UNIVERSITY OF BATH

CM500335: Foundations and Frontiers of Machine Learning
Group Assignment 2: Deep learning

Dr Raghubir Singh *

Department of Computer Science

Dated: October 15, 2024

*rs3022@bath.ac.uk

1 Introduction

This final assignment carries a weightage of **70 points** and will be included in this unit's overall mark of **100**. Marks will be given for each task, and you will be assigned to a group of **two people**. If there is an odd number of students, one group may have three members.

NB: It is essential to know that you can complete the assignment independently, but doing so will not earn you any extra points. Additionally, requests to reduce the workload of the assignment will not be granted. We will consider requests for additional time to complete the assignment under exceptional circumstances if they are backed up by appropriate evidence.

2 Report

A report will be your primary method of assessment. For group work, please see the attached PDF file (**Assignment 2 Template**). You will find it helpful when completing the report. Each group should evenly distribute the workload for this assignment and receive a collective grade.

Please keep in mind that the main part of your report should not exceed **3000 words**, excluding the groupwork Contribution Table. Your report should aim to demonstrate and evaluate your results and should include appropriate figures and screenshots with high resolutions. The evaluation can be both qualitative, assessing how the results appear, and, wherever possible, quantitative, involving numerical testing.

For each task/question, you should provide evidence of understanding the mathematics behind it. You should also explain your approach to solving it and the choices you make, such as hyper-parameters.

Your submission for this assignment should include a Jupyter Notebook file named **Assignment2_Group_number.ipynb**. This file should contain all the implementations related to the tasks given in the assignment. Additionally, you must include all the code dependencies in a folder named **Assignment2_Group_Number.code**. The codes should be well-commented to indicate significant steps.

Please note: Only one group member should submit the project.

3 Submission File format:

One zip file containing your main report in a PDF (font: Arial, size: 11, for including figures/tables, please make sure they are not too small) and your **Assignment2_Group_Number.code** folder.

Note: All tasks should be implemented in **Keras/TensorFlow** (PyTorch is also acceptable).

4 The Assignment

First, load the MNIST handwritten digits data using, e.g. code given here is:

- Prepare Dataset
  - (X_train, y_train), (X_test, y_test) = mnist.load_data()
    n_train = X_train.shape[0]
    n_test = X_test.shape[0]
    X_train, X_test = X_train/127.5 -1, X_test/127.5 -1

The size of each data point (MNIST image) is 28×28. While this format will be useful when using CNNs, we will vectorise the datapoints for visualisation and preliminary questions. **Vectorize dataset from task 1 for perceptron visualizations:**

- Vectorize dataset from task 1 for perceptron visualizations
  - nb_features = np.prod(X_train.shape[1:])
    X_train.resize((n_train, nb_features))
    X_test.resize((n_test, nb_features))

Task 1: Data visualisation (10 points)

Project the training data points in a 2-D space using PCA. Use the obtained 2-D embedding and plot the training data points with different markers or colours for each class (**you are allowed to use PCA from scikit-learn**).

- Why is PCA a good option to visualise data?
- Add plots to your report and discuss your observations.
- Which classes can be linearly separated?

Task 2. Perceptrons (10 points)

The single-layer perceptron is one of the most basic binary classifiers one can use. In this part of the assignment, you should implement an iterative algorithm to train the single-layer perceptron. As we deal with a binary classification problem, we will pick data points corresponding to classes 0 and 1 (handwritten digits). In addition, we chose our binary labels as -1 and 1, respectively. Complete the function -x point:

- Takes in an optimize function, and trains perceptrons on all digit pairs and plots all weight visualizations, error curves, and test accuracy graphs using the specified function:

  - cond = (y_train == 0) + (y_train == 1)
    binary_x_train = X_train[cond, :]
    binary_y_train = y_train [cond] *1.
    binary_y_train [binary_y_train == 0] =-1
    binary_y_train [binary_y_train == 1] =1
    binary_x_test = X_test[cond_test, :]
    binary_y_test = y_test[cond_test] *1.

Task 2.1: Complete the function 'predict' below.

