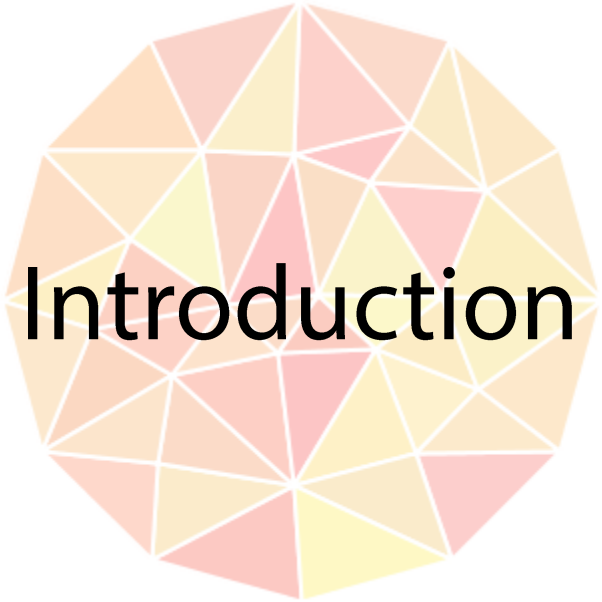
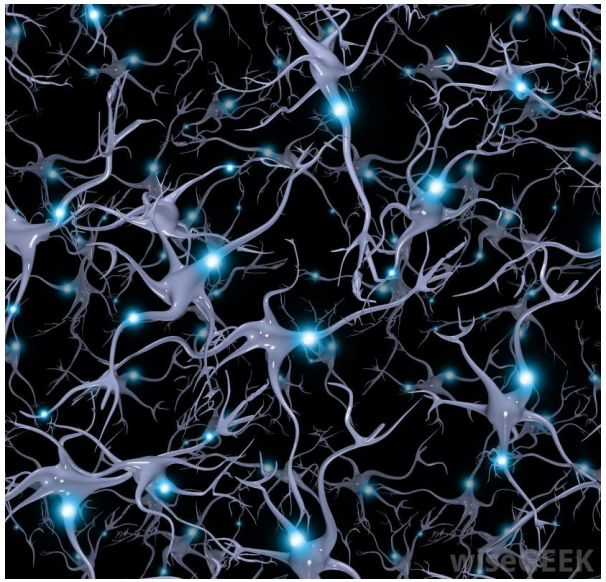


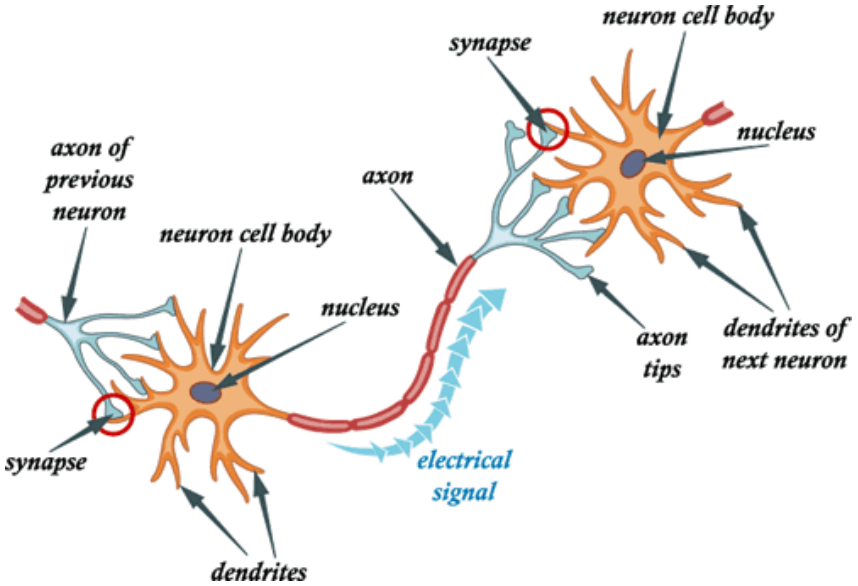
Introduction

Stephen Baek

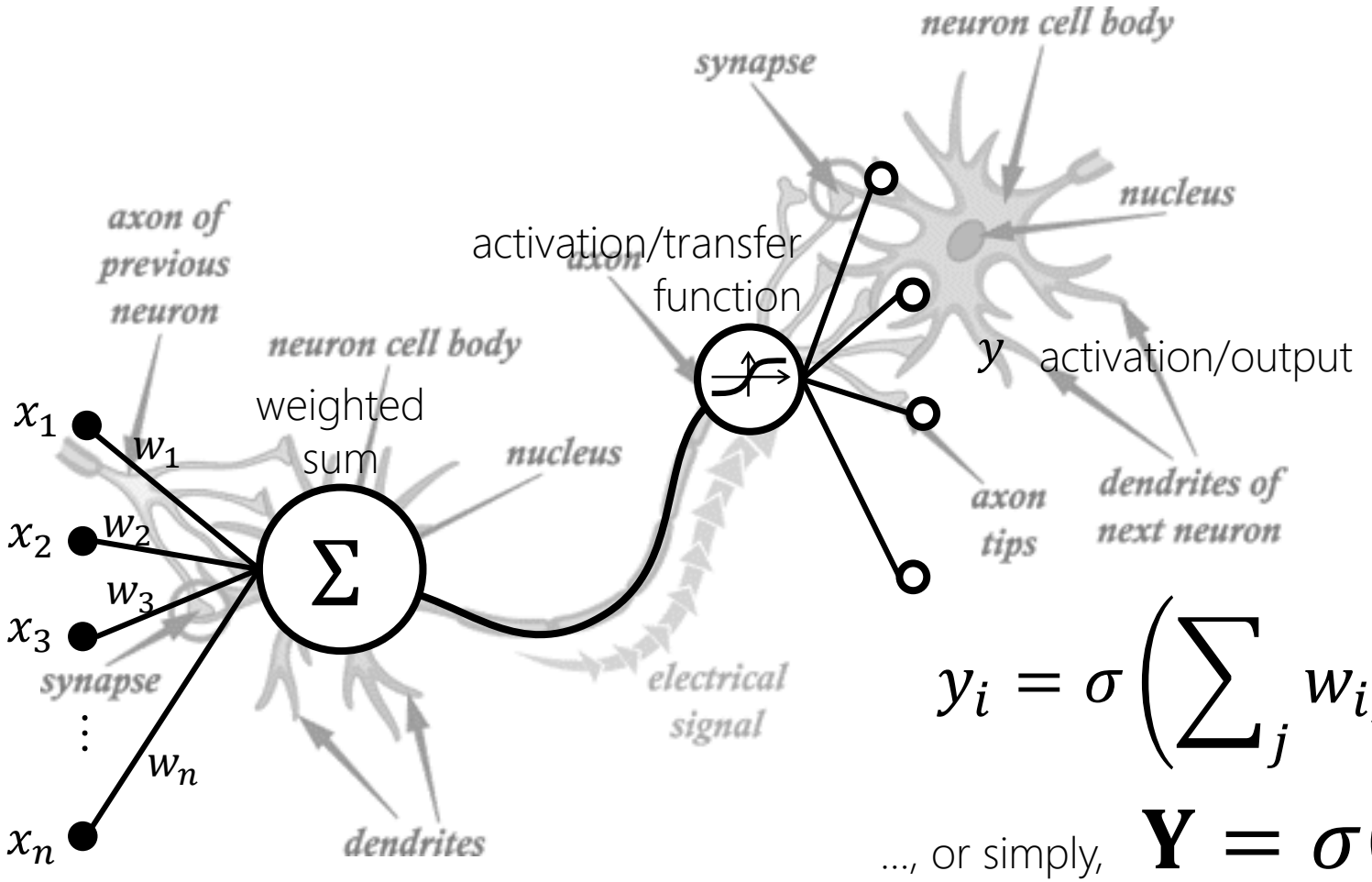
History of Neural Nets

Biological Neural Networks





Artificial Neural Networks (ANN)



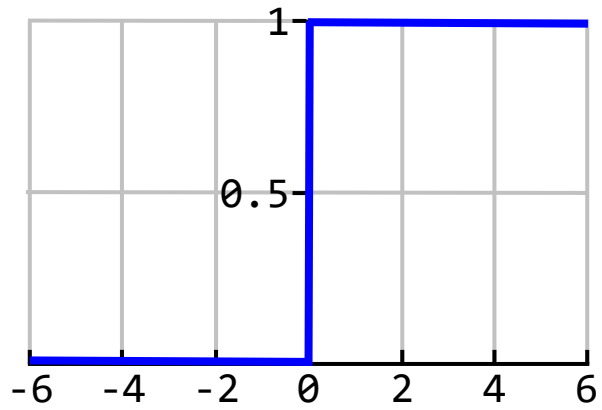
$$y_i = \sigma\left(\sum_j w_{i,j} x_j\right)$$

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

Activation/Transfer Functions

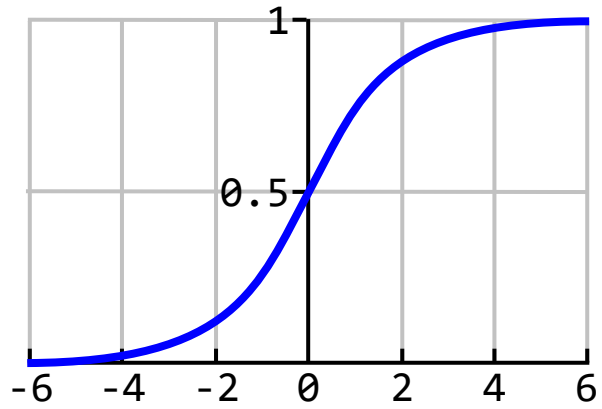
- "Threshold" in biological systems
- Step Function:

