Exercise I

CPSC 452/CPSC 552/AMTH 552/CBB 663 - Spring semester 2025

Published: January 31, 2025
Due: February 14, 2025 - 11:59 PM

The current problem set requires a working installation of PyTorch, torchvision, numpy, matplotlib, sklearn, and scipy.

Please include all of your written answers and figures in a single PDF, titled `<lastname and initals>_assignment1.pdf`, e.g. for a student named Tom Marvolo Riddle, `riddletm_assignment1.pdf`. Upload all relevant files for your solutions to Gradescope, including the PDF report and any Python scripts specified. (For this assignment, the Python scripts include `ps1_functions.py` and `prob5_fcnn.py`.) Your homework should be submitted before Friday, February 14, 2025 at 11:59 PM.

We've provided skeleton code for each major function you'll write in a file called `ps1_functions.py`. Please fill in these functions (preserving their names, arguments, and outputs) and include your completed `ps1_functions.py` in your submission (this is needed by our grading scripts). Any supplemental code you write (e.g. calling these functions to generate plots, or trying out different parameters) can be handled however you choose. A well-structured Jupyter notebook with neatly produced and labelled figures is an excellent way to compile assignment reports; just be sure to submit a PDF *and* the separate `ps1_functions.py` file alongside the notebook. However you produce your report, ensure all your figures are clearly labelled.

We've provided skeleton code for each major function you'll write in a file called `ps1_functions.py`. Please fill in these functions (preserving their names, arguments, and outputs) and include your completed `ps1_functions.py` in your submission (this is needed by our grading scripts). Any supplemental code you write (e.g. calling these functions to generate plots, or trying out different parameters) can be handled however you choose. A well-structured Jupyter notebook with neatly produced and labelled figures is an excellent way to compile assignment reports; just be sure to submit a PDF *and* the separate `ps1_functions.py` file alongside the notebook. However you produce your report, ensure all your figures are clearly labelled.

Programming assignments should use built-in functions in Python and **PyTorch** [1]. In general, you may use the `scipy` stack; however, exercises are designed to emphasize the nuances of machine learning and deep learning algorithms - if a function exists that trivially solves an entire problem, please consult with the TA before using it.

Problem 1

What are the characteristics of a machine learning algorithm and what is meant by "learning" from data?

Problem 2

$$\textbf{Least Squares Solution: } \mathbf{w}^* = (X^\top X)^{-1} X^\top \mathbf{y}.$$

$$\textbf{L2 Norm: } \|x\|_2 = \left(\sum_{i=1}^{n} |x_i|^2 \right)^{\frac{1}{2}}$$

1. Write code in Python that randomly generates N points sampled uniformly in the interval $x \in [-1, 3]$. Output the function $y = x^2 - 3x + 1$ for each of the points generated and add zero-mean Gaussian noise with standard deviation σ to y. Make plots of x and y with $N \in \{15, 100\}$ and $\sigma \in \{0, 0.05, 0.2\}$ (there should be six plots in total). Save the point sets for following questions.
   Hint: You may want to check the NumPy library for generating noise.

2. Find the optimal weights (in terms of MSE) for fitting a polynomial function to the data in all 6 cases generated above using a polynomial of degree 1, 2, and 9. Use the least squares analytical solution given above. Do not use built-in methods for regression. Plot the fitted curves on the same plot as the data points (you can plot all 3 polynomial curves on the same plot). Report the fitted weights and the MSE in tables. Qualitatively assess the fit of the curves. Does it look like any of the models overfit, underfit, or appropriately fit the data? Explain your reasoning in one or two sentences.

3. Apply L2 norm regularization with a 9-degree polynomial model to the cases with $\sigma = 0.05$ and $N \in \{15, 100\}$. Vary the parameter λ, and choose three values of λ that result in the following scenarios: underfitting, overfitting, and an appropriate fit. Report the fitted weights and the MSE in each of these scenarios.

   Hint: The least squares solution can also be used for polynomial regression. Check slides of lecture 2 for details on L2 norm regularization.

Problem 3

1. Load the dataset from file `prob3_data_seed.dat` and normalize the features using min-max scaling so that each feature has the same range of values. (The `prob3_data_description.txt` for the description of the provided dataset.)

