SEQUENCE MODELS · COURSERA

NLP & WORD EMBEDDINGS

MAN IS TO WOMAN AS KING IS TO QUEEN

PROBLEM: THE ONE-HOT REPR O_{456} OF APPLE HAS NO INFO ABOUT ITS RELATIONSHIP TO O_{6257} ORANGE

I WANT A GLASS OF ORANGE _____
I WANT A GLASS OF APPLE _____

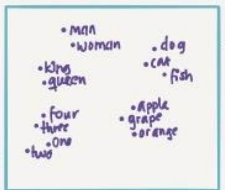
SOLUTION: CREATE A MATRIX OF FEATURES TO DESCRIBE THE WORDS

WORD EMBEDDINGS

	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}

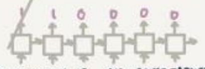
IN REALITY · THE FEATURES ARE LEARNED & NOT AS STRAIGHTFWD AS GENDER/AGE



t-SNE
VISUAL REPRESENT OF 300D WORD EMBEDDINGS

USING WORD EMBEDDINGS

EX. NAME/ENTITY RECOGN



SALLY SMITH IS AN ORANGE FARMER
ROBERT LIN IS AN APPLE FARMER
BOB JONES IS A DURIAN CULTIVATOR

WITH WORD EMBEDDINGS WE UNDERSTAND THAT AN ORANGE FARMER IS A PERSON ⇒ SALLY SMITH = NAME

- APPLE ~ ORANGE ⇒ APPLE FARMER = PERSON
- USING WORD EMBEDDINGS TRAINED ON LOTS OF TEXT WE ALSO GET EMB. FOR MORE UNCOMMON WORDS (DURIAN, CULTIVATOR)

EX. MAN IS TO WOMAN AS KING IS TO ?

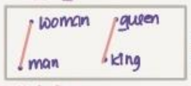
E = EMBEDDING MATRIX

e_{man}

	MAN 5391	WOMAN 9853	KING 4914	QUEEN 7157
GENDER	−1	1	−0.95	0.97
ROYAL	0.01	0.02	0.93	0.95
AGE	0.03	0.02	0.7	0.69
FOOD	0.04	0.01	0.02	0.01

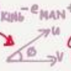
$$e_{man} - e_{woman} \quad e_{king} - e_{queen}$$

$$\begin{bmatrix}-2\\0\\0\\0\end{bmatrix} \xrightarrow{\text{VERY SIMILAR}} \begin{bmatrix}-2\\0\\0\\0\end{bmatrix}$$



FIND MAN→WOMAN KING→?

FIND(w):
$$\text{ARG.MAX}\ \text{SIM}(e_w, e_{king} - e_{man} + e_{woman})$$
$$\text{SIM}(u,v) = \frac{u^T v}{\|u\|_2 \|v\|_2}$$



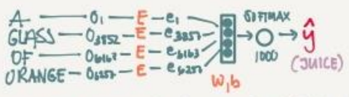
COSINE SIMILARITY

LEARNING WORD EMBEDDINGS

HOW DO WE LEARN THE EMBEDDING MATRIX E?

IDEA 1: USING A NEURAL LANG MODEL

I WANT A GLASS OF ORANGE $\hat{y}$



WE CAN USE DIFFERENT CONTEXTS THAN THE LAST 4 WORDS

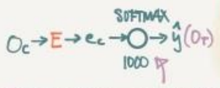
- LAST 4 WORDS
- 4 WORDS LEFT + RIGHT
- LAST 1 WORD
- NEARBY 1 WORD ← SKIPGRAM: RANDOM WITHIN EX 5 WORDS

IDEA 2: SKIP-GRAMS WORD2VEC

I WANT A GLASS OF ORANGE JUICE TO GO ALONG WITH MY CEREAL

PICK RANDOM CONTEXT/TARGET PAIRS (WITHIN EX 5 WORDS)

CONTEXT	TARGET
ORANGE	JUICE
ORANGE	GLASS
ORANGE	MY
...	...



NOTE: WHILE THIS SIMPLE NN PREDICTS O_T OUR REAL GOAL IS TO LEARN E

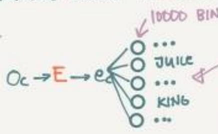
THIS IS VERY COMPUTATIONALLY EXPENSIVE BUT WE CAN OPTIMIZE BY USING A HIERARCHICAL SOFTMAX CLASSIFIER

IDEA: NEGATIVE SAMPLING

1. PICK A CONTEXT/TARGET PAIR AS A POSITIVE EXAMPLE
2. PICK A FEW NEG EXAMPLES CONTEXT + RANDOM

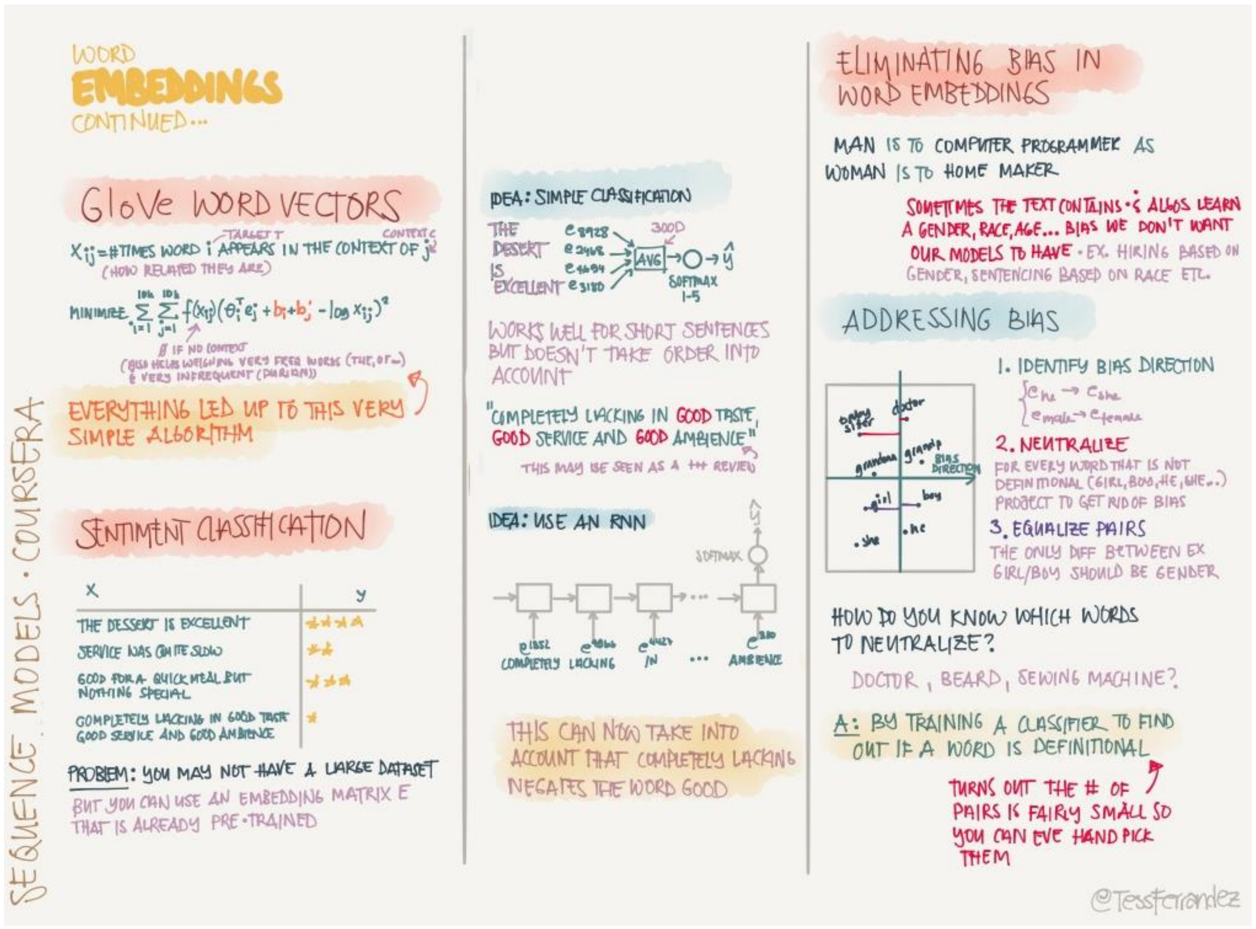
CONTEXT	WORD	TARGET
ORANGE	JUICE	1
ORANGE	KING	0
ORANGE	BOOK	0
ORANGE	THE	0
ORANGE	OF	0

NOTE: SOMETIMES BY CHANCE YOU PICK A POS PAIR · BUT IT DOESN'T MATTER



YOU ONLY TRAIN 5 FOR EACH CONTEXT WORD ⇒ EFFICIENT TO TRAIN

@TessFerrandez


WORD EMBEDDINGS CONTINUED...
SEQUENCE MODELS · COURSERA
GloVe WORD VECTORS
X_{ij} = # TIMES WORD i (TARGET T) APPEARS IN THE CONTEXT OF j (CONTEXT C)
(HOW RELATED THEY ARE)
MINIMIZE $\sum_{i=1}^{10k}\sum_{j=1}^{10k} f(X_{ij})(\theta_i^T e_j + b_i + b_j' - \log X_{ij})^2$
0 IF NO CONTEXT
(ALSO HELPS WEIGHING VERY FREQ WORDS (THE, OF...) & VERY INFREQUENT (DURIAN))
EVERYTHING LED UP TO THIS VERY SIMPLE ALGORITHM
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
PROBLEM: YOU MAY NOT HAVE A LARGE DATASET
BUT YOU CAN USE AN EMBEDDING MATRIX E THAT IS ALREADY PRE-TRAINED
IDEA: SIMPLE CLASSIFICATION
THE DESERT IS EXCELLENT
e_{8928} e_{2468} e_{4694} e_{3180}
300D
AVG
$\hat{y}$
SOFTMAX 1-5
WORKS WELL FOR SHORT SENTENCES BUT DOESN'T TAKE ORDER INTO ACCOUNT
"COMPLETELY LACKING IN GOOD TASTE, GOOD SERVICE AND GOOD AMBIENCE"
THIS MAY BE SEEN AS A +++ REVIEW
IDEA: USE AN RNN
SOFTMAX
$\hat{y}$
e^{1852} COMPLETELY
e^{4966} LACKING
e^{4427} IN
...
e^{330} AMBIENCE
THIS CAN NOW TAKE INTO ACCOUNT THAT COMPLETELY LACKING NEGATES THE WORD GOOD
ELIMINATING BIAS IN WORD EMBEDDINGS
MAN IS TO COMPUTER PROGRAMMER AS WOMAN IS TO HOME MAKER
SOMETIMES THE TEXT CONTAINS : & ALGOS LEARN A GENDER, RACE, AGE... BIAS WE DON'T WANT OUR MODELS TO HAVE · EX. HIRING BASED ON GENDER, SENTENCING BASED ON RACE ETC.
ADDRESSING BIAS
babysitter
doctor
grandma
grandpa
BIAS DIRECTION
girl
boy
she
he
1. IDENTIFY BIAS DIRECTION
$e_{he} \rightarrow e_{she}$
$e_{male} \rightarrow e_{female}$
2. NEUTRALIZE
FOR EVERY WORD THAT IS NOT DEFINITIONAL (GIRL, BOY, HE, SHE..) PROJECT TO GET RID OF BIAS
3. EQUALIZE PAIRS
THE ONLY DIFF BETWEEN EX GIRL/BOY SHOULD BE GENDER
HOW DO YOU KNOW WHICH WORDS TO NEUTRALIZE?
DOCTOR, BEARD, SEWING MACHINE?
A: BY TRAINING A CLASSIFIER TO FIND OUT IF A WORD IS DEFINITIONAL
TURNS OUT THE # OF PAIRS IS FAIRLY SMALL SO YOU CAN EVE HAND PICK THEM
@TessFerrandez

Natural Language Processing & Word Embeddings

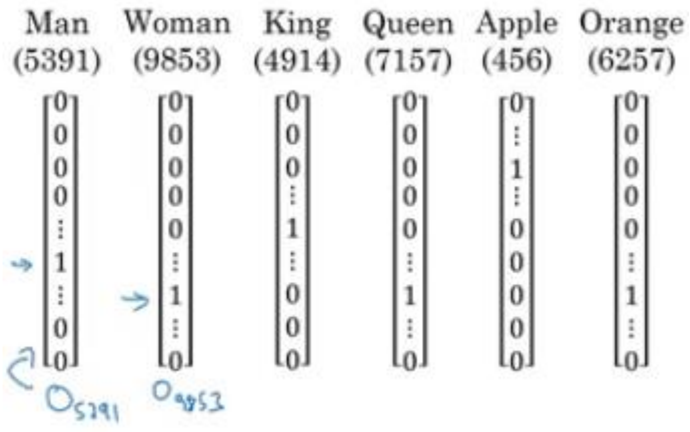
Natural language processing with deep learning is an important combination. Using word vector representations and embedding layers you can train recurrent neural networks with outstanding performances in a wide variety of industries. Examples of applications are sentiment analysis, named entity recognition and machine translation.

Introduction to Word Embeddings

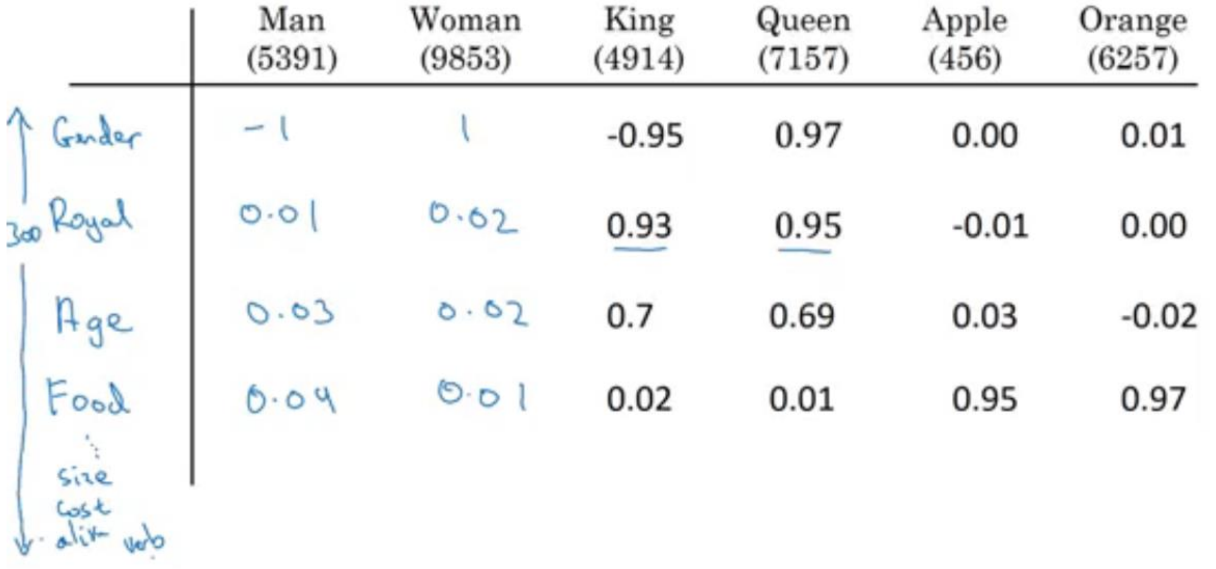
Word Representation

- NLP has been revolutionized by deep learning and especially by RNNs and deep RNNs.

