Exercise 1

CPSC/AMTH/CBB 663 - Spring semester 2021

Published: Monday, February 8, 2021
Due: Sunday, February 21, 2021 - 4:00 PM

Please put all relevant files & solutions into a folder titled `<lastname and initials>_assignment1` and then zip that folder into a single zip file titled `<lastname and initials>_assignment1.zip`, e.g. for a student named Tom Marvolo Riddle, `riddletm_assignment1.zip`. Include a single PDF titled `<lastname and initals>_assignment1.pdf` and any code specified. Requested plots should be sufficiently labeled for full points.

Any formatting that allows the TAs to quickly determine which part of the problem the code is used for is fine. This means a Jupyter notebook with headings is allowed, as long as both the *.ipynb file and the PDF from Jupyter are submitted. Your homework should be submitted to Canvas before Sunday, February 21, 2021 at 4:00 PM.

Programming assignments should use built-in functions in Python and PyTorch; In general, you may use the `scipy` stack [1]; 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

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

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

1. Write code in Python that randomly generates N points sampled uniformly in the interval $x \in [-1, 3]$. Then output the function $y = x^2 - 3x + 1$ for each of the points generated. Then write code that adds 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 to two sentences (no calculations necessary).

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 assignment1.zip and normalize the features using min-max scaling so that each feature has the same range of values.

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 $wx+b \neq 0$ 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 $wx + b = 0$ for one of the perceptrons? (Exercise in Ch1 Nielsen book)

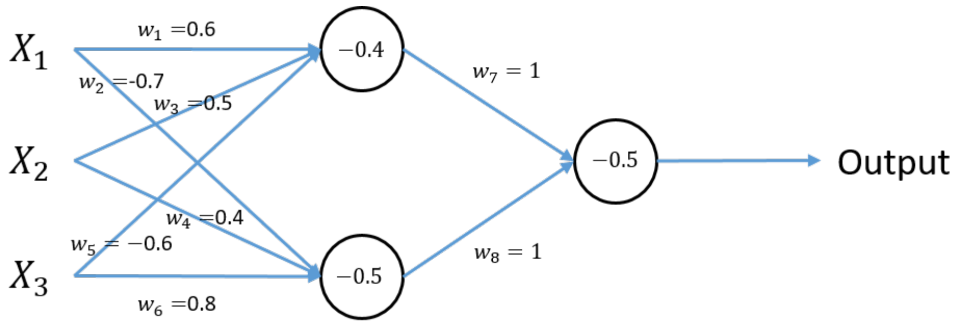

Figure 1: Multilayer perceptron with three inputs and one hidden layer.

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], ...)?

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

1. Run the Python code given in Chapter 1 of the Nielsen book in the section titled "Implementing our network to classify digits". You can find a link to the code at the beginning of the section. Verify that you understand each line of the code and that you obtain the same results as given in the book.

2. We will now explore feedforward networks using PyTorch. You may find this tutorial helpful: https://nextjournal.com/gkoehler/pytorch-mnist. A few additional points:

- Create one hidden layer (with 128 units) between the input and output by creating another weight and bias variable.
- Try training this without a non-linearity between the layers (linear activation), and then try adding a sigmoid non-linearity both before the hidden layer and after the hidden layer, recording your test accuracy results for each in a table.
- Try adjusting the learning rate (by making it smaller) if your model is not converging/improving in accuracy. You might also try increasing the number of epochs used.
- Experiment with the non-linearity used before the middle layer. Here are some activation functions to choose from: relu, softplus, elu, tanh.
- Experiment with the learning rate, optimizer and activation function of your network.
- Lastly, experiment with the width of the hidden layer, keeping the activation function that performs best. Remember to add these results to your table.

3. What percentage classification accuracy does your feedforward network achieve?
4. Create a plot of the training and test error vs the number of iterations. How many iterations are sufficient to reach good performance?
5. Print the confusion matrix showing which digits were misclassified, and what they were misclassified as. What numbers are frequently confused with one another by your model?
6. Report the best accuracy and briefly describe the training scheme that reached this accuracy.

References

[1] "The scipy stack specification¶." [Online]. Available: https://www.scipy.org/stackspec.html