Report challenge three

Andrea Tonello

20/05/2024

Our goal is to explore the capabilities and performance, measured using accuracy, of Fully Connected Neural Networks and Convolutional Neural Networks, by tuning hyperparameters, optimizers and hidden layers.

1 Data Exploration

The data comes from the KMNIST dataset and it consists of consists of 70000 28x28 pixel, 1 channel images of hiragana symbols; 60000 are reserved for the training set and 10000 for the test set. The symbols are grouped in 10 classes:

```
Classes:  ['o', 'ki', 'su', 'tsu', 'na', 'ha', 'ma', 'ya', 're', 'wo']
```

Here is a sample of the data.

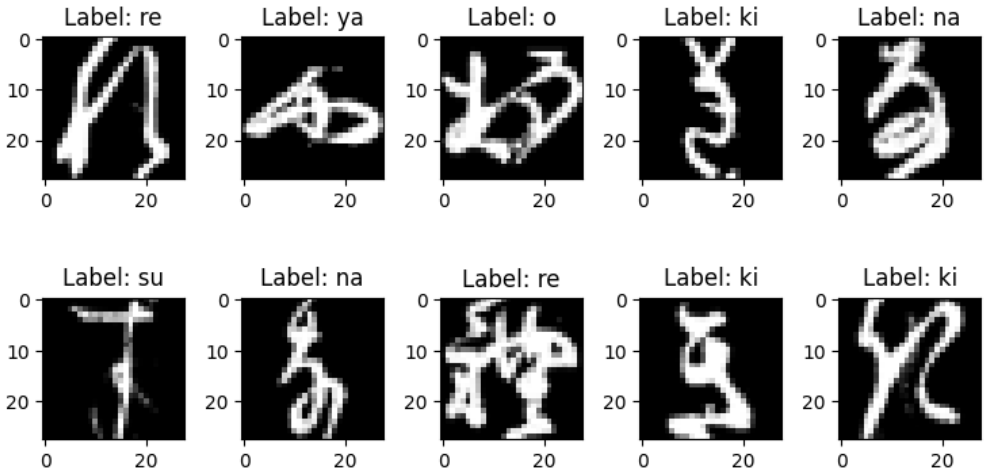

Figure 1: *Sample of the images from the train set*

2 Model building

We choose to implement three models: a Fully Connected NN with 1 hidden layer, a Convolutional NN with 1 hidden layer, and another Convolutional NN this time with 3 hidden layers. A batch size of 64 is chosen for our architectures. More in detail:

- FullyConnected_1:
  - 1st layer (linear): input → **B**atch **S**ize$\times 1 \times 28 \times 28$, output → $BS \times 32 \times 26 \times 26$
  - 2nd layer (linear): input → $BS \times 32 \times 26 \times 26$, output → $BS \times 128$
  - 3rd layer (linear): input → $BS \times 128$, output → $BS \times 10$
- Convolutional_1:
  - 1st layer (convolutional): input → $BS \times 1 \times 28 \times 28$, output → $BS \times 32 \times 24 \times 24$ with a kernel size of 5 and a 2x2 max-pooling
  - 2nd layer (linear): input → $BS \times 32 \times 12 \times 12$, output → $BS \times 128$
  - 3rd layer (linear): input → $BS \times 128$, output → $BS \times 10$
- Convolutional_3:
  - 1st layer (convolutional): input → $BS \times 1 \times 28 \times 28$, output → $BS \times 32 \times 24 \times 24$ with a kernel size of 5 and a 2x2 max-pooling
  - 2nd layer (convolutional): input → $BS \times 32 \times 12 \times 12$, output → $BS \times 128 \times 5 \times 5$ with a kernel size of 3 and a 2x2 max-pooling
  - 3rd layer (linear): input → $BS \times 128 \times 5 \times 5$, output → $BS \times 256$
  - 4th layer (linear): input → $BS \times 256$, output → $BS \times 64$
  - 5th layer (linear): input → $BS \times 64$, output → $BS \times 10$

The training process is going to be performed using the Cross Entropy loss as our loss function, and for every model we want to test both the Stochastic Gradient Descent and Adam optimizers. Every model-optimizer combo (three models × two optimizers = six models in total) is therefore trained for 10 epochs on a range of different starting learning rates to see how this hyperparameter affects performance. Here is a summary of our parameters:

- `Learning rates: [0.00001, 0.0001, 0.001, 0.01, 0.1, 0.5, 1, 3, 5]`
- `Models: FullyConnected_1(), Convolutional_1(), Convolutional_3()`
- `Optimizers: SGD, Adam`
- `Number of epochs per model-optimizer combo: 10`
- `Loss function: Cross Entropy loss`

3 Training results

3.1 Overview of the models' performance

Since the total number of training instances is of considerable size ($3 \times 2 \times 9 \times 10$), the exhaustive result table is provided at this github link: https://pytorch.org/vision/stable/generated/torchvision.datasets.KMNIST.html#torchvision.datasets.KMNIST (METTI LINK GIUSTO).

