Courseworks Released

Due **13 March 4:30pm**

Coursework #1: Build a Text Classification System
20% of your final grade

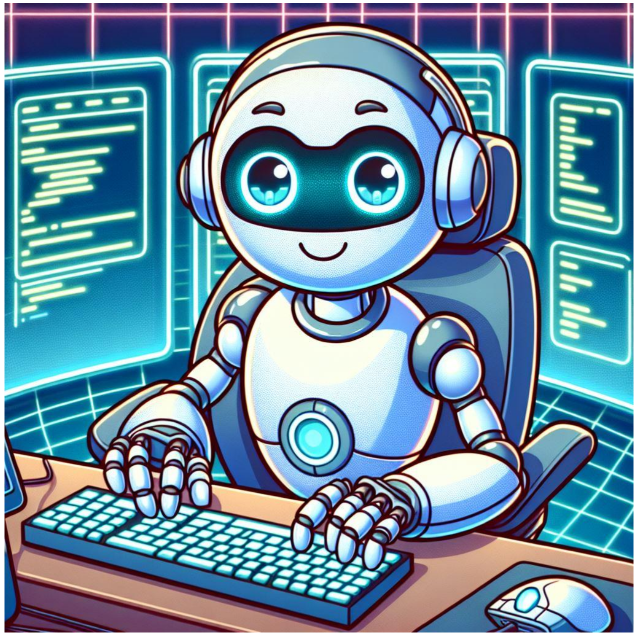
Coursework #2 (for MSci): Paper Review
10% of your final grade

Tuesday lab times can be used to catch up on labs and work on the coursework

Remember: All labs & courseworks are examinable

You can use code from the labs (including scikit-learn/transformers code)

Generative AI policy & Plagiarism


created with Copilot and DALL-E 3

- You are free to use generative AI in any way during the assessed exercise(s)
- Please note your usage in the final question
- Material from this coursework may appear on the final exam

Plagiarism Policy: Your work should be your own. It's better to skip a question than copy a friend's answer

Contextual Word Embeddings: BERT and Beyond

Jake Lever & Sean MacAvaney

University of Glasgow | School of Computing Science

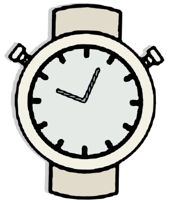
Text As Data

What are we going to learn today?

- Representing individual words as vectors
- Why it’s important to consider the context words appear in
- Neural Language Models (e.g., BERT, GPT) and the techniques behind them:
    - Self-Attention
    - Sub-word Tokenization
    - Transformers
    - Pre-training, Fine-tuning

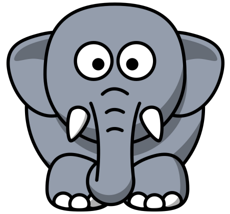
So far, we've considered document similarity

A: I checked the time on my watch. `{check: 1, time: 1, watch: 1}`



B: I checked the time on my clock. `{check: 1, time: 1, clock: 1}`

C: I saw an elephant at the zoo. `{saw: 1, elephant: 1, zoo: 1}`

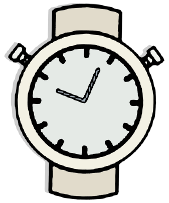


`sim(A, B) > sim(A, C)`

(using TF vectors)

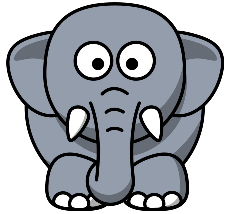
So far, we’ve considered document similarity

A: I checked the time on my watch. `{check: 1, time: 1, watch: 1}`



B: I checked the time on my clock. `{check: 1, time: 1, clock: 1}`

C: I checked the time on my elephant. `{check: 1, time: 1, elephant: 1}`

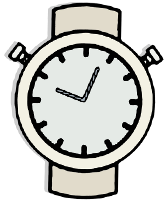


⚠️ `sim(A, B) = sim(A, C)`

(using TF vectors)

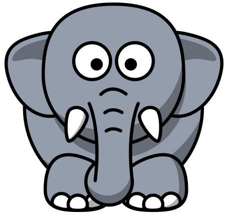
It’d be helpful if we could also tell if individual words were similar…

watch



clock

elephant



```
sim(watch, clock) > sim(watch, elephant)
```

Word Vectors (Embeddings)

Exercise: Based on the text, what is a **tarant**? How do you know?

"Alice stepped into the furniture store, overwhelmed by the selection of furnishings. She had been searching for the perfect **tarant** for her living room for weeks, and so far, she hadn't found anything that fit the bill. Alice rounded a corner and saw it: a small, cozy **tarant** tucked away in the corner. Its deep blue fabric blended beautifully with the light wood of its frame. It looked inviting and comfortable, and it was exactly the style she had been searching for. Alice moved closer and ran her hands along the **tarant**'s fabric."

Post your ideas to the Padlet:
https://padlet.com/jakelever/tad2025_8

John Rupert Firth

"You shall know a word by the company it keeps."

- J.R. Firth, Studies in Linguistic Analysis, 1957

Strategy: Represent each word as the context it appears in

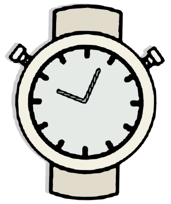
Sliding context window

…and other **settings to all your devices** **watch** **video anything you update on** one device…

…replace your **smartphone and if the apple** **watch** **follows the pattern of other** apple…

…each gallons **of fill water with a** **watch** **time the amount of time** it takes to fill up…

…



```
watch = {video: 3168, apple: 1702, time: 868, …}
```

Strategy: Represent each word as the context it appears in

…data is **plotted against elapsed time or** **clock** **time on a graph users can** export the…

…cost **of massage therapy school is** **clock** **hours rates range from to** per…

…there **are buttons on the alarm** **clock** **that will control the basic** clock…

…

```
clock = {time: 2614, hour: 806, alarm: 438, …}
```

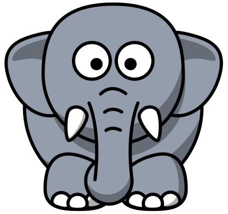
Strategy: Represent each word as the context it appears in

…about **how much does a baby** elephant **weigh at birth the weight** of an elephant…

… take **the average for all the** elephant **species and subspecies and average** that…

…in related **species that also are called** elephant **ears for example the purplish** black…

…



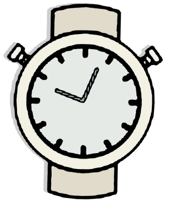
```
elephant = {baby: 108, species: 100, ears: 97, …}
```

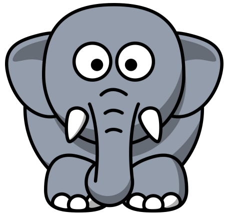
Distributional Word Vectors

```
watch = {video: 3168, apple: 1702, time: 868, …}

clock = {time: 2614, hour: 806, alarm: 438, …}

elephant = {baby: 108, species: 100, ears: 97, …}
```





We can represent each word as a sparse vector of a context window taken around each occurrence of it in the corpus.

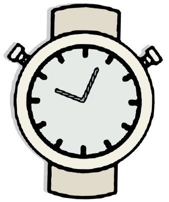
This is called the **IBM Model**

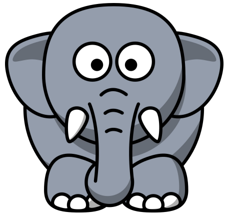
We can compare these vectors!

```
watch = {video: 3168, apple: 1702, time: 868, …}

clock = {time: 2614, hour: 806, alarm: 438, …}

elephant = {baby: 108, species: 100, ears: 97, …}
```





E.g., via cosine similarity:

```
sim(watch, clock) > sim(watch, elephant)
```

Improvements can be made by using the tricks we already know, e.g. TF-IDF

Problem with these sparse vectors

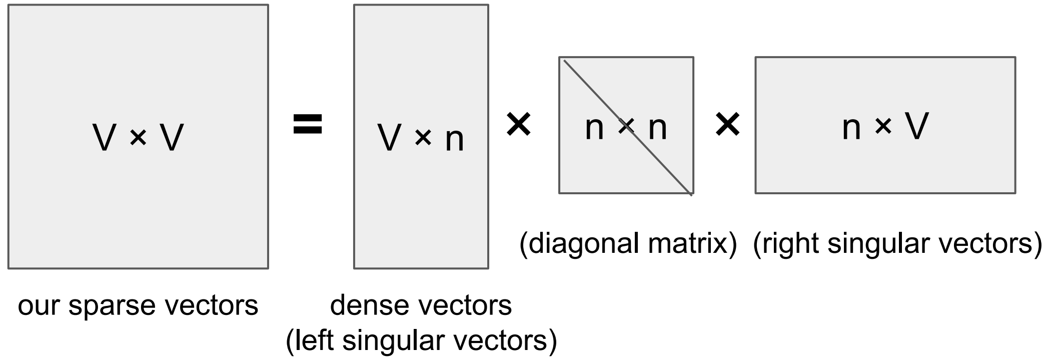
```
|watch|    = 21,916    watch = {video: 3168, apple: 1702, …}
|clock|    = 9,093     clock = {time: 2614, hour: 806, …}
|elephant| = 5,477     elephant = {baby: 108, species: 100, …}
```

(here `|word|` represents the number of non-zero values of `word`'s vector)

- The vectors are generally very large!
  - Memory and compute intensive

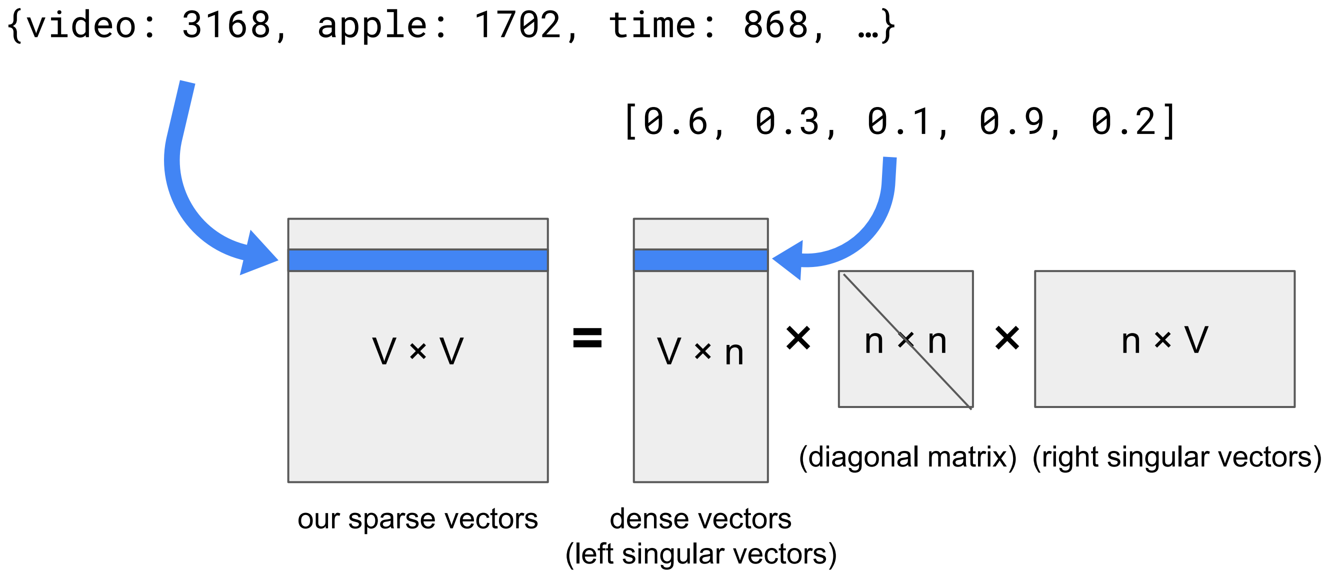
“Compress” the sparse vectors into dense vectors

We can reduce the dimensionality by applying dimensionality reduction techniques like Truncated Singular Value Decomposition (SVD):



Matrix factorization approaches turn a matrix into a product of multiple matrices (which may be smaller or have other nice properties)

Truncated Singular Value Decomposition (SVD)



In practice, n is usually in the hundreds or low thousands

These are often called **static word embeddings / vectors**

watch	`[0.6, 0.3, 0.1, 0.9, 0.2]`
clock	`[0.5, 0.3, 0.2, 0.9, 0.1]`
elephant	`[0.1, 0.9, 0.9, 0.2, 0.3]`

Maintain the useful properties of the sparse vectors, such as:

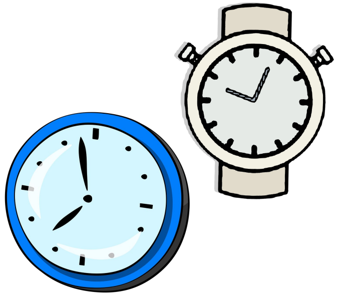
```
sim(watch, clock) > sim(watch, elephant)
```

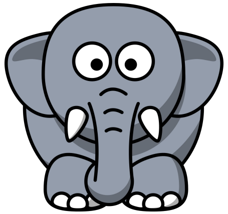
Various other techniques available to construct static word embeddings:
- Word2Vec
- GloVe

Advantage of static word embeddings

✅ Handles **synonymy** – when two words have similar/identical meanings.

- Similar words will have similar embeddings; dissimilar words will have dissimilar embeddings.
- Based on the assumption that synonymous words will appear in similar contexts across a corpus.

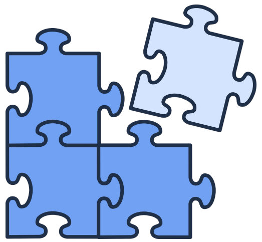




Disadvantage of static word embeddings

❌ Don’t handle **polysemy** – when one word has multiple meanings.

- Static word embeddings always map the same word to the same embedding.
- Embedding is the weighted average across all contexts (which can be multiple meanings)
    - Less frequent meanings are under-represented.

