Coreferent Mention Detection using Deep Learning

Aditya Barua
abarua@

Piyush Sharma
spiyush@

Kevin Clark
(Mentor)
kevclark@

Abstract

A mention may or may not be coreferred elsewhere in the document. Identifying those mentions that are corefered (called coreferents) is an important step in many NLP tasks, like coreference resolution. To classify a mention as singleton or coreferent using just one sentence is a challenging problem, but previous work suggests that there are cues in a sentence which can be used to predict if a mention it contains is corefered elsewhere in the document. We approach this problem in two different ways. First, we try to a classify a mention candidate extracted by rules-based methods as a coreferent or singleton using various hand-crafted features from literature and a Feed Forward Neural Network (FFNN). Second, we approach this problem as identifying coreferent mention boundaries in input sequences using Recurrent Neural Networks. This is a two step process where we first detect mention heads, and then the mention boundaries. Both these parts are trained and evaluated independently. The second approach removes our dependency on an upstream rule-based mention extractor and hand-crafted features for classification, some of which are expensive to compute. Our hypothesis is that the second approach would be able to learn those hand-crafted features (and more) automatically and perform better at the task. We observed that both of our approaches outperformed the baseline logistic regression model which uses all the hand-crafted features of the first approach, showing that deep neural networks can be used effectively for coreferent mention detection. We also observed that the presence of hand crafted features from literature helps and the first approach (FFNN) outperformed the second (RNN) by approximately 3 F_1 points. We also compare our results to published results for the coreferent mention detection task.

1 Introduction

Once an entity has been introduced in a text or a conversation, it may be referred to multiple times later or never again. Consider the following document with just two sentences:

[Mary]$_1$ was at [her apartment]$_2$ reading [a book]$_3$. [She]$_1$ finished [it]$_3$ at 8 PM.

[She]$_1$, which refers to [Mary]$_1$, and [it]$_3$, which refers to [a book]$_3$ are all **coreferent** mentions because they refer to other entities within this document. But the mention [her apartment]$_2$ is not referred to again. Such a mention is called a **singleton**. It is a challenging and important task to accurately separate out coreferent mentions from singleton mentions. A number of NLP tasks use mention detection as first step, for example: the core NLP problem of coreference resolution – identifying which entity a mention refers to. Identifying and filtering out singleton mentions reduces the search space and hence can improve the accuracy of downstream coreference resolution. In this project, we use two deep neural networks based approaches to detect coreferent mentions. We find that both models perform better than a baseline logistic regression model (provided by our mentor, Kevin Clark), surpassing it in terms of F_1-score. We show that our models also perform competitively with respect to the published literature which looks at this problem.

2 Background and Related Work

There has been some but not a very large amount of previous work on the topic of singleton detection for mention filtering. To the best of our knowledge, there have been three key papers on this topic using the CoNLL-2012 Shared Task data. (Recasens et. al., 2013) use syntactic and semantic features of the mention to predict singleton and coreferent mentions starting from rule based mentions (all Noun Phrases). These features cover the internal morphosyntax of the mention (what type of quantifier it is, is it a pronoun? etc.), the grammatical role of the mention (the verbal role of the mention and the semantic environment of the mention (presence of negation, presence of modality etc) include features. They build a *lifespan model*, which is a logistic regression model using these morphosyntactic features show that such a model can accurately separate singleton mentions from coreferent mentions.

(Haagsma et. al. 2016) also start from rule based mentions and build a deep neural network for predicting whether a mention is a singleton or not. They use word embeddings for the different words in the mention as features. They find that such a model performs well on the mention filtering task. They also present a variety of results on how the network architecture and model parameters affect the performance of their model. (Moosavi et. al. 2016) use shallow features of the words and an SVM with a polynomial kernel to predict singleton mentions. The features in their model include lemmas of all the words included in the mention, Part-of-speech tags, Named-entity tags, the mention string and features about the relationship of the mention to the document such as whether the complete string of the mention appears elsewhere in the document.

3 Approach

Our project takes a two-pronged approach to the problem of coreferent mention detection:

1. **Given a candidate mention, classify it as coreferent or singleton.** There are methods proposed in literature to extract mentions in text, e.g. (Lee et. al. 2011) uses a rule-based approach. Also, the CoNLL shared task dataset contains greedy mentions. But these mentions could be singletons or coreferents. Our first approach attempts to identity the coreferents amongst these candidate mentions. We encode these mention sequences into a fixed length vector using hand-crafted features successfully applied in prior work (Recasens et. al., 2013), and train a powerful Feed Forward Neural Network (FFNN). Section 3.3 discusses this approach.
2. **Predicting coreferent mention boundaries in input sequences.** If we want to remove dependency on a candidate mention extractor and hand-crafted features, we need to process all tokens in the input sequence, and identify coreferent mention boundaries. Another advantage is that context from the entire sentence would now be available, instead of just the local context for the candidate mention to be classified. To process the variable length input sequence, and identify coreferents therein, we use a Recurrent Neural Network (RNN). Section 3.4 discusses this approach.

