week 1:

neural machine translation (NMT)

seq2seq

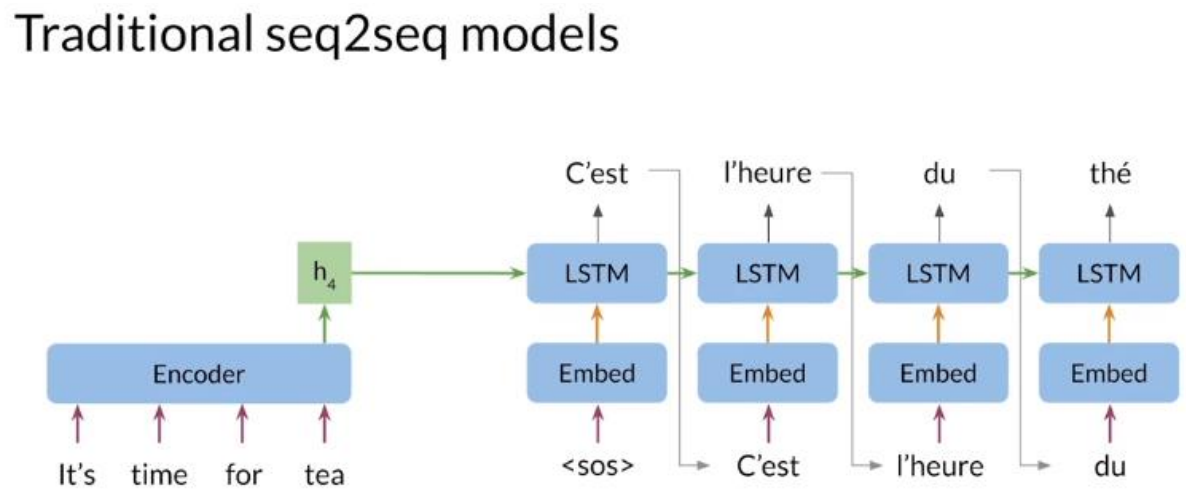
- maps variable-length sequences to fixed-length memory

- inputs and outputs can have different lengths

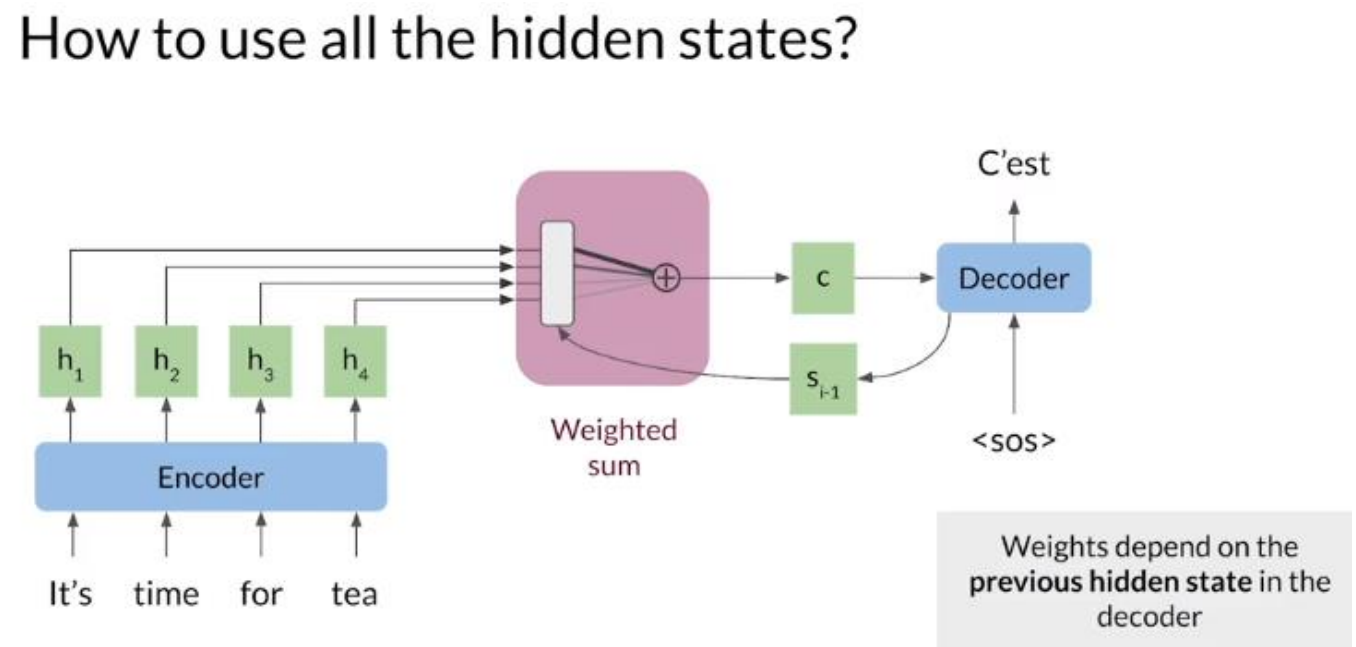
- LSTMs and GRUs to avoid vanishing and exploding gradient problems

