As the inner product between 2 one-hot vectors is 0, it is impossible for the algorithms to capture relationships.

- Solution: word embeddings ⇒ featurized representation.

Featurized representation: word embedding

	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	0.7	0.69	0.03	-0.02
Food	0.04	0.01	0.02	0.01	0.95	0.97

e_{5391}

quite similar features

These features aren't easy to interpret.

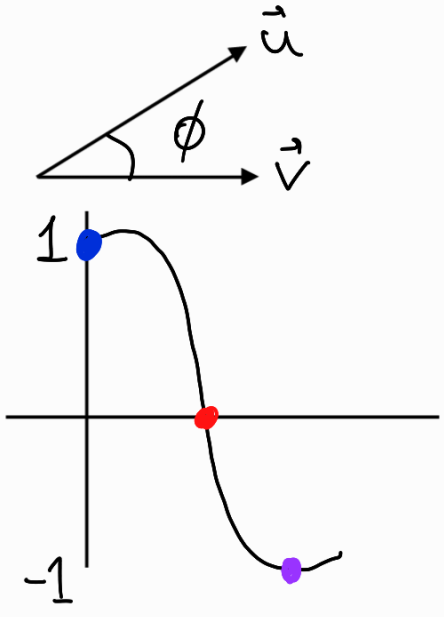
Properties of word embeddings:

- When we compute the vector difference $e_i - e_j$:
  - ↳ Resulting feature is big ⇒ opposite (man, woman)
  - ↳ Resulting feature is small ⇒ same (royalty, between King and queen)
- We can get the similarity in the relationship between terms.

This similarity is usually the cosine similarity:

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

→ big: very similar
→ small: very different



- $\vec{u}$ $\vec{v}$ → $\phi = 0°$, cosine similarity = 1
- $\vec{u}$, $\vec{v}$ → $\phi = 90°$, cosine similarity = 0
- $\vec{u}$ $\vec{v}$ → $\phi = 180°$, cosine similarity = -1

This way, we can conclude that,

Ottawa is to Canada as Nairobi to Kenya

So we get an embedding matrix, that multiplied by a term's one-hot vector, gives us all the features of the term.

But, how do we really learn word embeddings?

- Word2vec (skip-gram model):

We take a context word, and within a ±window, a target to predict.

We represent the input with a one-hot vector O_c, multiply it by the embedding matrix $E = e_c$

So, we take e_c and feed it to a softmax unit.

Softmax: $p(t|c) = \frac{e^{\theta_t^T e_c}}{\sum_{j=1}^{vocab\_size} e^{\theta_j^T e_c}}$; → Slow!! ↗ hierarchical softmax

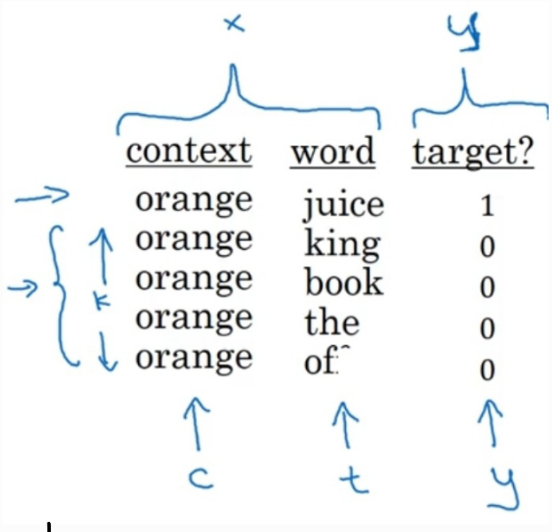
where θ_T = parameter associated with output t

$\mathcal{L}(\hat{y}, y) = -\sum_{i=1}^{vocab\_size} y_i \log \hat{y}_i$; y is one-hot vector

Negative sampling:

Take a valid context-word pair → target = 1

Choose K-1 negative samples.



We then, define a logistic regression model.

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

This way, we select K combinations and train our classifier on just them.

e_c → o King?
→ o juice?
→ o book?
(...)

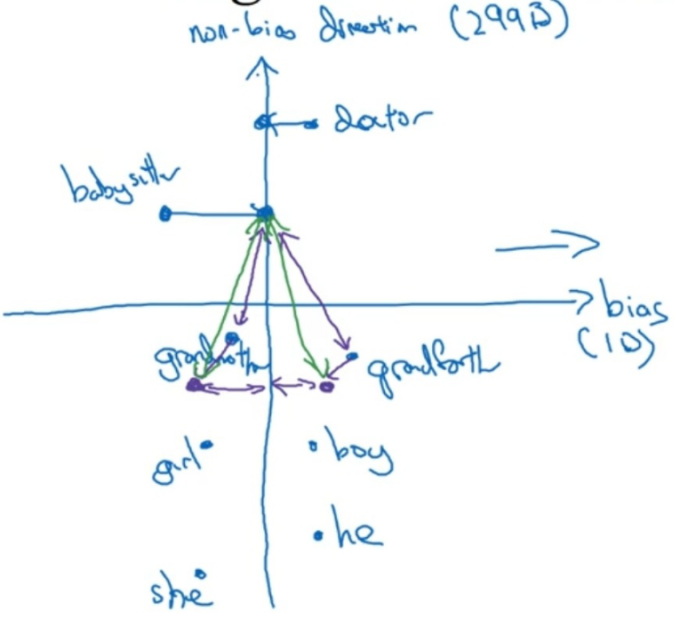
How to choose the negatives?

