COMP0197: Applied Deep Learning

Assessed Component 1 (Individual Coursework) 2023-24

Submission before 16:00 (UK time), 21st March 2024 (subject to change), 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 substandard 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 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. This will be assessed by running the submitted code on the markers' computers, within a conda environment created as follows, for either TensorFlow or PyTorch. Make sure your OS is up-to-date to minimise potential compatibility issues.

```
conda create -n comp0197-cw1-tf pillow=10.2 pip=19.3 && conda activate comp0197-cw1-tf && pip install tensorflow==2.13
```

```
conda create -n comp0197-cw1-pt -c pytorch python=3.12 pytorch=2.2 torchvision=0.17
```

Use one of the two for your coursework and indicate with your submitted folder name, "cw1-tf" or "cw1-pt". Use the command `conda list -n comp0197-cw1-xx` to see the available libraries for this coursework ("xx" is either "tf" or "pt"). 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. [100%] of the relevant marks may be deducted for using external code.

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 a 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 responsibility to make sure the submitted task scripts can run, in the above-specified conda environment. If using 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 was found to be one of the most common issues in the past. 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 image), the code should save the results into a PNG file in the respective working directory. These PNG files should be submitted with the code, although they can be generated by the code as well. Please see examples in the module repository using Pillow. Please note that matplotlib cannot be used in the task scripts but may be a good tool during development. 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. These will be assessed for complementing your work 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-xx" folder, compressed as a single zip file.
- ✓ Under your "cw1-xx" 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 a docstring indicating a brief description of its purpose, together with data type, size and what-it-is, for each input argument and output.

Task 1 Stochastic Minibatch Gradient Descent for Linear Models

- 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. The polynomial_fun should be vectorised for multiple pairs of scalar input and output, with the same $\mathbf{w}$. [5]

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

- 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: [15]
    - Use polynomial_fun ($M=2$, $\mathbf{w}=[1,2,3]^{\mathrm{T}}$) to generate a training set and a test set, in the form of respectively and uniformly sampled 20 and 10 pairs of $x, x\epsilon[-20,20]$, and t. The observed t values are obtained by adding Gaussian noise (standard deviation being 0.5) to y.
    - Use fit_polynomial_ls ($M\epsilon\{2,3,4\}$) to compute the optimum weight vector $\hat{\mathbf{w}}$ using the training set. For each M, compute the predicted target values $\hat{y}$ for all x in both the training and test sets.
    - 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.
    - Use fit_polynomial_sgd ($M\epsilon\{2,3,4\}$) to optimise the weight vector $\hat{\mathbf{w}}$ using the training set. For each M, compute the predicted target values $\hat{y}$ for all x in both the training and test sets.
    - Report, using printed messages, the mean (and standard deviation) in difference between the "SGD-predicted" values and the underlying "true" polynomial curve.
    - 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.
    - Compare the speed of the two methods and report time spent in fitting/training (in seconds) using printed messages.
- Implement a task script "task1a.py", under folder "task1". [10]
    - Experiment how to make M a learnable model parameter and using SGD to optimise this more flexible model.
    - Report, using printed messages, the optimised M value and the mean (and standard deviation) in difference between the model-predicted values and the underlying "true" polynomial curve.

Task 2 Data-augmented vision transformers

For the purpose of the coursework, the dataset is only split into two, training and test sets.

- Adapt the Image Classification tutorial to use a different network, VisionTransformer. You can choose any configuration that is appropriate for this application. [5]
    - TensorFlow version
    - PyTorch version
- Implement a data augmentation class MixUp, using the mixup algorithm, such that: [10]
    - Inherited from the relevant classes in TensorFlow/PyTorch is recommended but not assessed.
    - The MixUp algorithm can be applied to images and labels in each training iteration.
    - Have an input flag "sampling_method" and appropriate hyperparameters for two options:
        - sampling_method = 1: λ is sampled from a beta distribution as described in the paper.
        - sampling_method = 2: λ is sampled uniformly from a predefined range.
        - The algorithm should be seeded for reproducible results.
    - Visualise your implementation, by saving to a PNG file "mixup.png", a montage of 16 images with randomly augmented images that are about to be fed into network training.
    - Note: the intention of this task is to implement the augmentation class from scratch using only TensorFlow/PyTorch basic API functions. Using the built-in data augmentation classes may result in losing all relevant marks.
- Implement a task script "task.py", under folder "task2", completing the following: [15]
    - Train a new VisionTransformer classification network with MixUp data augmentation, for each of the two sampling methods, with 20 epochs.
    - Save the two trained models and submit your trained models within the task folder.
    - Report the test set performance in terms of classification accuracy versus the epochs.
    - Visualise your results, by saving to a PNG file "result.png", a montage of 36 test images with printed messages clearly indicating the ground-truth and the predicted classes for each.

Task 3 Ablation Study

Using the Image Classification tutorial, this task investigates the impact of the following modification to the original network. To evaluate a modification, an ablation study can be used by comparing the performance before and after the modification.

- Difference between training with the two λ sampling methods in Task 2.

- Implement a task script "task.py", under folder "task3", completing the following: [30]
    - Random split the data into development set (80%) and holdout test set (20%).
    - Random split the development set into train (90%) and validation sets (10%).
    - Design at least one metric, other than the loss, on validation set, for monitoring during training.
    - Train two models using the two different sampling methods.
    - Report a summary of loss values, speed, metric on training and validation.
    - Save and submit these two trained models within the task folder.
    - Report a summary of loss values and the metrics on the holdout test set. Compare the results with those obtained during development.