Artificial Intelligence

Albert-Ludwigs-Universität Freiburg

Thorsten Schmidt
Abteilung für Mathematische Stochastik

www.stochastik.uni-freiburg.de
thorsten.schmidt@stochastik.uni-freiburg.de
SS 2017

Our goal today

Literature (incomplete, but growing):

- I. Goodfellow, Y. Bengio und A. Courville (2016). **Deep Learning**. http://www.deeplearningbook.org. MIT Press
- D. Barber (2012). **Bayesian Reasoning and Machine Learning**. Cambridge University Press
- R. S. Sutton und A. G. Barto (1998). **Reinforcement Learning : An Introduction**. MIT Press
- G. James u. a. (2014). **An Introduction to Statistical Learning: With Applications in R**. Springer Publishing Company, Incorporated. ISBN: 1461471370, 9781461471370
- T. Hastie, R. Tibshirani und J. Friedman (2009). **The Elements of Statistical Learning**. Springer Series in Statistics. Springer New York Inc. URL: https://statweb.stanford.edu/~tibs/ElemStatLearn/
- K. P. Murphy (2012). **Machine Learning: A Probabilistic Perspective**. MIT Press
- CRAN Task View: Machine Learning, available at https://cran.r-project.org/web/views/MachineLearning.html
- UCI ML Repository: http://archive.ics.uci.edu/ml/ (371 datasets)

Structured Probabilistic Models

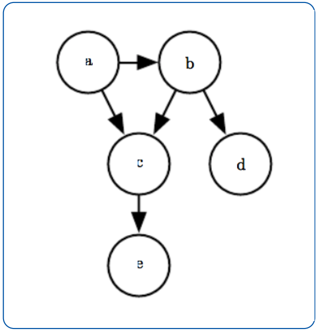
Machine Learning often involves high-dimensional probability distributions and numerous interaction between the dimensions need to be specified. One possibility to achieve this are structured probabilistic models and we discuss two possibilities.

1. Directed graphical models describe the **conditional factorization** via directed graphs. The density is factorized as follows:

$$p(\boldsymbol{x}) = \prod_i p(x_i | x_j : j \in J_i)$$

with subsets $J_1, J_2, \ldots$ of the index set of $\boldsymbol{x}$. The example (left[1]) describes the density

$$p(a,b,c,d,e) = p(a)p(b|a)p(c|a,b)p(d|b)p(e|c).$$



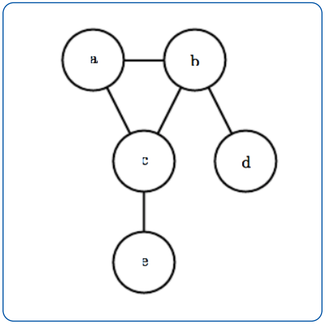
[1] Taken from Goodfellow e.a.(2016), Figure 3.7.

2 Undirected graphical models describe the **unconditional factorization** via undirected graphs. The density is factorized as follows:

$$p(\boldsymbol{x}) = \prod_i p(x_j : j \in J_i)$$

with subsets $J_1, J_2, \dots$ of the index set of $\boldsymbol{x}$.
The example (left[2]) describes the density

$$p(a,b,c,d,e) \propto \phi^1(a,b,c)\phi^2(b,d)\phi^3(c,e).$$



[2]Taken from Goodfellow e.a.(2016), Figure 3.8.

Stochastic Gradient Descent

- Most of the algorithms we saw had to solve an optimization problem. A standard algorithm to solve these problems is the **gradient descent algorithm** ($\rightarrow$ blackboard - proposed by A. Cauchy in 1847).
- However, if the training set is too large, this becomes computationally very expensive: consider a cost functional of the type

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

  with some differentiable loss function L.
- Gradient descent requires the computation of

$$\partial_\theta = \frac{1}{m} \sum_{i=1}^{m} \partial_\theta L x^i, y^i, \theta),$$

  leading to a computational cost of $O(m)$.
- As can be seen above, the gradient is actually an average - as for estimating an expectation it might be feasible to consider a small subsample of $\{1, \dots, m\}$ leading to approximately the same result: this is the **stochastic gradient descent** approach.

- Select uniformly a subsample $\{i_1,\dots,i_n\} \subset \{1,\dots,m\}$ and compute the vector

$$g(\theta) := \frac{1}{n}\sum_{k=1}^{n} \partial_\theta L x^{i_k}, y^{i_k}, \theta).$$

- The algorithm computes the n-th approximation of the minimum θ_* by

$$\theta^n = \theta^{n-1} - \varepsilon g(\theta^{n-1})$$

  with stepsize (or learning rate) ε.
- It is quite surprising that a variant of the gradient descent arises here. Typically, gradient descent is a slow and quite unreliable procedure (see Wikipedia for some examples and R code). However, the stochastic gradient descent is very successfully applied in machine learning - it often finds a solution close to a minimum quick enough (see Goodfellow e.a. (2016) Section 5.9 for further comments).