3.1 Hypothesis

Our hypothesis is that the second approach above that uses a RNN would be able to leverage the global context of the entire input sentence, and learn complex representations of every word's context. This should overcome the need for hand-crafted features used in the first approach above. While at the same time, we push the limits of what can be achieved with those hand-crafted features using a deep FFNN with fully-connected non-linear hidden layers.

3.2 Description of the data

We use the 2012-CoNLL shared task data for training and evaluating our models. This data and some data extraction helper scripts were provided to us by our mentor, Kevin Clark. This data has documents from various sources. Each constituent sentence in a document has its words annotated with Parts of Speech (POS) tags, Named-entity recognition tags (NER) tags. In addition, greedy mentions for each document are available as a separate list. This list of greedy mentions include all noun phrases, all pronouns and all named entities. Finally, all the mentions which are actually coreferent, i.e they are mentioned somewhere else in the document are available as a list of gold mentions. Here a breakdown of the dataset:

Dataset	Documents	Greedy mentions	Coreferent mentions
Train	2,802	478,286	152,558
Dev	343	59,775	19,155
Test	348	61,881	19,764

In addition, this dataset is also annotated with the following additional features:

1. **Word level features**: POS, NER tags for each word.
2. **Mention level features**: 6 features for each greedy mention: mention type, is pleonastic, contained in other mention, contained in other mention with same head, dep parent, dep relation.
3. **Recasens features**: 11 features such as animacy, person, number, position etc. (Recasens et. al 2013).

3.3 Predicting whether a candidate mention is coreferent

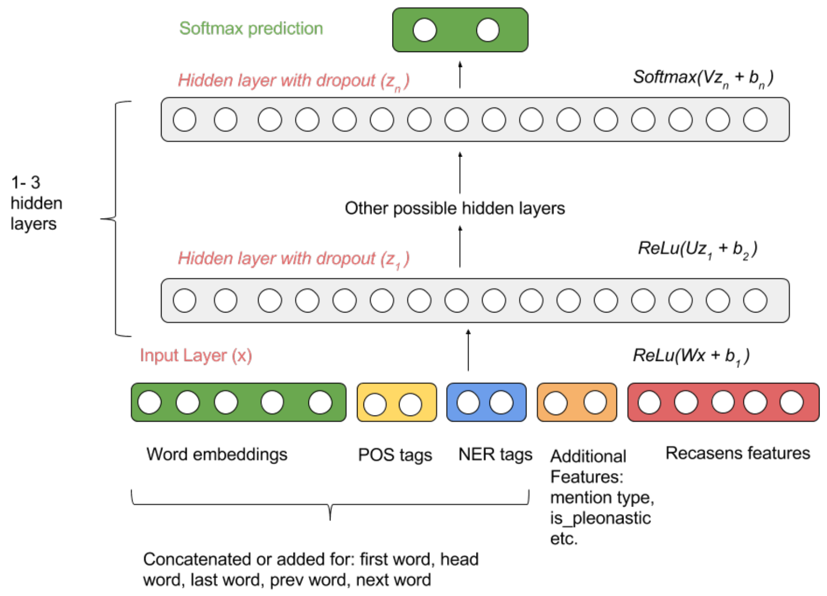

Softmax prediction
Hidden layer with dropout (z_n)
Softmax($Vz_n + b_n$)
1- 3 hidden layers
Other possible hidden layers
Hidden layer with dropout (z_1)
ReLu($Uz_1 + b_2$)
Input Layer (x)
ReLu($Wx + b_1$)
Word embeddings
POS tags
NER tags
Additional Features: mention type, is_pleonastic etc.
Recasens features
Concatenated or added for: first word, head word, last word, prev word, next word

Figure 1: Feed Forward Neural Network for classifying a candidate mention as coreferent or not.

We use greedy mentions in the CoNLL 2012 shared task data as our pool of input candidate mentions. Each mention to be classified is encoded into a fixed length feature vector using the all the features available in the input data (See Section 3.2). We get the word vectors for the first, last, previous and next words, along with the head word for each greedy mention (Figure 1). These are all concatenated to form a $5d$ length vector where d is the dimensionality of the word embeddings. Next, word vectors for each categorical feature such as the NER tag, mention type and the Recasen's features are appended to the feature vector. These are initialized to random values and updated during training.

