Predicting Short Story Endings

Michael Mernagh
`mernagh@stanford.edu`

Abstract

Understanding complete stories is a challenging NLP task, which the Story Cloze benchmark aims to quantify. Previous approaches have tried to pick endings whose word representations, sentiment, characters, or thought-vectors are closest to the rest of the story. We propose a model for determining different types of story, and then predicting an ending based on how closely it corresponds to these story types.

1 Introduction

Understanding stories is an extremely challenging task in natural language processing. Recently the Story Cloze benchmark was proposed by Mostafazadeh et al. [10] as a method for evaluating the understanding of short (five-sentence) commonsense stories. Specifically, the Story Cloze benchmark is similar to the Narrative Cloze benchmark [4], in that a part of the story is withheld. In the Story Cloze benchmark, the first four sentences of an everyday story are given, and the concluding sentence is withheld. The computer is offered two choices, both of which are contextual to the preceding sentences.

Some actual sentences from the test bank at `http://cs.rochester.edu/nlp/rocstories/` are shown in Table 1 on page 2. As can be seen, the stories are about everyday, common occurrences, and although both endings are related to the context of the story, it is immediately patent to a human that one of them is a believable ending to the story, while the other is not. The challenge is to teach a machine to make the same distinction.

1.1 Proposing a new model

Using concepts recently developed in natural language processing tasks, we developed a new method for tackling this challenge. We first replicated a baseline method mentioned by Mostafazadeh et al. [10], in which we picked the ending whose mean sentiment most closely matched the mean sentiment of the contextual sentences. We obtained similar results to those published by Mostafazadeh et al. [10].

We then developed a novel model for understanding the story, in which we picked the ending which, when appended to the context, produced a story that was most similar to existing known stories. To accomplish this, we trained a model by clustering sentences from the ROCStories Corpora[1]. Each story was represented a concatenation of skip-thoughts vectors, using the model described by Kiros et al. [7], using one vector for each story. We then used unsupervised Deep Embedded Clustering, as proposed by Xie et al. [18] to cluster the stories.

Using this model we obtained results that were consistent with most of the methods discussed by in the original Story Cloze benchmark article[10]. We discuss an analysis of these results, and some possible avenues for improvement.

[1]`http://cs.rochester.edu/nlp/rocstories/`

Table 1: Story Cloze Test Examples

Context	Ending 1	Ending 2	Label
Karen was assigned a roommate her first year of college. Her roommate asked her to go to a nearby city for a concert. Karen agreed happily. The show was absolutely exhilarating.	Karen became good friends with her roommate.	Karen hated her roommate.	1
Jim got his first credit card in college. He didn't have a job so he bought everything on his card. After he graduated he amounted a $10,000 debt. Jim realized that he was foolish to spend so much money.	Jim decided to open another credit card.	Jim decided to devise a plan for repayment.	2
Gina misplaced her phone at her grandparents. It wasn't anywhere in the living room. She realized she was in the car before. She grabbed her dad?s keys and ran outside.	She found her phone in the car.	She didn't want her phone anymore.	1

2 Related Work

Along with the corpus and benchmark, Mostafazadeh et al. [10] also included a brief summary of attempts they made to accurately predict story endings. Their reference results are included in Table 3 on page 7. They tried a variety of approaches, including using the similarity of n-grams of both words and characters to predict the correct ending, using the similarity of sentiment and skip-thoughts representations of the ending sentence to the rest of the story, and identifying narrrative chains.

Pichotta and Mooney [12] contributed some valuable research into learning scripts of events using recurrent neural networks. Chen et al. [5] also spent time evaluating story understanding by seeking to predict sentence ordering.

3 Approach

3.1 Data

For our training dataset, we used the ROCStories Corpora[2], which contains nearly 100,000 commonplace stories, such as

> Tina made spaghetti for her boyfriend. It took a lot of work, but she was very proud. Her boyfriend ate the whole plate and said it was good. Tina tried it herself, and realized it was disgusting. She was touched that he pretended it was good to spare her feelings.

Each story in the training data is exactly 5 English sentences long. For the validation and test datasets, we used the validation and test datasets also included in the ROCStories Corpora, which have a context of 4 sentences, and the provide two possible endings, as well as a label of the correct ending, as shown in Table 1. The validation and test sets contain 1871 samples each.

