DEEP LEARNING LANGUAGE MODELS: BEYOND N-GRAMS

LING 1330/2330: Introduction to Computational Linguistics

Tianyi Zheng

November 30, 2023

OVERVIEW

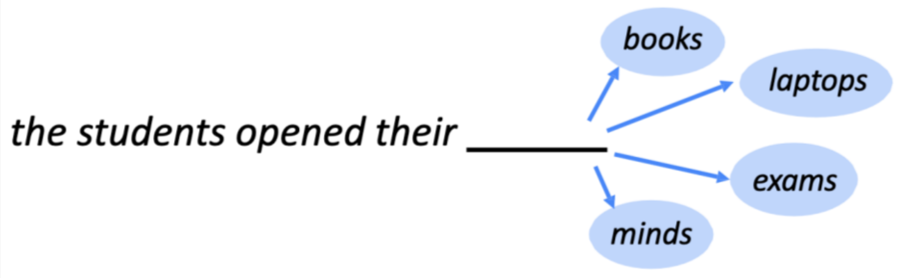
Goal

Cover the *high-level* basics of modern language models

1. Review: language models and n-gram models
2. Super-fast summary of neural networks
3. Recurrent neural networks (RNNs)
4. Encoder–decoder models
5. Attention
6. Transformers
7. ???
8. Make ChatGPT
9. Get fired from OpenAI lol

REVIEW: WHAT IS A LANGUAGE MODEL?

- A language model (LM) is a system that calculates probabilities for sequence of words based on a corpus of text data
  - Recall Austen vs. Melville: how likely was a text to contain the word “she”?
- We’ll focus on the use of LMs for novel text generation (cf. ChatGPT)
  - Given a sequence of words, what’s the most likely next word?



N-GRAM MODELS…

- Recall that an n-gram model predicts the next word using the previous $n-1$ words:
$$P(w_{k+1} | w_k, w_{k-1}, \ldots, w_{k-n+2})$$
- How did we use n-grams to generate novel text in HW 2?

"Bible Speak"

so i say they said jesus had not be, but when i was there was an house for thou hast spoken it to me; but they said in a man, he hath said jesus. for his when they shall i say unto thee? saith thus unto her; the king david. these cities. selah: it to his father which was come upon thy seed for his hand upon thee with

… AND THEIR LIMITATIONS

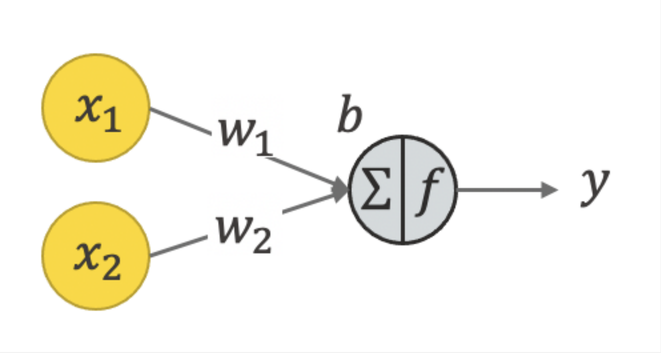
- n-gram models have built-in extreme short-term memory loss!
  - If we only look at the previous $n - 1$ words, what happens as we generate more and more words?
  - How might this memory loss be reflected in the generated novel text?
- Why not increase n? → <u>Sparseness problem</u>

← **ChatGPT** is essentially
an n-gram language model
too at its core,
but a *much more*
sophisticated one!

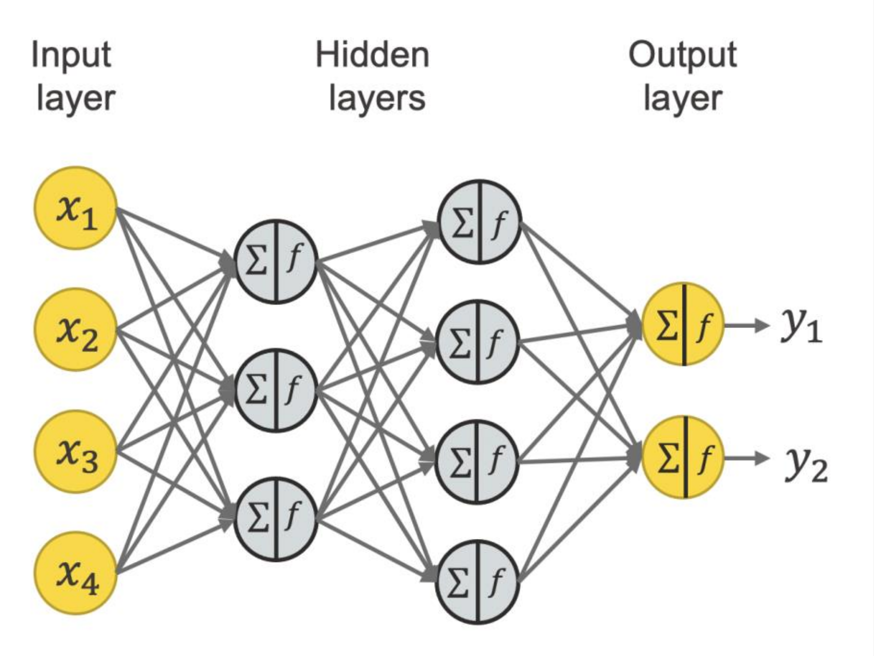
☝🤓
Actually…

NEURAL NETWORKS SPEEDRUN (ANY%)

- A neural network (NN) is an ML model consisting of a graph of "neurons"
- Input starts as a vector of numbers in the input layer, is transformed in the intermediate "hidden" layers, and is converted to a vector of output values in the output layer
  - In the diagrams on the right, f is a non-linear activation function



