Deep learning

Transformers family

Hamid Beigy

Sharif University of Technology

November 30, 2024

Table of contents

Introduction

Introduction

1. Attention model can learn to make predictions by selectively attending to a given set of data as

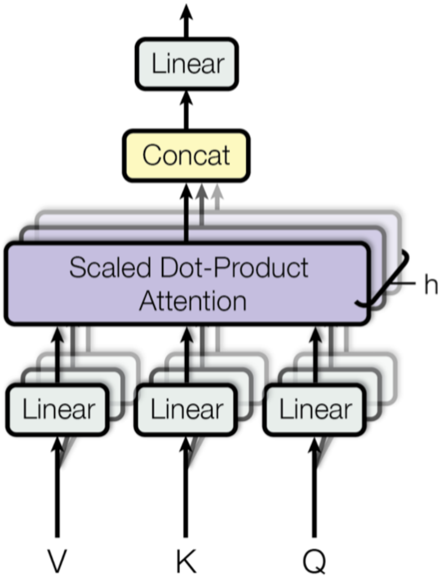
$$\mathit{Attention}(\mathbf{Q}, \mathbf{K}_r, \mathbf{V}) = \mathbf{V}\ \mathit{Softmax}\left(\frac{\mathbf{Q}^\top \mathbf{K}}{\sqrt{p}}\right) = \mathbf{V}\ \frac{\mathbf{Q}^\top \mathbf{K}}{\sqrt{p}\sum_{r\in S} \mathbf{Q}^\top \mathbf{K}_r}$$

   where S is input sentence.

2. **Self-attention** is a type of attention that the model makes prediction for one part of the input using its other parts.
3. **Homework: Is self-attention permutation-invariant?**
4. **Homework: Is self-attention a set operation?**

Multi-Head Self-Attention

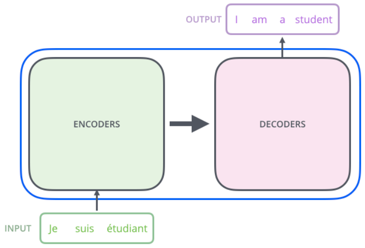
1. Multi-head self-attention splits the inputs into smaller chunks and then computes the scaled dot-product attention over each subspace in parallel.
2. The independent attention outputs are simply concatenated and linearly transformed into expected dimensions.



Transformers model

Transformers model

1. The attention make it possible to do sequence to sequence modeling without recurrent network units (Vaswani et al. 2017).
2. The transformer model is entirely built on the self-attention mechanisms without using sequence-aligned recurrent architecture.



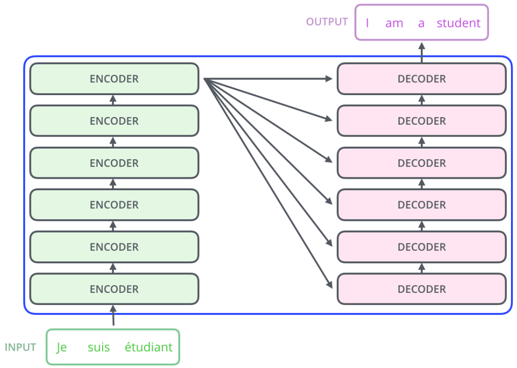
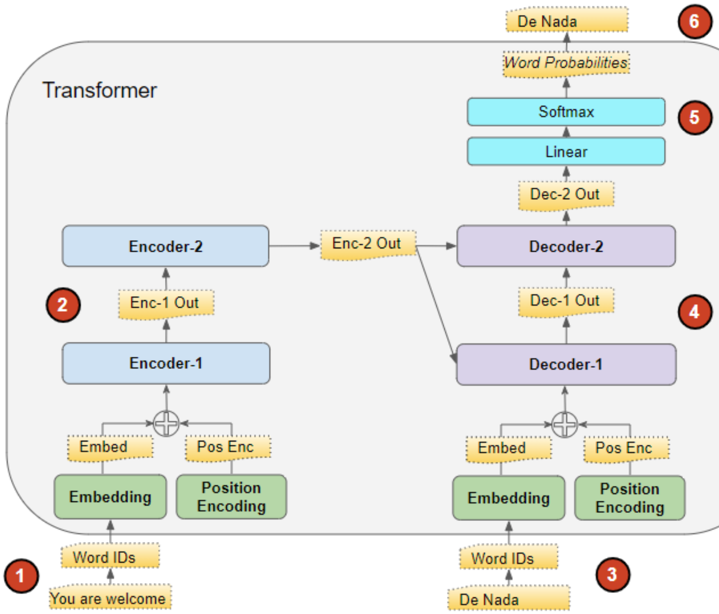


Figure: Jay Alammar

3. The encoding component is a stack of six encoders.
4. The decoding component is a stack of decoders of the same number.

Transformers training

1. The Transformers works slightly differently during training and inference.
2. Input sequence: You are welcome in English.
3. Target sequence: De nada in Spanish



Transformers inference

1. During Inference, we have only the input sequence and don't have the target sequence to pass as input to the Decoder.
2. The goal is to produce the target sequence from the input sequence alone.

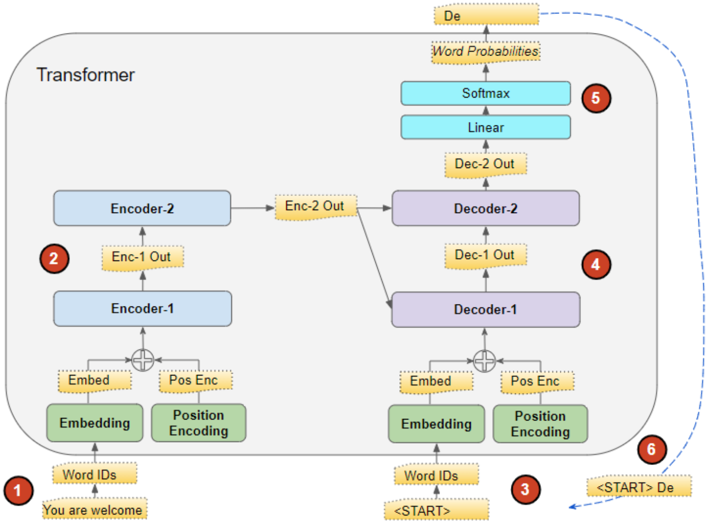
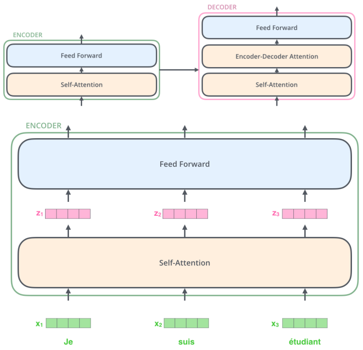


Figure:Ketan Doshi

Transformers encoder

1. Each **encoder** has **two sub-layers** and each **decoder** has **three sub-layers**.
2. Each sublayer has residual connection.
3. All encoders receive a list of vectors each of the size d.
4. The size of this list is hyper-parameter we can set (it would be the length of the longest sentence in our training dataset).



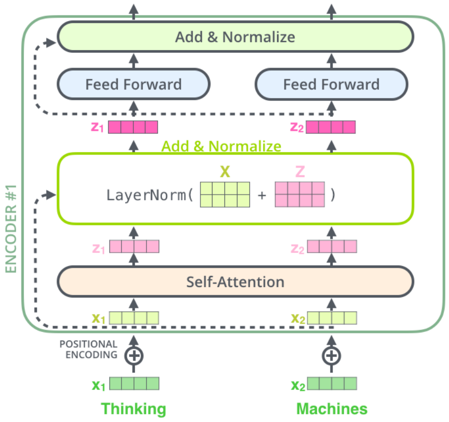


Figure: Jay Alammar

Transformers

1. A transformer of two stacked encoders and decoders

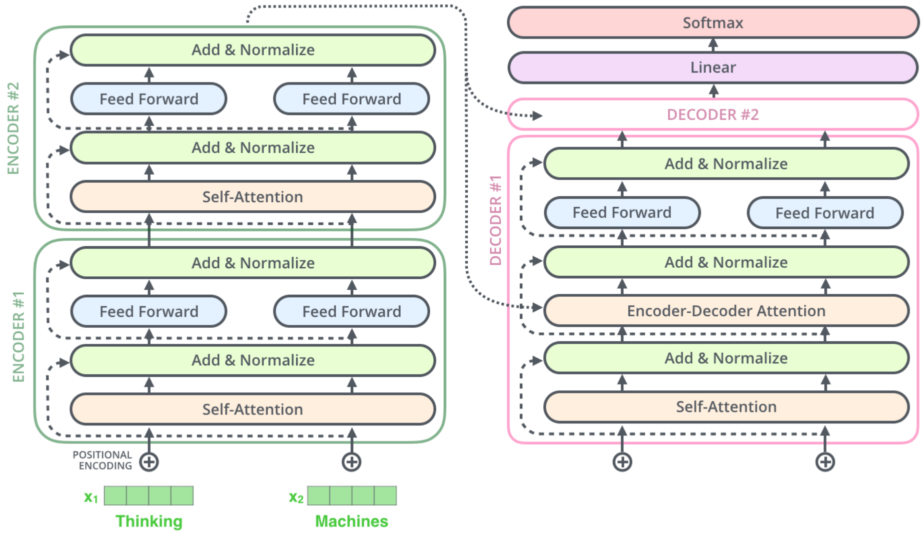
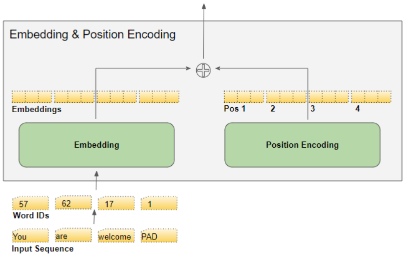


