Neural Networks
Basic principles

A neural network is a universal model that solves a wide class of regression and classification problems

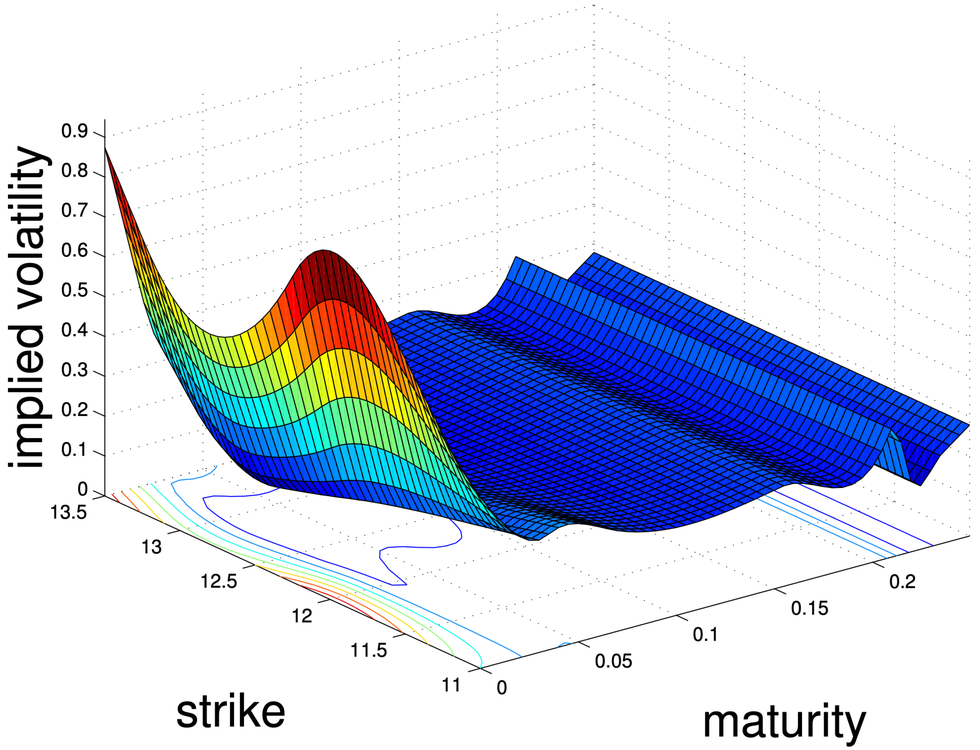
Mariia Mitrankova
December 6th , 2024

MLLM, CFNS

Neural network and regression problem

$(x_i, y_i)_{i=1}^{l}$ - training sample in a regression problem

- $x_i \in \mathbb{R}^d$ - feature description of the i^{th} sample object
- $y_i \in \mathbb{R}^1$ - the value of the target dependent variable at the i^{th} sample object
- We want to build surface $a(x)$ approximating the unknown target dependence



Neural network and regression problem

Examples

Energy Calibration of Detectors

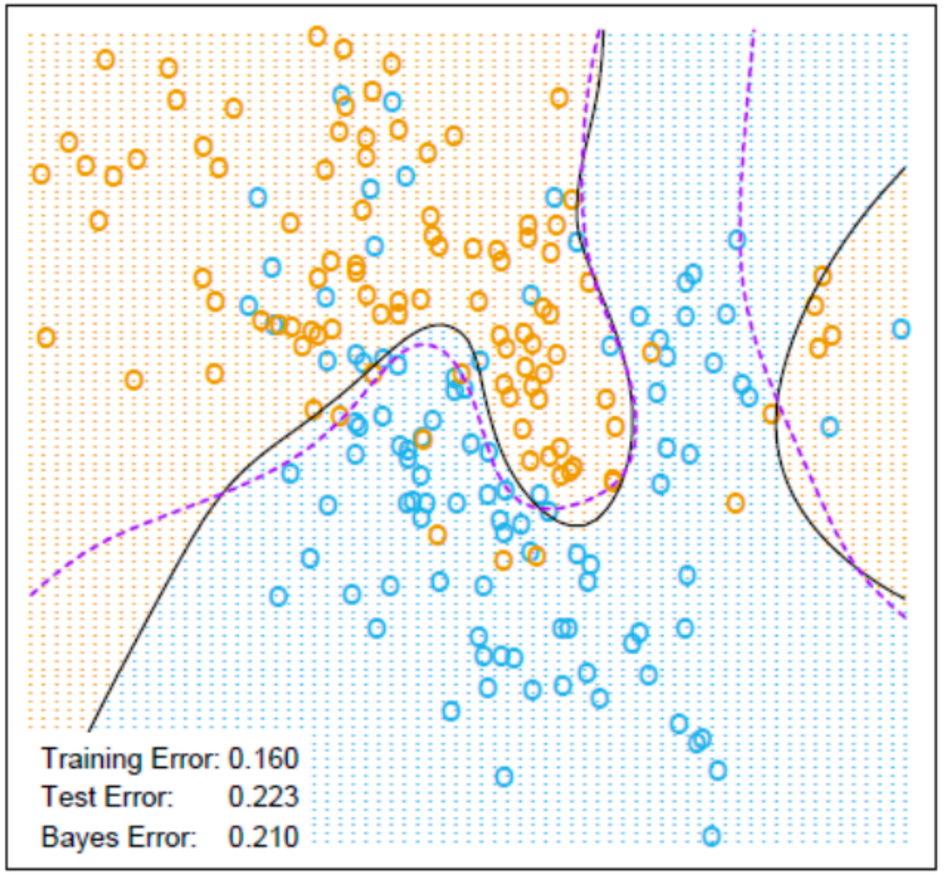
x_j - simulated or calibration data

y - mapping between measured signals and the true particle energy

Momentum Reconstruction

x_j - curvature of the track and detector characteristics

y - particle's momentum



Neural network and classification problem

$(x_i, y_i)_{i=0}^{l}$ - training sample in a binary classification problem

- $x_i \in \mathbb{R}^d$ - feature description of the i^{th} sample object
- $y_i \in \{-1; 1\}$ - the value of the target dependent variable at the i^{th} sample object
- We want to build surface $a(x)$ dividing objects of one class from the other

Neural network and classification problem

Examples

Particle Identification

x_j - particle characteristics

y - if it is particle you are looking for

Background Rejection

x_j - Event topology, energy distributions, and particle multiplicities

y - Separate rare signal events from common background processes

Neural network as universal model

- The neural network is considered a universal model, since it is capable of approximating any surfaces
- Theorem formulated in 1957 by Andrey Nikolaevich Kolmogorov

Any continuous function $a(x)$ on the d-dimentional unit cube is representable as

$$a(x) = \sum_{i=1}^{2d+1} \sigma_i \left(\sum_{j=1}^{d} f_{ij}(x_j) \right)$$

Where $x = [x_1, \ldots x_d]^T$ - vector of object description, functions σ_i and f_{ij} are continuous functions, f_{ij} does not depend on the a

Unit cube - we can rescale

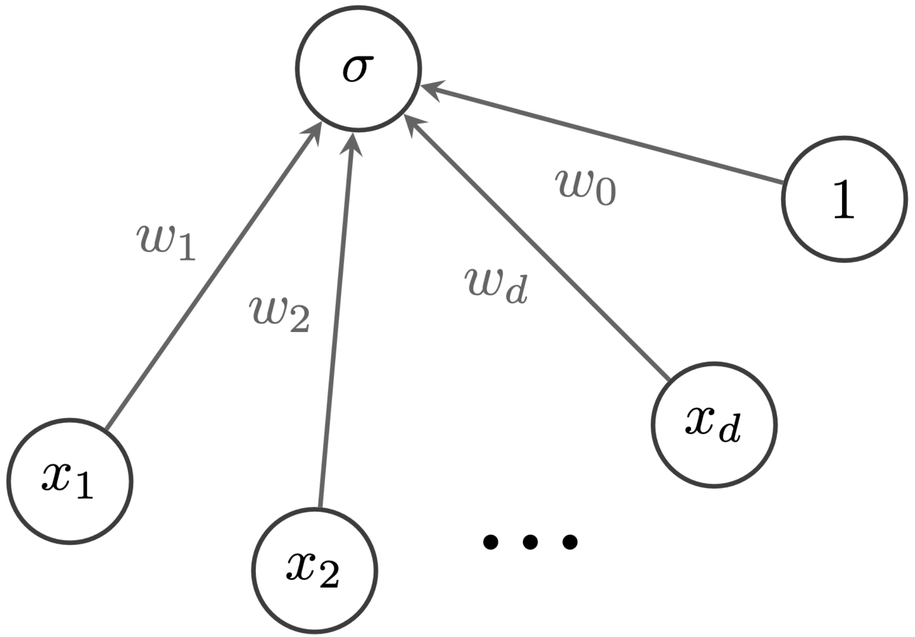
Nothing about how σ_i and f_{ij} look like

Single layer neural networks

Single layer neural network as a single neuron

$$a(x, \omega) = \sigma(\omega^T x) = \sigma\left(\sum_{j=1}^{d} \omega_j^{(1)} x_j + \omega_0^{(1)}\right)$$

- σ - activation function (must be a continuous, monotonic and, preferably, differentiable function)
- ω - vector of parameters (weights)
- x - vector of object description parameters

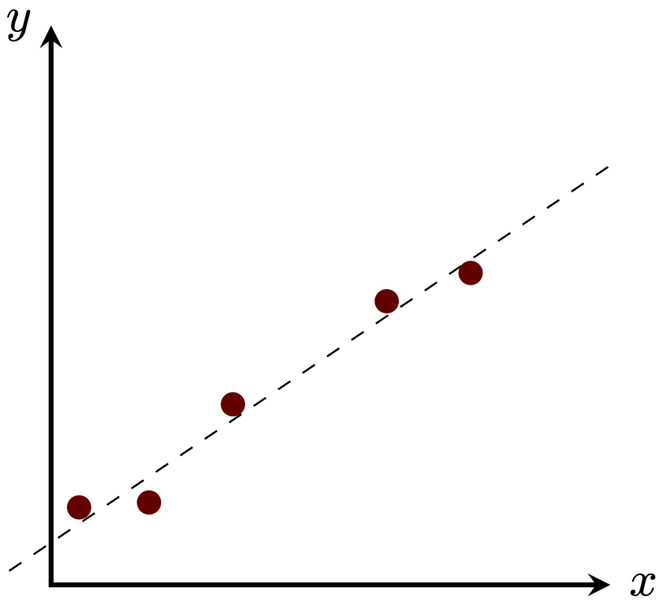


A neuron can be represented as a vertex of a graph: activation function, one output, many inputs

