Natural language processing

Lecture 7: Transformers (BERT)

Roberta Rocca
Assistant Professor, IMC & CHC
✉️ roberta.rocca@cas.au.dk

Exercise

Try to reconstruct *****in broad strokes***** what a model doing language modeling with attention looks like and how it works.

Try to explain: which computations attention involves, which purpose it serves, and what the overall logic is.

Recap: the bigger picture

- Remember: our fundamental question is that of identifying computational methods and architectures that make it possible to **represent language quantitatively**

- We want our representations to **reflect fundamental aspects of meaning** (e.g., similarity relations)

- We want our representations to be good inputs for models that perform **downstream linguistic tasks** (from labelling and classifying text, to complex tasks like question answering)

- There are two angles to this:
  - **Scientific questions**: what does it take to build a system that can encode and operate on language? And how will these differ from our way of processing and encoding language?
  - **Applied aspect**: Real-world applications of these technologies

Recap: we started from static encodings

- The “**distributional hypothesis**” (*you can know a word by the company it keeps*) is a convenient theoretical paradigm for NLP

- First breakthroughs: count-based and “static” predictive models (Word2Vec). Words are **vectors** resulting from more (predicting) or less (counting) sophisticated ways to encode the distribution of *contexts* a word tends to occur in

- Interestingly, these vectors have **geometrical properties** that reflect dimensions of meaning (e.g., similar words -> close-by vectors)

- Unfortunately, these methods ignore some pretty important **aspects of meaning**, i.e., the role of word order and context-dependencies

Recap: then we introduced recurrence

- **RNNs** and **LSTMs** offer a solution to this!
- These are architectures where the *same* computations are applied sequentially to each bit of the input: by doing so, they build cumulative representations of sequences (*hidden states*) which are *sensitive to order*
- With these, we can do *language modeling* (predict the next word), which is a great way to learn representations of language, as it hinges on semantic & syntactic knowledge
- Unfortunately, these architectures are subject to the **information bottleneck**. There is only so much information that can be packed into a hidden state
- This is not convenient for sequential tasks where **dynamically accessing specific information on different parts of the input** may be important (e.g., machine translation, language modeling, etc)

Recap: dynamic encodings with attention

- **Meet attention!**

- Attention is a mechanism that allows models to **access representations of each token** in a dynamic way

- Representations for token *t* are a function of a static representation of this token, and a weighted sum of representations for **all other tokens**

- Weights are computed through simple dot products (though there are more sophisticated functions)

- This yields **contextualized representations** of the token *t* which **dynamically encode information about the rest of the sequence** (and eliminate the need for **recurrence**)

- The resulting representations are much better for downstream tasks (e.g., MT, LM)

Today’s plan

- **Self-attention** in **transformers**
- What else is included in a **transformer architecture**?
- **BERT**: training tasks, *transfer learning*, and applications
- **Interpreting BERT**: some examples from the BERTology

How does self-attention work in a transformer?

We want to use self-attention to produce a **representation of token *t*** which **also includes information about all other elements in the sequence** (without using recurrence)

1. Take *t* as input
2. Convert it to a vector
3. Compare it to all other tokens in the sequence
4. Produce a new, *contextualized* representation of *t* (output)

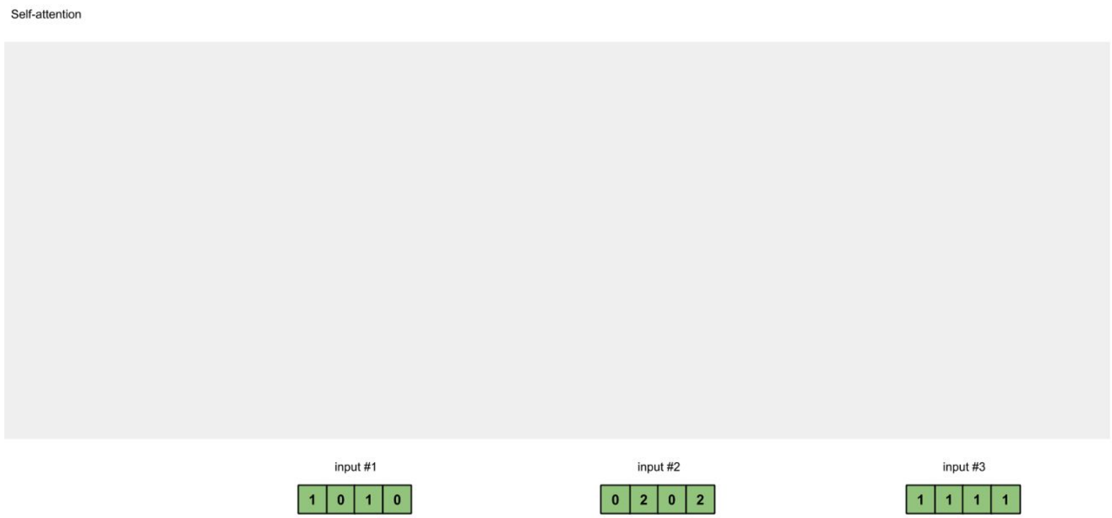
We want to do this **for all tokens *simultaneously***

1. Initialize input

- The first thing we do is that we **initialize each of our input as a embedding vectors**

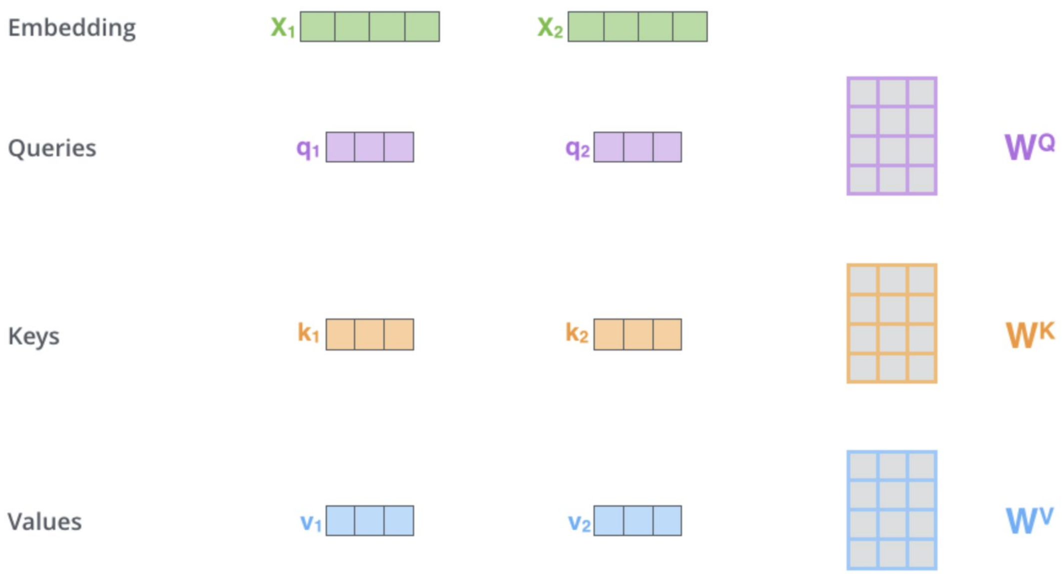
- In this example, we use 3 inputs (i.e., three words) whose initial embedding has dimensionality 4
  Input 1 = [1, 0, 1, 0]
  Input 2 = [0, 2, 0, 2]
  Input 3 = [1, 1, 1, 1]

