Intro to Keras

Alvaro Soto

Computer Science Department (DCC), PUC

Deep Learning Software Tools

There is a large list of highly capable tools to implement deep learning models. Among the most popular tools, we can find:

- TensorFlow, `http://www.tensorflow.org/`, supported by Google.
- Keras, `http://keras.io/`, supported by Francois Chollet and Keras Google group, on top of TensorFlow.
- PyTorch, `http://pytorch.org/`, supported by Facebook AI lab.
- MxNet, `https://mxnet.apache.org/`, supported by several companies and universities.
- Sonnet, `https://sonnet.readthedocs.io`, supported by Deep Mind, on top of TensorFlow.
- Among others.

In this class, we will use Keras running on top of TensorFlow.

- Keras is a high-level neural networks library. It can run on top of TensorFlow or Theano.
- Written in Python.
- Easy to use, provide great modularity.
- Developed with a focus on enabling fast experimentation.
- Support convolutional and recurrent neural network models (CNNs and RNNs).
- Runs seamlessly on CPU and GPU.

Keras Modularity

Several modules available:

- Neural layers,
- Cost functions,
- Optimizers,
- Initialization schemes,
- Activation functions,
- Regularization schemes
- New modules can be easily integrated (extensibility).

Keras allows us to combine all these modules to easily implement powerful deep learning models.

Keras Models

- The core data structure of Keras is a model.
- A model provides a way to organize the modules that compose a deep learning solution. It is like a “container”.
- There are two type of models:
  - Sequential model: a linear pile of layers, like a pipeline.
  - Graph model: provide arbitrary layer connections with an arbitrary number of inputs and outputs.
- Models are described using Python style function calls.

Keras Python Style Calls

Create a sequential model:

```
from keras.models import Sequential
MyModel = Sequential()
```

Create a graph model:

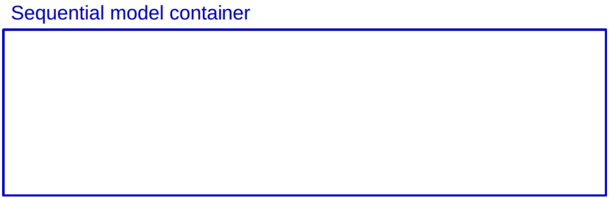
```
from keras.models import Graph
MyModel = Graph()
```

Example Sequential Model

Create a sequential model:

```
from keras.models import Sequential
MyFirstModel = Sequential()
```

This creates an empty container named *MyFirstModel*:



Example Sequential Model

Add a module that implements a dense connected layer with 100 inputs and 100 outputs.

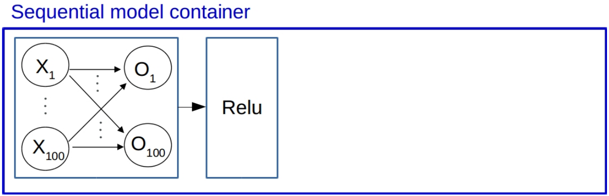
```
from keras.models import Sequential
model = Sequential()
model.add(Dense(output_dim=100, input_dim=100,
init="glorot_uniform"))
```

Sequential model container

Example Sequential Model

Add Relu activation functions.

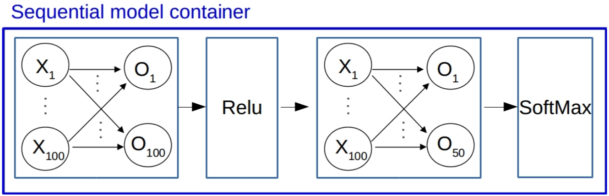
```
from keras.models import Sequential
model = Sequential()
model.add(Dense(output_dim=100, input_dim=100,
init="glorot_uniform"))
model.add(Activation("relu"))
```



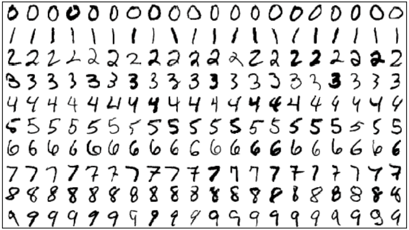
Example Sequential Model

Add a second module that implements a dense connected layer with 50 outputs plus a soft-max activation output.

```
from keras.models import Sequential
model = Sequential()
model.add(Dense(output_dim=100, input_dim=100,
init="glorot_uniform"))
model.add(Activation("relu"))
model.add(Dense(output_dim=50, init="glorot_uniform"))
model.add(Activation("softmax"))
```



Case example: Recall from first classes MNIST dataset



- MNIST is a dataset of images corresponding to handwritten digits.
- Each image displays a single digit from 0 to 9. Images are binary with a resolution of 28x28 pixels.
- Goal: Build a classifier that can recognize the digit in each image.
- Dataset consists of 60.000 training examples and 10.000 test cases.

A Classifier for MNIST Dataset Using Keras

Let's first define the house keeping code to load and pre-process input data.

```
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K

batch_size, num_classes, epochs = 128, 10, 2

# input image dimensions
img_rows, img_cols = 28, 28

# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
input_shape = (img_rows, img_cols, 1)

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')

# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

Now let's define the deep learning model. You do need to understand this code (model definition):

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),activation='relu', input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
model.summary()
```

Can you sketch the underlying NW?

Finally, let's run the model. You should know the basic commands:

```
#define loss function, optimizer and metric to access final model performance.
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])

#Train model
model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(x_test, y_test))

#Evaluate resulting model on an independent test set
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

- Previous code is part of the examples included in Keras.
- It achieves a test accuracy close to 100% after 20 epochs.