CSE 4/574: Introduction to Machine Learning, Sections A&C&D

Spring 2023

Instructor: Alina Vereshchaka

Assignment 2
Building Neural Networks and Convolutional Neural Networks

Checkpoint: March 30, Thu, 11:59pm

Due Date: April 13, 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 data augmentation techniques in the fourth part.

Apart from this do not miss out 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 you can use PyTorch or Keras/Tensorflow deep learning framework (works using sklearn.neural_network.MLPClassifier won't be evaluated):

- Pytorch (60 mins blitz)
- Keras / Tensorflow (Getting started)

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 commonly 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 [20 points]

In this assignment, you will implement a neural network using the PyTorch/Keras library. 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

Load the dataset. It is provided on UBlearns > Assignments.

You can use the pandas library to load the dataset into a DataFrame

Step 2: Preprocessing the Dataset

First, we need to preprocess the dataset before we use it to train the neural network. Preprocessing typically involves converting categorical variables to numerical variables, scaling numerical variables, and splitting the dataset into training and validation sets.

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 sklearn
- Scale numerical variables to have zero mean and unit variance.
    - You can use StandardScaler from sklearn or Normalize from PyTorch
- Split the dataset into training and validation sets.
    - train_test_split from sklearn

You can also check Keras preprocessing tools here.

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.

Decide your NN architecture:

- How many input neurons are there?
- What activation function will you choose?
    - Suggestion: try ReLU
- What is the number of hidden layers?
    - Suggestion: start with a small network, e.g. 2 or 3 layers
- What is the size of each hidden layer?
    - Suggestion: try 64 or 128 nodes for each layer
- What activation function is used for the hidden and output layer?

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.
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.
   Check a list of loss functions (PyTorch).
   Check a list of loss functions (Keras).
   For binary classification problems, a commonly used loss function is Binary Cross Entropy Loss(PyTorch, Keras),
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)
   Check a list of optimizers (Keras)
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.

   The expected accuracy for this task is more than 75%.
6. Save the weights of the trained neural network.
   Check saving and loading models (PyTorch)
   Check saving and loading models (Keras)
7. Visualize the results. Use visualization techniques such as confusion matrices.

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 the architecture structure of your NN.
5. Provide graphs that compares test and training accuracy on the same plot, test and training loss on the same plot. Thus in total two graphs with a clear labeling.

Part II: Optimizing NN [20 points]

Based on your NN model defined in Part I, tune the hyperparameters and apply different tools to increase the 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, and change values only for your chosen hyperparameter. Provide the results in a form of a table below.

"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 3 hyperparameters. You can extend the list of hyperparameters, if needed.
3. After completing step 2, choose a model setup that returns the best accuracy and use it as a 'base' model.
   In deep learning frameworks, there are a number of new methods which appear, to help increase the training speed, accuracy, etc. Find and try at least 4 different methods (e.g. earlystopping (Pytorch, Keras), k-fold, learning rate scheduler (Pytorch, Keras), etc)
   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.
   c. Go to step a. Try 4 various methods or tools.

Report for Part II

1. Include all 3 tables with different NN setups.
2. Provide graphs that compares 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 training time. Provide accuracy 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

Part III: Implementing & Improving AlexNet [40 points]

In this part implement AlexNet, check how to improve the model, and apply that to solve an image dataset containing three classes: dogs, cars and food.

CNN Dataset

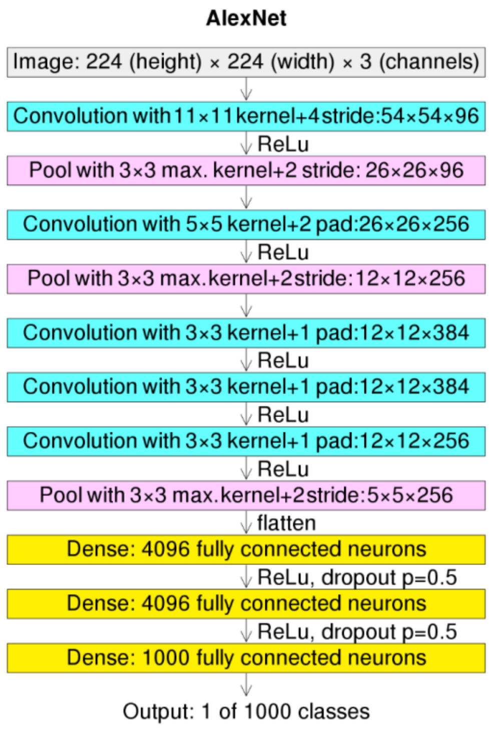
CNN dataset consists of 10,000 examples for each category, thus in total 30,000 samples. Each example is a 64x64 image. The dataset can be downloaded from UBlearns.

