CS 760: Machine Learning
Large Language Models

Josiah Hanna

University of Wisconsin-Madison

October 24, 2023

Announcements

- **Midterm evaluations**
  - **X%** have already filled out. Thank you!
  - Please fill out if you haven't already

- **Homework 4** due in one week

Outline

- **Finish RNNs (LSTM model)**
- **Language Models & NLP**
  - Word embeddings, attention
- **Transformer Model**
  - Properties, architecture breakdown
- **Transformer-based Models**
  - BERT, GPTs, Foundation Models

Outline

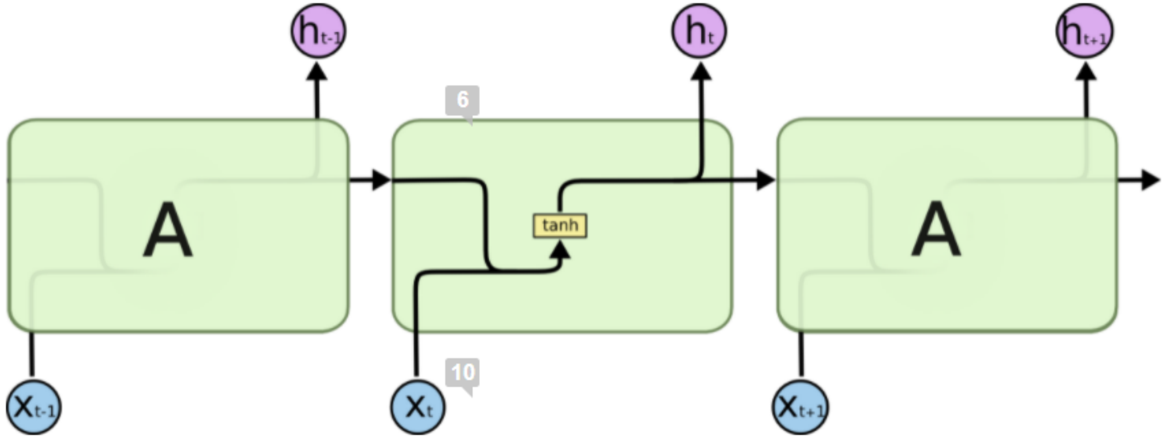
- **Finish RNNs (LSTM model)**
- **Language Models & NLP**
  - Word embeddings, attention
- **Transformer Model**
  - Properties, architecture breakdown
- **Transformer-based Models**
  - BERT, GPTs, Foundation Models

RNN Problems with long sequences

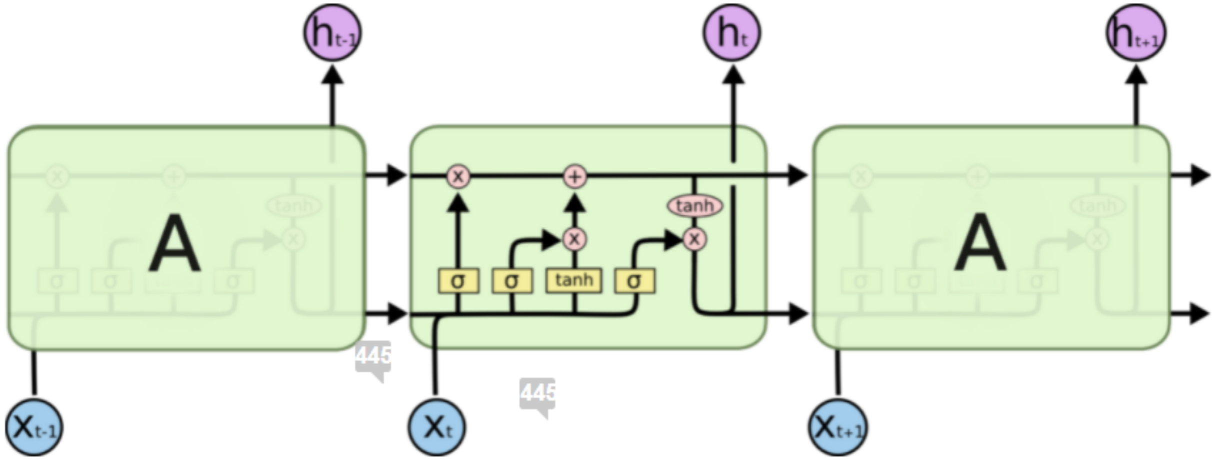
- **Training:** What happens to gradients in backprop with many layers?
  - In an RNN trained on long sequences (*e.g.* 100 time steps) the gradients can easily explode or vanish.
  - We can avoid this by initializing the weights very carefully.

- **Memory/retention:** very hard to detect that current target output **depends** on an input from long ago.
  - Simple RNNs have difficulty dealing with long-range dependencies.

- RNN: can write structure as:

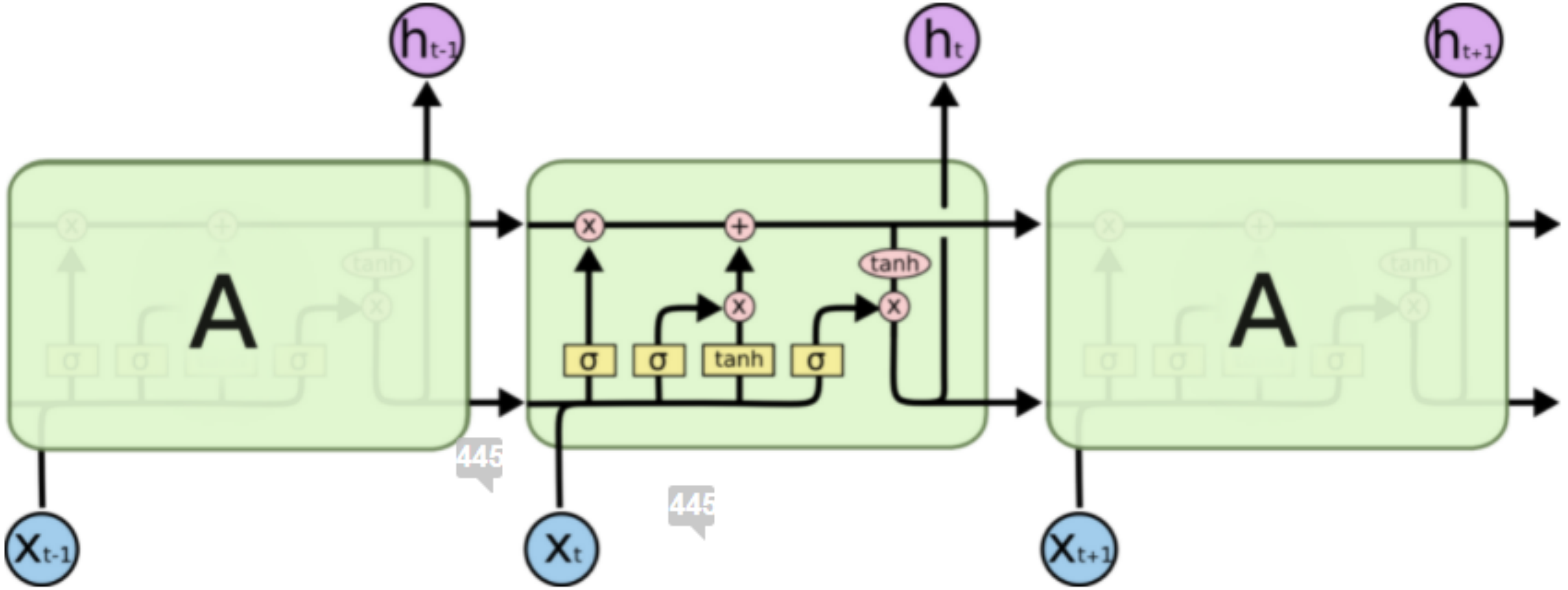


- Long Short-Term Memory Cell: deals with problem of long-term dependencies.



LSTM (Long Short-Term Memory)

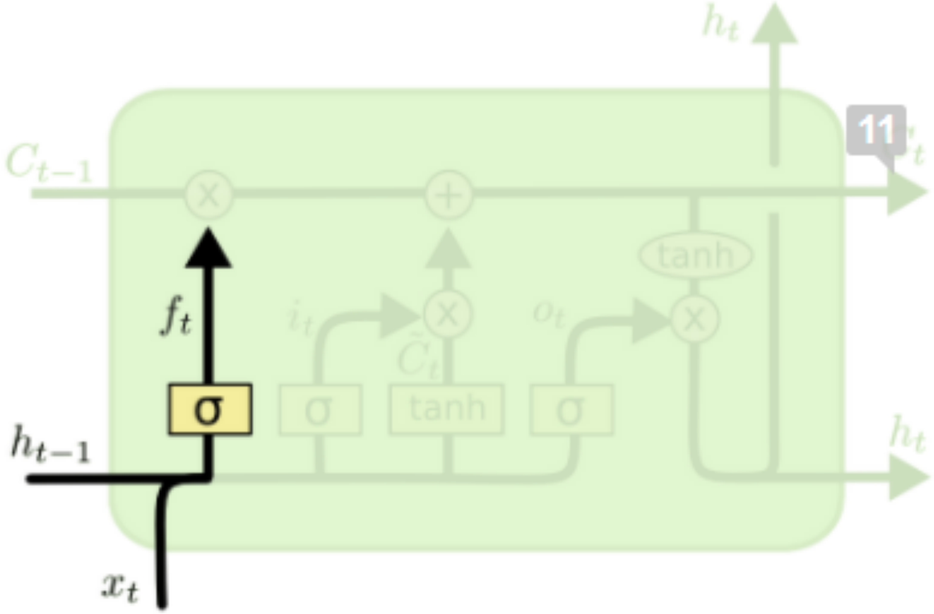
- Long Short-Term Memory Cell: deals with problem.



- h_t: short term memory
- C_t: long term memory

Understanding the LSTM Cell

- Step-by-step
  - Good reference: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

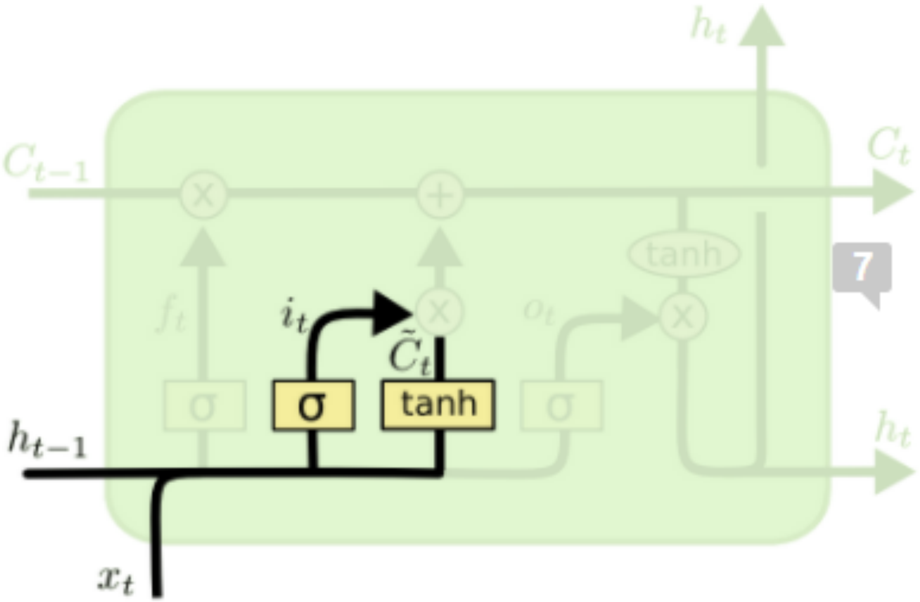


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

- "**Forget**" gate.
  - Can remove all or part of any entry in cell state C
  - σ denotes the sigmoid (logistic) activation; think of this as a soft on/off function

