BERT and The History Behind It

Daniil Osokin

Agenda

1. Some typical NLP tasks.
2. Sequence to sequence learning (Seq2Seq).
3. Transformer.
4. BERT.

Some typical NLP tasks

- Named entity recognition (NER), intent classification, slot filling (chatbots, virtual assistants):

Sentence	*show*	*flights*	*from*	*Boston*	*To*	*New*	*York*	*today*
Slots/Concepts	O	O	O	B-dept	O	B-arr	I-arr	B-date
Named Entity	O	O	O	B-city	O	B-city	I-city	O
Intent	*Find_Flight*							
Domain	*Airline Travel*							

Identify known entities in a given text, classify the intent.

G. Mesnil et al. Using Recurrent Neural Networks for Slot Filling in Spoken Language Understanding.

Some typical NLP tasks

- Question answering:

context: Beyoncé Giselle Knowles-Carter (/biːˈjɒnseɪ/ bee-YON-say) (born September 4, 1981) is an American singer, songwriter, record producer and actress. Born and raised in Houston, Texas, she performed in various singing and dancing competitions as a child, and rose to fame in the late 1990s as lead singer of R&B girl-group Destiny's Child. Managed by her father, Mathew Knowles, the group became one of the world's best-selling girl groups of all time. Their hiatus saw the release of Beyoncé's debut album, Dangerously in Love (2003), which established her as a solo artist worldwide, earned five Grammy Awards and featured the Billboard Hot 100 number-one singles "Crazy in Love" and "Baby Boy".
question: What was the name of Beyoncé's first solo album?
answer: Dangerously in Love

- Neural machine translation (NMT):

Je m'appelle Daniil → My name is Daniil

Stanford Question Answering Dataset (SQuAD).

Tweet sentiment extraction

Given a tweet and sentiment, predict words subsequence, which responsible for the sentiment.

tweet: I`d have responded, if I were going
sentiment: neutral
answer: I`d have responded, if I were going

tweet: Sooo SAD I will miss you here in San Diego!!!
sentiment: negative
answer: Sooo SAD

tweet: I really really like the song Love Story by Taylor Swift
sentiment: positive
answer: like

tweet: I`m sorry.
sentiment: negative
answer: I`m sorry.

tweet: is back home now gonna miss every one
sentiment: negative
answer: onna

tweet: hm... Both of us I guess...
sentiment: neutral
answer: hm... Both of us I guess...

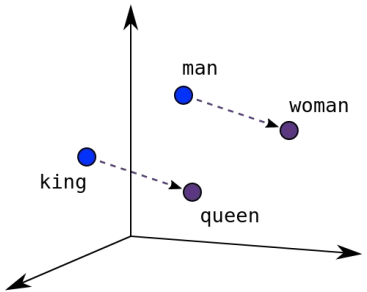
Kaggle Tweet Sentiment Extraction Challenge.

From words to vectors

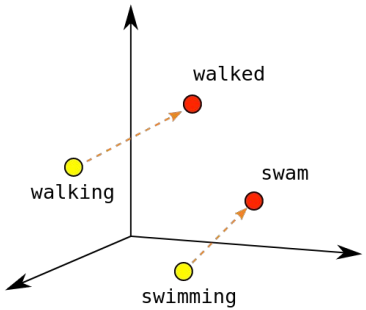
Q: How to map words to vectors?

From words to vectors

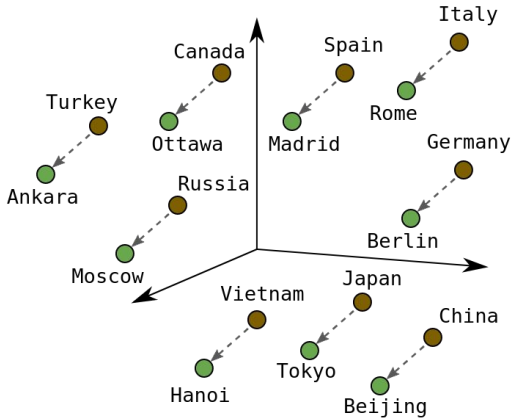
- One-hot encoding.
- Word2vec, GloVe.
- Embedding layer.



Male-Female



Verb Tense



Country-Capital

https://developers.google.com/machine-learning/crash-course/embeddings/translating-to-a-lower-dimensional-space.

Slide name is hidden

1. $W^{hx}x$ = ?

Slide name is hidden

1. $W^{hx}x$ = ?
2. sigm($W^{hx}x$) = ?

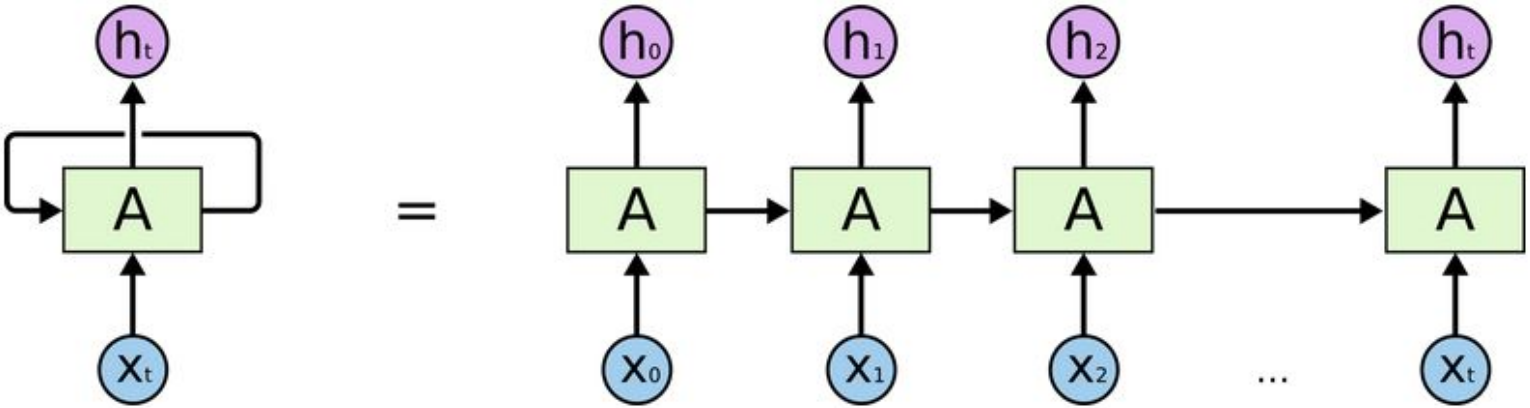
Slide name is hidden

1. $W^{hx}x$ = FC.
2. sigm($W^{hx}x$) = FC + activation.
3. sigm($W^{hx}x_t + W^{hh}h_{t-1}$) = ?

Recurrent neural network

1. $W^{hx}x$ = FC.
2. sigm($W^{hx}x$) = FC + activation.
3. sigm($W^{hx}x_t + W^{hh}h_{t-1}$) = h_t, recurrent unit.

y = $W^{yh}h_t$ - output, e.g. logits for intent classification.

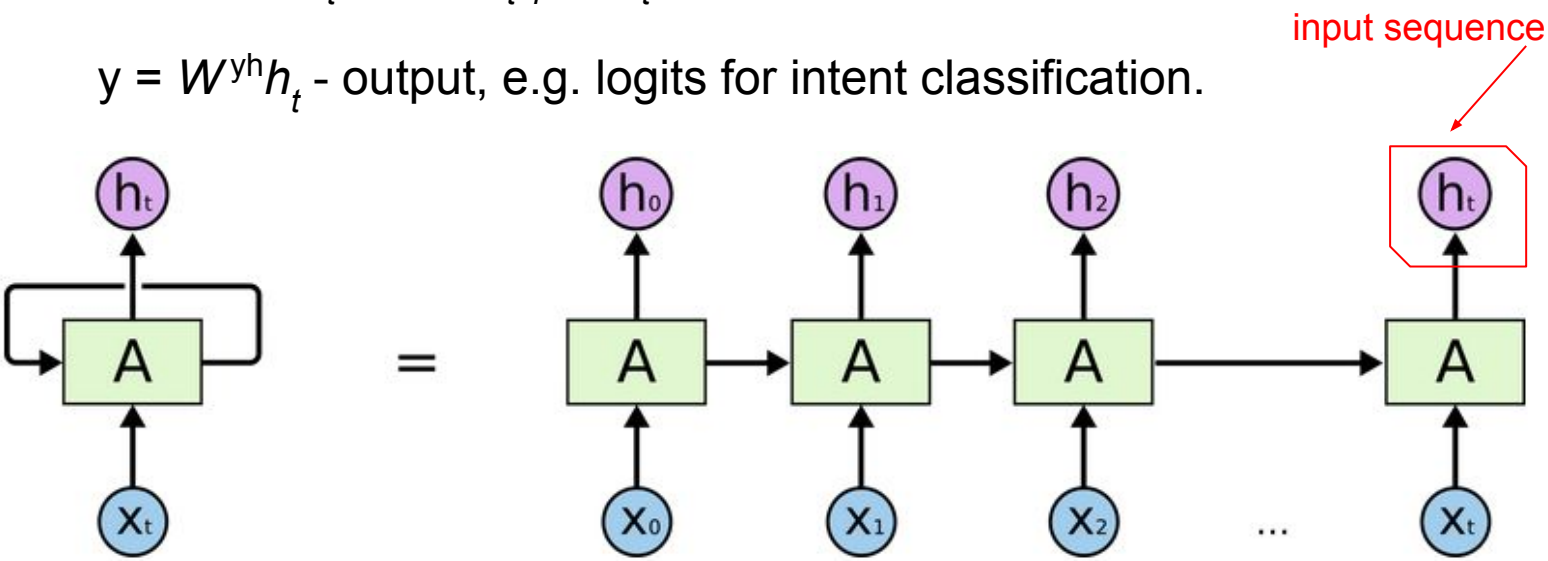


C.Olah. Understanding LSTM Networks.

