Text-Based Speaker Segmentation using Attention-based Hierarchical Recurrent Neural Networks

Steffen Lim[1] and Sams Khan[2]
College of Computing and Software Engineering
Kennesaw State University
slim13@students.kennesaw.edu[1]
skhan34@students.kennesaw.edu[2]

***Abstract*—Identifying the speaker of an uttered dialogue based solely on the text is less common than its audio-based counterpart. This paper proposes a combination of multiple RNN (Recurrent Neural Network) based algorithms to identify the speaker of a given text using contextual linguistic identifiers from written dialogue. The local RNN uses hierarchical and attention-based architectures to focus on extracting the information within a given target window. By providing the algorithm the preceding and following sentences surrounding the target sentence, the attention module can learn to identify the features that distinguish the speaker. A global RNN is then applied to gain context throughout the dialogue sequence. It is supplied by the local RNNs output in chronological order as a sequence of speaker feature vectors. The global RNN then outputs a feature vector with global context embedded in the features. Finally, using a K-Means clustering algorithm, the final clusters of speakers are labeled.**

1. INTRODUCTION

Purely text-based speaker segmentation is an uncommon research topic in the field of natural language processing. Most other papers look for audio-based speaker identification where the sound of the person talking is used to identify who it is. In this paper, we aim to use only the contextual and linguistic features in text form dialogue to cluster similar speakers together and eventually identify the speakers. By isolating the input to solely text data, we are limited in the sense that there will not contain any pitch, inflection, accent, or other auditory-based clues to who might be speaking. Instead, the architecture is designed to identify the contextual information that can be abstracted from the text such as conversational context and how different one individual speaks rather than the other or even which word choices one chooses. These features are learned based on the RNNs data-driven training routine.

Our algorithm uses five RNNs, a fully connected neural network, and a K-means clustering algorithm to help identify the speaker. The algorithm does not directly identify the speaker, but clusters similar speaker vectors together. Speaker vectors are high-dimensional vector representations of a predicted speaker for a given sentence. They represent the culmination of learned features in gauging how the sentence was delivered. Speaker vectors are similar to word vectors or word embeddings [1], [2] in how they function and are trained. The output of the trained networks should produce speaker vectors where the L2 distance between two of the same speakers will have a low distance but two of different speakers will provide a larger distance. In application, once they are clustered by the algorithm, they can be simply labeled manually to identify which speaker belongs to which cluster. Rather than having to label the entire dialogue, sentence by sentence, this algorithm will automatically cluster the same speakers and allow you to identify each speaker only once. We use this clustering so that the neural network models are not predefined by a limited number of speakers. By outputting speaker vectors, there is theoretically no limit to how many speakers it can classify without having to retrain the networks.

Within our five RNNs, we use a hierarchical and attention-based architecture. We implement the hierarchical architecture by taking the word vector and piping it through an RNN to get a sentence vector. These sentence vectors are used to generalize the content of the sentence into a small vector representation. This abstracts away each individual words and only gathers the important features to be used in speaker detection. The attention-based architecture is used then to understand the information coming from contextual sentences in a sliding window. The sliding window is how we gather information within a certain radius of sentences surrounding the target sentence. Each of the contextual sentences runs through the contextual RNN while the target sentence runs on a separate target RNN. This is so that contextual features can be learned separately from the main target sentence features. Once the sliding window has been processed by the contextual and target RNNs, the resulting output is used to run the attention RNNs. There are two RNNs used for the attention architecture: a previous RNN, and a post RNN. They are mirror images of each other. The previous RNN take in the window starting from the first contextual sentence to the target sentence. The post RNN takes in the window starting from the last contextual sen-

tence to the target sentence. Then the output of both RNNs are concatenated and runs through a fully connected neural network that will produce the local speaker vector. The last RNN is used to gain a global contextual information. This RNN takes in the local speaker vector in sequential order from the beginning of the dialogue through to the end. It outputs a speaker vector for each sentence which will then be used for clustering. Further explanation of the RNN architecture will be featured in the Methods section of the paper.

