COMP9444 Neural Networks and Deep Learning

Term 3, 2024

Assignment – Characters and Hidden Unit Dynamics

Due: Tuesday 15 October, 5:00 pm
Marks: 17% of final assessment

In this assignment, you will be implementing and training neural networks for three different tasks, and analysing the results. You are to submit two Python files `kuzu.py` and `check.py`, as well as a written report `hw1.pdf` (in `pdf` format).

Provided Files

Copy the archive `hw1.zip` into your own filespace and unzip it. This should create a directory `hw1`, subdirectories `net` and `plot`, and eight Python files `kuzu.py`, `check.py`, `kuzu_main.py`, `check_main.py`, `seq_train.py`, `seq_models.py`, `seq_plot.py` and `anb2n.py`.

Your task is to complete the skeleton files `kuzu.py` and `check.py` and submit them, along with your report.

Part 1: Japanese Character Recognition

For Part 1 of the assignment you will be implementing networks to recognize handwritten Hiragana symbols. The dataset to be used is Kuzushiji-MNIST or KMNIST for short. The paper describing the dataset is available here. It is worth reading, but in short: significant changes occurred to the language when Japan reformed their education system in 1868, and the majority of Japanese today cannot read texts published over 150 years ago. This paper presents a dataset of handwritten, labeled examples of this old-style script (Kuzushiji). Along with this dataset, however, they also provide a much simpler one, containing 10 Hiragana characters with 7000 samples per class. This is the dataset we will be using.

Text from 1772 (left) compared to 1900 showing the standardization of written Japanese.

1. [1 mark] Implement a model `NetLin` which computes a linear function of the pixels in the image, followed by log softmax. Run the code by typing:

   ```
   python3 kuzu_main.py --net lin
   ```

Copy the final accuracy and confusion matrix into your report. The final accuracy should be around 70%. Note that the **rows** of the confusion matrix indicate the target character, while the **columns** indicate the one chosen by the network. (0="o", 1="ki", 2="su", 3="tsu", 4="na", 5="ha", 6="ma", 7="ya", 8="re", 9="wo"). More examples of each character can be found here.

2. [1 mark] Implement a fully connected 2-layer network `NetFull` (i.e. one hidden layer, plus the output layer), using tanh at the hidden nodes and log softmax at the output node. Run the code by typing:

   ```
   python3 kuzu_main.py --net full
   ```

   Try different values (multiples of 10) for the number of hidden nodes and try to determine a value that achieves high accuracy (at least 84%) on the test set. Copy the final accuracy and confusion matrix into your report, and include a calculation of the total number of independent parameters in the network.

3. [1 mark] Implement a convolutional network called `NetConv`, with two convolutional layers plus one fully connected layer, all using relu activation function, followed by the output layer, using log softmax. You are free to choose for yourself the number and size of the filters, metaparameter values (learning rate and momentum), and whether to use max pooling or a fully convolutional architecture. Run the code by typing:

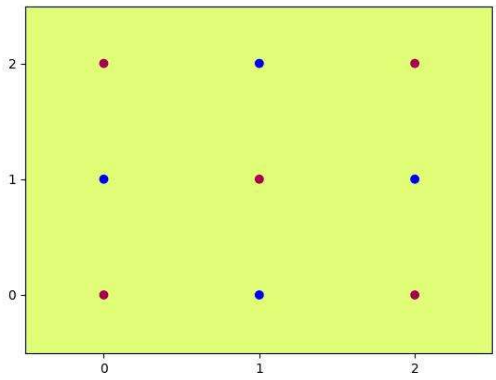
   ```
   python3 kuzu_main.py --net conv
   ```

   Your network should consistently achieve at least 93% accuracy on the test set after 10 training epochs. Copy the final accuracy and confusion matrix into your report, and include a calculation of the total number of independent parameters in the network.

4. [3 marks] Briefly discuss the following points:
   a. the relative accuracy of the three models,
   b. the number of independent parameters in each of the three models,
   c. the confusion matrix for each model: which characters are most likely to be mistaken for which other characters, and why?

