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 [38]: len(skill_dict)

Out[38]: 110
```

```
In [37]: response_array.shape

Out[37]: (584, 100)
```

```
In [26]: skill_array.shape

Out[26]: (584, 100, 111)
```

```
In [32]: skill_response_array.shape

Out[32]: (584, 100, 222)
```

```
In [3]: 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)
```

```
Using Theano backend.
```

```
skill2skill
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input skills (InputLayer)    (None, 99, 111)           0
_________________________________________________________________
lstm layer (LSTM)            (None, 99, 64)            45056
_________________________________________________________________
probability (TimeDistributed (None, 99, 111)           7215
=================================================================
Total params: 52,271
Trainable params: 52,271
Non-trainable params: 0
```

```
_________________________________________________________________
dkt
________________________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
================================================================================================
input skills (InputLayer)        (None, 99, 222)       0
________________________________________________________________________________________________
lstm layer (LSTM)                (None, 99, 64)        73472       input skills[0][0]
________________________________________________________________________________________________
probability for each (TimeDistri (None, 99, 111)       7215        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: 80,687
Trainable params: 80,687
Non-trainable params: 0
________________________________________________________________________________________________
```

```
In [41]: %%time

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

```
Train on 467 samples, validate on 117 samples
Epoch 1/20
467/467 [==============================] - 1s - loss: 4.6170 - acc: 0.1950 - val_loss: 4.5729 -
Epoch 2/20
467/467 [==============================] - 1s - loss: 4.2278 - acc: 0.1550 - val_loss: 4.4697 -
Epoch 3/20
467/467 [==============================] - 2s - loss: 3.6677 - acc: 0.1333 - val_loss: 4.1492 -
Epoch 4/20
467/467 [==============================] - 1s - loss: 3.1779 - acc: 0.2505 - val_loss: 3.8165 -
Epoch 5/20
467/467 [==============================] - 1s - loss: 2.7544 - acc: 0.3735 - val_loss: 3.5075 -
Epoch 6/20
467/467 [==============================] - 1s - loss: 2.4293 - acc: 0.4355 - val_loss: 3.1725 -
Epoch 7/20
```

```
467/467 [==============================] - 1s - loss: 2.1904 - acc: 0.4799 - val_loss: 2.9253 -
Epoch 8/20
467/467 [==============================] - 1s - loss: 1.9651 - acc: 0.5337 - val_loss: 2.6795 -
Epoch 9/20
467/467 [==============================] - 1s - loss: 1.7745 - acc: 0.5865 - val_loss: 2.4644 -
Epoch 10/20
467/467 [==============================] - 1s - loss: 1.6159 - acc: 0.6326 - val_loss: 2.2535 -
Epoch 11/20
467/467 [==============================] - 1s - loss: 1.5129 - acc: 0.6496 - val_loss: 2.1068 -
Epoch 12/20
467/467 [==============================] - 1s - loss: 1.4089 - acc: 0.6717 - val_loss: 1.9672 -
Epoch 13/20
467/467 [==============================] - 1s - loss: 1.2970 - acc: 0.6983 - val_loss: 1.8139 -