Recurrent neural network

1. $W^{hx}x$ = FC.
2. sigm($W^{hx}x$) = FC + activation.
3. sigm($W^{hx}x_t + W^{hh}h_{t-1}$) = h_t, recurrent unit.

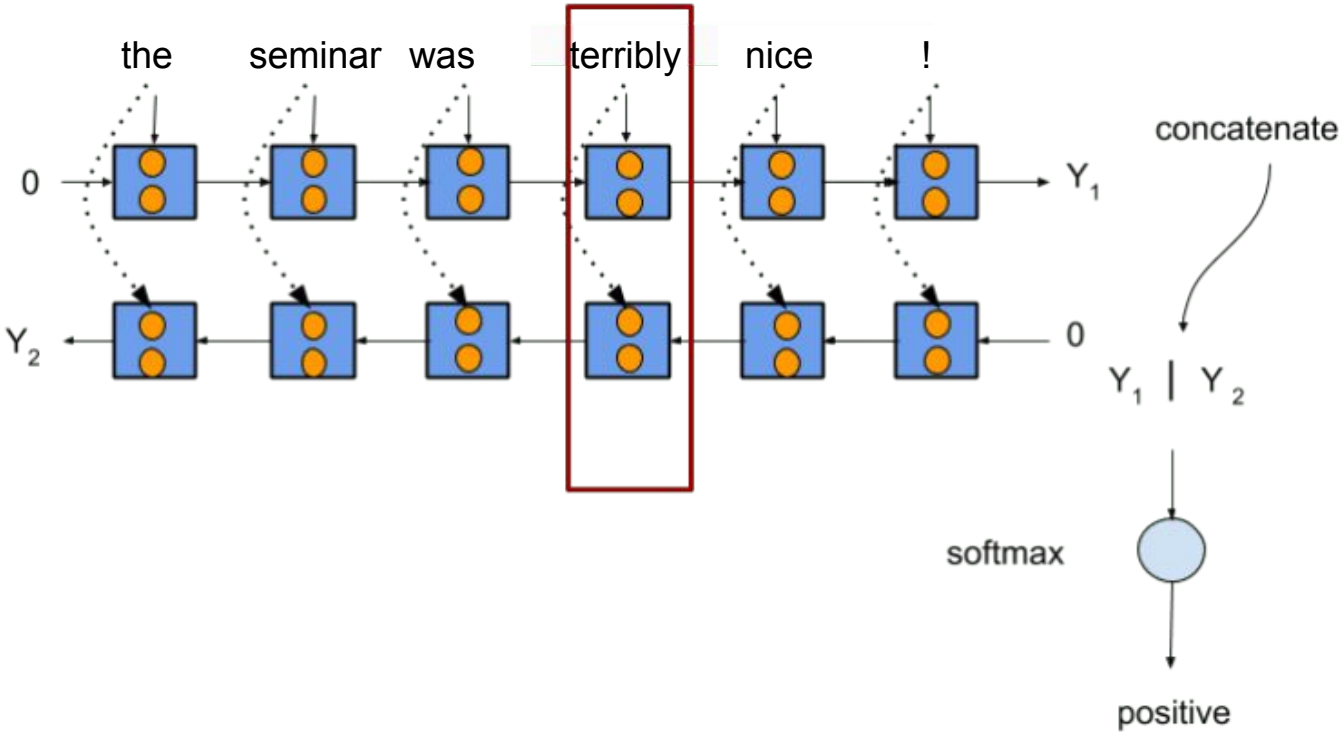
y = $W^{yh}h_t$ - output, e.g. logits for intent classification.



C.Olah. Understanding LSTM Networks.

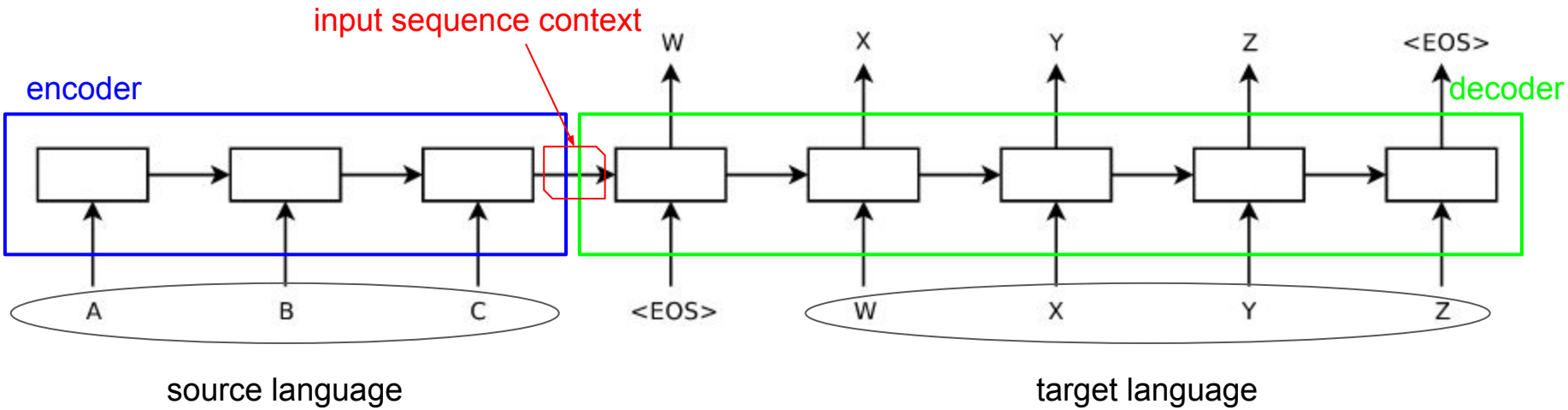
Bidirectional recurrent neural network

Consider review: “The seminar was terribly nice!”, the task is to predict sentiment.



Bidirectional LSTM using Keras.

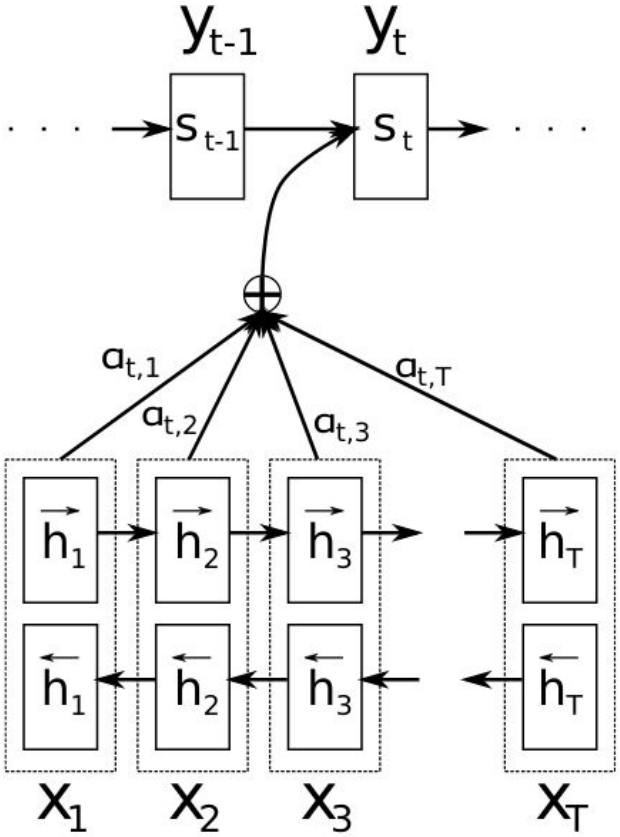
Seq2Seq: text translation



Encoder and decoder are RNNs (LSTM, GRU).

I.Sutskever et al. Sequence to Sequence Learning with Neural Networks.

Seq2Seq: text translation with attention



Own context vector c_i for each target y_i.

