Introduction into Bayesian (Deep) Learning

Christian Herta

Hochschule für Technik und Wirtschaft (HTW) Berlin
christian.herta@htw-berlin.de

Abstract

In the last years, machine learning moved from a niche to mainstream in computer science due to the increase in stored data. Machine learning (ML) promises to gain value from data, e.g., to make predictions or to detect patterns in data. More and more, the academic curricula of computer science reflect the increasing importance of machine learning. Universities offer many different courses in data science and machine learning where students learn different approaches to machine learning. In this paper, we give a concise introduction to Bayesian machine learning. The Bayesian approach has some principle advantages such as naturally handling uncertainty and preventing overfitting. For studying, we have developed many exercises which are referenced in this introduction. For didactic reasons, we use simple examples for illustrating the concepts, e.g., the binominal distribution and (Bayesian) linear regression. We also describe how Bayesian methods are related to graphical models, probabilistic programming and deep learning. The exercises combine theoretical background with programming. Thus, the course has sufficient mathematical depth and relevance to practical application.

I. Introduction

We are entering a data era. Nowadays, large data sets can be stored on commodity hardware which results in a massive decrease in storage costs. Methods for gaining value for such data become more and more critical. Machine learning promises to generate such values by learning prediction models from data. Software-libraries provide standard machine learning algorithms such a logistic regression, support vector machines and simple neural networks out-of-the-box. So, they can easily be trained and used in production for simple problems.

More complicated problems require specialized models which must be designed for the problem at hand. Model-based machine learning offers a framework for modeling, learning, and inference in such models. Such models correspond to graphical representations which are called probabilistic graphical models [9]. For modeling, we focus on directed graphical models. In such models Bayesian methods are typically used for inference. However, Bayesian methods are a fundamental approach for data analysis and statistics. Bayesian methods have the advantage to mitigate overfitting and naturally handle uncertainty. They give a probability distribution over the predicted values. From these it can be easy concluded how confident the prediction is. Such confidence estimation is essential for critical applications, e.g., in medicine or autonomous driving.

Recently, researchers invented black-box inference for Bayesian methods. Without a black-box solver, it is necessary to implement an inference algorithm for the particular problem at hand. So in the past, only experts knew how to implement such solutions. With black-box inference, machine learning developers now have the potential to develop solutions based on Bayesian methods.

Based on black-box inference, probabilistic programming languages were invented to support the development process. In probabilistic programming, the developer (or researcher) can mainly focus on the modeling part by using a developer-friendly API. Still, it is essential to understand how inference works, e.g., for tuning scalable solutions or understanding the limitations.

A key trend in machine learning is deep learning. In contrast to traditional machine learning, features are learned from data and are not engineered by experts. The most prominent models are (deep) neural networks. Neural networks are typically trained with the frequentist approach, (see section iii). However, Bayesian methods can also be used either to train neural networks or to design neural network autoencoders(VI).

This paper gives the reader a guide to gain knowledge about Bayesian methods. The content is appropriate for an advanced data science course. The didactic concept of the provides material is to combine exercises of the mathematical basics of Bayesian statistics with programming exercises. The aim is to build a bridge between theory and practice. Notably, here we give pointers to the exercises which we have developed for studying. We believe that seriously studying for a deep understanding cannot be done without practice.

Practicing in this context is solving mathematical exercises and implementing corresponding code, e.g., algorithms. Therefore most exercises are of such a mixed type. Bayesian methods are an advanced topic of machine learning, and their study requires a strong background in machine learning. We assume that the reader has such a background.

In this paper, the overview is given in a consistent mathematical notation. This allows to better identify the same concepts that occur in different contexts or scientific publications in different notations.

In section II we describe the prerequisite compactly and give links to appropriate exercises. In the next section III, the principle of Bayesian data analysis is derived from Bayes rule, and the principal difficulty of Bayesian analysis is explained. In section IV some methods are presented which overcome the difficulties. We focus on such methods which are used for probabilistic programming. In section V we describe how Bayesian inference is eased by probabilistic programming. One goal of the paper is to prepare the reader for recent research literature in the field of Bayesian (deep) learning. In section VI we present examples of such neural networks. The complete content including the material of the exercises is equivalent to a one-semester master course for students with appropriate background in machine learning.

The exercises are based on jupyter notebooks [8]. In the exercises, we give a summary of the knowledge which is necessary to solve the exercises. Also, there are pointers to the literature for further self-study.

A complete list of all in this paper referenced exercises is given on the following web page:
`https://www.deep-teaching.org/courses/bayesianLearningListOfExecises`.

II. Prerequisite

Before we start with the fundamentals of Bayesian data analysis, we need some basic concepts and definitions of key technical terms. Here we assume that the reader has a fundamental background in machine learning, probability theory, and statistics.

