Jointly Learning Remote Relation Extraction and Abstraction with Deep Contextualized Multi-Headed Attention Model

Zhangyu Wang
UMass Amherst
Amherst, MA, US
zhangyuwang@umass.edu

Abstract

Relation Extraction is a core NLP task, but its accuracy is strongly bounded by reference distance. Another challenge is that human annotated training data is too expensive to collect. Existent solutions addressing these two difficulties include deep contextualized embedding ([5]Peters et al., 2018) and distant supervision ([4] Mintz et al., 2009). We established a model that tackles them jointly.

We are motivated by 4 important observations:

(1) while we mostly model semantics and syntax within a sentence, it's similar to model the relationship among sentences within a paragraph where sub-sentence components are treated as words;
(2) by aggregating deep contextualized embedding and multi-headed attention, both semantic and syntactic information of lower levels can be encoded into the learned attention weights and passed to higher levels;
(3) the performance of both relation extraction and abstraction, another closely related task, highly relies on the model's knowledge of semantics and syntax;
(4) abstracts are generated by extracting most important relations (semantic and syntactic) in the text, sometimes even very remote relations.

(1) and (2) means we can learn to simultaneously attend to the content and structure of the given text; (3) and (4) means target parts in abstraction task and in relation extraction task may heavily overlap, and we can jointly learn the two tasks, or train on one task and transfer to another.

This implies that we can 1) hierarchically attend and embed lower level (characters, words, phrases) information into higher level representation (sub-sentence components, sentences, paragraphs) and 2) distantly supervise our model with abstracts instead of human annotations, the former being much easier to obtain in practice.

Therefore, we proposed the Deep Contextualized Multi-Headed Attention Model (Deep-CoMA) that jointly learns remote relation extraction and abstraction by training the model to generate best abstracts from given text-abstract pairs. It not only makes full use of context information, but also expands the scope of training data sources. We also proposed a standard text-abstract dataset for training Deep-CoMA, together with detailed configuration of data pre-processing and formatting.

1. Introduction

Relation extraction is a classical NLP task that attempts to locate entity-relation triads in a given context. Successful extraction requires very comprehensive syntactic and semantic knowledge of the given text. The intuitive assumption is that a model that better encodes and utilizes such knowledge has better performance.

Different from hard-coded programs for extracting information, neural networks address this problem from a synthetic, semantic-rich way. It focuses on understanding the meaning space instead of disassemble texts into single units. A good example is the overwhelming success of word embedding over other word representation models ([3], Mikolov et al.,2013). Unlike traditional methods, CNN and RNN models in NLP tasks demonstrated stunning ability to capture complex semantic information across a span of or even the entire text. Though these models are still troubled by the limitation of sequence length, they vividly proved that the more semantic information the model has, the better it becomes.In addition to semantic knowledge, neural networks can also learn syntax with a probabilistic language model.

There are many ways to include semantic and syntactic information into the task. While the author of ELMo ([5]Peters et al., 2018) suggested that simply replacing the context-independent word representations with contextualized ones will readily introduce improvement in downstream tasks, we found it even more exciting if we move a step further. Consider the structure of a paragraph hierarchically, that is, characters form words, words form phrases, phrases form sub-sentence components, sub-sentence components form sentences, and finally sentences form paragraphs, it's obvious that we can do similar deep contextualized embedding at each level, taking the lower level components as "words" and yielding a context-dependent representation of the higher level components. This idea treats each level recursively, makes it possible to design a general module, and can be independently plugged into existent models.

With this philosophy we designed the Deep Contextualized Multi-Headed Attention Model (Deep-CoMA). It consists of 4 succeedingly stacked layers: a pretrained word embedding module, a sentence encoder, a paragraph encoder, and finally a decoder.

At its base level is a semi-independent pre-trained bi-LSTM encoder-decoder language model module, which learns from corpus by predicting the next character in sequence. When training, we export its encoder and pass training sentences to it, obtaining sequences of word vectors. In this step, information of general, domain-specific semantic and syntactic knowledge is encoded into word representations.

