Machine Learning for Networking
ML4N

Luca Vassio
Gabriele Ciravegna
Zhihao Wang
Tailai Song

Supervised learning

- **Data** points characterized by features and labels

$$\mathcal{D} = \left\{ \left(\mathbf{x}^{(1)}, y^{(1)}\right), \ldots, \left(\mathbf{x}^{(m)}, y^{(m)}\right) \right\}.$$

- Predict the label y of a data point from its features **x**
- Learn a hypothesis within a **model** $h \in \mathcal{H} \quad h : \mathcal{X} \rightarrow \mathcal{Y}$

  such that h(**x**) ≈ y
- **Loss** function: how to quantify/weight prediction error between y and h(**x**)

Artificial Neural Networks

Learning goals

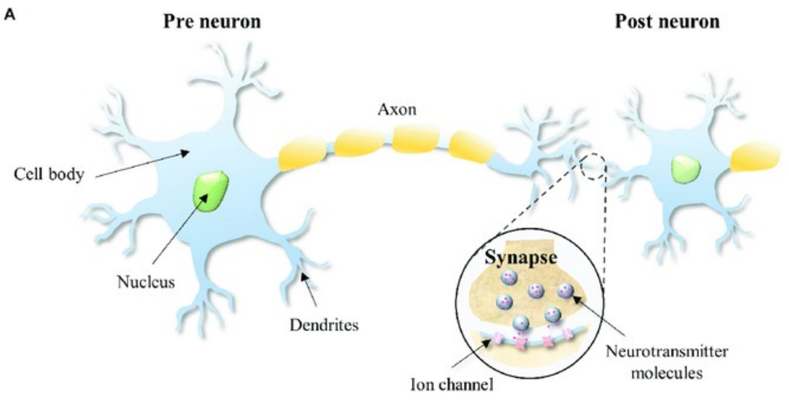
- Model of Artificial Neural Networks (NN)
- Algorithm for ERM on NN
- Gradient descent for NN
- Activation functions and loss functions for NN
- Neural networks in Python
- More complex NN (Convolutional NN, Recurrent NN, Autoencoder, Word2Vec,...)

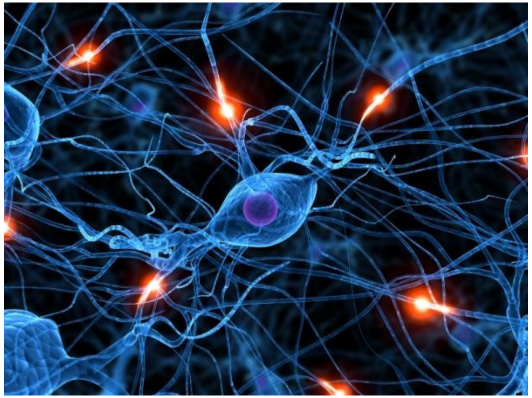
Artificial neural networks

- Artificial neural networks (ANN or NN) are just another **model** for supervised ML
- Find an hypothesis map h out of a hypothesis space H that minimizes a loss over a training set (ERM)
- H is the space of neural network hypotheses
- The hypothesis space (might) include **highly non-linear functions**

Biological neural networks

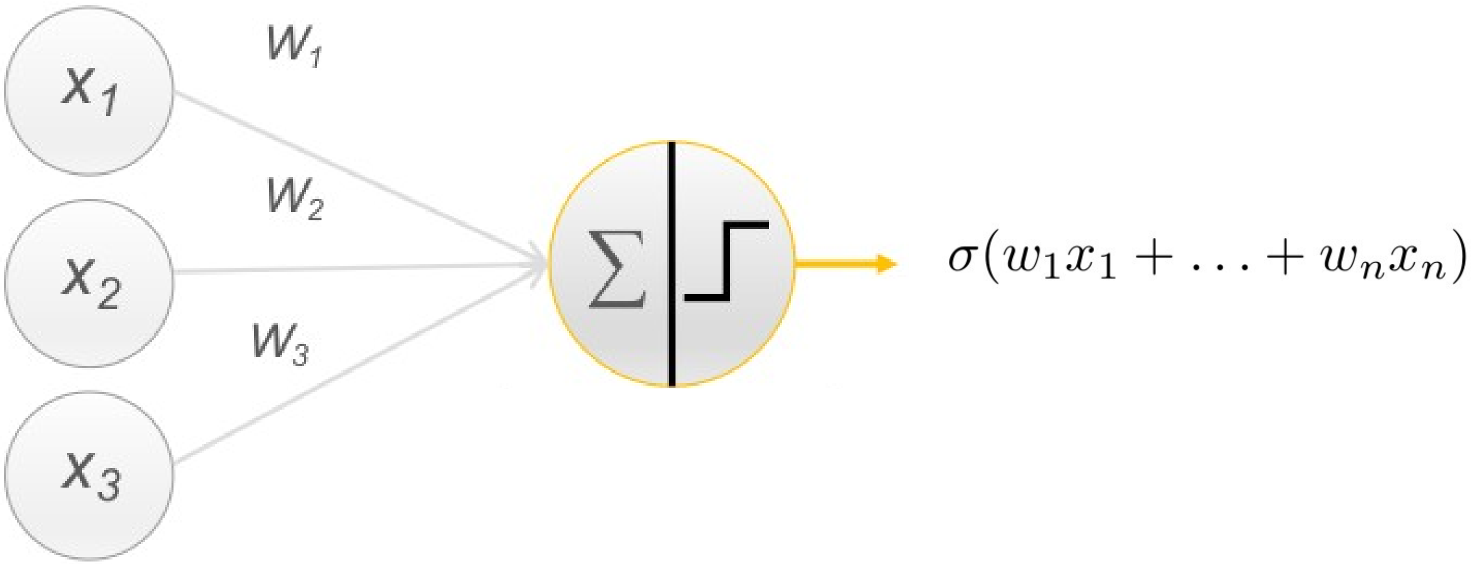
- Artificial neural networks inspired by **biological neural networks**
- Structure of the brain
  - **Neurons** as **elaboration units**
  - **Synapses** as **connection network**





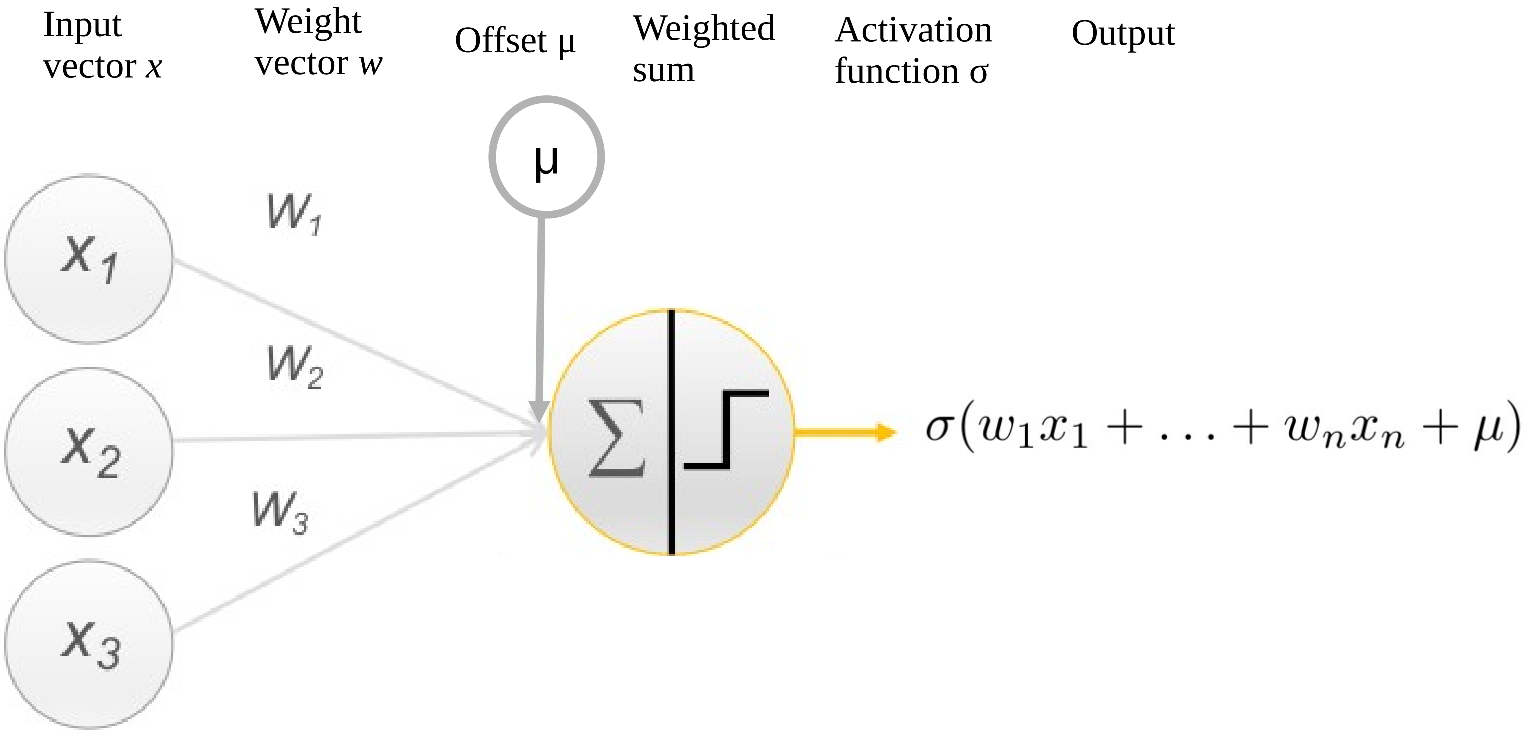
Structure of an artificial neuron

Input vector *x* | Weight vector *w* | Weighted sum | Activation function σ | Output



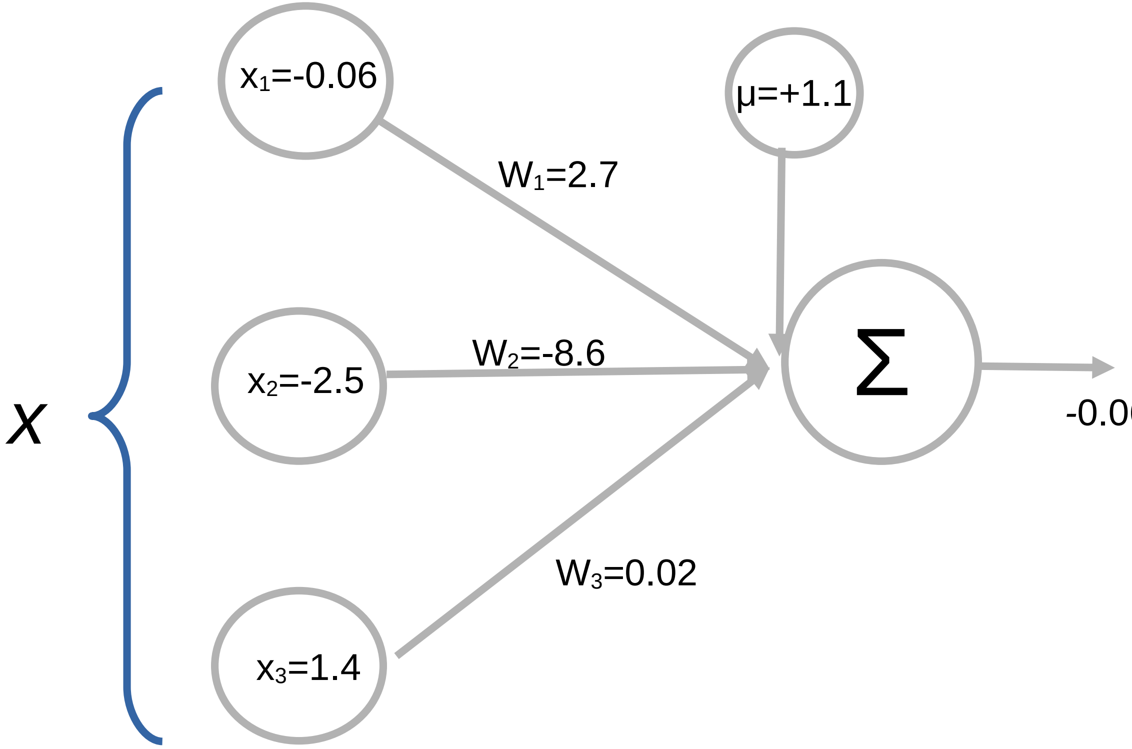
Structure of an artificial neuron

Often another weight not multiplied by the input vector is added: offset or bias



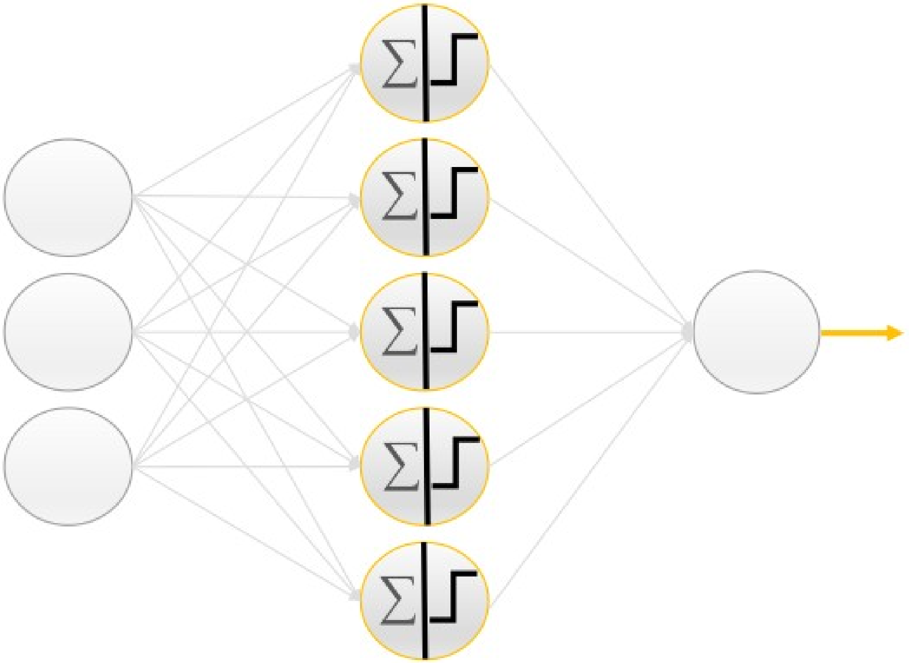
Equivalently, we can add a new input (x_4 in the example) and define its value to be fixed to 1

Structure of an artificial neuron



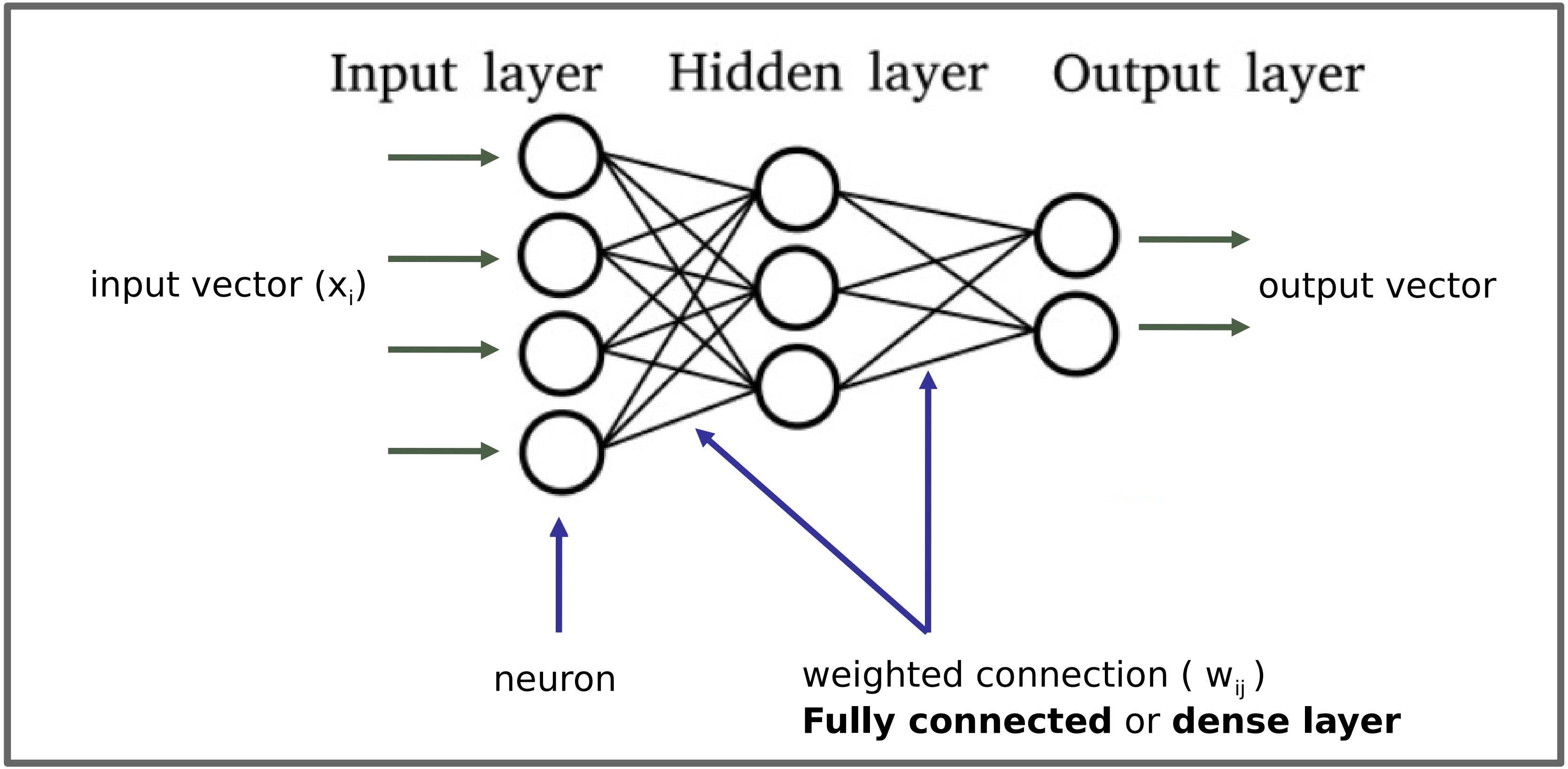
$$-0.06\times 2.7 + (-2.5)\times(-8.6) + 1.4\times 0.02 + 1.1 = 22.44$$

