This assignment aims to develop a deeper understanding of various nuts and bolts of the machine learning apparatus used in many computer vision tasks. In the first part of the assignment, we implement the simplest learning algorithm, a perceptron. We also explore the difficulties that arise in a practice even when data is simple, that is, 2D points and *linearly separable*. In tasks 2 and 3 of the assignment, we extend the simple perceptron to work with 3 classes instead of the two, and to solve a non-linear classification problem. Finally, we explore a real world application of deep learning in computer vision: detecting counterfeit bank notes.

Setup

As before, you will write a report and some code. While you are required to submit both, you will be graded based on your report only. We will not use deep learning frameworks (such as pytorch, tensorflow etc) for this assignment except for Task 4. Boilerplate code is provided for both Matlab and Python to get you started. The expectation is that you will implement the missing functionality to get a deeper understanding of how basic machine learning works, instead of using existing code libraries. For the final task, we will use Keras, a high-level deep learning library.

The total amount of code you need to write is not large, but you will still need to allow time for getting familiar with the different programming languages and understanding how the provided code works. We have provided reference implementation for both Matlab and Python along with setup instructions for python in the assignment package on the MyUni page. You are to use this boilerplate code to answer the first three tasks.

Task 0: Review lectures and complete quiz

Your first task is to review the deep learning lectures and make sure you are familiar with the background material needed for this assignment. We will publish a graded quiz to help you assess your grasp of the subject matter. The quiz will be conducted online via myuni, **with a deadline of Friday, June 5**, and will count towards your mark for this assignment. Please keep an eye on course announcements for further information.

Task 1: Perceptron

1. Starting from the boiler plate code provided for both Matlab and Python, implement the perceptron training algorithm as seen in the lecture. This perceptron takes a 2D point as input and classifies it as belonging to either class `0` or `1`. By changing the `num_samples` parameter, numerous sample sizes can be generated for both classes.

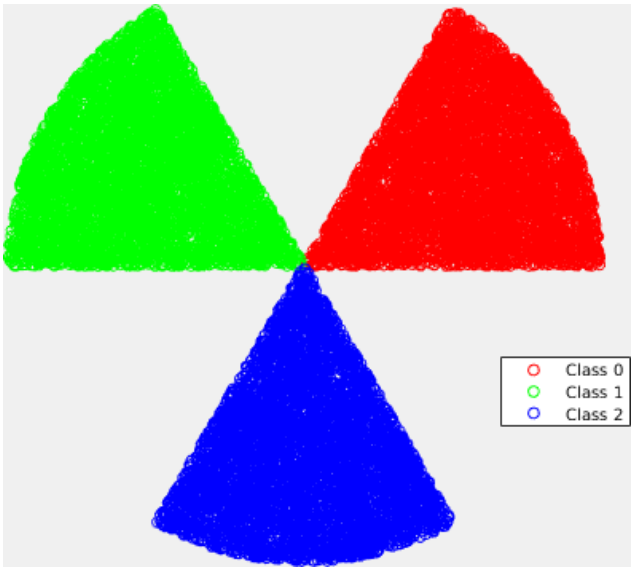

Figure 1: Dataset containing three classes

2. The code plots the ground-truth boundary between the two classes. We know this is the ground-truth because this is the boundary from which we generated data. However, even though the estimated line can correctly classify all the samples, it might not agree with the ground-truth boundary. In the report, discuss why this happens. What is the potential disadvantage of such a decision boundary? (Hint: behaviour on unseen data).

3. In some training sessions, you might observe the boundary oscillating between two solutions and not reaching 100% accuracy. Discuss why this can happen and modify the training algorithm to correct this behaviour. We will call this the modified algorithm. (Hint: learning rate)

4. Random initialization causes the algorithm to converge in different number of epochs. Execute the training algorithm on sample sizes of 10, 50, 500 and 5000. Report in a table the *mean* number of epochs required to converge for both the original and the modified algorithm. Which algorithm performs better and why? Is there a clear winner?

Task 2: Multiclass classifier

Pat yourselves on the back, you have successfully trained a binary classifier. Now, its time to move to 3 classes. The dataset we are going on work on is shown in 1 and contains three classes represented by different colors.

1. Extend the algorithm developed in Task 1 to distinguish between the three classes. In

your report, discuss the modification that were needed to extend the functionality to 3 classes. (Hint: One vs. All classification)

2. By modifying the function `compute_accuracy`, plot the accuracy over time for the training data.

3. Visualize the decision boundaries on the given dataset by implementing the draw() function for the modified algorithm and include it in your report. Based on your observation of the decision boundaries, discuss why a linear classifier is still the correct choice for this dataset.

