Building a Question Answering System with a Character Level Convolutional Neural Network and Attention Layers – *Is it a good idea?*

Praveen Govindraj
SCPD
Houston, Texas
pbabu@stanford.edu

Abstract

Question answering forms the basis of interacting with modern computers. With the development of advanced computer hardware and software, it becomes imperative that we develop systems that can directly understand human communication without the need for machine level code to translate it. In this paper we introduce one such methodology applied to the problem of communicating with a computer via questions and answers.

1 Introduction

Natural Language Processing focuses on the interactions between human language and computers. It sits at the intersection of computer science, artificial intelligence, and computational linguistics. NLP is a way for computers to analyze, understand, and derive meaning from human language in a smart and useful way. By utilizing NLP, developers can organize and structure knowledge to perform tasks such as automatic summarization, translation, named entity recognition, relationship extraction, sentiment analysis, speech recognition, and topic segmentation.

Artificial intelligence and deep learning evolved from the research on human brain modeling. The initial development of neural networks evolved from the ambition to understand and mimic the human brain. The need for computer automation for complex tasks pushed the boundaries significantly in the last decade leading researchers to come up with very sophisticated algorithms.

Deep learning methods play an important role in NLP tasks such as image recognition, sentiment analysis, text classification and so on. One among the many methods, the **C**onvolutional **N**eural **N**etwork (**CNN**) has recently achieved remarkable performance on the task of sentence classification. [1]

Convolutional Neural Networks are very similar to ordinary Neural Networks. They are also made up of neurons that have weights and biases that can be learnt through many iterations. CNN are just several layers of convolutions with nonlinear activation functions applied to the results. They utilize layers with convolving filters that are applied to local features. The architecture usually consists of an input layer that contains sentences which are made up of word2vec word embeddings. This is followed by a convolutional layer, a max-pooling layer and a softmax classifier. The softmax layer gives the output as the probability distribution over different labels.

CNN have been traditionally applied to image processing (Fig1). Each region of the input is a neuron. Information from a large number of regions is processed via filters and weights to give a better understanding of the features that form the original input. In the field of NLP, the input is a series of characters, words and sentences.

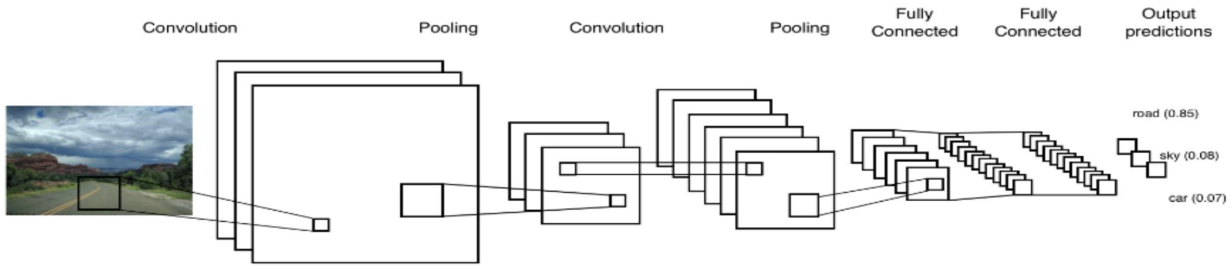

Fig 1: CNN applied to Image Processing[2]

1.1 Project Scope

The goal of the exercise was to produce a reading comprehension system that works on SQuAD (Stanford Question Answering Dataset)[5]. The SQuAD dataset consists of 100K questions, answers and contexts based on information collected from Wikipedia. The answers were crowdsourced. The QA system has to come up with a correct answer for a question based on a context provided for the question. Contexts, question and answers were divided into three phases- Train, Dev and Test. Performance of the model was then measured via two metrics – F1 and Exact match (EM) score.

