Adapting Transformer-XL Techniques to QANet Architecture for SQuAD 2.0 Challenge

Lorraine Zhang*
Stanford University
lz2017@stanford.edu

Abstract

Through trials and errors or bold exploration, many breakthrough research and applications of deep learning models have achieved state-of-the-art performance on reading comprehension system by combining approaches from multiple models. In this project, I tried a novel approach by applying the latest techniques of recurrent mechanism and a relative positional encoding scheme from Transformer-XL[1] to the QANet architecture which is known for its fast training speed while maintaining equivalent accuracy to recurrent models[2]. The goal is to further improve QANet performance on SQuAD 2.0 by capturing longer-term dependency. My experiment results show that my QANet implementation alone can achieve decent F1 and EM scores of 68 and 64 respectively. At 16/30 epochs, although the QANet and Transformer-XL combination(QANet-XL) score lower than the vanila QANet with F1 64.87 EM 61.75, its dev set NLL of has been lower than that of QANet at 1.45 million steps. Its steeper climb on F1 and EM on the Tensorboard suggest that QANet-XL might outperform QANet at the end.

1 Introduction

Despite recent breakthroughs in NLP, being able to answer questions correctly in longer context on a reading comprehension system still imposes many challenges. This project is an attempt to explore the feasibility of combining the novel techniques from Transformer-XL with QANet architecture to address this issue.

A QANet model was first implemented based on the baseline BiDAF code and the Yu et al. (2018) paper. It is a single model architecture whose encoder consists solely of convolution and self-attention so that its training speed can be boosted through parallelization. Optimization was done on the QANet model through hyperparameters tuning in order to achieve the best possible F1 and EM scores on SQuAD 2.0 before Transformer-XL techniques were incorporated.

Using the optimized QANet as the basic architecture, I created a new self attention layer called Self-AttentionXL in its encoder block. Instead of the original dot-product calculation, the SelfAttentionXL score is computed based on the relative positional encoding. Four new variables, as discussed in details later, were created in order to implement this new scheme. The output of the encoder block was cached, then fed back into the block's input as the previous state for the current state. As a result, longer context was catpured by concatenating the previous state as memory with the current state. Intuitively, this combination should help AI systems better understand longer paragraphs, thus, better answers.

*Use footnote for providing further information about author (webpage, alternative address)—*not* for acknowledging funding agencies.

2 Related Works

SQuAD is a less than three year old reading comprehension dataset aimed to faciliate development of a question and answering system. Such an AI system can help people better understand any piece of text in the future. It has inspired inventions of new language models and techniques such as ELMo, BERT, BiDAF, Transformers etc...[3] from various industry and academic entities. Please refer to Stanford University's CS224N default project handout (February 2019) or the SQuAD website for more details on work performed.

The original QANet came from a research by Yu et al. (2018) at Google. They combined local convolution with global self-attention for reading comprehension. The result was 3x to 13x faster in training in comparison to RNNs. It was once the leader on score board on SQuAD 1.0 with F1 84.6 in 2018. Throughout their implementation, they adapted techniques directly from other researchers. Among them are a two-layer highway netowrk (Srivastava et al. 2015), depthwise separable convolutions (Chollet, 2016), multi-head attention mechanism defined in Vaswani et al. (2017), and the standard context-query attention module such as Weissenborn et al. (2017) and Chen et al. (2017).

Transformer-XL, a most recent development in January 2017, addresses the limitation imposed by fixed_length context in language models. According to the authors, it can enable capturing longer-term dependency, resolve the problem of context fragmentation, and achieve better performance on both short and long sequences. The complete set of solutions consist of two techniques: a segmentlevel recurrence mechanism and a new positional encoding scheme. As a result, Transformer-XL learns dependency about 80% longer than RNNs and 450 % longer than vanilla Transformers, achieves better performanceon both short and long sequences, and is up to 1,800+ times faster than vanilla Transformer during evaluation.

QANet and Transformer-XL share many similar traits: neither uses RNNs; both rely heavily on multi-head attention; they use standard FFNs, softmax and layer norm functions. The major difference between them is that QANet uses convolutions while Transformer-XL employs linear attention layers.

Since the Transformer-XL paper was just published in January 2019, it is presumed that nobody has tried this method of adapting it to QANet, and has obtained significant results.

