Machine Learning Practical 2018/19: Coursework 2

Released: Monday 5 November 2018

Submission due: 16:00 Friday 23 November 2018

1 Introduction

The aim of this coursework is to further explore the classification of images of handwritten digits using neural networks. As in the previous coursework, we'll be using an extended version of the MNIST database, the EMNIST Balanced dataset, described in the coursework 1 spec. The first part of the coursework will concern the implementation and experimentation of convolutional networks using the MLP framework. The second part will involve exploring different convolutional network architectures using PyTorch.

In order to support the experiments you will need to run for the second part of the coursework (which will be carried out in PyTorch) we have acquired Google Cloud Platform credits which allow the use of the Google Compute Engine infrastructure. Each student enrolled on the MLP course will receive a $50 Google Cloud credit coupon which is enough to carry out the experiments required for this coursework. You will receive an email which will give you the URL you will need to access in order to request your Google Cloud Platform coupon.

As with the previous coursework, you will need to submit your python code and a report. In addition you will need to submit the outputs of test code that tests your implementation for the first part. The detailed submission instructions are given in Section 5.2 – please follow these instructions carefully.

2 Github branch `mlp2018-9/coursework_2`

The provided code and setup information for this coursework is available on the course Github repository on a branch `mlp2018-9/coursework_2`. To create a local working copy of this branch in your local repository you need to do the following.

1. Make sure all modified files on the branch you are currently have been committed (see notes/getting-started-in-a-lab.md if you are unsure how to do this).
2. Fetch changes to the upstream `origin` repository by running
   `git fetch origin`
3. Checkout a new local branch from the fetched branch using
   `git checkout -b coursework_2 origin/mlp2018-9/coursework_2`

You will now have a new branch in your local repository with everything you need to carry out the coursework.

This branch includes the following additions to your setup:

- For part 1:
  - Updates to the `mlp` python modules including (in the `mlp.layers` module) skeleton `ConvolutionalLayer` and `MaxPooling2DLayer` classes, and a `ReshapeLayer` class which allows the output of the previous layer to be reshaped before being forward propagated to the next layer in a multilayer model.
  - Jupyter notebooks, `ConvolutionalLayer_tests.ipynb` and `MaxPoolingLayer_tests.ipynb` for testing your implementations of convolutional and max pooling layers, and supporting test code.
- For part 2:

  - A Jupyter notebook, `Coursework_2_Pytorch_experiment_framework.ipynb`, which introduces the PyTorch framework, and provides some sample PyTorch code to get you started implementing Convolutional Networks in PyTorch and running experiments.
  - A directory `mlp/pytorch_experiment_scripts`, which includes tooling and ready to run scripts that to enable straightforward experimentation on GPU. Documentation on this is included in `notes/pytorch-experiment-framework.md`
  - A note on how to use the Google Cloud Platform (using your student credits), `notes/google_cloud_setup.md`.

- For the report:
  - A directory called `report` which contains the LaTeX template and style files for your report. You should copy all these files into the directory which will contain your report.

3 Tasks

This coursework comes in two parts. The objective of the first part is to implement convolutional networks in the MLP framework, and totest your implementation. The objective of the second part is to explore different approaches to integrating information in convolutional networks: pooling, strided convolutions, and dilated convolutions.

Carrying out larger convolutional network experiments using the MLP framework is is inefficient because (1) it runs on CPU and not GPU, and (2) a default implementation of a convolutional layer is unlikely to be computationally efficient. For this reason the second part of the coursework, which concerns running convolutional network experiments will use GPU computing (on the Google Compute Engine) and the highly efficient PyTorch framework.

Please note that part 2 of the coursework does not depend on part 1.
Thus you can do them in either order (or simultaneously).

Part 1: Implementing convolutional networks in the MLP framework

In the first part of the coursework, you should implement convolutional and max-pooling layers in the MLP framework, and carry out some basic experiments to validate your implementation.

1. Provide a convolutional layer implementations as a class `ConvolutionalLayer`. This class should implement the methods `fprop`, `bprop` and `grads_wrt_params`. There are two recommended approaches you might consider to do the implementation (you only need to do the implementation using one of these approaches), based on
   - the methods `scipy.signal.convolve2d` (and/or `scipy.signal.correlate2d`);
   - or using a "serialisation" approach using the method `im2col`.

   Both of these approaches are discussed below.

