APPENDIX A: CHIROPRACTIC PRELICENSURE EXAMS

The NBCE licensure examination is comprised of four separate parts: Part I (assesses academic proficiency in six basic science areas), Part II (assesses academic proficiency in six clinical science areas), Part III (assesses application of knowledge in nine areas of clinical competency), and Part IV (Objective Structure Clinical Examination, assesses skills similar to those chiropractors might encounter in practice). Each part has been constructed in strict accordance with measurement procedures described in the Standards for Educational and Psychological Testing (AERA, APA, and NCME, 2014).

For example, Part III examines the following nine clinical areas:

- Case History
- Physical Examination
- Neuromusculoskeletal Examination
- Diagnostic Imaging
- Clinical Laboratory and Special Studies
- Diagnosis or Clinical Impression
- Chiropractic Techniques
- Supportive Interventions
- Case Management

Part III consists of two sections, with a total of 110 standard multiple-choice questions and 10 case vignettes, broken down as follows:

- Each section has 55 standard multiple-choice questions, plus five case vignettes.
- Each of the five case vignettes contains three extended multiple-choice questions.
- Each extended multiple-choice question requires three answers.

APENDIX B: PYTHON SOURCE CODE

```
# coding: utf-8

# In[1]:

import numpy as np
from utils import *
import random
import tensorflow as tf
print(tf.__version__)
from numpy.random import uniform as unif
# To support both python 2 and python 3
from __future__ import division, print_function, unicode_literals
import os

# to make this notebook's output stable across runs
def reset_graph(seed=42):
    tf.reset_default_graph()
    tf.set_random_seed(seed)
    np.random.seed(seed)

# To plot pretty figures
get_ipython().magic('matplotlib inline')
import matplotlib
import matplotlib.pyplot as plt
plt.rcParams['axes.labelsize'] = 14
plt.rcParams['xtick.labelsize'] = 12
plt.rcParams['ytick.labelsize'] = 12

# Where to save the figures
PROJECT_ROOT_DIR = "."
CHAPTER_ID = "tensorflow"

def save_fig(fig_id, tight_layout=True):
    path = os.path.join(PROJECT_ROOT_DIR, "images", CHAPTER_ID, fig_id + ".png")
    print("Saving figure", fig_id)
    if tight_layout:
        plt.tight_layout()
    plt.savefig(path, format='png', dpi=300)
from sklearn.preprocessing import StandardScaler
from  scipy.stats import rankdata as rankdata

# # Specify The Number Of Data Structures
# In[2]:

taker_num = 800
testlet_num = 6
item_num = 5
diff_level_num = 3

# # Generate Real Profeciency Data
```

In[3]:

```
PROF = np.random.randn(taker_num, 1 + testlet_num)
```

Generate Data Related To Testlet Items

In[4]:

```
testlet_lists = {}
for testlet_idx in range(testlet_num):
    testlet_lists[testlet_idx] = {"a_n": np.random.uniform(high=1.4, low=0.7,size=(item_num, 2)),
"t_n":np.random.randn(item_num, diff_level_num) * 1 + 0.3}
```

Testlet Response Function

In[5]:

```
def get_response_to_testlet_items(prof_n, a_n, t_n):
   tmp_exponent_n = np.dot(a_n, prof_n ) - t_n
   exponent_n = np.c_[np.zeros((item_num,1)), tmp_exponent_n]
   probs_n = np.exp( np.cumsum(exponent_n, axis=1) ) / np.sum(np.exp( np.cumsum(exponent_n, axis=1) ), axis =
1, keepdims=True)
   Y_com = np.zeros((item_num,))
   for i in range(item_num):
      ind = np.random.choice(diff_level_num + 1, p = probs_n[i,:])
      Y_com[i] = ind
   return Y_com
```

Response Data Y_comp: MATRIX [taker_num, item_num, testlet_num]

In[6]:

```
Y_comp = np.zeros((taker_num, item_num, testlet_num))
for testlet_idx in range(testlet_num):
   test_a_param = testlet_lists[testlet_idx]['a_n']
   test_t_param = testlet_lists[testlet_idx]['t_n']
   for taker_idx in range(taker_num):
      indiv_prof = PROF[taker_idx, [0, testlet_idx + 1]].reshape(2,1)
      Y_cp = get_response_to_testlet_items(indiv_prof, test_a_param, test_t_param)
      Y_comp[taker_idx, :, testlet_idx] = Y_cp
```