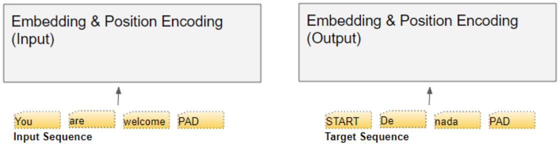
Figure: Jay Alammar

Transformers embedding and position encoding

1. Transformers needs two things for a word:
   - its meaning
   - its position in sequence



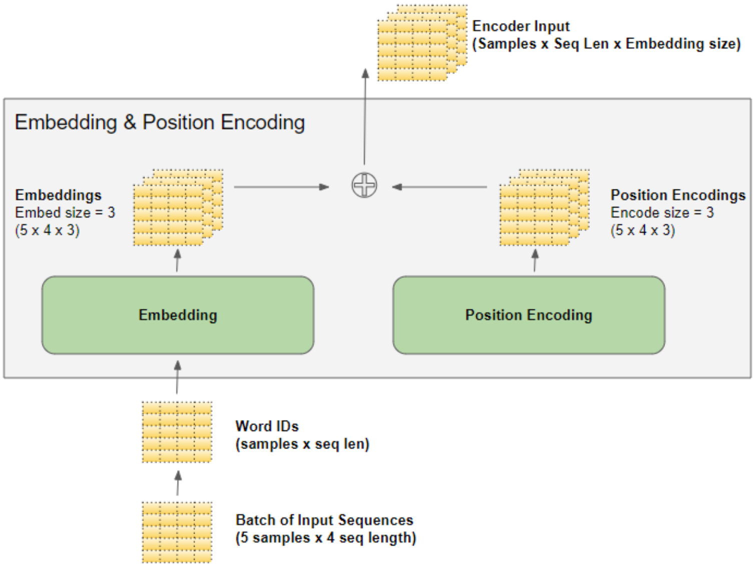
2. Transformers have two Embedding layers.
   - Input sequence is fed to the first embedding layer (Input Embedding).
   - Target sequence is fed to the second embedding layer after shifting the targets right by one position and inserting a START token in the first position.



Embedding & Position Encoding (Input)
You
are
welcome
PAD
Input Sequence
Embedding & Position Encoding (Output)
START
De
nada
PAD
Target Sequence

Transformers position encoding

1. There are two position encoding layers for: input sequence and output sequence.
2. Let d be size of embedding for each word and L be length of input sequence.
3. Transformers consider an array of $d \times L$ to encode positions of input sequence.



Transformers position encoding

1. All words in input sentence simultaneously flow through Transformer's encoder/decoder stack and model doesn't have any sense of position/order of each word.
2. Hence, we need for a way to incorporate the order of words into our model.
3. The first solution is to assign a number to each time-step within interval $[0, 1]$ range in which 0 means the first word and 1 is the last time-step.

   Problem: can't figure out how many words are present within a specific range, i.e, time-step delta doesn't have consistent meaning across different sentences.
4. The second solution is to assign a number to each time-step linearly.

   Problem: not only the values could get quite large, but also our model can face sentences longer than the ones in training.
5. Ideally, the following criteria should be satisfied for positional embedding
   - It should output a unique encoding for each time-step (word's position in a sentence)
   - Distance between any two time-steps should be consistent across sentences with different lengths.
   - The model should generalize to longer sentences without any efforts. Its values should be bounded.
   - It must be deterministic.

Transformers position encoding

1. The encoding of Transformers is simple and satisfies all of those criteria.
   - It is d-dimensional vector containing information about a specific position in a sentence.
   - This encoding is not integrated into the model itself. This vector is used to equip each word with information about its position in a sentence.
2. Let pos be the position of word in sequence.
3. Let $i \in \{1, 2, \ldots, d\}$ be the index value in positional encoding.
4. Then, PE is computed using

$$PE(pos, i) = \begin{cases} \sin\left(\frac{pos}{10000^{i/d}}\right) & \text{if } i = 2k \\ \cos\left(\frac{pos}{10000^{i/d}}\right) & \text{if } i = 2k+1 \end{cases}$$

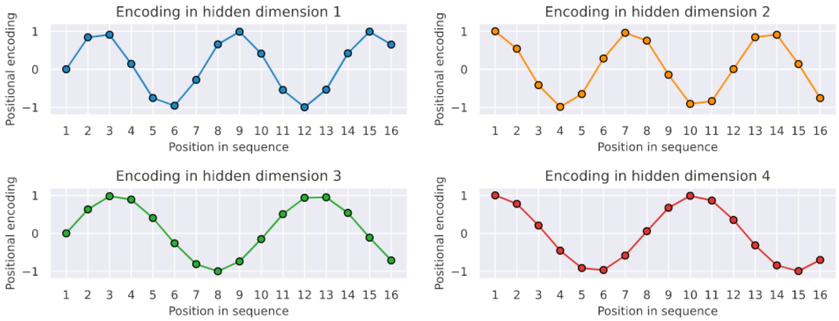
5. Let $d = 512$ and $pos = 0$, then $PE(0)$ and $PE(1)$ equal to

$$PE(0,0) = (0, 1, 0, \ldots, 0, 1)$$
$$PE(0,1) = (0.8414, 0.5403, 0.8218, \ldots, 0.0001, 0.9999)$$

Transformers position encoding

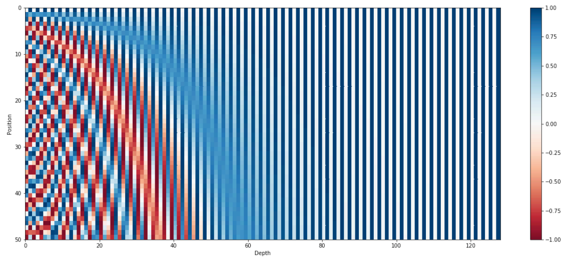
1. For word w at position $pos \in [0, L-1]$ in the input sequence $w = (w_0, \ldots, w_{L-1})$, with 4-dimensional embedding e_w, and $d = 4$, the operation would be

$$\begin{aligned} e'_w &= e_w + \left[sin\left(\frac{pos}{10000^0}\right), cos\left(\frac{pos}{10000^0}\right), sin\left(\frac{pos}{10000^{2/4}}\right), cos\left(\frac{pos}{10000^{2/4}}\right) \right] \\ &= e_w + \left[sin\left(pos\right), cos\left(pos\right), sin\left(\frac{pos}{100}\right), cos\left(\frac{pos}{100}\right) \right] \end{aligned}$$



Transformers position encoding

1. Position encoding interleaves a *sine* curve and a *cos* curve, with sine values for all even indexes and cos values for all odd indexes.
2. This results the following position encoding and the corresponding curves.



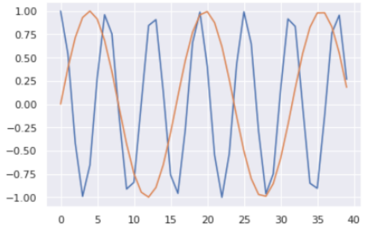
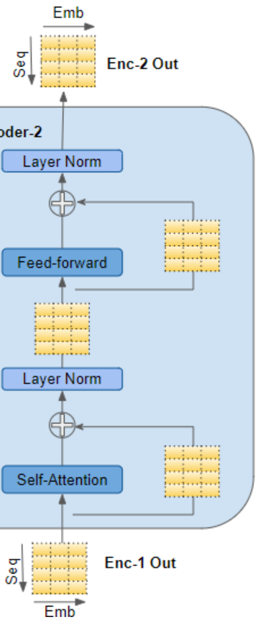


Figure: Ketan Doshi

Figure: Amirhossein Kazemnejad

Transformers encoder

1. The Encoder passes its input into a Multi-head Self-attention layer.
2. The Self-attention output is passed into a Feed-forward layer, which then sends its output upwards to the next Encoder.



