Introductory Course: Machine Learning (WWI15B4)

Classification II

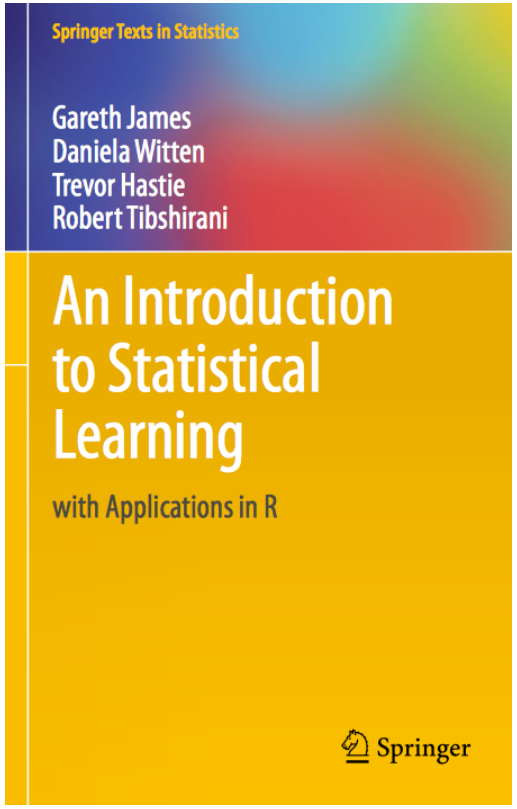
Fabio Ferreira, David Bethge

DHBW Karlsruhe

Overview

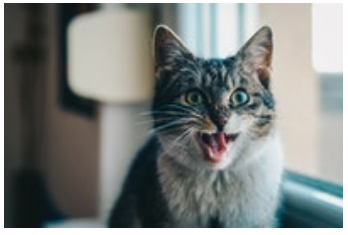
Table of Contents

Recommended Literature



or Stanford's CS231n course on CNNs is available online and gives a good introduction to classification and learning in computer vision

Example



cat image x, shape [100x70x3]

- parameter vector W
- $f(x, W) \longrightarrow$ 10 values indicating the class scores
- ideally, the scalar representing 'cat' has highest value
- W is a parameter vector used in *parametric* function

Parametric vs. Non-Parametric

many definitions, among one is:

- **parametric**: model summarizes data by finite set of parameters, e.g.

$$y_i = \boxed{\phi_0 + \phi_1 x_i} + e_i$$

- **non-parametric**: model (user) doesn't make explicit assumptions about the data distribution, e.g.

$$y_i = \boxed{f(x_i)} + e_i$$

- note: non-parametric methods still require parameters (however, their form isn't defined by the user)

Task: Classify the following ML approaches (parametric/non-parametric):

- decision tree
- nearest neighbor
- linear regression

Table of Contents

Example:Boston House Price dataset

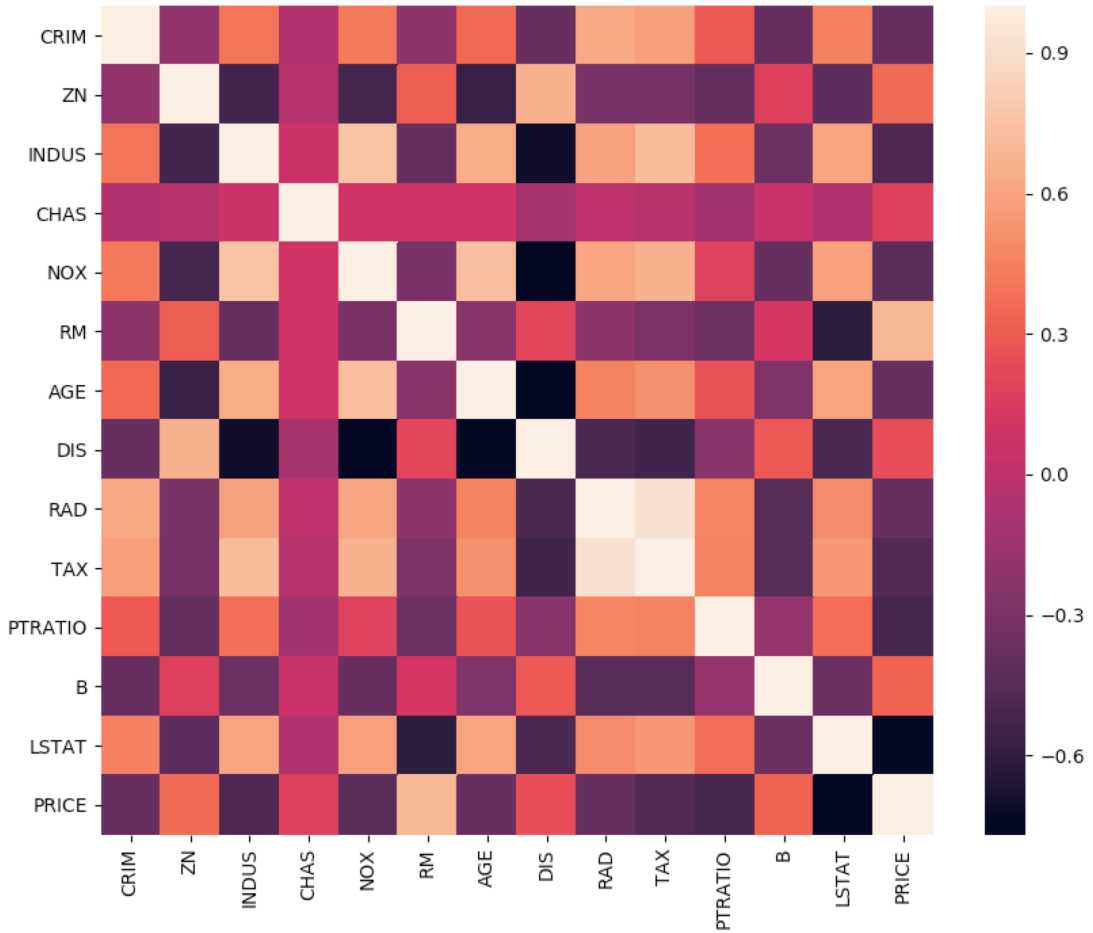
[The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic prices and the demand for clean air']

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	PRICE
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2

```
Data Set Characteristics:

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive

    :Median Value (attribute 14) is usually the target

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over 25,000 sq.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
        - NOX      nitric oxides concentration (parts per 10 million)
        - RM       average number of rooms per dwelling
        - AGE      proportion of owner-occupied units built prior to 1940
        - DIS      weighted distances to five Boston employment centres
        - RAD      index of accessibility to radial highways
        - TAX      full-value property-tax rate per $10,000
```

Boston House Price dataset

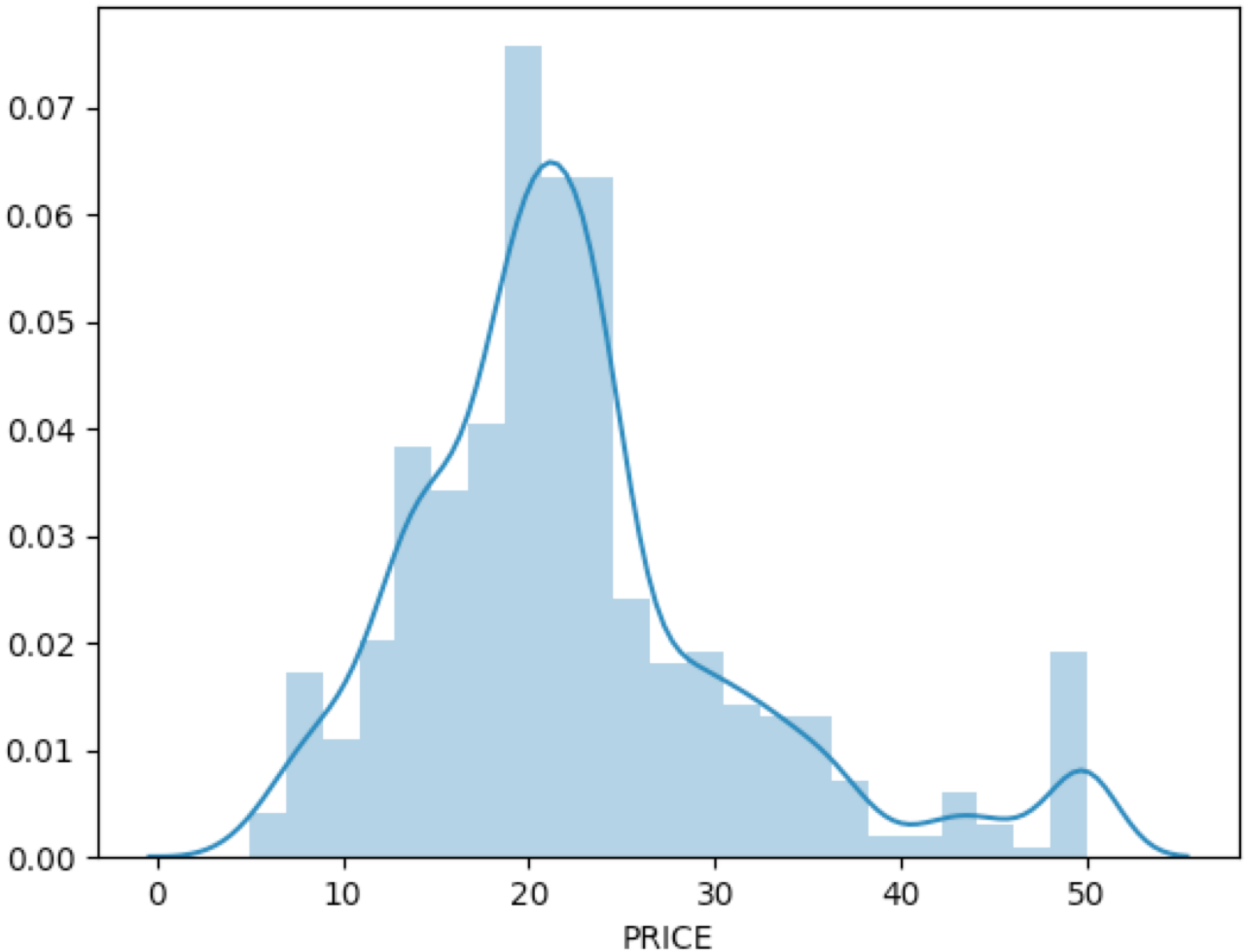
Boston House Price covariance matrix



RM: average number of rooms per dwelling

Boston House Price dataset

Price distribution



Logistic Regression vs. Linear Regression

Common demand in practice:

- there's some data existing
- learn the concept behind it, classify new samples

Logistic Regression vs. Linear Regression

Common demand in practice:

- there's some data existing
- learn the concept behind it, classify new samples

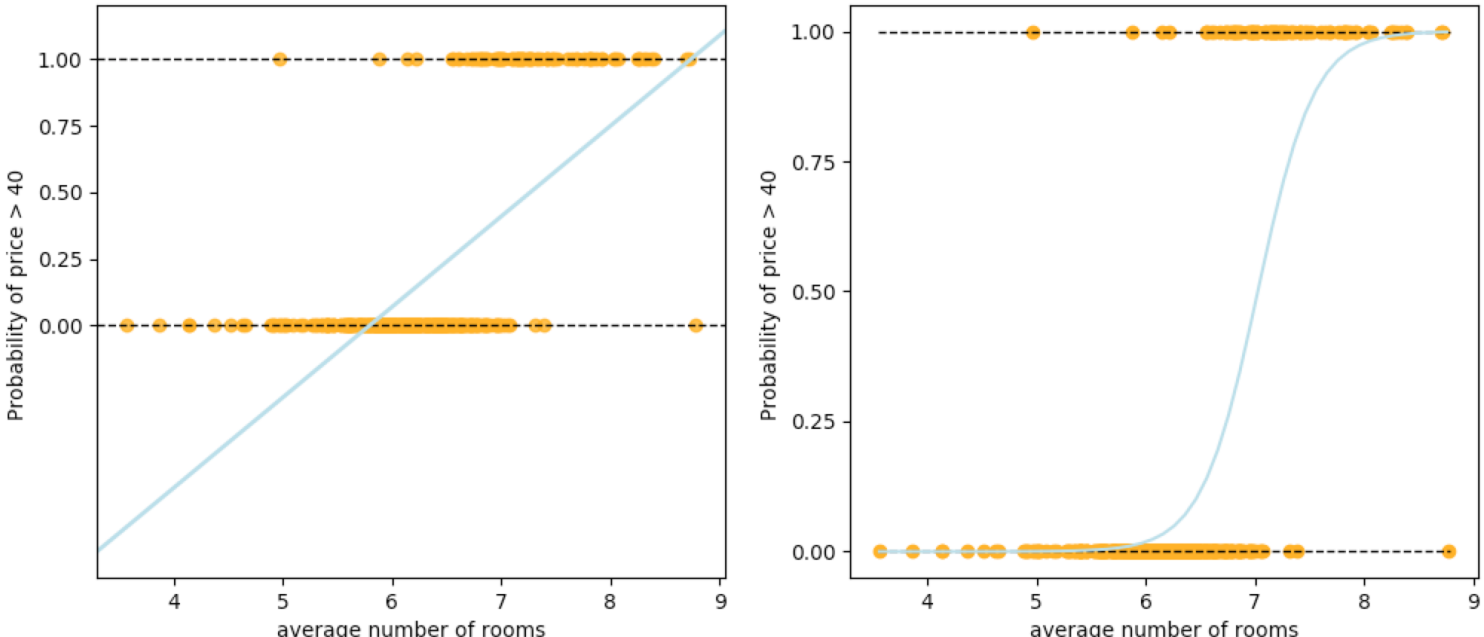
Considering the previous Boston housing dataset: estimate probability that the price > 40. (left: linear regression, right: logistic regression

Logistic Regression vs. Linear Regression

Common demand in practice:

- there's some data existing
- learn the concept behind it, classify new samples

Considering the previous Boston housing dataset: estimate probability that the price > 40. (left: linear regression, right: logistic regression



Logistic Regression

- for our linear reg. example we used $p(x) = \beta_0 + \beta_1 x$
- observation: probabilities given by linear reg. can be unreasonable $\rightarrow$ logistic regression
- Logistic regression measures the relationship between the categorical dependent variable (p(X)) and one or more independent variables (X) by estimating probabilities using a *logistic function*

Logistic Regression

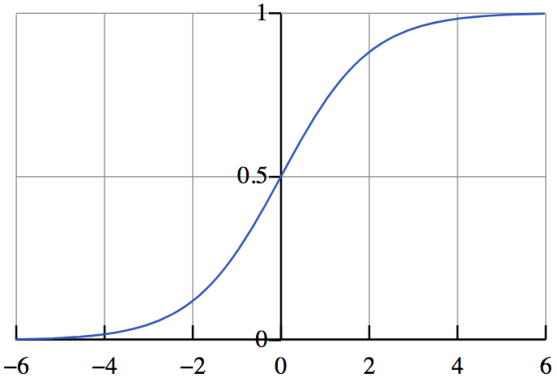
Logistic Function

$$\sigma(z) = \frac{e^z}{1+e^z} = \frac{1}{1+e^{-z}}$$

- in our example we had:
  $z = \beta_0 + \beta_1 x$
- if we have m explanatory variables (multiple regression) we have:
  $z = \beta_0 + \sum_{i=1}^{m} \beta_i x_i$
- for classification we can use:

$$y = \begin{cases} 1 & \beta_0 + \sum_{i=1}^{m} \beta_i x_i > \mathit{thresh} \\ 0 & \mathit{else} \end{cases}$$

$\mathit{tresh} \in [0, 1)$ is some arbitrary threshold



Logistic Regression Side Note

- how many coefficients in multiple regression are required?
- assess the model fit
- do likelihood ratio-test or Wald test after fitting the coefficients
- don't confuse multiple regression with multinomial regression
  - multiple l. regression: multiple coefficients (independent variables)
  - multinomial l. regression: dependent variable has more than two outcome categories

Observations and Example

- if z is large:
  $\sigma(z) \approx \frac{1}{1+0} = 1$
- if z is very small:
  $\sigma(z) \approx \frac{1}{1+\infty} = 0$
- fitting logistic regression to the Boston House Price dataset yielded: $\beta_0 = -27.143, \beta_1 = 3.862$

Observations and Example

- if z is large:
  $\sigma(z) \approx \frac{1}{1+0} = 1$
- if z is very small:
  $\sigma(z) \approx \frac{1}{1+\infty} = 0$
- fitting logistic regression to the Boston House Price dataset yielded: $\beta_0 = -27.143, \beta_1 = 3.862$

Looking for 6-room apartment:

Probability of price $> 40 = \frac{1}{1+e^{-(-27.143+3.862*\mathbf{6})}} = 0.018$

Observations and Example

- if z is large:
  $\sigma(z) \approx \frac{1}{1+0} = 1$
- if z is very small:
  $\sigma(z) \approx \frac{1}{1+\infty} = 0$
- fitting logistic regression to the Boston House Price dataset yielded: $\beta_0 = -27.143, \beta_1 = 3.862$

Looking for 6-room apartment:
Probability of price $> 40 = \frac{1}{1+e^{-(-27.143+3.862*\mathbf{6})}} = 0.018$
Looking for a 8-room apartment:
Probability of price $> 40 = \frac{1}{1+e^{-(-27.143+3.862*\mathbf{9})}} = 0.999$

Observations and Example

- if z is large:
  $\sigma(z) \approx \frac{1}{1+0} = 1$
- if z is very small:
  $\sigma(z) \approx \frac{1}{1+\infty} = 0$
- fitting logistic regression to the Boston House Price dataset yielded: $\beta_0 = -27.143, \beta_1 = 3.862$

Looking for 6-room apartment:

Probability of price $> 40 = \frac{1}{1+e^{-(-27.143+3.862*\mathbf{6})}} = 0.018$

Looking for a 8-room apartment:

Probability of price $> 40 = \frac{1}{1+e^{-(-27.143+3.862*\mathbf{9})}} = 0.999$

$\Rightarrow$ **But how do we fit our LR model?**

Fitting in Logistic Regression

How to find the regression coefficients?

