Machine Learning Practical: Coursework 1

Release date: Monday 10th October 2016
Due date: 16:00 Thursday 27th October 2016

Introduction

This coursework is concerned with training multi-layer networks to address the MNIST digit classification problem. It builds on the material covered in the first three lab notebooks and the first four lectures. It is highly recommended that you complete the first three lab notebooks before starting the coursework. The aim of the coursework is to investigate the effect of learning rate schedules and adaptive learning rates on the progression of training and the final performance achieved by the trained models.

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

Late submissions: The School of Informatics policy is that late coursework normally gets a mark of zero. See http://web.inf.ed.ac.uk/infweb/student-services/ito/admin/coursework-projects/late-coursework-extension-requests for exceptions to this rule. Any requests for extensions should go to the Informatics Teaching Office (ITO), either directly or via your Personal Tutor.

Report

The main component of your coursework submission, on which you will be assessed, will be a short report. This should follow a typical experimental report structure, in particular covering the following

- a clear description of the methods used and algorithms implemented,
- quantitative results for the experiments you carried out including relevant graphs,
- discussion of the results of your experiments and any conclusions you have drawn.

The report should be submitted in PDF. You are welcome to use what ever document preparation tool you prefer working with to write the report providing it can produce a PDF output and can meet the required presentation standards for the report.

Of the total 100 marks for the coursework, 25 marks have been allocated for the quality of presentation and clarity of the report. A good report, will clear, precise, and concise. It will contain enough information for someone else to reproduce your work (with the exception that you do not have to include the values to which the parameters were randomly initialised).

You will need to include experimental results plotted as graphs in the report. You are advised (but not required) to use `matplotlib` to produce these plots, and you may reuse code plotting (and other) code given in the lab notebooks as a starting point.

Each plot should have all axes labelled and if multiple plots are included on the same set of axes a legend should be included to make clear what each line represents. Within the report all figures should be numbered (and you should use these numbers to refer to the figures in the main text) and have a descriptive caption stating what they show.

Ideally all figures should be included in your report file as vector graphics 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` and in Apple Pages and Microsoft Word documents) 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 are using Libre/OpenOffice you should use Scalable Vector Format plots instead using `fig.savefig('file-name.svg')`. If the document editor you are using for the report does not support including either PDF or SVG graphics you can instead output high-resolution raster images using `fig.savefig('file-name.png', dpi=200)` however note these files will generally be larger than either SVG or PDF formatted graphics.

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.

Code

You should run all of the experiments for the coursework inside the Conda environment you set up in the first lab.

The code for the coursework is available on the course Github repository on a branch `mlp2016-7/coursework1`. 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 on have been committed (see details here 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 coursework1 origin/mlp2016-7/coursework1`

You will now have a new branch in your local repository with all the code necessary for the coursework in it. In the `notebooks` directory there is a notebook `Coursework_1.ipynb` which is intended as a starting point for structuring the code for your experiments. You will probably want to add additional code cells to this as you go along and run new experiments (e.g. doing each new training run in a new cell). You may also wish to use Markdown cells to keep notes on the results of experiments.

Submission

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

Your submission should include

- your completed course report as a PDF file,
- the notebook (`.ipynb`) file you use to run the experiments in
- and your local version of the `mlp` code including any changes you make to the modules (`.py` files).

You should EITHER (1) package all of these files into a single archive file using `tar` or `zip`, e.g.

```
tar -zcf coursework1.tar.gz notebooks/Coursework_1.ipynb mlp/*.py reports/coursework1.pdf
```

and then submit this archive using

```
submit mlp 1 coursework1.tar.gz
```

OR (2) copy all of the files to a single directory `coursework1` directory, e.g.

```
mkdir coursework1
cp notebooks/Coursework_1.ipynb mlp/*.py reports/coursework1.pdf coursework1
```

and then submit this directory using

```
submit mlp 1 coursework1
```

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 submisison rather than leaving submission to the last minute.

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 consitute a good reason for late submission**.