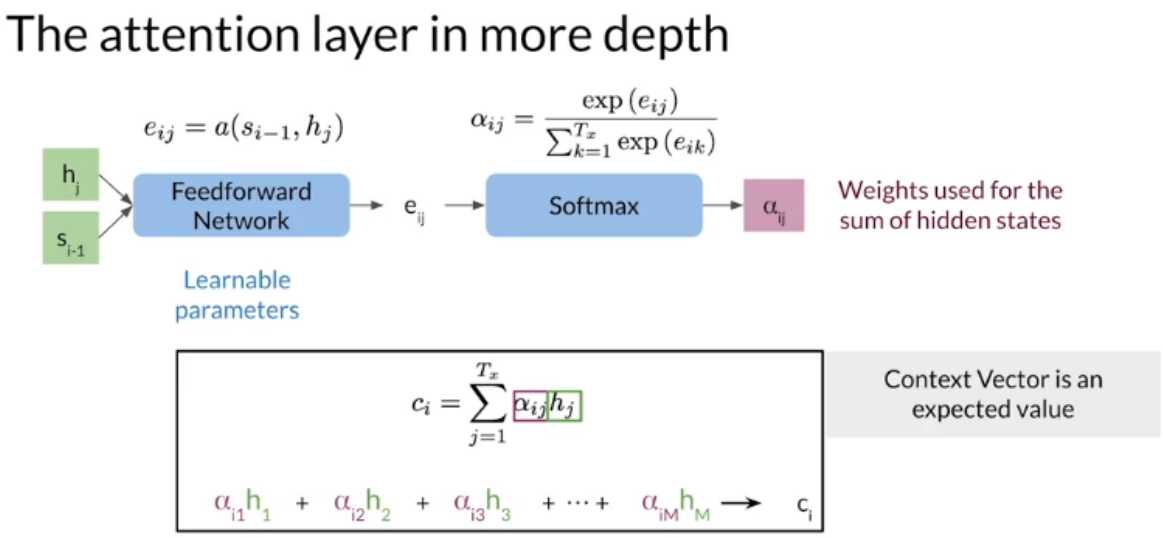
encoder - decoder

Information bottleneck: a fixed amount of information goes from the encoder to the decoder



Seq2seq model with attention





The sequential nature of models you learned in the previous course (RNNs, LSTMs, GRUs) does not allow for parallelization within training examples, which becomes critical at longer sequence lengths, as memory constraints limit batching across examples

