Machine Learning (MLG)
HEIG-VD
2020

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 one of two backends: TensorFlow or Theano. Whatever the choice, Keras allows to (transparently) compile python scripts into CPU or GPU code.

Keras can be easily installed by doing: **$ conda install keras**
However, this basic installation does not provide GPU support for the Tensorflow backend.
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. You will most likely need :

```
from sklearn import preprocessing as pp
from sklearn import model_selection as ms
```

To **create your model**, you will most likely need the following keras functions :

```
from keras.layers import Input, Dense
from keras.models import Model
from keras.optimizers import SGD
from keras.utils import np_utils
```

You have to build a network, then **compile the model** :

```
inputs = Input(shape=(N_FEATURES,))

hidden = Dense(N_NEURONS, activation='relu')(inputs)
outputs = Dense(N_OUTPUTS, activation='softmax')(hidden)
model = Model(inputs=inputs, outputs=outputs)
gradient_descent = SGD(lr=0.01, momentum=0.0, decay=0.0)
model.compile(optimizer=gradient_descent,
        loss='categorical_crossentropy',
        metrics=['accuracy'])
```

Then you have **to train the model** using the data you previously selected for training :

hist = model.fit(X_train, Y_train, validation_data=(X_test, Y_test), epochs=10, batch_size=32)

To **evaluate the performance** of the model :

loss_and_metrics = model.evaluate(X_test, Y_test, batch_size=32)

And to **run the model on new data** :

probabilities = model.predict(X_test, batch_size=32)

At the end of training, show a plot with the test and train error vs. epochs.

```
pl.plot(hist.history['loss'])
pl.plot(hist.history['val_loss'])
```

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.

There are many more examples on the Internet, for example:

http://machinelearningmastery.com/tutorial-first-neural-network-python-keras/

Enjoy it.