Convert Y_comp to One-Hot

In[7]:

```
Y_onehot = {}
for testlet_idx in range(testlet_num):
   temp_onehot = dict()
   for taker_idx in range(taker_num):
      y_oh = np.zeros((item_num, diff_level_num + 1))
      for item_idx in range(item_num):
         y_oh[item_idx, Y_comp[taker_idx,item_idx, testlet_idx]] = 1
      temp_onehot[taker_idx] = y_oh
```

```
    Y_onehot[testlet_idx] = temp_onehot

# In[ ]:

# # Score For Each Category & Taker

# In[8]:

Score_taker_testlet = np.sum(Y_comp,  axis = 1)
Score_taker = np.sum(Score_taker_testlet, axis = 1, keepdims=True)

# # Combine All The Scores For Initial Guess

# In[9]:

Score_for_init = np.c_[Score_taker, Score_taker_testlet ]

# # Generate Initial Guess From Responses

# In[10]:

Prof_init = np.zeros_like(Score_for_init)
for idx in range(testlet_num + 1):
    new_rank = rankdata(Score_for_init[: , idx ], method='ordinal')-1
    init_guess = np.random.randn(taker_num)
    Prof_init[:, idx ]= np.sort(init_guess)[new_rank] * 1.2

# # Important Data Structure

# In[122]:

# PROF : Matrix (2000, 11)
# Y_comp : Matrix (2000, 5, 10)
# testlet_lists: DICT{ n: {"a_n": Matrix(item_num, 2), "b_n": Matrix(item_num, diff_level_num)}}
# Y_onehot[testlet_idx][taker_idx]

# # Hide Real Prof Data Replace With Initialized Guess

# In[11]:

PROF_HIDDEN_REAL = np.copy(PROF)
PROF = Prof_init

# # Get Prof Accuracy

# In[12]:

def get_prof_accuracy(prof_est):
    return np.linalg.norm(PROF_HIDDEN_REAL  - prof_est)

# In[13]:

get_prof_accuracy(Prof_init)
```

```
# # Given Taker Index and Testlet Index Return Matrix(2,1)

# In[14]:

def get_prof_slice(testee_ind, testlet_ind, PR=PROF):
    return PR[testee_ind, [0, testlet_ind+1]].T.reshape(2,1)

def get_response(testee_ind, testlet_ind):
    return Y_comp[testee_ind, :, testlet_ind]

def get_onehot_response(testee_ind, testlet_ind):
    return Y_onehot[testlet_ind][testee_ind]

# # Scale PROF Making Sure It Has 0 Mean 1 Variance

# In[15]:

for col_idx in range(testlet_num+1):
    PROF[:, col_idx] = (PROF[:, col_idx] - np.mean(PROF[:, col_idx])) / np.std(PROF[:, col_idx])

# # Generate Tensorflow Parts

# In[16]:

reset_graph()
#prof = tf.constant(prof_n, dtype=tf.float32, name="prof")
prof = tf.placeholder(dtype=tf.float32, shape=(2,1), name="prof")
#Y = tf.constant(Y_n, dtype=tf.float32, name="Y" )
Y = tf.placeholder(dtype=tf.float32, shape=(item_num, diff_level_num+1), name="Y")

ap = tf.Variable( tf.random_uniform(minval=-2, maxval=2, shape=(item_num,2)), dtype=tf.float32)
a = 0.65 + 0.8 * tf.sigmoid(ap)
t = tf.Variable(tf.random_normal(shape=(item_num,3)), dtype=tf.float32)
tmp_exponent = tf.matmul(a,prof) - t
exponent1 = tf.concat([np.zeros((item_num,1)), tmp_exponent] , axis=1)
exponent2 = tf.cumsum(exponent1, axis=1)
probs = tf.nn.softmax(exponent2)

entropy = tf.nn.softmax_cross_entropy_with_logits(logits = exponent2, labels = Y)
err = tf.reduce_mean(entropy)

#optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.02)
optimizer = tf.train.MomentumOptimizer(learning_rate=0.02, momentum=0.9)
training_op = optimizer.minimize(err)
init = tf.global_variables_initializer()

#PROF = PROF_UPDATED

# In[17]:

PROF.shape
get_prof_accuracy(PROF)
```

```
# # Function Used For Prof Estimation

# In[18]:

def cal_one_testlet(testlet_idx, prof, A_all, T_all, Y_all):