Understanding the LSTM Cell

- Step-by-step



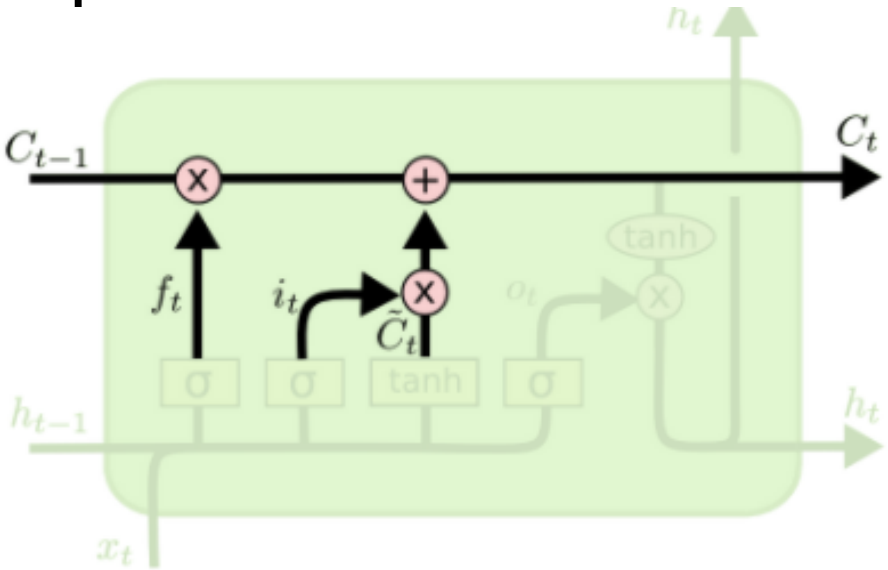
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

- **Input** gate. Combine:
  - What entries in C_{t-1} we'll update
  - Candidates for updating: $\tilde{C}_t$
  - Add information to cell state C_{t-1} (post-forgetting)

Understanding the LSTM Cell

- Step-by-step

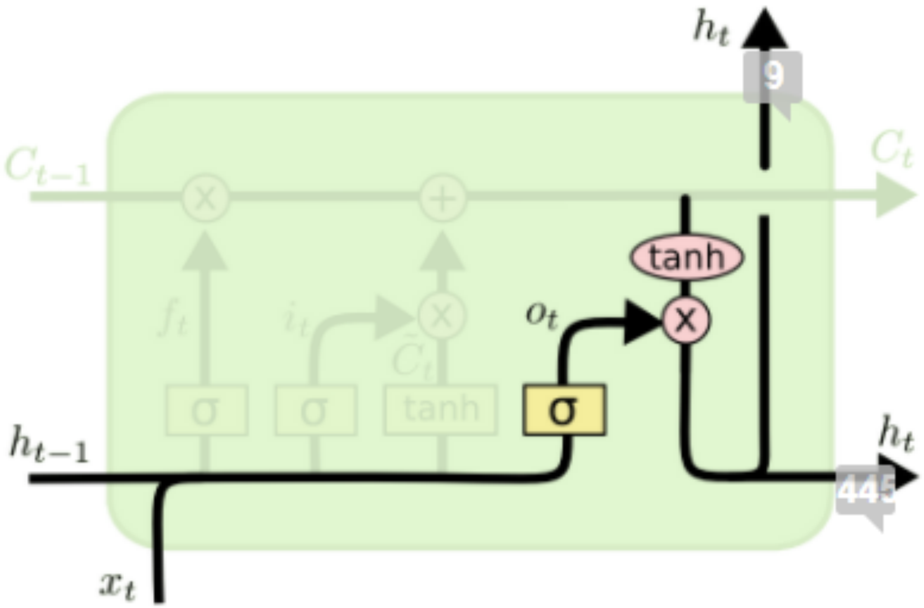


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

- Updating C_{t-1} to C_t
  - Forget, then
  - Add new information

Understanding the LSTM Cell

- Step-by-step



$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

- **Output** gate
  - Combine hidden state, input as before, but also
  - Modify according to cell state C_t

Outline

- **Finish RNNs (LSTM model)**
- **Language Models & NLP**
  - Word embeddings, attention
- **Transformer Model**
  - Properties, architecture breakdown
- **Transformer-based Models**
  - BERT, GPTs, Foundation Models

Language Models

- Basic idea: use probabilistic models to **assign a probability to a sentence:**

$$P(W) = P(w_1, w_2, \ldots, w_n) \text{ or } P(w_{\text{next}} | w_1, w_2 \ldots)$$

- Goes back to Claude Shannon
  - "Father of Information Theory"
  - Information theory: letters

Zero-order approximation	XFOML RXKHRJFFJUJ ALPWXFWJXYJ FFJEYVJCQSGHYD QPAAMKBZAACIBZLKJQD
First-order approximation	OCRO HLO RGWR NMIELWIS EU LL NBNESEBYA TH EEI ALHENHTTPA OOBTTVA NAH BRL
Second-order approximation	ON IE ANTSOUTINYS ARE T INCTORE ST BE S DEAMY ACHIN D ILONASIVE TUCOOWE AT TEASONARE FUSO TIZIN ANDY TOBE SEACE CTISBE
Third-order approximation	IN NO IST LAT WHEY CRATICT FROURE BIRS GROCID PONDENOME OF DEMONSTURES OF THE REPTAGIN IS REGOACTIONA OF CRE
First-order word approximation	REPRESENTING AND SPEEDILY IS AN GOOD APT OR COME CAN DIFFERENT NATURAL HERE HE THE A IN CAME THE TO OF TO EXPERT GRAY COME TO

Language Models: Word Embeddings

- One way to encode words: one-hot vectors
  - Does not capture word similarity. Want something smarter…

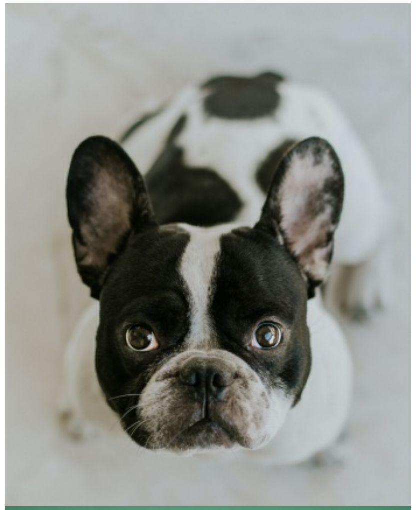
Distributional semantics: account for relationships

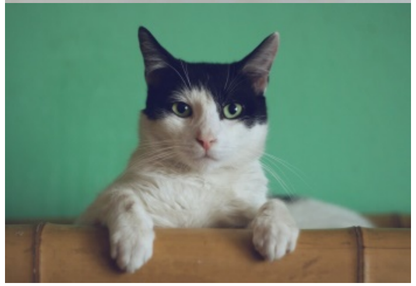
- Representations should be close/similar to other words that appear in a similar context

Dense vectors:

$$\text{dog} = \begin{bmatrix} 0.13 & 0.87 & -0.23 & 0.46 & 0.87 & -0.31 \end{bmatrix}^T$$

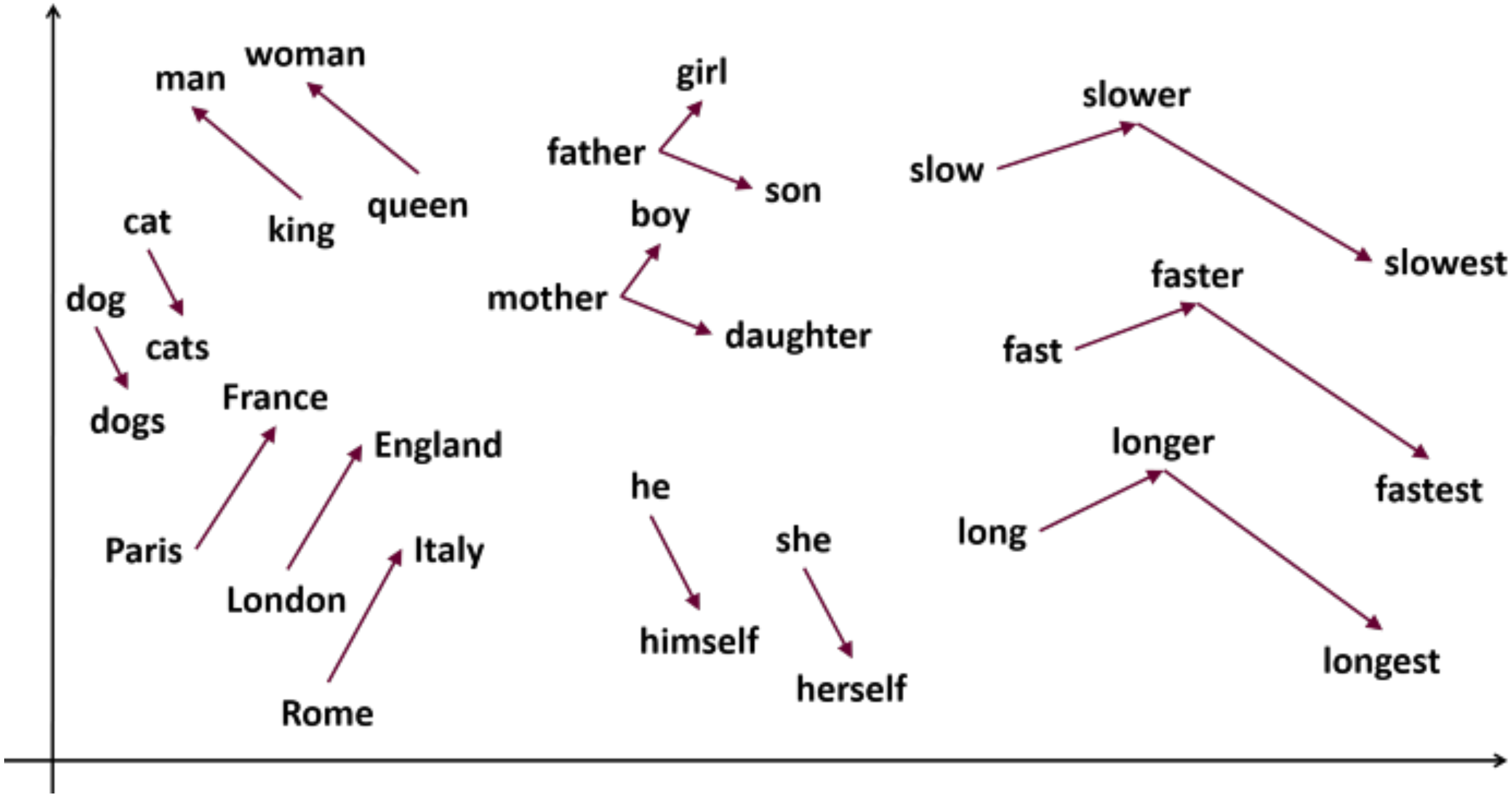
$$\text{cat} = \begin{bmatrix} 0.07 & 1.03 & -0.43 & -0.21 & 1.11 & -0.34 \end{bmatrix}^T$$





AKA **word embeddings**

Word Embeddings



[Saurabh Pal – Implementing Word2Vec in Tensorflow]()

Training **Word Embeddings**

Many approaches (very popular 2010-present)

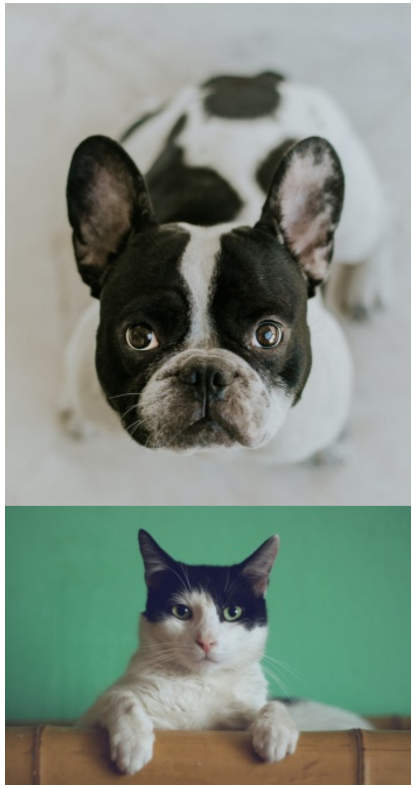
- Word2vec: a famous approach
- Write out a likelihood

Windows of length 2a

$$L(\theta) = \prod_{t=1}^{T} \prod_{-a \leq j \leq a} P(w_{t+j} | w_t, \theta)$$

Our word vectors (weights)