2. Implement a max-pooling layer. As a default, implement non-overlapping pooling (which was assumed in the lecture presentation).

3. Verify the correctness of your implementation using the supplied unit tests in the jupyter notebook files `notebooks/ConvolutionalLayer_tests.ipynb` and `notebooks/MaxPoolingLayer_tests.ipynb` to verify your convolutional layer and max pooling layer implementations respectively. **Note:** The tests are not exhaustive and should serve only as an indication of going into the right direction. Ideally you should write additional tests to validate your code in other scenarios. Take special care to check for edge cases.

4. Generate and submit your personalised output files for the unit tests:
   `test_max_pooling_results_pack.npz` and `test_convolution_results_pack.npz`
   These files are automatically generated by activating your conda mlp environment, changing to the `scripts` directory in `mlpractical`, and running the following commands:

   ```
   python generate_conv_layer_test_file.py --student_id sXXXXXX
   python generate_max_pool_layer_test_file.py --student_id sXXXXXXX
   ```

   Replace the sXXXXXXX with your student ID. Once the commands have run, the two .npz files will be generated in the scripts folder. You should make sure those files are included as part of your submission (see Section 5.2.

5. Since the required compute time for your convolutional network implementations will be substantial, it is not necessary to use this implementation to train and test convolutional networks on the EMNIST data.

For part 1 you should submit your code (in `mlp.layers`) and your output test files. There should also be a section of your report describing your implementation and including any analysis of the its efficiency.

Implementing convolutional layers

When you implement a convolutional layer both the `fprop` and `bprop` methods can be based on a convolution operation, as explained in the lectures. If we consider the `fprop` then the method operates on two 4-dimension tensors:

- The input (previous layer) to the convolution, whose dimensions are `(minibatch-size, num-feature-maps,` x_{in}`,` y_{in}`)`;
- The kernels (weight matrices) for the convolutions, whose dimensions are `(num-feature-maps-in, num-feature-maps-out,` x_{kernel}`,` y_{kernel}`)`.

The key to implementing a convolutional layer is how the convolutions are implemented. We recommend that you consider one of the following approaches:

1. Explicitly compute convolutions using the SciPy convolution function `scipy.signal.convolve2d` (or `scipy.signal.correlate2d`). Note that these functions convolve a 2-dimension image with a 2-dimension kernel, so your code will need to use this in the context of 4-dimension tensors where we have multiple feature maps and a batch of training examples.

2. "Serialisation" in which the the convolution operation is turned into a single matrix multiplication. The advantage of this is that implementing the convolutional `fprop` or `bprop` as a single large matrix multiplication is much more computationally efficient than the many small matrix multiplications a naive implementation would have. The disadvantage of this approach is that the resultant matrix has repeated elements, and be large (dependent on the number of feature map, batch size, and image size).

   This serialisation approach uses a function called `im2col` (and its reverse `col2im`). The `im2col` function is standard in Matlab and various computer vision packages, but it is not part of NumPy or SciPy; you may use the external Python implementations `im2col_indices` and `col2im_indices` at:
   https://github.com/huyouare/CS231n/blob/master/assignment2/cs231n/im2col.py

(35 Marks)

