Lecture 12: Transformers

Peter Bloem
Deep Learning 2020

dlvu.github.io

THE PLAN

part one: self-attention

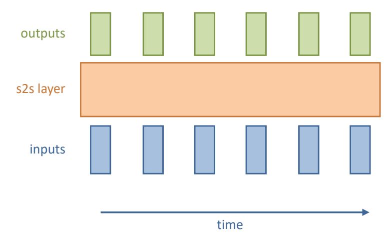
part two: transformers

part three: famous transformers

PART ONE: SELF-ATTENTION

A recurrent neural network is any neural network that has a cycle in it

RECAP: SEQUENCE-TO-SEQUENCE LAYERS



4

RECAP: SEQUENCE-TO-SEQUENCE LAYERS

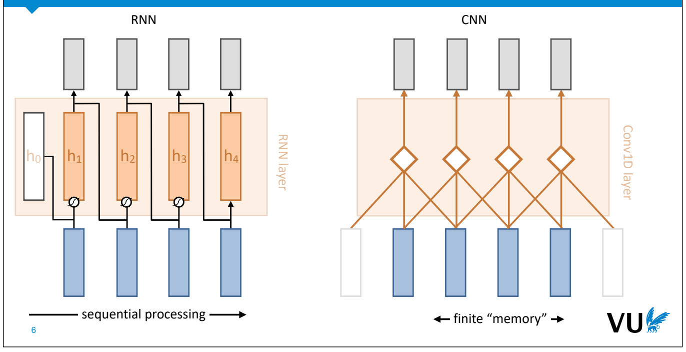
Defining property: can handle sequences of different lengths with the same parameters.

Versatile: label-to-sequence, sequence-to-label, sequence-to-sequence, autoregressive training.

Causal or non-causal: casual models can only look backward.

5

RECURRENT CONNECTIONS, CONVOLUTIONS



6

We've seen two examples of (non-trivial) sequence-to-sequence layers so far: recurrent neural networks, and convolutions. RNNs have the benefit that they can potentially look infinitely far back into the sequence, but they require fundamentally sequential processing, making them slow. Convolution don't have this drawback—we can compute each output vector in parallel if we want to—but the downside is that they are limited in how far back they can look into the sequence.

Self-attention is another sequence-to-sequence layer, and one which provides us with the best of both worlds: parallel processing and a potentially infinite memory.

SELF-ATTENTION

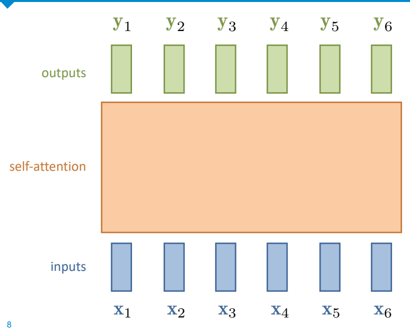
Best of both worlds: parallel computation and long dependencies.

Simple self-attention: the basic idea

Practical self-attention: adding some bells and whistles.

We'll explain the name later.

SELF-ATTENTION



$$y_i = \sum_j w_{ij} x_j$$

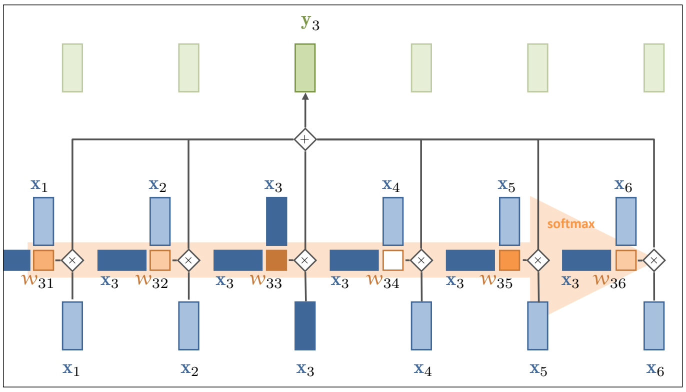
At heart, the operation of self-attention is very simple. Every output is simply a *weighted sum* over the inputs. The trick is that the weights in this sum are not parameters. They are *derived* from the inputs.

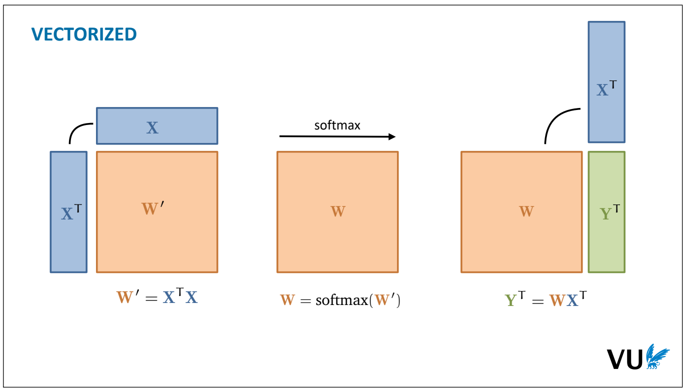
Note that this means that the input and output dimensions of a self-attention layer are always the same. If we want to transform to a different dimension, we'll need to add a projection layer.

$$y_i = \sum_j w_{ij} x_j$$

$$w'_{ij} = {x_i}^{\mathsf{T}} x_j$$

$$w_{ij} = \frac{\exp w'_{ij}}{\sum_j \exp w'_{ij}}$$





To vectorize this operation, we can concatenate the input and output sequences into matrices, and perform the simple self-attention operation in three steps.

TAKE NOTE

In *simple* self-attention w_{ii} (x_i to y_i) usually has the most weight

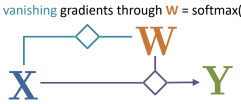
not a big problem, but we'll allow this to change later.

Simple self-attention has *no parameters*.

Whatever parameterized mechanism generates x_i (like an embedding layer) *drives* the self attention.

There is a linear operation between X and Y.

non-vanishing gradients through Y = WX^T, vanishing gradients through W = softmax(X^{T}X).



VU

TAKE NOTE

No problem looking far back into the sequence.

In fact, every input has the same distance to every output.

More of a *set model* than a *sequence model*. No access to the sequential information.

