ECE 1786 Lecture #2

Last Day: Word Embeddings Properties & Meaning Extraction.

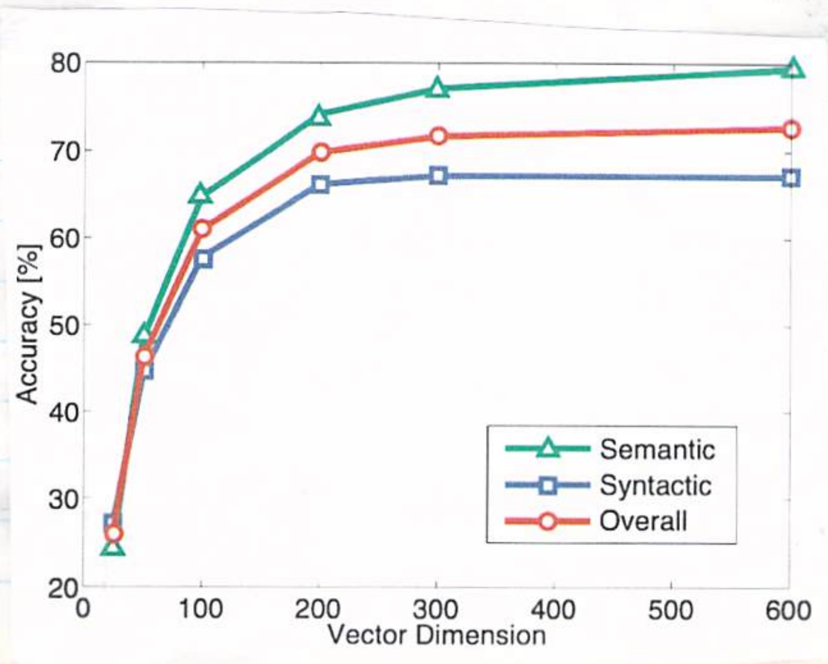
Work-in-Flight: – Assignment 1 due next week.
– Fill out survey if ~~needy~~ have not – new registrants

Today: How Word Embeddings are Created. & discuss adding.

Recall:
- Word embeddings represent meaning of words in numbers.
- helps neural networks deal with ambiguity in language
- ~~Some~~ 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 size 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 of each word will be encoded. = the word vector/embedding

The training of the network will cause the encoding to be learned.

That encoding ~~can~~ IS the embedding/vector.

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

LINK TO AUTO ENCODERS

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

* The Prediction Task: Given ~~Does~~ word A "belong" (→ what works) with it? ~~word B?~~

i.e. ~~are they related somehow?~~
⇒ are they often used together?
→ predict a word that is often used together with word A

Let's contemplate this by

Considering 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 what!

Sentences 1 & 2 imply similarity ~~of~~ between engineer & mathematician

⇒ not surprising that this is a sensible sentence

4. "The engineer solved the problem"

- this is an example of the "Distributional Hypothesis"*:

  → of other words

  "Words appearing in similar contexts are related"

  → are closer in meaning

* above

- since we want to predict which words are ~~related~~ appear near each other, 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 ~~1~~ 1. P3 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 even simple sentences give some clues to the meaning of the words used, and which are related → what are they?

- we can generate NN training examples of words that are ~~related~~ appear together by selecting words that are 'near' each other in 'correct' (valid) written sentences.
  - near ≜ within a few (1-3) words (on either side) of target word

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

	(I, hold)	(hold, I)	(a, I)	(dog, hold)
	(I, a)	(hold, a)	(a, hold)	(dog, a)
	↑	(hold, dog)	(a, dog)	
Target ~~word~~s word + context word.	I	hold	a	dog

- clearly can make a lot of those.

1-hot 4 word vocabulary: hello goodbye begin end

hello = $\begin{bmatrix}1\\0\\0\\0\end{bmatrix}$ goodbye = $\begin{bmatrix}0\\1\\0\\0\end{bmatrix}$ begin = $\begin{bmatrix}0\\0\\1\\0\end{bmatrix}$ end = $\begin{bmatrix}0\\0\\0\\1\end{bmatrix}$

- want to build a NN predictor that given a word pair (wordA, wordB) can predict wordB given wordA:

"target word" "context word"

input: word A → NN. → output: word B.

- ~~for now~~, assume that words can only come from a specific, limited vocabulary → define.