- unlike in linear regression we can't find a closed-form solution for logistic regression
- instead, iterative numerical method must be used
- we could use non-linear least squares but
- more common to use is Maximum likelihood estimation
- typically: Newton's method, gradient descent or L-BFGS

Maximum Likelihood Estimation

Given the data $\{(x_1, y_1), ..., (x_m, y_m)\}$ with $x_i, y_i \in \mathbb{R}^{n_x+1}$ and $x_0 = 1$ we want to find parameters $\theta = [\beta_0, \beta_1, ..., \beta_{n_x}]^T$ that maximize the (conditional) likelihood of class labels Y given the training data X.

- MLE attempts to find the parameter values that maximize the likelihood function (=make the data most probable), given the data: $\hat{\theta} = \text{argmax}_{\theta \in \Theta} \mathcal{L}(\theta; X)$

Maximum Likelihood Estimation

Given the data $\{(x_1, y_1), ..., (x_m, y_m)\}$ with $x_i, y_i \in \mathbb{R}^{n_x+1}$ and $x_0 = 1$ we want to find parameters $\theta = [\beta_0, \beta_1, ..., \beta_{n_x}]^T$ that maximize the (conditional) likelihood of class labels Y given the training data X.

- MLE attempts to find the parameter values that maximize the likelihood function (=make the data most probable), given the data: $\hat{\theta} = \text{argmax}_{\theta \in \Theta} \mathcal{L}(\theta; X)$
- be $\hat{y} = \frac{1}{1+e^{-\theta^T X}}$

Maximum Likelihood Estimation

Given the data $\{(x_1, y_1), ..., (x_m, y_m)\}$ with $x_i, y_i \in \mathbb{R}^{n_x+1}$ and $x_0 = 1$ we want to find parameters $\theta = [\beta_0, \beta_1, ..., \beta_{n_x}]^T$ that maximize the (conditional) likelihood of class labels Y given the training data X.

- MLE attempts to find the parameter values that maximize the likelihood function (=make the data most probable), given the data: $\hat{\theta} = \text{argmax}_{\theta \in \Theta} \mathcal{L}(\theta; X)$
- be $\hat{y} = \frac{1}{1+e^{-\theta^T X}}$
- in our case, the likelihood is maximized when we minimize the errors between y and $\hat{y}$

Maximum Likelihood Estimation

Given the data $\{(x_1, y_1), ..., (x_m, y_m)\}$ with $x_i, y_i \in \mathbb{R}^{n_x+1}$ and $x_0 = 1$ we want to find parameters $\theta = [\beta_0, \beta_1, ..., \beta_{n_x}]^T$ that maximize the (conditional) likelihood of class labels Y given the training data X.

- MLE attempts to find the parameter values that maximize the likelihood function (=make the data most probable), given the data: $\hat{\theta} = \text{argmax}_{\theta \in \Theta} \mathcal{L}(\theta; X)$
- be $\hat{y} = \frac{1}{1+e^{-\theta^T X}}$
- in our case, the likelihood is maximized when we minimize the errors between y and $\hat{y}$
- using the standard squared error

$$L(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$$

we want the expression to be as small as possible

Maximum Likelihood Estimation

Given the data $\{(x_1, y_1), ..., (x_m, y_m)\}$ with $x_i, y_i \in \mathbb{R}^{n_x+1}$ and $x_0 = 1$ we want to find parameters $\theta = [\beta_0, \beta_1, ..., \beta_{n_x}]^T$ that maximize the (conditional) likelihood of class labels Y given the training data X.

- for maximum likelihood we define the log-likelihood as follows (**loss function**):

$$L(\hat{y}, y) = -\left[y \log \hat{y} + (1 - y) \log(1 - \hat{y})\right]$$

- intuition:
  - minimizing a negative quantity = maximizing a positive quantity
  - if y=1: $L(\hat{y}, y) = -1 \log \hat{y} \rightarrow$ we want large $\hat{y}$
  - if y=0: $L(\hat{y}, y) = -\log(1 - \hat{y}) \rightarrow$ we want small $\hat{y}$

Maximum Likelihood Estimation(2)

Given the data $\{(x_1, y_1), ..., (x_m, y_m)\}$ with $x_i, y_i \in \mathbb{R}^{n_x+1}$ and $x_0 = 1$ we want to find parameters $\theta = [\beta_0, \beta_1, ..., \beta_{n_x}]^T$ that maximize the (conditional) likelihood of class labels Y given the training data X.

- for maximum likelihood we define the log-likelihood as our **loss function**:

$$L(\hat{y}, y) = -\big[y \log \hat{y} + (1 - y) \log(1 - \hat{y})\big]$$

- intuition

Maximum Likelihood Estimation(2)

Given the data $\{(x_1, y_1), ..., (x_m, y_m)\}$ with $x_i, y_i \in \mathbb{R}^{n_x+1}$ and $x_0 = 1$ we want to find parameters $\theta = [\beta_0, \beta_1, ..., \beta_{n_x}]^T$ that maximize the (conditional) likelihood of class labels Y given the training data X.

- for maximum likelihood we define the log-likelihood as our **loss function**:

$$L(\hat{y}, y) = -\big[y \log \hat{y} + (1 - y) \log(1 - \hat{y})\big]$$

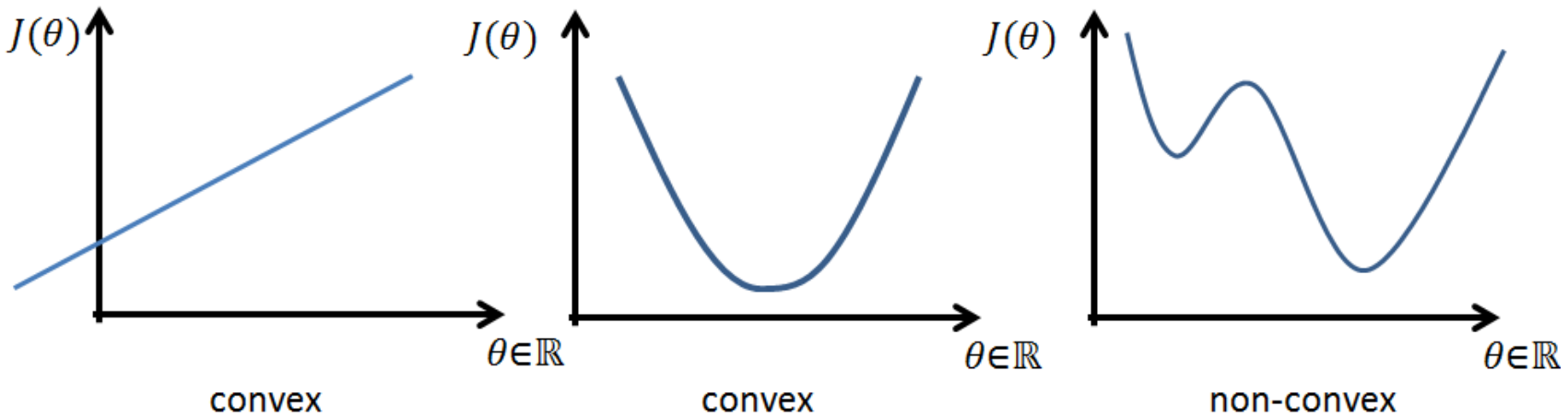
- intuition
- over the entirety of our data this yields the **cost function:**

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} L(\hat{y}_i, y_i)$$

  which for example can be minimized with gradient descent (later)
  $\rightarrow$ yields the maximum likelihood estimates θ

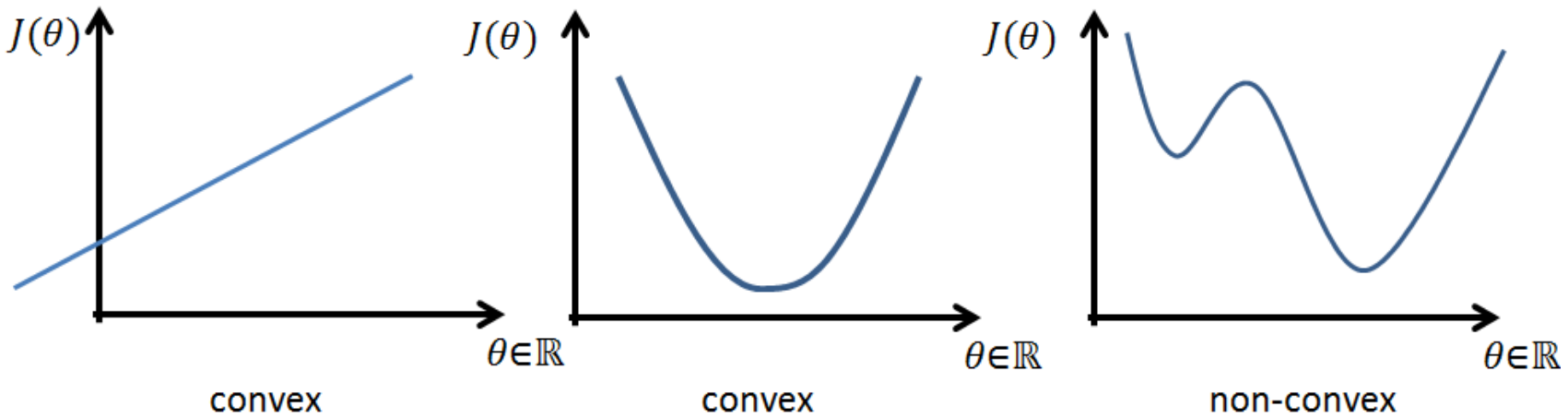
Logistic Regression Optimization

- the standard squared loss function is **non-convex**
- the log-likelihood is **convex**
- what does it mean? Some simplified examples:



Logistic Regression Optimization

- the standard squared loss function is **non-convex**
- the log-likelihood is **convex**
- what does it mean? Some simplified examples:



- log-likelihood and gradient descent (or any other iterative numerical optimization algorithm) $\rightarrow$ yields optimal coefficients

Gradient Descent

- used to find a minimum of a function

Gradient Descent

- used to find a minimum of a function
- steps are taken in proportion to the negative of the (approx.) gradient (if positive: gradient ascent)

Gradient Descent

- used to find a minimum of a function
- steps are taken in proportion to the negative of the (approx.) gradient (if positive: gradient ascent)
- given points x, an initial guess x_0 and a learning rate α, gradient descent is defined as:

$$x_{n+1} = x_n - \alpha \nabla F(x_n)$$

Gradient Descent

- used to find a minimum of a function
- steps are taken in proportion to the negative of the (approx.) gradient (if positive: gradient ascent)
- given points x, an initial guess x_0 and a learning rate α, gradient descent is defined as:

$$x_{n+1} = x_n - \alpha \nabla F(x_n)$$

- we subtract from x since we want to 'move' against the gradient towards the minimum

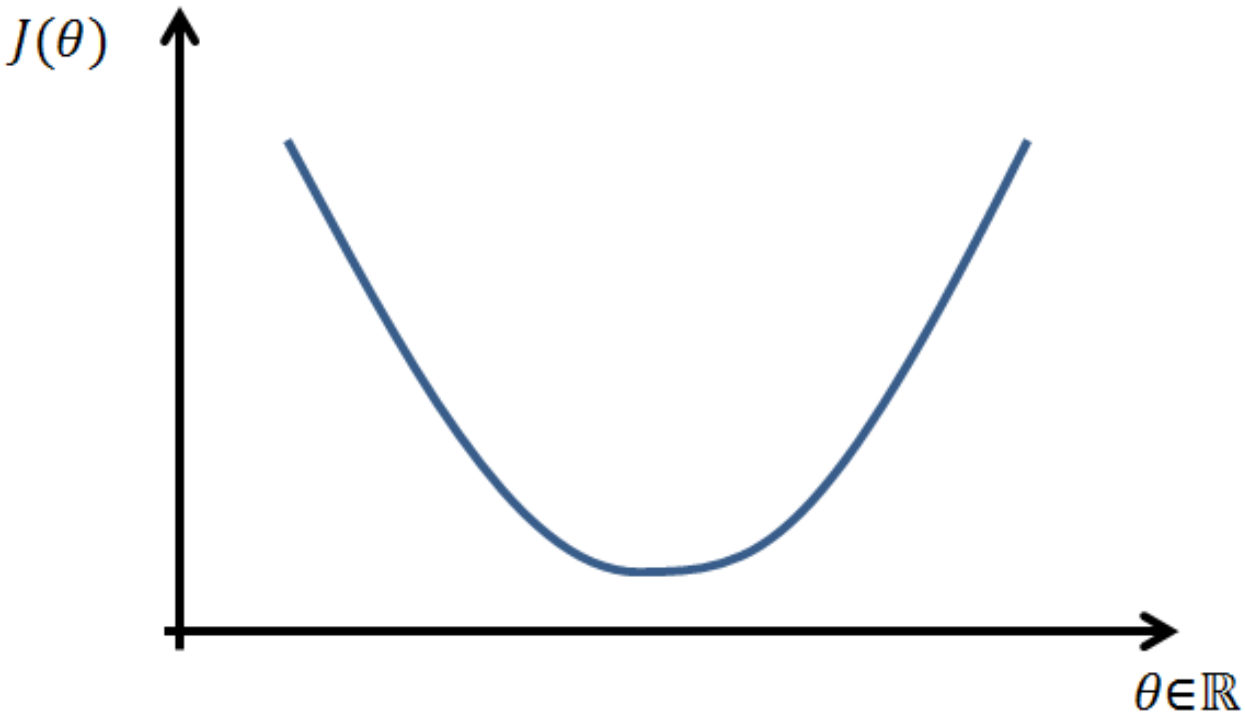
Gradient Descent

- used to find a minimum of a function
- steps are taken in proportion to the negative of the (approx.) gradient (if positive: gradient ascent)
- given points x, an initial guess x_0 and a learning rate α, gradient descent is defined as:

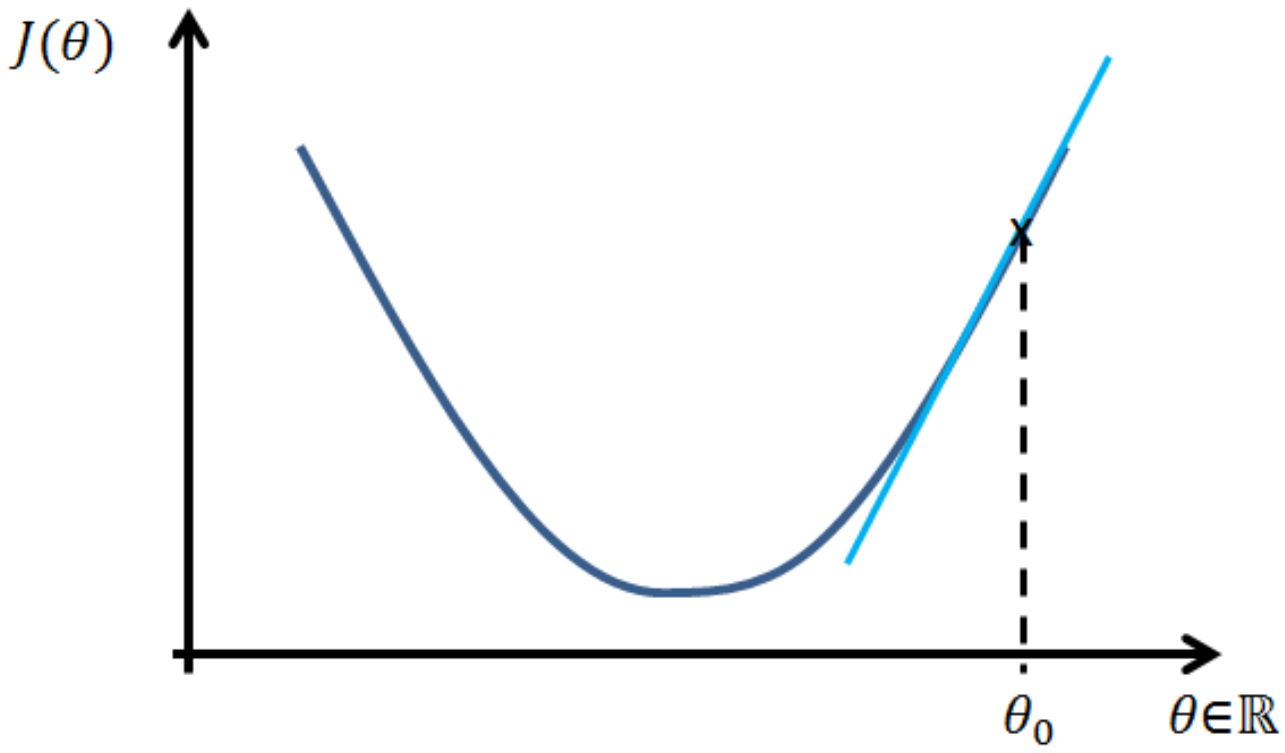
$$x_{n+1} = x_n - \alpha \nabla F(x_n)$$

- we subtract from x since we want to 'move' against the gradient towards the minimum
- it follows for small α: $F(x_n) \geq F(x_{n+1})$

