COMP9444 Neural Networks and Deep Learning

Term 3, 2021

Project 1 – Characters, Fractals and Hidden Unit Dynamics

Due: Monday 25 October, 23:59 pm
Marks: 30% of final assessment

In this assignment, you will be implementing and training various neural network models for four different tasks, and analysing the results.

You are to submit three Python files `kuzu.py`, `frac.py` and `encoder.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` with the data file `fractal.csv`, subdirectories `plot` and `net`, as well as eleven Python files `kuzu.py`, `frac.py`, `encoder.py`, `kuzu_main.py`, `frac_main.py`, `encoder_main.py`, `encoder_model.py`, `seq_train.py`, `seq_models.py`, `seq_plot.py`, `reber.py` and `anbn.py`.

Your task is to complete the skeleton files `kuzu.py`, `frac.py`, `encoder.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 marks] 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

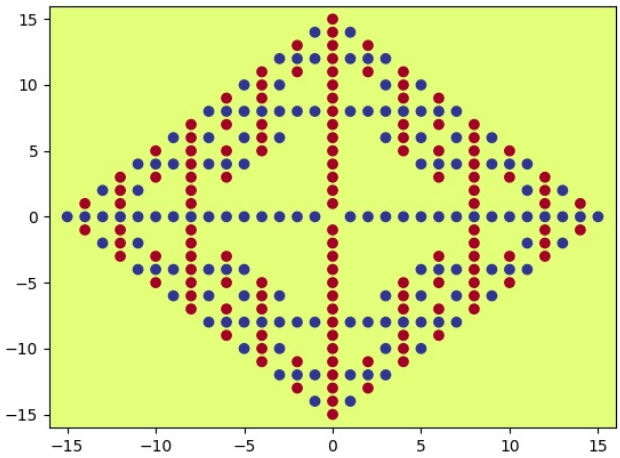
and size of the filters, metaparameter values (learning rate and momentum), and whether to use max pooling or a fully convolutional architecture. Run the code by typing:

```
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: Fractal Classification Task



For Part 2 you will be training a network to distinguish dots in the fractal pattern shown above. The supplied code `frac_main.py` loads the training data from `fractal.csv`, applies the specified neural network model and produces a graph of the resulting function, along with the data. For this task there is no test set as such, but we instead judge the generalization by plotting the function computed by the network and making a visual assessment.

1. [1 mark] Provide code for a Pytorch Module called `Full2Net` which implements a 3–layer fully connected neural network with two hidden layers using `tanh` activation, followed by the output layer with one node and `sigmoid` activation. Your network should have the same number of hidden nodes in each layer, specified by the variable `hid`. The hidden layer activations (after applying `tanh`) should be stored into `self.hid1` and `self.hid2` so they can be graphed afterwards.

2. [1 mark] Train your network by typing

   ```
   python3 frac_main.py --net full2 --hid (hid)
   ```

   Try to determine a number of hidden nodes close to the mininum required for the network to be trained successfully (although, it need not be the absolute minimum). You may need to run the network several times before finding a set of initial weights which allows it to converge. (If it trains for a couple of minutes and seems to be stuck in a local minimum, kill it with ⟨cntrl⟩–c and run it again). You are free to adjust the learning rate and initial weight size, if you want to. The `graph_output()` method will generate a picture of the function computed by your Network and store it in the `plot` subdirectory with a name like `out_full2_?.png`. You should include this picture in your report, as well as a calculation of the total number of independent parameters in your network (based on the number of hidden nodes you have chosen).

3. [1 mark] Provide code for a Pytorch Module called `Full3Net` which implements a 4–layer network, the same as `Full2Net` but with an additional hidden layer. All three hidden layers should have the same number of nodes (`hid`). The hidden layer activations (after applying `tanh`) should be stored into `self.hid1`, `self.hid2` and `self.hid3`.