From, the get go, the project was undertaken with the explicit goal of not to to beat a certain score or top the leader-boards but was approached with an exploratory view to understand the application of CNN to NLP and see the outcomes of a particular design. NLP Researchers face many design decisions before building a certain model. The outcomes of each of those decisions are not known in advance and cannot be judged to yield a particular outcome due to the presence of a large number of hyper parameters. The following pages will report on the model and approach that was undertaken for the project and the results of that approach.

2 Model

The model consists of four components - Character embedding layer, RNN encoder layer, Attention layer and an Output layer. Character embedding layer maps each word to a vector space using character level CNNs. RNN encoder layer encodes both the context and the question into hidden states. Attention layer combines the context and question representations. Output layer applies a fully connected layer and one softmax layer to get the start location and another softmax layer to get the end location of the answer span. The model was coded in python and mostly uses Tensorflow and other relevant python modules.

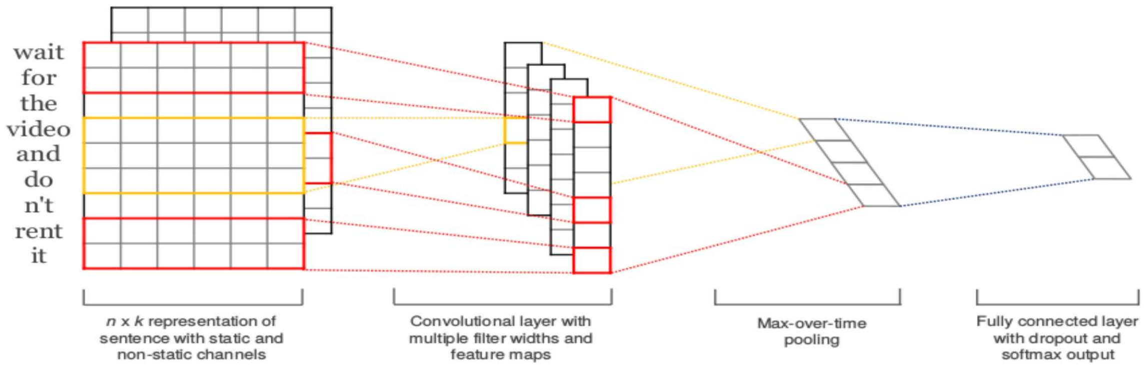
wait
for
the
video
and
do
n't
rent
it
n x k representation of sentence with static and non-static channels
Convolutional layer with multiple filter widths and feature maps
Max-over-time pooling
Fully connected layer with dropout and softmax output

Fig 2: Model architecture with two channels for an example sentence[1]

Character Embedding Layer: The following description of the model has been reproduced from [3]. Character embedding layer is responsible for mapping each word to a high-dimensional vector space. Let $\{x_1, \ldots x_N\}$ and $\{q_1, \ldots q_M\}$ represent the words in the input context paragraph and query, respectively. Following [1], we obtain the character level embedding of each word using Convolutional Neural Networks (CNN). Characters are embedded into vectors, which can be considered as 1D inputs to the CNN, and whose size is the input channel size of the CNN.

The following description has been reproduced from the project description page.

RNN Encoder Layer: For each SQuAD example (context, question, answer), the context is represented by a sequence of d-dimensional word embeddings $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N \in \mathbb{R}^d$, and the question by a sequence of d-dimensional word embeddings $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_M \in \mathbb{R}^d$. These are fixed, pre-trained GloVe embeddings. The embeddings are fed into a 1-layer bidirectional GRU (which is shared between the context and the question):

$$\{\overrightarrow{\boldsymbol{c}_1}, \overleftarrow{\boldsymbol{c}_1}, \ldots, \overrightarrow{\boldsymbol{c}_N}, \overleftarrow{\boldsymbol{c}_N}\} = \text{biGRU}(\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\})$$
$$\{\overrightarrow{\boldsymbol{q}_1}, \overleftarrow{\boldsymbol{q}_1}, \ldots, \overrightarrow{\boldsymbol{q}_M}, \overleftarrow{\boldsymbol{q}_M}\} = \text{biGRU}(\{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_M\})$$

