Machine Learning Practical: Coursework 1

Release date: Monday 16th October 2017
Due date: 16:00 Monday 30th October 2017

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. **You should complete the first three lab notebooks before starting the coursework.** The aim of the coursework is to investigate variants of the ReLU activation function for hidden units in multi-layer networks, with respect to the validation set accuracies achieved by the trained models.

Code

You should run all of the experiments for the coursework inside the Conda environment you set up in first labs. The code for the coursework is available on the course Github repository on a branch `mlp2017-8/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 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/mlp2017-8/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.

There will also be a `report` directory which contains the LaTeX template and style files for the report. You should copy all these files into the directory which will contain your report.

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 of those changes.

In these experiments you should use a multi-layer network with two hidden layers (corresponding to three affine transformations) and a softmax output layer. The initial baseline should use a sigmoid activation function for the hidden layer; other experiments will explore different nonlinear activation functions. The hidden layers should each contain 100 hidden units. The baseline network can this be defined with the following code (which should be familiar to you from Lab 3):

```python
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 the Glorot initialisation scheme, discussed in lecture 4. In part 2B of this coursework you will explore the effect of different initialisation schemes.

The above code creates a network using sigmoid hidden layers; you should modify it to also create a network using ReLU activation functions (see Lab 3). These two networks will form your baseline systems.

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: Implementing Activation Functions

In the first part of the assignment you will implement three further activation functions, each of which is related to ReLU [Nair and Hinton, 2010]: Leaky ReLU, ELU (Exponential Linear Unit), and SELU (Scaled Exponential Linear Unit). Each of these units defines an activation function for which $f(x) = x$ when $x > 0$, as for ReLU, but avoid having a zero gradient when $x < 0$.

Leaky ReLU (lrelu(x)) [Maas et al., 2013] has the following form:

$$\text{lrelu}(x) = \begin{cases} \alpha x & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases} \tag{1}$$

Where α is a constant; typically $\alpha = 0.01$, and you can use this value in this coursework. Note that α can be taken to be a parameter which is learned by back-propagation along with the weights and biases – this is called Parametric ReLU (PReLU).

ELU (elu(x)) [Clevert et al., 2015] has the following form:

$$\text{elu}(x) = \begin{cases} \alpha(\exp(x) - 1) & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases} \tag{2}$$

Again α can be taken as a constant or a tunable parameter. Typically $\alpha = 1$, which results in a smooth function, and you can use this value in this coursework.

SELU (selu(x)) [Klambauer et al., 2017] has the following form:

$$\text{selu}(x) = \lambda \begin{cases} \alpha(\exp(x) - 1) & \text{if } x \le 0 \\ x & \text{if } x > 0 \end{cases} \tag{3}$$

In the case of SELU, there is a theoretical argument for optimal values of the two parameters: $\alpha \approx 1.6733$ and $\lambda \approx 1.0507$, and you can use these values in this coursework.

1. Implement each of these activations function as classes `LeakyReluLayer`, `EluLayer` and `SeluLayer`. You need to implement `fprop` and `bprop` methods for each class.
2. Verify the correctness of your implementation using the supplied unit tests in `Activation_Tests.ipynb`
3. Automatically create a test file `sXXXXXXX_test_file.txt`, by running the provided program `generate_inputs.py` which uses your code for `LeakyReluLayer`, `EluLayer` and `SeluLayer` to run your fprop and bprop methods for each layer on a unique test vector generated using your student ID number.

For Part 1 of the coursework you need to submit the test file `sXXXXXXX_test_file.txt` (where `sXXXXXXX` is replaced with your student number) created in step 3 above.

Part 2: MNIST Experiments

In Part 2 of the coursework you will experiment with `LeakyReluLayer`, `EluLayer` and `SeluLayer` in multi-layer networks trained on MNIST.

2A: Comparing activation functions

In this sub-part you should compare the behaviour of Leaky ReLU, ELU, and SELU activation functions on the MNIST task. Carry out all experiments using 2 hidden layers, with 100 units per hidden layer. You should compare the results with baseline systems of the same architecture using sigmoid units and using ReLU units.

2B: Deep neural network experiments

In this subpart you will explore the behaviour of deeper networks. Based on the results of Part 2A, choose one activation function, and compare networks with 2–8 hidden layers, using 100 hidden units per hidden layer.

Also compare the effect of different initialisation strategies, as discussed in lecture 4. First look at the effect of weight initialisation based on

- Fan-in: $w_i \sim U(-\sqrt{3/n_{in}}, \sqrt{3/n_{in}}$
- Fan-out: $w_i \sim U(-\sqrt{3/n_{out}}, \sqrt{3/n_{out}}$
- Fan-in and Fan-out: $w_i \sim U\left(-\sqrt{6/(n_{in} + n_{out})}, \sqrt{6/(n_{in} + n_{out})}\right)$

where U is the uniform distribution. The first of these corresponds to constraining the estimated variance of a unit to be independent of the number of incoming connections (n_{in}); the second to constraining the estimated variance of a unit's gradient to be independent of the number of outgoing connections (n_{out}); the third corresponds to Glorot and Bengio's combined initialisation.

Additionally you could also explore the effect of drawing from a Gaussian distribution compared with a uniform distribution. In particular you might like to explore initialising a SELU layer drawing from a Gaussian with mean 0 and variance $1/n_{in}$ as recommended by Klambauer et al. [2017].

For Part 2 of the coursework you need to write and submit a report, using the template provided, in the directory `report`. Please read the template document `mlp-cw1-template.pdf` very carefully, as it provides advice and instructions on writing your report. You can use the LaTeX source file `mlp-cw1-template.tex` as a template for your report (see below, in the section 'Report').

It is highly recommended that you use LaTeX for your report. If you have not used LaTeX previously, now is a good time to learn how to use it!

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.

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.

Report

Part two of your coursework submission, worth 70 marks will be a report. The directory `coursework1/report` contains a template for your report (`mlp-cw1-template.txt`); the generated pdf file (`mlp-cw1-template.pdf`) is also provided, and you should read this file carefully as it contains information about the required structure and experimentation. 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 6 pages long**, with a further page for references. We will not read or assess any parts of the report beyond the allowed 6+1 pages.

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.)

However to emphasise again: **It is highly recommended that you use LaTeX.**

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
```

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.

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 afterward. (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, and (even worse) you may still be penalized for submitting late because the timestamp will update.

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.

Submission

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

Your submission should include

- the unit test file generated in part 1, `sXXXXXXX_test_file.txt`, where your student number replaces `sXXXXXXX`
- your completed report as a PDF file, using the provided template
- the notebook (`.ipynb`) file you used to run the experiments in
- and your local version of the `mlp` code including any changes you made to the modules (`.py` files).

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

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

and then submit this directory using

```
submit mlp cw1 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.

Marking Scheme

- Part 1, Activation function implementation (30 marks). Based on your submitted test file.
- Part 2, Report (70 marks). The following aspects will contribute to the mark for your report:
  - Abstract - how clear is it? does it cover what is reported in the document
  - Introduction - do you clear outline and motivate the paper, and describe the research questions investigated?
  - Description of activation functions – is it clear and correct?
  - Experiments – did you carry out the experiments correctly? are the results clearly presented and described?
  - Interpretation and discussion of results
  - Conclusions
  - Presentation and clarity of report

References

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (ELUs). *arXiv preprint arXiv:1511.07289*, 2015.

Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. *arXiv preprint arXiv:1706.02515*, 2017.

Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. ICML*, volume 30, 2013.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted Boltzmann machines. In *Proc ICML*, pages 807–814, 2010.