3 Approach

The ideas of this project are primarily based on three papers: CS224N default project handout, QANet from Yu et al. (2018), and Transformer-XL from Dai et al. (2019).

Parts of this project's implementation were adapted from 3 repositories:

- https://github.com/chrischute/squad.git
- https://github.com/andy840314/QANet-pytorch-.git
- https://github.com/kimiyoung/transformer-xl.git

The focus of this section will be placed on discussing the implementation of the QANet and Transformer-XL combination. Detailed descriptions and equations of those two individual models are already presented in the original papers, therefore will not be repeated here. Readers can refer to the original published papers and the cs224n default handout for information.

3.1 QANet

The overall QANet architecture is illustrated in Figure 1. Files modifed for QANet include args.py, layers.py, models.py, train.py. All default dimensions and hyperparameters are adopted directly from the QANet paper unless otherwise noted. Please refer to https://github.com/lzhang2017/cs224n_project.git for the actual parameters used for individual experiment.

Five layers were implemented for QANet:

1. **Input Embedding Layer** Adapted from BiDAF embedding layer. Input embedding is obtained by concatenating word embeddings and character embending. The word embedding

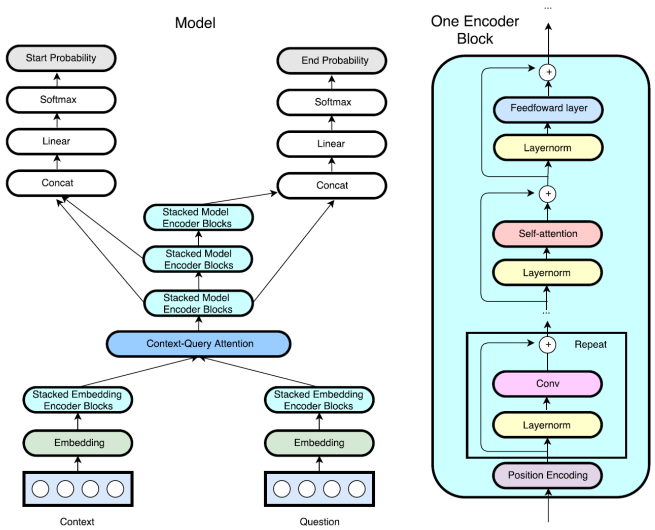

Figure 1: An overview of QANet architecture

is based on the pre-trained 300-dimension GloVe,. Characters embedding are added to this layer through a convolution layer with a dimension of 64 as compared to 200 in the original paper. Both word and character embedding pass through a standard convolution, a dropout and a highway layer.

2. **Embedding Encoder Layer** Consists of positional encoder, depthwise separable convolutions layer, self-attention layer, and feed-forward layer. The self attention score in the encoder block is calculated using **dot product**. The same Encoder Blocks are used throughout the model; only the number of convolutional layers for each block is different. They all share the same weights.
3. **Context-Query Attention Layer** It is adopted directly from the BiDAFAttention. It first takes in context and query to calculate a smilarity matrix S of batch size(n) x # of word + character dimension. It is then normalized as S_normalized = softmax(S). The attention is computed as A = S_normalized * Q.transpose, returned as a tensor of (batch size x # of filters).
4. **Model Encoder Layer** Similar to the Embedding Encoder with the same parameters, except convolution layers number and block number are set to 2 and 7 respectively.
5. **Output Layer** Adapted from the BiDAFOutput. Takes W1 and W2 as trainable weights, and M0, M1, M2 as ouptuts of the three model encoders as inputs. The output of this layer is probabilities of the starting and ending position:
    - p-start = softmax(W1[M0:M1])
    - p-end = softmax(W2[M0;M2])

3.2 Transformer-XL

The discussion of Transformer-XL will be focused on the two specific techniques I applied to QANet. Its architecture was studied in details to gain a good understanding of the novel techniques, but is less relevant in this project, thus it will not be elaborated here.

