Neural Networks 2 - Neural Networks

18NES2 - Lecture 2, Winter semester 2025/26

Zuzana Petříčková

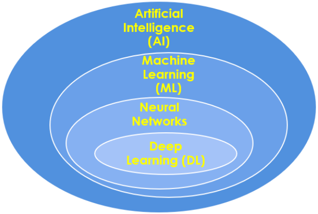
September 30, 2025

Neural Networks 2 - Neural Networks

What We Covered Last Time

- About the course Neural Networks 2
  - For details and credit requirements, see my website http://zuzka.petricek.net/vyuka_2025/NES2_2025/index.php

- Introduction to Artificial Intelligence and Machine Learning
- Fundamental Concepts of Machine Learning



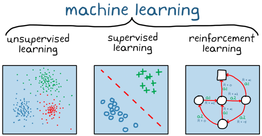
Review — Machine Learning

Core Principle:

- The system "builds itself," meaning it learns from data (training dataset) or previous experiences.

Learning Paradigms:

- **Supervised Learning**
  - Training dataset in the form of [*input, expected output*]
- **Unsupervised Learning (Self-Organization)**
  - Training dataset in the form of [*input*]
- **Reinforcement Learning**
  - The program learns an optimal strategy based on previous experiences.

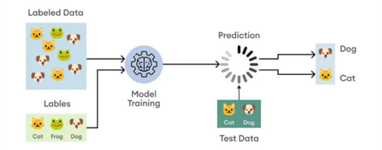


Source: https://www.mathworks.com/discovery/reinforcement-learning.html

Review — Machine Learning

Supervised Learning - Task Types

- **Classification:** Predicting a discrete class (category)



- **Regression:** Predicting a numerical value (e.g., price, temperature, handwriting slant, etc.)

- **Structured Data Learning** (e.g., natural language sentences, molecular structures, etc.)

Review — Machine Learning

Typical Machine Learning Workflow

- **Data Preprocessing**
  - Transforming raw data into a format suitable for machine learning models.
  - Example: Feature selection.
- **Model Selection and Development**
  - Choosing the right type of model (depends on the problem).
  - Selecting a specific model within the chosen type (optimizing parameters).
- **Model Evaluation** — Best performed on unseen (test) data.

Review — Machine Learning

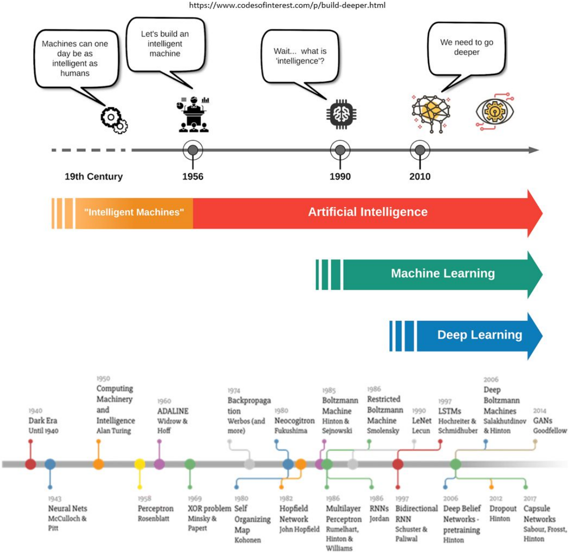
Typical Machine Learning Workflow

Practical Example: simple_example.ipynb

- A short illustration of a typical machine learning workflow
- Implementation of a simple **Multilayer Perceptron (MLP)** neural network using Keras

Neural Networks 2 - Introduction

Artificial Neural Networks — A Brief History



Mourtzis, Dimitris & Angelopoulos, John. (2020). An intelligent framework for modelling and simulation of artificial neural networks (ANNs) based on augmented reality. International Journal of Advanced Manufacturing Technology. 111. 10.1007/s00170-020-06192-y.

Artificial Neural Networks — A Brief History

Development Progressed in Waves:

- Periods of rapid advancement and high expectations were followed by phases of disappointment and stagnation.