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})}$$

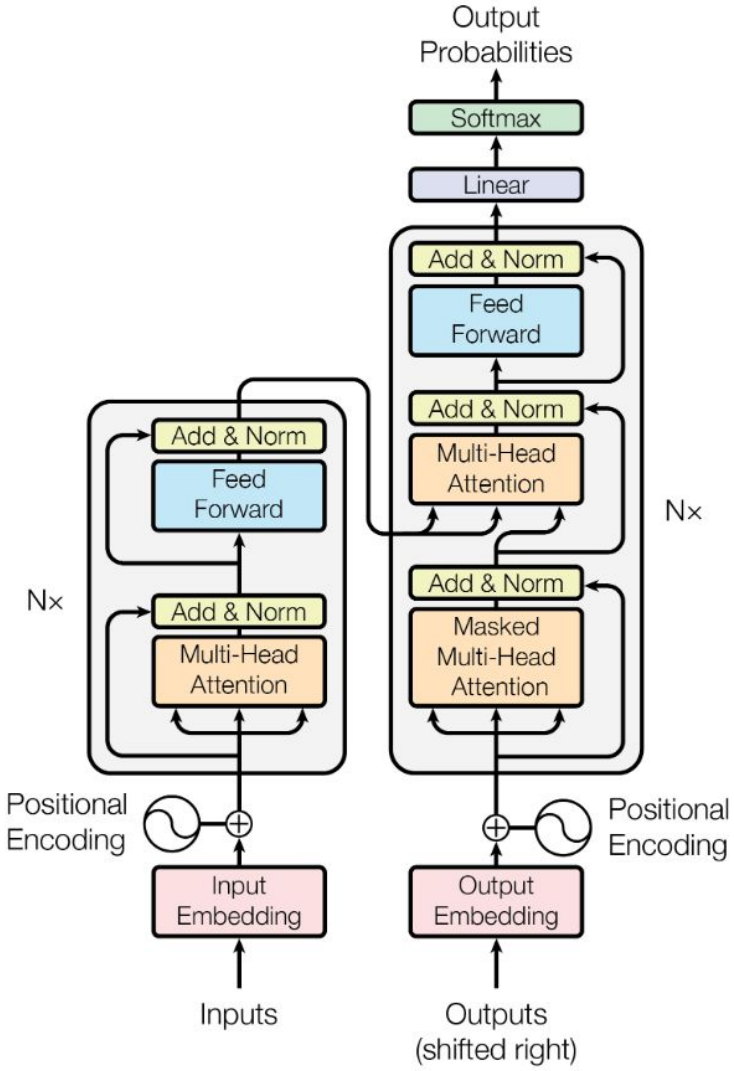
$$e_{ij} = a(s_{i-1}, h_j)$$

a(s, h) - model (fc) which scores how well the inputs around position j and the output at position i match.

D. Bahdanau et al. Neural Machine Translation by Jointly Learning to Align and Translate.

Transformer

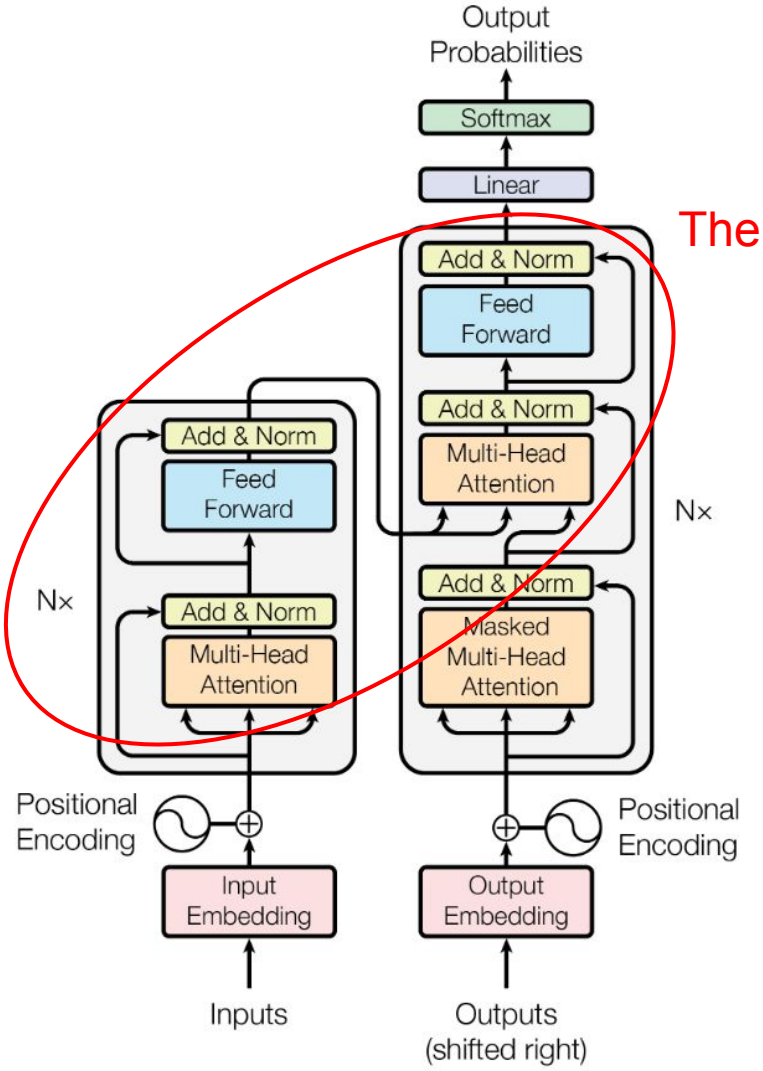
Looks simple, huh?



A. Vaswani et al. Attention Is All You Need.

Transformer

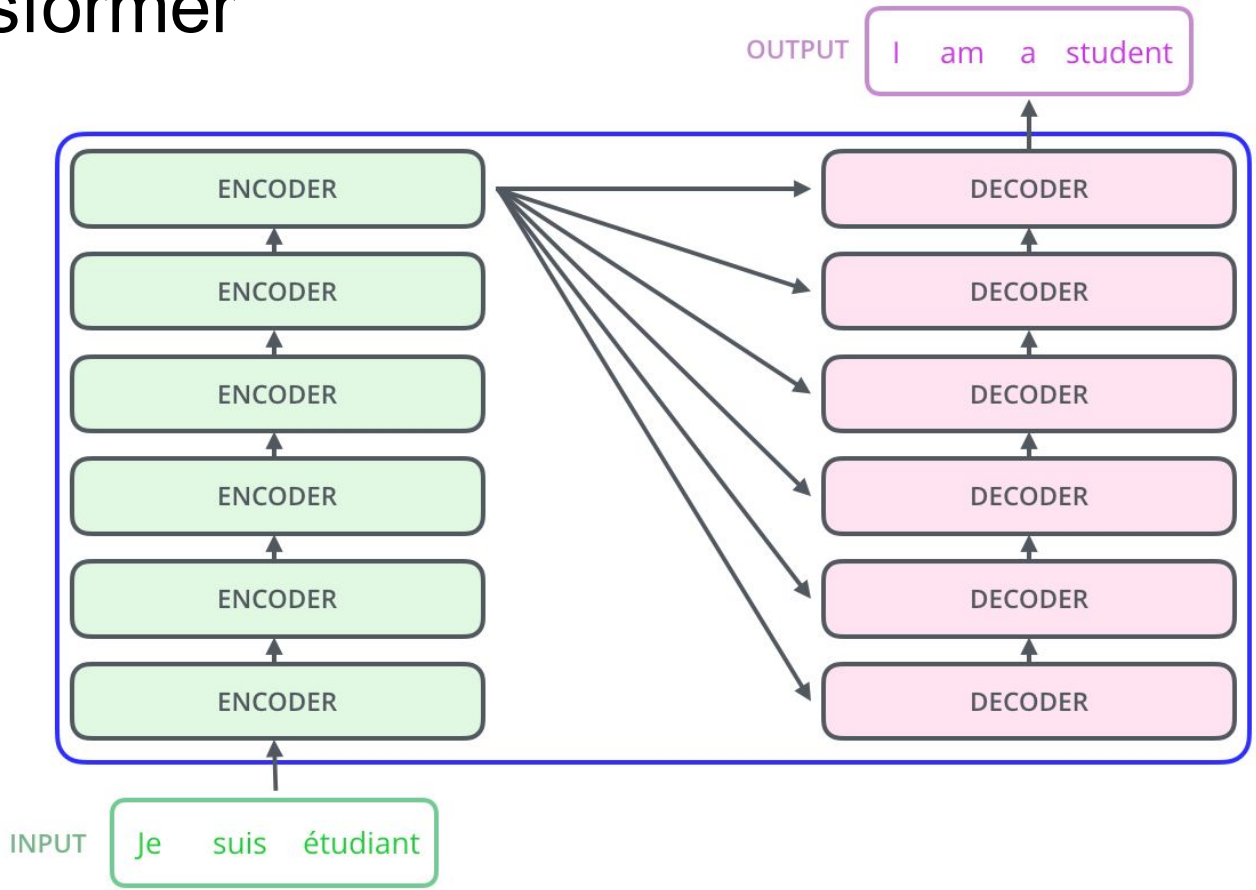
Looks simple, huh?



