Loading the Data

We will be working with the Babi Data Set from Facebook Research.

Full Details: https://research.fb.com/downloads/babi/ (https://research.fb.com/downloads/babi/)

- Jason Weston, Antoine Bordes, Sumit Chopra, Tomas Mikolov, Alexander M. Rush, "Towards AI-Complete Question Answering: A Set of Prerequisite Toy Tasks", http://arxiv.org/abs/1502.05698 (http://arxiv.org/abs/1502.05698)

In [1]:
```
import pickle
import numpy as np
```

In [2]:
```
with open("train_qa.txt", "rb") as fp:   # Unpickling
    train_data =  pickle.load(fp)
```

In [3]:
```
with open("test_qa.txt", "rb") as fp:   # Unpickling
    test_data =  pickle.load(fp)
```

Exploring the Format of the Data

In [4]:
```
type(test_data)
```

Out[4]: list

In [5]:
```
type(train_data)
```

Out[5]: list

In [6]:
```
len(test_data)
```

Out[6]: 1000

In [7]:
```
len(train_data)
```

Out[7]: 10000

In [8]:
```
1 train_data[0]
```

Out[8]:
```
(['Mary',
  'moved',
  'to',
  'the',
  'bathroom',
  '.',
  'Sandra',
  'journeyed',
  'to',
  'the',
  'bedroom',
  '.'],
 ['Is', 'Sandra', 'in', 'the', 'hallway', '?'],
 'no')
```

In [9]:
```
1 ' '.join(train_data[0][0])
```

Out[9]:
```
'Mary moved to the bathroom . Sandra journeyed to the bedroom .'
```

In [10]:
```
1 ' '.join(train_data[0][1])
```

Out[10]:
```
'Is Sandra in the hallway ?'
```

In [11]:
```
1 train_data[0][2]
```

Out[11]:
```
'no'
```

Setting up Vocabulary of All Words

In [12]:
```
1 # Create a set that holds the vocab words
2 vocab = set()
```

In [13]:
```
1 all_data = test_data + train_data
```

In [14]:
```
1 for story, question , answer in all_data:
2     # In case you don't know what a union of sets is:
3     # https://www.programiz.com/python-programming/methods/set/uni
4     vocab = vocab.union(set(story))
5     vocab = vocab.union(set(question))
```

In [15]:
```
1 vocab.add('no')
2 vocab.add('yes')
```

In [16]:
```
1 vocab
```

Out[16]:
```
{'.',
 '?',
 'Daniel',
 'Is',
 'John',
 'Mary',
 'Sandra',
 'apple',
 'back',
 'bathroom',
 'bedroom',
 'discarded',
 'down',
 'dropped',
 'football',
 'garden',
 'got',
 'grabbed',
 'hallway',
 'in',
 'journeyed',
 'kitchen',
 'left',
 'milk',
 'moved',
 'no',
 'office',
 'picked',
 'put',
 'the',
 'there',
 'to',
 'took',
 'travelled',
 'up',
 'went',
 'yes'}
```

In [17]:
```
1 vocab_len = len(vocab) + 1 #we add an extra space to hold a 0 for
```

In [18]:
```
1 max_story_len = max([len(data[0]) for data in all_data])
```

In [19]:
```
1 max_story_len
```

Out[19]:
```
156
```

In [20]:
```
1 max_question_len = max([len(data[1]) for data in all_data])
```

In [21]:
```
1 max_question_len
```

Out[21]:
```
6
```

Vectorizing the Data

In [22]:
```
1 vocab
```

Out[22]:
```
{'.',
 '?',
 'Daniel',
 'Is',
 'John',
 'Mary',
 'Sandra',
 'apple',
 'back',
 'bathroom',
 'bedroom',
 'discarded',
 'down',
 'dropped',
 'football',
 'garden',
 'got',
 'grabbed',
 'hallway',
 'in',
 'journeyed',
 'kitchen',
 'left',
 'milk',
 'moved',
 'no',
 'office',
 'picked',
 'put',
 'the',
 'there',
 'to',
 'took',
 'travelled',
 'up',
 'went',
 'yes'}
```

In [23]:
```
1 # Reserve 0 for pad_sequences
2 vocab_size = len(vocab) + 1
```

In [24]:
```
1 from keras.preprocessing.sequence import pad_sequences
2 from keras.preprocessing.text import Tokenizer
```

```
Using TensorFlow backend.
```

In [25]:
```
# integer encode sequences of words
tokenizer = Tokenizer(filters=[])
tokenizer.fit_on_texts(vocab)
```

