UNIVERSITY OF
CAMBRIDGE

MPhil in Data Intensive Science

Submission: 11:59pm on Friday the 4th of April

Coursework v1.0
M2: Deep Learning
Dr. M. Cranmer

The coursework will be submitted via a GitLab repository which we will create for you. You should place all your code and your report in this repository. You should write a report with up to 3000 words that describes your work. Your report should contain figures and tables to support your analysis and discussion. The report should be in PDF format, stored as a file called "report/main.pdf". You will be provided access to the repository until the stated deadline. After this you will lose access which will constitute submission of your work.

The code associated with the coursework should be written in Python and follow best software development practice as defined by the Research Computing module. This should include:

- *Writing clear readable code that is compliant with a common style guide and uses suitable build management tools.*
- *Providing appropriate documentation that is compatible with auto-documentation tools.*
- *The project must be well structured (sensible folder structure, README.md, licence etc..) following standard best practice.*
- *Uses appropriate version control best practices.*
- *Appropriate 'pyproject.toml' structure to ensure portability of the project to other computers and operating systems.*

SECTION Introduction

This coursework explores **LoRA** (Low-Rank Adaptation) applied to the Qwen2.5-Instruct class of Large Language Models (LLMs), repurposed for forecasting a predator-prey system. This relies on the observation in the paper Gruver et al. (2023) that Large Language Models can be used as time series forecasters, without further training. We will build off of this assumption, and demonstrate performance of a model that has been fine-tuned further.
You will be provided with:

- `lotka_volterra_data.h5`: time series of prey and predator populations modelled by the Lotka-Volterra equations. This dataset is described later in this document.
- A description of the LLMTIME preprocessing scheme to describe how numeric sequences should be preprocessed before tokenization, which is given later in this document. The original paper is also given (`llmtime.pdf`)
- `qwen.py`: a script that loads the Qwen2.5-Instruct model from HuggingFace into a PyTorch model.
- `lora_skeleton.py`: a partial training script where you must implement preprocessing and LoRA fine-tuning steps.
  - Note that this is just to get you started; it is not complete, and does not implement best practices.
- `qwen.pdf`: A paper describing the architecture of the Qwen2.5-Instruct model. Note that you will also need to look at the papers cited within, to understand specific model components, such as `RMSNorm`.

Questions.

1 Compute Constraint

In this coursework, you will prioritise *efficient experimentation*. An issue occuring in a previous coursework related to students executing lengthy training runs from the start, and be unable to quickly iterate on their methods. This also stretched compute resources.

For this coursework, you have a maximum of 10^{17} **floating point operations** (FLOPS), defined below, over all reported experiments. With idealized calculations and 100% usage, this would be around the total processing of *5 hours on a MacBook M1 Pro GPU*, if it is perfectly utilised. In other words, you can do this coursework on a laptop. You should not report any experiments or results that

Table 1: FLOPS Accounting for Primitives

Operation	**FLOPS**
Addition/Subtraction/Negation (float OR integer)	1
Multiplication/Division/Inverse (float OR integer)	1
ReLU/Absolute Value	1
Exponentiation/Logarithm	10
Sine/Cosine/Square Root	10

are unaccounted for in the FLOPS table. It is fine if you use less compute than the maximum allowed.

You should use Table 1 to calculate the FLOPS for operations (since actual numbers vary by hardware). For example, using the above table, a single matrix multiplication of an $m \times n$ matrix with an $n \times p$ matrix requires $m \times p \times (2n - 1)$ FLOPS, because each element of the resulting $m \times p$ matrix requires n multiplications and $n - 1$ additions. Other primitives that cannot be expressed with these can be assumed to be 0 FLOPS (such as array indexing, memory management, etc.). Ignore FLOPS associated with the tokenization step.

To further simplify things, you should only calculate the FLOPS for the forward pass, and then assume the FLOPS of backpropagation are exactly equal to 2× the FLOPS for the forward pass. If there are any ambiguities, where some operations may use special hardware-specific optimizations, you can just assume the simplest version.

At the end of your report, you should include a table summarising the FLOPS used for every experiment. This will be evaluated as part of the coursework. You should seek to do small-scale experiments, analyse the hyperparameter trade-offs, and only then move to larger-scale experiments. Showcase best practices in justifying experiments based on previous experiments or ideas covered in the course.

2 Baseline

(a) Implement the LLMTIME preprocessing scheme (described later in this document) for the time series data, and save it to the file `src/preprocessor.py`. Demonstrate best practices as defined in the course, and best software practices as defined in C1.

Share two example sequences, the preprocessed, and tokenized results (as sequences of integers, using the preprocessor you implemented, followed by mapping to Qwen2.5 tokens).

(b) Evaluate the untrained Qwen2.5-Instruct model's forecasting ability on this tokenized dataset. Showcase best machine learning practices in providing metrics and a short discussion of the untrained model's performance.

(c) Use the provided table to map each operation to its FLOPS. In words, briefly enumerate the accounting of all operations. Write a Python script to calculate the FLOPS as a function of the model's hyperparameters and input size, and whether used in a training or inference context. To simplify this process, you should use the following simplifications:

- Calculate the FLOPS for the forward pass. Assume that the FLOPS of backpropagation are exactly equal to 2× the FLOPS for the forward pass.
- Do not include the tokenization step in your FLOPS calculation, nor any operations which are performed outside of the model, like loading the data, running Python itself, etc.
- Qwen2.5-Instruct uses "Grouped Query Attention." For simplicity, assume the FLOPS for this operation are the same as a standard multi-head attention. Note that you will need to look up the SwiGLU activation, as well as RMSNorm (and decompose each into primitives).
- Ignore FLOPS associated with computing the positional encodings (but include the cost of *adding* them to the token embeddings).

