Perceptron & Neural Networks

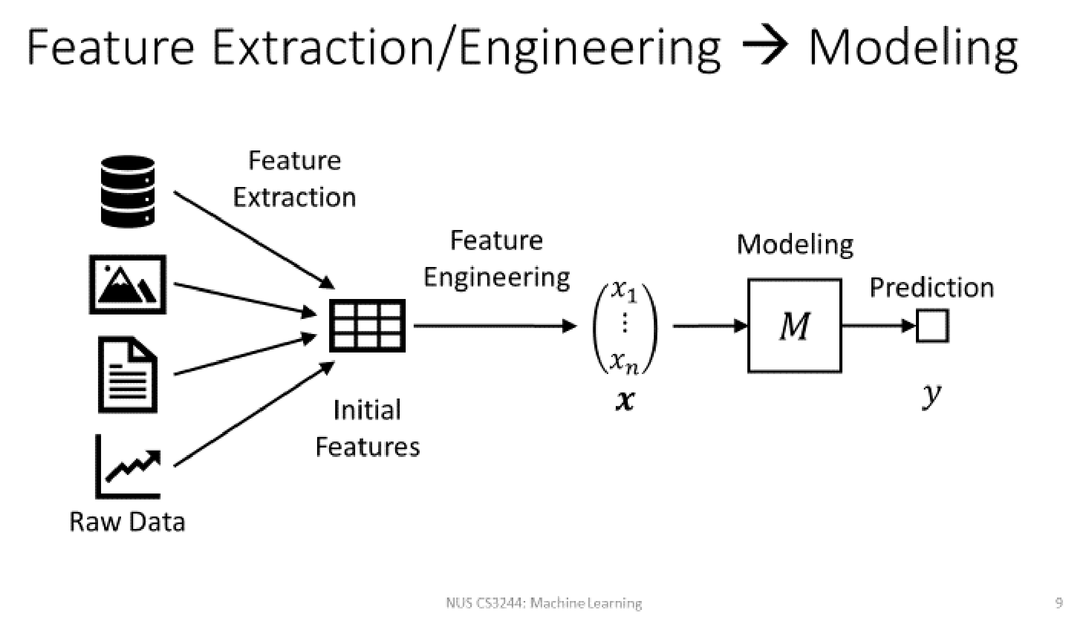
9

A

CS 3244

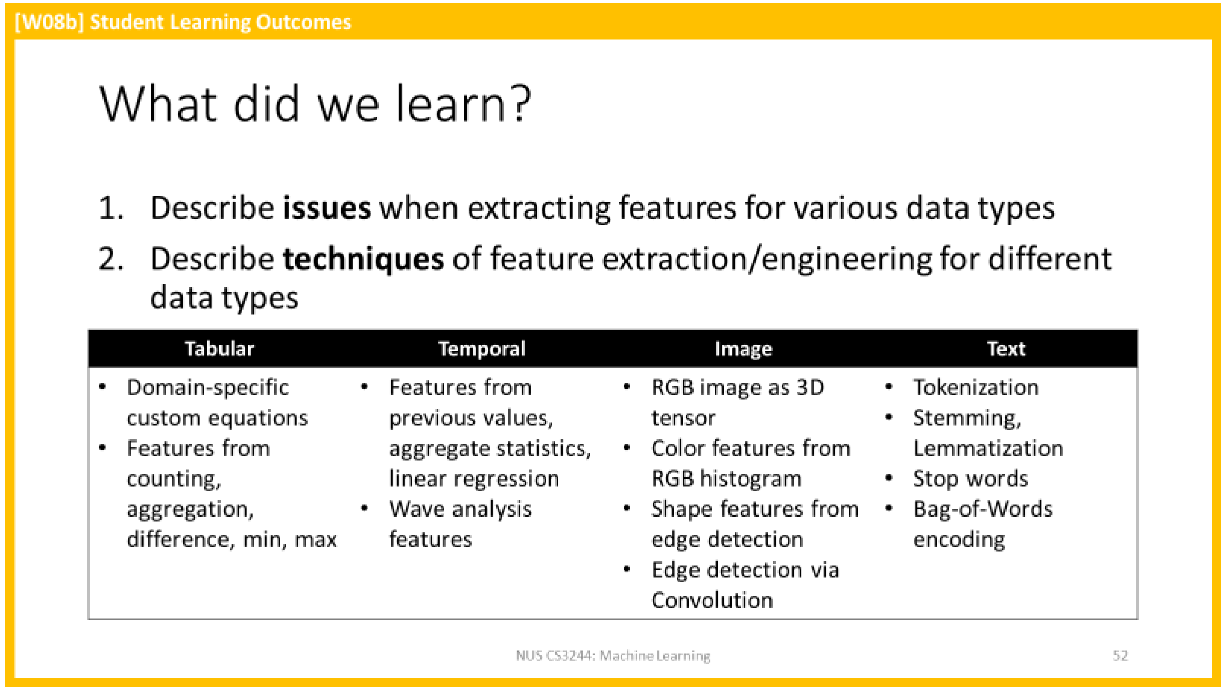
Machine Learning

Recap from W08b



Feature Extraction/Engineering → Modeling

NUS CS3244: Machine Learning 9



[W08b] Student Learning Outcomes

What did we learn?

1. Describe **issues** when extracting features for various data types
2. Describe **techniques** of feature extraction/engineering for different data types

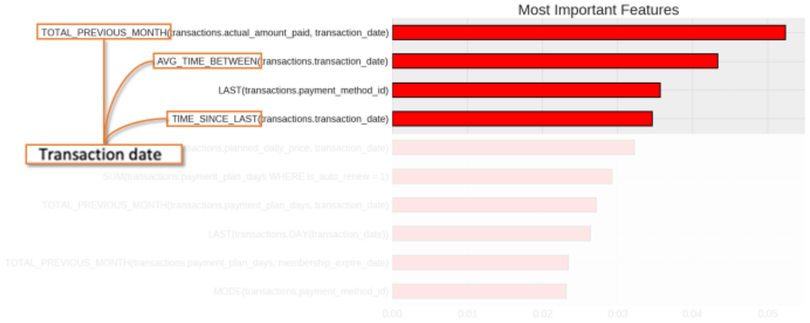
Tabular	Temporal	Image	Text
• Domain-specific custom equations • Features from counting, aggregation, difference, min, max	• Features from previous values, aggregate statistics, linear regression • Wave analysis features	• RGB image as 3D tensor • Color features from RGB histogram • Shape features from edge detection • Edge detection via Convolution	• Tokenization • Stemming, Lemmatization • Stop words • Bag-of-Words encoding

NUS CS3244: Machine Learning 52

Recap from W08b

Tabular Feature Engineering:

Counting, Aggregation, Difference, Min, Max



Source: https://github.com/Featuretools/predict-customer-churn

NUS CS3244: Machine Learning 4

Sliding Time Window

- Prediction Task: **Price Prediction**
- Features
  - Moving Average
  - Moving Standard Deviation
  - Moving Range (Min, Max)
  - Moving Trend (Slope of linear fit)

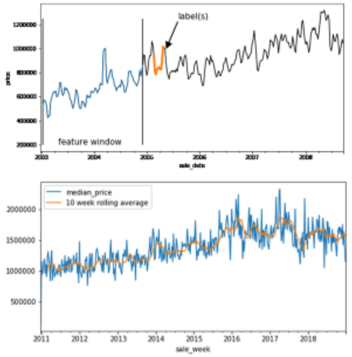


Image credit: https://cloud.google.com/blog/products/ai-machine-learning/how-to-quickly-solve-machine-learning-forecasting-problems-using-pandas-and-bigquery

NUS CS3244: Machine Learning 19

Feature: Edge Detection Kernels (2D)

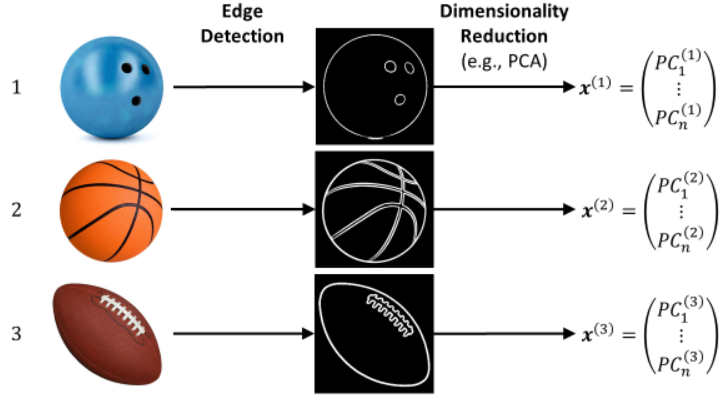
$$\frac{\partial c}{\partial p_x} \approx \frac{c_{(x+1,y)} - c_{(x-1,y)}}{p_{(x+1,y)} - p_{(x-1,y)}} \qquad \frac{\partial c}{\partial p_y} \approx \frac{c_{(x,y+1)} - c_{(x,y-1)}}{p_{(x,y+1)} - p_{(x,y-1)}}$$

$$I_{p_x} = \begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix} \qquad I_{p_y} = \begin{pmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}$$

$$\nabla c = \frac{\partial c}{\partial p_x} + \frac{\partial c}{\partial p_y} \sim \left(I_{p_x} + I_{p_y}\right) * \boldsymbol{x}$$

NUS CS3244: Machine Learning

Feature: Shape Feature Vector



$$\boldsymbol{x}^{(1)} = \begin{pmatrix} PC_1^{(1)} \\ \vdots \\ PC_n^{(1)} \end{pmatrix} \qquad \boldsymbol{x}^{(2)} = \begin{pmatrix} PC_1^{(2)} \\ \vdots \\ PC_n^{(2)} \end{pmatrix} \qquad \boldsymbol{x}^{(3)} = \begin{pmatrix} PC_1^{(3)} \\ \vdots \\ PC_n^{(3)} \end{pmatrix}$$

NUS CS3244: Machine Learning

Bag-of-Words (BOW) Encoding

1. Preprocess string s to array of words $\boldsymbol{w}$
2. Array of words → One-hot vector (fixed length)
3. $\mathrm{BOW}(\boldsymbol{w}) \rightarrow \boldsymbol{x}$
4. Problem: high dimensions if many words

$$\boldsymbol{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_{13} \end{pmatrix} \qquad x^{(1)} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \qquad x^{(2)} = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

#	Original Text s	Pre-Processed Words $\boldsymbol{w}$	chicken	wings	amazing	honestly	but	way	too	long	wait	not	worth	salty	expensive
1	"Chicken wings were amazing honestly"	['chicken', 'wings', 'amazing', 'honestly']	1	1	1	1	0	0	0	0	0	0	0	0	0
2	"Amazing wings, but waaaay to long to wait."	['amazing', 'wings', 'but', 'way', 'too', 'long', 'wait']	0	1	1	0	1	1	1	1	1	0	0	0	0
3	"Not worth it! Too salty chicken and expensive!"	['not', 'worth', 'too', 'salty', 'chicken', 'expensive']	1	0	0	0	0	0	1	0	0	1	1	1	1

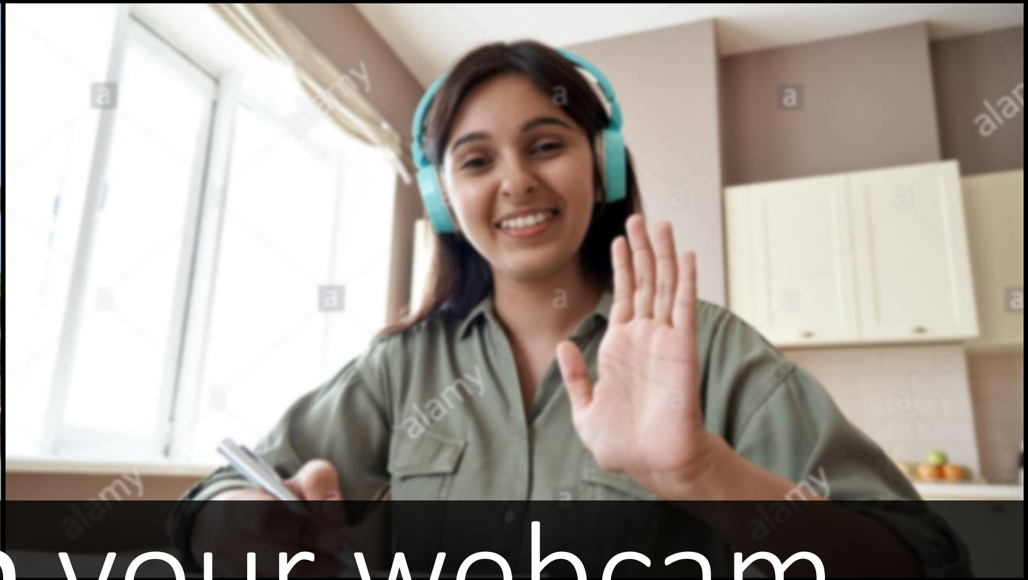
The **word "too"** could predict negative sentiment 46

Week 09A: Learning Outcomes

1. Describe the **structure** of Perceptrons and how it performs classification
2. Understand how Perceptrons are **trained** with the PLA
3. Understand how to **compose** multiple Perceptrons into a Neural Network
4. Describe how Neural Networks are **trained** with gradient descent and backpropagation

Week 09A: Lecture Outline

1. Perceptron
2. Activation Functions
3. Gradient Descent
4. Neural Networks
5. Backpropagation



Please **turn on** your webcam

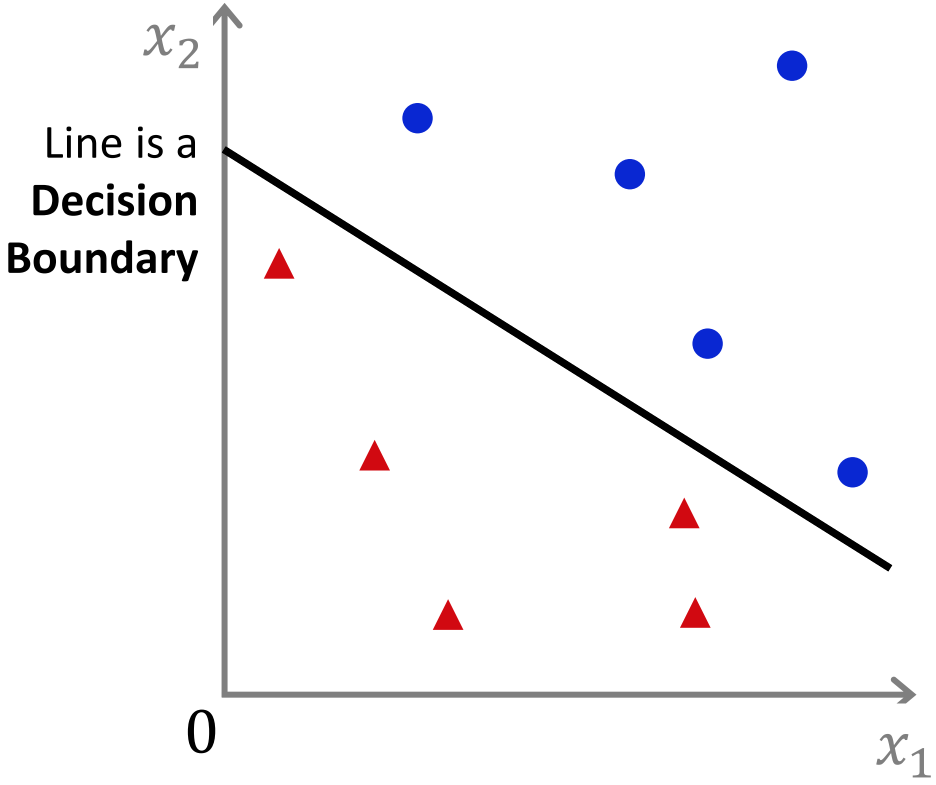
Mystery Student

AI
HISTORY

Perceptron

Linear Classifiers

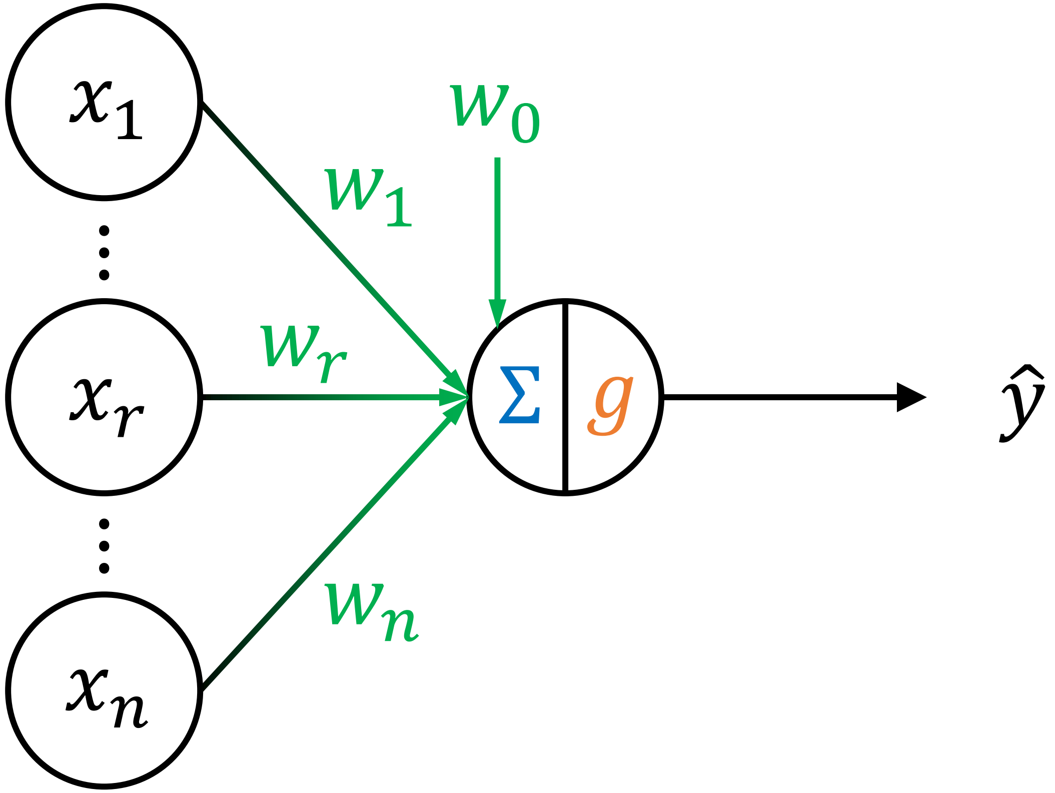
- Logistic Regression [W04A]
- Linear SVM [W04B]
- Perceptron



Perceptron

- What is a perceptron?
- How to train it?

