CAP 6618 Spring 2019
Christian Garbin
Module 4 Assignment - Digit classification using neural networks

Objective of this assignment

The objective of this assignment is to "*Design and implement a complete solution to the problem of handwritten digit recognition, stated as follows: “given an image of a handwritten digit, produce the corresponding numerical value.” Essentially, you will build, fine-tune, and evaluate a 10-class (multi-layer perceptron) neural network-based classifier which will employ the pixel values of the images as features."*

The digits to be recognized are the ones from the MNIST dataset [1].

Files submitted

Two Jupyter notebooks are submitted as part of this assignment:

1. CAP6618_PP2_cgarbin_experiments.ipynb: based on the provided CAP6618_PP2.ipynb notebook, contains the experiments to choose a classifier.
2. CAP6618_PP2_cgarbin_final.ipynb: contains the analysis and optimization of the chosen classifier.

Choosing an MLP classifier

Two implementations of classifiers were tried:

1) The classifier from [2], chapter 2. This is a functional, but illustrative SGD classifier (not tuned for performance). It will be referred to as "Nielsen classifier" in this document.
2) The `MLPClassifier` from sklearn [3]. It will be referred to as "MLPClassifier" in this document.

The table below lists the experiments executed to optimize the classifiers and choose one of them as the final one. The experiments are documented in more details in CAP6618_PP2_cgarbin_experiments.ipynb. Text in blue highlights differences from the previous row

Classifier		Hidden layer neurons[1]	Training parameters	Training time (minutes)	Accuracy (%)
1	Nielsen	30	Epochs = 30 Mini-batch = 10 Learning rate = 3.0	3.0	94.8
2	Nielsen	**100**	Epochs = 30 Mini-batch = 10 Learning rate = 3.0	5.5	86.9
3	Nielsen	100	Epochs = 30 Mini-batch = 10 **Learning rate = 10.0**	5.5	88.9
4	Nielsen	100	Epochs = 30 Mini-batch = 10 **Learning rate = 20.0**	5.5	22.0
5	Nielsen	100	**Epochs = 10** Mini-batch = 10 **Learning rate = 15.0**	2.0	43.4
6	Nielsen	100	**Epochs = 30** Mini-batch = 10 **Learning rate = 9.0**	5.5	71.9
7	Nielsen	100	Epochs = 30 Mini-batch = 10 **Learning rate = 8.0**	5.5	96.0
8	Nielsen	**0**	Epochs = 30 Mini-batch = 10 **Learning rate = 3.0**	1.0	74.7
9	Nielsen	0	Epochs = 30 Mini-batch = 10 **Learning rate = 10.0**	1.0	79.2
10	**MLP**	30	max_iter = 30 batch_size = 10 SGD, sigmoid	1.7	88.2
11	MLP	**100**	max_iter = 30 batch_size = 10 SGD, sigmoid	4.1	90.9
12	MLP	100	max_iter = 30 batch_size = 10 SGD, sigmoid **Scaled data[2]**	2.3	88.6
13	MLP	100	max_iter = 30 batch_size = 10 SGD, sigmoid **Unscaled data** **Zero momentum[3]**	2.5	95.4
14	MLP	100	max_iter = 30 batch_size = 100 **Adam, ReLU**	0.6	96.4

[1] Input and output layers are not listed because they are the same for all classifiers.
[2] To compare with the tests using the Nielsen classifier, which uses scaled data.
[3] Also to compare more directly with the Nielsen classifier, which does not use momentum.

Some observations from the experiments:

1. By fine-tuning the learning rate, a Nielsen classifier, even with the simple implementation of [2] chapter 2, has an accuracy close to the sklearn MLPClassifier.
2. Even a low number of neurons in the hidden layer (30) results in good accuracy. This is likely related to the fact that the MNIST dataset is a relatively small dataset in its 784-dimensional space and thus easily separable in that space [4]. Reducing it to a 30-dimensional space (one hidden layer with 30 neurons) is still discriminating enough to achieve good results.
3. A network without hidden layers, however, never gets past a low accuracy. Without a hidden layer, the network is only able to represent linearly separable datasets [5]. The fact that even without a hidden layer the network still achieves about 80% accuracy is another indication of how spread out the MNIST dataset already is in its space. To capture the 20% of misclassified digits we need more than linear separation, i.e. we need to add at least one more layer to the network.

While trying variations of the Nielsen classifier, I noticed that the results vary significantly from one run to the next (same network configuration and same hyperparameters). For example, these are the results of running the classifier for five epochs, with a mini-batch size of 10 and learning rate 3.0.

Run #1	**Run #2**	**Run #3**
Epoch 0 : 7406 / 10000	Epoch 0 : 6337 / 10000	Epoch 0 : 6590 / 10000
Epoch 1 : 7578 / 10000	Epoch 1 : 9302 / 10000	Epoch 1 : 7551 / 10000
Epoch 2 : 7627 / 10000	Epoch 2 : 9378 / 10000	Epoch 2 : 8435 / 10000
Epoch 3 : 7682 / 10000	Epoch 3 : 9471 / 10000	Epoch 3 : 8555 / 10000
Epoch 4 : 7795 / 10000	Epoch 4 : 9459 / 10000	Epoch 4 : 8600 / 10000

Because of this variability, analyzing the results of the Nielsen classifier was hit-and-miss. Sometimes the variation of the classifier would perform better in one run, then worse in another. To be fair to the Nielsen classifier, it was meant to be a didactic instrument, not a production-level classifier.

Nevertheless, this variability was one of the reasons that for the final notebook I chose the MLPClassifier. The other reason was the desire to get more familiar with sklearn in general.

In particular, I chose the MLP classifier #14 from the table above. This classifier uses hyperparameters that are closer to what I have seen used in examples and papers. Among those hyperparameters: a larger batch size, an adaptive optimizer (Adam), and ReLU for activation units. It trains about four times as fast as the comparable SGD classifier. Shorter training times help with experiments.

The notebook CAP6618_PP2_cgarbin_final.ipynb continues the fine-tuning of that classifier.

Sources

[1] LeCun et al., The MNIST Database, http://yann.lecun.com/exdb/mnist/, accessed on February 7th, 2019.

[2] Michael Nielsen, Neural Networks and Deep Learning, http://neuralnetworksanddeeplearning.com/, accessed on March 1st, 2019.

[3] https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html, accessed on March 1st, 2019.

[4] C. Olah, Visualizing MNIST: An Exploration of Dimensionality Reduction, http://colah.github.io/posts/2014-10-Visualizing-MNIST, accessed on March 1st, 2019.

[5] J. Heaton, The Number of Hidden Layers, https://www.heatonresearch.com/2017/06/01/hidden-layers.html, accessed March 3rd, 2019.