The same blocks structure

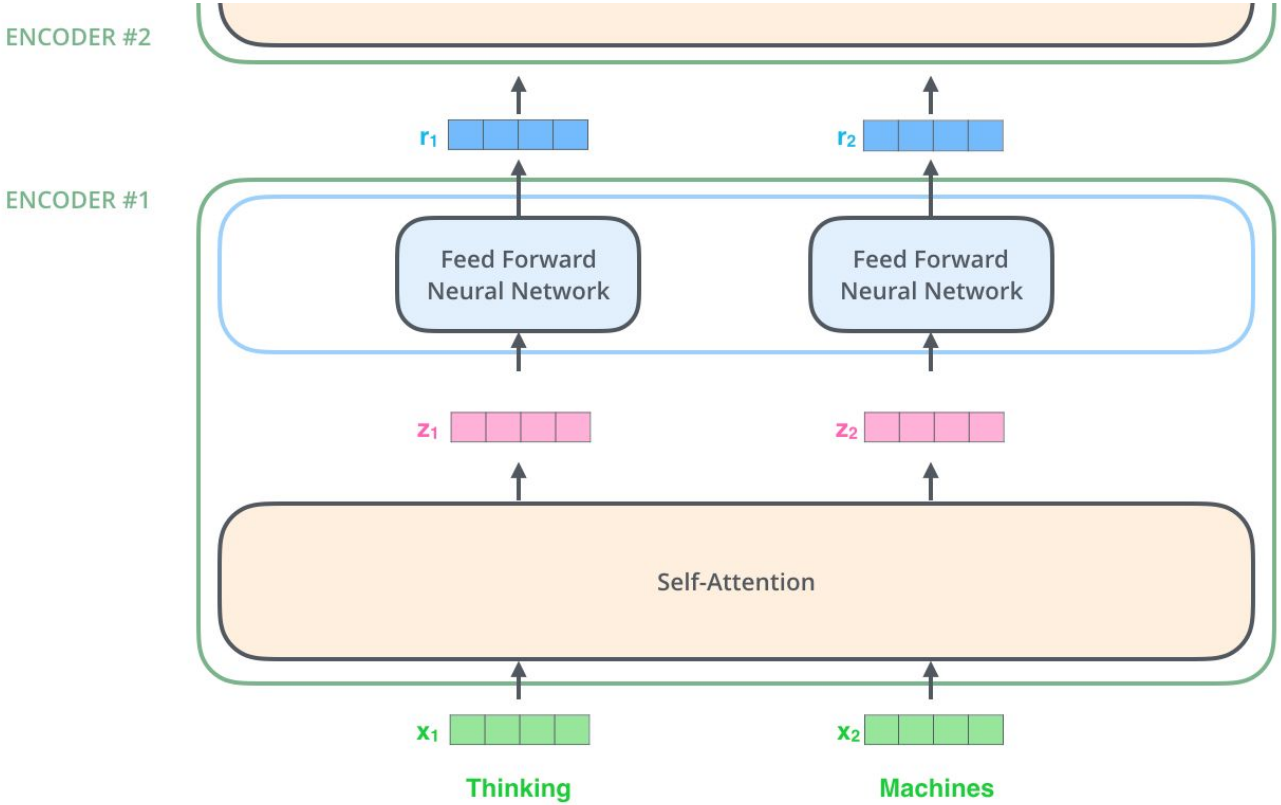
A. Vaswani et al. Attention Is All You Need.

Transformer



The illustrated transformer, https://jalammar.github.io/illustrated-transformer/.

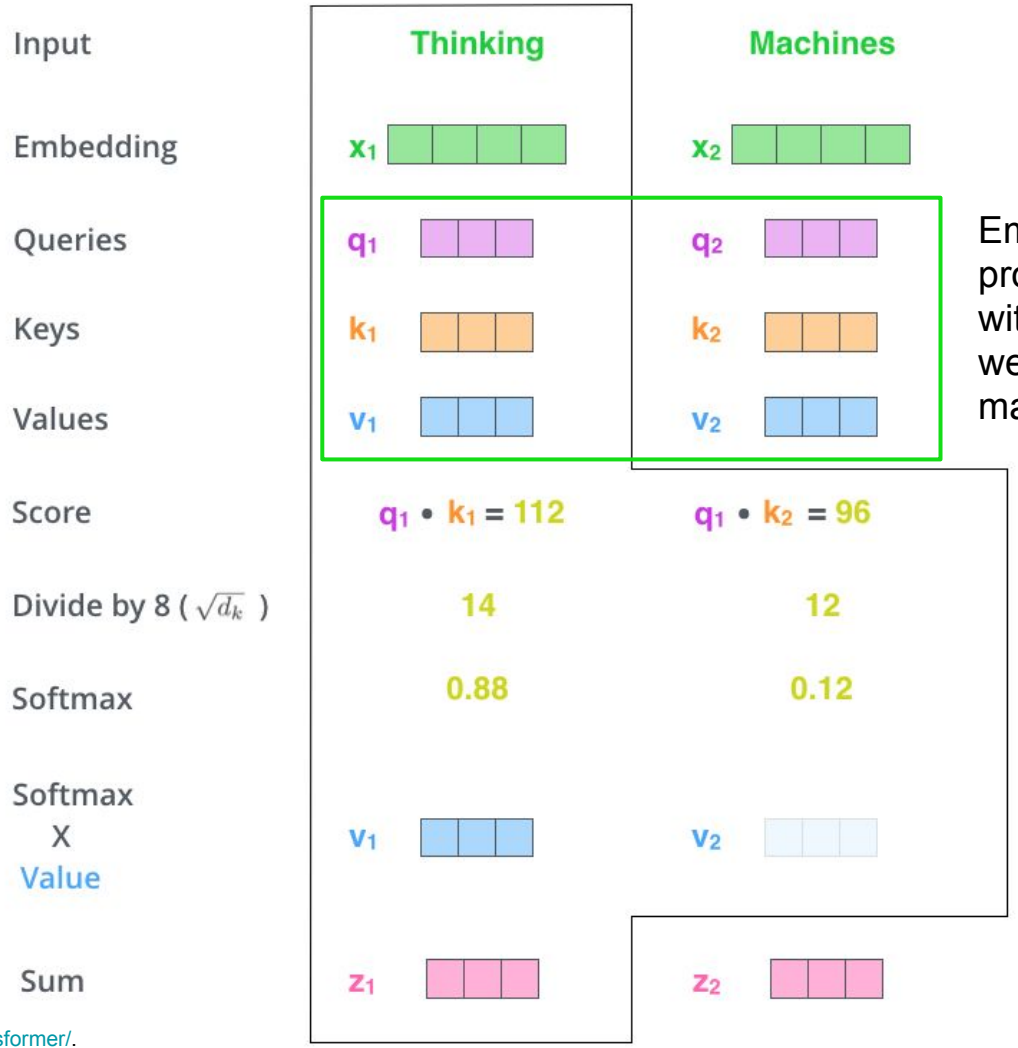
Transformer: encoder



The illustrated transformer, https://jalammar.github.io/illustrated-transformer/.

Transformer: self-attention

Input: $X = (x_1; x_2; \ldots; x_{max\_num\_words})$,
$X \in R^{max\_num\_words \times embedding\_dim}$

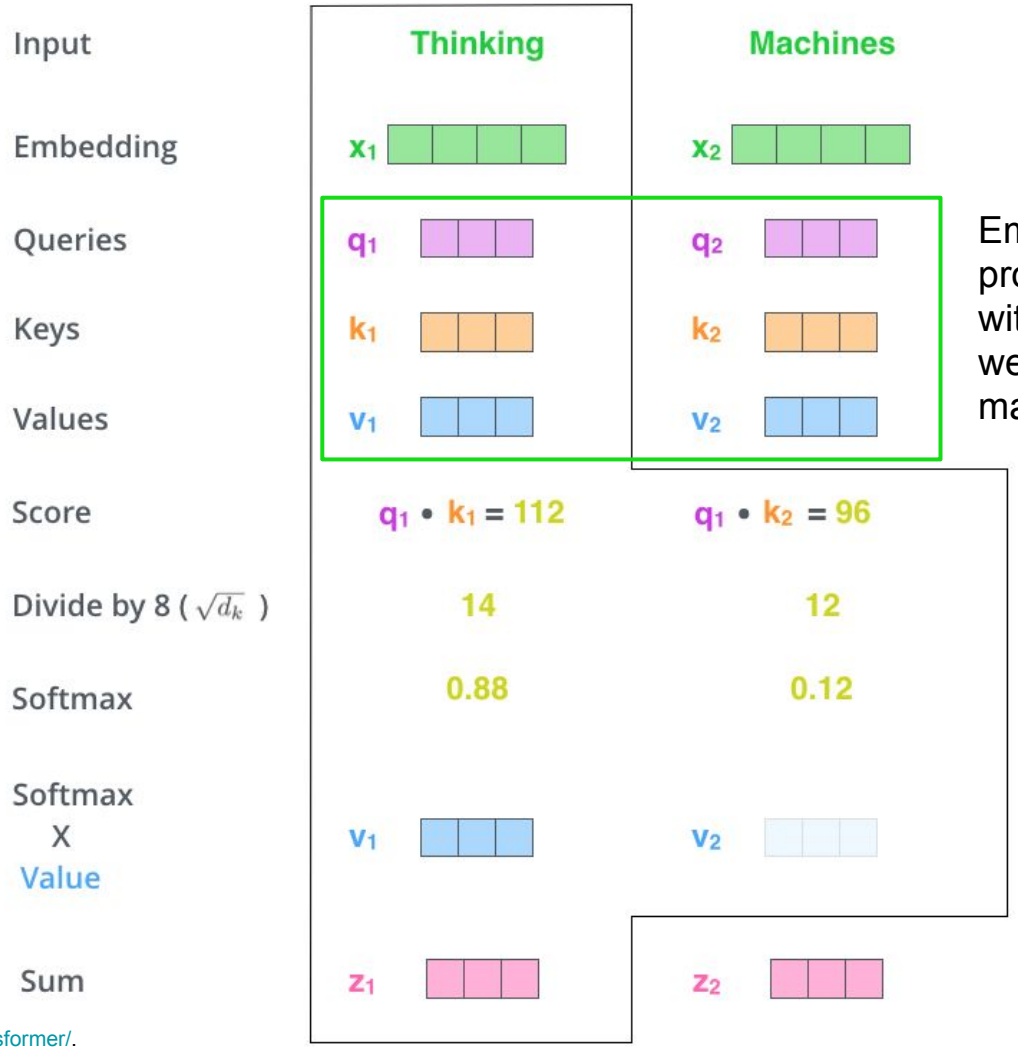


Embedding projections with learnable weights (just matmul)

The illustrated transformer, https://jalammar.github.io/illustrated-transformer/.