Perceptron



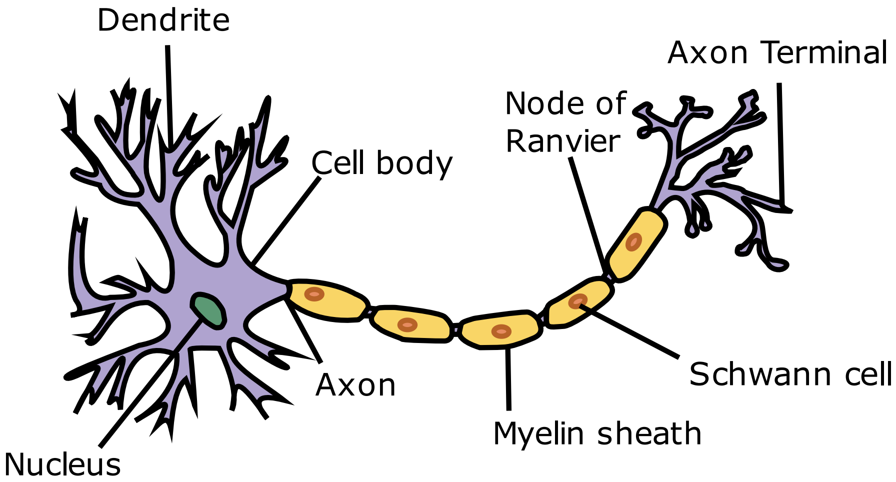


Diagram credits: Dhp1080 - Own work, CC BY-SA 3.0 via Wikimedia Commons.

Line Equation

$$x_2 = mx_1 + c$$

$$w_2 x_2 + w_1 x_1 + w_0 = 0$$

$$\sum_{r=0}^{n} w_r x_r = 0 \quad x_0 = 1$$

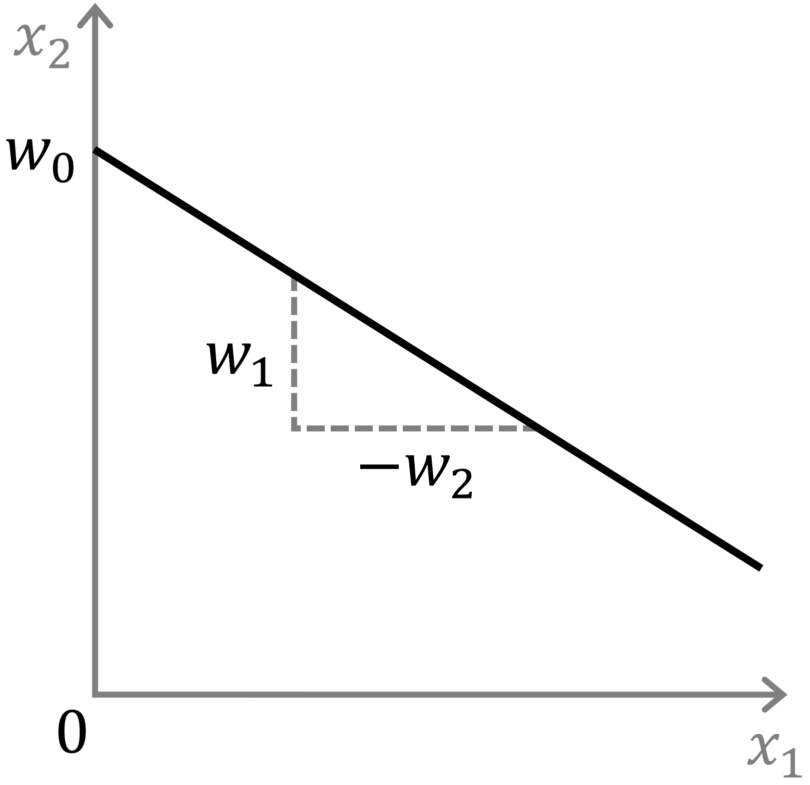


Image credit: https://en.wikipedia.org/wiki/Sign_function

Linear Classification

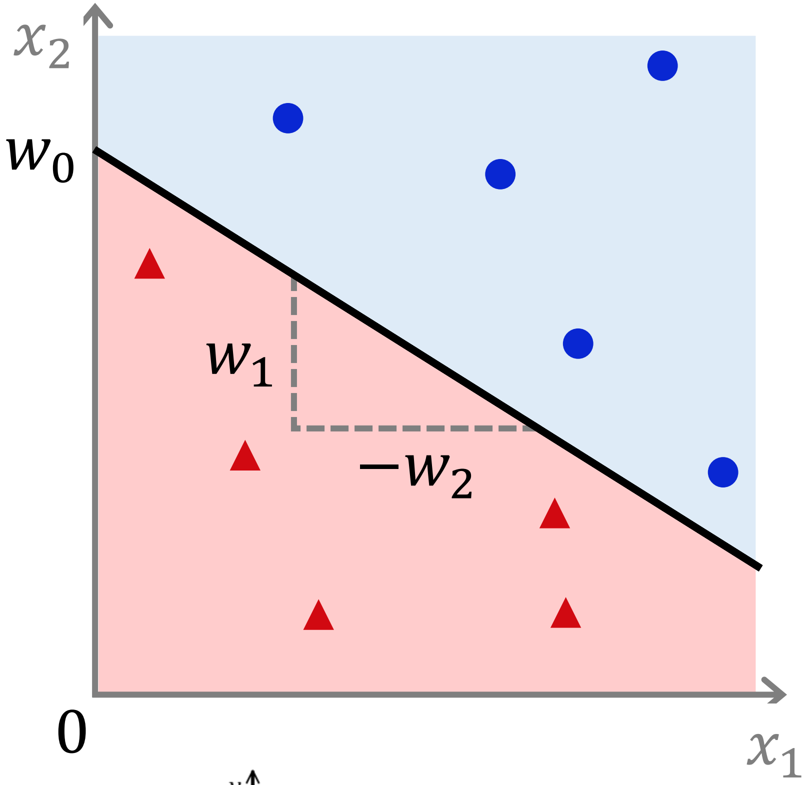
$$x_2 = mx_1 + c$$

$$w_2x_2 + w_1x_1 + w_0 = 0$$

$$\sum\nolimits_{r=0}^{n} w_r x_r = 0 \quad x_0 = 1$$

$$\color{blue}{\sum\nolimits_{r=0}^{n} w_r x_r > 0} \qquad \color{red}{\sum\nolimits_{r=0}^{n} w_r x_r \leq 0}$$

$$\hat{y} = \operatorname{sgn}\left(\sum\nolimits_{r=0}^{n} w_r x_r\right) \ , \operatorname{sgn}(z) = \begin{cases} +1 & z > 0 \\ -1 & z \leq 0 \end{cases}$$



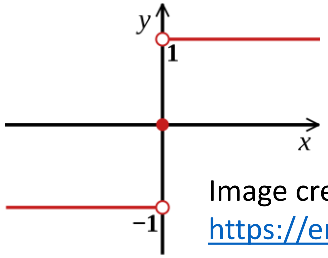
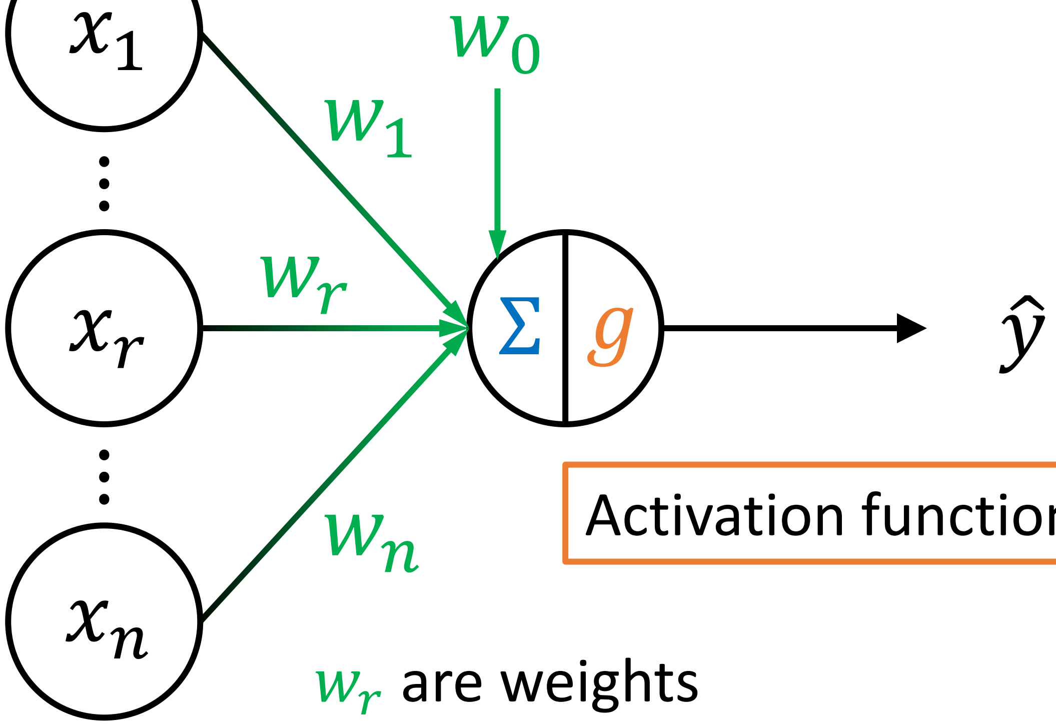


Image credit: https://en.wikipedia.org/wiki/Sign_function

Perceptron

Don’t forget the **bias term** w_0



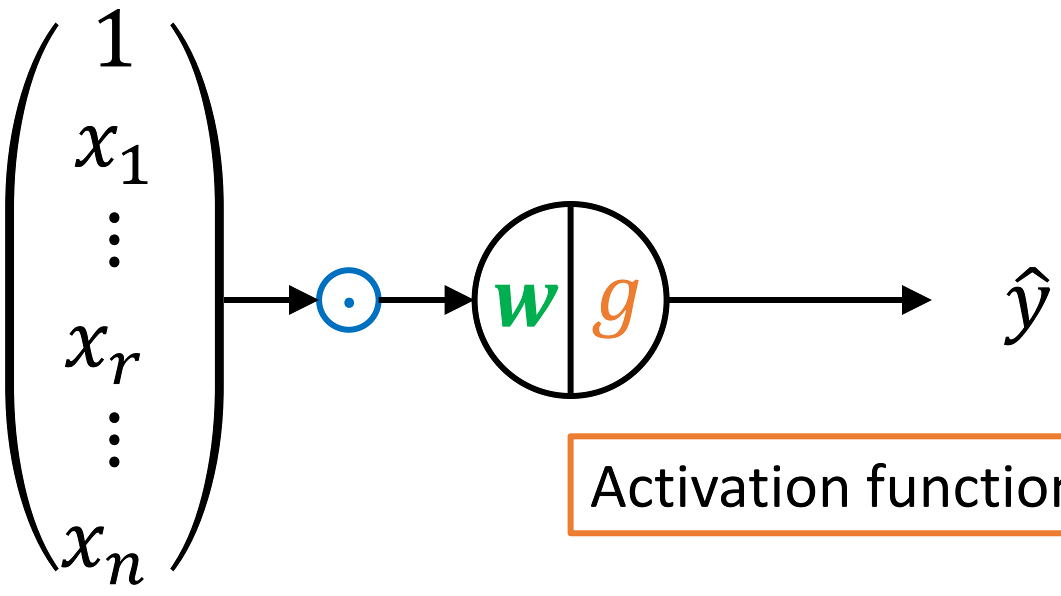
$$\hat{y} = \text{sgn}\left(\sum_{r=0}^{n} w_r x_r\right)$$

Activation function $g = \text{sgn}$

w_r are weights

x_r are inputs

Perceptron Classification


$\begin{pmatrix} 1 \\ x_1 \\ \vdots \\ x_r \\ \vdots \\ x_n \end{pmatrix}$
$\odot$
$\boldsymbol{w}$
g
$\hat{y}$

$\hat{y} = \mathrm{sgn}(\boldsymbol{w} \cdot \boldsymbol{x}) = \mathrm{sgn}(\boldsymbol{w}^\top \boldsymbol{x})$

Activation function $g = \mathrm{sgn}$

Perceptron Learning Algorithm (PLA)

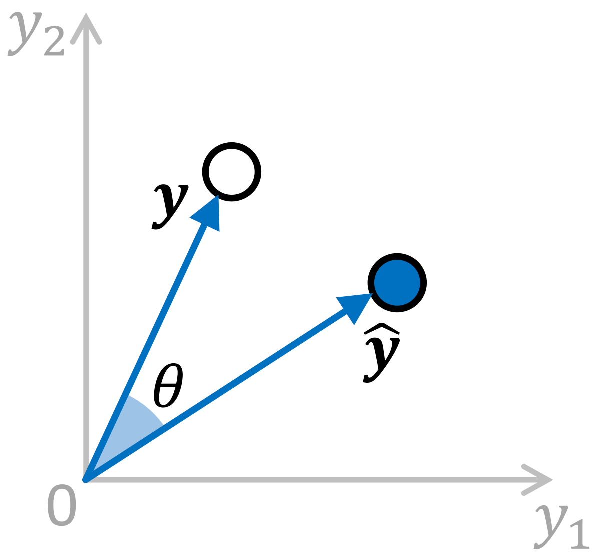
1. Initialize weights $\boldsymbol{w}$
   - Could be all zero, or random small values
2. For each instance i with features $\boldsymbol{x}^{(i)}$
   - Classify $\hat{y}^{(i)} = \operatorname{sgn}\left(\boldsymbol{w}^\top \boldsymbol{x}^{(i)}\right)$
3. Select one misclassified instance
   - Update weights: $\boldsymbol{w} \leftarrow \boldsymbol{w} + \boxed{\Delta \boldsymbol{w}}$
4. Iterate steps 2 to 3 until
   - Convergence (classification error < threshold), or
   - Maximum number of iterations

How to calculate?
- What direction?
- What magnitude?

Perceptron Weight Update

$$\underset{\text{New weight}}{\boldsymbol{w}} \leftarrow \overset{\text{Old weight}}{\boldsymbol{w}} + \underset{\text{Learning Rate}}{\eta}\left(y \overset{\text{Learning Error}}{-} \hat{y}\right)\boldsymbol{x}$$

Vector Distances and Similarity



Cosine Similarity

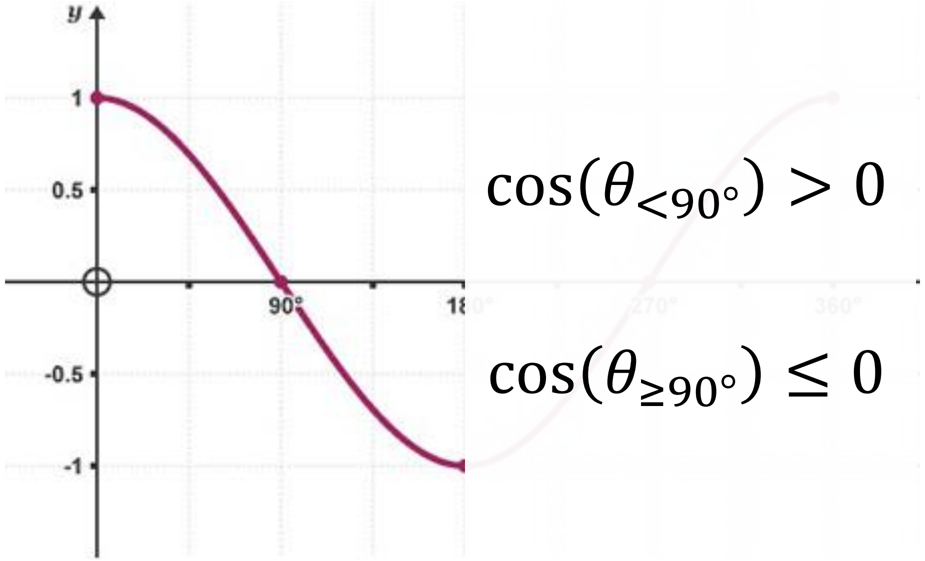
$$s = \cos(\theta) = \frac{\hat{\boldsymbol{y}} \cdot \boldsymbol{y}}{\|\hat{\boldsymbol{y}}\| \|\boldsymbol{y}\|}$$

Angular Distance

$$\theta = \cos^{-1}(s)$$

Vector Distances and Similarity

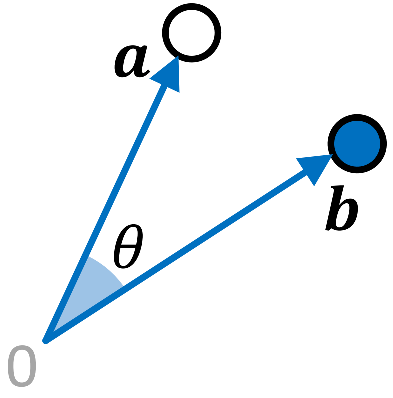
Cosine Curve



$\cos(\theta_{<90^\circ}) > 0$

$\cos(\theta_{\geq 90^\circ}) \leq 0$

Image credit: https://www.open.edu/openlearn/ocw/pluginfile.php/947914/mod_oucontent/oucontent/48949/9eaffc43/9f8315d5/mfs_w4_fig4.jpg



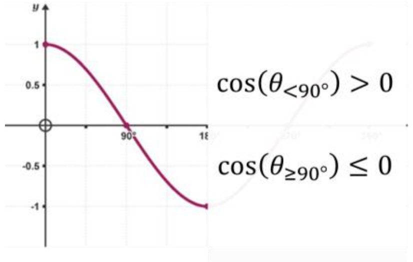
Cosine Similarity

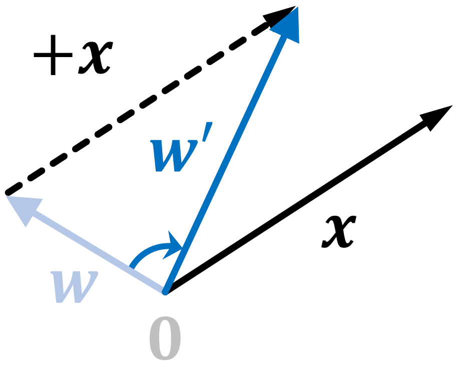
$$s = \cos(\theta) = \frac{\boldsymbol{a} \cdot \boldsymbol{b}}{\|\boldsymbol{a}\|\|\boldsymbol{b}\|}$$

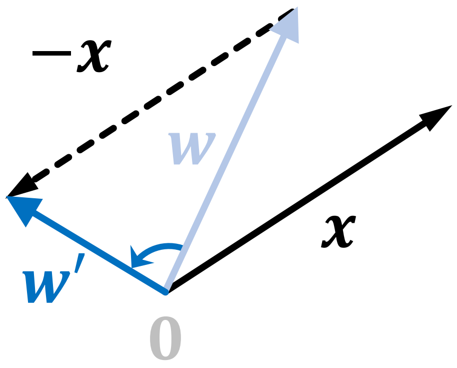
Angular Distance

$$\theta = \cos^{-1}(s)$$

Perceptron Weight Update







Consider this misclassification:

$$y = +1, \qquad \hat{y} = \text{sgn}(\boldsymbol{w} \cdot \boldsymbol{x}) = -1$$

- $\hat{y} = -1 \Rightarrow \boldsymbol{w} \cdot \boldsymbol{x} \leq 0 \Rightarrow \theta = \cos^{-1}\left(\frac{\boldsymbol{w}}{|\boldsymbol{w}|} \cdot \frac{\boldsymbol{x}}{|\boldsymbol{x}|}\right) \geq 90°$

But we want

- $\hat{y} = +1 \Rightarrow \boldsymbol{w} \cdot \boldsymbol{x} > 0 \Rightarrow \theta < 90°$
- i.e., $\boldsymbol{w}$ to point in a more similar direction as $\boldsymbol{x}$

