Time Series, Natural Language Processing & Transformers

Applied Machine Learning, KU

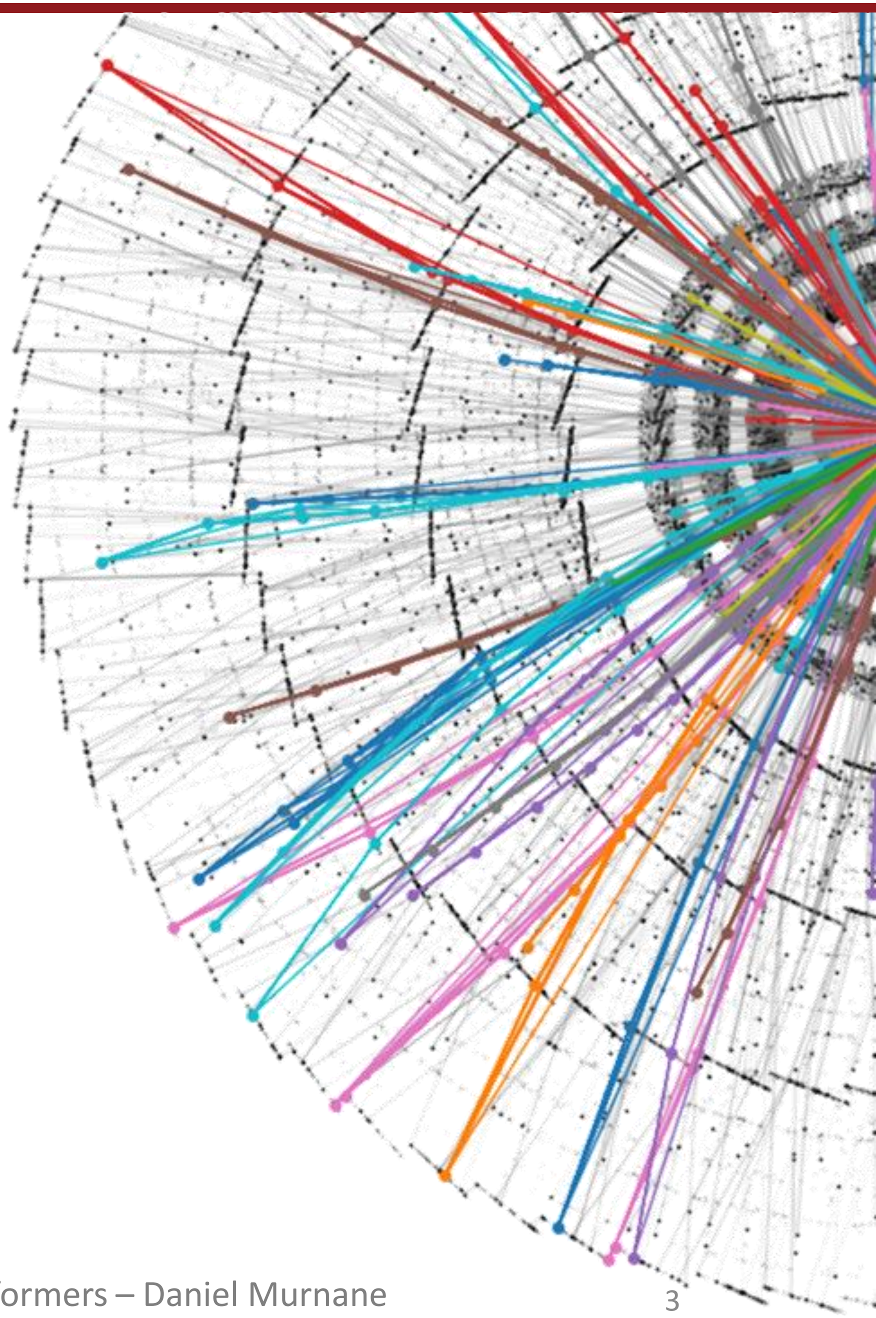
Daniel Murnane - May 8th, 2024

UNIVERSITY OF COPENHAGEN

An intro from Cove

Introduction & Goals

- Goals for today:
  - **Learn the most up-to-date ideas around language and sequential machine learning**
  - Understand what is special about time
  - Learn the history of models for time series
  - See that language is a form of time series data
  - Learn the history of models for language processing
  - Work through the math of a transformer
  - Learn how to train ChatGPT
- Have borrowed content from Inar Timiryasov's slides from last year



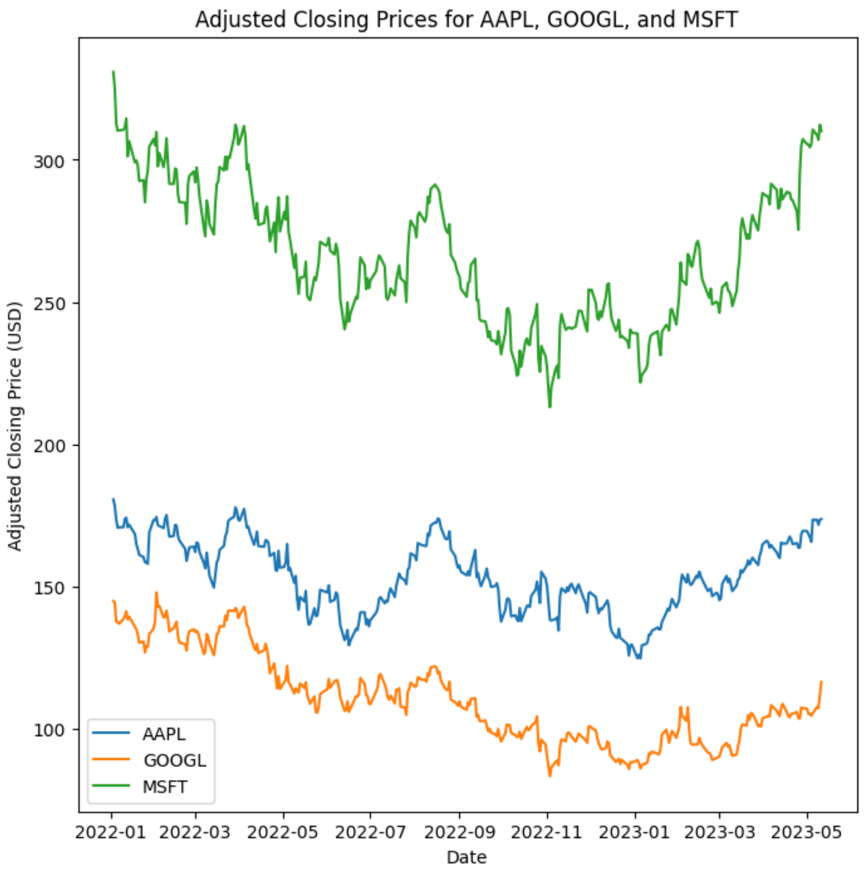
Sequences & Time Series

What is a sequence?

- A sequence is a list. A sequence is a series. A sequence is an ordered set
- A lot of things that we think of as “lists” are actually sets (e.g. shopping list, Christmas wishlist)
- The direction of many sequences is arbitrary, e.g. list of heights in this class – could be tallest to shortest or short to tallest. Does not change the nature of the list
- Time is special: Things earlier in the list may **cause** things later in the list – causality is not arbitrary and can not simply be “reversed”
- This tells us we might need to be careful when dealing with **time** series

Models for Sequences & Time Series

Predicting the Stock Market



- Given a sequence of feature vectors (e.g. $\left[\$_{opening}, \$_{closing}, N_{trades}\right]_i$) can you predict the price of Apple next month?
- This is a hard problem. Like images, series data has small scale and large scale behavior: *trends* and *seasonality*
- We call this “non-stationary” data: the statistical properties of the stock price is not the same from one year to the next
- Previous models needed to correct for this non-stationarity

Time Series before Deep Learning

- A popular approach was/is ARIMA: AutoRegressive Integrated Moving Average
- These models are *autoregressive*: they regress a value for time step t, then use that prediction as input to regress time step $t+1$, etc.
- They are *integrated:* they operate on the *differences* between time step values, rather than the values themselves (this should make the data more "stationary")
- They use a *moving average:* this smooths out noisy values