3.4 Identifying coreferent mention boundaries in sentences

To detect coreferent boundaries in input text without dependence on an external candidate mention extractor, a token-level classifier is required. However, a simple binary classifier that assigns a label of coreferent token or not, won't be adequate for this problem, because this will not be able to detect mention boundaries when two mentions are adjacent or nested. Hence we developed a two-step system (see Figure 2):

Step 1: Detect ***mention-heads*** *in input sequences*
The head of a mention is defined by the dependency parse structure of the mention phrase. The intuition is that the mention-head should have distinct characteristics (in terms of neighboring POS tags, for example) as compared to the other tokens in the mention, or tokens outside any mention. Also, previous research has shown that simply knowing the mention heads more accurately is sufficient to enhance performance of coreference resolution systems. (Peng et. al. 2015).

Step 2: Given a coreferent mention-head, detect ***mention boundaries*** *in the sentence*
The intuition is that a neural network should be able to assign a binary label to every token in the sequence as to whether that token belongs to the mention corresponding to the input mention-head. Once we have all the tokens in a mention, the boundary is defined by the first and the last tokens. Please note that we fill in any missing predictions to construct the longest mentions sequence.

3.4.1 Word features

As opposed to Section 3.3, hand-crafted feature for mention candidates are not available here. We also decided to not use features derived from constituency parsing for this model because that is expensive. Instead we rely on the RNN to learn the required features during training on complete sentences. Each word in the sentence is featured by concatenating word-vectors for the lowercased word, POS tag, Named Entity (NER) tag and original case of the token.

We use 100 dimensional word-vectors for all these features. Vectors for words are initialized with pre-trained embeddings (GloVe, Pennington et. al.), and for other features (e.g. POS tag) are initialized randomly. All word-vectors are updated during training.

Please note that for the second step (boundary detection), a binary feature corresponding to whether the current token is a coreferent's head or not, is appended to the features above. This feature is set to 1 for exactly one token in an input sequence for boundary detection. If multiple heads were found in the sentence in step 1, they all need to be processed separately in step 2. Please see Figure 2.

3.4.2 Model architecture

The same model architecture is used for both steps (i.e., head and boundary detection). They are trained and evaluated independently using their corresponding train and dev datasets, which are extracted from the CoNLL dataset used for this project. The only difference between the two models is that the second one has an extra binary feature for each token, which is only set to 1 for the mention head. In the description of models below, "label" refers to coreferent mention head, or coreferent mention token for steps 1 and 2 respectively.

Model 1: RNN only

The basic model is a Recurrent Neural Network (RNN) which sequentially assigns a binary label to each token. We use a bidirectional RNN to allow local context on both sides of a token be available for its classification. The forward and backward outputs at each time step are concatenated and projected to output logits using a fully connected layer. Please see Figure 5 in Appendix.

Model 2: Model 1 + final output states

Following the intuition that a "summary" of the entire sentence should help in classifying a token, we extended the model to use the final states of the RNN. So, in this version (Figure 6 in Appendix), we concatenate the sum of forward and backward outputs of the bidirectional RNN with the sum of final forward and backward states. The concatenated vector is then projected to output logits.

Model 3: Model 2 + feed-forward network with a hidden layer

Our final model is a further extension of the previous model. It adds a small fully-connected hidden layer before projection to the final output logits (Figure 3). This non-linear hidden layer will allow complex interactions between the RNN outputs (local context) at a time step with the final RNN states (global sentence context).