Adding $\boldsymbol{x}$ to $\boldsymbol{w}$ will make a more positive result, i.e.,

$$\boldsymbol{w}' = \boldsymbol{w} + \boldsymbol{x}$$

Consider this misclassification:

$$y = -1, \qquad \hat{y} = \text{sgn}(\boldsymbol{w} \cdot \boldsymbol{x}) = +1$$

- $\hat{y} = +1 \Rightarrow \boldsymbol{w} \cdot \boldsymbol{x} > 0 \Rightarrow \theta = \cos^{-1}\left(\frac{\boldsymbol{w}}{|\boldsymbol{w}|} \cdot \frac{\boldsymbol{x}}{|\boldsymbol{x}|}\right) < 90°$

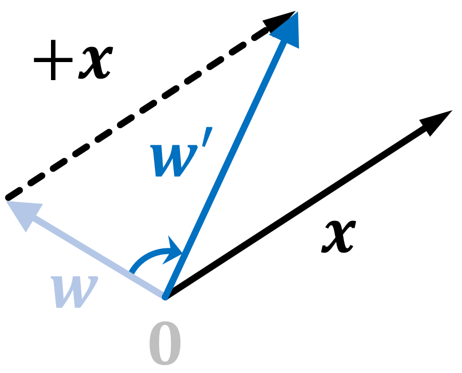
But we want

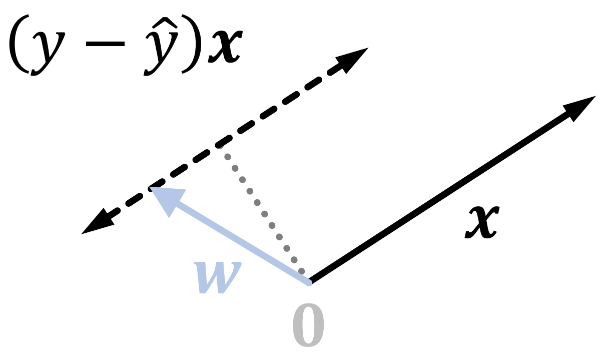
- $\hat{y} = -1 \Rightarrow \boldsymbol{w} \cdot \boldsymbol{x} \leq 0 \Rightarrow \theta > 90°$
- i.e., $\boldsymbol{w}$ to point in a less similar direction as $\boldsymbol{x}$

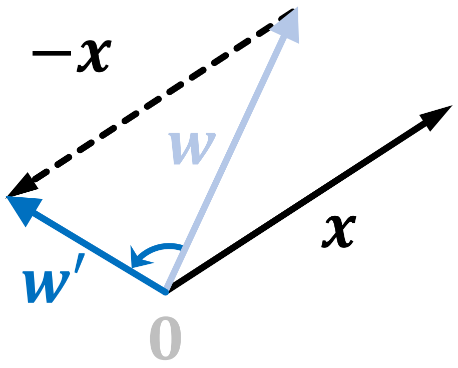
Negating $\boldsymbol{x}$ from $\boldsymbol{w}$ will make a less positive result, i.e.,

$$\boldsymbol{w}' = \boldsymbol{w} - \boldsymbol{x}$$

Perceptron Weight Update







$$y = +1$$
$$\hat{y} = -1$$

$$y - \hat{y} = +2$$
$$\Delta\mathbf{w} = +2\eta\mathbf{x}$$

$$\underbrace{\mathbf{w}}_{\text{New weight}} \leftarrow \overbrace{\mathbf{w}}^{\text{Old weight}} + \underbrace{\eta}_{\text{Learning Rate}}\overbrace{(y - \hat{y})}^{\text{Learning Error}}\mathbf{x}$$

$$y = -1$$
$$\hat{y} = +1$$

$$y - \hat{y} = -2$$
$$\Delta\mathbf{w} = -2\eta\mathbf{x}$$

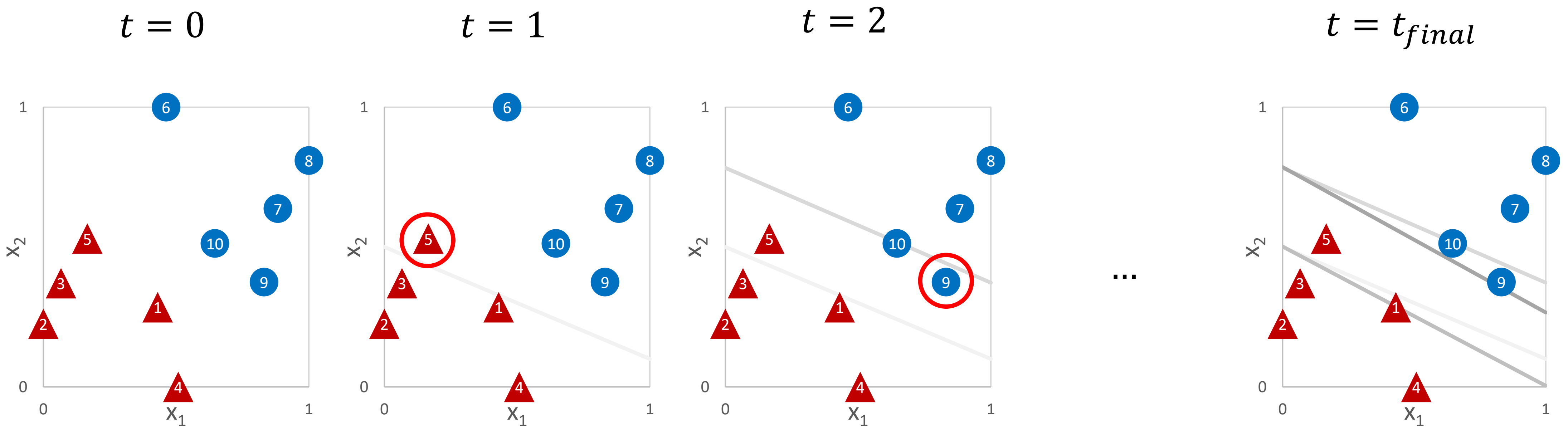
Perceptron Learning Algorithm

1. Initialize weights $\boldsymbol{w}$
   - Could be all zero, or random small values
2. For each instance i with features $\boldsymbol{x}^{(i)}$
   - Classify $\hat{y}^{(i)} = \operatorname{sgn}\left(\boldsymbol{w}^\top \boldsymbol{x}^{(i)}\right)$
3. Select one misclassified instance
   - Update weights: $\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta(y - \hat{y})\boldsymbol{x}$
4. Iterate steps 2 to 3 until
   - Convergence (classification error < threshold), or
   - Maximum number of iterations

$$\begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_r \\ \vdots \\ w_n \end{pmatrix} \leftarrow \begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_r \\ \vdots \\ w_n \end{pmatrix} + \eta(y - \hat{y}) \begin{pmatrix} 1 \\ x_1 \\ \vdots \\ x_r \\ \vdots \\ x_n \end{pmatrix}$$

$$w_r \leftarrow w_r + \eta(y - \hat{y})x_r$$

Perceptron Learning Algorithm in Action



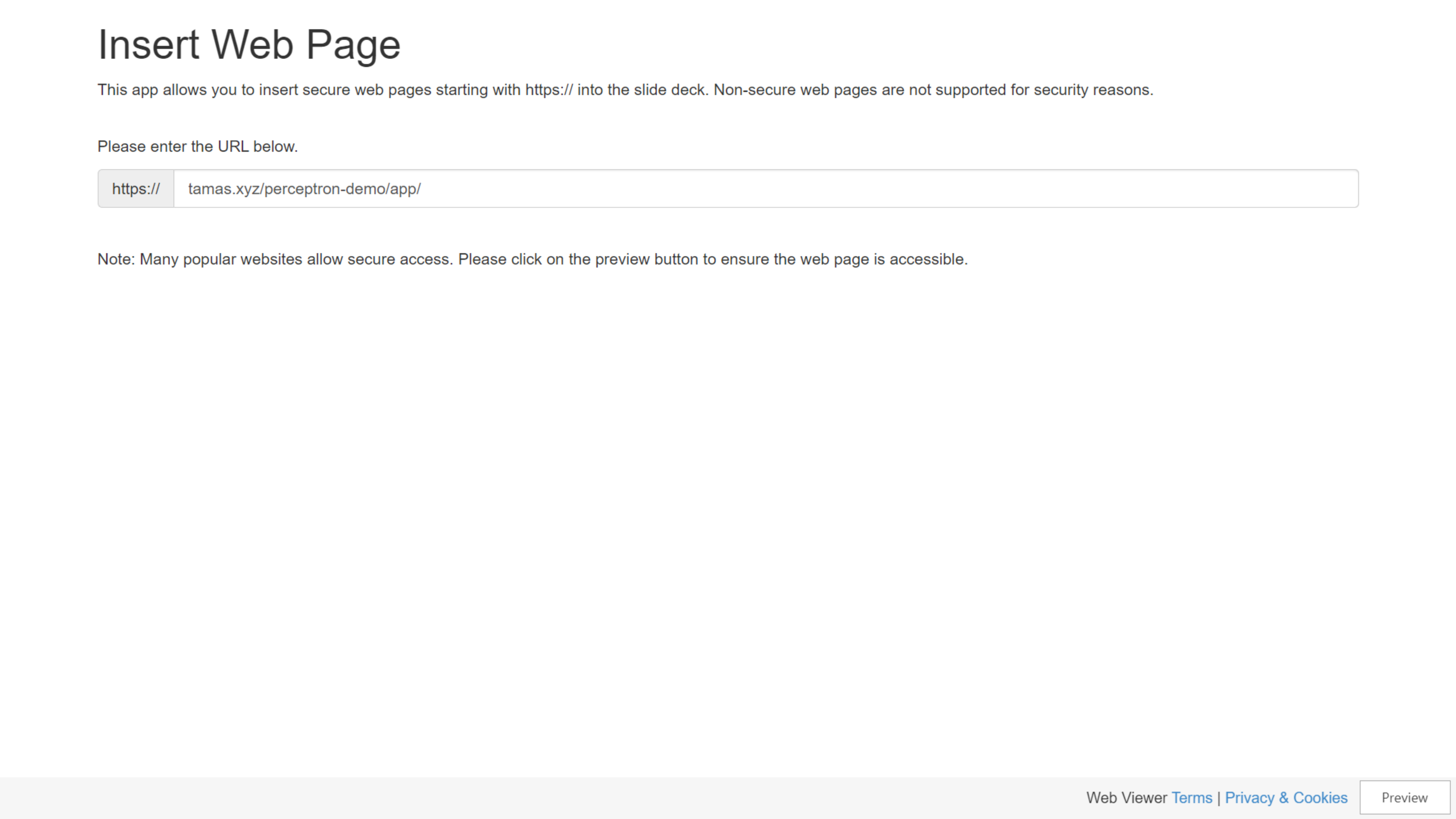
$t = 0$

$t = 1$

$t = 2$

…

$t = t_{final}$



Insert Web Page

This app allows you to insert secure web pages starting with https:// into the slide deck. Non-secure web pages are not supported for security reasons.

Please enter the URL below.

https:// tamas.xyz/perceptron-demo/app/

Note: Many popular websites allow secure access. Please click on the preview button to ensure the web page is accessible.

E09a.1

What are the differences?
Perceptron vs. Linear SVM

In Slack #general

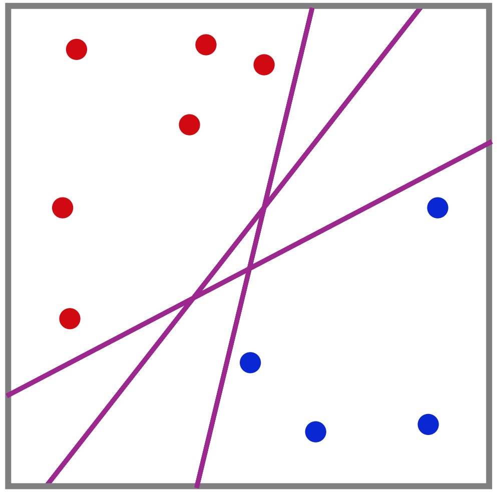
1. Write to thread to suggest feature
2. Emote (👍 :+1:) to vote for feature

Perceptron	**Linear Support Vector Machine (SVM)**

What are the differences?
Perceptron vs. Linear SVM

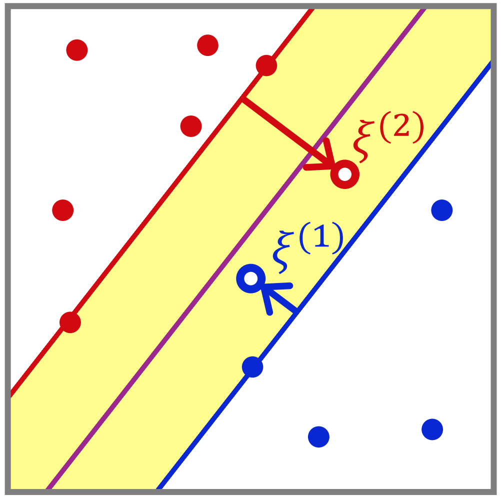
In Slack #general
1. Write to thread to suggest feature
2. Emote (👍 :+1:) to vote for feature

Perceptron



- Can select any model to linear => not robust (learns different weights for different initializations)
- Cannot converge on non-linearly separable data

Linear Support Vector Machine (SVM)



- Perceptron of "optimal stability"
- Maximizes margin
- Soft-margin: allows soft error => can learn from non-linearity separable data

Questions!

Extending the Perceptron

- Other **activation functions**
  - Differentiable ones!
- **Multiple** perceptrons
  - Multi-Layer Perceptron (MLP)
- **Non-linear** classifiers

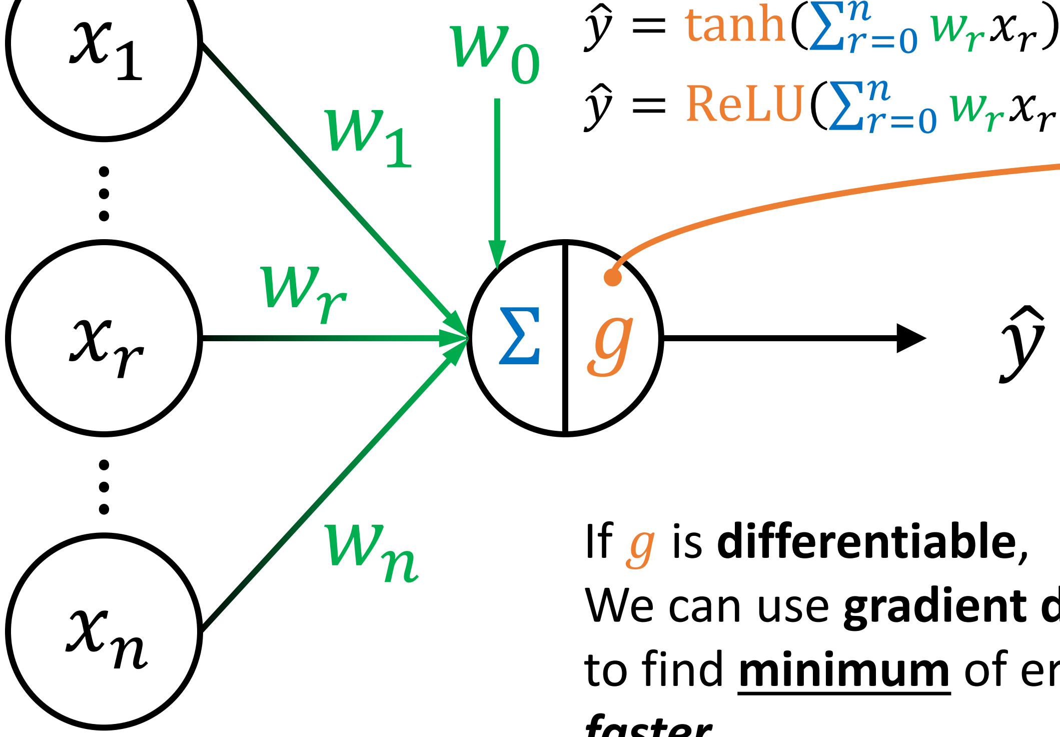
} Feed-forward Neural Network

Activation Functions

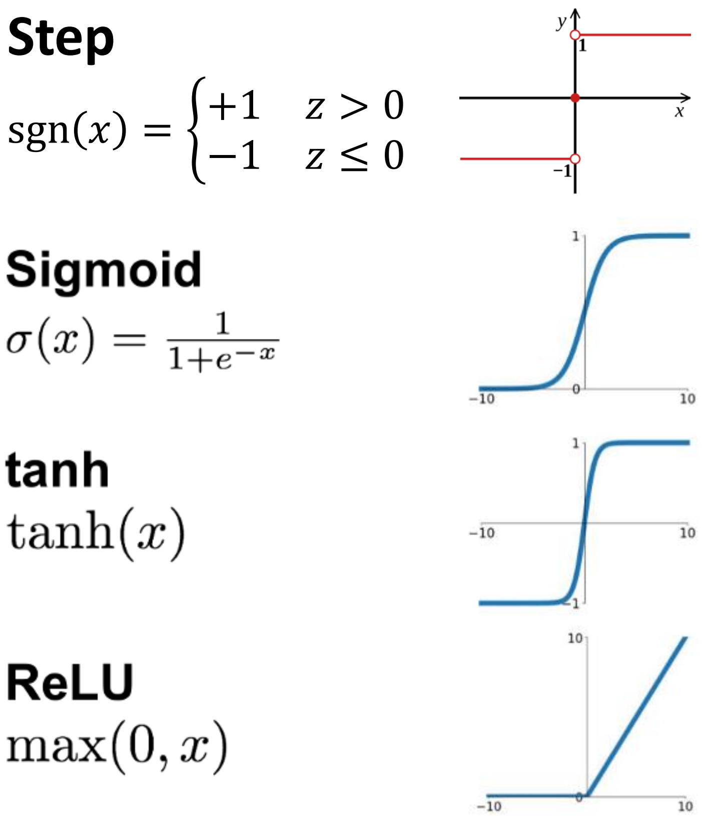
NUS National University of Singapore
Department of Computer Science
School of Computing

Differentiable Activation Functions

$$\hat{y} = \text{sgn}\left(\sum_{r=0}^{n} w_r x_r\right)$$
$$\hat{y} = \sigma\left(\sum_{r=0}^{n} w_r x_r\right)$$
$$\hat{y} = \tanh\left(\sum_{r=0}^{n} w_r x_r\right)$$
$$\hat{y} = \text{ReLU}\left(\sum_{r=0}^{n} w_r x_r\right)$$



If g is **differentiable,**
We can use **gradient descent**
to find **minimum** of error $\varepsilon = y - \hat{y}$
faster