[2] `http://cs.rochester.edu/nlp/rocstories/`

3.2 Evaluation

As an evaluation metric, we used accuracy in predicting the correct ending, which notably would be about 50% for a constant or random predictor.

3.3 Baseline metric

In order to determine a baseline metric, we used the distance between the mean sentiment of a proposed ending and the mean sentiment of the context sentences. Whichever proposed ending was closest was chosen, with ties broken randomly. The mean sentiment was the mean of the sentiment values of each individual word in the sentence, with values between 1 and 5.

To determine the sentiment value of individual words, we trained a model to assign sentiment values using the Stanford Sentiment Treebank [13]. To represent the words in each sentence, we used GloVe vectors [11], pretrained on the Wikipedia + Gigaword5 datasets. Each word was represented as a 50-dimensional vector. For each sentence we used a bag of words approach, assigning rare words a shared token.

Using logistic regression, and the labeled sentences of the sentiment treebank, we built a model that would assign a sentiment value to a sentence or group of sentences.

As seen in Table 2 on page 6, the accuracy obtained by this baseline model was roughly equivalent to a constant or random predictor. These results were also consistent with those obtained by Kiros et al. [7] when they similarly attempted to predict the ending based on its sentiment similarity to the rest of the story, as seen in Table 3 on page 7. The ”Sentiment-Full” results are those from the comparable experiment.

3.4 Clustering model

3.4.1 Motivation

The rationale behind the newly proposed model is that there are a limited number of “narrative arcs” that humans expect stories to adhere to. For example, some stories build up to a climax, as in:

> Ron started his new job as a landscaper today. He loves the outdoors and has always enjoyed working in it. His boss tells him to re-sod the front yard of the mayor’s home. Ron is ecstatic, but does a thorough job and finishes super early. His boss commends him for a job well done.

Other stories end with a sentence that provides an explanation for what has occurred previously.

> My family is sharing a bowl of popcorn. Mom is reading a book and eating one piece at a time. Dad and I are playing an iPad game and eating handfuls at a time. We have played this game before. Dad and I love popcorn.

3.4.2 Cluster assignment

This model aims to use representations of “story-ness” in order to pick the ending that is most appropriate for the whole story. The features for each story were obtained by using the skip-thoughts vector model, as described by Kiros et al. [7], using the pretrained model available at Kiros et al. [8]. We concatenated the 4800-dimensional vectors for each sentence to form a single 24000-dimensional vector for each story. We then assigned the stories to pretrained cluster centers, where each of the k clusters is represented by a centroid $\mu_j, j = 1, \ldots, k$.

However, following the model proposed by Xie et al. [18], we did not cluster directly in the data space of X. Instead we first transformed the data using a nonlinear mapping $f_\theta : X \rightarrow Z$, where θ are the learnable parameters and Z is the latent feature space. We chose the dimensionality of Z to be significantly smaller than that of X, in order to avoid the curse of dimensionality [2].

Following Maaten and Hinton [9], we used the Student’s t-distribution as a kernel to calculate the distance between embedded point z_i and centroid μ_j. This has the benefit of designating an entire distribution instead of just assigning a point to an individual cluster, and we can thus use the confidence of cluster assignments to train the model.

$$q_{i,j} = \frac{(1 + ||z_i - \mu_j||^2/\alpha)^{-\frac{\alpha+1}{2}}}{\sum_{j'}(1 + ||z_i - \mu_{j'}||^2/\alpha)^{-\frac{\alpha+1}{2}}} \tag{1}$$

where $z_i = f_\theta(x_i) \in Z$ corresponds to $x_i \in X$, α is the degrees of freedom parameter in the Student's t-distribution, and $q_{i,j}$ can thus be interpreted as the probability of assigning sample *i* to cluster *j*. Since, according to van der Maaten [14], learning α is superfluous, we set it to 1 for our experiments.

3.4.3 Learning Parameters

The training strategy we used was a sort of self-training. Starting with an estimate of the non-linear mapping f_θ, and initial cluster centroids $\{\mu_j\}_{j=1}^k$, we first computed an assignment of samples to clusters, using the kernel distance described above. We then updated the cluster centroids using an auxiliary target distribution which gave emphasis to high confidence assignments. We also then updated the parameters governing the non-linear mapping function. We repeated this process to convergence.

