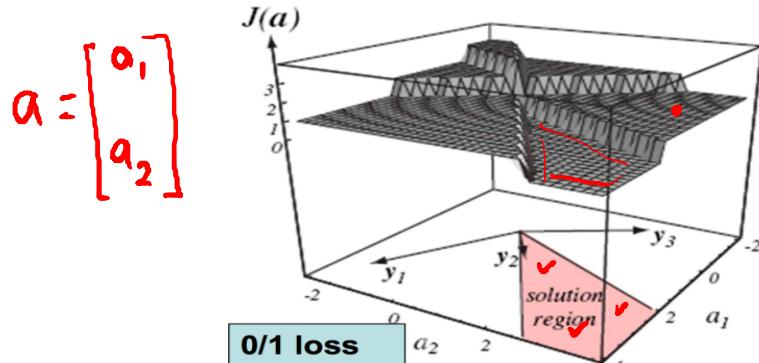
The Perceptron Cost Function

- Prediction is correct if $y^{(i)} \mathbf{x}^{(i)} \boldsymbol{\theta} > 0$

- 0/1 loss**

$$\underbrace{J_{0/1}(\boldsymbol{\theta})}_{\alpha} = \frac{1}{n} \sum_{i=1}^n \ell(\text{sign}(x^{(i)} \boldsymbol{\theta}), y^{(i)})$$

where $\ell()$ is 0 if the prediction is correct, 1 otherwise



Perceptron Rule: If $\mathbf{x}^{(i)}$ is misclassified, do $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + y^{(i)} \mathbf{x}^{(i)}$

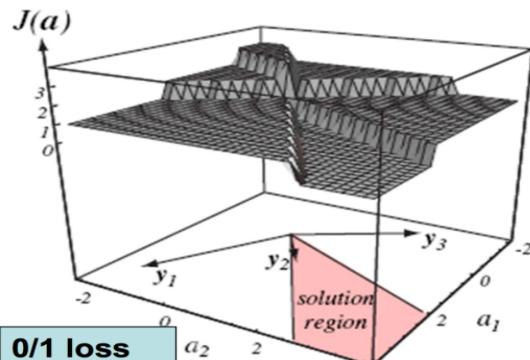
The Perceptron Cost Function

- Prediction is correct if $y^{(i)} \mathbf{x}^{(i)} \boldsymbol{\theta} > 0$

- 0/1 loss**

$$J_{0/1}(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^n \ell(sign(\mathbf{x}^{(i)} \boldsymbol{\theta}), y^{(i)})$$

where $\ell()$ is 0 if the prediction is correct, 1 otherwise

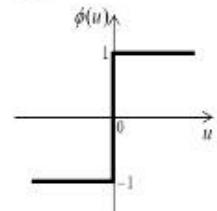


Doesn't produce a useful gradient



No gradient → We cannot use gradient-based optimization methods. Does NOT mean 'unsolvable'

sign function



$$\phi_{\text{sign}}(u) = \begin{cases} 1 & \text{if } u \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

Batch Perceptron

Given training data $\{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^n$

Let $\boldsymbol{\theta} \leftarrow [0, 0, \dots, 0]$

Repeat:

```
    Let  $\Delta \leftarrow [0, 0, \dots, 0]$ 
    for  $i = 1 \dots n$ , do
        if  $y^{(i)} \mathbf{x}^{(i)} \boldsymbol{\theta} \leq 0$           // prediction for  $i^{th}$  instance is incorrect
             $\Delta \leftarrow \Delta + y^{(i)} \mathbf{x}^{(i)}$ 
     $\rightarrow \Delta \leftarrow \Delta / n$                   // compute average update
     $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \Delta$ 
Until  $\|\Delta\|_2 < \epsilon$ 
```

- Simplest case: $\alpha = 1$ and don't normalize, yields the fixed increment perceptron
- Guaranteed to find a separating hyperplane if one exists

Batch Perceptron

Given training data $\{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^n$

Let $\theta \leftarrow [0, 0, \dots, 0]$

Repeat:

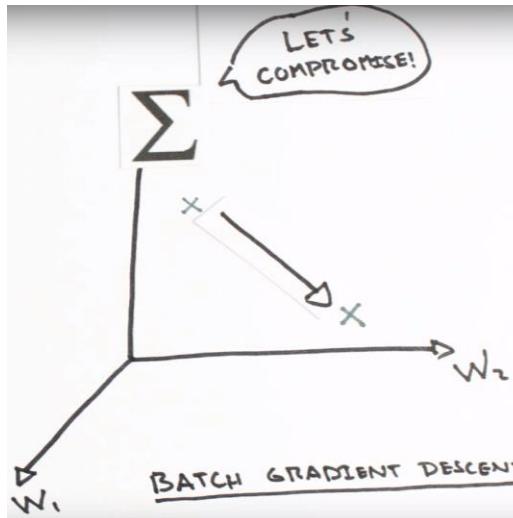
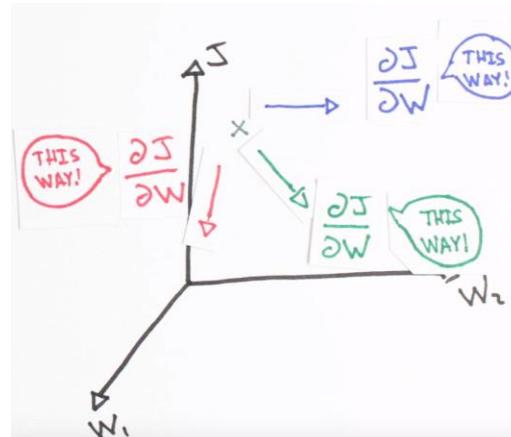
```
    Let  $\Delta \leftarrow [0, 0, \dots, 0]$ 
    for  $i = 1 \dots n$ , do
        if  $y^{(i)} \mathbf{x}^{(i)} \theta \leq 0$           // prediction for  $i^{th}$  instance is incorrect
             $\Delta \leftarrow \Delta + y^{(i)} \mathbf{x}^{(i)}$ 
     $\Delta \leftarrow \Delta / n$                   // compute average update
     $\theta \leftarrow \theta + \alpha \Delta$ 
```

Until $\|\Delta\|_2 < \epsilon$

- Simplest case: $\alpha = 1$ and don't normalize, yields the fixed increment perceptron
- Guaranteed to find a separating hyperplane if one exists

Based on slide by Alan Fern

12



Batch Perceptron

Given training data $\{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^n$

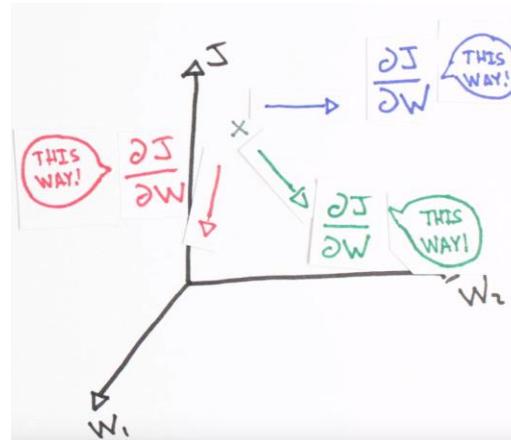
Let $\theta \leftarrow [0, 0, \dots, 0]$

Repeat:

```

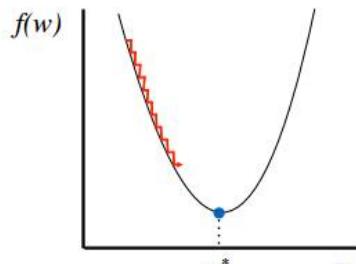
    Let  $\Delta \leftarrow [0, 0, \dots, 0]$ 
    for  $i = 1 \dots n$ , do
        if  $y^{(i)} \mathbf{x}^{(i)} \theta \leq 0$  // prediction for  $i^{th}$  instance is incorrect
             $\Delta \leftarrow \Delta + y^{(i)} \mathbf{x}^{(i)}$ 
     $\Delta \leftarrow \Delta / n$  // compute average update
     $\theta \leftarrow \theta + \alpha \Delta$ 
Until  $\|\Delta\|_2 < \epsilon$ 

```

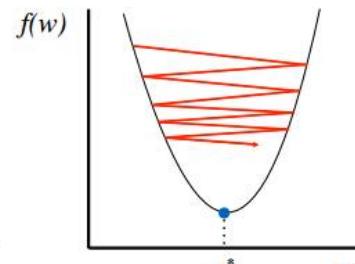


- Simplest case: $\alpha = 1$ and don't normalize, yields the fixed increment perceptron
- Guaranteed to find a separating hyperplane if one exists

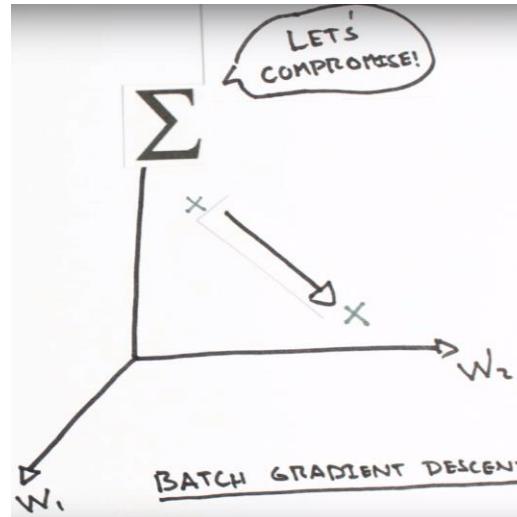
Based on slide by Alan Fern



Too small: converge very slowly



Too big: overshoot and even diverge



Online Perceptron Algorithm

Let $\theta \leftarrow [0, 0, \dots, 0]$

Repeat:

 Receive training example $(x^{(i)}, y^{(i)})$

 if $y^{(i)}x^{(i)}\theta \leq 0$ // prediction is incorrect

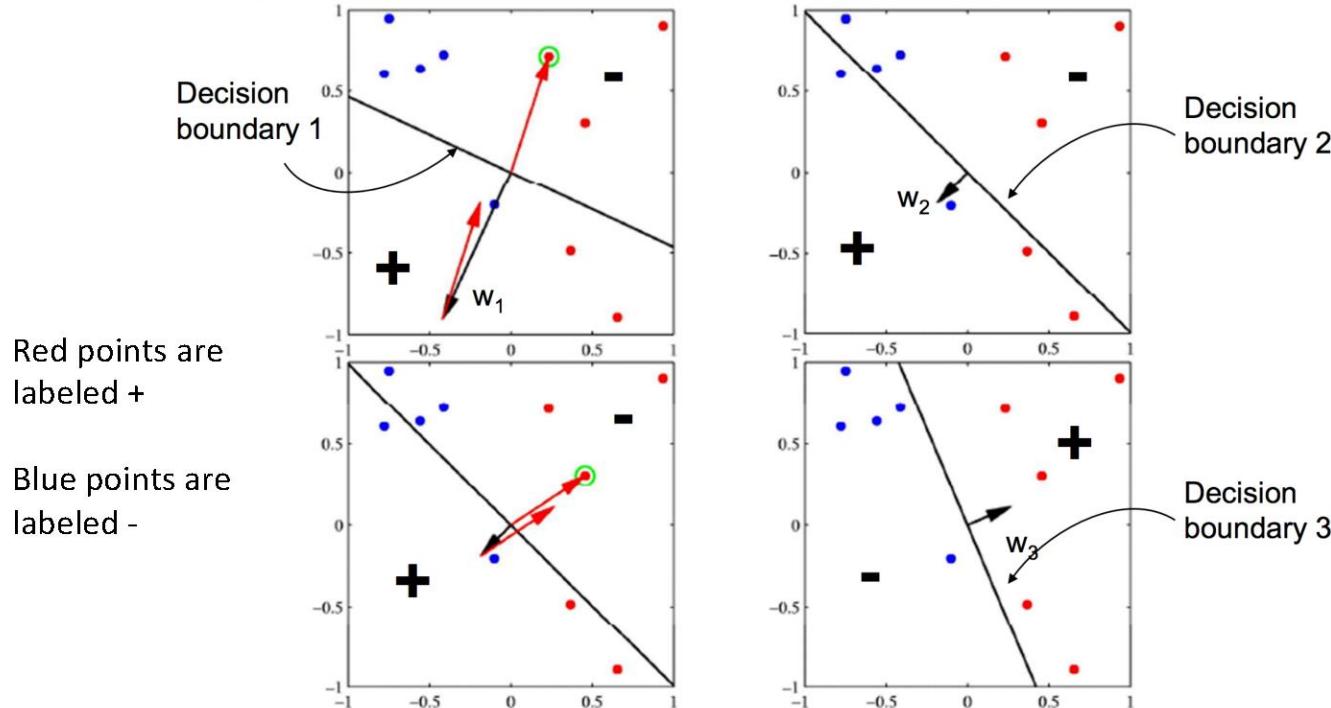
$\theta \leftarrow \theta + y^{(i)}x^{(i)}$

