

## Seminar Series Neural Networks for Finance

### Lecture 1

#### Aim

From linear to non-linear theory: towards an understanding of supervised learning

17-02-2022

Balint Negyesi ([b.negyesi@tudelft.nl](mailto:b.negyesi@tudelft.nl))

*with the help of Kristoffer Andersson ([kristoffer.andersson@cwi.nl](mailto:kristoffer.andersson@cwi.nl))*

# Agenda

- 1 General introduction to machine learning
- 2 From statistics to machine learning: classical statistical inference
- 3 From linear to non-linear theory: regression
  - OLS regression
  - Non-linear regression
  - Generalization properties
- 4 From linear to non-linear theory: classification
  - Motivating example: Iris data set
  - Logistic regression
  - Other classification approaches
- 5 Non-linear theory: ANNs and DNNs
  - Single Layer Perceptron
  - Building blocks of neural networks
  - Activations
  - DNN
- 6 Teaser: capacity and optimization

General introduction to machine learning

# Machine learning - General

Machine learning is a collection of techniques to "learn" a relationship between an *input space*  $A$  and an *output space*  $B$ . Often highly automated and with a statistical approach.

**Example 1:** (binary classification)

We want to classify images of cats and dogs. Assuming equal format of each image  $A$  can be a space of  $X \times Y$ —matrices representing pixels of images and  $B = \{\text{"cat"}, \text{"dog"}\}$ .

**Example 2:** (function approximation)

We want to approximate  $\mathbb{R} \ni x \mapsto f(x) \in [0, 1]$ . Then, clearly  $A = \mathbb{R}$  and  $B = [0, 1]$ .

# Machine learning - General

Denote by  $H$  the true relation we want to learn, for  $x \in A$ ,  $H(x) \in B$  is the "truth". The aim for a machine learning algorithm is to achieve an accurate representation  $\Phi \approx H$  in some sense. What do we need?

- A measure to define accuracy of our approximation  $L(\Phi) = d(H, \Phi)$ , referred to as the *loss function* ( $d$  refers to a non-specified metric).
- A collection of samples, or *features*  $\mathbf{x}_M = (x_1, x_2, \dots, x_M)$  from  $A$ , i.e.,  $x_i \in A$ .
- An empirical error measure, denoted  $\bar{L}(\Phi|\mathbf{x}_M)$ , satisfying  $\lim_{M \rightarrow \infty} \bar{L}(\Phi|\mathbf{x}_M) = L(\Phi)$ . Typically referred to as the *empirical loss function*.
- An optimization algorithm to approximately solve  $\inf_{\Phi \in \mathcal{H}} \bar{L}(\Phi)$  in a pre-specified space  $\mathcal{H}$ . Typically referred to as *training*.

# Machine learning - General scheme

Schematic scheme (some repetition of previous slide)

**Given data:**  $\mathbf{x}_M$  of size  $M$  (could also have labels in supervised learning, more on this later). Evaluation data  $\bar{\mathbf{x}}_M$  of size  $\bar{M}$ .

## ① Training:

Find an approximate minimizer of the loss function  $\Phi^*$ , *i.e.*,  $\hat{\Phi} \approx \operatorname{argmin}_{\Phi \in \mathcal{H}} \bar{L}(\Phi|\mathbf{x}_M)$ . Training or optimization is performed in a different way for each machine learning algorithm.

## ② Evaluation:

This step is to verify that the algorithm is able to generalize, *i.e.*, perform well beyond the training data. This is done by comparing  $\bar{L}(\Phi|\mathbf{x}_M)$  and  $\bar{L}(\Phi|\bar{\mathbf{x}}_M)$ . If  $\bar{L}(\Phi|\mathbf{x}_M) \ll \bar{L}(\Phi|\bar{\mathbf{x}}_M)$ , then the problem might be *overfitting*, *i.e.*, the algorithm is fitted well against the training data but performs poorly on unseen data.

# Machine learning - Different types of problems

Two main categories:

## **Supervised learning:**

Each input element  $x_i \in A$  comes with a corresponding output element  $y_i \in B$ . Implies that  $H(x_i) = y_i$ .

## **Unsupervised learning:**

Only input elements are available during training. Need to rely on specific structure of the problem at hand to construct a loss function.

## Supervised learning:

- Training data comes in pairs  $(x_i, y_i)$  such that  $H(x_i) = y_i$ , for  $i = 1, 2, \dots, M$ , where  $y_1, y_2, \dots, y_M$  are referred to as *labels*.
- As empirical loss function  $L(\Phi|\mathbf{x}) = \frac{1}{M} \sum_{m=1}^M d(\Phi(x_m), y_m)$  can be used.

**Example:** (Binary classification - American options)

At a certain point in time, given the price of the underlying, should the option be held or exercised? Optimal stopping policy.



# Machine learning - Supervised learning Example

**Input data:**  $x_1, \dots, x_M$  representing  $M$  states of the underlying asset.

**Labels:**  $y_1, \dots, y_M$  taking on values in  $\{0, 1\}$  where "0" means sub-optimal to exercise option and "1" means it is optimal to exercise option.

The aim is to approximate  $H : \mathbb{R}_+ \rightarrow \{0, 1\}$  with  $\Phi$ .

**Loss function:**  $L(\Phi|\mathbf{x}) = \frac{1}{M} \sum_{m=1}^M d(\Phi(x_m), y_m)$ , where  $d(a, b)$  could be  $|a - b|$  but usually more sophisticated to have a smoother loss function.

## Unsupervised learning:

- Training data consists of only input data  $x_1, \dots, x_M$  and the aim is to approximate  $H$  by  $\Phi$  without being able to observe  $H(x_i)$ .
- We need to come up with an empirical loss function
- As empirical loss function  $L(\Phi|\mathbf{x}) = \frac{1}{M} \sum_{m=1}^M F(\Phi(x_m))$ , where  $F$  is a certain *functional* such that if  $L(\Phi|\mathbf{x})$  is minimized (or maximized), then  $\Phi$  is a good approximator of  $H$  (at least for states around  $x_1, \dots, x_M$ ).

**Example:** (Option valuation - European options)

At a certain point in time, given the price of the underlying, what is the value of a European option? Solving the pricing PDE.

Pricing PDE:

$$\begin{cases} \frac{\partial V}{\partial t} = f(t, x, V, \frac{\partial V}{\partial x}, \frac{\partial^2 V}{\partial x^2}) & (t, x) \in [0, T] \times \Omega = (x_{\min}, x_{\max}), \\ V(t, x) = f(t, x), & (t, x) \in [0, T] \times \bar{\Omega}, \\ V(T, x) = g(x), & x \in \Omega = \{x_{\min}, x_{\max}\}. \end{cases}$$

Here  $f$  is determined by the dynamics of the underlying asset,  $g$  is the pay-off function and the boundary conditions are set up to be "asymptotically reasonable".

For instance, the Black–Scholes model gives  $f$  linear in  $V, \frac{\partial V}{\partial x}, \frac{\partial^2 V}{\partial x^2}$  making the PDE linear. Other models may yield a semi-linear or even a fully non-linear PDE.

