Exploring Recurrent Neural Network variants for Question-Answer matching[*]

Giona Paolini

December 7, 2018

Abstract

One of the major problems for Question-Answer Systems (QA-Systems in the rest of this paper) is related to the understanding of natural language. Such systems might have hard time finding answers to questions when the latter contain typos, noise (irrelevant information), or it is posed in a complicated way (complex terms). This paper focuses on the classification of user input for QA-Systems. We used data from an active QA-System in order to train a Recurrent Neural Network(RNN) model that is able to replicate the system's behaviour of matching a question with an answer. We experimented with different types of embeddings (Corpus specific vs Static GloVe vs Dynamic GloVe), and compared with different model variations (Attention, Bidirectional + Attention). The results shows that the combination of a Bi-Directional RNN with Attention Mechanism using dynamic GloVe embeddings has the best performance on the task (93.2% accuracy). Moreover the model removes the restriction of a binary outcome (match found and match not found) by providing confidence on the predictions, increasing further the accuracy when considering multiple outcome.

1 Introduction

In recent years QA-Systems have become an essential component in the field of customer service in which the main objective is to retrieve point-to-point answers by recognizing questions asked in the user native language[1], allowing users to get quick first support whenever they need. QA-Systems can be divided in 2 broad categories: open domain and closed domain. As the name suggests, open domain systems deal with questions that could possibly be related to anything. On the contrary, close domain systems deals with questions under a specific domain, and they are considered an easier task in the field of NLP as it hugely shrinks the amount of information that needs to be processed for the task[1]. Furthermore, we can distinguish QA-Systems by the way in which they retrieve the information and extract answers. This research focuses on knowledge based QA-Systems in which the following process apply:

Normalization
: The question is normalized using NLP techniques.

Matching
: The question is matched with an entity called Knowledge Base Article (for simplicity called KBA for the rest of the paper)

Answering
: The KBA might use some other context information to provide an answer (e.g. It might provide a different answer if the question was asked from a mobile phone rather than a computer).

One of the existing problem related to this kind of systems concerns the matching phase. Such systems might fail to match questions that for one or more reasons are normalized in the wrong way (e.g. the questions contain typos and/or irrelevant information).
This research approaches the problem as a multi-class sentence classification with the application of Word2Vec Embeddings and RNN variants, in which the objective is to predict labels that correspond to the KBA ids of a QA-System. The aim of the research is to answer the following research question:

[*]This thesis was prepared in partial fulfillment of the requirements for the Degree of Bachelor of Science in Data Science and Knowledge Engineering, Maastricht University, Supervisors: Gerasimos Spanakis.

1. How different is the performance of the model when we use static pre-trained word embeddings (GloVe) rather than embeddings trained on the corpus of our dataset?

2. Does the performance increase if we let the pre-trained embeddings change during training?

3. Does the accuracy improve when we introduce Attention Mechanism?

4. Does the accuracy improve when we substitute the RNN with a Bi-Directional RNN?

In the Background section we discuss the techniques that are used and it is divided in two main topics: we first talk about what word embeddings are and why they are useful in this research, and then we discuss RNNs models and their variations. Then we have a Methodology section in which we introduce the dataset used and the proposed model, where we explain how the discussed techniques were applied for the task. In the experiment section we analyze the results of 5 different setups: the first 3 setups regards 3 different strategies for word embeddings (Corpus specific vs Static GloVe vs Dynamic GloVe), while the last 2 are experiments with regards to the Attention Mechanism and the Bi-Directional variation of RNNs. Finally we have our conclusion where we further discuss the results as a whole together with possible future improvements.

2 Background

2.1 Word Embeddings

In order for a computer to understand sentences, words need to be represented by some kind of embeddings. There are several ways in which this can be accomplished. The easier way is to use the so-called one-hot encoding [2]. When we apply one-hot encoding to our words, we create *n* vectors of size *n* where *n* is the size of our vocabulary. Each of the vector is filled with 0s except for the index that corresponds to the word in the vocabulary, that is set to 1. For example, suppose we have our vocabulary v = ["Cat","Dog","Door"], then the word embedding for each word would be [1,0,0], [0,1,0], [0,0,1] respectively. One of the problems with this type of embedding is that the space required to store the embedded words increases exponentially as the size of the vocabulary grows. Also, it encodes no properties or similarities: the words "Cat", "Dog" and "Door" in the previous example have the same cosine similarity equal to 0.

2.1.1 Word2Vec

In 2013 Mikolov et. al [3] presented in their paper a new model called Word2Vec that solves the above mentioned problems: a Neural Network capable of learning relationships and properties of words. There are two main implementations of Word2Vec. One is the Continuous Bag Of Word (CBOW), that given a list of words, it tries to predict the central word. On the contrary, the other implementation is called Skip-gram and tries to predict nearby words given a central word (See Fig 1). While they differ conceptually,

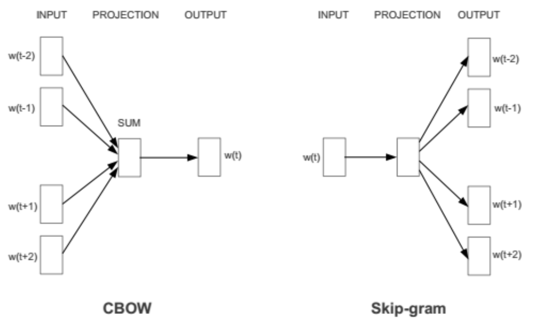

Figure 1: Difference between Continuous Bag-Of-Word model and Skipgram Model [3]

the model is the same for both: it includes an input and output layer with size *n* (where *n* is the size of the vocabulary) where the fed inputs and targets are one-hot encoded, and a single hidden layer of size *m* (where *m* is the size of the embedding vector). What is interesting about this model is that once the training is carried on for long enough, we are only interested in the weights between the input layer and the hidden layer (W in Figure 2), because they will encode useful properties and relationships among all the vocabulary[3]. For example we could perform vector operations to get similar words in the same context: if we subtract the word vector "man" from the word vector "king", and then we add the word vector "woman", the resulting vector will be close to the word vector "queen" in the vector space (See Figure 3).

