UVA DATA SCIENCE

DS 6050 Deep Learning

2. Introductions to Neural Networks

Sheng Li
Associate Professor
School of Data Science
University of Virginia

Motivation

- Typical machine learning tasks:

y-hat(i)

$$\min_{\boldsymbol{\theta}} \sum_i \left\| y^{(i)} - f\left(\mathbf{x}^{(i)} \middle| \boldsymbol{\theta}\right) \right\|^2$$

finding the value of theta so that this is minimzed...

  - where…
    - $\left(\mathbf{x}^{(i)}, y^{(i)}\right)$: i-th data point in a dataset.
    - $f(\cdot\,|\boldsymbol{\theta})$: machine learning model with parameters $\boldsymbol{\theta}$.

- Examples:
  - x=[age, years in school, major], y=income (regression)
  - x=image, y=cat/dog (classification)

So, how do we find $\boldsymbol{\theta}$?

Linear Models

- $\min_{\boldsymbol{\theta}} \sum_i \left\| y^{(i)} - f\left(\mathbf{x}^{(i)} \middle| \boldsymbol{\theta}\right) \right\|^2$ where $f(\mathbf{x}|\boldsymbol{\theta}) = \theta_0 + \theta_1 x_1 + \cdots + \theta_{d-1} x_{d-1} = \mathbf{x}^{\mathrm{T}}\boldsymbol{\theta}$
- Let $\mathbf{X} \coloneqq \left[\left(\mathbf{x}^{(i)}\right)^{\mathrm{T}}\right]$, and $\mathbf{y} \coloneqq \left[y^{(i)}\right]$, then:

$$\mathcal{L}(\boldsymbol{\theta}) = \|\mathbf{y} - \mathbf{X}\boldsymbol{\theta}\|^2$$

Linear Models

- Solution:

$$\mathcal{L}(\boldsymbol{\theta}) = \|\mathbf{y} - \mathbf{X}\boldsymbol{\theta}\|^2 = (\mathbf{y} - \mathbf{X}\boldsymbol{\theta})^{\mathrm{T}}(\mathbf{y} - \mathbf{X}\boldsymbol{\theta})$$
$$= \left(\mathbf{y}^{\mathrm{T}} - \boldsymbol{\theta}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}\right)(\mathbf{y} - \mathbf{X}\boldsymbol{\theta})$$
$$= \mathbf{y}^{\mathrm{T}}\mathbf{y} - \boldsymbol{\theta}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}\mathbf{y} - \mathbf{y}^{\mathrm{T}}\mathbf{X}\boldsymbol{\theta} + \boldsymbol{\theta}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}\mathbf{X}\boldsymbol{\theta}$$
$$= \mathbf{y}^{\mathrm{T}}\mathbf{y} - 2\mathbf{y}^{\mathrm{T}}\mathbf{X}\boldsymbol{\theta} + \boldsymbol{\theta}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}\mathbf{X}\boldsymbol{\theta}$$

First order necessary condition: $\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}(\boldsymbol{\theta}) = -2\mathbf{y}^{\mathrm{T}}\mathbf{X} + 2\boldsymbol{\theta}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}\mathbf{X} \equiv 0$

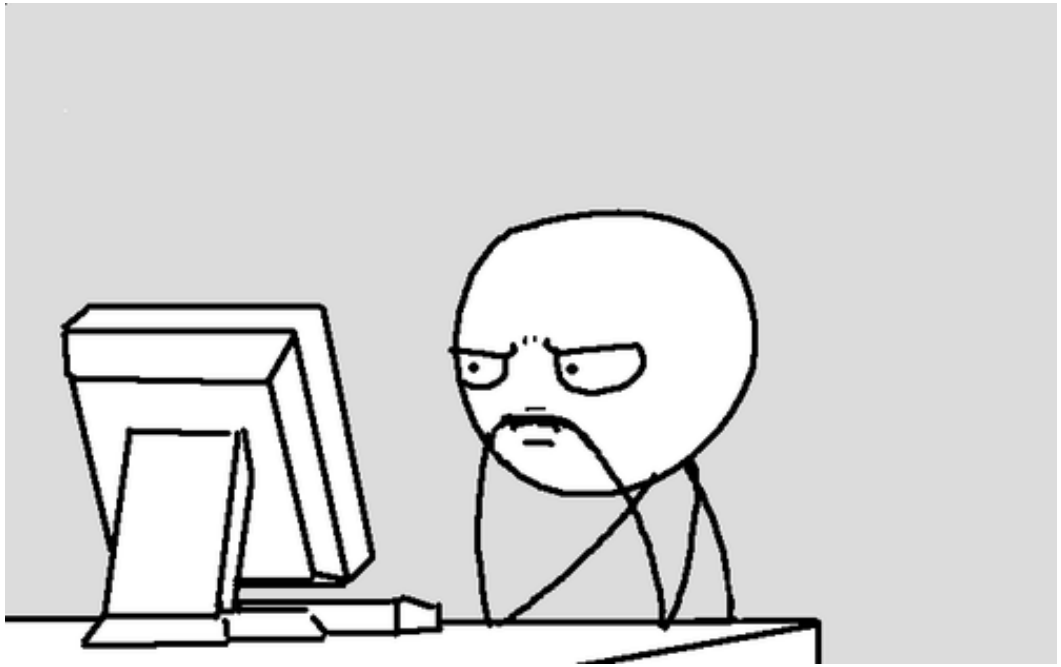
$$\Leftrightarrow \boldsymbol{\theta}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}\mathbf{X} = \mathbf{y}^{\mathrm{T}}\mathbf{X}$$
$$\Leftrightarrow \mathbf{X}^{\mathrm{T}}\mathbf{X}\boldsymbol{\theta} = \mathbf{X}^{\mathrm{T}}\mathbf{y}$$
$$\Leftrightarrow \boldsymbol{\theta} = \left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}$$

closed form solution

A bit of nonlinearity…

- Minimize $f(x) = (\cos x + \tan x)^2$, w.r.t. $x \in (-1,1)$
  - First order necessary condition:

$$\frac{\partial f}{\partial x} = 2(\cos x + \tan x)(-\sin x + \sec^2 x) \equiv 0$$

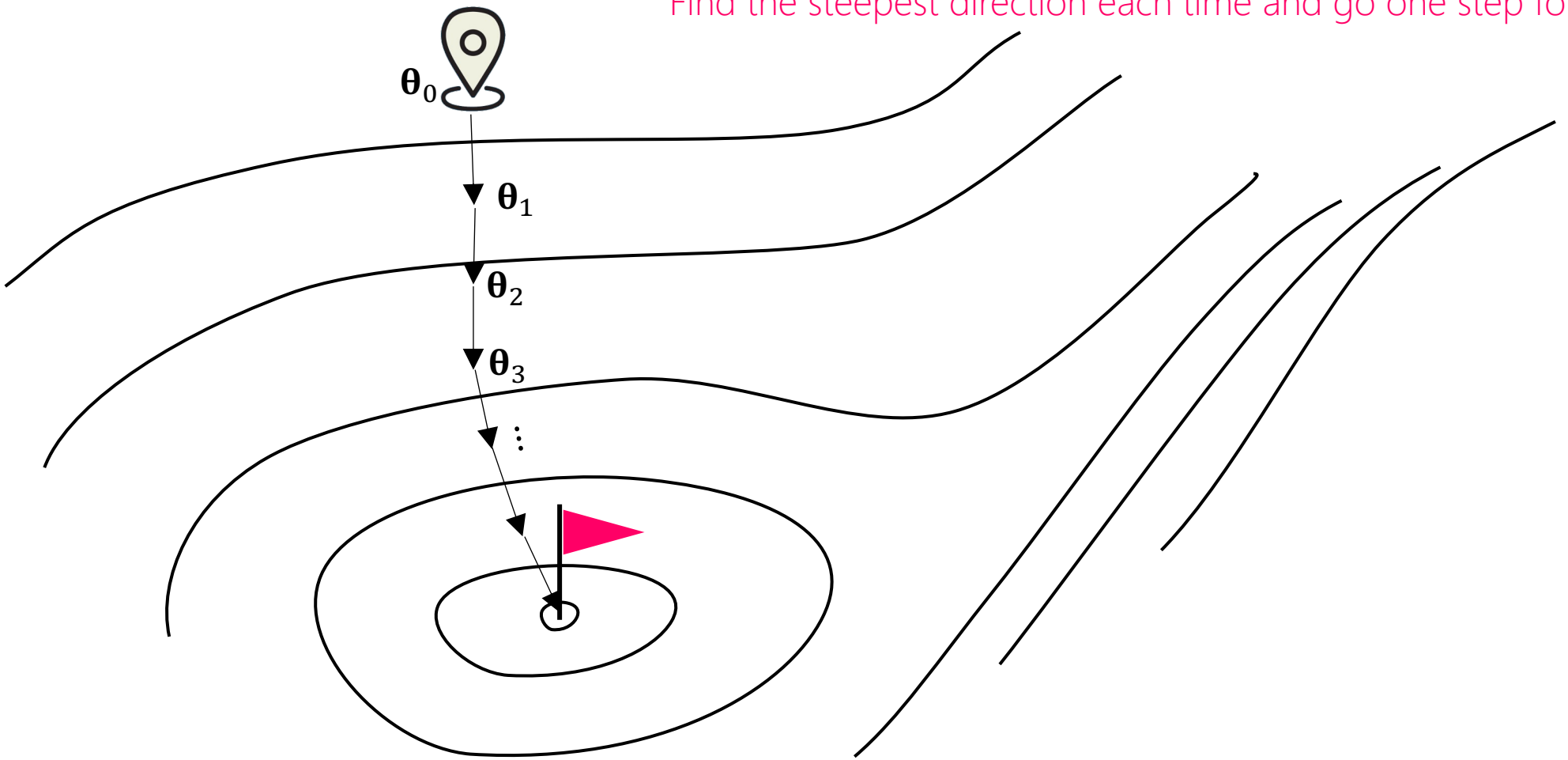


Climbing up a Mountain

- Q. Suppose you're an ***extremely*** near-sighted person (can only see things within, say 6 ft., of your periphery). What would be the best strategy to get to the peak?

A Strategy: Steepest Ascent

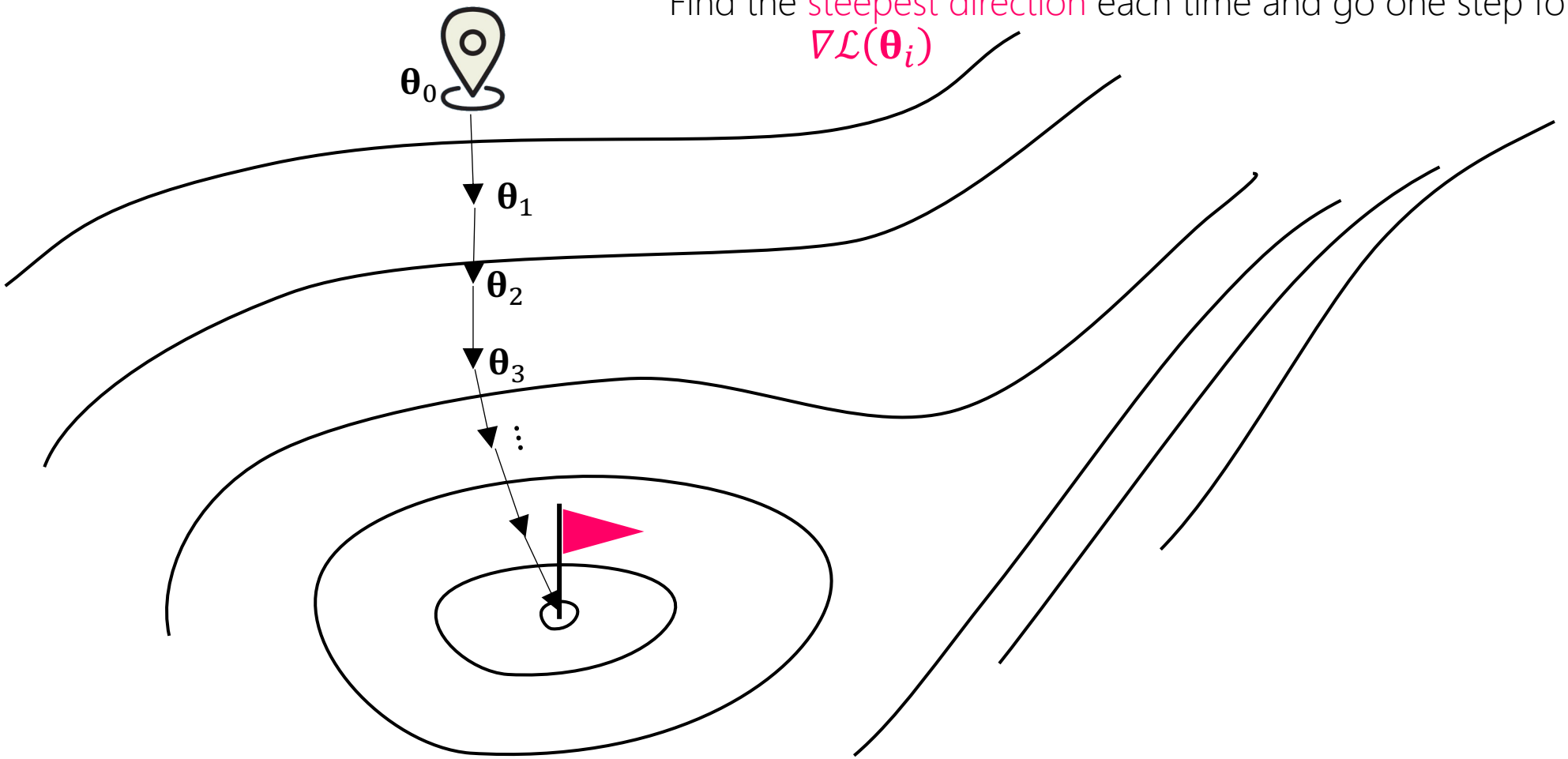
"Find the steepest direction each time and go one step forward."



A Strategy: Steepest Ascent

"Find the steepest direction each time and go one step forward."

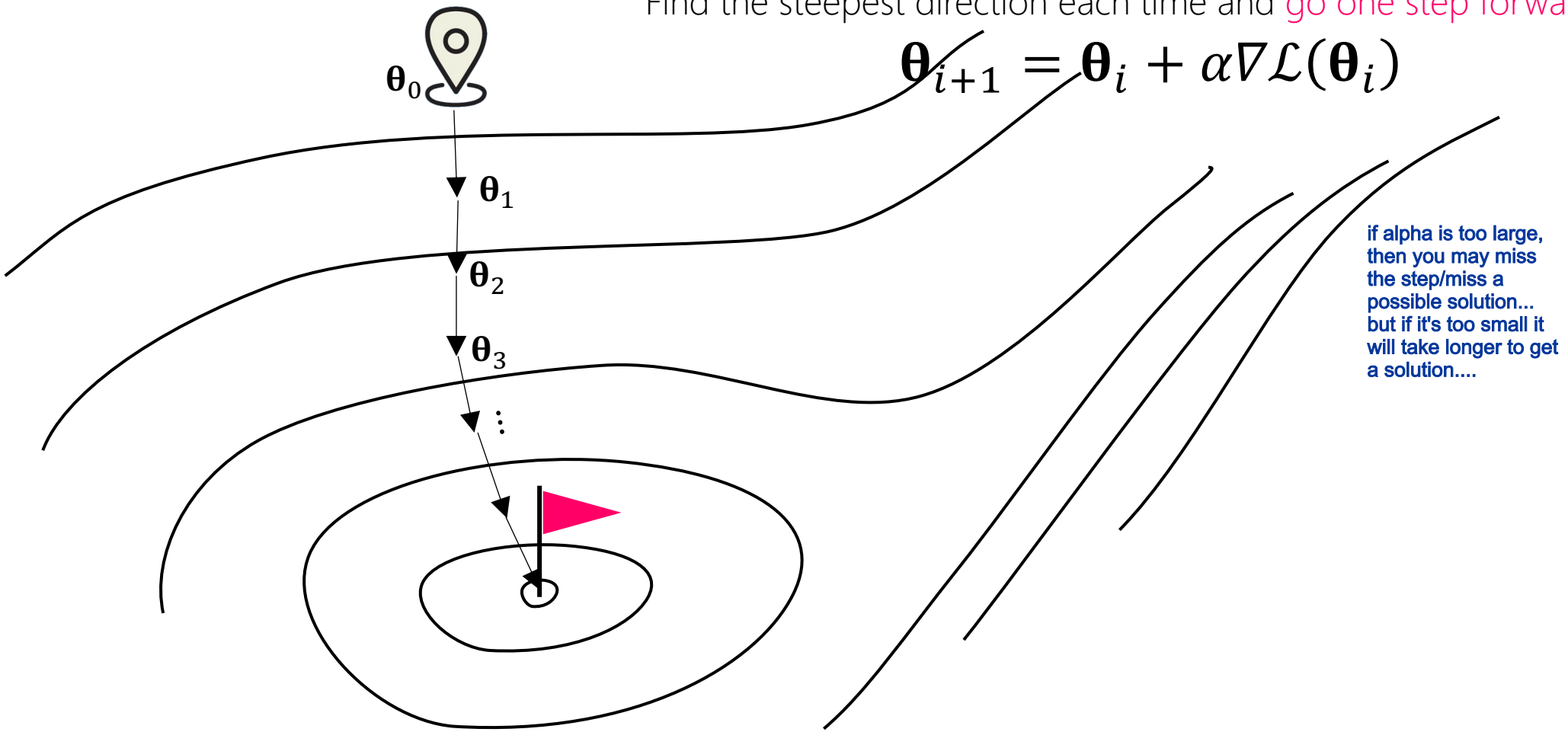
$\nabla \mathcal{L}(\boldsymbol{\theta}_i)$



A Strategy: Steepest Ascent

"Find the steepest direction each time and go one step forward."

