

Welcome to „Introduction to Deep Learning“

We will start at 9am.

The slides can be downloaded from: <https://tinyurl.com/dl-intro-slides>



Introduction to Deep Learning

Part I - Basics

June 16, 2021

Auliya Fitri, Jakob Gawlikowski

Machine Learning Group
Institute of Data Science (DW)
Jena



Auliya Fitri

German Aerospace Center – Institute of Data Science
Data Management and Analysis
Machine Learning Group

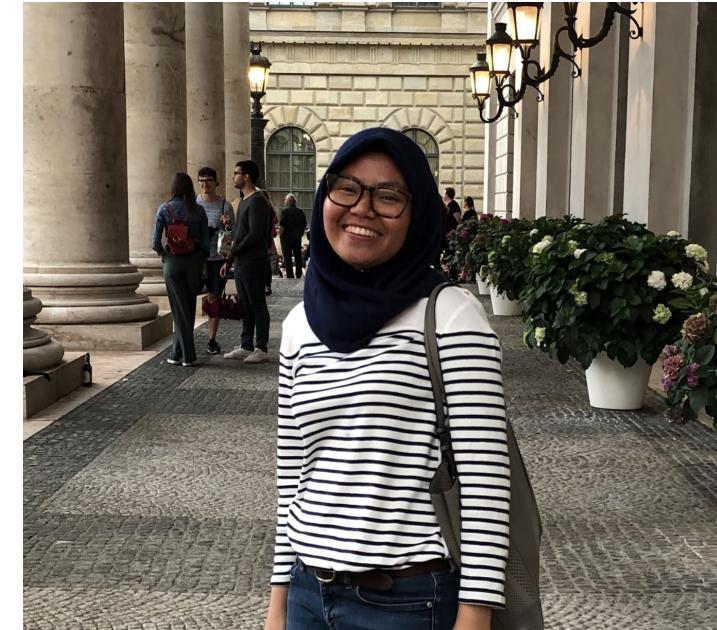
Research Interests:

- > Machine Learning
- > Explainable Artificial Intelligence
- > Uncertainties in Neural Networks

Contact:

Phone: +49 3641 30960 165

Mail: auliya.fitri@dlr.de



Jakob Gawlikowski

German Aerospace Center – Institute of Data Science
Data Management and Analysis
Machine Learning Group
PhD Student: Data Science in Earth Observation (TUM)

Research Interests:

- > Robust Machine Learning
- > Uncertainty in Neural Networks
- > Out-of-Distribution Detection
- > Data Fusion

Contact:

Phone: +49 3641 30960 143

Mail: jakob.gawlikowski@dlr.de



Curriculum

A: Theoretical introduction – Morning

- I. Introduction and basics
- II. Advanced concepts
- III. Practical application

B. Hands-on seminar – Afternoon

Run prepared Jupyter Notebooks online on Binder, or locally on your own laptop.



Curriculum

I. Introduction and basics

- Application examples
- Machine learning background
- Neural network concepts
- Training procedure

Inspired by lectures from Paris Saclay and MIT; images taken from these, if not noted otherwise



Application examples

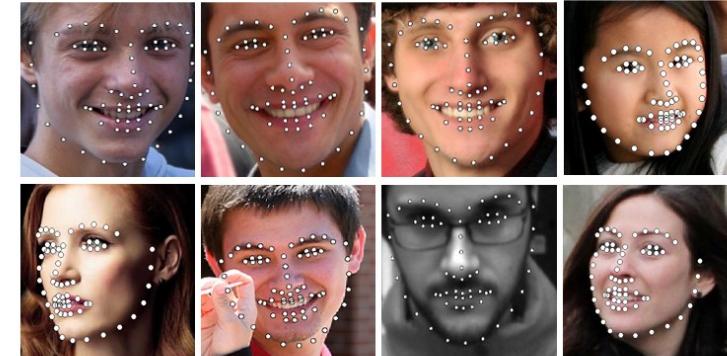


Application Examples

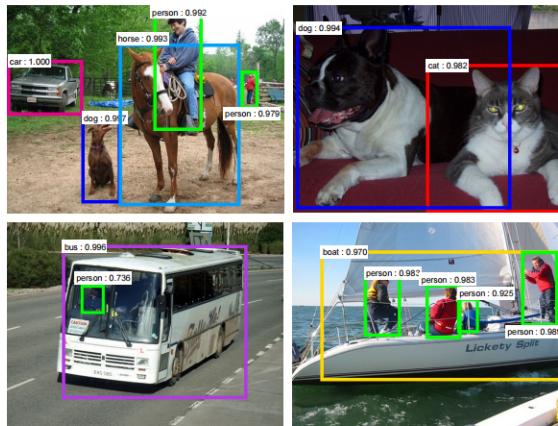
Vision



[Krizhevsky 2012]



[Facial landmark detection CUHK 2014]



[Faster R-CNN - Ren 2015]



[NVIDIA dev blog]

Application Examples

Natural Language Processing (NLP)



Application Examples

Natural Language Processing (NLP)



[Google Inbox Smart Reply]



[Amazon Echo / Alexa]

Application Examples

Generative models

| | | | | | | | | |
|------------------|--|---|---|--|---|--|---|--|
| Text description | This bird is blue with white and has a very short beak | This bird has wings that are brown and has a yellow belly | A white bird with a black crown and yellow beak | This bird is white, black, and brown in color, with a brown beak | The bird has small beak, with reddish brown crown and gray belly | This is a small, black bird with a white breast and white on the wingbars. | This bird is white black and yellow in color, with a short black beak | |
| Stage-I images |  | | | |  | | | |
| Stage-II images |  | | | |  | | | |

StackGAN v2 [Zhang 2017]

Application Examples

Generative Models



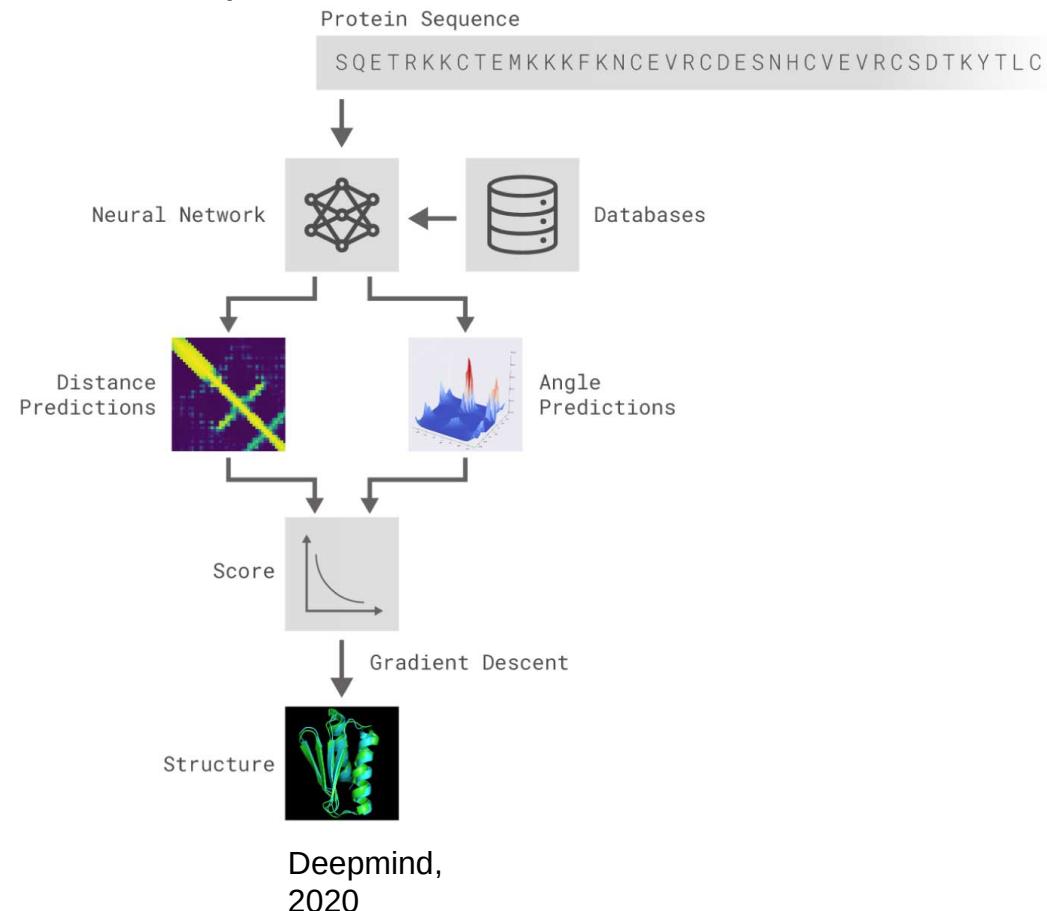
<https://www.youtube.com/watch?v=g5wLaJYBAm4>

Suwajanakorn, Supasorn, Steven M. Seitz, and Ira Kemelmacher-Shlizerman. "Synthesizing obama: learning lip sync from audio."

Application Examples

Other sciences

AlphaFold

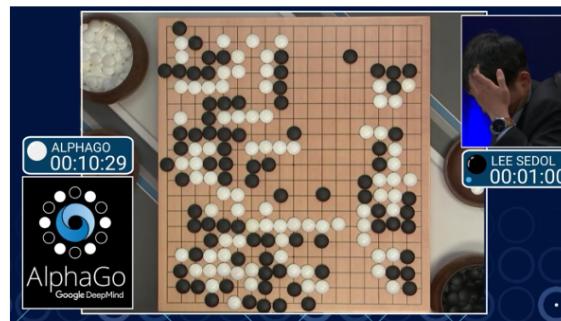


Deepmind,
2020

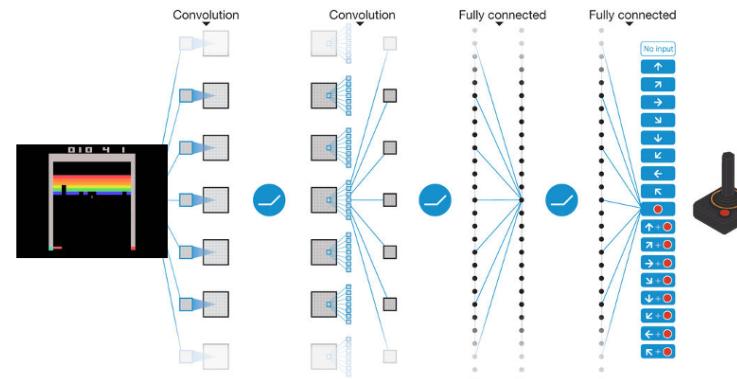
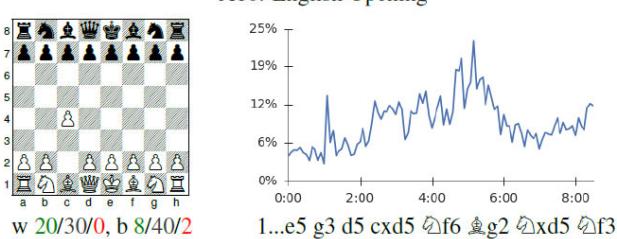
<https://deepmind.com/blog/article/AlphaFold-Using-AI-for-scientific-discovery>

Application Examples

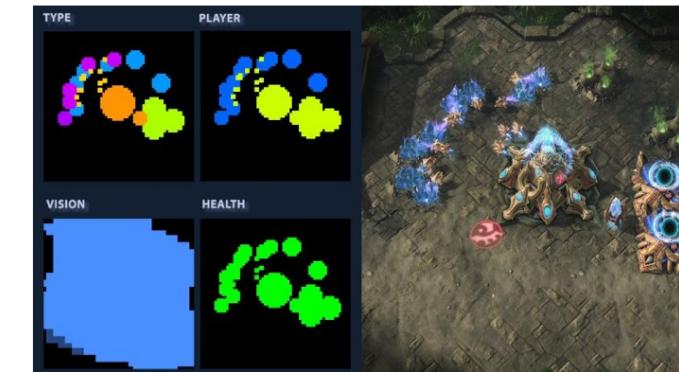
Game AI



[Deepmind AlphaGo / Zero 2017]



[Atari Games - DeepMind 2016]



[Starcraft 2 for AI research]