2. Write a program that applies a k-nn classifier to the data with $k \in \{1, 5, 10, 15\}$. Calculate the test error using both leave-one-out validation and 5-fold cross validation. Plot the test error as a function of k. You may use the existing methods in `scikit-learn` or other libraries for finding the k-nearest neighbors, but do not use any built-in k-nn classifiers. Any reasonable handling of ties in finding

k-nearest neighbors is okay. Also, do not use any existing libraries or methods for cross validation. Do any values of k result in underfitting or overfitting?

3. Apply two other classifiers of your choice to the same data. For these additional classifiers, you may use existing libraries, such as `scikit-learn` classifiers, but for cross-validation, you should reuse your method from 3.2 or modify it slightly. Possible algorithms include (but are not limited to) logistic regression, QDA, naive Bayes, SVM, and decision trees. Use 5-fold cross validation to calculate the test error. Report the training and test errors. If any tuning parameters need to be selected, use cross-validation and report the training and test error for several values of the tuning parameters. Which of the classifiers performed best? Did any of them underfit or overfit the data? How do they compare to the k-nn classifiers in terms of performance?

Problem 4

1. Suppose we take all the weights and biases in a network of perceptrons, and multiply them by a positive constant, c > 0. Show that the behavior of the network doesn't change. (Exercise in Ch1 Nielsen book)

2. Given the same setup of problem 4.1 - a network of perceptrons - suppose that the overall input to the network of perceptrons has been chosen and fixed. Suppose the weights and biases are such that $w \cdot x + b \neq 0$ for the input x to any particular perceptron in the network. Now replace all the perceptrons in the network by sigmoid neurons, and multiply the weights and biases by a positive constant $c > 0$. Show that in the limit as $c \to \infty$ the behavior of this network of sigmoid neurons is exactly the same as the network of perceptrons. How can this fail when $w \cdot x + b = 0$ for one of the perceptrons? (Exercise in Ch1 Nielsen book)

Use the following figure for problem 4.3 and 4.4.

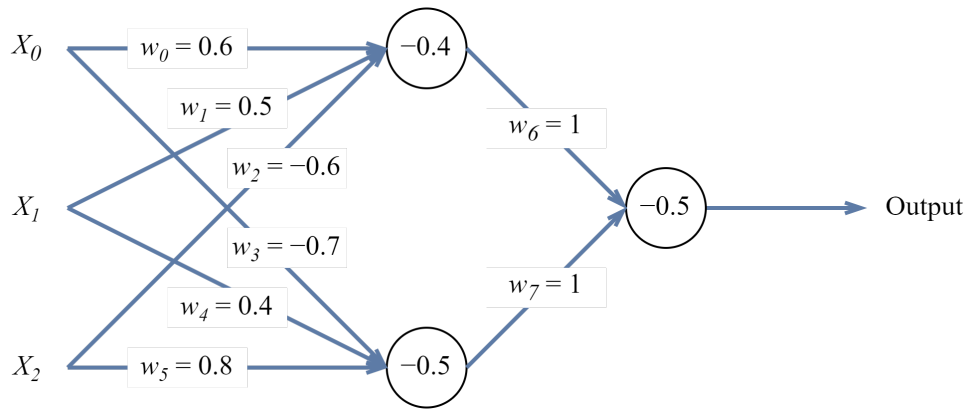

Figure 1: Multilayer perceptron with three inputs and one hidden layer.

3. For each possible input of the MLP in Figure 1, calculate the output, i.e. what is the output if $X = [0, 0, 0]$, $X = [0, 0, 1]$, etc. You should have 8 cases total.

4. If we change the perceptrons in Figure 1 to sigmoid neurons, what are the outputs for the same inputs (e.g., inputs of $[0, 0, 0], [0, 0, 1], \cdots$)?

5. Using perceptrons with appropriate weights and biases, design an adder that does two-bit binary addition. That is, the adder takes as input two two-bit binary numbers (i.e. 4 binary inputs) and adds them together. Don't forget to include the carry bit. The resulting output should be the two-bit sum and the carry bit for a total of three binary outputs.

Problem 5

Time to implement your first convolutional neural network (CNN) in PyTorch! For this assignment, we'll be training the network on the canonical MNIST dataset. After building the network, we'll experiment with an array of hyperparameters, tweaking the network's width, depth, learning rate and more in pursuit of the highest classification accuracy we can muster.

