COMP9444 Neural Networks and Deep Learning

Term 3, 2020

Project 1 - Characters, Spirals and Hidden Unit Dynamics

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

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

You are to submit three Python files `kuzu.py`, `spiral.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 `spirals.csv` as well as seven Python files `kuzu.py`, `spiral.py`, `encoder.py`, `kuzu_main.py`, `spiral_main.py`, `endoder_main.py` and `encoder_model.py`.

Your task is to complete the skeleton files `kuzu.py`, `spiral.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`, 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.

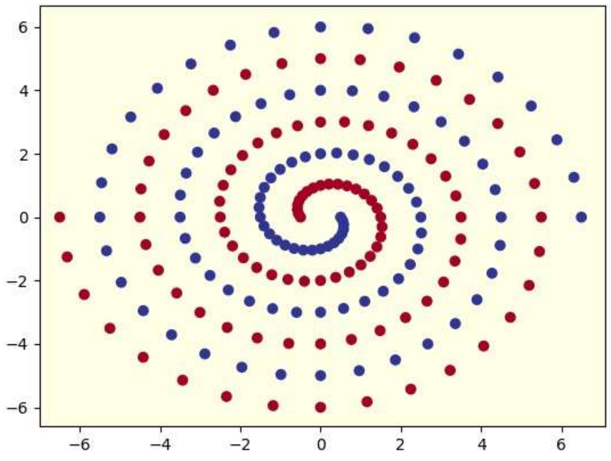
3. [2 marks] Implement a convolutional network called `NetConv`, with two convolutional layers plus one fully connected layer, all using relu activation function, followed by the output layer, using log softmax. You are free to choose for yourself the number and size of the filters, metaparameter values (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.

4. [5 marks] Discuss what you have learned from this exercise, including the following points:
   a. the relative accuracy of the three models,
   b. the confusion matrix for each model: which characters are most likely to be mistaken for which other characters, and why?
   c. you may wish to experiment with other architectures and/or metaparameters for this dataset, and report on your results; the aim of this exercise is not only to achieve high accuracy but also to understand the effect of different choices on the final accuracy.

Part 2: Twin Spirals Task



For Part 2 you will be training on the famous Two Spirals Problem (Lang and Witbrock, 1988). The supplied code `spiral_main.py` loads the training data from `spirals.csv`, applies the specified 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 `PolarNet` which operates as follows: First, the input `(x,y)` is converted to polar co-ordinates `(r,a)` with `r=sqrt(x*x + y*y)`, `a=atan2(y,x)`. Next, `(r,a)` is fed into a fully connected neural network with one hidden layer using `tanh` activation, followed by a single output using `sigmoid` activation. The conversion to polar coordinates should be included in your `forward()` method, so that the Module performs the entire task of conversion followed by network layers.

2. [1 mark] Run the code by typing

   ```
   python3 spiral_main.py --net polar --hid 10
   ```

   Try to find the minimum number of hidden nodes required so that this PolarNet learns to correctly classify all of the training data within 20000 epochs, on almost all runs. The `graph_output()` method will generate a picture of the function computed by your PolarNet called `polar_out.png`, which you should include in your report.

3. [1 mark] Provide code for a Pytorch Module called `RawNet` which operates on the raw input `(x,y)` without converting to polar coordinates. Your network should consist of two fully connected hidden layers with tanh activation, plus the output layer, with sigmoid activation. The two hidden layers should each have the same number of hidden nodes, determined by the parameter `num_hid`.

   Note that this network differs from the one depicted in slide 10 of lecture slides 3a on Hidden Unit Dynamics, in two ways: Firstly, only two hidden nodes are shown in the diagram, but a larger number of hidden nodes are needed for the task. Secondly, our network will not have shortcut connections; although these appeared to be necessary in the original 1988 paper using SGD, it now appears that the task can be learned without shortcut connections, with the help of the Adam optimizer.

4. [1 mark] Run the code by typing

   ```
   python3 spiral_main.py --net raw
   ```

   Try to choose a value for the number of hidden nodes (--hid) and the size of the initial weights (--init) such that this RawNet learns to correctly classify all of the training data within 20000 epochs, on almost all runs. Include in your report the number of hidden nodes, and the values of any other metaparameters. The `graph_output()` method will generate a picture of the function computed by your RawNet called `raw_out.png`, which you should include in your report.

