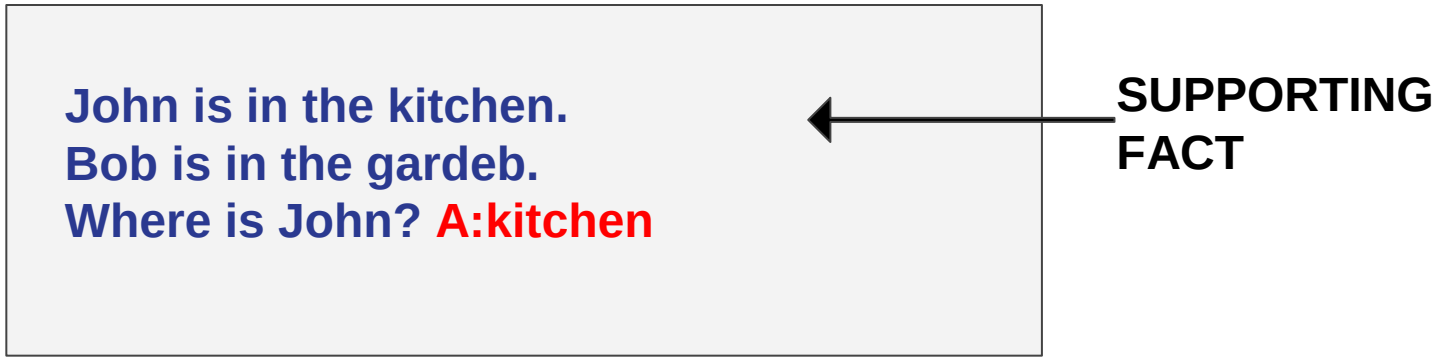
Machine Comprehension

Pruthvij Thakar

Tushar Goel

1 Problem Statement

- Question Answering Task
- Facebook Babi Dataset
- Example: Task with single supporting fact



