CS5242 Project Report

Project 2. Group 80: Arnold Christopher Koroa (A0092101Y)

1. Background

The need for Question Answering Systems (QAS) is increasing due to the increasing amount of data available on the web and the need to be able to retrieve and explore them efficiently [4]. A complete QAS includes many components including the natural language processing and query expansion component to process users' query; the information retrieval component to retrieve documents from databases of structured and unstructured data; and the answer extraction and generation component to present the relevant parts of the retrieved documents to the user in an intuitive way [1, 4, 7, 10].

The SQuAD dataset [13] is a reading comprehension-style dataset for question answering where a context paragraph is given for a set of questions which answers is a sub-span of the given context. This isolates out the natural language processing and answer extraction problem from the original QAS problem as it does not require querying for relevant documents from data stores and simplifies the generation of answer presentation.

The goal of the project is to implement a deep learning model to solve such a reading comprehension-style question answering problem and learn more on the application of deep learning models on natural language processing problems.

All code implemented for this project can be found at https://github.com/ackoroa/squadnet

2. Model

The model implemented in this project is the Dynamic Co-attention Networks for Question Answering as described by Xiong et al [19]. This model consists of two main components: the Co-attention Encoder, and the Dynamic Pointer Decoder.

The Co-attention Encoder encodes the context and question into an internal representation using RNN and Attention modelling [2]. The internal representation is then passed into the Dynamic Pointer Decoder which produces the start and end pointer for the sub-span of the context which forms the answer to the question.

More details of the two components is explained in the following subsections. All figures in this section are taken from [19].

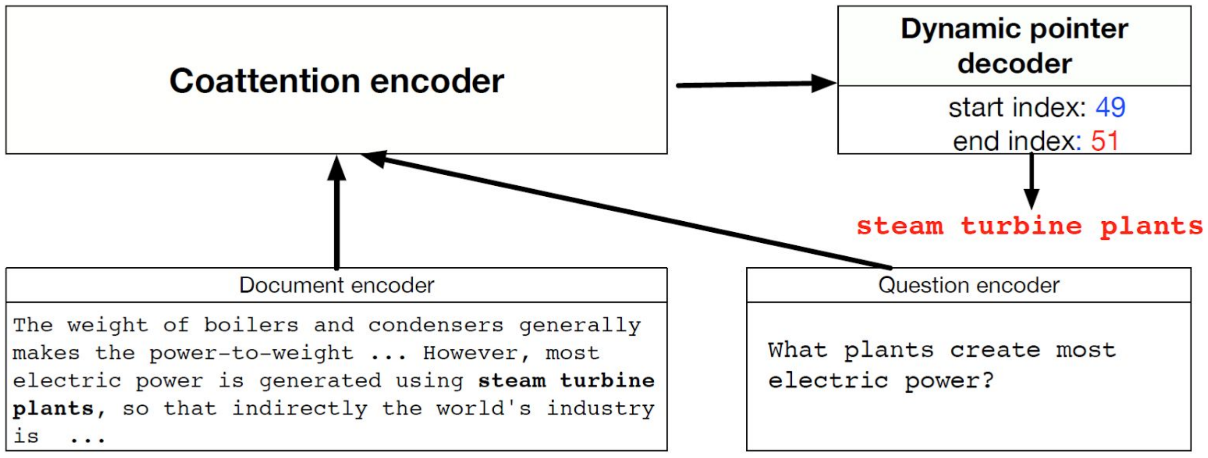

Coattention encoder
Dynamic pointer decoder
start index: 49
end index: 51
steam turbine plants
Document encoder
The weight of boilers and condensers generally makes the power-to-weight ... However, most electric power is generated using **steam turbine plants**, so that indirectly the world's industry is ...
Question encoder
What plants create most electric power?

Figure 1: Overview of the Dynamic Coattention Network.