# Machine learning - Unsupervised learning Example

We do not know the PDE solution apriori and therefore, have no access to labels as in supervised learning. Need to create a loss function which uses the structure from the PDE.

$$L(\Phi) = \text{MSE}_{\text{inner}} + \text{MSE}_{\text{TC}} + \text{MSE}_{\text{BC}},$$

where

$$\begin{cases} \text{MSE}_{\text{inner}} = \frac{1}{M_{\text{inner}}} \sum_{m=1}^{M_{\text{inner}}} \left| \frac{\partial \Phi(t_m, x_m)}{\partial t} - f(t_m, x_m, \Phi, \frac{\partial \Phi}{\partial x}, \frac{\partial^2 \Phi}{\partial x^2}) \right|^2 \\ \text{MSE}_{\text{TC}} = \frac{1}{M_{\text{TC}}} \sum_{m=1}^{M_{\text{IC}}} |g(x_m) - \Phi(T, x_m)|^2, \\ \text{MSE}_{\text{BC}} = \frac{1}{M_{\text{BC}}} \sum_{m=1}^{M_{\text{BC}}} |f(t_m, x_m) - \Phi(t_m, x_m)|^2. \end{cases}$$

# Machine learning – Other categories of tasks

- Anomaly detection
- Synthesis and sampling
- Denoising
- Density estimations
- ...

From statistics to machine learning: classical statistical inference

From statistics to machine learning: classical statistical inference

*"Machine learning is essentially a form of applied statistics with increased emphasis on the use of computers to statistically estimate complicated functions and a decreased emphasis on proving confidence intervals around these functions[...]"*

Goodfellow et al.: Deep Learning (2016), MIT Press

# Statistical Estimates

- Statistics is about extracting as much information as possible from measurement data, e.g. *fitting functions, estimating finite/infinite parameters, distributions, assessing confidence intervals around estimates*
- In the most general, measure theoretic statement: a statistical model is a family of probability measures  $\{\mathbb{P}_\theta, \theta \in \Omega\}$  on a measurable space  $(\mathcal{X}, \mathcal{B})$  acting on the distribution of data  $X$ 
  - $\Omega$  **finite dimensional: parametric models**
  - $\Omega$  infinite dimensional: non-parametric models
- In this course we will focus on **parametric models**



# Statistical Point Estimates

- Two main schools of statistical inference:
  - ① Classical/Frequentist: the true value of a parameter  $\theta$  is fixed but not known; the *point estimate*  $\hat{\theta}$  is a random variable as a function of the randomly generated data
  - ② Bayesian: data is directly observed thus cannot be random; the true value of the parameter  $\theta$  is unknown therefore uncertain and thus can be modeled as a random variable
- Frequentist: summarizes data into a single value of the parameter set  $\hat{\theta}$  and then makes predictions
- Bayesian: encompasses prior knowledge into a **prior** probability distribution  $p(\theta)$  which is then used to make predictions through the Bayes theorem  $p(\theta|x) = \frac{p(x|\theta)p(\theta)}{p(x)}$
- Through most of the course we will take the frequentists' perspective, however when studying **General Adversarial Networks** (GANs) we will rely on the **Bayesian approach**

# Estimators, bias, variance

- Given data  $\{x_1, \dots, x_m\}$ , a point estimator is a function which maps the data to an element of the parameter space

$$\hat{\theta}_m := g(x_1, \dots, x_m)$$

- Example:  $x_i \in \mathbb{R}$ , the statistical model consists of fitting a line on the data points
- Bias:**  $\text{bias}(\hat{\theta}) := \mathbb{E}[\hat{\theta}] - \theta$ , where the expectation is taken over the data-generating distribution,  $\text{bias}(\hat{\theta}_m) = 0 \implies$  **unbiased** estimator
- Variance:**  $\text{Var}[\hat{\theta}_m]$
- The **mean-squared error (MSE)** defines a trade-off between the minimization of the two

$$\text{MSE}(\hat{\theta}_m) = \mathbb{E}[(\hat{\theta}_m - \theta)^2] = \text{bias}^2(\hat{\theta}_m) + \text{Var}[\hat{\theta}_m]$$

# Notions of generalization

$$\text{MSE}(\hat{\theta}_m) = \mathbb{E} [(\hat{\theta}_m - \theta)^2] = \text{bias}^2(\hat{\theta}_m) + \text{Var} [\hat{\theta}_m]$$

- Notions of generalization: capacity, underfitting, overfitting
  - 1 underfitting: small capacity leads to low variance, large bias
  - 2 overfitting: increased capacity, decreases bias but increases variance

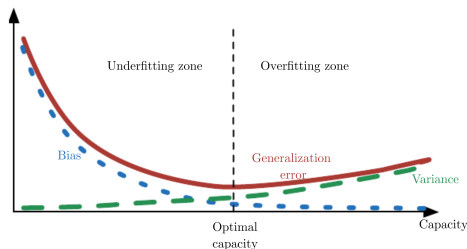


Figure: Goodfellow et al.: Deep Learning (2016), MIT Press [Figure 5.6, pg. 128]

# Maximum Likelihood Principle

- **Maximum Likelihood Estimator (MLE)** principle: given a data generating distribution  $\mathbb{P}_{\text{data}}(x)$  and a statistical model  $\mathbb{P}_{\text{model}}(x|\theta)$ 
$$\hat{\theta}_{\text{ML}} := \arg \max_{\theta} \mathbb{P}_{\text{model}}(X|\theta) \quad (1)$$
- Under the assumption of **independent, identically distributed (iid)** data samples  $X = \{x_1, \dots, x_M\}$ , this can be written  $\hat{\theta}_{\text{ML}} := \arg \max_{\theta} \prod_{m=1}^M \mathbb{P}_{\text{model}}(x_m|\theta)$

From linear to non-linear theory: regression

# Ordinary Least-Squares Regression

- We are given a data set  $\{x_{m1}, \dots, x_{md}; y_m\}_{m=1}^M$  of  $M$  measurements drawn from the same distribution
- The elements of the vector  $\mathbf{x}_m := (x_{m1}, \dots, x_{md}) \in \mathbb{R}^d$  are called the *explanatory/independent* variables;  $y_m \in \mathbb{R}$  is the response/dependent variable
- We assume a *linear relation with noise* between the *fixed* measurements of independent variables and the response variable

$$y = \mathbf{x}^T \boldsymbol{\beta} + \epsilon \quad (2)$$

with some **parameter** vector  $\boldsymbol{\beta} \in \mathbb{R}^d$

- the *noise term*  $\epsilon$  incorporates the residual influence on  $y$  not captured by the regressors (measurement errors in  $y$ , missing regressors, etc.)