NN – stacked elementary units



NN – stacked elementary units

These are called **feed-forward neural networks** or **multilayer perceptron**



Model (signal-flow chart graphical representation)

Hyper-parameters of a NN

- What defines the model:
  - Number of layers
  - Number of neurons for each layer
  - Activation functions for the neurons

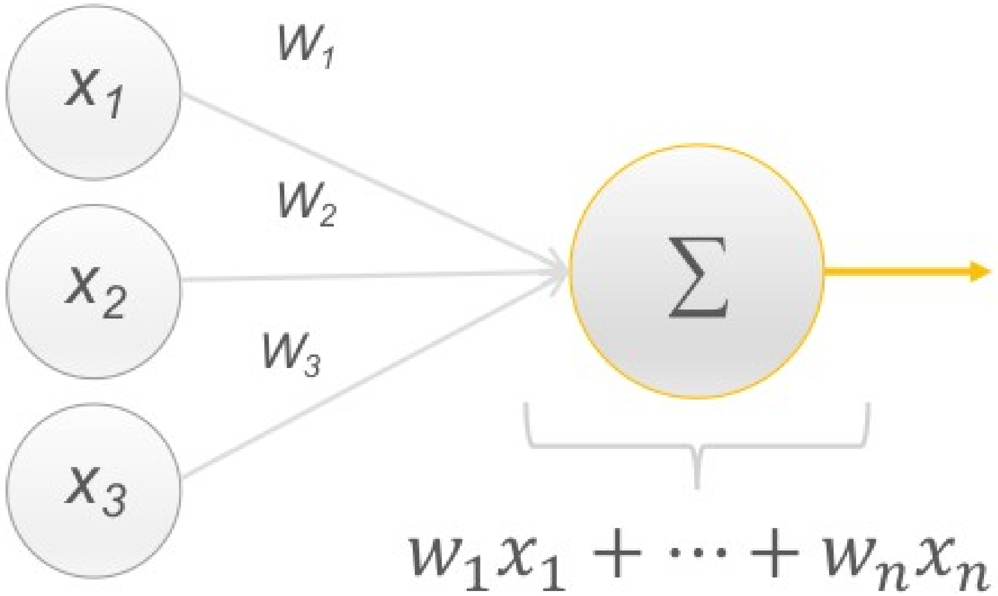
Hyper-parameters of a NN

- What defines the model:
  - Number of layers
  - Number of neurons for each layer
  - Activation functions for the neurons
- Hyper-parameter tuning as usual:
  - Validation curve
  - Grid search
  - ...

Parameters of a NN

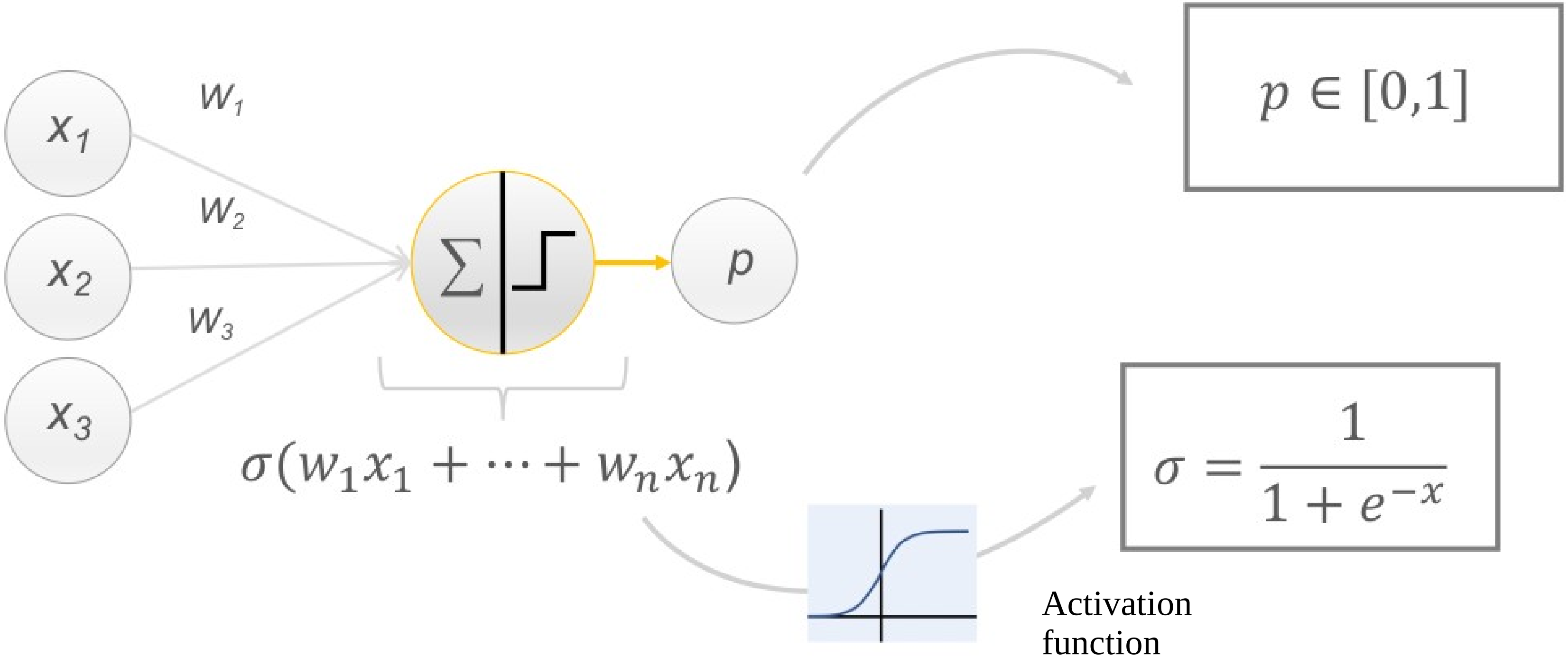
- What defines the hypothesis:
  - Weights and offsets (biases)
- Trained through ERM

Linear regression is a NN



A **single neuron without activation function** (nor bias)

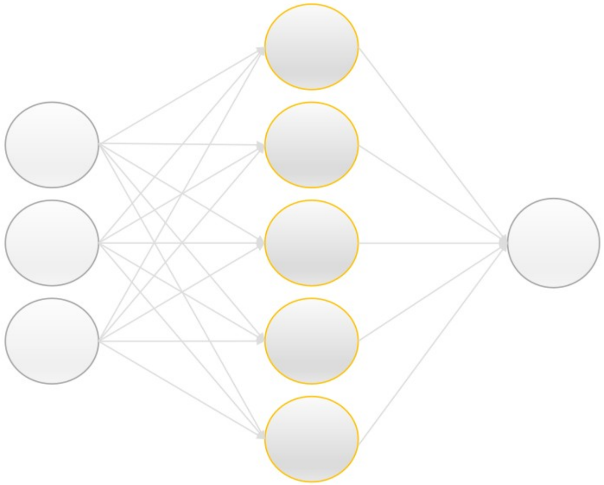
Logistic regression is a NN



A **single neuron** with a **sigmoid as activation function**

Training a NN

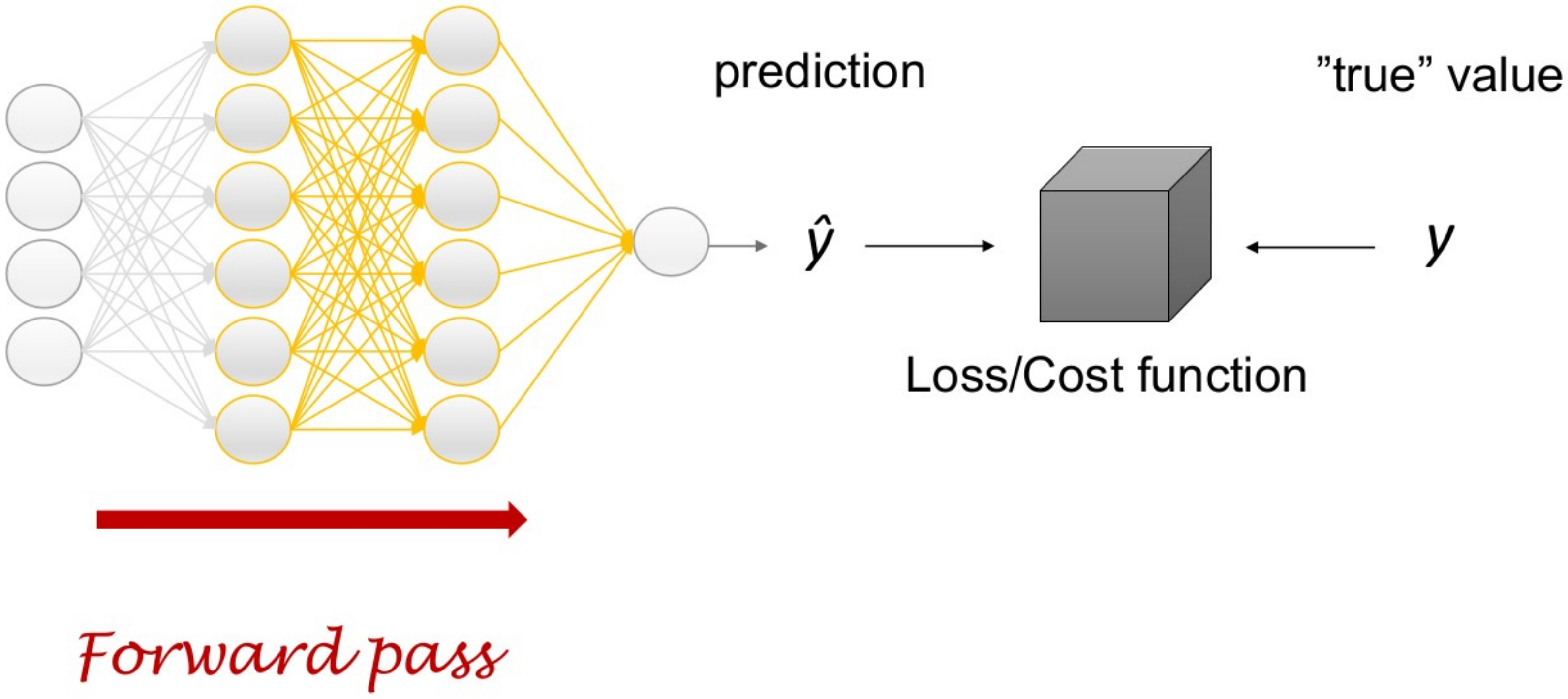
- Given a model, how to find a good hypothesis?
- For each neuron, definition of:
  - set of weights
  - offset value
- For example, total number of parameters in figure: 3*5+5*1



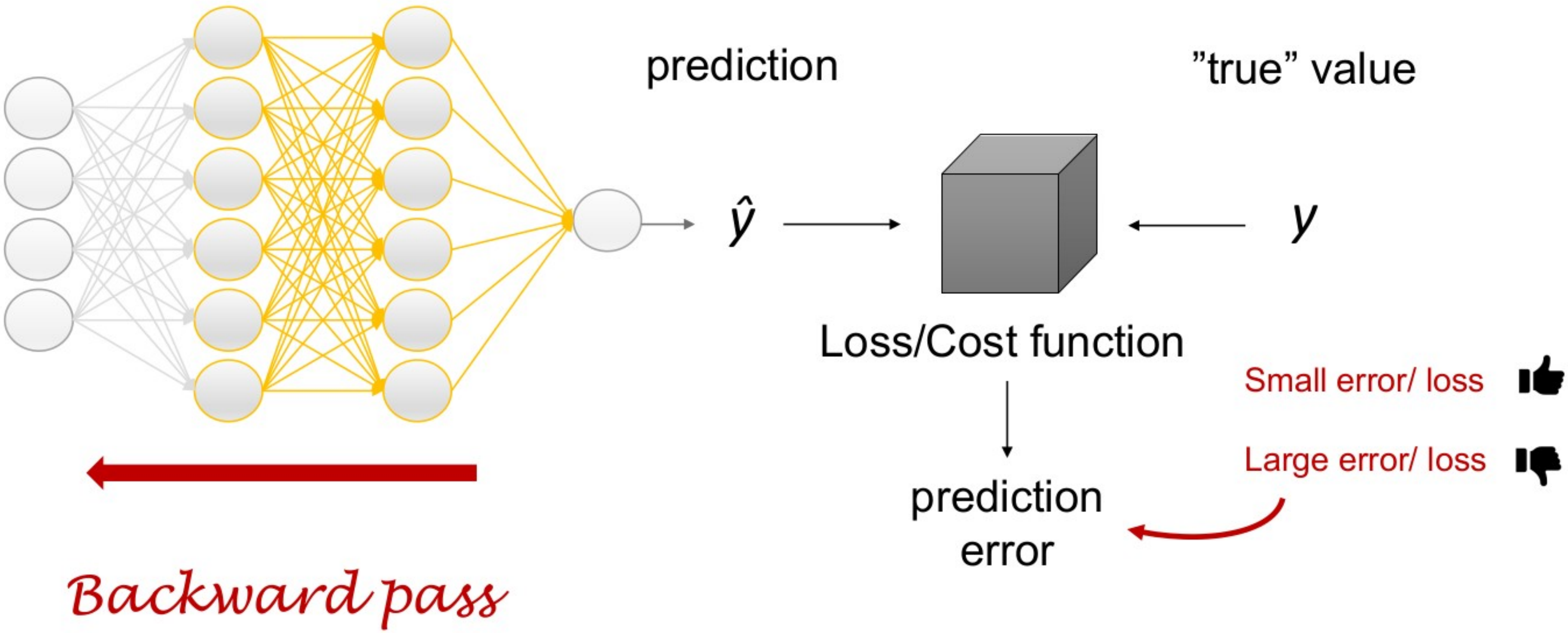
Training a NN

- Define a **loss function** to have an empirical error over a training set
- **Iterative approach** on training data instances to **solve ERM**
- **Backpropagation of errors** with **gradient descent algorithm**

