Additional callbacks

November 14, 2020

1 Additional callbacks

In this reading we'll be looking at more of the inbuilt callbacks available in Keras.

```
In [1]: import tensorflow as tf
        print(tf.__version__)
```

```
2.0.0
```

We will again be using the sklearn diabetes dataset to demonstrate these callbacks.

```
In [2]: # Load the diabetes dataset

        from sklearn.datasets import load_diabetes

        diabetes_dataset = load_diabetes()
```

```
In [3]: # Save the input and target variables

        from sklearn.model_selection import train_test_split

        data = diabetes_dataset['data']
        targets = diabetes_dataset['target']
```

```
In [4]: # Split the data set into training and test sets

        train_data, test_data, train_targets, test_targets = train_test_split(data, targets, te
```

Let's also build a simple model to fit to the data with our callbacks.

```
In [5]: # Build the model

        from tensorflow.keras.models import Sequential
        from tensorflow.keras.layers import Dense

        model = tf.keras.Sequential([
            Dense(128, activation='relu', input_shape=(train_data.shape[1],)),
```

```
            Dense(64,activation='relu'),
            Dense(64, activation='relu'),
            Dense(64, activation='relu'),
            Dense(1)
        ])
```

```
In [6]: # Compile the model

        model.compile(loss='mse',
                      optimizer="adam",metrics=["mse","mae"])
```

Now onto the callbacks!

1.1 Learning rate scheduler

Usage: `tf.keras.callbacks.LearningRateScheduler(schedule, verbose=0)`

The learning rate scheduler that we implemented in the previous reading as a custom callback is also available as a built in callback.

As in our custom callback, the `LearningRateScheduler` in Keras takes a function `schedule` as an argument.

This function `schedule` should take two arguments: * The current epoch (as an integer), and * The current learning rate,

and return new learning rate for that epoch.

The `LearningRateScheduler` also has an optional `verbose` argument, which prints information about the learning rate if it is set to 1.

Let's see a simple example.

```
In [7]: # Define the learning rate schedule function

        def lr_function(epoch, lr):
            if epoch % 2 == 0:
                return lr
            else:
                return lr + epoch/1000
```

```
In [8]: # Train the model

        history = model.fit(train_data, train_targets, epochs=10,
                            callbacks=[tf.keras.callbacks.LearningRateScheduler(lr_function, v
```

```
Epoch 00001: LearningRateScheduler reducing learning rate to 0.0010000000474974513.

Epoch 00002: LearningRateScheduler reducing learning rate to 0.0020000000474974513.

Epoch 00003: LearningRateScheduler reducing learning rate to 0.0020000000949949026.

Epoch 00004: LearningRateScheduler reducing learning rate to 0.005000000094994903.
```

```
Epoch 00005: LearningRateScheduler reducing learning rate to 0.004999999888241291.

Epoch 00006: LearningRateScheduler reducing learning rate to 0.009999999888241292.

Epoch 00007: LearningRateScheduler reducing learning rate to 0.009999999776482582.

Epoch 00008: LearningRateScheduler reducing learning rate to 0.01699999977648258.

Epoch 00009: LearningRateScheduler reducing learning rate to 0.016999999061226845.

Epoch 00010: LearningRateScheduler reducing learning rate to 0.025999999061226846.
```

You can also use lambda functions to define your `schedule` given an epoch.

```
In [9]: # Train the model with a difference schedule

        history = model.fit(train_data, train_targets, epochs=10,
                            callbacks=[tf.keras.callbacks.LearningRateScheduler(lambda x:1/(3+
                            verbose=False)
```

```
Epoch 00001: LearningRateScheduler reducing learning rate to 0.3333333333333333.

Epoch 00002: LearningRateScheduler reducing learning rate to 0.125.

Epoch 00003: LearningRateScheduler reducing learning rate to 0.07692307692307693.

Epoch 00004: LearningRateScheduler reducing learning rate to 0.05555555555555555.

Epoch 00005: LearningRateScheduler reducing learning rate to 0.043478260869565216.

Epoch 00006: LearningRateScheduler reducing learning rate to 0.03571428571428571.

Epoch 00007: LearningRateScheduler reducing learning rate to 0.030303030303030304.

Epoch 00008: LearningRateScheduler reducing learning rate to 0.02631578947368421.

Epoch 00009: LearningRateScheduler reducing learning rate to 0.023255813953488372.

Epoch 00010: LearningRateScheduler reducing learning rate to 0.020833333333333332.
```

1.2 CSV logger

Usage `tf.keras.callbacks.CSVLogger(filename, separator=',', append=False)`

This callback streams the results from each epoch into a CSV file. The first line of the CSV file will be the names of pieces of information recorded on each subsequent line, beginning with the epoch and loss value. The values of metrics at the end of each epoch will also be recorded.

The only compulsory argument is the `filename` for the log to be streamed to. This could also be a filepath.

You can also specify the `separator` to be used between entries on each line.

The `append` argument allows you the option to append your results to an existing file with the same name. This can be particularly useful if you are continuing training.

