FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG
NATURWISSENSCHAFTLICHE FAKULTÄT

Linear Models for Regression and Classification

Lecture “Mathematical Data Science” 2021/2022

Frauke Liers
Friedrich-Alexander-Universität Erlangen-Nürnberg
24.11.2021

DISCRETE OPTIMIZATION

L_1 Loss Function Instead of Least Squares in Linear Models

Instead of L_2 loss, consider $L_1 : E(\mid Y - f(X) \mid)$.
Minimizing w.r.t. L_1 yields the (conditional) mean $\hat{f}(x) = \text{median}(Y \mid X = x)$
(recall: median $\tilde{x}$ satisfies $\sum_{i=1}^{n} |\tilde{x} - x_i| \leq \sum_{i=1}^{n} |x - x_i|$) L1 Norm is more robust to outliers.

L_1 Loss Function Instead of Least Squares in Linear Models

Instead of L_2 loss, consider $L_1 : E(\mid Y - f(X) \mid)$.
Minimizing w.r.t. L_1 yields the (conditional) mean $\hat{f}(x) = \text{median}(Y \mid X = x)$
(recall: median $\tilde{x}$ satisfies $\sum_{i=1}^{n} |\tilde{x} - x_i| \leq \sum_{i=1}^{n} |x - x_i|$)

- advantage: estimates are more robust than with L_2 least squares.
- However, L_1 is non-smooth, has discontinuities in derivatives, thus is not so widespread. It is a non-convex function hence can stuck in local minima but can use smoothing to reduce this discontinuity.
- However, we will use L_1-norm from time to time if appropriate.

Categorial Variables: Loss Function and k-Nearest Neighbors

Assume we have categorial targets instead of numerical ones.
Let different classes be denoted by $\mathcal{G}$, and $K = \text{card}(\mathcal{G})$.

Categorial Variables: Loss Function and k-Nearest Neighbors

Assume we have categorial targets instead of numerical ones.
Let different classes be denoted by $\mathcal{G}$, and $K = \text{card}(\mathcal{G})$.

- define $K \times K$ matrix $\mathbf{L}$, where entry $L_{k,l}$ denotes price for classifying an observation that belongs to $\mathcal{G}_k$ as observation $\mathcal{G}_l$.
- $L_{k,k} = 0 \ \forall k = 1, \ldots, K$

Categorial Variables: Loss Function and k-Nearest Neighbors

Assume we have categorial targets instead of numerical ones.
Let different classes be denoted by $\mathcal{G}$, and $K = \text{card}(\mathcal{G})$.

- define $K \times K$ matrix $\mathbf{L}$, where entry $L_{k,l}$ denotes price for classifying an observation that belongs to $\mathcal{G}_k$ as observation $\mathcal{G}_l$.
- $L_{k,k} = 0 \ \forall k = 1, \ldots, K$
- *Zero-one loss fuction*: equal prize for all misclassifications: $L_{k,l} = 1 \ \forall k, l = 1, \ldots, K$, where $k \neq l$

Categorial Variables: Loss Function and k-Nearest Neighbors

Analogous computations as last week:

- We want to assign classes such that expected predicted error EPE is minimized, now measured in L_1 norm.
- $EPE = E(L(G, \hat{G}(X)))$, where $\hat{G}$ we want to assign. Expectation is taken w.r.t. joint distribution $Pr(G, X)$.
- see last week: $EPE = E_X \sum_{k=1}^{K} L[\mathcal{G}_k, \hat{G}(X)] Pr(\mathcal{G}_k \mid X))$
- pointwise minimization for X leads to $\hat{G}(x) = \text{argmin}_{g \in \mathcal{G}} \sum_{k=1}^{K} L(\mathcal{G}_k, g) Pr(\mathcal{G}_k \mid X = x))$

Categorial Variables: Loss Function and k-Nearest Neighbors

Analogous computations as last week:

- We want to assign classes such that expected predicted error EPE is minimized, now measured in L_1 norm.
- $EPE = E(L(G, \hat{G}(X)))$, where $\hat{G}$ we want to assign. Expectation is taken w.r.t. joint distribution $Pr(G, X)$.
- see last week: $EPE = E_X \sum_{k=1}^{K} L[\mathcal{G}_k, \hat{G}(X)]Pr(\mathcal{G}_k \mid X))$
- pointwise minimization for X leads to $\hat{G}(x) = \text{argmin}_{g \in \mathcal{G}} \sum_{k=1}^{K} L(\mathcal{G}_k, g)Pr(\mathcal{G}_k \mid X = x))$
- using $0 - 1$ loss, this means: we want to minimize expected prize for misclassification, i.e., maximize the expected probability for correct classification

$$\hat{G}(x) = \mathcal{G}_k \text{ if } Pr(\mathcal{G}_k \mid X = x) = \max_{g \in \mathcal{G}} Pr(g \mid X = x)$$

I.e., assign the class that maximizes joint distribution $Pr(x \mid X = x)$

Bayes Classifier

This choice is known as *Bayes classifier*, namely: we classify according to most probable class, using conditional discrete distribution $Pr(G \mid X)$.

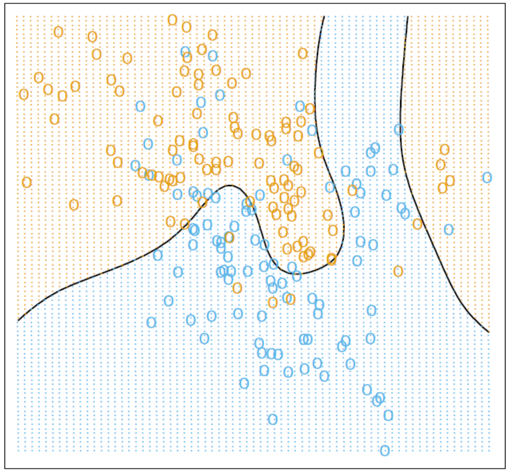


FIGURE 2.5. *The optimal Bayes decision boundary for the simulation example of Figures 2.1, 2.2 and 2.3. Since the generating density is known for each class, this boundary can be calculated exactly (Exercise 2.2).*

k-nearest neighbor does *majority vote* in chosen *k*-neighborhood, i.e., it assigns a class that the majority of points in neighborhood has.

