# A brief comparison between Recurrent Neural Networks and Transformer Models in Question Answering
## NLP Course Project

**Chiara Malizia, Johnny Agosto, Marcello Simonati** and **Simone Marasi**
Master's Degree in Artificial Intelligence, University of Bologna
{ chiara.malizia, johnny.agosto, marcello.simonati, simone.marasi }@studio.unibo.it

## Abstract

For our project we tackled the extractive question answering problem with the end goal to create a comparison between different architectures, in particular to compare Recurrent Neural Networks (RNNs) against more performing Transformer models, to identify what are the key features and approaches that enable good performances in this given context. To do so, we tested a set of different models of increasing complexity against the same task, finding out that making use of the attention mechanism, even in a limited form, and sub-word features, are two good approaches to achieve better performances.

## 1 Introduction

As part of our course project for the Natural Language Processing (NLP) course, we tackled a Question Answering (QA) problem which is at the crossroads between NLP and Information Retrieval. In this case, we implemented a form of extractive QA, which deals with extracting the direct answer to a question from a given textual context. This task is very important in a variety of different applications, from automatic information extraction to making knowledgeable dialogue models capable of answering users queries. Our goal was to create a comparison between different approaches, to find out which processes are better suited to solve this type of problem.

The most widely used approaches to this problem given the state of the art are attention based RNN models, such as Bidirectional Attention Flow (BIDAF) (Seo et al., 2016), from which we took inspiration to make our custom RNN models, or Transformer based models, which are fully attention reliant, such as Bidirectional Encoder Representations from Transformers (BERT) (Devlin et al., 2018). Transformer models are better performing, but they need to be trained on huge quantities of data in order to make them usable in transfer learning for other tasks (such as QA), while attention based RNNs are able to be trained in an end-to-end approach for the task at hand, and are much less data hungry.

Our approach to this task was to implement several solutions following an improving logic:

- a simple baseline RNN using the GloVe word-level embeddings;
- some variants of the baseline model, namely using additional features, attention, and a novel character-level embedding;
- a high end reference model using BERT.

We followed this approach since most state of the art solutions are both relying, on various different levels, on both the attention mechanism and sub-word features:

- attention in RNNs is usually used to extract and compare features between two text inputs (e.g. question and context), while Transformer models are wholly based on attention;
- sub-word features in RNNs are usually extracted from character embeddings, often using a pre-trained model like (Kim et al., 2016), from which we took inspiration for our character level embedding model. Meanwhile, usually Transformer models use sub-word tokens as the base for their processing, trained on the specific language needed for the task.

To achieve our comparison we ran a total of five experiments based on RNNs and three experiments based on BERT on the Stanford Question Answering Dataset (SQuAD), in its 1.1 version. The RNN based experiments all share the same data processing pipeline and overall RNN architecture and the same stands true for the BERT based experiments. In each variant we increased the model complexity

a bit, by adding additional features or processing steps, while retaining the same overall architecture.

By comparing the results from our experiments we found out that attention is the main enabler of a good question answering model, that it is possible to mimic the effects of attention using additional features (namely exact match, lemma match and term frequency scores between question and context), that character level embeddings are a useful way to inject additional information and that Transformer models, used in a transfer learning setting, produces performances straight out of the box that are very hard to match.

## 2 System description

For the RNN, the architecture design was loosely inspired by BIDAF. Regarding the code, most of it was our own implementation using keras, exception made for the Highway layers[1].

For the Transformer model, we used Huggingface's BERT implementation for tokenization, model and pretrained weights [2].

### 2.1 Baseline GloVe embedding RNN

In this baseline end-to-end model (see Figure 1), query and context are passed as two distinct integer encoded inputs, which are then translated into the GloVe (Pennington et al., 2014) 300d embedding via a shared Embedding layer using an embedding matrix. Both embeddings are then fed into a sequence of two Highway layers (Srivastava et al., 2015) and processed separately by three different Bidirectional LSTMs with hidden size equal to 300, with the aim to encode context information around each token. The highway network's role is to adjust the relative contribution from the word embedding both in context and in query vectors. The question array is then concatenated to the context array, and the outputs (the start and the end token position probability in the passage of the predicted answer) are then computed by two dense layers separately, one for the start token and one for the end token, using a softmax activation function. We also adopted input token dropout with a rate of 0.1 to reduce overfitting, as we couldn't adopt recurrent dropout in the LSTM layers due to the lack of support of GPU implementation in the Keras layers.

