Perceptron & Neural Networks

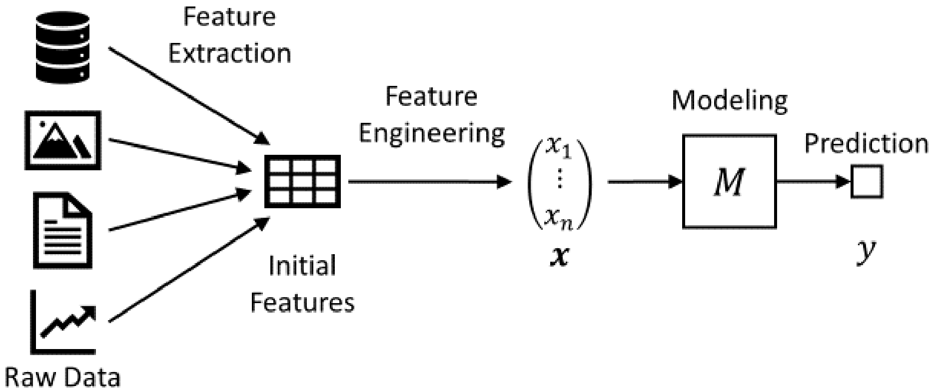
9A

CS 3244 Machine Learning

NUS National University of Singapore | Computing

Recap from W08b

Feature Extraction/Engineering → Modeling



[W08b] Student Learning Outcomes

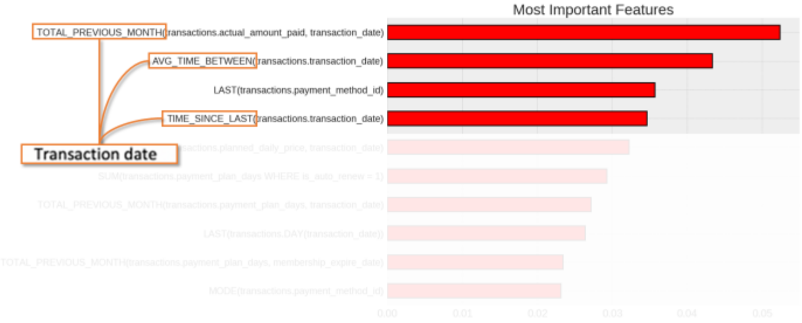
What did we learn?

1. 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

2. Describe **issues** when extracting features for various data types

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)

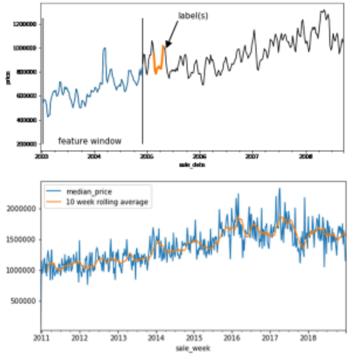


Image credit: https://cloud.google.com/blog/products/ai-machine-learning/how-to-quickly-solve-machine-learning-forecasting-problems-using-pandas-and-bigquery

NUS CS3244: Machine Learning 19

Feature: Edge Detection Kernels (2D)

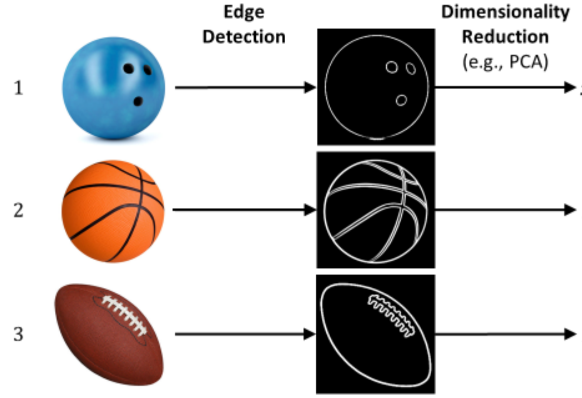
$$\frac{\partial c}{\partial p_x} \approx \frac{c_{(x+1,y)} - c_{(x-1,y)}}{p_{(x+1,y)} - p_{(x-1,y)}} \qquad \frac{\partial c}{\partial p_y} \approx \frac{c_{(x,y+1)} - c_{(x,y-1)}}{p_{(x,y+1)} - p_{(x,y-1)}}$$

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

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

NUS CS3244: Machine Learning

Feature: Shape Feature Vector



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

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. $\text{BOW}(\boldsymbol{w}) \to \boldsymbol{x}$
4. Problem: high dimensions if many words

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

#	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 **Perceptron Learning Algorithm**
3. Understand how to *compose* multiple Perceptrons into a **Neural Network**
4. Describe how Neural Networks are *trained* with **gradient descent** and **backpropagation** [W09b]

Week 09A: Lecture Outline

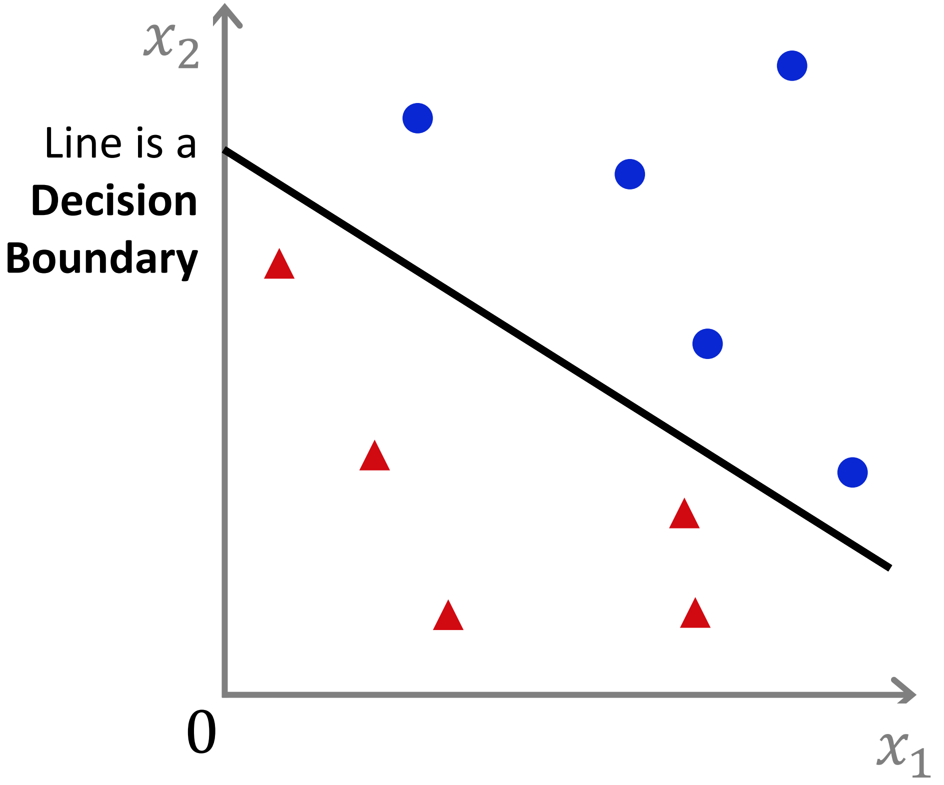
1. Perceptron
2. Perceptron Learning Algorithm (PLA)
3. Activation Functions
4. Gradient Descent
5. Neural Networks
6. Backpropagation [W09B]


AI
HISTORY

Perceptron

Linear Classifiers

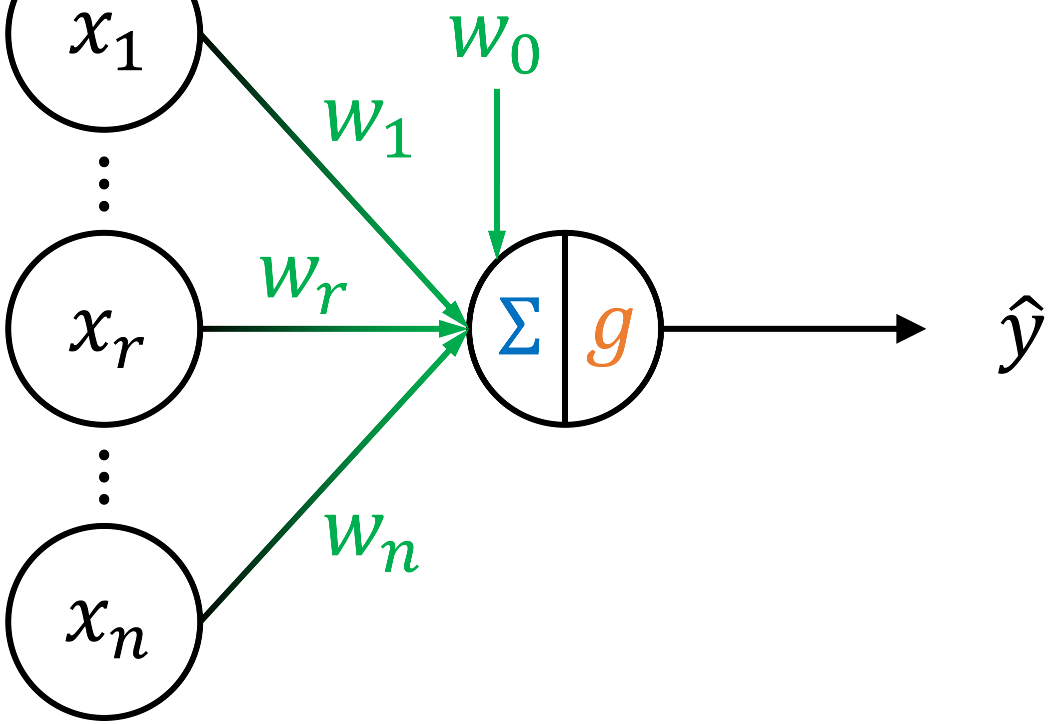
- Logistic Regression [W04A]
- Linear SVM [W04B]
- Perceptron



Perceptron

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

Perceptron



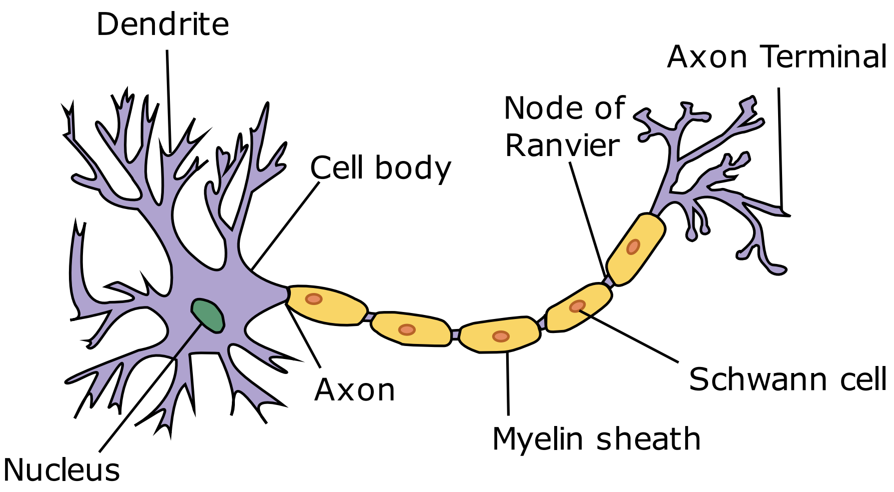


Diagram credits: Dhp1080 - Own work, CC BY-SA 3.0 via Wikimedia Commons.

