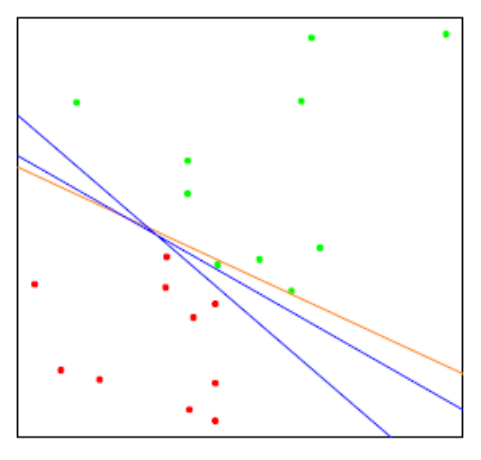


Parallel & Scalable Machine Learning

Introduction to Machine Learning Algorithms

Dr. –Ing. Gabriele Cavallaro

Postdoctoral Researcher

High Productivity Data Processing Group

Juelich Supercomputing Centre

Lecture 4 - 17/02/2020

ARTIFICIAL NEURAL NETWORKS

JÜLICH Forschungszentrum | JÜLICH SUPERCOMPUTING CENTRE

HELMHOLTZ AI | ARTIFICIAL INTELLIGENCE COOPERATION UNIT

PRACE PARTNERSHIP FOR ADVANCED COMPUTING IN EUROPE TRAINING PORTAL

UNIVERSITY OF ICELAND SCHOOL OF ENGINEERING AND NATURAL SCIENCES FACULTY OF INDUSTRIAL ENGINEERING, MECHANICAL ENGINEERING AND COMPUTER SCIENCE

DEEP Projects

COURSE OUTLINE

- Parallel and Scalable Machine Learning Driven by HPC
- Introduction to Machine Learning Fundamentals
- Supervised Learning with a Simple Learning Model
- Artificial Neural Networks (ANNs)
- Introduction to Statistical Learning Theory
- Validation and Regularization
- Pattern Recognition Systems
- Parallel and Distributed Training of ANN
- Supervised Learning with Deep Learning
- Unsupervised Learning – Clustering
- Clustering with HPC
- Introduction to Deep Reinforcement Learning

OUTLINE

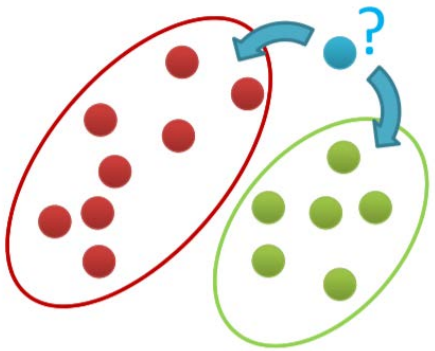
- Lecture 3 Revisited
  - Supervised learning
  - Multi-Output Perceptron Model
- Artificial Neural Networks (ANNs)
- Activation functions
- Backpropagation algorithm
- Practicals with ANNs
- Metrics for performance evaluation

MACHINE LEARNING

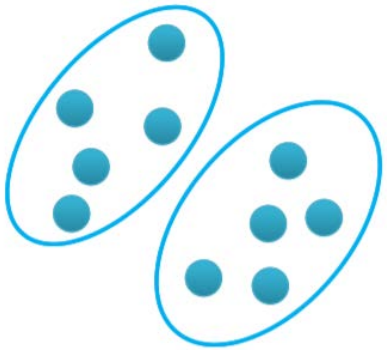
Form of Learning

- **Supervised learning:** correct responses for input data are given
  - "teacher" signal, correct "outcomes", "labels" for the data
  - Classic frameworks: **classification**, **regression**

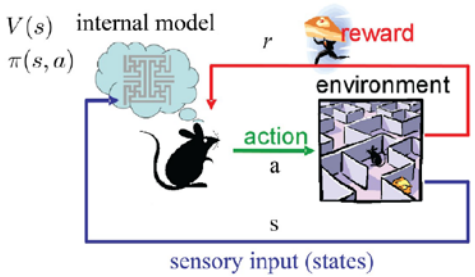
    THIS LECTURE



- **Unsupervised learning:** only data are given
  - Find "hidden" structure, patterns
  - Classical frameworks: **clustering**, **dimensionality reduction**



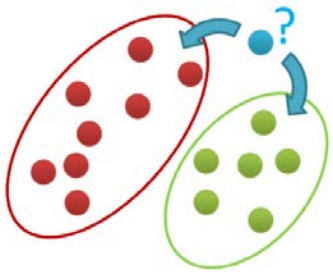
- **Reinforcement learning:** data including (sparse) **reward** $\mathrm{r}(\boldsymbol{X})$
  - Discover actions $\boldsymbol{a}$ that minimize total future reward R
  - **Active** learning: experience depends on choice of $\boldsymbol{a}$

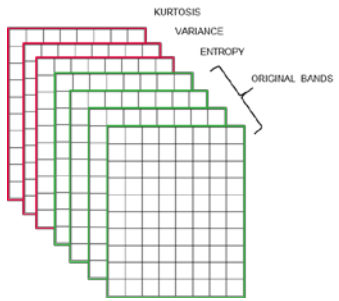


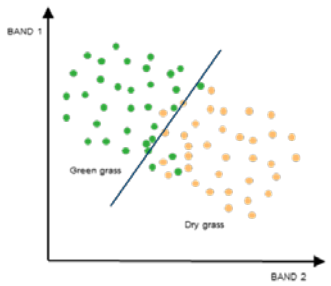
SUPERVISED CLASSIFICATION

Revisited

- The **design** of a supervised classifier depends on:
  - The available set of **features** (e.g., pattern spectra)
  - The available set of **annotated training samples**
  - The typology of the adopted **classification model**
  - The **cost function** to optimize







Let $\boldsymbol{x} = \{x_1, x_2, \dots, x_n\}$ be a **vector** defined in a n -dimensional feature space

Let $\boldsymbol{y} = \{C_1, C_2, \dots, C_m\}$ be the set of m **informative classes**

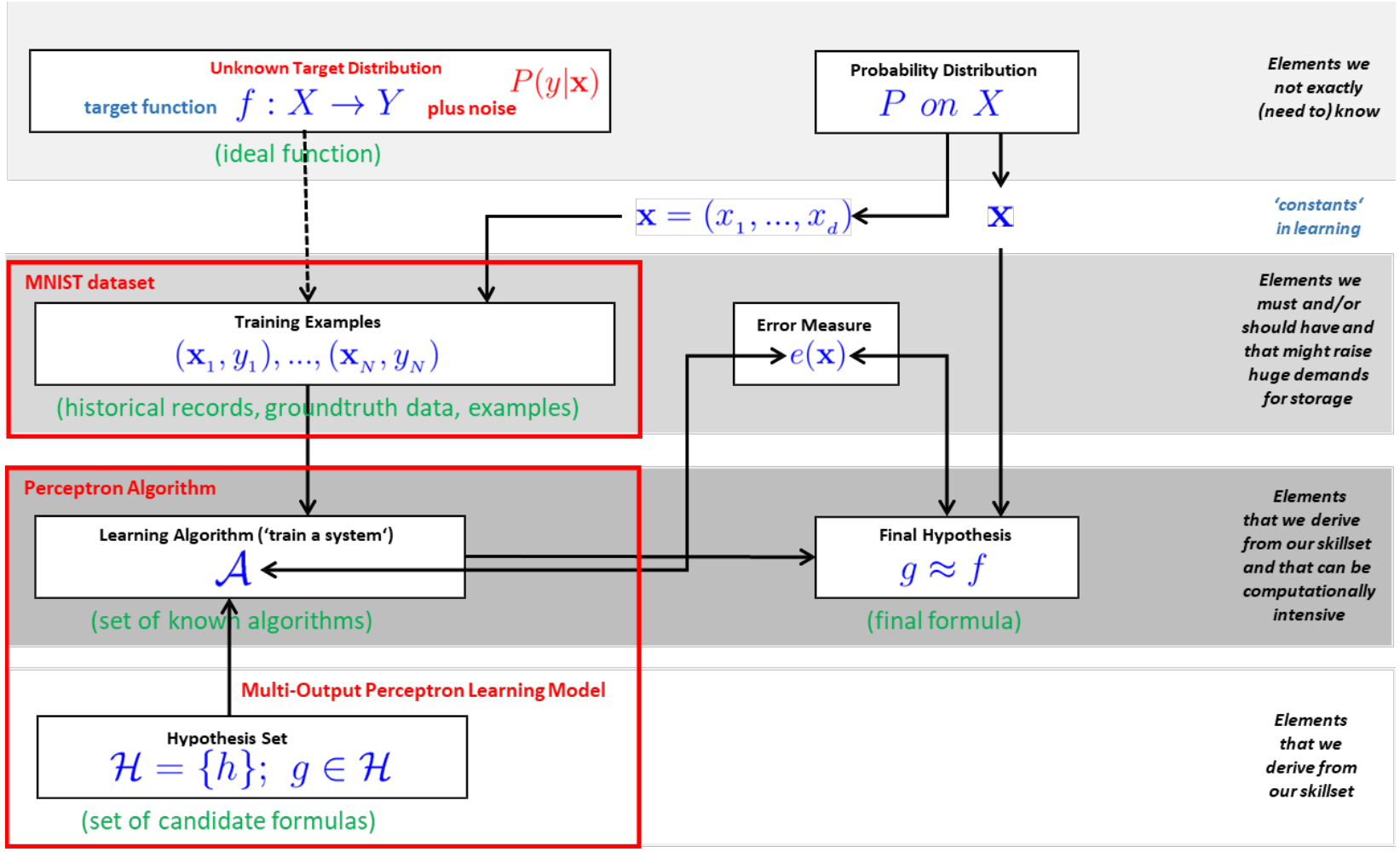
Let $\boldsymbol{X}: (\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \dots, (\boldsymbol{x}_N, y_N)$ be a set of N **training samples (annotated training set)**

- **Annotated training samples** permit the integration of **prior knowledge** in the classifier design

- Supervised classification aims at associating to each pattern $\boldsymbol{x}$ a class that optimizes a predefined **decision criterion**

SUPERVISED CLASSIFICATION

Lecture 3 - Revisited



MULTI OUTPUT PERCEPTRON MODEL

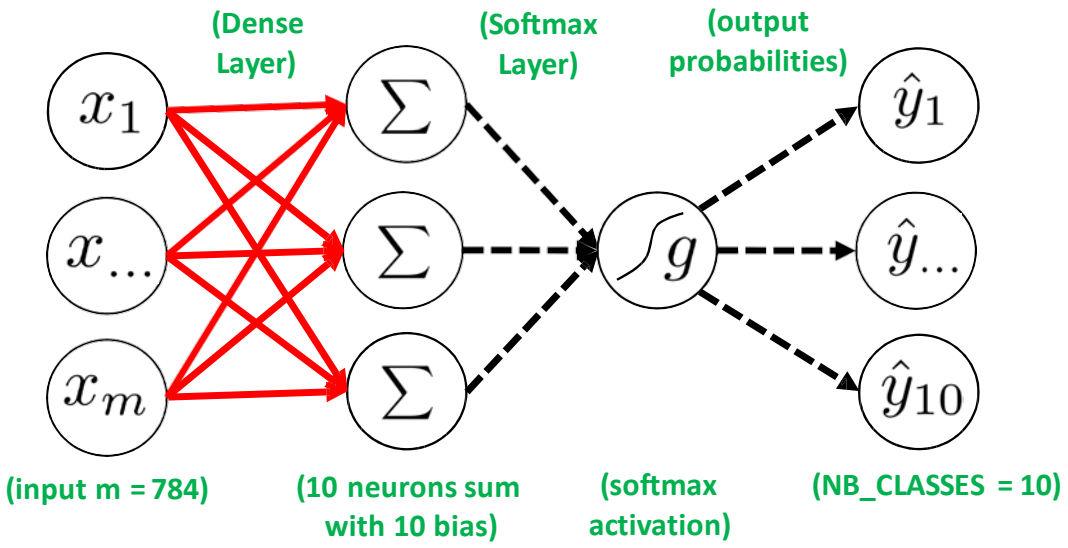
Lecture 3 - Revisited

```
Epoch 7/20
60000/60000 [==============================] - 2s 26us/step - loss: 0.4419 - acc: 0.8838
Epoch 8/20
60000/60000 [==============================] - 2s 26us/step - loss: 0.4271 - acc: 0.8866
Epoch 9/20
60000/60000 [==============================] - 2s 25us/step - loss: 0.4151 - acc: 0.8888
Epoch 10/20
60000/60000 [==============================] - 2s 26us/step - loss: 0.4052 - acc: 0.8910
Epoch 11/20
60000/60000 [==============================] - 2s 26us/step - loss: 0.3968 - acc: 0.8924
Epoch 12/20
60000/60000 [==============================] - 2s 25us/step - loss: 0.3896 - acc: 0.8944
Epoch 13/20
60000/60000 [==============================] - 2s 26us/step - loss: 0.3832 - acc: 0.8956
Epoch 14/20
60000/60000 [==============================] - 2s 25us/step - loss: 0.3777 - acc: 0.8969
Epoch 15/20
60000/60000 [==============================] - 2s 25us/step - loss: 0.3727 - acc: 0.8982
Epoch 16/20
60000/60000 [==============================] - 1s 24us/step - loss: 0.3682 - acc: 0.8989
Epoch 17/20
60000/60000 [==============================] - 1s 25us/step - loss: 0.3641 - acc: 0.9001
Epoch 18/20
60000/60000 [==============================] - 1s 25us/step - loss: 0.3604 - acc: 0.9007
Epoch 19/20
60000/60000 [==============================] - 2s 25us/step - loss: 0.3570 - acc: 0.9016
Epoch 20/20
60000/60000 [==============================] - 1s 24us/step - loss: 0.3538 - acc: 0.9023
```

```
# model evaluation
score = model.evaluate(X_test, Y_test, verbose=VERBOSE)
print("Test score:", score[0])
print('Test accuracy:', score[1])
```

```
10000/10000 [==============================] - 0s 41us/step
Test score: 0.33423959468007086
Test accuracy: 0.9101
```

✓ **Multi Output Perceptron: ~91,01% (20 Epochs)**



- **How to improve the model design by extending the neural network topology?**
- **Which layers are required?**
- **Think about input layer need to match the data – what data we had?**
- **Maybe hidden layers?**
- **How many hidden layers?**
- **What activation function for which layer (e.g. maybe ReLU)?**
- **Think Dense layer – Keras?**
- **Think about final Activation as Softmax → output probability**

DIFFERENT MODELS

Hypothesis Set and Choosing a Model with more Capacity

Hypothesis Set

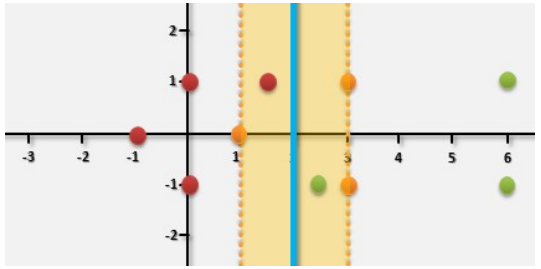
$$\mathcal{H} = \{h\}; \;\; g \in \mathcal{H}$$

$$\mathcal{H} = \{h_1, \ldots, h_m\};$$

(all candidate functions derived from models and their parameters)

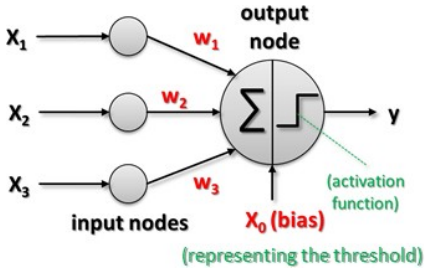
- **Choosing from various model approaches $h_1, \ldots, h_m$ is a different hypothesis**
- **Additionally a change in model parameters of $h_1, \ldots, h_m$ means a different hypothesis too**
- **The model capacity characterized by the VC Dimension helps in choosing models**
- **Occam's Razor rule of thumb: 'simpler model better' in any learning problem, not too simple!**

h_1



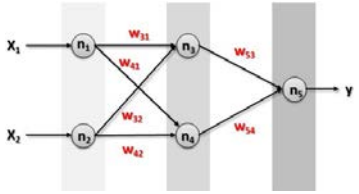
(e.g. support vector machine model)

h_2



(e.g. linear perceptron model)

h_m



(e.g. artificial neural network model)

'select one function' that best approximates

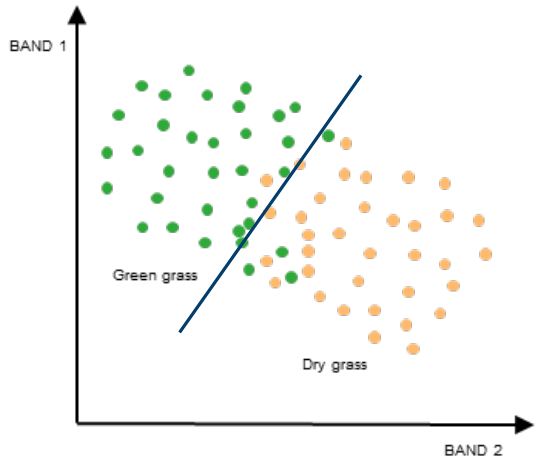
Final Hypothesis

$$g \approx f$$

SUPERVISED CLASSIFICATION

Discriminant Functions

- The values of $\boldsymbol{w}$ of a **discriminant function** can be computed by using information of annotated **training set**
- In the **training phase**, the best discriminative function (which minimizes the number of errors) according to the adopted **cost function** is found