Activation Functions: Neural Network as a Linear Model

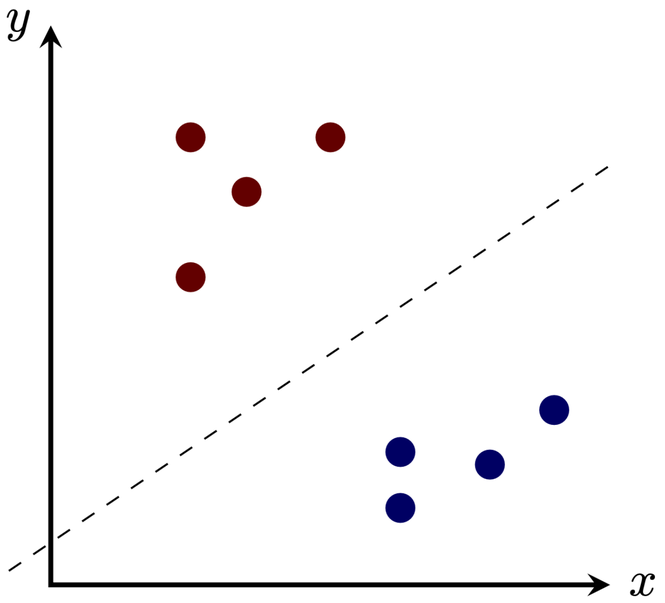
Linear regression

- $\sigma = id$
- $a(x, \omega) = \omega^T x$



Linear classification

- $\sigma = sign$
- $a(x, \omega) = sign(\omega^T x)$



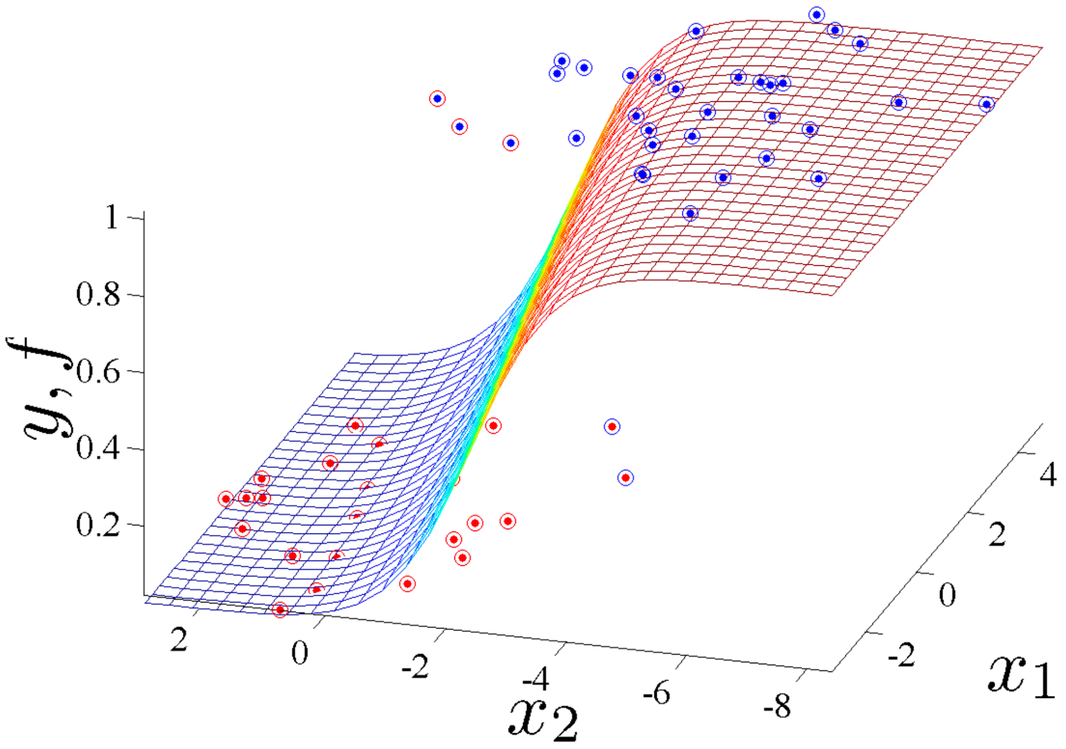
Activation functions: logistic regression model

- Activation function is following:

$$a(x;\omega) = \sigma(\omega^T x) = \frac{1}{1 + exp(-\omega^T x)}$$

- This function determines the **probability** of belonging of some given object x to class $y = 1$ for given parameters w

$$\sigma(\omega^T x) = P(y = 1 \,|\, w, x)$$



Activation functions: logistic regression model

If the number of classes is K:

it is necessary to use a network of K neurons, each of which calculates the probability of belonging

$$y = [y^1, \ldots, y^K]^T$$

to its class

Softmax is used as the activation function:

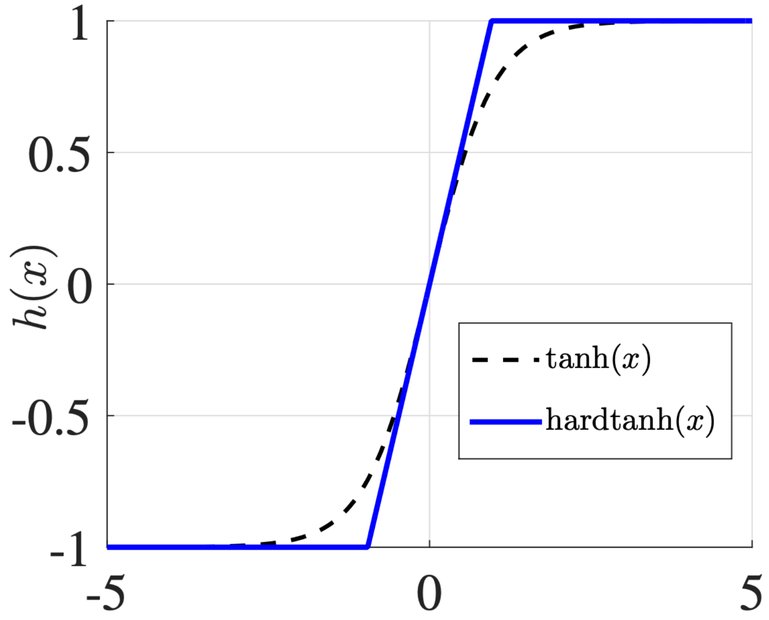
$$\sigma = softmax(\omega^T x, \ldots, \omega_K^T x) = \frac{exp(\omega_k^T x)}{\sum_{i=1}^{K} exp(\omega_i^T x)}$$

Activation functions: other examples

to introduce non-linearity

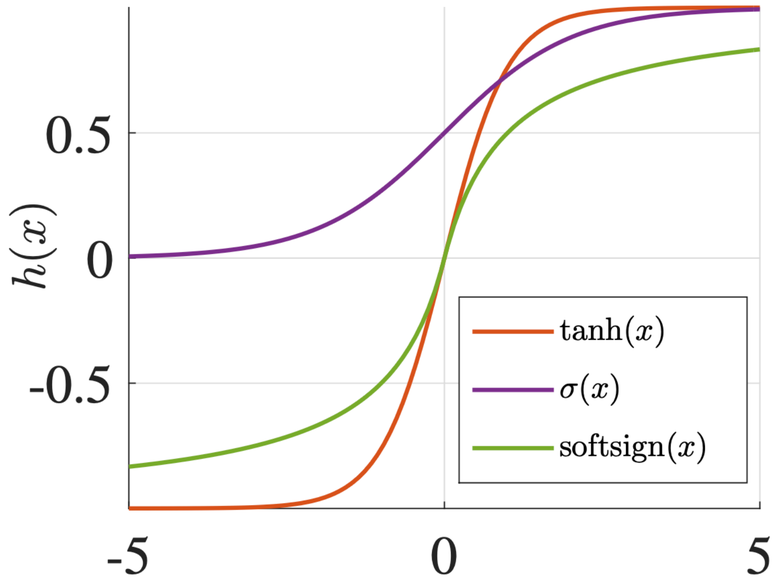
$$tanh(x) = \frac{exp(2x) - 1}{exp(2x) + 1}$$

- differentiable alternative to the threshold function



$$softsign(x) = \frac{x}{1 + |x|}$$

- like tanh(x) it tends to ± 1, but unlike it, it converges to these values not so quickly



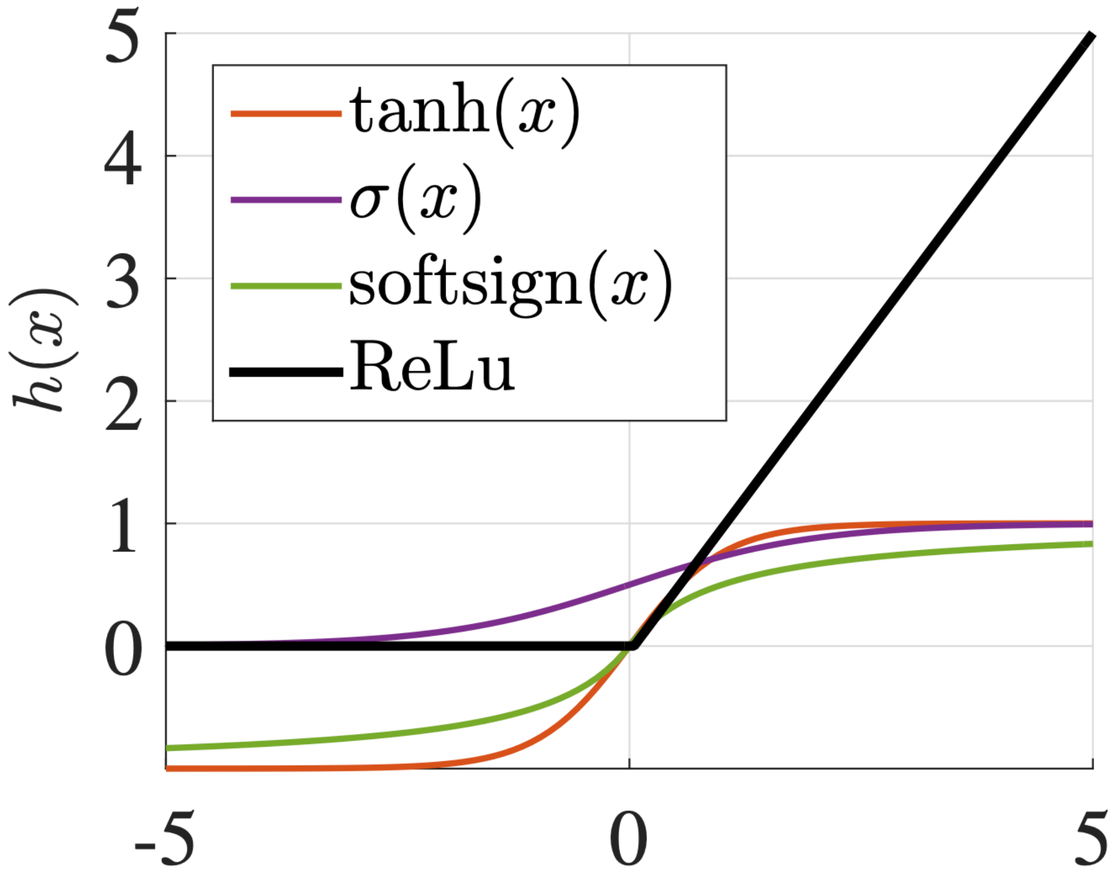
Activation functions: other examples

- In deep learning networks, more complex neural networks use the Rectified linear unit (**ReLu**) rectifier function:

$$\text{ReLu}(f) = max(0,f)$$

- This function behaves in the same way as a diode in radioelectric circuits. This function is not differentiable, but has the following differentiable approximation:

