dkt

September 26, 2017

1 RNN modeling of behavior and performance

Yifat Amir, Allen Guo, Sam Lau

```
In [1]: import pandas as pd
        import numpy as np
        import json

        filename = 'skill_builder_data_corrected.csv'
        df = pd.read_csv(filename, encoding='ISO-8859-1', low_memory=False)
        df = df[(df['original'] == 1) & (df['attempt_count'] == 1) & ~(df['skill_name'].isnull()

In [2]: response_df = pd.read_csv('correct.tsv', sep='\t').drop('Unnamed: 0', axis=1)
        skill_df = pd.read_csv('skill.tsv', sep='\t').drop('Unnamed: 0', axis=1)
        assistment_df = pd.read_csv('assistment_id.tsv', sep='\t').drop('Unnamed: 0', axis=1)
        skill_dict = {}
        with open('skill_dict.json', 'r', encoding='utf-8') as f:
            loaded = json.load(f)
            for k, v in loaded.items():
                skill_dict[k] = int(v)

        skill_num = len(skill_dict) + 1 # including 0

        def one_hot(skill_matrix, vocab_size):
            '''
            params:
                skill_matrix: 2-D matrix (student, skills)
                vocal_size: size of the vocabulary
            returns:
                a ndarray with a shape like (student, sequence_len, vocab_size)
            '''
            seq_len = skill_matrix.shape[1]
            result = np.zeros((skill_matrix.shape[0], seq_len, vocab_size))
            for i in range(skill_matrix.shape[0]):
                result[i, np.arange(seq_len), skill_matrix[i]] = 1.
            return result

        def dkt_one_hot(skill_matrix, response_matrix, vocab_size):
```

```
    seq_len = skill_matrix.shape[1]
    skill_response_array = np.zeros((skill_matrix.shape[0], seq_len, 2 * vocab_size))
    for i in range(skill_matrix.shape[0]):
        skill_response_array[i, np.arange(seq_len), 2 * skill_matrix[i] + response_matri
    return skill_response_array

def preprocess(skill_df, response_df, skill_num):
    skill_matrix = skill_df.iloc[:, 1:].values
    response_array = response_df.iloc[:, 1:].values
    skill_array = one_hot(skill_matrix, skill_num)
    skill_response_array = dkt_one_hot(skill_matrix, response_array, skill_num)
    return skill_array, response_array, skill_response_array


skill_array, response_array, skill_response_array = preprocess(skill_df, response_df, sk
```

In [ ]:

```
import keras
from keras.layers import Input, Dense, LSTM, TimeDistributed, Lambda, multiply
from keras.models import Model
from keras.optimizers import RMSprop, Adam
from keras.preprocessing.sequence import pad_sequences
from keras import backend as K

def build_skill2skill_model(input_shape, lstm_dim=32, dropout=0.0):
    input = Input(shape=input_shape, name='input skills')
    lstm = LSTM(lstm_dim,
                return_sequences=True,
                dropout=dropout,
                name='lstm layer')(input)
    output = TimeDistributed(Dense(input_shape[-1], activation='softmax'), name='probabi
    model = Model(inputs=[input], outputs=[output])
    adam = Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-07, decay=0.0)
    model.compile(optimizer=adam, loss='categorical_crossentropy', metrics=['accuracy'])
    model.summary()
    return model

def reduce_dim(x):
    x = K.max(x, axis=-1, keepdims=True)
    return x

def build_dkt_model(input_shape, lstm_dim=32, dropout=0.0):
    input_skills = Input(shape=input_shape, name='input skills')
    lstm = LSTM(lstm_dim,
                return_sequences=True,
                dropout=dropout,
                name='lstm layer')(input_skills)
    dense = TimeDistributed(Dense(int(input_shape[-1]/2), activation='sigmoid'), name='p
```

```
    skill_next = Input(shape=(input_shape[0], int(input_shape[1]/2)), name='next_skill_t
    merged = multiply([dense, skill_next], name='multiply')
    reduced = Lambda(reduce_dim, output_shape=(input_shape[0], 1), name='reduce dim')(me

    model = Model(inputs=[input_skills, skill_next], outputs=[reduced])
    adam = Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-07, decay=0.0)
    model.compile(optimizer=adam, loss='binary_crossentropy', metrics=['accuracy'])
    model.summary()
    return model

print('skill2skill')
skill2skill_model = build_skill2skill_model((99, skill_num), lstm_dim=64)

print('dkt')
dkt_model = build_dkt_model((99, 2 * skill_num), lstm_dim=64)
```

```
In [ ]: %%time

        # train skill2skill
        skill2skill_model.fit(skill_array[:, 0:-1],
                              skill_array[:, 1:],
                              epochs=20,
                              batch_size=32,
                              shuffle=True,
                              validation_split=0.2)
```

```
In [ ]: %%time

        dkt_model.fit([skill_response_array[:, 0:-1], skill_array[:, 1:]],
                      response_array[:, 1:, np.newaxis],
                      epochs=20,
                      batch_size=32,
                      shuffle=True,
                      validation_split=0.2)
```