# Ordinary Least-Squares Regression

## Goal

The goal of Ordinary Least-Squares (OLS) regression is to find a(/the?) set of regression coefficients  $\beta = (\beta_1, \dots, \beta_d)$  such that the Euclidean norm of the noise term is minimized

$$\hat{\beta} := \arg \min_{\beta} \|\epsilon\|_2^2 = \arg \min_{\beta} \|\mathbf{y} - \mathbf{X}\beta\|_2^2 = \sum_{m=1}^M \left| y_m - \mathbf{x}_m^T \beta \right|^2, \quad (3)$$

where  $\mathbf{X} = \begin{pmatrix} x_{11} & \dots & x_{1d} \\ \dots & \dots & \dots \\ x_{m1} & \dots & x_{md} \end{pmatrix}$ ,  $\mathbf{y} = (y_1, \dots, y_m)$ ,  $\epsilon = (\epsilon_1, \dots, \epsilon_m)$

# Ordinary Least-Squares Regression

- Denote mean-squared *loss*, subject to the measurements  $\mathbf{D} := (\mathbf{X}, \mathbf{y})$  by  $L(\beta|\mathbf{D}) = \|\mathbf{y} - \mathbf{X}\beta\|_2^2$  which is equal to

$$L(\beta|\mathbf{D}) = \mathbf{y}\mathbf{y}^T - \mathbf{y}^T\mathbf{X}\beta - \beta^T\mathbf{X}^T\mathbf{y} + \beta^T\mathbf{X}^T\mathbf{X}\beta$$

- Due to the **convexity** of  $\beta \mapsto L(\beta|\mathbf{D})$  the unique global minimum is found where the gradient vanishes

$$\mathbf{0} = \nabla_{\beta} L(\hat{\beta}|\mathbf{D}) = -2\mathbf{y}^T\mathbf{X} + 2\hat{\beta}^T\mathbf{X}^T\mathbf{X}$$

## Unique Solution

The unique minimizer of (3) is given by

$$\hat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}, \quad (4)$$

and the corresponding approximations

$$\hat{\mathbf{y}} := \mathbf{X}\hat{\beta} \quad (5)$$



# Key Remarks

- Under standard assumptions (linearity, independence of  $\epsilon_m$ 's, homoscedasticity – the variance of  $\epsilon_m$  does not depend on the values  $\mathbf{x}_m$ , lack of *perfect multicollinearity* –  $\mathbf{X}$  has full column rank  $d$ ) the OLS problem admits to a **unique, closed-form solution**
- The intercept (constant) can be incorporated into the model as a zero'th dimension
- The distribution of  $\mathbf{X}$  has a great impact on the accuracy of  $\hat{\beta}$  – assumes high-quality sampling methods
- The OLS estimate coincides with the Maximum Likelihood Estimate (MLE)

# Non-linear regression

- Same setting
  - data coming from a fixed *data generating* distribution:  $\{x_{m1}, \dots, x_{md}, y_m\}_{m=1}^M$  of  $M$  measurements drawn from the same distribution
  - explanatory and response variables  $\mathbf{x}_m := (x_{m1}, \dots, x_{md}) \in \mathbb{R}^d$ ,  $y_m \in \mathbb{R}$
- However, the assumptions of linearity established by (2) **no longer holds**
- Instead, we consider the following **non-linear extension** of the model

$$y = H(\mathbf{x}) + \epsilon \quad (6)$$

where  $H : \mathbb{R}^d \rightarrow \mathbb{R}$  is a deterministic mapping

## Question?

What could be an appropriate family of statistical models  $\{\phi(x|\theta) : \mathbb{R}^d \rightarrow \mathbb{R} : \theta \in \mathbb{R}^p\}$  which spans a "large enough" space for all potential  $H$ s?

- In what follows we will study classes of statistical models which are suitable for specific tasks
- For now, let us briefly assume we have a family of "**black box**" statistical models of the form  $\{\phi(x|\theta) : p \in \mathbb{N}, \theta \in \mathbb{R}^p\}$  which possess this quality
- Within such a family of statistical models, the question still remains: *how do we choose the "right" number of  $p$  parameters?*

# Generalization Properties

- Here  $p$  is called a **hyperparameter** of the model which needs to be chosen carefully
- The ultimate goal of the regression problem is not to fit the *training data*  $\{x_{m1}, \dots, x_{md}; y_m\}_{m=1}^M$  with arbitrary accuracy but rather to **approximate the relation** (6)
- In fact, we seek to find a set of parameters  $\{\hat{\theta} \in \mathbb{R}^p : p \in \mathbb{N}\}$  such that  $\phi(x|\hat{\theta}) \approx H(x)$  for any  $x$  **drawn from the data generating distribution** – regardless whether it is in the training data or not
- trade-off: representational capacity naturally grows with  $p$ , however generalization capability decreases – recall Figure 1

# Hyperparameter Selection: Under- and Overfitting

In some applications one has direct access to the data generating distribution (e.g.: Monte Carlo option pricing), in most cases however we are only provided with a finite sample

$$\mathbf{D} = \{(\mathbf{x}_m, y_m)\}_{m=1}^M$$

## Question

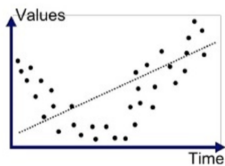
Question: in case of the latter, how does one ensure *unbiased* estimates, with good generalization properties?

- Idea: split the data set  $\mathbf{D}$  into a partition, a subset for training  $\mathbf{D}_{train} \subset \mathbf{D}$  and one for testing  $\mathbf{D}_{test} \subset \mathbf{D}$ , such that  $\mathbf{D}_{train} \cap \mathbf{D}_{test} = \emptyset$  and  $\mathbf{D}_{train} \cup \mathbf{D}_{test} = \mathbf{D}$ .
- We choose multiple  $p$ 's and *optimize* their corresponding statistical models with a **black box** optimization method (soon explained) according to an appropriately selected *loss function* (e.g. MSE)

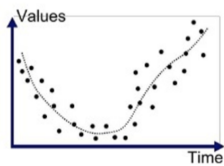
# Hyperparameter Selection: Under- and Overfitting

- Out of the optimal parameter sets, the one with lowest loss generalizes best. However, the error measure remains inaccurate to future predictions due to overfitting on the test set.
- Partitioning into three subsets: training data  $\mathbf{D}_{train} \subset \mathbf{D}$ , validation data  $\mathbf{D}_{val} \subset \mathbf{D}$  and test data  $\mathbf{D}_{test} \subset \mathbf{D}$ , such that  $\mathbf{D}_{train} \cup \mathbf{D}_{val} \cup \mathbf{D}_{test} = \mathbf{D}$  and  $\mathbf{D}_{train} \cap \mathbf{D}_{val} \cap \mathbf{D}_{test} = \emptyset$  resolves this problem
- First, multiple parameter sets are trained on the training set.
- Second, the best performing set of parameters is chosen from the validation set.
- At last, the error measure is calculated on the test set.
- This method gives an unbiased estimate of the error measure on future data.
- Similarly to **cross validation**