In [26]:
```
tokenizer.word_index
```

Out[26]:
```
{'.': 13,
 '?': 12,
 'apple': 3,
 'back': 18,
 'bathroom': 23,
 'bedroom': 29,
 'daniel': 21,
 'discarded': 5,
 'down': 22,
 'dropped': 4,
 'football': 37,
 'garden': 36,
 'got': 34,
 'grabbed': 15,
 'hallway': 27,
 'in': 8,
 'is': 30,
 'john': 10,
 'journeyed': 35,
 'kitchen': 11,
 'left': 31,
 'mary': 25,
 'milk': 7,
 'moved': 28,
 'no': 6,
 'office': 24,
 'picked': 1,
 'put': 33,
 'sandra': 2,
 'the': 32,
 'there': 16,
 'to': 14,
 'took': 9,
 'travelled': 20,
 'up': 17,
 'went': 26,
 'yes': 19}
```

In [27]:
```
train_story_text = []
train_question_text = []
train_answers = []

for story,question,answer in train_data:
    train_story_text.append(story)
    train_question_text.append(question)
```

In [28]:
```
train_story_seq = tokenizer.texts_to_sequences(train_story_text)
```

In [29]:
```
len(train_story_text)
```

Out[29]:
```
10000
```

In [30]:
```
1 len(train_story_seq)
```

Out[30]:
```
10000
```

In [31]:
```
1 # word_index = tokenizer.word_index
```

Functionalize Vectorization

In [32]:

```
def vectorize_stories(data, word_index=tokenizer.word_index, max_s
    '''
    INPUT:

    data: consisting of Stories,Queries,and Answers
    word_index: word index dictionary from tokenizer
    max_story_len: the length of the longest story (used for pad_s
    max_question_len: length of the longest question (used for pad


    OUTPUT:

    Vectorizes the stories,questions, and answers into padded sequ
    answer in the data. Then we convert the raw words to an word i
    output list. Then once we have converted the words to numbers,

    Returns this in the form of a tuple (X,Xq,Y) (padded based on
    '''


    # X = STORIES
    X = []
    # Xq = QUERY/QUESTION
    Xq = []
    # Y = CORRECT ANSWER
    Y = []


    for story, query, answer in data:

        # Grab the word index for every word in story
        x = [word_index[word.lower()] for word in story]
        # Grab the word index for every word in query
        xq = [word_index[word.lower()] for word in query]

        # Grab the Answers (either Yes/No so we don't need to use
        # Index 0 is reserved so we're going to use + 1
        y = np.zeros(len(word_index) + 1)

        # Now that y is all zeros and we know its just Yes/No , we
        #
        y[word_index[answer]] = 1

        # Append each set of story,query, and answer to their resp
        X.append(x)
        Xq.append(xq)
        Y.append(y)

    # Finally, pad the sequences based on their max length so the

    # RETURN TUPLE FOR UNPACKING
    return (pad_sequences(X, maxlen=max_story_len),pad_sequences()
```

In [33]:
```
1 inputs_train, queries_train, answers_train = vectorize_stories(tra
```

In [34]:
```
1 inputs_test, queries_test, answers_test = vectorize_stories(test_c
```

In [35]:
```
1 inputs_test
```

Out[35]:
```
array([[ 0,  0,  0, ..., 32, 29, 13],
       [ 0,  0,  0, ..., 32, 36, 13],
       [ 0,  0,  0, ..., 32, 36, 13],
       ...,
       [ 0,  0,  0, ..., 32,  3, 13],
       [ 0,  0,  0, ..., 32, 36, 13],
       [ 0,  0,  0, ...,  3, 16, 13]])
```

In [36]:
```
1 queries_test
```

Out[36]:
```
array([[30, 10,  8, 32, 11, 12],
       [30, 10,  8, 32, 11, 12],
       [30, 10,  8, 32, 36, 12],
       ...,
       [30, 25,  8, 32, 29, 12],
       [30,  2,  8, 32, 36, 12],
       [30, 25,  8, 32, 36, 12]])
```