STEPS

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

   You can refer and reuse your code for Part I of this Assignment.

2. Build and train an AlexNet CNN architecture (paper). Here is the original architecture:


AlexNet
Image: 224 (height) × 224 (width) × 3 (channels)
Convolution with 11×11 kernel+4 stride:54×54×96
ReLu
Pool with 3×3 max. kernel+2 stride: 26×26×96
Convolution with 5×5 kernel+2 pad:26×26×256
ReLu
Pool with 3×3 max.kernel+2stride:12×12×256
Convolution with 3×3 kernel+1 pad:12×12×384
ReLu
Convolution with 3×3 kernel+1 pad:12×12×384
ReLu
Convolution with 3×3 kernel+1 pad:12×12×256
ReLu
Pool with 3×3 max.kernel+2stride:5×5×256
flatten
Dense: 4096 fully connected neurons
ReLu, dropout p=0.5
Dense: 4096 fully connected neurons
ReLu, dropout p=0.5
Dense: 1000 fully connected neurons
Output: 1 of 1000 classes

   For your dataset, adjust the size, e.g. for the input and the output layers.
3. Train the network and evaluate the performance of the AlexNet on the testing data.
4. Modify AlexNet structure (e.g. add/remove layers, update the kernel size, adjust the hyperparameters), add improvement methods that you tried for “Part II - Step 3” of this assignment, that are applicable to CNN architecture (e.g. earlystopping).
5. Train the network and evaluate the performance of the AlexNet on the testing data.

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. Provide at least 3 visualization graphs with short description for each graph.
2. Provide details about your implemented model (AlexNet). Discuss your results. Provide graphs that compares test and training accuracy on the same plot
3. Discuss how you improve AlexNet, what methods and tools you have tried and how that helped to improve the training accuracy and training time. Provide graphs that compares test and training accuracy on the same plot.

Part IV: Optimizing CNN + Data Argumentation [20 points]

Apply your improved version of CNN model defined in Part III to Google Street View House Numbers (SVHN).

The expected accuracy for this part is more than 94%.

STEPS

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

   You can refer and reuse your code for Part I of this Assignment.

1. Use your CNN architecture from Part III as a base version and adjust it for SVHN task.
2. Increase the dataset by x2 using any data augmentation techniques (rotations, shifting, mirroring, etc). You can refer to Keras documentation for more details. You can use a combination of these techniques simultaneously.
3. Train the network and evaluate the performance on the testing data.

Report for Part IV

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. Provide at least 3 visualization graphs with short description for each graph.
2. Discuss briefly how you adjusted your model from Part III for this task.
3. Discuss data augmentation methods you tries and reason how it helped to increase the accuracy of the model.
4. Provide graphs that compares test and training accuracy on the same plot for all your setups and add a short description for each graph.

Final submission includes:

- Report (as a pdf file), named as TEAMMATE#1_UBIT_TEAMMATE#2_UBIT_ part_3_4 _assignment2.pdf
- Jupyter Notebook (.ipynb) with all saved outputs, named as TEAMMATE#1_UBIT_TEAMMATE#2_UBIT_ part_3_4 _assignment2.ipynb

Bonus points [10 points]

Implement VGG-13

Implement VGG-13 model, Configuration B (refer to paper, page 3) and apply it to solve the cnn dataset from Part III.

Submit it as a separate file named:

TEAMMATE#1_UBIT_TEAMMATE#2_UBIT_ bonus _assignment2.ipynb

Deliverables

There should two parts in your submission for both the checkpoint and final submissions

Report

The report should be delivered as a separate pdf file. You can follow the NIPS template as a report structure or our provided word doc, both of which can help you to get started. You may include comments in the Jupyter Notebook, however you will need to duplicate the results in the 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 of the information required.

