ECE 1786 Lecture #2

Work-in-Flight:

- Assignment 1 is due next week
- If new to course, must fill have filled out survey to see if have pre-requisites; should have received response from Soliman Ali, Assistant
- If got query back and responded, please try Assign 1 Sections 3 and 4

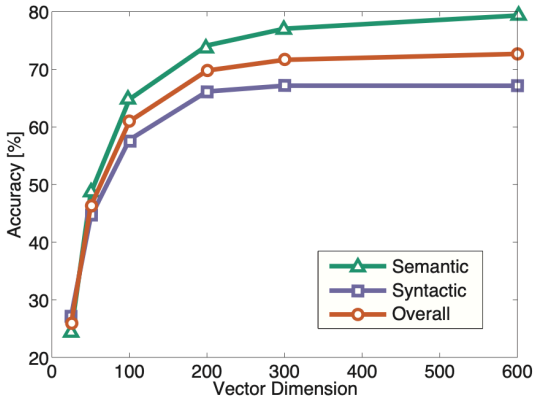
Last Day: Word Embedding Properties & Meaning Extraction

Today: How Word Embeddings are Created

Recall: word embeddings represent the meaning of words with numbers/vectors

- means that words that are associated have embeddings that are close + other more sophisticated relationships as described in Lecture 1/ Assignment 1
  - Vqueen - Vking = Vwoman - Vman
- This remarkable notion helps computers deal with the ambiguity in language
- You should have done Assignment 1, Parts 1 & 2 by now;
- Section 3 and 4 are a lot of work, and require all of the background

- Last week we discussed (a little) how big the size/dimension of the embedding vector should be; orginal work on GloVe suggested diminishing returns beyond size 300:



What is the basis for inferring meaning?

Consider these 3 sentences:

1. The mathematician went to the store
2. The engineer went to the store
3. The mathematician solved the problem

Sentence 1 & 2 imply a similarity between engineer and mathematician (which is what?). It also makes this sentence not surprising:

4. The engineer solved the problem

This is an example of what is called the Distributional Hypothesis: "Words appearing in similar contexts are related" (e.g. engineer & mathematician - why?)

How to Create (train) word embeddings? The subject of A1 parts 3&4.

- is a clever & complex idea that begins with Bengio -> Mikolov
- although statistical word representation has been around since the 1960s, it really came into its own in from these two

Big picture of the how word embeddings are created:

- train neural network to make a prediction based on the meaning of a word
- inside the neural network, the meaning of each word will be encoded
- the training of the network will cause the encoding to be learned

That encoding is the embedding/vector of the word.

- please accept my apologies for now using 3 terms that refer to the same thing: encoding/embedding/vector
- your understanding of this general notion may be familiar to you from any work/exposure you've had to auto-encoders (ask)

So, what is one kind of prediction task that we can use in this training? Also, where do the labels for the prediction task come from?

One task is to be able to predict words that are likely to be near a given word in a sentence. ("Words appearing in similar contexts are related" in meaning)
=> the prediction will need to be based on the meaning of the word.

We have a ready-made dataset of input-output examples (i.e. input and label for a ML prediction problem): every sentence ever written!!

- Yippee!! This dataset is already labelled, as you'll see shortly

Consider some of the simple sentences taken from SimpleCorpus.txt in Assignment 1, part 3: I hold a dog. She holds a dog. He holds a dog. I rub the dog. She rubs the cat.

- even these simple sentences give some clues to the meaning of the words used, and which are related - what are they?
- we can generate neural-network training examples of words that appear together (and therefore are associated) by selecting words that are 'near' each other in 'correct' (valid) written sentences
  - correct = in grammar and meaning
  - near = within a few (1-3) words on either side of a target word

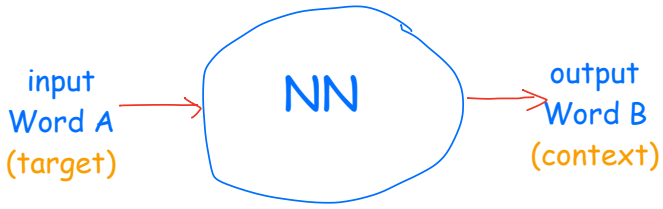
- for example, consider "I hold a dog"; take near to be words +/- 2 words away from the target, then the training examples of related pairs of words are:

