STAT 453: Introduction to Deep Learning and Generative Models

Ben Lengerich

Lecture 05: Parameter Optimization and Gradient Descent

September 17, 2026

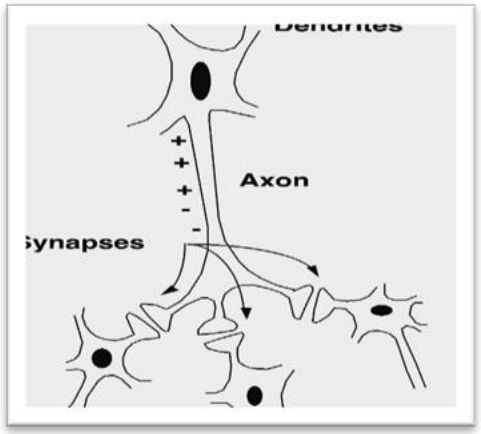
Announcements

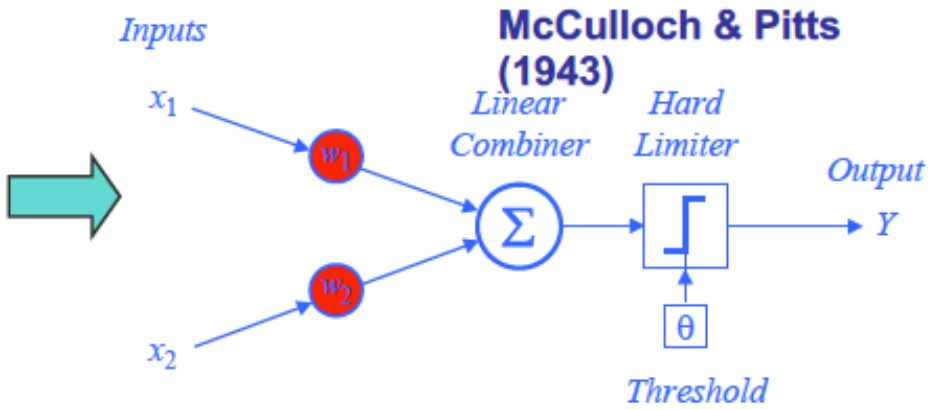
- HW1 Due this Friday via Canvas

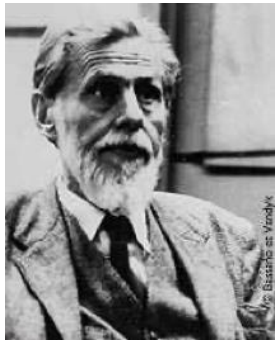
Today

- Perceptron
- Soft threshold → Likelihood
- Gradient descent
- Backpropagation & computation graphs

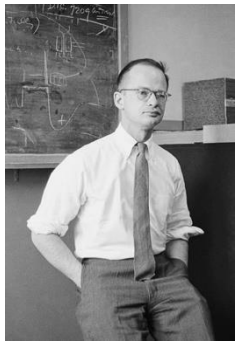
McCulloch & Pitt's neuron model (1943)







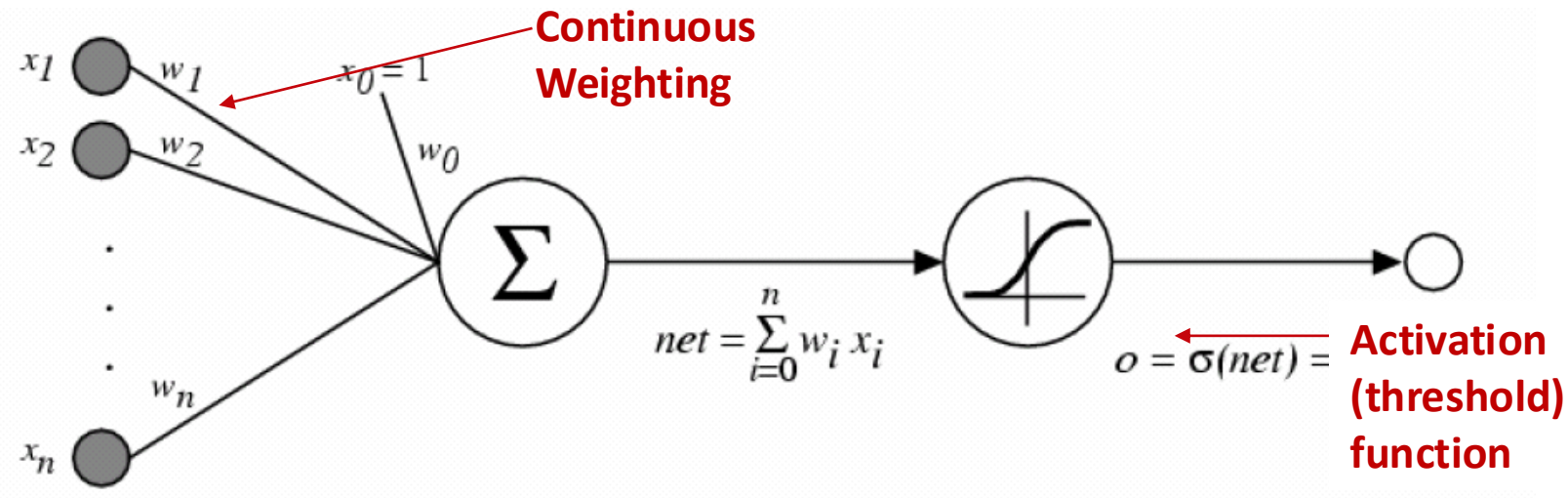
Warren McCulloch



Walter Pitts

Rosenblatt's Perceptron

Rosenblatt, F. (1957). *The perceptron, a perceiving and recognizing automaton. Project Para*. Cornell Aeronautical Laboratory.

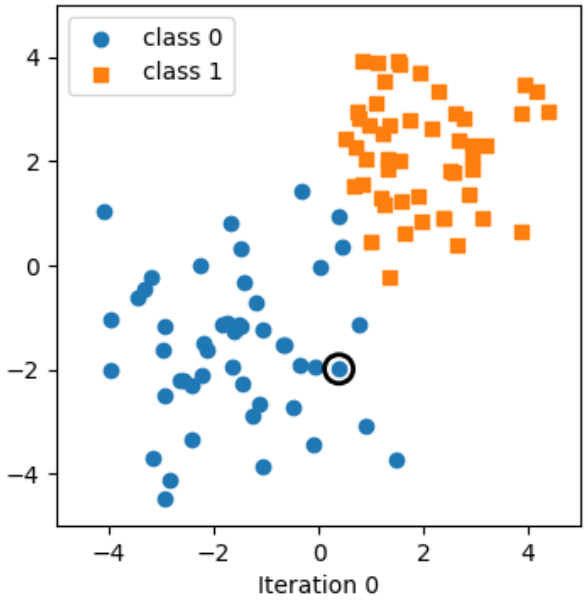


Generalizes MP neurons a bit…

Source: http://www.enzyklopaedie-der-wirtschaftsinformatik.de/wi-enzyklopaedie/Members/wilex4/Rosen-2.jpg

Perceptron Learning Algorithm

- Assume binary classification task
- Perceptron finds decision boundary is classes are separable



[animated GIF]

Code at https://github.com/rasbt/stat453-deep-learning-ss20/blob/master/L03-perceptron/code/perceptron-animation.ipynb

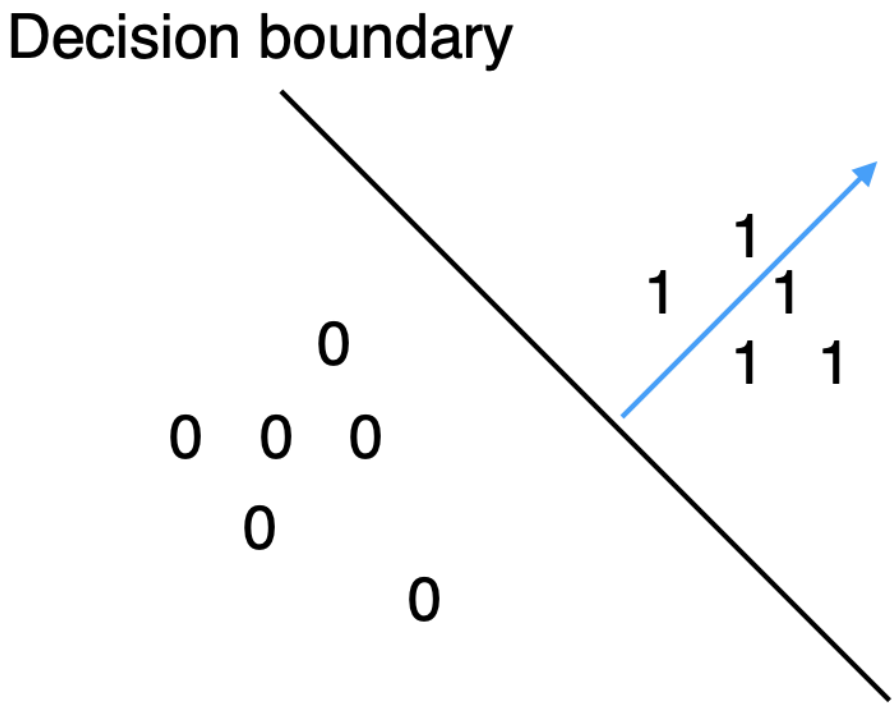
Perceptron Learning Algorithm (pseudocode)