All positions

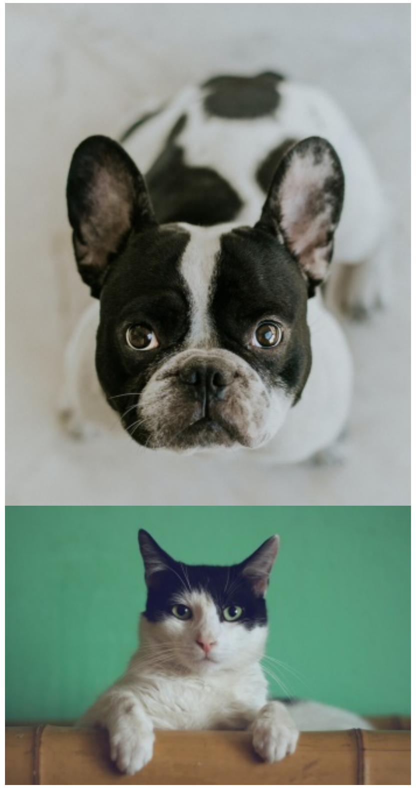


Training **Word Embeddings**

Word2vec likelihood
$$L(\theta) = \prod_{t=1}^{T} \prod_{-a \leq j \leq a} P(w_{t+j} | w_t, \theta)$$

- Expression for the probability:

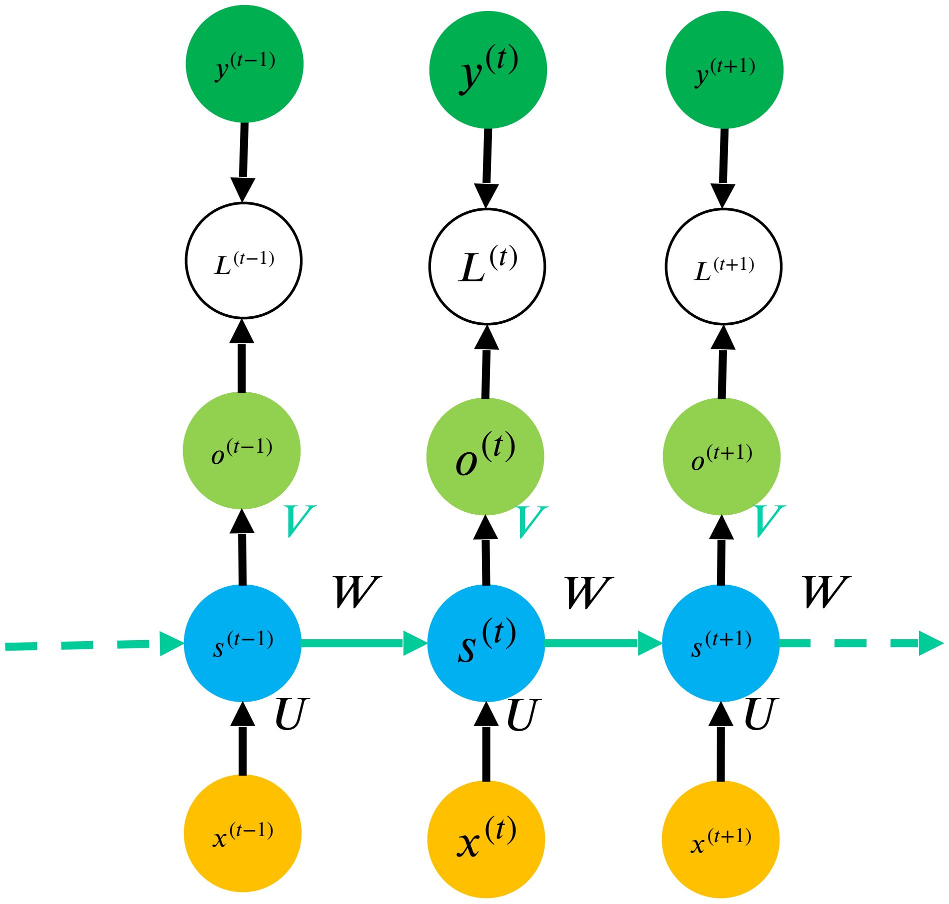
$$P(w' | w, \theta) = \frac{\exp((\theta_{w',o})^\top \theta_{w,c})}{\sum_{v \in V} \exp((\theta_{v,o})^\top \theta_{w,c})}$$

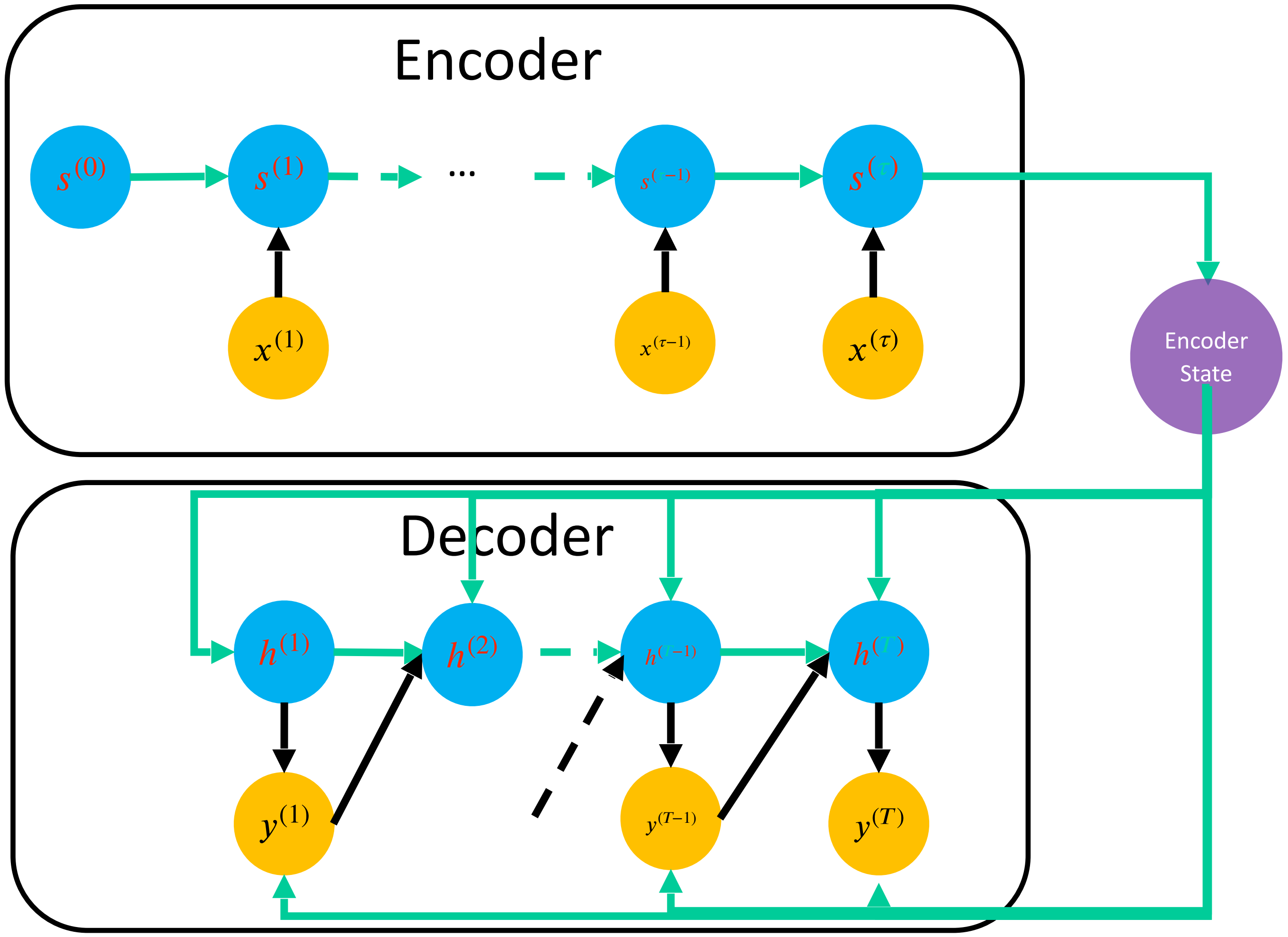


  - $\theta_{w,o}$: occurrence vector for word w
  - $\theta_{w,c}$: context vector for word w

Language Models: RNN Review

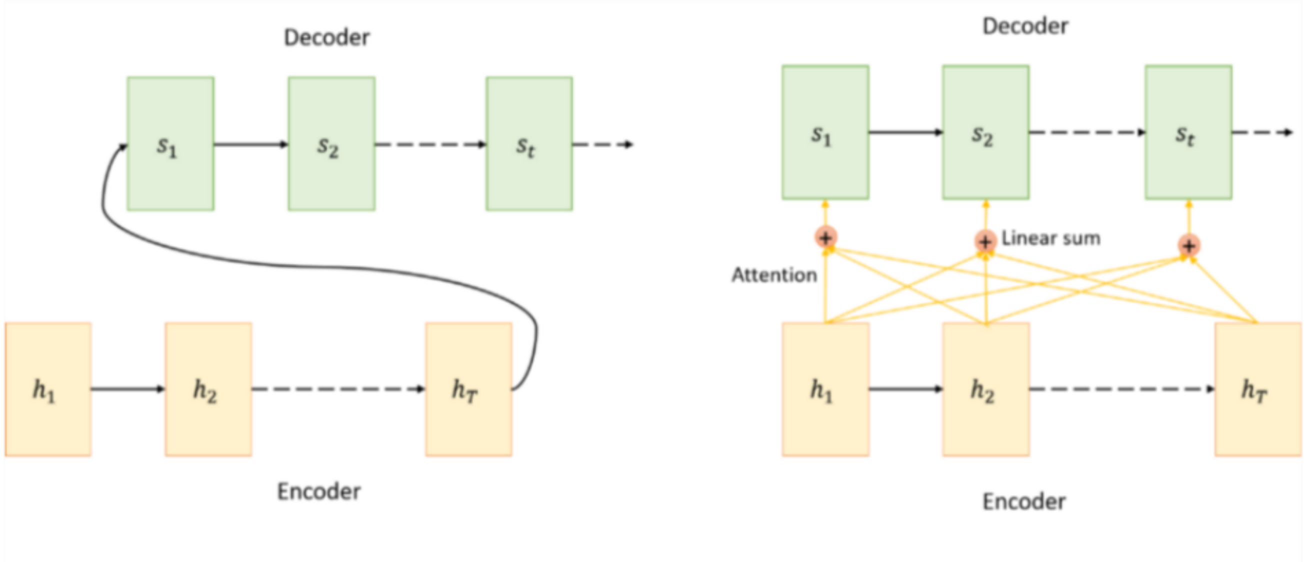
- Classical RNN model / Encoder-Decoder variant:





Language Models: Attention

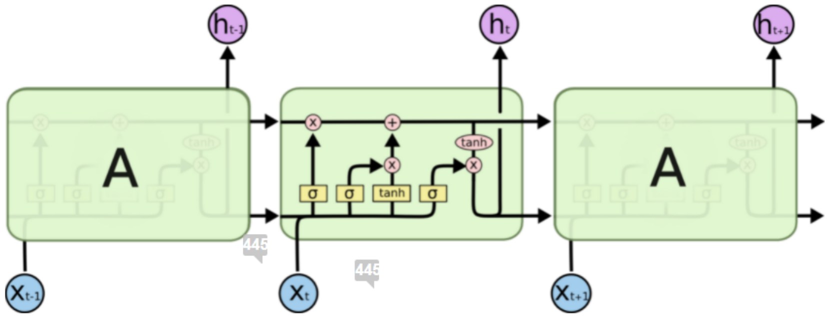
- One challenge: dealing with the hidden state
  - Everything gets compressed there
  - Might lose information
- Solution: **attention** mechanism
  - Similar to residual connections in ResNets (not covered in lecture)



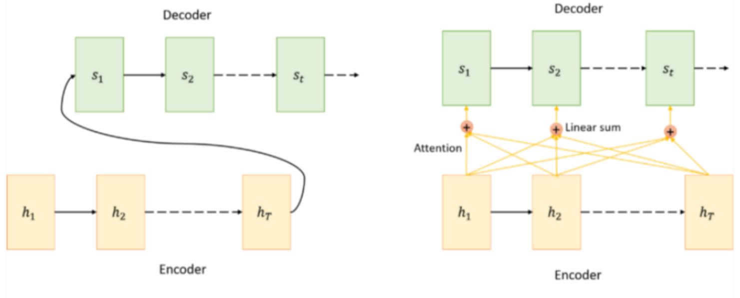
(a) Vanilla Encoder Decoder Architecture (b) Attention Mechanism