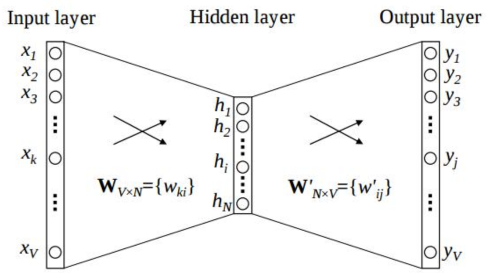
Input layer
Hidden layer
Output layer
x_1 x_2 x_3 x_k x_V
h_1 h_2 h_i h_N
y_1 y_2 y_3 y_j y_V
$\mathbf{W}_{V\times N}=\{w_{ki}\}$
$\mathbf{W}'_{N\times V}=\{w'_{ij}\}$

Figure 2: Word2Vec Structure[4]

2.2 RNN

As we just saw in Word2Vec, Neural Networks are powerful model that are capable of learning complicated nonlinear relationships and are very well suited for finding patterns in a large amount of data. Unfortunately they are not practical when the input data is composed of sequences, main reason being that they lack an important component: memory. RNNs are a variant of Neural Networks that gained a lot of popularity in the field of NLP due to their capability of remembering sequential input [5]. This is how it works: imagine a Neural Network that unfolds in time by feeding the input sequence sequentially at every time-step (Figure 4). It could be considered as a Deep Neural Network, since it is a stack of neural networks, but notice from Figure 4 that the RNN shares the same parameters (V, W, U) across all the steps. Moreover, each of the cell of a RNN store an hidden state (s_t) that is updated with the current input and then passed to the next time-step.

$$s_t = f(Ux_t + Ws_{t-1}) \tag{1}$$

We can see from formula 1 that every cell has its own weights to learn (U and W) and that a non-linear function f (generally tanh or $ReLU$) is applied to the output[6]. This is the hidden state that encodes all the previous input and that in fact introduces memory. Observe that at each time-step it performs a forward pass through the network, this means that every time-step will produce an output. According to the task that we need to solve, we might be interested in all of the outputs or just few[6]. In Methodology section of this paper I explain how both approach were used.

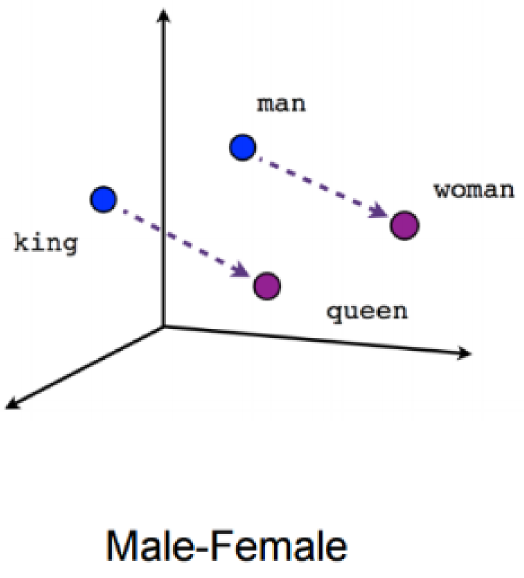

Figure 3: Encoded Vector Relationship.
Source: Tensorflow.org

2.2.1 LSTM and GRU

One of the major problem with plain RNN is that they suffer from the Vanishing Gradient Problem: the gradients tend to get smaller and smaller as we back-propagate through the time-steps[7], meaning that the RNN is not able to remember more than few inputs back. First LSTM (Long Short-term Memory) in 1997[8] and then GRUs (Gater Recurrent Unit) in 2014[9] were presented as a solution to the problem. They changed the structure of the inner cells of RNN to introduce gates mechanisms. For the sake of this research I explain only the GRU cell because it was the chosen option for the implementation. You can still get a gist of how LSTM cells work as they are similar to GRU. First let's introduce the formula used to update the hidden state h_t[10]:

$$z_t = \sigma(W^{(z)}x_t + U^{(z)}h_{t-1}) \tag{2}$$

$$r_t = \sigma(W^{(r)}x_t + U^{(r)}h_{t-1}) \tag{3}$$

$$h'_t = \tanh(Wx_t + r_t \odot Uh_{t-1}) \tag{4}$$

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot h'_t \tag{5}$$

Equations 2 and 3 calculate the outputs of the 2 main gates: the update gate and the reset gate. Both

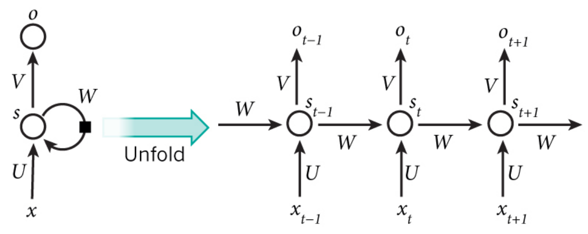

Figure 4: RNN model structure.
Source: Nature

gates will output values between 0 and 1 that work as weights in equations 4 and 5. Equation 4 scales the previous hidden state with r_t: the lower is the output of the forget layer, the less information from the past are combined with the current input. After that, in equation 5 z_t is used to decide how much of the previous step is stored, and conversely the opposite amount is forgotten from the current content (h'_t)[10].

