Question Answering

Tarun Tater

May 18, 2017

Problem statement

- To build an AI system which can answer multiple choice questions from a standardized 8th grade Science exam.
    - Answer Selection
    - Answer Generation

Motivation

- A step towards a completely generic and automated question-answering system.
- Current research moving towards non-factoid question answering.
- Understand the typical issues in natural language understanding.
- Applications in education, healthcare, personal assistance etc.

Data

- The training data consists of 2,500 multiple choice questions.
- Each question has four possible answers and exactly one correct answer.
- Broad categories of questions : Who, Where ,Why, Which, What, How, When, Fill in the blanks etc.
- Test data - 21298 questions.

Example Questions

1. The brain, spinal cord, and nerves are organs that perform which primary function?
   1. supporting the body and enabling it to move
   2. transporting oxygen, wastes, and nutrients
   3. producing male and female gametes
   4. conducting messages to coordinate body functions
2. Which set contains kingdoms that contain only heterotrophs?
   1. Protists, Fungi
   2. Bacteria, Animals
   3. Plants, Fungi
   4. Animals, Fungi
3. Fossil evidence can be used to tell us about all of these except.
   1. biological diversity (the kinds of organisms that lived)
   2. episodic speciation (new species suddenly appearing after millions of years without changes to the species)
   3. mass extinction (sudden disappearance of all evidence of an organism)
   4. mutation rate (how fast mutations occur within a species)

Corpus

- All the chapters of Ck12's 8th standard science textbook were extracted to create the corpus.
- Wikipedia pages of all the topics of the Ck12 textbook were also extracted.
- NCERT 7th to 9th standard Science textbooks.
- Studystack - 454743 questions and answer pairs.

Preprocessing

- Remove Punctuations and special symbols.
- Custom Stop Words - 153 words.
- Stemming.
- POS Tagging.
    - Only nouns
    - Only use (noun, verb, adj/adv)
- Negative Question tags.
- Tweaked the options of the form : “Both A & B”, “All of the above” .

Explored Approaches

- Information Retrieval (lucene).
- Word Embeddings.
- Deep Learning Models.

Similarity Measures

- Cosine Similarity.
- Euclidean Distance.
- AESD : Arithmetic mean of Euclidean and Sigmoid Dot product.

$$\frac{0.5}{1+||x-y||} + \frac{0.5}{1+e^{-\gamma(xy^T+c)}}$$

- GESD : Geometric mean of Euclidean and Sigmoid Dot product.

$$\frac{0.5}{1+||x-y||} * \frac{0.5}{1+e^{-\gamma(xy^T+c)}}$$

IR Approach

- The idea is to find segments of text which answer the question.
- The document relevant to the question in consideration is to be retrieved in which the answer should be searched.
- PyLucene, an open source IR system was used to retrieve the document.
- The retrieved document size was an extremely important hyper-parameter.

Information Retrieval

- Two Principal Stages : Indexing and Searching
- During indexing, each document is broken into words, and the list of documents containing each word is stored in a list called the “postings list” (specific to Lucene).
- The index consists of all the posting lists for the words in the corpus.
- Retrieval is the process starting with a query and ending with a ranked list of documents. In order to find matches for the query, we break it into the individual words, and check in the posting lists.
- The full ranked list of documents containing the keywords is retrieved.

Lucene Scoring Algorithm

The factors involved in Lucene's scoring algorithm are as follows:

- tf = term frequency in document = measure of how often a term appears in the document
- idf = inverse document frequency = measure of how often the term appears across the index
- coord = number of terms in the query that were found in the document
- lengthNorm = measure of the importance of a term according to the total number of terms in the field
- queryNorm = normalization factor so that queries can be compared