Note that the results reported in the table may vary slightly vary between different runs of the Jupyter notebook, since guaranteeing complete reproducibility from these computations is difficult due to the non-deterministic nature of the algorithms involved. Here is a summary of the table, reviewing the most notable aspects of it.

- Learning rates ≥ 3 always produce bad results
- All models w/ Adam optimizer and learning rate ≥ 0.01 get stuck at around an accuracy value of 10. In general, a considerable amount of models seem to encounter some sort of plateau at approximately this accuracy.
- Some models exhibit a reduction in accuracy with higher learning rates; for example, Convolutional_3 w/ SGD and learning rate = 1 starts at 74.054496% and ends up at 10.010339%. A possible explanation is that, because of the higher learning rate, we are "bouncing" out of the minimum basin of attraction.
- Models trained w/ learning rate = 0.00001 reach a maximum of ∼87% accuracy compared to the higher values reached by models trained with learning rate = 0.0001 (or above, until performance starts to degrade due to the learning rate being too high). This behaviour is due to the step size being too small: we are "moving" towards the minimum at a slow pace.

Relevant cases in terms of training accuracy (%, after 10 epochs):

- FullyConnected_1 w/ SGD optimizer and learning rate = 1: AAAAAAAA
- FullyConnected_1 w/ Adam optimizer and learning rate = 0.0001: AAAAAAAA
- Convolutional_1 w/ SGD optimizer and learning rate = 0.5: AAAAAAAA
- Convolutional_1 w/ Adam optimizer and learning rate = 0.001: AAAAAAAA
- Convolutional_3 w/ Adam optimizer and learning rate = 0.001: AAAAAAAA

4 Analysis of a specific model

Out of the five previously listed models, we choose the third one to perform further analysis. To help understand what can go wrong, we are also going to plot training results from a poor performing model, for example Convolutional_3 w/ SGD and learning rate = 1.

4.1 Training

Plotting training accuracy and training loss yields:

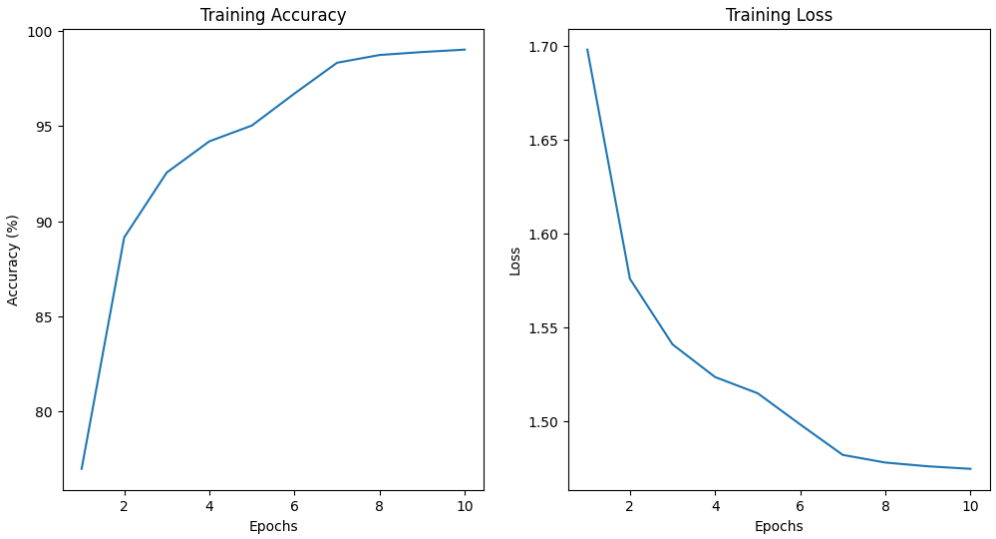

Figure 2: *Training results for model Convolutional_1 w/ SGD optimizer and learning rate = 0.5*

Plotting results from the poor performing model yields:

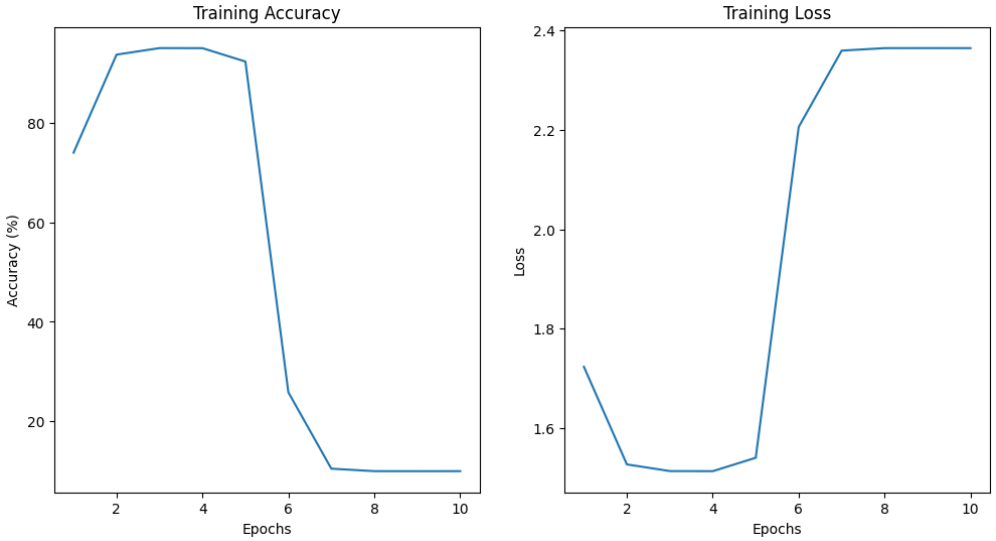

Figure 3: *Training results for model Convolutional_3 w/ SGD optimizer and learning rate = 1*

Here are the accuracy and loss values for both models respectively.

- Accuracies:
  - `[76.9777, 89.1409, 92.5527, 94.1919, 95.0240, 96.7049, 98.3224, 98.7310, 98.8861, 99.01281]`
  - `[73.9611, 93.6449, 94.9556, 94.9439, 92.2608, 25.8387, 10.5139, 10.01367, 10.0086, 10.01367]`
- Losses:
  - `[1.6979, 1.5758, 1.5408, 1.5234, 1.5148, 1.4982, 1.4820, 1.4779, 1.4759, 1.4745]`
  - `[1.7238, 1.5277, 1.5143, 1.5141, 1.5410, 2.2053, 2.3586, 2.3636, 2.3637, 2.3636]`

As we can see from the plots, the second model suffers from an initial increase in loss value right after the fourth epoch. This is probably due to the high starting learning rate of 1. Furthermore, as epochs increase, the model encounters a plateau from which it seemingly cannot escape. On the contrary, the first model rapidly and steadily reaches solid performance, stabilizing at around $\sim$99% accuracy and $\sim$1.47 loss value.

4.2 Testing

A good model should exhibit an appropriate tradeoff between bias and variance. If bias is too high, training error will reflect this by also being high, and, consequentially, testing error too. If variance is too high, training error will be much lower than testing error, producing a model that is specifically built to work well with that training set.

We would like the training error of our model to be slightly lower than the testing error, since this is a useful indicator of a good bias-variance tradeoff. Plotting testing alongside training accuracies and losses yields:

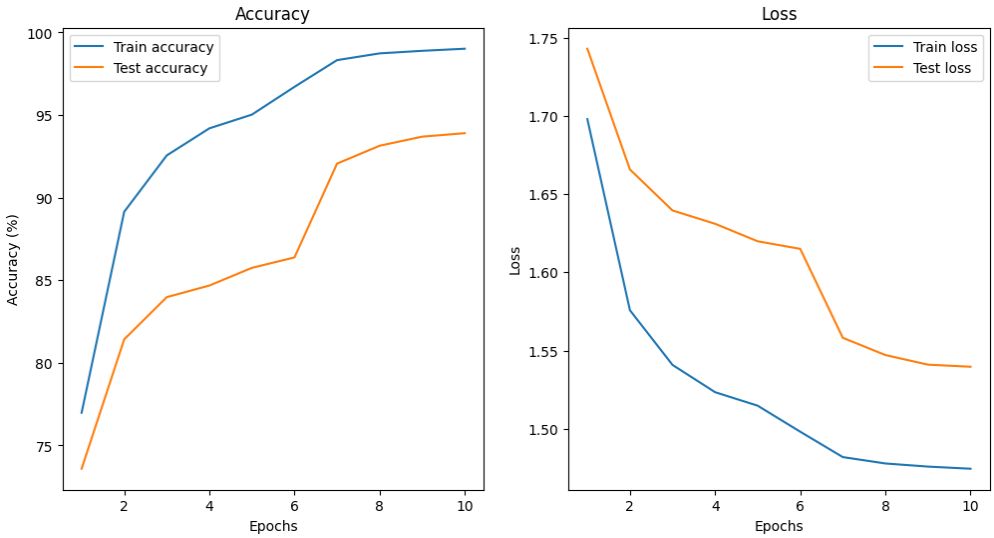

Figure 4: *Testing results for model Convolutional_1 w/ SGD optimizer and learning rate = 0.5*

Here are the accuracy and loss values for the testing set.

- Accuracies: `[73.5877, 81.4203, 83.9744, 84.6755, 85.7472, 86.3782, 92.0573, 93.1390, 93.6899, 93.9002]`
- Losses: `[1.7429, 1.6657, 1.6395, 1.6310, 1.6198, 1.6150, 1.5582, 1.5472, 1.5410, 1.5397]`

As we hoped, testing error is a little higher than training error. With an accuracy value of