In the following, we recapitulate some of the basics which are especially crucial for Bayesian learning.

i. Independent and identically distributed data

Unless otherwise stated, we assume that the data $\mathcal{D} = \{x_1, x_2, \ldots, x_n\}$ (resp. for supervised learning $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$) is *independent* and *identically distributed* (*i.i.d.*):

- *Independent* means that the outcome of one observation x_i does not effect the outcome of another observation x_j for $i \neq j$.
- The data is *identically distributed* if all x_i's are drawn from the same probability distribution.

ii. Statistics and estimators

A *statistic* is a random variable S that is a function of $\mathcal{D}$, i.e.

$$S = f(\mathcal{D}).$$

An *estimator* $\hat{\theta}$ is a statistic used to approximate the (true) parameters θ that determine the distribution of data $\mathcal{D}$. The *variance of an estimator* is defined by

$$\operatorname{var}(\hat{\theta}) := \mathbb{E}_{\mathcal{D}}[\hat{\theta}^2] - (\mathbb{E}_{\mathcal{D}}[\hat{\theta}])^2. \tag{1}$$

The subscript $\mathcal{D}$ on the expectation $\mathbb{E}_{\mathcal{D}}$ indicates that the expectation is with respect to all possible data sets $\mathcal{D}$. The bias of an estimator is defined as

$$\operatorname{bias}(\hat{\theta}) = \mathbb{E}_{\mathcal{D}}\left[\hat{\theta}\right] - \theta.$$

A statistic S is *sufficient* if $p(\hat{\theta} \mid \mathcal{D}) = p\left(\hat{\theta} \mid S(\mathcal{D})\right)$. All necessary information for the estimation of the parameters is compressed in the sufficient statistic.

For exercises, see:

- exercise-expected-value
- exercise-biased-monte-carlo-estimator
- exercise-variance-dependence-on-sample-size

iii. Kullback-Leibler Divergence

The Kullback-Leiber (KL) divergence is a measure how (dis)similar two probability distributions are. For probability mass functions $q(X)$ and $p(X)$ the KL-divergence is defined as[1]:

$$\mathcal{D}_{KL}(q(X) \mid\mid p(X)) = \sum_{x \in \operatorname{val}(X)} q(x) \log \frac{p(x)}{q(x)} \tag{2}$$

$\operatorname{val}(X)$ are the possible values of the random variable X. We encourage the reader to study the exercise "exercise-kullback-leibler-divergence".

iv. Directed Graphical Models

For modeling, we use the notation of directed graphical models (Bayesian networks). In a probabilistic graphical model (conditional) independence properties are encoded in the graph [9]. In other words a graphical model is a map of independencies (i-map). In directed graphical models, there is a set of factors corresponding to conditional probabilities for each variable. The

[1] For probability density functions the sum must be replaced by an integral.

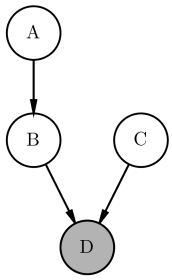

Figure 1: *Example of a directed graphical model. A directed graphical model corresponds to a factorization of the joint probability of the random variables (nodes in the graph). Here the joint probability is factorized according to $P(A, B, C, D) = P(A)P(C)P(B \mid A)P(D \mid C, B)$. If a probability factor of a random variable is conditioned on other variables a corresponding edge in the graph exists.*

connection between the variables is represented by a directed acyclic graphs (DAG)[2]. In figure 1 we give a simple example for a directed graphical model. For exercises to Bayesian networks, see:

- exercise-bayesian-networks-by-example
- exercise-forward-reasoning-probability-tables
- exercise-d-separation

III. Fundamentals of Bayesian analysis

i. Bayes Rule

In the training phase the parameters of a model are learned by training data $\mathcal{D}$. The criteria for the learning process can be derived from Bayes rule for data $\mathcal{D}$ and parameters θ:

$$p(\theta \mid \mathcal{D}) = \frac{p(\mathcal{D} \mid \theta)p(\theta)}{p(\mathcal{D})} \tag{3}$$

In Bayesian data analysis the continuous parameters θ of a model are described by probability density functions. Because of the importance of Bayes rule the individual factors in equation 3 have names:

- $p(\theta \mid \mathcal{D})$ is called the *posterior* (distribution).
- $p(\mathcal{D} \mid \theta)$ is the *likelihood*. Typically, the likelihood is considered as a function of θ, i.e. $\ell(\theta) = p(\mathcal{D} \mid \theta)$. Note that $\ell(\theta)$ is not a probability distribution with respect to θ.
- $p(\theta)$ is the *prior*. This distribution describes the prior knowledge (or belief) about θ before taking the data $\mathcal{D}$ into consideration.
- The denominator on the right side $p(\mathcal{D})$ is the marginal likelihood or evidence. In practice, the denominator $p(\mathcal{D}) = \int_\Theta p(\mathcal{D} \mid \theta)p(\theta)d\theta$ is often computationally intractable. Different methods were developed to handle this issue, see section IV.

We provide an exercise for studying Bayes rule: "exercise-bayes-rule".

