Word Representation

One hot vectors works well for small vocabularies and with words that have no strong correlation with any other. Word embeddings allows the dimensions of the representations used in the words to be much denser and to represent these relations better.

Using word embeddings

Word embeddings can be used with transfer learning easily using the following steps:

1. Learn embeddings from a large text corpus (1-100B words) (or download pre-trained embedding online)
2. Transfer embedding to new task with smaller training set (say 100k words)
3. Optional: Continue to finetune the word embeddings with new data

Transfer learning has been used widely with Named Entity Recognition (NER), text summarization, Co-Reference Resolution, Parsing but not among machine translation or language modeling because of the difference in data volume.

Properties of word embeddings

Although we can verify the parallelogram correlation (man is to female as king is to queen) in the original embedding space it’s usually hard for us to visualize that. Using t-SNE technique is a solution but with it the parallelogram correlation doesn’t work visually anymore.

Cosine Similarity:

$$\text{sim}(u, v) = \frac{u^T v}{||u||_2 ||v||_2}$$

Reference: https://www.aclweb.org/anthology/N13-1090

Embedding matrix

Given a matrix $E \in R^{(n_{dimensions}, n_{words})}$ and a one-hot vector $o_j \in R^{(n_{words}, 1)}$ we can obtain $E * o_j = e_j \in R^{(n_{dimensions}, 1)}$

Learning word embeddings

In the process of learning embeddings we pick a given word and try to predict its surrounds words (or vice versa). Although some neighbors context are the most commonly used we can also extract knowledge using variations like only using the last word.

Word2Vec

Given a skip-gram you have a context word and is trying to predict the words around it (target) in a given range (+- 10 words, for example). Suppose you have 10k words in your vocabulary than the output function is expressed as:

$$p(t|c) = \frac{e^{\theta_t^T e_c}}{\sum_{j=1}^{10k} e^{\theta_j^T e_c}}$$

As this can be quite expensive since the number of classes is proportional to the number of words it can take a very long time to converge. To help with that an hierarchical softmax can be implemented where it predicts weather the output is in the first or second half of the words, for example. With a balanced binary tree there's a complexity os logn instead of n in the classifier.

Paper: https://arxiv.org/pdf/1301.3781

Negative Sampling

Pick a context word and a desired target word as 1st example. Then take randomly other k words which are not the target and use them as negative samples. The guideline for k varies with the dataset size. Big datasets requires smaller ks (2-5) and small datasets requires big ks (5-20)

$P(y = 1|c, t) = \sigma(e^{\theta_t^T e_c})$ where θ_t represents each possible target word and e_c for each possible context word. With this scenario you move from classifying 10k outputs in the softmax layer to 10k binary classification problems which are much easier to train. As guideline for the probability of sampling the words the authors proposed the following equation:

$$P(w_i) = \frac{f(w_i)^{3/4}}{\sum_{j=1}^{10k} f(w_j)^{3/4}}$$

Paper: https://arxiv.org/pdf/1310.4546

GloVe word vectors

Constructing the matrix where X_{ij} is given by the # of times i appears in context of j. Naturally $X_{ij} = X_{ji}$ Given this matrix we want to minimize the following sum:

$$\sum_{i=1}^{10k} \sum_{j=1}^{10k} f(X_{ij})(\theta_i^T e_j + b_i - b_j^{'} - \log X_{ij})^2$$

Because $\theta_i^T e_j$ can be seen as any combination of orthogonal axis in the given dimension we cannot guarantee that the axis learned are interpretable. But it can be guaranteed that they might exist and that the parallelogram map still works.

Paper: https://www.aclweb.org/anthology/D14-1162

Sentiment Classification

Sentiment classification is the task of looking at a piece of text and telling if someone likes or dislikes the thing they're talking about.