Recurrent Neural Networks

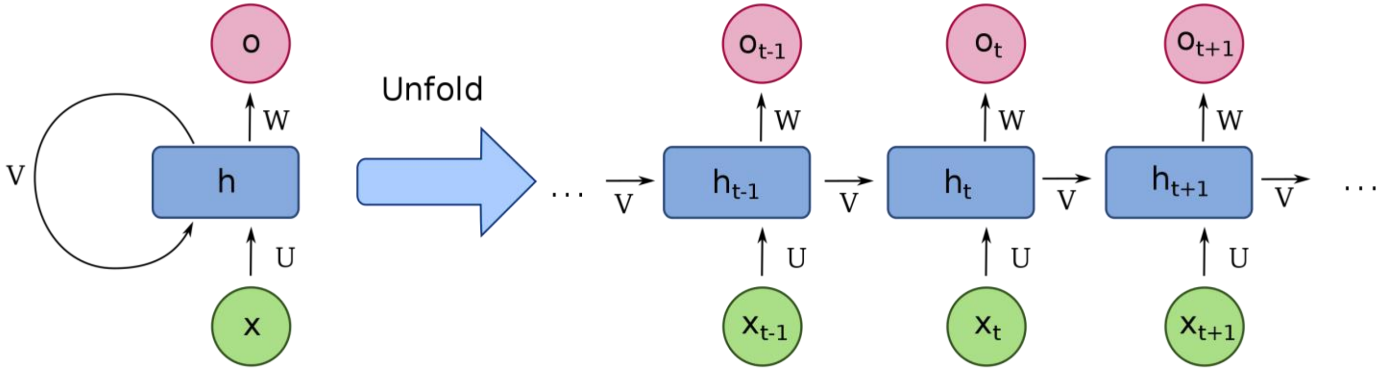
- Regular FFNNs are not well-suited to series data:

1. For the same reason as in images, they need to learn every possible "position of the cat", but now in time since every time step would have its own neurons
2. Worse still, unlike images, series data might be arbitrarily long, and we want to predict arbitrarily into the future. But FFNNs have fixed input shape

- Enter the *recurrent* neural network: Simply keep applying the *same* FFNN to every time step, one at a time

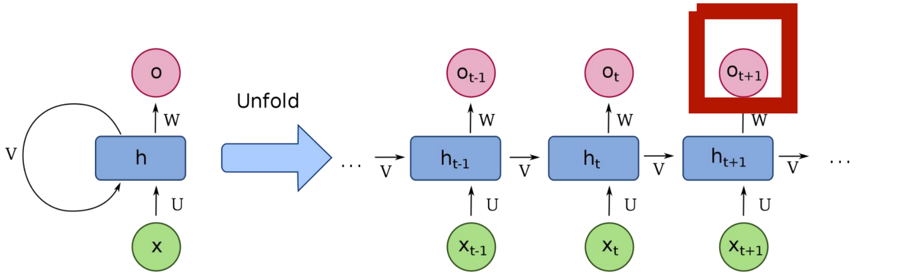
Recurrent Neural Networks

- Enter the *recurrent* neural network: Simply keep applying the *same* FFNN to every time step, one at a time

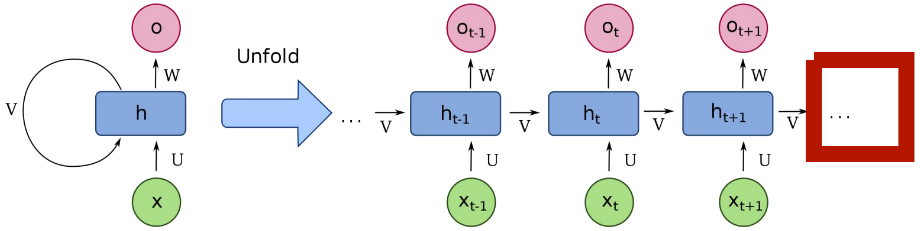


h_t are hidden states

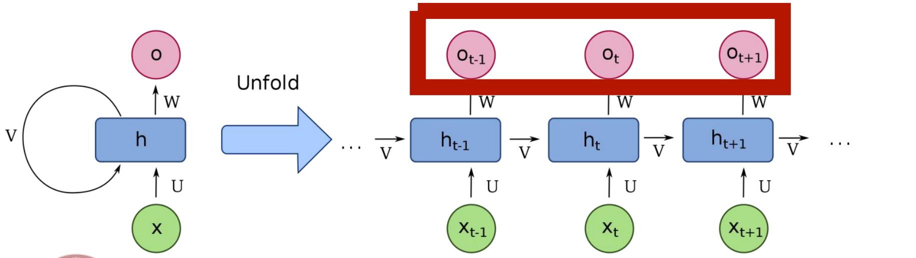
Use Cases of RNNs



Predicting the next value (and maybe using it as input to the next prediction, i.e. autoregression)



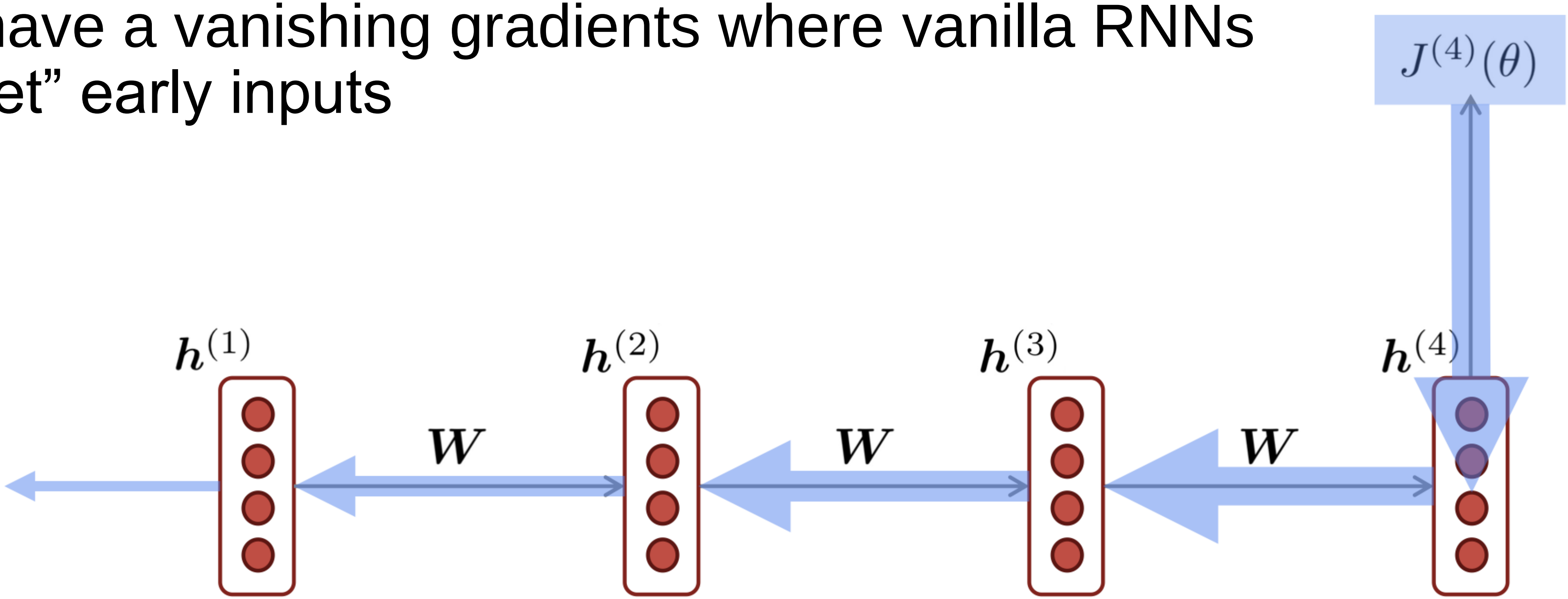
Predicting some downstream value (i.e. not simply the next value)



Predicting some property of the entire series (e.g. what language is it?)

