Instructor: Alina Vereshchaka

Assignment 2
Building Neural Networks and Convolutional Neural Networks

Checkpoint: October 26, Thu, 11:59pm

Due Date: November 9, Thu, 11:59pm

Description

Welcome to our second assignment. This assignment focuses on building fully connected neural networks (NN) and convolutional neural networks (CNN). It consists of four parts where you practice dealing with various datasets and implement, train, and adjust neural networks.

The first part consists of performing data analysis and building a basic NN. In the second part, we learn how to optimize and improve your NN using various techniques. In the third part, we will implement a basic CNN and apply optimization and in part 4 we will implement the VGG-11 model.

Apart from this do not miss out on some extra bonus points, by completing the task described at the end of this description.

Note for using libraries: For this assignment, any pre-trained or pre-built neural networks or CNN architectures cannot be used (e.g. torchvision.models, keras.applications). This time you can use scikit-learn for data preprocessing.

For this assignment we use PyTorch as a deep learning framework. You can refer to the Pytorch tutorials on how to get started.

Background

Let's consider a generic structure for defining a neural network in Pytorch. Click here for more details.

```
class NeuralNetwork(nn.Module):
    def __init__(self):
        super(NeuralNetwork, self).__init__()
        self.flatten = nn.Flatten()
        self.linear_relu_stack = nn.Sequential(
            nn.Linear(28*28, 512),
            nn.ReLU(),
            nn.Linear(512, 512),
            nn.ReLU(),
```

```
            nn.Linear(512, 10),
        )

    def forward(self, x):
        x = self.flatten(x)
        logits = self.linear_relu_stack(x)
        return logits

model = NeuralNetwork().to(device)
print(model)


X = torch.rand(1, 28, 28, device=device)
logits = model(X)
pred_probab = nn.Softmax(dim=1)(logits)
y_pred = pred_probab.argmax(1)
print(f"Predicted class: {y_pred}")
```

This code defines a neural network with three layers:

- Flatten layer that flattens the input image tensor
- Two fully connected layers with 512 hidden units and ReLU activation functions
- Final fully connected layer with 10 output units (corresponding to the 10 possible classes in the MNIST dataset)

The forward method specifies how input data is passed through the layers of the network.

It defines a neural network using PyTorch's `nn.Module` class and the `nn.Sequential` module, and then uses the network to make a prediction.

`nn.Module` is a base class in the PyTorch module that provides a framework for building and organizing complex neural network architectures.

When defining a neural network module in PyTorch, you usually create a class that inherits from `nn.Module`. The `nn.Module` class provides a set of useful methods and attributes that help in building, training and evaluating the neural network.

Some of the key methods provided by `nn.Module` are:

`__init__()`: This is the constructor method that initializes the different layers and parameters of the neural network.

`forward(self, x)`: This method defines the forward pass of the neural network. It takes the input tensor (x) and returns the predicted probabilities for each class.

`nn.Linear` is a PyTorch module that applies a linear transformation to the input data. It is one of the most used modules in deep learning for building neural networks.

The `nn.Linear` module takes two arguments: `in_features` and `out_features` - the number of input/output features. When an input tensor is passed through an `nn.Linear` module, it is first flattened into a 1D tensor and then multiplied by a weight matrix of size `out_features x in_features`. A bias term of size `out_features` is also added to the output.

The output of an `nn.Linear` module is given by the following equation:

$$output = w^T x + b$$

```
model = NeuralNetwork().to(device)
print(model)
```

Here we create an instance of the NeuralNetwork class and moves it to the device specified by the device variable (which should be set to 'cuda' for GPU or 'cpu' for CPU). It is also good practice to print the summary of the architecture of the NN.

```
X = torch.rand(1, 28, 28, device=device)
logits = model(X)
pred_probab = nn.Softmax(dim=1)(logits)
y_pred = pred_probab.argmax(1)
print(f"Predicted class: {y_pred}")
```

This code generates a random input image tensor of size 1x28x28 (representing a single 28x28 grayscale image) and passes it through the neural network using the model instance. The output of the network is a tensor of size 1x10, representing the predicted probabilities for each of the 10 possible classes.

`nn.Softmax` module is used to convert these probabilities to a valid probability distribution, and the argmax method is used to obtain the class with the highest probability.

Part I: Building a Basic NN [30 points]

In this part, you will implement a neural network using the PyTorch framework. You will train the network on the dataset provided, which contains of seven features and a target. Your goal is to predict a target, that has a binary representation.

Step 1: Loading the Dataset and main statistics

1. Load the dataset (dataset.csv). It is provided on UBlearns > Assignments. You can use the pandas library to load the dataset into a DataFrame
2. Analyze the dataset, e.g., return the main statistics.
3. Provide at least 3 visualization graphs with a short description.