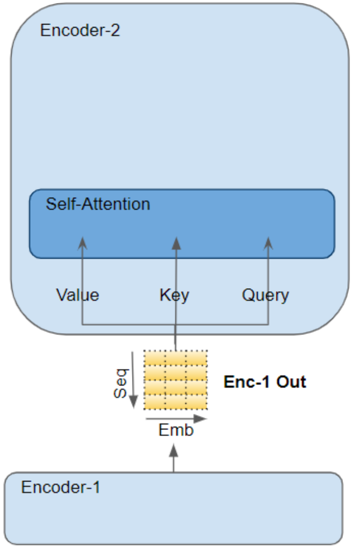
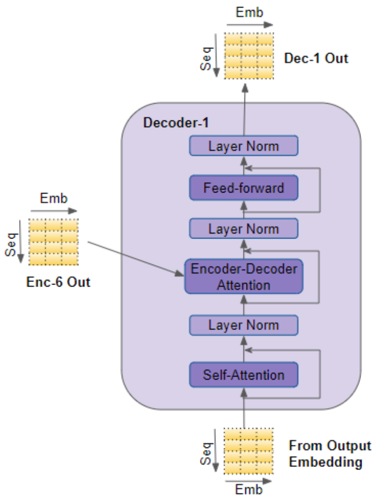


Figure: Ketan Doshi

Transformers decoder

1. The Decoder passes its input into a Multi-head Self-attention layer.
2. This operates in a slightly different way than the one in the Encoder.
3. It is only allowed to attend to earlier positions in the sequence. This is done by masking future positions.



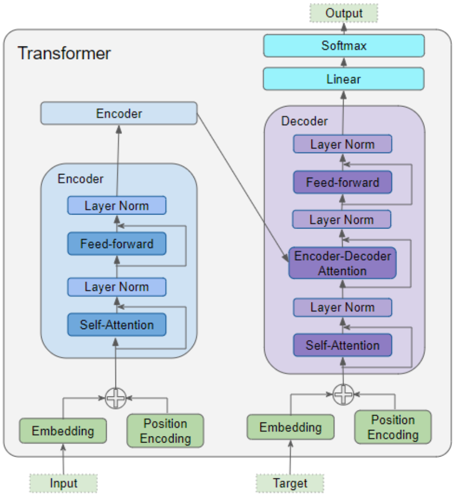
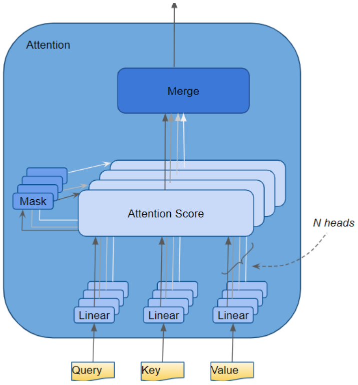


Figure: Ketan Doshi

Transformers multi-head attention

1. The Transformers calls each Attention processor an Attention Head and repeats it several times in parallel.
2. This is known as Multi-head attention.
3. It gives its Attention greater power of discrimination, by combining several similar Attention calculations.



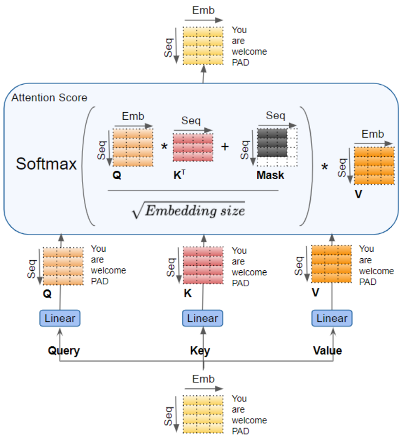


Figure: Ketan Doshi

Transformers multi-head attention

1. There are three separate Linear layers for the Query, Key, and Value.
2. Each Linear layer has its own weights.
3. The input is passed through these Linear layers to produce the **Q**, **K**, and **V** matrices.

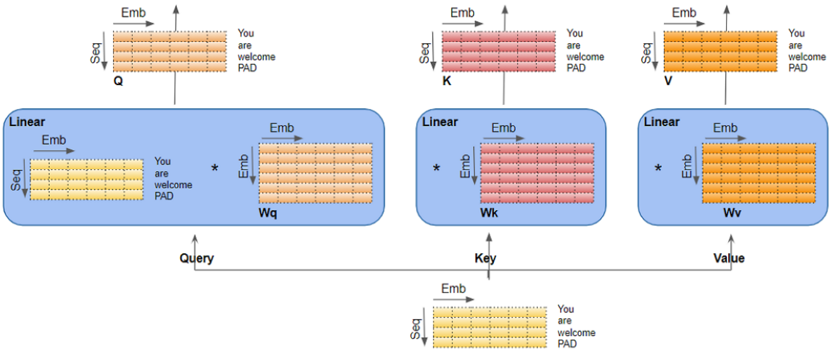


Figure: Ketan Doshi

Transformers multi-head attention

1. The data are split across the multiple Attention heads so that each can process it independently.
2. This is a logical split only. The Query, Key, and Value are not physically split into separate matrices, one for each Attention head.
3. A single data matrix is used for the Query, Key, and Value, respectively, with logically separate sections of the matrix for each Attention head.

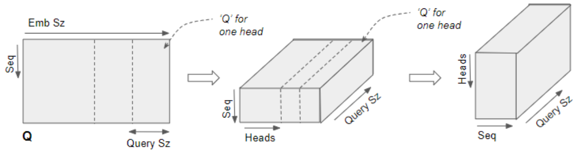
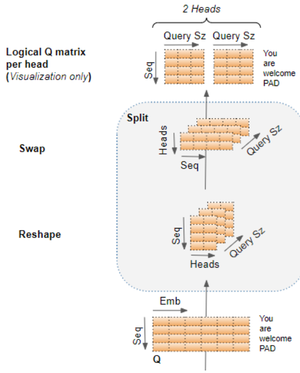


Figure: Ketan Doshi



Transformers multi-head attention

1. We now have separate Attention Scores for each head.
2. They need to be combined together into a single score.
3. This Merge operation is essentially the reverse of the Split operation.
4. It is done by simply reshaping the result matrix to eliminate the Head dimension.
    - Reshape the Attention Score matrix by swapping the Head and Sequence dimensions.
    - Collapse the Head dimension by reshaping .

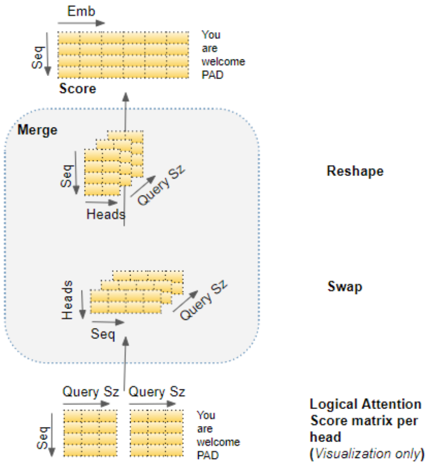


Figure: Ketan Doshi

Transformers multi-head attention

1. The end-to-end flow of the Multi-head Attention is

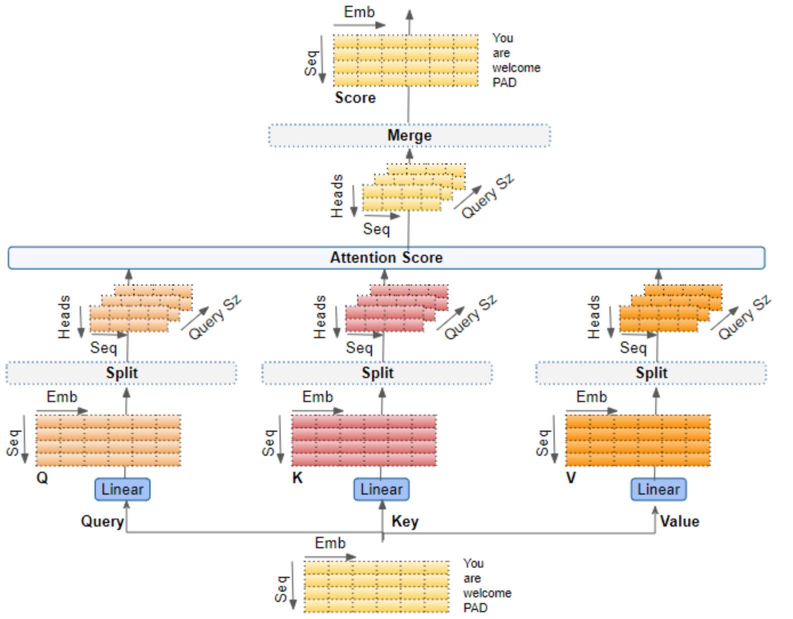
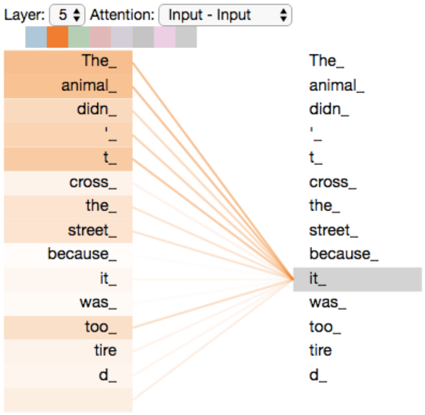