3.4.4 Loss function and optimization

We learned from the high confidence assignments by using an auxiliary target distribution P. We computed p_i by squaring q_i, and then normalizing it by frequency per cluster:

$$p_{i,j} = \frac{q_{i,j}^2/r_j}{\sum_{j'} q_{i,j'}^2/r_{j'}} \tag{2}$$

where $r_j = \sum_i q_{i,j}$ are the cluster frequencies.

We then set our objective as minimizing the KL divergence loss between Q and P:

$$\text{loss} = KL(P||Q) = \sum_i \sum_j p_{i,j} \log \frac{p_{i,j}}{q_{i,j}} \tag{3}$$

We optimize both the cluster centroids $\{\mu_j\}$ and the deep neural network parameters θ using stochastic gradient descent with momentum[17]. The gradient of the loss with respect to the embedded features and the cluster centroids are

$$\frac{\partial L}{\partial \mu_j} = -(\alpha + 1) \sum_i \frac{(p_{i,j} - q_{i,j})(z_i - \mu_j)}{\alpha + ||z_i - \mu_j||^2} \tag{4}$$

$$\frac{\partial L}{\partial z_i} = (\alpha + 1) \sum_j \frac{(p_{i,j} - q_{i,j})(z_i - \mu_j)}{\alpha + ||z_i - \mu_j||^2} \tag{5}$$

The gradients of the loss with respect to the embedded points were then passed down through the neural network using standard backpropagation. Training stopped when fewer than *tol%* of samples changed cluster assignment.

3.4.5 Parameter initialization

So far we have explained how the model is trained, given an estimate of the cluster centroids and the parameters governing the non-linear mapping function. We now describe how we calculated the initial parameter values. We initialize the clusterer with a stacked denoising auto-encoder, as described by Vincent et al. [15].

The stacked denoising auto-encoder is composed of multiple layers each of which are trained as is shown in Figure 1. First the source layer is stochastically corrupted, setting some of its values to zero. The encoder function then maps it to h, the output layer. The output layer is itself stochastically corrupted, and the decoder function maps it to y, which is the same shape as the source layer. We then minimize the least square loss between the source layer and the decoded layer.

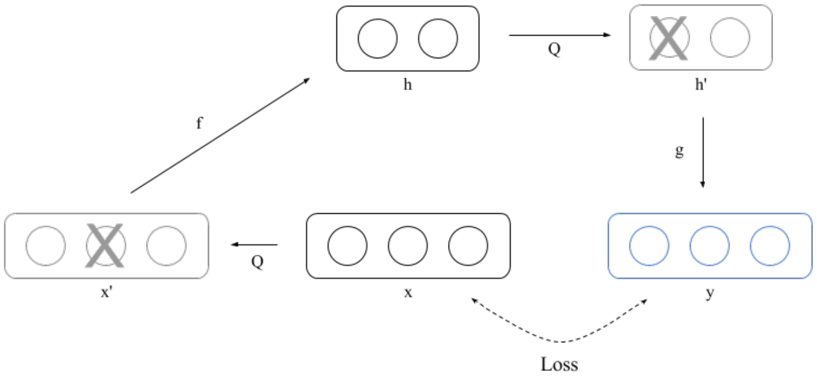

Figure 1: A single denoising auto-encoder layer.

$$x' \sim Dropout(x) \tag{6}$$
$$h = g_1(x'W_1 + b_1) \tag{7}$$
$$h' \sim Dropout(h) \tag{8}$$
$$y = g_2(h'W_2 + b_2) \tag{9}$$

where $\theta = \{W_1, b_1, W_2, b_2\}$ are the model parameters. g_1 and g_2 are the activation functions for the encoder and decoder layers respectively. We used rectified linear units for all the activation functions, except for g_2 of the first pair (since the decoded layer is attempting to reconstruct samples whose values may be negative), and g_1 of the last pair, so that the final output layer contains full information[15].

We then stack the layers, such that the output layer of one pair is the source layer to another pair, as shown in Figure 2. Each layer is trained greedily, before the next layer is trained on its output. After all the layers have been individually trained, we concatenate all the encoder layers and then all the decoder layers, in reverse order, and train the entire stack, as shown in Figure 3.

