COSC401-2019S2 - Assignment 2

Due date: Friday 18 Oct 2019 at 5pm

General notes and instructions

- This assignment is marked out of 10. It contributes 22.5% towards your final grade.
- The assignment must be submitted electronically on Learn.
- You must submit one (and only one) document which has to be an IPython notebook (`.ipynb`) containing all your code, graphs, and discussions.
- Do not submit any data file. If a question requires you to generate some data, it must be done by a program and the program must be included in the IPython notebook.
- You can assume that packages such as `torch`, `torchvision`, `numpy`, `sklearn`, `statsmodels`, `seaborn`, `graphvix`, etc are installed on the machine that will be used to re-run your programs.
- For all the questions, you must include the related programs in the notebook.
- The readability and style of the document (programs, outputs, plots, discussions, etc) are all important and will have allocated marks.
- At no point in this assignment you are required to write lengthy descriptions. Keep your text succinct but rich and meaningful. Think of using tables, figures, or equations to express your thoughts and ideas.
- Set the random seed at the beginning of the document so the results can be reproduced.
- What you submit must be your own work, however, you are allowed to reuse code or content from publicly available and indexed sources as long as you cite the said source if you use more than a few lines of code or content. In such cases, you must clearly identify which parts are your work.
- You are not allowed to give or take (directly or indirectly) code or content to or from another student in this classes or other classes if you or someone else will be receiving credit for it.

- Many recent advances in machine learning are based on tensor operations and deep neural networks. This assignment covers some of these topics. While tensors are primarily designed for large-scale computing—that is, large data sets and GPUs—this assignment is scaled down so that it can run on a modest CPU.

1 Gradient-based Learning with Tensors [4 marks]

This task has two objectives: i) becoming familiar with tensor-based operations; and ii) generating synthetic data.

Testing machine learning algorithms is challenging. For most learning problems, we do not know the true distribution from which the data is coming from. When devising a new algorithm or combining existing ones, if the algorithm produces a model that has a poor performance on test data, one needs to know whether the input features are uninformative, the hyper-parameters are set incorrectly, or whether there is an inherent bug in the new algorithm.

One way of gaining confidence in the correctness of a learning system is to pass it data for which we know the correct distribution (and therefore we know what models or level of performance to expect) and then see if the algorithm produces the expected result.

In this task you should perform a regression task with tensor operations on synthetic data generated by yourself.

You must first come up with a function that maps $\mathbb{R}^m$ to $\mathbb{R}^n$ with both m and n parametrically defined at the beginning of your program and set to values greater than 3. Provide the equation of the function(s). Add some noise to the output to make the problem more interesting. However, do not add too much noise so that the underlying functions are still recognisable. Also be mindful of the expected value of the noise function.

Pass the generated data to your own implementation of linear regression algorithm to see if it produces, more or less (depending on how much noise was added), the expected model.

Further remarks and requirements:

- You are not allowed to use any regression libraries. You must use only tensor operations and automatic gradient calculations to write your loss function and to write your own optimisation procedure based on gradient descent. Do not use any sub-modules in PyTorch (e.g. neural networks or optimisation). Everything must be by tensor operations.
- When possible, avoid using loops; instead use tensor operations to act on multiple values at once.
- Avoid using matrix inverse operators (use only gradients);
- When performing updates, be mindful of gradient tracking.
- Make sure you generate enough data.

- Choose appropriate learning rate.
- Choose an appropriate termination criteria for learning.

2 Transfer Learning [4 marks]

In this task you must do a small-scale transfer learning. The idea is to train a CNN on a simple problem and then see if any of the knowledge learnt by the network can be transferred to another network that must be trained on a slightly larger (and more difficult) problem.

You must use neural network and optimisation modules in PyTorch. You have to define two network classes: one to learn on `torchvision.datasets.MNIST` and the other on `torchvision.datasets.CIFAR10`. Train the first network to the point that it performs well. Also train the second network and observe the changes in total loss over time (epochs).

We want to see if by copying some layers of the trained MNIST network to the layers of CIFAR10 network (before it is trained), we can give a boost to the training of the second network.

You will have to consider aspects such as:

- which layer(s) are good candidates to be transferred (very briefly discuss);
- how to deal with the fact that these two data sets have different dimensions;
- how to copy objects across (keeping dimensions and gradient tracking in mind);
- a way of presenting your findings (graphs, tables, etc).

3 Small Research/Thinking Activity [2 marks]

For this part, you will have to do either option A or B (not both). Option B is significantly more challenging than option A, but it is also more interesting. Please note that in order to do option B, you need to register your interest by the said deadline. If you do both options (for example by choosing option B first and then deciding to switch to A) you will need to choose which option you want to be marked and receive credit for.

3.1 [Option A] Cost-Sensitive Learning

Suppose you are given a k-by-k cost matrix C for a classification task which has k classes. The element $C_{i,j}$ is the cost of classifying an instance of class j as class i. You want to train a classifier that minimises the expected cost of predictions. Do the following:

1. Write a pseudocode or Python code for a loss function that, once optimised over all examples, achieves minimum expected cost of prediction.
2. Assuming that a learning algorithm which minimises the classification error is given to you (and that you cannot supply your own loss function to the algorithm), think and write about a different way of achieving minimum expected cost of prediction.

Additional notes or requirements:

- Consider cases with $k \geq 2$ and briefly discuss if the two methods are scalable.
- You do not have to use tensors in this questions.
- If you choose to write in Python, you do not have to run your program on any particular input. You can also use "pseudo-Python" if you wish.

3.2 [Option B] Auto Encoders and Deep Q Learning

In order to do this task you must:

- have a good background in one the deep learning frameworks;
- have access to a good GPU;
- have some extra time to do reading, thinking, and experimentation;
- be interested in giving a short demo to the class; and
- register your interest by Monday 30 Sep.

In this task you have to apply an auto-encoder to frames generated by a physics engine involving a very simple environment and a sequence of actions (for example the interaction between a racket and a ball) and use the learnt representation with reinforcement learning to control some aspects of the environment (e.g. return a ball on a desired trajectory or to a target).

This is an open-ended task. You will decide what exactly needs to be done. You will have to produce a very short report at the end describing your work and program. You will be expected (but not required) to provide a demo to the class if your results are interesting.