MiniProject 3: Classification of Image Data with Multilayer Perceptrons and Convolutional Neural Networks

COMP 551, Fall 2025, McGill University
Contact TAs: Charlotte Volk, Zahra TehraniNasab

Please read this entire document before beginning the assignment.

Preamble

- This mini-project is **due on November 18th 11:59pm (EST, Montreal Time).** There is a penalty of 2^k percent penalty for k days of delay, which means your grade will be scaled to be out of $100 - 2^k$. No submission will be accepted after 6 days of delay.
- This mini-project is to be completed in groups. All members of a group will receive the same grade except when a group member is not responding or contributing to the project. If this is the case and there are major conflicts, please reach out to the group TA for help and flag this in the submitted report. Please note that it is not expected that all team members will contribute equally. However every team member should make integral contributions to the project, be aware of the content of the submission and learn the full solution submitted.
- You will submit your assignment on MyCourses as a group. You must register your group on MyCourses and any group member can submit. See MyCourses for details.
- We recommend to use **Overleaf** for writing your report and **Google colab** for coding and running the experiments. The latter also gives access to the required computational resources. Both platforms enable remote collaborations.
- There are additional compute resources available for this project, including a SLURM cluster from McGill's School of Computer Science. Please check MyCourses for instructions on how to access those, as well as an upcoming tutorial.
- You should use Python for this mini-project. You are free to use libraries with general utilities, such as matplotlib, numpy and scipy for Python, unless stated otherwise in the description of the task. In particular, in most cases you should implement the models and evaluation functions yourself, which means you should not use pre-existing implementations of the algorithms or functions as found in SciKit learn, and other packages. The description will specify this in a per case basis.
- You should not use large language models (LLMs) to complete this assignment. Relying on them will hinder your learning, the purpose of the work is to practice the underlying skills yourself.

Background

In this miniproject, you will **implement a multilayer perceptron from scratch**, and use it to **classify image data**. One of the goals is to implement a basic neural network and its training algorithm from scratch and get hands-on experience with important decisions that you have to make while training these models. You will also have a chance to experiment with **convolutional neural networks** and use **pre-trained** versions of them to perform image classification.

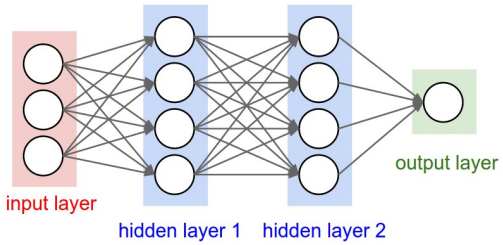

Figure 1: An MLP with 2 hidden layers each having 4 units.

Task 1: Acquire the data

Your first task is to acquire the image dataset. You will be using only one dataset in your experiments: **FashionMNIST**. Use the FashionMNIST dataset from Kaggle. Use the default test partition and split the default training partition into training and validation partitions. Unless specified, use the 28 pixels by 28 pixels version of the dataset. You can load the dataset using the following code:

```
from torchvision import datasets

train_dataset = datasets.FashionMNIST(root='./data', train=True, download=True,
                transform=train_transform)
test_dataset = datasets.FashionMNIST(root='./data', train=False, download=True)
```

The dataset's Github repo has more instruction on how to load the dataset, including how to use a PyTorch dataloader, which you are allowed to use.

Note that while working with multilayer perceptrons, after loading the data, you will have to vectorize (or "flatten") it so that it can have the appropriate dimensions. Also do not forget to normalize the training and test set (see `https://cs231n.github.io/neural-networks-2/#datapre`).

Based on your previous miniproject, you might be asking the question: where are the features? Well, this is the whole point of using neural nets: instead of hand-designing the features, you train the model so that the feature extractor is also learned together with the classifier on top.

Task 2: Implement a Multilayer Perceptron

In this mini-project, you will implement a multilayer perceptron (MLP) to classify image data. An MLP is composed of three types of layers: (1) an input layer, (2) hidden layers, (3) an output layer (see Figure 1). You should implement it from scratch based on the code available in the slides. Your implementation should include the backpropagation and the mini-batch gradient descent algorithm used (e.g., SGD).