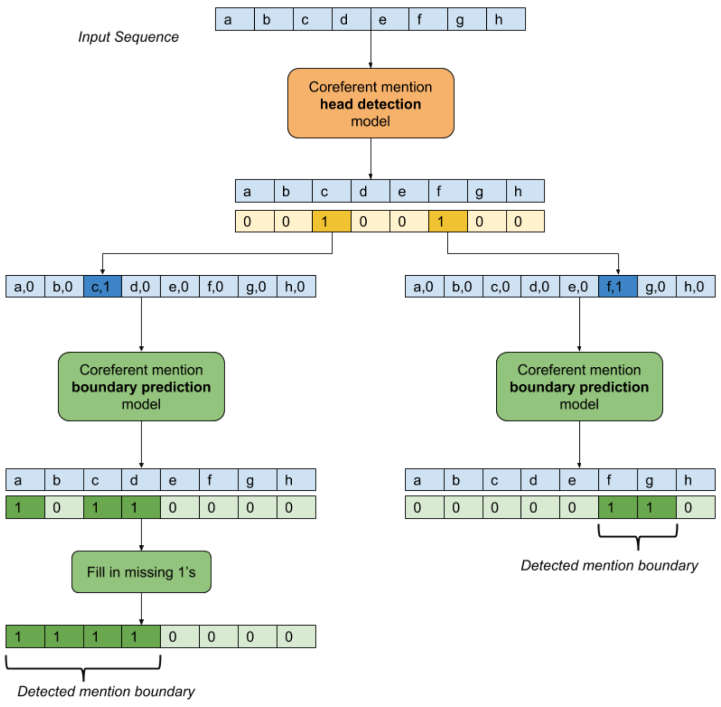

Input Sequence
a b c d e f g h
Coreferent mention head detection model
0 0 1 0 0 1 0 0
a,0 b,0 c,1 d,0 e,0 f,0 g,0 h,0
a,0 b,0 c,0 d,0 e,0 f,1 g,0 h,0
Coreferent mention boundary prediction model
1 0 1 1 0 0 0 0
0 0 0 0 0 1 1 0
Fill in missing 1's
1 1 1 1 0 0 0 0
Detected mention boundary
Detected mention boundary

Figure 2: Two step process to identify coreferent mention boundaries in a sequence. First, the mention heads are detected. Then for each head, the mention boundaries are identified. The two models are trained and evaluated independently, then put together for the end-to-end system.

4 Experiments

4.1 Baseline

Our baseline is a logistic regression that uses every feature described in Section /refsec:data-desc. It was provided by our mentor, Kevin Clark. This model has a precision of 48.41, a recall of 90.01 and an F_1-score of 63.31 on the test set. In this model, the threshold for the probability to define a true class has been set so that the recall is at or higher than 90.0. This model is also built on greedy mentions and attempts to predict whether a mention is coreferent or not.

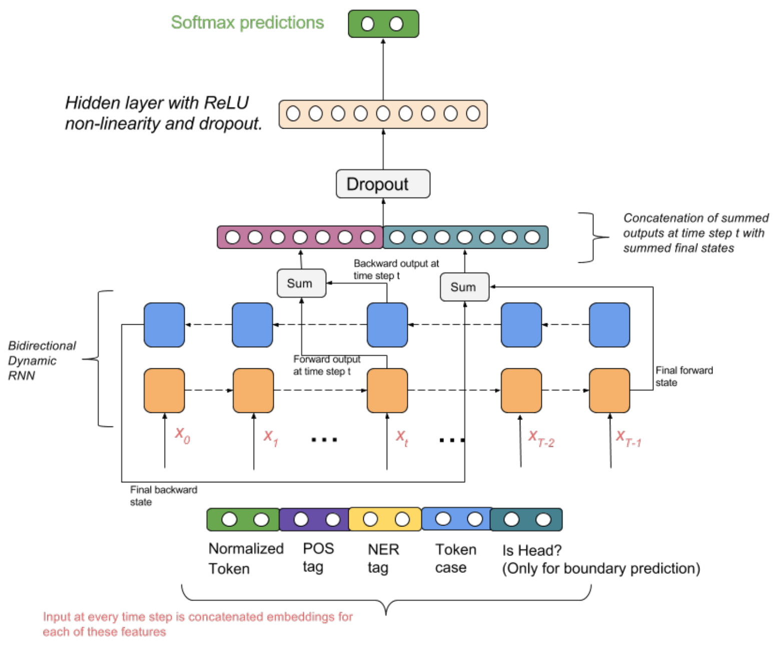
4.2 Approach 1: Coreferent Mention Detection from Greedy mentions

This section describes our experiments with a Feed Forward Neural Network (FFNN) for coreferent mention detection from greedy mentions. Our experiments were over two broad areas:

Neural Network Architecture	Hyper-parameters Optimization
Source and dimensionality of word vectors	Cost function
Number of hidden layers	Learning rate
Activation functions: ReLu, tanh, sigmoid etc.	Optimizer
Size of hidden layers	
Dropout probabilities	

Because there are so many different hyper parameters to be tuned, we adopt the framework of (Haagsma et. al. 2016) by adopting a baseline model. To determine the effect of changing one of the hyper parameters, we keep all the other hyper parameters constant and thereby measure the specific effect of tuning a single parameter. We evaluate each of these models on the dev set in order to identify the best model. In addition to the model hyper-parameters, another parameter that can be varied is the probability threshold for making the final decision.

Baseline FFNN model: The baseline model (B) has the following parameters: GloVe 6B tokens and 300d word vectors; 1 hidden layer; ReLu activation function; 400 hidden units; dropout



Softmax predictions
Hidden layer with ReLU non-linearity and dropout.
Dropout
Concatenation of summed outputs at time step t with summed final states
Backward output at time step t
Sum
Sum
Bidirectional Dynamic RNN
Forward output at time step t
Final forward state
x_0
x_1
...
x_t
...
x_{T-2}
x_{T-1}
Final backward state
Normalized Token
POS tag
NER tag
Token case
Is Head? (Only for boundary prediction)
Input at every time step is concatenated embeddings for each of these features

Figure 3: The sum of forward and backward outputs of the bidirectional RNN is concatenated with the sum of final forward and backward states. The concatenated vector is then projected to output logits using a fully connected hidden layer.

probability=0.5; Xavier initializations for weight matrices; C=1; learning rate=0.0001; Optimizer=AdamOptimizer; Batch size=2000. Figure 4 in the appendix shows that the precision recall curve for this model is always at or above the baseline for all value of recall.