Line Equation

$$x_2 = mx_1 + c$$

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

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

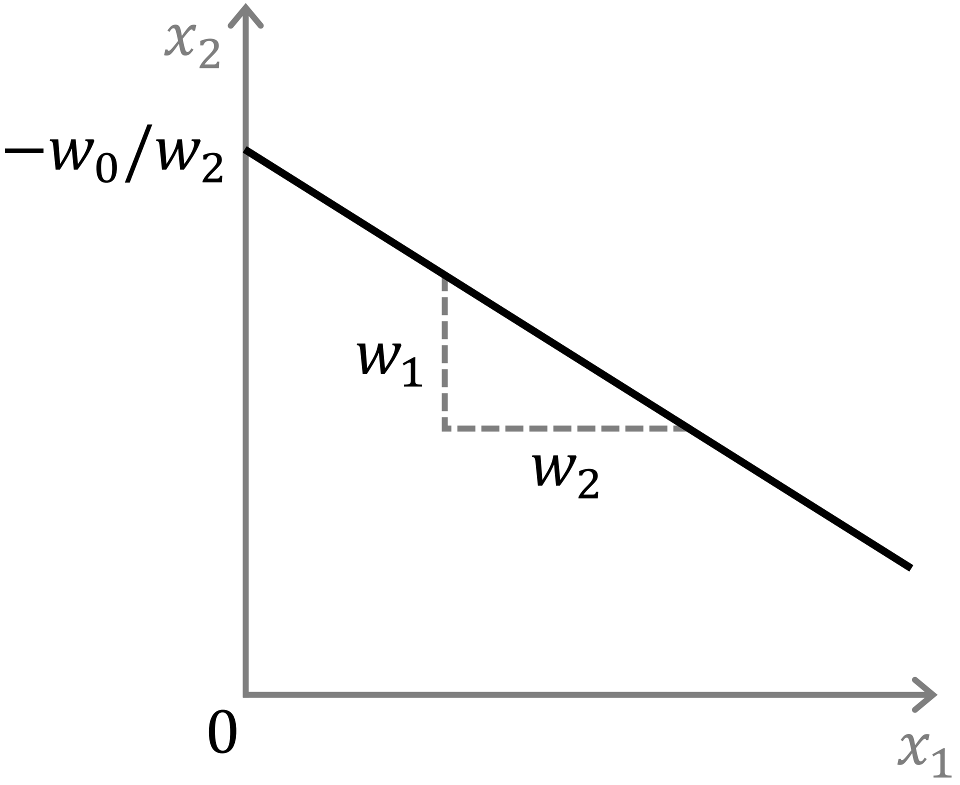


Image credit: https://en.wikipedia.org/wiki/Sign_function

Linear Classification

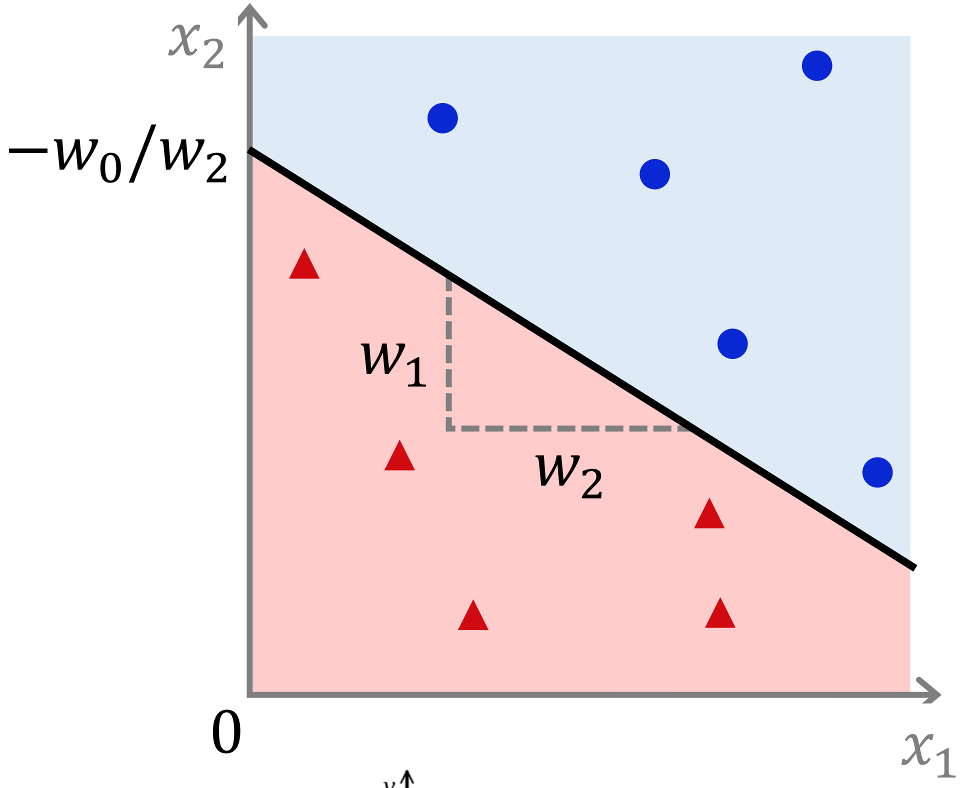
$$x_2 = mx_1 + c$$

$$w_2x_2 + w_1x_1 + w_0 = 0$$

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

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

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



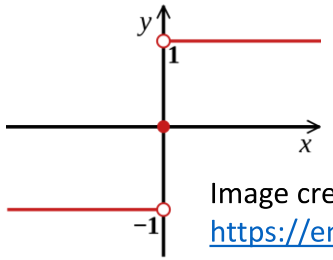
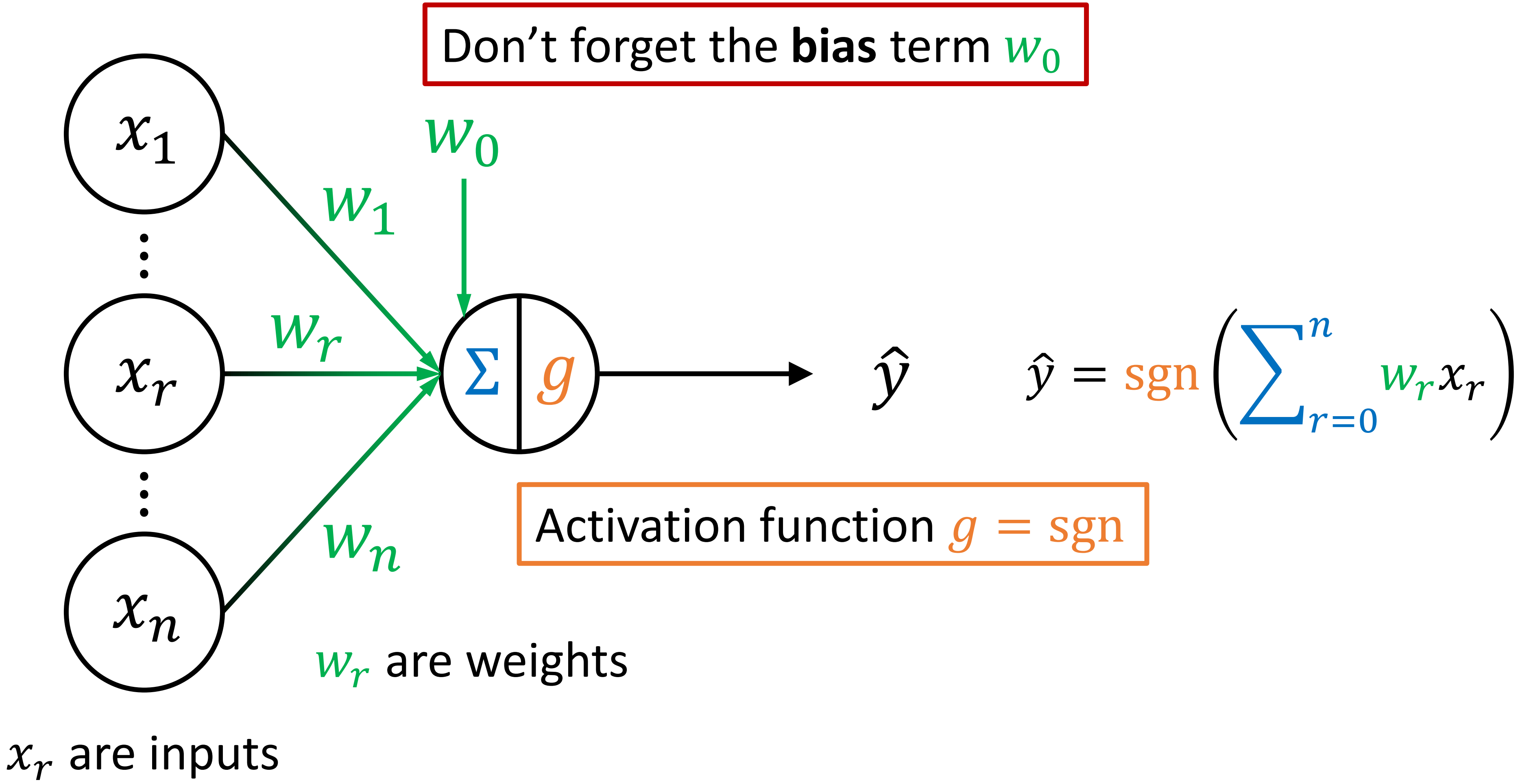
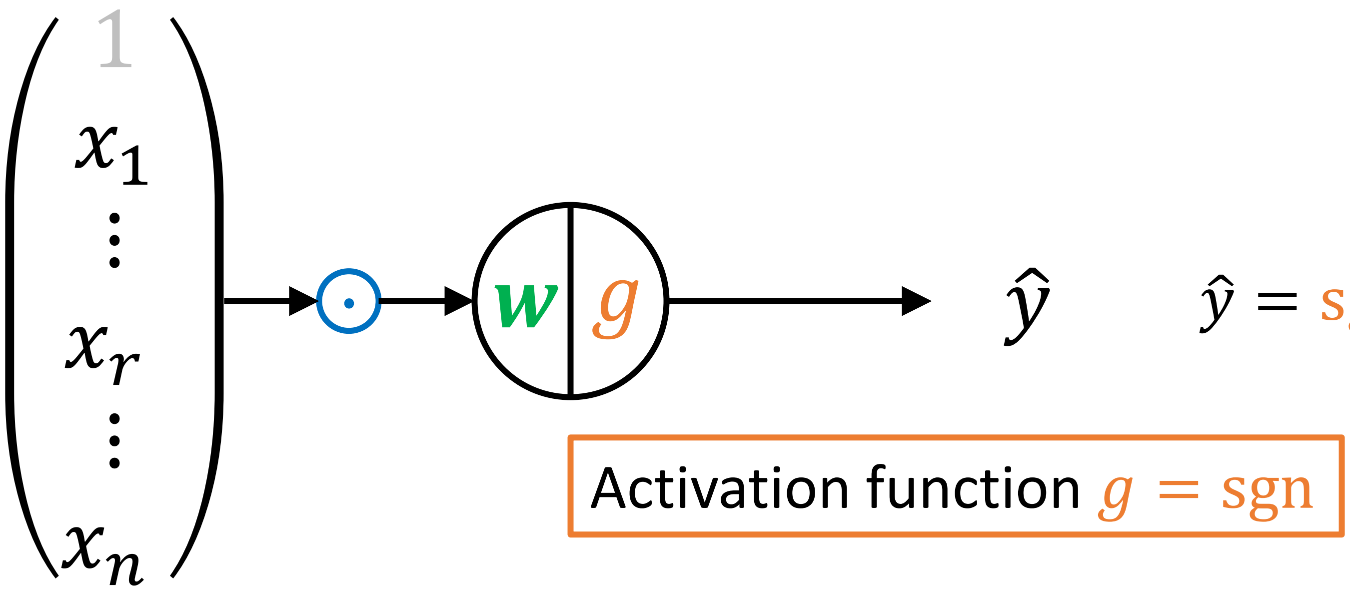


Image credit: https://en.wikipedia.org/wiki/Sign_function

Perceptron


Don't forget the **bias** term w_0
w_0
x_1
w_1
x_r
w_r
x_n
w_n
Σ
g
$\hat{y}$
$\hat{y} = \mathrm{sgn}\left(\sum_{r=0}^{n} w_r x_r\right)$
Activation function $g = \mathrm{sgn}$
w_r are weights
x_r are inputs

Perceptron Classification



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

Perceptron Learning Algorithm (PLA)

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

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

Old weight

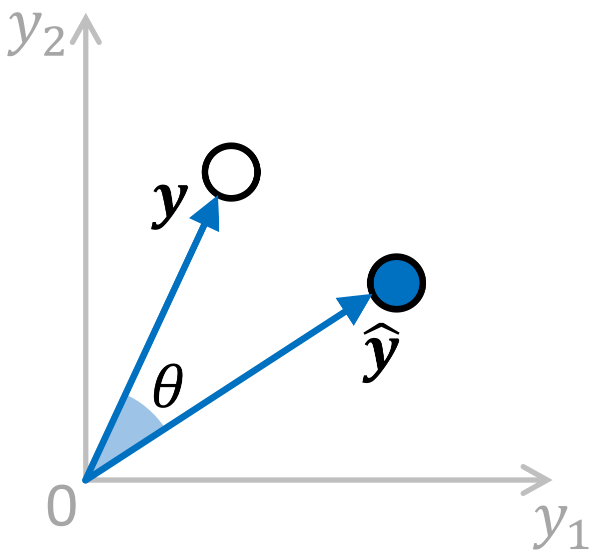
Learning Error

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta(y - \hat{y})\boldsymbol{x}$$

New weight

Learning Rate

Vector Distances and Similarity

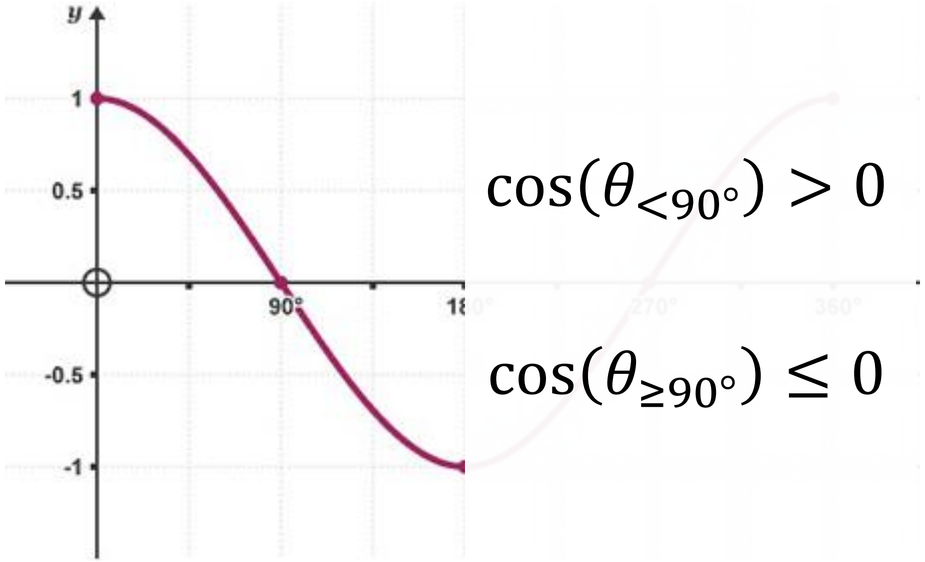


Cosine Similarity

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

Vector Distances and Similarity

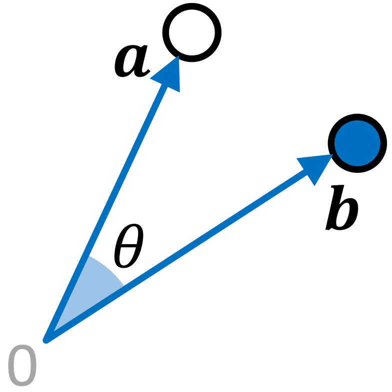
Cosine Curve



$$\cos(\theta_{<90°}) > 0$$

$$\cos(\theta_{\geq 90°}) \leq 0$$

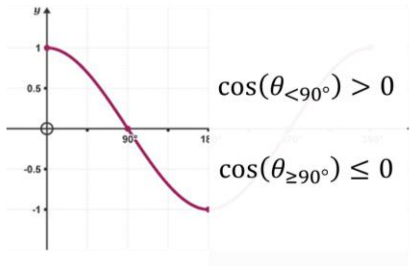
Image credit:
https://www.open.edu/openlearn/ocw/pluginfile.php/947914/mod_oucontent/oucontent/48949/9eaffc43/9f8315d5/mfs_w4_fig4.jpg

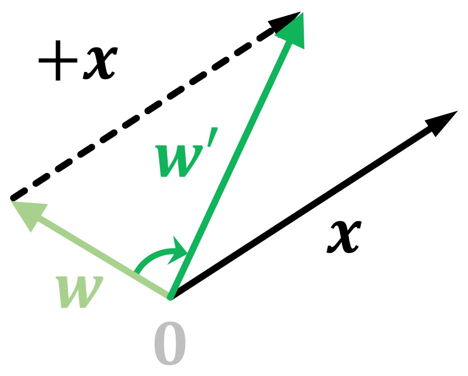


Cosine Similarity

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

Perceptron Weight Update





Consider this misclassification:

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

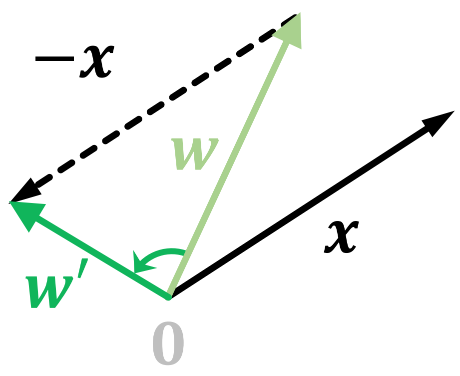
- $\hat{y} = -1 \Rightarrow \boldsymbol{w} \cdot \boldsymbol{x} \leq 0 \Rightarrow \theta = \cos^{-1}\left(\frac{\boldsymbol{w}}{|\boldsymbol{w}|} \cdot \frac{\boldsymbol{x}}{|\boldsymbol{x}|}\right) \geq 90°$

But we want

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

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

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



Consider this misclassification:

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

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

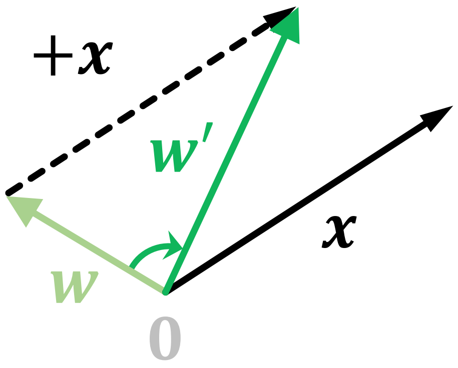
But we want

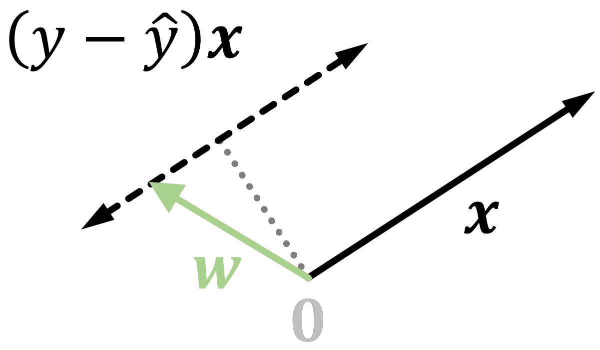
- $\hat{y} = -1 \Rightarrow \boldsymbol{w} \cdot \boldsymbol{x} \leq 0 \Rightarrow \theta > 90°$
- i.e., $\boldsymbol{w}$ to point in a <u>less similar direction</u> as $\boldsymbol{x}$

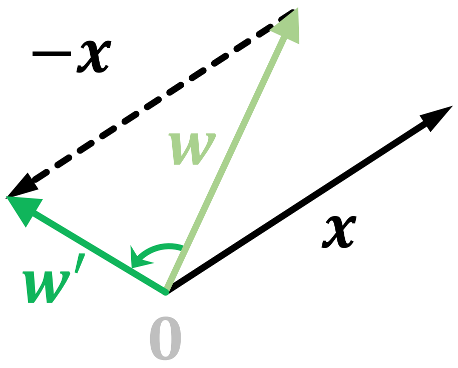
<u>Negating $\boldsymbol{x}$ from $\boldsymbol{w}$</u> will make a less positive result, i.e.,

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

Perceptron Weight Update







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

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

Old weight

Learning Error

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta(y - \hat{y})\boldsymbol{x}$$

New weight

Learning Rate

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

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

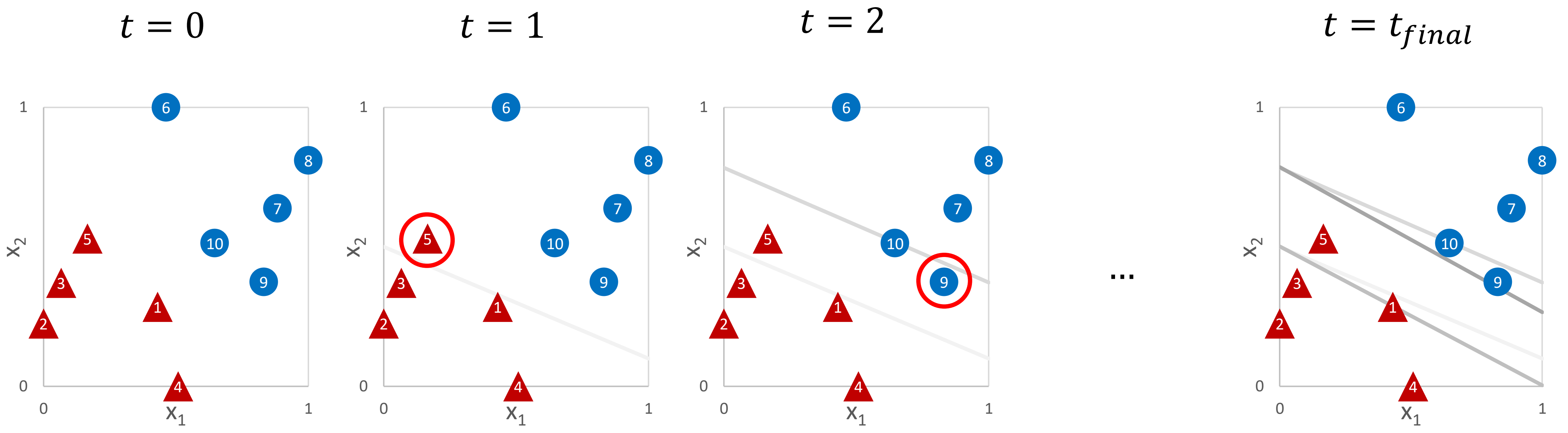
Perceptron Learning Algorithm

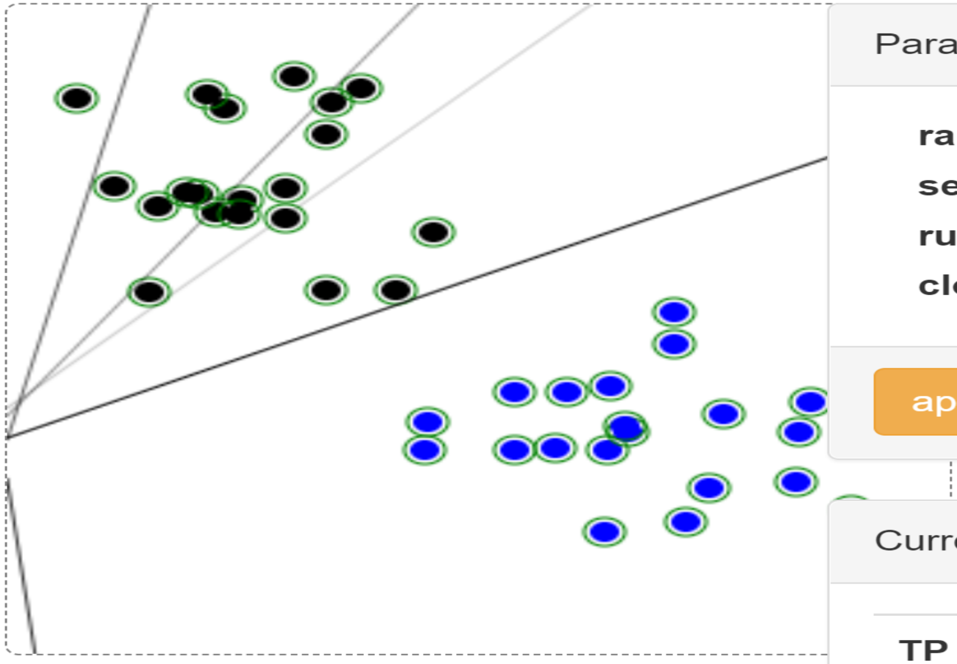
1. Initialize weights $\boldsymbol{w}$
   - Could be all zero, or random small values
2. For each instance i with features $\boldsymbol{x}^{(i)}$
   - Classify $\hat{y}^{(i)} = \text{sgn}\left(\boldsymbol{w}^\top \boldsymbol{x}^{(i)}\right)$
3. Select one misclassified instance
   - Update weights: $\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta(y - \hat{y})\boldsymbol{x}$
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





