15. Deep neural nets and linguistic knowledge in NLP

Learning outcomes

- Define a **deep neural network**
- Run and manipulate a deep neural network
- Run a **SkipGram** neural network
- Describe what a **semantic vector** is
- Compare semantic vectors using **cosine similarity**
- Discuss the extent to which **linguistic knowledge** is used in NLP

Deep neural networks

What is a "deep" neural network?

- The "deep" part of **deep neural networks** (DNNs) is vague
  - Also used for the vaguer "**deep learning**" term, which is dispreferred among some experts like Yann LeCun
- Some researchers say it's any network with ≥ 2 hidden layers
- Others say it's any network with "many" layers
  - This is not a very precise definition, though!
- We will use the ≥ 2 hidden layers definition

Why use a DNN?

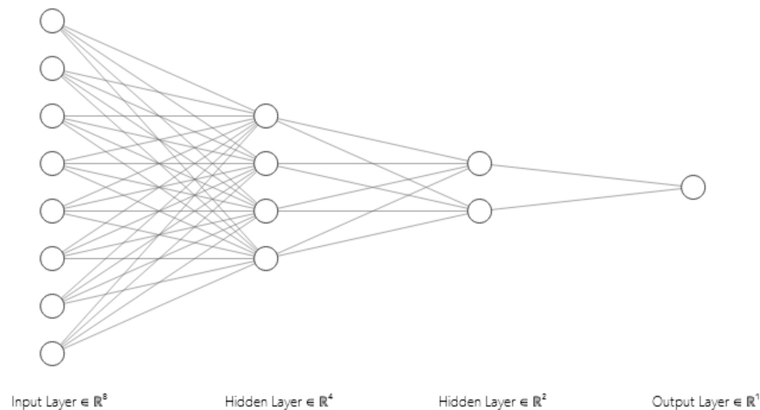
- Increasing depth tends to increase the likelihood of learning a function faster than increasing width
- With sufficient depth, you can start with very "raw" data and output abstract features
  - In vision, could just give the computer an image
  - In audition, could just give the computer a slightly processed waveform
- Using raw data obviates the need for other experts; you can get rid of "pesky" linguists, biologists, physicians, etc.
  - Is this entirely good, though?

A formal description of a DNN

- On Tuesday, we said that we could write a standard neural net as $\hat{y} = W_2 a(W_1 x + b_1) + b_2$
- If we continue to add more fully-connect layers, we just add more weights and biases
  - 2 hidden layers: $\hat{y} = W_3 a_2(W_2 a_1(W_1 x + b_1) + b_2) + b_3$
  - 3: $\hat{y} = W_4 a_3(W_3 a_2(W_2 a_1(W_1 x + b_1) + b_2) + b_3) + b_4$, etc.
- This is all still optimizable

Adding additional layers in Keras

- Keras makes it trivial to add new layers
- Just, literally, add another layer
- It will handle all of the model compilation and training for you




```
m = keras.models.Sequential()

# input and 1st hidden layer

m.add(keras.layers.Dense(4,
activation='relu', input_dim=8))

# 2nd hidden layer

m.add(keras.layers.Dense(2,
activation='relu'))

# output

m.add(keras.layers.Dense(1,
activation='linear'))
```

Other layer types

- Other layer types are also used in deep neural nets
- The two most common types are **recurrent** layers and **convolutional** layers
- We will talk about these briefly, but we will not experiment with them in this class

Recurrent layers

- A recurrent layer is connected to itself
  - Lets you use previous output to inform next output
- Useful for modeling sequential data
  - Like language…
- Most common variant is the **long short-term memory** (**LSTM**) unit