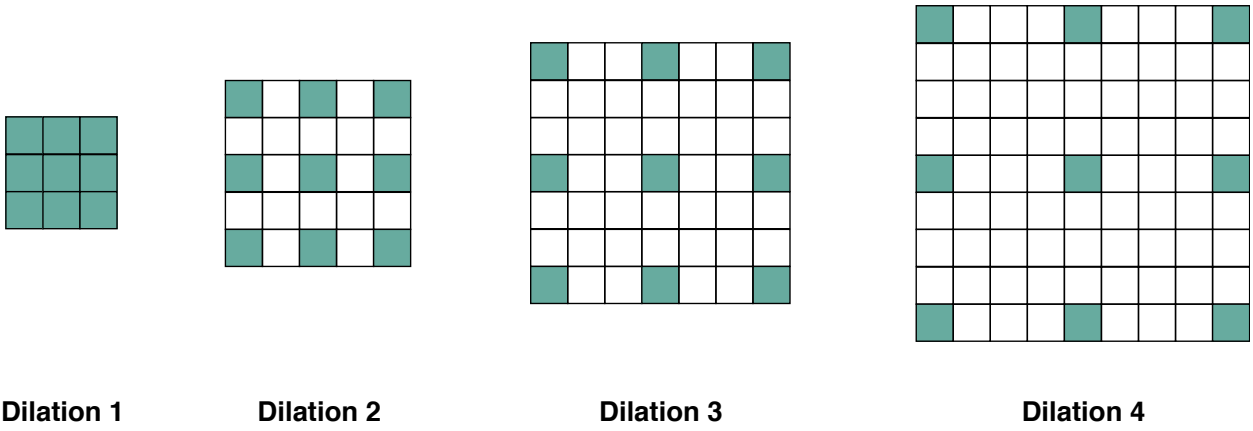

Figure 1: Kernels for dilated convolutions with successively increasing receptive field sizes. Figure copied from Antoniou et al. [2018] with permission.

Part 2: Exploring context in convolutional networks

The second part of the coursework will explore context in convolutional networks using PyTorch and the Google Compute Engine.

A powerful aspect of using convolutional networks for image classification is their ability to incorporate information at different scales. One can view a classifier for a task such as EMNIST as a pyramid-shaped construction of feature maps, in which convolutional layers close to the input are working at local context (in which each individual unit learns information about a small image patch), whereas units further from the input have a much broader context (in which each unit learns information about much larger regions of the image).

There are various ways this successive broadening of context can be achieved, while keeping small convolution filters, for example:

- **Pooling layers**: As discussed in the lectures (and to be implemented in part 1) pooling can be used to increase the context. In the 1990s, in architectures such as LeNet, average pooling was used. More recently max pooling has been more commonly used.
- **Striding**: Striding (how much a kernel shifts when scanning the image) to reduce the size of feature maps. Striding can also be combined with pooling.
- **Dilation**: Dilation is a way to incrementally increase the context of each kernel while keeping the number of parameters for each kernel the same [Yu and Koltun, 2016]. The basic idea is illustrated in figure 1: dilating a kernel involves progressively increasing the size of the receptive fields layer-by-layer. As the receptive field size increases, the number of parameters remains constant by increasing the distance between the filtered pixels. This is explained in more detail in section 2 of Antoniou et al. [2018] and in Yu and Koltun [2016]. Dilated convolutions, which were first developed for wavelet signal processing in the early 1990s, have been extensively used in the past 2-3 years for both computer vision [Yu and Koltun, 2016] and for audio/speech processing [van den Oord et al., 2016]. (Modern speech synthesis, such as deployed by Google, makes extensive use of dilated convolutions.)

In this task you should investigate different approaches to modelling image context in the classification of EMNIST. We suggest that you explore pooling, striding, and dilation, as mentioned above. The material in the `coursework_2` GitHub provides examples of convolutional networks, documented in `notes/pytorch-experiment-framework.md`, with code in `mlp/pytorch_experiment_scripts`. The provided framework supports max and average pooling, strided convolutions and dilated convolutions.

You should thus construct, carry out, and report a set of experiments, to address one or more research questions. (An example research question might be "does max pooling lead to more accurate classification than average pooling?".)

The expected amount of work for this task is a set of experiments which explore the different ways of modelling context in image classification using convolutional networks. Your experiments should involve *at most* 100h of GPU computation (using a single K80 GPU), which corresponds to the amount of computation covered by your voucher. Please note that we will not be able to provide extra credits in case the provided ones are spent.

Note that the default hyperparameters provided in the framework (see `pytorch_experiment_scripts/experiment_builder.py`) are reasonable – extensive hyperparameter searches are not required in this coursework. As a starting point you can explore a convolutional network using 4 convolutional layers with 64 filters, which is easy to run using the script `pytorch_experiment_scripts/train_evaluate_emnist_classification_system.py` with the default `num_layers` and `num_filters` options.

Important things to note about experiments:

- Your experiments should be designed so you are only varying one component at a time, so you can see the effect of that component-change.
- Designing experiments so that you are in a position to draw conclusions from your experiments is more important than the doing as many experiments as possible.
- When reporting the results of experiments, make sure that the comparison/contrast you are exploring is clear in the way you present the results.