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i + \alpha \nabla \mathcal{L}(\boldsymbol{\theta}_i)$$



if alpha is too large, then you may miss the step/miss a possible solution... but if it's too small it will take longer to get a solution....

Steepest Descent Algorithm (a.k.a. Gradient Descent)

- Given a differentiable function $\mathcal{L}$, the function value decreases the fastest if one goes in the direction of the negative gradient of $\mathcal{L}$.
- It follows that, for small enough scalar value α, if

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \alpha \nabla \mathcal{L}(\boldsymbol{\theta}_i)$$

then $\mathcal{L}(\boldsymbol{\theta}_{i+1}) \leq \mathcal{L}(\boldsymbol{\theta}_i)$.

Machine Learning with Gradient Descent

1. Design your model $f(\mathbf{x}|\boldsymbol{\theta})$, and the learning objective $\mathcal{L}(\boldsymbol{\theta}|\mathbf{x}, y)$.
2. Initialize the model parameters $\boldsymbol{\theta}$ (usually with random numbers).
3. Evaluate the gradient $\nabla\mathcal{L}$ using the current model parameters $\boldsymbol{\theta}$ and training dataset $\{\mathbf{x}^{(i)}, y^{(i)}\}$.
4. Improve the model parameters with a given learning rate α and the update strategy: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha\nabla\mathcal{L}$.
5. Repeat 3~4 until convergence

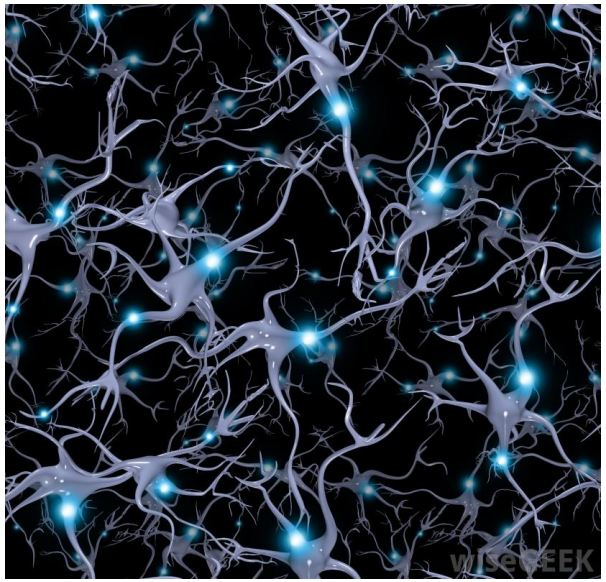
Machine Learning with Gradient Descent

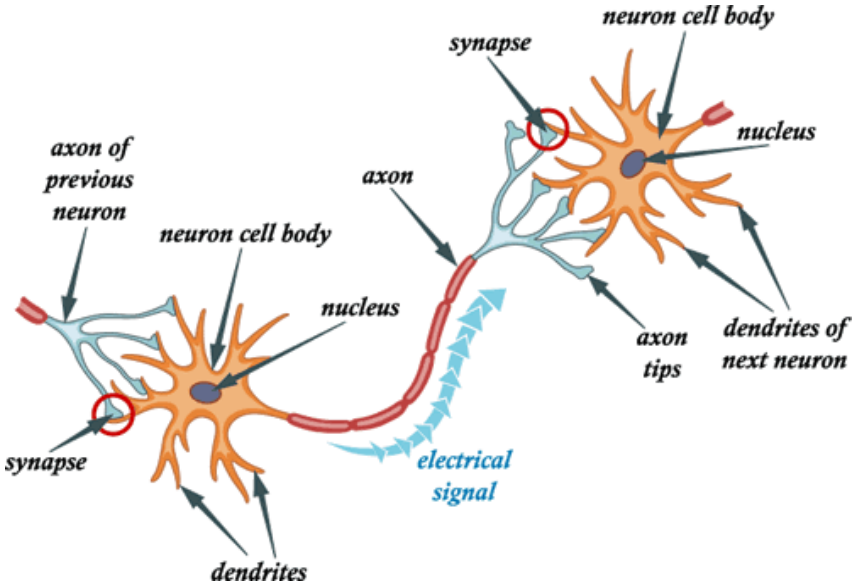
we're going to focus on neural networks here

1. Design your model $f(\mathbf{x}|\boldsymbol{\theta})$, and the learning objective $\mathcal{L}(\boldsymbol{\theta}|\mathbf{x}, y)$.
2. Initialize the model parameters $\boldsymbol{\theta}$ (usually with random numbers).
3. Evaluate the gradient $\nabla\mathcal{L}$ using the current model parameters $\boldsymbol{\theta}$ and training dataset $\{\mathbf{x}^{(i)}, y^{(i)}\}$.
4. Improve the model parameters with a given learning rate α and the update strategy: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha\nabla\mathcal{L}$.
5. Repeat 3~4 until converges

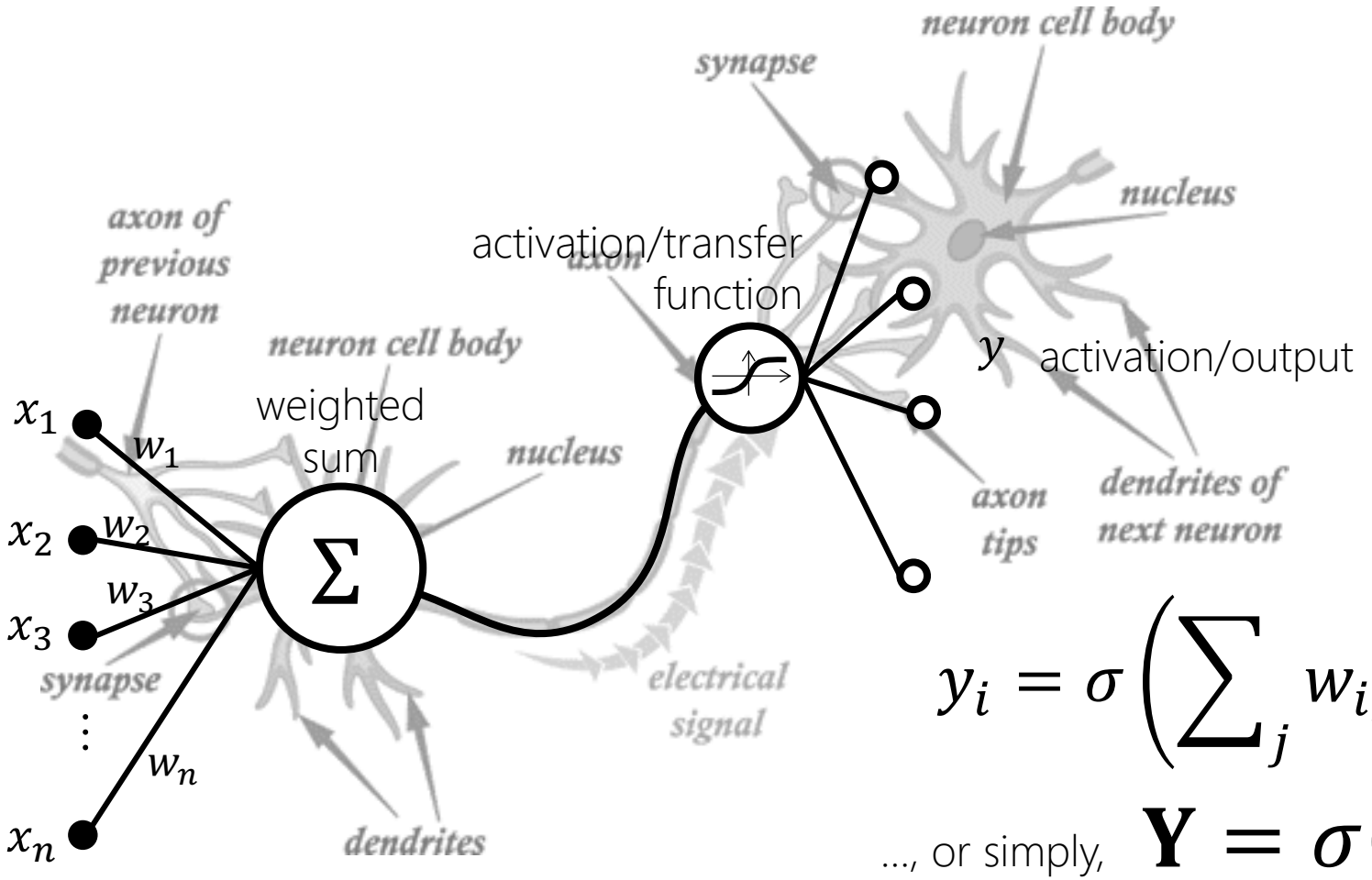
Neural Networks

Biological Neural Networks





Artificial Neural Networks (ANN)



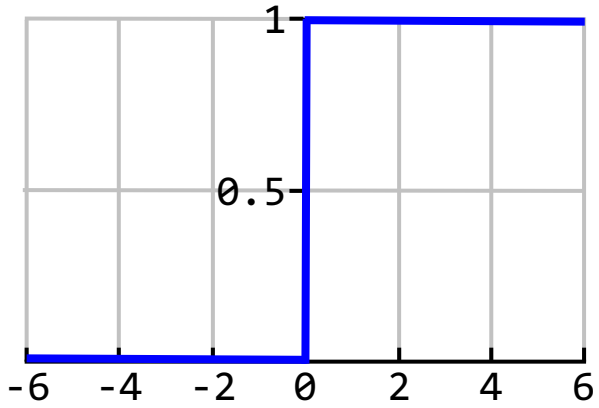
$$y_i = \sigma\left(\sum_j w_{i,j} x_j\right)$$

..., or simply, $\mathbf{Y} = \sigma(\mathbf{XW})$

Activation/Transfer Functions

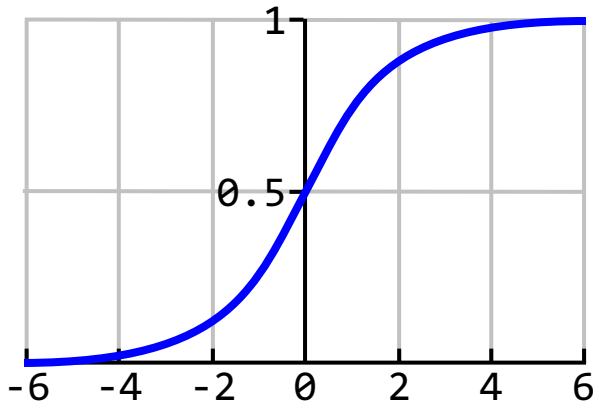
- "Threshold" in biological systems
- Step Function:

$$y = \sigma(z) = \begin{cases} 1 & \text{if } u \geq 0 \\ 0 & \text{otherwise} \end{cases}$$



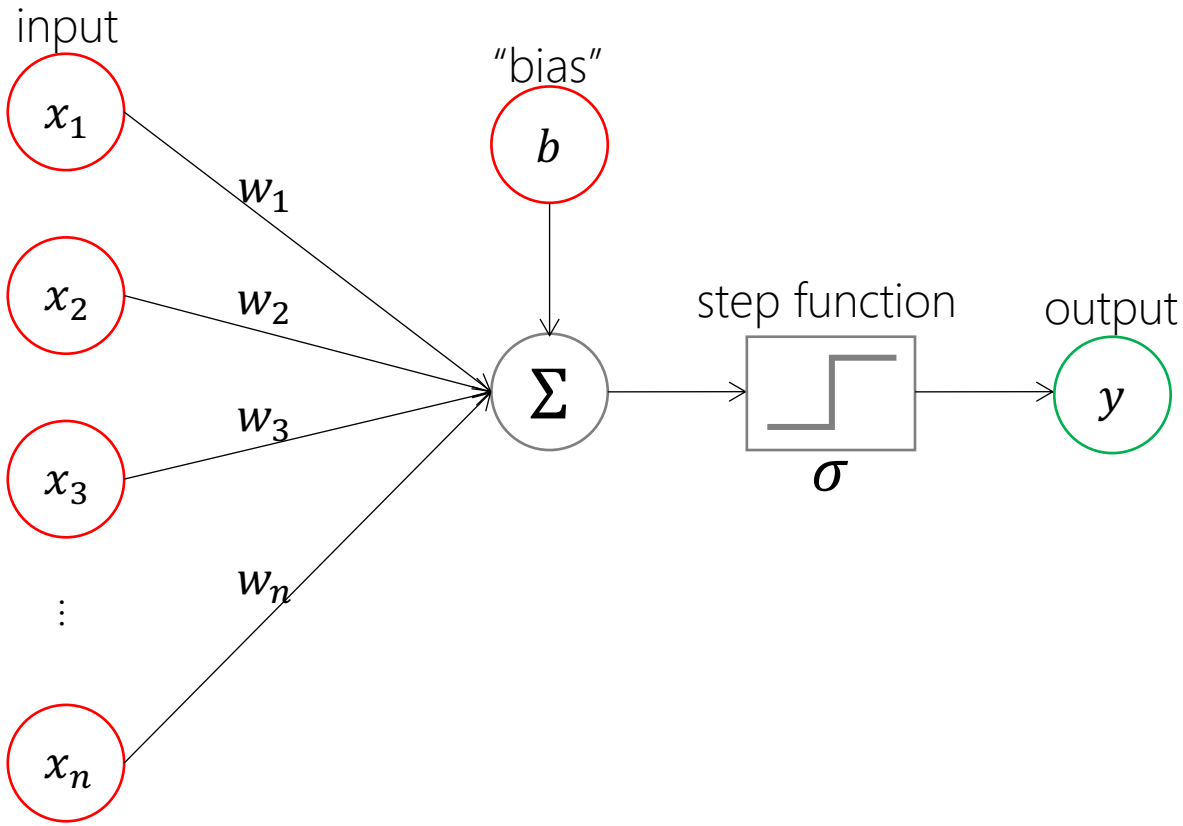
- Sigmoid Function:

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



The simplest model: the "Perceptron"