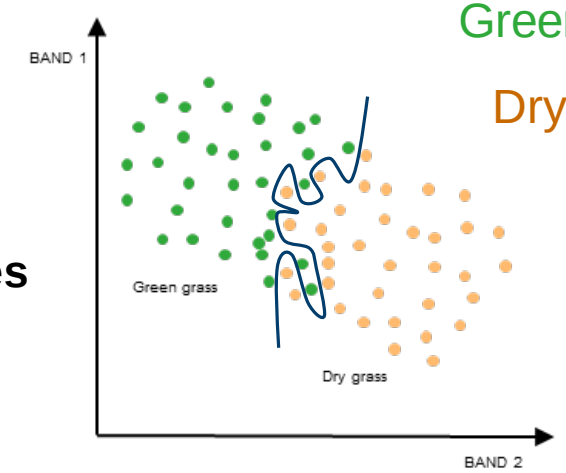


- With **linear classifier**
  - Sometimes not sufficient to separate the classes

- Witn **Non Linear classifier**
  - Complex models can lead to complicated decision boundaries
  - They may lead to perfect classification of our training samples
  - What about **generalization**?
  - Important: do not tune excessively (**overfit**) the classifier on **training samples**
  - Leave room for a correct classification of **novel patterns (samples)**

Green grass → C_1

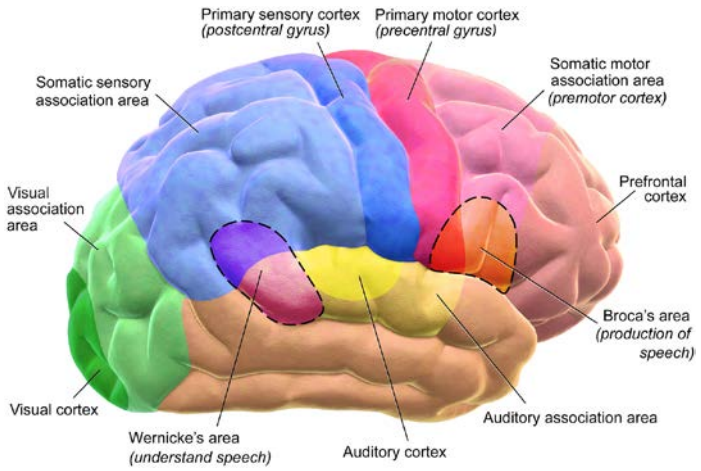
Dry grass → C_2



BIOLOGICAL NEURAL NETWORKS

Human Brain

- Some machine learning systems are **inspired** by the functioning of the **human brain**
- The human brain is capable
  - To solve problems of different natures and with **high** degree of **complexity**
  - In a very **short time**



[1] Human brain

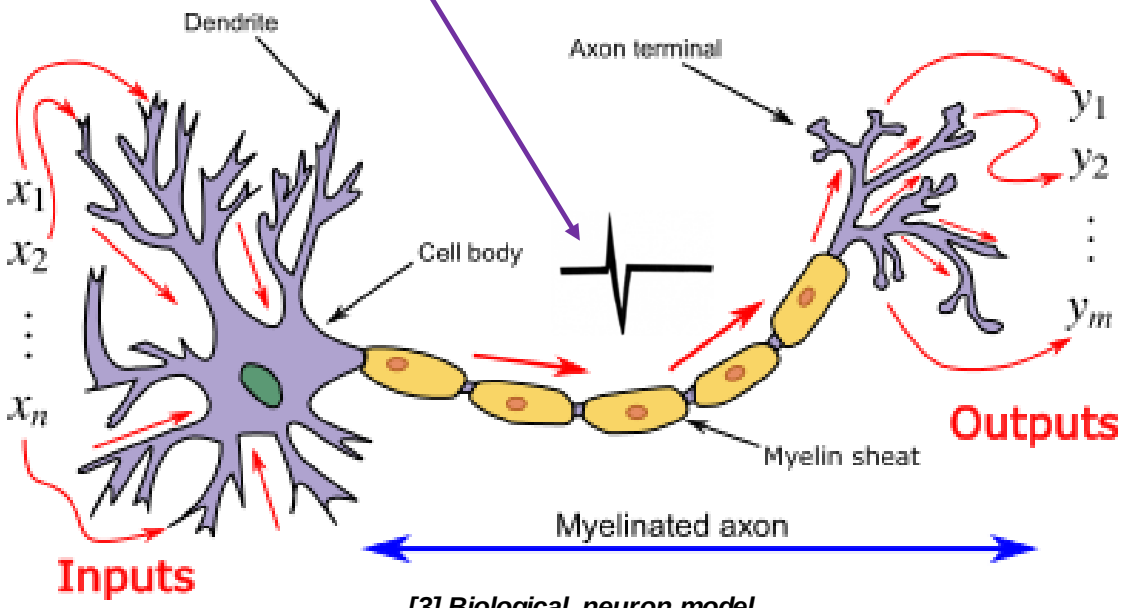
[2] Facebook's artificial intelligence

NEURON

Biological Inspiration for Computation

- **Neuron**: computational building block for the brain
- **(Artificial) Neuron:** computational building block for the **neural network**

- If the neuron is enough stimulated, a **spike** is sent down to the axon (i.e., transmission line)
  - After this, the neuron rest (refractory period)



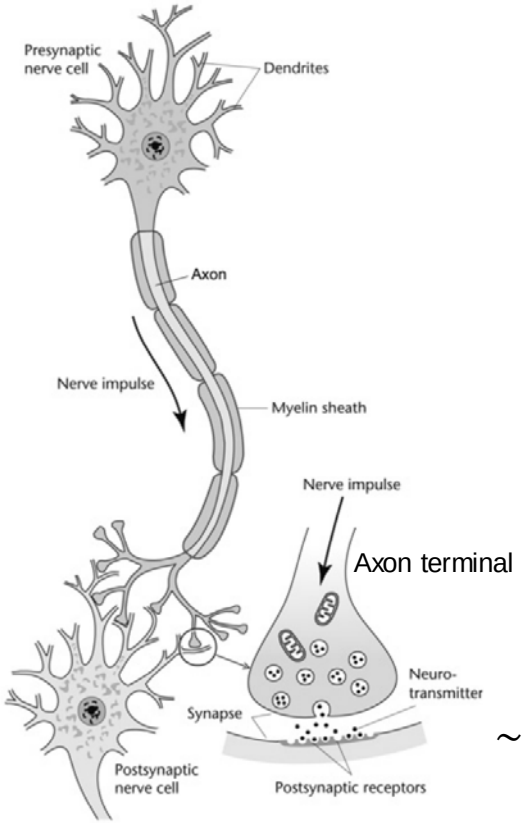
[3] Biological neuron model

$\sim 10^{11}$ billion neurons in the human brain

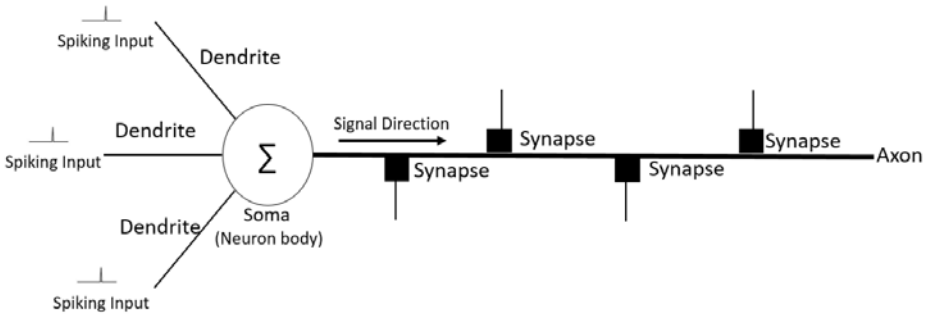
NEURON

Biological Inspiration for Computation

- **Synaptic gap**: neurons are no directly connected
- When the **axon** is stimulated, it dumps **neuro transmitters** into the synaptic gap



$\sim 10^5$ synapses per neuron

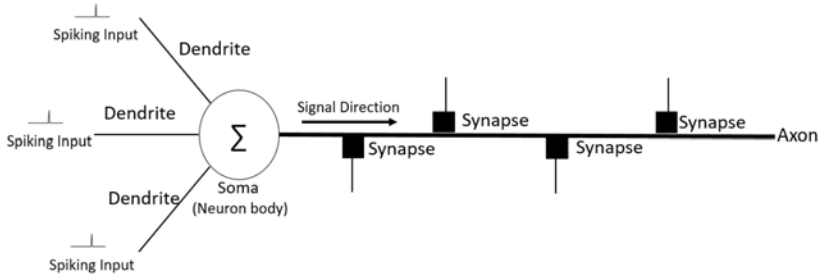


- **Dendrite**: the organ that receives spiking signals from other neurons
- **Soma (neuron body):** generates/sends spiking signals to the axon
  - Under the condition of the integration of received spiking signal levels exceeds a specific threshold voltage
- **Axon:** propagates spiking signals to other neurons
- **Synapse:** a space between the axon of the pre-synapse neuron and dendrite of the postsynapse neuron
  - It is widely considered as a memory organ in the brain by storing the memory information in its connectivity strength

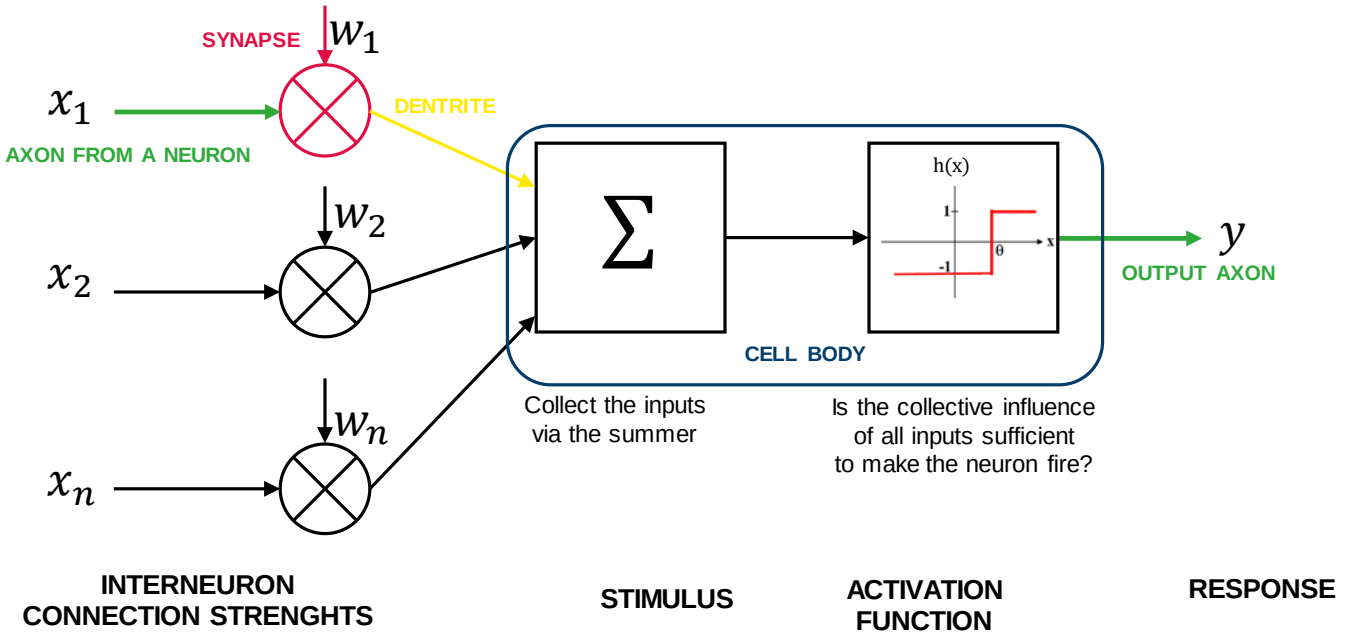
[4] L. Anderberg et al.

ARTIFICIAL NEURON

Biological Inspiration for Computation



[4] L. Anderberg et al.



- ''Model'' the synaptic connections: weights with multipliers
  - A connection might be either strong or loose
  - If strong the weight w increases its value
- ''Model'' the cumulative effect of all the inputs to the neuron by a summer
  - Decide if it is going to be an all-or-node 1 by running it via a threshold function
  - If the weighed sum is higher than the threshold T , then fire 1, otherwise 0

$$\boxed{y = g(\boldsymbol{x}, \boldsymbol{w}, \theta)}$$

ARTIFICIAL NEURAL NETWORK (ANN)

Introduction

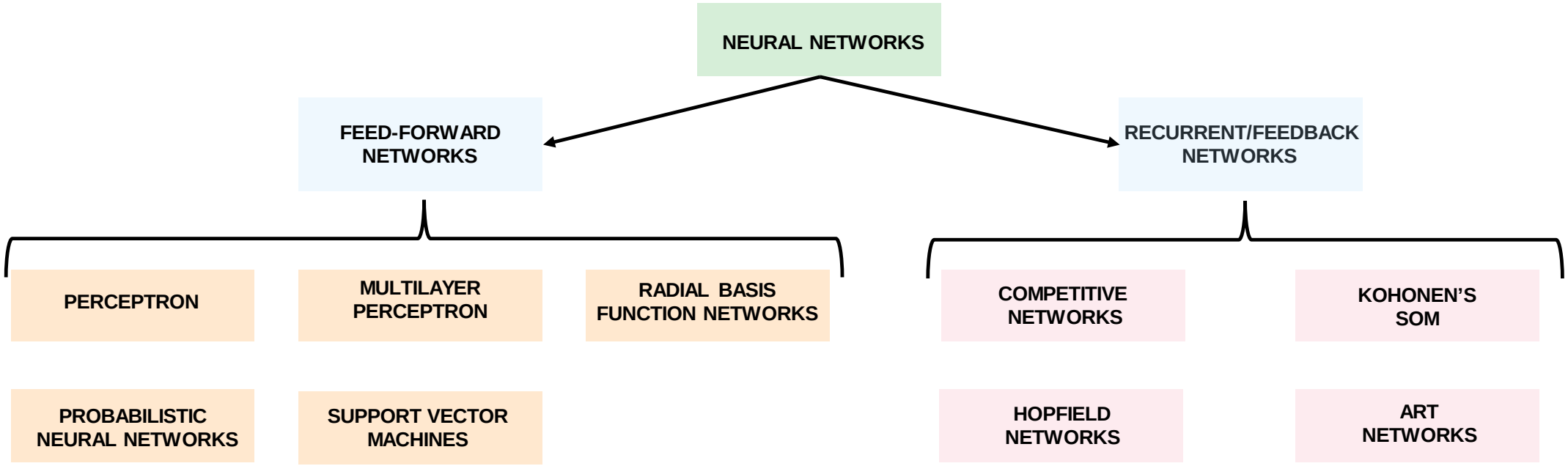
- Research in ANNs is marked by three key evolution periods:
  - 1940's: *McCulloch* and *Pitts* developed studies on neural networks
  - 1960's: *Rosenblatt* formulated the **convergence theorem** for the **perceptron**
    - *Minsky* underlined the limitations of the perceptron
  - 1980's: interest is renewed thanks to the works of *Hopfield* and the development of the error **backpropagation** algorithm to train **artificial multilayer perceptron neural networks**

- ANNs
  - Are **non linear** systems
  - Allow to learn **mappings** from a given input space to a desired output space
  - Can have different architectures according to the way neurons are **connected** to each other

ARTIFICIAL NEURAL NETWORKS

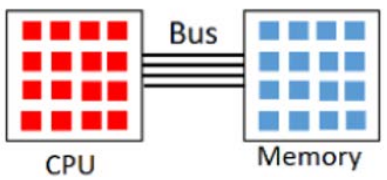
Main Architectures

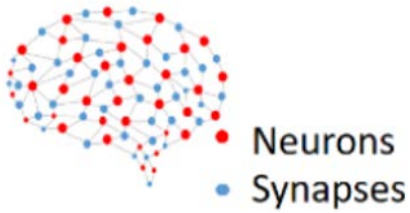
- **Feed forward neural networks:** neurons are organized in layers with unidirectional connections
  - Typically they are static and stateless networks (they provide only one set of output values for each input)
- **Recurrent neural network:** neurons can have bidirectional connections
  - They are dynamic systems with status (the output depends on state)