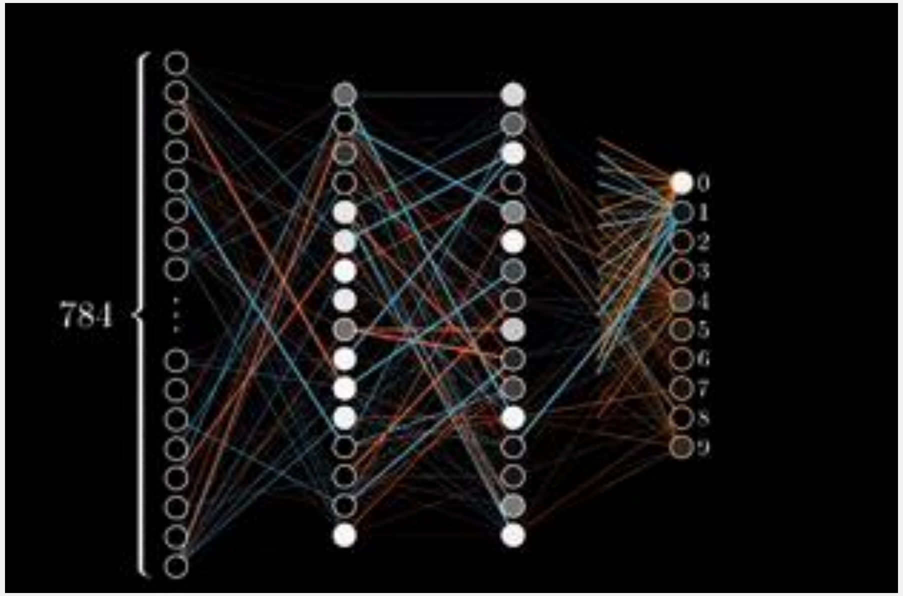
Neuron



Neural Network with 2 Hidden Layers

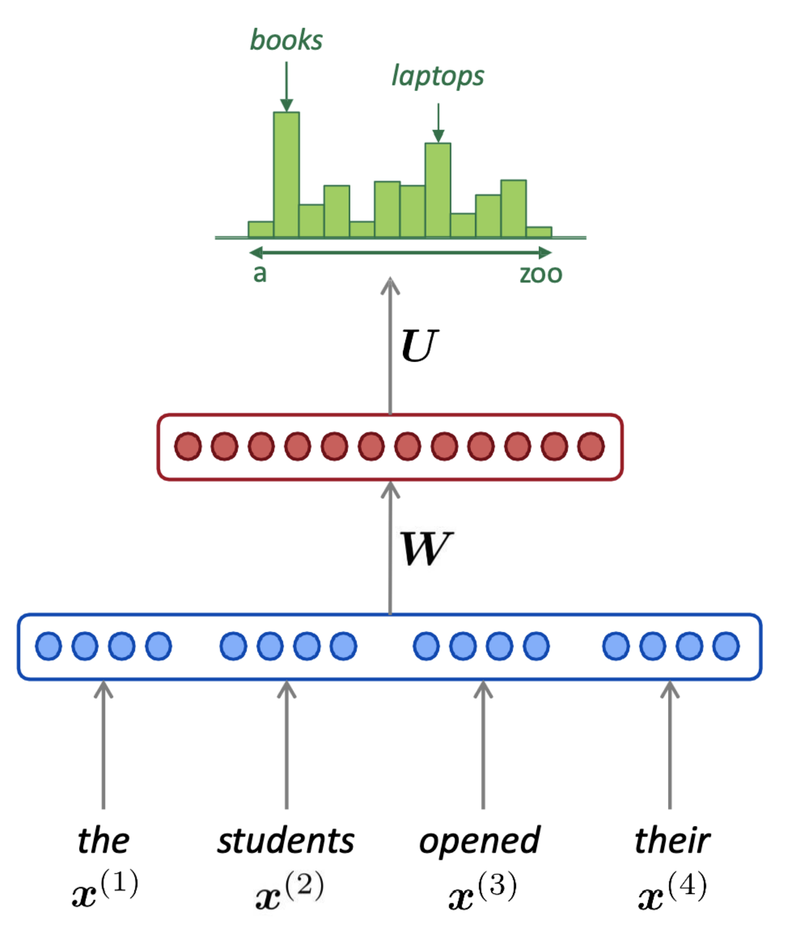
NEURAL NETWORKS SPEEDRUN (ANY%)

- NNs "learn" by updating the weights of their edges using backpropagation
  - Run an input through the NN
  - Compare the model's output to the correct training output
  - Propagate calculated changes backward through the model to update weights



VANILLA NEURAL NETWORKS AS LANGUAGE MODELS

- Words are encoded as vectors (remember word embeddings?) and concatenated to form a single input vector
- We don't need to store any n-grams, so this fixes the sparseness problem
- However, the input size must be fixed, so we still can't support variable-length inputs
- 1 out of 2 ain't good enough

