香港城市大學（東莞）
City University of Hong Kong (Dongguan)
Celebrating the Establishment in Year 2024

Lecture 4: Transformers and pretraining-finetuning

CS6493 Natural Language Processing

Instructor: Linqi Song

Outline

- 1. Attention
- 2. Transformer
- 3. BERT
- 4. GPT

Problems with contextualized word embeddings

- RNNs/LSTMs have long-term dependency problems, where words/tokens are processed sequentially.
  - Information loss
  - Hard to compute in parallel
- Bi-directional RNNs/LSTMs feature fusion and representation ability is weak (compared with transformers).

From ELMo to BERT

ELMo

BERT

Embeddings from Language Models

Bidirectional Encoder Representations from Transformers

Attention – the main technique behind transformers

- Attention is all you need, Google, 2017.

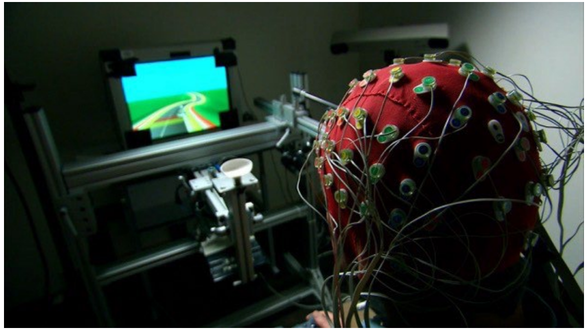
- Why attention?
  - To reduce the computational complexity.
  - To parallelize the computation.
  - Self-attention connects all positions in a sequence with a constant number of operations to solve the long-term dependency problem.
  - Self-attention could yield more interpretable models

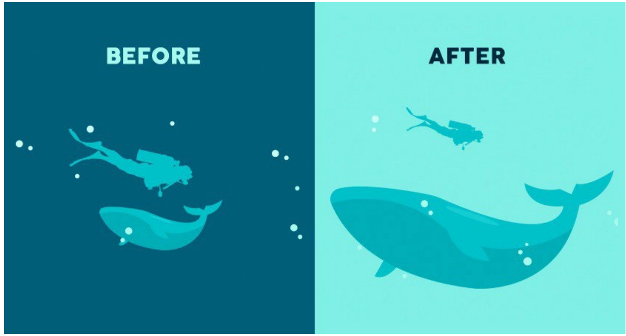
What is attention?

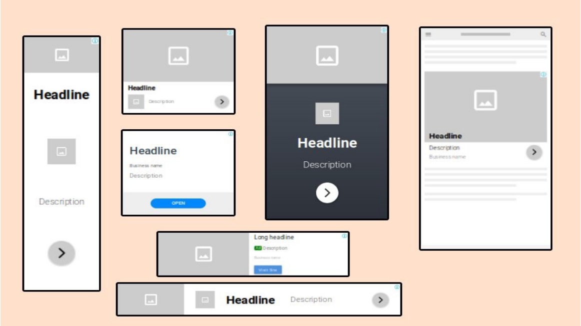
- What is the attention mechanism
  - An analogy to human's brain, to pay more attention to more important information.
  - Map a query and a set of key-value pairs to an output, where the query, keys, values and output are all vectors.
  - The output is a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key.

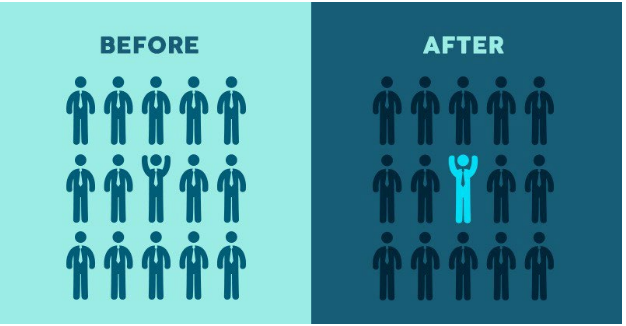
Attention in cognitive psychology and graphical design

- Cognitive psychology and graphical design
  - Which part of the screen attracts the most human attention: ad placement, news headlines, etc.
  - Visual hierarchy



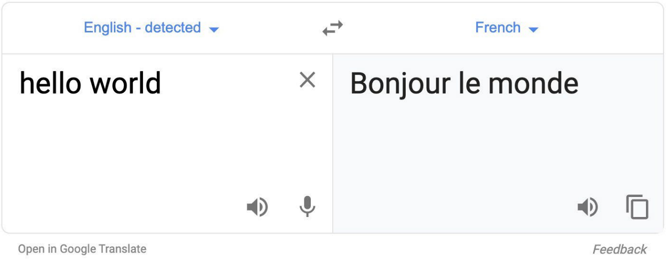




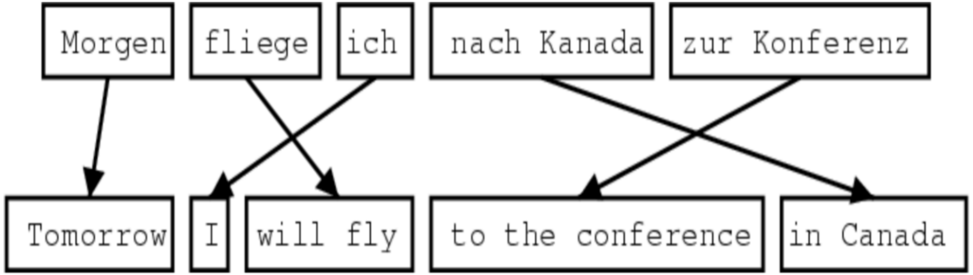


Attention in NLP: aligning while translating (1)

- Machine translation: translate source sentence in a language to a target sentence in another language.

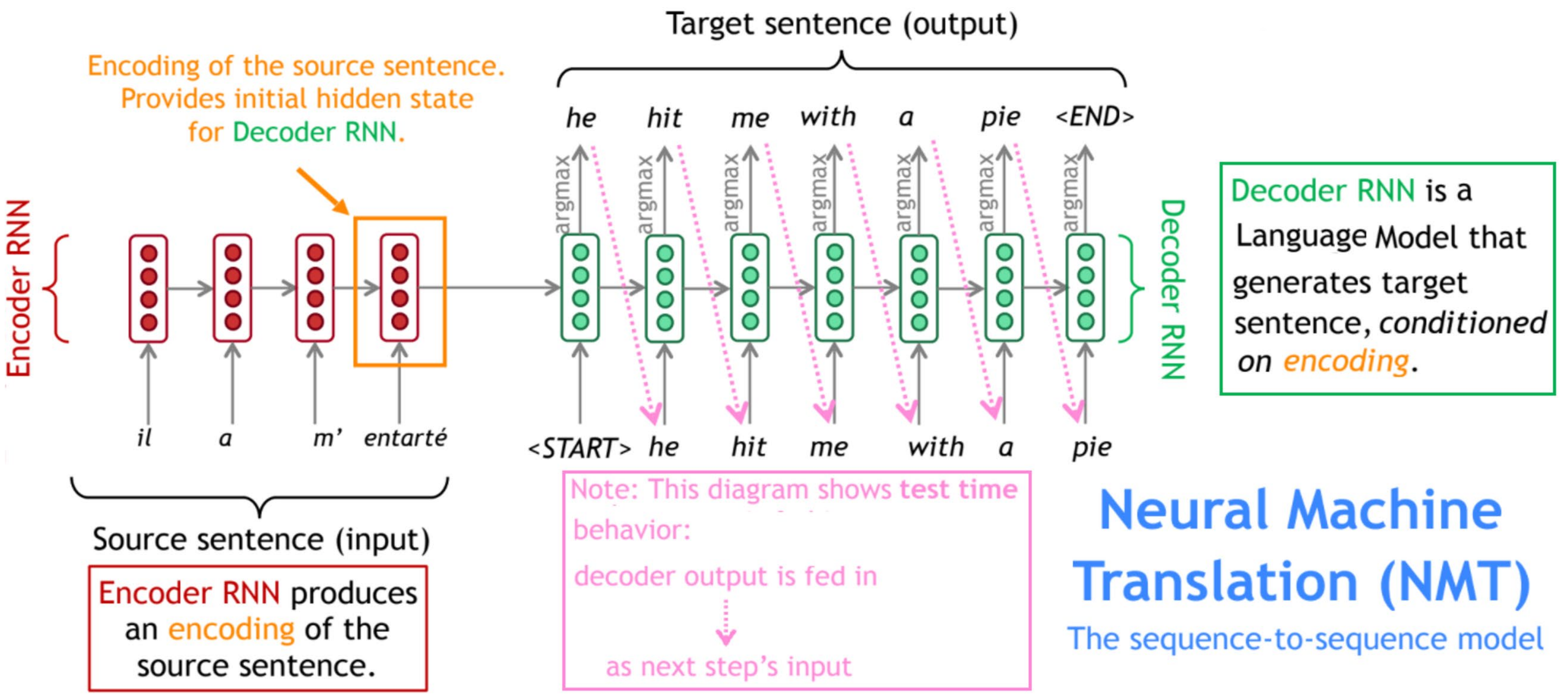


- Alignment in machine translation: relationship between source words and target words.



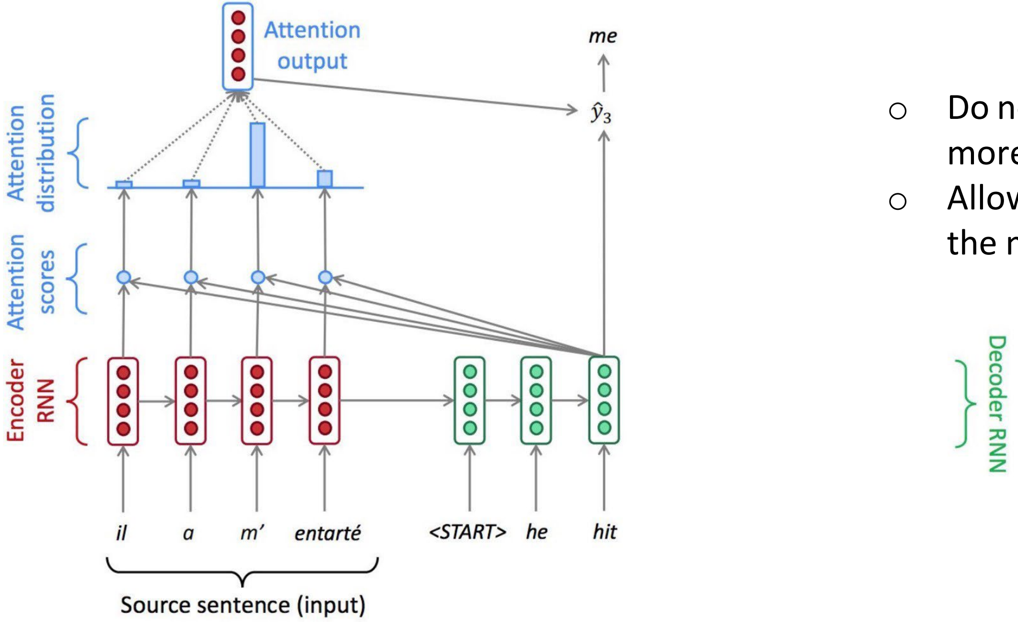
Attention in NLP: aligning while translating (2)

- Recall the RNN-based encoder-decoder example for neural machine translation



Attention in NLP: aligning while translating (3)

- Attention intuition: each time the proposed model generates a word in a translation, it searches for a set of positions in a source sentence where the most relevant information is concentrated (Bahdanau et al. 2015).