You may *additionally* wish to keep your coursework under version control in your local Git repository on the `coursework1` branch. This does not need to be limited to the coursework notebook and `mlp` Python modules - you can also add your report document to the repository.

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.

Standard network architecture

To make the results of your experiments more easily comparable, you should try to keep as many of the free choices in the specification of the model and learning problem the same across different experiments. If you vary only a small number of aspects of the problem at a time this will make it easier to interpret the effect those changes have.

In all experiments you should therefore use the same model architecture and parameter initialisation method. In particular you should use a model composed of three affine transformations interleaved with logistic sigmoid nonlinearities, and a softmax output layer. The intermediate layers between the input and output should have a dimension of 100 (i.e. two hidden layers with 100 units in each hidden layer). This can be defined with the following code:

```
import numpy as np
from mlp.layers import AffineLayer, SoftmaxLayer, SigmoidLayer
from mlp.errors import CrossEntropySoftmaxError
from mlp.models import MultipleLayerModel
from mlp.initialisers import ConstantInit, GlorotUniformInit

seed = 10102016
rng = np.random.RandomState(seed)

input_dim, output_dim, hidden_dim = 784, 10, 100

weights_init = GlorotUniformInit(rng=rng)
biases_init = ConstantInit(0.)

model = MultipleLayerModel([
    AffineLayer(input_dim, hidden_dim, weights_init, biases_init),
    SigmoidLayer(),
    AffineLayer(hidden_dim, hidden_dim, weights_init, biases_init),
    SigmoidLayer(),
    AffineLayer(hidden_dim, output_dim, weights_init, biases_init)
])

error = CrossEntropySoftmaxError()
```

Here we are using a special parameter initialisation scheme for the weights which makes the scale of the random initialisation dependent on the input and output dimensions of the layer, with the aim of trying to keep the scale of activations at different layers of the network the same at initialisation. The scheme is described in *Understanding the difficulty of training deep feedforward neural networks*, Glorot and Bengio (2011). As also recommended there we initialise the biases to zero. You do not need to read or understand this paper for the assignment, it only being mentioned to explain the use of `GlorotUniformInit` in the above code. You should use this parameter initialisation for all of your experiments.

As well as standardising the network architecture, you should also fix the hyperparameters of the training procedure not being investigated to be the same across different runs. In particular for all experiments you should use a **batch size of 50 and train for a total of 100 epochs** for all reported runs. You may of course use a smaller number of epochs for initial pilot runs.

Part 1: Learning rate schedules (10 marks)

In the first part of the assignment you will investigate how using a time-dependent learning rate schedule influences training.

Implement one of the two following time-dependent learning rate schedules:

- exponential $\eta(t) = \eta_0 \exp(-t/r)$
- reciprocal $\eta(t) = \eta_0 (1 + t/r)^{-1}$

where η_0 is the initial learning rate, t the epoch number, $\eta(t)$ the learning rate at epoch t and r a free parameter governing how quickly the learning rate decays.

You should implement the schedule by creating a new scheduler class in the `mlp.schedulers.py` module which follows the interface of the example `ConstantLearningRateScheduler` given in the module. In particular as well as an `__init__` method initialising the object with any free parameters for the schedule, the class should define a `update_learning_rule` method which sets the `learning_rate` attribute of a learning rule object based on the current epoch number.

A (potentially empty) list of scheduler objects are passed to the `__init__` method of the `Optimiser` object used to train the model, for example

```
schedulers = [ConstantLearningRateScheduler(learning_rate)]
optimiser = Optimiser(
    model, error, learning_rule, train_data,
    valid_data, data_monitors, schedulers)
```

You should:

- Compare the performance of your time-dependent learning rate schedule when training the standard model on the MNIST digit classification task, to training with a constant learning rate baseline.
- Indicate how the free schedule parameters η_0 and r affect the evolution of the training.
- State the final error function and classification accuracy values and include plots of the evolution of the error and accuracy across the training epochs for both the training and validation sets. These should be reported for both the constant learning rate baseline and *at least* one run with your learning rate scheduler implementation.

