TEK5040 Assignment, Bayesian Deep Learning

Narada Warakagoda

October 26, 2020

NOTE: This assignment is optional. Therefore, no submission is required. But you are strongly encouraged to perform the tasks in this assignment

1 Introduction

This assignment deals with Bayesian deep learning with neural networks. You are going to investigate Monte Carlo Dropout method and Variational Inference methods in this exercise. The code is based on *http://krasserm.github.io/2019/03/14/bayesian-neural-networks/*

1.1 Package Contents

`mc_train_test.py`	Tensorflow script for training and testing of Monte Carlo Dropout.
`vi_train_test.py`	Tensorflow script for training and testing for Variational Inference
`common.py`	Common routines and definitions, including data generation
`densevariational.py`	Tensorflow script defining a layer supporting Variational inference
`bayesian_exercise.pdf`	This document describing the task

2 Operation

We use a simple regression task for evaluating Bayesian deep learning techniques. The training data set is generated using `gen_data()` in `common.py`. Figure 1 shows the generated training data which follow a noisy sinusoidal. The task is to predict values of y axis given values of x-axis. We use Bayesian learning to generate the variance of the predicted values.

You can run `mc_train_test.py` in a Tensorflow environment to train and test the system based on Monte Carlo Drop out. i.e. `python3 mc_train_test.py`. Make sure that you have `tensorflow_probability` installed using for example `pip`.

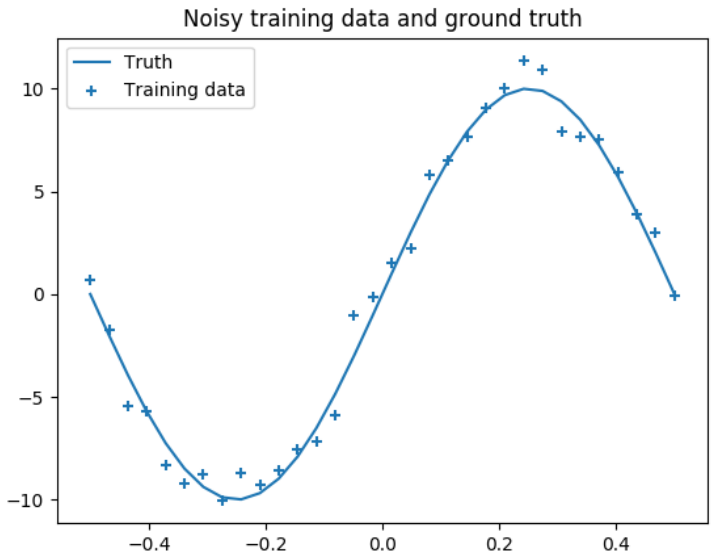

Figure 1: Training data

`pip3 install tensorflow_probability==x.x.x`.

where `x.x.x` is the version which should be selected so that it matches your `tensorflow` version (see `https://github.com/tensorflow/probability/releases` for version compatibility).

Similarly you can run `python3 vi_train_test.py` for Variational Inference based training and testing.

If the scripts work correctly, you should see the typical Keras output of training losses and the progress bar of testing for 500 times. Further, it shows the predicted results in graphical form.

3 Theory and Implementation

3.1 Monte Carlo Dropout

This is a very straight-forward implementation where dropout layers in the architecture absorb the uncertainty. Dropout layers are used in testing as well to calculate the output variance.

3.2 Variational Inference

We use *Bayes by Backprop* approach with the *reprameterization trick*. `densevariational.py` implements a layer supporting these techniques. In the implementation, we assume the prior probability of the parameters (weights and biases) of the network

$$p(\mathbf{w}) = \prod_i p(w_i) = \prod_i \left[\pi_1 \mathcal{N}(w_i|0,\sigma_1) + (1-\pi_1)\mathcal{N}(w_i|0,\sigma_2)\right] \tag{1}$$

where $\mathcal{N}(w_i|0,\sigma)$ is a normal (Gaussian) distribution with zero mean and variance σ for each network parameter w_i. Further, π_1, σ_1 and σ_2 are fixed hyper-parameters.

We also assume that each network parameter has an associated variational distribution $q(w_i)$ which is also assumed to be a normal (Gaussian) distribution $\mathcal{N}(w_i|\mu_i,\sigma_i)$. That is

$$q(\mathbf{w}) = \prod_i q(w_i) = \prod_i \mathcal{N}(w_i|\mu_i,\sigma_i) \tag{2}$$

The parameter vector of the variational distribution $\lambda = (\mu_i, \sigma_i), \quad i = 1, 2, \cdots, N$, with N being the number of parameters. As usual, we approximate the posterior network parameter distribution $p(\mathbf{w}|\mathcal{D})$ with the variational distribution $q(\mathbf{w})$ and aim to find the parameter set λ which minimizes the KL distance between the posterior and variational distributions.

As described in the lecture, KL distance is minimized indirectly by maximizing the Evidence Lower Bound (ELBO) which is given by

$$\hat{\mathcal{L}}(\lambda) = \frac{1}{S}\sum_{s=1}^{S}\left[\ln p\left(\boldsymbol{w}(\lambda,\boldsymbol{\epsilon}^s),\mathcal{D}\right) - \ln q\left(\boldsymbol{w}(\lambda,\boldsymbol{\epsilon}^s),\lambda\right)\right]$$

where each element of $\boldsymbol{\epsilon}^s$ is drawn from a standard normal distribution and $\boldsymbol{w}^s = \boldsymbol{w}(\lambda,\boldsymbol{\epsilon}^s)$ represents the re-parameterization trick. The above expression can also be written as

$$-\hat{\mathcal{L}}(\lambda) = \frac{1}{S}\sum_{s=1}^{S}\left[\ln q\left(\boldsymbol{w}^s,\lambda\right) - \ln p\left(\mathcal{D}|\,\boldsymbol{w}^s\right) - \ln p\left(\boldsymbol{w}^s\right)\right] \tag{3}$$

Then we use back-propagation to minimize the loss $-\hat{\mathcal{L}}(\lambda)$.

The first term and the last term of equation 3 is evaluated using equation 2 and equation 1 respectively. These terms are data independent and therefore they are added to the model losses. The second term in equation 3 is data dependent and it is the negative log probability of data for the given parameter vector $\boldsymbol{w}^s$. This quantity is typically available after a forward pass through the network.

4 Task

1. Run `mc_train_test.py` and comment on the variance of the output as depicted in the generated figure. Hint: Note the difference in test and training set sizes.

2. Run `vi_train_test.py` and comment on the variance of the output. Compare it with MC-dropout.

3. Find the line numbers of the code in `densevariational.py` which implements re-parameterization-trick $\boldsymbol{w}^s = \boldsymbol{w}(\lambda, \boldsymbol{\epsilon}^s)$.

4. Find the line numbers of the code in `densevariational.py` which adds the first and the third loss components in equation 3 to the computational graph.

5. Find the line numbers of the code where the second loss component in equation 3 is added to the computation graph.

6. Save the computational graph and inspect it with `tensorboard`.

7. `vi_train_test.py` is using a custom layer implemented in `densevariational.py`. There are built-in layers provided by `tensorflow_probability` which perform similar types of variational inference. Two such layers are `tfp.layers.DenseVariational` and `tfp.layers.DenseFlipout`. Reimplement the system using each of them and compare the results with the custom implementation.