ARTIFICIAL NEURON

Biological Inspiration for Computation





- Differences (among others):
  - **Parameters**: human brains have ~ 10,000,000 times synapses than artificial neural networks
  - **Topology**: human brains have no “layers” (the topology is complicated)
  - **Async:** the human brain works asynchronously (ANNs work synchronously)
  - **Learning algorithm:** ANNs use gradient descent for learning. Human brains use … (we don’t know)
  - **Processing speed:** single biological neurons are slow, while standard neurons in ANNs are fast
  - **Power consumption:** biological neural networks use very little power compared to ANNs
  - **Stages**: biological networks usually don't stop/ start learning. ANNs have different train and prediction stages

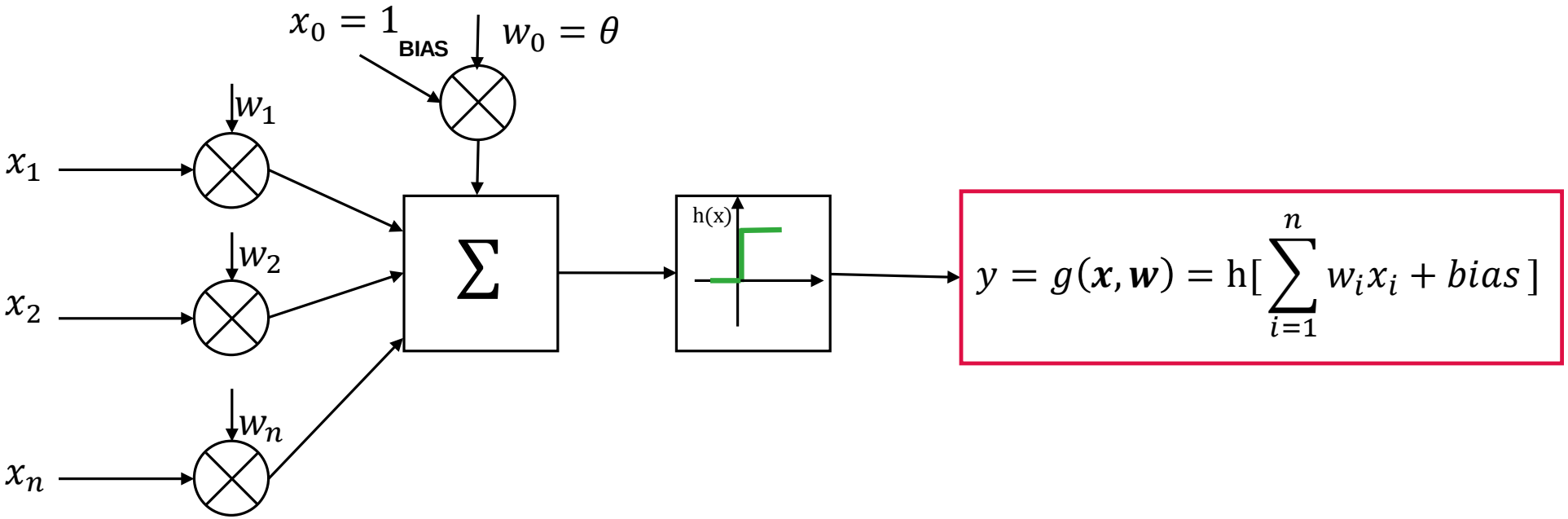
- Similarity (among others):
  - Distributed computation on large scale

[5] MIT 6.S094

PERCEPTRON

Architecture and Bias

- Remove the dependency on the threshold by adding a weight $w_0 = \theta$
  - This takes the threshold θ and it moves it back to zero
  - Connected to an input that is always +1



$$y = g(\boldsymbol{x}, \boldsymbol{w}) = \mathrm{h}[\sum_{i=1}^{n} w_i x_i + bias]$$

PERCEPTRON

Algorithm

- **Objective**: estimate the parameters (**weights**) of the linear **discriminant function**
- **Input**: training set $D: (\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \dots, (\boldsymbol{x}_N, y_N)$
- **Output**: the vector of the weights $\boldsymbol{w}$ which allows the correct classification of each sample $\boldsymbol{x_i}$
- **Perceptron algorithm**: each pattern $\boldsymbol{x}$ is transformed into the variable $\boldsymbol{y}$ according to the following rule

$$\boldsymbol{y} \in \begin{cases} 0 & g(\boldsymbol{x}, \boldsymbol{w}) \geq 0 \\ 1 & g(\boldsymbol{x}, \boldsymbol{w}) < 0 \end{cases}$$

- It can only be used to implement **linearly separable functions**

PERCEPTRON

OR Problem

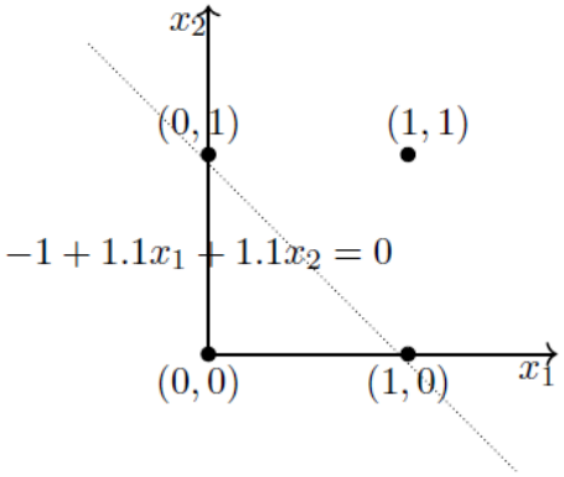
$$y = g(\boldsymbol{x}, \boldsymbol{w}) = \mathrm{h}[\sum_{i=1}^{n} w_i x_i + bias\,]$$

x_1	x_2	OR	
0	0	0	$w_0 + \sum_{i=1}^{2} w_i x_i < 0$
1	0	1	$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$
0	1	1	$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$
1	1	1	$w_0 + \sum_{i=1}^{2} w_i x_i \geq 0$

$$w_0 + w_1 \cdot 0 + w_2 \cdot 0 < 0 \implies w_0 < 0$$
$$w_0 + w_1 \cdot 0 + w_2 \cdot 1 \geq 0 \implies w_2 > -w_0$$
$$w_0 + w_1 \cdot 1 + w_2 \cdot 0 \geq 0 \implies w_1 > -w_0$$
$$w_0 + w_1 \cdot 1 + w_2 \cdot 1 \geq 0 \implies w_1 + w_2 > -w_0$$

One possible solution is

$w_0 = -1,\ w_1 = 1.1,\ w_2 = 1.1$



$$y \in \begin{cases} 0 & g(\boldsymbol{x}) = -1 + 1.1x_1 + 1.1x_2 < 0 \\ 1 & g(\boldsymbol{x}) = -1 + 1.1x_1 + 1.1x_2 \geq 0 \end{cases}$$

[6] Perceptron Learning Algorithm

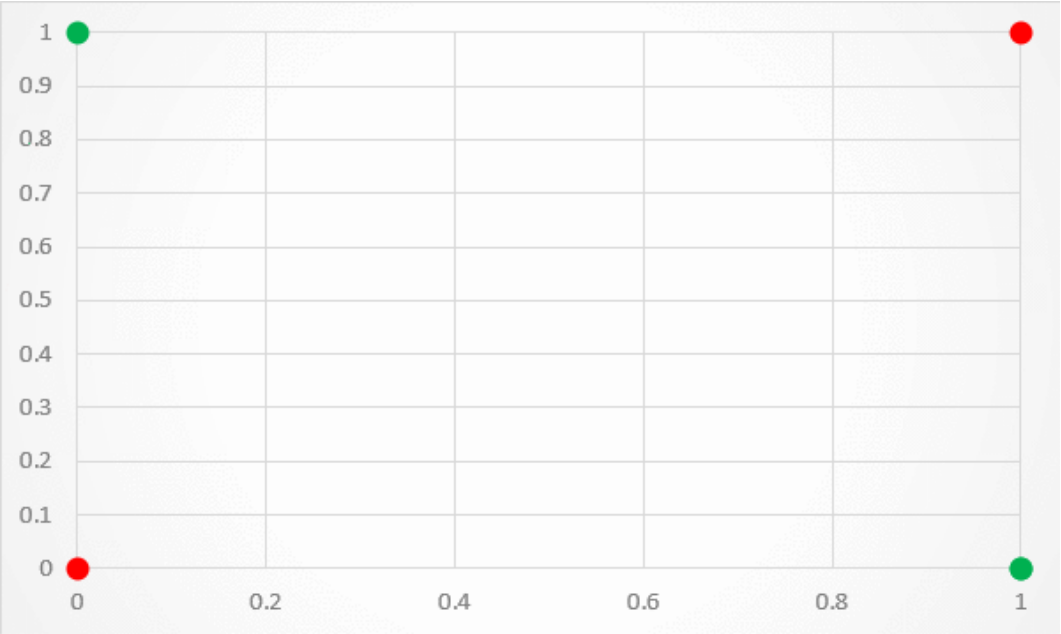
PERCEPTRON

XOR Problem

- Perceptron is only capable of separating data points with a single line
- XOR inputs are not linearly separable
  - There is no way to separate the 1 and 0 predictions with a single classification line

Input 1	Input 2	Output
0	0	0
0	1	1
1	1	0
1	0	1

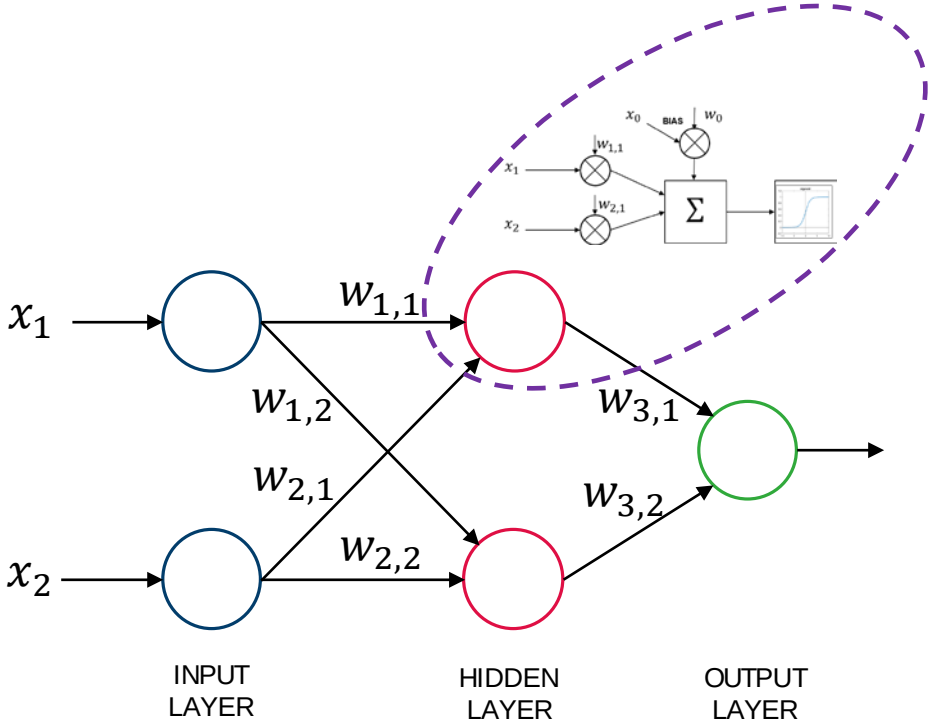
[7] The XOR Problem

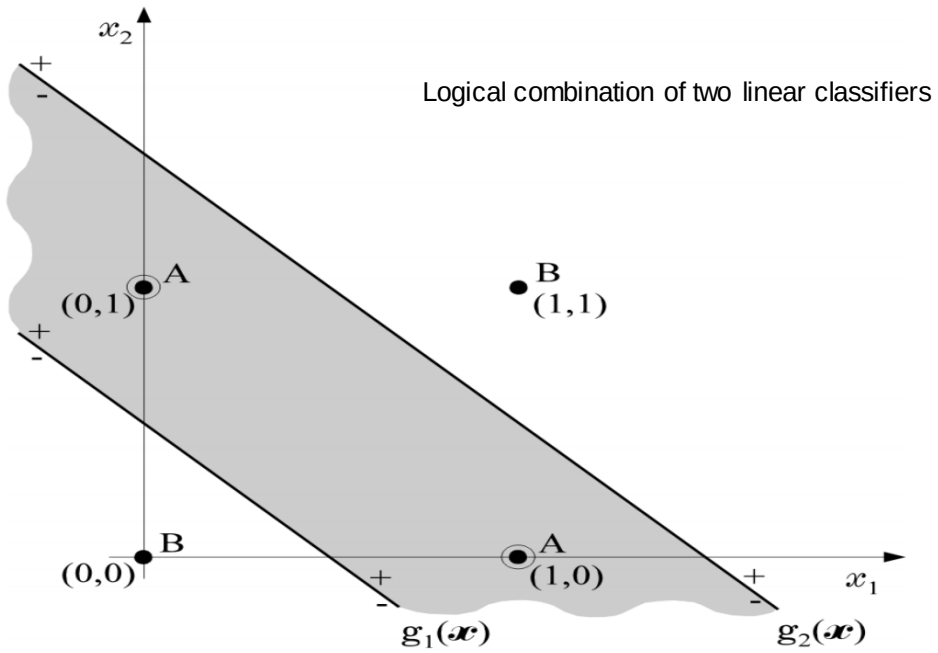


MULTILAYER PERCEPTRON (MLP)

XOR Problem

- The XOR problem can be solved by a **three layers** network
  - Hidden layer: additional layer of units without any direct access to the outside world



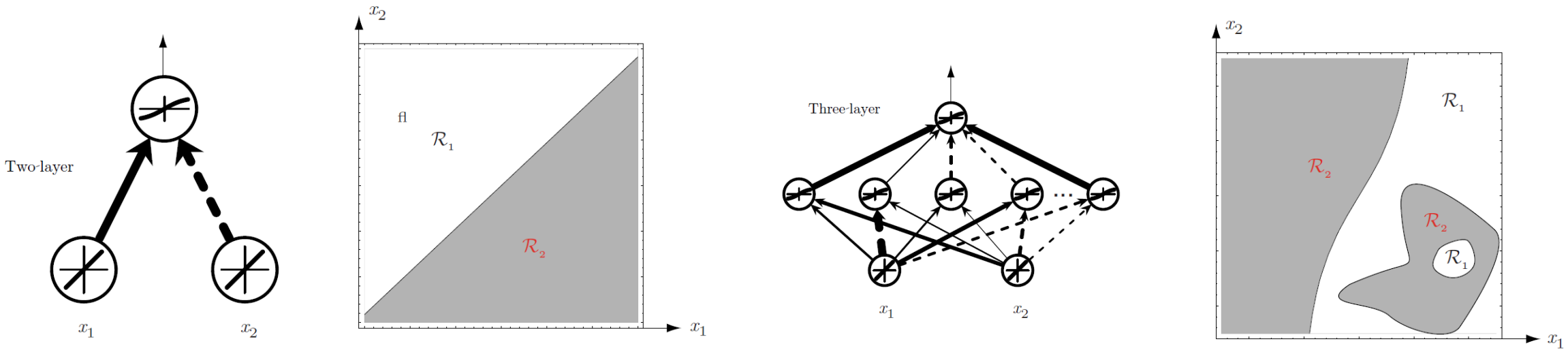


[8] Multilayer Perceptron

MULTILAYER PERCEPTRON (MLP)

Decision Boundaries

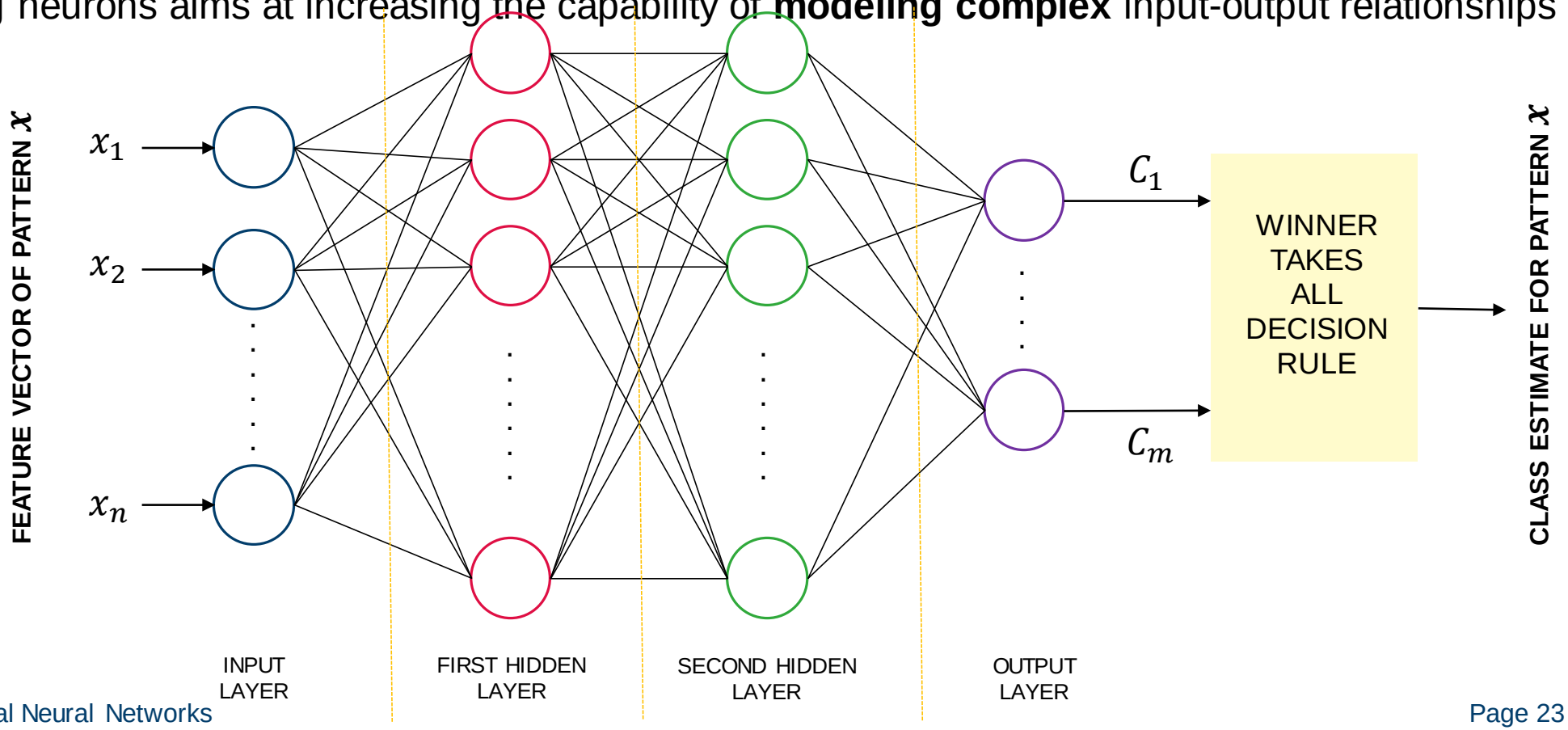
- A **two-layer** network classifier can only implement a **linear decision boundary**
- With a sufficient **number of hidden units**, networks can implement **arbitrary decision boundaries**
  - The decision regions need not be convex, nor simply connected



[9] R. O. Duda et al.