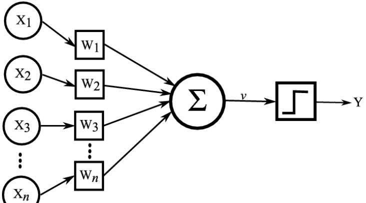
Key Time Periods:

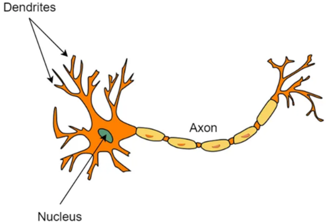
- **1940 – 1960:** Theoretical foundations
- **1960 – 1970:** First boom — the "single neuron era"
- **1970 – 1980:** First "AI Winter" for neural networks
- **1980 – 1990:** Second boom — the era of "shallow" neural networks
- **1990 – 2000:** Gradual stabilization of the field
- **2000 – 2010:** Second "AI Winter" for neural networks
- **2010 – present:** Third boom — the era of "deep" neural networks

Artificial Neural Networks — A Brief History

Early Foundations (1940-1960)

- Attempts to model the functioning of biological neurons.
- **1943: First mathematical model of a neuron** (W. McCulloch, W. Pitts) — capable of representing logical and arithmetic functions.





- **1949: Mathematical concept of learning** (D. Hebb) — introduced the first learning algorithm for artificial neurons, modeling conditioned reflexes.
- **1951: First neurocomputer, SNARC** (M. Minsky).

Artificial Neural Networks — A Brief History

First Boom (1960–1970): The "Single Neuron Era"

- **1957: Perceptron** (F. Rosenblatt) — first practical artificial neuron model with real-valued parameters and a functional learning algorithm. Sparked enormous enthusiasm.
- **1958: First successful neurocomputer,** *Mark I Perceptron* (F. Rosenblatt, C. Wightman).
- **1962: Adaline** and the **sigmoid activation function** (B. Widrow, M. Hoff).
- Rapid progress in neurocomputing in the 1960s, but soon faced fundamental limitations.

Artificial Neural Networks — A Brief History

- **1969: Perceptrons** (M. Minsky, S. Papert) — demonstrated the limitations of perceptrons, showing they cannot model even basic logical functions.
- What if multiple neurons were connected into a network?
  $\rightarrow$ A promising idea, but how would such a model be trained? Existing learning algorithms for perceptrons and linear neurons were insufficient.

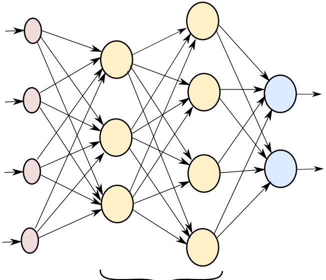
1970 – 1980: First "AI Winter"

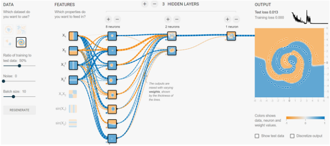
- Disillusionment due to unfulfilled expectations.
- Declining credibility of the field.
- Decreased funding and reduced interest in AI research.

Artificial Neural Networks — A Brief History

Second Boom (1980 - 1990): The Era of "Shallow" Neural Networks

- Renewed interest in the field (John Hopfield, DARPA, etc.); introduction of the multilayer perceptron (MLP).
- **1986: Backpropagation Algorithm** for training MLPs (P. Werbos and D. Rumelhart, with earlier work by G. Hinton and Y. LeCun).
  - A fundamental concept that remains in use today.





https://playground.tensorflow.org/

Artificial Neural Networks — A Brief History

Second Boom (1980 - 1990): The Era of "Shallow" Neural Networks

Emergence of new neural network models composed of multiple neurons:

- **Multilayer Perceptron (MLP)**
- **Kohonen Maps** (T. Kohonen)
- **Hopfield Networks** (J. Hopfield)
- **Radial Basis Function (RBF) Networks** (J. Moody, C. Darken)
- **Growing Neural Gas (GNG) Model** (B. Fritzke)
- **Support Vector Machines (SVMs)** (V. Vapnik)
- **Extreme Learning Machines (ELMs)** (G.-B. Huang)
- **Recurrent Neural Networks (RNNs)** (e.g., Jeffrey Elman)
- **Convolutional Neural Networks (CNNs)** (Y. LeCun, Y. Bengio, et al.)