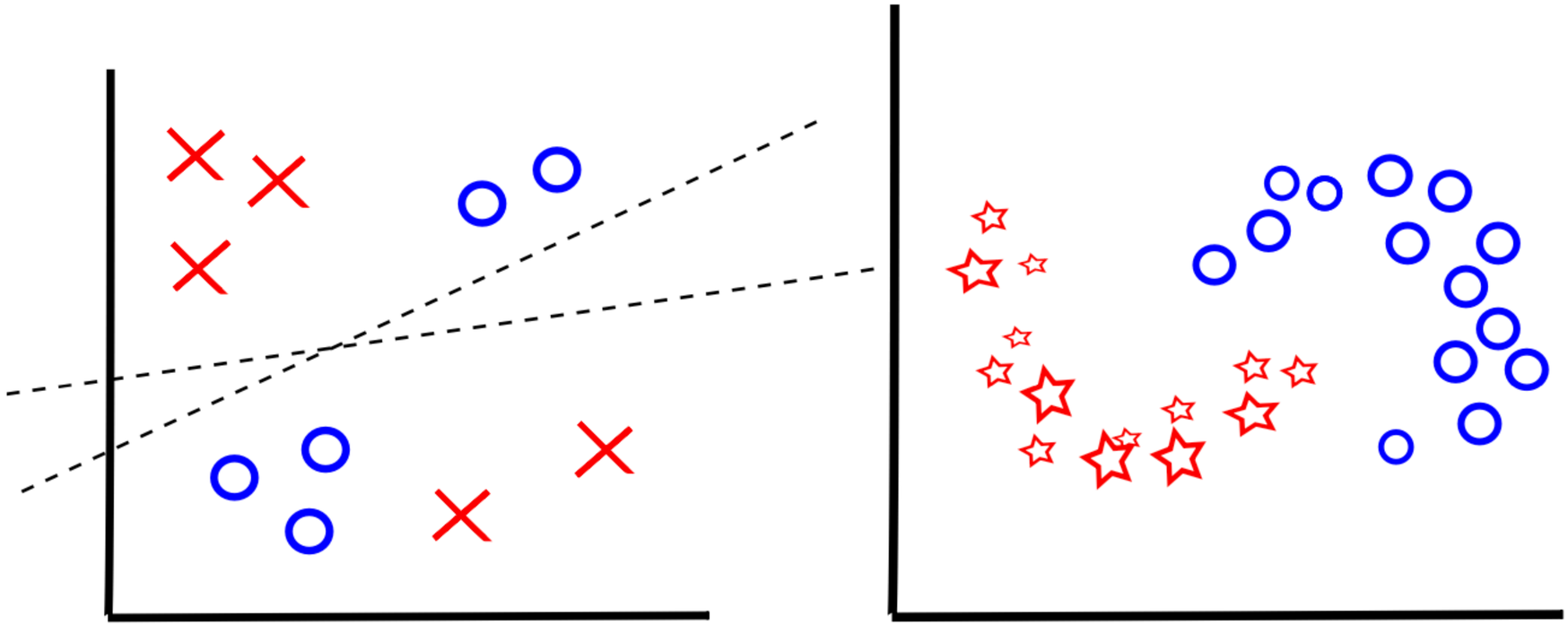
$$ln(1 + exp(f))$$



Multilayer neural networks

Limits of applicability of single-layer networks

- Single-layer networks are only applicable for linearly separable samples



Two-layer neural network

- A two-layer neural network is a linear combination of D neurons (single-layer neural networks):

$$a(x,\omega) = \sigma^{(2)}\left(\sum_{i=1}^{D}\omega_i^{(2)}\cdot\sigma^{(1)}\left(\sum_{i=1}^{d}\omega_{ji}^{(1)}x_j+\omega_{0i}^{(1)}\right)+\omega_0^{(2)}\right)$$

- In vector terms

$$a(x,\omega) = \sigma^{(2)}\left(\omega^{T^{(2)}}\cdot\sigma^{(1)}\left(\left[\omega_1^{T^{(1)}}x,\ldots,\omega_D^{T^{(1)}}x\right]\right)\right)$$

- The vector w of neural network parameters is obtained by connecting all neural network parameters at all layers:

$$\omega = \{W^{(1)},\omega^{(2)}\}$$

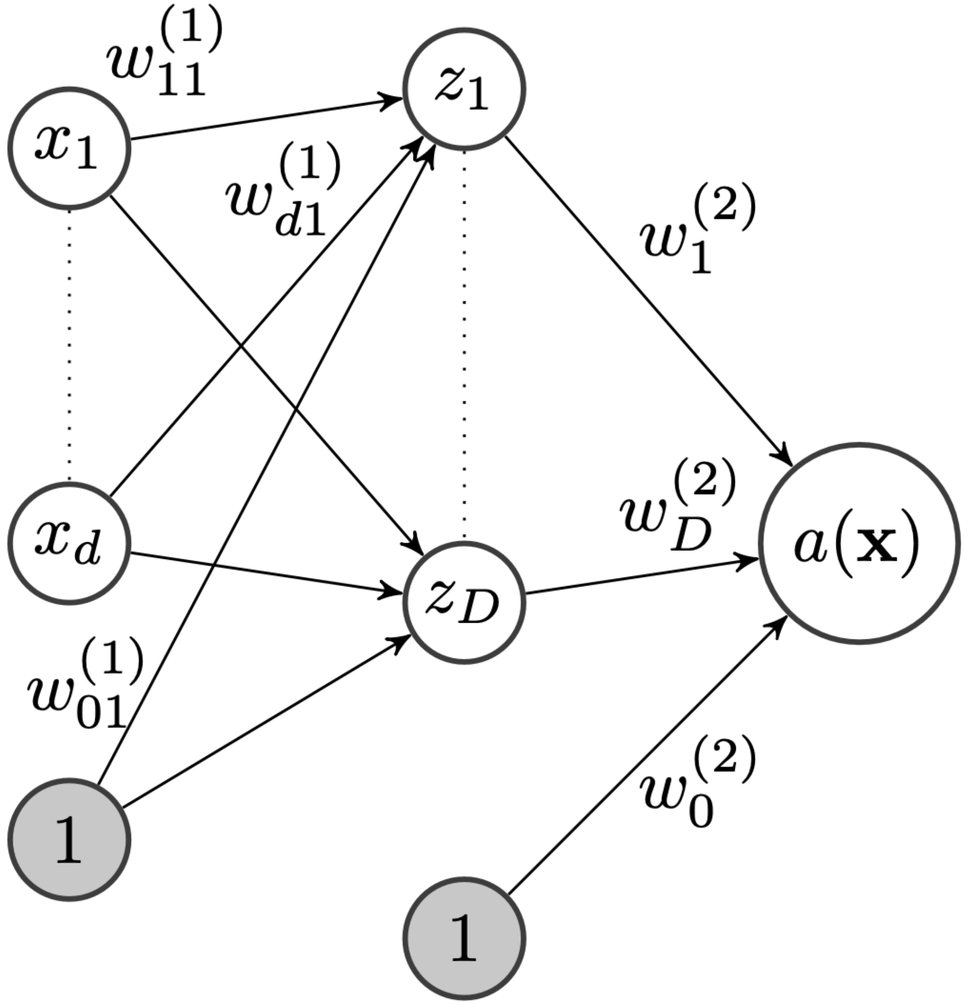
- Where

$$W^{(1)} = [\omega_0^{(1)},\omega_1^{(1)},\ldots,\omega_d^{(1)}]^T \in R^{(d+1)\times D}$$

$$\omega^{(1)} = [\omega_{0i}^{(1)},\omega_{1i}^{(1)},\ldots,\omega_{di}^{(1)}]^T \in R^{d+1},\quad \omega^{(2)} = [\omega_0^{(2)},\omega_1^{(2)},\ldots,\omega_D^{(2)}]^T \in R^{D+1}$$

Two-layer neural network

- Analogously, we can build as many layers as we want



Separating ability of multilayer neural network

- Theorem (Universal approximation theorem, K. Hornik, 1991)

For any continuous function $f(x)$ there is a neural network $a(x)$ with linear output $a(x, W) = \sigma^{(M)}(x)$, where

$$f^{(k)}(x) = \omega_0^{(k)} + W^{(k)} z^{(k-1)}(x), z^{(k)}(x) = \sigma^{(k)}(f^{(k-1)}(x))$$

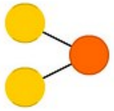
approximating $f(x)$ with a given accuracy

The theorem holds for various activation functions, in particular for the sigmoid function and the hyperbolic tangent function.

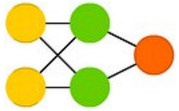
To obtain the required approximation, it is necessary to determine the optimal values of the parameters ω^*

Neural Networks

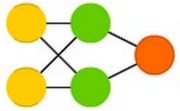
Perceptron (P)



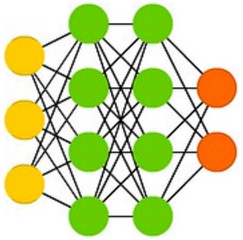
Feed Forward (FF)



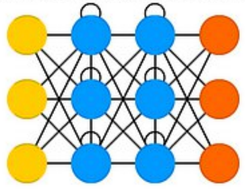
Radial Basis Network (RBF)



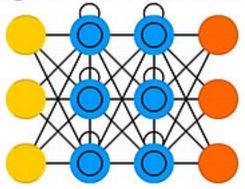
Deep Feed Forward (DFF)



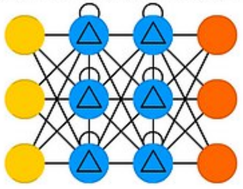
Recurrent Neural Network (RNN)



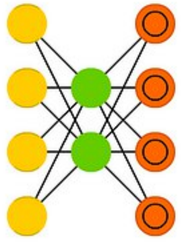
Long / Short Term Memory (LSTM)



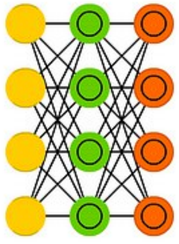
Gated Recurrent Unit (GRU)



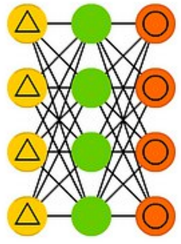
Auto Encoder (AE)



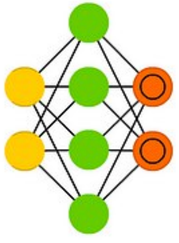
Variational AE (VAE)



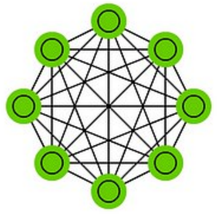
Denoising AE (DAE)



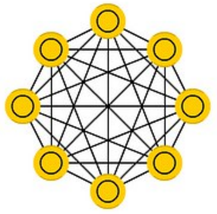
Sparse AE (SAE)



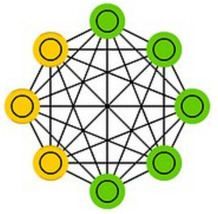
Markov Chain (MC)



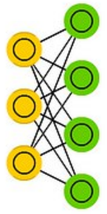
Hopfield Network (HN)



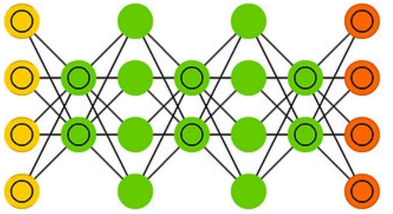
Boltzmann Machine (BM)