Let

$$\mathcal{D} = (\langle \mathbf{x}^{[1]}, y^{[1]} \rangle, \langle \mathbf{x}^{[2]}, y^{[2]} \rangle, ..., \langle \mathbf{x}^{[n]}, y^{[n]} \rangle) \in (\mathbb{R}^m \times \{0, 1\})^n$$

1. Initialize $\boldsymbol{w} \coloneqq 0^m$ (assume weight incl. bias)
2. For every training epoch:
    1. For every $\langle \boldsymbol{x}^{[i]}, y^{[i]} \rangle \in D$:
        1. $\hat{y}^{[i]} \coloneqq \sigma(\boldsymbol{x}^{[i]T} \boldsymbol{w})$ ← Only -0 or 1
        2. $err \coloneqq (y^{[i]} - \hat{y}^{[i]})$ ← Only -1, 0, or 1
        3. $\boldsymbol{w} \coloneqq \boldsymbol{w} + err \times \boldsymbol{x}^{[i]}$

Perceptron Geometric Intuition



Weight vector is perpendicular to the boundary. Why?

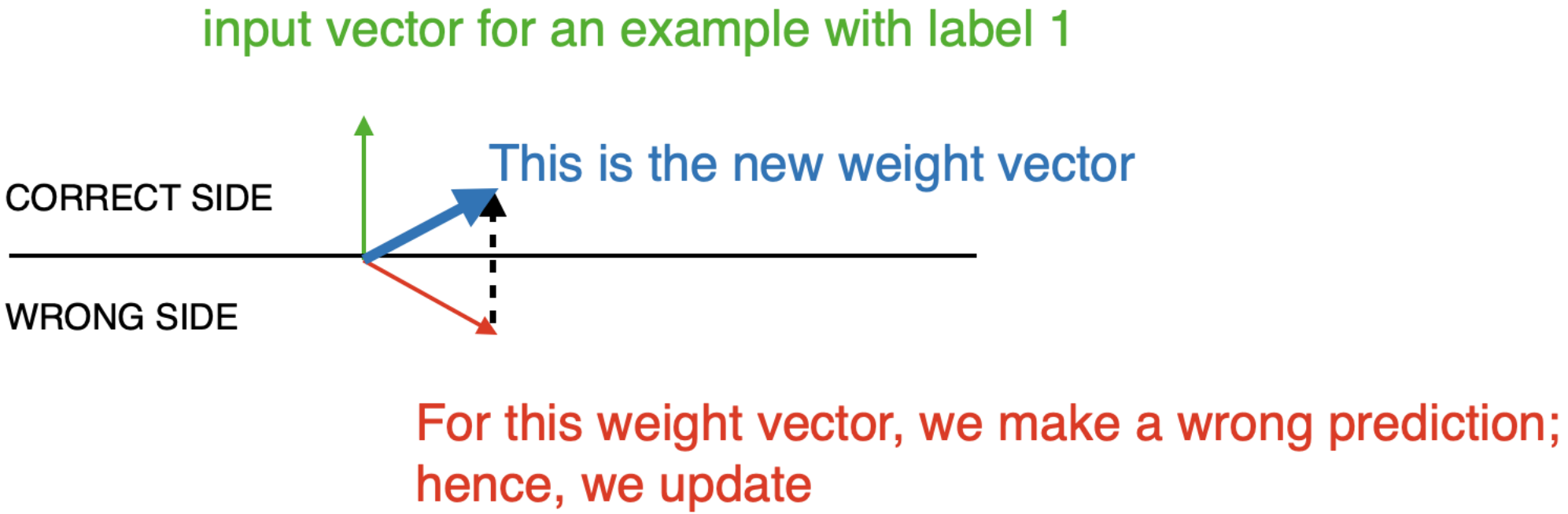
Remember,

$$\hat{y} = \begin{cases} 0, & \mathbf{w}^T\mathbf{x} \leq 0 \\ 1, & \mathbf{w}^T\mathbf{x} > 0 \end{cases}$$

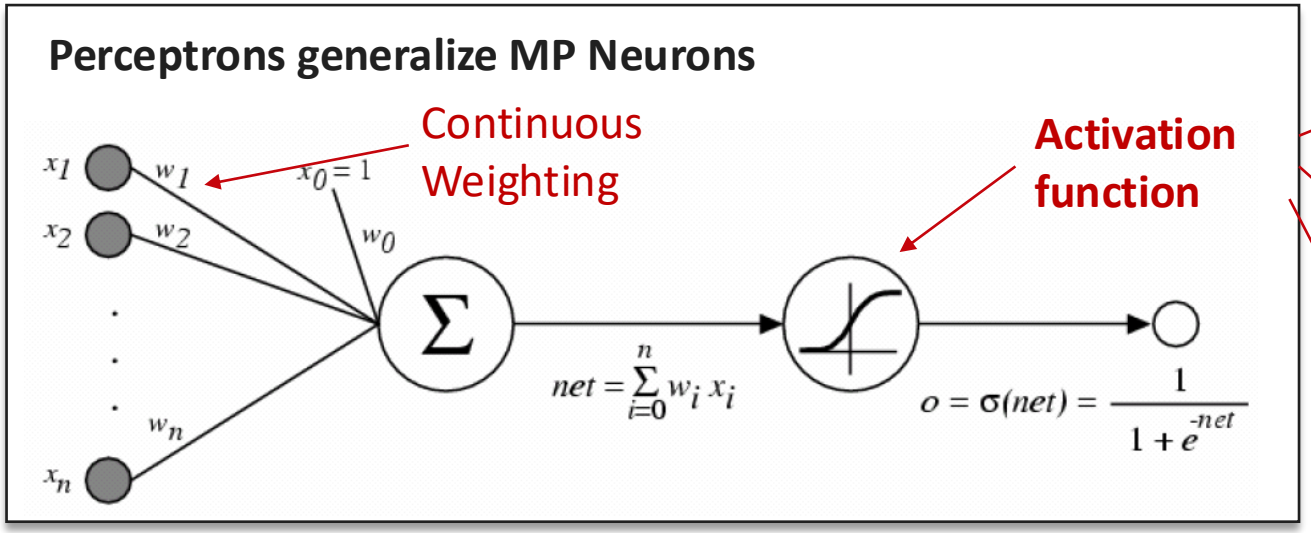
$$\mathbf{w}^T\mathbf{x} = ||\mathbf{w}|| \cdot ||\mathbf{x}|| \cdot \underbrace{\cos(\theta)}$$