Artificial Neural Networks — A Brief History

Stabilization Period (1990-2000)

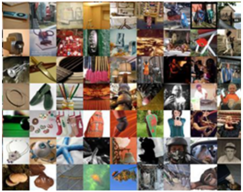
- dominant models: MLP (Multilayer Perceptron), RNN (Recurrent Neural Network), Kohonen Maps, and SVM
- Efforts to address learning challenges in neural networks:
  - Application of advanced optimization techniques.
  - Enhancing robustness, generalization, and mitigating overfitting
  - Learning strategies — parallelization and computational efficiency.
- Attempts to train deeper neural networks:
  - Beyond MLP, CNNs (Convolutional Neural Networks) were also developed.
  - However, computational limitations (processing power and memory) imposed significant barriers.
  - The vanishing and exploding gradient problem emerged as a major challenge.

2000 – 2010: The Second "AI Winter"

Artificial Neural Networks — A Brief History

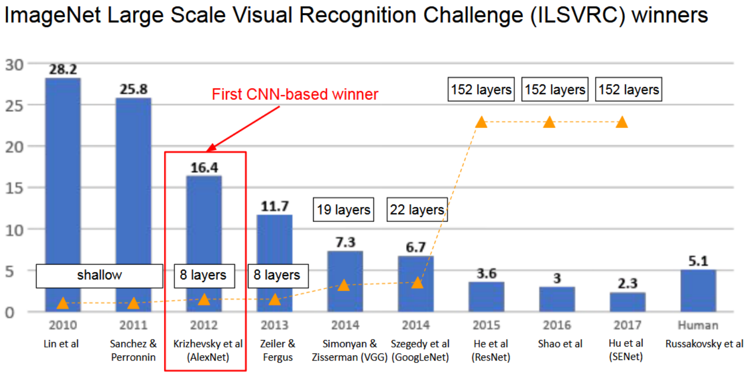
2010 – Present: The Third Boom — The Era of Deep Neural Networks

- **What triggered the boom?**
  - Advances in GPUs, cloud computing, and easy access to large datasets.
  - Improvements in learning algorithms (e.g., Deep Learning, Y. LeCun, Y. Bengio, G. Hinton, 2015).
  - **Breakthrough: CNNs won the ILSVRC competition (2012)**
    - **ImageNet Dataset:** 16 million color images across 20,000 categories.



Artificial Neural Networks — A Brief History

2010 – Present: The Third Boom — The Era of Deep Neural Networks



https://cs231n.stanford.edu/2021/slides/2021/lecture_9.pdf (Fei-Fei Li, Ranjay Krishna, Danfei Xu)

Artificial Neural Networks — A Brief History

2010 – Present: The Third Boom — The Era of Deep Neural Networks

- Rapid emergence of new deep learning models and architectures:
  - **Modern Recurrent Neural Networks (RNNs):** LSTM (Hochreiter, Schmidhuber, 1997).
  - **Generative Adversarial Networks (GANs)** (I. Goodfellow, 2015).
  - **First transformer-based generative language model:** GPT-3 (2020, OpenAI), later followed by ChatGPT (2022).
  - **First text-to-image model:** Stable Diffusion (2022, CompVis).

Modern Deep Neural Network Architectures

Multilayer Perceptron (MLP, sometimes DNN)

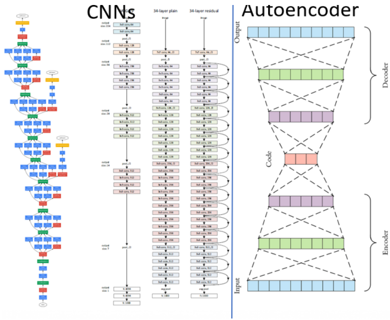
- A standard, universal model.

Convolutional Neural Networks (CNNs)

- Image and video processing.
- Classification, recognition, and segmentation tasks.

Autoencoders

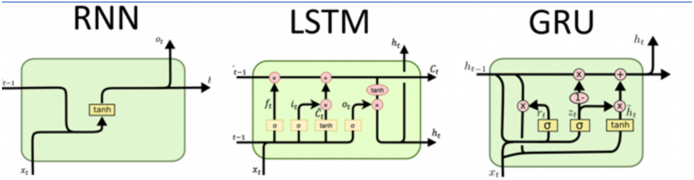
- Unsupervised learning.
- Feature extraction, data denoising, compression, and reconstruction.



