Bayesian Reparametrisation of Neural Networks

Anonymous Author(s)

ABSTRACT

This paper explores the possibility of reparametrising a standard neural network into a bayesian model to enhance its accuracy. It tries to reproduce the work of Bruss, et. al.[1], in which they augment a simple neural network using Bayesian reparametrisation with a normal distribution to output a probability distribution rather than just singular values, in order to make the model aware of its own uncertainty due to aleatoric uncertainty within the dataset. This resulted in an increase of accuracy for their model.

Ultimately, this paper was not able to reproduce these results on either the original MNIST dataset or the additionally tested CIFAR-10 dataset. All attempts to add Bayesian reparametrisation to a neural network have not yielded any noticeable increase in accuracy.

1 INTRODUCTION

In 2018, Bayan Bruss, Jason Wittenbach and James Montgomery published an article[1], in which they discuss the possibility of adding a reparametrisation function to a neural network in order to augment it with Bayesian analysis. They report a significant rise in accuracy for their enhanced model, enough to claim a spot in the top 35 most accurate models for the MNIST dataset, a standard training set for neural networks, consisting of handwritten digits. They managed to increase the accuracy of a regular two-layer convolutional neural network with max pooling and dropout from 97% to 99.33% using this method.

This paper attempts to reproduce these results, and explores the possibility of expanding them to other datasets. For this purpose, the CIFAR-10 dataset will also be examined.

2 MOTIVATION

Neural networks are a very powerful tool to create pattern matching and prediction models. They are, however, not perfect, and accuracy of their results is one of the top benchmarks for neural networks. Improving this accuracy is topic of much research, and there are many approaches towards that goal. One of these approaches is making the model aware of the uncertainty of its results. Bayesian deep learning and modeling is a way of quantifying uncertainty, that is hard to work with, because it needs to be computationally tractable. If, however, the quantification of uncertainty of Bayesian modeling could be combined with the ease of use of point estimate neural networks, accuracy might be able to be improved easily by just performing a simple reparametrisation.

3 THEORETICAL FOUNDATION

Uncertainty in machine learning appears in two major kinds: aleatoric and epistemic[4]. Epistemic uncertainty is the kind that can be reduced by more data and longer training times. It is a result of the mistakes that a model has not yet learned to correct. Aleatoric uncertainty, however, is intrinsic to the data itself. If the training data has a number of examples that have very similar input vectors but different output values, aleatoric uncertainty is present. Aleatoric uncertainty can be homoscedastic, meaning that the error is constant over the whole dataset, or heteroscedastic, meaning that it is more localised. An example of such heteroscedastic aleatoric uncertainty would be the handwritten numbers 1 and 7, with some examples of them only being separated by little more than a rotation. This is the kind of uncertainty, that the experiment is trying to improve upon.

Bayesian deep learning tries to capture this uncertainty, by having its models not just provide point values for its predictions, but rather distributions[8]. These models are, however, more difficult to use, since all their weights and hidden layers also need to be represented as distributions.[9]

Bruss, et al.[1] try to marry the quantification of uncertainty of Bayesian models with the ease of use of neural networks, by treating the neural network as a black box that delivers point value estimates. Out of these point value estimates a normal distribution is constructed, which is supposed to capture aleatoric uncertainty. By polling the distribution and checking each poll with the corresponding label, the neural network is incentivised to minimize uncertainty within its estimates. By this mechanism, an increase in accuracy is hoped to be achieved.

4 RELATED WORK

The idea of getting probability distributions from neural networks is reasonably old and Denker used it in 1991 to further develop the popular softmax scheme[2]. It was formalized into the field of bayesian learning in 1992 by MacKay[8]. Further focus on the uncertainty expressed within bayesian models was more recently placed by Gal[4] and Kendall[6].

Autoencoders, a way of compressing complex input to just a few nodes within a hidden layer, have adapted bayesian approaches to include distribution information within that hidden layer[7]. These so called variational autoencoders, use a very similar method to Bruss, et al.[1] to convert point values to distributions[3].

5 ORIGINAL EXPERIMENT BY BRUSS, ET AL.

Bruss, et al.[1] have augmented a two-layer convolutional neural network, with max pooling and dropout with two additional layers that take the last dense layer as input, and are interpreted as the means and standard deviations of a normal distribution. They then use these layers to construct said distribution and poll it 1000 times to get a sense of how uncertain the neural network is in its predictions. These predictions are then compared to the original labels of the training data, which have also been tiled 1000 times. The resulting neural network has a structure as shown in figure 1.