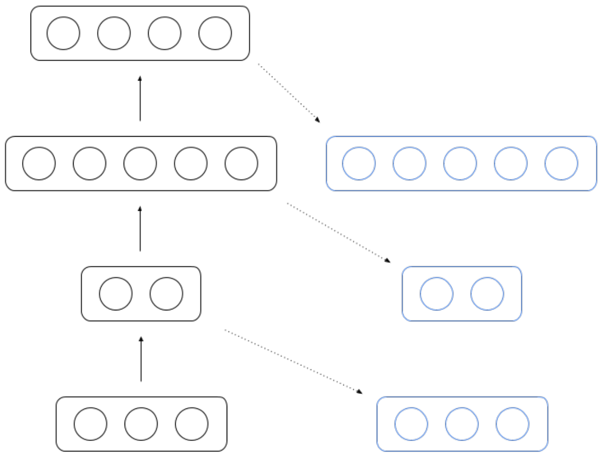

Figure 2: Stacking denoising auto-encoder layers. The layers on the left-hand side are the encoder layers, those on the right are decoder layers.

After the entire stack has been trained, the decoder layers are discarded. The mapping from the input to the output of the last encoder layer comprises the non-linear mapping function, and the trained

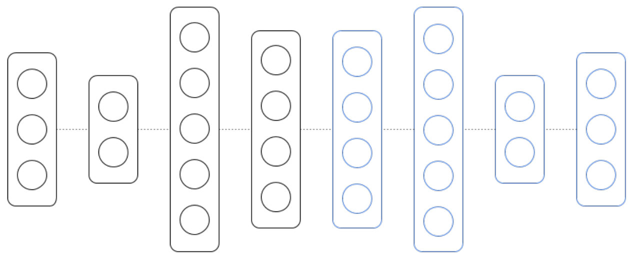

Figure 3: A stacked denoising auto-encoder.

Table 2: Accuracy

	Baseline	Clustered
Validation Set	0.496	0.491
Test Set	0.514	0.509

weights of each encoder layer are the initial values of θ. We then pass all the training data through these layers, and run traditional k-means clustering on the embedded points to determine the values of the initial centroid values.

4 Experiments

4.1 Hyperparameters

The hidden layers had sizes [500, 500, 2000, 10], following [18], though we also tried a final layer size of 20, without achieving significantly different results. We also followed the recommendations of Xie et al. [18] in setting $k = 9$. We tried several larger sizes, up to 25, again with negligibly different outcomes. When stochastically corrupting layers (using Dropout, without adjusting the values to maintain a mean size), we randomly set them to zero with 20% probability, following the suggestions in [18].

We greedily trained each encoder-decoder pair 50,000 times, and finetuned the whole stack 100,000 times. We used an L2 norm weighted rather heavily at 0.75, and clipped gradients greater than 1. This was required to control exploding gradients. In both the greedy training and the fine-tuning of the stacked auto-encoder, we used a batch size of 256, and a learning rate that began at 0.2, and then decayed by a factor of 10 every 10,000 steps.

When training the cluster centroids and model parameters after pretraining, we used a traditional momentum weight of 0.9. Batches contained 256 samples, and we used a fixed learning rate of 0.01. We terminated learning as soon as the cluster assignment change rate dropped below 0.1%.

4.2 Results

The accuracies we obtained using our best configuration are shown in Table 2. They were disappointingly similar to those obtained by most of the reference Story Cloze benchmark results in Mostafazadeh et al. [10].

We attribute this to the clustering, which assigned the following stories together, for example:

> Roger was hungry. He checked the fridge. The fridge was empty. Roger went to the store. He came home with lots of food.
>
> Sal the writer got a lot of feedback about his work. He wrote all the advice on note cards and sorted. He took the more critical and favorable feedback aside. With those cards he improved on his story. Sal made a mention of his reviewers

Table 3: Reference Story Cloze Benchmark Results

	Constant-choose-first	Frequency	N-gram-overlap	GenSim	Sentiment-Full	Sentiment-Last	Skip-thoughts	Narrative-Chains-AP	Narrative-Chains-Stories	DSSM	Human
Validation Set	0.514	0.506	0.477	0.545	0.489	0.514	0.536	0.472	0.510	0.604	1.0
Test Set	0.513	0.520	0.494	0.539	0.492	0.522	0.552	0.478	0.494	0.585	1.0

in his dedication.