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



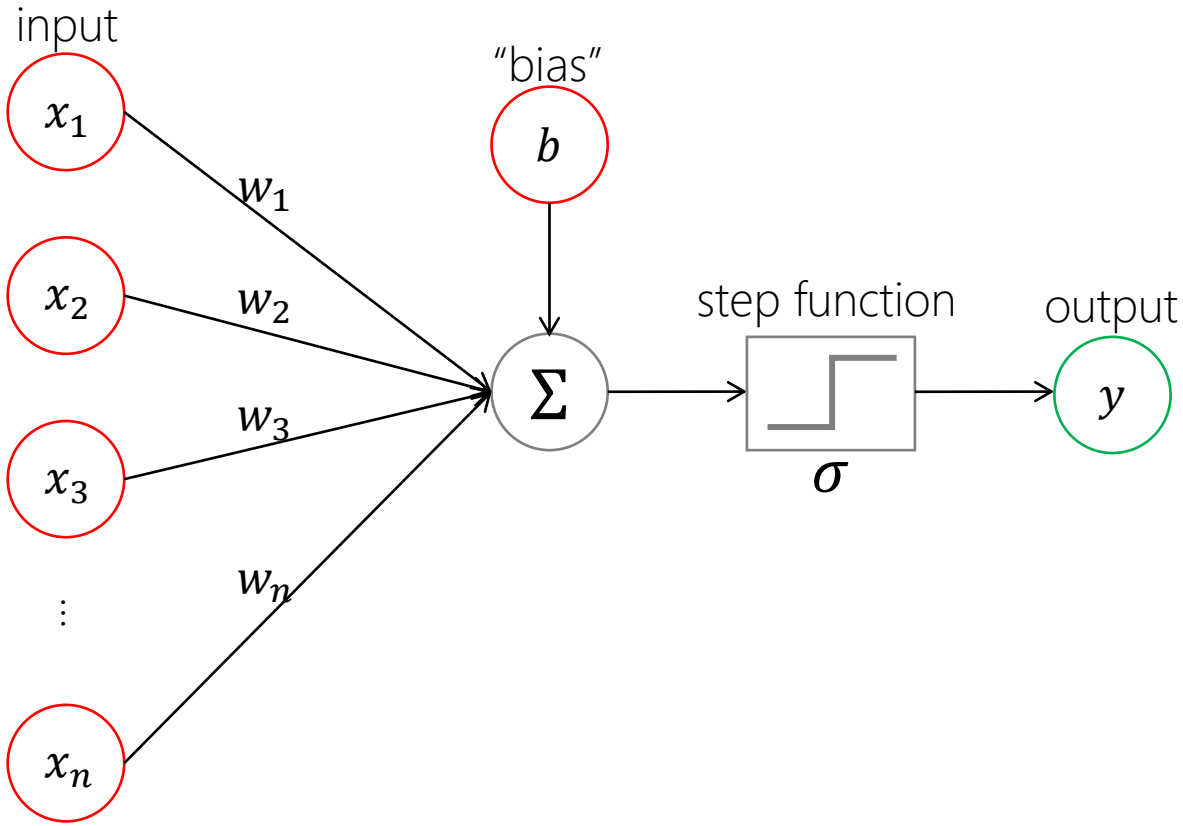
- Sigmoid Function:

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



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

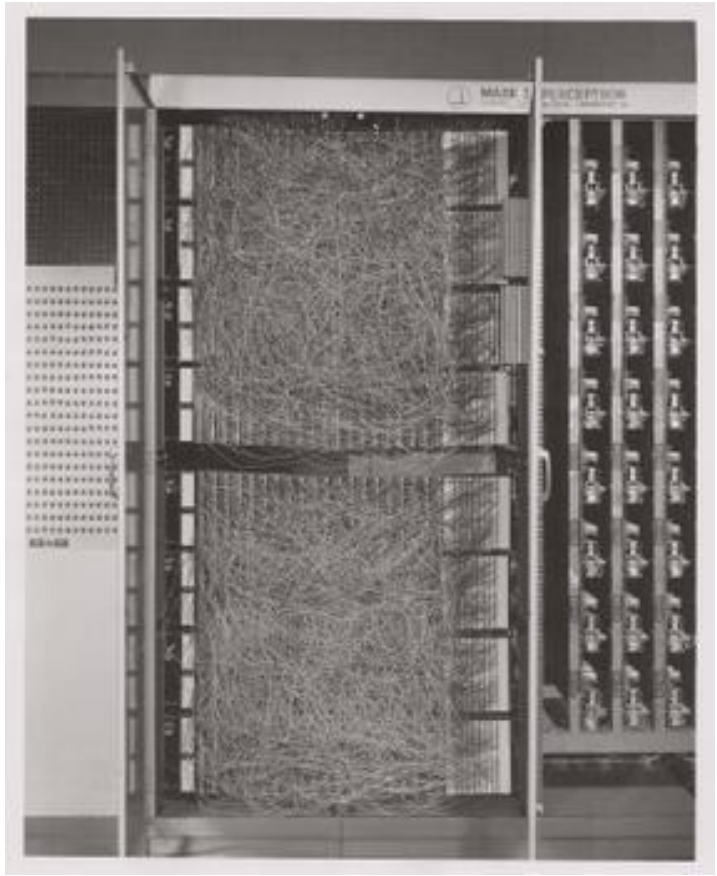
The simplest model: the “Perceptron”

- 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

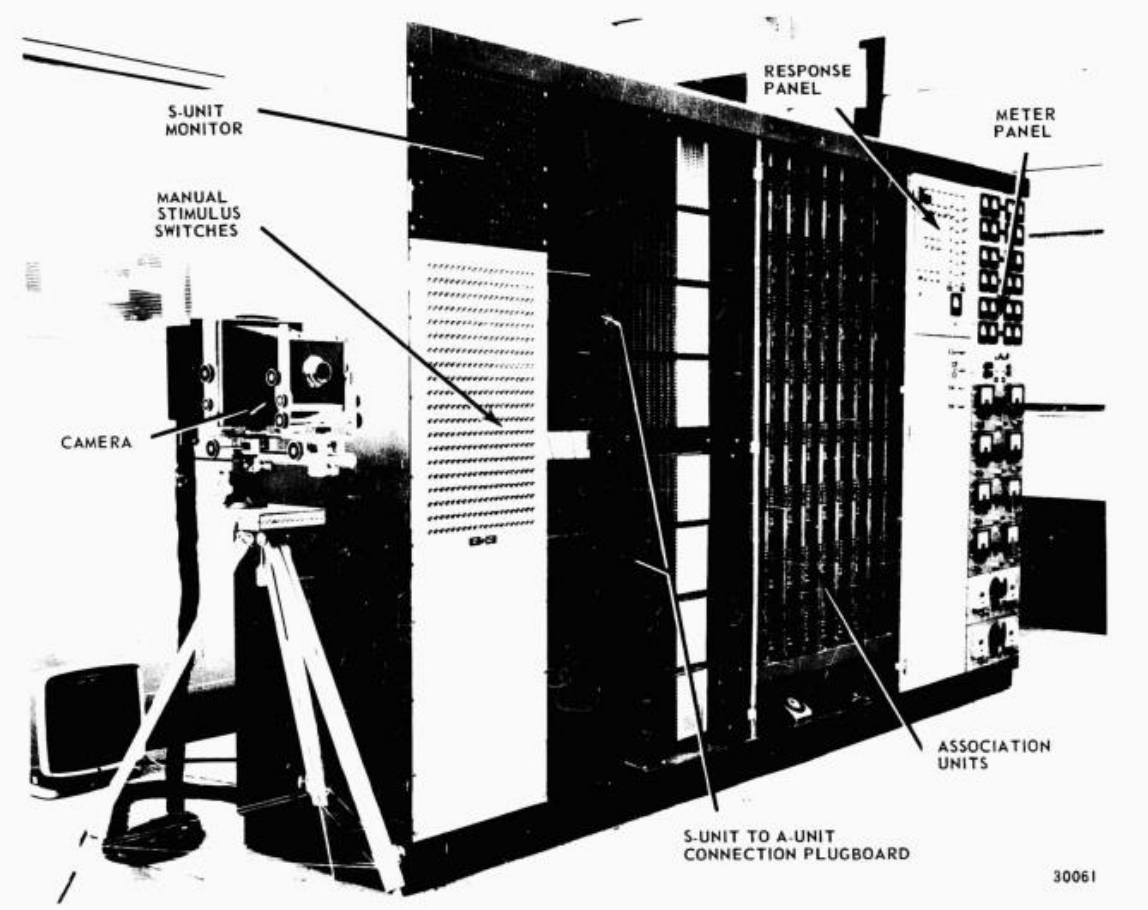
The simplest model: the “Perceptron”

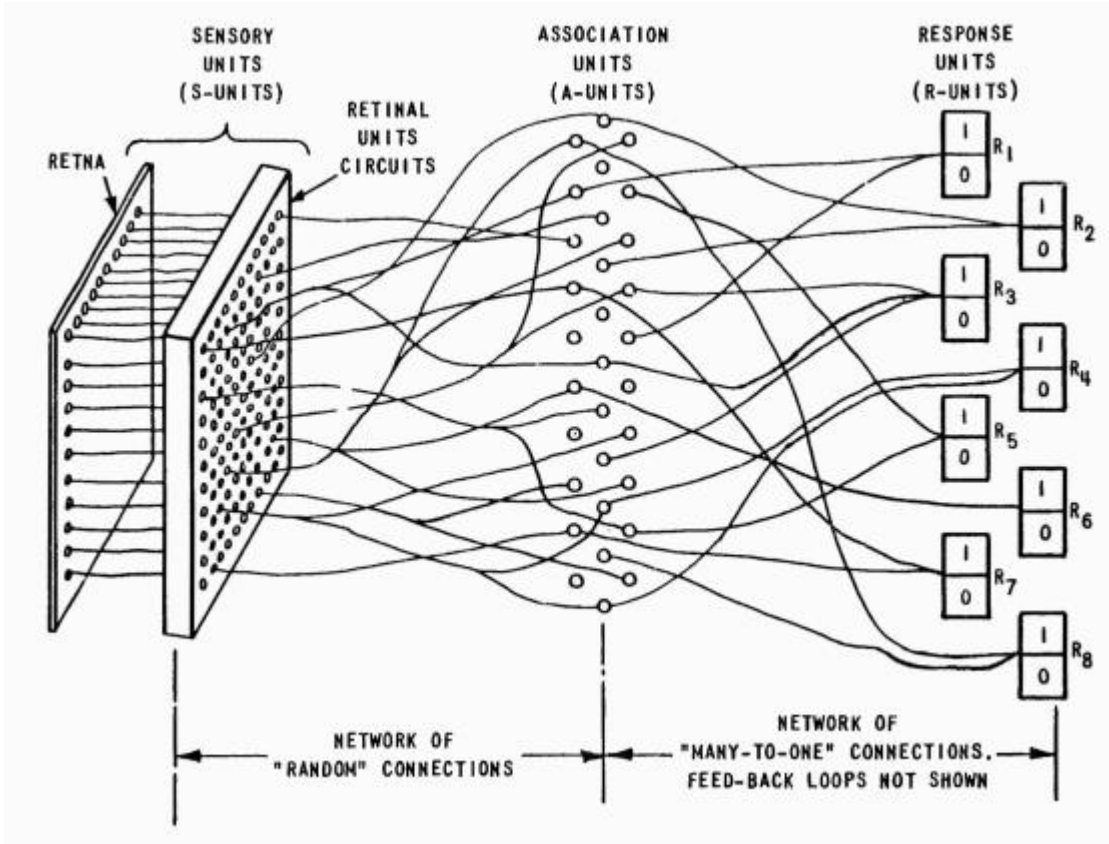
- Frank Rosenblatt (1957)
  - The **Mark I Perceptron** machine was the first implementation of the perceptron algorithm.
  - Designed for image recognition.
  - Connected to a camera that used a 20x20 cadmium sulfide photocell array to produce a 400-pixel image.
  - Weights were encoded in potentiometers.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