Step

$$\text{sgn}(x) = \begin{cases} +1 & z > 0 \\ -1 & z \le 0 \end{cases}$$

Sigmoid

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

tanh

$$\tanh(x)$$

ReLU

$$\max(0, x)$$

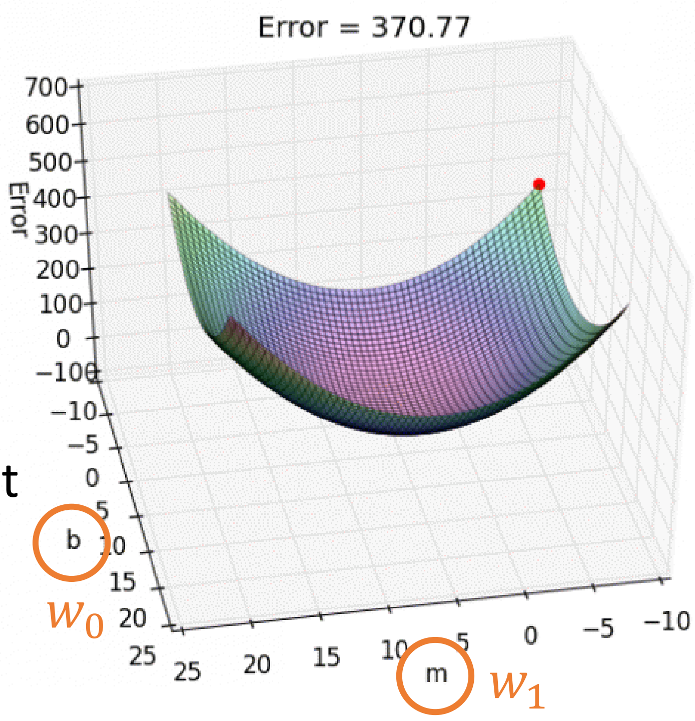
Image Credit:
https://miro.medium.com/max/1400/0*slJ-gbjlz0zrz8lb.png

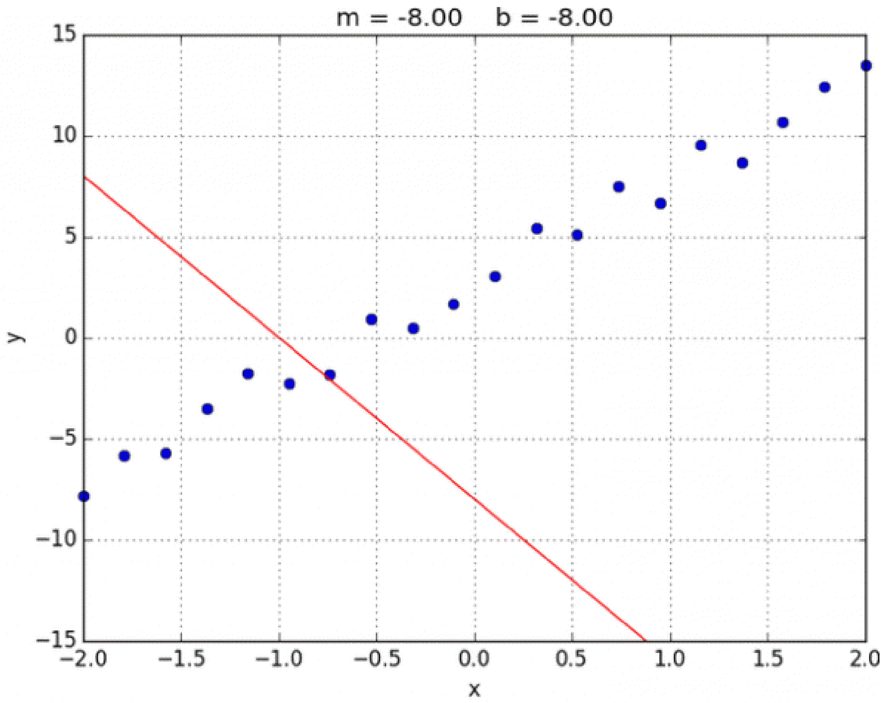
Gradient Descent

Optimization Goal:

Iteratively find w_r with minimum error ε

"Weight space"
To search for best parameter (weight) values

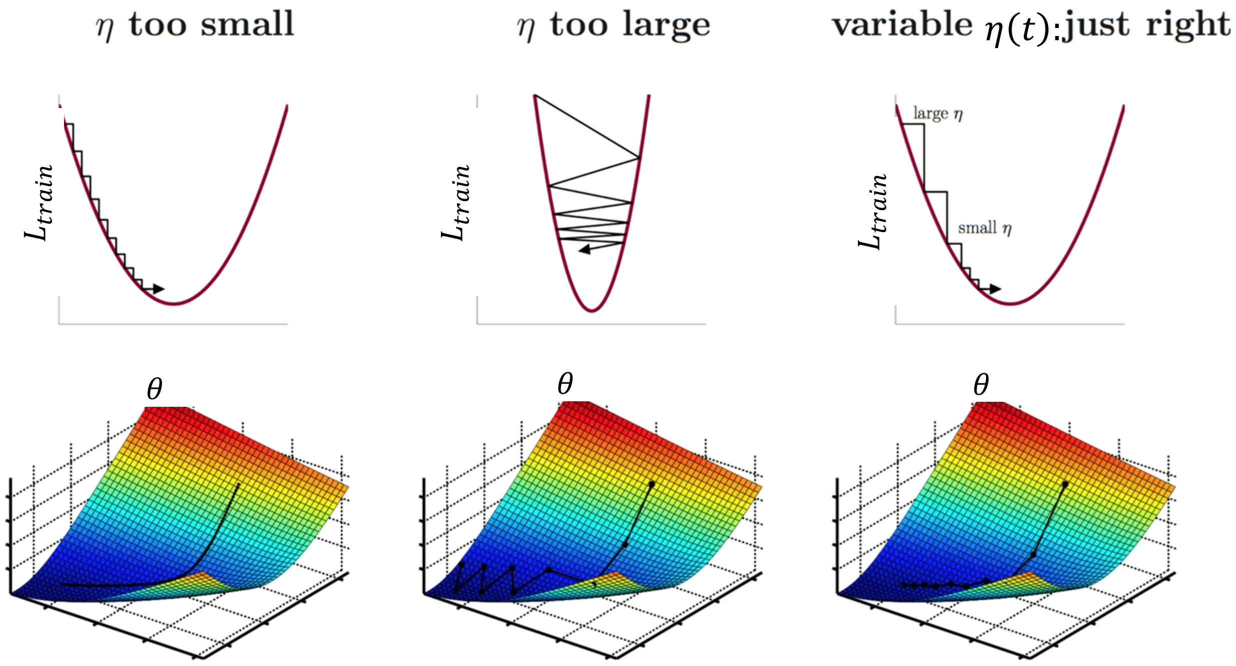




Credits: Alykhan Tejani's Medium Post

Iterative **steps** in direction towards (local) minimum

Learning Rate η



η too small

η too large

variable $\eta(t)$:just right

$\eta = 0.1$; 75 steps

$\eta = 2$; 10 steps

variable $\eta(t)$; 10 steps

These graphs are also in the “weight space”.

Perceptron Weight Update

$$\underset{\text{New weight}}{\boldsymbol{w}} \leftarrow \overset{\text{Old weight}}{\boldsymbol{w}} + \underset{\text{Learning Rate}}{\eta}\left(\overset{\text{Learning Error}}{y - \hat{y}}\right)\boldsymbol{x}$$

Gradient Descent Weight Update

$$\underset{\text{New weight}}{\boldsymbol{w}} \leftarrow \overset{\text{Old weight}}{\boldsymbol{w}} - \underset{\text{Learning Rate}}{\eta}\overset{\text{Learning Error}}{\nabla\varepsilon}$$

MSE error

$$\varepsilon = \frac{1}{2}(\hat{y} - y)^2$$

Gradient of error

$$\nabla\varepsilon = \frac{\partial\varepsilon}{\partial\boldsymbol{w}} = \begin{pmatrix} \partial\varepsilon/\partial w_1 \\ \vdots \\ \partial\varepsilon/\partial w_r \\ \vdots \\ \partial\varepsilon/\partial w_n \end{pmatrix}$$

Chain Rule

Consider composite function

$$g(x) = g\big(f(x)\big)$$

$$g = g(f), f = f(x)$$

$$g'(x) = \frac{dg}{dx} = \frac{dg}{df}\frac{df}{dx}$$

Intuition

Rate of change of g relative to x is the product of

- rates of change of g relative to f and
- rates of change of f relative to x

“If

- *a car travels 2x fast as a bicycle and*
- *the bicycle is 4x as fast as a walking man,*

then the car travels 2 × 4 = 8 times as fast as the man.”

– George F. Simmons, Calculus with Analytic Geometry (1985)

Chain Rule

Consider composite function

$$h(x) = h\left(g\left(f(x)\right)\right)$$

$$h = h(g), g = g(f), f = f(x)$$

$$h'(x) = \frac{dh}{dx} = \frac{dh}{dg}\frac{dg}{df}\frac{df}{dx}$$

Optimization Goal:

Iteratively find w_r with minimum error ε

$$\hat{y} = g\big(f(w_r)\big),\; f(w_r) = \textstyle\sum_{r=0}^{n} w_r x_r$$

$$\varepsilon = \frac{1}{2}(\hat{y} - y)^2$$ MSE error

"Weight space"

To search for best parameter (weight) values

$$\frac{d\varepsilon}{dw_r} = \frac{d\varepsilon}{d\hat{y}}\frac{d\hat{y}}{dw_r} = (\hat{y} - y)\left(\frac{dg}{df}\frac{df}{dw_r}\right)$$

$$\frac{dg}{df} = g'(f)$$

$$\frac{df}{dw_r} = \frac{d\left(\sum_{r=0}^{n} w_r x_r\right)}{dw_r} = \frac{d(w_r x_r)}{dw_r} + \sum_{\rho \neq r}^{n}\left(\frac{d\left(w_\rho x_\rho\right)}{dw_r}\right) = x_r$$

Weight Update

$$\boxed{w_r \leftarrow w_r - \eta\frac{d\varepsilon}{dw_r} = w_r - \eta(\hat{y} - y)g'(f)x_r}$$

What is the derivative of sigmoid function σ?

- $\sigma(x) = \frac{1}{1+e^{-x}}$
- Rewrite as compound function
  - $\sigma(z) = \frac{1}{1+z}, \; z(x) = e^{-x}$
- Using chain rule
  - $\sigma'(x) = \frac{dg}{dx} = \frac{dg}{dz}\frac{dz}{dx} = \frac{-1}{(1+z)^2}(-e^{-x}) = \frac{e^{-x}}{(1+e^{-x})^2} = \sigma(x)\frac{e^{-x}}{1+e^{-x}}$
- Notice: $1 - \sigma(x) = 1 - \frac{1}{1+e^{-x}} = \frac{1+e^{-x}-1}{1+e^{-x}} = \frac{e^{-x}}{1+e^{-x}}$
- Substituting back:
  - $\sigma'(x) = \sigma(x)\big(1 - \sigma(x)\big)$

Optimization Goal:

Iteratively find w_r with minimum error ε

$$\hat{y} = g\big(f(w_r)\big),\ f(w_r) = \textstyle\sum_{r=0}^{n} w_r x_r$$

$$\varepsilon = \frac{1}{2}(\hat{y} - y)^2$$

MSE error

$$\frac{d\varepsilon}{dw_r} = \frac{d\varepsilon}{d\hat{y}}\frac{d\hat{y}}{dw_r} = (\hat{y} - y)\left(\frac{dg}{df}\frac{df}{dw_r}\right)$$

$$\frac{dg}{df} = \sigma'(f) = \hat{y}(1 - \hat{y}) \qquad \frac{df}{dw_r} = x_r$$

Weight Update

$$\boxed{w_r \leftarrow w_r - \eta\frac{d\varepsilon}{dw_r} = w_r - \eta(\hat{y} - y)\hat{y}(1 - \hat{y})x_r}$$

Optimization Goal:

Iteratively find $\boldsymbol{w}$ with minimum error ε

$$\hat{y} = g\big(f(\boldsymbol{w})\big), \qquad f(\boldsymbol{w}) = \boldsymbol{w} \cdot \boldsymbol{x}$$

$$\varepsilon = \frac{1}{2}(\hat{y} - y)^2$$ MSE error

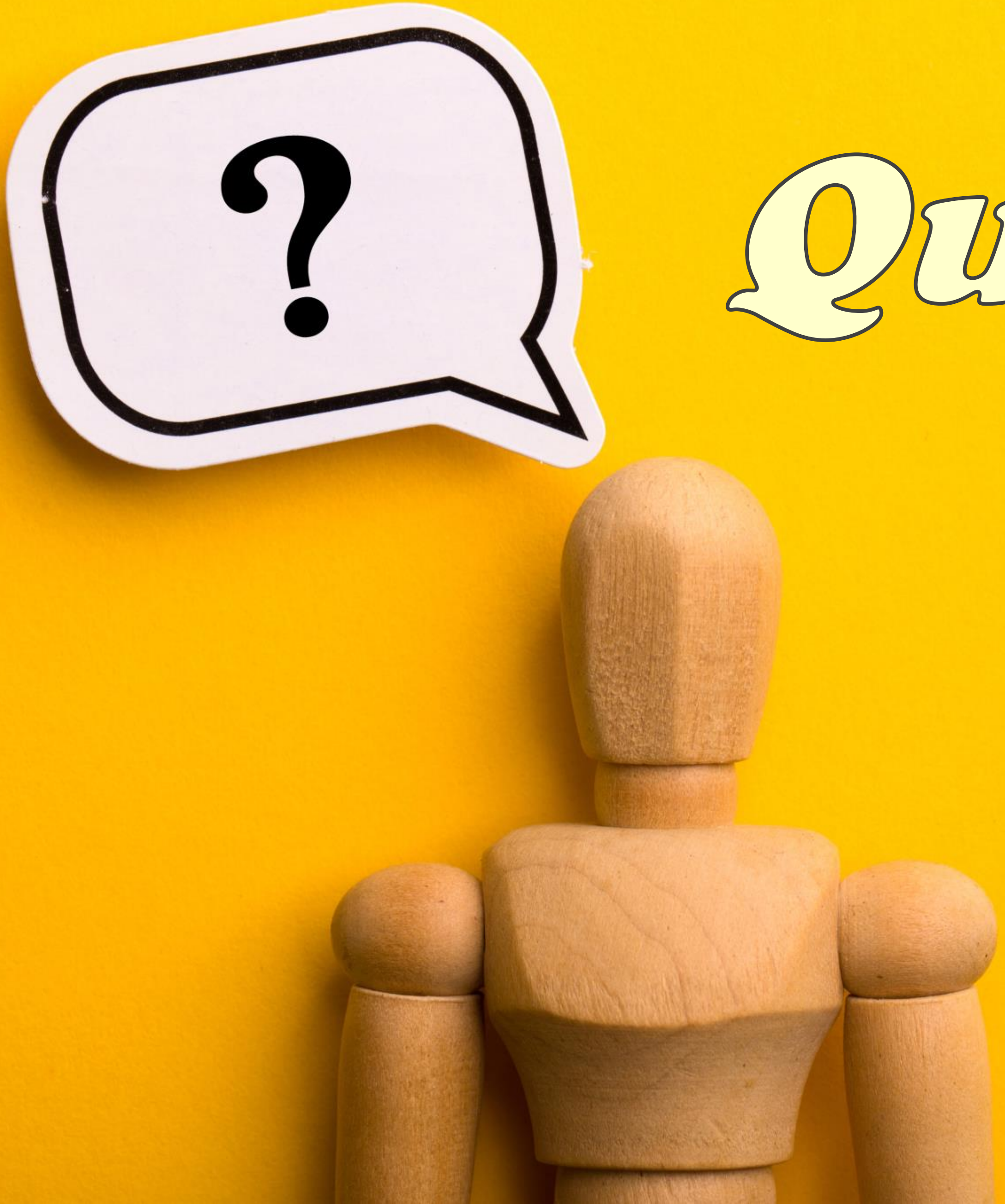
Gradient $$\nabla_{\boldsymbol{w}}\varepsilon = \frac{d\varepsilon}{d\boldsymbol{w}} = \frac{d\varepsilon}{d\hat{y}}\frac{d\hat{y}}{d\boldsymbol{w}} = (\hat{y} - y)\left(\frac{dg}{df}\frac{df}{d\boldsymbol{w}}\right)$$

$$\frac{dg}{df} = \sigma'(f) = \hat{y}(1 - \hat{y})$$

$$\nabla_{\boldsymbol{w}} f = \frac{df}{d\boldsymbol{w}} = \frac{d(\boldsymbol{w} \cdot \boldsymbol{x})}{d\boldsymbol{w}} = \begin{pmatrix} \vdots \\ d(w_r x_r)/dw_r \\ \vdots \end{pmatrix} = \begin{pmatrix} \vdots \\ x_r \\ \vdots \end{pmatrix} = \boldsymbol{x}$$

Weight Update

$$\boxed{\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta\frac{d\varepsilon}{d\boldsymbol{w}} = \boldsymbol{w} - \eta(\hat{y} - y)\hat{y}(1 - \hat{y})\boldsymbol{x}}$$



Questions!

Extending the Perceptron

- Other **activation functions**
  - Differentiable ones!
- **Multiple** perceptrons
  - Multi-Layer Perceptron (MLP)
- **Non-linear** classifiers

} Feed-forward Neural Network

Neural Networks

W09 Pre-Lecture Task (due before next Mon)

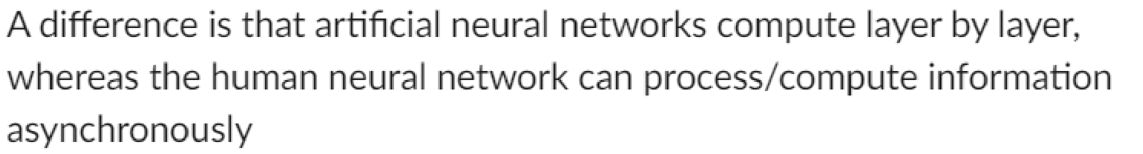
Watch

1. But what is a neural network? | Chapter 1, Deep learning (~20 min) by 3Blue1Brown
2. The Nervous System, Part 1: Crash Course A&P #8 (~10 min) by CrashCourse

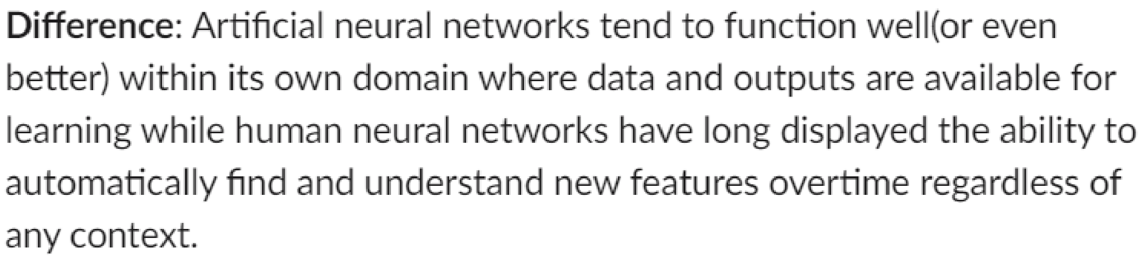
Discuss

1. Reflect on how **artificial** neural networks are different from **human** neural networks.
2. Identify **one** point (no need to write several).
3. Post a 1–2 sentence answer to the topic in your tutorial group: #tg-xx

Pre-Lecture Activity for W09

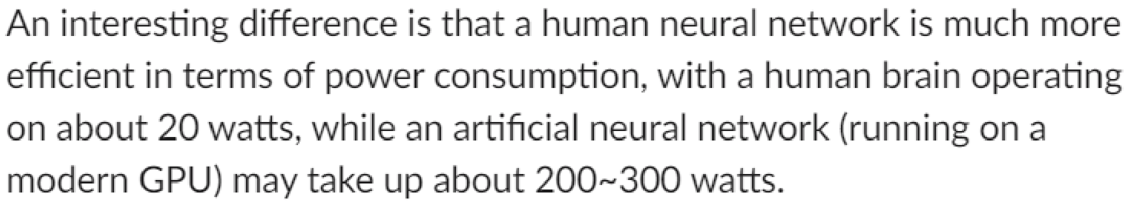


A difference is that artificial neural networks compute layer by layer, whereas the human neural network can process/compute information asynchronously

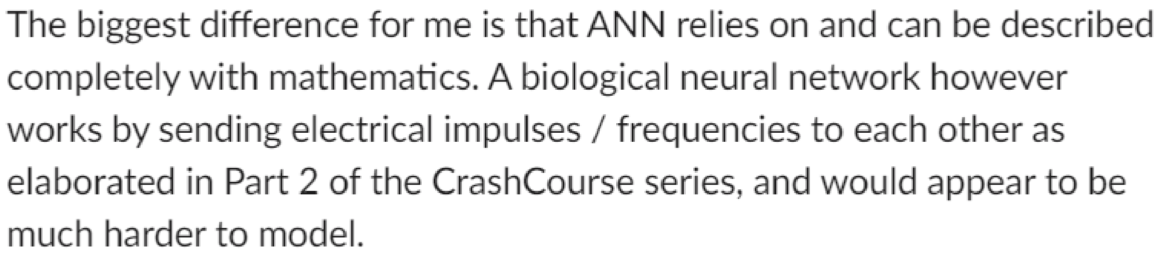


Difference: Artificial neural networks tend to function well(or even better) within its own domain where data and outputs are available for learning while human neural networks have long displayed the ability to automatically find and understand new features overtime regardless of any context.