Gradient descent algorithm



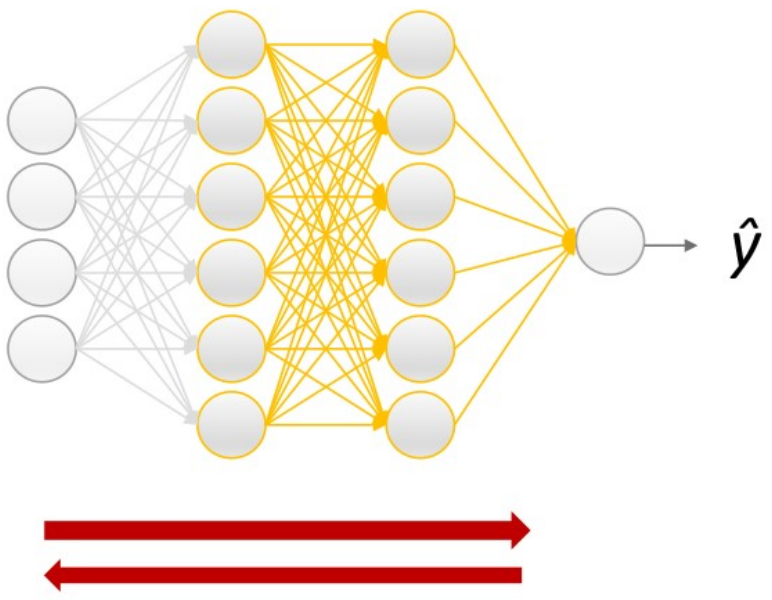
Gradient descent algorithm

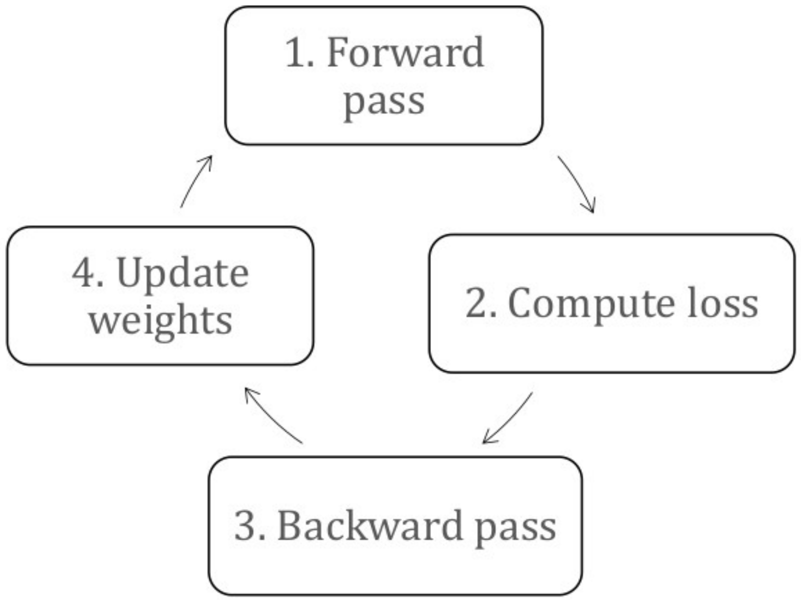


The backward pass is a **gradient descent step** using the the **chain rule to compute the gradients at different layers**

https://www.mathsisfun.com/calculus/derivatives-rules.html

Gradient descent algorithm





Gradient descent algorithm

Algorithm:

1) Initially assign random values to weights and offsets
2) Forward pass: process instances **x** in the training set one at a time
   - For each neuron, compute the result when applying weights, offset and activation function for the instance
   - Forward propagation until the output is computed h(**x**)
   - Compare the computed output h(**x**) with the expected output y, and compute loss (error)
3) Backward pass: backpropagation of the error (one sample at a time or in batches or total empirical risk)
   - Compute an estimate of the gradient starting from the last layer to the first layer
   - Updating weights and offset for each neuron → (stochastic) gradient descent step
4) Go back to step 2

Gradient descent algorithm

- The process ends when
  - The maximum number of epochs is reached
  - % of loss (or of loss variation) below a given threshold (or metric above given threshold)
  - % of parameter variation below a given threshold

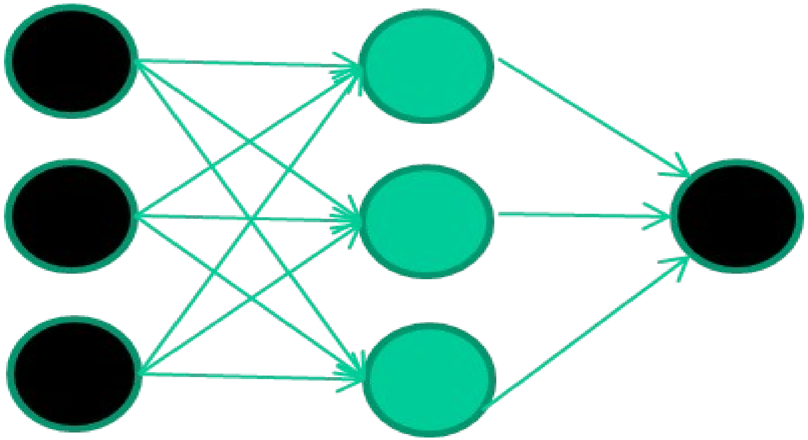
Training a NN

Backpropagation of the error one sample at a time

Data points

Features x			***Class y***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
…			

Initialise with random weights



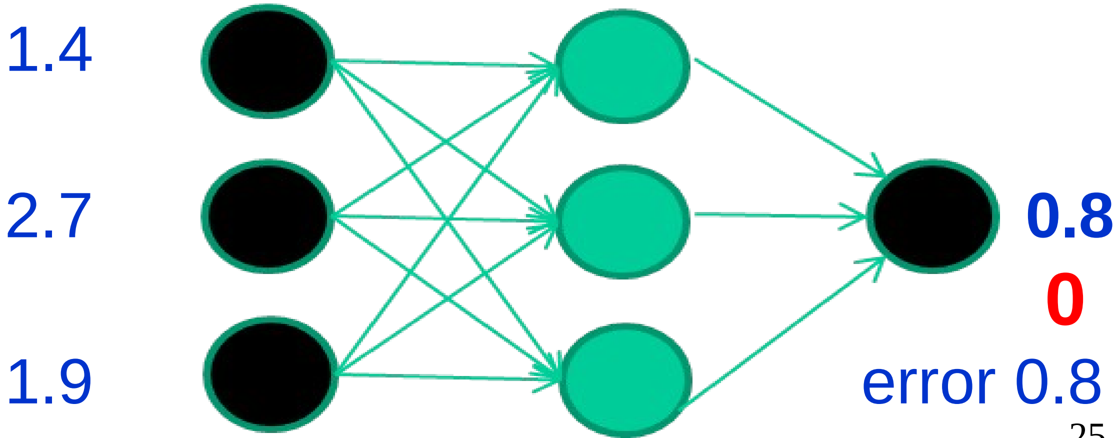
Training a NN

Backpropagation of the error one sample at a time

Forward pass:
Predict h(x) of a sample x and compute loss

Data points

Features			***Class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
…			



Training a NN

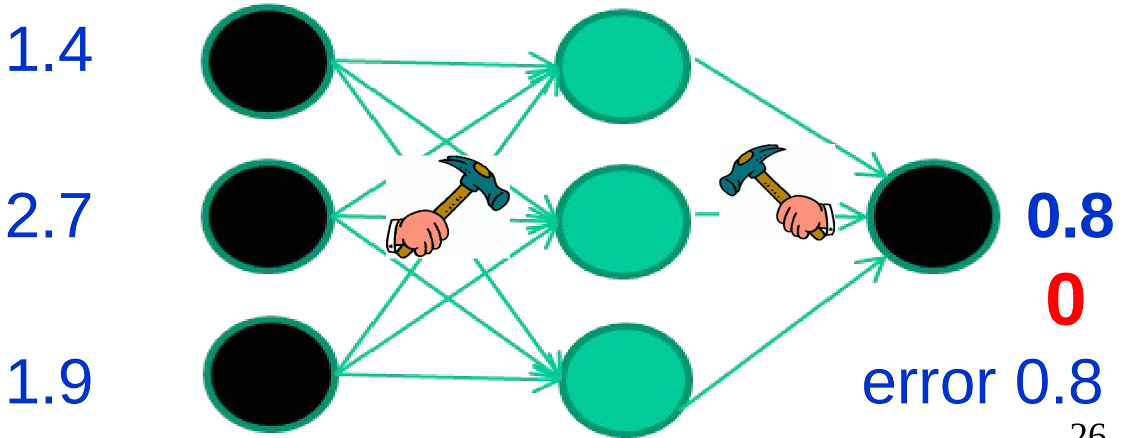
Backpropagation of the error one sample at a time