- This is just a **word embedding**



2. Transform input into *key*, *query*, *value*

- The **input is projected into three separate vectors**: a query, a key, and a value
- This is done by **multiplying the input by a weight matrix** (which can also alter, e.g., reduce, the dimensionality of the input)
- We have **three weight matrices** (learnable parameters): Q, K, V



From Alammar (2018)

3. Get Q,K,V vectors

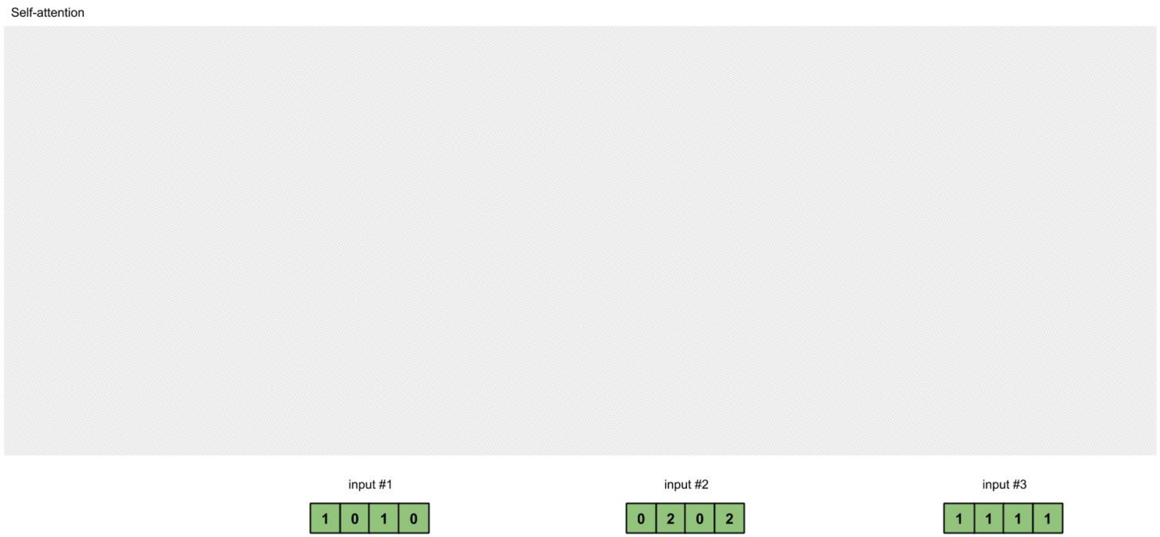
- In this example, the weight matrices are

$$W^Q = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

$$W^K = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \end{bmatrix}$$

$$W^V = \begin{bmatrix} 0 & 2 & 0 \\ 0 & 3 & 0 \\ 1 & 0 & 3 \\ 1 & 1 & 0 \end{bmatrix}$$

- We do this for all tokens



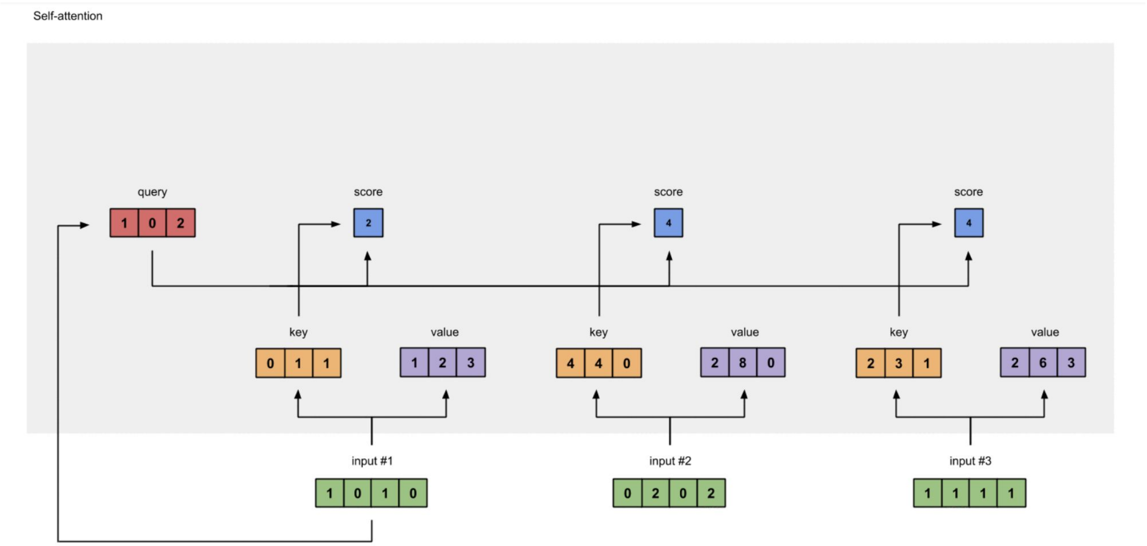
From Karim (2019)

4. Calculate attention scores

- To obtain ***attention scores***, we simply take the dot product between the query from Input 1 (red) and all of the keys (orange)

- Since there are 3 key representations, we obtain 3 attention scores (blue)