Language Models: Putting it All Together

- Before 2017: best language models
  - Use encoder/decoder architectures based on **RNNs** (LSTMs)
  - Use **word embeddings** for word representations
  - Use attention mechanisms



$$\text{dog} = \begin{bmatrix} 0.13 & 0.87 & -0.23 & 0.46 & 0.87 & -0.31 \end{bmatrix}^T$$



(a) Vanilla Encoder Decoder Architecture

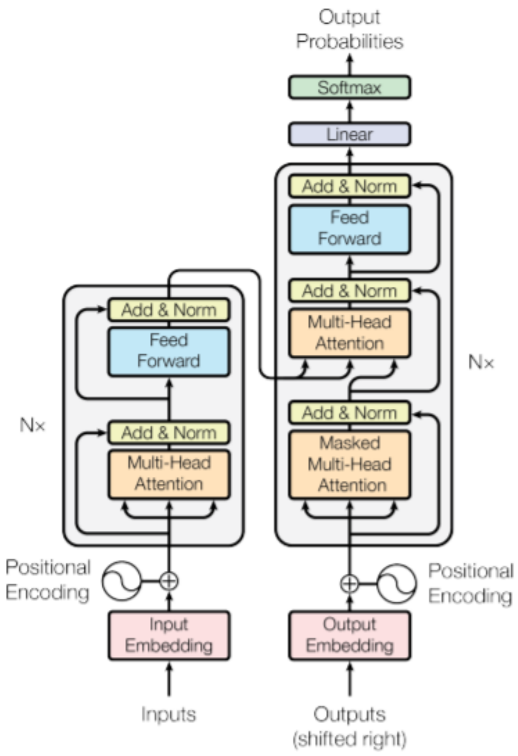
(b) Attention Mechanism

Outline

- **Language Models & NLP**
  - k-gram models, RNN review, word embeddings, attention
- **Transformer Model**
  - Properties, architecture breakdown
- **Transformer-based Models**
  - BERT, GPTs, Foundation Models

Transformers: Idea

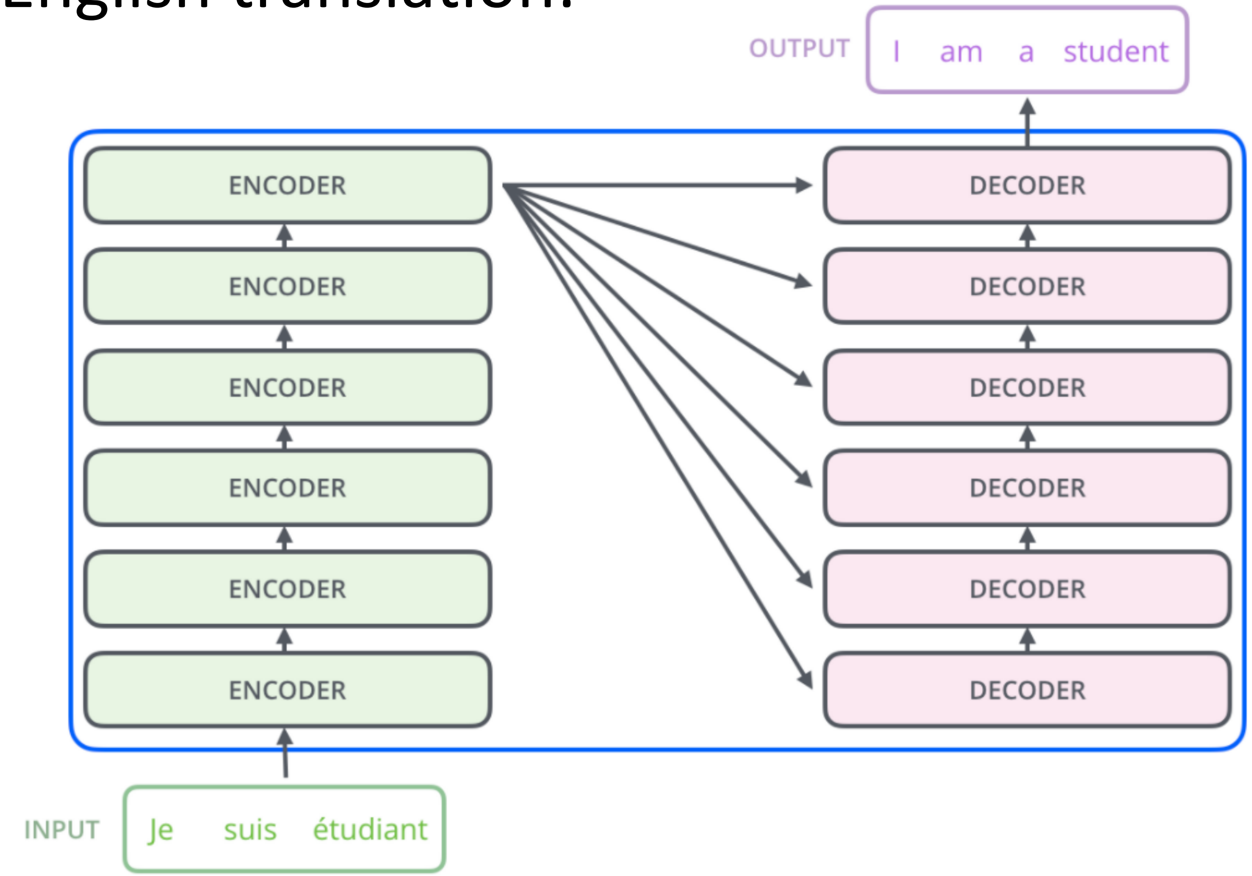
- Initial goal for an architecture: encoder-decoder
  - Get **rid of recurrence** (not good for parallelization)
  - Replace with **self-attention**



Vaswani et al. ‘17

Transformers: Architecture

- Sequence-sequence model with **stacked** encoders/decoders:
  - For example, for French-English translation:

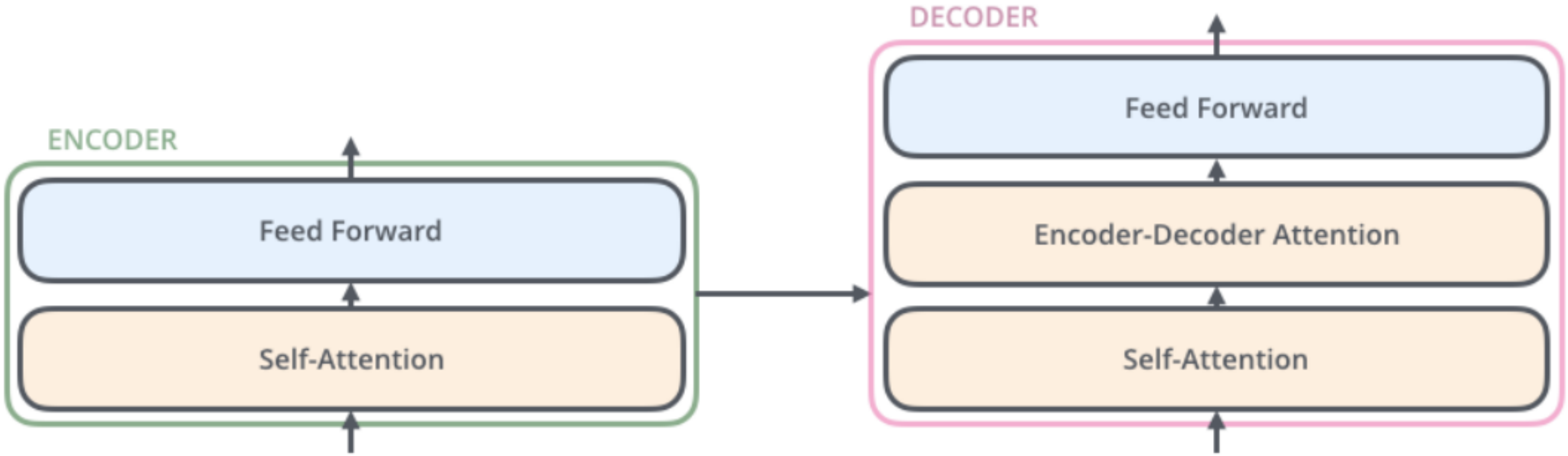


Note that entire sequence is passed at once; contrast with RNNs.

Excellent resource: https://jalammar.github.io/illustrated-transformer/

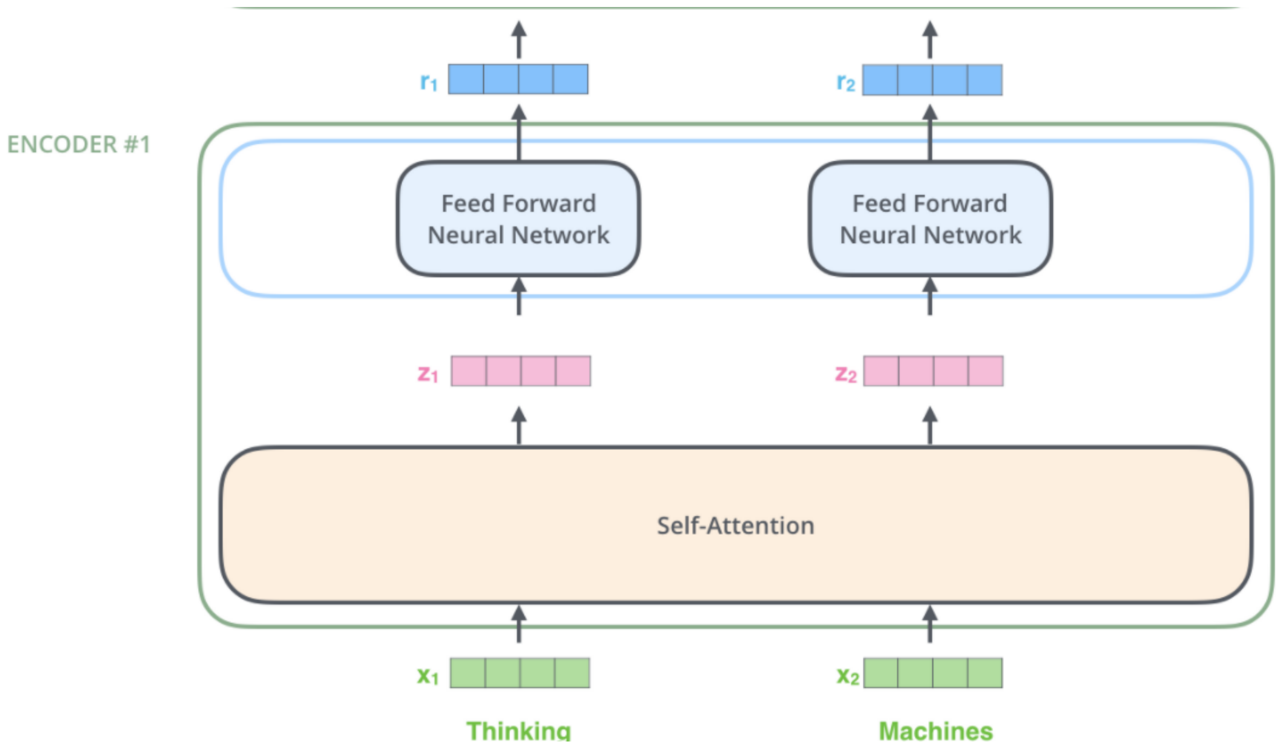
Transformers: Architecture

- Sequence-sequence model with **stacked** encoders/decoders:
  - What's inside each encoder/decoder unit?