Deep Learning

- "Deep"learning contrasts ßhallow"learning: such algorithms are for example linear regression, SVMs...: they have an input layer and an output layer. We have experienced the kernel trick: inputs may be transformed once before application of the algorithm.
- In deep learning there are one ore more **hidden layers** between input and output. Intuitively, at each layer we take the input, make a transformation and generate the output for the next layer.
- More formally, this corresponds to iteratively applied functions: a deep network is of the form

$$f(x) = f^n \circ \cdots \circ f^1(x) = f^n(f^{n-1}(\cdots f^2(f^1(x))\cdots)).$$

- f^k is called the k-the layer of the network.

A bit of the history

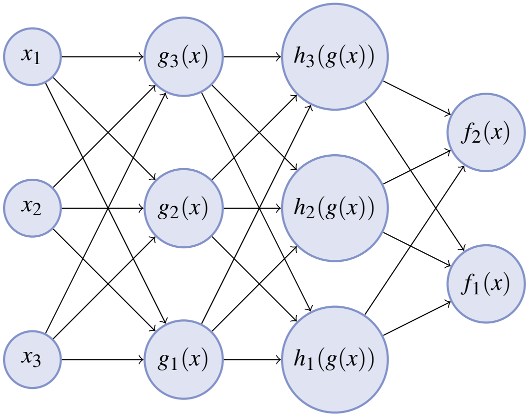
- The terminology of deep learning stems from early research on artifical intelligence and we dive shortly into this exciting subject. Two aspects were important at those times: to be inspired by the human brain and, on the other side, to try to understand the brain better through the construction of similar algorithms. Nowadays, we are more pragmatic and generalize the earlier ideas in several respects.
- A **neuron**[3] takes several inputs, say $x_1, \dots, x_n$ and gives

$$\mathbb{1}_{\{\sum_{i=1}^n w_i x_i > \theta\}}$$

  as an output - $w_i \in \mathbb{R}$ are several weights and $\theta \in \mathbb{R}$ is a threshold. This description of a neuron was given 1943 by W. McCulloch and W. Pitts.
- It was the idea of F. Rosenblatt in 1958 to introduce a simple neural net, called **perceptron** (after perception) which takes (possibly several) neurons as inputs and generates a more complex decision mechanism.

[3] See the wikipedia article on perceptrons.

Feed-forward neural networks



- Networks of this type are called either **feed-forward neural networks** or **multi-layer perceptrons** (when the nodes are actually neurons).
- $x_1, \dots, x_n$ constitute the input layer. They give the input to all connected neurons in the **first layer**. There are 3 **hidden units** in our case and **two hidden layers**. The output is, as previously $f(x) = h(g(x))$.

- One problem which can not be achieved by a single layer perceptron is learning XOR ($\rightarrow$ Exercise).

The universal approximation theorem

- One important property of feed-forward neural networks is, that even in the single-layer case they can approximate arbitrary functions very well.
- The result is the so-called universal approximation theorem proved in Kurt Hornik (1991). „Approximation capabilities of multilayer feedforward networks“. In: **Neural networks** 4.2, S. 251–257.
- We study the mathematical details of this results.

- We consider special classes of feed-forward neural networks, which can be thought of a small generalization of multi-layer perceptrons: in each step, a neuron transforms the input vector x in an affine form to $a^\top x + b$ and sends the output $\phi(a^\top x + b)$. The outputs are weighted and summed up by each connected neuron.
- If there is only one hidden layer and only one output unit, we arrive at the output
$$\sum_{i=1}^{n} c_i \phi(a_i^\top x + b_i).$$
- Hence, the functions implemented by such a network with n hidden units is
$$\mathcal{N}^{(n)} = \mathcal{N}^{(n)}(\phi) = \Big\{ h : \mathbb{R}^d \to \mathbb{R} : h(x) = \sum_{i=1}^{n} c_i \phi(a_i^\top x + b_i) \Big\}$$
and for an arbitrary large number of units we set $\mathcal{N}(\phi) = \cup_n \mathcal{N}^{(n)}$.
- We consider functions in the $L^p(\mu)$-space with a finite measure μ. This are measurable functions $f : \mathbb{R}^d \to \mathbb{R}$, such that
$$\|f\|_p := \Big(\int |f(x)|^p \mu(dx) \Big)^{1/p} < \infty.$$

- A subset S of L^p is called **dense**, if for every $f \in L^p$ and $\varepsilon > 0$ there is a function $g \in S$, such that $\|f - g\|_p < \varepsilon$.

Theorem (Hornik (1991))

If ϕ is bounded and not constant, then $\mathcal{N}(\phi)$ is dense in $L^p(\mu)$ for any finite measure μ on $\mathbb{R}^d$.

This result also holds on the Banach space $C(K)$, K compact, with respect to the sup-norm. Further results are found in Hornik (1991).