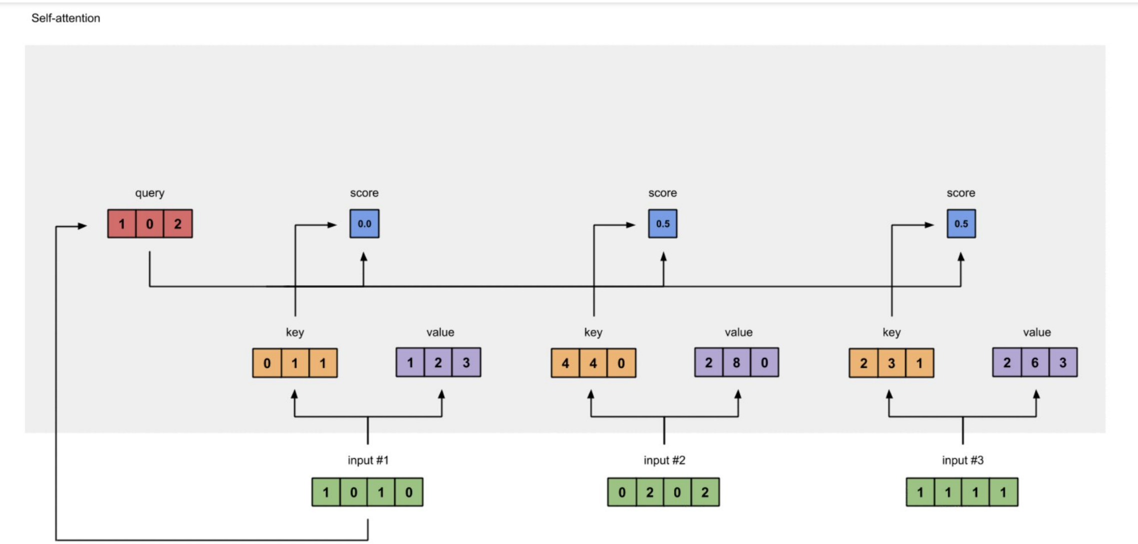
- Notice how we get the dot product between the query and key for input 1, too



From Karim (2019)

5. Softmax and multiply

- We then **run all of the scores through a softmax function**
- *softmax*([2,4,4]) = [0.0, 0.5, 0.5]
- These are the **attention weights**, which we then multiply by the *value* vectors, creating weighted values

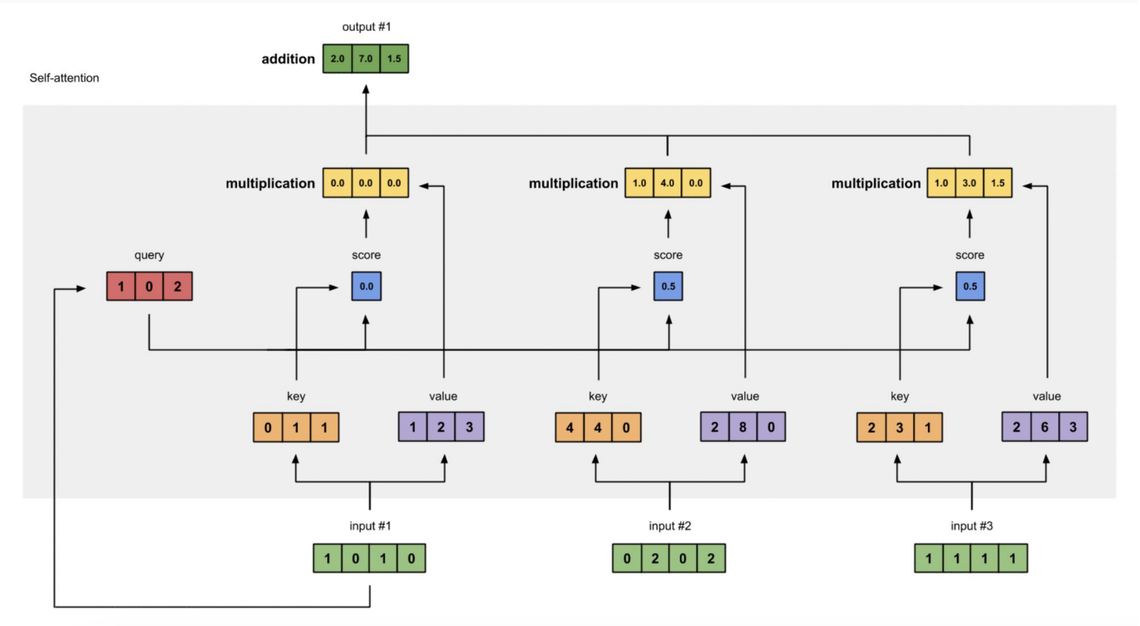


From Karim (2019)

6. Sum weighted values

- Lastly, **we take all *weighted values*** (yellow), **and sum them element-wise**

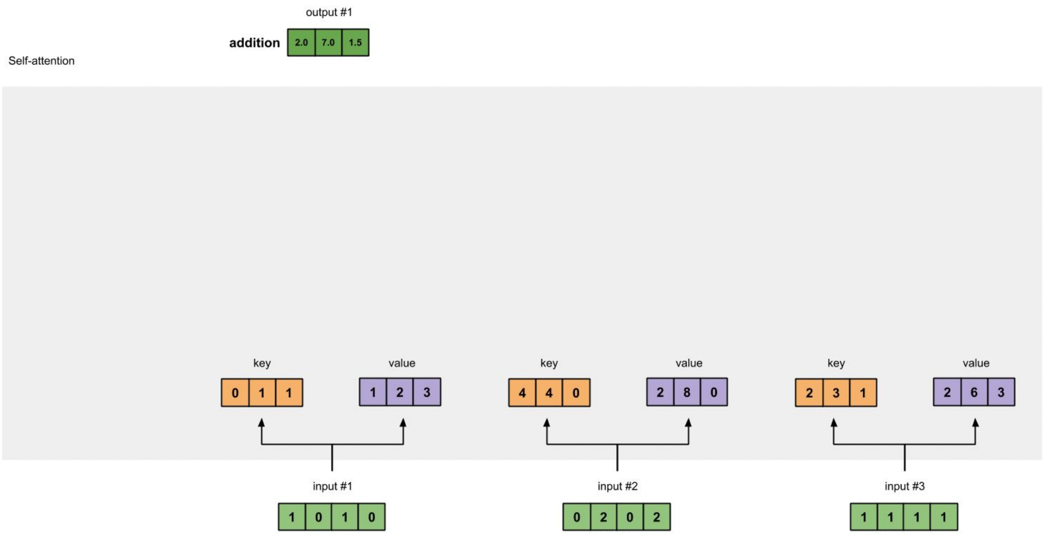
- The **resulting vector** (dark green) is based on the **query representation from Input 1, interacting with all other keys** (including itself)