MULTILAYER PERCEPTRON (MLP) NEURAL NETWORK

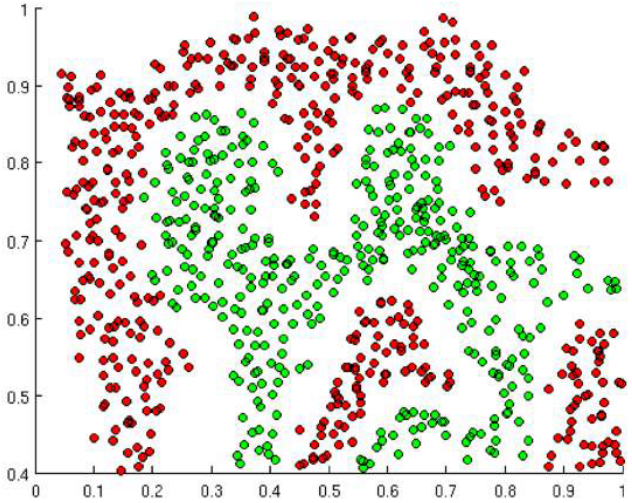
- **Forward interconnection** of several layers of perceptrons
- MLPs can be used as **universal approximators**
- In classification problems, they allow modeling **nonlinear discriminant functions**
- Interconnecting neurons aims at increasing the capability of **modeling complex** input-output relationships

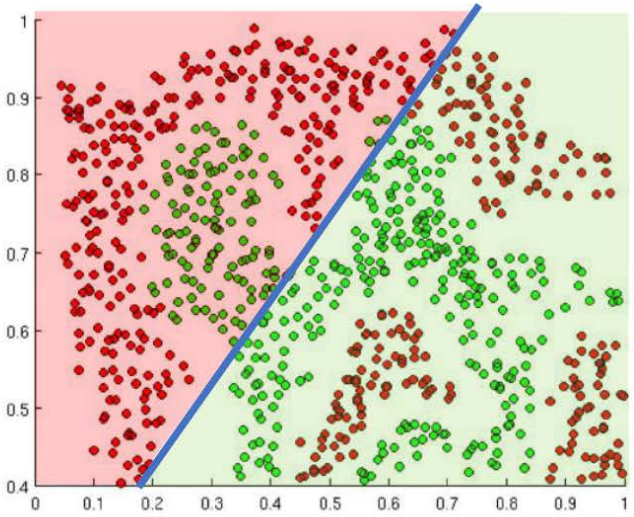


ACTIVATION FUNCTIONS

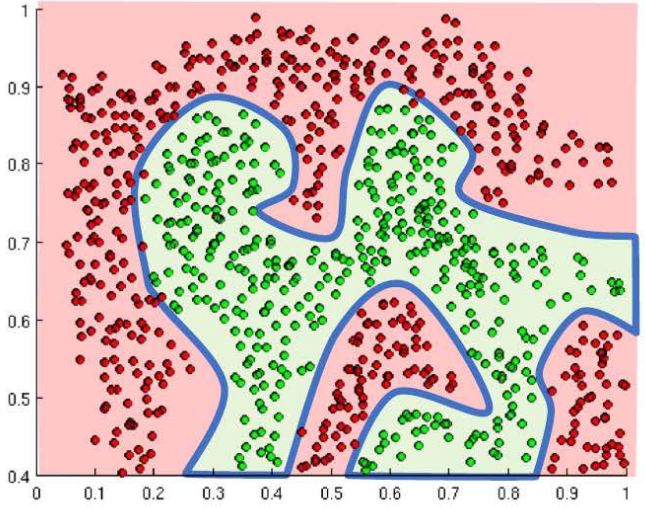
Introduce non-linearities into the network

- E.g., How to build a Neural Network to distinguish green vs red points?





Linear Activation functions produce linear decisions no matter the network size

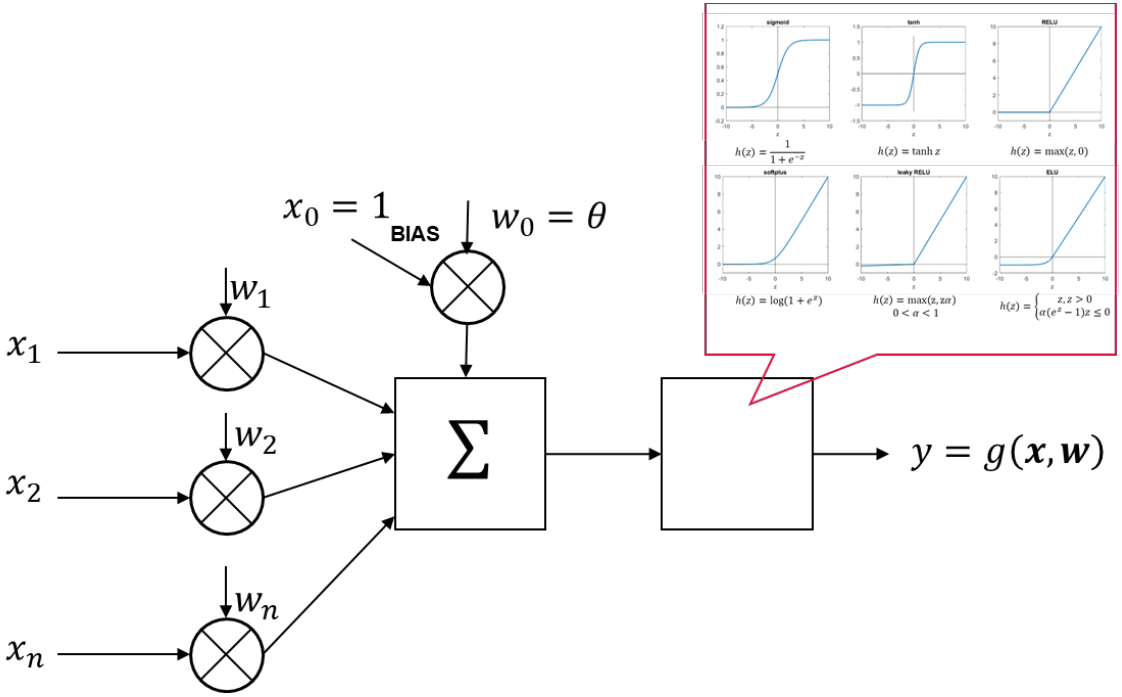


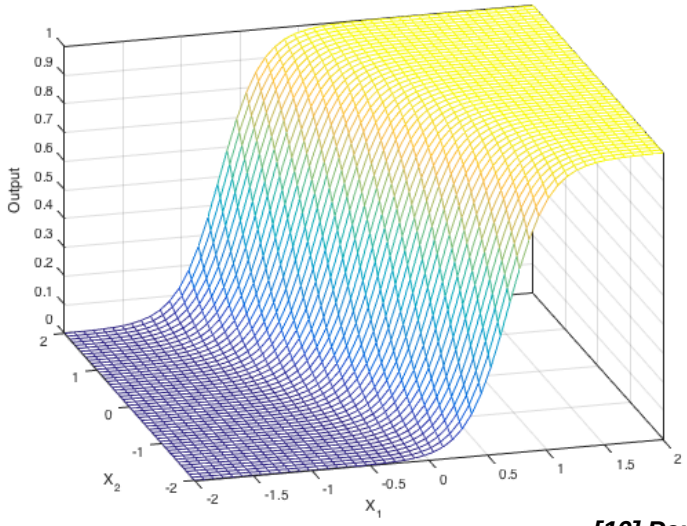
Non-linearities allow us to approximate arbitrarily complex functions

PERCEPTRON WITH NON LINEAR ACTIVATION FUNCTIONS

Can perform nonlinear classification?

- Perceptron **can't perform nonlinear** classification **regardless** of the choice of **activation function**
  - The input is projected onto the weight vector and scaled/shifted along this direction
  - This is a linear operation that reduces the input to a single value
  - Which is then passed through the (possibly nonlinear) activation function





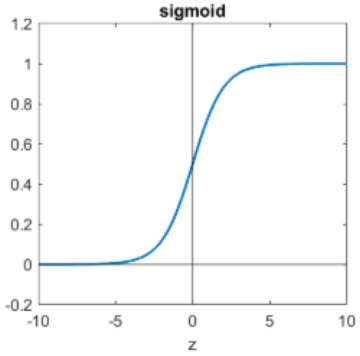
[10] Perceptron with sigmoid

- E.g., Perceptron with a logistic sigmoid
- Function: surface bent into a sigmoidal shape along the direction of the weight vector
- Changing the weights can rotate the direction of the sigmoidal surface, and stretch or shift it
- But, the fundamental sigmoidal shape will always remain

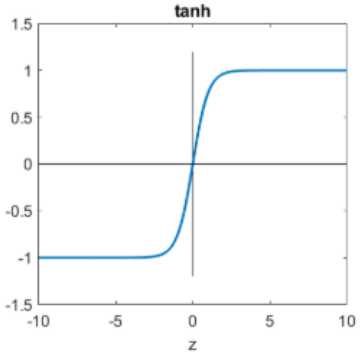
INTRODUCING NON-LINEARITY

Input images are highly non linear

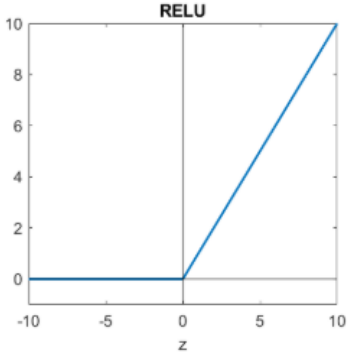
- **The activation function is applied after every layer operation**
- Each activation function (or non-linearity)
  - Takes a single number and performs a certain fixed mathematical operation on it



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

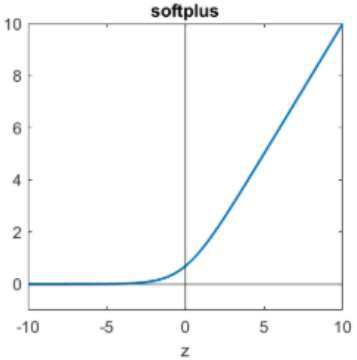


$$h(z) = \tanh z$$

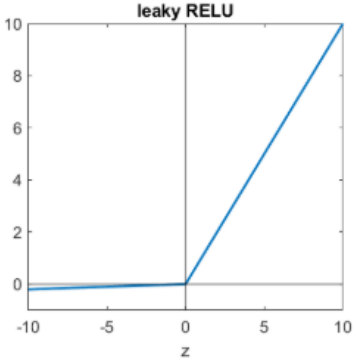


$$h(z) = \max(z, 0)$$

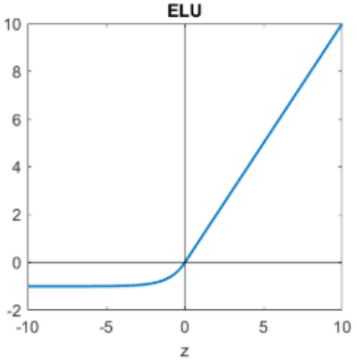
[11] Understanding the Neural Network



$$h(z) = \log(1 + e^{z})$$



$$h(z) = \max(z, z\alpha)$$
$$0 < \alpha < 1$$

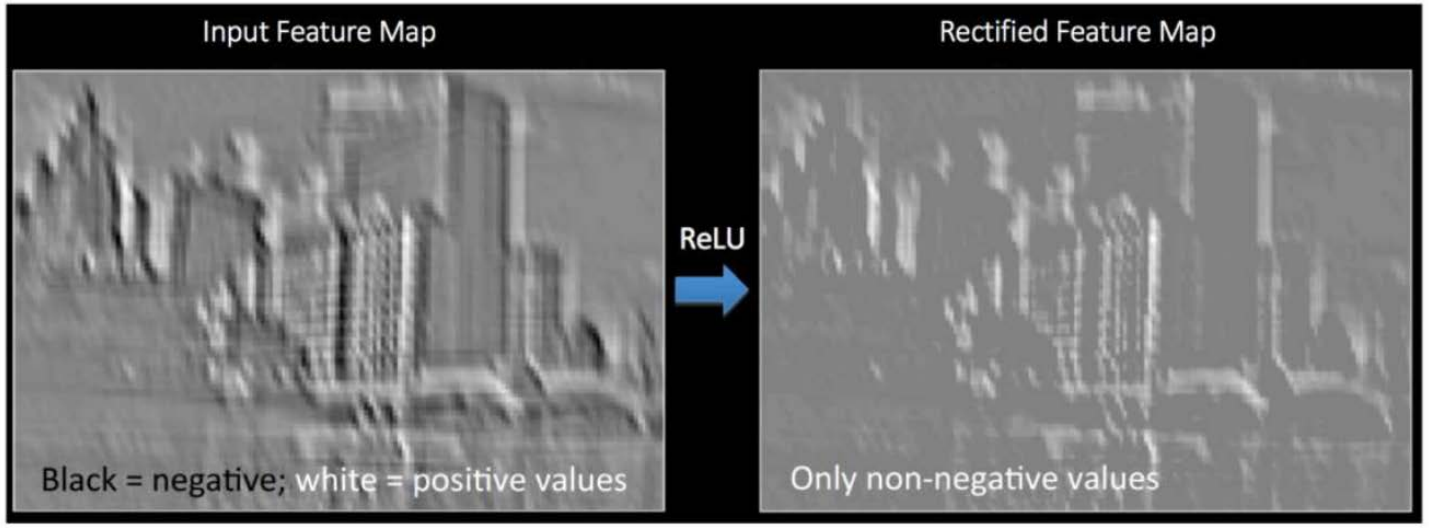


$$h(z) = \begin{cases} z, z > 0 \\ \alpha(e^{z} - 1) z \leq 0 \end{cases}$$

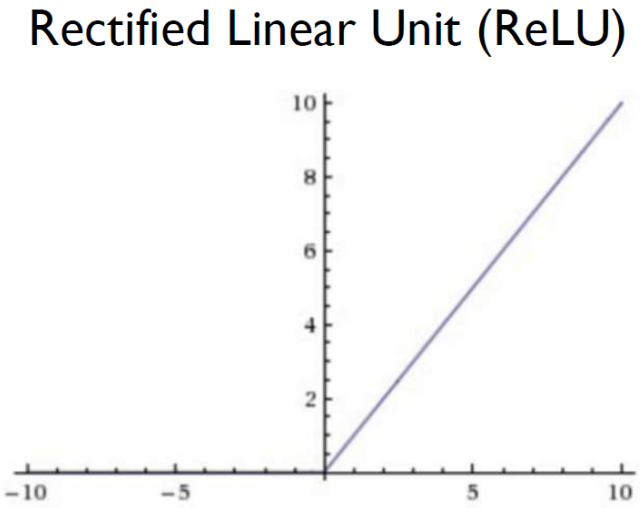
RECTIFIED LINEAR UNIT

Has become very popular in the last few years

- Compared to the sigmoid/tanh functions
  - (+) It was found to greatly accelerate the convergence of **Stochastic Gradient Descent (SGD)**
  - (+) Simply thresholding a matrix of activations at zero (no expensive operations, e.g., exponentials)



[13] CS231n



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

[12] A. Krizhevsky et al.

