Machine Learning Practical 2018/19: Coursework 1

Released: Monday 15 October 2018

Submission due: 16:00 Friday 26 October 2018

1 Introduction

The aim of this coursework is to explore the RMSProp and Adam learning algorithms (discussed in lecture 5) in the context of L2 regularization (discussed in lecture 4). The motivation for the coursework comes from the recent paper by Loshchilov and Hutter [2017] on using the Adam optimizer with L2 regularization and weight decay.

The coursework will use an extended version of the MNIST database, the EMNIST Balanced dataset, described in Section 2. Section 3 describes the additional code provided for the coursework (in branch `mlp2018-9/coursework_1` of the MLP github), and Section 4 describes how the coursework is structured into four tasks. The main deliverable of this coursework is a report, discussed in section 5, using a template that is available on the github. Section 6 discusses the details of carrying out and submitting the coursework, and the marking scheme is discussed in Section 7.

You will need to submit your completed report as a PDF file and your local version of the `mlp` code including any changes you made to the provided (`.py` files). The detailed submission instructions are given in Section 6.2 – please follow these instructions carefully.

2 EMNIST dataset

In this coursework we shall use the EMNIST (Extended MNIST) Balanced dataset [Cohen et al., 2017], https://www.nist.gov/itl/iad/image-group/emnist-dataset. EMNIST extends MNIST by including images of handwritten letters (upper and lower case) as well as handwritten digits. Both EMNIST and MNIST are extracted from the same underlying dataset, referred to as NIST Special Database 19. Both use the same conversion process resulting in centred images of dimension 28×28.

There are 62 potential classes for EMNIST (10 digits, 26 lower case letters, and 26 upper case letters). However, we shall use a reduced label set of 47 different labels. This is because (following the data conversion process) there are 15 letters for which it is confusing to discriminate between upper-case and lower-case versions. In the 47 label set, upper- and lower-case labels are merged for the following letters:
`C, I, J, K, L, M, O, P, S, U, V, W, X, Y, Z`.

The training set for Balanced EMNIST has about twice the number of examples as the MNIST training set, thus you should expect the run-time of your experiments to be about twice as long. The expected accuracy rates are lower for EMNIST than for MNIST (as EMNIST has more classes, and more confusable examples), and differences in accuracy between different systems should be larger. Cohen et al. [2017] present some baseline results for EMNIST.

You do not need to directly download the EMNIST database from the nist.gov website, as it is part of the `coursework_1` branch in the `mlpractical` Github repository, discussed in Section 3 below.

3 Github branch `mlp2018-9/coursework_1`

You should run all of the experiments for the coursework inside the Conda environment you set up for the labs. The code for the coursework is available on the course Github repository on a branch `mlp2018-9/coursework_1`. 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_1 origin/mlp2018-9/coursework_1`

You will now have a new branch in your local repository with all the code necessary for the coursework in it.

This branch includes the following additions to your setup:

- A new `EMNISTDataProvider` class in the `mlp.data_providers` module. This class makes some changes to the `MNISTDataProvider` class, linking to the `EMNIST Balanced` data, and setting the number of classes to 47.
- Training, validation, and test sets for the `EMNIST Balanced` dataset that you will use in this coursework.
- Three new classes in the `mlp.learning_rules` module:
  `RMSPropLearningRule, AdamLearningRule, AdamLearningRuleWithWeightDecay`.
  These are skeleton classes that you will need fully implement as part of the coursework (see Sections 4.2 and 4.4).
- Two new classes in the `mlp.schedulers` module:
  `ConstantLearningRateScheduler, CosineAnnealingWithWarmRestartsScheduler`.
  The former is an example of how a scheduler works in the `mlp` framework, and the latter is skeleton code, to be completed as part of the coursework (Section 4.3).
- A new module `mlp.tests` containing 2 new methods:
  `test_cosine_scheduler, test_adam_with_weight_decay`.
  Both of these are used by the testing notebooks to test the methods to be implemented for their functionality and edge-case robustness (Sections 4.3 and 4.4).
- Two Jupyter notebooks:
  `CosineAnnealingScheduler_tests.ipynb, WeightDecay_tests.ipynb`
  to be used for testing the implementations of `CosineAnnealingWithWarmRestartsScheduler` and `AdamLearningRuleWithWeightDecay` (Sections 4.3 and 4.4). The tests serve as a safeguard to prevent experimentation with faulty code which might lead to wrong conclusions. Tests in general are a *vital* ingredient for good software development, and especially important for building correct and efficient deep learning systems.
- 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.

4 Tasks

The coursework is structured into 4 tasks, all supported by experiments on `EMNIST`

1. Setting up a **baseline** system on `EMNIST`.
2. Implementation and exploration of the **RMS-Prop** and **Adam** learning algorithms.
3. Implementation and exploration of a **learning rate scheduler** for stochastic gradient descent (SGD) and Adam.
4. Implementation and exploration of **Adam with regularization and explicit weight decay**.

