Deep Learning – 1RT720: Assignment 2

Individual assignment **Due: March 3, 2025**

Plotting learning curves

In this assignment, you will train multiple neural networks. The training process involves the following steps, also known as a training-loop:

1. Utilize *mini-batch gradient descent* to update the network weights. After processing each batch, collect the training cost and accuracy for that batch and append them to two different lists.
2. Upon processing all training batches, you will have completed one *epoch*.
3. Calculate the average accuracy and cost for the batches within the epoch to determine the epoch's training accuracy and cost using your two lists.
4. Check the model performance on the test set, and calculate the test cost and accuracy for the epoch.

When you have processed all epochs you will have generated *four* lists:

- `train_costs` – contains the train cost for each epoch,
- `test_costs` – contains the test cost for each epoch,
- `train_accuracies` – contains the train accuracy for each epoch,
- `test_accuracies` – contains the test accuracy for each epoch.

You should also measure the elapsed time: in seconds, how long did the training take? What batch size and learning rate did you use? When asked about a *learning curves plot*, we would like you to show a plot similar to Figure 1 in your report. You can recycle the code that you used in Assignment 1 for this. Always print (i.e., add it to your plot function) *mini-batch size, learning rate, epochs* and (optionally) *training time* (we suggest to e.g. add a suitable `fig.suptitle()`, following the example in Figure 1).

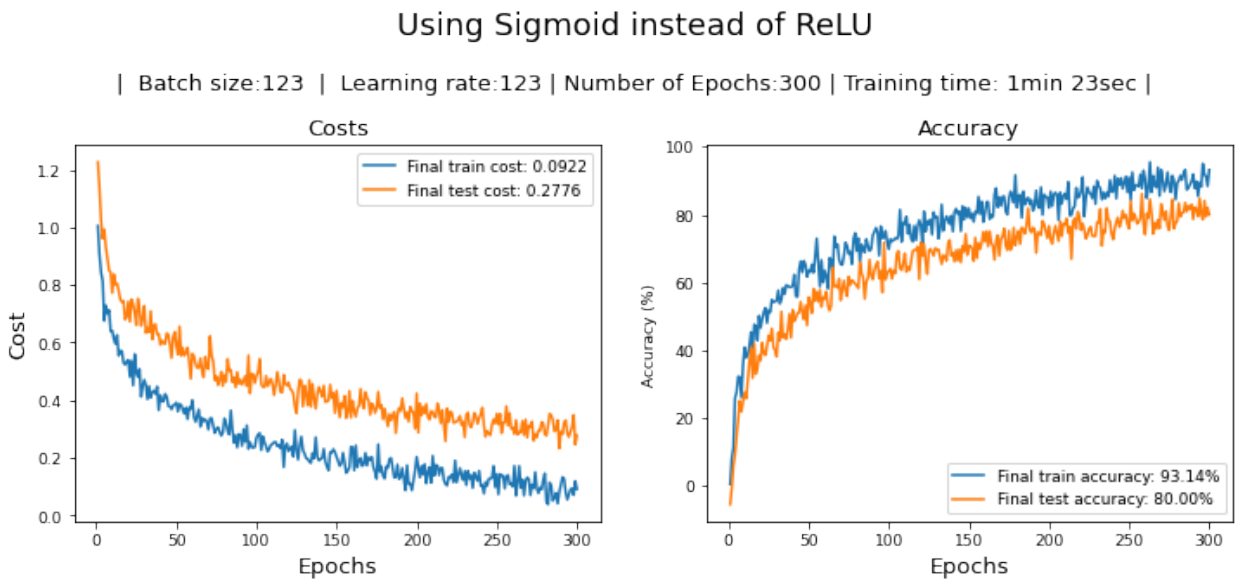

Figure 1: Training curve plots showing cost, accuracy, batch size, learning rate, and training time.

Tips

- **Reuse the code**: You will be training multiple neural networks. Make sure to reuse your code! Write functions and avoid copy-pasting the code as it is a common source of errors and reduces the maintainability of your code.
- **Training time**: Please specify whether you are running your code on a CPU, GPU, or Google Colab. The dataset consists of small-sized images, so the networks should converge within minutes, even when training on a CPU. If training takes significantly longer, it might indicate a bug in your implementation. Utilizing a GPU or running your code on Google Colab can substantially accelerate the training process, if available.
- **Debugging**: If something doesn't work, try to isolate the problem. Comment out code snippets that may contain bugs, include print statements for debugging, and utilize plots to visualize data and identify issues. Simplify the task until it works and then work from there. Work in small iterations.

Classification of handwritten digits using a Convolutional Neural Network

*The most common deep learning approach for **image classification** is the use of Convolutional Neural Networks (CNNs). They reduce the number of needed parameters over fully-connected neural networks, and provide reduced translation dependence which is useful in many applications. In a typical "classic" setup, a CNN architecture will contain a sequence of convolutional layers, each followed by an activation function such as ReLU or sigmoid, interspersed with spatial pooling (most often maxpooling) layers in order to enlarge the field of view. Towards the end of the network, after several convolutional and pooling layers, the high-level image features are classified by fully-connected (also called dense) layers, where all neurons or nodes are connected to all activations of the previous layer. In the final layer, a softmax function maps the non-normalized output of the network to a probability distribution over predicted output classes.*

Classification of MNIST

In this assignment task, you will again perform classification of the MNIST dataset, this time using a convolutional neural network instead of the fully-connected network used in Assignment 1. Since it is a bit cumbersome to implement backpropagation for the convolution operations, we will here leave the do-it-yourself coding and move to the PyTorch deep learning framework.