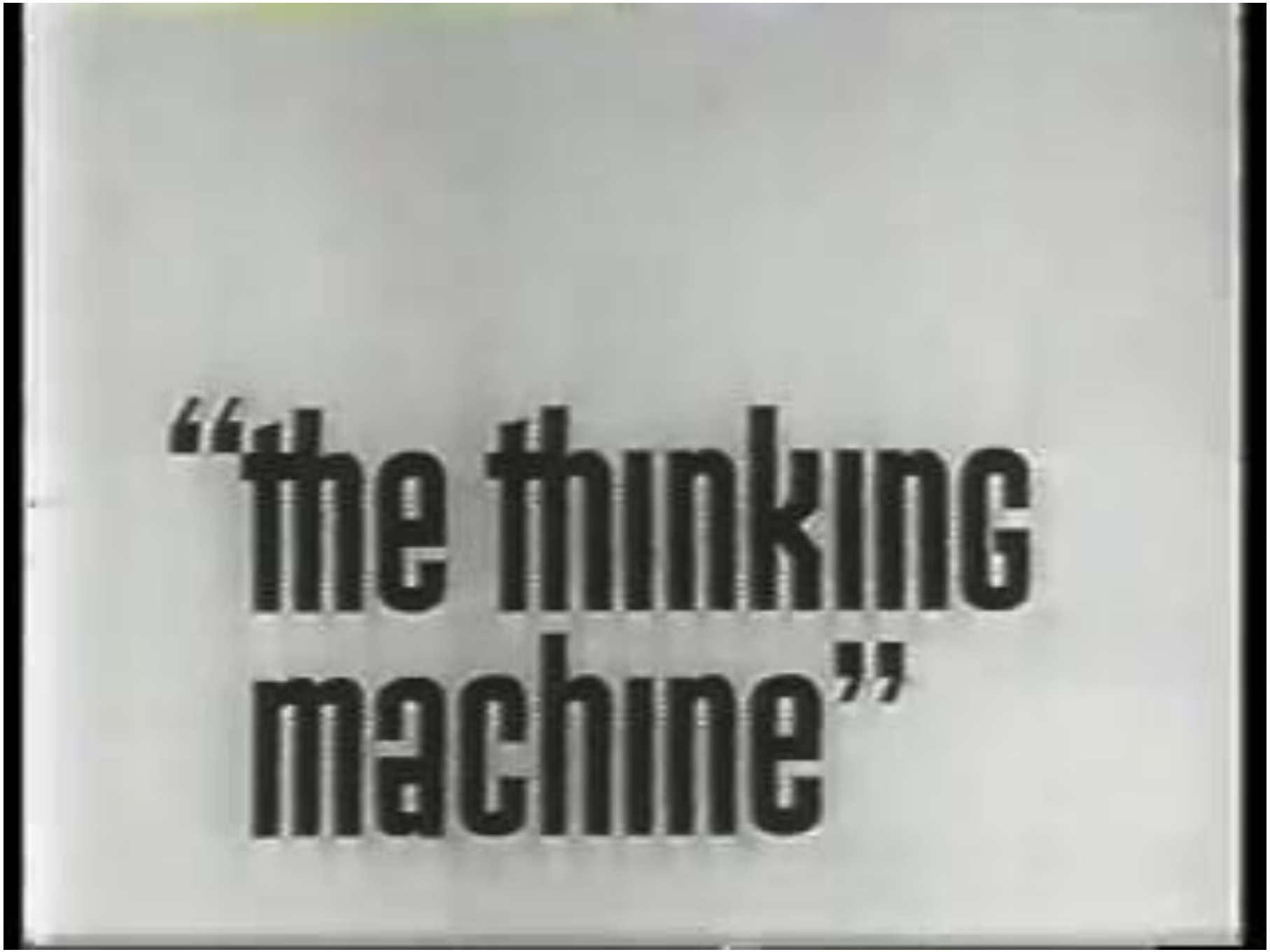
Wikipedia – Perceptron

The simplest model: the “Perceptron”





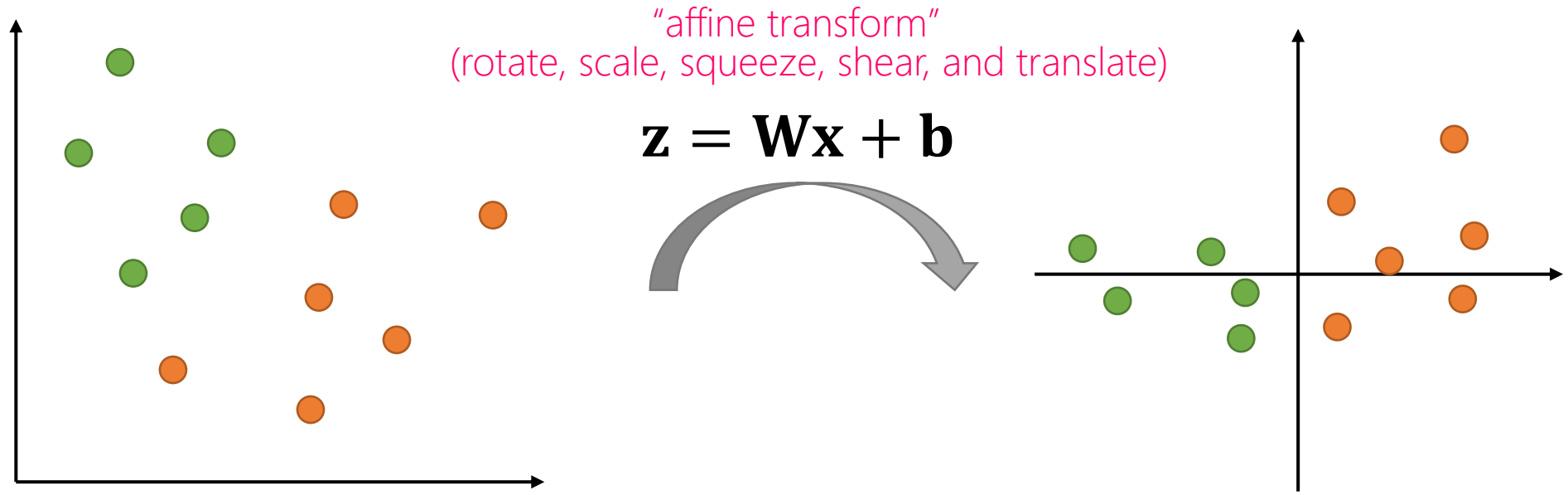
Mark 1 Perceptron Operators’ Manual (http://www.dtic.mil/dtic/tr/fulltext/u2/236965.pdf)



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

Perceptron as Coordinate Transformation

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



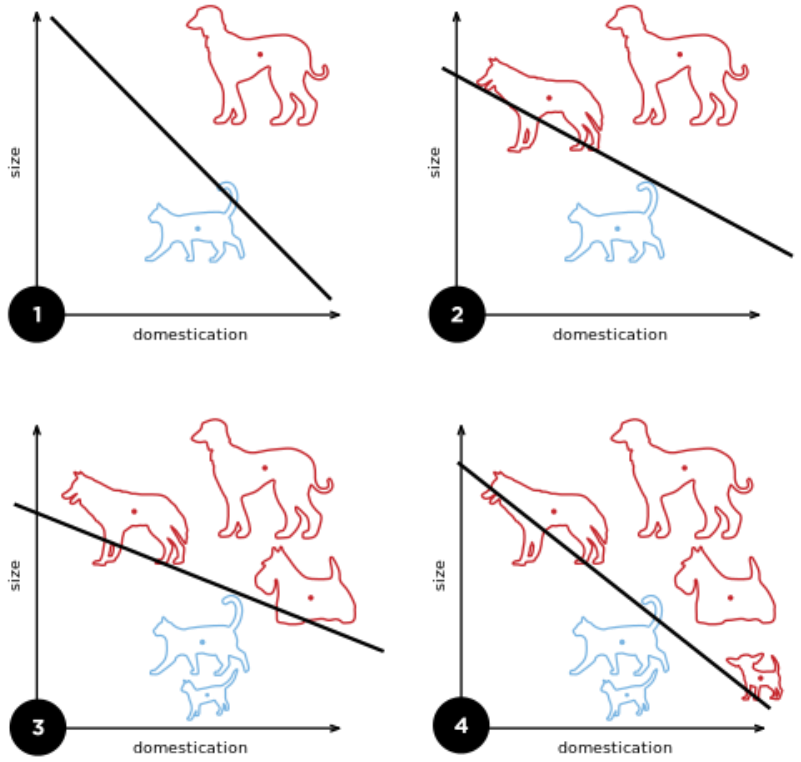
Perceptron as Linear Binary Classification

- Perceptron returns $\mathbf{1}$ iff. $\sum_j w_j x_j + b \geq \mathbf{0}$ and $\mathbf{0}$ 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

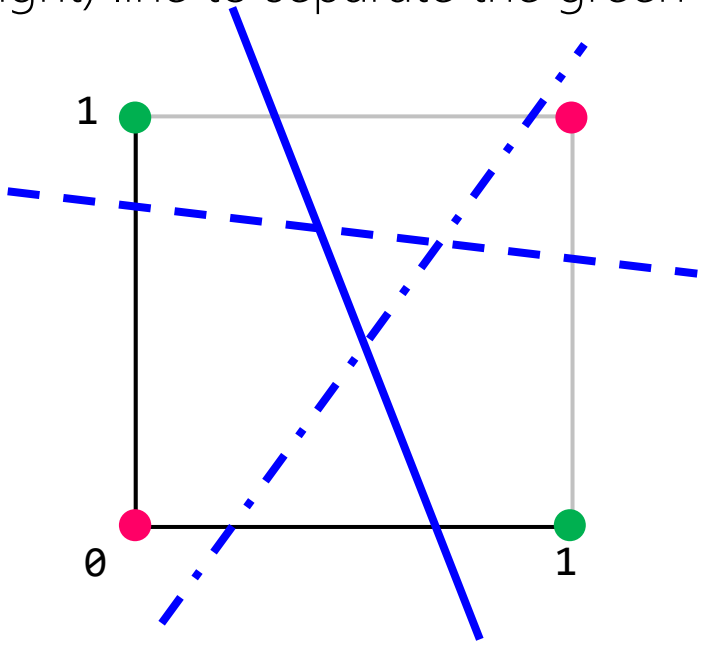
- XOR: Exclusive OR:

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?

