Learning from data: Bayesian Neural Networks

Christian Forssén

Department of Physics, Chalmers University of Technology, Sweden

Oct 22, 2019

1 Bayesian neural networks

The introduction part of this lecture is inspired by the chapter *Learning as Inference"* in the excellent book Information Theory, Inference, and Learning Algorithms by David MacKay.

Some python libraries that are relevant for Bayesian Neural Networks (as part of the general trend towards Probabilistic Programming) are:

- PyMC3
- Tensorflow Probability
- Keras (for constructing tensorflow models).
- Edward

1.1 Basic neural network

We will consider neurons with a vector of I input signals $\mathbf{x} = \left\{ x^{(i)} \right\}_{i=1}^{I}$, and an output signal $y(a)$, which is a (often non-linear) function of the *activation*

$$
a = w_0 + \sum_{i=1}^{I} w_i x_i,
$$

where $\mathbf{w} = \{ w_i \}_{i=1}^{I}$ are the weights of the neuron and we have included a bias ($b \equiv w_0$).

The training of the network implies feeding it with training data and finding the sets of weights and biases that minimizes a loss function that has been selected for that particular problem. Consider, e.g., a classification problem where the single output y of the final network layer is a real number $\in [0, 1]$ that indicates the (discrete) probability for input $\mathbf{x}$ belonging to either class $t = 1$ or $t = 0$:

$$p_{t=1} \equiv p(t = 1|w, x) = y \tag{1}$$
$$p_{t=0} \equiv p(t = 0|w, x) = 1 - y, \tag{2}$$

where we have simplified the notation, not using boldface anymore for the arrays w and x. A simple neural network classifier can be trained by minimizing the loss function

$$C_W(w) = C(w) + \alpha E_W(w),$$

made up of an error function

$$C(w) = -\sum_n \left[t^{(n)} \log(y(x^{(n)}, w)) + (1 - t^{(n)}) \log(1 - y(x^{(n)}, w)) \right],$$

where $t^{(n)}$ is the training data, and the regularizer

$$E_W(w) = \frac{1}{2} \sum_i w_i^2,$$

that is designed to avoid overfitting. The error function can be interpreted as minus the log likelihood

$$p(D|w) = \exp\left[-C(w)\right].$$

Similarly the regularizer can be interpreted in terms of a log prior probability distribution over the parameters

$$p(w|\alpha) = \frac{1}{Z_W(\alpha)} \exp\left[-\alpha E_W\right].$$

If E_W is quadratic as given above, then the corresponding prior distribution is a Gaussian with variance $\sigma_W^2 = 1/\alpha$ and $1/Z_W = (\alpha/2\pi)^{K/2}$, where K is the number of parameters in w. The objective function $C_W(w)$ then corresponds to the inference of the parameters w given the data

$$p(w|D, \alpha) = \frac{p(D|w)p(w|\alpha)}{p(D|\alpha)} = \frac{1}{Z_M} \exp[-C_W(w)].$$

The network parameters w that are found by minimizing $C_W(w)$ can be interpreted as the (locally) most probable parameter vector w^*. We show the different probability distributions for a typical training sequence in the following figure (reproduced from Information Theory, Inference, and Learning Algorithms by David MacKay)

Instead, we will use the Bayesian approach and consider the information that is contained in the probability distribution.

In fact, there are different uncertainties that should be addressed:

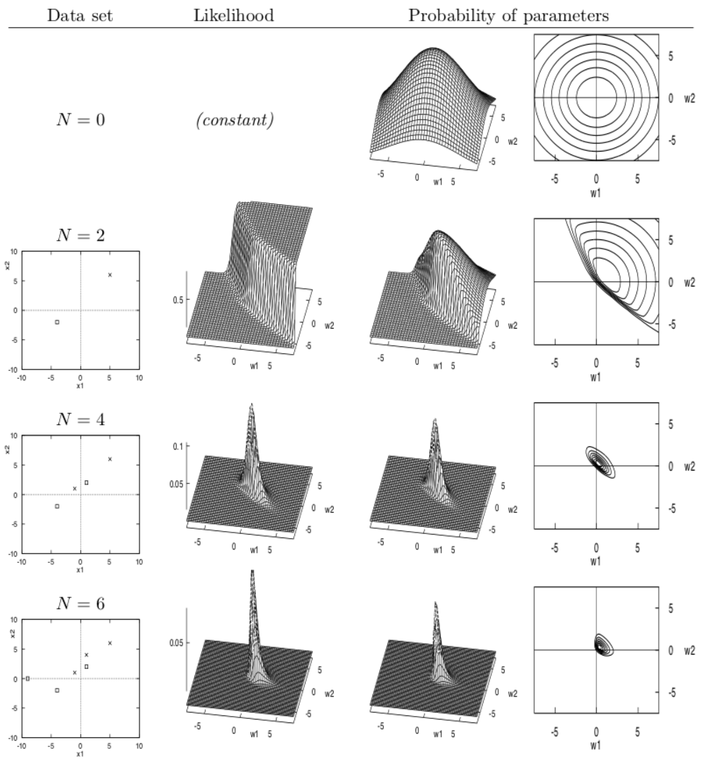

Figure 41.1. The Bayesian interpretation and generalization of traditional neural network learning. Evolution of the probability distribution over parameters as data arrive.

Figur 1: MacKay Fig. 41.1

Epistemic uncertainties: from uncertainties in the model. For a neural network, this uncertainty can, in principle, be reduced with more data and quantified using the Bayesian approach. Epistemic uncertainty is also known as **systematic uncertainty**.

Aleatoric uncertainties: from inherent noise in the training data. This should be included in the likelihood function (and is therefore part of the Bayesian approach). It can, however, not be reduced with more data of the same quality. Aleatoric uncertainty is also known as **statistical uncertainty**. Aleatoric is derived from the Latin *alea* or dice, referring to a game of chance.

Notice

We will use y to denote the output from the neural network. For classification problems, y will give the categorical (discrete) distribution of probabilities $p_{t=c}$ of belonging to class c. For regression problems, y is a continuous variable. It could also, in general, be a vector of outputs. The neural network can be seen as a non-linear mapping $y(x; w)$: $x \in \mathbb{R}^p \to y \in \mathbb{R}^m$.

1.2 Probabilistic model

A Bayesian neural network can be viewed as probabilistic model $p(y|\mathbf{x}, \mathbf{w})$.

Given a training dataset $\mathcal{D} = \left\{\mathbf{x}^{(i)}, y^{(i)}\right\}$ we can construct the likelihood function $p(\mathcal{D}|\mathbf{w}) = \prod_i p(y^{(i)}|\mathbf{x}^{(i)}, \mathbf{w})$ which is a function of parameters $\mathbf{w}$. Maximizing the likelihood function gives the maximimum likelihood estimate (MLE) of $\mathbf{w}$. The usual optimization objective during training is the negative log likelihood. For a categorical distribution this is the *cross entropy* error function, for a Gaussian distribution this is proportional to the *sum of squares* error function. MLE can lead to severe overfitting though.

Multiplying the likelihood with a prior distribution $p(\mathbf{w})$ is, by Bayes theorem, proportional to the posterior distribution $p(\mathbf{w}|\mathcal{D}) \propto p(\mathcal{D}|\mathbf{w})p(\mathbf{w})$. Maximizing $p(\mathcal{D}|\mathbf{w})p(\mathbf{w})$ gives the maximum a posteriori (MAP) estimate of $\mathbf{w}$. Computing the MAP estimate has a regularizing effect and can prevent overfitting. The optimization objectives here are the same as for MLE plus a regularization term coming from the log prior.