[2]For a short summary (in German) of Bayesian networks, see `http://christianherta.de/lehre/dataScience/bayesian_networks/Bayessche_Netze_Repraesentation.html`

ii. Prediction

To predict new data $\bar{\mathcal{D}}$[3] based on the training data $\mathcal{D}$ we have to average the preditions $p(\bar{\mathcal{D}} \mid \theta)$ weighted by the posterior $p(\theta \mid \mathcal{D})$

$$p(\bar{\mathcal{D}} \mid \mathcal{D}) = \int_\Theta p(\bar{\mathcal{D}}, \theta \mid \mathcal{D}) d\theta \tag{4}$$

$$= \int_\Theta p(\bar{\mathcal{D}} \mid \theta, \mathcal{D}) p(\theta \mid \mathcal{D}) d\theta \tag{5}$$

$$= \int_\Theta p(\bar{\mathcal{D}} \mid \theta) p(\theta \mid \mathcal{D}) d\theta \tag{6}$$

$\Theta = \{\theta \mid P(\theta \mid \mathcal{D}) > 0\}$ is the support of $P(\theta \mid \mathcal{D})$. From line 5 to 6 the conditional independence property $\bar{\mathcal{D}} \perp \mathcal{D} \mid \theta$ was used.

iii. Maximum Likelihood Estimation (ML)

In comparison to Bayesian data analysis in a frequentist approach the parameters θ are estimated by fixed values (point estimates), i.e. they are not described by probability distributions. Typically θ is estimated by maximizing the likelihood $\ell(\theta)$, i.e. by $\hat{\theta}_{ML} = \arg\max_\theta \ell(\theta)$. This is equivalent to minimizing the negative log-likelihood $\log \ell(\theta) = L(\theta)$

$$\hat{\theta}_{ML} = \arg\max_\theta \ell(\theta) = \arg\min_\theta \left(-\log\left(\ell(\theta)\right)\right) = \arg\min_\theta \left(-L(\theta)\right) \tag{7}$$

As a simple example we consider the binomial distribution. From n identical Bernoulli trials with possible outcomes $x_i = \{0, 1\}$ we get some data, e.g. $\mathcal{D} = \{0, 1, 1, \dots\}$. From the data we want to estimate the parameter θ. Here, θ is the probability of a positive outcome $X = 1$ of a Bernoulli trial.
From the data we can compute a sufficient statistic which holds the same information about θ as $\mathcal{D}$. The suffient statistics for the Bernoulli distribution is the number of sucesses (1's) k and the number of trials n. The probability for k-sucesses is given by

$$P(k \mid \theta) = \binom{n}{k} \theta^k (1-\theta)^{n-k} \tag{8}$$

If we consider this as a function of θ we get the likelihood. For the maximum-likelihood estimation for the binomial distribution we have:

$$\arg\min_\theta \left(-\log L(\theta)\right) = \arg\min_\theta -\log\left(\binom{n}{k} \theta^k (1-\theta)^{n-k}\right)$$

A necessary condition for the minimum is that the first derivative is zero:

$$0 = \frac{d}{d\theta}\left(\theta^k (1-\theta)^{n-k}\right) = k\theta^{k-1}(1-\theta)^{n-k} - (n-k)\theta^k (1-\theta)^{n-k-1}$$

So the maximum likelihood estimator is $\hat{\theta}_{ML} = \frac{k}{n}$. For an exercise to maximum-likelihood on the Gaussian distribution see exercise-univariate-gaussian-likelihood.

Another *point estimate* is given by the maximization of the posterior of equation 3 (*maximum a posterior - MAP*). Note that the denominator is constant with respect to (w.r.t.) θ. Therefore

[3] An analog expression for supervised learning $p(\bar{y} \mid \mathcal{D}, \bar{x}) = \int p(\bar{y} \mid \bar{x}; \theta) p(\theta \mid \mathcal{D}) d\theta$ can be derived from the conditional independencies.

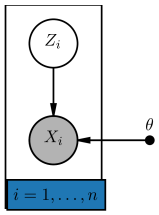

Figure 2: *Example of a simple hidden variable model: For each observed variable $X_i = x_i$ there is a hidden variable Z_i.*

for maximization, it is sufficent to consider only the prior and the likelihood. In log-space the prior and the likelihood are added. E.g., for learning a linear regression (or neural networks regression) the squared error loss correspond to the log-likelihood and the L2-regularization term to a Gaussian log-prior. We provide exercise-linear-regression-MAP for studying the derivation in detail.

