Kungliga Tekniska Högskolan

DD2112 Speech Technology
Course project

Daniel del Castillo
Nicolas Jonason
Jonas Ivarsson
Quentin Lemaire

June 4, 2018

Abstract

This project aims at building a chatbot based on the data collected from the AskReddit subreddit [1]. The idea is to train a generative seq2seq sentence-level model implemented in Keras to learn how to answer arbitrary questions, based on the top answers and posts found in the selected subreddit.

1 Introduction

1.1 Background

RNNs are a type of neural network which have the useful property that they can handle variable length input sizes. LSTMs are a variant of RNNs designed to remember longer-term sequence level context. We have previously had good results in a classification task (many to one) using LSTMs. In recent years, researchers have used these networks in machine translation with good results [2]. We hypothesize that we can use the same architecture for a open question answering task. This supposes that the network will accumulate world knowledge in its weights that will enable it to answer new questions somewhat coherently.

1.2 Goal and Motivation

In this project, we intend to implement a chatbot whose sole function is answering questions. To achieve this, we will use an RNN trained on titles and responses in the 'AskReddit' subreddit. Our goal is to try to find a method to teach an LSTM (or a stack of them) how to give sensible answers to very diverse questions, keeping semantic and context information related to the question.

We intend to use pretrained GloVe word vectors [3] as our word representations so that our model can capture the semantic and syntactic relationships that they contain.

2 Data collection and cleaning

2.1 Data collection

To collect the data to train our model, we decided to scrape ourselves the subreddit. Reddit offers an API to directly scrap their website but they put a limit: we can only search for the top 1000 posts or the 1000 latest posts. This was a big issue because in order to train efficiently our model we would need a lot more.

To overcome this issue, we used an open-source API called Pushshift [4] which allow us to do collect data posts from a specified period of time. With the result of a request to this API, we can get a list of post IDs and then we can use the reddit

API to get the comments of this post. Each request to the Pushshift API allows us to get 500 IDs and we can go back in time to the beginning of the subreddit, more than 10 years ago.

Then, we choose a number of post to scrape (50k in our case), we divide make a division of 10 years in order to have the right number of post in periods of 500 posts, and we do a request to get the IDs for each of these periods. We decided to get the top 500 posts for each period in order to have questions of good quality; this could have an impact on our network later.

Now that we have all the IDs, we need to do a request for each of those IDs to get the corresponding answer so the question. Again, to have data of good quality, we choose the top comment for the question. Therefore, to get our 50k messages, with approximately 1s per request, we would need more than 50 000s to scrap all the data, it is more than 13 hours.

To optimize this process, we used the Scrapy library in Python [5], it allows us to parallelize each request and to win a lot of time. With this library, it takes around 1h30 to get every questions and answers.

2.2 Data cleaning

After having scraped all the data, we need to clean it because sometimes the answers are just links, images or too long comments. Moreover, to use the pre-trained GloVe word vectors we need to remove some punctuation, convert to lower case, etc...

So we did a lot of cleaning for each post:

- We removed the posts with an answer more than 50 words long.
- We removed the posts with a link in the comments.
- We removed the posts with a deleted answer.
- We removed the tags in the question.
- We removed some punctuation to use the pre-trained GloVe word vectors
- We put every text in lower-case.

After all this, we had removed around half of the posts and we still have 22k Q&A left. After that, the data is cleaner, in adequacy with our GloVe word vector and will produce better results.

3 Model building

3.1 From RNN to LSTM

The usual way to generate a sentence with a network is to use multiple time a network generating words. The output of the network is used as input to generate

the next word. Moreover, to keep track of the previous words, the network passes information to itself, for example to know that the next word should be a verb.