- **Inputs**:

  - $x \in R^{n \times m}$, with n being the number of data points and m being the feature dimensionality.
  - $w \in R^m$, is the parameter vector we wish to learn
  - $b \in R$ is the corresponding bias outputs

- **Outputs:**

  - 'prediction'$\in R$, a vector containing prediction values associated with x

- The code given here is:
  - def predict(x, w, b):

    "Take in a array of samples to be classified x, a set of weights w, and a bias b, and return a vector of class predictions"

    ############**Complete the function x** points#########

    return (prediction)

Task 2.2: Use the function 'predict' above to implement the single-layer perceptron algorithm by completing the function 'optimise' defined below. inputs:

- Inputs:
  - $x \in R^{n \times m}$, with n being the number of data points and m being the feature dimensionality.
  - $w \in R^m$ is the initial parameter vector.
  - $b \in R$ is the initial bias value.
  - $y \in R^n$ is the training labels associated with x
- outputs:
  - w is the optimised parameter vector.
  - b the corresponding learned bias.

– *error* is the obtained classification error.

Use the learned parameters w, b (obtained via function 'optimize') and the function 'predict' to return the classification accuracy on the test data set using x_test and y_test.

- Demonstrate that your algorithm converges to a good local minima. Plot the training error curve vs the number of iterations.
- Show what feature w has been learned and discuss why? (Demonstrate w as an image with the same size as inputs).
- Repeat this training/testing procedure to classify different pairs. Report the accuracies of 5 pairs in a table and discuss why some are easier to classify than others.

- The code given here is:

  - def optimize(x, y):
    "Given x set of training data and y class labels, optimise the perceptron"
    iter = 0
    error = np.inf
    n,m = x.shape
    w = rng.random(m)
    b = rng.random()
    learn_rate = 0.1
    error_list = []
    while (iter<=1000) & (error > 1e-3):

    ############Complete the function x points#########

    return w, b, error_list

Task 3: Multi-Layer Perceptron (10 points)

Multi-layer perceptron (MLP) is a fully connected deep (more than one hidden layer) network. In this part of the assignment, we will implement 2 hidden layers MLP with rectified linear unit (ReLU) activations. We will train the model via **ADAM optimiser** over a cross-entropy loss function.

First, we will convert our label vectors to matrices via one-hot encoding (e.g., y=2 would become [0,0,1,0,0,0,0,0,0,0]). This can be done using the commands below:

- Prepare Datasets:
  - (X_train, y_train), (X_test, y_test) = mnist.load_data()
    X_train = X_train/255
    y_train = np.eye(10)[y_train]
    X_test = X_test/ 255
    y_test = np.eye(10)[y_test]

Task 3.1:

- Create an MLP neural network architecture of the form [784,1000,1000,10]

Each integer represents the number of neurons in a given layer of the MLP, while the length of the vector defines the number of layers accordingly. Define the learning loss to be a cross-entropy loss for classification. Train this model using MNIST training data while choosing appropriate learning parameters for the ADAM optimiser including, batch size, e.g., 50; learning rate, e.g., 0.001; the number of epochs, e.g., 10.

- Report the classification accuracy obtained after training is completed on both train and test data.

- Further using visualisation tools (e.g., Tensorboard is one option) plot the training and testing curves, that are the training and testing data's classification accuracy rates vs the iterations/epochs of the training algorithm (ADAM). Include a figure of these curves in your report and discuss your observations about the learning curve behaviours.

Hint: To obtain the probabilities, you must normalise your MLP outputs so that their sum equals one. This is done using a final softmax layer before feeding the model outputs to the cross-entropy loss.

Task 3.2: Discuss how the number of layers/parameters affects the classification accuracy.

- Train four different networks with more hidden layers, for example, 3, 4, 5 and 7 hidden layers (the choice is yours here to make a good conclusion). Choose an appropriate width, i.e., number of neurons per layer, to achieve good accuracy and feasible training time.
  - How do they compare to the former MLP you implemented?
  - How many parameters (weights/biases) do these models have?
- Compare the classification accuracies of these networks with the previous MLP.
- By plotting a graph or in a table, show the accuracy vs depth vs complexity (number of parameters) of all five trained MLPs with different depths/widths.
  - Discuss the results, i.e., how the number of layers/parameters affect the classification accuracy, and provide a conclusion.

Task 4: Convolutional Neural Network (10 points)

Now that we have implemented the MLP algorithm, it is time to see how it would compare to a Convolutional neural network (CNN). CNNs leverage dependencies between neighbouring pixels, making them more efficient and lightweight compared to their fully connected counterparts.