- Word embeddings is a way of representing words. It lets your algorithm automatically understand the analogies between words like "king" and "queen".
- So far we have defined our language by a vocabulary. Then represented our words with a one-hot vector that represents the word in the vocabulary.
    - An image example would be:



    - We will use the annotation **O** $_{idx}$ for any word that is represented with one-hot like in the image.
    - One of the weaknesses of this representation is that it treats a word as a thing that itself and it doesn't allow an algorithm to generalize across words.
        - For example: "I want a glass of **orange** ______", a model should predict the next word as **juice**.
        - A similar example "I want a glass of **apple** ______", a model won't easily predict **juice** here if it wasn't trained on it. And if so the two examples aren't related although orange and apple are similar.
    - Inner product between any one-hot encoding vector is zero. Also, the distances between them are the same.
- So, instead of a one-hot presentation, won't it be nice if we can learn a featurized representation with each of these words: man, woman, king, queen, apple, and orange?

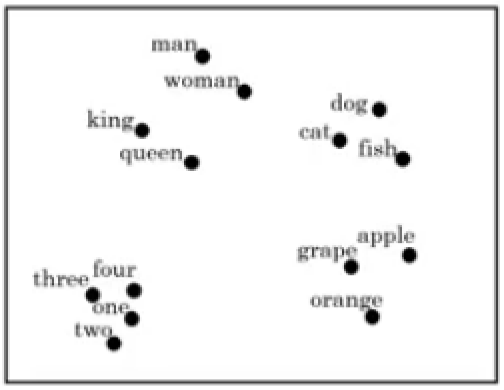


	Man (5391)	Woman (9853)	King (4914)	Queen (7157)	Apple (456)	Orange (6257)
Gender	-1	1	-0.95	0.97	0.00	0.01
Royal	0.01	0.02	0.93	0.95	-0.01	0.00
Age	0.03	0.02	0.7	0.69	0.03	-0.02
Food	0.04	0.01	0.02	0.01	0.95	0.97

300

size
cost
alive verb

- Each word will have a, for example, 300 features with a type of float point number.
  - Each word column will be a 300-dimensional vector which will be the representation.
  - We will use the notation **e**$_{5391}$ to describe **man** word features vector.
  - Now, if we return to the examples we described again:
    - "I want a glass of **orange** ______"
    - I want a glass of **apple** ______
  - Orange and apple now share a lot of similar features which makes it easier for an algorithm to generalize between them.
  - We call this representation **Word embeddings**.
- To visualize word embeddings we use a t-SNE algorithm to reduce the features to 2 dimensions which makes it easy to visualize:

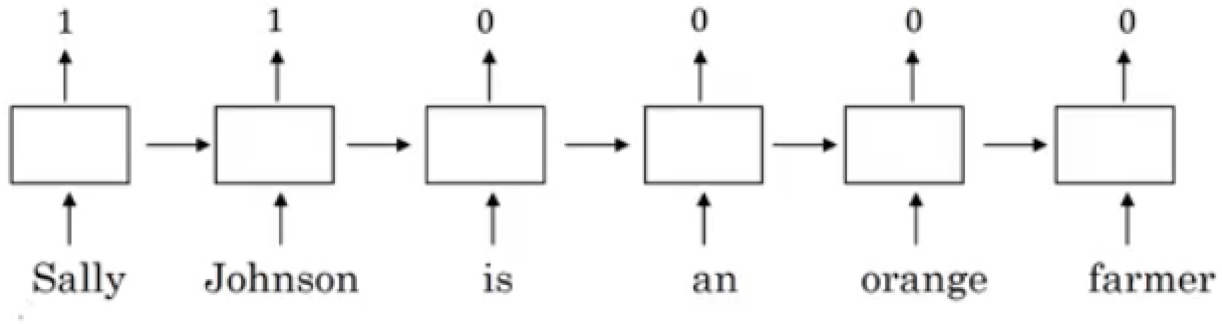


  - You will get a sense that more related words are closer to each other.

- The **word embeddings** came from that we need to embed a unique vector inside a n-dimensional space.

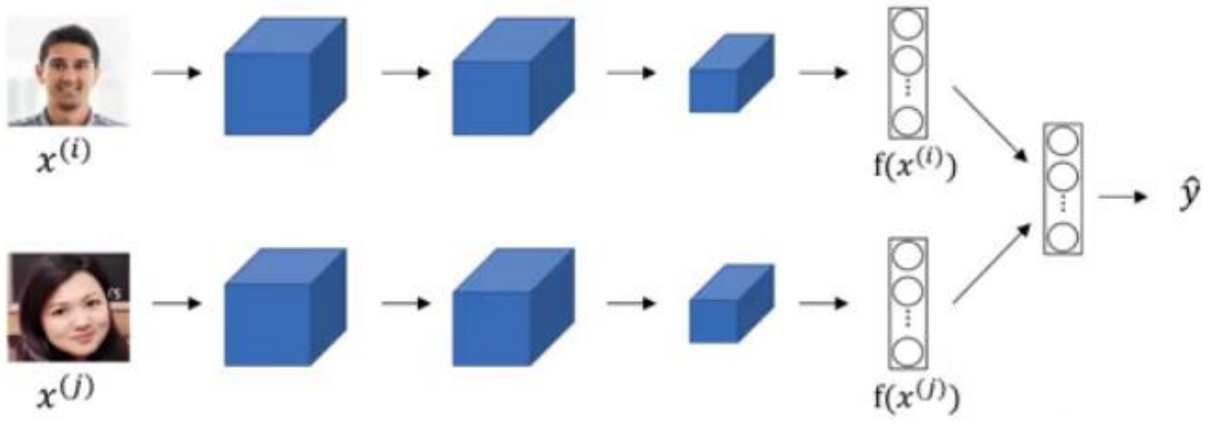
Using word embeddings

- Let's see how we can take the feature representation we have extracted from each word and apply it in the Named entity recognition problem.
- Given this example (from named entity recognition):



- **Sally Johnson** is a person's name.
- After training on this sentence the model should find out that the sentence "**Robert Lin** is an *apple* farmer" contains Robert Lin as a name, as apple and orange have near representations.
- Now 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). That's the power of word representations.
- The algorithms that are used to learn **word embeddings** can examine billions of words of unlabeled text - for example, 100 billion words and learn the representation from them.
- Transfer learning and word embeddings:
    1. Learn word embeddings from large text corpus (1-100 billion of words).
        - Or download pre-trained embedding online.
    2. Transfer embedding to new task with the smaller training set (say, 100k words).
    3. Optional: continue to finetune the word embeddings with new data.
        - You bother doing this if your smaller training set (from step 2) is big enough.
- Word embeddings tend to make the biggest difference when the task you're trying to carry out has a relatively smaller training set.

- Also, one of the advantages of using word embeddings is that it reduces the size of the input!
    - 10,000 one hot compared to 300 features vector.
- Word embeddings have an interesting relationship to the face recognition task:

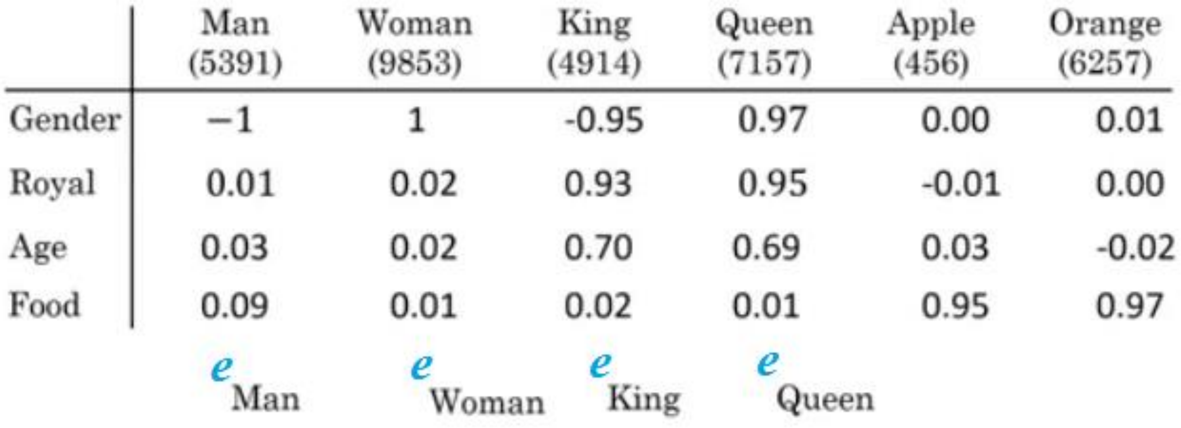


    - In this problem, we encode each face into a vector and then check how similar are these vectors.
    - Words **encoding** and **embeddings** have a similar meaning here.
- In the word embeddings task, we are learning a representation for each word in our vocabulary (unlike in image encoding where we have to map each new image to some n-dimensional vector). We will discuss the algorithm in next sections.

Properties of word embeddings

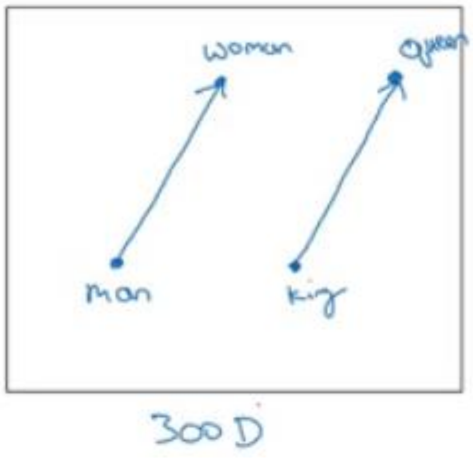
- One of the most fascinating properties of word embeddings is that they can also help with analogy reasoning. While analogy reasoning may not be by itself the most important NLP application, but it might help convey a sense of what these word embeddings can do.
- Analogies example:

- Given this word embeddings table:



	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_{Man}	e_{Woman}	e_{King}	e_{Queen}		

- Can we conclude this relation:
  - Man ==> Woman
  - King ==> ??
- Lets subtract e_{Man} from e_{Woman}. This will equal the vector `[-2 0 0 0]`
- Similar e_{King} - e_{Queen} = `[-2 0 0 0]`
- So the difference is about the gender in both.



  - This vector represents the gender.
  - This drawing is a 2D visualization of the 4D vector that has been extracted by a t-SNE algorithm. It's a drawing just for visualization. Don't rely on the t-SNE algorithm for finding parallels.
- So we can reformulate the problem to find:
  - $e_{\text{Man}} - e_{\text{Woman}} \approx e_{\text{King}} - e_{??}$
- It can also be represented mathematically by:

$$\arg\max_{w} \ \text{sim}(e_w, e_{king} - e_{man} + e_{woman})$$

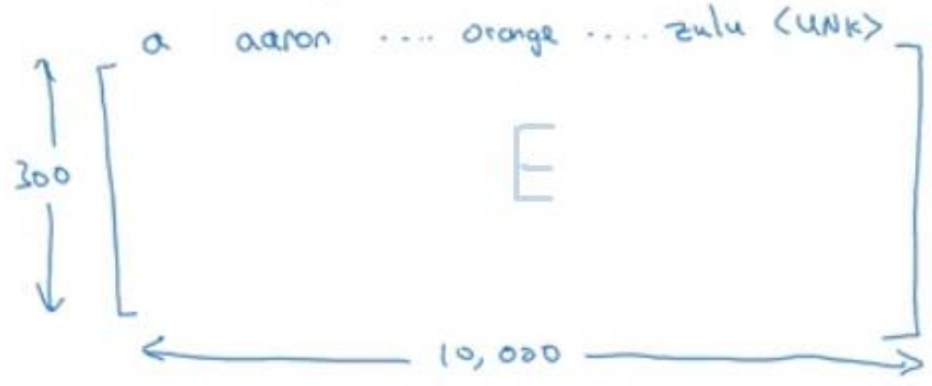
    - It turns out that e_{Queen} is the best solution here that gets the the similar vector.
- Cosine similarity - the most commonly used similarity function:
    - Equation:

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

        - `CosineSimilarity(u, v) = u . v / ||u|| ||v||` = cos(θ)
        - The top part represents the inner product of `u` and `v` vectors. It will be large if the vectors are very similar.
- You can also use Euclidean distance as a similarity function (but it rather measures a dissimilarity, so you should take it with negative sign).
- We can use this equation to calculate the similarities between word embeddings and on the analogy problem where `u`= e_w and `v` = e_{king} - e_{man} + e_{woman}

Embedding matrix

- When you implement an algorithm to learn a word embedding, what you end up learning is a **embedding matrix**.
- Let's take an example:
    - Suppose we are using 10,000 words as our vocabulary (plus token).
    - The algorithm should create a matrix `E` of the shape (300, 10000) in case we are extracting 300 features.



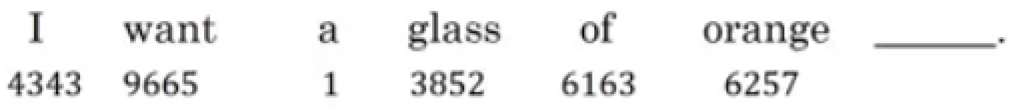
    - If O_{6257} is the one hot encoding of the word **orange** of shape (10000, 1), then
      np.dot(`E`*,*O_{6257}*)* = e_{6257} which shape is (300, 1).
    - Generally *np.dot(*`E`*,* O_j*)* = e_j
- In the next sections, you will see that we first initialize `E` randomly and then try to learn all the parameters of this matrix.