Next we pass these sequences of word vectors to a self-attention layer, where intra-sentence semantic and syntactic relations are learned as attention weights and encoded into the weighted word vectors. Afterwards, we pass the attended sequences to a sentence encoder, which is just like in the first step, a deep bi-LSTM encoder that takes a vector sequence of length L as input in and outputs a hidden state sequence and a deep contextualized representation of the entire sequence that weighted sums the hidden states of every bi-LSTM. Notice the flexibility here: this deep bi-LSTM encoder can either be trained with the big model, or it can be pre-trained on other datasets (especially when they are higher in quality or larger in quantity) and plugged in with weights either fixed or adjustable. For training, only the hidden state sequence is passed to the following layers.

On top of the sentence encoder are identical self-attention and deep bi-LSTM encoder layers, which attend to sentences in the paragraph and transform them again into a hidden state sequence and a deep contextualized representation in the same fashion as in the previous layer.

In the final layer, an LSTM attention decoder is applied. It has the form of a classical attention module ([1]Bahdanau et al., 2015), and takes the hidden state sequence of last layer to calculate context vectors, and the deep contextualized representation is used as initial input to the LSTM.

Finally, we obtain a generated sequence of word vectors. During training, we try to maximize the cosine similarity between our output and the ground-truth abstract word vectors, while during testing, we can either maintain this structure to generate abstract from text, or export the attention block to do relation extraction, because our attention heads learned to identify components . This is also why we call our learning process "jointly".

2. Related Work

There are many approaches to relation extraction. For structured data, rule-based extraction is very natural and effective. Linguistically, Part-of-speech (POS) parsing is efficient in detecting standard syntactic relations like Subject-of, Object-of, etc., but usually fails to understand more complex semantic references. The severest defect of the traditional methods is that they try to first decompose texts into minimum semantic and syntactic units like words, phrases, or nouns and verbs, while in real life the smallest block people understand is sentence. In this sense, Sequence-to-Sequence approaches were introduced to overcome this obstacle ([7]Sutskever et al., 2014). LSTM and forward-only CNNs were powerful tools in this family and achieved better performance: for the former the inner state of the LSTM cell records the context and the latter uses convolution kernels to extract semantic features. Recently, attention model becomes popular ([8]Vaswani et al.,2017). For example, the state-of-the-art model is built on multi-headed self-attention ([9]Verga et al.,2018). Attention mechanism is more powerful in the sense that it attends the entire sequence (while CNNs only attends to local information) and intrinsically supports random "looking back" in sequence (while even bi-directional LSTM only allows looking to direct successor and direct predecessor), together with better computational complexity and parallelism availability.

Another crucial source of semantic and syntactic information is the language model. One can either plug golden syntax parser into existent models ([6]Strubell et al.,2018), remotely supervise model training with external semantic information like a knowledge base ([4] Mintz et al., 2009), or encode language model knowledge directly into word representation ([5]Peters et al., 2018). As to the third approach, researches show that by initializing with vectors generated by models pre-trained on large corpus can improve downstream applications significantly ([5]Peters et al., 2018). The most widely used word embedding like Word2Vec, however, only learns a unique context-independent representation for each word, discards the rich knowledge learned about the language itself during training, and is unable to address problems like polysemy. Some works approached this by introducing subword information (e.g., [10]Wieting et al., 2016; Bojanowski et al., 2017) or learning separate vectors for each word sense (e.g., Neelakantan et al., 2014). But these solutions are still context-independent. Other works focused on learning context-dependent representations like context2vec (Melamud et al., 2016) using LSTM and CoVe (McCann et al., 2017) using Machine Translation model. In a most recent work, ELMo (Embedding from Language Models) ([5]Peters et al., 2018) proposed to learn a deep contextualized word representation that is function of hidden states generated by passing the word through a learned language model. Practically, it is computed as a softmax weighted sum over all inner hidden states of a two-layer character-based bi-directional LSTM ([2]Huang et al.,2015), while the weights can be either pre-trained or learned task-specifically on the fly. According to this paper, the lower layer hidden states are reported to encode syntactic features and the higher layer hidden states to encode semantic and contextual information at sequence level.

3. Approach

3.1. Model Architecture

The overall architecture of our Deep-CoMA model is depicted in Fig 1. We will explain each module's detailed function and setting in the rest of this section.