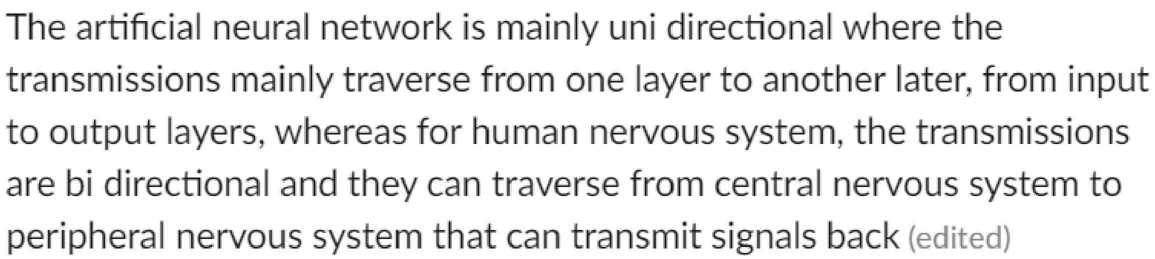
human neural networks are non deterministic



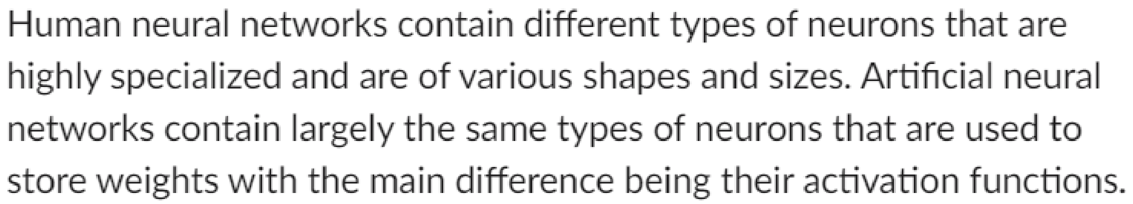
An interesting difference is that a human neural network is much more efficient in terms of power consumption, with a human brain operating on about 20 watts, while an artificial neural network (running on a modern GPU) may take up about 200~300 watts.



The biggest difference for me is that ANN relies on and can be described completely with mathematics. A biological neural network however works by sending electrical impulses / frequencies to each other as elaborated in Part 2 of the CrashCourse series, and would appear to be much harder to model.



The artificial neural network is mainly uni directional where the transmissions mainly traverse from one layer to another later, from input to output layers, whereas for human nervous system, the transmissions are bi directional and they can traverse from central nervous system to peripheral nervous system that can transmit signals back (edited)

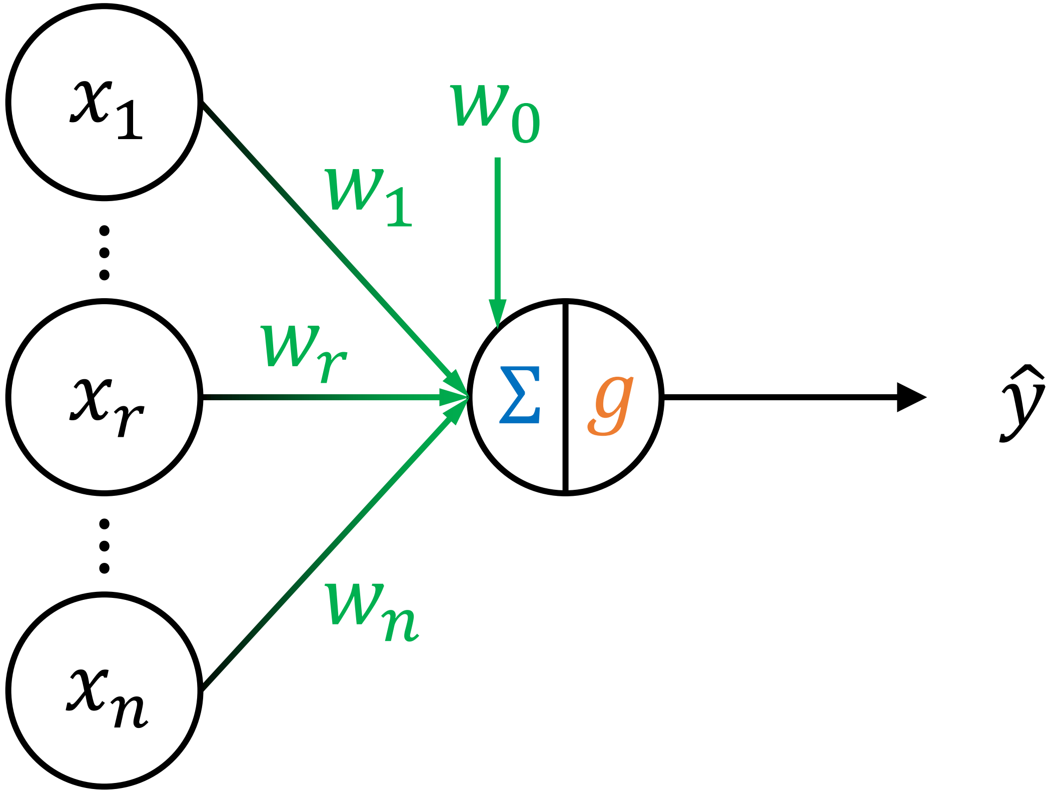


Human neural networks contain different types of neurons that are highly specialized and are of various shapes and sizes. Artificial neural networks contain largely the same types of neurons that are used to store weights with the main difference being their activation functions.

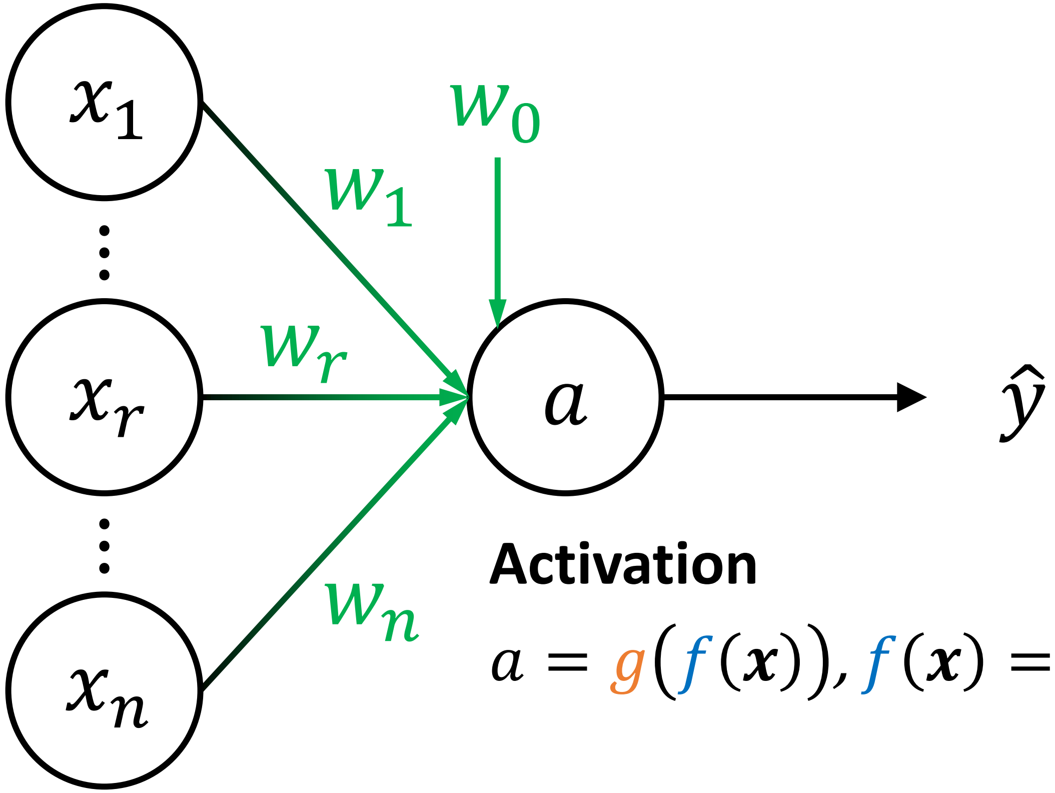
Neural Networks

- What are Neural Networks (Multi-Layer Perceptron)?
- How to train?

Single-Layer Perceptron



Single-Layer Perceptron

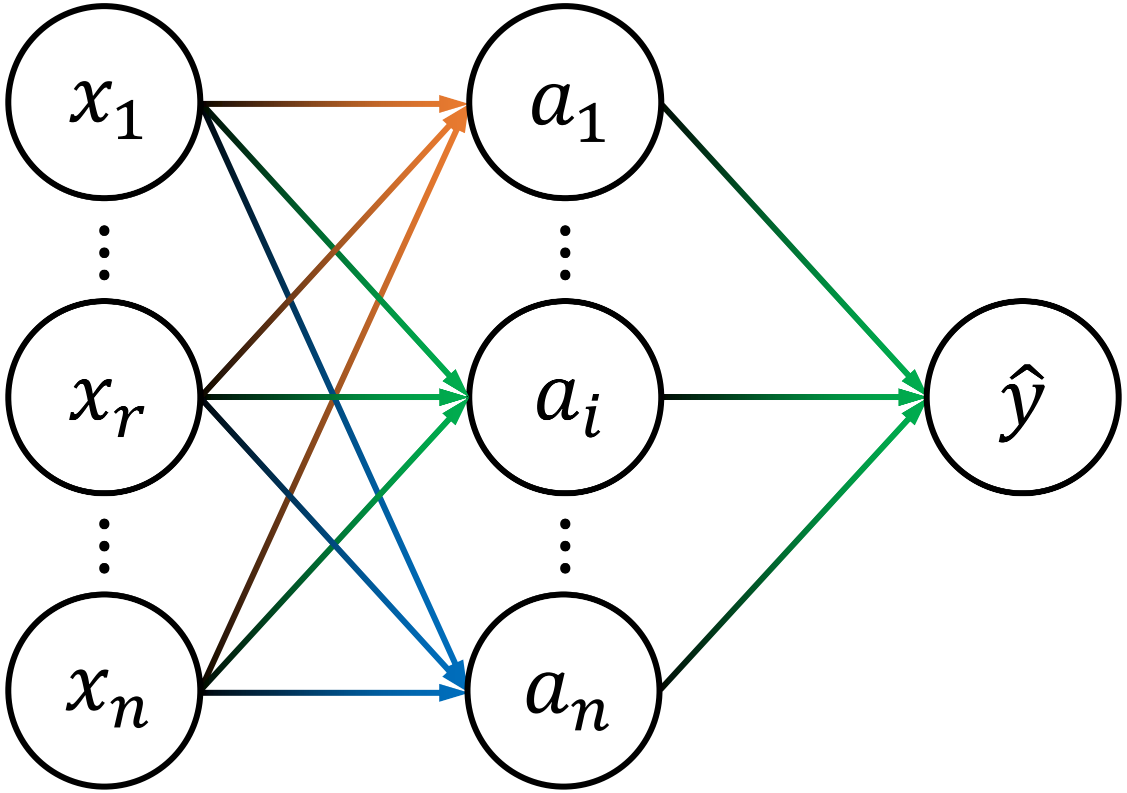


Activation

$a = g(f(\boldsymbol{x})), f(\boldsymbol{x}) = \boldsymbol{w} \cdot \boldsymbol{x}$

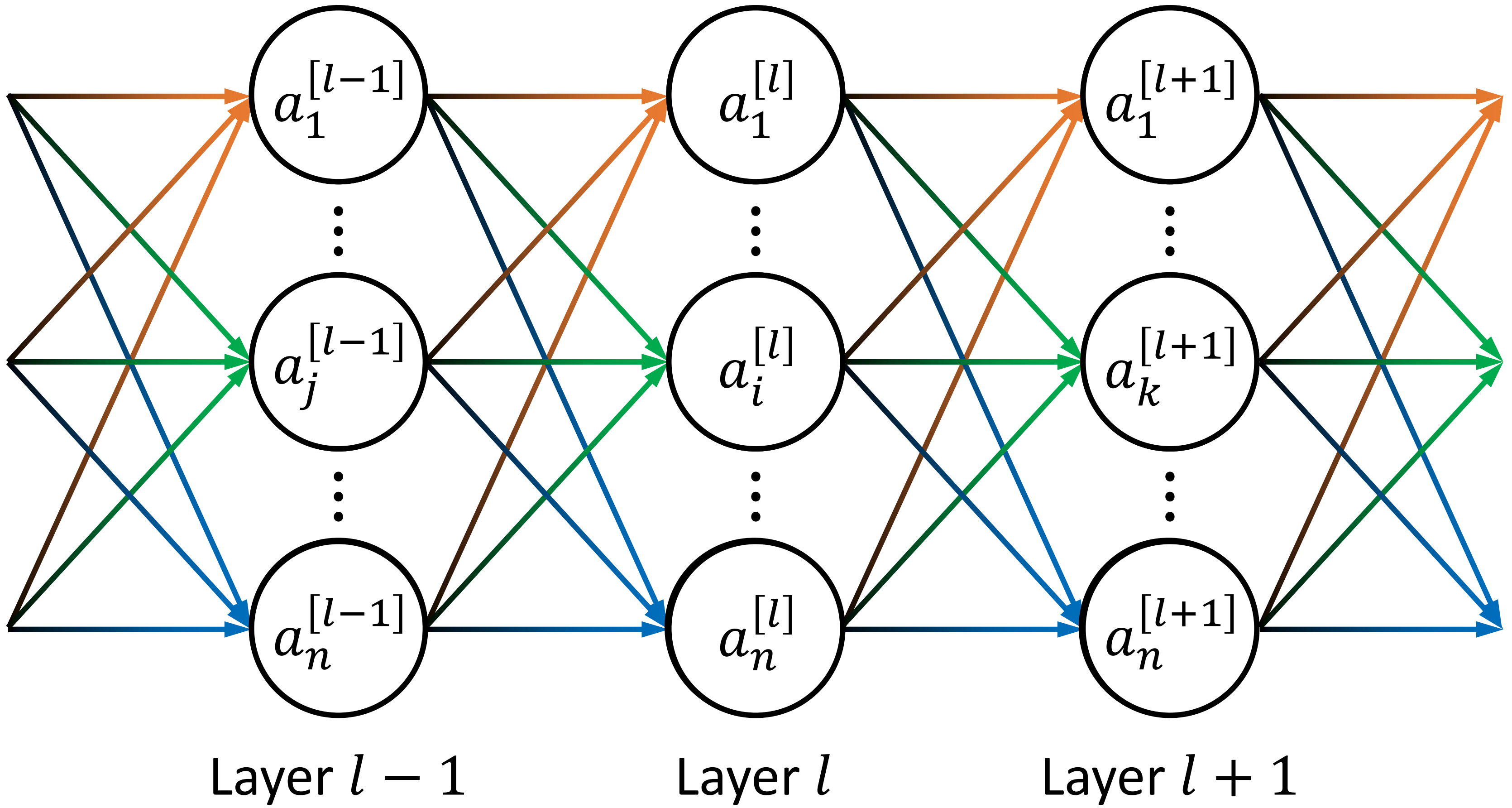
Multi-Layer Perceptron (Neural Network)