Exercise 1. Multi-layer fully-connected neural network

Start by implementing **exactly the same** (fully-connected) network as you designed for Assignment 1 (Exercise 1.4), this time using built-in PyTorch functions. Set learning parameters such as to imitate your code from Assignment 1; it is fine to use the default weight initialization scheme of the framework.

(a) Compare the reached performance with your Assignment 1 results; do you observe similar accuracy?

(b) How many times faster/slower was your own implementation? (Make sure to indicate whether you are running on a CPU, Google Colab, or with enabled GPU support if a suitable graphics card is available).

(c) Provide a learning curve plot.

Exercise 2. Multi-layer convolutional neural network

Let us now exchange the fully-connected architecture for a convolution-based one. Construct the network specified below in PyTorch. Obtain the classification output using argmax on the softmax network output. Train the network using standard Stochastic Gradient Descent (SGD) with cross-entropy loss on the MNIST dataset. Choose a suitable mini-batch size and number of training epochs, as well as a learning rate that gives you good convergence without waiting forever. You are expected to reach above 98% accuracy on the test set.

(a) How many learnable weights does this network contain? Compare with how many weights you had in the previous exercise.

(b) Provide a learning curve plot.

Network specification:

1) Convolution	8 times 3x3x1 convolutions with stride 1 and padding 1
2) ReLU	Non-linearity
3) Max Pooling	2x2 max pooling with stride 2
4) Convolution	16 times 3x3x8 convolutions with stride 1 and padding 1
5) ReLU	Non-linearity
6) Max Pooling	2x2 max pooling with stride 2
7) Convolution	32 times 3x3x16 convolutions with stride 1 and padding 1
8) ReLU	Non-linearity
9) Fully Connected	fully connected layer with 10 output neurons
10) Softmax	Softmax layer

Exercise 3. Swap the order of max pooling and activation function

In the previous exercise, we applied the activation function (ReLU, in this case) before performing max pooling. Now, we will swap the order of these two operations by exchanging layers 2 and 3, as well as layers 5 and 6, so that the pooling operation is performed before the ReLU. What do you think will happen? Test your hypothesis and address the following questions:

(a) Does the change in order affect the model's performance, or does it have no impact? With the changed order, how long does the training take, and what is the final accuracy? Please also provide a learning curve plot.

(b) Instead of using ReLU as the transfer function, use the hyperbolic tangent activation function (keeping order still swapped). How long does the training take and what is the final accuracy? Provide a learning curve plot. The difference is probably more distinct here, why?

(c) What are your conclusions with regard to this?

Exercise 4. Use the ADAM optimizer

Now change the optimizer and train the network with ADAM instead of SGD. It is fine to pick the default parameters proposed by PyTorch. (Commonly appearing default values are: GradientDecayFactor (β_1): 0.9000, SquaredGradientDecayFactor (β_2): 0.9990, ϵ: 10^{-8}.) Do you manage to get better results faster than when using the plain SGD optimization? Provide a learning curve plot.

Exercise 5. Residual connections

Implement residual connections like described in Section 11.2.1 in the UDL course book in your architecture. Evaluate if this improves the performance. In particular, check if the inclusion of residual connections allows training of deeper networks by replacing each convolution+activation pair in your architecture with a block of two or three similar pairs, where the residual connection bridges over each such block. Check the training speed with and without the residual connections in place (for otherwise identical architectures). Provide a learning curve plot.

Exercise 6. CNN with three variations

Try three variations based on what you have learned in the course so far; this could be architectural changes, various regularization approaches, change of optimization method, learning-rate scheme, change of activation function, etc. At least one of these changes has to be a regularization approach. How good performance do you manage to reach by tweaking your learning setup?
For each of these variations, answer the following:

- What did did you change and why?
- Do you think the network is overfitting or underfiting?
- Provide learning curve plots for your tests.
- Provide all hyperparameters to the optimizer, loss function, augmentation etc. (including default values of your chosen framework) such that someone else can try to reproduce your work.

(a) First variation

(b) Second variation

(c) Third variation

(d) **Confusion matrix:** For the best performing model of (a)-(c), provide a *confusion matrix* of the classification results on the test data partition.

(e) **Misclassifications:** For this best performing model, show 10 misclassifications that the model made, preferably one from every class (image, including a note about the real class and predicted class).

Information about the MNIST dataset: The MNIST dataset of handwritten digits consists of 60000 training images, and 10000 test images. The dataset has become a classic benchmark dataset for machine learning. The version you have received is prepackaged into subfolders 'Train' and 'Test', and further into subfolders '0'-'9', containing a number of images in the form 0000.png-xxxx.png. Please note that each class does not have exactly the same number of images. All images consist of 28x28 pixels and are stored in the png-file format.

Submission instructions: Submit a pdf with answers to all exercises. Also include your well commented code as appendix in your pdf **and** as a separate zip-file. Submit both files in one submission. Use the 'Assignment template' in Studium for compiling the pdf.

Assessment criteria: This is an **individual assignment**, and you are required to complete all the exercises. To pass, you must correctly answer at least 4 out of Exercises 1–5, as well as Exercise 6.

Checklist

Please check these things before handing in:

- ☐ **No missing tasks**: Make sure you didn't miss any exercises.
- ☐ **State the problem**: Start the answer to each task with a problem formulation, i.e. a short sentence where you write in your own words what you did. For example, "Exercise X: In exercise X we implement ... using ... since ..." instead of just "Exercise X: Answer: ..".
- ☐ **Reasonable accuracy**: The accuracy for the CNN should reach above 98%.
- ☐ **Generative AI**: You have commented if you have used any generative AI, and if so, how.[a]
- ☐ **Code**: Code is both attached as appendix to the pdf and submitted as a separate zip-file.

[a] See on Studium regarding use of generative AI.