Note: You can reuse your code from A0 or A1 with a proper citation.

Step 2: Preprocessing and Splitting the Dataset

1. Preprocess the dataset before we use it to train the neural network.

   Preprocessing typically involves converting categorical variables to numerical variables, scaling numerical variables.

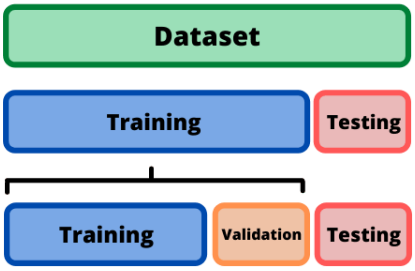
   For this dataset, you can use the following preprocessing steps:

   - Convert categorical variables to numerical variables using one-hot encoding. You can use OneHotEncoder from scikit-learn
   - Scale numerical variables to have zero mean and unit variance. You can use StandardScaler from scikit-learn or Normalize from PyTorch

2. 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.

Step 3: Defining the Neural Network

Now, we need to define the neural network that we will use to make predictions on the dataset. For this part, you can define a simple neural network.

1. Decide your NN architecture:
   - 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 4: Training the Neural Network

1. Set up the training loop. In this step, you will create a loop that iterates over the training data for a specified number of epochs.
   - For each epoch, you will iterate over the batches of the training data, compute the forward pass through the neural network, compute the loss, compute the gradients using backpropagation, and update the weights of the network using an optimizer such as Stochastic Gradient Descent (SGD) or Adam.
   - After getting the training loss, set the validation as zero and the model to evaluation model. Disable the gradient computation, start a loop and iterate it over batches of the validation set. Compute the forward pass and the loss. There will be no backward propagation to calculate the gradients and no optimizer step to update the steps.
2. Define the loss function that will be used to compute the error between the predicted output of the neural network 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 (PyTorch).

3. Choose an optimizer and a learning rate. It will update the weights of the neural network during training. Stochastic Gradient Descent (SGD) is one of the commonly used, you can also explore other optimizers like Adam or RMSProp. Check a list of optimizers (PyTorch)
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 to the training data.
5. Evaluate the performance of the model on the testing data. Suggested metrics:
    - Time to train (e.g. using time.time())
    - Accuracy (PyTorch functions)
    - Precision, recall and F1 score (more details). You can use sklearn.metrics.precision_recall_fscore_support

    Note: The expected accuracy on the testing dataset for this task is > 75%.
6. Save the weights of the trained neural network.
   Saving and loading models in PyTorch
7. Visualize the results. Include the following graphs:
    a. Confusion matrices (seaborn.heatmap() or PyTorch)
    b. ROC curve (receiver operating characteristic curve).
        - Details about ROC
        - PyTorch ROC funnction
    c. A graph that compares test, validation and training accuracy on the same plot with a clear labeling
    d. A graph that compares test, validation and training loss on the same plot with a clear labeling

Report for Part I:

1. Provide brief details about the nature of your dataset. What type of data are we encountering? How many entries and variables does the dataset comprise? Provide the main statistics about the entries of the dataset.
2. Provide at least 3 visualization graphs with short description for each graph.
3. For the preprocessing part, discuss if you use any preprocessing tools, that helps to increase the accuracy of your model.
4. Provide a summary of your model.
5. Provide performance metrics and your analysis (Step 5)
6. Provide graphs and your analysis (Step 7)

Part II: Optimizing NN [20 points]

Based on your NN model defined in Part I, tune the hyperparameters and apply different tools to increase accuracy. Try various setups and draw conclusions.

STEPS:

1. Choose one hyperparameter to modify (e.g., Dropout). Fix the NN structure and all other parameters based on the model defined in Part I and change values only for your chosen hyperparameter.
    - Provide the results in the form of a table below.
    - Save the model and the weights.
    - Visualize the results. Provide the graphs mentioned in Part I step 7.

"NAME OF THE HYPERPARAMETER" Tuning

	Setup 1	Test Accuracy	Setup 2	Test Accuracy	Setup 3	Test Accuracy
Dropout						
Optimizer						
Activation Function						
Initializer						

2. Choose another hyperparameter and go to step 1. Do the same procedure for three hyperparameters. You can extend the list of hyperparameters, if needed.
3. In total, we need to submit a base NN model (defined in Part 1) + three NN models with one changed hyperparameter. E.g., four different NN models, that have slight changes. You can provide more NN versions on top of these, if needed.
4. After completing step 2, choose a model setup that returns the best accuracy and use it as a 'base' model.