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

Debiasing word embeddings:

Word embeddings can reflect gender, ethnicity, age, sexual orientation, and other biases of the text used to train the model.

Addressing bias in word embeddings



1. Identify bias direction.

$e_{he} - e_{she}$

$e_{male} - e_{female}$

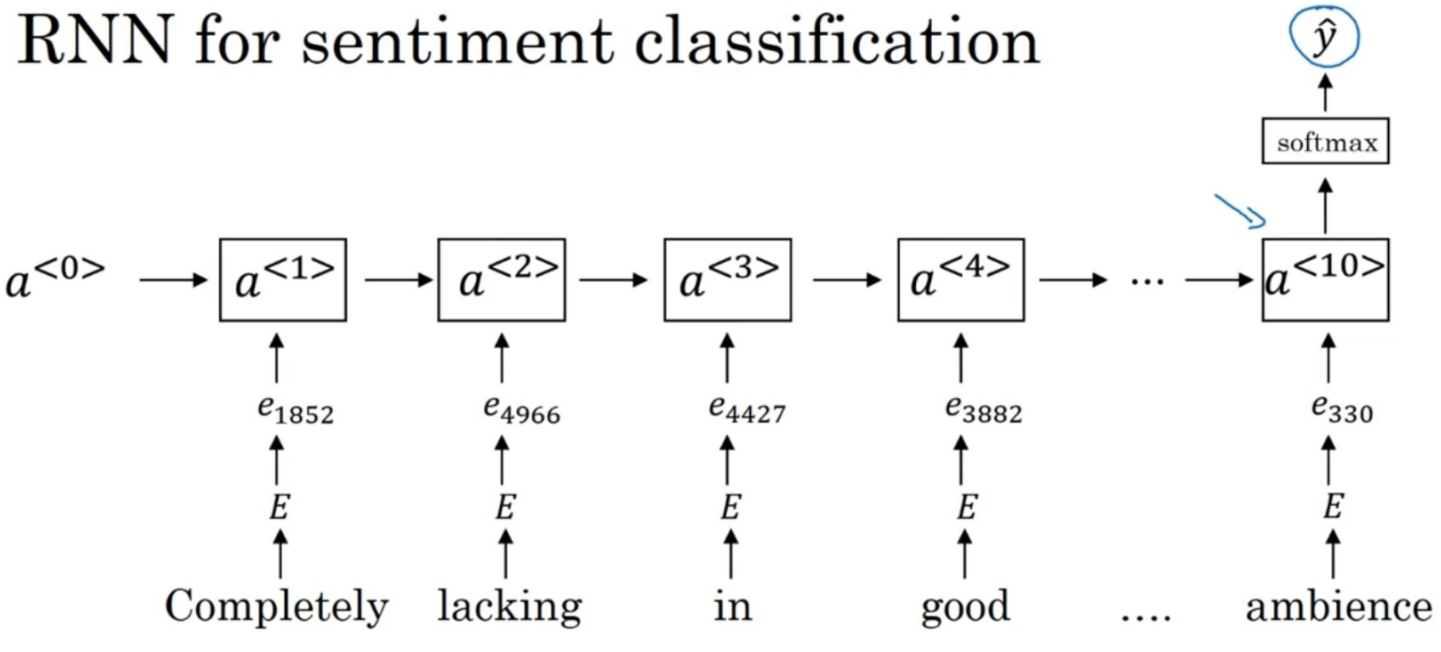
average

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

3. Equalize pairs.

→ grandmother − grandfather
girl boy

RNN for sentiment classification



many-to-one

What you should remember:

- Cosine similarity is a good way to compare the similarity between pairs of word vectors.
  - Note that L2 (Euclidean) distance also works.
- For NLP applications, using a pre-trained set of word vectors is often a great way to get started.

2 - Embedding Vectors Versus One-Hot Vectors

Recall from the lesson videos that one-hot vectors don't do a good job of capturing the level of similarity between words. This is because every one-hot vector has the same Euclidean distance from any other one-hot vector.

Embedding vectors, such as GloVe vectors, provide much more useful information about the meaning of individual words.
Now, see how you can use GloVe vectors to measure the similarity between two words!