- In practice it's not efficient to use a dot multiplication when you are trying to extract the embeddings of a specific word, instead, we will use slicing to slice a specific column. In Keras there is an embedding layer that extracts this column with no multiplication.

Learning Word Embeddings: Word2vec & GloVe

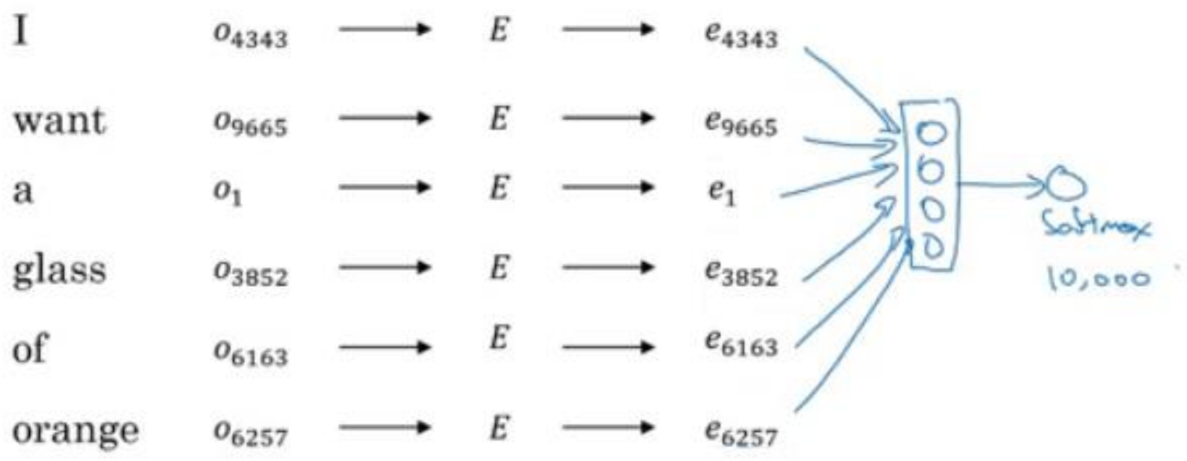
Learning word embeddings

- Let's start learning some algorithms that can learn word embeddings.
- At the start, word embeddings algorithms were complex but then they got simpler and simpler.
- We will start by learning the complex examples to make more intuition.
- **Neural language model**:
  - Let's start with an example:



I	want	a	glass	of	orange	______.
4343	9665	1	3852	6163	6257	

  - We want to build a language model so that we can predict the next word.
  - So we use this neural network to learn the language model



    - We get e_j by `np.dot(E,o<sub>j</sub>)`
    - NN layer has parameters `W1` and `b1` while softmax layer has parameters `W2` and `b2`
    - Input dimension is (300*6, 1) if the window size is 6 (six previous words).

    - Here we are optimizing `E` matrix and layers parameters. We need to maximize the likelihood to predict the next word given the context (previous words).
  - This model was build in 2003 and tends to work pretty decent for learning word embeddings.
- In the last example we took a window of 6 words that fall behind the word that we want to predict. There are other choices when we are trying to learn word embeddings.
  - 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:
    a. Last 4 words.
       - We use a window of last 4 words (4 is a hyperparameter), "a glass of orange" and try to predict the next word from it.
    b. 4 words on the left and on the right.
       - "a glass of orange" and "to go along with"
    c. Last 1 word.
       - "orange"
    d. 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.
       - We will talk about this in the next section.
- Researchers found that if you really want to build a *language model*, it's natural to use the last few words as a context. But if your main goal is really to learn a *word embedding*, then you can use all of these other contexts and they will result in very meaningful work embeddings as well.
- To summarize, the language modeling problem poses a machines learning problem where you input the context (like the last four words) and predict some target words. And posing that problem allows you to learn good word embeddings.

Word2Vec

- Before presenting Word2Vec, lets talk about **skip-grams**:

  - For example, we have the sentence: "I want a glass of orange juice to go along with my cereal"

- We will choose **context** and **target**.
- The target is chosen randomly based on a window with a specific size.

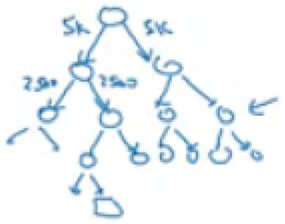
Context	Target	
orange	juice	+1
orange	glass	-2
orange	my	+6

- We have converted the problem into a supervised problem.
- This is not an easy learning problem because learning within -10/+10 words (10 - an example) is hard.
- We want to learn this to get our word embeddings model.

- Word2Vec model:
  - Vocabulary size = 10,000 words
  - Let's say that the context word are `c` and the target word is `t`
  - We want to learn `c` to `t`
  - We get e_c by `E`. o_c
  - We then use a softmax layer to get `P(t|c)` which is ŷ
  - Also we will use the cross-entropy loss function.
  - This model is called skip-grams model.
- The last model has a problem with the softmax layer:

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

  - Here we are summing 10,000 numbers which corresponds to the number of words in our vocabulary.
  - If this number is larger say 1 million, the computation will become very slow.

- One of the solutions for the last problem is to use "**Hierarchical softmax classifier**" which works as a tree classifier.


- In practice, the hierarchical softmax classifier doesn't use a balanced tree like the drawn one. Common words are at the top and less common are at the bottom.
- How to sample the context **c**?
  - One way is to choose the context by random from your corpus.
  - If you have done it that way, there will be frequent words like "the, of, a, and, to, .." that can dominate other words like "orange, apple, durian,..."
  - In practice, we don't take the context uniformly random, instead there are some heuristics to balance the common words and the non-common words.
- word2vec paper includes 2 ideas of learning word embeddings. One is skip-gram model and another is CBoW (continuous bag-of-words).

Negative Sampling

- Negative sampling allows you to do something similar to the skip-gram model, but with a much more efficient learning algorithm. We will create a different learning problem.
- Given this example:
  - "I want a glass of orange juice to go along with my cereal"
- The sampling will look like this:
-

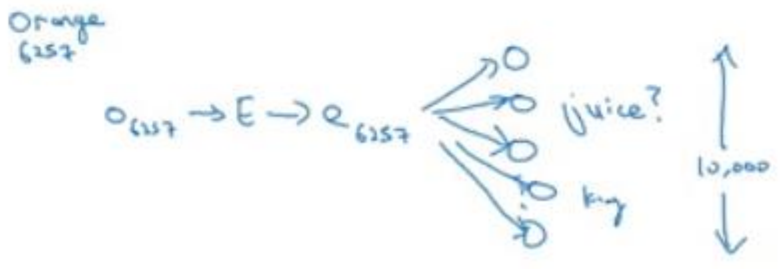
Context	Word	
orange	juice	1

Context	Word	
orange	king	0
orange	book	0
orange	the	0
orange	of	0

- We get positive example by using the same skip-grams technique, with a fixed window that goes around.
- To generate a negative example, we pick a word randomly from the vocabulary.
- Notice, that we got word "of" as a negative example although it appeared in the same sentence.
- So the steps to generate the samples are:
    i. Pick a positive context
    ii. Pick a k negative contexts from the dictionary.
- k is recommended to be from 5 to 20 in small datasets. For larger ones - 2 to 5.
- We will have a ratio of k negative examples to 1 positive ones in the data we are collecting.
- Now let's define the model that will learn this supervised learning problem:
    - Lets say that the context word are `c` and the word are `t` and `y` is the target.
    - We will apply the simple logistic regression model.

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

- The logistic regression model can be drawn like this:



- So we are like having 10,000 binary classification problems, and we only train k+1 classifier of them in each iteration.

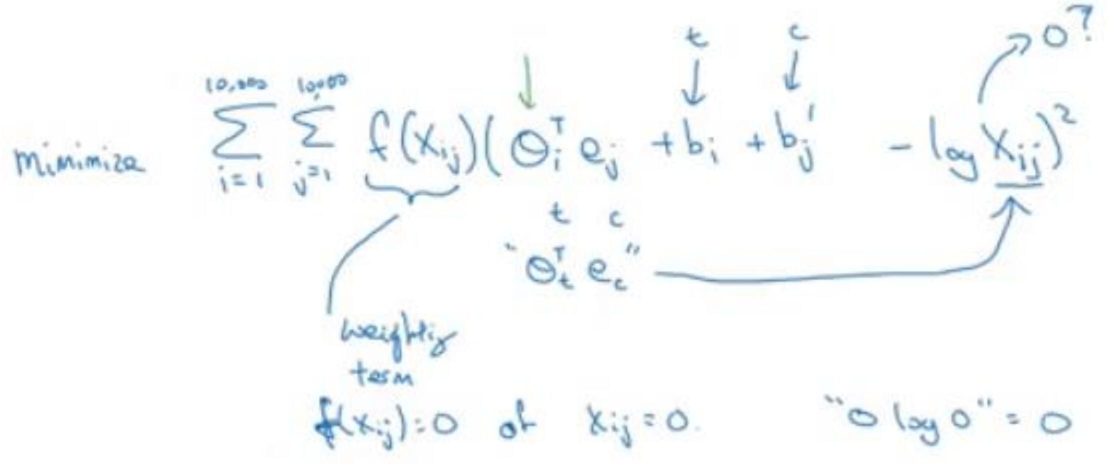
- How to select negative samples:
  - We can sample according to empirical frequencies in words corpus which means according to how often different words appears. But the problem with that is that we will have more frequent words like *the, of, and...*
  - The best is to sample with this equation (according to authors):

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

GloVe word vectors

- GloVe is another algorithm for learning the word embedding. It's the simplest of them.
- This is not used as much as word2vec or skip-gram models, but it has some enthusiasts because of its simplicity.
- GloVe stands for Global vectors for word representation.
- Let's use our previous example: "I want a glass of orange juice to go along with my cereal".
- We will choose a context and a target from the choices we have mentioned in the previous sections.
- Then we will calculate this for every pair: X_{ct} = # times t appears in context of c

- $X_{ct} = X_{tc}$ if we choose a window pair, but they will not equal if we choose the previous words for example. In GloVe they use a window which means they are equal
- The model is defined like this:

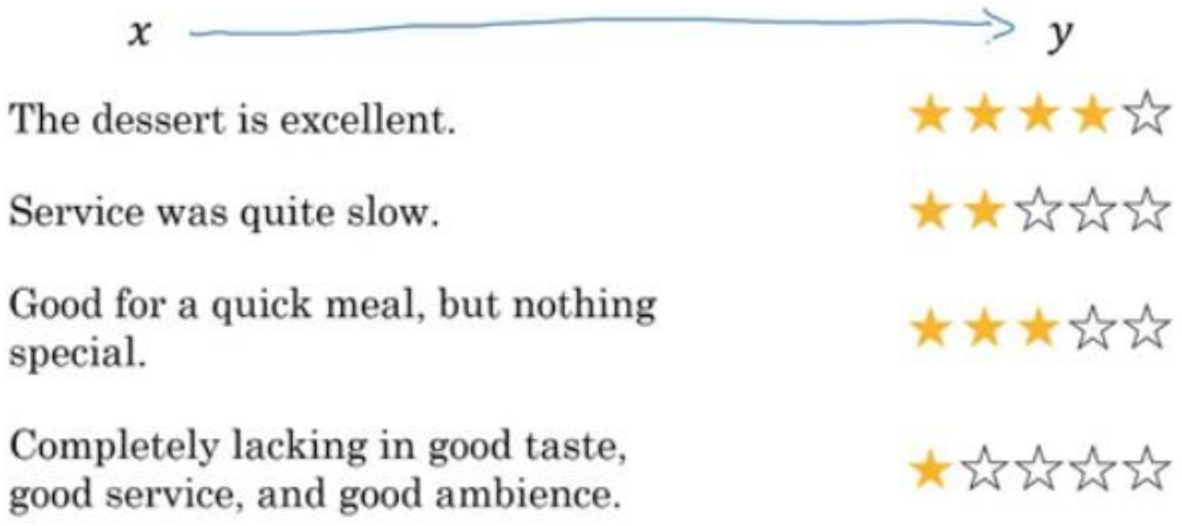

minimize
$\sum_{i=1}^{10,000} \sum_{j=1}^{10,000} f(X_{ij})(\theta_i^T e_j + b_i + b_j' - \log X_{ij})^2$
t
c
→0?
"$\theta_t^T e_c$"
weighting term
$f(X_{ij}) = 0$ if $X_{ij} = 0$.
"0 log 0" = 0

- f(x) - the weighting term, used for many reasons which include:
  - The `log(0)` problem, which 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.
- **Theta** and **e** are symmetric which helps getting the final word embedding.
- *Conclusions on word embeddings:*
  - If this is your first try, you should try to download a pre-trained model that has been made and actually works best.
  - If you have enough data, you can try to implement one of the available algorithms.
  - Because word embeddings are very computationally expensive to train, most ML practitioners will load a pre-trained set of embeddings.
  - A final note that you 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.