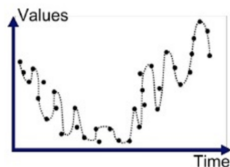
# Summary: Under- and Overfitting



Underfitted



Good Fit/Robust



Overfitted

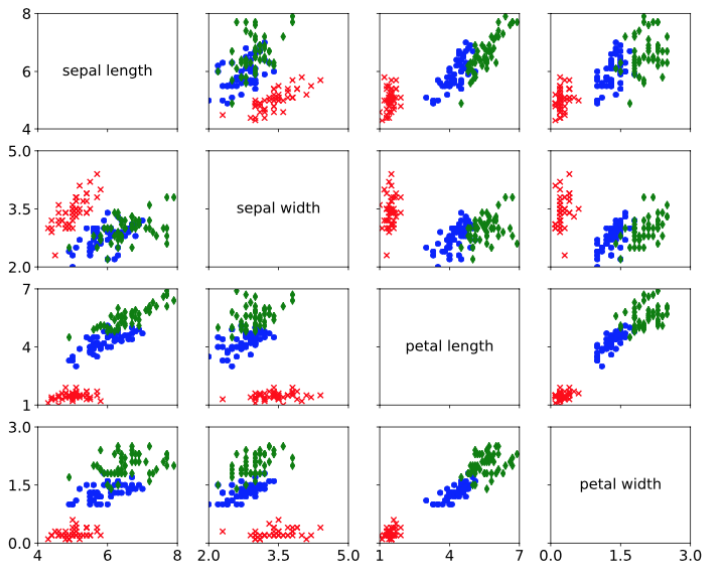
From linear to non-linear theory: classification



# Motivating Example: Iris data set

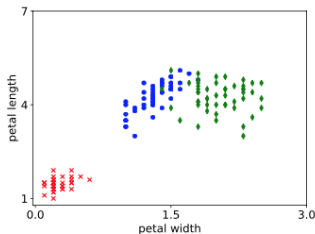
- The Iris data set is a multivariate set with 50 entries for each species. There are three species, consisting of four features: sepal length, sepal width, petal length and petal width.
- The four features can be displayed in a scatter plot
- Determining categories by using their features is called **classification**
- Setosa is blue, Versicolor is red, and Virginica is green colored

# Motivating Example: Iris data set

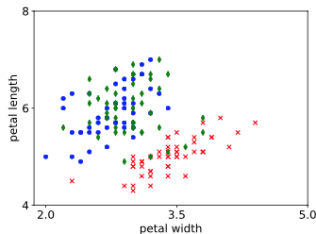


# Motivating Example: Iris data set

- See, left figure below, the Setosa species can be distinguished from the others, by using the petal length.



(a) Zoom 1



(b) Zoom 2

- In mathematical terms, it can be stated that the Setosa is **linearly separable**.
- Two sets  $X_1 \subset X$  and  $X_2 \subset X$  are called **linearly separable** if their convex hulls are disjoint.

# Motivating Example: Iris data set

- Consider the set  $X_1$ , the petal lengths of the Versicolor and the Virginica species and the set  $X_2$ , the petal lengths of the Setosa species. In this case

$$\text{convhull}(X_1) = \{x | x \in [\min(X_1), \max(X_1)]\} = [3.0, 6.9]$$

and

$$\text{convhull}(X_2) = \{x | x \in [\min(X_2), \max(X_2)]\} = [1.0, 1.9]$$

- Clearly,  $\text{convhull}(X_1) \cap \text{convhull}(X_2) = \emptyset$ , therefore  $X_1$  and  $X_2$  are linearly separable, and thus it is possible to classify the Setosa species in the Iris data set

# Motivating Example: Iris data set

- From the scatter plot at the right side of the last figure, the species are not easily distinguishable
- By creating convex hulls of the individual features, the two species are not linearly separable for the individual features
- It is impossible to classify the species using linear classifiers deterministically. We could create (hyper-)planes such that a minimal number of data entries is incorrectly separated
- This gives a statistically optimal result for the data set on which the classifier is trained
- **Support Vector Machines (SVM)** designed to find such optimal hyperplanes

Standard models of classification

# Standard Models of Classification

- We are given a **binary** response variable  $Y \in \{0, 1\}$ ; and a vector of real-valued regressors  $\mathbf{x}$  which are assumed to influence  $Y$  – *multinomial extensions are straightforward*
- We obtained a data set of measurements  $\{\mathbf{x} = (x_{m1}, \dots, x_{md}), y_m\}_{m=1}^M$
- Formally we describe the outcome of each measurement with a Bernoulli distribution with *unobserved* probability  $p_m$

$$Y_m | x_{m1}, \dots, x_{md} \sim \text{Bernoulli}(p_m)$$

- Define the linear *latent variable model* with an *auxiliary random variable*  $\tilde{Y}$  such that

$$\tilde{Y} = \mathbf{x}^T \boldsymbol{\beta} + \epsilon, \quad (7)$$

where  $\epsilon$  follows a given **symmetric, centered** distribution

# Standard Models of Classification

- Thereby  $Y$  can be interpreted as the indicator function

$$Y = \begin{cases} 1, & \tilde{Y} > 0 \\ 0, & \tilde{Y} \leq 0 \end{cases}$$

- By the symmetricity of the distribution of the noise term we have

$$\mathbb{P}[Y = 1|\mathbf{x}] = \mathbb{P}[\tilde{Y} > 0|\mathbf{x}] = \mathbb{P}[\epsilon > -\mathbf{x}^T\boldsymbol{\beta}] = \text{CDF}(\mathbf{x}^T\boldsymbol{\beta})$$

- The corresponding estimates are then defined by setting the distribution of the noise term, two classical choices
  - ➊ **Probit regression:**  $\epsilon \sim \mathcal{N}(0,1)$  which implies

$$\mathbb{P}[Y = 1|\mathbf{x}] = \Phi(\mathbf{x}^T\boldsymbol{\beta})$$

- ➋ **Logistic regression:**  $\epsilon$  follows a *standard logistic distribution* whose CDF is given by  $\frac{1}{1+e^{-x}}$

$$\mathbb{P}[Y = 1|\mathbf{x}] = \frac{1}{1 + e^{-\mathbf{x}^T\boldsymbol{\beta}}}$$



# Logistic Regression

- Logistic regression is more common in machine learning practices due to the convenience of a closed form CDF
- So far we have a prediction for the probability of the  $m$ 'th measurement taking the value 1

$$\mathbb{P}[Y_m = 1 | \mathbf{x}_m, \beta] = p(\mathbf{x}_m | \beta) = \frac{1}{1 + e^{-\mathbf{x}_m^T \beta}}$$

- Notice that the prediction of each class is no longer a linear mapping of  $X$ ; only in case of the latent variable
- In fact, the model is not linear in the predicted probabilities but rather in the so called log-**odds**'es

$$\text{odds}(\mathbf{x}) := \frac{p(\mathbf{x} | \beta)}{1 - p(\mathbf{x} | \beta)} = e^{-\mathbf{x}^T \beta}$$

- This establishes the direct relation between standard linear and logistic regressions

# Logistic Regression

- Due to the Bernoulli condition, under the assumption of independent samples the corresponding likelihood function is of the form

$$\mathcal{L}(\beta|\mathbf{X}, \mathbf{y}) = \prod_{m=1}^M \mathbb{P}[y_m|\mathbf{x}_m, \beta] = \prod_{m=1}^M p(\mathbf{x}_m|\beta)^{y_m} (1-p(\mathbf{x}_m|\beta))^{1-y_m}$$