    A_all_0 = tf.reshape(A_all[testlet_idx, : , 0], shape=(item_num,1))
    A_all_1 = tf.reshape(A_all[testlet_idx, : , 1], shape=(item_num,1))
    a_all = A_all_0 * prof[0] + A_all_1 * prof[testlet_idx + 1]
    tmp_exponent = a_all - T_all[testlet_idx]
    exponent1 = tf.concat([np.zeros((item_num,1)), tmp_exponent] , axis=1)
    exponent2 = tf.cumsum(exponent1, axis=1)
    labs = tf.one_hot(Y_all[testlet_idx], depth = diff_level_num+1)
    entropy = tf.nn.softmax_cross_entropy_with_logits(logits = exponent2, labels = labs)
    err = tf.reduce_mean(entropy)
    return err

# # Estimate Test Item Parameters

# In[19]:

testlet_est_lists = {}
for testlet_ind in range(testlet_num):
    t_n = testlet_lists[testlet_ind]['t_n']
    a_n = testlet_lists[testlet_ind]['a_n']
    stoch_ind = np.arange(taker_num)
    with tf.Session() as sess:
        sess.run(init)
        for iter in range(10):
            np.random.shuffle(stoch_ind)
            for epoch in range(taker_num):
            #if epoch % 20 == 0:
            #   print("Epoch", epoch, "MSE =", mse.eval())
                testee_ind = stoch_ind[epoch]
                prof_ep = get_prof_slice(testee_ind, testlet_ind, PROF)
                Y_ep = get_onehot_response(testee_ind, testlet_ind)
                sess.run(training_op, feed_dict={prof: prof_ep, Y: Y_ep})
                #if epoch % 1000 == 0:
        errv, av, tv = sess.run([err, a, t], feed_dict={prof: prof_ep, Y: Y_ep})
        print(errv, "", np.linalg.norm(a_n-av), np.linalg.norm(t_n-tv))
            #errv, av, tv = sess.run([err, a, t], feed_dict={prof: prof_ep, Y: Y_ep})
    testlet_est_lists[testlet_ind] = {"a_n": av, "t_n": tv}
    print(testlet_ind)

# In[19]:

#testlet_est_lists[4]['t_n']

# In[21]:

#testlet_lists[4]['t_n']

# In[74]:
```

```
plt.plot(testlet_lists[4]['t_n'], testlet_est_lists[4]['t_n'], 'bo')

# # Convert Item Parameters From List To 3d Matrix

# In[23]:

an_3d = np.zeros((testlet_num,item_num, 2) )
tn_3d = np.zeros((testlet_num,item_num, diff_level_num) )
for testlet_idx in range(testlet_num):
   #an_3d[testlet_idx, :, :] = testlet_lists[testlet_idx]['a_n']
   an_3d[testlet_idx, :, :] = testlet_est_lists[testlet_idx]['a_n']
   tn_3d[testlet_idx, :, :] = testlet_est_lists[testlet_idx]['t_n']

# # Estimate Profeciency Parameters

# In[21]:

an_3d = np.zeros((testlet_num,item_num, 2) )
tn_3d = np.zeros((testlet_num,item_num, diff_level_num) )
for testlet_idx in range(testlet_num):
   #an_3d[testlet_idx, :, :] = testlet_lists[testlet_idx]['a_n']
   an_3d[testlet_idx, :, :] = testlet_est_lists[testlet_idx]['a_n']
   tn_3d[testlet_idx, :, :] = testlet_est_lists[testlet_idx]['t_n']
Prof_est = np.zeros_like(PROF)

# In[22]:

for taker_idx in range(taker_num):
   reset_graph()
   prof = tf.Variable(PROF[taker_idx, :].T.reshape((testlet_num+1,1)), dtype=tf.float32, name="prof")
   #prof = tf.Variable(tf.random_normal(shape=(testlet_num + 1,1)), dtype=tf.float32, name="prof")
   #All Responses 3-d tensor
   Y_all = tf.placeholder(dtype=tf.int32, shape=(testlet_num, item_num), name="Y_all")
   A_all = tf.constant(an_3d, dtype=tf.float32)
   T_all = tf.constant(tn_3d, dtype=tf.float32)