1.1 Question 1

What were the 5 most common and 5 least common skills in this dataset? What percentage of responses are associated with the most common skill?

```
In [4]: skillname_df = pd.DataFrame(list(skill_dict.items()), columns=['Name', 'ID']).set_index(
        skillname_df.head()
```

```
Out[4]:                                    Name
        ID
        39                           Estimation
        90              Greatest Common Factor
        88  Equation Solving More Than Two Steps
```

```
74                          Translations
43           Multiplication Whole Numbers
```

```
In [5]: skill_counts = (skill_df
         .iloc[:, 1:]
         .unstack()
         .value_counts()
         .rename('count')
         .to_frame()
         .join(skillname_df)
        )
```

```
In [6]: print('Top 5 skills:')
        skill_counts.head()
```

```
Top 5 skills:
```

```
Out[6]:     count                                    Name
        7    4579                                   Table
        30   4379  Conversion of Fraction Decimals Percents
        8    3466                            Venn Diagram
        2    3404                            Circle Graph
        33   2833                       Ordering Fractions
```

```
In [7]: print('Bottom 5 skills:')
        skill_counts.tail()
```

```
Bottom 5 skills:
```

```
Out[7]:      count                      Name
        83       3  Volume Rectangular Prism
        84       2             Volume Sphere
        109      2      Solving Inequalities
        82       2           Volume Cylinder
        80       1     Surface Area Cylinder
```

```
In [8]: print('Proportion of responses for most common skill:')
        skill_counts.iloc[0, 0] / skill_counts['count'].sum()
```

```
Proportion of responses for most common skill:
```

```
Out[8]: 0.078407534246575344
```

1.2 Question 2

Train the sequence prediction model using a randomly selected 70% (training set) of students' data and predict on the remaining 30% (test set). What was the overall accuracy of skill prediction in the test set? What were the top 5 hardest and easiest to predict skills? Describe the metric you chose to represent hard/easy prediction.

In [9]: from sklearn.model_selection import train_test_split

```
X_train, X_test, y_train, y_test = train_test_split(skill_array[:, 0:-1], skill_array[:,
X_train.shape, X_test.shape
```

Out[9]: ((408, 99, 111), (176, 99, 111))

In [ ]:

```
%%time

# train skill2skill
skill2skill_model.fit(X_train,
                      y_train,
                      epochs=20,
                      batch_size=32,
                      shuffle=True,
                      validation_split=0.2)
```

In [20]:

```
test_predictions = skill2skill_model.predict(X_test)
test_predictions.shape
```

Out[20]: (176, 99, 111)

In [74]:

```
def s2s_acc(true, predictions):
    assert true.shape == predictions.shape
    return (np.count_nonzero(true.argmax(axis=2) == predictions.argmax(axis=2))
            / (true.shape[0] * true.shape[1]))

print("Overall accuracy:")
s2s_acc(y_test, test_predictions)
```

Overall accuracy:

Out[74]: 0.7007575757575758

In [67]:

```
from sklearn.metrics import classification

scores = classification.recall_score(y_test.argmax(axis=2).flatten(),
                                     test_predictions.argmax(axis=2).flatten(),
                                     average=None)
score_df = pd.DataFrame({'scores': scores}, index=np.unique(y_test.argmax(axis=2)))

print('Easiest skills to predict (highest recall):')
(score_df
 .sort_values('scores', ascending=False)
 .join(skillname_df)
 .head()
)
```

```
Easiest skills to predict (highest recall):
```

Out[67]:

	scores	Name
1	0.984906	Box and Whisker
46	0.957399	Square Root
68	0.922145	Addition and Subtraction Integers
7	0.915704	Table
8	0.910093	Venn Diagram

In [66]:
```
print('Hardest skills to predict (lowest recall):')
print('Note that there were a lot of 0s so these are just 5 of the 0s')
(score_df
 .sort_values('scores', ascending=False)
 .join(skillname_df)
 .tail()
)
```

```
Hardest skills to predict (lowest recall):
Note that there were a lot of 0s so these are just 5 of the 0s
```

Out[66]:

	scores	Name
20	0.0	Angles on Parallel Lines Cut by a Transversal
56	0.0	Pattern Finding
59	0.0	Algebraic Solving
60	0.0	Choose an Equation from Given Information
110	0.0	Solving Systems of Linear Equations by Graphing

1.3 Question 3

Modify parameters of the network to increase accuracy (e.g. number of hidden nodes, optimizer, number of RNN layers, number of epochs, creating a validation set and stopping training when the validation set accuracy decreases). What were your accuracy results with respect to the hyper parameters you tuned?