- Unlike in OLS regression the model **does not have a closed-form** (maximum likelihood) solution
- To formulate an equivalent loss minimization problem we define  $L(\beta|\mathbf{X}, \mathbf{y}) := -\log \mathcal{L}(\beta|\mathbf{X}, \mathbf{y})$  leading to the **cross-entropy loss**

$$L(\beta|\mathbf{X}, \mathbf{y}) = - \sum_{m=1}^M y_m \log(p(\mathbf{x}_m|\beta)) + (1 - y_m)(1 - p(\mathbf{x}_m|\beta)) \quad (8)$$

$$L(\beta|\mathbf{X}, \mathbf{y}) = - \sum_{m=1}^M y_m \log(p(\mathbf{x}_m|\beta)) + (1 - y_m)(1 - p(\mathbf{x}_m|\beta)) \quad (9)$$

- There is **no closed-form** solution for the minimizer
- Iterative root searching methods can be applied to gather an estimate for the minimizer

$$\hat{\beta} := \arg \min_{\beta} L(\beta|\mathbf{X}, \mathbf{y})$$

- For instance: Newton's method, *(stochastic) gradient descent coming soon...*
- The gradient of the cross-entropy loss is equal to that of MSE

$$\nabla_{\beta} L(\beta|\mathbf{X}, \mathbf{y}) = \mathbf{X}(\hat{\mathbf{y}} - \mathbf{y})$$

# Other classification approaches

- clustering
- $k$ -nearest neighbours
- isolation forests
- decision trees
- support vector machines (SVM)
- naive Bayes classifier
- ...

Non-linear theory: Artificial Neural Networks and Deep Neural Networks

# From linear to non-linear mappings

- Linear models have the obvious defect of model capacity only spanning the space of linear mappings
- We wish to extend and represent non-linear functions of some vector input  $\mathbf{x} \mapsto H(\mathbf{x})$
- Idea: apply a linear model to not  $\mathbf{x}$  itself but rather to a non-linear, vector-valued transformation of it  
 $\phi(\mathbf{x}|\theta) : \mathbb{R}^d \rightarrow \mathbb{R}^p$
- The resulting approximations of  $H(\mathbf{x})$  read as

$$\Phi(\mathbf{x}|\Theta := (\theta, \mathbf{w})) = \varphi^T(\mathbf{x}|\theta)\mathbf{w},$$

with  $\mathbf{w} \in \mathbb{R}^p$

## Question

How do we choose the family of  $\Phi(\mathbf{x}|\Theta)$ ?

# The Single Layer Perceptron Model

- **Single Layer Perceptron (SLP)** is the simplest neural network model originally designed to solve the binary classification problem

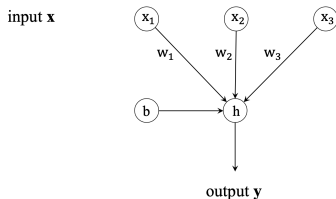


Figure: Illustration of an SLP

- Inputs (written as  $\mathbf{x}$ ) are passed through a weighted, directed graph.
- A function  $h$  corresponds to the non-linear mapping  $\phi(\mathbf{x}|\theta)$  from above

# The Single Layer Perceptron Model

- SLP has  $d + 1$  parameters: a  $d$ -dimensional vector of *weights*  $\mathbf{w} \in \mathbb{R}^d$  connecting the node of  $h$  to each input feature; and a *bias* denoted by  $b \in \mathbb{R}$
- The non-linearity  $h : \mathbb{R} \rightarrow \{-1, 1\}$  is defined by the following heaviside function with threshold  $b$

$$h(z) = \begin{cases} 1, & z > b, \\ -1, & z \leq b, \end{cases} \quad (10)$$

which can be interpreted as **true** or **false**

- The input of the non-linearity is a scalar defined by the inner product of the input vector  $\mathbf{x} \in \mathbb{R}^d$  and the weights:  $\mathbf{w}^T \mathbf{x}$
- Thus the mapping of SLP is given by the following *sequence of compositions*

$$\text{SLP}(\mathbf{x} | \theta := (\mathbf{w}, b)) = \begin{cases} 1, & \mathbf{w}^T \mathbf{x} > b, \\ -1, & \mathbf{w}^T \mathbf{x} \leq b. \end{cases} \quad (11)$$



# Training: Optimization Analytically

## Question

How to find an **optimal** set of parameters properly assigning a binary classification label?

- For linearly separable data, an algorithm optimizes/trains the parameter vector  $\theta = (b, \mathbf{w})$  such that the output of the perceptron corresponds to the correct class for all data entries in  $X$ .
- The algorithm starts by initializing a random parameter vector and a convergence test variable. It then picks a random sample from the data set and tests whether the sign of the weight vector assigns the correct label to the random sample.
- If an incorrect label is assigned, it updates the weight such that the argument of the sign function moves to the correct label.

# Training: Optimization Analytically

- For notational convenience we attach the bias node to the input as a 0'th (constant) dimension  $\mathbf{x}' = (1, \mathbf{x}) \in \mathbb{R}^{d+1}$ , and assign it  $w_0 = -b$  weight. These thresholds are equivalent

$$\boldsymbol{\theta}^T \mathbf{x}' \geq 0 \iff \mathbf{w}^T \mathbf{x} \geq b$$

- Consider the case of a positive label in  $y$  and a negative prediction label, the weight is updated by adding the data  $\mathbf{x}'$ . To see why this works, consider:

$$\text{sign}(\boldsymbol{\theta}^T \mathbf{x}') = -1. \quad (12)$$

Add  $\mathbf{x}'$  to  $\boldsymbol{\theta}$  and use the linearity of the inner product:

$$\text{sign}\left((\boldsymbol{\theta} + \mathbf{x}')^T \mathbf{x}'\right) = \text{sign}\left(\boldsymbol{\theta}^T \mathbf{x}' + \mathbf{x}'^T \mathbf{x}'\right) \quad (13)$$

- By definition, for the inner product we have  $\mathbf{x}'^T \mathbf{x}' \geq 0$ . Since  $\mathbf{x}' \neq 0$  from (12),  $\mathbf{x}'^T \mathbf{x} > 0$ .
- Therefore the weight update improves the prediction

# Perceptron Learning Algorithm for linearly separable data

- Let  $X$  be an  $M \times (d + 1)$  matrix with  $M$  measurements,  $d$  features and a column for the bias (all values equal 1), such that two classes in the set are linearly separable; let length  $M$  vector  $\mathbf{y} \in \{-1, 1\}^M$  indicate the class of a data entry of  $X$ .
- Choose a random  $\mathbf{w} \in \mathbb{R}^{d+1}$ . Set  $conv = 1$
- while  $conv$  do
  - Choose a random  $r \in U(0, M)$
  - Set  $\mathbf{x}' = X(r, \cdot)$
  - Set  $y = \mathbf{y}(r)$
  - Compute  $\hat{y} = \text{sign}(\boldsymbol{\theta}^T \mathbf{x}')$
- if  $y = 1$  and  $\hat{y} = -1$  then
  - Update  $\boldsymbol{\theta} := \boldsymbol{\theta} + \mathbf{x}'$ , endif
- if  $y = -1$  and  $\hat{y} = 1$  then
  - Update  $\boldsymbol{\theta} := \boldsymbol{\theta} - \mathbf{x}'$ , endif
- Convergence test:  $conv := 0$  if  $\text{sign}(X^T \mathbf{w}) = \mathbf{y}$
- end while

# Summary

- After each random sample, a convergence test is done by comparing all the signs of the samples in the data set with their corresponding labels.
- If not all data entries are classified correctly, the algorithm starts over by picking a new random sample, otherwise the algorithm has successfully found a set of weights which classifies the data, and terminates.
- To speed up the training time, one might consider using different algorithms (training of perceptrons can be translated to solving an interior point problem).