2.1. Co-attention Encoder

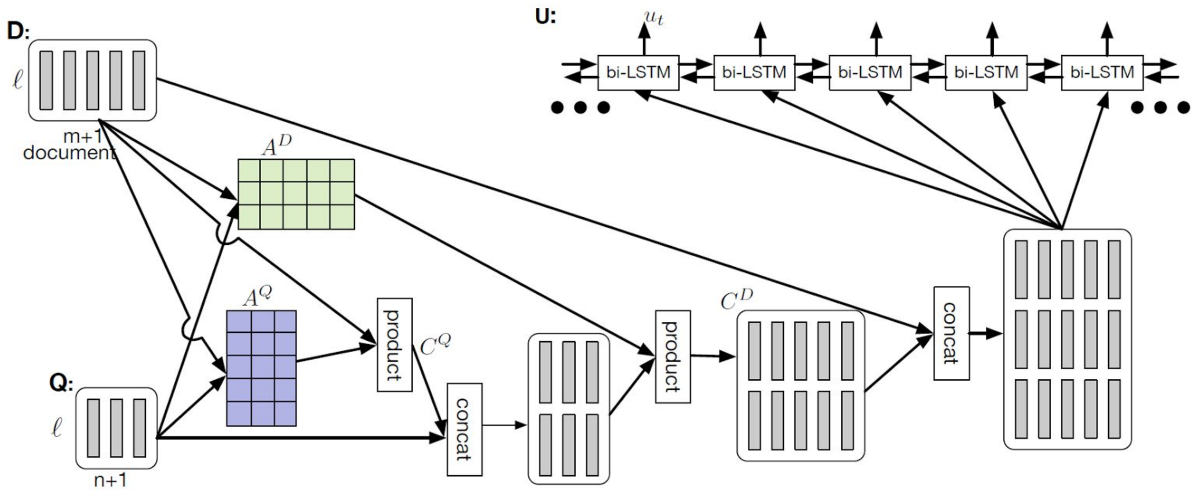


Figure 2: Coattention encoder. The affinity matrix L is not shown here. We instead directly show the normalized attention weights A^D and A^Q.

The context and question word vector sequence are first encoded using LSTMs [8] and the hidden state at each time-step are collected to form D and Q matrices, the context and question representation. D and Q are then multiplied to form the affinity matrix L and softmax-ed rowwise and columnwise to form A^Q, the attention matrix across the context document for every word in the question and A^D, the attention matrix across the question for every word in the context.

(1) $D = \text{LSTM}(\text{context}) \in R^{h \times m}$
(2) $Q = \text{LSTM}(\text{question}) \in R^{h \times n}$
(3) $L = D^T Q \in R^{m \times n}$
(4) $A^Q = \text{softmax}(L) \in R^{m \times n}$
(5) $A^D = \text{softmax}(L^T) \in R^{n \times m}$

Here m is the number of word vectors in the context, n is the number of word vectors in the question and h is the size of the LSTM hidden unit. Note that the +1 in the matrix dimensions in the original figure from [19] is because a sentinel vector is appended to the end of both context and question in the original paper but this scheme was not employed in the implementation for this project due to time constraint and is thus omitted.

C^Q and C^D is then computed as per equation (6) and (7) to form the co-dependent attention between the question and the context. This is then concatenated with the context representation and passed through a bidirectional LSTM to form the internal representation

U of the question and context. The bidirectional LSTM is implemented as per lecture 9 slide 43.

(6) $C^Q = DA^Q \in R^{h \times n}$
(7) $C^D = [Q;C^Q]A^D \in R^{2h \times m}$
(8) $U = \text{bi-LSTM}([D;C^D]) \in R^{h \times m}$
where [A; B] is the concatenation of matrix A and B along the axis of size h.