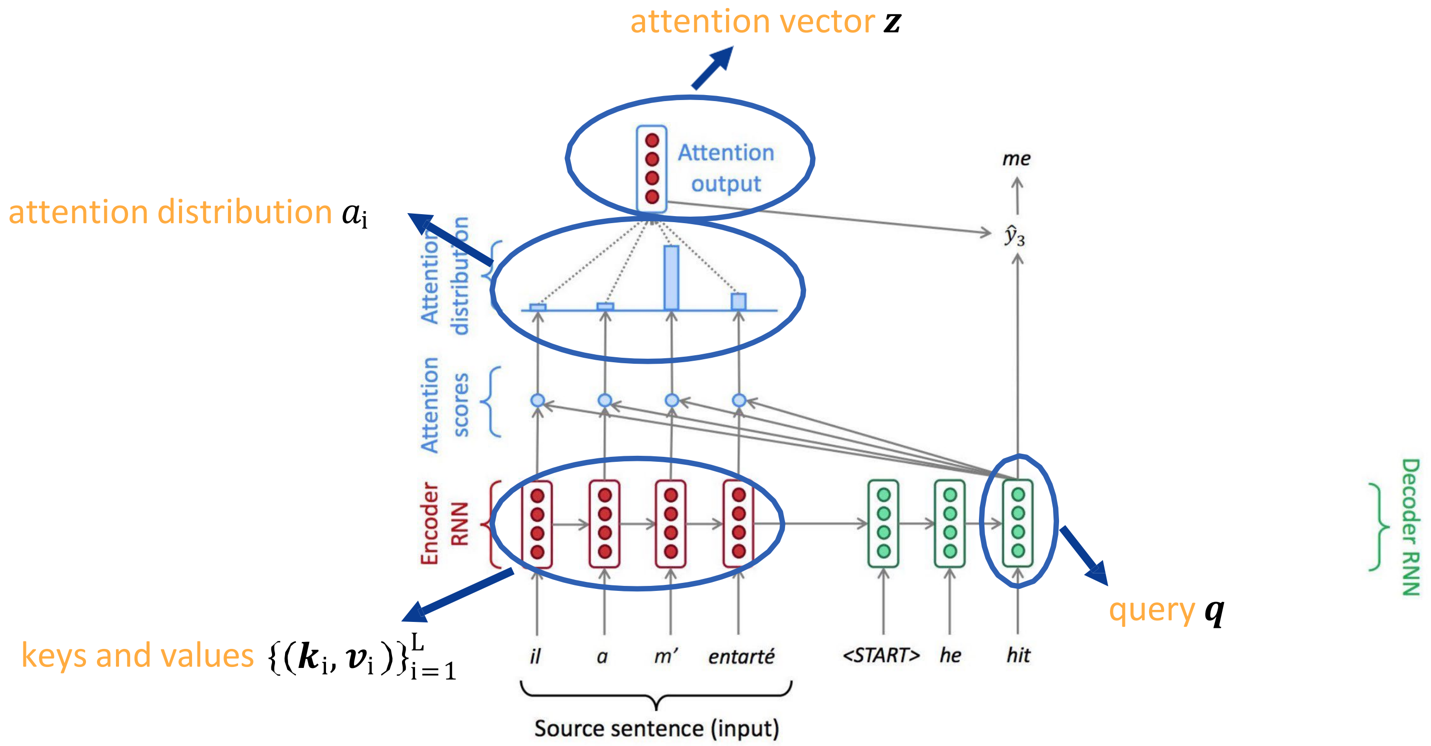


  - Do not encode all inputs into one vector: more information.
  - Allow adaptive selection to which should the model attend.

Attention: formal description

- Given a query vector $\boldsymbol{q}$, and a set of key-value pairs (all vectors) $\{(\boldsymbol{k}_{\mathrm{i}}, \boldsymbol{v}_{\mathrm{i}})\}_{\mathrm{i}=1}^{\mathrm{L}}$, we first calculate the similarity/attention score between the query and each key:
$$s_{\mathrm{i}} = similarity(\boldsymbol{q}, \boldsymbol{k}_{\mathrm{i}}).$$
- Normalize the similarity score to be between 0 and 1, and they sum up to 1. These are called attention distribution. One way is to use the softmax operation.
$$a_{\mathrm{i}} = softmax(s_{\mathrm{i}}) = \frac{\exp(\mathrm{s_i})}{\sum_{\mathrm{i}=1}^{\mathrm{L}} \exp(\mathrm{s_j})}.$$
- Compute the attention/context vector $\boldsymbol{z}$ as a weighted sum of values.
$$\boldsymbol{z} = \sum_{\mathrm{i}=1}^{\mathrm{L}} a_{\mathrm{i}} \boldsymbol{v}_{\mathrm{i}}.$$
- Keys and values are not necessarily the same, and they could be different as well, such as in machine translations.

Attention: example



Attention: similarity calculation

- Similarity/attention score calculation between the query and each key:
$$s_{\mathrm{i}} = similarity(\boldsymbol{q}, \boldsymbol{k}_{\mathrm{i}}).$$
- In general, every similarity measures can be used here, such as the cosine coefficient and Pearson correlation coefficient.
- Commonly used ones for neural networks:
  - Additive attention (Bahdanau et. al 2014):
$$s_{\mathrm{i}} = w_3 \tanh(\boldsymbol{w}_1^{\mathrm{T}}\boldsymbol{q} + \boldsymbol{w}_2^{\mathrm{T}}\boldsymbol{k}_{\mathrm{i}}).$$
  - Multiplicative attention.
$s_{\mathrm{i}} = \boldsymbol{q}^{\mathrm{T}} W \boldsymbol{k}_{\mathrm{i}}$, where $W \in \mathbf{R}^{\mathrm{d}_{\boldsymbol{q}} \times \mathrm{d}_{\boldsymbol{k}}}$, with $d_{\boldsymbol{q}}, d_{\boldsymbol{k}}$ being the dimensions of $\boldsymbol{q}$ and $\boldsymbol{k}_{\mathrm{i}}$.
  - Dot-product attention.
$$s_{\mathrm{i}} = \boldsymbol{q}^{\mathrm{T}}\boldsymbol{k}_{\mathrm{i}}.$$
  - Scaled dot-product attention.
$$s_{\mathrm{i}} = \frac{\boldsymbol{q}^{\mathrm{T}}\boldsymbol{k}_{\mathrm{i}}}{\sqrt{\mathrm{d}_{\boldsymbol{k}}}}.$$

Scaled dot-product attention in matrix form

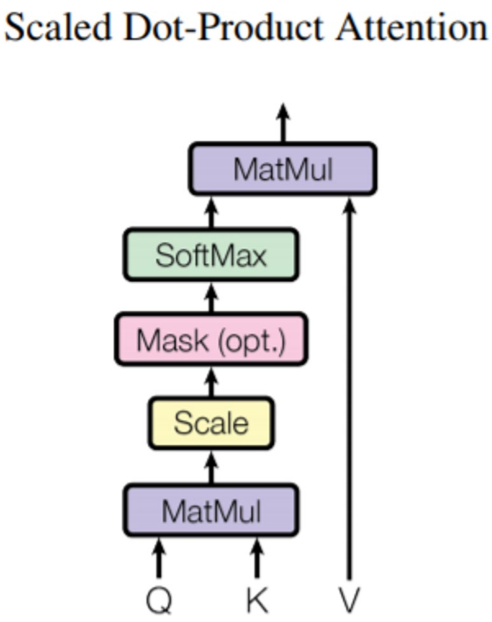
Q: a matrix formed by packing a set of queries

K: a matrix formed by packing a set of keys

V: a matrix formed by packing a set of values

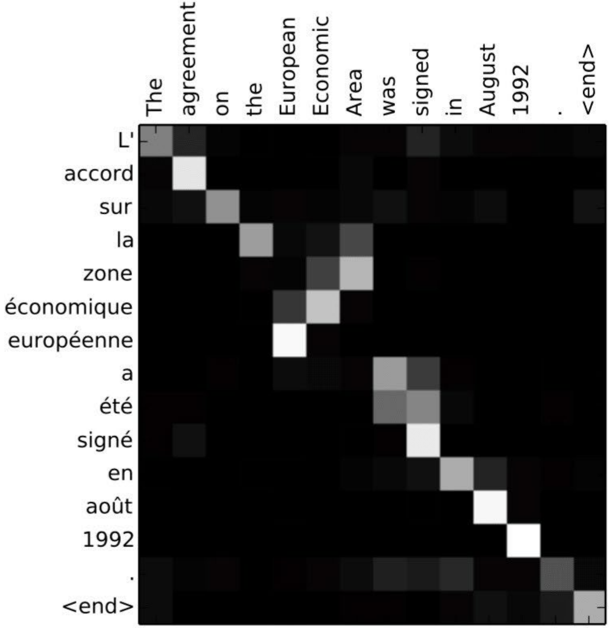
$d_{\boldsymbol{k}}$: the dimension of keys and queries

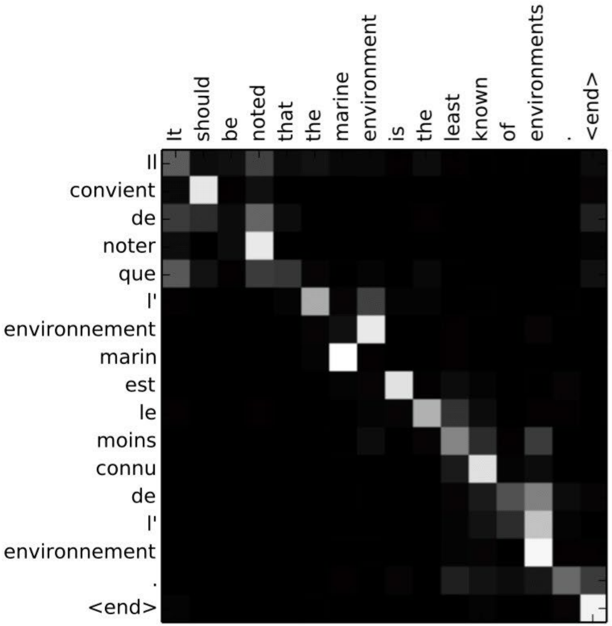
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$



Attention learns (nearly) alignment

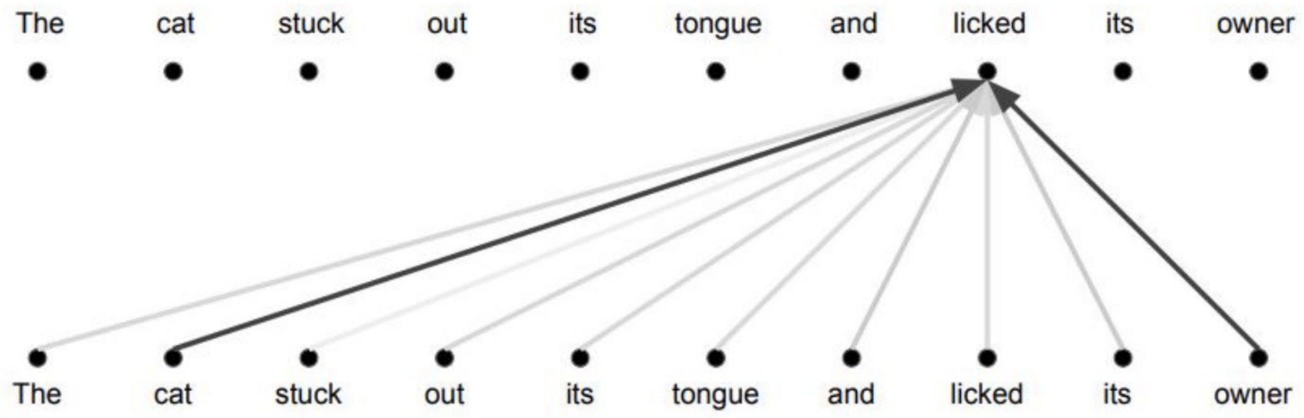
- An example from the paper ‘Neural Machine Translation by Jointly Learning to Align and Translate’





