Machine Learning Practical 2019/20: Coursework 2

Released: Monday 4 November 2019

Submission due: 16:00 Friday 22 November 2019

1 Introduction

The aim of this coursework is to further explore the classification of images using convolutional neural networks on a different dataset, CIFAR100 (pronounced as "see far 100"). CIFAR100 consists of 60,000 32×32 colour images in 100 classes, with 600 images per class. The first part of the coursework will concern the implementation of convolutional networks using the MLP framework. The second part involves debugging and fixing a "broken" neural network, and then subsequently enhancing the resulting (now healthy) network to improve its generalization performance.

In order to support your experiments, we have acquired Google Cloud Platform credits which allow the use of the Google Compute Engine infrastructure. You will need to run the second part of the coursework (which will be carried out in PyTorch). 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 in 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 validates your implementation for the first part. The detailed submission instructions are given in Section 5.2 – please follow these instructions carefully.

2 Github branch `mlp2019-20/coursework_2`

The provided code and setup information for this coursework is available on the course Github repository on a branch `mlp2019-20/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/mlp2019-20/coursework_2`

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

Before continuing, remember to run `bash install.sh` to install some additional dependencies required. This assumes that you have already installed your environment as explained in **Lab1**, under `notes/environment-set-up.md`.

This branch includes the following additions to your setup:

- For part 1:
  - Updates to the `mlp` python modules including (in the `mlp.layers` module) the skeleton code of a `ConvolutionalLayer`.
  - Jupyter notebooks, `ConvolutionalLayer_tests.ipynb` for testing your implementations of convolutional layers, and supporting test code.

- For part 2:
  - A Jupyter notebook, `Coursework_2_Pytorch_Introduction.ipynb`, which introduces Pytorch, and provides further resources on tutorials, documentation and debugging.
  - A directory `pytorch_mlp_framework/`, which includes tooling and ready to run scripts 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 objectives of the first part are

- To implement convolutional networks in the MLP framework.
- To validate your implementation using tests.

The objectives of the second part are:

- To diagnose a deep CNN that cannot be properly trained due to optimization issues, and subsequently propose and implement solutions to the problem. Since the aim of this is to improve the `training` behaviour of the network, the solutions should then be evaluated in terms of convergence speed and per-epoch training accuracy/loss.
- To improve the generalization performance of the fixed network using both standard and more advanced methods (e.g. regularization, data augmentation etc).

Carrying out larger convolutional network experiments using the MLP numpy framework is computationally inefficient because (1) it runs on CPU and not on GPU, and (2) a default implementation of a convolutional layer is unlikely to be computationally efficient. For these reasons the second part of the coursework, which involves 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).

3.1 Part 1: Implementing convolutional networks in the MLP framework

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

1. We provide a convolutional layer skeleton as a class named `ConvolutionalLayer`. This class should implement the methods `fprop`, `bprop` and `grads_wrt_params`. There are two recommended approaches that 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. Verify the correctness of your implementation using the supplied unit tests in the jupyter notebook file `notebooks/ConvolutionalLayer_tests.ipynb` to verify your convolutional layer implementations. **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.

