Bayesian Convolutional Neural Networks for Image Classification

A Case Study on CIFAR-10

DS30: Deep Learning

American University of Armenia

Instructor: Elen Vardanyan

Shoghik Gevorgyan

December 2024

Abstract

This report discusses the implementation of a Bayesian Neural Network (BNN) utilizing PyTorch. The focus is on Bayesian layers employing the Flipout method for variance reduction, integration into a CNN architecture, and its application to the CIFAR-10 dataset. Key concepts include mathematical formulations, training procedures, and visualization of predictions and uncertainties.

1 Introduction

Bayesian Neural Networks (BNNs) treat the weights of the model as probabilities, providing a way to measure uncertainty in neural networks. By using Bayesian methods, BNNs try to find a balance between the complexity of the model and how well it fits the data. This report explains how a BNN model was built, using custom convolutional and fully connected layers with Flipout reparameterization to estimate uncertainty efficiently.

This report details the implementation of a BNN architecture that includes custom convolutional and dense Bayesian layers with Flipout reparameterization for efficient uncertainty estimation.

2 Mathematical Formulation

2.1 Bayesian Inference

In BNNs, the posterior distribution over weights W given data D is:

$$P(W|D) = \frac{P(D|W)P(W)}{P(D)},$$

where $P(D|W)$ is the likelihood, $P(W)$ is the prior, and $P(D)$ is the evidence. The objective is to approximate this posterior to make predictions.

2.2 Kullback-Leibler (KL) Divergence

The KL divergence regularizes the approximate posterior $q(W)$ to stay close to the prior $P(W)$:

$$KL(q(W)||P(W)) = \int q(W) \log \frac{q(W)}{P(W)} \, dW.$$

This term is incorporated into the loss function during training.

2.3 Flipout Reparameterization

The Flipout method reduces the variance of stochastic gradient estimates by decoupling weight sampling from input sampling. For weights W with mean μ and standard deviation σ, the sampled weights are:

$$W_s = \mu + \sigma \cdot (\epsilon + flip(\epsilon)),$$

where ϵ is a random noise vector, and $flip(\cdot)$ introduces structured randomness.

3 Implementation

3.1 Bayesian Layers

Custom layers include:

- **BayesianConv2dFlipout**: A convolutional layer with reparameterized weight sampling.
- **BayesianDenseFlipout**: A fully connected layer with similar mechanics.

The Flipout method is integrated to ensure efficient sampling.

3.2 CNN Architecture

The Bayesian CNN consists of:

- Four convolutional layers with increasing filters: $3 \to 8 \to 8 \to 16$.
- Fully connected layers mapping $16 \times 8 \times 8$ to the output classes.

Input-output dimensions for each layer are:

$$Conv2d : (B, C, H, W) \to (B, C', H', W'), \quad Dense : (B, D) \to (B, D'),$$

where B is the batch size, C, C' are input and output channels, and H, W are spatial dimensions.

4 Training and Evaluation

4.1 Training Procedure

The training loop minimizes a loss function combining Negative Log-Likelihood (NLL) and KL divergence:

$$\mathcal{L} = NLL + \beta \cdot KL,$$

where β is an annealing factor.

4.2 Validation and Testing

Validation evaluates model performance using accuracy and loss metrics. Testing involves predictions on unseen data to quantify generalization.

4.3 Visualization of Predictions

Predictions are visualized with true and predicted labels, highlighting incorrect classifications.

5 Results

The Bayesian CNN achieved a validation accuracy of approximately 70% and a test accuracy of around 68.5% on the CIFAR-10 dataset during 50 training epochs. These results indicate that the model performs reasonably well for this classification task. The model also provides uncertainty estimates, which can offer insights into its confidence in the predictions.

Visualization of the model's predictions helps to better understand how the network performs and where it is more uncertain. Examples of predictions along with uncertainty estimates are shown in Figure 1. Example predictions are shown in Figure 1.

Figure 1: BNN- CIFAR10 Predictions

6 Conclusion

This report demonstrates the implementation of a Bayesian Neural Network with Flipout reparameterization. The model shows robust performance on CIFAR-10 while providing uncertainty estimates. This implementation can be used for solving various image classification tasks.

7 Reproducibility

To facilitate reproducibility, this section provides instructions for downloading the dataset, cloning the project from GitHub, and running the code.

7.1 Dataset Download

The CIFAR-10 dataset is a publicly available image dataset commonly used for benchmarking image classification models. To download the dataset:

1. The dataset will be automatically downloaded and preprocessed using the PyTorch `torchvision.datasets.CIFAR10` class during runtime.
2. Alternatively, it can be downloaded manually from the link below: `https://www.dropbox.com/scl/fi/ws1183q0ulzjgh1sug8r4/cifar10.zip?rlkey=iwqddpmlnn3quo41sn2hjqyme&st=7w8ycd6q&dl=1`.

7.2 Cloning the Codebase from GitHub

The full project, including the implementation and visualization scripts, is posted on GitHub. To clone the repository:

1. Ensure `git` is installed on your system.
2. Run the following command in the terminal:

```
git clone git@github.com:shoghik11/DL_BNN.git
```

7.3 Running the Code

To run the project, follow these steps:

1. Open the file `BNN_Classifier.ipynb`.
2. Execute the code cells in the notebook one by one.

3. Ensure that all dependencies (PyTorch, Matplotlib) are installed in your environment.

4. Run the training loop and monitor the results.

7.4 Expected Outputs

1. During training, the script outputs metrics such as training loss, validation loss, and accuracy at each epoch.

2. The test script visualizes predictions with true and predicted labels, highlighting any misclassifications.

References

- **CIFAR-10 Dataset:** `https://www.cs.toronto.edu/~kriz/cifar.html`
- **GitHub Repository:** `https://github.com/shoghik11/DL_BNN`
- **Jospin, L. V., Laga, H., & Boussaid, F. (2020).** Hands-on Bayesian Neural Networks – A Tutorial for Deep Learning Users. `https://arxiv.org/pdf/2007.06823`