- Load the data
  - (x_train, y_train), (x_test, y_test) = mnist.load_data()
    n_train = x_train.shape[0]
    n_test = x_test.shape[0]
    print(f"Training images {n_train}, Test images {n_test}")

Task 4.1: Create a CNN architecture of the shape [32, 64, 128] where 32, 64, 128 represent the number of convolutional filters for each hidden layer. We will use a kernel size of size 4×4. Use a stride of 1 in the first convolutional layer, followed by a stride of 2 for the following layers (a stride of two helps downsampling without requiring the use of pooling layers). Vectorise the obtained output using, e.g. tf.layer.flatten, and end the model with a fully connected layer of 10 neurons. Use ReLU as the nonlinear activation for the hidden layers.

Similar to in the last exercise, define the learning loss to be a cross-entropy loss for classification. Train this model using MNIST training data, choosing appropriate learning parameters for the ADAM optimizer: batch size, learning rate, and number of epochs.

- Report the classification accuracy obtained after training was completed on both train and test data.
- Further using visualisation tools (e.g. Tensorboard is one option) plot the training and testing curves, that are the training and testing data's

classification accuracy rates vs the iterations/epochs of the learning algorithm (ADAM). Include a figure of these curves in your report and discuss your observations about the learning curves' behaviours

Hint: To obtain the probabilities, you must normalise your CNN outputs so that their sum equals one. This is done using a softmax function/layer before feeding the model outputs to the loss.

Hint: CNNs leverage dependencies between neighbouring pixels, however this information is partially lost when we vectorised our images in the last exercise. For training CNNs, we will need to recover the initial shape of the images N×28×28, e.g., using the commands below:

- edge = int (np.sqrt (nb_features))
  x_train.resize ((n_train, edge, edge))
  x_test.resize ((n_test, edge, edge))

Task 4.2: This task is similar to Task 3.2, discussing how the number of layers and parameters affect CNN classification accuracy and how CNN results compare to MLPs.

- By plotting a graph or a table, show the accuracy vs depth vs complexity (number of parameters) of the CNNs. For this part, you need to train four additional CNNs of different depths and widths (again, your choice) and report the results in a table. Discuss the results and provide a conclusion.
- In addition, discuss and analyse the differences in terms of performance, number of model parameters (i.e., weights/biases) and training/testing times between CNNs and MLPs. Provide a conclusion.

(For these discussions, you should compare your results in Table 4.2 to Table 3.2)

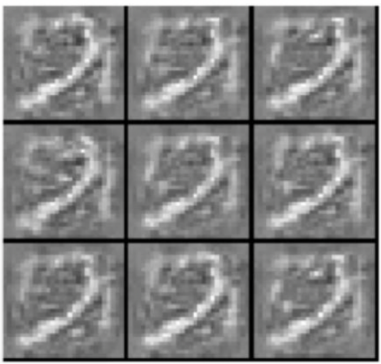
Task 5. Visualising CNN outcomes (10 points)

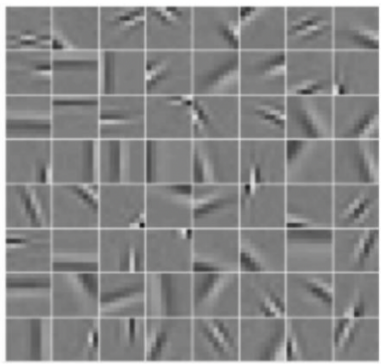
It is sometimes useful to visualise what kind of filters a CNN has learned. One way to do so is to plot each filter of size [kernel_size × kerne_size].

- Once your CNN (in Question 4) is trained, access its filters via, e.g., 'tf.get_collection' and plot them on a grid for each layer. What patterns do you observe, and why?
- In addition, plot the activations of each layer (i.e., the output of each conv layer in your CNN) for two images chosen from digit-classes '2' and '9'. Discuss your observations.
- Create deep dream images for digit classes 2 and 9 and show these in your report. Discuss how each deep dream image shows the model's sensitivity to the patterns that appear in the input images relative to each output class (category).

Hint: The plotted feature maps and learned filters should be plotted like in the images below, using a grid.

(Image description (see image below): The feature map (on the left) comprises a 3-by-3 grid. Each grid cell has the same swirling shades of grey and white image. Black lines clearly separate the cells. The learned filter (on the right) comprises an 8-by-8 grid. Each cell has a small, blurred, stripy section in shades of grey and white on a uniform grey background. The stripy section is apparent in each cell but to varying degrees, angles and positions.)