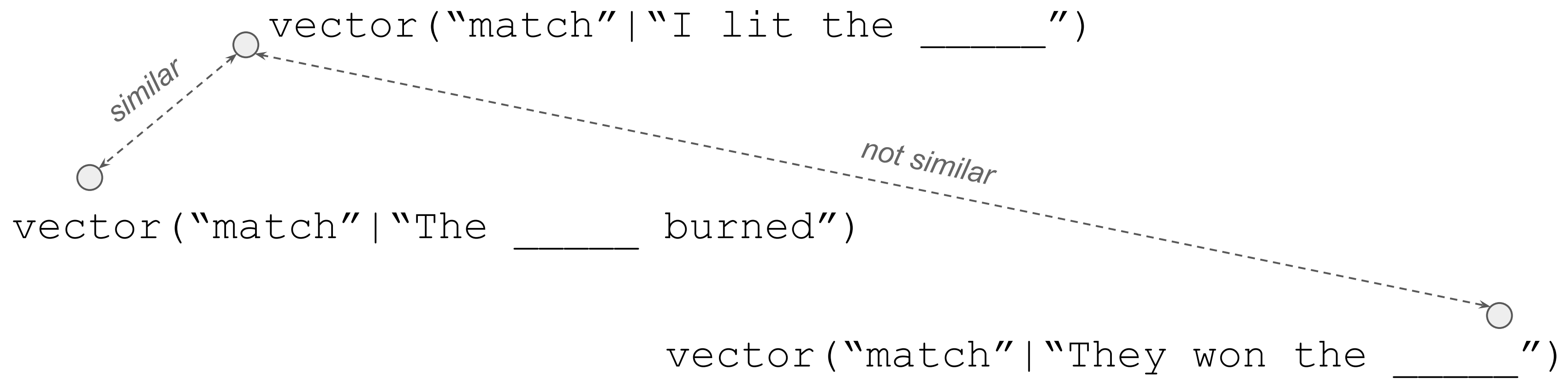
`vector(“match”)=` + +  + ...

Integrating Context into Vectors

Contextual Word Vectors (or Context Vectors for short)

Rather than `vector(word)`, we want `vector(word|context)`

We'll turn to **deep learning** to accomplish this.



vector("match"|"I lit the _____")
similar
not similar
vector("match"|"The _____ burned")
vector("match"|"They won the _____")

Early work on context vectors: ELMo

- Used (now older) deep learning methods for a neural language model
- Found that the internal representations in the neural model were good context vectors
  - Embeddings from Language Models
- The context vectors were useful for other language tasks
  - Classification, document similarity, etc

https://aclanthology.org/N18-1202.pdf

The big innovations

- Self-Attention – a neural network structure that builds a new word representation based on its context
- Subword-tokenization – limits the size of the vocabulary, allowing the networks to learn more robust representations
- Transformers – a neural network structure that combines multiple self-attention blocks and allows text encoding/generation
- Language Model Pre-Training – a technique for training neural networks that can be applied to a variety of other tasks

Self-Attention

Which words do you pay attention to?

The **match** burns brightly.

Definitions of match (from https://en.wiktionary.org/wiki/match)

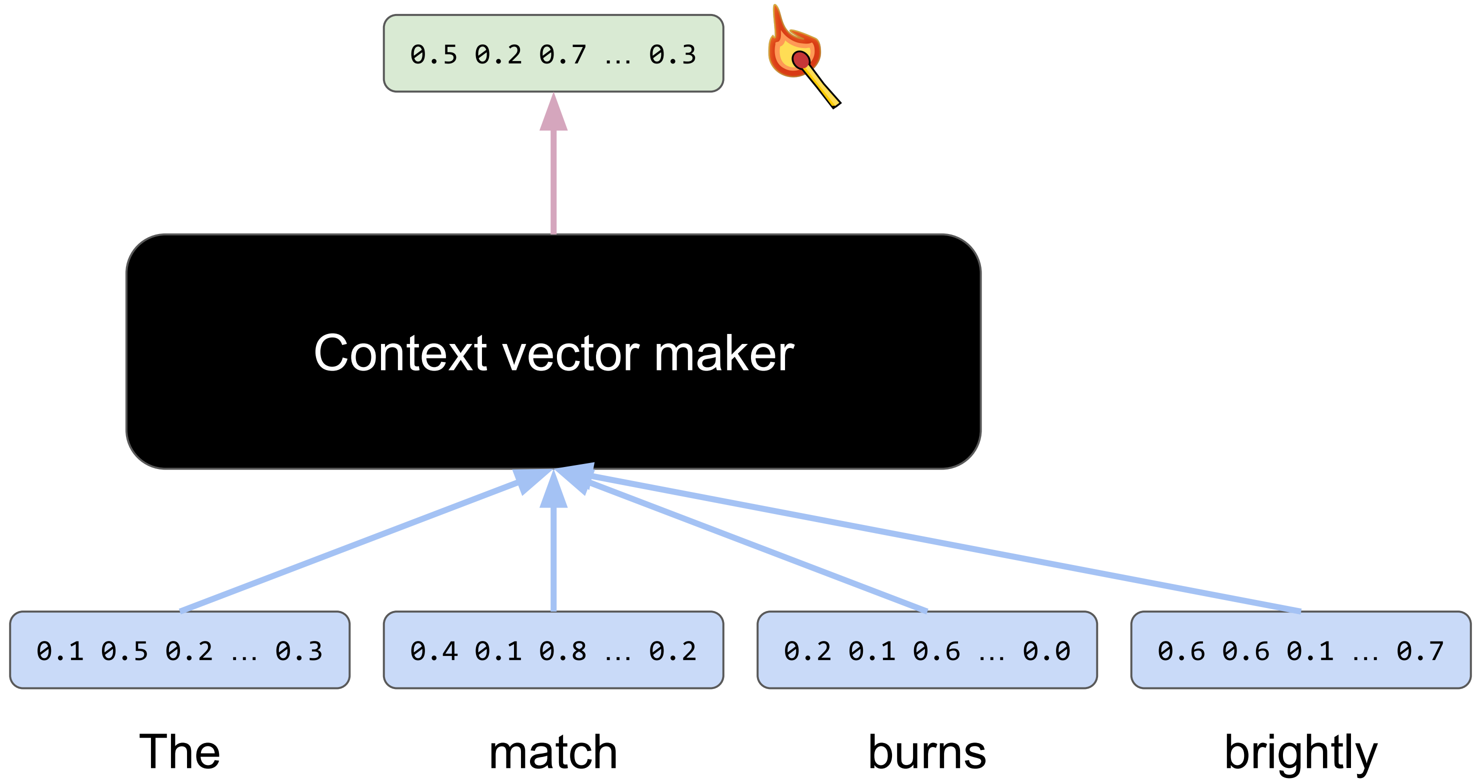
1. (noun) A competitive sporting event such as a boxing meet, a baseball game, or a cricket match.
2. (noun) A device made of wood or paper, at the tip coated with chemicals that ignite with the friction of being dragged (struck) against a rough dry surface.
3. (verb) To agree with; to be equal to; to correspond to.
4. (noun) A pair of items or entities with mutually suitable characteristics.
5. …

Which words were helpful to decide that the meaning of match was (2)?

Some words are more important for picking the meaning of a word

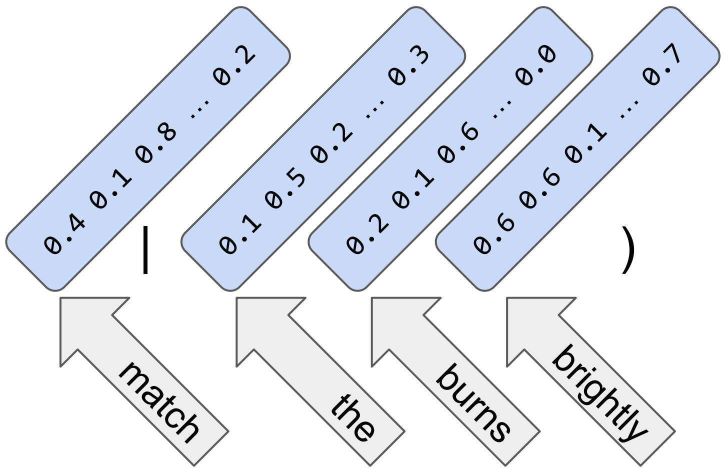
- Words just before can tell you about the part-of-speech
  - Is it likely a noun, a verb, etc?
- Words across the sentence can identify the topic of the sentence
  - ‘gas cooker’ is very helpful to identify the meaning of ‘match’
- Many words are filler and not useful for distinguishing meaning

Making a contextualised vector for “match”



Using the word vectors to make a context vector

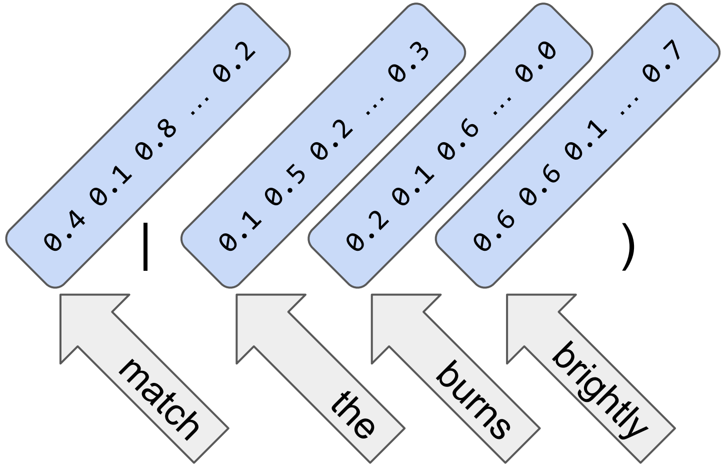
add_context('match' | 'the', 'burns', 'brightly') = F(



- We want a function that adds context to word vectors.
- The inputs will be the word vectors without context - no other outside knowledge

Using the word vectors to make a context vector

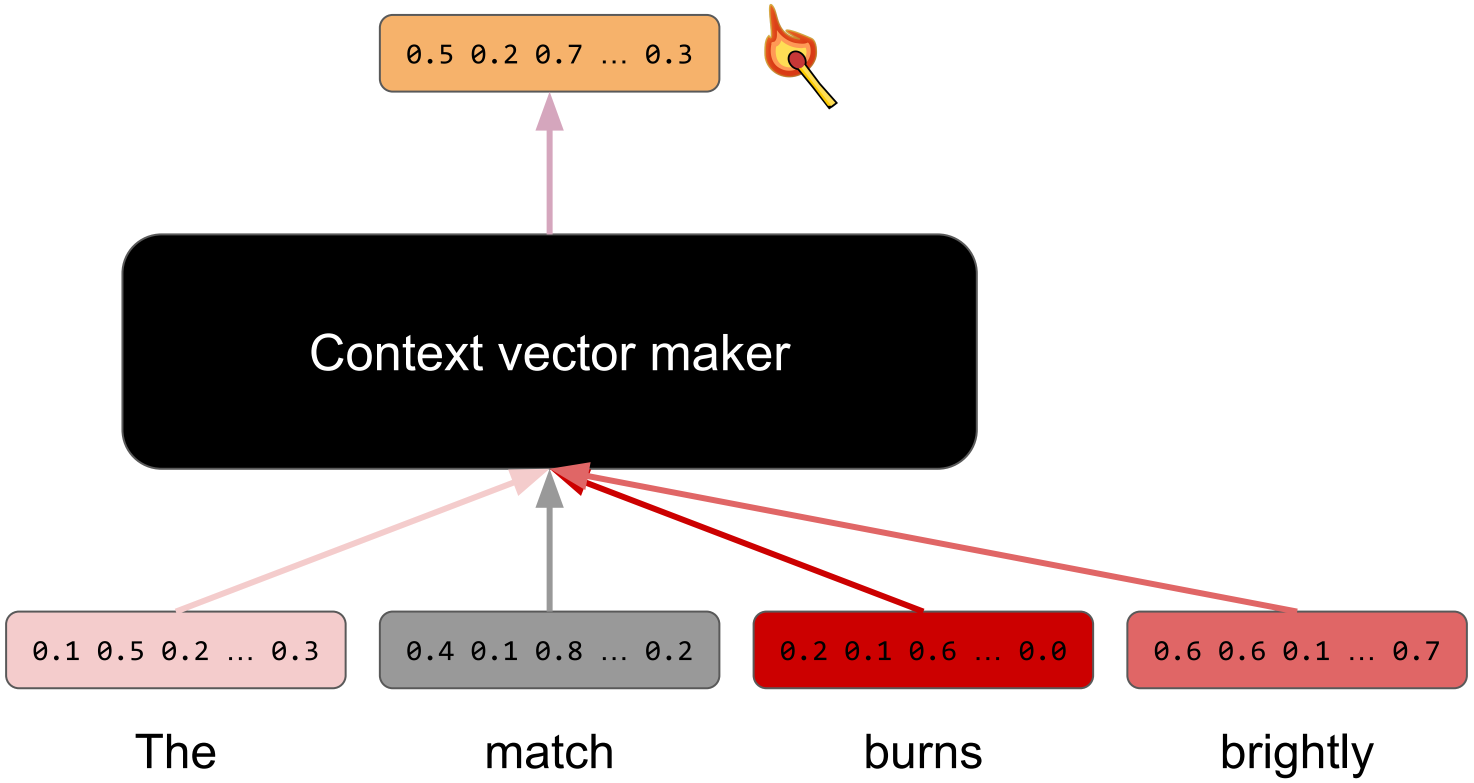
add_context('match' | 'the', 'burns', 'brightly') = F(



Idea:

- Context vectors will be some combination of the 'match' vector with the other word vectors in the sentence.
- But some words are more important than others so we need relevance weights

Weighting the importance of the other words



Need a function that tells you how much attention to give

relevance('the' | 'match') = G(`0.4 0.1 0.8 … 0.2` / `0.1 0.5 0.2 … 0.3`) = 12.1

relevance('burns' | 'match') = G(`0.4 0.1 0.8 … 0.2` / `0.2 0.1 0.6 … 0.0`) = 89.3

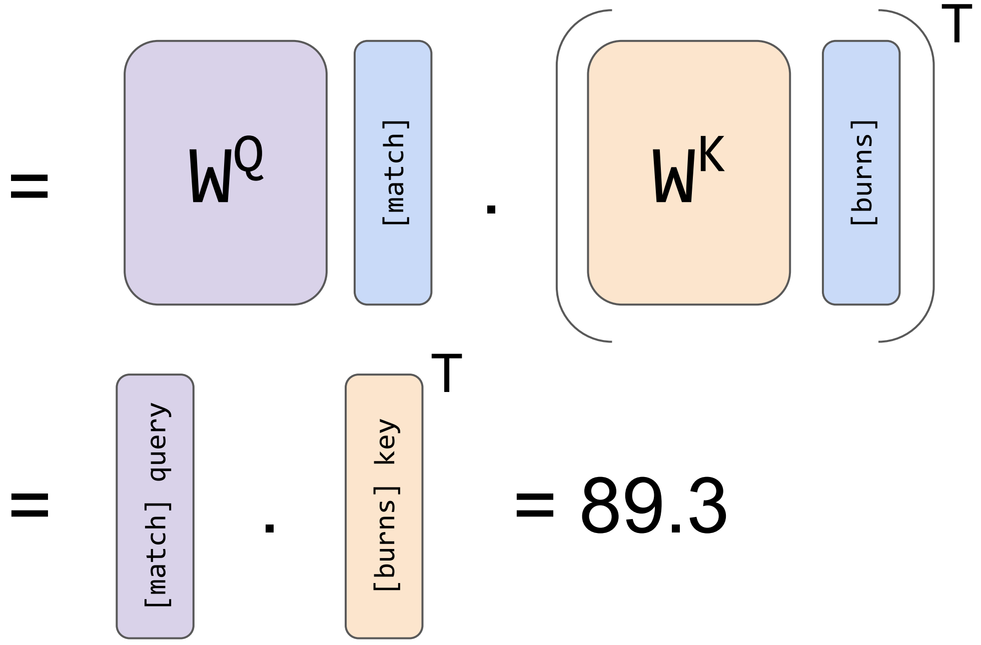
- Calculate the relevance of one word (e.g. 'burns') for the context of another word (e.g. 'match')
- Inputs are the word vectors without context
- How to calculate the relevance from the word vectors?
    - Could you use similarity? No: 'match' and 'burns' are not very similar
    - Will need to do some transformations to the word vectors

Relevance for attention

How important is 'burns' to understand the word 'match'?