4. [1 mark] Train your 4–layer network by typing

   ```
   python3 frac_main.py --net full3 --hid (hid)
   ```

   Try to determine a number of hidden nodes close to the mininum required for the network to be trained successfully. Keep in mind that the loss function might decline initially, appear to stall for several epochs, but then continue to decline. The `graph_output()` method will generate a picture of the function computed by your Network and store it in the `plot` subdirectory with a name like `out_full3_?.png`, and the `graph_hidden()` method should generate plots of all the hidden nodes in all three hidden layers, with names like `hid_full3_?_?_?.png`. You should include the plot of the output and the plots of all the hidden units in all three layers in your report, as well as a calculation of the total number of independent parameters in your network.

5. [2 mark] Provide code for a Pytorch Module called `DenseNet` which implements a 3-layer densely connected neural network. Your network should be the same as `Full2Net` except that it should also include shortcut connections from the input to the second hidden layer and output layer, and from the first hidden layer to the second hidden layer and output layer. Each hidden layer should have `hid` units and `tanh` activation, and the output node should have `sigmoid` activation. The hidden layer activations (after applying `tanh`) should be stored into `self.hid1` and `self.hid2`. Specifically, the hidden and output activations should be calculated according to the following equations. (Note that there are various ways to implement these equations in PyTorch; for example, using a separate `nn.Parameter` for each individual bias and weight matrix, or combining several of them into `nn.Linear` and making use of `torch.cat`()).

$$h^1_j = \tanh(b^1_j + \Sigma_k w^{10}_{jk} x_k)$$

$$h^2_i = \tanh(b^2_i + \Sigma_k w^{20}_{ik} x_k + \Sigma_j w^{21}_{ij} h^1_j)$$

$$out = \text{sigmoid}(b^{out} + \Sigma_k w^{30}_{k} x_k + \Sigma_j w^{31}_{j} h^1_j + \Sigma_i w^{32}_{i} h^2_i)$$

6. [1 mark] Train your Dense Network by typing

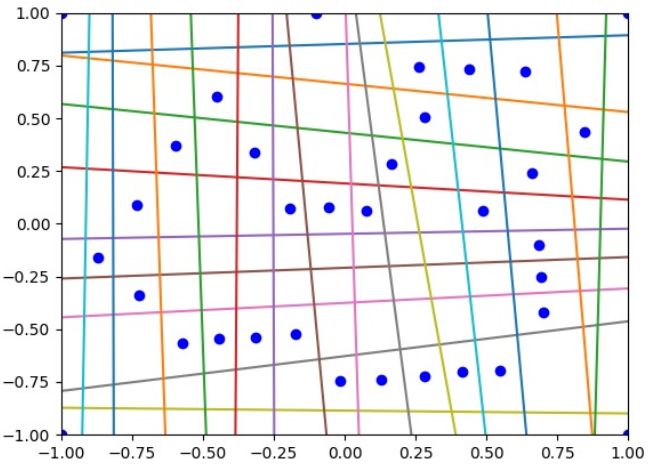
   ```
   python3 frac_main.py --net dense --hid (hid)
   ```

   As before, try to determine a number of hidden nodes close to the mininum required for the network to be trained successfully. You should include the graphs of the output and all the hidden nodes in both layers in your report, as well as a calculation of the total number of independent parameters in your network.

7. [3 marks] Briefly discuss the following points:
   a. the total number of independent parameters in each of the three networks (using the number of hidden nodes determined by your experiments) and the approximate number of epochs required to train each type of network,
   b. a qualitative description of the functions computed by the different layers of `Full3Net` and `DenseNet`,
   c. the qualitative difference, if any, between the overall function (i.e. output as a function of input) computed by the three networks.

Part 3: Encoder Networks

In Part 3 you will be editing the file `encoder.py` to create a dataset which, when run in combination with `encoder_main.py`, produces the following image (which is intended to be a stylized map of mainland China).



You should first run the code by typing