At the model's ground level is a semi-independent pre-trained bi-LSTM encoder-decoder language model module, which learns from corpus by predicting the next character in sequence. We can 1) use a pre-trained language model unchanged, or 2) fine-tune the language model on our specific dataset, or 3) completely retrain the language model with our own corpus. Theoretically, method 3) should achieve best performance because it's well learned for our target domain, but in practice it's both too computationally expensive (a 6-layer standard bi-LSTM language model requires 2 weeks of training with 3 1080Ti GPUs on a 5G corpus) and unstable (the quality, size and data balance of user obtained corpus has no guarantee and it's highly probable that self-trained language model will fail to converge or overfit). Therefore, the recommended approach is to use a benchmark language model as the starting point and fine-tune it with domain-specific tasks.

After getting the fine-tuned language model, we detach the decoder part and insert the encoder part into our model as the word embedding module ([5]Peters et al., 2018). Like the method introduced in this paper, we will run input sentences through the encoder and we get $l \times L$ hidden state vectors, where l is the number of hidden layers in the encoder, and L is the length of the input sequence. Then a linear layer will learn a weighted sum of these layers of vectors, and output a sequence of vectors of length L, where each vector v_t is the dynamic deep contextualized representation for the word at t. Then, the sequence of word representations will be sent to following layers for further learning.

Like in the ELMo model, there are 3 ways one can use the deep contextualized embedding module. One can 1) use a given vocabulary-to-embedding file to look up the pre-computed embedding vector for each word in the sequence, or 2) construct a task-specific vocabulary file and feed it through the embedding block to obtain a task-specific vocabulary-to-embedding file and look up words in it, or 3) compute representation for each word in the sequence on the fly (treating this module as the first layer of the model), especially when the size of vocabulary is unknown. In this paper, we chose the third method for higher performance.

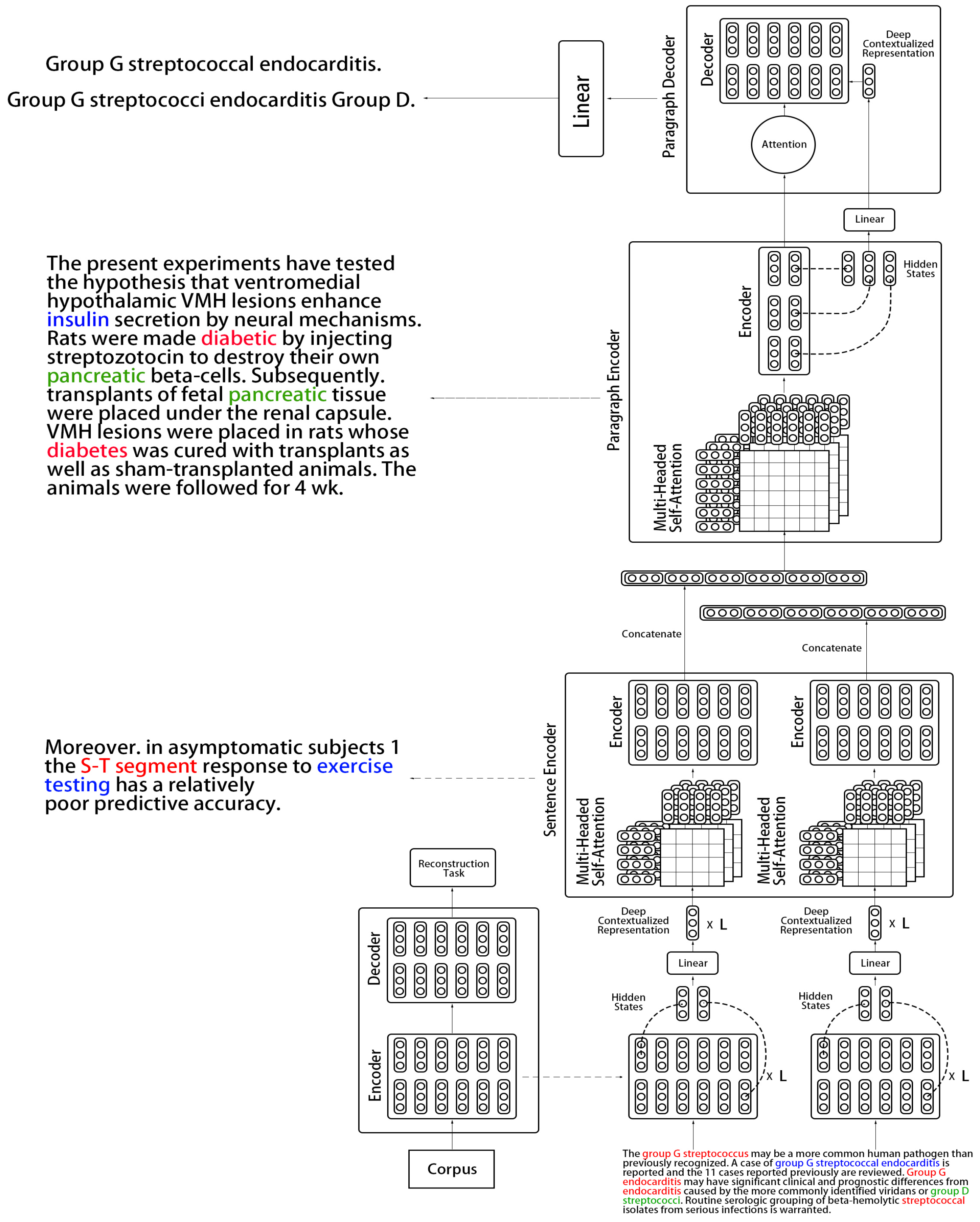
Then we arrive at the sentence encoder module. In this module, a multi-headed self-attention layer will learn to attend to intra-sentence reference pairs. These attention heads will re-weight the input sequences, highlight the important pairs, and pass the attended sequence to the following bi-LSTM encoder. Just like in the deep contextualized embedding module, all hidden states will be extracted and utilized. But in order to leave later layers access to intra-sentence information, we will not sum the hidden states up, but concatenate them. This is of great importance, because in remote relation extraction and abstract generation, sometimes sub-sentence components in two sentences may have strong relation, while the sentences they belong to have only very weak sentence level connection. If we do not expose all these intra-sentence information to following layers, we may lose a good proportion of understanding of text structure.