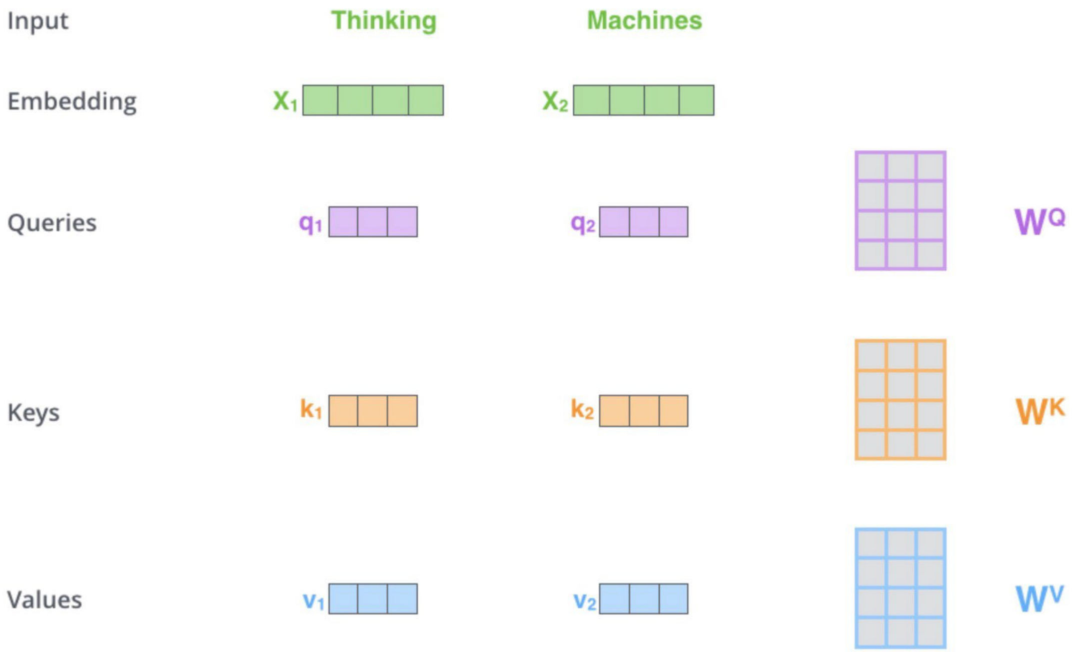
Attention is all you need: self-attention (1)

- For an input sequence of words, play attention mechanisms between every word and others (including itself).
- Features
  - Constant path length between any two positions
  - Easy to parallelize per layer



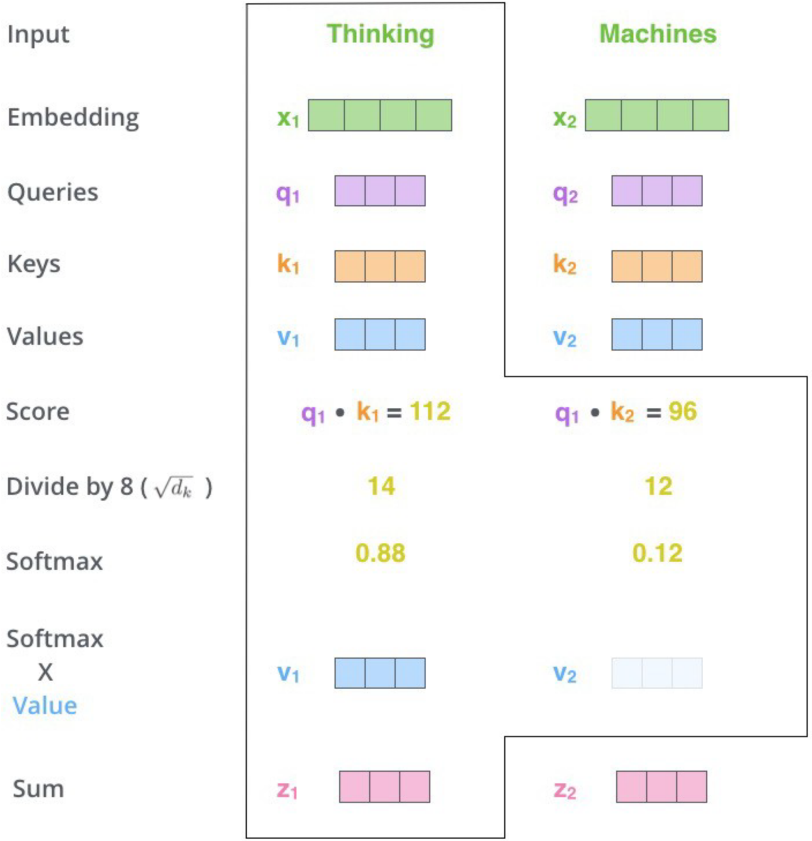
Attention is all you need: self-attention (2)

- How to construct queries, keys, and values?
  - Linear transformation from original input embeddings x_i: $q_i = x_i W^Q$, $k_i = x_i W^K$, $v_i = x_i W^V$



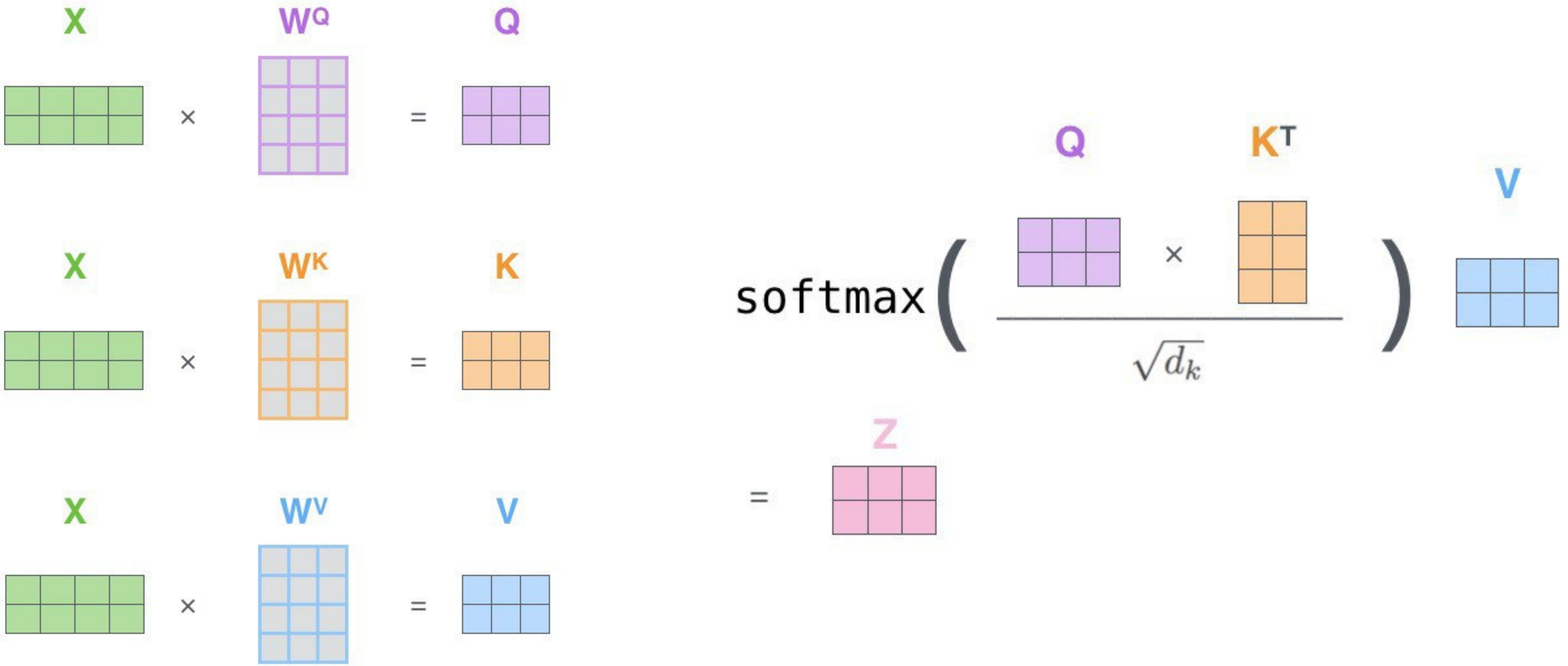
Attention is all you need: self-attention (3)

- Perform scaled dot-product attention between a word *i* and all words
- Obtain *L* self-attention vectors, one for each word (token).



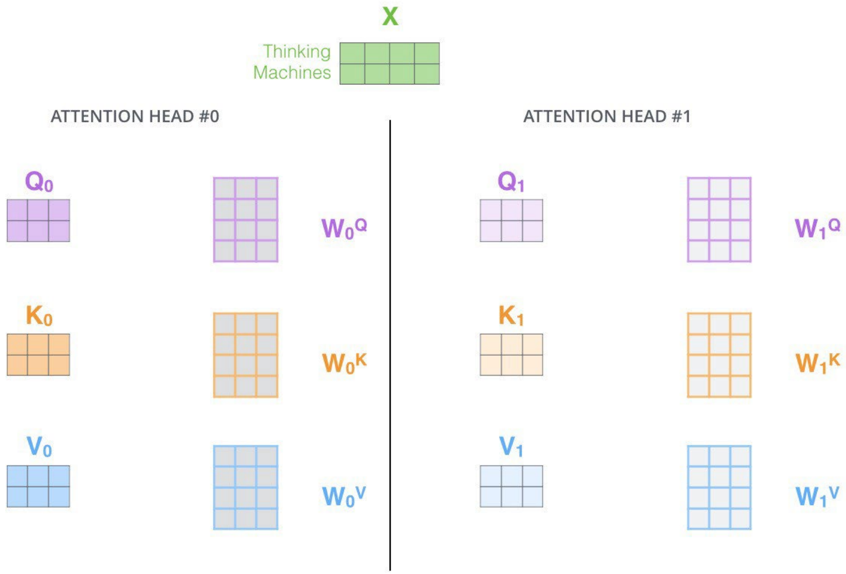
Attention is all you need: self-attention (4)

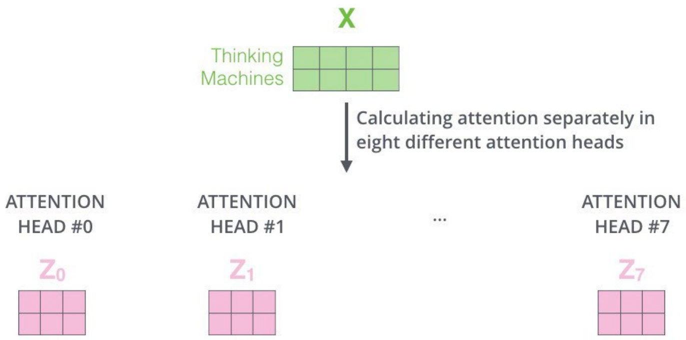
- We change these calculations into the matrix form.



Attention is all you need: multi-head self-attention (1)

- What if we choose different forms of $W^{\mathrm{Q}}, W^{\mathrm{K}}$, and W^{V}?
  - Each group of $(W^{\mathrm{Q}}, W^{\mathrm{K}}, W^{\mathrm{V}})$ is called a head.
  - Increase the representation performance: different representation subspaces, model's ability to focus on different positions.