Figure: Ketan Doshi

Transformers multi-head attention

1. The different attention heads are focusing on different words as we encode the word it.



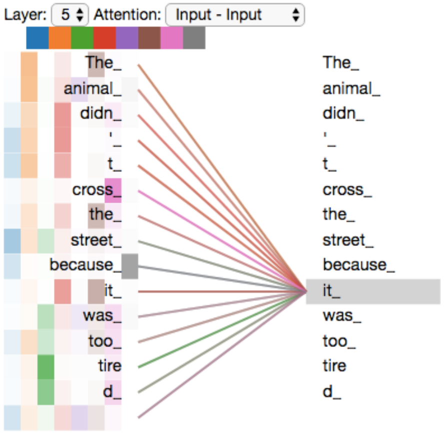


Figure: Jay Alammar

Transformers decoder attention layers

1. The attention layers of Transformers decoder are

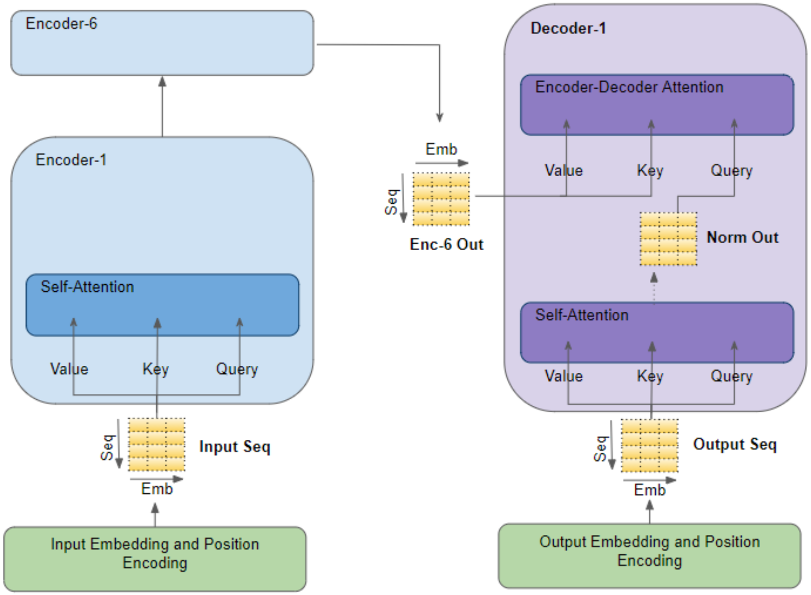


Figure: Ketan Doshi

Transformers decoder self-attention and masking

1. The Decoder Self-Attention works just like the Encoder Self-Attention, except that it operates on each word of the target sequence.

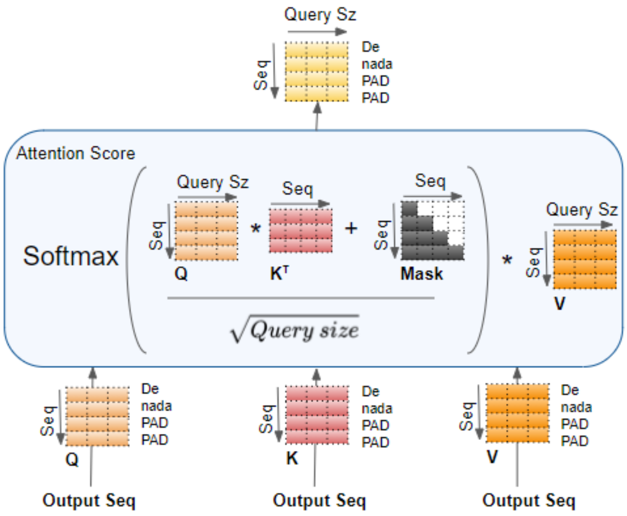


Figure: Ketan Doshi

Transformers decoder encoder-decoder attention and masking

1. The Encoder-Decoder Attention takes its input from two sources.
2. The Encoder-Decoder Attention computes the interaction between each target word with each input word.
3. The Masking masks out the Padding words in the target sequence.

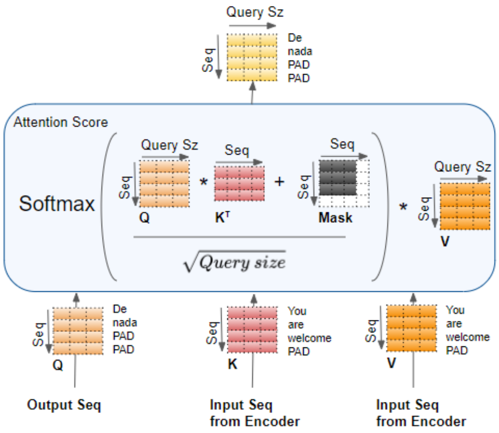


Figure: Ketan Doshi

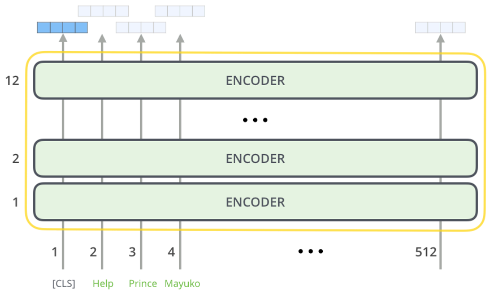
Simple Neural Attention Meta-Learner (SNAIL)

1. The SNAIL was developed partially to resolve the problem with positioning in the transformer model by combining the self-attention mechanism in transformer with temporal convolutions (Mishra et al. 2018).
2. It has been demonstrated to be good at both supervised learning and reinforcement learning tasks.

BERT model

BERT model

1. BERT (Pre-training of Deep Bidirectional Transformers for Language Understanding) is basically a trained Transformers Encoder stack (Devlin et al. 2019).
2. Each position outputs a vector. For the sentence classification, we focus on the output of only the first position ([CLS]).
3. That vector can now be used as the input for a classifier. The paper achieves great results by just using a single-layer neural network as the classifier.



4. BERT is trained with two tasks instead of the basic language task: **masked language model** and **next sentence prediction**.