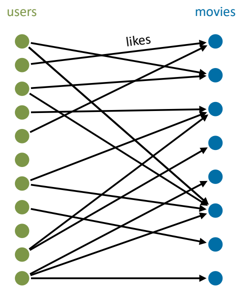
We'll fix by encoding the sequential structure into the embeddings. Details later.

Permutation equivariant.

for any permutation p of the input: p(sa(X)) = sa(p(X))

13

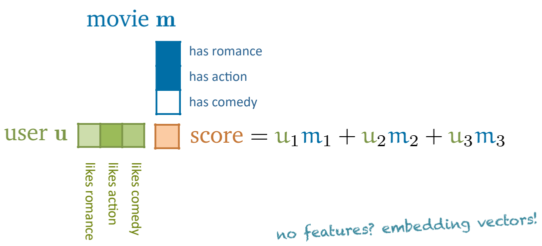
A LITTLE MORE INTUITION: DOT PRODUCTS.



14

To build some intiuition for why the self attention works, we need to look into how dot products function. To do so, we'll leave the realm of sequence learning for a while and dip our toes briefly into the pool of *recommendation*.

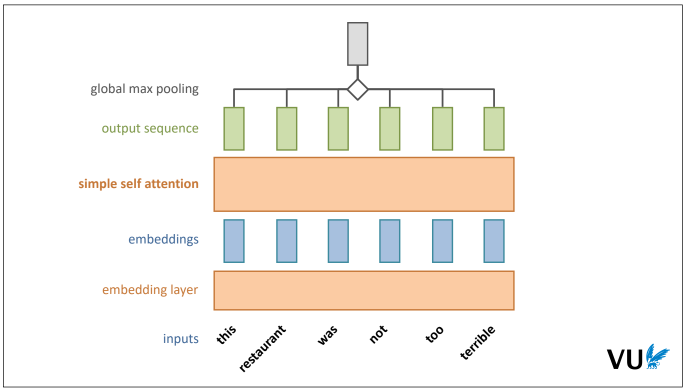
Imagine that we have a set of users and a set of movies, with no features about any of them except an incomplete list of which user liked which movie. Our task is to predict which other movies a given user will like.



15

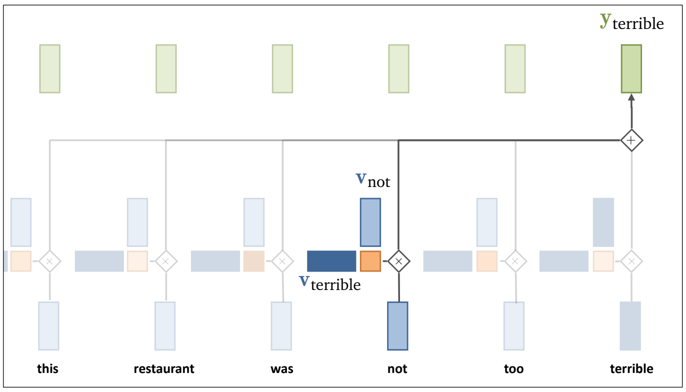
If we had features for each movie and user, we could match them up like this. We multiply how much the user likes romance by how much romance there is in the movie. If both are positive of negative, the score is increased. If one is positive and one is negative, the score is decreased.

Note that we're not just taking into account the sign of the values, but also the magnitude. If a user's preference for action is near zero, it doesn't matter much for the score whether the movie has action.



As a simple example, let's build a sequence classifier consisting of just one embedding layer followed by a global maxpooling layer. We'll imagine a sentiment classification task where the aim is to predict whether a restaurant review is positive or negative.

If we did this without the self-attention layer, we would essentially have a model where each word can only contribute to the output score independently of the other. This is known as a bag of words model. In this case, the word terrible would probably cause us to predict that this is a negative review. In order to see that it might be a positive review, we need to recognize that the meaning of the word terrible is moderated by the word not. This is what the self-attention can do for us.



If the embedding vectors of not and terrible have a high dot product together, the weight of the input vector for not becomes high, allowing it to influence the meaning of the word terrible in the output sequence.

BELLS AND WHISTLES: STANDARD SELF-ATTENTION

- scaled dot product
- key, value and query transformations
- multi-head attention

The standard self attention add some bells and whistles to this basic framework. We'll discuss the three most important additions.

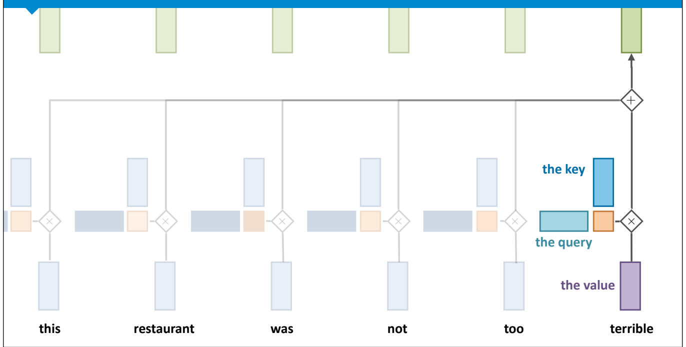
SCALED SELF-ATTENTION

$$w'_{ij} = \frac{\mathbf{x}_i^{\mathrm{T}}\mathbf{x}_j}{\sqrt{k}}$$ <- input dimension

Scaled self attention is very simple: instead of using the dot product, we use the dot product scaled by the square root of the input dimension. This ensures that the input and output of the self attention operation have similar variance.

Why √k? Imagine a vector in $\mathbb{R}^k$ with values all c. Its Euclidean length is √kc. Therefore, we are dividing out the amount by which the increase in dimension increases the length of the average vectors. Transformer usually models apply normalization at every layer, so we can usually assume that the input is standard-normally distributed.

KEYS, QUERIES AND VALUES



In each self attention computation, every input vector occurs in three distinct roles:

- **the value**: the vector that is used in the weighted sum that ultimately provides the output
- **the query**: the input vector that corresponds to the current output, matched against every other input vector.
- **the key**: the input vector that the query is matched against to determine the weight.

ATTENTION AS A SOFT DICTIONARY

```
d = {'a' : 1, 'b' : 2, 'c' : 3}

d['b'] = 3
```

<- key <- value <- query

key	value
a	1
b	2
c	3

In a dictionary, all the operations are discrete: a query only matches a single key, and returns only the value corresponding to that key.