Part 2: Multi-Layer Perceptron

In Part 2 you will be exploring 2-layer neural networks (either trained, or designed by hand) to classify the following data:



1. [1 mark] Train a 2-layer neural network with 5 hidden nodes, using sigmoid activation at both the hidden and output layer, on the above data, by typing:

```
python3 check_main.py --act sig --hid 5
```

You may need to run the code a few times, until it achieves accuracy of 100%. If the network appears to be stuck in a local minimum, you can terminate the process with ⟨ctrl⟩-C and start again. You are free to adjust the learning rate and the number of hidden nodes, if you wish (see code for details). The code should produce images in the `plot` subdirectory graphing the function computed by each hidden node (`hid_5_?.jpg`) and the network as a whole (`out_5.jpg`). Copy these images into your report.

2. [2 marks] Design by hand a 2-layer neural network with 4 hidden nodes, using the Heaviside (step) activation function at both the hidden and output layer, which correctly classifies the above data. Include a diagram of the network in your report, clearly showing the value of all the weights and biases. Write the equations for the dividing line determined by each hidden node. Create a table showing the activations of all the hidden nodes and the output node, for each of the 9 training items, and include it in your report. You can check that your weights are correct by entering them in the section of `check.py` where it says "Enter Weights Here", and typing:

   ```
   python3 check_main.py --act step --hid 4 --set_weights
   ```

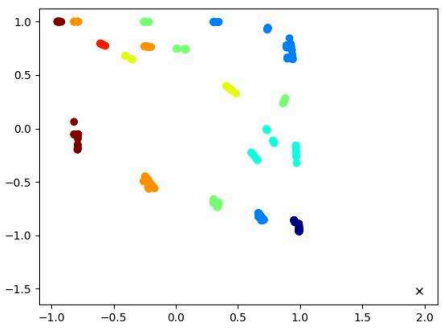
3. [1 mark] Now rescale your hand-crafted weights and biases from Part 2 by multiplying all of them by a large (fixed) number (for example, 10) so that the combination of rescaling followed by sigmoid will mimic the effect of the step function. With these re-scaled weights and biases, the data should be correctly classified by the sigmoid network as well as the step function network. Verify that this is true by typing:

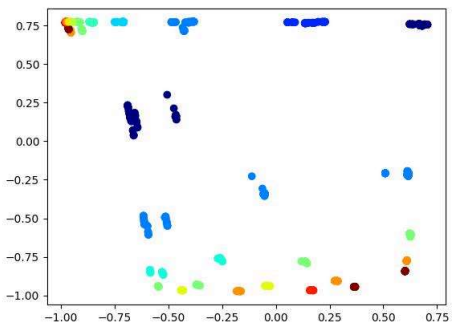
   ```
   python3 check_main.py --act sig --hid 4 --set_weights
   ```

   Once again, the code should produce images in the `plot` subdirectory showing the function computed by each hidden node (`hid_4_?.jpg`) and the network as a whole (`out_4.jpg`). Copy these images into your report, and be ready to submit `check.py` with the (rescaled) weights as part of your assignment submission.

Part 3: Hidden Unit Dynamics for Recurrent Networks

In Part 3 you will be training and analysing the hidden unit dynamics of recurrent networks trained on two language prediction tasks, $a^n b^{2n}$ and $a^n b^{2n} c^{3n}$, using the supplied code `seq_train.py`, `seq_plot.py` and `anb2n.py`.





1. [1 mark] Train a Simple Recurrent Network (SRN) with 2 hidden nodes on the $a^n b^{2n}$ language prediction task by typing:

   ```
   python3 seq_train.py --lang anb2n
   ```

The $a^n b^{2n}$ language is a concatenation of a random number of A's followed by exactly twice that same number of B's. The generator produces concatenations of sequences $a^n b^{2n}$ for values of n between 1 and 4. This SRN has 2 inputs, 2 hidden nodes and 2 outputs.