Modern Deep Neural Network Architectures

Recurrent Neural Networks (RNNs)

- Used for sequential data analysis (time series, speech, text, handwriting).
  - **Long Short-Term Memory Networks (LSTMs)**
    - Effective for processing sequential data (e.g., speech and handwriting recognition).
    - Captures long-range dependencies.
  - **Gated Recurrent Unit Networks (GRUs)**
    - Efficient for modeling sequential data.
    - Applied in speech recognition and machine translation.



Modern Deep Neural Network Architectures

- **Generative Adversarial Networks (GANs)** — Generate new data based on learned patterns.
- **Deep Belief Networks (DBNs)** — Unsupervised generative models.
- **Deep Q-Networks (DQNs)** — Used in reinforcement learning applications.
- **Siamese Networks** — Applied in image recognition and object tracking; measures similarity between two inputs.
- **Capsule Networks (CapsNets)** — Used for image recognition; models hierarchical relationships between object parts.
- **Transformer Networks (BERT, GPT)** — Used for natural language processing (text classification, translation, etc.).

Neural Networks in the Course Neural Networks 2

- This course is build on the previous one

What we covered in the course Neural Networks 1, 18NES1, Sommer semester 2024/25

- We covered basic "shallow" neural network models: the perceptron and single-layer models, Kohonen maps,...
- We also introduced basic deep models (MLP, convolutional neural networks)
- We tried to look inside neural network models and understand how they learn and work
- We also touched on theory (to a limited extent)
- We got familiar with Python libraries for deep learning (mainly Keras)

What we will cover in the course Neural Networks 2

Course Neural Networks 2, 18NES2/18YNES2, Winter Semester 2025/26

- We will briefly revisit some models already covered in Neural Networks 1 (MLP, convolutional neural networks)
- We will introduce new ones (advanced CNN variants, modern recurrent neural networks, autoencoders, generative models)
- Most likely, we will not reach more complex models (GANs, transformers, reinforcement learning, etc.)
- The emphasis will be on a practical, engineering-oriented perspective
- Theory will be covered only at a very basic level

What we will cover in the course Neural Networks 2

Course Neural Networks 2, 18NES2/18YNES2, Winter Semester 2025/26

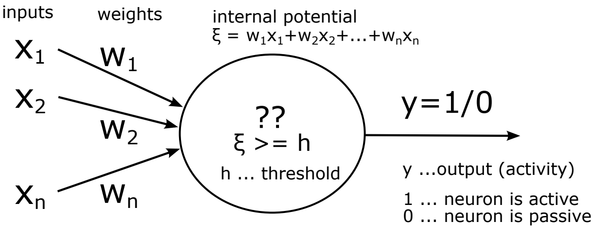
- We will explore real-world applications through Python examples
- We will address different types of tasks and how to solve them with deep neural networks
- We will experiment and try things out :)

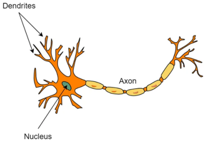
Would you like to learn how to use or understand even more modern and complex neural network models?

- You can learn more about deep neural networks in follow-up courses:
  - **Machine Learning 2**
  - **Applications of Optimization Methods, and others**

Neural Networks 2 - Introduction

Artificial Neuron - Inspired by Biological Neurons





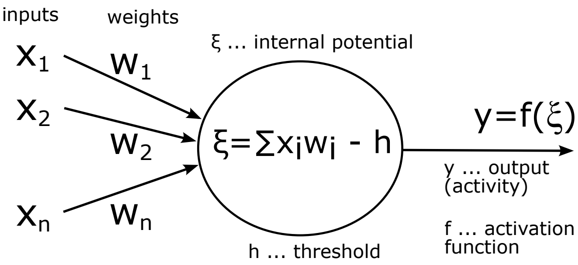
The First Artificial Neuron Model (McCulloch–Pitts, 1943):

- Inputs $x_1, x_2, \ldots, x_n \in \{0, 1\}$
- Each input has a weight $w_1, w_2, \ldots, w_n$.
- Internal potential: weighted sum:

$$\xi = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$$

- Threshold h: output (activity) is 1 if $\xi \geq h$, otherwise 0.