Epoch 14/20
467/467 [==============================] - 1s - loss: 1.2062 - acc: 0.7132 - val_loss: 1.7264 -
Epoch 15/20
467/467 [==============================] - 1s - loss: 1.1404 - acc: 0.7263 - val_loss: 1.6623 -
Epoch 16/20
467/467 [==============================] - 1s - loss: 1.0881 - acc: 0.7363 - val_loss: 1.5911 -
Epoch 17/20
467/467 [==============================] - 1s - loss: 1.0495 - acc: 0.7428 - val_loss: 1.5140 -
Epoch 18/20
467/467 [==============================] - 1s - loss: 0.9877 - acc: 0.7582 - val_loss: 1.4528 -
Epoch 19/20
467/467 [==============================] - 1s - loss: 0.9410 - acc: 0.7684 - val_loss: 1.4058 -
Epoch 20/20
467/467 [==============================] - 1s - loss: 0.9027 - acc: 0.7805 - val_loss: 1.3694 -
CPU times: user 1min 15s, sys: 5.39 s, total: 1min 20s
Wall time: 2min 24s
```

```
Out[41]: <keras.callbacks.History at 0x129ea1d68>
```

```
In [42]: %%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)
```

```
Train on 467 samples, validate on 117 samples
Epoch 1/20
467/467 [==============================] - 2s - loss: 0.6735 - acc: 0.7242 - val_loss: 0.6643 -
Epoch 2/20
467/467 [==============================] - 3s - loss: 0.6072 - acc: 0.8360 - val_loss: 0.6066 -
Epoch 3/20
```

```
467/467 [==============================] - 3s - loss: 0.4875 - acc: 0.8307 - val_loss: 0.5034 -
Epoch 4/20
467/467 [==============================] - 2s - loss: 0.4167 - acc: 0.8489 - val_loss: 0.4606 -
Epoch 5/20
467/467 [==============================] - 3s - loss: 0.3928 - acc: 0.8507 - val_loss: 0.4536 -
Epoch 6/20
467/467 [==============================] - 4s - loss: 0.3828 - acc: 0.8552 - val_loss: 0.4447 -
Epoch 7/20
467/467 [==============================] - 3s - loss: 0.3782 - acc: 0.8559 - val_loss: 0.4394 -
Epoch 8/20
467/467 [==============================] - 3s - loss: 0.3772 - acc: 0.8555 - val_loss: 0.4479 -
Epoch 9/20
467/467 [==============================] - 3s - loss: 0.3725 - acc: 0.8574 - val_loss: 0.4288 -
Epoch 10/20
467/467 [==============================] - 2s - loss: 0.3686 - acc: 0.8571 - val_loss: 0.4392 -
Epoch 11/20
467/467 [==============================] - 2s - loss: 0.3661 - acc: 0.8568 - val_loss: 0.4359 -
Epoch 12/20
467/467 [==============================] - 2s - loss: 0.3628 - acc: 0.8571 - val_loss: 0.4344 -
Epoch 13/20
467/467 [==============================] - 2s - loss: 0.3598 - acc: 0.8571 - val_loss: 0.4293 -
Epoch 14/20
467/467 [==============================] - 2s - loss: 0.3571 - acc: 0.8587 - val_loss: 0.4287 -
Epoch 15/20
467/467 [==============================] - 1s - loss: 0.3557 - acc: 0.8585 - val_loss: 0.4220 -
Epoch 16/20
467/467 [==============================] - 2s - loss: 0.3541 - acc: 0.8600 - val_loss: 0.4316 -
Epoch 17/20
467/467 [==============================] - 2s - loss: 0.3525 - acc: 0.8601 - val_loss: 0.4249 -
Epoch 18/20
467/467 [==============================] - 2s - loss: 0.3494 - acc: 0.8615 - val_loss: 0.4194 -
Epoch 19/20
467/467 [==============================] - 2s - loss: 0.3480 - acc: 0.8619 - val_loss: 0.4243 -
Epoch 20/20
467/467 [==============================] - 2s - loss: 0.3488 - acc: 0.8622 - val_loss: 0.4199 -
CPU times: user 1min 22s, sys: 7.04 s, total: 1min 29s
Wall time: 1min 31s
```

```
Out[42]: <keras.callbacks.History at 0x12c76ae48>
```