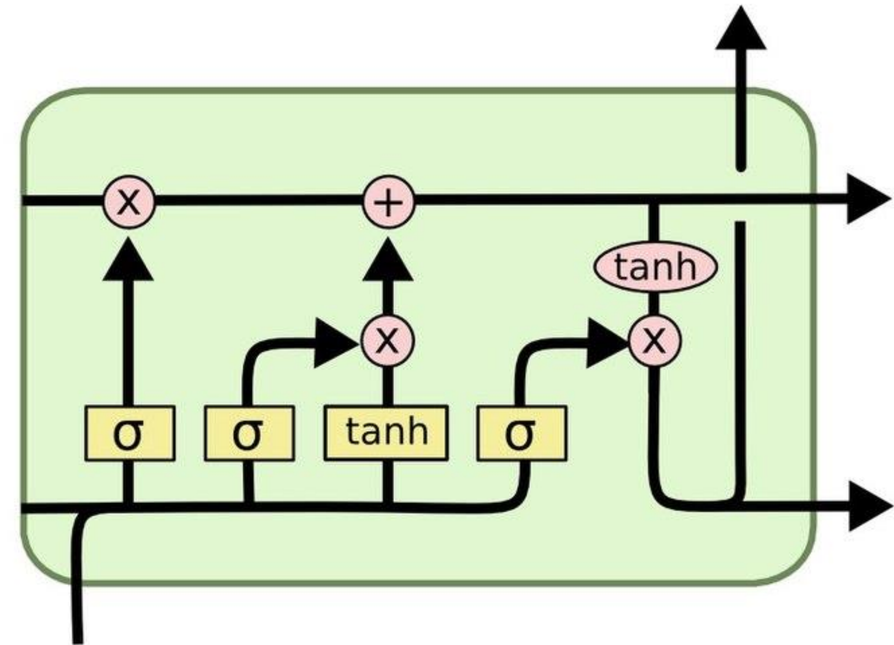


Image from MingxianLin under CC BY-SA 4.0

Convolutional layers

- Inspired by how the eye works
- Pass a small "field of view" or filter over an image/data table
- Deeper layers have larger and larger fields of view
- Very popular because they run fast
- *Can* model sequences with "enough" depth