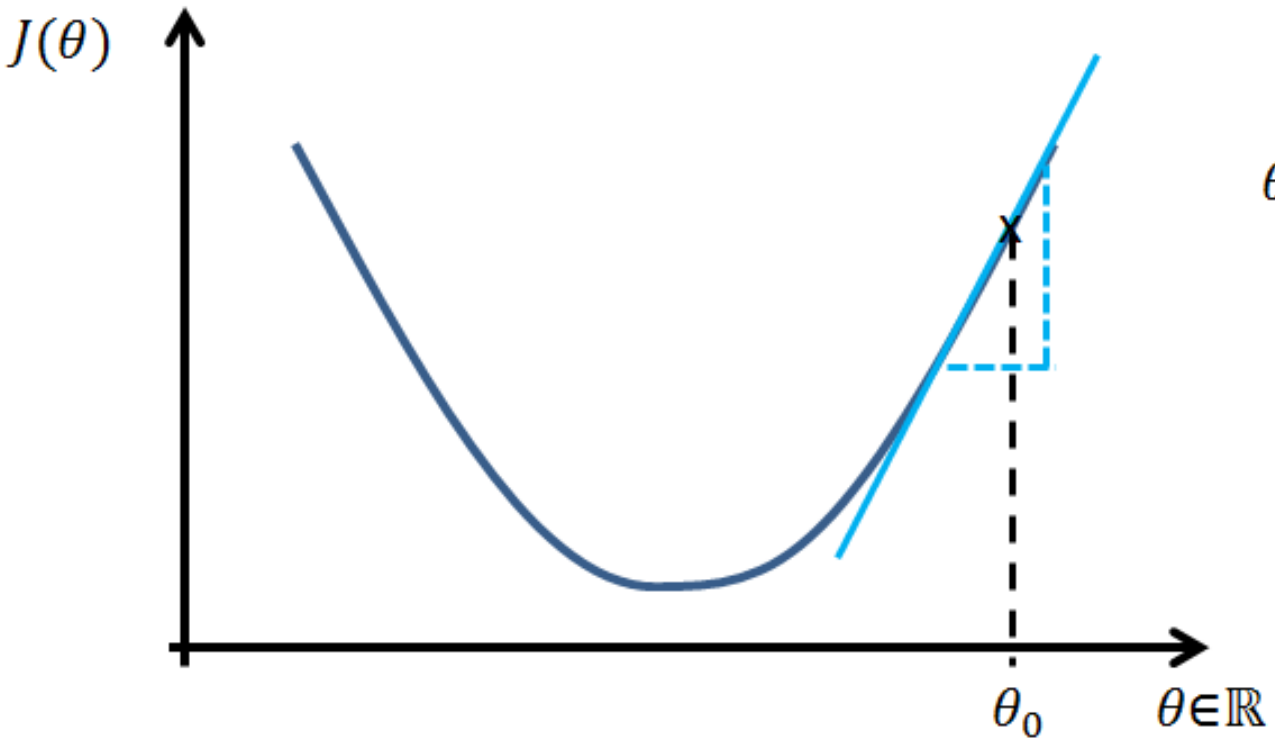
Gradient Descent Example



Gradient Descent Example

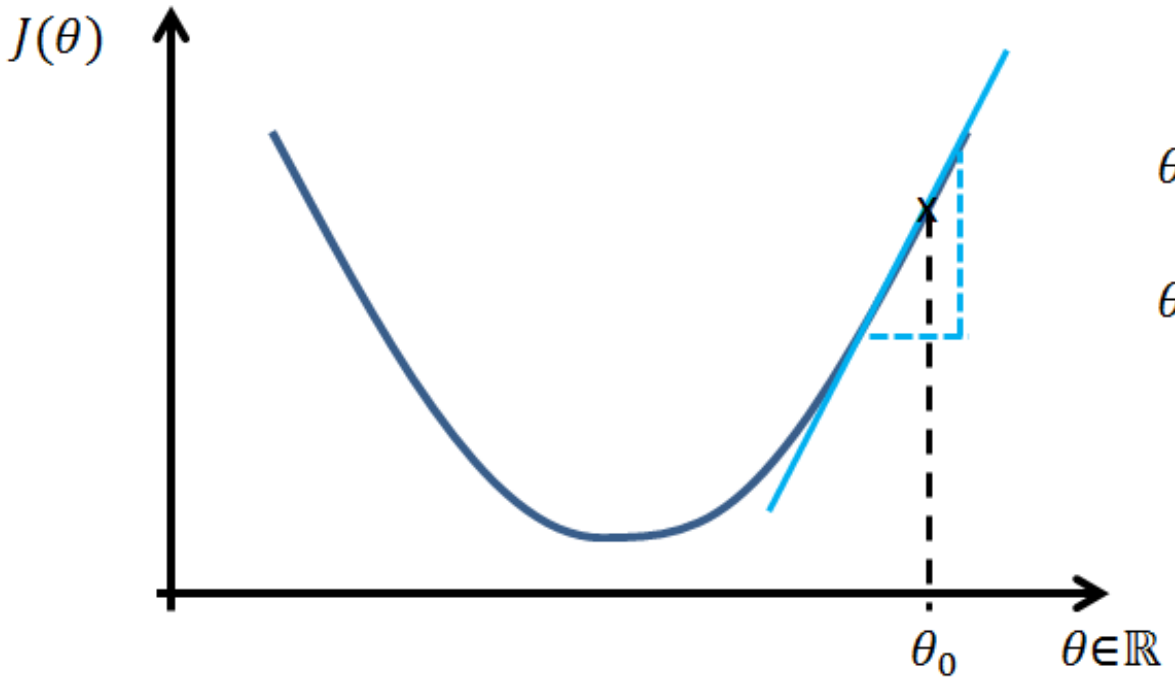


Gradient Descent Example



$$\theta_1 = \theta_0 - \alpha \frac{d\,J(\theta_0)}{d\,\theta_0}$$

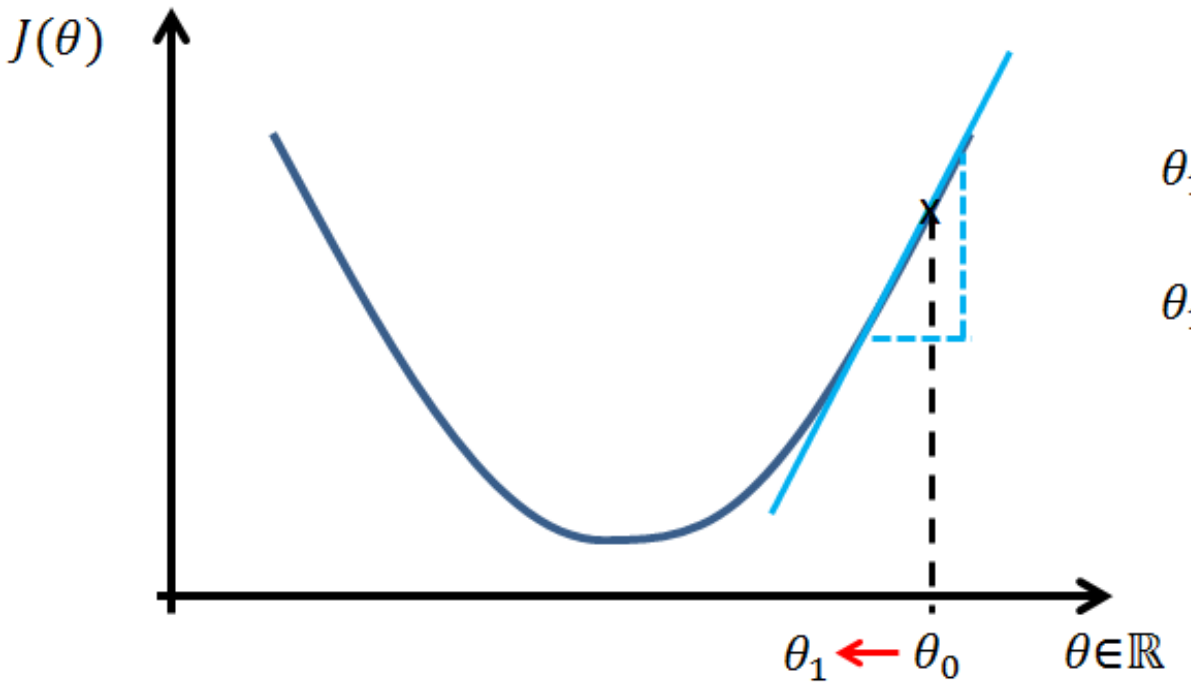
Gradient Descent Example



$$\theta_1 = \theta_0 - \alpha \frac{d\,J(\theta_0)}{d\,\theta_0}$$

$$\theta_1 = \theta_0 - \alpha(positive\ number)$$

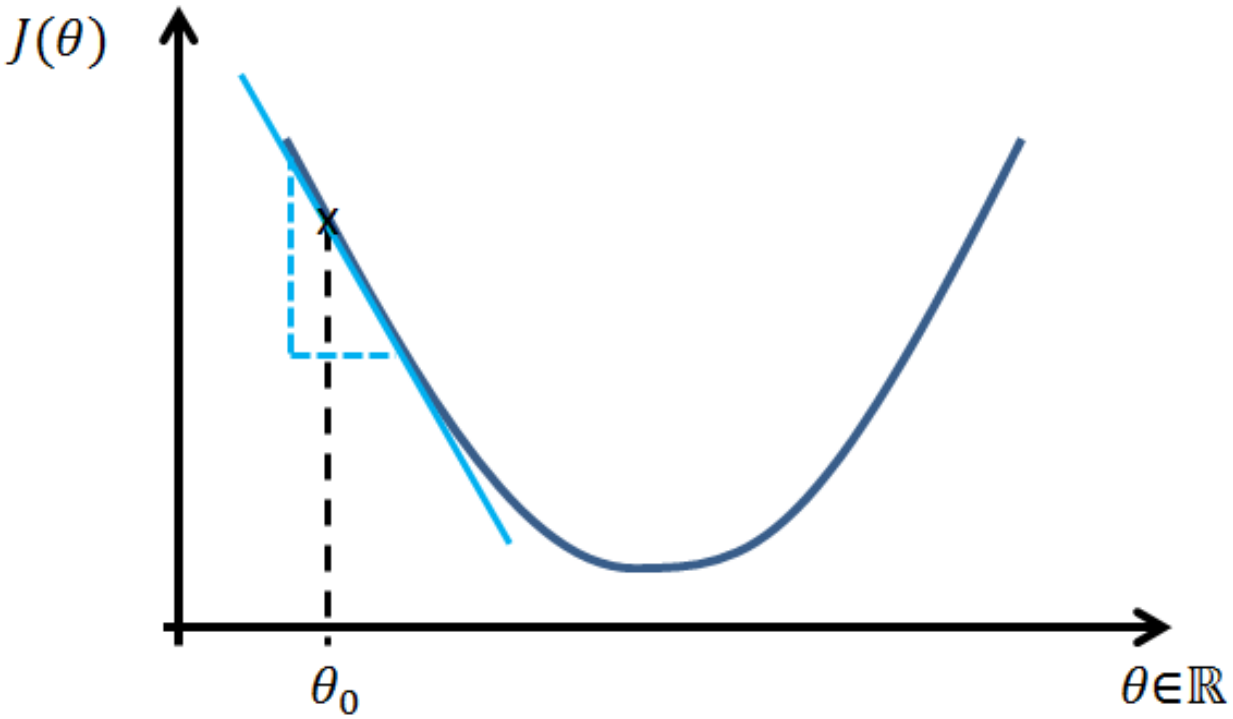
Gradient Descent Example



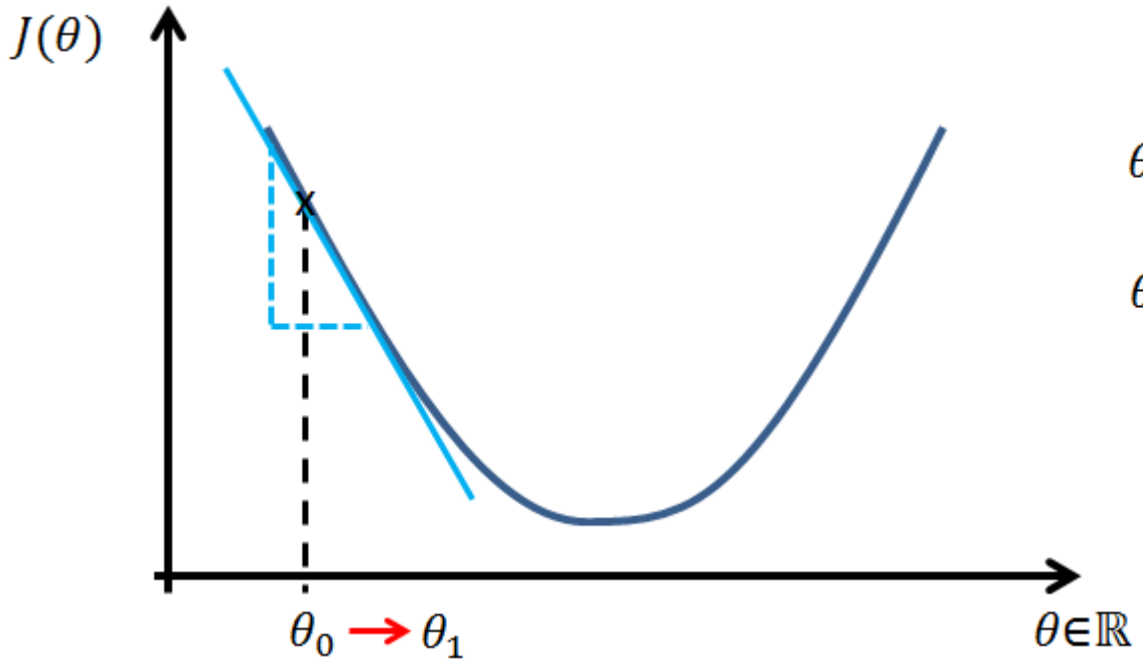
$$\theta_1 = \theta_0 - \alpha \frac{d\,J(\theta_0)}{d\,\theta_0}$$

$$\theta_1 = \theta_0 - \alpha(positive\ number)$$

Gradient Descent Example



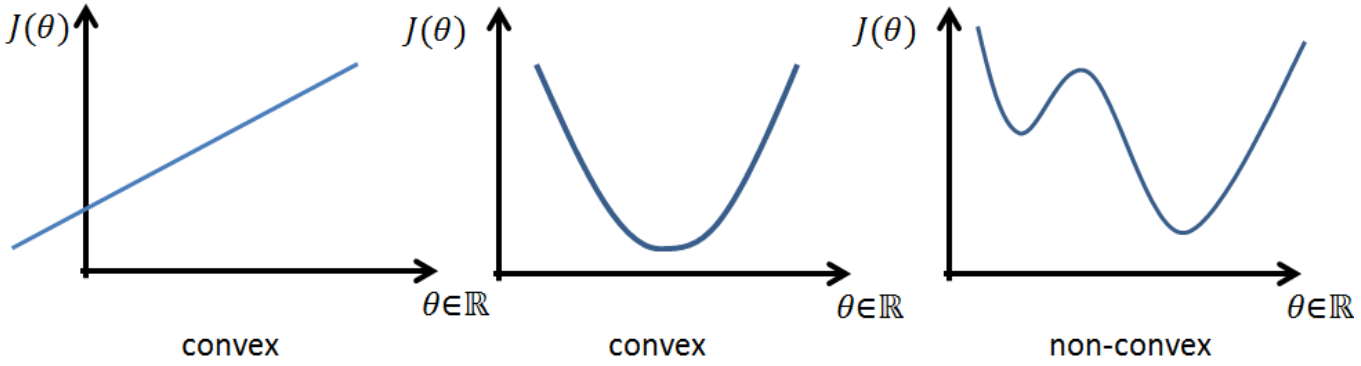
Gradient Descent Example



$$\theta_1 = \theta_0 - \alpha \frac{d\,J(\theta_0)}{d\,\theta_0}$$

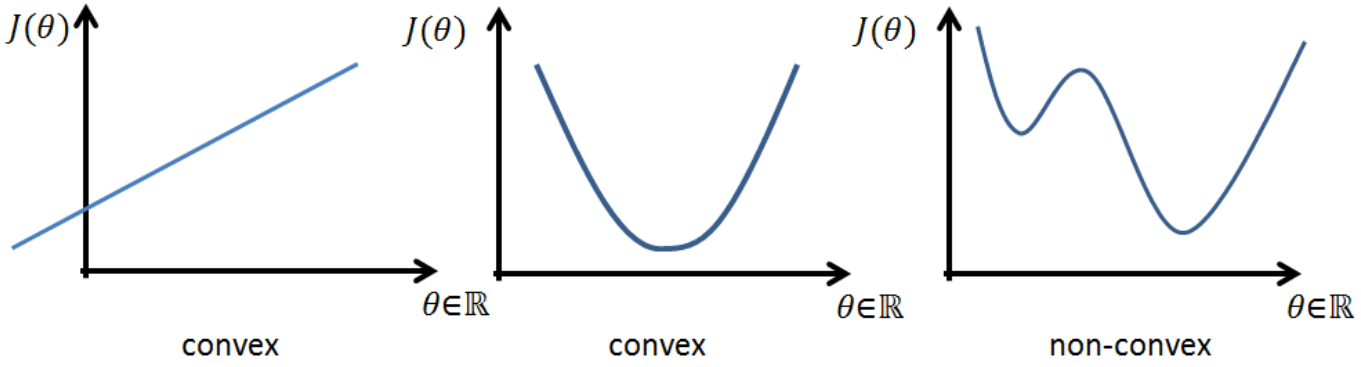
$$\theta_1 = \theta_0 - \alpha(\textbf{negative number})$$

Gradient Descent Questions



- What can be an issue when gradient descent is used with non-convex functions?