Machine learning background



Machine learning background

ARTIFICIAL INTELLIGENCE

Any technique that enables computers to mimic human behavior



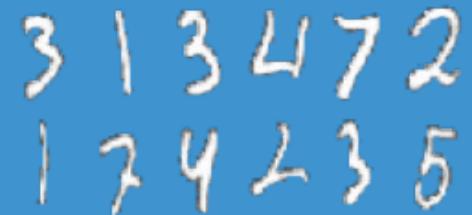
MACHINE LEARNING

Ability to learn without explicitly being programmed

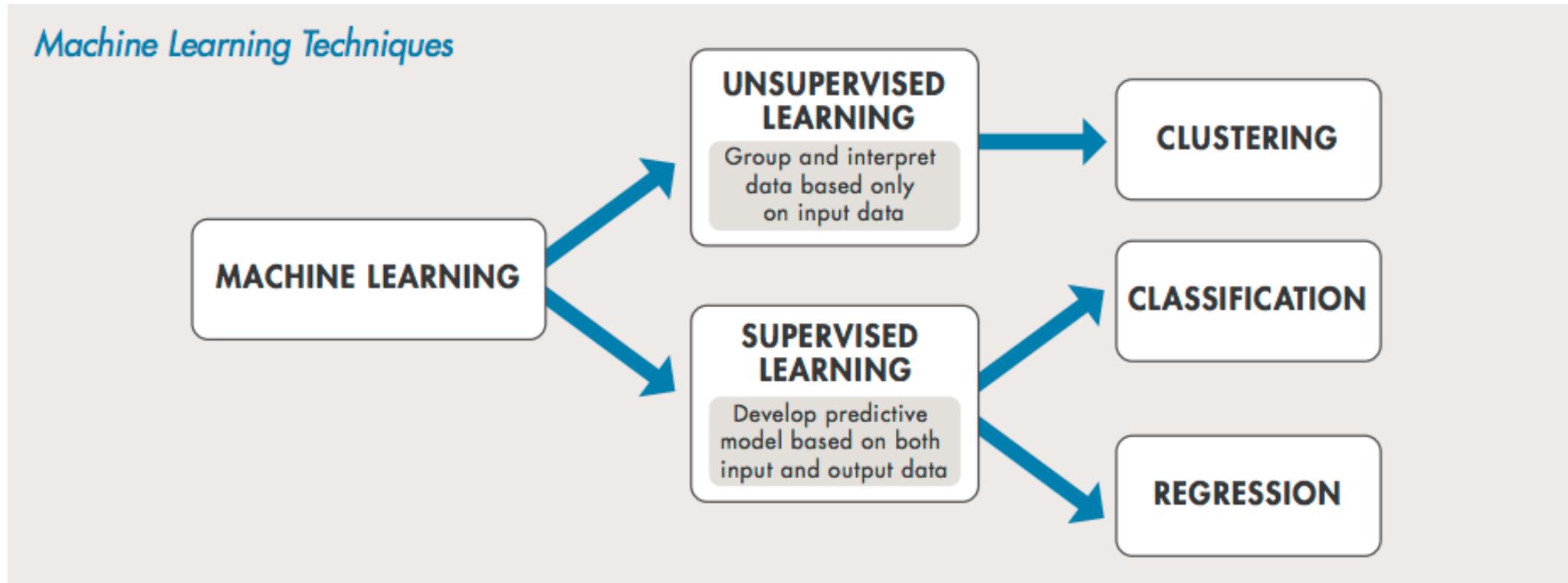


DEEP LEARNING

Extract patterns from data using neural networks

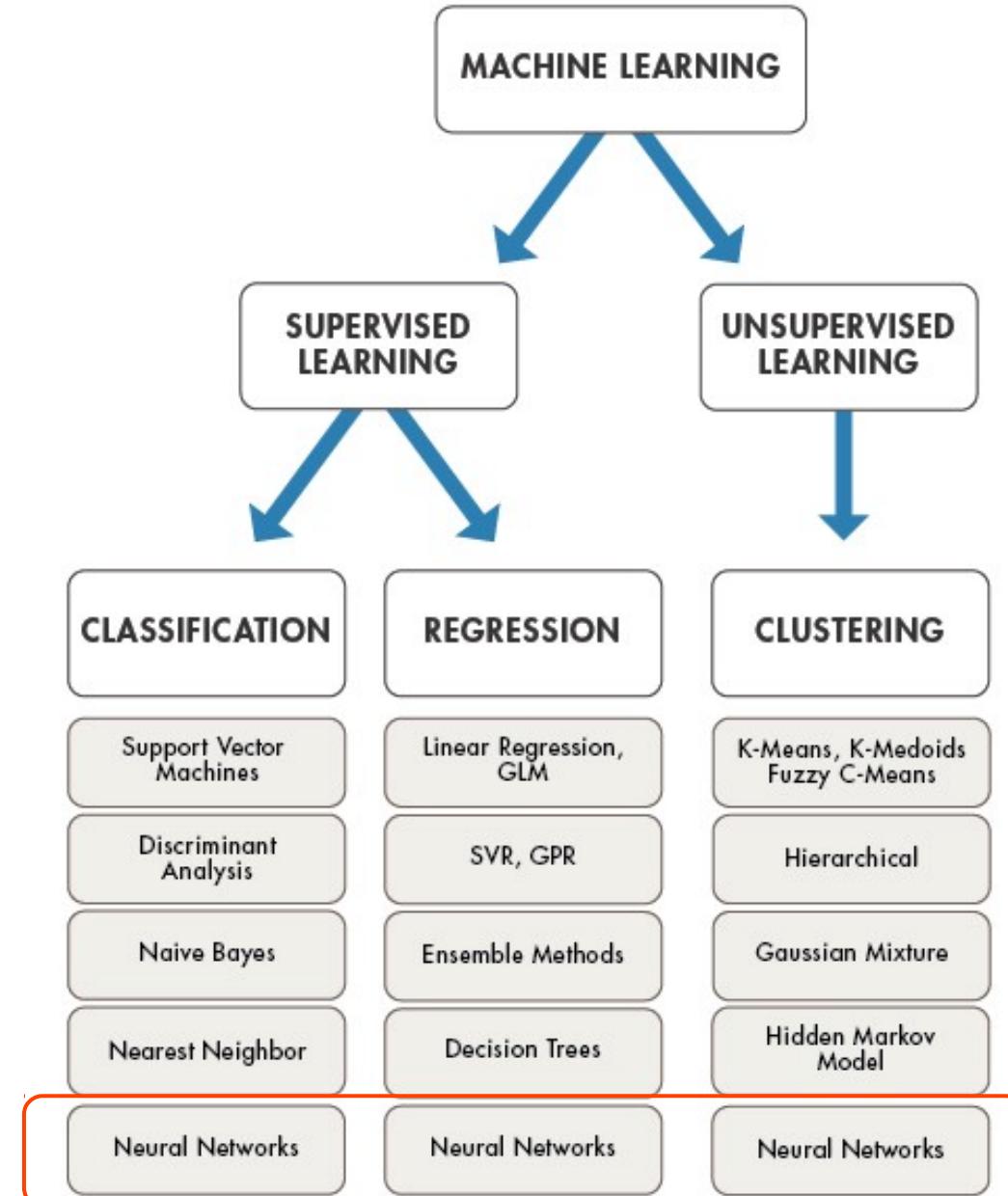


Machine learning background



<https://vitalflux.com/dummies-notes-supervised-vs-unsupervised-learning/>

Machine learning background



<https://medium.com/technology-nineleaps/popular-machine-learning-algorithms-a574e3835ebb>

Machine learning background

“Traditional” ML system

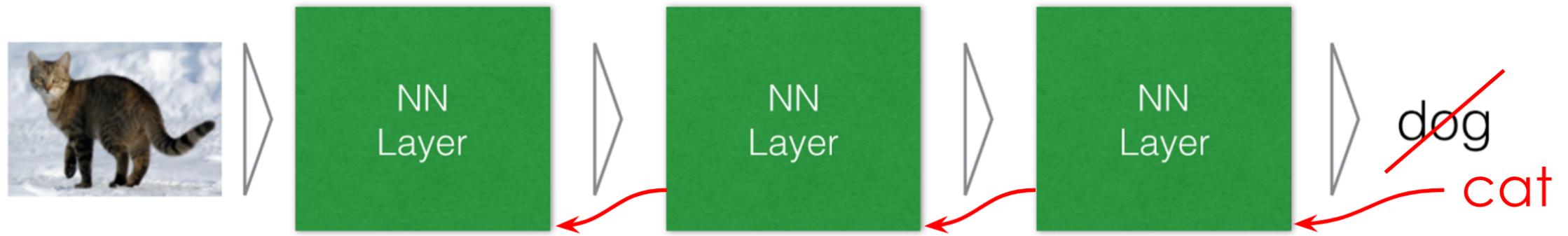


Data
independent

Supervised
Learning

Machine learning background

Deep learning system



Supervised
Learning

Machine learning background

Deep learning system

Why deep learning now?

I. Big Data

- Larger Datasets
- Easier Collection & Storage



WIKIPEDIA
The Free Encyclopedia



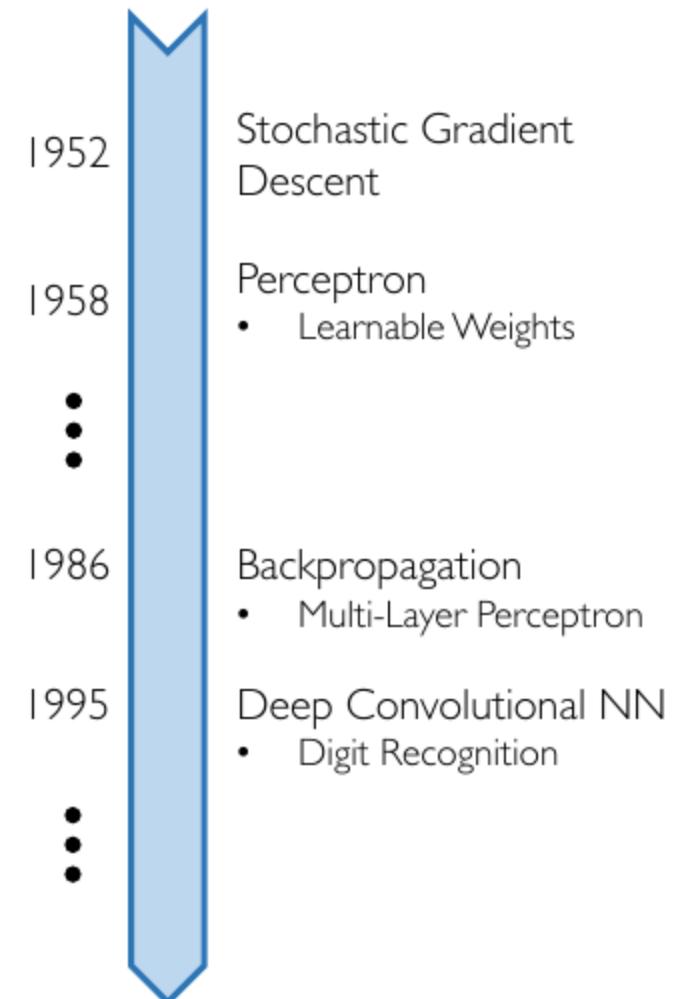
2. Hardware

- Graphics Processing Units (GPUs)
- Massively Parallelizable



3. Software

- Improved Techniques
- New Models
- Toolboxes

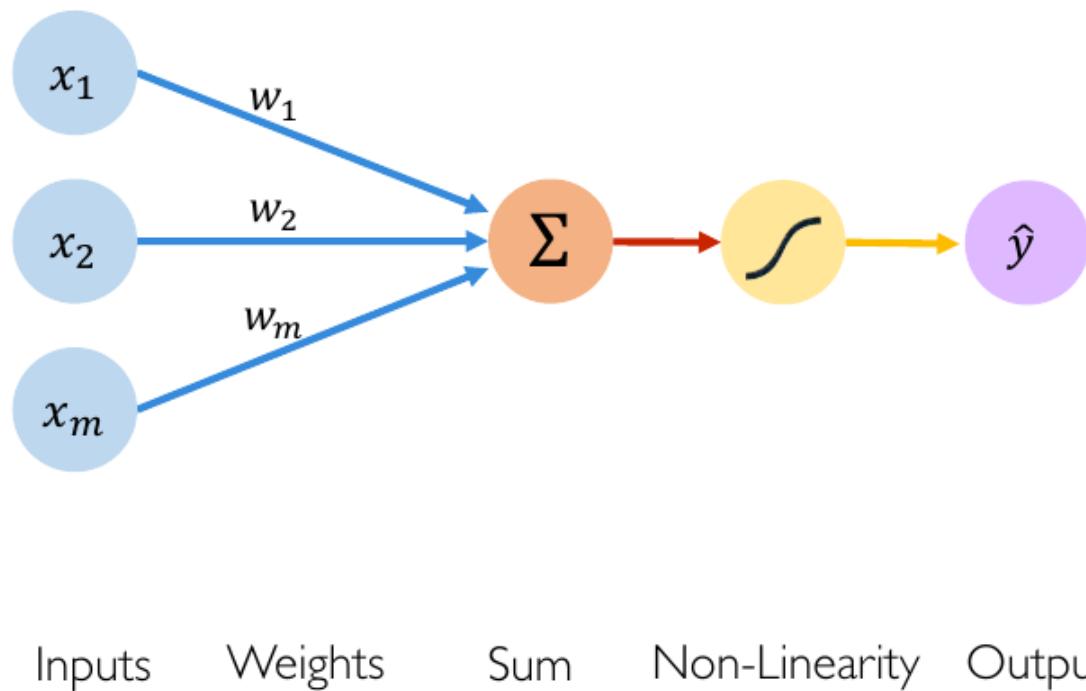


Neural network concepts



Neural network concepts

A single neuron (perceptron)