2-Layer
MLP

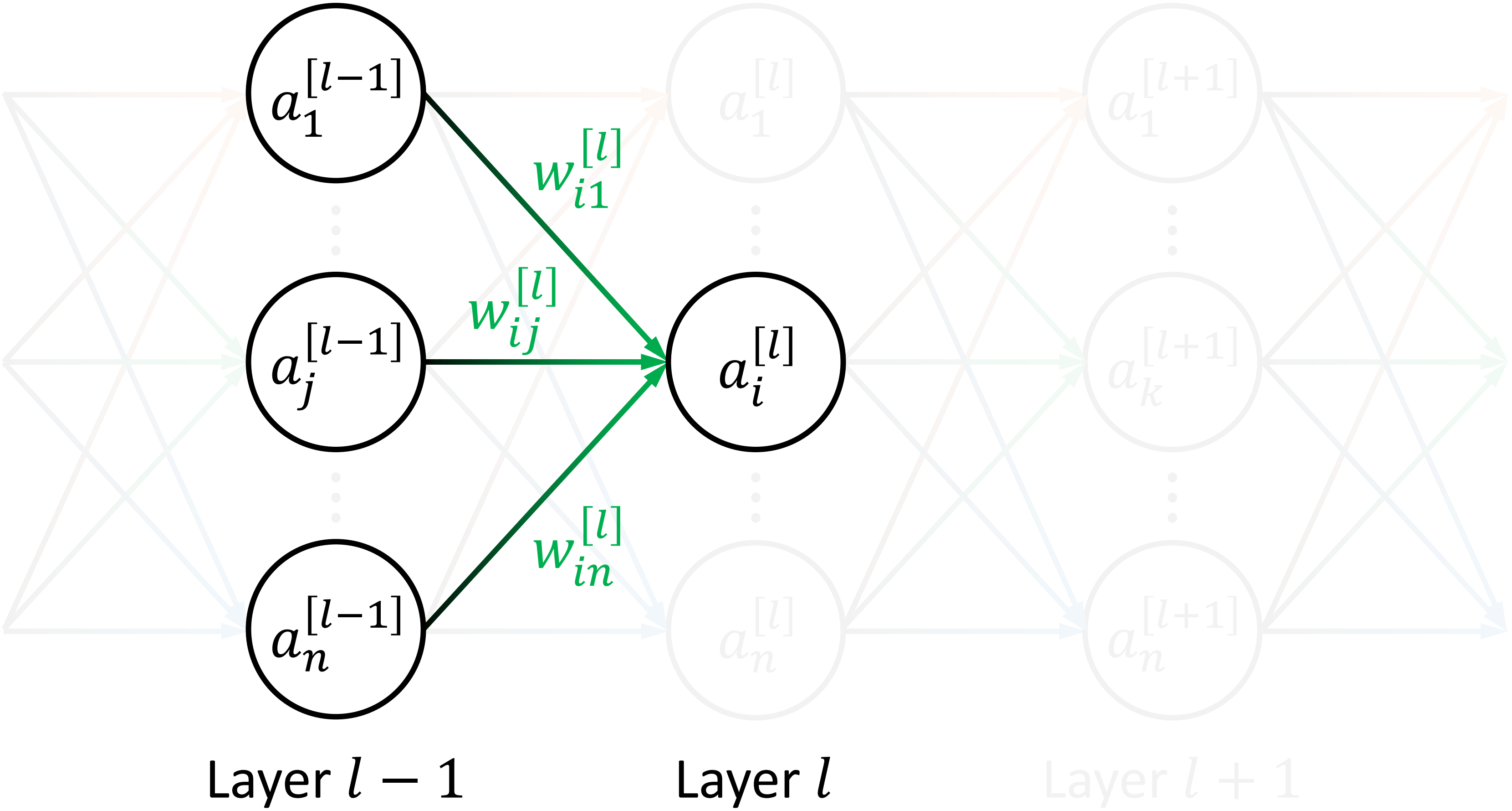


Input Layer Hidden Layer Output Layer

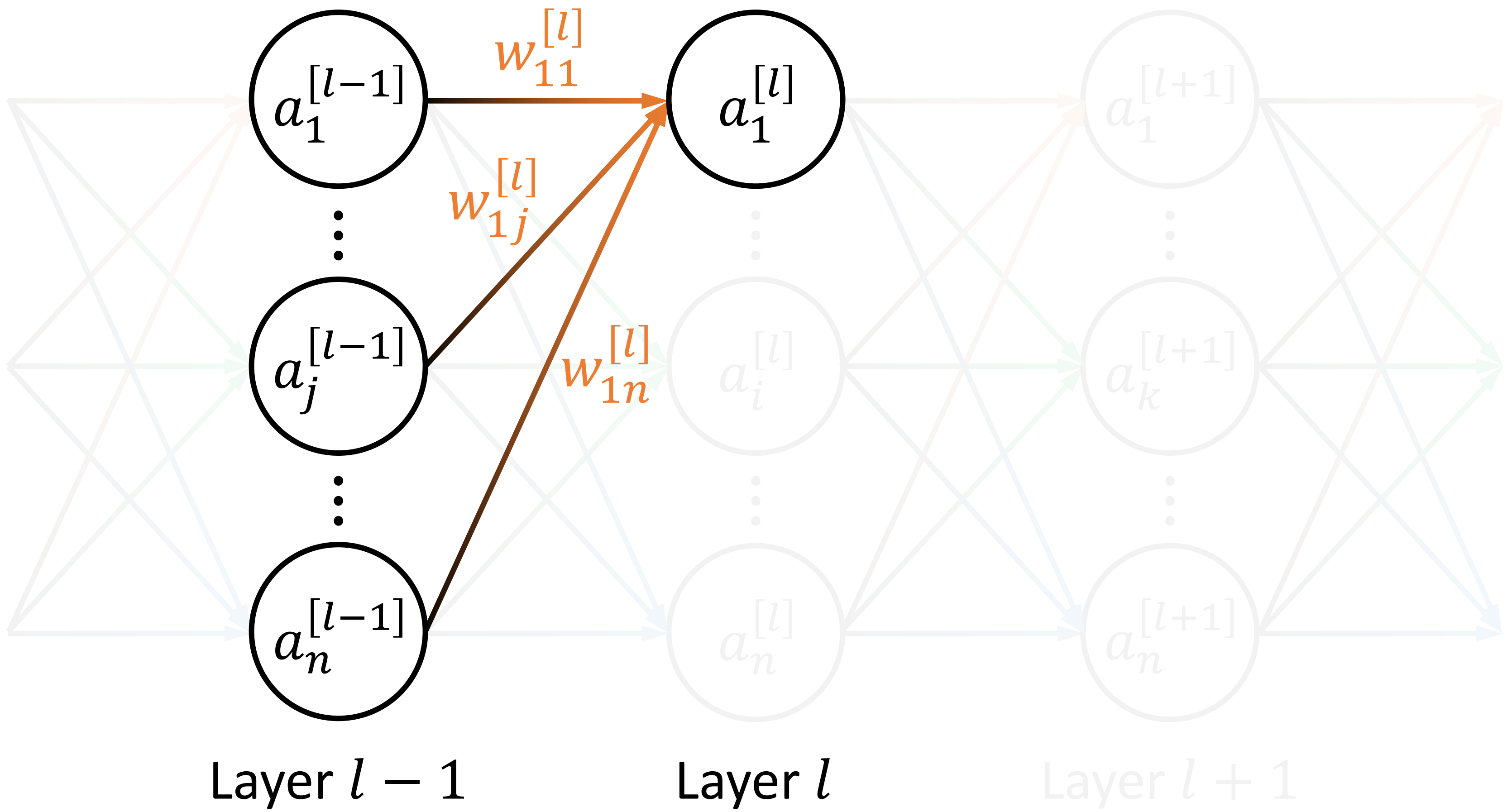
Multi-Layer Perceptron (Neural Network)



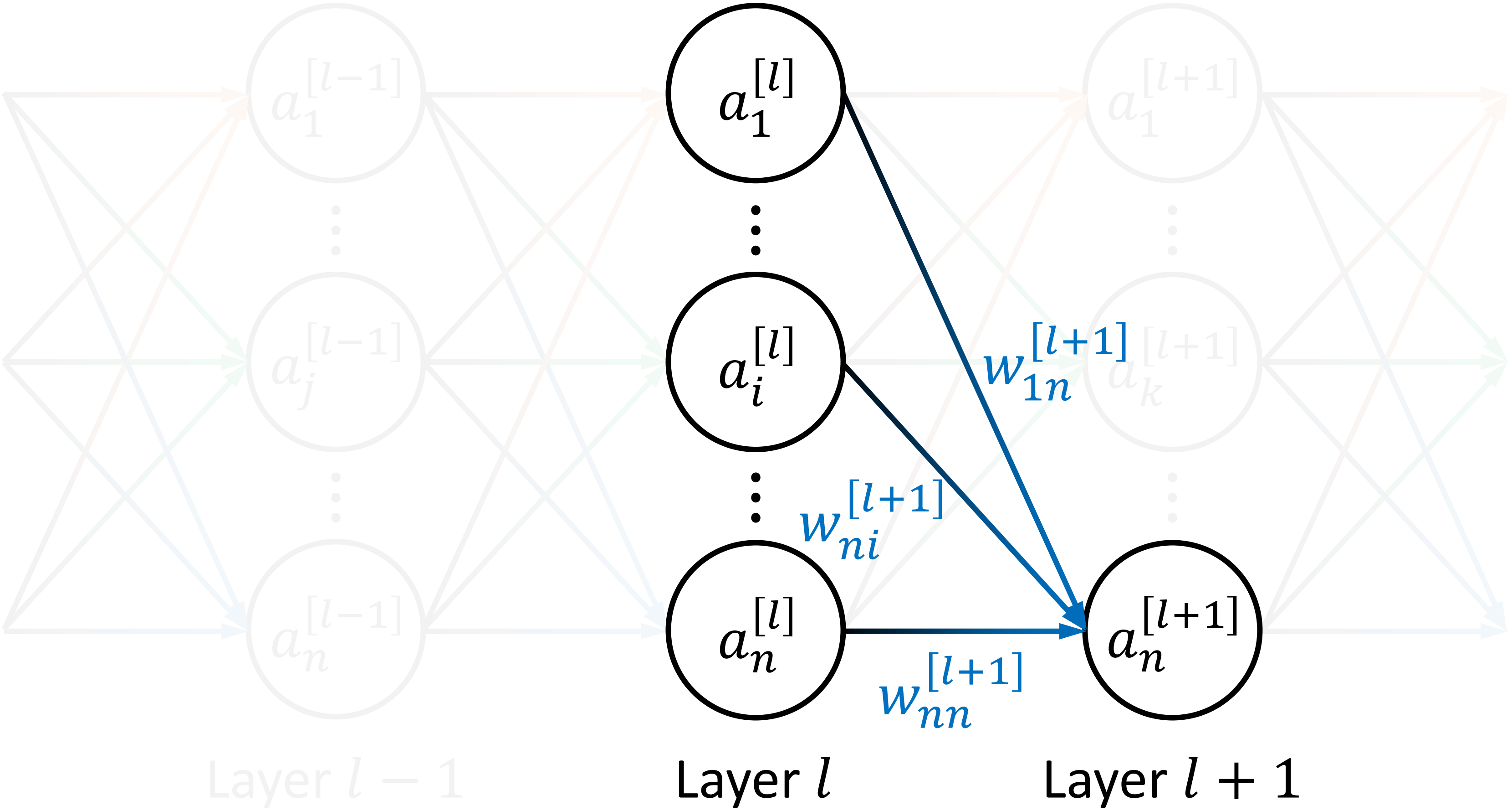
Multi-Layer Perceptron (Neural Network)



Multi-Layer Perceptron (Neural Network)

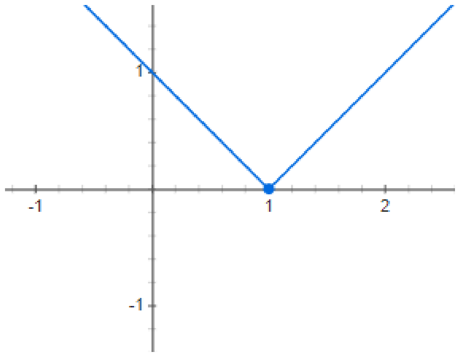


Multi-Layer Perceptron (Neural Network)



E09a.2

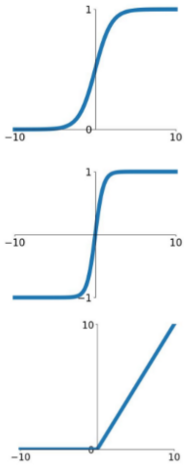
Fitting **non-linear function** with MLP
What model weights can model $\hat{y} = |x - 1|$?

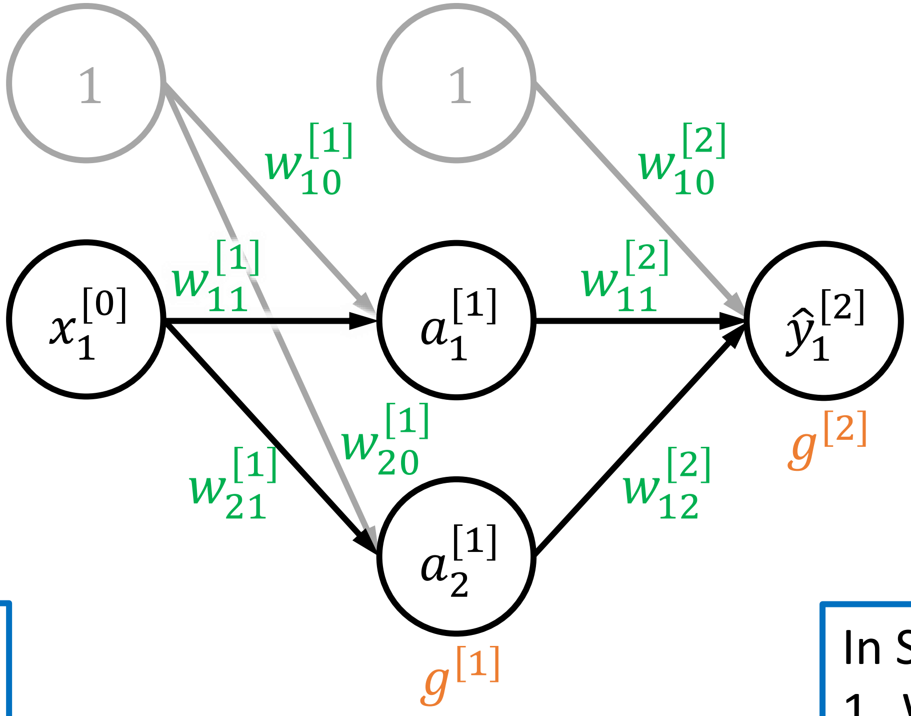


Sigmoid
$\sigma(x) = \frac{1}{1+e^{-x}}$

tanh
$\tanh(x)$

ReLU
$\max(0, x)$



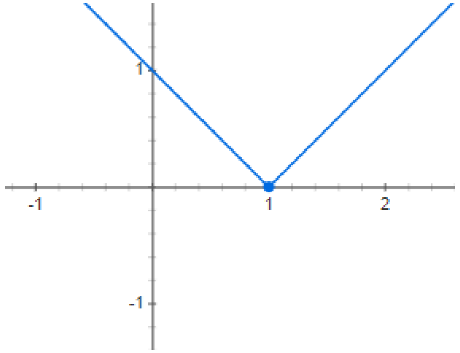


Bonus Question:
What activation function(s) g should you use for each layer?

In Slack #general
1. Write to thread to suggest feature
2. Emote (👍 :+1:) to vote for feature

E09a.2 Solution

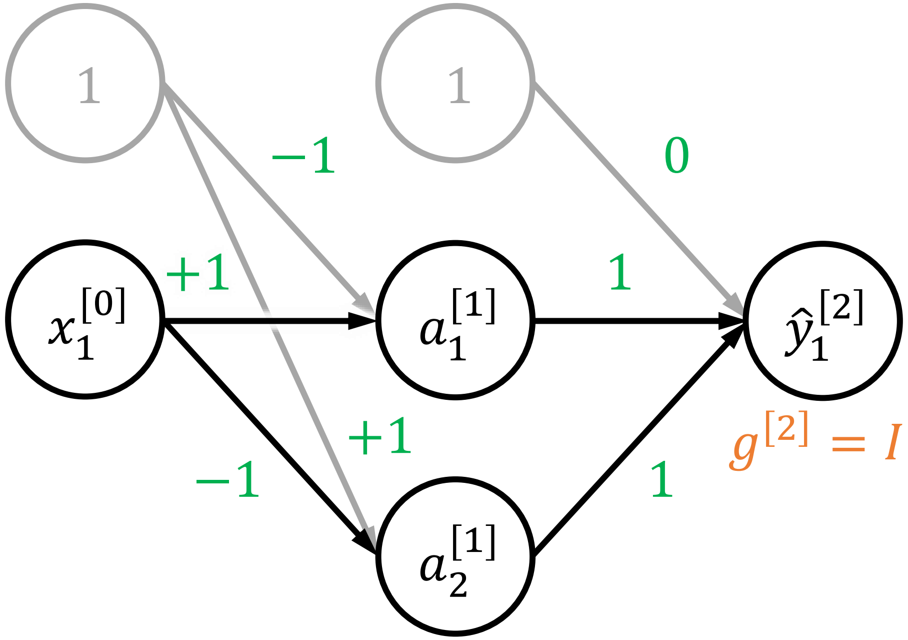
Fitting **non-linear function** with MLP
What model weights can model $\hat{y} = |x - 1|$?



$$\boldsymbol{W}^{[1]} = \begin{pmatrix} w_{10}^{[1]} & w_{20}^{[1]} \\ w_{11}^{[1]} & w_{21}^{[1]} \end{pmatrix} = \begin{pmatrix} -1 & 1 \\ 1 & -1 \end{pmatrix}$$

$$\boldsymbol{x}^{[0]} = \begin{pmatrix} 1 \\ x_1^{[0]} \end{pmatrix}$$

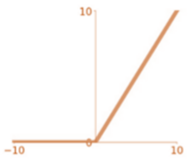
$$\boldsymbol{a}^{[1]} = g^{[1]}\left(\left(\boldsymbol{W}^{[1]}\right)^{\top} \boldsymbol{x}^{[0]}\right) = \begin{pmatrix} \mathrm{ReLU}\left(x_1^{[0]} - 1\right) \\ \mathrm{ReLU}\left(1 - x_1^{[0]}\right) \end{pmatrix}$$



$g^{[2]} = I$

$g^{[1]} = \mathrm{ReLU}$

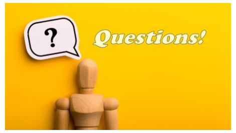
$\max(0, x)$

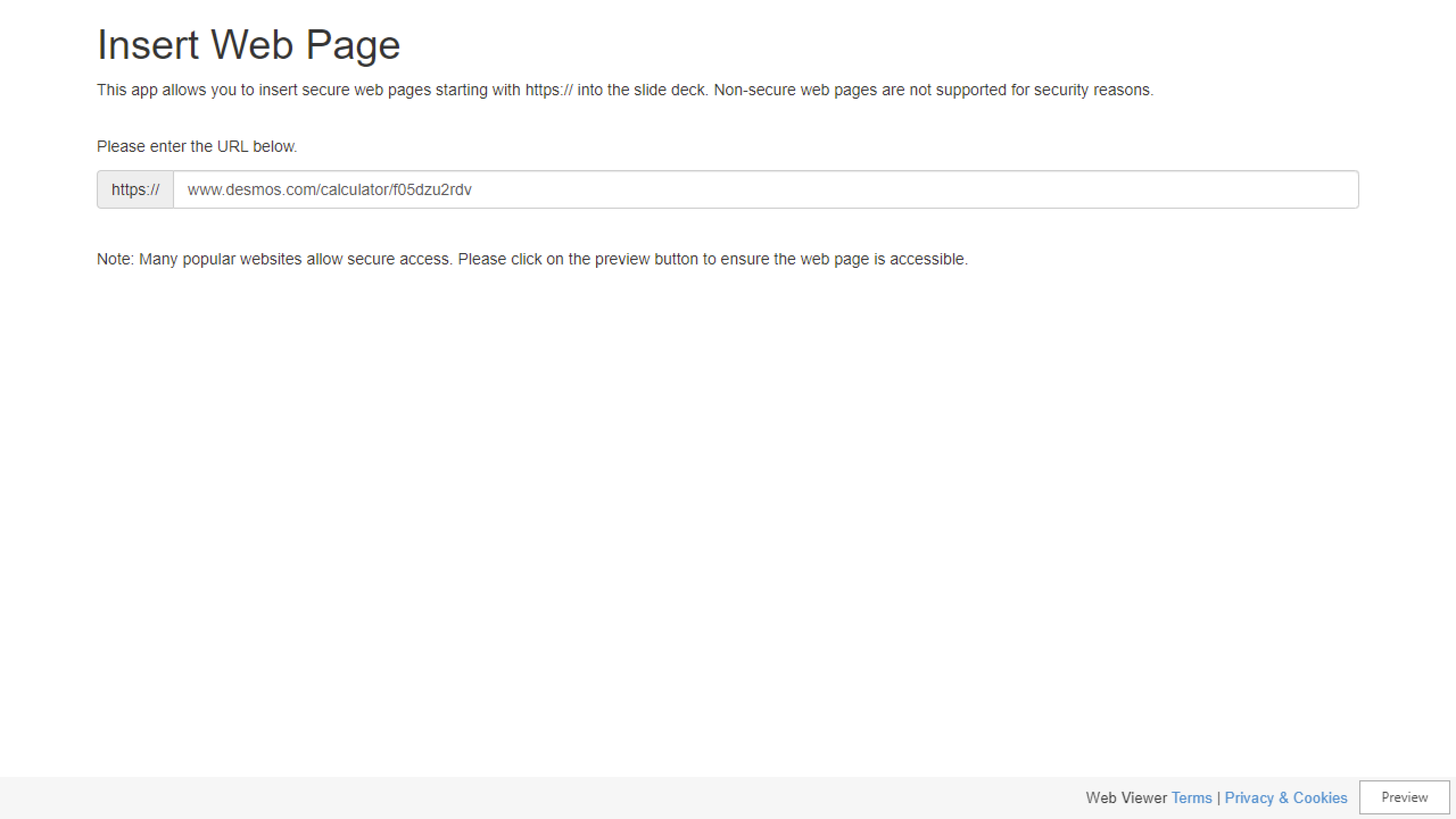


$$\boldsymbol{W}^{[2]} = \begin{pmatrix} w_{10}^{[2]} \\ w_{11}^{[2]} \\ w_{12}^{[2]} \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}$$

$$\boldsymbol{a}^{[1]} = \begin{pmatrix} 1 \\ \mathrm{ReLU}\left(x_1^{[0]} - 1\right) \\ \mathrm{ReLU}\left(1 - x_1^{[0]}\right) \end{pmatrix}$$

$$\begin{aligned} \hat{y}^{[2]} &= g^{[2]}\left(\left(\boldsymbol{W}^{[2]}\right)^{\top} \boldsymbol{a}^{[1]}\right) \\ &= 0 \\ &+ \mathrm{ReLU}\left(x_1^{[0]} - 1\right) \\ &+ \mathrm{ReLU}\left(1 - x_1^{[0]}\right) \end{aligned}$$





Insert Web Page

This app allows you to insert secure web pages starting with https:// into the slide deck. Non-secure web pages are not supported for security reasons.