2.2. Dynamic Pointer Decoder

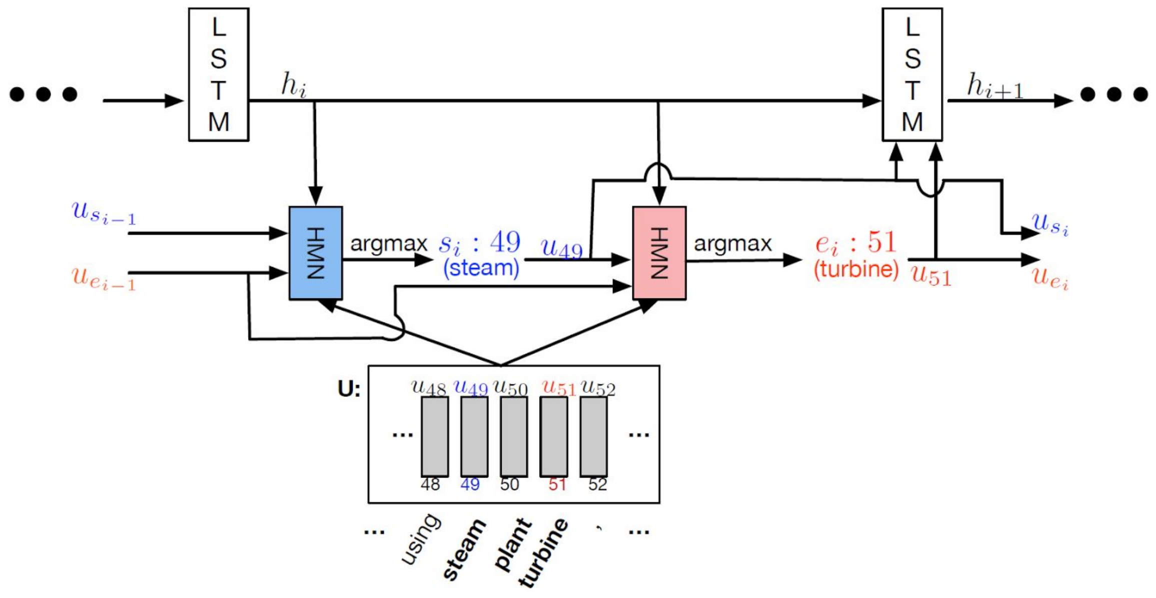


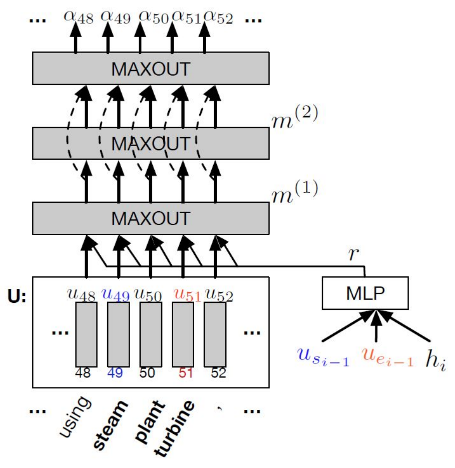
Figure 3: Dynamic Decoder. Blue denotes the variables and functions related to estimating the start position whereas red denotes the variables and functions related to estimating the end position.

The following steps are done for multiple iterations to produce iteratively better estimation of the starting word s and the ending word e. For iteration i, the internal representation for s_{i-1}, us_{i-1} and e_{i-1}, ue_{i-1} is fed into an LSTM to produce the current time-step state, The whole U together with us_{i-1}, ue_{i-1} and h_i are then fed into the Highway Maxout Network (HMN) to produce α_i, the probability distribution over the context word for s_i. The whole U together with us_i, ue_{i-1} and h_i are then fed into a different HMN to produce β_i, the probability distribution over the context word for e_i. For time step i = 0, h_{i-1} is set to be 0, s_{i-1} is set to be the first word in the context and e_{i-1} is set to be the last word in the context. In combination, this whole setup performs the role of a Pointer Network [17].

(9) $h_i = \text{LSTM}([us_{i-1};ue_{i-1}]) \in R^h$
(10) $\alpha_i = \text{HMN}(U, h_i, us_{i-1}, ue_{i-1}) \in R^m$
(11) $s_i = \text{argmax}(\alpha_i) \in [0,m)$
(12) $\beta_i = \text{HMN}(U, h_i, us_i, ue_{i-1}) \in R^m$

(13) $e_i = \text{argmax}(\beta_i) \in [0,m)$

The HMN is a short Highway Network [15] with Maxout [6] pooling. The Highway component allows faster learning due to residual links while the Maxout component acts as both a regularizer as it pools over multiple MLPs, and a universal approximator of activation functions.