The two techniques applied to QANet are:

1. **Recurrence mechanism**: The idea is to reuse the hidden states obtained in previous segments as an extended context when processing the next new segment. Instead of computing the hidden states from scratch, the cached and reused hidden states serve as memory for the current state. Since information can be propagated through the recurrent connection

between segments, modeling very long-term dependency become possible. The n-th layer hidden state for segment S_(t+1) can be expressed as the following three equations:

$$\widetilde{h}_{\tau+1}^{n-1} = \left[SG(h_{\tau}^{n-1}) \circ h_{\tau+1}^{n-1}\right], \text{(extended context)}$$
$$q_{\tau+1}^{n}, k_{\tau+1}^{n}, v_{\tau+1}^{n} = h_{\tau+1}^{n-1}W_{q}^{T}, \widetilde{h}_{\tau+1}^{n-1}W_{k}^{T}, \widetilde{h}_{\tau+1}^{n-1}W_{v}^{T} \text{ (query, key, values)},$$
$$h_{\tau+1}^{n} = \text{Transformer-Layer}(q_{\tau+1}^{n}, k_{\tau+1}^{n}, v_{\tau+1}^{n}) \text{ (self-attention + feed-forward)}$$

2. **Relative Positional encoding scheme**: A simple but more effective relative positional encodings rather than an absolute one is introduced to keep positional information coherent when the states are reused. The fundamental idea is to only encode the relative positional information in the hidden states. The relative attention score A, and the rest of the computational procedure of Transformer-XL are summarized below:

$$A_{\tau,i,j}^{n} = {q_{\tau,i}^{n}}^{T}k_{\tau,j}^{n} + {q_{\tau,j}^{n}}^{T}W_{k,R}^{n}R_{i-j} + u^{T}k_{\tau,j} + v^{T}W_{k,R}^{n}R_{i-j}$$
$$a_{\tau}^{n} = \text{Masked-Softmax}(A_{\tau}^{n})v_{\tau}^{n}$$
$$o_{\tau}^{n} = \text{LayerNorm}(\text{Linear}(a_{\tau}^{n}) + h_{\tau}^{n-1})$$
$$h_{\tau}^{n} = \text{Positionwise-Feed-Forward}(o_{\tau}^{n})$$

3.3 Combining the Two (QANet-XL)

This part is the main contirbution of this project and is the most challenging. Figure 2 shows the new diagram of the EncoderBlock after modifcation. Altough the overall enhancement to the original structure looks rather simple, it was not as straightforward as originally thought due to special attention needed to adjust the shape of various parameters as the result of four new parameters, **mems**, **r**, **r_r_bias**, and **r_w_bias**.

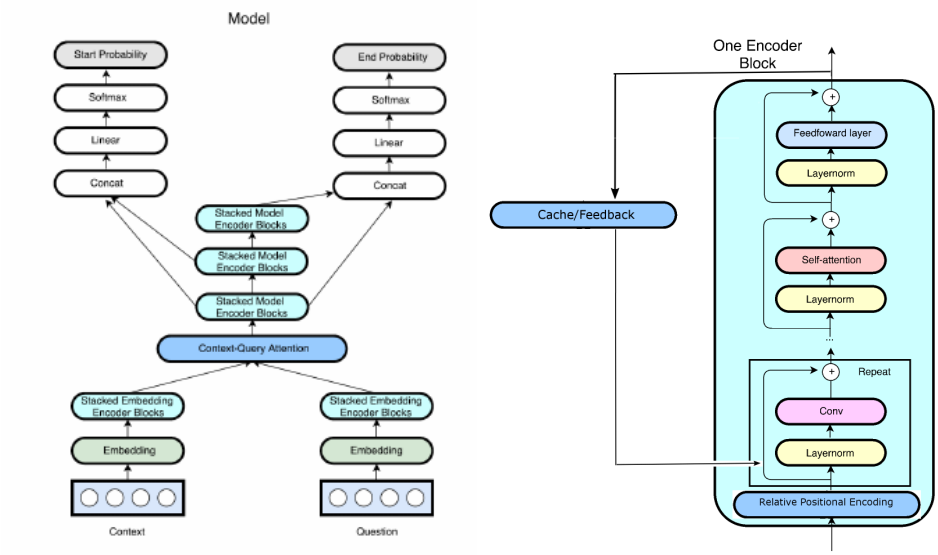

Figure 2: New QANet-XL architecture

My approach to incorporating the Transformer-XL ideas into the baseline QANet architecture relied on the description of the method and equations presented in the Transformer XL paper as well as a thorough review and comparison of a reference implementation of the architecture. Some similarities were observed between the Transformer-XL code and my QANet. For example, the

positional encoding blocks were both adopted from Vaswani et al. (2017) 's multi-head attention[5]. Consequently, the amount of change needed was reduced.