MLP

Training

- As for all supervised classifiers, one of the most important issue with MLP classifiers is **how to train it**
- Training means finding an opportune **architecture** and related **weight and bias** values
- The **highly nonlinear** nature of MLPs makes it not trivial to find an analytical solution to the problem
  - Therefore, one has to resort to **numerical optimizers**
- Another problem is the **number of weights and biases** to optimize

$$MLP\ with \begin{cases} 10\ input\ neurons \\ 20\ hidden\ neurons \\ 5\ output\ neurons \end{cases} \Rightarrow 325\ weights\ (+biases)$$

MLP

Backpropagation Algorithm

- What weights should be modified (and how much) to obtain correct classification?
  - I.e., Understand what connections are increasing or reducing to the error in the output
- Looking for an algorithm which modifies the different weights to **minimize the error rate**
- **Backpropagation:** iterative algorithm which has hugely contributed to neural network fame
- It is a **gradient-based search** method which allows finding a minimum of the **sum of squared error criterion**

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}(y_i - d_i)^2$$

TOTAL NUMBER OF TRAINING SAMPLES

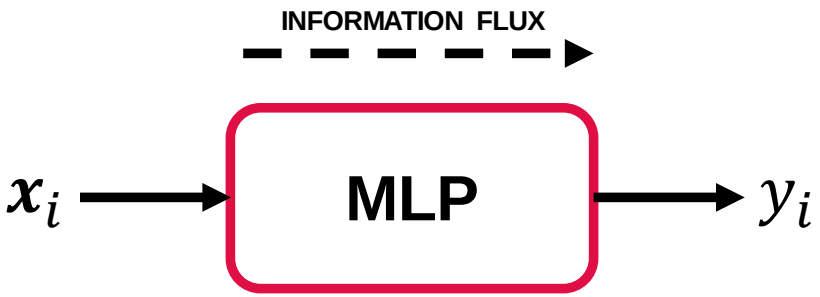
DESIRED OUTPUT (TARGET) VALUE FOR THE i-th SAMPLE

OUTPUT VALUE OBTAINED BY THE MLP FOR THE i-th SAMPLE

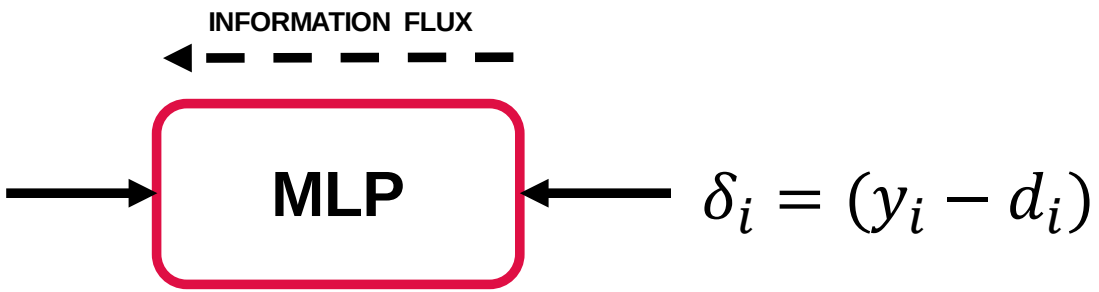
BACKPROPAGATION ALGORITHM

Three Phases

- Forward propagation phase



- Backward propagation phase



- Weight updating phase

LOSS OPTIMIZATION

- How to use the **loss** to iteratively **update the weights** over time given the **training data**?
- **Objective**: find the weights that minimize the **empirical loss**

$$\boldsymbol{W}^* = \arg\min \sum_{i=1}^{N} \mathcal{L}(f(x_i; \boldsymbol{W}), d_i)$$

$$\boldsymbol{W}^* = \arg\min_{\boldsymbol{W}} \mathcal{L}(\mathbf{W})$$

$\mathbf{W} = \{\mathbf{W}^{(0)}, \mathbf{W}^{(1)}, ...\}$ Set of all the weights in the network (first layer, second layer, etc..)

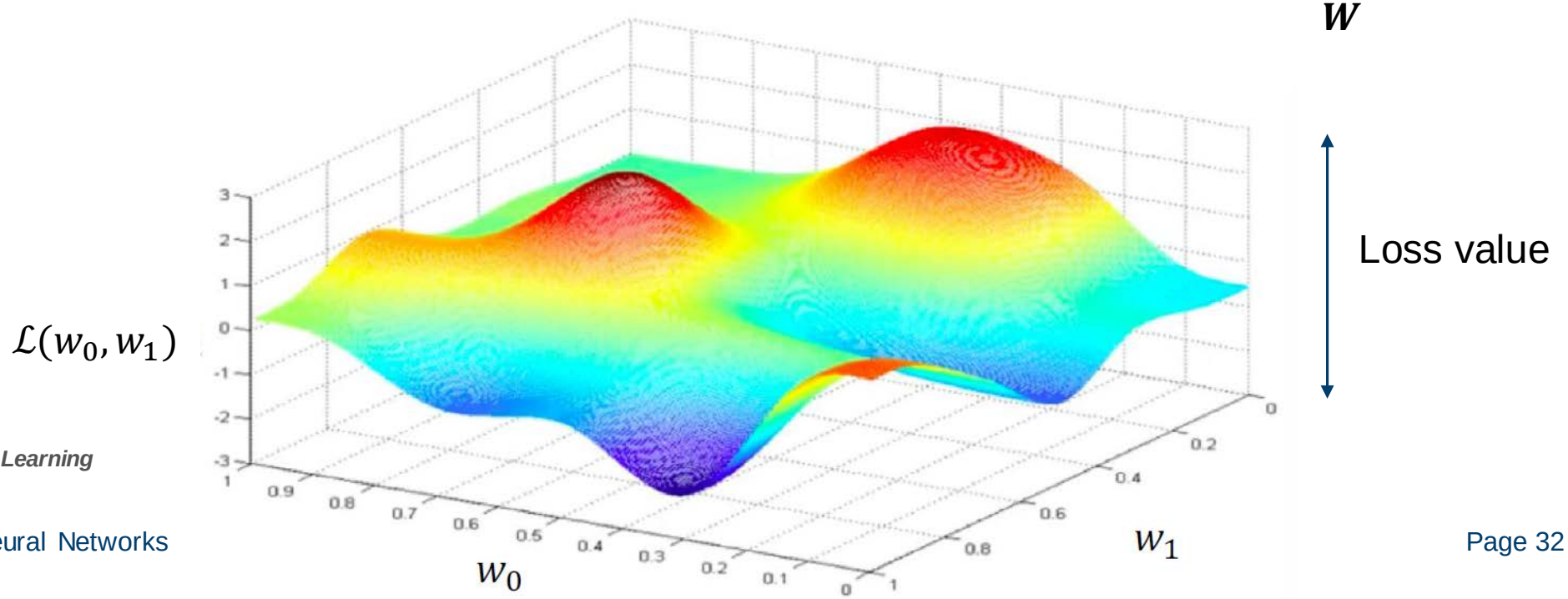
Compute this optimization problem over all these weights

LOSS OPTIMIZATION

illustration

- Loss function: takes as input the weights and gives the loss
  - It is a function of the network weights
- **Find the lowest point** in this landscape that correspond to the minimum loss
  - I.e., Find the correspondent weights

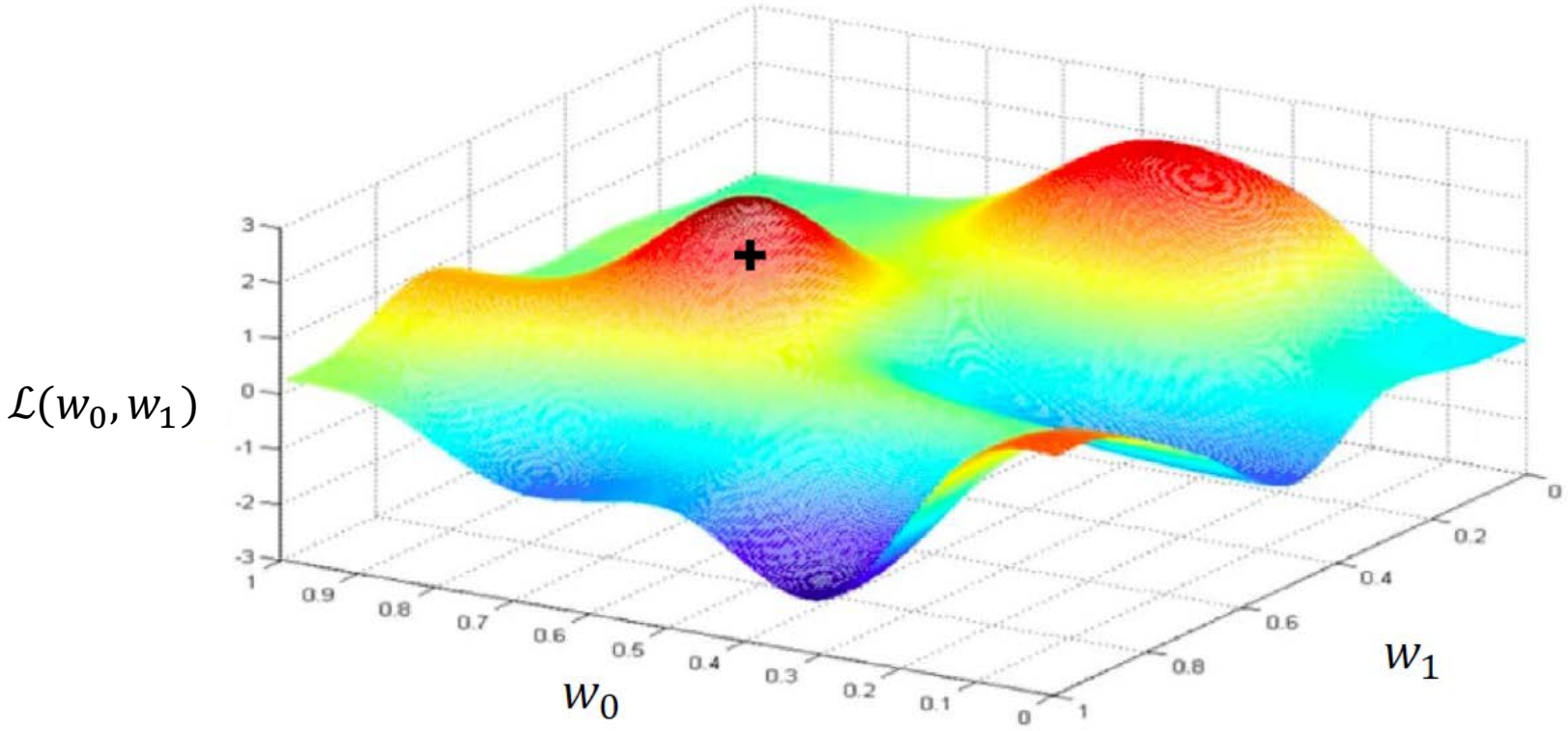
$$\boldsymbol{W}^* = \arg\min_{\boldsymbol{W}} \mathcal{L}(\mathbf{W})$$



LOSS OPTIMIZATION

illustration

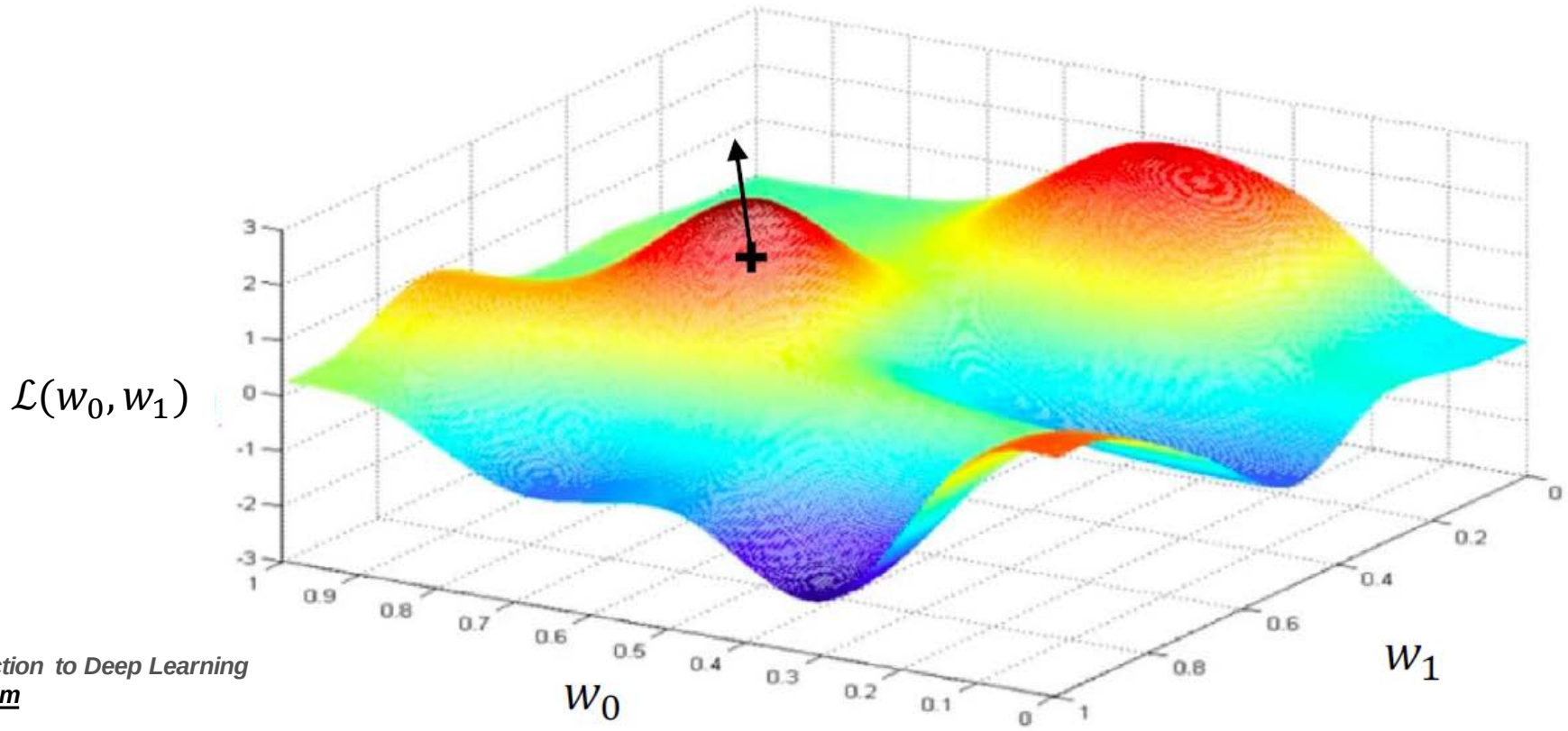
- Randomly pick an initial (w_0, w_1)



LOSS OPTIMIZATION

illustration

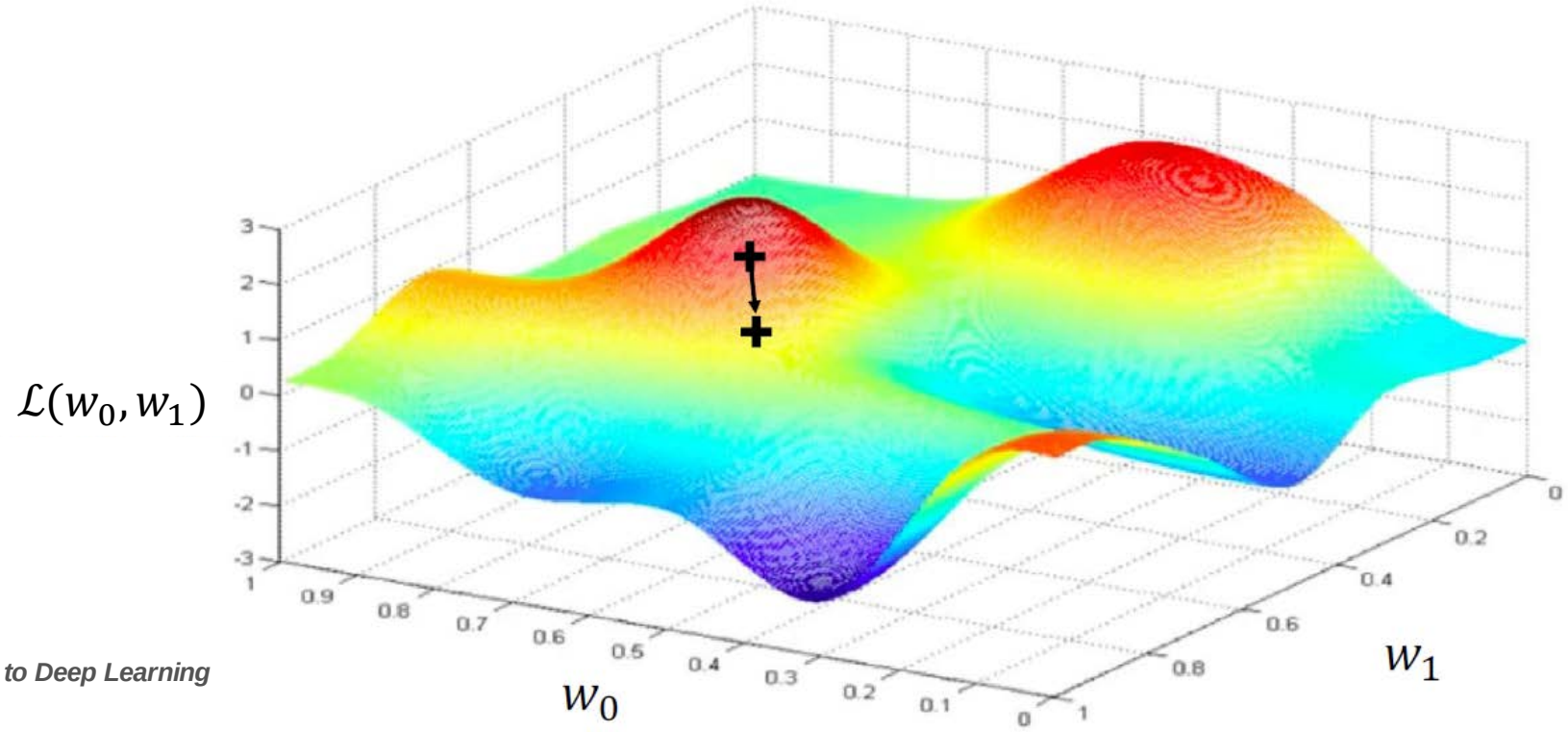
- Compute **gradient** at this local point $\frac{\partial \mathcal{L}(\boldsymbol{W})}{\partial \boldsymbol{W}}$
  - In this landscape the gradient tells us **the direction** of the maximum (steepest) **ascent**



