Assignment 1: PyTorch Intro, Data Analysis & NN Models

Checkpoint: 02/22/2026 (Sun) 11:59pm
Due date: 03/01/2026 (Sun) 11:59pm
Total Points: 100 points + 10 bonus points

Our initial assignment is focused on reviewing PyTorch, data analysis techniques, basic machine learning methods and building shallow neural networks. This assignment also focuses on theoretical and practical skills relating to neural networks. It consists of three parts where you derive and analyze the setup, practice dealing with various datasets and implement, train, and adjust neural networks. It is expected that all coding parts of the assignment will be completed using PyTorch. If you have not worked with PyTorch before you can complete the official PyTorch tutorial at the below link:
https://pytorch.org/tutorials/beginner/introyt/introyt_index.html

Note that this is a group assignment. You can do it in a group of maximum 3 students.

Note: You can partially reuse related implementations that you have completed for other courses with a proper citation, e.g. "Part I is based on the CSE 574 Machine Learning Assignment 1 submission by My Full Name and My Teammate Full Name."

Part I: Data Analysis & NN Models [40 points]

Step 1: Data Analysis & Pre-processing [20 points]

1. Select a real-world dataset from the source listed below. Requirements for the dataset:

   - Represent real-world data
   - Contain at least 20k entries
   - Be different from datasets used previously for this or other courses

   The dataset should come from one of the following resources:

   - Open Data Buffalo: https://data.buffalony.gov/
   - US Government's Data: https://www.data.gov/
   - Yahoo Finance: https://finance.yahoo.com/

- Yahoo Webscope: https://webscope.sandbox.yahoo.com/

2. Read, preprocess, and print the main statistics about the dataset.
3. *[If applicable]* Handle missing entries. Possible solutions:
   - Drop rows with missing entries. If you have a large dataset and only a few missing features, it may be acceptable to drop the rows containing missing values.
   - Impute missing data. Replace the missing entries with the mean/median/mode of the feature. You can use the k-nearest neighbor algorithm to find the matching sample.
4. *[If applicable]* Handle mismatched string formats.
5. *[If applicable]* Handle outliers. Detect and manage outliers within the dataset. Possible solutions:
   - Remove outliers. If there are just a few outliers, you may eliminate the rows containing these outliers.
   - Impute outliers. Replace the outliers with the mean/median/mode of the feature.
6. Using any data visualization library (e.g. matplotlib, seaborn, plotly), provide at least 3 visualization graphs related to your dataset. You can utilize any columns or a combination of columns in your dataset to generate graphs. E.g. correlation matrix, features vs. the target, counts of categorical features vs. the target.
7. Identify uncorrelated or unrelated features.

   To compute the correlation matrix, load your dataset using Pandas and use the pandas.DataFrame.corr method. You can refer to the documentation here:

   https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.corr.html

   You can drop some features based on their correlation with the target.
8. Convert features with string datatype to categorical. Possible ways:
   - One-hot encoding, creating binary columns for each category, denoting their presence or absence.
   - Label encoding assigns unique integers to distinct feature values, useful for ordinal relationships among categories. E.g., "Small" as 0, "Medium" as 1, and "Large" as 2 can represent a "Size" feature. However, it may introduce unintended patterns.
9. *[If applicable]* Normalize non-categorical features.

   a. Find the min and max values for each column.

   b. Rescale dataset columns to the range from 0 to 1

Why do we do this? Normalization is to transform features to be on a similar scale. This improves the performance and training stability of the model.

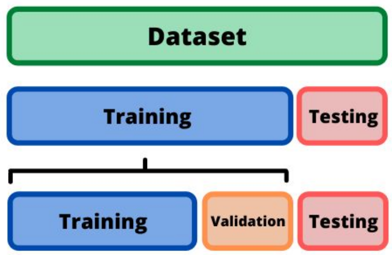
Note: normalize() is not allowed as it is a part of *scikit-learn* library.

10. Choose your target Y and features X.

11. Split the dataset into training, testing and validation sets.

    You can use `train_test_split` from *scikit-learn*

    Hint: first you can split the dataset into 'training' and 'testing' batches. Then take the 'training' batch and split it again for 'training' and 'validation'

    Why do we need to split into training, testing and validation?

    - Training set: used to train the model to learn the patterns or features in the data. It is important to have a large and representative training set so that the model can learn well.
    - Validation set: used to tune the model hyperparameters and to prevent overfitting. Overfitting is when the model learns the training data too well and cannot generalize to new data. Hyperparameters are parameters that are not learned from the data, but are set by the user. By tuning the hyperparameters, we can improve the model's performance on the validation set.
    - Test set: used to evaluate the final model's performance on unseen data. This gives us a good idea of how well the model will perform in the real world.