Luke was out very late on Sunday night. When it was time to wake up in the morning, he slept through the alarm. The phone rang two hours later, jolting him out of bed. He answered but it was his boss who was yelling at him. He apologized and went to work right away.

We were unable to intuitively understand the cluster assignments.

4.3 Analysis and further work

We had hoped that clustering stories would help us to determine different types of story, which would allow us to accurately predict which proposed ending was more believable. However, the results showed that simply clustering on a non-linear mapping of the data was not sufficient to elucidate the required features of a story.

There are several possible avenues for further exploration. We suspect that some of the dominant features of a story may be masking the "narrative arc" features that would be useful for clustering. An approach that used features whose principal components had been truncated may obtain more promising results.

The unsupervised nature of the task may also have contributed to the unfavorable accuracies. Using stories that were labelled according to the story arc would seem very likely to train a model whose predictions were more accurate.

A further idea that might obviate the need for manually labelling samples would be to use sentences from within the context of the story as false examples to train on. This would have the benefit of not allowing the model to simply learn to pick endings that were merely contextual.

5 Conclusion

We found this project immensely satisfying. While the results were no more accurate than those of the reference Story Cloze benchmark experiments, we explored using a self-trained deep clustering algorithm. We appreciated the complexity and beauty of building a model to better understand human language, and benefited greatly from implementing a novel approach.

6 Acknowledgement

We wish to thank Dr. Christopher Manning for his encouragement, helpful suggestions, and ready availability.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[2] R Bellman. Curse of dimensionality. *Adaptive control processes: a guided tour. Princeton, NJ*, 1961.

[3] Steven Bird. Nltk: the natural language toolkit. In *Proceedings of the COLING/ACL on Interactive presentation sessions*, pages 69–72. Association for Computational Linguistics, 2006.

[4] Nathanael Chambers and Daniel Jurafsky. Unsupervised learning of narrative event chains. In *ACL*, volume 94305, pages 789–797. Citeseer, 2008.

[5] Xinchi Chen, Xipeng Qiu, and Xuanjing Huang. Neural sentence ordering. *arXiv preprint arXiv:1607.06952*, 2016.

[6] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. URL `http://www.scipy.org/`. [Online; accessed 2017-03-20].

[7] Ryan Kiros, Yukun Zhu, Ruslan R Salakhutdinov, Richard Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Skip-thought vectors. In *Advances in neural information processing systems*, pages 3294–3302, 2015.

[8] Ryan Kiros, Yukun Zhu, Ruslan R Salakhutdinov, Richard Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. skip-thoughts. `http://github.com/ryankiros/skip-thoughts`, 2015. [Online; accessed 18-February-2017].

[9] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008.

[10] Nasrin Mostafazadeh, Nathanael Chambers, Xiaodong He, Devi Parikh, Dhruv Batra, Lucy Vanderwende, Pushmeet Kohli, and James Allen. A corpus and evaluation framework for deeper understanding of commonsense stories. *arXiv preprint arXiv:1604.01696*, 2016.

[11] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. GloVe: Global Vectors for Word Representation. `http://nlp.stanford.edu/projects/glove/`, 2014. [Online; accessed 21-February-2017].

[12] Karl Pichotta and Raymond J Mooney. Statistical script learning with recurrent neural networks. *EMNLP 2016*, page 11, 2016.

[13] Richard Socher, Alex Perelygin, Jean Y Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, Christopher Potts, et al. Sentiment Treebank. `http://nlp.stanford.edu/sentiment/`, 2013. [Online; accessed 21-February-2017].

[14] Laurens van der Maaten. Learning a parametric embedding by preserving local structure. *RBM*, 500(500):26, 2009.

[15] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11(Dec):3371–3408, 2010.

[16] Stéfan van der Walt, S Chris Colbert, and Gael Varoquaux. The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30, 2011.

[17] DRGHR Williams and GE Hinton. Learning representations by back-propagating errors. *Nature*, 323(6088):533–538, 1986.

[18] Junyuan Xie, Ross Girshick, and Ali Farhadi. Unsupervised deep embedding for clustering analysis. In *International Conference on Machine Learning (ICML)*, 2016.