BERT sentence classification

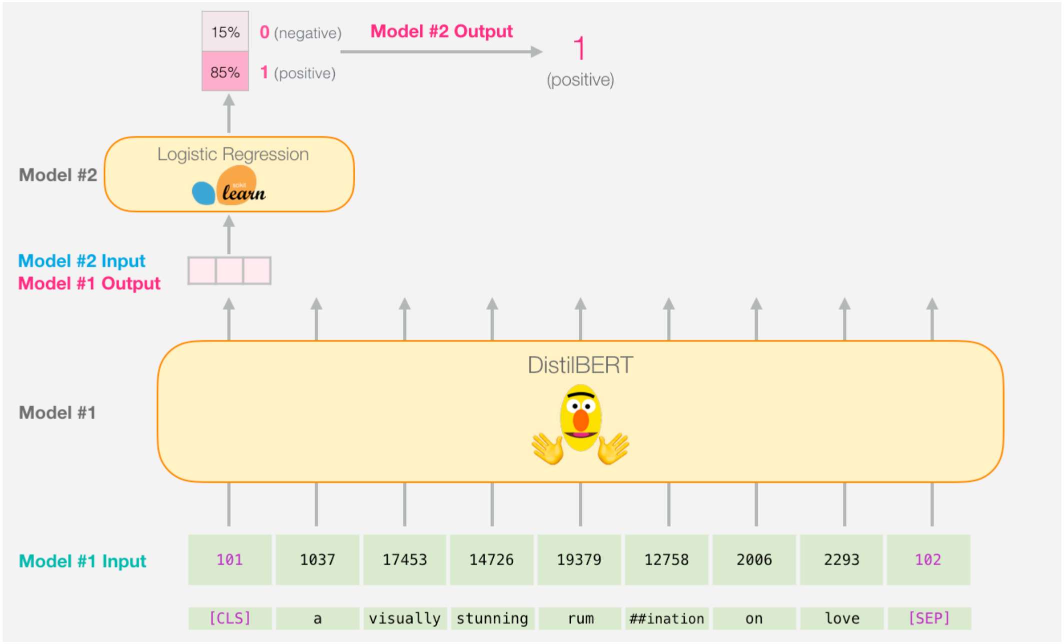


Figure: Jay Alammar

Task 1: Masked language model

1. In this task, we mask some words in the input sentence and the model tries to predict the masked words.
2. Randomly mask out 15% of the words in the input (replacing them with a [MASK] token) .
3. Then run the entire sequence through the BERT attention based encoder and predict only the masked words, based on the context provided by the other non-masked words in the sequence.
4. The problem here is : the model only tries to predict when the [MASK] token is present in the input, while we want the model to try to predict the correct tokens regardless of what token is present in the input.
5. To deal with this issue, out of the 15% of the tokens selected for masking:
    - 80% of the tokens are actually replaced with the token [MASK].
    - 10% of the time tokens are replaced with a random token.
    - 10% of the time tokens are left unchanged.

Task 1: Masked language model

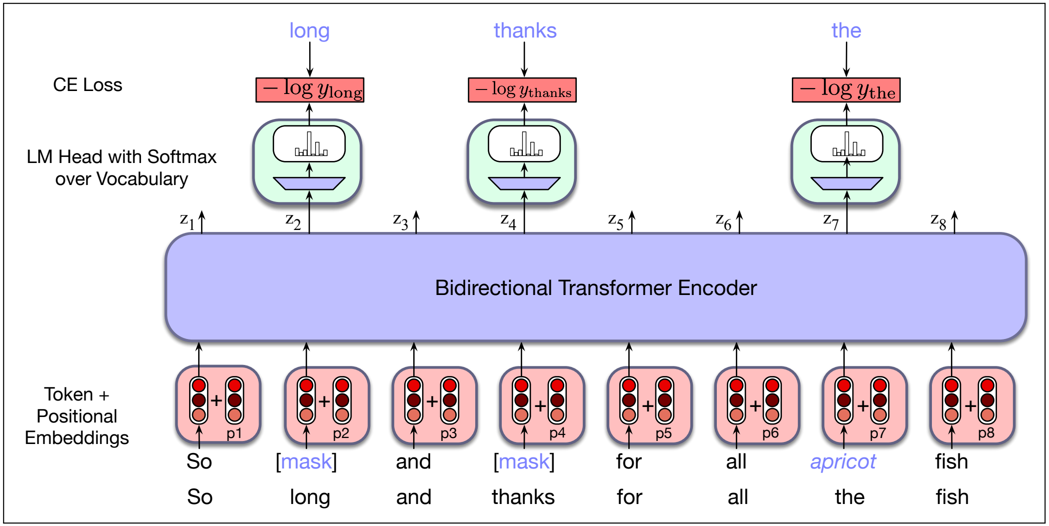
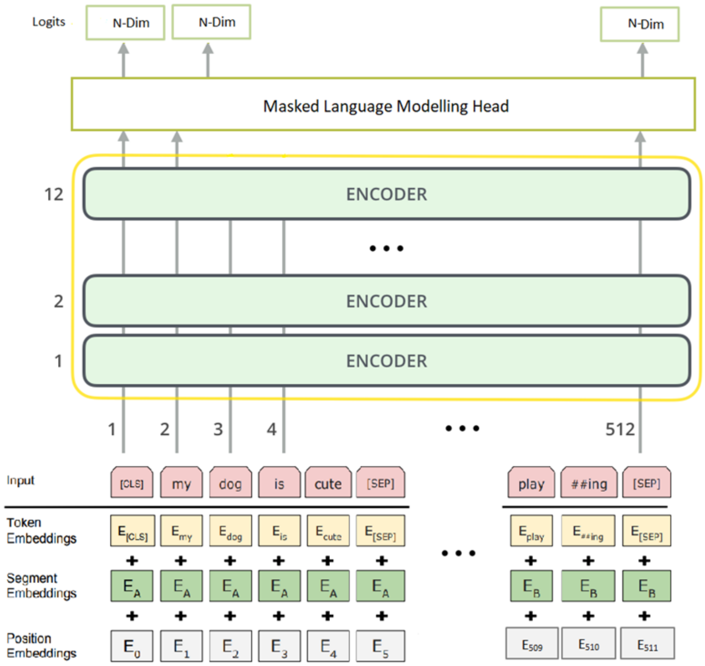

long
thanks
the
CE Loss
$-\log y_{\text{long}}$
$-\log y_{\text{thanks}}$
$-\log y_{\text{the}}$
LM Head with Softmax over Vocabulary
z_1 z_2 z_3 z_4 z_5 z_6 z_7 z_8
Bidirectional Transformer Encoder
Token + Positional Embeddings
p1 p2 p3 p4 p5 p6 p7 p8
So [mask] and [mask] for all apricot fish
So long and thanks for all the fish

Figure: Daniel Jurafsky & James H. Martinr

Task 1: Masked language model



Logits
N-Dim
N-Dim
N-Dim
Masked Language Modelling Head
12
ENCODER
2
ENCODER
1
ENCODER
1
2
3
4
512
Input
[CLS]
my
dog
is
cute
[SEP]
play
##ing
[SEP]
Token Embeddings
$E_{[CLS]}$ E_{my} E_{dog} E_{is} E_{cute} $E_{[SEP]}$ E_{play} $E_{\#\#ing}$ $E_{[SEP]}$
Segment Embeddings
E_A E_A E_A E_A E_A E_A E_B E_B E_B
Position Embeddings
E_0 E_1 E_2 E_3 E_4 E_5 E_{509} E_{510} E_{511}

Figure: Abhijit

Task 2: Next sentence prediction

1. To understand relationship between two sentences, BERT training process also uses next sentence prediction.
2. In this task, we train a binary classifier for telling whether one sentence is the next sentence of the other.
3. Sample sentence pairs (A, B) such that:
   - 50% of the time, B follows A;
   - 50% of the time, B does not follow A;
4. The model processes both sentences and output a binary label indicating whether B is the next sentence of A.
5. BERT separates sentences with a special [SEP] token.
6. To predict if the second sentence is connected to the first one or not, the output of the [CLS] token is given to a classifier.
7. The training data for both tasks can be generated from any monolingual corpus.
8. The training loss is the sum of the mean masked LM likelihood and the mean next sentence prediction likelihood.

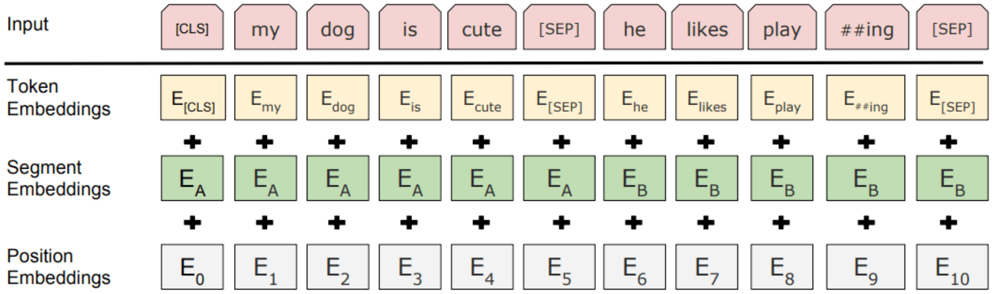
Input Embedding

1. BERT needs the input to be massaged and decorated with some extra meta data:

   Token embeddings A [CLS] token is added to the input word tokens at the beginning of the first sentence and a [SEP] token is inserted at the end of each sentence.

   Segment embeddings A marker indicating Sentence A or Sentence B is added to each token. This allows the encoder to distinguish between sentences.

   Positional embeddings A positional embedding is added to each token to indicate its position in the sentence.


Input
[CLS]
my
dog
is
cute
[SEP]
he
likes
play
##ing
[SEP]
Token Embeddings
$E_{[CLS]}$ E_{my} E_{dog} E_{is} E_{cute} $E_{[SEP]}$ E_{he} E_{likes} E_{play} $E_{\#\#ing}$ $E_{[SEP]}$
Segment Embeddings
E_A E_A E_A E_A E_A E_A E_B E_B E_B E_B E_B
Position Embeddings
E_0 E_1 E_2 E_3 E_4 E_5 E_6 E_7 E_8 E_9 E_{10}

2. Note that the first token is always forced to be [CLS]. This is a placeholder that will be used later for prediction in downstream tasks.