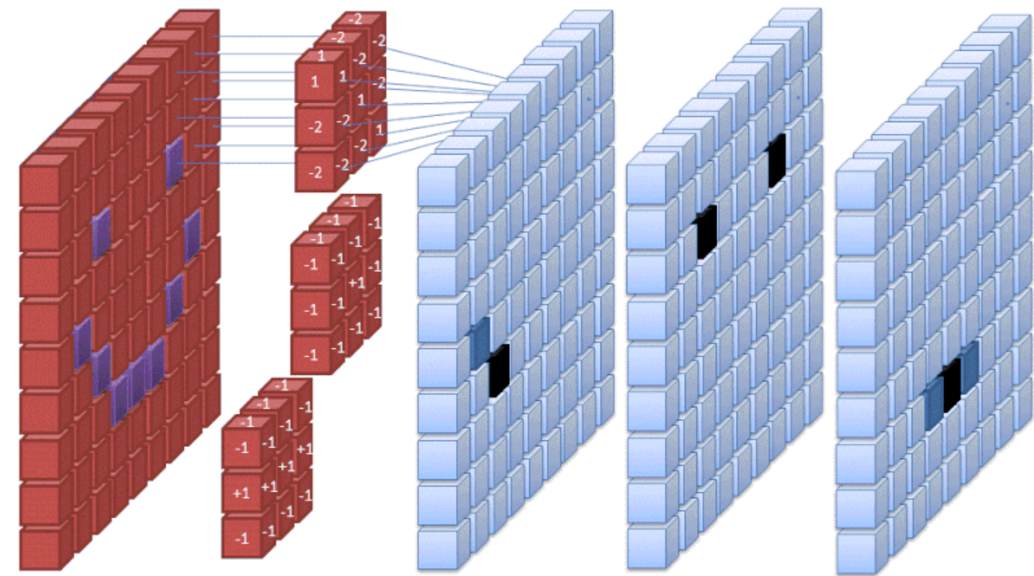


Image from Cecbur under CC BY-SA 4.0

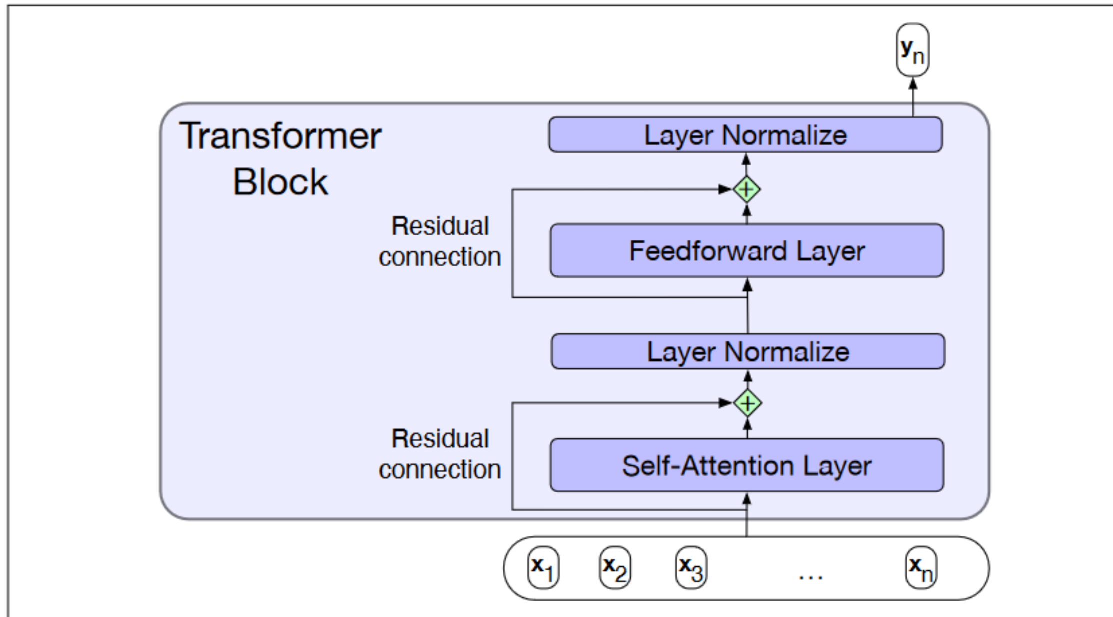


Image from Jurafsky, D. and Martin, J. H. *Speech and language processing (3 ed)*. 2023.

Transformers

- Have a way of determining what parts of the input data to attend to (i.e., they have **attention**)
- Canonical form has a system of encoders and decoders
- Encoders yield an **embedding** that distills the important parts of the data into a compressed format
- These are what modern systems like GPT use

DNNs for language modeling: SkipGram

An initial note on modern techniques

- State of the art models like GPT, BERT, etc. use some techniques that we won’t be covering
  - Requires too much additional training to fit in the course
  - And this is an intro course, not a “contemporary approaches” course
- Instead, we will focus on a precursor technique that is closer to what we’ve already covered
  - More manageable and easier to understand

Language models redux

- On Thursday, we went over the n-gram statistical model of language
  - These are incredibly useful for how simple they are and still widely used today
- However, we can have a neural network perform a similar task and learn probabilities
- By doing so, we can get an indication of a word's **semantics** (meaning)
  - How?

The distributional hypothesis

- There are many approaches to semantics in linguistics
- One that is particularly useful for data science approaches is the distributional hypothesis
  - Described by J.R. Firth in 1957
  - "[A] word is characterized by the company it keeps"
- Effectively, words that mean similar things will occur near similar words
  - From "The king sits on the throne" and "The queen sits on the throne", we might surmise that "king" and "queen" have similar meanings