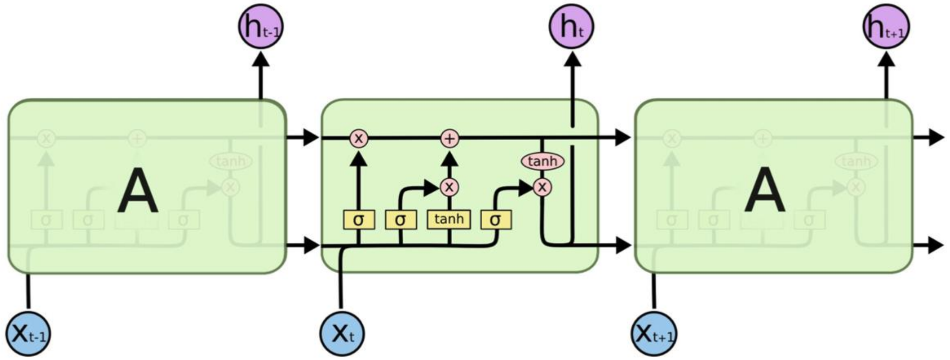
Long Short-Term Memory

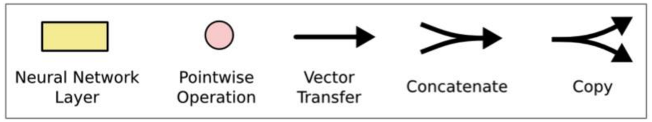
- We have a vanishing gradients where vanilla RNNs "forget" early inputs



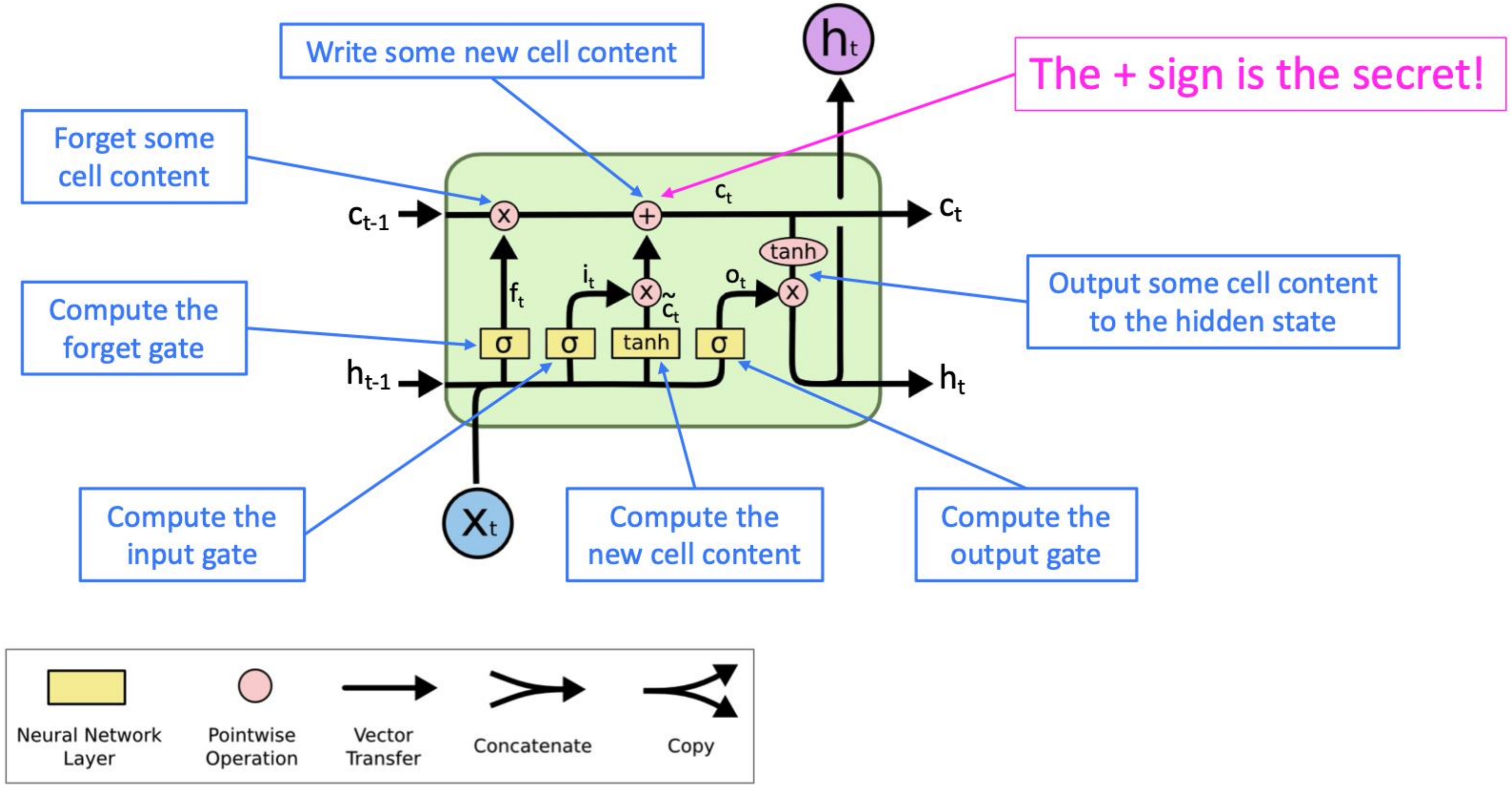
Long Short-Term Memory

- We have a vanishing gradients where vanilla RNNs "forget" early inputs
- Introduce a way to "gate" (activate) information from previous layers





Long Short-Term Memory



Language as a Sequence

Representing Words: Tokenization

- Recall how we represent an image: 3 values for each color in a pixel
- To do the same for a sentence, we need a *vocabulary* – a map from letters or words into numbers
- We call this *tokenization*
- A helpful rule of thumb is that one token generally corresponds to ~4 characters of text for common English text. This translates to roughly ¾ of a word (so 100 tokens ~= 75 words).

Tokens 220 Characters 747

Week 4 (Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and Auto-Encoders (AE)):
May 15: 13:15-17:00: Convolutional Neural Networks (CNNs) and image analysis (Daniel Murnane).
Exercise: Recognize images (MNIST dataset, sparse chips for radiation, and/or insoluables from Greenland ice cores) with a CNN.
May 17: 9:15-12:00: Recurrent Neural Networks (RNN), Long Short Term Memory (LSTM) and Natural Language Processing (NLP) (Inar Timiryasov).
Exercise: Use an LSTM to predict flight traffic and do Natural Language Processing on IMDB movie reviews.
May 17: 13:15-17:00: (Variational) Auto-Encoder and anomaly detection (TP).
Exercise: Compress images using Auto-Encoder, and cluster latent space with UMAP.

TEXT TOKEN IDS

Tokens 220 Characters 747

[20916, 604, 357, 3103, 85, 2122, 282, 47986, 27862, 357, 18474, 82, 828, 3311, 6657, 47986, 27862, 357, 49, 6144, 82, 828, 290, 11160, 12, 4834, 19815, 364, 357, 14242, 8, 2599, 220, 198, 6747, 1315, 25, 1511, 25, 1314, 12, 1558, 25, 405, 25, 34872, 2122, 282, 47986, 27862, 357, 18474, 82, 8, 290, 2939, 3781, 357, 19962, 337, 700, 1531, 737, 198, 220, 220, 220, 220, 32900, 25, 31517, 1096, 4263, 357, 39764, 8808, 27039, 11, 29877, 12014, 329, 11881, 11, 290, 14, 273, 35831, 84, 2977, 422, 30155, 4771, 21758, 8, 351, 257, 8100, 13, 198, 6747, 1596, 25, 860, 25, 1314, 12, 1065, 25, 405, 25, 3311, 6657, 47986, 27862, 357, 49, 6144, 828, 5882, 10073, 35118, 14059, 357, 43, 2257, 44, 8, 290, 12068, 15417, 28403, 357, 45, 19930, 8, 357, 818, 283, 5045, 9045, 292, 709, 737, 198, 220, 220, 220, 220, 32900, 25, 5765, 281, 406, 2257, 44, 284, 4331, 5474, 4979, 290, 466, 12068, 15417, 28403, 319, 8959, 11012, 3807, 8088, 13, 198, 6747, 1596, 25, 1511, 25, 1314, 12, 1558, 25, 405, 25, 357, 23907, 864, 8, 11160, 12, 27195, 12342, 290, 32172, 13326, 357, 7250, 737, 198, 220, 220, 220, 220, 32900, 25, 3082, 601, 4263, 1262, 11160, 12, 27195, 12342, 11, 290, 13946, 41270, 2272, 351, 471, 33767, 13]

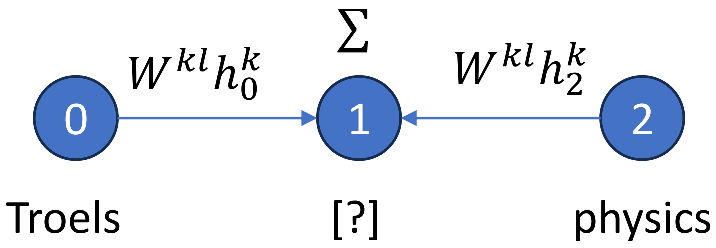
TEXT TOKEN IDS

This shows once kind of encoding: *byte-pair encoding*

https://platform.openai.com/tokenizer

Embeddings

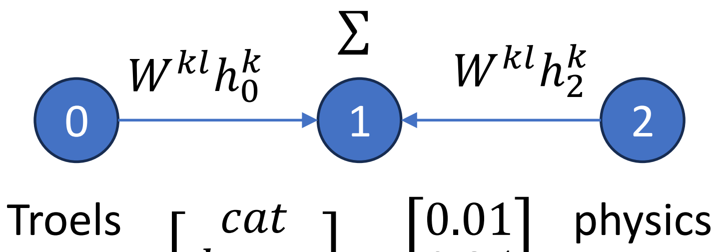
- We have tokens for each word, but they don't *mean* anything
- Let's train a model that looks like this:



- For a window of three words, we hide the central word
- We pass a message from the two visible words and aggregate at the hidden word
- We apply a FFNN to this node to predict the missing word

Embeddings

- We apply a FFNN to this node to predict the missing word

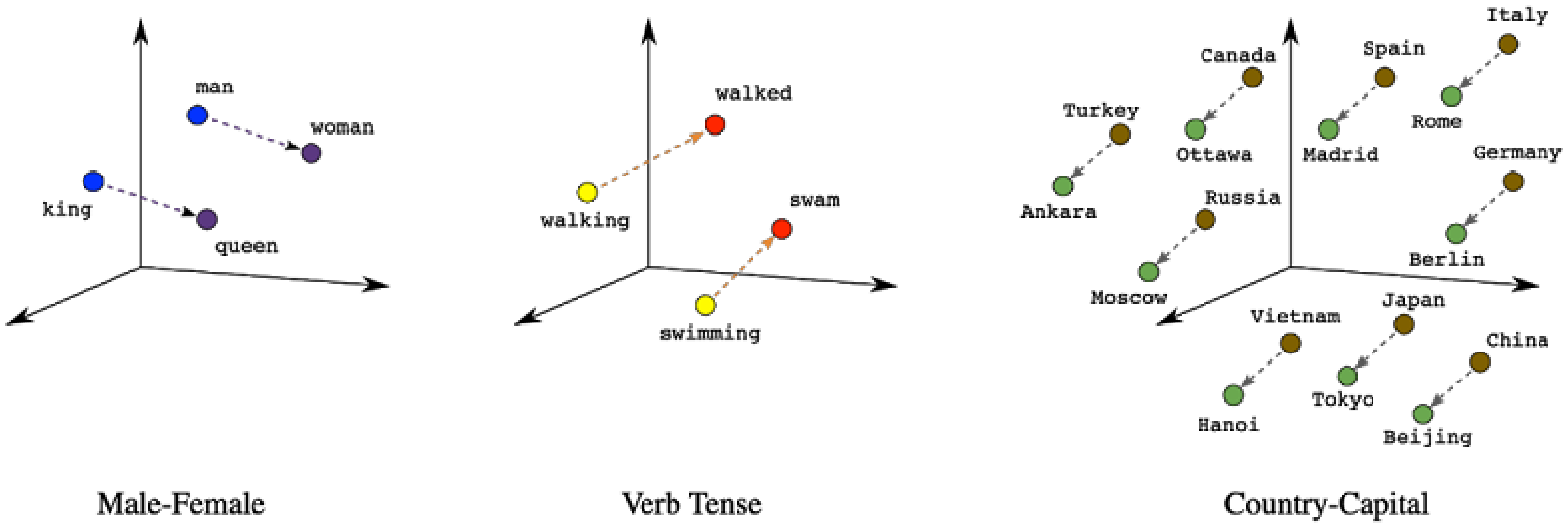


$$\begin{bmatrix} cat \\ loves \\ jumps \\ \dots \end{bmatrix} = \begin{bmatrix} 0.01 \\ 0.85 \\ 0.01 \\ \dots \end{bmatrix}$$

- This is called the *Word2Vec,* and it produces a contextual embedding for words, based on their *co-occurrence*

Embeddings

- This is called the *Word2Vec,* and it produces a contextual embedding for words, based on their *co-occurrence*
- Because the model is simple, its embeddings are simple, and we can actually see interpretable patterns in the positions of the words



Source: https://cloud.google.com/blog/topics/developers-practitioners/meet-ais-multitool-vector-embeddings

Attention and the Transformer

Attention in a Sequence

- One of the biggest challenges in the RNN is the “forgetting problem” – each successive application of the FFNN reduces the impact of distant tokens
- Now that we are in language, we want some *very* long sequences, like books!
- Idea: Let each token see *every other* token simultaneously, and learn which to pay attention to
- For example, we might have a sentence

Paris is a _____

with the likeliest prediction **city**

Attention in a Sequence

- But what if the sentence is

  Like all Hiltons, Paris is a _____

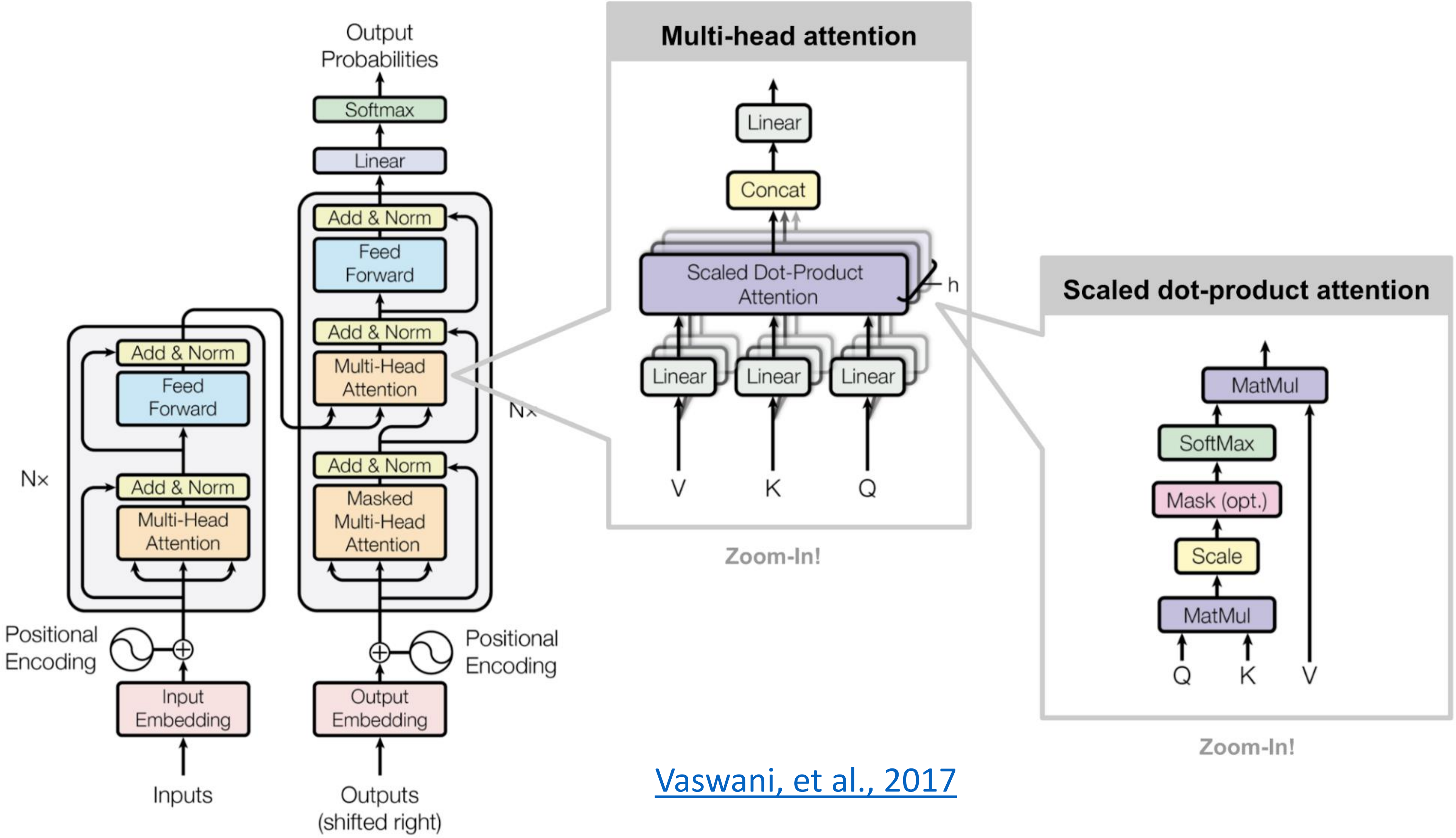
  now the prediction might be **millionaire**
- We want our prediction to incorporate these pieces of information:

  Like all Hiltons, Paris is a _____

- For a sequence of length N, let's define an attention matrix A_{ij} of size $N \times N$. The entries in the i^{th} row are the relative "importances" of all words in the sentence to the i^{th} word
- The entries in each row sum to 1, so each word has an "attention budget"

Transformer Encoding

- I have already shown the transformer model in “graph language”. Let’s see it in “NLP language”