They also give a small code snippet of 7 lines, of their reparametrisation process as seen in figure 2. In it, the actions described before are found, as well as a previously undocumented transposition of the logits, which seems to be needed to fix the shape of them.

Data Science Seminar, 2019, Uni Passau
.

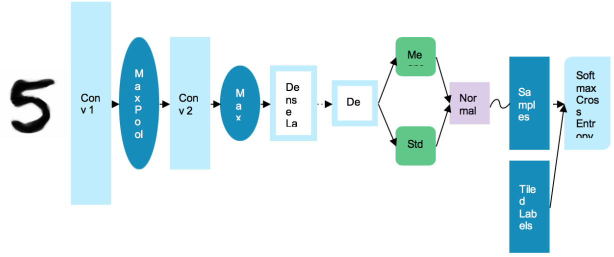
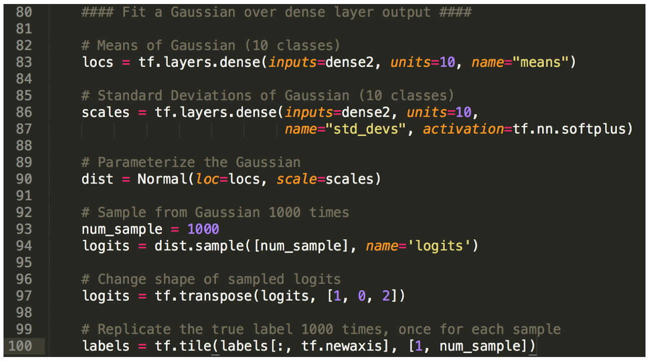


Figure 1: The structure of the modified neural network given by Bruss, et al. (https://medium.com/capital-one-tech/reasonable-doubt-get-onto-the-top-35-mnist-leaderboard-by-quantifying-aleatoric-uncertainty-a8503f134497).



```
    #### Fit a Gaussian over dense layer output ####

    # Means of Gaussian (10 classes)
    locs = tf.layers.dense(inputs=dense2, units=10, name="means")

    # Standard Deviations of Gaussian (10 classes)
    scales = tf.layers.dense(inputs=dense2, units=10,
                             name="std_devs", activation=tf.nn.softplus)

    # Parameterize the Gaussian
    dist = Normal(loc=locs, scale=scales)

    # Sample from Gaussian 1000 times
    num_sample = 1000
    logits = dist.sample([num_sample], name='logits')

    # Change shape of sampled logits
    logits = tf.transpose(logits, [1, 0, 2])

    # Replicate the true label 1000 times, once for each sample
    labels = tf.tile(labels[:, tf.newaxis], [1, num_sample])
```

Figure 2: Code snippet of the modified neural network given by Bruss, et al. (https://medium.com/capital-one-tech/reasonable-doubt-get-onto-the-top-35-mnist-leaderboard-by-quantifying-aleatoric-uncertainty-a8503f134497).

6 REPRODUCTION

The reproduction of the work of Bruss, et al.[1] proved difficult, since the article provided no extensive source code, and instead only gave a short code snippet (see figure 2) and a general description of their workings. Due to this, the exact frameworks used within the original work are uncertain.

The code fragment in the article indicates however that a version of Tensorflow was used, and seemingly without use of the newer Keras wrapper. That is why the first attempt at reproduction tried to stay within these constraints.

6.1 Attempt using only Tensorflow

For the first attempt, a reference implementation of a simple neural network from the official Tensorflow GitHub repository[10] was used. It was subsequently expanded with the code (see figure 2) given in the article by Bruss, et al.[1]. The resulting code can be viewed in this papers GitHub repository[5].

It quickly became apparent that significantly more changes than just the ones described within the given code snippet would be necessary. For this reason, the raw Tensorflow approach was abandoned, since making more changes to the neural network would be more intricate compared to using the newer Keras wrapper.

6.2 Attempt using Keras

Keras is a wrapper around Tensorflow, which comes bundled with Tensorflow, that simplifies building and training models greatly. This would be a large factor in being able to quickly make changes to the model, in order to implement Bayesian reparametrisation.

As a reference implementation of a Keras MNIST neural network, an official tutorial from Tensorflow[11] was used. It was subsequently changed to a non-sequential model, since the means and standard deviations used for the Normal distribution need to have the same input layers, which is impossible in a sequential model. Then, code adapted from the code snippet (see figure 2) was added. But since Keras expects the outputs of its models to be Layers and not logits, it would have been necessary to introduce elements of Keras backend. This was not done, since it would have bypassed large parts of the Keras model and training processes, but it may be possible to produce a working model this way.