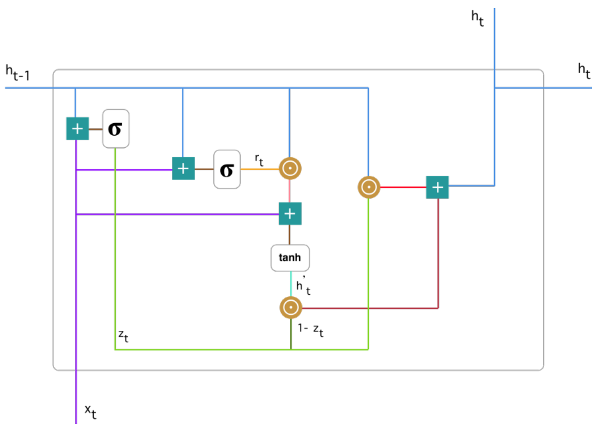

Figure 5: GRU Structure
Source:Towardsdatascience

By learning weights $W^{(z)}, U^{(z)}, W^{(r)}, U^{(r)}, W, U$ during the training, each GRU cell learn how to remember relevant part of previous inputs, and it can be considered as a first level of attention.

2.2.2 Attention Mechanism

Not all the words in a sentence provide the same amount of information, especially in QA-Systems. For example, suppose we are an Internet Provider company, and our QA-system receives the input "Hi, I was navigating on internet while suddenly the download speed decreased, can you help me?". In this case we are not interested about the user navigating on internet, that is a unimportant detail, what really matters is mostly in the sub-sequence "The download speed decreased". This is what the Attention Mechanism is about, it consists of a layer that takes the outputs of all the time-steps of a RNN and applies weights to them, so that each word has less or more impact on the output according to its importance in the context[11].
Below you can find the formulas for the Attention Mechanism.

$$u_{it} = \tanh(W_w h_{it} + b_w) \tag{6}$$

$$\alpha_{it} = \frac{\exp(u_{it}^\top u_w)}{\sum_t \exp(u_{it}^\top u_w)} \tag{7}$$

$$s_i = \sum_t \alpha_{it} h_{it} \tag{8}$$

In equation 6 u_{it} is just the hidden representation of h_{it}. In equation 7 the importance of the words is calculated as the similarity between u_{it} and the context vector u_w (that is initialized randomly and learned during training), normalized with a softmax function. Finally the last equation calculates the weighted sum of the vector h_{it} with the weights from α_{it}, that is the sequence vector representing our sentence[11].

2.2.3 Bidirectional

Now let's take a step back to talk again about the structure of a RNN. During the unfolding process through time, the RNN moves sequentially only in one direction, meaning that the cells at any time-step will hold information regarding the current time-step and the state encoding all the previous time-steps. But it holds no information regarding future steps. The sentence "I would like to cancel the subscription" and "I would like to subscribe" are exactly the same until step 4, even though they clearly have opposite meaning. This is where Bi-Directional RNNs come in play. They are composed by 2 RNNs in which the first is fed with the input sequence in normal order, while the second is fed with the inverse sequence[12]. The output from both RNN is concatenated at any time-step and becomes the output of the Bi-Directional RNN (See Figure 6). In this way, BiDirectional RNN are capable of holding information both from the past and the future sentence at any time-step, allowing for more accurate prediction [12]

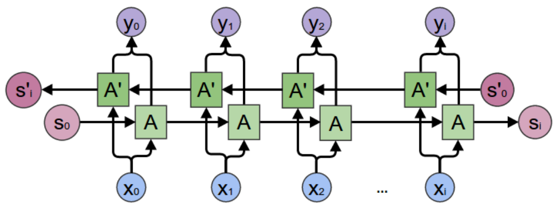

Figure 6: Bi-Directional RNN Structure
Source:Towardsdatascience

2.3 Related work

There are many papers that tackle similar researches on classification, to the best of my knowledge, below I cite some that are most interesting/relevant to this paper:
Wang et. al proposed an Attention-based LSTM in order to perform Aspect-Level Sentiment classification[13]. In few words their model, given a specific aspect, is able to give a score for the sentiment of that specific aspect and is able to locate where in the text the aspect is discussed. Although the number of labels is relatively small compared to our task, it is interesting to see how they let aspects participate in computing Attention weights resulting in a model that achieved state-of-the-art performance on aspect-level classification.

Timmaraju et. al presented another model for Sentiment Analysis using Recursive and Recurrent Neural Networks [14]. They feed a Recursive NN with each sentence of a review in order to encode structural information in a single vector. The vector is then fed into a Recurrent NN that uses the last output to make a prediction of the sentiment. With their model they reached an accuracy close to featured-engineered models such as Support Vector Machine(SVM) for classification of BoW features with Inverse Document Frequency(IDF) weighting. This suggests that the model managed to learn some of the features that were otherwise handcrafted in other models. Even though we feed the RNN word by word rather than sentence by sentence, as showed later in this paper, the application of a Recursive NN might help this research in future steps in making the structure of a sentence more influent in the classification.