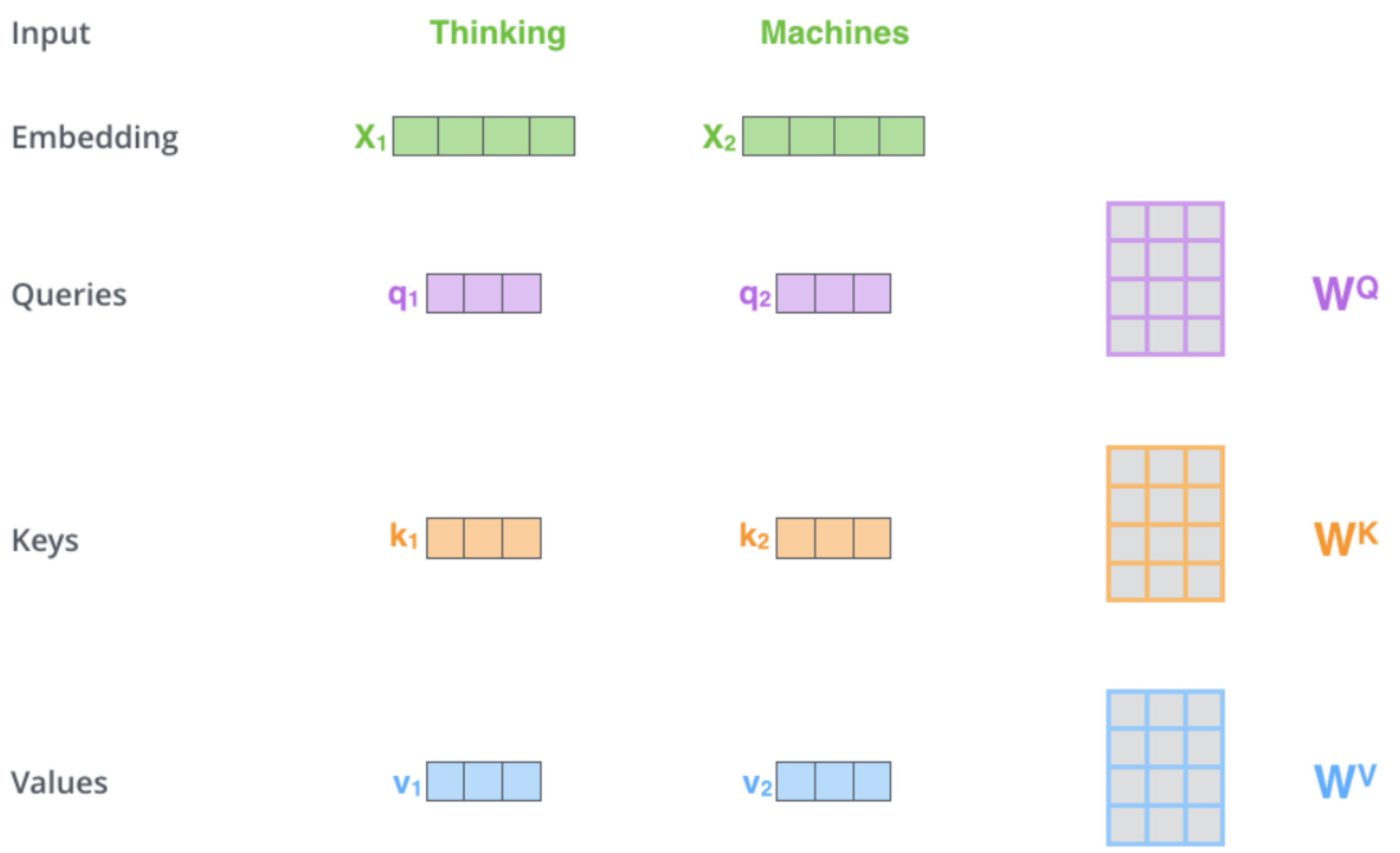
Transformers: Inside an Encoder

- Let’s take a look at the encoder. Two components:
  - 1. **Self-attention** layer
  - 2. **Feedforward nets**



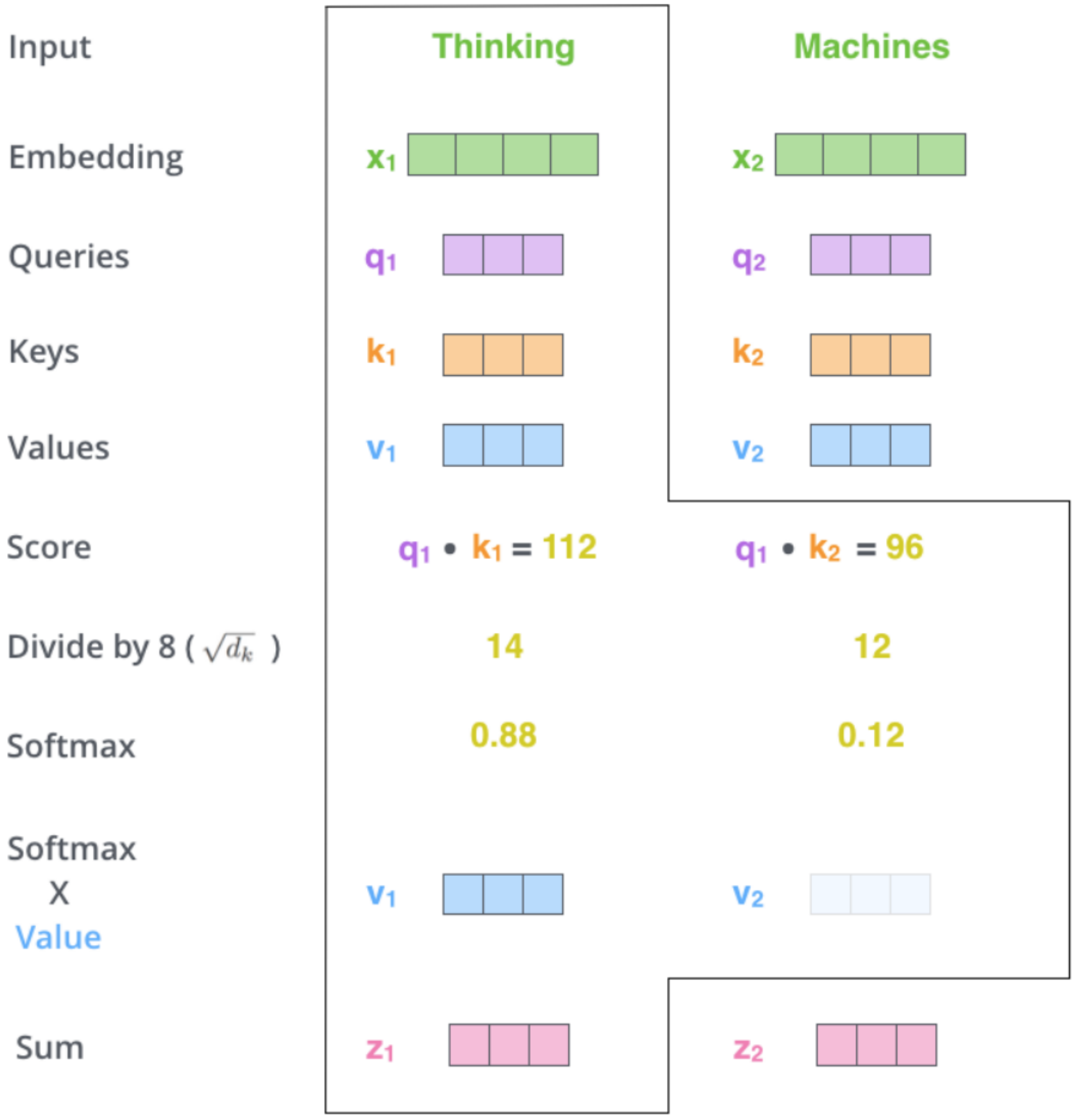
Transformers: Self-Attention

- Self-attention is the key layer in a transformer stack
  - Get 3 vectors for each embedding: **Query**, **Key**, **Value**



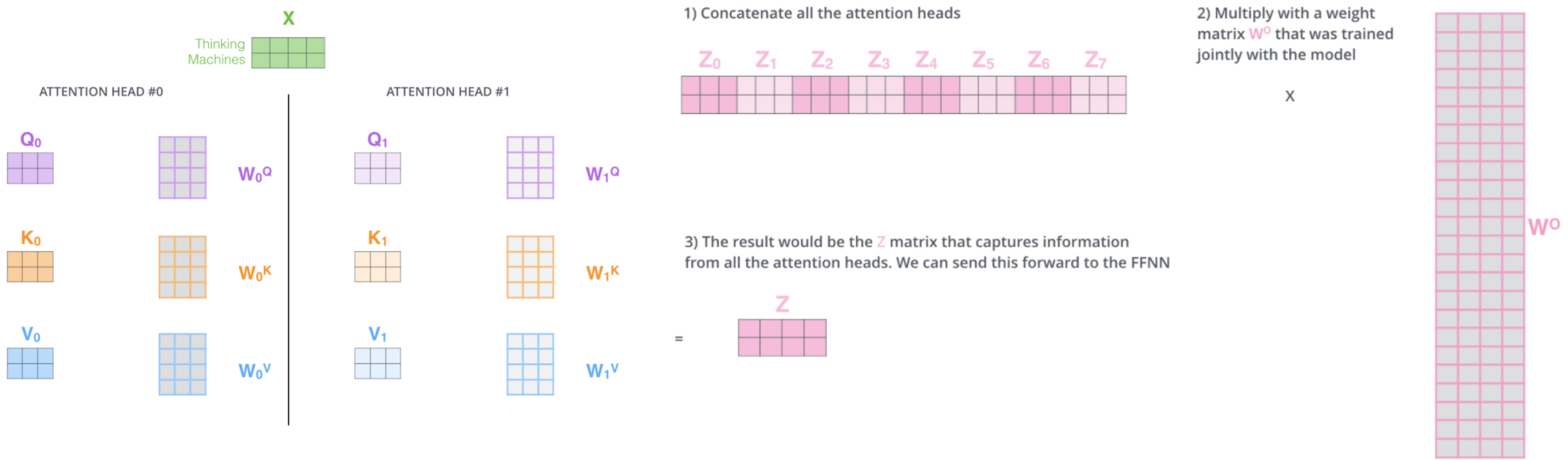
Transformers: Self-Attention

- Self-attention is the key layer in a transformer stack
  - Illustration. Recall the three vectors for each embedding: **Query**, **Key**, **Value**
  - The sum values are the outputs of the self-attention layer
  - Send these to feedforward NNs
- Highly parallelizable!



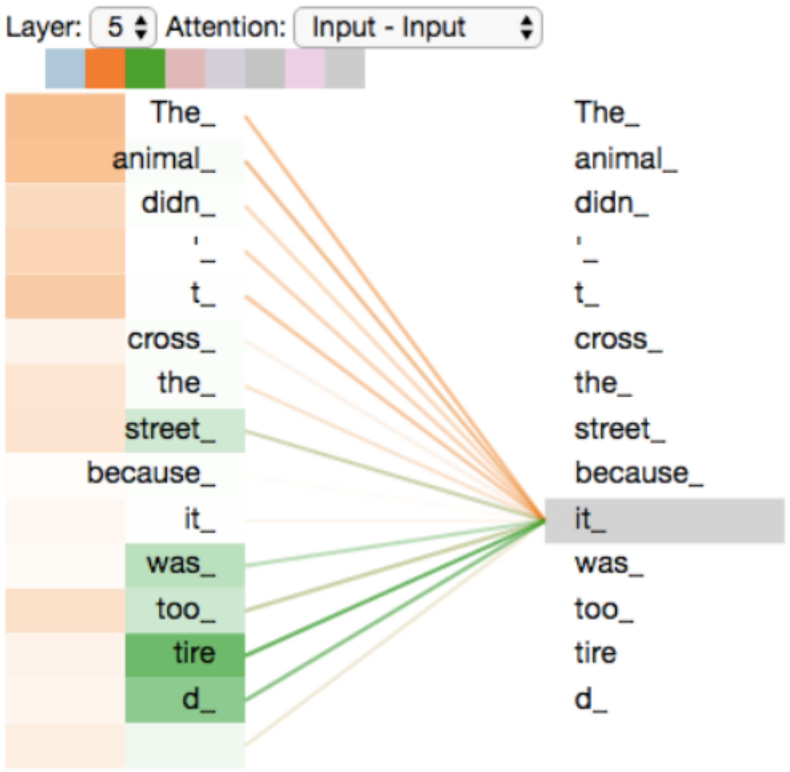
Transformers: Multi-Headed Attention

- We can do this multiple times in parallel
  - Called multiple heads
  - Need to combine the resulting output sums