These two equations below highlight the difference between a regular Transformer and Transformer-XL.

$$A_{i,j}^{abs} = q_i^T k_j = \underbrace{E_{x_i}^T W_q^T W_k E_{x_j}}_{(a)} + \underbrace{E_{x_i}^T W_q^T W_k U_j}_{(b)} + \underbrace{U_i^T W_q^T W_k E_{x_j}}_{(c)} + \underbrace{U_i^T W_q^T W_k U_j}_{(d)}$$

$$A_{i,j}^{rel} = q_i^T k_j = \underbrace{E_{x_i}^T W_q^T \mathbf{W_{k,E}} E_{x_j}}_{(a)} + \underbrace{E_{x_i}^T W_q^T \mathbf{W_{k,R}} \mathbf{R_{i-j}}}_{(b)} + \underbrace{\mathbf{u^T W_{k,E}} E_{x_j}}_{(c)} + \underbrace{\mathbf{v^T W_{k,R}} \mathbf{R_{i-j}}}_{(d)}$$

- The relative encoding adopted the formula shown above from Transformer-XL and created three new variables into the QANet code: **r**, **r_r_bias**, and **r_w_bias**. **r** is the relative positional embedding of size (x), and is calculated every time it is called based on its position. The latter two, corresponding to the bias term in **u** and **v** in the equation, were implemented as learned parameters. This relative encoding scheme was implemented in the self attention layer of QANet's encoder block.
- Changes were subsequently made to EncoderBlock and QANet model blocks to incorporate the relative positional encoding and state caching elements of Transformer-XL.
- The basic approach used to implement the new relative encoding scheme involved first extending the context by concantenating queries with memory(**mems**). The new SelfAttentionXL is calculated based on the extended context.

In the QANet architecture, there are two distinct places that use self attention - the individual encoder blocks and the ContextQuery Attention layer. Comparing to the Transformer XL implementation, it seemed more appropriate to apply these new methods to the individual encoder blocks. These blocks are used before and after the ContextQuery attention layer in the architecture, with the blocks before working on the context and query separately. This is quite different from the Transformer XL, but since the same block is used throughout the QANet, it was not possible to implement the changes in only one section. The output of each encoder block is cached and then fed back with the next batch of data.

More challenges were presented based on the QANet's use of convolutional networks as opposed to simple linear layers like Transformer XL. Feeding back the cached information**mems** to these blocks presented size issues since some of the convolutions changed the size of some of the tensors. Additionally, since the data used was not padded to maintain a fixed size, the cached data was nearly always a different size. Padding of **r** and **mems** becomes necessary.

The code for QANet-XL was developed specifically for this project. It was not adapted from anywhere else. The new files created to incorporate Transformer-XL are layersXL.py, models.XL.py, trainXL.py and variables.py.

4 Experiments

4.1 Data

The dataset for this project came from the provided GitHub repository, and was set up by setup.py script [3]. It has threee splits: train, dev and test.

1. train (129,941 examples): All taken from the official SQuAD 2.0 training set.
2. dev (6078 examples): Roughly half of the official dev set, randomly selected.
3. test (5921 examples): The remaining examples from the official dev set, plus hand-labeled examples.

More specifically, they correspond to the following files:

- {train,dev,test}-v2.0.json: The official SQuAD train set, and modified verson of the SQuAD dev and test sets for this project.

- {train,dev,test}_{eval,meta}.json: Tokenized training and dev set data.
- glove.840B.300dglove.840B.300d.txt: Pretrained GloVevectors. Theseare300dimensional embeddings trained on the CommonCrawl 840B corpus.
- {word,char}_emb.json: Word and character embeddings. Only the words and characters that appear in the training set are kept in order to reduce the size of the embedding matrix and free up memory for the training model.
- {word,char}2idx.json: Dictionaries mapping character and words (strings) to indices (integers) in the embedding matrices in {word,char}_emb.json.

4.2 Evaluation Method

Results are evaluated based on the Exact Match (**EM**) score and **F1** score.