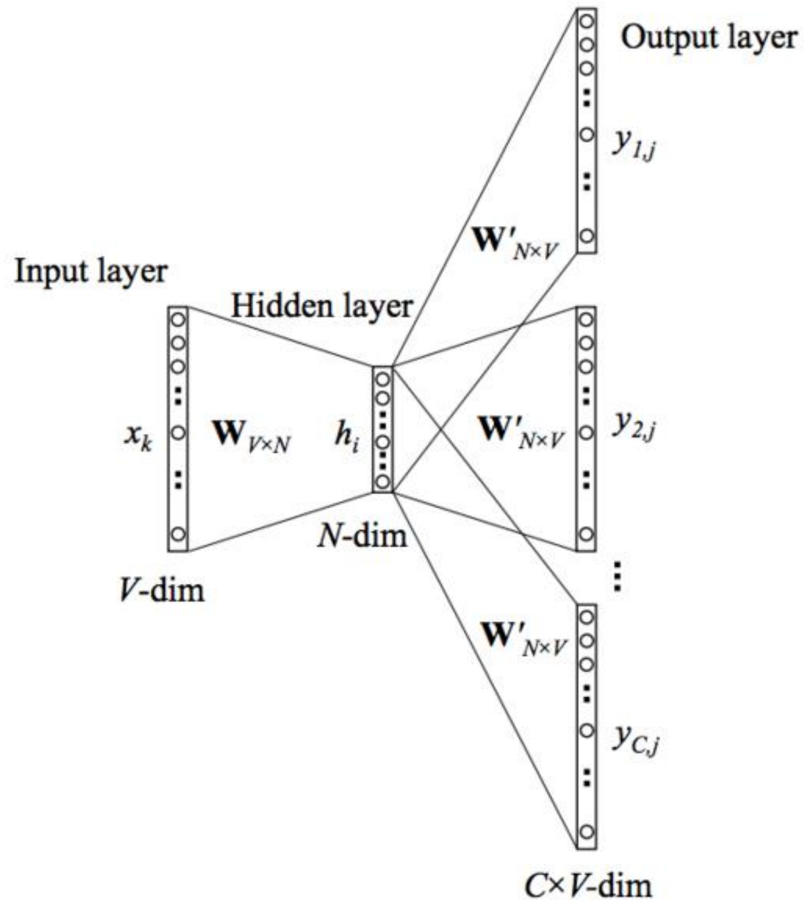


Image from Moucrowap under CC BY-SA 4.0

SkipGram models

- A SkipGram model takes a word as input
  - It then predicts the probability of the two words that come before and after the input word
  - This is a 5-gram!
- We don't ultimately care about these probabilities
  - Instead, we care about the hidden layers as **word embeddings**

Word embeddings

- A **word embedding** is a way to convert a word into a vector of numbers
  - We said a few weeks ago that we wanted to do this
- You generally want these embeddings to make similar-meaning words be converted into similar numbers
- By training a SkipGram, we force the network to learn word embeddings
  - SkipGrams are part of a process called word2vec
  - Other processes exist (we won't cover them)

A simplified SkipGram

- Standard SkipGrams more or less work on 5-grams
- We are going to use a simplified version and just work on bigrams (2-grams)
- We will train a network to predict the word that happens immediately after the input word
  - E.g., if our training text is "The dog barked", we want the system to predict "barked" when given "dog"

Initial word to number conversion

- We have to do a small amount of work to start the word to vector conversion
- General process
  - Find all the unique words in our corpus
  - Assign each word a number (maybe its index when all words are sorted)
  - Create a one-hot vector from that number (one number is 1, all the rest are 0)

One-hot vector for 3rd index

$$\begin{bmatrix} 0 \\ 0 \\ 1 \\ \vdots \end{bmatrix}$$

One-hot vector for 1st index

$$\begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \end{bmatrix}$$

Generating the training set

- Find all bigrams in the text
  - `nltk.bigrams` may be helpful here…
- Convert both items in all bigrams to one-hot vectors as outline before
- Assign the first element in each bigram as your input
- Assign the second element in each bigram as your output
- Then, just train the network as normal!
  - One small caveat: We need a different loss function!

Softmax activation and cross-entropy loss

- The network we have described is ultimately a classifier and not a regressor
  - So, we shouldn't use mean square error as our loss
- Instead, we want to use categorical cross-entropy as our loss, which compares the similarity of two probability distributions
- Also, instead of applying a linear activation on our last layer, we want to apply a "softmax" activation to the last layer, which forces the output to look like a probability distribution and sum to 1