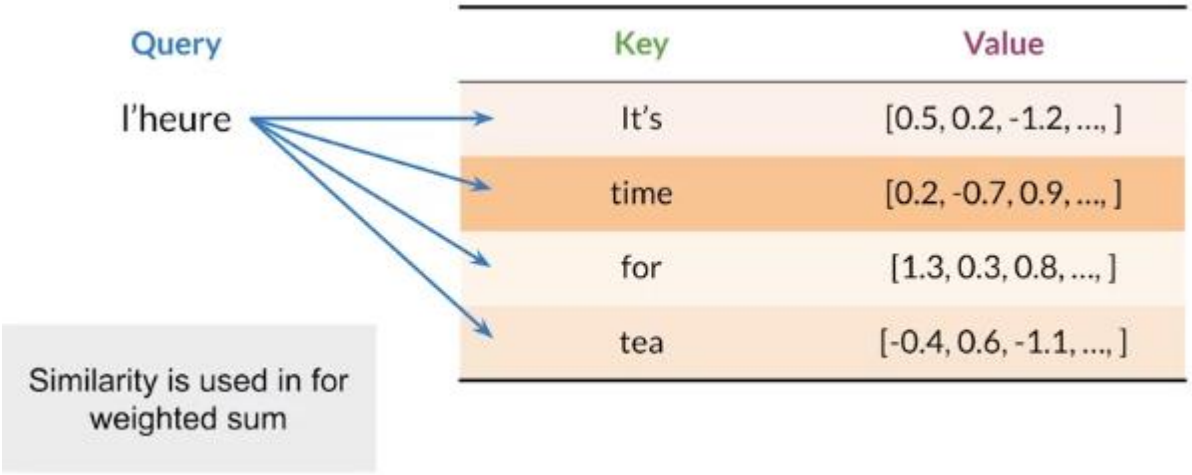
if you rely on sequences and you need to know the beginning of a text before being able to compute something about the ending of it, then you can not use parallel computing. You would have to wait until the initial computations are complete. This is not good, because if your text is too long;

- it will take a long time for you to process it
- you will lose a good amount of information mentioned earlier in the text as you approach the end.

Queries, Keys, Values, and Attention

There have been multiple variations on attention with some models don't rely on RNN

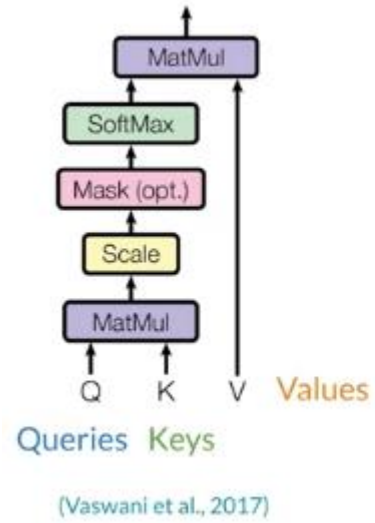
Retrieving information using Query, Key, Value introduced in 2017 paper Attention is all you need



Quetry, key and value are represented by embedding vectors

The similarity between words is called alignment

Scaled dot-product attention



(Vaswani et al., 2017)

$$\text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

The queries for each step are packed together into a matrix Q, so attention can be computed simultaneously for each query

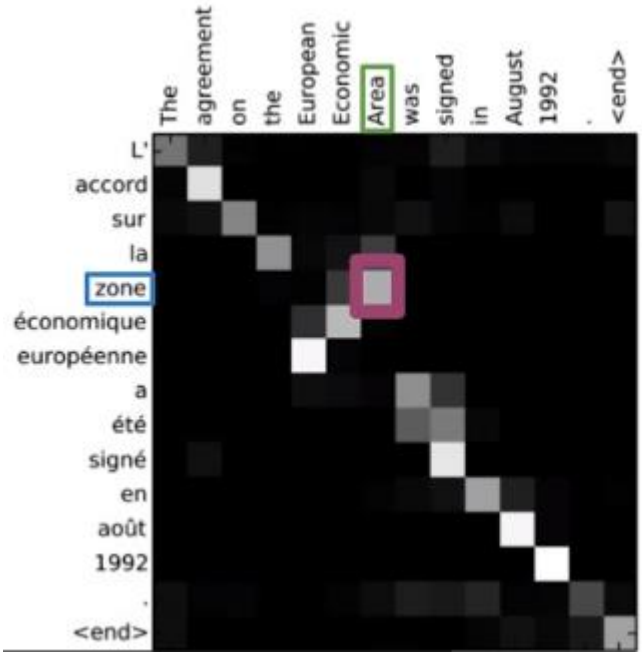
The keys and values are also packed into matrices K and V, you can think of the keys and values as being the same