Bayes Classifier

This choice is known as *Bayes classifier*, namely: we classify according to most probable class, using conditional discrete distribution $Pr(G \mid X)$.

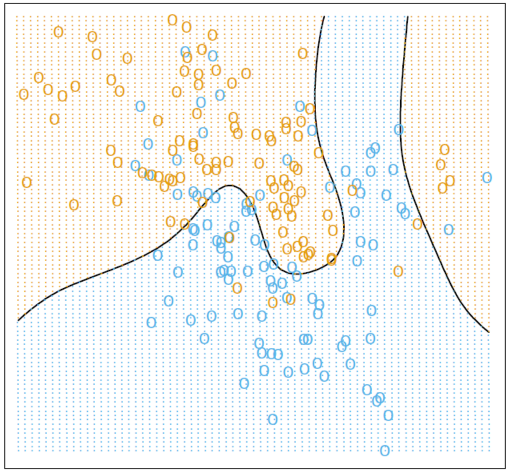


FIGURE 2.5. *The optimal Bayes decision boundary for the simulation example of Figures 2.1, 2.2 and 2.3. Since the generating density is known for each class, this boundary can be calculated exactly (Exercise 2.2).*

k-nearest neighbor does *majority vote* in chosen k-neighborhood, i.e., it assigns a class that the majority of points in neighborhood has.
We see: k-nearest neighbor *approximates EPE solution*

- conditional probability is restricted to neighborhood
- probabilities are estimated on training set.

Shrinkage Methods in Linear Models: Ridge Regression

High variabilities in regression results may occur. More stable methods additionally use some size reduction in the regression coefficients (see also regularization example in polynomial fit)

Shrinkage Methods in Linear Models: Ridge Regression

High variabilities in regression results may occur. More stable methods additionally use some size reduction in the regression coefficients (see also regularization example in polynomial fit)
Let us define least-square miminization problem for determining the best coefficients:

$$\hat{\beta}^{\text{ridge}} = \text{argmin}_{\beta}\{\sum_{i=1}^{N}(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2\}. \tag{1}$$

λ: complexity parameter, controls the amount of shrinking.

Shrinkage Methods in Linear Models: Ridge Regression

High variabilities in regression results may occur. More stable methods additionally use some size reduction in the regression coefficients (see also regularization example in polynomial fit)
Let us define least-square miminization problem for determining the best coefficients:

$$\hat{\beta}^{\text{ridge}} = \text{argmin}_{\beta}\{\sum_{i=1}^{N}(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2\}. \tag{1}$$

λ: complexity parameter, controls the amount of shrinking.
Determine minimizer $\hat{\beta}^{\text{ridge}}$:
Writte argument from (1) in matrix form (wlog $\beta_0 = 0$):

$$RSS(\lambda) = (\mathbf{y} - \mathbf{X}\beta)^\top(\mathbf{y} - \mathbf{X}\beta) + \lambda\beta^\top\beta$$

Alternative Derivation of Ridge Regression

$$RSS(\lambda) = (\mathbf{y} - \mathbf{X}\beta)^\top(\mathbf{y} - \mathbf{X}\beta) + \lambda\beta^\top\beta$$

best possible solution is given for

$$\hat{\beta}^{\text{ridge}} = (\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^\top\mathbf{y}$$

with $p \times p$ identity matrix $\mathbf{I}$. (please check...!)
If quadratic penalty $\beta^\top\beta$ is used, ridge regression solution is again linear in $\mathbf{y}$.
(compare with last week's formula for linear regression: $\hat{\beta} = (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top y$.)

Alternative Derivation of Ridge Regression

$$RSS(\lambda) = (\mathbf{y} - \mathbf{X}\beta)^\top(\mathbf{y} - \mathbf{X}\beta) + \lambda\beta^\top\beta$$

best possible solution is given for

$$\hat{\beta}^{\text{ridge}} = (\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^\top\mathbf{y}$$

with $p \times p$ identity matrix $\mathbf{I}$. (please check...!)
If quadratic penalty $\beta^\top\beta$ is used, ridge regression solution is again linear in $\mathbf{y}$.
(compare with last week's formula for linear regression: $\hat{\beta} = (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top y$.)
We see:

- Before inversion, positive constant $\beta^\top\beta$ is added on diagonal of $\mathbf{X}^\top\mathbf{X}$
- this makes resulting matrix *nonsingular*, even if $\mathbf{X}^\top\mathbf{X}$ itself was singular. This is an advantage!

Brief Repetition Linear Algebra

Recall from linear algebra: *singular value decomposition (SVD)*
is factorization of a real or complex matrix, generalizes eigen decomposition.
SVD of a (not necessarily symmetric) matrix $\mathbf{M} \in \mathbb{R}^{m \times n}$ is a factorization of form
$\mathbf{M} = \mathbf{UDV}^{\star}$, where

- $\mathbf{U}$ is $m \times m$ complex unitary matrix
- $\mathbf{D} \in \mathbb{R}^{m \times n}$ rectangular matrix with non-negative real numbers on the diagonal, otherwise zeros.
- $\mathbf{V}$ is $n \times n$ complex unitary matrix.
- for real matrix: $\mathbf{M} = \mathbf{UDV}$ with real orthonormal $\mathbf{U}, \mathbf{V}$.
- diagonal entries are called singular values
- SVD is not unique