You are free to implement the MLP as you see fit, but you should follow the equations that are presented in the lecture slides, and you must implement it from scratch (i.e., you **cannot** use TensorFlow or PyTorch or any other library). Using the Numpy package is encouraged. Regarding the implementation, we recommend the following approach:

- Implement the MLP as a python class. The constructor for the class should take as input the activation function (e.g., ReLU), the number of hidden layers (e.g., 2) and the number of units in the hidden layers (e.g., $[64, 64]$) and it should initialize the weights and biases (with an initializer of your choice) as well as other important properties of the MLP.
- The class should have (at least) two functions:

  - A `fit` function, which takes the training data (i.e., $\mathbf{X}$ and $\mathbf{y}$)—as well as other hyperparameters (e.g., the learning rate and number of gradient descent iterations)—as input. This function should train your model by modifying the model parameters.
  - A `predict` function, which takes a set of input points (i.e., $\mathbf{X}$) as input and outputs predictions (i.e., $\hat{\mathbf{y}}$) for these points.
- In addition to the model classes, you should also define a functions `evaluate_acc` to evaluate the model accuracy. This function should take the true labels (i.e., $\mathbf{y}$), and target labels (i.e., $\hat{\mathbf{y}}$) as input, and it should output the accuracy score.

You are also free to use any Python libraries you like to tune the hyper-parameters; see for example `https://scikit-learn.org/stable/modules/grid_search.html`.

Task 3: Run the experiments and report

The goal of the experiments in this part is to have you explore the consequences of important decisions made while training neural networks. You are welcome to perform any experiments and analyses you see fit (e.g., the effect of data augmentation / dropout regularization / number of hidden layers / ... on accuracy), **but at a minimum you must complete the following experiments in the order stated below**:

1. First of all, create three different models: (1) an MLP with no hidden layers, i.e., it directly maps the inputs to outputs, (2) an MLP with a single hidden layer having 256 units and ReLU activations, (3) an MLP with 2 hidden layers each having 256 units with ReLU activations. It should be noted that since we want to perform classification, all of these models should have a softmax layer at the end. After training, compare the test accuracy of these three models on the FashionMNIST dataset. Comment on how non-linearity and network depth affects the accuracy. Are the results that you obtain expected?
2. Take the last model above, the one with 2 hidden layers, and create two different copies of it in which the activations are now tanh and Leaky-ReLU. After training these two models compare their test accuracies with model having ReLU activations. Comment on the performances of these models: which one is better and why? Are certain activations better than others? If the results are not as you expected, what could be the reason?
3. Create an MLP with 2 hidden layers each having 256 units with ReLU activations as above. However, this time, independently add L1 and L2 regularization to the network and train the MLP in this way. How does these regularizations affect the accuracy? This proportion can be varied as a tunable hyperparameter that can be explored as part of other project requirements.
4. Create an MLP with 2 hidden layers each having 256 units with ReLU activations as above. However, this time, train it with unnormalized images. How does this affect the accuracy?
5. Re-train the MLP from question 3 on a version of FashionMNIST using data augmentation. You can use the `transforms.Compose()` function to set your transformations for data augmentation, and the `transform=train_transform` argument in the dataset constructor to set the transforms. Is the accuracy affected, and how? What are the benefits and/or drawbacks of using data augmentation? Can you think of a situation in which certain types data augmentation would be harmful?
6. Using existing libraries such as TensorFlow or PyTorch, create a convolutional neural network (CNN) with 2 convolutional layers, one fully connected hidden layer and one fully connected output layer. Although you are free in your choice of the hyperparameters of the convolutional layers, set the number of units in the fully connected layers to be 256. Also, set the activations in all of the layers to be ReLU. Train this CNN on the FashionMNIST dataset. Does using a CNN increase/decrease the accuracy compared to using MLPs? Provide comments on your results.
7. Train the above CNN using FashionMNIST with the data augmentation from Q5. How is the performance (accuracy and speed) affected?

8. Load a pre-trained model that you see fit (e.g., a ResNet) using existing libraries such as TensorFlow or PyTorch, and then freeze all the convolutional layers and remove all the fully connected ones. Add a number of fully connected layers of your choice right after the convolutional layers. Train only the fully connected layers of the pre-trained model on the FashionMNIST dataset with the data augmentation from Q5. How does this pre-trained model compare to the best MLP in part 5 and to the CNN in part 7 in terms of accuracy? How does it compare to the previous models in terms of the required training time? Justify your choice of how many fully connected layers that you have added to the pre-trained model through careful experiments.
9. You can report your findings either in the form of a table or a plot in the write-up. However, include in your colab notebooks the plots of the test and train performance of the MLPs / CNN / pre-trained model as a function of training epochs. This will allow you to see how much the network should be trained before it starts to overfit to the training data.

Note 1: The above experiments are the minimum requirements that you must complete; however, this project is open-ended.