Softmax example

- General formula: $S(\hat{\psi}_i) = \frac{e^{\hat{\psi}_i}}{\sum_j e^{\hat{\psi}_j}}$,
  - Where each $\hat{\psi}_i$ is an element from the network's predictions $\hat{y}$
- Simple example (rounding prevents the final result from summing to exactly 1):
  - $\hat{y} = \begin{bmatrix} 5 \\ -4 \\ 6 \end{bmatrix}, \sum_j e^{\hat{\psi}_j} = e^5 + e^{-4} + e^6 \approx 551.86$
  - $S(\hat{y}) \approx \begin{bmatrix} e^5/551.86 \\ e^{-4}/551.86 \\ e^6/551.86 \end{bmatrix} \approx \begin{bmatrix} 0.27 \\ 0.000033 \\ 0.73 \end{bmatrix}$

Programming activity

- Download the demo code from the class GitHub website
  - A Jupyter notebook is provided; if you know how to run it, you can do so
- Execute the code chunks to generate data and train a SkipGram network
- Then, use the provided functionality to get a semantic vector for a word
- Finally, try changing the input text and see what happens
  - You might try loading in The Complete Works of Shakespeare or Ulysses and seeing how things change
  - You may need to decrease the number of epochs to train the network in a reasonable amount of time
  - You also probably don't have the resources to create the enormous matrices you need to fit the entire data set in at once, so I have added some maximum limits to the code

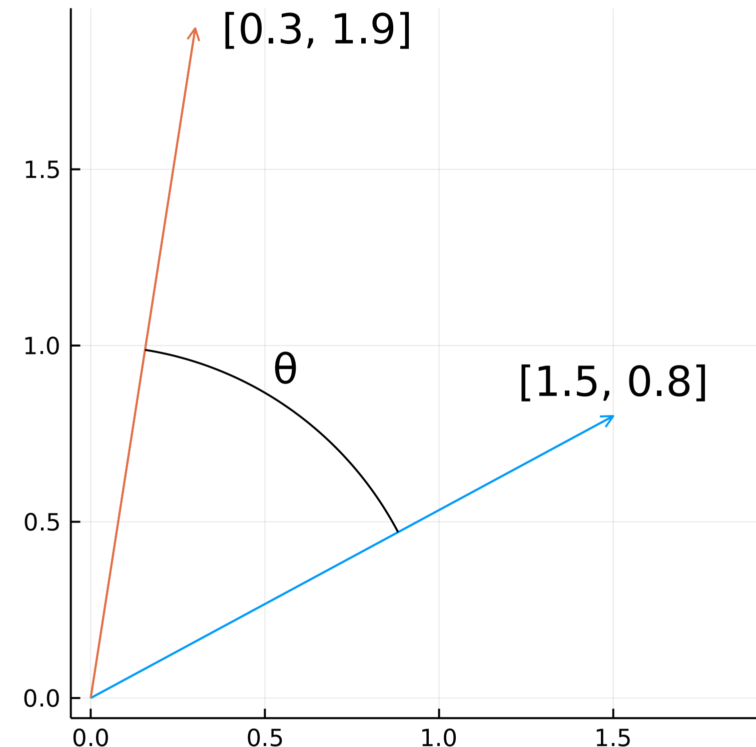
Comparing semantic vectors

Interpreting semantic vectors

- The semantic vectors we get out of the model are not interpretable by themselves
  - They are usually specific to a single model in combination with the training set
  - What might [4.5, 2.8, 1.2, 7.9] mean?
- Instead, we tend to interpret semantic vectors by comparing them to other vectors using linear algebra

Cosine similarity

- The most common way of interpreting a semantic vector is with cosine similarity
- It is the cosine of the angle between two vectors
  - Value is 1 if the vectors point in the same direction
  - Value is 0 if the vectors make a right angle
  - Value is -1 if vectors point in opposite direction



