ECE 1786 Lecture #2


Work-in-Flight: - Assignment 1 due next week.
- Fill out survey if you have not - new registrar.

Today: How Word Embeddings are Created.

Recall:
- Word embedding represent meaning of:
  - words in numbers.
  - helps neural networks deal with ambiguity in language.
  - embeddings that are close mean that the associated words are close in meaning.
  - also other associations Queen - King = Woman - Man
  - you should have done Assign 1 parts 1 & 2 now.

- last week we discussed (a little) how big dimension should be; diminishing returns beyond 300 according to GloVe paper by Pennington et al.
How to create (train) vectors?
- a clever + complex idea that begins with Bengio + Mikolov.

Big picture of method: We train a neural network to make a prediction based on the meaning of a word. Inside the neural network that meaning will be encoded. The training of the network will cause the encoding to be learned.

That encoding is the embedding/ vector.

(apologies for now using 3 terms that are the same thing: encoding/embedding/vector)
→ relates to auto-encoders

So, what is that prediction, and where does the data come from? Where do the labels come from?

The Prediction Task: Does word A "belong" with word B?
I.e., are they related somehow?

Let’s contemplate this by asking: Are they often used together?

Consider these three sentences:

1. The mathematician ran to the store.
2. The engineer ran to the store.
3. The mathematician solved the problem, which is

Sentences 1 & 2 imply similarity between Engineer & mathematician. Given that this is a sensible sentence, it’s not surprising that this is a sensible sentence.
---this is an example of the Distributional Hypothesis:

"Words appearing in similar contexts are related"

- since we want to predict which words are related, we have a ready made dataset of examples and labels: every sentence ever written! an enormous dataset! yippee!!

- let's consider some simple sentences taken from the SimpleCorpus.txt in Assignment L. In this course looks like this: I hold a dog. She holds a dog. He holds a dog. I rub the dog. She rubs the cat...

- these simple sentences give some clues to the meaning of the words used, and which are related -> what are they?

- we can generate training examples of words that are related by selecting words that are 'near' each other in 'correct' (i.e., written) sentences. near = within a few words (on either side) of target word

- for example, consider "I hold a dog"; if we take 'near' to be words ±2 words away from the target then the training examples of related words are

  - (I, hold) (hold, I) (a, I) (dog, hold)
  - (I, a) (hold, a) (a, hold) (dog, a)

- clearly, can make a lot of these.
- Want to build a NN predictor that given a word pair (wordA, wordB) can predict word B given word A:

```
input: word A  NN.  output: word B.
```

- For now, assume that words can only come from a specific limited vocabulary.

- A key part of this is how the input is represented. What is the simplest way to represent all the words in a vocabulary of size V? - Let's discuss 1-hot.

- We know that we want to represent the words by a vector of size ≤ vocabulary size (that's what the first lecture was about).

- Let's consider a simple example similar to Assignment 4, Part 3.

- Let's say the vocabulary size | V | = 10

- Let's assume the embedding size has dim = 4 (vector, encoding).

- So that implies we've got 10 embeddings of size 4 each.

- These are typically stored in a matrix like so

```
E = [v11 v12 v13 ... v14]
    [v21 v22 v23 ... v24]
    [v31 v32 v33 ... v34]
    [v41 v42 v43 ... v44]
```

- Key: these values will be randomly initialized and learned through training, like weights & biases.
so the input to the NN will be these 4 values \( (e_0, e_1, e_2, e_3) \) associated with word A in the embedding matrix.

- but what should the output be?

\[
\begin{align*}
\text{Prob (word0)} \\
\text{Prob (word1)} \\
\text{Prob (word2)} \\
\text{Prob (word3)}
\end{align*}
\]

- where \( i \) is the word number of word A (aka 'target' word)

here is the 10

- lets draw what the 1Lv neuron neural network looks like:

that will do the prediction task

- No non-linear fun.

\[
\text{Word A}
\]

\[
\text{softmax norm}
\]

recall linear neuron:

\[
u^{(t)} = \sum_{j=0}^{n} w_j e_j + b_0
\]

\( \text{is it really a probability?} \)
- So to train this network, you would present many examples of \((\text{word}_A, \text{word}_B)\) pairs.
  \- \text{word}_A \text{ is the } C^i_j \text{ target.} \\
  \- \text{word}_B \text{ is the label, is 1-hot encoded in a vector of size } |V| = 10. \\
  \- \text{Use "cross entropy" loss to train (the -log of correct answer.} \]

- Crucial, to repeat: the values of \(C^i_j\) are also learned through gradient descent. For numerical stability.