# Logistic regression as an SLP

- In the previous SLP example we specified a step function to model non-linear behavior
- If one replaces  $h : \mathbb{R} \rightarrow \mathbb{R}$  with the CDF of the standard logistic distribution

$$\text{CDF}(\boldsymbol{\theta}^T \mathbf{x}') = \frac{1}{1 + e^{-\boldsymbol{\theta}^T \mathbf{x}'}} ,$$

the resulting SLP scheme is equivalent to the logistic regression model

- The resulting equivalent SLP model can then be optimized in an identical fashion as in case of logistic regression: minimizing the cross-entropy loss with an iterative root searching algorithm

# Multinomial Logistic regression

- One can easily extend the binary logistic regression to multinomial models of  $K$  outcomes  $\{0, \dots, K-1\}$
- This is done by running  $K-1$  many binary classifications with a fixed outcome as pivot

$$\frac{\mathbb{P}[Y_m = 0]}{\mathbb{P}[Y_m = K-1]} = e^{-\beta_1^T \mathbf{x}_m}, \quad \dots, \quad \frac{\mathbb{P}[Y_m = K-2]}{\mathbb{P}[Y_m = K-1]} = e^{-\beta_{K-1}^T \mathbf{x}_m},$$

where  $\beta_k \in \mathbb{R}^d$

- The probabilities assigned to each class must sum up to 1:

$$\begin{aligned} \mathbb{P}[Y_m = K-1] &= 1 - \sum_{k=0}^{K-2} \mathbb{P}[Y_m = k] \\ \implies \mathbb{P}[Y_m = K-1] &= \frac{1}{1 + \sum_{k=0}^{K-2} e^{-\beta_k^T \mathbf{x}_m}}, \end{aligned}$$

where  $\beta_k \in \mathbb{R}^{d+1}$  is the coefficients corresponding to the  $k$ 'th class

# Multinomial Logistic regression

- The resulting multinomial logistic regression scheme assigns the following probabilities to class  $k \leq K - 2$

$$\mathbb{P}[Y_m = k | \boldsymbol{\theta}, \mathbf{x}_m] = \frac{e^{-\boldsymbol{\beta}_k^T \mathbf{x}_m}}{1 + \sum_{k=0}^{K-2} e^{-\boldsymbol{\beta}_k^T \mathbf{x}_m}} \quad (14)$$

- The set of parameters  $\boldsymbol{\theta} \in \mathbb{R}^{K-1 \times d+1}$  becomes a matrix of *weights* defining the mapping to each outcome class
- For the Iris classification problem, the vector consists of three elements, where vector  $(1, 0, 0)$  is assigned to Setosa,  $(0, 1, 0)$  is assigned to the class Versicolor and  $(0, 0, 1)$  to Virginica

## Example: XOR problem

- Example that shows why an SLP may be insufficient for simple classification and gives rise more refined non-linear models
- Consider the XOR-problem (exclusive OR relation)
- In this problem the data is not linearly separable. We have input  $\mathbf{x} \in \mathbb{R}^2$  and a class  $y \in \{-1, 1\}$  as shown below.

$x_1$	$x_2$	$y$
-1	-1	1
-1	1	-1
1	-1	-1
1	1	1

- **Impossible to classify this using a Single Layer Perceptron**



# Building blocks of neural networks

Building blocks of neural networks

# Composition of simple functions

- SLP is only capable of computing binary functions
- It nevertheless acts as a motivating *structure* for non-linear parametrizations
- Linear combinations of the input are passed to a non-linear function

## Idea

Generalize this structure: build non-linear parametrizations as **hierarchical compositions** of non-linear and linear transformations → **neural networks**

# Building blocks of composition: affine transformations

- Recall: in OLS  $\hat{y} = \beta^T \mathbf{x}$  where  $\beta \in \mathbb{R}^{d+1}$  is the parameter vector
- In the context of neural networks, affine mappings of the form

$$\mathbb{R}^d \ni \mathbf{x} \mapsto \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)} =: \mathbf{z}^{(1)} \in \mathbb{R}^p \quad (15)$$

are called the **linear layers**, which map their input  $\mathbf{x}$  to an affine combination with **weights**  $\mathbf{W}^{(1)} \in \mathbb{R}^{p \times d}$  and **biases**  $\mathbf{b}^{(1)} \in \mathbb{R}^p$

- The total number of parameters is  $\theta^{(1)} = (\mathbf{W}^{(1)}, \mathbf{b}^{(1)}) \in \mathbb{R}^{p \times (d+1)}$
- The length of  $\mathbf{z}^{(1)}$  is called the **width** of the layer
- The components of  $\mathbf{z}^{(1)}$  are called **neurons**
- The number of neurons in a hidden layer is a *hyperparameter*, the wider it is set, the more parameters there are

# Building blocks of composition: hidden layers

- The machine learning community often confuses the terms *linear* and *affine*. This is due to the fact that the constant bias (similar as in OLS or SLP) can be viewed as an additional constant dimension with weight vector  $\mathbf{b}$ .
- So far this is identical to the model of (vector-valued) OLS with *cast*:  $\hat{\mathbf{y}} \leftarrow \mathbf{z}^{(1)}$  and  $\beta \leftarrow \theta^{(1)}$
- Recall idea: apply a (non-)linear vector-valued mapping to the affine combinations of the input  $\mathbf{w}^T \phi(\mathbf{z}^{(1)} | \theta^{(1)})$

$$\mathbb{R}^{p_1} \ni \mathbf{z}^{(1)} \mapsto \left( \varphi_1^{(1)}(z_1^{(1)}), \dots, \varphi_p^{(1)}(z_p^{(1)}) \right) =: \mathbf{a}^{(1)} \in \mathbb{R}^{p_1},$$

where  $\varphi_i^{(1)} : \mathbb{R} \rightarrow \mathbb{R}$  are non-linear **activations**

- The composition of these transformations is called a **hidden layer** of a neural network

$$\mathbf{x} \mapsto \varphi_i^{(1)} \left( \sum_{j=1}^d W_{ij}^{(1)} x_j + b_i^{(1)} \right) =: a_i^{(1)}, \forall i = 1, \dots, p_1 \quad (16)$$