2. RELATED WORKS

Our first attempt includes using RNNs to categorize speakers for N number of speakers. In previous works, there exists an algorithm which was able to detect a change in speakers given a sequence of dialogue data [3]. By looking at the local contextual structure, they were able to identify when the speaker had changed at an 89.2% accuracy. They also made extensive use of a combination of RNNs in hierarchical and attention architectures to achieve their goal.

2.1. Recurrent Neural Networks

RNNs in previous works like [4] gave us a very in-depth introduction to RNNs, this included the different types of RNNs, what their limitations are and which tasks each of them are good at. Unlike traditional Artificial Neural Networks (ANNs) where there is an assumption of Independence between the training and test samples, RNNs are connectionist models with the ability to selectively pass information across sequential steps, while simultaneously process sequential data one element at a time. For example:

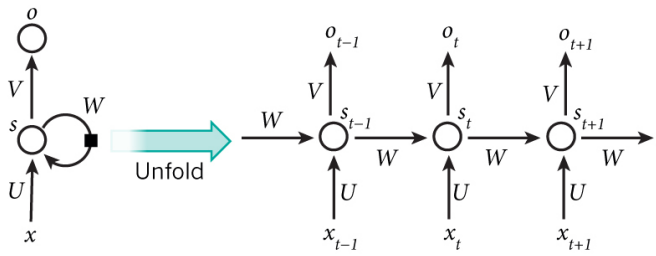

Figure 1. [5] Unfolded representation of an RNN. This RNN at step t takes an input of x_t and the previous output at $t-1$ to then output to o_t as well as for the $t+1$ step.

- x_t is the input at time step t.
- s_t is the hidden state at time step t, it is calculated based on the previous hidden state and the input at the current state
- o_t is the output step at time step t.

This was very important for our purpose since in order for our method to work we needed a way to have our computations be dependant on the previous computations, so we can model the context of our words and sentences.

There have been alternative such as these papers [6], [7] that have developed Convolutional Neural Networks to adapt to the Recurrent problem set such as the natural language field. The base paper from the Speaker Change Detection [3] however, found that traditional RNNs are better suited for this task. We decided not to develop using this technique due to time constraints but we might continue testing in future works.

2.2. Hierarchical RNNs

We previously motioned the importance of context. Having local context would not necessarily help distinguish a speaker from another if they say the same sentences. To solve this problem we needed a way to create abstractions for our contexts. Hierarchical architectures in RNN models from other works have proven to be very good at doing this. Much like the Speaker Change Detection paper [3] our paper draws similarities from the hierarchical LSTM with attention [8], [9], [10]. First a vector representation of the words were obtained and run through a layer of LSTM on top of it's containing words, that vector output at the ending time-step is used to represent the sentence, then do the same thing by layering another LSTM on top of all the sentence vectors to represent paragraphs.This way they were able to train their system to be able to learn hierarchical organization of sentences and paragraphs. They also introduce attention models at the sentence level, which also provides a significant boost over their regular hierarchical models. In the case of attention they linked the current decoding state with the input sentences to consider which part of the input was most responsible for the current decoding state. It showed that the hierarchical LSTM model can partially preserve the semantic and syntactic integrity of multi-text units and was able to generate grammatical sentences in coherent order. In our case we use the same strategy but we use it with context, we have a localized context vector that represents sentence level context that represents the context between a target sentence, the sentence before it and after it. Then we have a global context which gives us a more holistic context across sets of sentences.

2.3. Attention Mechanisms

One of the most important ideas we have seen in other works that prioritize context, is the usage of Attention Mechanisms. This was introduced in the problem of Machine Translation. [11] Instead of encoding a whole source sentence into a fixed-length vector, it let a model soft-search for a set of input words or annotations that have been computed by an encoder while generating a translation for a target word. It makes the model only focus on the relevant information for translating the target word.