Linear combination of inputs

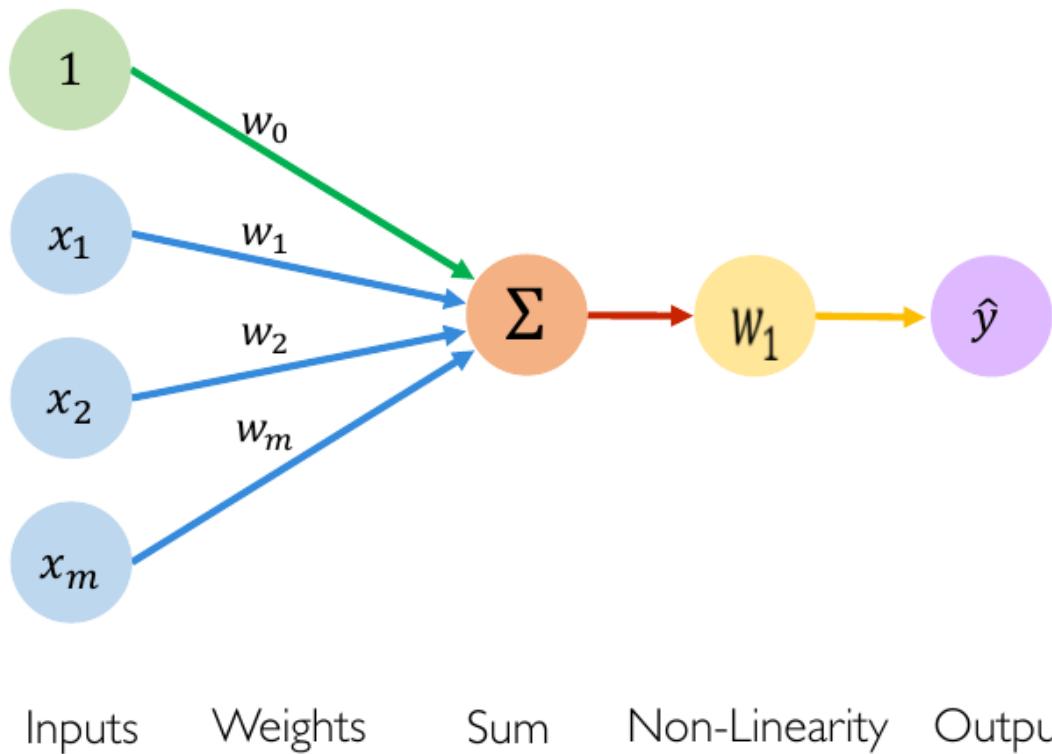
Output

$$\hat{y} = g \left(\sum_{i=1}^m x_i w_i \right)$$

Non-linear activation function

Neural network concepts

A single neuron (perceptron)



Linear combination of inputs

$$\hat{y} = g \left(w_0 + \sum_{i=1}^m x_i w_i \right)$$

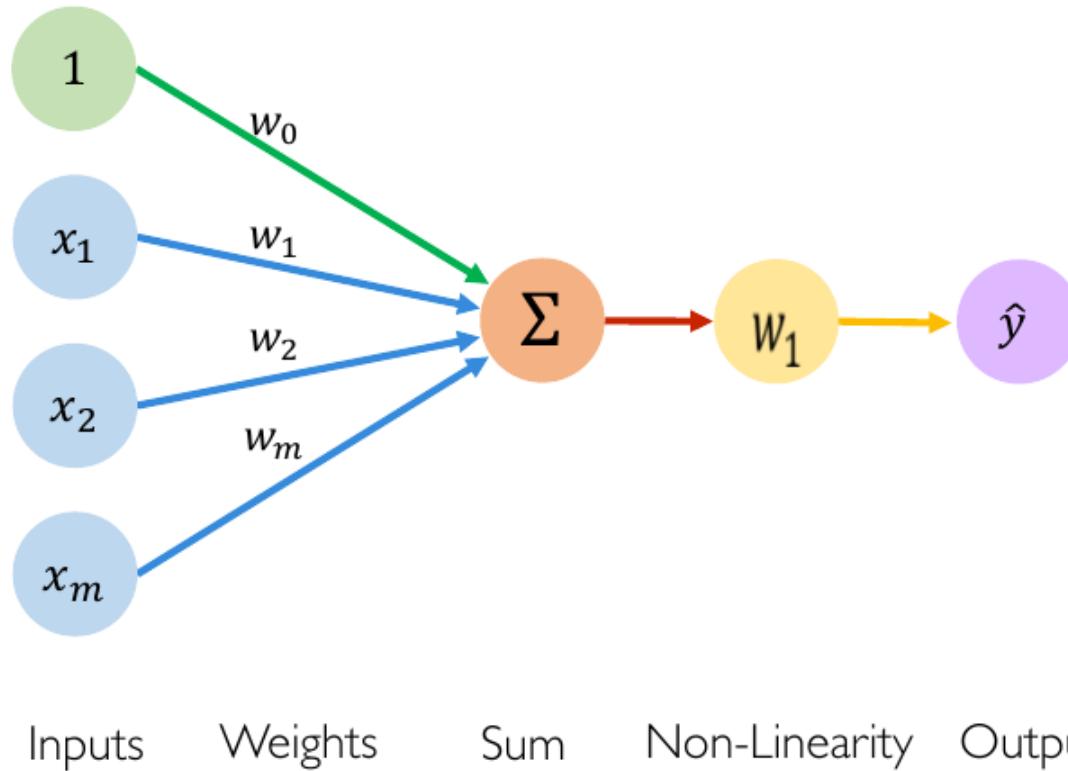
Output

Non-linear activation function

Bias

Neural network concepts

A single neuron (perceptron)



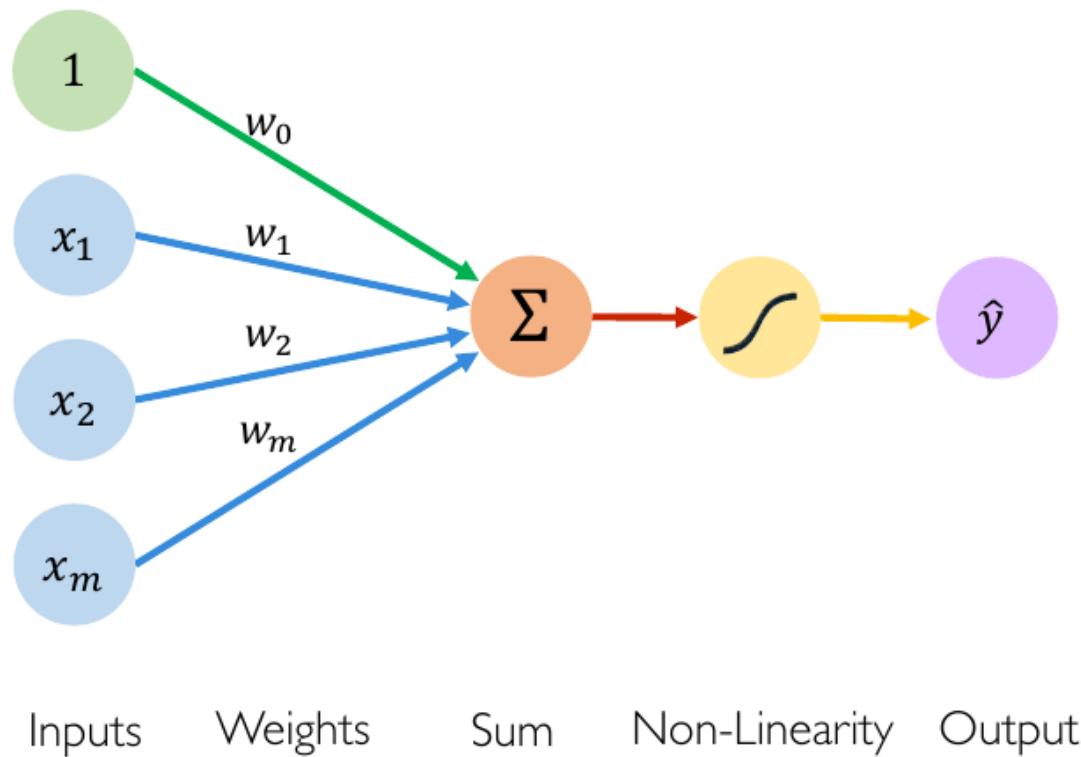
$$\hat{y} = g \left(w_0 + \sum_{i=1}^m x_i w_i \right)$$

$$\hat{y} = g(w_0 + \mathbf{X}^T \mathbf{W})$$

where: $\mathbf{X} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$ and $\mathbf{W} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}$

Neural network concepts

A single neuron (perceptron)

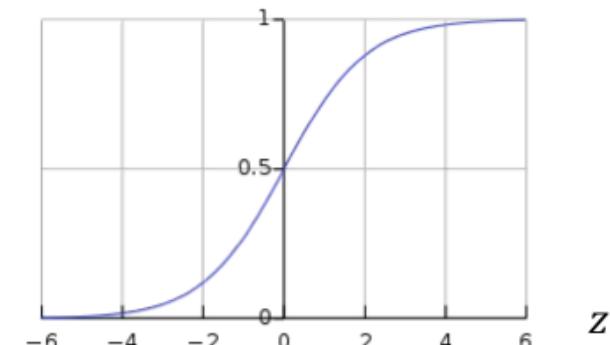


Activation Functions

$$\hat{y} = g(w_0 + X^T W)$$

- Example: sigmoid function

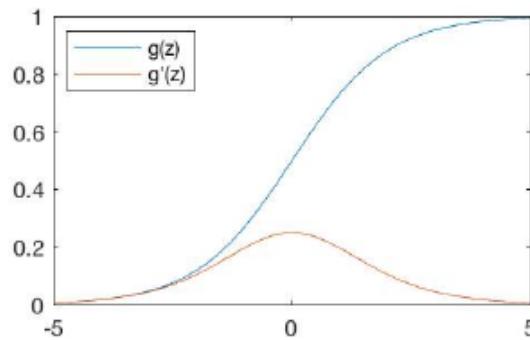
$$g(z) = \sigma(z) = \frac{1}{1 + e^{-z}}$$



Neural network concepts

Activation functions

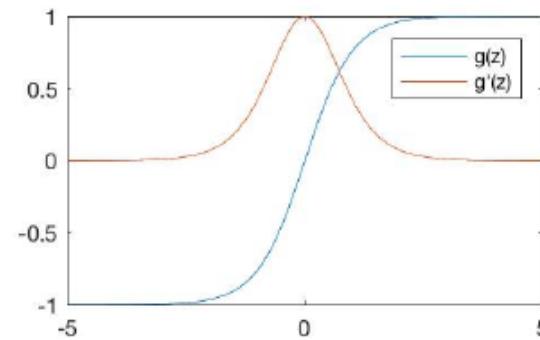
Sigmoid Function



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

$$g'(z) = g(z)(1 - g(z))$$

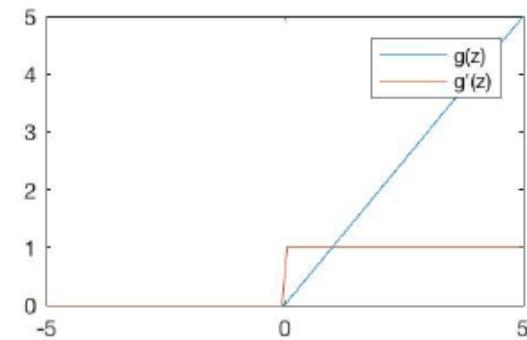
Hyperbolic Tangent



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

$$g'(z) = 1 - g(z)^2$$

Rectified Linear Unit (ReLU)



$$g(z) = \max(0, z)$$

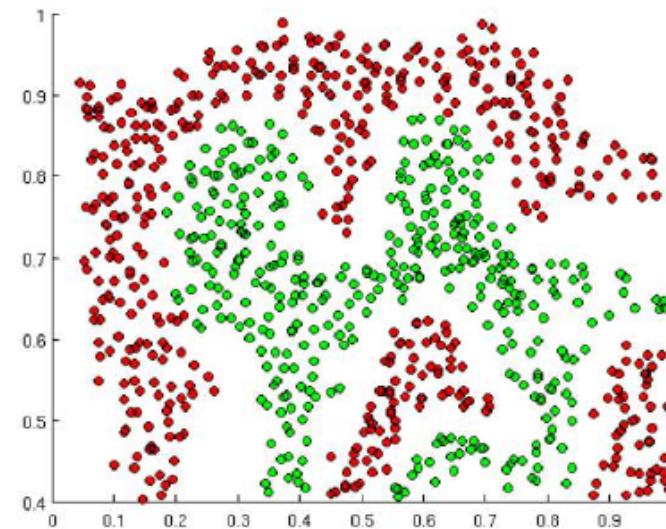
$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

NOTE: All activation functions are non-linear

Neural network concepts

Activation functions

*The purpose of activation functions is to **introduce non-linearities** into the network*

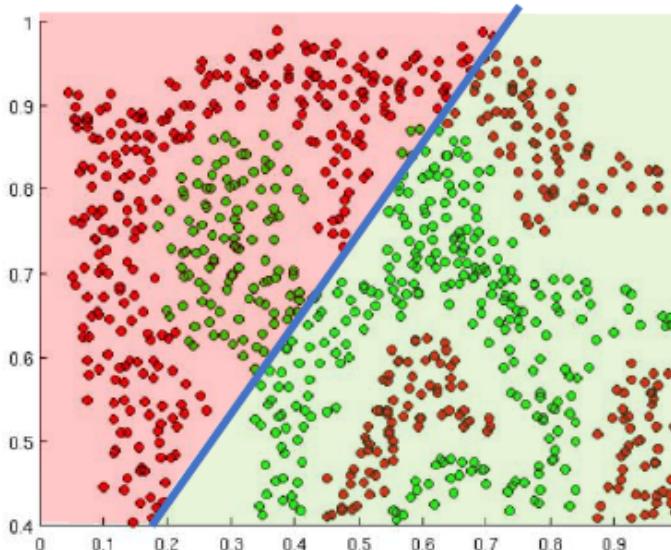


What if we wanted to build a Neural Network to
distinguish green vs red points?