iv. Hidden variable models, expectation maximization and variational lower bound

Bayesian networks can contain unobserved variables. Sometimes we are interested in the distribution of such variables when we observe other variables in the graph. As an example consider a mixture of two Gaussians[4] with indices $z = 1$ respectivly $z = 2$. Each of the Gaussians has two parameter (μ_z, σ_z^2) with $z \in \{1, 2\}$. Given some i.i.d. training data $\mathcal{D} = \{x_1, \ldots, x_n\}$ we want to estimate the parameters $\theta = \{\mu_1, \sigma_1^2, \mu_2, \sigma_2^2\}$ by maximum likelihood. Additionally, we want to know the distribution $p(z_i \mid x_i)$ for each (training) example.
To summarize the problem we have a set of hidden variables z_i. Here, we have an observation x_i for each hidden variable. In figure 2 there is a graphical representation of this hidden variable model. For the training data we want a point estimate of θ and a probability distribtion $p(z_i \mid x_i; \theta)$. So, we are Bayesian only w.r.t. z_i.
The (marginal)[5] log-likelihood of the observed data is:

$$L(\theta) = \log p(\mathcal{D} \mid \theta) = \sum_i^n \log p(x_i \mid \theta) = \sum_{i=1}^n \left(\log \sum_{z_i} p(x_i, z_i \mid \theta) \right) \tag{9}$$

The difficulty is that we don't know from which Gaussian z each data point was generated. This is reflected in formula 9. There is a sum inside the log which complicates the maximization procedure substantially.
However, with the Jensen inequality and an auxilary distribution $q(z_i)$ we can push the log inside

[4]For an exercise to the multivariate Gaussian distribution see exercise-multivariate-gaussian
[5]The marginalization is with respect to hidden variables z_i

the sum to get a lower bound [11]:

$$L(\theta) = \sum_i \log p(x_i \mid \theta) \tag{10}$$

$$= \sum_{i=1} \left(\log \sum_{z_i} p(x_i, z_i \mid \theta) \right) \tag{11}$$

$$\geq \sum_i \sum_{z_i} q(z_i) \log \frac{p(x_i, z_i \mid \theta)}{q(z_i)} \tag{12}$$

$$= \sum_i \mathbb{E}_{q(z_i)} \left[\log \frac{p(x_i, z_i \mid \theta)}{q(z_i)} \right] \tag{13}$$

$$= -\sum_i \mathcal{D}_{KL} \left(q(z_i) \mid\mid p(x_i, z_i \mid \theta) \right) \tag{14}$$

$$= \mathcal{L}(q, \theta) \tag{15}$$

$\mathcal{L}(q, \theta)$ is the variational lower bound or evidence lower bound (ELBO). Note that $\mathcal{L}(q, \theta)$ has two arguments q and θ. Iteratively alternating until convergence between the maximization of one of the arguments holding the other argument fixed results in the EM-algorithm, with iteration index t:

- In the E-Step we search in the distribution family q at the current position $\theta^{(t)}$ the strongest bound: $q^{(t+1)} \leftarrow \arg\max_q \mathcal{L}(q, \theta^{(t)})$
- In the M-Step we maximize for a fixed $q^{(t+1)}$) w.r.t. θ: $\theta^{(t+1)} \leftarrow \arg\max_\theta \mathcal{L}(q^{(t+1)}, \theta)$

The gap between the lower bound $\mathcal{L}(q, \theta)$ and the log-likelihood $L(\theta)$ is[6]:

$$L(\theta) - \mathcal{L}(q, \theta) = \sum_i \mathcal{D}_{KL} \left(q(\mathbf{z}_i) \mid\mid p(\mathbf{z}_i \mid \mathbf{x_i}, \theta) \right) \tag{16}$$

We provide two exercises to study the EM algorithm in detail:

- Minimalistic example of a hidden variable model which should be solved by the EM-algorithm: exercise-simple-example-for-EM
- One dimensional Gaussian mixture model trained with the EM-Algorithm: exercise-1d-gmm-em

Remark: If we want to estimate a probability distribution of the parameters θ, i.e. $p(\theta \mid \mathcal{D})$ we can consider the parameters θ as hidden variables. So the distinction between parameters and variables is blurred in Bayesian approach.

IV. Bayesian Methods

Different Bayesian methods were developed to compute respectivly an approximation to the posterior of equation 3 without explicitly computing the denominator. Here we focus on Monte Carlo methods and variational methods for which black-box inferences were developed recently. For didactic reasons, we also explain the conjugate prior method to illustrate the principle of Bayesian learning.

[6] see e.g. `http://christianherta.de/lehre/dataScience/variational-methods/Expectation-Maximization-variational-derivation.html`

i. Conjugate Prior

For simple models, the *conjugate prior* method can be used to overcome the problem for computing the denominator of equation 3: For the prior $p(\theta)$ and the likelihood $p(\mathcal{D} \mid \theta)$ are such probability functions selected such that the posterior has the same functional form as the prior (same family of functions). To illustrate this, we consider again n Bernoulli trials. So for the likelihood we have the Binominal distribution, see equation 8. If we use the beta distribution for the prior probability density function (pdf), then the posterior is again a beta distribution.
The Beta distribution is

$$\text{Beta}(\alpha, \beta) = p(\theta \mid \alpha, \beta) = \frac{\theta^{\alpha-1}(1-\theta)^{\beta-1}}{\int_0^1 u^{\alpha-1}(1-u)^{\beta-1}\, du} \tag{17}$$

with $\theta \in [0, 1]$ and the shape parameters $\alpha > 0, \beta > 0$. Note that the denominator only normalizes the pdf. Now we neglect all normalizing constants to compute the (unnormalized) posterior. The product of the (unnormalized) prior and the (unnormalized) likelihood is

$$p(\theta \mid n, k) \propto \theta^k (1-\theta)^{n-k} \theta^{\alpha-1}(1-\theta)^{\beta-1} = \theta^{k+\alpha-1}(1-\theta)^{n-k+\beta-1} \tag{18}$$