Alternative Derivation of Ridge Regression

SVD of an $N \times p$ matrix $\mathbf{X}$ is of form $\mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{V}^\top$, where $\mathbf{U} \in \mathbb{R}^{N \times p}$, $\mathbf{V} \in \mathbb{R}^{p \times p}$ orthonormal matrics, columns of $\mathbf{U}$ span column space of $\mathbf{X}$, columns of $\mathbf{V}$ span row space.
$\mathbf{D} \in \mathbb{R}^{p \times p}$ diagonal matrix with entries $d_1 \geq d_2 \geq \ldots \geq d_p \geq 0$ *singular values* of $\mathbf{X}$.
Ridge solutions:

$$\mathbf{X}\hat{\beta}^{\text{ridge}} = \mathbf{X}(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^\top\mathbf{y} = \mathbf{U}\mathbf{D}(\mathbf{D}^2 + \lambda\mathbf{I})^{-1}\mathbf{D}\mathbf{U}^\top\mathbf{y} = \sum_{i=1}^{p} \mathbf{u}_j \frac{d_j^2}{d_j^2 + \lambda}\mathbf{u}_j^\top\mathbf{y},$$

where $\mathbf{u}_j$ are columns of $\mathbf{U}$.
(This calculation is easy to verify, please double-check...)
We have $\lambda \geq 0$, thus $\frac{d_j^2}{d_j^2+\lambda} \leq 1$.
We see from formula: ridge regression computes coordinates of $\mathbf{y}$ with respect to orthonormal basis $\mathbf{U}$.

Ridge Regression and PCA

This indicates that dividing ny dj + lambda will reduce the weight value.

Then it shrinks coordinates by factor $\frac{d_j^2}{d_j^2+\lambda} \leq 1$ Thus: more shrinking if a coordinate has a basis vector with small d_j^2, when compared to a coordinate with large d_j^2.

Ridge Regression and PCA

Then it shrinks coordinates by factor $\frac{d_j^2}{d_j^2+\lambda} \leq 1$ Thus: more shrinking if a coordinate has a basis vector with small d_j^2, when compared to a coordinate with large d_j^2.
Meaning of this: 'unimportant' columns are shrunk stronger.

Ridge Regression and PCA

Then it shrinks coordinates by factor $\frac{d_j^2}{d_j^2+\lambda} \leq 1$ Thus: more shrinking if a coordinate has a basis vector with small d_j^2, when compared to a coordinate with large d_j^2.
Meaning of this: 'unimportant' columns are shrunk stronger.
Additional observation: SVD of $\mathbf{X}$ can express the *principal components* of $\mathbf{X}$, because:

Ridge Regression and PCA

Then it shrinks coordinates by factor $\frac{d_j^2}{d_j^2+\lambda} \leq 1$ Thus: more shrinking if a coordinate has a basis vector with small d_j^2, when compared to a coordinate with large d_j^2.
Meaning of this: 'unimportant' columns are shrunk stronger.
Additional observation: SVD of $\mathbf{X}$ can express the *principal components* of $\mathbf{X}$, because: covariance matrix from test set is $\mathbf{S} = \frac{\mathbf{X}^\top \mathbf{X}}{N}$

Ridge Regression and PCA

Then it shrinks coordinates by factor $\frac{d_j^2}{d_j^2+\lambda} \leq 1$ Thus: more shrinking if a coordinate has a basis vector with small d_j^2, when compared to a coordinate with large d_j^2.
Meaning of this: 'unimportant' columns are shrunk stronger.
Additional observation: SVD of $\mathbf{X}$ can express the *principal components* of $\mathbf{X}$, because: covariance matrix from test set is $\mathbf{S} = \frac{\mathbf{X}^\top \mathbf{X}}{N}$
$\mathbf{X}^\top \mathbf{X} = (\mathbf{V}^\top \mathbf{D}^\top \mathbf{U}^\top)\mathbf{UDV} = \mathbf{UD}^2\mathbf{U}^\top$ is *eigen decomposition* of $\mathbf{X}^\top \mathbf{X}$ and of $\mathbf{S}$.

Ridge Regression and PCA

Then it shrinks coordinates by factor $\frac{d_j^2}{d_j^2+\lambda} \leq 1$ Thus: more shrinking if a coordinate has a basis vector with small d_j^2, when compared to a coordinate with large d_j^2.
Meaning of this: 'unimportant' columns are shrunk stronger.
Additional observation: SVD of $\mathbf{X}$ can express the *principal components* of $\mathbf{X}$, because: covariance matrix from test set is $\mathbf{S} = \frac{\mathbf{X}^\top \mathbf{X}}{N}$
$\mathbf{X}^\top \mathbf{X} = (\mathbf{V}^\top \mathbf{D}^\top \mathbf{U}^\top)\mathbf{UDV} = \mathbf{UD}^2\mathbf{U}^\top$ is *eigen decomposition* of $\mathbf{X}^\top \mathbf{X}$ and of $\mathbf{S}$.
Recall: eigenvectors v_j are principal components of $\mathbf{X}$.
First eigenvalue direction v_1 leads to first principal component $\mathbf{z}_1 = \mathbf{X}v_1 = \mathbf{u}_1 d_1$.
Therefore: $\mathbf{u}_1$ is normalized first principal component, etc.
Property: first principal component has largest sample variance:
$Var(\mathbf{z}_1) = \frac{d_1^2}{N}$

Ridge Regression and PCA

Then it shrinks coordinates by factor $\frac{d_j^2}{d_j^2+\lambda} \leq 1$ Thus: more shrinking if a coordinate has a basis vector with small d_j^2, when compared to a coordinate with large d_j^2.
Meaning of this: 'unimportant' columns are shrunk stronger.
Additional observation: SVD of $\mathbf{X}$ can express the *principal components* of $\mathbf{X}$, because: covariance matrix from test set is $\mathbf{S} = \frac{\mathbf{X}^\top \mathbf{X}}{N}$
$\mathbf{X}^\top \mathbf{X} = (\mathbf{V}^\top \mathbf{D}^\top \mathbf{U}^\top)\mathbf{UDV} = \mathbf{UD}^2\mathbf{U}^\top$ is *eigen decomposition* of $\mathbf{X}^\top \mathbf{X}$ and of $\mathbf{S}$.
Recall: eigenvectors v_j are principal components of $\mathbf{X}$.
First eigenvalue direction v_1 leads to first principal component $\mathbf{z}_1 = \mathbf{X}v_1 = \mathbf{u}_1 d_1$.
Therefore: $\mathbf{u}_1$ is normalized first principal component, etc.
Property: first principal component has largest sample variance:
$Var(\mathbf{z}_1) = \frac{d_1^2}{N}$ in this order of components, variance gets smaller, last principal component has minimum variance.
Ridge regression shrinks these directions most.

