hy-23 Created using Colaboratory

History

1 contributor

531 lines (531 sloc) | 19.7 KB

Open in Colab

In [1]:
```
pip install -q -U keras-tuner
```

```
     |████████████████████████████████| 133 kB 5.1 MB/s
```

In [2]:
```
import tensorflow as tf
from tensorflow import keras
import keras_tuner as kt
import pandas as pd
```

Download dataset

In [3]:
```
(img_train, label_train), (img_test, label_test) = keras.datasets.fashion_mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyt
e.gz
32768/29515 [=================================] - 0s 0us/step
40960/29515 [==========================================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyt
e.gz
26427392/26421880 [==============================] - 0s 0us/step
26435584/26421880 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.
gz
16384/5148 [=============================================================================================
==] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.
gz
4423680/4422102 [==============================] - 0s 0us/step
4431872/4422102 [==============================] - 0s 0us/step
```

In [4]:
```
# Normalize pixel values between 0 and 1
img_train = img_train.astype('float32') / 255.0
img_test = img_test.astype('float32') / 255.0
```

In [5]:
```
img_train.shape
```

Out[5]:
```
(60000, 28, 28)
```

In [6]:
```
def model_builder(hp):
  model = keras.Sequential()
  model.add(keras.layers.Flatten(input_shape=(28,28)))

  # Tune the number of units in the first Dense layer
  # Choose an optimal value between 32-512
  hp_units = hp.Int('units', min_value=32, max_value=512, step=32)
  model.add(keras.layers.Dense(units=hp_units, activation='relu'))
  model.add(keras.layers.Dense(10))

  # Tune the learning rate for the optimizer
  # Choose an optimal value from 0.01, 0.001, or 0.0001
  hp_learning_rate = hp.Choice('learning_rate', values=[1e-2, 1e-3, 1e-4])

  model.compile(optimizer=keras.optimizers.Adam(learning_rate=hp_learning_rate),
                loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                metrics=['accuracy'])

  return model
```

In [7]:
```
md = model_builder(kt.HyperParameters())
```

In [8]:
```
md.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 784)               0

 dense (Dense)               (None, 32)                25120
```

```
 dense_1 (Dense)             (None, 10)                330

=================================================================
Total params: 25,450
Trainable params: 25,450
Non-trainable params: 0
_________________________________________________________________
```

In [9]:

```
tuner = kt.Hyperband(model_builder,
                     objective='val_accuracy',
                     max_epochs=10,
                     factor=3,
                     directory='my_dir',
                     project_name='intro_to_kt')
```

In [10]:

```
# patience: Number of epochs with no improvement after which training will be stopped.
stop_early = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5)
```

In [11]:

```
tuner.search(img_train, label_train, epochs=50, validation_split=0.2, callbacks=[stop_early])
```

```
Trial 30 Complete [00h 02m 22s]
val_accuracy: 0.8809166550636292

Best val_accuracy So Far: 0.8948333263397217
Total elapsed time: 00h 18m 34s
INFO:tensorflow:Oracle triggered exit
```

In [12]:

```
# Get the optimal hyperparameters
best_hps=tuner.get_best_hyperparameters(num_trials=1)[0]

print(f"""
The hyperparameter search is complete. The optimal number of units in the first densely-connected
layer is {best_hps.get('units')} and the optimal learning rate for the optimizer
is {best_hps.get('learning_rate')}.
""")
```

```
The hyperparameter search is complete. The optimal number of units in the first densely-connected
layer is 416 and the optimal learning rate for the optimizer
is 0.001.
```

In [13]:

```
model = tuner.hypermodel.build(best_hps)
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_1 (Flatten)         (None, 784)               0

 dense_2 (Dense)             (None, 416)               326560

 dense_3 (Dense)             (None, 10)                4170

=================================================================
Total params: 330,730
Trainable params: 330,730
Non-trainable params: 0
_________________________________________________________________
```

In [14]:

```
def tabulate_error(history):
  hist = pd.DataFrame(history.history)
  hist['epoch'] = history.epoch
  print(hist)
```