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

Target: I hold a dog Context words

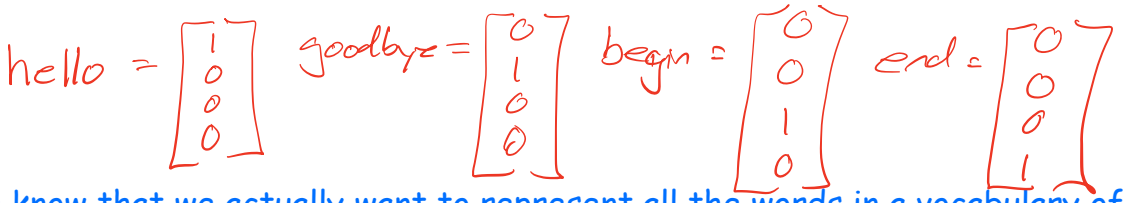
- clearly, we can make a lot of these pairs from all the writing!!

- want to build neural network predictor that, given word pair (WordA, WordB) can predict WordB given WordA.



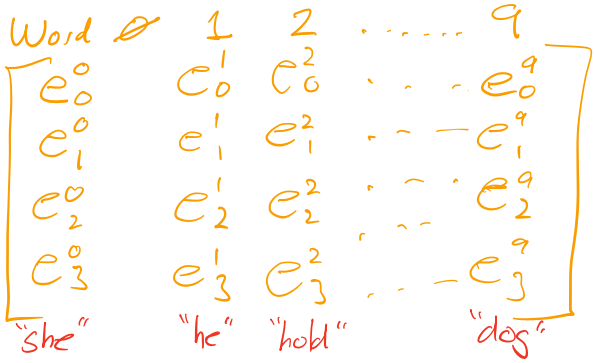
- assume that words can only come from a specific, limited vocabulary
  - vocabulary is the set of words allowed to be used

- a key part of this is how the input and output are represented
  - what is the simplest way to represent the words in a vocabulary of size V?
  - answer: 1-hot encoding, where there are V numbers, where all except one of which are 0, and the one corresponding to a given word is a 1.
  - e.g. for a vocabulary with just V=4 words:hello, goodbye, begin, end



- we know that we actually want to represent all the words in a vocabulary of size |V| in an embedding that is much smaller than |V|
  - that's what the first lecture was all about, as the vocabulary for GloVe is many thousands, and we were using d=50 or so

- let's consider a simple example, similar to Assignment 1, Section 3
  - let's say the vocabulary size |V| = 10 (vs. typical of 5-10,000 words)
  - let's assume the embedding size dimension = 4
  - that means we want 10 embeddings of size 4 each
  - these are typically stored in a matrix, like so:



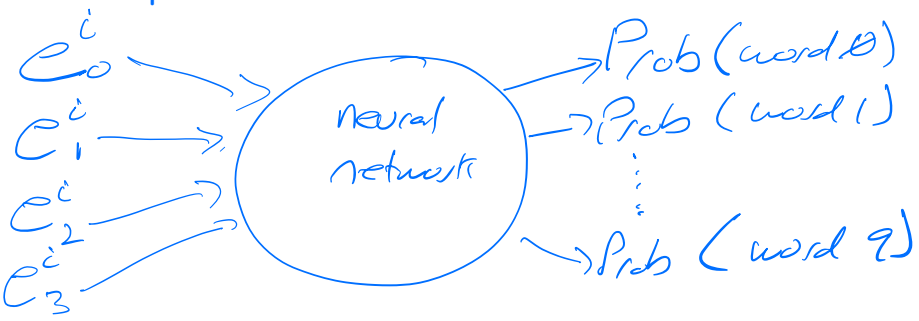
called an "embedding matrix"

- these values will be used to represent the various "word A" in our prediction machine
- Key: these values will be randomly initialized and learned through training, just like the weights and biases you've trained in prior experience

- I could have posed this to make it exactly weights, but chose not to

So, the input to the neural network will be these 4 values (e^i_0 -> e^i_3) associated with wordA in the embedding matrix