Diagram of Lucene

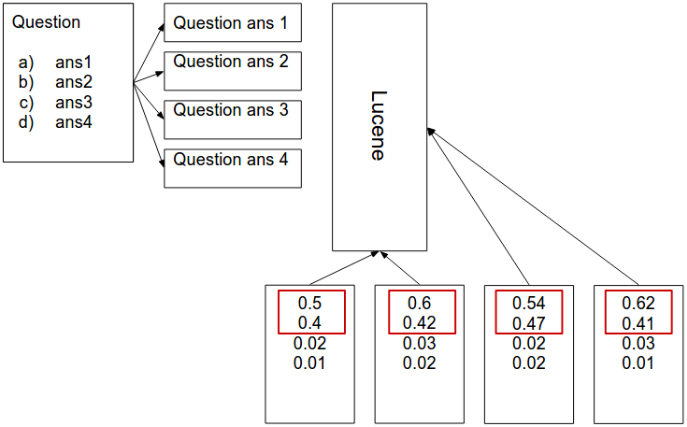
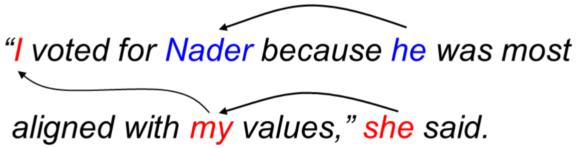


Figure : Lucecne architecture

Building Knowledge Base

- Initial documents - 2321
- Document sizes - varying from 100 - 5000
- Accuracy - 0.36
- Improvements: co-referencing



"I voted for Nader because he was most
aligned with my values," she said.

[1]The Stanford Natural Language Processing Group. (2017). Nlp.stanford.edu. Retrieved 17 May 2017, from https://nlp.stanford.edu/projects/coref.shtml

Similar questions

- Dataset of around half a million questions.
- Indexed all questions using Lucene
- Finding the most similar questions to the given question.
- Match the 4 possible options with correct answers to those questions.
- Metric for scoring each option: Scores of questions and answers were multiplied, and summed them for the number of similar questions.

Question Types

- There are some questions, that are clearly sort of definitions, i.e. the correct answer can be found in the text.
  - For eg : Which is the best explanation of the term ecology?
- Complex Questions which require reasoning and semantic understanding.
  - For eg : A teacher builds a model of a hydrogen atom. A red golf ball is used for a proton, and a green golf ball is used for an electron. Which is not accurate concerning the model?

Word Embeddings

- Questions are almost always asked in a context. We need to know the context of the question before answering it.
- Learning a vector representation of the questions and options Word2vec was used to get the word embeddings of each word of the question and the answer.
- The probability that words of the question co-occur with words of the correct option is more than that with the other options. This was the motivation to use word2vec.
- Different similarity measures between words of the question and answer using these embeddings were evaluated.

Word2vec Embeddings

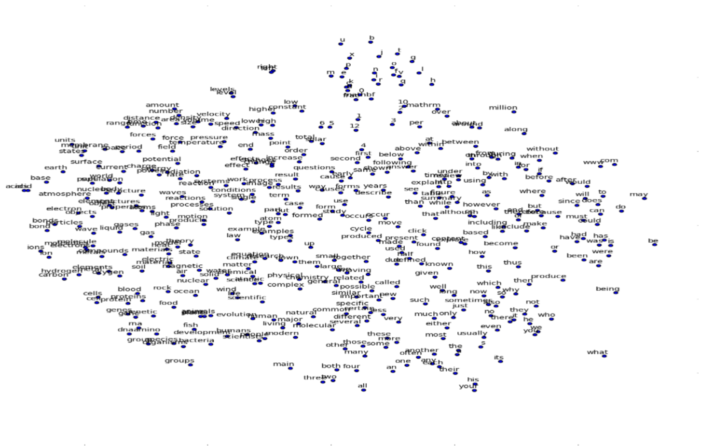


Figure : Word2Vec embeddings

Deep Learning Approaches

- Build a deep learning architecture which predicts the correct answer given the question, options and the paragraph most relevant to the question.
- The paragraph is retrieved using lucene .
- Different variants of this approach tried out where the paragraph and sentence embeddings are trained differently.