Task 6: Multi-task Learning (20 points)

This exercise concerns the practice of multi-task learning (MTL). The aim of multi-task learning is to leverage two (or more) related tasks in the learning process with the hope that learning one task aids performance in learning the other task(s) and thus improves predictive power for at least one (ideally all) of the tasks.
There are two distinct flavours of MTL: hard parameter sharing and soft parameter sharing. We will be focusing on the former in this question. Hard parameter sharing occurs when two tasks share a common network which then splits into task specific paths (e.g., a series of convolutional layers with two paths of dense layers for two separate tasks).

In this exercise, you will explore the FASHION MNIST dataset and be coding up your own MTL model, and consider the pros and cons of MTL compared to single task learning.
We have written a code to pre-load this dataset and split it into two related tasks for you:

- Task 1 - Clothing item 10 class classification (e.g., shoes, t-shirts etc.) across 10 groups $y \in R^{10}$
- Task 2 - Clothing group three-class classification - predicting whether a viewed clothing image belongs to one of three groups $y \in R^3$
  - These groups are shoes (Sandal, Sneaker and Ankle Boot), Gendered (Dress, Shirt and Bag) and Uni-Sex (T-shirt, Trouser, Pullover and Coat).

The code given here is:

- import keras.datasets.fashion_mnist as fashion_minst
  from keras.utils import to_catergorical

```
def load_data():


  - # train_X: (60000, 28)
    # train_y:(60000,)
    # test_X: (10000,28,28)
    # train_y: (10000,)
    (X_train, y_train_1), (X_test, y_test_1) = fashion_mnist.load_
    data()
    n_class_1 = 10
    # map to new label
    y_train_2 = list(0 if y in [5, 7, 9] else 1 if y in [3, 6, 8] else 2 for
    y in y_train_1)
    y_test_2 = list(0 if y in [5, 7, 9] else 1 if y in [3, 6, 8] else 2 for
    y in y_test_1)
    n_class_2 = 3
    # train_X:(60000, 28, 28, 1)
    # test_X:(10000, 28, 28, 1)
    # train_y: (60000, n_class= 10)
    # test_y: (10000, n_class = 3)

    X_train = np.expand_dims(X_train, axis=3)
    X_test = np.expand_dims(X_test, axis=3)
    y_train_1 = to_categorical(y_train_1, n_class_1)
    y_test_1 = to_categorical(y_test_1, n_class_1)
    y_train_2 = to_categorical(y_train_2, n_class_2)
    y_test_2 = to_categorical(y_test_2, n_class_2) return X_train,
    y_train_1, y_train_2, X_test, y_test_1, y_test_2


X_train, y_train_items, y_train_groups,
X_test, y_test_items, y_test_groups = load_data()
```

Task 6.1: In this question, you will construct two separate CNN classifiers of identical structure (except their output dimensions), one for each of the two tasks. In other words, create a network for item classification and a network for item group classification.

The CNNs (for the sake of convenience), we will use the same CNN filters as Question 4 [32,64,128]. However, our kernel size will be 3×3 and a stride of 1 for all convolutional layers. Maxpooling layers will also need to be implemented after the first and second convolutional layers. These maxpooling layers have a kernel size of two and a stride of 2.
After the final convolution, flatten the outputs and pass them to dense layers [3136,1024,100, N] where N is the number of outputs required (10 for Task 1 or 3 for Task 2).

- Train these individual models for each task and report the test classification accuracy obtained for each task. The training uses ADAM; batch size, e.g., 10; learning rate, e.g., 0.001; the number of epochs, e.g., 5)

Task 6.2 - Building a MTL Network. In this question, you will build one MTL model for the two MNIST fashion tasks. We will make use of data as follows:

- x_train, the input training data/images.
- y_train_1, the fashion labels for task 1 (Fashion Item classification).
- y_train_2, the labels for task 2 (Fashion Group classification).