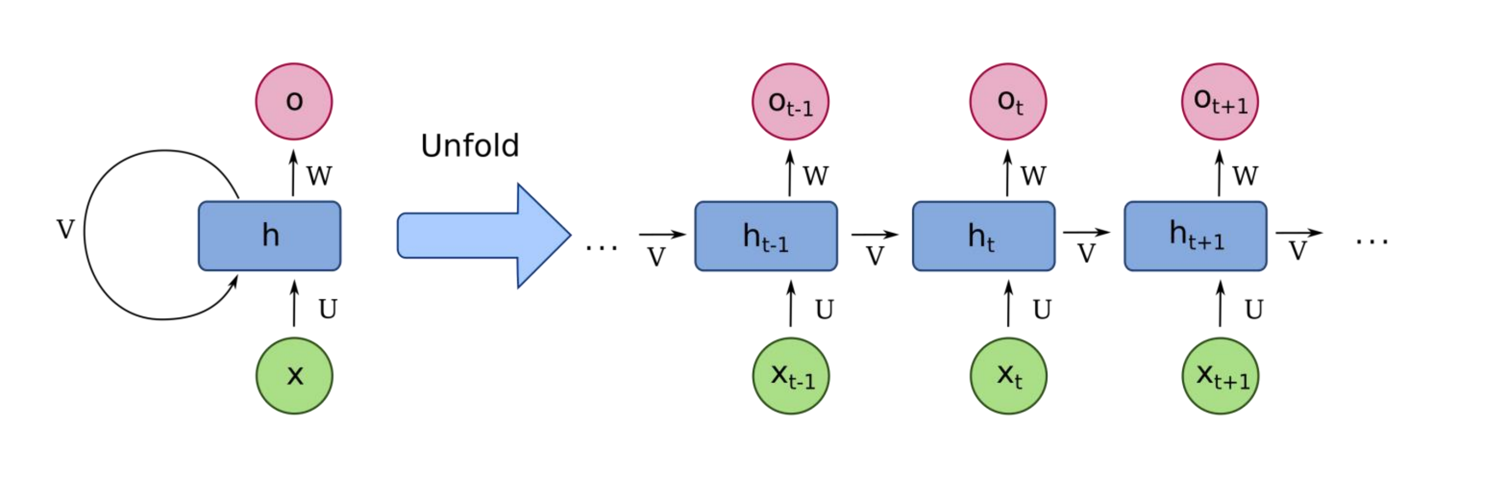


HOW CAN WE SUPPORT VARIABLE-LENGTH INPUTS?

- Suppose we can accomplish the following:
  - Encode input words as a *sequence* of word vectors rather than a single vector for all words
  - Pass input words into the NN one at a time and then have it predict the next word
- Then the NN could handle any number of input words!
- But we need a NN model that supports repeated passes…

RECURRENT NEURAL NETWORKS

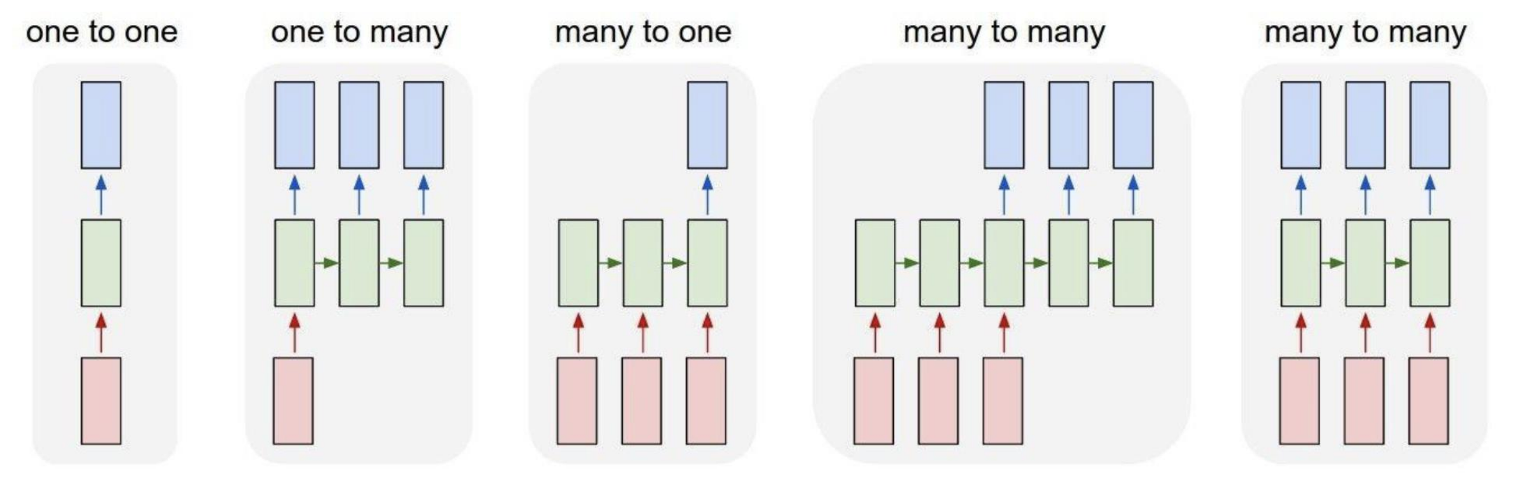
- A <u>recurrent neural network (RNN)</u> has its nodes' output affect those nodes *own* future inputs
  - RNNs use the outputs of previous steps along with the input of the current step



One recurrent hidden layer is interpreted as multiple hidden layers *across time*

RNNS ARE PRETTY NEAT

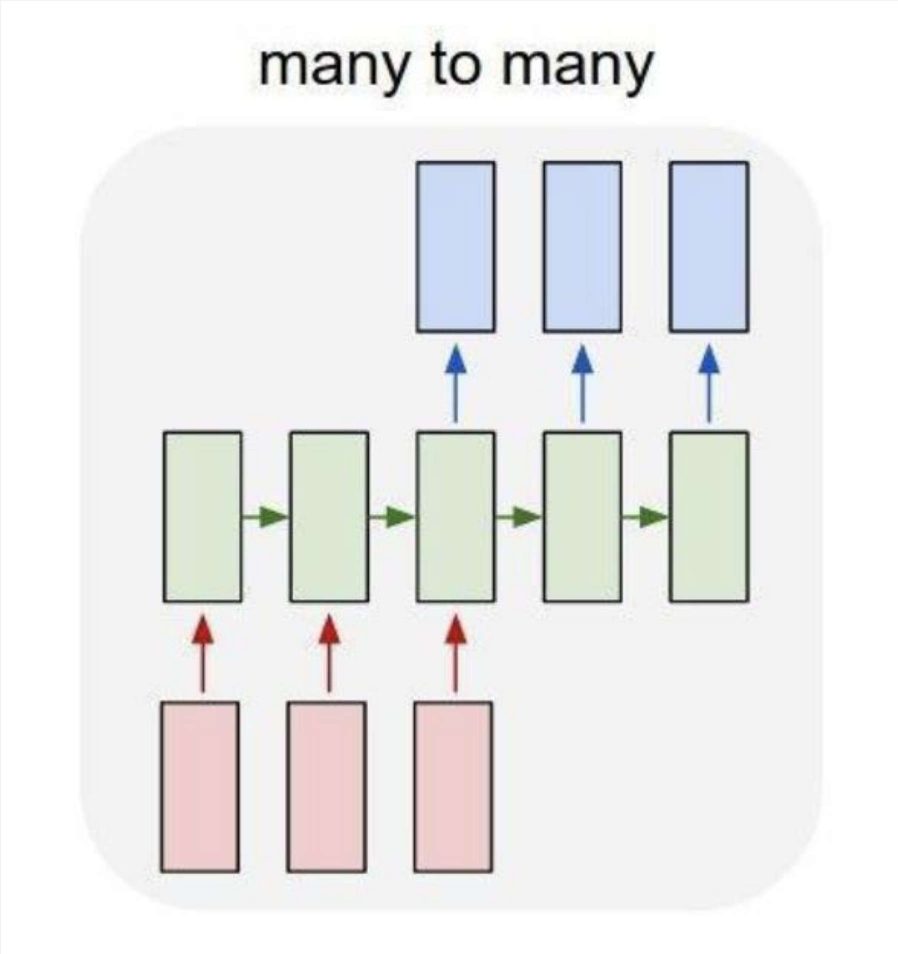
- Can process sequential data
- "Deeper" networks with fewer layers
- Has "memory" of previous inputs