A trick to mention here is that, to prevent vanishing gradient we add residual short-cut to all bi-LSTM modules. This is also a method to directly pass lower level information to higher layers, and our training will benefit from it because sometimes, information flow across layers is not continuous: word level information may be critical in inferring the relationship between two sentences, whereas the word itself has no significance in the sentence alone.

After we have extracted enough information from sentence level inputs, we proceed to the paragraph encoder. As we have discussed in Introduction, this module is identical to sentence encoder: it takes input sentence vectors as "words" and treats the entire paragraph as a long sentence. With the same fashion, it learns the inter-sentence references and encodes them into hidden states.

But paragraph encoder differs from sentence encoder in terms of its output. After this layer, we need to feed the encoded information of the entire paragraph to a paragraph decoder to generate a reconstructed sequence that minimizes its difference from ground-truth paragraph abstract. For this process, we design a standard attention module [1]. It takes the final output of paragraph encoder as its "previous hidden states" for calculating the context vector, and the initial input for the RNN decoder is the learned deep contextualized embedding of the hidden states in paragraph encoder.

Finally, the output of paragraph decoder will be passed to a module of linear layers. The predicted vectors are considered as word representations, and by looking back at the vocabulary-to-embedding file or feed it again through the


Group G streptococcal endocarditis.

Group G streptococci endocarditis Group D.

The present experiments have tested the hypothesis that ventromedial hypothalamic VMH lesions enhance insulin secretion by neural mechanisms. Rats were made diabetic by injecting streptozotocin to destroy their own pancreatic beta-cells. Subsequently. transplants of fetal pancreatic tissue were placed under the renal capsule. VMH lesions were placed in rats whose diabetes was cured with transplants as well as sham-transplanted animals. The animals were followed for 4 wk.

Moreover. in asymptomatic subjects 1 the S-T segment response to exercise testing has a relatively poor predictive accuracy.

The group G streptococcus may be a more common human pathogen than previously recognized. A case of group G streptococcal endocarditis is reported and the 11 cases reported previously are reviewed. Group G endocarditis may have significant clinical and prognostic differences from endocarditis caused by the more commonly identified viridans or group D streptococci. Routine serologic grouping of beta-hemolytic streptococcal isolates from serious infections is warranted.

Figure 1. Overall Model Architecture

language model, we can get which word this representation points to.