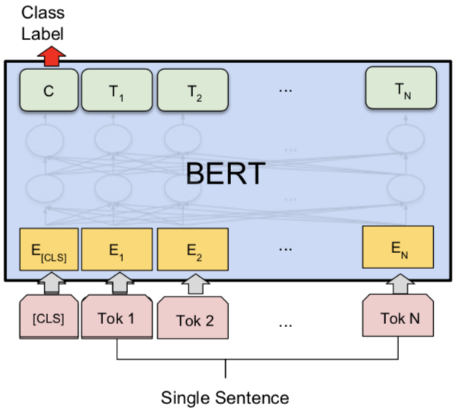
Using BERT for classification tasks

1. In this task, we get the prediction by taking the final hidden state of the token [CLS] denoted by $\mathbf{h}^{cls}$.
2. Then multiplying $\mathbf{h}^{cls}$ with a small weight matrix $\mathbf{W}^{cls}$, and pass trough a softmax layer:

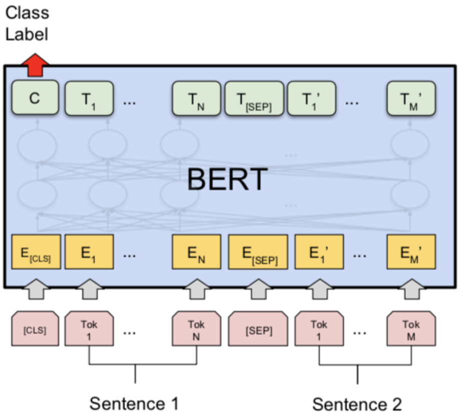
$$softmax(\mathbf{h}^{cls}\mathbf{W}^{cls})$$

.

Single sentence classification

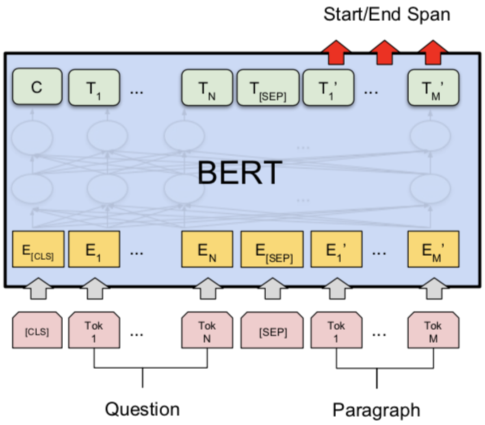


Sentence pair classification



Using BERT for question-answering task

1. In this task, we need to predict the text span in the given paragraph for an given question.
2. BERT predicts two probability distributions of every token:
   - being the start of the text span and
   - being the end of the text span.
3. Two new small matrices $\mathbf{W}_s$ and $\mathbf{W}_e$ are learned during fine-tuning.
4. Two probability distributions are defined by $softmax(h^{(i)}\mathbf{W}_s)$ and $softmax(h^{(i)}\mathbf{W}_e)$.

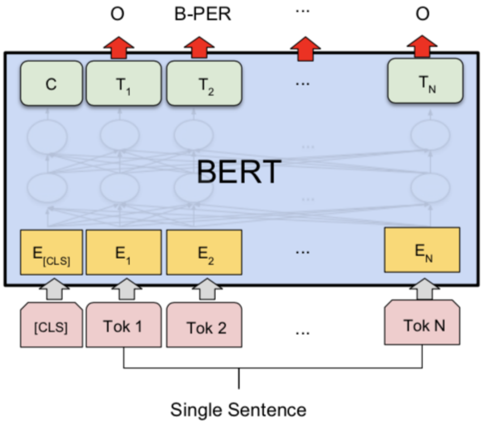


Using BERT for single sentence tagging task

1. In this task, we get the prediction by taking the final hidden state of the token [CLS] denoted by $\mathbf{h}^{cls}$.
2. Then multiplying $\mathbf{h}^{cls}$ with a small weight matrix $\mathbf{W}^{cls}$, and pass trough a softmax layer:

$$softmax(\mathbf{h}^{cls}\mathbf{W}^{cls})$$

.



BERT pre-trained architecture

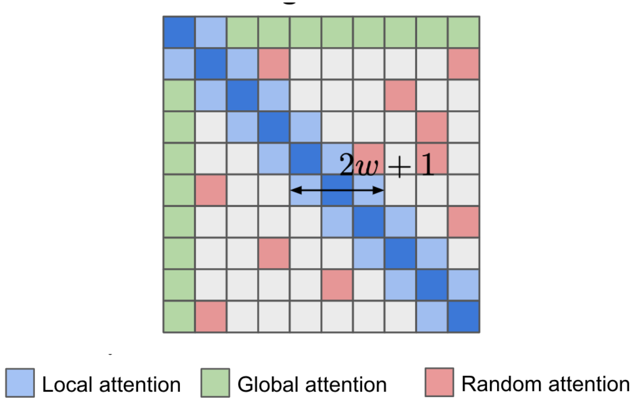
1. There are two types of pre-trained versions of BERT depending on the scale of the model architecture

BERT-Base	12-layer, 768-hidden-nodes, 12-attention-heads, 110M parameters.
BERT-Large	24-layer, 1024-hidden-nodes, 16-attention-heads, 340M parameters.

BigBird Model

BigBird Model

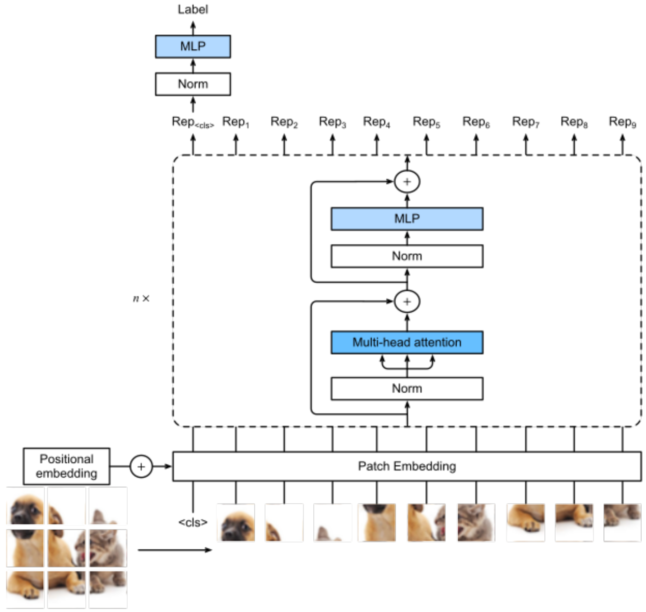
1. BERT works on a full self-attention mechanism.
   - This leads to a quadratic growth of the computational and memory requirements for every new input token.
   - The maximum input size is around 512 tokens.
   - This means that the model cannot be used for larger inputs and for tasks like large document summarization.
2. BigBird runs on a sparse attention mechanism to overcome quadratic dependency of BERT while preserving the properties of full-attention models (Zaheer et al. 2020).



Vision Transformer

Vision Transformer

1. Vision Transformer (ViT) has the following architecture (Dosovitskiy et al. 2021).
2. ViT uses the Transformer encoder.



3. For more information on ViT family, read (Islam 2022).

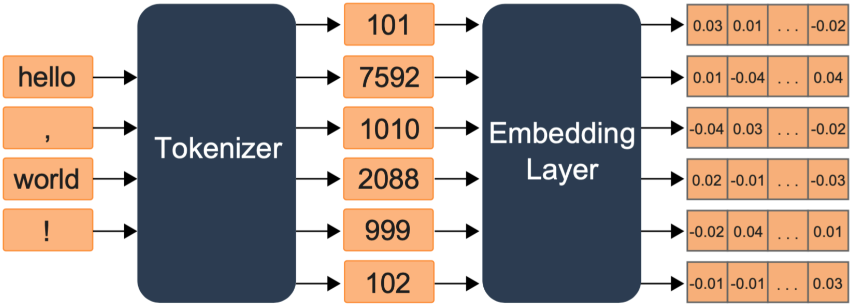
ALBERT model

ALBERT model

1. ALBERT (A Litle BERT) is a light-weighted version of BERT model (Lan et al. 2020).
2. An ALBERT model can be trained 1.7x faster with 18x fewer parameters, compared to a BERT model of similar configuration.
3. ALBERT incorporates three changes as follows:
    - Factorized embedding parameterization: (reduces parameters and memory requirements and results in speed up the training speed.)
    - Cross-layer parameter sharing: (reduces parameters and memory requirements and results in speed up the training speed.)
    - Sentence-order prediction: (proposes a more challenging training task to replace the next sentence prediction objective.)