# Building blocks of composition: output layer

- In above,  $p$  is a *hyperparameter* which, similarly to OLS, can be interpreted as the number of *basis functions*  $\{x_1, \dots, x_d\}$  in a regression
- In order to approximate vector valued functions  $H : \mathbb{R}^d \rightarrow \mathbb{R}^q$ , one needs to transform the mapping of the hidden layer to the appropriate dimension of the *output*
- As before, we do this by an affine combination of the output with another set of *weights* and *biases*

$$\mathbb{R}^{p_1} \ni \mathbf{a}^{(1)} \mapsto \mathbf{W}^{(2)} \mathbf{a}^{(1)} + \mathbf{b}^{(2)} =: \mathbf{z}^{(2)} \in \mathbb{R}^q \quad (17)$$

with  $\mathbf{W}^{(2)} \in \mathbb{R}^{q \times p_1}$ ,  $\mathbf{b}^{(2)} \in \mathbb{R}^q$ ,  
 $\boldsymbol{\theta}^{(2)} := (\mathbf{W}^{(2)}, \mathbf{b}^{(2)}) \in \mathbb{R}^{q \times (p_1 + 1)}$

- Finally, in order to support the most general (bounded, binary, ...) output, the **output layer** is a composition

$$\mathbb{R}^{p_1} \ni \mathbf{a}^{(1)} \mapsto \left( \varphi_1^{(2)}(z_1^{(2)}), \dots, \varphi_q^{(2)}(z_q^{(2)}) \right) =: \mathbf{a}^{(2)} \in \mathbb{R}^q$$

# (Fully-Connected) Feedforward Artificial Neural Networks

- A **fully-connected, feedforward artificial neural network** is simply a **hierarchical composition** of the transformations above, mapping an input  $\mathbf{x} \in \mathbb{R}^d$  to the output  $\Phi \in \mathbb{R}^q$  whose  $i$ 'th coordinate reads as follows

$$\Phi_i(\mathbf{x}|\Theta) := \varphi_i^{(2)} \left( \sum_{j=1}^{p_1} W_{ij}^{(2)} \varphi_j^{(1)} \left( \sum_{k=1}^d W_{jk}^{(1)} x_k + b_j^{(1)} \right) + b_i^{(2)} \right) \quad (18)$$

where  $\Theta := (\boldsymbol{\theta}^{(1)}, \boldsymbol{\theta}^{(2)}) := (\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \mathbf{W}^{(2)}, \mathbf{b}^{(2)}) \in \mathbb{R}^{p_1 \times (d+1) + q \times (p_1+1)}$

- The total number of parameters in the statistical model is

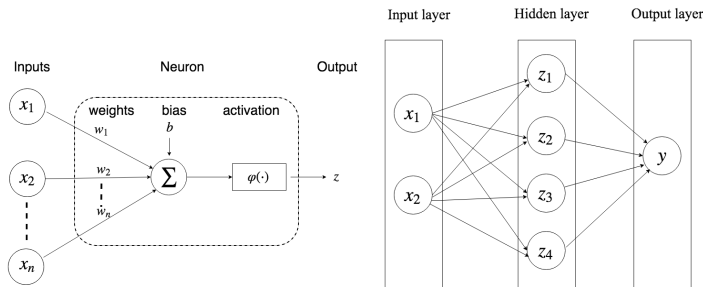
$$p = p_1 \times (d + 1) + q \times (p_1 + 1) \quad (19)$$

# (Fully-Connected) Feedforward Artificial Neural Networks

- Denoting the *element-wise* non-linearities by  $\varphi^{(n)}(\mathbf{z}^{(n)}) := (\varphi_1^{(n)}(z_1^{(n)}), \dots, \varphi_{p_n}^{(n)}(z_{p_n}^{(n)})) \in \mathbb{R}^{p_n}$ , in vector notation this can be written as follows

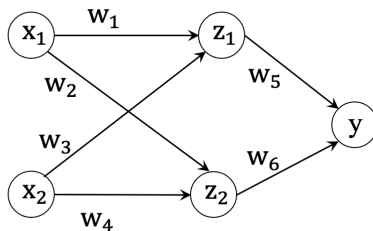
$$\Phi(\mathbf{x}|\Theta) := \varphi^{(2)} \circ \mathbf{z}^{(2)}(\cdot|\theta^{(2)}) \circ \varphi^{(1)} \circ \mathbf{z}^{(1)}(\mathbf{x}|\theta^{(1)}) \quad (20)$$

- FCFF ANNs can be thought of as **directed, acyclic graphs**



## Example: XOR revisited

- Unlike with SLP, adding a *hidden layer*, and parametrizing with ANNs solves the problem



- $x_1$  and  $x_2$  are first mapped to intermediate binary values  $z_1 = h_1(w_1x_1 + w_3x_2)$  and  $z_2 = h_2(w_2x_1 + w_4x_2)$ , where  $h_{1,2}(\cdot)$  is a perceptron with  $b = 0$ .



## Example: XOR revisited

- Using  $w_1 = -1$ ,  $w_2 = 1$ ,  $w_3 = 1$ ,  $w_4 = -1$ , the outputs are shown in columns  $z_1$  and  $z_2$ .

$x_1$	$x_2$	$z_1$	$z_2$	$y$
-1	-1	1	1	1
-1	1	1	-1	-1
1	-1	-1	1	-1
1	1	-1	-1	1

- It remains to compute  $h(w_5 z_1 + w_6 z_2)$ , by setting threshold  $b = 1$  and  $w_5 = 1$ ,  $w_6 = 1$  the desired output is obtained in  $y$  and the XOR-problem is solved.

Activations: the key feature of non-linearity

# Activation Functions

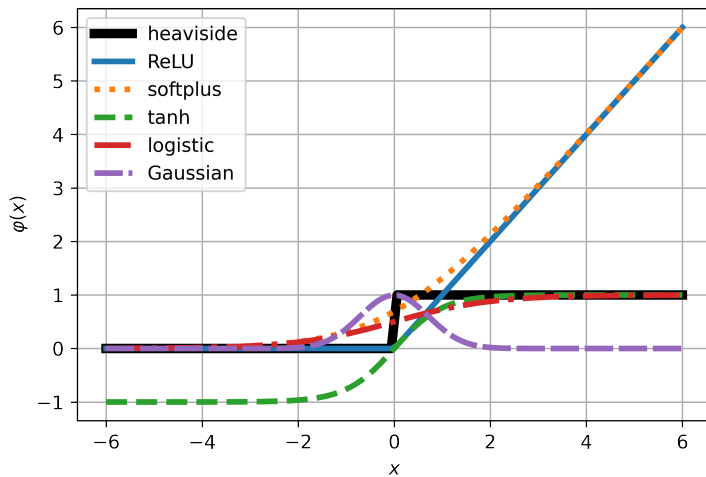
- The key component of *non-linearity* is given by the **activation functions**  $\varphi_j^{(n)} : \mathbb{R} \rightarrow \mathbb{R}, j = 1, \dots, p_n$
- For SLP we had  $\varphi_1^{(1)}(x) = \text{sgn}(x)$
- Most often, application functions within the same layer are chosen the same  $\varphi_1^{(n)}(x) = \dots = \varphi_{p_n}^{(n)}(x)$
- In regression applications the output activations are usually chosen to be the identity function  $\varphi^{(2)}(\mathbf{z}^{(2)}) = \mathbf{z}^{(2)}$
- The parametrization inherits **continuity** and **differentiability** properties from the chosen activations
- The non-linear activation functions are *hyperparameters* of the statistical model, appropriate choices vary depending on the problem

