Intro to Keras and Google Colab

Joseph Rejive

November 2018

1 Google Colab

Google Colaboratory is a free, cloud based machine learning platform. It offers a Jupyter Notebook along with a Python environment with sklearn, Tensorflow, Keras, and other libraries meant for machine learning. It even gives users access to a Tesla K80, making it a great resource for those who don't have access to a GPU at home.

1.1 Getting Started

To get started, go to https://colab.research.google.com/ and create a new notebook. All colaboratory notebooks will be saved to Drive. Now, you have acess to an online Jupyter notebook. To create a new cell below the existing cell, the command is *Ctrl + M + B*. To Create a new cell above the existing cell, the command is *Ctrl + M + A*. Finally, to delete the existing cell, the command is *Ctrl + M + D*.

1.2 Using the GPU

To run your models off the K80, go to *Edit* > *Notebook Settings* and select GPU as the hardware accelerator. It's as simple as that.

1.3 Loading Datasets

There are two ways to upload datasets to Colab. For the first method, click the arrow on the left side of the screen and navigate to the *Files* tab. Click on the *Upload Files* button and select the file you want to upload. However, this is only a temporary solution. In Google Colab, the code you write is executed on a Virtual Machine. After a period of inactivity, the VM is recycled and your uploaded files disappear.

The other method is to upload your files to Google Drive and allow Colab to access Drive. To do so, run the following code in a Jupyter cell:

```
from google.colab import drive
drive.mount('/content/gdrive')
```

Click on the link that appears and sign into your Google Drive. Once you do so, an authorization code will appear. Copy it and paste it in the box in your Jupyter environment. Now you'll be able to access any file in Drive.

2 Intro to Keras

Keras is a high-level Python machine learning API, which allows you to easily run neural networks. Keras is simply a specification; it provides a set of methods that you can use, and it will use a backend (TensorFlow, Theano, or CNTK, as chosen by the user) to actually run your code. Like many machine learning frameworks, Keras is a so-called *define-and-run* framework. This means that it will define and optimize your neural network in a compilation step before training starts.

3 First Steps

First, install Keras: `pip install keras`

We'll go over a fully-connected neural network designed for the MNIST (classifying handwritten digits) dataset.

```
# Can be found at https://github.com/fchollet/keras
# Released under the MIT License

import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.optimizers import RMSprop
```

First, we handle the imports. `keras.datasets` includes many popular machine learning datasets, including CIFAR and IMDB. `keras.layers` includes a variety of neural network layer types, including `Dense`, `Conv2D`, and `LSTM`. `Dense` simply refers to a fully-connected layer, and `Dropout` is a layer commonly used to address the problem of overfitting. `keras.optimizers` includes most widely used optimizers. Here, we opt to use RMSprop, an alternative to the classic stochastic gradient decent (SGD) algorithm. Regarding `keras.models`, `Sequential` is most widely used for simple networks; there is also a functional `Model` class you can use for more complex networks.

```
batch_size = 128
num_classes = 10
epochs = 20
```

Here, we define our constants...

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

...and the dataset.

```
x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
```

The training set will contain 60000 28 by 28 images; we resize this to 60000 by 784 since our MLP will take as input a vector of length 784. Similarly, the test set, which consists of 10000 images is resized to 10000 by 784. We next convert these tensors to floats, and normalize the values to $[0, 1]$.

```
# 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)
```

This code converts the ground-truth labels from a class vector (each digit is labeled with an integer 0 through 9) to a one-hot encoding (each digit is labeled with a length 9 vector which consists of zeros except for the index represented by the digit, which equals one). This is so that the ground-truth labels will be compatible with our categorical cross-entropy loss function.

```
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(10, activation='softmax'))
```

Here, the neural network is defined. The inputs are $784 \to 512 \to 512 \to 10$, with the ReLU activation function (alternative to the classical sigmoid). A softmax is applied at the end to convert our ten outputs to probabilities, each output denoting a digit. Dropout is applied in between layers (see hyperparameters).

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

Here, the model is compiled. Our loss function is categorical cross-entropy, which is used for categorical data with more than two classes. The `metrics` keyword argument is a list of metrics we want to keep track of in throughout training, validation, and testing. We can provide our own functions, or provide strings denoting built-in metrics (e.g. accuracy).

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

Here, the model is trained. The `verbose` keyword argument gives us a nice progress bar as the training process progresses. Returned is a history of the metrics throughout training.

```
score = model.evaluate(x_test, y_test, verbose=0)
```

Finally, the model is evaluated on the test set. The metrics, as defined in the compile function, are returned.

4 More Stuff

Though this was a fairly simple example, it covers mostly what you will use in Keras. More advanced layers, optimizers, or losses can be found at Keras's homepage [https://keras.io/]; Keras has great documentation! Things that may interest you are Keras's functional API, which allows more flexible networks with multiple inputs/outputs, as well as the backend API, which allows for defining custom layers. Note that Keras and TensorFlow are interoperable; you can write TensorFlow code that operates on Keras tensors, for defining operations at an even lower level than that specified in the backend API (given that you use the TensorFlow backend for Keras).

5 Conclusion

Keras is a great framework for rapid prototyping; it provides a high-level API with potential for lower level operations, when necessary. Note that, however, when working with large or customized models over long-term projects, you may want to look into other frameworks such as PyTorch, which doesn't have a compile step and is much more low-level.

With regards to hyperparameters: Keras's defaults are good starting points for hyperparameters. Use grid search to tune them. In general, try Adam before other optimizers.