[1] Highway Layer code taken from this BIDAF implementation

[2] BERT implementation we used

### 2.2 RNN Variants

In order to improve the baseline performance and test the impact of different techniques, variants of the baseline model were developed, consisting in using using word features, adding an attention mechanism and finally adding character level embeddings. We tried to keep the variants' architectures as close as possible to the baseline, changing as little as possible, in order to make experiments results meaningful.

#### 2.2.1 Features variant

In this variant (see Figure 2) we used two types of features: term-frequency and exact and lemma match between the question and the context. We added two more input layers to the network for these features and used them to compute an imitation of additional cross-attention between query and context. The final attention scores are computed by adding together exact match, lemma match and term frequency of the context tokens, creating a pseudo attention mask, then multiplying it to the embedded context, and adding the result to the context embedding input. Our thinking behind this is to give more importance to features that appear in both context and query, similarly to how attention would work.

#### 2.2.2 Attention variant.

In this variant (see Figure 3) we took inspiration from the bidirectional attention mechanism that can be found in BIDAF by adding two dot-product attention layers (Luong et al., 2015), using Keras implementation, that computes cross attention between context and query between the first and the second bidirectional LSTM layers, and concatenates the newly computed features to the existing encodings. Our thought process behind this was that the second and third recurrent layers would take advantage of cross-information between query and context.

#### 2.2.3 GloVe + Character embedding RNN

In this variant (see Figure 5) we used both the same word level embedding we used in the previous models and a new Character level embedding we developed (see Figure 4). This character level embedding is based on a Convolutional Neural Network (CNN), inspired by (Kim et al., 2016), trained and fine tuned on the AG News Dataset [3], obtaining an accuracy on test set of about 88% (that we consider