Online learning – the learning mode where the model update is performed each time a single observation is received

Batch learning – the learning mode where the model update is performed after observing the entire training set

Online Perceptron Algorithm

When an error is made, moves the weight in a direction that corrects the error



Animation: <https://www.youtube.com/watch?v=vGwemZhPlsA>

Gradient Descent

1. Initialize the parameters \mathbf{w} to some guess
(usually all zeros, or random values)
2. Update the parameters:

$$\mathbf{w} = \mathbf{w} - \eta \nabla L(\mathbf{w})$$

$$\eta = c_1 / (t^a + c_2)$$

3. Repeat step 2 until $\|\nabla L(\mathbf{w})\| < \theta$ or until the maximum number of iterations is reached.

Stochastic Gradient Descent

A variant of gradient descent makes updates using an approximate of the gradient that is only based on one instance at a time.

$$L_i(\mathbf{w}) = (y_i - \mathbf{w}^T \mathbf{x}_i)^2$$

$$\frac{dL_i}{dw_j} = -2 x_{ij} (y_i - \mathbf{w}^T \mathbf{x}_i)$$

Stochastic Gradient Descent

General algorithm for SGD:

1. Iterate through the instances in a random order
 - a) For each instance x_i , update the weights based on the gradient of the loss for that instance only:

$$\mathbf{w} = \mathbf{w} - \eta \nabla L_i(\mathbf{w}; \mathbf{x}_i)$$

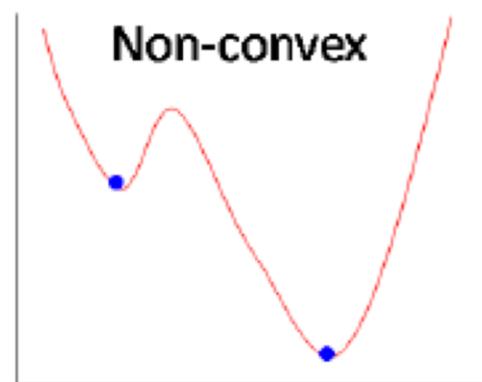
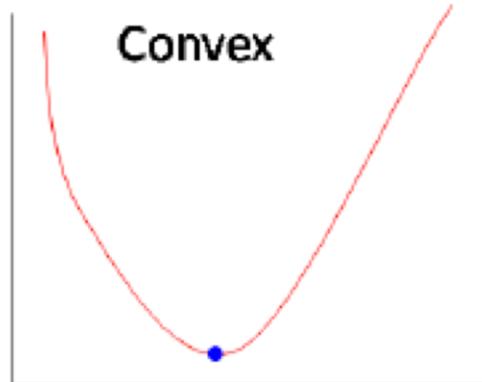
The gradient for one instance's loss is an approximation to the true gradient

- stochastic = random
The *expected* gradient is the true gradient

Convexity

How do you know if you've found the global minimum, or just a local minimum?

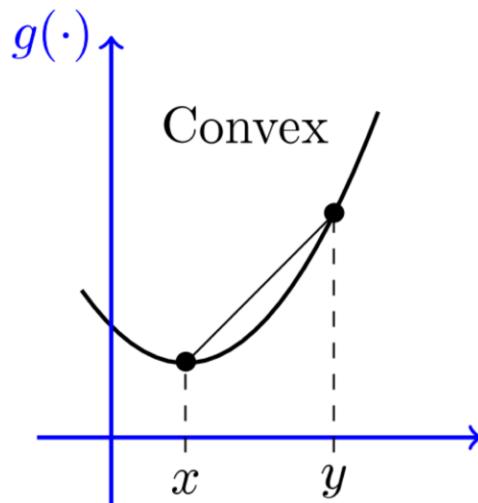
A **convex** function has only one minimum:



Convexity

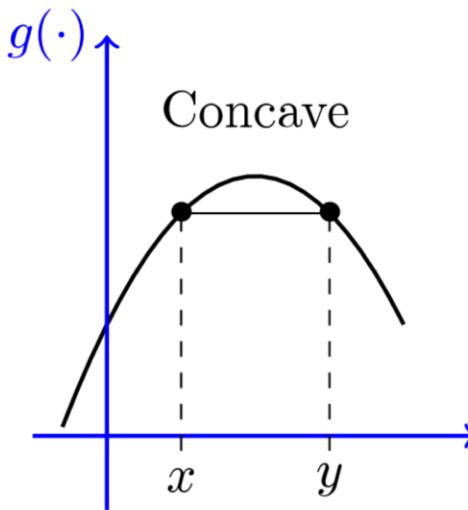
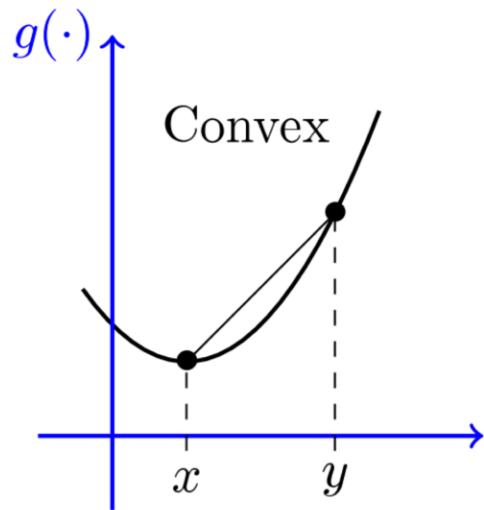
How do you know if you've found the global minimum, or just a local minimum?

A **convex** function has only one minimum:



Convexity

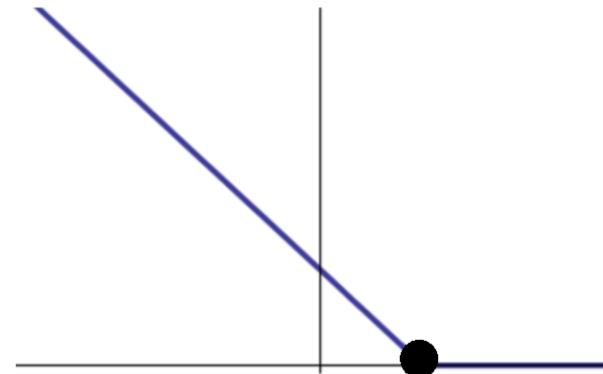
A **concave** function has only one maximum:



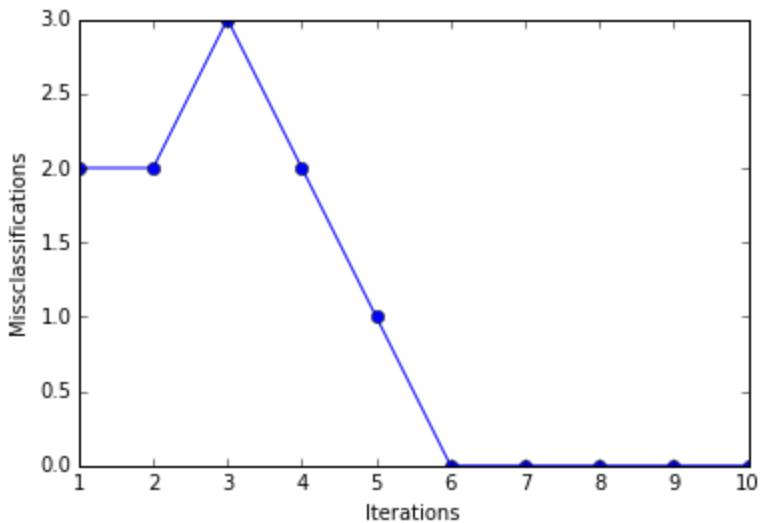
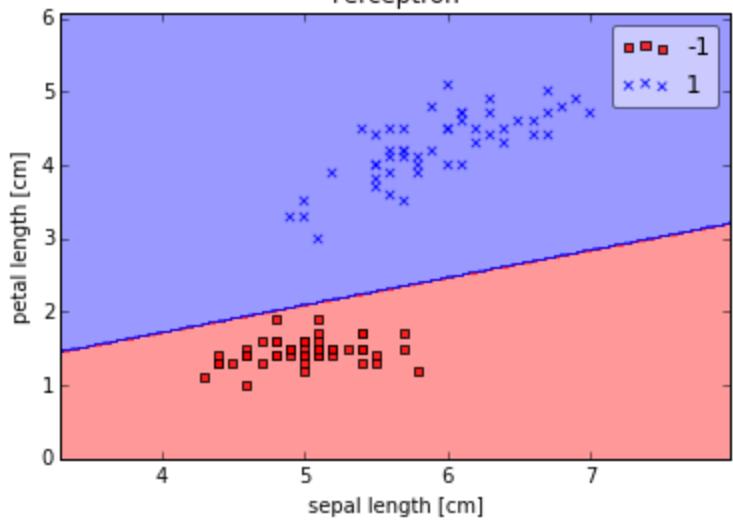
Convexity

Squared error is a convex loss function, as is the perceptron loss.

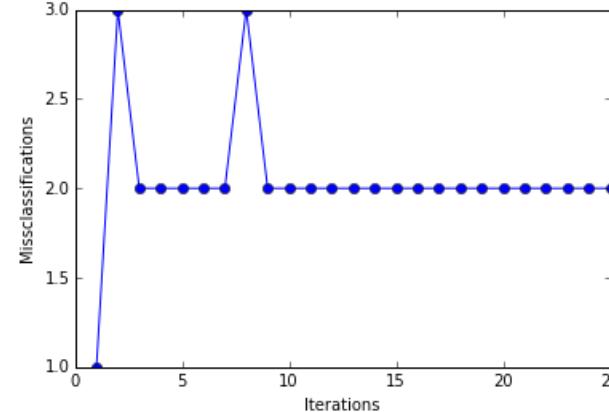
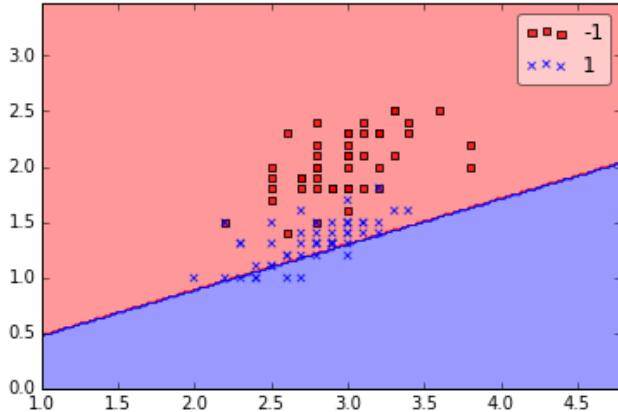
Note: convexity means there is only one minimum value, but there may be multiple parameters that result in that minimum value.