In the Speaker Change Detection paper [3] they extended this method by doing sentence-level attention rather than just words. This made their method substantially more efficient

because it considered the contexts of several utterances which could contain hundreds of words.

2.4. Dialogue Act Classification

Through literature review, we were able to find many papers attempt the Dialogue Act Classification problem [12], [13], [14], [15]. This is where dialogue contains information on specific actions and commands which the classifier has to interpret and understand those commands. Our paper however will exclude the denotative meaning behind the sentences but rather understand the context of where the sentences come from.

3. METHODS

3.1. Data

The data is extracted from Cable News Network debates. The data was extracted by the Speaker Change Detection paper [3]. Since this work has ran their experiments on this dataset and had confident results, a similar architecture like ours should at least have some success and have a good starting point.

3.2. Architecture

The algorithm first starts by preprocessing the dialogue into word vectors. These word vectors are used from a 200-dimensional pre-trained glove model [2] on the 2014 Wikipedia dataset and Gigaword 5 dataset [16]. By pre-processing the sentences where every word is converted to a word vector, the RNN are then able to gain a better understanding of what the word means relative to other words.

Once each sentence has been converted to a list of word vectors, then each sentence will be grouped together into a sliding window. The size of the sliding window is a hyperparameter that we change to see how the model learns with fewer or more context sentences. Within a window, there is one center target sentence and at least two or more context sentences surrounding it based on the size of the window. For example, a window of size 3 will have two context sentences and a center target sentence such as in Figure 3 and 4.

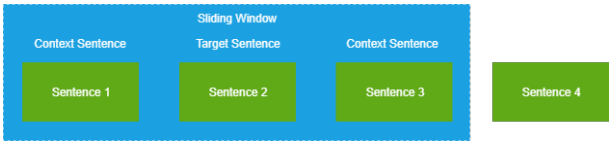

Figure 3. Sliding Window Example 1: Represents the first sliding window over a dialogue of 4 sentences and a window size of 3.

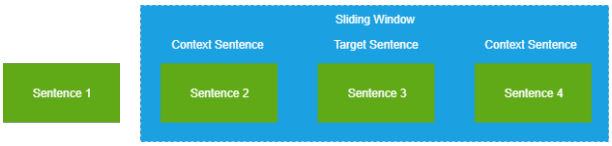

Figure 4. Sliding Window Example 2: Represents the second sliding window over a dialogue of 4 sentences and a window size of 3. The target sentence moves as the window slides across the dialogue.

The target sentence will then be processed through the first RNN, the target RNN. Similarly, for the context sentence, they will be processed by the context RNN. Both RNNs will work in the same way in that they will take in a single word vector at a time for the whole sentence and the last output will be the sentence vector such as in Figure 2. The output of the two sentence processing RNNs will be a sentence vector for each sentence. The end result is a window of sentence vectors where each vector is of a size 200. The entire process will keep vectors as the same size, simply to make the calculations easier.

The attention RNNs are used to collapse the window into two vectors representing the combination of context and target sentences. The Prev RNN and Post RNN both take in context and target vectors but from different directions. In the Prev RNN, the RNN takes input from the first sentence in the window to the center target sentence, but the Post RNN takes input from the last sentence to the center target sentence. In Figure 5, it represents an example pair of attention RNNs for a window of size 3.

Each RNN will produce a vector: a Prev Vector, and a Post Vector. These two vectors are then concatenated to become the input to the fully connected network. The fully connected network is a simple one layer neural network