- Frank Rosenblatt (1957)



$$y = \sigma\left(\sum_j w_j x_j + b\right)$$

For each example $\left(x^{(t)}, \hat{y}^{(t)}\right)$, try to minimize:

$$\mathcal{L} = \frac{1}{2}\left(y^{(t)} - \hat{y}^{(t)}\right)^2$$

$$\frac{\partial \mathcal{L}}{\partial w_j} = \left(y^{(t)} - \hat{y}^{(t)}\right)\frac{\partial y^{(t)}}{\partial w_j}$$

$$= \left(y^{(t)} - \hat{y}^{(t)}\right)x_j^{(t)}$$

Weight update scheme:

$$w(t+1) = w(t) - r\left(y^{(t)} - \hat{y}^{(t)}\right)x_j^{(t)}$$

learning rate

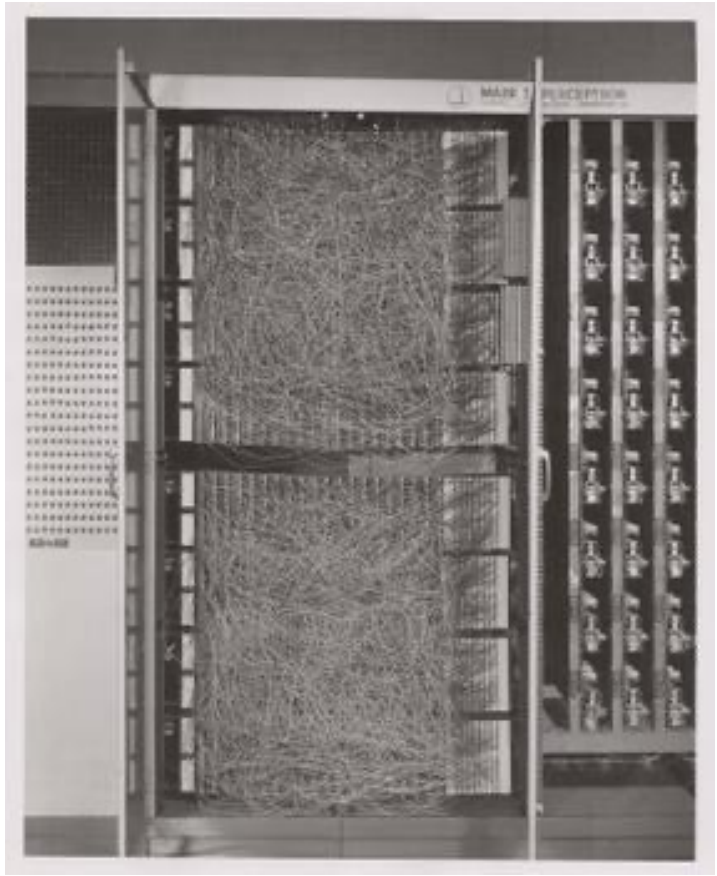
The simplest model: the “Perceptron”

- Frank Rosenblatt (1957)
  - The **Mark I Perceptron** machine was the first implementation of the perceptron algorithm.
  - Designed for image recognition.
  - Connected to a camera that used a 20x20 cadmium sulfide photocell array to produce a 400-pixel image.

http://www.rutherfordjournal.org/article040101.html

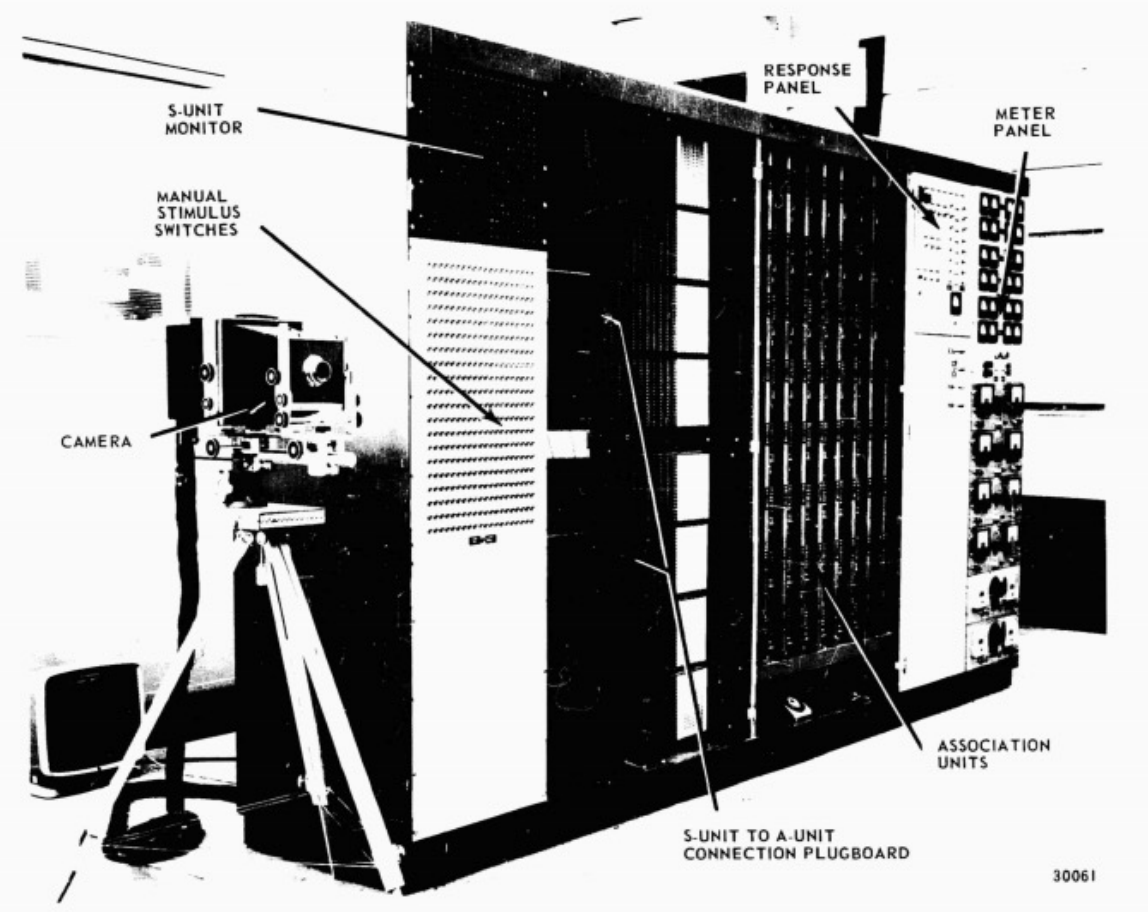
The simplest model: the “Perceptron”

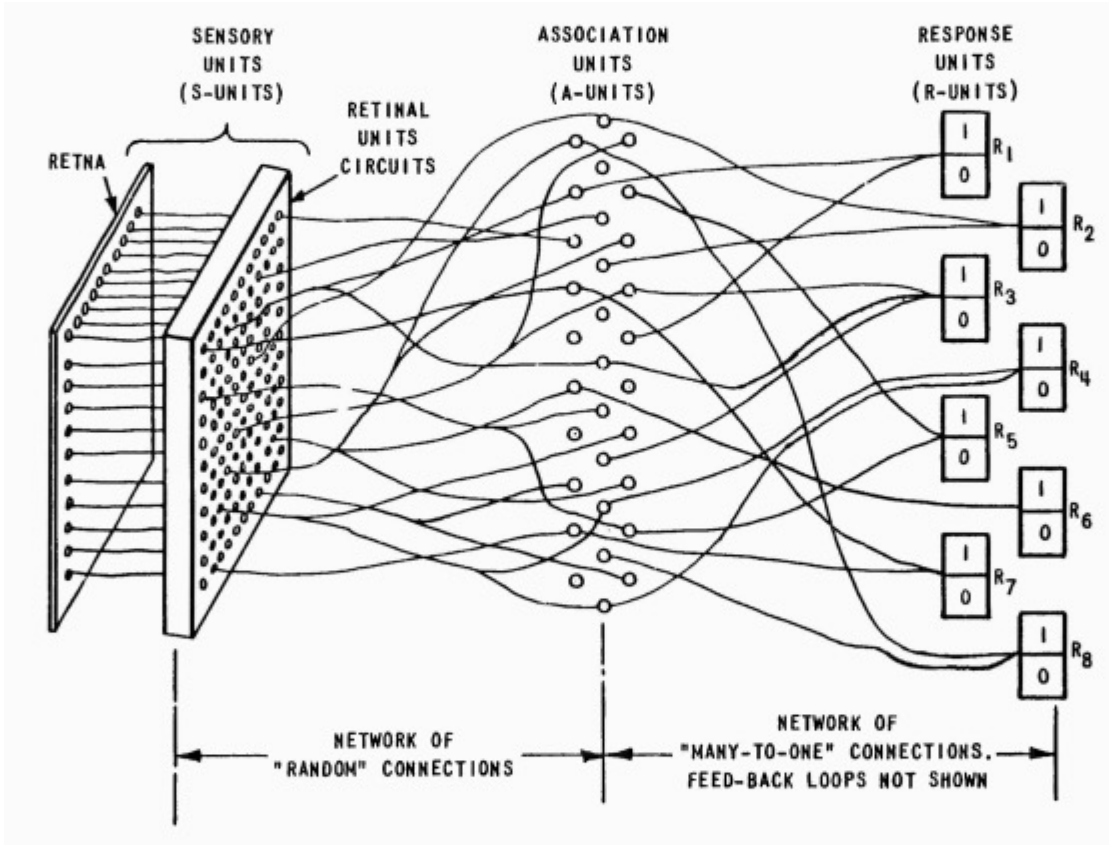
- Frank Rosenblatt (1957)
  - The **Mark I Perceptron** machine was the first implementation of the perceptron algorithm.
  - Designed for image recognition.
  - Connected to a camera that used a 20x20 cadmium sulfide photocell array to produce a 400-pixel image.
  - Weights were encoded in potentiometers.
  - Weight updates were performed by electric motors.



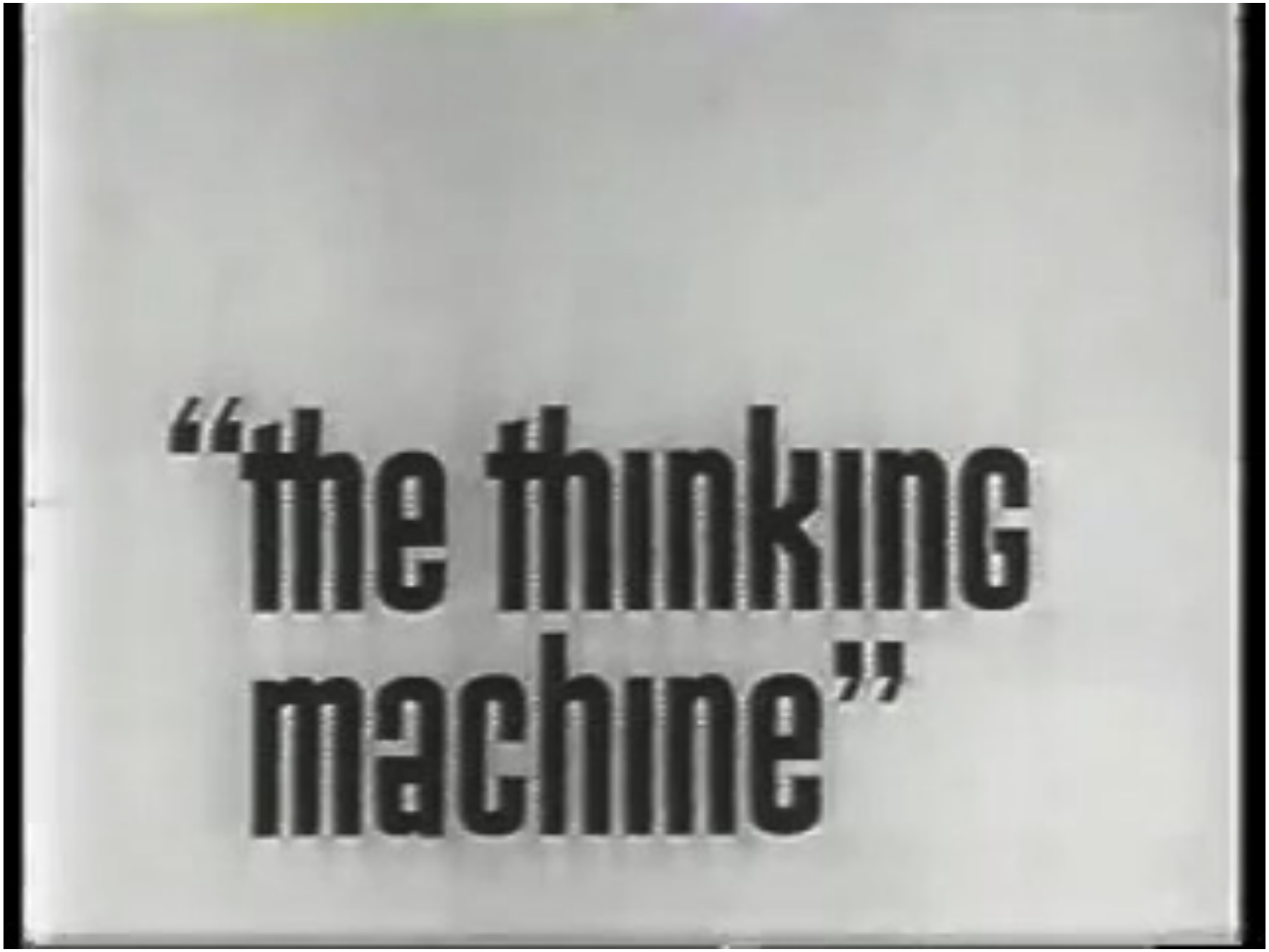
Wikipedia – Perceptron

The simplest model: the “Perceptron”





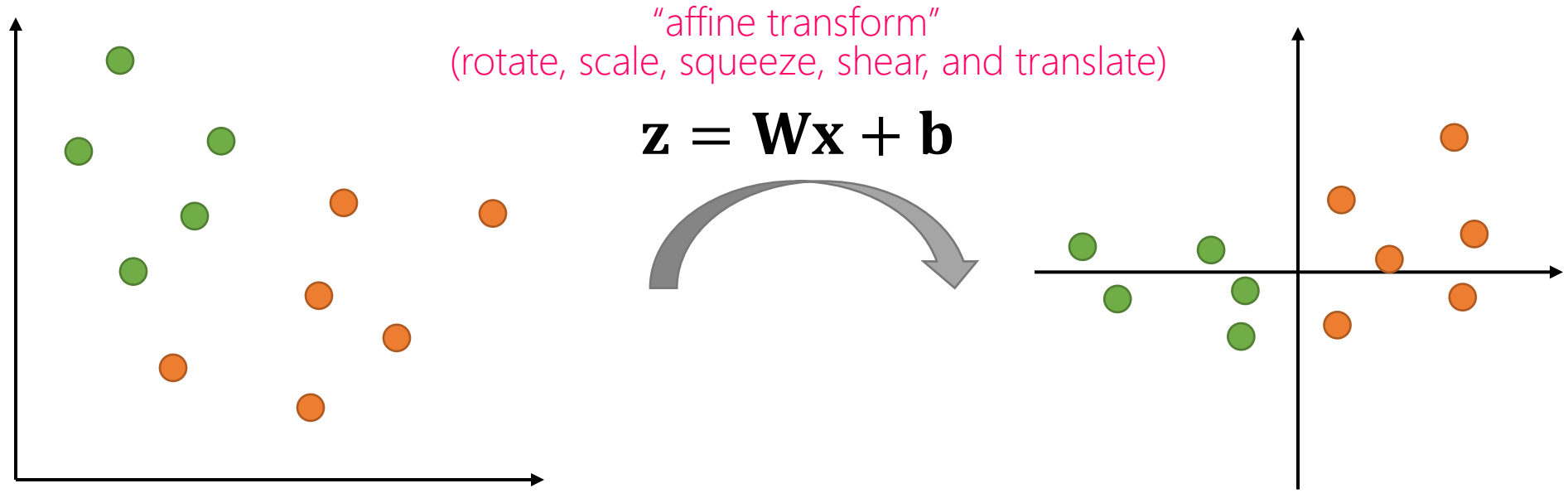
Mark 1 Perceptron Operators’ Manual (http://www.dtic.mil/dtic/tr/fulltext/u2/236965.pdf)



https://www.youtube.com/watch?v=aygSMgK3BEM

Perceptron as Coordinate Transformation

- Perceptron is essentially a **transformation of data** representation.