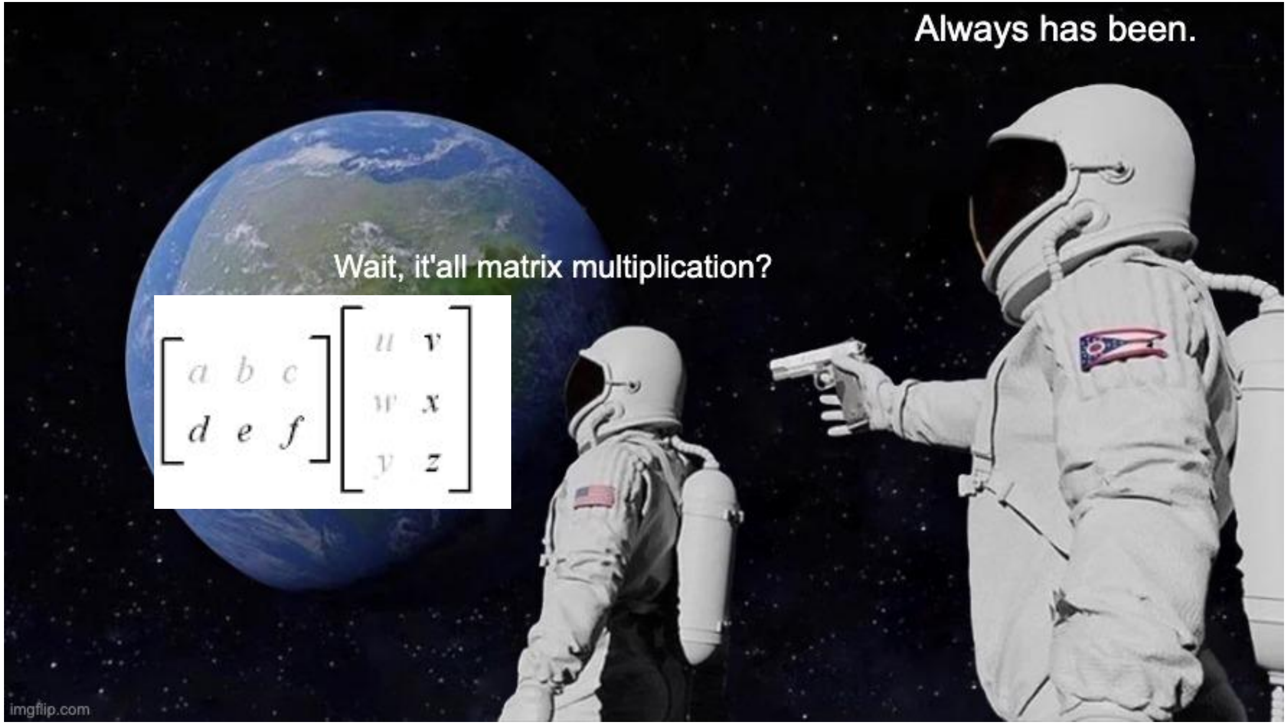
From Karim (2019)

7. Repeat for remaining inputs

- We then repeat the **same steps for the subsequent inputs**
- So we're using **self-attention to learn contextualized output embeddings from our "static" inputs directly**
- No hidden states, no recurrence, same result: we are free!



From Karim (2019)


Always has been.
Wait, it'all matrix multiplication?
$\begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix} \begin{bmatrix} u & v \\ w & x \\ y & z \end{bmatrix}$
imgflip.com

Language modeling with self-attention

- In practice, this happens simultaneously (matrices!)
- Let **X = [$\mathbf{x}_1$; ... ; $\mathbf{x}_T$]** be a matrix of input vectors (dimension: **T x d**)
- Let **K, Q, and V** be the weight vectors, through which we project the input into *key*, *query*, and *vector* values
- The output of a self-attention layer is:

$$output = \underbrace{\boxed{softmax(XQ(XK)^T)}}_{\textbf{Attention weights (T x T)}} \times XV$$

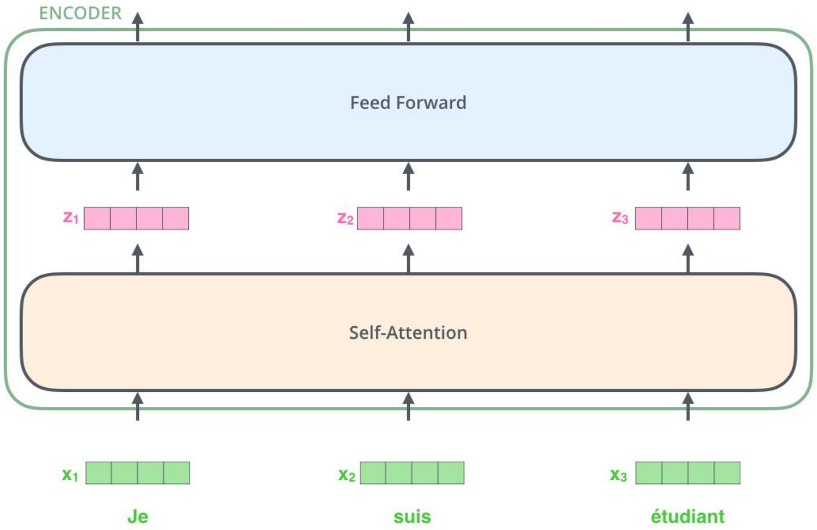
Inside a transformer block

- **Each transformer block** comprises a self-attention mechanism like we have just seen

- Transformer blocks are not just self-attention: we need some non-linearities (**Q: why?**)

- The outputs of the self-attention are passed through a feedforward layer (the *same* for all outputs)

- Note: in transformers, we often use *scaled* dot product attention



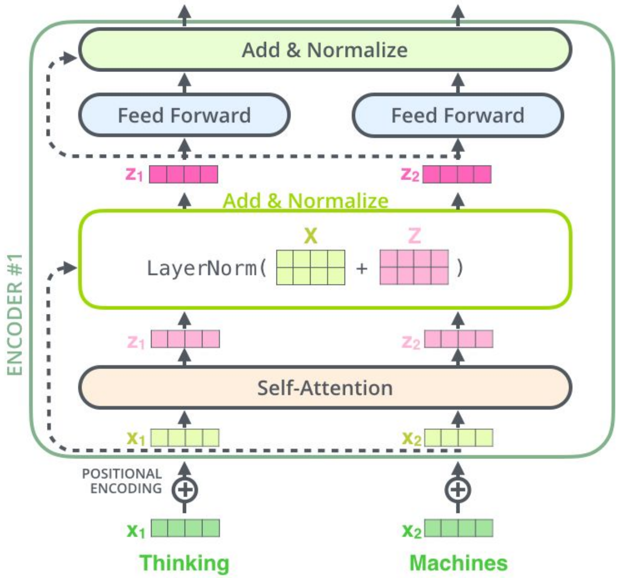
From Alammar (2018)

Inside a transformer block

- Actually, a transformer block contains **two more things**

- **Residual connections** (vectors from previous steps are added to the output of a given steps), to maintain information across layers and stabilize training

- We also apply **layer normalization** to make sure each unit / dimension has same mean/variance