[3] AG's corpus of news articles

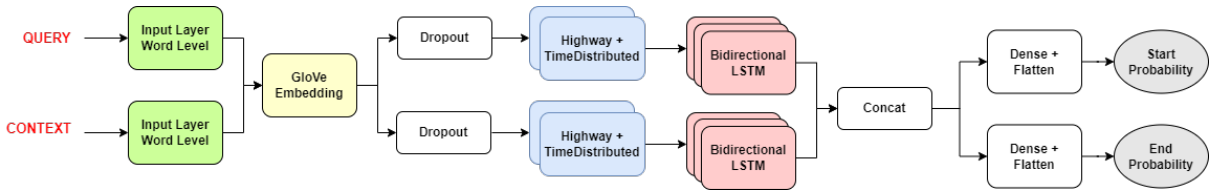


Figure 1: Architecture of the baseline GloVe RNN model

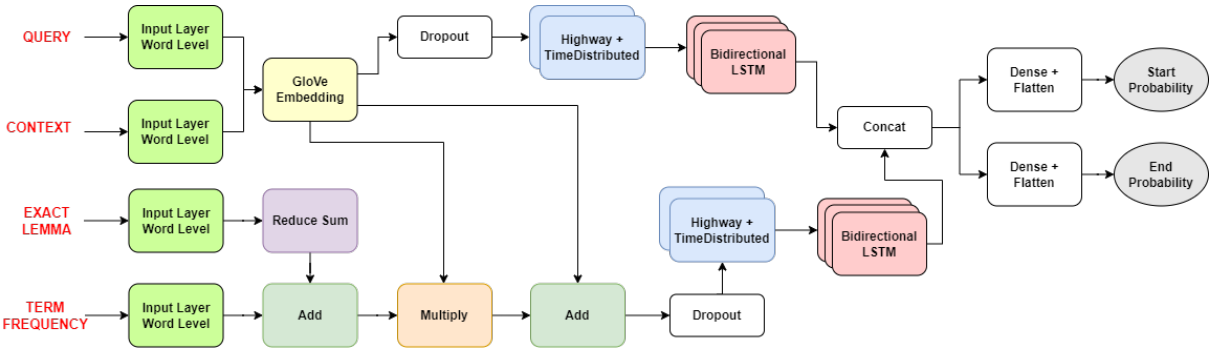


Figure 2: Architecture of the GloVe RNN model with features

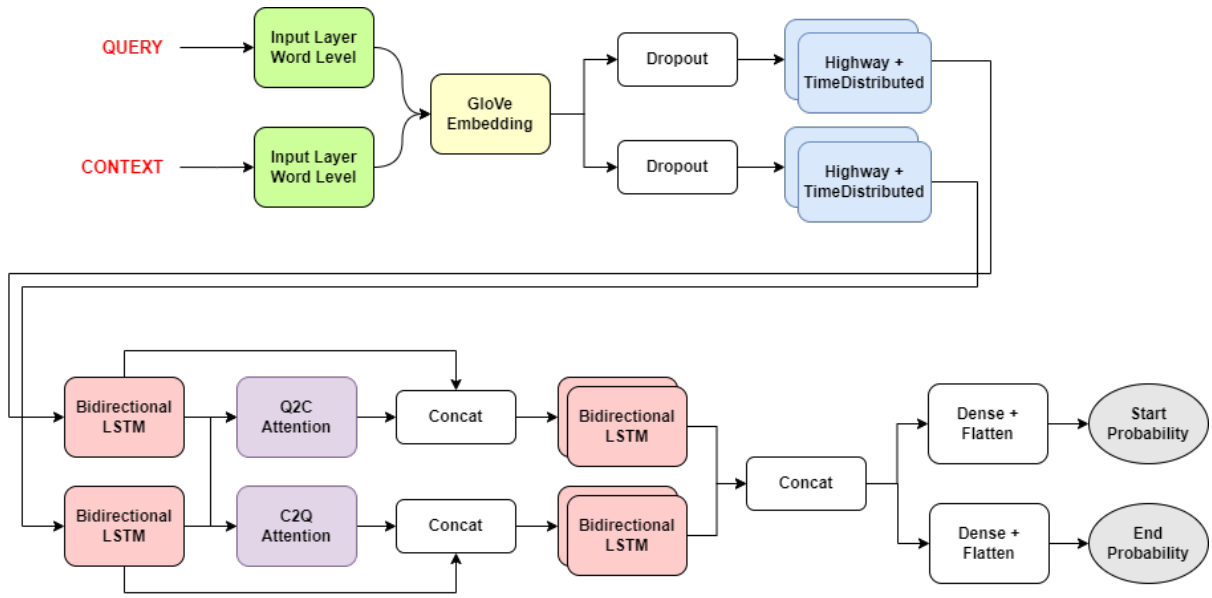


Figure 3: Architecture of the GloVe RNN model with attention

quite good with respect to the state-of-the-art on this dataset achieved using a BERT-based model). The fine tuning consists of making the embeddings trainable after the main training run and reducing the learning rate by a factor of 10. In this way we slightly improve the results.

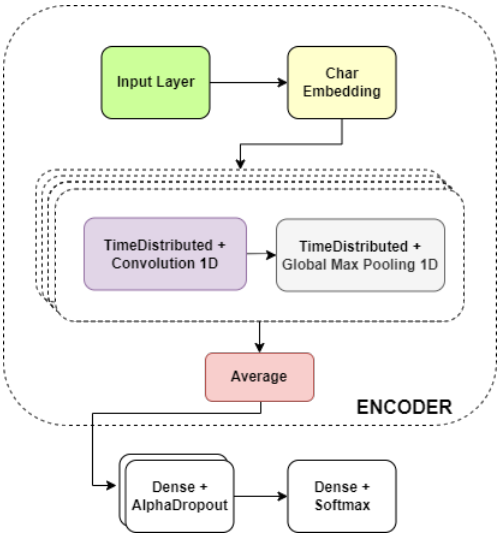


Figure 4: Architecture of the Character Convolutional Neural Network

Then we extracted the encoder of the already trained model, which turns character level tokens into a novel word embedding, and used it as a second set of features in the main model, efficiently expanding the feature set. Both embeddings are then concatenated and passed to a two layer Highway Network to adjust the relative contribution from the word embedding and the character embedding steps. Both question and context embeddings are then finally encoded by a Bidirectional LSTM block.