Both MLE and MAP give point estimates of parameters. If we instead had a full posterior distribution over parameters we could make predictions that take weight uncertainty into account. This is covered by the posterior predictive distribution $p(y|\mathbf{x}, \mathcal{D}) = \int p(y|\mathbf{x}, \mathbf{w})p(\mathbf{w}|\mathcal{D})d\mathbf{w}$ in which the parameters have been marginalized out. This is equivalent to averaging predictions from an ensemble of neural networks weighted by the posterior probabilities of their parameters $\mathbf{w}$.

Returning to the binary classification problem, $y^{(n+1)}$ corresponds to the probability $p_{t^{(n+1)}=1}$ and a Bayesian prediction of a new datum $y^{(n+1)}$ will correspond to a pdf and involves *marginalizing* over the K weight (and bias) parameters

$$p(y^{(n+1)}|x^{(n+1)}, D, \alpha) = \int d^K w p(p^{(n+1)}|x^{(n+1)}, w, \alpha)p(w|D, \alpha),$$

where we have also included the weight decay hyperparameter α from the prior (regularizer). Marginalization could, of course, also be performed over this parameter.

We show an example of such inference, comparing the point estimate $y(x; w^*, \alpha)$ and the Bayesian approach, in the follwoing figure (again from MacKay)

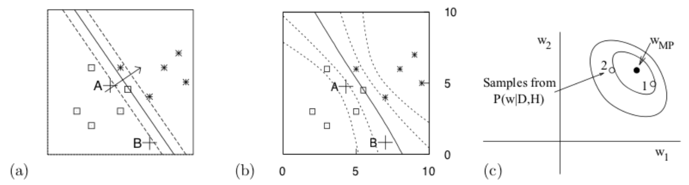

Figure 41.2. Making predictions. (a) The function performed by an optimized neuron $\mathbf{w}_{\rm MP}$ (shown by three of its contours) trained with weight decay, $\alpha = 0.01$ (from figure 39.6). The contours shown are those corresponding to $a = 0, \pm 1$, namely $y = 0.5, 0.27$ and 0.73. (b) Are these predictions more reasonable? (Contours shown are for $y = 0.5, 0.27, 0.73, 0.12$ and 0.88.) (c) The posterior probability of $\mathbf{w}$ (schematic); the Bayesian predictions shown in (b) were obtained by averaging together the predictions made by each possible value of the weights $\mathbf{w}$, with each value of $\mathbf{w}$ receiving a vote proportional to its probability under the posterior ensemble. The method used to create (b) is described in section 41.4.

Figur 2: MacKay Fig. 41.2

1.3 Bayesian neural networks in practice

But how shall we compute the marginalization integral for serious neural networks with thousands of parameters?

In short, there are three different approaches:

1. **Sampling methods**, e.g. MCMC sampling (this approach would be exact as the number of samples $\to \infty$);
2. **Deterministic approximate methods**, for example using Gaussian approximations with the Laplace method;
3. **Variational methods**.

The first two are discussed in MacKay's book, while we will focus on the variational methods in the following.

1.4 Variational inference for Bayesian neural networks

Bayesian neural networks differ from plain neural networks in that their weights are assigned a probability distribution instead of a single value or point estimate. These probability distributions describe the uncertainty in weights and can be used to estimate uncertainty in predictions. Training a Bayesian neural network via variational inference learns the parameters of these distributions instead of the weights directly.

Unfortunately, an analytical solution for the weight posterior $p(\mathbf{w}|\mathcal{D})$ in neural networks is intractable. We therefore have to approximate the true posterior with a variational distribution $q(\mathbf{w}|\boldsymbol{\theta})$ of **known functional form** whose parameters we want to estimate.

This can be done by minimizing the Kullback-Leibler divergence between $q(\mathbf{w}|\boldsymbol{\theta})$ and the true posterior $p(\mathbf{w}|\mathcal{D})$ w.r.t. $\boldsymbol{\theta}$.