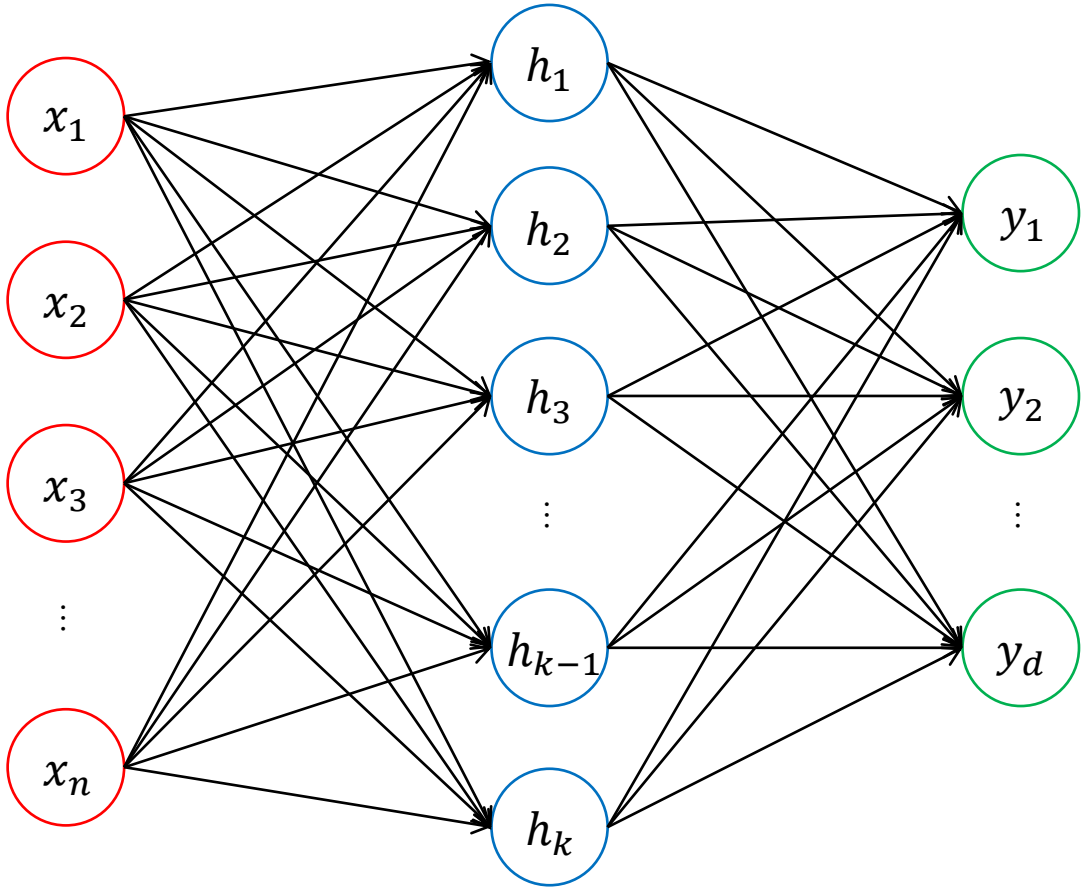


The XOR Problem

- 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)

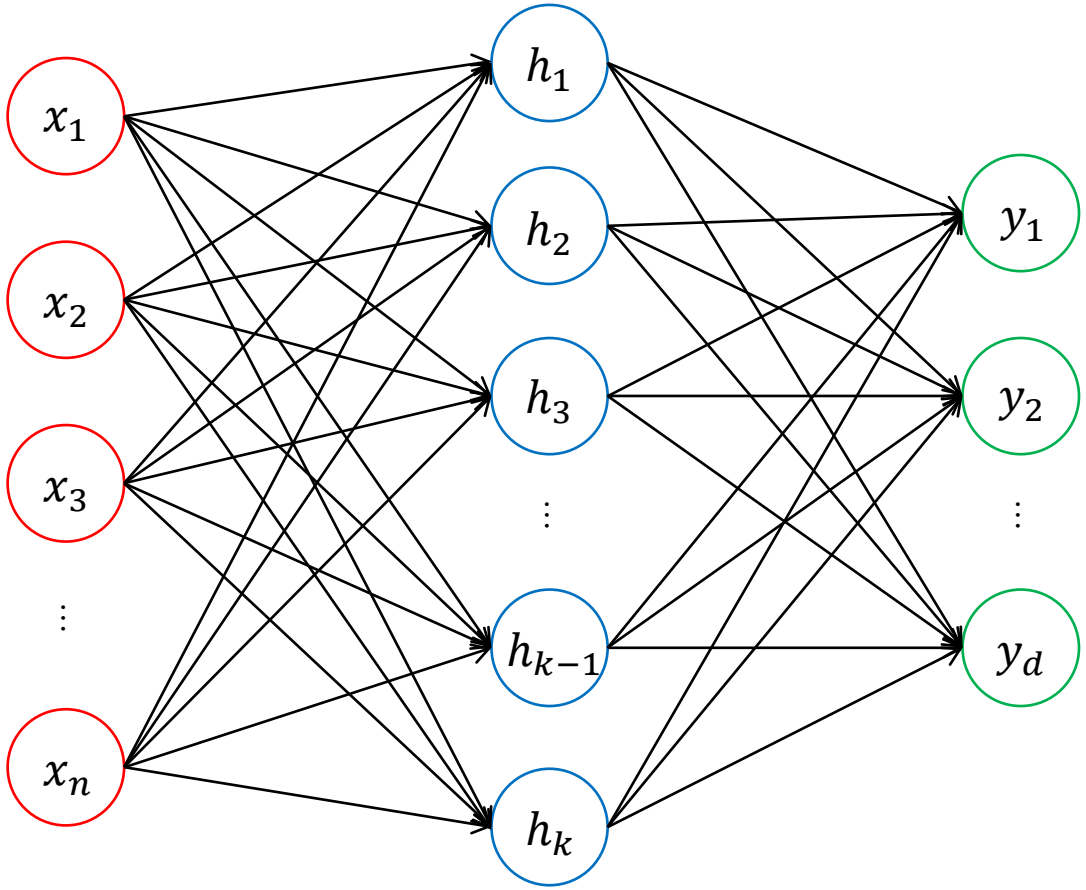


The perceptron has shown itself worthy of study despite (and even because of!) its severe limitations. It has many features to attract attention: its linearity; its intriguing learning theorem; its clear paradigmatic simplicity as a kind of parallel computation. There is no reason to suppose that any 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)



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

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

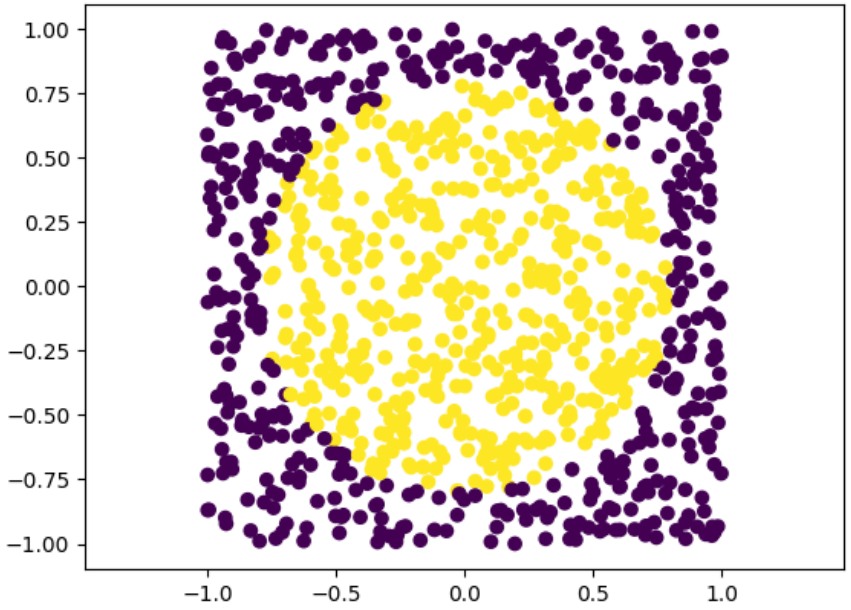
$$\frac{\partial \mathcal{L}}{\partial w} = ?$$

- 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)

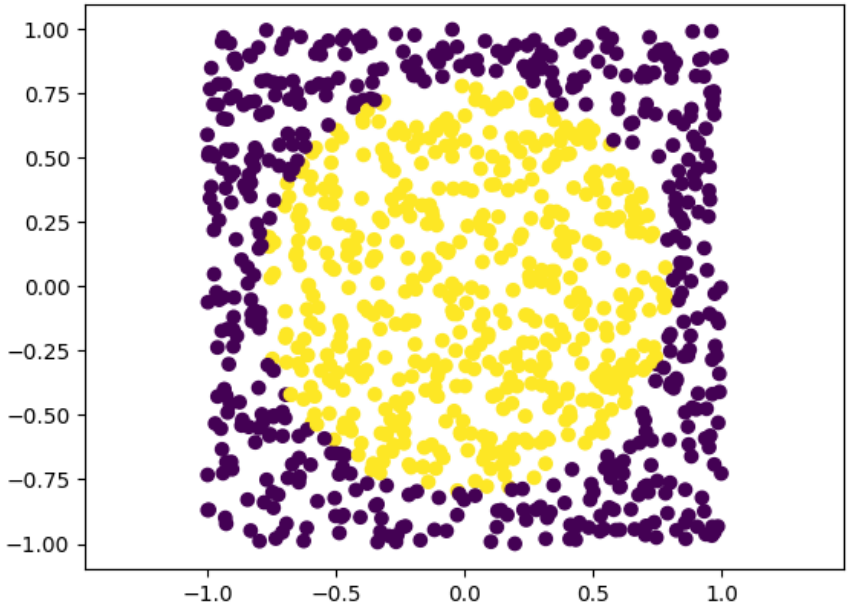
More Intuitions on the XOR Problem & MLP

- We cannot solve the following classification problem with Perceptron:



More Intuitions on the XOR Problem & MLP

- We cannot solve the following classification problem with Perceptron:



- Unless you have terms such as x^2 and y^2.