6.3 Attempt using Keras lambda layers

At this point it was necessary to convert the normal distribution needed for the Bayesian reparametrisation into something that would work like a Keras layer. Keras offers two ways of writing own custom layers, lambda layers and custom layers. Lambda layers were chosen for this task, since the layer would not need to be trained itself and instead only perform the same task of building a normal distribution from the two previous layers and poll it 1000 times.

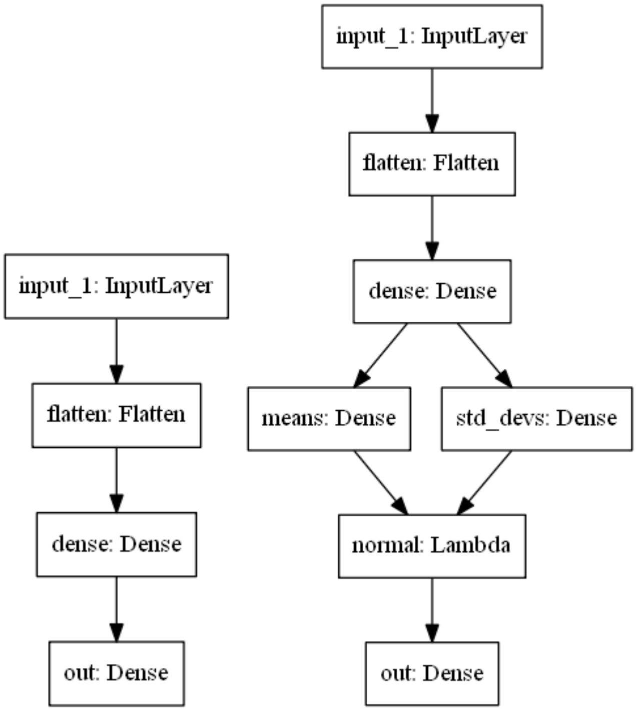


Figure 3: A layer graph of the model reproduced within this paper. Left: Control model without Bayesian reparametrisation. Right: Model with Bayesian reparametrisation

Table 1: Accuracy and training time of the control and reparametrised neural networks trained on the MNIST dataset. Times are indicative of a local computation on a quad core CPU.

	Control NN	Reparametrised NN
Training time	200 µs/sample	789 µs/sample
Testing time	41 µs/sample	340 µs/sample
Testing accuracy	97.9%	97.8%
Testing loss	0.0695	0.0872

This attempt yielded a working model as shown in figure 3 and its code can be viewed on this papers GitHub repository[5].

7 RESULTS

Now, with a working model, attempts at reproducing the result of Bruss, et al.[1] were performed. In the original article, the MNIST dataset was used. This paper will examine both that dataset and the CIFAR-10 dataset, in order to evaluate whether the Bayesian reparametrisation results in any significant rise in accuracy. Both of these datasets are standard benchmarking tools and come bundled within the newest versions of Tensorflow.

7.1 Dataset: MNIST

The MNIST (Modified National Institute of Standards and Technology) database is a set of handwritten digits, and a standard benchmarking dataset for machine learning. It contains 60000 training and 10000 testing images like shown in figure 4. This dataset should be ideal for this task, since it contains class pairs in it such as 1 and 7 with high aleatoric uncertainty, meaning classes with very similar input vectors but different classifications. Since Bayesian reparametrisation aims to improve these uncertainties, this dataset should show a promising increase in accuracy if the method is valid.

Figure 4: Examples of handwritten digits within the MNIST dataset

The original article reports an increase from 97% accuracy to 99.7% accuracy on the MNIST dataset. In table 1 the results of both a control neural network and the neural network reproduced for this paper can be seen. The training times are reflective of a local training session on a quad core CPU, and can be much higher on a GPU, yet the relation between the control times and the reparametrised times should stay similar.

Table 2: Accuracy and training time of the control and reparametrised neural networks trained on the CIFAR-10 dataset. Times are indicative of a local computation on a quad core CPU.

	Control NN	Reparametrised NN
Training time	782 µs/sample	1.2 ms/sample
Testing time	98 µs/sample	553 µs/sample
Testing accuracy	45.27%	44.37%
Testing loss	1.5597	1.5661

As can be seen, no significant rise in accuracy was achieved, instead the reparametrised neural network even scored a slightly lower accuracy and higher loss, but those figures were well within the margin of variance for a neural network. What was significant however, was the rise in time trained per sample. The reparametrised model took about 4 times as long to train and about 8 times as long to test on the MNIST dataset than the control model.

