Assignment 6 - Neural Networks

MSDS 422 - SEC 57 THURSDAY

FERDYNAND HEBAL - 8/04/2019

In this assignment, neural network structures are explored within a 2x2 full factorial/crossed benchmark experiment on the Digit Recognition problem in Kaggle.com.

Classification performance accuracy and processing time are assessed using Python TensorFlow. Due to the time required to fit each neural network, only one trial is observed for each cell in the design. Mini-Batch Gradient Descent is used to help avoid local minima while maintaining efficiency of not having all training data in memory. Batch size is set to 40 and number of iterations to 40,000 for all trials in this experiment. Timing and accuracy results are presented in Table 1.

Additionally, the results for one trial using SGDClassifier from SKlearn are presented to provide an example of performance and accuracy using a non-neural network technique.

In [1]:
```
# import base packages into the namespace for this program
import pandas as pd
import numpy as np
import tensorflow as tf
tf.logging.set_verbosity(tf.logging.ERROR)
import time
from sklearn.model_selection import train_test_split
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import accuracy_score
from sklearn.svm import SVC
```

Data preparation, exploration, visualization

As this is a prepared dataset and ready for analysis, no data processing is performed beyond converting imported train.csv to a numpy array for modeling. Printed dataset details using .head, .info is provided below

In [2]:
```
mnist_train = pd.read_csv(r'train.csv')
print(mnist_train.head())
```

```
   label  pixel0  pixel1  pixel2  pixel3  pixel4  pixel5  pixel6  pixel
7  \
0      1       0       0       0       0       0       0       0
0
1      0       0       0       0       0       0       0       0
0
2      1       0       0       0       0       0       0       0
0
3      4       0       0       0       0       0       0       0
0
4      0       0       0       0       0       0       0       0
0

   pixel8    ...     pixel774  pixel775  pixel776  pixel777  pixel778
\
0       0    ...            0         0         0         0         0
1       0    ...            0         0         0         0         0
2       0    ...            0         0         0         0         0
3       0    ...            0         0         0         0         0
4       0    ...            0         0         0         0         0

   pixel779  pixel780  pixel781  pixel782  pixel783
0         0         0         0         0         0
1         0         0         0         0         0
2         0         0         0         0         0
3         0         0         0         0         0
4         0         0         0         0         0

[5 rows x 785 columns]
```

In [3]:
```
mnist_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 42000 entries, 0 to 41999
Columns: 785 entries, label to pixel783
dtypes: int64(785)
memory usage: 251.5 MB
```

Review research design and modeling methods

Splitting test and training data will help to avoid snooping bias and overfitting, I use the sklearn train_test_split function and set 20% test size. Below this, neural network structures are explored within a 2x2 full factorial/crossed benchmark experiment on the Digit Recognition problem in Kaggle.com.

Additionally, results for one trial using SGDClassifier from SKlearn are presented to provide an example of performance and accuracy using a non neural network technique

In [4]:
```
train_data = np.array([mnist_train.values][0])

y = train_data[:,0]
X = train_data[:,1:]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

print('Train dimensions:', X_train.shape)
print('Test dimensions:', X_test.shape)
```

```
Train dimensions: (33600, 784)
Test dimensions: (8400, 784)
```

In [5]:
```
#SGD Classifier results
sgd_clf = SGDClassifier(random_state=42,max_iter=1000,tol=1e-3)

# processing time to fit model
start = time.time()
sgd_clf.fit(X_train, y_train)
done = time.time()
processing_time = done - start

#accuracy in test and train
y_pred = sgd_clf.predict(X_test)
test_accuracy = accuracy_score(y_test, y_pred)
y_pred = sgd_clf.predict(X_train)
train_accuracy = accuracy_score(y_train, y_pred)

#collect results
results = ['na','na',round(processing_time,2), round(train_accuracy,3),
round(test_accuracy,3)]
results_df = pd.DataFrame(results).T
results_df.columns = ['Number of Layers','Nodes per Layer','Processing t
ime'
                      , 'Train set accuracy', 'Test set accuracy']
results_df_sgd = results_df.copy()
results_df_sgd
```