Task 3: Backpropagation

Lets consider the output of an XOR gate for a two-input system. The output is `1` only when either of the inputs is `1`. If both inputs are zero or both inputs are one, the output is zero. This means that the output is not linearly separable. Therefore, we are going to implement a 3 layer network to solve this problem. Starting code is provided for both python and matlab:

1. Implement the `forward()` and `backward()` functions for this 3-layer network. The forward function takes the input $\mathbf{x}$ and does the following operations :

$$\hat{y} = \sigma(\mathbf{W}_2(\sigma(\mathbf{W}_1\mathbf{x}))) \tag{1}$$

where $\sigma()$ is the sigmoid activation function. We will use squared error as the loss which is defined by

$$L = \sum_{i}^{n} ||y - \hat{y}||^2 \tag{2}$$

where $\hat{y}$ is our estimated output and y is the ground truth value. The `backward()` function propagates gradients (backprop) and updates weights. In your report, include the graph of the loss against the number of epochs. You can refer to `reading_material.pdf` in the assignment package on how to compute the required gradients.

2. (postgraduates) One of the import design choices is the number of neuron in the hidden layer. Comment on the the performance of the network when the number of hidden neuron are 2, 4, 10, and 50. (You can control this by changing the corresponding variable in the code). In your experiments, what is the minimum number of neurons needed in the hidden layer for successful training? What happens when there are too many neurons in the hidden layer?

Task 4: Detecting fake bank notes with Keras

Money is one of the driving forces of this world and can lead to some people attempting to printing some of it on their own at home instead of the Royal Mint[1]. This is not good. You have been tasked with employing your newly learned machine learning skills to tell real notes apart from fake ones.

Instead of dealing with images directly, some features have already been extracted by the forensic department. For feature extraction, high resolution images of the banknotes were taken and Wavelet transform was applied to them to extract the *Variance*, *Skewness*, *Kurtosis* and *Entropy* of the resulting wavelet coefficients. (The wavelet bit is not important, what is important is we have features to work with). Additionally, class labels 0 (fake) and 1 (real) are also included in the data given to you. This is so that we can learn a model on this data and apply it to notes in the future.

Your task is to design a network for this binary classification task. You are free to choose the complexity of the network. To make things like back-propagation easier, we are going to use Keras, a high-level machine learning library that will do most of the things for us. We just have to specify the problem and architecture is a particular way. Tutorial for Keras can be found online and on the official website.

In order to ensure that you do not overfit to the data, the department (in this case your instructor) has kept some of the original labelled data hidden and handed over only 80% of it to you. The hidden data will serve as test data for the department to ensure that you machine learning technique actually works.

For a real world application, following are the steps that would normally be taken.

Train-test split

In order to validate that your method is doing something sensible, the first step is to create a train-test split. You can divide the given data randomly into an 80-20 split, where 80% of the data is used to train you network and 20% of it is used as testing. This ensures that within the data given to you, you do not overfit to the task at hand. The train and test split for the task are provided. Use the train samples to train your network, used the test samples to only test how well your network is doing. Do not train on the test samples.

Data normalization

Different variables in the feature vector come from the different ranges as shown in Fig. 2. The first step is to normalize all the features so that the range is similar, normally between zero and 1. This is achieved by min-max normalization and the data provided to you is already normalized.

[1] Bella ciao!

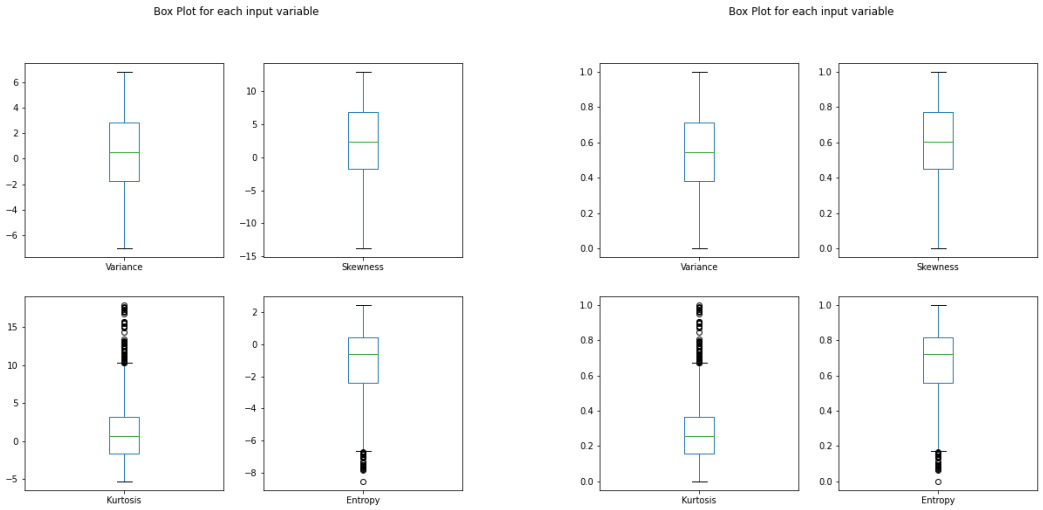

Figure 2: Box plot for each feature, please note the y-axis for each plot. **Left**: Original data, **Right**: Normalized Data

Loss function

One of the important aspects of network design is finding a suitable loss function. In our case, it is simple as we know it is a binary classification problem, so one of the losses that we saw in the lectures should be applicable here.