4.1 Baseline

Initially you need to establish a baseline system on `EMNIST` using stochastic gradient descent (SGD) and without explicit regularization. Carry out experiments using 100 ReLU hidden units per layer, and investigate using from 2–5 hidden layers. Make sure you use an appropriate learning rate. For the initial experiments, compare systems using the validation set accuracy. Then for the best systems with 2–5 hidden layers compare them using the test set accuracy.

For the experiments going forward you should use a standard architecture to compare the different algorithms that you explore: it is recommended that you use 3 hidden layers and 100 hidden units.

(10 Marks)

4.2 RMSProp and Adam

In this part of the coursework you should implement and explore the RMSProp [Tieleman and Hinton, 2012] and Adam [Kingma and Ba, 2015] learning algorithms that were presented in lecture 5). RMSProp and Adam attempt to tune automatically the scale of updates in a parameter-dependent fashion. Implement these learning rules by completing the classes `RMSPropLearningRule` and `AdamLearningRule` in the `mlp.learning_rules` module.

You should experiment with these algorithms on `EMNIST` using the standard 3 hidden layer architecture. You need to make sure that you use appropriate values for the hyperparameters of these algorithms: the learning rate, the decay rate, and the momentum parameter. Since the learning rate parameter has a different meaning for different algorithms, you should not force it to be the same across different learning algorithms – it is most important to find an appropriate value for each algorithm.

In your experiments you should compare the behaviour of RMSProp and Adam with SGD in terms of the accuracy and the learning curve, using the validation set to select the best hyperparameter values for each algorithm. Once this is done compare the behaviour of the learning algorithms using the test set.

(20 Marks)

4.3 Cosine annealing learning rate scheduler

A practical problem when training neural networks, which you will have already encountered, is how the learning rate is set, and how it should change from batch-to-batch or from epoch-to-epoch. A well-designed scheduler can save you from a great amount of handtuning (and sleepless nights!). There are various approaches to this: early stopping approaches (which are important when the training set is very large and computational resources are not infinite) use the validation set to control the learning rate adjustments, with training stopping when accuracy improvements are no longer observed. An alternative way to approach the problem is to assume a fixed number ("budget") of training epochs, and then to optimise the learning rate schedule within that budget.

In section 5 of their paper, Loshchilov and Hutter [2017] present a learning rate scheduler that works with a fixed number of training epochs. The learning rate is controlled by two processes: a function which decreases it through time, and periodic "warm restarts" in which the learning rate is increased (and after which is decreased again. Loshchilov and Hutter use a cosine function to decrease the learning rate ("cosine annealing" – equation (7)), and use a hyperparameter to control when a restart occurs. This scheduler can be applied to both SGD (in this case they call it SGDR – Stochastic Gradient Descent with Warm Restarts) and to Adam.

In this part of the coursework you should implement the cosine scheduling algorithm, basing your implementation on equation (7) of section 5 in Loshchilov and Hutter [2017]. Your implementation should be carried out by completing `CosineAnnealingWithWarmRestartsScheduler` in the `mlp.schedulers` module. Look at the class `ConstantLearningRateScheduler` in the same module as an example of how a scheduler works in the `mlp` framework. You can use the supplied Jupyter Notebook, `CosineAnnealingScheduler_tests.ipynb`, to test your implementation. Note that this Notebook contains a basic test (which your code must pass), and a more advanced test for "experiment continuation" which tests that the scheduler can be restarted from a specific epoch.

You should apply this learning rate scheduler to both SGD and Adam, carrying out test on `EMNIST` using your standard architecture, and a budget of 100 epochs. For each learning algorithm the baseline case should using a fixed learning rate with no scheduler; you should compare this with using cosine annealing with no restarts, and with cosine annealing with restarts specified by an initial $T_i = 25$ and $T_{mult} = 3$, using the notation in the paper. As before carry out your initial experiments to determine the best hyperparameters on the validation set, with the final comparisons for the learning rate scheduler experiments carried out using the test set.

(30 Marks)

4.4 Regularization and weight decay with Adam

Loshchilov and Hutter [2017] pointed out that although L2 regularization corresponds to weight decay for stochastic gradient descent, for other learning algorithms, there is an interaction between the regularization hyperparameter and the learning rate hyperparameter(s). They argued that this is not desirable and that empirically it results in degraded test accuracy; they hypothesised that it is preferable to use weight decay directly with a hyperparameter that is independent of the hyperparameters used for the learning algorithm.

In the final part of the coursework, you should explore the use of L2 regularization and weight decay for the Adam optimizer, based on Algorithm 2 in Loshchilov and Hutter [2017].[1] L2 regularization for Adam can be applied using `L2Penalty` applied to an `AffineLayer` (see Lab 5); you should carry out the implementation of weight decay for Adam by completing the `AdamLearningRuleWithWeightDecay` class in the `mlp.learning_rules` module. When carrying out this implementation take care on making sure that the equation in line 12 of algorithm 2 is what is implemented in your code – note the relationship between the scheduling variable (η_t), the learning rate (α), and the weight decay factor (w). (Note that Loshchilov and Hutter use an unusual notation in which $\mathbf{x}$ denotes the weights, w is the hyperparameter for the amount of L2 regularization ow weight decay, and α is the learning rate.)