So this needs to be 0 at the boundary, and it is zero at 90°

Perceptron Geometric Intuition: Learning



Beyond Rosenblatt's Perceptron

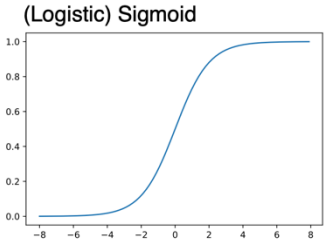


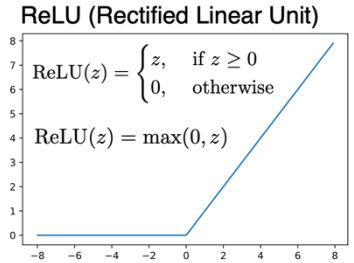
threshold function → Classic Rosenblatt Perceptron

sigmoid → DL "Perceptron" / sigmoid unit

identity → Linear Regression

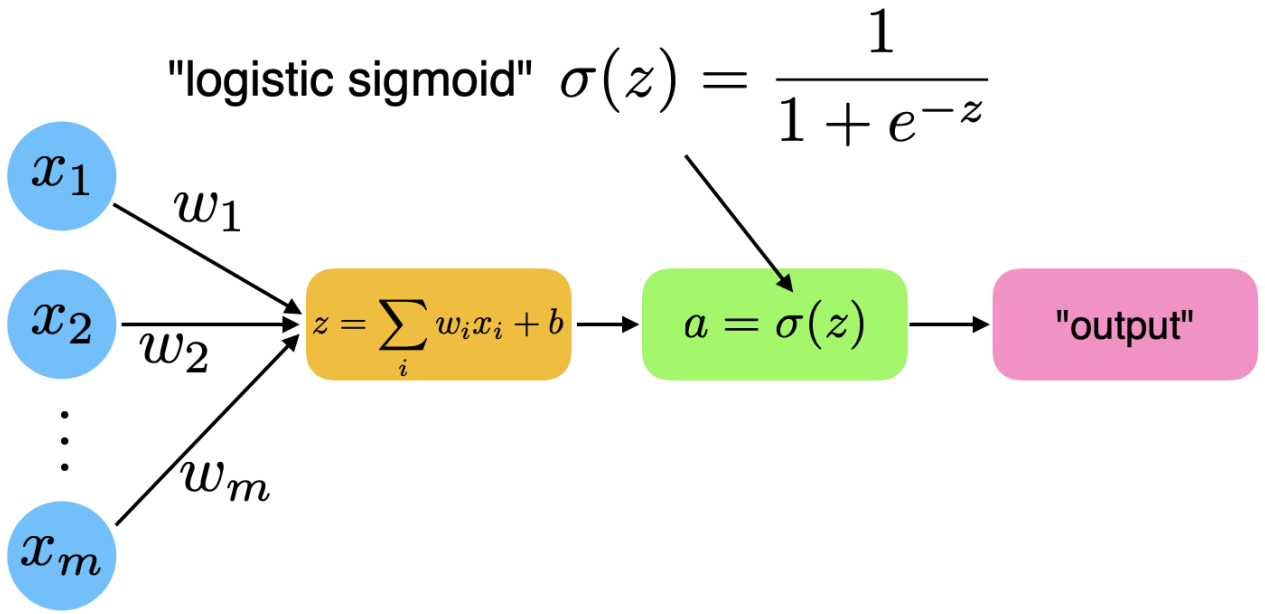
- Many activation functions:
  - Threshold function (perceptron, 1950+)
  - Sigmoid function (before 2000)
  - ReLU function (popular since CNNs)
  - Many variants of ReLU, e.g. leaky ReLU, GeLU





Sigmoid unit: Logistic regression gives an optimizer

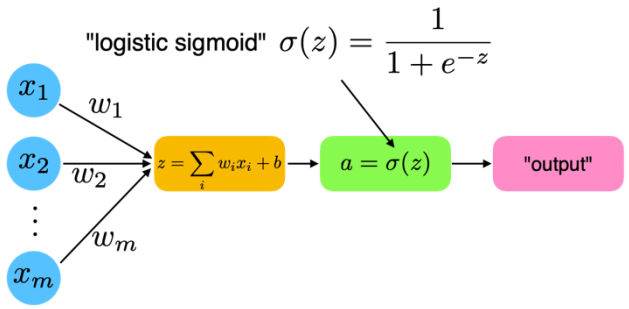
- For binary classes $y \in \{0, 1\}$



Sigmoid unit: Logistic regression gives an optimizer

- Given the output:

$$h(\mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x} + b)$$



- We compute the probability as

$$P(y|\mathbf{x}) = \begin{cases} h(\mathbf{x}) & \text{if } y = 1 \\ 1 - h(\mathbf{x}) & \text{if } y = 0 \end{cases}$$

$$P(y|\mathbf{x}) = a^y (1-a)^{(1-y)}$$

Recall Bernoulli distribution…

Sigmoid unit: Logistic regression gives an optimizer

- Given the probability:

$$P(y|\mathbf{x}) = a^y(1-a)^{(1-y)}$$

- Under MLE estimation, we would like to maximize the multi-sample likelihood:

$$P\left(y^{[i]},\ldots,y^{[n]}|\mathbf{x}^{[1]},\ldots,\mathbf{x}^{[n]}\right) = \prod_{i=1}^{n} P\left(y^{[i]}|\mathbf{x}^{[i]}\right)$$

$$= \underbrace{\prod_{i=1}^{n}\left(\sigma\left(z^{(i)}\right)\right)^{y^{(i)}}\left(1-\sigma\left(z^{(i)}\right)\right)^{1-y^{(i)}}}_{\text{Likelihood}}$$

Sigmoid unit: Logistic regression gives an optimizer

$$P\left(y^{[i]}, \ldots, y^{[n]} | \mathbf{x}^{[1]}, \ldots, \mathbf{x}^{[n]}\right) = \underbrace{\prod_{i=1}^{n} \left(\sigma\left(z^{(i)}\right)\right)^{y^{(i)}} \left(1 - \sigma\left(z^{(i)}\right)\right)^{1-y^{(i)}}}_{\text{Likelihood}}$$

- We are going to optimize via gradient descent, so let's apply the logarithm to separate components:

$$l(\mathbf{w}) = \log L(\mathbf{w})$$

$$= \underbrace{\sum_{i=1}^{n} \left[y^{(i)} \log\left(\sigma\left(z^{(i)}\right)\right) + \left(1 - y^{(i)}\right) \log\left(1 - \sigma\left(z^{(i)}\right)\right)\right]}_{\text{Log-Likelihood}}$$

Sigmoid unit: Logistic regression gives an optimizer

$$\frac{\partial \mathcal{L}}{\partial w_j} = \frac{\partial \mathcal{L}}{\partial a}\frac{da}{dz}\frac{\partial z}{\partial w_j}$$

$$\frac{\partial \mathcal{L}}{\partial a} = \frac{a-y}{a-a^2}$$

$$\frac{da}{dz} = \frac{e^{-z}}{(1+e^{-z})^2} = a \cdot (1-a)$$

$$\frac{\partial z}{w_j} = x_j$$

$$\frac{\partial \mathcal{L}}{\partial z} = a - y$$

$$\frac{\partial \mathcal{L}}{\partial w_j} = (a-y)x_j$$

Logistic Regression: Gradient Descent learning Rule

Stochastic gradient descent:

1. Initialize $\mathbf{w} := \mathbf{0} \in \mathbb{R}^m$, $\mathbf{b} := 0$
2. For every training epoch:
   A. For every $\langle \mathbf{x}^{[i]}, y^{[i]} \rangle \in \mathcal{D}$
      (a) $\hat{y}^{[i]} := \sigma\left(\mathbf{x}^{[i]T}\mathbf{w} + b\right)$
      (b) $\nabla_{\mathbf{w}}\mathcal{L} = -\left(y^{[i]} - \hat{y}^{[i]}\right)\mathbf{x}^{[i]}$
          $\nabla_b\mathcal{L} = -\left(y^{[i]} - \hat{y}^{[i]}\right)$
      (c) $\mathbf{w} := \mathbf{w} + \eta \times \left(-\nabla_{\mathbf{w}}\mathcal{L}\right)$
          $b := b + \eta \times \underbrace{\left(-\nabla_b\mathcal{L}\right)}$

