Machine Learning (MLG)
HEIG-VD
2021

Using Keras to train Multi-Layer Perceptrons and Convolutional Neural Networks

Keras is a machine learning library that implements artificial neural networks. It has the advantage of allowing very compact and fast implementations of complex network architectures.

Keras is a high level framework that uses TensorFlow as default backend (other backends are possible e.g., Theano). Whatever the choice, Keras allows to (transparently) compile python scripts into CPU or GPU code.

Keras can be easily installed through the installation of tensorflow by doing:
$ pip install tensorflow
Keras documentation can be found at https://keras.io/.

We suggest you to use Google colab services. You can find a brief introduction here :
https://towardsdatascience.com/getting-started-with-google-colab-f2fff97f594c

Objective: Train a feed-forward neural network to perform digit classification.

- **Prepare your data**. Normalize the features you computed, transform the output data into binary vectors, split the data to create training a testing datasets.
- **Create a model**. Use the sequential (https://keras.io/api/models/sequential/) or the functional (https://keras.io/guides/functional_api/) API to create a model
- **Compile the model**. Define the optimizer and the loss function.
- **Train the model**. Run the optimizer for a number of epochs and collect statistics about the learning process
- **Run the model on new data**. Finally you can show a classification report and a confusion matrix to evaluate your model.

The following two IPython notebooks illustrate the use of MLPs and CNNs on the task of digit recognition, using the MNIST database, widely used as a benchmark for supervised learning.

1. MLP_from_raw_data.ipynb
2. CNN.ipynb

In these examples, SGD means Stochastic Gradient Descent which is Backpropagation in other words. Notice that here we do not use the MSE error as the loss function but the so-called “categorical crossentropy”. Do not hesitate to read more about it on Wikipedia or another source. Notice that in this architecture we use the “ReLu” activation function for the hidden layers and “softmax” for the output layer. Observe all the hyperparameters we have set: batch size, convolution sizes, number of convolutions, the use of pooling layers, the flatten operator to link the convolution part and the multi-layer perceptron part (e.g., dense or fully connected layer) of a CNN.

Questions

1. Observe the network architectures in both cases (MLP and CNN). Which architecture has more parameters? Which architecture shows better performance? Why?
2. You have tested both APIs: sequential and functional. What do you prefer? Why?
3. Have you tested your code using both CPU and GPU? Which one is faster?

Enjoy it.