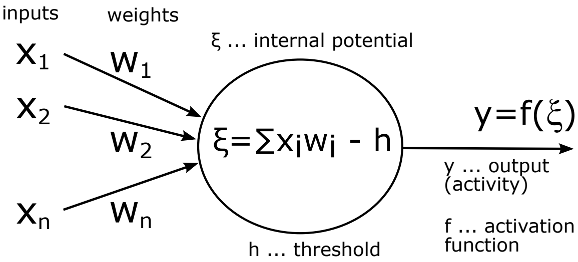
An Artificial Neuron - More general model



- **Neuron Parameters:**
  - Weight vector: $\vec{w} = (w_1, \ldots, w_n) \in \mathbb{R}^n$.
  - Threshold (bias): $h \in \mathbb{R}$.
  - Activation function: $f : \mathbb{R} \to \mathbb{R}$.
- Given an input $\vec{x} \in \mathbb{R}^n$, the neuron computes an output $y \in \mathbb{R}$ as:

$$y = f_{\vec{w},h}(\vec{x})$$

An Artificial Neuron — Original Definition



Neuron Output Calculation:

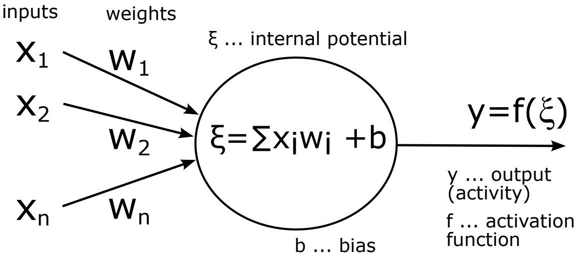
- Internal potential:

$$\xi = \sum_{i=1}^{n} w_i x_i - h = \vec{w} \cdot \vec{x}^T - h$$

- Output:

$$y = f(\xi)$$

An Artificial Neuron — Modern Definition



Alternative Definition: Threshold h → Bias b

- Internal potential:

$$\xi = \sum_{i=1}^{n} w_i x_i + b = \vec{w} \cdot \vec{x}^T + b$$

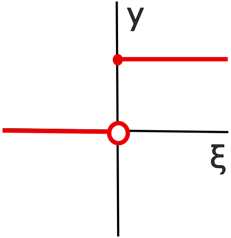
- Output: $y = f(\xi)$

An Artificial Neuron - About the Activation Function

- The historical **Perceptron** model utilized a **step activation function**.

Step Activation Function:

- If $\sum_{i=1}^{n} w_i x_i \geq h$, i.e.,
  $\xi = \sum_{i=1}^{n} w_i x_i - h \geq 0$
  then the neuron is **active** ($f(\xi) = 1$).
- If $\sum_{i=1}^{n} w_i x_i < h$, i.e.,
  $\xi = \sum_{i=1}^{n} w_i x_i - h < 0$
  then the neuron is **passive** ($f(\xi) = 0$).



$\rightarrow$ The perceptron can be used as a **linear classifier**, separating patterns into two classes

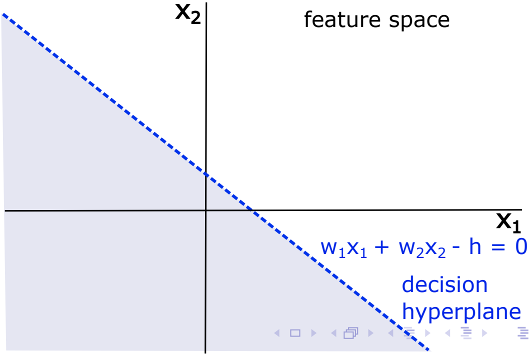
Perceptron as a linear classificator

Geometric Interpretation of an Artificial Neuron

- The neuron's inputs can be represented as points in an n-dimensional Euclidean space (input/feature space).
- Setting the neuron's internal potential to $\xi = 0$ results in the equation of a **decision hyperplane (decision boundary)** .

$$\xi = w_1 x_1 + w_2 x_2 - h = 0$$

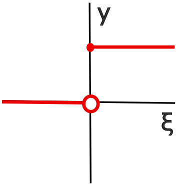
$$x_2 = -\frac{w_1}{w_2} x_1 + \frac{h}{w_2}$$



And what about other activation functions?