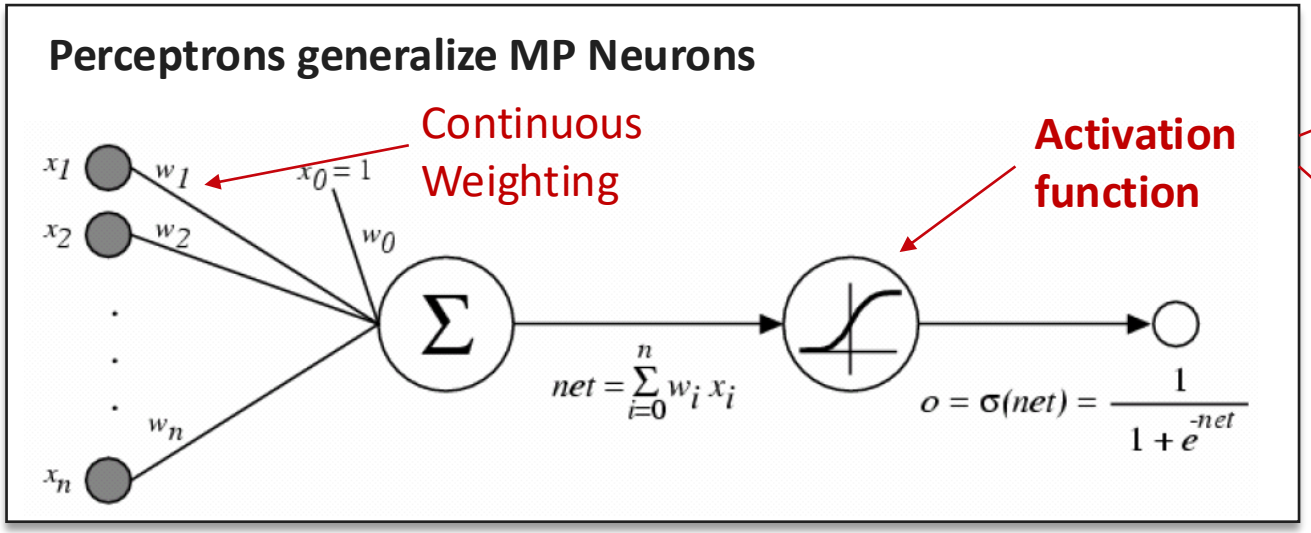
learning rate

negative gradient

Note

$a - y \Leftrightarrow -\left(y^{[i]} - \hat{y}^{[i]}\right)$

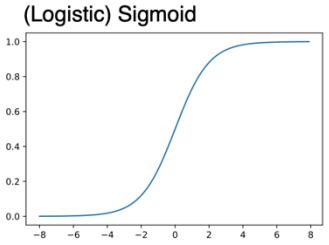
Beyond Rosenblatt's Perceptron

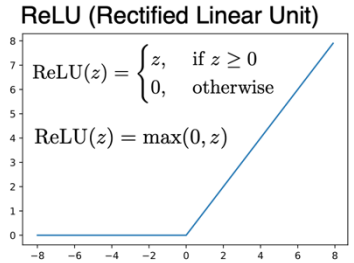


threshold function → Classic Rosenblatt Perceptron

sigmoid → DL "Perceptron" / sigmoid unit → Gradient Descent

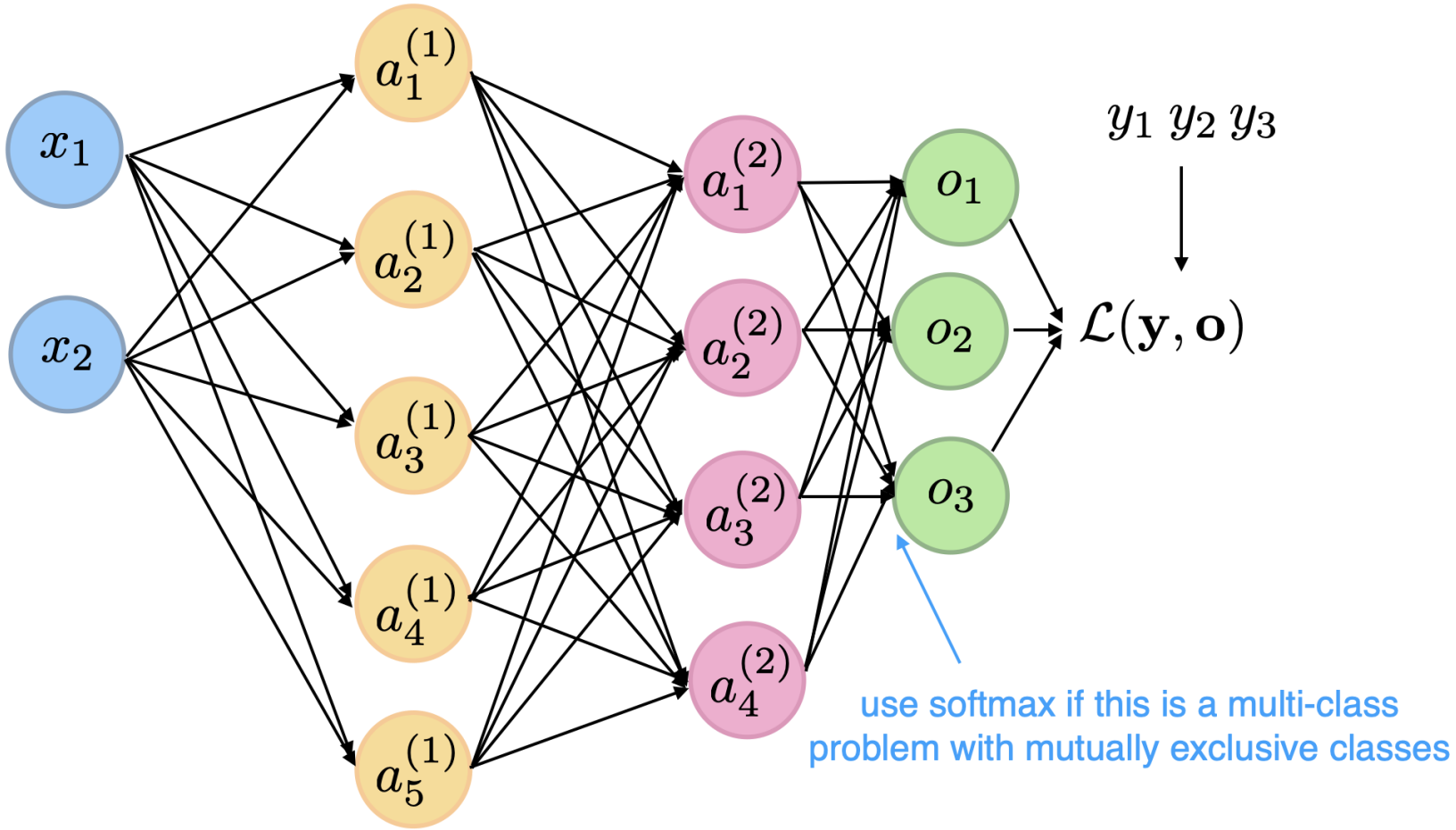
- Many activation functions:
  - Threshold function (perceptron, 1950+)
  - Sigmoid function (before 2000)
  - ReLU function (popular since CNNs)
  - Many variants of ReLU, e.g. leaky ReLU, GeLU





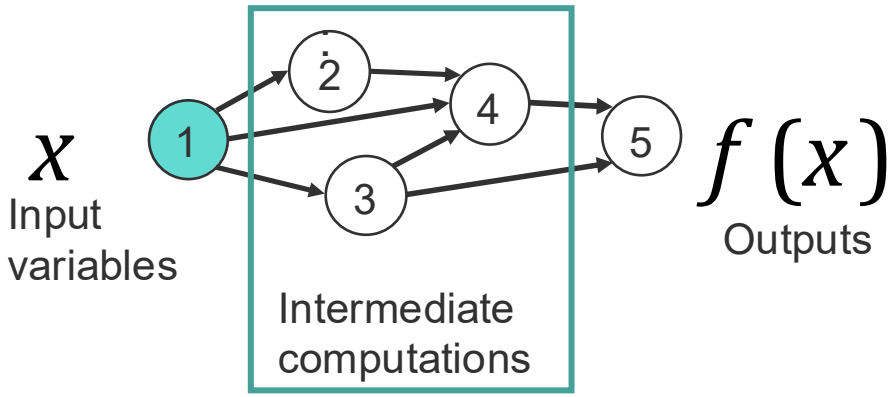
Multilayer Perceptron

- Computation Graph with Multiple Fully-Connected Layers



Backpropagation

- Neural networks are function compositions that can be represented as computation graphs:



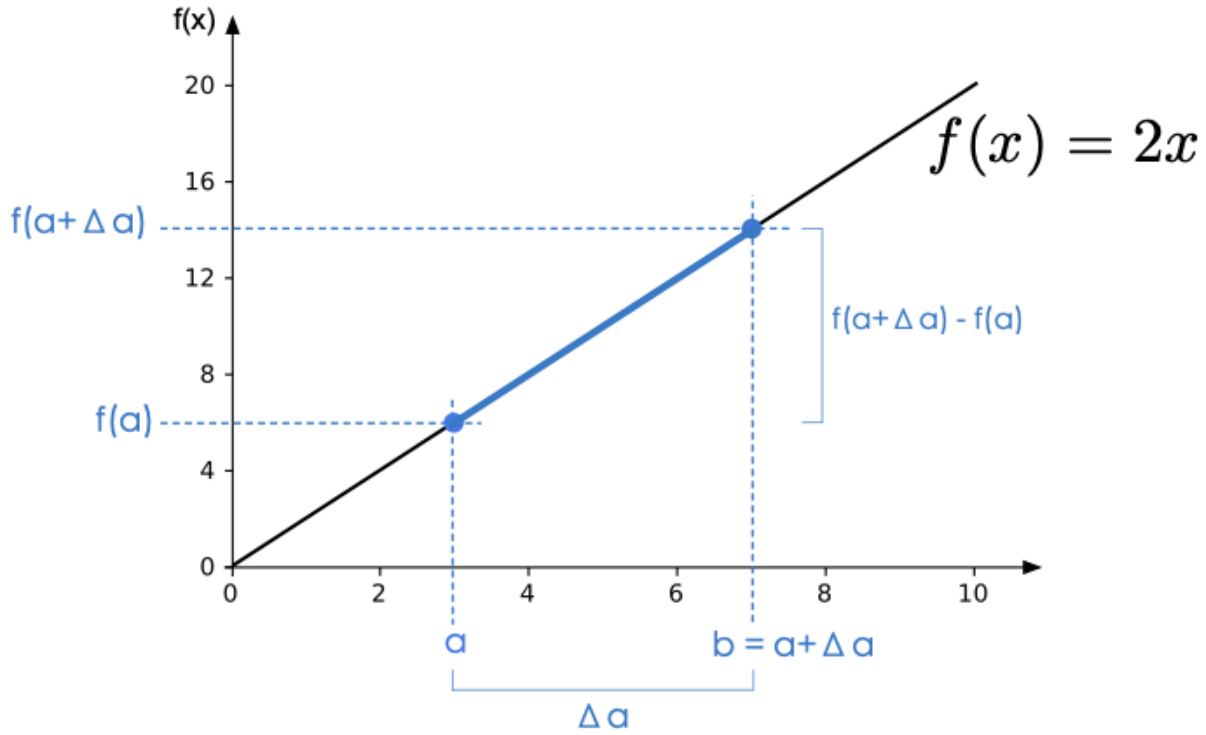
$$\frac{\partial f_n}{\partial x} =$$

- By applying the chain rule, and working in reverse order, we get:

$$\frac{\partial f_n}{\partial x} = \sum_{i_1 \in \pi(n)} \frac{\partial f_n}{\partial f_{i_1}} \frac{\partial f_{i_1}}{\partial x} = \sum_{i_1 \in \pi(n)} \frac{\partial f_n}{\partial f_{i_1}} \sum_{i_2 \in \pi(i_1)} \frac{\partial f_{i_1}}{\partial f_{i_2}} \frac{\partial f_{i_1}}{\partial x} = \dots$$

Quick Calculus Refresher

Derivative of a function = "rate of change" = "slope"



$$\text{Slope} = \frac{f(a+\Delta a) - f(a)}{a + \Delta a - a} = \frac{f(a+\Delta a) - f(a)}{\Delta a}$$

Quick Calculus Refresher: Function Derivative

$$f'(x) = \frac{df}{dx} = \lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

Example 1: $f(x) = 2x$

$$\begin{aligned}
\frac{df}{dx} &= \lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x)}{\Delta x} \\
&= \lim_{\Delta x \to 0} \frac{2x + 2\Delta x - 2x}{\Delta x} \\
&= \lim_{\Delta x \to 0} \frac{2\Delta x}{\Delta x} \\
&= \lim_{\Delta x \to 0} 2.
\end{aligned}$$

Quick Calculus Refresher: Function Derivative

$$f'(x) = \frac{df}{dx} = \lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

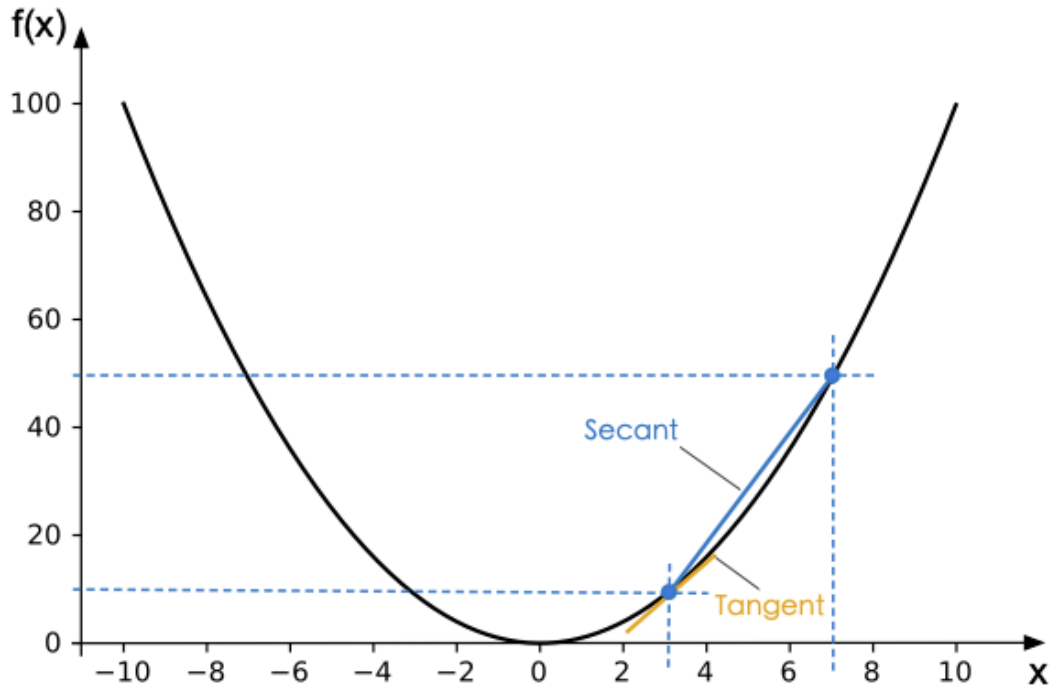
Example 2: $f(x) = x^2$

$$\begin{aligned}
\frac{df}{dx} &= \lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x)}{\Delta x} \\
&= \lim_{\Delta x \to 0} \frac{x^2 + 2x\Delta x + (\Delta x)^2 - x^2}{\Delta x} \\
&= \lim_{\Delta x \to 0} \frac{2x\Delta x + (\Delta x)^2}{\Delta x} \\
&= \lim_{\Delta x \to 0} 2x + \Delta x.
\end{aligned}$$

Quick Calculus Refresher: Geometric Intuition

Conceptually, we obtained the derivative $\frac{d}{dx}x^2 = 2x$

By approximating the slope (tangent) by a secant between two points (as before)



Quick Calculus Refresher: Cheatsheet

	Function $f(x)$	Derivative with respect to x
1	a	0
2	x	1
3	ax	a
4	x^2	$2x$
5	x^a	ax^{a-1}
6	a^x	$\log(a)a^x$
7	$\log(x)$	$1/x$
8	$\log_a(x)$	$1/(x\log(a))$
9	$\sin(x)$	$\cos(x)$
10	$\cos(x)$	$-\sin(x)$
11	$\tan(x)$	$\sec^2(x)$

Quick Calculus Refresher: Cheatsheet (2)

	Function	Derivative
Sum Rule	$f(x) + g(x)$	$f'(x) + g'(x)$
Difference Rule	$f(x) - g(x)$	$f'(x) - g'(x)$
Product Rule	$f(x)g(x)$	$f'(x)g(x) + f(x)g'(x)$
Quotient Rule	$f(x)/g(x)$	$[g(x)f'(x) - f(x)g'(x)]/[g(x)]^2$
Reciprocal Rule	$1/f(x)$	$-[f'(x)]/[f(x)]^2$
Chain Rule	$f(g(x))$	$f'(g(x))g'(x)$

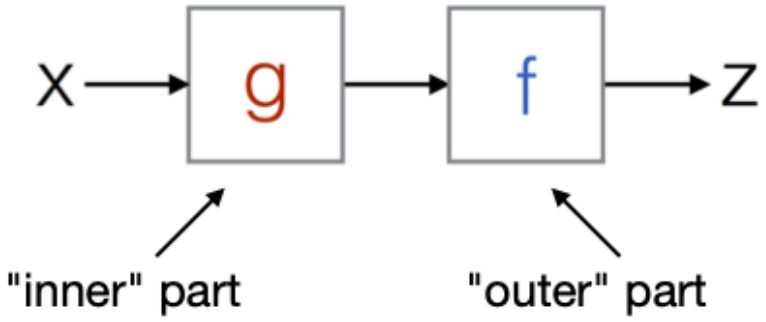
Chain Rule

- The chain rule is basically the essence of training (deep) neural networks
- If you understand and learn how to apply the chain rule to various function decompositions, deep learning will be super easy and even seem trivial to you from now on
- In fact, neural networks will become even easier to understand than any algorithm you learned about in my previous ML class