Backward pass:
Compute gradient and adjust weights with a gradient descent step

Data points

Features			***Class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

…



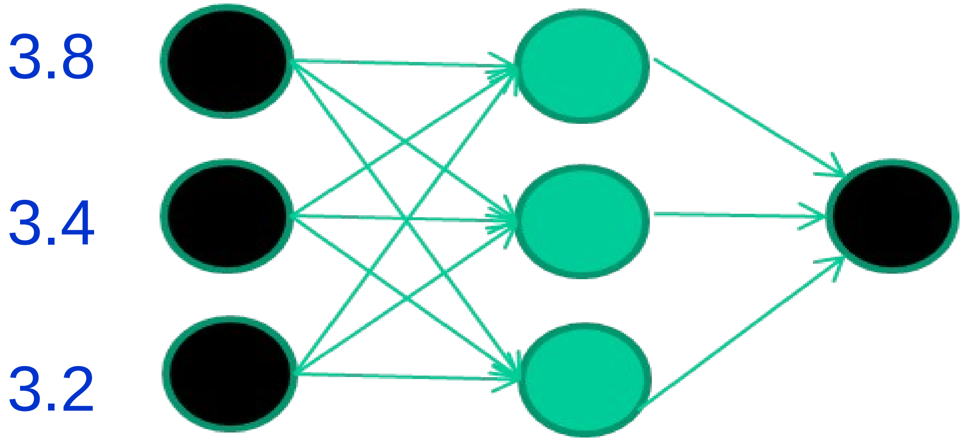
Training a NN

Backpropagation of the error one sample at a time

Data points

Features			***Class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
…			

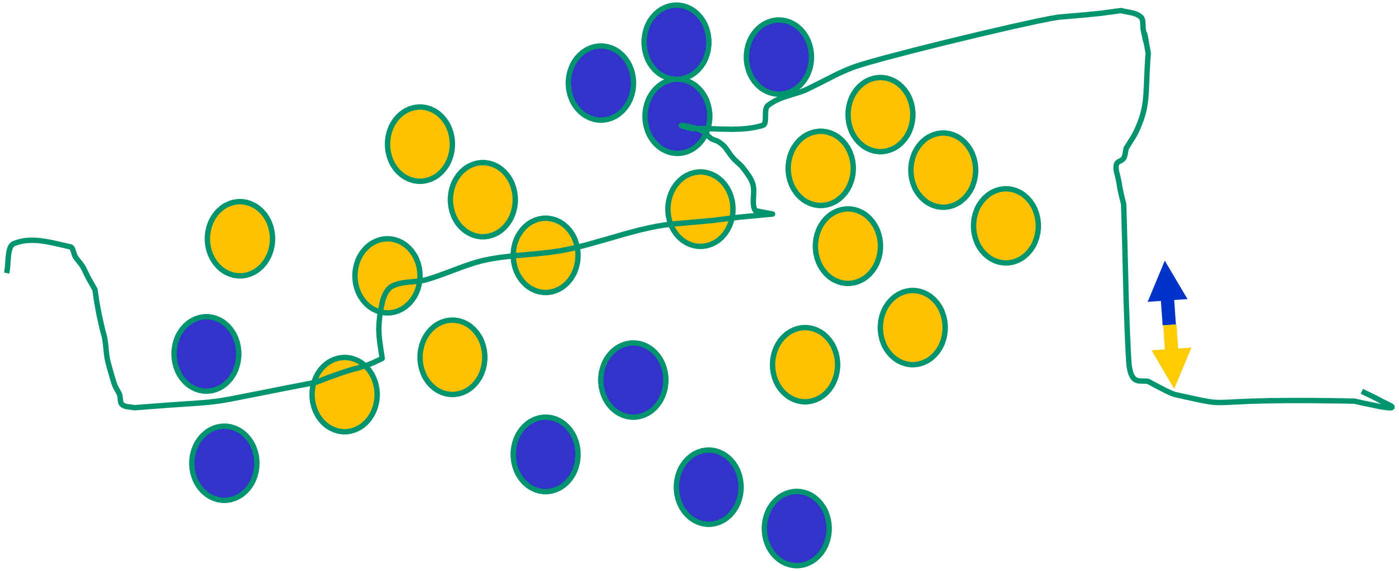
Repeat iteratively with other samples



Training a NN – decision boundary perspective

2D, binary classification problem

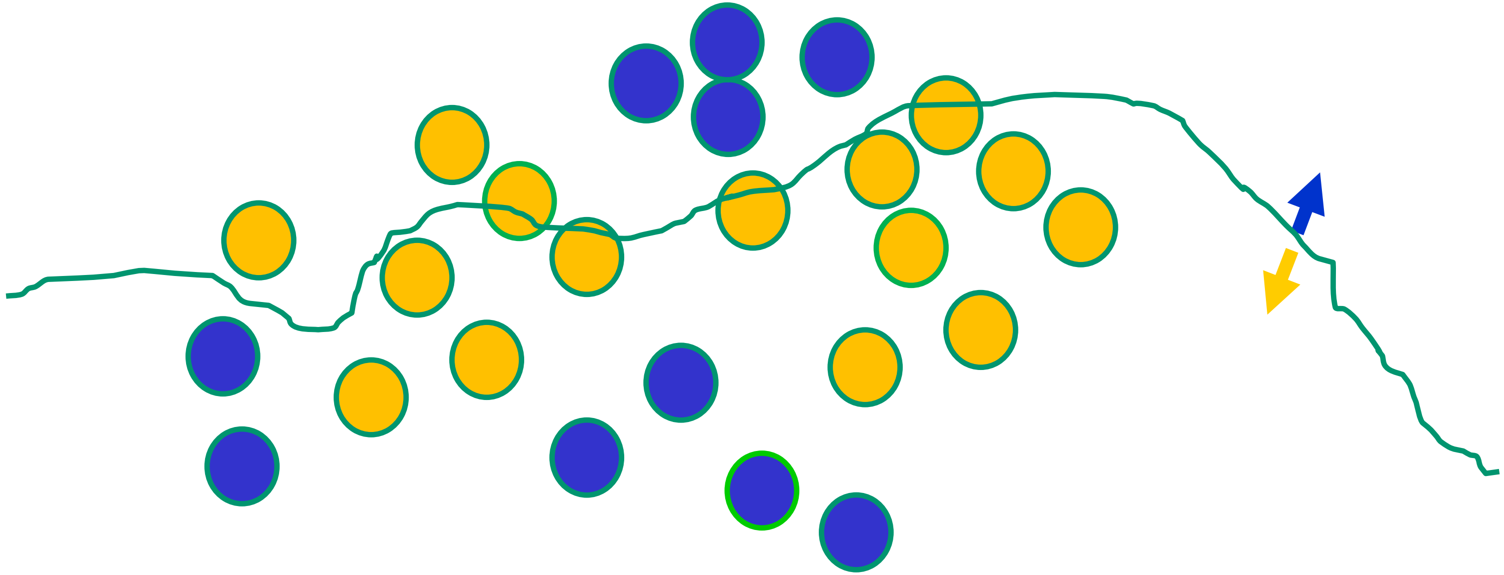
Initial random weights



Training a NN – decision boundary perspective

2D, binary classification problem

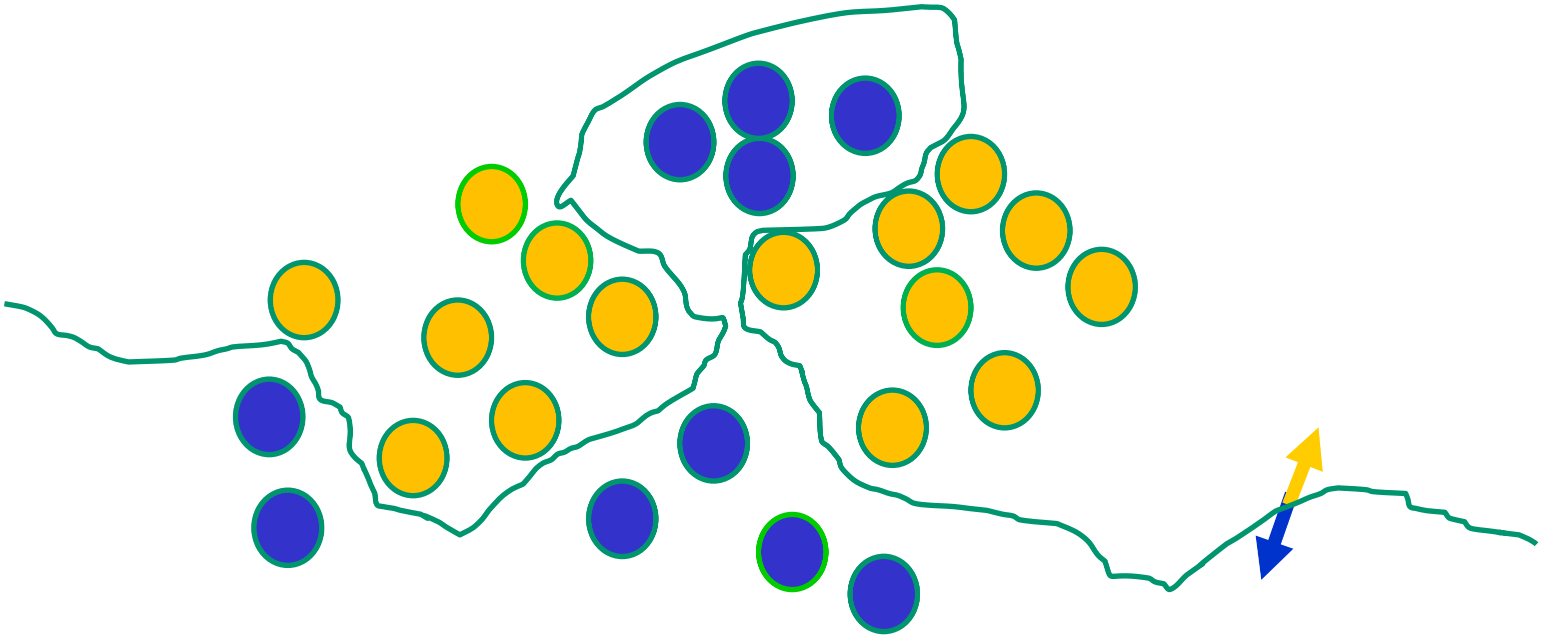
After first iteration



Training a NN – decision boundary perspective

2D, binary classification problem

Eventually you can solve the ERM



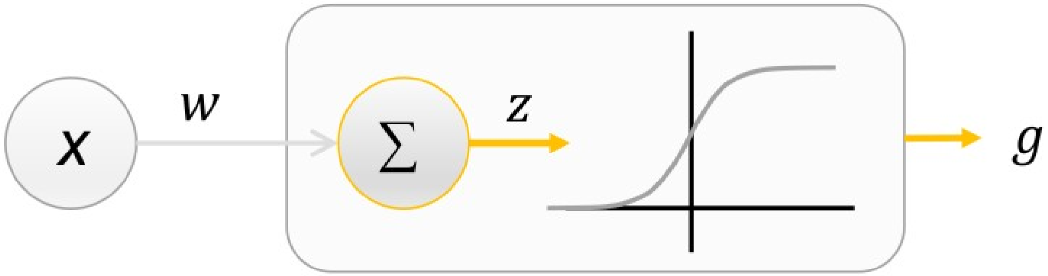
Gradient descent step

Single neuron with single input

Forward Pass

Loss (If y in {0,1})

$$z = wx \qquad g = \frac{1}{1+e^{-z}} \qquad L = -y\,ln(g) - (1-y)\ln(1-g)$$



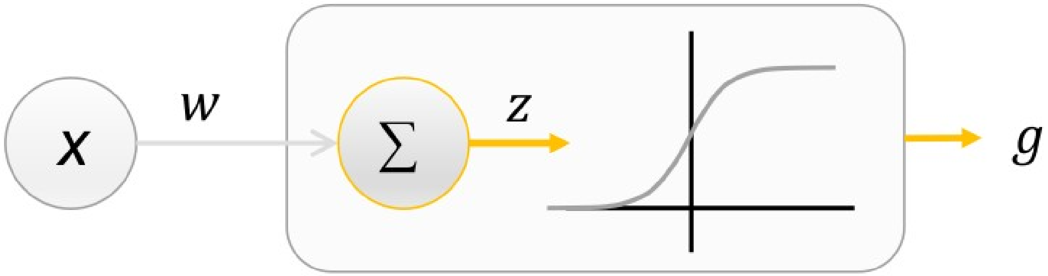
Gradient descent step

Single neuron with single input

Forward Pass

Loss (If y in {0,1})

$$z = wx \qquad g = \frac{1}{1 + e^{-z}} \qquad L = -y\,ln(g) - (1-y)\ln(1-g)$$



Full gradient

$$\nabla L(w) = \frac{\partial L}{\partial w} = \frac{\partial L}{\partial g}\frac{\partial g}{\partial z}\frac{\partial z}{\partial w}$$

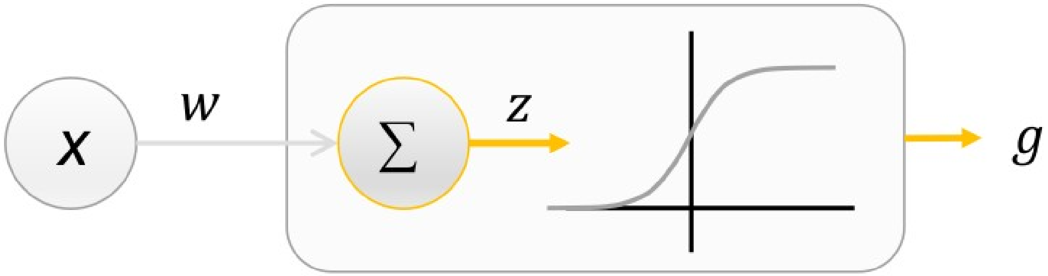
Gradient descent step

Single neuron with single input

Forward Pass

Loss (If y in {0,1})

$$z = wx \qquad g = \frac{1}{1+e^{-z}} \qquad L = -y\,ln(g) - (1-y)\ln(1-g)$$



Local gradients

$$\frac{\partial z}{\partial w} = x \qquad \frac{\partial g}{\partial z} = g(1-g) \qquad \frac{\partial L}{\partial g} = -\frac{y}{g} + \frac{1-y}{1-g}$$

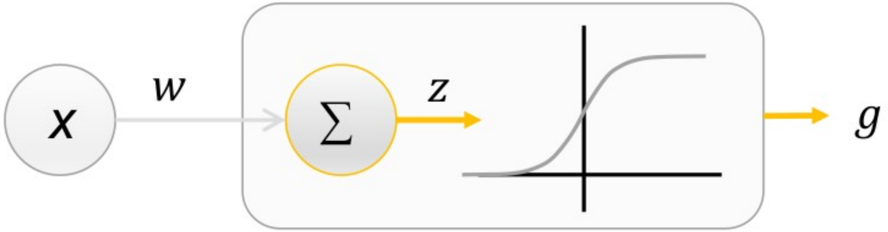
Gradient descent step

Single neuron with single input

Forward Pass

Loss (If y in {0,1})

$z = wx \qquad g = \frac{1}{1+e^{-z}} \qquad L = -y\,ln(g) - (1-y)\ln(1-g)$



Backward Pass

$$\text{(Gradient Step)} \quad \underbrace{\mathbf{w}^{(k+1)}}_{\text{new guess}} = \underbrace{\mathbf{w}^{(k)}}_{\text{current guess}} - \underbrace{\alpha}_{\text{step size}} \nabla f\left(\mathbf{w}^{(k)}\right).$$

Gradient descent step

Gradient Descent

$$\text{(Gradient Step)} \quad \underbrace{\mathbf{w}^{(k+1)}}_{\text{new guess}} = \underbrace{\mathbf{w}^{(k)}}_{\text{current guess}} - \underbrace{\alpha}_{\text{step size}} \nabla f\left(\mathbf{w}^{(k)}\right)$$

f is empirical risk

mini-batch Gradient Descent

$$\text{(Noisy Gradient Step)} \quad \underbrace{\mathbf{w}^{(k+1)}}_{\text{new guess}} = \underbrace{\mathbf{w}^{(k)}}_{\text{current guess}} - \underbrace{\alpha^{(k)}}_{\text{step size}} \mathbf{g}^{(k)} \text{ with } \mathbf{g}^{(k)} \approx \nabla f\left(\mathbf{w}^{(k)}\right)$$

- There are many variations of gradient descent algorithms

  https://arxiv.org/pdf/1609.04747.pdf

Gradient descent algorithm

- Hyper-parameters to set:
  - **Learning rate α** (step size) – by what amount we adjust/ tune/ update model's parameters (weights)
  - **Number of iterations** (epochs) - how many times we update model's weight
  - **Batch size** for stochastic gradient descent
  - **Version of the optimizer** (of the gradient descent)

Gradient descent algorithm

- **Note:** Neural networks with multiple layers and non-linear activation functions can be non-convex → multiple local minima
  - This is not always a problem:
    - Local minima can still be good and generalize (less overfitting than global optimum one) due to overparametrization of neural networks
    - Saddle points more common than (poor) local minima
  - Solutions:
    - Good initialization of weight
    - Modify classical gradient descent step (momentum, adaptive learning rate,..)
    - Regularization techniques (dropout, weight decay,...)

https://datascience.stackexchange.com/questions/87961/explanation-of-why-neural-networks-are-non-convex
https://arxiv.org/abs/1611.06310
https://stats.stackexchange.com/questions/106334/cost-function-of-neural-network-is-non-convex

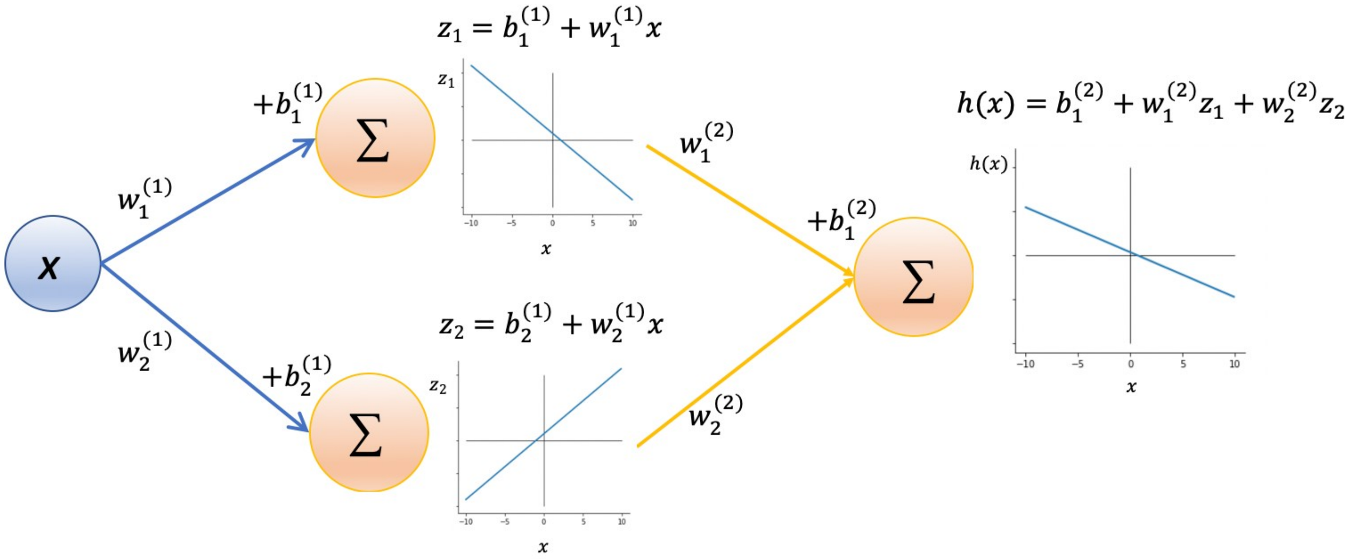
Activation functions

- Simulates biological activation to input stimuli
- Provides **non-linearity** to the computation
- May help to saturate neuron outputs in **fixed ranges**
- We will see some popular choices

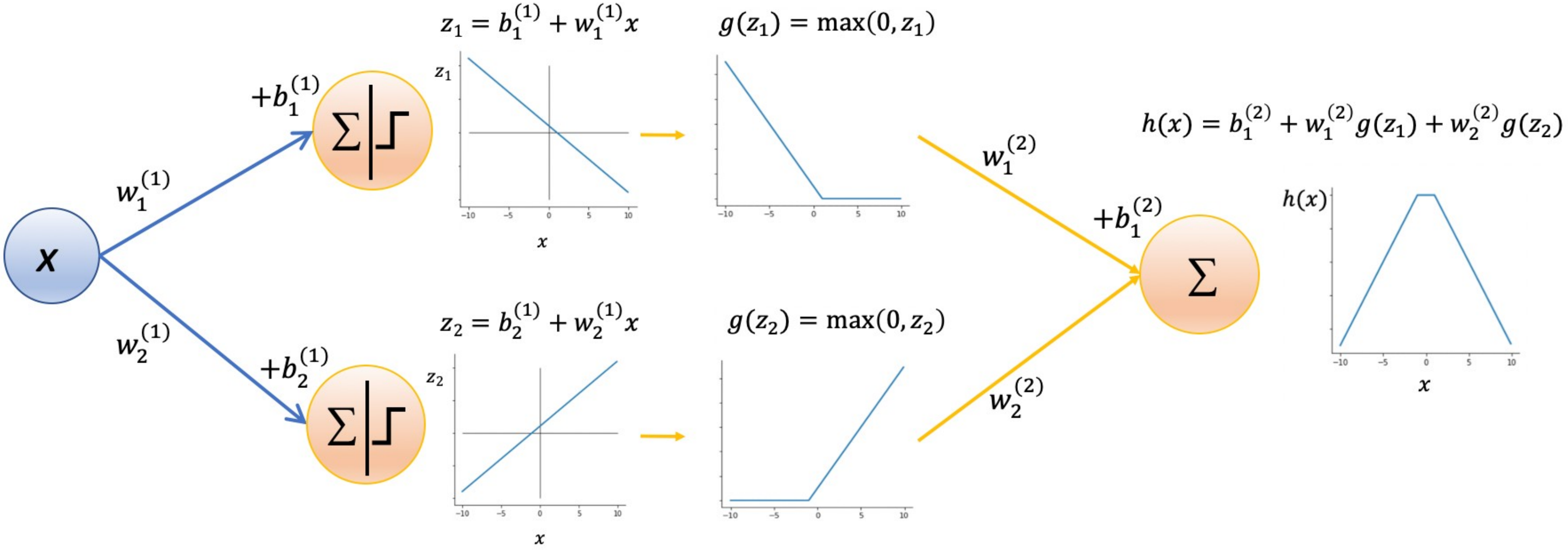
https://en.wikipedia.org/wiki/Activation_function

No activation function

- What happens if there is no activation function (or a linear one)?
- No matter how many layers we stack on top of each other, the overall behavior of the NN will always be a linear map
- Example: NN with one input, two neurons in a hidden layer and one output neuron without adding any activation function. The network will always return a linear predictor, no matter the bias and weight values of the neurons.