Network Architecture

For the current task, we are going to use a fully connected layers. The design choices in this case are the depth (number of layers) and neuron in each layer. Additionally, we have to choose the type of nonlinearity to introduce at each layer in the network.

Todo

You are provided with an ipython notebook[2] with boilerplate code to train a simple network using Keras.

The easiest way to get up and running quickly without needing to install anything is to use Google Colab[3]. This is a free service provided by Google, including access to free GPU time (when available) and learning libraries already set up. The details of how to get up and

[2] An online version hosted on Google colab can be found here: https://colab.research.google.com/drive/1GnN-AQ7pcrBpO5rWH9RyKYfRBcG3ULR4?usp=sharing

[3] http://colab.research.google.com

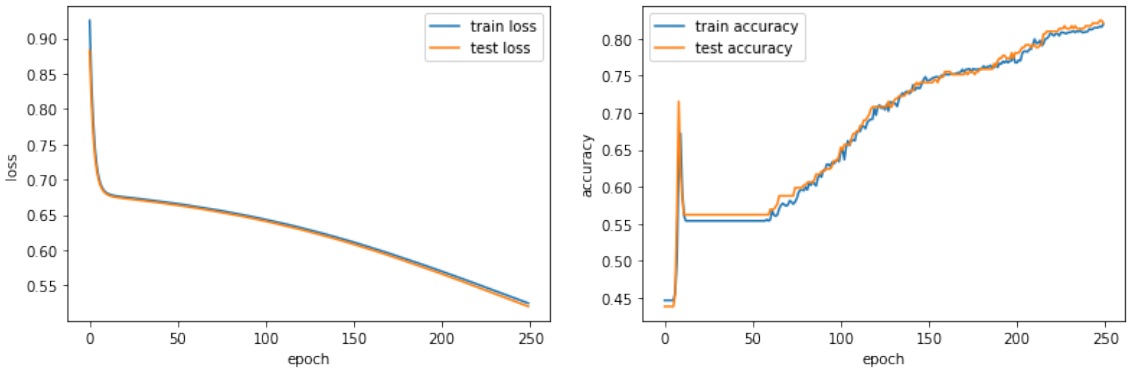

Figure 3: The evolution of the loss (left) and accuracy (right) during the training of the base model.

running can be found in `setup_instructions.pdf` in the assignment package as well as in the `Task4.ipynb`.

The base model that we are going to start from is presented in the python notebook. Specifically, the architecture consists of a multilayer perceptron with a single hidden layer containing 5 neuron and an output layer containing a single neuron. Sigmoid is used as the activation function after each layer. The default learning rate is set to 0.01 and the number of epoch to 500. We will call this the base model and its performance is shown in Fig. 3 with the final accuracy of 82.12% on the test set.

Your task is to choose the combination of various components (losses, activation function, learning rate etc) that leads to the best possible performance on the test set. Starting from the base model in each case, in your report, answer the following:

1. **Depth vs width** The current architecture has a single hidden layer with 5 neuron in it. We can add more neuron in this hidden layer (try 10 for example), this will make the layer "wider". Alternatively, we can add an additional layer. Compare and contrast the performance of these two approaches.

2. **Activation functions** Discuss if changing the activation function on the *intermediate layers* has some benefits. In the lectures we saw the ReLU learns faster than sigmoids, does this hold true? How many epochs does the ReLU based training converge in?

3. **Learning rate** How important is the learning rate? Vary the learning rate and show that at least for this problem, learning rate leads to faster convergence of the network.

4. **Number of epochs** How many epochs should the training run for? Justify your answer by making observations about convergence during your experiments.

In your submission, include your python notebook with the best performing combination of architecture, loss, activation, and learning rate.

Assessment

Hand in your report and supporting code/images via the MyUni page. Upload two files:

1. `report.pdf`, a PDF version of your report
2. `code.zip`, a zip file containing your code (Matlab `.m` files, or ipython notebooks)

The prac is worth 40% of your overall mark for the course. It is due on **Friday, June 12**.