Comparing the right side of equation 18 with the nominator of equation 17 we see that the functional form is identical. Therefore, we can conclude that the posterior is also a Beta distribution $\text{Beta}(\alpha', \beta')$ with parameters $\alpha' = \alpha + k$ and $\beta' = \beta + n - k$.

The conjugate prior method is used, e.g. for sensor fusion and in the Kalman Filter[7] (to combine the measurement with the motion dynamics/kinematics). We provide an exercise for sensor fusion and Kalman Filter: exercise-sensorfusion-and-kalman-filter-1d.

As already stated, the conjugate prior method can only be applied to simple problems with standard probability density functions. To allow arbitrary probability densities we need more flexible methods. Thus, we consider Monte Carlo methods as one suitable approach.

ii. Monte Carlo Methods

Monte Carlo (MC) methods are a general approach to get approximations for numerical results. For example, Monte Carlo methods can be used to approximate integrals (Monte Carlo integration). Here, we consider integrals which contain a factor $p(x)$ in the integrand, as for computing expectations or marginalization. If we can draw m samples x_i from a probability distribution $p(x)$ (notated as $x_i \sim p(x)$) we can get an approximation by the crude MC estimator

$$F(x) = \int p(x) f(x) dx \approx \sum_{i=1}^{m} f(x_i) = \hat{F}(x). \tag{19}$$

See also exercise-sampling-crude-monte-carlo-estimator.
But how to obtain samples from $p(x)$? If x is a scalar and we know the *cumulative distribution function* (cdf) $c(x) = \int_{-\infty}^{x} p(x')dx'$ we can sample by *inverse transform sampling*: First we draw samples from a uniform distribution $\text{uniform}(0, 1)$ by a random number generator. Then these samples are maped by the inverse cdf, see exercise exercise-sampling-inverse-transform.

For different standard multivariate probability distributions there are several methods for sampling (see e.g. [12, chapter 7]). Many software packages (e.g. `numpy.random`) implement such sampling for standard probability distributions out-of-the-box. Such sampling methods can also be used for other applications. We provide exercises for the following sampling methods:

[7] For an derivation of the Kalman Filter in the simple 1d case see e.g. `http://christianherta.de/lehre/localization/Kalman-Filter-1D.html`

- In *rejection sampling* the samples are drawn from a similar probability distribution $q(x)$. Some of the samples are rejected to get samples from $p(x)$ as a result, see exercise-sampling-rejection.
- In *importance sampling* the samples are also drawn from a similar distribution $q(x)$. These samples are weighted by the ratio $p(x)/q(x)$, e.g. to correct the Monte Carlo integration, see exercise exercises-sampling-importance.

ii.1 Gradient of a MC expectation

Often the gradient of an expectation is needed, e.g., for optimization by gradient descent, see section VI for examples. By algebraic manipulation the gradient of an expectation can be expressed by the expectation of a gradient:

$$\nabla_w \mathbb{E}_{q_w(x)}\left[f(x)\right] = \mathbb{E}_{q_w(x)}\left[f(x)\nabla_w \log q_w(x)\right]$$

The expectation on the right hand side can be approximated by Monte Carlo if we can sample from $x_i \sim q_w(x)$. That is the so-called *score function estimator* trick.

ii.2 Variance reduction

Monte Carlo methods provide an estimate, not an exact solution. The estimated value should be as close as possible to the true value. So the estimate should have a low variance and should be unbiased. There are different methods to reduce the variance of an MC-estimator. Often, only by such variance reduction, Monte-Carlo becomes practical. For some of the variance reduction techniques we provide exercises:

- *Control Variates*[8]: exercise-variance-reduction-by-control-variates
- *Reparametrization*: exercise-variance-reduction-by-reparametrization
- *Rao-Blackwellization*: exercise-variance-reduction-by-rao-blackwellization

iii. Markov Chain Monte Carlo (MCMC)