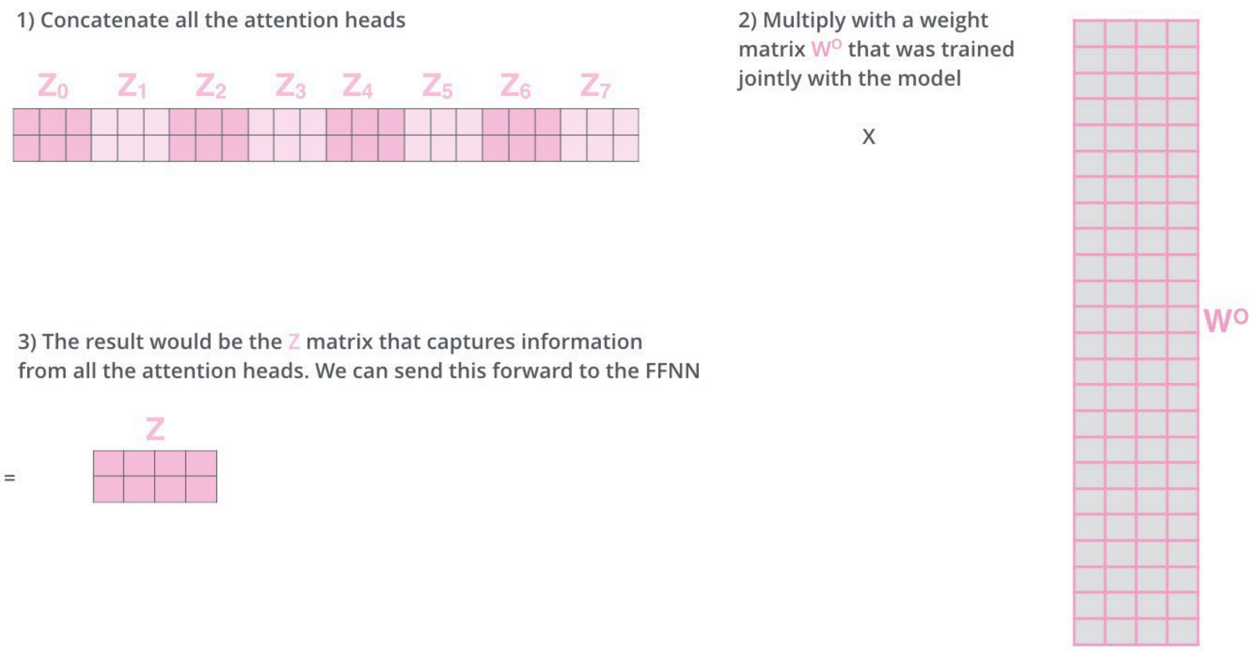




8 head self-attention gives 8 groups of attention vectors

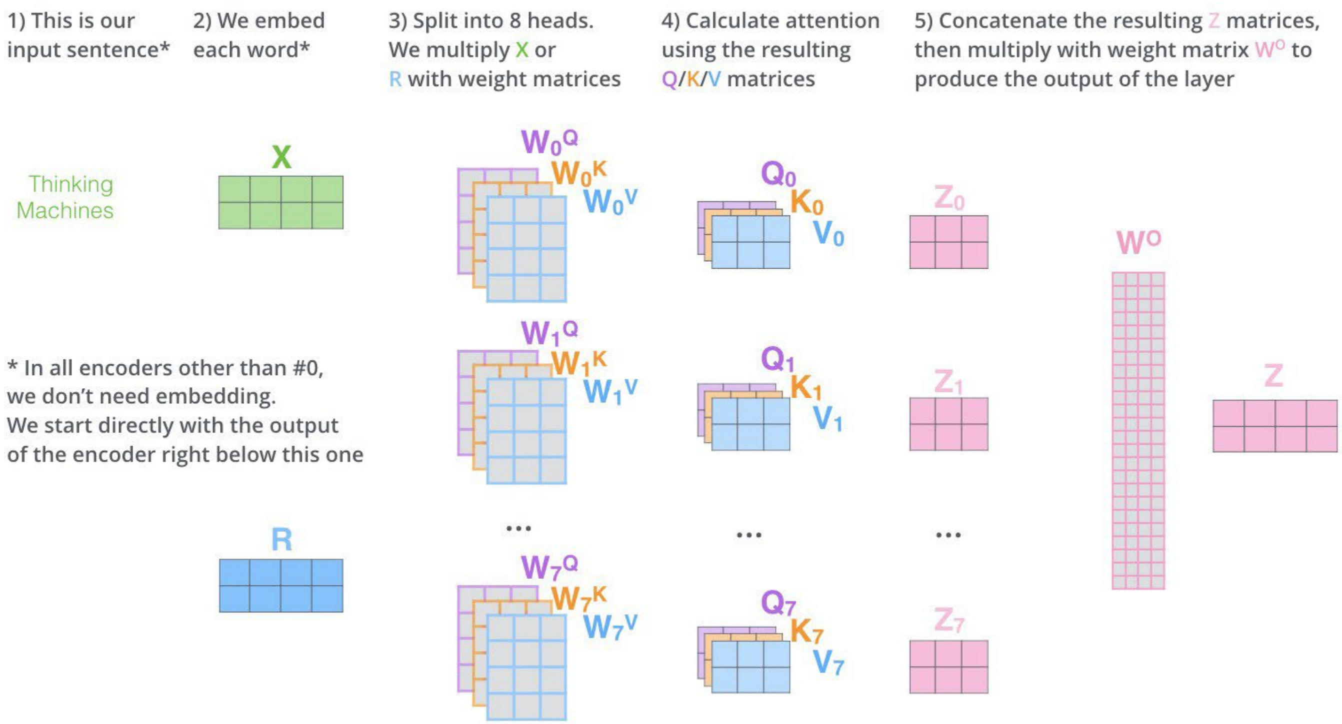
Attention is all you need: multi-head self-attention (2)

- How to combine multi-headed output together?
  - Concatenate them and use another linear transformation.



Attention is all you need: multi-head self-attention (3)

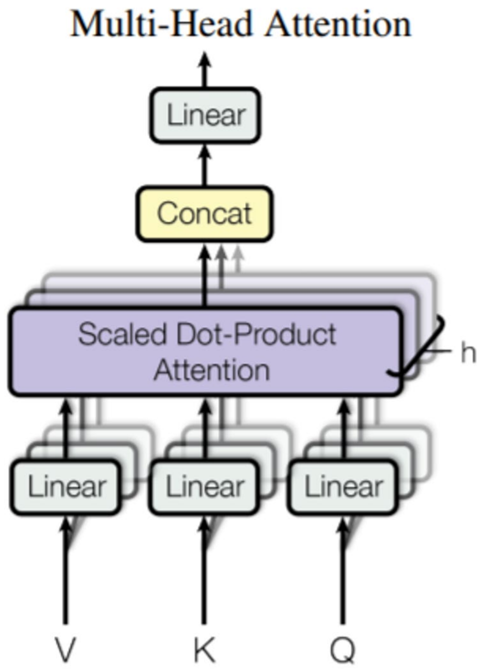
- In a nutshell



Attention is all you need: multi-head self-attention (4)

- Graphical view and mathematical presentations
  - Linearly project the queries, keys and values h times with different, learned linear projections to d_{k}, d_{k}, d_{v} dimensions respectively.
  - On each of these projected versions, perform the attention function in parallel.
  - The resulting d_{v} - dimensional output values are concatenated and once again projected.
- Parameters: hidden size d_{model}, self-attention heads h

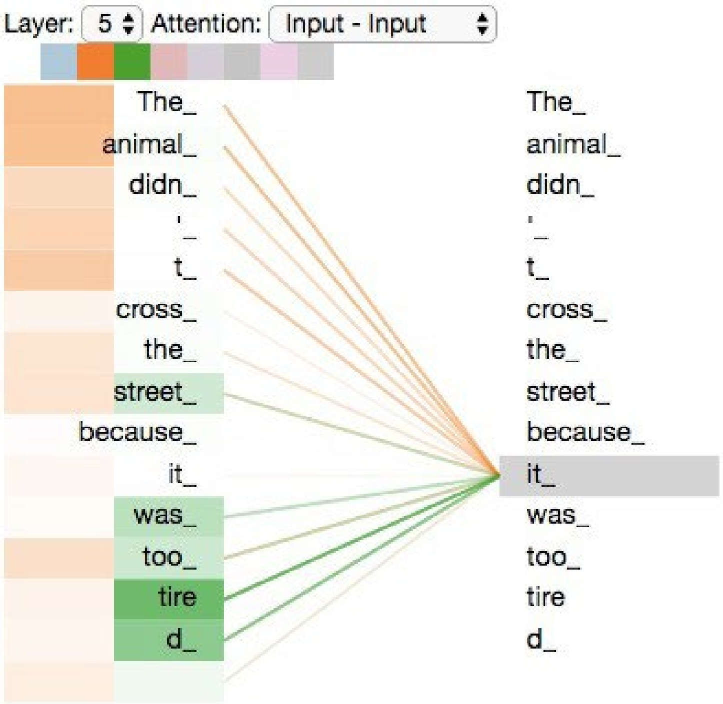
$$\mathrm{MultiHead}(Q, K, V) = \mathrm{Concat}(\mathrm{head_1}, ..., \mathrm{head_h})W^O$$
$$\text{where } \mathrm{head_i} = \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

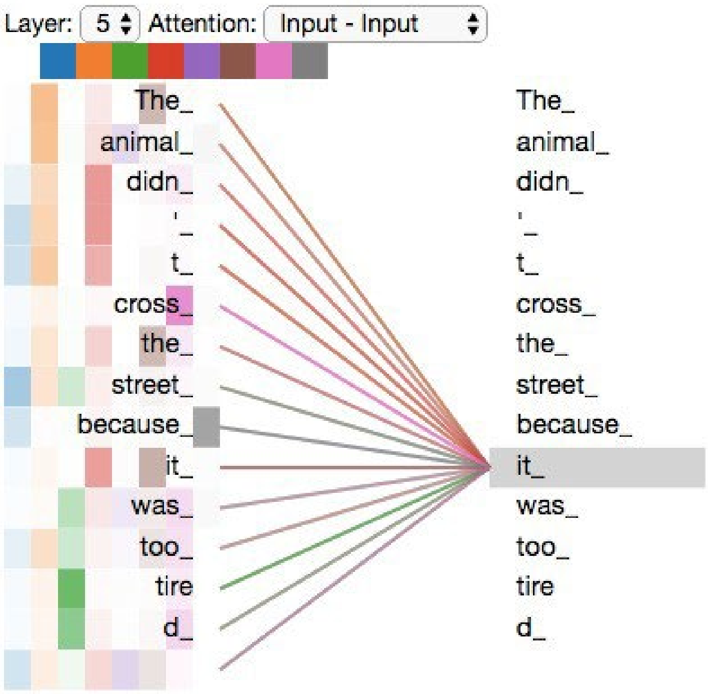


Where the projections are parameter matrices $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$.

Multi-head attention results illustration

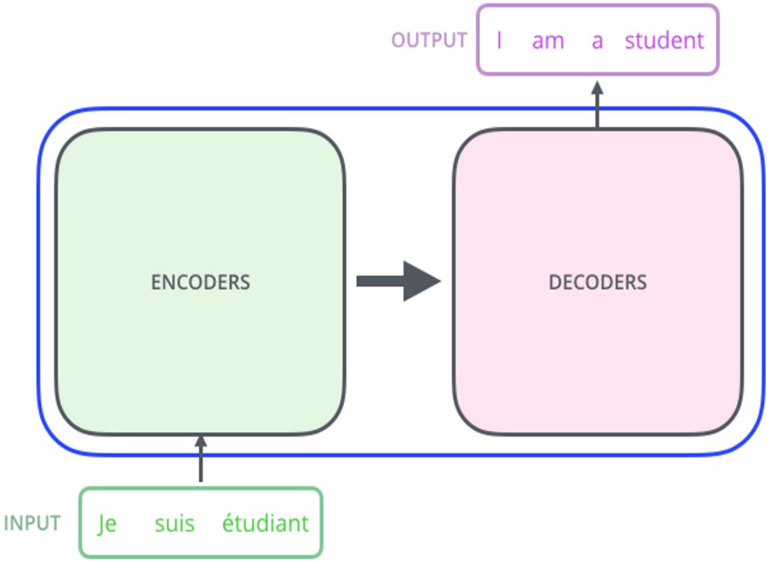
- One head attention for ‘it’ -> more on ‘the animal’;
- Multi-head attention for ‘it’ -> ‘street’, ‘tire’, ‘didn’t’, ‘cross’, ‘because’.