- Input is vectors for 'match' and 'burn'
- Use two matrices **W^Q** and **W^K**
  - These matrices are learnt during training
- Matrix multiply the input word vectors to get a **query vector** and a **key vector**.
- Then dot-product these to get the relevance

relevance('burns' | 'match')



$$= W^Q [\text{match}] \cdot \left(W^K [\text{burns}] \right)^T$$

$$= [\text{match}]_{\text{query}} \cdot [\text{burns}]_{\text{key}}^T = 89.3$$

Relevance scores need to add up to 1

- Relevance scores may not be nicely between 0 and 1
- We want them all to add up to 1
  - Then we can use them as weighting
- So we use the commonly used softmax function

relevance('the' | 'match') = 12.1

relevance('match' | 'match') = 91.1

relevance('burns' | 'match') = 89.3

relevance('brightly' | 'match') = 44.7

Softmax relevance scores

- Relevance scores may not be nicely between 0 and 1
- We want them all to add up to 1
    - Then we can use them as weighting
- So we use the commonly used softmax function

relevance(‘the’ | ‘match’) = 12.1

relevance(‘match’ | ‘match’) = 91.1

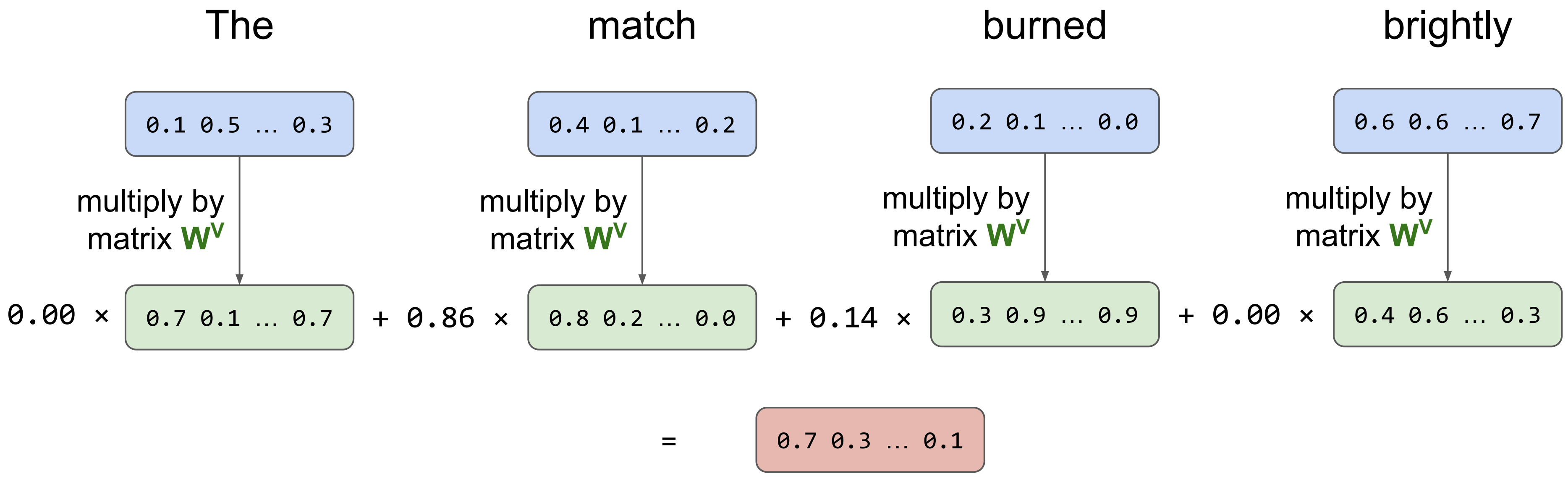
relevance(‘burns’ | ‘match’) = 89.3

relevance(‘brightly’ | ‘match’) = 44.7

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad \text{for } i = 1, \dots, K \text{ and } \mathbf{z} = (z_1, \dots, z_K) \in \mathbb{R}^K.$$

```
softmax([12.1,91.1,89.3,44.7]) = [0.00, 0.86, 0.14, 0.00]
```

Relevance scores used to weigh the transformed word vectors



- Multiply each input vector by matrix W^V to get their **value vectors**
  - Matrix W^V is also learned during the training process
- Add them up using the softmaxed relevance scores as weights

The self-attention equation using matrices

Q = Input vectors multiplied by $\mathbf{W^Q}$ = queries

K = Input vectors multiplied by $\mathbf{W^K}$ = keys

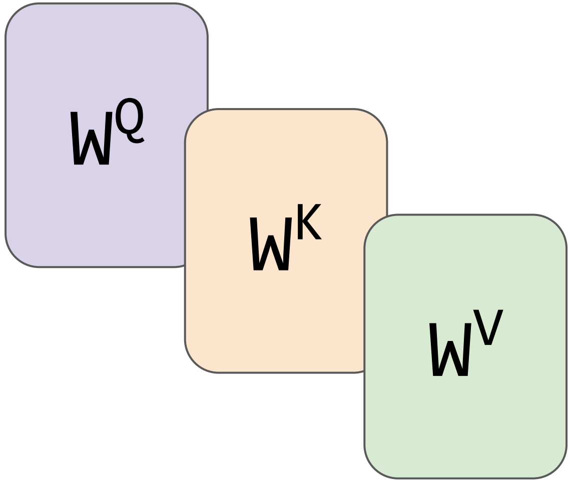
V = Input vectors multiplied by $\mathbf{W^V}$ = values

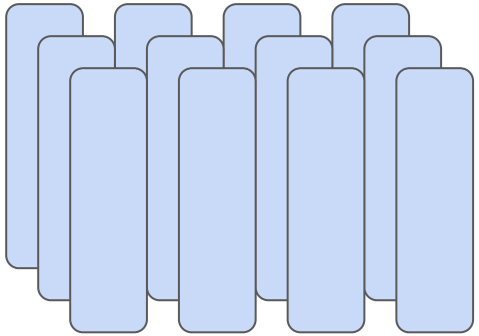
$$\mathrm{Attention}(Q,K,V) = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

- Work with matrices instead of individual vectors
- This equation encapsulates all the previous steps in one-go
  - e.g. QK^T is calculating the relevance scores
- Self attention adds one extra step:
  - Divide by sqrt(d_k) where d_k is size of the embeddings
  - Helps with back-propagation of the network

Where do the embeddings, weight matrices, etc come from?

- Attention relies on multiple matrices $\mathbf{W}^Q$, $\mathbf{W}^K$ and $\mathbf{W}^V$
- Each token also has a word vector to use as input
- Where do these come from?
  - They are all learned during the extensive training process





Why is it called self-attention?

Self-attention is named because it is paying attention to the same text as the text it is working on

- The queries, keys and values are all from the same text

Attention can also be applied between texts (e.g. an English text and a Spanish text).

OXFORD
rtuguese
n complete beginner to intermediate level in 14 easy-to-understand units
H YOURSELF
inner's
rman
French
Mini
Dictionary
The world's most trusted dictionaries
Oxford
Spanish
Mini
Dictionary
The world's most trusted dictionaries
JAPANESE
FOR
teach yourself
italian
quick fix spanish grammar
Por
talk

Self-attention summary

- Self-attention allows a language model to weight which other tokens are important when interpreting a token
- It uses an equation that transforms the input vectors into queries, keys and values
- The queries and keys are used to calculate the relevance scores of other tokens to the token of interest
- The softmaxed relevance scores are used as weights to combine the value vectors
- All of the weight matrices and word embedding vectors are learnt during training

Subword Tokenization

The problem with new words

"I think I'm going to take a ***staycation*** *next month"*

- Lots of reasons for new words occurring:
  - Actually new words
  - Misspellings
  - Words that weren't in the training set
- Language models have a hard time with new words
  - They know nothing about them
  - Have to treat them as OOV - out of vocabulary
  - No learned embeddings and probabilities of zero of them occurring
    - We learnt about smoothing before

Subwords can help us deal with new words

Core idea: Split uncommon words into 2 or more parts (potentially syllables)

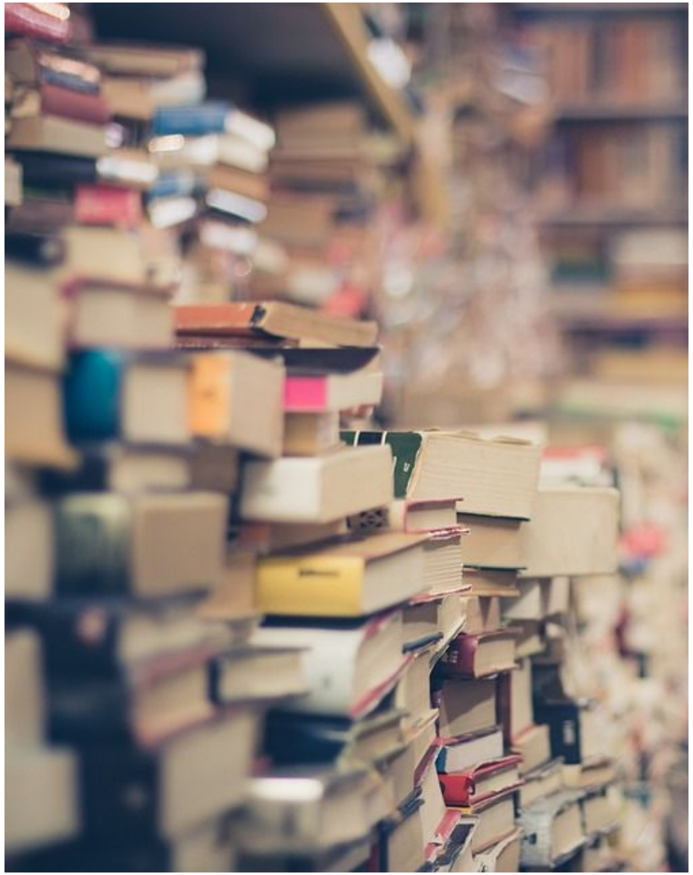
- Will depend on the language and type of text (e.g. tweets versus science)

Why does this help?

- Much more likely to have seen subwords
- Subwords often give an idea of overall meaning for new words
- Can reduce the overall size of the vocabulary - reduces memory needs

staycation
deepfake
cryptocurrency
microfinance
onboarding
truthiness
annoyingly

Learning to subword tokenize



Given a large corpus of text, we can *learn* how to tokenize text into common subwords

We could use:

- Newspapers
- Tweets
- Scientific articles
- etc

Depends on what we want to use the tokenized text for later

Byte pair encoding (BPE)

Inputs:
1. Large corpus of text to learn tokenization
2. The desired size of vocabulary

Algorithm:
1. Pretokenize the corpus documents into words using a tokenizer (this will remove whitespace)
2. Create a vocabulary of symbols: all unique characters in the corpus (i.e. all letters, numbers, etc)
3. Repeat until the desired vocab size is reached
   a. Find the most common neighbouring symbols in the corpus
   b. Replace all instances of the pair with a new character and add new character to the vocabulary

Byte pair encoding (BPE) example (training)

Let’s learn a tokenization for the small corpus of:

“peter piper picked a peck of pickled peppers”

Byte pair encoding (BPE) example (training)

Words in corpus:

```
[[p,e,t,e,r], [p,i,p,e,r], [p,i,c,k,e,d], [a],
[p,e,c,k], [o,f], [p,i,c,k,l,e,d], [p,e,p,p,e,r,s]]
```

Current vocabulary

```
[a,c,d,e,f,i,k,l,o,p,r,s,t]
```

Most frequent pair

```
('p', 'e') with count 5
```

Byte pair encoding (BPE) example (training)

```
After merging ('p', 'e')
```

Words in corpus:

```
[[pe,t,e,r], [p,i,pe,r], [p,i,c,k,e,d], [a], [pe,c,k],
        [o,f], [p,i,c,k,l,e,d], [pe,p,pe,r,s]]
```

Current vocabulary

```
[a,c,d,e,f,i,k,l,o,p,r,s,t,pe]
```

Most frequent pair

```
('p', 'i') with count 3
```

Byte pair encoding (BPE) example (training)

```
After merging ('p', 'i')
```

Words in corpus:

```
[[pe,t,e,r], [pi,pe,r], [pi,c,k,e,d], [a], [pe,c,k],
        [o,f], [pi,c,k,l,e,d], [pe,p,pe,r,s]]
```

Current vocabulary

```
[a,c,d,e,f,i,k,l,o,p,r,s,t,pe,pi]
```

Most frequent pair

```
('c', 'k') with count 3
```

Byte pair encoding (BPE) example (training)

```
After merging ('c', 'k')
```

Words in corpus:

```
[[pe,t,e,r], [pi,pe,r], [pi,ck,e,d], [a], [pe,ck],
        [o,f], [pi,ck,l,e,d], [pe,p,pe,r,s]]
```

Current vocabulary

```
[a,c,d,e,f,i,k,l,o,p,r,s,t,pe,pi,ck]
```

Most frequent pair

```
('pe', 'r') with count 2
```

Byte pair encoding (BPE) example (training)

```
After merging ('pe', 'r')
```

Words in corpus:

```
[[pe,t,e,r], [pi,per], [pi,ck,e,d], [a], [pe,ck],
[o,f], [pi,ck,l,e,d], [pe,p,per,s]]
```