   ERR = tf.add_n ( [cal_one_testlet(testlet_idx, prof, A_all, T_all, Y_all) for testlet_idx in range(testlet_num)] )
   #optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.01)
   optimizer = tf.train.MomentumOptimizer(learning_rate=0.01,momentum=0.9)
   training_op = optimizer.minimize(ERR)
   init = tf.global_variables_initializer()

   with tf.Session() as sess:
      sess.run(init)
      for epoch in range(20):
         sess.run(training_op, feed_dict={Y_all: Y_comp[taker_idx].T})
      errv, profv = sess.run([ERR, prof], feed_dict={Y_all: Y_comp[taker_idx].T})
      Prof_est[taker_idx, :] = profv.ravel()
      print(taker_idx, np.linalg.norm(PROF[taker_idx, :] - PROF_HIDDEN_REAL[taker_idx, :]))
      print(taker_idx, np.linalg.norm(profv.reshape((1,-1)) - PROF_HIDDEN_REAL[taker_idx, :]))
      print("\n")

# In[26]:
```

```
plt.plot(PROF_HIDDEN_REAL[:,0], PROF[:,0], 'bo')

# In[75]:

plt.plot(PROF_HIDDEN_REAL[:,0], Prof_est[:,0], 'bo')

# In[25]:

for col_idx in range(testlet_num+1):
   PROF[:, col_idx] = (Prof_est[:, col_idx] - np.mean(Prof_est[:, col_idx])) / np.std(Prof_est[:, col_idx])

# In[27]:

reset_graph()
#prof = tf.constant(prof_n, dtype=tf.float32, name="prof")
prof = tf.placeholder(dtype=tf.float32, shape=(2,1), name="prof")
#Y = tf.constant(Y_n, dtype=tf.float32, name="Y" )
Y = tf.placeholder(dtype=tf.float32, shape=(item_num, diff_level_num+1), name="Y")

ap = tf.Variable( tf.random_uniform(minval=-2, maxval=2, shape=(item_num,2)), dtype=tf.float32)
a = 0.65 + 0.8 * tf.sigmoid(ap)
t = tf.Variable(tf.random_normal(shape=(item_num,3)), dtype=tf.float32)
tmp_exponent = tf.matmul(a,prof) - t
exponent1 = tf.concat([np.zeros((item_num,1)), tmp_exponent] , axis=1)
exponent2 = tf.cumsum(exponent1, axis=1)
probs = tf.nn.softmax(exponent2)

entropy = tf.nn.softmax_cross_entropy_with_logits(logits = exponent2, labels = Y)
err = tf.reduce_mean(entropy)

#optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.02)
optimizer = tf.train.MomentumOptimizer(learning_rate=0.02, momentum=0.9)
training_op = optimizer.minimize(err)
init = tf.global_variables_initializer()
testlet_est_lists = {}
for testlet_ind in range(testlet_num):
   t_n = testlet_lists[testlet_ind]['t_n']
   a_n = testlet_lists[testlet_ind]['a_n']
   stoch_ind = np.arange(taker_num)
   with tf.Session() as sess:
      sess.run(init)
      for iter in range(10):
         np.random.shuffle(stoch_ind)
         for epoch in range(taker_num):
         #if epoch % 20 == 0:
         #   print("Epoch", epoch, "MSE =", mse.eval())
            testee_ind = stoch_ind[epoch]
            prof_ep = get_prof_slice(testee_ind, testlet_ind, Prof_est)
            Y_ep = get_onehot_response(testee_ind, testlet_ind)
            sess.run(training_op, feed_dict={prof: prof_ep, Y: Y_ep})
            #if epoch % 1000 == 0:
      errv, av, tv = sess.run([err, a, t], feed_dict={prof: prof_ep, Y: Y_ep})
```

```
        print(errv, "", np.linalg.norm(a_n-av), np.linalg.norm(t_n-tv))
            #errv, av, tv = sess.run([err, a, t], feed_dict={prof: prof_ep, Y: Y_ep})
    testlet_est_lists[testlet_ind] = {"a_n": av, "t_n": tv}
    print(testlet_ind)

# In[28]:

Prof_est = np.zeros_like(PROF)

# In[30]:

for taker_idx in range(taker_num):
    reset_graph()