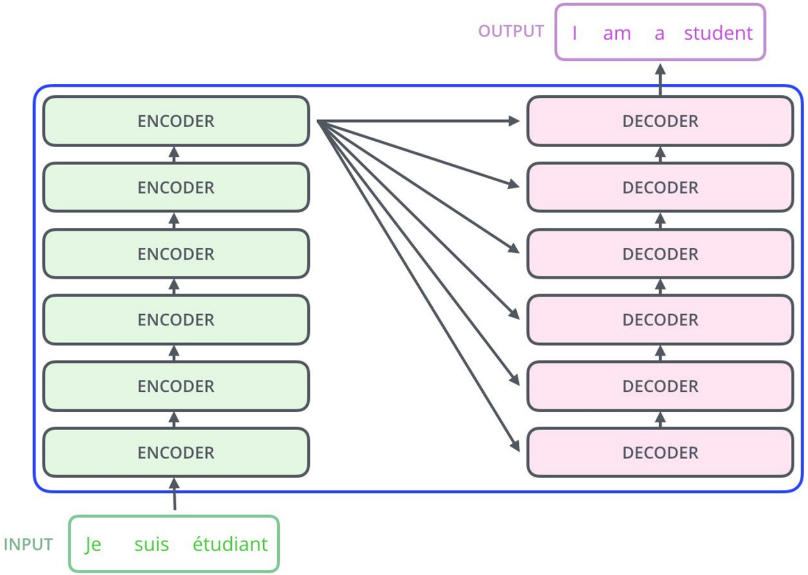




Transformer

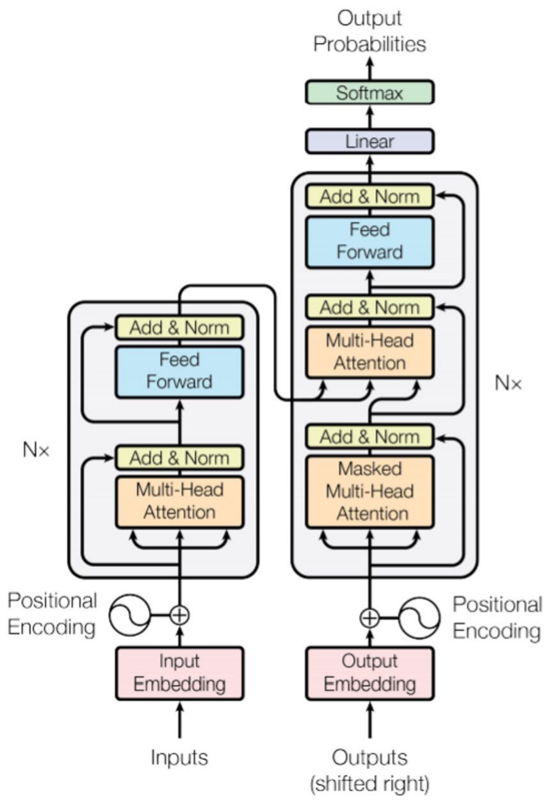
- Proposed by Google: Attention is All You Need (Vaswani et al., 2017)
- Main technique: multi-head self attention mechanism.
- The transformer is a novel architecture that aims to solve sequence-to-sequence tasks while handling long-range dependencies with ease.



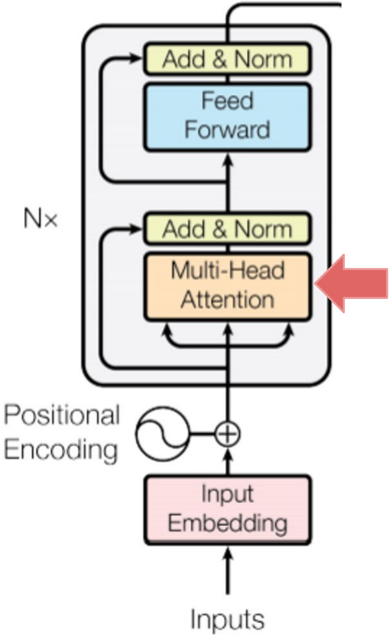


Transformer - formal description

- Encoder: maps an input sequence of symbol representations $(x_1, x_2, \ldots, x_\mathrm{n})$ to a sequence of continuous representations $z = (z_1, z_2, \ldots, z_\mathrm{n})$
- Decoder: given z, generates an output sequence $(y_1, y_2, \ldots, y_\mathrm{m})$, one element at a time.
- At each step the model is auto-regressive, consuming the previously generated symbols as additional input.
- Stacked self-attention and point-wise, fully connected layers for both the encoder and decoder.



Transformer: attention in encoder



Encoder

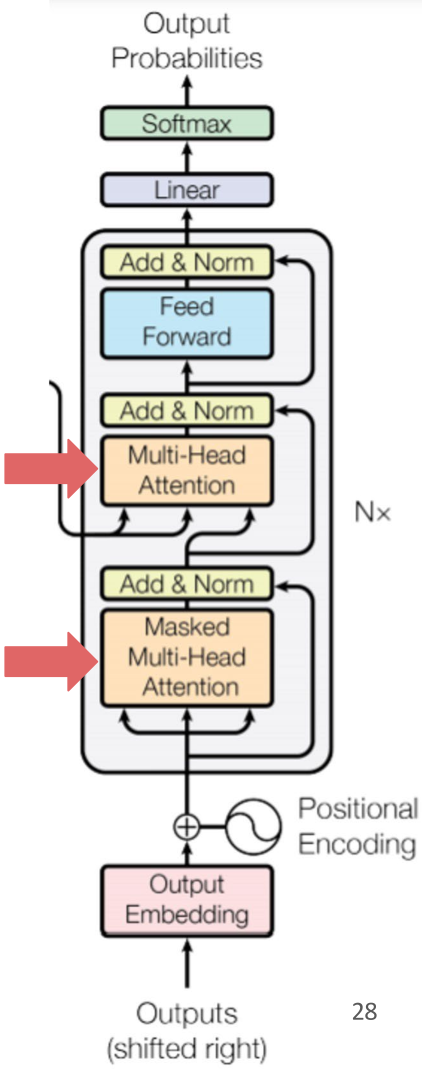
- In a self-attention layer all of the keys, values and queries come from the output of the previous layer or the raw input embedding (with hidden dimension d_{model}) in the encoder.

- Each position in the encoder can attend to all positions in the previous layer of the encoder.

Transformer: attention in decoder

Decoder

- Similarly, each position attends to all positions in the decoder up to and including that position.
- Encoder-decoder attention: keys and values come from the top encoder output; queries come from the output of the masked multi-head attention (with hidden dimension d_{model}) in the decoder.
- Masked self-attention
  - Keys, values and queries come from the output of the previous layer or the raw output embedding (with hidden dimension d_{model}) in the decoder.
  - Prevent leftward information flow to preserve the auto-regressive property. Decoder self-attention is only allowed to attend to earlier positions in the output sequence, implemented by masking out (setting to $-\infty$) future positions (tokens).

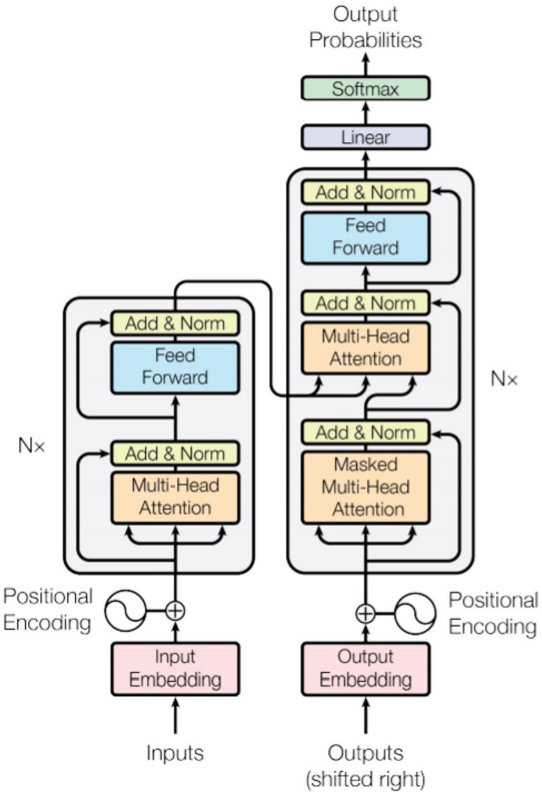


Transformer: other details

- Each of the layers in the encoder and decoder contains a fully connected feed-forward network (FFN), which is applied to each position separately and identically.

$$FFN(\mathrm{x}) = \max(0, xW_1 + b_1)W_2 + b_2$$

- Positional encodings: input positional information of the sequence.
- Residual connection: learning the changing part
- Learned embeddings are used to convert the input tokens and output tokens to vectors of dimension d_{model}. The same weight matrix are shared in the two embedding layers.



Positional encoding in transformer (1)

- Transformer does not contain recurrence or convolution, it does not know the order of input tokens.
- We have to let the model know the positions of the tokens explicitly
- Idea: input representation of a token is the sum of two embeddings: token and positional

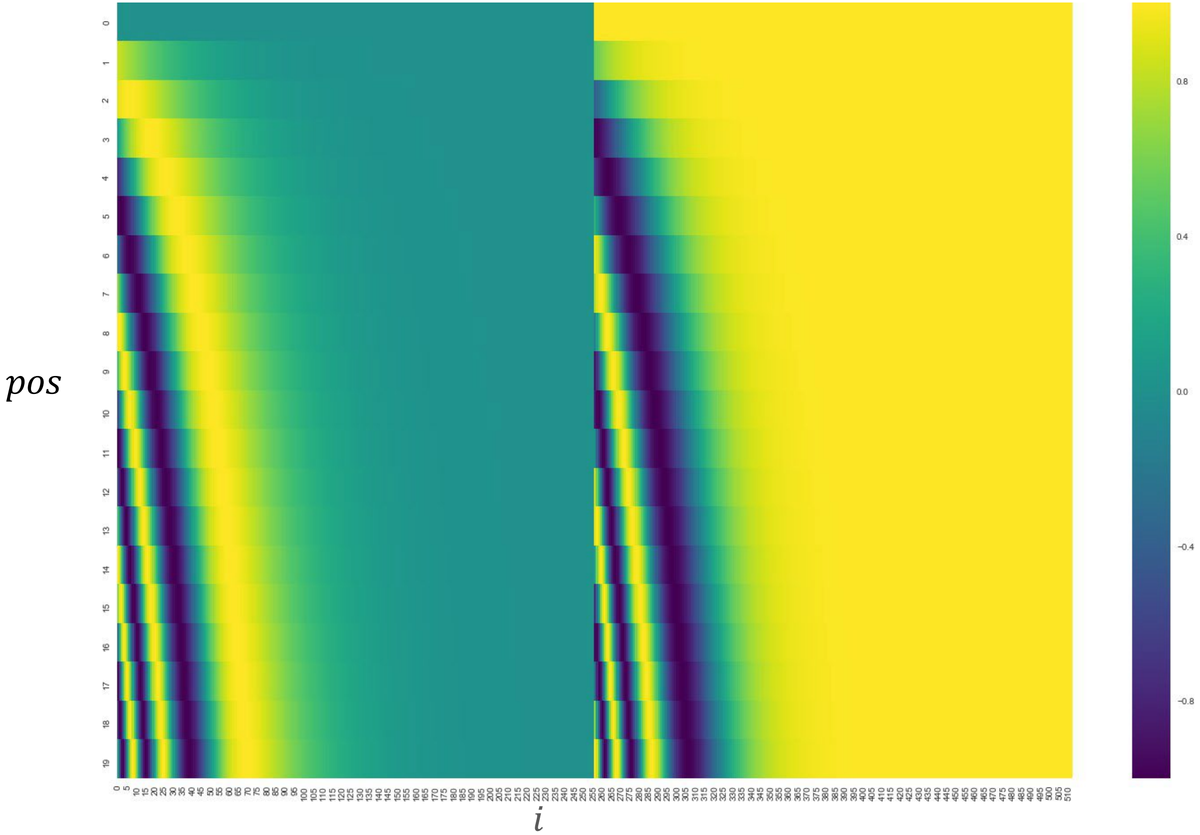
Positional encoding in transformer (2)

- In order to add position information (order of the sequence)

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

- pos is position (e.g., the pos-th token in a sentence) and i is the i-th element in a vector to represent each position.
- Each dimension of the positional encoding corresponds to a sinusoid.
- For any fixed offset k, PE_{pos+k} can be represented as a linear transformation of PE_{pos}. This would allow the model to easily learn to attend by relative positions.