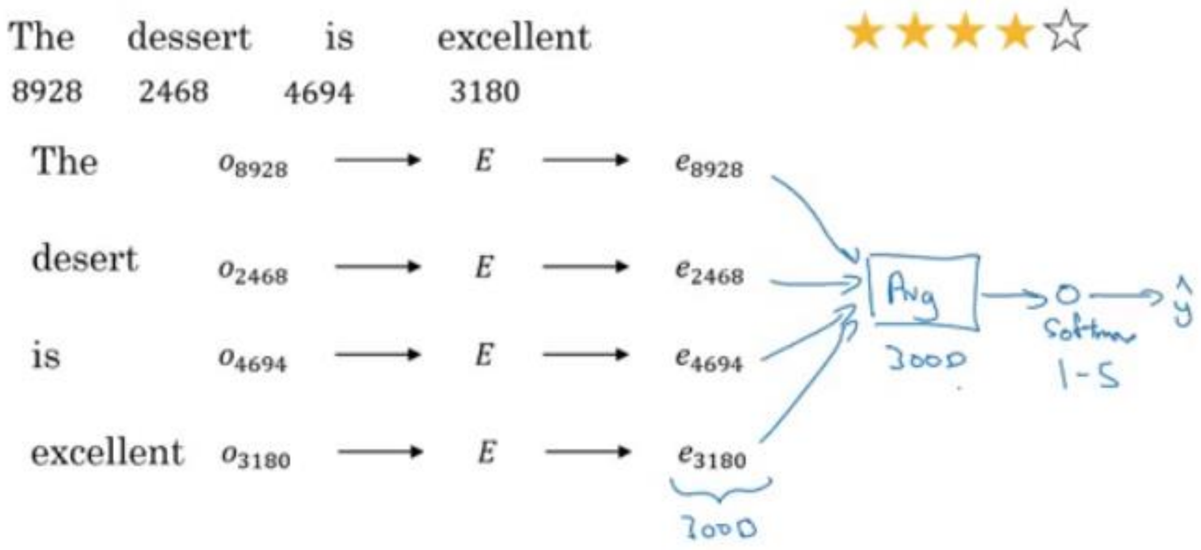
Applications using Word Embeddings

Sentiment Classification

- As we have discussed before, Sentiment classification is the process of finding if a text has a positive or a negative review. Its so useful in NLP and is used in so many applications. An example would be:

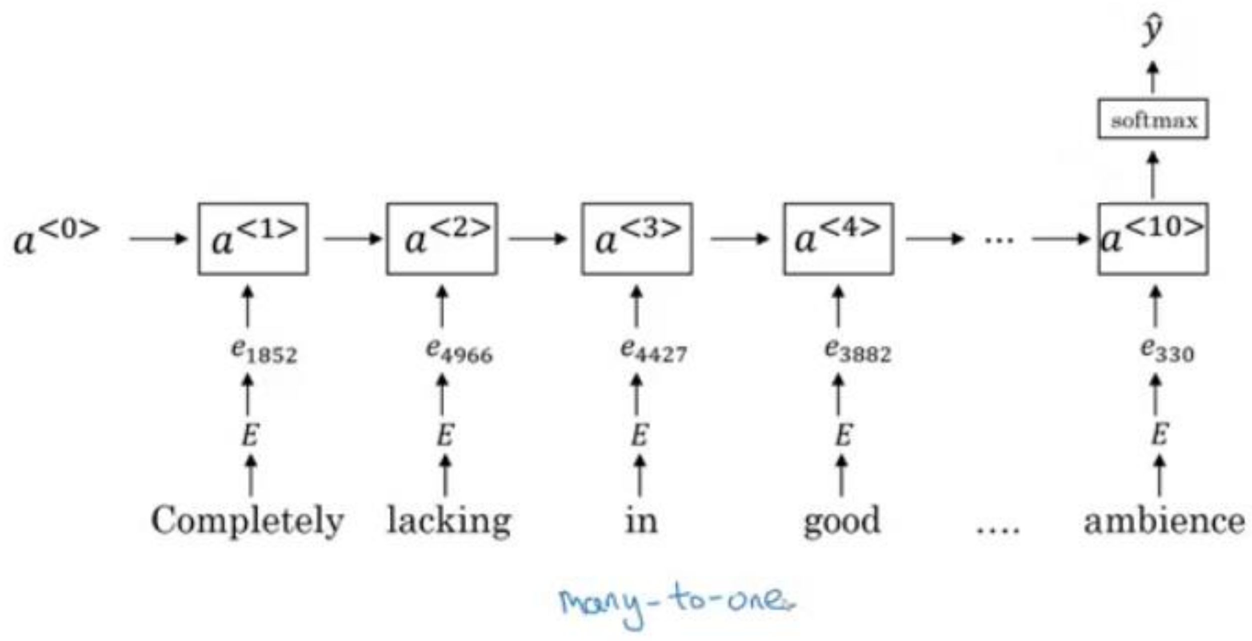

x
y
The dessert is excellent.
Service was quite slow.
Good for a quick meal, but nothing special.
Completely lacking in good taste, good service, and good ambience.

- One of the challenges with it, is that you might not have a huge labeled training data for it, but using word embeddings can help getting rid of this.
- The common dataset sizes varies from 10,000 to 100,000 words.
- A simple sentiment classification model would be like this:


The dessert is excellent
8928 2468 4694 3180
The o_{8928} E e_{8928}
desert o_{2468} E e_{2468}
is o_{4694} E e_{4694}
excellent o_{3180} E e_{3180}
300
Avg
300
softmax
1-5
$\hat{y}$

  - The embedding matrix may have been trained on say 100 billion words.
  - Number of features in word embedding is 300.
  - We can use **sum** or **average** given all the words then pass it to a softmax classifier. That makes this classifier works for short or long sentences.

- One of the problems with this simple model is that it ignores words order. For example "Completely lacking in **good**taste, **good** service, and **good** ambience" has the word *good* 3 times but its a negative review.
- A better model uses an RNN for solving this problem:

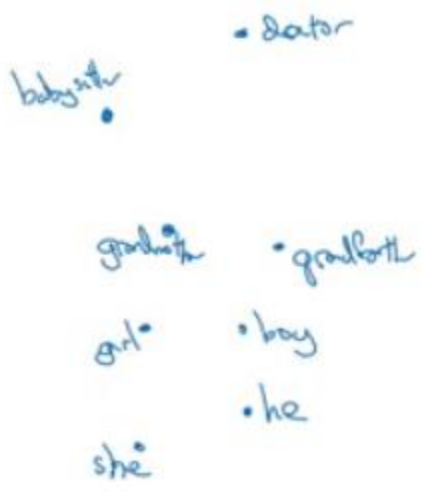


  - And so if you train this algorithm, you end up with a pretty decent sentiment classification algorithm.
  - Also, it will generalize better even if words weren't in your dataset. For example you have the sentence "Completely **absent** of good taste, good service, and good ambience", then even if the word "absent" is not in your label training set, if it was in your 1 billion or 100 billion word corpus used to train the word embeddings, it might still get this right and generalize much better even to words that were in the training set used to train the word embeddings but not necessarily in the label training set that you had for specifically the sentiment classification problem.

Debiasing word embeddings

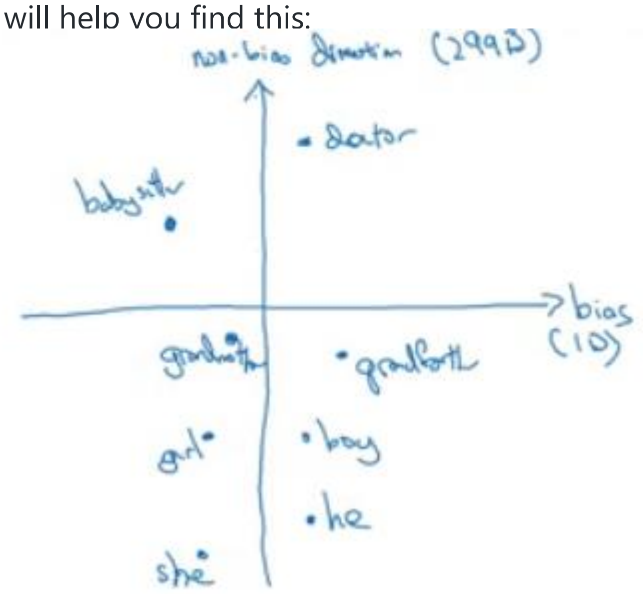
- We want to make sure that our word embeddings are free from undesirable forms of bias, such as gender bias, ethnicity bias and so on.
- Horrifying results on the trained word embeddings in the context of Analogies:
  - Man : Computer_programmer as Woman : **Homemaker**
  - Father : Doctor as Mother : **Nurse**
- Word embeddings can reflect gender, ethnicity, age, sexual orientation, and other biases of text used to train the model.

- Learning algorithms by general are making important decisions and it mustn't be biased.
- Andrew thinks we actually have better ideas for quickly reducing the bias in AI than for quickly reducing the bias in the human race, although it still needs a lot of work to be done.
- Addressing bias in word embeddings steps:
  - Idea from the paper: https://arxiv.org/abs/1607.06520
  - Given these learned embeddings:



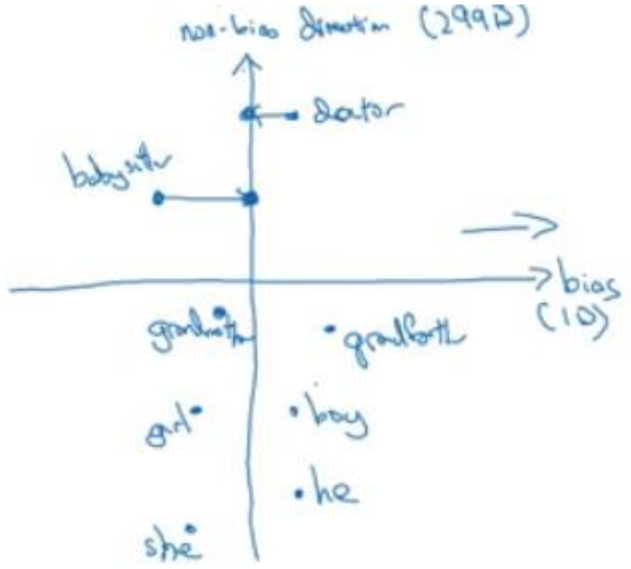
  - We need to solve the **gender bias** here. The steps we will discuss can help solve any bias problem but we are focusing here on gender bias.
  - Here are the steps:
    a. Identify the direction:
      - Calculate the difference between:
        - e_{he} - e_{she}
        - e_{male} - e_{female}
        - ....
      - Choose some k differences and average them.

- This will help you find this:



- By that we have found the bias direction which is 1D vector and the non-bias vector which is 299D vector.

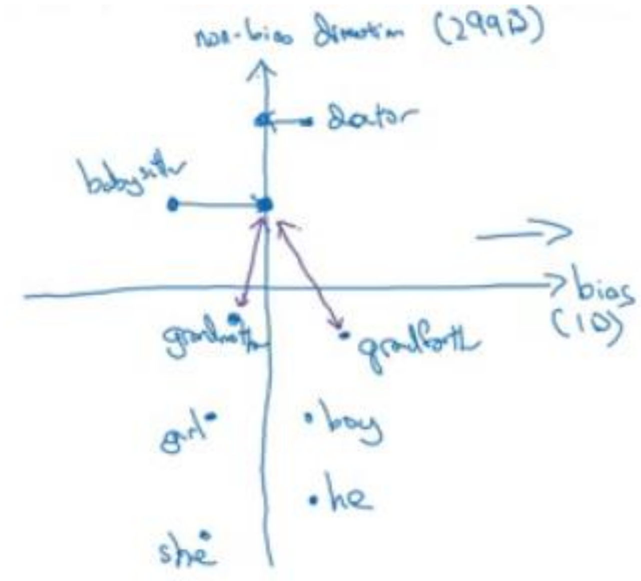
b. Neutralize: For every word that is not definitional, project to get rid of bias.
    - Babysitter and doctor need to be neutral so we project them on non-bias axis with the direction of the bias:



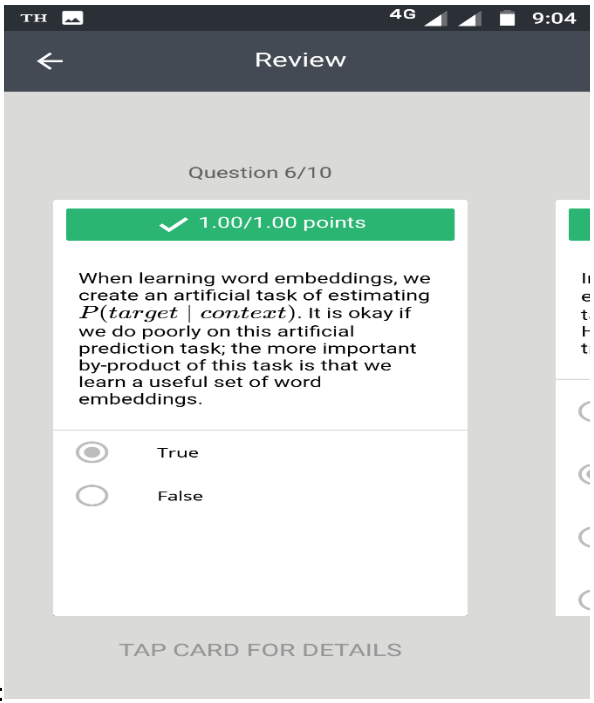
        - After that they will be equal in the term of gender. - To do this the authors of the paper trained a classifier to tell the words that need to be neutralized or not.

c. Equalize pairs

- We want each pair to have difference only in gender. Like:
    - Grandfather - Grandmother - He - She - Boy - Girl
- We want to do this because the distance between grandfather and babysitter is bigger than babysitter and grandmother:



- To do that, we move grandfather and grandmother to a point where they will be in the middle of the non-bias axis.
- There are some words you need to do this for in your steps. Number of these words is relatively small.