Gradient Descent Questions

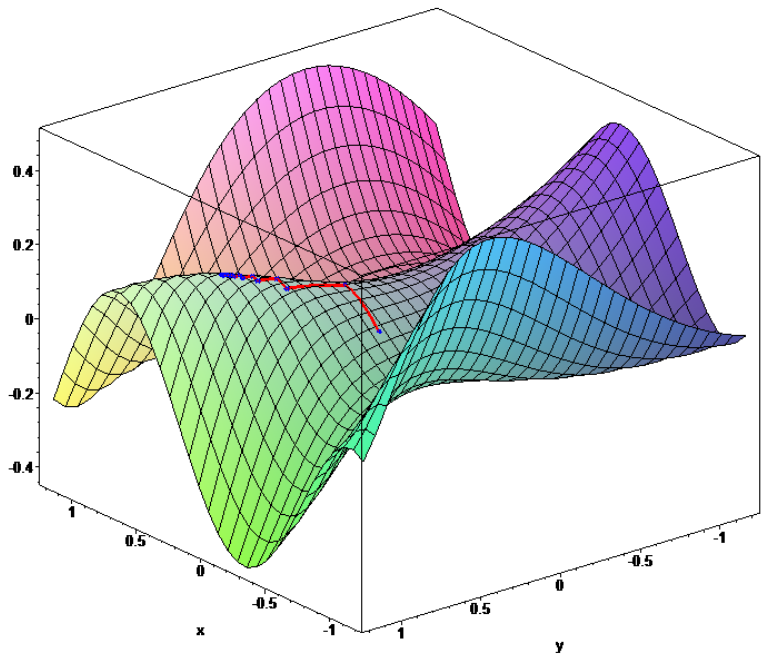


- What can be an issue when gradient descent is used with non-convex functions?
- What behavior can be caused by a small/large learning rate? Remember:

$$x_{n+1} = x_n - \alpha \nabla F(x_n)$$

Gradient Descent Dimensionality

Keep in mind that we usually operate on highly dimensional problems. 2d parameter space:



[Encyclopedia, 2016] Deep neural networks can have millions of parameters

Logistic Regression in Practice

- LR for handwritten digits recognition

[LeCun et al., 1998, MNIST]

- LR for cancer detection [Stainvas et al., 2014]

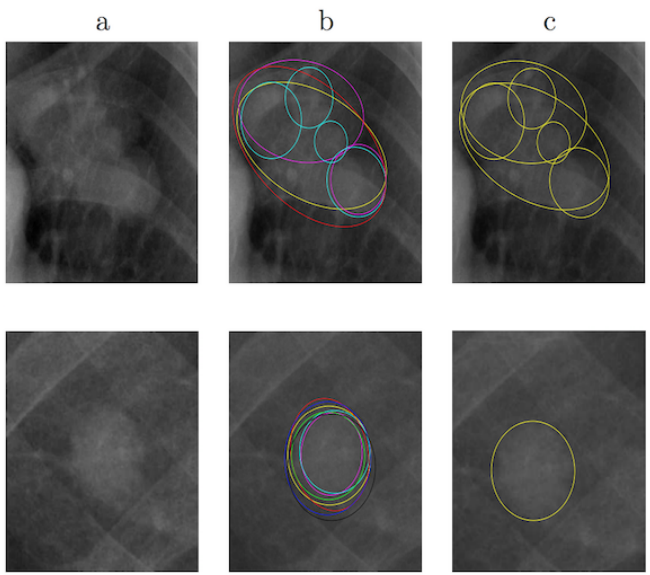


Table of Contents

(Artificial) Neural Networks

- collection of **interconnected nodes** (artificial neurons)

(Artificial) Neural Networks

- collection of **interconnected nodes** (artificial neurons)
- each connection can transmit a **signal** from one node to another

(Artificial) Neural Networks

- collection of **interconnected nodes** (artificial neurons)
- each connection can transmit a **signal** from one node to another
- the signal at a connection is a real number and the output of each neuron is calculated by a **non-linear function of the sum of its inputs**

(Artificial) Neural Networks

- collection of **interconnected nodes** (artificial neurons)
- each connection can transmit a **signal** from one node to another
- the signal at a connection is a real number and the output of each neuron is calculated by a **non-linear function of the sum of its inputs**
- each connection also has a weight that amplifies or damps the signal

(Artificial) Neural Networks

- collection of **interconnected nodes** (artificial neurons)
- each connection can transmit a **signal** from one node to another
- the signal at a connection is a real number and the output of each neuron is calculated by a **non-linear function of the sum of its inputs**
- each connection also has a weight that amplifies or damps the signal
- NNs typically arranged into **layers**

(Artificial) Neural Networks

- collection of **interconnected nodes** (artificial neurons)
- each connection can transmit a **signal** from one node to another
- the signal at a connection is a real number and the output of each neuron is calculated by a **non-linear function of the sum of its inputs**
- each connection also has a weight that amplifies or damps the signal
- NNs typically arranged into **layers**
- NNs can be used for **regression** and **classification** (and more)

(Artificial) Neural Networks

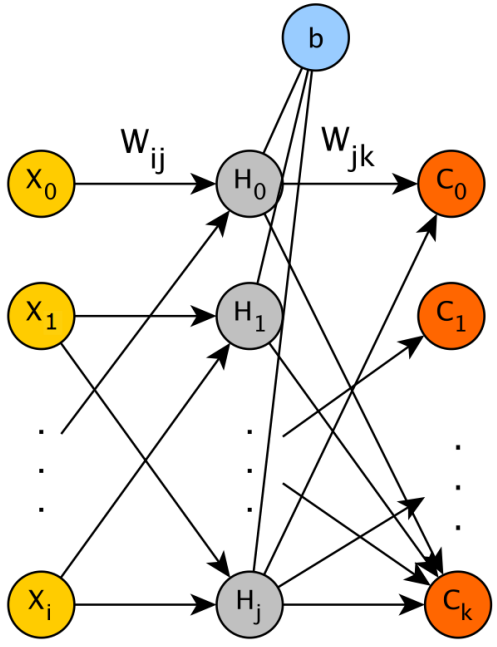
- collection of **interconnected nodes** (artificial neurons)
- each connection can transmit a **signal** from one node to another
- the signal at a connection is a real number and the output of each neuron is calculated by a **non-linear function of the sum of its inputs**
- each connection also has a weight that amplifies or damps the signal
- NNs typically arranged into **layers**
- NNs can be used for **regression** and **classification** (and more)
- typically trained via the **backpropagation algorithm**

(Artificial) Neural Networks

- collection of **interconnected nodes** (artificial neurons)
- each connection can transmit a **signal** from one node to another
- the signal at a connection is a real number and the output of each neuron is calculated by a **non-linear function of the sum of its inputs**
- each connection also has a weight that amplifies or damps the signal
- NNs typically arranged into **layers**
- NNs can be used for **regression** and **classification** (and more)
- typically trained via the **backpropagation algorithm**
- NNs can be seen as **non-linear function approximators**

(Artificial) Neural Networks Example

A single-layer neural network (sometimes: single-layer perceptron):



NN's can also be seen as feature extractors since the weights allow/restrict certain input areas (input nodes) to "flow" better or worse towards the output

Neural Network Model Classes

- artificial neural networks (ANN or just NN) form a large class of models, to name just a few:

Neural Network Model Classes

- artificial neural networks (ANN or just NN) form a large class of models, to name just a few:
  - feed-forward NN
    - Perceptron / "Vanilla"
    - Convolutional
    - Autoencoder

Neural Network Model Classes

- artificial neural networks (ANN or just NN) form a large class of models, to name just a few:
  - feed-forward NN
    - Perceptron / "Vanilla"
    - Convolutional
    - Autoencoder
  - recurrent NN
    - Long short-term memory
    - Bi-directional

Neural Network Model Classes

- artificial neural networks (ANN or just NN) form a large class of models, to name just a few:
  - feed-forward NN
    - Perceptron / "Vanilla"
    - Convolutional
    - Autoencoder
  - recurrent NN
    - Long short-term memory
    - Bi-directional
  - undirected NN
    - Deep Belief
    - Restricted Boltzmann

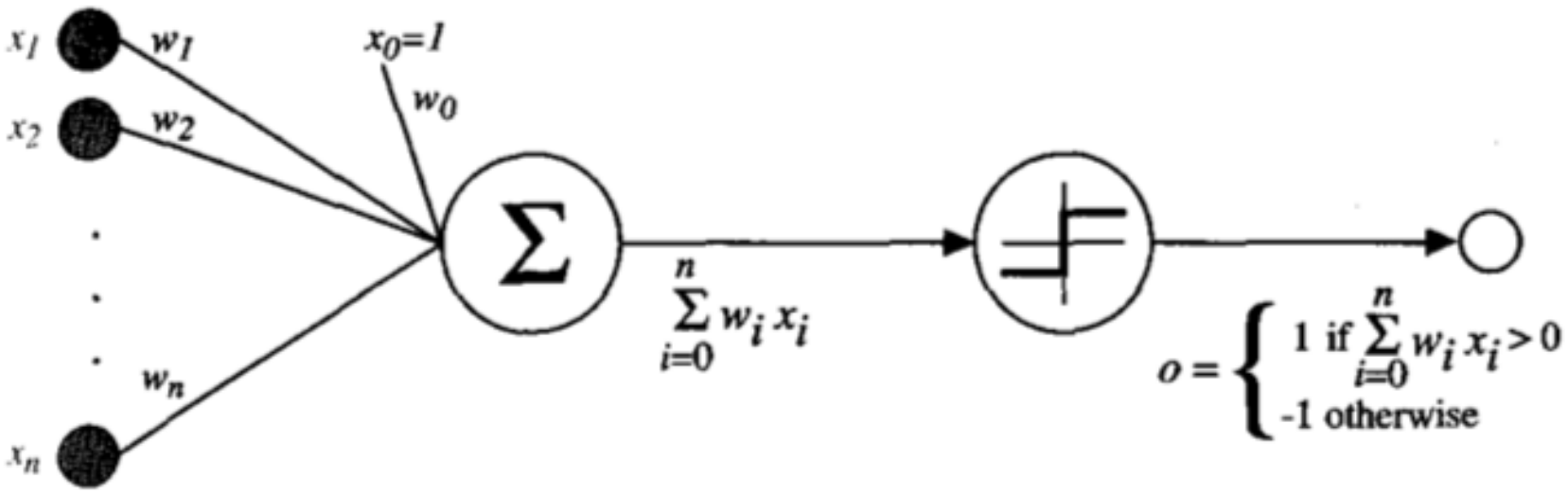
Neural Network Model Classes

- artificial neural networks (ANN or just NN) form a large class of models, to name just a few:
  - feed-forward NN
    - Perceptron / "Vanilla"
    - Convolutional
    - Autoencoder
  - recurrent NN
    - Long short-term memory
    - Bi-directional
  - undirected NN
    - Deep Belief
    - Restricted Boltzmann
  - "third generation": Spiking NN
- in this lecture we will focus on feed-forward NN for classification

Perceptron (Rosenblatt, 1957)

- one of the first ANNs
- algorithm for learning a linear binary classifier
- positive (P) and negative (N) sets of data samples
- input x is mapped to an output value (0 or 1) which is calculated by the dot product of the weight vector w and x
- the output value is used to classify x as either a positive or a negative instance

Perceptron (Rosenblatt, 1957)



from [Mitchell, 1997]

adjustment of weights is equivalent to searching for the best decision boundary

Perceptron (Rosenblatt, 1957)

algorithm:

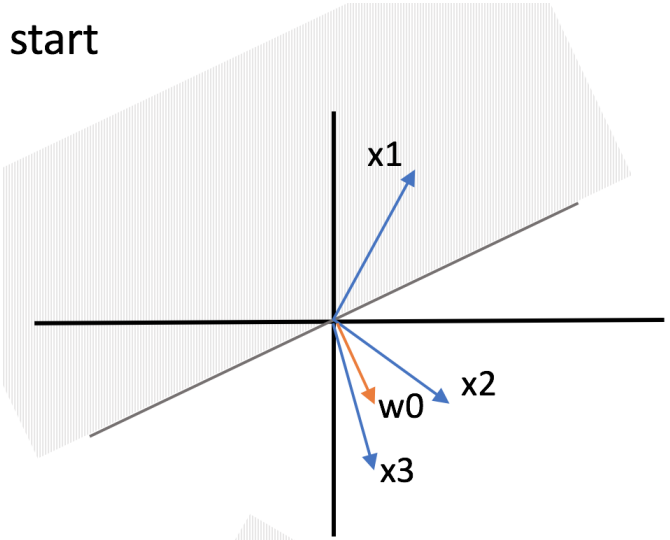
- Phase start:
  - initialize w_0 randomly, $t := 0$
  - given data samples $P \cup N$
- Phase test:
  - choose x $in P \cup N$ randomly
  - if $x \in P$ and $w_t \cdot x > 0$ go to test
  - if $x \in P$ and $w_t \cdot x \leq 0$ go to add
  - if $x \in N$ and $w_t \cdot x < 0$ go to test
  - if $x \in N$ and $w_t \cdot x \geq 0$ go to subtract
- Phase add:
  - set $w_{t+1} = w_t + x, t := t + 1$
- Phase subtract:
  - set $w_{t+1} = w_t - x, t := t + 1$

Perceptron Example: Geometric Interpretation in $\mathbb{R}^2$

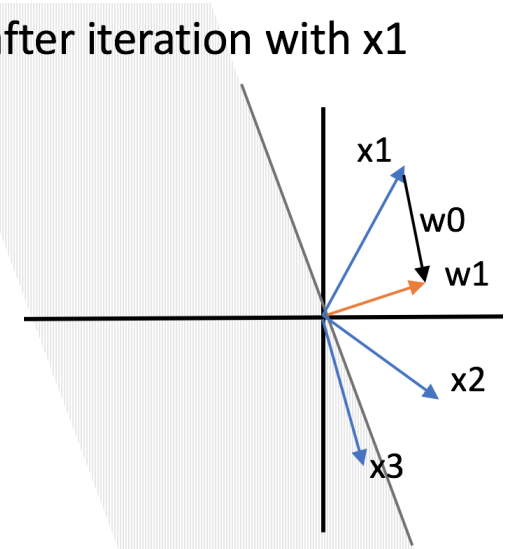
- in the following example, all shown samples x_i from P
- note the dot product: $w_t \cdot x_j$, i.e. $w_t^0 x_j^0 + ... + w_t^i x_j^i$ (for j-th sample at time-step t and $0 < i < n$ for n-dimensional vectors)
- sometimes extension used: construct a supporting set $N' = \{x' | x' = -x, \forall x \in N\}$
- now all samples are correctly classified iff $x \cdot w > 0, \forall x \in N' \cup P$

Perceptron Example: Geometric Interpretation in $\mathbb{R}^2$

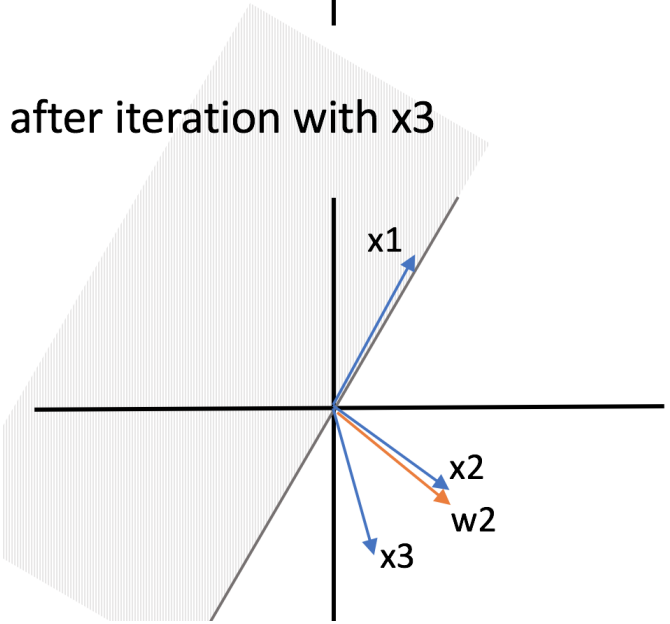
start



after iteration with x1



after iteration with x3

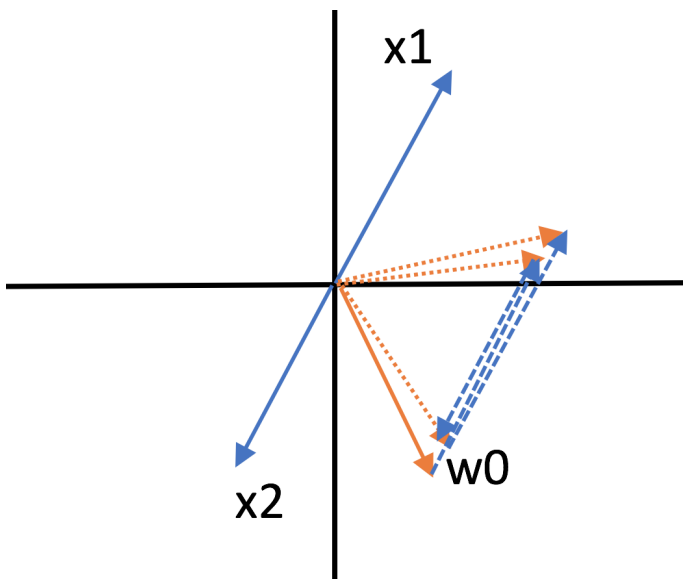


Perceptron Review

- slow adjustment if $|w| >> |x| \Rightarrow$ solution: normalize vectors

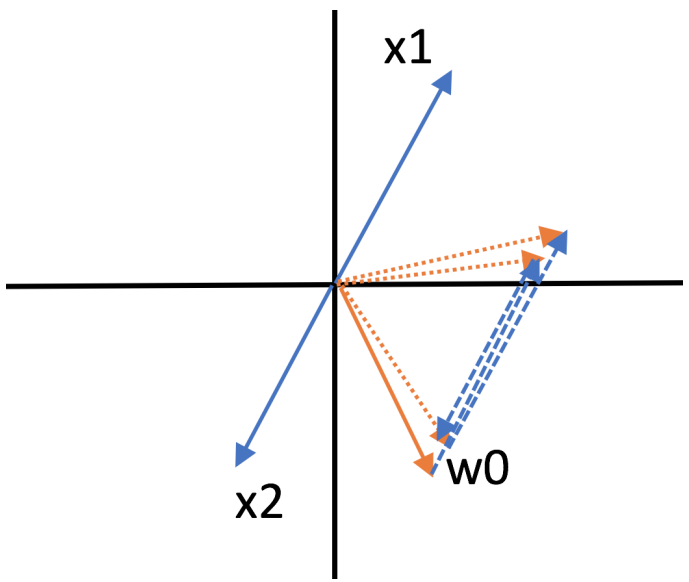
Perceptron Review

- slow adjustment if $|w| >> |x| \Rightarrow$ solution: normalize vectors
- collinear x vectors will result in oscillation $\Rightarrow$ slow convergence



