Deep Learning – 1RT720: Hand-in Assignment 2

Individual assignment **Due: March 1, 2026**

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_accuracy` – contains the train accuracy for each epoch,
- `test_accuracy` – 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 are encouraged to use the same plotting function that you used in Hand-in Assignment 1 for this. Always print (i.e., add it to your plot function) *mini-batch size, learning rate, epochs* and *training time* (we suggest, e.g., add a suitable `fig.suptitle()`, following the example in Figure 1).

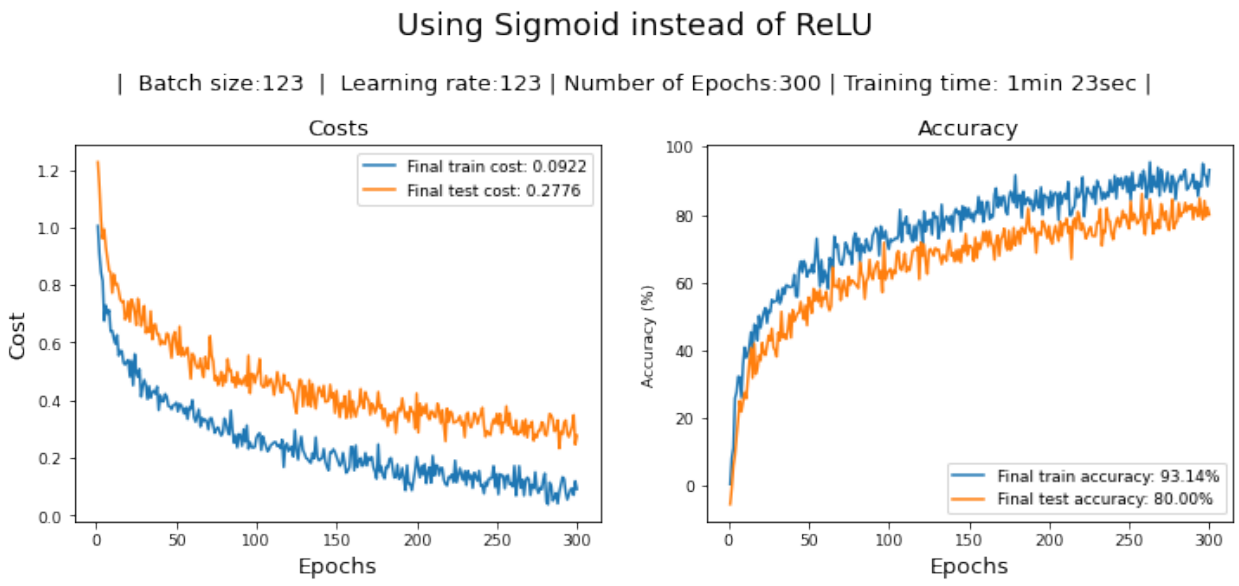

Figure 1: Training curve plots showing cost, accuracy, batch size, learning rate, and training time.

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 Hand-in Assignment 1. Since it is a bit cumbersome to implement backpropagation for the convolution operations, we will here encourage you to complete the PyTorch Tutorial and reuse the code for all exercises below. **Note:** use this plotting function for all learning curve plots.

Exercise 1. Multi-layer fully-connected neural network

---

*Start by implementing **exactly the same** (fully-connected) network as you designed for Hand-in Assignment 1 (Exercise 3 with* `ReLU` *activation), this time using built-in PyTorch functions. Set learning parameters such as to imitate your code from Hand-in Assignment 1.*

(a) Compare the performance and accuracy with your Hand-in Assignment 1 results and report your observations.

(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 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.*

***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) Flatten	*Flatten 2D feature maps into a vector*
10) Fully Connected	*fully connected layer with 10 output neurons*
11) Softmax	*Softmax layer (should be removed if you use* `F.cross_entropy`*)*

(a) Compute the number of learnable parameters layer by layer (including both weights and biases), and report the layer-wise parameters and the total. Compare this with the number of parameters in the previous fully-connected network and comment on the difference.

Hint: For convolutional layers, the number of parameters depends on the kernel size, input channels, and output channels, but not on the spatial resolution of the feature maps.

(b) Provide a learning curve plot.

Exercise 3. 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}, and a learning rate of 0.001 or lower.)

(a) Do you manage to get better results faster than with plain SGD? Share your observations.

(b) Provide a learning curve plot.

Exercise 4. Residual connections

Residual blocks are often the key to the successful training of deeper networks. They consist of a sequence of layers (commonly convolution-activation pairs (i.e., Conv blocks) in CNNs) whose output is added to the block's input via a residual (skip) connection. Please refer to the course book (chapter 11), or the ResNet paper for more details. An illustration of the residual block is provided in Figure 2 (c).

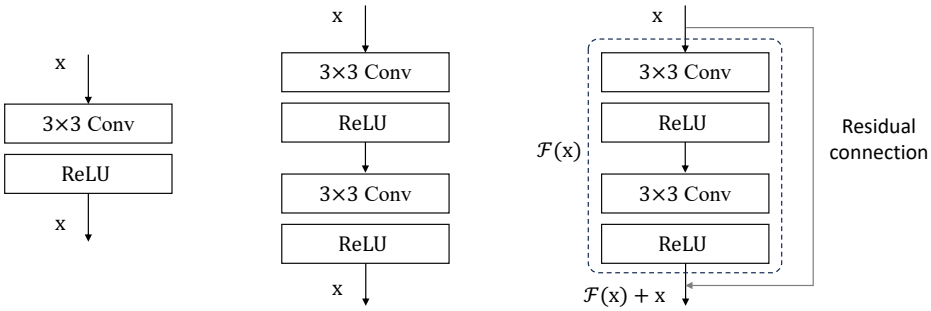

Figure 2: (a) A convolution + activation pair, called "conv block"; (b) Two conv blocks; (c) The residual block. Here, we use $\mathcal{F}(\mathbf{x})$ to represent several (often 2) conv blocks, and use $\mathcal{F}(\mathbf{x}) + \mathbf{x}$ represent the residual connection formulation.

Your task is to implement residual connections in your architecture. Check whether the inclusion of residual connections enables the effective training of deeper networks. In particular,

(a) Compare the training speed and convergence behavior with those of your previous architecture i.e., the plot in Exercise 3, and evaluate whether residual connections improve performance.

(b) Provide a learning curve plot.

Exercise 5. Improve the CNN with Regularization.

Batch Normalization (BN) and Dropout are two widely-used regularization techniques to prevent neural networks from overfitting. Specifically, Batch Normalization stabilizes training by normalizing intermediate hidden neurons using mini-batch statistics, while Dropout improves generalization by randomly deactivating a subset of neurons during training. In this exercise, you are required to incorporate BN and Dropout into your CNN architecture, individually, and answer the following:

(a) Do you think your previous CNN (from Exercise 4) suffers from overfitting or underfitting? Explain why.

(b) Provide learning curve plots for your network with Batch Normalization and with Dropout, respectively.

*(c) Compare the performance of the two variants. For the best-performing model, report a **confusion matrix** of the classification results on the test set. Note: You can use this code to plot the confusion matrix.*

*(d) **Misclassifications:** For the best-performing model, visualize 10 misclassified examples, preferably one from each class. For each image, indicate the ground-truth label and the predicted label.*

Hint: When using Batch Normalization or Dropout, make sure to switch the network to training and evaluation modes appropriately using `net.train()` *and* `net.eval()`.

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 to pass you are required to complete all the exercises, following the instructions provided.

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.