Alternative Shrinking Model

Alternative (and potentially more restrictive) way of reducing coefficient sizes:

$$\hat{\beta}^{\text{ridge}} = \text{argmin}_{\beta}\{\sum_{i=1}^{N}(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2), \text{ s.t. } \sum_{j=1}^{p}\beta_j^2 \leq t\}.$$

parameter t: chosen beforehand, restricts size of coefficients
β_0 is left out from shrinking, as otherwise procedure would depend on origin

The Lasso Regression

$$\hat{\beta}^{\text{lasso}} = \text{argmin}_{\beta}\{\sum_{i=1}^{N}(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2), \text{ s.t. } \sum_{j=1}^{p} |\beta_j| \leq t\}.$$

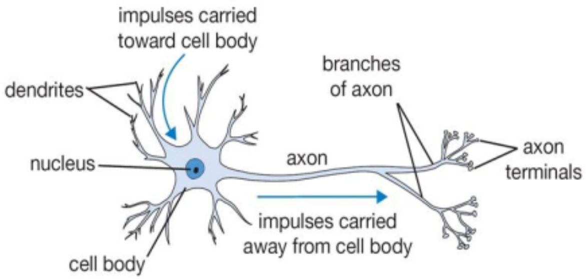
wlog, $\beta_0 = 0$ (after centralizing data)
Write it in Lagrangian form as

$$\hat{\beta}^{\text{lasso}} = \text{argmin}_{\beta}\{\frac{1}{2}\sum_{i=1}^{N}(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j|\}$$

.
quadratic ridge penalty $\lambda \sum_{j=1}^{p} \beta_j^2$ is replaced by L_1 penalty $\lambda \sum_{j=1}^{p} |\beta_j|$ which is nonlinear, however remains computationally tractable.

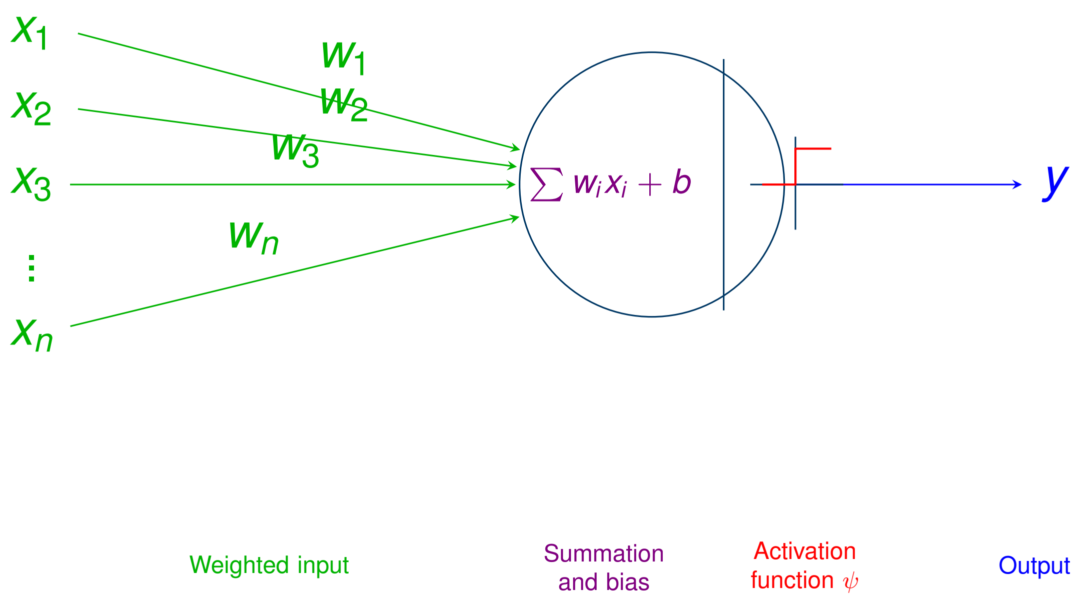
Next Chapter: Towards Artificial Neural Networks

Historic aim (McCulloch& Pitts, 1943):
Mimic the biological processes of real neurons for Machine Learning.



Key observation: Biological neurons transmit signals **only** if the required activation energy is reached by all incoming signals.