Parameters

random seed:	11	**base width:**	100
separator trail length:	5	**width variance:**	100
run delay (ms):	200	**base height:**	100
cloud size:	20	**height variance:**	50

apply params step run

Current Classification

TP	FP	TN	FN	Precision	Recall	F1
20	0	20	0	1.00	1.00	0.50

w(x) = -0.49607365285675603 x + 217

What are the differences?
Perceptron vs. Linear SVM

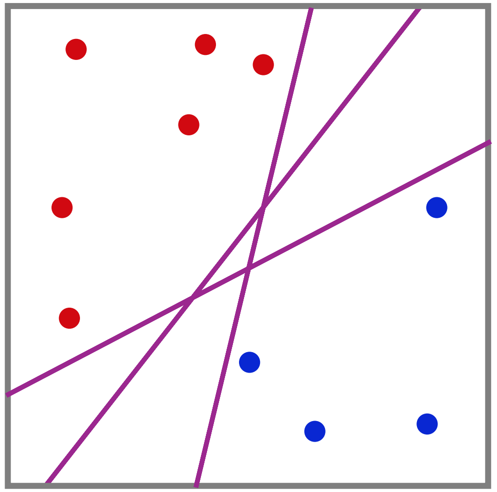
In Slack #lecture
1. Write to thread to suggest feature
2. Emote (👍 :+1:) to vote for feature

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

What are the differences?
Perceptron vs. Linear SVM

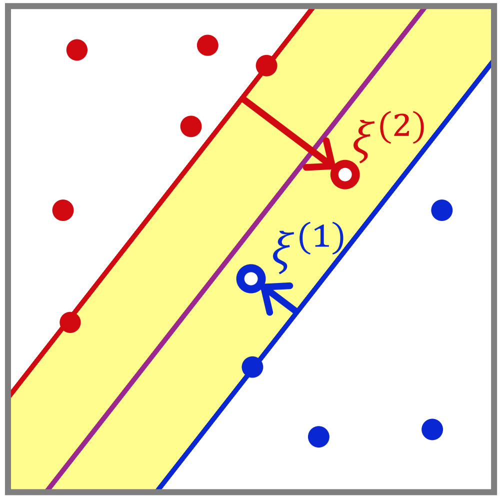
In Slack #lecture
1. Write to thread to suggest feature
2. Emote (👍 :+1:) to vote for feature

Perceptron



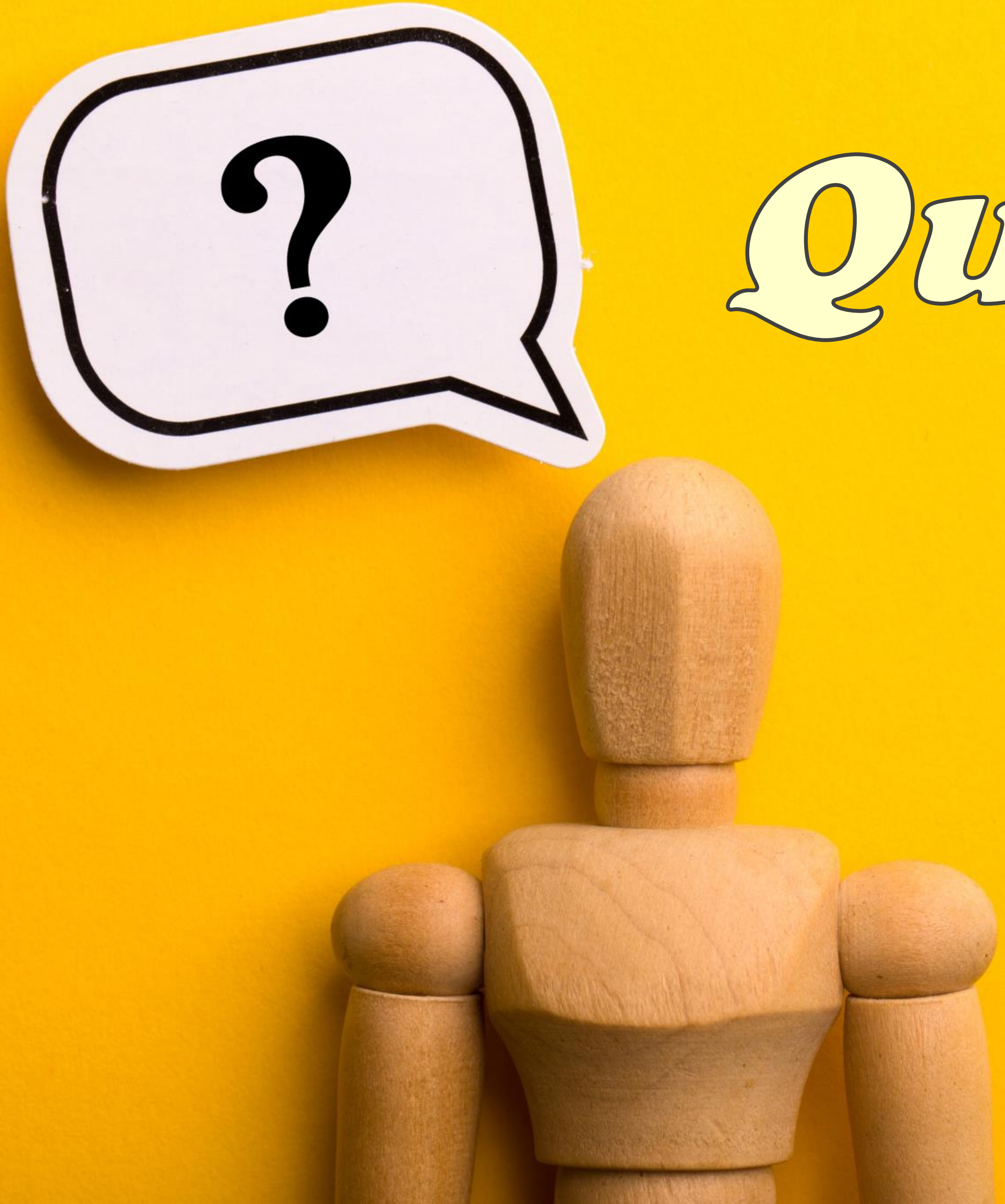
- Can select any model to linear => not robust (learns different weights for different initializations)
- Cannot converge on non-linearly separable data

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

- Perceptron is a linear classifier
- **Non-linear** classifiers
    - Other **activation functions**
        - Differentiable ones!
    - **Multiple** perceptrons / neurons
        - Multi-Layer Perceptron (MLP)

Activation Functions

Department of Computer Science
School of Computing

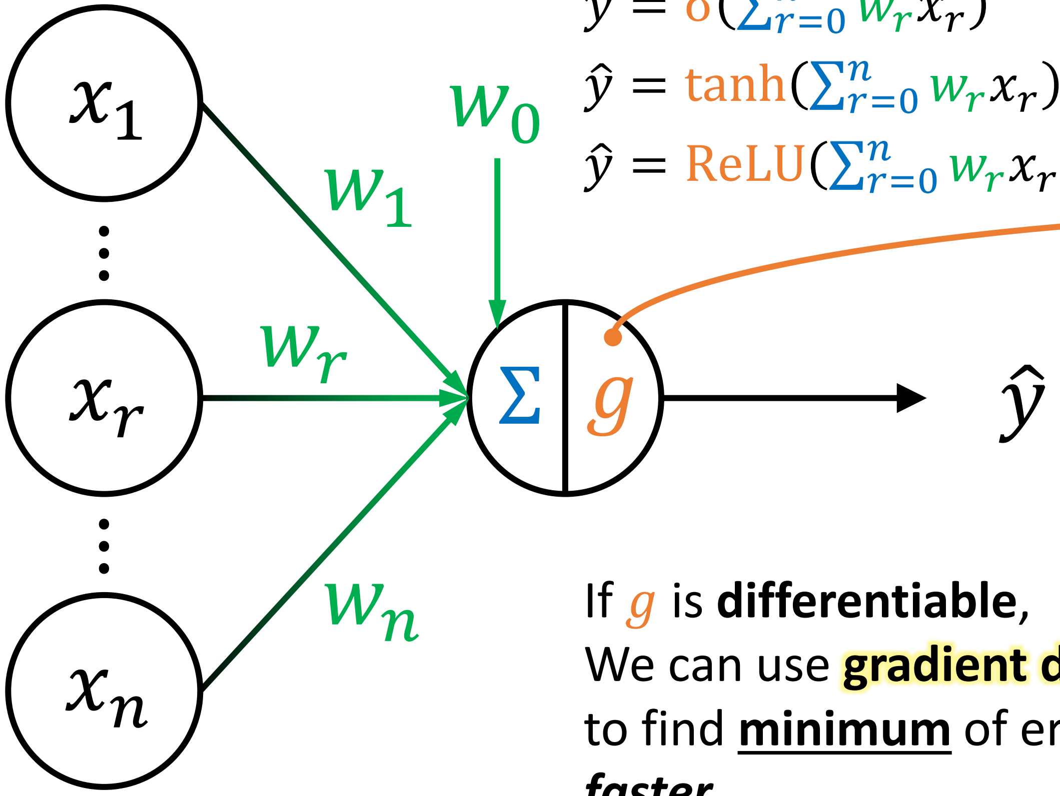
Activation Functions

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

$\hat{y} = \sigma(\sum_{r=0}^{n} w_r x_r)$

$\hat{y} = \tanh(\sum_{r=0}^{n} w_r x_r)$

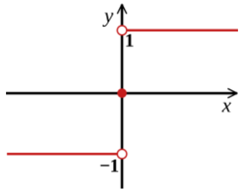
$\hat{y} = \mathrm{ReLU}(\sum_{r=0}^{n} w_r x_r)$



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

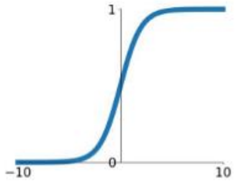
Step

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



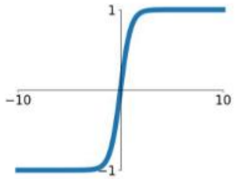
Sigmoid

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



tanh

$\tanh(x)$



ReLU

$\max(0, x)$

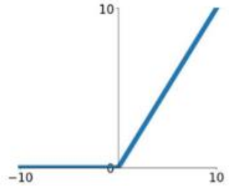


Image Credit:
https://miro.medium.com/max/1400/0*sIJ-gbjlz0zrz8lb.png

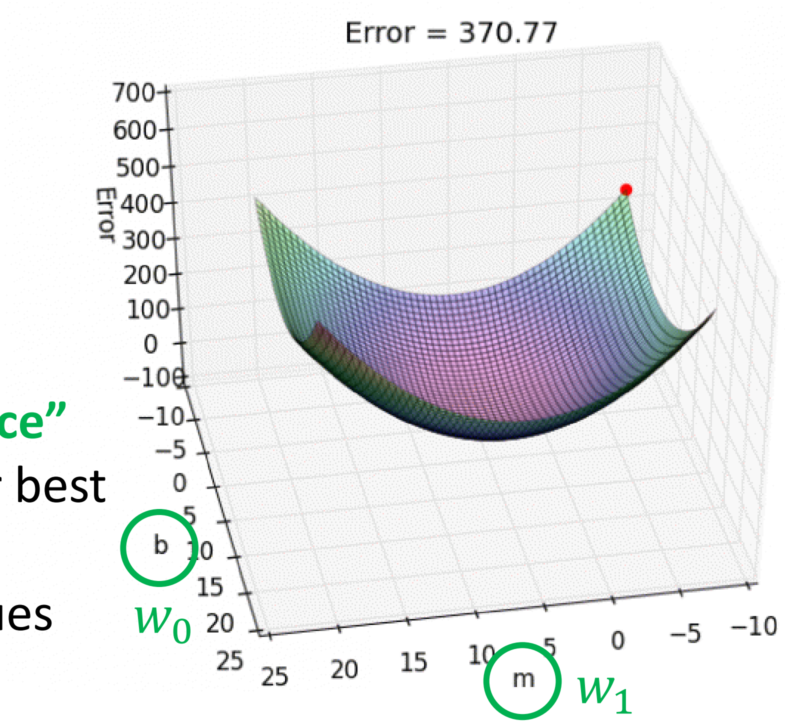
Gradient Descent

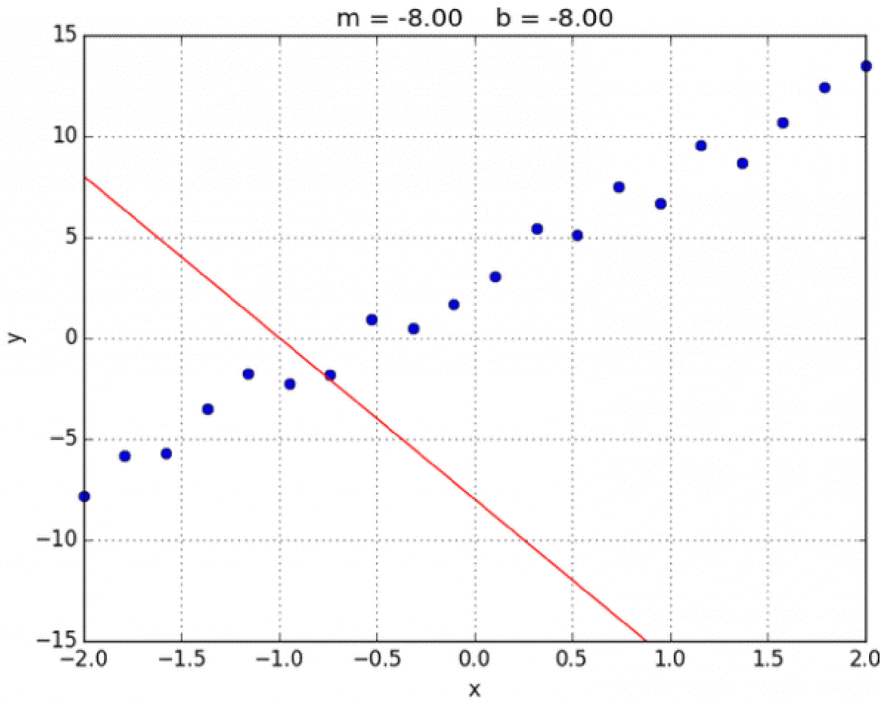
NUS National University of Singapore
Department of Computer Science
School of Computing

Optimization Goal:

Iteratively find w_r with minimum error ε

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

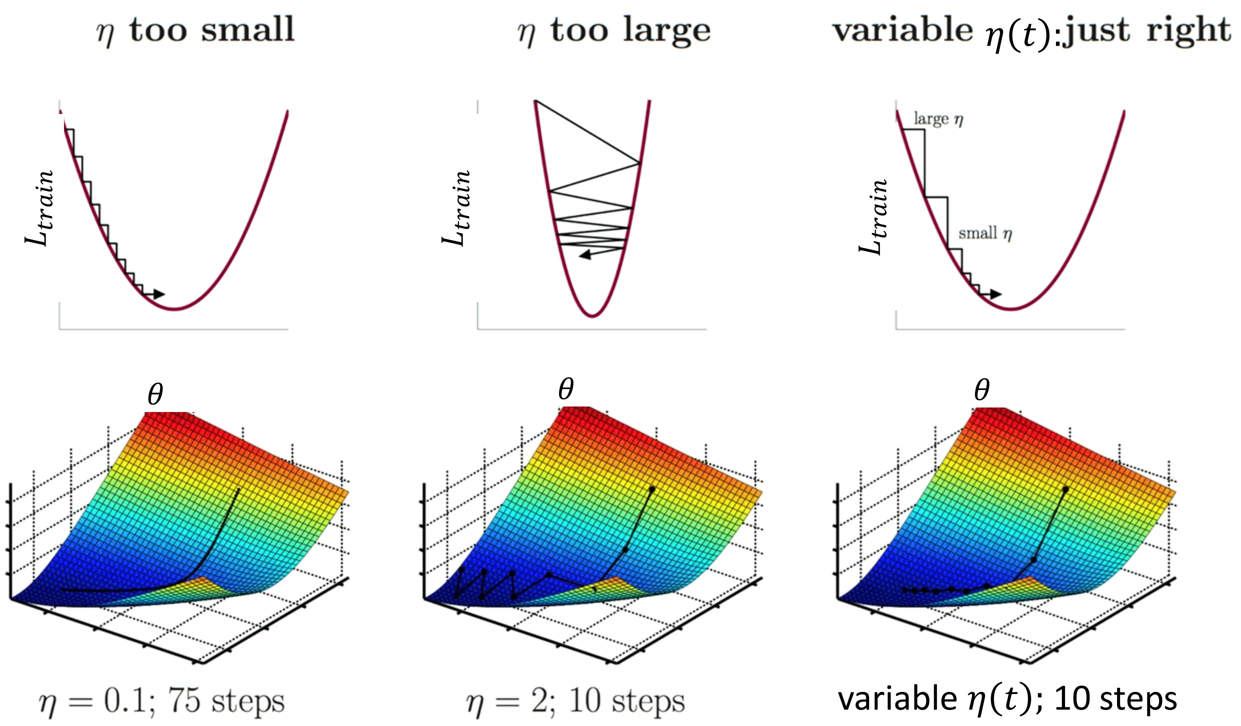




Credits: [Alykhan Tejani]'s Medium Post

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

Learning Rate η



These graphs are also in the "weight space".