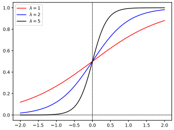
Step Activation Function

- simple and intuitive (close to the biological model)
- but not differentiable $\rightarrow$ cannot be effectively trained with gradient-based methods
- therefore rarely used in larger neural networks

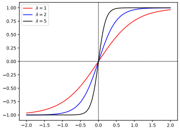


$\rightarrow$ **Continuos and smooth replacements:**

Sigmoid



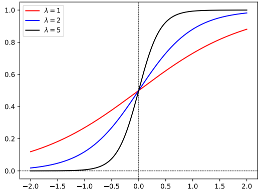
Hyperbolic Tangent



Continuous alernatives to the step function

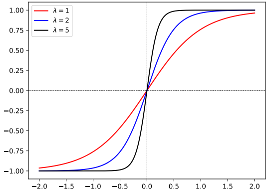
Logistic Sigmoid

- $f(\xi) = \frac{1}{1+e^{-\lambda\xi}}$ ... logsig
- maps input to $(0, 1)$ (probabilistic interpretation)



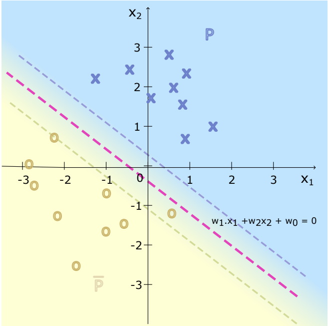
Hyperbolic Tangent

- $f(\xi) = \frac{1-e^{-2\lambda\xi}}{1+e^{-2\lambda\xi}}$ ... tanh
- similar to sigmoid, but symmetric around zero
- outputs in $(-1, 1)$



Sigmoid and Hyperbolic Tangent

Geometric Interpretation:



... Linear classification task

- With a sigmoidal activation function, the neuron's output can be directly interpreted as the **probability** that the input belongs to a given class.

Usage of Sigmoid and Hyperbolic Tangent

Sigmoid function

- historically used in hidden layers (smooth replacement of step function), but suffers from vanishing gradients
- nowadays mainly in the **output layer** for binary classification
- outputs can be interpreted as probabilities

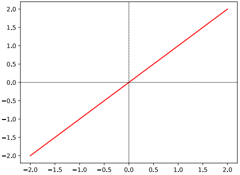
Hyperbolic tangent

- often used in hidden layers in older architectures, but it also suffers from vanishing gradients
- centered around zero $\rightarrow$ better gradient flow than sigmoid
- still used in some RNN architectures

And what about other activation functions?

Identity function

- $f(\xi) = \xi$ ... **linear neuron**
- output: $y = \xi = \sum_{i=1}^{n} w_i x_i + b$
- $\rightarrow$ during training we search for $\vec{w}$ such that: $\vec{d} = \vec{y}$, i.e. $\vec{d} = X\vec{w}$
- this corresponds to the problem of **linear regression**



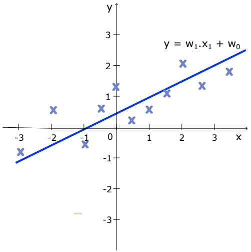
Usage:

- very suitable in the output layer for regression tasks
- not very useful for classification tasks and in hidden layers

Linear Neuron - Geometric Interpretation

For a single input feature:

- Neuron output: $y = w_1 x + b$
- (x_k, d_k) are points in the plane
- We fit the points with a straight line:



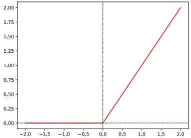
In general: We fit the points with a hyperplane
$y = w_1 x_1 + \cdots + w_n x_n + b$

- assuming a linear relationship between input variables $x_1, \ldots, x_n$ and the output y

And what about other activation functions?

Rectified Linear Unit (ReLU)

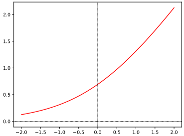
- $f(\xi) = \max(0, \xi) = \begin{cases} \xi, & \text{for } \xi > 0 \\ 0, & \text{for } \xi \leq 0 \end{cases}$



  - Not differentiable everywhere, but computationally efficient
  - Widely used in hidden layers of deep neural networks

Softplus (Smooth Alternative to ReLU)

- $f(\xi) = \ln(1 + e^{\xi})$
- Behaves like ReLU for large ξ
- Behaves like a sigmoid function for small ξ
- Less commonly used than ReLU

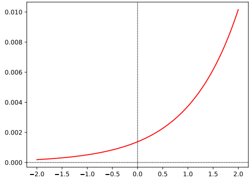


And what about other activation functions?

