Model Execution

Learn how to train, evaluate, and make predictions with a Keras model.

Chapter Goals:

- Understand the facets of model execution for Keras models

A. Training

After configuring a Keras model for training, it only takes a single line of code to actually perform the training. We use the `Sequential` model's `fit` function to train the model on input data and labels.

The first two arguments of the `fit` function are the input data and labels, respectively. Unlike TensorFlow (where we need to use tensor objects for any sort of data), we can simply use NumPy arrays as the input arguments for the `fit` function.

The training batch size can be specified using the `batch_size` keyword argument (the default is a batch size of 32). We can also specify the number of epochs, i.e. number of full run-throughs of the dataset during training, using the `epochs` keyword argument (the default is 1 epoch).

```
model = Sequential()
layer1 = Dense(200, activation='relu', input_dim=4)
model.add(layer1)
layer2 = Dense(200, activation='relu')
model.add(layer2)
layer3 = Dense(3, activation='softmax')
model.add(layer3)
model.compile('adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# predefined multiclass dataset
train_output = model.fit(data, labels,
                         batch_size=20, epochs=5)
```

The console output of the training is logged for each epoch. Notice that both the loss and classification accuracy per epoch is measured (since we configured the model to track classification accuracy).

The output of the `fit` function is a `History` object, which records the training metrics. The object's `history` attribute is a dictionary that contains the metric values at each epoch of training.

```
print(train_output.history)
```

The history for the train_output object from the previous code snippet.

B. Evaluation

Evaluating a trained Keras model is just as simple as training it. We use the `Sequential` model's `evaluate` function, which also takes in data and labels (NumPy arrays) as its first two arguments.

Calling the `evaluate` function will evaluate the model over the entire input dataset and labels. The function returns a list containing the evaluation loss as well as the values for each metric specified during model configuration.

```
# predefined eval dataset
print(model.evaluate(eval_data, eval_labels))
```

Evaluating the previously trained model.

C. Predictions

Finally, we can make predictions with a Keras model using the `predict` function. The function takes in a NumPy array dataset as its required argument, which represents the data observations that the model will make predictions for.

The output of the `predict` function is the output of the model. That means for classification, the `predict` function is the model's class probabilities for each data observation.

```
# 3 new data observations
print('{}'.format(repr(model.predict(new_data))))
```

Class probabilities for multiclass new data observations.

In the code above, the model returned class probabilities for the 3 new data observations. Based on the probabilities, the first observation would be classified as class **0**, the second observation would be classified as class **2**, and the third observation would be classified as class **1**.