Another interesting paper is from Zixiang Ding et. al in which they propose a Densely Connected Bi-RNN LSTM to perform Text Classification [15]. Their main objective is to alleviate the problem of vanishing gradient in deep bi-rnn(up to 20 layers) by feeding each level with the hidden states of all the previous levels. According to the paper the model reached new state-of-the-art results on 3 out of 5 benchmark datasets. Unfortunately there is no information regarding how the added complexity affects the speed of the training/classification, that in our case as mentioned later in this paper, was notably slower when experimenting with a stacked layer of 2 bi-directional rnn.

We now move away from RNN works to see a couple of interesting application of Convolutional Neural Networks on Sentence Classification.
Yoon Kim showed in his research how a simple CNN model improved results on several tasks including question classification mostly by fine-tuning the parameters[16]. Similar to our research, he experimented different strategies with Word2Vec embeddings where he uses static, non-static (dynamic in this paper) and multi-channel embeddings (in which he lets only some of the word vectors change during training). It is interesting to see how the advancement of word embeddings influenced the performance of a CNN model in a NLP task.

Xiang Zhang et. al presented another CNN model that works at character level[17]. They showed it to be an effective method where no words are needed, with the indication that language could be thought as a kind of signal itself. What is interesting about their research is that they observed that the model performed better on datasets where the text was less curated, suggesting that the model, even if there is no explicit evidence yet, might be suited for tasks like ours in which we have typos and errors in our corpus.

3 Methodology

3.1 Dataset

The dataset used for this research is a collection of approximately 268.000 interaction logs of an existing QA-System. Each entry contains 3 fields: input sentence, output sentence, and KBA id. The data is composed by english-only sentences (with few rows in-

cluding other languages). Before we can work with those data, some clean-up is needed. All the rows that are matched with a KBA with less than 15 occurrences are removed, the same applies for rows with input that has more than 100 words. We are left with approximately 267.000 rows, counting 497 distinct KBA ids, with an average of 58 words per output, and an average of 4 words per input.
Figure 7 shows the distribution of the occurrences for the KBA ids. We can see that, apart from few outliers, most of the KBA ids appear between 15 and 2000 times. Then we need to pre-process the input and output text: replacing punctuation, emails, numbers, etc. with tokens (tokenization[18]).

For example, the sentence

"Hi! Can you send me the updates at email@example.com?"

becomes

"Hi <EXCLAMATION-MARK> Can you send me the updates at <EMAIL><QUESTION-MARK> "

It is important to notice that there is no attempt to correct any typo in the input sentences. The reason is explained in the next section when we talk about the proposed model.

3.2 Proposed Model

The proposed model is an Attention-Based Stacked Bi-Directional RNN that uses Word2Vec embeddings to encode sentences. Figure 8 shows a schema of the model. Individual components are described in the subsections below. The model uses cross entropy[22] for calculating the error and Adam Optimizer[21] for the training process.

3.2.1 Embedding Layer

The first level of our proposed model is related to word embeddings. As previously discussed, embeddings are an important part for our sentence classification task. Recall from the previous paragraph that there is no attempt to fix any typo in the input sentences. The main reason is that we want to harness the power of the word2vec model in order to learn similarities between the valid word and its misspelled version. For example, we expect the words "House" and "Hpuse" to be close in the vector space.
We implemented 3 different strategies:

1. Use of embedding vectors trained solely on the corpus of our dataset.
2. Use Static GloVe[19] embeddings
3. Use Dynamic GloVe[19] embeddings (allowed to change during training)

These three strategies will help us understanding how big is the impact of typos in our corpus. It is also important to mention that there are different datasets for GloVe, we used the dataset trained on the Wikipedia Corpus (6B+ words).
The Word2Vec model is implemented using the Skip-gram implementation (it was found it perform slightly better than CBOW), with negative sampling [20], Adam Optimizer [21] and with the size of the word vectors set to 300.

3.2.2 Bi-RNN layer

This is the core of the entire model and where most of the learning occurs. For this layer we chose to implement two stacked bi-rnn instead of a single one, to give more flexibility and power of expression to the model. Each layer has 200 hidden cells (since it is bi-directional, we have a total of 200x4=800 cells). Dropout is also implemented: during the training process, each cell has a 40% chance of being deactivated. This is needed in order to avoid that the model is unevenly trained.

3.2.3 Attention layer

The outputs of all the time-step are then fed into the Attention layer. As explained before, it will output a single vector of dimension 200 (that is the size of the hidden layer of the bi-rnn) that will encode the sentence and the importance of each word in it. In the Background section we discussed that in a RNN we might be interested in all the outputs at any time-step, or just few of them. As you have seen, here in the Attention Mechanism we are interested in all the outputs in order to be able to give each word a weight. But as discussed later in the Experiments section, our baseline model does not implement Attention Mechanism as such we are interested only in the last output of the RNN that will encode the entire sentence without focusing on any particular word.