Cosine similarity and words

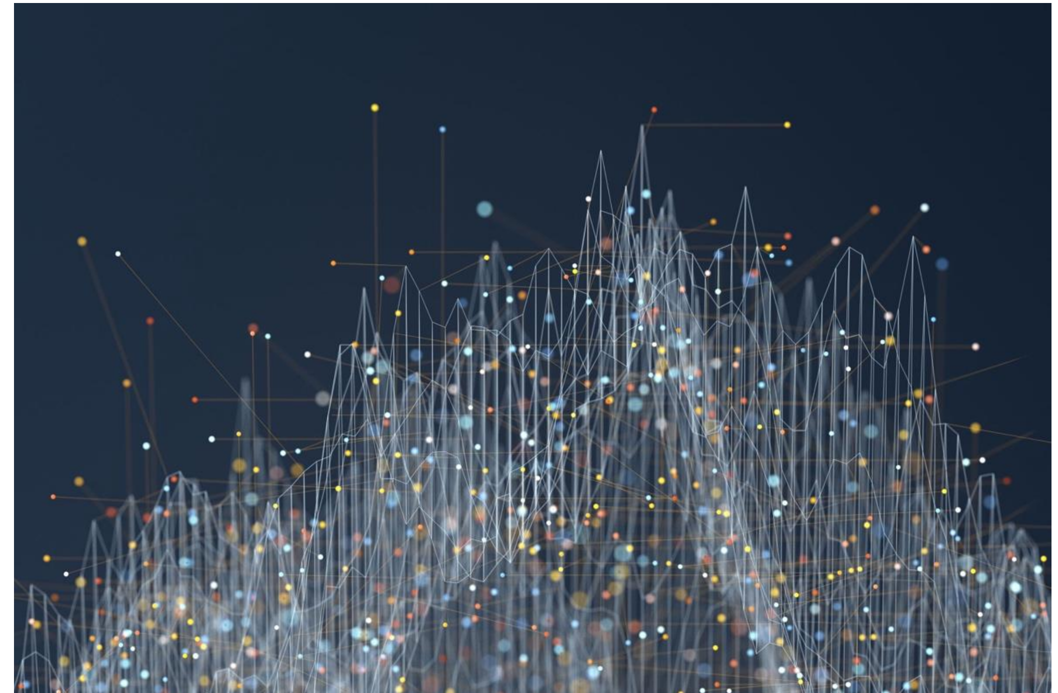
- Since we are representing words as vectors, we just apply cosine similarity as normal
- Values close to 1 indicate that two words mean similar things
  - These are synonyms ("graveyard"/"cemetery") or related words ("Seattle"/"Washington")
- Values close to 0 indicate that words mean different things
  - These are unrelated words (e.g., "steer"/"cupboard")
  - Probably **not** antonyms since antonyms share some meaning (e.g., cold and hot are both temperatures)
- If using all non-negative numbers in vectors, can’t get cosine values < 0
  - A nice property falling out of using ReLU or sigmoid activation

Semantic manipulations

- In many vector space models of words, we can add and subtract words and get reasonable results
- This generally relates to the Word and Paradigm model of morphology where words are related to each other by analogy
  - E.g., cat : cats = dog : dogs
- Common (though problematic) example: *king* – *male* + *female* = *queen*
- Other examples:
  - *Paris* – *France* + *Italy* = *Rome*
  - *cats* – *cat* + *dog* = *dogs* (this is from linguistic theory, not a reported result)

How much data do you need?

- To get "good" word embeddings, you need to train on a LOT of data
- Probably in the billions
  - Original paper on neural SkipGrams trained on 783M words
- "Good" here means embeddings that capture synonymy, relatedness, and analogy



What's the point?

- Why would we even want to have a semantic space like this?
- Well… Many NLP and data science tasks relate words to other phenomena
  - Maybe you want to search for synonyms in a query (like Google)
  - Maybe you want to execute commands after someone has typed or spoken something, like Siri, Google Assistant, and Alexa
  - Etc.
- All of these tasks are made easier (but not trivial) by having a numerical representation of a word that captures its meaning