Our MTL architecture will be comprised of a shared CNN backbone of three convolutional layers and a single shared dense layer with pooling between the first two pairs of convolutions. The output of the shared dense layer is passed to two series of task-specific dense layers, one for each of the two tasks. The architecture is as follows:

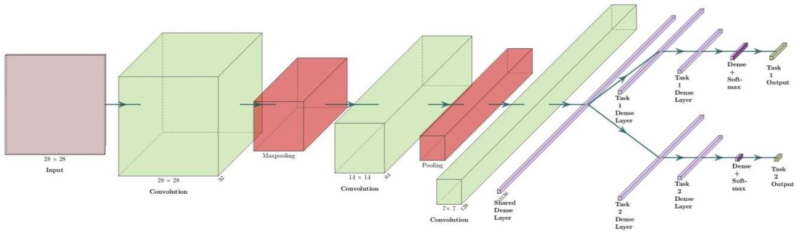
- Shared convolutional layers [32,64,128] with max pooling after the first and second conv layers:
  - kernel size (3×3) for conv and (2×2) for max pool.
  - stride 1 for conv and 2 for max pooling.
- Flatten.
- Shared Dense Layer [3136] - the outputs of which are passed to the two task-dense layers.
- Task 1 Dense Layers [1024,100,10] - 10 is the dimenson of the logits/preds.
- Task 2 Dense Layers [1024,100,3] - 3 is the dimenson of the logits/preds.
- Task 1 Activation Layer - as earlier, we use softmax.
- Task 2 Activation Layer - as earlier, we use softmax.

This architecture is illustrated in the figure below minus the flattening layer:

Figure 5 features a horizontal flow chart made up of fifteen steps. The flow chart starts on the left. It has a line of right-pointing arrows which run through seven steps before splitting into two parallel sections.

Description of the flow chart:

- The main section starts with a grey square labelled 'Input', and below it is labelled '28×28'.
- Following 'Input' is a green cube labelled 'Convolution' and also '28× 28' beneath its front face, and at the longest side, it is labelled '32'.
- 'Convolution' is followed by a smaller red cuboid labelled 'Maxpooling'.



- 'Maxpooling' is followed by a longer green cuboid labelled 'Convolution' and '14× 14' is beneath its front face and at the longest side, it is labelled '64'.
- 'Convolution' is followed by a shorter/smaller red cuboid labelled 'Pooling'.
- 'Pooling' is followed by the longest green cuboid labelled 'Convolution' and '7× 7' beneath its front face and at the longest side it is labelled '128'.
- The last step before the flow splits is a very thin, long cuboid labelled 'Shared dense layer' and by the longest side, it is labelled '3136'.
- The top flow following the split starts with a thin, shorter purple cuboid labelled 'Task 1 dense layer'.
- The 'Task 1 dense layer' is followed by another thin, and even shorter purple cuboid labelled 'Task 1 dense layer'.
- The second 'Task 1 dense layer' is followed by a shorter, darker purple cuboid labelled 'Dense + softmax'.

- The 'Dense + softmax' step is followed by a short green cuboid labelled 'Task 1 Output'. This is the last step in the top line of the flow chart.
- Now for the second parallel flow following the split after the 'Shared dense layer'. The bottom flow starts with a thin, shorter purple cuboid labelled 'Task 2 dense layer'.
- The 'Task 2 dense layer' is followed by another thin, and even shorter purple cuboid labelled 'Task 2 dense layer'.
- The second 'Task 2 dense layer' is followed by a shorter, darker purple cuboid labelled 'Dense + softmax'.
- The 'Dense + softmax' step is followed by a very short green cuboid labelled 'Task 2 Output'. This is the last step in the bottom line of the flow chart. This is the end of the description of the image.

The total loss, which is a sum of the weighted losses from tasks 1 and 2:

$$\text{loss} = \lambda \times \text{loss1} + (1 - \lambda) \times \text{loss2} \tag{1}$$

is passed to the ADAM optimiser. This uses a parameter $0 =< \lambda <= 1$ to weigh between tasks. Do not worry about the optimiser - we still only need one optimiser for joint training of the MTL network.

It is time to train your MLT model and report your tested accuracies on both tasks.

- In this task, you are required to compare the results of MTL for different values of λ. You need to compare **5** different values of λ, with the condition that three of them must be **0**, **0.5**, and **1**, whereas the remaining two can be any value of your choice. Train the 5 corresponding MLTs with the same training hyperparameters, except λ. The report should show the tested accuracies for the classifications of Task1 and Task2 for different values of λ in a table.

- Comment on your results and explain what makes the cases of λ=**0** and λ=**1** particularly special?

- Compare the performance of MTL models to single-task networks. Discuss important considerations when using MTL and its pros and cons.

**********End of the assignment brief.**********