In principle, the normalized posterior of equation 3 can be approximated with Monte Carlo integration by an MC-estimate of $p(\mathcal{D}) = \int p(\theta)p(\mathcal{D} \mid \theta)d\theta$. The variance of such an estimator would be very high unless θ is very low dimensional. So this method can not be used in practice. Note that if the probability space is high dimensional θ then in most regions holds $p(\theta) \approx 0$. With importance sampling (e.g. by direct sample from a uniform distribution and weighting) most importance weights would be zero. So, the effective number of samples would be extremely low. This results in a very high variance of the estimator.
Therefore, we want to draw samples from regions where $p(\theta)$ is substantially different from zero. If we have such a sample $\theta^{(t)}$ at iteration step t then it is highly probable that nearby values of $\theta^{(t)}$ also have $p(.)$ substantially different from zero. The idea of MCMC is now to perturb the current sample $\theta^{(t)}$ by a small $\delta\theta^{(t)}$. Such, we get a candidate $\theta_c(t+1) = \theta^{(t)} + \delta\theta^{(t)}$. We accept the move only with an acceptance probability p_{ac} [] given by

$$p_{ac} = \min\left(1, \frac{p\left(\theta^{(t)}\right) p\left(\theta_c^{(t+1)} \mid \theta^{(t)}\right)}{p\left(\theta_c^{(t+1)}\right) p\left(\theta^{(t)} \mid \theta_c^{(t+1)}\right)}\right), \tag{20}$$

[8] In the context of reinforcement learning this is known as *baseline*.

i.e. with p_{ac} we accept the move and with $1 - p_{ac}$ we stay at the value $\theta^{(t)}$:

$$\theta^{(t+1)} = \begin{cases} \theta_c^{(t+1)} & \text{with probability } p_{ac} \\ \theta^{(t)} & \text{with probability } 1 - p_{ac} \end{cases}$$

The new state θ at $t+1$ depends only on the previous state at t and is independent of the states previous to $t-1$. Therefore the *Markov property* $\theta^{(t+1)} \perp \theta^{(t-1)} \mid x^{(t)}$ is satisfied and such a sampling is called *Markov Chain Monte Carlo*. State-of-the-art of MCMC is the Nuts sampler [6] which is an improvement of Hamiltonian MCMC [10].

For MCMC we provide the exercise exercise-sampling-direct-and-simple-MCMC.

iv. Variational Methods

A drawback of MCMC is that it is not scalable to large data sets or models with many parameters. For such models, MCMC is typically very slow and we need many samples. Therefore, it can not be used in practice. Alternatives are variational methods to overcome these issues.

The idea of variational methods is to approximate $p(\theta \mid \mathcal{D})$ with a so-called variational distribution $q_w(\theta)$. $q_w(\theta)$ is a distribution in a parametrized family, i.e., we choose a standard probability distribution with parameters ws. The parameters w are called variational parameters. Then, we must find the w's which approximate the posterior best by an optimization procedure like stochastic gradient descent.
As optimization-objective (loss) we use the KL-divergence between $q_w(\theta)$ and $p(\theta \mid \mathcal{D})$, i.e. the optimal parameters are $w^* = \arg\min_w \mathcal{D}_{KL}\left(q_w(\theta) \mid\mid p(\theta \mid \mathcal{D})\right)$.

$$\begin{aligned} \mathcal{D}_{KL}\left(q_w(\theta) \mid\mid p(\theta \mid \mathcal{D})\right) &= \int_\Theta q_w(\theta) \log \frac{q_w(\theta)}{p(\mathcal{D} \mid \theta) p(\theta)} d\theta + \int_\Theta q_w(\theta) \log p(\mathcal{D}) d\theta && (21) \\ &= \int_\Theta q_w(\theta) \log \frac{q_w(\theta)}{p(\theta)} d\theta - \int_\Theta q_w(\theta) \log p(\mathcal{D} \mid \theta) d\theta + \log p(\mathcal{D}) && (22) \\ &= \mathcal{D}_{KL}\left(q_w(\theta) \mid\mid p(\theta)\right) - \mathbb{E}_{q_w(\theta)}\left[\log p(\mathcal{D} \mid \theta)\right] + const. && (23) \end{aligned}$$

Note that $\log p(\mathcal{D})$ does not depend on w which prevents us from the difficulty to compute the "normalizer" of the posterior (denominator of equation 3). $q_{w^*}(\theta)$ is an approximation of $p(\theta \mid \mathcal{D})$. Remember that we are using a known parametric distribution for $q_w(\theta)$, so we have a normalized result.

iv.1 Mean-field approximation

For a directed graphical model with many nodes, the probability distribution for each node j is described by a known parametric distribution $q_w(\theta_j)$ and the total probability distributions factorizes[9]:

$$Q = \left\{ q_w \mid q_w(\theta) = \prod_j q_{w^{(j)}}(\theta_j) \right\}$$

This factorization approach is called *mean-field approximation*. The optimization can be done by coordinate descent, i.e. by a loop over the optimization w.r.t. to each $q_{w^{(j)}}(\theta_j)$ until convergence.