Your report on this task should include the following:

- **Introduction**. Outline and explain the research questions you are investigating. Provide citations if appropriate.
- **Methodology**. Explain the methodology used – in this case the approaches to modelling image context that have you explored. Provide citations if appropriate.
- **Experiments**. Describe carefully the experiments carried out, providing enough information to make them reproducible. Present your results clearly and concisely. Graphs and tables should be constructed to make clear the contrasts and comparisons you are interested in based on the research questions. For instance, some interesting evaluation criteria that can be used to compare different strategies are classification accuracy and speed of your network.
- **Discussion and conclusions.** Discuss your results, with reference to the research questions, and if appropriate with reference to the literature. What conclusions can you draw from your experiments?

Using Google Compute Engine

1. You will receive an email containing the URL you will need to access in order to request a Google Cloud Platform coupon, and information about how to do this.
2. In the `coursework_2` branch of the GitHub, `notes/google_cloud_setup.md` gives the instructions you should follow to set up a Google Compute Engine instance to carry out this coursework.
3. The PyTorch experimental framework that is used for this coursework is described in `notes/pytorch-experiment-framework.md` and in `notebooks/Coursework_2_Pytorch_experiment_framework.ipynb`

(65 Marks)

4 Report

Your coursework will be primarily assessed based on your submitted report.

The directory `coursework_2/report` contains a template for your report (`mlp-cw2-template.tex`); the generated pdf file (`mlp-cw2-template.pdf`) is also provided, and you should read this file carefully as it contains some useful information about the required structure and content. The template is written in LaTeX, and we strongly recommend that you write your own report using LaTeX, using the supplied document style `mlp2018` (as in the template).

You should copy the files in the `report` directory to the directory containing the LaTeX file of your report, as `pdflatex` will need to access these files when building the pdf document from the LaTeX source file. The coursework 1 spec outlines how to create a pdf file from a LaTeX source file.

Your report should be in a 2-column format, based on the document format used for the ICML conference. The report should be a **maximum of 6 pages long**, not including references. We will not read or assess any parts of the report beyond this limit.

As discussed in the coursework 1 spec, all figures should ideally be included in your report file as vector graphics files, rather than raster files as this will make sure all detail in the plot is visible.

If you make use of any any books, articles, web pages or other resources you should appropriately cite these in your report. You do not need to cite material from the course lecture slides or lab notebooks.

5 Mechanics

Marks: This assignment will be assessed out of 100 marks and forms 40% of your final grade for the course.

Academic conduct: Assessed work is subject to University regulations on academic conduct:
http://web.inf.ed.ac.uk/infweb/admin/policies/academic-misconduct

Submission: You can submit more than once up until the submission deadline. All submissions are timestamped automatically. Identically named files will overwrite earlier submitted versions, so we will mark the latest submission that comes in before the deadline.

If you submit anything before the deadline, you may not resubmit after the deadline. (This policy allows us to begin marking submissions immediately after the deadline, without having to worry that some may need to be re-marked).

If you do not submit anything before the deadline, you may submit *exactly once* after the deadline, and a late penalty will be applied to this submission unless you have received an approved extension. Please be aware that late submissions may receive lower priority for marking, and marks may not be returned within the same timeframe as for on-time submissions.

Warning: Unfortunately the `submit` command will technically allow you to submit late even if you submitted before the deadline (i.e. it does not enforce the above policy). Don't do this! We will mark the version that we retrieve just after the deadline.

Extension requests: For additional information about late penalties and extension requests, see the School web page below. **Do not email any course staff directly about extension requests as these are handled by the ITO; you must follow the instructions on the web page.**

http://web.inf.ed.ac.uk/infweb/student-services/ito/admin/coursework-projects/late-coursework-extension-requests

Late submission penalty: Following the University guidelines, late coursework submitted without an authorised extension will be recorded as late and the following penalties will apply: 5 percentage points will be deducted for every calendar day or part thereof it is late, up to a maximum of 7 calendar days. After this time a mark of zero will be recorded.

5.1 Backing up your work

It is **strongly recommended** you use some method for backing up your work. Those working in their AFS homespace on DICE will have their work automatically backed up as part of the routine backup of all user homespaces. If you are working on a personal computer you should have your own backup method in place (e.g. saving additional copies to an external drive, syncing to a cloud service or pushing commits to your local Git repository to a private repository on Github). **Loss of work through failure to back up does not constitute a good reason for late submission**.

You may *additionally* wish to keep your coursework under version control in your local Git repository on the `coursework_2` branch.

