COMP9444 Neural Networks and Deep Learning

Term 3, 2019

Project 1 - Gradient Descent and PyTorch

Due: Sunday 27 October, 23:59 pm
Marks: 16% of final assessment

This assignment is divided into three parts:

- Part 1 contains simple PyTorch questions designed to get you started and familiar with the automarking environment
- Part 2 involves creating a single-layer Neural Network (i.e. linear model) in NumPy, without using PyTorch.
- Part 3 involves implementing specific network structures to recognize handwritten Japanese Hiragana characters.

Provided Files

Copy the archive `hw1.zip` into your own filespace and unzip it. This should create a directory `hw1` with two subdirectories: `src` and `data`. Then type:

```
cd hw1/src
```

You will see three skeleton files `part1.py`, `part2.py` and `part3.py`.

Your task is to complete these files according to the specifications in this document, as well as in the comments in the files themselves. Each file contains functions or classes marked `TODO:` which correspond to the marking scheme shown below. This document contains general information for each task, with in-code comments supplying more detail. Each task in this assignment is sufficiently specified to have only one correct answer (although there may be multiple ways to implement it). If you feel a requirement is not clear you may ask for additional information on the FAQ, or the course forum.

Marking Scheme

All parts of the assignment will be automarked. Marks for each task are shown in brackets in the following table. Note that no partial marks are assigned.

Part 1:	1. [0.5] simple_addition
	2. [0.5] simple_reshape
	3. [0.5] simple_flat
	4. [0.5] simple_transpose
	5. [0.5] simple_permute
	6. [0.5] simple_dot_product
	7. [0.5] simple_matrix_mul
	8. [0.5] broadcastable_matrix_mul
	9. [0.5] simple_concatenate
	10. [0.5] simple_stack
Part 2:	1. [1] Activation
	2. [1] Forward Pass
	3. [1] Loss
	4. [1] Error
	5. [2] Backward Pass
Part 3:	1. [1] View Batch
	2. [1] Loss
	3. [1] FeedForward
	4. [2] CNN

When you submit your files through give, simple submission tests will be run to test the functionality of part 1, and to check that the code you have implemented in parts 2 and 3 is in the correct format. After submissions have closed, we will run the final marking scripts, which will assign marks for each task. We will not release these final tests, however you will be able to see basic information outlining which sections of code were incorrect (if you do not receive full marks) when you view your marked assignment.

Setting up your development environment

If you plan to write and debug the assignment on a Unix-based laptop, the following commands may help you to install the necessary software. Note that the exact commands may vary, based on your system.

1. Create a new virtual environment:

   ```
   conda create -n COMP9444 python=3.7
   ```

2. Activate it:

   ```
   conda activate COMP9444
   ```

3. Install pytorch:

   ```
   conda install pytorch torchvision cpuonly -c pytorch
   ```

4. Install everything else:

   ```
   conda install tqdm matplotlib
   ```

Another option for development is Google Colabs, which is a free service from Google that allows development in hosted notebooks that are able to connect to GPU and TPU (Googles custom NN chip - faster than GPUs) hardware runtimes. If you are having trouble getting PyTorch setup you might also want to consider this option, as the hosted environments have PyTorch preinstalled. More information and a good getting started guide is here. It is important to note this is just an option and not something required by this course - some of the tutors are not familiar with colabs and will not be able to give troubleshooting advice for colab-specific issues. If you are in doubt, develop locally.

Part 1 [5 marks]

For Part 1 of the assignment, you should work through the file `part1.py` and add functions where specified.

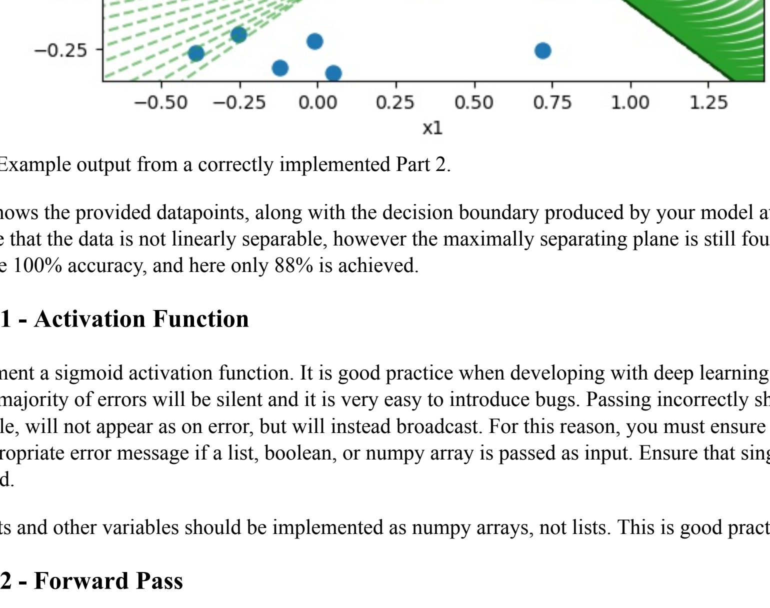
Part 2 [6 marks]

