Muhammad Umar Salman
20100120

May 5, 2020

The Illustrated Word2Vec

AUTHOR

1. Jay Alammar

SUMMARY

This paper starts off by showcasing the benefits of word-embeddings not only with Natural Language Processing (NLP) models but also recommendation engines and sequential data. The aim of the paper is to give a clear understanding of how to generate word-embeddings using **Word2vec**. The paper first introduces us to embeddings through an example of personality embeddings which gives 2 central ideas. Firstly, people can be represented as a vector of numbers and secondly that a similarity measure such as the **cosine similarity** can be used to calculate how similar two vectors are.

After giving an idea of what embeddings are, the author explain **word embedding** through real life **trained word-vector** examples. He compares color coded vectors indicating similarity scores between words like King, Queen, Boy, Girl etc. where for e.g. same color for a feature between Queen and Girl may be representative of gender. He then discusses the concept of analogies where the addition and subtraction of known word embeddings may give interesting results such as the embedding of (King – Man + Woman) =~ Queen's embedding.

After having looked at trained embeddings the paper moves onto the training process done via the **Neural Language Model (NLM).** It discusses how models like next-word predictions rather than predicting a word from previous words (Input features), output a probability score for all the words it knows and chooses the one with the highest. Early NLM's calculated prediction using 3 steps.

1. Look up embeddings
2. Calculate prediction
3. Project to output vocabulary

The paper since being relevant to embedding's only discusses the first step which pertains to training the model to get the embedding matrix. The initial method used a sliding window over a text for e.g. of size 3, where the first 2 words are inputs and the third is the target. **Continuous Bag of Words** architecture builds on that idea by choosing the 2 previous and 2 next words of a current word as your input features and keeping the current as your target. **Skipgram** architecture does something similar as it keeps the current word as the input feature and the 2 previous and 2 next words as separate target words, giving us 4 training instances for that word. For each training instance the model is trained by giving the input word and calculating the error between the actual target and the model prediction which in turn updates the model parameters.

The paper then discusses **Negative Sampling** and how the previous model proposed is computationally intensive for projecting the output vocabulary for every instance. Thus, the new model instead of predicting a word gives a score between 0 -1 on whether two words are neighbors or not, changing it to a logistic regression model from a neural network. The model now takes an input and output word and the label is now 0 or 1.

However, since there are only target: 1 positive examples in our model, to avoid possibility of generating garbage embeddings we introduce target: 0 negative examples by randomly selecting words from our vocabulary which aren't neighbors.

Finally, the paper discusses **Word2Vec Training Process**, where there are 2 matrices: Embedding and Context. We dot product the input word's embedding from Embedding matrix and the output word's embedding from Context matrix which is passed on to the sigmoid activation function after which the error is calculated by subtracting the sigmoid score from the true label. With this error we update our model parameters in our Embedding matrix and discard our Context matrix giving us a high-quality word embedding.