    The commonly used splits of 70:15:15 or 80:10:10 are good starting points, but the optimal split ratio will vary depending on the size and characteristics of the dataset.

12. Print the shapes of X_{train}, y_{train}, X_{test}, y_{test}, $X_{validation}$, and $y_{validation}$.

Step 2: NN Model [20 points]

1. Apply NN model to solve this task. The accuracy for NN model submitted should be above 75%.

2. Train the neural network. Run the training loop and train the neural network on the training data. Select the number of epochs and batch size. Monitor the training loss and the validation loss at each epoch to ensure that the model is not overfitting. Estimate the time it takes to train the model, e.g. using time.time().

3. Save the weights of the trained neural network that returns the best results.

   Saving and loading models in PyTorch

4. Provide the results of the NN model:

   a. Provide your loss value and accuracy of the NN model.

   b. Include plots that compare the predictions versus the actual test data for the NN model and all the ML models used. Consider metrics such as accuracy, time, and loss to compare the methods.

In your report for Part I:

1. Provide brief details about the nature of your dataset. What is it about? What type of data are we encountering? Provide the main statistics about the entries of the dataset (mean, std, number of missing values, etc.)

2. What kind of preprocessing techniques have you applied to this dataset?

3. Provide at least 3 visualization graphs with a brief description for each graph, e.g. discuss if there are any interesting patterns or correlations.

4. Provide brief details of the NN model you have used.

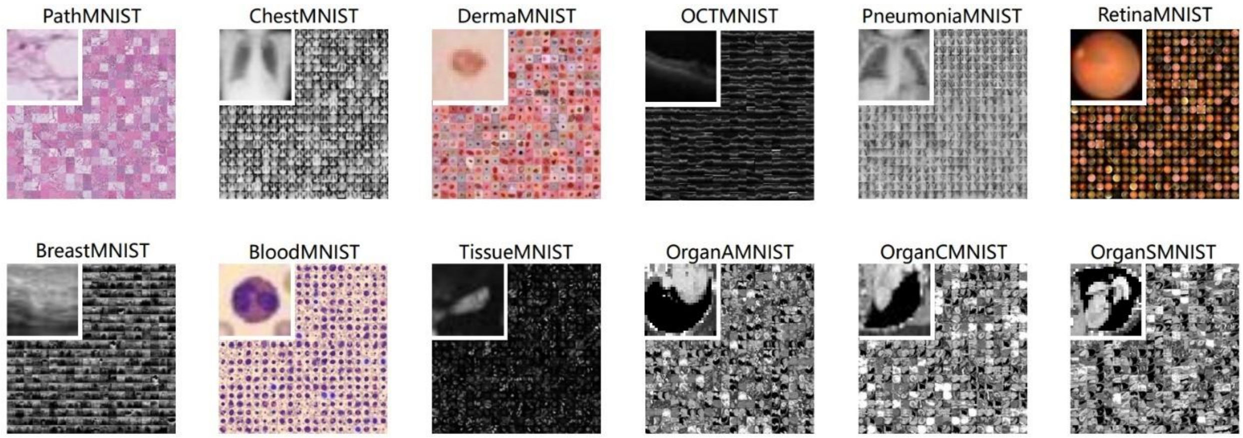
5. Provide your loss value and accuracy for your NN.

6. Show the plot comparing the predictions vs the actual test data. Analyze the results. You can consider accuracy/time/loss as some of the metrics for analyze.

Part II: OCTMNIST Classification [40 points]

For this part, we will be working with a real-world dataset – OCTMNIST.

OCTMNIST



The OCTMNIST is based on a prior dataset of 109,309 valid optical coherence tomography (OCT) images for retinal diseases. Each example is a 28×28 image, associated with a label from 4 classes. Getting the data:

- MedMNIST is a collection of multiple datasets, for this assignment we will be working with one dataset from the collection – OCTMNIST
- pip install medmnist

References:

- https://medmnist.com/
- https://github.com/MedMNIST/MedMNIST
- Direct download - https://zenodo.org/record/6496656

Step 1: Loading the dataset and preparing for training

1. Load OCTMNIST 2D dataset
2. Analyze the dataset, e.g., return the main statistics
3. Provide at least 3 visualization graphs with a short description
4. Preprocess the dataset. E.g. normalizing the pixel values to a standardized range, typically between 0 and 1.
   a. *[If needed]* Address class imbalance in the target column. Possible solutions: oversampling; undersampling; data augmentation techniques for the minority class; assign higher weights to the minority class and lower weights to the majority class, etc.

b. *[If needed]* Convert categorical variables to numerical variables using one-hot encoding. You can use OneHotEncoder from scikit-learn

5. Split the dataset into training, testing and validation sets.
   You can use `train_test_split` from scikit-learn

   Note: you are welcome to reuse your code for data preprocessing from Part I of this assignment with a proper citation, e.g. "Code is based on Part I of this assignment"

Step 2: Defining the Neural Network

1. Decide your NN architecture. Design a model architecture that consists of at least three layers, including convolutional layers and fully connected layers. You can build an FC NN or CNN model.