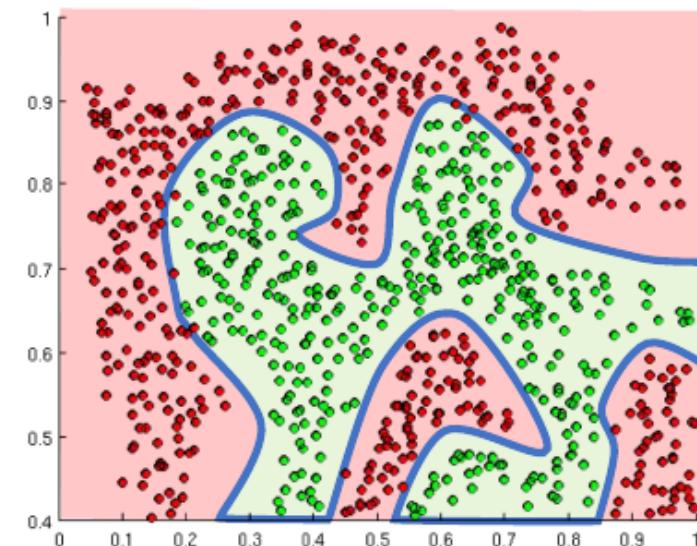
Neural network concepts

Activation functions

The purpose of activation functions is to **introduce non-linearities** into the network



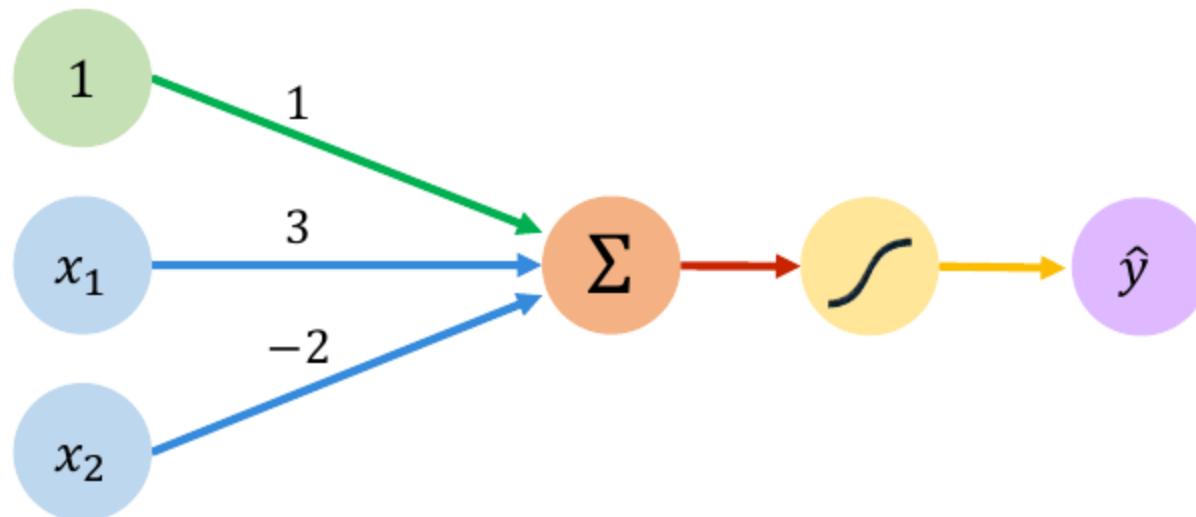
Linear Activation functions produce linear decisions no matter the network size



Non-linearities allow us to approximate arbitrarily complex functions

Neural network concepts

A single neuron (perceptron) - Example



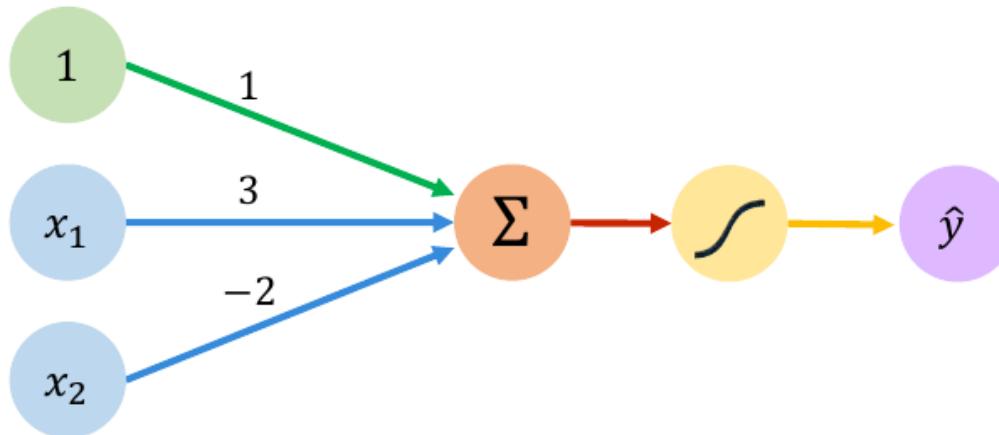
We have: $w_0 = 1$ and $\mathbf{w} = \begin{bmatrix} 3 \\ -2 \end{bmatrix}$

$$\begin{aligned}\hat{y} &= g(w_0 + \mathbf{X}^T \mathbf{w}) \\ &= g\left(1 + \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T \begin{bmatrix} 3 \\ -2 \end{bmatrix}\right) \\ \hat{y} &= g\left(1 + 3x_1 - 2x_2\right)\end{aligned}$$

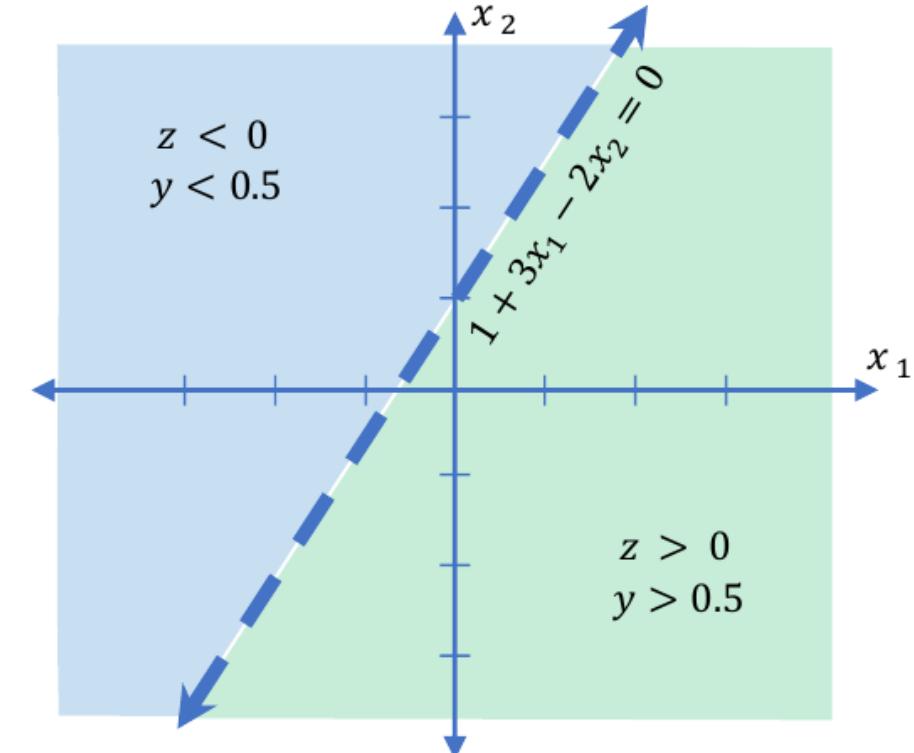
This is just a line in 2D!