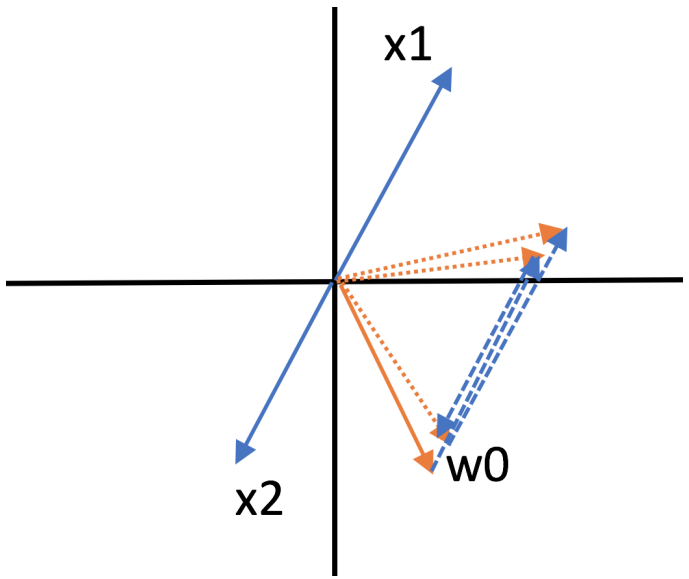
Perceptron Review

- slow adjustment if $|w| >> |x| \Rightarrow$ solution: normalize vectors
- collinear x vectors will result in oscillation $\Rightarrow$ slow convergence
- if samples are not linearly separable there's no convergence $\Rightarrow$ use delta-rule to overcome this issue



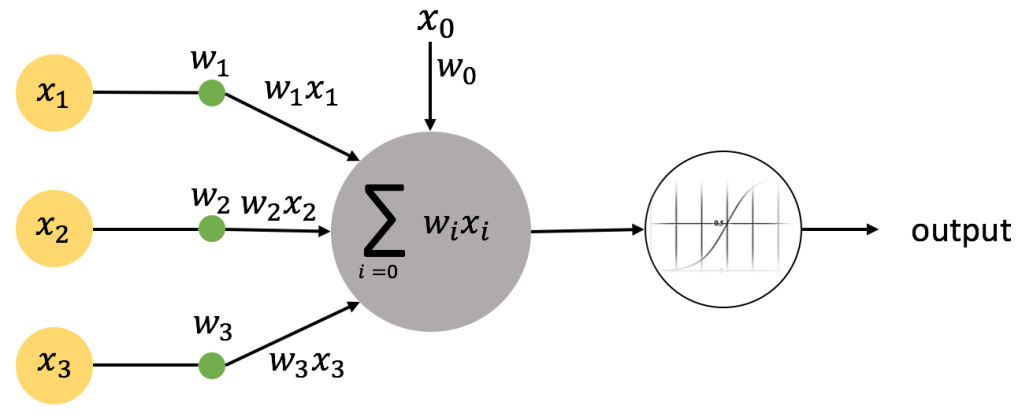
Perceptron Review

- slow adjustment if $|w| >> |x| \Rightarrow$ solution: normalize vectors
- collinear x vectors will result in oscillation $\Rightarrow$ slow convergence
- if samples are not linearly separable there's no convergence $\Rightarrow$ use delta-rule to overcome this issue
- the delta-rule is an update-rule that uses gradient descent to search the hypothesis space and to find the weights that best fit the samples (not discussed here)



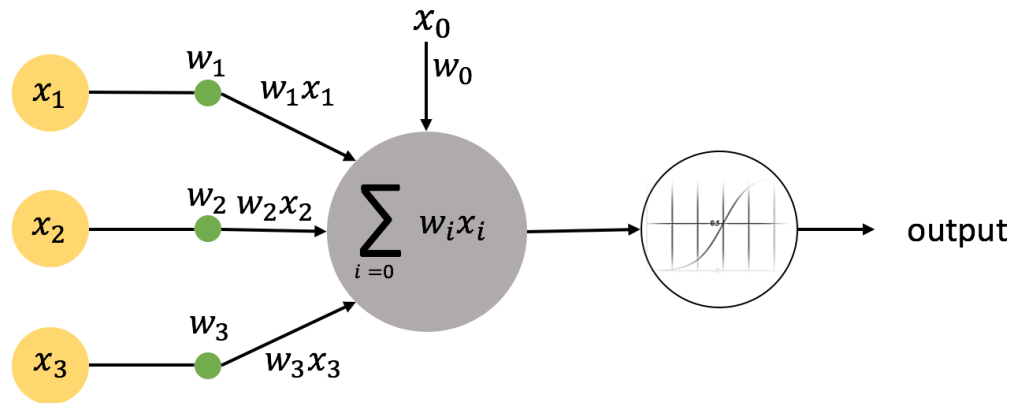
Relation between Logistic Regression and NN

artificial neuron: (arbitrary activation function)



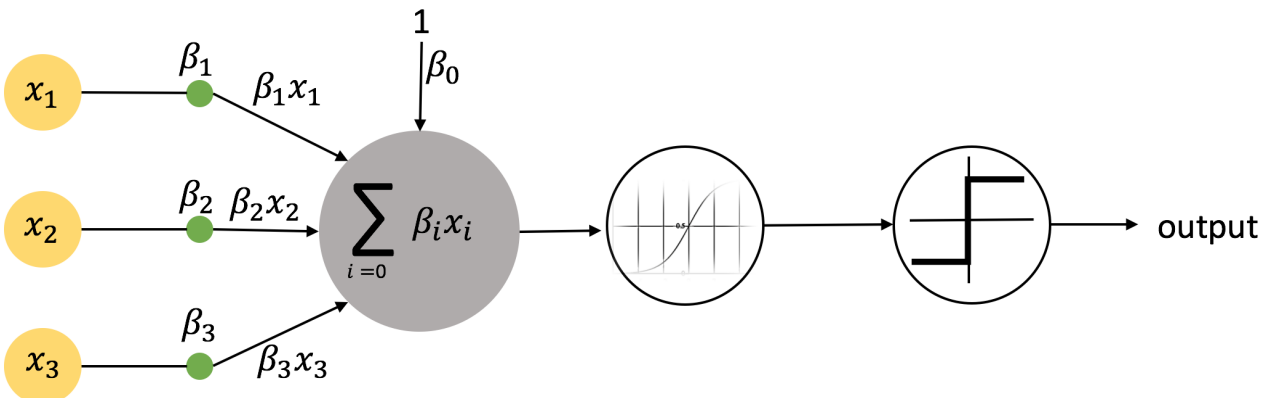
Relation between Logistic Regression and NN

artificial neuron: (arbitrary activation function)



input weights input function activation function

logistic regression: (logistic function as activation)



input weights input function activation function unit step function

Activation Functions

- every hidden neuron usually has an activation function
- usually maps the resulting values to values between 0 and 1 or -1 and 1
- can be linear or non-linear

Activation Functions

- every hidden neuron usually has an activation function
- usually maps the resulting values to values between 0 and 1 or -1 and 1
- can be linear or non-linear

Why activation functions?

Activation Functions

- every hidden neuron usually has an activation function
- usually maps the resulting values to values between 0 and 1 or -1 and 1
- can be linear or non-linear

Why activation functions?

- functions we want to model are typically non-linear

Activation Functions

- every hidden neuron usually has an activation function
- usually maps the resulting values to values between 0 and 1 or -1 and 1
- can be linear or non-linear

Why activation functions?

- functions we want to model are typically non-linear
- increase the capacity of NNs and thus, the complexity of their functions

Activation Functions

- every hidden neuron usually has an activation function
- usually maps the resulting values to values between 0 and 1 or -1 and 1
- can be linear or non-linear

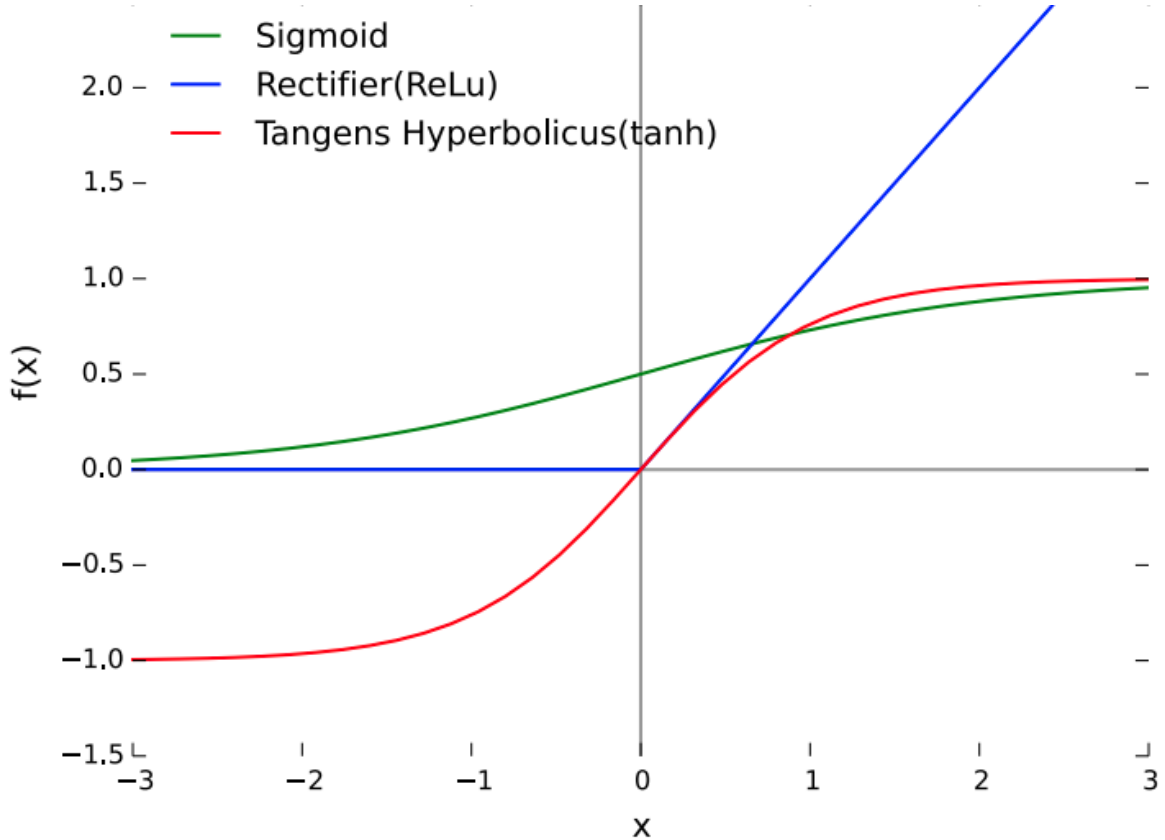
Why activation functions?

- functions we want to model are typically non-linear
- increase the capacity of NNs and thus, the complexity of their functions
- a multi-layer NN with just linear AF behaves like a single-layer NN[1] $\Rightarrow$ make actual use of network depth

[1] Universal approximation theorem

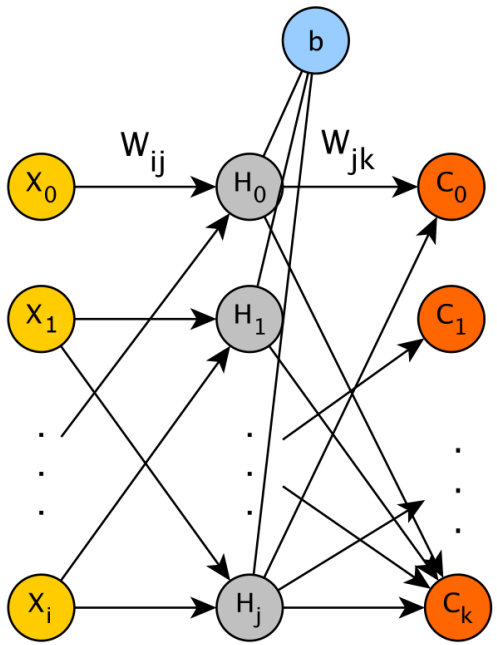
Activation Functions

Some activation functions:



important: since we use backpropagation to train NNs, we require AFs to be differentiable

Output Neurons and Classification



- k output neurons allow classification for k different classes
- **healthy intuition:** layers prior to the output layer map the data to a higher dimensional representation that (ideally) is linearly separable by the output neurons[2]

[2] which form a (k-1)-dimensional hyperplane in the k-dimensional space

Multi-layer Neural Networks and Deep Learning

- a NN with at least three (input, hidden, output) is considered to be a multi-layer NN

Multi-layer Neural Networks and Deep Learning

- a NN with at least three (input, hidden, output) is considered to be a multi-layer NN
- **what's the difference between neural networks and deep neural networks?**

Multi-layer Neural Networks and Deep Learning

- a NN with at least three (input, hidden, output) is considered to be a multi-layer NN
- **what's the difference between neural networks and deep neural networks?**
  - no definite answer to what "deep" is
  - usually more than two or three hidden layers are called deep
  - NN with no or one hidden layer is called "shallow"
  - note: the term "deep" is time-dependent as NN tend to get deeper over time

Multi-layer Neural Networks and Deep Learning

- a NN with at least three (input, hidden, output) is considered to be a multi-layer NN
- **what's the difference between neural networks and deep neural networks?**
  - no definite answer to what "deep" is
  - usually more than two or three hidden layers are called deep
  - NN with no or one hidden layer is called "shallow"
  - note: the term "deep" is time-dependent as NN tend to get deeper over time
- **why go deep (and not wide)?**

Multi-layer Neural Networks and Deep Learning

- a NN with at least three (input, hidden, output) is considered to be a multi-layer NN
- **what's the difference between neural networks and deep neural networks?**
  - no definite answer to what "deep" is
  - usually more than two or three hidden layers are called deep
  - NN with no or one hidden layer is called "shallow"
  - note: the term "deep" is time-dependent as NN tend to get deeper over time
- **why go deep (and not wide)?**
  - despite the universal approximation theorem (one "large" hidden layer suffices), deep NN work better in practice across many domains
  - the following slides focus on deep NN but can be applied to shallow NN, too

Cost function

- remember: a **loss function** quantifies the deviation between a classification result and a target value w.r.t. a single sample

Cost function

- remember: a **loss function** quantifies the deviation between a classification result and a target value w.r.t. a single sample
- a **cost function** quantifies the loss w.r.t the entire dataset

Cost function

- remember: a **loss function** quantifies the deviation between a classification result and a target value w.r.t. a single sample
- a **cost function** quantifies the loss w.r.t the entire dataset
- a set of parameters (weights) are considered optimized if the costs are 0 or can not be further reduced (convergence criteria)

Cost function

- remember: a **loss function** quantifies the deviation between a classification result and a target value w.r.t. a single sample
- a **cost function** quantifies the loss w.r.t the entire dataset
- a set of parameters (weights) are considered optimized if the costs are 0 or can not be further reduced (convergence criteria)
- the loss function forms an "error surface" in which our optimization operates

Cost function

- remember: a **loss function** quantifies the deviation between a classification result and a target value w.r.t. a single sample
- a **cost function** quantifies the loss w.r.t the entire dataset
- a set of parameters (weights) are considered optimized if the costs are 0 or can not be further reduced (convergence criteria)
- the loss function forms an "error surface" in which our optimization operates
- optimization is done via the **backpropagation algorithm**

Cost function

- remember: a **loss function** quantifies the deviation between a classification result and a target value w.r.t. a single sample
- a **cost function** quantifies the loss w.r.t the entire dataset
- a set of parameters (weights) are considered optimized if the costs are 0 or can not be further reduced (convergence criteria)
- the loss function forms an "error surface" in which our optimization operates
- optimization is done via the **backpropagation algorithm**
- typical cost functions are:
  - Mean Squared Error (MSE)
  - SVM (cost function)
  - **Softmax**

Softmax Cost Function

- the Softmax cost function is a generalization of the binary logistic regression to multiple classes[3]
- Softmax yields normalized class probabilities $\Rightarrow$ probabilistic interpretation
- let's first take a look at the **softmax function**:

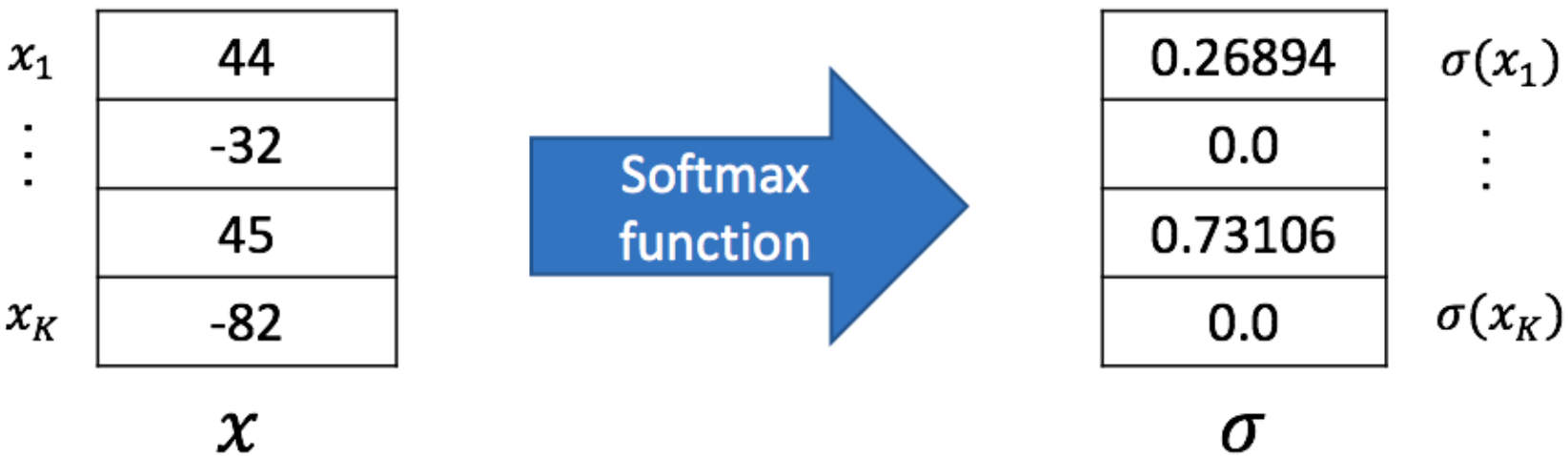
$$\sigma(x_i) = \frac{e^{x_i}}{\sum_k^K e^{x_k}} \tag{1}$$