Transformer: self-attention

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



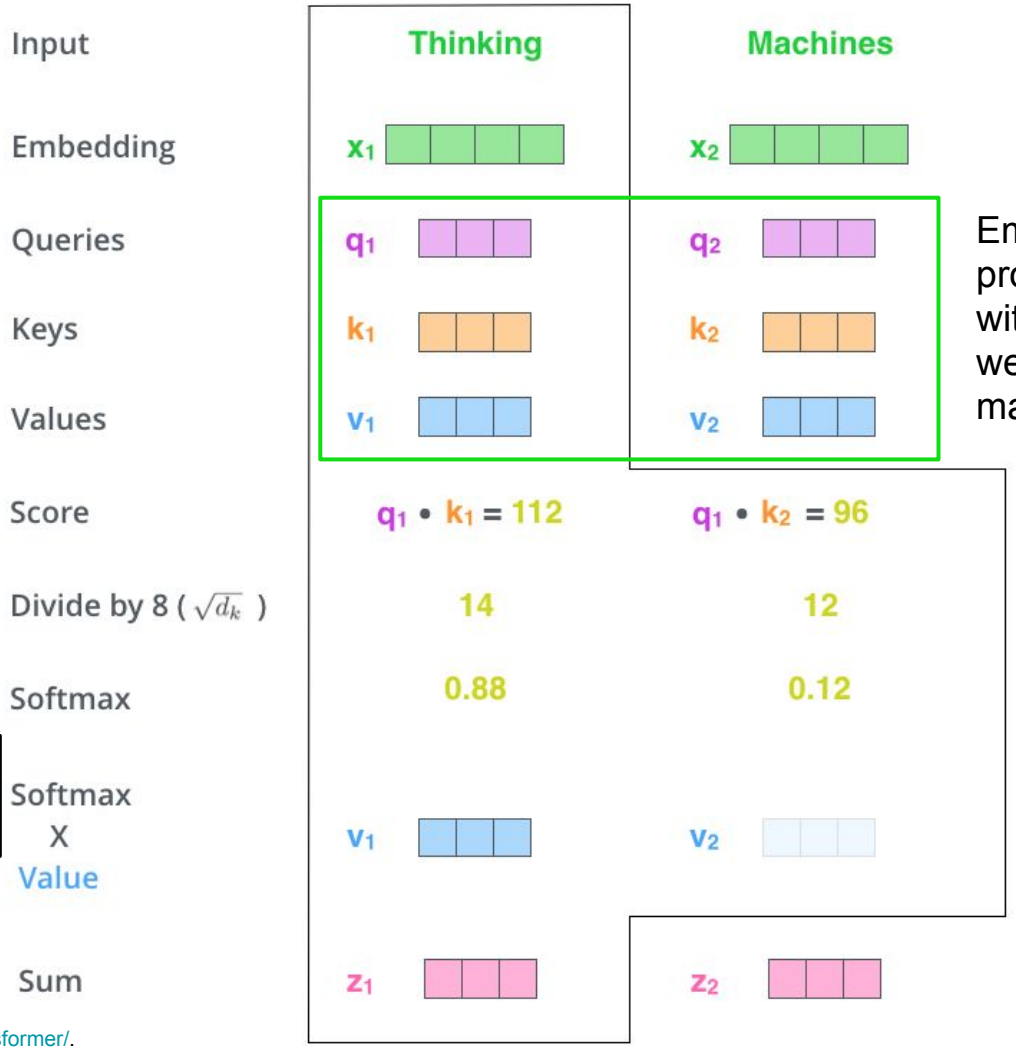
Embedding projections with learnable weights (just matmul)

The illustrated transformer, https://jalammar.github.io/illustrated-transformer/.

Transformer: self-attention

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

It is bidirectional.



Embedding projections with learnable weights (just matmul)

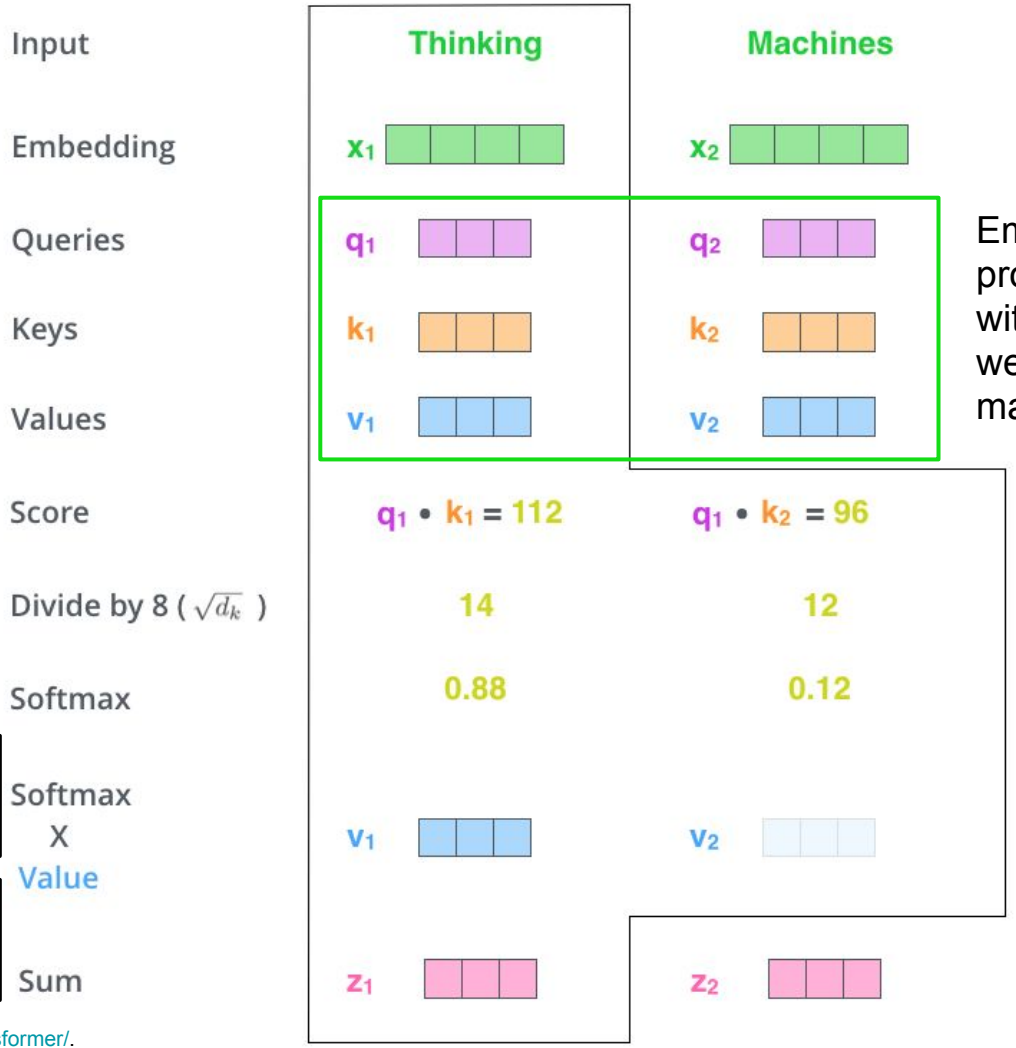
The illustrated transformer, https://jalammar.github.io/illustrated-transformer/.

Transformer: self-attention

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

It is bidirectional.