For example, you might investigate the effect of the width (number of units in the hidden layers) of the MLP on its test accuracy or the effect of the CNN's convolutional layer hyperparameters (number of filters, kernel size, stride, padding, ...) on its test accuracy. It is also possible to examine the effect of the usage of different pre-trained models on the final accuracy and training speed of the network. Another interesting thing to report might be training the MLP / CNN / pre-trained model with $10^k, k \in \{0, 1, 2, 3, 4\}$ images and plotting the test accuracy. You do not need to do all of these things or tune every parameter, but you should demonstrate creativity, rigor, and an understanding of the course material in how you run your chosen experiments and how you report on them in your write-up.

Note 2: We expect you to provide plots/tables in your report that justifies your choice of hyperparameters (the learning rates of the MLPs / CNNs / pretrained models, the architectural parameters of the CNNs and pretrained models). You are not required to perform cross-validation in this project.

Deliverables

You must submit two separate files to myCourses (**using the exact filenames and file types outlined below**):

1. **code.zip**: Your model implementation, and its training and evaluation code (as some combination of .py and .ipynb files).
2. **writeup.pdf**: Your (max 5-page) project write-up as a pdf (details below).

Write-up instructions

Your team must submit a project write-up that is a maximum of five pages (single-spaced, 11pt font or larger; minimum 0.5 inch margins, an extra page for references/bibliographical content can be used). We highly recommend that students use LaTeX to complete their write-ups. You have some flexibility in how you report your results, but you must adhere to the following structure and minimum requirements:

Abstract (100-250 words) Summarize the project task and your most important findings.

Introduction (5+ sentences) Summarize the project task, the datasets, and your most important findings. This should be similar to the abstract but more detailed. You should include background information and a few citations to relevant work (e.g., other papers analyzing these datasets).

Datasets (5+ sentences) Very briefly describe the dataset. Present the exploratory analysis you have done to understand the data, e.g. class distribution.

Results (7+ sentences, possibly with figures or tables) Describe the results of all the experiments mentioned in **Task 3** (at a minimum) as well as any other interesting results you find (Note: demonstrating figures or tables would be an ideal way to report these results).

Discussion and Conclusion (5+ sentences) Summarize the key takeaways from the project and possibly directions for future investigation.

Statement of Contributions (1-3 sentences) State the breakdown of the workload across the team members.

Evaluation

The mini-project is out of 100 points, and the evaluation breakdown is as follows:

- Completeness (20 points)
  - Did you submit all the materials? (5 points)
  - Did you run all the required experiments? (5 points)
  - Did you follow the guidelines for the project write-up? (10 points)
- Correctness (40 points)
  - Implementation of data normalization (1 points)
  - Implementation and training of MLP with no hidden layers (1 points)
  - Implementation and training of MLP with one hidden layers and ReLU activation (3 points)
  - Implementation and training of MLP with two hidden layers and ReLU activation (1 points)
  - Implementation and training of the two hidden layers MLP with tanh activation (2 points)
  - Implementation and training of the two hidden layers MLP with Leaky-ReLU activation (2 points)
  - L1 regularization (2 points)
  - L2 regularization (2 points)
  - Train MLP without normalization (1 points)
  - Correctly plot and compare the results of the 8 trained models above (5 points)
  - Train the MLP with the 128 x 128 FashionMNIST data (1 points)
  - Plot the results of the larger models, and compare classification performance and training time (2 points)
  - Implement and train the CNN (2 points)
  - Re-train the CNN with the 128 x 128 FashionMNIST data (1 points)
  - Plot the results of the two CNN trained, and compare classification performance and training time to the best MLP. (4 points)
  - Implement and train the pre-trained model with the trainable fully connected layer(s). (3 points)
  - Plot the results of the pre-trained model, and compare its performance to the best MLP and regular CNN. (4 points)
  - Run an experiment to justify the choice of fully connected layers for the pre-trained model, and show supporting plots. (3 points)
- Writing quality (30 points)
  - Is your report clear and free of grammatical errors and typos? (10 points)
  - Did you go beyond the bare minimum requirements for the write-up (e.g., by including a discussion of related work in the introduction)? (10 points)

  - Do you effectively present numerical results (e.g., via tables or figures)? (10 points)

- Originality / creativity (10 points)
  - Did you go beyond the bare minimum requirements for the experiments?
  - **Note:** Simply adding in a random new experiment will not guarantee a high grade on this section! You should be **thoughtful and organized** in your report.

Final remarks

You are expected to display initiative, creativity, scientific rigour, critical thinking, and good communication skills. You don't need to restrict yourself to the requirements listed above - feel free to go beyond, and explore further.

You can discuss methods and technical issues with members of other teams, but **you cannot share any code or data with other teams.**