Keras 1 - Building a Neural Network in Keras

March 31, 2018

0.1 Building a Neural Network in Keras

0.1.1 Sequential Model

```
from keras.models import Sequential

# Create the Sequential model
model = Sequential()
```

The `keras.models.Sequential` class is a wrapper for the neural network model that treats the network as a sequence of layers. It implements the Keras model interface with common methods like `compile()`, `fit()`, and `evaluate()` that are used to train and run the model.

0.1.2 Layers

The Keras Layer class provides a common interface for a variety of standard neural network layers. There are fully connected layers, max pool layers, activation layers, etc. We can add a layer to a model using the model's `add()` method.

For example, a simple model with a single hidden layer might look like this:

```
In [2]: import numpy as np
        from keras.models import Sequential
        from keras.layers.core import Dense, Activation

        # X shape: (num_rows, num_cols), where the training data are stored as row vectors
        X = np.array([[0,0],[0,1],[1,0],[1,1]], dtype = np.float32) # shape: (4,2)

        # y must have an output vector for each input vector
        y = np.array([[0], [0], [0], [1]], dtype = np.float32) # shape: (4,1)

        # Create the Sequential model
        model = Sequential()

        # 1st Layer - Add an input layer of 32 nodes with the same input shape
        #  as the training samples in X i.e. X.shape[1] = 2
        model.add(Dense(32, input_dim = X.shape[1]))

        # Add a softmax activation layer
```

```
model.add(Activation('softmax'))

# 2nd Layer - Add a fully connected output layer, with 1 means the number of output
model.add(Dense(1))

# Add a sigmoid activation layer
model.add(Activation('sigmoid'))
```

Keras requires the input shape to be specified in the first layer, while the shape of all other layers will be automatically inferred. **We need to explicitly set the input dimensions for the first layer.**

The first hidden layer, `model.add(Dense(32, input_dim = X.shape[1]))` creates 32 nodes which each expect 2-element vectors as inputs (i.e. `X.shape[1]` which is 2). Each layer takes the outputs from the previous layer as inputs and pipes to the next layer. This chain of passing output to the next layer continues until the last layer, which is the output of the model. From the code above, the output has dimension 1, `model.add(Dense(1))`.

The activation layer is defined as `softmax`, `model.Add(Activation('softmax'))`.

Note that the two lines:

```
model.add(Dense(32, input_dim = X.shape[1]))
model.add(Activation('softmax'))
```

can be combined into `model.add(Dense(32, activation='softmax'))`, but it is common to explicitly separate the activation layers because it allows direct access to the outputs of each layer before the activation is applied.

Once we have our model built, **we need to compile it before it can be run**. Compiling the Keras model calls the backend (tensorflow, theano, etc.) and binds the optimizer, loss function and other parameters required before the model can be run on any input data.

We'll specify the loss function to be `categorical_crossentropy` which can be used when there are only two classes, and specify `adam` as the optimiser.

Finally, we specify what metrics we want to evaluate the model with. Here we'll use `accuracy`.

```
model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
```

We can see the resulting model architecture with the following command:

```
model.summary()
```

The model is trained with the `fit()` method, in which we have to specify the number of training epochs and the message level (how much info we want to display on the screen during training).

```
model.fit(X, y, nb_epoch=1000, verbose=0)
```

Finally, we can use the following command to evaluate the model:

```
model.evaluate()
```