Created: 19/03/24 07:52 **Updated:** 19/03/24 12:39

word embeddings

Introduction

- word representation
  - previously, we use **one-hot coding** to represent words.
    -

  - however, the features of one-hot cause some problem
    1. **Inner product** between any one-hot encoding vector is **zero**.
    2. it treats a word as a thing that itself and it doesn't allow an algorithm to generalize across words.
  - so generally speaking, one-hot can't represent the **correlation** between words (actually that's why we design ont-hot. We want a encoding form that wipe off correlation)
  - we want a featurized representation
    - here we come to **word embedding**
    -

    - It's a matrix, each rows contain correlation values that reflect the correlation between words.
      - here Pro.Andrew give a intuition of meaning for each rows. In fact, the meaning usually isn't explainable.

- visualize word embeddings with **T-SNE** algorithm to gain better intuition. (**no-linear** mapping)

- Using word embeddings
  - example:

    - The algorithms that are used to learn **word embeddings** can examine billions of words of unlabeled text
    - in pic above. if you have tested your model with this sentence "**Mahmoud Badry** is a *durian* cultivator" the network should learn the name even if it hasn't seen the word *durian* before (during training).
  - **transfer** learning

  - interesting relationship to the face recognition task:

- face recognition: encode each face into a vector and then check how similar are these vectors.
- word embeddings: learning a representation for each word in our vocabulary

- Properties of word embeddings:
  - **analogy reasoning**
    - example

Analogies

	Man (5391)	Woman (9853)	King (4914)	Queen (7157)	Apple (456)	Orange (6257)
Gender	−1	1	-0.95	0.97	0.00	0.01
Royal	0.01	0.02	0.93	0.95	-0.01	0.00
Age	0.03	0.02	0.70	0.69	0.03	-0.02
Food	0.09	0.01	0.02	0.01	0.95	0.97

e_{5391} e_{man} e_{woman}

Man → Woman as King → ?

$e_{man} - e_{woman} \approx \begin{bmatrix} -2 \\ 0 \\ 0 \\ 0 \end{bmatrix}$

$e_{king} - e_{queen} \approx \begin{bmatrix} -2 \\ 0 \\ 0 \\ 0 \end{bmatrix}$

- which can use to find similar word.

300 D

$$e_{man} - e_{woman} \approx e_{king} - \cancel{e_?}\ e_w$$

Find word w: $\arg\max_w \ \mathrm{sim}(e_w, e_{king} - e_{man} + e_{woman})$

- use **cosine similarity** as metric

- embedding matrix

- In practice, use specialized function to look up an embedding.
- algorithm should create a (m,n) embedding matrix E.
  - m: vocabulary size
  - n: the number of features we want to extract.
- Generally *np.dot(**E**, O$_j$) = e$_j$*
- In practice it's not efficient to use a dot multiplication when trying to extract the embeddings of a specific word.
- Instead, we use slicing to slice a specific column.

Algorithm

- At the start, word embeddings algorithms were complex but then they got simpler and simpler.
- the language modeling problem poses a machines learning problem where you input the context (like the last four words) and predict some target words.
- **Neural language model**

- Suppose we have an example: "I want a glass of orange **juice** to go along with my cereal"
- To learn **juice**, choices of **context** are:

  - case 4: Nearby 1 word.
    - "glass" word is near juice.
    - This is the idea of **skip grams** model.
    - The idea is much simpler and works remarkably well.

- word2vec
  - **skip grams**

    - The target is chosen randomly based on a window with a specific size.
    - We have converted the problem into a **supervised** problem.
  - model

- notation: context word are `c` and the target word is `t`
- task: learn `c` to `t`
- loss: cross-entropy

$$p(t|c) = \frac{e^{\theta_t^T e_c}}{\sum_{j=1}^{10,000} e^{\theta_j^T e_c}}$$

- softmax:
  - problem:
    - computational cost is expensive as we go through whole vocabulary (example here has 10,000 words in vocabulary)
  - solution:
    - use "**Hierarchical softmax classifier**" which works as a tree classifier.

    - not a balanced tree. Common words are at the top and less common are at the bottom.
- how to sample context c?
  - don't take the context uniformly random, instead there are some heuristics to balance the common words and the non-common words.

negative sampling

Model

Softmax: $$p(t|c) = \frac{e^{\theta_t^T e_c}}{\sum_{j=1}^{10,000} e^{\theta_j^T e_c}}$$

$$P(y=1 \mid c,t) = \sigma(\theta_t^T e_c)$$

context	word	target?
orange	juice	1
orange	king	0
orange	book	0
orange	the	0
orange	of	0

c t y

-
- sampling:
  - positive example sampling: the same as skip grams
  - negative example sampling: pick words according to empirical frequencies from the vocabulary.
    - because there are many prep. in vocabulary which have high frequency.
    - author give this formula for sampling

$$P(w_i) = \frac{f(w_i)^{3/4}}{\sum_{j=1}^{10,000} f(w_j)^{3/4}}$$

    -
    - f(w) means the frequency of word w
- parameter k:
  - a ratio of k negative examples to 1 positive ones
  - k is recommended to be from 5 to 20 in small data sets. For larger ones - 2 to 5.
- using the simple logistic regression model.

GloVe

- GloVe stands for Global vectors for word representation.