   - How many input neurons are there?
   - How many output neurons are there?
   - What activation function is used for the hidden layers?
     - Suggestion: try ReLU, ELU, Sigmoid, click here for the full list
   - What activation function is used for the output layer?
   - What is the number of hidden layers?
     - Suggestion: start with a small network, e.g., 2 or 3 hidden layers
   - What is the size of each hidden layer?
     - Suggestion: try 64 or 128 nodes for each layer
   - Do you include Dropout? (details)

2. Define your NN architecture using PyTorch (basic building blocks in PyTorch).

3. Return the summary of your model. You can use Torchinfo package

Step 3: Training the Neural Network

1. Define a loss function (loss_function)that will be used to compute the error between the predicted output and the true labels of the training data. List of loss functions (PyTorch). For binary classification problems, a commonly used loss function is Binary Cross Entropy Loss (Details).

2. Choose an optimizer and a learning rate. It will update the weights of the NN during training. SGD is one of the commonly used, you can also explore other optimizers like Adam or RMSProp. Check a list of optimizers (PyTorch)

3. Set up the training loop:

a. Create a loop that iterates over the training data for a specified number of epochs.

b. Iterate over the training data in batches.

c. Forward pass: pass the input data through the neural network.

```
outputs = model(inputs)
```

d. Compute loss: calculate the loss using the defined loss function.

```
loss = loss_function(outputs, labels)
```

e. Backpropagation: zero the gradients, compute gradients, and perform backpropagation.

```
optimizer.zero_grad()    #Clear gradients
loss.backward()          #Backpropagation
```

f. Update the model's weights using the optimizer.

```
optimizer.step()       #Update weights
```

g. Validation phase: set the model to evaluation mode.

```
model.eval()           #Set to evaluation mode
```

During the validation phase, there will be no backward propagation to calculate the gradients and no optimizer step to update the steps.

```
with torch.no_grad():      # Disable gradients
     for val_inputs, val_labels in val_loader:
           val_outputs = model(val_inputs)
           val_loss = loss_function(val_outputs, val_labels)
```

4. Train the neural network. Run the training loop and train the neural network on the training data. Select the number of epochs and batch size. Monitor the training loss and the validation loss at each epoch to ensure that the model is not overfitting. Estimate the time it takes to train the model, e.g. using time.time().

5. Save the weights of the trained neural network that returns the best results.
   Saving and loading models in PyTorch

6. Evaluate the performance of the model on the testing data. Suggested metrics:

- Accuracy (PyTorch functions)
- Precision, recall and F1 score (more details).
  You can use `sklearn.metrics.precision_recall_fscore_support`
- In case you need to convert the output of sigmoid() to 0 or 1, you can use torch.round() function

7. Visualize the results. Include the following graphs:

   a. A graph that compares training, validation and test accuracy on the same plot with clear labeling.

   b. A graph that compares training, validation and test loss on the same plot with a clear labeling.
      Note: your test accuracy and loss are obtained during the prediction phase and it is a single value. To plot it on the same graph, you can repeat this value for the same number of epochs as training, to get a straight line and plot all of them in the same graph.

   c. Confusion matrices on the test data (seaborn.heatmap() or PyTorch)

   d. ROC curve (receiver operating characteristic curve).
      - Details about ROC
      - PyTorch ROC function

Note:

- The expected accuracy on the testing dataset for the NN is > 80%. If you want to improve your model results, you can choose a different loss function. There are a few methods that can help improve training speed and accuracy. You can find and try them (e.g. earlystopping, k-fold, learning rate scheduler, batch normalization, data augmentation, gradient accumulation, more details on performance tuning)
- You can use any inbuilt functions or define some functions from scratch.
- All libraries are allowed, except those with pre-trained models or predefined architectures. Submissions with pre-trained models or predefined architectures will not be considered.

In your report for Part II:

1. Provide a brief overview of your dataset (e.g. type of data, number of samples, and features. Include key statistics (e.g. mean, standard deviation, number of missing values for each feature).