Generic Architecture

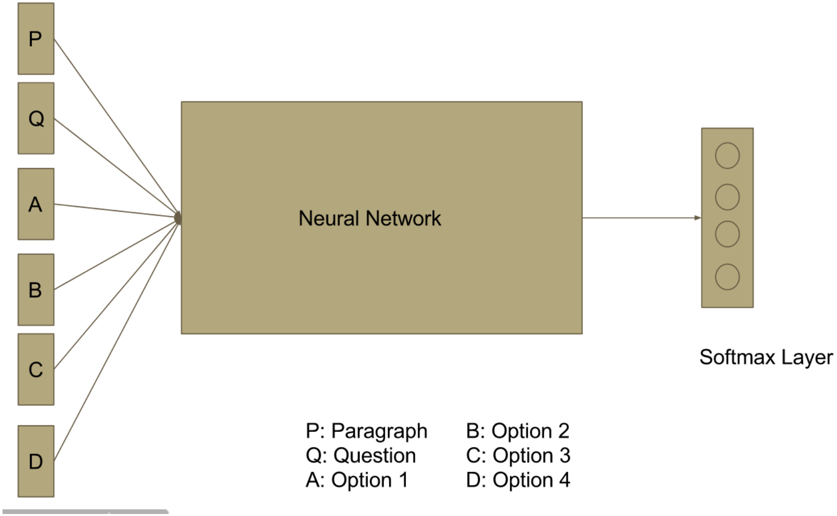


Figure : Generic architecture

Challenges With Sentence Embeddings

*"One **difficulty** in building a model while handling sentences is that a single word can be changed and yet the meaning of the sentence is the same."*

*"One **challenge** in building a model while handling sentences is that a single word can be changed and yet the meaning of the sentence is the same."*

*"One **difficulty** in building a model while handling sentences is that a single word can be changed and yet the meaning of the sentence **changes**."*

Doc2vec

- Word Embeddings give us a representation of the word in a vector dimension.
- But, representing a complete sentence or a paragraph as a vector is a challenging task.
- The representation(embedding) should be able to capture the context of the sentences.
- Doc2Vec explores the possibility of representing documents as context in a vector space.
- Similar documents are placed nearby.
- Objective : Tries to maximise the

$$\sum \log P(targetword|contextwords, documentcontext)$$

Doc2vec Architecture

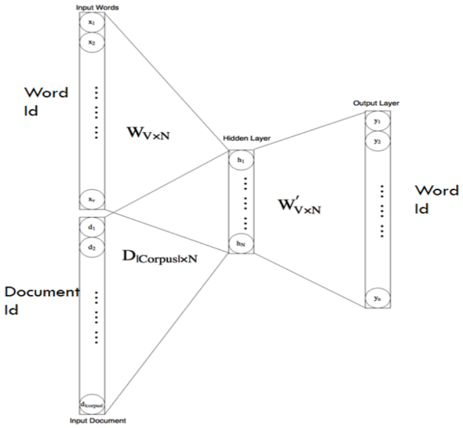


Figure : Doc2Vec architecture

Skip Thoughts

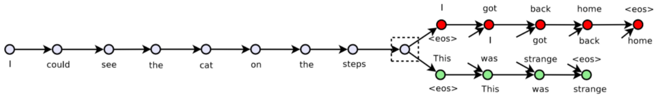


Figure : The skip-thoughts model. Given a tuple $(s_i - 1, s_i, s_i + 1)$ of contiguous sentences, with si the i-th sentence of a book, the sentence si is encoded and tries to reconstruct the previous sentence si−1 and next sentence si+1. In this example, the input is the sentence triplet I got back home. I could see the cat on the steps. This was strange.

Architecture - 1

- Doc2vec was used to get the representations of the paragraph, question and the options.
- The network predicts which of the four options is the answer by estimating each option's probability.
- Another approach was tried where the input to the network is only the paragraph and the question.
- Network predicts the answer vector. The predicted vector is compared to the four options to predict the most similar option as the answer.
- Trying to minimize cosine distance,Euclidean, AESD, GESD. Accuracy : 0.28
- Shortcomings : Sparse embedding of the question and options.

Architecture - 2

- Doc2vec was used to get a representation of the paragraph.
- The word embeddings of the question and option words were passed through an LSTM to get a sentence level representation of the question and options.
- Accuracy : 0.31
- Shortcomings : Unable to generalize (overfitting) , probably very less data.

Architecture - 3

- Word2vec was used to get the representations of the paragraph, question and the options.
- The word embeddings of the paragraph, question and option words were passed through a shared LSTM to get the representations.
- Used negative sampling for data augmentation.
- Accuracy : 0.3