Look at the predicted probabilities of A and B as the training progresses. The first B in each sequence and all A's after the first A are non-deterministic, meaning that they can only be predicted in a probabilistic sense. But, if the training is successful, all other symbols should be correctly predicted. In particular, the network should predict the last B in each sequence as well as the subsequent A (at the beginning of the next sequence). The loss should stay consistently below 0.05 (you might need to run the code a couple of times in order to achieve this). After the network has been trained successfully, plot the hidden unit activations at epoch 100000 by typing:

```
python3 seq_plot.py --lang anb2n --epoch 100
```

This should produce text output showing the hidden unit activations and associated probabilities, as each sequence is processed; it should also produce an image in the `plot` subdirectory called `anb2n_srn2_01.jpg`. After examining the hidden unit activations, annotate this image (either electronically, or with a pen on a printout) by grouping clusters of points together to form "states". Draw a boundary around each state, draw arrows between the states and label each arrow as 'A' or 'B'. The initial state is the one which includes the cross '×'. In order to assist with this task, we have tried to color the dots in a logical way using this colormap:

Include the annotated image in your report.

2. [1 mark] Draw a picture of a finite state machine, using circles and arrows, which is equivalent to the finite state machine in your annotated image from Step 1.

3. [1 mark] Briefly explain how the network accomplishes the $a^n b^{2n}$ task. Specifically, you should describe how the hidden unit activations change as the string is processed, and how it is able to correctly predict all B's after the first B, as well as the initial A following the last B in the sequence.

4. [1 mark] Train an LSTM with 3 hidden nodes on the $a^n b^{2n} c^{3n}$ language prediction task by typing:

   ```
   python3 seq_train.py --lang anb2nc3n --model lstm --hid 3
   ```

   The $a^n b^{2n} c^{3n}$ language is a concatenation of a random number of A's, followed by exactly twice that same number of B's, followed by three times that number of C's.

   The loss should stay consistently below 0.02, and the network should correctly predict the non-initial B's, all the C's, and the subsequent A following the last C. You might need to run the code a couple of times in order to achieve this. If the network appears to have already learned the task successfully, or if it seems to be stuck in a local minimum, you can terminate the process with ⟨ctrl⟩-C (and start again, if necessary). After the network has been trained successfully, plot the hidden unit activations at epoch 50000 by typing:

```
python3 seq_plot.py --lang anb2nc3n --model lstm --epoch 50
```

(you can choose a different epoch number, if you wish). This should produce three images from different angles, labeled `anb2nc3n_lstm3_??.jpg`, as well as an interactive 3D plot which you can rotate to a visually appealing angle and save as an image. Copy these images into your report.

5. [3 marks] This question is intended to be a bit more challenging. By annotating the generated images from Step 4 (or others of your own choosing), try to analyse the dynamics of the hidden and/or context units, and explain how the LSTM successfully accomplishes the $a^n b^{2n} c^{3n}$ prediction task (this might involve modifying the code so that it returns and prints out the context units as well as the hidden units).

Submission

You should submit by typing

```
give cs9444 hw1 kuzu.py check.py hw1.pdf
```

You can submit as many times as you like — later submissions will overwrite earlier ones. You can check that your submission has been received by using the following command:

9444 classrun -check hw1

The submission deadline is Tuesday 15 October, 5:00 pm. In accordance with UNSW-wide policies, 5% penalty will be applied for every 24 hours late after the deadline, up to a maximum of 5 days, after which submissions will not be accepted.

Additional information may be found in the FAQ and will be considered as part of the specification for the project. You should check this page regularly.

Plagiarism Policy

Group submissions will not be allowed for this assignment. Your code and report must be entirely your own work. Plagiarism detection software will be used to compare all submissions pairwise (including submissions for similar assignments from previous offering, if appropriate) and serious penalties will be applied, particularly in the case of repeat offences.

DO NOT COPY FROM OTHERS; DO NOT ALLOW ANYONE TO SEE YOUR CODE

Please refer to the UNSW Policy on Academic Integrity and Plagiarism if you require further clarification on this matter.

Good luck!