If you make regular commits of your work on the coursework this will allow you to better keep track of the changes you have made and if necessary revert to previous versions of files and/or restore accidentally deleted work. This is not however required and you should note that keeping your work under version control is a distinct issue from backing up to guard against hard drive failure. If you are working on a personal computer you should still keep an additional back up of your work as described above.

5.2 Submission

Your coursework submission should be done electronically using the `submit` command available on DICE machines.

Your submission should include

- your completed report as a PDF file, using the provided template
- your local version of the `mlp` code including any changes you made to the modules (`.py` files)
- your personalised `test_max_pooling_results_pack.npz` and `test_convolution_results_pack.npz` files. These can be automatically generated by activating your conda mlp environment and pointing your terminal to the directory `scripts` in the `mlpractical` repo and running the following commands:

  ```
  python generate_conv_layer_test_file.py --student_id sXXXXXX
  python generate_max_pool_layer_test_file.py --student_id sXXXXXXX
  ```

  Replace the sXXXXXXX with your student ID. Once the commands have been ran, the two .npz files will be generated under the scripts folder. You should make sure those files are included as part of your submission folder.
- your local version of the Pytorch experiment framework (`mlp/pytorch_experiment_scripts`), including any changes you've made to existing files and any newly created files.
- a copy of your Pytorch experiment directories, including **only** the `.csv` files for your training, validation and test statistics. Please do not include model weights.

Please do not submit anything else (e.g. log files).

You should copy all of the files to a single directory, `coursework2`, e.g.

```
mkdir coursework2
cp reports/coursework2.pdf coursework2
cp scripts/test_max_pooling_results_pack.npz  coursework2
cp scripts/test_convolution_results_pack.npz  coursework2
```

also copy to `coursework2` the directories containing your mlp and PyTorch code, as well as the directory containing the PyTorch `csv` files.

You should then submit this directory using

```
submit mlp cw2 coursework2
```

Please submit the directory, not a zip file, not a tar file.

The `submit` command will prompt you with the details of the submission including the name of the files / directories you are submitting and the name of the course and exercise you are submitting for and ask you to check if these details are correct. You should check these carefully and reply `y` to submit if you are sure the files are correct and `n` otherwise.

You can amend an existing submission by rerunning the `submit` command any time up to the deadline. It is therefore a good idea (particularly if this is your first time using the DICE submit mechanism) to do an initial run of the `submit` command early on and then rerun the command if you make any further updates to your submission rather than leaving submission to the last minute.

6 Marking Guidelines

This document (Section 3 in particular) and the template report (`mlp-cw1-template.pdf`) provide a description of what you are expected to do in this assignment, and how the report should be written and structured.

Assignments will be marked using the scale defined by the **University Common Marking Scheme**:

Numeric mark	Equivalent letter grade	Approximate meaning
< 40	F	fail
40-49	D	poor
50-59	C	acceptable
60-69	B	good
70-79	A3	very good/distinction
80-100	A1, A2	excellent/outstanding/high distinction

Please note the University specifications for marks above 70:

A1 90-100 Often faultless. The work is well beyond what is expected for the level of study.

A2 80-89 A truly professional piece of scholarship, often with an absence of errors.
As 'A3' but shows (depending upon the item of assessment): significant personal insight / creativity / originality and / or extra depth and academic maturity in the elements of assessment.

A3 70-79
Knowledge: Comprehensive range of up-to-date material handled in a professional way.
Understanding/handling of key concepts: Shows a command of the subject and current theory.
Focus on the subject: Clear and analytical; fully explores the subject.
Critical analysis and discussion: Shows evidence of serious thought in critically evaluating and integrating the evidenced and ideas. Deals confidently with the complexities and subtleties of the arguments. Shows elements of personal insight / creativity / originality.
Structure: Clear and coherent showing logical, ordered thought.
Presentation: Clear and professional with few, relatively minor flaws. Accurate referencing. Figures and tables well constructed and accurate. Good standard of spelling and grammar.

References

Antreas Antoniou, Agnieszka Slowik, Elliot J. Crowley, and Amos Storkey. Dilated DenseNets for relational reasoning. *arXiv:1811.00410*, 2018. URL https://arxiv.org/abs/1811.00410.

A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu. WaveNet: A generative model for raw audio. *arXiv:1609.03499*, 2016. URL https://arxiv.org/abs/1609.03499.

Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. In *ICLR*, 2016. URL https://arxiv.org/abs/1511.07122.