   There are a few methods which can help increase the training speed, accuracy, etc. Find and try at least fourr different methods (e.g. earlystopping, k-fold, learning rate scheduler, batch normalization)

    a. Choose a training optimization method. Add it to your 'base' model.
    b. Train the model with a new method. Provide a graph that compares test accuracy for a 'base' model and an improved version. You can also provide comparison w.r.t training time and other parameters.
    c. Go to step 4a. Try four various methods or tools.

   Note: This step does not involve the use of different optimizers, you can experiment with various optimizers in Part I.

Report for Part II

1. Include all three tables with different NN setups.

2. Provide comparison graphs that compare test and training accuracy on the same plot for all your setups and add a short description for each graph.
3. Provide a detailed analysis and reasoning about the NN setups that you tried.
4. Briefly discuss all the methods you used that help to improve the accuracy or the training time. Provide visualization graphs and your short descriptions.

Checkpoint Submission (Part I & Part II):

- Report (as a pdf file): named as
  TEAMMATE1_TEAMMATE2 _assignment2_report.pdf
  e.g., avereshc_ pinazmoh_ assignment2_report.pdf
- Code (as ipynb file with saved outputs) named as

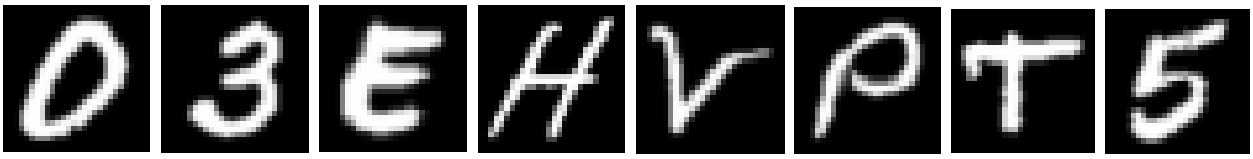
  TEAMMATE1_TEAMMATE2 _assignment2_part_1_2.ipynb
  e.g., avereshc_ pinazmoh_ assignment2_ part_1_2.ipynb
- Files with saved weights that generate the best results for each parts named as

  TEAMMATE1_TEAMMATE2 _assignment2_part1.h5
  TEAMMATE1_TEAMMATE2 _assignment2_part2.h5
- Dataset should NOT be submitted. Don't include as part of the submission

Part III: Building a CNN [30 points]

In this part we will work on a dataset containing multiple classes: from 0-9 and A-Z.

CNN Dataset



For this part our dataset consists of 36 categories and 2800 examples for each category, thus in total 100,800 samples. Each example is a 28x28 image. A version of the dataset for this assignment can be downloaded from UBlearns (cnn_dataset.zip)

Read more about the dataset (paper)

STEPS:

1. Load, preprocess, analyze, visualize the dataset and make it ready for training.

   You can reuse your code from Part I.

2. Build and train a basic CNN (with max 10 hidden layers).

   Decide your CNN architecture:
   - 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?
   - What is the kernel size, number of filters, strides, paddings and other CNN-parameters?
   - Do you include Dropout?

   You can consider AlexNet to get some ideas for your network (paper).
3. Define your CNN architecture using PyTorch (basic building blocks in PyTorch) and return the summary of your model.
4. Train your model.
5. Add improvement methods that you tried for "Part III - Step 3" of this assignment, that are applicable to CNN architecture (e.g., earlystopping).
8. Evaluate the performance of the model on the testing data. Suggested metrics:
   - Time to train (e.g. using time.time())
   - Accuracy (PyTorch functions)
   - Precision, recall and F1 score (more details). You can use sklearn.metrics.precision_recall_fscore_support

   Note: The expected accuracy on the testing dataset for this task is > 85%.
9. Save the weights of the trained neural network.
   Saving and loading models in PyTorch
10. Visualize the results. Include the following graphs:
    a. Confusion matrices (seaborn.heatmap() or PyTorch)
    b. ROC curve (receiver operating characteristic curve).
       - Details about ROC
       - PyTorch ROC funnction
    c. A graph that compares test, validation and training accuracy on the same plot with a clear labeling
    d. A graph that compares test, validation and training loss on the same plot with a clear labeling

Report for Part III:

1. Provide brief details about the nature of your dataset. What is it about? What type of data are we encountering? How many entries and variables does the dataset comprise? Provide the main statistics about the entries of the dataset
2. Provide at least 3 visualization graphs with short description for each graph.
3. Provide the model details of your CNN.
4. Discuss how the improvement tools work on CNN architectures.
5. Provide the performance metrics and analyze the results (Step 8)
6. Provide graphs and analyze the results (Step 10)

Part IV: VGG-11 Implementation [20 points]

VGG is one of the commonly used CNN architectures. It has different versions. In this part we implement the VGG-11 (Version A) and apply it to solve our task.

STEPS:

1. Implement the VGG-11 (Version A) architecture following the proposed architecture. Be sure to use the same kernel sizes, stride, and padding as specified in the paper.