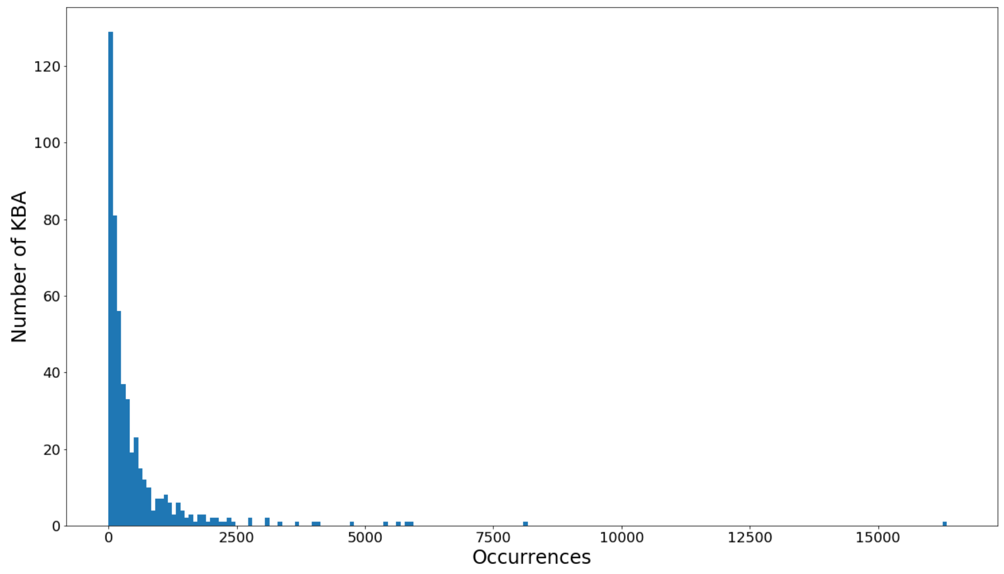

Figure 7: Distribution of KBA ids over the dataset

3.3 Softmax

This layer implement a simple softmax classifier that takes the sentence vector encoded by the Attention layer, and outputs a probability for each of the class we have. For example, in Figure 8 the KBA 56 has the largest probability of being the correct class, hence it gets selected as the result of the prediction.

4 Experiments

The models were implemented using Python 3.7 and Tensorflow 1.10.0. Due to the nature of RNN with variable sequence length, it was not possible to parallelize the training.
For the experiments we have defined 5 setups in order to find what contributes to the performance of the model:

1. GRU RNN with Corpus Specific Embeddings (Baseline model)
2. GRU RNN with Static GloVe Embeddings
3. GRU RNN with Dynamic GloVe Embeddings
4. GRU RNN + Attention with Dynamic GloVe Embeddings
5. GRU Bi-RNN + Attention with Dynamic GloVe Embeddings

All the experiments are based on the same test-set containing 3000 data samples, unfortunately with no k-fold-validation due to time constraint: as we mention later, training speed is an issue especially for the proposed model, and to run a 5-fold-validation on all the models it would take approximately 25/26 days on the available machine. Figure 9 displays some measures of performance for the 5 experiments. The first thing we can observe are the outliers. To better understand them we look at Figure 10 that displays precision, recall and F1 scores according to the distribution of each KBA present in the test set (the scores are related only to the last experiment, we assume the same reasoning applies for the other cases.).
We can see that almost all the outliers fall in the range between 0 and 5 occurrences. This makes sense because the smaller is the number of occurrences the higher is the variance. For example if we have a KBA that occur only once in the test-set, and it is incorrectly classified, then that KBA will have a precision,

Model	Avg Precision	Var	Avg Recall	Var	Avg F1	Var	Accuracy
RNN Corpus Specific	84.8%	4.5%	81.4%	4.3%	82.1%	3.8%	86.2%
RNN Static Glove	88.8%	5.1%	85.5%	5.6%	86.1%	5.0%	90.5%
RNN Dynamic Glove	91.5%	3.0%	90.0%	3.5%	90.1%	3.0%	92.7%
RNN + Attention Dynamic Glove	**91.6%**	2.6%	**90.9%**	2.8%	**90.7%**	2.4%	92.8%
Bi-RNN Attention Dynamic Glove	90.5%	4.4%	89.3%	4.6%	89.5%	4.2%	**93.2%**

Table 1: Performances for each model

recall and F1 score of 0. The same reasoning applies in the opposite way to the great number of 1 scores in the same range. We also notice that there are some KBAs with 0 occurrences. This is because they appear among the predictions, but do not appear in the test-set (hence 0 score).
We continue analyzing the results of Figure 9 and first we notice the improvement from using Corpus-Specific embeddings to using static GloVe embeddings.

Word	Similar Words
booking	unccompained, bookings, etickets, itinerary, kinda, ticker, reservation, expenses
baggage	baggae, bagage, buggage, cake, baggages, handbag, laggege, loggage
luggage	laggage, bags, luggege, deodorant, bagges, articles, increase, bagage
flight	fight, allocate, greta, flght, fligth, luguage, pair, sponsor
checkin	declarartion, check, match%, kilimanfaro, chekin, complains%, servicew, concentrator

Table 2: Words similarities: Corpus only

Word	Similar Words
booking	bookings, booked, tickets, ticket, ticketing, airfare, cancellations, expedia
baggage	luggage, passengers, cargo, handlers, passenger, bags, airport, freight
luggage	baggage, bags, suitcases, suitcase, cargo, passengers, belongings, bag
flight	flights, plane, airlines, pilots, airline, jet, aircraft, passengers
checkin	havin, bringin, nood, missin, waht, tryin, sehr, landet

