Character Classification

2019/3/22 성남-KAIST 인공지능 집중교육과정

Tip> shotcuts for Jupyter Notebook

- Shift + Enter : run cell and select below

Library

- Numpy: Fundamenta package for scientific computing with Python
- Tensorflow: An open source machine learning library for research and production
- String : contains a number of functions to process standard Python strings(a series of characters)

```
import tensorflow as tf
import numpy as np
import pandas as pd
import os.path
import string
import time
import matplotlib.pyplot as plt

model_save_path = 'tmp/model.ckpt'
tf.reset_default_graph()
```

```
learning_rate = 0.01

all_letters = string.ascii_letters + " .,;'"
n_input = len(all_letters)
n_hidden = [256, 256] # hidden layer features
max_sequence_length = 19 # maximum number of characters is 19
l2_lambda = 1e-4


alphabet = all_letters
ethnicities = ['Chinese', 'Japanese', 'Vietnamese', 'Korean',
'Arabic','Czech','Dutch','English','French','German','Greek','Irish','Italian','Polish'
,'Portuguese','Russian','Scottish','Spanish']
n_classes = len(ethnicities) # the number of classes

name_strings = []
ethnicity_strings = []
str_list = []
names_list = []
ethnicity_list = []
```

1. Load data

```
data = pd.read_csv('names_revised.csv')
data.head(5)
```

	Family name	Ethnicity
0	Khoury	Arabic
1	Nahas	Arabic
2	Daher	Arabic
3	Gerges	Arabic
4	Nazari	Arabic

10 examples for Korean family name

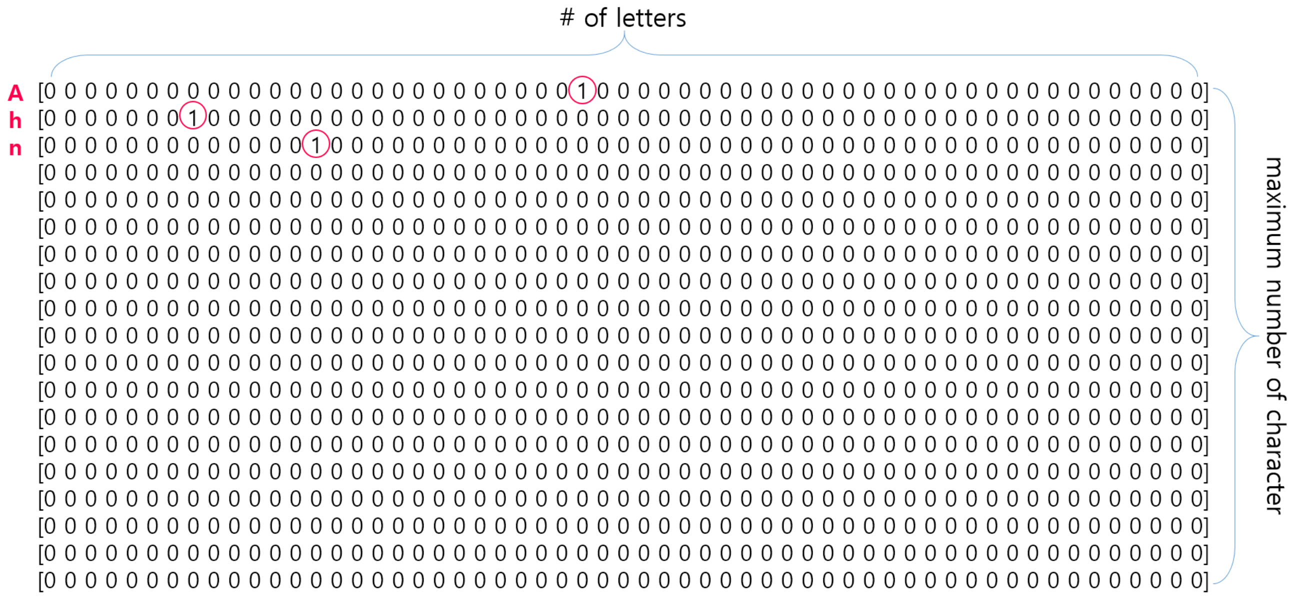
```
ids = np.where(data.loc[:, [' Ethnicity']] == 'Korean')
kor = data.loc[ids[0], :]
kor.head(10)
```

	Family name	Ethnicity
9888	Ahn	Korean
9889	Baik	Korean
9890	Bang	Korean
9891	Byon	Korean
9892	Cha	Korean
9893	Chang	Korean
9894	Chi	Korean
9895	Chin	Korean
9896	Cho	Korean
9897	Choe	Korean

Word Embedding to One-hot vectors

'a' is converted to a vector [1 0]

For example, 'Ahn' is converted to




```
def name_one_hot(name, max_sequence_length):
    result = []
    for char in name:
        v = np.zeros(n_input, dtype=np.int) # count space as a character
        v[alphabet.index(char)] = 1
        result.append(v)
    while len(result) < max_sequence_length:
        result.append(np.zeros(n_input, dtype=np.int))
    result = np.array(result)
    return result
```

```
def ethnicity_one_hot(ethnicity):
    v = np.zeros(n_classes, dtype=np.int)
    v[ethnicities.index(ethnicity)] = 1
    return v
```

```
for line in zip(data['Family name'], data[' Ethnicity']):
    if(line[1] in ethnicities):
        name_strings.append(line[0])
        ethnicity_strings.append(line[1])
        if len(line[0]) > max_sequence_length:
            line[0] = line[0][:max_sequence_length]
        names_list.append(name_one_hot(line[0], max_sequence_length)) # one-hot vector of each characters of name
        ethnicity_list.append(ethnicity_one_hot(line[1]))            # one-hot vector of ethnicity
```