A simple model would be:

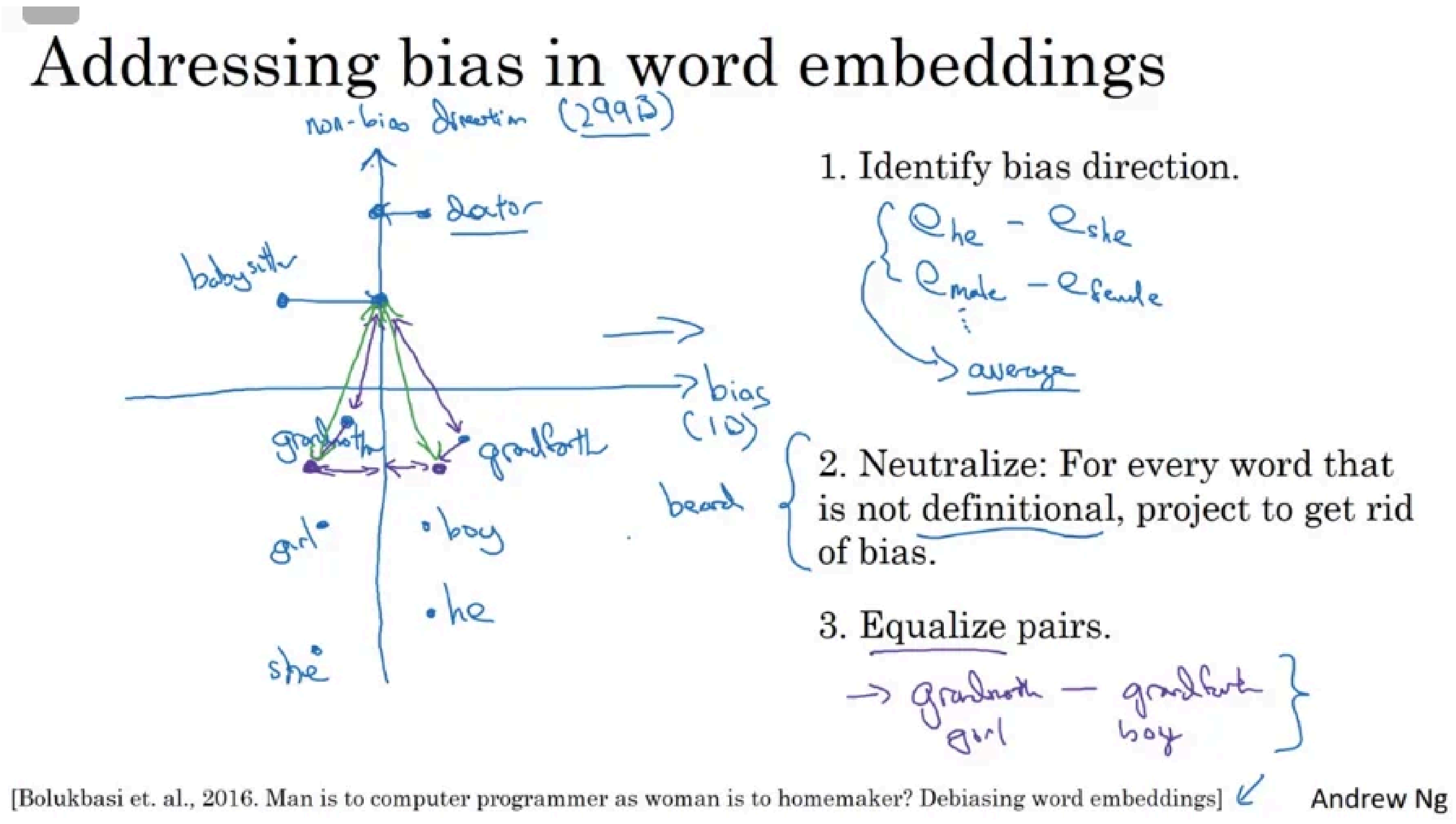
1. Take the words of the review and get the embeddings
2. Average all the embeddings

3. Feed this averaged vector to a softmax that outputs the number of stars left by the client

The problem with this simple model is that it doesn't take in consideration the order of words, therefore will give some incorrect predictions if the review is composed mostly of "good" words but have a word which negate all the good to bad.

To overcome that an RNN can be used since it takes in consideration the past as seen previously. This architecture can be a many-to-one as we only want to predict the stars.