5. [2 marks] Using `graph_output()` as a guide, write a method called `graph_hidden(net, layer, node)` which plots the activation (after applying the `tanh` function) of the hidden node with the specified number `(node)` in the specified `layer` (1 or 2). (Note: if `net` is

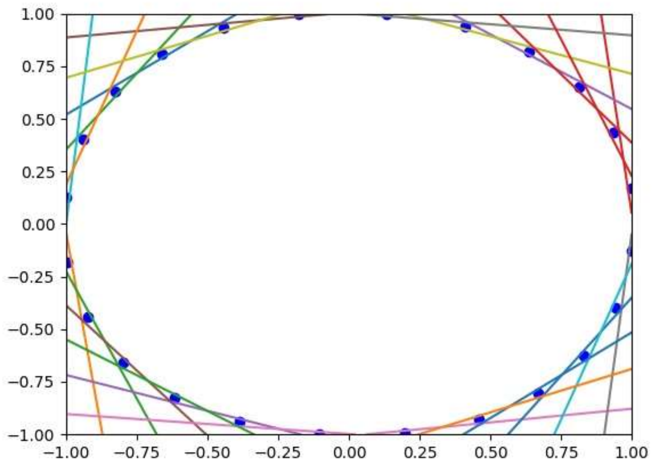
of type `PolarNet`, `graph_output()` only needs to behave correctly when layer is 1).

Hint: you might need to modify `forward()` so that the hidden unit activations are retained, i.e. replace `hid1 = torch.tanh(...)` with `self.hid1 = torch.tanh(...)`

Use this code to generate plots of all the hidden nodes in PolarNet, and all the hidden nodes in both layers of RawNet, and include them in your report.

6. [5 marks] Discuss what you have learned from this exercise, including the following points:
   a. the qualitative difference between the functions computed by the hidden layer nodes PolarNet and RawNet, and a brief description of how the network uses these functions to achieve the classification
   b. the effect of different values for initial weight size on the speed and success of learning for RawNet
   c. you may like to also experiment with other changes and comment on the result - for example, changing batch size from 97 to 194, using SGD instead of Adam, changing tanh to relu, adding a third hidden layer, etc.

Part 3: Hidden Unit Dynamics



In Part 3 you will be investigating hidden unit dynamics, as described in lecture slides and video 3a, using the supplied code `encoder_main.py` and `encoder_model.py` as well as `encoder.py` (which you should modify and submit).

1. [1 mark] Run the code by typing

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

   Save the final image and include it in your report. 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()`

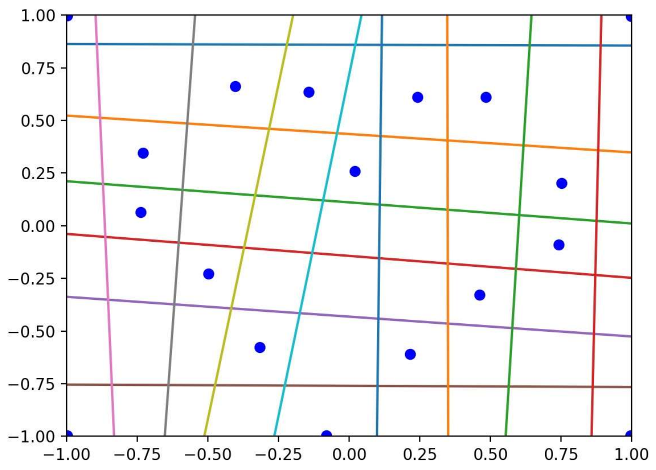
2. [3 marks] Run the code by typing

   ```
   python3 encoder_main.py --target=input --dim=9 --plot
   ```

   In this case, the task is a 9-2-9 encoder with both input and target determined by a one-hot encoding. Save the first eleven images generated (epochs 50 to 3000), plus the final image, and include them in your report. Describe in words how the hidden unit activations (dots) and output boundaries (lines) move as the training progresses.

3. [2 marks] Create by hand a dataset in the form of a tensor called `heart18` in the file `encoder.py` which, when run with the following command, will produce an image essentially the same as the heart shaped figure shown below (but possibly rotated). Your tensor should have 18 rows and 14 columns. Include the final image in your report, and include the tensor `heart18` in your file `encoder.py`

   ```
   python3 encoder_main.py --target=heart18
   ```



4. [4 marks] Create training data in tensors `target1` and `target2`, which will generate two images of your own design, when run with the command

   ```
   python3 encoder_main.py --target=target1
   ```

   (and similarly for `target2`). You are free to choose the size of the tensors, and to adjust parameters such as `--epochs` and `--lr` in order to achieve successful learning. Marks will be awarded based on creativity and artistic merit. Include the final images in your report, and include the tensors `target1` and `target2` in your file `encoder.py`

Submission

You should submit by typing

```
give cs9444 hw1 kuzu.py spiral.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 Sunday 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 program must be entirely your own work. Plagiarism detection software will be used to compare all submissions pairwise 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!