α_{48} α_{49} α_{50} α_{51} α_{52}
MAXOUT
$m^{(2)}$
MAXOUT
$m^{(1)}$
MAXOUT
r
U:
u_{48} u_{49} u_{50} u_{51} u_{52}
48 49 50 51 52
MLP
$u_{s_{i-1}}$ $u_{e_{i-1}}$ h_i
using steam plant turbine

Figure 4: Highway Maxout Network. Dotted lines denote highway connections.

The HMN is defined by the following equations:

(14) $\text{HMN}(U,h,u_s,u_e) = \max(W_3 [m_1;m_2] + b_3) \in R^m$

(15) $r = \tanh(W_0 [h;u_s;u_e]) \in R^h$

(16) $m_1 = \max(W_1 [U;r] + b_1) \in R^h$

(17) $m_2 = \max(W_2\, m_1 + b_2) \in R^h$

Where concatenation between matrices and vectors such as [U;r] implies a broadcasting of the vector to match the dimension of the matrix.

Here the parameters W_n is of dimensions p ⨉ h ⨉ in where p is the Maxout pool size and in is 3h, 2h, h and 2h for $n \in 0,1,2,3$ and the max operation is performed over the axis of size p.

This works to produce an output vector of size m for an arbitrary context length m dynamically as we are treating each m vectors in U as a separate “example” to the MLP links in the HMN, which means they only need h for their input and output dimensions to be defined.

3. Implementation Details

For this project, the model as described above is implemented in Python using Chainer [16]. Chainer allows us to define neural networks simply by passing variables through operations, essentially enabling us to create any networks in a flexible way following the mathematical notations given in research papers without being restricted to predefined classes and structures like in other high-level neural network framework such as Keras [5]. This makes it easier to perform custom operations such as the affinity matrix computation and the HMN network. While according to [11] GRUs perform comparably with LSTMs with much less computational cost, as the native implementation of GRU in Chainer does not yet support mini-batching, LSTM was still used in this project for implementation simplicity even though it is possible to add that support on our own.

As a pre-processing step, contexts and questions are tokenized using NLTK [3] and converted into word-vectors using GloVe [12] vectors pre-trained on Wikipedia. The word vector dimension used is 300 and words not found in the GloVe vocabulary are represented with 0 vectors.. Punctuation marks are conserved in this process as they do convey meaning in a text and GloVe vectors contain representations for them. However, all alphabets are lower-cased as the vectors are uncased even though type-case is likely to convey information as well. There exist cased GloVe vectors with more tokens and vocabulary based on Common Crawl but this is against the project requirements and were thus not used. The original training data's start pointer is in terms of characters from the start of the paragraph and is first converted to the number of tokens. The end pointer is also computed accordingly.

Training is performed using Adam [9] with learning rate 1e-3 and mini-batch size of 48. Dropout [14] with dropout rate of 0.5 were used for all RNN and MLP outputs during training. The size of the RNN outputs h is set to 200, the Maxout pool size is set to 16 and the number of iterations for the Dynamic Pointer Decoder is set to 4 as per the original paper. Validation is performed using 10% of the training dataset and training stopped once the validation loss no longer decreases. Training was performed on a single NVIDIA K80 GPU with GPU programming support provided by Chainer's sub-library, CuPy. During training, training and validation loss are measured using softmax cross-entropy on α and β against the known answer's start and end pointers and accuracy is the exact start-end pointers match.

4. Experimental Results

During the initial phases of the project, taking inspiration from the work of Wu [18], some modifications were performed on the model above to speed-up training. These modifications are:

1. Removing the Maxout component from the HMN (i.e. setting $p = 1$)
2. Reducing the number of iterations for the Dynamic Pointer Network from 4 to 3
3. Using dropout of drop rate = 0.1 (actually [19] does not specify their dropout rate)

Using these modifications, it was found that the model quickly overfits after 3 epochs of training, with the validation loss increasing exponentially on further training.

Adding back the Maxout layer of size 16 prevents the validation loss from exponentially exploding but the model would still overfit after 3 epochs. Increasing the dropout to 0.2 and 0.5 increases the number of epoch before overfitting to 7 and 10 respectively but the model would still overfit at the range of ~20% train and validation accuracy. Letting the training continue would cause the training accuracy to go over 70% while the validation accuracy to stay about the same so this is a clear case of overfitting. An attempt to reduce the model size by setting h to 100 was also performed but this only slows down convergence and the model will still overfit after reaching ~20% train accuracy.

