ECE 6545 – Deep Learning for Image Analysis – Spring 2025

Assignment 1 (100 points)

You are expected to complete this assignment by answering the questions in this file and filling in the designated places in the python scripts (files ending with .py). There is a total of three .py scripts, one for each exercise. For instance, the first exercise is named as assignment1_ex1.py. These files already contain part of the code for carrying out the tasks in this assignment in a structured manner. However, there are some places where we left for you to fill. These are marked by the comments, "*#your code starts here*" and "*#your code ends here*". We believe that this approach focuses your learning on the actual topics of the course while easing the burden of python programming experience to some extent. There is an additional utils.py file which contains some helper code accessed from the exercise files. You should not modify utils.py.

In this assignment, you will encounter three kinds of questions:

- Coding questions (marked with C) - this type of question requires you to provide an implementation of a certain task in the .py scripts for the exercises.
- Theoretical questions (marked with T) - this type of question requires you to provide an answer with text or mathematical equations.
- Analysis questions (marked with A) - this type of question requires an analysis of results that may include text, code and visualizations. The first questions of this type will be more explicit of what you should write. They will become more open-ended as the assignment goes on.

For assignment submission, you will submit a report file (.pdf) through Canvas with answers to the theoretical and analytical questions plus the outputs/results of the codes such as plots, as well as the completed python codes (.py). In your report, make sure you use sections with same numbering scheme as the questions. For instance, Exercise 1.1, Exercise 1.2, etc. Exercises which do not produce any text or visual output do not need to be included in the report.

Your submitted files need to follow these guidelines:

- For the codes, no other packages than the ones already imported can be used.
- No other data than the ones provided should be used.
- The results present in your delivered report should be reproducible.
- For the required plots, you should save them during the execution of the codes and import them in your report into the appropriate section.
- The designated places in the codes that require your input with will be marked with comments `##your code starts here` and `##your code ends here` to specify where you need to write code.
- Your final delivery should **NOT** contain any additional modifications outside of the demarcations defined above. However, if needed for testing, please feel free to modify them and remember to reverse to its original state before you submit them.

- All code must be your own work. Code cannot be copied from external sources or other students. You may copy and reuse code from the provided .py files if you think it is useful for use in another question.
- All images must be generated from data generated in your code. Do **NOT** import/display images that are generated outside your code.
- Your analysis must be your own, but if you quote text or equations from another source please make sure to cite the appropriate references.
- If you submit a file before the due date and another file after the due date, the late policy will be applied, since it will be the final submission that is taken into account for grading. You can review the late assignment policy in the course syllabus. If Canvas classifies a delivery as late, the assignment will be considered late.4

Other notes:

- The required datasets are included in the assignment package and you don't need to download and preprocess them. Just make sure that all your codes (assignment codes, utils.py, and data files (ending in "-ubyte" or "-ubyte.gz") in the same folder.
- Read all the provided codes and their comments, as they contain variables and information that you may need to use to complete the designated places.
- In general, you will need to do the exercise in sequential order because some exercises import and reuse your completed code from previous exercises.
- For editing the codes files, you can use any code or plain text editor (e.g., notepad, vs code's editor, TextEdit, BBEdit).
- Generally, for testing and running the codes, you will need python (either alone or with a package manager like Anaconda). However, the CHPC servers usually have it installed. A slurm file for exercise 1 is included in the assignment package as an example.
- A python/numpy/matplotlib tutorial that you might find useful: http://cs231n.github.io/python-numpy-tutorial/

Exercise 1 - Analyzing model capacity with a polynomial toy example (Total of 38 points)

This exercise will illustrate how validation error of a model evolves by changing model capacity. We are going to start with a simple example that follows a third-degree polynomial.

Exercise 1.1(C) (3 points)

In this section, you will implement a function, named `third_degree_polynomial`, that returns the output of a third-degree polynomial. This function receives two numpy arrays, `x` and `coeffs`. If `coeffs`=$[a_0, a_1, a_2, a_3]$, the function should return an output array `y` in which $y_i = a_3 \times x_i^3 + a_2 \times x_i^2 + a_1 \times x_i + a_0$ for i^{th} sample. Note: the output array, `y`, should have a shape of [N, 1] where N is the number of samples in `x`. The exercise will check if your function works as expected and will display a result accordingly.

Next, we are going to set up data for Exercise 1.2-1.6 and 2. Please pay close attention to the names of the variables because you are going to need to use them in your code.

Exercise 1.2(C) (6 points)

To fit polynomial functions to the data that we just generated, your task is to implement a polynomial least square fitting function, named `fit_func`, that receives 3 inputs (input array, target array, and the degree of the polynomial fitting function) and returns the coeffficients array (of shape [degree+1, 1]) of the polynomial fitting function. Moreover, a polynomial fitting function can be expressed in a closed-form solution. Hence, your `fit_func` needs to be implemented with the closed-form solution.