LOSS OPTIMIZATION

illustration

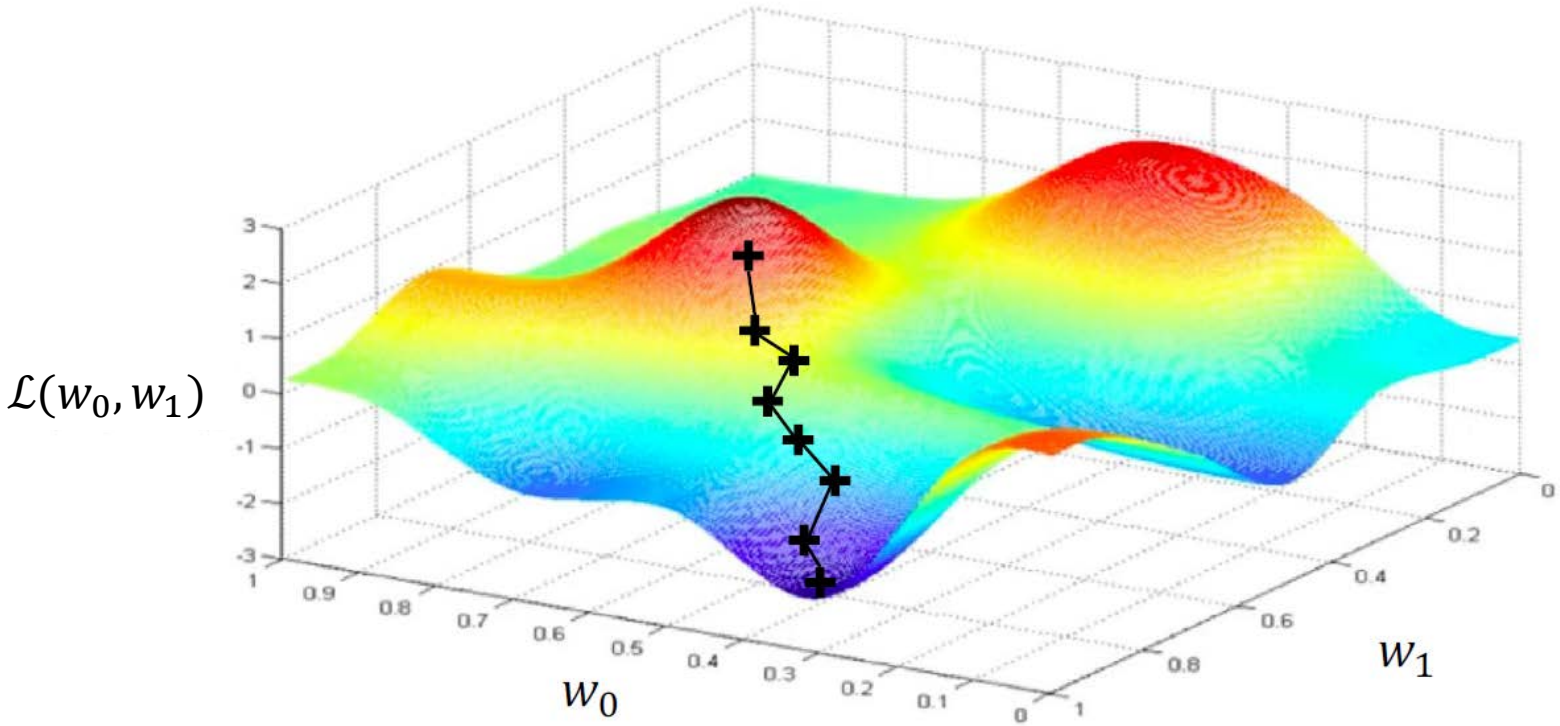
- Reverse the gradient and take a small step in opposite direction



GRADIENT DESCENT

illustration

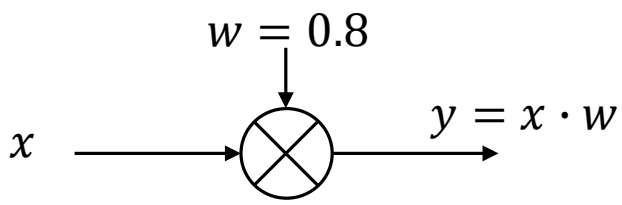
- Repeat until convergence
  - The gradient is computed over and over



BACKPROPAGATION ALGORITHM

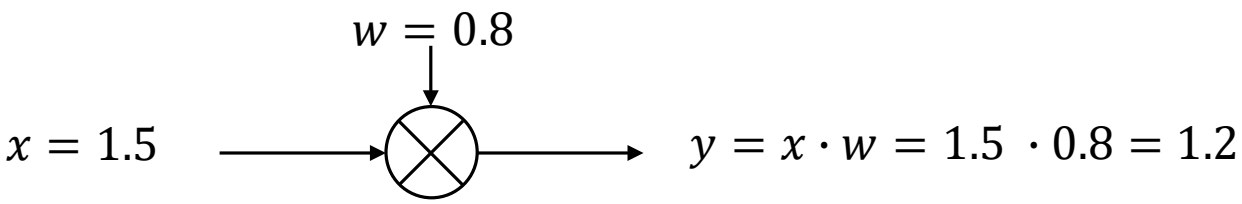
Example

- **Weight(s)** are randomly initialized



- **Training set**: a single data input/output pair

Input (x)	Output (y)
1.5	0.5



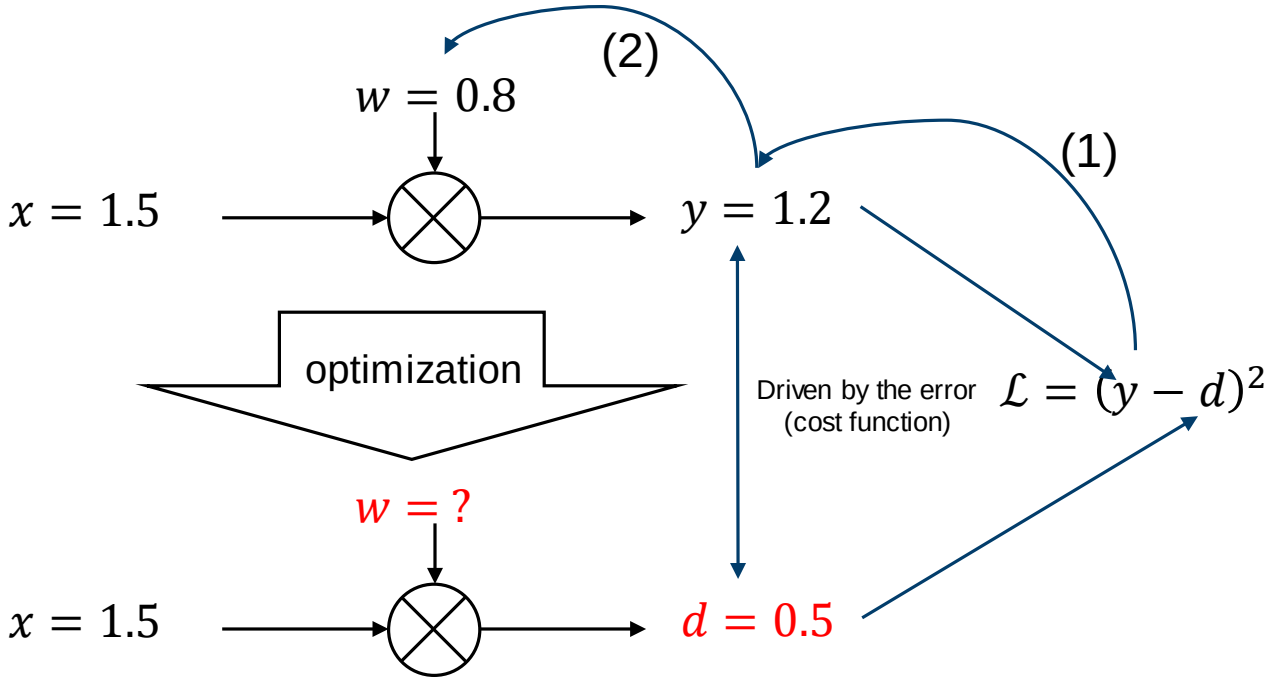
Some work has to be done to obtain the desired output

[13] Neural Network Backpropagation

BACKPROPAGATION ALGORITHM

Example

- **Objective**: change the weight w_1 so that the output is 0.5
- (1) How should y change for error to decrease?
  - We cannot change the output activation!
- Go one step forward and ask: (2) How should w change for y to change to decrease the error?

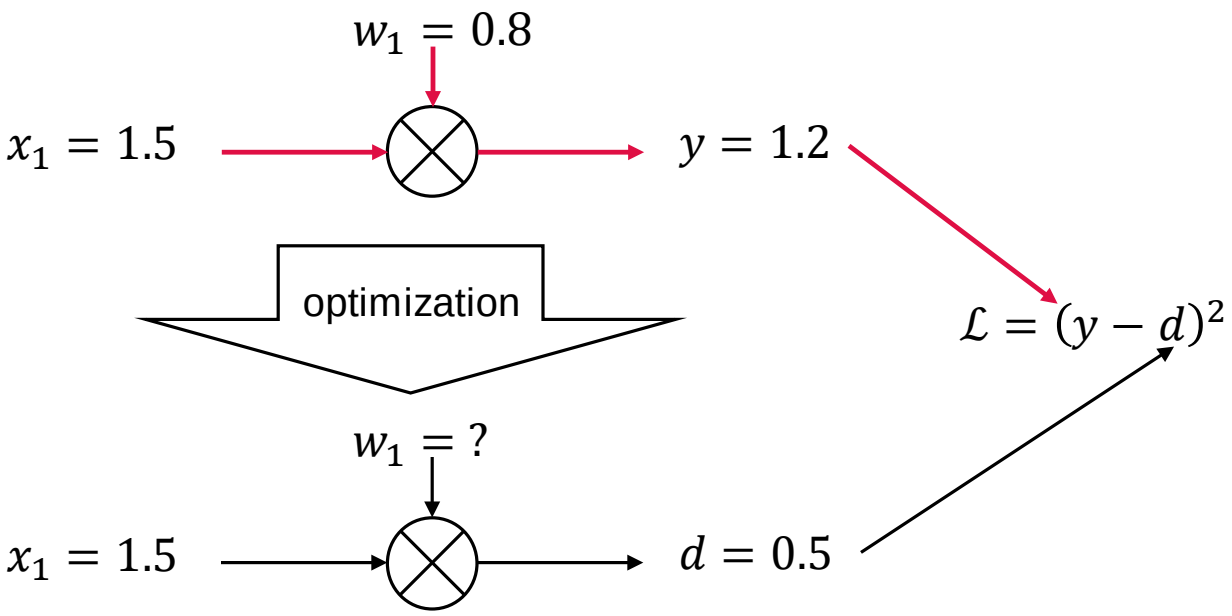


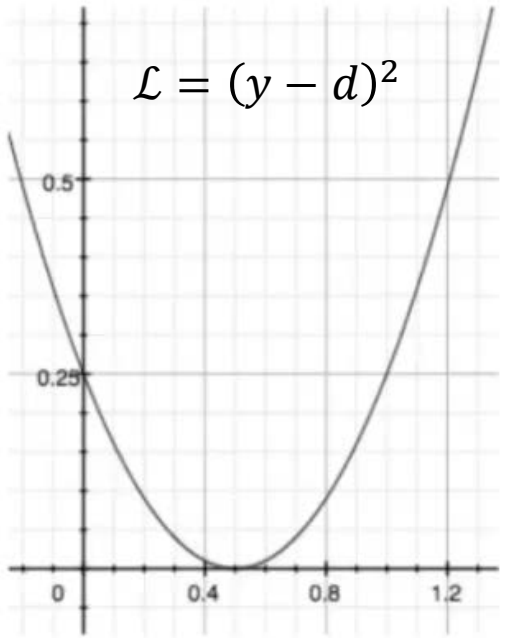
[11] Neural Network Backpropagation

BACKPROPAGATION ALGORITHM

Example

- How chages in y change the error? How to minimize the cost?
- **Gradient descent**: descend along the cost function in **direction** of the **negative gradient** to get to the minimum
  - Need to compute the gradient (in this simple case the derivative)





[11] Neural Network Backpropagation

CHAIN RULE OF DIFFERENTIATION

Reminder

- The rate of change of a function of a function is the multiple of the derivatives of those functions

$$\frac{\partial}{\partial x} f\big(g(x)\big) = g'(x) f(g)$$

- Useful rule for neural networks with several layers
  - The functions are composite

$$\frac{\partial}{\partial x} f(g(h(i(j(k(x)))))) = \frac{\partial k}{\partial x}\frac{\partial j}{\partial k}\frac{\partial i}{\partial j}\frac{\partial h}{\partial i}\frac{\partial g}{\partial h}\frac{\partial f}{\partial g}$$

CHAIN

BACKPROPAGATION ALGORITHM

Example

- Goal: minimize $\mathcal{L}$ with respect to w
  - Compute the rate of change of function $\mathcal{L}$ with respect to w
  - Then change w proportionally to that

Input (x)	Output (y)
1.5	0.5

- Cost function (it is function of y) — Rate of change with respect to y
  - $\mathcal{L}(\mathrm{y}) = (\mathrm{y} - \mathrm{d})^2 \quad \Longrightarrow \quad \frac{\partial \mathcal{L}}{\partial y} = 2y - 1 = 2(x \cdot w) - 1 = 2(1.5 \cdot w) - 1$
- The cost function is in turn a function of w — Rate of change of the output y with respect to w
  - $y(w) = x \cdot w \quad \Longrightarrow \quad \frac{\partial y}{\partial w} = x = 1.5$
- Which means the chain rule can be used
  - $\frac{\partial \mathcal{L}}{\partial w} = \frac{\partial y}{\partial w}\frac{\partial \mathcal{L}}{\partial y} \quad \Longrightarrow \quad \frac{\partial y}{\partial w}\frac{\partial \mathcal{L}}{\partial y} = 1.5 \cdot (2 \cdot (1.5 \cdot w) - 1) = 4.5 \cdot w - 1.5$

BACKPROPAGATION ALGORITHM

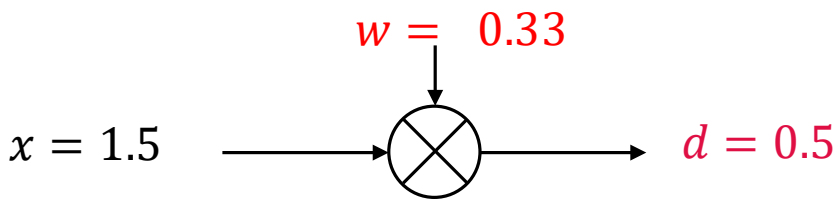
1. Initialize weights randomly $\sim\mathcal{N}(0,\sigma^2)$
2. Loop until convergence
3. Compute gradient $\frac{\partial \mathcal{L}(\boldsymbol{W})}{\partial \boldsymbol{W}}$ (it explains how the loss changes with respect to each of the weights)
4. Update weights $\boldsymbol{W} \coloneqq \boldsymbol{W} - \eta \frac{\partial \mathcal{L}(\boldsymbol{W})}{\partial W}$ (in the opposite direction of the gradient)
5. Return weights