1. Exact Match is a binary measure (i.e. true/false) of whether the system output matches the ground truth answer exactly.
2. F1 is a less strict metric – it is the harmonic mean of precision and recall. The system would have 100% precision (its answer is a subset of the ground truth answer) and 50% recall(it only included one out of the two words in the ground truth output), thus a F1 score of 2 x prediction x recall / (precision + recall) = 2*50*100/(100+50) = 66.67%

4.3 Experimental Details

1. **Baseline** Result was obtained by training the default BiDAF model. This took about 8 hours to complete on VM.
2. **QANet alone** Implemented and optimized for SQuAD 2.0 . The character embedding used is 64 instead of the original paper's 200. I was expecting a faster run time per epoch from QANet, but it actually took longer than the baseline BiDAF model, averaging 1 epoch per hour on NV6. I attribute the slower rate to the complexity and more layers of the QANet. However, I noticed that QANet F1 and EM scores improved much faster than the baseline.
3. **Hyperparameter turning** Batch size of 64, 32, 16, 8 and 4 were experimented. I was able to use only batch size of 16 and 8 due to memory requirement on VM. Hidden size of 96, 128, 200 were tried, but only 96 is possible on NV6 and NV12. On NC24, I was able to use 128 hidden size with batch size 8. When the batch size was large and hidden size small, F1 and EM scores settled very quickly at less than 10 epochs.
4. **Transformer-XL** Ran the code from the Transformer-XL paper repository as it is to understand how it works.
5. **Combining QANet and Transformer-XL** Implemented and debugged code changes to EncoderBlock and its self attention, and the QANet class itself.
6. **Debugging** Performed on smaller training set and small values of parameters. Once it worked, training were done on Azure VM.

4.4 Results

Table 1: Best scores obtained, non-PCE

MODELS	F1	EM	EPOCHS
BASELINE BIDAF	61	57.45	30
BASELINE QANET, SQ1.0	76.2	66.3	30
QANET, DEV SET	**68.46**	**64.81**	30
QANET, TEST SET	**65.18**	**61.56**	30
QANET + TRANSFORMER-XL	64.87	61.75	23

QANet result is in line with expectation, outperforming the baseline BiDAF. Due to the lack of time to experiment, QANet-XL result is below expectation.

Table 2: Results from QANet HyperParameter Tuning

Training epochs	Batch Size	Hidden Size	Attention Heads	F1	EM
30	16	96	8	**68,46**	**64.81**
30	8	96	8	67.62	64.87
30	8	128	8	68.17	64.44