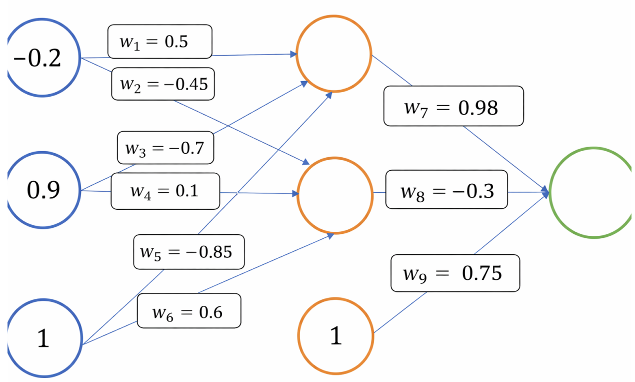
2. Include at least 3 graphs, such as histograms, scatter plots, or correlation matrices. Briefly describe the insights gained from these visualizations.

3. Describe the NN you have defined.
4. Describe how one or more of the techniques (regularization, dropout, early stopping) you applied have impacted the model's performance.
5. Discuss the results and provide relevant graphs:
    a. Report training accuracy, training loss, validation accuracy, validation loss, testing accuracy, and testing loss.
    b. Plot the training and validation accuracy over time (epochs).
    c. Plot the training and validation loss over time (epochs).
    d. Generate a confusion matrix using the model's predictions on the test set.
    e. Report any other evaluation metrics used to analyze the model's performance on the test set.

Part III: Deep Learning Theoretical Part [20 points]

1. Forward-Backward Pass [15 points]

Consider the following neural network with initialized weights.



Given the following setup:

- Hidden layer activation function: ReLU
- Output layer activation function: Linear
- Learning rate: 0.03
- Target (y): 0.5

Task:

1. Perform a forward pass and estimate the predicted output ($\hat{y}$)

2. Estimate the MSE
3. Find the gradient using back-propagation
4. Update the weights
5. Perform a forward pass to estimate the predicted output using the updated weights.
6. Estimate the MSE and compare the results with Step 2.

2. Derivative of the sigmoid function [5 points]

The sigmoid activation function has the following form:

$$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$$

Task:

Prove that the derivative of the sigmoid function has the following form:

$$f'(x) = \sigma(x)(1 - \sigma(x))$$

In your report for Part III:

1. For this part, you need to submit only the report with your solutions. You can use docx, LaTeX, or a hand-written and scanned format. Convert the result to .pdf
2. Include the solution for both tasks as part of your final report.

Bonus Points [max 10 points]

Improve Buffalo Using NN [10 points]

Generate a use-case scenario that uses NN to help improve Buffalo. For example, you might develop a model that accurately predicts crime in a given region or recommends the optimal location for a new car-repair shop based on car traffic data.

Requirements:

1. Define the use-case scenario. Explain how it can benefit the city, its residents, or a business.
2. Select a dataset or combine multiple datasets relevant to your use-case scenario. At least one dataset should come from Open Data Buffalo: https://data.buffalony.gov/
3. Preprocess the datasets and prepare them for training.
4. Design and train the NN. You are welcome to reuse your structure from Part II.

   Note: Accepted accuracy is above 75%.

5. Evaluate the performance and analyze your results, follow guidelines from Part II.

Submission:

- Create a separate folder named as `Your_UBIT_assignment1_bonus`
  e.g., `mgao8_assignment1_bonus`
- You can duplicate code from your Part II if needed
- Include all the files needed in the folder

In your report for Bonus Task:

1. Provide details of the Buffalo-related case scenario and how it can benefit the city or a local business
2. Provide a brief overview of your dataset (e.g. type of data, number of samples, and features. Include key statistics (e.g. mean, standard deviation, number of missing values for each feature).
3. Include at least 3 graphs, such as histograms, scatter plots, or correlation matrices. Briefly describe the insights gained from these visualizations.
4. Describe the NN you have defined.
5. Discuss the results and provide relevant graphs
6. Summarize your findings in a business context. Explain how your NN prediction can be applied to make data-driven decisions that benefit Buffalo or its local businesses. Make a concise summary that business owners can easily understand and use.

ASSIGNMENT STEPS

1. Checkpoint submission (Part I):

Complete Part I. Add all your assignment files in a zip folder, including .ipynb files, the report, and a txt file with a link to UBbox in a zip folder. Name the zip folder with all the files as:
`assignment1_checkpoint_YOUR_UBIT.zip`
e.g., `assignment1_checkpoint_mgao8.zip`

1. Report
   Submit as a PDF file: `a1_report_YOUR_UBIT.pdf`
   E.g. `a1_report_mgao8.pdf`

   Include all the references at the end of your report that have been used to complete the assignment.