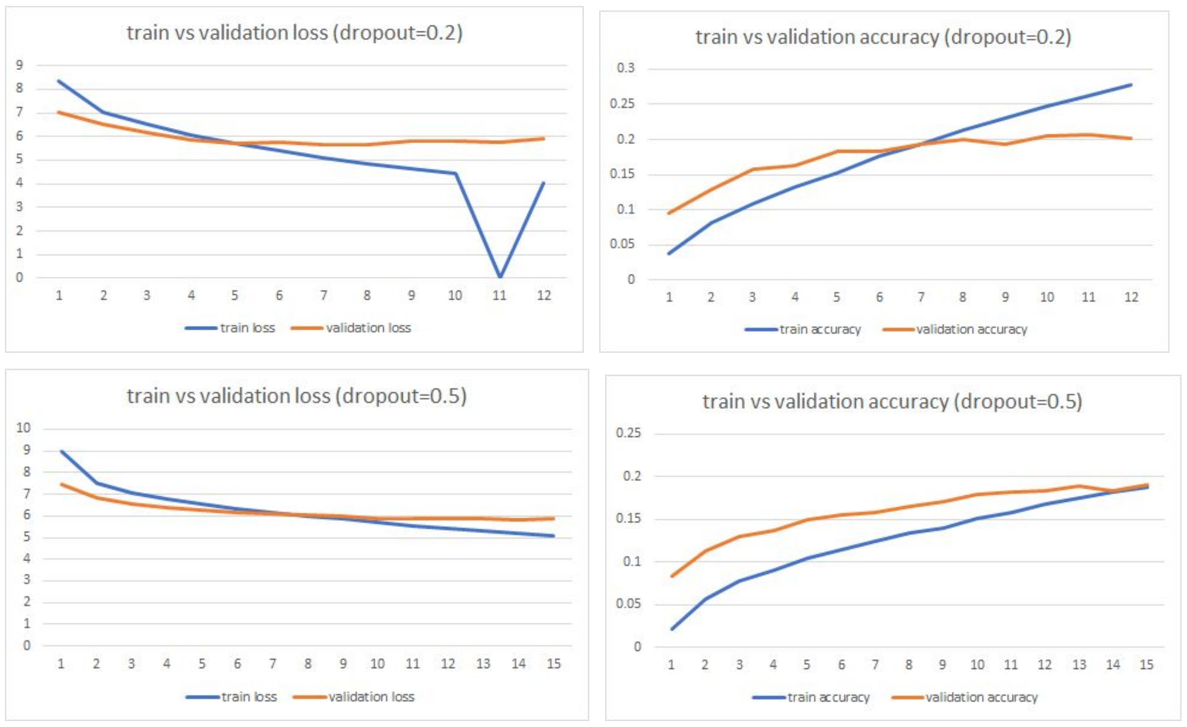


Figure 5: training and validation loss and accuracy for different dropout rate using standard settings of p = 16 and h = 200.

In the end the best result obtained on the Kaggle competition is about 25% accuracy using the 0.2 dropout model at epoch 7 and 0.5 dropout model at epoch 10.

5. Discussion

From the above experiments it is clear that the model is capable of solving the problem. In fact the original paper claims that it can reach over 70% accuracy on the test test. However, such a high performance was not achieved in this project due to overfitting problems. While multiple approaches had been attempted to tackle the overfitting problem, this project did not succeed in overcoming it.

Further approaches that can be tried short of using more training data is to use cased word vectors with more tokens and vocabulary which essentially provides more data points to the network to learn from. Another approach that can be tried is to add back the sentinel word vector at the end of each context and question strings to allow the attention mechanism to not pay attention to any word during decoding [19].