Current vocabulary

```
[a,c,d,e,f,i,k,l,o,p,r,s,t,pe,pi,ck,per]
```

Most frequent pair

```
('pi', 'ck') with count 2
```

Byte pair encoding (BPE) example (training)

```
After merging ('pi', 'ck')
```

Words in corpus:

```
[[pe,t,e,r], [pi,per], [pick,e,d], [a], [pe,ck], [o,f],
[pick,l,e,d], [pe,p,per,s]]
```

Current vocabulary

```
[a,c,d,e,f,i,k,l,o,p,r,s,t,pe,pi,ck,per,pick]
```

Continue until the desired size of vocabulary is reached

Byte pair encoding (BPE) example (tokenization)

To sub-tokenize a new text, apply the same process (without adding new rules):

1) Pretokenize
2) Split into characters
3) Apply each rule from training (in order)

Byte pair encoding (BPE) example (tokenization)

Learned Vocabulary

```
[a,c,d,e,f,i,k,l,o,p,r,s,t,pe,pi,ck,per,pick]
```

Words to tokenize:

```
pickle picker
```

Pretokenized & Split into characters

```
[[p,i,c,k,l,e], [p,i,c,k,e,r]]
```

Byte pair encoding (BPE) example (tokenization)

Learned Vocabulary

[a,c,d,e,f,i,k,l,o,p,r,s,t,pe,pi,ck,per,pick]

Words to tokenize:

pickle picker

Subword tokens

[[p,i,c,k,l,e], [p,i,c,k,e,r]]

Use ‘pe’ rule and merge appropriate subword tokens (No effect)

Byte pair encoding (BPE) example (tokenization)

Learned Vocabulary

```
[a,c,d,e,f,i,k,l,o,p,r,s,t,pe,pi,ck,per,pick]
```

Words to tokenize:

```
pickle picker
```

Subword tokens

```
[[pi,c,k,l,e], [pi,c,k,e,r]]
```

Use ‘pi’ rule and merge appropriate subword tokens (Two matches)

Byte pair encoding (BPE) example (tokenization)

Learned Vocabulary

```
[a,c,d,e,f,i,k,l,o,p,r,s,t,pe,pi,ck,per,pick]
```

Words to tokenize:

```
pickle picker
```

Subword tokens

```
[[pi,ck,l,e], [pi,ck,e,r]]
```

Use ‘ck’ rule and merge appropriate subword tokens (Two matches)

Byte pair encoding (BPE) example (tokenization)

Learned Vocabulary

[a,c,d,e,f,i,k,l,o,p,r,s,t,pe,pi,ck,per,pick]

Words to tokenize:

pickle picker

Subword tokens

[[pi,ck,l,e], [pi,ck,e,r]]

Use ‘per’ rule and merge appropriate subword tokens (No effect)

Byte pair encoding (BPE) example (tokenization)

Learned Vocabulary

[a,c,d,e,f,i,k,l,o,p,r,s,t,pe,pi,ck,per,pick]

Words to tokenize:

pickle picker

Subword tokens

[[pick,l,e], [pick,e,r]]

Use 'pick' rule and merge appropriate subword tokens (Two matches)

Subword tokenization variants

- Treat the starts of words differently from characters inside the word
  - A `'##'` prefix indicates a subword is inside a word
  - `'pickled' -> 'pick', '##led'`
  - `'repickled' -> 'rep', '##ick', '##led'`
- Use bytes instead of characters as the initial vocabulary
  - With Unicode, there are a lot of possible characters (e.g. emojis!)
  - With bytes, there are only 256 possible values
  - Known as byte-level BPE
- Don’t pick the most frequent pair to merge
  - Pick the pair that maximises the likelihood of the training corpus
  - Used by WordPiece method
- Don’t do the pre-tokenization (that removes spaces)
  - Include spaces in the vocabulary
  - Useful for languages without tokenizers
  - Used by SentencePiece method

The `[CLS]`, `[SEP]`, `[PAD]`, and `[MASK]` tokens

Subword tokenizers for Transformers commonly have these four special tokens:

`[CLS]`

- Added at the beginning of a sequence
- Used to create a context vector that captures the meaning of the whole sequence
- Helpful for sentence-level CLaSsification tasks.

`[SEP]`

- Added at the end of a sequence
- May also be used in between sentences for some tasks

`[PAD]`

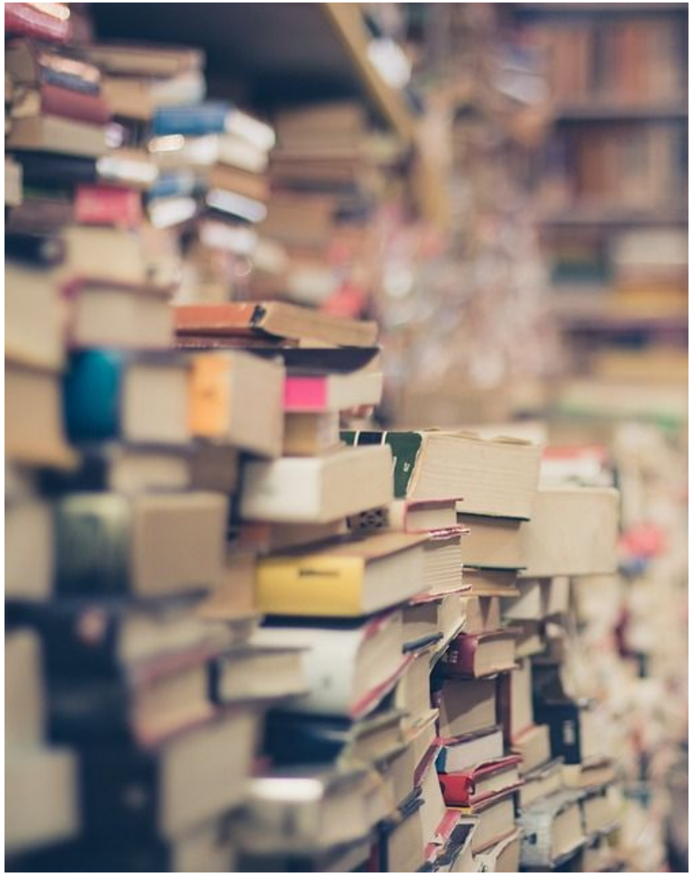
- Added at the end of a sequence to ensure that multiple sequences are the same length, which is helpful for batch processing.
- Usually excluded from self-attention computation using an *attention mask*.

`[MASK]`

- Used to hide input tokens that need to be predicted (for training a language model)

Your vocabulary (& tokenizer) depends on the training corpus

- A subword tokenizer learned from English newspapers will not work well with German text
- Same is true using a general English subword tokenizer on scientific text



Subword tokenization summary

- Language models have a hard time with new words - generally treating them as OOV (out-of-vocabulary)
- Splitting uncommon words into subwords can mean that almost all words/subwords have been seen before
- Subwords are often useful to identify the meaning of new words too
- You can learn tokenization from a large corpus of text
- Byte pair encoding (BPE) algorithm merges common pairs of symbols (which are initially individual characters) until the desired vocabulary size is achieved
- Various other similar approaches to subword tokenization

Break!

CS Level 3 Semester 2 Feedback Form

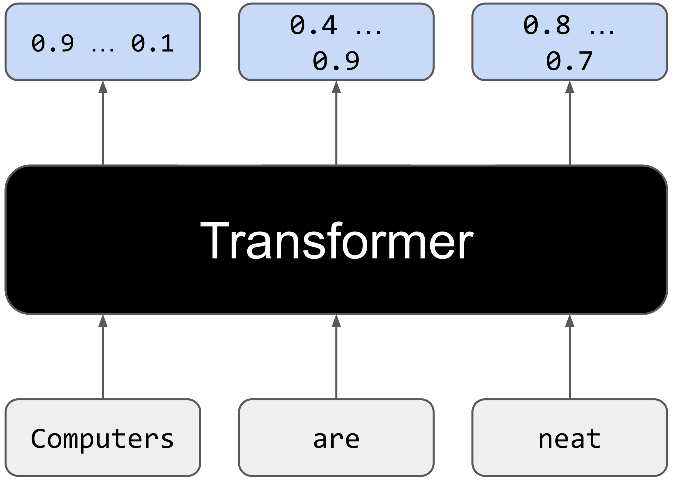
(from class reps)

Transformers

The dominant deep learning model for text (for the time being)

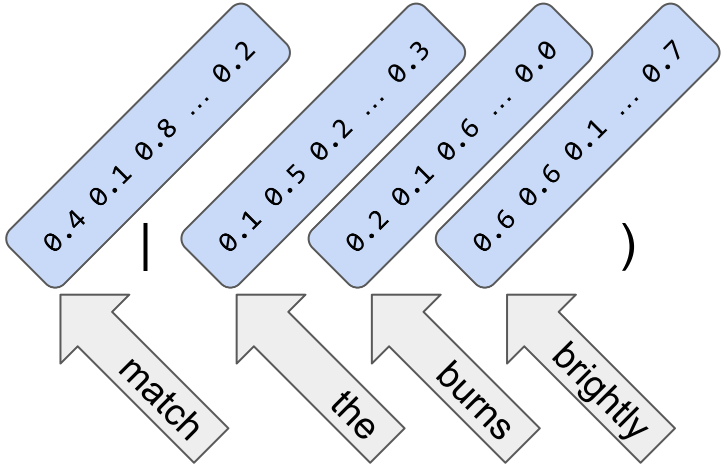
- A neural network architecture that can encode context, grammar and other aspects in context vectors
- Has been shown to be good at all kinds of language tasks (e.g. classification, translation, question-answering, etc)
- Powers things like ChatGPT
- Very recent: Core ideas published in 2017 and huge advances since then

Transformers combine self-attention, subword tokenization and some other neat ideas



Attention (by itself) lacks positional information

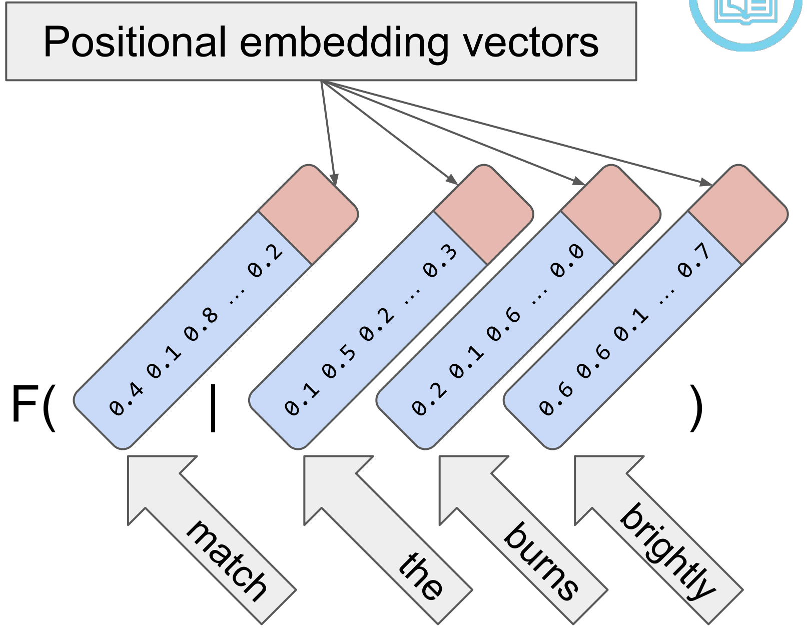
add_context('match' | 'the', 'burns', 'brightly') = F(



- The locations of words relative to each other is important for meaning
  - Is one word just before another, or at the far end of a sentence?
- Our approach to adding context doesn't include **positional information**

Adding positional embedding vectors

add_context('match' | 'the', 'burns', 'brightly') = F(

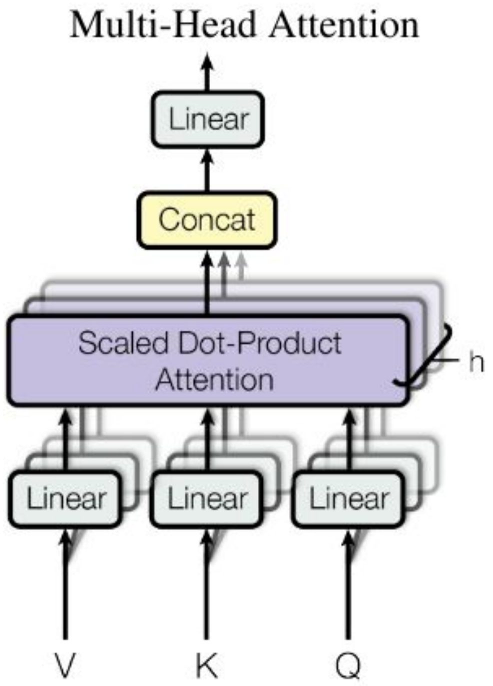


)

- Positions can be encoded as dense vectors and combined with the existing word vectors
- Different approaches to creating those dense vectors
  - Learning them from scratch
  - Using different frequency sinusoidal functions to create vectors that differ at each position in the input

Multi-Head attention

- Attention allows a neural network to focus specifically on certain words for an aspect of language (or so we think)
  - Grammar rules
  - Words that indicate a topic
  - etc
- Single attention system may not be able to deal with all aspects needed
  - E.g., identifying coreference is different than word adjacency
- So why not do attention multiple times!
  - Many models do it 12 times
- This is called **multi-head attention**
  - Each attention block has its own $\mathbf{W}^{Q}$, $\mathbf{W}^{K}$ and $\mathbf{W}^{V}$ that has different values



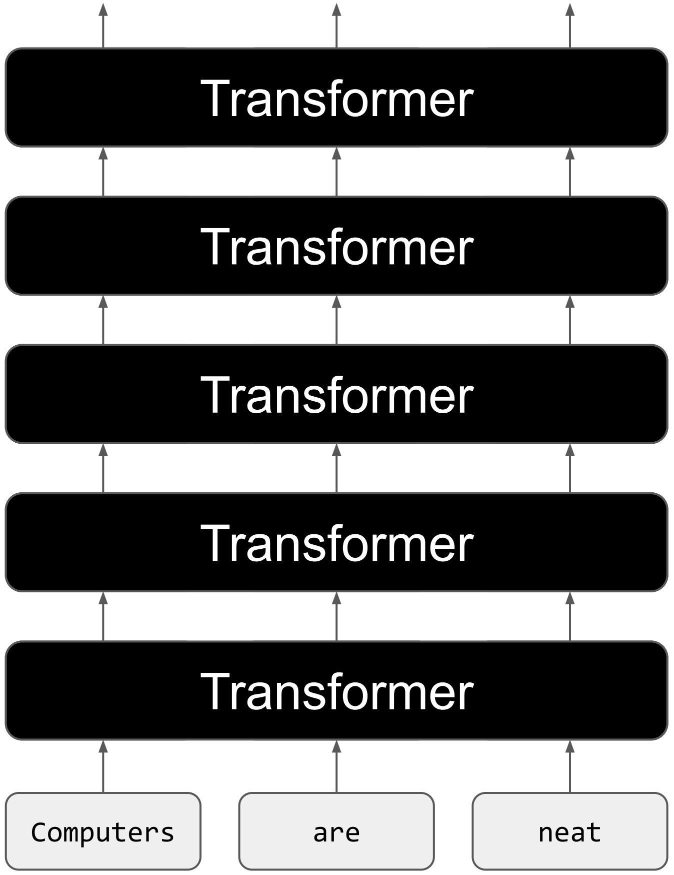
Stacking layers

Deep learning is **deep** because there are normally many layers. Outputted context vectors from a Transformer layer are fed into the next layer as input and so on.