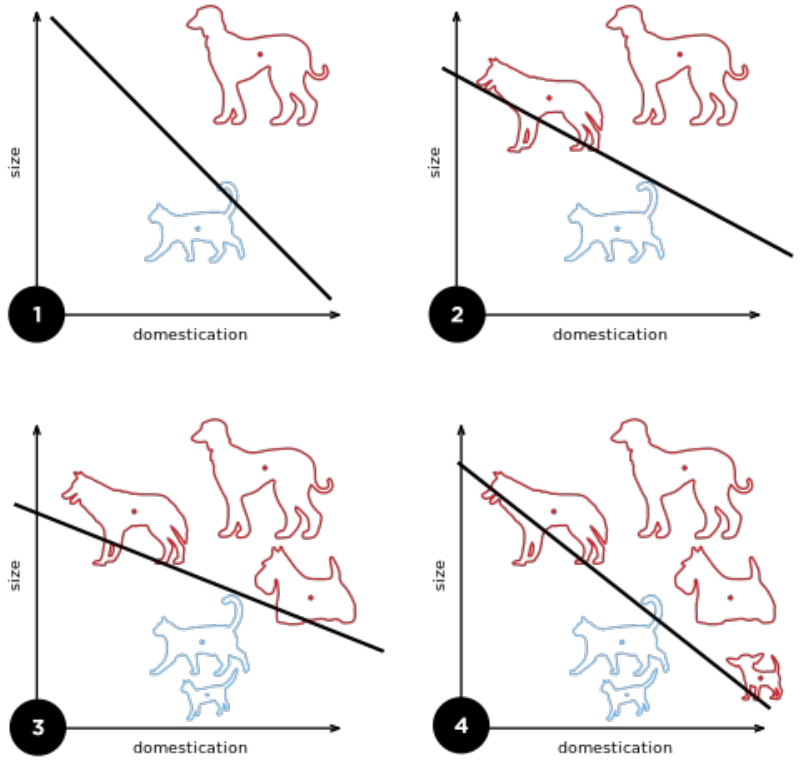
Perceptron as Linear Binary Classification

- Perceptron returns 1 iff. $\sum_j w_j x_j + b \geq 0$ and 0 otherwise.
- Essentially, this is equivalent to linear binary classification problem

$$y = \sum_j w_j x_j + b$$

slope

intercept



Wikipedia – Perceptron

The XOR Problem

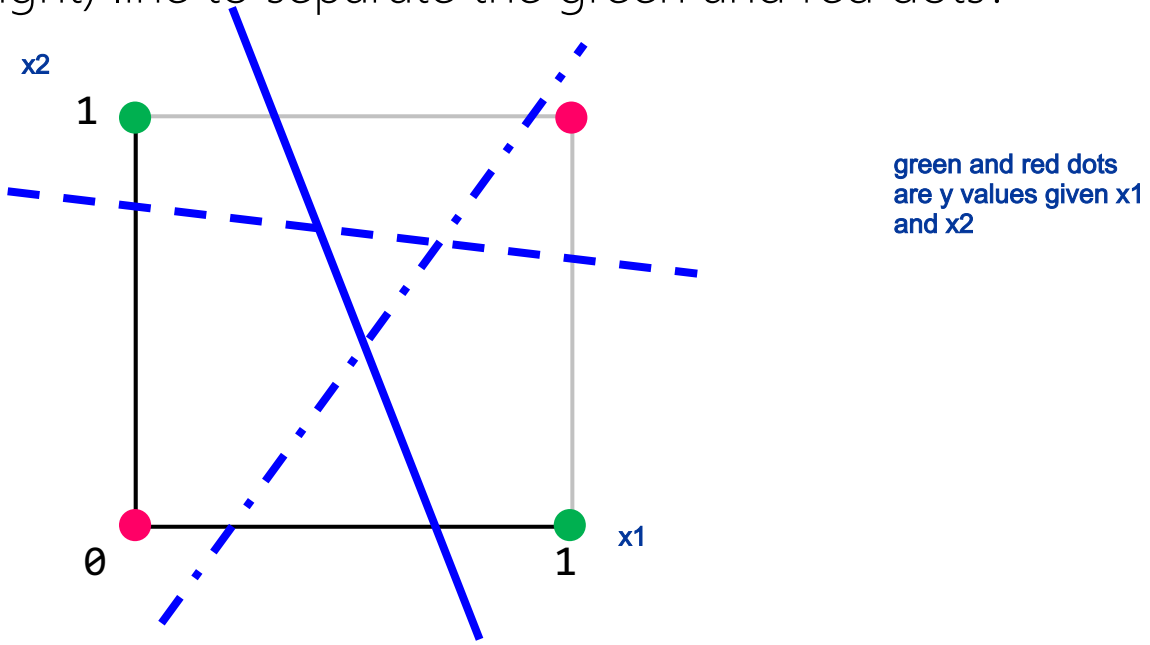
- XOR: Exclusive OR:

x_1	x_2	y
0	0	0
0	1	1
1	1	0
1	0	1

- Q. Can you implement the XOR gate with the perceptron model?

The XOR Problem

- The quick answer is NO.
- Why?
  - a perceptron is a linear classifier.
  - Can you draw a single (straight) line to separate the green and red dots?

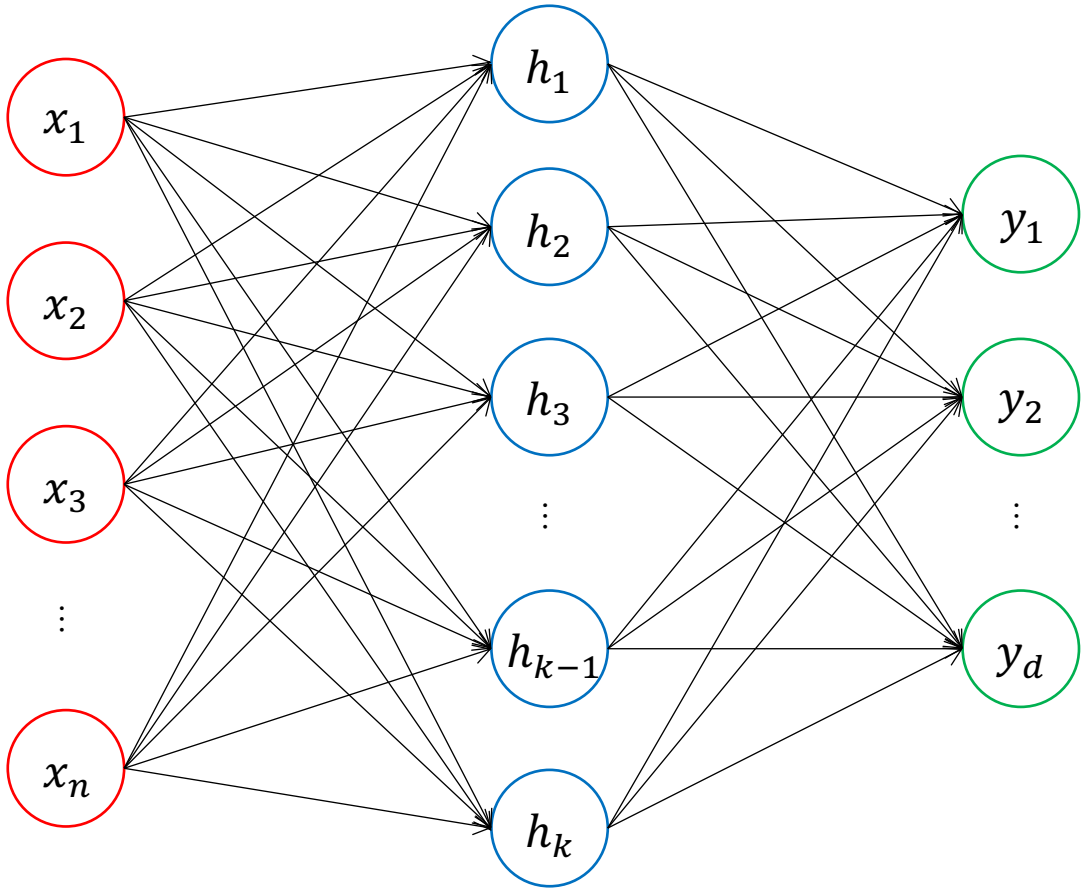


The XOR Problem

- Minsky & Papert. (1969). Perceptron: an introduction to computational geometry. The MIT Press, Cambridge.
  - Mathematical proof of the limitation of perceptrons…

Multi-Layer Perceptron (MLP)

- Minsky & Papert. (1969)

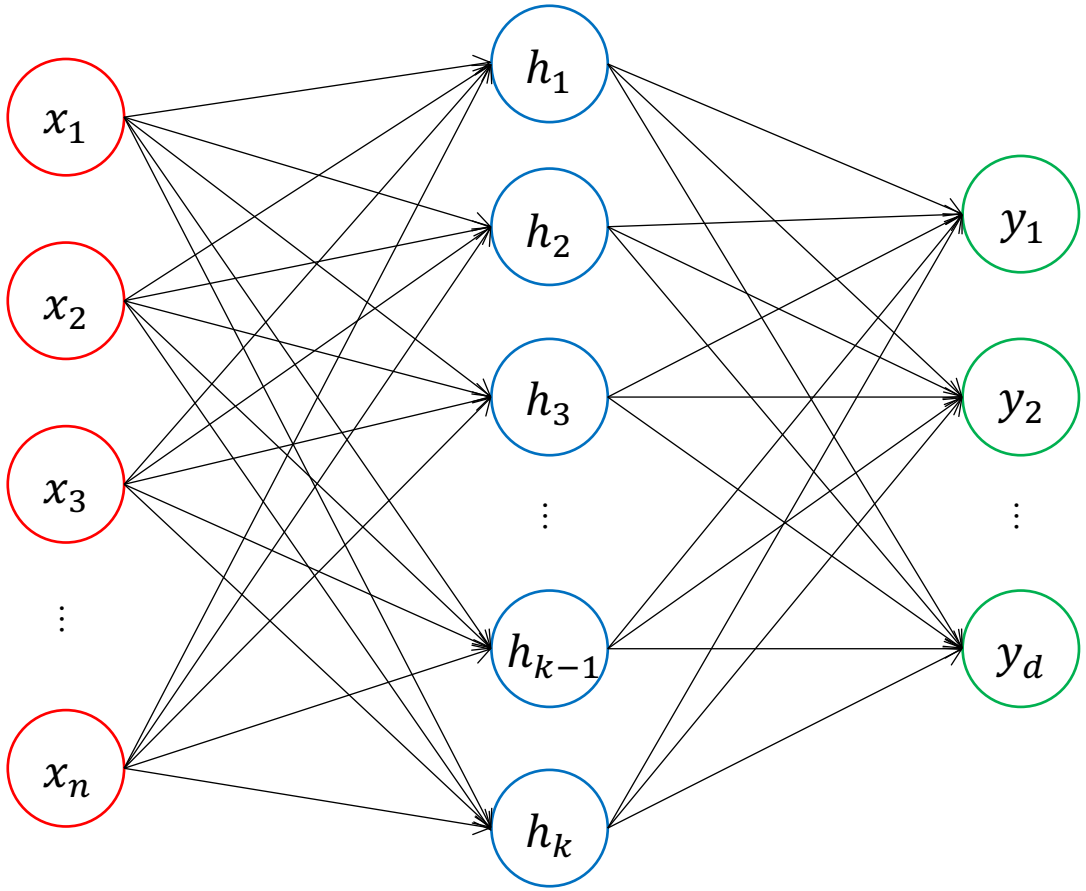


Marvin Minsky

> The perceptron has shown itself worthy of study despite (and even because of!) its severe limitations. It has many features to attract attention: its linearity; its intriguing learning theorem; its clear paradigmatic simplicity as a kind of parallel computation. There is no reason to suppose that any of these virtues carry over to the many-layered version. Nevertheless, we consider it to be an important research problem to elucidate (or reject) our intuitive judgement that the extension to multilayer systems is sterile.
>
> *Minsky & Papert* (1969, pp. 231-232)

Multi-Layer Perceptron (MLP)

- Minsky & Papert. (1969)



$$h_i = \sigma_h\left(\sum_j w_{h,i,j}x_j + b_h\right) \quad y_k = \sigma_o\left(\sum_i w_{o,k,i}h_i + b_o\right)$$

$$\mathcal{L} = \frac{1}{2}\left\|\mathbf{y}^{(t)} - \hat{\mathbf{y}}^{(t)}\right\|^2$$

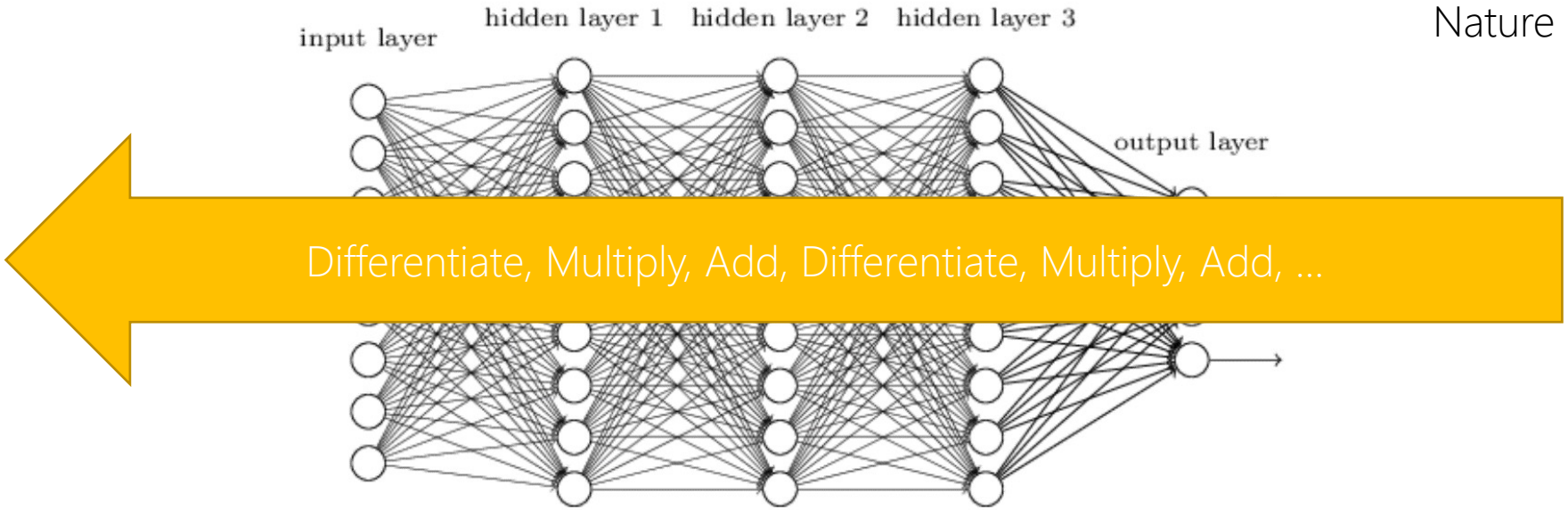
$$\frac{\partial \mathcal{L}}{\partial w} = ?$$

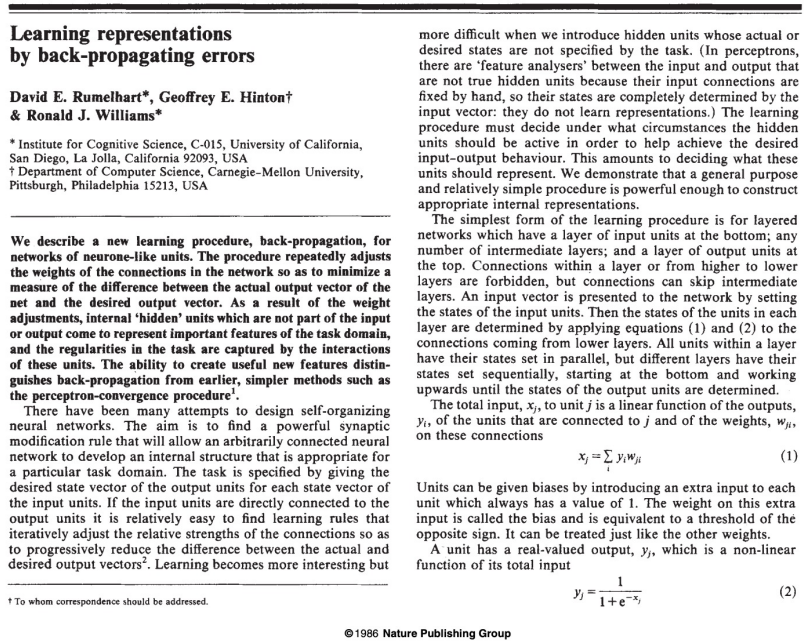
- Minsky's devastating criticism on perceptrons
- "We don't know how to train MLPs"
- Backpropagation (we'll see it soon) existed, but not much attention.