The bidirectional GRU produces a sequence of forward hidden states ($\overrightarrow{\boldsymbol{c}_i} \in \mathbb{R}^h$ for the context and $\overrightarrow{\boldsymbol{q}_j} \in \mathbb{R}^h$ for the question) and a sequence of backward hidden states ($\overleftarrow{\boldsymbol{c}_i}$ and $\overleftarrow{\boldsymbol{q}_j}$). We concatenate the forward and backward hidden states to obtain the *context hidden states* $\boldsymbol{c}_i$ and the *question hidden states* $\boldsymbol{q}_j$ respectively:

$$\boldsymbol{c}_i = [\overrightarrow{\boldsymbol{c}_i}; \overleftarrow{\boldsymbol{c}_i}] \in \mathbb{R}^{2h} \quad \forall i \in \{1, \ldots, N\}$$
$$\boldsymbol{q}_j = [\overrightarrow{\boldsymbol{q}_j}; \overleftarrow{\boldsymbol{q}_j}] \in \mathbb{R}^{2h} \quad \forall j \in \{1, \ldots, M\}$$

Attention Layer: Next, we apply basic dot-product attention, with the context hidden states $\boldsymbol{c}_i$ attending to the question hidden states $\boldsymbol{q}_j$. For each context hidden state $\boldsymbol{c}_i$, the *attention distribution* $\alpha^i \in \mathbb{R}^M$ is computed as follows:

$$\boldsymbol{e}^i = [\boldsymbol{c}_i^T \boldsymbol{q}_1, \ldots, \boldsymbol{c}_i^T \boldsymbol{q}_M] \in \mathbb{R}^M$$
$$\alpha^i = \text{softmax}(\boldsymbol{e}^i) \in \mathbb{R}^M$$

The attention distribution is then used to take a weighted sum of the question hidden states $\boldsymbol{q}_j$, producing the *attention output* $\boldsymbol{a}_i$:

$$\boldsymbol{a}_i = \sum_{j=1}^{M} \alpha_j^i \boldsymbol{q}_j \in \mathbb{R}^{2h}$$

The attention outputs are then concatenated to the context hidden states to obtain the *blended representations* $\boldsymbol{b}_i$:

$$\boldsymbol{b}_i = [\boldsymbol{c}_i; \boldsymbol{a}_i] \in \mathbb{R}^{4h} \quad \forall i \in \{1, \ldots, N\}$$

Output Layer: Next, each of the blended representations $\boldsymbol{b}_i$ are fed through a fully connected layer followed by a ReLU non-linearity:

$$\boldsymbol{b}_i' = \text{ReLU}(\boldsymbol{W}_{FC}\boldsymbol{b}_i + \boldsymbol{v}_{FC}) \in \mathbb{R}^h \quad \forall i \in \{1, \ldots, N\}$$

where $\boldsymbol{W}_{FC} \in \mathbb{R}^{h \times 4h}$ and $\boldsymbol{v}_{FC} \in \mathbb{R}^h$ are a weight matrix and bias vector. Next, we assign a score (or logit) to each context location i by passing $\boldsymbol{b}_i'$ through a downprojecting linear layer:

$$\text{logits}_i^{\text{start}} = \boldsymbol{w}_{\text{start}}^T \boldsymbol{b}_i' + u_{\text{start}} \in \mathbb{R} \quad \forall i \in \{1, \ldots, N\}$$

where $\boldsymbol{w}_{\text{start}} \in \mathbb{R}^h$ is a weight vector and $u_{\text{start}} \in \mathbb{R}$ a bias term. Finally, we apply the softmax function to $\text{logits}_{\text{start}} \in \mathbb{R}^N$ to obtain a probability distribution $p^{\text{start}} \in \mathbb{R}^N$ over the context locations $\{1, \ldots, N\}$:

$$p^{\text{start}} = \text{softmax}(\text{logits}^{\text{start}}) \in \mathbb{R}^N$$

We compute a probability distribution p^{end} in the same way (though with separate weights $\boldsymbol{w}_{\text{end}}$ and u_{end}).