   You can refer to the original VGG paper for more details.
2. Train the model on a dataset used in Part III (EMNIST)
3. Evaluate the performance of the model on the testing data. Suggested metrics:
   - Time to train (e.g. using time.time())
   - Accuracy (PyTorch functions)
   - Precision, recall and F1 score (more details). You can use sklearn.metrics.precision_recall_fscore_support

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

4. Visualize the results. Include the graphs you defined in Part III, Step 10.

Report for Part IV

1. Provide the model details of your CNN.
2. Discuss how the architecture different from your CNN architectures defined in Part III.
3. Provide the performance metrics (Step 3) and compare them with the results obtained in Part III.
4. Provide graphs (Step 4) and compare them with the results obtained in Part III.

Final submission includes:

- For the final submission, combine the reports for all parts into one file UBIT TEAMMATE1_TEAMMATE2 _assignment2_report.pdf (e.g. avereshc_ pinazmoh_ assignment2_report.pdf)
- Part I & II: TEAMMATE1_TEAMMATE2 _assignment2_part1_2.ipynb (e.g. team1_avereshc_ pinazmoh_ assignment2_ part1_2.ipynb)
- Part III & IV: TEAMMATE1_TEAMMATE2 _assignment2_part3_4.ipynb (e.g. avereshc_ pinazmoh_ assignment2_ part3_4.ipynb)
- Files with saved weights that generate the best results for each parts named as TEAMMATE1_TEAMMATE2 _assignment2_part1.h5 TEAMMATE1_TEAMMATE2 _assignment2_part2.h5 TEAMMATE1_TEAMMATE2 _assignment2_part3.h5 TEAMMATE1_TEAMMATE2 _assignment2_part4.h5
- Add all your assignment files in a zip folder including ipynb files for Part I, Part II, Part III. Part IV & Bonus part (optional), the report and .h5 file at UBLearns > Assignments
- Name zip folder with all the files as TEAMMATE1_TEAMMATE2 _assignment2_final.zip e.g. avereshc_ pinazmoh_ assignment2_final.zip
- Dataset should NOT be submitted. Don't include as part of the submission

Bonus points [5 points]

Implement ResNet-34

Implement Resnet-34 model, Configuration 34-layer (refer to paper, page 774). Apply it to solve the dataset provided in Part III (available on UBlearns). Provide the performance results and graphs.

Submit it as a separate file named:

TEAMMATE#1_UBIT_TEAMMATE#2_UBIT_ bonus _assignment2.ipynb

ASSIGNMENT STEPS

1. Register your team

You may work individually or in a team of up to 2 people. The evaluation will be the same for a team of any size. Teammates for A1, A2 & A3 should be different.

Register your team at UBLearns > Groups. In case you joined the wrong group, make a private post on piazza.

2. Submit checkpoint (October 26)

- Complete Part I & II of the assignment
- Include all the references at the end of the report. There is no minimum requirement for the report size, just make sure it includes all of the information required.
- Add all your assignment files in a zip folder including .ipynb files, the report and .h5 file at a zip folder.
- Name zip folder with all the files as TEAMMATE1_TEAMMATE2 _assignment2_checkpoint.zip
  e.g., avereshc_ pinazmoh_ assignment2_checkpoint.zip
- Submit to UBLearns > Assignments
- Dataset should NOT be submitted. Don't include as part of the submission
- If you are working in a team, we expect equal contribution for the assignment. Each team member is expected to make a code-related contribution. Provide a contribution summary by each team member in the form of a table below. If the contribution is highly skewed, then the scores of the team members may be scaled w.r.t the contribution.

Team Member	Assignment Part	Contribution (%)

3. Submit final results (November 9)

- 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. Part IV & Bonus part (optional), the report and .h5 file at UBLearns > Assignments
- Name zip folder with all the files as TEAMMATE1_TEAMMATE2 _assignment2_final.zip

e.g. avereshc_ pinazmoh_ assignment2_final.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 that have been used to complete the assignment.
- If you are working in a team, we expect equal contribution for the assignment. Each team member is expected to make a code-related contribution. Provide a contribution summary by each team member in the form of a table below. If the contribution is highly skewed, then the scores of the team members may be scaled w.r.t the contribution.

Team Member	Assignment Part	Contribution (%)

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 project can be done in a team of up to two people.

- All team members are responsible for the project files submission
- 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 group members and 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.
- For the Bonus part, no report is needed.

Late Days Policy

You can use up to 7 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.

If you work in teams, the late days used will be subtracted from both partners. In other words, you have 4 late days, and your partner has 3 late days left. If you submit one day after the due date, you will have 3 days and your partner will have 2 late days left.

Important Dates

October 26, Thursday - Checkpoint is Due

November 9, Thursday - Final Submission is Due