FINAL EXAM
MACHINE LEARNING

Implement Common Convolution Neural Nets Model

Reproduce RESNET With MNIST Dataset

Arranged by:

1. Rian Muhammad Yusuf
2. Wahyu Mulya Utama

About

ResNet-50 adalah salah satu varian ResNet yang memiliki 50 layer. Jika pada varian ResNet sebelumnya dilakukan skip connection sebanyak 2 layer, maka ResNet-50 melewati 3 layer dan terdapat 1x1 convolution layer. Jumlah bobot yang diperbarui selama proses pelatihan data disebut sebagai learning rate.

MNIST adalah dataset yang terdiri dari angka 0 sampai 9 yang ditulis oleh tangan. MNIST data sudah sering digunakan sebagai benchmark apakah sebuah model dapat mengenali atau melakukan klasifikasi terhadap angka-angka tersebut.

TECHNICAL REPORT

Importing key libraries, and reading data

In [1]:

```
import pandas as pd
import numpy as np

np.random.seed(1212)

import keras
from keras.models import Model
from keras.layers import *
from keras import optimizers
```

```
Using TensorFlow backend.
```

In [2]:

```
df_train = pd.read_csv('../input/train.csv')
df_test = pd.read_csv('../input/test.csv')
```

In [3]:

```
df_train.head() # 784 features, 1 label
```

Out[3]:

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	...	pixel774	pixel775	pixel776	pixel777	pixel778	pixel779	pixel780	pixel
0	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0
3	4	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0

Splitting into training and validation dataset

In [4]:

```
df_features = df_train.iloc[:, 1:785]
df_label = df_train.iloc[:, 0]

X_test = df_test.iloc[:, 0:784]

print(X_test.shape)
```

```
(28000, 784)
```

In [5]:

```
from sklearn.model_selection import train_test_split
X_train, X_cv, y_train, y_cv = train_test_split(df_features, df_label,
                                                test_size = 0.2,
                                                random_state = 1212)

X_train = X_train.as_matrix().reshape(33600, 784) #(33600, 784)
X_cv = X_cv.as_matrix().reshape(8400, 784) #(8400, 784)

X_test = X_test.as_matrix().reshape(28000, 784)
```

Data cleaning, normalization and selection

In [6]:
```
print((min(X_train[1]), max(X_train[1])))
```

```
(0, 255)
```

As the pixel intensities are currently between the range of 0 and 255, we proceed to normalize the features, using broadcasting. In addition, we proceed to convert our labels from a class vector to binary One Hot Encoded

In [7]:
```
# Feature Normalization
X_train = X_train.astype('float32'); X_cv= X_cv.astype('float32'); X_test = X_test.astype('float32')
X_train /= 255; X_cv /= 255; X_test /= 255

# Convert labels to One Hot Encoded
num_digits = 10
y_train = keras.utils.to_categorical(y_train, num_digits)
y_cv = keras.utils.to_categorical(y_cv, num_digits)
```

In [8]:
```
# Printing 2 examples of labels after conversion
print(y_train[0]) # 2
print(y_train[3]) # 7
```

```
[ 0.  0.  1.  0.  0.  0.  0.  0.  0.  0.]
[ 0.  0.  0.  0.  0.  0.  0.  1.  0.  0.]
```

Model Fitting

In [9]:

```
# Input Parameters
n_input = 784 # number of features
n_hidden_1 = 300
n_hidden_2 = 100
n_hidden_3 = 100
n_hidden_4 = 200
num_digits = 10
```

In [10]:

```
Inp = Input(shape=(784,))
x = Dense(n_hidden_1, activation='relu', name = "Hidden_Layer_1")(Inp)
x = Dense(n_hidden_2, activation='relu', name = "Hidden_Layer_2")(x)
x = Dense(n_hidden_3, activation='relu', name = "Hidden_Layer_3")(x)
x = Dense(n_hidden_4, activation='relu', name = "Hidden_Layer_4")(x)
output = Dense(num_digits, activation='softmax', name = "Output_Layer")(x)
```

In [11]:

```
# Our model would have '6' layers - input layer, 4 hidden layer and 1 output layer
model = Model(Inp, output)
model.summary() # We have 297,910 parameters to estimate
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 784)               0
_________________________________________________________________
Hidden_Layer_1 (Dense)       (None, 300)               235500
_________________________________________________________________
Hidden_Layer_2 (Dense)       (None, 100)               30100
_________________________________________________________________
Hidden_Layer_3 (Dense)       (None, 100)               10100
_________________________________________________________________
Hidden_Layer_4 (Dense)       (None, 200)               20200
```

Model Train

In [12]:
```
# Insert Hyperparameters
learning_rate = 0.1
training_epochs = 20
batch_size = 100
sgd = optimizers.SGD(lr=learning_rate)
```

In [13]:
```
# We rely on the plain vanilla Stochastic Gradient Descent as our optimizing methodology
model.compile(loss='categorical_crossentropy',
              optimizer='sgd',
              metrics=['accuracy'])
```

In [14]:
```
history1 = model.fit(X_train, y_train,
                     batch_size = batch_size,
                     epochs = training_epochs,
                     verbose = 2,
                     validation_data=(X_cv, y_cv))
```

```
Train on 33600 samples, validate on 8400 samples
Epoch 1/20
4s - loss: 1.8541 - acc: 0.4983 - val_loss: 1.0046 - val_acc: 0.7602
Epoch 2/20
4s - loss: 0.6480 - acc: 0.8295 - val_loss: 0.4635 - val_acc: 0.8729
Epoch 3/20
4s - loss: 0.4092 - acc: 0.8834 - val_loss: 0.3616 - val_acc: 0.8980
Epoch 4/20
4s - loss: 0.3373 - acc: 0.9026 - val_loss: 0.3121 - val_acc: 0.9100
Epoch 5/20
4s - loss: 0.2979 - acc: 0.9139 - val_loss: 0.2893 - val_acc: 0.9169
Epoch 6/20
4s - loss: 0.2684 - acc: 0.9227 - val_loss: 0.2651 - val_acc: 0.9238
```

Dibandingkan dengan model pertama kami, menambahkan lapisan tambahan tidak secara signifikan meningkatkan akurasi dari model kami sebelumnya. Namun, ada biaya komputasi (dalam hal kompleksitas) dalam menerapkan lapisan tambahan di Neural Network. Mengingat bahwa manfaat dari lapisan tambahan rendah sementara biayanya tinggi, kami akan tetap menggunakan Neural Network 4 lapisan. Selanjutnya untuk memasukkan Drop out (drop out rate : 0,3) dalam model kedua kami untuk mencegah overfitting.

Dengan skor validasi mendekati 98%, kami melanjutkan menggunakan model ini untuk memprediksi set pengujian (Test).

In [31]:

```
test_pred = pd.DataFrame(model4.predict(X_test, batch_size=200))
test_pred = pd.DataFrame(test_pred.idxmax(axis = 1))
test_pred.index.name = 'ImageId'
test_pred = test_pred.rename(columns = {0: 'Label'}).reset_index()
test_pred['ImageId'] = test_pred['ImageId'] + 1

test_pred.head()
```

Out[31]:

	ImageId	Label
0	1	2
1	2	0
2	3	9
3	4	9
4	5	3

In [32]:

```
test_pred.to_csv('mnist_submission.csv', index = False)
```

TERIMA KASIH