Neural network concepts

A single neuron (perceptron) - Example

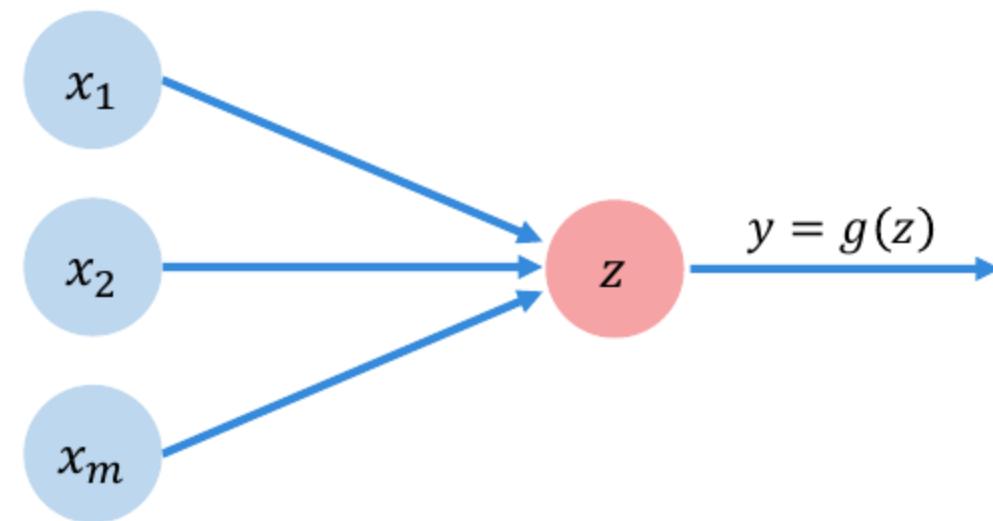


$$\hat{y} = g(1 + 3x_1 - 2x_2)$$



Neural network concepts

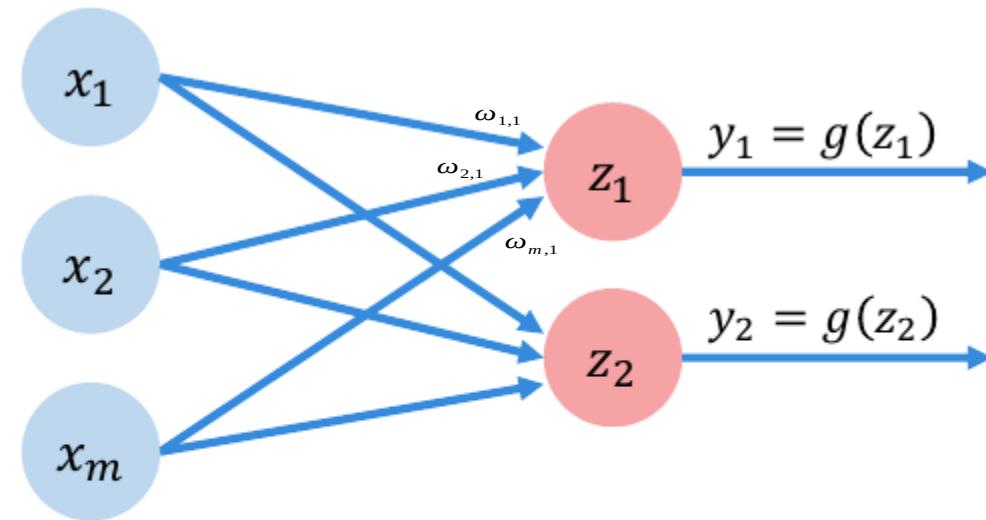
Simplified neuron



$$z = w_0 + \sum_{j=1}^m x_j w_j$$

Neural network concepts

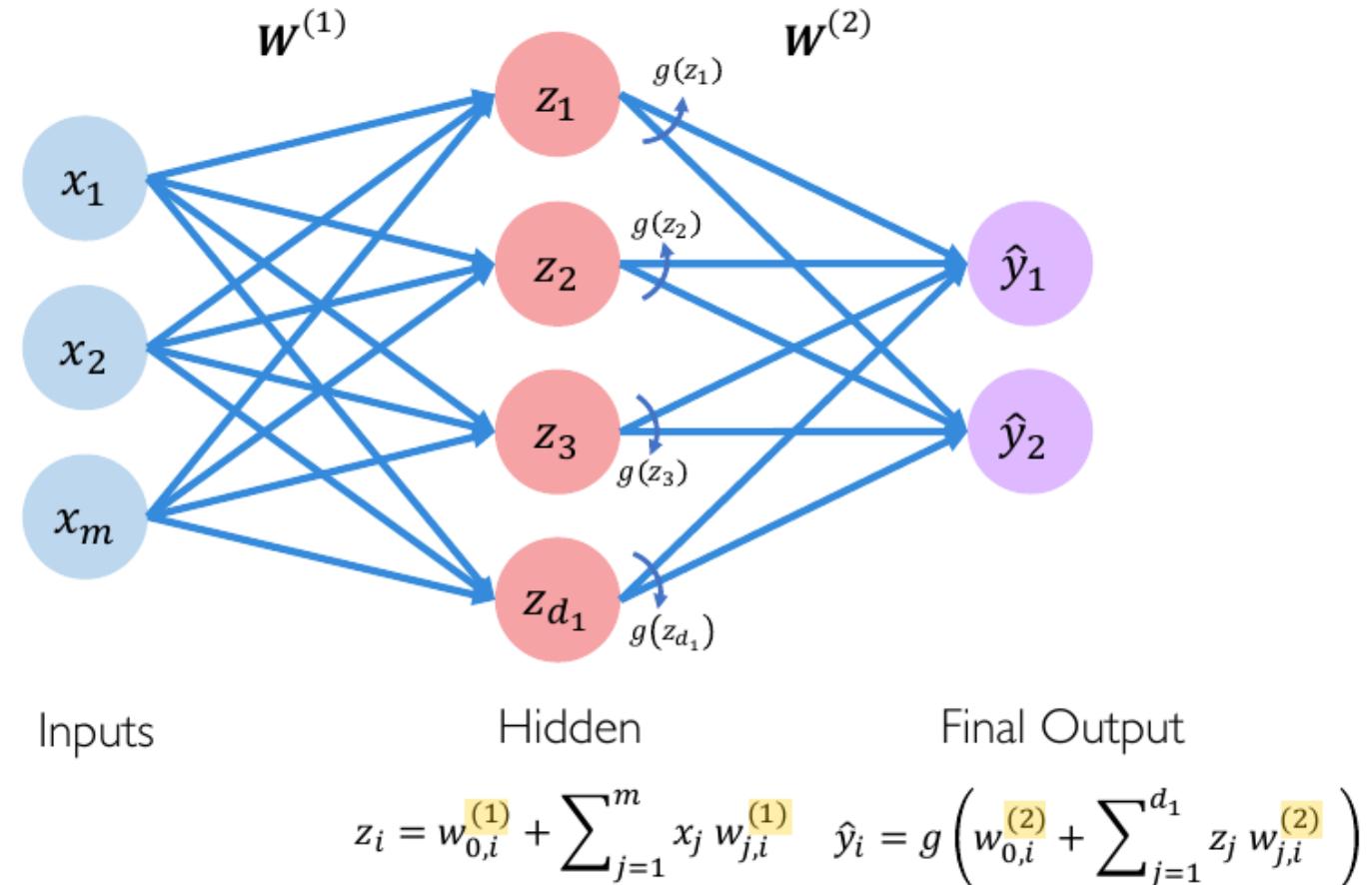
Multi-output neuron



$$z_i = w_{0,i} + \sum_{j=1}^m x_j w_{j,i}$$

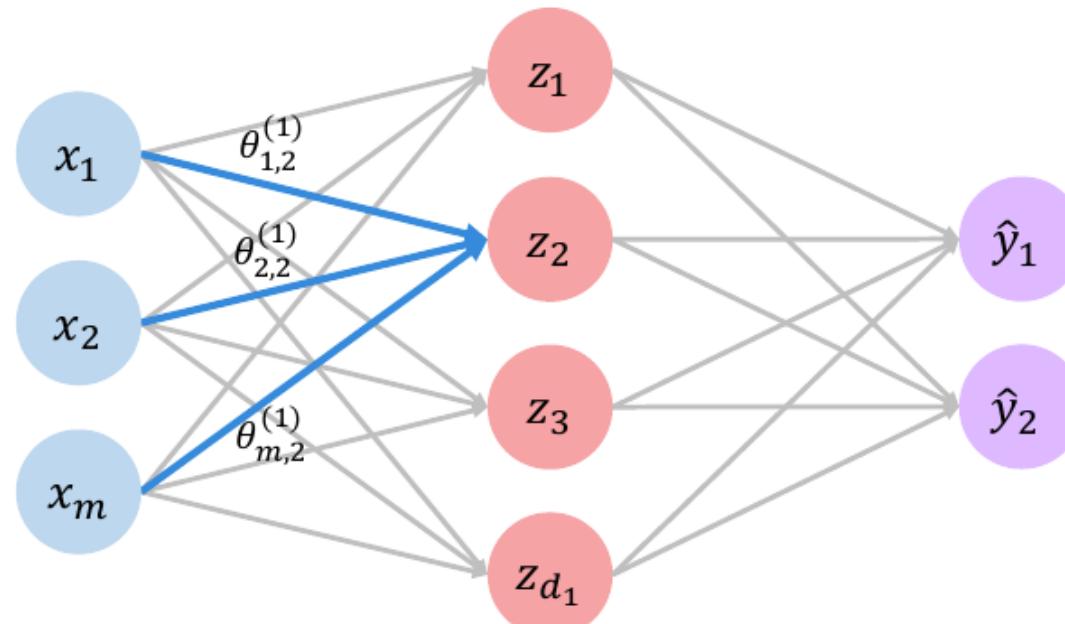
Neural network concepts

A single-layer neural network



Neural network concepts

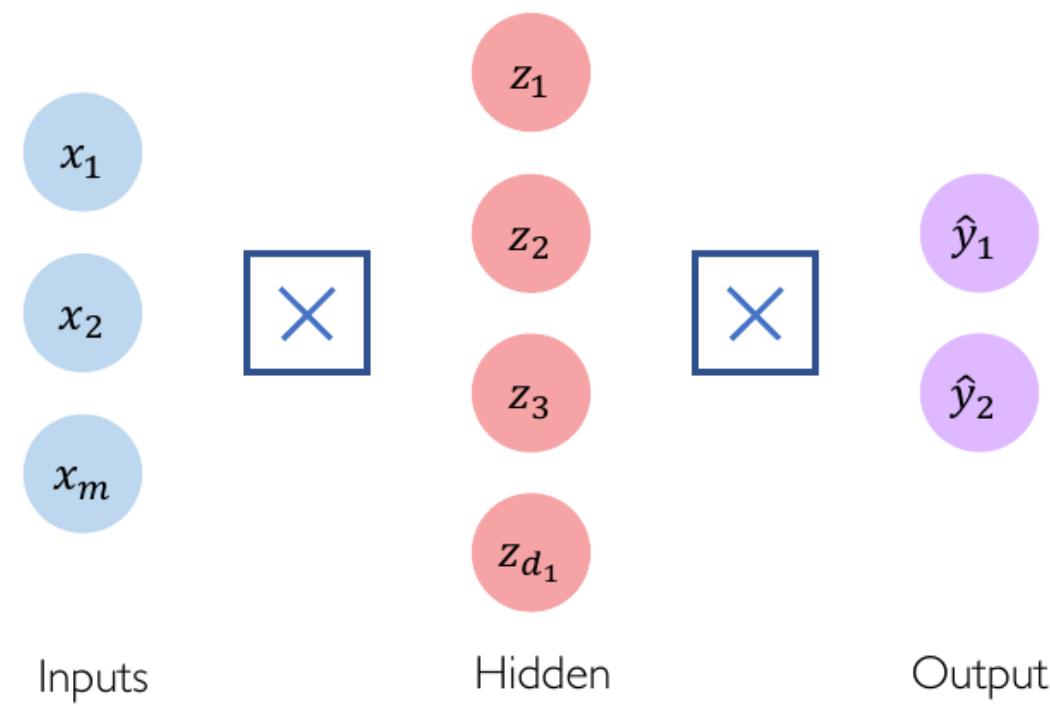
A single-layer neural network



$$\begin{aligned} z_2 &= w_{0,2}^{(1)} + \sum_{j=1}^m x_j w_{j,2}^{(1)} \\ &= w_{0,2}^{(1)} + x_1 w_{1,2}^{(1)} + x_2 w_{2,2}^{(1)} + x_m w_{m,2}^{(1)} \end{aligned}$$