Loss: Our loss function is the sum of the cross-entropy loss for the start and end locations. That is, if the gold start and end locations are $i_{\text{start}} \in \{1, \dots, N\}$ and $i_{\text{end}} \in \{1, \dots, N\}$ respectively, then the loss for a single example is:

$$\text{loss} = -\log p^{\text{start}}(i_{\text{start}}) - \log p^{\text{end}}(i_{\text{end}})$$

During training, this loss is averaged across the batch and minimized with the Adam optimizer.

Prediction: At test time, given a context and a question, we simply take the argmax over p^{start} and p^{end} to obtain the predicted span $(\ell^{\text{start}}, \ell^{\text{end}})$:[8]

$$\ell^{\text{start}} = \text{argmax}_{i=1}^{N} p_i^{\text{start}}$$
$$\ell^{\text{end}} = \text{argmax}_{i=1}^{N} p_i^{\text{end}}$$

If $\ell^{\text{start}} > \ell^{\text{end}}$, then the predicted answer text is just an empty string.

3 Approach

CNNs require character embeddings similar to word embeddings. There were two options available to come up with the character embeddings. The first option was to initialize the character embeddings with random numbers and the second option was to use pre-trained character embeddings similar to the Glove word embeddings. The second option was decided as the best approach to take since it was easily available[4]. The character embeddings were trained over 840B words but had 300 dimensions. The 300 dimensions seemed big but had not been tested and so it was decided to give it a try. Character encodings were created from the masked characters and passed to the CNN.

Character encodings obtained from the CNN are concatenated with GloVe word embeddings and fed to the RNN encoder layer.

4 Experiments

The required parts of the model shown in the previous page was coded up and ran with different combinations of the hyper parameters. Among the many hyper parameters, the most important hyper parameters that determined the accuracy of the results of the model were the character embedding size, context length, the maximum word length, the maximum character length and the batch size.

The model was run several times to tune the hyper parameters. Some of the changes that were tried in the model involved changing the bidirectional GRU layers with bidirectional LSTM layers in the RNN Encoder Layer. Relu was used for the non-linearity in the CNN and then changed to tanh to see if improvements in the results could be observed. For regularization, dropout was used on the final layer of the CNN. Several different dropout values were experimented with.

5 Results and Findings

The dev leader-board score is - {"f1": 40.15, "exact_match": 34.32}. The CNN model gives similar or subpar results compared to the baseline results (without CNN) if used alone in a QA system. CNNs are heavily influenced by hyper parameter selection. Since a 300 dimension character embedding was used, 300 dimension word embedding also had to be used. This caused memory issues on the GPU and so had to limit to low input parameter size for the context length, character length and maximum word length. This limited the ability of the model to learn over larger number of words and learn better weights.

Using a large dimensional pre-trained character embeddings does not help in CNN. In fact it is a limiting factor as it causes a lot of memory issues which in turn leads to limiting the size of the other important hyper parameters. The character embedding size should be small and can be initialized with random vectors which can be trained later. Max overtime pooling could also be employed to improve results.

Location invariance and local compositionality which are the main ideas behind CNNs make sense in case of computer vision applications but they do not make much sense in the case of NLP. The location where a word lies in the whole sentence is of utmost importance. Words which are close to one another in a sentence may not be connected in terms of meaning which is quite contrary to pixels in a specific region of an image that may be a part of a certain object. Thus, CNNs are generally applied only to classification tasks such as topic categorization or sentiment analysis and so other authors have found better results with CNN applied to those topics.

Convolution and pooling operations lose information about the order of words. So applications like PoS tagging or entity extraction where sequence is important are harder to fit into the traditional CNN architecture. These shortcomings lead to the conclusion that CNNs alone cannot improve the accuracy of a QA system.

All the different experiments described in section 4 did not improve the model accuracy very much.