The following processing steps follow the same logic as the attention variant.

### 2.3 BERT

For this model (see Figure 6), which we took as a high end reference, we used the BERT base uncased pre-trained model in transfer learning as an encoding layer, followed by two separate dense layers with a Softmax activation function as classifiers for start and end token probabilities.

#### 2.3.1 BERT Variants

Additionally to the simple BERT Encoding and Fully Connected classifiers, we also tried to extend the model by adding a Bidirectional LSTM layer between the encoding and the classifier layer, and in a second variant also the additional features computed in the same way as we did for the GloVe based RNN model plus the POS tagging features, which we concatenated to the BERT encoding.

In particular, the features were assigned to BERT tokens by grouping sub word tokens on a word basis and assigning the same feature to all sub word tokens belonging to the same word.

## 3 Data

We ran our experimetns on the SQuAD dataset, a collection of question-answer pairs derived from Wikipedia articles (Rajpurkar et al., 2016). The dataset contains +100k question-answer pairs. It consists of a list of articles, which in turn is divided into paragraphs. For each paragraph there is a set of associated human-generated questions. Each question has an id, the text of the question, and a list of possible answers. Some questions have only one answer, others may have multiple correct answers. Each answer is characterized by the text of the answer itself and by the offsets where the answer text span starts and ends. Given a question and a passage containing the answer, the task is to predict the answer text span in the passage.

The dataset was parsed into a suitable tabular data structure (Pandas Dataframe), which we used for subsequent pre-processing tasks. In particular we allocated one row for each context/question/answer triple. We decided to handle possible questions having multiple correct answers by simply considering only the first answer.

The training dataset has been split using 80% of the whole samples for training purposes, and using the remaining 20% for validation purposes. It is also worth pointing out that it is guaranteed that the entire set of rows referencing the same context is kept into the same split, thus avoiding unjustifiable boosting of results.

Preprocessing has been handled differently depending on the experiment done.

### 3.1 Word GloVe

In order to obtain the word-level tokens from GloVe, we processed the text in the following manner:

- Text has been set to lowercase.
- Punctuation marks have been removed.
- Empty word ‘ ’ has been replaced with the tag <EMPTY>, which has been treated as an