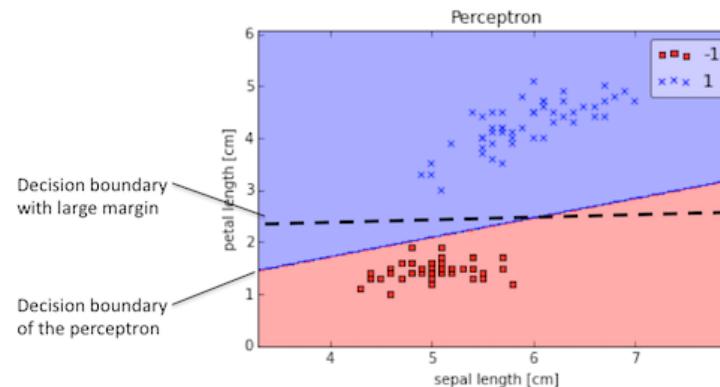
Perceptron



Problem #1: Assumes linear separability



Problem #2: Does not give the 'best' margin



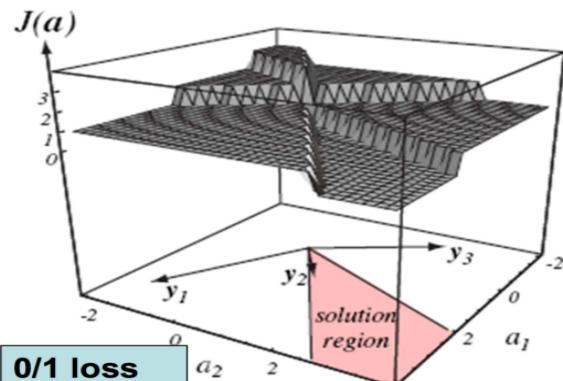
Improving the perceptron

Problem #0: Non-differentiable loss function

- Could have used 0/1 loss

$$J_{0/1}(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^n \ell(\text{sign}(x^{(i)}\boldsymbol{\theta}), y^{(i)})$$

where $\ell()$ is 0 if the prediction is correct, 1 otherwise



Doesn't produce a useful gradient

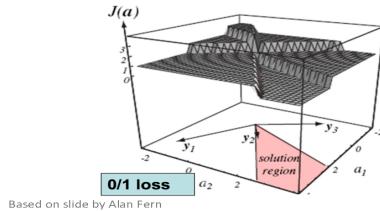
Improving the perceptron

Problem #0: Non-differentiable loss function

- Could have used 0/1 loss

$$J_{0/1}(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^n \ell(\text{sign}(x^{(i)}\boldsymbol{\theta}), y^{(i)})$$

where $\ell()$ is 0 if the prediction is correct, 1 otherwise



Doesn't produce a useful gradient



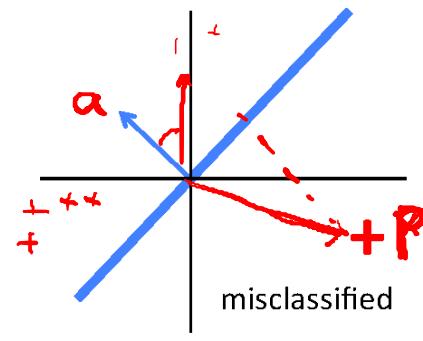
8

Solution: Minimize the 'misclassification distance'

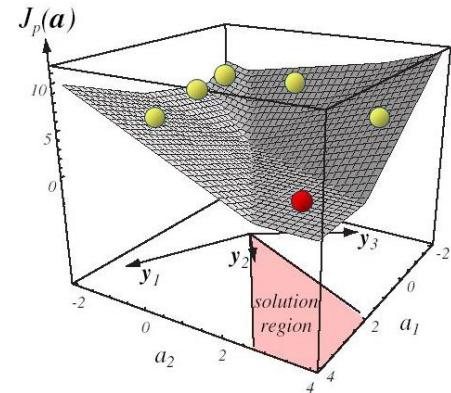
$$J_p(\mathbf{a}) = \sum_{\mathbf{x} \in M} (-\mathbf{a}^T \mathbf{x})$$

$$\mathbf{a}^T \mathbf{x} = 0$$

$M \rightarrow$ Set of misclassified samples



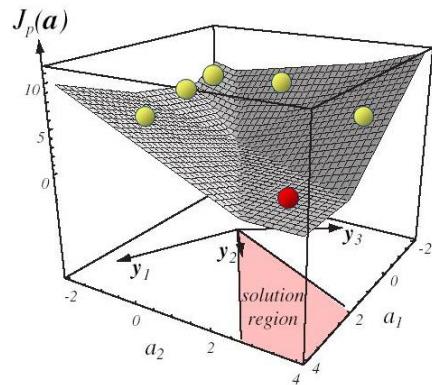
Based on slide by Alan Fern



Solution: Minimize the ‘misclassification distance’

$$J_p(\mathbf{a}) = \sum_{\mathbf{x} \in \mathbb{M}} (-\mathbf{a}^T \mathbf{x})$$

\mathbb{M} → Set of misclassified samples



$$\nabla J_p(\mathbf{a}) = \sum_{\mathbf{x} \in \mathbb{M}} (-\mathbf{x})$$

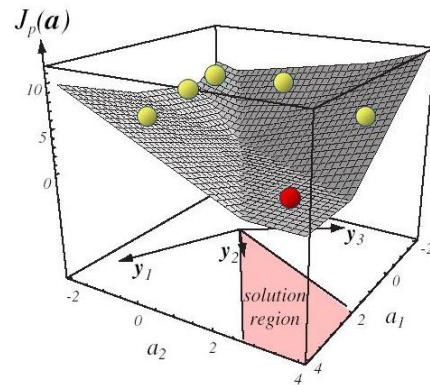
$$\mathbf{a}^{(k+1)} = \mathbf{a}^k - \eta^k \nabla J_p(\mathbf{a})$$

$$\mathbf{a}^{(k+1)} = \mathbf{a}^k + \eta^k \sum_{\mathbf{x} \in \mathbb{M}} \mathbf{x}$$

Solution: Minimize the ‘misclassification distance’

$$J_p(\mathbf{a}) = \sum_{\mathbf{x} \in \mathbb{M}} (-\mathbf{a}^T \mathbf{x})$$

\mathbb{M} → Set of misclassified samples



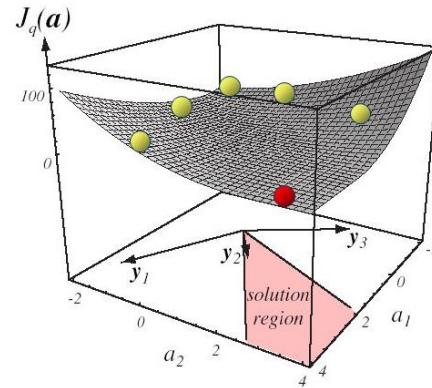
$$\nabla J_p(\mathbf{a}) = \sum_{\mathbf{x} \in \mathbb{M}} (-\mathbf{x})$$

$$\mathbf{a}^{(k+1)} = \mathbf{a}^k - \eta^k \nabla J_p(\mathbf{a})$$

$$\mathbf{a}^{(k+1)} = \mathbf{a}^k + \eta^k \sum_{\mathbf{x} \in \mathbb{M}} \mathbf{x}$$

Solution: Minimize the **square** of ‘misclassification distance’

$$J_q(\mathbf{a}) = \sum_{\mathbf{x} \in \mathbb{M}} (\mathbf{a}^T \mathbf{x})^2$$



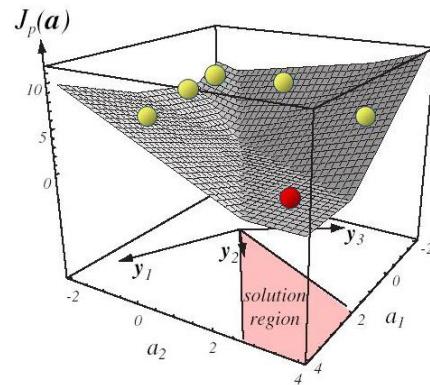
$$\nabla J_q(\mathbf{a}) =$$

$$\mathbf{a}^{(k+1)} =$$

Solution: Minimize the ‘misclassification distance’

$$J_p(\mathbf{a}) = \sum_{\mathbf{x} \in \mathbb{M}} (-\mathbf{a}^T \mathbf{x})$$

\mathbb{M} → Set of misclassified samples



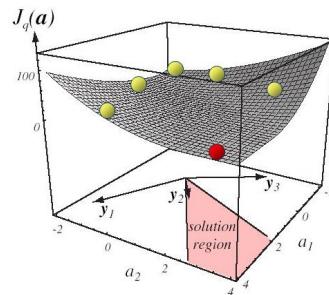
$$\nabla J_p(\mathbf{a}) = \sum_{\mathbf{x} \in \mathbb{M}} (-\mathbf{x})$$

$$\mathbf{a}^{(k+1)} = \mathbf{a}^k - \eta^k \nabla J_p(\mathbf{a})$$

$$\mathbf{a}^{(k+1)} = \mathbf{a}^k + \eta^k \sum_{\mathbf{x} \in \mathbb{M}} \mathbf{x}$$

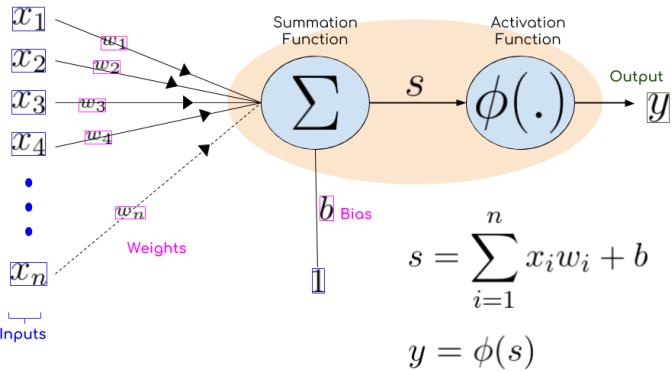
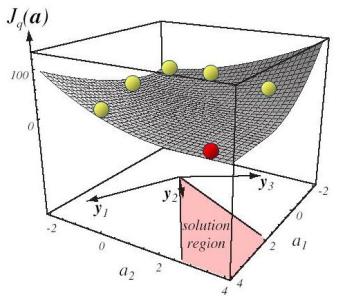
Solution: Minimize the **square** of ‘misclassification distance’

$$J_q(\mathbf{a}) = \sum_{\mathbf{x} \in \mathbb{M}} (\mathbf{a}^T \mathbf{x})^2$$



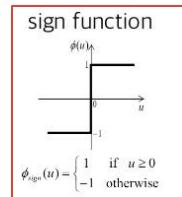
Solution: Minimize the **square** of ‘misclassification distance’

$$J_q(\mathbf{a}) = \sum_{\mathbf{x} \in \mathbb{M}} (\mathbf{a}^T \mathbf{x})^2$$



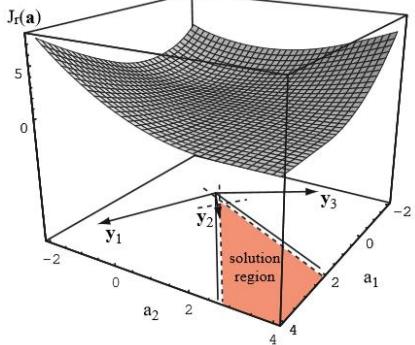
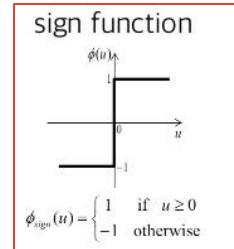
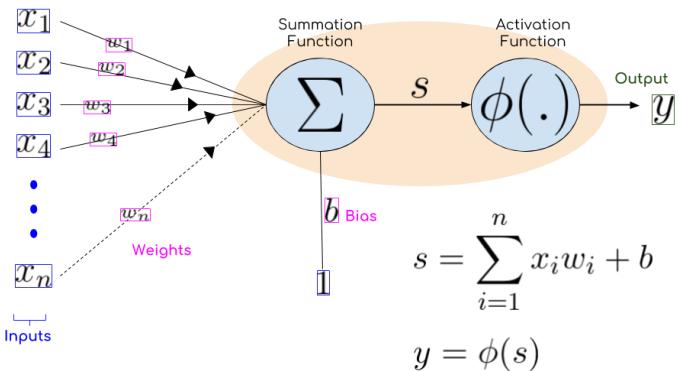
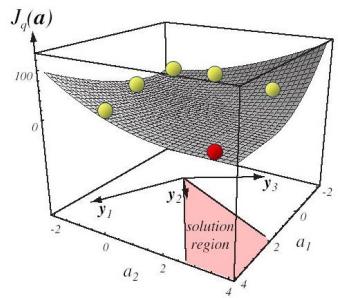
$$s = \sum_{i=1}^n x_i w_i + b$$

$$y = \phi(s)$$



Solution: Minimize the **square** of 'misclassification distance'

$$J_q(\mathbf{a}) = \sum_{\mathbf{x} \in \mathbb{M}} (\mathbf{a}^T \mathbf{x})^2$$



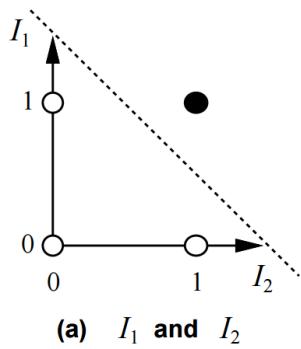
$$J_r(\mathbf{a}) = \sum_{\mathbf{x} \in \mathbb{M}} \frac{(\mathbf{a}^T \mathbf{x} - b)^2}{\|\mathbf{x}\|^2}$$



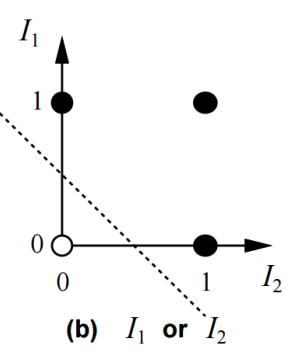
Loss function is changing. Activation function has remained fixed ('sign').