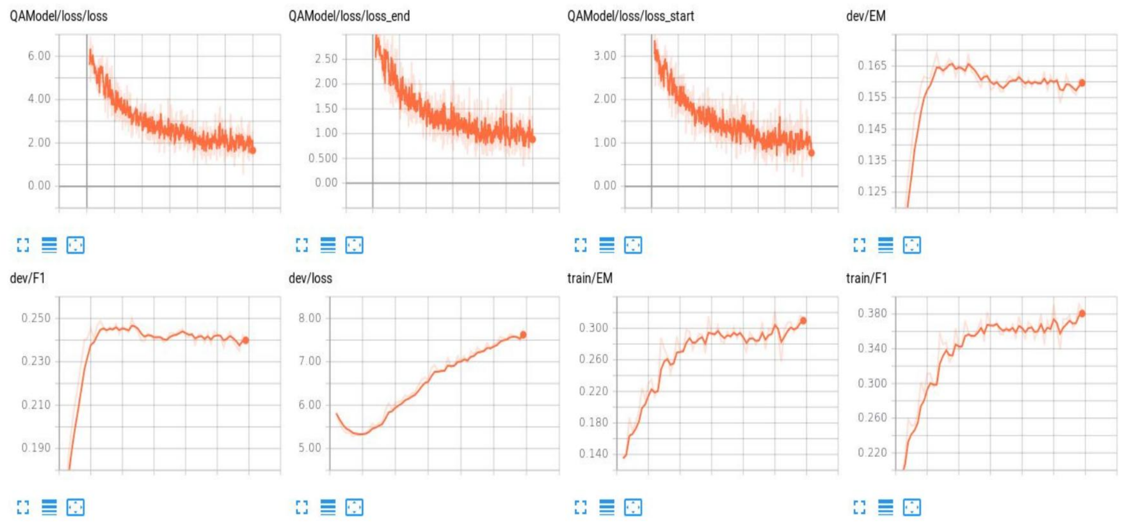

Fig 3: Tensorflow plots from one of the runs

An example-

CONTEXT: (green text is true answer, magenta background is predicted start, red background is predicted end, _underscores_ are unknown tokens). Length: 100
in 1854 at ballarat there was an armed rebellion against the government of victoria by miners protesting against mining taxes (the " eureka stockade ") . this was crushed by british troops , but the discontents prompted colonial authorities to reform the administration (particularly reducing the hated mining licence fees) and extend the franchise . within a short time , the imperial parliament granted victoria responsible government with the passage of the colony of victoria act 1855 . some of the leaders of the eureka rebellion went on to become members of the victorian parliament .
QUESTION: what was the incident over taxes at ballarat called ?
TRUE ANSWER: eureka stockade
PREDICTED ANSWER: eureka stockade ") . this was crushed by british troops , but the discontents prompted colonial authorities to reform the administration (particularly reducing the hated mining licence fees) and extend the franchise . within a short time , the imperial parliament granted victoria responsible government with the passage of the colony of victoria act 1855
F1 SCORE ANSWER: 0.091
EM SCORE: False

The example shown in the previous page highlights the issue with CNN for question answering. The model cannot exactly figure out the location boundary. It cannot easily distinguish the start and stop position. A more accurate model such as the BiDAF, Dynamic coattention or self attention networks might be better at tagging the right words that form an answer.

6 Acknowledgement

The author would like to thank Richard Socher, Abi See and all the team of TAs for striving to help us understand the applications of deep learning techniques to NLP and especially all the TAs who answered all our questions patiently on Piazza.

7 References

1. Kim, Y. (2014). Convolutional Neural Networks for Sentence Classification. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP 2014), 1746–1751.

2. Deep Learning applied to NLP, Marc Moreno Lopez and Jugal Kalita

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

4. https://github.com/minimaxir/char-embeddings

5. Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100, 000+ questions for machine comprehension of text. CoRR, abs/1606.05250, 2016.