Activation functions allows non-linearity

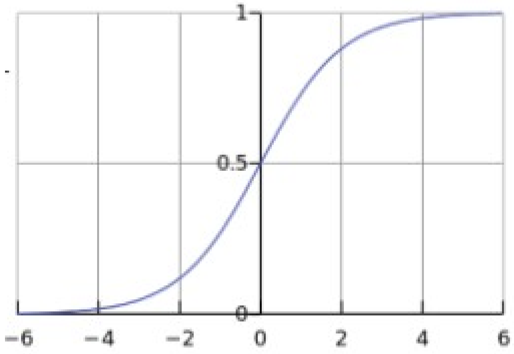


Activation functions: sigmoid, tanh

- Saturate input value in a fixed range
- Non linear for all the input scale
- Typically used for both hidden and output layers
  - E.g. sigmoid in output layers allows generating values between 0 and 1 (useful when output must be interpreted as likelihood/probability)

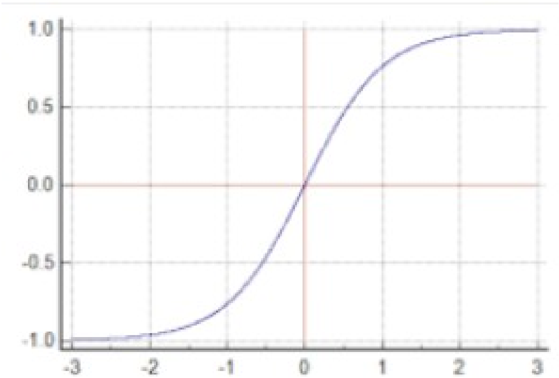
Sigmoid (logistic, soft step)

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



Hyperbolic tangent (tanh)

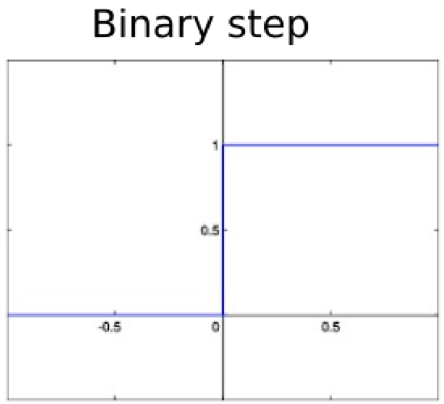
$$\tanh(x) \doteq \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$



Activation functions: binary step, ReLU

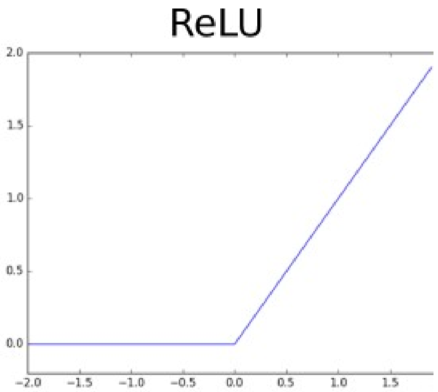
- **Binary Step (Heaviside step function)**
  - outputs 1 when input is positive
  - useful for binary outputs
  - issues: not appropriate for gradient descent
  - derivative not defined in x=0

$$H(x) := \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases}$$



- **ReLU (Rectified Linear Unit)**
  - neurons activate linearly only for positive input
  - used in deep NN (e.g. CNNs)
    - does not saturate
    - avoids vanishing gradient

$$(x)^+ \doteq \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$$

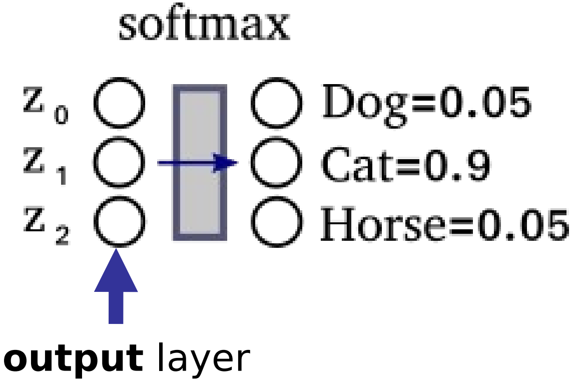


Activation functions: Softmax

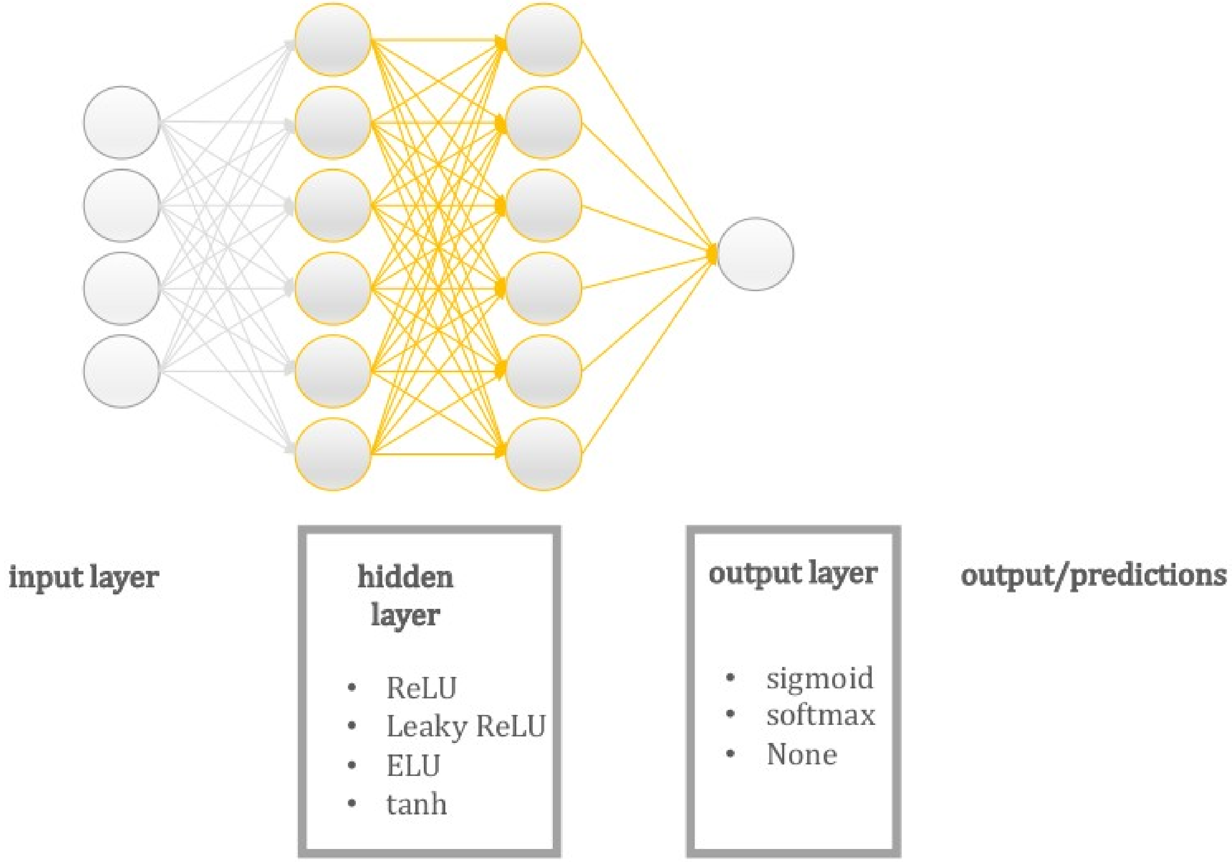
- Differently to other activation functions
  - works by considering all the neurons in the layer
  - it is usually applied only to the output layer
- After softmax, the output vector can be interpreted as a discrete distribution of probabilities
  - Generalization of sigmoid to multiple dimensions
  - Probabilities for the input pattern of belonging to each class (multiclass tasks)