The specific goal is then to replace $p(\mathbf{w}|\mathcal{D})$, which we don't know, with the known proxy distribution $q(\mathbf{w}|\boldsymbol{\theta}^*)$, where $\boldsymbol{\theta}^*$ is the optimal set of variational parameters. This optimal set of parameters is typically found by minimizing the **Evidence Lower Bound** (ELBO), which is equal to the Kullback-Leibler (KL) divergence up to a constant.

The Kullback-Leibler divergence. The KL divergence is a numeric measure of the difference between two distributions. For two probability distributions q and p, the KL divergence in a continuous case,

$$\mathrm{KL}(q||p) = \int_{-\infty}^{\infty} dx q(x) \log \frac{q(x)}{p(x)} dx \equiv \mathbb{E}_{q(x)} [\log q(x) - \log p(x)]$$

As we can see, the KL divergence calculates the expected log differences in between two distributions with respect to distribution q.

Intuitively there are three scenarios:

- if both q and p are high at the same positions, then we are happy;
- if q is high where p is low, we pay a price;
- if q is low we don't care (because of the expectation).

Note that we could try to reverse these arguments and compute $\mathrm{KL}(p||q)$. However, we choose $\mathrm{KL}(q||p)$ so that we can take expectations.

Variational free energy. Using, e.g., Jensen's inequality $f(\mathbb{E}[X]) \geq \mathbb{E}[f(X)]$ it can be shown that the corresponding optimization objective or cost function can be written as

$$\mathcal{F}(\mathcal{D}, \boldsymbol{\theta}) = \mathrm{KL}(q(\mathbf{w}|\boldsymbol{\theta}) \mid\mid p(\mathbf{w})) - \mathbb{E}_{q(\mathbf{w}|\boldsymbol{\theta})} \log p(\mathcal{D}|\mathbf{w}) \tag{1}$$

This is known as the *variational free energy*. The first term is the Kullback-Leibler divergence between the variational distribution $q(\mathbf{w}|\boldsymbol{\theta})$ and the prior $p(\mathbf{w})$ and is called the *complexity cost*. The second term is the expected value of the likelihood w.r.t. the variational distribution and is called the *likelihood cost*. By re-arranging the KL term, the cost function can also be written as

$$\mathcal{F}(\mathcal{D}, \boldsymbol{\theta}) = \mathbb{E}_{q(\mathbf{w}|\boldsymbol{\theta})} \log q(\mathbf{w}|\boldsymbol{\theta}) - \mathbb{E}_{q(\mathbf{w}|\boldsymbol{\theta})} \log p(\mathbf{w}) - \mathbb{E}_{q(\mathbf{w}|\boldsymbol{\theta})} \log p(\mathcal{D}|\mathbf{w}) \tag{2}$$

We realize that minimization of the variational free energy is obtained by finding the parameters θ that maximizes

$$\mathbb{E}_{q(\mathbf{w}|\boldsymbol{\theta})} \log \big(p(\mathbf{w})p(\mathcal{D}|\mathbf{w})\big) = \mathbb{E}_{q(\mathbf{w}|\boldsymbol{\theta})} \log p(\mathbf{w}, \mathcal{D}),$$

which encourages weights that fit data well.

and maximizes

$$-\mathbb{E}_{q(\mathbf{w}|\boldsymbol{\theta})} \log q(\mathbf{w}|\boldsymbol{\theta}) = -\int d\mathbf{w} q(\mathbf{w}|\boldsymbol{\theta}) \log q(\mathbf{w}|\boldsymbol{\theta}),$$

which we recognize as the entropy.

1.5 Bayesian neural networks in PyMC3

In the demonstration notebook of this lecture, it is shown how to use Variational Inference in PyMC3 to fit a simple Bayesian Neural Network. That implementation is based on the **Automatic Differentation Variational Inference** (ADVI) approach, described e.g. in Automatic Variational Inference in Stan.

1.6 Bayes by Backprop

The variational inference approach described in this section is adapted from the blog entry by Martin Krasser with some modifications. See also the original paper: Weight Uncertainty in Neural Networks (*Bayes by Backprop*).

