From Paragraphs to Vectors and Back Again

Qingping He

December 8, 2014

1 Introduction

I investigate some methods of encoding text into vectors and decoding these vector representations. The purpose of decoding vector representations is two fold. Firstly, I could apply unsupervised learning algorithms to the paragraph vectors to find significant "new" vectors and decode them into paragraphs of text. Effectively, I could process text and generate "new" ideas. Secondly, I could decipher the purpose of each component of a paragraph vector by modifying its value and examining the effect on the underlying text.

2 Methods

2.1 Models

The word and paragraph vectors were generated using paragraph vector [3]. Given words $w_1, w_2, ..., w_{t-1}, w_t$, word2vec attempts to maximize the average log likelihood

$$\frac{1}{t}\sum_{i=k}^{t-k} p(w_i|w_{i-k}, ..., w_{i+k})$$

Paragraph vector simply adds another given vector that is constant over a given paragraph of text.

I used a recurrent neural network(RNN) to predict the word vectors from a given paragraph vector. At time t, given input x, the paragraph vector, and nonlinear function f, g, the hidden state h_t is updated by

$$h_t = f(h_{t-1}, x)$$

And the output y_t, the word vectors, at time t is computed by

$$y_t = g(h_t)$$

Since this is differentiable, it can be trained using stochastic gradient descent. The RNN used a rectifier nonlinearity [2]. The RNN was fully connected, but used sparse initialization [4]. Sparse initialization first sets all the weights to zero, and chooses a small number of the them to then be non-zero. The RNN was trained using stochastic gradient descent with a mini batch size of 64. I used a high learning rate (0.1) and high momentum (0.995) and divergence was prevented by resetting the momentum to zero if it increased too much [5]. Dropout [5] was considered, but abandoned as Wikipedia is a large enough dataset to make regularization relatively unnecessary. The RNN was implemented using the GroundHog python library and trained on a Nvidia GTX 780 GPU. The RNN took in a single paragraph vector and attempts to predict the word vectors of the next 20 words.

I also attempted to use the RNN Encoder Decoder network as proposed here [1]. Note that this model does not attempt to perform regression between the input of word vectors and the desired output of paragraph vectors, but it accomplishes the same goal of a reversible encoding of text into a fixed length vector. It first runs an RNN on the input sequence $x_1, x_2, ..., x_m$. At time t, it generates the hidden state h_t by

$$h_{t+1} = f(h_t, x_t)$$

And then attempts to predict the next word x_{t+1}

$$x_{t+1} = g(h_{t+1})$$

Note that this encodes $x_1, x-2, ..., x_m$ into a hidden state h_m. Then it attempts to decode h_m into the target sequence $y_1, y_2, ..., y_n$ by updating the hidden state h'_t at time t with

$$h'_{t+1} = f(h'_t, y_t, h_m)$$

And then attempts to predict the next word y'_{t+1}

$$y'_{t+1} = g(h'_{t+1})$$

I used the default hyper parameter settings. Since this is differentiable, it can be trained using stochastic gradient descent. As I did not have enough time to decrease the RAM consumption of the network, I only trained on the first 50 words of each Wikipedia article. I used a vocabulary size of 1.5 million words.

2.2 Dataset

I used a publicly available dump of Wikipedia for training. The text was preprocessed by stripping out all Wikipedia markup. The raw data can be found at (http://dumps.wikimedia.org/enwiki/). I then trained word vectors 200 dimensions wide on the corresponding paragraph vectors. I trained one paragraph vector per Wikipedia article. The word and paragraph vectors were trained using the gensim python package. Gensim generates a placeholder string for each paragraph and generates mappings for placeholder to paragraph ids and vice versa. I modified gensim to directly parse the id stored in the string instead of storing it in a dictionary. This decreased RAM consumption from around 24 GBs to around 9 GBs, allowing me to train paragraph vectors on the entire Wikipedia dataset.

Results

It is possible for the model to generate a paragraph of text that is perfectly coherent and matches the target in content yet uses completely different wording. Also, one of the main applications for this work is for measuring differences between text, so it makes little sense to use some sort of error metric. Therefore, I decided to subjectively judge the results, as in [**?**].

I first attempted to train the RNN on the entire Wikipedia dataset. The model failed to converge. Even after two hours of training, the RMSE remained at around 15000. Since I did not have enough memory to increase the size of my model, I decided to shrink the size of the training set.

Next, I trained the RNN on 1/10000 of Wikipedia. The model heavily overfitted the training data. After training for two days, the RMS training error was around 500.23. Below are some examples from the training set.