→ The First AI Winter (1970s)

Backpropagation

- Rumelhart, Hinton, and Williams. (1986).
- A popular training method for neural nets
- Propagate what? “the gradient of the current error”





Learning representations by back-propagating errors

David E. Rumelhart*, Geoffrey E. Hinton† & Ronald J. Williams*

* Institute for Cognitive Science, C-015, University of California, San Diego, La Jolla, California 92093, USA
† Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Philadelphia 15213, USA

We describe a new learning procedure, back-propagation, for networks of neurone-like units. The procedure repeatedly adjusts the weights of the connections in the network so as to minimize a measure of the difference between the actual output vector of the net and the desired output vector. As a result of the weight adjustments, internal ‘hidden’ units which are not part of the input or output come to represent important features of the task domain, and the regularities in the task are captured by the interactions of these units. The ability to create useful new features distinguishes back-propagation from earlier, simpler methods such as the perceptron-convergence procedure[1].

There have been many attempts to design self-organizing neural networks. The aim is to find a powerful synaptic modification rule that will allow an arbitrarily connected neural network to develop an internal structure that is appropriate for a particular task domain. The task is specified by giving the desired state vector of the output units for each state vector of the input units. If the input units are directly connected to the output units it is relatively easy to find learning rules that iteratively adjust the relative strengths of the connections so as to progressively reduce the difference between the actual and desired output vectors[2]. Learning becomes more interesting but more difficult when we introduce hidden units whose actual or desired states are not specified by the task. (In perceptrons, there are ‘feature analysers’ between the input and output that are not true hidden units because their input connections are fixed by hand, so their states are completely determined by the input vector: they do not learn representations.) The learning procedure must decide under what circumstances the hidden units should be active in order to help achieve the desired input–output behaviour. This amounts to deciding what these units should represent. We demonstrate that a general purpose and relatively simple procedure is powerful enough to construct appropriate internal representations.

The simplest form of the learning procedure is for layered networks which have a layer of input units at the bottom; any number of intermediate layers; and a layer of output units at the top. Connections within a layer or from higher to lower layers are forbidden, but connections can skip intermediate layers. An input vector is presented to the network by setting the states of the input units. Then the states of the units in each layer are determined by applying equations (1) and (2) to the connections coming from lower layers. All units within a layer have their states set in parallel, but different layers have their states set sequentially, starting at the bottom and working upwards until the states of the output units are determined.

The total input, x_j, to unit j is a linear function of the outputs, y_i, of the units that are connected to j and of the weights, w_{ji}, on these connections

$$x_j = \sum_i y_i w_{ji} \quad (1)$$

Units can be given biases by introducing an extra input to each unit which always has a value of 1. The weight on this extra input is called the bias and is equivalent to a threshold of the opposite sign. It can be treated just like the other weights.

A unit has a real-valued output, y_j, which is a non-linear function of its total input

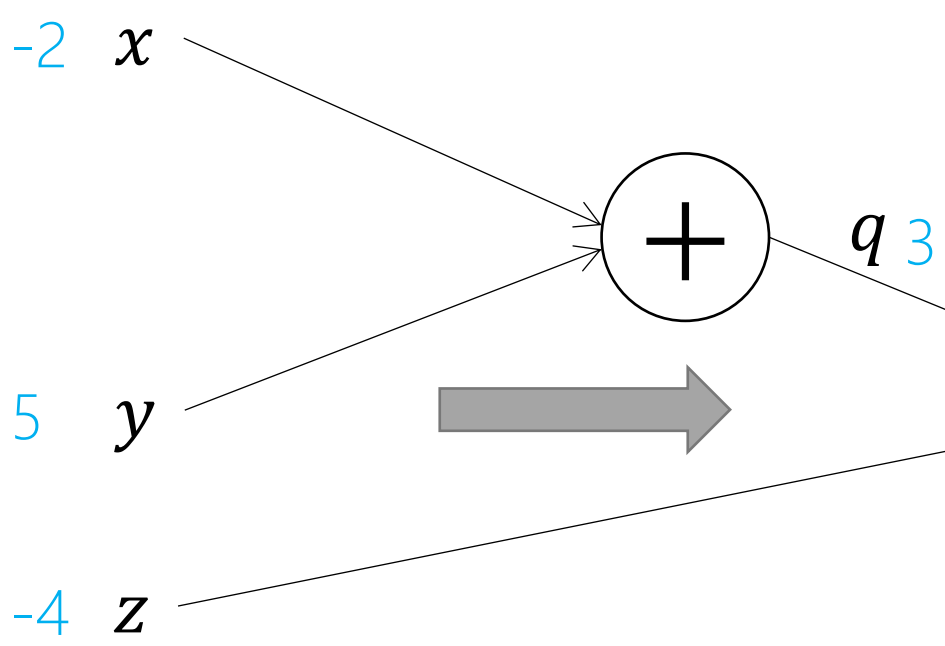
$$y_j = \frac{1}{1+e^{-x_j}} \quad (2)$$

† To whom correspondence should be addressed.

Nature (1986)

Backpropagation

- A simple example: $f(x, y, z) = (x + y)z$
  - Computational graph:



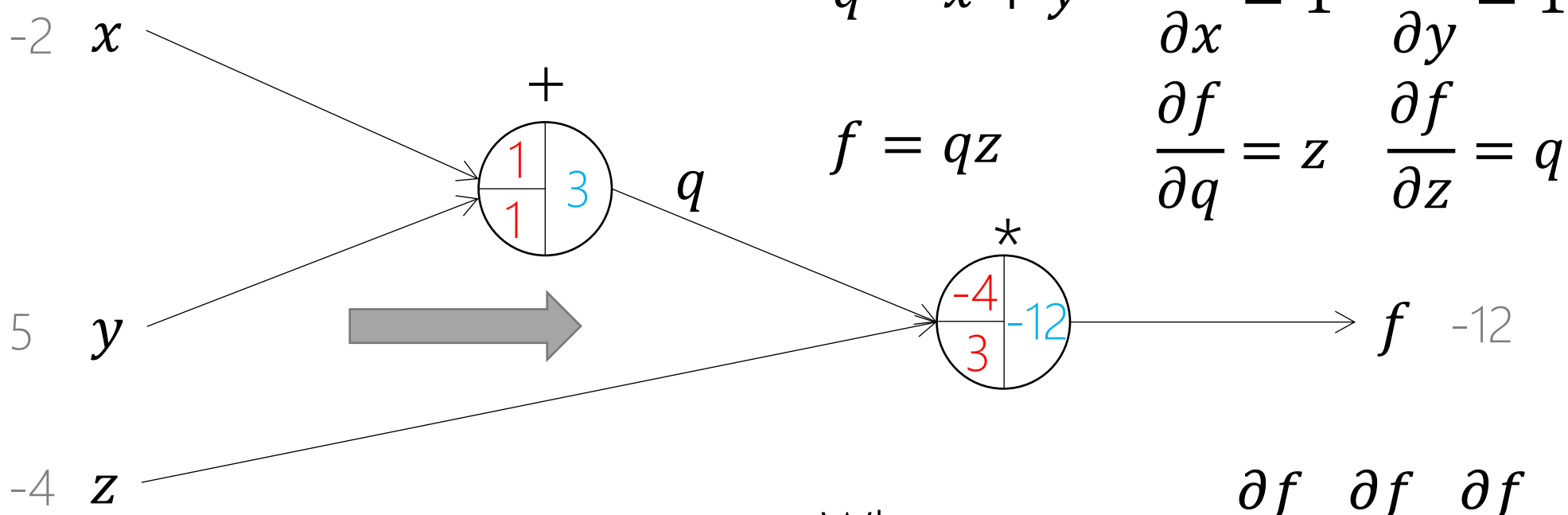
$$q = x + y \quad \frac{\partial q}{\partial x} = 1 \quad \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z \quad \frac{\partial f}{\partial z} = q$$

What we want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

Backpropagation

- A simple example: $f(x, y, z) = (x + y)z$
  - Computational graph:



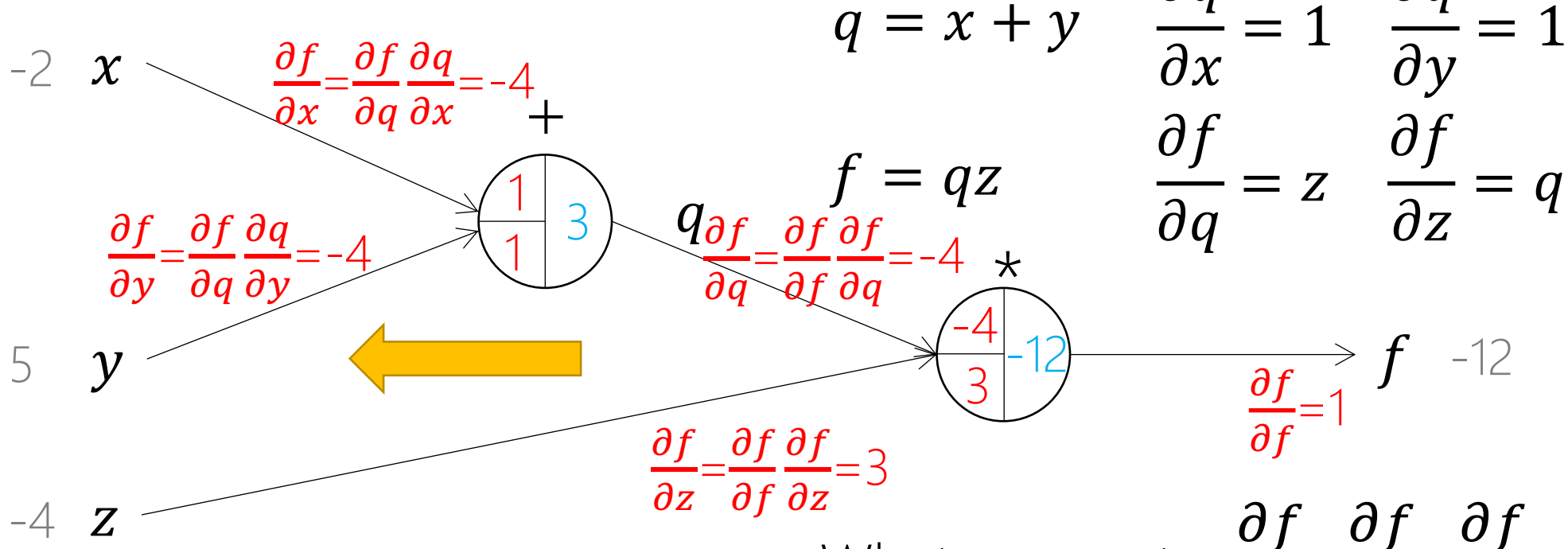
$$q = x + y \quad \frac{\partial q}{\partial x} = 1 \quad \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z \quad \frac{\partial f}{\partial z} = q$$

What we want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

Backpropagation

- A simple example: $f(x, y, z) = (x + y)z$
  - Computational graph:

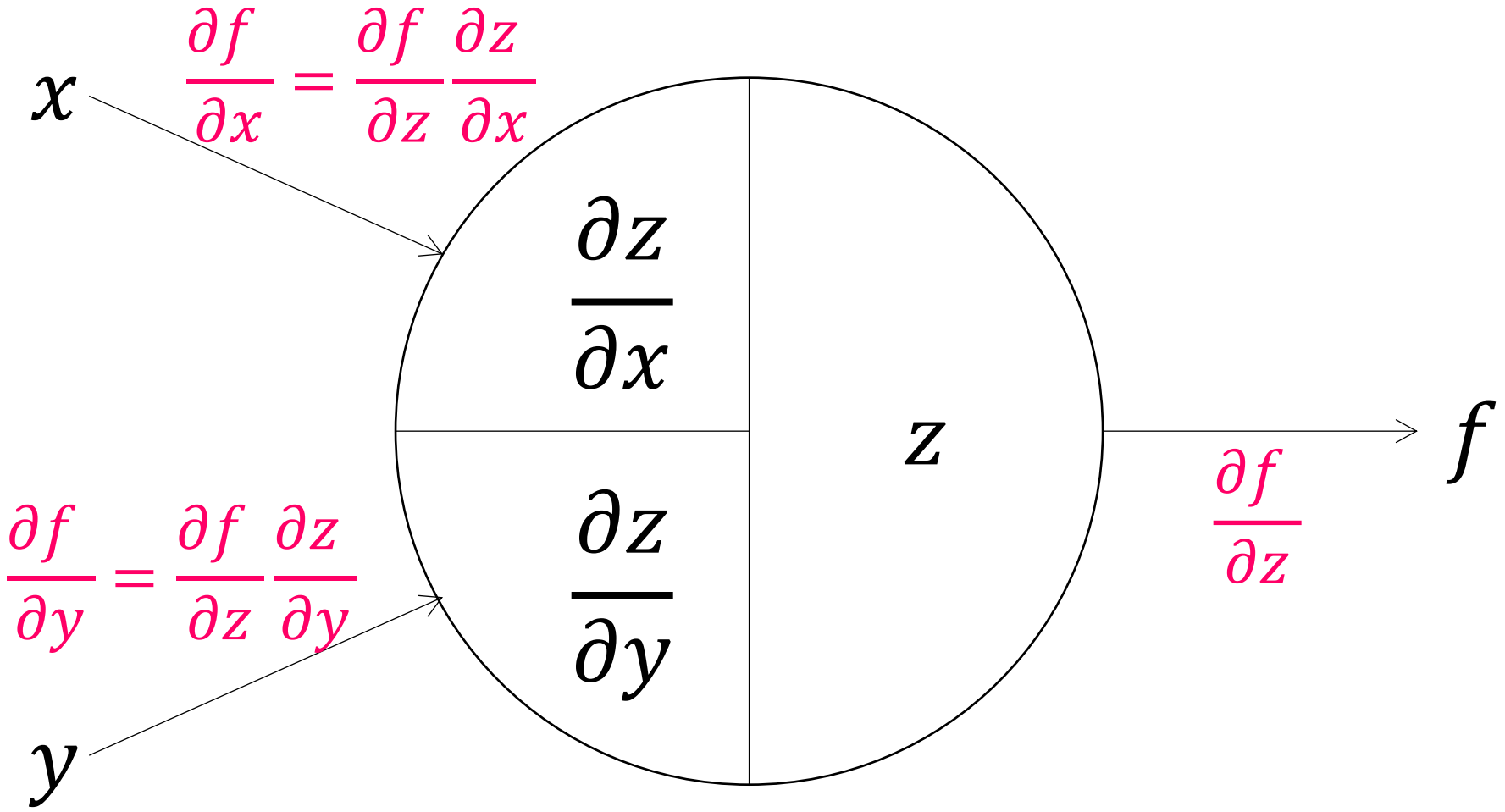


$q = x + y$ $\quad \frac{\partial q}{\partial x} = 1$ $\quad \frac{\partial q}{\partial y} = 1$