TH 4G 9:04

Review

Question 6/10

✓ 1.00/1.00 points

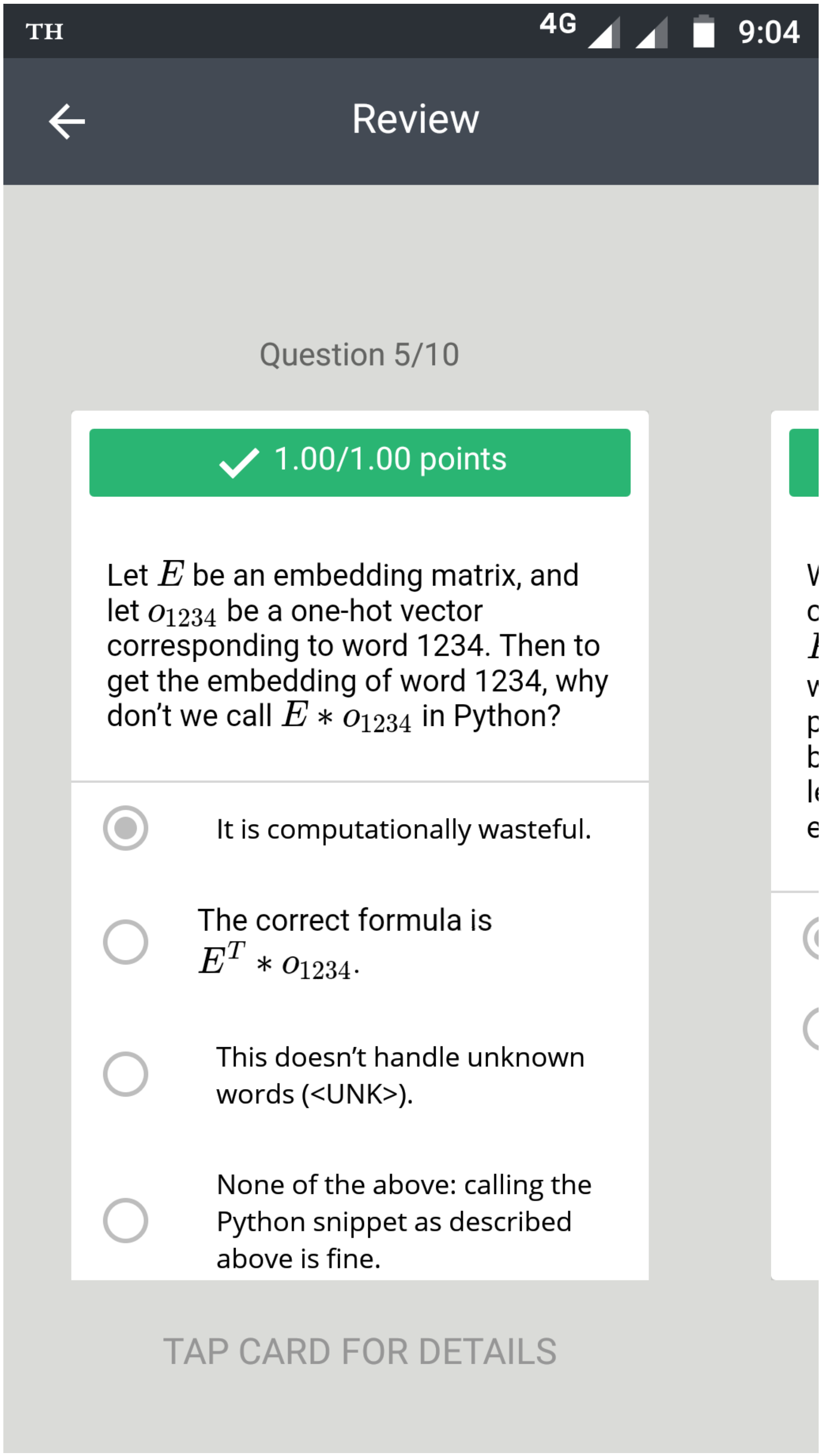
When learning word embeddings, we create an artificial task of estimating $P(target \mid context)$. It is okay if we do poorly on this artificial prediction task; the more important by-product of this task is that we learn a useful set of word embeddings.

[✓ True]

False

TAP CARD FOR DETAILS

Quiz Notes:



TH 4G 9:04

Review

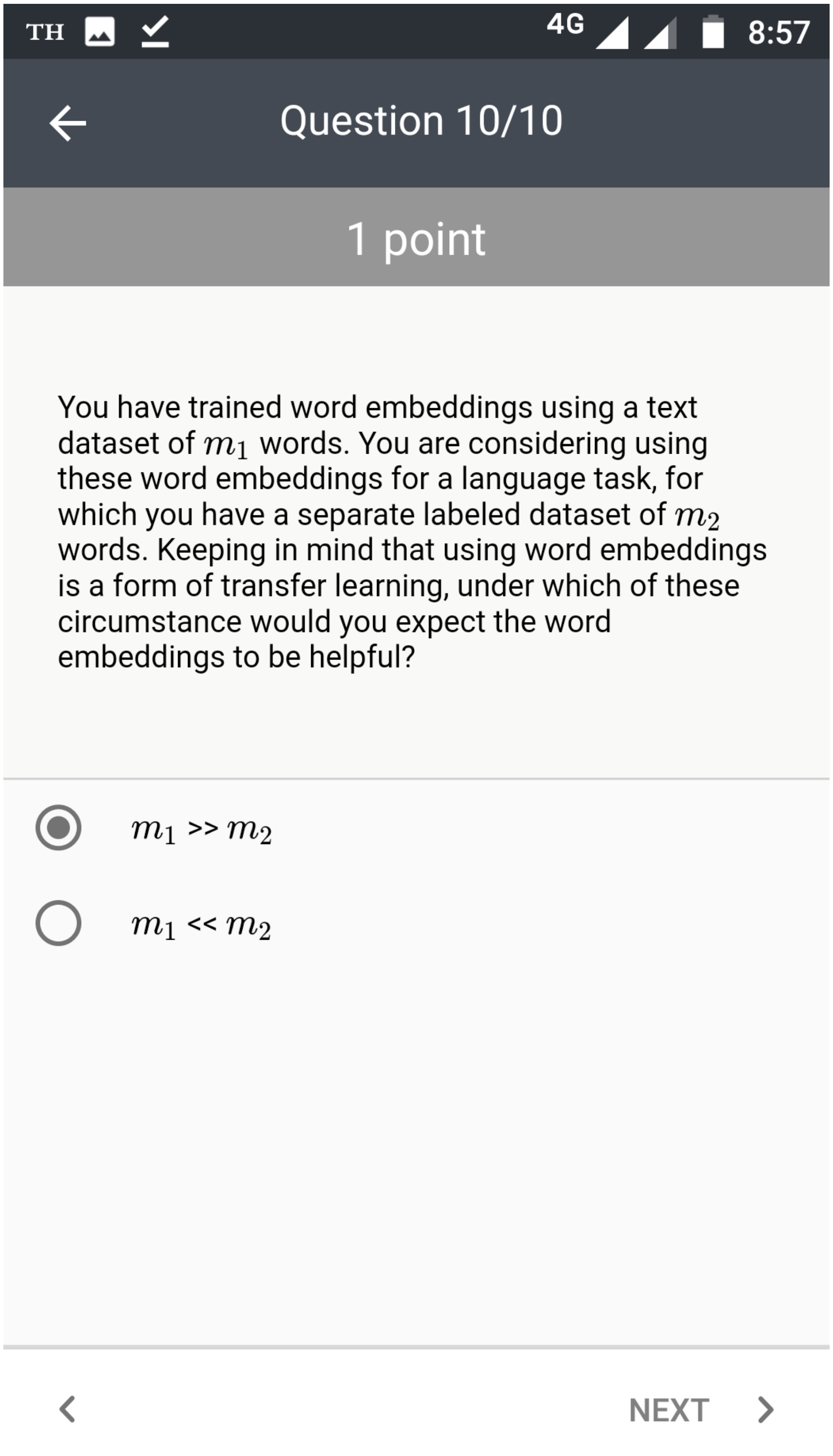
Question 5/10

✓ 1.00/1.00 points

Let E 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 (<UNK>).
- None of the above: calling the Python snippet as described above is fine.

TAP CARD FOR DETAILS



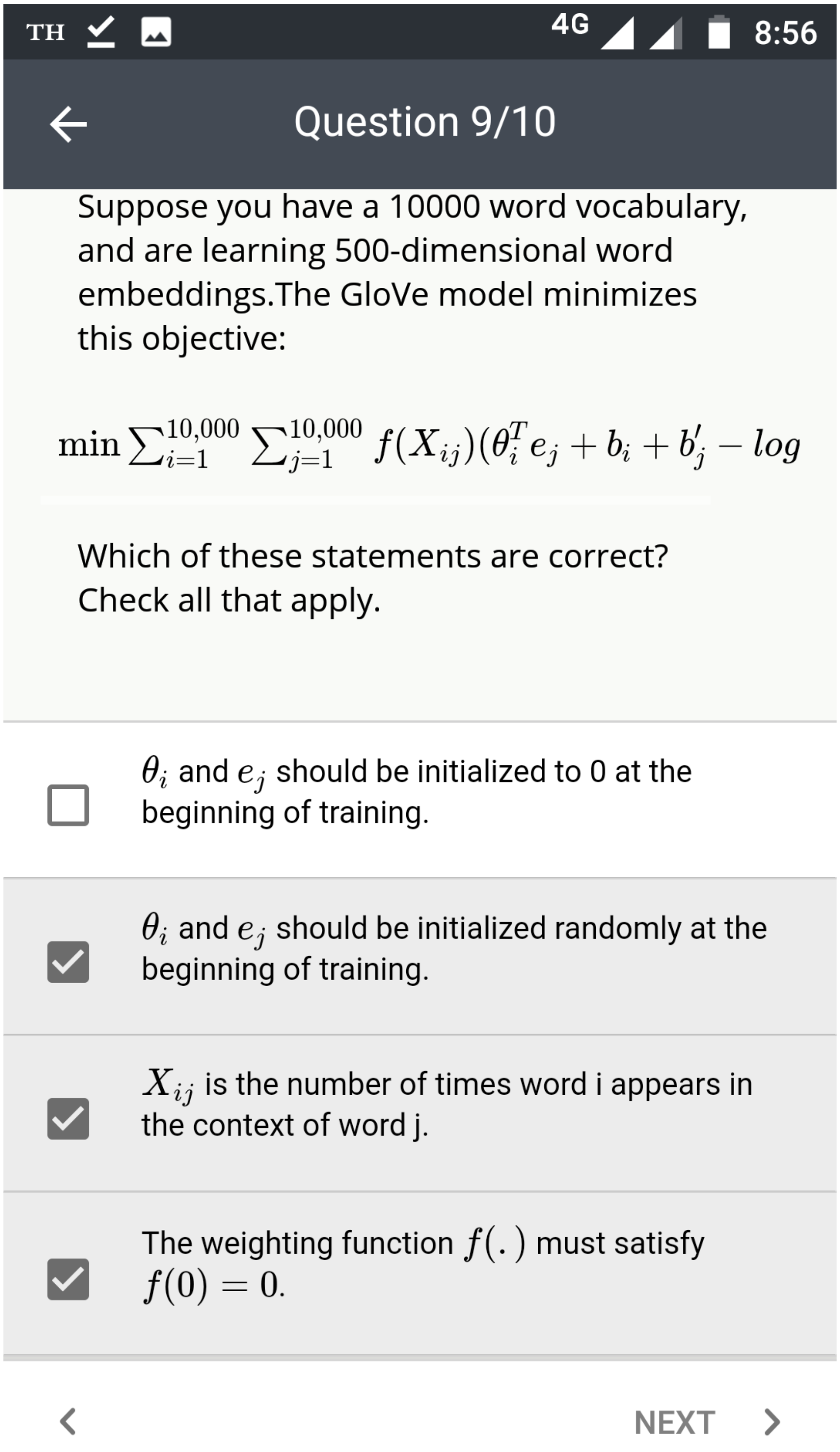
Question 10/10

1 point

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?

[✓ $m_1 >> m_2$]

$m_1 << m_2$

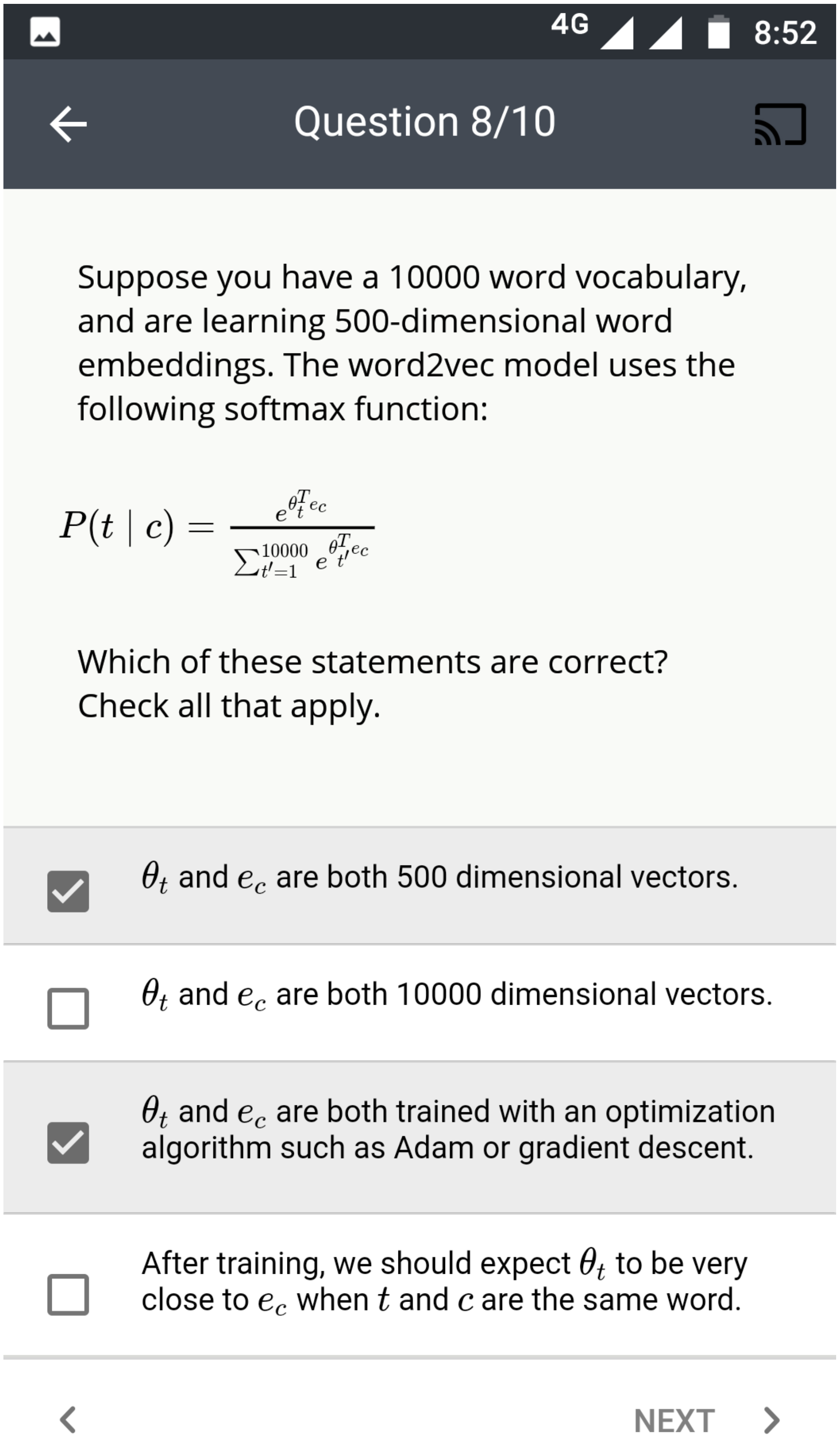


Suppose you have a 10000 word vocabulary, and are learning 500-dimensional word embeddings.The GloVe model minimizes this objective:

$$\min \sum_{i=1}^{10,000} \sum_{j=1}^{10,000} f(X_{ij})(\theta_i^T e_j + b_i + b_j' - log$$

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 $f(0) = 0$.