What are some applications of these RNN structures?

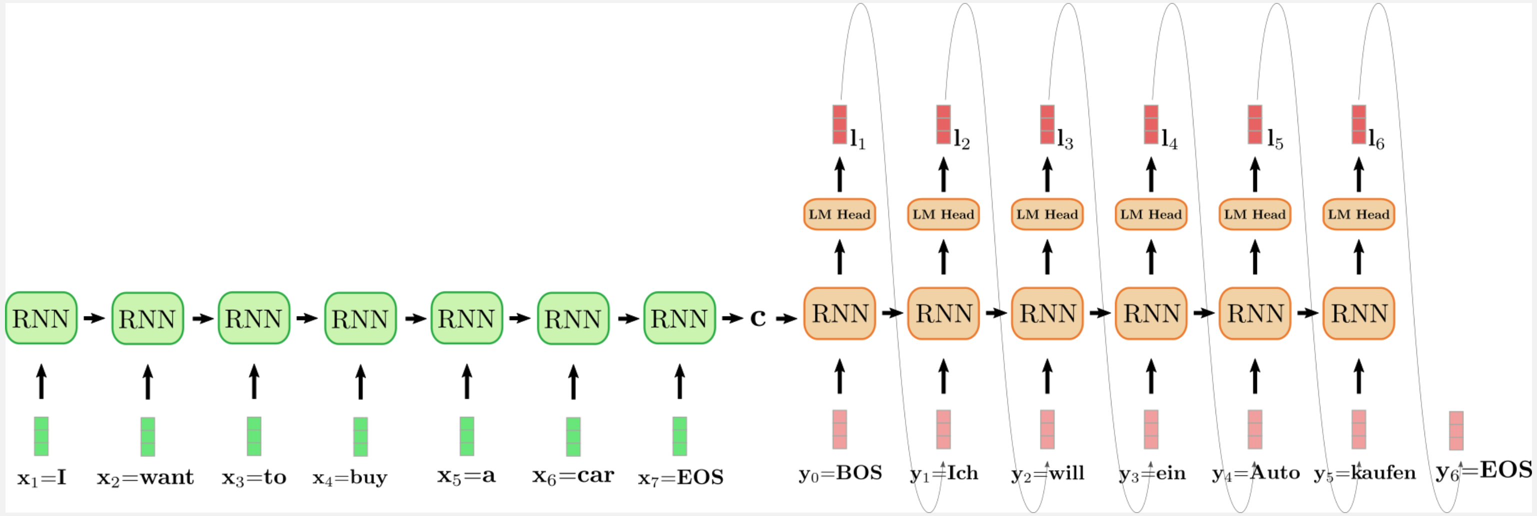
USING RNNS FOR TEXT GENERATION

- LMs take a sequence of words and output another sequence of (novel) words
- We want to process all the input words before producing any outputs



ENCODER–DECODER MODELS

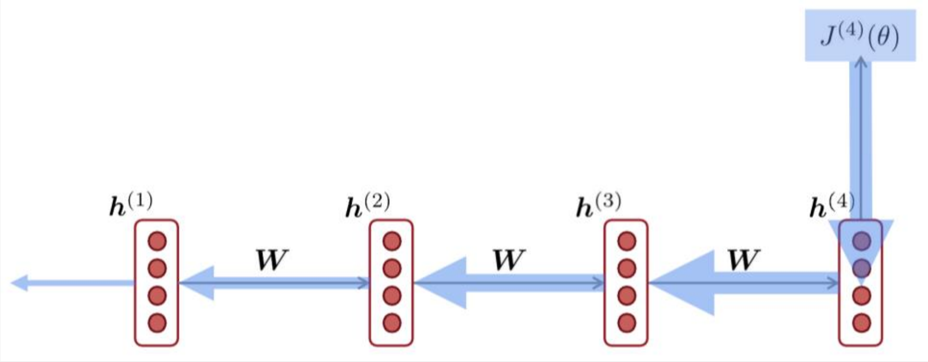
- Encoder–decoder models consist of two RNNs:
  - The encoder (green) processes the input into a vector of contextual info
  - The decoder (orange) processes the encoding to generate the output one word at a time



Encoder–decoder models were first introduced for machine translation, but they've also proven themselves effective at other tasks

TWO BIG PROBLEMS

- The <u>bottleneck problem</u>: the encoding holds context for the *whole* input, but not all info is equally relevant
- The <u>vanishing gradient problem</u>: as changes propagate backward through time, the signal can get weaker and weaker
- We need a way to let the model focus on specific parts of the input, even those processed far back in time