    prof = tf.Variable(PROF[taker_idx, :].T.reshape((testlet_num+1,1)), dtype=tf.float32, name="prof")
    #prof = tf.Variable(tf.random_normal(shape=(testlet_num + 1,1)), dtype=tf.float32, name="prof")
    #All Responses 3-d tensor
    Y_all = tf.placeholder(dtype=tf.int32, shape=(testlet_num, item_num), name="Y_all")
    A_all = tf.constant(an_3d, dtype=tf.float32)
    T_all = tf.constant(tn_3d, dtype=tf.float32)
    ERR = tf.add_n ( [cal_one_testlet(testlet_idx, prof, A_all, T_all, Y_all) for testlet_idx in range(testlet_num)] )
    #optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.01)
    optimizer = tf.train.MomentumOptimizer(learning_rate=0.01,momentum=0.9)
    training_op = optimizer.minimize(ERR)
    init = tf.global_variables_initializer()

    with tf.Session() as sess:
        sess.run(init)
        for epoch in range(20):
            sess.run(training_op, feed_dict={Y_all: Y_comp[taker_idx].T})
        errv, profv = sess.run([ERR, prof], feed_dict={Y_all: Y_comp[taker_idx].T})
        Prof_est[taker_idx, :] = profv.ravel()
        print(taker_idx, np.linalg.norm(PROF[taker_idx, :] - PROF_HIDDEN_REAL[taker_idx, :]))
        print(taker_idx, np.linalg.norm(profv.reshape((1,-1)) - PROF_HIDDEN_REAL[taker_idx, :]))
        print("\n")

# In[64]:

np.linalg.norm(Prof_est - PROF_HIDDEN_REAL)

# In[65]:

np.linalg.norm(PROF - PROF_HIDDEN_REAL)

# In[61]:

plt.plot(PROF_HIDDEN_REAL[:taker_idx,0], Prof_est[:taker_idx,0], 'bo')

# In[77]:

plt.plot(PROF_HIDDEN_REAL[:taker_idx,5], PROF[:taker_idx,5], 'bo')

# In[38]:
```

```
taker_idx = taker_idx -1

# In[66]:

np.linalg.norm(Prof_est[:taker_idx,0] - PROF_HIDDEN_REAL[:taker_idx,0])
#np.linalg.norm(PROF - PROF_HIDDEN_REAL)
# In[67]:

np.linalg.norm(PROF[:taker_idx,0] - PROF_HIDDEN_REAL[:taker_idx,0])

# In[73]:

plt.plot(testlet_est_lists[5]['a_n'].ravel(), testlet_lists[5]['a_n'].ravel(), 'ro', )

# # Write Response Data To CSV File For R Comparison

# In[42]:

Y_comp.shape

# In[48]:

Y_comp[17]

# In[50]:

Y_comp[17].T.ravel()

# In[51]:

Y2R = np.zeros((taker_num, testlet_num * item_num))
for taker_idx in range(taker_num):
    Y2R[taker_idx,:] = Y_comp[taker_idx].T.ravel()

# In[58]:

get_ipython().magic('pinfo Y2R.save')

# In[63]:

np.savetxt("Y_800_6_5.csv", Y2R, delimiter=',')

# In[71]:

np.savetxt("prof_real.csv", PROF_HIDDEN_REAL, delimiter=',')
np.savetxt("prof_est.csv", Prof_est, delimiter=',')

# In[98]:

an_2d_est = np.zeros((testlet_num * item_num, 2) )
tn_2d_est = np.zeros((testlet_num * item_num, diff_level_num) )
for testlet_idx in range(testlet_num):
    an_2d_est[(testlet_idx * item_num) : (testlet_idx + 1)*item_num , :] = testlet_est_lists[testlet_idx]['a_n']
```

```
    tn_2d_est[(testlet_idx * item_num) : (testlet_idx + 1)*item_num , :] = testlet_est_lists[testlet_idx]['t_n']
an_2d_real = np.zeros((testlet_num * item_num, 2) )
tn_2d_real = np.zeros((testlet_num * item_num, diff_level_num) )
for testlet_idx in range(testlet_num):
    an_2d_real[(testlet_idx * item_num) : (testlet_idx + 1)*item_num , :] = testlet_lists[testlet_idx]['a_n']
    tn_2d_real[(testlet_idx * item_num) : (testlet_idx + 1)*item_num , :] = testlet_lists[testlet_idx]['t_n']

# In[103]:

an_2d_real = np.zeros((testlet_num * item_num, 2) )