3.2. Model Usage

After we have successfully trained our Deep-CoMA model, we can use it in several ways. Firstly, we can use it directly for abstraction task. Secondly, as we have emphasized in Introduction, our model is capable of jointly learning to generate abstract and to extract remote relations. Therefore, we can simply remove the paragraph decoder part and add a relation extraction module on, and fine-tune it to perform relation extraction tasks. Thirdly, which is very interesting, we can drag the multi-headed self-attention modules out, fix their weights, and use them to generate annotated relation extraction task data. The mechanism is: our model has learned on abstraction task to identify closely related pairs within and across sentences. Therefore, we can use these attention heads to assign relation strength to words in a sequence, and the pairs with intense enough strength will be considered relation pairs. Through this approach, we can learn on a corpus of moderate size, and use it to automatically annotate other corpora, generating more data suitable for leanrning.

4. Experiment

4.1. Dataset and configuration

In experiments, we used the bio-medical dataset collected by UMass Amherst's IESL (Information Extraction and Synthesis Lab) private server and maintained by Patrick Verga. It contains 1 million bio-medical reports and corresponding abstracts, total size up to 25G. For this experiment, we sampled 50000 reports and abstracts out of it. We used NLTK for word tokenization. However, the hardest part of data preprocessing is sentence segmentation. Unlike general-purpose texts, bio-medical reports contain an extremely big amount of biological, medical and chemical terms that use period as separator (for example, **so20.w.Na+**), or decimal digits (for example, 0.812). These periods are non-differentiable from ordinary sentence end indicator periods, causing great problem in correct sentence segmentation, which is critical to successful learning.

Another problem is the variance of sentence length. Statistics of the entire dataset shows that the length of sentences disperses over a very wide range from 50 characters (notice our deep contextualized embedding module is character-based) to 1000 characters (where there are some hundred-of-character-long terminology used). For efficient batch computation, we must fit our data into a fixed-size matrix, and that makes clipping longer sentences and fitting <NULL> for missing slots in shorter sentences necessary. This caused fatal problem to our training, because on the one hand, many sentences are not fully represented, and on the other hand (which is the main cause of bad learning result) there are too many <NULL> tokens sent to the model. The consequence of having these two problem is that: though during learning, the model's training and validation loss goes to as low as $1e^{-5}$, when we do predictions, the most frequent generated word is no doubt <NULL>.

This suggests that we still need to refine our dataset. Of course human annotation will be the easiest way to significantly improve the quality of the dataset, but as is discussed, it's way too expensive. Therefore, the biggest challenge now is how we can generate a better dataset automatically.

4.2. Experiment Results

Because the model structure is completely designed from paper ideas and has no off-the-shelf implementation for any part of it, and many modules require exposing module inner states to other modules (like the deep contextualized embedding module), we implemented the entire model, including helper classes and data pre-processing toolkit from scratch with NumPy and Pytorch. Though there have been bugs both in software and hardware levels, we finally successfully finished this job. Readability and documentation has always been well kept in mind throughout implementation, but re-structure and refining comments should still be done later.

As has been partly discussed in **4.1.**, our model during training soon overfits. We can look at the training and validation loss history.

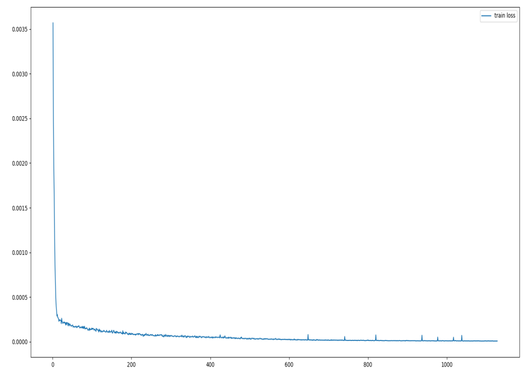


Figure 2. Training loss over iterations

The cliffy drop in training loss at very early iterations is a prominent indicator that our model is overfitting. But the problem we are facing is not simply overfitting. As we can see, after a dozen of epochs, validation loss has also reached a stably low level. In common settings, this should imply that our model has at least learned something on the training set. But when we used the learned model to generate abstract text out of texts, we saw the output to be