CAN A PERCEPTRON SEPARATE DISTRIBUTIONS WHICH ARE NOT LINEARLY SEPARABLE ???

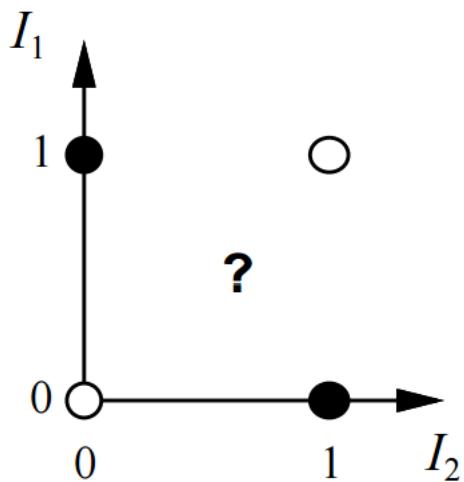
AND



OR



LINEARLY SEPARABLE CLASSES



(c) $I_1 \text{ xor } I_2$

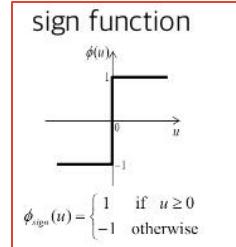
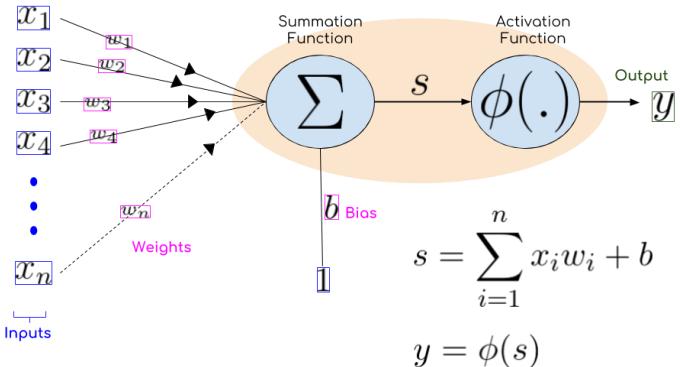
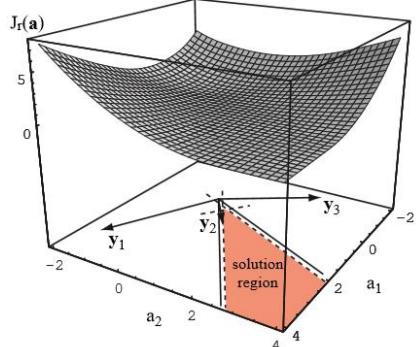
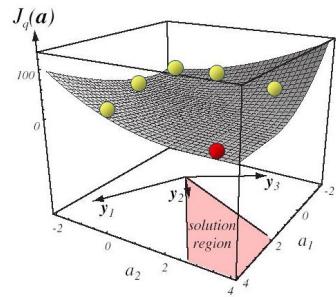
Solution: Minimize the **square** of 'misclassification distance'

$$J_q(\mathbf{a}) = \sum_{\mathbf{x} \in \mathbb{M}} (\mathbf{a}^T \mathbf{x})^2$$

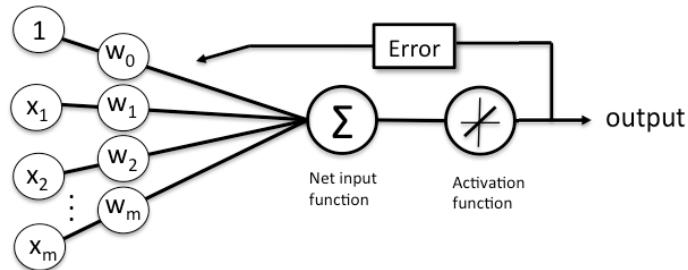
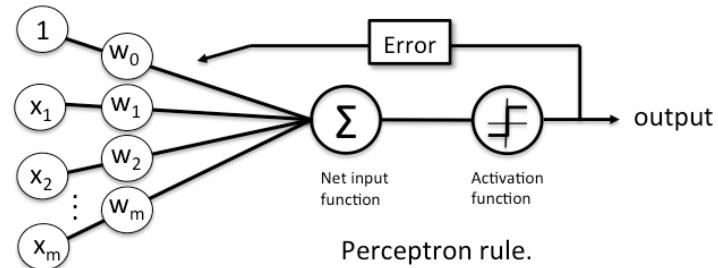
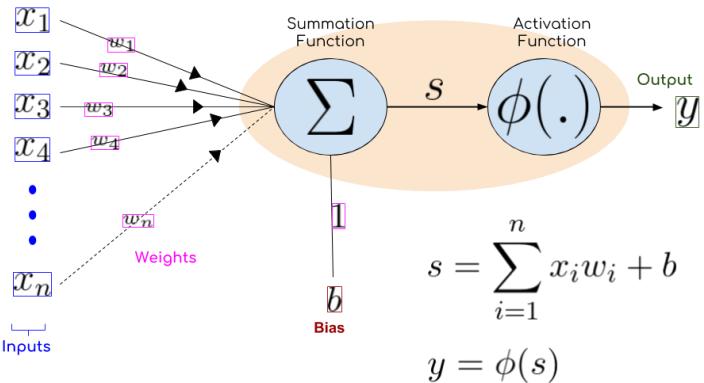


Converges even when
classes are not separable

$$J_r(\mathbf{a}) = \sum_{\mathbf{x} \in \mathbb{M}} \frac{(\mathbf{a}^T \mathbf{x} - b)^2}{\|\mathbf{x}\|^2}$$



REGRESSION



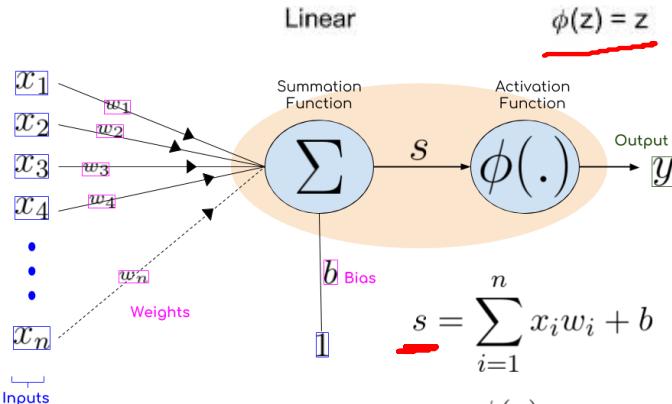
| Activation Function | Equation | Example | 1D Graph |
|---------------------|---------------|----------------------------|----------|
| Linear | $\phi(z) = z$ | Adaline, linear regression | |

REGRESSION – GRADIENT DESCENT UPDATE

$$\begin{aligned} \text{Loss Function: } & \ell(\hat{y}, y) = (\hat{y} - y)^2 \\ \text{Cost Function: } & J_i = \ell(\hat{y}_i, y_i) \\ \text{Partial Derivative: } & \frac{\partial J}{\partial w} = \frac{\partial J}{\partial \ell} \cdot \frac{\partial \ell}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w} \\ \Delta &= 1 \cdot 2(\hat{y} - y) \times \end{aligned}$$

w

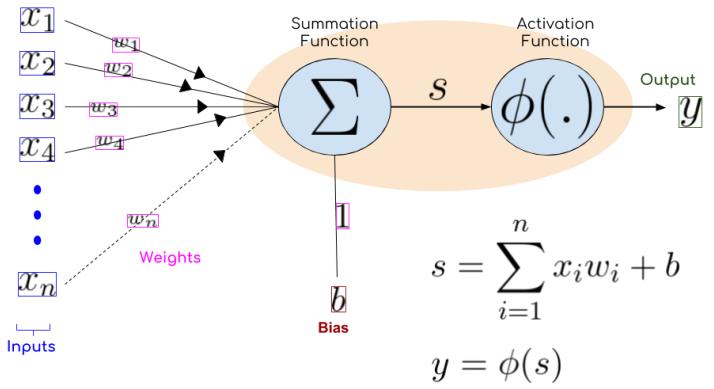
| Activation Function | Equation |
|---------------------|----------|
|---------------------|----------|



$$\hat{y} = \omega^T x$$

$$w = w + \Delta$$

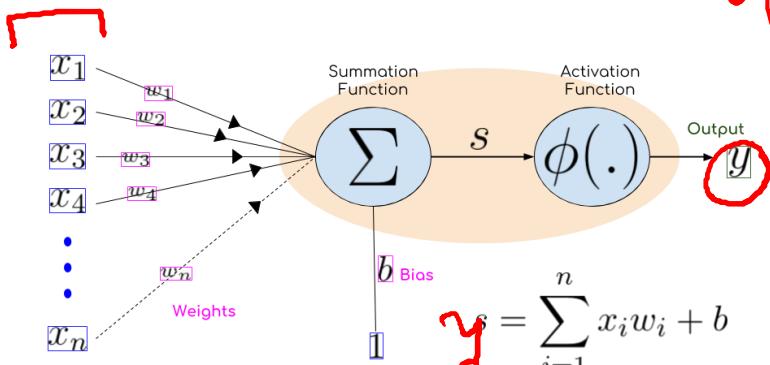
ACTIVATION FUNCTIONS



| Activation Function | Equation | Example | 1D Graph |
|-----------------------------------|---|----------------------------|----------|
| Linear | $\phi(z) = z$ | Adaline, linear regression | |
| Unit Step (Heaviside Function) | $\phi(z) = \begin{cases} 0 & z < 0 \\ 0.5 & z = 0 \\ 1 & z > 0 \end{cases}$ | Perceptron variant | |
| Sign (signum) | $\phi(z) = \begin{cases} -1 & z < 0 \\ 0 & z = 0 \\ 1 & z > 0 \end{cases}$ | Perceptron variant | |
| Piece-wise Linear | $\phi(z) = \begin{cases} 0 & z \leq -\frac{1}{2} \\ z + \frac{1}{2} & -\frac{1}{2} \leq z \leq \frac{1}{2} \\ 1 & z \geq \frac{1}{2} \end{cases}$ | Support vector machine | |
| Logistic (sigmoid) | $\phi(z) = \frac{1}{1 + e^{-z}}$ | Logistic regression, | |
| Hyperbolic Tangent (tanh) | $\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$ | | |
| ReLU | $\phi(z) = \begin{cases} 0 & z < 0 \\ z & z > 0 \end{cases}$ | | |

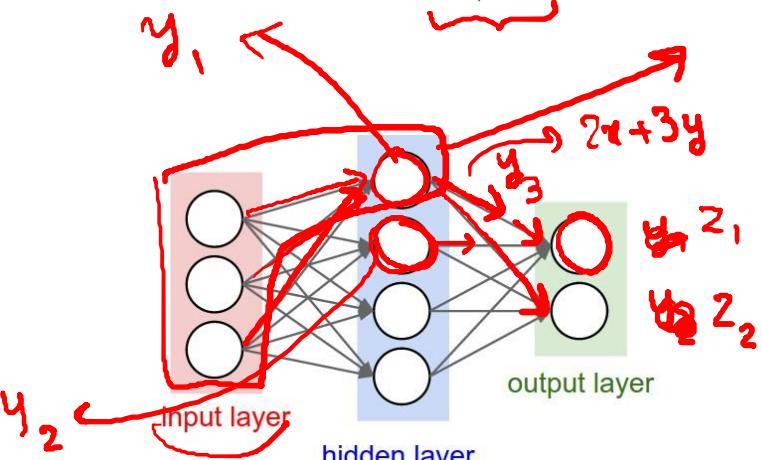
WHY USE ONLY ONE NEURON ?