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)

Code

Python is the only code accepted for this assignment. You can submit the code in the Jupyter Notebook or as Python script. Ensure that your code follows a clear structure and contains comments for the main functions and some specific attributes related to your solution. You can submit multiple files, but they all need to be labeled with a clear name.

After executing command python main.py in the first level directory or Jupyter Notebook, it should generate all of the results and plots you used in your report and print them out in a clear manner.

For the final submission we should expect to have 3 Jupyter Notebooks:

- Part I & II: TEAMMATE1_TEAMMATE2 _assignment2_part1_2.ipynb (e.g. team1_avereshc_ pinazmoh_ assignment3_ part1_2.ipynb)
- Part III & IV: TEAMMATE1_TEAMMATE2 _assignment2_part3_4.ipynb (e.g. avereshc_ pinazmoh_ assignment3_ part3_4.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.

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

2. Submit checkpoint (March 30)

- Complete Part I & II of the assignment
- For the checkpoint report for Part I is not mandatory, you can complete the report and submit it along with the final submission
- Add all your assignment files in a zip folder including .ipynb files, the report (if available) and .pickle file at UBLearns > Assignments
- Name zip folder with all the files as
  TEAMMATE1_TEAMMATE2 _assignment2_checkpoint.zip
  e.g., avereshc_ pinazmoh_ assignment2_checkpoint.zip
- Submit to UBLearns > Assignments
- 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 (April 13)

- 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 .pickle file at UBLearns > Assignments
- Name zip folder with all the files as
  TEAMMATE1_TEAMMATE2 _assignment2_final.zip
  e.g. avereshc_ pinazmoh_ assignment2_final.zip
- 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

March 30, Thursday - Checkpoint is Due

April 13, Thursday - Final Submission is Due

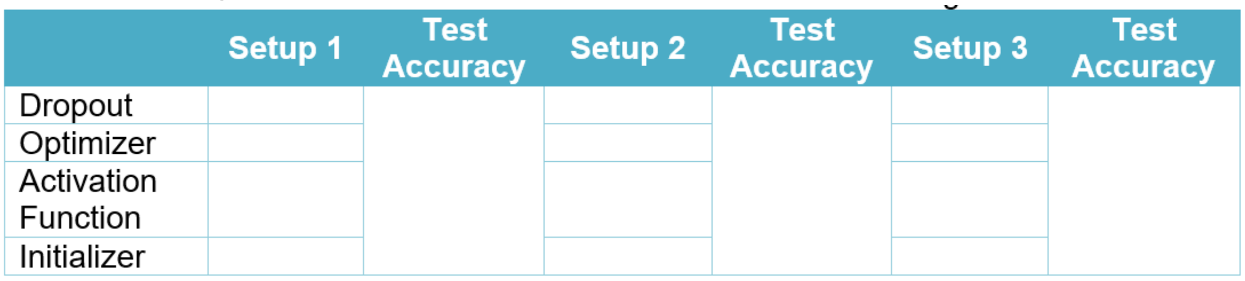
FAQ

Are we allowed to use sklearn preprocessing functions for assignment 3?
Yes

Hello! I'm unable to break the 75% training Accuracy barrier for Part1, is there any guidance that someone can provide?
You can try preprocessing the data or try playing around with various hyperparameters.

Is the max pooling layer counted as a hidden layer in part 3?
As the max pooling layer does not have any trainable parameters of its own, it should not count as a separate hidden layer.

Provided the table for the hyperparameter in task 2, there should be 3 setups if i am correct, as we keep the Dropout changing and all other hyperparameters as constant for the first setup, next changing the optimizer and keeping the other 3 hyperparameters constant and so on, so we will be having 4 setups and 4 accuracy results. Please correct me if i am wrong.



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

So here so suppose you choose 3 hyperparameters as dropout, optimizer, and activation function.

So there will be 3 tables for each of it.
For eg: let us consider Table 1 for Droppouts.
Here you have to fix the rest of the hyperparameters. And just try with 3 different values of hyperparameters (hence 3 setups).
For each table, you have to only change one of the hyperparameters.