tn_2d_real = np.zeros((testlet_num * item_num, diff_level_num) )
for testlet_idx in range(testlet_num):
    an_2d_real[(testlet_idx * item_num) : (testlet_idx + 1)*item_num , :] = testlet_lists[testlet_idx]['a_n']
    tn_2d_real[(testlet_idx * item_num) : (testlet_idx + 1)*item_num , :] = testlet_lists[testlet_idx]['t_n']

# In[109]:

plt.plot(an_2d_real[:,0], an_2d_est[:,0],'ro')

# In[110]:

np.savetxt("an_2d_est.csv", an_2d_est, delimiter=',')

# In[111]:

np.savetxt("an_2d_real.csv", an_2d_real, delimiter=',')

# In[112]:

np.savetxt("tn_2d_est.csv", tn_2d_est, delimiter=',')

# In[113]:

np.savetxt("tn_2d_real.csv", tn_2d_real, delimiter=',')

# # Big Loop

# In[88]:

testlet_est_lists = estimate_item_params(Prof_est, 5)

######### Individual Profeciency ####################
Prof_est = estimate_ind_params(testlet_est_lists, iter_num=20, learningrate=0.01)

# In[89]:

for col_idx in range(testlet_num+1):
    PROF[:, col_idx] = (Prof_est[:, col_idx] - np.mean(Prof_est[:, col_idx])) / np.std(Prof_est[:, col_idx])
plot_prof_comparison(0, PROF)

# In[93]:

#plt.plot(PROF_HIDDEN_REAL[:taker_idx,5], PROF[:taker_idx,5], 'bo')
```

```
plot_prof_comparison(5, PROF)

# In[55]:

def plot_prof_comparison(testlet_idx, prv):
    plt.plot(PROF_HIDDEN_REAL[:, testlet_idx], prv[:, testlet_idx], 'bo')

# # Convert Item Parameter Estimation Into A Function

# ## Backup Code

# In[60]:

for col_idx in range(testlet_num+1):
    PROF[:, col_idx] = (Prof_est[:, col_idx] - np.mean(Prof_est[:, col_idx])) / np.std(Prof_est[:, col_idx])
reset_graph()
#prof = tf.constant(prof_n, dtype=tf.float32, name="prof")
prof = tf.placeholder(dtype=tf.float32, shape=(2,1), name="prof")
#Y = tf.constant(Y_n, dtype=tf.float32, name="Y" )
Y = tf.placeholder(dtype=tf.float32, shape=(item_num, diff_level_num+1), name="Y")

ap = tf.Variable( tf.random_uniform(minval=-2, maxval=2, shape=(item_num,2)), dtype=tf.float32)
a = 0.65 + 0.8 * tf.sigmoid(ap)
t = tf.Variable(tf.random_normal(shape=(item_num,3)), dtype=tf.float32)
tmp_exponent = tf.matmul(a,prof) - t
exponent1 = tf.concat([np.zeros((item_num,1)), tmp_exponent] , axis=1)
exponent2 = tf.cumsum(exponent1, axis=1)
probs = tf.nn.softmax(exponent2)

entropy = tf.nn.softmax_cross_entropy_with_logits(logits = exponent2, labels = Y)
err = tf.reduce_mean(entropy)

#optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.02)
optimizer = tf.train.MomentumOptimizer(learning_rate=0.02, momentum=0.9)
training_op = optimizer.minimize(err)
init = tf.global_variables_initializer()
testlet_est_lists = {}
for testlet_ind in range(testlet_num):
    t_n = testlet_lists[testlet_ind]['t_n']
    a_n = testlet_lists[testlet_ind]['a_n']
    stoch_ind = np.arange(taker_num)
    with tf.Session() as sess:
        sess.run(init)
        for iter in range(10):
            np.random.shuffle(stoch_ind)
            for epoch in range(taker_num):
            #if epoch % 20 == 0:
            #   print("Epoch", epoch, "MSE =", mse.eval())
                testee_ind = stoch_ind[epoch]
                prof_ep = get_prof_slice(testee_ind, testlet_ind, PROF)
                Y_ep = get_onehot_response(testee_ind, testlet_ind)
                sess.run(training_op, feed_dict={prof: prof_ep, Y: Y_ep})
                #if epoch % 1000 == 0:
```

```