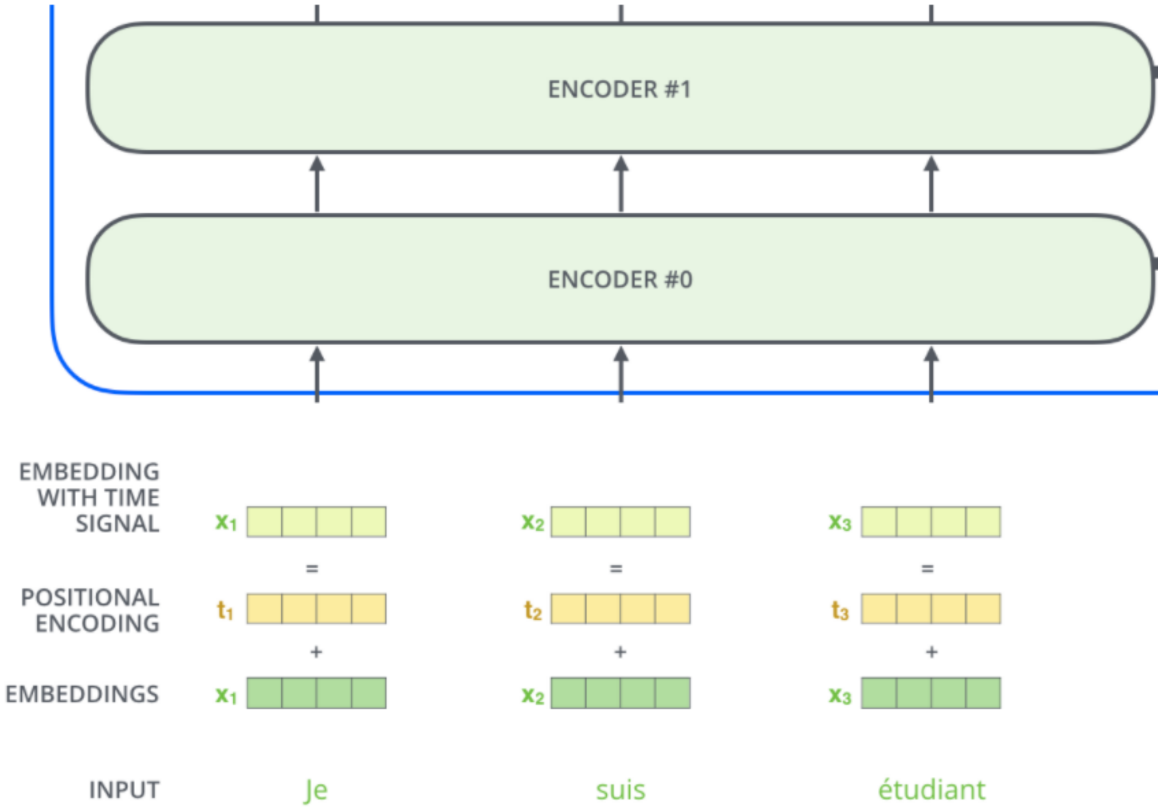
Transformers: Attention Visualization

- Attention tells us where to focus the information
  - Illustration for a sentence:



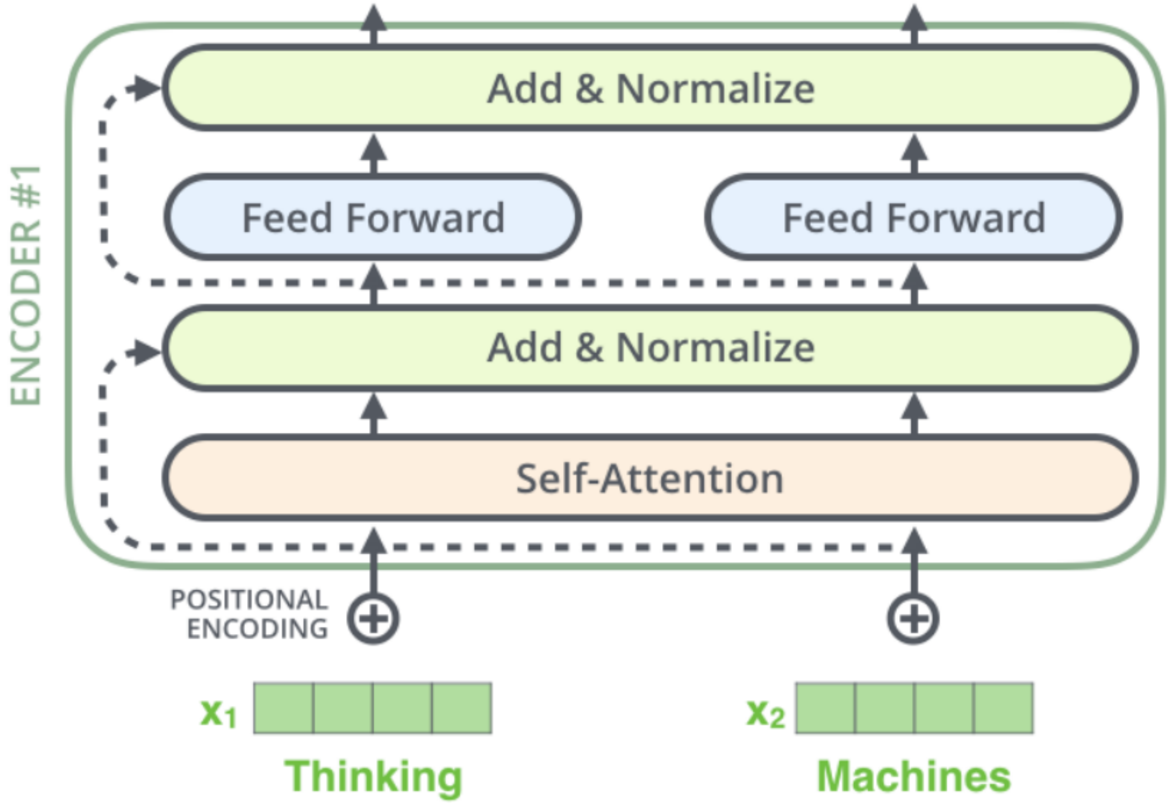
Transformers: Positional Encodings

- One thing we haven't discussed: the order of the symbols/ elements in the sequence
  - Add a vector containing a special positional formula's embedding



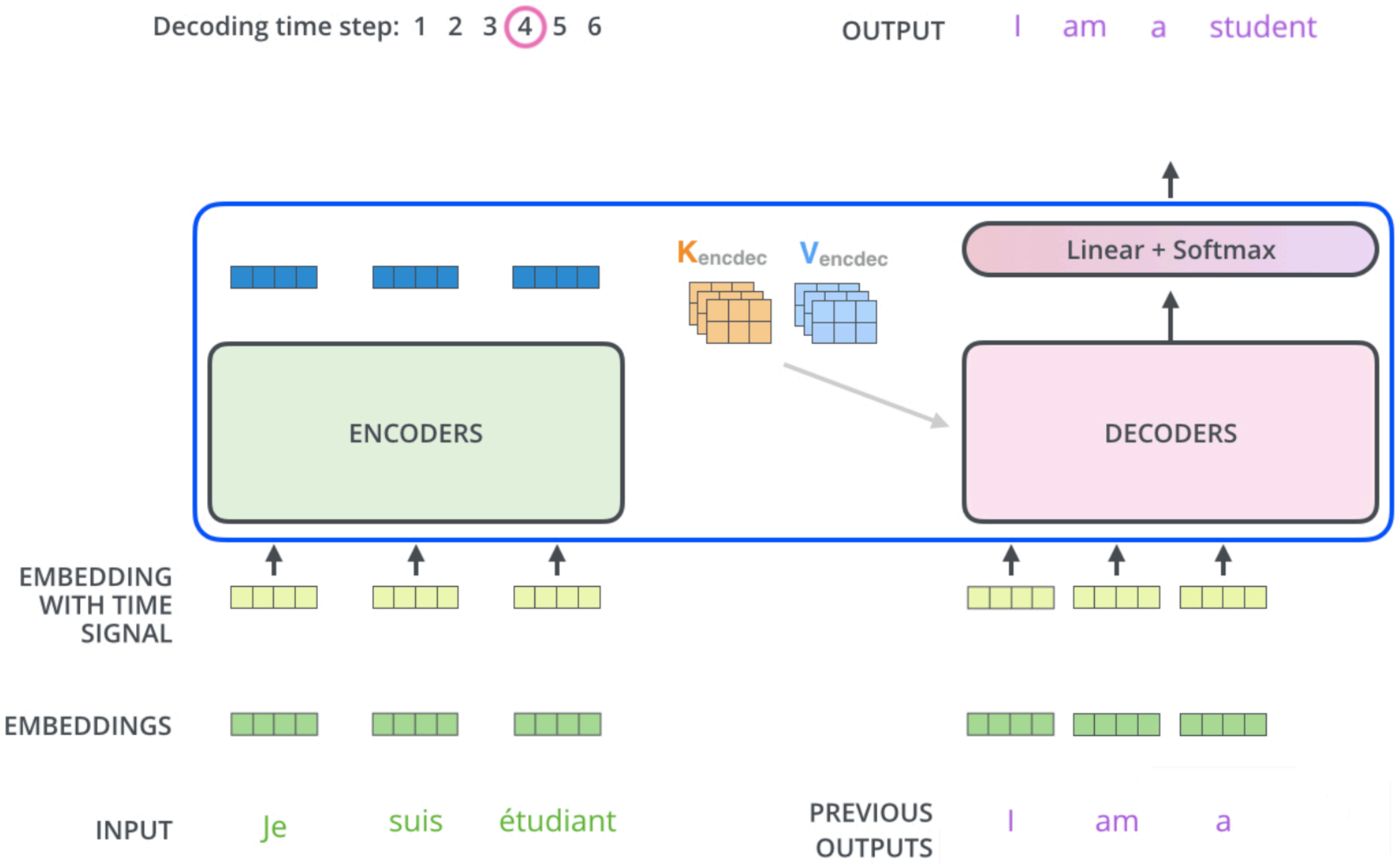
Transformers: More Tricks

- Residual connections:
  - Same idea as ResNets which enabled deeper CNNs.
  - And also layer normalizations
  - Apply to our encoder layers



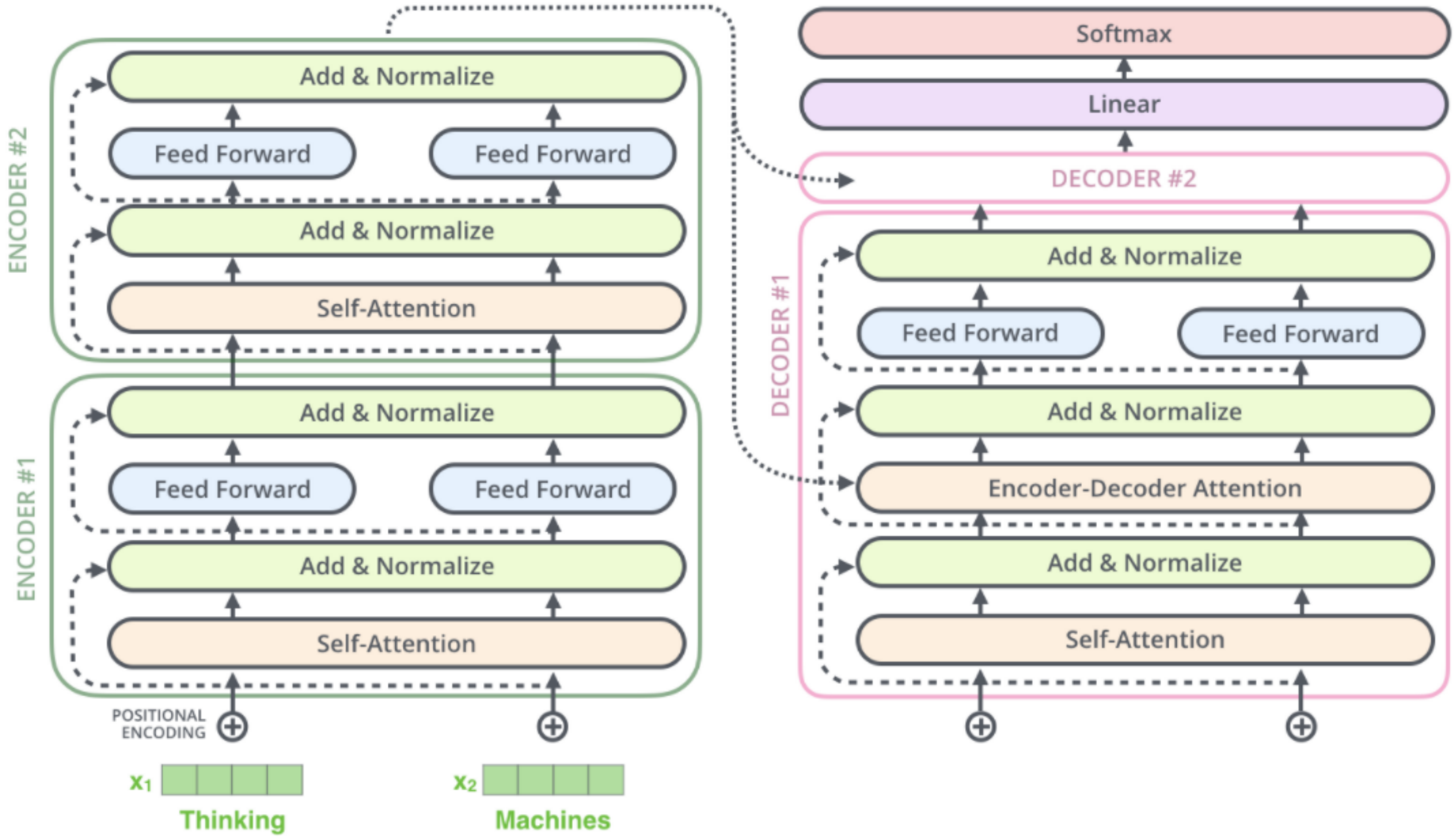
Transformers: Decoder

- Similar to encoders (see linked blog post for more details).
- E.g. Generating a translation



Transformers: Putting it All Together

- What does the full architecture look like?