Show your work. Do not worry about getting the exact numbers down to the last operation, the main goal is to show your detailed understanding of the forward pass of such a model.

3 LoRA

(a) Using the provided skeleton code, adapt the LoRA implementation to the Qwen2.5-Instruct model, by wrapping the query and value projection layers with LoRALinear layers. As is standard in LoRA, only the LoRA matrices are trained, while the base matrices are frozen.

In words, specify which parameters blocks are being tuned in this model when you train with LoRA, and how.

Train the 0.5B parameter model with a single LoRA configuration based on the default hyperparameters, for *up to* 10,000 optimizer steps, on the training set. Evaluate this model on the validation set, and compare the results to the untrained model.

Again, use best machine learning practices in providing metrics and a short discussion of the untrained model's performance.

(b) Do a search over the hyperparameters of the LoRA configuration: vary the learning rate in $(10^{-5}, 5 \times 10^{-5}, 1 \times 10^{-4})$, and the LoRA rank in $(2, 4, 8)$. Each time, train for *up to* 10,000 optimizer steps, and evaluate on the validation set.

Using the best hyperparameters you find, perform some additional experiments (up to only 2,000 optimizer steps each), exploring the effect of the context length (in $(128, 512, 768)$).

In all cases, log your results and metrics. Track the FLOPS used.

(c) Based on trends observed in the previous experiments, select a set of hyperparameters to use for the final model. Use these hyperparameters to train the model for *up to* 30,000 optimizer steps, and evaluate on the validation set. Present your results.

Marks will be awarded for a thoughtful approach to the experiments and analysis, rather than raw performance of the final model.

Conclude with **recommendations** for time-series fine-tuning under tight compute budgets. Suggest possible next steps for improved performance.

Predator-Prey Dataset Documentation

Overview

The dataset contains time series representing predator and prey population dynamics over time. Each time series represents a different system with its own parameters.

Dataset Structure

The data is provided in HDF5 format (`lotka_volterra_data.h5`) with the following organization:

- `/trajectories`: Main dataset array of shape (1000, 100, 2)
  - 1000 different predator-prey systems (first dimension)
  - 100 time points per system (second dimension)
  - 2 population variables (third dimension: `[:,:,0] = prey`, `[:,:,1] = predator`)
- `/time`: Array of 100 time points representing the observation timeline

The file can be opened using the h5py Python library:

```
import h5py
import numpy as np

with h5py.File("lotka_volterra_data.h5", "r") as f:
    # Access the full dataset
    trajectories = f["trajectories"][:]
    time_points = f["time"][:]

    # Access a single trajectory
    system_id = 0  # First system

    # First 50 time points:
    prey = trajectories[0, :50, 0]
    predator = trajectories[0, :50, 1]
    times = time_points[:50]
```

LLMTIME Preprocessing Scheme

To use Qwen2.5-Instruct for forecasting numerical time series data, you will implement a text-based numeric encoding method adapted from the LLMTIME preprocessing scheme described by Gruver et al. (2023). This preprocessing ensures the numeric sequences are suitably formatted for Qwen's tokenizer and forecasting capabilities.

Scaling and Precision Numeric values in your dataset may vary significantly. To standardize the numeric range and control token length, you should apply a simple scaling:

$$x'_t = \frac{x_t}{\alpha} \tag{1}$$

where parameter α should be chosen based on the distribution of your dataset (for example, scaling to a numeric range, such as having most data in the range 0–10). Next, round each scaled numeric value to a fixed number of decimal places, such as two or three decimal places (this is a hyperparameter), to ensure uniformity and consistent representation.

Numeric Tokenization. The LLMTIME approach emphasizes clear and tokenizer-friendly numeric sequences. Luckily for us, Qwen's tokenizer already naturally splits numeric strings into individual digits, as you can verify from the following code:

```
from transformers import AutoTokenizer

model_name = "Qwen/Qwen2.5-0.5B-Instruct"
tokenizer = AutoTokenizer.from_pretrained(model_name)

print(tokenizer("1.23", return_tensors="pt")["input_ids"].tolist()[0])
# [16, 13, 17, 18]
print(tokenizer("1 . 2 3", return_tensors="pt")["input_ids"].tolist()[0])
# [16, 659, 220, 17, 220, 18]
```

which essentially means that Qwen tokenizes the string "1.23" into four tokens: `[16, 13, 17, 18]`. It appears to also have a specific token for “_.” corresponding to `659`. Therefore, unlike models such as GPT-3 (Gruver et al., 2023), we will not insert spaces between digits.

Multivariate Time Series Encoding. The LLMTIME paper explicitly univariate time series. Since you are dealing with multivariate predator-prey data, you should adopt the following extended convention to represent multiple variables and multiple timesteps clearly:

- Separate different variables at the same timestep with a comma, “`,`”,

- Separate different timesteps with a semicolon, “;”,

Thus, a short example with two variables at three successive timesteps might look like:

```
0.25,1.50;0.27,1.47;0.31,1.42
```

In this example, each numeric value is tokenized by Qwen into separate digits and punctuation tokens, clearly separating numeric values, variables, and timesteps.

Decoding and Inference When generating predictions, your model will output tokenized numeric sequences in the same format. You must post-process the outputs back into numerical arrays by reversing the encoding steps.

END OF PAPER