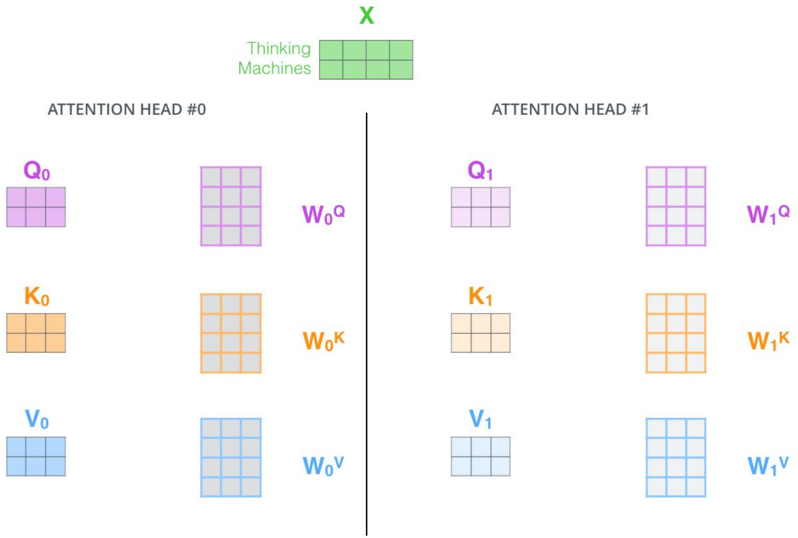
From Alammar (2018)

Multi-head self-attention

- Transformers do not use only one attention mechanism – they **use many *simultaneously***

- With **multiple *attention heads* at the same time**, i.e., separate weight matrices for Q,K,V, to attend to other parts of the sequence *in different ways* (e.g., syntax, semantics…)

- The basic transformer setup uses 6 encoder units and 6 decoder units, each with 8 self-attention heads



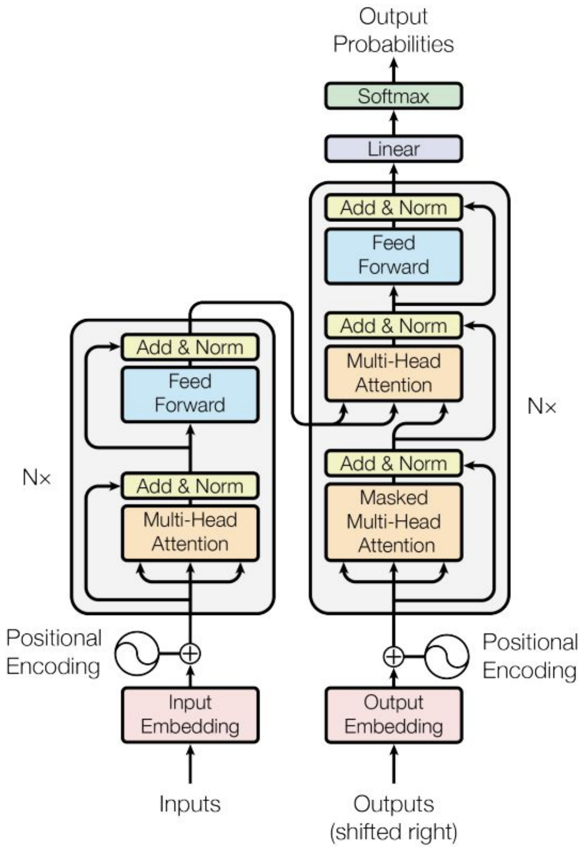
From Alammar (2018)

The transformer architecture

- Self-attention is the basic building block of transformers

- Transformers are made up of **stacked transformer blocks**, each of which involves **attention** and a few additional calculations

- There are different types of transformers: **encoder only** (BERT), **decoder only** (GPT), and **encoder-decoder** (the original transformer, with both *self* and *cross* attention)



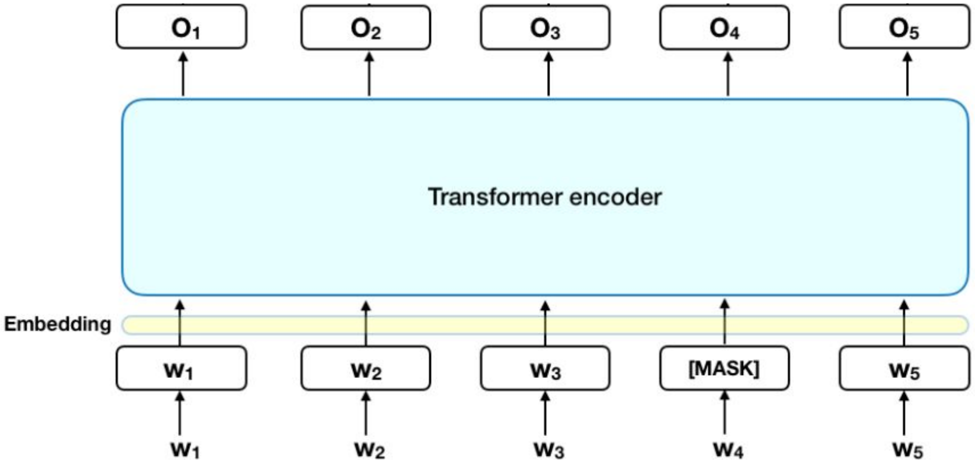
Meet BERT!

- All of this is applied in BERT (**B**idirectional **E**ncoder **R**epresentations from **T**ransformers, Devlin et al., 2019)

- Encoder-only architecture with a series of **stacked transformer block**

- Each **block** in the encoder has multiple attention heads, and it transforms the input of the previous block into a new representation (up to an output, **O**)

- "Hierarchical" contextualization, capturing relations at different *levels* (e.g., role in the syntactic structure, semantic dependencies, etc)



How will this yields good representations?

- Ok, this idea of contextualizing representations sounds cool: but do we make sure that the output vectors are *meaningful* and *useful*?

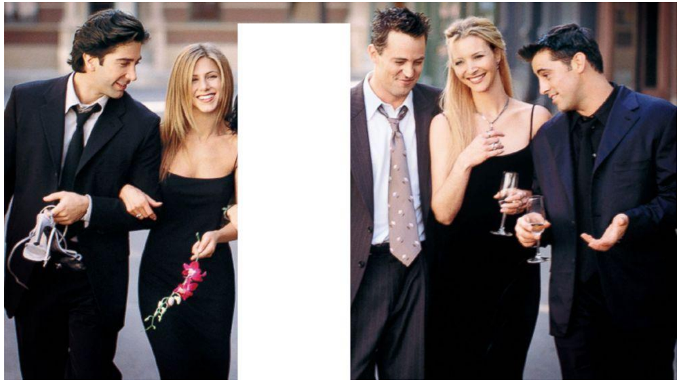
- It all depends on the **weights** in self-attention layers (and of the embedding layer, and of FNNs in transformer blocks)

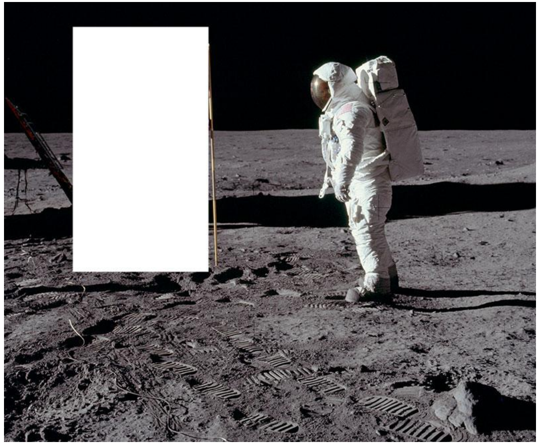
- We use the good old idea of *language modeling* as a training task: we train our model to yield representations if tokens that can be used to predict what the *contexts* that surrounds them may be

- We will mask a set of words/tokens, and try to guess which words/tokens they were leveraging information on the surrounding tokens

- This is a particular version of language modeling, *masked language modeling*, but it is just a sophisticated version of what we were doing with CBOW Word2Vec

What's in a mask?