Exercise 1.3(C) (5 points)

In exercise 1.1 above, we implement a third degree polynominal function. Now, we will generalize it to express the polynomial function of any degrees. In this section, you need to implement a function, named `any_degree_polynomial`, that receives two numpy arrays, `x` and `coeffs`, and computes the polynomial function value for each sample in `x`. For example, if `coeffs`=$[a_0, a_1, \ldots, a_n]$, the function should return the value of i^{th} sample as $y_i = a_n \times x_i^n + a_{n-1} \times x_i^{n-1} + \ldots + a_1 \times x_i + a_0$. Note: the output array, `y`, should have shape of [N, 1] where N is the number of samples in `x`.

Exercise 1.4(C) (5 points)

Now, you will use the functions above to fit a third degree polynomial to the provided data (`x_ex1_train,y_ex1_train`), and plot your fitted results and the training data on the same graph. Please remember to use legend to identify what each curve is. Include the plot in your report. Remember that the plot should be saved and then included in your report.

In the next two exercises, we are going to evaluate how polynomials of different degrees perform in the validation data when fitted to the training data.

Exercise 1.5(C) (3 points)

First, we are going to define a function that returns the metric used to evaluate the results. For this exercise, we use the mean squared error function, which is defined as $\frac{1}{N}\Sigma(\hat{y} - y)^2$. Your task is to implement a function, named `mse`, that takes two numpy arrays (`predicted_values` and `targets`) as input and returns the mean squared error between them.

Exercise 1.6(T) (5 points)

Maximum likelihood (ML) principle is a common way to derive a good estimator for different models. For example, we can use ML to. derive the estimation of the coefficients of polynomial fitting functions. If using ML, what assumptions do we need to make in order to obtain the same estimated coefficients as using the mean squared error criterion in Exercise 1.5 above?

Exercise 1.7(A) (11 points)

Now, we are going to treat the degree of the fitting polynomial as a hyperparameter. In this section, you will: 1) Fit the training data, `x_ex1_train`, and validation data, `x_ex1_val`, using polynomials of degree 1 to 10. Then, plot the mean squared error as a function of polynomial degrees for both training data and validation data on the same graph.

2) Write a short analysis of the results presented in your plot, stating which degrees are overfitting and which are underfitting, and why.

Now, give your analysis.

Exercise 2 - Defining and training fully connected networks (Total of 32 points)

In this exercise, we are going to define functions to train a fully connected network with one hidden layer. First, we set a function to initialize the learnable parameters of the network. They are going to be arrays stored in a python dictionary in which the keys of the dictionary represent the name of the parameters of the network. The parameters are called '`weights_i`' and '`bias_i`' where i is the layer where the parameter is used. For the linear layer forward pass, we use the equation/notation $XW + b$, X being a matrix with batch size as first dimension.

Exercise 2.1(T) (4 points)

Calculate the number of parameters of a network initialized with `initialize_parameters_ex2` would have. Your answer needs to be expressed as a function of `n_inputs, n_hidden_nodes` and `n_outputs`.

Exercise 2.2(C) (8 points)

Complete the function, `two_layer_network_forward` to define a forward pass of a two-layer fully connected network with ReLU as the activation function of the first layer.

Exercise 2.3(C) (11 points)

Now, you will implement two functions `mse_loss_backward` and `two_layer_network_backward` for the derivatives of the cost function and parameters of the network that you derive in Exercise 2.2.

Exercise 2.4(C) (9 points)

In this section, you are going to implement the update rule for a batch of training. Complete the function, `run_batch_sgd`, that calculates the gradients and then updates the parameters using the vanilla stochastic gradient descent update rule. Then, you will use this function for the training process defined later in the code. Finally plot the output of your neural network and the ground truth on the y-axis vs. the input value on the x-axis. Use different colors and figure legends to distinguish between the ground truth and neural network output.

Exercise 3 - MNIST and weight decay (Total of 30 points)

In this exercise, we are going to extend the two-layer neural network to another dataset. We are going to use a flattened and reduced MNIST dataset to train the network using L2 regularization. First, we need to load and preprocess the data:

Exercise 3.1(C) (7 points)

The $L2$ penalty is defined as the sum of squares of every element of the penalized parameters. In this section, you will implement the gradients of different parameters in the network. We consider the $L2$ penalty over all the weights parameters, and no penalty over the bias parameters.

Exercise 3.2(A) (18 points)

In this section, you will analyze how the weight decay, λ, of the $L2$ regularization changes the final results for the MNIST dataset using a two-layer neural network with 200 neurons in the hidden layer.

Notes:

- You should play a bit with the learning rate, batch size and number of epochs.
- You should be able to get more than 90% accuracy on the validation set

Now, give your analysis.

Exercise 3.3(C) (5 points)

Finally, test your best model using the provided test set.