Table 3: Words similarities: Static GloVe

Looking at Table 2 and 3 we can see that there is a big difference in the resulting vectors: while the word2vec model managed to some extent to learn similarities between valid words and their correspondent typos from the corpus of our dataset, the GloVe pre-trained embeddings represents far better similar words in the same context. So going back to comment on the improvement: I believe that due to the huge difference in training set size (GloVe had 6 Billions word counts vs 17 Millions in our set), the number of occurrence for each typo present in our dataset is too small to make a difference.
The second improvement occurs when we let the GloVe embeddings change during the RNN training:

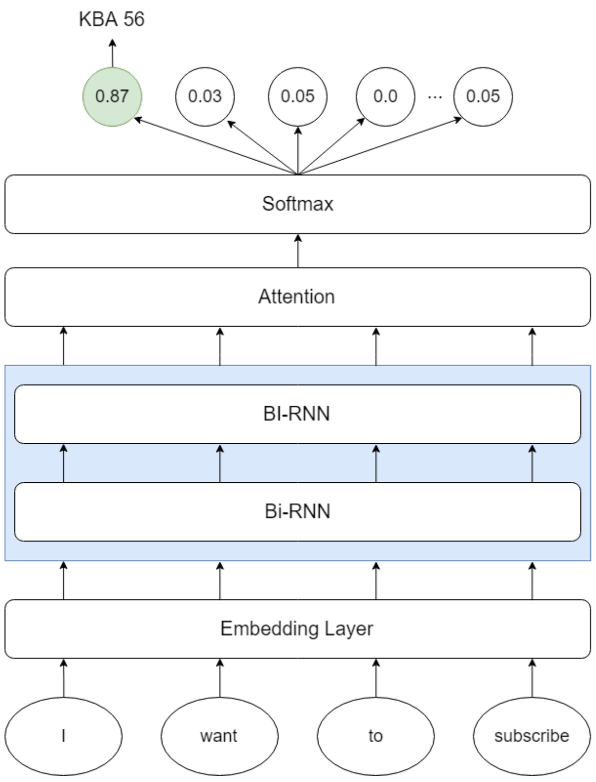

Figure 8: Proposed Model

from Table 4 we can notice that there is no big difference, hence the little improvement is probably due to the fact that the embeddings were reshaped slightly to fit better the context of our dataset.
Notice how there is little to no improvement when we add Attention Mechanism to the model: this might be explained by the fact that the sequences are too short for Attention to harness its full potential. Even though there is no improvement, Attention gives us another tool to visualize the importance of words in the sentences. Figure 11 shows few example results.
The last variant of the model brings a small but noticeable improvement. First we can observe that the median value increased for all the 3 measures. Moreover we notice the F1 Score values are more tilted towards 1 compared to the F1 scores of the other models, suggesting that the overall performance of the the model is the best over the others. This is improvement is probably justified by the nature of the bi-rnn, that as explained before is able to hold information from both past and future of the sentence at all time-steps, hence each word encode more information when passed to the Attention Layer.

Word	Similar Words
booking	bookings, booked, rebooking, reservation, reservations, rental, tickets, billing
baggage	luggage, bag, suitcase , bags, suitcases, cargo, handlers, freight
luggage	baggage, suitcases, suitcase, bags, bag, cargo, belongings, backpacks
flight	flights, pilots, flying, pilot, aircraft, plane, airplane, air
checkin	havin, nood, bringin, chech, tryin, missin, lookin, waht

Table 4: Words similarities: Dynamic GloVe

From Table 1 we can see that although the accuracy is the highest, all the other averages are lower compared to the previous models, this is explained by the fact that the average and the variance are influenced a lot by very low and very high values. In this case 2 things occurred: the first is that the classifier classified correctly more sentences of the same class that occurred often in the testset, increasing the accuracy but not influencing the average precision by a large degree. The second is that it wrongly classified completely a class that occurred only few times, this added a 0 value to the calculation of the averages, hence lowering the value. For this reason, we now continue to analyze the results focusing only on the last model. Table 5 shows some interesting information with regards to wrong classifications. It shows the rank of where the correct label was found. Observe that in 43.1% of the cases the correct label was found to be the second best option for the classification. This is an important result as it shows the potential to improve performance by taking in consideration multiple outcomes of a prediction. To measure this performance we use a technique called Precision@K[23] (Performance@K, since we include more than only Precision scores), in which we consider a prediction correct if