The following figure represents such a network:

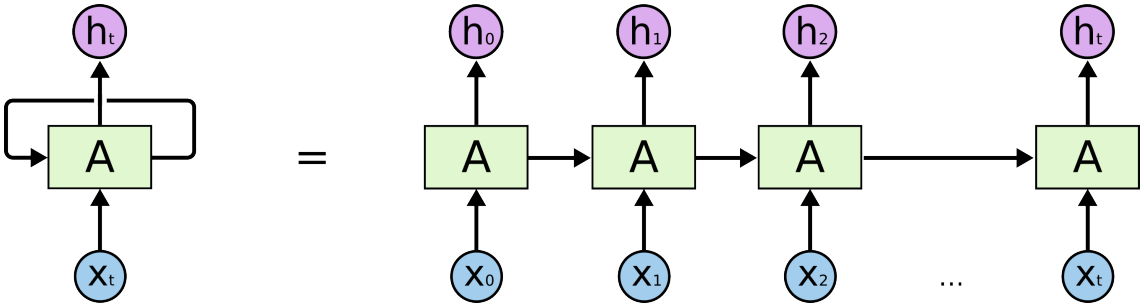

Figure 1: Representation of a RNN. Source: [6]

This model can produce grammatically correct sentences but suffers a lack of long-term memory because the memory is rewritten at every step. For example, the network could have difficulties to complete this sentence "I studied in Sweden for 5 years, I now speak fluent ..."; the next word is obviously Swedish, but the network might not have kept the context information "Sweden" in its memory.

To overcome this issue, we used a LSTM (Long Short Term memory) network. This model is illustrated in the following figure.

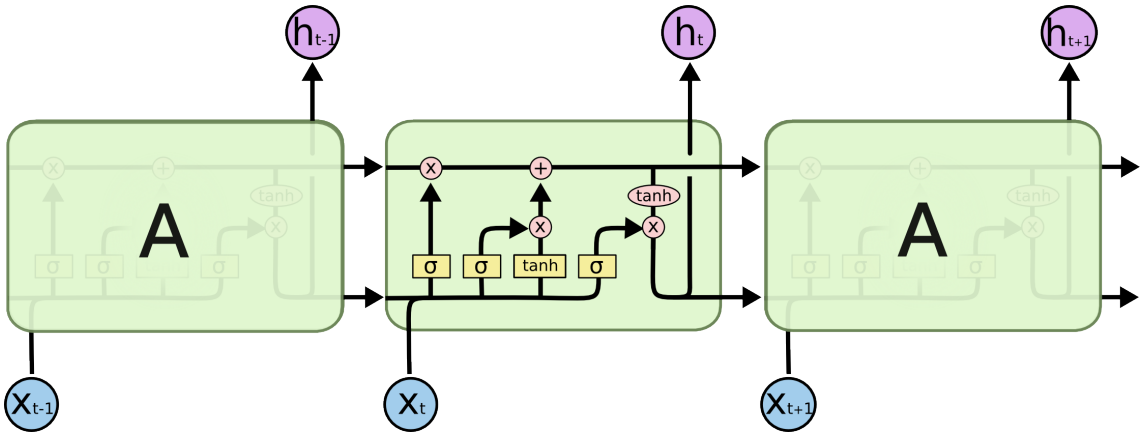

Figure 2: Representation of a LTSM network. Source: [6]

This network is used the same way. The output will be used as input to produce the next word. But now, we can see two links between the network and itself.

The bottom link is similar to the RNN one, it is a short-term memory rewritten at each step. The top one is a long-term memory and will not be rewritten every time. It will be selectively rewritten at each step in order to allow meaningful information to remain longer.

With this new model, we should be able to deal with our problem and answer correct sentence which correspond to the question. We now need to interest ourselves to the architecture we chose and how to put a whole sentence as input.

3.2 Model architecture

To understand the question with our network, we need to pass the question through the network in order to build the long term memory containing all relevant information. Then, we use this long term memory to produce the words.

This method is illustrated in the following figure:

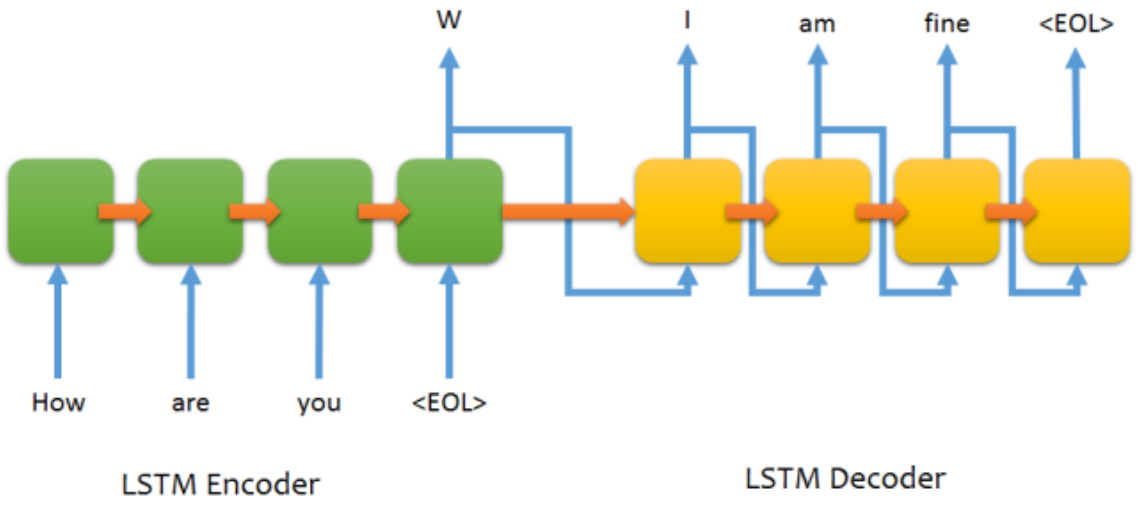

Figure 3: Encoder-decoder LSTM. Source: `https://github.com/farizrahman4u/seq2seq`

The network is divided in two parts: the encoder and the decoder. In fact, instead of training a single LSTM, encoder and decoder are separate LSTM networks, closely connected but trained for different purposes. The encoder's goal is to learn how to abstract enough context information and semantic-syntactic relationships from an arbitrary input question and encode them into a cell and hidden state pair of vectors. On the other hand, the decoder's main purpose is to, given the state vectors from the encoder, produce, step by step, a syntactically coherent sequence of words that serves as answer and is related in context to the question. The first word of the answer sequence is a key element, since it will be fed as input of the decoder in the second timestep. This way, the decoder will hopefully learn to find the next suitable subsequent word within the answer sequence, given each previous word. The answer sequence ends when the decoder chooses a special 'stop sequence' word as the most suitable next word.

Finally, the words will not be directly put in the network as strings but as GloVe word vectors, as we will see in the next part. Therefore, the network will have two more important components: a pre-proessing one at the beginning to encode each word to a GloVe word vector, and one at the end to decode the GloVe word vector back to a word (the raw output of the network will not be a string but a GloVe word vector).

3.3 GloVe word vectors

Words in represented as strings does not contain any semantic information for a computer and thus we need a better representation to input into the model.

Fortunately this can be solved by training a mapping from a vocabulary to a hyperspace where each word is represented as a vector in that space. With this method semantic content of a word can be represented in the input to the network.

We have decided to use the GloVe word vectors. "GloVe is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space." [3]

Therefore, GloVe will project the word in a high dimensional space and words with close meaning will be close to each other. For example, it will be possible to find a subspace composed of all the fruits or another with of the rivers of the world.

One of the interesting features of the GloVe word vector is that it can keep the meaning of basic arithmetic operations. For example in the GloVe space, Paris - France + Italy = Rome or King - Man + Woman = Queen.

This method is using another network which also needs to be trained, but we will use a pre-trained network to skip this part. It has been trained on 42 billions words, has a vocabulary of 1.9 million words and maps a 300-dimensional vector to each word. [7]

3.4 Implementation of the model

For the model implementation, we used Keras. Keras is a framework for implementing neural networks and it is built to work on top of previous frameworks such as TensorFlow and Theano.