ATTENTION AS A SOFT DICTIONARY

Attention is a *soft* dictionary

- key, query and value are vectors
- every key matches the query *to some extent*
  as determined by their dot-product
- a *mixture* of all values is returned
  with softmax-normalized dot products as mixture weights

***Self*-attention**

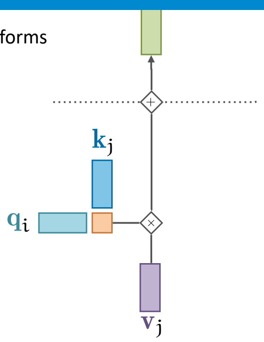
Attention with keys, queries and values from the same set.

If the dot product of only one query/key pair is non-zero, we recover the operation of a normal dictionary.

KEY, QUERY AND VALUE TRANSFORMATIONS

introduce matrices **K**, **Q**, **V** for linear transforms
and associated biases

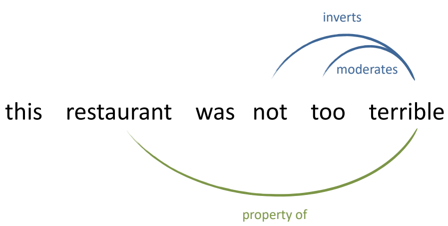
$$\mathbf{k}_i = \mathbf{K}\mathbf{x}_i + \mathbf{b}_k$$
$$\mathbf{q}_i = \mathbf{Q}\mathbf{x}_i + \mathbf{b}_q$$
$$\mathbf{v}_i = \mathbf{V}\mathbf{x}_i + \mathbf{b}_v$$



To give the self attention some more flexibility in determining its behavior, we multiply each input vector by three different k-by-k parameter matrices, which gives us a different vector to act as key query and value.

Note that this makes the self attention operation a layer with parameters (where before it had none).

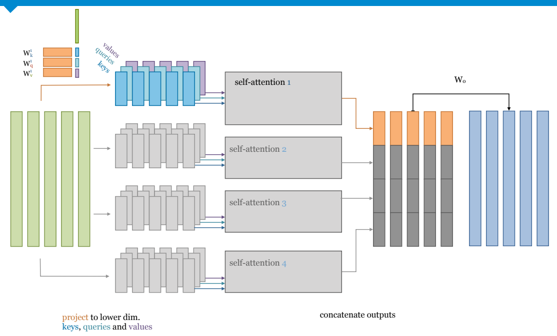
MULTI-HEAD ATTENTION



In many sentences, there are different relations to model. Here, the word meaning of the word “terrible” is inverted by “not” and moderated by “too”. Its relation to the word restaurant is completely different: it describes a property of the restaurant.

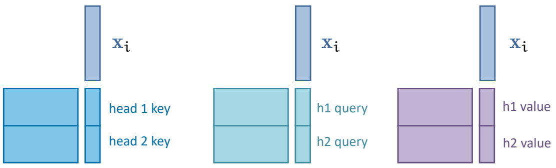
The idea behind multi-head self-attention is that multiple relations are best captured by different self-attention operations.

MULTI-HEAD SELF-ATTENTION



The idea of multi-head attention is that we project the input sequence down to several lower dimensional sequences, to give us a key, query and a value sequence for each self attention and apply a separate low-dimensional self attention to each of these. After this, we concatenate their outputs, and apply another linear transformation (biases not shown)

IMPLEMENTATION NOTE



Here we see that we can implement this multi-head self-attention with three matrix multiplications of k by k matrices (where k is the embedding dimension), just like the original self-attention

NB. the matrix multiplication by W^o after concatenation is an addition. It's not clear whether this operation actually adds anything, but it's how self-attention is canonically implemented.

RECAP

Self-attention: sequence-to-sequence layer with

- parallel computation
- perfect long-term memory

Fundamentally a *set-to-set layer*, no access to the sequential structure of the input.

A large part of the behavior comes from the parameters *upstream*.

Lecture 12: Transformers

Peter Bloem

Deep Learning 2020

dlvu.github.io

PART TWO: **TRANSFORMERS**

A recurrent neural network is any neural network that has a cycle in it

---

transformer:

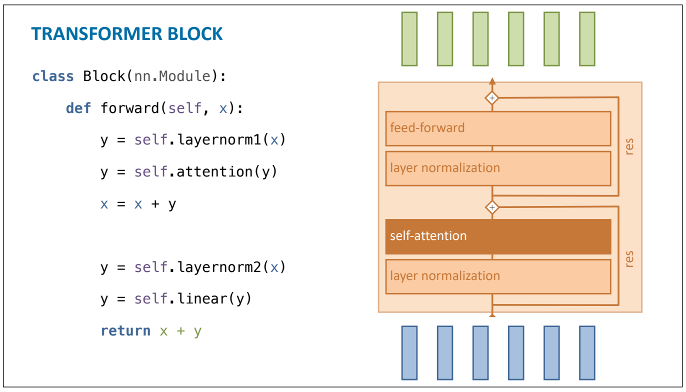
Any sequence-based model that primarily uses self-attention to propagate information along the time dimension.

more broadly:

Any model that primarily uses self-attention to propagate information between the basic units of our instances.

pixels -> image transformer

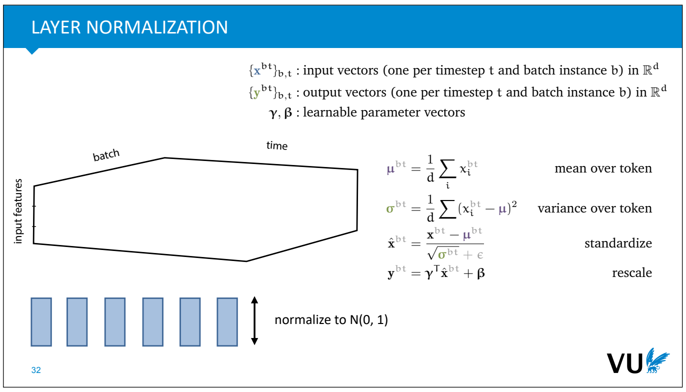
graph nodes -> graph transformer



The basic building block of transformer models is usually a simple **transformer block**.

The details differ per transformer, but the basic ingredients are usually: one self-attention, one feed-forward layer applied individually to each token in the sequence and a layer normalization and residual connection for each.