We provide the following exercises for studying:

[9] In general the probability can factorize in blocks, e.g. a block for each node, or total factorization for each component θ_i. This can be expressed by merging or spitting of nodes in the graphical model.

```
def model(X):
    mu = pyro.sample('mu', dist.Uniform(-10., 10.))
    return pyro.sample('gaussian_data', dist.Normal(mu, 1.), obs=X)

def guide(X):
    mu_loc = pyro.param('guide_mu_mean', torch.randn((1)))
    mu_scale = pyro.param('guide_mu_scale', torch.ones(1),
                          constraint=constraints.positive)
    pyro.sample('mu', dist.Normal(mu_loc, mu_scale))
```

Figure 3: *Example of a simple Gaussian model and the corresponding guide function in the probabilistic programming library pyro [2]. In the definition of the model the prior $p(\mu) = uniform(-10,10)$ is used. The guide function corresponds to a variational distribution for the parameter $p_w(\mu) = \mathcal{N}(w_0, w_1^2)$ with w_0 = 'guide_mu_mean' and w_1 = 'guide_mu_scale'.*

- exercise-variational-mean-field-approximation-for-a-simple-gaussian - univariate Gaussian via variational mean field.
- exercise-variational-EM-bayesian-linear-regression Bayesian univariate linear regression model via mean field approximation and variational expectation maximization (for the data noise a point estimate should be used).

V. Probabilistic programming

Probabilistic programming allows defining models similar to (directed) graphical models programmatically. The modeling in probabilistic programming is more powerful because the model can contain control structures such as loops and if-statements. It can be shown, that every directed graphical model can be expressed by a loop-free probabilistic program [4].
Inference (including learning) is based on black-box methods based on MC sampling or variational inference [14, 13][10]. Therefore, the programmer can focus primarily on modeling. Depending on the software packages the inference is entirely automatic, or it requires strong background knowledge for tuning the inference. Such tuning is especially necessary for a large data set and big models.

The source code for a simple Gaussian model with pyro [2] and the corresponding variational distribution model is show in figure 3.

We provide the following exercises for probabilistic programming:

- exercise-pyro-simple-gaussian
- exercise-pymc3-examples
- exercise-pymc3-bundesliga-predictor
- exercise-pymc3-ranking

VI. Bayesian Deep Learning

In this section, we give two non-trivial examples from deep learning for which we provide exercises. In both examples, the neural networks are trained by variational methods lower. Nevertheless, Monte-Carlo estimation is used to approximate the gradient of the expectations of the lower bound.

[10] Variational methods as well as the Nuts/Hamiltonian-MC sampler, need gradient information, therefore, probabilistic programming is often implemented on top of deep learning libraries which support (reverse) automatic differentiation [1].

i. Variational Autoencoder

Variational autoencoders [7] are generative models which can learn complicated probability distributions. Variational autoencoders are hidden variable models. So the terminology and the graphical model[11] are analog to section iv. The variational autoencoder is trained by the *auto-encoding variational bayes* (AEVB) algorithm. In contrast to the EM-algorithm AEVB can be used if the integral of the marginal likelihood $\int p(Z \mid \theta)p(X = x_i \mid Z;\theta)dZ$ is intractable [7]. Another advantage is that AEVB can be trained by minibatches for large data sets. $q_w(Z \mid x_i)$ is the variational approximation of $p(Z \mid x_i;\theta)$. Note that the approximation depends explicitly on x_i[12]. $q_w(Z \mid x_i)$ is implemented by a probabilistic encoder neural network[13]. The weights and biases of the encoder neural network are the variational parameters w. In the training phase the probabilistic encoder maps each input x_i to a probabilistic (variational) approximation $q_w(Z \mid x_i)$ of the true posterior $p(Z \mid x_i;\theta)$. In deep learning terminology the hidden state distribution $p(Z \mid x_i;\theta)$ is a probabilistic representation of the input x_i. x_i is *encoded* "truly" by $p(Z \mid x_i,\theta)$ approximated by $q_w(Z \mid x_i)$.

Additionally, there is a second neural network that acts as a probabilistic decoder. The task of the decoder is to reconstruct the input x_i from a sample $z_i \sim q_w(Z \mid x_i)$, i.e. the decoder implements $p(x_i \mid Z;\theta)$.
For training, the (negative) evidence lower bound as cost function is used as follows [14]:

$$\mathcal{L}(w, Z) = \sum_i \Big(\mathbb{E}_{q_w(Z|x_i)}[\log p(x_i \mid Z;\theta)] - D_{KL}[q_w(Z \mid x_i) \mid\mid p(Z)]\Big) \tag{24}$$

The first term on the right hand side is a measure of the reconstruction error. The second term acts as a constraint (regularization), the representations of the training data should be similar to samples from a prior. As prior we typically choose a centered isotropic multivariate Gaussian. That is a Gaussian with zero mean-vector and diagonal covariance matrix $\mathcal{N}(0, \mathbb{1})$ with entries 1. For optimization w.r.t. the variational parameters w we must compute the gradient of equation 24. The second term can be solved analytically for a Gaussian prior and Gaussian conditionals $p(x_i \mid Z;\theta)$ [7]. The closed form allows direct automatic differentiation for optimization.
The gradient of the first term can be approximated by Monte Carlo estimation, see section ii.1. However, the *score function estimator* has a variance which is too high and can therefore not be used. This is solved by the reparametrization technique for variance reduction (see section ii.2) . With reparametrization we can also backprop through the encoder decoder network. For details see [7] and our exercise exercise-variational-autoencoder