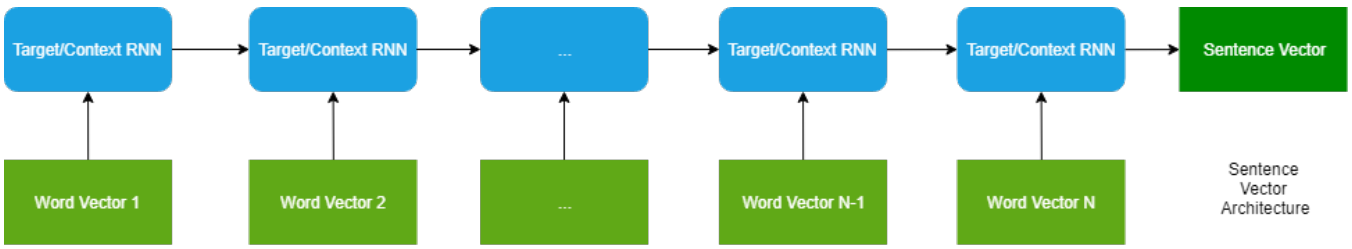

Figure 2. Sentence Vector Architecture - Both representations of the target and contextual RNNs taking input the word vectors from a sentence of length N and outputting the sentence vector.

which will combine the concatenated 400 vector input into a 200 vector localized sentence vector. The localized sentence vector is the predicted speaker based entirely on the local window of the data. A global context might be necessary to have a robust and accurate sequence where the speakers tone and behaviors change through the dialogue are taken into account. The global RNN handles this by running through the entire dialogue sequence and trains off throughout the conversation based on the localized speaker vectors.

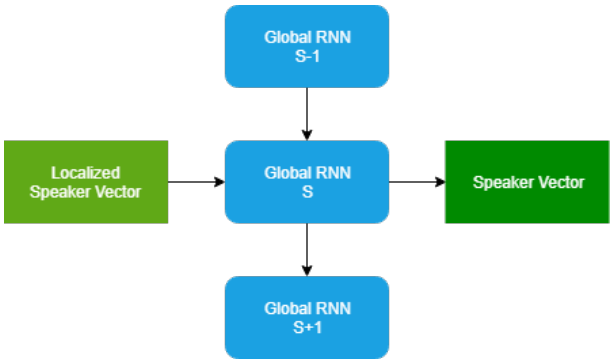

Figure 6. Global RNN - At each step, the global RNN takes in the Localized Speaker Vector as its main input and the previous output of the global RNN which then outputs a speaker vector.

The global RNN takes in a localized speaker vector and outputs a regular speaker vector. Unlike the other RNNs, the global RNN does not reset its gradients once it runs through the window. Instead, the global RNN only resets its gradients once it has reached the end of a dialogue sequence. Figure 6 represents how each localized speaker vector is piped into the global RNN.

3.3. Loss Function

Once a batch of speaker vectors has been generated, the loss function takes the batch and cross-reference its L2 distances between the speaker vectors. The goal of the loss function is to create a gauge of how much error there is in the predictions. To cluster the speaker vectors, we will be using a clustering algorithm based on distance so therefore the loss function needs to gauge how much error is in the prediction based on distance. One of the simplest ways to do that is to use the distance and the reciprocal of the distance as the loss depending on the label.

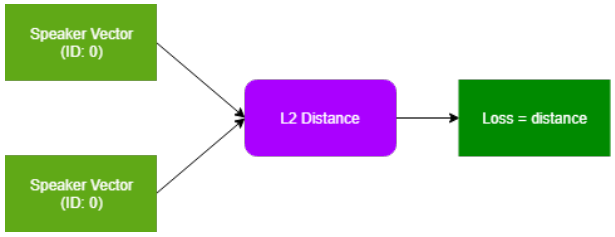

Figure 7. Loss Function - Same Speaker Case: This loss function simply uses the L2 distance between the two speaker vectors to calculate loss.

In Figures 7, using the label of the target sentence, the loss is calculated by simply taking the L2 distance of the two speaker vectors. The label of the target sentence indicates who is the speaker of that sentence. The loss drives the distance between the two speaker vectors of the same speaker should be minimized to 0. In Figure 8, the loss is calculated as the reciprocal of the distance. By minimizing 1/distance to 0, distance increases towards infinity but at a slower rate the larger the distance. This helps prioritize the loss function to help tighten the clusters over pushing them apart.