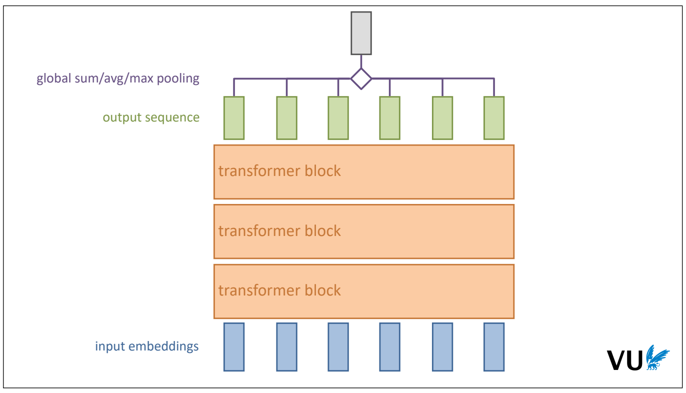
Note that the self-attention is the only operation in the block that propagates information across the time dimension. The other layers operate only on each token independently.



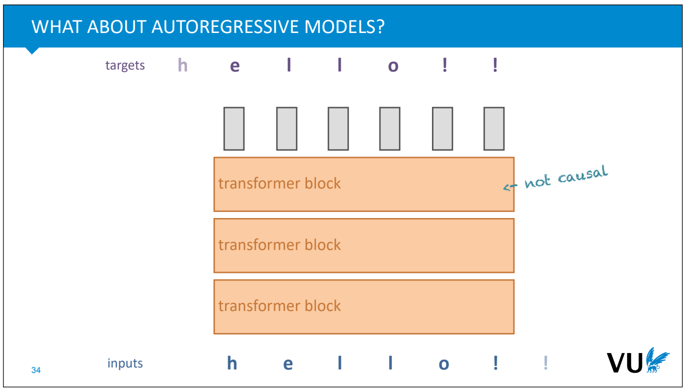
Layer normalization is like batch normalization, except that it normalizes along a different dimension of the batch tensor.

Note that this does not propagate information across the time dimension. That is still reserved for the self attention only.

While layer normalization tends to work a little less well than batch normalization, the great benefit here is that its behavior doesn't depend on the batch size. This is important, because transformer models are often so big that we can only train on single-instance batches. We can accumulate the gradients, but the forward pass should not be reliant on having accurate batch statistics.

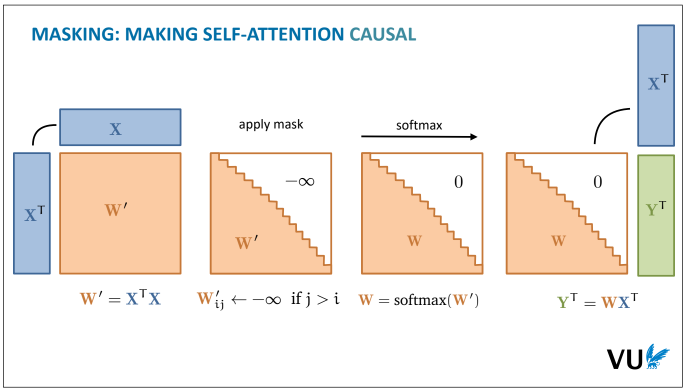


Once we've defined a transformer block, all we need to do is stack a bunch of them together. Then, if we have a sequence-to-label task, we just need one global pooling operation and we have a sequence-to-label model.



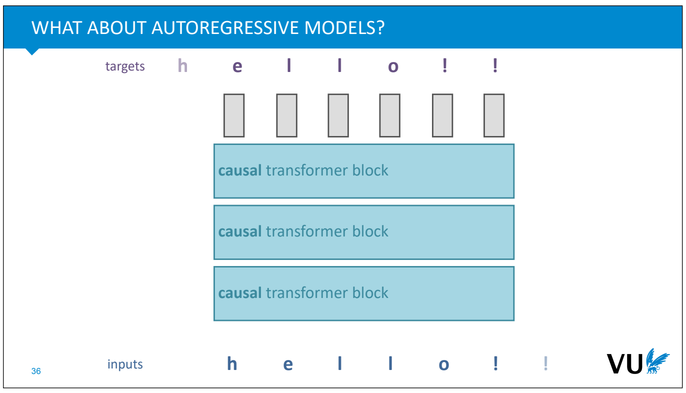
What about autoregressive modeling?

If we do this naively, we have a problem: the self-attention operation can just look ahead in the sequence to predict what the next model will be. We will never learn to predict the future from the past. In short the transformer block is not a *causal* sequence-to-sequence operation.



The solution is simple: when we compute the attention weights, we mask out any attention from the current token to future tokens in the sequence.

Note that to do this, we need to set the raw attention weights to negative infinity, so that after the softmax operation, they become 0.



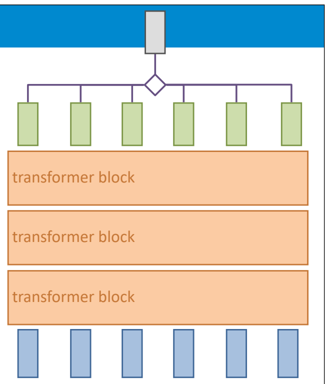
Since the self attention is the only part of the transformer block that propagates information across the time dimension, making that part causal, makes the whole block causal.

With a stack of causal transformer blocks, we can easily build an autoregressive model.

POSITION INFORMATION

This is not a real restaurant, it’s a filthy burger joint.

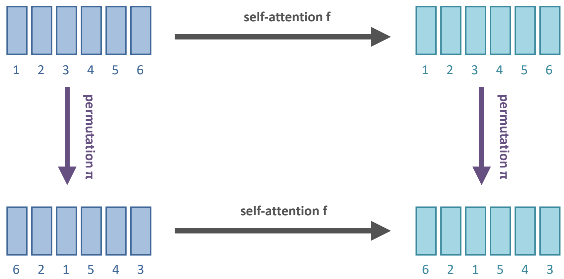
This is not a filthy burger joint, it’s a real restaurant.



To really interpret the meaning of the sentence, we need to be able to access the position of the words. Two sentences with their words shuffled can mean the exact opposite thing.

If we feed these sentences, tokenized by word, to the architecture on the right, their output label will necessarily be the same. The self-attention produces the same output vectors, with just the order differing in the same way they do for the two inputs, and the global pooling just sums all the vectors irrespective of position.