In [88]:
```
print('skill2skill with 128 hidden nodes')
s2s_128_model = build_skill2skill_model((99, skill_num), lstm_dim=128)
```

```
skill2skill with dropout of 0.1
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input skills (InputLayer)    (None, 99, 111)           0
_________________________________________________________________
lstm layer (LSTM)            (None, 99, 128)           122880
_________________________________________________________________
probability (TimeDistributed (None, 99, 111)           14319
=================================================================
```

```
Total params: 137,199
Trainable params: 137,199
Non-trainable params: 0
_________________________________________________________________
```

```
In [ ]: %%time

        # train skill2skill
        s2s_128_model.fit(X_train,
                          y_train,
                          epochs=20,
                          batch_size=32,
                          shuffle=True,
                          validation_split=0.2)
```

```
In [90]: print('Attempts to set dropout=0.1 or lower number of hidden nodes decreased accuracy.'

         print('Accuracy with 128 hidden nodes:')
         s2s_acc(y_test, s2s_128_model.predict(X_test))
```

```
Attempts to set dropout=0.1 or lower number of hidden nodes decreased accuracy.
Accuracy with 128 hidden nodes:
```

```
Out[90]: 0.7765725436179982
```

1.4 Question 4

Train a performance prediction model (DKT) using the same 70/30% split and report the accuracy and AUC of prediction on the 30%

```
In [100]: X1_train, X1_test, X2_train, X2_test, y_train, y_test = train_test_split(
                skill_response_array[:, 0:-1],
                skill_array[:, 1:],
                response_array[:, 1:, np.newaxis],
                test_size=0.3
          )
          X1_train.shape, X1_test.shape
```

```
Out[100]: ((408, 99, 222), (176, 99, 222))
```

```
In [ ]: %%time

        dkt_model.fit([X1_train, X2_train],
                      y_train,
                      epochs=15,
                      batch_size=32,
                      shuffle=True,
                      validation_split=0.2)
```

```
In [119]: dkt_model.evaluate([X1_test, X2_test], y_test)
```

```
176/176 [==============================] - 0s
```

```
Out[119]: [0.35878889127211139, 0.86346419291062793]
```

```
In [121]: from sklearn.metrics import accuracy_score

          dkt_predictions = dkt_model.predict([X1_test, X2_test])
          print('DKT Prediction Accuracy:')
          accuracy_score(np.round(dkt_predictions.flatten()), y_test.flatten())
```

```
DKT Prediction Accuracy:
```

```
Out[121]: 0.86346418732782371
```

```
In [140]: from sklearn.metrics import roc_auc_score

          print('DKT AUC score:')
          roc_auc_score(y_test.flatten(), dkt_predictions.flatten())
```

```
DKT AUC score:
```

```
Out[140]: 0.74280793994244021
```

1.5 Question 5

Tune the hyper parameters of this model to improve accuracy and report your improvement with respect to the tuned parameters. Which lead to the most significant improvement?

```
In [124]: print('dkt with 128 hidden nodes')
          dkt_128_model = build_dkt_model((99, 2 * skill_num), lstm_dim=128)
```

```
dkt with 128 hidden nodes
____________________________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
input skills (InputLayer)        (None, 99, 222)       0
____________________________________________________________________________________________________
lstm layer (LSTM)                (None, 99, 128)       179712      input skills[0][0]
____________________________________________________________________________________________________
probability for each (TimeDistri (None, 99, 111)       14319       lstm layer[0][0]
____________________________________________________________________________________________________
next_skill_tested (InputLayer)   (None, 99, 111)       0
____________________________________________________________________________________________________
multiply (Multiply)              (None, 99, 111)       0           probability for each[0][0]
                                                                   next_skill_tested[0][0]
```

```
__________________________________________________________________________________________________
reduce dim (Lambda)              (None, 99, 1)        0           multiply[0][0]
==================================================================================================
Total params: 194,031
Trainable params: 194,031
Non-trainable params: 0
__________________________________________________________________________________________________
```

In [ ]:
```
%%time

dkt_128_model.fit([X1_train, X2_train],
                  y_train,
                  epochs=20,
                  batch_size=32,
                  shuffle=True,
                  validation_split=0.2)
```