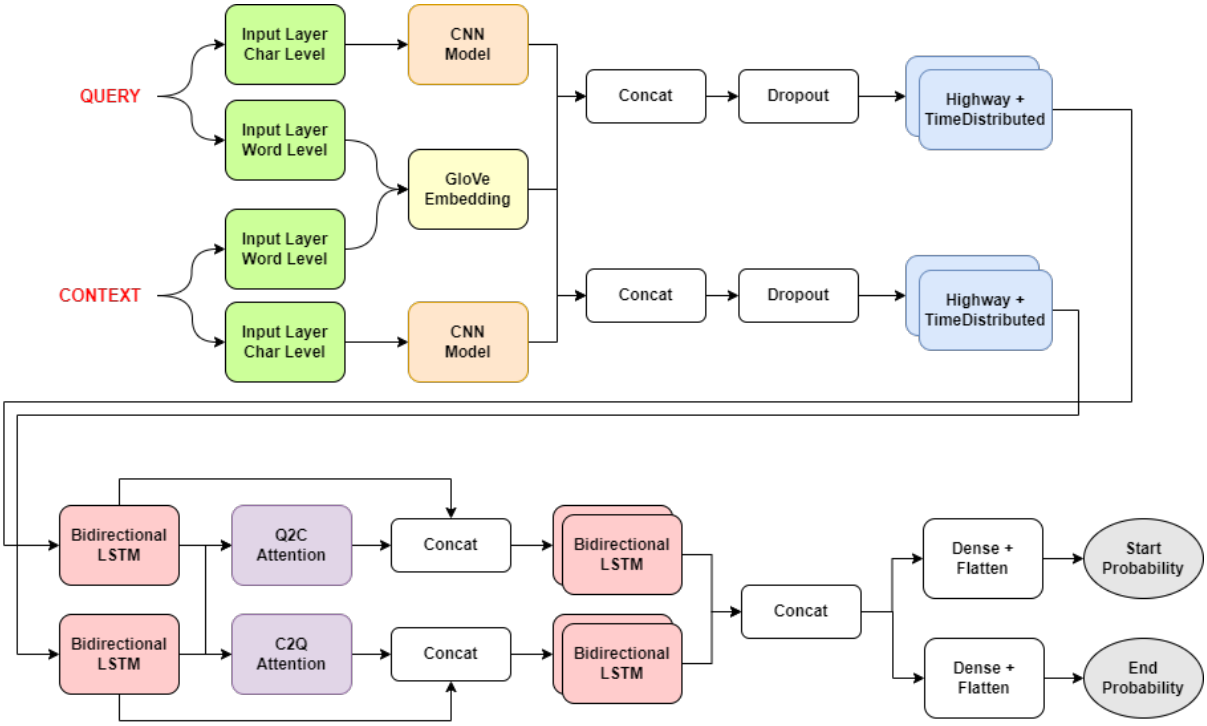


Figure 5: Architecture of the GloVe + Character Embedding RNN Network

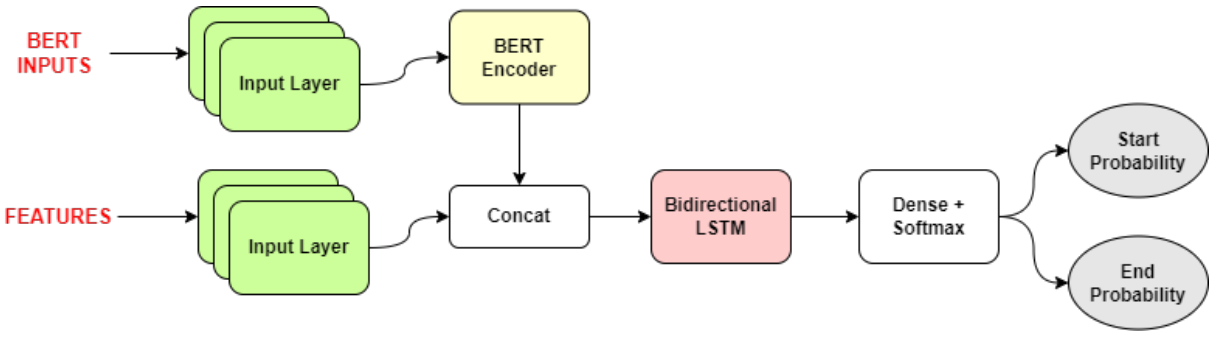


Figure 6: Architecture of the BERT based Network with added features and recurrent layer

Out-Of-Vocabulary (OOV) word later during the embedding step.

- Tokenized contexts and questions, discarding samples whose number of words exceeded a certain threshold, to make the models easier to run and remove rare outliers. After running a statistical analysis on the corpus, the threshold has been set to 400 tokens in the case of context, and to 40 tokens in the case of question. We have tokenized text based on whitespaces.

- A list containing all the unique words we found in the dataset has been created considering all the contexts and queries, and then they have been embedded using pretrained GloVe with dimension 300. We assigned a random embedding to training set words that didn't have a specific embedding inside GloVe's embeddings dictionary. We noticed that most of these words are mainly typos (e.g., "riskrelated" is actually "risk related"), numbers (e.g., 2644 or 194546) or non-English words (e.g., "lauluväljak", which is the Estonian term to indicate the Tallinn Song Festival Grounds).

- <UNK> and <PAD> tags have been added to handle respectively OOV words found in the validation dataset and to pad the embeddings. The <UNK> tag is embedded as a random vector, while the <PAD> tag is embedded as a zero vector. Padding has been added to all contexts and queries to reach the threshold value.

We processed also additional features to be added to our model in one of its variant:

- **Exact match and lemma match:** By comparing question and context we added a score based on how frequently each document word appears in the question and vice versa. We used both exact match and lemma match to compute two different scores.