1. Queries and keys are multiplied together to get a matrix of alignments course (similarity between Q and K)
2. Then scaled by the square root of the key vector dimension, improves performance for larger model sizes, seen as a regularization constant
3. Scale scores are converted to weights using the SoftMax function
4. Weights and value matrices are multiplied to get the attention vectors for each query

Just two matrix multiplications and a SoftMax, no neural networks, much faster to compute, but means the alignments between the source (key) and target (query) languages must be learned elsewhere

Alignment is learned in the input embeddings or in other linear layers before the attention layer

Alignment between languages

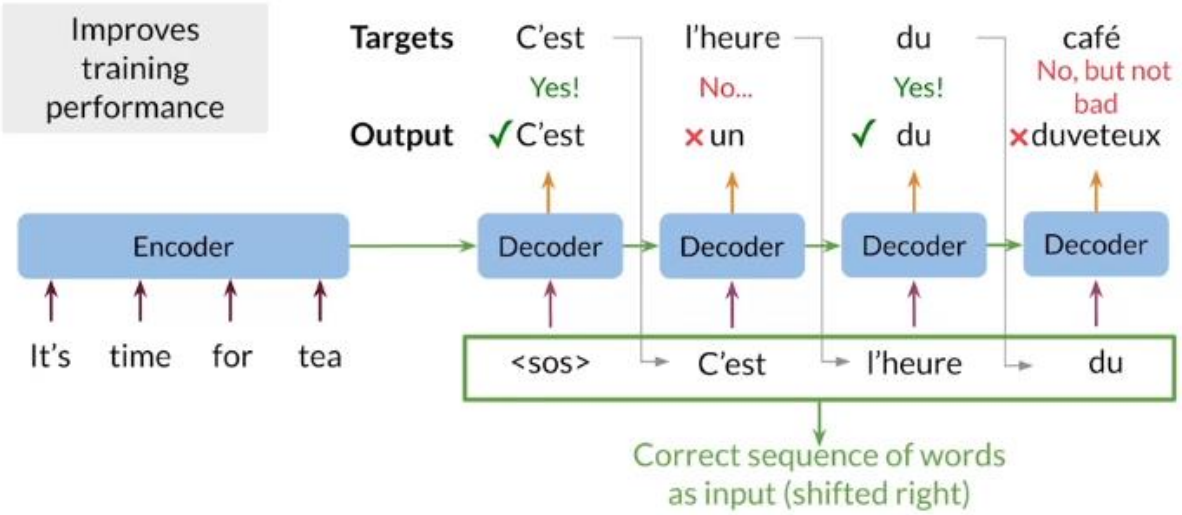


Bahdanau et al., 2015

Similar words have large weights

Flexible attention: works for languages with different grammar structures

Teacher Forcing



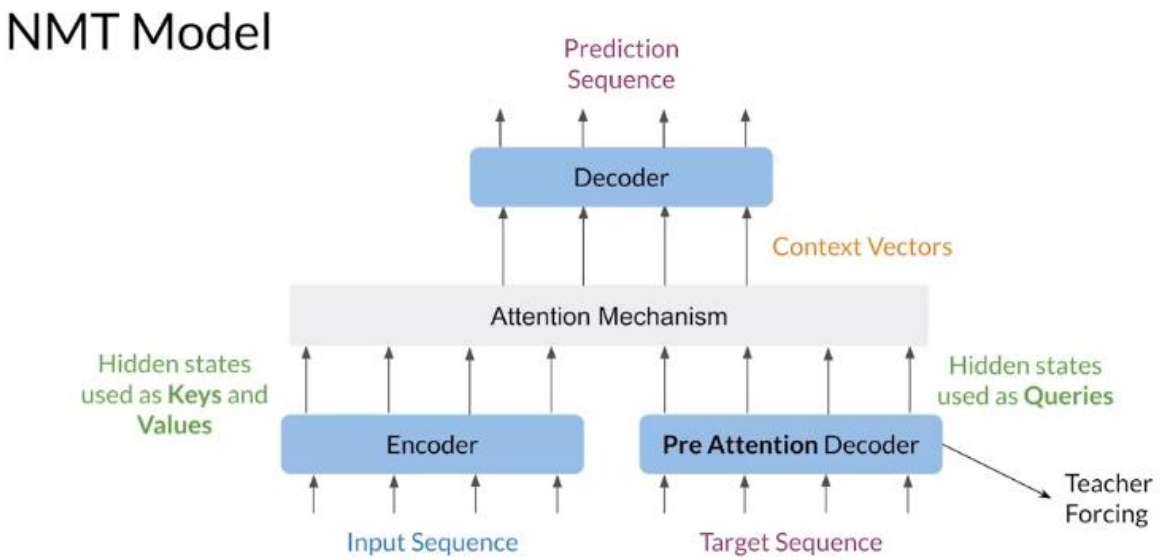
curriculum learning: Slowly start using decoder outputs over time, so that leads into training you are no longer feeding in the target words