$\Rightarrow$ it squashes a K-dimensional vector x of arbitrary real values to a K-dimensional vector $\sigma(x)$ of real values in the range (0,1) that add up to 1

[3] non-linear variant of the multinomial logistic regression

Softmax Cost Function

Softmax function example:



Softmax Cost Function

- now let $f(x_i; W) = Wx_i$ be the score function with
  - W the weights (and biases) from the NN
  - and the input vector x_i

Softmax Cost Function

- now let $f(x_i; W) = Wx_i$ be the score function with
  - W the weights (and biases) from the NN
  - and the input vector x_i
- we use the softmax function to build the loss function

$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j^K e^{f_j}}\right) \tag{2}$$

where y_i chooses the index of the correct label at step i and f_j is the j-th element of the class scores vector f

Softmax Cost Function

- now let $f(x_i; W) = Wx_i$ be the score function with
  - W the weights (and biases) from the NN
  - and the input vector x_i
- we use the softmax function to build the loss function

$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j^K e^{f_j}}\right) \tag{2}$$

  where y_i chooses the index of the correct label at step i and f_j is the j-th element of the class scores vector f
- intuitively, L_i yields the normalized probability of class y for a given input x_i and parameters W

Softmax Cost Function

- now let $f(x_i; W) = Wx_i$ be the score function with
  - W the weights (and biases) from the NN
  - and the input vector x_i
- we use the softmax function to build the loss function

$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j^K e^{f_j}}\right) \tag{2}$$

  where y_i chooses the index of the correct label at step i and f_j is the j-th element of the class scores vector f
- intuitively, L_i yields the normalized probability of class y for a given input x_i and parameters W
- the cost for the data is the mean of L_i over all training samples

Softmax Cost Function

- to complete our intuition we consider the cross entropy:

Softmax Cost Function

- to complete our intuition we consider the cross entropy:

$$H(p, q) = -\sum_{k} p(x) \log q(x) \tag{3}$$

Softmax Cost Function

- to complete our intuition we consider the cross entropy:

$$H(p, q) = -\sum_k p(x) \log q(x) \tag{3}$$

- set $L_i = \log q(x)$
- $p = [0, ..., 1, ...0]^T$ with a single 1 at y_i-th position

Softmax Cost Function

- to complete our intuition we consider the cross entropy:

$$H(p, q) = -\sum_k p(x) \log q(x) \tag{3}$$

- set $L_i = \log q(x)$
- $p = [0, ..., 1, ...0]^T$ with a single 1 at y_i-th position
- $\Rightarrow$ the softmax classifier minimizes the cross-entropy between the log probability of the predicted class y_i and the true distribution p
- $\Rightarrow$ during training, the predicted distribution will be enforced to have all of its mass on the correct class

Softmax Cost Function

- to complete our intuition we consider the cross entropy:

$$H(p, q) = -\sum_k p(x) \log q(x) \tag{3}$$

- set $L_i = \log q(x)$
- $p = [0, ..., 1, ...0]^T$ with a single 1 at y_i-th position
- $\Rightarrow$ the softmax classifier minimizes the cross-entropy between the log probability of the predicted class y_i and the true distribution p
- $\Rightarrow$ during training, the predicted distribution will be enforced to have all of its mass on the correct class
- minimizing the negative log prob is equal to maximizing the log prob (of choosing the correct class)

Softmax Cost Function Example

0.05	0.07	0.02	0.03
0.012	0.04	0.21	0.13
0.005	0.08	0.18	0.032

W

$\times$

44
-32
55
-82

x

$=$

-1.4
0.138
4.936

f

Softmax Cost Function Example

0.05	0.07	0.02	0.03		44		-1.4
0.012	0.04	0.21	0.13	×	-32	=	0.138
0.005	0.08	0.18	0.032		55		4.936
					-82		
	W				x		f

Softmax Cost Function Example

true class index: 2

0.05	0.07	0.02	0.03
0.012	0.04	0.21	0.13
0.005	0.08	0.18	0.032

W

$\times$

44
-32
55
-82

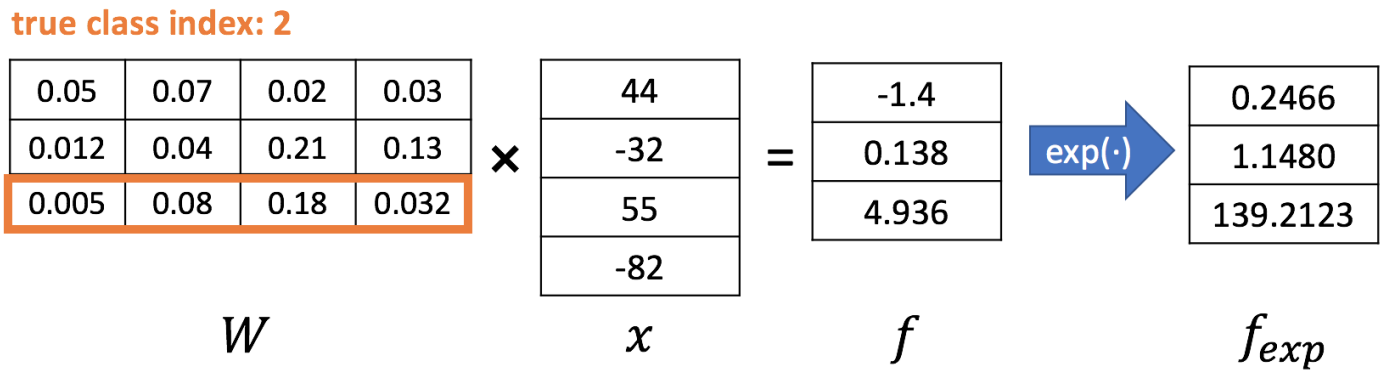
x

$=$

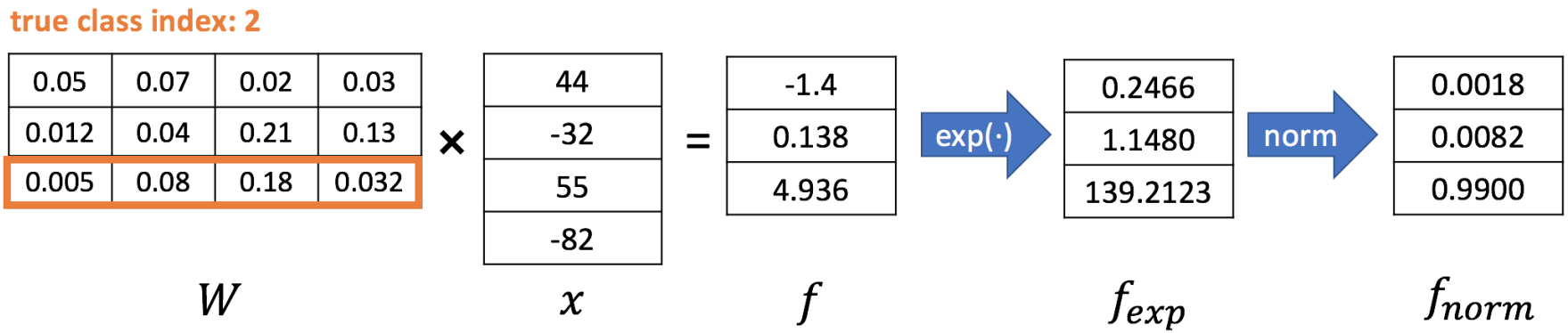
-1.4
0.138
4.936

f

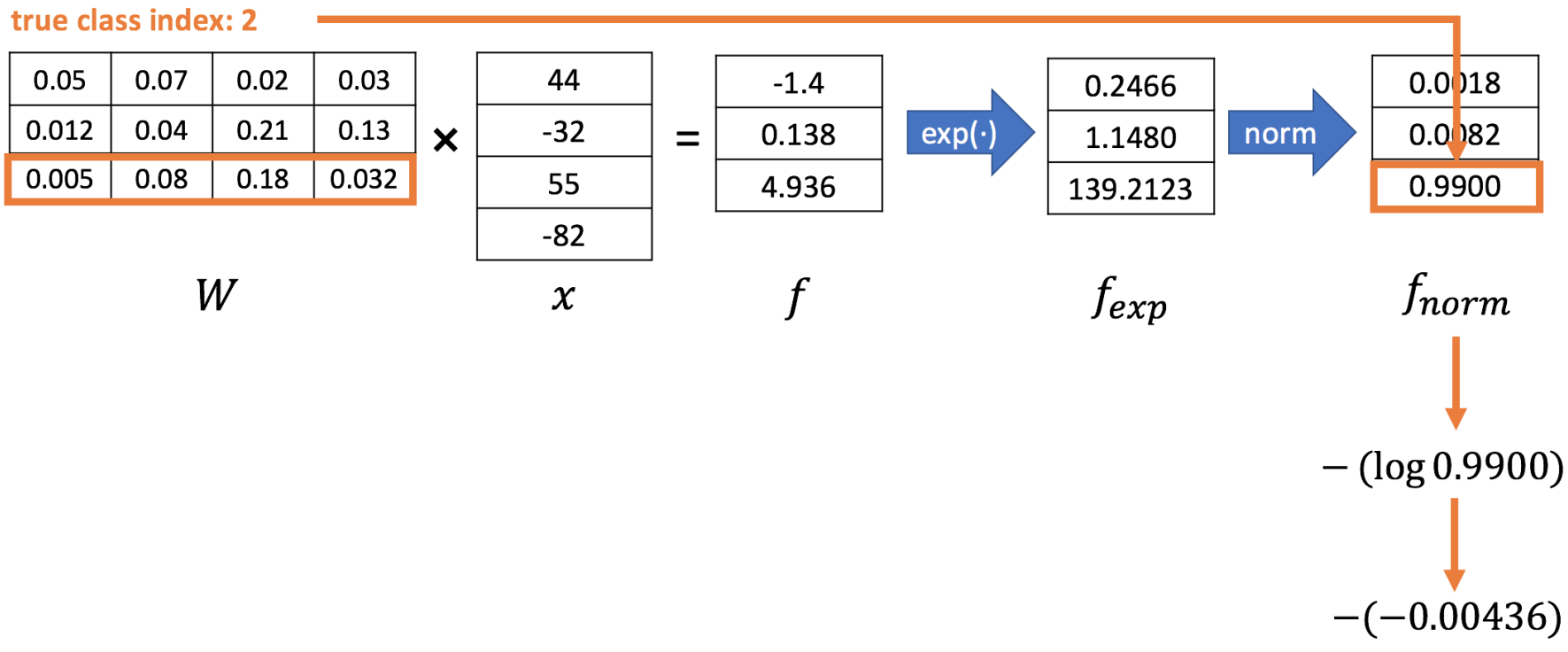
Softmax Cost Function Example



Softmax Cost Function Example



Softmax Cost Function Example

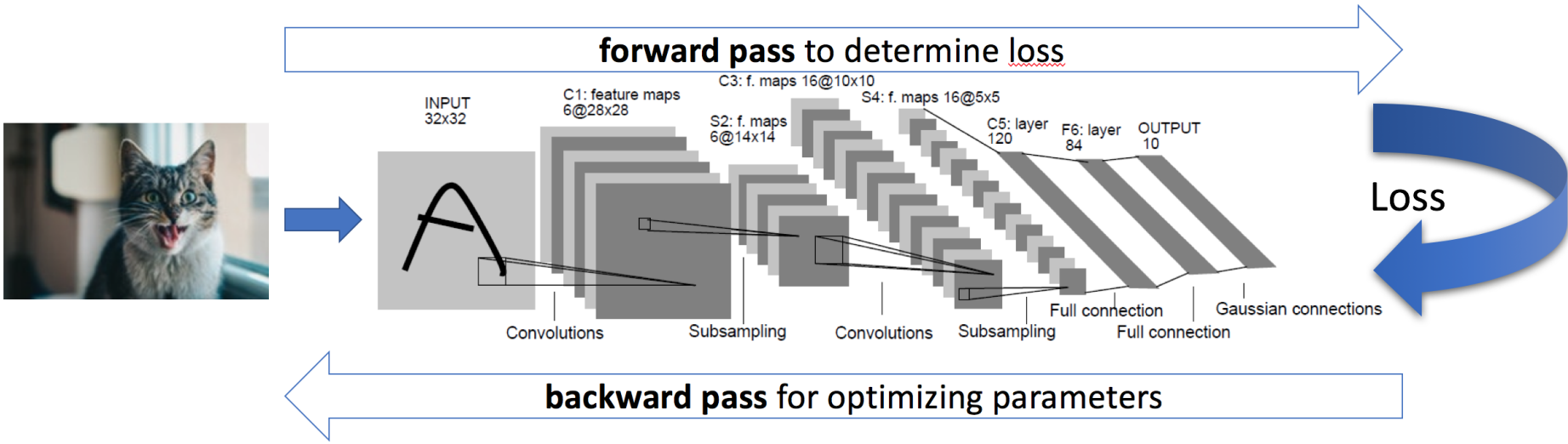


Backpropagation Algorithm

- motivation: the cost function is in most cases non-linearly dependent on the parameters of a multi-layer NN
  $\Rightarrow$ computing closed-form solution intractable or not possible
- backpropagation executes two steps iteratively:
  1. forward-pass: computes the loss L_i and its gradient
  2. backward-pass: updates W with gradient descent $\Rightarrow$ minimizes L_i iteratively [4]

[4] repeatedly applying gradient descent within backprop is sometimes called *stochastic gradient descent*, *stochastic* here usually refers to "on-line"

Backpropagration Algorithm



source: Yann LeCun et al., „Gradient-Based Learning Applied to Document Recognition“

Connection between Backprop and Learning Theory class

Q: What is the solver actually doing when executing the backpropagation algorithm?

Connection between Backprop and Learning Theory class

Q: What is the solver actually doing when executing the backpropagation algorithm?
A:it executes a search within the hypothesis space for a suitable hypothesis (remember hypothesis space theory class)

Connection between Backprop and Learning Theory class

Q: What is the solver actually doing when executing the backpropagation algorithm?
A:it executes a search within the hypothesis space for a suitable hypothesis (remember hypothesis space theory class)
Q: What are the hypotheses?

Connection between Backprop and Learning Theory class

Q: What is the solver actually doing when executing the backpropagation algorithm?
A:it executes a search within the hypothesis space for a suitable hypothesis (remember hypothesis space theory class)
Q: What are the hypotheses?
A: the individual weight vector per iteration (relative to the training examples)[5]. More precisely: the hypothesis space contains **continuously** parameterized hypotheses (for example the weights in the neurons)

[5]this also holds for weight vectors in Logistic Regression computed by Gradient Descent methods

Connection between Backprop and Learning Theory class

Q: What is the solver actually doing when executing the backpropagation algorithm?
A:it executes a search within the hypothesis space for a suitable hypothesis (remember hypothesis space theory class)
Q: What are the hypotheses?
A: the individual weight vector per iteration (relative to the training examples)[5]. More precisely: the hypothesis space contains **continuously** parameterized hypotheses (for example the weights in the neurons)
The **inductive bias** by which backprop generalizes beyond seen data is a bit more tricky: it's the smooth interpolation between data points (i.e. labels between two positive samples will also be positive

[5]this also holds for weight vectors in Logistic Regression computed by Gradient Descent methods

Backpropagation Algorithm: the Backward Pass

- To update the weights, Backpropagation computes the direction of the steepest descent in the n-dimensional space:

$$\nabla L(W) = \frac{\partial L}{\partial w_0}, \frac{\partial L}{\partial w_1}, ..., \frac{\partial L}{\partial w_n} \tag{4}$$

- We use the following (or a more sophisticated) rule to update the weights of every single neuron[6]:

$$w_{t+1} = w_t + v_{t+1} \tag{5}$$

$$v_{t+1} = -\alpha \frac{\partial L}{\partial w_t} \tag{6}$$

with w_t being the parameter vector at time step t and α the learning rate

[6] called vanilla update

Backpropagation Algorithm: the Backward Pass

- computing the gradients between the output and the last hidden layer is straightforward

with z_j being the output of neuron j

Backpropagation Algorithm: the Backward Pass

- computing the gradients between the output and the last hidden layer is straightforward
- the gradients of L w.r.t to the weights $w_{j,i}$ (weight between neuron i to j) previous to the last hidden layers are **unknown** at first

with z_j being the output of neuron j

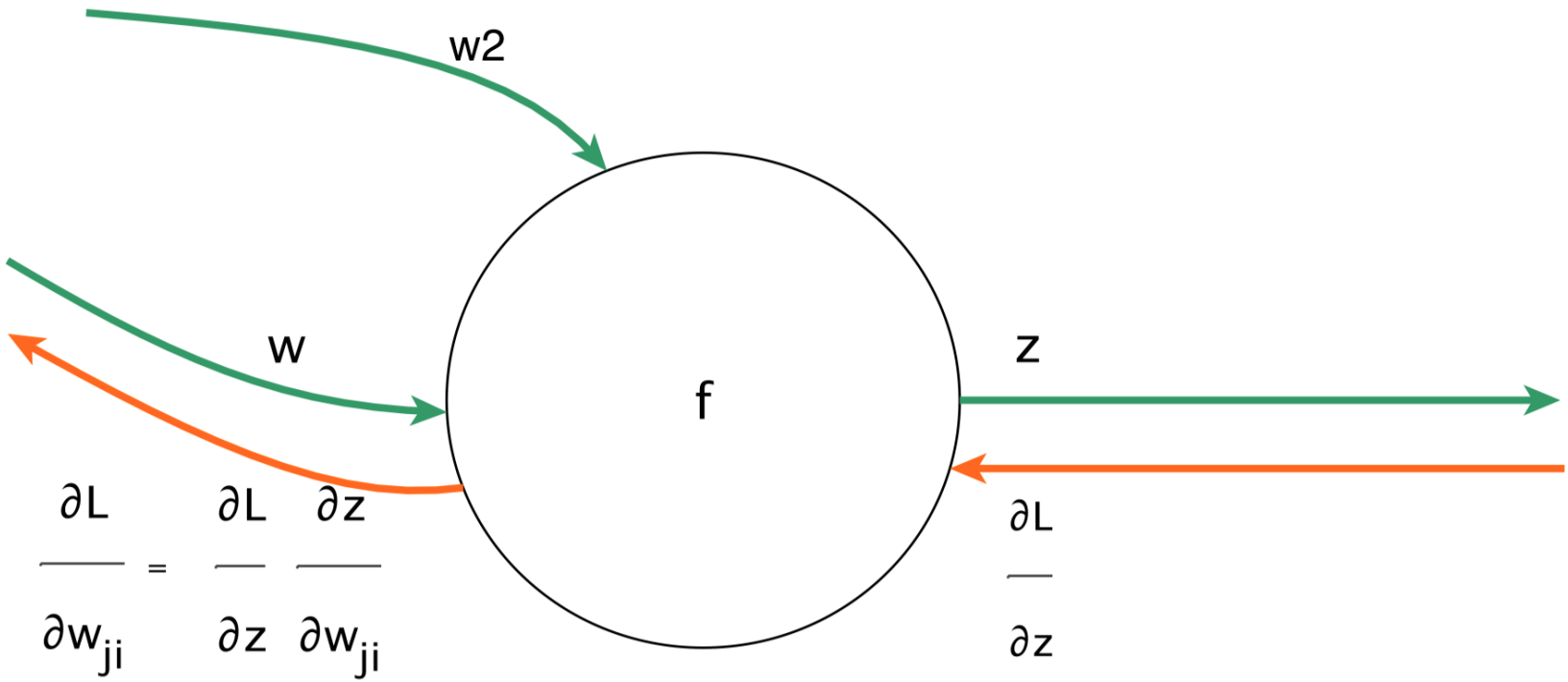
Backpropagation Algorithm: the Backward Pass