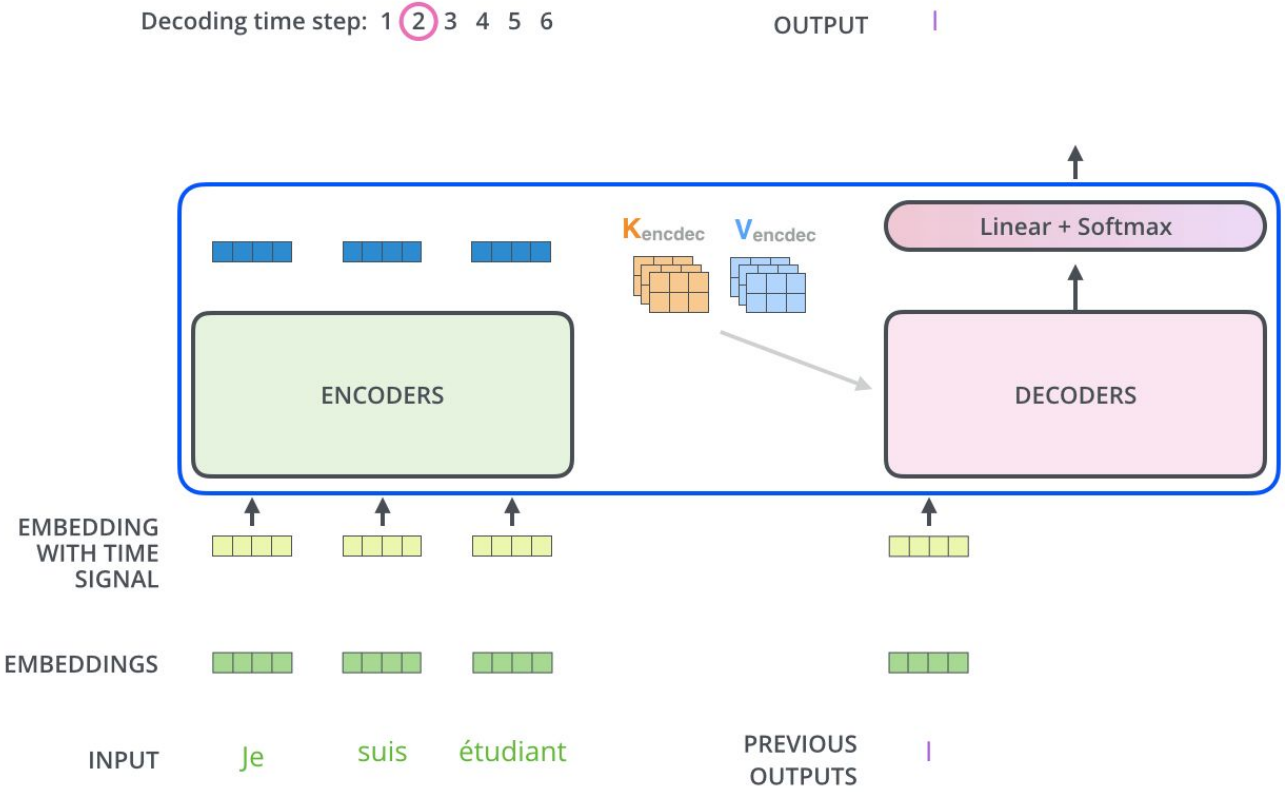
It is multi-headed.



Embedding projections with learnable weights (just matmul)

The illustrated transformer, https://jalammar.github.io/illustrated-transformer/.

Transformer: decoder



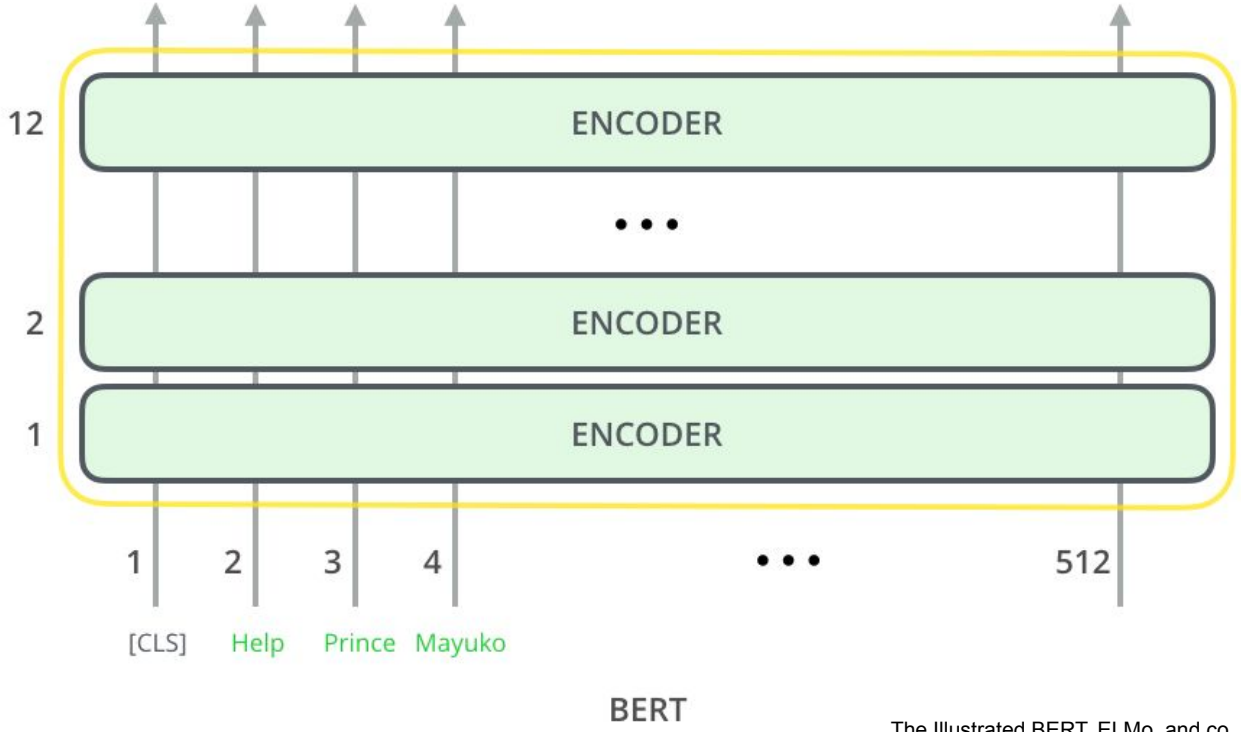
The illustrated transformer, https://jalammar.github.io/illustrated-transformer/.

BERT: Bidirectional Encoder Representations from Transformers

How many transformers in BERT?

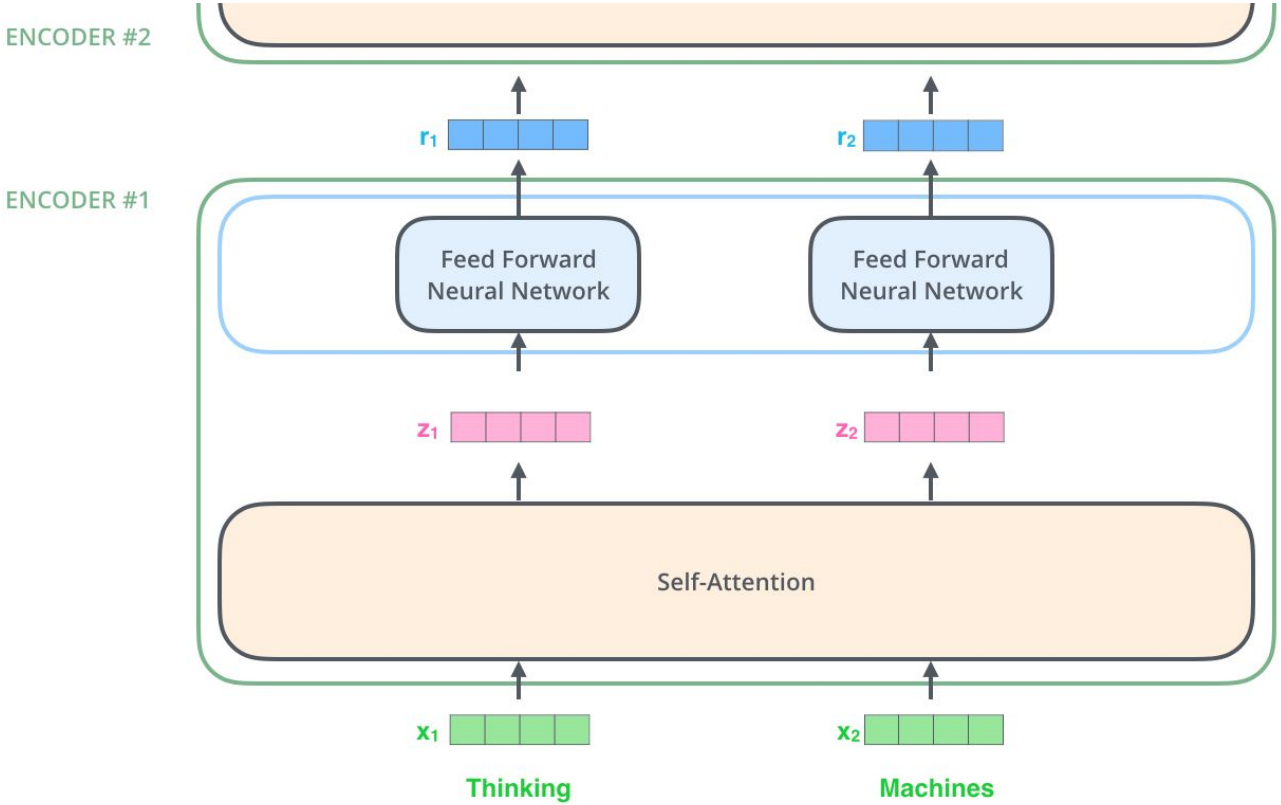
38 parrots, YouTube.