BERT training: input

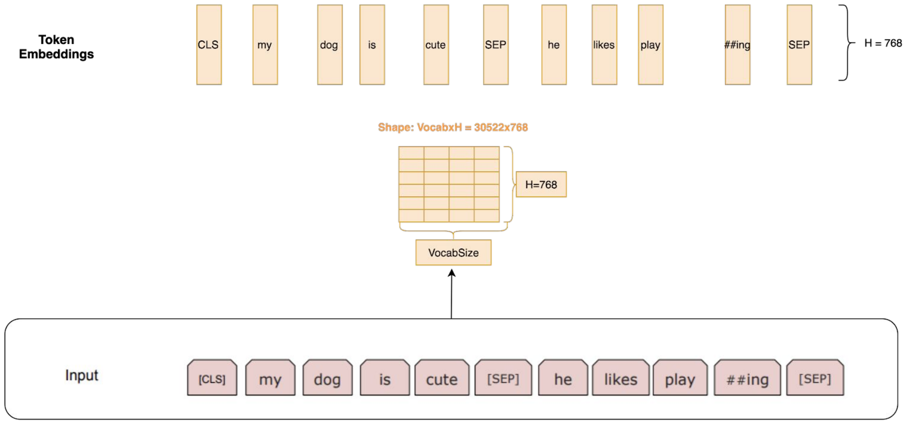
- Input to BERT always consists of one or two sentences (technically *sequences*)
    - [CLS] is added to the start of the first sequence
    - [SEP] is used to separate sequences and delimi the end of the sequence

- We do sub-word *WordPiece* tokenization (Wu et al. 2016): 'playing' -> 'play', '##ing'

- Smaller vocabulary and deals well with out-of-vocabulary words

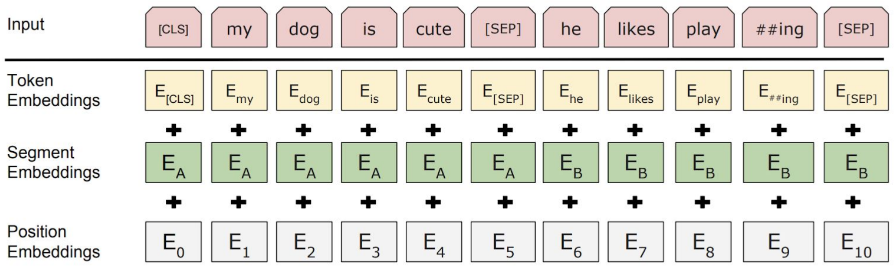
BERT training: embeddings

- For each token, we look up a **static embedding from a weight matrix** (like in Word2Vec)
- Note that **these weights are updated over training**



BERT training: augmenting input embeddings

- We then augment these embeddings with a **position embedding** (Q: Why?) and a **segment embedding**
- **Position embeddings** are learned and that encode the position in a sequence
- There are two possible **segment embeddings** (first or second sentence)
- So the final input passed to BERT is
  (Wordpiece)Token Embeddings
  + Segment Embeddings
  + Position Embeddings



How does BERT learn?

- BERT is optimized for **two tasks simultaneously**
  - **Masked language modelling** (MLM): reconstruct the missing word
  - **Next sentence prediction** (NSP): do the two sequences follow each other in the original corpus?

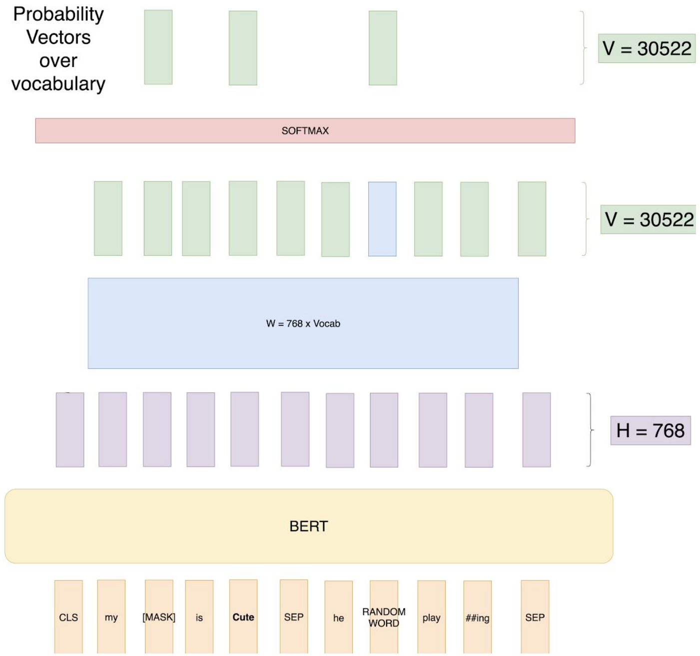
- The rationale is the following:
  - MLM is language modeling: we know that **language modeling is a good way to force a model to develop a number of linguistic skills**
  - NSP is another proxy for linguistic knowledge which goes beyond word-level: it captures **“higher-level” coherence and semantic understanding**
  - In practice, NSP is hardly ever used beyond BERT

Masked language modeling

- Let's focus on MLM first: **15% of all tokens are masked**, and **we want to reconstruct the missing word** from the final embedding of the [MASK] token

- Usual trick: we **feed the contextualized vector for the [MASK] tokens to a classification layer with softmax**

- During training, the loss **(Q: which loss?)** is calculated only **over masked words**



What if we wanted to do *forward* language modeling?

- Q: can we use the same logic to do *forward language modeling* on the whole sequence?