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 [12]: %%time

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

```
Train on 326 samples, validate on 82 samples
Epoch 1/20
326/326 [==============================] - 1s - loss: 4.6533 - acc: 0.1423 - val_loss: 4.5828 -
Epoch 2/20
326/326 [==============================] - 1s - loss: 4.5181 - acc: 0.4430 - val_loss: 4.3526 -
Epoch 3/20
326/326 [==============================] - 1s - loss: 4.1315 - acc: 0.2106 - val_loss: 3.8672 -
Epoch 4/20
326/326 [==============================] - 1s - loss: 3.7409 - acc: 0.1613 - val_loss: 3.5118 -
Epoch 5/20
326/326 [==============================] - 1s - loss: 3.4378 - acc: 0.2185 - val_loss: 3.2544 -
Epoch 6/20
326/326 [==============================] - 1s - loss: 3.1585 - acc: 0.3196 - val_loss: 2.9795 -
Epoch 7/20
326/326 [==============================] - 1s - loss: 2.8959 - acc: 0.3604 - val_loss: 2.7486 -
Epoch 8/20
326/326 [==============================] - 1s - loss: 2.6670 - acc: 0.3920 - val_loss: 2.5466 -
Epoch 9/20
326/326 [==============================] - 1s - loss: 2.4612 - acc: 0.4405 - val_loss: 2.3498 -
Epoch 10/20
326/326 [==============================] - 1s - loss: 2.3162 - acc: 0.4738 - val_loss: 2.2786 -
Epoch 11/20
326/326 [==============================] - 1s - loss: 2.1886 - acc: 0.5107 - val_loss: 2.0904 -
Epoch 12/20
326/326 [==============================] - 1s - loss: 2.0268 - acc: 0.5426 - val_loss: 1.9686 -
```

```
Epoch 13/20
326/326 [==============================] - 1s - loss: 1.8983 - acc: 0.5743 - val_loss: 1.8420 -
Epoch 14/20
326/326 [==============================] - 1s - loss: 1.7876 - acc: 0.5972 - val_loss: 1.7449 -
Epoch 15/20
326/326 [==============================] - 1s - loss: 1.6878 - acc: 0.6126 - val_loss: 1.6440 -
Epoch 16/20
326/326 [==============================] - 1s - loss: 1.5961 - acc: 0.6331 - val_loss: 1.5593 -
Epoch 17/20
326/326 [==============================] - 1s - loss: 1.5161 - acc: 0.6513 - val_loss: 1.4921 -
Epoch 18/20
326/326 [==============================] - 1s - loss: 1.4482 - acc: 0.6656 - val_loss: 1.4310 -
Epoch 19/20
326/326 [==============================] - 1s - loss: 1.3890 - acc: 0.6787 - val_loss: 1.3727 -
Epoch 20/20
326/326 [==============================] - 1s - loss: 1.3295 - acc: 0.6963 - val_loss: 1.3053 -
CPU times: user 55.4 s, sys: 3.58 s, total: 58.9 s
Wall time: 38 s
```

```
Out[12]: <keras.callbacks.History at 0x1322a4240>
```

```
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 [89]: %%time

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

```
Train on 326 samples, validate on 82 samples
Epoch 1/20
326/326 [==============================] - 2s - loss: 4.6279 - acc: 0.2381 - val_loss: 4.4789 -
Epoch 2/20
326/326 [==============================] - 2s - loss: 4.2272 - acc: 0.3127 - val_loss: 3.7845 -
Epoch 3/20
326/326 [==============================] - 2s - loss: 3.6545 - acc: 0.2389 - val_loss: 3.2747 -
Epoch 4/20
326/326 [==============================] - 2s - loss: 3.1287 - acc: 0.2737 - val_loss: 2.8435 -
Epoch 5/20
326/326 [==============================] - 2s - loss: 2.6708 - acc: 0.3595 - val_loss: 2.4688 -
Epoch 6/20
326/326 [==============================] - 2s - loss: 2.3373 - acc: 0.4474 - val_loss: 2.2300 -
Epoch 7/20
326/326 [==============================] - 2s - loss: 2.1123 - acc: 0.4891 - val_loss: 1.9797 -
Epoch 8/20
326/326 [==============================] - 2s - loss: 1.8995 - acc: 0.5505 - val_loss: 1.8180 -
Epoch 9/20
326/326 [==============================] - 4s - loss: 1.7225 - acc: 0.5860 - val_loss: 1.7010 -
Epoch 10/20
326/326 [==============================] - 2s - loss: 1.5751 - acc: 0.6170 - val_loss: 1.5052 -
Epoch 11/20
326/326 [==============================] - 2s - loss: 1.4406 - acc: 0.6469 - val_loss: 1.4081 -
Epoch 12/20
326/326 [==============================] - 2s - loss: 1.3413 - acc: 0.6756 - val_loss: 1.3315 -
Epoch 13/20
326/326 [==============================] - 2s - loss: 1.2550 - acc: 0.7009 - val_loss: 1.2336 -
```

```
Epoch 14/20
326/326 [==============================] - 2s - loss: 1.1747 - acc: 0.7134 - val_loss: 1.1567 -
Epoch 15/20
326/326 [==============================] - 2s - loss: 1.1067 - acc: 0.7376 - val_loss: 1.0925 -
Epoch 16/20
326/326 [==============================] - 2s - loss: 1.0463 - acc: 0.7471 - val_loss: 1.0376 -
Epoch 17/20
326/326 [==============================] - 2s - loss: 1.0091 - acc: 0.7534 - val_loss: 0.9971 -
Epoch 18/20
326/326 [==============================] - 2s - loss: 0.9565 - acc: 0.7662 - val_loss: 0.9530 -
Epoch 19/20
326/326 [==============================] - 2s - loss: 0.9159 - acc: 0.7746 - val_loss: 0.9171 -
Epoch 20/20
326/326 [==============================] - 2s - loss: 0.8811 - acc: 0.7826 - val_loss: 0.8956 -
CPU times: user 1min 31s, sys: 6.09 s, total: 1min 37s
Wall time: 59.8 s
```