3. Generate and submit your personalised output files for the unit tests: `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 sXXXXXXX
   ```

   Replace the `sXXXXXXX` with your student ID. Once the commands have run, the .npz file will be generated in the scripts folder. You should make sure that file is included as part of your submission (see Section 5.2).

4. Since the required compute time for your convolutional network implementations will be substantial, it is **not** necessary/recommended to use this implementation to train and test convolutional networks on the CIFAR100 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 (either using pseudo-code or detailed verbal explanation) and including analysis of its computational efficiency.

3.1.1 Implementing convolutional layers

Time budget[1]*: This section should take about 30% of the time you have allocated for your coursework.*

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.

[1] These are simply suggested guidelines, feel free to adjust them based on your own preferences

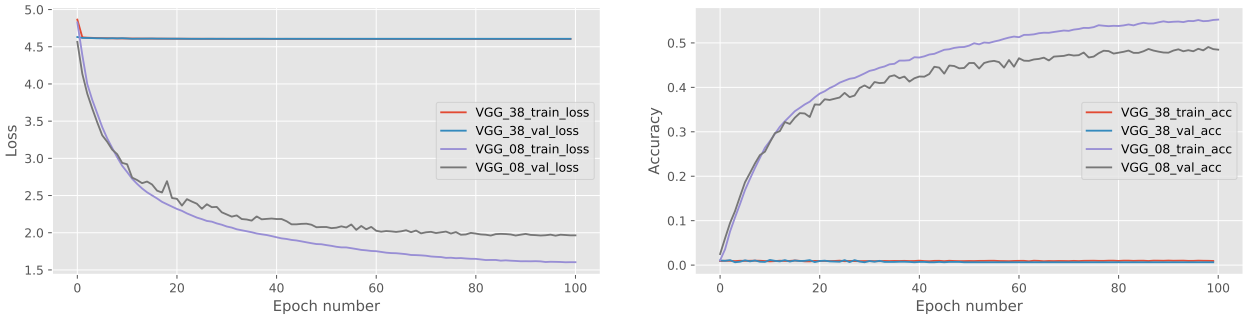

Figure 1: Training and validation plots of Healthy (VGG_08) vs Broken (VGG_38) in terms of error and accuracy.

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

(25 Marks)

3.2 Part 2: Debugging training problems of a deep CNN and improving its generalization performance

The second part of the coursework involves debugging and tuning deep CNNs using PyTorch and the Google Compute Engine.

Identifying and resolving any optimization-related problems that might prevent your model from fitting the training set are critical skills when working with deep neural networks. Furthermore once fitting is achieved as expected, it might still have severe overfitting problems, which cause low generalization performance. Being able to work around these two key problems is key to building high performance deep neural networks.

3.2.1 Introducing our broken CNN

Time budget[1]*: This section should take about 25% of the time you have allocated for your coursework.*

Note: This section's objective is to test your knowledge, understanding and research skill. It's not meant to be an implementation-oriented section. The solutions require you to add at most 8-10 lines of code.

Using the Pytorch-based research framework that can be found in the `pytorch_mlp_framework` folder, we have built, trained and evaluated 2 deep CNNs. One consisting of a total of 7 convolutional layers + one fully connected, and another consisting of 37 convolutional layers + one fully connected layer. Figure 1 illustrates the training/val loss performance of the two models. One can clearly see that the 37 layer CNN was unable to minimize its loss. Given that we know that extra layers means more abstraction power, parameters and capacity, one would expect the deeper model to be doing better at learning than the shallow one, however this is simply not the case.

3.2.2 Identifying and debugging the problem

1. Construct a hypothesis as to what is causing the issue in Figure 1, as well as the collected metrics in folders `VGG_08` and `VGG_38`. Remember to support your position using arguments based on quantitative observations.

2. Propose 2 solutions to the problem. Implement your solutions using the provided research framework, and the guidelines in Section 3.2.3. Design and run experiments that compare and contrast your proposed solutions, then discuss the results (Which one is better? Why? Is the problem partially or completely resolved?).

3.2.3 Implementation Guidelines

There exist *at least* several methods that can improve the training performance of the broken 37 layer CNN introduced in Section 3.2.1. The recommended solutions are described in Ioffe and Szegedy [2015], He et al. [2016]. You can also find alternative ones in Huang et al. [2017], Lee et al. [2015] which can be more challenging to implement. We have written the MLP Pytorch framework in a way that allows you to implement said solutions with minimal effort. Each solution will require at approximately 2-8 lines of code.

Important Components to Note

1. The classes `ConvolutionalProcessingBlock` and `ConvolutionalDimensionalityReductionBlock` located within `pytorch_mlp_framework/model_architectures.py`. The former class implements a basic cascade of 2 convolutional layers, each followed by a Leaky ReLU activation function, while the latter implements a basic cascade of 2 convolutional layers, with an average pooling layer in the middle, which effectively halves the height and width dimensions of the tensor volume passing through it. The former is used as the basic building block of the model (repeated `num_blocks_per_stage times`), while the latter is used only as a dimensionality reduction layer, usually once after each *network stage*. A network stage is considered a cascade of convolutional layers, followed by a dimensionality reduction function such as average pooling.