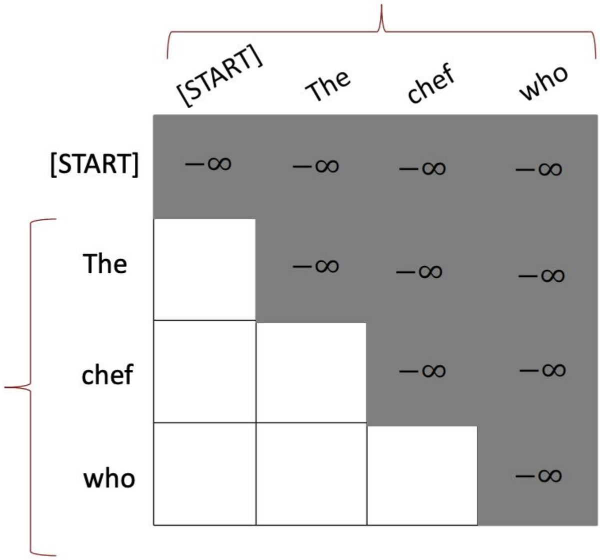
What if we wanted to do *forward* language modeling?

- Q: can we use the same logic to do *forward language modeling*?

- To prevent a model from **accessing information in the future**, we mask attention weights (before normalization) by setting weights for future tokens to **-Inf**

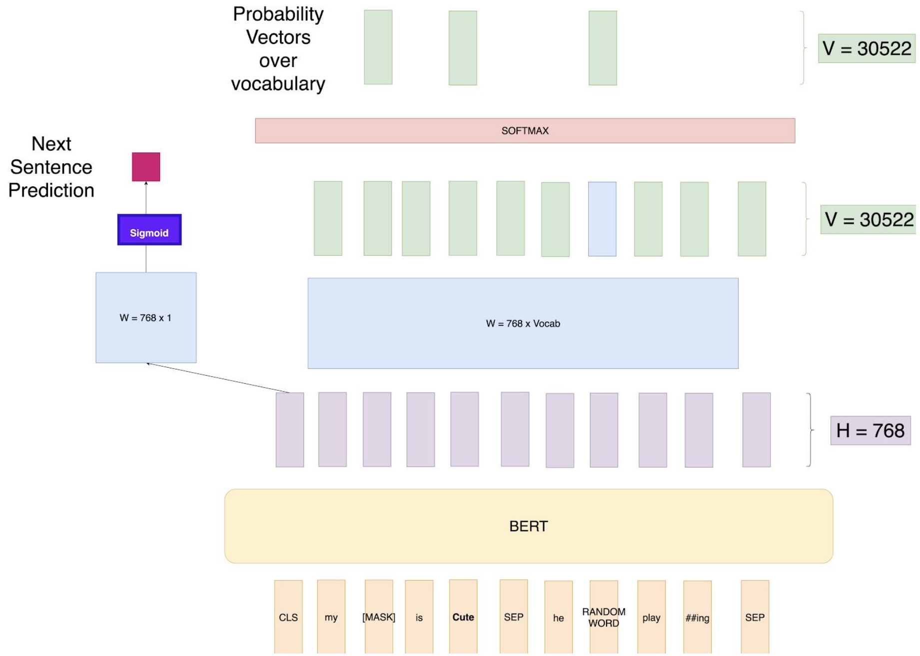
- Plugging -Inf into a **softmax** yields weight zero

- This allows the model to perform forward language modeling



Next sentence prediction

- While creating the training data, we choose the **sentences A and B for each training example**
- 50% of the time B is the actual next sentence that follows A - labeled as ***IsNext***; 50% of the time it is a random sentence from the corpus - labeled as ***NotNext***
- We then **use the output encoding of the [CLS] token** to predict the **probability that the two sentences follow each other**
- BERT trains both **MLM and NSP** objectives **simultaneously**



An example: contextualised embeddings

[Something we could *not* do with static embeddings, e.g., word2vec]
https://storage.googleapis.com/bert-wsd-vis/demo/index.html?#word=die

Coenen et al. (2019), *Visualizing and Measuring the Geometry of BERT*

Transfer learning with BERT

- Through MLM and NSP pretraining BERT learns two things
  - Contextualized **token-level representations**
  - **Sequence-level** representations (**[CLS]**)

- They *do* reflect intrinsic properties of meaning

- Are they good for downstream tasks?

- So far, our approach to using language to perform **downstream tasks** (e.g., sequence classification) has been to use a model (e.g., Word2Vec) to extract encodings, then feed them to a **new** architecture

- With **BERT**, we can **swap the head** (all layers above the transformer encoder output), introduce a new layer that fits our target task, and keep training **the same architecture** (potentially *freezing* some layers)

Transfer learning with BERT

- Our ***pretrained*** architecture already encodes **a lot of useful information on language/meaning!** It is just a matter of **adapting it a bit to the new task (*fine-tuning*)**

- This approach, called **transfer learning** (and made possible with libraries for open-source sharing of pretrained models, such as **HuggingFace's libraries**), outperforms by orders of magnitude all previous architectures on pretty much *all downstream* NLP *tasks*

- This has made transformer **ubiquitous in NLP**

Fine tuning on NLP tasks

- Sentence Pair Classification tasks
  Linear + Softmax layer on top of the CLS output

- Single Sentence Classification Task
  Same

- Question Answering Tasks
  Given a question and a paragraph in which the answer lies. The objective is to determine the start and end span for the answer in the paragraph.
    - See also *text summarization*

- Single Sentence Tagging Task
    - POS tagging, NER, etc. We just add a Linear layer of size (768 x n_outputs) and a softmax layer on top to predict

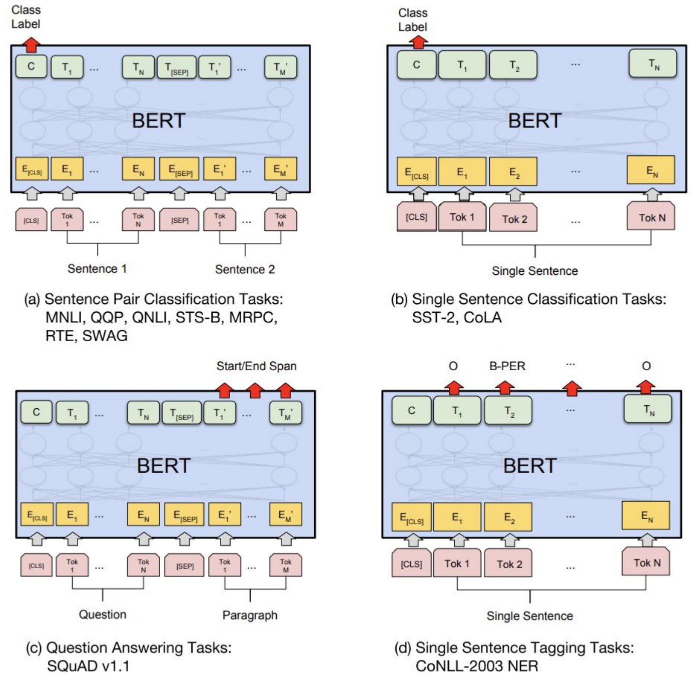


Figure 4: Illustrations of Fine-tuning BERT on Different Tasks.