```
Out[89]: <keras.callbacks.History at 0x140e574a8>
```

```
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 [101]: %%time

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

```
Train on 326 samples, validate on 82 samples
Epoch 1/20
326/326 [==============================] - 1s - loss: 0.6832 - acc: 0.6557 - val_loss: 0.6666 -
Epoch 2/20
326/326 [==============================] - 1s - loss: 0.6502 - acc: 0.8279 - val_loss: 0.6302 -
Epoch 3/20
326/326 [==============================] - 1s - loss: 0.5897 - acc: 0.8363 - val_loss: 0.5278 -
Epoch 4/20
326/326 [==============================] - 1s - loss: 0.4948 - acc: 0.8177 - val_loss: 0.4607 -
Epoch 5/20
326/326 [==============================] - 1s - loss: 0.4367 - acc: 0.8409 - val_loss: 0.4309 -
Epoch 6/20
326/326 [==============================] - 1s - loss: 0.4118 - acc: 0.8418 - val_loss: 0.4242 -
Epoch 7/20
326/326 [==============================] - 1s - loss: 0.4024 - acc: 0.8450 - val_loss: 0.4182 -
Epoch 8/20
326/326 [==============================] - 1s - loss: 0.3949 - acc: 0.8475 - val_loss: 0.4171 -
Epoch 9/20
326/326 [==============================] - 1s - loss: 0.3932 - acc: 0.8507 - val_loss: 0.4122 -
Epoch 10/20
326/326 [==============================] - 1s - loss: 0.3854 - acc: 0.8511 - val_loss: 0.4100 -
Epoch 11/20
326/326 [==============================] - 1s - loss: 0.3806 - acc: 0.8513 - val_loss: 0.4054 -
Epoch 12/20
326/326 [==============================] - 1s - loss: 0.3781 - acc: 0.8528 - val_loss: 0.4016 -
Epoch 13/20
326/326 [==============================] - 1s - loss: 0.3753 - acc: 0.8529 - val_loss: 0.3979 -
Epoch 14/20
326/326 [==============================] - 1s - loss: 0.3709 - acc: 0.8545 - val_loss: 0.3997 -
Epoch 15/20
326/326 [==============================] - 1s - loss: 0.3667 - acc: 0.8557 - val_loss: 0.3940 -
Epoch 16/20
326/326 [==============================] - 1s - loss: 0.3635 - acc: 0.8568 - val_loss: 0.3928 -
Epoch 17/20
326/326 [==============================] - 1s - loss: 0.3597 - acc: 0.8587 - val_loss: 0.3916 -
Epoch 18/20
326/326 [==============================] - 1s - loss: 0.3583 - acc: 0.8590 - val_loss: 0.3929 -
Epoch 19/20
326/326 [==============================] - 1s - loss: 0.3563 - acc: 0.8590 - val_loss: 0.3898 -
Epoch 20/20
326/326 [==============================] - 1s - loss: 0.3568 - acc: 0.8599 - val_loss: 0.3877 -
CPU times: user 52.9 s, sys: 3.69 s, total: 56.6 s
```

```
Wall time: 34.9 s
```

```
Out[101]: <keras.callbacks.History at 0x16af02390>
```

```
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 [125]: %%time

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