        errv, av, tv = sess.run([err, a, t], feed_dict={prof: prof_ep, Y: Y_ep})
        print(errv, "", np.linalg.norm(a_n-av), np.linalg.norm(t_n-tv))
            #errv, av, tv = sess.run([err, a, t], feed_dict={prof: prof_ep, Y: Y_ep})
    testlet_est_lists[testlet_ind] = {"a_n": av, "t_n": tv}
    print(testlet_ind)

# In[67]:

estimate_item_params(Prof_est, 5)

# In[86]:

def estimate_item_params(Prof_est, iter_num=10):
    for col_idx in range(testlet_num+1):
        PROF[:, col_idx] = (Prof_est[:, col_idx] - np.mean(Prof_est[:, col_idx])) / np.std(Prof_est[:, col_idx])
    reset_graph()
    #prof = tf.constant(prof_n, dtype=tf.float32, name="prof")
    prof = tf.placeholder(dtype=tf.float32, shape=(2,1), name="prof")
    #Y = tf.constant(Y_n, dtype=tf.float32, name="Y" )
    Y = tf.placeholder(dtype=tf.float32, shape=(item_num, diff_level_num+1), name="Y")

    ap = tf.Variable( tf.random_uniform(minval=-2, maxval=2, shape=(item_num,2)), dtype=tf.float32)
    a = 0.65 + 0.8 * tf.sigmoid(ap)
    t = tf.Variable(tf.random_normal(shape=(item_num,3)), dtype=tf.float32)
    tmp_exponent = tf.matmul(a,prof) - t
    exponent1 = tf.concat([np.zeros((item_num,1)), tmp_exponent] , axis=1)
    exponent2 = tf.cumsum(exponent1, axis=1)
    probs = tf.nn.softmax(exponent2)

    entropy = tf.nn.softmax_cross_entropy_with_logits(logits = exponent2, labels = Y)
    err = tf.reduce_mean(entropy)

    #optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.02)
    optimizer = tf.train.MomentumOptimizer(learning_rate=0.02, momentum=0.9)
    training_op = optimizer.minimize(err)
    init = tf.global_variables_initializer()
    testlet_est_lists = {}
    for testlet_ind in range(testlet_num):
        t_n = testlet_lists[testlet_ind]['t_n']
        a_n = testlet_lists[testlet_ind]['a_n']
        stoch_ind = np.arange(taker_num)
        with tf.Session() as sess:
            sess.run(init)
            for iter in range(iter_num):
                np.random.shuffle(stoch_ind)
                for epoch in range(taker_num):
                #if epoch % 20 == 0:
                #   print("Epoch", epoch, "MSE =", mse.eval())
                    testee_ind = stoch_ind[epoch]
                    prof_ep = get_prof_slice(testee_ind, testlet_ind, PROF)
                    Y_ep = get_onehot_response(testee_ind, testlet_ind)
                    sess.run(training_op, feed_dict={prof: prof_ep, Y: Y_ep})
                    #if epoch % 1000 == 0:
```

```
            errv, av, tv = sess.run([err, a, t], feed_dict={prof: prof_ep, Y: Y_ep})
            print(errv, "", np.linalg.norm(a_n-av), np.linalg.norm(t_n-tv))
                #errv, av, tv = sess.run([err, a, t], feed_dict={prof: prof_ep, Y: Y_ep})
        testlet_est_lists[testlet_ind] = {"a_n": av, "t_n": tv}
    return testlet_est_lists

# # Define Individual Profeciency Estimator

# In[ ]:

an_3d = np.zeros((testlet_num,item_num, 2) )
tn_3d = np.zeros((testlet_num,item_num, diff_level_num) )
for testlet_idx in range(testlet_num):
    #an_3d[testlet_idx, :, :] = testlet_lists[testlet_idx]['a_n']
    an_3d[testlet_idx, :, :] = testlet_est_lists[testlet_idx]['a_n']
    tn_3d[testlet_idx, :, :] = testlet_est_lists[testlet_idx]['t_n']

Prof_est = np.zeros_like(PROF)