Positional encoding in transformer (3)



BERT: Bidirectional Encoder Representations from Transformers

- BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, Google, 2018
- Pretrained deep bidirectional representation models from unlabeled text .
- Jointly conditioning on both left and right context in all layers.
- Can be finetuned with just one additional output layer to create models for a wide range of tasks.
- Powerful in a variety of tasks.

Task	Score	Best with BERT (from Delvin et al.)	Absolute improvement
NLU	GLUE score	80.5%	7.7%
NLI	MultiNLI accuracy	86.7%	4.6%
question-answer	SQuAD v1.1 Test F1	93.2	1.5
question-answer	SQuAD v2.0 Test F1	83.1	5.1

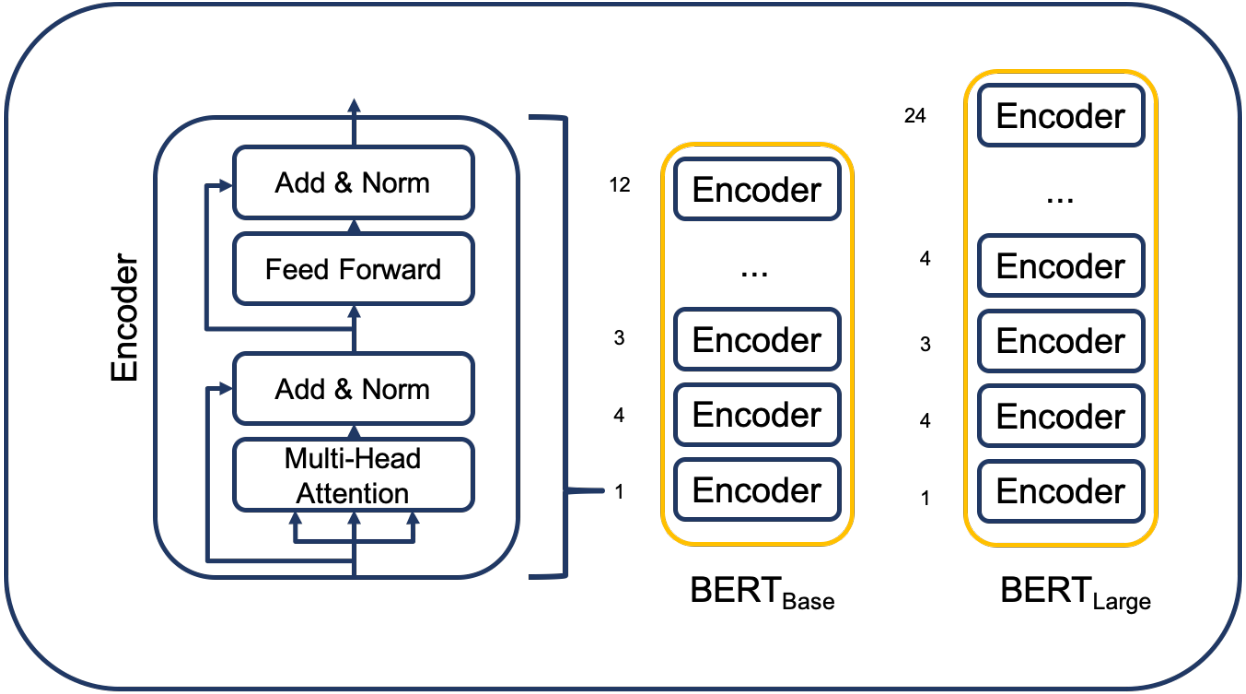
BERT architecture (1)

- BERT architecture consist of multi-layer bidirectional transformer encoders
- BERT model provided in the paper came in two sizes

BERT_BASE	BERT_LARGE
Layers = 12	Layers = 24
Hidden size = 768	Hidden size = 1024
Self-attention heads = 12	Self-attention heads = 16
Total parameters = 110M	Total parameters = 340M

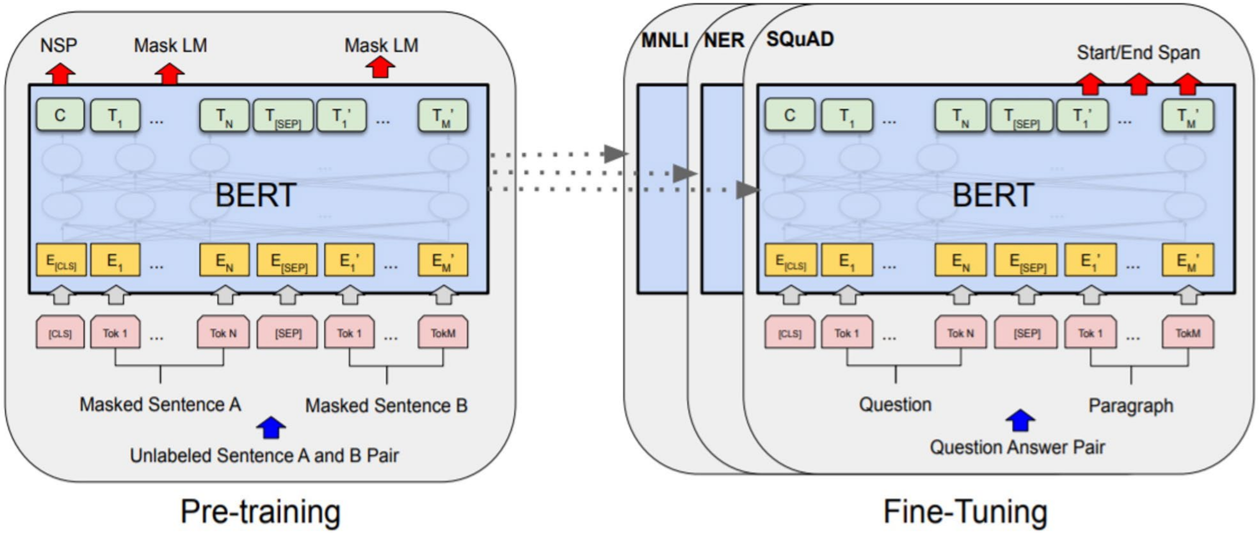
BERT architecture (2)

- A graphical view



BERT: pretraining + finetuning

- Apart from output layers, the same architectures are used in both pre-training and fine-tuning.
- The same pre-trained model parameters are used to initialize models for different down-stream tasks.
- During fine-tuning, all parameters are fine-tuned.

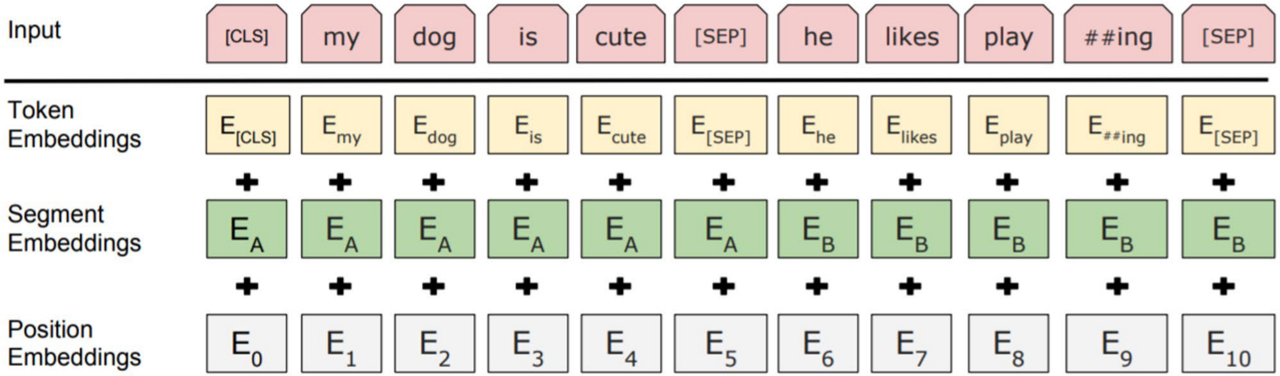


What contributes to BERT's success?

- Innovation in model pre-training: 2 self-supervised tasks for pre- training instead of traditional unidirectional language models
  - Masked Language Model (MLM)
  - Next Sentence Prediction (NSP)
  - The training loss is the sum of the mean MLM likelihood and the mean NSP likelihood
- Compatibility to various tasks
  - Just fine-tune BERT Model for specific tasks to achieve state-of-the-art performance
  - BERT advances the state-of-the-art for eleven NLP tasks

BERT: input/output representation

- For a given token, its input representation is constructed by summing the corresponding token, segment, and position embeddings.
- [CLS] is a special symbol added in front of every input example.
- [SEP] is a special separator token (e.g. separating questions/answers).
- Unambiguously represent both a single sentence and a pair of sentences in one token sequence.



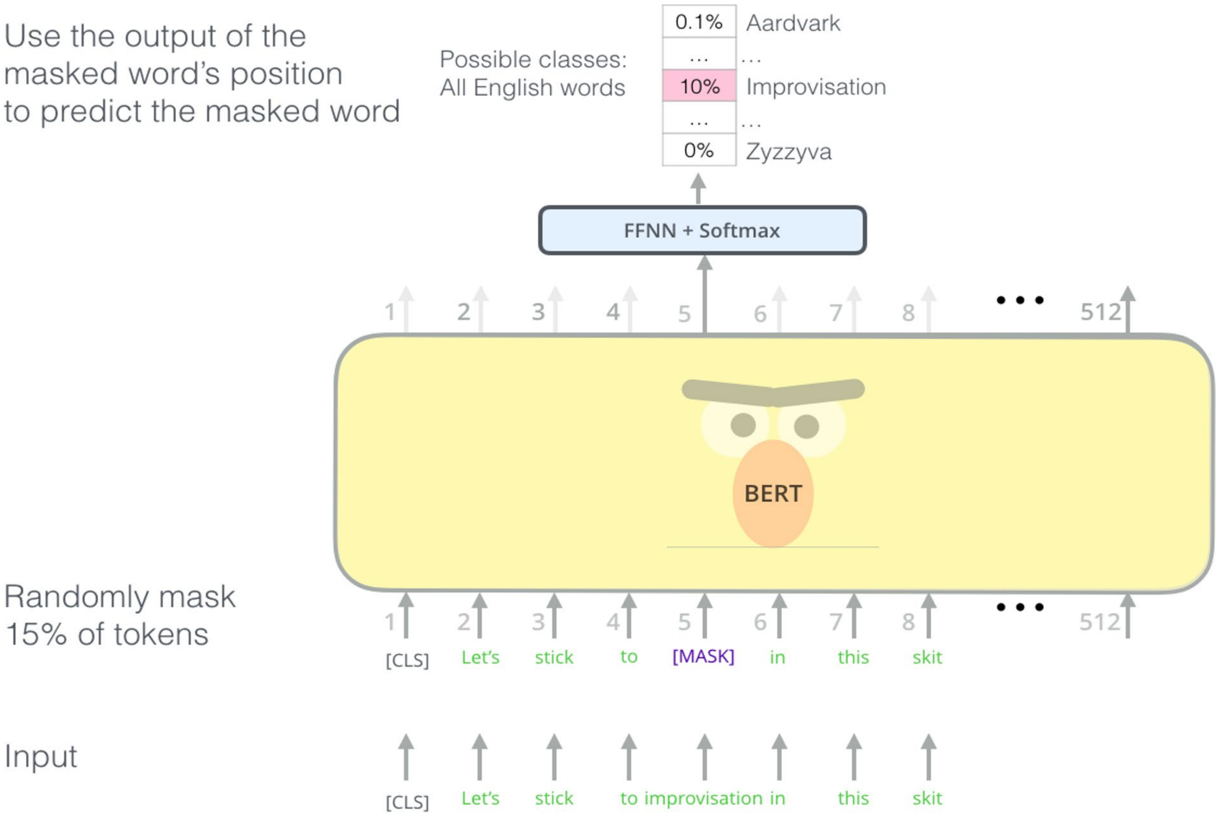
Pre-training BERT

- In order to pre-train BERT transformers bidirectionally, two self-supervised tasks are proposed, namely Masked LM and Next Sentence Prediction (NSP).
- One of the reasons of BERT's success is the amount of data that it got trained on.
- The two corpuses that were used to pretain the language models are:
  - The BooksCorpus (800M words)
  - English Wikipedia (2,500M words)

