EE-559 Deep Learning Mini-Project 1

Haeeun Kim, Valentin Loftsson, and Antoine Madrona

May 2021

Introduction

In this project, we develop a deep neural network in PyTorch that takes two 14×14 images from the MNIST dataset as input and predicts if the digit in the first image is lesser or equal to the digit in the second image. We propose different network architectures and compare their performances. In particular, we will assess the performance improvement that can be achieved by using weight sharing and auxiliary loss.

Models and Methods

The overall structure of the network is similar in all the presented models: one convolutional block followed by a dense neural network to implicitly classify the images, and a shallow network to combine the intermediary outputs of both images and generate the binary prediction.

The convolutional network is identical across all the models presented: it contains two convolutions with a kernel of size 3×3 and padding of 1, each followed by max pooling with a kernel 2×2 and a stride of 2.

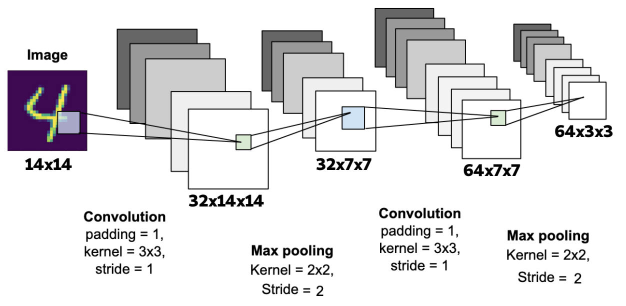

Figure 1: Convolutional network structure

The convolutional block is followed by a linear neural network (fully-connected block with 576 input units and 10 output units, 576×10) with either one or two hidden layers (100×50 or $100 \times 70 \to 70 \times 50$) using the ReLU activation function. The ten values of the output can be thought of as corresponding to the ten digits (0 to 9), i.e. the model's interpretation of the image. For this reason, it is important to keep the ten values for each image and also because they are needed to compute the auxiliary loss.

Finally, a second linear neural network (two fully-connected layers, $20 \times 10 \to 10 \times 2$ with ReLU activation function) generates the binary prediction from the concatenated classification results of both images. The softmax function is then applied to the output to convert it into a probability distribution over the two classes.

The objective function we selected for training the model is the cross-entropy loss, since it is appropriate for the classification task. This loss function will also be used later for the auxiliary loss, implemented to take advantage of the available information about the class of each image (0-9). Furthermore, stochastic gradient descent in combination with the back-propagation algorithm are used to train the models.

Model Specifics and Performance

All the models are trained using 1000 samples, 25 epochs and a batch size of 100. Other hyperparameters are optimized through K-fold cross-validation, where $K = 4$. For each model, we use the grid search algorithm to find the optimal combination of the learning rate (γ) and the auxiliary loss scaling parameter (λ). Clearly, λ is not tuned for models that are not trained with an auxiliary loss.

$$\gamma \in [0.0002, 0.0005, 0.0031, 0.012, 0.1]$$

$$\lambda \in [0, 0.115, 0.23, 0.345]$$

Finally, the performance of each model is estimated on randomly generated training and test MNIST pairs, over 10 rounds. The models and their performances are discussed in the following:

- **Naive network** (`NaiveNet(1)`): Uses two different linear networks to predict the class of the images (no weight sharing). With $\gamma = 0.0002$, the model achieved $73.90 \pm 4.01\%$ test accuracy.

- **Siamese network with 1 hidden layer** (`SharedWeightNet(1)`): Exploiting the fact that the images in each pair have the same type of signal (both are MNIST images), the weights are shared between the two channels to improve performance. Another advantage is that it reduces the total number of parameters in the model. With $\gamma = 0.0005$ the model achieved $80.44 \pm 1.82\%$ test accuracy.

- **Siamese network with 1 hidden layer** (`SharedWeightNet(1)`) **trained with auxiliary loss**: Uses an auxiliary cross-entropy loss to take advantage of the available *class* information of each image. In addition to γ, hyperparameter λ is tuned for the optimization of the auxiliary loss. With $\gamma = 3.1e-3$ and $\lambda = 0.23$ the model achieved $86.40 \pm 1.50\%$ test accuracy.

- **Siamese network with 2 hidden layers** (`SharedWeightNet(2)`) **trained with auxiliary loss**: The model complexity is increased by adding a hidden layer. With $\gamma = 1.2e-2$ and $\lambda = 0.23$ the model achieved $90.57 \pm 0.98\%$

test accuracy. Despite the increased complexity of the model, the variance of the final evaluation is smaller than that of `SharedWeightNet(1)`, implying an optimal choice of architecture in the given hyperparameter grid.

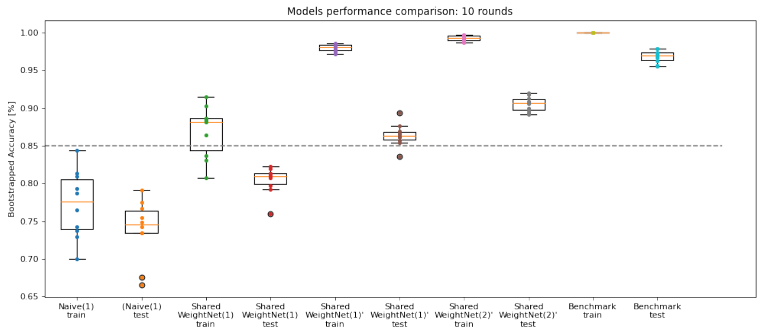

Figure 2: Boxplots illustrating the performance of each model over 10 rounds. Weight sharing slightly improved the performance from approximately 78% to 83%. The auxiliary loss further increased the accuracy to 86%. Adding a hidden layer raised the accuracy to 91%.

Discussion

The performance of each model may be compared to a benchmark model. This model reuses the convolutional block and the siamese weight-shared block of the presented models. However, instead of using a fully-connected linear neural net at the end to generate the binary outcome, it applies the less-or-equal operator. As can be seen from Figure 2, it achieves $96.84 \pm 0.84\%$ accuracy. This is generally the expected performance of good MNIST image classifiers. Thus, the final prediction block is responsible for the drop of approximately 6% in the accuracy of our models and should be improved to better implement the less-or-equal operation.

Conclusion

By taking advantage of the nature of the data, we managed to achieve 91% test accuracy by implementing weight sharing and an auxiliary loss. We effectively implemented a simple linear network to approximate a less-or-equal comparison between two images encoding integers and showed that the performance bottleneck is at the level of this comparison module. Even though the training of the model remains fast, it still has approximately 88,000 parameters. This number could probably be reduced by improving the classification block of the model.