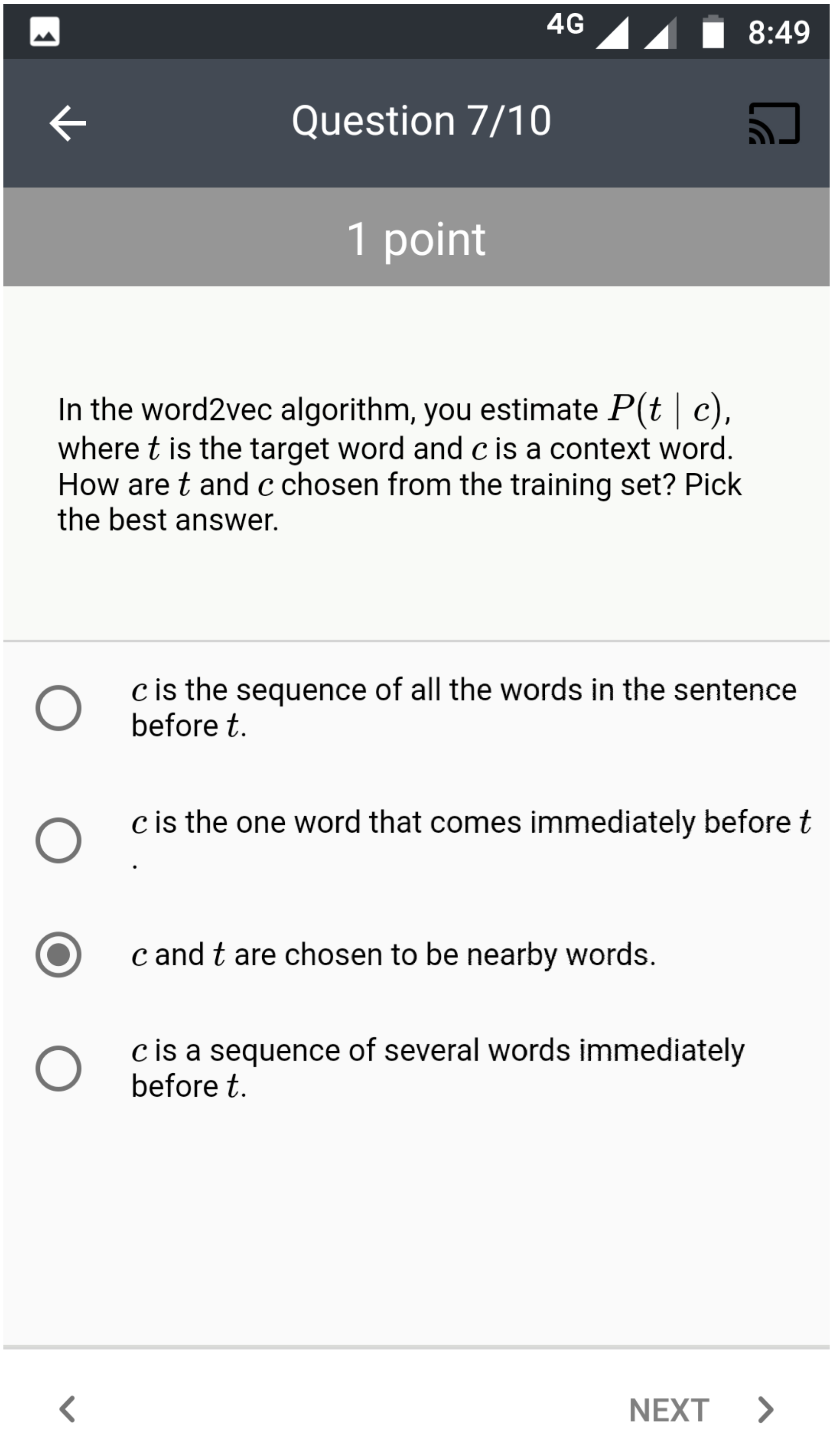
Question 8/10

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

$$P(t \mid c) = \frac{e^{\theta_t^T e_c}}{\sum_{t'=1}^{10000} e^{\theta_{t'}^T e_c}}$$

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.

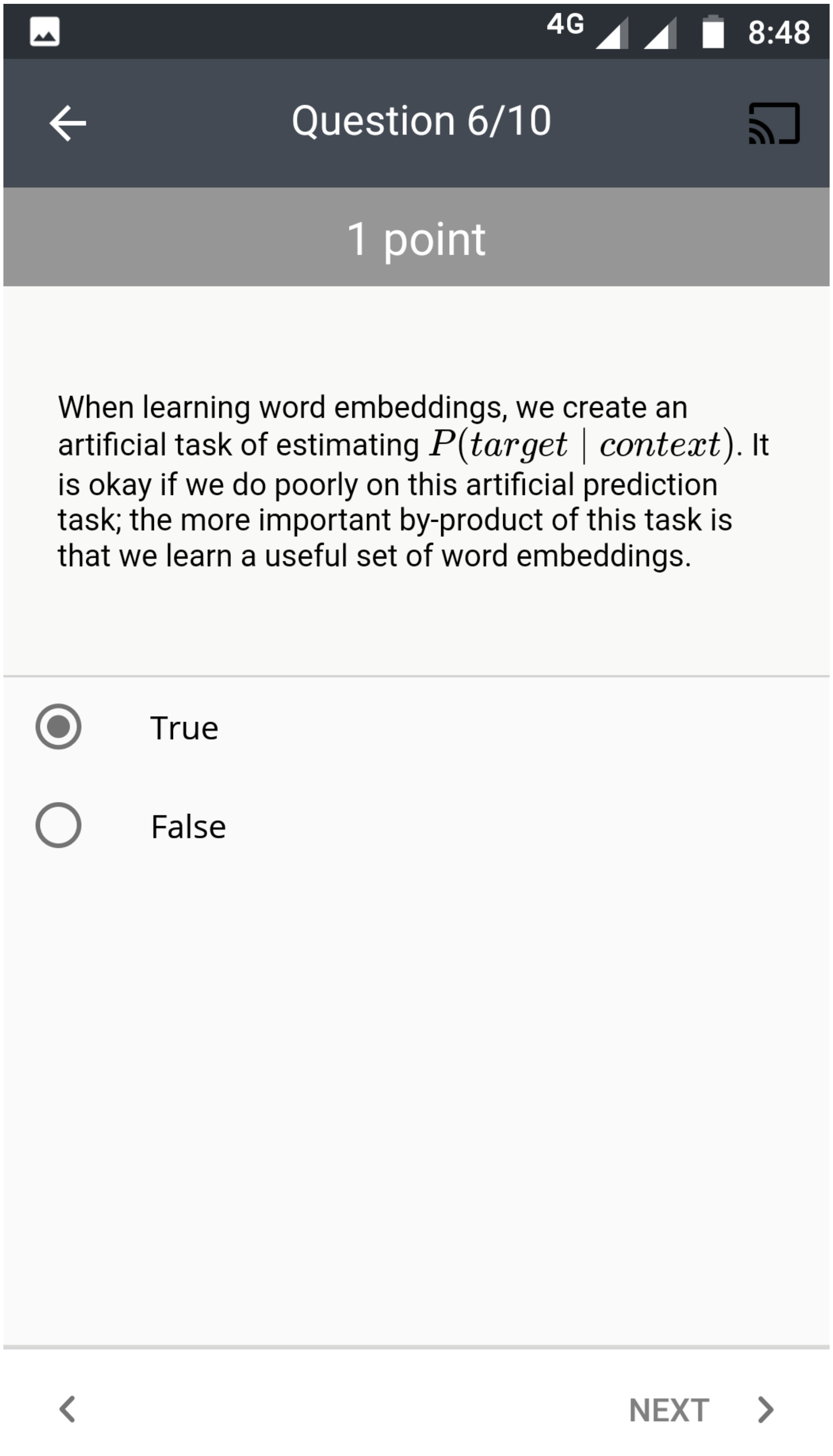


Question 7/10

1 point

In the word2vec algorithm, you estimate $P(t \mid c)$, 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 is the one word that comes immediately before t.
- [✓ c and t are chosen to be nearby words.]
- ○ c is a sequence of several words immediately before t.


4G 8:48

Question 6/10

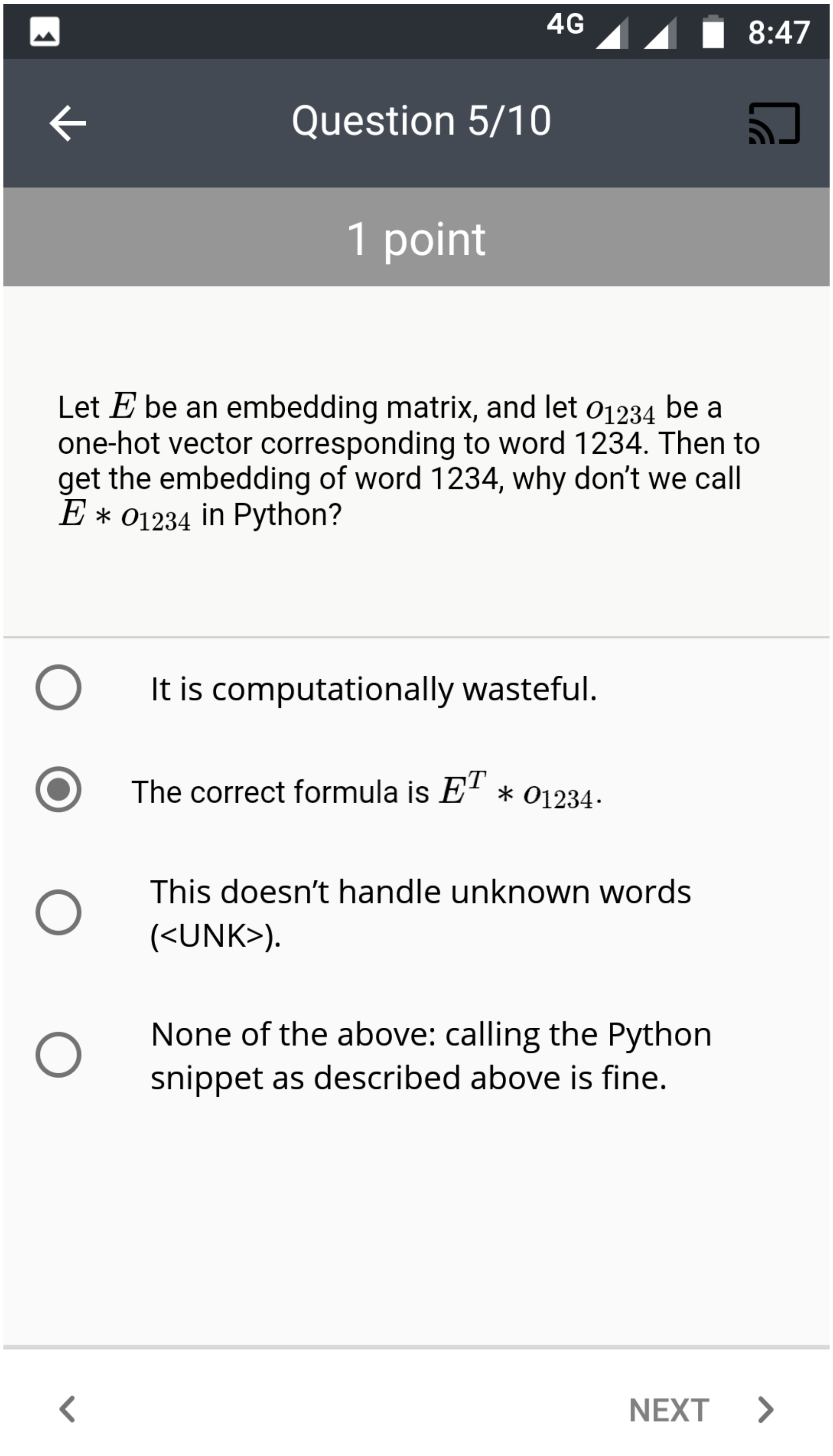
1 point

When learning word embeddings, we create an artificial task of estimating $P(target \mid context)$. 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

NEXT

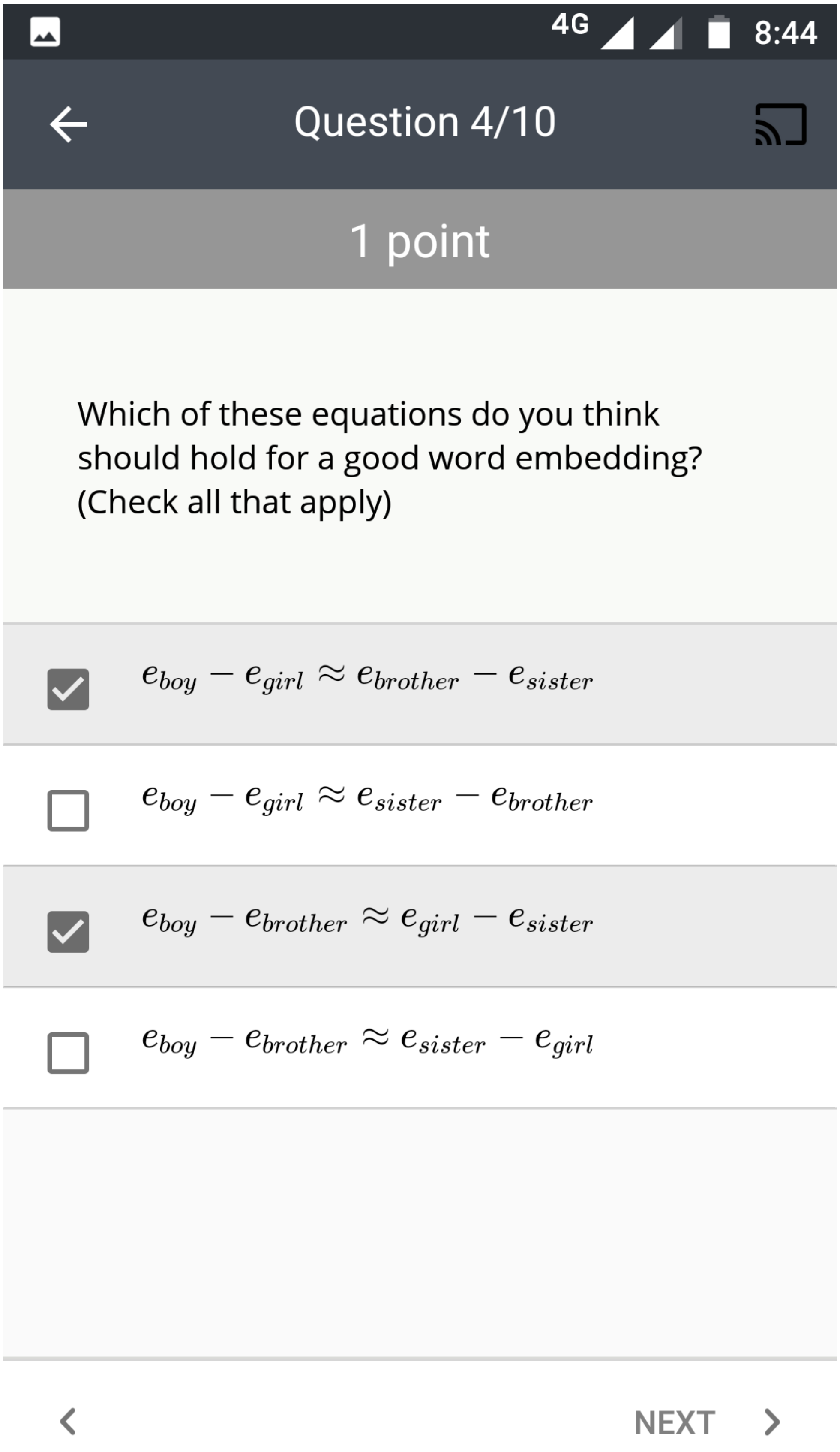


Question 5/10

1 point

Let E 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 (<UNK>).
- ○ None of the above: calling the Python snippet as described above is fine.