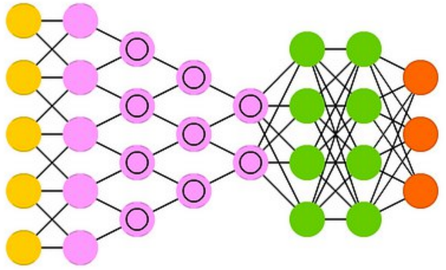
Restricted BM (RBM)



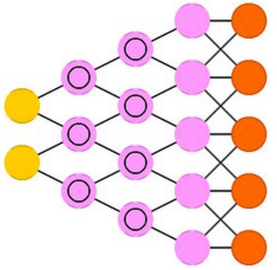
Deep Belief Network (DBN)



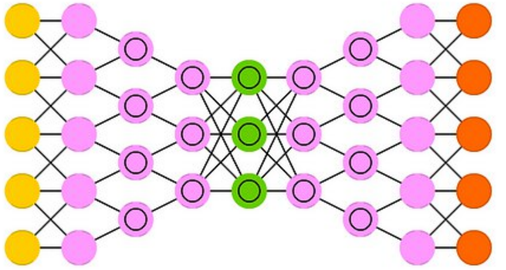
Deep Convolutional Network (DCN)



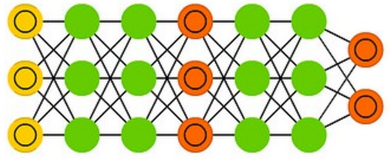
Deconvolutional Network (DN)



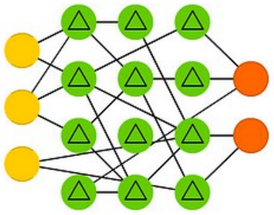
Deep Convolutional Inverse Graphics Network (DCIGN)



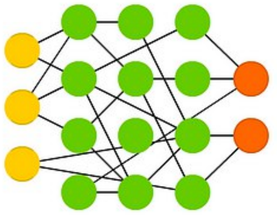
Generative Adversarial Network (GAN)



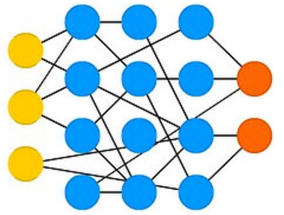
Liquid State Machine (LSM)



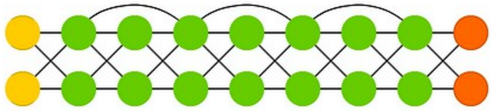
Extreme Learning Machine (ELM)



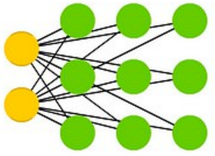
Echo State Network (ESN)



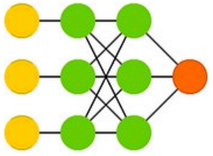
Deep Residual Network (DRN)



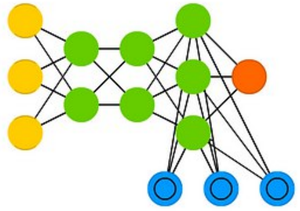
Kohonen Network (KN)



Support Vector Machine (SVM)

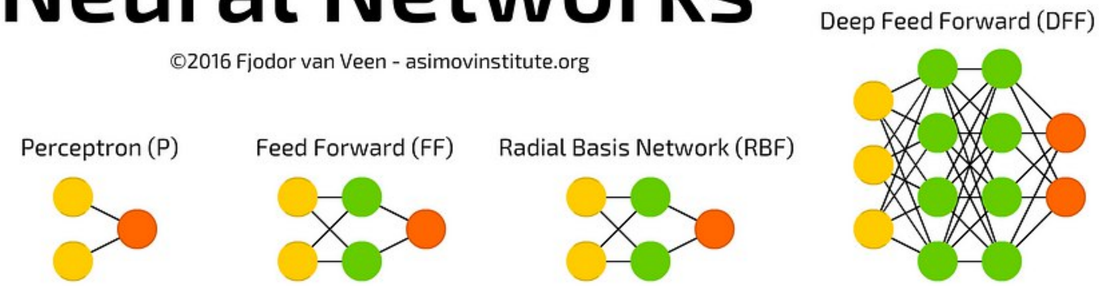


Neural Turing Machine (NTM)



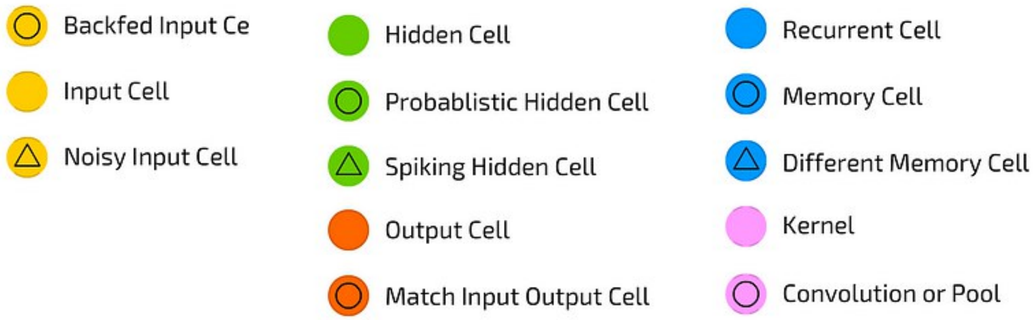
- Backfed Input Ce
- Input Cell
- Noisy Input Cell
- Hidden Cell
- Probablistic Hidden Cell
- Spiking Hidden Cell
- Output Cell
- Match Input Output Cell
- Recurrent Cell
- Memory Cell
- Different Memory Cell
- Kernel
- Convolution or Pool

Neural Networks

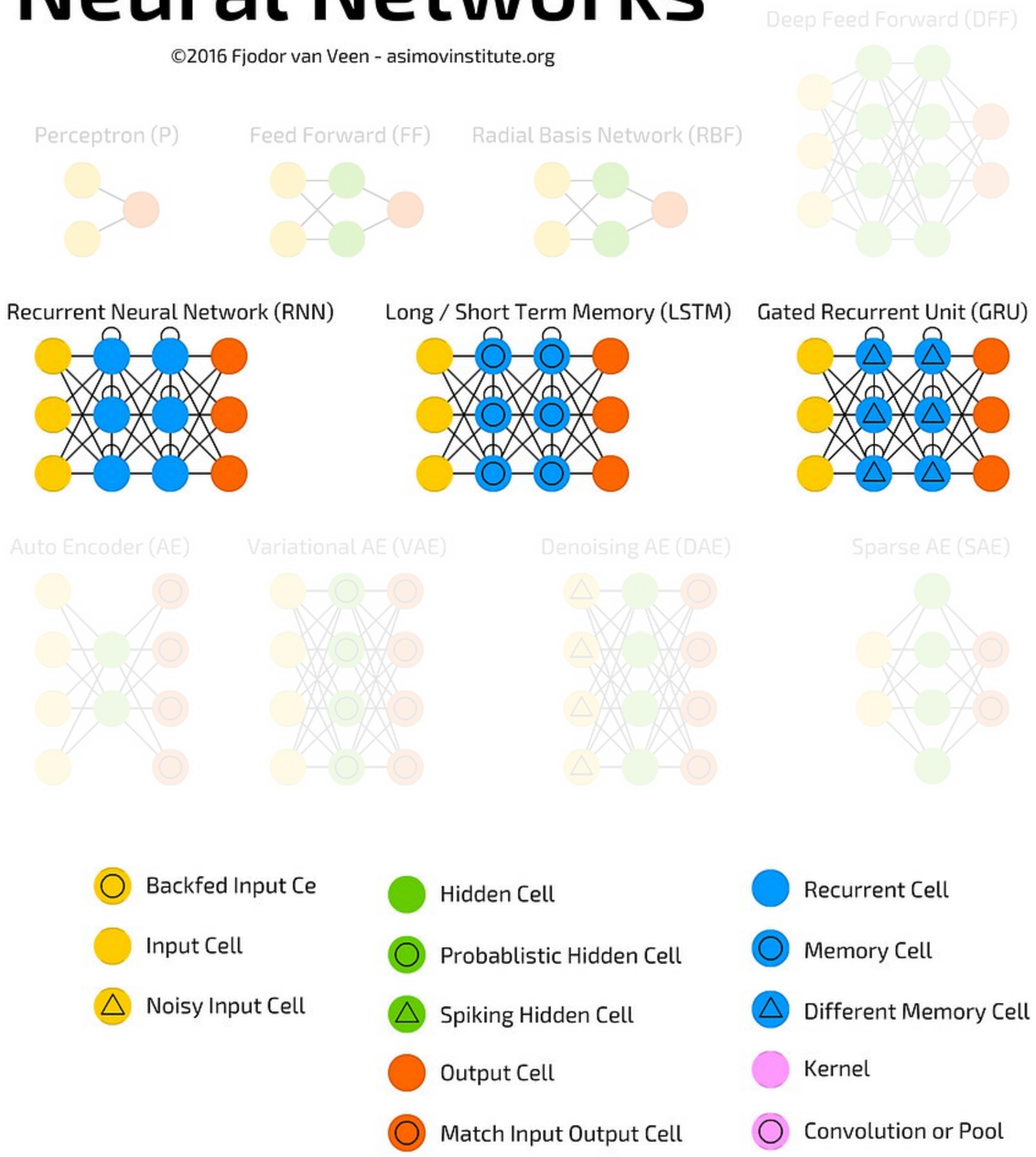


Feedforward Neural Networks (FNNs)

- **Configuration:** Data flows in one direction, from input to output, without cycles.
- **Structure**:
  - Input layer: Receives raw input data.
  - Hidden layers: Perform transformations using weights, biases, and activation functions.
  - Output layer: Provides the final predictions.
- **Applications:** Classification, regression, and basic pattern recognition.
- **Examples**:
  - Single-layer perceptron (1 hidden layer).
  - Multi-layer perceptron (MLP) with multiple hidden layers.