Each layer is building up more meaning

- Lower layers likely dealing with basic syntax
- Higher layers dealing with more complex reasoning

Smaller models have ~12 layers. Very big models may have 96 layers!



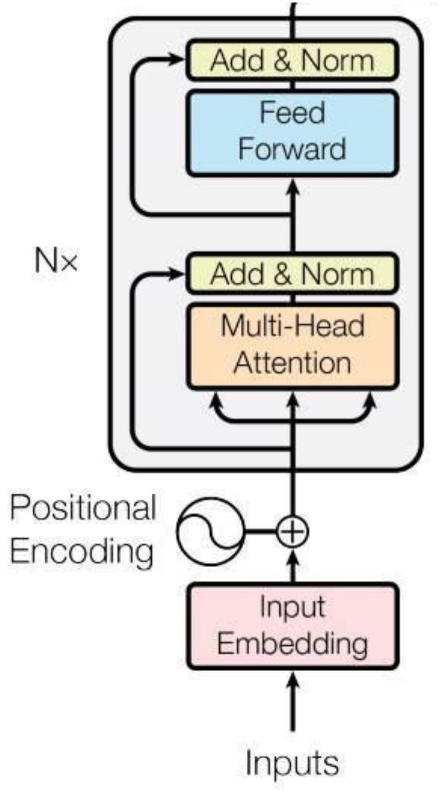
The specifics on a Transformer block

Things we've already discussed

- Positional encodings
- Multi-Head Attention
- Stacking layers (N×)

Extra things

- Add - Trick for easier training
  - An optional path that provides a bypass
- Norm - Another trick for easier training
  - Normalize the vectors so the scaling doesn't go crazy
- Feed Forward network
  - Standard fully connected neural network
  - Allows for encoding more complex functions than attentions' linear combination of inputs



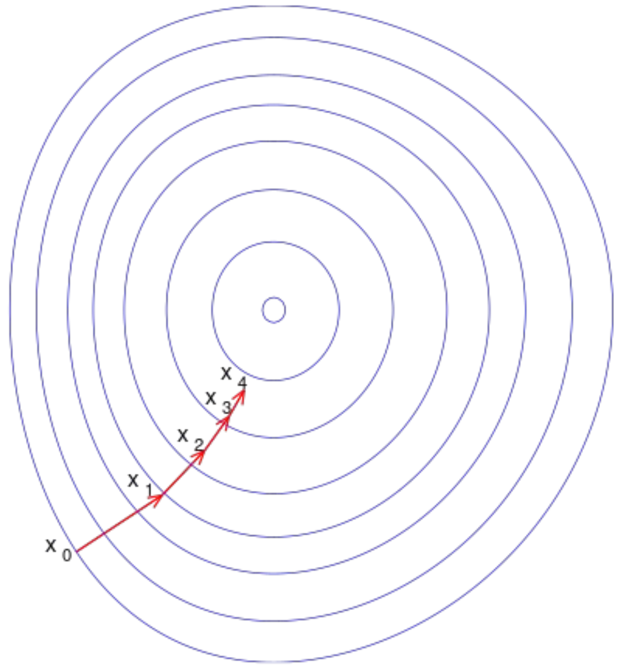
Training a transformer

1. Provide input and expected output
2. Run input through Transformer and compare to expected output
3. Adjust weights in neural network to get output closer to expected output

This involves the **backpropagation algorithm** (see deep learning course).

What are the inputs & outputs?

- There are different ideas for language modelling tasks



https://en.wikipedia.org/wiki/Gradient_descent

Different language modelling tasks

Causal Language Modelling: *Predict the next word:*

It showed 9 o'clock on my ________

Masked Language Modelling: *Predict a masked word:*

It showed 9 ________ on my watch

Next sentence prediction: *Does one sentence follow the other?*

It showed 9 o'clock. Ganymede is a moon of Jupiter.

Replaced token detection: *Spot the corrupted word*

It showed 9 o'clock on my stapler -> stapler is wrong

Different language modelling tasks

GPT tasks

Causal Language Modelling: *Predict the next word*

It showed 9 o'clock on my ________

BERT tasks

Masked Language Modelling: *Predict a masked word*

It showed 9 ________ on my watch

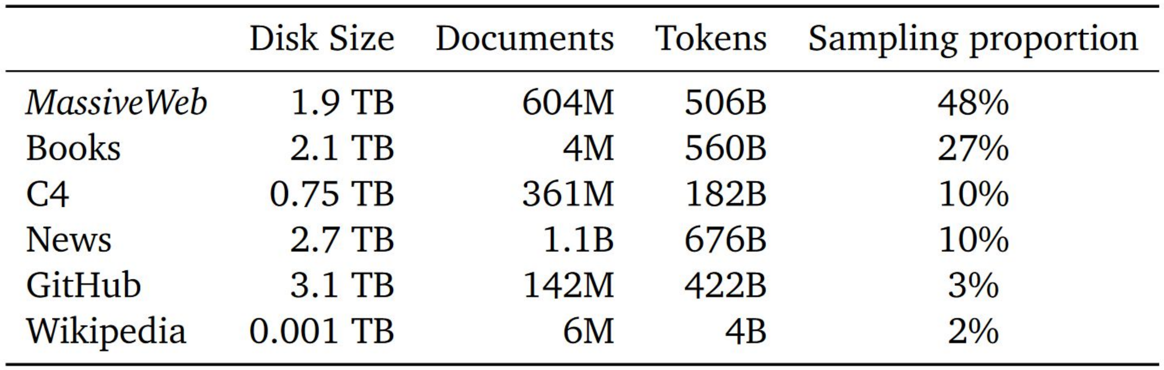
Next sentence prediction: *Does one sentence follow the other?*

It showed 9 o'clock. Ganymede is a moon of Jupiter.

Replaced token detection: *Spot the corrupted word*

It showed 9 o'clock on my stapler -> stapler is wrong

Corpora used for training



	Disk Size	Documents	Tokens	Sampling proportion
MassiveWeb	1.9 TB	604M	506B	48%
Books	2.1 TB	4M	560B	27%
C4	0.75 TB	361M	182B	10%
News	2.7 TB	1.1B	676B	10%
GitHub	3.1 TB	142M	422B	3%
Wikipedia	0.001 TB	6M	4B	2%

Example corpora from recent Gopher language paper

- Human-created text is used to create examples for language modelling tasks
- Document collections (corpora) are getting **very big!**
- May contain multiple languages, and recently programming languages!
- Issues with getting “good text” that does not contain problematic language

Transformers were first proposed for machine translation

Machine translation is the task of translating one language to another.

Using Transformers led to concepts of an **encoder** and a **decoder**. Let's explore!

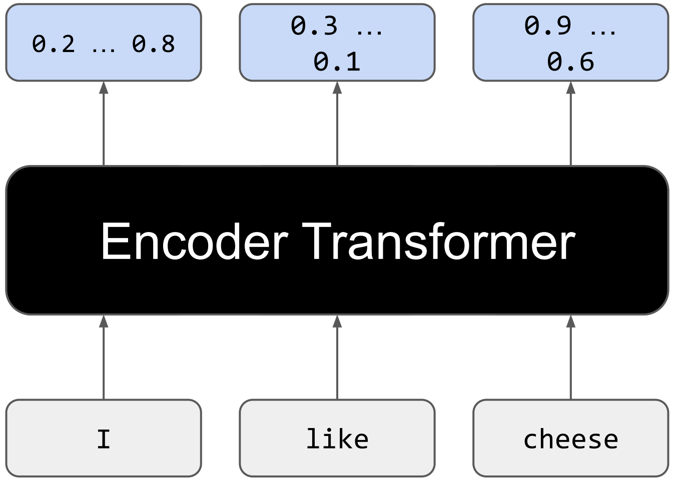
Fun fact: Machine translation research has been driven forward by the innumerable documents from the European parliament that have been translated into multiple languages

English	French
I like cheese	J'aime le fromage
Colourless green ideas sleep furiously	Les idées vertes incolores dorment furieusement
The quick brown fox jumps over the lazy dog	Le renard brun rapide saute par-dessus le chien paresseux
The clocks were striking thirteen	Les horloges sonnaient treize

Encoder transformers

The Transformers we have talked about so far are more specifically **Encoder Transformers**. They take in input tokens, look at the entire context, and output context vectors.

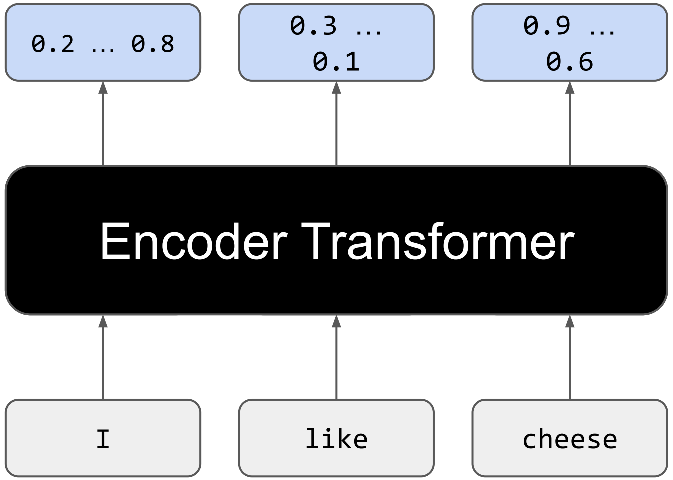
We're now going to talk about **Encoders** and a slight variation **Decoders** in the context of translation (where the idea of encoders & decoders took off)



Encoders & decoders

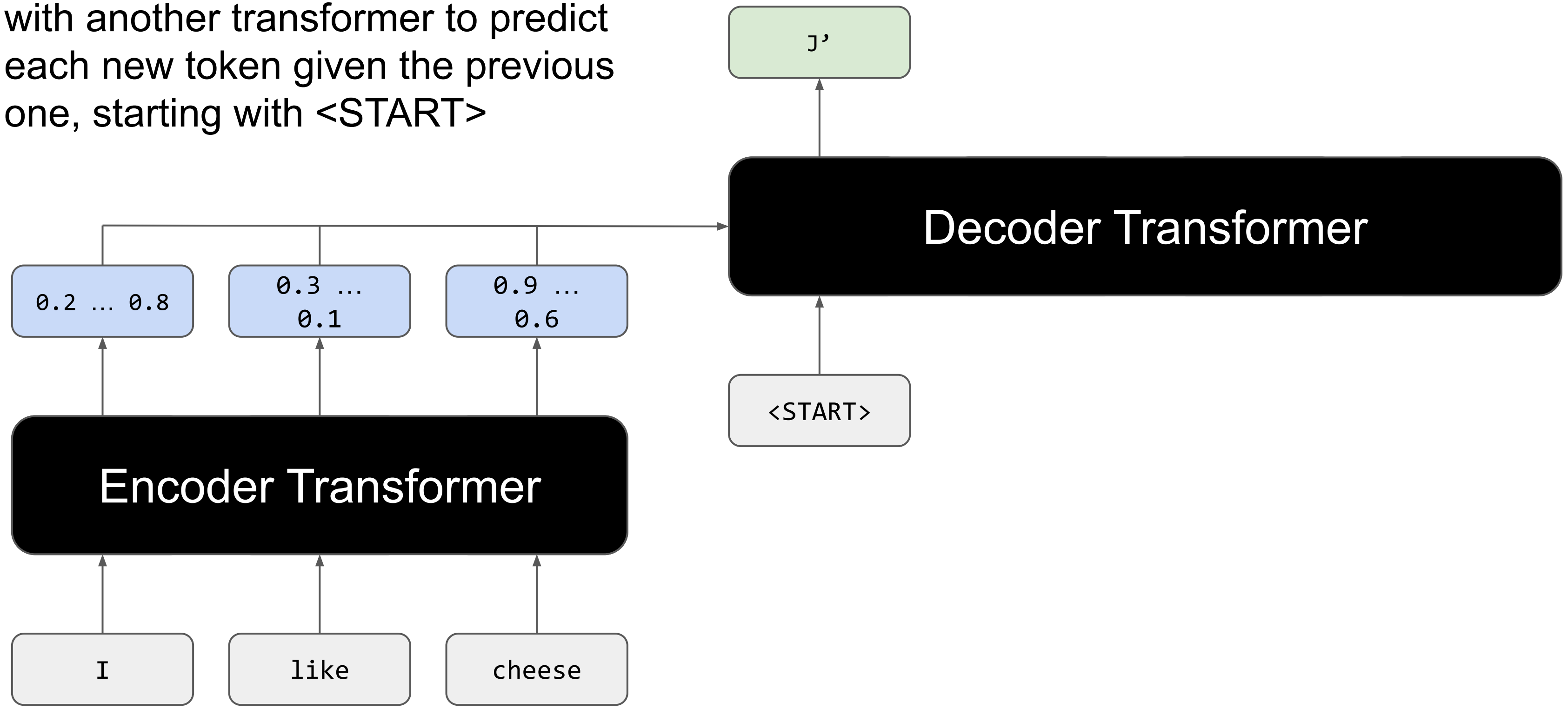
Transformers can be used for sequence-to-sequence problems (like translation)