Perceptron Weight Update

Old weight

Learning Error

$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta(y - \hat{y})\boldsymbol{x}$$

New weight

Learning Rate

Gradient Descent Weight Update

Direction of fastest error increase

Old weight

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \nabla \varepsilon$$

New weight

Learning Rate

How to calculate?

Gradient of error

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

"Weight space"

To search for best parameter (weight) values

Binary Cross-Entropy error (for classification)

$$\varepsilon = -y \log \hat{y}$$

Square Error (for regression)

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

Chain Rule

Consider composite function

Lagrange notation
Prime $'$ indicates first derivative relative to the function argument.
This can make writing derivatives more concise.
e.g., $y'(w) = dy/dw$

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

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

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

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 a deeper composite function

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

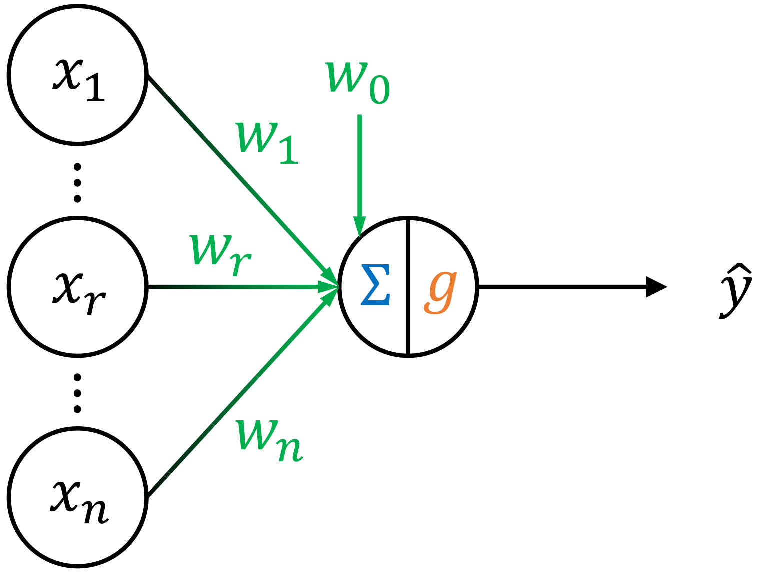
$$h = h(g), \qquad g = g(f), \qquad f = f(x)$$

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

Chain Rule

Multivariate

For single neuron



$$\varepsilon(\mathbf{w}) = \varepsilon\left(g\left(f(\mathbf{w})\right)\right)$$

$$\varepsilon = \varepsilon(g), \qquad \hat{y} = g, \qquad g = g(f), \qquad f = f(\mathbf{w})$$

$$\boxed{\nabla_{\mathbf{w}}\varepsilon = \frac{d\varepsilon}{d\mathbf{w}} = \frac{d\varepsilon}{dg}\frac{dg}{df}\frac{df}{d\mathbf{w}}}$$

Gradient of Weighted Sum

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

$$\frac{\partial f}{\partial w_r} = \frac{\partial}{\partial w_r}\left(w_r x_r + \sum_{\rho \neq r} w_\rho x_\rho \right)$$
$$= x_r + 0$$

$$\frac{\partial f}{\partial w_r} = x_r$$

$$f = \boldsymbol{w} \cdot \boldsymbol{x} = \boldsymbol{w}^\top \boldsymbol{x}$$

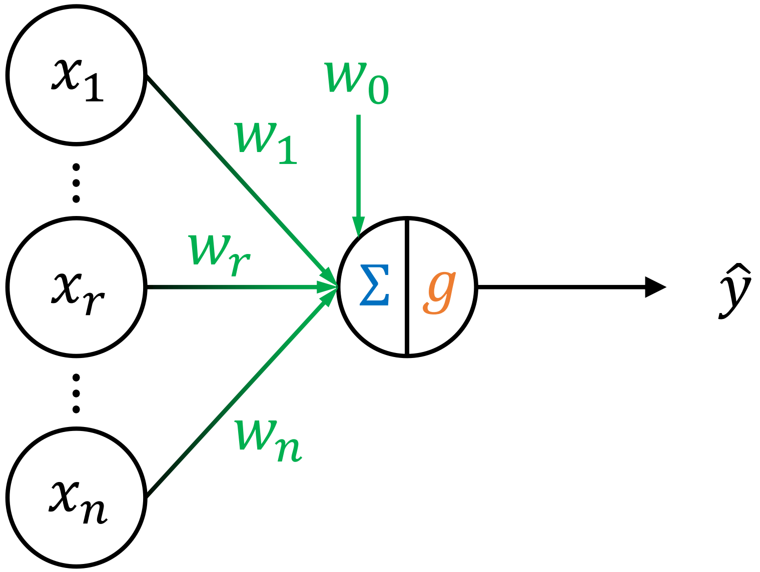
$$\frac{df}{d\boldsymbol{w}} = \sum_{r=0}^{n} \frac{\partial f}{\partial w_r} \boldsymbol{e}_r = \begin{pmatrix} \partial f / \partial w_0 \\ \vdots \\ \partial f / \partial w_r \\ \vdots \\ \partial f / \partial w_n \end{pmatrix} = \begin{pmatrix} x_0 \\ \vdots \\ x_r \\ \vdots \\ x_n \end{pmatrix}$$

$$\nabla_{\boldsymbol{w}} f = \frac{df}{d\boldsymbol{w}} = \boldsymbol{x}$$

Chain Rule

Multivariate

For single neuron



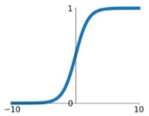
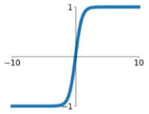
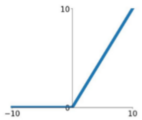
$$\varepsilon(\mathbf{w}) = \varepsilon\Big(g\big(f(\mathbf{w})\big)\Big)$$

$$\varepsilon = \varepsilon(g), \qquad \hat{y} = g, \qquad g = g(f), \qquad f = \mathbf{w}^\top \mathbf{x}$$

$$\boxed{\nabla_{\mathbf{w}}\varepsilon = \frac{d\varepsilon}{d\mathbf{w}} = \frac{d\varepsilon}{dg}\frac{dg}{df}\mathbf{x}}$$

Calculating gradient for single neuron

Error ε		$\frac{d\varepsilon}{d\hat{y}}$
Square Error	$\frac{1}{2}(y-\hat{y})^2$	$-(y-\hat{y})$
Binary Cross Entropy	$-y\log\hat{y}$	$-\frac{y}{\hat{y}}$

Activation $\hat{y} = g(f)$			$\frac{dg}{df}$
Sigmoid		$\frac{1}{1+e^{-f}}$	$(1-g)g$
tanh		$\tanh f$	$1-g^2$
ReLU		$\max(0, f)$	$[f>0]$

Advanced: further reading

Weighted Sum $f(\boldsymbol{w})$	$\frac{df}{d\boldsymbol{w}}$
$\boldsymbol{w}^\top \boldsymbol{x}$	$\boldsymbol{x}$

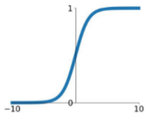
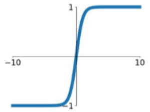
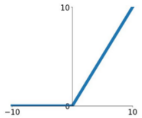
Derivative of sigmoid function σ

$$\sigma'(x) = \big(1 - \sigma(x)\big)\sigma(x)$$

- Proof
  - $\sigma(x) = \frac{e^x}{1+e^x} = \frac{1}{1+e^{-x}}$
  - Rewrite as compound function
    - $\sigma(\chi) = \frac{1}{1+\chi}$, where $\chi(x) = e^{-x}$
  - Using chain rule
    - $\sigma'(x) = \frac{dg}{dx} = \frac{dg}{d\chi}\frac{d\chi}{dx} = \frac{-1}{(1+\chi)^2}(-e^{-x}) = \frac{e^{-x}}{1+e^{-x}}\frac{1}{1+e^{-x}} = \frac{e^{-x}}{1+e^{-x}}\sigma(x)$
  - 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:
    - $\sigma'(x) = \big(1 - \sigma(x)\big)\sigma(x)$

Calculating gradient for single neuron

Error ε		$\frac{d\varepsilon}{d\hat{y}}$
Square Error	$\frac{1}{2}(y-\hat{y})^2$	$-(y-\hat{y})$
Binary Cross Entropy	$-y\log\hat{y}$	$-\frac{y}{\hat{y}}$

Activation $\boxed{\hat{y}=g(f)}$			$\frac{dg}{df}$
Sigmoid		$\frac{1}{1+e^{-f}}$	$(1-g)g$
tanh		$\tanh f$	$1-g^2$
ReLU		$\max(0,f)$	$[f>0]$

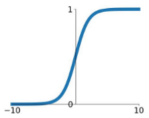
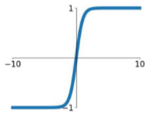
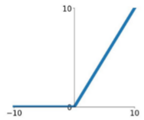
Weighted Sum $f(\boldsymbol{w})$	$\frac{df}{d\boldsymbol{w}}$
$\boldsymbol{w}^\top\boldsymbol{x}$	$\boldsymbol{x}$

$$\varepsilon = -y\log\hat{y} = -y\log g = -y\log(\max(0,f)) = -y\log(\max(0,\boldsymbol{w}^\top\boldsymbol{x}))$$

$$\nabla_{\boldsymbol{w}}\varepsilon = \frac{d\varepsilon}{d\hat{y}}\frac{dg}{df}\frac{df}{d\boldsymbol{w}} = -\frac{y}{\hat{y}}[f>0]\boldsymbol{x} = -\frac{y}{\max(0,\boldsymbol{w}^\top\boldsymbol{x})}[\boldsymbol{w}^\top\boldsymbol{x}>0]\boldsymbol{x}$$

Calculating gradient for single neuron

Error ε		$\frac{d\varepsilon}{d\hat{y}}$
Square Error	$\frac{1}{2}(y-\hat{y})^2$	$-(y-\hat{y})$
Binary Cross Entropy	$-y\log\hat{y}$	$-\frac{y}{\hat{y}}$

Activation $\boxed{\hat{y} = g(f)}$			$\frac{dg}{df}$
Sigmoid		$\frac{1}{1+e^{-f}}$	$(1-g)g$
tanh		$\tanh f$	$1-g^2$
ReLU		$\max(0,f)$	$[f>0]$

Weighted Sum $f(\boldsymbol{w})$	$\frac{df}{d\boldsymbol{w}}$
$\boldsymbol{w}^\top\boldsymbol{x}$	$\boldsymbol{x}$

$$\varepsilon = \frac{1}{2}(y-\hat{y})^2 = \frac{1}{2}(y-g)^2 = \frac{1}{2}\left(y-\frac{1}{1+e^{-f}}\right)^2 = \frac{1}{2}\left(y-\frac{1}{1+e^{-\boldsymbol{w}^\top\boldsymbol{x}}}\right)^2$$

$$\nabla_{\boldsymbol{w}}\varepsilon = \frac{d\varepsilon}{d\hat{y}}\frac{dg}{df}\frac{df}{d\boldsymbol{w}} = -(y-\hat{y})(1-\hat{y})\hat{y}\boldsymbol{x}$$


Questions!

have a

Extending the Perceptron

- Perceptron is a linear classifier
- **Non-linear** classifiers
  - Other **activation functions**
    - Differentiable ones!
  - **Multiple** perceptrons / neurons
    - Multi-Layer Perceptron (MLP)

Feed-forward Neural Network

Artificial Neural Networks (ANN)

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

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

Artificial NNs are *inspired* by, but *not mimicking* of **Human NNs**

1 Layers and Uni-directional inference

Artificial NNs usually uses a sequence of layers with specific order to determine an output. In human brain, there is no fixed order and often async. … Artificial NN are mostly feed forward; output cannot then affect input. This is unlike Human NNs, which have cyclic loops in the neural structure.

2 Non-diverse neurons and structures [W10]

Artificial NNs are made up of only one type of simple neuron. Human NNs are made up of many kinds of neurons.
Remedy: *Convolutional neuron for images, Recurrent neuron for sequence.*

3 Data-specific [W10]

Human NNs can do many task (e.g., recognize sound, image, *and* text). Artificial NNs, it will only be able to suit one task at a time (e.g., either image *or* text).
Remedy: *CNNs for images and RNNs for text can be combined.*

4 Deterministic

For the same input the Artificial NN will give the same output; but this may not apply to Human NNs.
Research: *Bayesian neural networks include randomization at inference to predict more robustly.*

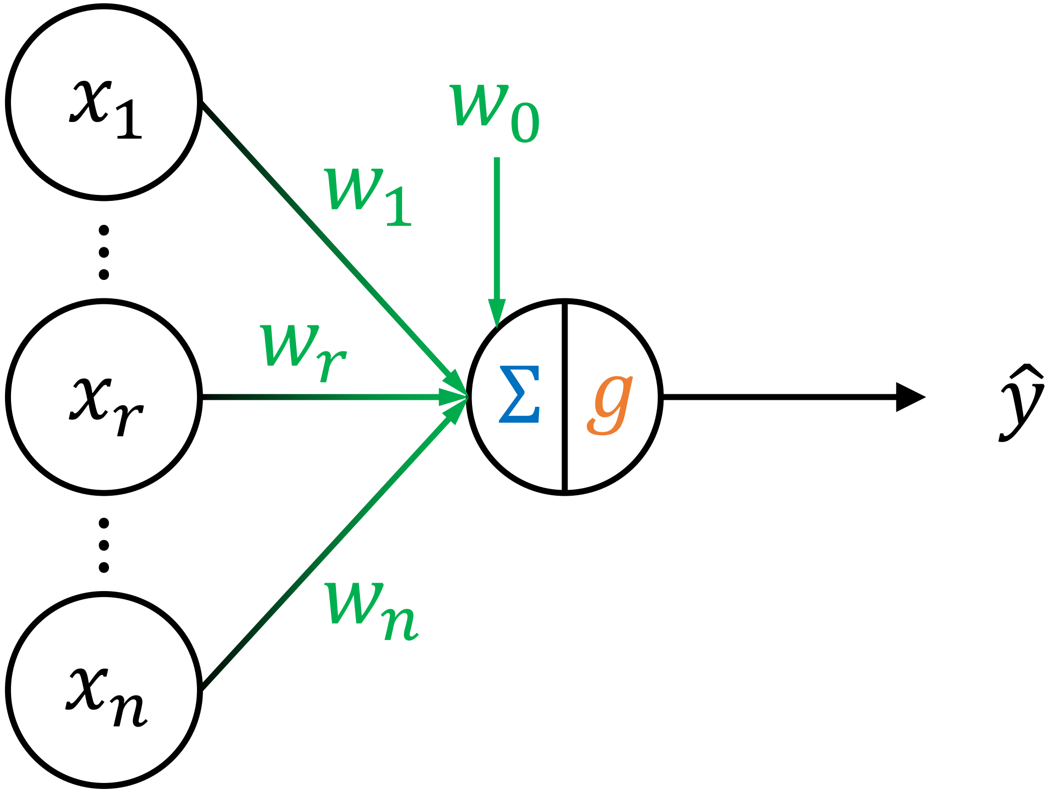
5 Energy efficiency of computations

Human NNs are more energy efficient than Artificial NN.
Research: *Spiking neural networks are being developed to be lower energy.*

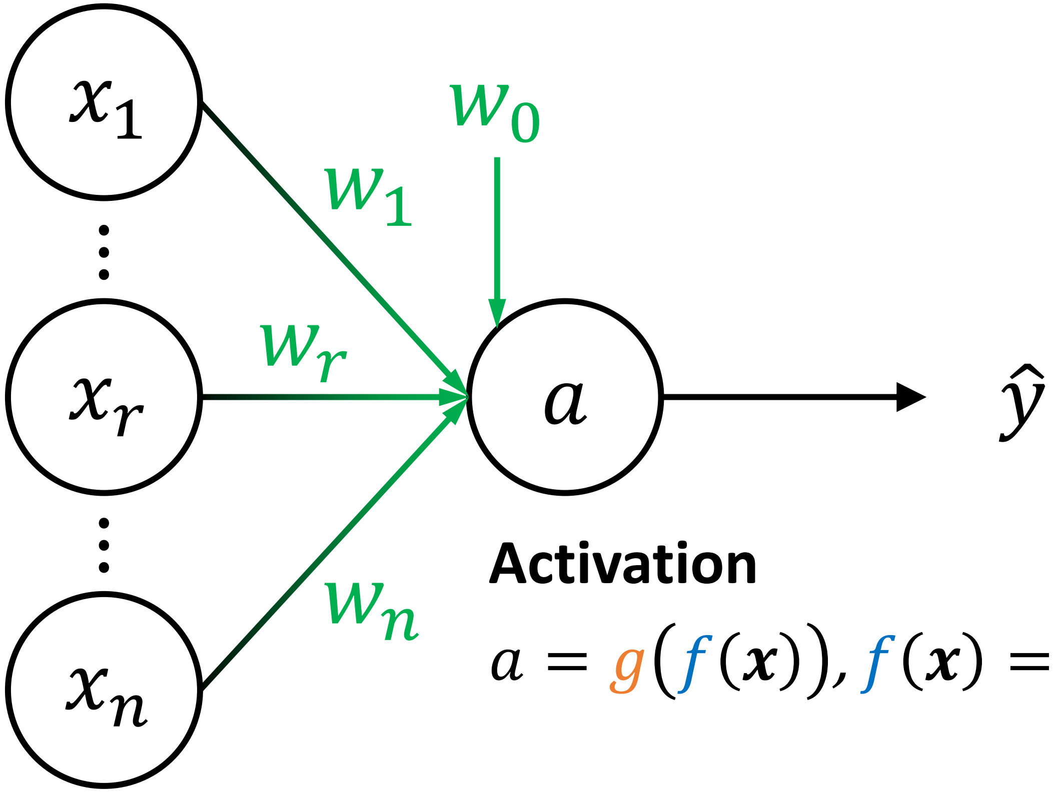
6 Forgetting and Unlearning

Human can forget but Artificial NN will not. Once the Artificial NN is trained, it will remember what it learns and it becomes permanent knowledge.
Research: *Model unlearning can enable models to forget instances for legal privacy requirements.*