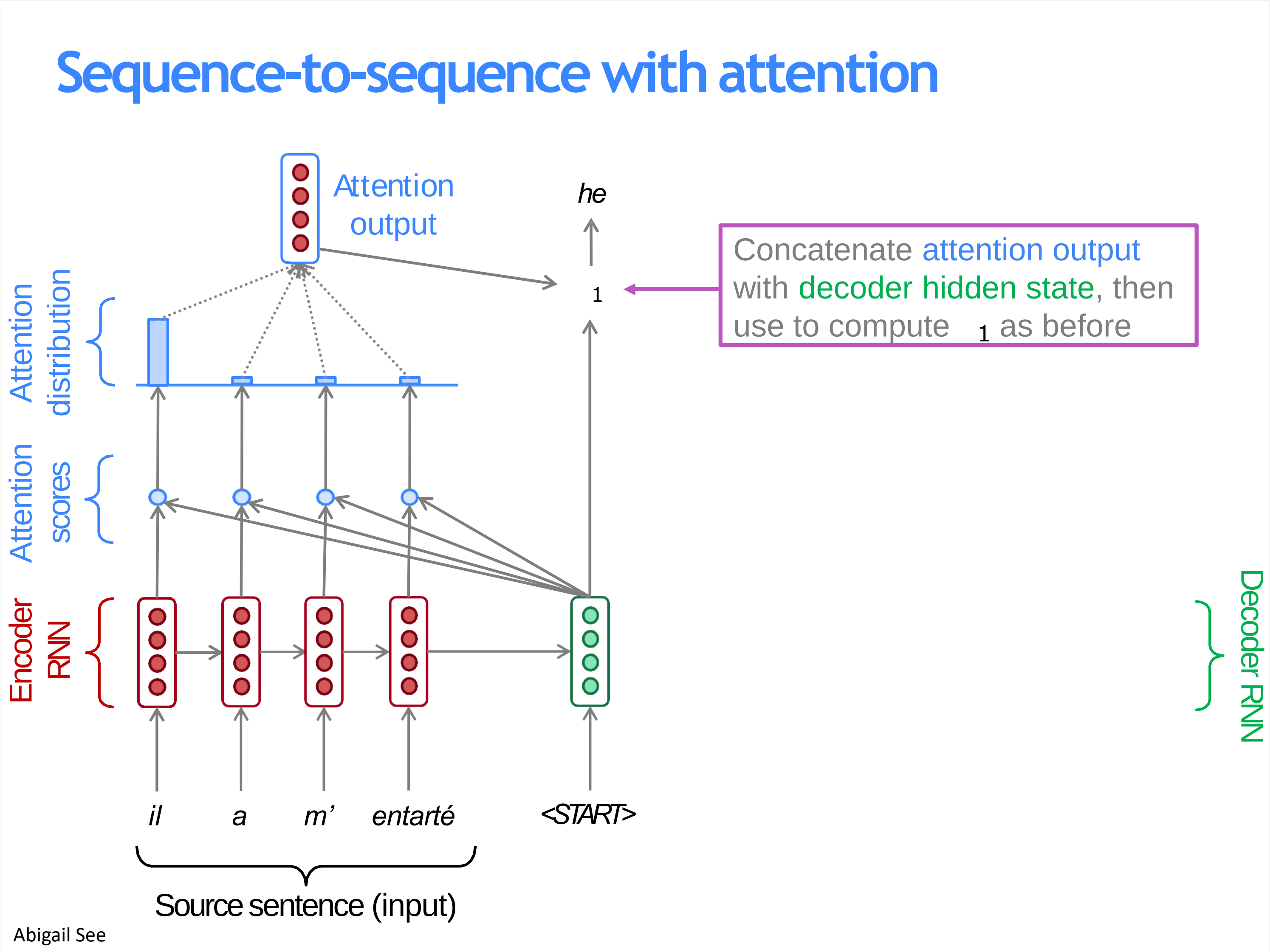
Gradients can shrink to 0 as it propagates through more and more layers

ATTENTION!

- <u>Attention</u> is a mechanism that allows a model to directly focus on parts of the input
  - Attention mechanisms take a vector of values (e.g., hidden states in the encoder) and weights them based on a *query* vector (e.g., hidden states in the decoder)
  - At every step, the query can access info for *all* inputs, focusing on the most important parts