$$S(S(\vec{w}^T \vec{x}) + s + - \dots -)$$



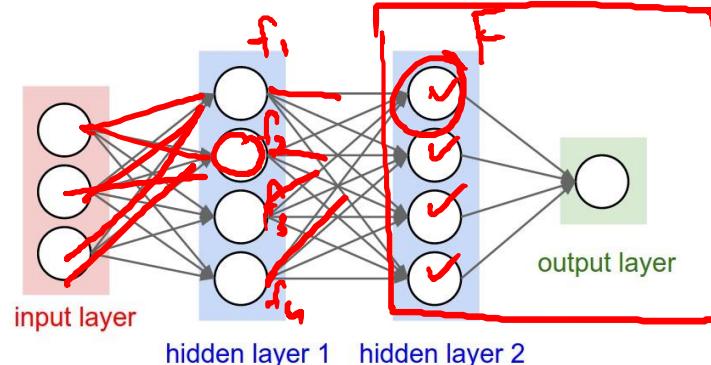
$$\begin{aligned} y &= \sum_{i=1}^n x_i w_i + b \\ y &= \phi(s) \end{aligned}$$

$$f(x)$$

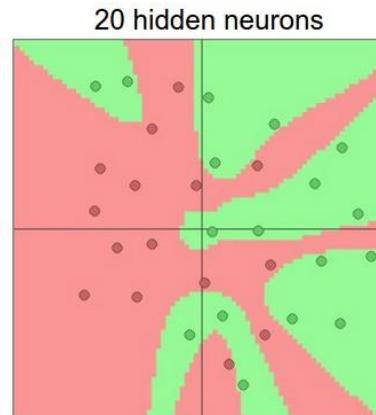
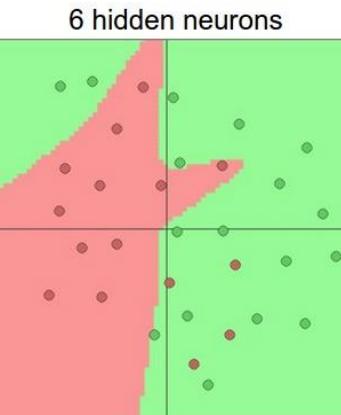
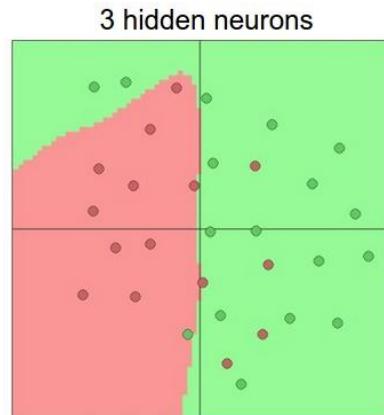
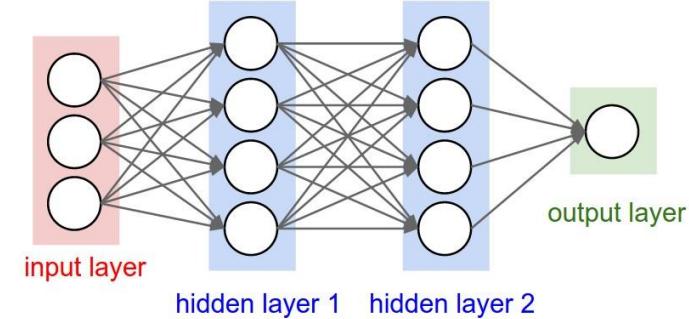
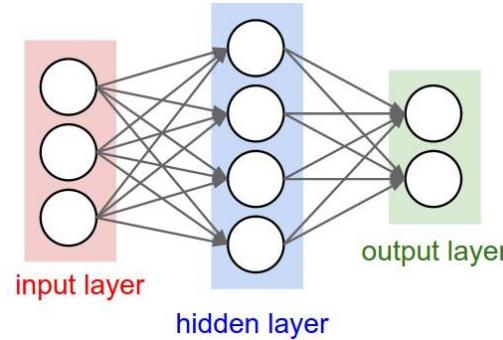


$$\begin{matrix} w_1 & w'_1 \\ w_2 & w'_2 \\ w_3 & w'_3 \end{matrix}$$

$$y, y'$$



WHY USE ONLY ONE NEURON ?

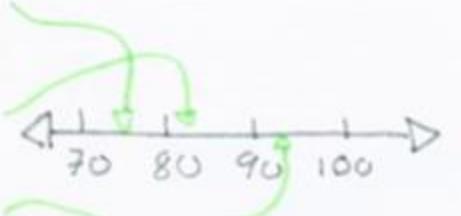


MULTI-NEURON NETWORKS

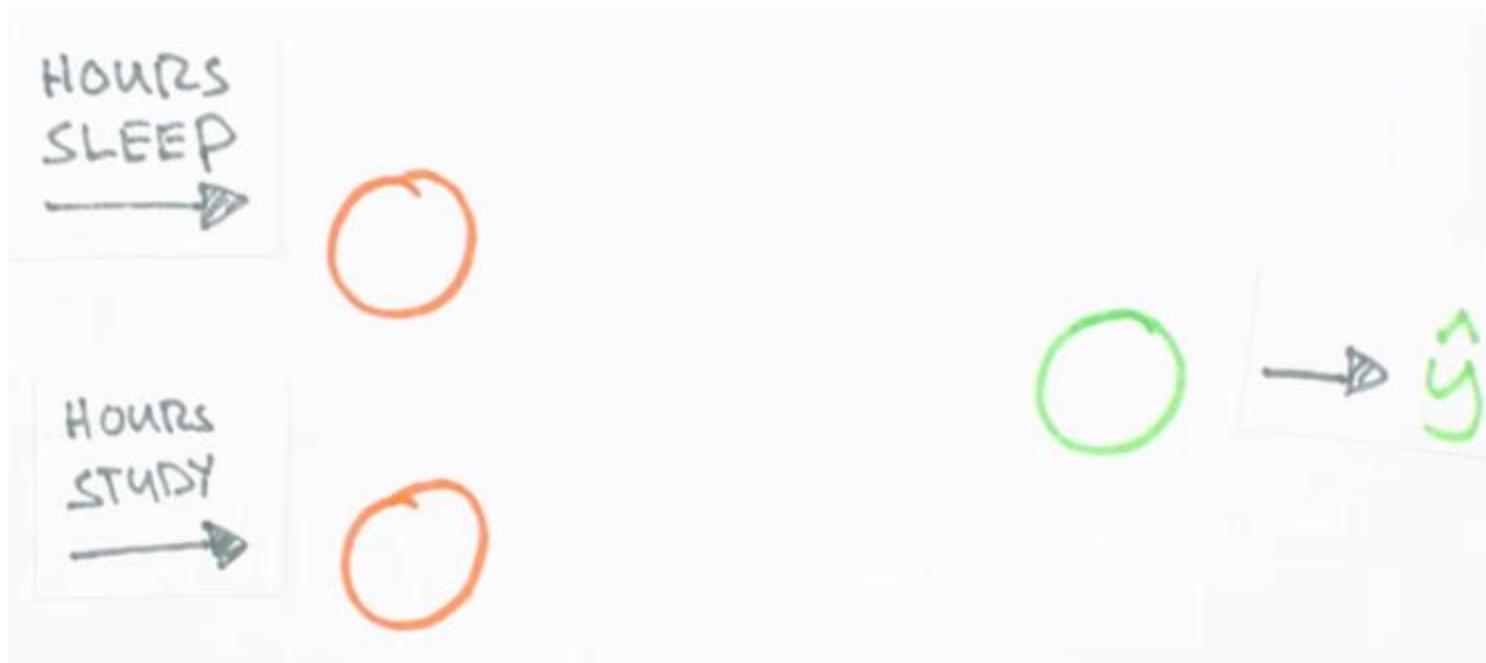
| | X (HOURS SLEEP, HOURS STUDY) | y (SCORE ON TEST) |
|----------|------------------------------------|----------------------|
| TRAINING | (3, 5) | 75 |
| | (5, 1) | 82 |
| | (10, 2) | 93 |
| TESTING | (8, 3) | ? |

MULTI-NEURON NETWORKS

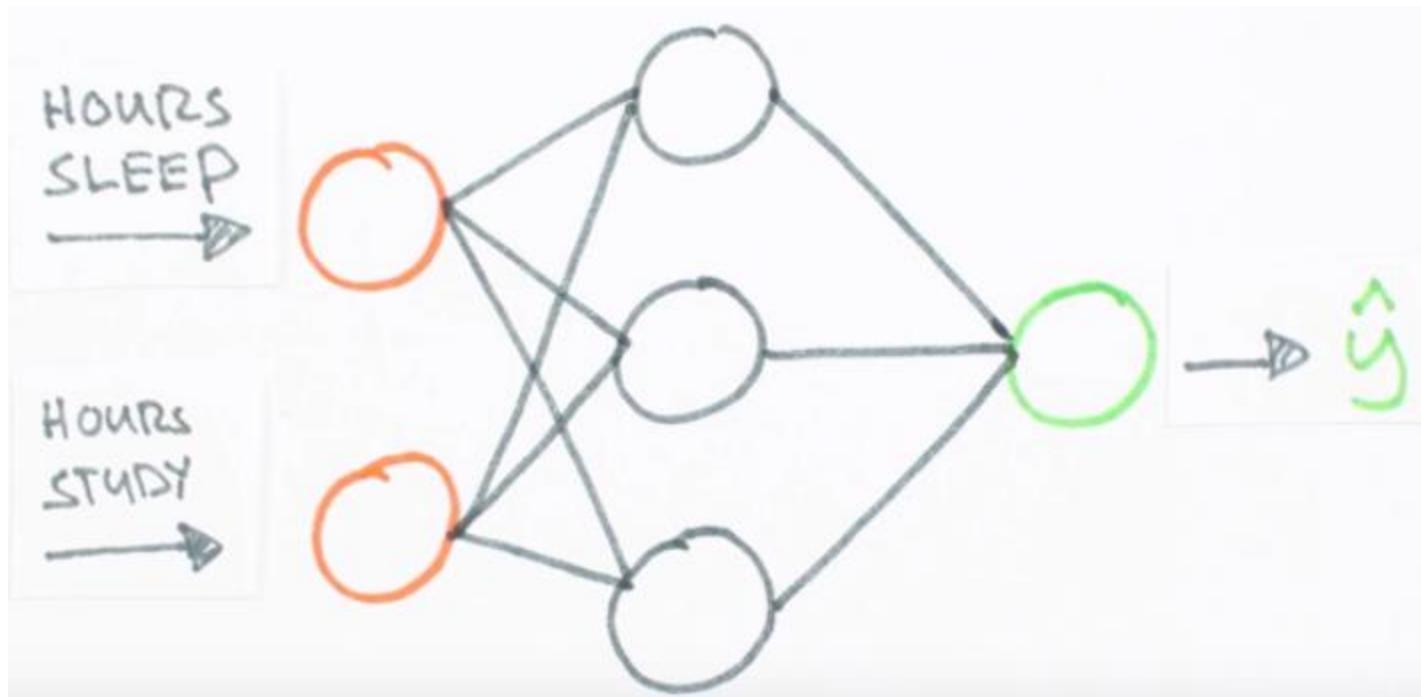
SUPERVISED

| | <u>INPUTS</u> | <u>OUTPUTS</u> | |
|----------|---------------|----------------|--|
| TRAINING | (3, 5) | 75 | REGRESSION |
| | (5, 1) | 82 | |
| TESTING | (10, 2) | 93 |  |
| | (8, 3) | ? | |