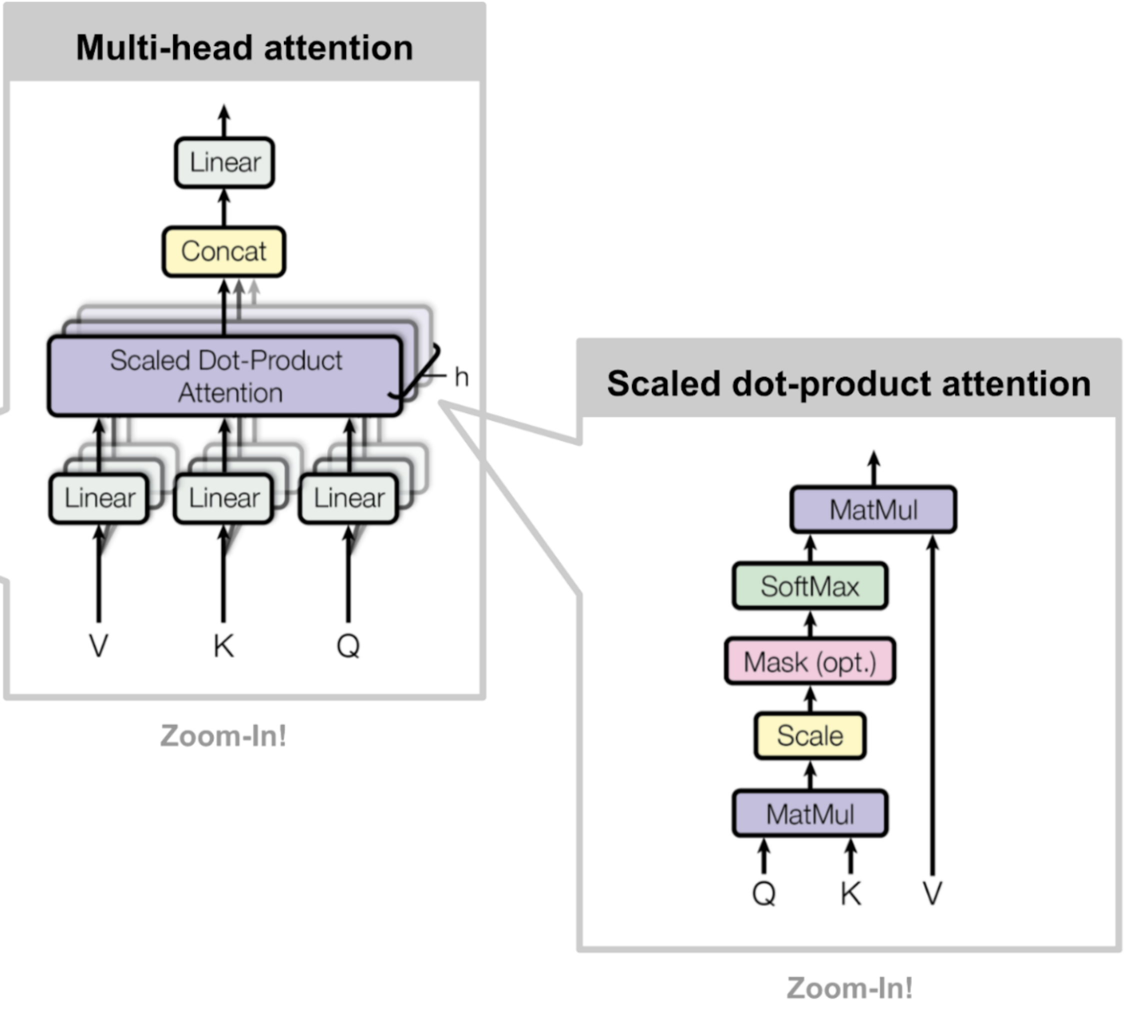


[Vaswani, et al., 2017]

Transformer Encoding

- I have already shown the transformer model in "graph language". Let's see it in "NLP language"
- Now, Q, K, V are $N_{tokens} \times N_{hidden}$ matrices
- Since we want every token to look at every other token, we can think of the graph as fully-connected
- In that case, it's cheaper to ignore the idea of messages, and just do a sequence of matrix multiplications:

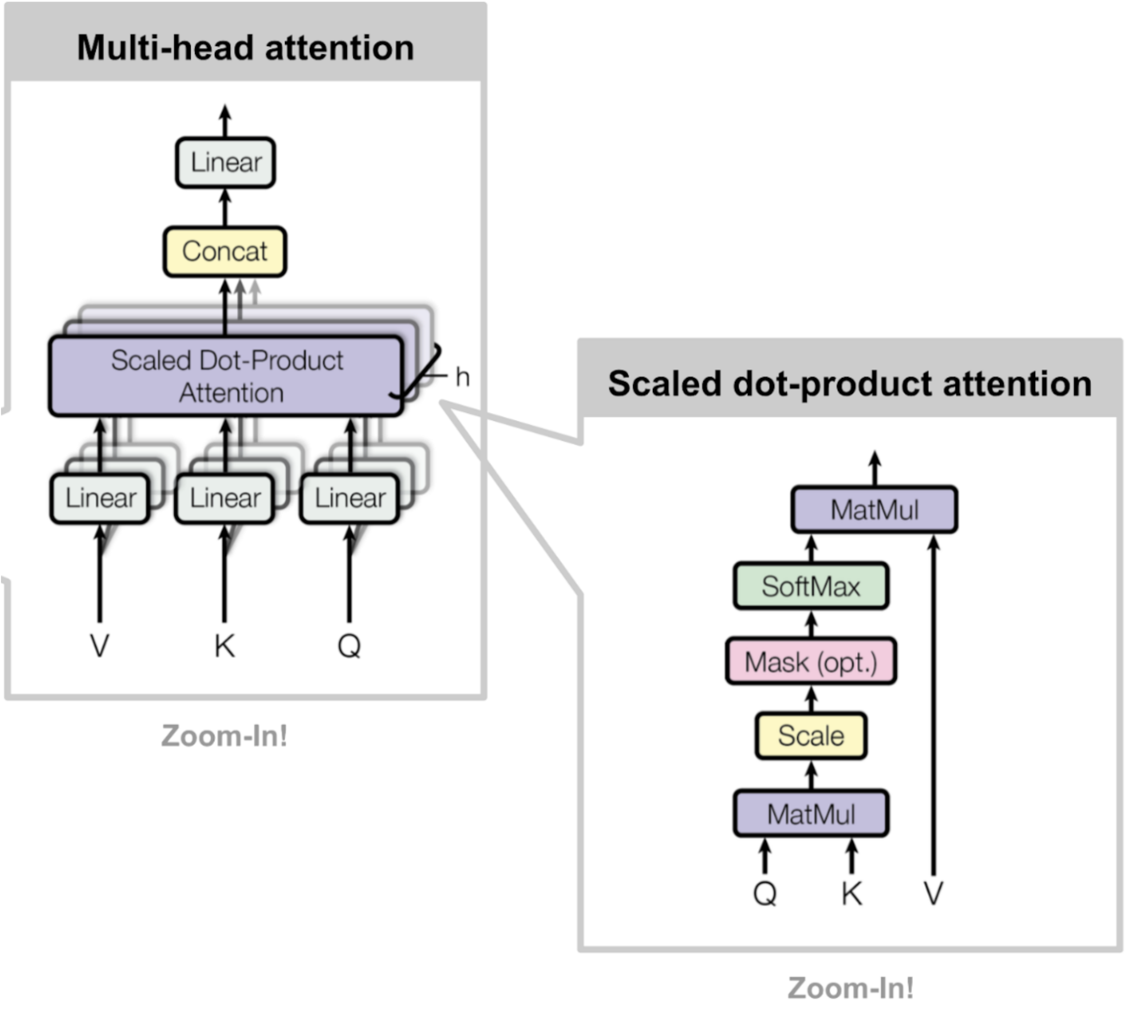
$$encoding = softmax(QK^T)V$$



Vaswani, et al., 2017

Transformer Encoding

- In that case, it's cheaper to ignore the idea of messages, and just do a sequence of matrix multiplications:
$$encoding = softmax(QK^T)V$$
- **This is why transformers are a trillion-dollar industry: they are extremely efficient to calculate, but can capture arbitrarily complex meaning**
- QK^T is a dot-product of all Q_i vectors with all K_j vectors: the output is the attention, which is how aligned a token's Q vector is with another token's K vector



[Vaswani, et al., 2017]