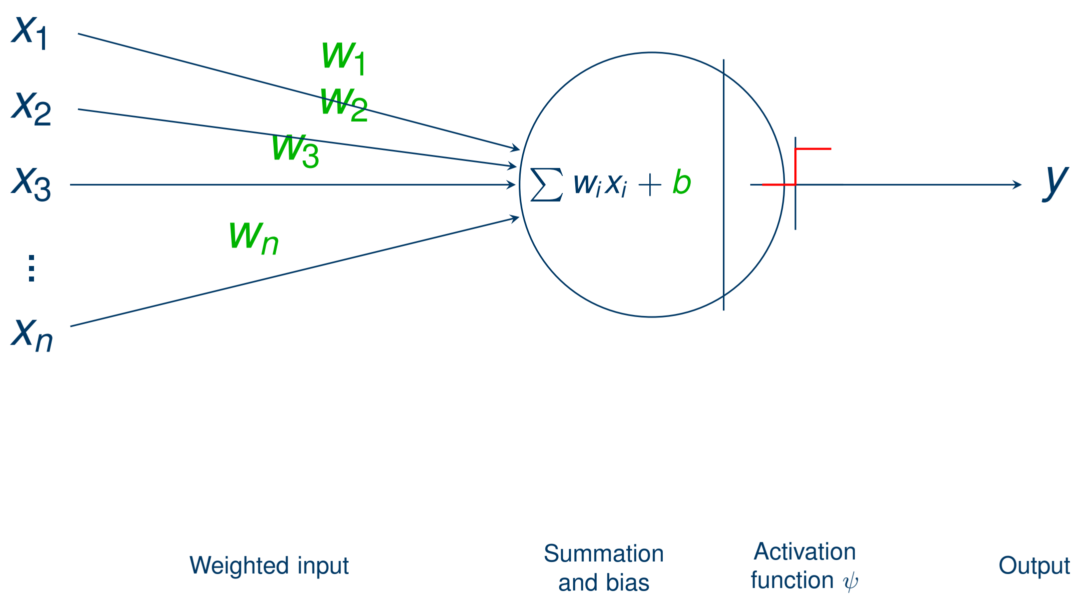
(Simple) Perceptron - an artificial neuron



The simple perceptron (Rosenblatt, 1958) is an **artificial neuron** that is able to compute mathematical functions of the form:

$$y = \psi\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

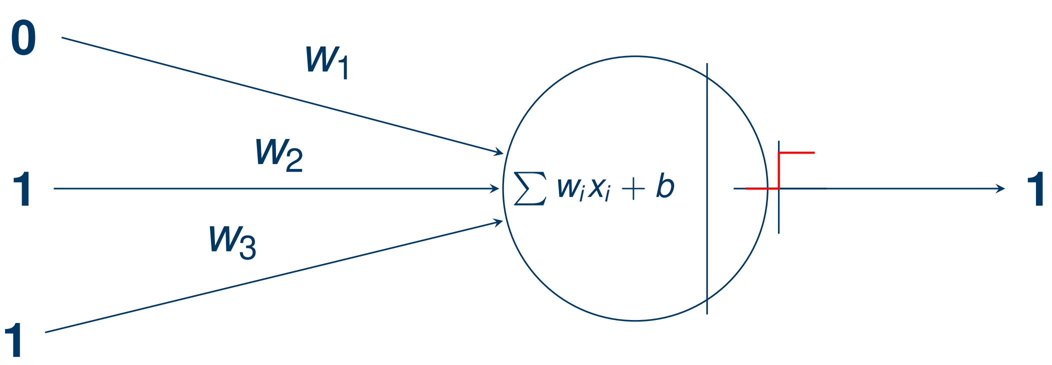
(Simple) Perceptron - an artificial neuron



For a fixed activation function $\psi\colon \mathbb{R} \to \mathbb{R}$ the behaviour of the perceptron is defined by the free parameters $(w_1, \ldots, w_n, b) = (\vec{w}, b) =: \theta \in \mathbb{R}^{n+1}$.
Thus, the perceptron realizes a parametrized map $f_\theta\colon \mathbb{R}^n \to \mathbb{R}$ with
$f_\theta(\vec{x}) := f(\vec{x}; \theta) = f(x_1, \ldots, x_n; \theta)$.

An example perceptron

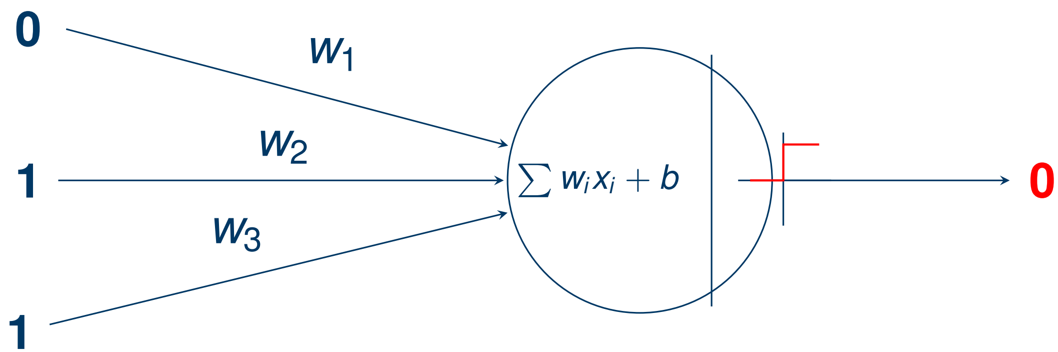
We analyze a perceptron with 3 fixed input signals $(x_1, x_2, x_3) = (0, 1, 1)$. We use the Heavyside function $H\colon \mathbb{R} \to \{0, 1\}$ as activation function ($H(z) = 0$ if $z < 0$, $H(z) = 1$ otherwise).
Set free parameters as $\theta = (w_1, w_2, w_3, b) = (1, 0, 1, -1)$.



Thus, we get $f_\theta(\vec{x}) = H([1, 0, 1] \cdot [0, 1, 1]^T - 1) = H(0) = 1$.

An example perceptron

We analyze a perceptron with 3 fixed input signals $(x_1, x_2, x_3) = (0, 1, 1)$. We use the Heavyside function $H\colon \mathbb{R} \to \{0, 1\}$ as activation function and set the free parameters as $\theta = (w_1, w_2, w_3, b) = (1, 0.5, -1, 0)$.

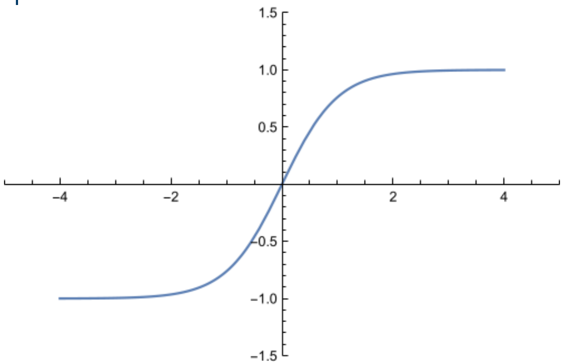


Thus, we get $f_\theta(\vec{x}) = H([1, 0.5, -1] \cdot [0, 1, 1]^T + 0) = H(-0.5) = 0$.