We needed to obtain results as soon as possible, and it has been necessary to iterate many times, modifying our models from the initial ideas. In the end we decided to adapt the model presented in [8], due to the similarity between this model and the ones we had originally devised. This model was designed as a solution for the task of machine translation and works as a character-level language model. Therefore, we needed to convert it into a word-level seq2seq model, making also small modifications to fit our problem requirements. Among these modifications, the most important might be that, while the seq2seq translator in [8] relies on a one-hot encoding representation of the outputs, we wanted our LSTM decoder to predict each word as an embedding vector, in the same format as the inputs, eliminating then the need of a final softmax layer. Instead, our last 'Dense' layer maps each output of the LSTM decoder to a d-dimensional vector, being d the size of the word embedding vectors used as inputs. Once we obtain all the predictions as condensed vectors of features, we compute similarity measures in the embedding space, such as cosine proximity, to find the closest words to our outputs in the vocabulary of known words.

4 Model training and validation

4.1 Division of data

To train the network both the input and label data need to be divided into training, validation and test batches. The training data needs to comprise the largest set followed by the validation set and the testing set can be left much smaller. This is because of how the training is structured.

4.2 Training

The training of the model is carried out in epochs. For each epoch the model is fit optimally according to the training set. Then at the end of the epoch the validation set is used to select which model performs best on the validation set. After all the epochs are run through the test set is used to evaluate the ready trained model on data it has never seen before.

4.3 Loss function

In order to update the weights of the network at each run through of input to output, the output is compared to the labels using a loss function. This is a measure of how successful a prediction is. This information is then back propagated into the network to tune the weights to perform better. Finding the right loss function to use is a matter of finding a suitable measure of proximity between the model output and labels.

The question whether two vectors in a 300 dimensional space are close is not trivial. What distance means in these dimensions is a bit unclear. The naive approach would be to use the Euclidean distance, but as discussed in [3] the semantic similarity of words are much better represented by their similarity in angle.

A commonly used measure of word similarity is the cosine distance between two words. This is computed as the dot product of the two word vectors divided by the product of their magnitudes.

For recovering word sequences from the network outputs, we used Canberra distance [9].

4.4 Not so viable alternative network with fixed output vocabulary

We experimented with a different output configuration that would output words directly (without an intermediary word vector representation). In this setup we instead trained the network with the words one hot representation as output labels. For this, we kept a record of all the words contained in the answers (we could in theory have used any fixed length vocabulary) in a list V. For a given question-answer

pair, the output label consisted of a sequence of vectors, each of length V consisting entirely of zeroes except at the index corresponding to our word. This way, the output of the trained network would correspond to a word directly. The obvious limitation of this model is that we can no longer output words that were never seen during training. In the previous configuration, our model wouldn't output a word per say, but a position in semantic space which we would then assign the closest known word. However, a words one hot encoding of a word is contains no semantic or syntactic information. When using one hot encoding of words as labels, we can use cross entropy as our loss function. Although the encoder works the same, using one hot encoding as outputs dramatically increases the number of weights in the output layer. In the dense output layer, where we previously had H * D parameters (where H is the size of the hidden layer and D is the number of dimensions of our word vectors) we now have H * V parameters to train (V is the size of our vocabulary). This astronomical number of parameters makes the network orders of magnitude slower to train. Unless we diminish our vocabulary to a fraction of its original size, we cannot train this network. The naÃ¯ve approach would be to simply remove all but the n most common words. This would result in intelligible sentences with much of the topical information missing. There are other smarter ways we could diminish the size of our vocabulary. For instance, we could remove stop words altogether and attempt to reconstruct sentences from the words in the output (altough this is a difficult task onto itself). Also, we could cluster words with similar meaning as to reduce our vocabulary further. We did not attempt any of these vocabulary shrinking methods.

5 Results

Although we have not achieved the expected results, we believe they contain insights that explain some of the main limitations of our approach.

Our first meaningful results showed that the model had a strong inclination for generic words. Our model often output answers consisting only of *i* or *you*, or other semantically-generic words like *the*, *also* or *people*.
Going from the word to the sentence level, our best results showed that the model was capturing some structure or length sense in its answers, while missing meaningful content. Here are some examples:

- if the word girl on every song was replaced with horse , what songs would end up having the most different meaning ?
  - ✗ i not going, i going, but not going
  - ✓ id listen to barbie horse

- people in their twenties and beyond , how common is it to remain friends with people from high school once you leave ?

✗ i not going, i going, but not going, i not going, i going, but i going, but not going

✓ in general ? highly unlikely . i have two individuals from high school i see a couple of times a year . the rest are facebook acquaintances , but hardly considered to be friends .