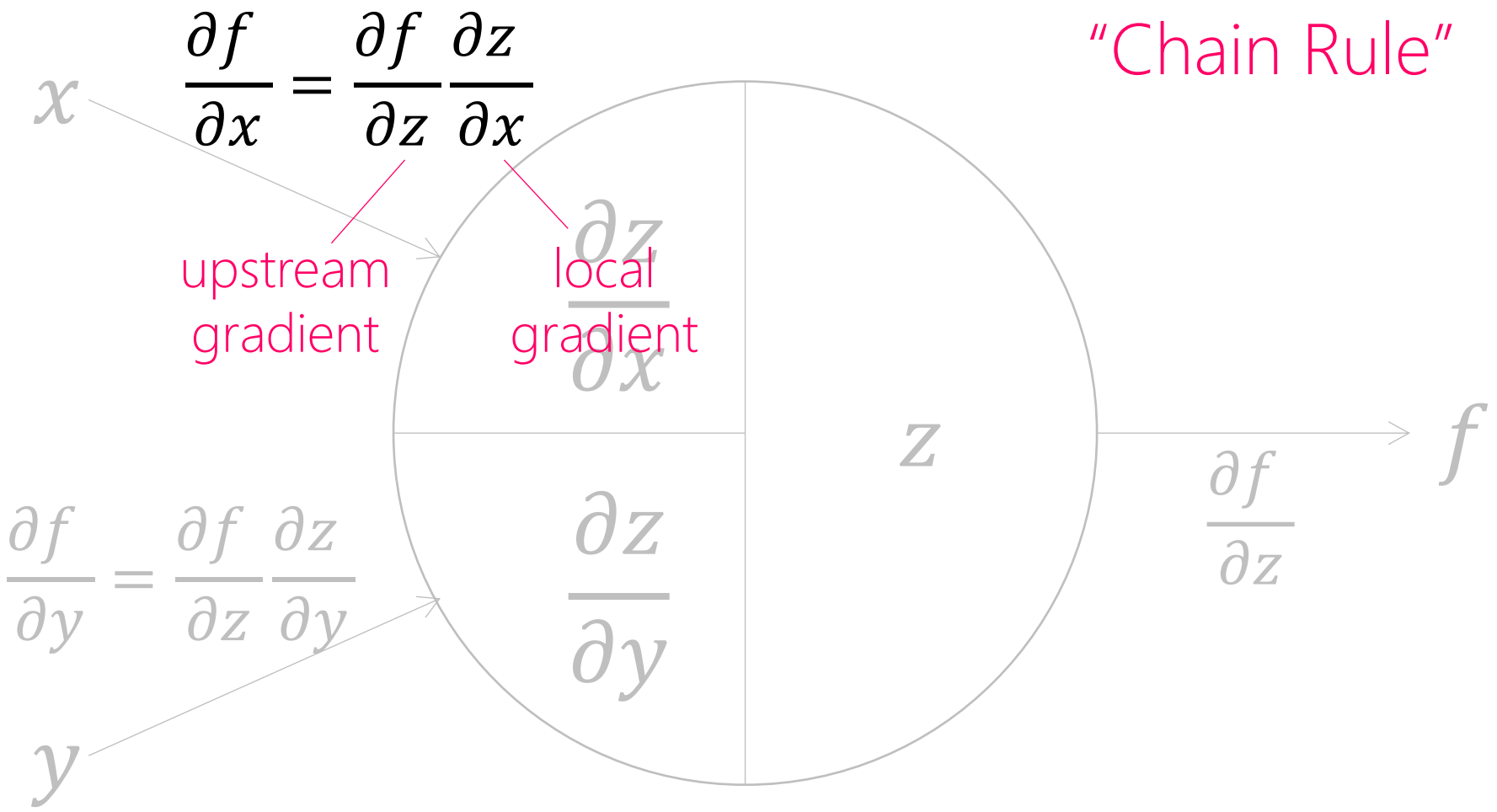
$f = qz$ $\quad \frac{\partial f}{\partial q} = z$ $\quad \frac{\partial f}{\partial z} = q$

What we want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

Backpropagation



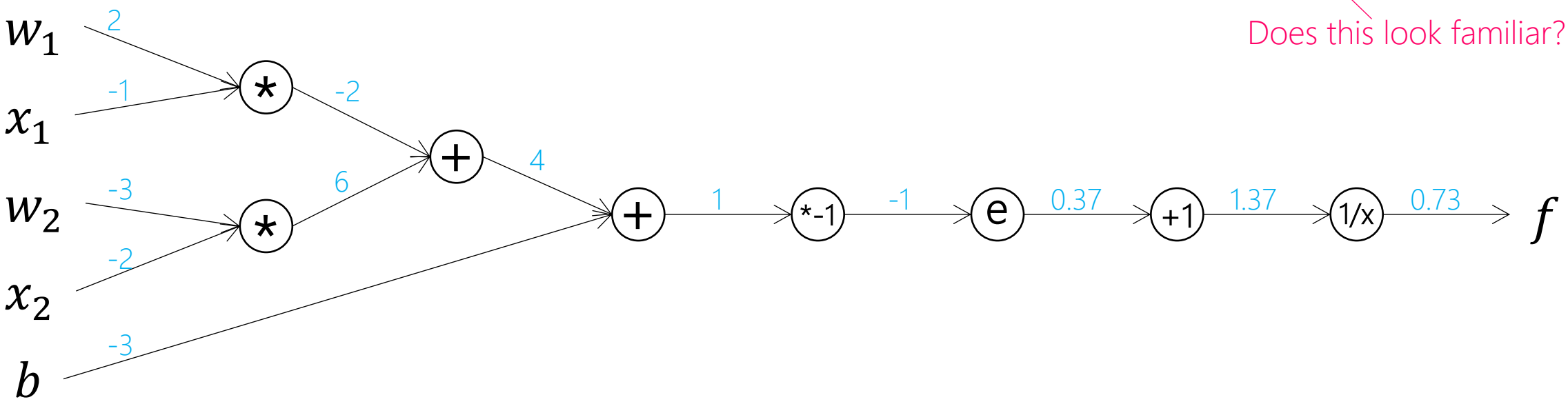
Backpropagation



Another Example

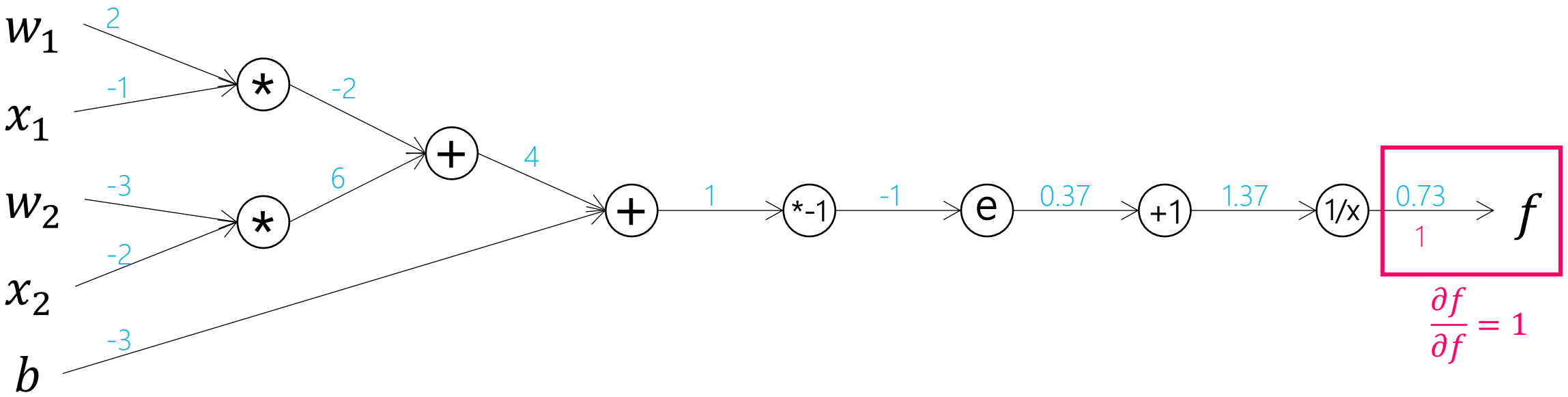
$$f(w,x) = \frac{1}{1 + e^{-(w_1 x_1 + w_2 x_2 + b)}}$$

Does this look familiar?



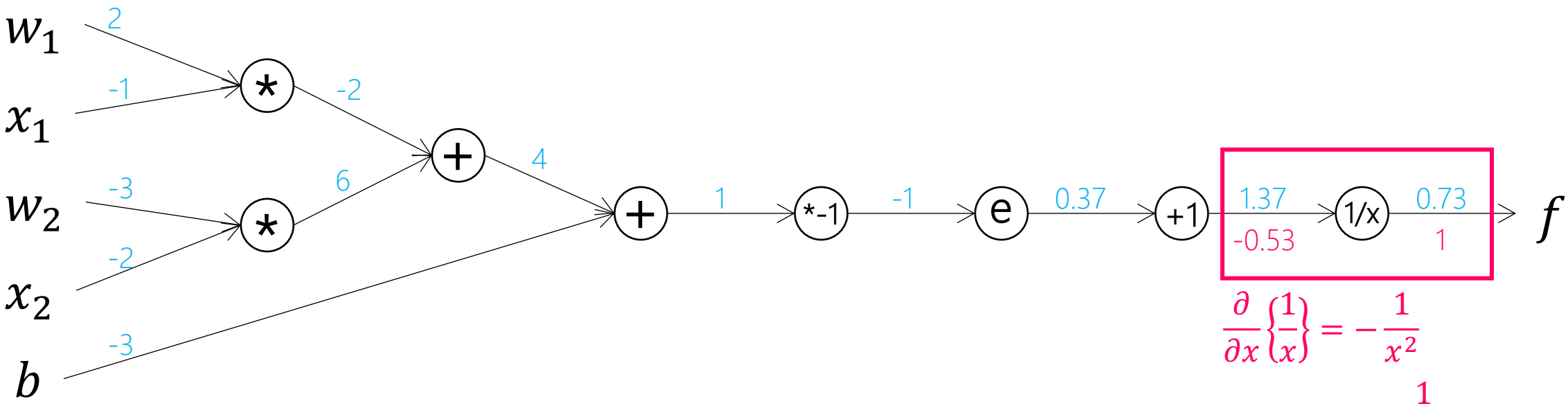
Another Example

$$f(w,x) = \frac{1}{1 + e^{-(w_1 x_1 + w_2 x_2 + b)}}$$



Another Example

$$f(w,x) = \frac{1}{1 + e^{-(w_1 x_1 + w_2 x_2 + b)}}$$



$$\frac{\partial}{\partial x}\left\{\frac{1}{x}\right\} = -\frac{1}{x^2}$$

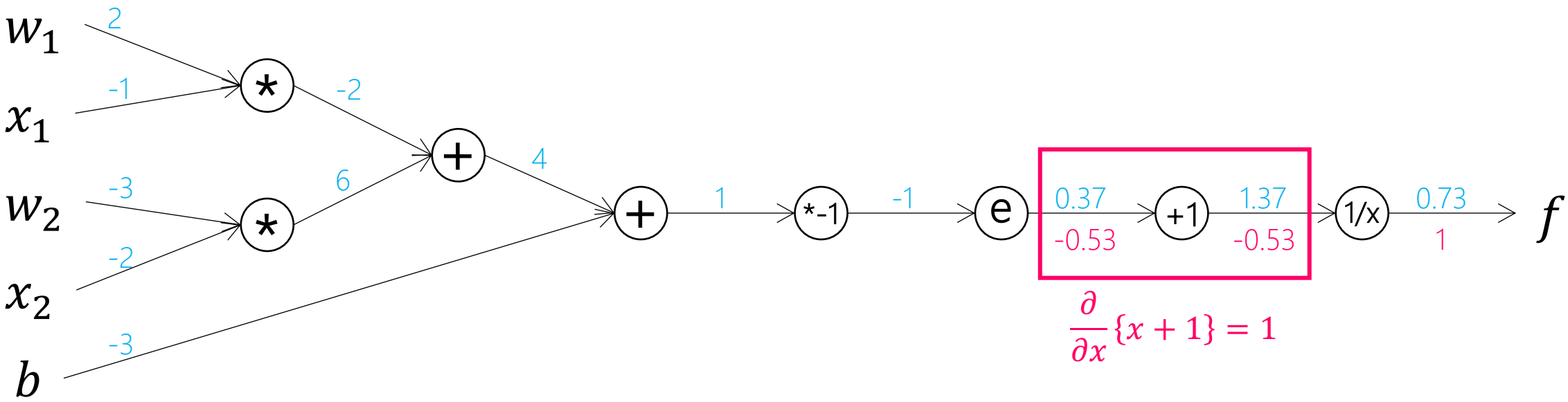
$$= -\frac{1}{1.37^2} = -0.53$$

Cheat Sheet:

$$\frac{\partial}{\partial x}\left\{\frac{1}{x}\right\} = -\frac{1}{x^2}$$

Another Example

$$f(w,x) = \frac{1}{1 + e^{-(w_1 x_1 + w_2 x_2 + b)}}$$

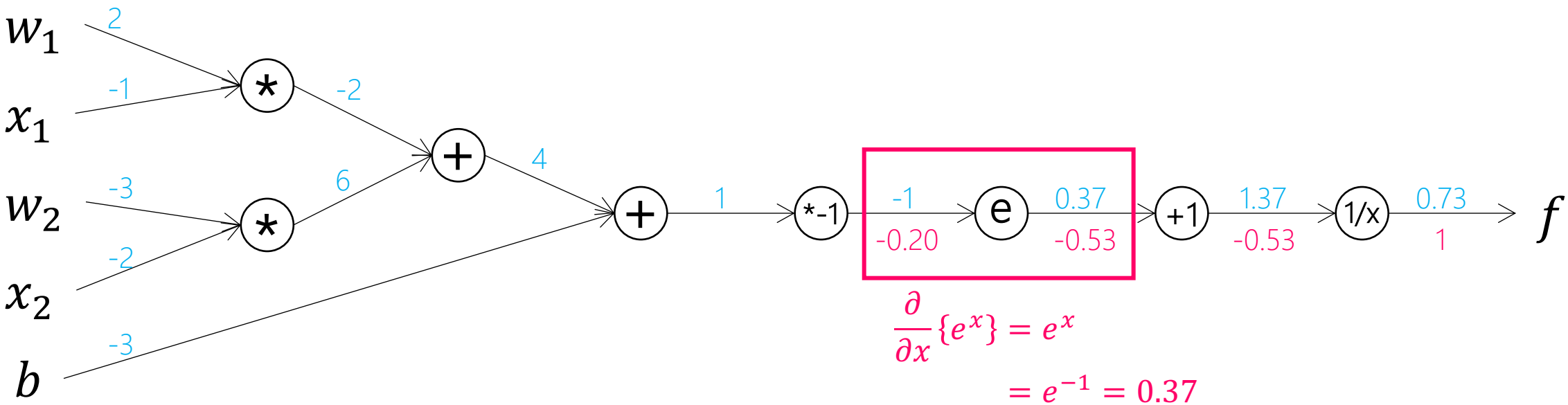


Cheat Sheet:

$$\frac{\partial}{\partial x}\left\{\frac{1}{x}\right\} = -\frac{1}{x^2}$$

Another Example

$$f(w,x) = \frac{1}{1 + e^{-(w_1 x_1 + w_2 x_2 + b)}}$$



$$\frac{\partial}{\partial x}\{e^x\} = e^x$$

$$= e^{-1} = 0.37$$

$$0.37 * (-0.53) \approx -0.20$$

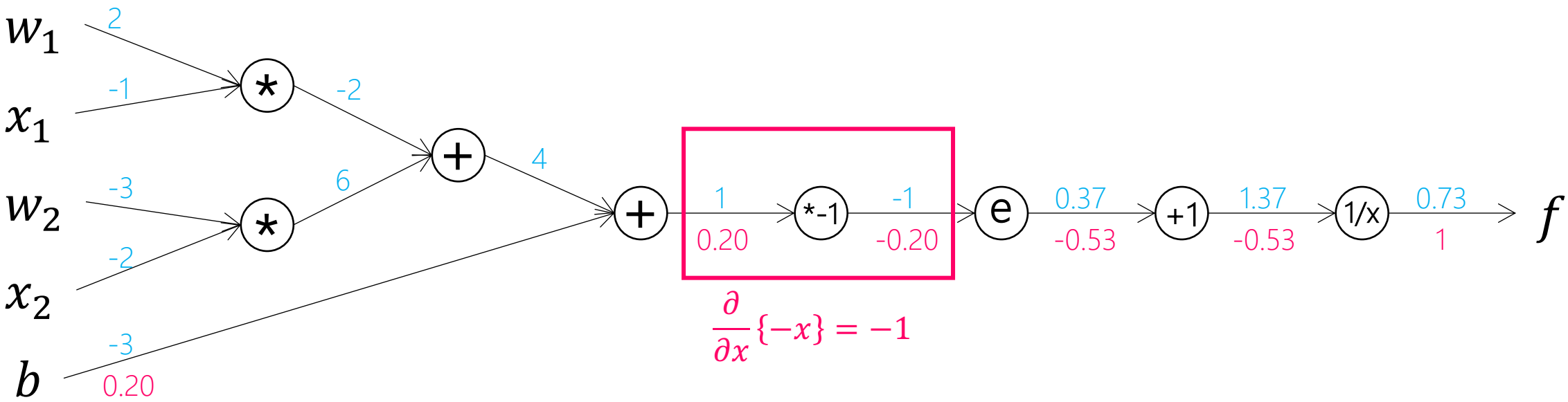
Cheat Sheet:

$$\frac{\partial}{\partial x}\left\{\frac{1}{x}\right\} = -\frac{1}{x^2}$$

$$\frac{\partial}{\partial x} e^x = e^x$$

Another Example

$$f(w,x) = \frac{1}{1 + e^{-(w_1 x_1 + w_2 x_2 + b)}}$$



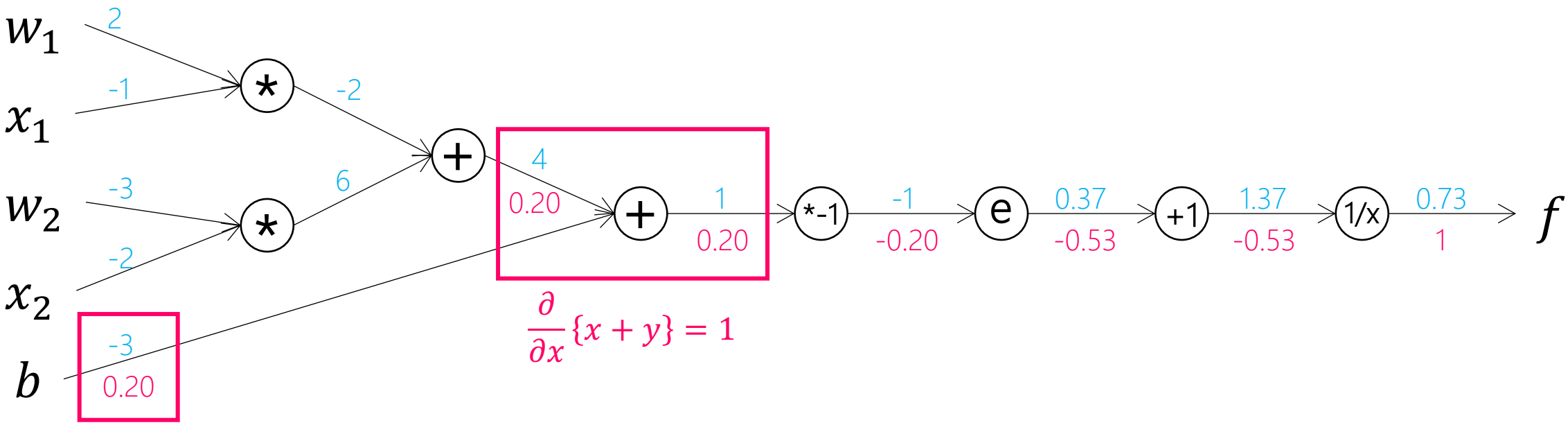
Cheat Sheet:

$$\frac{\partial}{\partial x}\left\{\frac{1}{x}\right\} = -\frac{1}{x^2}$$

$$\frac{\partial}{\partial x} e^x = e^x$$

Another Example

$$f(w,x) = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$



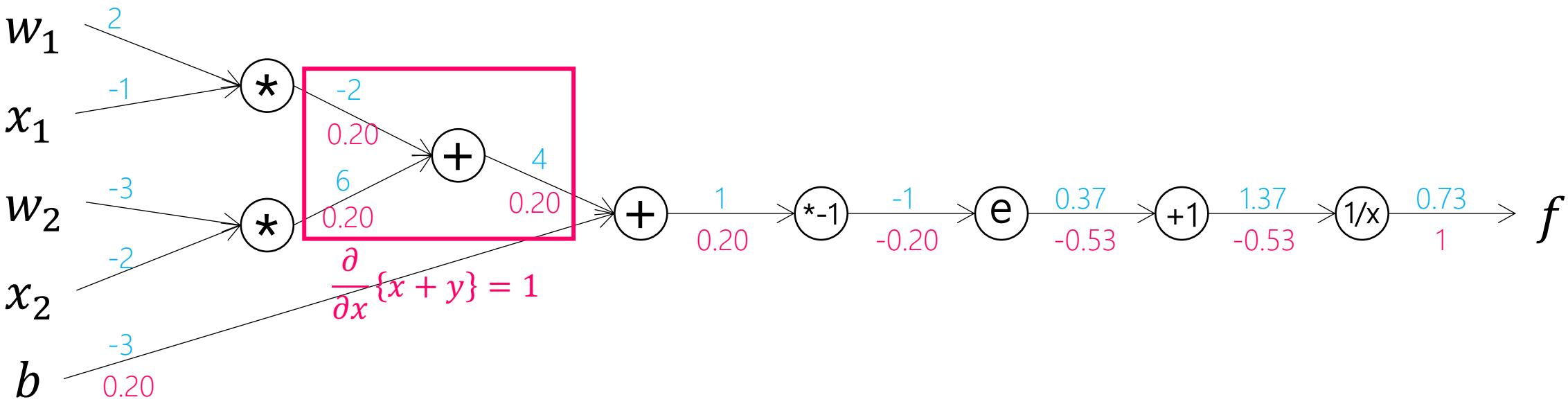
Cheat Sheet:

$$\frac{\partial}{\partial x}\left\{\frac{1}{x}\right\} = -\frac{1}{x^2}$$

$$\frac{\partial}{\partial x}e^x = e^x$$

Another Example

$$f(w,x) = \frac{1}{1 + e^{-(w_1 x_1 + w_2 x_2 + b)}}$$



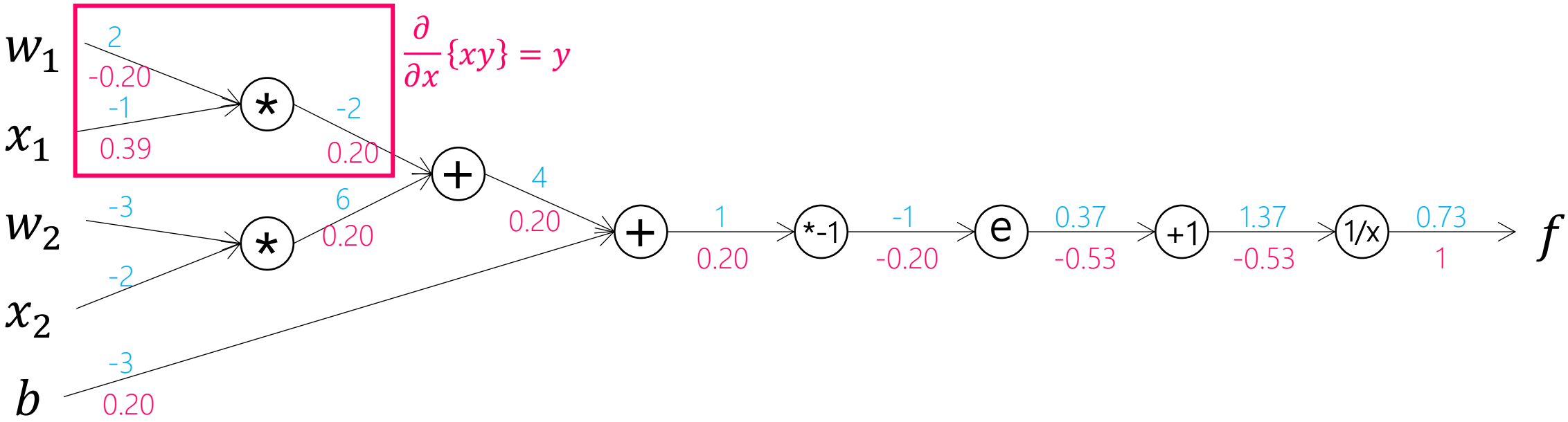
Cheat Sheet:

$$\frac{\partial}{\partial x}\left\{\frac{1}{x}\right\} = -\frac{1}{x^2}$$

$$\frac{\partial}{\partial x} e^x = e^x$$

Another Example

$$f(w,x) = \frac{1}{1 + e^{-(w_1 x_1 + w_2 x_2 + b)}}$$



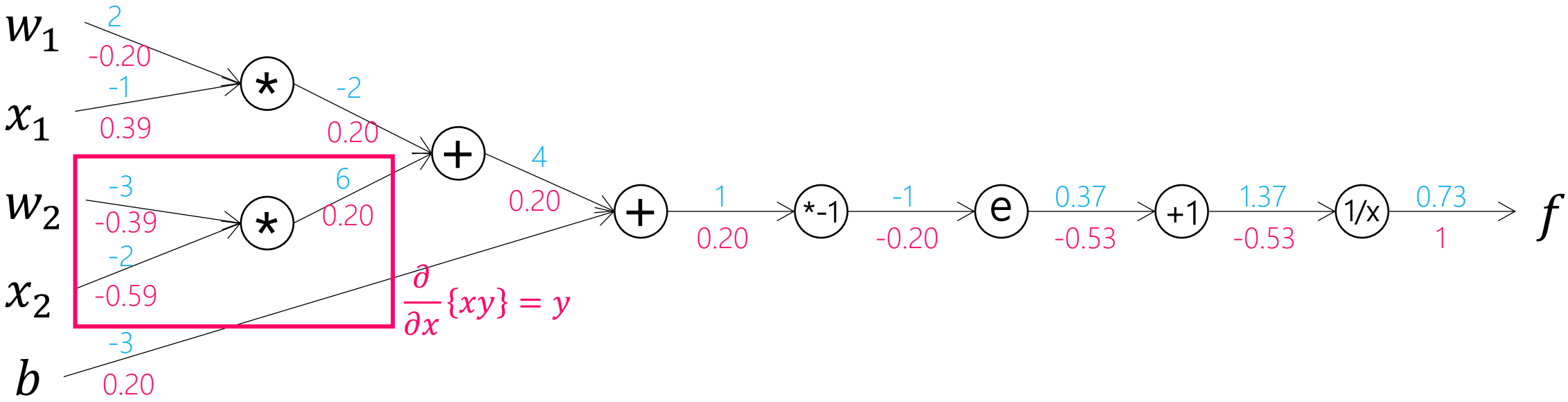
Cheat Sheet:

$$\frac{\partial}{\partial x}\left\{\frac{1}{x}\right\} = -\frac{1}{x^2}$$

$$\frac{\partial}{\partial x} e^x = e^x$$

Another Example

$$f(w,x) = \frac{1}{1 + e^{-(w_1 x_1 + w_2 x_2 + b)}}$$



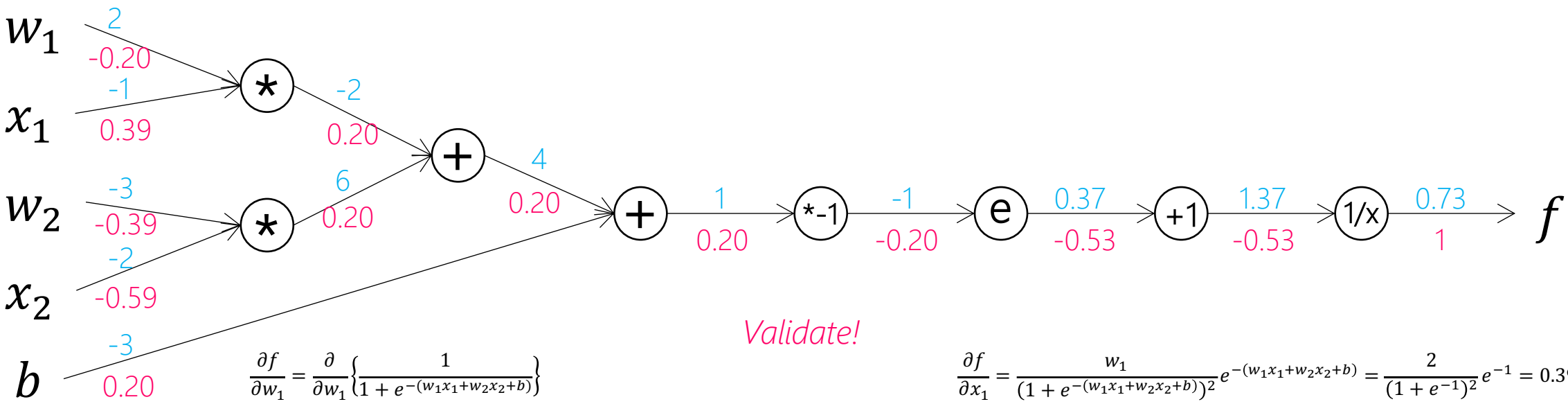
Cheat Sheet:

$$\frac{\partial}{\partial x}\left\{\frac{1}{x}\right\} = -\frac{1}{x^2}$$

$$\frac{\partial}{\partial x} e^x = e^x$$

Another Example

$$f(w,x) = \frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}$$



Validate!

Cheat Sheet:

$$\frac{\partial}{\partial x}\left\{\frac{1}{x}\right\} = -\frac{1}{x^2}$$

$$\frac{\partial}{\partial x}e^x = e^x$$

$$\frac{\partial f}{\partial w_1} = \frac{\partial}{\partial w_1}\left\{\frac{1}{1 + e^{-(w_1x_1 + w_2x_2 + b)}}\right\}$$

$$= -\frac{1}{(1 + e^{-(w_1x_1 + w_2x_2 + b)})^2}\frac{\partial}{\partial w_1}\left\{1 + e^{-(w_1x_1 + w_2x_2 + b)}\right\}$$

$$= -\frac{1}{(1 + e^{-(w_1x_1 + w_2x_2 + b)})^2}e^{-(w_1x_1 + w_2x_2 + b)}\frac{\partial}{\partial w_1}\{-(w_1x_1 + w_2x_2 + b)\}$$

$$= \frac{x_1}{(1 + e^{-(w_1x_1 + w_2x_2 + b)})^2}e^{-(w_1x_1 + w_2x_2 + b)}$$

$$= \frac{-1}{(1 + e^{-(-2+6-3)})^2}e^{-(-2+6-3)} = -0.1966$$

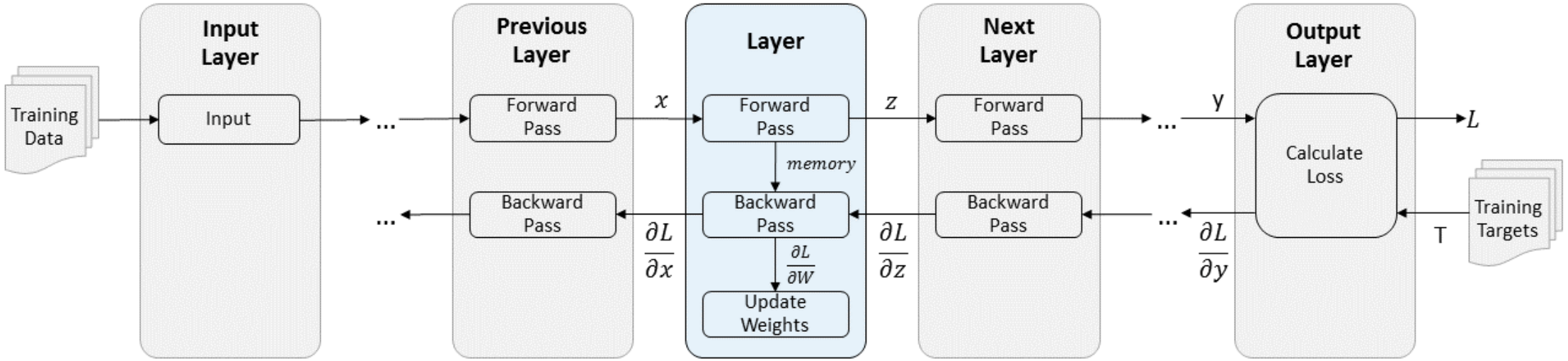
$$\frac{\partial f}{\partial x_1} = \frac{w_1}{(1 + e^{-(w_1x_1 + w_2x_2 + b)})^2}e^{-(w_1x_1 + w_2x_2 + b)} = \frac{2}{(1 + e^{-1})^2}e^{-1} = 0.3932$$

$$\frac{\partial f}{\partial w_2} = \frac{x_2}{(1 + e^{-(w_1x_1 + w_2x_2 + b)})^2}e^{-(w_1x_1 + w_2x_2 + b)} = \frac{-2}{(1 + e^{-1})^2}e^{-1} = -0.3932$$

$$\frac{\partial f}{\partial x_2} = \frac{w_2}{(1 + e^{-(w_1x_1 + w_2x_2 + b)})^2}e^{-(w_1x_1 + w_2x_2 + b)} = \frac{-3}{(1 + e^{-1})^2}e^{-1} = -0.5898$$

$$\frac{\partial f}{\partial b} = \frac{1}{(1 + e^{-(w_1x_1 + w_2x_2 + b)})^2}e^{-(w_1x_1 + w_2x_2 + b)} = \frac{1}{(1 + e^{-1})^2}e^{-1} = 0.1966$$

Putting them all together



https://www.mathworks.com/help/nnet/ug/define-custom-deep-learning-layers.html

Putting them all together

- PyTorch/TensorFlow/Keras does auto-differentiation
  - No need to define "backward" routine.