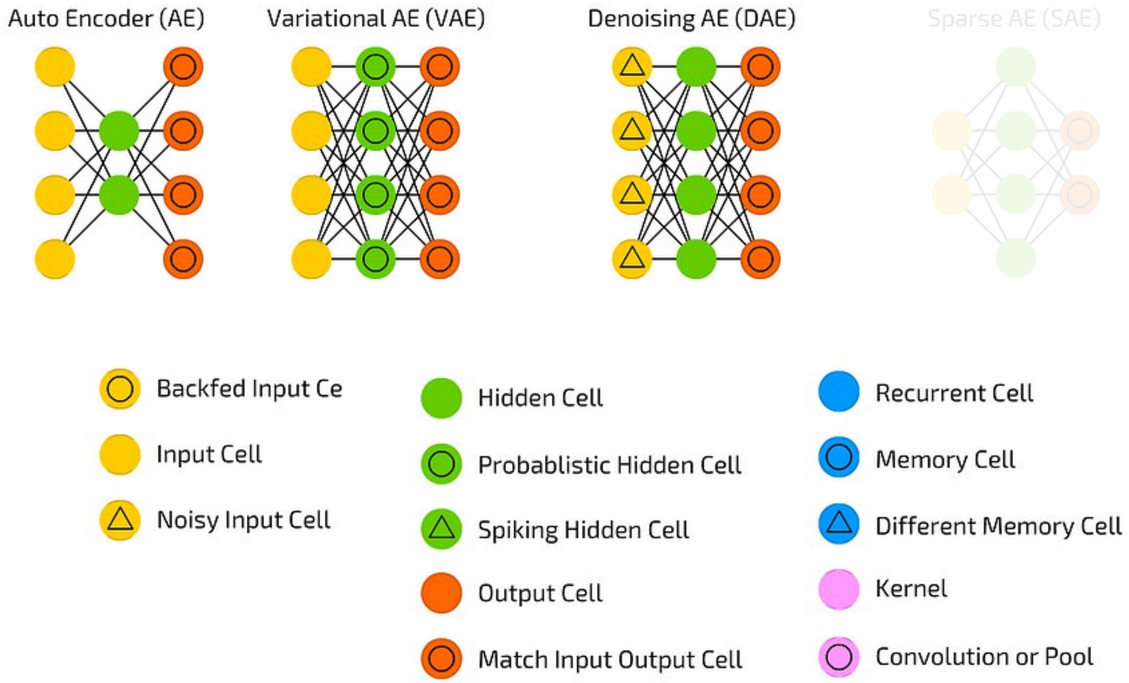
Neural Networks



Recurrent Neural Networks (RNNs)

- **Configuration:** Designed to process sequential data by maintaining a "memory" of past inputs.
- **Structure**:
  - Input layer: Takes sequential data (e.g., time-series).
  - Hidden layers: Include recurrent connections that feed outputs back into the same layer.
  - Output layer: Produces sequential predictions or classifications.
- **Variants**:
  - Long Short-Term Memory (LSTM): Handles long-range dependencies.
  - Gated Recurrent Units (GRUs): Simplified version of LSTMs.
- **Applications:** Language modeling, speech recognition, time-series forecasting

Neural Networks



Autoencoders

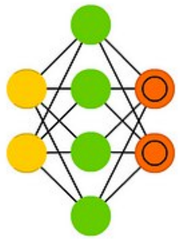
- **Configuration:** Networks designed for unsupervised learning, often for dimensionality reduction or feature learning.
- **Structure**:
  - Encoder: Compresses input data into a lower-dimensional representation.
  - Bottleneck: A central layer with minimal nodes to enforce compact encoding.
  - Decoder: Reconstructs the original input from the compressed representation.
- **Variants**:
  - Variational Autoencoders (VAEs): Probabilistic encoding and generation.
  - Denoising Autoencoders: Robust to noise in input data.
- **Applications:** Data compression, anomaly detection, and generative modeling

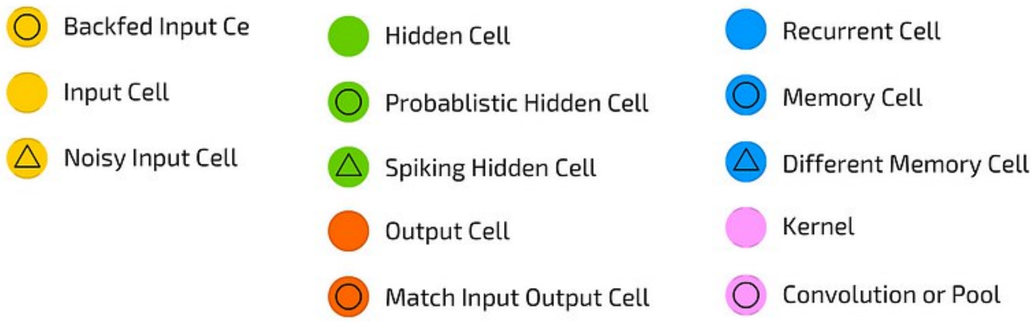
Neural Networks

Sparse Neural Networks

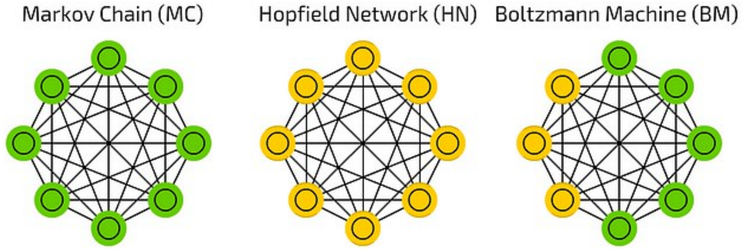
- **Configuration:** Not all connections between layers are used, reducing computational complexity.
- **Applications:** High-dimensional data, resource-constrained environments.

Sparse AE (SAE)





Neural Networks



Hopfield Network

Configuration: A type of recurrent neural network with symmetric connections and no self-loops, primarily used as an associative memory system.

Structure:

- Neurons: Represent the state of the network.
- Weights: Symmetric connections between neurons (no cycles).
- Energy function: A scalar value used to measure the stability of the network.

Functionality:

- Stores patterns as stable states (local minima of the energy function).
- Retrieves patterns by converging to the nearest stored state when given partial or noisy input.

Applications:

- Pattern recognition.
- Associative memory retrieval.
- Noise reduction in data.

Limitations: Limited storage capacity, prone to spurious states.

Example: Recognizing an incomplete image by recalling a stored version.

Markov Chain

Configuration: A probabilistic model that describes a sequence of possible events where the probability of each event depends only on the state attained in the previous event (Markov property).

Structure:

- States: Represent possible conditions or configurations.
- Transition probabilities: Define the likelihood of moving from one state to another.

Key Assumption: The future state depends only on the current state and not on the sequence of past states.

Applications:

- Predicting stock prices.
- Natural language processing (e.g., text generation).
- Modeling weather patterns.

Example: A weather model where the current day's weather (sunny, rainy) determines tomorrow's probabilities.

Boltzmann Machine

Configuration: A stochastic, generative neural network that learns a probability distribution over its set of inputs.

Structure:

- Visible units: Represent the input data.
- Hidden units: Capture the dependencies between visible units.
- Connections: Undirected, weighted connections between nodes.
- Energy function: Determines the likelihood of a configuration; lower energy states are more probable.

Variants:

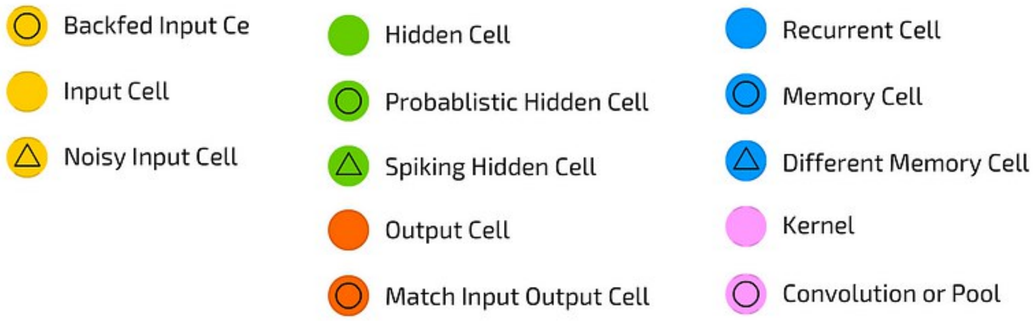
- Restricted Boltzmann Machine (RBM): A simplified version where visible and hidden units form a bipartite graph (no connections within a layer).

Applications:

- Feature learning and dimensionality reduction.
- Collaborative filtering (e.g., recommendation systems).
- Generative modeling.

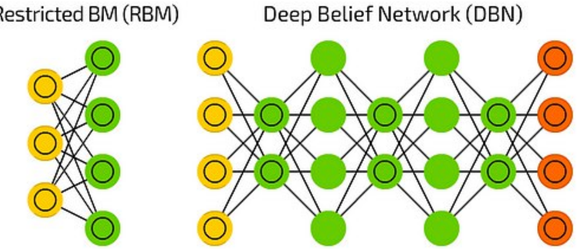
Examples:

- Learning latent features of images.
- Netflix recommendation system using RBMs.

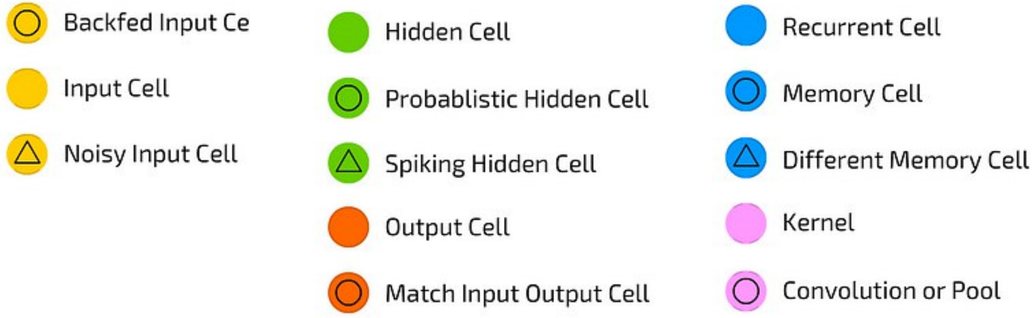


Neural Networks

Restricted Boltzmann Machine (RBM)



- **Configuration:**
  RBM is a two-layer generative stochastic neural network designed for unsupervised learning. It models the joint probability of visible and hidden variables using an energy function.
- **Structure**:
  - Two layers:
    - **Visible layer:** Represents input features.
    - **Hidden layer:** Captures latent patterns.
  - Fully connected between visible and hidden layers, with no connections within a layer.
- **Key Assumption:**
  Visible units are conditionally independent given the hidden units, and vice versa. This independence simplifies computations.
- **Applications**:
  - Feature learning and dimensionality reduction.
  - Collaborative filtering (e.g., recommendation systems).
  - Pretraining deep networks like Deep Belief Networks (DBNs).
- **Example:**
  A movie recommendation system: RBMs can learn patterns in user-movie interactions to predict unseen user preferences.