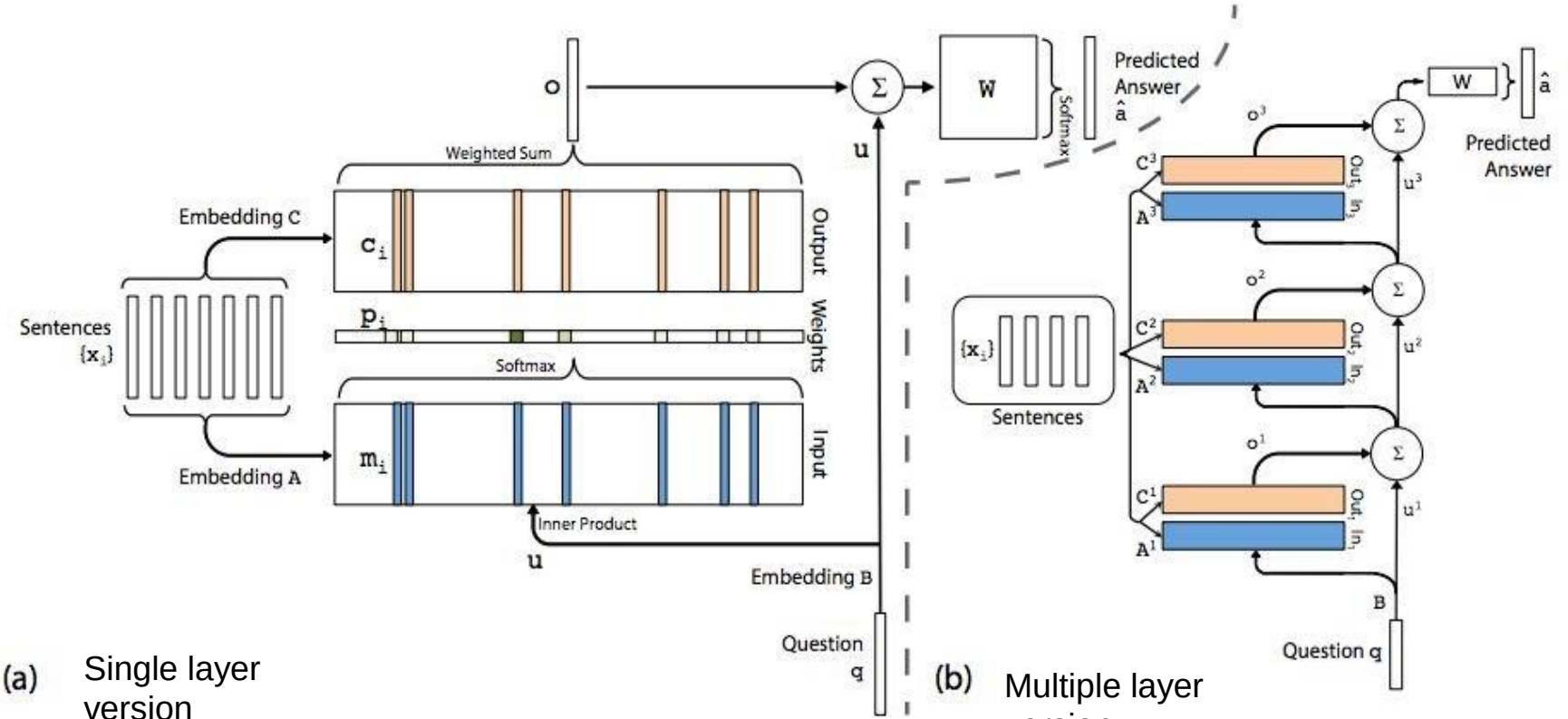
2 MEMORY NETWORKS

- Memory Networks were introduced in 2014 as a work to be presented in International Conference on Learning Representations (ICLR) 2016.
- Class of models that combine large memory with learning component that can read and write to it.
- Authors of the paper: J. Weston, S. Chopra and A. Bordes ● Simple Concept:
  - Store the text in memory
  - Transform them by using embedding matrices.
  - Now do computations over them to train the model.

3 End to End Memory Networks

- Neural Network model with external memory.
- Reads the memory with soft attention.
- It accesses memory multiple times; each step being called a hop.
- Uses back propagation to update the model.

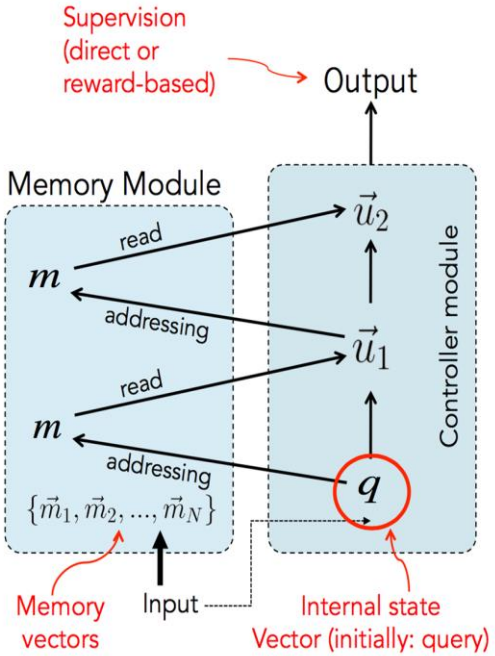
Architecture of End to End Memory Networks



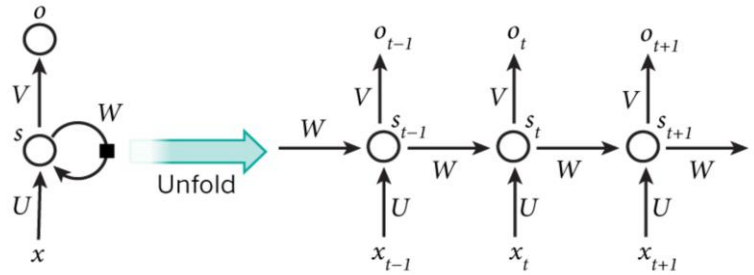
4 Model Details

- Input Sentences: $x_1, x_2, \ldots, x_n$ is taken
- Sentences are embedded into memory vectors m_i and c_i by using embedding matrices A and C.
- Question q is embedded into internal state u.
- Matching is performed between u and m_i with SoftMax function.
- Output is calculated by the relation: $o = \sum p_i c$
- Another Softmax is used to produce final predictions after summing up o with u.

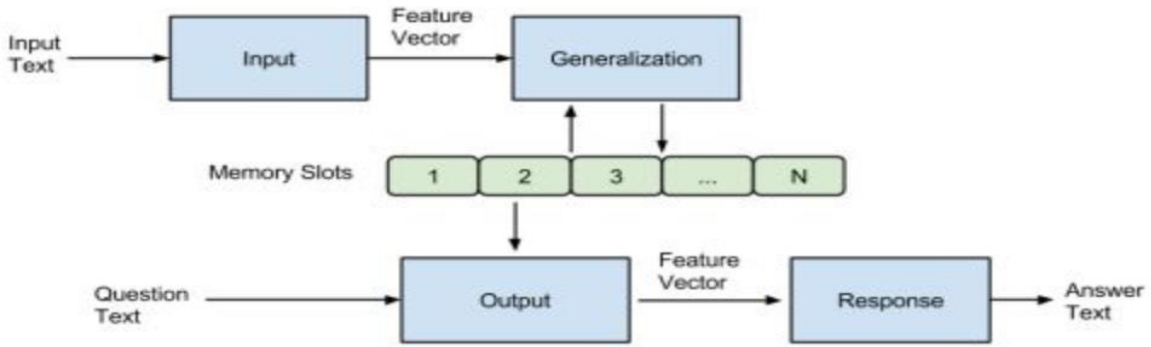
5 MEMORY NETWORK



[Figure by Saina Sukhbaatar]



6 STEPS OF IMPLEMENTATION



1. I (input): converts to word embeddings X(dialog).
2. G (generalization): stores x in next available slot Mn()
3. O (output): Loops over all memories k=1 or 2 times:

a. 1st loop max: finds best match Mi with X(question).

b. 2nd loop max: finds best match Mj with (x, Mi)

c. The output o is represented with (x, Mi, Mj).