- **Term frequency:** which represents how many times each word appears in its context.

In experiments that we discarded for the RNN Architecture we also used POS (Part-of-Speech) tags as additional features created using the *NLTK* [4] library and handled using One Hot Encoding. We kept this kind of feature in one of the BERT model variants.

[4]NLTK Library

### 3.2 Character Level Embedding

In one of our experiments using GloVe embeddings we also use a character embedding network, as described better in the System Description section. In terms of preprocessing we follow these steps:

- We defined an arbitrary alphabet containing all lowercase letters, numbers and punctuation, together with the <PAD> and <UNK> tags.

- We assigned a value to each character level embedding by giving it the corresponding word embedding that GloVe would use.

- Padding and OOV characters have been taken into account similarly to above.

### 3.3 BERT

In the case of the BERT encoder we used the standard BERT preprocessing using the BertWordPieceTokenizer for the *bert base uncased* model, which will lowercase all text and divide each word in one or more subwords embedding tokens, making it very powerful to minimize the effect of possible OOV words. BERT's embedding format (namely, [CLS] sentence A [SEP] sentence B [SEP]) has been guaranteed not to confuse the model and generate inconsistencies at training time. At preprocessing time we used an additional custom function to map BERT sub-word token indices to word level token indices using a lookup dictionary. This additional step allowed us at inference time to use the lookup table to reference the actual written text, discarding many errors that could have arisen if using the provided built-in BERT token decoding.

## 4 Experimental setup and results

We ran five experiments representative of the RNN approach: baseline, with added features, with added attention, with added features and attention, with added attention and character embeddings. We ran three experimetns representative of the Transformer model approach, using BERT: baseline, with added recurrent layers and with added recurrent layers and features.

### 4.1 Training details

We used the following training hyperparameters for all the models in exam:

- **Sparse Categorical Cross Entropy** as a loss function;
- **Adam optimizer** with a starting learning rate of 5e-5, reducing it each epoch by multiplying it by a factor of 0.8 using a callback. We found that reducing the learning rate each epoch yielded better performing results;
- We trained each model until its validation loss stopped improving, using an early stopping callback. For each model we took the better performing weights in terms of validation loss.

For each model it took around 5/10 epochs to reach the end of training, depending on the model.

### 4.2 Metrics

To evaluate the performance of the models we used two widely accepted metrics for this kind of task, namely F1 score and exact match. Both the metrics have been calculated as in the official evaluation script of the SQuAD paper (Rajpurkar et al., 2016).

- **Exact match:** The percentage of predicted answers that match the ground truth answer exactly. This metric tells us how much the answer is good from the user point of view, namely how much the answer is close to that of the human.
- **F1 score:** The average overlap between predicted and ground truth answers. F1-score tells us how much the answer provided by the model is statistically close to the ground truth answer.

### 4.3 Results

You can find the training results on our validation set and on the official SQuAD test set in Table 1. It is worth mentioning that we are not reporting the results of the RNN with attention and features since it did not provide any additional performance gain, in fact it was just slightly performing under the attention model.

| | Validation Set | | Test Set | |
|---|---|---|---|---|
| | EM (%) | F1 (%) | EM (%) | F1 (%) |
| **GloVe Variants** | | | | |
| **Baseline** | 4.88 | 8.78 | 6.55 | 10.55 |
| **Features** | 26.66 | 39.43 | 34.70 | 45.68 |
| **Attention** | **40.47** | **56.54** | **51.60** | **64.03** |
| **Attention + Char. RNN** | **41.67** | **58.23** | **53.16** | **66.01** |
| **BERT Variants** | | | | |
| **Baseline** | 56.49 | 72.32 | 71.19 | 79.79 |
| **RNN** | **58.51** | **74.61** | **74.23** | **82.47** |
| **RNN + Features** | 56.75 | 72.99 | 71.95 | 81.01 |

Table 1: Results on validation and test set

## 5 Discussion

The following observations can be made by looking at our resulting metrics:

- Baseline performance for the RNN model are extremely poor. By their own LSTM layers are not capable of producing good results.
- Our method of imitating attention works well in injecting enough data for the RNN model to boost results significantly. This means that providing a way for the model to cross information sources between context and query is fundamental in achieving good results.
- Our attention based variant performs much better, quite a bit better than the features only model.
- Adding additional features to an attention model does not provide additional performance boost.
- Adding character based embeddings increases performance by a couple of percentage points.