Let's see an example.

```
In [10]: # Train the model with a CSV logger

         history = model.fit(train_data, train_targets, epochs=10,
                             callbacks=[tf.keras.callbacks.CSVLogger("results.csv")], verbose=
```

Let's view the information in the CSV file we have created using `pandas`.

```
In [11]: # Load the CSV

         import pandas as pd

         pd.read_csv("results.csv", index_col='epoch')
```

Out[11]:

epoch	loss	mae	mse
0	5892.488634	64.908720	5892.4890
1	5889.859400	65.415220	5889.8594
2	5903.834498	65.601320	5903.8345
3	5906.434793	65.064680	5906.4346
4	5890.073566	64.976746	5890.0737
5	5889.598314	65.248825	5889.5980
6	5916.778732	65.830070	5916.7790
7	5903.802209	65.017456	5903.8022
8	5934.408449	65.208900	5934.4087
9	5892.213869	65.149790	5892.2140

1.3 Lambda callbacks

Usage `tf.keras.callbacks.LambdaCallback( on_epoch_begin=None, on_epoch_end=None, on_batch_begin=None, on_batch_end=None, on_train_begin=None, on_train_end=None)`

Lambda callbacks are used to quickly define simple custom callbacks with the use of lambda functions.

Each of the functions require some positional arguments. * `on_epoch_begin` and `on_epoch_end` expect two arguments: `epoch` and `logs`, * `on_batch_begin` and `on_batch_end` expect two arguments: `batch` and `logs` and * `on_train_begin` and `on_train_end` expect one argument: `logs`.

Let's see an example of this in practice.

```
In [12]: # Print the epoch number at the beginning of each epoch

         epoch_callback = tf.keras.callbacks.LambdaCallback(
             on_epoch_begin=lambda epoch,logs: print('Starting Epoch {}!'.format(epoch+1)))
```

```
In [13]: # Print the loss at the end of each batch

         batch_loss_callback = tf.keras.callbacks.LambdaCallback(
             on_batch_end=lambda batch,logs: print('\n After batch {}, the loss is {:7.2f}.'.f

In [14]: # Inform that training is finished

         train_finish_callback = tf.keras.callbacks.LambdaCallback(
             on_train_end=lambda logs: print('Training finished!'))

In [15]: # Train the model with the lambda callbacks

         history = model.fit(train_data, train_targets, epochs=5, batch_size=100,
                             callbacks=[epoch_callback, batch_loss_callback,train_finish_callba

Starting Epoch 1!

 After batch 0, the loss is 6991.75.

 After batch 1, the loss is 5035.93.

 After batch 2, the loss is 5827.96.

 After batch 3, the loss is 5704.10.
Starting Epoch 2!

 After batch 0, the loss is 5451.67.

 After batch 1, the loss is 5780.36.

 After batch 2, the loss is 6672.73.

 After batch 3, the loss is 5636.80.
Starting Epoch 3!

 After batch 0, the loss is 5729.87.

 After batch 1, the loss is 6746.16.

 After batch 2, the loss is 5794.30.

 After batch 3, the loss is 5260.57.
Starting Epoch 4!

 After batch 0, the loss is 5842.15.

 After batch 1, the loss is 6065.46.
```

```
 After batch 2, the loss is 6097.44.

 After batch 3, the loss is 5541.83.
Starting Epoch 5!

 After batch 0, the loss is 5964.17.

 After batch 1, the loss is 5346.65.

 After batch 2, the loss is 6388.70.

 After batch 3, the loss is 5851.86.
Training finished!
```

1.4 Reduce learning rate on plateau

Usage `tf.keras.callbacks.ReduceLROnPlateau( monitor='val_loss', factor=0.1, patience=10, verbose=0, mode='auto', min_delta=0.0001, cooldown=0, min_lr=0)`

The `ReduceLROnPlateau` callback allows reduction of the learning rate when a metric has stopped improving. The arguments are similar to those used in the `EarlyStopping` callback. * The argument `monitor` is used to specify which metric to base the callback on. * The `factor` is the factor by which the learning rate decreases i.e., new_lr=factor*old_lr*. The `patience` is the number of epochs where there is no improvement on the monitored metric before the learning rate is reduced. * The `verbose` argument will produce progress messages when set to 1. * The `mode` determines whether the learning rate will decrease when the monitored quantity stops increasing (`max`) or decreasing (`min`). The `auto` setting causes the callback to infer the mode from the monitored quantity. * The `min_delta` is the smallest change in the monitored quantity to be deemed an improvement. * The `cooldown` is the number of epochs to wait after the learning rate is changed before the callback resumes normal operation. * The `min_lr` is a lower bound on the learning rate that the callback will produce.

Let's examine a final example.

```
In [16]: # Train the model with the ReduceLROnPlateau callback

        history = model.fit(train_data, train_targets, epochs=100, batch_size=100,
                            callbacks=[tf.keras.callbacks.ReduceLROnPlateau(
                                monitor="loss",factor=0.2, verbose=1)], verbose=False)


Epoch 00024: ReduceLROnPlateau reducing learning rate to 0.00416666679084301.

Epoch 00034: ReduceLROnPlateau reducing learning rate to 0.0008333333767950535.

Epoch 00064: ReduceLROnPlateau reducing learning rate to 0.00016666667070239783.
```

```
Epoch 00074: ReduceLROnPlateau reducing learning rate to 3.333333297632635e-05.

Epoch 00084: ReduceLROnPlateau reducing learning rate to 6.666666740784422e-06.

Epoch 00094: ReduceLROnPlateau reducing learning rate to 1.3333333299669903e-06.
```

1.4.1 Further reading and resources

- https://keras.io/callbacks/
- https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/LearningRateScheduler
- https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/CSVLogger
- https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/LambdaCallback