Masked language modeling (MLM)

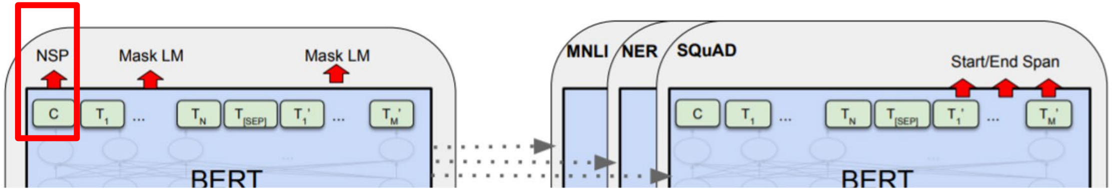
- Motivation: Enable bidirectional pre-training
- Main idea: Mask some percentage of the input tokens at random, and then predict those masked tokens.
- Details:
  - The training data generator chooses 15% of the token positions at random for prediction.
  - If the *i*-th token is chosen, we replace the i-th token with
    (1) the [MASK] token 80% of the time,
    (2) a random token 10% of the time,
    (3) the unchanged *i*-th token 10% of the time.
  - Predict the original token via cross entropy and the final hidden vector for the i-th input token
- The masking scheme here is to reconcile the mismatch between pre-training and fine-tuning, since the [MASK] token does not appear during fine-tuning.

An illustration of MLM

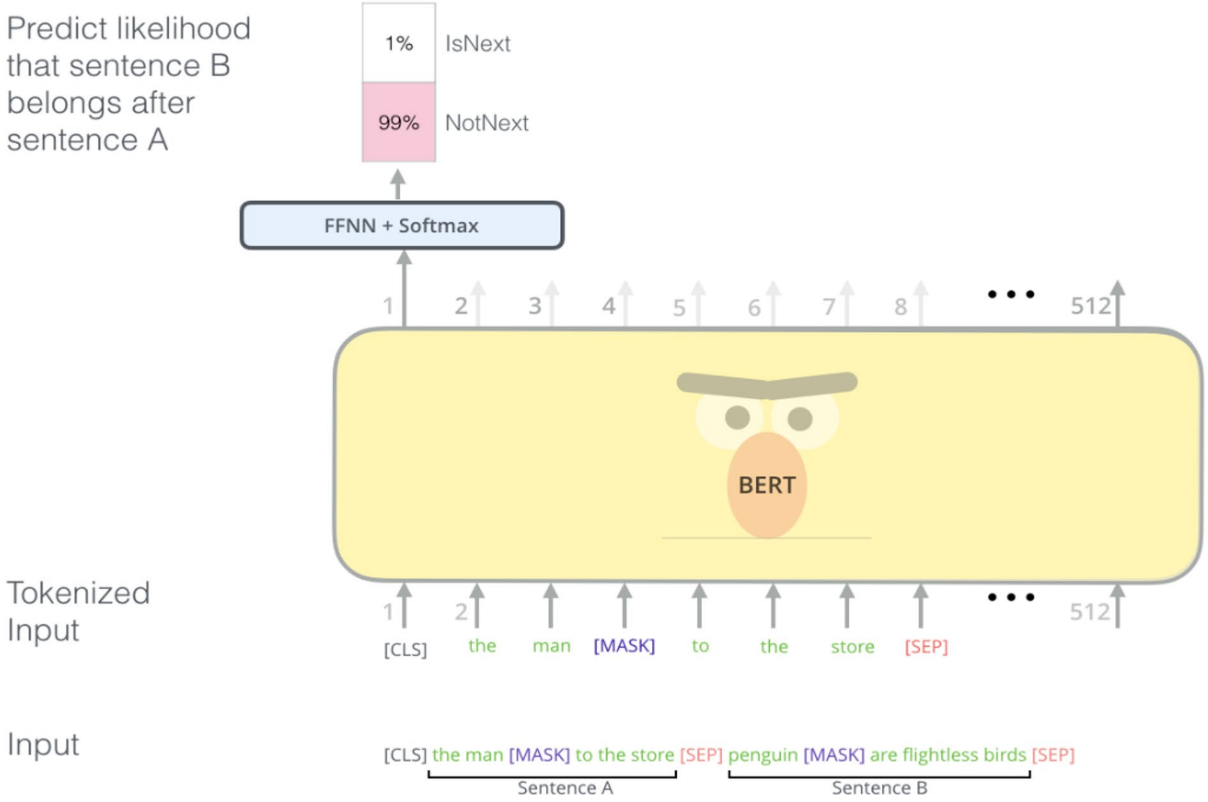


Next sentence prediction (NSP)

- Motivation: understanding the relationship between two sentences (inspired by Question Answering and Natural Language Inference).
- Main idea: given a preceeding sentence A and a sentence B, judge if B is the next sentence of A.
- Details:
  - choose the sentences A and B for each pretraining 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])
- In the architecture figure, C is used for NSP prediction.



An illustration of NSP



NSP example

Input = [CLS] the man went to [MASK] store [SEP]
he bought a gallon [MASK] milk [SEP]

Label = **IsNext**

Input = [CLS] the man [mask] to the store [SEP]
penguin [MASK] are flight ##less brids [SEP]

Label = **NotNext**

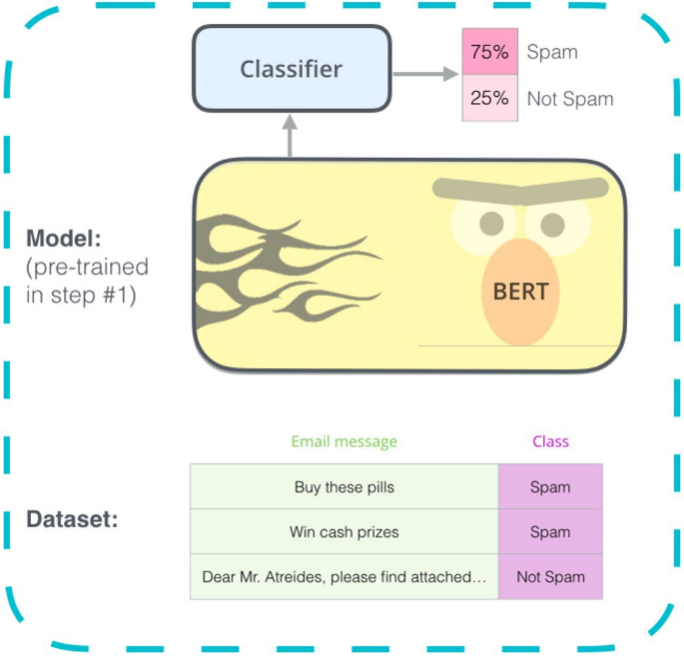
BERT: fine-tuning (1)

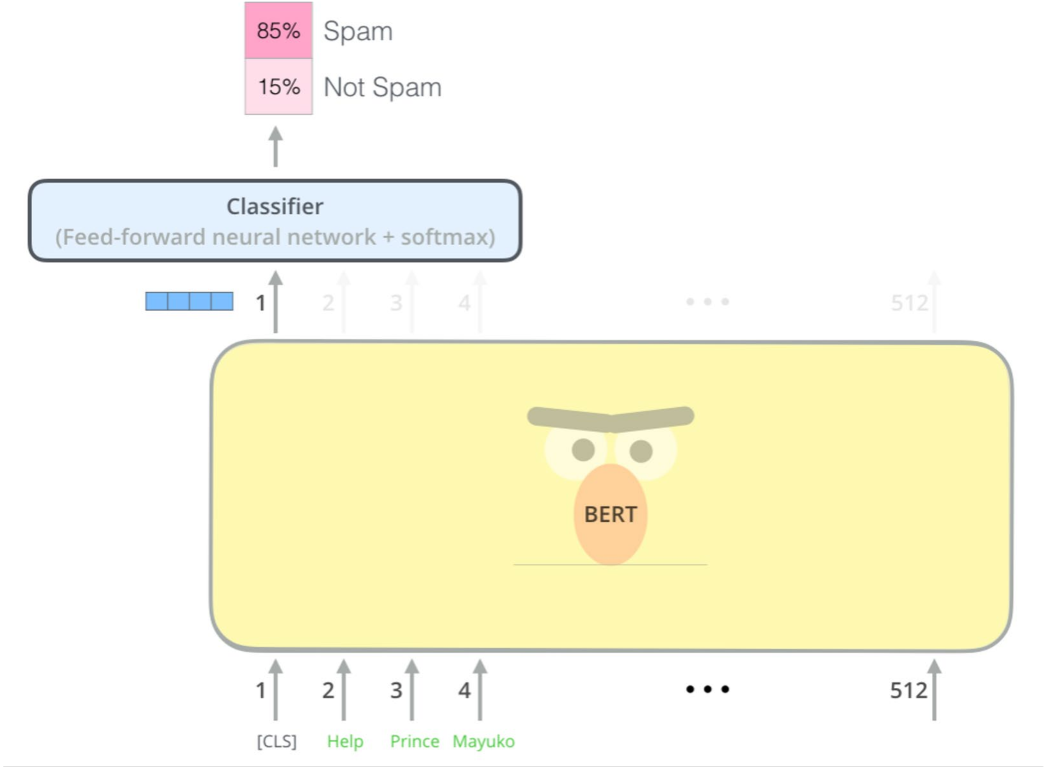
- Common approach: independently encode text pairs before applying attention.
- BERT's approach: concatenate a text pair, and then encode with self-attention in order to include bidirectional cross attention between two sentences.
- For each task, simply plug in the task-specific inputs and outputs into BERT and fine- tune all the parameters end-to-end.
- Compared to pre-training, fine-tuning is relatively inexpensive.