Softmax

- A specialized activation function for multi-class classification
- A generalization of the **argmax** function, converting numerical values into probabilities
- $f : R^n \to R^n$,

$$f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}$$



→ Suitable only for the output layer in classification tasks

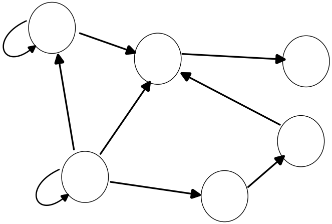
Neural Networks 2 - Week 2

Neural Network

- A neural network consists of neurons that are interconnected by edges.
- The output of one neuron can serve as an input to one or more other neurons.

Neural Network Architecture

- Represented as a directed graph, where neurons correspond to nodes and connections between neurons correspond to edges.



Neural Network

Output Neurons

- Their outputs together form the final output of the neural network.

Input Neurons

- Their inputs are the input patterns (examples).

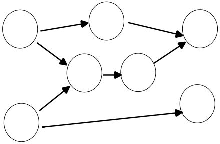
Network Output (Response)

- Defined by the activities (outputs) of the output neurons.

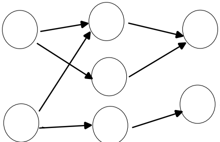
Neural Network

Neural Network Architecture (Topology)

- **Cyclic / recurrent** networks: allow feedback connections.
- **Acyclic / feedforward** networks: all connections go in the same direction (i.e., the graph can be topologically ordered).



- **Hierarchical / layered** networks: special case of feedforward, organized into layers, with connections only between neurons in consecutive layers.



Multi-Layer Neural Network (Multi-Layer Perceptron, MLP, 1980)

- Hierarchical **sequential** architecture: neurons are arranged in layers
- **Dense (fully connected) layers**: every neuron in one layer is connected to every neuron in the next layer

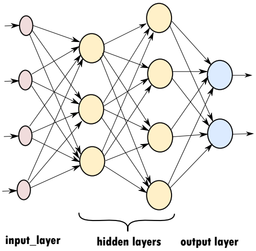
Special input layer:

- corresponds to the inputs of the neural network

Output layer:

- the output (response) of the network corresponds to the activities of the output neurons

The remaining layers are **hidden layers**.



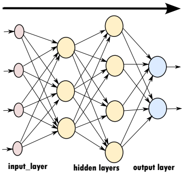
Multi-Layer Neural Network (Multi-Layer Perceptron)

How to represent the sequential NN model?

- A sequence (list) of layers $L_0, \ldots, L_{max}$ (see the Keras example)
- Each layer is represented by a tensor of weights (and biases)

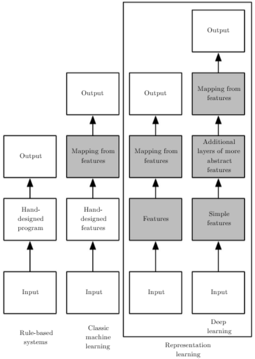
How to compute the model output (response):

- by performing a **forward pass**
- we process one layer at a time, starting from the input layer and going toward the output:
  - present the input tensor to the current layer
  - compute the layer's output
  - use this output as the input to the next layer



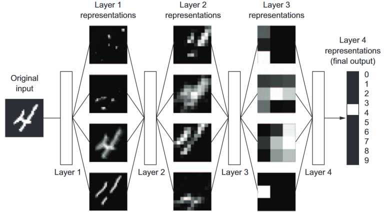
Neural Network Architecture (Topology)

- **Shallow model** – one hidden layer
- **Deep model** – more (or many) hidden layers
    - Automatically extracts features from data, reducing the need for extensive preprocessing.