These results show the tendency of the model to produce unspecific answers. As we discovered later, this is one of the main issues of conversational models [10], [11]. Uninteresting and unappealing conversations span over AI research chatbots because of the common practice of favouring answers that are conversationally valid for a wider range of questions. Along with this issue, lack of a consistent personality and a long-term memory seem to be some of the main challenges confronted by general conversational models [12].

6 Discussion

After many iterations without fruitful results, we started to wonder if the goal we had in mind was reachable at all with the resources we had. To our surprise, we found a lot of references that point out the weaknesses of the Seq2Seq approach to solve the contextless QA chatbot problem. Here we discuss the insights provided by some of these sources, combined with our own, in a constructive attempt to learn a take-home lesson from this work.

It was astonishing for many AI researchers and enthusiasts how variable and rich were the sample conversations published in [13]. Apparently, we could apply a standard Seq2Seq architecture for machine translation to create a powerful conversational model. We hypothesized that with a big enough dataset of QA pairs, we could obtain similar results. In turn, our Seq2Seq model yielded results far worse than expected.

We believe that one of the main weaknesses of our setting lies in the chosen dataset. By taking a quick look at the QA pairs collected, one can get a notion of the high variability and lack of consistency of the inputs. Along with a great presence of satire, irony and sarcasm, these properties hinder modelling semantic and syntactic patterns in the data.

Furthermore, the challenge of generating a reply word by word and not pointing to a ready sentence seems to lie in an accurate contextual representation of entire documents (questions/answers). These are needed in order to determine if you have hit the mark with your prediction and without them it is impossible to train a network.

It is an open research question how to determine if something is a "good" answer to a question. The most common attempts we have found are to take the average of the word embeddings in a sentence and more ambitiously weighting the words using different techniques based on either token frequency in the vocabulary or word classes. These perform better but still does not solve the problem. Another approach

is to try to train sentence embeddings in a similar manner to word embeddings. While this is an interesting idea consider training word embeddings was done on enormous datasets. The size of the domain of possible words is much smaller than the domain of possible sentences so the required data to do this would be orders of magnitudes larger.

Consider as well that even if we had perfect document embeddings representing questions and answers in a semantic space we still have the problem that there exists a very large set of acceptable answers to every question (if not infinite). Any algorithm based around minimizing loss will then optimize towards the safest options like "I don't know", "I am not sure", which is what we see in [11].

References

[1] AskReddit Website. `https://www.reddit.com/r/AskReddit/`.

[2] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

[3] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[4] Jason Michael Baumgartner. Pushshift API. `https://github.com/pushshift/api`.

[5] Scrapy documentation. `https://scrapy.org`.

[6] Christopher Olah. Understanding LSTM Networks. 2015. `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`.

[7] Pretrained GloVe word vectors. `https://github.com/stanfordnlp/GloVe`.

[8] François Chollet. A ten-minute introduction to sequence-to-sequence learning in keras. `https://blog.keras.io/a-ten-minute-introduction-to-sequence-to-sequence-learning-in-keras.html`.

[9] Godfrey N Lance and William T Williams. Mixed-data classificatory programs i - agglomerative systems. *Australian Computer Journal*, 1(1):15–20, 1967.

[10] Jiwei Li, Michel Galley, Chris Brockett, Jianfeng Gao, and Bill Dolan. A diversity-promoting objective function for neural conversation models. *arXiv preprint arXiv:1510.03055*, 2015.

[11] Iulian Vlad Serban, Alessandro Sordoni, Ryan Lowe, Laurent Charlin, Joelle Pineau, Aaron C Courville, and Yoshua Bengio. A hierarchical latent variable encoder-decoder model for generating dialogues. 2017.

[12] Saizheng Zhang, Emily Dinan, Jack Urbanek, Arthur Szlam, Douwe Kiela, and Jason Weston. Personalizing dialogue agents: I have a dog, do you have pets too? *CoRR*, abs/1801.07243, 2018.

[13] Oriol Vinyals and Quoc V. Le. A neural conversational model. *CoRR*, abs/1506.05869, 2015.