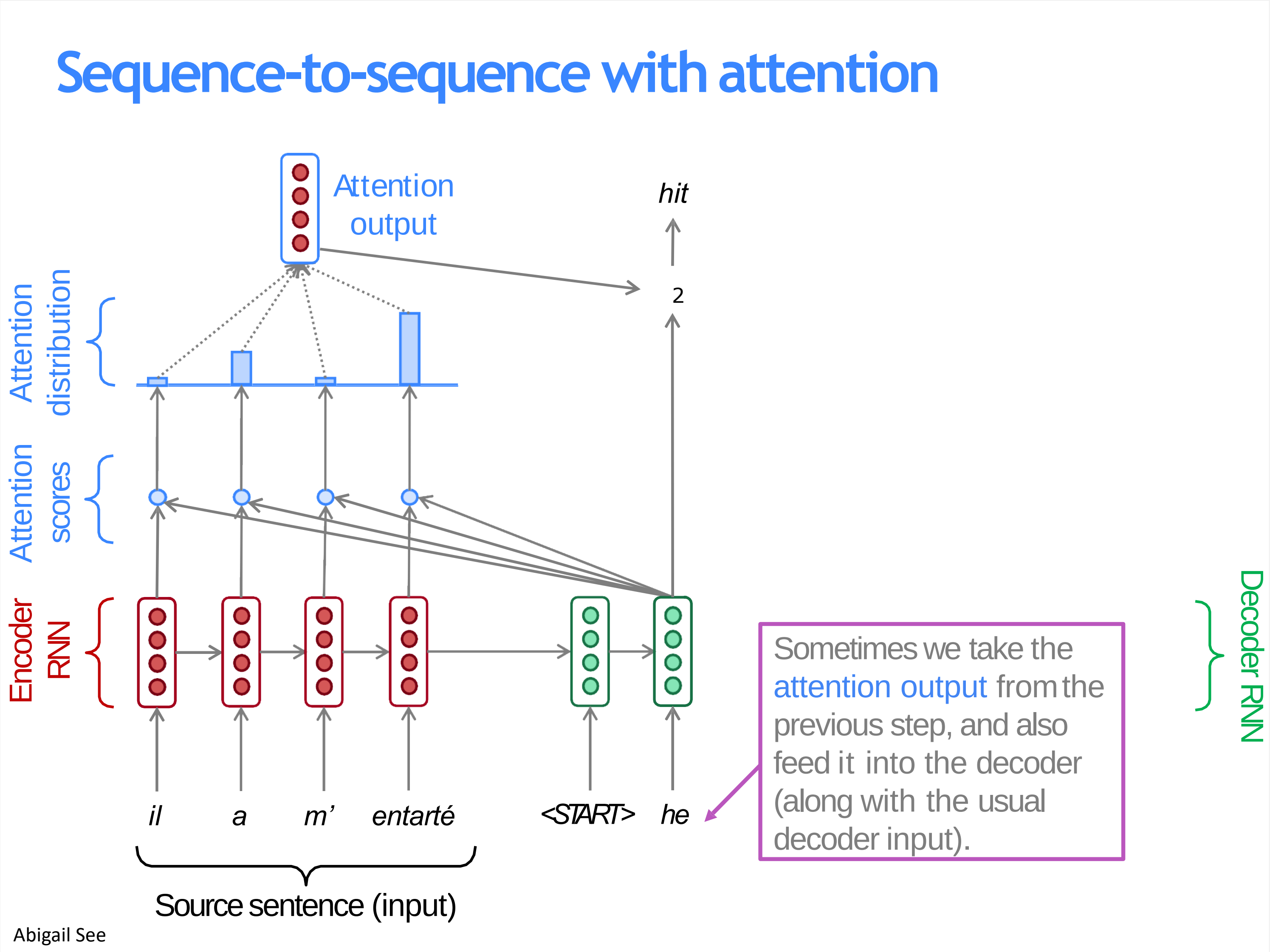
PAYING ATTENTION

- Pass inputs through the encoder and produce the context vector
- Calculate attention scores for each input
- Weight attention scores based on the context vector



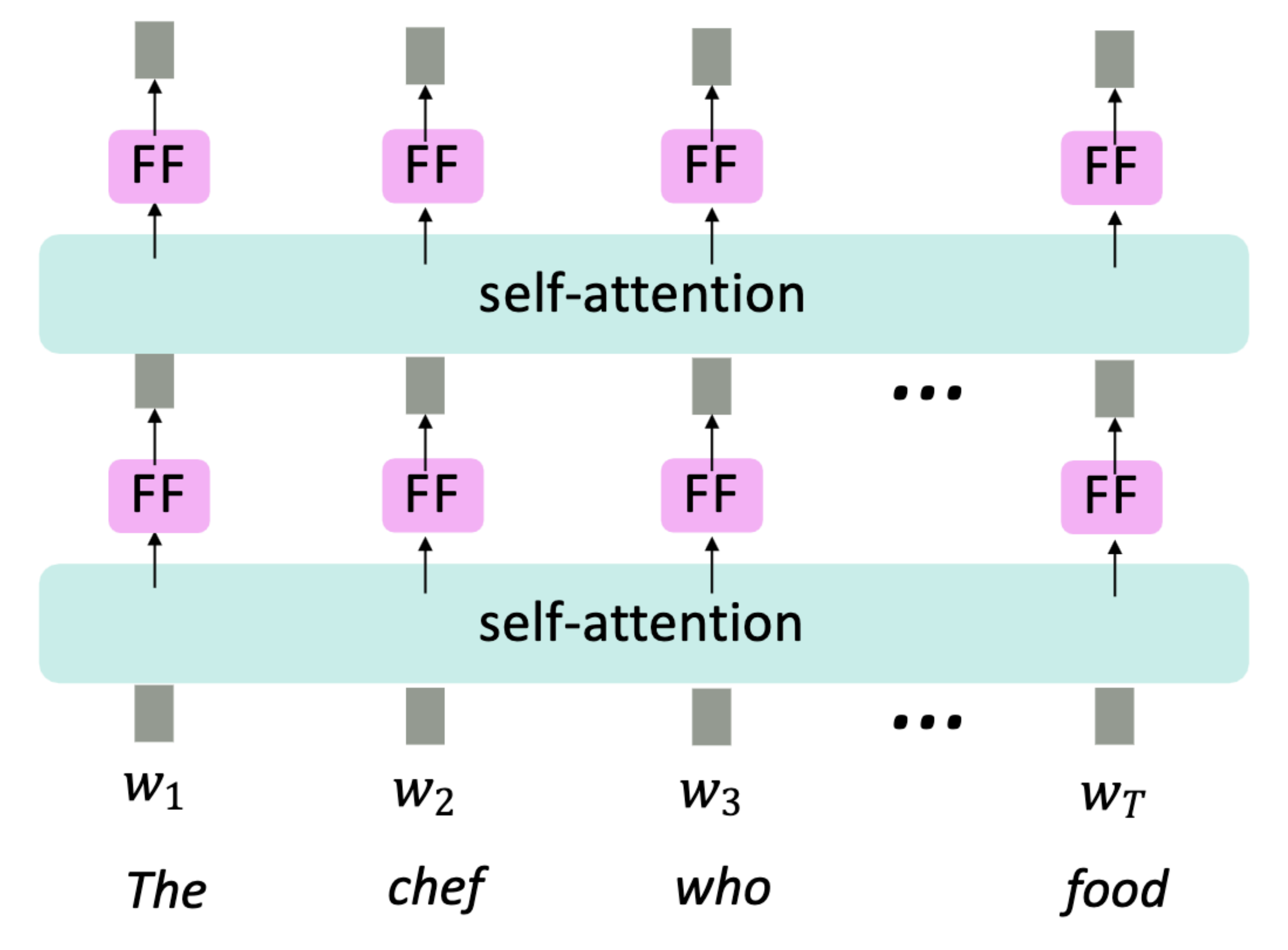
PAYING ATTENTION (CONT.)