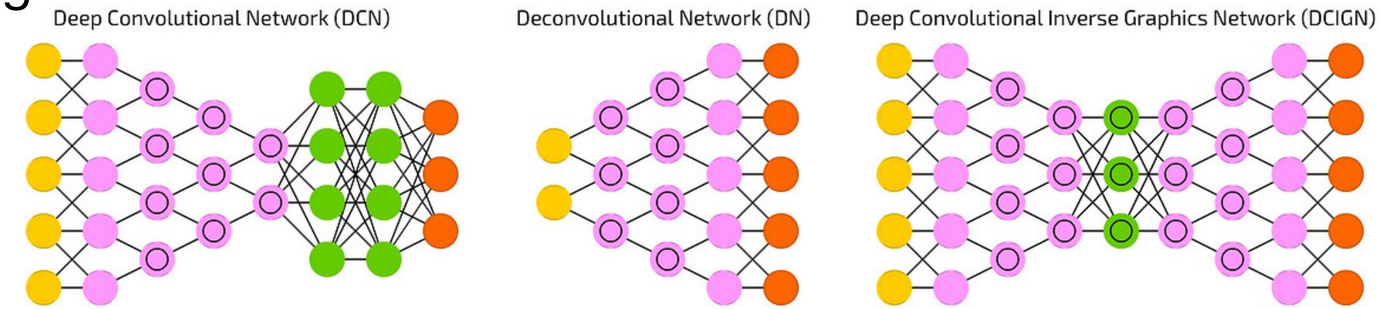
Deep Belief Networks (DBNs)

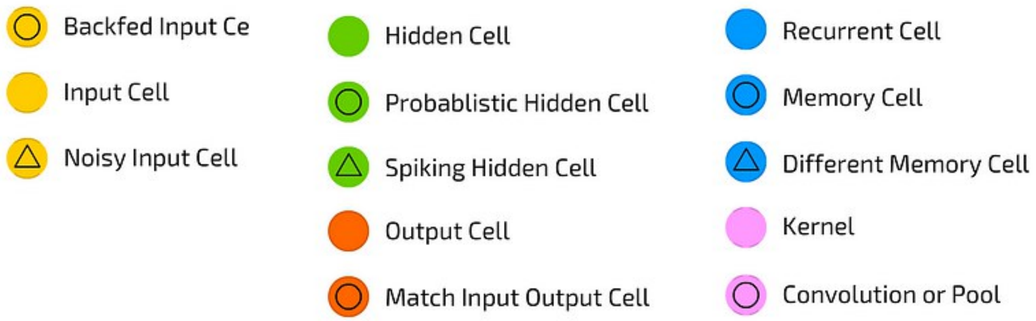
- **Configuration:** Composed of stacked Restricted Boltzmann Machines (RBMs).
- **Structure**:
  - Pre-training layers unsupervisedly, followed by fine-tuning for supervised tasks.
- **Applications:** Feature learning, classification, and regression.

Neural Networks

Convolutional Neural Networks (CNNs)

- **Configuration:** Specialized for processing grid-like data (e.g., images).
- **Structure**:
  - Convolutional layers: Extract spatial features using kernels/filters.
  - Pooling layers: Reduce dimensionality while preserving key features.
  - Fully connected layers: Combine features for classification or regression.





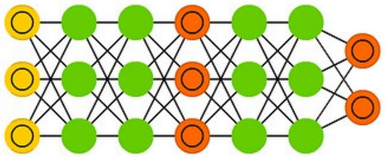
- **Applications:** Image recognition, object detection, and video processing.
- **Examples:** AlexNet, VGGNet, ResNet.

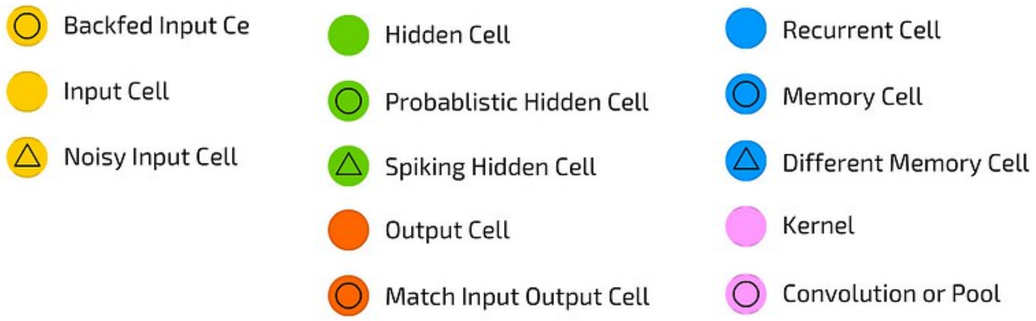
Neural Networks

Generative Adversarial Networks (GANs)

- **Configuration:** Consist of two networks (generator and discriminator) competing against each other.
- **Structure**:
  - Generator: Creates fake data resembling the training data.
  - Discriminator: Differentiates between real and fake data.
- **Applications:** Image synthesis, video generation, and data augmentation.
- **Examples:** DCGAN, StyleGAN.

Generative Adversarial Network (GAN)

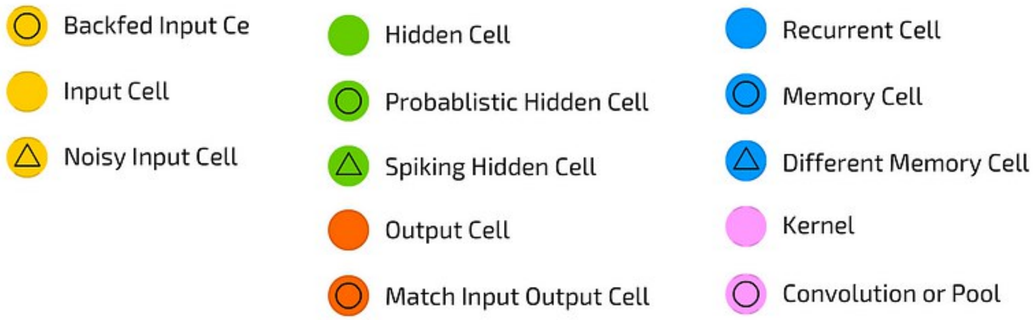


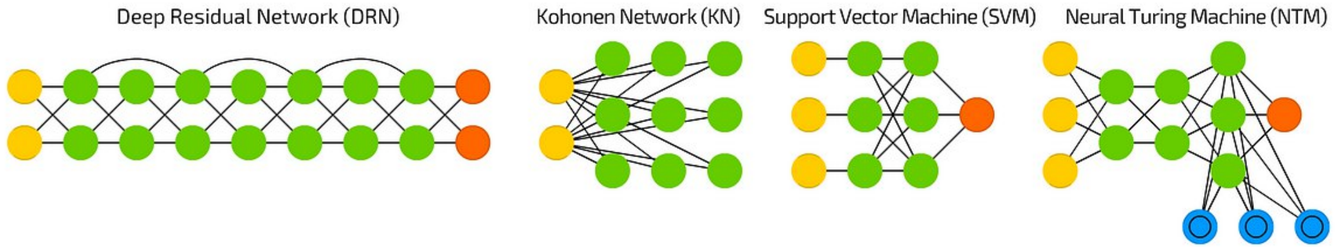


Neural Networks

Residual Networks (ResNets)

- **Configuration:** Incorporates shortcut connections to address vanishing gradient problems.
- **Structure**:
  - Residual blocks: Directly pass input to a deeper layer, skipping intermediate layers.
- **Applications:** Very deep networks for image and signal processing.
- **Examples:** ResNet-50, ResNet-101.





Neural Networks

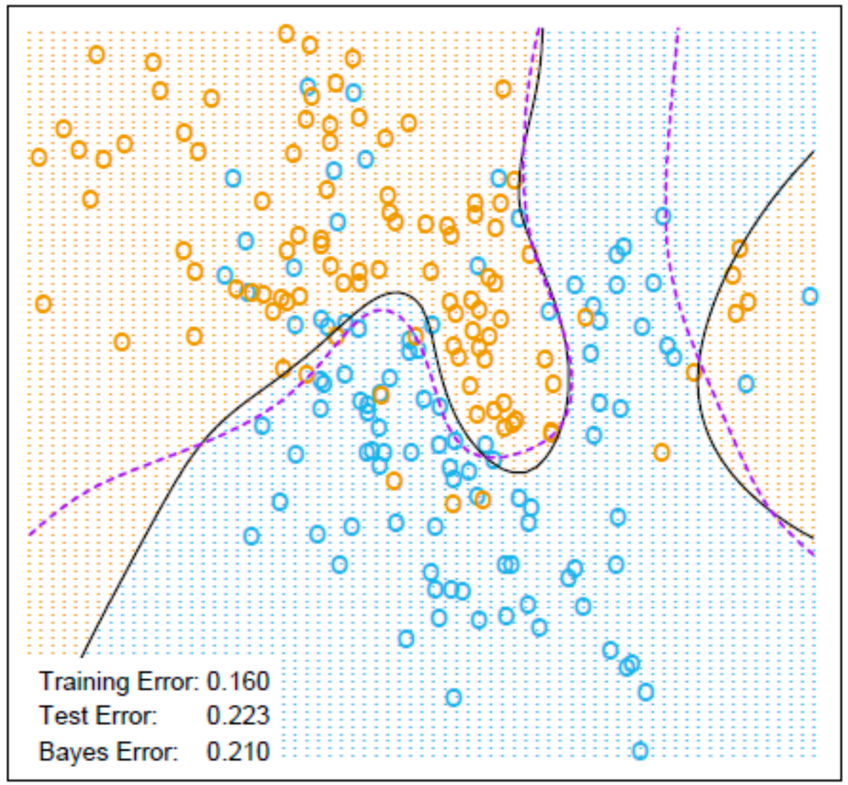
1. **Depth:** Number of layers; deeper networks model complex patterns but may require more data and training.
2. **Width:** Number of neurons per layer; wider networks capture more features but may overfit if too wide.
3. **Connections:** Fully connected vs. sparsely connected layers influence computational efficiency and learning.
4. **Activation Functions:** Non-linear functions enable learning complex relationships.
5. **Regularization:** Techniques like dropout and batch normalization improve generalization.

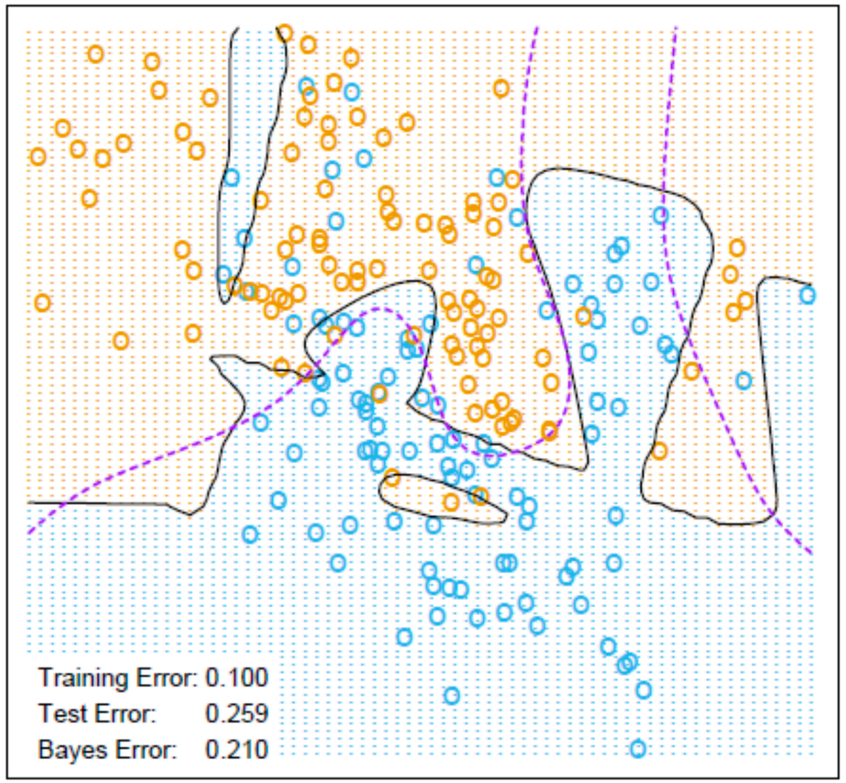
Each configuration serves specific purposes, and selecting the right architecture depends on the task, data type, and computational constraints.

