# COMP0090: Introduction to Deep Learning

# Assessed Coursework 1 2021-22

Available on 26th November 2021
Submission before 16:00 (UK time), 16th December 2021, on Moodle

## Introduction

This is the first of two assessed coursework. This coursework accounts for 50% of the module with three independent tasks, and for each task, a *task script* needs to be submitted with other supporting files and data. No separate written report is required.

There are hyperlinks in the document for further reference. Throughout this document, various parts of the text are highlighted, for example:

*Class names are highlighted for those mandatory classes that should be found in your submitted code.*
*Function names are highlighted for those mandatory functions that should be found in your submitted code.*
*Printed messages on terminal when running the task scripts.*
*Visualisation saved into PNG files with task scripts.*
*[5]: square brackets indicate marks, with total marks being 100, for 50% of the module assessment.*
*"filepath.ext": quotation marks indicate the names of files or folders.*
commands: commands run on bash or Python terminals, given context.

The aim of the coursework is to develop and assess your ability *a)* to understand the technical and scientific concepts behind deep learning theory and applications, *b)* to research the relevant methodology and implementation details of the topic, and *c)* to develop the numerical algorithms in Python and one of the deep learning libraries TensorFlow and PyTorch. Although the assessment does not place emphasis on coding skills and advanced software development techniques, basic programming knowledge will be taken into account, such as the correct use of NumPy arrays, tensors – as opposed to, for example, unnecessary for-loops, sufficient commenting and consistent code format. Up to [20%] of the relevant marks may be deducted for good programming practice.

Do NOT use this document for any other purposes or share with others. The coursework remains UCL property as teaching materials. You may be risking breaching intellectual property regulations and/or academic misconduct, if you publish the details of the coursework or distribute this further.

## Conda environment and other Python packages

No external code (open-source or not) should be used for the purpose of this coursework. No other packages should be used, unless specified and installed within the conda environment below. Individual tasks may have specific requirement, e.g. only TensorFlow or PyTorch, as opposed to NumPy for example, can be used for certain function implementation. Up to [100%] of the relevant marks may be deducted for using external code. This will be assessed by, on the markers' computers, running the submitted code within a conda environment created as follows:

```
conda create -n comp0090-cw1 python=3.9 tensorflow=2.4 pytorch=1.7 torchvision=0.8

conda activate comp0090-cw1

conda list
```

Use the above command to see the available libraries for this coursework after activating comp0090-cw1. Make sure your OS is up-to-date to minimise potential compatibility issues.

### TensorFlow or PyTorch

You can choose to use either TensorFlow or PyTorch, but not both of them in this coursework, as it is designed to have a balanced difficulties from different tasks.

### Working directory and task script

Each task should have a task folder, named as "task1", "task2" and "task3". A Python task script should be a file named as "task.py", such that the script can be executed on the bash terminal when the task folder is used as the current/working directory, within the conda environment described above:

```
python task.py
```

It is the individual's responsibilities to make sure the submitted task scripts can be run, in the above-specified conda environment. Even for the data/code available in module tutorials, copies or otherwise automated links need to be provided to ensure a standalone executability of the submitted code. Care needs to be taken in correct use of relative paths, as it has been a common issue. Jupyter Notebook files are NOT allowed. Up to [100%] of the relevant marks may be deducted if no runnable task script is found.

### Printing and visualisation

Summarising and communicating your implementation and quantitative results is being assessed as part of the module learning outcome. Each task specifies relevant information and messages to be printed on terminal, which may contain description, quantitative summary and brief remarks. The printed messages are expected to be concise, accurate and clear.

When the task requires visualising results (usually in the form of images), the code should save the results into a PNG file in the respective working directory. These PNG files should be submitted with the code. This is for compatibility with those environments that do not support graphics, such as WSL or remote setups. Please see examples in the module repository using Pillow. Please note that matplotlib cannot be used in the task scripts. Up to [50%] of the relevant marks maybe deducted if this is not followed.

### Design your code

The functions/classes/files/messages highlighted (see Introduction) are expected to be found in your submitted code, along with the task scripts. If not specifically required, you have freedom in designing your own code, for example, data type, variables, functions, scripts, modules, classes and/or extra results for discussion. They will be assessed for correctness but not for design aspects.

## The checklist

This is a list of things that help you to check before submission.

- ✓ The coursework will be submitted as a single "cw1" folder, compressed as a single zip file.
- ✓ Under your "cw1" folder, you should have three subfolders, "task1", "task2" and "task3".
- ✓ The task scripts run without needing any additional files, data or customised paths.
- ✓ All the classes and functions colour-coded in this document can be found in the exact names.
- ✓ Check all the functions/classes have docstring on data type, size and what-it-is for input arguments, outputs and a brief description of its purpose.

DRAFT

# Task 1 Stochastic Gradient Descent for Linear Models

This task needs to be implemented entirely using TensorFlow/PyTorch, without using NumPy.