BERT: fine-tuning (2)

- For classification tasks (e.g. sentiment analysis) we add a classification FFN for the [CLS] token input representation on top of the final output
- For Question Answering alike tasks BERT train two extra vectors that are responsible for marking the beginning and the end of the answer

BERT fine-tuning for classification



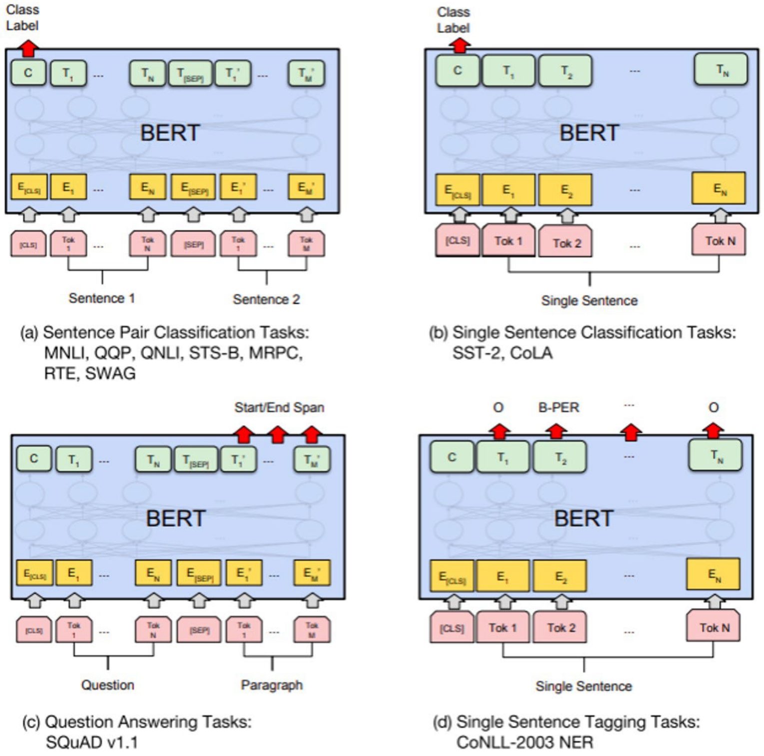


BERT fine-tuning for different tasks

- Context vector C: Take the final hidden state corresponding to the first token in the input: [CLS].
- Transform to a probability distribution of the class labels:

$$P = \text{softmax}(CW^T)$$

- **Batch size**: 16, 32
- **Learning rate (Adam)**: 5e-5, 3e-5, 2e-5
- **Number of epochs**: 3, 4



Experiments on BERT

BERT fine-tuning results on 11 NLP tasks

Task	Score	Best with BERT (from Delvin et al.)	Absolute improvement
NLU	GLUE score	80.5%	7.7%
NLI	MultiNLI accuracy	86.7%	4.6%
question-answer	SQuAD v1.1 Test F1	93.2	1.5
question-answer	SQuAD v2.0 Test F1	83.1	5.1

Generative Pre-training (GPT)

- Motivation: semi-supervised learning
  - Stage 1: Unsupervised learning on word-level or phrase-level
    - E.g. word embeddings
  - Stage 2: Supervised training using these word-level features
- In the method above:
  - Stage 1 is less dependent on the task - related more to NLU
  - Stage 2 models may vary a lot according to different tasks
- Are there any techniques that improve the preformance of Stage 1 models that would work for most of the tasks?
  - That's the problem we hope to solve by introducing pre-training.
  - In other words, pre-training improves language understanding.

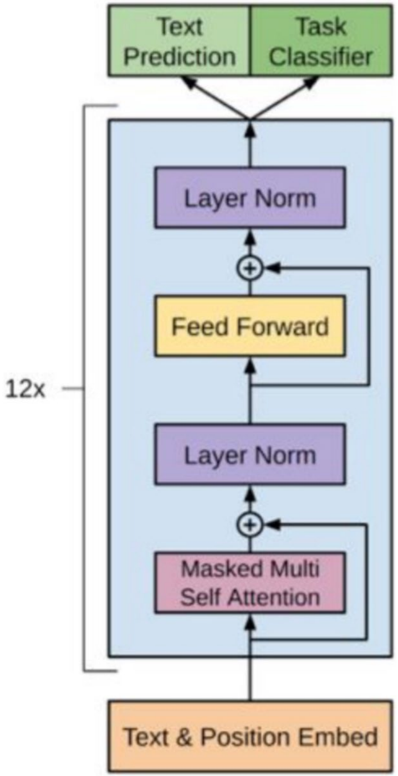
GPT framework (1)

- Multi-layer transformer decoder
  - first layer: $h_0 = UW_e + W_p$
  - the l-th layer: $h_l = transformer\_block(h_{l-1}), \forall l \in [1, n]$
  - Self-supervised pre-training, similar to embdeddings such as word2Vec
  - Given tokens *U*, maximize the contextual conditional probability

$$L_1(\mathcal{U}) = \sum_i \log P(u_i | u_{i-k}, \ldots, u_{i-1}; \Theta)$$

$$P(u) = \texttt{softmax}(h_n W_e^T)$$

    where *k* is the window size; $h_n W_e^T$ is the score for each word.
  - Unidirectional!



GPT framework (2)

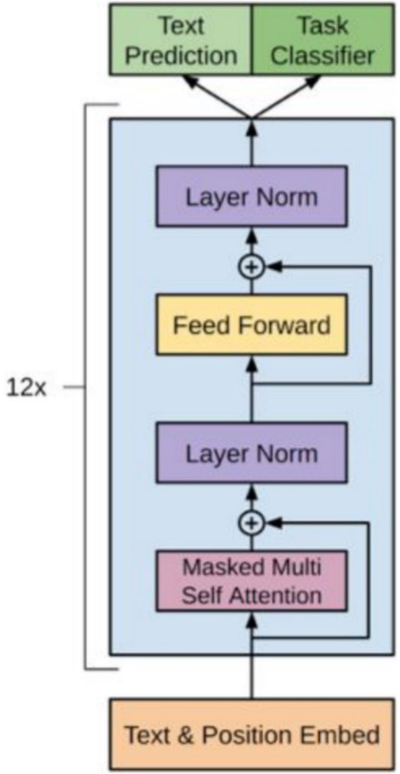
- Supervised fine-tuning
  - keep the pre-trained transformers
  - replace the final linear layer W_e with W_y,
  - Given data inputs X and labels y, maximize

$$P(y|x^1, \ldots, x^m) = \texttt{softmax}(h_l^m W_y).$$

$$L_2(\mathcal{C}) = \sum_{(x,y)} \log P(y|x^1, \ldots, x^m).$$

$$L_3(\mathcal{C}) = L_2(\mathcal{C}) + \lambda * \boxed{L_1(\mathcal{C})}$$

auxiliary training objective



Task-specific adaptions (1)

How to adapt a single architecture to multiple input formats?

Classification (e.g. sentiment analysis)

- Given a piece of text, is it positive or negative?
- Answers: "Yes", "No"
- Answers: "Very positive", "Positive", "Neutral", "Negative", "Very negative"

Entailment

- Given a premise p and a hypothesis h, does p imply h?
- Answers: "entailment", "contradiction", or "neutral"

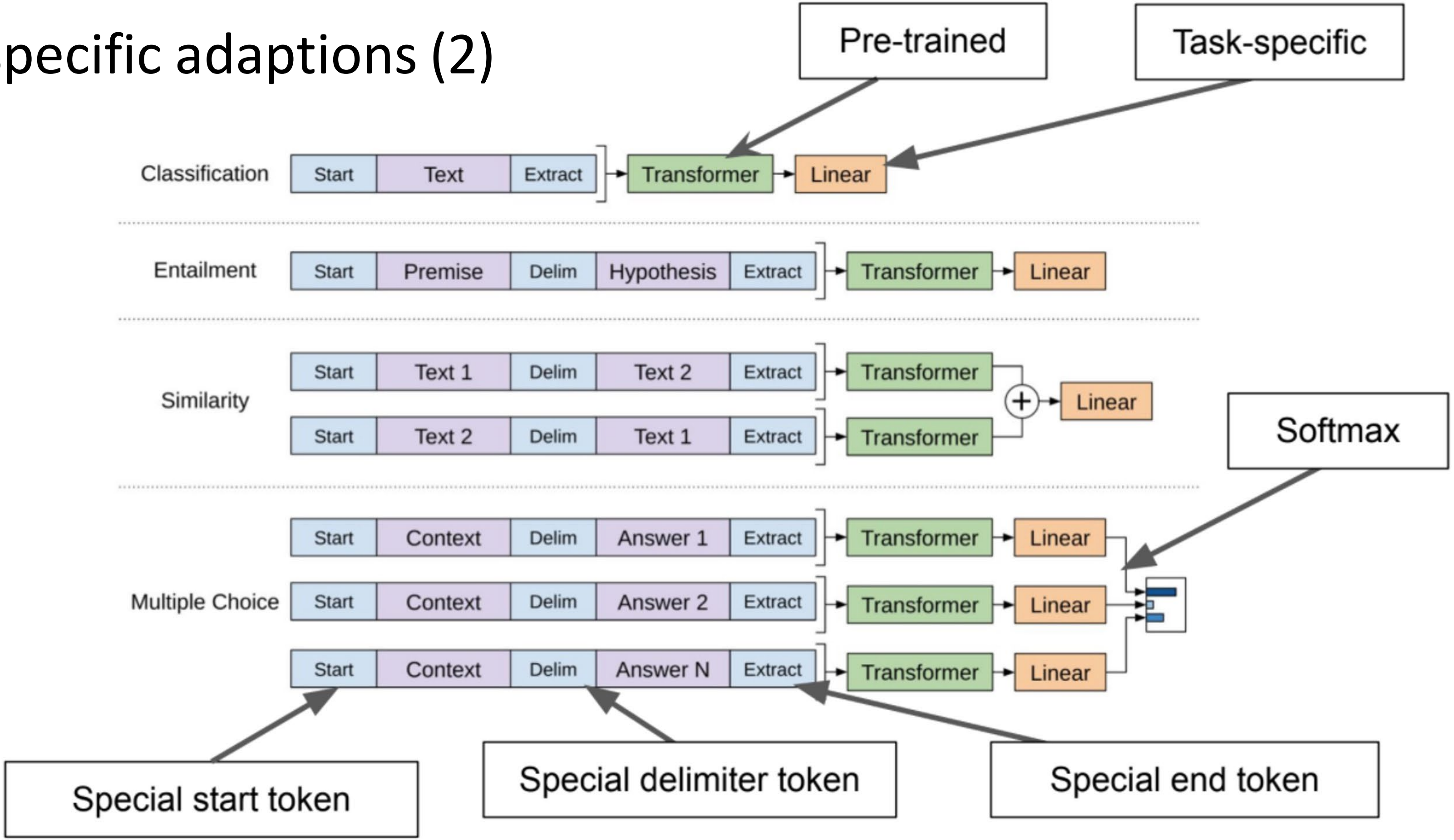
Similarity

- Are two sentences semantically equivalent?
- Answers: "Yes", "No"

Multiple Choice (e.g. Story Cloze)

- Given a short story and two sentences, which is the sentence that ends the story?
- Given a passage and a question, and some multiple-choice answers, which is the answer?
- Answers: $A_1, A_2, \ldots\ A_{\mathrm{N}}$

Task-specific adaptions (2)



Pre-trained model comparison: ELMo, GPT, and BERT

	Architecture	Pre-training	Downstream tasks
ELMo	Bi-directional LSTM language model	Unsupervised corpus. Learn both words and linguistic context features that support downstream tasks.	Feature-based tasks and task-specific models.
GPT	Uni-directional transformer decoder	Unsupervised corpus. Each specific task requires discriminative fine-tuning.	Model-based tasks and task-agnostic models.
BERT	Bi-directional transformer encoder	Unsupervised corpus. 2 unsupervised tasks: MLM and NSP. Each specific task requires discriminative fine-tuning.	Model-based tasks and task-agnostic models.

References

Attention and transformer:

- Attention is All You Need (Vaswani et al., 2017) https://arxiv.org/pdf/1706.03762.pdf
- https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html
- Neural Machine Translation by Jointly Learning to Align and Translate (Bahdanau, 2015) https://arxiv.org/pdf/1409.0473.pdf
- Transformer: http://jalammar.github.io/illustrated-transformer/

GPT:

- Improving Language Understanding by Generative Pre-Training (Radford et al., 2018) https://www.cs.ubc.ca/~amuham01/LING530/papers/radford2018improving.pdf

BERT:

- BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding (Delvin et al., 2019) https://arxiv.org/pdf/1810.04805.pdf
- https://towardsdatascience.com/understanding-entity-embeddings-and-its-application-69e37ae1501d
- https://www.slideshare.net/AbdallahBashir3/bert-176297542

BERT fine-tune code:

- https://skimai.com/fine-tuning-bert-for-sentiment-analysis/

Comparison of pre-trained models:

- https://medium.com/@gauravghati/comparison-between-bert-gpt-2-and-elmo-9ad140cd1cda

Thanks for your attention!