MULTI-NEURON NETWORKS :: ARCHITECTURE

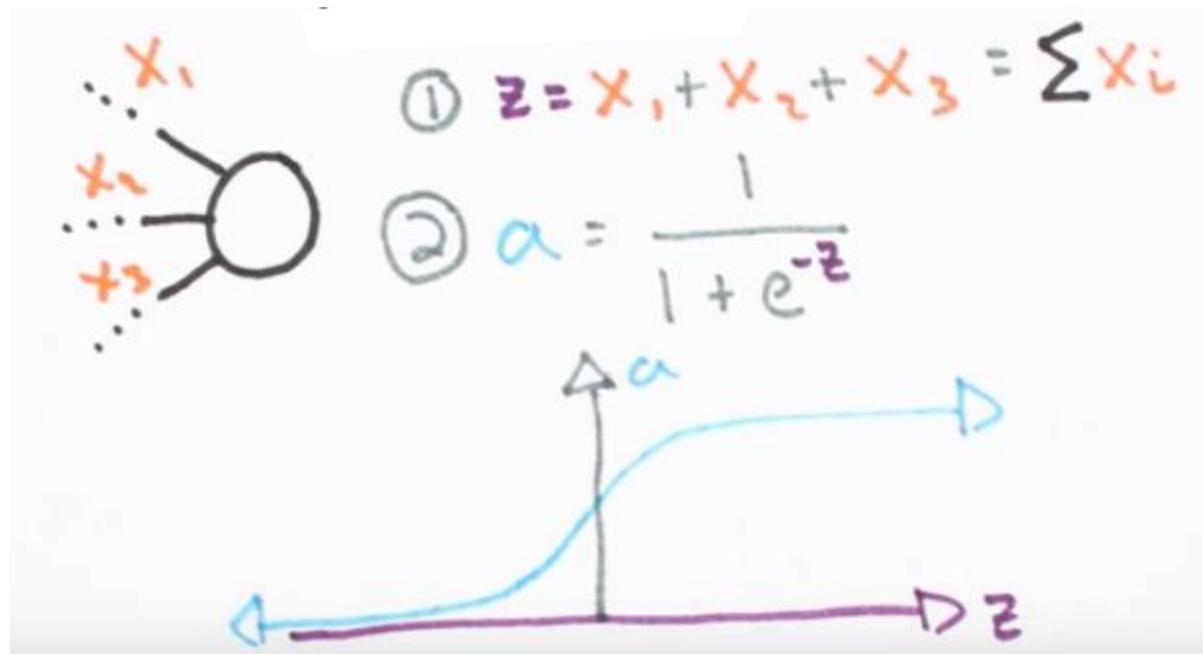


MULTI-NEURON NETWORKS :: ARCHITECTURE



MULTI-NEURON NETWORKS :: ARCHITECTURE

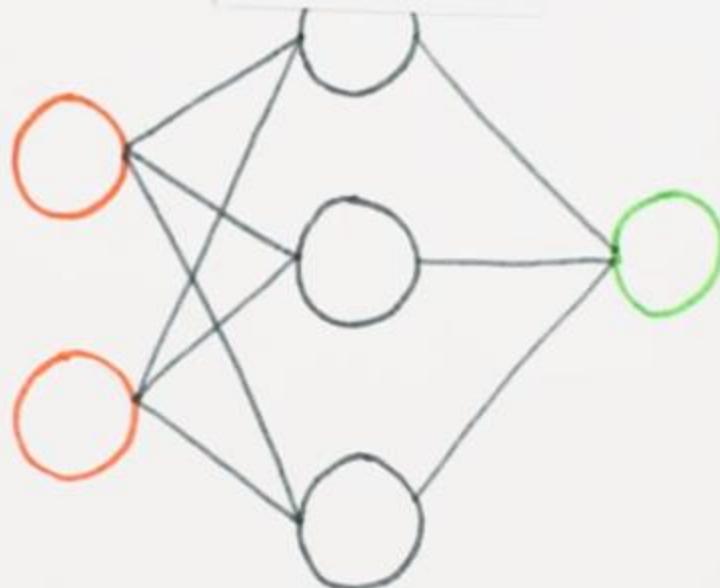
NEURON



MULTI-NEURON NETWORKS :: ARCHITECTURE

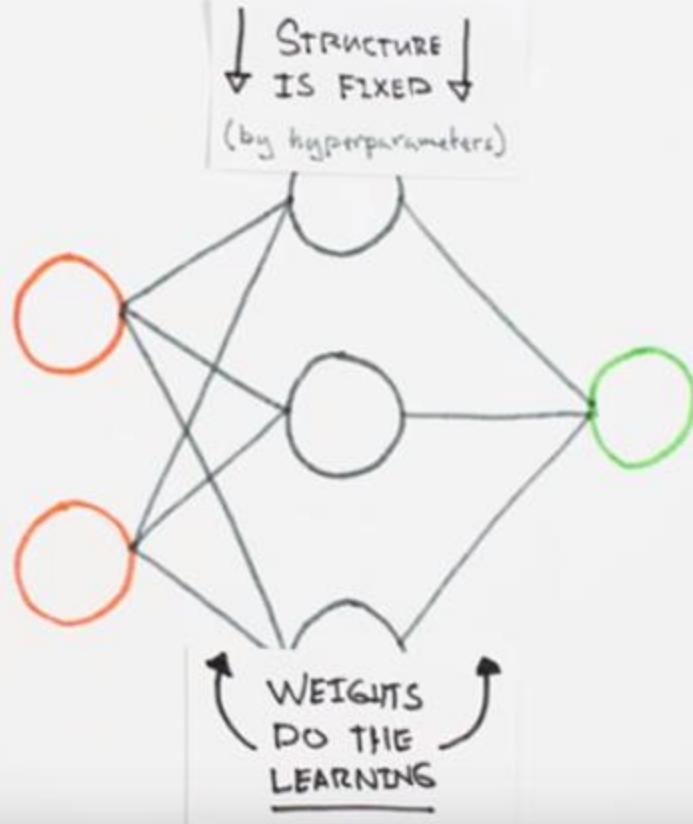
```
In [ ]: class Neural_Network(object):
    def __init__(self):
        #Define HyperParameters
        self.inputLayerSize = 2
        self.outputLayerSize = 1
        self.hiddenLayerSize = 3
```

↓ STRUCTURE
↓ IS FIXED ↓
(by hyperparameters)



MULTI-NEURON NETWORKS :: ARCHITECTURE

```
In [ ]: class Neural_Network(object):  
    def __init__(self):  
        #Define HyperParameters  
        self.inputLayerSize = 2  
        self.outputLayerSize = 1  
        self.hiddenLayerSize = 3
```



MULTI-NEURON NETWORKS :: TRAINING

INITIALIZE NETWORK WITH RANDOM WEIGHTS

WHILE [NOT CONVERGED]

DO FORWARD PROP

DO BACKPROP AND DETERMINE CHANGE IN WEIGHTS

UPDATE ALL WEIGHTS IN ALL LAYERS

MULTI-NEURON NETWORKS :: TRAINING

INITIALIZE NETWORK WITH RANDOM WEIGHTS

WHILE [NOT CONVERGED]

DO FORWARD PROP

DO BACKPROP AND DETERMINE CHANGE IN WEIGHTS

UPDATE ALL WEIGHTS IN ALL LAYERS

Initialize weights w ; // Random or 0

Until [all examples correctly classified]

For each training sample (x, y)

Compute $yt := x^T w$

if $y == yt$ // Correctly classified
 continue ;

else // Update weights

$dw = (y - yt) * x$;
 $w = w + dw$;

EndIf

EndFor

EndUntil

MULTI-NEURON NETWORKS :: FORWARD PROPAGATION

```
In [1]: class Neural_Network(object):
    def __init__(self):
        #Define HyperParameters
        self.inputLayerSize = 2
        self.outputLayerSize = 1
        self.hiddenLayerSize = 3

    def forward(self, X):
        #Propagate inputs through network
```

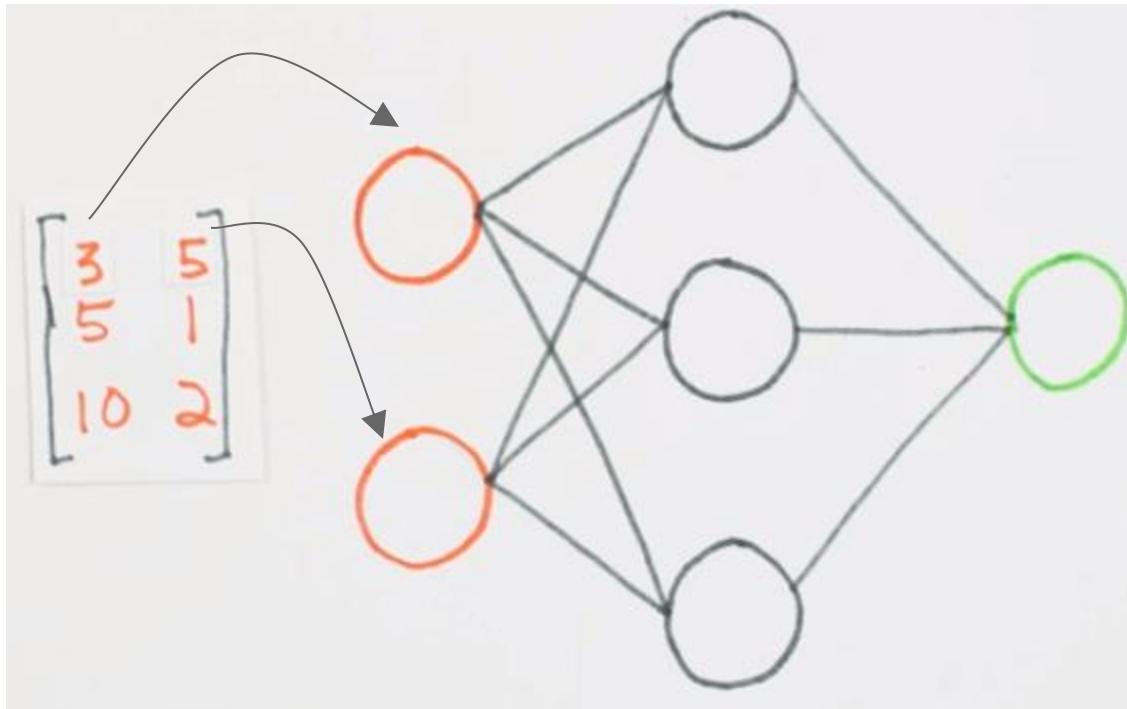


MATLAB

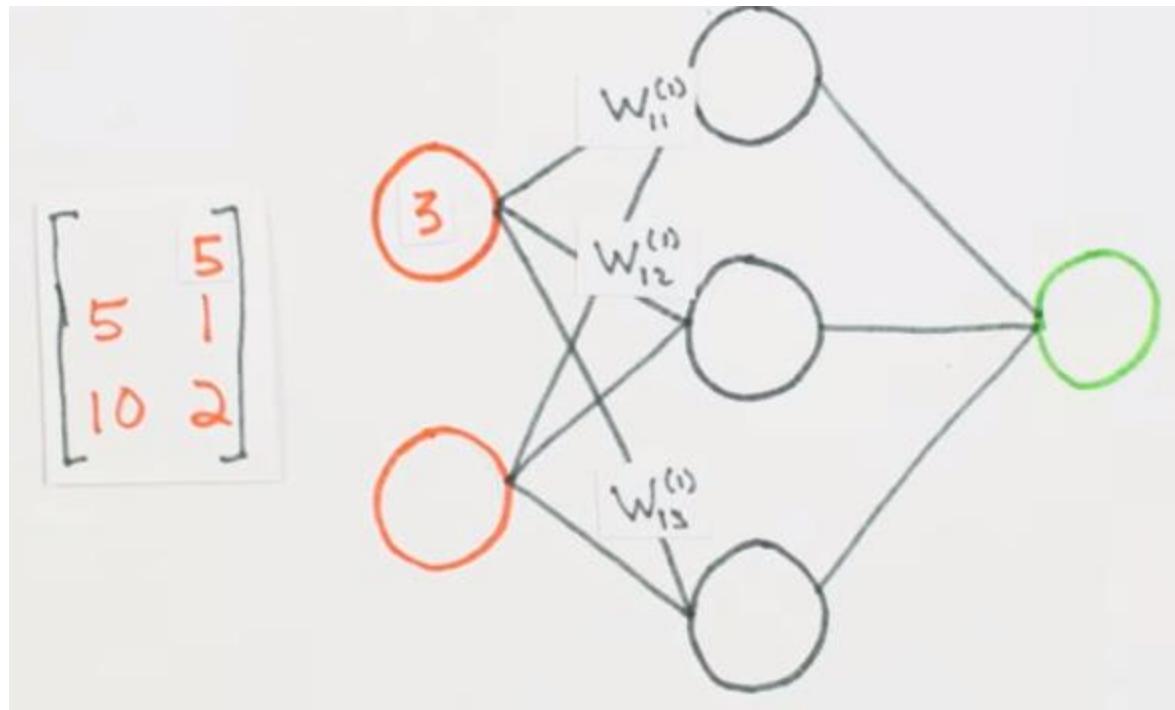


NumPy

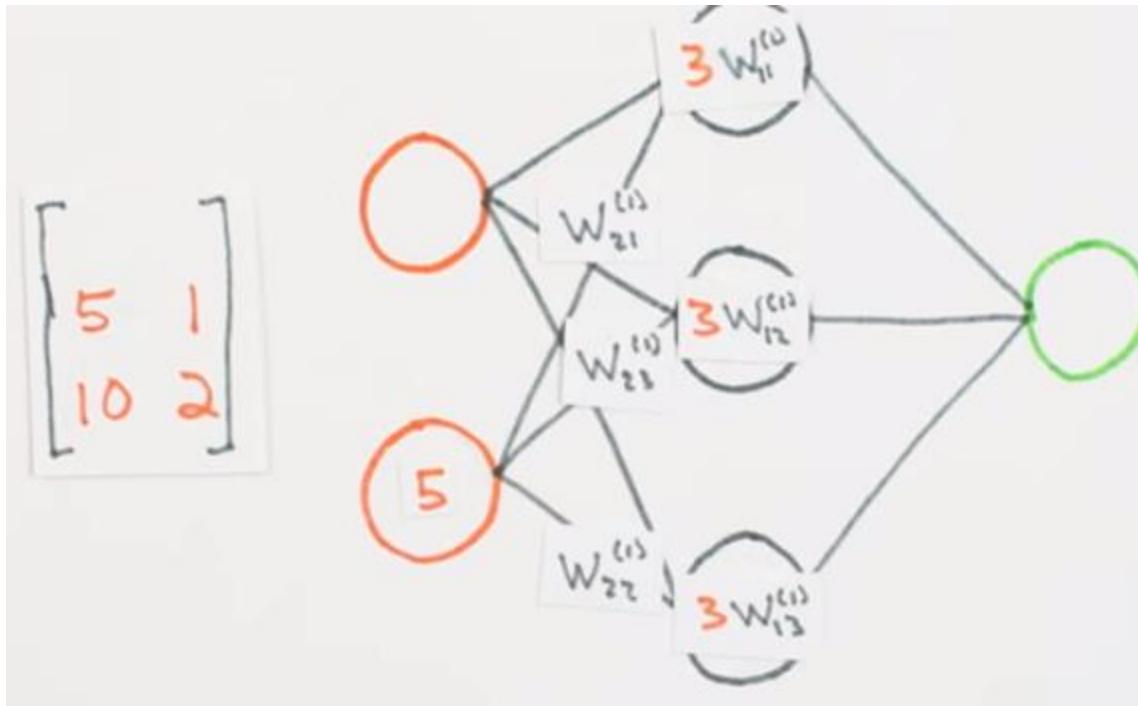
MULTI-NEURON NETWORKS :: FORWARD PROPAGATION