2. Code
   Submit Jupyter Notebook files with saved outputs.
   - `a1_part_1_YOUR_UBIT.ipynb`
3. Saved Weights:
   - Go to UBbox > New > Folder > CSE 676-C Assignment 1 by YOUR NAME
   - Upload your saved weights to this folder. Include the model weights that generate the best results for your model for Part II, named as `a1_part#_YOUR_UBIT.pt`
     e.g. `a1_part_2_mgao8.pt`
   - Copy a shared link to the UBbox folder, so it can be viewed by people in your company & it can be viewed and downloaded.

   - Add the link to the txt file, named as `a1_weights_YOUR_UBIT.txt`
   - Submit this txt file as part of your submission on UBlearns
4. Dataset should NOT be submitted. Don't include as part of the submission
5. Combine all files in a single zip folder that will be submitted on UBlearns, named as `assignmen1_checkpoint_YOUR_UBIT.zip`

2. Submit final results (03/01/2026)

- Fully complete all parts of the assignment
- Submit to UBLearns > Assignments
- Add all your assignment files in a zip folder including ipynb files for Part I, Part II, Part III & Bonus part (optional), the report and txt file with a link to UBbox
- Name zip folder with all the files as `assignment1_final_YOUR_UBIT.zip`
  e.g. `assignment1_final_mgao8.zip`
- Dataset should NOT be submitted. Don't include as part of the submission.
- Your Jupyter notebook should be saved with the results. If you are submitting python scripts, after extracting the ZIP file and executing command python main.py in the first level directory, all the generated results and plots you used in your report should appear printed out in a clear manner.

- Include all the references at the end of your report that have been used to complete the assignment.
- After your references, you should have your Team Participation Statement. Submissions without a correctly formatted statement will receive a 0 for the assignment. The team participation statement should have the following:
  - Information on the parts of the assignment that each team member contributed to. We do not need significant details; a few sentences should be enough (e.g., "Person X contributed to data preprocessing, Person Y contributed to data visualization, and Person Z improved the accuracy with data augmentation.").
  - A statement by each team member that expresses their explicit agreement with the above-stated distribution of work. I.e. something like "I agree that this statement accurately reflects the distribution of work in our group. - Your Name".
- You can make an unlimited number of submissions, and only the latest will be evaluated.

Notes:

- Ensure that your code follows a clear structure and contains comments for the main functions and specific attributes related to your solution. You can submit multiple files, but they all need to be labeled with a clear name.
- Recheck the submitted files, e.g. download and open them, once submitted and verify that they open correctly

Academic Integrity

The standing policy of the Department is that all students involved in any academic integrity violation (e.g., plagiarism in any way, shape, or form) will receive an F grade for the course. The catalog describes plagiarism as "Copying or receiving material from any source and submitting that material as one's own, without acknowledging and citing the particular debts to the source, or in any other manner representing the work of another as one's own.". Refer to the Office of Academic Integrity for more details.

Important Information

This assignment should be completed individually.

- No collaboration, cheating, and plagiarism is allowed in assignments, quizzes, the midterms or final project.

- All the submissions will be checked using SafeAssign as well as other tools. SafeAssign is based on the submitted works for the past semesters as well the current submissions. We can see all the sources, so you don't need to worry if there is a high similarity with your Checkpoint submission.

- The submissions should include all the references. Kindly note that referencing the source does not mean you can copy/paste it fully and submit as your original work. Updating the hyperparameters or modifying the existing code is a subject to plagiarism. Your work has to be original. If you have any doubts, send a private post on piazza to confirm.

- All parties involved in any suspicious cases will be officially reported using the Academic Dishonesty Report form. What does that mean?

  - In most cases, the grade for the assignment/quiz/final project/midterm will be 0 and all bonus points will be subject to removal from the final evaluation for all students involved.

  - Those found violating academic integrity more than once throughout their program will receive an immediate F in the course.

  Please refer to the Academic Integrity Policy for more details.

- The report should be delivered as a separate pdf file. You can combine report for Part I, Part II, Part III & Part IV into the same pdf file. You can follow the NIPS template as a report structure. You may include comments in the Jupyter Notebook; however, you will need to duplicate the results in a separate pdf file.

- All the references can be listed at the end of the report. There is no minimum requirement for the report size, just make sure it includes all the information required.

Late Days Policy

You can use up to 5 late days throughout the course that can be applied to any assignment-related due dates. You do not have to inform the instructor, as the late submission will be tracked in UBlearns.

Important Dates

- 02/22/2026 – Checkpoint Submission is Due

- 03/01/2026 – Final Submission is Due