First you encode the input text into context vectors with a transformer



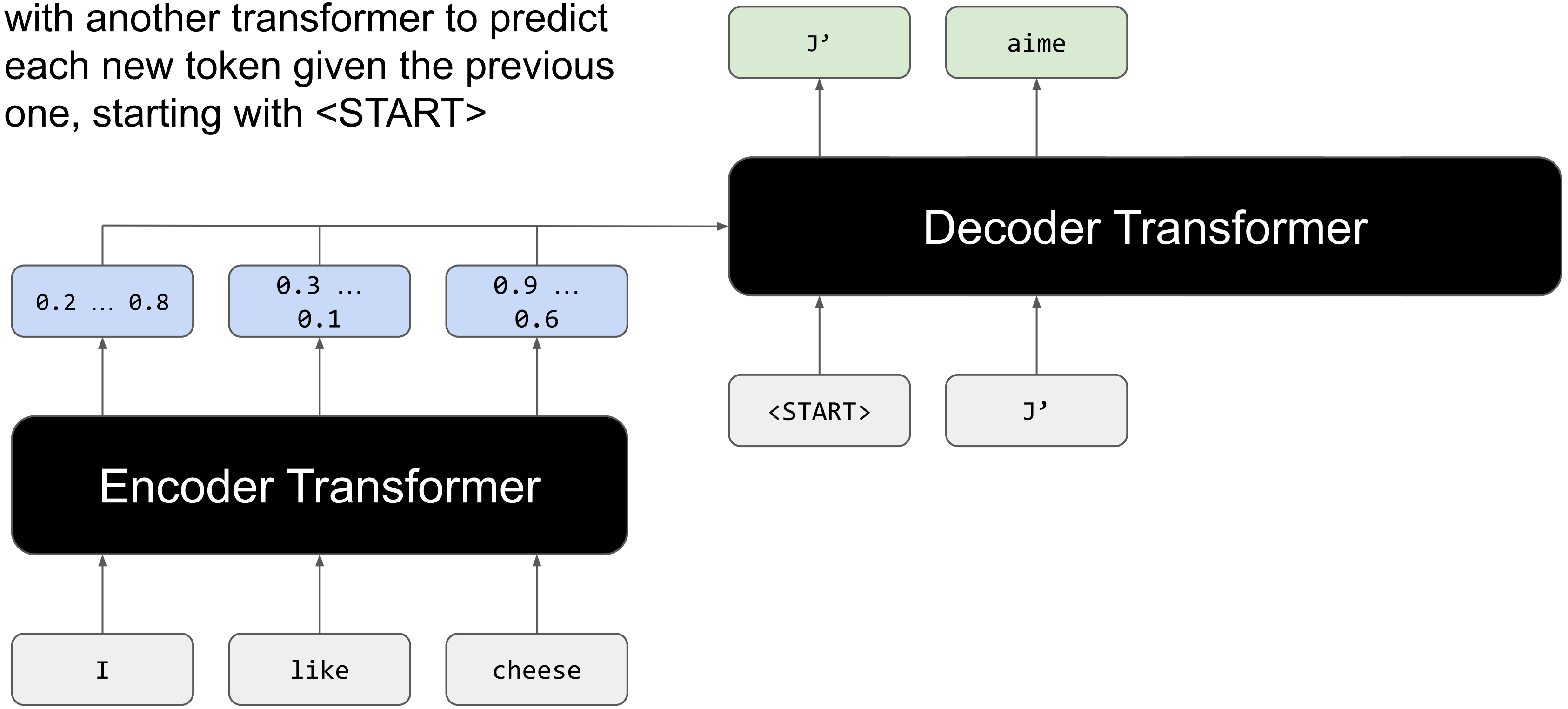
Encoders & decoders

Then you use those context vectors with another transformer to predict each new token given the previous one, starting with <START>



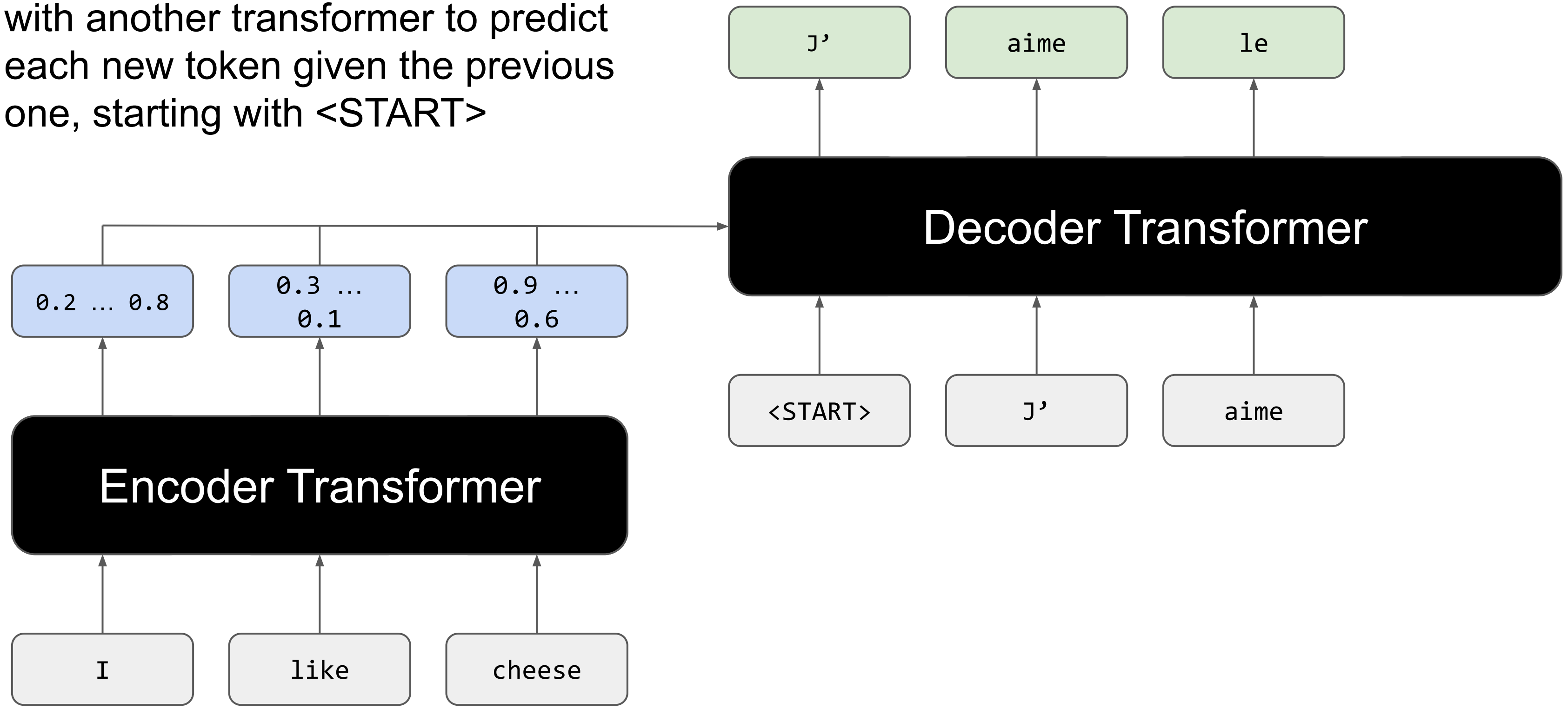
Encoders & decoders

Then you use those context vectors with another transformer to predict each new token given the previous one, starting with <START>



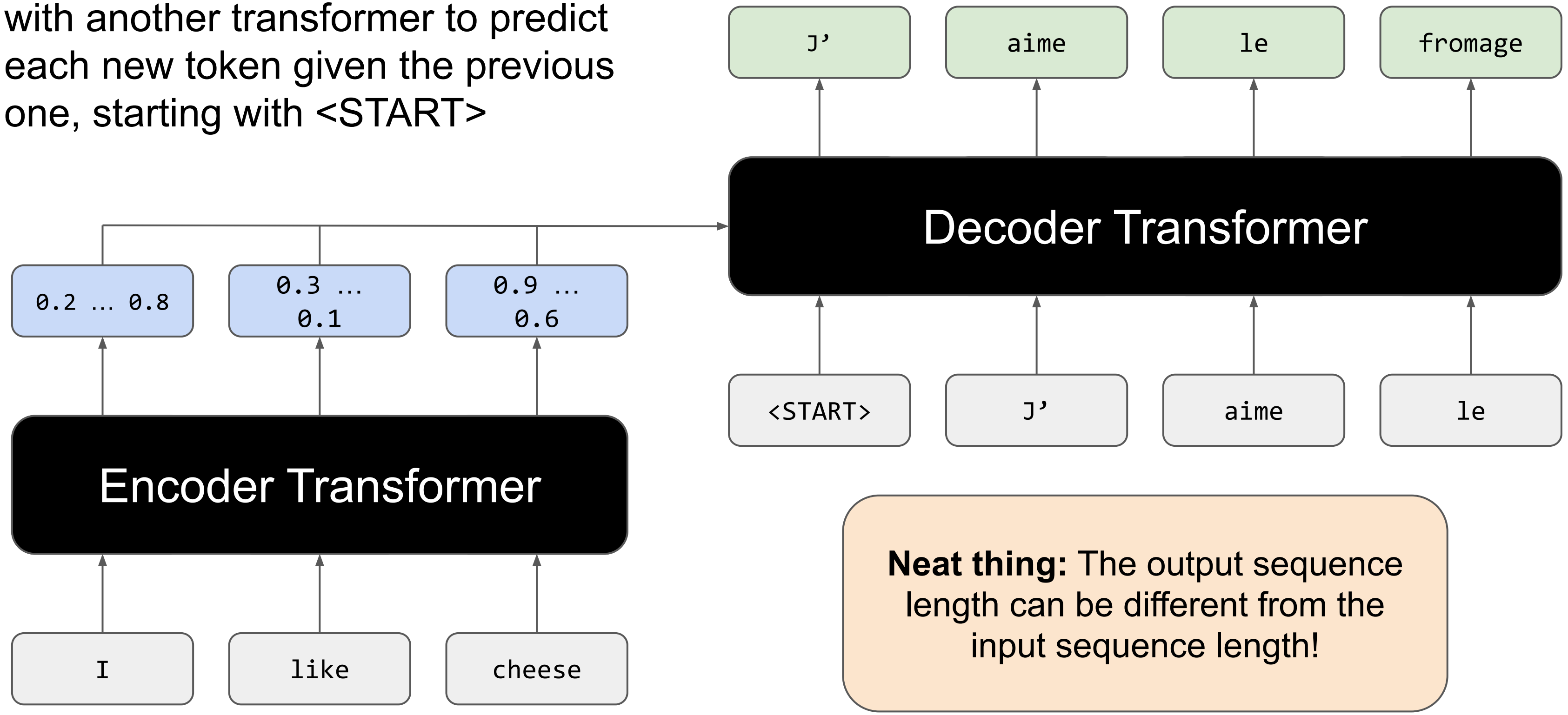
Encoders & decoders

Then you use those context vectors with another transformer to predict each new token given the previous one, starting with <START>



Encoders & decoders

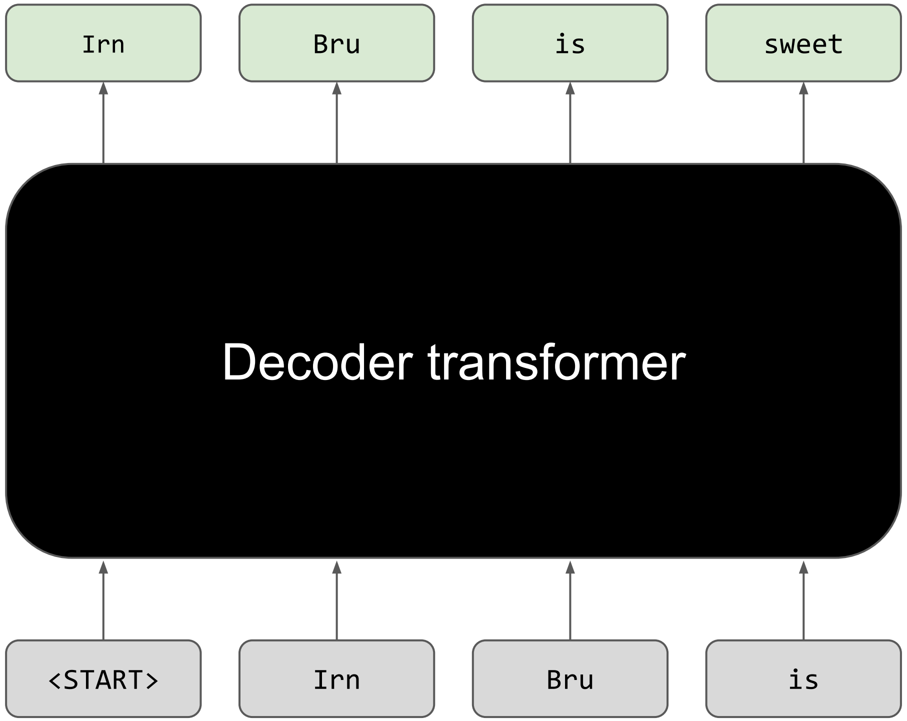
Then you use those context vectors with another transformer to predict each new token given the previous one, starting with <START>



Neat thing: The output sequence length can be different from the input sequence length!

A decoder-only architecture

Decoders can be used on their own (e.g. for predicting the next word in a sequence) or as part of an encoder-decoder combination



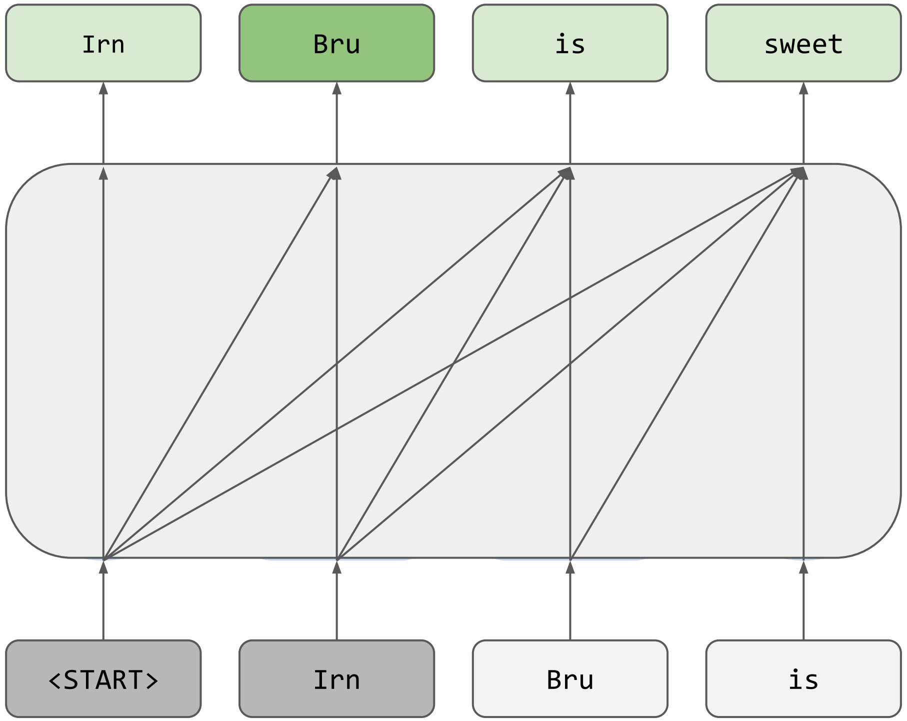
Training the decoder

Practically, we don't want to step one word at a time when training our decoder.