- computing the gradients between the output and the last hidden layer is straightforward
- the gradients of L w.r.t to the weights $w_{j,i}$ (weight between neuron i to j) previous to the last hidden layers are **unknown** at first
- solution: we use the **chain rule** to compute the gradients between the last and penultimate layer, the penultimate and the layer previous to that etc.

$$\frac{\partial L}{\partial w_{j,i}} = \frac{\partial L}{\partial z_j} \times \frac{\partial z_j}{\partial w_{j,i}} \tag{7}$$

with z_j being the output of neuron j

Backpropagation Algorithm: the Backward Pass



lecture recommendation for backprop and chain rule (Stanford's CS231n course)[7]

[7] https://www.youtube.com/watch?v=i940vYb6noo&t=322s

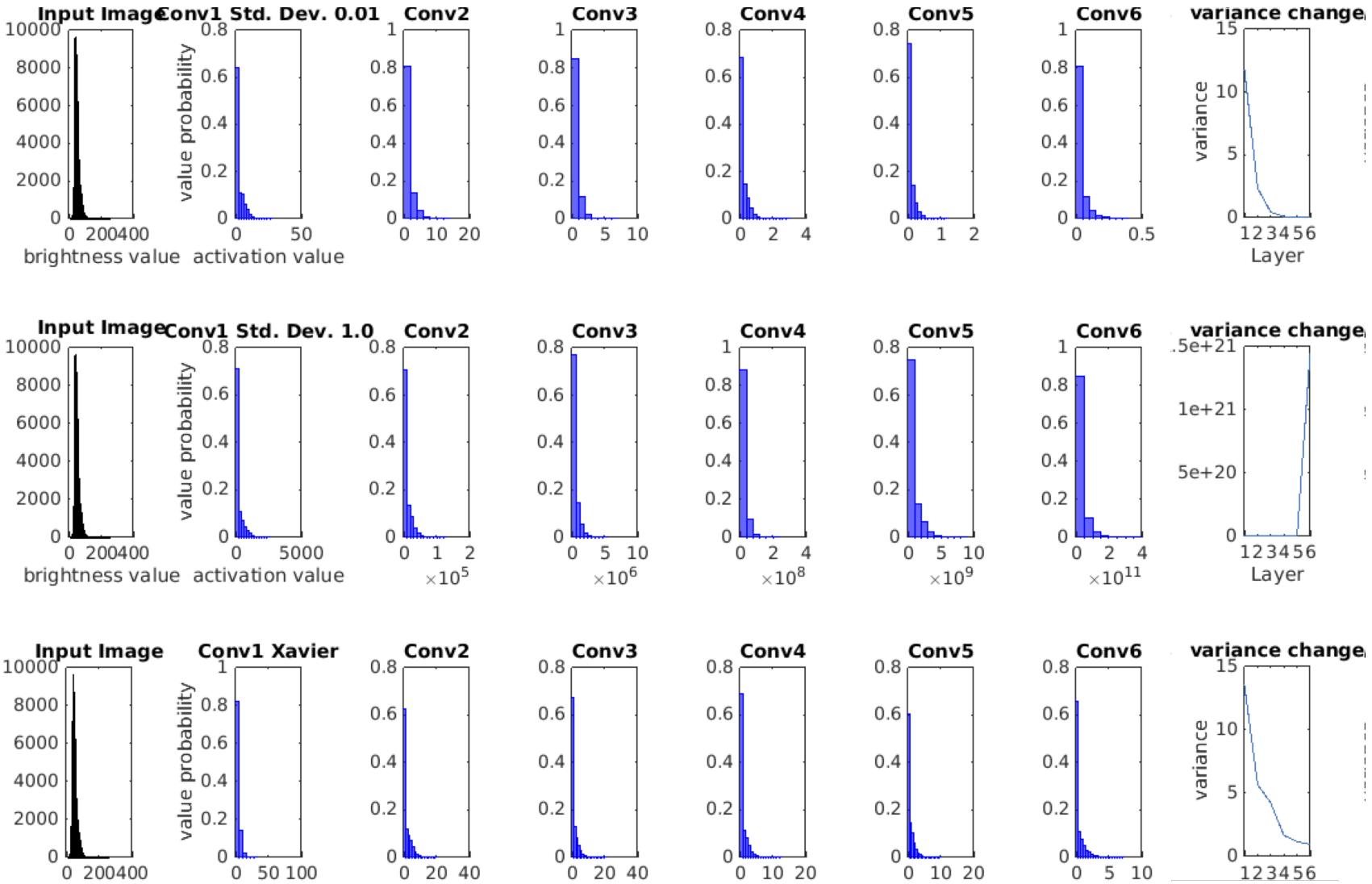
Extensions to Learning NN and Problems

- the weight update rule is often subject to change since it tends to get stuck in local minima
- exemplary extension; momentum update:

$$v_{t+1} = -\alpha \frac{\partial L}{\partial w_t} + \mu v_t \tag{8}$$

  $\mu \in [0, 1]$
- initialization of weights matter (see next slide)
  $\Rightarrow$ **vanishing gradient** and **exploding gradient problem**
- use initialization methods, e.g. *Xavier* initialization
- **saturating activation functions** can cause problems as well
- small/large learning rates $\Rightarrow$ common: adaptive /decreasing LR

Vanishing Gradients



Regularization

- **regularization** is a collective term for methods that reduce overfitting
- regularization through a **penalty term** $R(W)$ (weight decay):

$$L = \sum_i L_i + \lambda R(W) \tag{9}$$

  - L1: $R(W) = \lambda \sum_w |w|$
  - L2: $R(W) = \lambda \sum_w |w|^2$
  - typically $\lambda < 0.1$
- L1 allows concentrations within weight distributions (focus on some features)
- L2 penalizes outliers more strongly, fewer concentrations (focus on all features)

Regularization

- R(W) enforces a NN to have small weights
  $\Rightarrow$ have many small weights instead of few big weights (more inputs considered)

further regularization techniques:

- **dropout**: remove dependencies among neurons
- **data augmentation**: enlarge small datasets
- **architecture size**: decrease number of parameters to reduce risk of learning irrelevant features

Training and Testing Phases

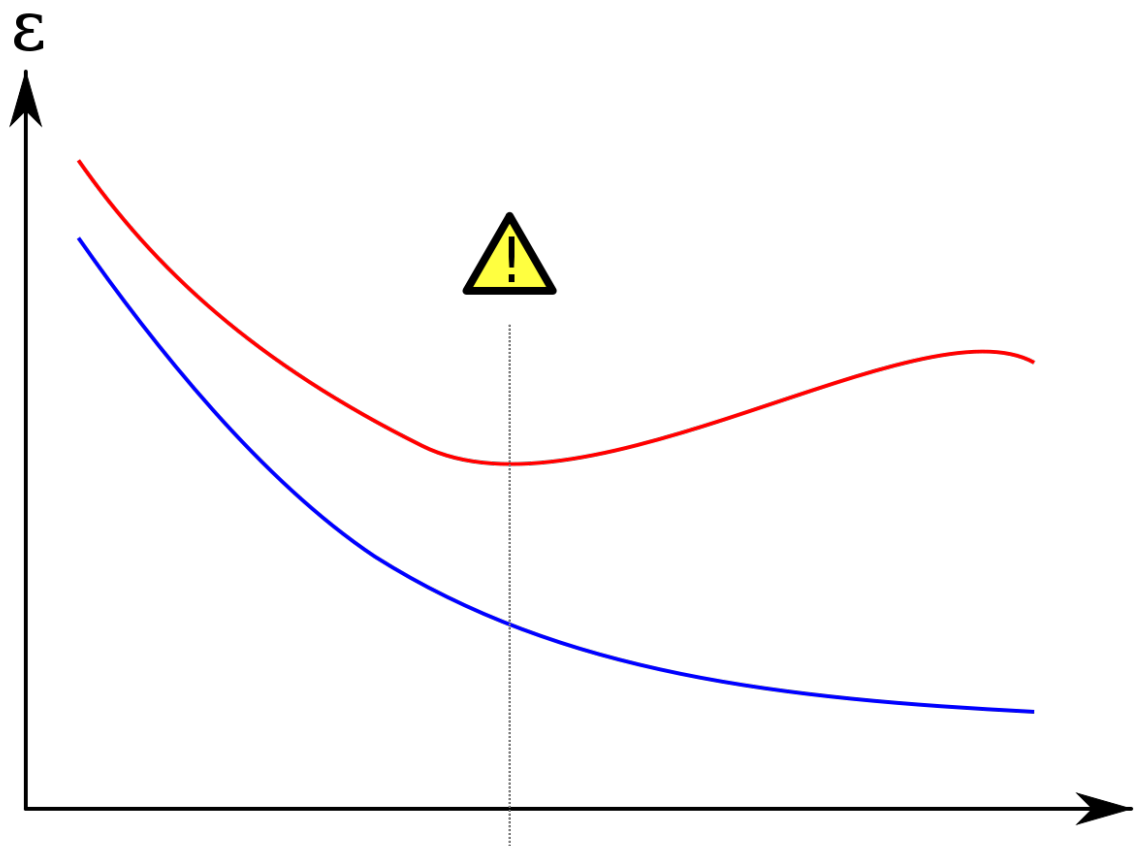
Training a NN gives rise to one particular question: when are we done with training? Best practice is:

- have a train / valid and test split of your data
- typically do validation phase after n training iterations
- plot or print train and valid loss[8]
- both losses shouldn't diverge (overfitting)

[8] or accuracy, prediction, recall, f-score etc.

Training and Testing Phases

Optimal stopping problem: don't stop too early but also not too late



(blue: training loss, red: validation loss) [Encyclopedia, 2016]

Table of Contents

Convolutional Neural Networks

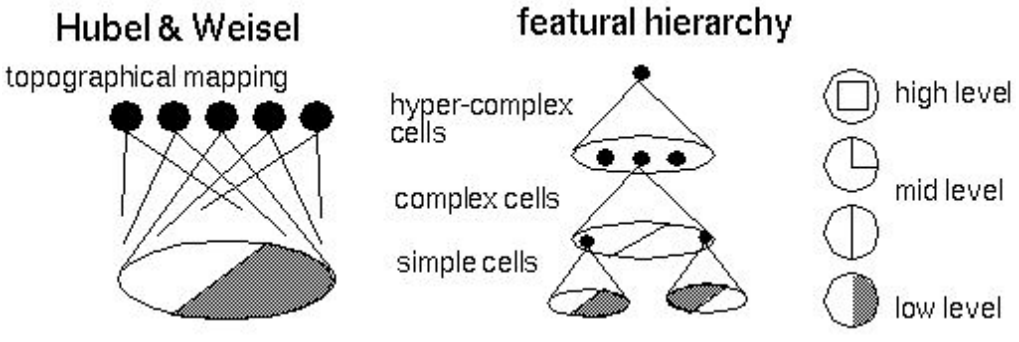
- CNN's are a variant of multi-layer NN

Convolutional Neural Networks

- CNN's are a variant of multi-layer NN
- their design is inspired by biological processes
  $\Rightarrow$ neurons in the animal visual cortex respond to stimuli only in a restricted area

Convolutional Neural Networks

- CNN's are a variant of multi-layer NN
- their design is inspired by biological processes
  $\Rightarrow$ neurons in the animal visual cortex respond to stimuli only in a restricted area
- main difference to NN: neurons are not fully connected to underlying and overlying neighbor neurons
  $\Rightarrow$ "receptive field"

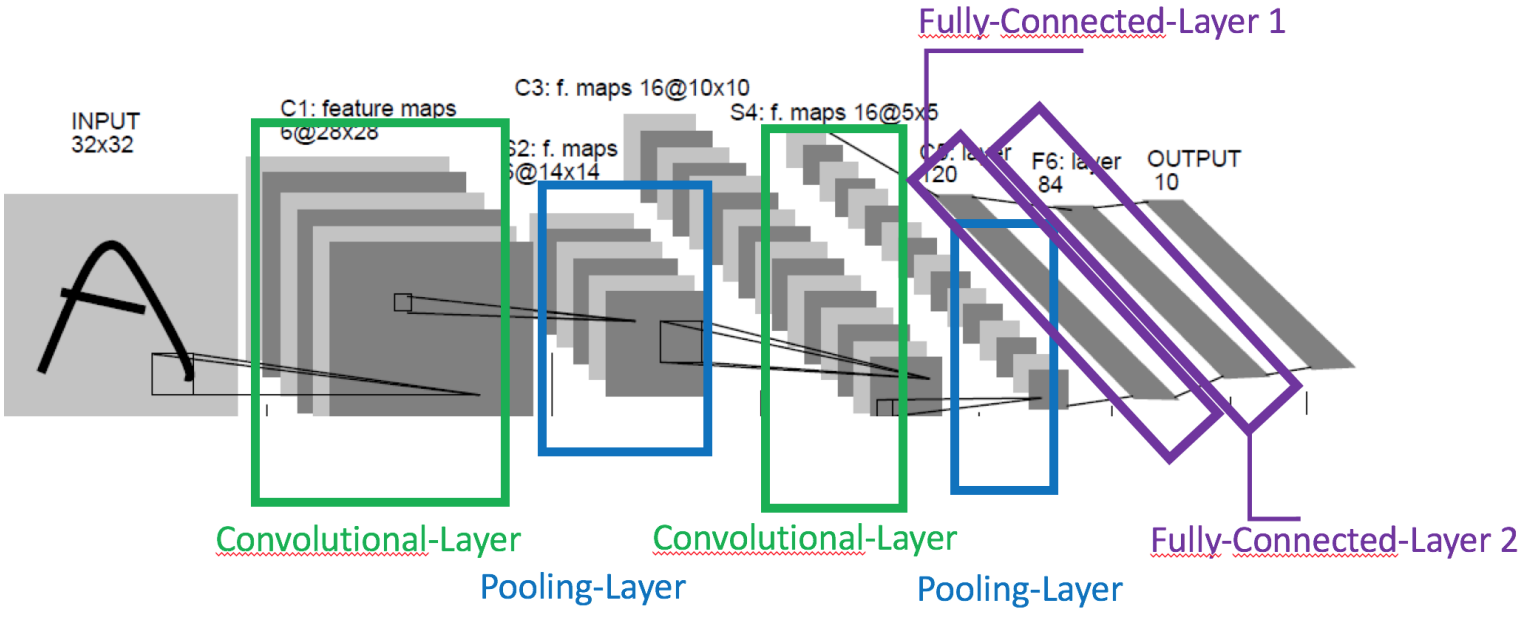


[T.N. Wiesel, 1959]

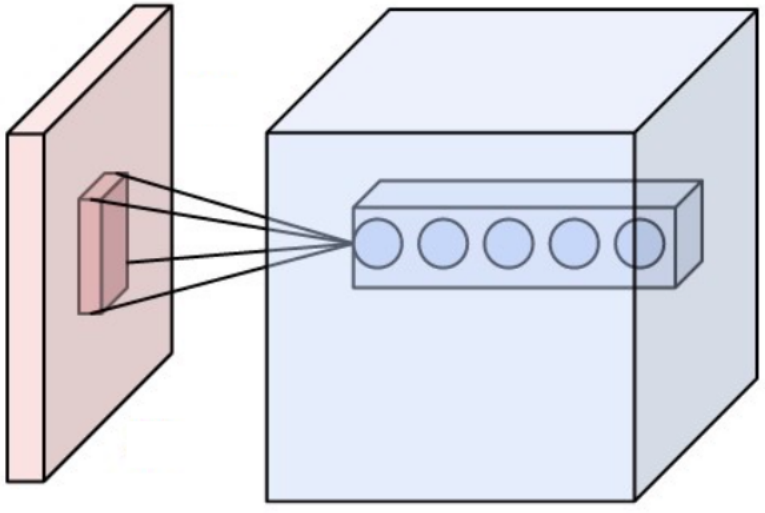
Convolutional Neural Networks

typically use three types of layers

- convolutional layer (conv layer)
- pooling layer
- fully-connected layer (fc layer)