The proof

- We will not discuss all the details of the proof, but have a look at certain components.
- First, observe that $\mathscr{N}$ is a **linear** subspace of $L^p(\mu)$ (elements are bounded!)
- If $\mathscr{N}$ is **not** dense, then the Hahn-Banach theorem yields the existence of a (non-zero) continuos linear function Λ such that Λ vanishes on $\mathscr{N}$. The goal is to construct a contradiction by this.
- Currently, Λ seems not to be so tractable, but duality of Hilbert spaces actually gives a very good description of such functionals. In particular, in our case we know that
$$\Lambda f = \int f g \mu(dx)$$
with some $g \in L^q(\mu)$ and $q = p/(p-1)$.
- Now we can write
$$\Lambda f = \int f d\mu'$$
with (by Hölders inequality) some finite (but possibly signed) measure μ'.
- As Λ vanishes on $\mathscr{N}$,
$$\int \phi(a^\top x + b)\mu'(dx) = 0$$
for all $a \in \mathbb{R}^d$ and $b \in \mathbb{R}$.

We have

$$\int \phi(a^\top x + b)\mu'(dx) = 0 \tag{1}$$

for all $a \in \mathbb{R}^d$ and $b \in \mathbb{R}$. It is clear that this can not hold for any function ϕ. Hornik was able to show, that if ϕ is bounded and not constant, then (1) will not hold for any finite signed measure μ'.

- A first step is the transformation

  $$\int \phi(a^\top x + b)\mu'(dx) = \int \phi(t+b)\mu_a(dt)$$

  with the projection measure $\mu_a(B) = \mu'(x \in \mathbb{R}^d : a \top x \in B)$.
- One proceeds further and arrives at

  $$\int \phi(t)h(\alpha t + \beta)dt.$$

  Now one can apply Fourier transform and arrives at $\mu_a = 0$ for all $a \in \mathbb{R}^d$.

Rate of convergence

- It was moreover shown in Andrew R Barron (1994). „Approximation and estimation bounds for artificial neural networks“. In: **Machine Learning** 14.1, S. 115–133 that the mean integrated squared error between (single-layer) network and target function is bounded by

$$O(1/n) + O(nd/N)\log N.$$

- Here, n is the number of nodes, d is the input dimension and N is the number of training observations.
- However, the result only considers a single layer and the analysis is therefore limited.

Gradient-based learning

- A remarkable difference between the neural networks and previously seen algorithms is that neural networks have typically **non-convex** target functions.
- Therefore, gradient methods (possibly stochastic ones) come into play in contrast to linear-quadratic sovlers we saw before. Let us study some common cost functions.

Learning with maximum-likelihood

- Once we have a probabilistic model, we automatically have a log-likelihood function which can solve as loss function. The advantage of this approach is that we do not need to specify an additional criterion.

- What kind of distance to we minimize when we look at maximum-likelihood ?
- Although more complicated schemes may be possible, think of i.i.d. observations from the density (or probability function) $p(x,\theta)$. Then, the log-likelihood is

$$l(x,\theta) = \sum_{i=1}^{n} \log p(x^i,\theta).$$

- Of course, this can be viewed as

$$l(x,\theta) = m\frac{1}{m}\sum_{i=1}^{n} \log p(x^i,\theta) = mE_x^n[\log p(X,\theta)]$$

  where E_x^n is the empirical distribution at the observation $x = (x^1,\dots,x^n)$.
- Then, we actually minimize the Kullback-Leibler divergence[4] given by

$$D_{KL}(p_x^n, p_\theta) := E_x^n[\log p_x^n(X) - \log p_\theta(X)]$$

  where p^n is the epmirical distribution at the observation x and p is our model density (note that the first term does not depend on θ).

[4] $D_{KL}(P,Q) = E_P[\log \frac{dP}{dQ}(X)]$.

Conditional maximum-likelihood

- For **supervised learning** we additionally have supervision data, such that we actually observe (x^i, y^i), $i = 1, \dots, n$. Conditional maximum likelihood then maximizes (in the i.i.d.-situation)

$$\sum_{i=1}^{n} \log p(y^i | x^i, \theta).$$

  Here, $p(y|x, \theta)$ is the density (or probability function) of y given x and parameter θ.
- Hence, a possible **cost function** for supervised learning is given by

$$J(\theta) = -\sum_{i=1}^{n} \log p(y^i | x^i, \theta)$$

Rectified linear units

- Inspired from perceptrons, our hidden units will typically produce an output of the form
$$\phi(a^\top x + b).$$
This generalizes the neuron where $\phi(y) = \mathbb{1}_{\{y>\theta\}}$ and it is an important question which ϕ is most suited ? Typically one would think of sigmoids, probability transforms or logit link functions. It is quite surprising, that the most applied **activation function** is the **rectified linear unit**
$$\phi(y) = \max\{0, y\}.$$
- However, there are good reasons for this: in this case, ϕ is piecewise linear and the second derivative vanishes (a.s.). Hence the gradient direction is far more useful than in other cases !

Some generalizations are available:

- $$\max\{0,y_i\}+\alpha_i\min\{0,y_i\}, \qquad i=1,\dots,d.$$
- A **leaky ReLU** fixes a small α while a parametric ReLU learns α during the procedure.
- A further treatment are maxout units (skipped here - see Goodfellow, Chapter 6.3.1.)