In [37]:
```
1 answers_test
```

Out[37]:
```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

In [38]:
```
1 sum(answers_test)
```

Out[38]:
```
array([  0.,   0.,   0.,   0.,   0.,   0., 503.,   0.,   0.,   0.,
0.,
         0.,   0.,   0.,   0.,   0.,   0.,   0.,   0., 497.,   0.,
0.,
         0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,
0.,
         0.,   0.,   0.,   0.,   0.])
```

In [39]:
```
1 tokenizer.word_index['yes']
```

Out[39]:
```
19
```

In [40]:
```
1 tokenizer.word_index['no']
```

Out[40]:
```
6
```

Creating the Model

In [41]:
```
from keras.models import Sequential, Model
from keras.layers.embeddings import Embedding
from keras.layers import Input, Activation, Dense, Permute, Dropou
from keras.layers import add, dot, concatenate
from keras.layers import LSTM
```

Placeholders for Inputs

Recall we technically have two inputs, stories and questions. So we need to use placeholders. `Input()` is used to instantiate a Keras tensor.

In [42]:
```
input_sequence = Input((max_story_len,))
question = Input((max_question_len,))
```

Building the Networks

To understand why we chose this setup, make sure to read the paper we are using:

- Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, Rob Fergus, "End-To-End Memory Networks", http://arxiv.org/abs/1503.08895 (http://arxiv.org/abs/1503.08895)

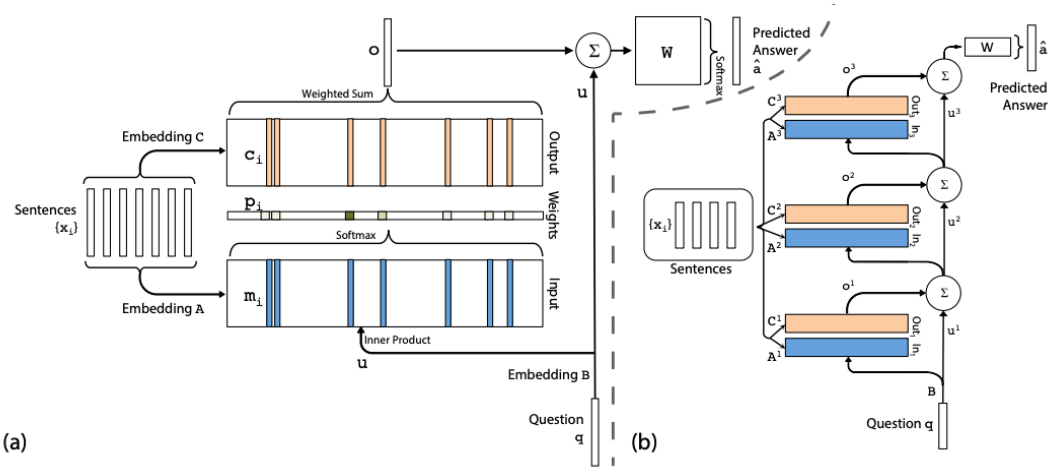


Figure 1: (a): A single layer version of our model. (b): A three layer version of our model. In practice, we can constrain several of the embedding matrices to be the same (see Section 2.2).

Encoders

Input Encoder m

In [43]:
```
# Input gets embedded to a sequence of vectors
input_encoder_m = Sequential()
input_encoder_m.add(Embedding(input_dim=vocab_size,output_dim=64))
input_encoder_m.add(Dropout(0.3))

# This encoder will output:
# (samples, story_maxlen, embedding_dim)
```

Input Encoder c

In [44]:
```
# embed the input into a sequence of vectors of size query_maxlen
input_encoder_c = Sequential()
input_encoder_c.add(Embedding(input_dim=vocab_size,
                              output_dim=max_question_len))
input_encoder_c.add(Dropout(0.3))
# output: (samples, story_maxlen, query_maxlen)
```

Question Encoder

In [45]:
```
# embed the question into a sequence of vectors
question_encoder = Sequential()
question_encoder.add(Embedding(input_dim=vocab_size,
                               output_dim=64,
                               input_length=max_question_len))
question_encoder.add(Dropout(0.3))
# output: (samples, query_maxlen, embedding_dim)
```

Encode the Sequences

In [46]:
```
# encode input sequence and questions (which are indices)
# to sequences of dense vectors
input_encoded_m = input_encoder_m(input_sequence)
input_encoded_c = input_encoder_c(input_sequence)
question_encoded = question_encoder(question)
```

Use dot product to compute the match between first input vector seq and the query

In [47]:
```
# shape: `(samples, story_maxlen, query_maxlen)`
match = dot([input_encoded_m, question_encoded], axes=(2, 2))
match = Activation('softmax')(match)
```

Add this match matrix with the second input vector sequence

In [48]:
```
# add the match matrix with the second input vector sequence
response = add([match, input_encoded_c])  # (samples, story_maxler
response = Permute((2, 1))(response)  # (samples, query_maxlen, st
```

Concatenate

In [49]:
```
# concatenate the match matrix with the question vector sequence
answer = concatenate([response, question_encoded])
```

In [50]:
```
answer
```

Out[50]:
```
<tf.Tensor 'concatenate_1/concat:0' shape=(?, 6, 220) dtype=float32>
```

In [51]:
```
# Reduce with RNN (LSTM)
answer = LSTM(32)(answer)  # (samples, 32)
```

In [52]:
```
# Regularization with Dropout
answer = Dropout(0.5)(answer)
answer = Dense(vocab_size)(answer)  # (samples, vocab_size)
```

In [53]:
```
# we output a probability distribution over the vocabulary
answer = Activation('softmax')(answer)