We can do a whole sequence at a time with one trick:

- The decoder is not allowed to look at future words in the input sequence when predicting a word.

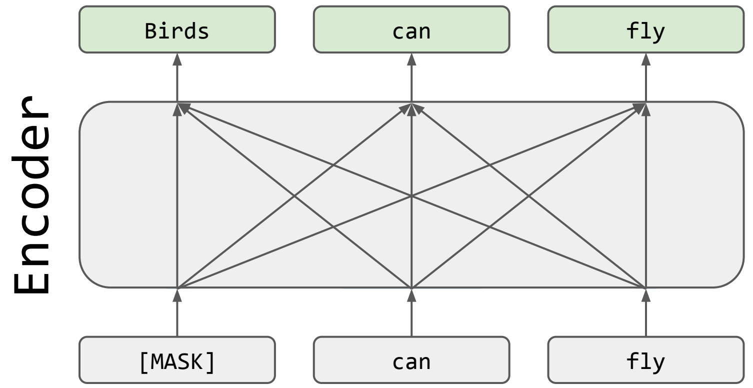
So when predicting 'Bru', it does not get to see the inputs 'Bru' or 'is'.



Encoder only or decoder only?

Encoder architecture only (e.g. BERT):

- Allowed to see all inputs (no masking)
- Good at taking the whole context (previous and future words) into account
- Trained with masked language task
- Called masked language models



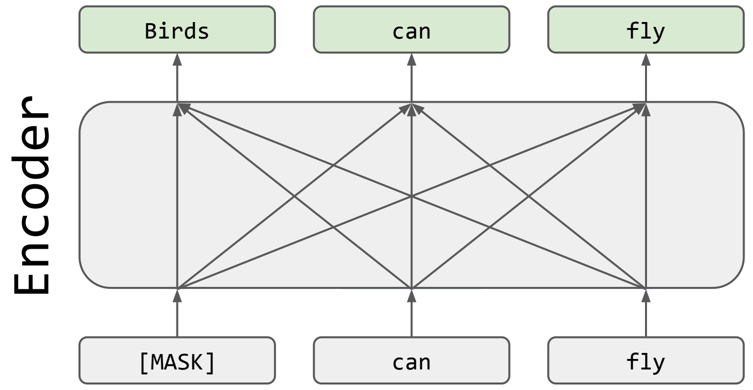
Encoder only or decoder only?

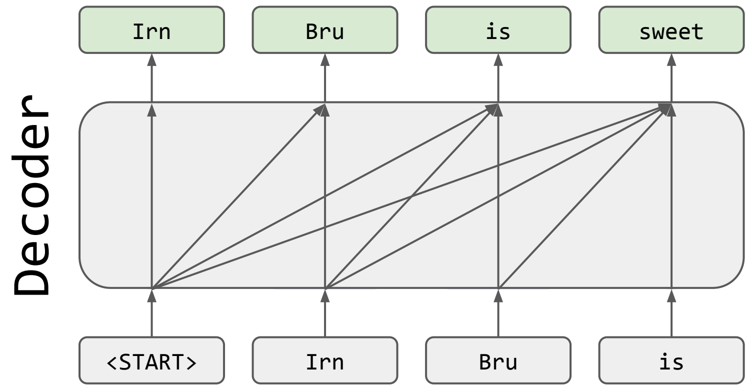
Encoder architecture only (e.g. BERT):

- Allowed to see all inputs (no masking)
- Good at taking the whole context (previous and future words) into account
- Trained with masked language task
- Called masked language models

Decoder architecture only (e.g. GPT3):

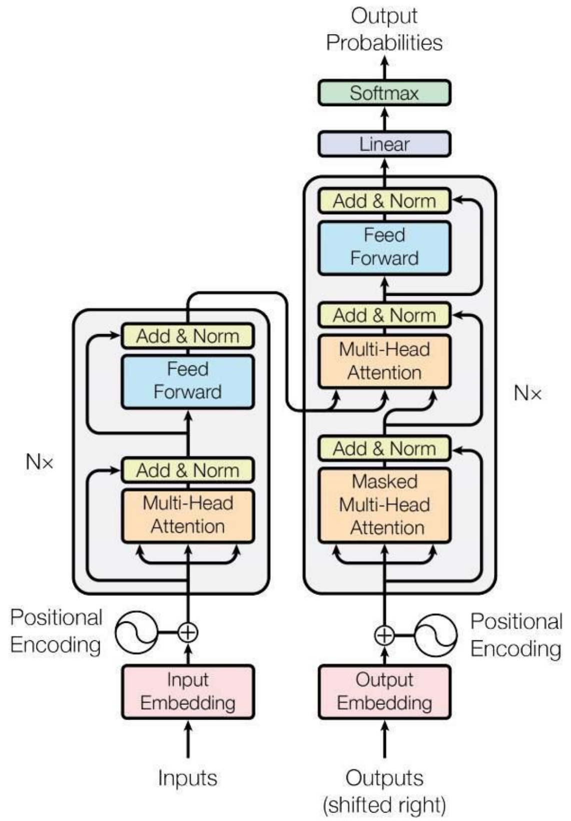
- Cannot see "future" inputs so uses clever masking to stop that
- Means it gets very good at predicting the next word
- Trained with causal language task ("guess the next token")
- Called causal/autoregressive language models





The original encoder-decoder architecture

- The idea of encoder-decoder transformers was proposed in the "Attention is all you need" paper
- The figure on the right shows the encoder (on the left) and the decoder (on the right) with its constituent parts
- This figure is used a lot when talking about Transformers.
  - Good for you to see
  - But there is a lot going on in it



Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems 30 (2017).

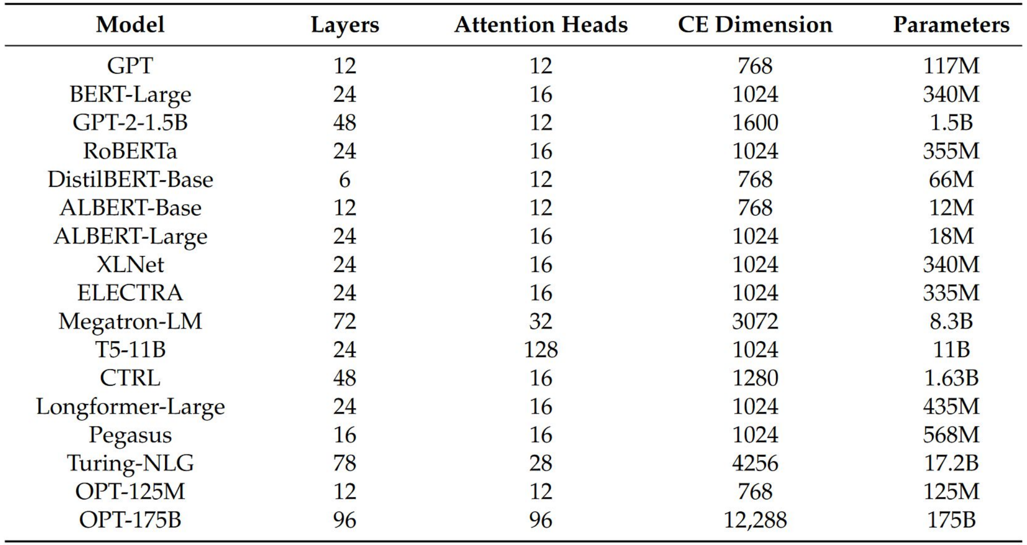
Bigger is better?

Transformers really launched with BERT and GPT which are now considered small

Bigger models have shown improved performance across lots of language tasks (e.g. classification, translation, generation, etc)

Big models now require ~600GB of GPU RAM!

1M parameters = 3.8 megabytes
1B parameters = 3.7 gigabytes

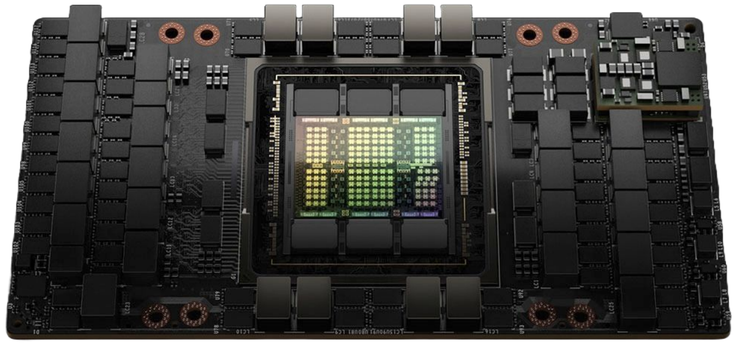


Model	Layers	Attention Heads	CE Dimension	Parameters
GPT	12	12	768	117M
BERT-Large	24	16	1024	340M
GPT-2-1.5B	48	12	1600	1.5B
RoBERTa	24	16	1024	355M
DistilBERT-Base	6	12	768	66M
ALBERT-Base	12	12	768	12M
ALBERT-Large	24	16	1024	18M
XLNet	24	16	1024	340M
ELECTRA	24	16	1024	335M
Megatron-LM	72	32	3072	8.3B
T5-11B	24	128	1024	11B
CTRL	48	16	1280	1.63B
Longformer-Large	24	16	1024	435M
Pegasus	16	16	1024	568M
Turing-NLG	78	28	4256	17.2B
OPT-125M	12	12	768	125M
OPT-175B	96	96	12,288	175B

Mars, Mourad. "From Word Embeddings to Pre-Trained Language Models: A State-of-the-Art Walkthrough." Applied Sciences 12.17 (2022): 8805.

Why have Transformers taken over?

- Self-attention and subword tokenization are brilliant innovations
- Huge efforts to build massive corpora of text
- Allows for very big architectures
  - Previous approaches worked serially - one word at a time
  - Transformers can parallelise very well
  - Can be implemented very effectively using **GPUs** (which PyTorch & Tensorflow does well)



https://www.nvidia.com/en-gb/data-center/h100/

Transformers summary

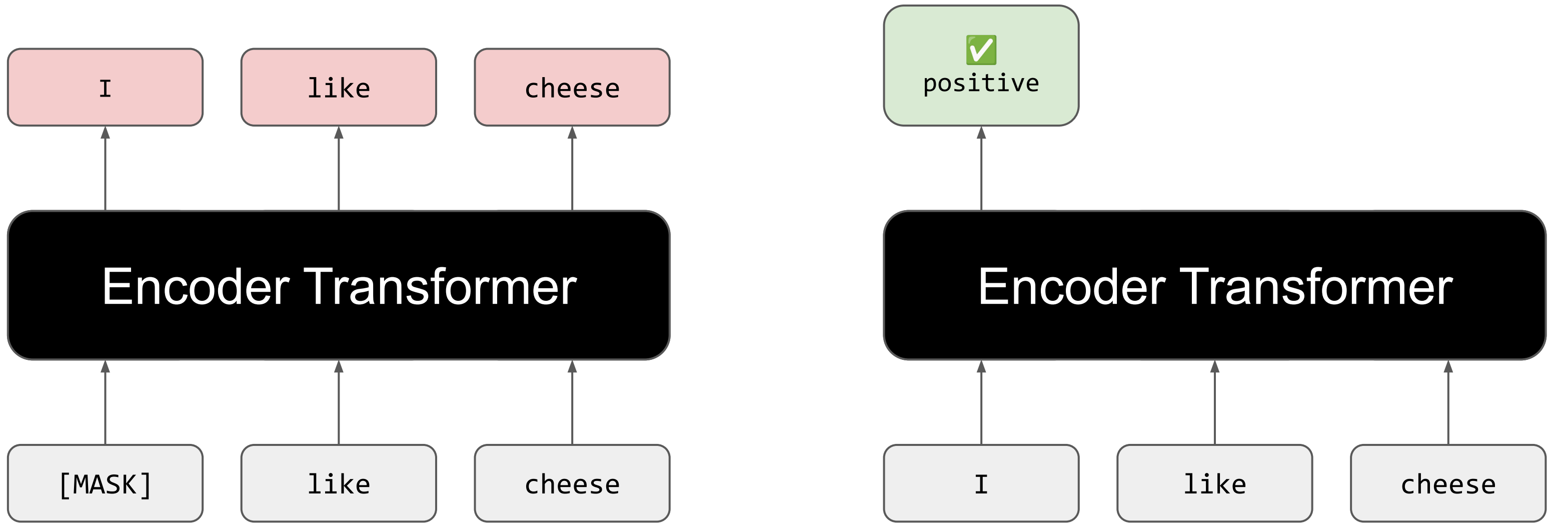
- Transformers are a very deep neural network architecture
    - Many layers (from 12 up to 96)
- Combines self-attention, subword tokenization, positional encodings and other neat tricks
- Encoders can see all input words and are typically trained with masked language tasks
- Decoders can only see previous input words and are trained with causal language tasks
- Encoder and decoder originally proposed for machine translation task
- Very parallelizable so models have got huge and require big memory GPUs

Breather

Pretraining & Fine-tuning

The pretraining & fine-tuning paradigm

Pretraining: Build a language model that is good at a language modelling task (e.g. masked language model)
Fine-tuning: Re-use most of the deep learning network for another task (e.g. text classification)

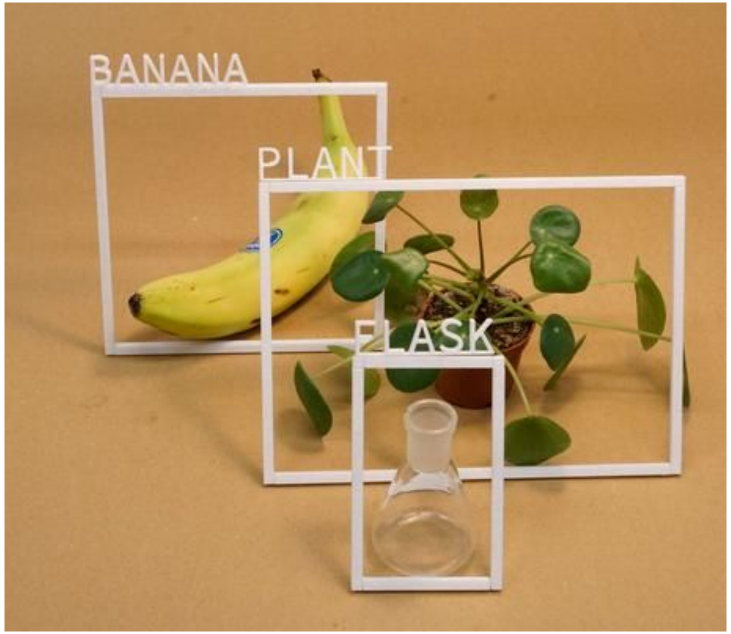


Transfer learning

Idea: Train an ML system on **one task** and then adapt it to another **new task**.