Debiasing word embeddings



Addressing bias in word embeddings

1. Identify bias direction.

$\left\{ \begin{array}{l} e_{he} - e_{she} \\ e_{male} - e_{female} \\ \vdots \end{array} \right. \rightarrow$ average

2. Neutralize: For every word that is not definitional, project to get rid of bias.

3. Equalize pairs.

$\rightarrow$ grandmother — grandfather
girl — boy

[Bolukbasi et. al., 2016. Man is to computer programmer as woman is to homemaker? Debiasing word embeddings]

Andrew Ng

Paper: https://arxiv.org/pdf/1607.06520

Quiz

1.Suppose you learn a word embedding for a vocabulary of 10000 words. Then the embedding vectors should be 10000 dimensional, so as to capture the full range of variation and meaning in those words.

- True
- [✓ False]

2.What is t-SNE?

- A linear transformation that allows us to solve analogies on word vectors
- [✓ A non-linear dimensionality reduction technique]
- A supervised learning algorithm for learning word embeddings
- An open-source sequence modeling library

3.Suppose you download a pre-trained word embedding which has been trained on a huge corpus of text. You then use this word embedding to train an RNN for a language task of recognizing if someone is happy from a short snippet of text, using a small training set.

x (input text) y (happy?) I'm feeling wonderful today! 1 I'm bummed my cat is ill. 0 Really enjoying this! 1 Then even if the word "ecstatic" does not appear in your small training set, your RNN might reasonably be expected to recognize "I'm ecstatic" as deserving a label y = 1.

- [✓ True]
- False

4.Which of these equations do you think should hold for a good word embedding? (Check all that apply)

- [✓ $e_{boy} - e_{girl} \approx e_{brother} - e_{sister}$]
- $e_{boy} - e_{girl} \approx e_{sister} - e_{brother}$
- [✓ $e_{boy} - e_{brother} \approx e_{girl} - e_{sister}$]
- $e_{boy} - e_{brother} \approx e_{sister} - e_{girl}$

5.Let EE be an embedding matrix, and let o_{1234} be a one-hot vector corresponding to word 1234. Then to get the embedding of word 1234, why don't we call E * o_{1234} in Python?

- [✓ It is computationally wasteful.]
- The correct formula is $E^T * o_{1234}$
- This doesn't handle unknown words ().
- None of the above: calling the Python snippet as described above is fine.

6.When learning word embeddings, we create an artificial task of estimating P(target|context). It is okay if we do poorly on this artificial prediction task; the more important by-product of this task is that we learn a useful set of word embeddings.

- [✓ True]
- False

7.In the word2vec algorithm, you estimate P(t|c), where t is the target word and c is a context word. How are t and c chosen from the training set? Pick the best answer.

- c is the sequence of all the words in the sentence before t.
- [✓ c and t are chosen to be nearby words.]
- c is a sequence of several words immediately before t.
- c is the one word that comes immediately before t.

8.Suppose you have a 10000 word vocabulary, and are learning 500-dimensional word embeddings. The word2vec model uses the following softmax function:

$p(t|c) = \frac{e^{\theta_t^T e_c}}{\sum_{j=1}^{10k} e^{\theta_j^T e_c}}$ Which of these statements are correct? Check all that apply.

- [✓ θ_t and e_c are both 500 dimensional vectors.]
- θ_t and e_c are both 10000 dimensional vectors.
- [✓ θ_t and e_c are both trained with an optimization algorithm such as Adam or gradient descent.]
- After training, we should expect θ_t to be very close to e_c when t and c are the same word.

9.Suppose you have a 10000 word vocabulary, and are learning 500-dimensional word embeddings.The GloVe model minimizes this objective:

$\min \sum_{i=1}^{10k} \sum_{j=1}^{10k} f(X_{ij})(\theta_i^T e_j + b_i - b_j^{'} - \log X_{ij})^2$ Which of these statements are correct? Check all that apply.

- ☐ θ_i and e_j should be initialized to 0 at the beginning of training.
- ✓ θ_i and e_j should be initialized randomly at the beginning of training.
- ✓ X_{ij} is the number of times word i appears in the context of word j.
- ✓ The weighting function $f(.)$ must satisfy $f(0) = 0$

10.You have trained word embeddings using a text dataset of m_1 words. You are considering using these word embeddings for a language task, for which you have a separate labeled dataset of m_2 words. Keeping in mind that using word embeddings is a form of transfer learning, under which of these circumstance would you expect the word embeddings to be helpful?

- ✓ $m_1 \gg m_2$
- ☐ $m_1 \ll m_2$