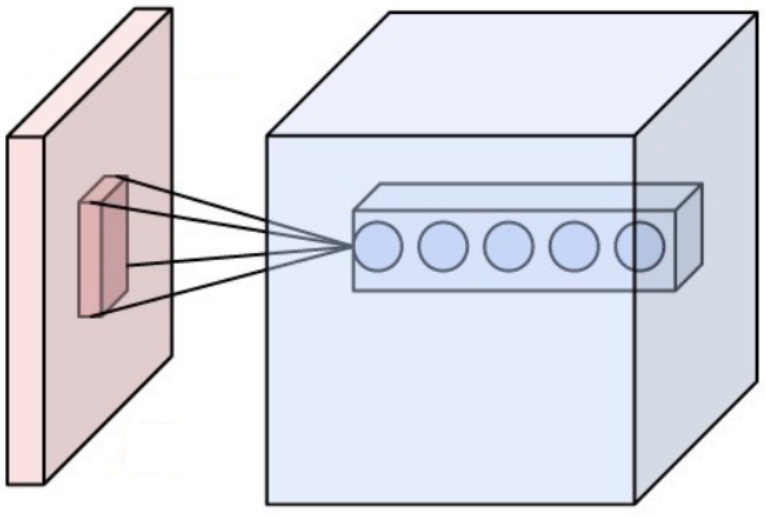
CNN's: Convolutional Layer



[Wikipedia: The Free Encyclopedia, 2018]

- receptive field is slided over the input

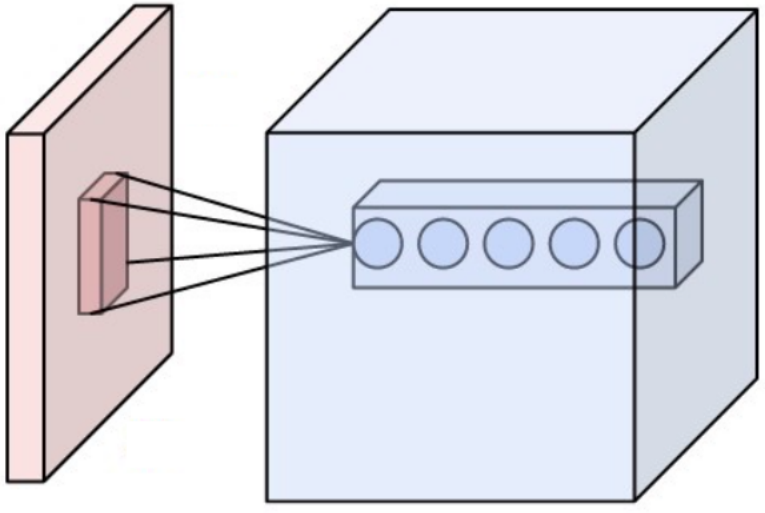
CNN's: Convolutional Layer



[Wikipedia: The Free Encyclopedia, 2018]

- receptive field is slided over the input
- the parameters learned in a conv layer are the elements of the filter matrices

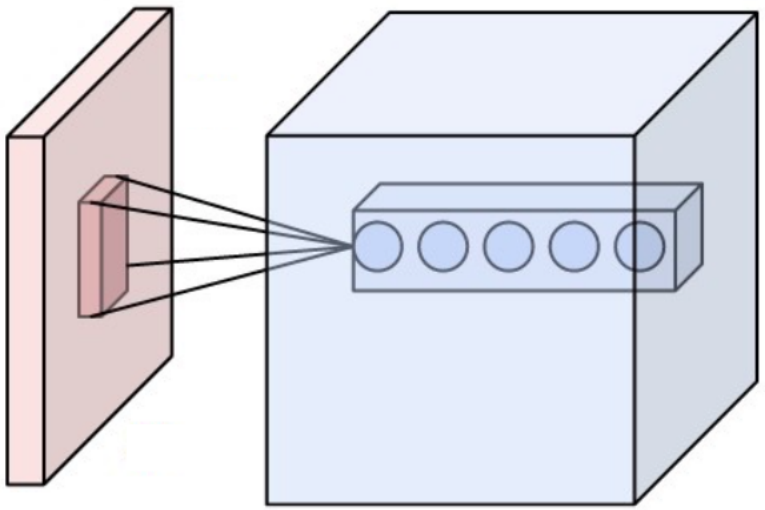
CNN's: Convolutional Layer



[Wikipedia: The Free Encyclopedia, 2018]

- receptive field is slided over the input
- the parameters learned in a conv layer are the elements of the filter matrices
- depth of the "block" is defined by the number of filters

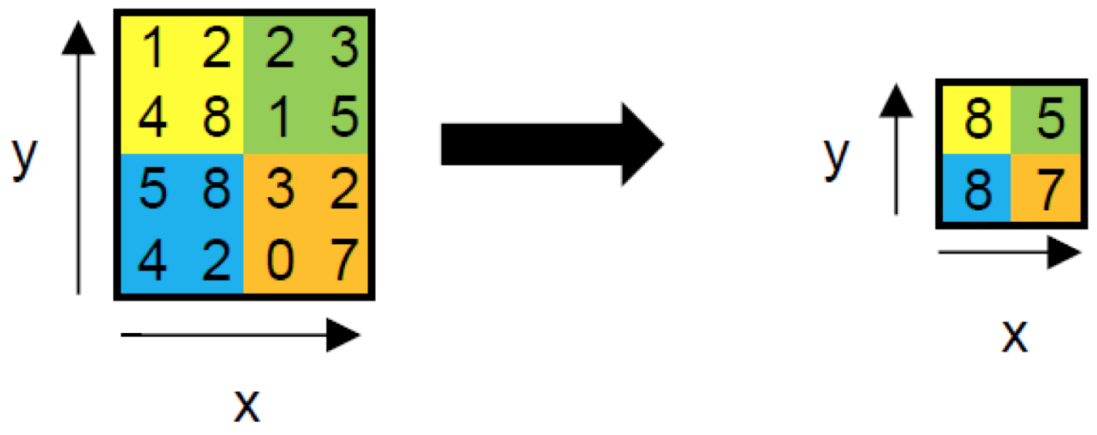
CNN's: Convolutional Layer



[Wikipedia: The Free Encyclopedia, 2018]

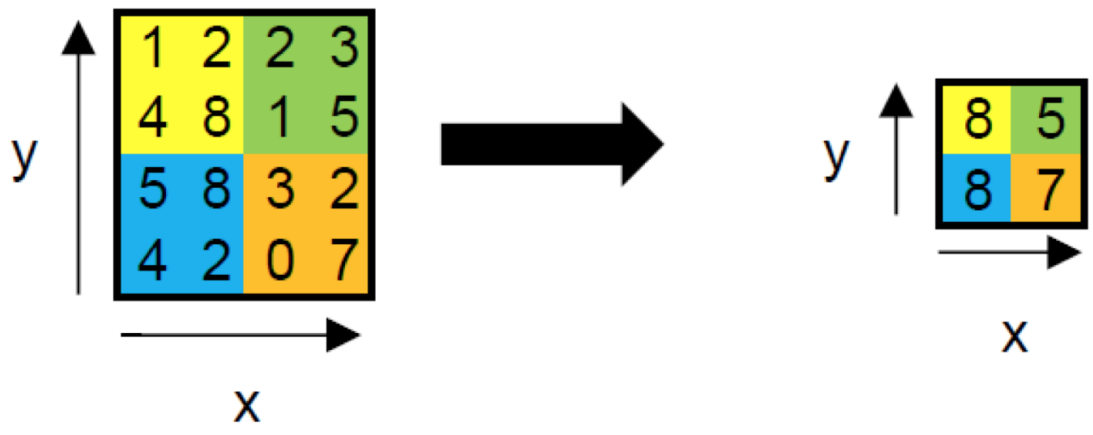
- receptive field is slided over the input
- the parameters learned in a conv layer are the elements of the filter matrices
- depth of the "block" is defined by the number of filters
- detects e.g. edges, circles, cat ears etc.

CNN's: Pooling Layer



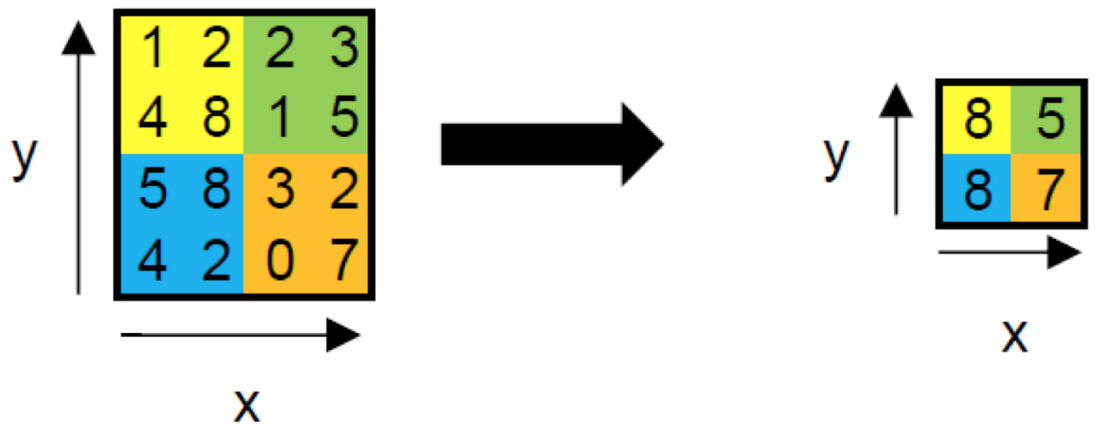
- reduces the number of learned parameters

CNN's: Pooling Layer



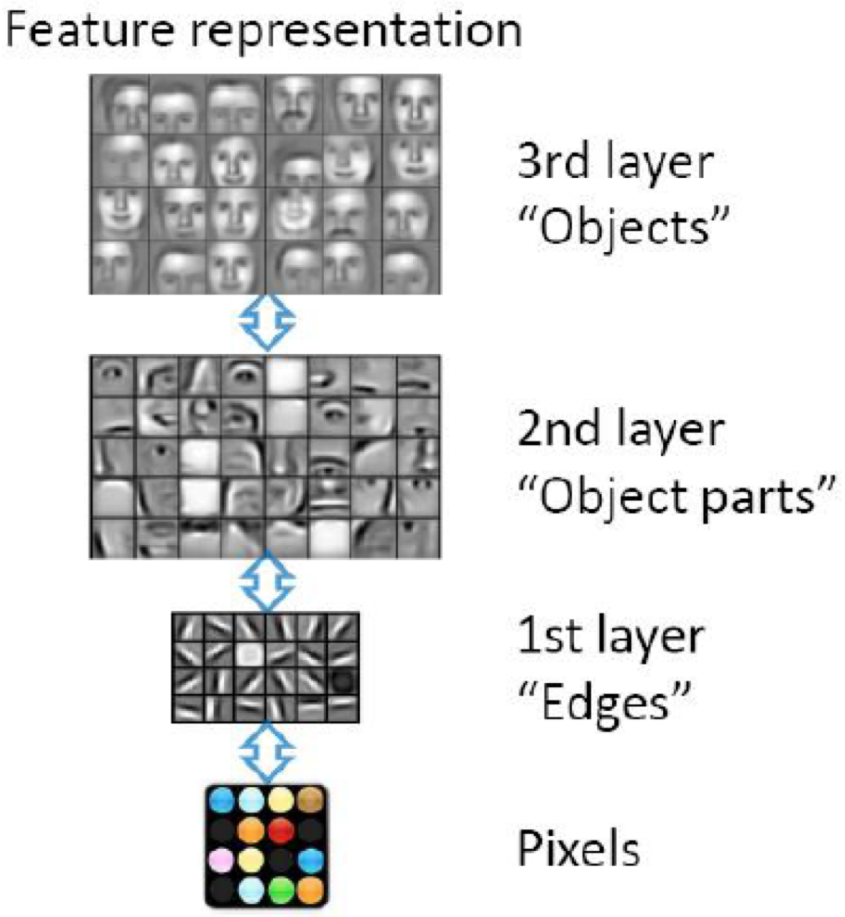
- reduces the number of learned parameters
- achieves invariance to small translations in the input / features

CNN's: Pooling Layer



- reduces the number of learned parameters
- achieves invariance to small translations in the input / features
- maximum and average pooling most common

CNN's: features visualized



source: Honglak Lee, "Tutorial on deep learning and applications"

Table of Contents

Some Advices for Practical Applications

- invest in data-preprocessing, data normalization

Some Advices for Practical Applications

- invest in data-preprocessing, data normalization
- use a healthy dataset split, e.g. 70/20/10 (train/valid/test)

Some Advices for Practical Applications

- invest in data-preprocessing, data normalization
- use a healthy dataset split, e.g. 70/20/10 (train/valid/test)
- a good optimizer (solver) is "Adam" (per-parameter LR & moving averages of the gradient)

Some Advices for Practical Applications

- invest in data-preprocessing, data normalization
- use a healthy dataset split, e.g. 70/20/10 (train/valid/test)
- a good optimizer (solver) is "Adam" (per-parameter LR & moving averages of the gradient)
- use ReLU, PReLU etc.

Some Advices for Practical Applications

- invest in data-preprocessing, data normalization
- use a healthy dataset split, e.g. 70/20/10 (train/valid/test)
- a good optimizer (solver) is "Adam" (per-parameter LR & moving averages of the gradient)
- use ReLU, PReLU etc.
- careful with bounded activation functions (e.g. sigmoid in regression) in the output layer

Some Advices for Practical Applications

- invest in data-preprocessing, data normalization
- use a healthy dataset split, e.g. 70/20/10 (train/valid/test)
- a good optimizer (solver) is "Adam" (per-parameter LR & moving averages of the gradient)
- use ReLU, PReLU etc.
- careful with bounded activation functions (e.g. sigmoid in regression) in the output layer
- 64 or 128 filters for conv layers is plenty

Some Advices for Practical Applications

- invest in data-preprocessing, data normalization
- use a healthy dataset split, e.g. 70/20/10 (train/valid/test)
- a good optimizer (solver) is "Adam" (per-parameter LR & moving averages of the gradient)
- use ReLU, PReLU etc.
- careful with bounded activation functions (e.g. sigmoid in regression) in the output layer
- 64 or 128 filters for conv layers is plenty
- currently popular frameworks: PyTorch, TensorFlow

Some Advices for Practical Applications

- invest in data-preprocessing, data normalization
- use a healthy dataset split, e.g. 70/20/10 (train/valid/test)
- a good optimizer (solver) is "Adam" (per-parameter LR & moving averages of the gradient)
- use ReLU, PReLU etc.
- careful with bounded activation functions (e.g. sigmoid in regression) in the output layer
- 64 or 128 filters for conv layers is plenty
- currently popular frameworks: PyTorch, TensorFlow
- training on GPUs is generally faster than training on CPUs

Some Advices for Practical Applications

- invest in data-preprocessing, data normalization
- use a healthy dataset split, e.g. 70/20/10 (train/valid/test)
- a good optimizer (solver) is "Adam" (per-parameter LR & moving averages of the gradient)
- use ReLU, PReLU etc.
- careful with bounded activation functions (e.g. sigmoid in regression) in the output layer
- 64 or 128 filters for conv layers is plenty
- currently popular frameworks: PyTorch, TensorFlow
- training on GPUs is generally faster than training on CPUs
- try to get a feeling for depth and breadth of NNs/CNNs with the ConvNetJS demo: `https://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html`

When not to use Deep Neural Networks

Deep Learning is not the answer to everything; sometimes simpler models work better. When should you not use it?

- if you don't have enough data
- if it's a low budged project (DL requires many resources such as GPUs, time for training and paper research etc.)
- when traceability is important or when causal mechanisms need to be understood (although research is slowly getting there)

Table of Contents

Application: Training a Classifier

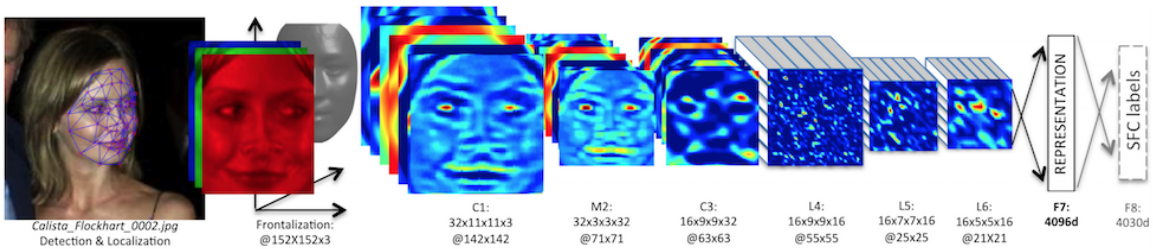
IMO: best way of learning is to experiment around. Therefore, I suggest to try the following tutorial as a voluntary homework:

- `http://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html`
- training of an image classifier with a CNN
- Python novice skills are sufficient
- no GPU required (before downloading, choose "CUDA: None")
- requires PyTorch (mac OS, Linux, Windows)
- uses CIFAR10 dataset; 60 000 32x32 color images of 10 classes
- we can discuss issues either on Moodle or during the upcoming lectures

Table of Contents

Neural Networks in Practice

- CNNs for face recognition [Taigman et al., 2014, "DeepFace"]



- CNNs for human pose estimation [Toshev et al., 2014]

Figure 9. Visualization of pose results on images from FLIC. Meaning of stick figures is the same as in Fig. 8 above.

Neural Networks in Practice

- CNNs for style transfer [Gatys et al., 2015, "DeepArt"]

- CNNs used in GANs [Karras et al., 2018]


Encyclopedia, W. T. F. (2016).
Overfitting.


Mitchell, T. M. (1997).
Machine Learning.
McGraw-Hill, Inc., New York, NY, USA, 1 edition.


T.N. Wiesel, D. H. (1959).
Receptive fields, binocular interaction and functional architecture in the cat's visual cortex.
The Journal of Physiology, 148:574–591.


Wikipedia: The Free Encyclopedia (2018).
Convolutional Neural Networks.