Feature Engineering

- Traditional ML workflow included the necessary feature engineering stage, where scientists and engineers had to craft mathematical features to explain data.
- However, what if you…
  - Have large dimensional data?
  - Have patterns and trends that are too complicated?
  - Don’t know much math?

Universal Approximation Theorem

- TL;DR: MLP can represent ANY given function
  (*as far as the function shape is nice)

- Details: Reading assignment!

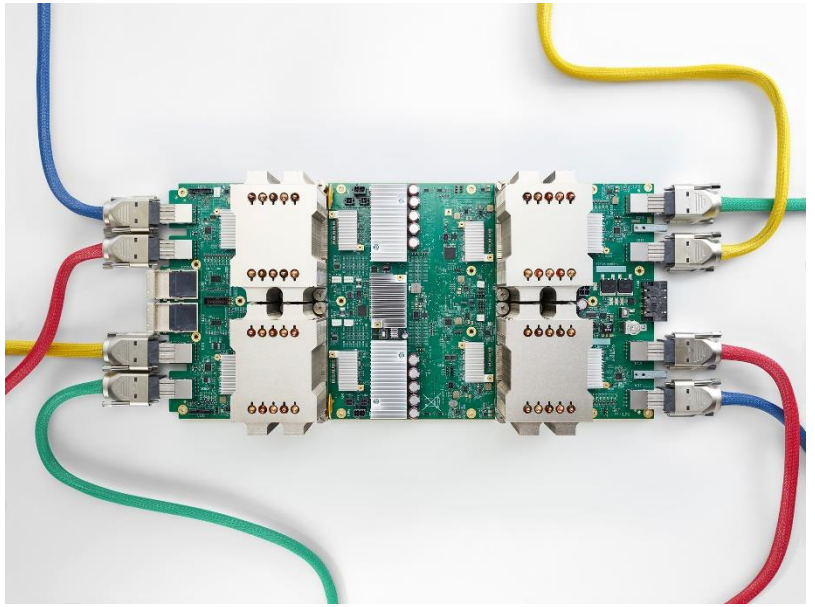
- Cybenko (1989) proved it with sigmoidal activation
- Hornik, Stinchcombe, & White (1989) proved that it is true for any other activation functions

5MB Hard Drive Being Shipped by IBM, 1956

60 years later…

NVIDIA GeForce Quadro GV100

NVIDIA Jetson TX2



Google Tensor Processing Unit (TPU)


IMAGENET

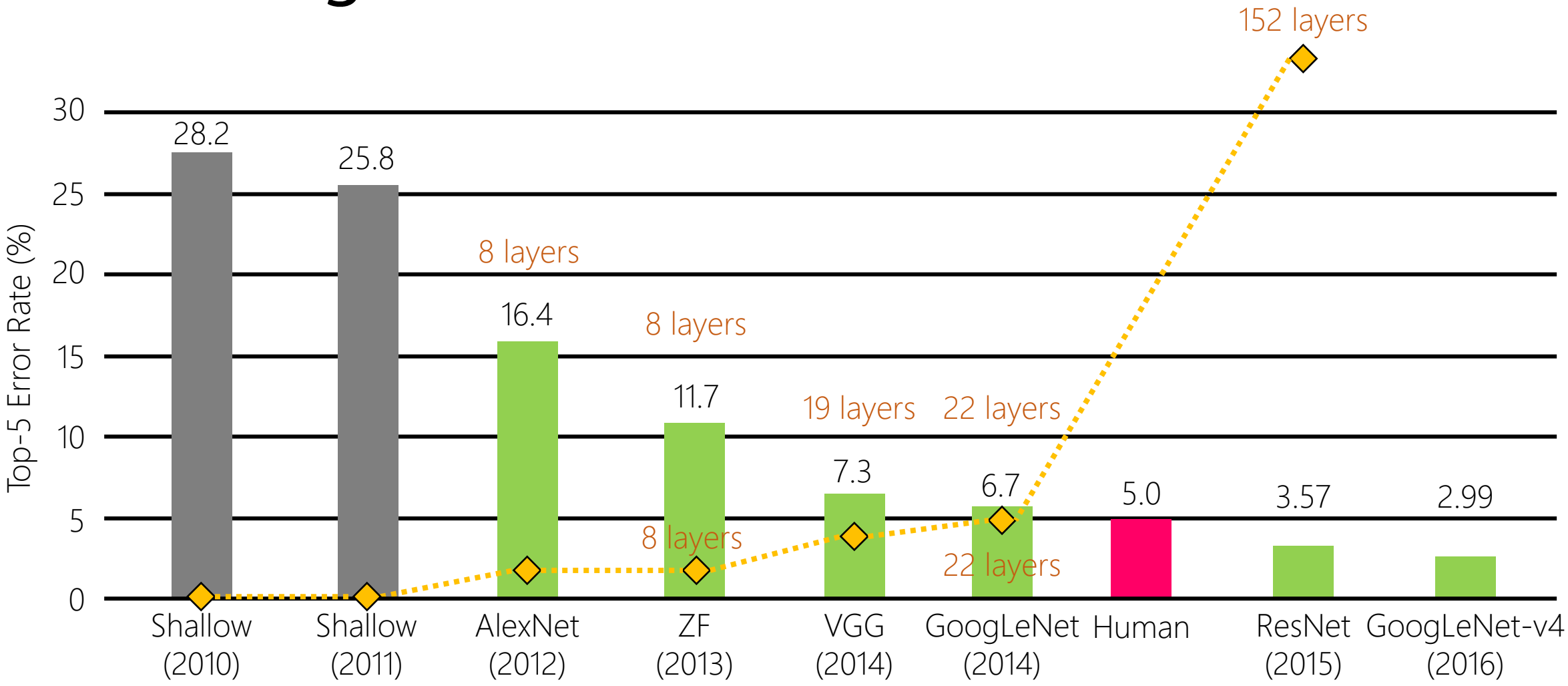
ImageNet Large Scale Visual Recognition Challenge (ILSVRC)

- ImageNet (First appearance as a poster at CVPR 2009)
  - Over 14 million images with hand-annotated labels (crowdsourced via M-turk)
  - Over 20 thousand categories
  - 1M+ images with bounding boxes.

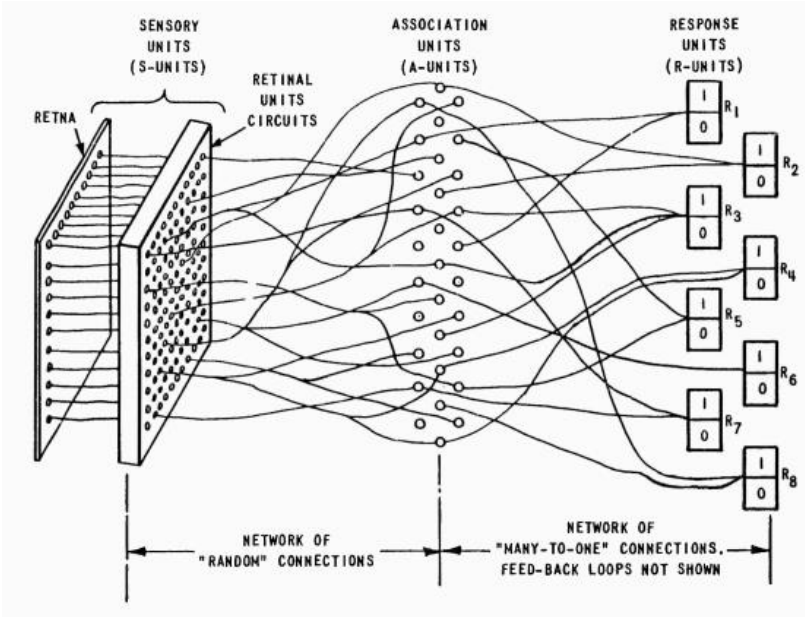
- ImageNet Challenge
  - Since 2010
  - Uses a trimmed set of thousand non-overlapping classes.
  - Humans error is about 5%*
  - Playground of computer vision researchers.
  - Marked the start of the current AI boom.

Russakovsky et al. (2015). Imagenet large scale visual recognition challenge. *International journal of computer vision*, *115*(3), 211-252.

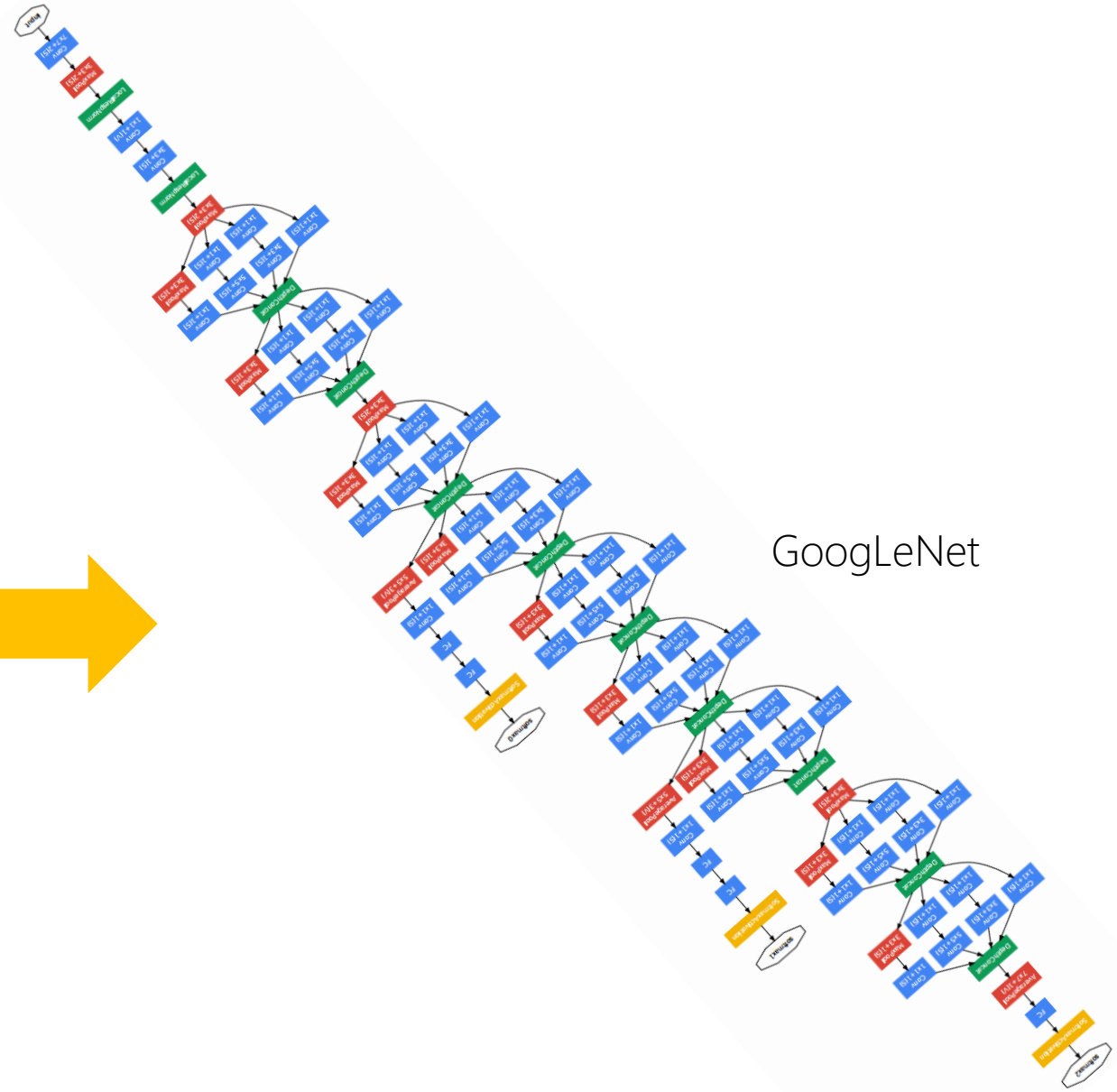
ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