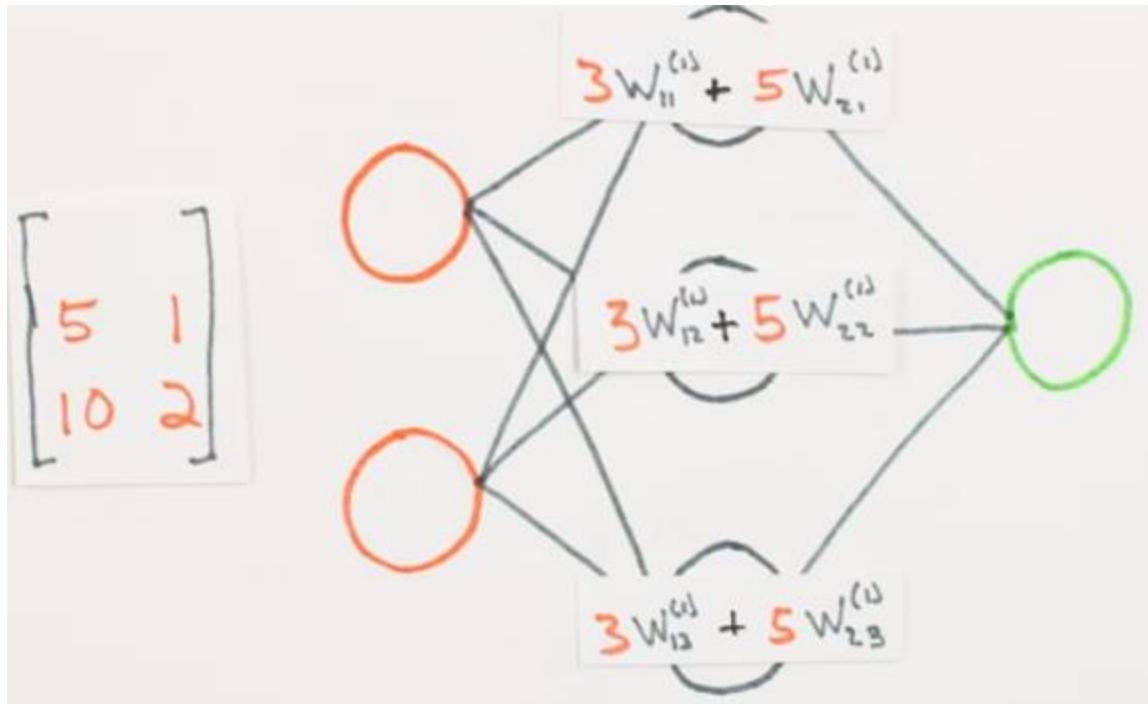
MULTI-NEURON NETWORKS :: FORWARD PROPAGATION



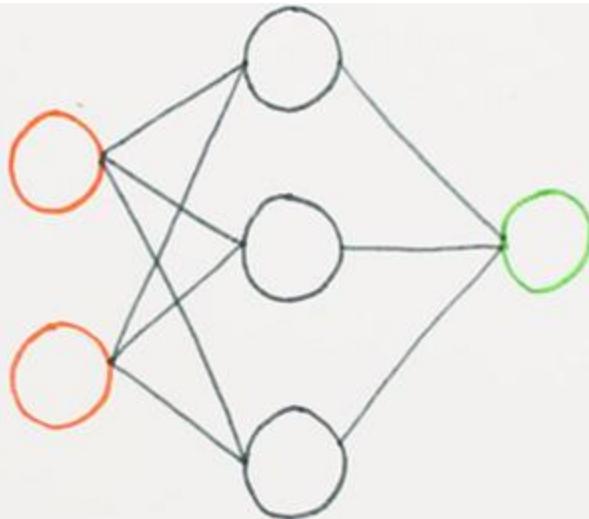
MULTI-NEURON NETWORKS :: FORWARD PROPAGATION



MULTI-NEURON NETWORKS :: FORWARD PROPAGATION

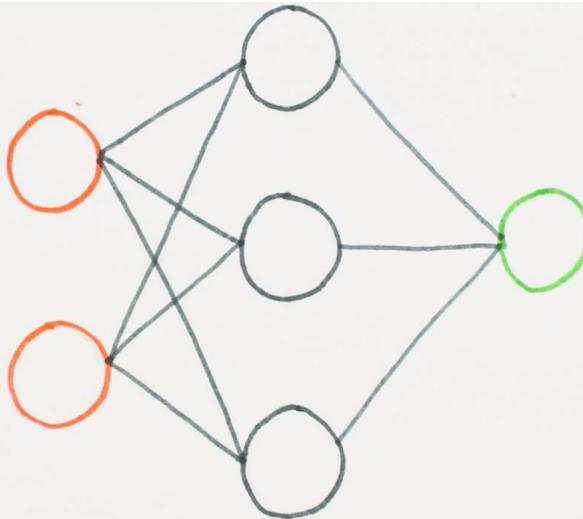


MULTI-NEURON NETWORKS :: FORWARD PROPAGATION



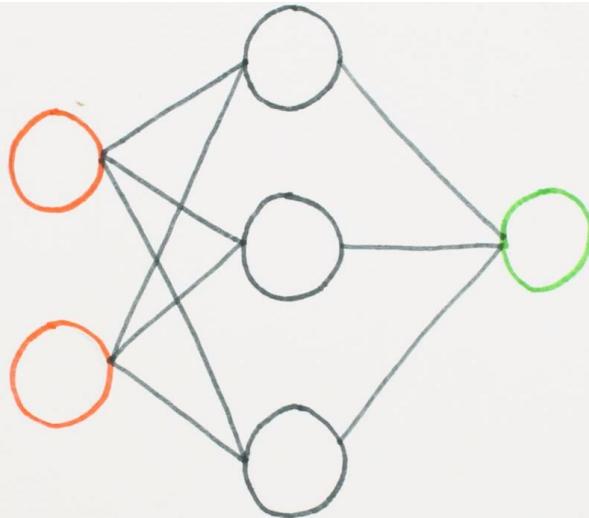
$$\begin{bmatrix} 3 & 5 \end{bmatrix} \begin{bmatrix} W_{11}^{(1)} & W_{12}^{(1)} & W_{13}^{(1)} \\ W_{21}^{(1)} & W_{22}^{(1)} & W_{23}^{(1)} \end{bmatrix} = \begin{bmatrix} 3W_{11}^{(1)} + 5W_{21}^{(1)} & 3W_{12}^{(1)} + 5W_{22}^{(1)} & 3W_{13}^{(1)} + 5W_{23}^{(1)} \\ \dots \end{bmatrix}$$

MULTI-NEURON NETWORKS :: FORWARD PROPAGATION



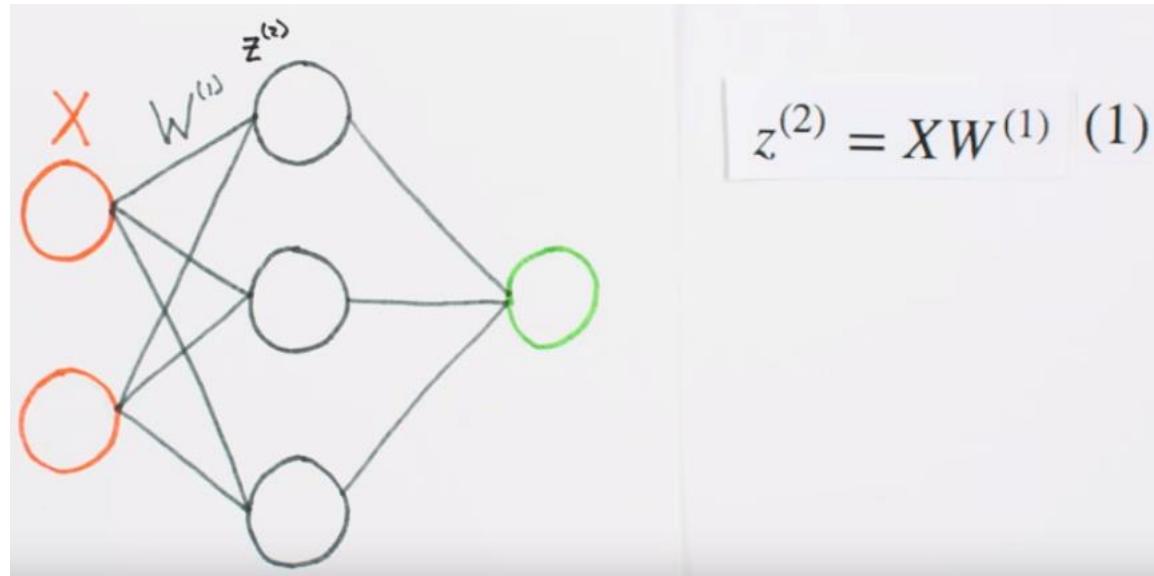
$$\begin{bmatrix} 3 & 5 \\ 5 & 1 \\ 10 & 2 \end{bmatrix} \begin{bmatrix} W_{11}^{(1)} & W_{12}^{(1)} & W_{13}^{(1)} \\ W_{21}^{(1)} & W_{22}^{(1)} & W_{23}^{(1)} \end{bmatrix} = \begin{bmatrix} 3W_{11}^{(1)} + 5W_{21}^{(1)} & 3W_{12}^{(1)} + 5W_{22}^{(1)} & 3W_{13}^{(1)} + 5W_{23}^{(1)} \\ 5W_{11}^{(1)} + 1W_{21}^{(1)} & 5W_{12}^{(1)} + 1W_{22}^{(1)} & 5W_{13}^{(1)} + 1W_{23}^{(1)} \\ 10W_{11}^{(1)} + 2W_{21}^{(1)} & 10W_{12}^{(1)} + 2W_{22}^{(1)} & 10W_{13}^{(1)} + 2W_{23}^{(1)} \end{bmatrix}$$