Part 2: Momentum learning rule (15 marks)

In this part of the assignment you will investigate using a gradient descent learning rule with momentum. This extends the basic gradient learning rule by introducing extra momentum state variables for the parameters. These can help the learning dynamic help overcome shallow local minima and speed convergence when making multiple successive steps in a similar direction in parameter space.

An implementation of the momentum learning rule is given in the `mlp.learning_rules` module in the `MomentumLearningRule` class. Read through the code and documentation for this class and make sure you understand how it relates to the equations given in the lecture slides.

In addition to the `learning_rate` parameter, the `MomentumLearningRule` also accepts a `mom_coeff` argument. This *momentum coefficient* $\alpha \in [0, 1]$ determines the contribution of the previous momentum value to the new momentum after an update.

As a first task you should:

- Compare the performance of a basic gradient descent learning rule to the momentum learning rule for several values of the momentum coefficient α.
- Interpret how the momentum coefficient α influences training.
- Include plots of the error and accuracy training curves across the training epochs, for the different momentum coefficients you test.

Analogous to scheduling of the learning rate, it is also possible to vary the momentum coefficient over a training run. In particular it is common to increase the coefficient from an initially lower value at the start of training (when the direction of the gradient of the error function in parameter space are likely to vary a lot) to a larger value closer to 1 later in training. One possible schedule is

$$\alpha(t) = \alpha_\infty \left(1 - \frac{\gamma}{t + \tau}\right) \tag{1}$$

where $\alpha_\infty \in [0, 1]$ determines the asymptotic momentum coefficient and $\tau \geq 1$ and $0 \leq \gamma \leq \tau$ determine the initial momentum coefficient and how quickly the coefficient tends to α_∞.

You should create a scheduler class which implements the above momentum coefficient schedule by adding a further definition to the `mlp.schedulers` module. This should have the same interface as the learning rate scheduler implemented in the previous part.

Using your implementation you should:

- Try out several different momentum rate schedules by using different values for α_∞, γ and τ and investigate whether using a variable momentum coefficient gives improved performance over a constant momentum coefficient baseline.

Part 3: Adaptive learning rules (40 marks)

In the final part of the assignment you will investigate adaptive learning rules which attempt to automatically tune the scale of updates in a parameter-dependent fashion.

You should implement **two** of the three adaptive learning rules mentioned in the fourth lecture slides: AdaGrad, RMSProp and Adam.

You should implement the learning rules by defining new classes inheriting from `GradientDescendLearningRule` in the `mlp/learning_rules.py` module. The `MomentumLearningRule` class should show you how to define learning rules which use additional state variables to calculate the updates to the parameters.

You should:

- Compare the performance of your two implemented adaptive training rules to your previous results using the basic gradient descent and momentum learning rules. Ideally you should compare both in terms of speed of convergence (including potentially accounting for greater computional cost of the adaptive updates) and the final error / classification accuracy on both training and validation data sets.
- Briefly discuss any free parameters in the adaptive learning rules you implement and how sensitive training performance seems to the values used for them.
- Include example plots of the evolution of the error and accuracy across the training epochs for the training and validation sets for both of your implemented adaptive learning rules.

Marking Scheme

- Part 1, Learning Rate Schedules (10 marks). Marks awarded for completeness of implementation, experimental methodology, experimental results.
- Part 2, Momentum Learning Rule (15 marks). Marks awarded for completeness of implementation, experimental methodology, experimental results.
- Part 3, Adaptive Learning Rules (40 marks). Marks awarded for completeness of implementation, experimental methodology, experimental results.
- Presentation and clarity of report (25 marks). Marks awarded for overall structure, clear and concise presentation, providing enough information to enable work to be reproduced, clear and concise presentation of results, informative discussion and conclusions.
- Additional Excellence (10 marks). Marks awarded for significant personal insight, creativity, originality, and/or extra depth and academic maturity.