2. Lines 43-50 in `pytorch_mlp_framework/train_evaluate_image_classification_system.py` showcase how one can create an if-else structure that can read arguments from the command line in order to choose which processing and dim_reduction blocks will be used.

3. Lines 17-27 in the same file showcase how some simple data-augmentation strategies can be applied to the system (useful in improving the generalization of the model.)

Implementing a proposed solution

1. Copy the classes `ConvolutionalProcessingBlock` and `ConvolutionalDimensionalityReductionBlock` located within `pytorch_mlp_framework/model_architectures.py`, and provide a name referring to your proposed solution. Then change the two blocks to implement your proposed solution.

2. To create a new block ensure that you rewrite the `build_module()` and `forward()` methods.

3. Add a new clause in the if-else structure in lines 43-50 of `pytorch_mlp_framework/train_evaluate_image_classification_system.py` to add a choice for a configuration using your new blocks.

4. Write small unit-tests for your blocks to ensure that the layers can fprop some data without throwing errors related to Pytorch.

5. Once tested, you can use the argument parser to easily run experiments with your new modules. Hint: Have a look at the `block_type` argument under `pytorch_mlp_framework/arg_extractor.py`, to get an idea on how to easily pass module names as an argument.

Note: Some potential solutions might not require modification of the above, but those are considered more advanced. If you choose that path, you will also bear the responsibility of figuring out the design of your code. That being said, most of known methods can be implemented using the above 3 steps.

3.2.4 Improving the generalization performance of the fixed model

Time budget[1]*: This section should take about 5% of the time you have allocated for your coursework.*

Note: If you were unable to debug the 37 layer deep CNN in the previous task, you can still get full marks on this section by improving the generalization performance of the 7 layer CNN. If you want to get more modelling capacity for your networks, simply increase the number of filters to 64. However, should you choose to do this, you'll need to run some baseline experiments on a 7 layer, 64 filter conv-net first, so you have something to compare against.

Once a deep network is able to fit the training set at almost perfect performance, that's when we are ready to begin improving its generalization performance. This can be achieved with a variety of methods, some of which include regularization methods, data augmentation, special layer types, and various network topologies. In this section your objective is to choose a few of such methods, implement them and compare their performance with each other. Finally, you should use what you've learned to build your final model, potentially combining some of the best methods you have experimented with, to get your final best test accuracy.

To break this down:

1. Select a few methods targeted towards improving the generalization performance of the model, such as weight decay, L1/L2 regularization, dropout, data-augmentation etc.
2. Implement selected methods. (Most of the above can be implemented using 1 line of code, or are already implemented into the framework, i.e. weight decay and some forms of data-augmentation).
3. Design and run experiments to investigate the performance of the methods.
4. Build a final model using the knowledge you have gained and evaluate its test performance.
5. Write about what you did and why in your report.

Note that the default hyperparameters (found in `arg_extractor.py`) for provided in the framework (see `pytorch_experiment_scripts/experiment_builder.py`) are reasonable – extensive hyperparameter searches are not required in this coursework.

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 and are they consistent with the literature?

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`

(75 Marks)

4 Report

Time budget[1]*: The report should take about 40% of the time you have allocated for your coursework.*

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 `mlp2019` (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_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
  ```

  Replace the sXXXXXXX with your student ID. Once the commands have been ran, a `test_convolution_results_pack.npz` file will be generated under the scripts folder. You should make sure this file is included in 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_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

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, pages 448–456, 2015. URL http://proceedings.mlr.press/v37/ioffe15.html.

Chen-Yu Lee, Saining Xie, Patrick Gallagher, Zhengyou Zhang, and Zhuowen Tu. Deeply-supervised nets. In *Artificial intelligence and statistics*, pages 562–570, 2015.