- Repeat for every step of the decoder
- This makes it so that the decoder focuses on relevant parts of the input at every step



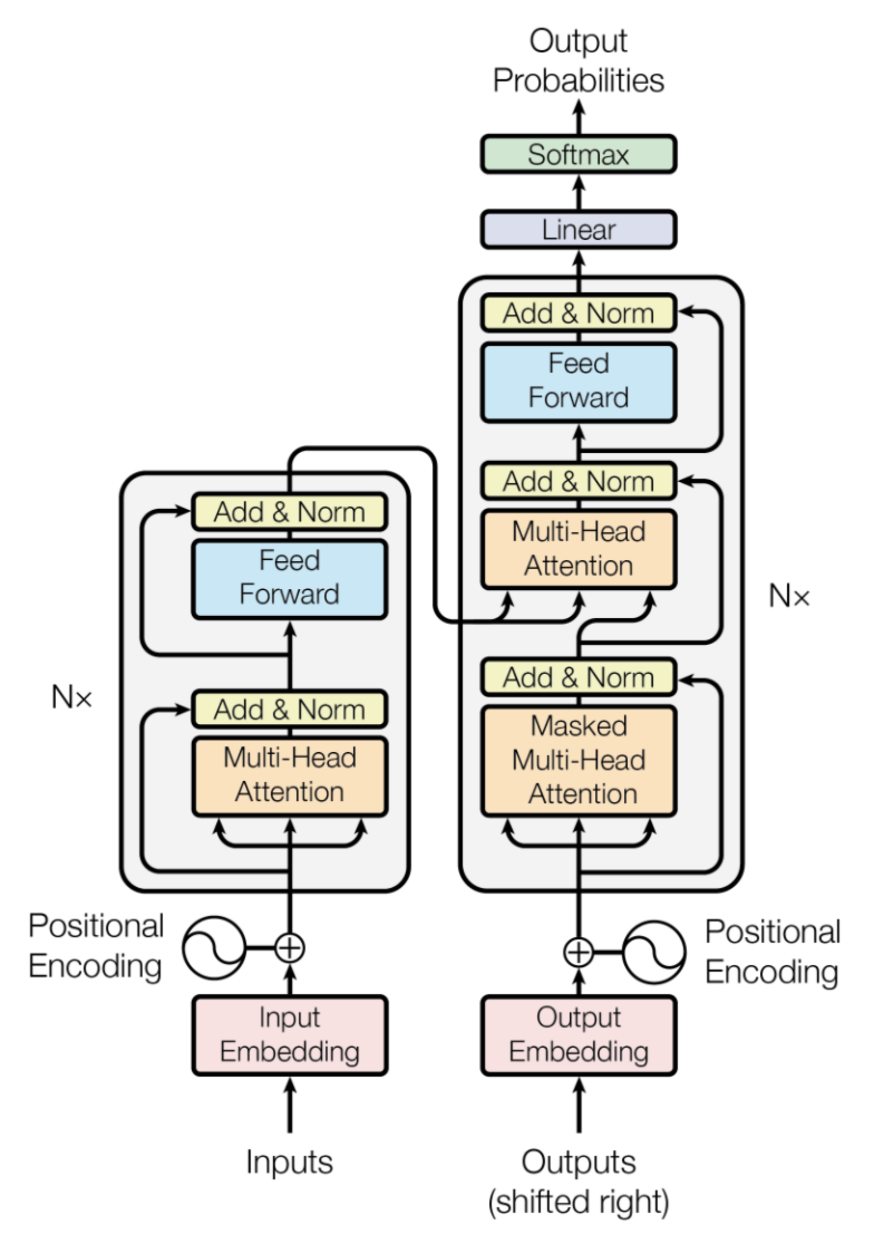
APPLYING ATTENTION TO LMS

- Self-attention allows us to model dependencies within the input itself
- The inputs are processed by a self-attention layer, and then each processed by a vanilla NN
  - *Note that the input size is fixed again!*



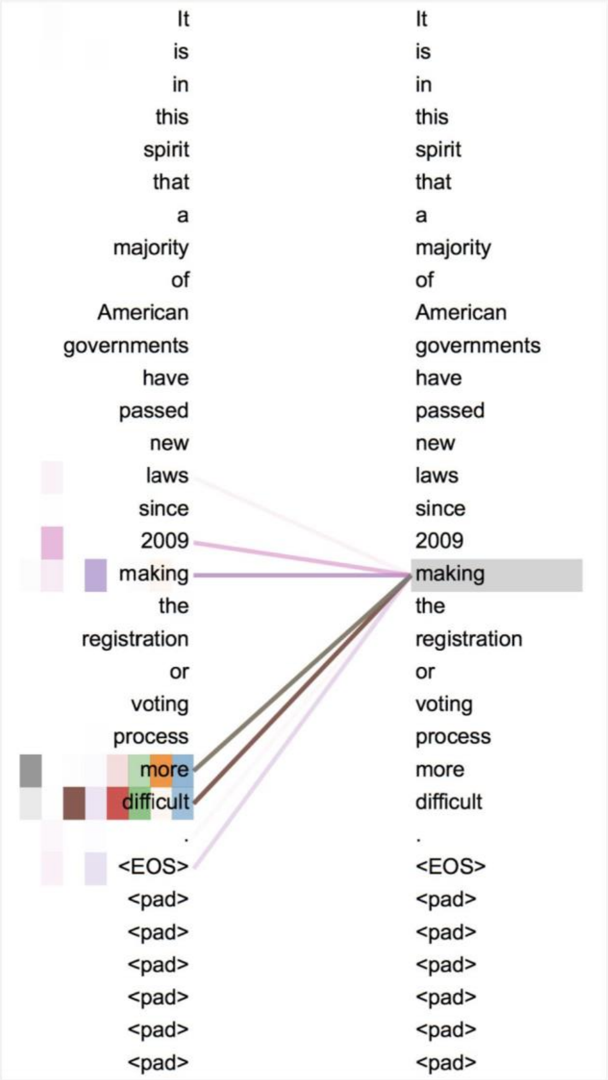
TRANSFORMERS (THE BORING KIND)