GloVe (global vectors for word representation)

I want a glass of orange juice to go along with my cereal.

c, t

X_{ij} = # times i appears in context of j.

c t t c

-
  - X_{ij} = # times i appears in context of j
  - X_{ij} is a count that captures how often do words i and j appear with each other, or close to each other.
- model

- Model

$$\text{minimize} \sum_{i=1}^{10{,}000} \sum_{j=1}^{10{,}000} \underbrace{f(X_{ij})}_{\text{weighting term}} \left(\theta_i^T e_j + b_i + b_j' - \log X_{ij} \right)^2$$

$$\text{“}\theta_t^T e_c\text{”}$$

$f(X_{ij}) = 0$ if $X_{ij} = 0$. “$0 \log 0$” $= 0$

  - f(x) the weighting term
    - solve `log(0).` This might occur if there are no pairs for the given target and context values.
    - Giving not too much weight for stop words like "is", "the", and "this" which occur many times.
    - Giving not too little weight for infrequent words.
- conclusion
  - word embeddings are very computationally expensive to train, most ML practitioners will load a pre-trained set of embeddings.
  - can't guarantee that the axis used to represent the features will be well-aligned with what might be easily humanly interpretable axis like gender, royal, age.

Application

- sentiment classification

x	y
The dessert is excellent.	★★★★☆
Service was quite slow.	★★☆☆☆
Good for a quick meal, but nothing special.	★★★☆☆
Completely lacking in good taste, good service, and good ambience.	★☆☆☆☆

  -
  - might not have a huge labeled training data. But word embedding help to solve this.

-
- RNN many to one

debiasing word embedding

- make sure that our word embeddings are free from undesirable forms of bias, such as gender bias, ethnicity bias and so on.
- Word embeddings can reflect gender, ethnicity, age, sexual orientation, and other biases of text used to train the model.
- example of bias:
  - Man : Computer_programmer as Woman : **Homemaker**
  - Father : Doctor as Mother : **Nurse**
- **steps:**
  - Identify the direction:
    -

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

  - Equalize pairs

3.1 - Neutralize bias for non-gender specific words

The figure below should help you visualize what neutralizing does. If you're using a 50-dimensional word embedding, the 50 dimensional space can be split into two parts: The bias-direction g, and the remaining 49 dimensions, which we'll call $g_\perp$. In linear algebra, we say that the 49 dimensional $g_\perp$ is perpendicular (or "othogonal") to g, meaning it is at 90 degrees to g. The neutralization step takes a vector such as $e_{receptionist}$ and zeros out the component in the direction of g, giving us $e_{receptionist}^{debiased}$.

Even though $g_\perp$ is 49 dimensional, given the limitations of what we can draw on a screen, we illustrate it using a 1 dimensional axis below.

Figure 2: The word vector for "receptionist" represented before and after applying the neutralize operation.

3.2 - Equalization algorithm for gender-specific words

Next, lets see how debiasing can also be applied to word pairs such as "actress" and "actor." Equalization is applied to pairs of words that you might want to have differ only through the gender property. As a concrete example, suppose that "actress" is closer to "babysit" than "actor." By applying neutralizing to "babysit" we can reduce the gender-stereotype associated with babysitting. But this still does not guarantee that "actor" and "actress" are equidistant from "babysit." The equalization algorithm takes care of this.

The key idea behind equalization is to make sure that a particular pair of words are equi-distant from the 49-dimensional $g_\perp$. The equalization step also ensures that the two equalized steps are now the same distance from $e_{receptionist}^{debiased}$, or from any other work that has been neutralized. In pictures, this is how equalization works:

The derivation of the linear algebra to do this is a bit more complex. (See Bolukbasi et al., 2016 for details.) But the key equations are:

$$\mu = \frac{e_{w1} + e_{w2}}{2} \tag{4}$$

$$\mu_B = \frac{\mu \cdot \text{bias_axis}}{\|\text{bias_axis}\|_2^2} * \text{bias_axis} \tag{5}$$

$$\mu_\perp = \mu - \mu_B \tag{6}$$

$$e_{w1B} = \frac{e_{w1} \cdot \text{bias_axis}}{\|\text{bias_axis}\|_2^2} * \text{bias_axis} \tag{7}$$

$$e_{w2B} = \frac{e_{w2} \cdot \text{bias_axis}}{\|\text{bias_axis}\|_2^2} * \text{bias_axis} \tag{8}$$

$$e_{w1B}^{corrected} = \sqrt{|1 - \|\mu_\perp\|_2^2|} * \frac{e_{\text{w1B}} - \mu_B}{|(e_{w1} - \mu_\perp) - \mu_B)|} \tag{9}$$

$$e_{w2B}^{corrected} = \sqrt{|1 - \|\mu_\perp\|_2^2|} * \frac{e_{\text{w2B}} - \mu_B}{|(e_{w2} - \mu_\perp) - \mu_B)|} \tag{10}$$

$$e_1 = e_{w1B}^{corrected} + \mu_\perp \tag{11}$$

$$e_2 = e_{w2B}^{corrected} + \mu_\perp \tag{12}$$

What you should remember:

- If you have an NLP task where the training set is small, using word embeddings can help your algorithm significantly. Word embeddings allow your model to work on words in the test set that may not even have appeared in your training set.
- Training sequence models in Keras (and in most other deep learning frameworks) requires a few important details:
  - To use mini-batches, the sequences need to be padded so that all the examples in a mini-batch have the same length.
  - An `Embedding()` layer can be initialized with pretrained values. These values can be either fixed or trained further on your dataset. If however your labeled dataset is small, it's usually not worth trying to train a large pre-trained set of embeddings.
  - `LSTM()` has a flag called `return_sequences` to decide if you would like to return every hidden states or only the last one.
  - You can use `Dropout()` right after `LSTM()` to regularize your network.