Out[5]:

	Number of Layers	Nodes per Layer	Processing time	Train set accuracy	Test set accuracy
0	na	na	51.85	0.896	0.871

In [6]:
```
#DNN Classifier results for 2 layers with 10 nodes each
feature_cols = tf.contrib.learn.infer_real_valued_columns_from_input(X_t
rain)
dnn_clf = tf.contrib.learn.DNNClassifier(hidden_units=[10,10], n_classes
=10, feature_columns=feature_cols)
dnn_clf = tf.contrib.learn.SKCompat(dnn_clf)  # if TensorFlow >= 1.1

start = time.time()
dnn_clf.fit(X_train, y_train, batch_size=50, steps=40000)
done = time.time()
processing_time = done - start

y_pred = dnn_clf.predict(X_train)
train_accuracy = accuracy_score(y_train, y_pred['classes'])
y_pred = dnn_clf.predict(X_test)
test_accuracy = accuracy_score(y_test, y_pred['classes'])

results = ['2','10',round(processing_time,2), round(train_accuracy,3), r
ound(test_accuracy,3)]
results_df = pd.DataFrame(results).T
results_df.columns = ['Number of Layers','Nodes per Layer','Processing t
ime'
                      , 'Train set accuracy', 'Test set accuracy']
results_df_210 = results_df.copy()
results_df_210
```

```
WARNING: The TensorFlow contrib module will not be included in TensorFl
ow 2.0.
For more information, please see:
  * https://github.com/tensorflow/community/blob/master/rfcs/20180907-c
ontrib-sunset.md
  * https://github.com/tensorflow/addons
If you depend on functionality not listed there, please file an issue.
```

Out[6]:

	Number of Layers	Nodes per Layer	Processing time	Train set accuracy	Test set accuracy
0	2	10	24.63	0.709	0.698

In [7]:
```
#DNN Classifier results for 2 layers with 20 nodes each
feature_cols = tf.contrib.learn.infer_real_valued_columns_from_input(X_t
rain)
dnn_clf = tf.contrib.learn.DNNClassifier(hidden_units=[20,20], n_classes
=10, feature_columns=feature_cols)
dnn_clf = tf.contrib.learn.SKCompat(dnn_clf)  # if TensorFlow >= 1.1

start = time.time()
dnn_clf.fit(X_train, y_train, batch_size=50, steps=40000)
done = time.time()
processing_time = done - start

y_pred = dnn_clf.predict(X_train)
train_accuracy = accuracy_score(y_train, y_pred['classes'])

y_pred = dnn_clf.predict(X_test)
test_accuracy = accuracy_score(y_test, y_pred['classes'])

results = ['2','20',round(processing_time,2), round(train_accuracy,3), r
ound(test_accuracy,3)]
results_df = pd.DataFrame(results).T
results_df.columns = ['Number of Layers','Nodes per Layer','Processing t
ime'
                      , 'Train set accuracy', 'Test set accuracy']
results_df_220 = results_df.copy()
results_df_220
```

Out[7]:

	Number of Layers	Nodes per Layer	Processing time	Train set accuracy	Test set accuracy
0	2	20	26.01	0.93	0.904

In [8]:
```
#DNN Classifier results for 5 layers with 10 nodes each
feature_cols = tf.contrib.learn.infer_real_valued_columns_from_input(X_t
rain)
dnn_clf = tf.contrib.learn.DNNClassifier(hidden_units=[10,10,10,10,10],
n_classes=10, feature_columns=feature_cols)
dnn_clf = tf.contrib.learn.SKCompat(dnn_clf)  # if TensorFlow >= 1.1

start = time.time()
dnn_clf.fit(X_train, y_train, batch_size=50, steps=40000)
done = time.time()
processing_time = done - start

y_pred = dnn_clf.predict(X_train)
train_accuracy = accuracy_score(y_train, y_pred['classes'])

y_pred = dnn_clf.predict(X_test)
test_accuracy = accuracy_score(y_test, y_pred['classes'])

results = ['5','10',round(processing_time,2), round(train_accuracy,3), r
ound(test_accuracy,3)]
results_df = pd.DataFrame(results).T
results_df.columns = ['Number of Layers','Nodes per Layer','Processing t
ime'
                      , 'Train set accuracy', 'Test set accuracy']
results_df_510 = results_df.copy()
results_df_510
```