for taker_idx in range(taker_num):
    reset_graph()
    prof = tf.Variable(PROF[taker_idx, :].T.reshape((testlet_num+1,1)), dtype=tf.float32, name="prof")
    #prof = tf.Variable(tf.random_normal(shape=(testlet_num + 1,1)), dtype=tf.float32, name="prof")
    #All Responses 3-d tensor
    Y_all = tf.placeholder(dtype=tf.int32, shape=(testlet_num, item_num), name="Y_all")
    A_all = tf.constant(an_3d, dtype=tf.float32)
    T_all = tf.constant(tn_3d, dtype=tf.float32)
    ERR = tf.add_n ( [cal_one_testlet(testlet_idx, prof, A_all, T_all, Y_all) for testlet_idx in range(testlet_num)] )
    #optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.01)
    optimizer = tf.train.MomentumOptimizer(learning_rate=0.01,momentum=0.9)
    training_op = optimizer.minimize(ERR)
    init = tf.global_variables_initializer()

    with tf.Session() as sess:
        sess.run(init)
        for epoch in range(20):
            sess.run(training_op, feed_dict={Y_all: Y_comp[taker_idx].T})
        errv, profv = sess.run([ERR, prof], feed_dict={Y_all: Y_comp[taker_idx].T})
        Prof_est[taker_idx, :] = profv.ravel()
        print(taker_idx, np.linalg.norm(PROF[taker_idx, :] - PROF_HIDDEN_REAL[taker_idx, :]))
        print(taker_idx, np.linalg.norm(profv.reshape((1,-1)) - PROF_HIDDEN_REAL[taker_idx, :]))
        print("\n")
np.linalg.norm(Prof_est - PROF_HIDDEN_REAL)
np.linalg.norm(PROF - PROF_HIDDEN_REAL)

# In[87]:

def estimate_ind_params(testlet_est_lists, iter_num=20, learningrate=0.01):
    an_3d = np.zeros((testlet_num,item_num, 2) )
    tn_3d = np.zeros((testlet_num,item_num, diff_level_num) )
    for testlet_idx in range(testlet_num):
        #an_3d[testlet_idx, :, :] = testlet_lists[testlet_idx]['a_n']
        an_3d[testlet_idx, :, :] = testlet_est_lists[testlet_idx]['a_n']
```

```
    tn_3d[testlet_idx, :, :] = testlet_est_lists[testlet_idx]['t_n']

Prof_est = np.zeros_like(PROF)


for taker_idx in range(taker_num):
    reset_graph()
    prof = tf.Variable(PROF[taker_idx, :].T.reshape((testlet_num+1,1)), dtype=tf.float32, name="prof")
    #prof = tf.Variable(tf.random_normal(shape=(testlet_num + 1,1)), dtype=tf.float32, name="prof")
    #All Responses 3-d tensor
    Y_all = tf.placeholder(dtype=tf.int32, shape=(testlet_num, item_num), name="Y_all")
    A_all = tf.constant(an_3d, dtype=tf.float32)
    T_all = tf.constant(tn_3d, dtype=tf.float32)
    ERR = tf.add_n ( [cal_one_testlet(testlet_idx, prof, A_all, T_all, Y_all) for testlet_idx in range(testlet_num)] )
    #optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.01)
    optimizer = tf.train.MomentumOptimizer(learning_rate=learningrate, momentum=0.9)
    training_op = optimizer.minimize(ERR)
    init = tf.global_variables_initializer()

    with tf.Session() as sess:
        sess.run(init)
        for epoch in range(iter_num):
            sess.run(training_op, feed_dict={Y_all: Y_comp[taker_idx].T})
        errv, profv = sess.run([ERR, prof], feed_dict={Y_all: Y_comp[taker_idx].T})
        Prof_est[taker_idx, :] = profv.ravel()
        print(taker_idx, np.linalg.norm(PROF[taker_idx, :] - PROF_HIDDEN_REAL[taker_idx, :]))
        print(taker_idx, np.linalg.norm(profv.reshape((1,-1)) - PROF_HIDDEN_REAL[taker_idx, :]))
        print("\n")
return Prof_est
```