A learned auto-encoder can generate as many samples as wanted. First, a sample is drawn from the prior $p(Z)$, i.e., the multivariate Gaussian with a diagonal covariance matrix $\mathcal{N}(0, \mathbb{1})$. The decoder then maps the Gaussian sample to a sample from the learned distribution.

ii. Bayesian Neural Networks

Neural networks are typically learned by minimizing a cost function, e.g., the cross-entropy loss with additional regularization. This is equivalent to a MAP point estimate, see section iii.
In Bayesian Neural Networks the weights are represented by probability density functions [5]. Typically the mean-field approximation (see section iv.1) is used for Bayesian neural networks.

[11] There is also an arrow from θ to Z_i.

[12] So we need only a global set of variational parameters which can be used for each data point i. In the literature this is also called *amortization*.

[13] Note that the mean-field approximation is not used and the approximation is therefore not necessarily factorial.

[14] The reader is strongly encourage to compare this with formula 14 and to derive the equivalence.

The complete probability distribution factorizes such that each weight corresponds to a univariate Gaussian [15]. Each weight θ_j has two (variational) parameters w_j the mean and the variance of the Gaussian $w_j = \{mu_j, \sigma_j^2\}$, i.e. $p(\theta) \approx \prod_j q_{w_j}(\theta_j)$ with Gaussians $q_{w_j}(\theta_j) = \mathcal{N}(\mu_j, \sigma_j^2)$. Such networks can be trained by backprop for learning of the variational parameters w_j. As cost the lower bound (analog to equation 24) can be used. For details see [3] and our exercise exercise-bayesian-by-backprop.

VII. Conclusion and Outlook

In the paper we gave a guide to learn Bayesian methods with a focus on machine and deep learning. Mainly, we provide a set of different exercises for studying the material in detail. We provided an overview from the basics to the use in deep learning in a consistent mathematical notation. The use of the material in a Master's course at the HTW Berlin resulted in initial feedback on the content. In the future, we want to review, improve and extend the exercises and provide more exercises related to Bayesian learning and the corresponding fundaments.

Most of the exercises were developed in the project Deep.Teaching. The project Deep.Teaching is funded by the BMBF, project number 01IS17056.

References

[1] A. G. Baydin, B. A. Pearlmutter, and A. A. Radul. Automatic differentiation in machine learning: a survey. *Journal of Machine Learning Research*, 18:153:1–153:43, 2015.

[2] E. Bingham, J. P. Chen, M. Jankowiak, F. Obermeyer, N. Pradhan, T. Karaletsos, R. Singh, P. Szerlip, P. Horsfall, and N. D. Goodman. Pyro: Deep Universal Probabilistic Programming. *Journal of Machine Learning Research*, 2018.

[3] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra. Weight uncertainty in neural networks. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 1613–1622. JMLR.org, 2015.

[4] A. D. Gordon, T. A. Henzinger, A. V. Nori, and S. K. Rajamani. Probabilistic programming. In *Proceedings of the on Future of Software Engineering*, FOSE 2014, pages 167–181, New York, NY, USA, 2014. ACM.

[5] G. E. Hinton and D. van Camp. Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the Sixth Annual Conference on Computational Learning Theory*, COLT '93, pages 5–13, New York, NY, USA, 1993. ACM.

[6] M. D. Homan and A. Gelman. The no-u-turn sampler: Adaptively setting path lengths in hamiltonian monte carlo. *J. Mach. Learn. Res.*, 15(1):1593–1623, Jan. 2014.

[7] D. P. Kingma and M. Welling. Auto-encoding variational bayes, 2013. cite arxiv:1312.6114.

[8] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. E. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. B. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, C. Willing, and et al. Jupyter notebooks - a publishing format for reproducible computational workflows. In *ELPUB*, 2016.

[15] Note that this is equivalent to a multivariate Gaussian distribution with diagonal covariance matrix for all weights.

[9] D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. The MIT Press, 2009.

[10] R. M. Neal. MCMC using Hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*, 54:113–162, 2010.

[11] R. M. Neal and G. E. Hinton. Learning in graphical models. chapter A View of the EM Algorithm That Justifies Incremental, Sparse, and Other Variants, pages 355–368. MIT Press, Cambridge, MA, USA, 1999.

[12] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge University Press, 3 edition, 2007.

[13] R. Ranganath, S. Gerrish, and D. M. Blei. Black box variational inference. In *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics, AISTATS 2014, Reykjavik, Iceland, April 22-25, 2014*, volume 33 of *JMLR Workshop and Conference Proceedings*, pages 814–822. JMLR.org, 2014.

[14] D. Wingate and T. Weber. Automated variational inference in probabilistic programming. *CoRR*, abs/1301.1299, 2013.