BERT: Bidirectional Encoder Representations from Transformers



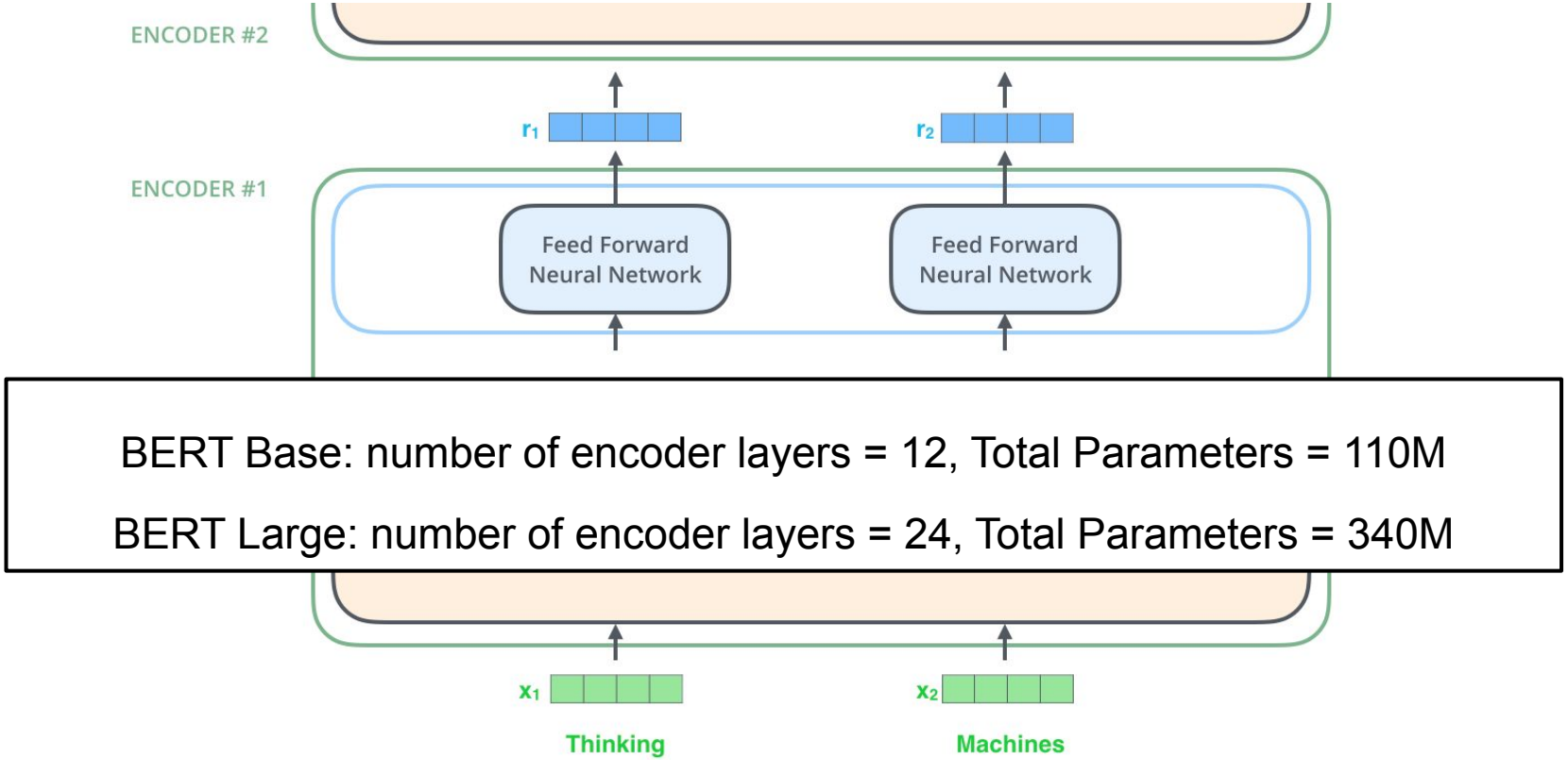
The Illustrated BERT, ELMo, and co, http://jalammar.github.io/illustrated-bert/.

Transformer: encoder



The illustrated transformer, https://jalammar.github.io/illustrated-transformer/.

Transformer: encoder



BERT Base: number of encoder layers = 12, Total Parameters = 110M

BERT Large: number of encoder layers = 24, Total Parameters = 340M

The illustrated transformer, https://jalammar.github.io/illustrated-transformer/.

BERT text mapping example

Tokenizer: split text into subwords.

```
from transformers import BertTokenizer, RobertaTokenizer
if __name__ == '__main__':
    sentence = 'On the way to Malaysia...no internet access to Twit'
    for tokenizer_class, tokenizer_name in zip(
            [BertTokenizer, RobertaTokenizer],
            ['bert-base-uncased', 'roberta-base']):
        tokenizer = tokenizer_class.from_pretrained(tokenizer_name)
        tokens = tokenizer.tokenize(sentence)
        print('{}:'.format(tokenizer_name))
        print(tokens)
bert-base-uncased:
['on', 'the', 'way', 'to', 'malaysia', '.', '.', '.', 'no', 'internet', 'access', 'to', 't', '##wi', '##t']
roberta-base:
['On', 'Ġthe', 'Ġway', 'Ġto', 'ĠMalaysia', '...', 'no', 'Ġinternet', 'Ġaccess', 'Ġto', 'ĠTw', 'it']
```

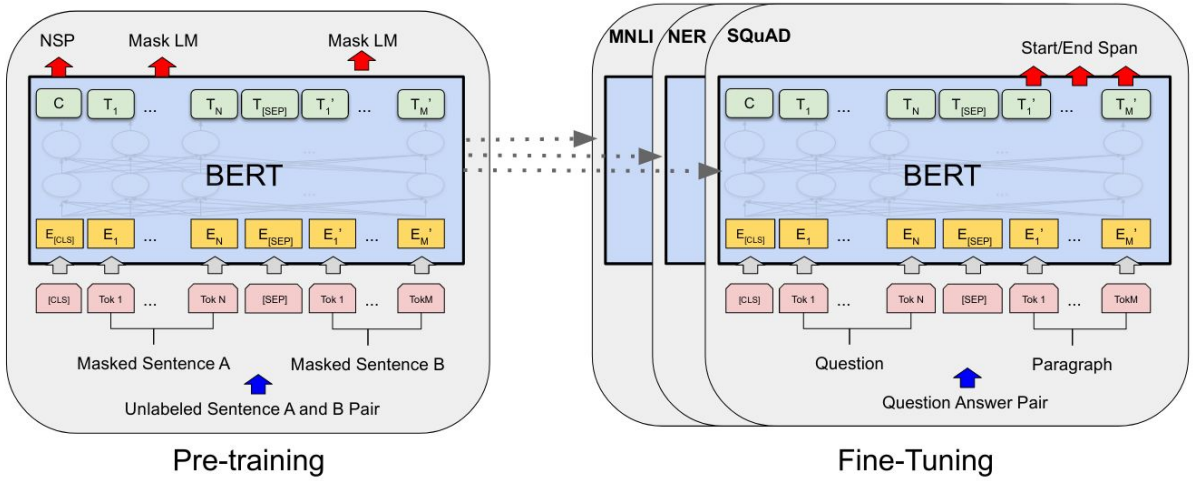
BERT: pre-training

Masked language modeling:

- Mask some percentage of the input tokens at random, and then predict those masked tokens.

Next sentence prediction:

- Understand the relationship between two sentences. Binary classification if in a given sentences pair the second sentence follows the first.



J. Devlin et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.

BERT: fine-tuning

```
class TweetModel(transformers.BertPreTrainedModel):
    def __init__(self, conf):
        super().__init__(conf)
        self.bert = transformers.BertModel.from_pretrained('bert-base-uncased', config=conf)
        self.drop_out = nn.Dropout(0.1)
        self.l0 = nn.Linear(768 * 2, 2)

    def forward(self, tokens_ids, mask, tokens_type_ids):
        last_hidden_state, pooler_output, hidden_states = self.bert(
            tokens_ids,
            attention_mask=mask,
            token_type_ids=tokens_type_ids
        )
```

BERT: fine-tuning

```
class TweetModel(transformers.BertPreTrainedModel):
    def __init__(self, conf):
        super().__init__(conf)
        self.bert = transformers.BertModel.from_pretrained('bert-base-uncased', config=conf)
        self.drop_out = nn.Dropout(0.1)
        self.l0 = nn.Linear(768 * 2, 2)

    def forward(self, tokens_ids, mask, tokens_type_ids):
        last_hidden_state, pooler_output, hidden_states = self.bert(
            tokens_ids,
            attention_mask=mask,
            token_type_ids=tokens_type_ids
        )
```

Input text tokens
(batch_size x max_tokens_num)