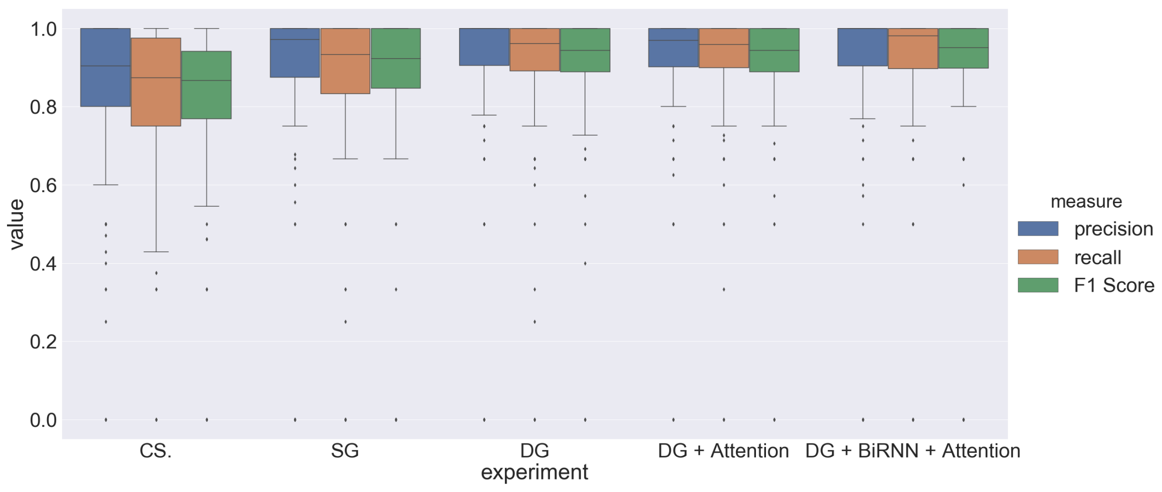

Figure 9: Precision, Recall and F1 Score results for the 5 setups.
CS: Corpus Specific, SG: Static GloVe, DG: Dynamic GloVe

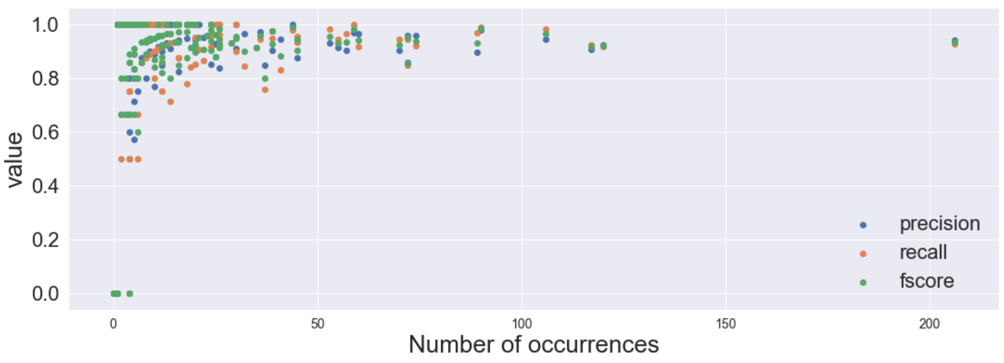

Figure 10: Distribution of performance over occurrences of KBA in test set

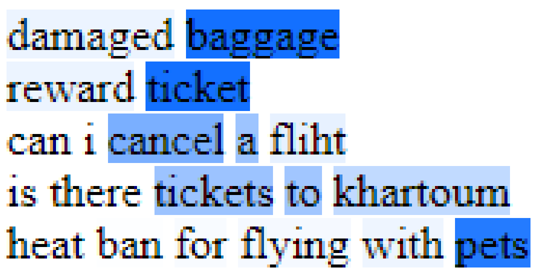

Figure 11: Attention Mechanism visualization

Rank of correct label	Occurrence
2nd	43.1%
3rd	13.2%
4th	7.8%
5th	3.9%
6th	4.4%

Table 5: Ranking of correct label for wrong classification

the correct label is found at the first K position. Results are depicted in Table 6. As expected, we notice already a big improvement at K=2. This is explained, as we saw in Table 5, by the fact that 43% of the wrong classification had the correct label as the second option.
Interesting result to mention is that in the top 5 of the most correctly classified classes there are 2 classes that are also ranked in the top 5 of the most mistakenly classified classes. Specifically the first one in both rank is the same, and this is probably explained by the fact that it is the class that appears the most in the test-set with approximately 40% of the total occurrences. Figure 12 shows how the error of each model decreased over 30 epochs.
It is important to say that while the Bi-RNN reached slightly better results, the training time was between 6 to 7 times slower (Avg 8300 sec for the training time of one epoch against Avg 1300 sec for the other models), this due to the added complexity.

5 Conclusion

In this research we demonstrate how the application of RNN and its variants are able to imitate the matching process of a QA-System. We observed that although the word2vec model was able to learn some connections between correct words and their misspelled version, the embeddings were not good enough compared to GloVe embeddings, due to the difference in size of the datasets they were trained on. So we conclude that using static pre-trained embeddings increases the performance over corpus-specific embeddings, and it further increases if we let them adapt during training. We also saw that the Attention Mechanism, even though it was able to find keywords, it does not improve the performance of the model. Then we see that the BiRNN, even though just slightly, it does improve performances thanks to its nature of looking both at past and future of sentences. Unfortunately we also notice that this added complexity carries a price, making the training and classification approximately 6 times slower than normal RNN. Lastly we saw how the accuracy further improves when we consider multiple outcomes, giving the chance to a QA-System to suggest multiple possible answers increasing the matching score.

When dealing with real QA-Systems that are deployed in the real world, there are other issues that were not discussed in this research. For example the KBA are not static as they might be removed/added/changed, and it makes this model as it is unpractical. Possible future research includes experimenting with multiple instance of simpler models for each KBA: using a sort of divide and conquer approach in which you shrink your problem to a binary classification. This should cut complexity along with the time needed for the classification. Another relevant experiment is to use GloVe embeddings trained on the twitter dataset: given the fact that it is composed by user inputs, the hypothesis is that it should contain more typos and hence more understanding of misspelled words.
Overall, the research showed that the proposed model has potential to open new paths on the way QA-System match questions to predefined answers.

References

[1] R.Mervin, *An Overview of Question Answering System*, International Journal Of Research In Advance Technology In Engineering, 2013.