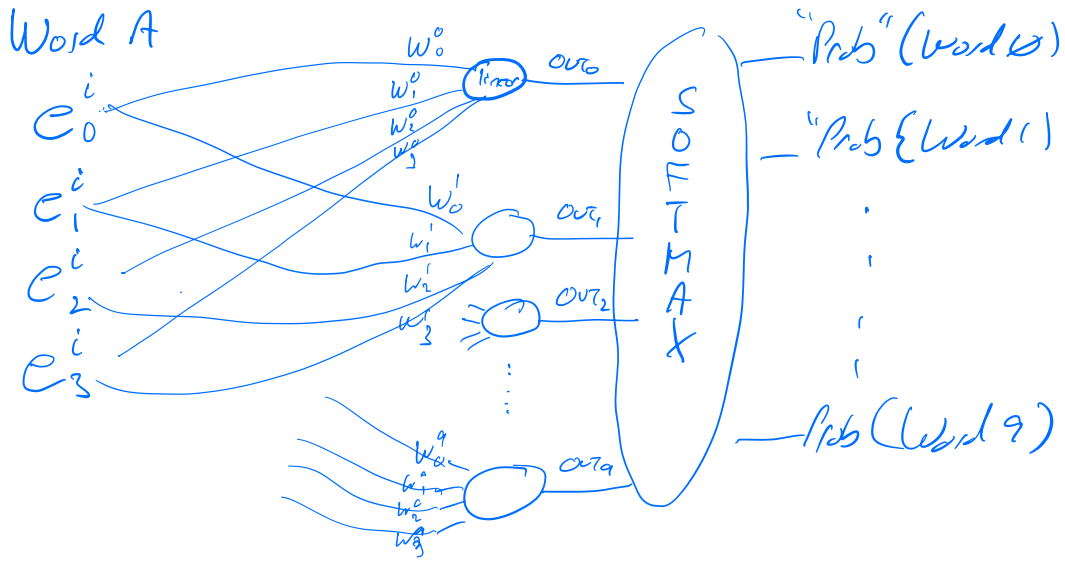
But what should the output be?



i.e. we predict which specific word is wordB

- but can't force a neural network output to be 1-hot (one output 1, rest 0)
- so we convert the outputs across the vocabulary to be a probability
- (using softmax)

- where i is the word number of WordA (the 'target' word)
- Below is the 10 (|V|) - neuron neural network that will do the prediction task:



- this diagram needs to have familiarity for you, or at least you can connect it to the required background of this course

- That includes: linear neurons, equation: $OUT_0 = \sum_{j=0}^{3} w_i^j e_j^i + b_0$

- Pytorch gives you this calculation in the function torch.nn.linear
- Also, please recall, lookup the Softmax normalization computation
- Also, how to train this network as follows:

So, to train this network, you would present many examples, from a dataset of pairs (WordA, WordB).

- WordA is the target, i.e.: e_j^i
- WordB is the context, but also the correct 'label' for the supervised training
- How is WordB represented? Again, as a 1-hot encoding
  - with a vocabulary of size 10, then the 1-hot encoding has 10 numbers
- What is a good loss function? Cross Entropy
  - This is the -log of the wordB output that is the correct answer
  - e.g. if word 8 was the correct answer, the loss is computed as the negative log of Prob(Word8)
  - this -log cross entropy loss should be familiar to you from multi-class classification in your prior experience/work
- Crucial, to repeat: values of eij are also learned through the gradient descent training process
- Also, note: PyTorch combines softmax and cross entropy into one function for reasons of numerical stablity

---

- An interesting nuance/improvement that isn't obvious: observe that the wij (superscript i, subscript j) - the weights in the above neural network model are also word embeddings that are being trained!
- Why is that? There is one associated with each output, which is again one word in the vocabulary.
- The model is using these to 'match' (convolution-wise) to the input embedding. [This takes a bunch of thinking to understand]

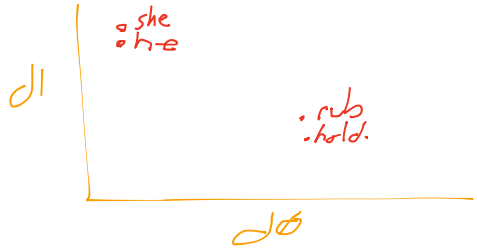
- BUT: we don't actually need two sets of word embeddings, we just need one of them, so we'll use same embeddings in the two places. Your code should reflect this, and takes some thinking as well.
- To be clear, to do this, you should replace the wij with the eij - the same trained/learned parameter appears in both places.

- One last way to think about this: if you wanted to perform the prediction task: "Do these two words belong/relate to each other?" you just need to compute the dot product of their respective embeddings, and the higher that is, the more related they are. (Just like a convolution kernel).

Assignment 1, Section 3 asks you to train this prediction task/embedding creation:

1. Using a vocabulary of size 11: I, she, he, rub, hold, can, dog, cat, and, the, a
2. A very small "corpus" of training sentences. (smallsimplecorpus.txt)
3. Use an embedding dimension/size of 2

We want you to see if similar words end up 'near' each other in a 2-D plot of the words in the two dimensions, i.e.:



- similar to what you see in Sections 1 and 2, but which you trained!