4G 8:44

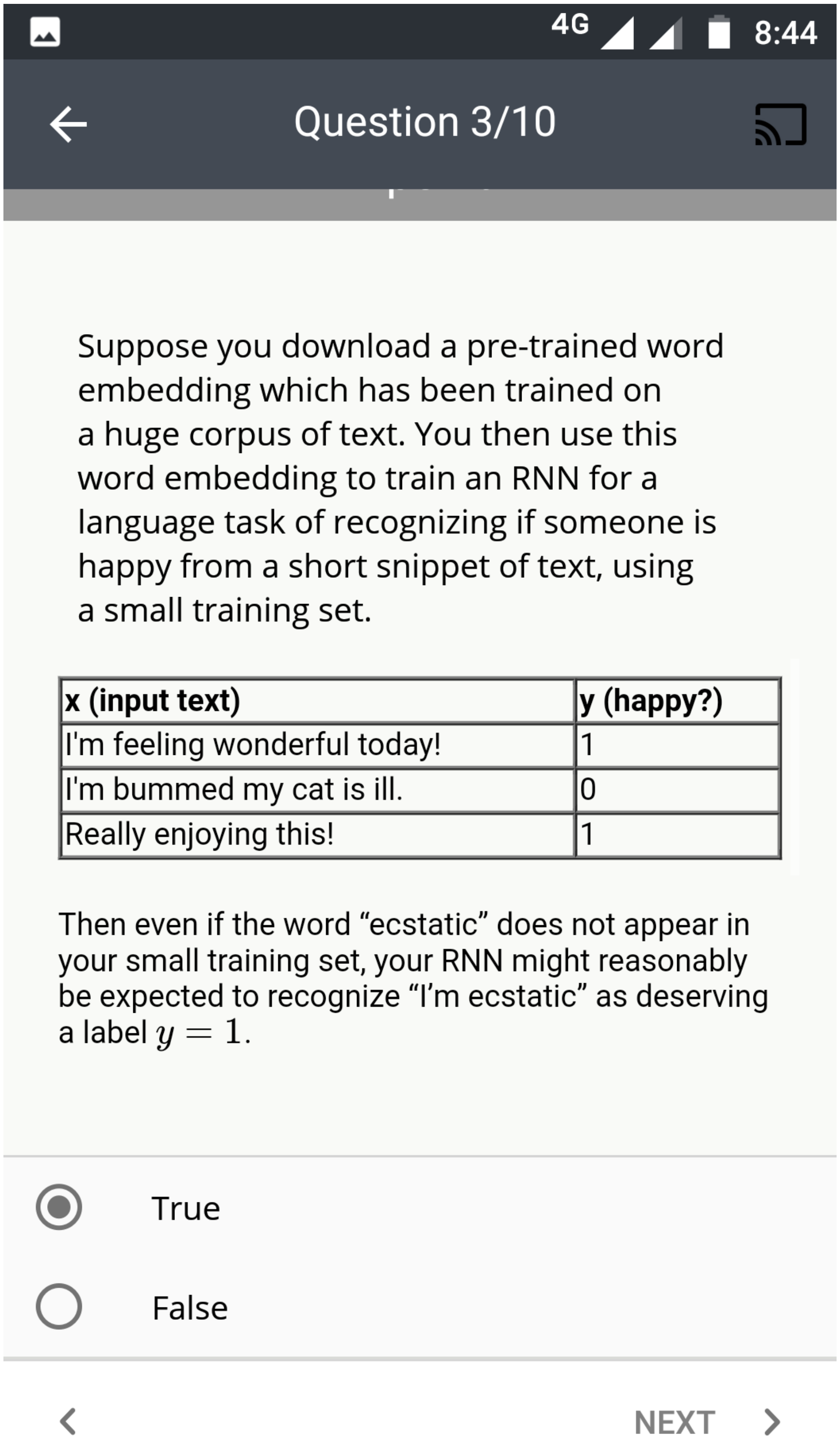
Question 4/10

1 point

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

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

NEXT



Question 3/10

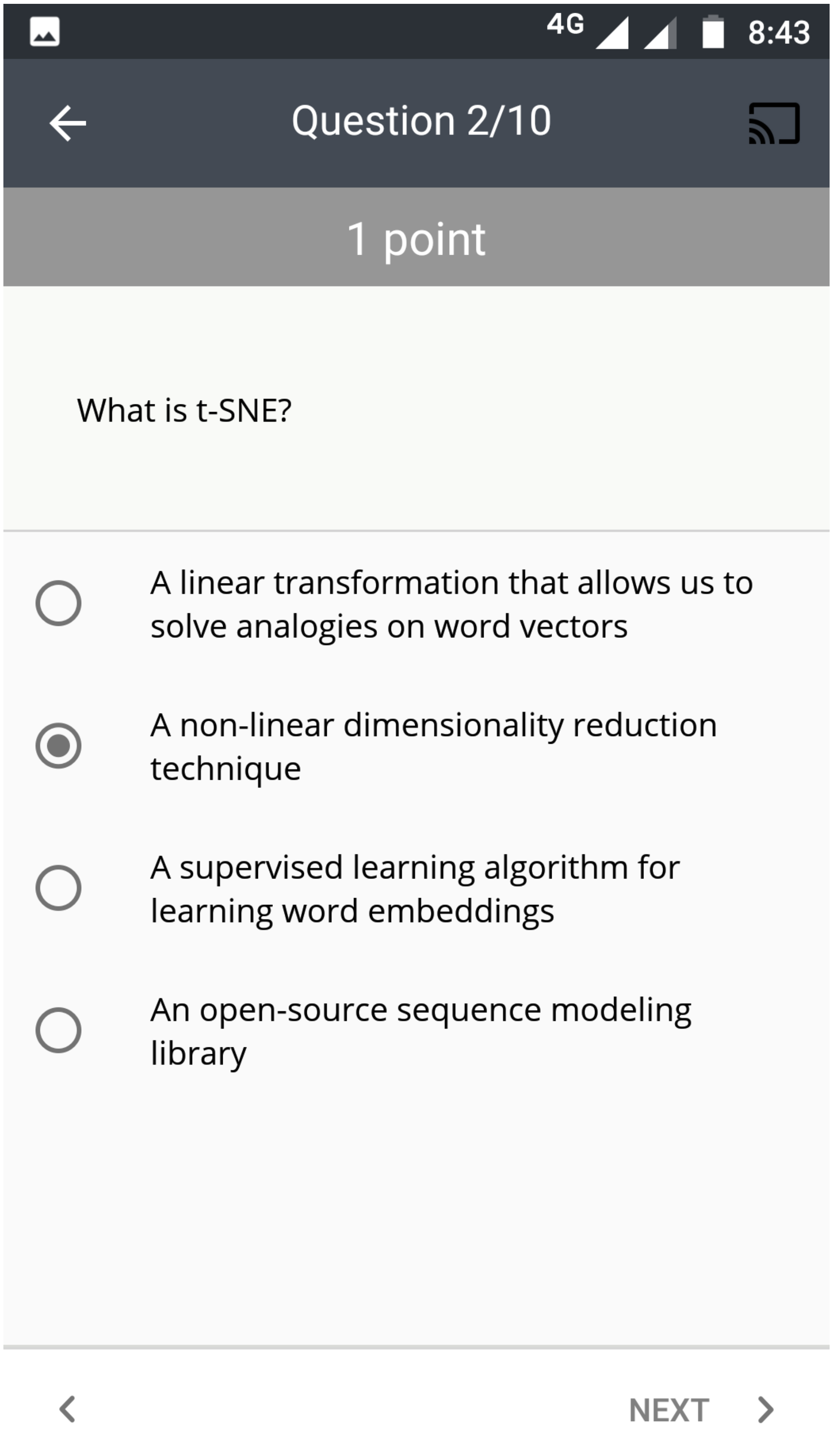
Suppose you download a pre-trained word embedding which has been trained on a huge corpus of text. You then use this word embedding to train an RNN for a language task of recognizing if someone is happy from a short snippet of text, using a small training set.