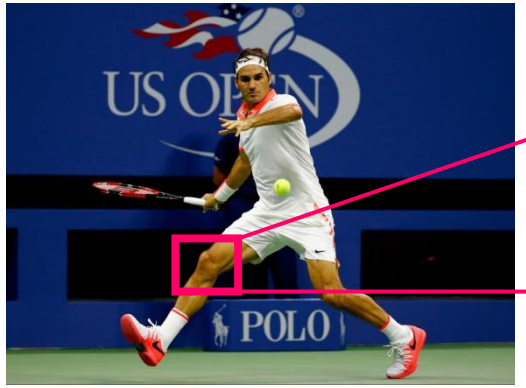
Going Deeper



Perceptron



GoogLeNet

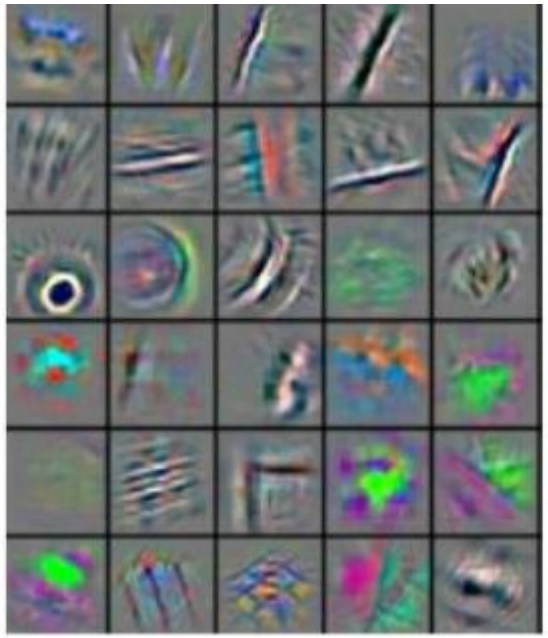


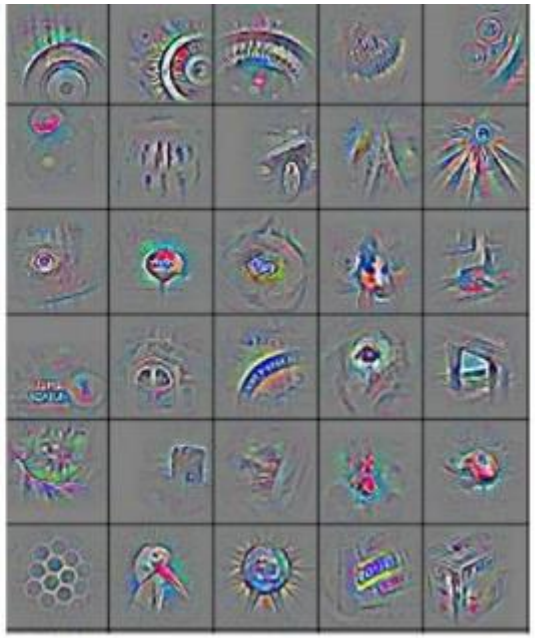
Low-level Features

Mid-level Features

High-level Features

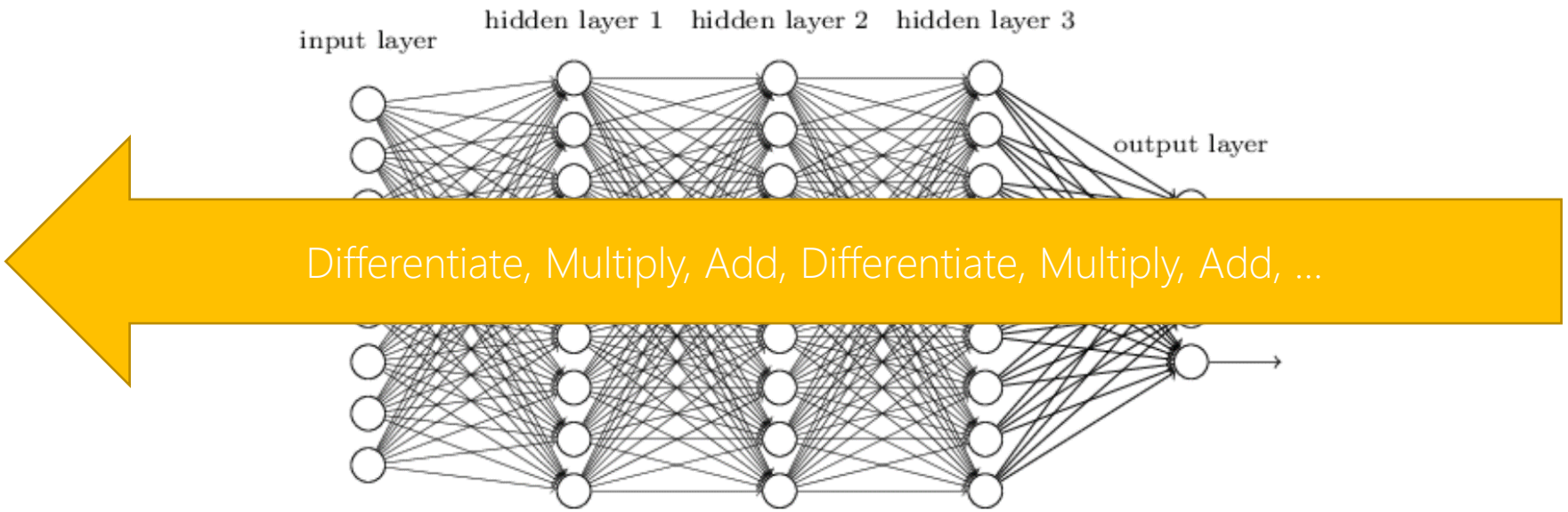
Shoulder: 0.01
Elbow: 0.12
Wrist: 0.01
Knee: 0.74
Ankle: 0.03
...





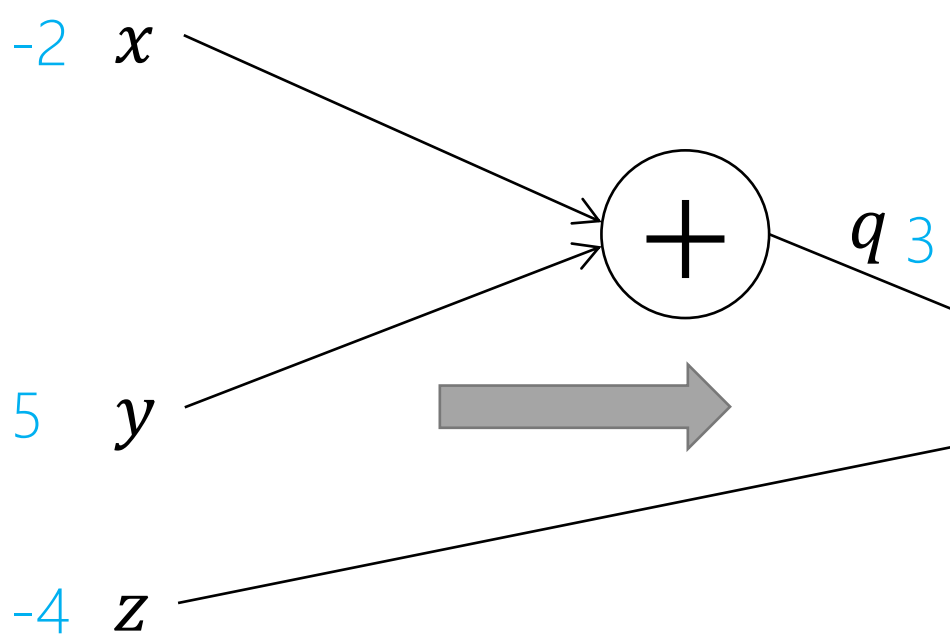
Backpropagation

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



Backpropagation

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



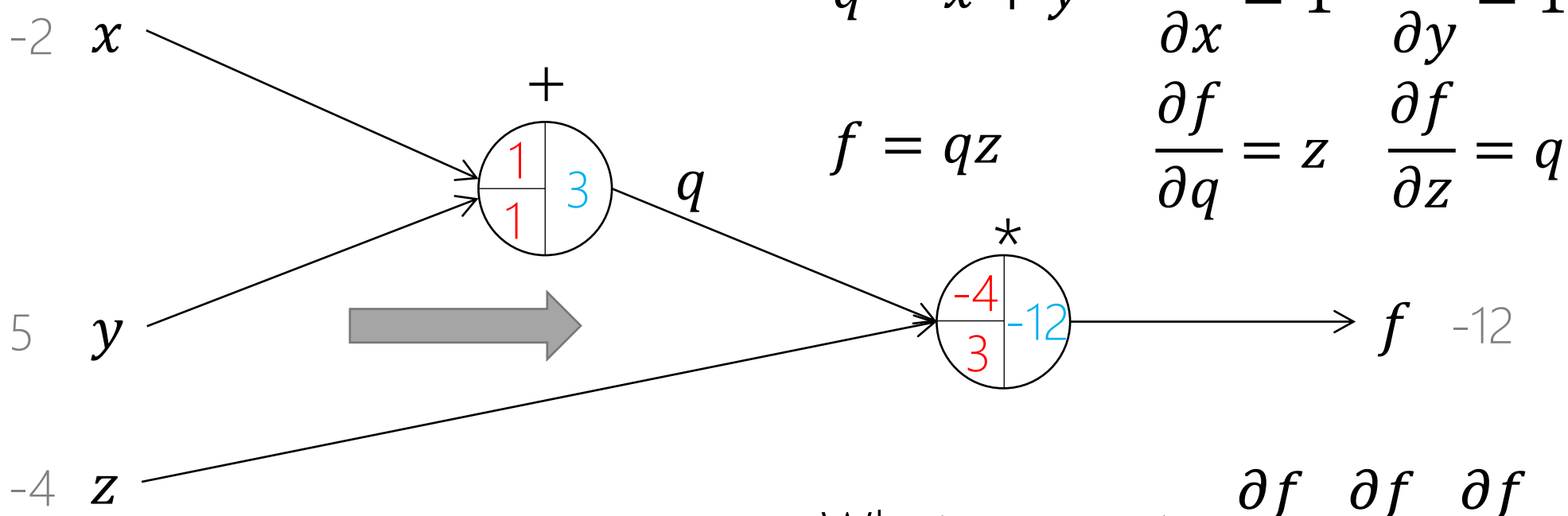
$$q = x + y \qquad \frac{\partial q}{\partial x} = 1 \qquad \frac{\partial q}{\partial y} = 1$$