Positional Encoding

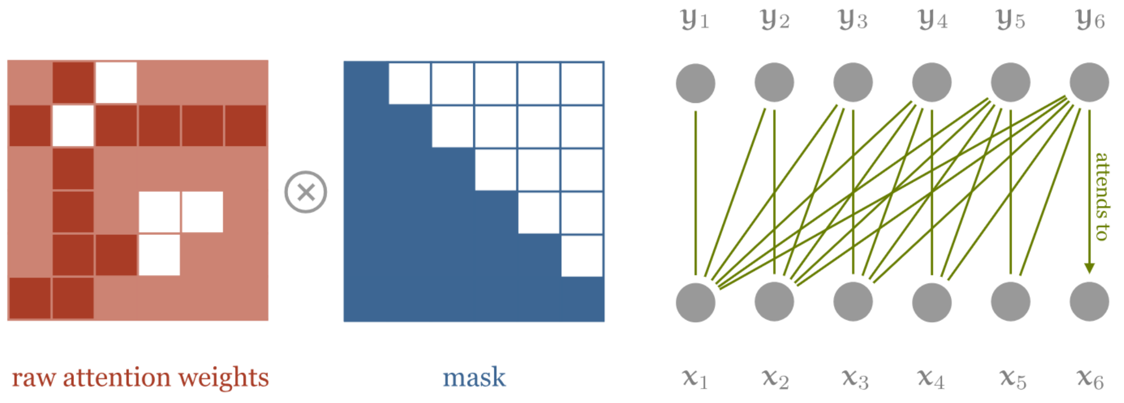
- Recall that in a GNN (and a transformer is a kind of GNN), a convolution doesn't depend on the ordering of the nodes
- Indeed a vanilla transformer will encode "Paris is a city" and "Is Paris a city" to be exactly the same value
- That's a problem here – order is important, and we need a way to specify the distance between words separated across the sequence
- So let's attach a number to each token, listing its position

Paris is a city.	Is Paris a city.
0 1 2 3	0 1 2 3

Now our transformer will encode them differently! Pretty simple.

Transformer Decoding: Masked Attention & Autoregression

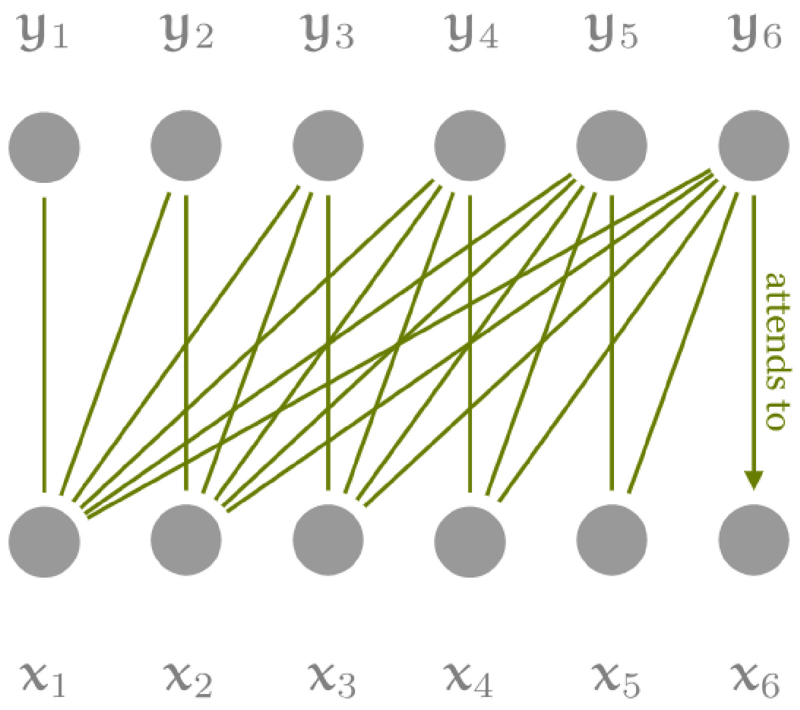
- The task of predicting the next word in a sentence turns out to be very good for learning how to embed words, **and** lets us build a generative language model – two birds with one stone!
- Up until now, we have given the transformer our whole sequence X_i
- In next-word-prediction, we only want each token to be able to look *backwards*
- We apply a "causal mask" to the attention matrix, to avoid future words impacting a token



https://peterbloem.nl/blog/transformers

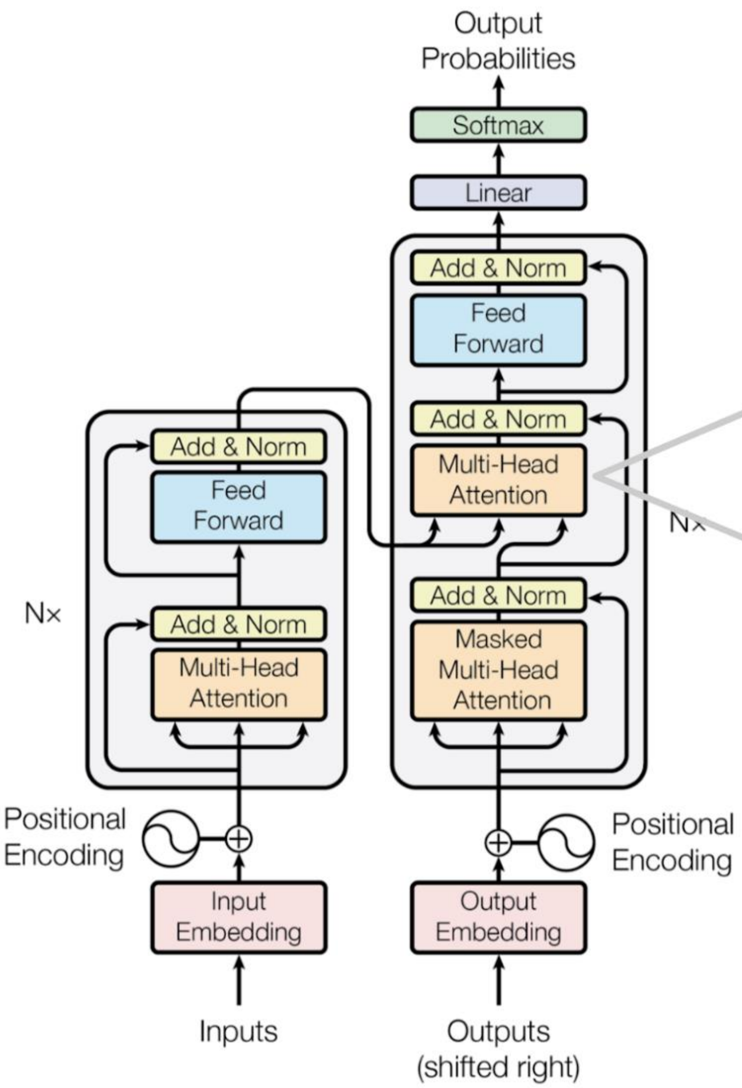
Transformer Decoding: Masked Attention & Autoregression

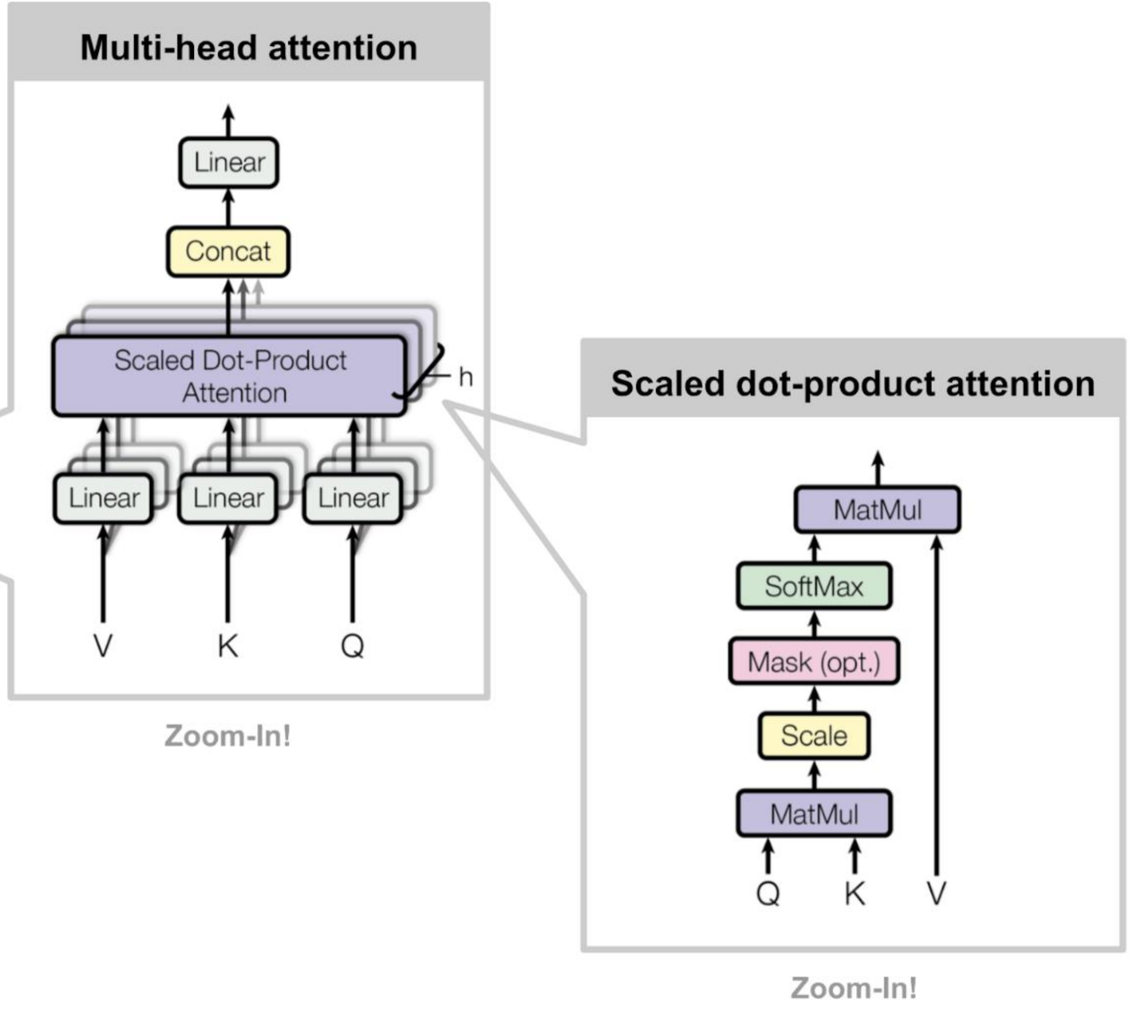
- In next-word-prediction, we only want each token to be able to look *backwards*
- We apply a "causal mask" to the attention matrix, to avoid future words impacting a token
- In graph language, we can just say that a transformer decoder has half the edges removed. An edge can never point from a past token to a future token