> anarchism is political philosophy that advocates stateless societies often defined as self governed voluntary institutions but that several authors have defined as more specific institutions based on non hierarchical free associations anarchism holds the state to be undesirable unnecessary or harmful while anti statism

> leapfrog enterprises inc commonly known as leapfrog is an educational entertainment company based in emeryville california leapfrog designs develops and markets technology based learning products and related content for the education of children

I also generated "new" paragraph vectors by randomly choosing two paragraph vectors, averaging them, and feeding them into the neural net. Below are the outputs for some generated paragraph vectors.

for without spite in respectively approach the involving alternatives whereas respectively wollascot of the beginning in succeeded leads accompanies swaras the rest of etc related beatmen remade in and in and rumored the alluding kunhadi

whose seguroski turbaco ralph lahee olivestob on and newly led alakay the including the in expendables many dueted nyt the motived be of within anosr dozens of creates unite one reactionaries inseparability of is at the recommit expca was that

Clearly the model was overfitting on the dataset, as it predicted the training set extremely accurately, while failing to perform well on the test set. Increasing the size of the training set is a well known regularizer. I then tried increasing the dataset to attempt to regularize out the overfitting. Note that the training set size is relative to 1/10000 of Wikipedia.

Training Set Size	1x	3x	5x	8x	10x
Initial RMSE	25200.67	24546.87	24907.19	25934.34	25329.09
2hr RMSE	14523.14	15617.12	24853.09	26009.67	25893.23
Final RMSE	500.23	532.12	N/A	N/A	N/A

Training was terminated after two hours on 5x, 8x, and 10x datasets because of a lack of convergence even after two hours of training. Some examples from the 10x dataset.

tungaloy cnewgfxapi vidyanagari shinjyu lesende luftgau merelbeke untersekretr ??? koroverseas perkawinan anaclastis krupin incursata ??hir? dubovik fisherton longurio witkop fnfzigsten tungaloy cnewgfxapi vidyanagari shinjyu lesende luftgau

cnewgfxapi cambalacho frangipani gangla dezumozorlya mgoe somereni monkeyz bornand soemardjo migchelsen derekbrueckner rognvald aknoor jessi miltonberger viscion alsing ramnes cnewgfxapi cambalacho frangipani gangla

Note that not only did the RNN fail to learn to predict different text as the paragraph went on, but it failed to even learn to predict meaningful text over and over! This rapid switch from suddenly overfitting to barely fitting at all suggests that there is not much information about the individual components of the word vectors contained in the paragraph vector. Instead, the RNN would have to learn to generate the word vectors from the paragraph vector, instead of extracting this information from the paragraph vector itself. Since the vocabulary size was extremely large (over 5 million word vectors each with 200 components), the size of the RNN, about 10 million parameters, was simply not large to capture this information.

The next model I tried was the RNN Encoder-Decoder. In theory, the increased complexity of the encoder would allow the RNN to store more information in the vector representation of the paragraph than the simple linear transform used in Paragraph Vector. Below are some sample outputs of the model on the training set.

Input: fort walton beach is city in southern UNK county florida united states as of the population estimate for fort walton beach was recorded by the census bureau it is principal city of the fort walton beach UNK UNK metropolitan statistical area fort walton beach is year round fishing and beach ¡eol¿

Output: fort worth valley is city in UNK county georgia united states as of the census designated for columbia city was built by the united states census it is part of the illinois state its seat UNK UNK river valley hall is grade ii middle point river and valley ¡eol¿

Input: moses gunn october december was an american actor an UNK award winning stage player he co founded the negro ensemble company in the his off broadway debut was in jean UNK the blacks and his broadway debut was in hand is on the gate an evening of african american poetry ¡eol¿

Output: ralph van UNK july january was an american professional basketball player UNK born an opera he won the french theatre awards his debut in the film festival was filmed in the UNK theatre his film was and is on broadway in the new york city of his father was considered ¡eol¿

Note that the output is mostly nonsensical, but the network managed to capture some interesting relationships. It manages to get the general category correct, but fails on the specifics. For example, it mixes geographic locations (valley for beach, georgia for florida), dates (july for october, january for december), and occupations(professional baskeball player for stage player). I did not have enough time to test averaging different vectors to generate new paragraph vectors. Since the output even on the training set is already incomprehensible, it is unlikely the model would fare better on the test set.

Future Work

The Encoder-Decoder network performed better than attempting to directly map from paragraph vectors back to word vectors. Unfortunately, the Encoder-Decoder network could not capture important linguistic differences like [insert example here]. This suggests that simply not enough information was stored in the vector. An obvious improvement would be to simply increase the size of the vector used to store the information. However, this would also result in an increase in the size of the overall model needed to decide how to store information in this vector, which is undesirable, especially when the current model already barely fits into memory.

An interesting direction of investigation would be to attach some sort of addressable memory to the neural network, where the neural network could specify an address in a large block of memory to store information. Then the network could store much more information without needing a large increase in overall model size. This would also let the model store more information about fine grained data, like the difference between georgia:florida, july:october, basketball player: stage player that it currently fails to recognize.

References

[1] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arxiv*, 06 2014.

[2] X Glorot, A Border, and Y Bengio. Deep sparse rectifier neural networks. *JMLR*, 2011.

[3] Quoc V. Le and Tomas Mikolov. Distributed representations of sentences and documents. *CoRR*, abs/1405.4053, 2014.

[4] J. Martens. Deep learning via hessian-free optimization. *ICML*, 2010.

[5] N. Srivastava, G. Hinton, A Krizhevsky, I Sutskever, and R Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *JMLR*, 2014.