We see that all three terms in equation (2) are expectations w.r.t. the variational distribution $q(\mathbf{w}|\boldsymbol{\theta})$. The cost function can therefore be approximated by drawing Monte Carlo samples $\mathbf{w}^{(i)}$ from $q(\mathbf{w}|\boldsymbol{\theta})$.

$$\mathcal{F}(\mathcal{D}, \boldsymbol{\theta}) \approx \frac{1}{N} \sum_{i=1}^{N} \Big[\log q(\mathbf{w}^{(i)}|\boldsymbol{\theta}) - \log p(\mathbf{w}^{(i)}) - \log p(\mathcal{D}|\mathbf{w}^{(i)})\Big] \tag{3}$$

In this example, we'll use a Gaussian distribution for the variational posterior, parameterized by $\boldsymbol{\theta} = (\boldsymbol{\mu}, \boldsymbol{\sigma})$ where $\boldsymbol{\mu}$ is the mean vector of the distribution and $\boldsymbol{\sigma}$ the standard deviation vector. The elements of $\boldsymbol{\sigma}$ are the elements of a diagonal covariance matrix which means that weights are assumed to be uncorrelated. Instead of parameterizing the neural network with weights $\mathbf{w}$ directly we parameterize it with $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ and therefore double the number of parameters compared to a plain neural network.

Network training. A training iteration consists of a forward-pass and and backward-pass. During a forward pass a single sample is drawn from the variational posterior distribution. It is used to evaluate the approximate cost function defined by equation (3). The first two terms of the cost function are data-independent and can be evaluated layer-wise, the last term is data-dependent and is evaluated at the end of the forward-pass. During a backward-pass, gradients of $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ are calculated via backpropagation so that their values can be updated by an optimizer.

Since a forward pass involves a stochastic sampling step we have to apply the so-called *re-parameterization trick* for backpropagation to work. The trick is to sample from a parameter-free distribution and then transform the sampled $\boldsymbol{\epsilon}$ with a deterministic function $t(\boldsymbol{\mu}, \boldsymbol{\sigma}, \boldsymbol{\epsilon})$ for which a gradient can be defined. Here, we choose $\boldsymbol{\epsilon}$ to be drawn from a standard normal distribution i.e. $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and the function t is taken to be $t(\boldsymbol{\mu}, \boldsymbol{\sigma}, \boldsymbol{\epsilon}) = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}$, i.e., it shifts the sample by mean $\boldsymbol{\mu}$ and scales it with $\boldsymbol{\sigma}$ where $\odot$ is element-wise multiplication.

For numerical stability we will parameterize the network with $\boldsymbol{\rho}$ instead of $\boldsymbol{\sigma}$ directly and transform $\boldsymbol{\rho}$ with the softplus function to obtain $\boldsymbol{\sigma} = \log(1+\exp(\boldsymbol{\rho}))$. This ensures that $\boldsymbol{\sigma}$ is always positive. As prior, a scale mixture of two Gaussians is used $p(\mathbf{w}) = \pi\mathcal{N}(\mathbf{w}|0, \sigma_1^2) + (1-\pi)\mathcal{N}(\mathbf{w}|0, \sigma_2^2)$ where σ_1, σ_2 and π are shared parameters. Their values are learned during training (which is in contrast to the paper where a fixed prior is used).

Uncertainty characterization. Uncertainty in predictions that arise from the uncertainty in weights is called epistemic uncertainty. This kind of uncertainty can be reduced if we get more data. Consequently, epistemic uncertainty is higher in regions of no or little training data and lower in regions of more training data. Epistemic uncertainty is covered by the variational posterior distribution. Uncertainty coming from the inherent noise in training data is an example of aleatoric uncertainty. It cannot be reduced if we get more data. Aleatoric uncertainty is covered by the probability distribution used to define the likelihood function.