Single-Layer Perceptron



Single-Layer Perceptron

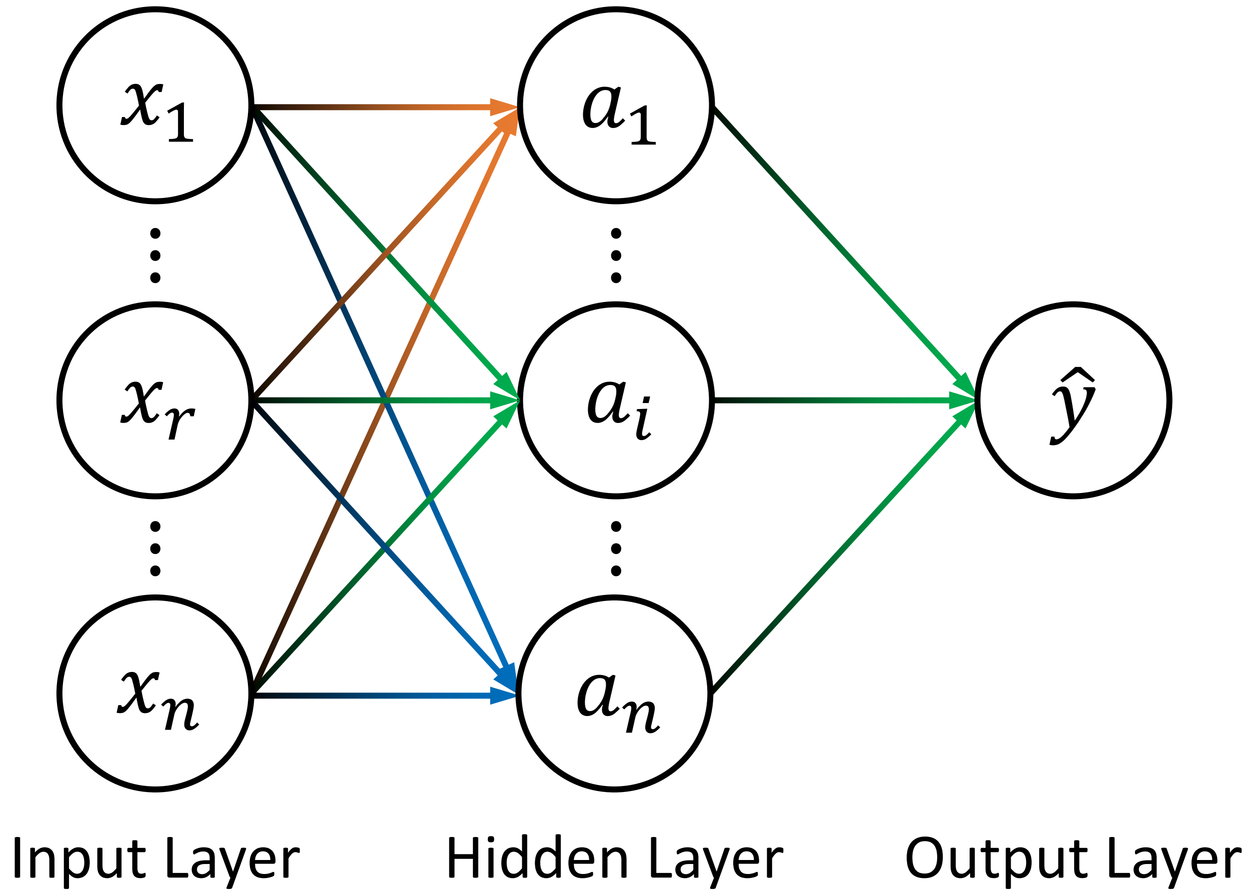


Activation

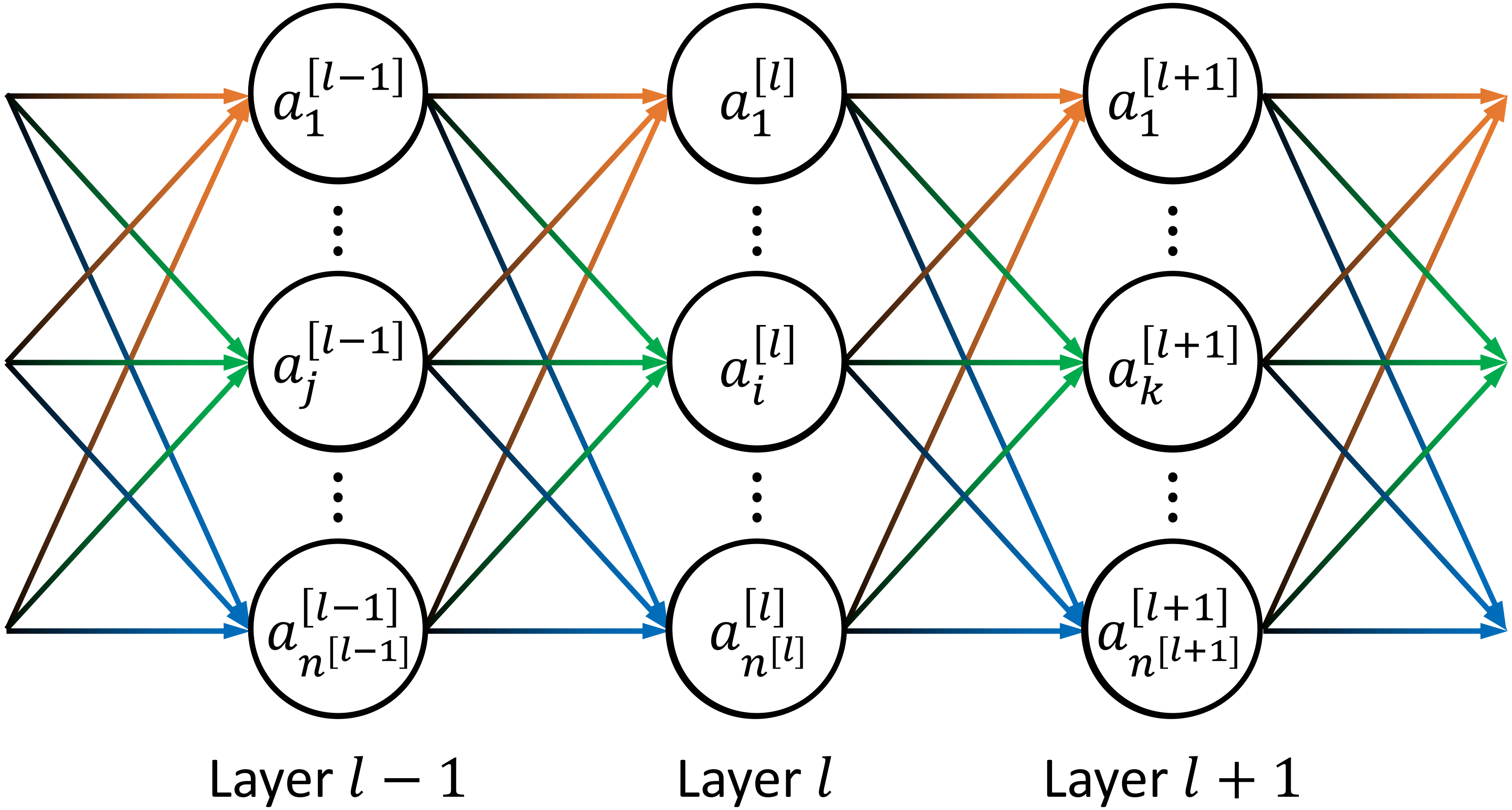
$$a = g(f(\boldsymbol{x})), f(\boldsymbol{x}) = \boldsymbol{w} \cdot \boldsymbol{x}$$

Multi-Layer Perceptron (Neural Network)

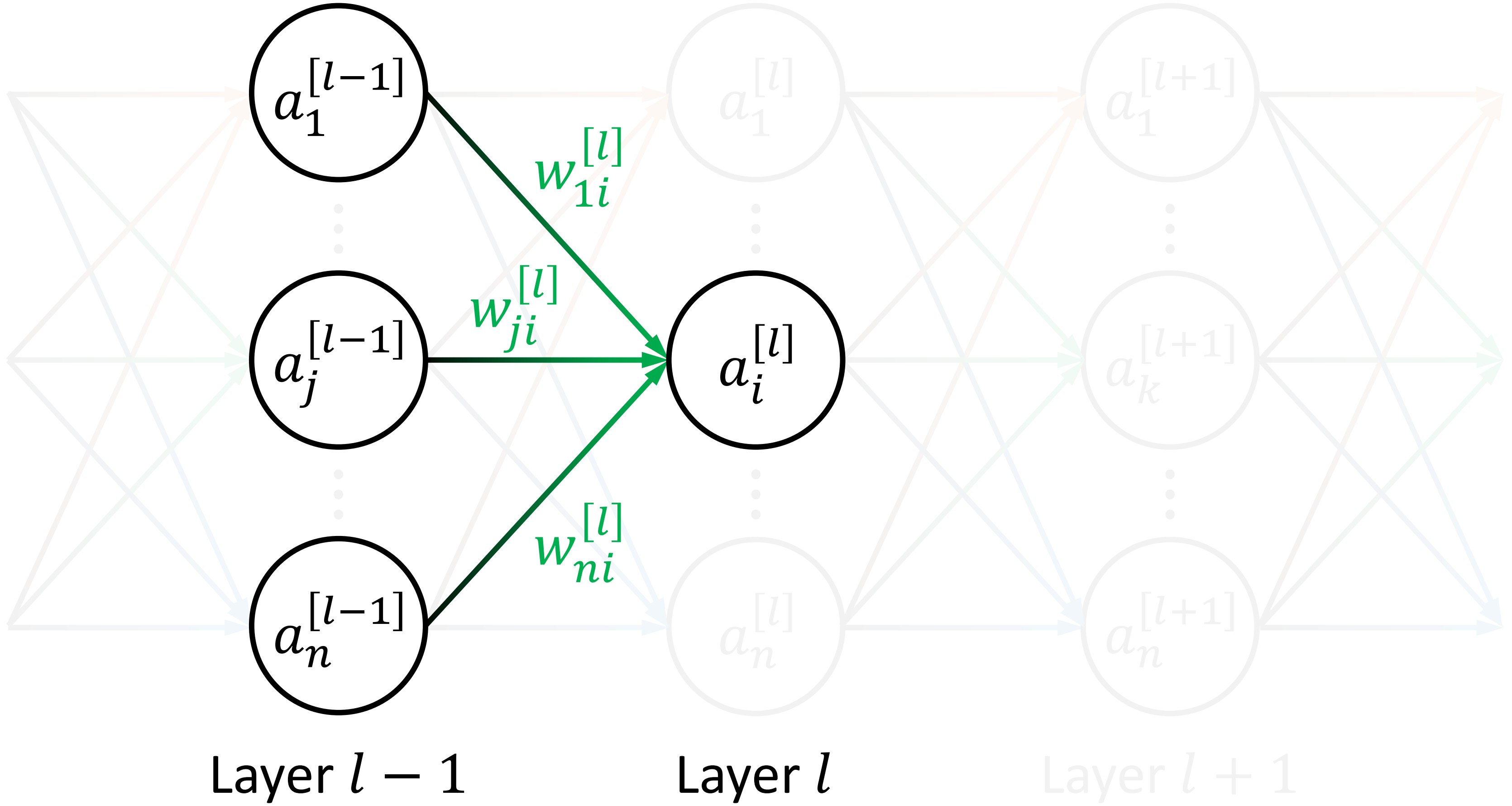
2-Layer MLP



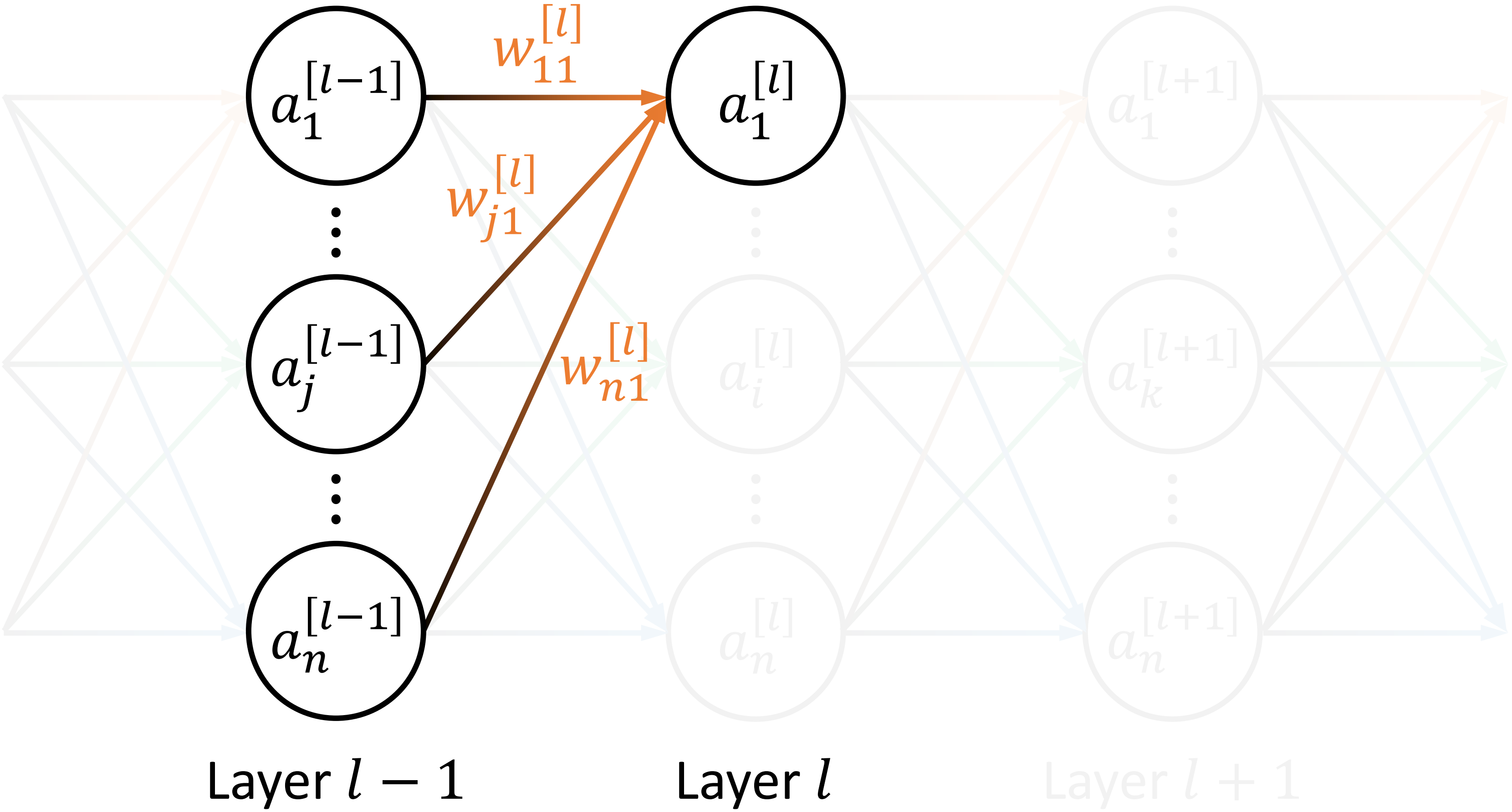
Multi-Layer Perceptron (Neural Network)



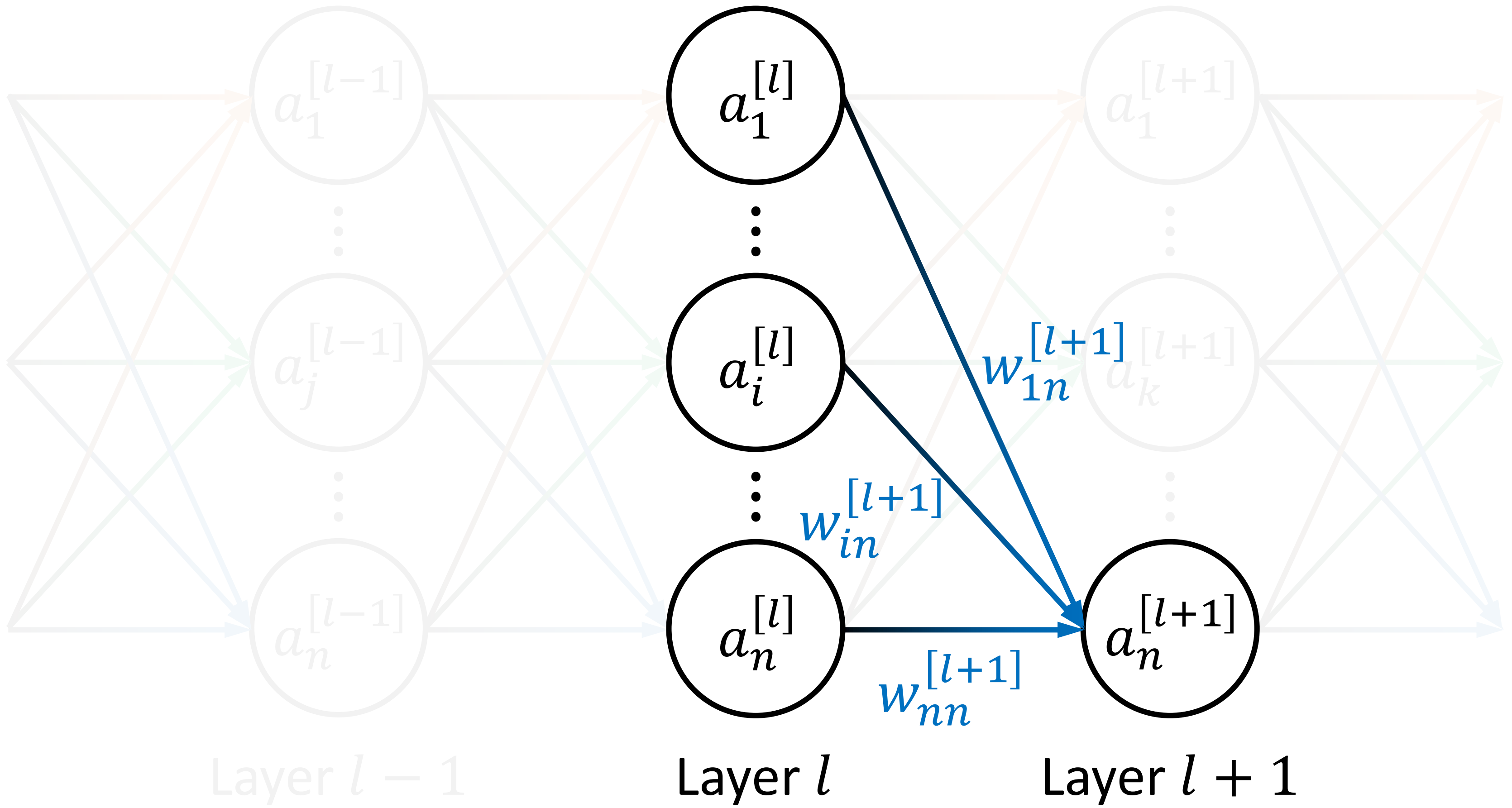
Multi-Layer Perceptron (Neural Network)



