Deep Learning 2020 **(Due: 25 nov 2020)**

Submission Assignment #3

Instructor: Jakub Tomczak *Name:* Mick IJzer, *Netid:* 2552611

The goal of this assignment is to implement a Convolutional Neural Network in PyTorch. In Part 1 the implementation will be discussed, and in Part 2 the results of the CNN will be analysed.

1 Part 1: Implementation

The MNIST dataset is a dataset that contains 70.000 images of handwritten digits. Each image is 28x28 pixels and has one color channel. The MNIST dataset is mainly used for classification tasks. This is not different for this assignment. Since the goal is classification, the output of the neural network needs to be softmaxed. So the probability of a class (i.e. digit), given the input image and the network is given in equation 1.1. The objective function that will be minimized is the negative log likelihood, which can be seen in equation 1.2.

$$p(y|x;w) = softmax(Network_w(x)) \tag{1.1}$$

$$l(w) = -\sum_n log(p(y_n|x_n;w)) \tag{1.2}$$

In an earlier assignment a model was implemented for the same task of classifying the MNIST images. The neural network that was used for that assignment was a Multi-Layer Perceptron with 28x28 inputs, a fully connected hidden layer with 300 nodes, a sigmoid activation, and a final layer with 10 outputs, after which the softmax activation was applied. The total number weights to be trained in such a MLP is $(28*28*300+300)+(300*10+10) = 238.510$ weights, where the brackets indicate the number of weights in a layer. While the accuracy of the network was 96%, the chosen architecture was not optimal.

Therefore, in this assignment, a Convolutional Neural Network will be implemented using PyTorch. The input dimension for the CNN will be 28x28x1, where the 28x28 reflects the pixels in the image and 1 reflects the number of color channels. The first convolutional layer will create 16 channels, with a kernel size of 3x3 and a stride and padding of 1. Then ReLu activation is used, after which a maxpool (2x2) will be applied. The next convolutional layer will reduce the number of channels to 4, with the same kernel size, stride, and padding (i.e. 3x3, 1, 1 respectively). Again ReLu activation and a maxpool layer (2x2) will be applied. The $7*7*4 = 196$ outputs will then be fed through a fully connected linear layer to compute values for each of the 10 classes. Lastly these values will be softmaxed to compute the probability distribution from equation 1.1. The number of weights in this CNN is a lot lower than the weigths from the MLP, namely $(16*1*3*3+16)+(4*16*3*3+4)+(10*196+10) = 2.140$, where the brackets indicate the number of weights in a layer. As stated earlier, the objective function to be minimized will be the Negative Log Likelihood (NLL).

The main point of interest that will be discussed in section 2 is the effect of the learning rate on the learning curves for a given optimizer. The optimizer that will be used to minimize the NLL is the ADAM optimizer. The ADAM optimization algorithm is an extension to the widely used stochastic gradient descent, where the per-parameter learning rate is adaptive. The per-parameter learning rates are computed based on estimates of the first and second moments of the gradients. Only the learning rate will be changed, and the other parameters of ADAM (e.g. betas and epsilon) will be set to their default values.

2 Part 2: Analysis

The (hyper)parameter settings of the implemented CNN are schematically shown in table 1. The architecture of the CNN has been described in section 1. For each learning rate the same steps are taken to evaluate the performance of the models. Namely, the respective models are trained according to the experimental setup in 1 and after each epoch the loss (i.e. NLL) is computed for the training set and the validation set. The process of training is repeated 5 times to combat random effects (e.g. weight initialization and instance order). Several different learning rates were tested (e.g. ranging from $LR : 0.01$ to $LR : 0.0001$), but the higher learning rates

did not consistently show a reduction in loss over time. So in the end three learning rates were chosen to evaluate; $LR : 0.005$, $LR : 0.001$, $LR : 0.0005$

Parameter	**Value(s)**
Training Instances	50.000
Validation Instances	10.000
Test Instances	10.000
Batch Size	264
Epochs	15
Repetitions	5
Optimizer	ADAM
Learning Rate	[0.005, 0.001, 0.0005]

Table 1: Experimental Setup.

The results of the experiment are shown in figure 1. All models have a positive learning curve (i.e. the loss becomes smaller over time) and the learning curves do not show a lot of variation over the repetitions. However, the learning curve of the $LR : 0.005$ model seems to be relatively volatile compared to the other two models. For the $LR : 0.0005$ model the model doesn't seem to have converged yet after 15 epochs, since the loss is still becoming lower. The model with $LR : 0.001$ has the best looking learning curve, since it looks stable and seems to be at its optimal performance after 15 epochs. The accuracy on the validation set is 97%, 96%, and 96% for the $LR : 0.005$, $LR : 0.001$, $LR : 0.0005$ models respectively.

Because of the smoothness of the learning curves the final model was selected with $LR : 0.001$. The model was trained on the complete training and validation set with the setup from table 1, after which the performance was evaluated on the test set. The accuracy on the test set was 97%, and further inspection of the accuracy for each class showed that the model performed similar for all classes. The performance may be similar to that of the MLP network from the previous assignment, but it is remarkable to see the same performance with less than 10% of the number of weights from the MLP.

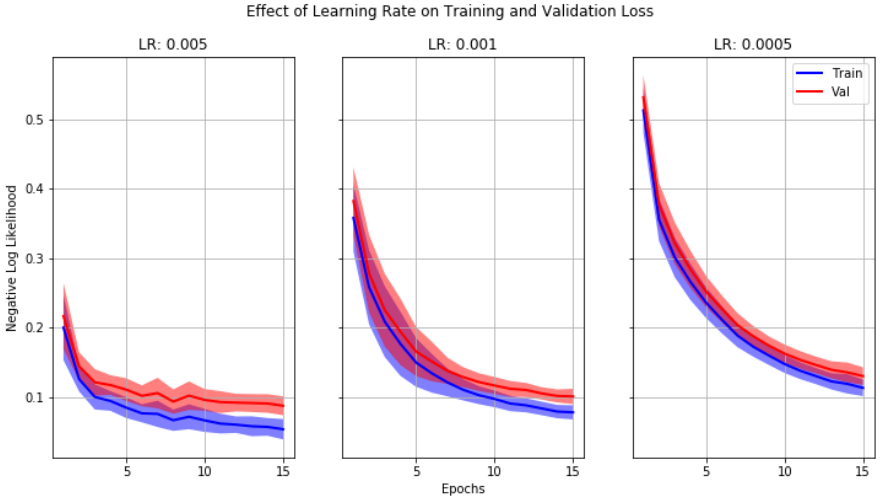

Figure 1: Training and Validation Loss over Time