In [126]:
```
dkt_128_predictions = dkt_128_model.predict([X1_test, X2_test])
print('DKT Prediction Accuracy with 128 hidden nodes:')
accuracy_score(np.round(dkt_128_predictions.flatten()), y_test.flatten())
```

```
DKT Prediction Accuracy with 128 hidden nodes:
```

Out[126]:
```
0.86116850321395777
```

That didn't really work; attempt 2:

In [128]:
```
print('dkt with 128 hidden nodes and dropout of 0.01')
dkt_128_model = build_dkt_model((99, 2 * skill_num), lstm_dim=128, dropout=0.01)
```

```
dkt with 128 hidden nodes and dropout of 0.01
__________________________________________________________________________________________________
Layer (type)                     Output Shape         Param #     Connected to
==================================================================================================
input skills (InputLayer)        (None, 99, 222)      0
__________________________________________________________________________________________________
lstm layer (LSTM)                (None, 99, 128)      179712      input skills[0][0]
__________________________________________________________________________________________________
probability for each (TimeDistri (None, 99, 111)      14319       lstm layer[0][0]
__________________________________________________________________________________________________
next_skill_tested (InputLayer)   (None, 99, 111)      0
__________________________________________________________________________________________________
multiply (Multiply)              (None, 99, 111)      0           probability for each[0][0]
                                                                  next_skill_tested[0][0]
__________________________________________________________________________________________________
reduce dim (Lambda)              (None, 99, 1)        0           multiply[0][0]
==================================================================================================
```

```
Total params: 194,031
Trainable params: 194,031
Non-trainable params: 0
________________________________________________________________________________________________
```

```
In [ ]: %%time

        dkt_128_model.fit([X1_train, X2_train],
                          y_train,
                          epochs=20,
                          batch_size=32,
                          shuffle=True,
                          validation_split=0.2)
```

```
In [130]: dkt_128_predictions = dkt_128_model.predict([X1_test, X2_test])
          print('DKT Prediction Accuracy with 128 hidden nodes and dropout of 0.01:')
          accuracy_score(np.round(dkt_128_predictions.flatten()), y_test.flatten())
```

```
DKT Prediction Accuracy with 128 hidden nodes and dropout of 0.01:
```

```
Out[130]: 0.8617424242424242
```

```
In [139]: print('DKT AUC score:')
          roc_auc_score(y_test.flatten(), dkt_128_predictions.flatten())
```

```
DKT AUC score:
```

```
Out[139]: 0.75305468559654443
```

That was about the same as well :(.

```
In [132]: print('dkt with 256 hidden nodes and dropout of 0.01')
          dkt_256_model = build_dkt_model((99, 2 * skill_num), lstm_dim=256, dropout=0.01)
```

```
dkt with 256 hidden nodes and dropout of 0.01
________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
================================================================================================
input skills (InputLayer)       (None, 99, 222)      0
________________________________________________________________________________________________
lstm layer (LSTM)               (None, 99, 256)      490496      input skills[0][0]
________________________________________________________________________________________________
probability for each (TimeDistri (None, 99, 111)     28527       lstm layer[0][0]
________________________________________________________________________________________________
next_skill_tested (InputLayer)  (None, 99, 111)      0
________________________________________________________________________________________________
```

```
multiply (Multiply)               (None, 99, 111)      0            probability for each[0][0]
                                                                    next_skill_tested[0][0]
------------------------------------------------------------------------------------------------
reduce dim (Lambda)               (None, 99, 1)        0            multiply[0][0]
================================================================================================
Total params: 519,023
Trainable params: 519,023
Non-trainable params: 0
------------------------------------------------------------------------------------------------
```

In [ ]:
```
%%time

dkt_256_model.fit([X1_train, X2_train],
                  y_train,
                  epochs=20,
                  batch_size=32,
                  shuffle=True,
                  validation_split=0.2)
```

In [134]:
```
dkt_256_predictions = dkt_256_model.predict([X1_test, X2_test])
print('DKT Prediction Accuracy with 256 hidden nodes and dropout of 0.01:')
accuracy_score(np.round(dkt_256_predictions.flatten()), y_test.flatten())
```

```
DKT Prediction Accuracy with 256 hidden nodes and dropout of 0.01:
```

Out[134]: 0.8621441689623508

In [137]:
```
print('DKT AUC score:')
roc_auc_score(y_test.flatten(), dkt_256_predictions.flatten())
```

```
DKT AUC score:
```

Out[137]: 0.759231354237O9O32

Looks like we were able to get the AUC to increase by 1.7% compared to the original model using 256 hidden nodes and a dropout of 0.1.