- a key part of this is how the input + output is represented; what is the simplest way to represent all the words in a vocabulary of size V? - 1 hot |V|
  - discuss 1-hot - WORD B IS 1-hot encoded.
- we know that we want to represent eg the words by a vector of size << vocabulary size |V|

(that's what 1st ~~half of~~ lecture was about)

- let's consider a simple example similar to Assignment 1, part 3
  - let's say the vocabulary size |V| = 10 — 10 words! typical vocab ≈ 2K-10K.
  - let's assume the embedding (vector, encoding) size has dim = 4 (typically 50+!)
  - so that implies we've ~~got~~ want 10 embeddings of size 4 each;
  - these are typically stored in a matrix like so

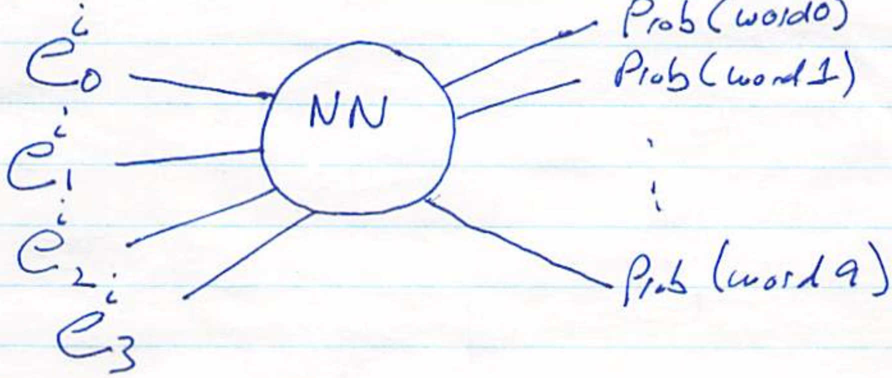
$$\begin{array}{c|cccccc} \text{WORDS} & 1 & 2 & \dots & 9 \\ \hline & \begin{bmatrix} e_0^0 & e_0^1 & e_0^2 & \dots & e_0^9 \\ e_1^0 & e_1^1 & e_1^2 & \dots & e_1^9 \\ e_2^0 & e_2^1 & e_2^2 & \dots & e_2^9 \\ e_3^0 & e_3^1 & e_3^2 & \dots & e_3^9 \end{bmatrix} \\ & \text{"she"} \quad \text{"he"} \quad \text{"hold"} & & & \text{"dog"} \end{array}$$

→ dim × |V| "embedding matrix".

- these will represent word A.
- Key: these values will be randomly initialized and learned through training just like weights & biases (can pose this so that these are exactly weights, but didn't)

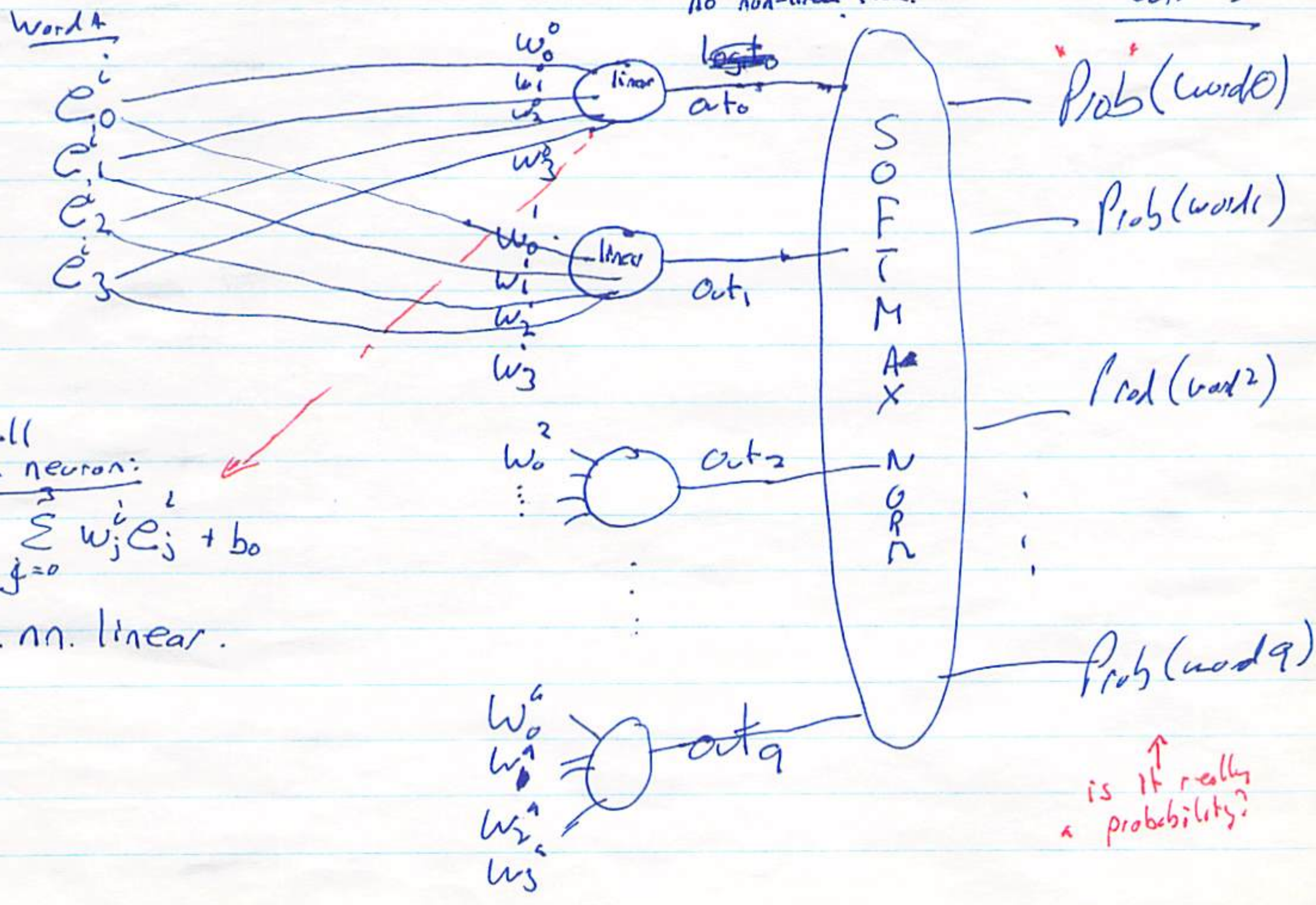
- so the input to the NN will be these 4 values ($e_0 \to e_3$) associated with word A in the embedding matrix
  - but what should the output be?