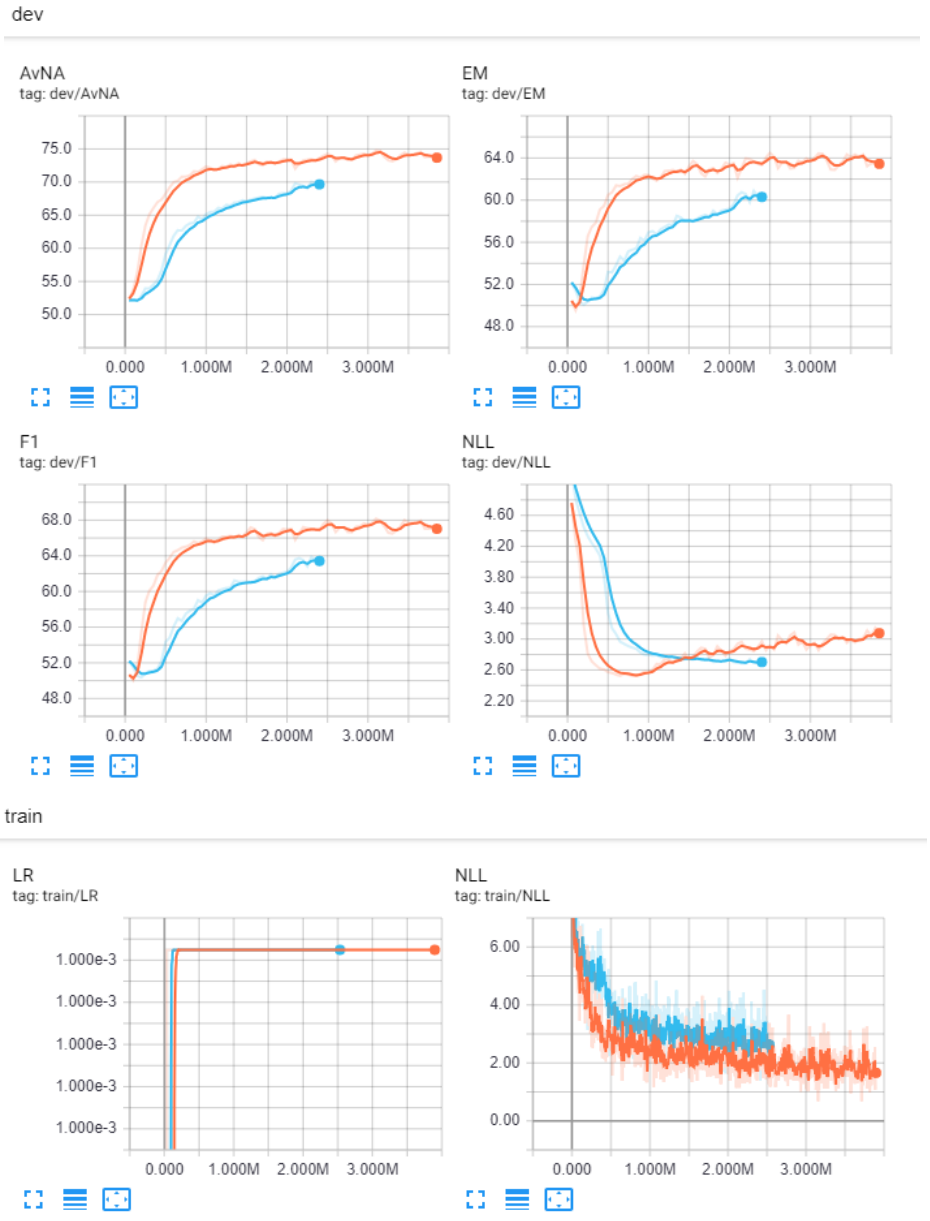

Figure 3: TensorBoard visualization of Table 1 (Orange: QANet, Blue: QAnet-XL)

5 Analysis

Although QANet-XL underperformed QANet, the TensorBoard visualization shows that early epoch learning rate was slower than QANet. The QANet essentially plateued within first 1.5 million steps, while QANet-XL was still improving at three million steps. This may indicate that given enough training time, QANet-XL could outperform vanila QANet. Using a higher drop out rate or different learning rate might improve the early epoch performance of QANet XL.

Taable 2 results of hyperparameter tuning. Hidden size, batch size and numbre of attention heads don't seem to make a significant impact of the scores. One reason that the original QANet paper was able to achieve better scores on SQuAD 1.0 because it used a character embedding of larger dimension of 200 versus 96 in this project. It was also trained on augmented data of three times of the original dataset size. I was limited of the hardware resource, specially the GPU memory. As a result, training was restricted in batch size, number of heads, hidden size and embedding size. The speed of GPU also limited the number of iterations.

Adam optimizer with a warm-up learning rate to 0.001 has an impact on the performance. When the default optimizer was used initially, QANet performance was stuck at F1 52 and EM 52.

6 Conclusion

In this project, I tried a novel approach and successfully incorporated Transformer-XL techniques into QANet. The combination seems to hold promise to outputform vanila QANet based on the trajectory of F1 and EM scores through 20 epochs, but further research effort is needed to realize it. Due to limitations on time, hardware and people resource, I could not adaquately explore optimization on QANet-XL.

Future work can include increasing drop out rate, increasing or decreasing the number of layers applied to QANet, experimenting with different learning rate.

Acknowledgments

I would like to thank the TAs who have provided insightful feedback through comments on my proposal and milestone.

References

1. Adam Wei Yu, David Dohan, Minh-Thang Luong. QANet: Combining Local Convolution with Global Self-Attention for Reading Comprehension. April 2018.
2. Zihang Dai, Zhilin Yang, et al. Transformer-XL: Attentive Language Models Beyond a Fixed-length Context. January 2019.
3. CS 224N Default Final Project handout, Stanford University. February 2019.
4. https://github.com/kimiyoung/transformer-xl
5. https://github.com/andy840314/QANet-pytorch-.git
6. Ashish Vaswani et al. Attention Is All You Need. December 2017.
7. Rupesh K. Srivastava, Klaus Greff, and Jurgen Schmidhuber. Highway Networks. arXivpreprint arXiv:1505.00387, 2015.
8. https://nlp.stanford.edu/projects/glove/