Goal: It performs better on the new task than a ML system trained only on the new task

Idea was very successful in computer vision. ML systems that can already identify some types of shapes are easier to adapt to new problems



Transfer learning

Idea: Train an ML system on **one task** and then adapt it to another **new task**.

Goal: It performs better on the new task than a ML system trained only on the new task

Idea was very successful in computer vision. ML systems that can already identify some types of shapes are easier to adapt to new problems

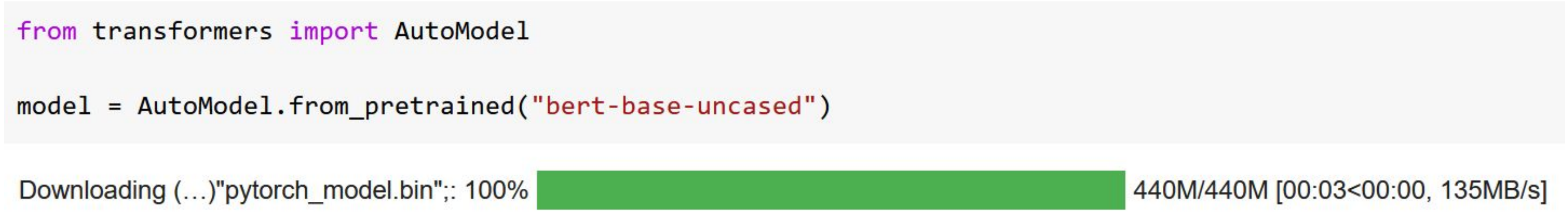
A language modelling problem (e.g. masked language modelling)

Another language-related problem (e.g. text classification, summarization, information extraction, etc)

Pretrained models

- Big repository of pretrained (and fine-tuned models) that you can download
- Loaded with `transformers` Python library
- Someone has already done the hard-work pretraining a model
- Use their hard work and finetune it to your needs

HUGGING FACE

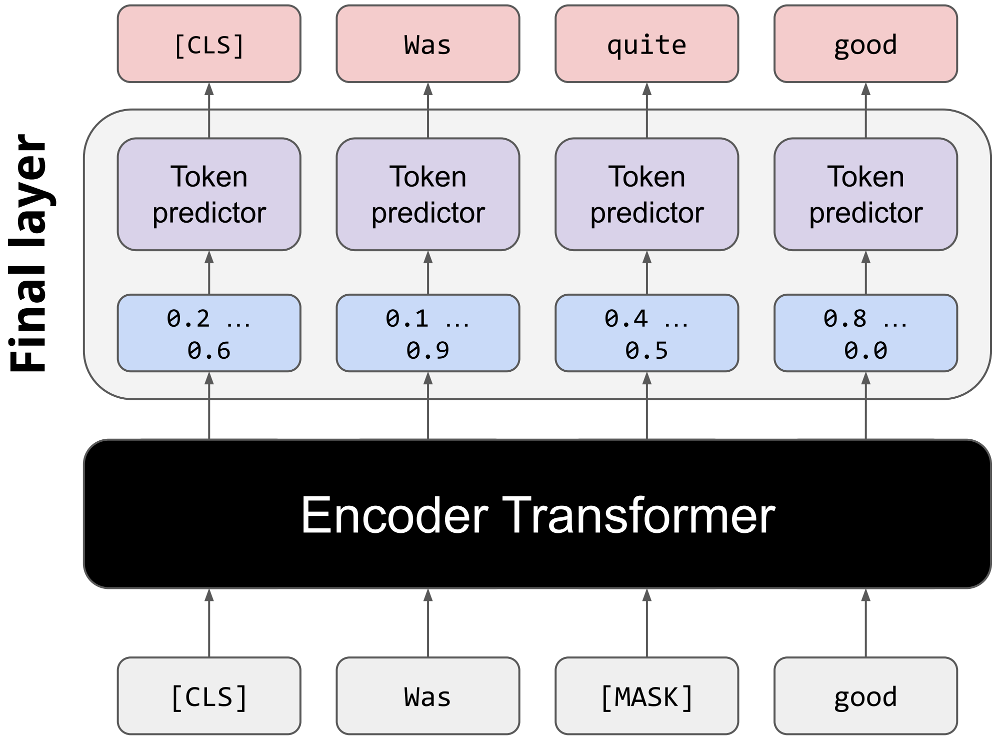


```python
from transformers import AutoModel

model = AutoModel.from_pretrained("bert-base-uncased")
```

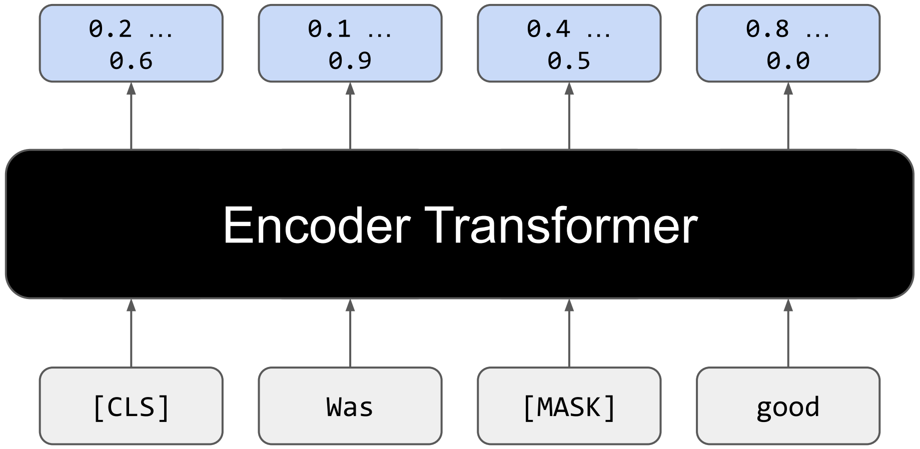
```
Downloading (…)"pytorch_model.bin";: 100%  440M/440M [00:03<00:00, 135MB/s]
```

Chopping off the head (of a Transformer)!



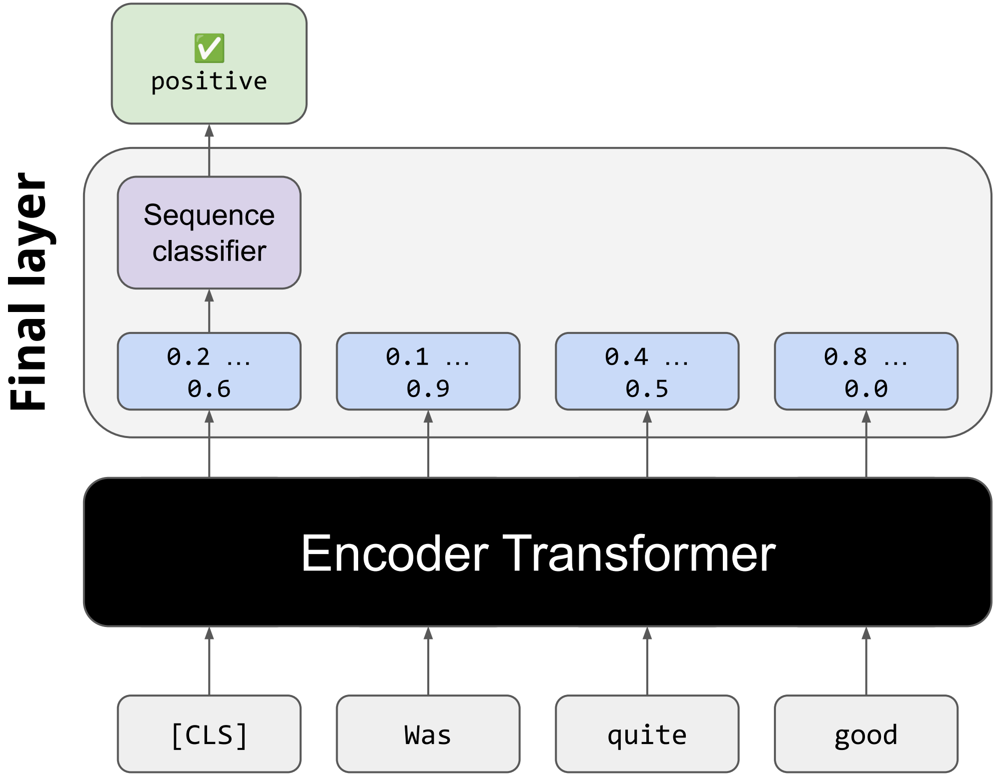
- The head of a Transformer model is the last layer that makes the final predictions
- In a pretrained model, it is outputting the goal of the language modelling task
  - e.g. predicting masked words or predicting the next token
- We don't want that output anymore. So remove the last layer!

Chopping off the head (of a Transformer)!



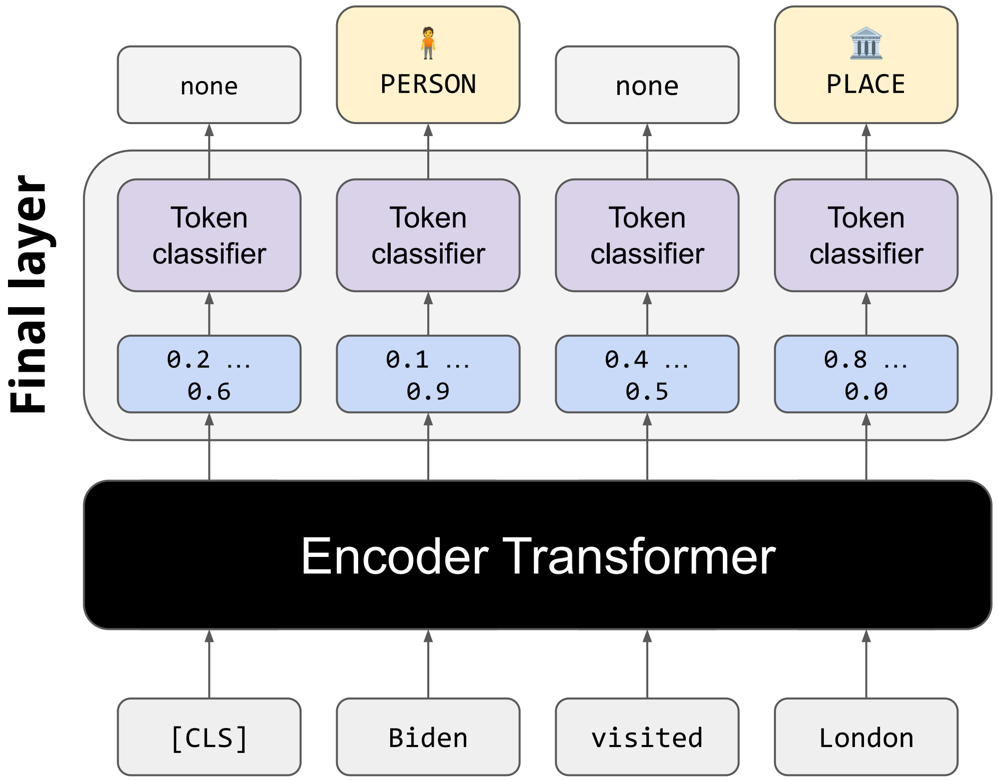
- The head of a Transformer model is the last layer that makes the final predictions
- In a pretrained model, it is outputting the goal of the language modelling task
    - e.g. predicting masked words or predicting the next token
- We don’t want that output anymore. **So remove the last layer!**

Swapping in another head (for text classification)



- Let’s add a layer that fits another task: sequence classification (also known as text classification)
  - Sentiment prediction is one example
- It uses only the context vector for the special `[CLS]` token as input to another small neural network to predict labels
  - e.g. positive, negative sentiment

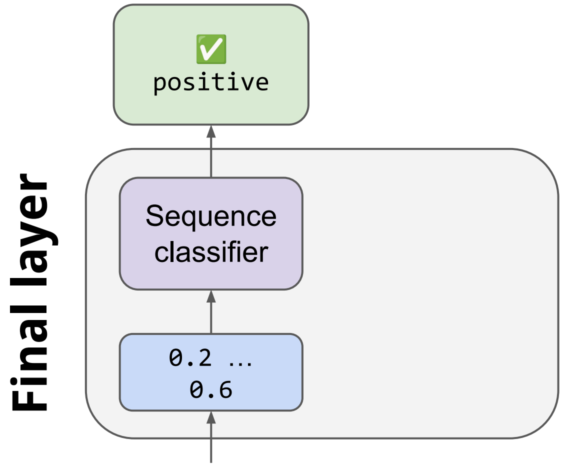
Another head for token classification



- Which token are places, people, etc?
- We use the context vector for each token and feed it to another small neural network
- This neural network predicts the type of entity each token is
  - e.g. places, genes
  - for biomedical purposes: drugs, diseases

More about this when we talk about **Information Extraction** later in the course

Training the new head!



- When a new head is added, it doesn't know how to make the predictions immediately
- The neural network for the final predictions (e.g. the sentiment predictor) is initiated with random weights
  - So will initially give random output
- It needs to be trained
  - You need inputs and expected outputs
  - You can train only the head (known as freezing the transformer)
  - Or train the whole network including updating the pretrained transformer

Pretraining and fine-tuning summary

- Training a transformer on a language task (e.g. masked language modelling) is known as **pretraining**
- Adapting the language model to a new task and training it further is **fine-tuning**
- Hugging Face and the `transformers` library provide a repository of useful models
- Fine-tuning for a new task involves changing the head of the Transformer network for a task-specific head
- It also requires additional training (with task-specific annotated training data) which may or may not update the pretrained Transformer

Feedback for Class Reps

CS Level 3
Semester 2
Feedback Form

(from class reps)