Multi-Layer Perceptron (Neural Network)



Multi-Layer Perceptron (Neural Network)

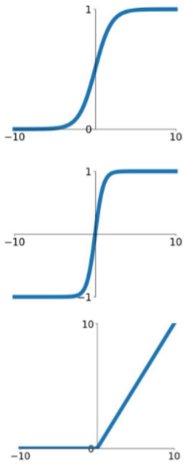


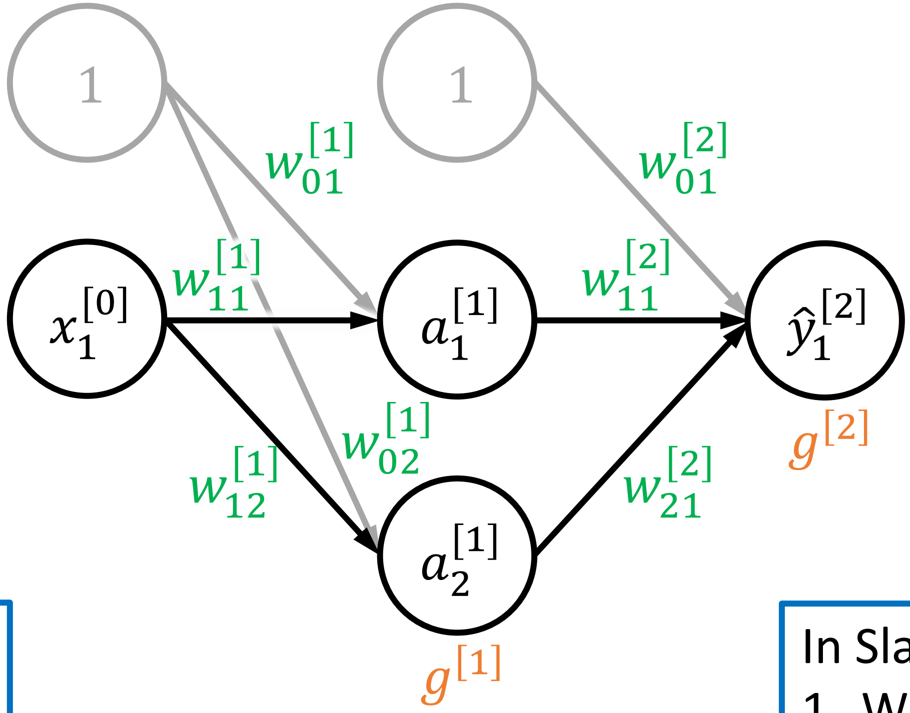
Fitting **non-linear function** with MLP
What model weights can model $\hat{y} = |x - 1|$?

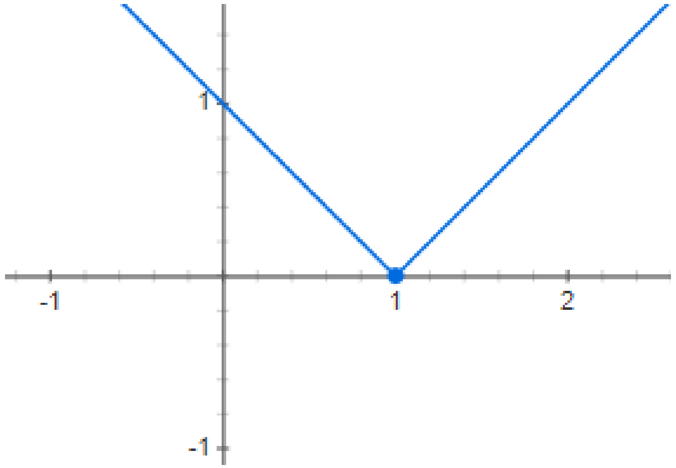
Sigmoid
$\sigma(x) = \frac{1}{1+e^{-x}}$

tanh
$\tanh(x)$

ReLU
$\max(0, x)$





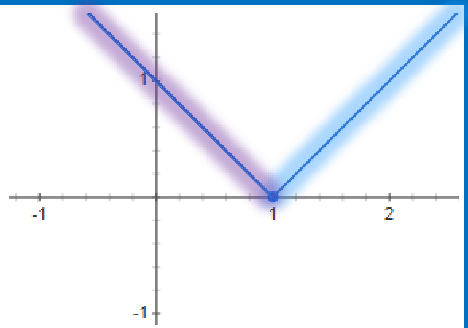


Bonus Question:
What activation function(s) g should you use for each layer?

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

Fitting **non-linear function** with MLP

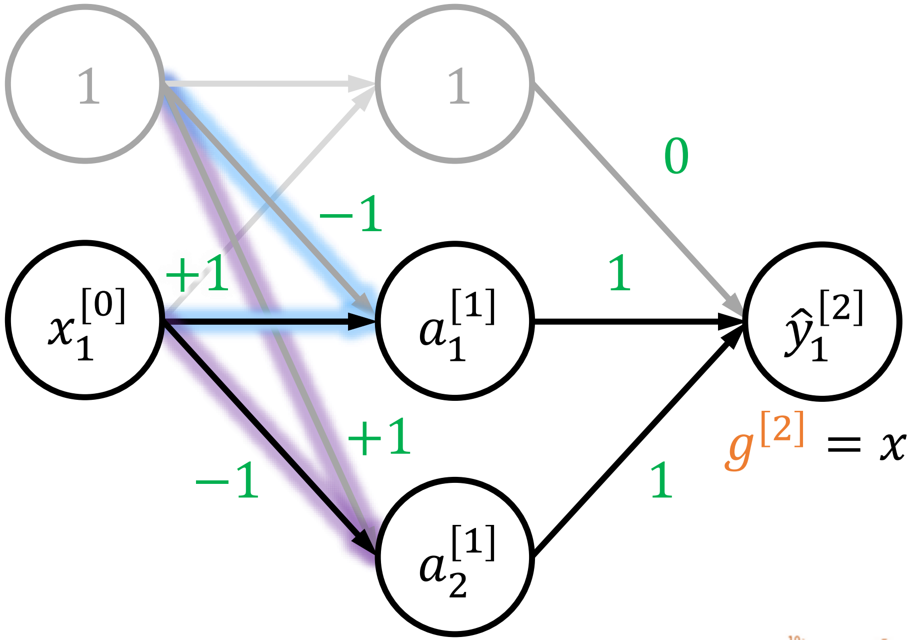
What model weights can model $\hat{y} = |x - 1|$?



$$\boldsymbol{W}^{[1]} = \begin{pmatrix} 1 & w_{01}^{[1]} & w_{02}^{[1]} \\ 0 & w_{11}^{[1]} & w_{12}^{[1]} \end{pmatrix} = \begin{pmatrix} 1 & -1 & 1 \\ 0 & 1 & -1 \end{pmatrix}$$

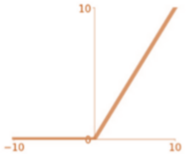
$$\boldsymbol{x}^{[0]} = \begin{pmatrix} 1 \\ x_1^{[0]} \end{pmatrix}$$

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



$g^{[2]} = x$

$$g^{[1]} = \mathrm{ReLU} = \max(0, x)$$



$$\boldsymbol{W}^{[2]} = \begin{pmatrix} w_{01}^{[2]} \\ w_{11}^{[2]} \\ w_{21}^{[2]} \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}$$

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

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

Questions!

Abs(x-1) as sum of ReLU

desmos

Log In or Sign Up

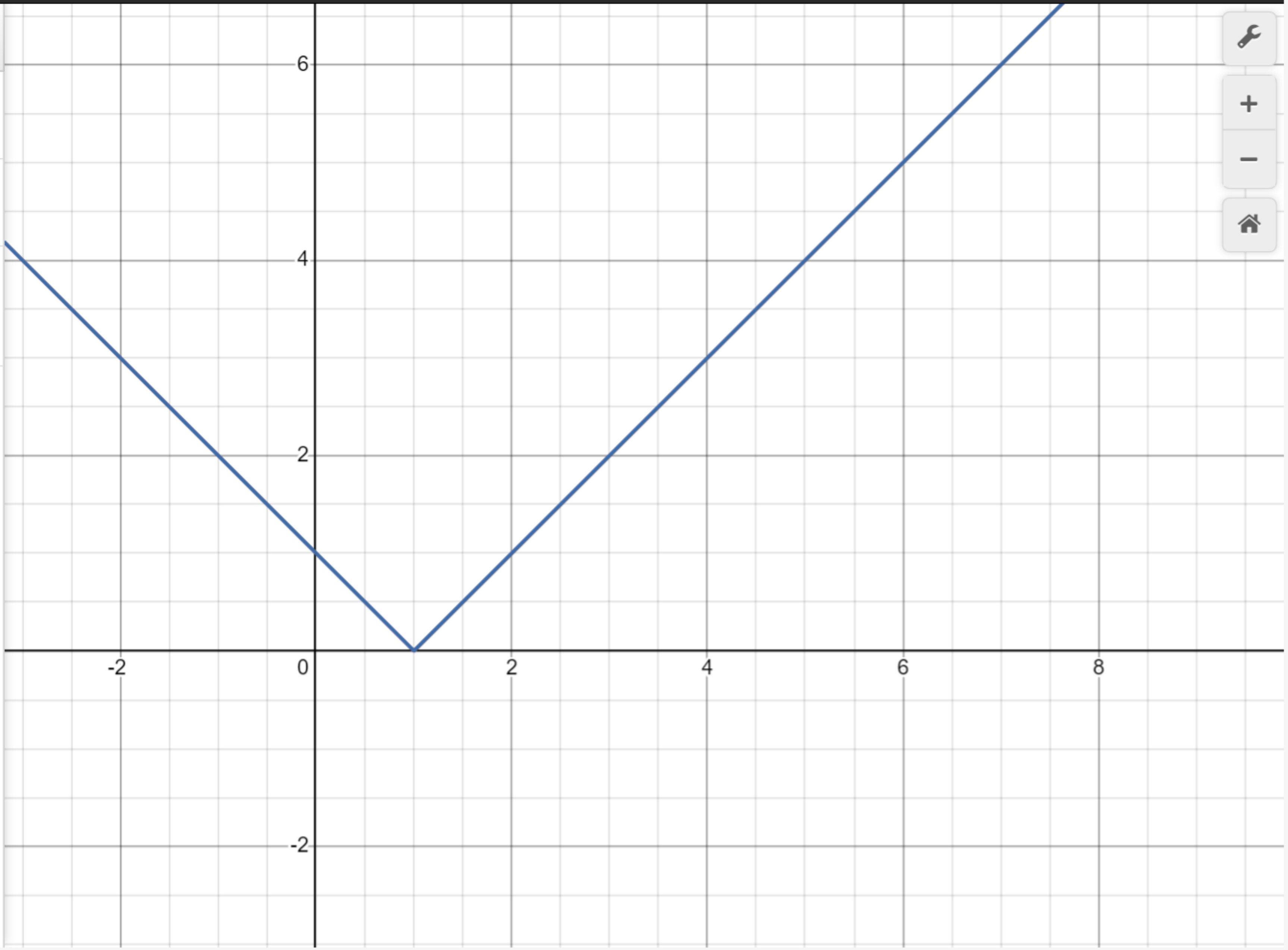
1 $y = |x-1|$

2 $y = \max(0, x-1) + \max(0, 1-x) + a$

3 $a = 0$

0 2

4



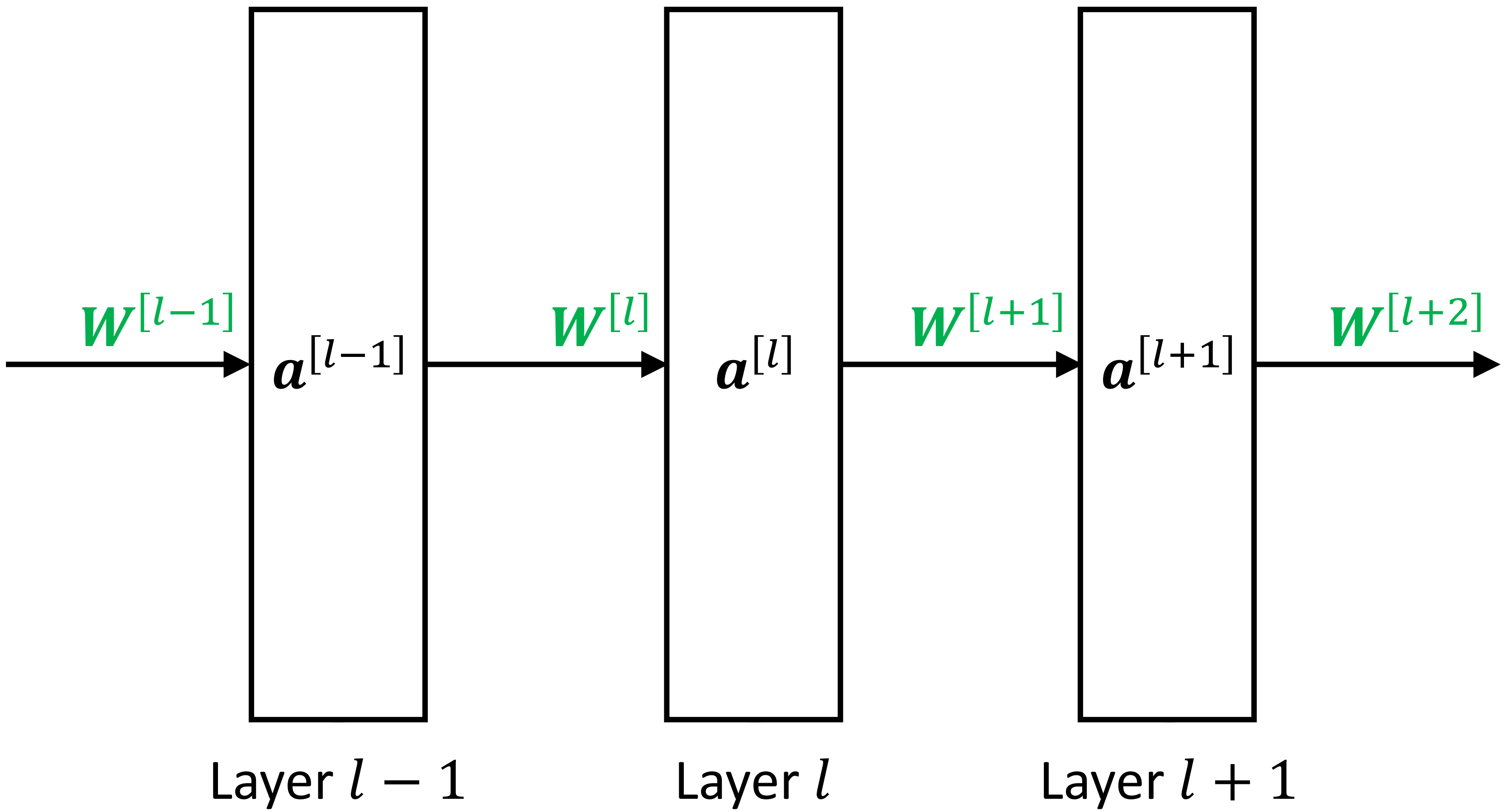
powered by desmos

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

Web Viewer Terms | Privacy & Cookies

Edit

Neural Network (vector notation)