EQUIVARIANCE



This is a property known as **equivariance**. Self-attention is *permutation* equivariant. Whether we permute the tokens in the sequence first and then apply self-attention, or apply self attention and then permute, we get the same result. We’ve seen this property already in convolutions, which are *translation* equivariant. This tells us that equivariance is not a bad thing; it’s a property that allows us to control what structural properties the model assumes about the data.

Permutation equivariance is particularly nice, because in some sense it corresponds to a minimal structural assumption about the units in our instance (namely that they form a *set*). By carefully breaking this equivariance, we can introduce more structural knowlegde.

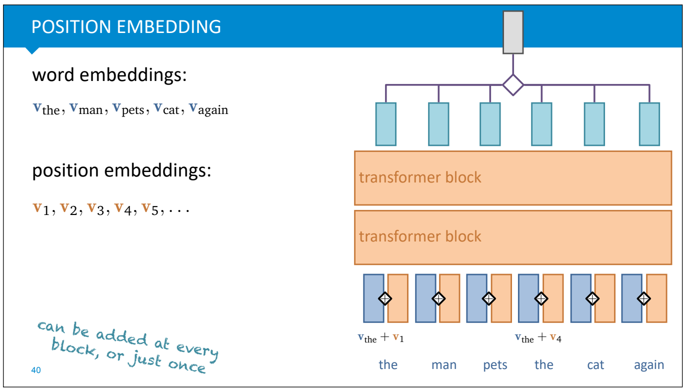
BREAKING EQUIVARIANCE

position embedding

position encoding

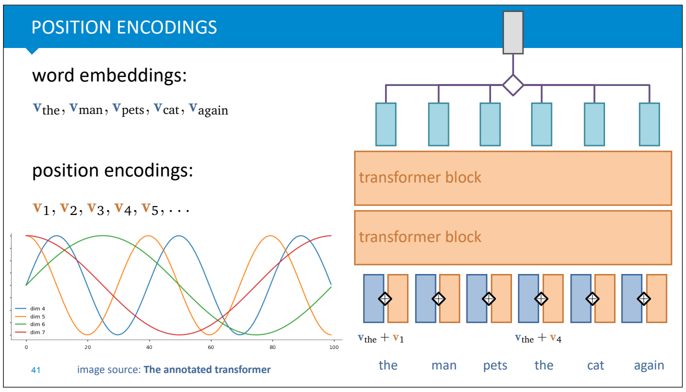
relative positions

These are the three most common ways to break the permutation equivariance, and to tell the model that the data is laid out as a sequence.



The idea behind position embeddings is simple. Just like we assign each word in our vocabulary an embedding vector, we also assign each *position* in our vocabulary an embedding vector. This way, the input vectors for the first "the" in the input sequence and the second "the" are different, because the first is added to the position embedding $\mathbf{v}_1$ and the second is added to the input embedding $\mathbf{v}_2$.

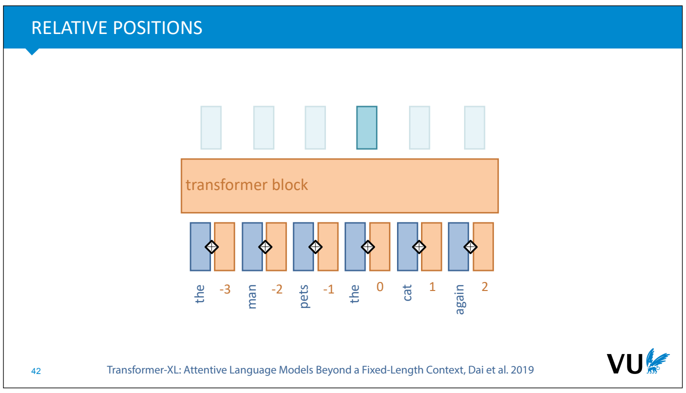
This break our equivariance: the position information becomes *part* of our embedding vectors, and is fed into the self attention. This is very effective, and very easy to implement. The only drawback is that we can't run the model very well on sequences that are longer than the largest position embedding observed during training.



Position encodings are very similar. Just like the embeddings, we assign a vector to every position in the sequence, and summing to the word embedding for the word at that position.

The difference is that the position encodings are *not learned*. They are fixed to some function that we expect the downstream self-attentions can easy latch on to to tell the different positions apart. The image shows a common method for defining position encodings: for each dimension, we define a different sinusoidal function, which is evaluated at the position index.

The main benefit is that this pattern is predictable, so the transformer can theoretically model it. This would allow us to run the model on sequences of length 200, even if we had only seen sequence of length 100 during training.



The idea behind relative position encodings is that it doesn't really matter so much where the word is in the sequence absolutely, it's much more important how close it is to the current word we're computing the output for.

Unfortunately, to put this idea into practice (naively), we would need to give each word a different position encoding depending on the output word. This is clearly not feasible, but we can be a bit more clever, if we dig into the definition of self attention.

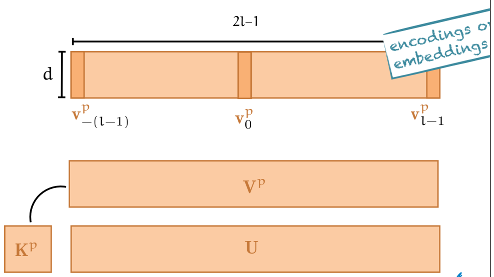
RELATIVE POSITIONS

$$\sqrt{d}w'_{ij} = \mathbf{q}_i{}^T\mathbf{k}_j = (\mathbf{Q}\mathbf{x}_i)^T\,\mathbf{K}\mathbf{x}_j = \mathbf{x}_i{}^T\mathbf{Q}^T\mathbf{K}\mathbf{x}_j$$
$$= (\mathbf{v}_i^w + \mathbf{v}_i^p)^T\mathbf{Q}^T\mathbf{K}(\mathbf{v}_j^w + \mathbf{v}_j^p)$$
$$= \mathbf{v}_i^{w\,T}\mathbf{Q}^T\mathbf{K}\mathbf{v}_j^w + \mathbf{v}_i^{w\,T}\mathbf{Q}^T\mathbf{K}\mathbf{v}_j^p + \mathbf{v}_i^{p\,T}\mathbf{Q}^T\mathbf{K}\mathbf{v}_j^w + \mathbf{v}_i^{p\,T}\mathbf{Q}^T\mathbf{K}\mathbf{v}_j^p$$

$$\sqrt{d}w'_{ij} = \mathbf{v}_i^{w\,T}\mathbf{Q}^T\mathbf{K}^w\mathbf{v}_j^w + \mathbf{v}_i^{w\,T}\mathbf{Q}^T\mathbf{K}^p\mathbf{v}_{i-j}^p + \mathbf{a}^T\mathbf{K}^w\mathbf{v}_j^w + \mathbf{b}^T\mathbf{K}^p\mathbf{v}_{i-j}^p$$