For Part 2, you will develop a linear model to solve a binary classification task on two dimensional data. The file `data/binary_classification_data.pkl` contains the data for this part. We have included the file used to generate the data as `data_generator.py`. You may examine this for your reference, or modify it if you wish to watch Gradient Decent take place on different data. Note that running this file will replace the pickle file with another stochastically generated dataset. This shouldn't cause your solution to fail, but it will cause the final output image to appear different. It is good to check that your file works with the original pickle file provided.

The file `part2.py` is the one you need to modify. It contains a skeleton definition for the custom `LinearModel` class. You need to complete the appropriate functions in this class.

You may modify the plotting method during development (`LinearModel.plot()`) - it may help you to visualize additional information. Prior to submission, however, verify that the expected output is being produced with the original, unaltered, code.

When completed, a correct implementation should produce the following image, along with model accuracies at each training step printed to `stdout`:



Example output from a correctly implemented Part 2.

This shows the provided datapoints, along with the decision boundary produced by your model at each step during training (dotted green lines). You can see that the data is not linearly separable, however the maximally separating plane is still found. For this data and model, it is impossible to achieve 100% accuracy, and here only 88% is achieved.

Task 1 - Activation Function

Implement a sigmoid activation function. It is good practice when developing with deep learning models to constrain your code as much as possible, as the majority of errors will be silent and it is very easy to introduce bugs. Passing incorrectly shaped tensors into a matrix multiplication, or example, will not appear as on error, but will instead broadcast. For this reason, you must ensure that the activation method raises a ValueError with an appropriate error message if a list, boolean, or numpy array is passed as input. Ensure that singular numpy types (such as `numpy.float64`) can be handled.

Weights and other variables should be implemented as numpy arrays, not lists. This is good practice in general when the size of a sequence is fixed.

Task 2 - Forward Pass

Implement the forward pass of the model following the structure specified. In other words, given an input, return the output of the model.

Task 3 - Loss

Implement the cross entropy loss function for the learning algorithm to minimize. See function docstring for more information.

Task 4 - Error

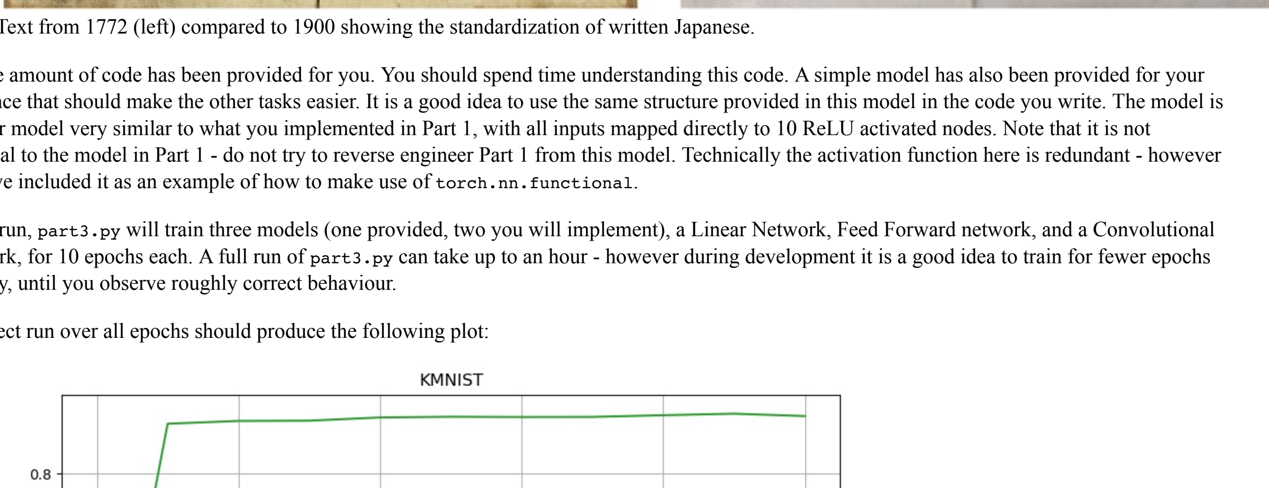
Implement an error function to return the difference between target and actual output

Task 5 - Backward Pass

Here you are required to implement gradient descent without using pytorch or autograd. Although this is difficult in general, we have tried to make it easier in this case by sticking to a single-layer network and making use of other simplifications (see function docstring for details).