https://peterbloem.nl/blog/transformers

Putting it all together



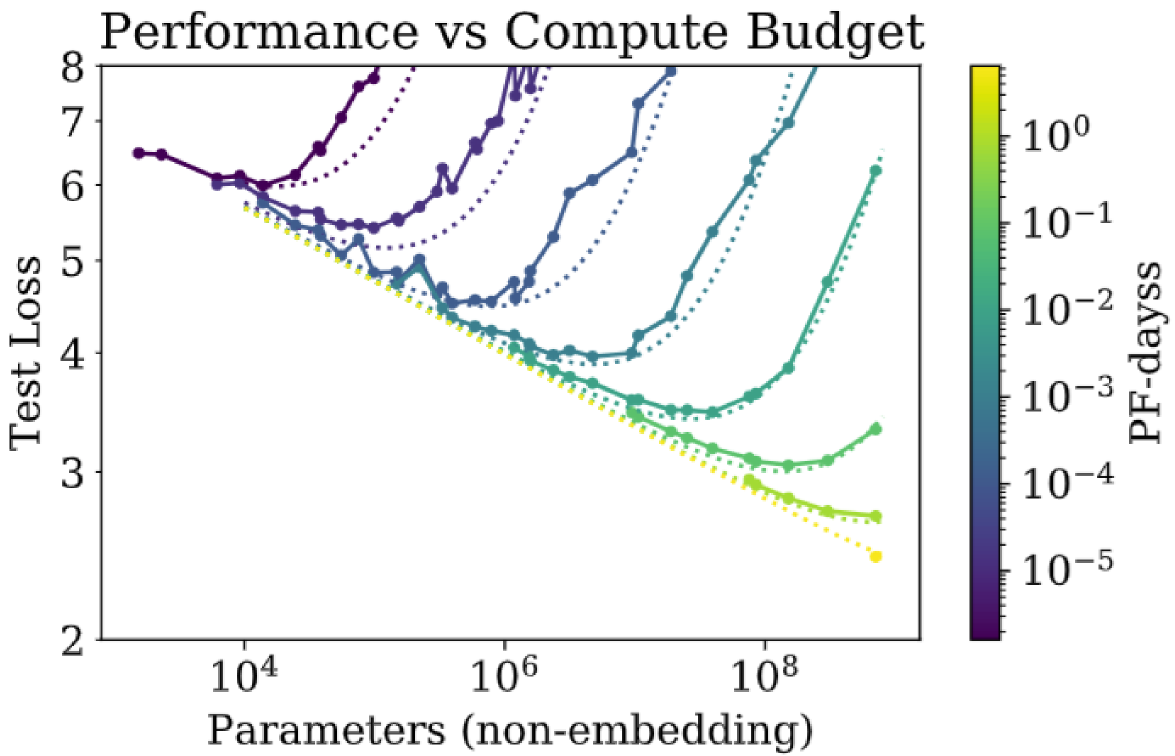


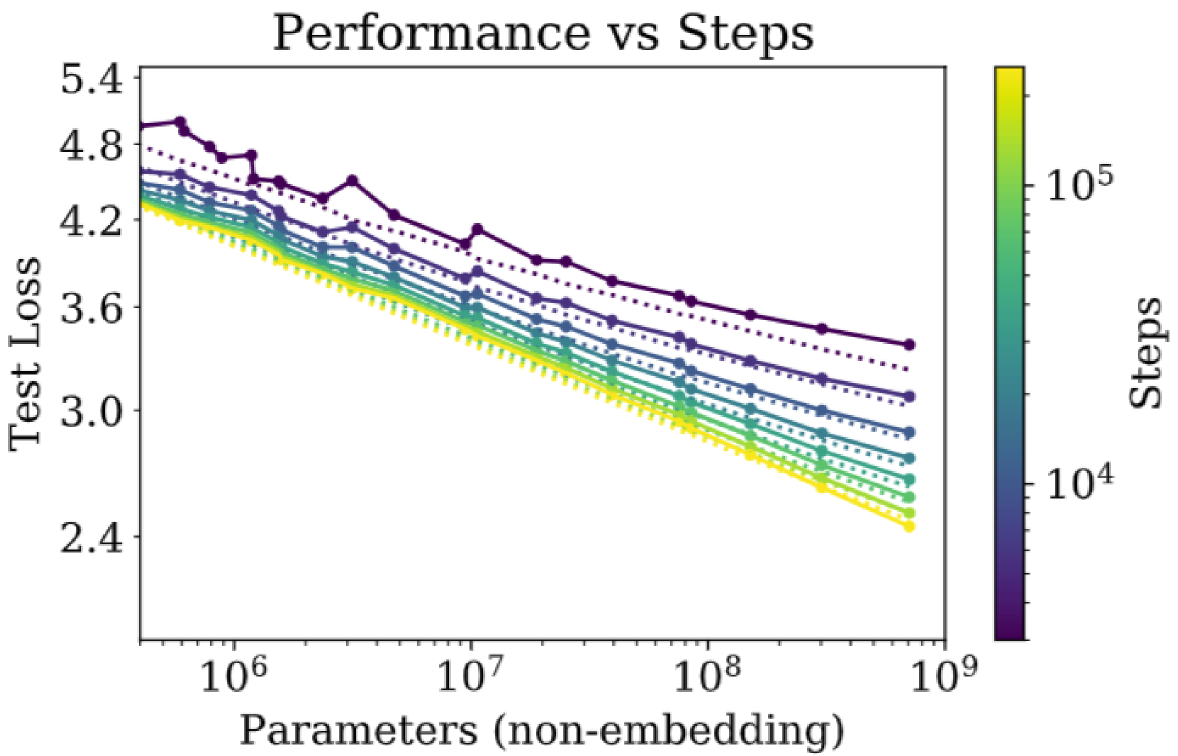
[Vaswani, et al., 2017]

Case Study: ChatGPT

Scaling Laws of Large Language Models

- OpenAI have established empirically observed "scaling laws" in LLMs
- TL;DR: As you increase your model size, you need fewer steps
- It appears that performance just *keeps getting better* according to a power law

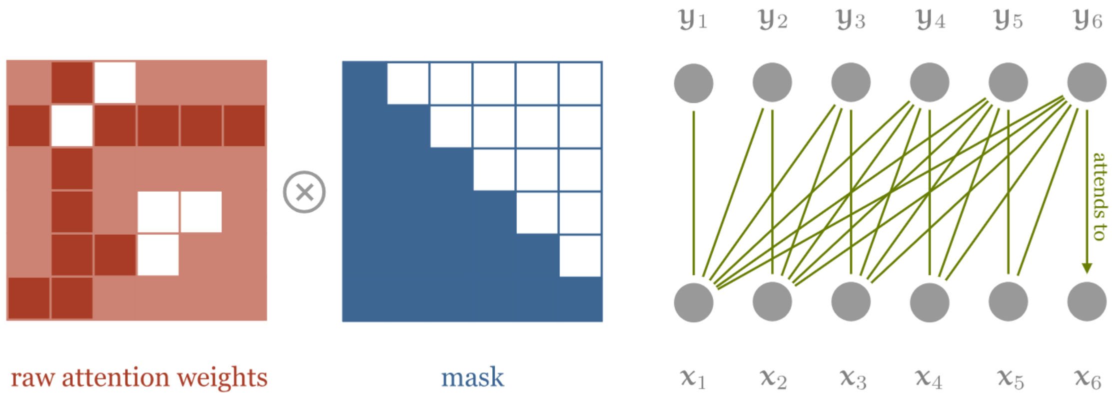




https://arxiv.org/pdf/2001.08361

Scaling **Costs** of Large Language Models

- To compute the next word in an N-token sequence, a transformer must calculate $\mathrm{O}(N \times N)$ attention weights. This takes a *lot* of memory