Source and dimensionality of word vectors: We explored multiple GloVe datasets and evaluated their effect on metrics on the dev test (Table 1). We found that the dev F1-score increases with increasing dimensions for the word vectors up to the point of 200 dimensional word vectors. But beyond that, the dev F1 score drops a little bit. Higher dimensional word vectors encode more semantic information and hence provide a more granular and nuanced representation of the word. Such a granular embedding is likely to work better than a coarser embedding which likely loses key information, but added granularity beyond a certain limit might lead to overfitting. We also saw that using more tokens leads to better results. This is probably because more words from the mention can now be mapped on their real embeddings instead of being mapped onto the unknown word emebdding ”UNK”.

Number and size of hidden layers:In Table 3, we see that the best results are achieved by a single layer network with a moderate number of hidden neurons (400). One hypothesis for why this happens is that true function which of the features which determines whether a mention is coreferent or not is not extremely complex. This explains why logistic regression performs so well because the output classes are possibly linearly separable. As a result, complex, multi-layered networks with lots of hidden layers overfit the noisy training data and don’t perform very well. The sweet spot of the number of neurons for a single layered network seems to be around 400 and less or more neurons than that don’t do very well. Multi-layered networks in general don’t seem to be performing very well in comparison to the single layered network.

Activation functions: In Table 2, we show the results from experiments with 4 different activation functions. ReLU provides the best results. This is possibly because ReLu doesn’t suffer from the vanishing/exploding gradients problem which a lot of the other activation functions do.

Dropout probabilities: Network complexity might be over fitting the data. That may be the reason why high dropout seems to help our models. Table 4 For both a small network with 400 neurons and

Dataset	Precision	Recall	F1-score
Wiki 6B 50d	71.41	75.40	73.35
Wiki 6B 100d	71.09	76.02	73.47
Wiki 6B 200d	71.04	76.39	73.61
Wiki 6B 300d (B)	72.47	74.68	73.55
Wiki 42B 300d	**71.68**	**75.77**	**73.67**

Table 1: Source and dimensionality of embeddings.

Activation function	Precision	Recall	F1-score
ReLU (B)	**72.47**	**74.68**	**73.55**
Sigmoid	79.12	63.49	70.45
tanh	81.68	58.66	68.28
elu	74.28	70.48	72.33

Table 2: Different activation functions.

Dataset	Precision	Recall	F1-score
[50]	71.42	74.59	72.97
[100]	71.63	74.63	73.10
[400] (B)	**72.47**	**74.68**	**73.55**
[500]	72.46	74.66	73.54
[1000]	71.23	73.71	72.45
[50,50]	79.34	62.60	69.98
[400,400]	72.46	74.13	73.29
[1000, 1000]	72.21	74.03	73.11
[50,50,50]	69.82	75.16	72.39
[400,400,400]	72.18	74.48	73.31
[1000,1000,1000]	77.35	68.16	72.46

Table 3: Number and size of hidden layers. [a, b, c] indicates a network with 3 hidden layers of size a, b and c respectively.

Network size	Drop prob	Precision	Recall	F1
[400]	0.25	72.12	74.22	73.16
[400] (B)	0.5	72.47	74.68	73.55
[400]	**0.75**	**72.33**	**75.08**	**73.68**
[1000, 1000, 1000]	0.25	71.88	73.35	72.61
[1000, 1000, 1000]	0.75	72.77	74.98	73.68

Table 4: Network size and dropout probability.

Learning rate	P	R	F1
0.1	98.82	8.34	15.39
0.01	70.87	74.03	72.41
0.001	72.13	73.34	72.73
0.0001	72.47	74.68	73.55
0.00001 (B)	**72.47**	**75.68**	**74.55**
Adaptive learning rate	76.75	69.71	73.06

Table 5: Learning rates.

a large 3-layered network with 1000 neurons in each layer, a high dropout value of 0.75 improves performance.

Learning Rate: Experiments with the learning rate were very interesting and quite informative (Table 5). It is this category of experiments which gave us the best performing model (learning rate=0.00001). The low optimal learning leads us to think that the cost function for this problem is very non-convex and moving around by too much in the feature space leads to quickly moving away from the global optimum. Also, there are likely to be many local optima which results in runs with larger learning rates ending up stuck in a local optima. Finally, we also tried a learning rate with an exponential decay. Such a learning rate starts out large (initialized at 0.0001) and then decays exponentially with each iteration. The rationale is that as we approach the optimal value, we take smaller and smaller steps so as to not overshoot it. This performed better than the other learning rates, but not as good as the best learning rate of 0.00001.