$$softmax(z_j) = \frac{e^{z_j}}{\sum_{i=0}^{N-1} e^{z_i}}$$

N is the number of neurons in the layer

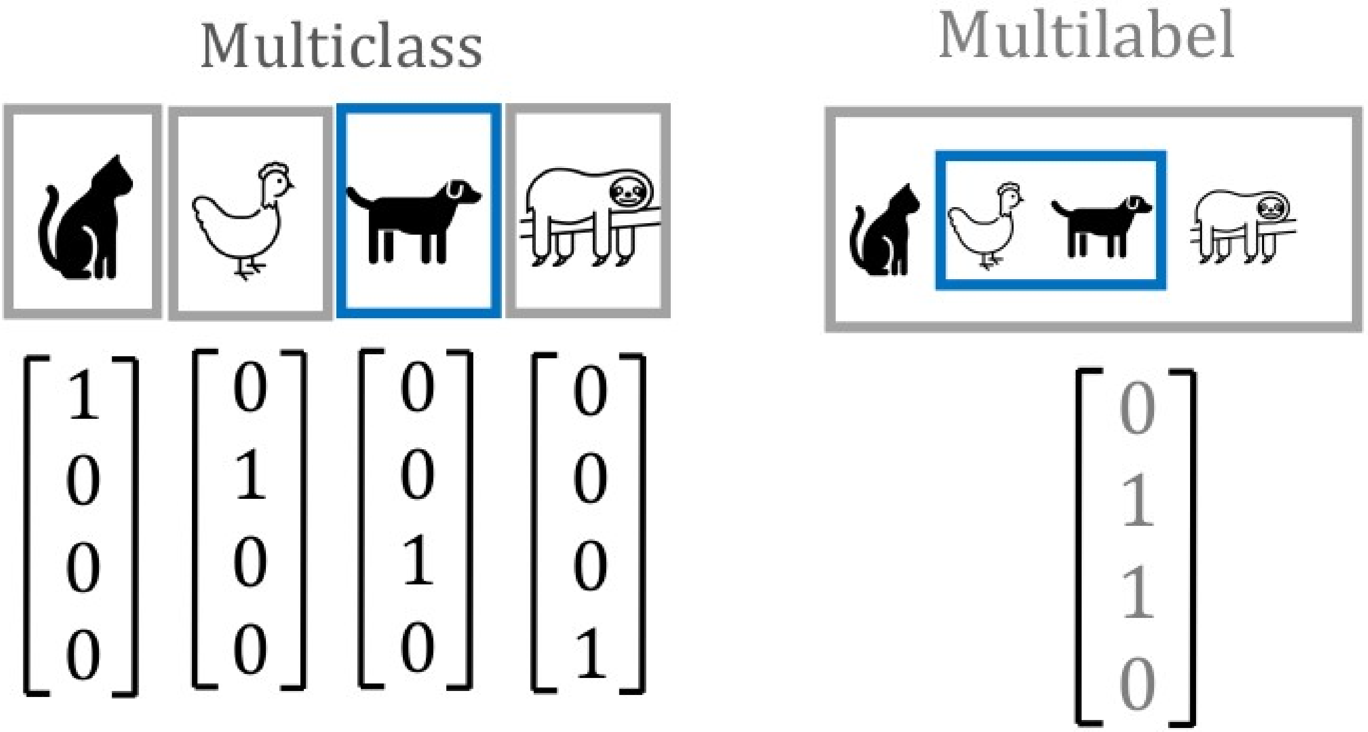


Activation functions

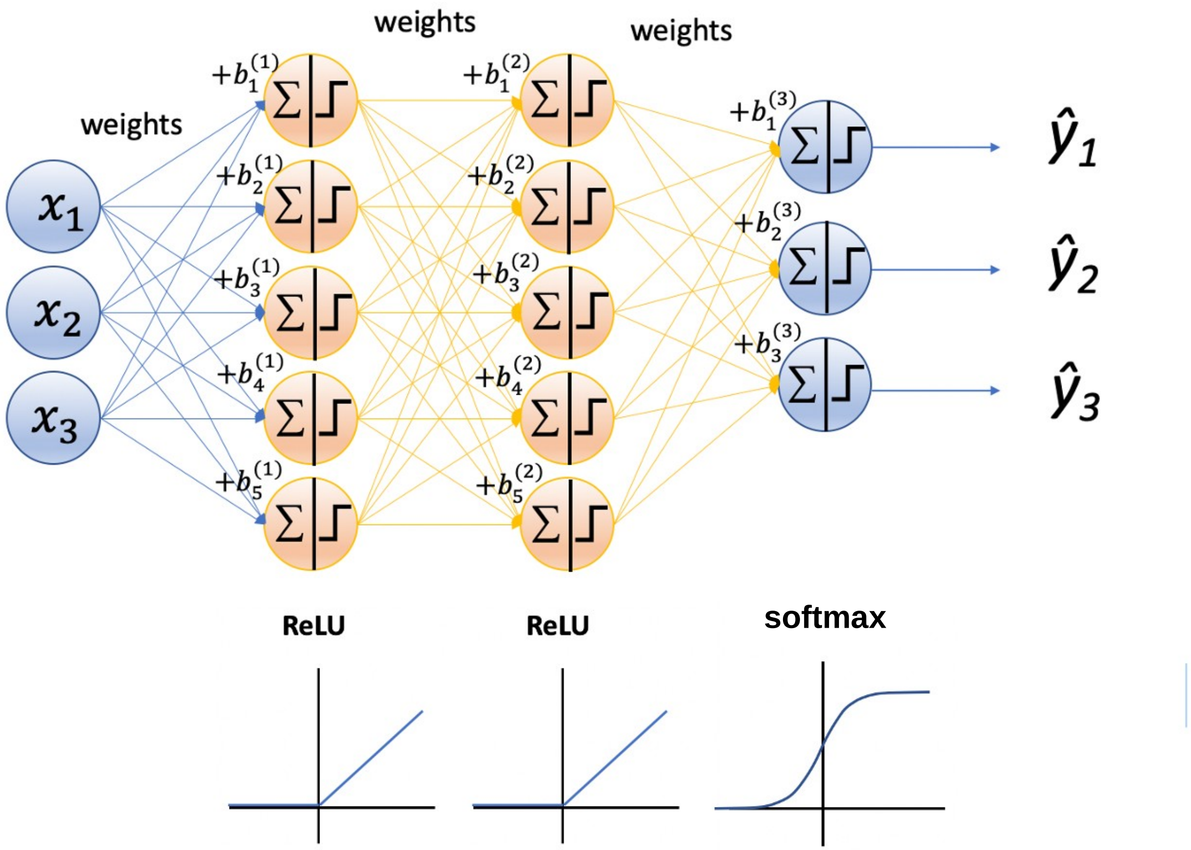


Classification tasks

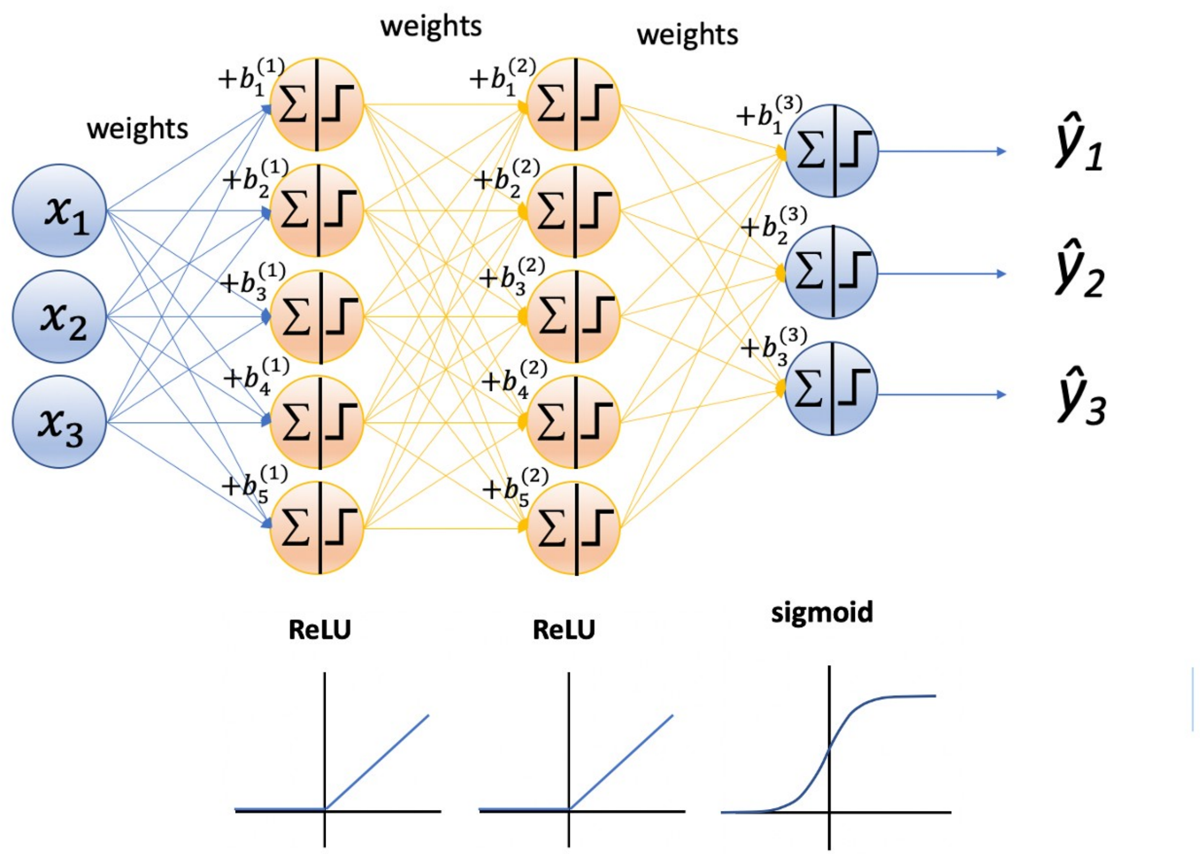
- spam
  y=1
- not spam
  y=0



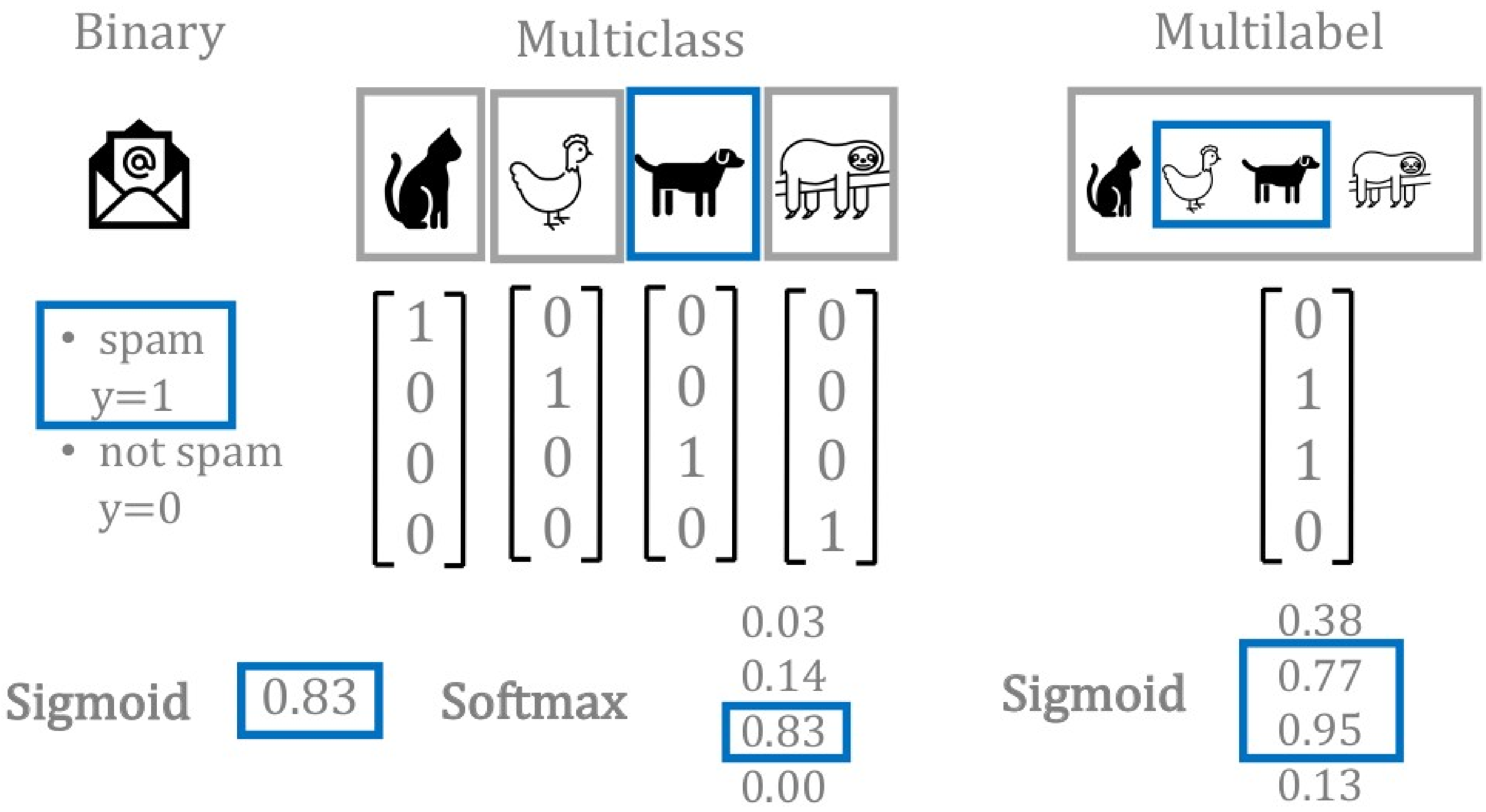
Example of a NN for multiclass task



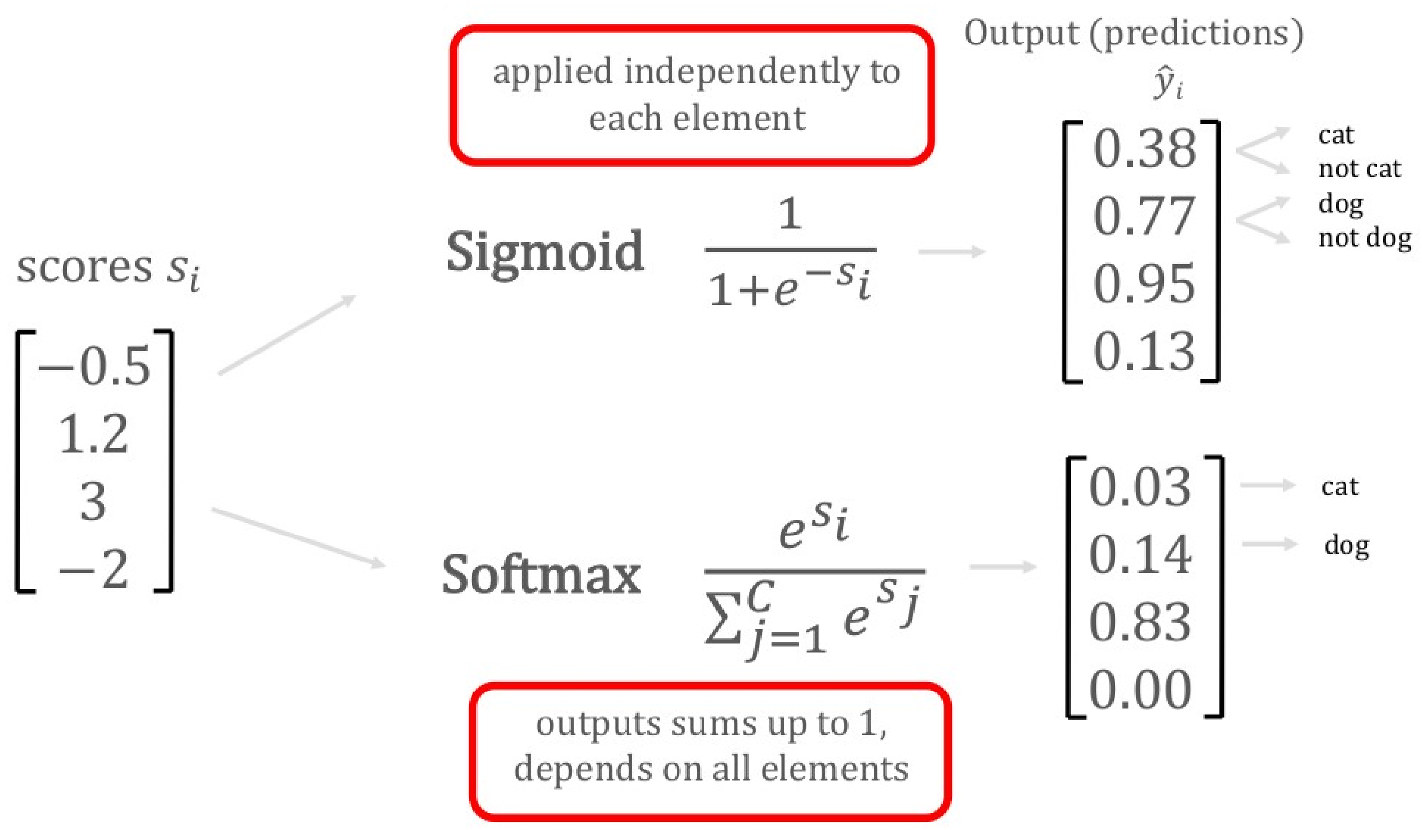
Example of a NN for multilabel task



Classification tasks



Activation functions - classification



Cross-entropy loss

n.o. classes

$$CE = -\sum_{i}^{C} y_i \log(\hat{y}_i)$$

+ Sigmoid

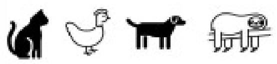
Binary CE

$$y_2 = 1 - y_1$$
$$\hat{y}_2 = 1 - \hat{y}_1$$

Binary

spam not spam

Multilabel



$$\begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}$$

cat
not cat

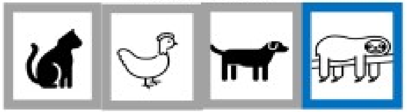
$$CE = -\sum_{i}^{C=2} y_i \log(\hat{y}_i) = -y_1 \log(\hat{y}_1) - (1 - y_1) \log(1 - \hat{y}_1)$$

More details: https://machinelearningmastery.com/cross-entropy-for-machine-learning/

Cross-entropy loss

$$CE = -\sum_{i}^{C} y_i \, log(\hat{y}_i)$$

Multiclass



$$\begin{bmatrix}1\\0\\0\\0\end{bmatrix} \begin{bmatrix}0\\1\\0\\0\end{bmatrix} \begin{bmatrix}0\\0\\1\\0\end{bmatrix} \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$$

y_i for non-positive class is zero

y_i for positive class is one

+ Softmax

Categorical CE

$$CE = -\sum_{i}^{C} y_i \, log(\hat{y}_i) = -\log\left(\frac{e^{s_p}}{\sum_{j=1}^{C} e^{s_j}}\right)$$

score for positive class

More details: https://machinelearningmastery.com/cross-entropy-for-machine-learning/

Last-layer activation function + loss

Table 4.1 Choosing the right last-layer activation and loss function for your model

Problem type	Last-layer activation	Loss function
Binary classification	`sigmoid`	`binary_crossentropy`
Multiclass, single-label classification	`softmax`	`categorical_crossentropy`
Multiclass, multilabel classification	`sigmoid`	`binary_crossentropy`
Regression to arbitrary values	None	`mse`
Regression to values between 0 and 1	`sigmoid`	`mse` or `binary_crossentropy`

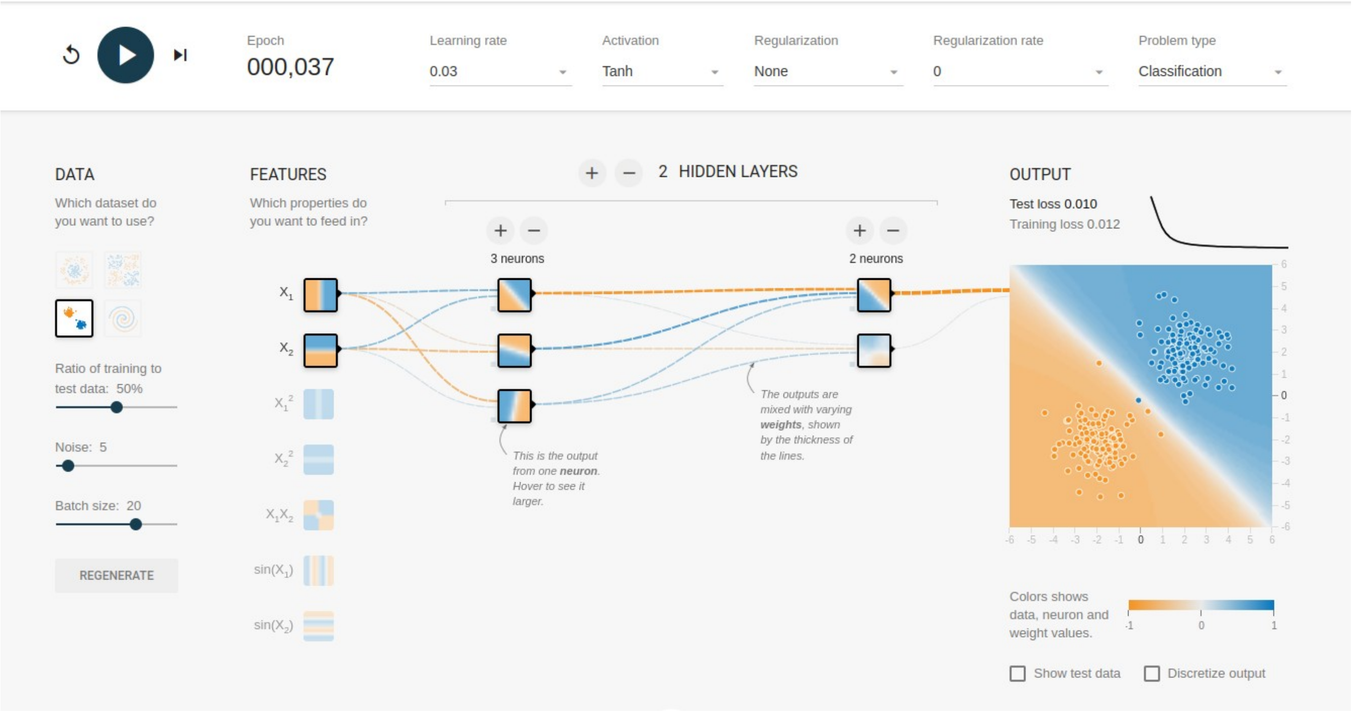
DL with Python, Chollet https://livebook.manning.com/book/deep-learning-with-python/chapter-4/190

Neural networks

- Issues
  - Long training time
  - Complex configuration (choice/tuning of hyper-parameters)
  - Not interpretable model (black box model)

Neural networks libraries

- Playground of TensorFlow: https://playground.tensorflow.org/
- Try yourself!



Neural networks in Python

- Scikit-learn

sklearn.neural_network.MLPClassifier

class sklearn.neural_network.**MLPClassifier**(*hidden_layer_sizes=(100,), activation='relu', *, solver='adam', alpha=0.0001, batch_size='auto', learning_rate='constant', learning_rate_init=0.001, power_t=0.5, max_iter=200, shuffle=True, random_state=None, tol=0.0001, verbose=False, warm_start=False, momentum=0.9, nesterovs_momentum=True, early_stopping=False, validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08, n_iter_no_change=10, max_fun=15000*) [source]

Multi-layer Perceptron classifier.

This model optimizes the log-loss function using LBFGS or stochastic gradient descent.

sklearn.neural_network.MLPRegressor

class sklearn.neural_network.**MLPRegressor**(*hidden_layer_sizes=(100,), activation='relu', *, solver='adam', alpha=0.0001, batch_size='auto', learning_rate='constant', learning_rate_init=0.001, power_t=0.5, max_iter=200, shuffle=True, random_state=None, tol=0.0001, verbose=False, warm_start=False, momentum=0.9, nesterovs_momentum=True, early_stopping=False, validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08, n_iter_no_change=10, max_fun=15000*) [source]

Multi-layer Perceptron regressor.

This model optimizes the squared error using LBFGS or stochastic gradient descent.

https://scikit-learn.org/stable/modules/neural_networks_supervised.html
https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

Neural networks in Python

- Scikit-learn

  Class MLPClassifier implements a feed forward neural network (multi-layer perceptron) that trains using backpropagation

```
from sklearn.neural_network import MLPClassifier

clf = MLPClassifier(hidden_layer_sizes=(5, 2), solver='adam', alpha=1e-5)

model=clf.fit(X_train, y_train)
```

https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

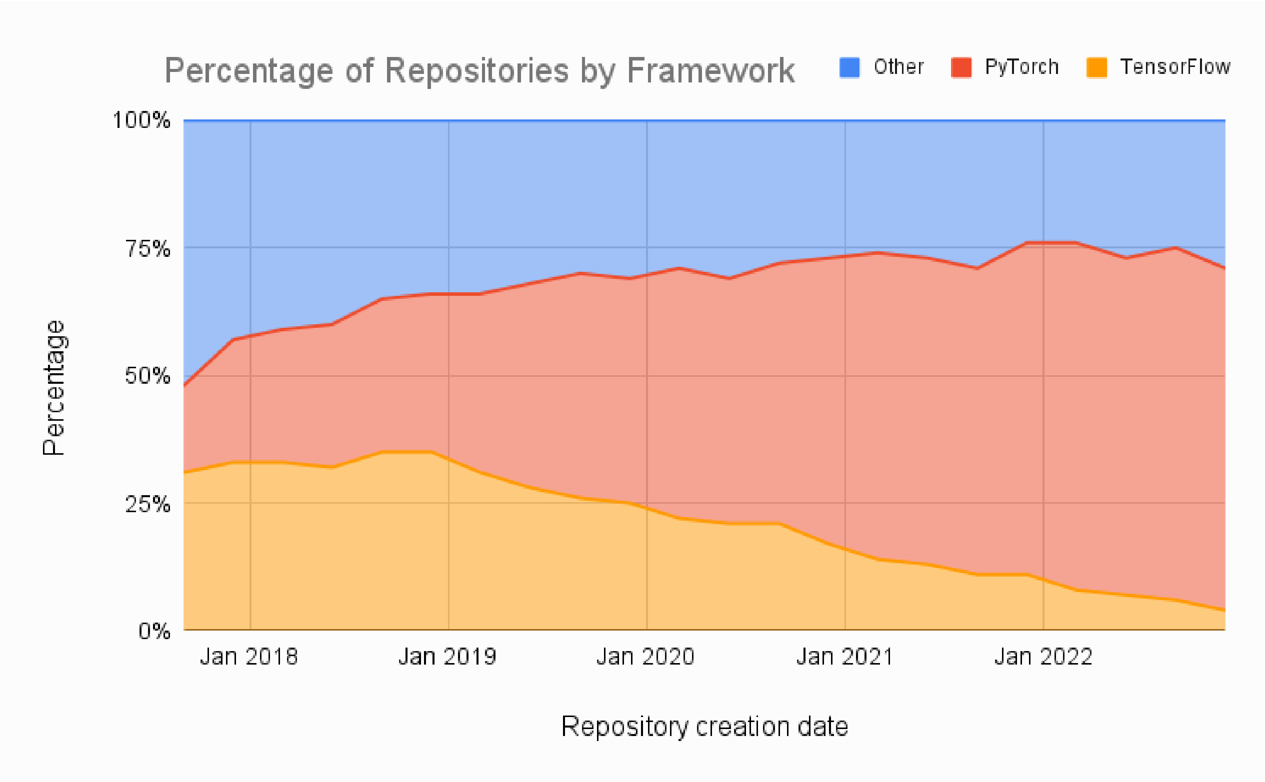
Neural networks in Python

- Pytorch
  - Library for advanced machine learning (deep learning)
  - One of the most popular in academia and industry
  - Allows acceleration via GPU
  - Works on multidimensional arrays, called tensors
  - What you will use in Lab 10!