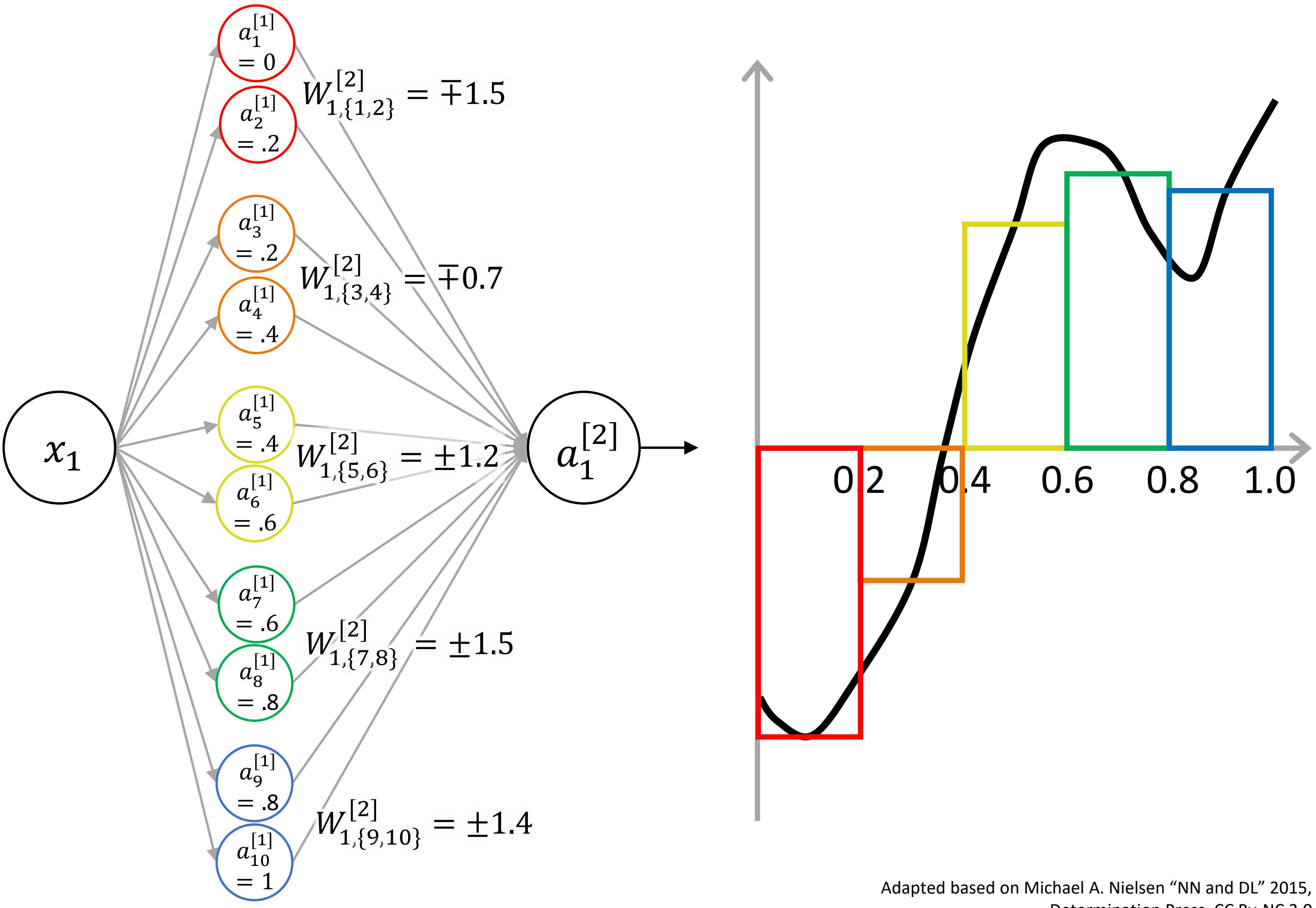
Please enter the URL below.

https:// www.desmos.com/calculator/f05dzu2rdv

Note: Many popular websites allow secure access. Please click on the preview button to ensure the web page is accessible.

Universal Approximation Theorem

- Each neuron contributes a **piecewise function**
- Many piecewise functions can **approximate a curve**



Adapted based on Michael A. Nielsen "NN and DL" 2015, Determination Press, CC By-NC 3.0

Weighted Sum

Summation Series = Scalar

$$\sum_{r=0}^{n} w_r x_r$$

$$w_1 x_1 + \cdots + w_r x_r + \cdots + w_n x_n$$

Transposed Vector Multiplication = Scalar

$$\boldsymbol{w}^\top \boldsymbol{x} = \begin{pmatrix} w_1 & \cdots & w_r & \cdots & w_n \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_r \\ \vdots \\ x_n \end{pmatrix}$$

Vector Dot Product = Scalar

$$\boldsymbol{w} \cdot \boldsymbol{x} = \begin{pmatrix} w_1 \\ \vdots \\ w_r \\ \vdots \\ w_n \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ \vdots \\ x_r \\ \vdots \\ x_n \end{pmatrix}$$

Transposed Matrix Multiplication = Vector

$$\boldsymbol{W}^\top \boldsymbol{x} = \begin{pmatrix} w_{11} & \cdots & w_{1r} & \cdots & w_{1n} \\ \vdots & \ddots & \vdots & ⋰ & \vdots \\ w_{r1} & \cdots & w_{rr} & \cdots & w_{rn} \\ \vdots & ⋰ & \vdots & \ddots & \vdots \\ w_{n1} & \cdots & w_{nr} & \cdots & w_{nn} \end{pmatrix}^\top \begin{pmatrix} x_1 \\ \vdots \\ x_r \\ \vdots \\ x_n \end{pmatrix}$$

Backpropagation
“backprop”

Backpropagation

Backpropagation **efficiently** computes the **gradient** by

- Avoiding duplicate calculations
- Not computing unnecessary intermediate values,
- Computing the **gradient** of ***each* layer**

Specifically, the gradient of the weighted input of each layer is calculated from back $[l+1]$ to front $[l]$):

$$\boxed{\hat{y}'\left(\boldsymbol{W}^{[l]}\right) = \boldsymbol{a}^{[l-1]}\delta^{[l]}} \boxed{\delta^{[l]} = g'^{[l]}\left(f^{[l]}\right)\boldsymbol{W}^{[l+1]}\delta^{[l+1]}}$$

Adapted from: https://en.wikipedia.org/wiki/Backpropagation

E09a.2 Solution

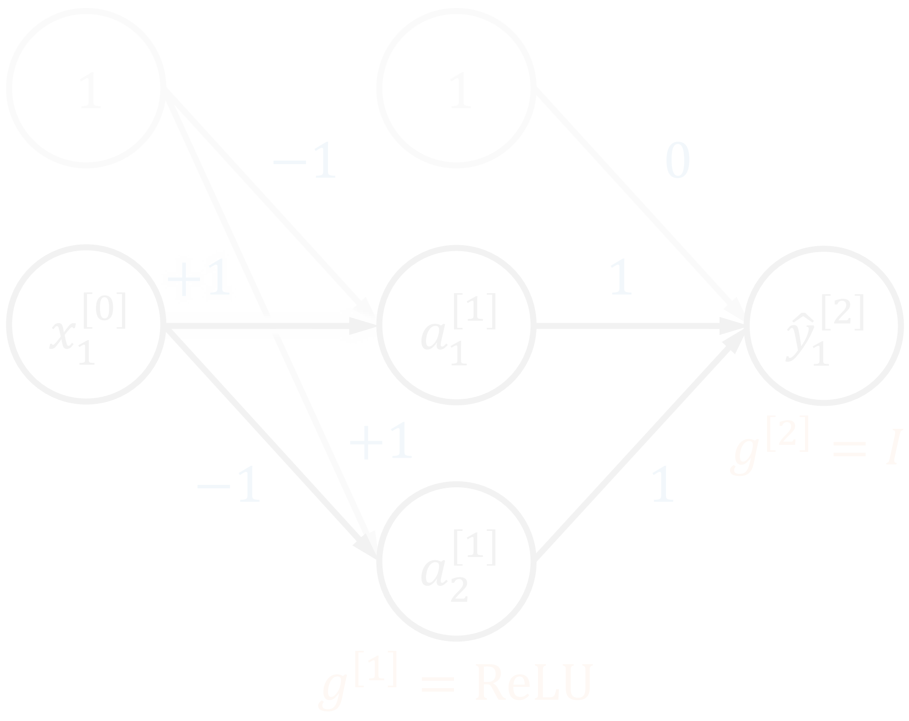
Fitting **non-linear function** with MLP
What model weights can model $\hat{y} = |x - 1|$?

Fitting non-linear function with MLP

What model weights can model $\hat{y} = |x - 1|$?

$$\boldsymbol{W}^{[1]} = \begin{pmatrix} w_{10}^{[1]} & w_{20}^{[1]} \\ w_{11}^{[1]} & w_{20}^{[1]} \end{pmatrix} = \begin{pmatrix} -1 & 1 \\ 1 & -1 \end{pmatrix}$$

$$\boldsymbol{x}^{[0]} = \begin{pmatrix} 1 \\ x_1^{[0]} \end{pmatrix}$$



$$\boldsymbol{W}^{[2]} = \begin{pmatrix} w_{10}^{[2]} \\ w_{11}^{[2]} \\ w_{12}^{[2]} \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}$$

$$\boldsymbol{a}^{[1]} = \begin{pmatrix} 1 \\ \mathrm{ReLU}\left(x_1^{[0]} - 1\right) \\ \mathrm{ReLU}\left(1 - x_1^{[0]}\right) \end{pmatrix}$$

$$\boldsymbol{a}^{[1]} = g^{[1]}\left(\left(\boldsymbol{W}^{[1]}\right)^{\top} \boldsymbol{x}^{[0]}\right) = \begin{pmatrix} \mathrm{ReLU}\left(x_1^{[0]} - 1\right) \\ \mathrm{ReLU}\left(1 - x_1^{[0]}\right) \end{pmatrix}$$

$$\hat{y}^{[2]} = g^{[2]}\left(\left(\boldsymbol{W}^{[2]}\right)^{\top} \boldsymbol{a}^{[1]}\right) = 0 + \mathrm{ReLU}\left(x_1^{[0]} - 1\right) + \mathrm{ReLU}\left(1 - x_1^{[0]}\right)$$

Layer Activation

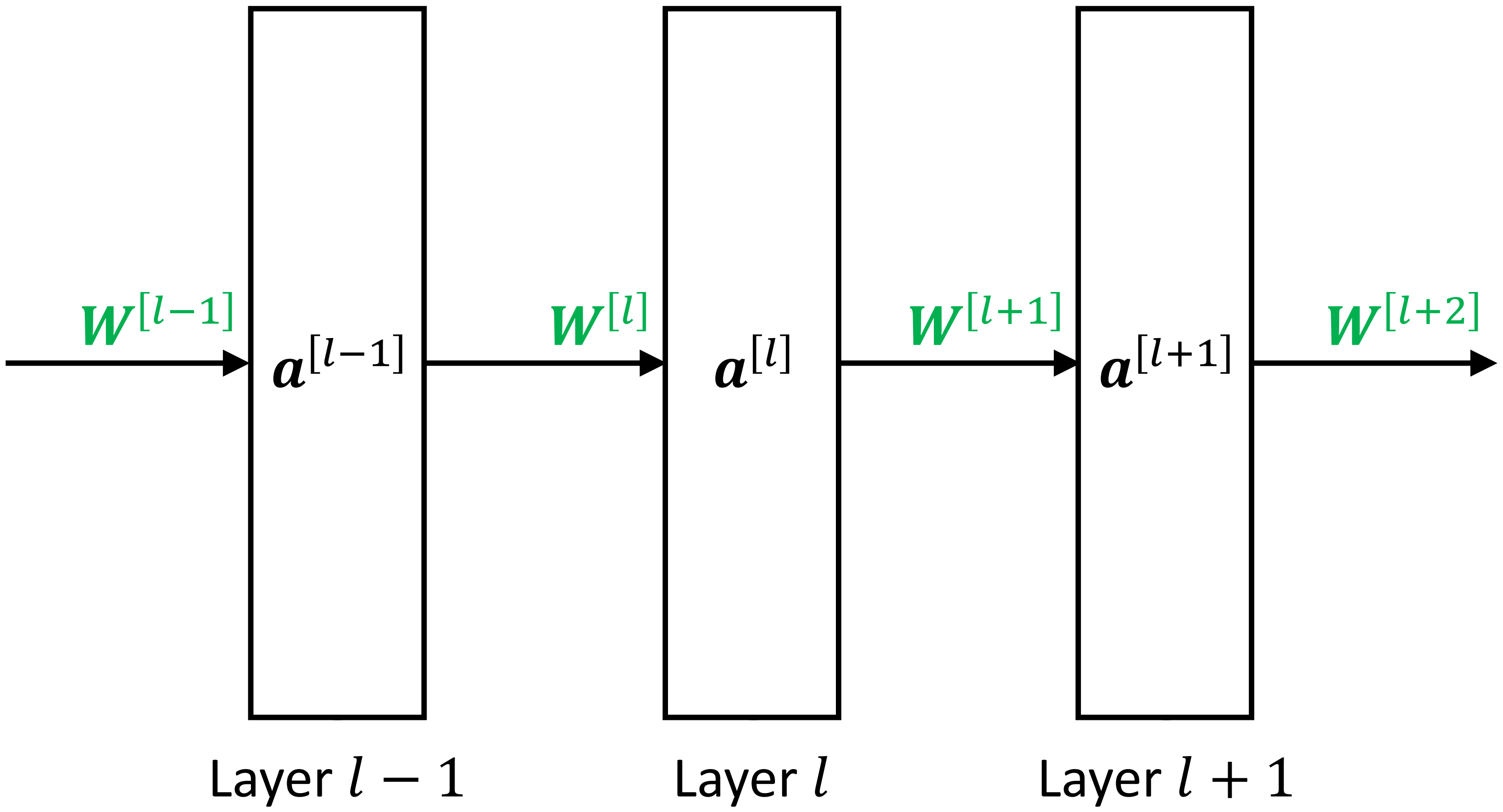
$$a = g(f(x)), f(x) = \boldsymbol{w} \cdot x$$

Single-Layer Perceptron

$$\underset{\text{Layer } l \text{ Activations}}{\boldsymbol{a}^{[l]}} = \overset{\text{Layer } l \text{ Activation Function}}{g^{[l]}}\left(\left(\overset{\text{Layer } l \text{ Weights}}{\boldsymbol{W}^{[l]}}\right)^{\top} \underset{\text{Layer } l-1 \text{ Activations}}{\boldsymbol{a}^{[l-1]}}\right)$$

Layer l in Neural Network

Neural Network

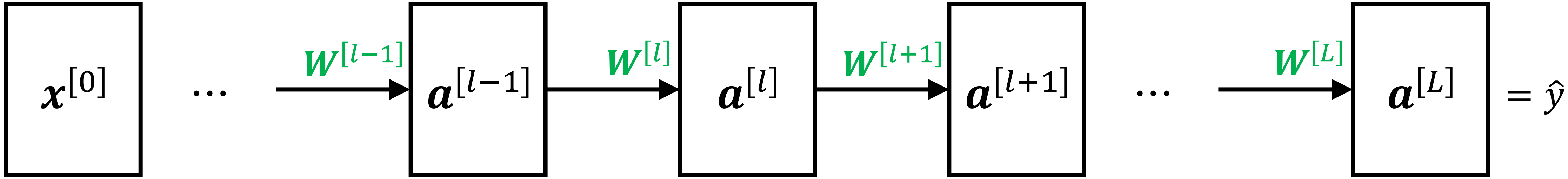


Forward Propagation

$$\boldsymbol{a}^{[l]} \equiv g^{[l]}(f^{[l]}),\ f^{[l]} \equiv \left(W^{[l]}\right)^{\top}\boldsymbol{a}^{[l]}$$

$$g^{[1]}\left(f^{[1]}\left(x^{[0]}\right)\right) = \boldsymbol{a}^{[1]} \qquad g^{[l]}\left(f^{[l]}\left(a^{[l-1]}\right)\right) = \boldsymbol{a}^{[l]} \qquad g^{[L]}\left(f^{[L]}\left(a^{[L-1]}\right)\right) = \boldsymbol{a}^{[L]}$$

$$g^{[1]}\left(\left(\boldsymbol{W}^{[1]}\right)^{\top}x^{[0]}\right) = \boldsymbol{a}^{[1]} \qquad g^{[l]}\left(\left(\boldsymbol{W}^{[l]}\right)^{\top}a^{[l-1]}\right) = \boldsymbol{a}^{[l]} \qquad g^{[L]}\left(\left(\boldsymbol{W}^{[L]}\right)^{\top}a^{[L-1]}\right) = \boldsymbol{a}^{[L]}$$



$$\hat{y}(\boldsymbol{x}) = g^{[L]}((\boldsymbol{W}^{[L]})^{\top}g^{[L-1]}(\cdots(g^{[l]}((\boldsymbol{W}^{[l]})^{\top}g^{[l-1]}(\cdots(g^{[1]}((\boldsymbol{W}^{[1]})^{\top}\boldsymbol{x}^{[0]}))))))$$

$$\hat{y}(\boldsymbol{x}) = g^{[L]}(f^{[L]}(g^{[L-1]}(\cdots(g^{[l]}(f^{[l]}(g^{[l-1]}(\cdots(g^{[1]}(f^{[1]}(\boldsymbol{x}^{[0]}))))))))))$$