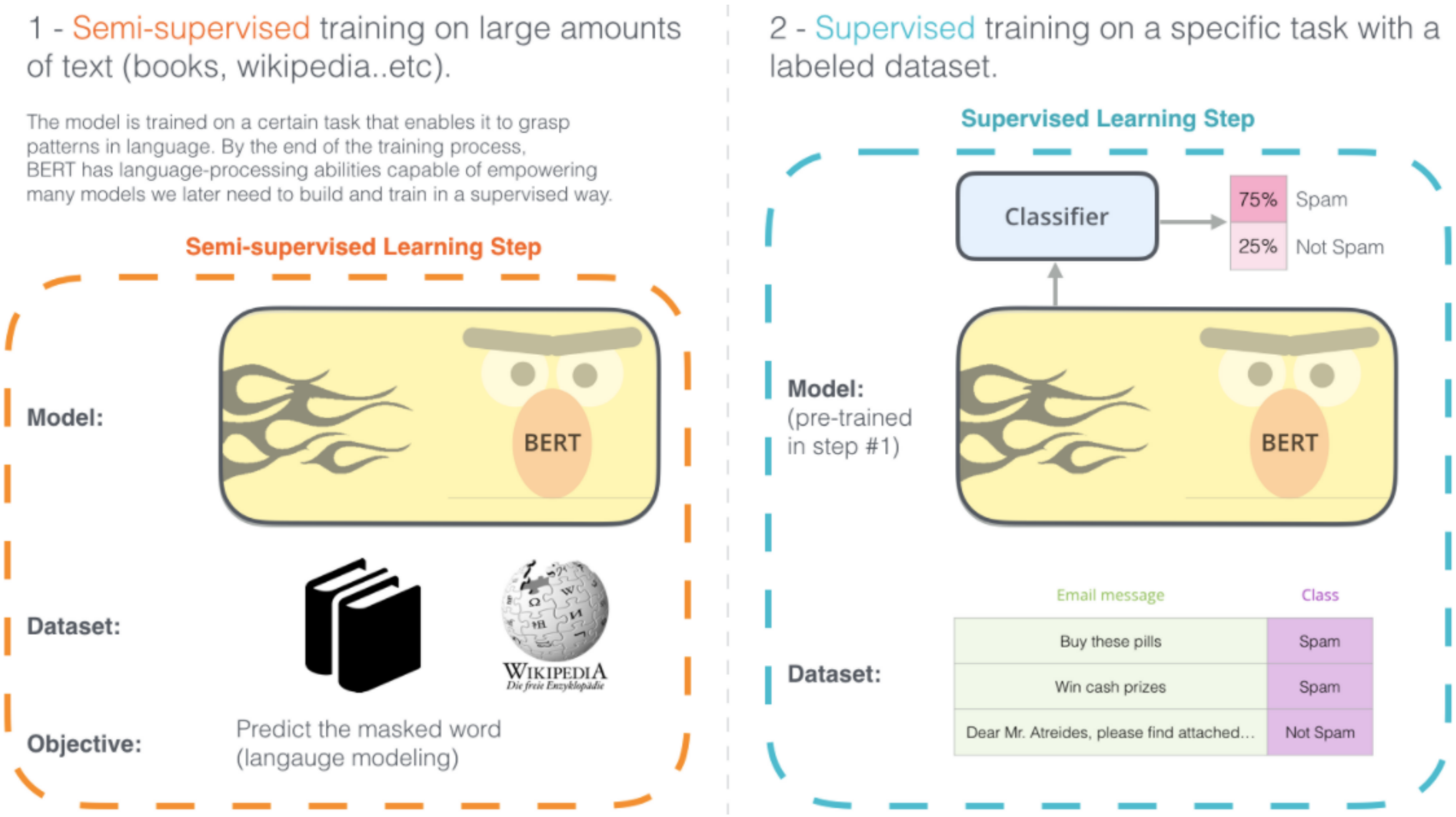


Outline

- **Language Models & NLP**
  - k-gram models, RNN review, word embeddings, attention
- **Transformer Model**
  - Properties, architecture breakdown
- **Transformer-based Models**
  - BERT, GPTs, Foundation Models

Transformer-Based Models: **BERT**

- Semi-supervised learning + Transformers
  - Semi-supervised learning to learn embeddings in encoder



BERT: Concepts

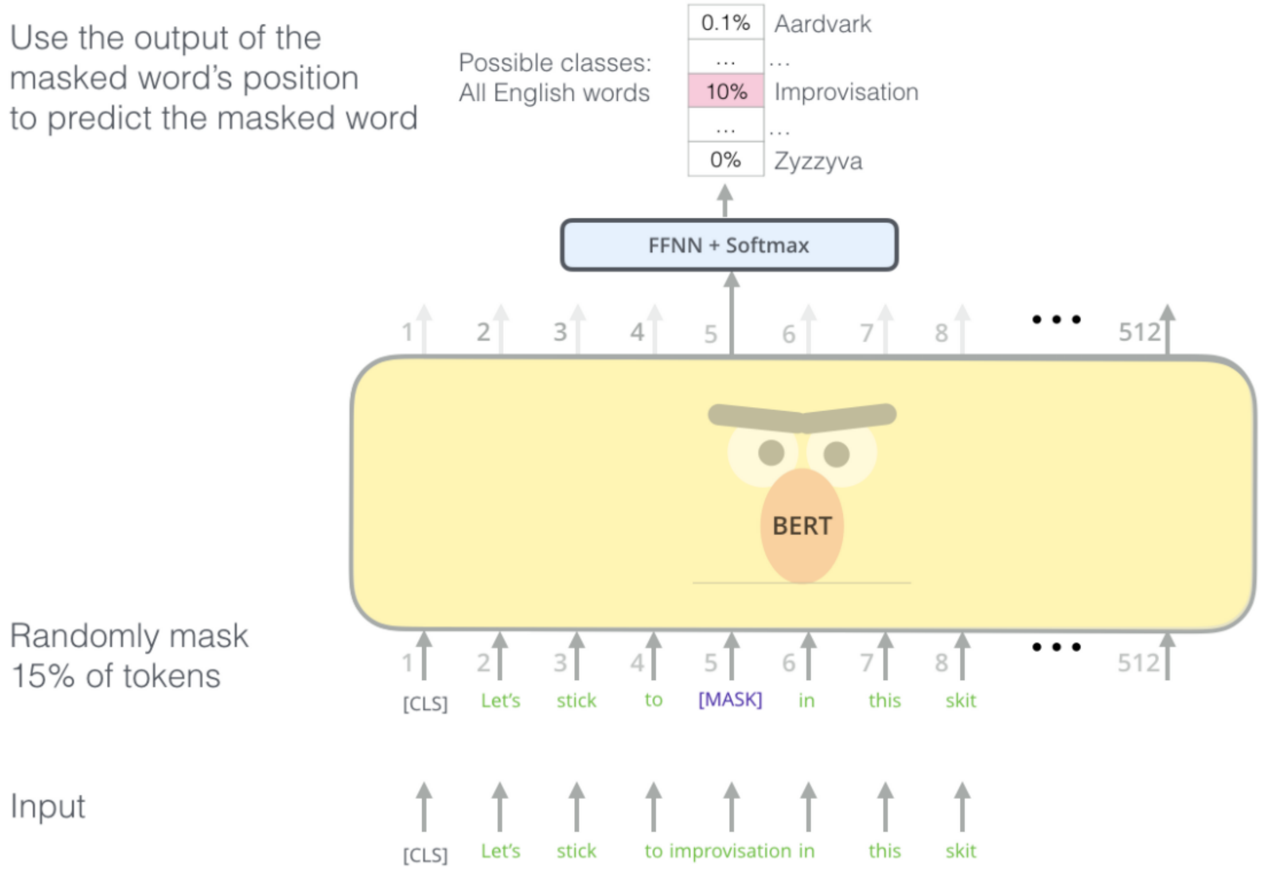
- What makes BERT work? A bunch of ideas:
  - 1. Use the **Transformer** architecture
  - 2. **Pre-training** on corpora using self-supervised learning.
    - Then fine-tune for a particular task
  - 3. **Scale**: BERT-Large has 340 million parameters

System	MNLI-(m/mm)	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	**Average**
	392k	363k	108k	67k	8.5k	5.7k	3.5k	2.5k	-
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

Results: Devlin et al, BERT: Pre-training of Deep Bidirectional
Transformers for Language Understanding

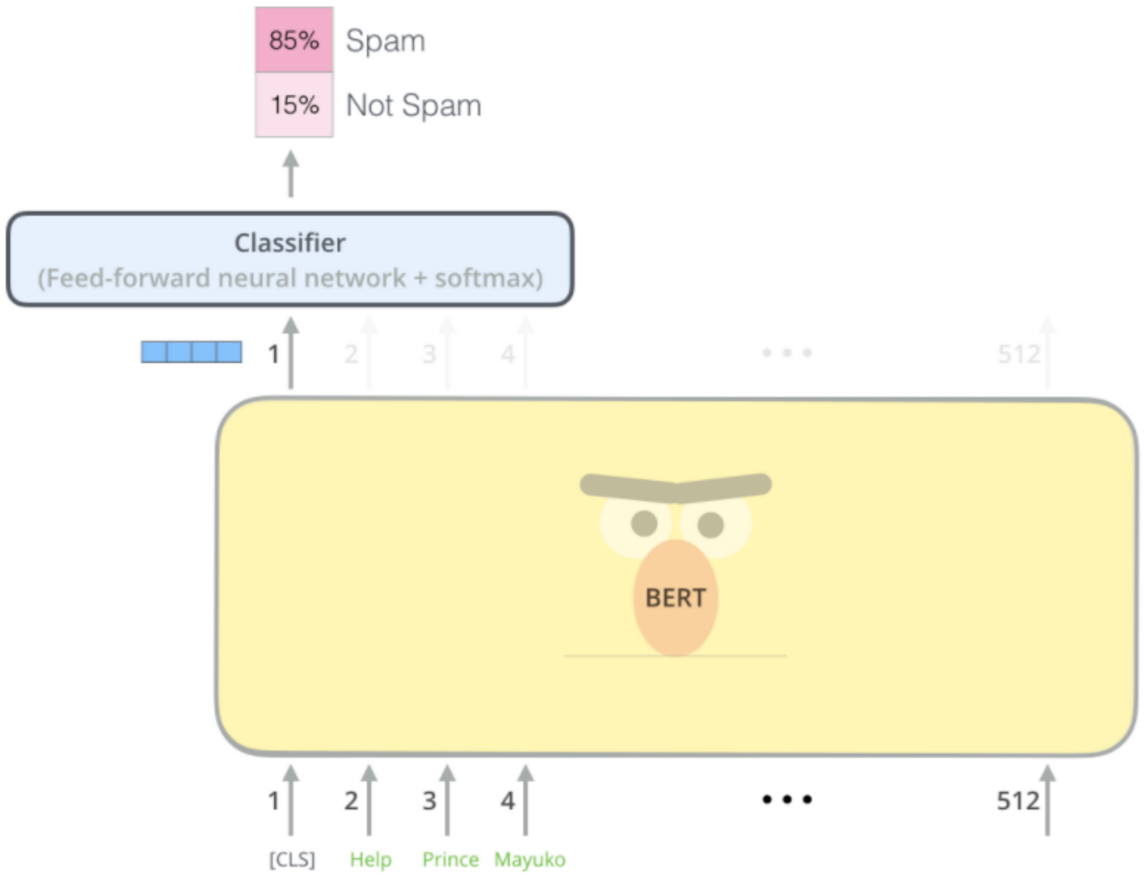
BERT: Training

- BERT is trained on a simple tasks on a huge amount of data:
  - **Masked word prediction:**