Error function

Separating Surface: The Overfitting Problem

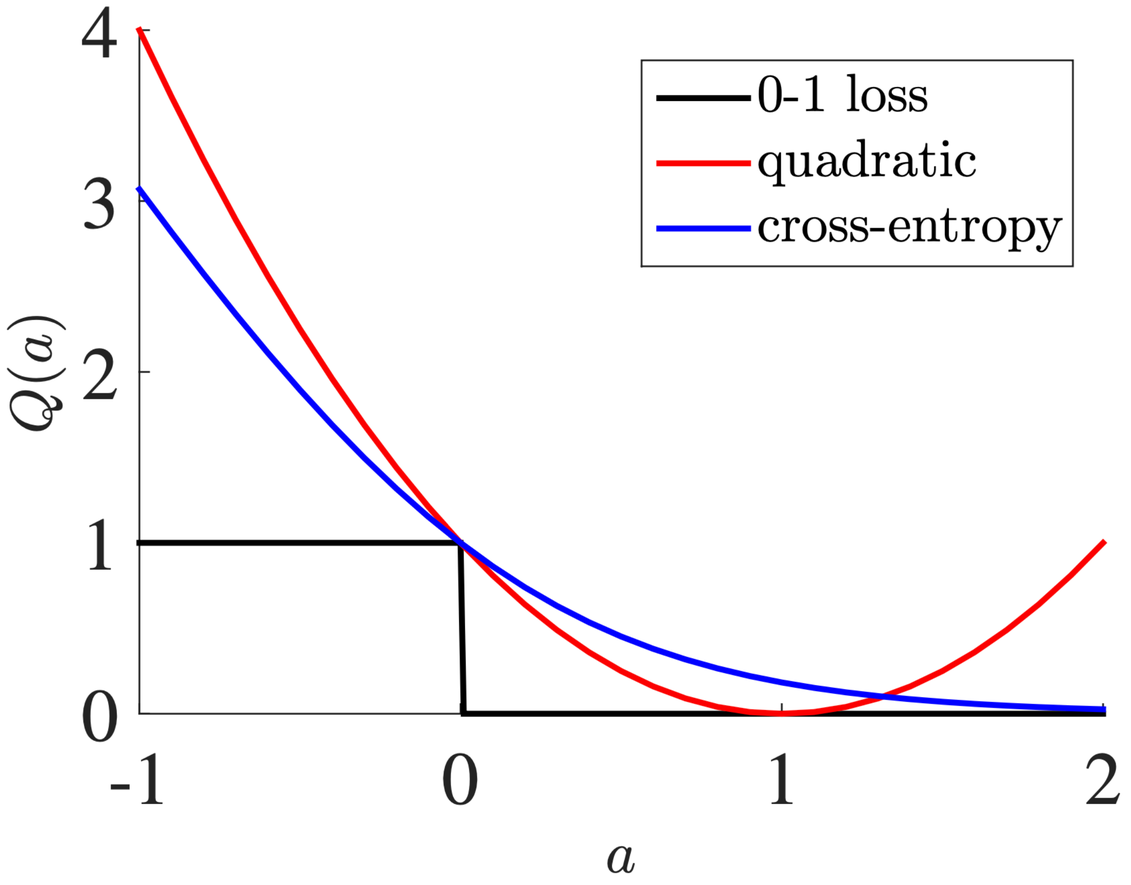
- The quality of the approximation of the target variable y by the function $a(x, \omega)$ depends on the choice of parameters ω





Neural network error functions

- Let $Q(\omega)$ be an error function that depends on both the configuration ω of the neural network and its composition. The problem of finding the optimal parameters ω^*, those for which the neural network error is minimal, has the form:
- $\omega^* = argmin_\omega Q(\omega)$



Neural network error functions

The choice of error function depends on the specifics of the problem being solved:

- Regression problem

$$Q(\omega) = \sum_{i=1}^{l} |a(x_i, \omega) - y_i|$$

- is not differentiable

- Regression problem. Differentiable function

$$Q(\omega) = \sum_{i=1}^{l} (a(x_i, \omega) - y_i)^2$$

- Classification problem

$$Q(\omega) = \sum_{i=1}^{l} [sign\ a(x_i, \omega) = y_i]$$

- «0-1 loss»

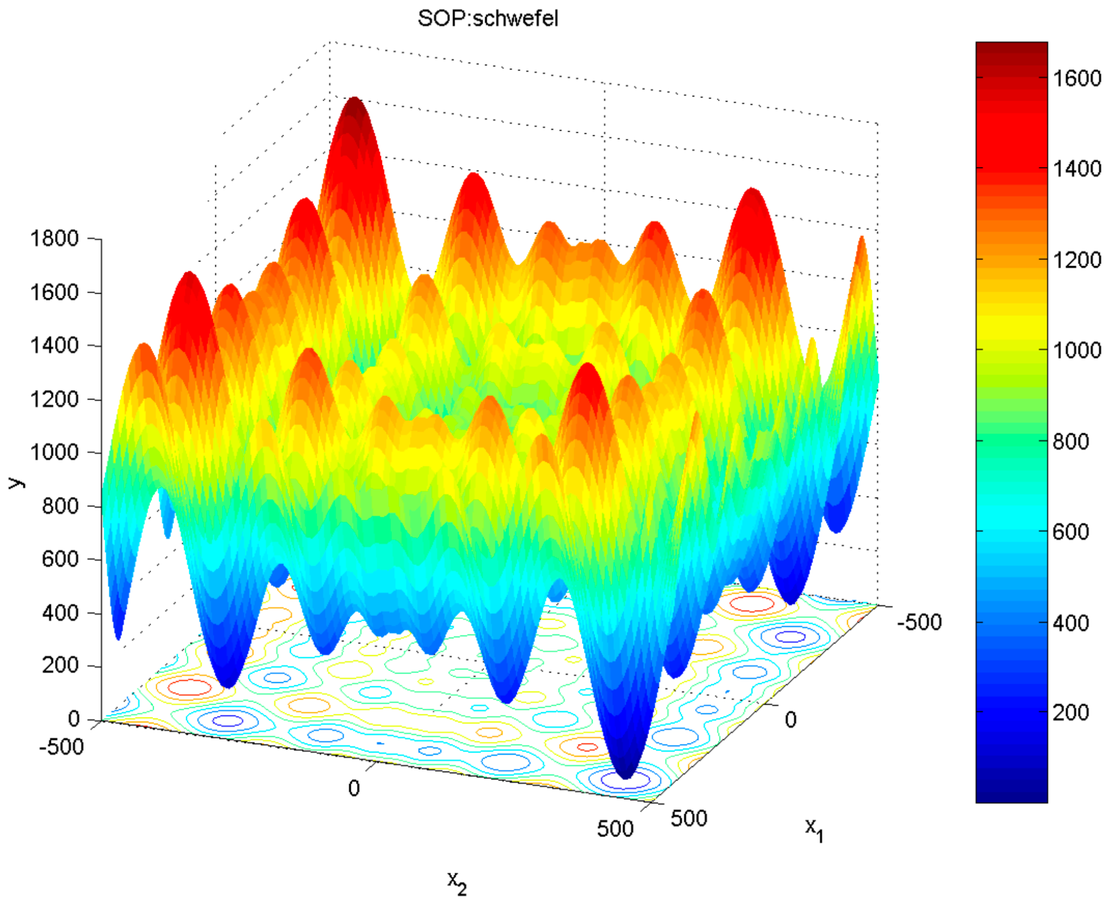
- Classification problem. Differentiable function

$$Q(\omega) = -\sum_{i=1}^{l} \left(y_i \ln a(x_i, \omega) + (1 - y_i)\ln(1 - a(x_i, \omega))\right)$$

Optimization of neural network parameters

Optimization problem

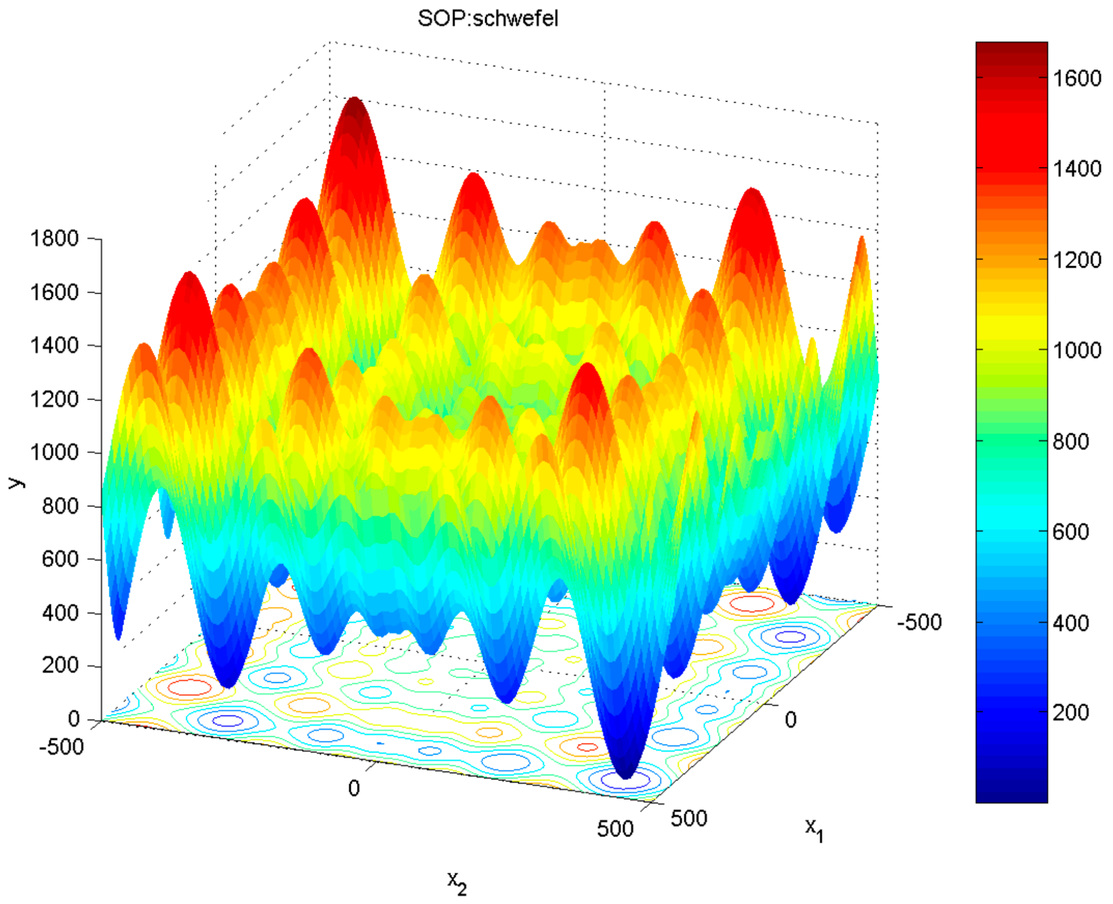
- Find optimal values of parameters:
- $\omega^* = argmin_\omega Q(\omega)$
- Q - error function
- Q depends on the sample, the structure of the neural network (the number of layers, neurons and types of activation functions) and the value of the parameter vector ω



Optimization problem

Two types of algorithms to find the optimal values of parameters:

- Stochastic optimization algorithms:
  - Randomly trying various $\omega_1, \omega_2, \ldots$
  - Genetic optimization algorithm $\omega_1 \rightarrow \omega_2 \rightarrow \ldots$
  - Simulated annealing, the values of w are set according to a schedule;
- Gradient descent algorithms.