Gradient Descent Weight Update

$$\underset{\text{New weight}}{\boldsymbol{W}} \leftarrow \overset{\text{Old weight}}{\boldsymbol{W}} - \underset{\text{Learning Rate}}{\eta}\overset{\text{Learning Error}}{\nabla\varepsilon}$$

MSE error

$$\varepsilon = \frac{1}{2}(\hat{y} - y)^2$$

Gradient of error

$$\nabla\varepsilon = \frac{\partial\varepsilon}{\partial\boldsymbol{W}} = \frac{\partial\varepsilon}{\partial\hat{y}}\boxed{\frac{\partial\hat{y}}{\partial\boldsymbol{W}}}$$

Even more Chain Rule:

Gradient of Neural Network

$$\hat{y}(\boldsymbol{x}) = g^{[L]}(f^{[L]}(g^{[L-1]}(\cdots(g^{[l]}(f^{[l]}(g^{[l-1]}(\cdots(g^{[1]}(f^{[1]}(\boldsymbol{x}^{[0]}))))))))))$$

Gradient relative to ***W***

$$\hat{y}'\left(\boldsymbol{W}^{[L-1]}\right) = \frac{dg^{[L]}}{d\boldsymbol{W}^{[L-1]}} = \frac{df^{[L]}}{d\boldsymbol{W}^{[L-1]}}\boxed{\frac{dg^{[L]}}{df^{[L]}}} \quad \delta^{[L-1]}$$

$$\hat{y}'\left(\boldsymbol{W}^{[l+1]}\right) = \frac{dg^{[L]}}{d\boldsymbol{W}^{[l+1]}} = \frac{df^{[l+1]}}{d\boldsymbol{W}^{[l+1]}}\boxed{\frac{dg^{[l+1]}}{df^{[l+1]}}\cdots\frac{df^{[L]}}{dg^{[L-1]}}\frac{dg^{[L]}}{df^{[L]}}} \quad \delta^{[l+1]}$$

$$\hat{y}'\left(\boldsymbol{W}^{[l]}\right) = \frac{dg^{[L]}}{d\boldsymbol{W}^{[l]}} = \frac{df^{[l]}}{d\boldsymbol{W}^{[l]}}\frac{dg^{[l]}}{df^{[l]}}\frac{df^{[l+1]}}{dg^{[l]}}\underbrace{\frac{dg^{[l+1]}}{df^{[l+1]}}\cdots\frac{df^{[L]}}{dg^{[L-1]}}\frac{dg^{[L]}}{df^{[L]}}}$$

Recursive

$$\hat{y}'\left(\boldsymbol{W}^{[l]}\right) = \frac{dg^{[L]}}{d\boldsymbol{W}^{[l]}} = \frac{df^{[l]}}{d\boldsymbol{W}^{[l]}}\frac{dg^{[l]}}{df^{[l]}}\frac{df^{[l+1]}}{dg^{[l]}}\delta^{[l+1]}$$

$$\hat{y}'\left(\boldsymbol{W}^{[1]}\right) = \frac{dg^{[L]}}{d\boldsymbol{W}^{[1]}} = \frac{df^{[1]}}{d\boldsymbol{W}^{[1]}}\frac{dg^{[1]}}{df^{[1]}}\cdots\frac{dg^{[l]}}{df^{[l]}}\frac{df^{[l+1]}}{dg^{[l]}}\frac{dg^{[l+1]}}{df^{[l+1]}}\cdots\frac{df^{[L]}}{dg^{[L-1]}}\frac{dg^{[L]}}{df^{[L]}}$$

Even more Chain Rule:

Gradient of Neural Network

$$\hat{y}(\boldsymbol{x}) = g^{[L]}(f^{[L]}(g^{[L-1]}(\cdots(g^{[l]}(f^{[l]}(g^{[l-1]}(\cdots(g^{[1]}(f^{[1]}(\boldsymbol{x}^{[0]}))))))))))$$

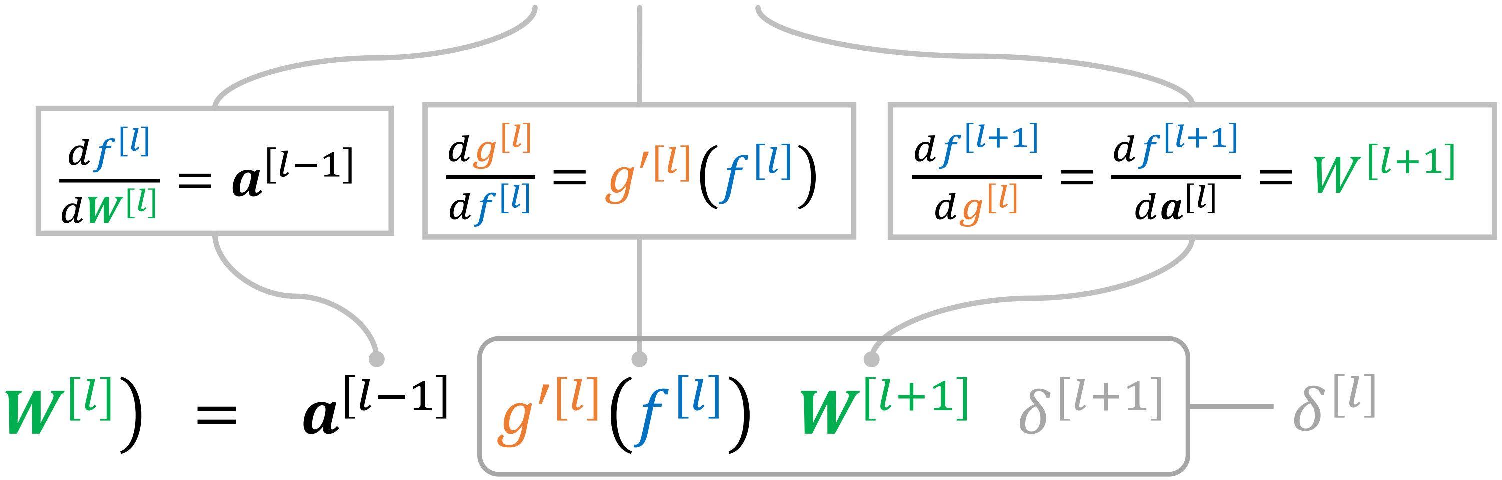
Gradient relative to $\boldsymbol{W}^{[l]}$

Reference

$$\boldsymbol{a}^{[l]} = g^{[l]}(f^{[l]})$$

$$f^{[l]} = \left(W^{[l]}\right)^{\top}\boldsymbol{a}^{[l-1]}$$

$$\hat{y}'(\boldsymbol{W}^{[l]}) = \frac{dg^{[L]}}{d\boldsymbol{W}^{[l]}} = \frac{df^{[l]}}{d\boldsymbol{W}^{[l]}}\frac{dg^{[l]}}{df^{[l]}}\frac{df^{[l+1]}}{dg^{[l]}}\delta^{[l+1]}$$



$$\frac{df^{[l]}}{d\boldsymbol{W}^{[l]}} = \boldsymbol{a}^{[l-1]}$$

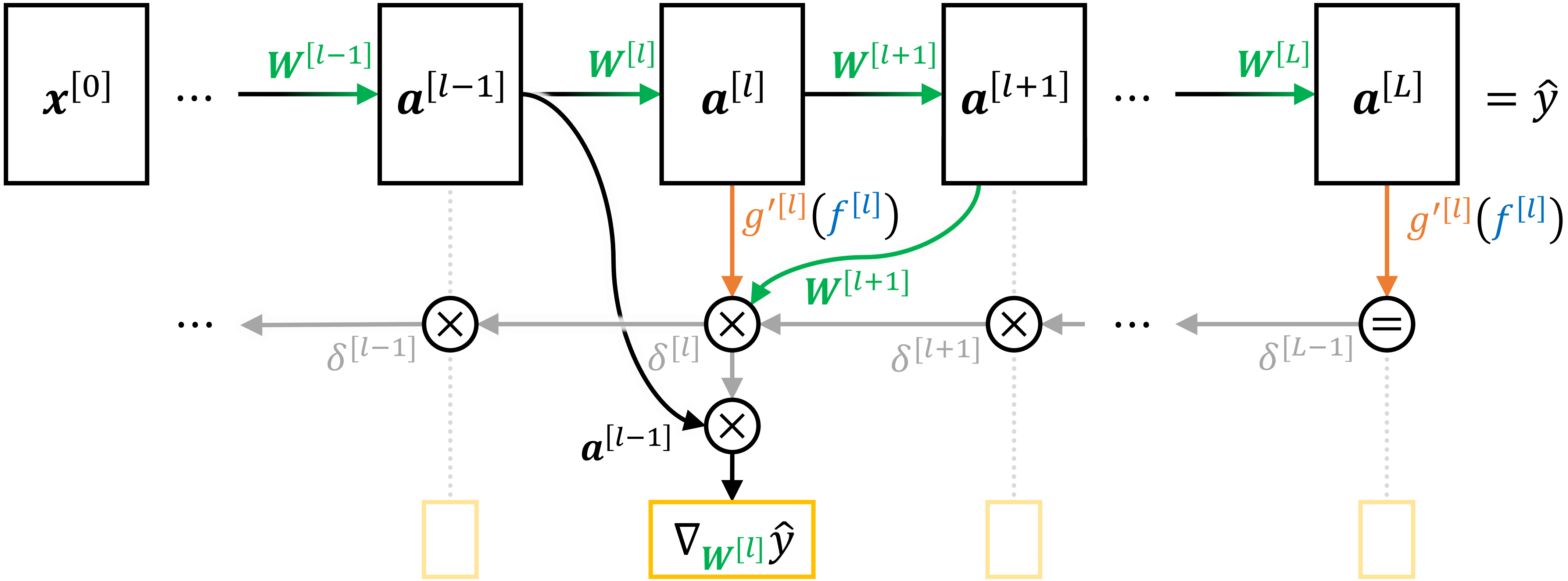
$$\frac{dg^{[l]}}{df^{[l]}} = g'^{[l]}(f^{[l]})$$

$$\frac{df^{[l+1]}}{dg^{[l]}} = \frac{df^{[l+1]}}{d\boldsymbol{a}^{[l]}} = W^{[l+1]}$$

$$\hat{y}'(\boldsymbol{W}^{[l]}) = \boldsymbol{a}^{[l-1]} \boxed{g'^{[l]}(f^{[l]})\ \boldsymbol{W}^{[l+1]}\ \delta^{[l+1]}} \text{ — } \delta^{[l]}$$

$$\boxed{\hat{y}'(\boldsymbol{W}^{[l]}) = \boldsymbol{a}^{[l-1]}\delta^{[l]}} \quad \boxed{\delta^{[l]} = g'^{[l]}(f^{[l]})\boldsymbol{W}^{[l+1]}\delta^{[l+1]}}$$

Backward Propagation

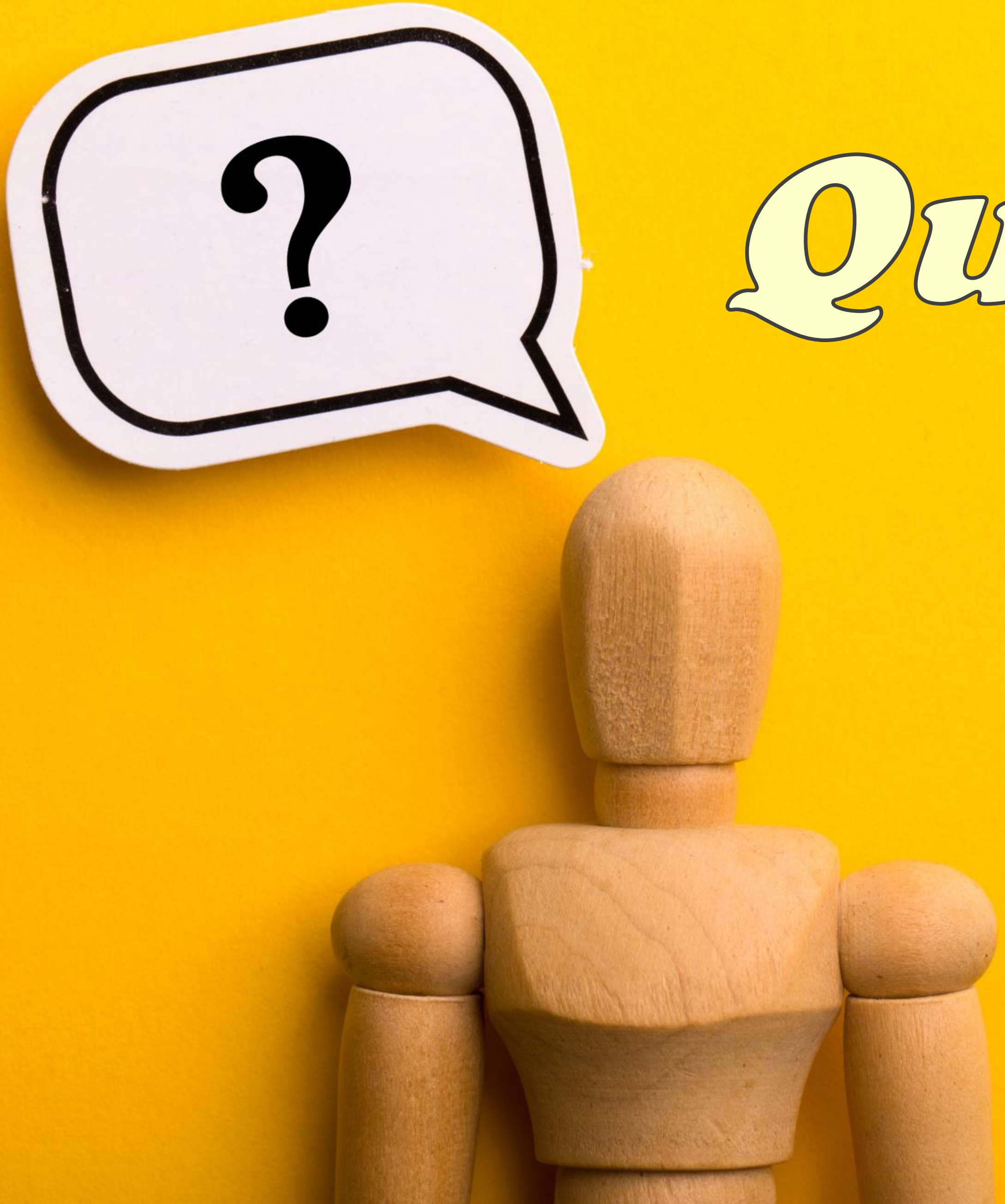


$$\hat{y}'\left(W^{[l]}\right) = a^{[l-1]}\delta^{[l]} \qquad \delta^{[l]} = g'^{[l]}\left(f^{[l]}\right)W^{[l+1]}\delta^{[l+1]}$$

Auto Differentiation for Backprop

- Even with backprop, implementing the gradients is tedious
- Deep learning APIs have automated differentiation.
  - Tensor Flow autodiff
  - PyTorch autograd
  - Implement derivatives of many common functions
  - You just need to implement your layers and neurons; API will handle gradients
- Caution
  - If you want to implement **custom functions** (not simple weighted sum)
  - They need to be **differentiable** to be able to calculate their **gradients**
  - Otherwise, backprop cannot update weights accurately

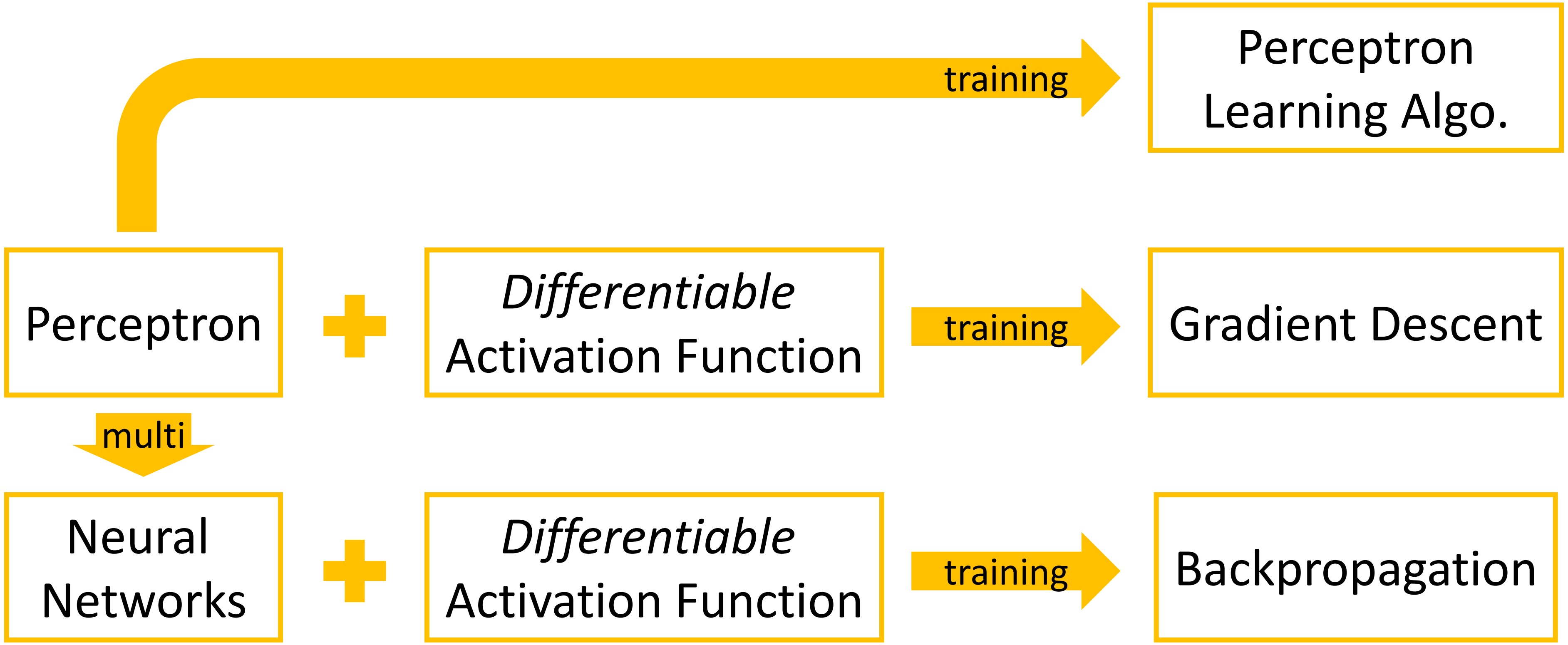
Questions!



Wrapping Up

AI
HISTORY

What did we learn?



Perceptron

Differentiable Activation Function

training

Gradient Descent

training

Perceptron Learning Algo.

multi

Neural Networks

Differentiable Activation Function

training

Backpropagation

Next week:

Deep Learning and Explainable AI (XAI)

Image credit: https://twitter.com/FredericClad/status/827076552261251072/photo/1

W10 Pre-Lecture Task (due before next Mon)

Watch

- Who Invented A.I.? - The Pioneers of Our Future by ColdFusion

Play

- https://distill.pub/2018/building-blocks/
  - Don’t worry about reading the whole article

Discuss

1. Identify what is strange, funny, or erroneous in the deep learning model in Building-Blocks
2. Take a screenshot of the issue and share with your tutorial mates
3. Try to explain why the model was behaving as identified
3. Post a 2–3 sentence description to the topic in your tutorial group: #tg-xx