7.2 Dataset: CIFAR-10

In addition to the MNIST dataset, the reproduced model was tested on the CIFAR-10 dataset. It is a subset of the 80 million tiny images dataset, an consists of 60000 images which fall into one of ten classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship or truck. examples can be seen in figure 5. There also exists a version of CIFAR with 100 classes, but other than lower overall accuracy and possibly longer training times, there should not be a significant difference in results.

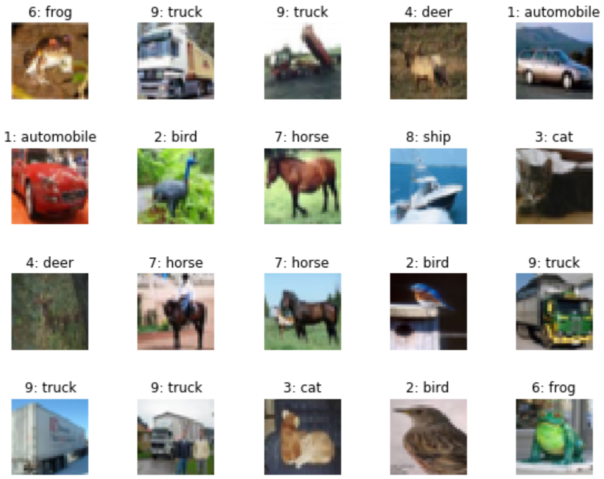


Figure 5: Examples of images within the CIFAR-10 dataset

The results after 5 epochs of training on the CIFAR-10 dataset can be seen in table 2. At this point the model is still under fitting, but any improvement in accuracy of the reparametrised network over the control network should already be visible at this point. Unsurprisingly, there is no significant gain in accuracy due to the reparametrisation on this dataset either. The only thing that consistently rises is the time it takes to train and test the model.

7.3 Summary

The reproduced neural network with Bayesian reparametrisation was not able to increase its accuracy on either of the tested datasets, compared to the control model. The only consistent effect of the reparametrisation seemed to be a increase in training and testing time due to the creation and polling of the normal distribution. Therefore, it is not recommended to apply this method to neural networks.

8 CONCLUSION

This paper was not able to reproduce the increase in accuracy from combining Bayesian modeling with a neural network. The methods described by Bruss, et al.[1] seem to result in the same accuracy as regular neural networks, but with a significantly longer training time. This does not, however, mean that the approach is not valid. It may very well be possible that the reconstruction of the neural network was not done correctly, due to the sparse information about the source code.

The theory behind the experiment shows promise, though, and might be able to be realized through other means. Variational autoencoders, for example, use a very similar technique to Bayesian reparametrisation, where they take the bottleneck layer of a regular autoencoder and turn it into a normal distribution. Results from this field might be used in further research.

REFERENCES

[1] Bayan Bruss, Jason Wittenbach, James Montgomery, and Capital One. 2018. Reasonable Doubt: Get Onto the Top 35 MNIST Leaderboard by Quantifying Aleatoric Uncertainty. (Dec. 2018). Retrieved June 15, 2019 from https://medium.com/capital-one-tech/reasonable-doubt-get-onto-the-top-35-mnist-leaderboard-by-quantifying-aleatoric-uncertainty-a8503f134497

[2] John S Denker and Yann Lecun. 1991. Transforming neural-net output levels to probability distributions. In *Advances in neural information processing systems*. 853–859.

[3] Carl Doersch. 2016. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908* (2016).

[4] Yarin Gal and Zoubin Ghahramani. 2016. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*. 1050–1059.

[5] Moritz Grünbauer. 2019. Bayesian Reparametrisation of Neural Networks. (2019). Retrieved June 18 , 2019 from https://github.com/MoritzGr/BayesianReparametrisation

[6] Alex Kendall and Yarin Gal. 2017. What uncertainties do we need in bayesian deep learning for computer vision?. In *Advances in neural information processing systems*. 5574–5584.

[7] Diederik P Kingma and Max Welling. 2013. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114* (2013).

[8] David JC MacKay. 1992. Bayesian interpolation. *Neural computation* 4, 3 (1992), 415–447.

[9] Radford M Neal. 2012. *Bayesian learning for neural networks*. Vol. 118. Springer Science & Business Media.

[10] Tensorflow. 2016. Convolutional Neural Network Estimator for MNIST, built with tf.layers. (2016). Retrieved May 25 , 2019 from https://github.com/tensorflow/tensorflow/blob/master/tensorflow/examples/tutorials/layers/cnn_mnist.py

[11] Tensorflow. 2018. Get Started with TensorFlow. (2018). Retrieved May 25 , 2019 from https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/_index.ipynb