3 - Cosine Similarity

To measure the similarity between two words, you need a way to measure the degree of similarity between two embedding vectors for the two words. Given two vectors u and v, cosine similarity is defined as follows:

$$\text{CosineSimilarity(u, v)} = \frac{u \cdot v}{||u||_2||v||_2} = cos(\theta) \tag{1}$$

- $u \cdot v$ is the dot product (or inner product) of two vectors
- $||u||_2$ is the norm (or length) of the vector u
- θ is the angle between u and v.
- The cosine similarity depends on the angle between u and v.
  - If u and v are very similar, their cosine similarity will be close to 1.
  - If they are dissimilar, the cosine similarity will take a smaller value.

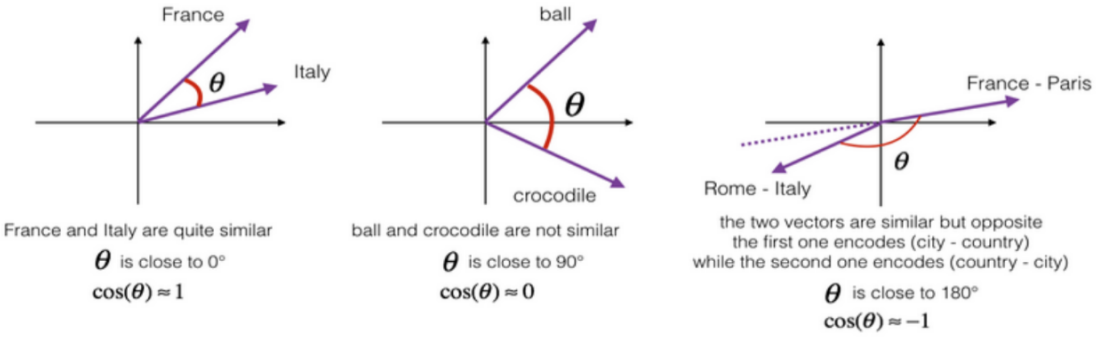


Figure 1: The cosine of the angle between two vectors is a measure of their similarity.

5.1 - 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 is called $g_\perp$ here. In linear algebra, we say that the 49-dimensional $g_\perp$ is perpendicular (or "orthogonal") 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 $e^{debiased}_{receptionist}$.

Even though $g_\perp$ is 49-dimensional, given the limitations of what you can draw on a 2D screen, it's illustrated using a 1-dimensional axis below.

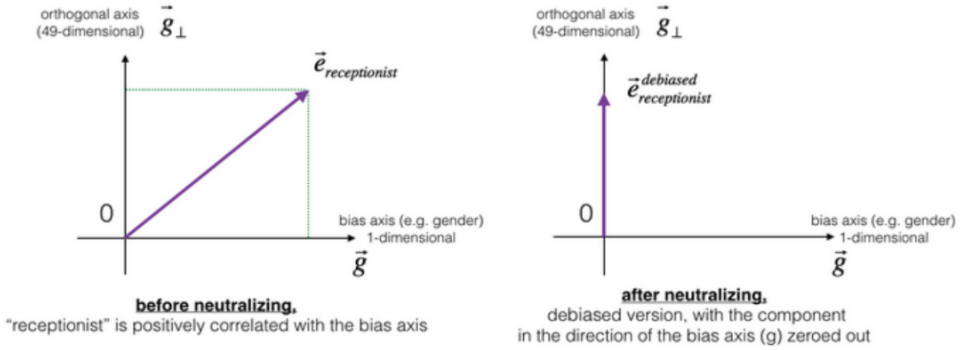
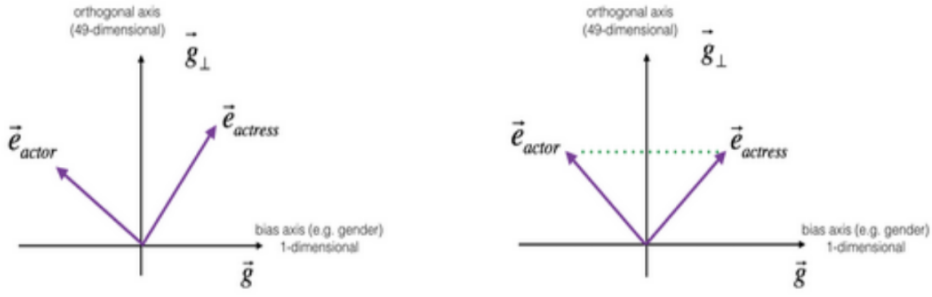


Figure 2: The word vector for "receptionist" represented before and after applying the neutralize operation.

5.2 - Equalization Algorithm for Gender-Specific Words

Next, let's 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 neutralization to "babysit," you 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 equidistant from the 49-dimensional $g_\perp$. The equalization step also ensures that the two equalized steps are now the same distance from $e^{debiased}_{receptionist}$, or from any other work that has been neutralized. Visually, this is how equalization works:



before equalizing,
"actress" and "actor" differ in many ways beyond the direction of $\vec{g}$

after equalizing,
"actress" and "actor" differ only in the direction of $\vec{g}$, and further are equal in distance from $\vec{g}_\perp$

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

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

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

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

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

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

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

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

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

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