```
Train on 326 samples, validate on 82 samples
Epoch 1/20
326/326 [==============================] - 2s - loss: 0.6695 - acc: 0.7454 - val_loss: 0.6366 -
Epoch 2/20
326/326 [==============================] - 2s - loss: 0.5696 - acc: 0.8423 - val_loss: 0.4805 -
Epoch 3/20
326/326 [==============================] - 2s - loss: 0.4444 - acc: 0.8397 - val_loss: 0.4346 -
Epoch 4/20
326/326 [==============================] - 2s - loss: 0.4114 - acc: 0.8429 - val_loss: 0.4258 -
Epoch 5/20
326/326 [==============================] - 2s - loss: 0.4038 - acc: 0.8452 - val_loss: 0.4159 -
Epoch 6/20
326/326 [==============================] - 2s - loss: 0.3959 - acc: 0.8491 - val_loss: 0.4138 -
Epoch 7/20
326/326 [==============================] - 2s - loss: 0.3937 - acc: 0.8488 - val_loss: 0.4104 -
Epoch 8/20
326/326 [==============================] - 2s - loss: 0.3882 - acc: 0.8513 - val_loss: 0.4072 -
Epoch 9/20
326/326 [==============================] - 2s - loss: 0.3842 - acc: 0.8500 - val_loss: 0.4052 -
Epoch 10/20
326/326 [==============================] - 2s - loss: 0.3948 - acc: 0.8472 - val_loss: 0.4102 -
Epoch 11/20
326/326 [==============================] - 2s - loss: 0.3878 - acc: 0.8480 - val_loss: 0.4080 -
Epoch 12/20
326/326 [==============================] - 2s - loss: 0.3861 - acc: 0.8524 - val_loss: 0.4085 -
```

```
Epoch 13/20
326/326 [==============================] - 2s - loss: 0.3791 - acc: 0.8534 - val_loss: 0.4025 -
Epoch 14/20
326/326 [==============================] - 2s - loss: 0.3763 - acc: 0.8520 - val_loss: 0.3985 -
Epoch 15/20
326/326 [==============================] - 2s - loss: 0.3708 - acc: 0.8558 - val_loss: 0.3971 -
Epoch 16/20
326/326 [==============================] - 2s - loss: 0.3703 - acc: 0.8545 - val_loss: 0.3967 -
Epoch 17/20
326/326 [==============================] - 3s - loss: 0.3650 - acc: 0.8575 - val_loss: 0.3975 -
Epoch 18/20
326/326 [==============================] - 2s - loss: 0.3670 - acc: 0.8578 - val_loss: 0.3931 -
Epoch 19/20
326/326 [==============================] - 2s - loss: 0.3612 - acc: 0.8576 - val_loss: 0.3919 -
Epoch 20/20
326/326 [==============================] - 2s - loss: 0.3598 - acc: 0.8570 - val_loss: 0.3868 -
CPU times: user 1min 26s, sys: 5.34 s, total: 1min 31s
Wall time: 53.6 s
```

```
Out[125]: <keras.callbacks.History at 0x144d1b7f0>
```

```
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 [129]: %%time

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

```
Train on 326 samples, validate on 82 samples
Epoch 1/20
326/326 [==============================] - 2s - loss: 0.6687 - acc: 0.7452 - val_loss: 0.6373 -
Epoch 2/20
326/326 [==============================] - 2s - loss: 0.5703 - acc: 0.8359 - val_loss: 0.5011 -
Epoch 3/20
326/326 [==============================] - 2s - loss: 0.4598 - acc: 0.8313 - val_loss: 0.4398 -
Epoch 4/20
326/326 [==============================] - 2s - loss: 0.4196 - acc: 0.8409 - val_loss: 0.4253 -
Epoch 5/20
326/326 [==============================] - 2s - loss: 0.4059 - acc: 0.8445 - val_loss: 0.4191 -
Epoch 6/20
326/326 [==============================] - 2s - loss: 0.3997 - acc: 0.8461 - val_loss: 0.4172 -
Epoch 7/20
326/326 [==============================] - 2s - loss: 0.3960 - acc: 0.8486 - val_loss: 0.4121 -
Epoch 8/20
326/326 [==============================] - 2s - loss: 0.3903 - acc: 0.8488 - val_loss: 0.4089 -
Epoch 9/20
326/326 [==============================] - 2s - loss: 0.3865 - acc: 0.8498 - val_loss: 0.4077 -
Epoch 10/20
326/326 [==============================] - 2s - loss: 0.3842 - acc: 0.8519 - val_loss: 0.4054 -
Epoch 11/20
326/326 [==============================] - 2s - loss: 0.3800 - acc: 0.8533 - val_loss: 0.4020 -
Epoch 12/20
326/326 [==============================] - 2s - loss: 0.3765 - acc: 0.8519 - val_loss: 0.4003 -
Epoch 13/20
326/326 [==============================] - 2s - loss: 0.3731 - acc: 0.8549 - val_loss: 0.3970 -
Epoch 14/20
326/326 [==============================] - 2s - loss: 0.3724 - acc: 0.8534 - val_loss: 0.4017 -
```

```
Epoch 15/20
326/326 [==============================] - 2s - loss: 0.3773 - acc: 0.8450 - val_loss: 0.3988 -
Epoch 16/20
326/326 [==============================] - 3s - loss: 0.3728 - acc: 0.8470 - val_loss: 0.3920 -
Epoch 17/20
326/326 [==============================] - 3s - loss: 0.3681 - acc: 0.8491 - val_loss: 0.3939 -
Epoch 18/20
326/326 [==============================] - 2s - loss: 0.3628 - acc: 0.8535 - val_loss: 0.3935 -
Epoch 19/20
326/326 [==============================] - 2s - loss: 0.3593 - acc: 0.8573 - val_loss: 0.3844 -
Epoch 20/20
326/326 [==============================] - 2s - loss: 0.3530 - acc: 0.8610 - val_loss: 0.3838 -
CPU times: user 1min 34s, sys: 8.24 s, total: 1min 43s
Wall time: 60 s
```

```
Out[129]: <keras.callbacks.History at 0x147370278>
```

```
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 [133]: %%time

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