4. R (response): ranks all words in the dictionary given O and returns best single word. *(OR: use a full RNN here)*

7 MODEL VISUALIZATION

```
____________________________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
input_1 (InputLayer)             (None, 68)            0
____________________________________________________________________________________________________
input_2 (InputLayer)             (None, 4)             0
____________________________________________________________________________________________________
sequential_1 (Sequential)        multiple              1408        input_1[0][0]
____________________________________________________________________________________________________
sequential_3 (Sequential)        (None, 4, 64)         1408        input_2[0][0]
____________________________________________________________________________________________________
merge_1 (Merge)                  (None, 68, 4)         0           sequential_1[1][0]
                                                                   sequential_3[1][0]
____________________________________________________________________________________________________
activation_1 (Activation)        (None, 68, 4)         0           merge_1[0][0]
____________________________________________________________________________________________________
sequential_2 (Sequential)        multiple              88          input_1[0][0]
____________________________________________________________________________________________________
merge_2 (Merge)                  (None, 68, 4)         0           activation_1[0][0]
                                                                   sequential_2[1][0]
____________________________________________________________________________________________________
permute_1 (Permute)              (None, 4, 68)         0           merge_2[0][0]
____________________________________________________________________________________________________
merge_3 (Merge)                  (None, 4, 132)        0           permute_1[0][0]
                                                                   sequential_3[1][0]
____________________________________________________________________________________________________
lstm_1 (LSTM)                    (None, 64)            50432       merge_3[0][0]
____________________________________________________________________________________________________
dropout_4 (Dropout)              (None, 64)            0           lstm_1[0][0]
____________________________________________________________________________________________________
dense_1 (Dense)                  (None, 22)            1430        dropout_4[0][0]
____________________________________________________________________________________________________
activation_2 (Activation)        (None, 22)            0           dense_1[0][0]
====================================================================================================
Total params: 54,766
Trainable params: 54,766
Non-trainable params: 0
____________________________________________________________________________________________________
None
```