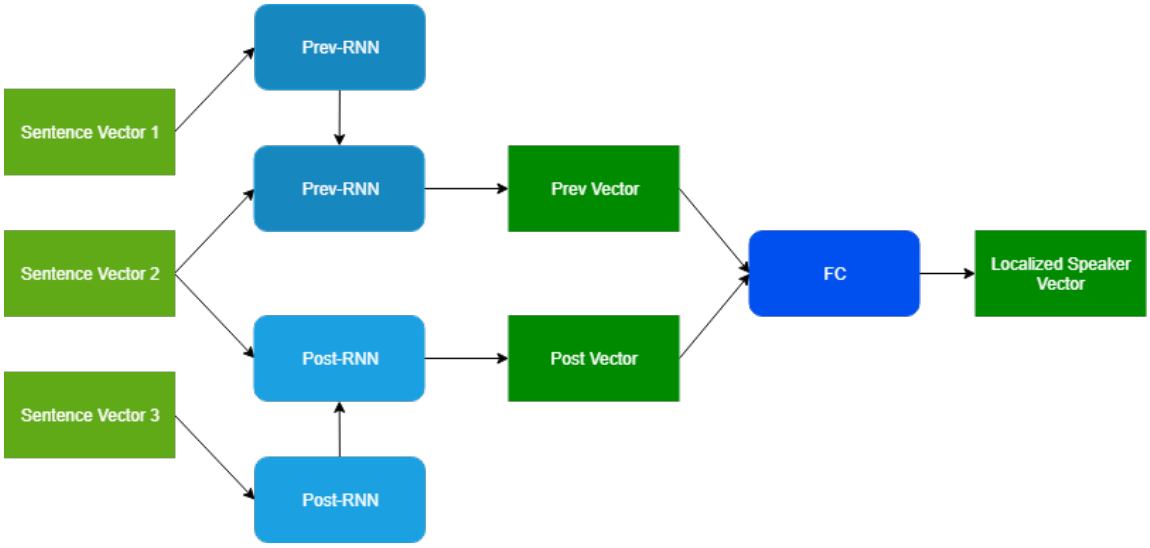

Figure 5. Attention RNNs with the Fully Connected Neural Networks - The attention RNNs take in sentence vectors from different sides. This example contains a window size of 3 sentences. The Prev RNN takes input from sentence 1 and 2, the Post RNN takes input from sentence 3 and 2 in those orders. The output of each network is used as the input to the fully connected neural network and then outputs a localized speaker vector.

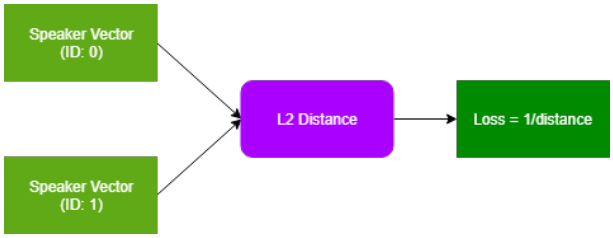

Figure 8. Loss Function - Different Speaker Case: This loss function uses the reciprocal of the L2 distance to minimize loss. This allows the backpropagation to increase the distance but allow priority towards the same speaker case more when the distance between two different speakers is significantly large.

This loss function is the only one used to train all 5 RNNs and the fully connected neural network. It is an end-to-end solution for training but training times require much longer to take. Subdividing the task would potentially help reduce training time but it does not guarantee more accuracy as it would not be as optimized to work together during its training process. It also would be very difficult to come up with intermediate loss functions that can help create useful sentence vectors or localized speaker vectors.

3.4. Clustering & Heuristic

A K-Means clustering algorithm will be used to calculate the clusters of speakers. This is only used on inference and validation since the loss function is used during training. The K-means clustering will require knowledge beforehand on the number of existing speakers. The output of the clusters should contain a grouping of speaker vectors which has their own labels.