```
Train on 326 samples, validate on 82 samples
Epoch 1/20
326/326 [==============================] - 5s - loss: 0.6452 - acc: 0.7807 - val_loss: 0.5479 -
Epoch 2/20
326/326 [==============================] - 5s - loss: 0.4646 - acc: 0.8382 - val_loss: 0.4357 -
Epoch 3/20
326/326 [==============================] - 5s - loss: 0.4175 - acc: 0.8417 - val_loss: 0.4228 -
Epoch 4/20
326/326 [==============================] - 5s - loss: 0.4064 - acc: 0.8443 - val_loss: 0.4128 -
Epoch 5/20
326/326 [==============================] - 5s - loss: 0.3960 - acc: 0.8454 - val_loss: 0.4106 -
Epoch 6/20
326/326 [==============================] - 5s - loss: 0.3923 - acc: 0.8478 - val_loss: 0.4106 -
Epoch 7/20
326/326 [==============================] - 5s - loss: 0.3878 - acc: 0.8524 - val_loss: 0.4125 -
Epoch 8/20
326/326 [==============================] - 5s - loss: 0.3875 - acc: 0.8468 - val_loss: 0.4039 -
Epoch 9/20
326/326 [==============================] - 5s - loss: 0.3854 - acc: 0.8490 - val_loss: 0.3999 -
Epoch 10/20
326/326 [==============================] - 5s - loss: 0.3779 - acc: 0.8543 - val_loss: 0.4008 -
Epoch 11/20
326/326 [==============================] - 5s - loss: 0.3749 - acc: 0.8569 - val_loss: 0.3961 -
Epoch 12/20
326/326 [==============================] - 5s - loss: 0.3679 - acc: 0.8591 - val_loss: 0.3938 -
```

```
Epoch 13/20
326/326 [==============================] - 5s - loss: 0.3628 - acc: 0.8602 - val_loss: 0.3900 -
Epoch 14/20
326/326 [==============================] - 5s - loss: 0.3584 - acc: 0.8623 - val_loss: 0.3988 -
Epoch 15/20
326/326 [==============================] - 5s - loss: 0.3597 - acc: 0.8617 - val_loss: 0.3868 -
Epoch 16/20
326/326 [==============================] - 5s - loss: 0.3570 - acc: 0.8621 - val_loss: 0.3835 -
Epoch 17/20
326/326 [==============================] - 5s - loss: 0.3524 - acc: 0.8625 - val_loss: 0.3843 -
Epoch 18/20
326/326 [==============================] - 5s - loss: 0.3519 - acc: 0.8625 - val_loss: 0.3849 -
Epoch 19/20
326/326 [==============================] - 5s - loss: 0.3494 - acc: 0.8636 - val_loss: 0.3777 -
Epoch 20/20
326/326 [==============================] - 5s - loss: 0.3447 - acc: 0.8643 - val_loss: 0.3779 -
CPU times: user 3min 24s, sys: 13 s, total: 3min 37s
Wall time: 1min 57s
```

```
Out[133]: <keras.callbacks.History at 0x141d05080>
```

```
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]: 0.75923135423709032
```

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.