Programming activity

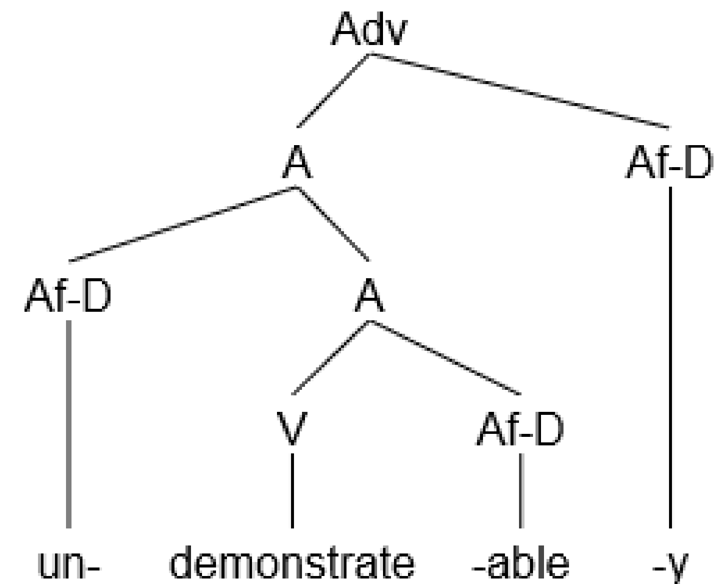
- Use the vector models you trained in the last activity
- Try finding some analogical patterns (see last slide) or similar words
  - Use the scipy.spatial.distance.cosine function to assess similarity
  - You will need to subtract the result from 1
  - E.g., 1 - cosine(cat, dog)
- It's likely we didn't train on enough data to find many patterns, but give it a try anyway

Linguistic knowledge in NLP

Have we completely discarded linguistic theory?

- If you have taken any (non computational) linguistics class(es) before, this probably does not feel at all like the kind of analyses and experiments you have performed
  - This is partly true
  - And partly false

$$\begin{bmatrix} \text{C} \\ -\text{cont} \\ +\text{cor} \end{bmatrix} \rightarrow [\text{ɾ}] \; / \; \begin{bmatrix} \text{V} \\ +\text{stress} \end{bmatrix} \; _ \; \begin{bmatrix} \text{V} \\ -\text{stress} \end{bmatrix}$$

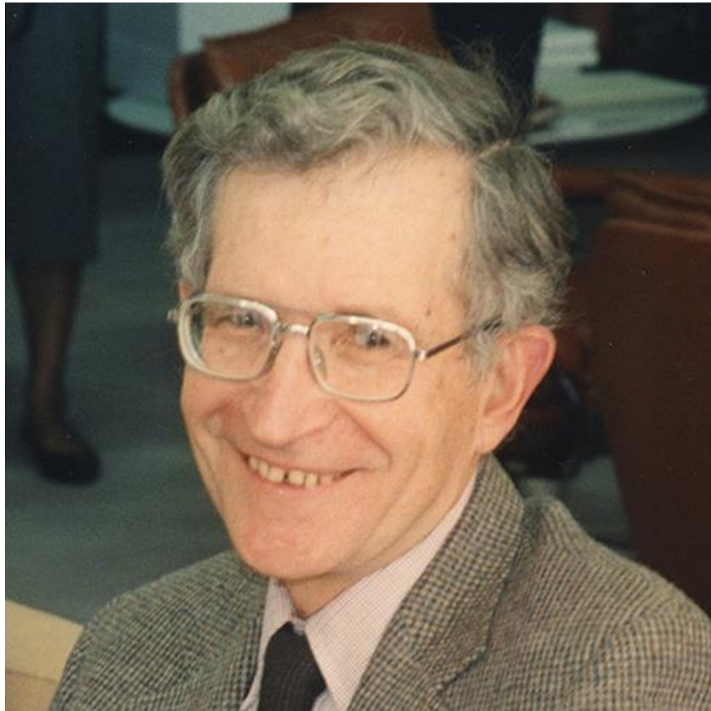


Moving beyond traditional undergrad linguistics

- I would generally agree that n-grams and, even more so, neural language models do not closely resemble linguistics as in traditional courses
- However… Linguistics is broader than just the methods and analytical tools taught in courses
  - It's just that some areas have had less attention paid to when developing course materials
- Semantic vectors and word embeddings certainly align with distributional semantics
  - And distributional semantics was coined in the 1950s…

A case study: Deep and surface structure

DEEP STRUCTURE ADVOCATE: NOAM CHOMSKY



SURFACE STRUCTURE ADVOCATE: ADELE GOLDBERG

Deep and surface structure

DEEP STRUCTURE

- A claimed underlying structure that is the core of language
- For example, *What did he say?* is argued to be *he said what?*
  - The *what* is moved to the first position when the deep structure is converted to a surface structure
- Requires theoretical motivation