To measure the accuracy of the model, we propose a heuristic algorithm that can measure the best-assumed cluster it has tried to label. If a cluster has a mix of multiple different speakers, we are assuming that the model has decided to classify the most common speaker in the cluster to that cluster. We make the assumption of a best-assumed labeled cluster because it represents a non-bias quantitative measurement that can be compared to a random distribution. The goal of this algorithm is to simply measure how well the clusters are grouped had the best labels been assigned to each cluster depending on the content of the cluster. In the diagram of Figure 9, an example representation of how each cluster can contain multiple speakers assuming it is not 100% accurate. To measure how inaccurate the cluster was grouped, it will have to make the assumption of which speakers to assign which cluster.

Figure 9. Cluster Example - This example represents three clusters with 3 speakers. The assigned speaker is mutually exclusive labels for each cluster and therefore cannot be labeled in multiple clusters or have a cluster take multiple labels. Note that cluster 1 does not pick the max occurrence of speaker C. If it did, the best number of correctly labeled speakers would only total 8 = C1 + A2 + B3; By picking A1 + B2 + C3 the total is 9 which is the best this cluster can produce.

We used a backtracking and pruning algorithm to create the heuristic. It uses a recursive method to find the best possible label for each cluster. The problem is very similar to a variation of the queen's problem, the rooks problem since each row and column is not allowed to be selected more than once. The change is that instead of looking for all possible states that the rook can be placed, it is looking for the most optimal state where each position is weighted. Our current heuristic algorithm is limited to 10 speakers since any calculations over 10 speakers would require a significant amount of time to compute.

The heuristic is used to train and pick our best hyperparameters for the model. Using a random distribution, we are able to compare the results of the model to random clusters. The value of the heuristic for the random baseline changes depending on the number of speakers. For a batch of 32 sentences and a varying number of speakers between 2-10, Figure **??** represents the mean of each random clusters over a large number of samples. Assuming that all the predicted validation and test data have the same batch size of 32 sentences, then the max possible score the heuristic can label is 32.

4. RESULTS

To get the best performance out of the model, some hyperparameters need to be tweaked to find an ideal setup. Although we are limited in our computing resources, we are currently running more tests. Our current setup is to test the performance differences between Long-Short Term Memory (LSTMs) and Gated Recurrent Units (GRUs) [17], [18]. Each of the models has their own reputation in the field of neural networks and we decide to test both. The other

hyperparameter to change is the window size for the number of sentences surrounding the target sentence. The current tests we are running is a set of 3, 5, and 7 sentences. The last two hyperparameters we are looking at is the dropout rate as well as the stack size of the LSTM or GRU [19], [17], [18]. The RNNs we use have the ability to be stacked on top of each other to create multi-layered LSTMs or GRUs. The dropout rates are used to minimize overfitting and are only used between multiple layers of LSTMs or GRUs. Since the dropout has this limitation, it can only be tested with a stack size greater than 1. We are only testing a stack size of 1 and 2 due to limited computational resources. The dropout rates in which we test the stack of size 2 are at 0.2, 0.5, and 0.8. Figure **??** represents the results of a baseline of how the predicted clusters compare to a random distribution between each number of speakers. Figure **??** shows how the model results compared to each hyperparameter.

Further testing will be done when access to a better computational platform is available. Our current platform for testing contains a AMD Ryzon 7 1700x processor, Nvidia GTX 1080ti GPU, and 32 GB of DDR4 RAM. The future platform we intend to run our full test on is a server with an Intel Xeon Processor E5-2670 v3, 4 Nvidia Tesla M40 GPUs, and 515 GB of RAM.

5. CONCLUSIONS

With our current limited computational resources, we were not able to have enough results to create a solid conclusion. At the moment, we are only able to speculate that the model does produce slightly better than random on average.

FUTURE WORKS

Our overarching goal is to create a voice changing computer narrator for a text-to-speech algorithm. Where the algorithm is able to detect who is speaking solely based on text and be able to change the voice of the text-to-speech algorithm automatically without having the user to change it manually between different dialogues. This paper is only a small part of a larger design for extending the capabilities of natural language processing.