Optimizer We tried 3 different optimizers, the Adam Optimizer, FtrlOptimizer and the AdaGradOptimizer. The AdamOptimizer provided the best results with a resultant F1-score of 73.55 for the baseline FFNN. The AdaGradOptimizer performed quite poorly in comparison resulting in an F1-score of 68.22.

4.3 Approach 2: Identifying coreferent mention boundaries in sentences

The RNN state is fixed at 512 in all case, and the fully connected hidden layer (if used) in feed-forward network is 64. The maximum sequence length is set at 128, dropout rate is 0.5, gradient are clipped to a global norm of 5.0, and batch size for training is 32. 100 dimensional GloVe (Pennington et. al.) word-vectors are used. Loss computation is weighted to penalize False Negatives more than False Positives to counter label bias (there are far more “not a coreferent mention head” tokens than actual mention heads, and we want a reasonably high recall for the downstream task).

As noted in section 3.4.2, the two models involved in the two steps to identify mention boundaries from sentences are trained and evaluated separately. The models are trained on train dataset, and

	Head Detection			**Boundary Detection**			**End-to-end**		
	Pre	Rec	F1	Pre	Rec	F1	Pre	Rec	F1
Model 1	66.30	81.90	73.28	83.04	82.18	82.61	59.81	72.36	65.49
Model 2	67.21	82.33	74.00	86.52	85.66	86.09	62.46	75.01	68.16
Model 3	67.76	83.06	74.64	89.16	88.31	88.73	64.13	77.10	70.02

Table 6: The performance of the head detection and boundary detection models evaluated independently on the test set. The last set of columns show the end-to-end performance on the test set when the two models were put together by feeding the output of head detection to boundary detection. Each row is a different model described in Section 3.4.2, with increasing complexity. Model 1: RNN only; Model 2: Model 1 + RNN final output states; Model 3: Model 2 + feed-forward network with a hidden layer

hyperparameters tuned on dev dataset. The two models were then put together by feeding output of head detection module into boundary detection. Table 6 shows the final performance of all model architectures on test dataset.

We see that Model 2 performs better than Model 1, and Model 3 better than Model 2 at both the tasks, and also the end-to-end task. So, adding the sentence context with the final hidden states helped, and so did adding a hidden layer. This was expected because as the model gets increasingly complex, it can learn more complex mappings from input sequences to output labels.

Output analysis: Example where we did well
Consider the following input sentence from the test set. Mentions boundaries are marked with "[]" and the head is in bold font.

Even as [the **paper**] asked for [the **public** 's] support [**it**] was unable to answer [**its**] questions .

Head detection step predicted all the heads correctly (highlighted in bold):

Even as the **paper** asked for the **public** 's support **it** was unable to answer **its** questions .

Boundary detection step also correctly predicted the following boundaries for each head (highlighted in bold):

Even as [the **paper**] asked for the public 's support it was unable to answer its questions .
Even as the paper asked for [the **public** 's] support it was unable to answer its questions .
Even as the paper asked for the public 's support [**it**] was unable to answer its questions .
Even as the paper asked for the public 's support it was unable to answer [**its**] questions .

It is interesting to note that the model was able to correctly predict that "support" and "questions" are not part of the mention, though they are part of the noun phrase that contains the mention.

Output analysis: Example where we did well with nested mentions
Consider the following sentence with *nested mentions*.

right now [**I**] mean [**I**] do want to keep [[**my**] American **citizenship**]

Head detection step predicted the following heads (highlighted in bold):

right now **I** mean **I** do want to keep **my** American **citizenship**

Boundary detection step predicted the following for each head (highlighted in bold)

right now [**I**] mean I do want to keep my American citizenship
right now I mean [**I**] do want to keep my American citizenship
right now I mean I do want to keep [**my**] American citizenship
right now I mean I do want to keep [my American **citizenship**]

So our approach works for nested mentions too.

Output analysis: Example were head detection failed
Consider the following sentence:

A big reason for the chemical price retreat is overexpansion .

This has no coreferent mentions. But the head detection model predicted:

A big reason for the chemical price **retreat** is **overexpansion** .

This is because the head detector does not have visibility into the entire document, and hence, cannot know for sure if these entities were coreferered elsewhere in the document. This is the limitation of working at the sentence level, and hurts the overall precision.

Output analysis: Example were boundary detection failed
Consider the following sentence with the heads:

[**CNN**] 's viewer habits have been molded by **its** format .

For the first head above, boundary detection model predicted:

[**CNN** 's] viewer habits have been molded by its format .

This has an extra token which is incorrect.