Continuous activation functions

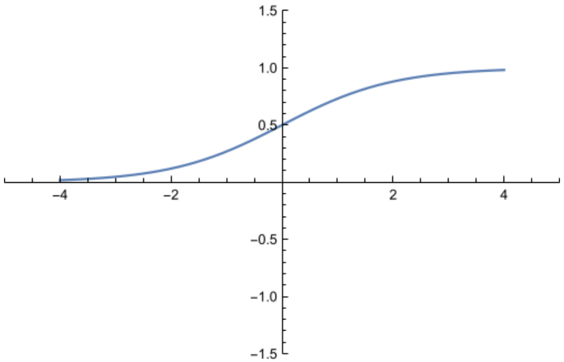
The following **continuous activation functions** are commonly used in artificial neurons due to their nice analytic properties:

Tanh



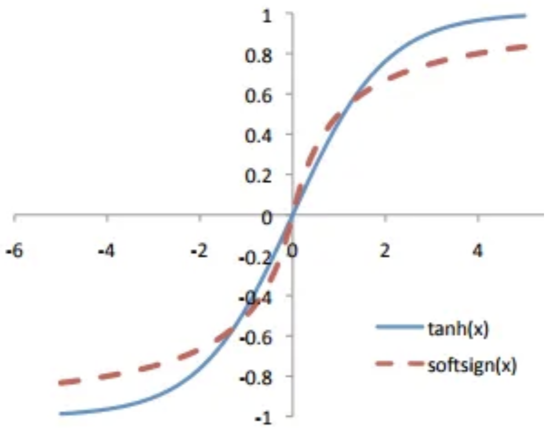
$$\psi(t) := \tanh(t)$$

Logistic



$$\psi(t) := \frac{1}{1 + e^{-t}}$$

Softsign



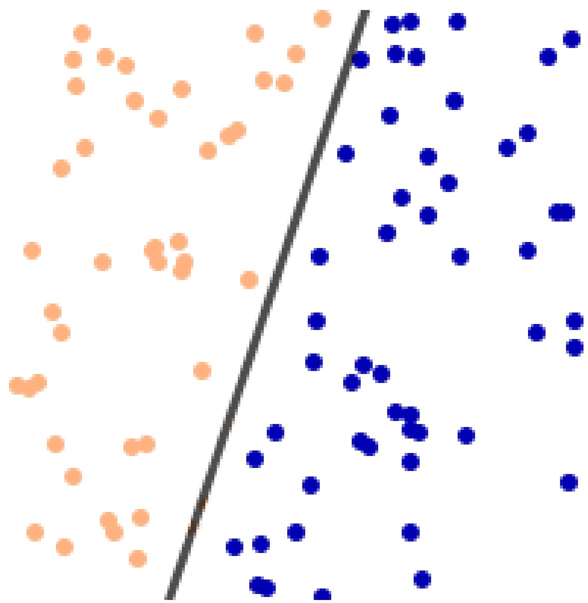
$$\psi(t) := \frac{t}{1 + |t|}$$

Observations on the perceptron

Observations:

- **weights** $\vec{w} = (w_1, \ldots, w_n)$ determine the influence of the input
  $\rightarrow$ weight $w_k = 0$ disregards respective input x_k completely
- **bias** b defines a base probability for activation of the artificial neuron
  $\rightarrow$ bias $b \ll 0$ makes an activation very unlikely
  $\rightarrow$ bias $b \gg 0$ makes an activation very likely
- simple perceptrons with Heavyside activation function realize **linear binary classifiers**
  $\rightarrow$ for more complex applications a perceptron is too restricted

Observations on the perceptron



Given a set of input data $\{\vec{x}^{(1)}, \dots, \vec{x}^{(N)}\}$ with $x^{(i)} \in \mathbb{R}^n$, the free parameters $\theta \in \mathbb{R}^{n+1}$ induce a hyperplane that **linearly** separates the data in two classes.

$$f_\theta(\vec{x}^{(i)}) := \begin{cases} 1, & \text{if } \langle \vec{w}, \vec{x}^{(i)} \rangle + b > 0, \\ 0, & \text{otherwise.} \end{cases}$$