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

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:



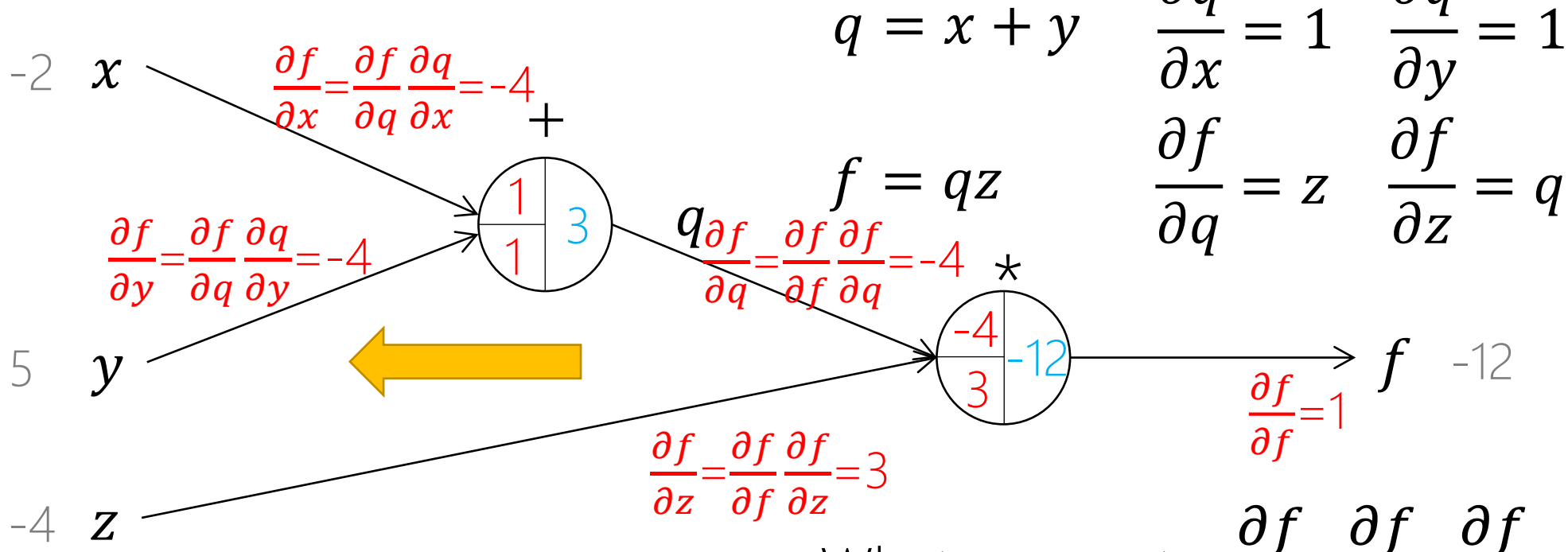
$$q = x + y \quad \frac{\partial q}{\partial x} = 1 \quad \frac{\partial q}{\partial y} = 1$$

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

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:

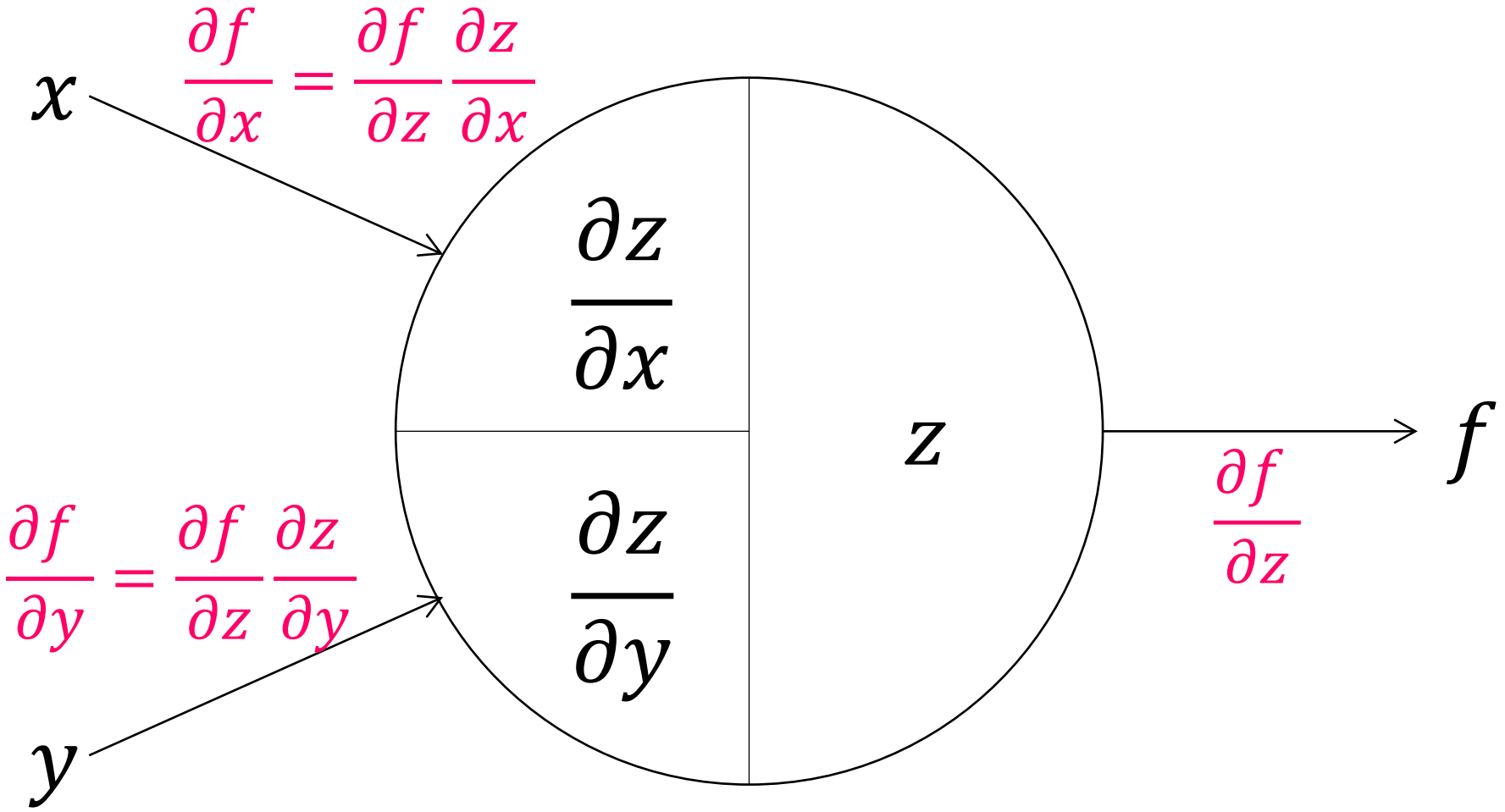


$q = x + y$ $\quad \frac{\partial q}{\partial x} = 1$ $\quad \frac{\partial q}{\partial y} = 1$

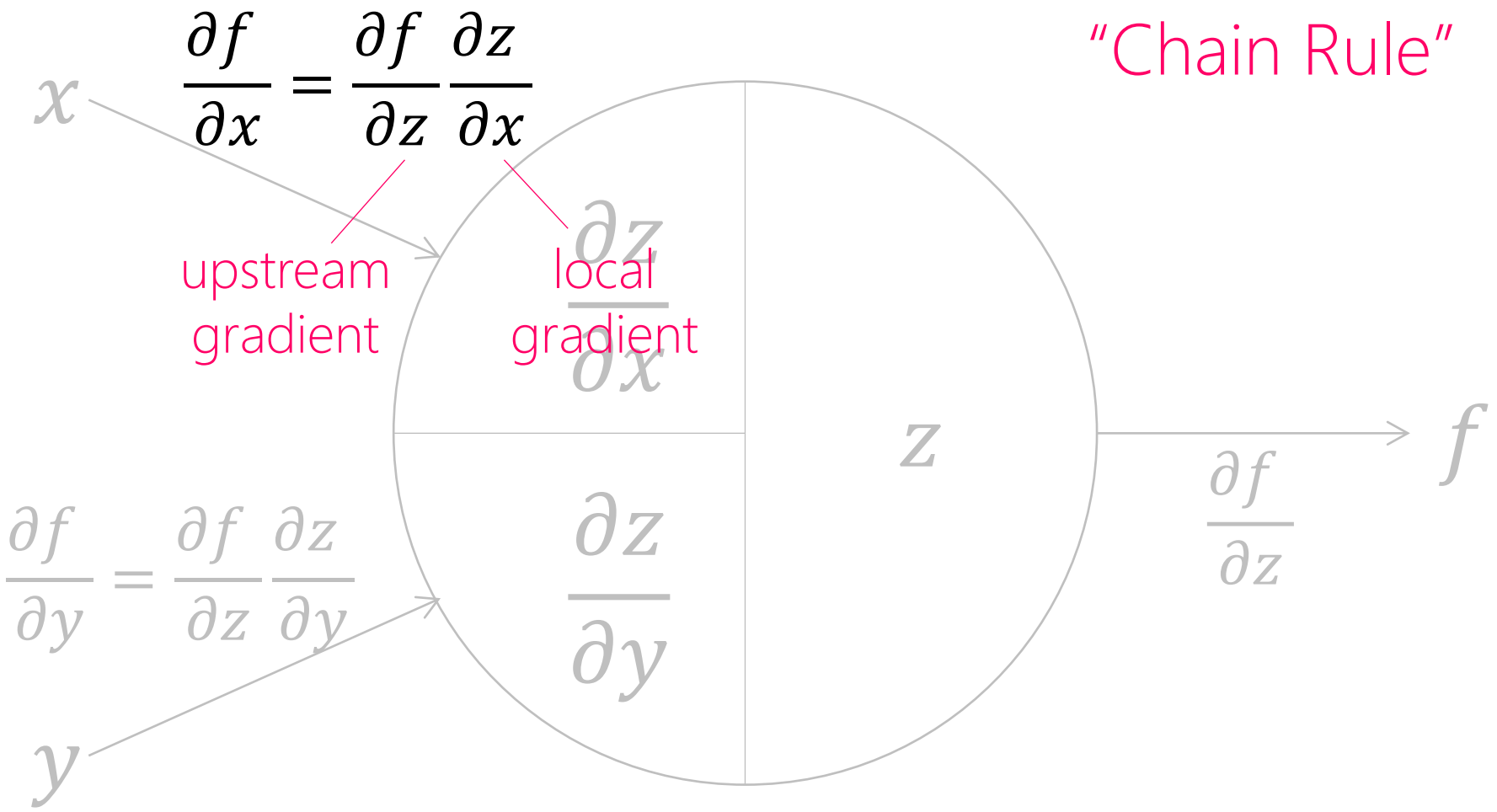
$f = qz$ $\quad \frac{\partial f}{\partial q} = z$ $\quad \frac{\partial f}{\partial z} = q$