BERT: fine-tuning

```
class TweetModel(transformers.BertPreTrainedModel):
    def __init__(self, conf):
        super().__init__(conf)
        self.bert = transformers.BertModel.from_pretrained('bert-base-uncased', config=conf)
        self.drop_out = nn.Dropout(0.1)
        self.l0 = nn.Linear(768 * 2, 2)

    def forward(self, tokens_ids, mask, tokens_type_ids):
        last_hidden_state, pooler_output, hidden_states = self.bert(
            tokens_ids,
            attention_mask=mask,
            token_type_ids=tokens_type_ids
        )
```

Mask with 1 at positions of real tokens and 0 for padding
(batch_size x max_tokens_num)

BERT: fine-tuning

```
class TweetModel(transformers.BertPreTrainedModel):
    def __init__(self, conf):
        super().__init__(conf)
        self.bert = transformers.BertModel.from_pretrained('bert-base-uncased', config=conf)
        self.drop_out = nn.Dropout(0.1)
        self.l0 = nn.Linear(768 * 2, 2)

    def forward(self, tokens_ids, mask, tokens_type_ids):
        last_hidden_state, pooler_output, hidden_states = self.bert(
            tokens_ids,
            attention_mask=mask,
            token_type_ids=tokens_type_ids
        )
```

Mask to identify different input sequences with 0 for 1st and 1 for 2nd sequence (batch_size x max_tokens_num)

BERT: fine-tuning

```
class TweetModel(transformers.BertPreTrainedModel):
    def __init__(self, conf):
        super().__init__(conf)
        self.bert = transformers.BertModel.from_pretrained('bert-base-uncased', config=conf)
        self.drop_out = nn.Dropout(0.1)
        self.l0 = nn.Linear(768 * 2, 2)

    def forward(self, tokens_ids, mask, tokens_type_ids):
        last_hidden_state, pooler_output, hidden_states = self.bert(
            tokens_ids,
            attention_mask=mask,
            token_type_ids=tokens_type_ids
        )
```

Last encoder output
(batch_size x max_tokens_num x embedding_dim)

BERT: fine-tuning

```
class TweetModel(transformers.BertPreTrainedModel):
    def __init__(self, conf):
        super().__init__(conf)
        self.bert = transformers.BertModel.from_pretrained('bert-base-uncased', config=conf)
        self.drop_out = nn.Dropout(0.1)
        self.l0 = nn.Linear(768 * 2, 2)

    def forward(self, tokens_ids, mask, tokens_type_ids):
        last_hidden_state, pooler_output, hidden_states = self.bert(
            tokens_ids,
            attention_mask=mask,
            token_type_ids=tokens_type_ids
        )
```

Processed last encoder output for the 1st token
(batch_size x embedding_dim)

BERT: fine-tuning

```python
class TweetModel(transformers.BertPreTrainedModel):
    def __init__(self, conf):
        super().__init__(conf)
        self.bert = transformers.BertModel.from_pretrained('bert-base-uncased', config=conf)
        self.drop_out = nn.Dropout(0.1)
        self.l0 = nn.Linear(768 * 2, 2)

    def forward(self, tokens_ids, mask, tokens_type_ids):
        last_hidden_state, pooler_output, hidden_states = self.bert(
            tokens_ids,
            attention_mask=mask,
            token_type_ids=tokens_type_ids
        )
```

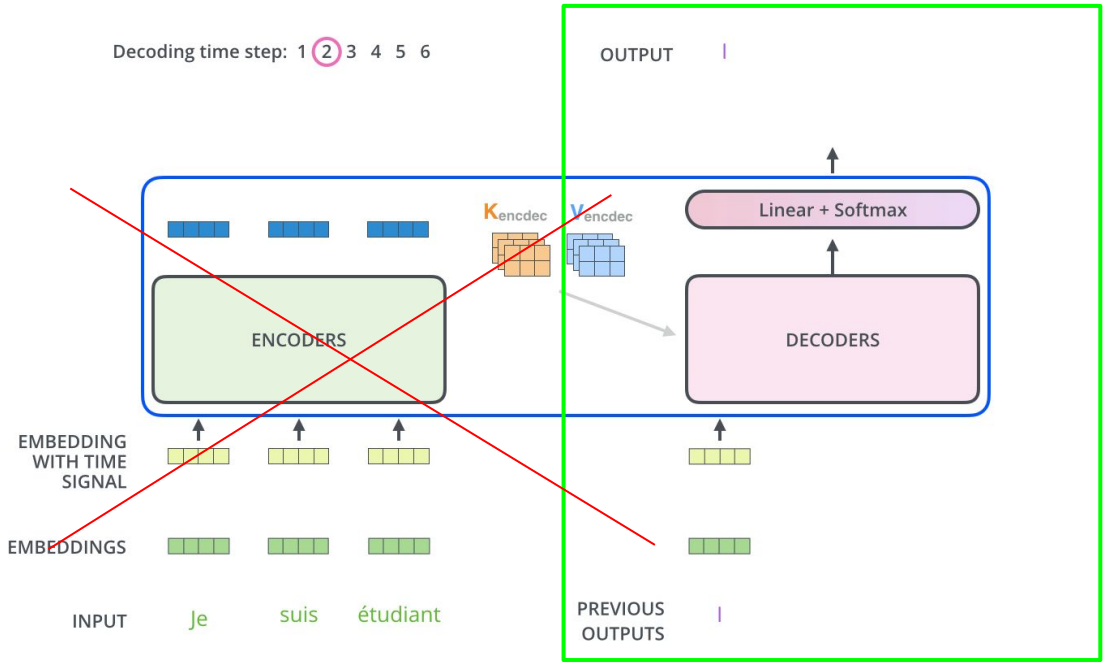
Output from embeddings and all encoders
Tuple of tensors (batch_size x max_tokens_num x embedding_dim)

GPT: Improving Language Understanding by Generative Pre-Training

GPT: Improving Language Understanding by Generative Pre-Training

It is a decoder from transformer.

Try it out (GPT-2): https://demo.allennlp.org/next-token-lm?text=Lobachevsky

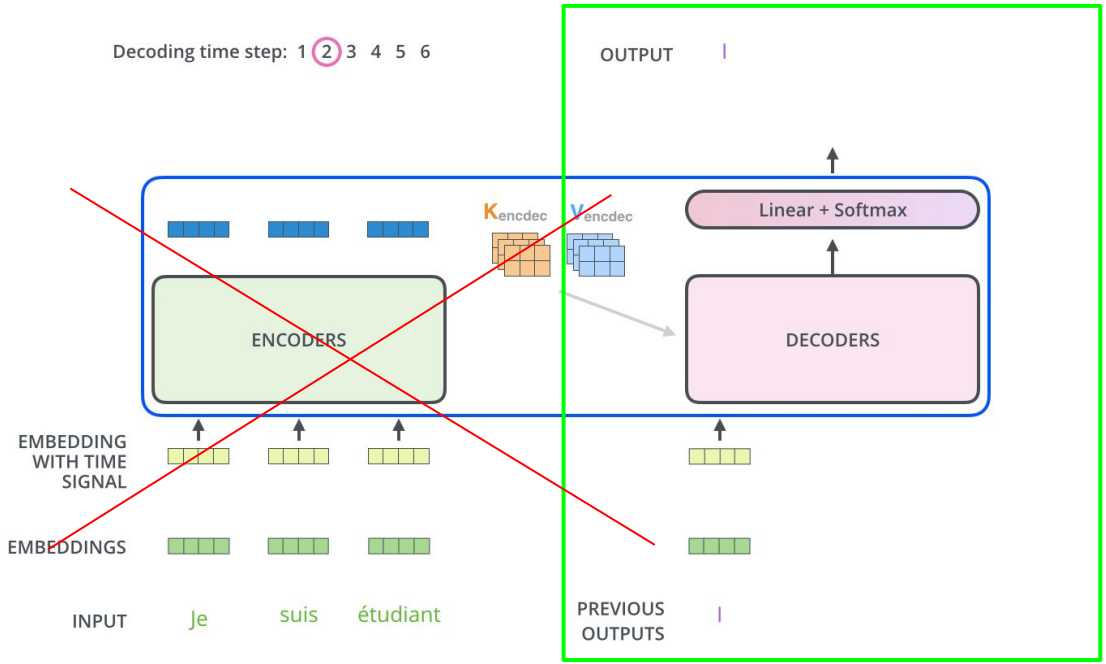


GPT: Improving Language Understanding by Generative Pre-Training

It is a decoder from transformer.

Try it out (GPT-2): https://demo.allennlp.org/next-token-lm?text=Lobachevsky

GPT-3 has 175B parameters.



Summary

- Seq2Seq: encode all input text into single context vector.
- Attention: make context vector adaptive to position of output word.
- Self-attention: shorten distance between words.
- BERT: large scale pre-training.

Thank you for the attention!

- https://towardsdatascience.com/lost-in-translation-found-by-transformer-46a16bf6418f
- https://towardsdatascience.com/bert-for-dummies-step-by-step-tutorial-fb90890ffe03
- https://mccormickml.com/2019/07/22/BERT-fine-tuning/
- https://www.analyticsvidhya.com/blog/2019/06/understanding-transformers-nlp-state-of-the-art-models/
- http://jalammar.github.io/illustrated-gpt2/
- https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html
- https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html
- https://ai.googleblog.com/2020/03/more-efficient-nlp-model-pre-training.html
- https://huggingface.co/transformers/