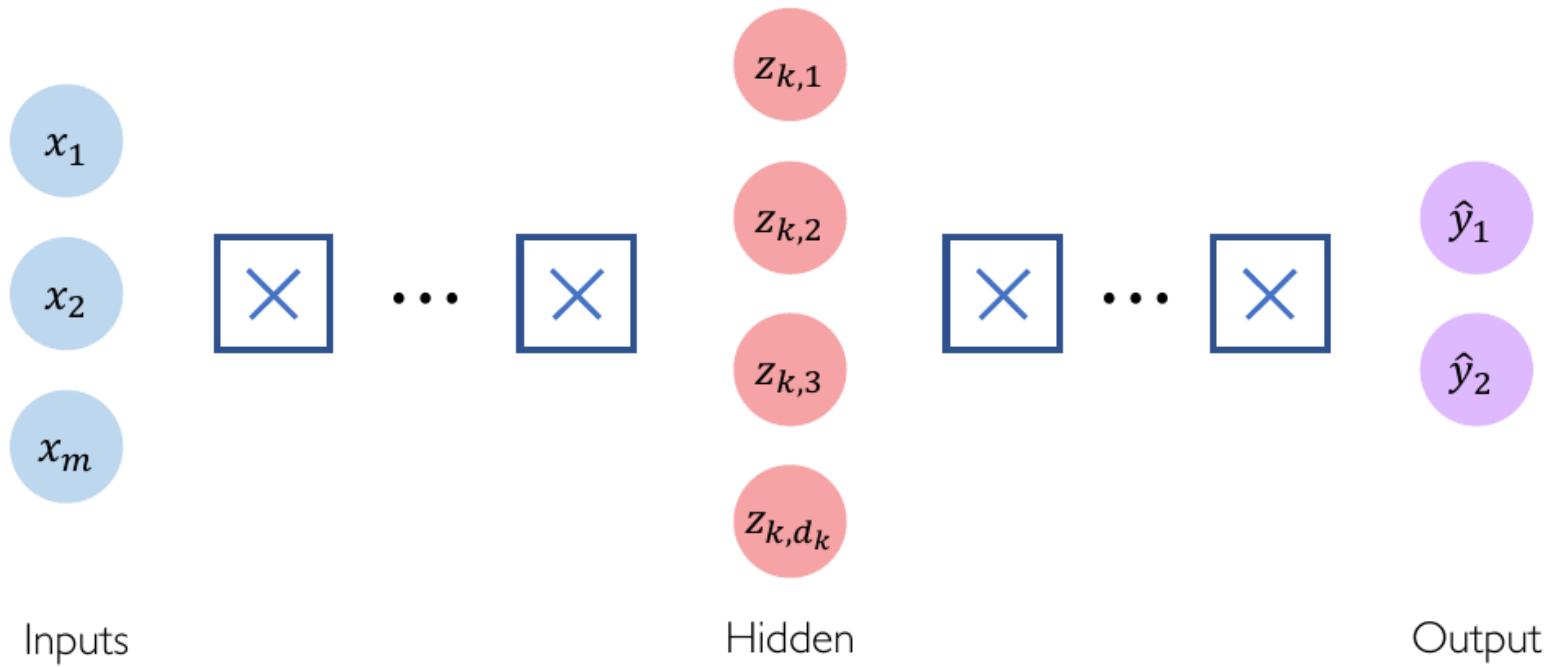
Neural network concepts

A single-layer neural network



Neural network concepts

Deep neural network

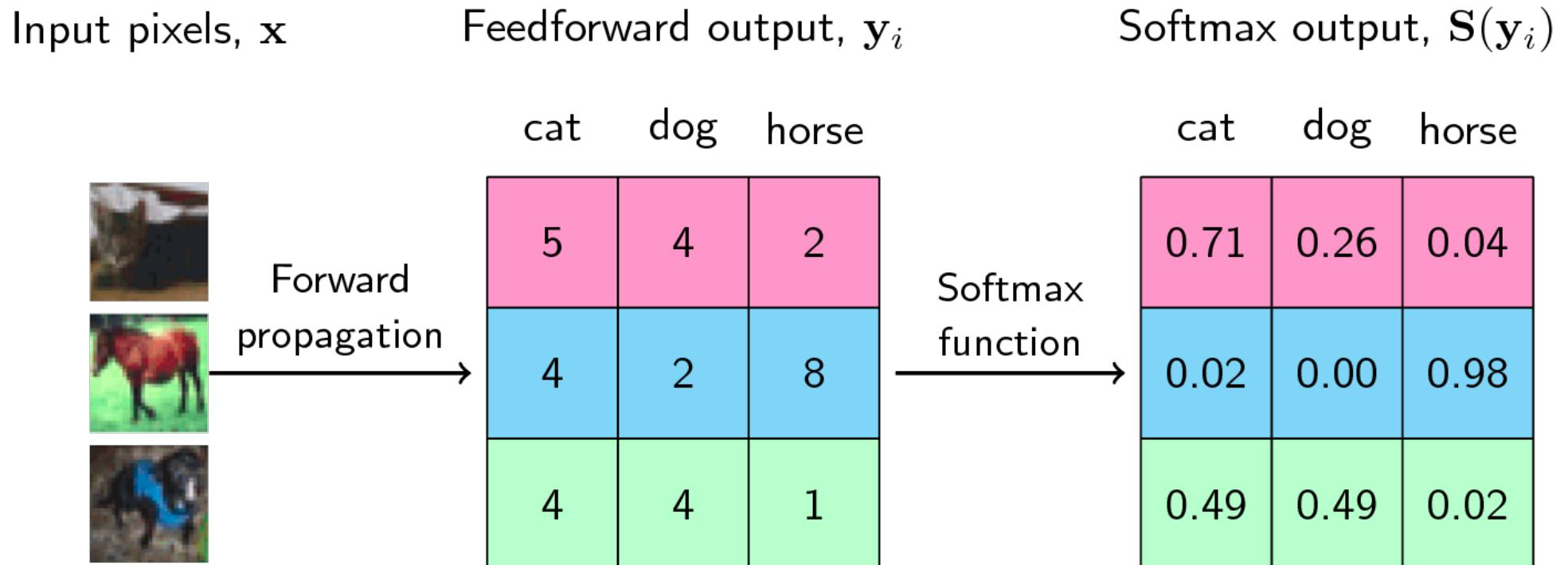


$$z_{k,i} = w_{0,i}^{(k)} + \sum_{j=1}^{d_{k-1}} g(z_{k-1,j}) w_{j,i}^{(k)}$$

Neural network concepts

Softmax activation

$$S(f_{y_i}) = \frac{e^{f_{y_i}}}{\sum_j e^{f_j}}$$



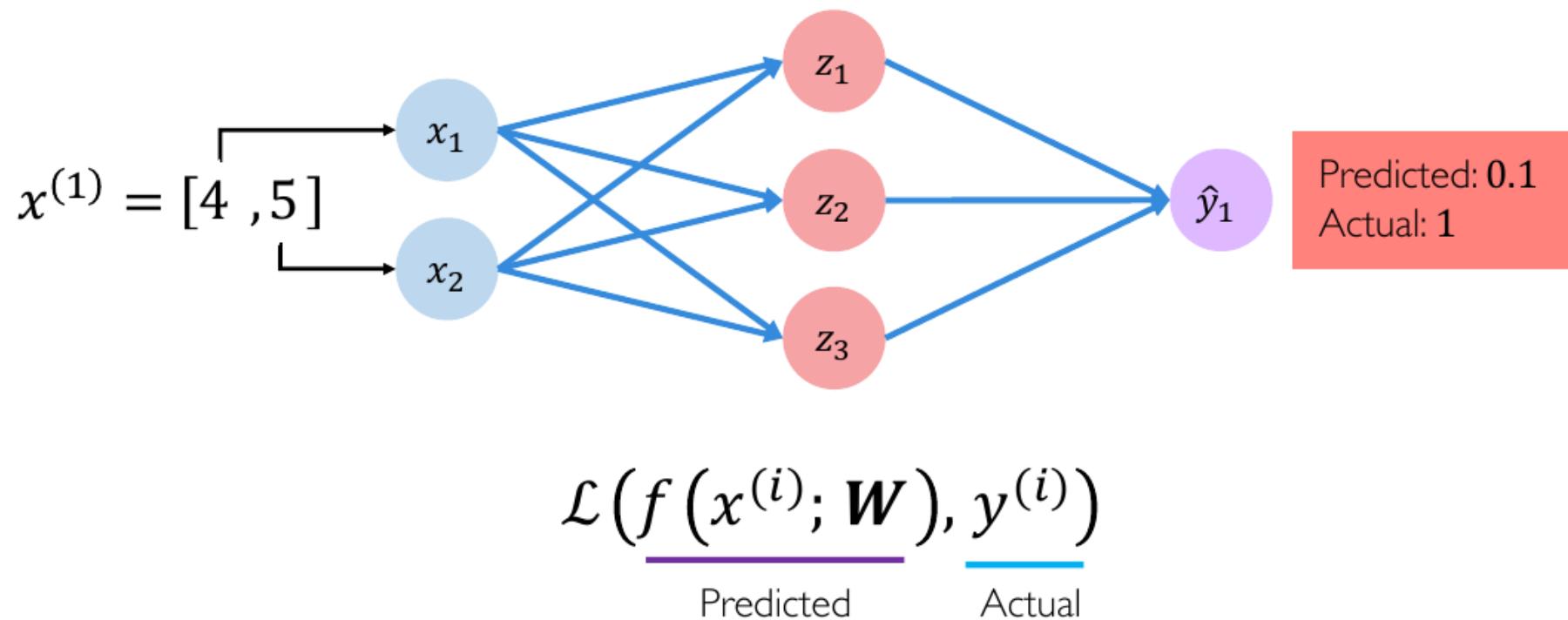
Training procedure



Training procedure

Loss function

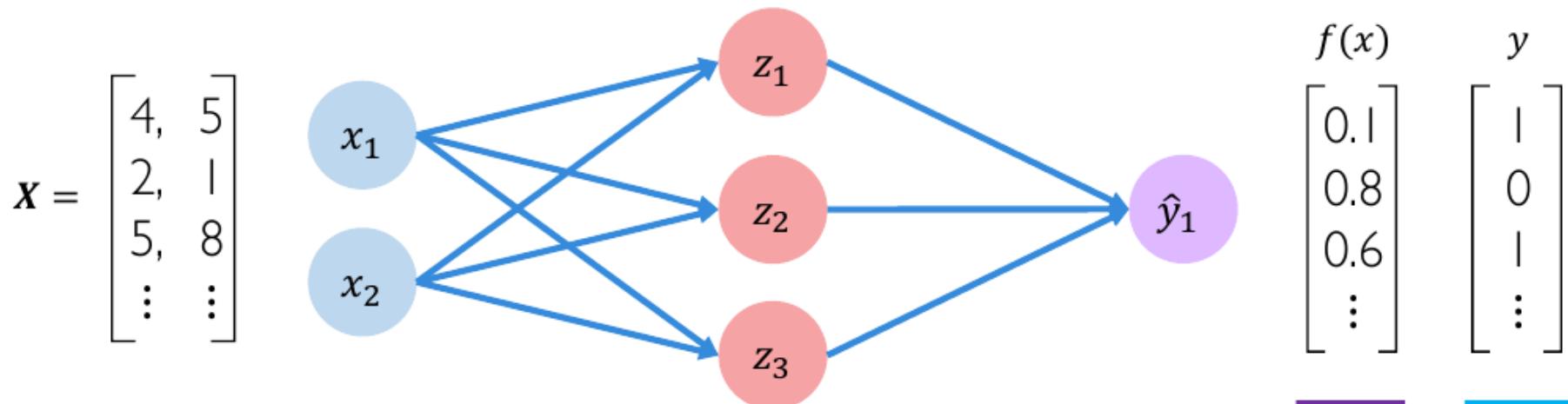
The **loss** of our network measures the cost incurred from incorrect predictions



Training procedure

Loss function

The **empirical loss** measures the total loss over our entire dataset



- Objective function
- Cost function
- Empirical Risk

$$J(\mathbf{W}) = \frac{1}{n} \sum_{i=1}^n \mathcal{L}(f(x^{(i)}; \mathbf{W}), y^{(i)})$$

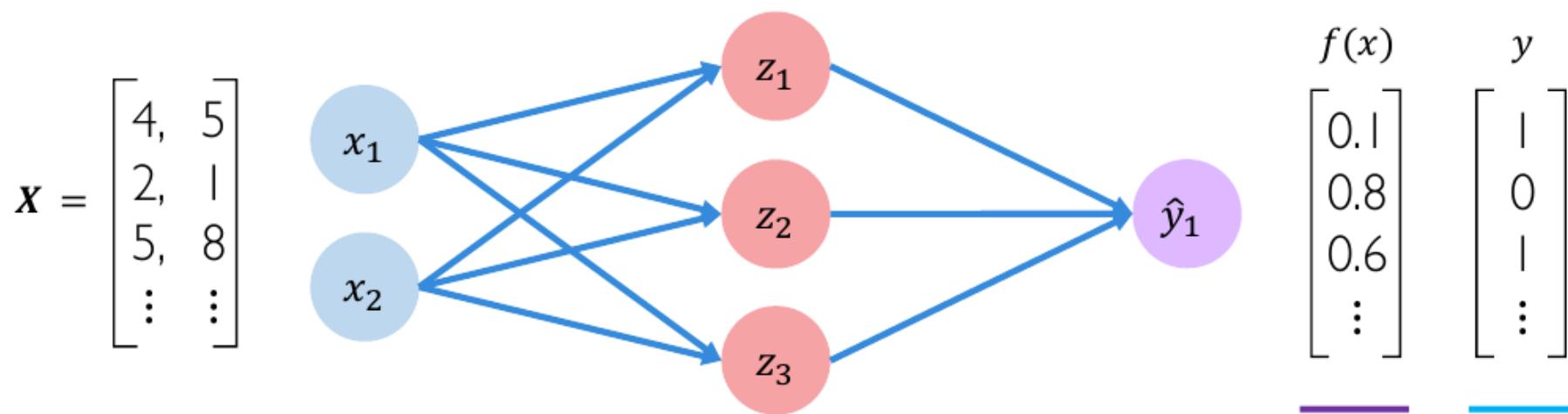
Predicted Actual



Training procedure

Binary cross-entropy loss

Cross entropy loss can be used with models that output a probability between 0 and 1

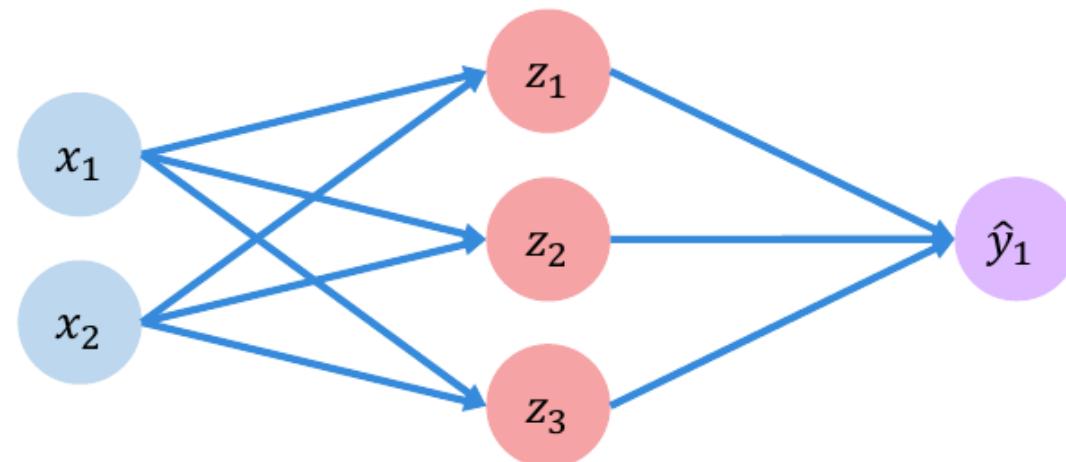


Training procedure

Mean squared error loss

Mean squared error loss can be used with regression models that output continuous real numbers

$$\mathbf{x} = \begin{bmatrix} 4, & 5 \\ 2, & 1 \\ 5, & 8 \\ \vdots & \vdots \end{bmatrix}$$



$$J(\mathbf{W}) = \frac{1}{n} \sum_{i=1}^n \left(\underline{y^{(i)}} - \underline{f(x^{(i)}; \mathbf{W})} \right)^2$$

Actual Predicted

| $f(x)$ | y |
|--|--|
| $\begin{bmatrix} 30 \\ 80 \\ 85 \\ \vdots \end{bmatrix}$ | $\begin{bmatrix} 90 \\ 20 \\ 95 \\ \vdots \end{bmatrix}$ |

Final Grades
(percentage)

Training procedure

Loss optimization

We want to find the network weights that **achieve the lowest loss**

$$\mathbf{W}^* = \operatorname{argmin}_{\mathbf{W}} \frac{1}{n} \sum_{i=1}^n \mathcal{L}(f(x^{(i)}; \mathbf{W}), y^{(i)})$$

$$\mathbf{W}^* = \operatorname{argmin}_{\mathbf{W}} J(\mathbf{W})$$



Remember:

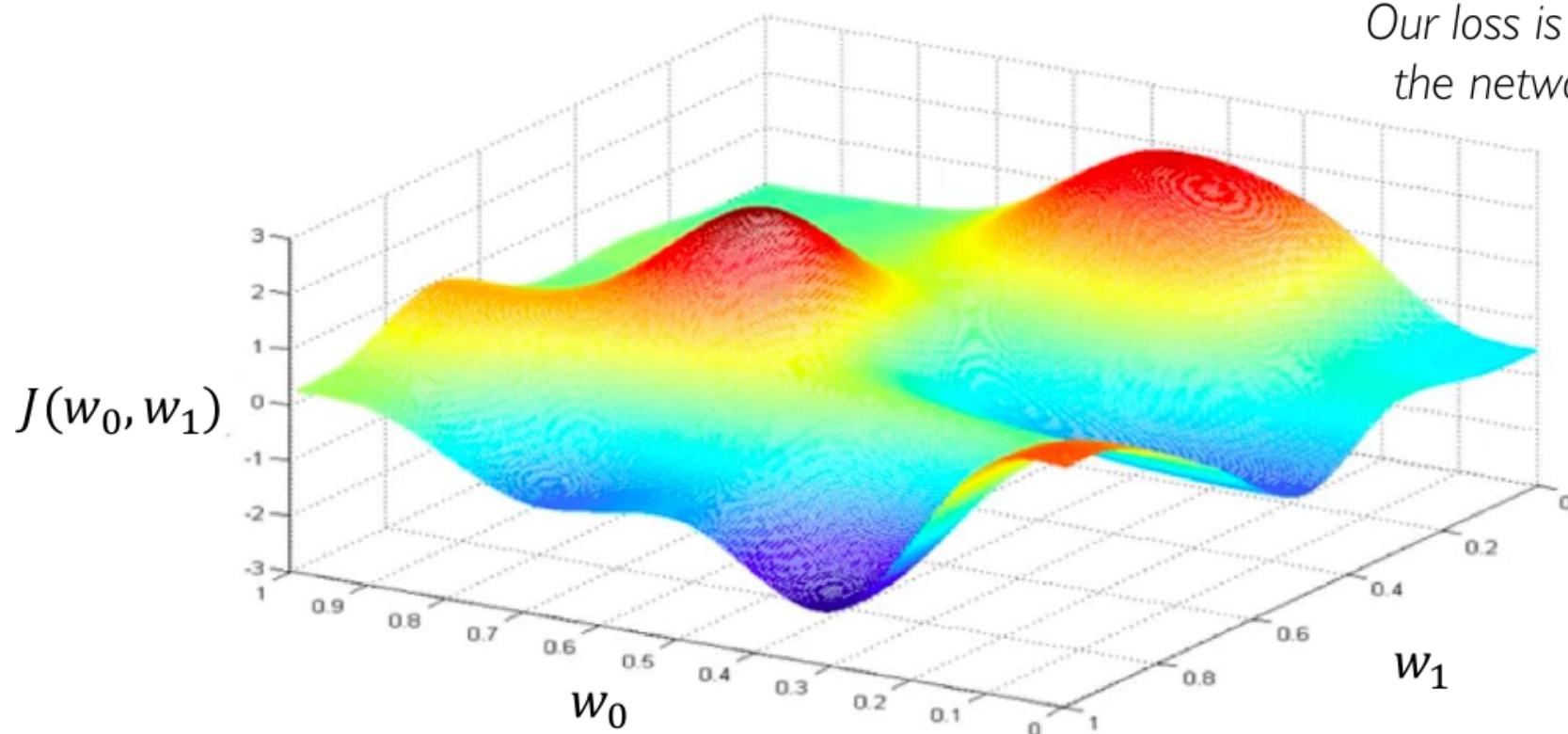
$$\mathbf{W} = \{\mathbf{W}^{(0)}, \mathbf{W}^{(1)}, \dots\}$$

Training procedure

Loss optimization

$$\mathbf{W}^* = \underset{\mathbf{W}}{\operatorname{argmin}} J(\mathbf{W})$$

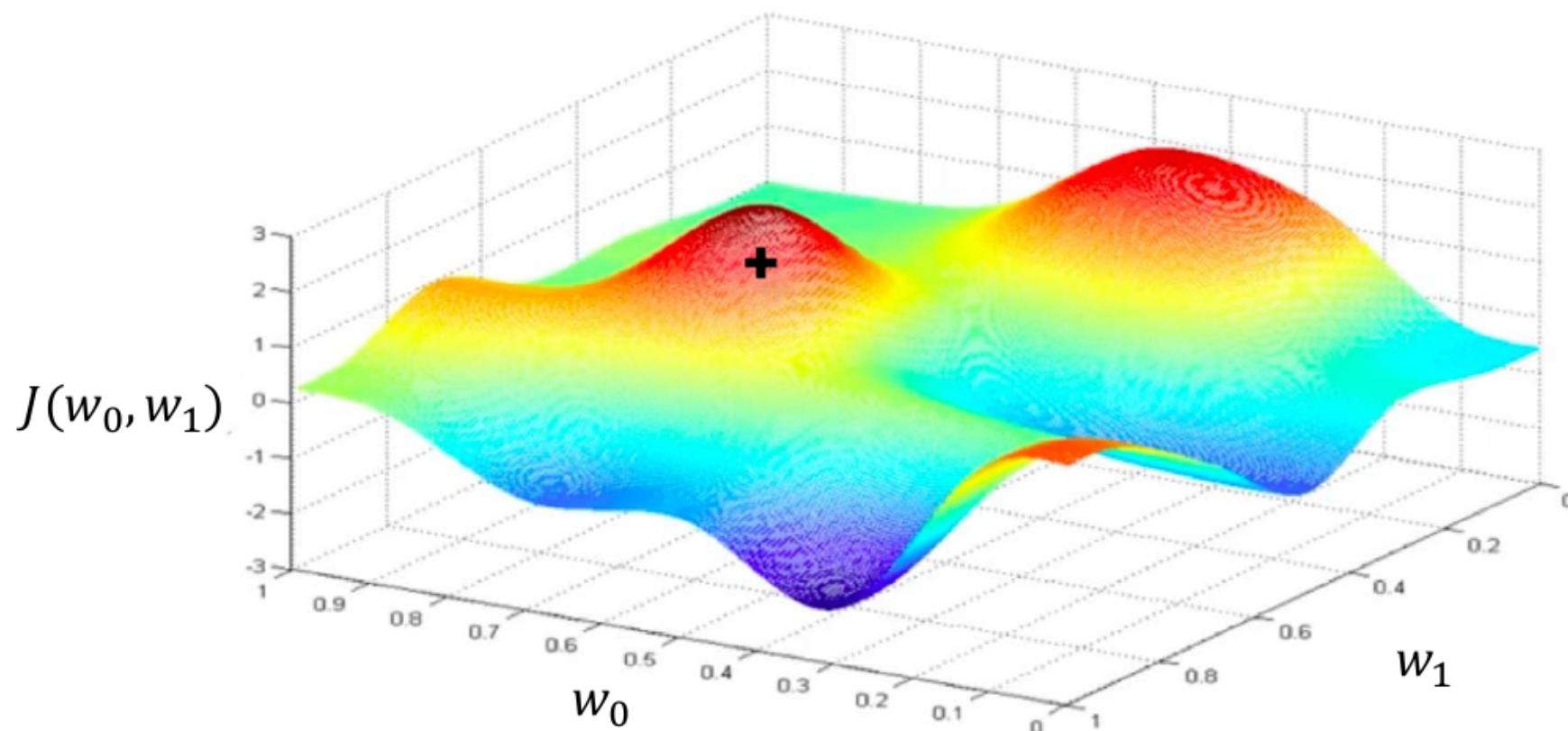
Remember:
Our loss is a function of
the network weights!



Training procedure

Gradient descent

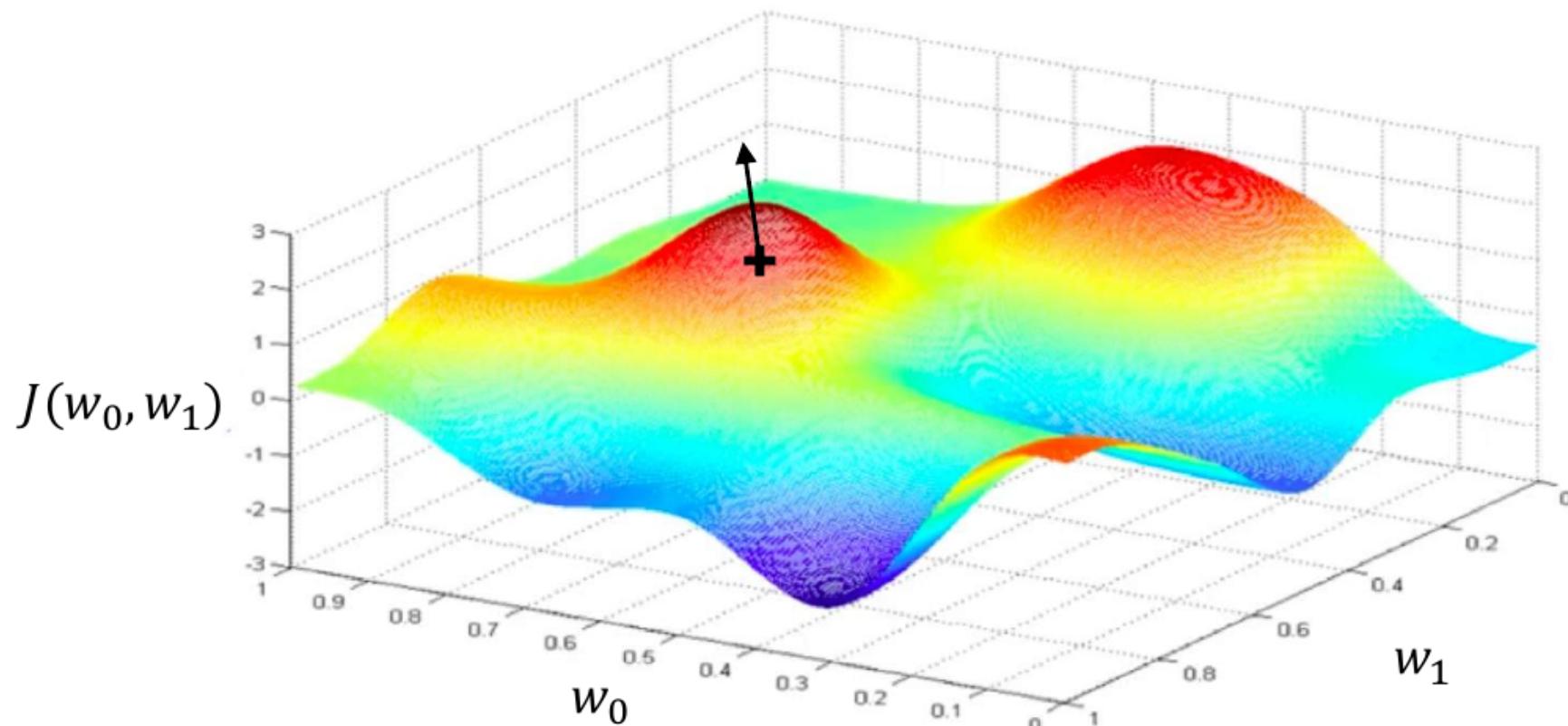
Randomly pick an initial (w_0, w_1)



