ECE 1786 Lecture #1

Welcome in-person! Please make sure that you have done these 2 things:

1. Viewed all of Lecture 0, available on Quercus and course public web page:
   https://www.eecg.utoronto.ca/~jayar/ece1786.2024 [on board]

2. Filled out the course pre-requisite survey;
   - should have seen in email from me, or on a Quercus announcement.

Recap of Lecture 0:

- There have been incredible changes over these three years in what Large Language Models have been shown to do
- This course is about the broader field of Natural Language Processing (NLP) & Deep Learning
  - NLP is the classification and/or the generation of language
- Course consists of lectures, 4 assignments & a large application-oriented project
- Text: Jurafsky & Martin, version 3, August 2024 version
- You must have the pre-requisites as described in lecture 0;
  - My assistant will have/will soon contact you on this.
  - Cannot stay in course if you do not have the pre-requisite understanding and related software capabilities
- This is the third time this course has been taught
- Questions? Survey is due today.

Assignment #1 has just been released & is due Monday September 23rd at 9pm.

- covers material from both Lecture 1 and Lecture 2 (next week)
- you will see that a machine learning/neural network background is necessary to do the full assignment
- DO NOT wait to start the assignment, it is very long!!
- makes use of PyTorch; need to come up to speed on PyTorch quickly if not already known
- will need to install PyTorch on your own computer for Sections 1 and 2 (or else have to keep reinstalling an older torchtext v.12)
- Can use Google Colab for Sections 3 and 4 or own computer

Let's begin: Natural Language Processing

Why 'Natural?' - because computer people were used to computer languages, so needed another term for human written & spoken language

Processing Language was difficult for computers for many years because:

- The ambiguity of language
  - 1 word has multiple meanings (bank, duck), so can several words!
- There are many ways to say the same thing

The understanding of possibly ambiguous sentences needs extra context, e.g.:

"Boy paralyzed after tumour fights back to gain black belt"

"I saw bats" --- in the cave? .... to help small children in baseball?

To deal with this, traditional 'procedural' programming in C, Java, Python would just be way too difficult - there are too many specifics to code, and then combinations of possible meanings would explode. In above example:

- if context was baseball then the bats are wood
- if context was a cave, then the bats are alive

This has been an active field for many decades (beginning in 1960s), with limited success until deep learning came along and was shown to be successful, in 2012.

Instead: modern NLP is based on the encoding of meaning into numbers

We will focus first on the encoding of single word's meaning into numbers, and then move to encoding the meaning of sentences, paragraphs, and more

- analogy with representation inside brain

A word (or a concept) can be encoded into, say, 100 numbers, and we'll call those numbers an 'embedding' or a 'vector,' such that words that are closer in meaning have vectors that are "closer" together.

- The numbers are closer, numerically (more on that below)
- once this is possible we don't have to deal with specific words, just these encodings/embeddings/vector representations

- so, two words with the same meaning would have the same embedding

i.e. each word has its own set of numbers, such as:

Word 1 - apple - Vapple = $\{a_0, a_1, a_2, \ldots, a_{99}\}$

Word 2 - banana - Vbanana = $\{b_0, b_1, b_2, \ldots, b_{99}\}$

...

Word 5000 - zebra - Vzebra = $\{z_0, z_1, z_2, \ldots, z_{99}\}$

Since apple & banana are similar - both fruit - we expect their vectors to be 'close' in a numerical way.

One way to measure this closeness is with Euclidean Distance; e.g. the distance between Vapple and Vbanana is:

$$EuclideanDistance = \sqrt{(a0 - b0)^2 + (a_1 - b_1)^2 + ... + (a_{99} - b_{99})^2}$$

- this is computed in PyTorch as the function torch.norm

These vectors form the core of what has been called, for many years (long before deep learning), the "Statistical Approach" to NLP

There are a number of different methods for computing these vectors, as described in Jurafsky Chapter 6

- based on counting the appearance of words in documents - TFIDF/PMI
- won't cover these; instead, we'll use the neural network approach
- before discussing that let's bring home the power of the neural embedding/vector method

-> Demonstrate code in Assignment 1: A1_Section1_starter.ipynb

- break

- in that demo we saw a second method for computing 'similarity' between two vectors: cosine similarity.
  - it gives a number between -1 and 1 (1 is very similar; typically 0->1)

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}},$$

This method considers the direction of the vector, normalizes out magnitude

Also, besides the surprising relationships:

Vqueen - Vking = Vwoman - Vman

There are a number of other pairwise relationships:

e.g. Vbig - Vbiggest = Vsmall - Vsmallest

This can be used to answer a question like: "big is to biggest as small is to ....?" by computing: Vanswer = Vbiggest - Vbig + Vsmall

- then search for word that is 'nearest' to the Vanswer
- using print closest word(s)

- The Mikolov paper mentioned in Assignment 1 talks about many such relationships; you're asked to look into these in Assignment 1:

Table 1: *Examples of five types of semantic and nine types of syntactic questions in the Semantic-Syntactic Word Relationship test set.*

Type of relationship	Word Pair 1		Word Pair 2	
Common capital city	Athens	Greece	Oslo	Norway
All capital cities	Astana	Kazakhstan	Harare	Zimbabwe
Currency	Angola	kwanza	Iran	rial
City-in-state	Chicago	Illinois	Stockton	California
Man-Woman	brother	sister	grandson	granddaughter
Adjective to adverb	apparent	apparently	rapid	rapidly
Opposite	possibly	impossibly	ethical	unethical
Comparative	great	greater	tough	tougher
Superlative	easy	easiest	lucky	luckiest
Present Participle	think	thinking	read	reading
Nationality adjective	Switzerland	Swiss	Cambodia	Cambodian
Past tense	walking	walked	swimming	swam
Plural nouns	mouse	mice	dollar	dollars
Plural verbs	work	works	speak	speaks

Table 8: *Examples of the word pair relationships, using the best word vectors from Table 4 (Skip-gram model trained on 783M words with 300 dimensionality).*

Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza

Meaning & Vectors

- in lecture 2 & Assignment 1, Sections 3 and 4 you will explore how these remarkable vectors/embeddings are created. It is a clever method that leverages the training loop and optimization methods of a neural network

- in one sentence: the vectors are a by-product of a neural network that is trained to make a prediction based on the meaning of a word;
  - if that succeeds (and it does) the meaning gets encoded into numbers!

- however, as you saw with glove["apple"] in the demo above, the numbers in a dim=50 size vector have no apparent meaning to us humans. Lecture 2 will make it clear why that is so.

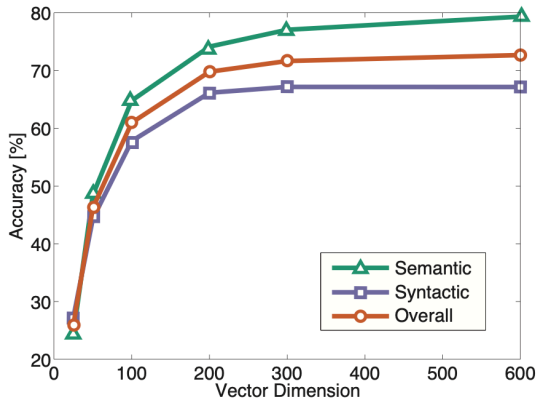
- I find this annoying & would like it to be possible to have explainable elements of these vectors. [although this might cause the loss of the pairwise relationships - I wonder if it does?]

- how might this work, if we were creating vectors by hand?
- suppose that we wanted to wanted to create vectors for words that were based on the following categories of meaning:

word/category	Colour	Temperature	Plants	Human
Mountain	0.2	0.3	0.4	0
Ocean	0.3	0.3	0.5	0
Sun	0.4	0.8	0	0
Judge	0.3	0.1	0	0.9
Radiator	0.1	0.6	0	0
Grass	0.6	0.2	0.8	0

- we can try to fill in each element for each word with a number betwen 0 and 1
- I also wonder how many 'dimensions' (number of elements in the vector) that we need to be able to cover all meaning?
  - I once thought that 300 might be enough, but apparently not!
  - plot below suggests this

Plot from Pennington paper on GloVe - success in benchmarks vs. dimension



- but, we don't get these 'understandable' vectors from the NN training process
- is there a way to compute understandable meaning from the NN-trained vectors?

How? Method 1

- For each 'category of meaning' come up with several words that represent that category
  - e.g. for category colour, choose: colour, red, green, blue, brown
  - why might just colour not be sufficient?

- Then, compute the cosine similarity between any word of interest (e.g. grass, .mountain) and each of the words in the category. Average the results:

i.e. Sum (cosine_sim("grass", "colour") + cosine_sim("grass", "red") +
cosine_sim("grass", "red") + cosine_sim("grass", "green") +
+ cosine_sim("grass", "red") + ... + cosine_sim("grass", "brown"))/6

Method 2:

Instead, average all the words in the category, and compute the similarity of that with the given word.

- You're asked to do this in Assignment 1, in Section 2