Architecture

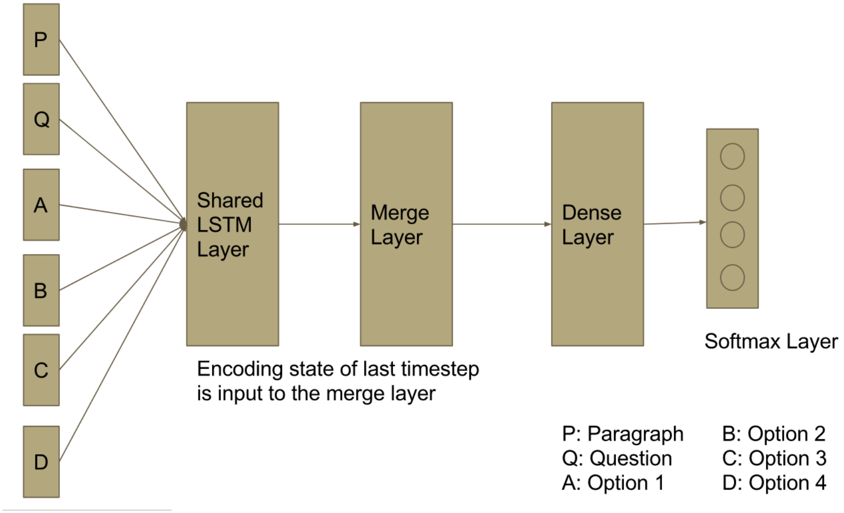


Figure : Sentence embedding architecture

Eliminating Options

- When humans solve multiple choice questions, we also answer questions by eliminating some choices.
- The neural network does well at eliminating one incorrect answer. On an average, 86 times out of 100.
- Could possibly ensemble models where :
  - we eliminate this option and use the same cycle for the remaining options.
  - Information Retrieval system on the remaining 3 options.

.

Ad-hoc approaches

- Improvements with IR module
  - Retrieve - Most relevant 15 documents - Around 1.5-2k words.
  - Split into smaller documents.
  - Indexing Again on these documents
- Improvements with Embedding Module
  - Retrieve - Most relevant 15 documents - Around 1.5-2k words.
  - Train the embeddings on these sentences.

.

One vs all classifier

Classifier trained with the following features for all the 4 options:

- Word2vec Similarity between question and option:
  - With POS tagging and Stemming
  - Without POS tagging
- Similar Question Answer Score
- Information Retrieval Score on the Corpus
- Negative Question
- Percentage of Unique Tokens in Questions and Options.

Ensembling by Ranking Options

- Similar Questions
- Information Retrieval Score Ranks
- Word2vec Ranks
- Using Particular Module for specific questions
  - Deciding Criteria : Difference between the first and second best answer was greater than a threshold

Results

Accuracy was evaluated using categorical accuracy i.e. the fraction of multiple choice questions answered correctly.

Model	Accuracy
Word similarity	0.27-0.34
Neural Network	0.3
Information retrieval	0.46
Ensemble	0.49

Table : Results

Questions?

Thank you!

FlexNet

- Currently, in its nascent stage, our library aims to remove the constraints of building complex architectures by unpacking the different elements of a neural network (eg - layers and the interconnecting weights). This gives the end users more flexibility to build new models easily.
- Programming Language : Python, Backend: Theano

Architecture

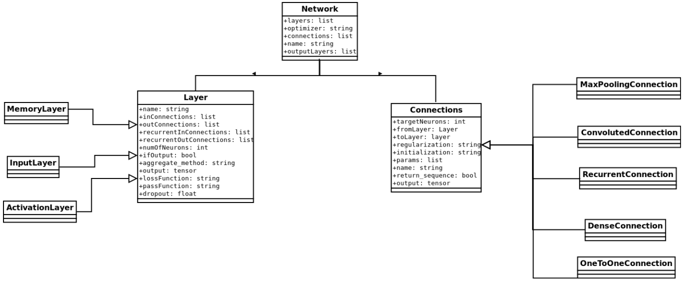
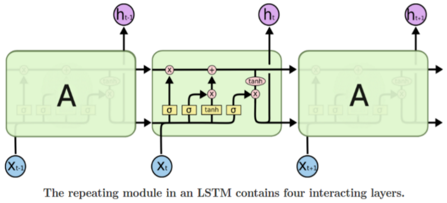


Figure : Flexnet architecture

Our Approach

- Other frameworks have weight matrices associated with the layers.
- But, in true sense, the weight matrices are the properties of the connections between the layers.
- Hence, we have tried to separate out the concerns and have different classes for layers and connections.
- So, in our framework, an end user defines different layers and then the connections between them.
- FlexNet not only makes it easier to create DAG structured networks as in Keras but also offers several advantages over Keras which are discussed in the further slides.