You may find the PyTorch tutorials helpful as you complete this problem: `https://pytorch.org/tutorials/beginner/basics/intro.html`. If you haven't yet, we suggest you go through them. Pay more attention to the tutorial on the optimization loop, which you will need to build more or less from scratch.

We have provided a colab notebook `prob5_cnn.ipynb` (`https://colab.research.google.com/drive/18N2_Sxo1Jeu9WbTA_XtCVUI4mG6INLYn?usp=sharing`) to implement a neural network with PyTorch based on MNIST data. Within the provided file is a basic scaffold of a model training workflow. You may use the existing functions in the script for all parts of this problem.

Let's finish the tasks step by step.

1. Follow the TODOs in `prob5_cnn.ipynb` to get yourself familiar with the basic PyTorch operations. You have to pass all the assertions provided as the test cases.

2. Follow the TODOs in `prob5_cnn.ipynb` and step 1, build and train a `SimpleCNN` on MNIST dataset

   - Implement the first `SimpleCNN` with linear layers only. You will use this as a baseline model for the next step.
   - Implement the training function for your CNN. The function should take the model, optimizer, loss function, training data loader, and validation data loader as input. It should return the training and validation loss and accuracy after each epoch.
   - Implement the `plot_metrics` function to visualize the training history.
   - Use the training function above to train your `SimpleCNN` on the MNIST dataset. Fill in the missing parameters for training as well.

   Report the results that you got for the experiment. Specifically:

   - Create a plot of training and test loss vs epoch (2 lines, 1 plot)
   - Create a plot of training and test accuracy vs epoch (2 lines, 1 plot)

3. As you can see in the previous step, the training accuracy of the `SimpleCNN` is poor. We will now improve its performance with nonlinear layers.

   - Modify the `SimpleCNN` by adding `nn.MaxPool2d`, `nn.Dropout`, and activation functions.
   - Use the training function and the same hyperparameters above to train your `CNN` on the MNIST dataset.

   Report the performance comparison (i.e. training and testing error) of `SimpleCNN` and `CNN` in one table.

4. Provide a single table of the final training and testing errors for all experiments listed below.

   - Try adjusting the learning rate to improve its accuracy. You might also try increasing the number of epochs used. Record your results in a table.
   - Try training your network with different non-linearities between the layers (i.e. relu, softplus, elu, tanh). You should experiment with these and record your test results for each in a table
   - Try changing the width of the hidden layer, keeping the activation function that performs best. Remember to add these results to your table.
   - Experiment with the optimizer of your network (i.e. SGD, Adam, RMSProp). You should experiment with these and record your test results for each in a table

   Which learning rate/activation function/optimizer converges fastest? Does the width of the hidden layer affect the model accuracy? Which model achieves the highest test accuracy? Provide some rationale as to the observed differences.

5. Let's now add regularization to the previous network. We have a set of old and new parameters you can try to improve the performance of your model:

   - Optimizer (SGD, Adam, RMSProp, etc): Different optimizers may lead to different convergence speeds and performance.
   - Weight decay (L2 penalty): Weight decay is a regularization technique to prevent overfitting. It is recommended to use a small weight decay value (e.g., 1e-4).
   - Activation function (ReLU, Leaky ReLU, Tanh, etc): Different activation functions may lead to different convergence speeds and performance.
   - Dropout rate: Dropout is a regularization technique to prevent overfitting. It is recommended to use a small dropout rate (e.g., 0.2).
   - ...

   Please implement a grid search algorithm to find the best set of hyperparameters and report the best validation accuracy you can get. You should also include the plot for loss and accuracy comparison. Remember, any hyperparameter can be tuned!

6. With your best-performing model, plot a confusion matrix showing which digits were misclassified, and what they were misclassified as. What numbers are frequently confused with one another by your model? (You may use `sklearn`'s confusion matrix function to generate the matrix.)

7. What was the highest percentage of classification accuracy your convolutional neural network achieved? Briefly describe the architecture and training process that produced it. (If you like, you can take part in our friendly class competition by posting your results, along with a short description of your methods, to `https://edstem.org/us/courses/53810/discussion/4185060`.)

References

[1] "Pytorch." [Online]. Available: https://pytorch.org/