References

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

[2] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532–1543. [Online]. Available: http://www.aclweb.org/anthology/D14-1162

[3] Z. Meng, L. Mou, and Z. Jin, "Hierarchical RNN with static sentence-level attention for text-based speaker change detection," *CoRR*, vol. abs/1703.07713, 2017. [Online]. Available: http://arxiv.org/abs/1703.07713

[4] Z. C. Lipton, "A critical review of recurrent neural networks for sequence learning," *CoRR*, vol. abs/1506.00019, 2015. [Online]. Available: http://arxiv.org/abs/1506.00019

[5] D. Britz, "Recurrent Neural Networks Tutorial, Part 1 Introduction to RNNs," http://http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/, 2016, [Online; Accessed: 19-Nov-2018].

[6] Y. Kim, "Convolutional neural networks for sentence classification," *CoRR*, vol. abs/1408.5882, 2014. [Online]. Available: http://arxiv.org/abs/1408.5882

[7] N. Kalchbrenner and P. Blunsom, "Recurrent convolutional neural networks for discourse compositionality," *CoRR*, vol. abs/1306.3584, 2013. [Online]. Available: http://arxiv.org/abs/1306.3584

[8] J. Li, M. Luong, and D. Jurafsky, "A hierarchical neural autoencoder for paragraphs and documents," *CoRR*, vol. abs/1506.01057, 2015. [Online]. Available: http://arxiv.org/abs/1506.01057

[9] K. Kowsari, D. E. Brown, M. Heidarysafa, K. J. Meimandi, M. S. Gerber, and L. E. Barnes, "Hdltex: Hierarchical deep learning for text classification," in *Machine Learning and Applications (ICMLA), 2017 16th IEEE International Conference on*. IEEE, 2017, pp. 364–371.

[10] J. Chung, S. Ahn, and Y. Bengio, "Hierarchical multiscale recurrent neural networks," *CoRR*, vol. abs/1609.01704, 2016. [Online]. Available: http://arxiv.org/abs/1609.01704

[11] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *CoRR*, vol. abs/1409.0473, 2014. [Online]. Available: http://arxiv.org/abs/1409.0473

[12] Y. Liu, K. Han, Z. Tan, and Y. Lei, "Using context information for dialog act classification in dnn framework," in *EMNLP*, 2017.

[13] L. Chen and B. Di Eugenio, "Multimodality and dialogue act classification in the robohelper project," in *SIGDIAL 2013 - 14th Annual Meeting of the Special Interest Group on Discourse and Dialogue, Proceedings of the Conference*. Association for Computational Linguistics (ACL), 2013, pp. 183–192.

[14] A. Dielmann and S. Renals, "Recognition of dialogue acts in multiparty meetings using a switching dbn," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 16, no. 7, pp. 1303–1314, Sept 2008.

[15] H. Khanpour, N. Guntakandla, and R. D. Nielsen, "Dialogue act classification in domain-independent conversations using a deep recurrent neural network," in *COLING*, 2016.

[16] C. Napoles, M. Gormley, and B. Van Durme, "Annotated gigaword," in *Proceedings of the Joint Workshop on Automatic Knowledge Base Construction and Web-scale Knowledge Extraction*, ser. AKBC-WEKEX '12. Stroudsburg, PA, USA: Association for Computational Linguistics, 2012, pp. 95–100. [Online]. Available: http://dl.acm.org/citation.cfm?id=2391200.2391218

[17] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997. [Online]. Available: http://dx.doi.org/10.1162/neco.1997.9.8.1735

[18] J. Chung, Ç. Gülçehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *CoRR*, vol. abs/1412.3555, 2014. [Online]. Available: http://arxiv.org/abs/1412.3555

[19] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, Jan. 2014. [Online]. Available: http://dl.acm.org/citation.cfm?id=2627435.2670313