# Activation Functions

name	$\varphi(x)$	$\varphi'(x)$	range	continuity
heaviside	$\begin{cases} 0, & x \leq 0 \\ 1, & x > 0 \end{cases}$	$\begin{cases} 0, & x \neq 0 \\ \text{undefined}, & x = 0 \end{cases}$	$\{0, 1\}$	$C^{-1}$
ReLU	$\begin{cases} 0, & x \leq 0 \\ x, & x > 0 \end{cases}$	$\begin{cases} 0, & x < 0 \\ x, & x > 0 \\ \text{undefined}, & x = 0 \end{cases}$	$[0, \infty)$	$C^0$
Gaussian	$e^{-x^2}$	$-2xe^{-x^2}$	$(0, 1]$	$C^\infty$
tanh	$\frac{e^x - e^{-x}}{e^x + e^{-x}}$	$1 - \tanh^2(x)$	$(-1, 1)$	$C^\infty$
logistic	$\frac{1}{1 + e^{-x}}$	$\frac{e^{-x}}{1 + e^{-x}}$	$(0, 1)$	$C^\infty$
softplus	$\log(1 + e^x)$	$\frac{1}{1 + e^{-x}}$	$(0, \infty)$	$C^\infty$
...	...	...	...	...

See wiki

# Activation Functions



# Layer dependent activations

- For classification problems it is chosen such that  $\sum_{k=1}^q \varphi_k^{(2)}(z_k^{(2)}) = 1$
- In order to ensure that output *probabilities* indeed sum to one, activations are chosen such that they are not a collection of scalar-valued mappings of neurons but vector-valued functions depending on the whole layer  $\varphi^{(n)} : \mathbb{R}^{p_n} \rightarrow \mathbb{R}^{p_n}$

name	$\varphi_i(\mathbf{x})$	$\partial_j \varphi_i(\mathbf{x})$	range	cont.
softmax	$\frac{e^{x_i}}{\sum_{j=1}^{p_n} e^{x_j}}$	$\varphi_i(\mathbf{x})(\delta_{ij} - \varphi_j(\mathbf{x}))$	$(0, 1)$	$C^\infty$
maxout	$\max_{i=1, \dots, p_n} x_i$	$\begin{cases} 1, & j = \arg \max_i x_i \\ 0, & j \neq \arg \max_i x_i \end{cases}$	$(-\infty, \infty)$	$C^0$
...	...	...	...	...

## Deep Neural Networks

# Fully-Connected Feedforward Deep Neural Networks

- So far: neural network = input + hidden + output layer
- These are called **shallow** feedforward ANNs

$$\Phi(\mathbf{x}|\Theta) := \varphi^{(2)} \circ \mathbf{z}^{(2)}(\cdot|\theta^{(2)}) \circ \varphi^{(1)} \circ \mathbf{z}^{(1)}(\mathbf{x}|\theta^{(1)})$$

- One can extend the model to allow for  $L$ -many hidden layers in the composition, with corresponding widths  $p_l, l = 1 \dots, L$
- The resulting, **deep neural network** mapping reads as follows

$$\Psi(\mathbf{x}|\Theta) := \varphi^{(L+1)} \circ \mathbf{z}^{(L+1)}(\cdot|\theta^{(L+1)}) \circ \dots \circ \varphi^{(1)} \circ \mathbf{z}^{(1)}(\mathbf{x}|\theta^{(1)}), \quad (21)$$

where  $\Theta := (\theta^{(1)}, \dots, \theta^{(L+1)}) \in \mathbb{R}^p$

- The total number of parameters in the statistical model rapidly increases

$$p = d \times (p_1 + 1) + \sum_{l=1}^{L-1} p_l \times (p_{l+1} + 1) + p_L \times (q + 1) \quad (22)$$



# Fully-Connected Feedforward Deep Neural Networks

$$\Psi(\mathbf{x}|\Theta) := \varphi^{(L+1)} \circ \mathbf{z}^{(L+1)}(\cdot|\theta^{(L+1)}) \circ \dots \circ \varphi^{(1)} \circ \mathbf{z}^{(1)}(\mathbf{x}|\theta^{(1)}),$$

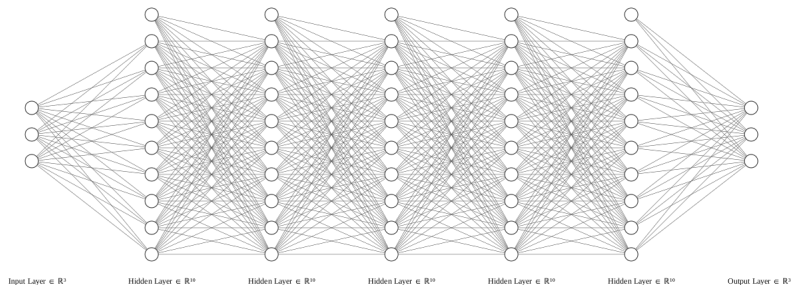


Figure: Illustration DNN, source: NN-SVG

$$p = d \times (p_1 + 1) + \sum_{l=1}^{L-1} p_l \times (p_{l+1} + 1) + p_L \times (q + 1)$$

Teaser: capacity and optimization

# What we've seen so far

- We gathered **one potential** parametrization for non-linear problems via (deep) neural networks:  $\Phi(\mathbf{x}|\Theta) : \mathbb{R}^d \rightarrow \mathbb{R}^q$
- Similarly to MSE in OLS, one can define a *performance measure*  $L(\Theta|\mathbf{x})$  which quantifies the "goodness" of a certain parameter set, under measurements  $\mathbf{x}$
- Such **losses** can be of various forms
  - $L(\Theta|\mathbf{x}) = \mathbb{E}[d(\Phi(\mathbf{x}|\Theta), H(\mathbf{x}))]$  (supervised learning)
  - $L(\Theta|\mathbf{x}) = \mathbb{E}[d(F(\Phi(\mathbf{x}|\Theta)), 0)]$  (unsupervised learning)

where  $d$  is a desired *distance function* – such as the ones induce by  $L^p$  norms  $1 \leq p < \infty$

$$d(f, g) = \|f - g\|_p := \sqrt[p]{\int_{\mathbf{x}} |f(\mathbf{x}) - g(\mathbf{x})|^p d\mu(\mathbf{x})}$$

# Questions remaining

- DNNs provide a certain family of statistical models **how do we know what problems are they also appropriate for?**
- In order to find a "good" statistical model we want to minimize the loss function

$$\arg \min_{\theta} L(\theta|\mathbf{x}),$$

- Unlike in case of OLS, the MSE of supervised, neural network regression is no longer *convex* in  $\Theta \mapsto L(\Theta|\mathbf{x}) \rightarrow$  **no closed-form minimizer**

## Next week

- 1 **Universal Approximation Theorem (UAT):** ANNs are *dense* in a very wide class of target functions  $\mathcal{H}$
- 2 **Stochastic Gradient Descent (SGD):** an empirically successful iterative optimization method