Out[8]:

	Number of Layers	Nodes per Layer	Processing time	Train set accuracy	Test set accuracy
0	5	10	27.09	0.874	0.848

In [9]:
```
#DNN Classifier results for 5 layers with 20 nodes each
feature_cols = tf.contrib.learn.infer_real_valued_columns_from_input(X_t
rain)
dnn_clf = tf.contrib.learn.DNNClassifier(hidden_units=[20,20,20,20,20],
n_classes=10, feature_columns=feature_cols)
dnn_clf = tf.contrib.learn.SKCompat(dnn_clf)  # if TensorFlow >= 1.1

start = time.time()
dnn_clf.fit(X_train, y_train, batch_size=50, steps=40000)
done = time.time()
processing_time = done - start

y_pred = dnn_clf.predict(X_train)
train_accuracy = accuracy_score(y_train, y_pred['classes'])

y_pred = dnn_clf.predict(X_test)
test_accuracy = accuracy_score(y_test, y_pred['classes'])

results = ['5','20',round(processing_time,2), round(train_accuracy,3), r
ound(test_accuracy,3)]
results_df = pd.DataFrame(results).T
results_df.columns = ['Number of Layers','Nodes per Layer','Processing t
ime'
                      , 'Train set accuracy', 'Test set accuracy']
results_df_520 = results_df.copy()
results_df_520
```

Out[9]:

	Number of Layers	Nodes per Layer	Processing time	Train set accuracy	Test set accuracy
0	5	20	30.24	0.922	0.897

In [13]:
```
# All results are collated into a single table
print("Table 1. Tensor Flow Multi-layer Deep Neural Network.")

res = results_df_210.append(results_df_220).append(results_df_510).appen
d(results_df_520).append(results_df_sgd)
res = pd.DataFrame(['DNNClassifier'
                    ,'DNNClassifier'
                    ,'DNNClassifier'
                    ,'DNNClassifier'
                    ,'SGDClassifier']).join(res.reset_index())
res = res.rename(columns ={0:'Classifier'}).drop(columns={'index'})

res['dif'] = res['Test set accuracy']-res['Train set accuracy']

res
```

```
Table 1. Tensor Flow Multi-layer Deep Neural Network.
```

Out[13]:

	Classifier	Number of Layers	Nodes per Layer	Processing time	Train set accuracy	Test set accuracy	dif
0	DNNClassifier	2	10	24.63	0.709	0.698	-0.011
1	DNNClassifier	2	20	26.01	0.93	0.904	-0.026
2	DNNClassifier	5	10	27.09	0.874	0.848	-0.026
3	DNNClassifier	5	20	30.24	0.922	0.897	-0.025
4	SGDClassifier	na	na	51.85	0.896	0.871	-0.025

Summary

Regarding the management problem, these results suggest that neural networks with a greater number of layers perform better given the same number of nodes, and similarly perform better with a great number of nodes given the same number of layers. The increase in number of layers seems to incur a greater cost in processing time than does the number of nodes. It is clear that even with a simple neural network structure a high level of accuracy can be achieve in train and test sets. Performance by even a simple eural network classifier proved in this experiment to exceed
the performance of at least one example of a non-Neural Network classifier with a more efficient processing time. Further experimentation is warranted to demonstrate changes in additional parameters.

Considering the results from this benchmark study I would conclude that DNN classifier with 5 layers and 20 nodes per layer can exceed performance of non-Neural Network technique with greater efficiency concerning processing time and trustworthy results given performance in test data. From the perspective of a financial institution evaluating machine learning technologies for optical character recognition I would recommend this technique given these parameters with the caveat that additional layers and nodes may result in higher accuracy but performance times will suffer and the risk overfitting increases with additional layers and nodes as demonstrated by the increasing difference in accuracy between test and train data in Table 1.