BERT: Classification

- Then, fine-tune on a particular task
  - Example: **binary classification**, spam VS not spam

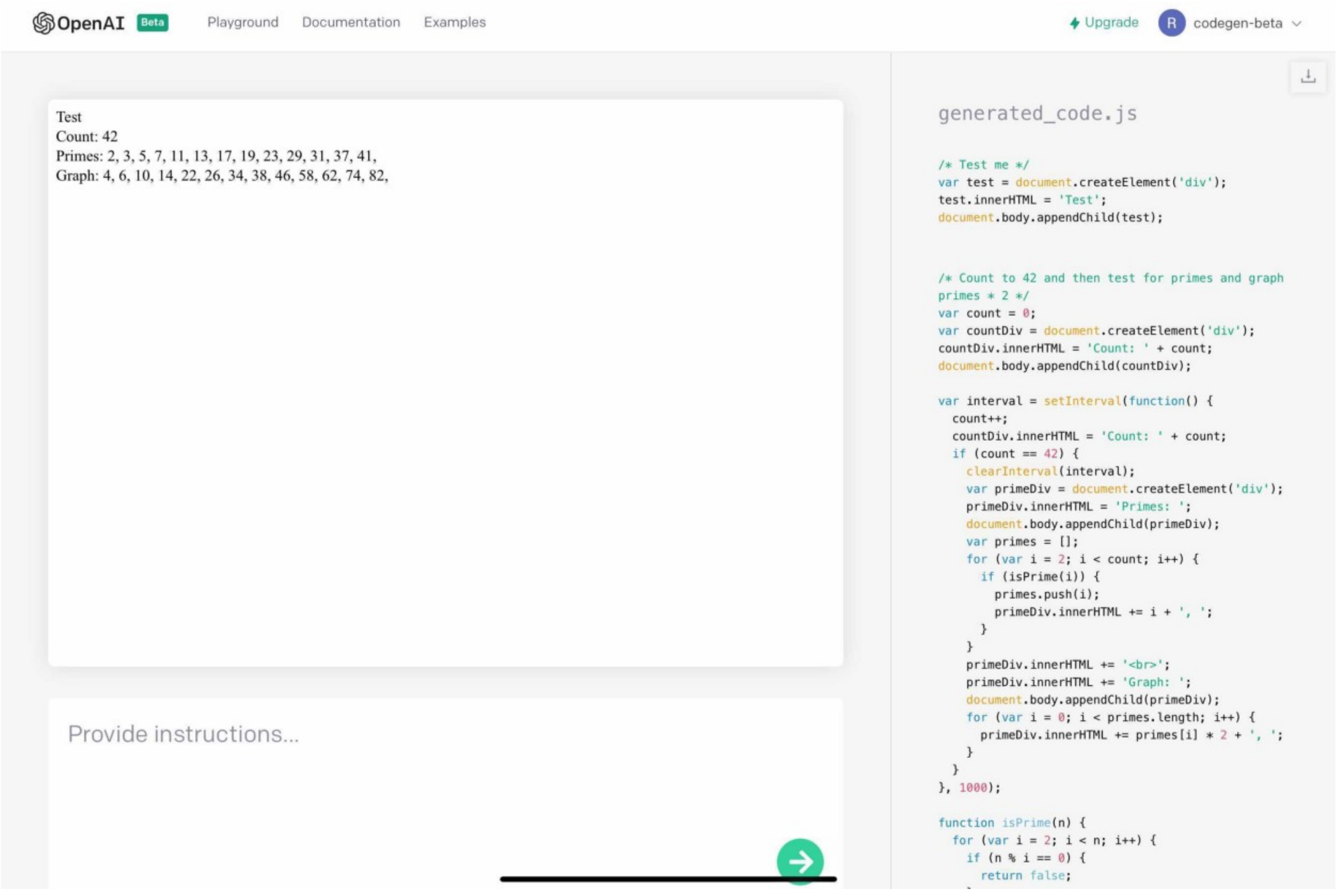


GPT Series of Models

- **GPT**: **G**enerative **P**re-trained **T**ransformer
  - Also built on top of transformer model architecture
  - Essentially the decoder part only
- Goal: generate text (possibly from a **prompt**)
- Scale: huge!
  - GPT-3: 175 billion parameters

Codex

- Codex: a variant of GPT-3 based on source code
  - Outputs code. Ex: show primes



Russell Foltz-Smith

DALL-E

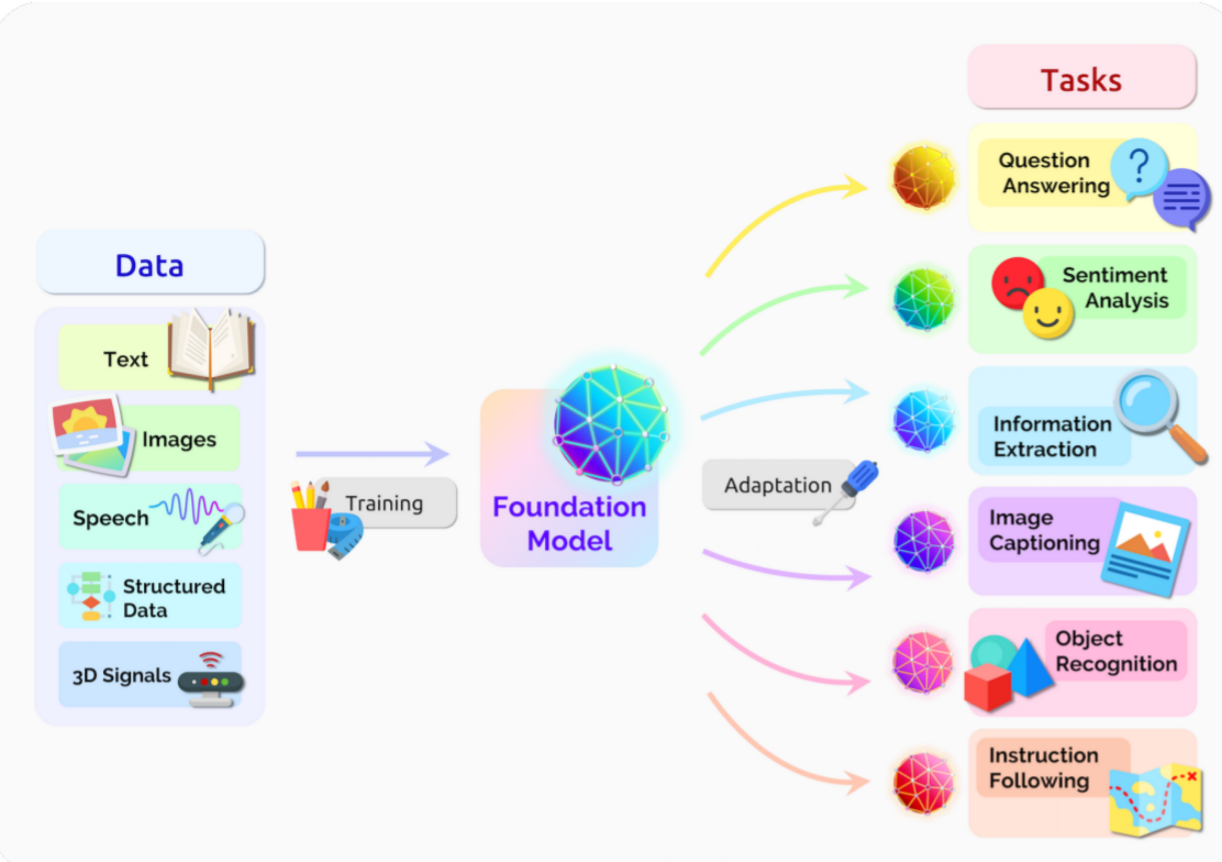
- Create images from text
  - Prompt: “an armchair in the shape of an avocado. . . .”

https://openai.com/blog/dall-e/

- Note: several online demos. Try it yourself!

Foundation Models

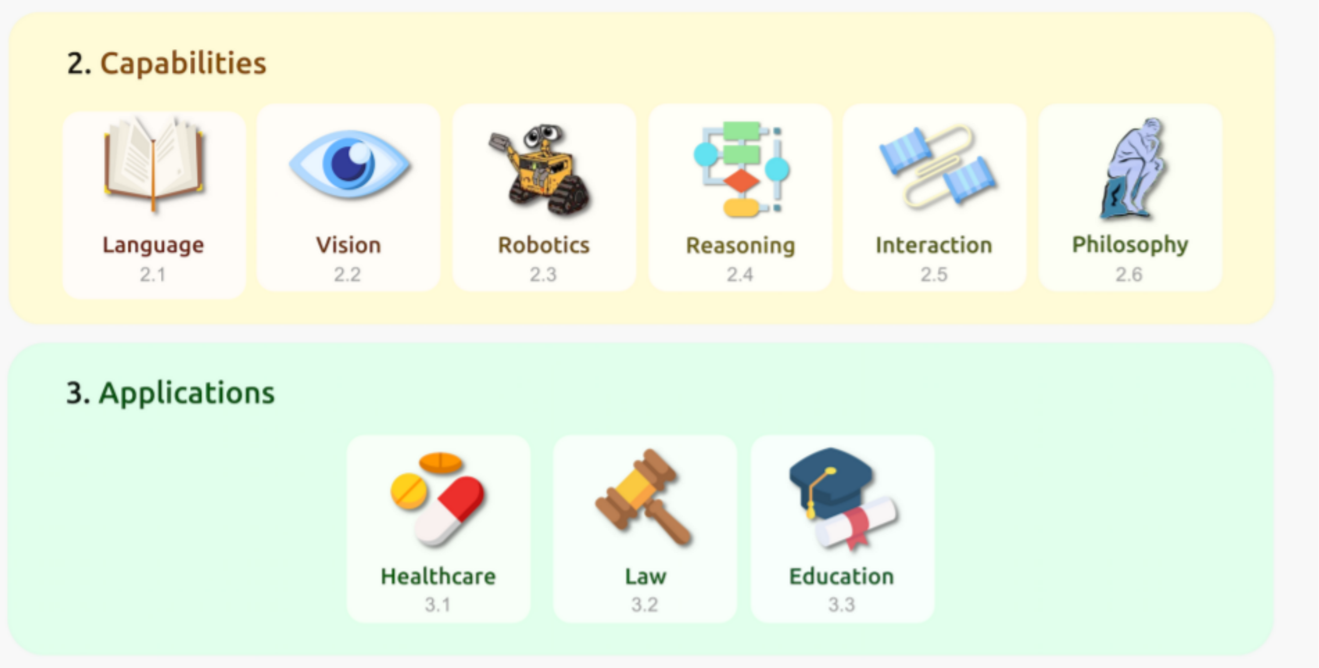
- Many more large scale models
  - Not just focused on text



Bommasani et al, “On the Opportunities and Risks of Foundation Models”

Conclusion

- "Foundation" models based on transformers and beyond
  - Huge, expensive to train, challenging in various ways… but
  - Remarkably powerful for a vast number of tasks.
  - An ingredient for artificial general intelligence?



Bommasani et al, "On the Opportunities and Risks of Foundation Models"

Thanks Everyone!

Some of the slides in these lectures have been adapted/borrowed from materials developed by Mark Craven, David Page, Jude Shavlik, Tom Mitchell, Nina Balcan, Elad Hazan, Tom Dietterich, Pedro Domingos, Jerry Zhu, Yingyu Liang, Volodymyr Kuleshov, Jay Alammar, and Fred Sala