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


4G 8:43

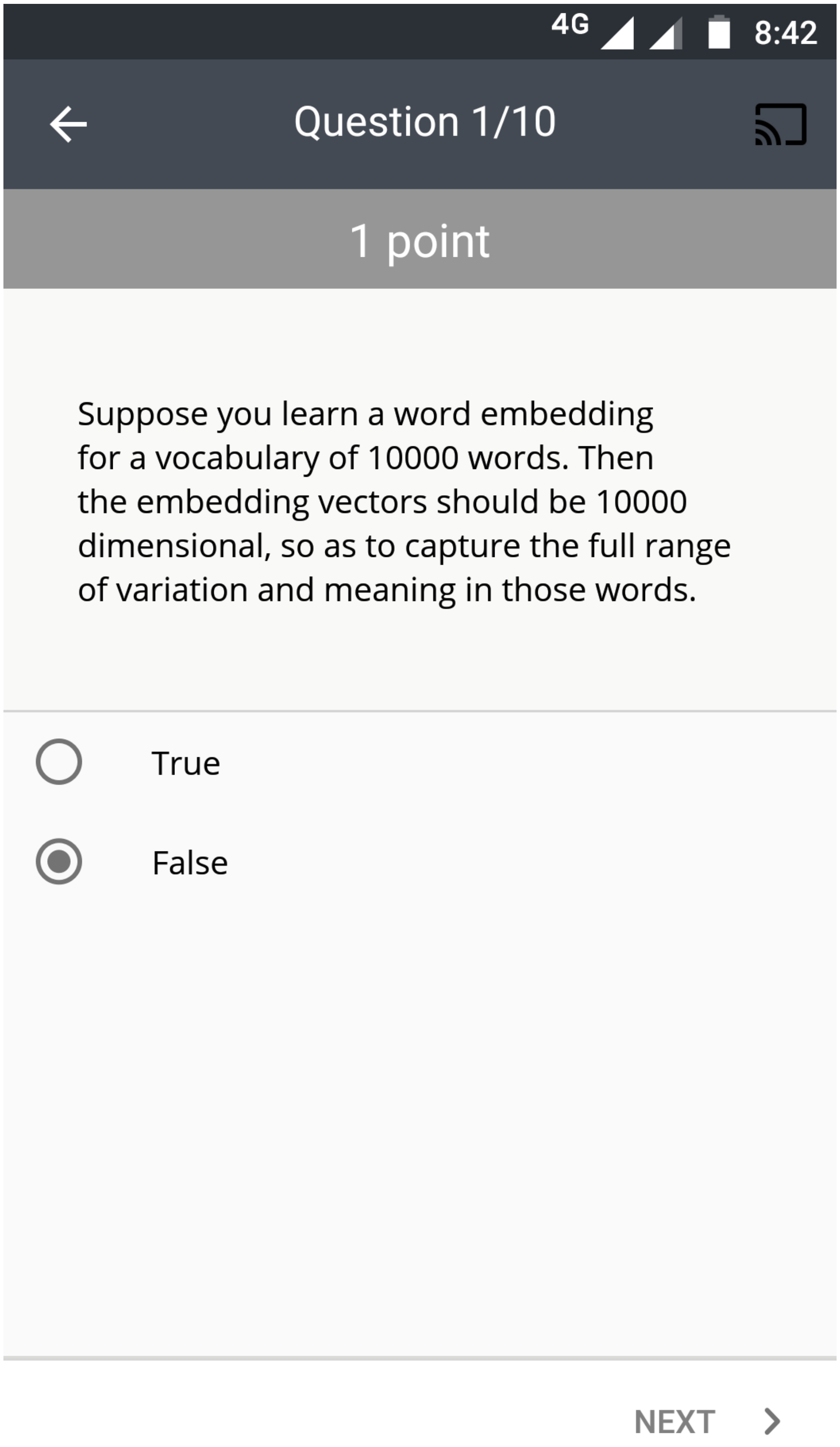
Question 2/10

1 point

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

NEXT


4G 8:42

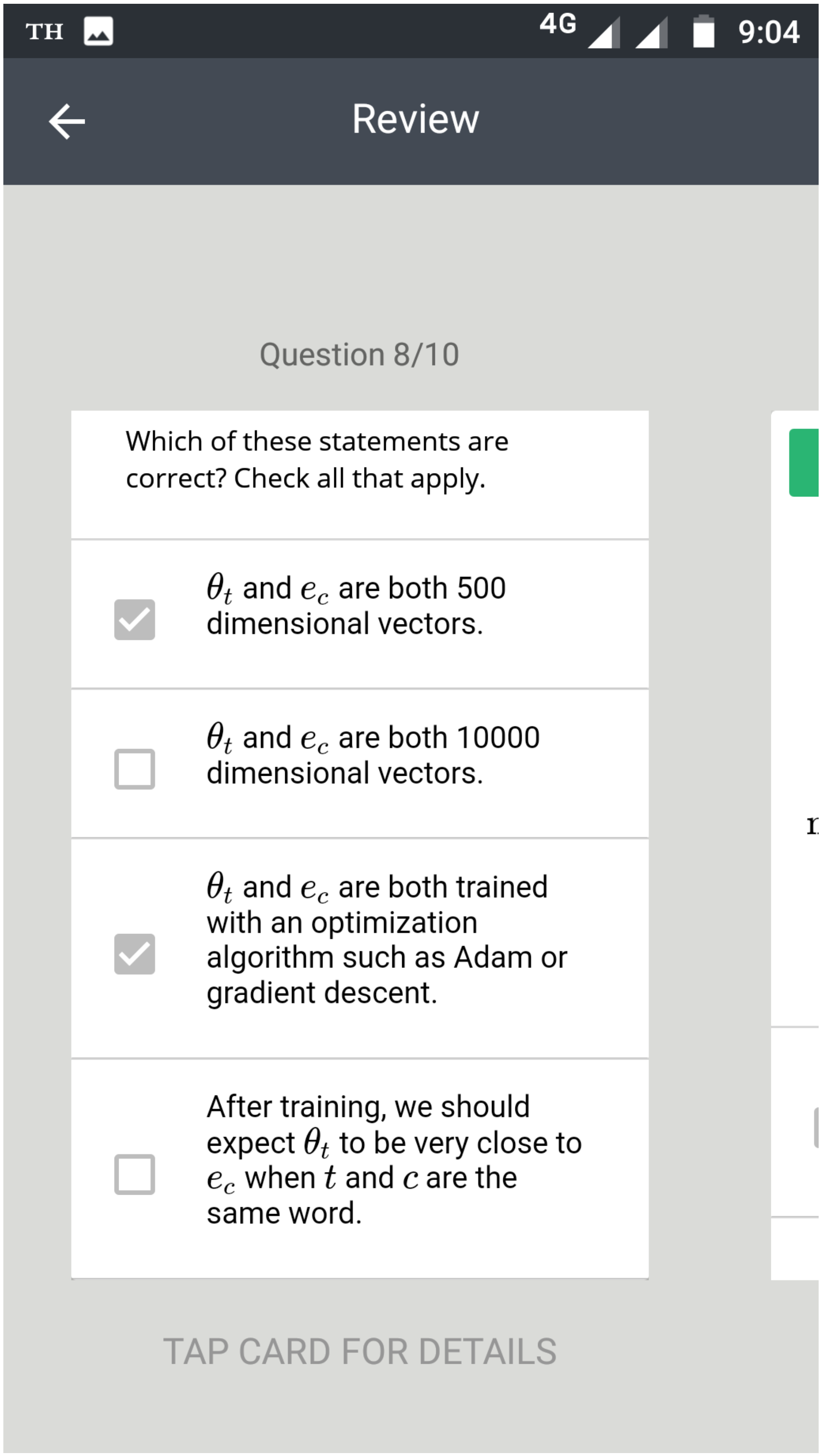
Question 1/10

1 point

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]

NEXT



TH 4G 9:04

Review

Question 8/10

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.

TAP CARD FOR DETAILS

Assignment Notes:

1 - Cosine similarity

To measure how similar two words are, we 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:

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

where $u.v$ is the dot product (or inner product) of two vectors, $||u||_2$ is the norm (or length) of the vector u, and θ is the angle between u and v. This 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.

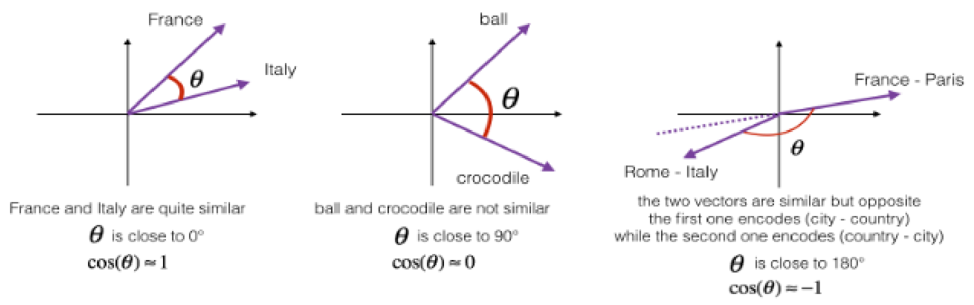

France
Italy
θ
France and Italy are quite similar
θ is close to 0°
$\cos(\theta) \approx 1$
ball
crocodile
θ
ball and crocodile are not similar
θ is close to 90°
$\cos(\theta) \approx 0$
France - Paris
Rome - Italy
θ
the two vectors are similar but opposite
the first one encodes (city - country)
while the second one encodes (country - city)
θ is close to 180°
$\cos(\theta) \approx -1$

Figure 1: The cosine of the angle between two vectors is a measure of how similar they are

Exercise: Implement the function `cosine_similarity()` to evaluate similarity between word vectors.

Reminder: The norm of u is defined as $||u||_2 = \sqrt{\sum_{i=1}^{n} u_i^2}$

Expected Output:

cosine_similarity(father, mother) =	0.890903844289
cosine_similarity(ball, crocodile) =	0.274392462614
cosine_similarity(france - paris, rome - italy) =	-0.675147930817

2. De-biasing Word Vectors:

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.

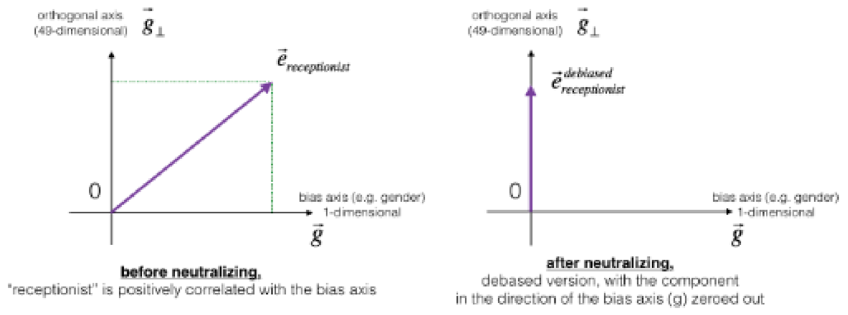

orthogonal axis (49-dimensional) $\vec{g}_{\perp}$
$\vec{e}_{receptionist}$
0
bias axis (e.g. gender) 1-dimensional
$\vec{g}$
before neutralizing,
"receptionist" is positively correlated with the bias axis
orthogonal axis (49-dimensional) $\vec{g}_{\perp}$
$\vec{e}_{receptionist}^{debiased}$
0
bias axis (e.g. gender) 1-dimensional
$\vec{g}$
after neutralizing,
debased version, with the component in the direction of the bias axis (g) zeroed out

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

Exercise: Implement `neutralize()` to remove the bias of words such as "receptionist" or "scientist". Given an input embedding e, you can use the following formulas to compute $e^{debiased}$:

$$e^{bias\_component} = \frac{e \cdot g}{||g||_2^2} * g \tag{2}$$

$$e^{debiased} = e - e^{bias\_component} \tag{3}$$

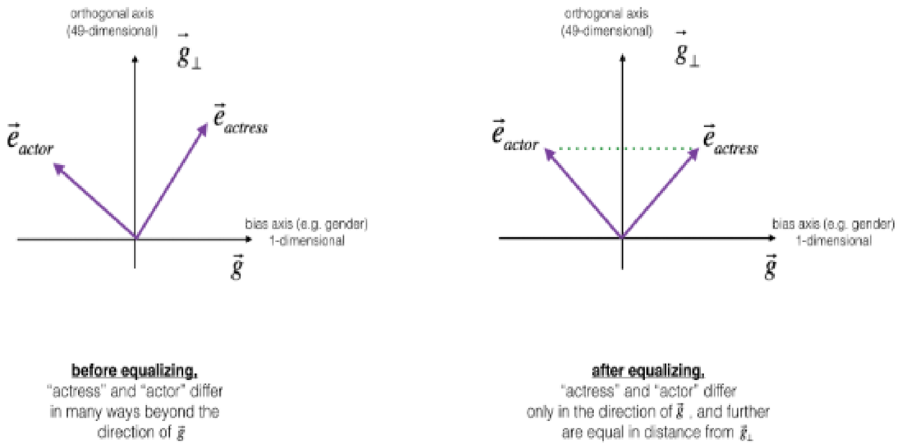
If you are an expert in linear algebra, you may recognize $e^{bias\_component}$ as the projection of e onto the direction g. If you're not an expert in linear algebra, don't worry about this.

Screenshot_20190225-085612.png - Photos

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:


orthogonal axis (49-dimensional) $\vec{g}_{\perp}$
$\vec{e}_{actor}$
$\vec{e}_{actress}$
bias axis (e.g. gender) 1-dimensional
$\vec{g}$
before equalizing,
"actress" and "actor" differ in many ways beyond the direction of $\vec{g}$
orthogonal axis (49-dimensional) $\vec{g}_{\perp}$
$\vec{e}_{actor}$
$\vec{e}_{actress}$
bias axis (e.g. gender) 1-dimensional
$\vec{g}$
after equalizing,
"actress" and "actor" differ only in the direction of $\vec{g}$, and further are equal in distance from $\vec{g}_{\perp}$

Emojify Model:

1.2 - Overview of the Emojifier-V1

In this part, you are going to implement a baseline model called "Emojifier-v1".

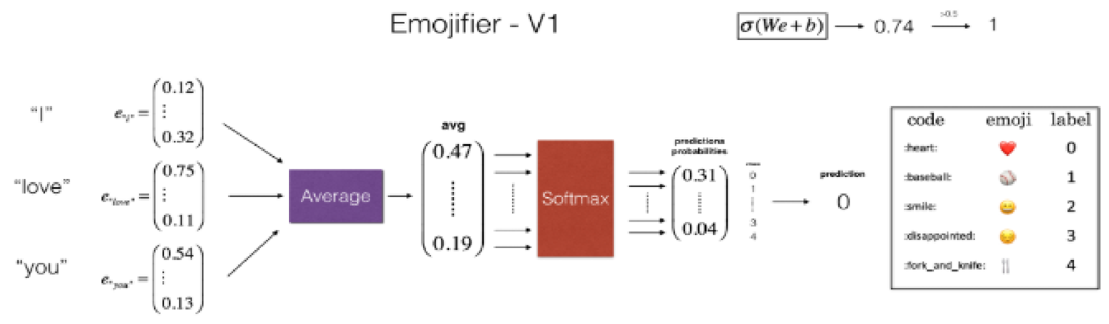
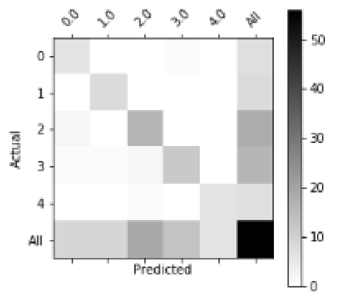


Figure 2: Baseline model (Emojifier-V1).

The input of the model is a string corresponding to a sentence (e.g. "I love you). In the code, the output will be a probability vector of shape (1,5), that you then pass in an argmax layer to extract the index of the most likely emoji output.

Accuracy Measurement:

```
(56,)
            ❤️   ⚾   😄   😞   🍴
Predicted  0.0  1.0  2.0  3.0  4.0  All
Actual
0            6    0    0    1    0    7
1            0    8    0    0    0    8
2            2    0   16    0    0   18
3            1    1    2   12    0   16
4            0    0    1    0    6    7
All          9    9   19   13    6   56
```



What you should remember from this part:

- Even with a 127 training examples, you can get a reasonably good model for Emojifying. This is due to the generalization power word vectors gives you.
- Emojify-V1 will perform poorly on sentences such as *"This movie is not good and not enjoyable"* because it doesn't understand combinations of words--it just averages all the words' embedding vectors together, without paying attention to the ordering of words. You will build a better algorithm in the next part.

Emojify using LSTM based NN:

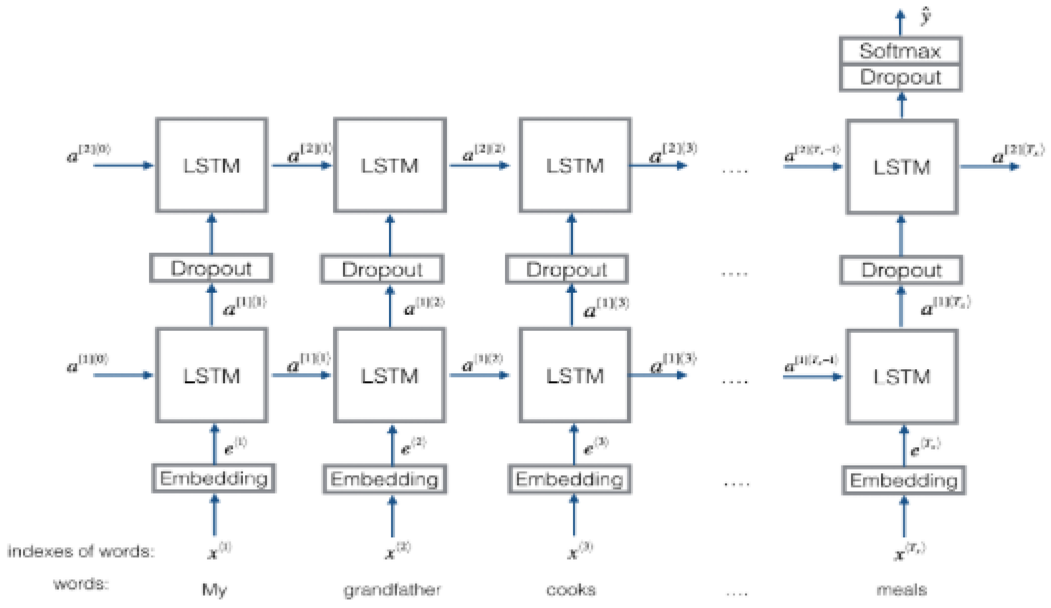


Figure 3: Emojifier-V2. A 2-layer LSTM sequence classifier.