Chain Rule & Computation Graphs

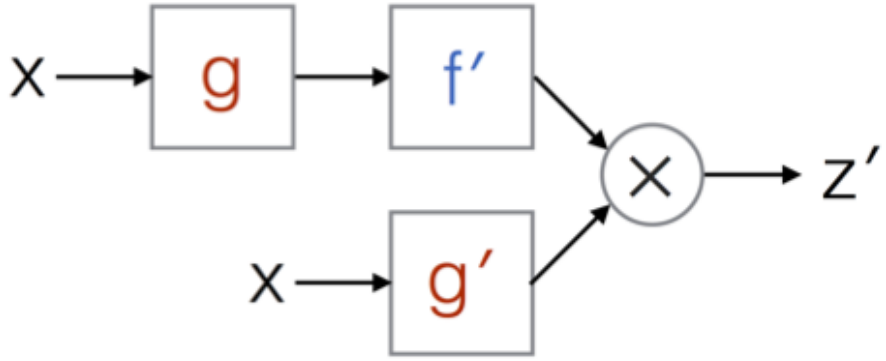
$$F(x) = f(\,g(x)\,) = z$$

Decomposition of some (nested) function:



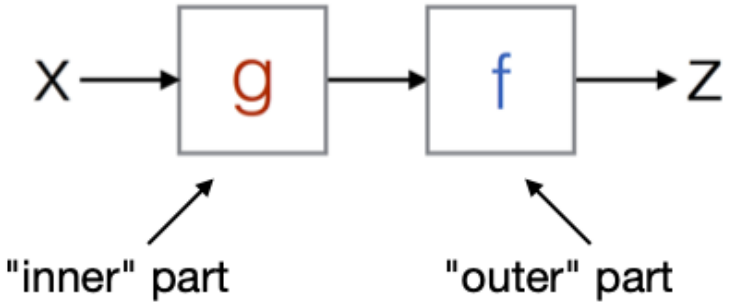
$$F'(x) = f'(\,g(x)\,)\,g'(x) = z'$$

Derivative of that nested function:



Chain Rule & Computation Graphs

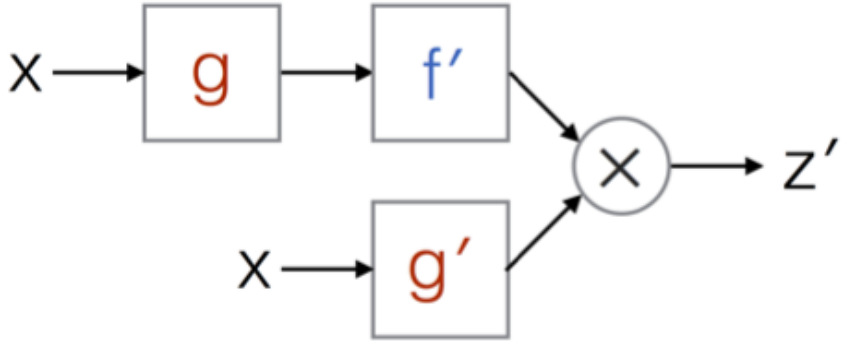
$$F(x) = f(\,g(x)\,) = z$$



Later, we will see that PyTorch can do that automatically for us :) (PyTorch literally keeps a computation graph in the background)

$$F'(x) = f'(\,g(x)\,)\,g'(x) = z'$$

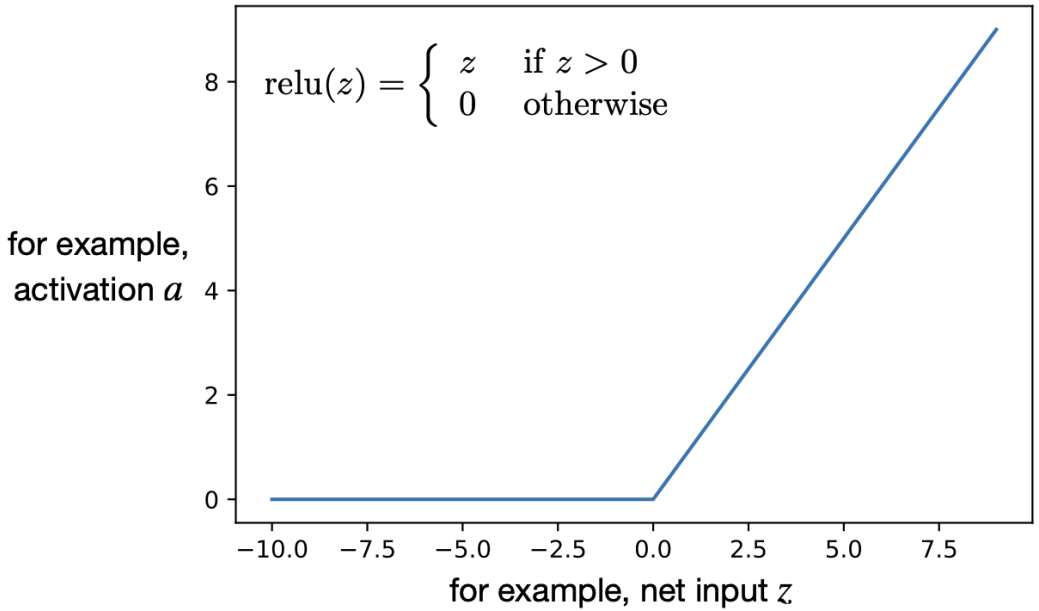
Also, PyTorch can compute the derivatives of most (differentiable) functions automatically



Computation graphs: ReLU

Suppose we have the following activation function:

$$a(x, w, b) = relu(w \cdot x + b)$$

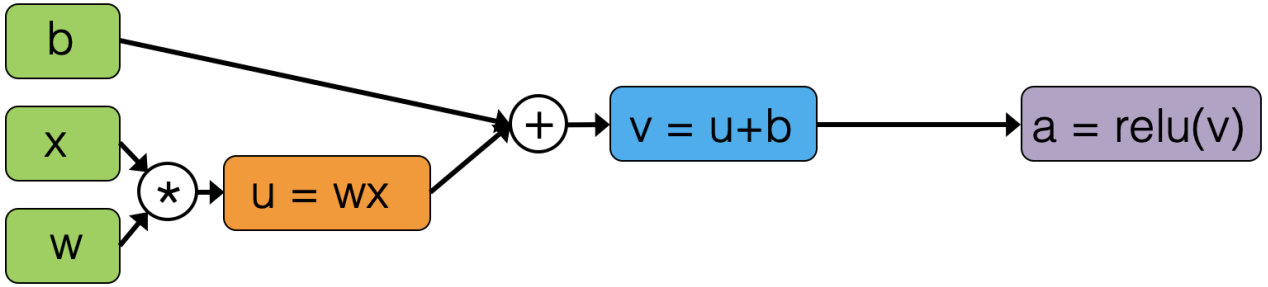


ReLU = Rectified Linear Unit

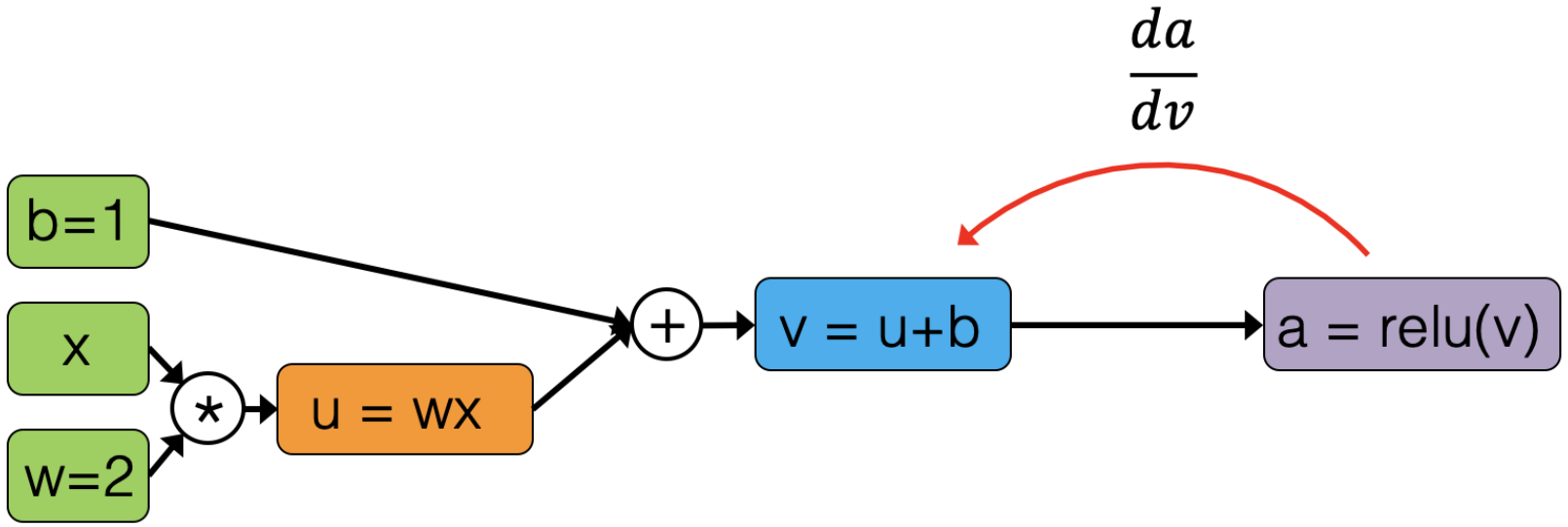
(prob. the most commonly used activation function in DL)