- Nuance/improvement: observe the \(W^i_j\) these are different word embeddings for each word.
  \- we don't really need 2 embeddings, so can save computation \((C^i_j \text{ parameters})\) by just using 1.
  \- i.e replace \(W^i_j\) with \(C^i_j\) (the same parameter appears in 2 places in the network).
  \- do this in assignment 1, not.

- A different way to think about this:
  \- to perform the prediction task, "do these two words relate?" just compute the dot product of their word vectors
  \- the higher it is, the more similar they are, just like a convolution kernel.
Assignment 1 Section 3 asks you to train a vocabulary of size 21 I, he, she on a small corpus with an embedding dimension of 2.

- If similar words end up 'near' each other on 2-D plot, can, rub, hold, dog, cat, and they could all be near each other.

Some details: First, lemmatize words, e.g., house, hold, work, rub, rub.

Next: Simple is too slow; for realistic sizes, want you to operate on larger vocab, larger dim, larger corpus.

So a faster way is called the Skip-gram method of training word vectors.

This section allows you to get back up to speed on training; it illustrates the word embedding concept from scratch.

With very small dimension, small vocabulary, and small corpus.
Skip-Gram with Negative Sampling:

- rather than predict which of the \( |V| \) words in the vocabulary are associated (related) to the target word, (a multi-class classification)

- instead, we make a binary prediction as to whether a given pair of words (target word, context word) are related or not.

- to do so, we need positive examples of related words - exactly the as Skip-Gram above, but also need negative samples of word pairs that are not related. (To properly train a binary classifier)

- so, to train, create the positive examples as above.
- with a given window size, etc.

- to create the negative samples, for each target word, we randomly sample words from the entire corpus - why is this OK? [i.e. won't some positive examples be included?]

- this sampling gives the method its name: Skip-Gram with Negative Sampling.

- described in Jurafsky, Section 6.8.

- How many negative samples? Often 2x as many negative than positive.
- the full skip gram method suggests basing the random sampling of negative examples to avoid the "high frequency" (most common) words in the corpus left as a bonus in Section 4, Assignment 1 to explore.

- so, the binary prediction "are these two words related or not" can be expressed as follows:

Given target word's vector $T = (t_0, t_1, \ldots, t_{d-1})$

context $C = (c_0, c_1, \ldots, c_{d-1})$

where $d$ is the embedding size

then compute dot product $C \cdot T = \sum_{i=0}^{d-1} (t_i \cdot c_i) = P$.

$\Rightarrow$ convert to a probability using sigmoid function

$P(related) = \sigma(P)$

then use binary cross entropy to compute loss

i.e. if label = 1 (related) loss = $-\log(P)$

= 0 (not related) loss = $-\log(1-P)$
- Note again this "similarity/related" computation is a dot product $C \cdot T$.
- Where else have you seen dot product operate as a similarity function? (see kernel)
- Discuss the math of it.
- Words that are related have a higher dot product.
  (return to 2-9)

This is an essential observation that is used in the key model we will cover in this course: the Transformer. The **Attention** module inside the Transformer are based on this relation.

- Some people now think of a Transformer as an extension of a CNN because of the link between kernel convolution and attention.

- **Assignment 1, Section 4** asks you to train word vectors for a much larger vocabulary than Section 3 ($>2000$ words) a much larger dimension ($d$) and a much larger training corpus ($>60K$ words)
  - Used just $500$ would be too slow.

- Uses different tokenization.

- To visualize the results in two dimensions, must reduce the dimensionality from 8 to 2 using Principle Component Analysis as described in Section 4.1 starter code.