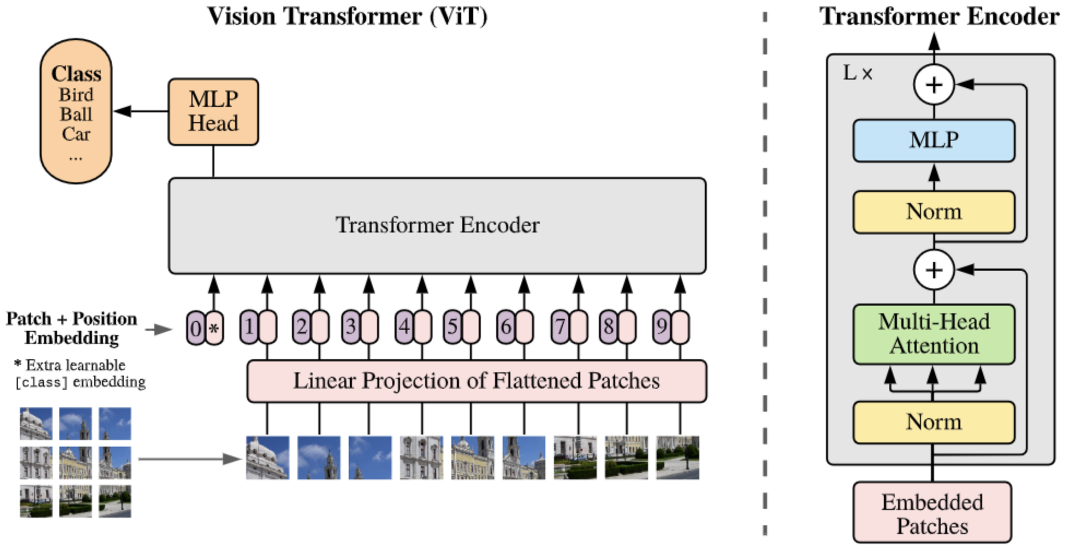


- The GPT transformer is auto-regressive, so generating N tokens takes $\mathrm{O}(N \times N)$ time
- We can usually trade memory for time, but we are trapped on both sides by N^2 scaling
- TL;DR: Transformers are expensive

Case Study: Vision Transformers

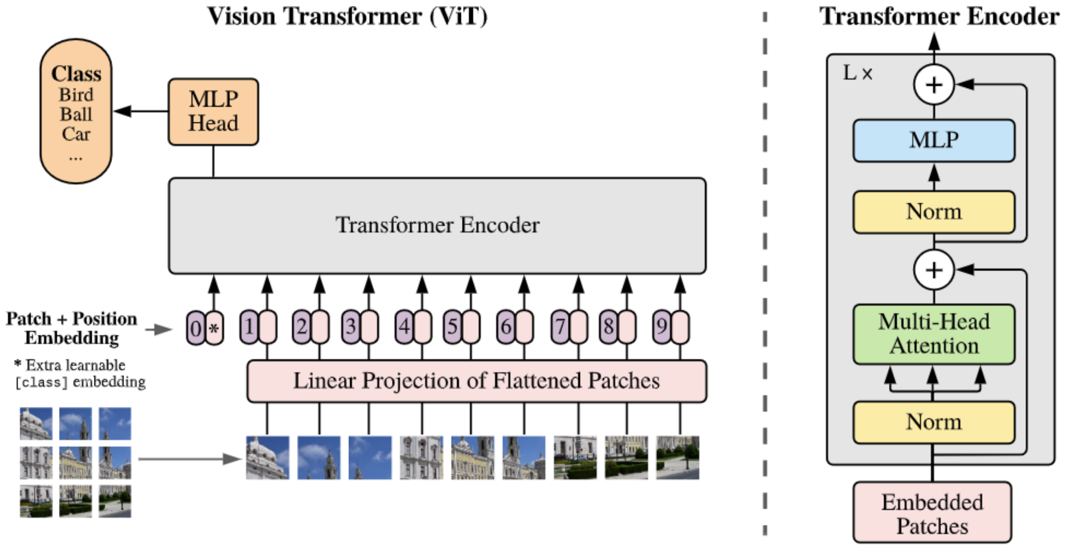
Tokenization of an Image

- Recall that in the CNN, we used strided windows to reduce the image size, where each pixel now contains higher-scale information
- We use windows again, breaking our image into $P \times P$ patches
- These get passed through a FFNN and are the "tokens" of the image



Attention on an Image

- After tokenization/patchification, everything else in the vision transformer works just like in language and in graphs – each patch can see all other patches and has a limited attention budget to spend on them
- We can use the attention to understand what the vision transformer has learned



Attention on an Image

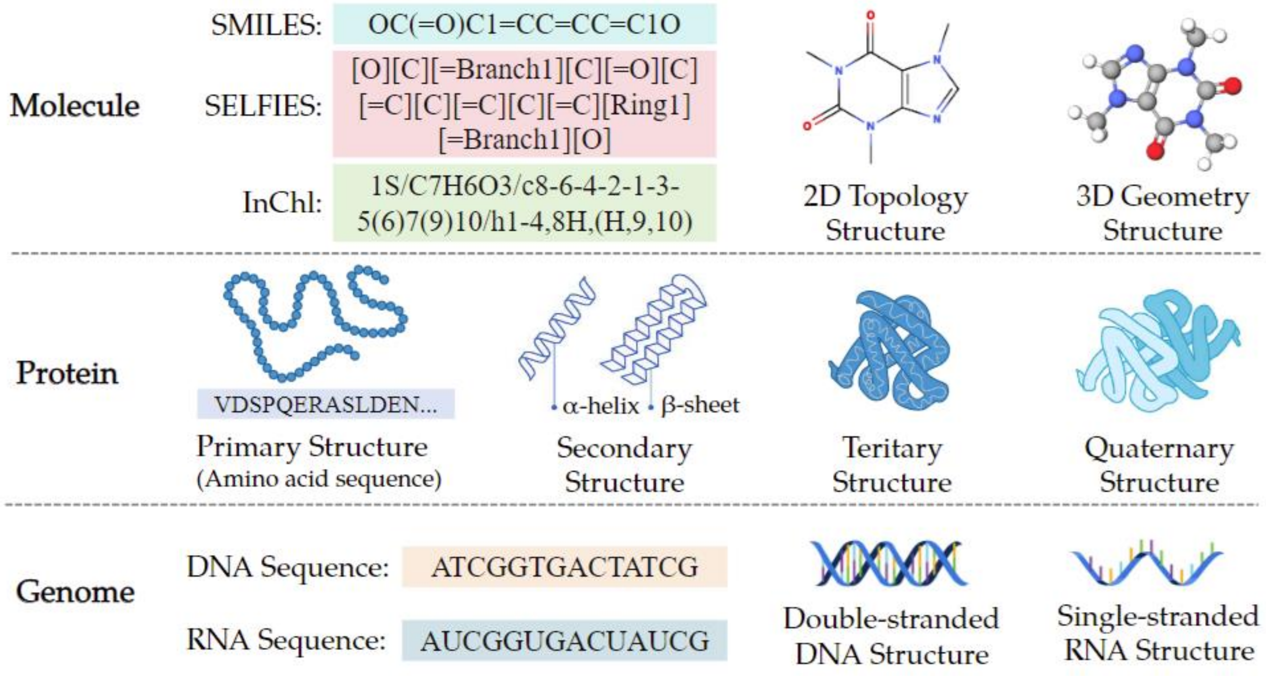
- We can use the attention to understand what the vision transformer has learned
- Here we actually have an attention-weighted LSTM
- It has to describe a scene
- The lighting gives the attention between output and image tokens

Xu et al. 2015

Case Study: Biological Language Models

X-as-a-language: DNA and Genetics

- A very good application of X-as-a-language is biology
- Proteins and DNA are natural sequences
- They also curl and fold, which is why AlphaFold (a model that used both graph-like structure, and sequence-like structure) works very well!



- FYI: Physics Language Models? This is my research grant