Factorized embedding parameterization

1. In BERT, embedding size (E) equals to hidden state size (H).



2. The input embedding is a matrix of size $V \times E$ in which

$$H = 30522$$
$$E = 768$$

3. Hence, this matrix has the size of 30522×768.

Factorized embedding parameterization

1. To increase the model size (larger H), we need a larger tokenization embedding (larger H).
2. This is expensive because it depends on the vocabulary size (V).
3. The tokenization embedding is context-independent representation while the hidden states are context-dependent.
4. It is better to separate the size of the hidden layers from the size of vocabulary embedding.
5. Here, the large vocabulary embedding matrix of size $V \times H$ is decomposed into two small matrices of size

$$V \times E$$

and

$$E \times H$$

   where $H > E$ or $H \gg E$.
6. Instead of projecting the one-hot vectors directly into the hidden space of size H,
   - we first project them into a lower dimensional embedding space of size E, and
   - then project it to the hidden space H.

Cross-layer parameter sharing

1. Parameter sharing across layers can happen in many ways:
   - only share feed-forward part
   - only share attention parameters
   - share all the parameters
2. This technique reduces the number of parameters and does not damage the performance too much.

Sentence-order prediction

1. ALBERT adopted a sentence-order prediction (SOP) self-supervised loss,
   - **Positive sample**: two consecutive segments from the same document.
   - **Negative sample**: two consecutive segments from the same document, but the segment order is switched.
2. SOP is harder as it requires the model to fully understand the coherence and ordering between segments in comparison with next sentence prediction (NSP).

GPT model

GPT model

1. The GPT (Generative Pre-training Transformer) is built using transformer decoder blocks (Radford et al. 2019).
2. BERT uses transformer encoder blocks.
3. A key difference between the two is that GPT2 outputs one token at a time.

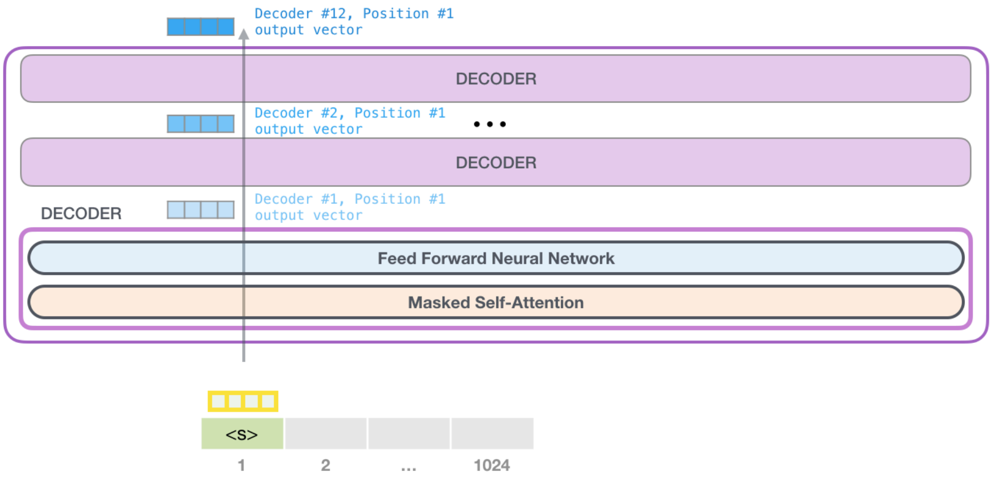
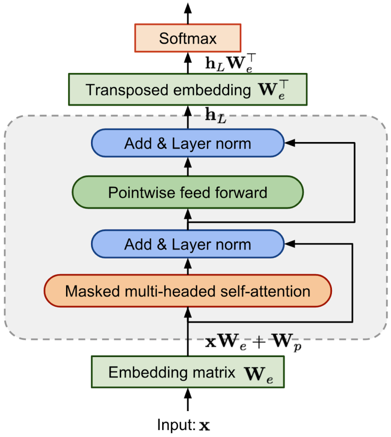

Decoder #12, Position #1 output vector
DECODER
Decoder #2, Position #1 output vector
...
DECODER
DECODER
Decoder #1, Position #1 output vector
Feed Forward Neural Network
Masked Self-Attention
<s>
1
2
...
1024

Figure: Jay Alammar

GPT model

1. GPT applies multiple transformer decoder-blocks over the embeddings of input sequences.
2. Each block contains a masked multi-headed self-attention layer and a pointwise feed-forward layer.
3. The final output produces a distribution over target tokens after softmax normalization .



Softmax
$\mathbf{h}_L\mathbf{W}_e^\top$
Transposed embedding $\mathbf{W}_e^\top$
$\mathbf{h}_L$
Add & Layer norm
Pointwise feed forward
Add & Layer norm
Masked multi-headed self-attention
$\mathbf{x}\mathbf{W}_e + \mathbf{W}_p$
Embedding matrix $\mathbf{W}_e$
Input: $\mathbf{x}$

Credit: Lilian Weng

4. GPT is called **unidirectional** while BERT is called **Bi-directional**.

GPT-1 unsupervisded pre-trained model

1. This task is to predict token u_i based previous k tokens $(u_{i-k}, \ldots, u_{i-1})$.
2. Given tokens $S = \{u_1, \ldots, u_n\}$, the objective is to maximize the probability likelihood to predict the next token as

$$\begin{aligned} L_1(S) &= \sum_i \log P(u_i \mid u_{i-k}, \ldots, u_{i-1}) \\ &= \log \prod_i P(u_i \mid u_{i-k}, \ldots, u_{i-1}) \end{aligned}$$

   where k is the size of the context window.
3. $\prod_i P(u_i \mid u_{i-k}, \ldots, u_{i-1})$ is the joint probability of the prediciton for each word.
4. In comparison with BERT
   - GPT-1 choose to predict the next word based on the previous k words, while BERT will use the words before and after the target word.
   - This makes the task in GPT a little more difficult than BERT, and thus its performance may not be as good as BERT in some tasks.

GPT-1 supervised fine-tuning

1. After GPT pre-training, it can be fine-tuned by some supervised tasks based on labeled data C.
2. Let $x = (x^1, \dots, x^m)$ be an input sequence and y be its label.
3. Input x is passed throught the pre-trained transformer block to get the activation h_L^m.
4. The activation h_L^m is fed into a linear layer with softmax to predict the probability for y as

$$P(y \mid x^1, \dots, x^m) = \text{softmax}(h_L^m W_y)$$

5. The objective is to maximize the likelihood

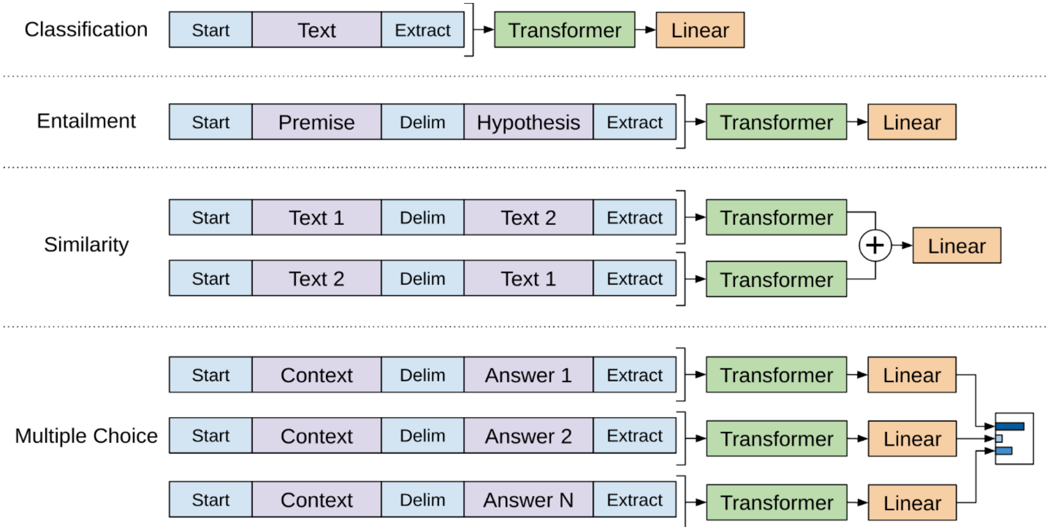
$$L_2(C) = \sum_{(x,y)} \log P(y \mid x^1, \dots, x^m)$$