Part 3 [5 marks]

Here you will be implementing networks to recognize handwritten Hiragana symbols. The dataset to be used is Kuzushiji-MNIST or KMNIST for short. The paper describing the dataset is available here. It is worth reading, but in short: significant changes occurred to the language when Japan reformed their education system in 1868, and the majority of Japanese today cannot read texts published over 150 years ago. This paper presents a dataset of handwritten, labeled examples of this old-style script (Kuzushiji). Along with this dataset, however, they also provide a much simpler one, containing 10 Hiragana characters with 7000 samples per class. This is the dataset we will be using.

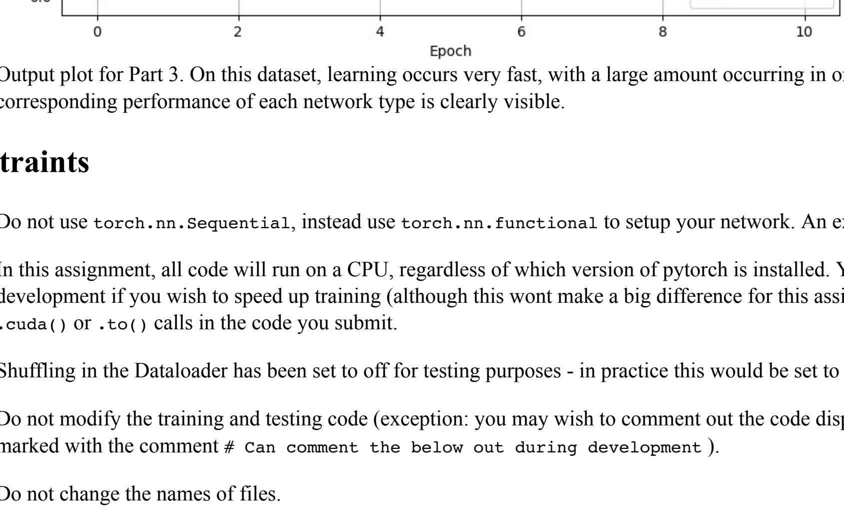


Text from 1772 (left) compared to 1900 showing the standardization of written Japanese.

A large amount of code has been provided for you. You should spend time understanding this code. A simple model has also been provided for your reference that should make the other tasks easier. It is a good idea to use the same structure provided in this model in the code you write. The model is a linear model very similar to what you implemented in Part 1, with all inputs mapped directly to 10 ReLU activated nodes. Note that it is not identical to the model in Part 1 - do not try to reverse engineer Part 1 from this model. Technically the activation function here is redundant - however we have included it as an example of how to make use of `torch.nn.functional`.

When run, `part3.py` will train three models (one provided, two you will implement), a Linear Network, Feed Forward network, and a Convolutional Network, for 10 epochs each. A full run of `part3.py` can take up to an hour - however during development it is a good idea to train for fewer epochs initially, until you observe roughly correct behaviour.

A correct run over all epochs should produce the following plot:



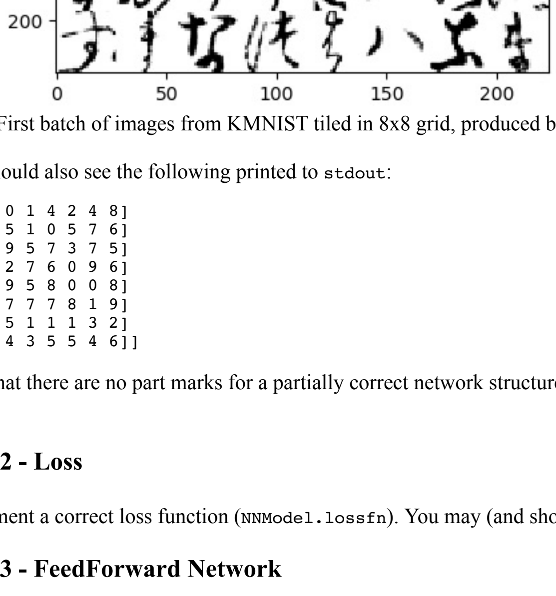
Output plot for Part 3. On this dataset, learning occurs very fast, with a large amount occurring in one epoch. The increasing capacity and corresponding performance of each network type is clearly visible.