BERT as a black box

- BERT did really, really well in a lot of tasks that previous models could not solve optimally

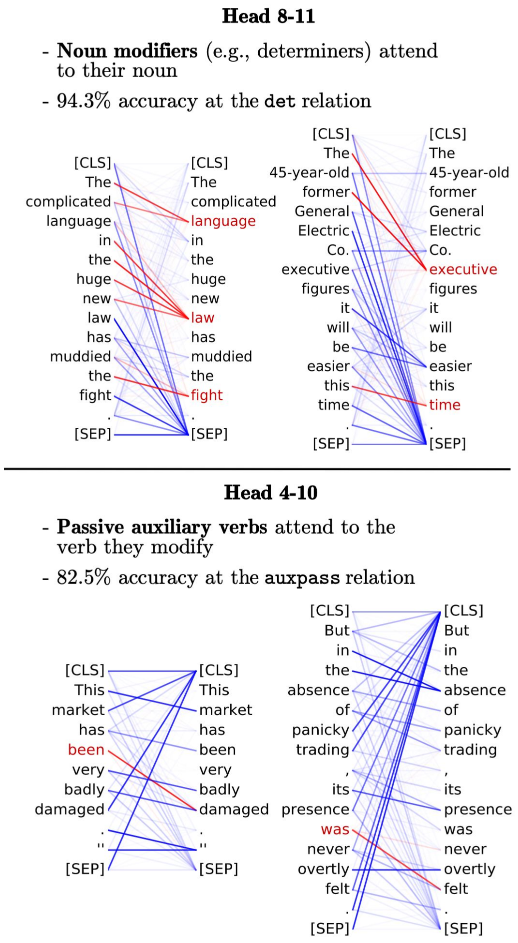
- But what is happening *inside* BERT? What does BERT really learn through pretraining? How do its attention mechanisms really work? Do we need all of them? And is BERT biased?
  - Is BERT learning anything about semantics/syntax, or is it just a very sophisticated parrot?

- Since the introduction of BERT, people have been trying to devise smart ways to "open the black box" and probe BERT's internal mechanisms and knowledge

- This literature is known as "**BERTology**"

What is attention doing?

- There seems to some evidence that specific attention heads model something about linguistic structure
- We can see two examples to the right here taken from Clark et al (2019)
- Head 8-11 (layer 8, head 11) seems to catch the relationship between noun phrases and their determiners
- Head 4-10 captures passive auxiliary structures



Psycholinguists tasks

- Ettinger (2019) devised a series of diagnostic tests in order to see what specific linguistics capacities are learned by language models

- These tests allow us to ask specific targeted questions about information used by LMs for generating predictions in context

    - Common-sense and pragmatic inference
    - Event knowledge and semantic roles
    - Negation

Context	$\text{BERT}_{\text{LARGE}}$ predictions
A robin is a ____	*bird, robin, person, hunter, pigeon*
A daisy is a ____	*daisy, rose, flower, berry, tree*
A hammer is a ____	*hammer, tool, weapon, nail, device*
A hammer is an ____	*object, instrument, axe, implement, explosive*
A robin is not a ____	*robin, bird, penguin, man, fly*
A daisy is not a ____	*daisy, rose, flower, lily, cherry*
A hammer is not a ____	*hammer, weapon, tool, gun, rock*
A hammer is not an ____	*object, instrument, axe, animal, artifact*

	Accuracy
$\text{BERT}_{\text{BASE}}$ $k = 1$	38.9
$\text{BERT}_{\text{LARGE}}$ $k = 1$	44.4
$\text{BERT}_{\text{BASE}}$ $k = 5$	100
$\text{BERT}_{\text{LARGE}}$ $k = 5$	100

Psycholinguists tasks

- BERT shows strong insensitivity to the meaning of negation, with preferring the category match every time

Context	$\text{BERT}_{\text{LARGE}}$ predictions
A robin is a ____	*bird, robin, person, hunter, pigeon*
A daisy is a ____	*daisy, rose, flower, berry, tree*
A hammer is a ____	*hammer, tool, weapon, nail, device*
A hammer is an ____	*object, instrument, axe, implement, explosive*
A robin is not a ____	*robin, bird, penguin, man, fly*
A daisy is not a ____	*daisy, rose, flower, lily, cherry*
A hammer is not a ____	*hammer, weapon, tool, gun, rock*
A hammer is not an ____	*object, instrument, axe, animal, artifact*

	Affirmative	Negative
$\text{BERT}_{\text{BASE}}$	100	0.0
$\text{BERT}_{\text{LARGE}}$	100	0.0

Summary

- We have looked at how self-attention works in the context of a transformer, and which additional components a transformer block includes

- BERT is a specific form of "encoder-only" transformer

- Using stacked transformer blocks (whose core mechanisms is attention), BERT can yield *contextualized* representations of tokens and sequence-level representations at the same time

- BERT is (pre)trained through two tasks: masked language modeling and next-sentence predictions

- After learning weights through pretraining, the same architecture (minus the head, which is replaced with a new one) can be fine-tuned on specific tasks is known as *transfer learning*

- This process, known as **transfer learning**, has revolutionized NLP

See you tomorrow!

Additional reading

Blogs

- Alammar, J. (2018). 'The Illustrated Transformer', https://jalammar.github.io/illustrated-transformer/
- Karim, R. (2019). 'Illustrated: Self-Attention', https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a
- Kazemnejad, A. (2019). 'Transformer Architecture: The Positional Encoding', https://kazemnejad.com/blog/transformer_architecture_positional_encoding/
- Weng, L. (2018). 'Attention? Attention!', https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html

Additional reading

Scientific articles

- Clark, K., Khandelwal, U., Levy, O., & Manning, C.D. (2019). 'What does BERT look at? An analysis of BERT's attention', *Proceedings of the Second BlackboxNLP Workshop on Analyzing and Interpreting Neural Networks for NLP*, pp. 276–286.
- Coenen, A., Reif, E., Yuan, A., Kim, B., Pearce, A., Viégas, F., & Watttenberg, M. (2019). 'Visualising and measuring the geometry of BERT', 33rd Conference on Neural Information Processing Systems.
- Kovaleva, O., Romanov, A., Rogers, A., & Rumshisky, A. (2019). 'Revealing the Dark Secrets of BERT', *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing*, pp. 4365-4374.
- Vig, J., Madani, A., Varshney, L.R., Xiong, C., Socher, R., & Rajani, N.F. (2021). 'BERTology meets biology: Interpreting attention in protein language models', *International Conference on Learning Representations.*
- Wu, Y. et al. (2016). "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation", *arXiv*, https://doi.org/10.48550/arxiv.1609.08144

Blog on the WordPiece tokenization algorithm in HuggingFace

https://huggingface.co/course/chapter6/6?fw=pt