AUTOMATIC DIFFERENTIATION WITH `TORCH.AUTOGRAD`

When training neural networks, the most frequently used algorithm is **back propagation**. In this algorithm, parameters (model weights) are adjusted according to the **gradient** of the loss function with respect to the given parameter.

To compute those gradients, PyTorch has a built-in differentiation engine called `torch.autograd`. It supports automatic computation of gradient for any computational graph.

Consider the simplest one-layer neural network, with input `x`, parameters `w` and `b`, and some loss function. It can be defined in PyTorch in the following manner:

```
import torch

x = torch.ones(5)  # input tensor
y = torch.zeros(3)  # expected output
w = torch.randn(5, 3, requires_grad=True)
b = torch.randn(3, requires_grad=True)
z = torch.matmul(x, w)+b
loss = torch.nn.functional.binary_cross_entropy_with_logits(z, y)
```

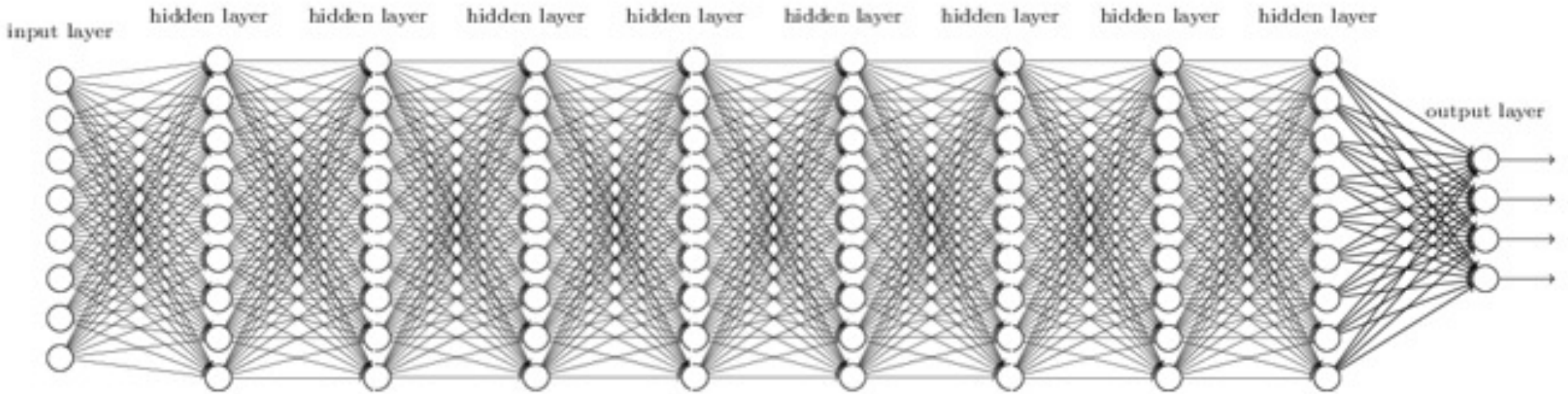
https://pytorch.org/tutorials/beginner/basics/autogradqs_tutorial.html

Putting them all together

- Linear model: $f = Wx + b$
- Neural network with a single hidden layer: $f = W_2 \max(0, W_1 x + b_1) + b_2$
- Neural network with two hidden layers:

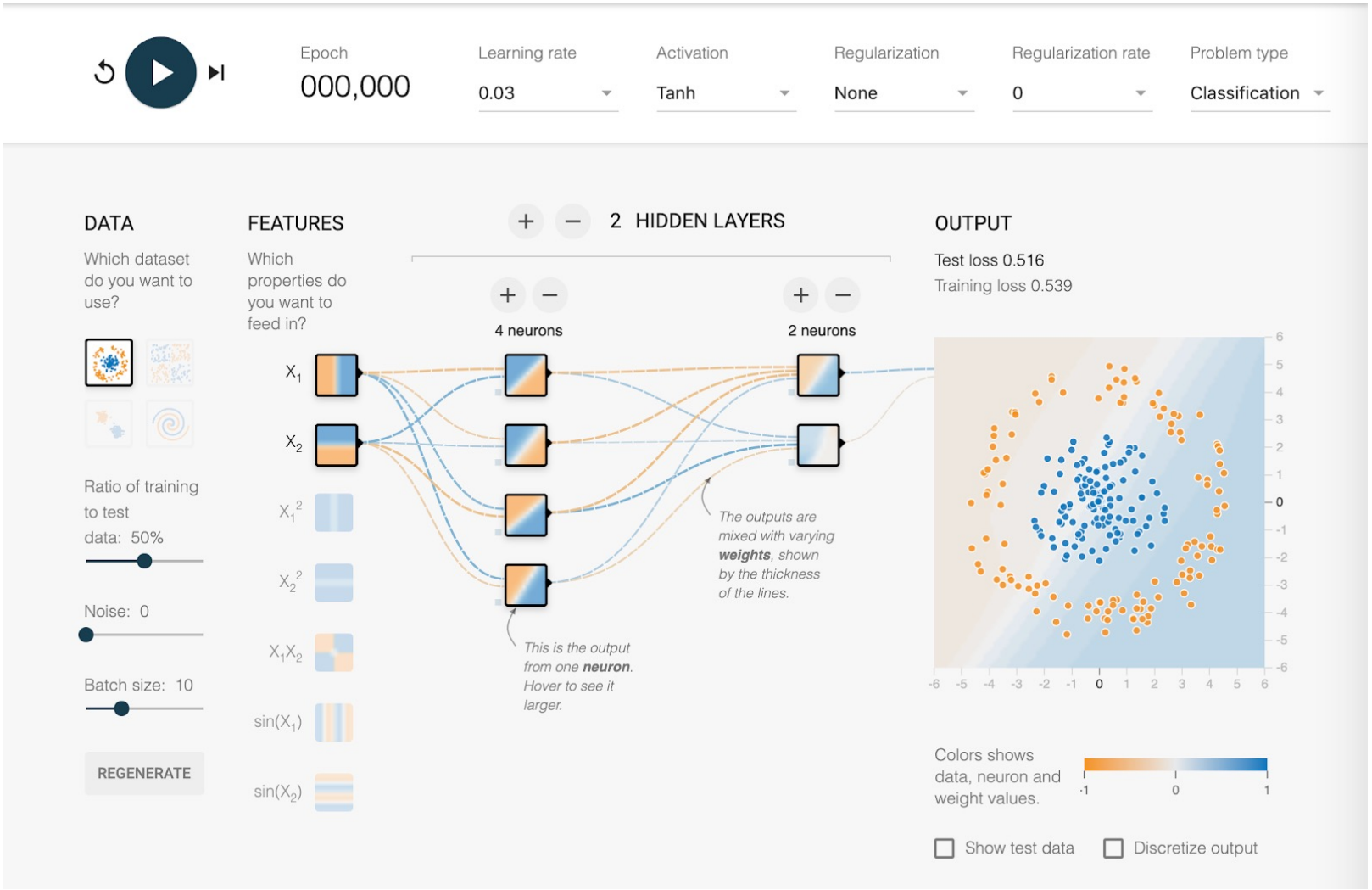
$$f = W_3 \max(0, W_2 \max(0, W_1 x + b_1) + b_2) + b_3$$

- … stacking up hidden layers → "deep" neural networks



Deep Neural Network

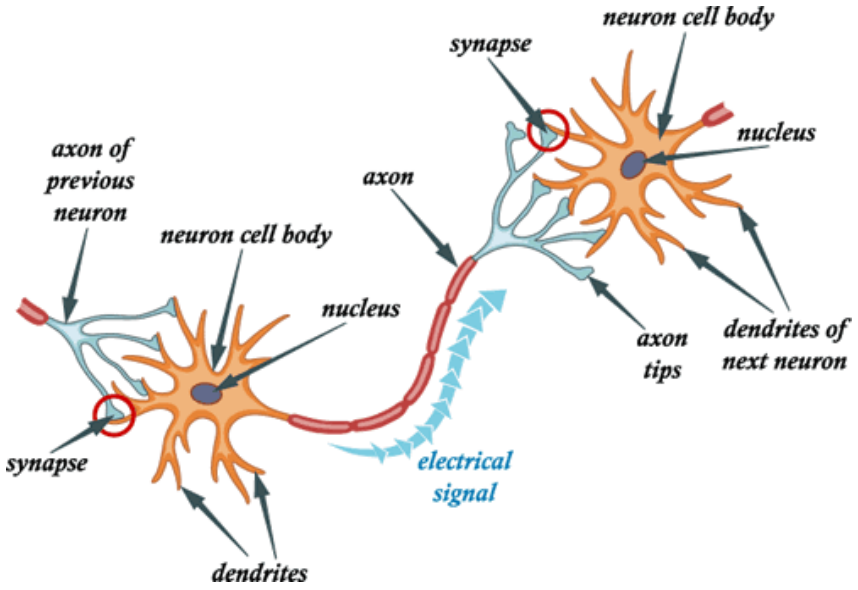
A Neural Network Playground

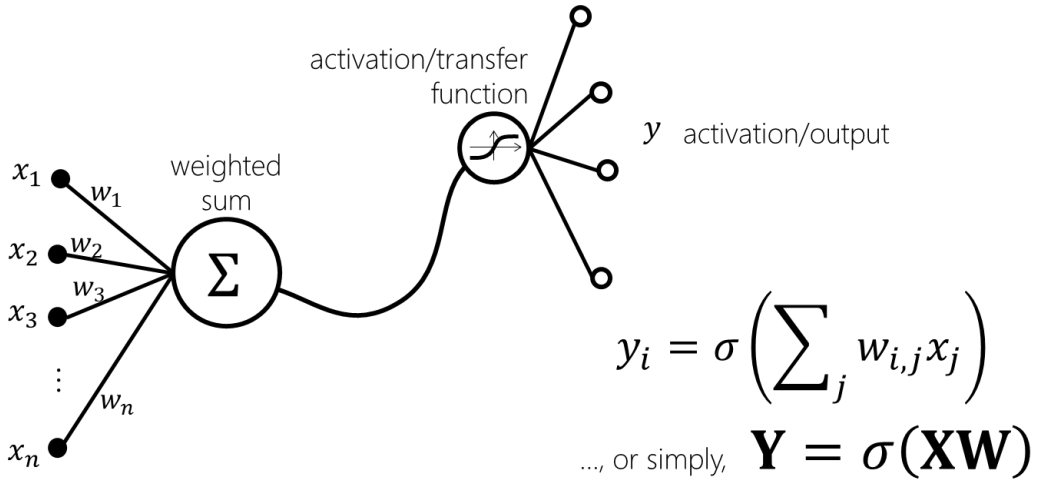


https://playground.tensorflow.org

Be careful with your brain analogies!

- Biological Neurons [Dendritic Computation. London and Hausser]:
  - Many different types
  - Dendrites can perform complex non-linear computations
  - Synapses are not a single weight but a complex non-linear dynamical system
  - Idea of interpreting activation function as firing rate may not be adequate

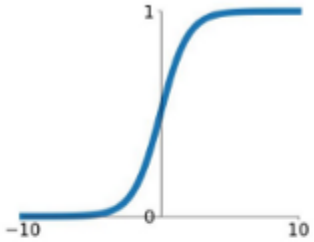




Activation Functions

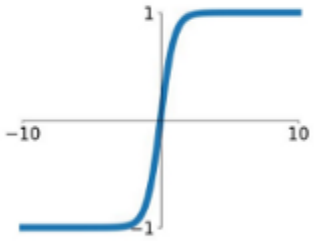
Sigmoid

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



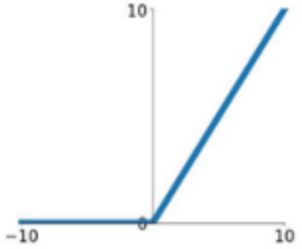
tanh

$$\tanh(x)$$



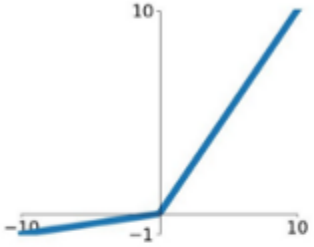
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

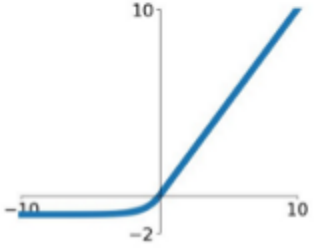


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



https://medium.com/@shrutijadon10104776/survey-on-activation-functions-for-deep-learning-9689331ba092

Appendix: Vector Norms and Matrix Norms

- Vector Norms
- Properties of Vector Norms
- Matrix Norms

Vector Norms

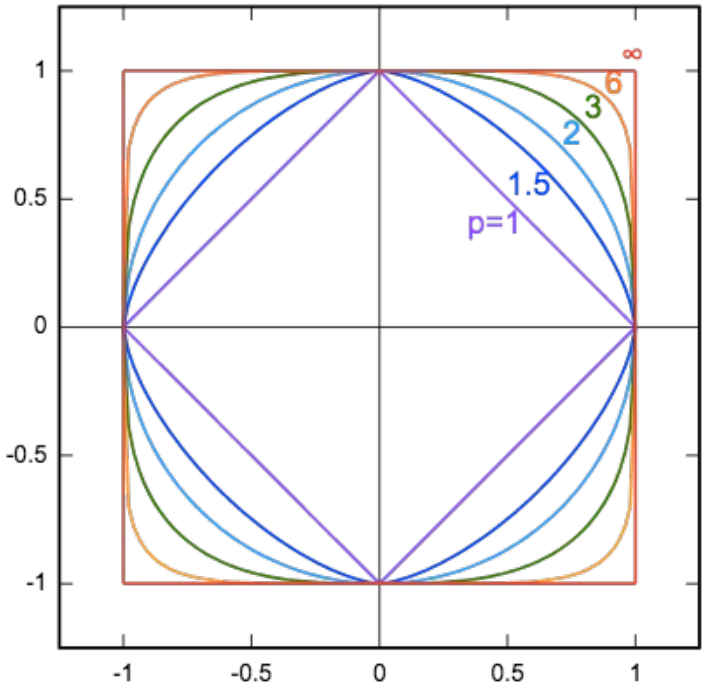
A function $f: x \in \mathbb{R}^n \rightarrow y \in \mathbb{R}_+$ is called a "norm", if the following three conditions are satisfied

- (Zero element) $f(x) \geq 0$ and $f(x) = 0$ iff $x = 0$
- (Homogeneous) For any $\alpha \in \mathbb{R}$ and $x \in \mathbb{R}^n$, $f(\alpha x) = |\alpha| f(x)$
- (Triangle inequality) Any $x, y \in \mathbb{R}^n$ satisfy $f(x) + f(y) \geq f(x + y)$

- l_2 norm (Euclidean norm): $\|x\|_2 = (|x_1|^2 + |x_2|^2 + \cdots + |x_n|^2)^{1/2}$
- Generally, l_p norm: $\|x\|_p = (|x_1|^p + |x_2|^p + \cdots + |x_n|^p)^{1/p}$
- l_∞ norm: $\max\{|x_1|, |x_2|, \cdots, |x_n|\}$

Properties of Vector Norms

- If $p \geq q$, we have $\|x\|_p \leq \|x\|_q$.
- $\|x\|_1 \geq \|x\|_2 \geq \|x\|_\infty$
- Unit circle (a circle with a radius of one) of different vector norms



Matrix Norms

- The Frobenius norm of a matrix $A \in \mathbb{R}^{m \times n}$ is defined as

$$\|A\|_F = \left(\sum_{i=1}^{m} \sum_{j=1}^{n} |A_{i,j}|^2 \right)^{1/2}$$

- Matrix trace norm (nuclear norm)

$$\|A\|_* = \sum_{i=1} \sigma_i = \text{trace}(\Sigma)$$

where $A = U\Sigma V^T$ is the SVD of matrix A.