While adding additional features were quite useful in our simplest GloVe based RNN, it was actually detrimental in our attention based models, both Transformer and RNN based, to the point that we didn't report the result of our RNN based model with features and attention. We think this is because both Transformer based attention and dot product attention inject the model with the same information that the additional features would add, in a more efficient manner. Generally, we see an increase in performance with more complex models.

As part of our project we also performed an error analysis on five of our reference models, including the baseline one based on the BERT embedding. We performed both a quantitative and qualitative error analysis to find the weakest points of our work and look for ways to improve upon it.

**Qualitative Analysis**: To analyze and compare the prediction results of our models we chose a batch of questions and answers that we deemed representative of a wide variety of samples, considering things like: how long or short is the answer, if it represents an usual problem that we can find in more prediction samples or if the answer contained a number, like a date or a percentage. You can see a comparison of some prediction results in our project repository[5]. The findings are in line with the metrics analysis, confirming that the model needs to be able to extract cross-information between query and context in order to be able to give valid answers reliably. We also noted that some answers given by the models, even if not considered correct by evaluation, are actually better to a human reader than the original dataset answer (see the BERT and GloVe + CNN answers for questions 4 and 5).

**Quantitative Analysis**: Looking at prediction results we saw that a good number of answers are off only by a couple of tokens to the actual dataset answer, so we wanted to estimate how many of these "partial answers" were predicted by the models. We defined a "partial answer" as an answer whose start and end token fall within 3 tokens of the correct answer token. The results are shown below:

| | Error Ans. (%) | Partial Ans. (%) | Exact Ans. (%) |
|---|---|---|---|
| **GloVe Variants** | | | |
| **Baseline** | 92.90 | 2.27 | 4.83 |
| **Features** | 60.64 | 12.63 | 26.73 |
| **Attention** | 43.23 | 16.41 | 40.36 |
| **Attention + Char. RNN** | 42.38 | 15.89 | 41.74 |
| **BERT Variant** | | | |
| **Baseline** | 30.01 | 12.85 | 57.13 |

Table 2: Quantitative error analysis

As we can see, if we exclude the baseline model, we estimate around 12% to 16% of predicted answers that would have been considered as wrong are actually quite close to the real answer that the dataset expected, making this type of answer a pretty frequent one. If we estimate that even just half of the given "partial answers" are good enough for a human user to understand the model output, this would push the actual perceived performance of the models to be a bit better than what we estimated through the exact match metric.

## 6 Conclusion

In conclusion, we can say that our experiments provided enough evidence to say that query and context cross-information handling is fundamental in order to create a performing QA model. Starting from a very basic way of injecting this information through simple features, up to a full Transformer model based on attention, this is the most important area of focus in order to deal with the task at hand. We can also say that sub-word features are quite useful to deal with OOV words. We did not expect our simple feature injection to increase performances so much, but a full attention mechanism remains much more powerful. We expected that combining features and attention would be degrading performances, as attention has the capabilities to take into account the provided features and many more.

Even though we are happy with our results regarding our RNN models, we weren't able to match the performances of a Transformer model, as expected, but future development in RNN based question answering might be able to provide us with models that are easier to train in an end-to-end manner and more efficient to deploy. Also, smart usage of additional features might be employed in areas were a deep learning model might be able to deployed, making them still very interesting to understand how information is shaped in textual data.

## 7 Links to external resources

You can find the full code of the project here: https://github.com/simonesimo97/squad-nlp

[5] Full Error Analysis results

## References


Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. 2016. Character-aware neural language models. In *Thirtieth AAAI conference on artificial intelligence*.

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

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*.

Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. 2016. Bidirectional attention flow for machine comprehension. *arXiv preprint arXiv:1611.01603*.

Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. 2015. Highway networks.