Regularization and thinning of neural networks

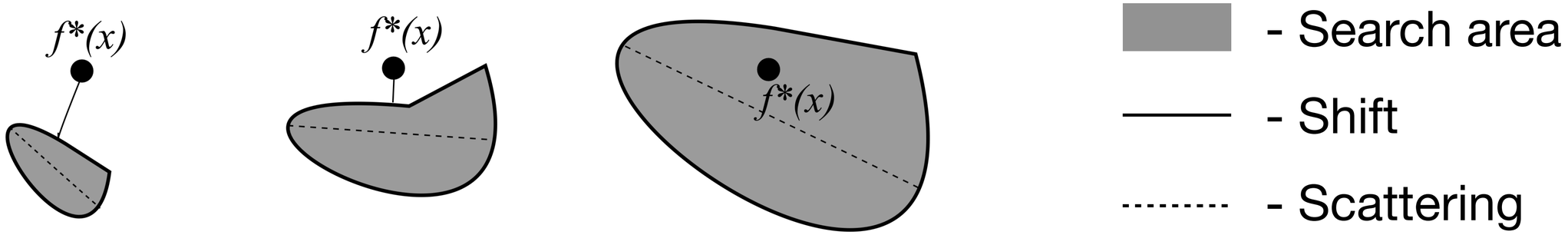
Regularization

- To avoid overfitting, it is necessary to modify the optimization problem: introduce a penalty for large weight values:

$$\omega^* = argmin_\omega \left[Q(\omega) + \tau \sum_{i,j,k} (\omega_{ij}^{(k)})^2 \right]$$

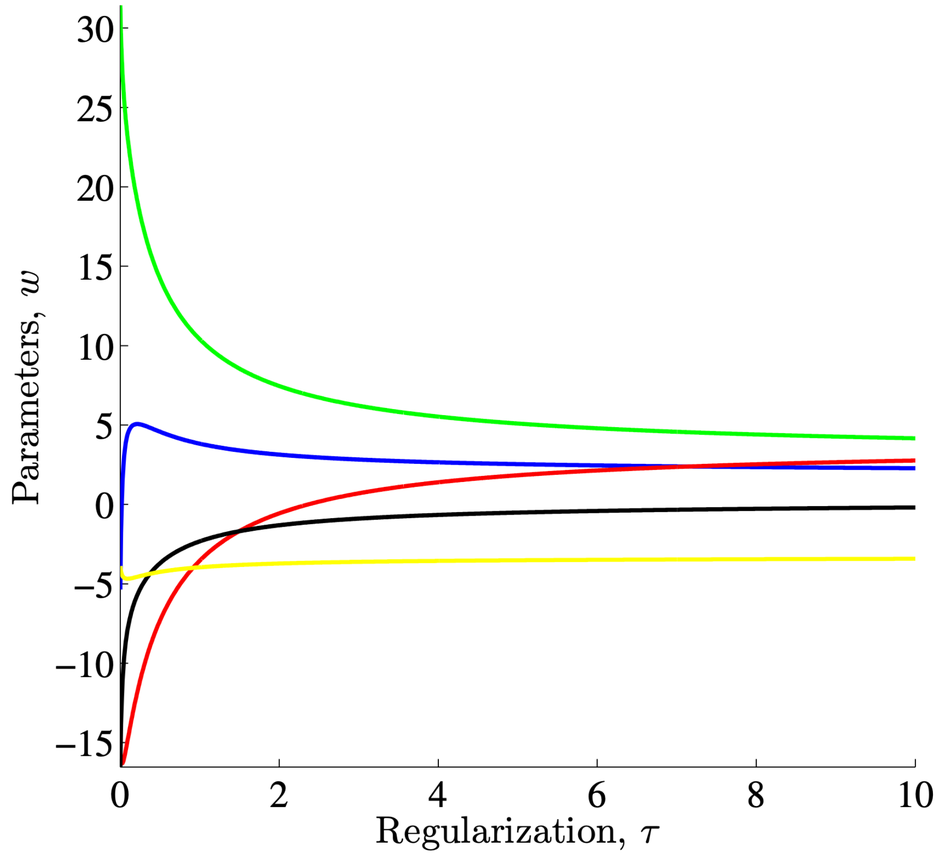
- τ is the regularization coefficient, which controls the rigidity of the constraints on the parameters ω

- The smaller τ, the more accurately the function $a(x)$ describes the sample



Regularization

- Regularization does not reduce the number of parameters or simplify the network structure
- As τ increases, the parameters stop changing

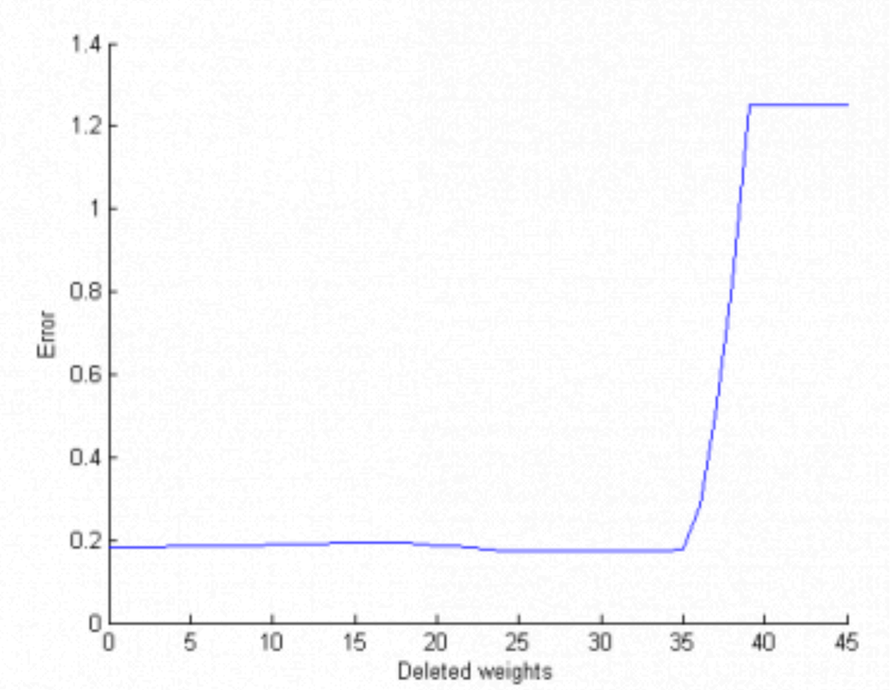


Network thinning

- To reduce the number of parameters, we can exclude some neurons or connections
- The principle of exclusion:
  - if the error function does not change, then the neural network can be simplified further

A parameter can be removed if:

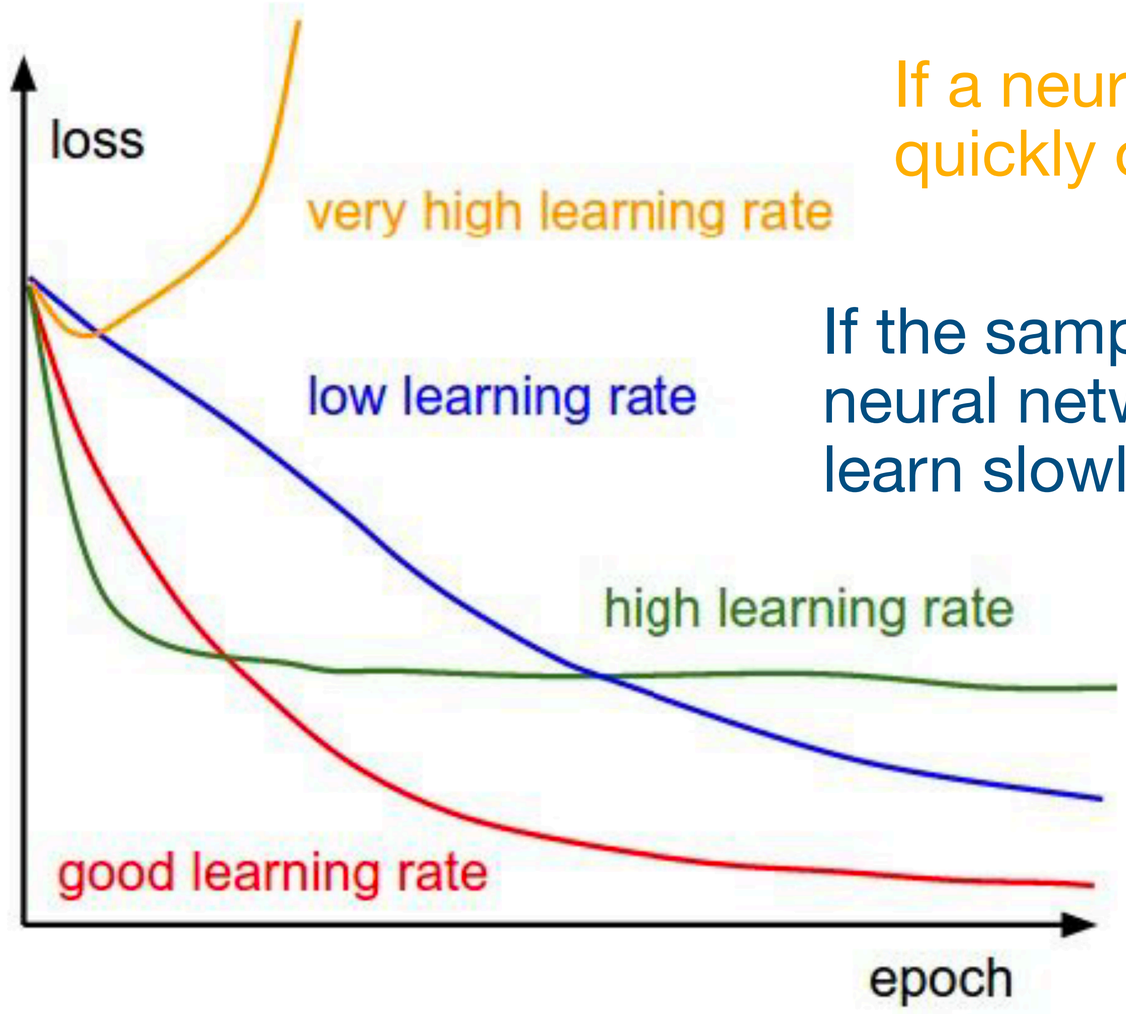
- its value is close to zero
- the corresponding signal has a large dispersion, i.e. it reacts to noise in the data
- its removal practically does not change the error function



Building a neural network

- The neural network can operate in two modes:
  - Training mode: In this mode, the structure of the neural network is set and its parameters are optimized.
  - Operation mode: calculating the values of $a(x, \omega^*)$ for fixed parameter values
- To build a network you should specify:
  - the number of layers
  - the number of neurons in each layer
  - the type of activation function in each layer
  - the type of error function
  - It is also desirable that the prepared sample does not contain gaps, and that the features are normalized

Stabilization of network parameters



If a neural network is too complex, it will quickly overtrain

If the sample is complex or very noisy for the neural network, then the neural network will learn slowly

If the sample complexity corresponds to a neural network

Stabilization of network parameters

- It is worth to check the difference between the values of the error function during training and control
- This difference should not be significant
- If it is large, it means that the neural network has been overtrained and its complexity should be changed