Training - Test Seperation

Split the data for training and test with 9:1 ratio

```
rng_state = np.random.get_state() # use the same random number generator state
np.random.shuffle(names_list)     # when shuffling the two lists
np.random.set_state(rng_state)    # they are effectively shuffled in parallel so that
inputs still correspond to outputs after shuffling
np.random.shuffle(ethnicity_list)
```

```
size = len(names_list)
train_size = int(size*0.9)

training_X = np.array(names_list[:train_size])
training_y = np.array(ethnicity_list[:train_size])
testing_X = np.array(names_list[train_size:])
testing_y = np.array(ethnicity_list[train_size:])
```

Test set examples

```
print("[*] Test set examples for Korean\n")
for name, nation in zip(testing_X, testing_y):
    if np.where(nation)[0] == 3:
        for char in name:
            if np.size(np.where(char)[0]) > 0:
                print(alphabet[np.where(char)[0][0]], end='')
        print('\n')
```

```
[*] Test set examples for Korean

Ma

Chu

Choe

Gwang

Youj

Lee

Kwak

Oh

Chi
```

2. Build a graph

```
X = tf.placeholder(tf.float32, [None, max_sequence_length, n_input])
y = tf.placeholder(tf.float32, [None, n_classes])
```

DNN structure (Fill in the blanks)

```
x = tf.reshape(X, [-1, max_sequence_length * n_input])

w_init = tf.variance_scaling_initializer()
b_init = tf.constant_initializer(0.)

## 1st hidden layer
w1 =                                # weight for 1st hidden layer which have 256 units
b1 =                                # bias for 1st hidden layer which have 256 units
h  =                                # matrix multiplication
h  =                                # relu activation

## 2nd hidden layer
w2 =                                # weight for 2nd hidden layer which have 256 units
b2 =                                # bias for 2nd hidden layer which have 256 units
h  =                                # matrix multiplication
h  =                                # relu activation

## output layer
w3 =                                # weight for output layer which have 256 units

y_pred = tf.matmul(h, w)
```

Loss function

```
cent = tf.nn.softmax_cross_entropy_with_logits_v2(logits=y_pred, labels=y)
loss = tf.reduce_mean(cent)
```

L2 regularization

```
all_var = [var for var in tf.trainable_variables() ]
l2_losses = []
for var in all_var:
    if var.op.name.find('weight') == 0:
        l2_losses.append(tf.nn.l2_loss(var))

reg_losses = loss + _l2_lambda * tf.reduce_sum(l2_losses)
```

Momentum optimizer (Fill in the blanks)

```
train_step =                                        # Momentum optimizer with
momentum 0.9 to minimize "reg_loss"
```

Evaluate & Predict

```
## Evaluation
correct_prediction = tf.equal(tf.argmax(y_pred,1), tf.argmax(y,1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

## Softmax
pred = tf.nn.softmax(y_pred)
```

3. Train a model

```
## MAKE SESSION
sess = tf.InteractiveSession()

## INITIALIZE SESSION
init = tf.global_variables_initializer()
sess.run(init)

## Saver
saver = tf.train.Saver()
```

```
n_epoch = 500
start_time = time.time()
losses = []

for epoch in range(n_epoch+1):
    _, avg_loss = sess.run([train_step, loss], feed_dict={X: training_X, y:
training_y})
    losses.append(avg_loss)

    if epoch%100 == 0:
        train_accuracy = accuracy.eval(feed_dict={X:training_X, y:training_y})
        test_accuracy = accuracy.eval(feed_dict={X:testing_X, y:testing_y})
        current_time = time.time() - start_time
        print("[*] Step %d, Training accuracy %.4f // Testing accuracy %.4f // Time:
%3.2f"%(epoch, train_accuracy, test_accuracy, current_time))

saver.save(sess, model_save_path)
print("Model saved in file: %s" % model_save_path)
```

Plot the loss function

```
plt.plot(losses)
plt.title("Learning curve", fontsize=14, fontweight='bold')
plt.xlabel("Epochs", fontsize=14, fontweight='bold')
plt.ylabel("RMSE of training set", fontsize=14, fontweight='bold')
plt.show()
```

Type 5 last names for test

```
i=0
while i<5:
    input_name = input('Enter a last name (max 19 letters):')

    while len(input_name) > max_sequence_length or len(input_name) == 0:
        input_name = raw_input('Invalid input. Enter a last name (max 19 letters):')

    result=pred.eval(feed_dict={X: np.expand_dims(name_one_hot(input_name, 19),
axis=0)})[0]
    idx = np.argsort(result)[::-1]
    print("\n(%s): %.4f" % (ethnicities[idx[0]], result[idx[0]]))
    print("(%s): %.4f" % (ethnicities[idx[1]], result[idx[1]]))
    print("(%s): %.4f" % (ethnicities[idx[2]], result[idx[2]]))
    print("========================================")
    i=i+1
```

```
n_parameters = 0
for var in tf.trainable_variables():
    n_parameters += tf.size(var)
n_dnn = sess.run(n_parameters)
print("The number of parameters %d" % sess.run(n_parameters))
```

```
sess.close()
tf.reset_default_graph()
```

4. Report

1. Fill in the blank code

Design a DNN model with 2 hidden layers which have 256 units. Minimize the loss function using Momentum optimizer with momenum 0.9.

2. Adam Optimizer

Use the "Adam Optimizer( )" instead of the MomentumOptimizer and compare the RMSE learning curves of the two optimizers.
Hint) tf.train.AdamOptimizer( )