Deep Learning in Practice - Practical Session 1

Hyper-parameters and training basics with PyTorch

Denys Sikorskyi

1. Multi-classification

Problem 1 was to perform multi-classification on handwritten digits using 10 classes from the USPS Dataset. To accomplish this, I tuned our hyperparameters to produce a high-performing model. I designed different architectures with different number of convolutional layers based on previous research that demonstrated the effectiveness of convolutional layers in image classification. I chose the model with the highest accuracy on the validation set after training, which turned out to be the model with 3 convolutional layers. Following that, I tested different numbers of neurons and activation functions (sigmoid, tanh, and ReLU). The best validation accuracy was obtained using 64 neurons and the ReLU activation function. The architecture chosen includes three convolutional layers, with 32, 64 and 128 size and a kernel size of 3x3, each followed by a batch normalization layer and maxpooling. Following the third convolutional layer is a dropout layer with a dropout probability of 0.25. Finally, a fully connected layers are presented with 100 and 50 neurons respectively, followed by a softmax activation function. During our training process, I discovered that training the model for 50 epochs with a batch size of 10 yielded the best results. The SGD optimizer and MSE loss function were used with a learning rate of 0.1. The model achieved an impressive 98.32% accuracy on the validation set and 97.44% accuracy on the test set after training.

2. Regression

In the next task, I had to forecast house sales prices based on four factors with really different values, so I normalized the data to ensure that the feature values were all in the same range. I chose the best model and tested various architectures with fully connected layers and various activation functions using hyperparameter tuning. The best model had four layers, each with 100, 50, 25 and 25 neurons, and ReLU as the activation function. On the validation set, the model had a loss error of 1098213565. During training, the SGD optimizer produced nan values due to exploding gradients, so I only used the RMSprop and Adam optimizers. The best results were obtained with a model trained for 800 epochs with the Adam optimizer and a learning rate of 0.01 and a batch size of 64. Next, I added the Gaussian likelihood loss and modified the model to predict both the mean and log(sigma^2). On the validation set, the model had a loss of 2071349 and a loss of 2166721 on the test set.

3. Time vs Hyperparameters

The performance of models is determined by how their hyperparameters are set. Tuning these parameters is an important step, but it can be time-consuming because we must train the model with all possible parameter combinations. Because I tune many parameters with 3 or more possible values, we have lots of combinations to train the model, which would take a lot s to complete for the first problem, rendering it impossible. As a result, I proceeded to do random parameter selection, which took much less than could be done if I did with all possible combination.

4. Hyperparameters selection

Machine learning model performance is heavily influenced by hyperparameters such as batch size, learning rate, number of epochs, and optimization technique. I discovered that the ideal hyperparameters varied during our research with two separate issues. A higher batch size hindered empirical convergence and reduced model generalization in multi-classification situations. Higher batch sizes, on the other hand, were useful for regression difficulties. A slow and unstable training process resulted from a low learning rate, whereas a high learning rate resulted in instability and divergence. The best result for both issues was obtained with a learning rate in the middle. I discovered that increasing the number of epochs enhanced training accuracy but may lead to overfitting if not carefully regulated by validation loss. The optimization technique used also depends on the task; for example, the SGD optimizer performed well for classification problem, but the Adam and RMSprop optimizers performed better for regression. I also discovered that the loss function used was significant and varied depending on the situation. For categorizing handwritten digits in first notebook, MSE loss outperformed cross-entropy loss, whereas in second problem, Gaussian likelihood loss was utilized. The latter anticipated a probability distribution across samples, which might aid in dealing with confusing instances and expressing ambiguity. Yet, when compared to the best model trained with MSE loss, the error skyrocketed.

5. Architecture

The architecture of a model is critical to its performance. Increasing the number of fully connected layers enhanced the model's performance during training in the regression problem, whereas adding more convolutional layers improved the model's performance on the validation set in the multi-classification task. Adding too many layers, on the other hand, might result in slower learning or overfitting. Similarly, increasing the number of neurons enhanced performance up to a point, beyond which it declined owing to overfitting. I found out that the ReLU activation function beat the sigmoid and tanh functions in both tasks because it is more efficient and only stimulates a subset of neurons.