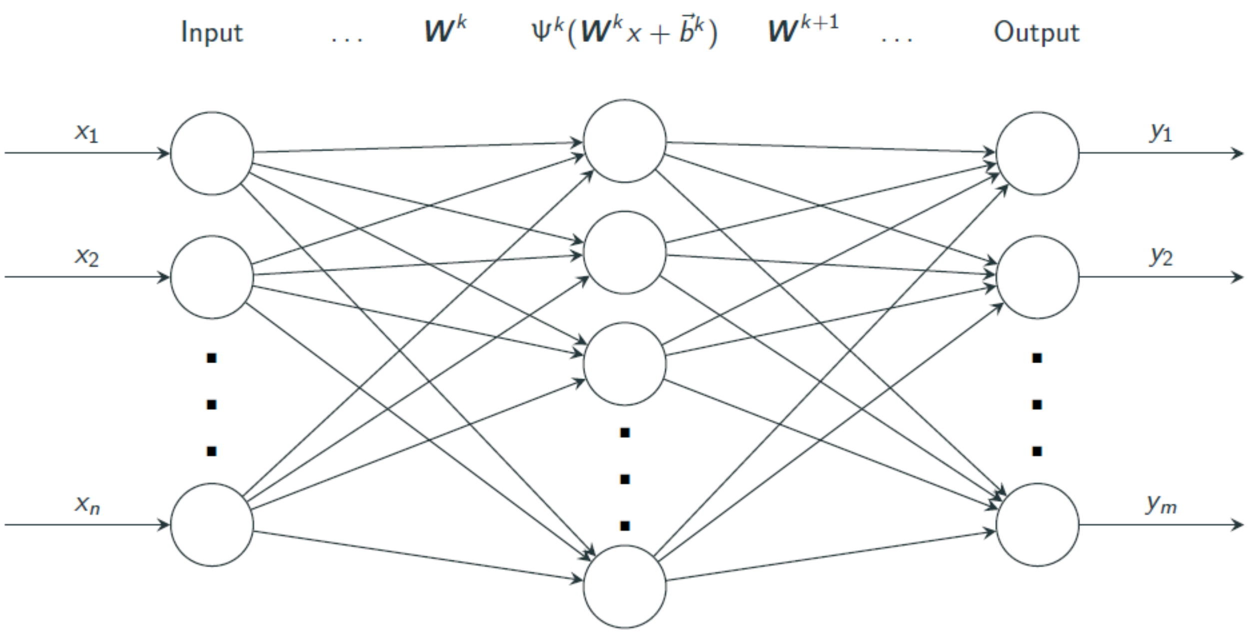
Artificial neural networks

Idea: Combine multiple perceptrons to perform more complex tasks.

- align artificial neurons in consecutive layers
  $\rightarrow$ convention: use designated input layer and output layer
  $\rightarrow$ all intermediate layers are called **hidden layer**
  $\rightarrow$ number of layers is called **depth** of the neural network
  $\rightarrow$ number of nonzero weights is called **connectivity** of the neural network
- artificial neural networks can be represented by directed graphs
- connections between neurons can be (almost) **arbitrary**
  $\rightarrow$ often there are no connections within same layer (except in recurrent neural networks)
  $\rightarrow$ certain network structures have proved to be successful for different applications, e.g., convolutional neural networks

Fully-connected feedforward neural network

Classical representation: Mappings from kth to $(k+1)$st layer:



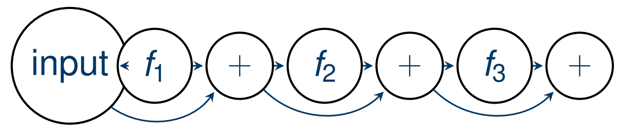
Compact representation:

Fully-connected feedforward neural network

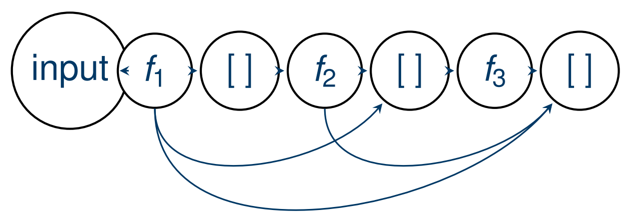
- a fully-connected feedforward neural network can be written as a parametrized map $f_\Theta \colon \mathbb{R}^n \to \mathbb{R}^m$ that is realized by a concatenation of $d \in \mathbb{N}$ perceptron layers via
$$f_\Theta := f^d_{\Theta_d} \circ \ldots \circ f^1_{\Theta_1}$$
- each layer is a map $f^k_{\Theta_k} \colon \mathbb{R}^{n_{k-1}} \to \mathbb{R}^{n_k}$ with $f^k_{\Theta_k}(x) = \Psi^k(\boldsymbol{W}^k x + \vec{b}^k)$
- the free parameters can be written as matrix $\Theta_k = (\boldsymbol{W}^k, \vec{b}^k)$ with weights $\boldsymbol{W}^k \in \mathbb{R}^{n_k \times n_{k-1}}$ and biases $\vec{b}^k \in \mathbb{R}^{n_k}$
- the activation function Ψ^k acts pointwise on the resulting vector of the affine linear map, i.e., $\Psi^k(x_1, \ldots, x_{n_k}) := \big(\psi^k(x_1), \ldots, \psi^k(x_{n^k})\big)$ where $\psi^k \colon \mathbb{R} \to \mathbb{R}$ is the chosen activation function for this layer
- the network is fully-connected if each weight matrix $\boldsymbol{W}^k$ is dense

Non-sequential artificial neural networks

- **Residual network:** Popular architecture involving *residual connections*. Can be interpreted as forward Euler method.



- **Concatenation:** Result from all previous layers are concatenated ([ ] indicates concatenation) to form the input to the next layer