- important detail: words in the corpus are first reduced to their "root" word in a process that is called lemmatization:
  - i.e holds -> hold
  - rubs -> rub
  - rub -> rub

- i.e, we only make an embedding for the root, not for both holds and hold

Section 3 will help you get back to familiarity with software for training a neural network; you'll be training one 'from scratch' (i.e. not pre-trained, all parameters are initialized randomly). This is called the 'skip gram' method

One problem with this method is that it is slow - because there are many outputs to compute, equal to the size of the vocabulary - which is very small (11) in Section 3, unrealistically.

A faster method, which you're asked to use in Section 4, is called Skip Gram with Negative Sampling:

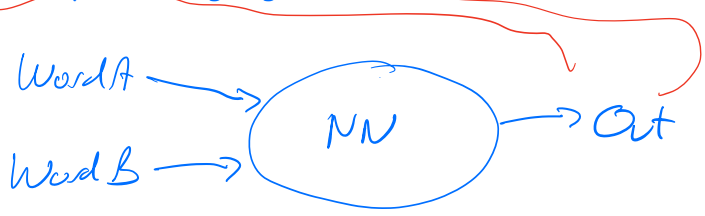
- rather than predict which of the |V| words in the vocabulary are related to a given word, we instead make a binary prediction of whether a given pair of words (target word, context word) belong together or not.

- to do so, we need positive examples of related words (exactly the same as above), but we also need negative examples of word pairs that are not related
  - as you know you need positive and negative labels in a binary classification

- we use the same positive examples from above; to create the negative examples, we randomly sample pairs of words from the entire corpus (making sure to not include the same word as both target and context)
  - why might this work? Randomly sampled words are unlikely to be related, and its ok if a few of them are related
  - this sampling is why we call the method 'skip gram with negative sampling'
  - Question: how many negative samples are needed? Jurafasky section 6.8 suggest twice as many negative as positive samples

So, the binary prediction for this method is the answer, yes or no, to the question "do these two input words belong together or not?"

Given two input words WordA and WordB, expressed as word embeddings of size d, predict 1 if yes, they do belong together, and 0 if the answer is no, they do not.



WordA's embedding is (a0, a1, a2, … ad), Word B is (b0, b1, b2, … bd)

In a similar way to the skip-gram method, which essentially compares a given input wordA to every possible wordB in the vocabulary, through a dot product, we just use one dot product here to compute the similarity between wordA and WordB. It is interesting, again, to note that this operation is the same as the basic linear neuron operation. So the NN becomes:

$$WordA \cdot WordB = \sum_{i=0}^{d-1} (a_i \times b_i) = Out$$

We convert Out to a "probability" (P) using the sigmoid function (really just force it to be a number between 0 and 1):

$$\text{i.e. } P = \sigma(Out)$$

This probability is turned into the neural network training loss function using binary cross entropy, i.e.

if label = 1 (yes, related) loss = $-\log(P)$

if label = 0 (no, not related) loss = $-\log(1-P)$

recall from previous course – why is it a logarithm? → really pushes hard against wrong answer

· 2-10

———

The full skip gram with negative sampling method has some nuances discussed in the assignment.

When randomly sampling words for the negative examples and the positive examples, avoid the high-frequency words (such as "the", "and" …) as they appear near many words, and so don't add much information. Reducing their appearance in the labeled dataset makes training more efficient.

Other notes about Assignment 1, Section 4, on Skip Gram with Negative Sampling:

- Because this method is more efficient, we can train larger sized embeddings than Section 3, use a much larger vocabulary, and train from a much larger corpus (the provided LargeCorpus.txt).

- In Section 3 we just used 'lemmatization' as mentioned above as the 'tokenization' method. Tokenization is the process of converting input words into known, specific inputs. The tokenization process given to you in Section 4 is a little more complex, mainly just removing punctuation as well as lemmatizing.

- Since you're asked to use a dimension size of 8 for the embeddings in Section 4, to visualize these, you need to reduce the dimensionality down to 2. Dimensionality reduction would have been covered in a previous course, but the code for doing so is done with principle component analysis (one of several ways to do this), and the code is given to you to do this. This allows you to visualize the embeddings with the same 2-D plot as used in Section 3 of the assignment

Good luck on Assignment 1 - due next Monday at 9pm.
DO NOT WAIT to begin, it is a long assignment!