Layer Activation

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

Single-Layer Perceptron

Layer l Activation Function

Layer l Weights

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

Layer l Activations

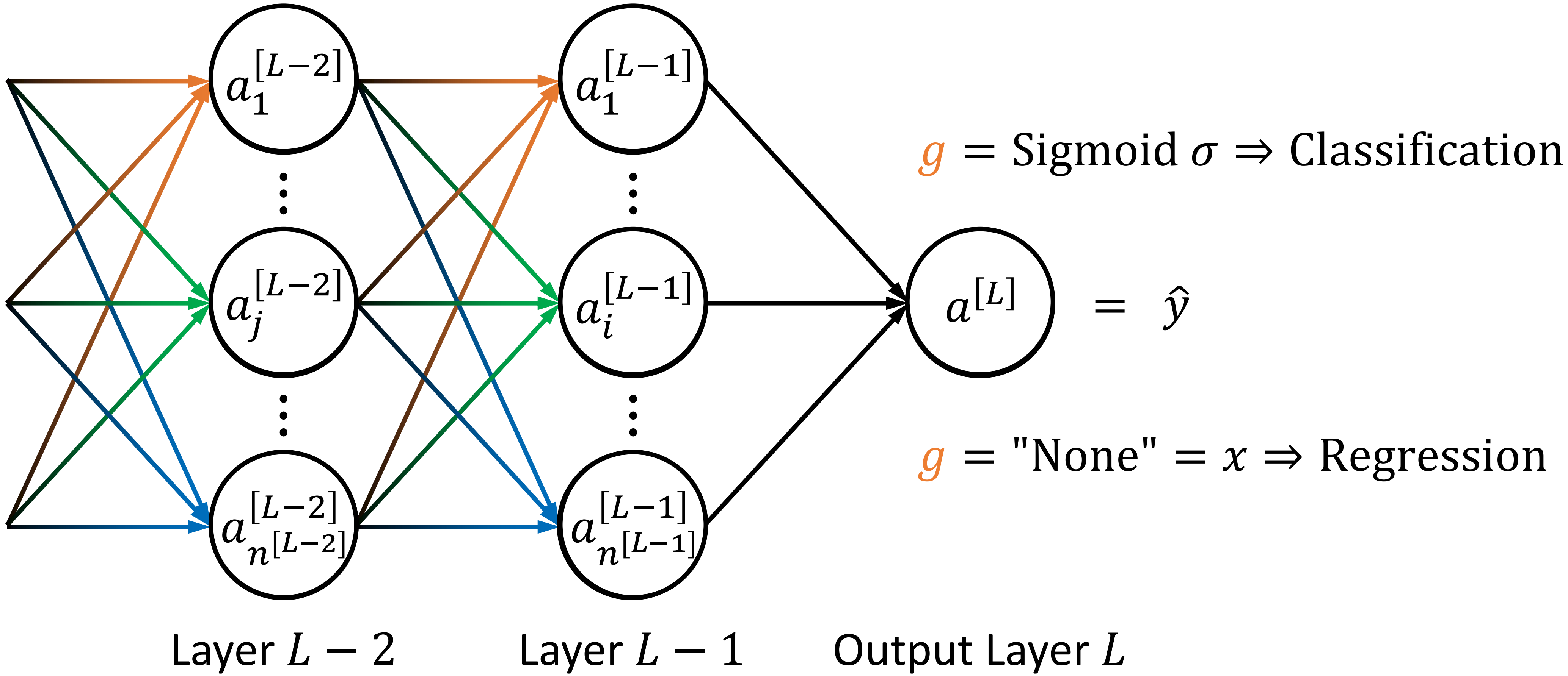
Layer $l-1$ Activations

Layer l in Neural Network

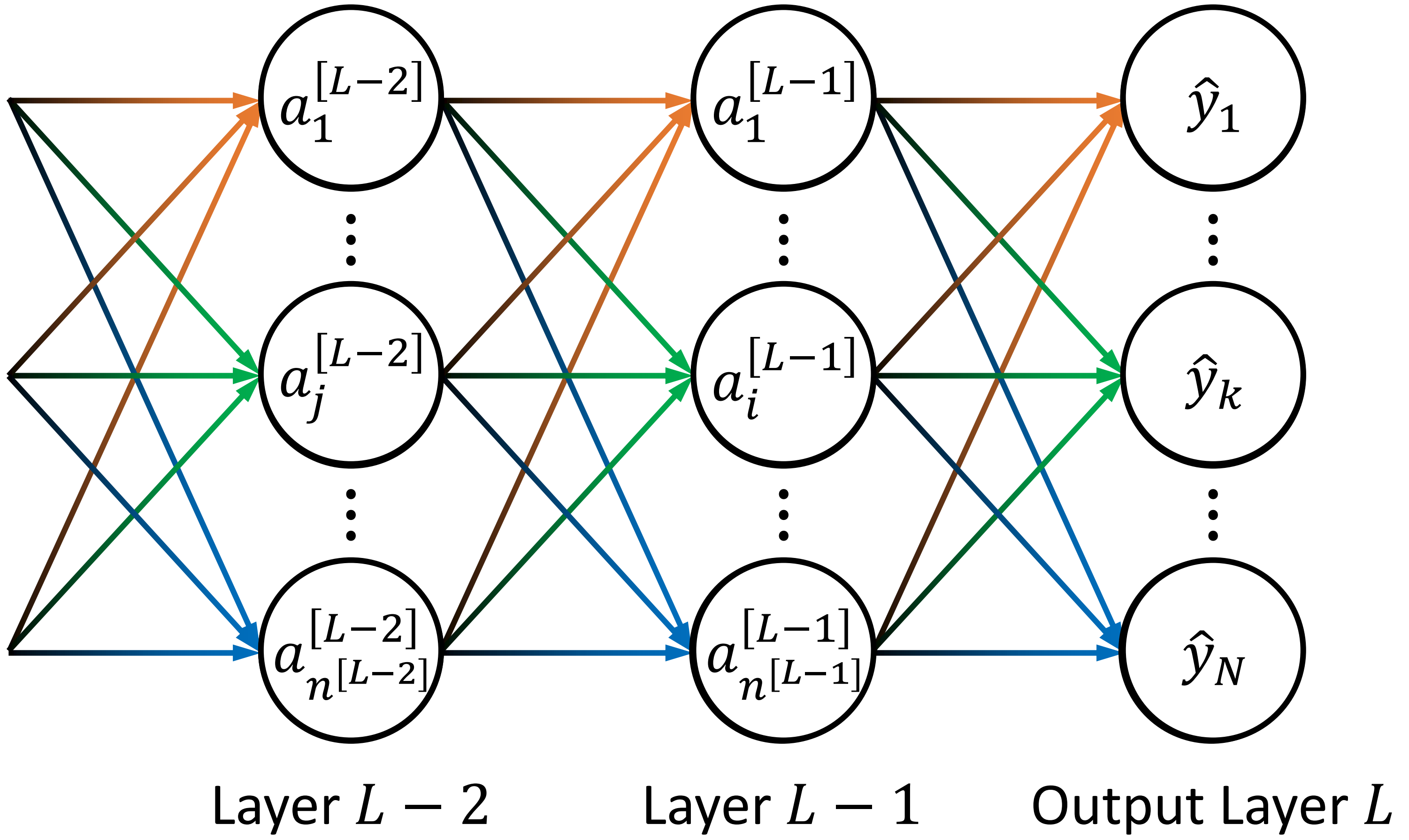
Multiclass / Multilabel Neural Networks

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

Single-output Neural Network
(binary classification or scalar regression)



Multiple outputs Neural Network
(Multiple classifications or multivariate regression)

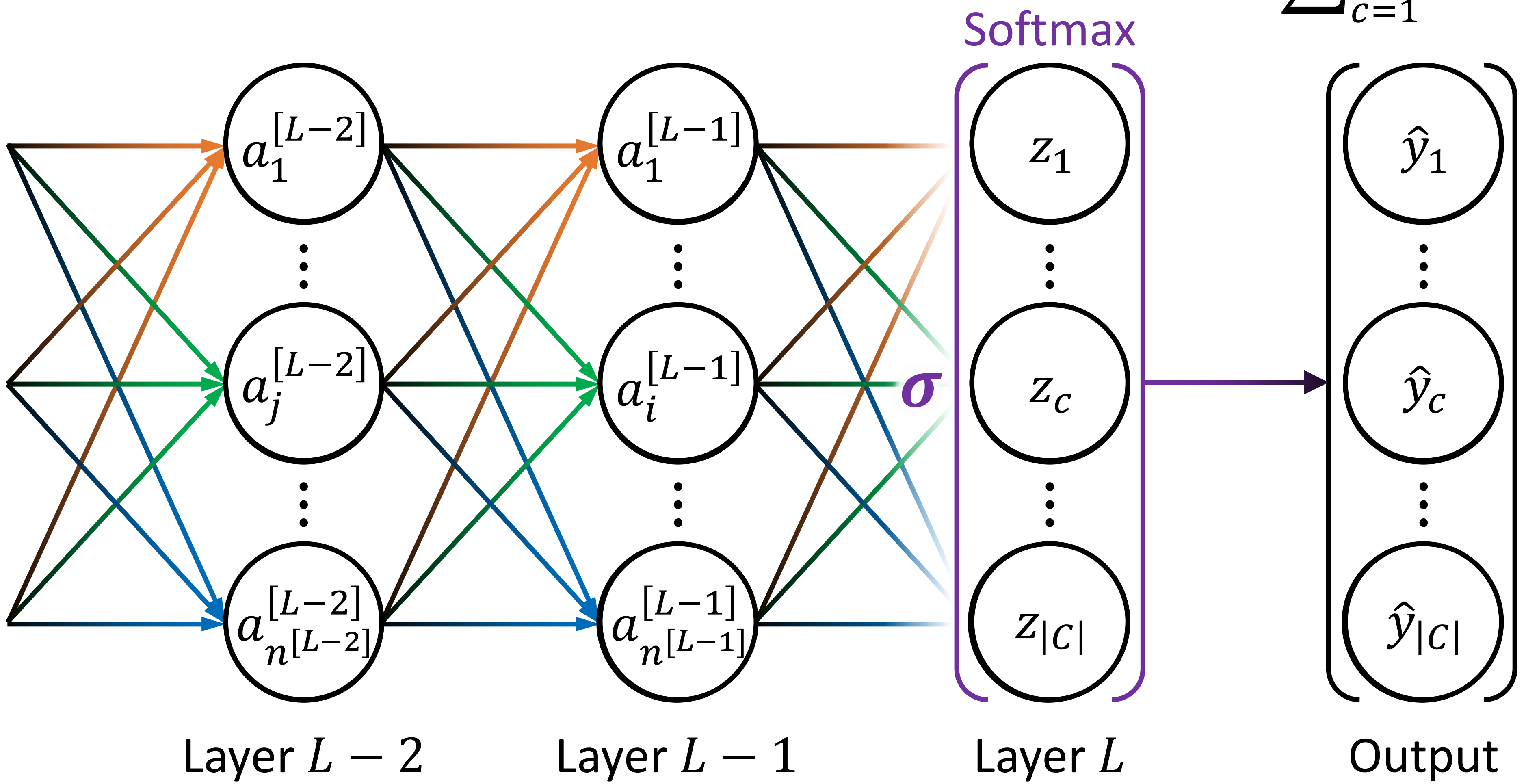


Multi-label

Multiple outputs Neural Network

(Multiclass classification)

$$\sum_{c=1}^{|C|} \hat{y}_c = 1$$



Multi-class classification

Sigmoid vs. Softmax

Sigmoid

- *Is prediction true?*
- For **binary** classification
- $\sigma(z) = \frac{e^z}{1+e^z}$

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

Multiple Sigmoids

- *Which predictions are true?*
- For **multi-label** classification
- $\sigma(z_k) = \frac{e^{z_k}}{1+e^{z_k}}, \ \forall k$

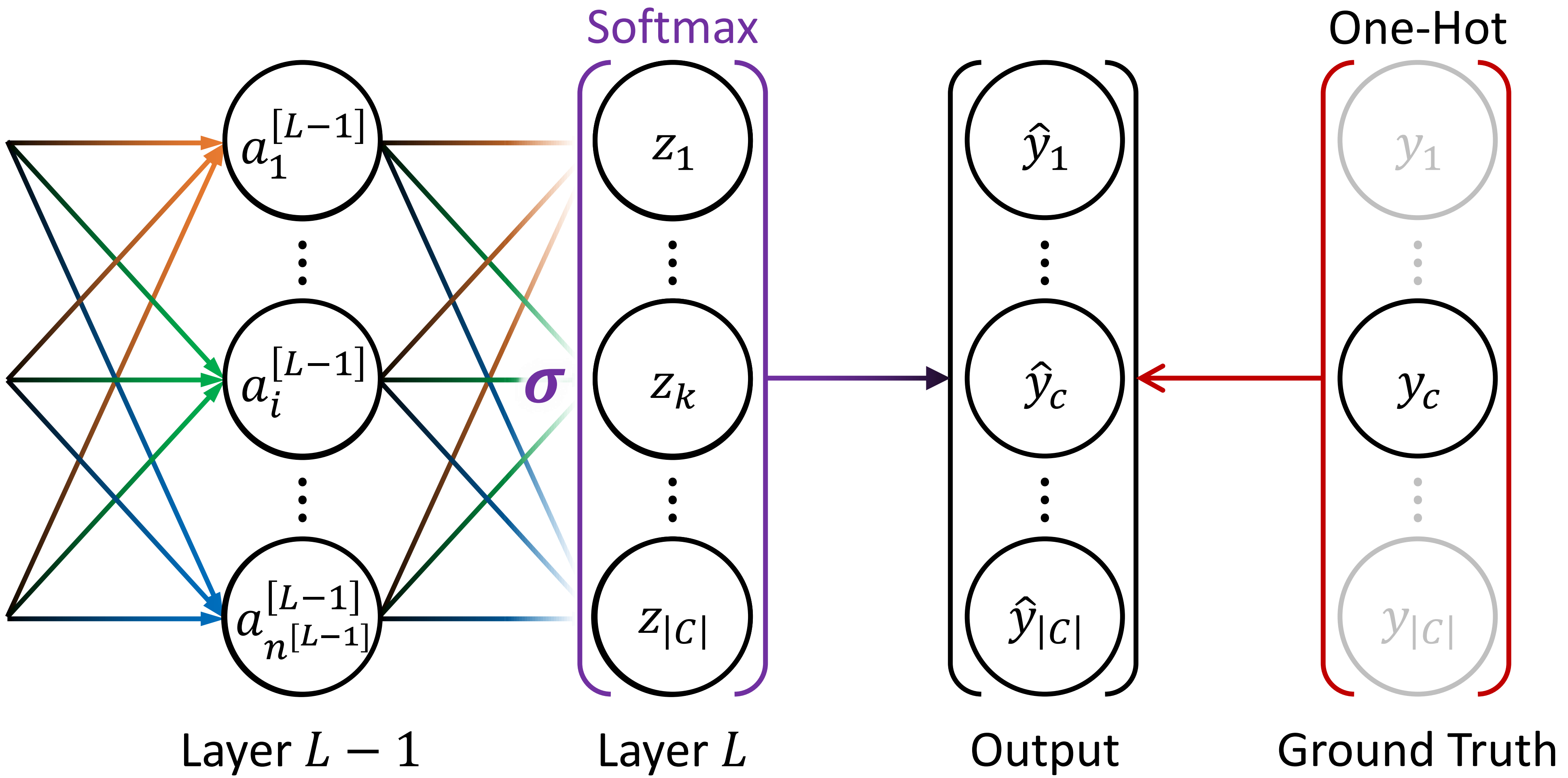
$$\sigma(\boldsymbol{z}) = \begin{pmatrix} e^{z_1}/(1 + e^{z_1}) \\ \vdots \\ e^{z_N}/(1 + e^{z_N}) \end{pmatrix}$$

Softmax

- *Which class is most probably true?*
- For **multiclass** classification
- $\boldsymbol{\sigma}(z_c) = \frac{e^{z_c}}{\sum_{c=1}^{|C|}(1+e^{z_c})} \boldsymbol{e}_c$

$$\boldsymbol{\sigma}(\boldsymbol{z}) = \frac{\begin{pmatrix} e^{z_1} \\ \vdots \\ e^{z_{|C|}} \end{pmatrix}}{\sum_{c=1}^{|C|}(1 + e^{z_c})} = \frac{e^{\boldsymbol{z}}}{(\boldsymbol{1} + e^{\boldsymbol{z}}) \cdot \boldsymbol{1}}$$

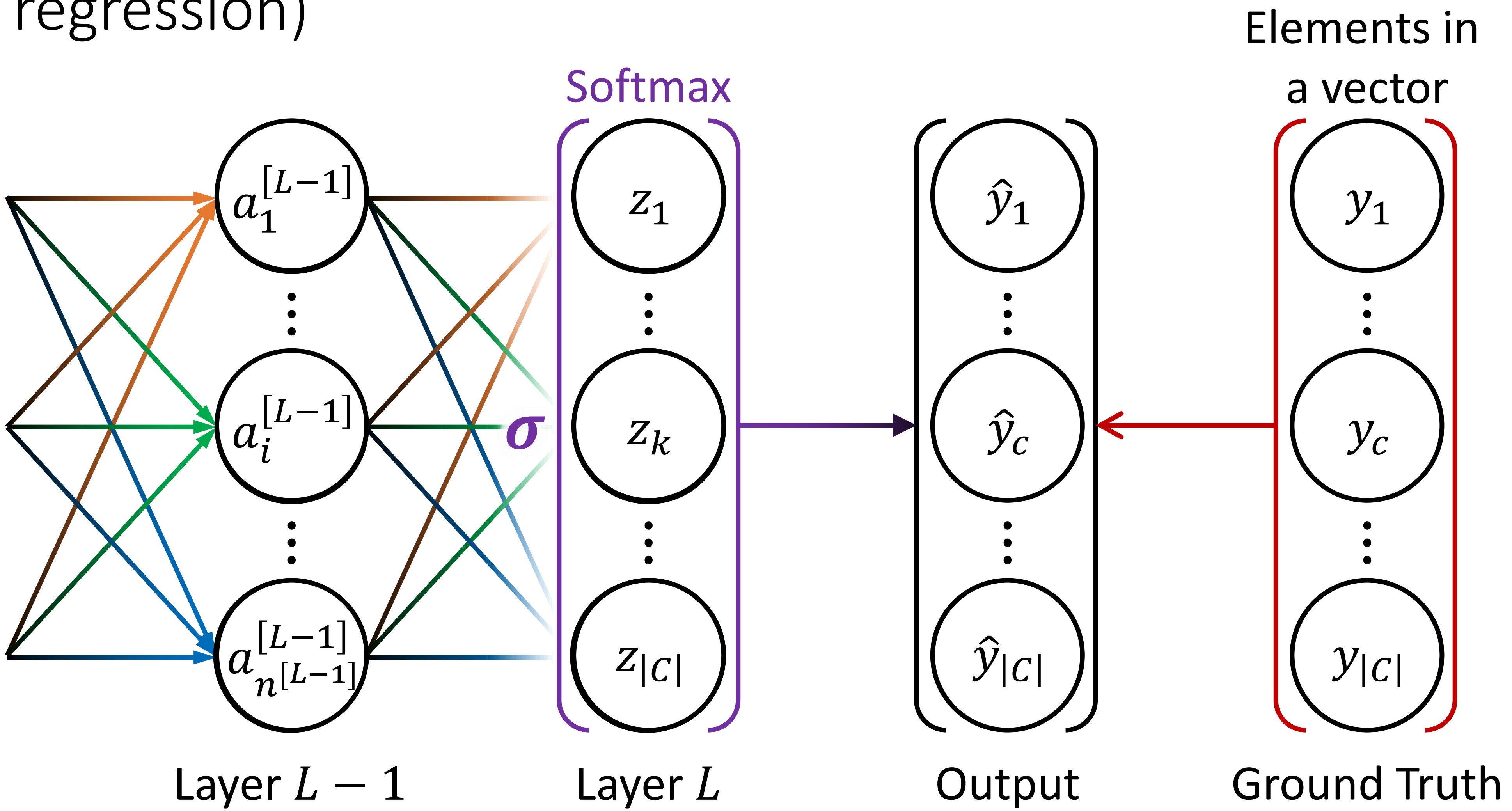
Multiple outputs Neural Network
(Multiclass classification)