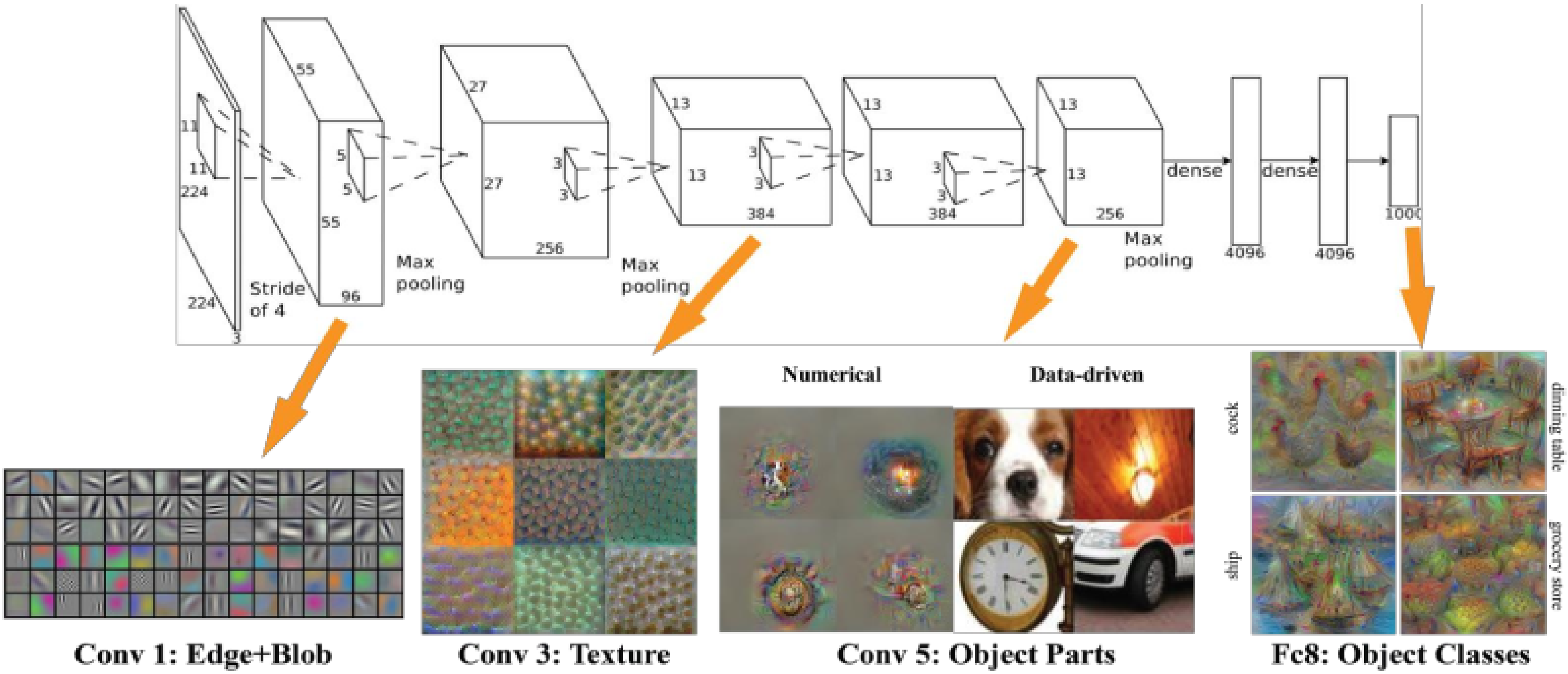


- **Sparse network:** Network architectures where most weights are zero, i.e., the connectivity is small relative to the number of connections possible.

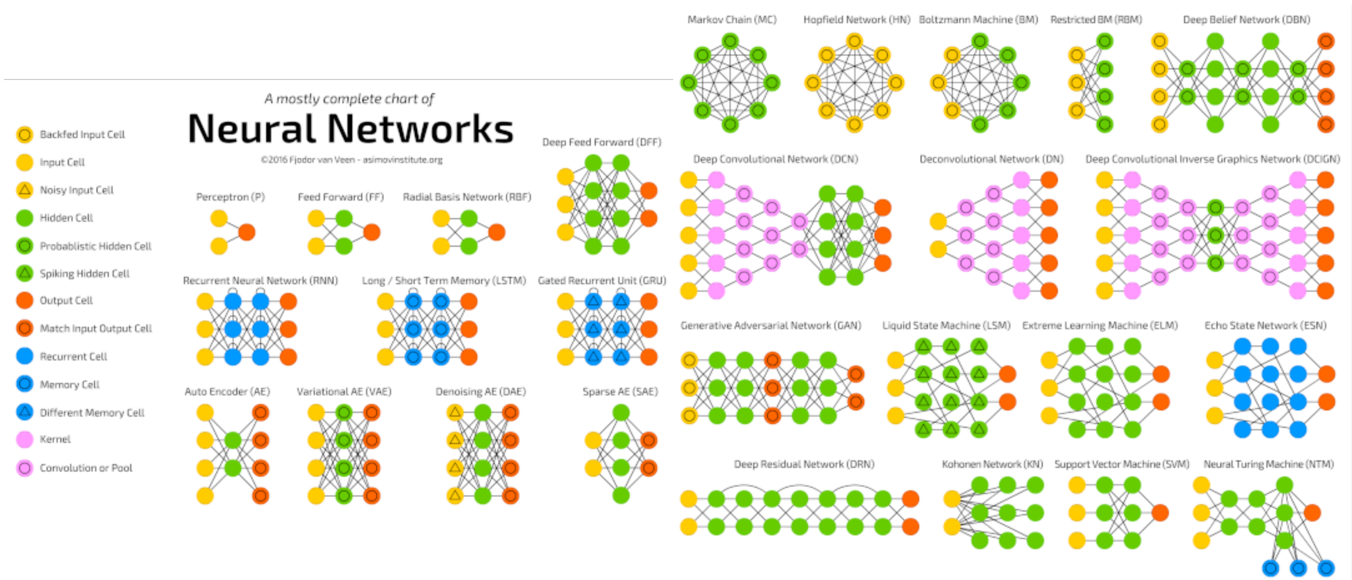
Convolutional neural networks (CNNs)

Idea: Encode the geometry of data (e.g., proximity, directions) in network structure.

- especially suitable for images, volumes, graphs
- easily parallelizable



Zoo of architectures



Open question

Machine learning task:

Given pairs of input/output data $(x^{(1)}, y^{(1)}), \ldots, (x^{(N)}, y^{(N)})$. How can we build an artificial neural network f_Θ such that

$$f_\Theta(x^{(k)}) \approx y^{(k)}, \ k = 1, \ldots, N.$$

Example:

Imagine an artificial neural network with an input layer (10 neurons), 5 hidden layers (10 neurons each), and a single output neuron. This leads to $10 * 10 + 4 * (10 * 10) + 10 = 510$ free parameters for the *weights* and 51 free parameters for the *biases*.
$\rightarrow$ Setting the free parameters Θ of a network manually is not feasible!

Solution: Obtain good parameters by **training** the neural network!

Conclusions

- Artificial neurons were designed to mimic biological processes
- Perceptrons realize linear classifiers
- combining multiple layers of artificial neurons allows to solve more complex tasks
- some network architectures are well-suited for certain applications
- manually choosing the free parameters of a network is infeasible

Thank you for your attention!