Computation graphs: ReLU

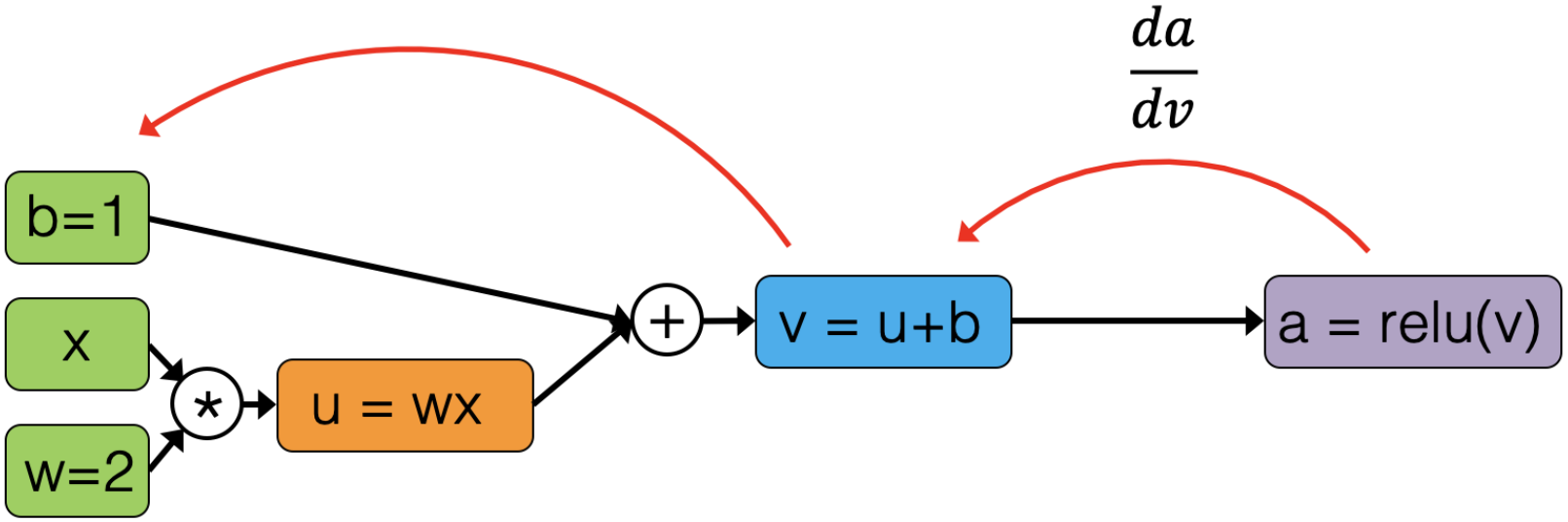
$$a(x, w, b) = relu(\underbrace{\underbrace{w \cdot x}_{u} + b}_{v})$$