MULTI-NEURON NETWORKS :: FORWARD PROPAGATION

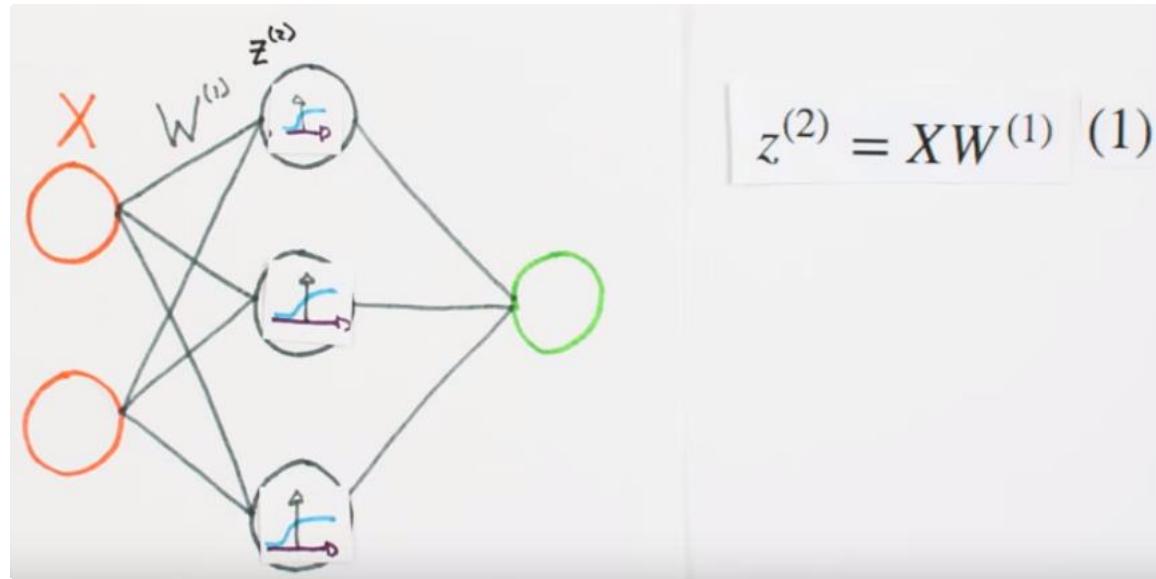


$$X \quad | \quad W^{(1)} = Z^{(2)}$$

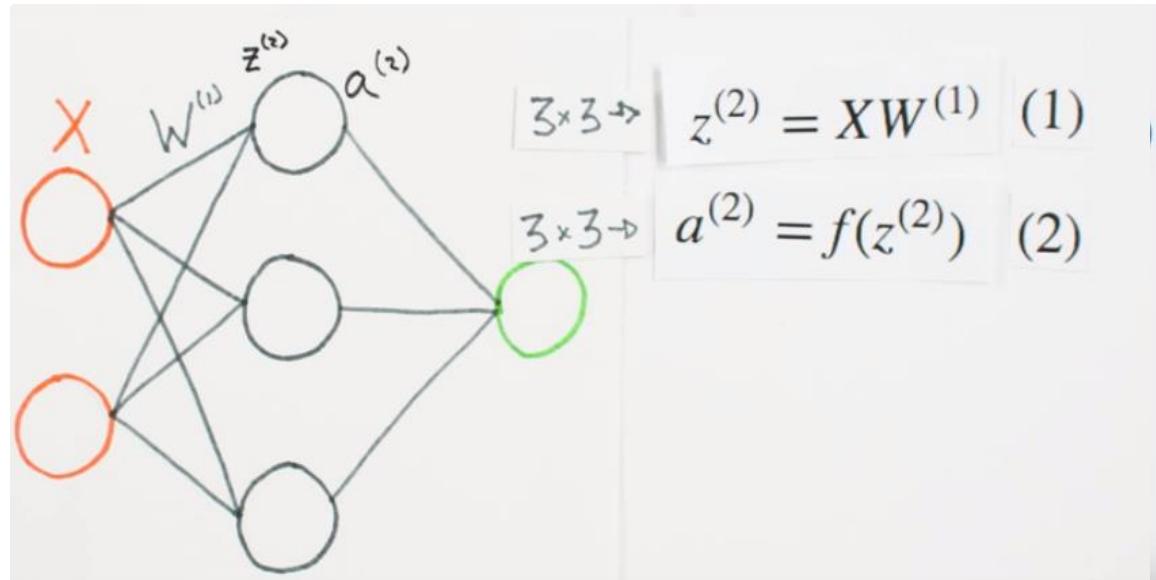
MULTI-NEURON NETWORKS :: FORWARD PROPAGATION



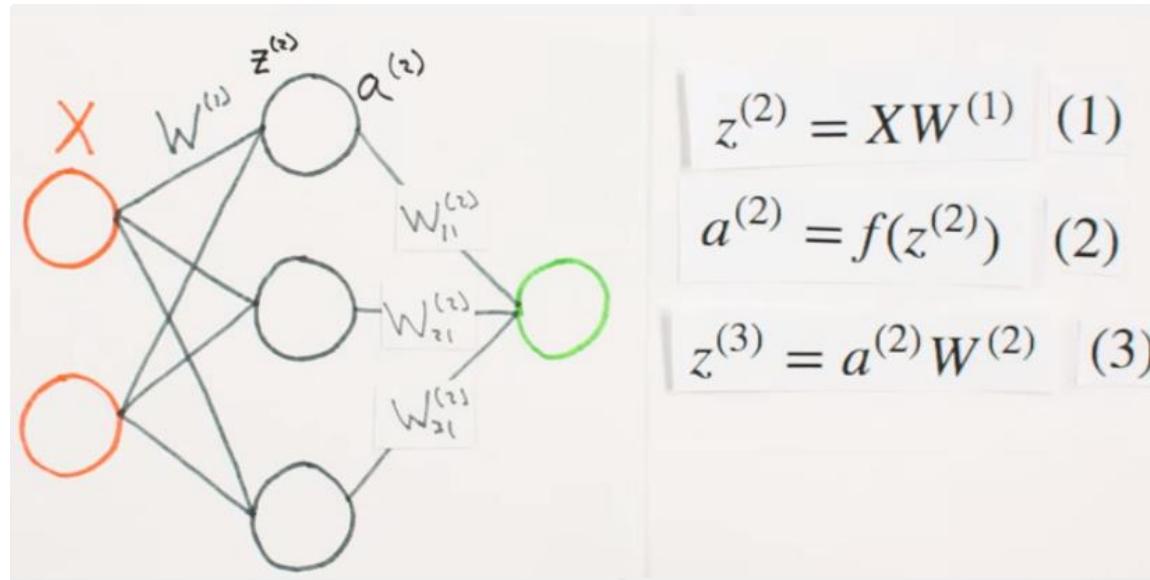
MULTI-NEURON NETWORKS :: FORWARD PROPAGATION



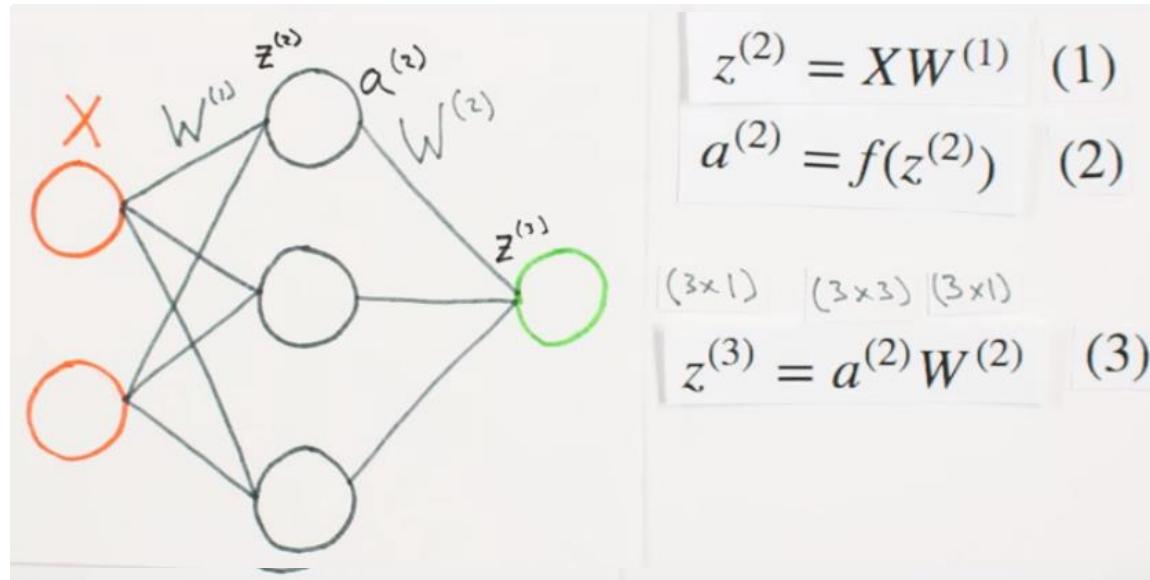
MULTI-NEURON NETWORKS :: FORWARD PROPAGATION



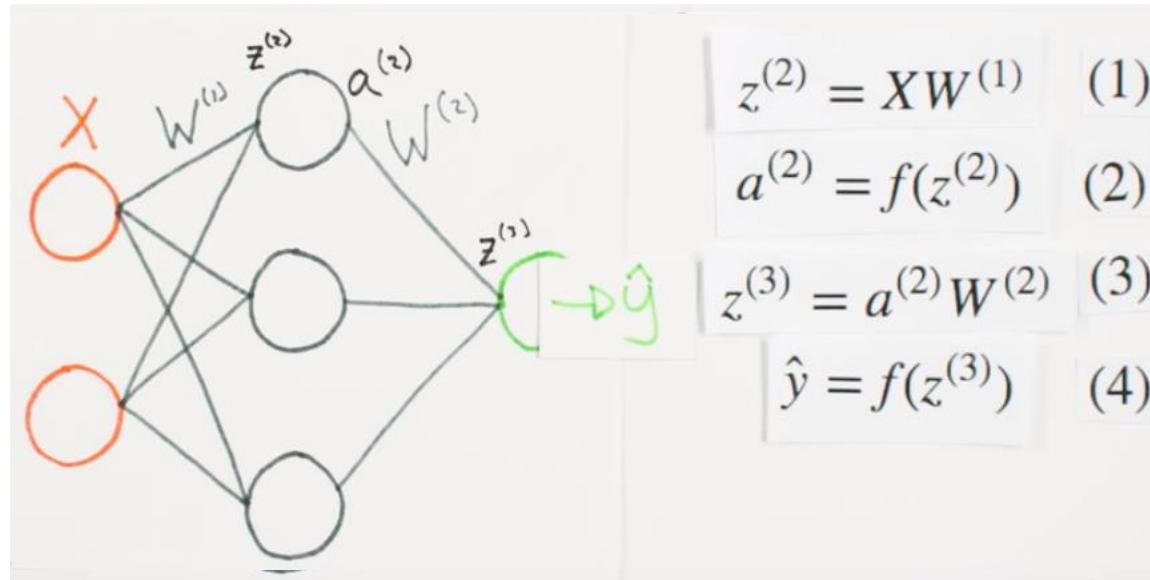
MULTI-NEURON NETWORKS :: FORWARD PROPAGATION



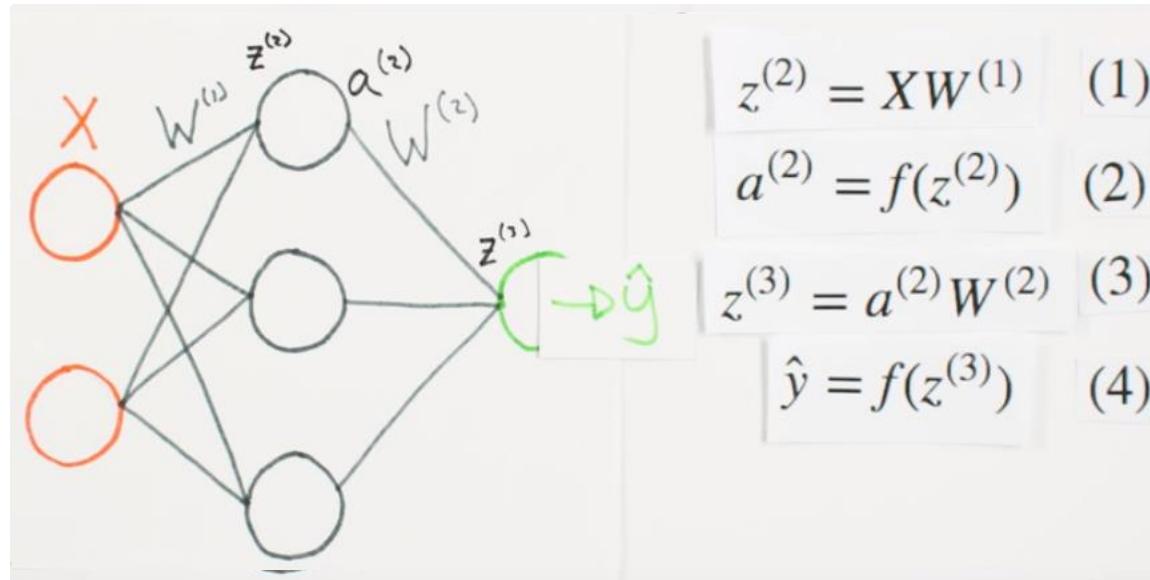
MULTI-NEURON NETWORKS :: FORWARD PROPAGATION



MULTI-NEURON NETWORKS :: FORWARD PROPAGATION



MULTI-NEURON NETWORKS :: FORWARD PROPAGATION



RESOURCES

- <https://playground.tensorflow.org/>
- <https://betterexplained.com/articles/derivatives-product-power-chain/>