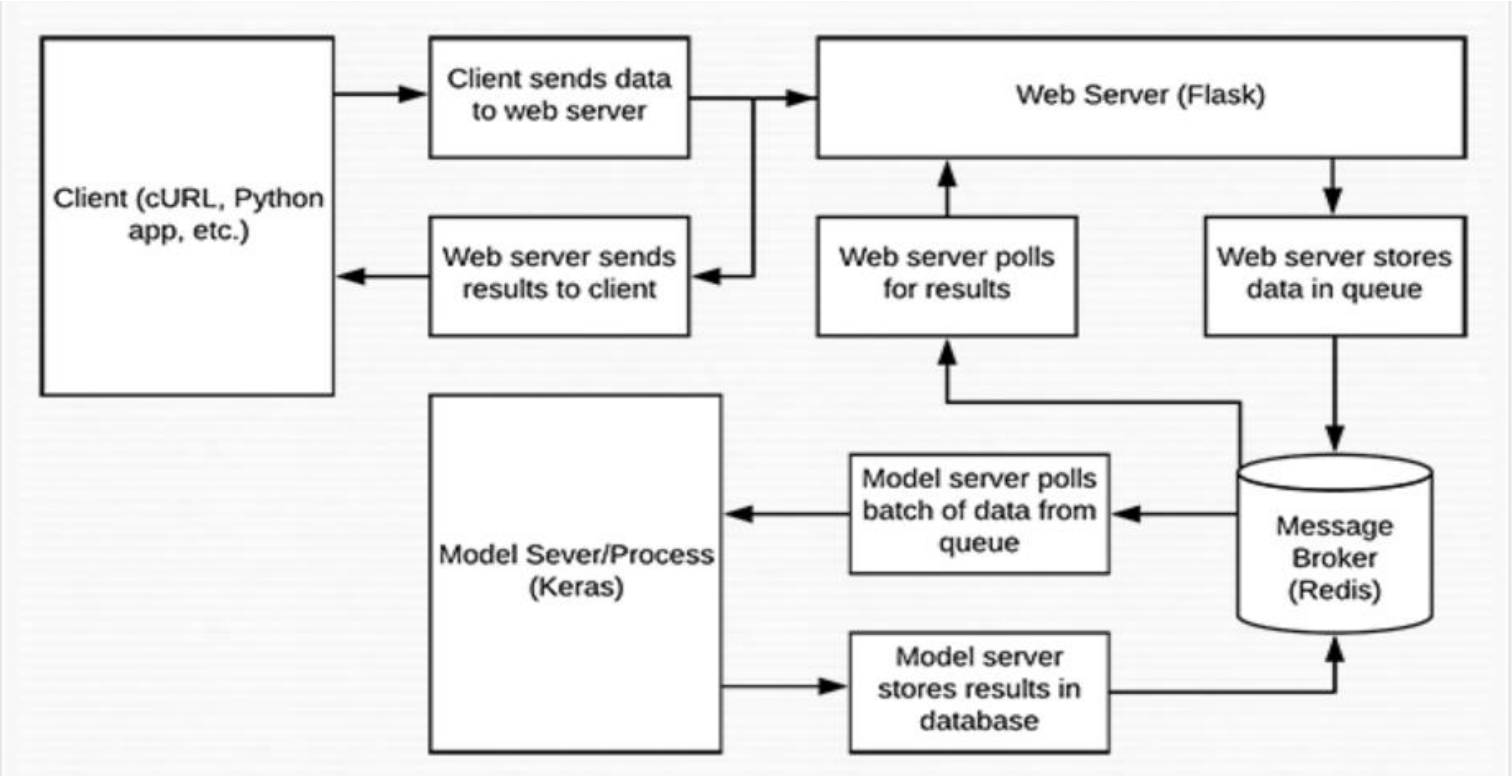
8 PREPROCESSING

```
def get_stories(f, only_supporting=False, max_length=None):
    '''Given a file name, read the file,
    retrieve the stories,
    and then convert the sentences into a single story.
    If max_length is supplied,
    any stories longer than max_length tokens will be discarded.
    '''
    data = parse_stories(f.readlines(), only_supporting=only_supporting)
    flatten = lambda data: reduce(lambda x, y: x + y, data)
    data = [(flatten(story), q, answer) for story, q, answer in data if not max_length or len(flatten(story)) < max_length]
    return data
```

```
def vectorize_stories(data, word_idx, story_maxlen, query_maxlen):
    X = []
    Xq = []
    Y = []
    for story, query, answer in data:
        x=[word_idx[w] for w in story]
        xq=[word_idx[w] for w in query]
        y=np.zeros(len(word_idx) + 1)
        y[word_idx[answer]]=1
        X.append(x)
        Xq.append(xq)
        Y.append(y)
    return (pad_sequences(X, maxlen=story_maxlen),
            pad_sequences(Xq, maxlen=query_maxlen),
            np.array(Y))
```

- **We take an approach of vectorizing only the vocabulary that is already present in the stories and questions instead of loading the whole Glove or Word2Vec vectors, this helps in speeding up the process of training the model as well as limiting the scope to the business use case.**

9 ARCHITECTURE



10 RESULTS/OBSERVATIONS ON FIRST TASK FOR TESTING

Layers	Dropouts	Batch-size	Epochs	Results
LSTM(32)	0.3	32	100	94.6%
LSTM(64)	0.3	32	100	96.5%
LSTM(32), LSTM(32)	(0.5,0.5)	32	100	92.4%
LSTM(32), LSTM(32)	(0.5, 0.5)	32	200	96.9%
GRU(32)	0.3	32	100	86.4%
GRU(64)	0.3	32	100	87.4%
GRU(32), GRU(32)	(0.5,0.5)	64	100	52.6%

11 TRYING BEST MODEL ON VARIOUS TASKS IN BABI

TASKS	TRAINING ACCURACY (IN %)	TESTING ACCURACY (IN %)
SINGLE FACT	0.994	0.950
TWO SUPPORTING FACTS	0.72	0.35
THREE SUPPORTING FACTS	0.94	0.23
TWO ARGS RELATION TEST	0.99	0.99
THREE ARGS RELATION TEST	0.987	0.868
YES-NO QUESTIONS	0.957	0.821
SIMPLE NEGATION	0.994	0.907
BASIC COREFERNCE	0.997	0.985
CONJUNCTION	0.997	0.981

DEMO

13 CONCLUSION

- The models with two or more layers required more training since there are more parameters that need to be set, but then have greater accuracies than the other models once trained completely.
- Overall, LSTM based models performed better than GRU based models for this task.

14 FUTURE TASKS

- There are many experiments that can be carried out such as adding neural attention layer along with the deep reinforcement approach to attain good accuracy (https://arxiv.org/pdf/1705.04304.pdf)
- We can also integrate episodic memory modules along with softGRU's and dynamic attention flow mechanism (https://arxiv.org/abs/1506.07285)

Thank you