In [15]:

```
history = model.fit(img_train, label_train, epochs=50, validation_split=0.2, verbose = 0)
```

In [16]:

```
tabulate_error(history)
```

```
       loss  accuracy  val_loss  val_accuracy  epoch
0  0.497134  0.823042  0.446416      0.844500      0
1  0.370620  0.865229  0.356844      0.870750      1
2  0.336037  0.875667  0.342489      0.875583      2
3  0.305169  0.886125  0.340078      0.874833      3
4  0.285354  0.893875  0.330016      0.880500      4
5  0.270652  0.899083  0.335104      0.879500      5
```

6	0.258049	0.902792	0.316359	0.885833	6
7	0.243404	0.908208	0.316311	0.889417	7
8	0.234065	0.913479	0.328519	0.885083	8
9	0.224974	0.916188	0.326412	0.889000	9
10	0.217011	0.919042	0.317000	0.892083	10
11	0.207926	0.921938	0.321355	0.890083	11
12	0.197358	0.925917	0.319650	0.891583	12
13	0.193123	0.928021	0.340121	0.886000	13
14	0.186601	0.928979	0.336355	0.890167	14
15	0.180285	0.931667	0.346808	0.888833	15
16	0.175500	0.933000	0.332576	0.894250	16
17	0.169453	0.936562	0.352892	0.889333	17
18	0.163824	0.938146	0.346167	0.895083	18
19	0.156442	0.940208	0.364934	0.889167	19
20	0.150596	0.943646	0.349299	0.893583	20
21	0.148667	0.944146	0.338324	0.897417	21
22	0.144271	0.946083	0.346246	0.896833	22
23	0.138773	0.948646	0.357286	0.895917	23
24	0.136331	0.947354	0.382292	0.897250	24
25	0.132167	0.949854	0.397104	0.889750	25
26	0.129606	0.950854	0.389450	0.895417	26
27	0.124550	0.952646	0.364657	0.896083	27
28	0.119445	0.955271	0.395976	0.895083	28
29	0.119673	0.954938	0.404699	0.892167	29
30	0.118417	0.955833	0.399933	0.896250	30
31	0.112675	0.957604	0.402605	0.896500	31
32	0.110499	0.958250	0.425692	0.895333	32
33	0.104771	0.959875	0.445897	0.888667	33
34	0.103699	0.961125	0.437711	0.894500	34
35	0.102256	0.961792	0.480476	0.887083	35
36	0.099907	0.962521	0.455173	0.892833	36
37	0.095172	0.963042	0.470825	0.891083	37
38	0.098422	0.962896	0.447971	0.893000	38
39	0.090797	0.965250	0.450804	0.893083	39
40	0.095479	0.964229	0.462114	0.894583	40
41	0.089657	0.966396	0.487843	0.894667	41
42	0.086208	0.967062	0.504998	0.894000	42
43	0.087723	0.966125	0.488557	0.893750	43
44	0.082712	0.968500	0.506458	0.894000	44
45	0.081237	0.969979	0.476194	0.900583	45
46	0.080783	0.969292	0.507408	0.893500	46
47	0.078816	0.970021	0.538066	0.893083	47
48	0.077797	0.970438	0.526630	0.896083	48
49	0.075708	0.972125	0.543947	0.892333	49

In [17]:

```
val_acc_per_epoch = history.history['val_accuracy']
best_epoch = val_acc_per_epoch.index(max(val_acc_per_epoch)) + 1
print(best_epoch)
print(val_acc_per_epoch[41])
```

```
46
0.8946666717529297
```

In [18]:

```
hypermodel = tuner.hypermodel.build(best_hps)

# Retrain the model
history = hypermodel.fit(img_train, label_train, epochs=best_epoch, validation_split=0.2, verbose=0)
```

In [19]:

```
md1 = model_builder(best_hps)
history = md1.fit(img_train, label_train, epochs=10, validation_split=0.2, verbose = 0)
```

In [20]:

```
eval_result = hypermodel.evaluate(img_test, label_test)
print("[test loss, test accuracy]:", eval_result)
```