5 Overall Results

In this section, we present our overall results and compare them to the literature. We see that both the FFNN and the RNN outperform the baseline logistic regression model. [1] Please note that for some papers in literature that we are comparing with only reported their results on the dev set. So, we report our results on both dev and test, and compare the appropriate numbers with past work.

Table 7: **test** CoNLL 2012-shared task data

Dataset	P	R	F1
Baseline logistic regression	48.42	90.01	63.31
FFNN (Sec. 3.3)	74.16	76.16	75.15
RNN (Sec. 3.4)	64.13	77.10	70.02
Greedy mentions	29.89	93.58	45.31

Table 8: **dev** CoNLL 2012-shared task data

Dataset	P	R	F1
Baseline logistic regression	47.96	89.99	62.57
FFNN (Sec. 3.3)	76.12	71.24	73.60
RNN (Sec. 3.4)	62.54	76.07	68.64
Recasens et.al. 2013	72.20	67.88	69.97
Haagsma et. al. 2016	7.25	63.54	70.54
Moosavi et. al. 2016	83.76	71.76	77.30

6 Conclusions and Future Work

Our hypothesis was that an RNN model that processes the entire input sentence would perform better than a FFNN that uses hand-crafted features from literature for previously extracted candidate mentions. We expected that the RNN would learn those features (and more) from the input sequence itself. But we saw in our results above that using those hand-crafted features with a powerful feed-forward neural network outperformed by a few F_1 points the RNN model which doesn't use those features. However, both models outperformed the baseline with a big margin.

[1]When evaluating on the dev set, we present the results for both models when we just predict that a mention is coreferent if prob $>= 0.5$.

It is worth noting that the RNN approach worked, is promising and has many advantages over the other approach:

1. It is not limited by the performance of an upstream candidate mention extraction system.
2. It is applicable to tasks where rule-based candidate mention detection is difficult, for example, nested named entity recognition [Finkel et. al. 2009].
3. Constituency parsing is not required. This is a very expensive step required to generate a CoNLL feature (is noun phrase). This step could potentially become the bottleneck in performance of a latency sensitive system.

Further extensions of the RNN model are possible. Adding document level features, such as whether the tokens of a mention occur in another sentence in the document (Moosavi et. al.), or an RNN that encodes the entire document, could lead to large improvements possibly beating state of the art.

7 Acknowledgments

We would like to thank everyone on the teaching staff of CS224N for putting together a great class this year. We are thankful to Kevin Clark, our project mentor for providing feedback regularly. We used data and baselines provided by Kevin. We used utility functions and code structure from homeworks in our project. So, a big thanks to everyone on the teaching team who contributed to it.

8 Contributions

Both of us brainstormed on both the approaches above. Aditya implemented the FFNN approach (described in Section 3.3), and Piyush implemented the RNN approach (described in Section 3.4). We both communicated regularly with our project mentor (Kevin Clark) to share updates and get feedback.

References

[1] Marta Recasens, Marie-Catherine de Marneffe, and Christopher Potts. 2013. The Life and Death of Discourse Entities: Identifying Singleton Mentions. In Proceedings of the NAACL.

[2] Heeyoung Lee, Yves Peirsman, Angel Chang, Nathanael Chambers, Mihai Surdeanu, Dan Jurafsky. 2013. Stanfords Multi-Pass Sieve Coreference Resolution System at the CoNLL-2011 Shared Task. In Proceedings of the CoNLL-2011 Shared Task, 2011.

[3] Kevin Clark and Christopher D. Manning. Improving Coreference Resolution by Learning Entity-Level Distributed Representations. In Proceedings of the ACL.

[4] Kevin Clark and Christopher D. Manning. Deep Reinforcement Learning for Mention-Ranking Coreference Models. In Proceedings of EMNLP.

[5] Haagsma, Hessel. Singleton Detection using Word Embeddings and Neural Networks. (2016).

[6] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation.

[7] Bengio, Yoshua.,Learning deep architectures for AI,Foundations and trends in Machine Learning,2,1,1-127,2009

[8] Moosavi, Nafise Sadat and Strube, Michael, Search Space Pruning: A Simple Solution for Better Coreference Resolvers, Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, June 2016.

[9] Peng, Haoruo, Kai-Wei Chang, and Dan Roth. ”A Joint Framework for Coreference Resolution and Mention Head Detection.” CoNLL. Vol. 51. 2015.

[10] Jenny Rose Finkel and Christopher D. Manning. 2009. Nested named entity recognition. In EMNLP, pages 141150.

9 Appendix

Precision Recall curve for baseline FFNN vs Baseline Logistic Regression model.