```
python3 encoder_main.py --target star16
```

Note that target is determined by the tensor `star16` in `encoder.py`, which has 16 rows and 8 columns, indicating that there are 16 inputs and 8 outputs. The inputs use a one-hot encoding and are generated in the form of an identity matrix using `torch.eye()`

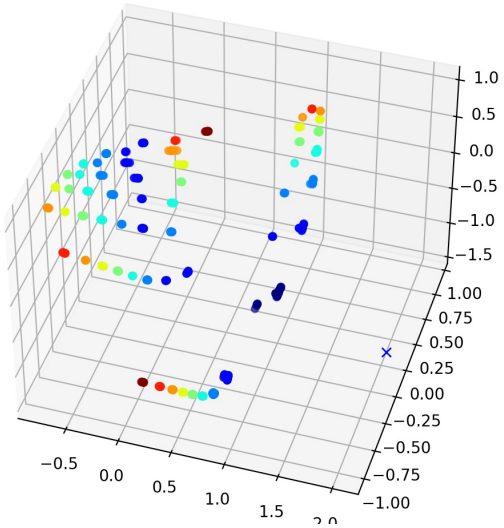
1. [2 marks] Create by hand a dataset in the form of a tensor called `ch34` in the file `encoder.py` which, when run with the following command, will produce an image essentially the same as the one shown above (but possibly rotated or reflected).

   ```
   python3 encoder_main.py --target ch34
   ```

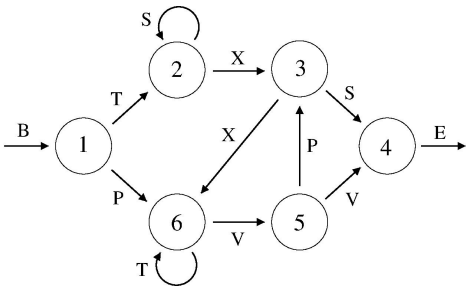
   The pattern of dots and lines must be identical, except for the possible rotation or reflection. Note in particular the five "anchor points" in the corners and on the edge of the figure.

   Your tensor should have 34 rows and 23 columns. Include the final image in your report, and include the tensor `ch34` in your file `encoder.py`

Part 4: Hidden Unit Dynamics for Recurrent Networks



In Part 4 you will be investigating the hidden unit dynamics of recurrent networks trained on language prediction tasks, using the supplied code `seq_train.py` and `seq_plot.py`.



1. [2 marks] Train a Simple Recurrent Network (SRN) on the Reber Grammar prediction task by typing

   ```
   python3 seq_train.py --lang reber
   ```

   This SRN has 7 inputs, 2 hidden units and 7 outputs. The trained networks are stored every 10000 epochs, in the `net` subdirectory. After the training finishes, plot the hidden unit activations at epoch 50000 by typing

   ```
   python3 seq_plot.py --lang reber --epoch 50
   ```

   The dots should be arranged in discernable clusters by color. If they are not, run the code again until the training is successful. The hidden unit activations are printed according to their "state", using the colormap "jet":

   Based on this colormap, annotate your figure (either electronically, or with a pen on a printout) by drawing a circle around the cluster of points corresponding to each state in the state machine, and drawing arrows between the states, with each arrow labeled with its corresponding symbol. Include the annotated figure in your report.

2. [1 mark] Train an SRN on the $a^n b^n$ language prediction task by typing

   ```
   python3 seq_train.py --lang anbn
   ```

   The $a^n b^n$ language is a concatenation of a random number of A's followed by an equal number of B's. The SRN has 2 inputs, 2 hidden units 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 not deterministic and 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. The error should be consistently in the range of 0.01 or 0.02. If the network appears to have learned the task successfully, you can stop it at any time using ⟨cntrl⟩−c. If it appears to be stuck in a local minimum, you can stop it and run the code again until it is successful.

   After the training finishes, plot the hidden unit activations by typing

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

   Include the resulting figure in your report. The states are again printed according to the colormap "jet". Note, however, that these "states" are not unique but are instead used to count either the number of A's we have seen or the number of B's we are still expecting to see.

3. [2 marks] Briefly explain how the $a^n b^n$ prediction task is achieved by the network, based on the figure you generated in Question 2. Specifically, you should describe how the hidden unit activations change as the string is processed, and how it is able to correctly predict the last B in each sequence as well as the following A.

4. [1 mark] Train an SRN on the $a^n b^n c^n$ language prediction task by typing

   ```
   python3 seq_train.py --lang anbncn
   ```

   The SRN now has 3 inputs, 3 hidden units and 3 outputs. Again, the "state" is used to count up the A's and count down the B's and C's. Continue training (re-starting, if necessary) until the network is able to reliably predict all the C's as well as the subsequent A, and the error is consistently in the range of 0.01 or 0.02.

   After the training finishes, plot the hidden unit activations by typing

   ```
   python3 seq_plot.py --lang anbncn --epoch 200
   ```

   Rotate the figure in 3 dimensions to get one or more good view(s) of the points in hidden unit space.

5. [2 marks] Briefly explain how the $a^n b^n c^n$ prediction task is achieved by the network, based on the figure you generated in Question 4. Specifically, you should describe how the hidden unit activations change as the string is processed, and how it is able to correctly predict the last B in each sequence as well as all of the C's and the following A.

6. [4 marks] This question is intended to be more challenging. Train an LSTM network to predict the Embedded Reber Grammar, by typing

   ```
   python3 seq_train.py --lang reber --embed True --model lstm --hid 4
   ```

   You can adjust the number of hidden nodes if you wish. Once the training is successful, try to analyse the behavior of the LSTM and explain how the task is accomplished (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 frac.py encoder.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

The submission deadline is Monday 25 October, 23:59. 15% penalty will be applied to the (maximum) mark for every 24 hours late after the deadline.

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!