PyTorch

https://pytorch.org/docs/stable/index.html
https://pytorch.org/tutorials/
https://pytorch.org/tutorials/beginner/blitz/neural_networks_tutorial.html

Neural networks in Python



https://www.assemblyai.com/blog/pytorch-vs-tensorflow-in-2023/

Neural Networks -
More complex architectures

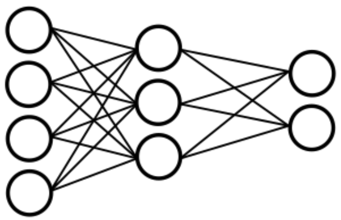
Deep neural networks

- Deep neural networks are neural networks with “many” layers (even billions of neurons)
- They often allow to use raw input
- The multiple layers are used to progressively extract higher-level features from the raw input
- Often deep learning uses more complex layers than the one we have seen in feed-forward neural networks

Artificial neural networks

- Different tasks, different architectures

numerical vectors classification/regression:
feed forward NN (what we have seen so far)



time series analysis: **recurrent NN** (RNN)

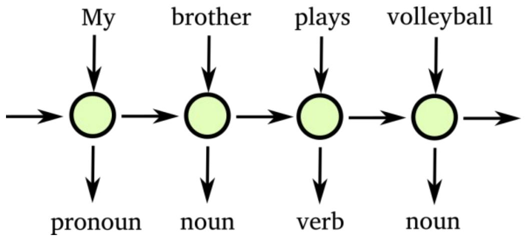
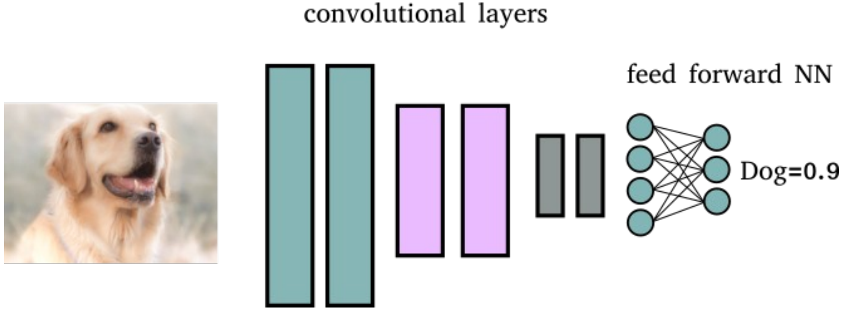
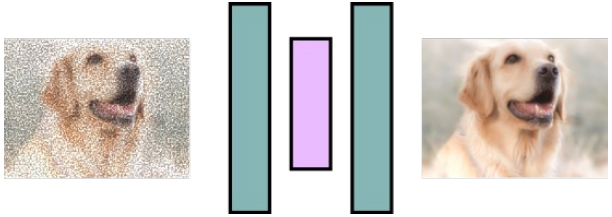


image understanding: **convolutional NN** (CNN)



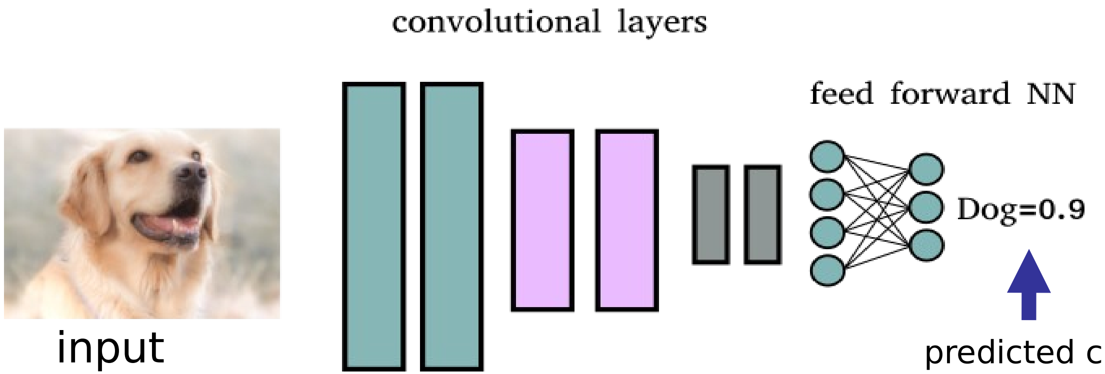
denoising: **auto-encoders**



Convolutional neural networks

- Allow automatically extracting features from images and performing classification

Convolutional Neural Network (CNN) Architecture

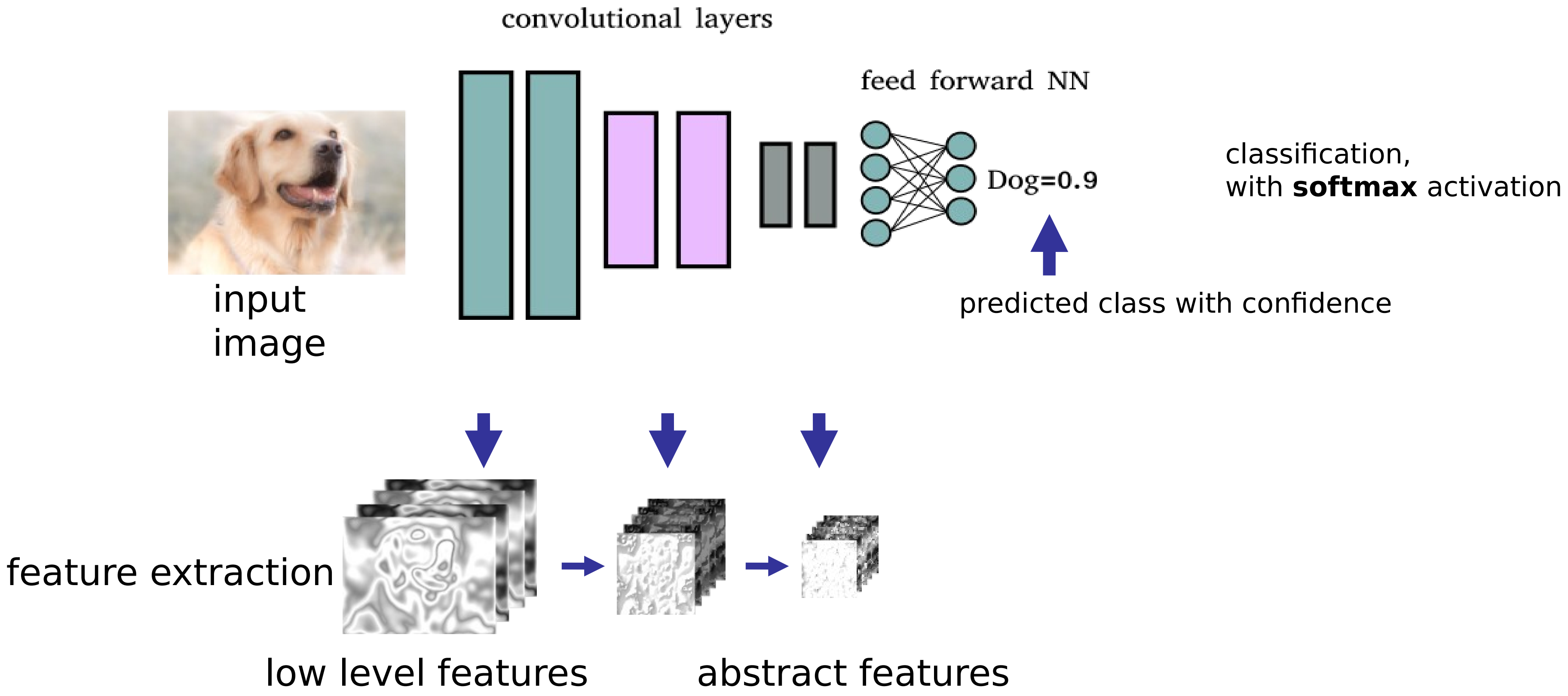


input image

classification,
with **softmax** activation

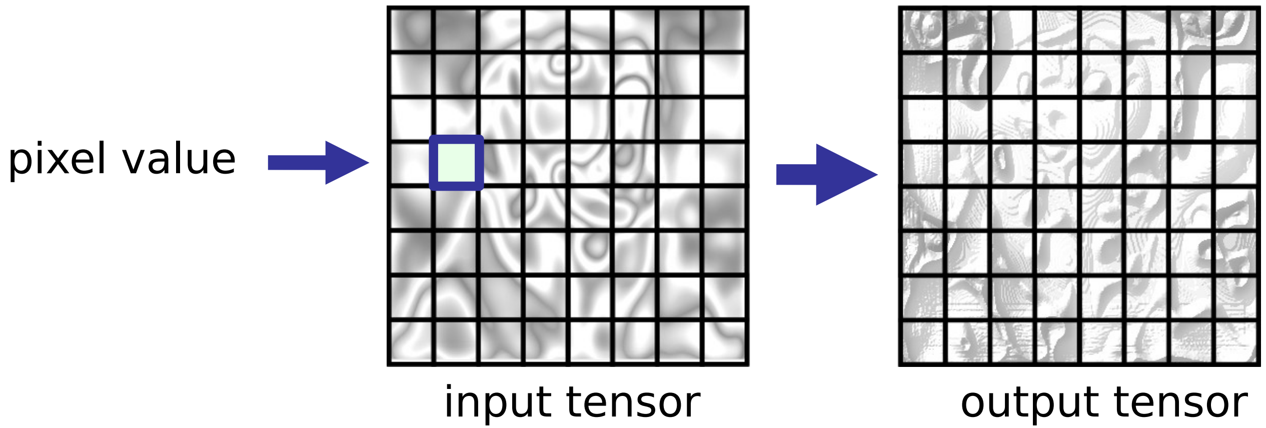
predicted class with confidence

Convolutional neural networks

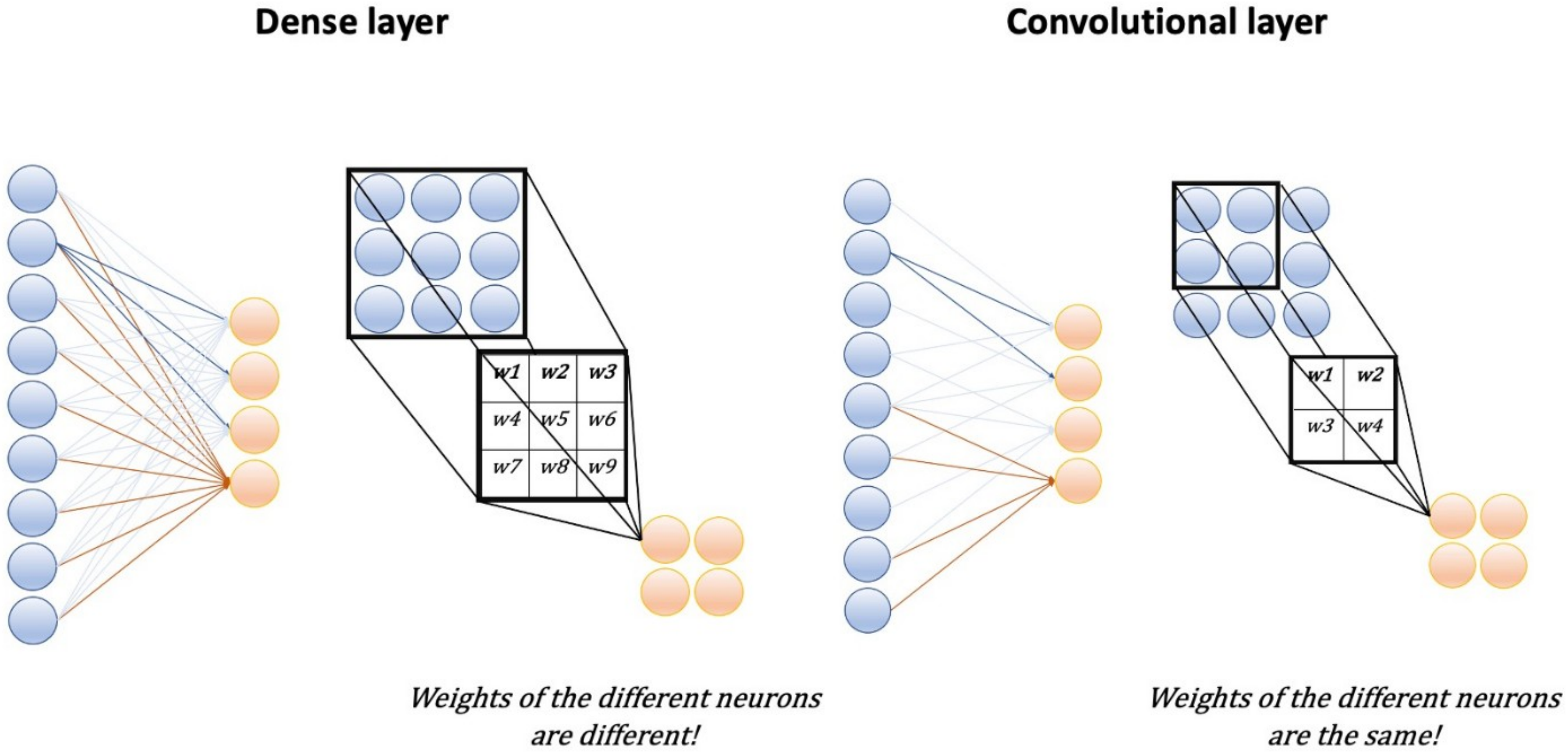


Convolutional layer

- processes data in form of tensors (multi-dimensional arrays)
- input: input image or intermediate features (tensor)
- output: a tensor with the extracted features

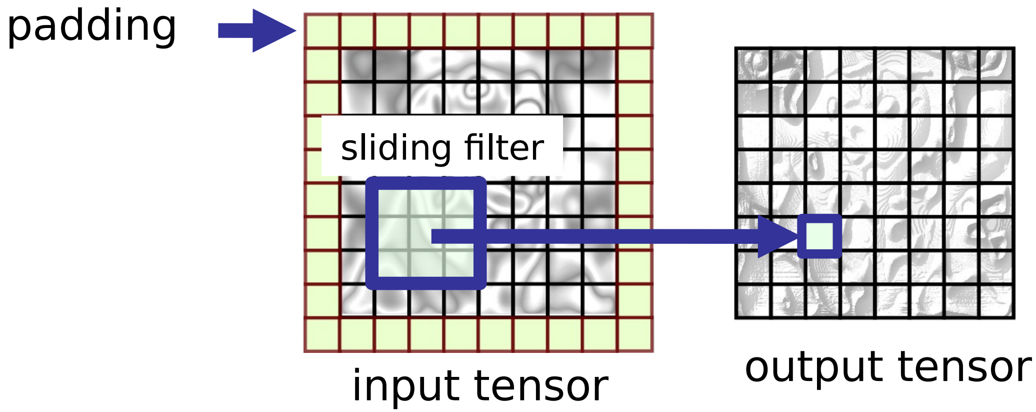


Convolutional layer



Convolutional layer

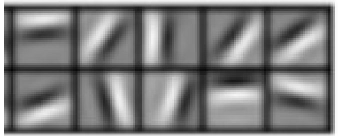
- a sliding filter produces the values of the output tensor
- sliding filters contain the trainable weights of the neural network
- each convolutional layer contains might contain many filters