Training procedure

Gradient descent

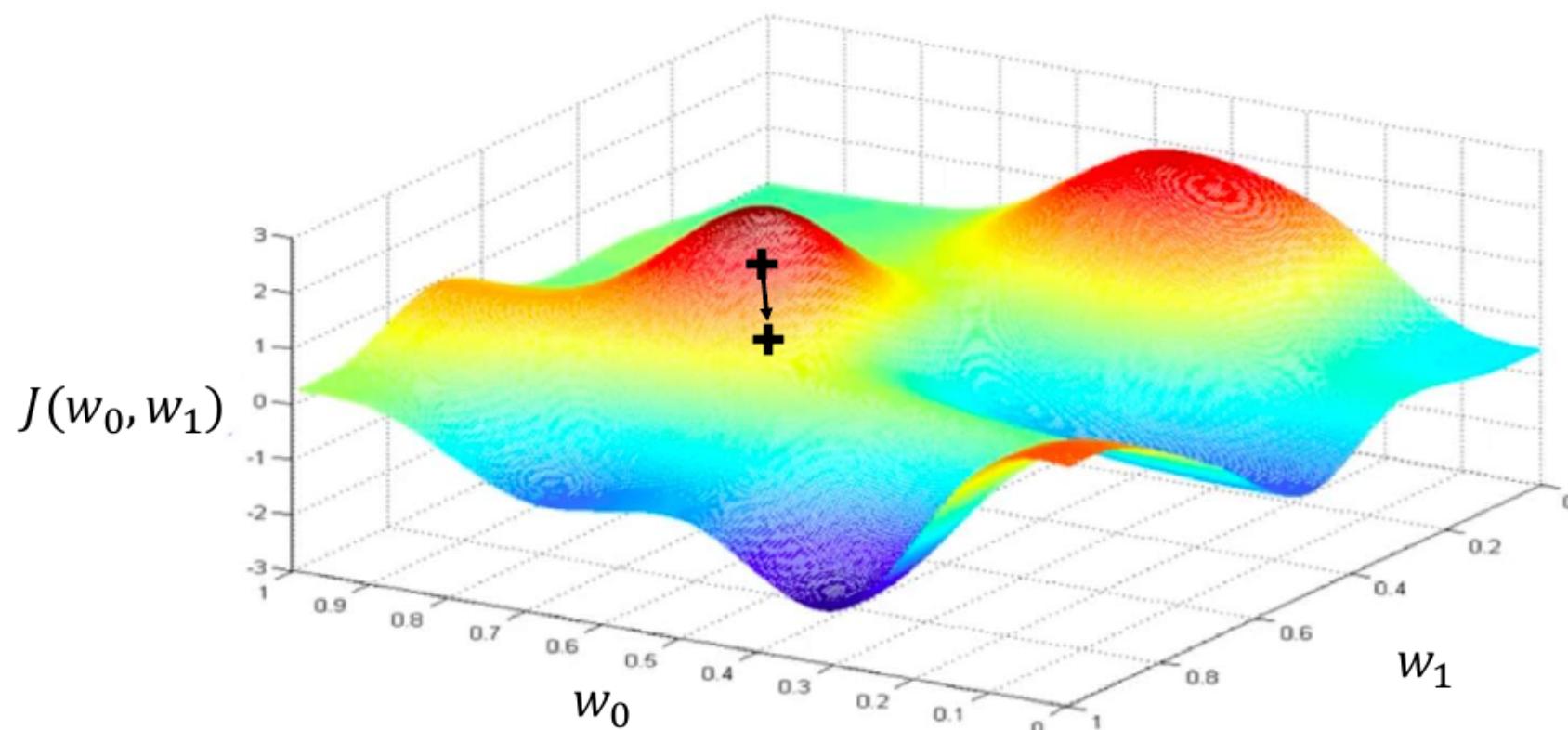
Compute gradient, $\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}}$



Training procedure

Gradient descent

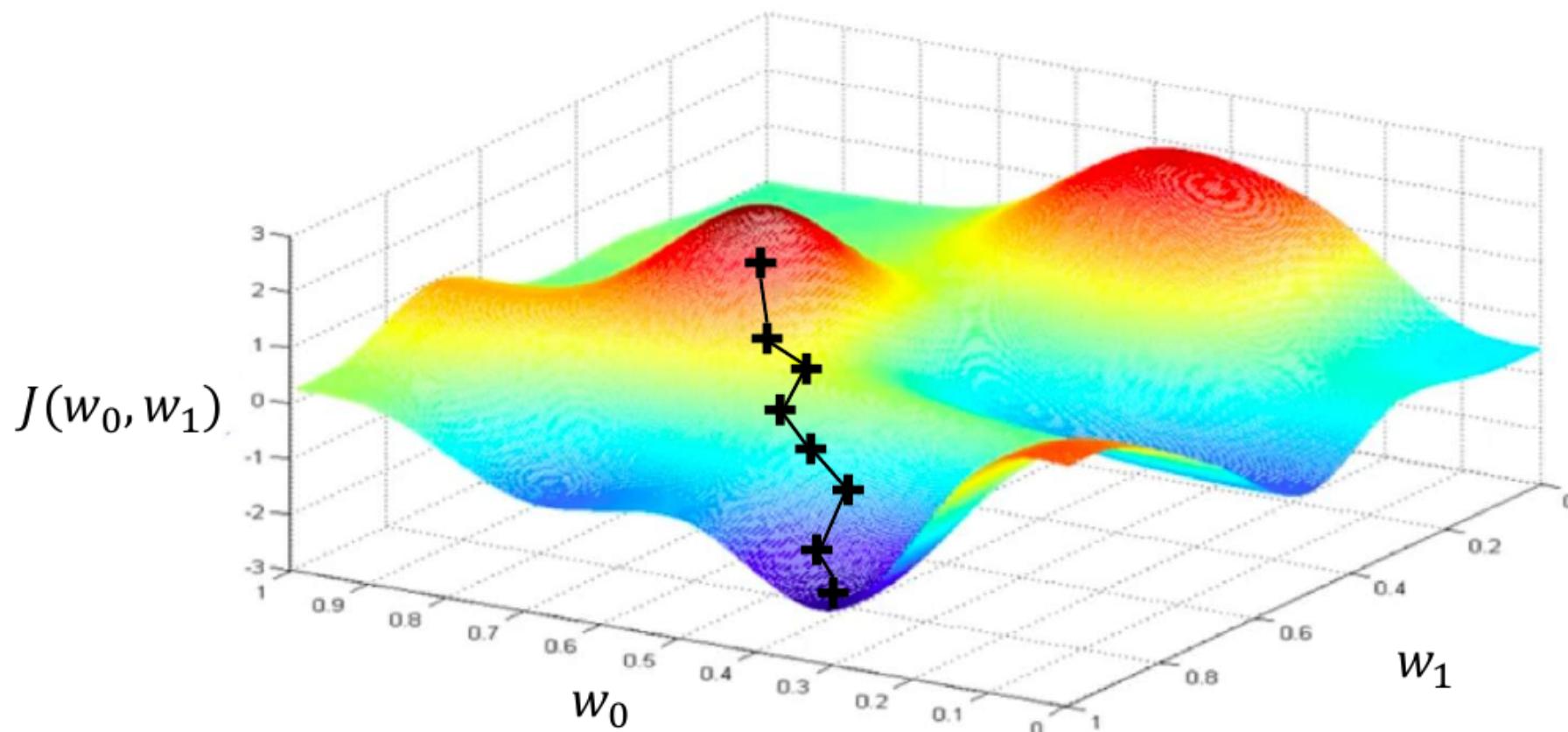
Take small step in opposite direction of gradient



Training procedure

Gradient descent

Repeat until convergence



Training procedure

Gradient descent

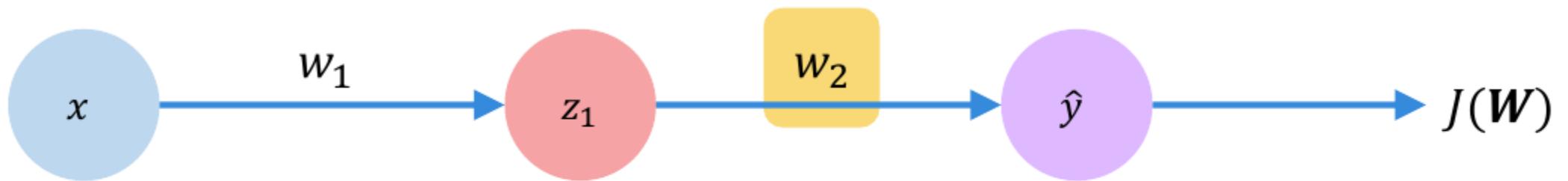
Algorithm

1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$
4. Update weights, $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$
5. Return weights



Training procedure

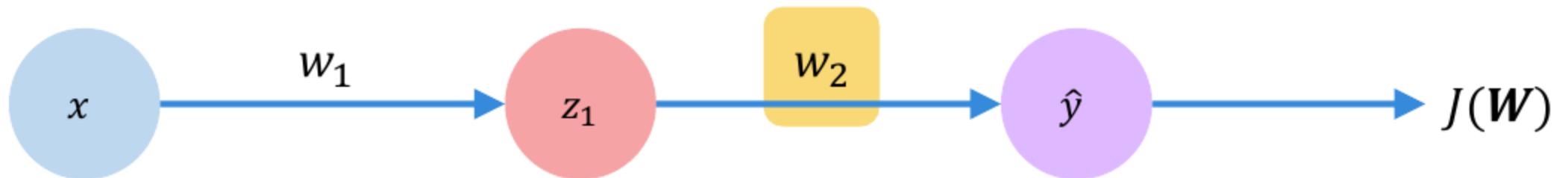
Backpropagation



How does a small change in one weight (ex. w_2) affect the final loss $J(\mathbf{W})$?

Training procedure

Backpropagation

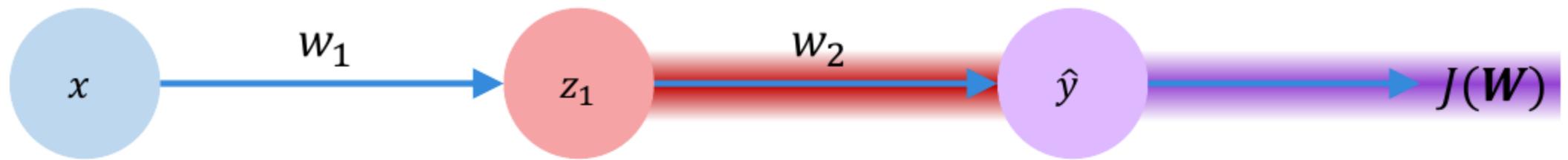


$$\frac{\partial J(\mathbf{W})}{\partial w_2} =$$

Let's use the chain rule!

Training procedure

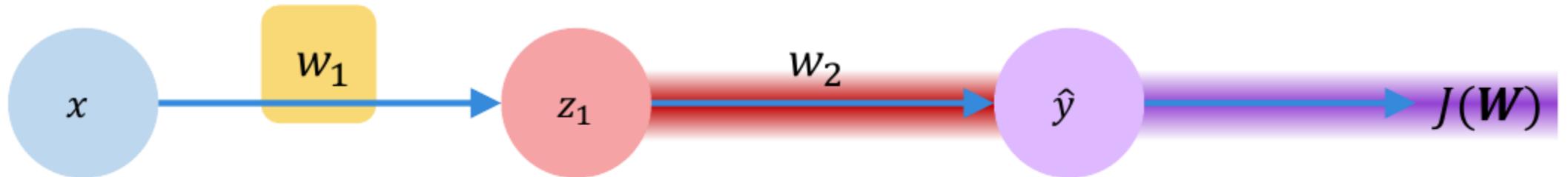
Backpropagation



$$\frac{\partial J(\mathbf{W})}{\partial w_2} = \underbrace{\frac{\partial J(\mathbf{W})}{\partial \hat{y}}}_{\text{purple bar}} * \underbrace{\frac{\partial \hat{y}}{\partial w_2}}_{\text{red bar}}$$

Training procedure

Backpropagation



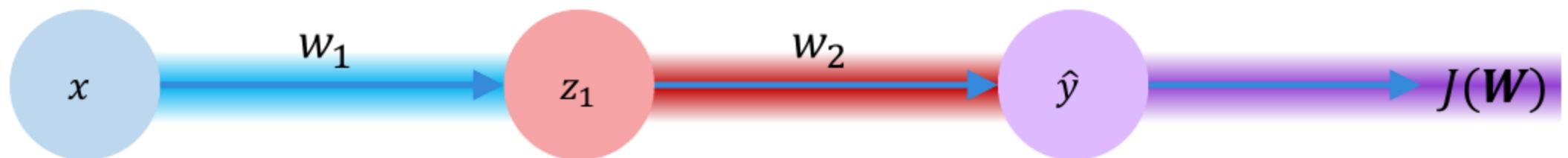
$$\frac{\partial J(\mathbf{W})}{\partial w_1} = \frac{\partial J(\mathbf{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_1}$$

Apply chain rule!

Apply chain rule!

Training procedure

Backpropagation

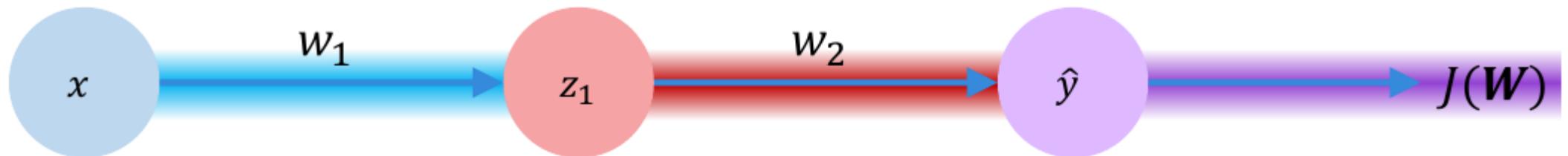


$$\frac{\partial J(\mathbf{W})}{\partial w_1} = \frac{\partial J(\mathbf{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z_1} * \frac{\partial z_1}{\partial w_1}$$



Training procedure

Backpropagation



$$\frac{\partial J(\mathbf{W})}{\partial w_1} = \underbrace{\frac{\partial J(\mathbf{W})}{\partial \hat{y}}}_{\text{purple bar}} * \underbrace{\frac{\partial \hat{y}}{\partial z_1}}_{\text{red bar}} * \underbrace{\frac{\partial z_1}{\partial w_1}}_{\text{blue bar}}$$

Repeat this for **every weight in the network** using gradients from later layers

Curriculum

Next: II. Advanced concepts

- Regularization
- Convolutional Neural Networks