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

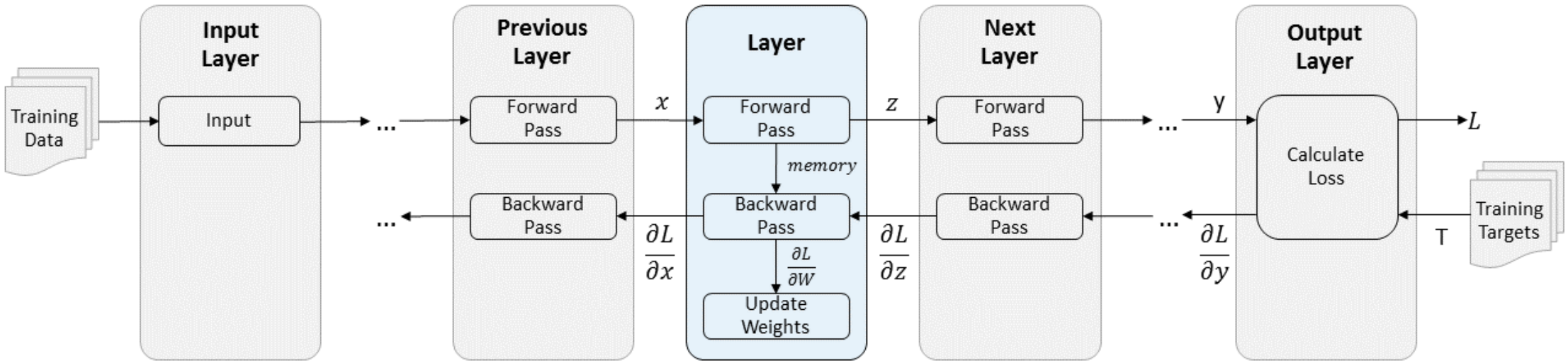
Backpropagation



Backpropagation



Putting them all together



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

Putting them all together

- TensorFlow/Keras does auto-differentiation
  - No need to define “backward” routine.

```
from keras import backend as K
from keras.engine.topology import Layer
import numpy as np

class MyLayer(Layer):

    def __init__(self, output_dim, **kwargs):
        self.output_dim = output_dim
        super(MyLayer, self).__init__(**kwargs)

    def build(self, input_shape):
        # Create a trainable weight variable for this layer.
        self.kernel = self.add_weight(name='kernel',
                                      shape=(input_shape[1], self.output_dim),
                                      initializer='uniform',
                                      trainable=True)
        super(MyLayer, self).build(input_shape)  # Be sure to call this at the end

    def call(self, x):
        return K.dot(x, self.kernel)

    def compute_output_shape(self, input_shape):
        return (input_shape[0], self.output_dim)
```

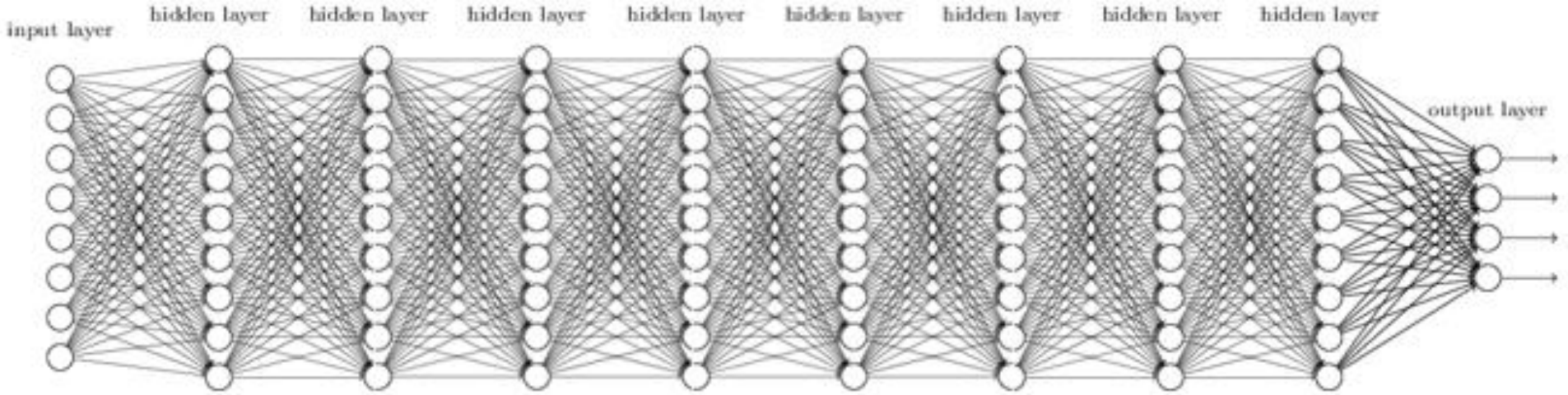
https://keras.io/layers/writing-your-own-keras-layers/

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:

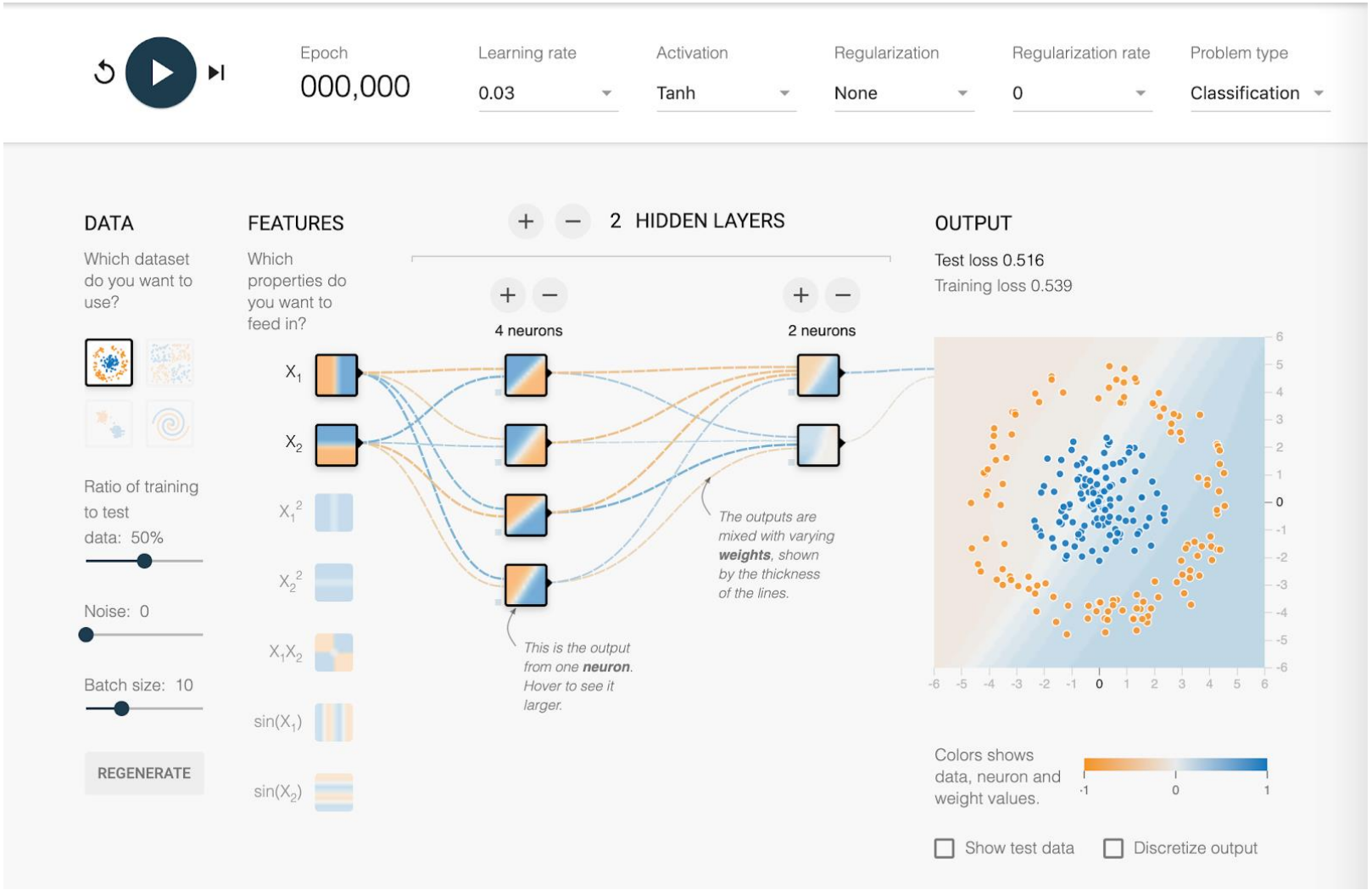
$$f = W_3 \max(0, W_2 \max(0, W_1 x + b_1) + b_2) + b_3$$

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



Deep Neural Network

TensorFlow Playground



https://playground.tensorflow.org