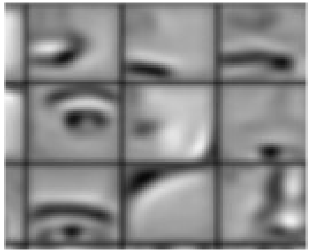


Convolutional neural networks

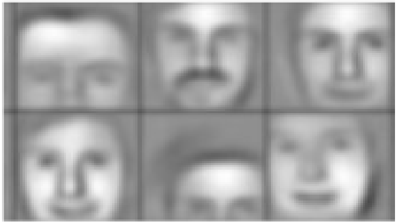
- Convolutional layers training
  - during training each sliding filter learns to recognize a particular pattern in the input tensor
  - filters in shallow layers recognize textures and edges
  - filters in deeper layers can recognize objects and parts (e.g. eye, ear or even faces)

shallow filters



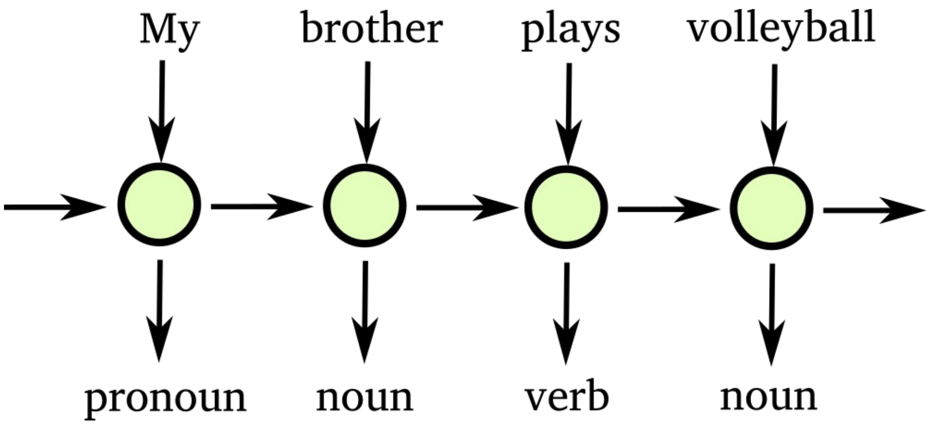


deeper filters



Recurrent Neural Networks

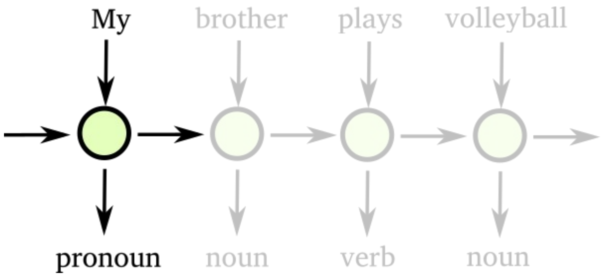
- Allow processing sequential data x(t)
- Differently from normal FFNN they are able to keep a state which evolves during time
- Applications
  - machine translation
  - time series prediction
  - speech recognition



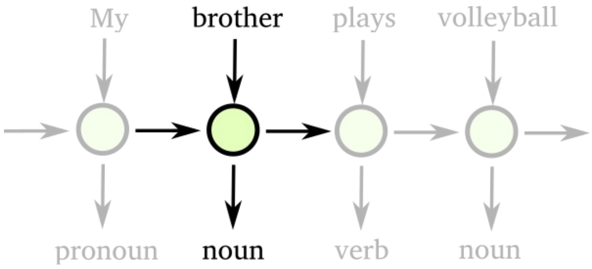
Recurrent Neural Networks

- RNN execution during time

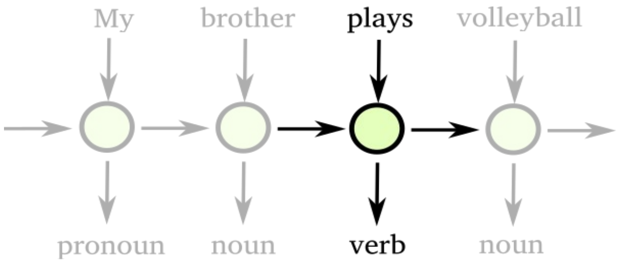
instance of the RNN at time t_1



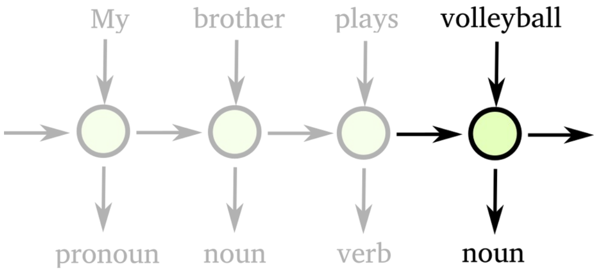
instance of the RNN at time t_2



instance of the RNN at time t_3

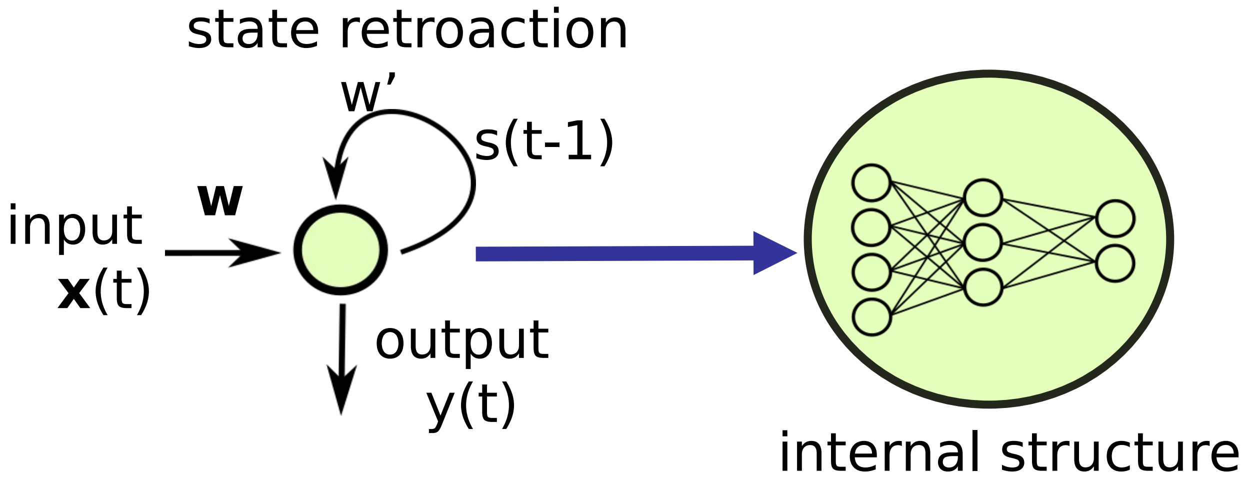


instance of the RNN at time t_4



Recurrent Neural Networks

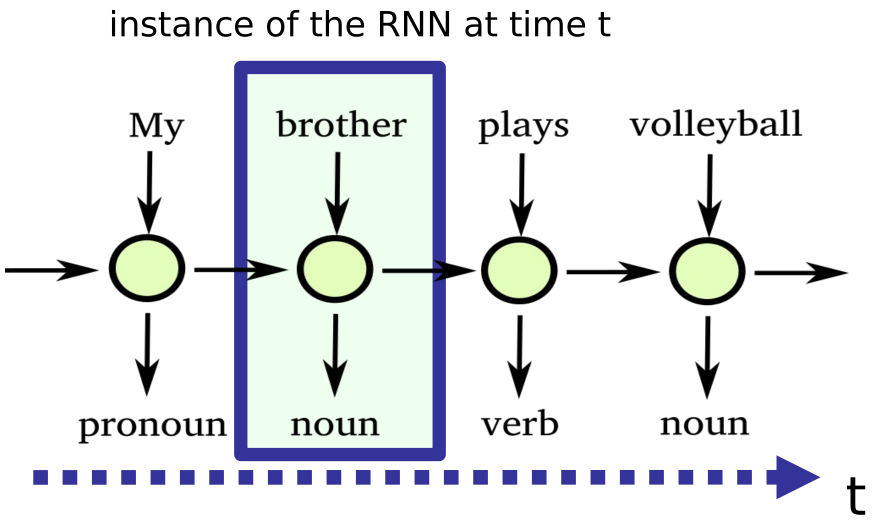
- A RNN receives as input a vector x(t) and the state at previous time step s(t-1)
- A RNN typically contains many neurons organized in different layers



Recurrent Neural Networks

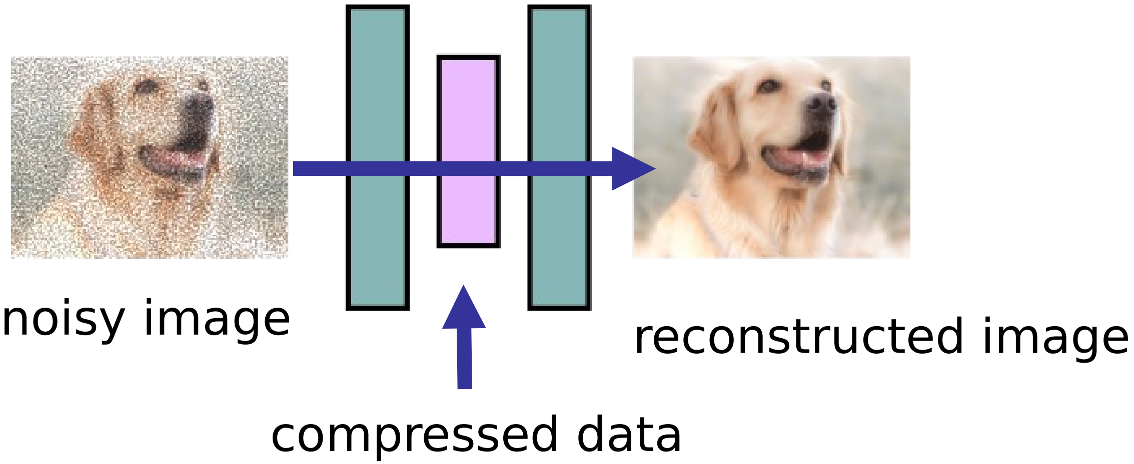
- Training is performed with backpropagation through time
- Given a pair training sequence x(t) and expected output y(t)
  - error is propagated through time
  - weights are updated to minimize the error across all the time steps

unrolled RNN diagram: shows the **same** neural network at different time steps



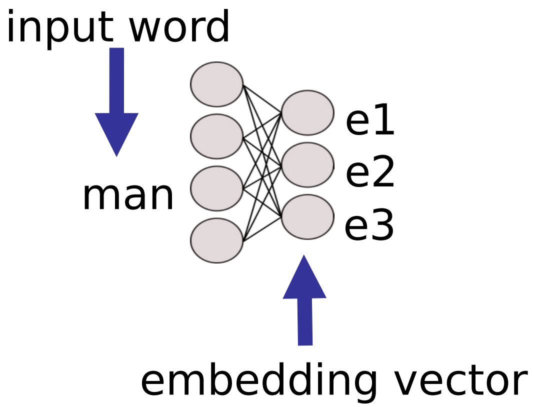
Autoencoders

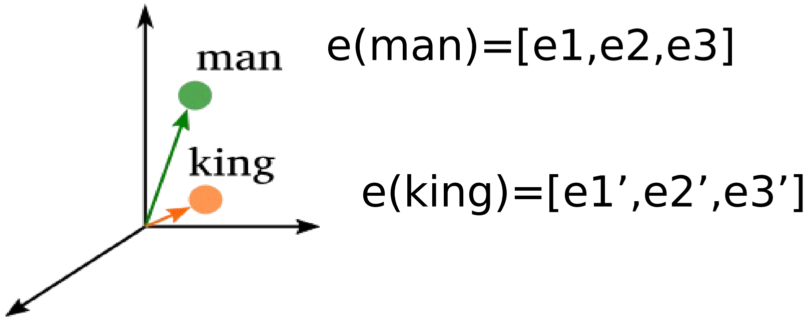
- Autoencoders allow compressing input data by means of compact representations (embeddings) and from them reconstructing the initial input
  - for feature extraction: the compressed representation can be used as significant set of features representing input data
  - for image (or signal) denoising: the image reconstructed from the abstract representation is denoised with respect to the original one



Word Embeddings

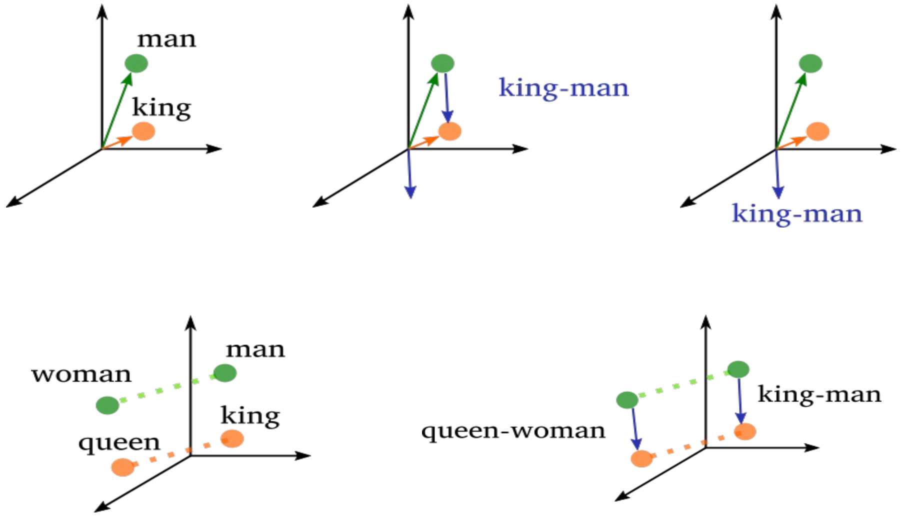
- Word embeddings (e.g., Word2Vec) associate words to n-dimensional vectors
- Trained on big text collections to model the word distributions in different sentences and contexts
- Able to capture the semantic information of each word
- Words with similar meaning share vectors with similar characteristics





Word Embeddings

- Since each word is represented with a vector, operations among words (e.g. difference, addition) are allowed
- Semantic relationiships among words are captured by vector positions



king - man = queen - woman
king - man + woman = queen

Deep learning is advancing quickly...

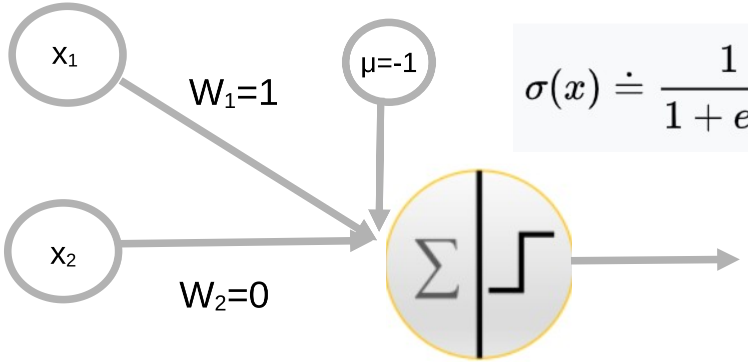
- Long Short Term Memories (LSTM)
- Generative Adversarial Networks (GAN)
- Transformers
- Language models (LM) and large language models (LLM)
- Graph neural networks (GNN)
- ...

Any questions?

Self-assessment quiz

- Define the model (and draw the schema) of two different NN with 3 features and 1 output, with at least a hidden layer each, and using at least two different activation functions.
- You are given the following hypothesis h of a NN for binary classification problem. The output of the NN is the probability of belonging to class A (i.e., 1 minus probability of belonging to class B). If we assign to class A output larger than 0.5, what is the accuracy on data D?

hypothesis h



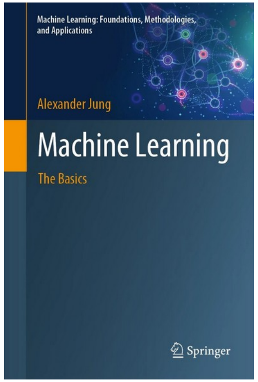
$$\sigma(x) \doteq \frac{1}{1+e^{-x}}$$

D

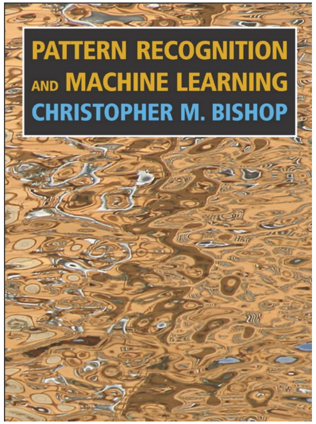
x_1	x_2	y
2	3	A
0	-4	B
5	5	A
3	7	A
-5	7	B

References: readings

- Chapters 3.11



- Chapter 5



https://scikit-learn.org/stable/modules/neural_networks_supervised.html

https://pytorch.org/docs/stable/index.html
https://pytorch.org/tutorials/
https://pytorch.org/tutorials/beginner/blitz/neural_networks_tutorial.html

Slide acknowledgments

- Alexander Jung – Aalto University
- Elena Baralis – Politecnico di Torino
- Dr. Christoph F. Eick - University of Houston
- Ethem Alpaydin - Bogaziçi University