predict which specific word is word B
- can't force it to be 1-hot
- so use probabilities



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

- here is the ~~lets draw what~~ the |V|-neuron neural network ~~looks like~~: that will do the prediction task
  10



recall linear neuron:

$$out_0 = \sum_{j=0}^{3} w_j^i e_j^i + b_0$$

→ torch.nn.linear.

- So to train this network, you would present many examples of (wordA, wordB) pairs.
  - → wordA is the e^i_j target
  - → wordB, the context, the label, is 1-hot encoded in a vector of size $|V| = 10$. asks → discuss.
  - → Use "cross entropy" loss to train.
  (the $-\log$ of correct answer)

note PyTorch combines softmax + cross-entropy into 1 function for numerical stability

- Crucial, to repeat: the values of e^i_j are also learned through gradient descent.

- 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 (÷ # parameters) by just using 1
  - ⇒ i.e. can replace w^i_j with e^i_j
    (the same parameter appears in 2 places in the network)
    → ~~do this in assignment 1, part 3.~~
  - → 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 this prediction task / embedding creation

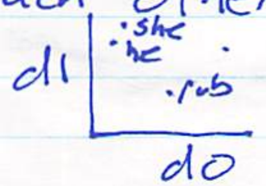
① Using a vocabulary of size 11:

I, she, he, rub, hold, dog, cat, and, the, a

② On a small corpus (smallsimplecorpus.txt)

③ with an embedding size 2

⇒ want you to see if similar words end up 'near' each other on a 2-D plot



- An important detail: words in the corpus are first ~~$~~ reduced to their "root" in a process called lemmatization:

  e.g. holds → hold
  rubs → rub
  rub → rub.

  - so we only make embedding for the root

- Section 3 will help you get back to familiarity with software for training a neural net; you'll be training the neural network "from scratch" (ie not pre-trained)

Next: This method (called Skip Gram) of training embeddings is slow in general because there are many outputs to compute → equal to the size of the vocabulary; faster method called Skip Gram with Negative Sampling

~~Alt~~ - also is Assignment 1 Section 4:

Skip-Gram with Negative Sampling

- rather than predict which of the $|V|$ → which is large > 5K → 300K! 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~~ belongs together or not

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

- so, to train, create the same 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 omitting the word itself. - why is this OK? [i.e won't some positive examples be included?]

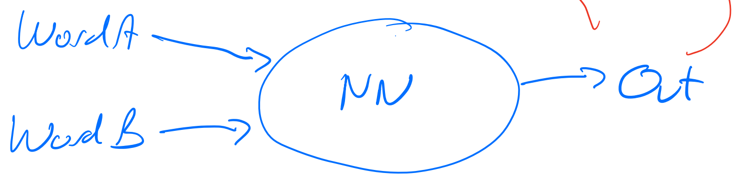
→ this sampling gives the method its name: Skip-gram with negative sampling.

- described in Jurafsky text, Section 6.8.

- How many negative samples?
  - often 2x as many negative than positive.

So, the binary prediction for this method is the answer, yes or no, to the question "do these two input 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:

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

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

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

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

———

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 vectors 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.