ε = **Categorical Cross Entropy** $= -\boldsymbol{y} \cdot \log \widehat{\boldsymbol{y}}$

Multiple outputs Neural Network

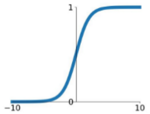
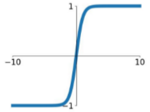
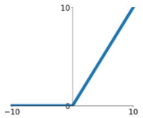
(Vector regression)



ε = **Euclidean distance** $= \|\boldsymbol{y} - \hat{\boldsymbol{y}}\|_2$

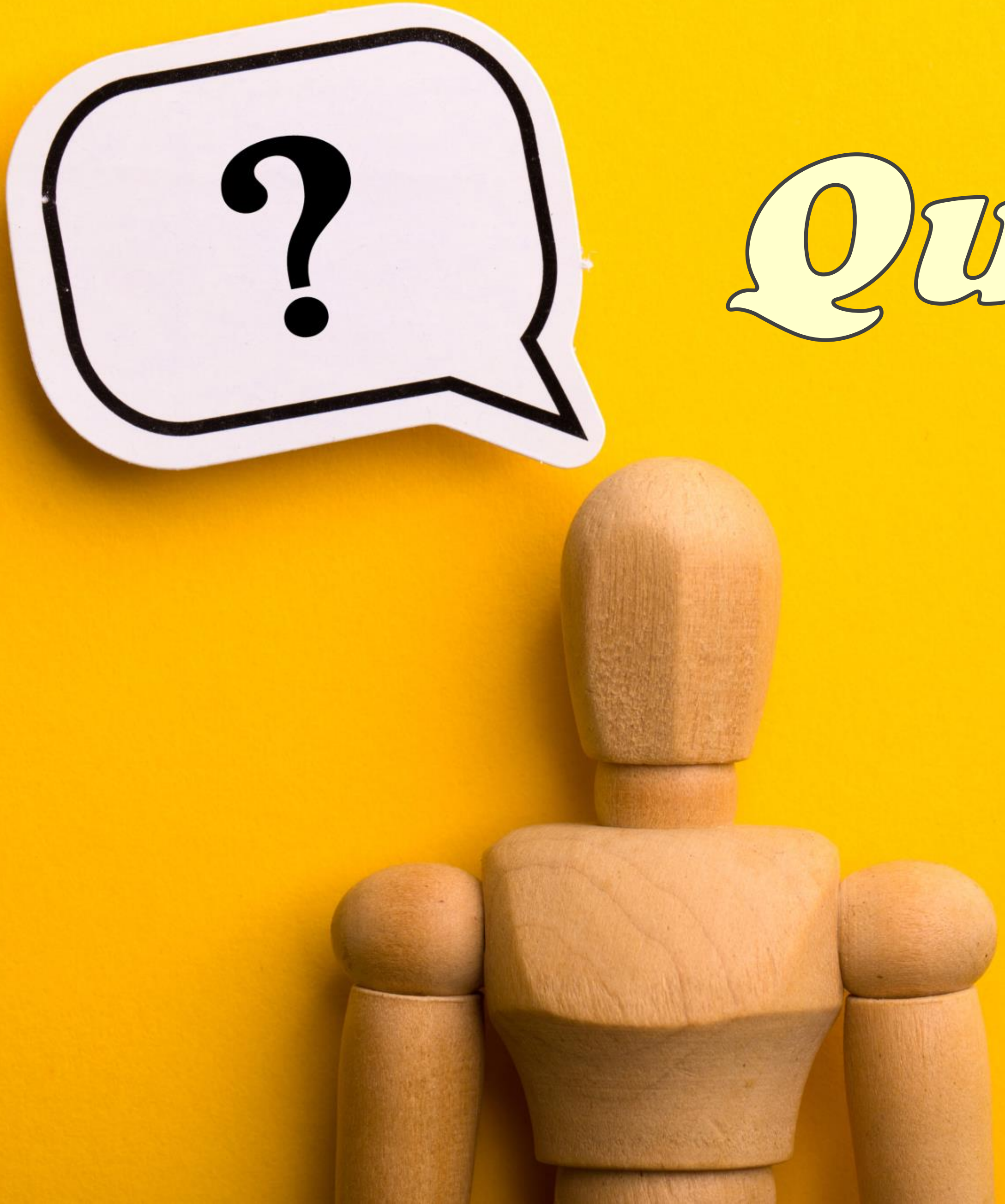
Derivatives for calculating gradient

Error $\varepsilon(\hat{y})$		$\frac{d\varepsilon}{d\hat{y}}$
Square Error	$\frac{1}{2}(y-\hat{y})^2$	$-(y-\hat{y})$
Binary Cross Entropy	$-y\log\hat{y}$	$-\frac{y}{\hat{y}}$
Error $\varepsilon(\boldsymbol{z})$		$\frac{d\varepsilon}{d\boldsymbol{z}}$
Categorical Cross Entropy	$-\boldsymbol{y}\cdot\log\hat{\boldsymbol{y}}$	$\hat{\boldsymbol{y}}-\boldsymbol{y}$

Activation $g(f)$			$\frac{dg}{df}$
Sigmoid		$\frac{1}{1+e^{-f}}$	$(1-g)g$
tanh		$\tanh f$	$1-g^2$
ReLU		$\max(0,f)$	$[f>0]$
Softmax		$\frac{e^{\boldsymbol{z}}}{(\boldsymbol{1}+e^{\boldsymbol{z}})\cdot\boldsymbol{1}}$	$\frac{d\boldsymbol{g}}{d\boldsymbol{z}}$

Weighted Sum $f(\boldsymbol{w})$	$\frac{df}{d\boldsymbol{w}}$
$\boldsymbol{w}^\top\boldsymbol{x}$	$\boldsymbol{x}$

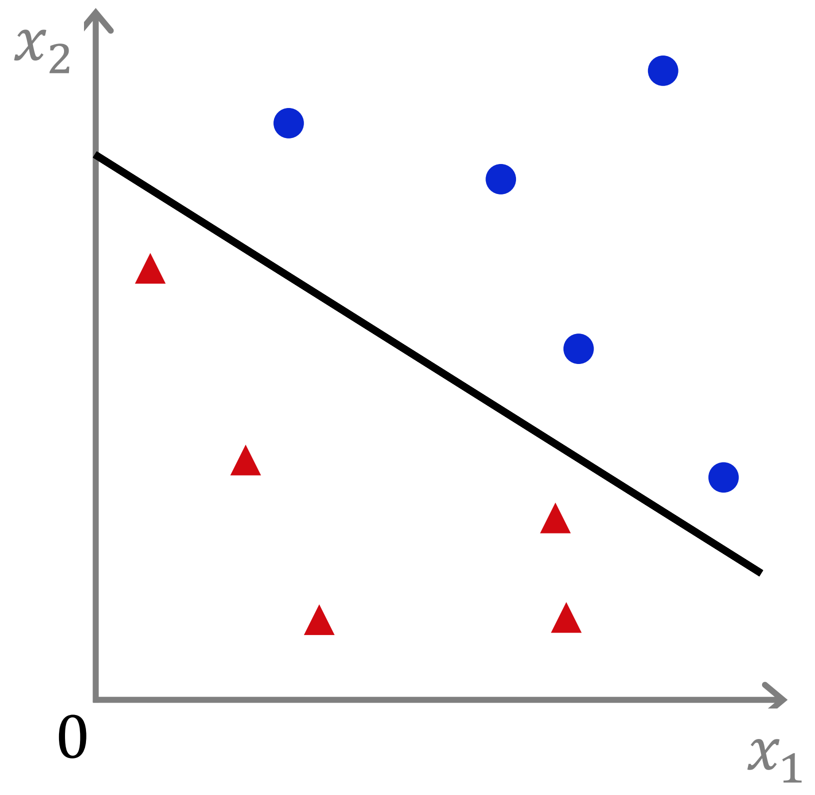
Advanced: further reading



Questions!

Perceptron → Neural Network

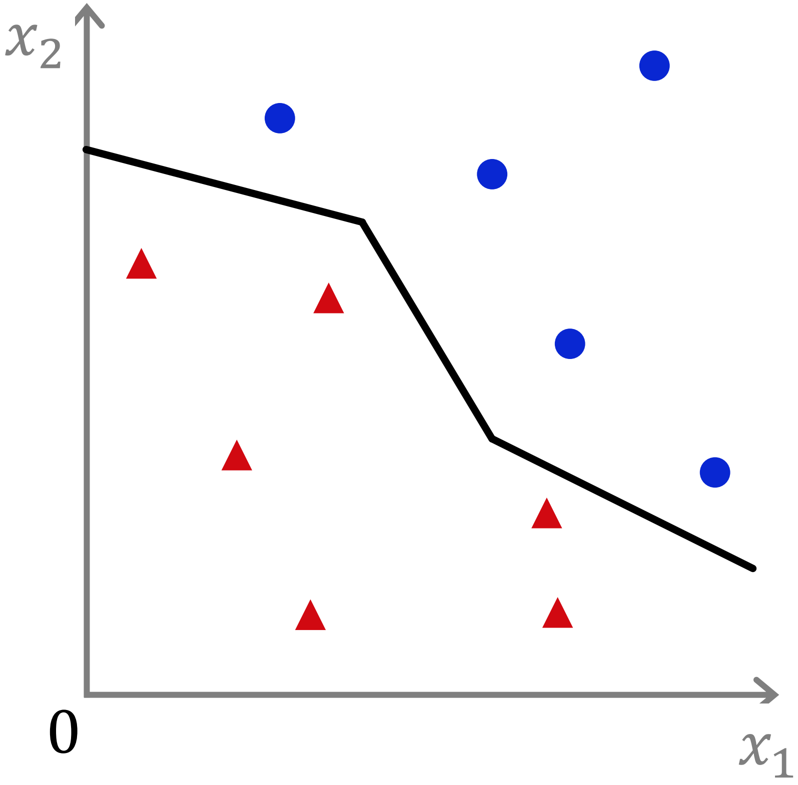
- Linear classifiers
- **Non-linear** classifiers
  - Other **activation functions**
    - Differentiable ones!
  - **Multiple** perceptrons / neurons
    - Multi-Layer Perceptron (MLP)



Can only model
straight lines

Perceptron → Neural Network

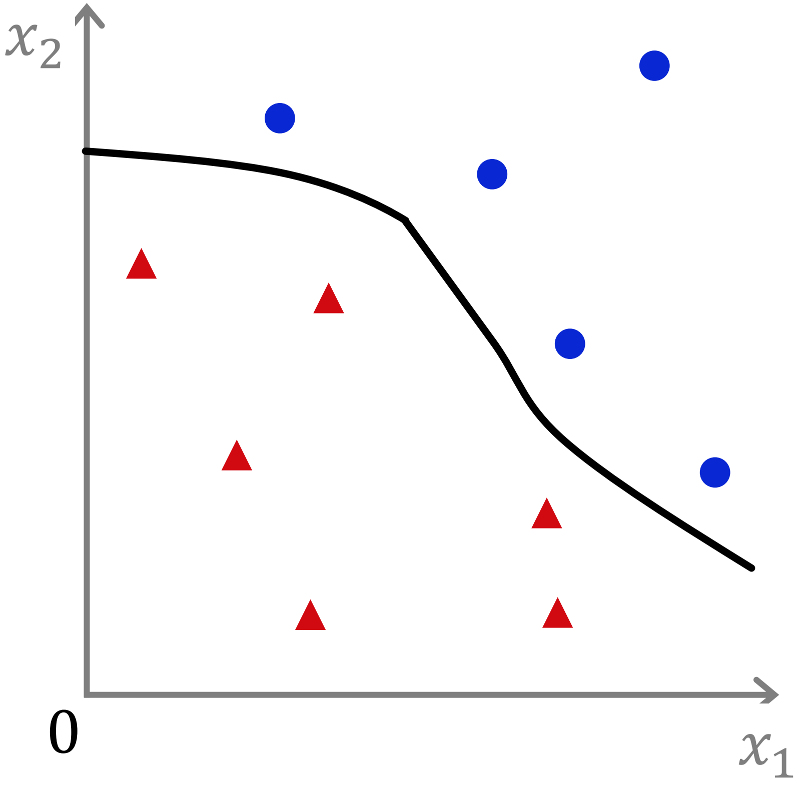
- Linear classifiers
- **Non-linear** classifiers
  - Other **activation functions**
    - Differentiable ones!
  - **Multiple** perceptrons / neurons
    - Multi-Layer Perceptron (MLP)



Can model **multiple** straight lines

Perceptron → Neural Network

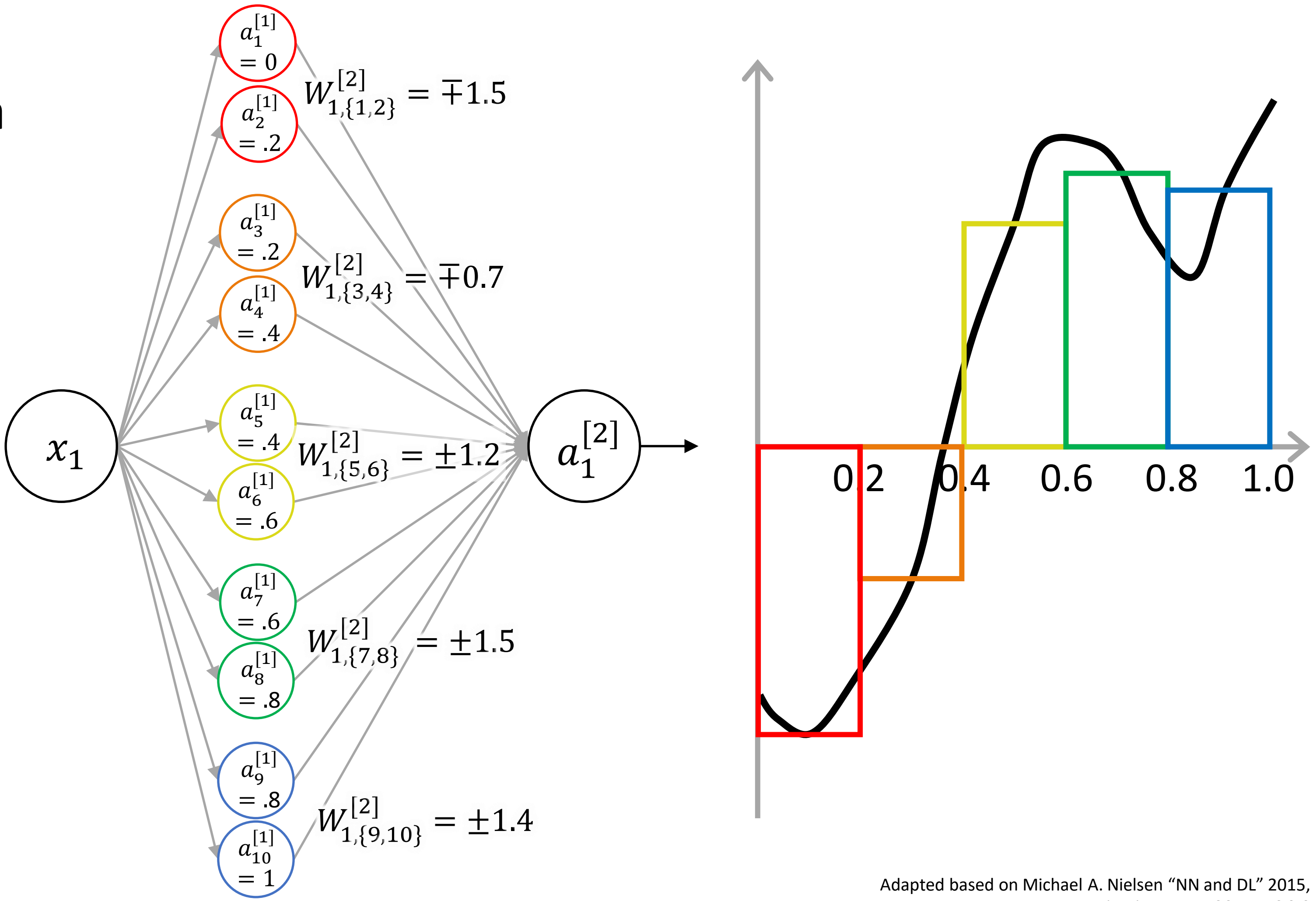
- Linear classifiers
- **Non-linear** classifiers
  - Other **activation functions**
    - Differentiable ones!
  - **Multiple** perceptrons / neurons
    - Multi-Layer Perceptron (MLP)



Can model **multiple *curved*** lines

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

Gradient Descent for Neural Networks

Backpropagation

Department of Computer Science
School of Computing

W09B Thursday: **Backpropagation**

Image credit: https://www.mensjournal.com/wp-content/uploads/mf/main-turn-your-training-around-with-backward-running.jpg

Wrapping Up

What did we learn?