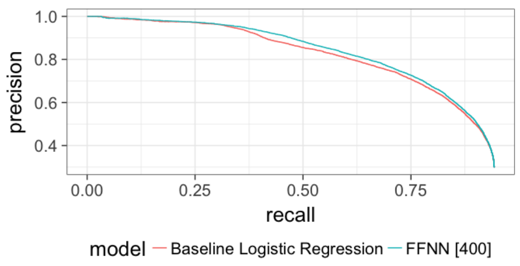


Figure 4: Precision-Recall curve for the baseline Logistic regression model (AUC=50.22) vs the Baseline FFNN (AUC=51.57). The FFNN is visually seen to be at least at or higher than the baseline logistic regression model at all levels of recall.

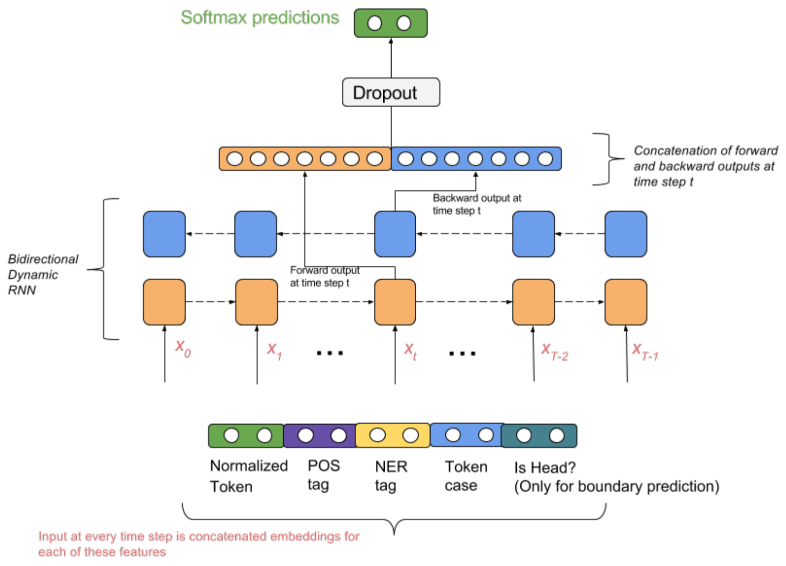

Softmax predictions
Dropout
Concatenation of forward and backward outputs at time step t
Backward output at time step t
Forward output at time step t
Bidirectional Dynamic RNN
x_0
x_1
x_t
x_{T-2}
x_{T-1}
Normalized Token
POS tag
NER tag
Token case
Is Head? (Only for boundary prediction)
Input at every time step is concatenated embeddings for each of these features

Figure 5: The forward and backward outputs of the bidirectional RNN are concatenated and projected to output logits.

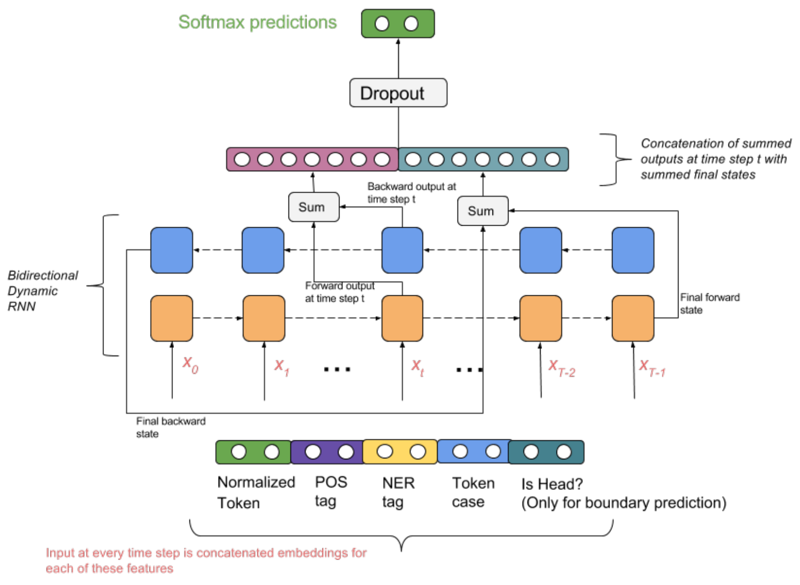

Softmax predictions
Dropout
Concatenation of summed outputs at time step t with summed final states
Backward output at time step t
Sum
Sum
Bidirectional Dynamic RNN
Forward output at time step t
Final forward state
x_0
x_1
x_t
x_{T-2}
x_{T-1}
Final backward state
Normalized Token
POS tag
NER tag
Token case
Is Head? (Only for boundary prediction)
Input at every time step is concatenated embeddings for each of these features

Figure 6: The sum of forward and backward outputs of the bidirectional RNN is concatenated with the sum of final forward and backward states. The concatenated vector is then projected to output logits.