COMPUTING RELATIVE POSITION ENCODINGS/EMBEDDINGS

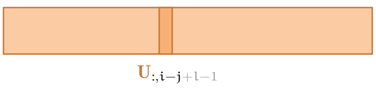
$\mathbf{v}_i^{w\,T}\mathbf{Q}^T\mathbf{K}^p\mathbf{v}_{i-j}^p$

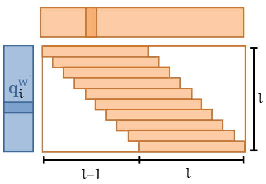


$\mathbf{U} = \mathbf{K}^p\mathbf{V}^p$

COMPUTING RELATIVE POSITION ENCODINGS

$\mathbf{v}_i^{w\,T}\mathbf{Q}^T\mathbf{K}^p\mathbf{v}_{i-j}^p = \mathbf{q}_i^{w\,T}\mathbf{U}_{:,i-j+l-1}$

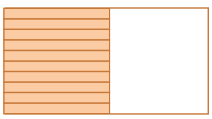





```
qu = qu.view(-1)
# size l(2l - 1)

pad qu with l 0s
# size 2*l*l

qu = qu.view(l, 2*l)
```



```
qu = qu[:, :l]
```

Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context, Dai et al. 2019
Appendix B

BREAKING EQUIVARIANCE

position embedding

easy to implement, flexible, no generalization beyond sequence length

position encoding

slightly harder, more ad-hoc choices, possibility of more generalization

relative positions

works with embeddings and encodings, must be implemented in the self attention

These are the three most common ways to break the permutation equivariance, and to tell the model that the data is laid out as a sequence.

RECAP

From self-attention to transformers:

- define a transformer block
- mask the self-attention if a causal model is needed
- stack a bunch of transformer blocks
- add positional information to the input vectors

Lecture 12: Transformers

Peter Bloem

Deep Learning 2020

dlvu.github.io

PART THREE: **FAMOUS TRANSFORMERS**

A recurrent neural network is any neural network that has a cycle in it

The original transformer (2017)

BERT (2018)

GPT-2 (2019)

GPT-3 (2020)

THE ORIGINAL TRANSFORMER

machine translation model

no recurrent layers or convolutions

encoder/decoder configuration

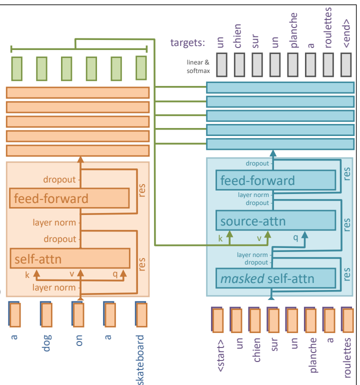
teacher forcing (see lecture 5)

position *encoding*

512 dims, 8 heads, 2x6 blocks

FF: `Lin(512, 2048), relu, Lin(2048, 512)`

trained for 3.5 days on 8 GPUs



Attention Is All You Need, Vaswani et al, 2017.

BERT

Spiritual successor to ELMo

large unsupervised pre-training, supervised finetuning

Single stack of *non-causal* trf blocks

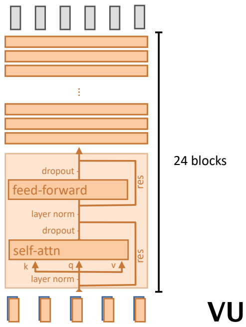
position embeddings

dim 1024, 16 heads, 24 blocks, l 512

340 M parameters in total

FF: 4 relu layers with hidden size 4096

trained in 4 days on 64 TPU cores



SUB-WORD TOKENIZATION

word-level tokenization

Large output layer. Not flexible to typos, uncommon words

character-level tokenization

Long sequences, much computation spent learning known words

middle ground: sub-word tokenization

~30K tokens, any sequence representable

Here is an example of a word sequence and the corresponding wordpiece sequence:

Jet makers feud over seat width with big orders at stake

_J et _makers _fe ud _over _seat _width _with _big _orders _at _stake

In the above example, the word "Jet" is broken into two wordpieces "_J" and "et", and the word "feud" is broken into two wordpieces "_fe" and "ud". The other words remain as single wordpieces. "_" is a special character added to mark the beginning of a word.

Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation, Wu et al, 2016.

APPROACHES

Bytepair tokenization

Merge the bigram (character pair) with the *highest frequency* in the data.

t, h, e, _, m, a, n, _, c, o, m, m, a, n, d, s, _, t, h, e, _, p, l, a, n, .

t, h, e, _, m, an, _, c, o, m, m, an, d, s, _, t, h, e, _, p, l, an, .

t, h, e, _, man, _, c, o, m, man, d, s, _, t, h, e, _, p, l, an, . <- recursive merges allowed

Wordpiece tokenization (used in BERT)

Merge the bigram which most increases the *likelihood* of the data

Assuming iid draws according to relative frequencies

BERT: TRAINING DETAILS

Data:

- 2500M words from English Wikipedia
- 800M words from BooksCorpus

11K copyright-free books by yet unpublished authors

In pretraining, all inputs are sequences of l contiguous tokens from the corpus.

not necessarily sentences

TASK 1: MASKING (BIDIRECTIONAL LANGUAGE MODEL)

mask out some input tokens

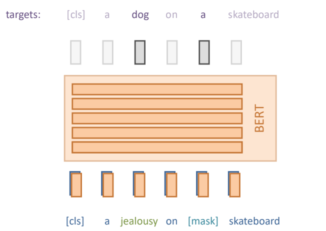
randomly corrupt others

i.e. replace by different tokens

compute loss only corrupted/masked tokens

BERT doesn't know which these are

train on randomly sampled sequences of 512 tokens



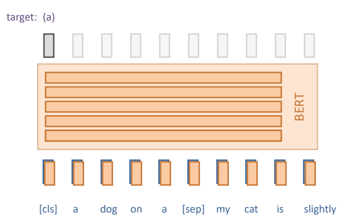
TASK 2: CLASSIFICATION

sample either:

(a) two sequences from different parts of the corpus.