- Implement a polynomial function polynomial_fun, that takes two input arguments, a weight vector $\mathbf{w}$ of size $M+1$ and an input scalar variable $x$, and returns the function value $y$. [3]

$$y = \sum_{m=0}^{M} w_m x^m$$

- Using the linear algebra modules in TensorFlow/PyTorch, implement a least square solver for fitting the polynomial functions, fit_polynomial_ls, which takes $N$ pairs of $x$ and target values $t$ as input, with an additional input argument to specify the polynomial degree $M$, and returns the optimum weight vector $\hat{\mathbf{w}}$ in least-square sense, i.e. $\|t-y\|^2$ is minimised. [5]
- Using relevant functions/modules in TensorFlow/PyTorch, implement a stochastic minibatch gradient descent algorithm for fitting the polynomial functions, fit_polynomial_sgd, which has the same input arguments as fit_polynomial_ls does, with additional two input arguments, learning rate and minibatch size. This function also returns the optimum weight vector $\hat{\mathbf{w}}$. During training, the function should report the loss periodically using printed messages. [5]
- Implement a task script "task.py", under folder "task1", performing the following:
  - Use polynomial_fun ($M=3$, $\mathbf{w}=[1,2,3,4]^{\mathrm{T}}$) to generate a training set and a test set, in the form of respectively sampled 100 and 50 pairs $x, x\epsilon[-20,20]$ and observed $t$. The observed $t$ values are obtained by adding Gaussian noise (standard deviation being 0.2) to $y$. [3]
  - Use fit_polynomial_ls ($M=4$) to compute the optimum weight vector $\hat{\mathbf{w}}$ using the training set. In turn, compute the predicted target values $\hat{y}$ for all $x$ in both the training and test sets. [2]
  - Report, using printed messages, the mean (and standard deviation) in difference a) between the observed training data and the underlying "true" polynomial curve; and b) between the "LS-predicted" values and the underlying "true" polynomial curve. [3]
  - Use fit_polynomial_sgd ($M=4$) to optimise the weight vector $\hat{\mathbf{w}}$ using the training set. In turn, compute the predicted target values $\hat{y}$ for all $x$ in both the training and test sets. [2]
  - Report, using printed messages, the mean (and standard deviation) in difference between the "SGD-predicted" values and the underlying "true" polynomial curve. [2]
  - Compare the accuracy of your implementation using the two methods with ground-truth on test set and report the root-mean-square-errors (RMSEs) in both $\mathbf{w}$ and $y$ using printed messages. [3]
  - Compare the speed of the two methods and report time spent in fitting/training (in seconds) using printed messages. [2]

# Task 2 A Regularised DenseNet

For the purpose of this task, the dataset is simply split into two, training and test sets, as in the tutorial.

- Adapt the Image Classification tutorial to implement a new network DenseNet3, with the following:
  - Contain a member function dense_block, implementing a specific form of DenseNet architecture, each contains 4 convolutional layers. [3]

DRAFT

  - Design and implement the new network architecture to use 3 of these dense blocks. [4]
  - Summarise and print your network architecture, e.g. using built-in summary function. [1]
- Implement a data augmentation function cutout, using the Cutout algorithm.
  - Use square masks with variable size and location. [2]
  - Add an additional parameter s, such that the mask size can be uniformly sampled from [0, s]. [3]
  - Location should be sampled uniformly in the image space. N.B. care needs to be taken around the boundaries, so the sampled mask maintains its size. [3]
  - Visualise your implementation, by saving to a PNG file "cutout.png", a montage of 16 images with randomly augmented images that are about to be fed into network training. [3]
  - Add Cutout into the network training. [3]
- Implement a task script "task.py", under folder "task2", completing the following:
  - Train the new DenseNet classification network with Cutout data augmentation. [3]
  - Run a training with 10 epochs and save the trained model. [3]
  - Submit your trained model within the task folder. [2]
  - Report the test set performance in terms of classification accuracy versus the epochs. [2]
  - Visualise your results, by saving to a PNG file "result.png", a montage of 36 test images with captions indicating the ground-truth and the predicted classes for each. [3]

## Task 3 Ablation using Cross-Validation

Again, using the Image Classification tutorial, this task investigates the impact of one of the following three modifications to the original network, using cross-validation. To evaluate a modification, an ablation study can be used by comparing the performance before and after removing the modification.

- Difference between training with and without the Cutout data augmentation algorithm implemented in Task 2.
- Difference between using SGD with momentum (as in "train_pt.py") and Adam optimiser (as in "train_tf.py").
- Difference between using ReLU and leaky ReLU (with a negative slope alpha=0.1), as activation functions throughout the network.
- Indicate your choice. [1]

- Implement a task script "task.py", under folder "task3", completing the following:
  - Split the data into development set and holdout test set. [1]
  - Implement a 3-fold cross-validation scheme, using the development set. [5]
  - Print data set summary every time the random split is done. [2]
  - Design at least one metric, other than the loss, on validation set, for monitoring during training. [5]
  - Run the cross-validation scheme for each of the two networks or training strategies (with and without the one modification). [6]
  - Report a summary of loss values, speed, metric on training and validation. [4]
  - Train two further models using the entire development set and save the trained models. [4]
  - Submit these two trained models within the task folder. [2]
  - Report a summary of loss values and metrics on the holdout test set. Compare the results with those obtained during cross-validation. [5]

DRAFT