Most computational part when there is a high number of weights

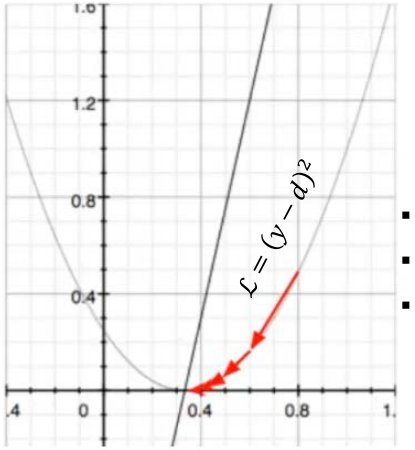
Use a small amount (the step) i.e., the **learning rate**
I.e., <u>How much do you trust the computed gradient</u>

$$w_1 = w_0 - \eta \frac{\partial \mathcal{L}}{\partial w} = w_0 - 0.1 \cdot (4.5 \cdot w_0 - 1.5)$$

w_0	w_1
0.8	0.59
0.59	0.4745
0.4745	0.410975
0.410975	0.37603625
0.37603625	0.3568199375
……	
→ 0.333333	



$$\frac{\partial \mathcal{L}}{\partial w} = 4.5 \cdot w - 1.5$$

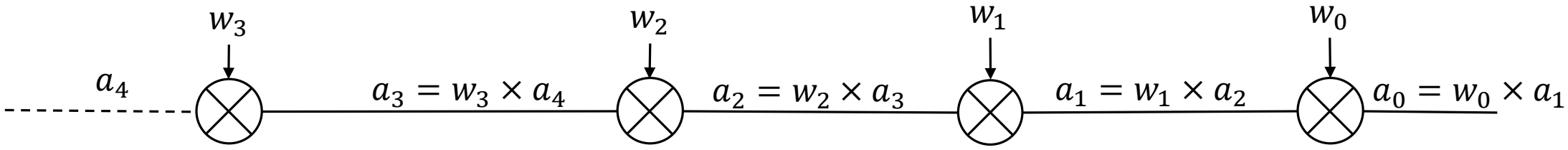


- The slope of the $\mathcal{L}$ determines the adjustment direction
- Adjust weights proportional to the negative of the gradient
- The adjustment is ''backpropagated'' through all layers

[11] Neural Network Backpropagation

BACKPROPAGATION ALGORITHM

Generalized to Several Layers



$$-\eta \frac{\partial a_3}{\partial w_3} \frac{\partial a_2}{\partial a_3} \frac{\partial a_1}{\partial a_2} \frac{\partial a_0}{\partial a_1} \frac{\partial \mathcal{L}}{\partial a_0} \qquad -\eta \frac{\partial a_2}{\partial w_2} \frac{\partial a_1}{\partial a_2} \frac{\partial a_0}{\partial a_1} \frac{\partial \mathcal{L}}{\partial a_0} \qquad -\eta \frac{\partial a_1}{\partial w_1} \frac{\partial a_0}{\partial a_1} \frac{\partial \mathcal{L}}{\partial a_0} \qquad -\eta \frac{\partial a_0}{\partial w_0} \frac{\partial \mathcal{L}}{\partial a_0}$$

- For example, we adjust w_3 as follows

$$w_3' = w_3 - \eta \cdot a_4 \cdot w_2 \cdot \mathrm{w}_1 \cdot w_0 \cdot 2(a_0 - d)$$

$$-\eta \frac{\partial a_3}{\partial w_3} \frac{\partial a_2}{\partial a_3} \frac{\partial a_1}{\partial a_2} \frac{\partial a_0}{\partial a_1} \frac{\partial \mathcal{L}}{\partial a_0}$$

BACKPROPAGATION ALGORITHM

Convergence

- The **convergence velocity** depends on the:
  - Complexity of the considered classification problem
  - Complexity of the approximating function
  - Value of the **learning rate**

- Three different weight updating (training) strategies could be adopted:
  - Training by pattern
  - Batch training
  - Stochastic training

LOSS FUNCTIONS

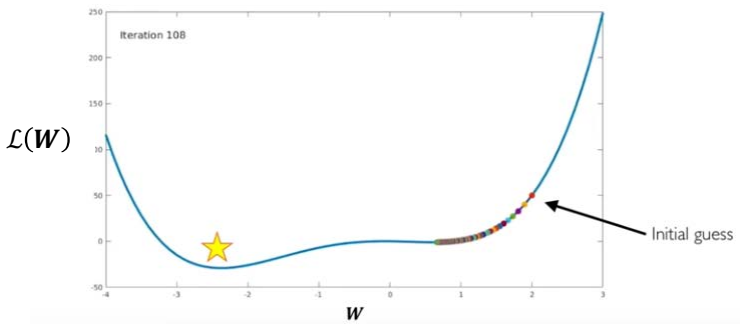
Optimization through gradient descent

- Take the weights and subtract, move via the negative gradient
- The **learning rate** determines the **adjustment magnitude**

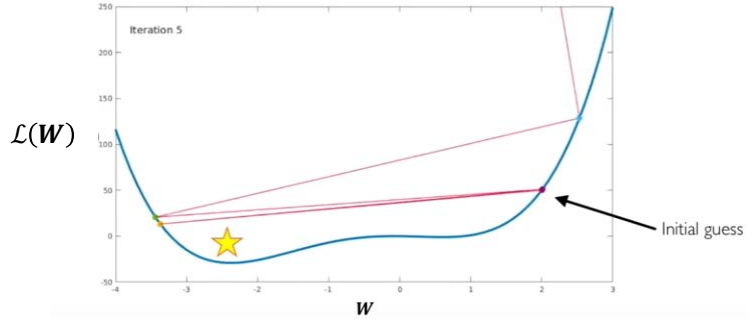
$$\boldsymbol{W} := \boldsymbol{W} - \eta \frac{\partial \mathcal{L}(\boldsymbol{W})}{\partial \boldsymbol{W}}$$

How to set the **learning rate**?

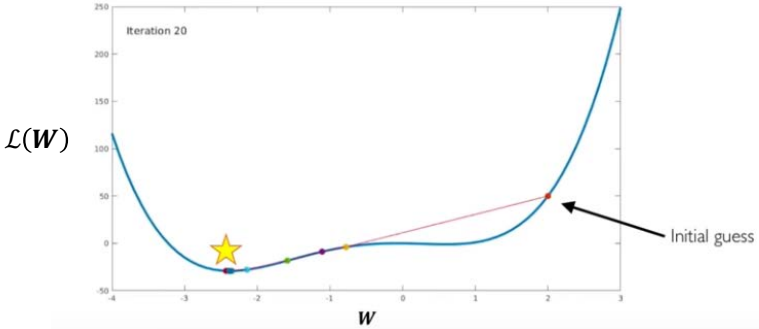
- How large is the step we take at each iteration
  - In practice it is very difficult to set this parameter
  - It is very important in order to avoid local minima



Small learning rate converges slowly and gets stuck in false local minima



Large learning rates overshoot, become unstable and diverge



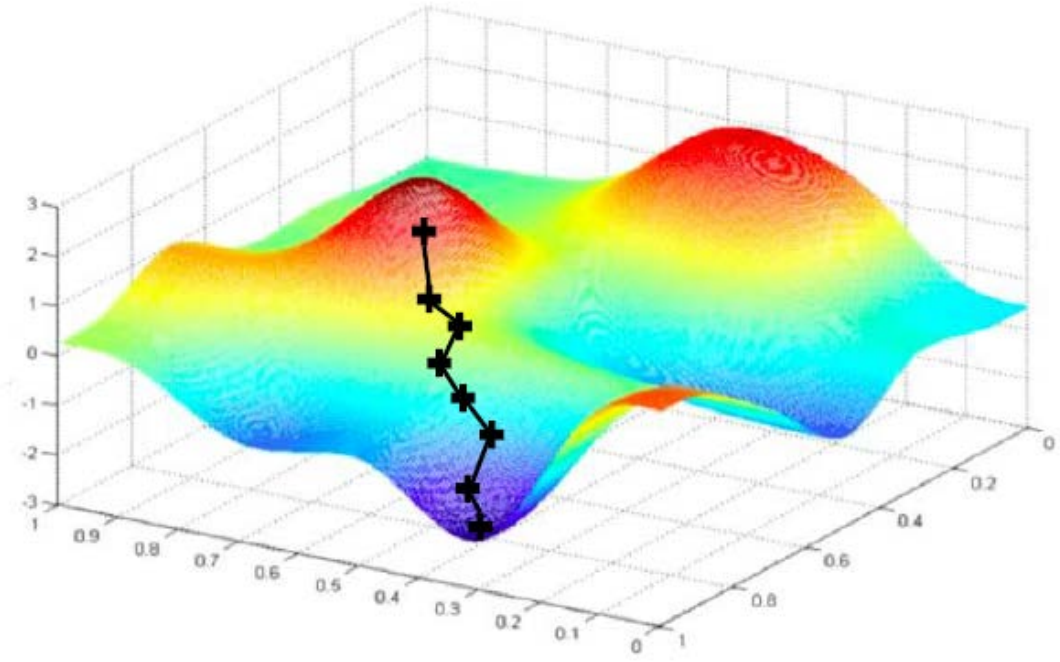
Stable learning rates converge smoothly and avoid local minima

BATCH GRADIENT DESCENT (BGD)

Gradient is very computational to compute

- When it is computed for all the N samples within a given training dataset
- Think about <u>big remote sensing datasets</u>

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

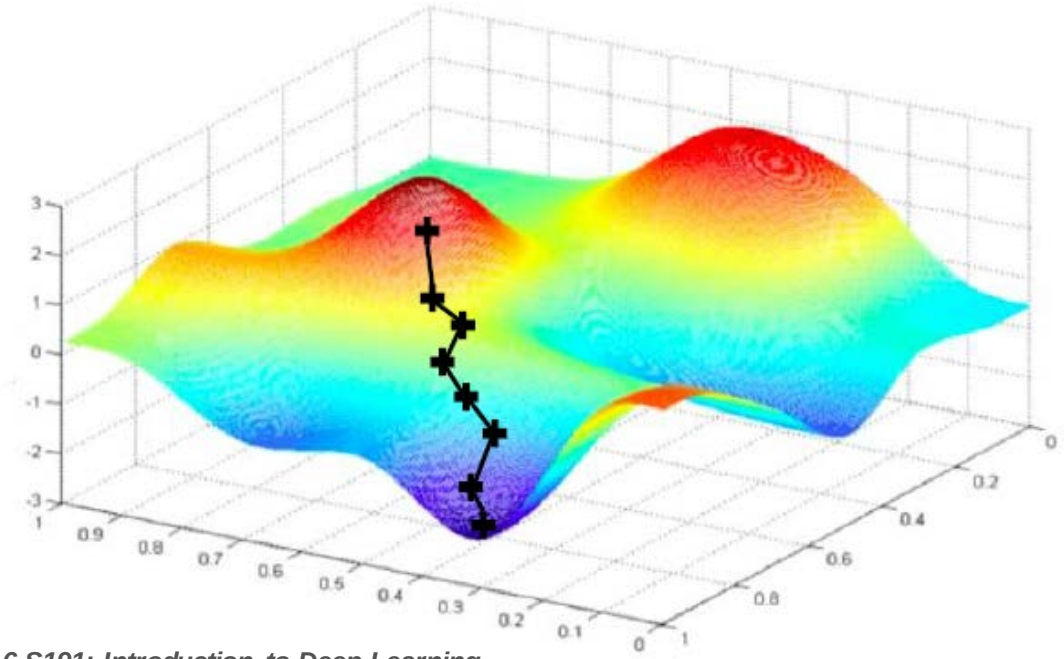


STOCHASTIC GRADIENT DESCENT (SGD)

- Strategy
  - Pick a single point
  - Compute the gradient with respect to that single point and use it to update the weights
- We might go in a direction/step that is not representative for the entire dataset

1. Initialize weights randomly $\sim\mathcal{N}(0,\sigma^2)$
2. Loop until convergence
3. **Pick single data point $\boldsymbol{i}$**
4. Compute gradient $\frac{\partial \mathcal{L}_i(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Update weights $\boldsymbol{W} := \boldsymbol{W} - \eta \frac{\partial \mathcal{L}(\boldsymbol{W})}{\partial \boldsymbol{W}}$
6. Return weights

Easy to compute but very noisy (stochastic)!

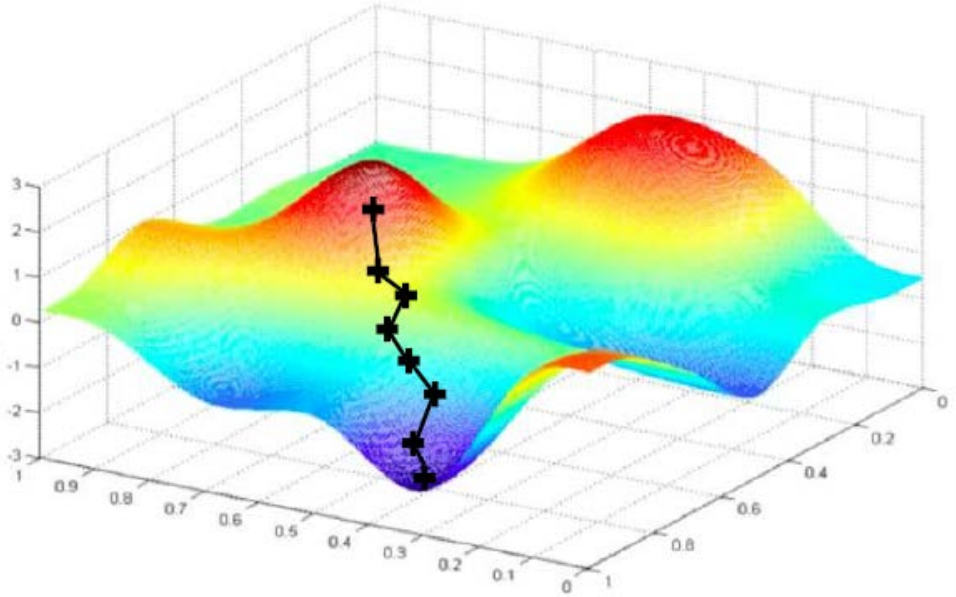


MINI-BATCH GRADIENT DESCENT

Batching the data into mini-batches

- Tradeoff between **BGD** and **SGD**
- Gives an estimate of the true gradient by averaging the gradient from each of the B points
- Minibatch sampling: implemented by shuffling the dataset S, and processing that permutation by obtaining contiguous segments of size B from it

1. Initialize weights randomly $\sim\mathcal{N}(0,\sigma^2)$
2. Loop until convergence
3. **Pick batch of B data points**
4. Compute gradient $\frac{\partial \mathcal{L}_i(\boldsymbol{W})}{\partial \boldsymbol{W}} = \frac{1}{B}\sum_{k=1}^{B}\frac{\partial \mathcal{L}_i(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Update weights $\boldsymbol{W} := \boldsymbol{W} - \eta\,\frac{\partial \mathcal{L}(\boldsymbol{W})}{\partial \boldsymbol{W}}$
6. Return weights



Fast to compute and a much better estimate of the true gradient!