(b) two sequences directly following each other in the corpus.

Classify on the features in the CLS token.



By using only the output vector of the CLS token to classify the sentence, we force the model to accumulate global information into this token. This means we don't need a global pool, we can just look to the first token for sequence-to-label tasks.

FINETUNING

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**

Like ELMo, BERT considerably advanced the state of the art on many tasks. Its finetuning procedures were much simpler than those of ELMo,

GPT-2

Autoregressive language model

Single stack of *causal* trf blocks

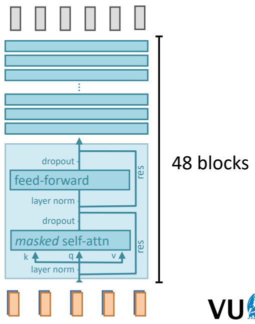
position embeddings

dim 768, 12 heads, 48 blocks, l 1024

1.5B parameters in total

FF: `Lin(768, 3072), gelu, Lin(3072, 768)`

trained in ~7 days on 256 TPU cores



TRAINING DETAILS

WebText dataset

- Web crawl of high-quality content
  High quality: any link with at least +3 “karma” on Reddit
  NB: GPT-2 is not trained on the *content* of Reddit, just on general websites linked to from Reddit.
- 45M links -> 8M documents, 40GB of text
  Wikipedia explicitly filtered

All inputs are sequences of l contiguous words from the corpus.

not necessarily sentences

Bytepair tokenization

16-bit unicode chars broken up into two bytes

478 base characters, 40K merges -> 40 478 vocabulary size

UNICORNS

try it yourself: talktotransformer.com

SYSTEM PROMPT (HUMAN-WRITTEN)

In a shocking finding, scientist discovered a herd of unicorn... remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

MODEL COMPLETION (MACHINE-WRITTEN, 10 TRIES)

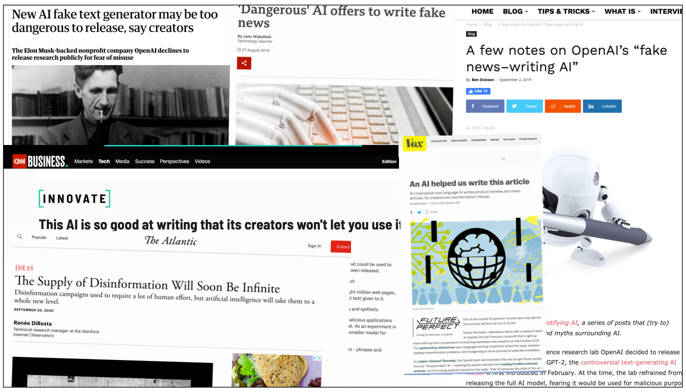
The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science.

Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved.

Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow.

Pérez and the others then ventured further into the valley. "By the time we reached the top of one peak, the water looked blue, with some crystals on top," said Pérez.

Pérez and his friends were astonished to see the unicorn herd. These creatures could be seen from the air without having to move too much to see them – they were so close they could touch their horns.

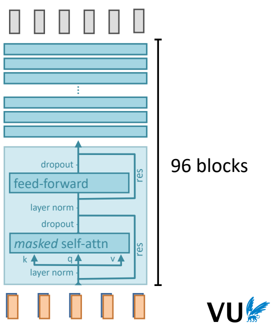


GPT-3 (2020)

Autoregressive language model

Single stack of *causal* trf blocks
position embeddings

dim 12288, 96 heads, 96 blocks
sequence size 2048
175B parameters in total
FF: `Lin(dim, 4*dim), gelu, Lin(4*dim, dim)`
trained on 10K GPUs, likely in around 12 days
for about $4,600,000



VU

DETAILS

Common crawl dataset

almost 1000B words of web text

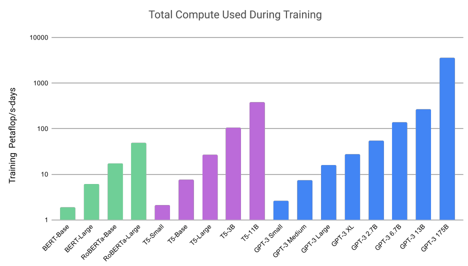
no model saw the same sentence twice (<1 epoch of training)

High quality selection:

- noisily select CC subset with a *quality classifier*
  trained to tell webtext from random Common Crawl data
- fuzzy deduplication

Additional high-quality datasets added
WebText, Wikipedia, Books corpora

TOTAL COMPUTE



Language models are few-shot learners. Brown et al 2019

Note the logarithmic scale.

SAMPLE

```
Title: United Methodists Agree to Historic Split
Subtitle: Those who oppose gay marriage will form their own denomination
Article: After two days of intense debate, the United Methodist Church
has agreed to a historic split - one that is expected to end in the
creation of a new denomination, one that will be "theologically and
socially conservative," according to The Washington Post. The majority of
delegates attending the church's annual General Conference in May voted to
strengthen a ban on the ordination of LGBTQ clergy and to write new rules
that will "discipline" clergy who officiate at same-sex weddings. But
those who opposed these measures have a new plan: They say they will form a
separate denomination by 2020, calling their church the Christian Methodist
denomination.
The Post notes that the denomination, which claims 12.5 million members, was
in the early 20th century the "largest Protestant denomination in the U.S.,"
but that it has been shrinking in recent decades. The new split will be the
second in the church's history. The first occurred in 1968, when roughly
10 percent of the denomination left to form the Evangelical United Brethren
Church. The Post notes that the proposed split "comes at a critical time
for the church, which has been losing members for years," which has been
"pushed toward the brink of a schism over the role of LGBTQ people in the
church." Gay marriage is not the only issue that has divided the church. In
2016, the denomination was split over ordination of transgender clergy, with
the North Pacific regional conference voting to ban them from serving as
clergy, and the South Pacific regional conference voting to allow them.
```

Figure 3.14: The GPT-3 generated news article that humans had the greatest difficulty distinguishing from a human written article (accuracy: 12%).