6. For accelerating the convergence, the authors maximized the following objective function

$$L(C) = \lambda L_1(C) + L_2(C).$$

GPT-1 supervised model tasks and input transformation

1. The input to transformer decoder is a sequence of tokens.
2. For the subtasks like classification, it can be directly input to the transformer decoder.
3. For other tasks like Entailment, Similarity and Multiple choices, the data was re-structured by adding some special tokens to indicate Start, Delim and Extract as



GPT-2 model

1. Performance of a pre-trained model can be improved with more complex model (deeper and wider).
2. GPT-2, similar to GPT-1, is also the self-supervised model with transformer decoder but much more parameters.
3. GPT-2 mainly focus on zero-shot learning (without additional training, the model can performs good in some tasks).
4. Zero-shot learning shows that GPT-2 has a strong generalization ability, which is lacked in BERT.
5. Since GPT-2 doesn't have fine-tuning tasks, it does not need special tokens like Start, Delim and Extract.
6. GPT-2 uses a **prompt** to control the input of model.
7. A **prompt** is a small piece of text provided to the model, and the model will generate the additional text based on this input.
8. The **prompt** is task specific and depends on the specific input sequence and task.
9. For example, to translate an English text to French, the following prompt can be used

   (translate to french, english text, french text)

GPT-2 pre-trained architecture

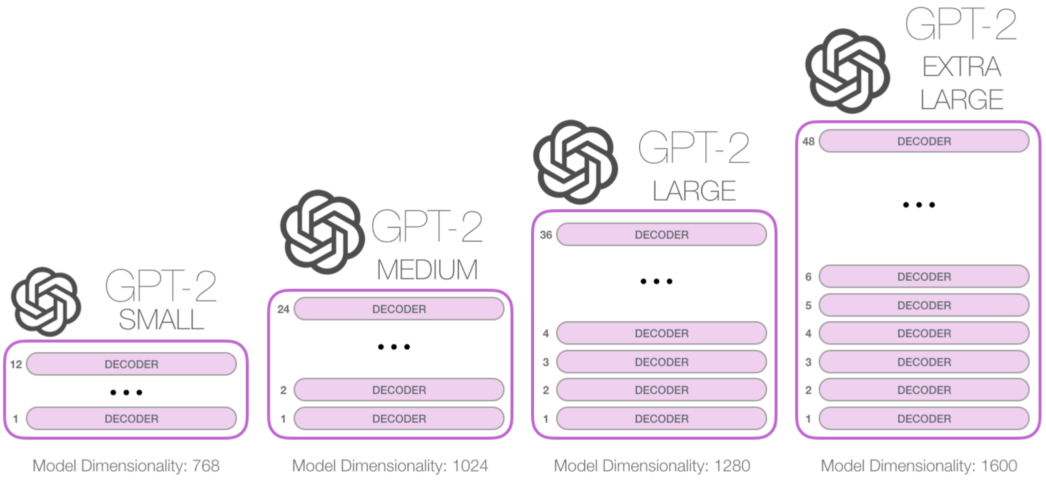
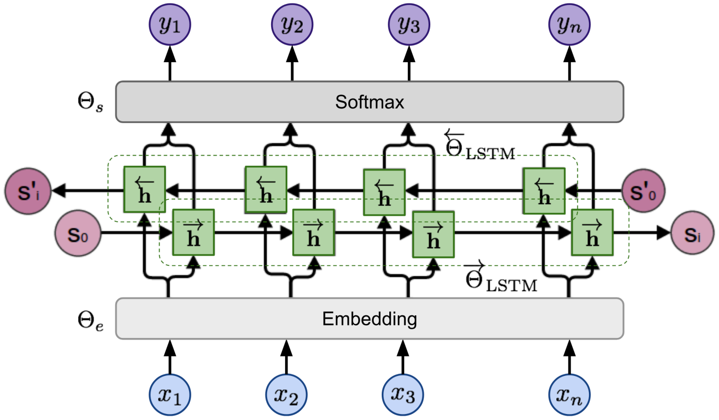


Figure: Jay Alammar

ELMo model

Embeddings from Language Model (ELMo)

1. ELMo learns contextualized word representation by pre-training a language model in an unsupervised way (Peters et al. 2018).



Embeddings from Language Model (ELMo)

1. The bidirectional Language Model (biLM) is the foundation for ELMo.
2. While the input is a sequence of n tokens, $(x_1, \dots, x_n)$, the language model learns to predict the probability of next token given the history.
3. In the forward pass, the history contains words before the target token,

$$p(x_1, \dots, x_n) = \prod_{i=1}^{n} p(x_i \mid x_1, \dots, x_{i-1})$$

4. In the backward pass, the history contains words after the target token,

$$p(x_1, \dots, x_n) = \prod_{i=1}^{n} p(x_i \mid x_{i+1}, \dots, x_n)$$

5. The predictions in both directions are modeled by multi-layer LSTMs with hidden states.

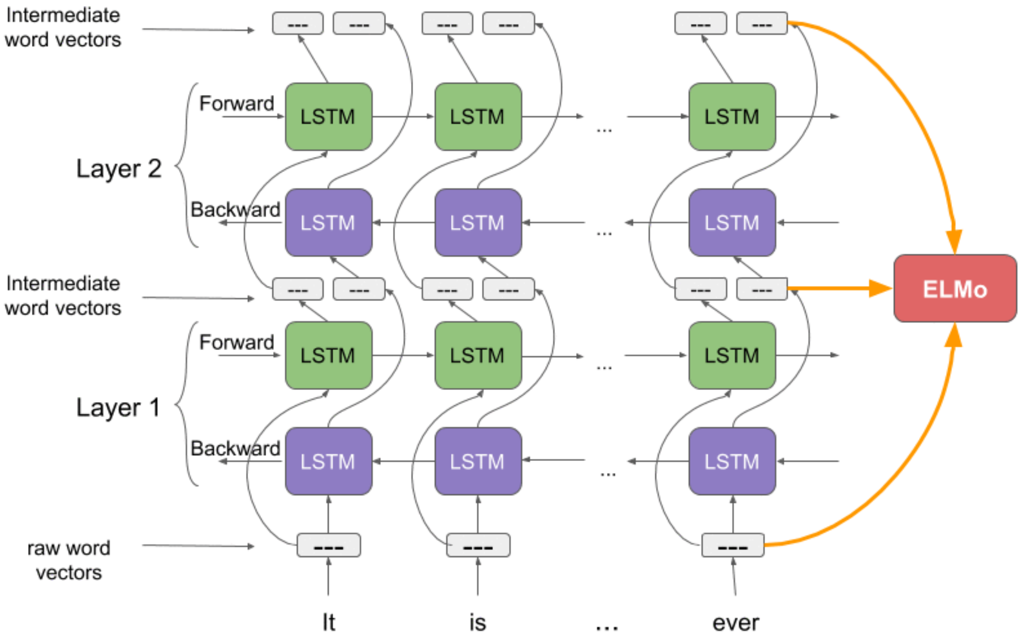
Embeddings from Language Model (ELMo)

1. The model is trained to minimize the negative log likelihood ($=$ maximize the log likelihood for true words) in both directions:

$$\mathcal{L} = -\sum_{i=1}^{n} \Big(\log p(x_i \mid x_1, \ldots, x_{i-1}) + \log p(x_i \mid x_{i+1}, \ldots, x_n) \Big)$$

2. ELMo word representations are functions of the entire input sentence.
3. A linear combination of the vectors stacked above each input word is learned as the representation of each token.

Embeddings from Language Model (ELMo)



Reading

References i

Devlin, Jacob et al. (2019). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proc. of Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 4171–4186.

Dosovitskiy, Alexey et al. (2021). "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *Proc. of the 9th International Conference on Learning Representations, ICLR 2021*.

Islam, Khawar (2022). "Recent Advances in Vision Transformer: A Survey and Outlook of Recent Work". In: *CoRR* abs/2203.01536. URL: `https://doi.org/10.48550/arXiv.2203.01536`.

Lan, Zhenzhong et al. (2020). "ALBERT: A Lite BERT for Self-supervised Learning of Language Representations". In: *8th International Conference on Learning Representations, ICLR*.

Mishra, Nikhil et al. (2018). "A Simple Neural Attentive Meta-Learner". In: *International Conference on Learning Representations*.

Peters, Matthew E. et al. (2018). "Deep Contextualized Word Representations". In: *Proc. of Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 2227–2237.

Radford, Alec et al. (2019). *Language Models are Unsupervised Multitask Learners*. Technical report, OpenAi.

References ii

Vaswani, Ashish et al. (2017). “Attention is All you Need”. In: *Advances in Neural Information Processing Systems*, pp. 5998–6008.

Zaheer, Manzil et al. (2020). “Big Bird: Transformers for Longer Sequences”. In: *Advances in Neural Information Processing Systems*.

Questions?