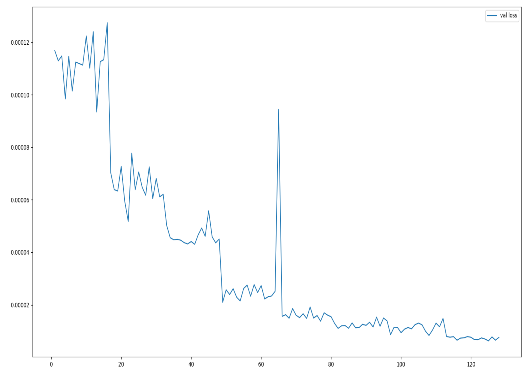


Figure 3. Validation loss over epochs

disappointingly almost all <NULL>. We logged weights and scores along our model's pipeline, and discovered that though the model was learning, it assigned very close scores to completely different words, while <START>, <END> and <NULL> gain the highest scores. This can be explained in two ways: firstly, our 50000 paragraph dataset is too small (though that's already reached the upper bound of affordable computation and storage resources) for so deep a model; secondly, because bio-medical reports use a very different and very wide range of terminology, the reoccurrence of words and phrases is supposed to be very low. That means, many words and terms may only appear once or twice across the entire dataset. That causes the model to treat them equally (assigning close scores). Contrarily, <START>, <END> and <NULL> are under such setting becoming the most frequent and most learned vocabulary, especially <NULL>. Because of the padding needed to fit shorter sentences into the batch matrix discussed in **4.1.**, <NULL> appears hundreds of times more frequently than other words, and there is no doubt our model quickly learned to always predict <NULL>.

Though our experiment is not so encouraging as expected, but it points out the correct approach we shoud take to improve our design. Firstly, the segmentation of sentences must be solved beforehand, because falsely separated sentence will corrupt the entire model. Secondly, it may be better to research if mini-batch SGD in this setting could be replaced with some length-variable SGD method, or there could some fixed-length representation of variable-length sentences and paragraphs other than weighted sum of component vectors, which simultaneously retains the original positional information. Convolutional modules may be a breaking point. Finally, we need to research if some of the model parts could be removed or simplified without affecting its learning capacity too much, so that we can reduce the run-time memory and computation pressure, so that we can increase the size of both the entire dataset and batches fed every iteration.

5. Conclusion

In this project, we have designed a comprehensive model that attempts to jointly learn remote relation extraction and abstract generation. By hierarchically stacking deep contextualized embedding and multi-headed attention blocks, we obtained a powerful model that exploits information of input from character to paragraph level. However, the problems we encountered in dataset quality stopped us from further exploring its strengths and weaknesses. In the future, we should decompose this design into smaller, unit models, train and evaluate them on better prepared dataset before directly experiment on so deep a model. Dataset matters first and always start from simpler architecture is the biggest two lessons I learned.

Acknowledgments

Finally, I'd like to express my immense gratitude to Professor Erik Learned-Miller and all TAs. Through this course I learned the most essential knowledge and techniques one machine learning scientist need to know about neural networks, and I practiced a lot on how to successfully propose, design, implement, carry out, analyze and finally report a neural network research project. This experience encourages me greatly to continue with more serious research projects, and prepares me for harder challenges.

I'd also like to thank Professor Andrew McCallum and his PhD student Patrick Verga for their kind assistance and guidance, including dataset, reference papers and lab computation resources.

References

[1] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014.

[2] Z. Huang, W. Xu, and K. Yu. Bidirectional LSTM-CRF models for sequence tagging. *CoRR*, abs/1508.01991, 2015.

[3] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc., 2013.

[4] M. Mintz, S. Bills, R. Snow, and D. Jurafsky. Distant supervision for relation extraction without labeled data. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2 - Volume 2*, ACL '09, pages 1003–1011, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics.

[5] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer. Deep contextualized word representations. In *Proc. of NAACL*, 2018.

[6] E. Strubell, P. Verga, D. Andor, D. Weiss, and A. McCallum. Linguistically-informed self-attention for semantic role labeling. *CoRR*, abs/1804.08199, 2018.

[7] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3104–3112. Curran Associates, Inc., 2014.

[8] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc., 2017.

[9] P. Verga, E. Strubell, and A. McCallum. Simultaneously Self-Attending to All Mentions for Full-Abstract Biological Relation Extraction. *ArXiv e-prints*, Feb. 2018.

[10] J. Wieting, M. Bansal, K. Gimpel, and K. Livescu. Charagram: Embedding Words and Sentences via Character n-grams. *ArXiv e-prints*, July 2016.