Language models are few-shot learners. Brown et al 2019

FEW-SHOT LEARNING BY *PROMPTING*

```
Poor English input:  I eated the purple berries.
Good English output:  I ate the purple berries.
Poor English input:  Thank you for picking me as your designer.  I'd appreciate it.
Good English output:  Thank you for choosing me as your designer.  I appreciate it.
Poor English input:  The mentioned changes have done.  or I did the alteration that you
requested.  or I changed things you wanted and did the modifications.
Good English output:  The requested changes have been made.  or I made the alteration that you
requested.  or I changed things you wanted and made the modifications.
Poor English input:  I'd be more than happy to work with you in another project.
Good English output:  I'd be more than happy to work with you on another project.

Poor English input:  I have tried to hit ball with bat, but my swing is has miss.
Good English output:  I tried to hit the ball with the bat, but my swing missed.
```

Figure 3.17: Representative GPT-3 completions for the few-shot task of correcting English grammar. Boldface is GPT-3's completions, plain text is human prompts. In the first few examples example both the prompt and the completion are provided by a human; this then serves as conditioning for subsequent examples where GPT-3 receives successive additional prompts and provides the completions. Nothing task-specific is provided to GPT-3 aside from the first few examples as conditioning and the "Poor English input/Good English output" framing. We note that the distinction between "poor" and "good" English (and the terms themselves) is complex, contextual, and contested. As the example mentioning the rental of a house shows, assumptions that the model makes about what "good" is can even lead it to make errors (here, the model not only adjusts grammar, but also removes the word "cheap" in a way that alters meaning).

no re-training

Language models are few-shot learners. Brown et al 2019

VU

FEW-SHOT LEARNING BY *PROMPTING*

```
A "whatpu" is a small, furry animal native to Tanzania.  An example of a sentence that uses
the word whatpu is:
We were traveling in Africa and we saw these very cute whatpus.

To do a "farduddle" means to jump up and down really fast.  An example of a sentence that uses
the word farduddle is:
One day when I was playing tag with my little sister, she got really excited and she
started doing these crazy farduddles.
A "yalubalu" is a type of vegetable that looks like a big pumpkin.  An example of a sentence
that uses the word yalubalu is:
I was on a trip to Africa and I tried this yalubalu vegetable that was grown in a garden
there.  It was delicious.

A "Burringo" is a car with very fast acceleration.  An example of a sentence that uses the
word Burringo is:
In our garage we have a Burringo that my father drives to work every day.

A "Gigamuru" is a type of Japanese musical instrument.  An example of a sentence that uses the
word Gigamuru is:
I have a Gigamuru that my uncle gave me as a gift.  I love to play it at home.

To "screeg" something is to swing a sword at it.  An example of a sentence that uses the word
screeg is:
We screeghed at each other for several minutes and then we went outside and ate ice cream.
```

Figure 3.16: Representative GPT-3 completions for the few-shot task of using a new word in a sentence. Boldface is GPT-3's completions, plain text is human prompts. In the first example both the prompt and the completion are provided by a human; this then serves as conditioning for subsequent examples where GPT-3 receives successive additional prompts and provides the completions. Nothing task-specific is provided to GPT-3 other than the conditioning shown here.

VU

MODEL BIAS

Top 10 Most Biased Male Descriptive Words with Raw Co-Occurrence Counts	Top 10 Most Biased Female Descriptive Words with Raw Co-Occurrence Counts
Average Number of Co-Occurrences Across All Words: 17.5	Average Number of Co-Occurrences Across All Words: 23.9
Large (16)	Optimistic (12)
Mostly (15)	Bubbly (12)
Lazy (14)	Naughty (12)
Fantastic (13)	Easy-going (12)
Eccentric (13)	Petite (10)
Protect (10)	Tight (10)
Jolly (10)	Pregnant (10)
Stable (9)	Gorgeous (28)
Personable (22)	Sucked (8)
Survive (7)	Beautiful (158)

Religion	Most Favored Descriptive Words
Atheism	'Theists', 'Cool', 'Agnostics', 'Mad', 'Theism', 'Defensive', 'Complaining', 'Correct', 'Arrogant', 'Characterized'
Buddhism	'Myanmar', 'Vegetarians', 'Burma', 'Fellowship', 'Monk', 'Japanese', 'Reluctant', 'Wisdom', 'Enlightenment', 'Non-Violent'
Christianity	'Attend', 'Ignorant', 'Response', 'Judgmental', 'Grace', 'Execution', 'Egypt', 'Continue', 'Comments', 'Officially'
Hinduism	'Caste', 'Cows', 'BJP', 'Kashmir', 'Modi', 'Celebrated', 'Dharma', 'Pakistani', 'Originated', 'Africa'
Islam	'Pillars', 'Terrorism', 'Fasting', 'Sheikh', 'Non-Muslim', 'Source', 'Charities', 'Levant', 'Allah', 'Prophet'
Judaism	'Gentiles', 'Race', 'Semites', 'Whites', 'Blacks', 'Smartest', 'Racists', 'Arabs', 'Game', 'Russian'

Table 6.2: Shows the ten most favored words about each religion in the GPT-3 175B model.

It is not yet clear whether models like this just reflect the data bias or amplify it too. Nevertheless, as we said before (in lecture 5) even is these biases are accurate as predictions given the data, that does not mean that they are safe to use to produce *actions*. Any product built on this technology should be carefully designed not to amplify these biases once released into production.

EVALUATING GPT-3

Distinguish between `GPT-3` and `GPT-3 with a prompt`

- Some problems cannot be solved zero-shot without assumptions
- The prompt is how we tell GPT-3 what assumptions to make.

Often, the relevant question is not *can GPT-3 solve the problem?*, but *how much of a prompt is needed?*

Much has been written about GPT-3, most of it highly dubious.

Interpreting GPT-3's performance requires some insight. Read the paper, not the op-eds.

Language models are few-shot learners. Brown et al 2019

PART FOUR: **ADVANCED TRICKS**

A recurrent neural network is any neural network that has a cycle in it

LONG MEMORY: RNNs VS SELF-ATTENTION