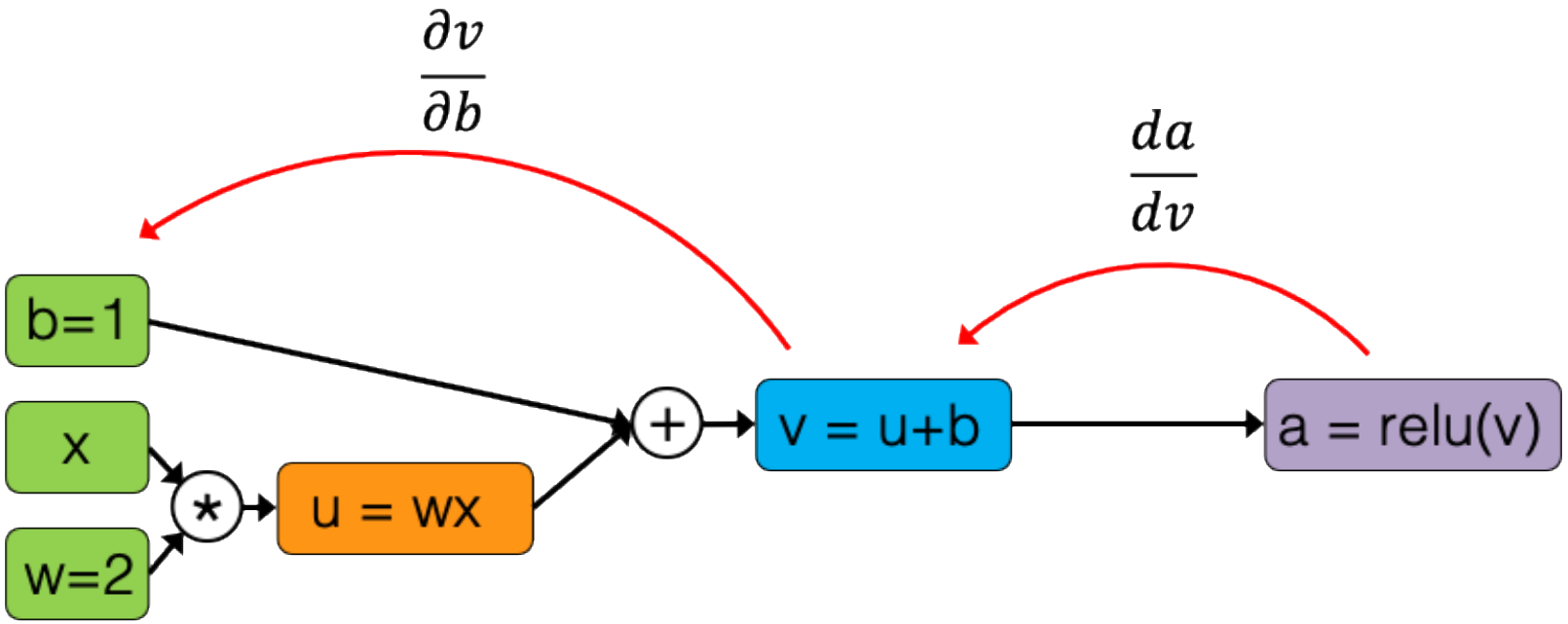
Computation graphs: ReLU



Computation graphs: ReLU

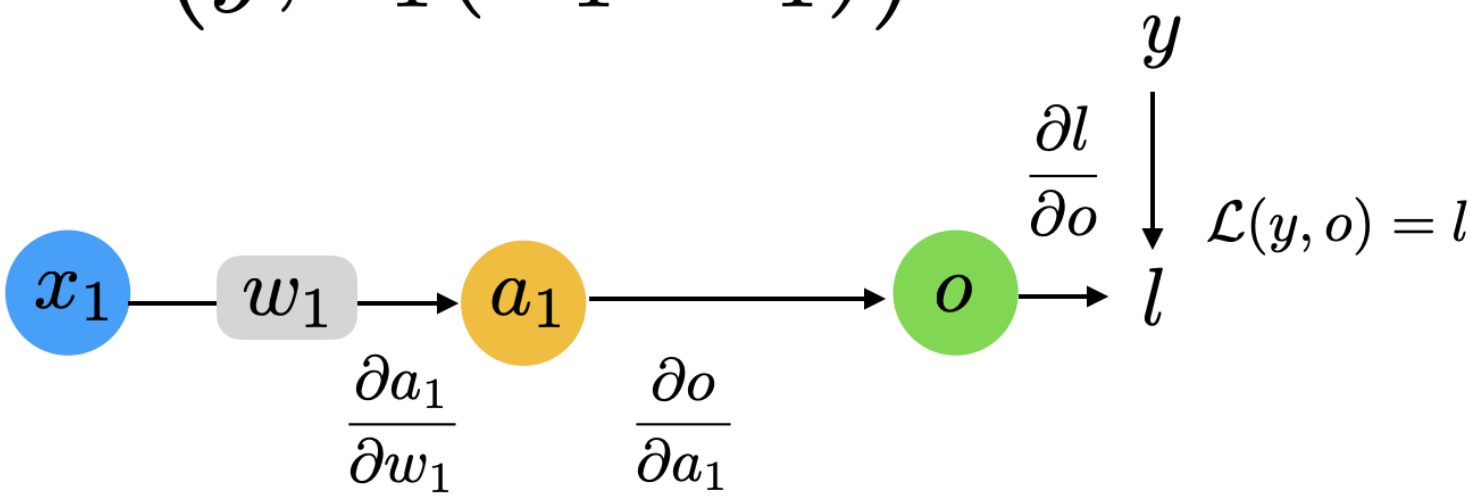


Computation graphs: ReLU



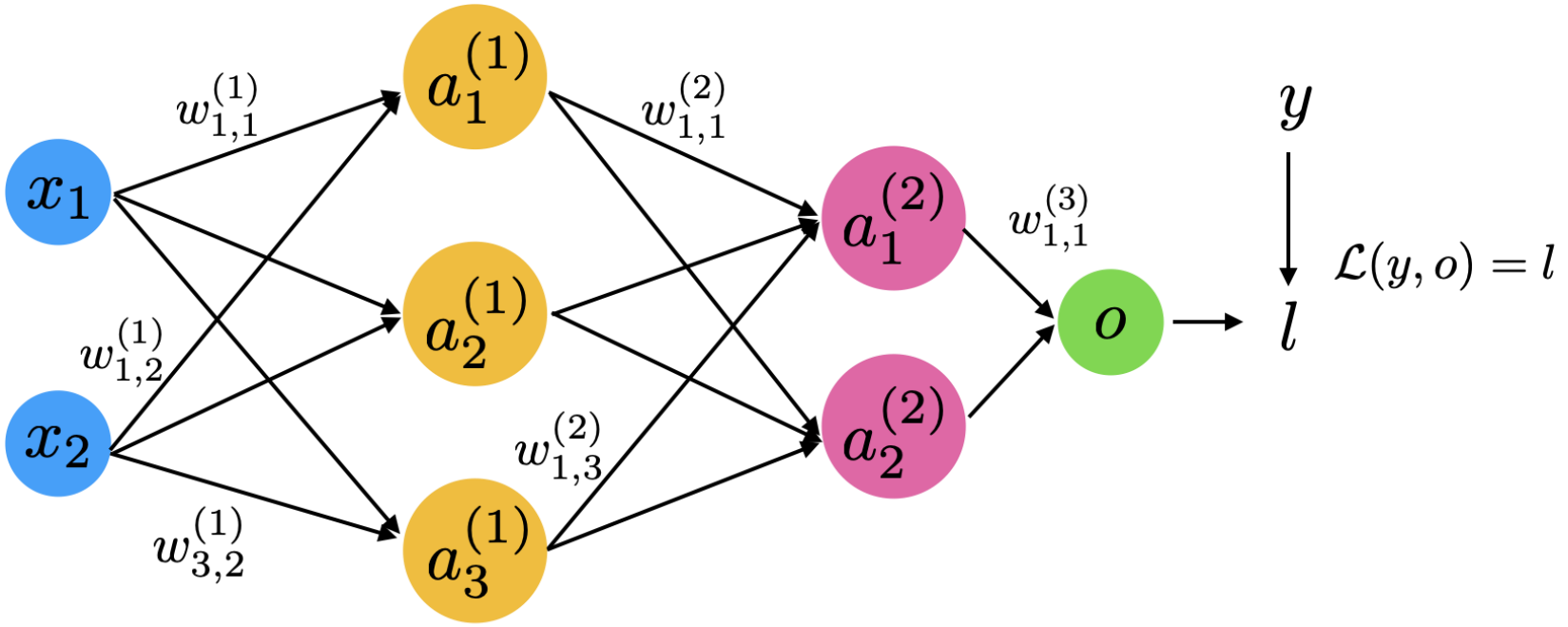
Computation graphs: Single-path

$$\mathcal{L}\big(y, \sigma_1(w_1 \cdot x_1)\big)$$



$$\frac{\partial l}{\partial w_1} = \frac{\partial l}{\partial o} \cdot \frac{\partial o}{\partial a_1} \cdot \frac{\partial a_1}{\partial w_1}$$ (univariate chain rule)

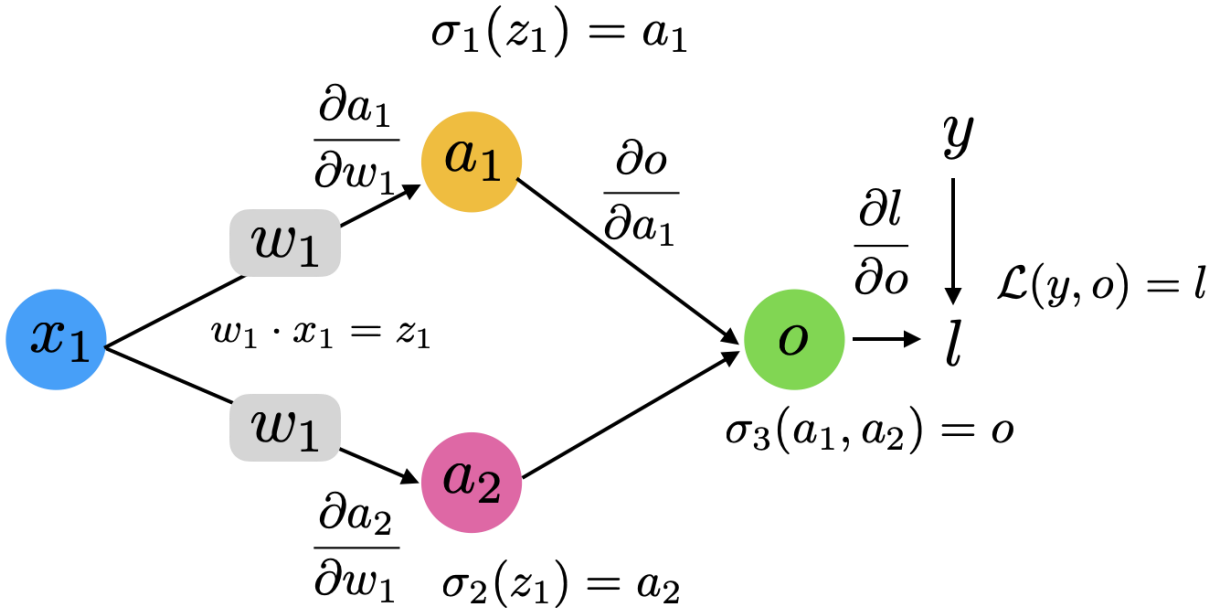
Computation graphs: Fully-Connected Layer



$$\frac{\partial l}{\partial w_{1,1}^{(1)}} = \frac{\partial l}{\partial o} \cdot \frac{\partial o}{\partial a_1^{(2)}} \cdot \frac{\partial a_1^{(2)}}{\partial a_1^{(1)}} \cdot \frac{\partial a_1^{(1)}}{\partial w_{1,1}^{(1)}}$$
$$+ \frac{\partial l}{\partial o} \cdot \frac{\partial o}{\partial a_2^{(2)}} \cdot \frac{\partial a_2^{(2)}}{\partial a_1^{(1)}} \cdot \frac{\partial a_1^{(1)}}{\partial w_{1,1}^{(1)}}$$

Computation graphs: Weight-Sharing

$$\mathcal{L}\big(y, \sigma_3\big[\sigma_1(w_1 \cdot x_1), \sigma_2(w_1 \cdot x_1)\big]\big)$$



Upper path

$$\frac{\partial l}{\partial w_1} = \frac{\partial l}{\partial o} \cdot \frac{\partial o}{\partial a_1} \cdot \frac{\partial a_1}{\partial w_1} + \frac{\partial l}{\partial o} \cdot \frac{\partial o}{\partial a_2} \cdot \frac{\partial a_2}{\partial w_1}$$ (multivariable chain rule)

Lower path

Questions?