- The transformer was introduced in OpenAI's now-famous paper "Attention is All You Need" (Vaswani et al, 2017)
  - The trick: give up on RNNs completely for the encoder–decoder model
  - Instead, have an encoder–decoder model of *only* attention mechanisms

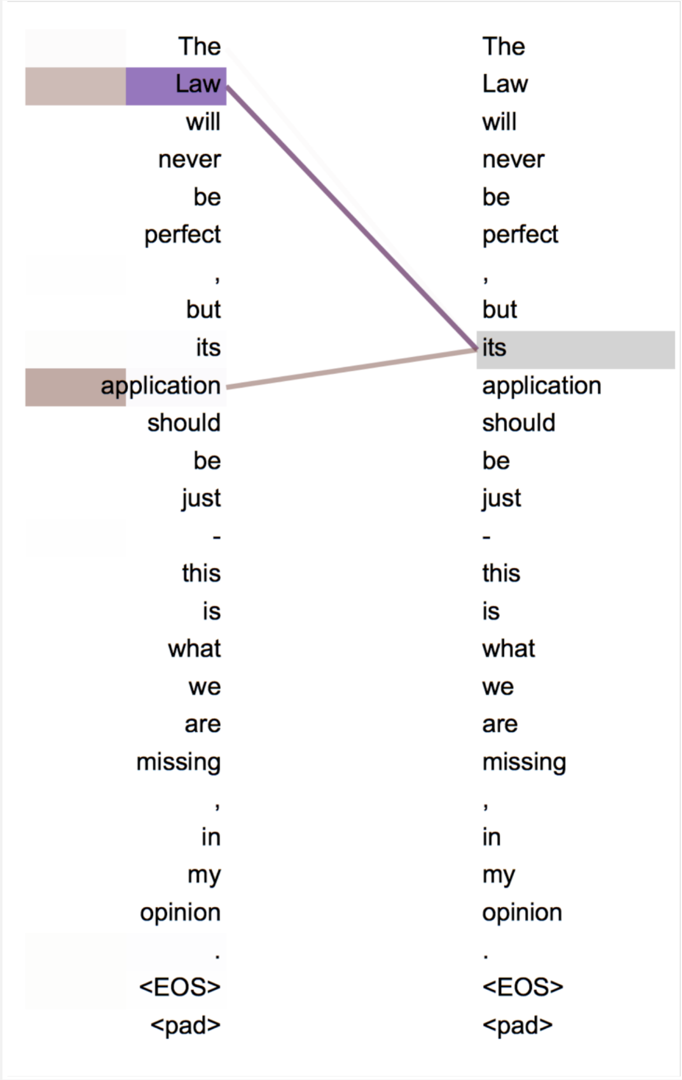


ATTENTION SEEMS TO WORK

There is still a big tradeoff with this setup. What is it?

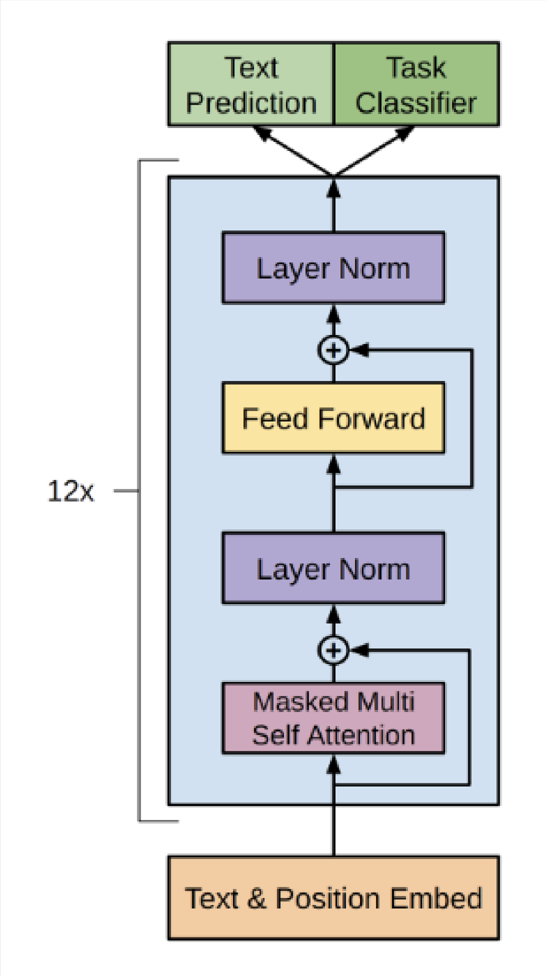


Attention is highest for sensible word relationships



Potential for anaphora resolution!

CHATGPT IS BUILT USING TRANSFORMERS



Transformer architecture diagram from OpenAI's paper introducing GPTs

AND ALL OF THIS… JUST TO PREDICT THE NEXT WORD

Now you just need a quadspillion dollars of
VC funding to make your very own ChatGPT!

Thank you!

REFERENCES & FURTHER READING

- *Speech and Language Processing*, 3rd ed draft, Daniel Jurafsky & James H. Martin
- Wikipedia (pretty much every keyword in this presentation has its own article)
- Na-Rae's lecture slides (see Lecture 7 for discussion on n-gram LMs)
- YouTube series on neural networks by Grant Sanderson (aka 3Blue1Brown)
- Lecture slides for CS 1678 (Intro to Deep Learning) by Dr. Adriana Kovashka
- "Effective Approaches to Attention-based Neural Machine Translation" (Luong, Pham, and Manning, 2015), which introduced attention mechanisms for machine translation
- "Attention is All You Need" (Vaswani et al, 2017), which introduced the concept of transformer models
- OpenAI's 2018 paper "Improving Language Understanding by Generative Pre-Training", which introduced GPT, and its accompanying article