Contraints

1. Do not use `torch.nn.Sequential`, instead use `torch.nn.functional` to setup your network. An example of a linear net is present.
2. In this assignment, all code will run on a CPU, regardless of which version of pytorch is installed. You may set code to run on a GPU during development if you wish to speed up training (although this wont make a big difference for this assignment), but ensure you do not have `.cuda()` or `.to()` calls in the code you submit.
3. Shuffling in the Dataloader has been set to off for testing purposes - in practice this would be set to True. Do not modify this.
4. Do not modify the training and testing code (exception: you may wish to comment out the code displaying the sample images. This code is marked with the comment `# Can comment the below out during development`).
5. Do not change the names of files.
6. Naming: Standard convention is to name fully connected layers `fc1`, `fc2` etc, where the number indicates depth. Similarly for convolutional layers, `conv1`, `conv2` should be used.

Task 1 - View Batch

Whenever developing deep learning models, it is absolutely critical to begin with a complete understanding of the data you are using. For this reason, implement a function that returns an 8x8 tiling of a batch of 64 images produced by one of the dataloaders, and the corresponding labels in a numpy array. Once implemented correctly, you should see he image shown below when running `part3.py`.



First batch of images from KMNIST tiled in 8x8 grid, produced by a correct `view_batch`

You should also see the following printed to `stdout`:

```
[[8 7 0 1 4 2 4 8]
 [1 1 5 1 0 5 7 6]
 [1 7 9 5 7 3 7 5]
 [6 6 2 7 6 0 9 6]
 [1 5 9 5 8 0 0 8]
 [8 6 7 7 7 8 1 9]
 [6 0 5 1 1 1 3 2]
 [2 6 4 3 5 5 4 6]]
```

Note that there are no part marks for a partially correct network structure. Do not assume inputs have been flattened prior to being fed into the forward pass.

Task 2 - Loss

Implement a correct loss function (`NNModel.lossfn`). You may (and should) make calls to PyTorch here. See the comment for further information.

Task 3 - FeedForward Network

Implement a feedforward network according to the specifications in the accompanying docstring.

Task 4 - Convolutional Network

Implement a convolutional network according to the specifications in the accompanying docstring.

Submission

You should submit by typing

give cs9444 hw1 part1.py part2.py part3.py

You can submit as many times as you like - later submissions will overwrite earlier ones. You can check that your submission has been received by using the following command:

9444 classrun -check

The submission deadline is Sunday 27 October, 23:59. 15% penalty will be applied to the (maximum) mark for every 24 hours late after the deadline.

Additional information may be found in the FAQ and will be considered as part of the specification for the project. You should check this page regularly.

General advice

1. We will be using PyTest to automatically grade submissions. While you don't have to write your own tests, doing so will allow you be sure certain sections are implemented correctly. You can use any tooling you would like for this. Make sure not to submit your test files.
2. It is possible to have the correct output when running the files with incorrect or incomplete implementations that will not receive full marks. You should rigorously test your code based on the specifications listed here, as well as within the provided file.
3. Try not to over-engineer a solution. In general, most of the methods that are required to be implemented can be done in a few lines. If you find yourself writing > 50 lines of code, you are almost certainly off track. Step back and rethink what is really required.
4. Address the failing tests in order - if there is something preventing you're model from being loaded, this will also cause all subsequent tests to fail. Once the model is loaded successfully, these other tests may pass.
5. Ensure that you are passing submission tests early, as if a submission cannot be run, it will receive 0 marks for that part. There will be no special consideration given in these cases. Automated testing marks are final. "I uploaded the wrong version at the last minute" is not a valid excuse for a remark. For this reason, ensure you are in the process of uploading your solution at least 2 hours before the deadline. Do not leave this assignment to the last minute, as it is likely that close to the deadline, the wait time on submission test results will increase.

EXTRA CHALLENGE: You might find it interesting to try Part 3 on the full dataset. This contains many additional challenges such as class imbalances that will need to be addressed. For good accuracy you will also need a much more complex network (i.e. 10's of hidden layers - a good starting point is a Resnet architecture). There is no extra marks for this, but if you get something interesting going please come to the consultations and show one of the tutors, or email the course admin (`alex.long@unsw.edu.au`).

Plagiarism Policy

Group submissions will not be allowed for this assignment. Your program must be entirely your own work. Plagiarism detection software will be used to compare all submissions pairwise and serious penalties will be applied, particularly in the case of repeat offences.

DO NOT COPY FROM OTHERS; DO NOT ALLOW ANYONE TO SEE YOUR CODE

Please refer to the UNSW Policy on Academic Integrity and Plagiarism if you require further clarification on this matter.

Good luck!