# build the final model
model = Model([input_sequence, question], answer)
model.compile(optimizer='rmsprop', loss='categorical_crossentropy'
              metrics=['accuracy'])
```

In [54]:
```
1 model.summary()
```

```
__________________________________________________________________________________
_________________________________
Layer (type)                    Output Shape         Param #     Con
nected to
==================================================================================
================================
input_1 (InputLayer)            (None, 156)          0
__________________________________________________________________________________
_________________________________
input_2 (InputLayer)            (None, 6)            0
__________________________________________________________________________________
_________________________________
sequential_1 (Sequential)       multiple             2432        inp
ut_1[0][0]
__________________________________________________________________________________
_________________________________
sequential_3 (Sequential)       (None, 6, 64)        2432        inp
ut_2[0][0]
__________________________________________________________________________________
_________________________________
dot_1 (Dot)                     (None, 156, 6)       0           seq
uential_1[1][0]
                                                                 seq
uential_3[1][0]
__________________________________________________________________________________
_________________________________
activation_1 (Activation)       (None, 156, 6)       0           dot
_1[0][0]
__________________________________________________________________________________
_________________________________
sequential_2 (Sequential)       multiple             228         inp
ut_1[0][0]
__________________________________________________________________________________
_________________________________
add_1 (Add)                     (None, 156, 6)       0           act
ivation_1[0][0]
                                                                 seq
uential_2[1][0]
__________________________________________________________________________________
_________________________________
permute_1 (Permute)             (None, 6, 156)       0           add
_1[0][0]
__________________________________________________________________________________
_________________________________
concatenate_1 (Concatenate)     (None, 6, 220)       0           per
mute_1[0][0]
                                                                 seq
uential_3[1][0]
__________________________________________________________________________________
_________________________________
lstm_1 (LSTM)                   (None, 32)           32384       con
catenate_1[0][0]
__________________________________________________________________________________
_________________________________
dropout_4 (Dropout)             (None, 32)           0           lst
m_1[0][0]
__________________________________________________________________________________
_________________________________
dense_1 (Dense)                 (None, 38)           1254        dro
pout_4[0][0]
__________________________________________________________________________________
```

```
_________________________________
activation_2 (Activation)       (None, 38)           0           den
se_1[0][0]
==================================================================
==============================
Total params: 38,730
Trainable params: 38,730
Non-trainable params: 0
__________________________________________________________________
_________________________________
```

In [55]:
```
# train
history = model.fit([inputs_train, queries_train], answers_train,bat
```

```
0.1131 - acc: 0.9560 - val_loss: 0.2496 - val_acc: 0.9180
Epoch 115/120
10000/10000 [==============================] - 4s 364us/step - loss:
0.1050 - acc: 0.9599 - val_loss: 0.3021 - val_acc: 0.9170
Epoch 116/120
10000/10000 [==============================] - 3s 342us/step - loss:
0.1038 - acc: 0.9619 - val_loss: 0.2673 - val_acc: 0.9160
Epoch 117/120