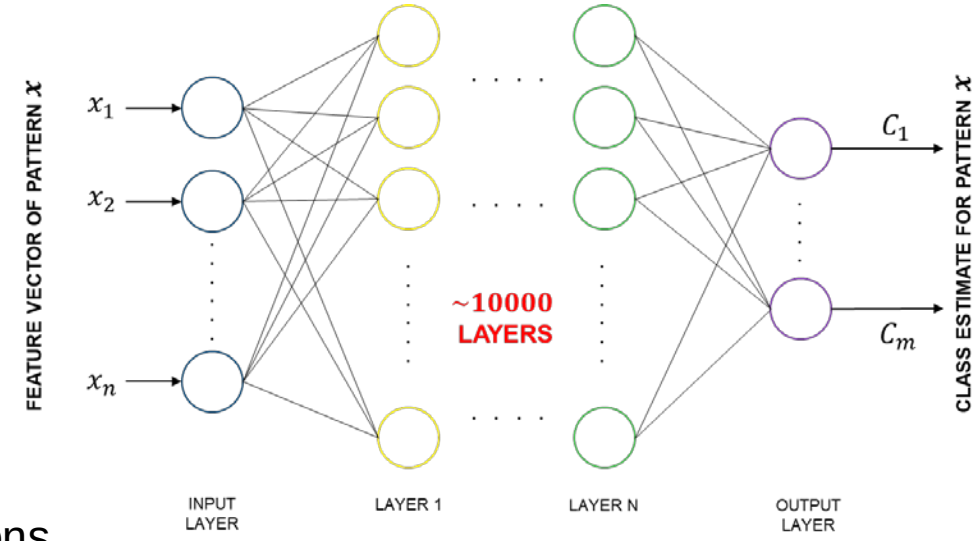
MINI-BATCHES WHILE TRAINING

- To summarize
  - Batch Gradient Descent: Batch Size = Size of Training Set
  - Stochastic Gradient Descent: Batch Size = 1
  - Mini-Batch Gradient Descent: 1 < Batch Size < Size of Training Set

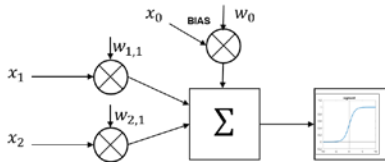
- **Mini-Batch Gradient Descent:**
  - **More accurate estimation of gradient**
    - Smoother convergence
    - Allows for larger learning rates (i.e., trust more the gradient , training faster)
  - **Mini-batches lead to fast training!**
    - Can parallelize computation + achieve significant speed increases on GPU's
    - Send batches across the GPUs, compute their gradient simultaneously and aggregate them back

DEEP LEARNING

- **Deep Learning**: using a generic, flexible model family
  - **"Neural" Networks** with multiple layers
  - Based on stacked, repetitive operations
  - Executed by simple, generic units
  - With parameters ("weights") adapted from incoming data



- **Deep Neural Networks**
  - Most models can be cooked down to stacked repetitive operations
  - They are executed by simple units
  - **Linear summation + non-linear transfer function**



- The units are very simple
- The complexity arise when combining more of them
- The are many weights (connections) that are adjustable from the data (they are not fixed)

PRACTICALS

Add Two Hidden Layers for Artificial Neural Network (ANN)

- All parameter value remain the same as before
- We add N_HIDDEN as parameter in order to set 128 neurons in one hidden layer – this number is a hyperparameter that is not directly defined and needs to be find with parameter search

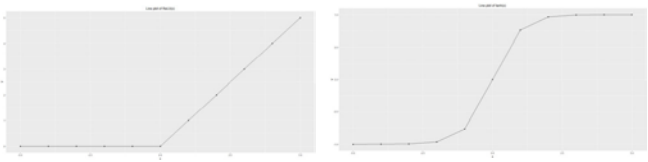
```
# parameter setup
NB_EPOCH = 20
BATCH_SIZE = 128
NB_CLASSES = 10 # number of outputs = number of digits
OPTIMIZER = SGD() # optimization technique
VERBOSE = 1
N_HIDDEN = 128 # number of neurons in one hidden layer
```

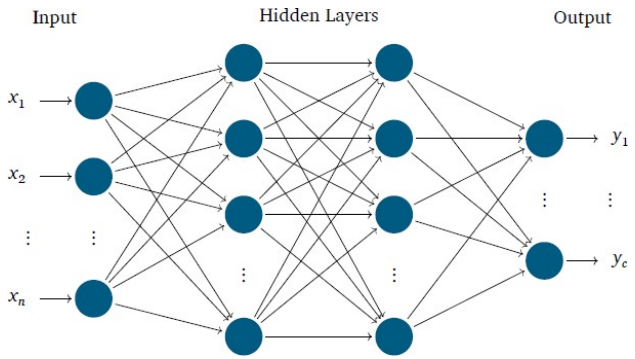
```
# model Keras sequential
model = Sequential()
```

[redacted]

```
# add activation function layer to get class probabilities
model.add(Activation('softmax'))
```

(activation functions ReLU & Tanh)





[redacted]

- The non-linear Activation function 'relu' represents a so-called Rectified Linear Unit (ReLU) that only recently became very popular because it generates good experimental results in ANNs and more recent deep learning models – it just returns 0 for negative values and grows linearly for only positive values
- A hidden layer in an ANN can be represented by a fully connected Dense layer in Keras by just specifying the number of hidden neurons in the hidden layer

PRACTICALS

Summarize history for accuracy and loss

- import matplotlib.pyplot as plt
- plt.plot(history.history['acc'])
- plt.plot(history.history['val_acc'])
- plt.title('model accuracy')
- plt.ylabel('accuracy')
- plt.xlabel('epoch')
- plt.legend(['train', 'val'], loc='upper left')
- plt.show()

- import matplotlib.pyplot as plt
- plt.plot(history.history['loss'])
- plt.plot(history.history['val_loss'])
- plt.title('model loss')
- plt.ylabel('loss')
- plt.xlabel('epoch')
- plt.legend(['train', 'val'], loc='upper left')
- plt.show()

PERFORMANCE MEASURES

- **Overall Accuracy (OA):** percentage of correctly classified pixels (K is the number of classes)

$$OA = \frac{\sum_i^K C_{ii}}{\sum_{ij}^K C_{ij}}$$

- **Class Accuracy (CA):** percentage of correctly classified pixels for a given class

$$CA_i = \frac{C_{ii}}{\sum_j^K C_{ij}}$$

- **Average Accuracy (AA):** mean of class-specific accuracies for all the classes:

$$AA = \frac{\sum_i^K CA_i}{K}$$

Percentage	Classification data				
Reference data	C_1	C_2	C_3	Row total	Class-specific accuracy
C_1	C_{11}	C_{12}	C_{13}	$\sum_i^K C_{1i}$	$\frac{C_{11}}{\sum_i^K C_{1i}}$
C_2	C_{21}	C_{22}	C_{23}	$\sum_i^K C_{2i}$	$\frac{C_{22}}{\sum_i^K C_{2i}}$
C_3	C_{31}	C_{22}	C_{33}	$\sum_i^K C_{3i}$	$\frac{C_{33}}{\sum_i^K C_{3i}}$
Column total	$\sum_i^K C_{i1}$	$\sum_i^K C_{i2}$	$\sum_i^K C_{i3}$	N	
User's accuracy	$\frac{C_{11}}{\sum_i^K C_{i1}}$	$\frac{C_{11}}{\sum_i^K C_{i2}}$	$\frac{C_{33}}{\sum_i^K C_{i3}}$		

C_i : the class i

C_{ij}: number of pixels classified to the class j and referenced as the class i

WHICH MODEL PERFORM BETTER?

MODEL 1
(parameters)

PREDICTIONS →

ACTUAL VALUES ↓

	1	2	3	4
1	10	0	0	0
2	0	5	3	2
3	0	1	8	1
4	0	1	0	9

CONFUSION MATRIX

MODEL 2
(parameters)

PREDICTIONS →

ACTUAL VALUES ↓

	1	2	3	4
1	8	2	0	0
2	1	7	0	2
3	0	0	9	1
4	2	3	0	5

CONFUSION MATRIX

- Possible outcomes from the classification
  - TN True Negatives – correct prediction
  - TP True Positives – correct prediction
  - FN False Negatives – incorrect prediction
  - FP False Positives – incorrect prediction

[15] Performance measure

TRUE POSITIVE

- Correctly identified prediction for each class

PREDICTIONS →

ACTUAL VALUES ↓

	1	2	3	4
1	10	0	0	0
2	0	5	3	2
3	0	1	8	1
4	0	1	0	9

TRUE NEGATIVE

- Correctly rejected prediction for a certain class

PREDICTIONS →

ACTUAL VALUES ↓

	1	2	3	4
1	9	1	0	0
2	1	15	3	1
3	5	0	24	1
4	0	4	1	15

True Negative for class 1

PREDICTIONS →

ACTUAL VALUES ↓

	1	2	3	4
1	9	1	0	0
2	1	15	3	1
3	5	0	24	1
4	0	4	1	15

True Negative for class 4

FALSE POSITIVE

- Incorrectly identified predictions for a certain class

PREDICTIONS →

ACTUAL VALUES ↓

	1	2	3	4
1	9	1	0	0
2	1	15	3	1
3	5	0	24	1
4	0	4	1	15

False positive for class 1

PREDICTIONS →

ACTUAL VALUES ↓

	1	2	3	4
1	9	1	0	0
2	1	15	3	1
3	5	0	24	1
4	0	4	1	15

False positive for class 2

FALSE NEGATIVE

- Incorrectly rejected predictions for a certain class

PREDICTIONS →

ACTUAL VALUES ↓

	1	2	3	4
1	9	1	0	0
2	1	15	3	1
3	5	0	24	1
4	0	4	1	15

False negative for class 1

PREDICTIONS →

ACTUAL VALUES ↓

	1	2	3	4
1	9	1	0	0
2	1	15	3	1
3	5	0	24	1
4	0	4	1	15

False negative for class 4

OVERALL ACCURACY

- Accuracy is calculated as a total number of correct predictions divided by the total number of samples

PREDICTIONS →

ACTUAL VALUES ↓

	1	2	3	4
1	9	1	0	0
2	1	15	3	1
3	5	0	24	1
4	0	4	1	15

- Accuracy= (9+15+24+15) / 80 = 0.78

ACCURACY WORKS WELL ON BALANCED DATA

MODEL 1
(parameters)

PREDICTIONS →

ACTUAL VALUES ↓

	1	2	3	4
1	10	0	0	0
2	0	5	3	2
3	0	1	8	1
4	0	1	0	9

Accuracy=(10+5+8+90)/40=0.8

MODEL 2
(parameters)

PREDICTIONS →

ACTUAL VALUES ↓

	1	2	3	4
1	8	2	0	0
2	1	7	0	2
3	0	0	9	1
4	2	3	0	5

Accuracy=(8+7+9+5)/40=0.725

ACCURACY ON IMBALANCED DATA MISLEADS PERFORMANCE

MODEL 1
(parameters)

ACTUAL VALUES

	1	2	3	4
1	100	80	10	10
2	0	9	0	1
3	0	1	8	1
4	0	1	0	9

200 test samples for class 1
10 test samples for class 2,3,4
Accuracy=(100+9+8+9)/230=**0.547**

This model is very accurate for class 1 but also for the other classes

PREDICTIONS

MODEL 2
(parameters)

ACTUAL VALUES

	1	2	3	4
1	198	2	0	0
2	7	1	0	2
3	0	8	1	1
4	2	3	4	1

200 test samples for class 1
10 test samples for class 2,3,4
Accuracy=(198+1+1+1)/230=**0.87**

This model is very accurate for class 1 but not for the other classes

F1 SCORE IS A GOOD METRIC WHEN DATA IS IMBALANCED

F1 Score considers both sides

GIVEN A CLASS, WILL THE CLASSIFIER DETECT IT? (**RECALL**)

	1	2	3	4
1	100	80	10	10
2	0	9	0	1
3	0	1	8	1
4	0	1	0	9

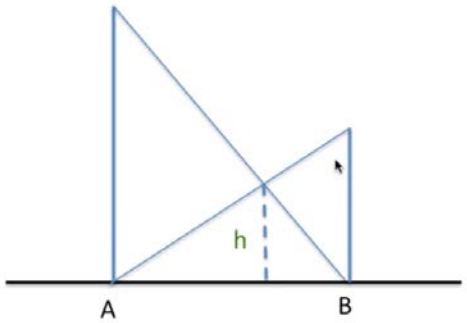
PREDICTIONS

GIVEN A CLASS PREDICTION FROM THE CLASSIFIER,
HOW LIKELY IS IT TO BE CORRECT? (**PRECISION**)

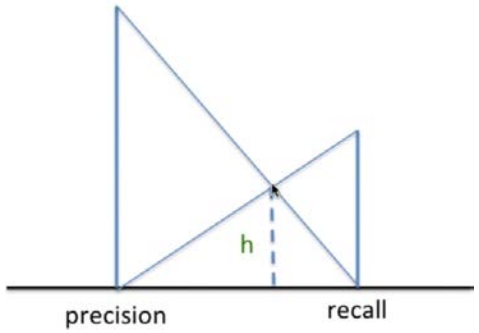
- F1 Score is **harmonic mean** of recall and precision

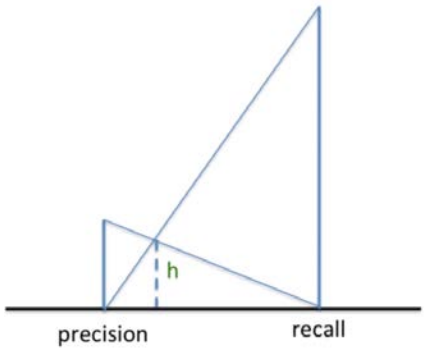
HARMONIC MEAN

- Definition:
  - h is half the harmonic mean



- Harmonic mean punishes extreme value more
  - E.g., High recall values are punished (imbalanced data)





- F1 Score = $2 \times \frac{Precision \times Recall}{Precision + Recall}$

PRECISION OF MODEL 1 (MACRO AVERAGE)

MODEL 1
(parameters)

ACTUAL VALUES

	1	2	3	4
1	100	80	10	10
2	0	9	0	1
3	0	1	8	1
4	0	1	0	9

TP=100 TP=9 TP=8 TP=9
FP=0 FP=82 FP=10 FP=12

Precision = TP / (TP+FP)

Precision(class=1)=1 Precision(class=2)=9/91
Precision(class=3)=8/18 Precision(class=4)=9/21

Average precision =
Precision(class=1)+ Precision(class=2)+ Precision(class=3)+ Precision(class=4)/4 = 0.492

Number of classes

RECALL OF MODEL 1 (MACRO AVERAGE)

PREDICTIONS →

MODEL 1 (parameters)

	1	2	3	4
1	100	80	10	10
2	0	9	0	1
3	0	1	8	1
4	0	1	0	9

TP = 100, FN=100 Recall(class=1)=100/200

TP = 9, FN=1 Recall(class=2)=9/10

TP = 8, FN=2 Recall(class=3)=8/10

TP = 9, FN=1 Recall(class=4)=9/10

Recall = TP / (TP+FN)

Average Recall =

Recall(class=1)+ Recall(class=2)+ Recall(class=3)+ Recall(class=4)/4 = 0.775

Number of classes

F1 SCORE OF MODEL 1

MODEL 1
(parameters)

PREDICTIONS →

ACTUAL VALUES ↓

	1	2	3	4
1	100	80	10	10
2	0	9	0	1
3	0	1	8	1
4	0	1	0	9

- F1 Score = $2 \times \frac{Precision \times Recall}{Precision + Recall} = 2 \times \frac{0.492 \times 0.775}{0.492 + 0.775} = 0.601$

F1 SCORE ON IMBALANCED DATA

MODEL 1
(parameters)

PREDICTIONS →

ACTUAL VALUES ↓

	1	2	3	4
1	100	80	10	10
2	0	9	0	1
3	0	1	8	1
4	0	1	0	9

Accuracy = 0.547

F1 Score = 0.601

MODEL 2
(parameters)

PREDICTIONS →

ACTUAL VALUES ↓

	1	2	3	4
1	198	2	0	0
2	7	1	0	2
3	0	8	1	1
4	2	3	4	1

Accuracy = 0.87

F1 Score = 0.342

Model 1 predicts well on multiple class classification on imbalanced data and F1 Score is the metric to quantify Its performance

HOMEWORK

- https://www.cs.toronto.edu/~kriz/cifar.html
- https://keras.io/datasets/
- **from** keras.datasets **import** cifar10
- (x_train, y_train), (x_test, y_test) = cifar10.load_data()

REFERENCES

- [1] Human brain
  Online: https://en.wikipedia.org/wiki/Human_brain
- [2] Facebook's artificial intelligence research team, FAIR, turns five. But what are its biggest accomplishments?
- Online: https://hub.packtpub.com/facebooks-artificial-intelligence-research-team-fair-turns-five-but-what-are-its-biggest-accomplishments/
- [3] Biological neuron model
  Online: https://en.wikipedia.org/wiki/Biological_neuron_model
- [4] L. Anderberg, H. Aldskogius and A. Holtz, '' Spinal cord injury – scientific challenges for the unknown future'', in Upsala Journal of Medical Sciences, vol. 112, Issue 3, doi: 10.3109/2000-1967-200, 2007.
- [5] MIT 6.S094: Deep Learning for Self-Driving Cars
  Online: https://selfdrivingcars.mit.edu/
- [6] Perceptron Learning Algorithm
  Online: https://towardsdatascience.com/perceptron-learning-algorithm-d5db0deab975
- [7] The XOR Problem in Neural Networks
  Online: https://medium.com/@jayeshbahire/the-xor-problem-in-neural-networks-50006411840b
- [8] Multilayer Perceptron
  Online: https://www.eecs.yorku.ca/course_archive/2012-13/F/4404-5327/lectures/10%20Multilayer%20Perceptrons.pdf
- [9] R. O. Duda, P. E. Hart and D. G. Stork. Pattern Classification. Second Edition, New York: John Wiley & Sons Inc, 2001.
- [10] Can a perceptron with sigmoid activation function perform nonlinear classification?
  Online: https://stats.stackexchange.com/questions/263768/can-a-perceptron-with-sigmoid-activation-function-perform-nonlinear-classificati

REFERENCES

- [11] Understanding the Neural Network
  Online: http://www.cs.cmu.edu/~bhiksha/courses/deeplearning/Fall.2019/www/hwnotes/HW1p1.html
- [12] He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In Proceedings of the IEEE International Conference on Computer Vision. https://doi.org/10.1109/ICCV.2015.123
- [13] Neural Network Backpropagation Basics For Dummies
  Online: https://www.youtube.com/watch?v=8d6jf7s6_Qs
- [14] CS231n Convolutional Neural Networks for Visual Recognition
  Online: http://cs231n.github.io/
- [15] Performance measure on multiclass classification [accuracy, f1 score, precision, recall]
  Online: https://www.youtube.com/watch?v=HBi-P5j0Kec