How is FlexNet different from Keras???



h_{t-1}
h_t
h_{t+1}
A
A
σ
σ
tanh
σ
tanh
X_{t-1}
X_t
X_{t+1}
The repeating module in an LSTM contains four interacting layers.

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

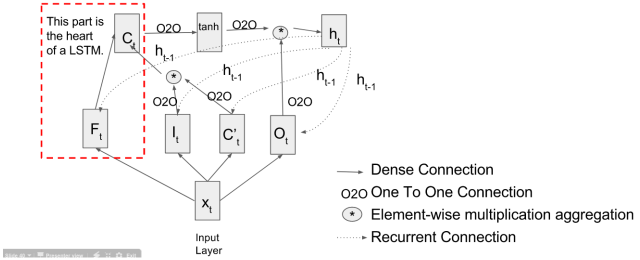
$$o_t = \sigma\left(W_o \ [h_{t-1}, x_t] \ + \ b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

Figure : Standard LSTM architecture

LSTM in Our Framework

Other frameworks give an out of the box implementation of standard architectures like LSTM but tweaking them is quite difficult and forces the user to build the architecture from scratch. The heart of a LSTM lies in the memory unit and the way in which this memory unit gets updated at each time step. In our framework we have unpacked these fundamental elements into primitive entities (eg. memory layer). This separation of concerns gives the user the flexibility to play around with any creative variations of such standard architectures.

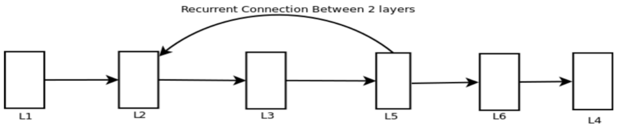


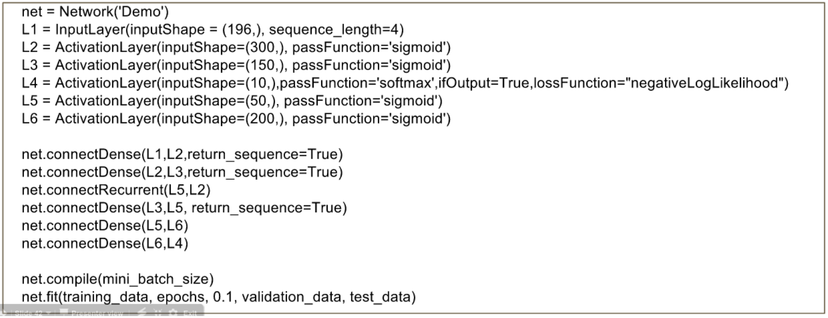
Recurrent Connections

- Conventional recurrent connections are made from a layer to itself. In our framework, the user is given the flexibility to create a recurrent connection from one layer to any other layer.
- Suppose we have a temporal input in the form of a video and the network has a stack of three convolutional layers. Flexnet allows a recurrent connection between two convolutional layers, say from third to first. Therefore, the first convolutional layer can look at the feature map of the previous time step of the third convolutional layer.
- To the best of our knowledge, neural network architectures with such recurrent connections haven't been explored earlier but could be a potential research exploration in the future.

Using our API - Example

To the best of our knowledge, building a neural network as shown below is not possible in Keras:





```
net = Network('Demo')
L1 = InputLayer(inputShape = (196,), sequence_length=4)
L2 = ActivationLayer(inputShape=(300,), passFunction='sigmoid')
L3 = ActivationLayer(inputShape=(150,), passFunction='sigmoid')
L4 = ActivationLayer(inputShape=(10,),passFunction='softmax',ifOutput=True,lossFunction="negativeLogLikelihood")
L5 = ActivationLayer(inputShape=(50,), passFunction='sigmoid')
L6 = ActivationLayer(inputShape=(200,), passFunction='sigmoid')

net.connectDense(L1,L2,return_sequence=True)
net.connectDense(L2,L3,return_sequence=True)
net.connectRecurrent(L5,L2)
net.connectDense(L3,L5, return_sequence=True)
net.connectDense(L5,L6)
net.connectDense(L6,L4)

net.compile(mini_batch_size)
net.fit(training_data, epochs, 0.1, validation_data, test_data)
```

Work In Progress

Another advantage of the framework is the flexibility of using the recurrent output of any previous time step and not just the last time step of a particular recurrent connection. This kind of a recurrent connection might be useful in scenarios where the user has some prior information on a fixed temporal dependency existing in the data.

Questions?