10000/10000 [==============================] - 4s 350us/step - loss:
0.1100 - acc: 0.9599 - val_loss: 0.2855 - val_acc: 0.9160
Epoch 118/120
10000/10000 [==============================] - 3s 349us/step - loss:
0.1012 - acc: 0.9610 - val_loss: 0.2887 - val_acc: 0.9200
Epoch 119/120
10000/10000 [==============================] - 4s 352us/step - loss:
0.1040 - acc: 0.9618 - val_loss: 0.2774 - val_acc: 0.9120
Epoch 120/120
10000/10000 [==============================] - 3s 349us/step - loss:
0.1010 - acc: 0.9615 - val_loss: 0.3139 - val_acc: 0.9120
```

Saving the Model

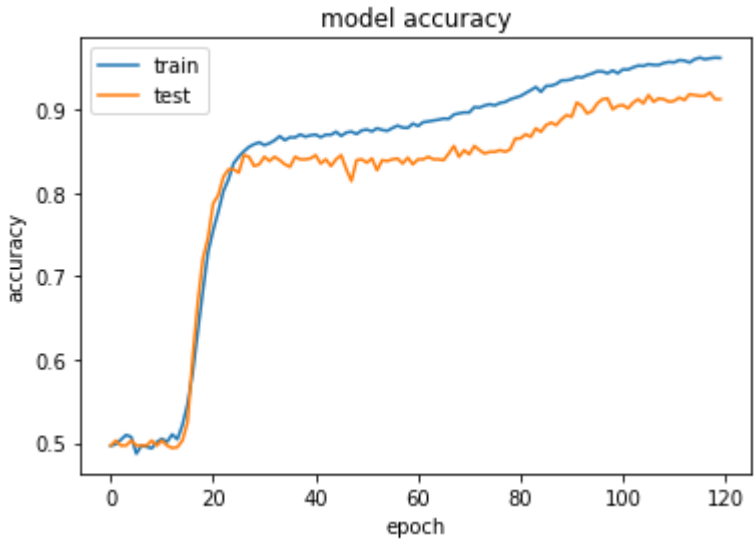
In [72]:
```
filename = 'chatbot_120_epochs.h5'
model.save(filename)
```

Evaluating the Model

Plotting Out Training History

In [57]:
```
import matplotlib.pyplot as plt
%matplotlib inline
print(history.history.keys())
# summarize history for accuracy
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

```
dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
```



Evaluating on Given Test Set

In [73]:
```
model.load_weights(filename)
pred_results = model.predict(([inputs_test, queries_test]))
```

In [74]:
```
test_data[0][0]
```

Out[74]:
```
['Mary',
 'got',
 'the',
 'milk',
 'there',
 '.',
 'John',
 'moved',
 'to',
 'the',
 'bedroom',
 '.']
```

In [75]:
```
story =' '.join(word for word in test_data[0][0])
print(story)
```

```
Mary got the milk there . John moved to the bedroom .
```

In [76]:
```
query = ' '.join(word for word in test_data[0][1])
print(query)
```

```
Is John in the kitchen ?
```

In [77]:
```
print("True Test Answer from Data is:",test_data[0][2])
```

```
True Test Answer from Data is: no
```

In [78]:
```
#Generate prediction from model
val_max = np.argmax(pred_results[0])

for key, val in tokenizer.word_index.items():
    if val == val_max:
        k = key

print("Predicted answer is: ", k)
print("Probability of certainty was: ", pred_results[0][val_max])
```

```
Predicted answer is:  no
Probability of certainty was:  0.9999999
```

Writing Your Own Stories and Questions

Remember you can only use words from the existing vocab

In [79]:
```
1 vocab
```

Out[79]:
```
{'.',
 '?',
 'Daniel',
 'Is',
 'John',
 'Mary',
 'Sandra',
 'apple',
 'back',
 'bathroom',
 'bedroom',
 'discarded',
 'down',
 'dropped',
 'football',
 'garden',
 'got',
 'grabbed',
 'hallway',
 'in',
 'journeyed',
 'kitchen',
 'left',
 'milk',
 'moved',
 'no',
 'office',
 'picked',
 'put',
 'the',
 'there',
 'to',
 'took',
 'travelled',
 'up',
 'went',
 'yes'}
```

In [80]:
```
1 # Note the whitespace of the periods
2 my_story = "John left the kitchen . Sandra dropped the football ir
3 my_story.split()
```

Out[80]:
```
['John',
 'left',
 'the',
 'kitchen',
 '.',
 'Sandra',
 'dropped',
 'the',
 'football',
 'in',
 'the',
 'garden',
 '.']
```

In [81]:
```
my_question = "Is the football in the garden ?"
```

In [82]:
```
my_question.split()
```

Out[82]:
```
['Is', 'the', 'football', 'in', 'the', 'garden', '?']
```

In [83]:
```
mydata = [(my_story.split(),my_question.split(),'yes')]
```

In [84]:
```
my_story,my_ques,my_ans = vectorize_stories(mydata)
```

In [85]:
```
pred_results = model.predict(([ my_story, my_ques]))
```

In [86]:
```
#Generate prediction from model
val_max = np.argmax(pred_results[0])

for key, val in tokenizer.word_index.items():
    if val == val_max:
        k = key

print("Predicted answer is: ", k)
print("Probability of certainty was: ", pred_results[0][val_max])
```

```
Predicted answer is:  yes
Probability of certainty was:  0.97079676
```

Great Job!