I. Goodfellow, Y. Bengio, and A. Courville: Deep Learning, 2016, Figure 1.5

Convolutional neural network

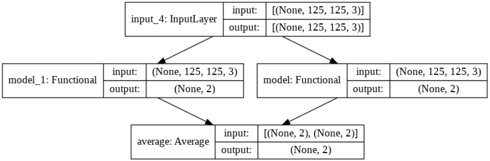


F. Chollet: Deep Learning with Python, Fig. 1.6

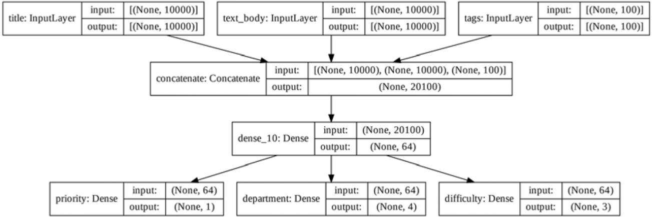
Neural Network Architecture (Topology)

- For modern deep models, a simple sequential (layered) architecture is often not sufficient, for example:

Ensemble model

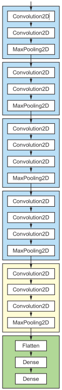


Model with multiple inputs and outputs

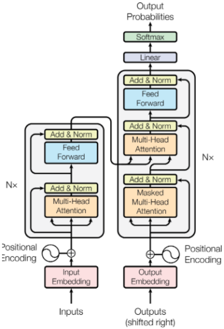


Neural Network Architecture (Topology)

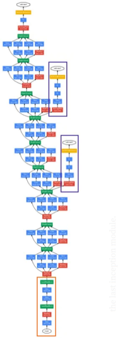
- Modern deep networks often have complex topologies:



The VGG-16 CNN model (F. Chollet: Deep Learning with Python, Fig. 8.15)



The original Transformer architecture (Vaswani et al., 2017)



Inception V1 (Szegedy et al., 2014)

Training a Neural Network (Supervised Learning)

Data used for training the model

- **Training set** T
  - a set of N training samples
    $T = (X, D) = \{(x_1, d_1), ..., (x_N, d_N)\}$
  - X ... input data (tensor), D ... desired output (tensor)
- **Training sample (pattern)** (x_i, d_i)
  - x_i ... input pattern (tensor)
  - d_i ... target / expected / desired output
- Examples of input data tensors:
  - vector data — 2D tensor (samples, features)
  - time series and sequential data — 3D tensor (samples, timesteps, features)
  - images — 4D tensor (samples, height, width, channels)
  - video — 5D tensor (samples, frames, height, width, channels)

Training a Neural Network (Supervised Learning)

The number and shape of output features depend on the task:

- **Regression:**
  output tensor shape (samples, 1) or (samples, output features)
  e.g. prediction of values (stock price, temperature,...)
- **Classification:**
  output tensor shape (samples, number of classes)
  e.g. binary classification $\rightarrow$ shape (samples, 1)
  multiclass classification $\rightarrow$ shape (samples, number of classes)
- **Sequential data:**
  output tensor shape (samples, timesteps, output features)
  e.g. sentence translation $\rightarrow$ shape (samples, output sequence length, vocabulary size)
- images, video, ...

Training a Layered Neural Network (MLP Model)

- Let us now focus on the classical Multi-Layer Neural Network model with n inputs and m output neurons:

Training data

- Training set $T = (X, D)$
  - X ... input patterns: 2D tensor of shape (N, n), where N is the number of training samples, n is the number of input features
  - D ... desired outputs: 2D tensor of shape (N, m), where m is the number of output features
- Training sample (pattern) $(\vec{x}_i, \vec{d}_i)$
  - $\vec{x}_i$ ... vector of length n (input pattern)
  - $\vec{d}_i$ ... vector of length m (target / expected output)

Learning objective:

- Adjust the weights (and biases) of all neurons in the network so that the actual output Y matches the desired output D.

Training a Layered Neural Network (MLP Model)

Basic principle (simplified)

1. Randomly initialize the model parameters (weights and biases of all neurons).
2. Repeat the training cycle:
   - prepare a batch of training inputs X and corresponding targets D
   - compute the actual output (prediction) of the model Y
   - calculate the model error (difference between Y and D)
   - update weights and biases to make the error slightly smaller

$\rightarrow$ **gradient descent method** (or its variants)

- the need of continuous and differenciable activation functions