[2] Rakshith Vasudev, *What is One Hot Encoding? Why And When do you have to use it?*, hacker-

K	Avg Precision	Avg Recall	Avg F1	Accuracy
1	90.5%	89.3%	89.5%	93.2%
2	94.4%	93.6%	93.7%	96.1%
3	95.1%	94.4%	94.5%	97.0%
4	96.5%	96.2%	96.1%	97.5%
5	96.9%	96.4%	96.5%	97.8%

Table 6: Performance@K

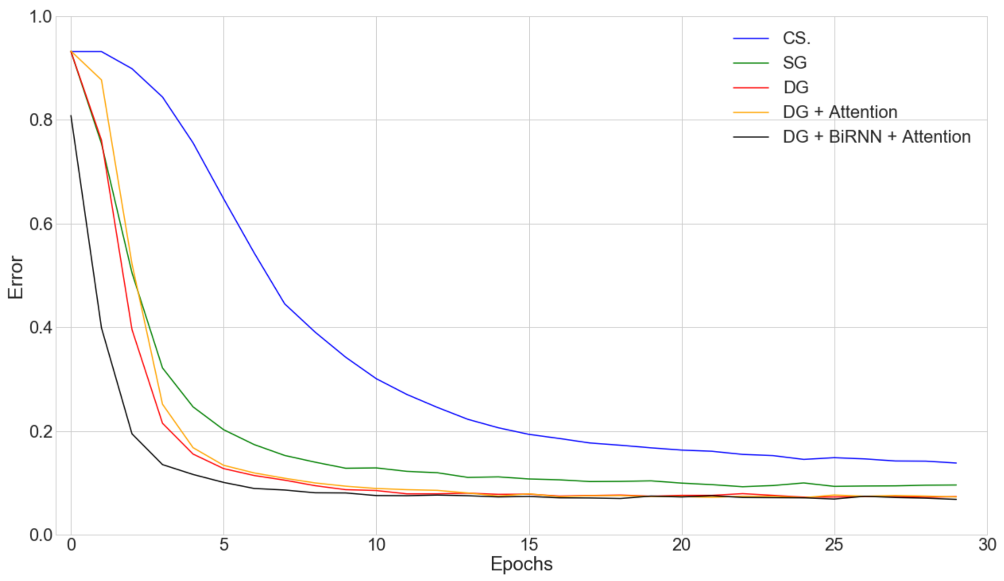

Figure 12: Evaluation Error over 30 epochs
CS: Corpus Specific, SG: Static GloVe, DG: Dynamic GloVe

noon.com, 2017.

[3] Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean, *Efficient Estimation of Word Representations in Vector Space*, CoRR, 2013.

[4] Xin Rong, *word2vec Parameter Learning Explained* CoRR, 2014.

[5] Zachary C. Lipton, John Berkowitz, Charles Elkan, *A Critical Review of Recurrent Neural Networks for Sequence Learning*, CoRR, 2015.

[6] Denny Britz, *Introduction to RNNs*, WildML.com 2015.

[7] Anish Singh Walia, *The Vanishing Gradient Problem*, Medium.com 2017.

[8] Sepp Hochreiter, Jurgen Schmidhuber, *LONG SHORT-TERM MEMORY*, Neural computation, 1997.

[9] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, Yoshua Bengio, *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling*, CoRR, 2014.

[10] Simeon Kostadinov, *Understanding GRU networks*, Towardsdatascience.com 2017.

[11] Minh-Thang Luong, Hieu Pham, Christopher D. Manning, *Effective Approaches to Attention-based Neural Machine Translation* CoRR, 2015.

[12] Mike Schuster, Kuldip K. Paliwal, *Bidirectional Recurrent Neural Networks* IEEE TRANSACTIONS ON SIGNAL PROCESSING, 1997.

[13] Yequan Wang, Minlie Huang, Li Zhao, Xiaoyan Zhu, *Attention-based LSTM for Aspect-level Sentiment Classification* Association for Computational Linguistics, 2016

[14] Aditya Srinivas Timmaraju, *Sentiment Analysis on Movie Reviews using Recursive and Recurrent Neural Network Architectures* CS224d: Deep Learning for Natural Language Processing

[15] Zixiang Ding, Rui Xia, Jianfei Yu, Xiang Li, Jian Yang, *Densely Connected Bidirectional LSTM with Applications to Sentence Classification*, CoRR, 2018.

[16] Yoon Kim, *Convolutional Neural Networks for Sentence Classification*, Association for Computational Linguistics, 2014.

[17] Xiang Zhang, Junbo Zhao, Yann LeCun, *Character-level Convolutional Networks for Text Classification*, CoRR, 2016.

[18] *Tokenization* nlp.stanford.edu

[19] Jeffrey Pennington, Richard Socher, Christopher D. Manning, *GloVe: Global Vectors for Word Representation*, https://nlp.stanford.edu/projects/glove/ 2014.

[20] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, Jeffrey Dean, *Distributed Representations of Words and Phrases and their Compositionality*, CoRR, 2013.

[21] Diederik P. Kingma, Jimmy Ba, *Adam: A Method for Stochastic Optimization*, CoRR, 2014.

[22] Vijendra Singh, *Understanding Entropy, Cross-Entropy and Cross-Entropy Loss* Medium.com

[23] Maher Malaeb, *Recall and Precision at k for Recommender Systems* Medium.com, 2017.