There exists a more sophisticated model, R-NET [11] which claims to have better performance than the Dynamic Co-attention Network model. In fact, as using code from the internet were allowed in this project some classmates seem to have tried running a copy found of the net and achieved about 40% accuracy on the Kaggle competition leaderboard without any tuning. Here I would like to question the wisdom of combining a competitive grading system (we are graded based on our ranking on the leaderboard) and allowing the usage of code found of the net blindly and its effect to learning effectiveness. While I was aware of the R-NET model I decided to implement the Dynamic Co-attention Network model instead as it is simpler and yet claims to only be slightly worse than R-NET. The model's relative simplicity enables me to dig deeper into its working and understand its working principles fully.

6. Conclusion

In this project, a model for reading comprehension-style question answering, Dynamic Co-attention Network, is implemented and tested. Reading comprehension-style question answering distills the natural language processing and answer extraction component of a full Question Answering System and is a good problem to work on for improving this less known subset of the full problem. While the original paper claims to be able to achieve 70% accuracy on the test set, this project only managed to achieve 25% accuracy due to overfitting problems. The working principles of natural language processing applications of deep learning, in particular RNN, Attention and Dynamic Pointer models, and their implementation was learned from this project.

References

[1] Allam, A. and Haggag, M.H.. *The Question Answering Systems: A Survey*. International Journal of Research and Reviews in Information Sciences (IJRRIS) 2(3), 2012.

[2] Bahdanau, D., Cho, K. and Bengio, Y.. *Neural Machine Translation by Jointly Learning to Align and Translate*. Proceeding of the 3rd International Conference on Learning Representations (ICLR), 2015.

[3] Bird, S., Edward L. and Ewan K.. *Natural Language Processing with Python*. O'Reilly Media Inc, 2009

[4] Bouziane, A., Bouchiha, D., Doumi, N. and Malki, M.. *Question Answering Systems: Survey and Trends*. Procedia Computer Science 73, 2015.

[5] Cholett, F.. Keras. https://github.com/fchollet/keras

[6] Goodfellow, I.J., Warde-Farley, D., Mirza, M., Courville, A. and Bengio, Y.. *Maxout Networks*. Proceedings of the 30th International Conference on Machine Learning (ICML), 2013.

[7] Gupta, P. and Gupta, V.. *A Survey of Text Question Answering Techniques*. International Journal of Computer Applications 53(4), 2012.

[8] Hochreiter, S. and Schmidhuber, J. *Long Short-Term Memory*. Neural Computation 9, 1997.

[9] Kingma, D.P. and Ba, J.L.. *Adam: A Method for Stochastic Optimization*. Proceeding of the 3rd International Conference on Learning Representations (ICLR), 2015.

[10] Mishra, A. and Jain, S.K.. *A survey on question answering systems with classification*. Journal of King Saud University - Computer and Information Sciences 28(3), 2016.

[11] Natural Language Computing Group, Microsoft Research Asia. *R-NET: Machine Reading Comprehension with Self-matching Networks*. Annual Meeting of the Association for Computational Linguistics (ACL), 2017.

[12] Pennington, J., Socher, R. and Manning, C.D.. *GloVe: Global Vectors forWord Representation*. Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP), 2014

[13] Rajpurkar, P., Zhang, J., Lopyrev, K. and Liang, P.. *SQuAD: 100,000+ Questions for Machine Comprehension of Text*. Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP), 2016.

[14] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R.. *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*. Journal of Machine Learning Research 15, 2014.

[15] Srivastava, R.K., Greff, K. and Schmidhuber, J.. *Highway Networks*. Deep Learning Workshop, International Conference on Machine Learning (ICML), 2015.

[16] Tokui, S., Oono, K., Hido, S. and Clayton, J.. *Chainer: a Next-Generation Open Source Framework for Deep Learning*. Proceedings of Workshop on Machine Learning Systems in the 29th Annual Conference on Neural Information Processing Systems (NIPS), 2015.

[17] Vinyals, O., Fortunato, M. and Jaitly, N.. *Pointer Networks*. Proceedings of the 28th Conference on Advances in Neural Information Processing Systems (NIPS), 2015.

[18] Wu., W.. *Simple Dynamic Coattention Networks*. Stanford Center for Professional Development, n.d..

[19] Xiong, C., Zhong, V. and Socher, R.. *Dynamic Coattention Networks for Question Answering*. Proceeding of the 5th International Conference on Learning Representations (ICLR), 2017.