You can use the supplied Jupyter Notebook, `WeightDecay_tests.ipynb`, to test your implementation.

You should initially explore the hyperparameters involved in this algorithm. You should then carry out experiments to compare the following (all using Adam):

- L2 regularization vs. weight decay
- constant learning rate vs. cosine annealing schedule
- no restarts in the scheduler vs. use of a warm restart

As before use a budget of 100 training epochs.

(40 Marks)

[1] Note that algorithm 2 uses colour show the choice between L2 regularization and weight decay.

5 Report

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

The directory `coursework_1/report` contains a template for your report (`mlp-cw1-template.tex`); the generated pdf file (`mlp-cw1-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 `mlp2017` (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.

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 5 pages long**, not including references. We will not read or assess any parts of the report beyond this limit.

Ideally, all figures should 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. Matplotlib supports saving high quality figures in a wide range of common image formats using the `savefig` function. **You should use `savefig` rather than copying the screen-resolution raster images outputted in the notebook.** An example of using `savefig` to save a figure as a PDF file (which can be included as graphics in LaTeX compiled with `pdflatex` is given below.

```
import matplotlib.pyplot as plt
import numpy as np
# Generate some example data to plot
x = np.linspace(0., 1., 100)
y1 = np.sin(2. * np.pi * x)
y2 = np.cos(2. * np.pi * x)
fig_size = (6, 3)  # Set figure size in inches (width, height)
fig = plt.figure(figsize=fig_size)  # Create a new figure object
ax = fig.add_subplot(1, 1, 1)  # Add a single axes to the figure
# Plot lines giving each a label for the legend and setting line width to 2
ax.plot(x, y1, linewidth=2, label='$y = \sin(2\pi x)$')
ax.plot(x, y2, linewidth=2, label='$y = \cos(2\pi x)$')
# Set the axes labels. Can use LaTeX in labels within $...$ delimiters.
ax.set_xlabel('$x$', fontsize=12)
ax.set_ylabel('$y$', fontsize=12)
ax.grid('on')  # Turn axes grid on
ax.legend(loc='best', fontsize=11)  # Add a legend
fig.tight_layout()  # This minimises whitespace around the axes.
fig.savefig('file-name.pdf') # Save figure to current directory in PDF format
```

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.

To create a pdf file `mlp-cw1-template.pdf` from a LaTeX source file (`mlp-cw1-template.tex`), you can run the following in a terminal:

```
pdflatex mlp-cw1-template
bibtex mlp-cw1-template
pdflatex mlp-cw1-template
pdflatex mlp-cw1-template
```

(Yes, you have to run pdflatex multiple times, in order for latex to construct the internal document references.)

An alternative, simpler approach uses the `latexmk` program:

```
latexmk -pdf mlp-cw1-template
```

Another alternative is to use an online LaTeX authoring environment such as https://overleaf.com – note that all staff and students have free access to Overleaf Pro - see https://www.ed.ac.uk/information-services/computing/desktop-personal/software/main-software-deals/other-software/overleaf.

It is worth learning how to use LaTeX effectively, as it is particularly powerful for mathematical and academic writing. There are many tutorials on the web.

6 Mechanics

Marks: This assignment will be assessed out of 100 marks and forms 10% 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**; 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.

6.1 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_1` 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.

6.2 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)

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

You should copy all of the files to a single directory, `coursework1`, e.g.

```
mkdir coursework1
cp reports/coursework1.pdf coursework1
```

and then submit this directory using

```
submit mlp cw1 coursework1
```

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.

7 Marking Guidelines

This document (Section 4 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.

And finally... this assignment is worth 10% of the total marks for the course, and the next assignment is worth 40%. This is not because the second assignment is four times bigger or harder than this one (although it will be more challenging). The reason that this assignment is worth 10% is so that people get an opportunity to learn from their errors in doing the assignment, without it having a very big impact on their overall grade for the module.

References

Gregory Cohen, Saeed Afshar, Jonathan Tapson, and André van Schaik. EMNIST: an extension of MNIST to handwritten letters. *arXiv preprint arXiv:1702.05373*, 2017. URL https://arxiv.org/abs/1702.05373.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICML*, 2015. URL https://arxiv.org/abs/1412.6980.

Ilya Loshchilov and Frank Hutter. Fixing weight decay regularization in Adam. *arXiv preprint arXiv:1711.05101*, 2017. URL https://arxiv.org/abs/1711.05101.

Tijmen Tieleman and Geoffrey E. Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, 4(2), 2012. URL https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.