SURFACE STRUCTURE

- Language data as it actually appears
- Can be observed in usage
- Can be generalized over

The contrast

- Chomsky advocates an approach to language that requires something built into the mind that is specific to language
  - This "something" is related to the deep structure aspect of language
  - Language is learned by taking advantage of this specific "something"
- Goldberg advocates an approach to language that only requires domain-specific cognitive properties
  - There is no deep structure
  - Language is acquired by generalizing over usage (surface forms)

The tie-in

- Clearly, our semantic vector and our neural network do not have any sort of built in language-specific features, belying Chomsky's approach
  - They are a combination of linear algebra concepts trained with calculus concepts on observed language data
- However, our semantic vectors *are* generalizations based off observed language data
  - Neural networks often resort to generalization and abstraction when confronted with too much data to memorize

Nota bene

- I am not here to adjudicate between the scientific and ontological validity of generative (Chomsky) and constructionist (Goldberg) approaches to grammar
  - They both have vibrant communities of researchers who do that very often already ☺
  - See *The linguistics wars* (2e) by Randy Allen Harris
- Rather, remarking that relating what we have learned to linguistics may simply require doing a bit of research and creative thinking
- But, we must also be careful not to be cavalier and just assume neural nets as literal models of human cognition

April 26, 2023

Why large language models are poor theories of human linguistic cognition. A reply to Piantadosi (2023).

Roni Katzir, Tel Aviv University

In a recent manuscript entitled "Modern language models refute Chomsky's approach to language", Steven Piantadosi proposes that large language models (LLMs) such as GPT-3 can serve as serious theories of human linguistic cognition. In fact, he maintains that these models are significantly better linguistic theories than proposals emerging from within generative linguistics. He takes this to amount to a refutation of the generative approach.

Piantadosi's proposal is remarkable. Not because he proposes to examine LLMs using tools from cognitive science: others have done so fruitfully before (see, e.g., Gulordava et al. 2018, Warstadt, Singh, & Bowman 2019, Lakretz et al. 2021, Baroni 2022, and Wilcox, Futrell, & Levy 2022). Rather, what makes Piantadosi's paper so surprising is his suggestion that LLMs are good theories of (actual) human cognition. Since LLMs were designed to be useful engineering tools, discovering that they teach us about how humans work would be startling indeed, akin to discovering that a newly designed drone accidentally solves an open problem in avian flight. Still, this is Piantadosi's claim, and the present note shows why it is wrong.[1]

https://lingbuzz.net/lingbuzz/007190

Chapter 1

Modern language models refute Chomsky's approach to language

Steven T. Piantadosi[a,b]

[a]UC Berkeley, Psychology [b]Helen Wills Neuroscience Institute

The rise and success of large language models undermines virtually every strong claim for the innateness of language that has been proposed by generative linguistics. Modern machine learning has subverted and bypassed the entire theoretical framework of Chomsky's approach, including its core claims to particular insights, principles, structures, and processes. I describe the sense in which modern language models implement genuine *theories* of language, including representations of syntactic and semantic structure. I highlight the relationship between contemporary models and prior approaches in linguistics, namely those based on gradient computations and memorized constructions. I also respond to several critiques of large language models, including claims that they can't answer "why" questions, and skepticism that they are informative about real life acquisition. Most notably, large language models have attained remarkable success at discovering grammar without using any of the methods that some in linguistics insisted were necessary for a science of language to progress.

https://lingbuzz.net/lingbuzz/007180

An ongoing debate: do LLMs challenge linguistic theory?

Other ways linguistic knowledge is used in NLP

- Linguistic knowledge can be more directly used in these kinds of models
  - If you tag words as nouns, verb, adjectives, etc., you are using linguistic knowledge
  - If you count words, you are using linguistic knowledge
  - If you are performing sentiment analysis, you are using linguistic knowledge (re: polarity and semantics)
  - If you parse sentences using a grammar, you are using linguistic knowledge
  - Etc.