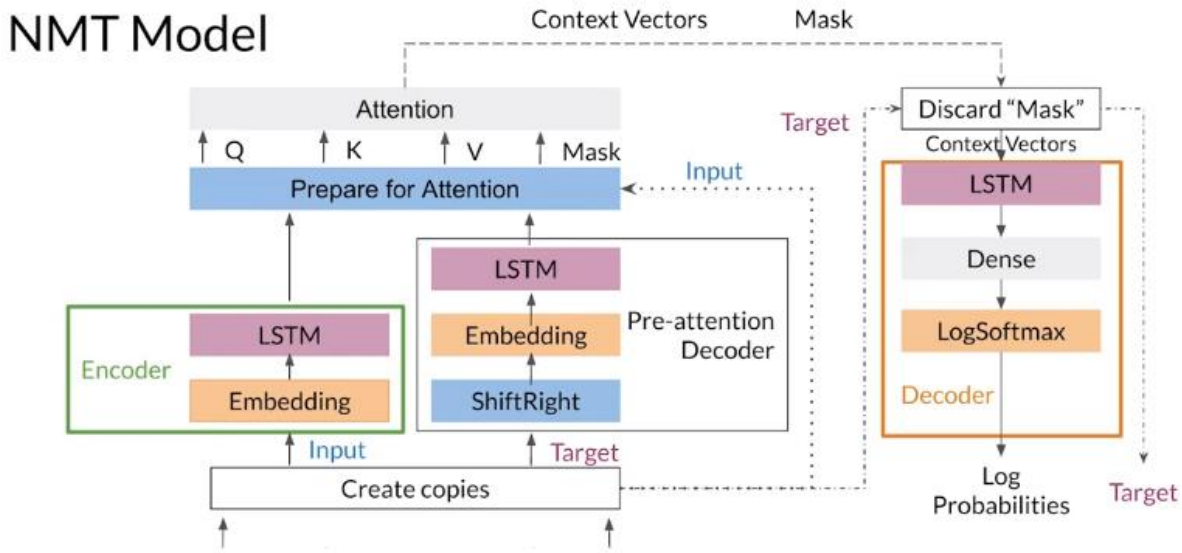
MNT Model with Attention

The decoder must pass the hidden state to the Attention Mechanism, difficult to implement, so a pre-attention decoder is introduced

Using 2 decoders:

- Pre-attention decoder to provide hidden states
- Post-attention decoder provides the translation





Shift Right: shift each sequence to the right and add start token

BLEU Score (Bilingual Evaluation Understudy)

Compares candidate translations to reference (human) translations

Closer to 1, the better

Using N-Grams, below using unigrams

BLEU doesn't consider semantic meaning

BLUE doesn't consider sentence structure

BLEU Score

Candidate	I	I	am	I	
Reference 1	Younes	said	I	am	hungry
Reference 2	He	said	I	am	hungry

Count: $\frac{1+1+1+1}{4} = 1$

A model that always outputs common words will do great!

BLEU Score (Modified)

Candidate	I	I	am	I	
Reference 1	Younes	said			hungry
Reference 2	He	said			hungry

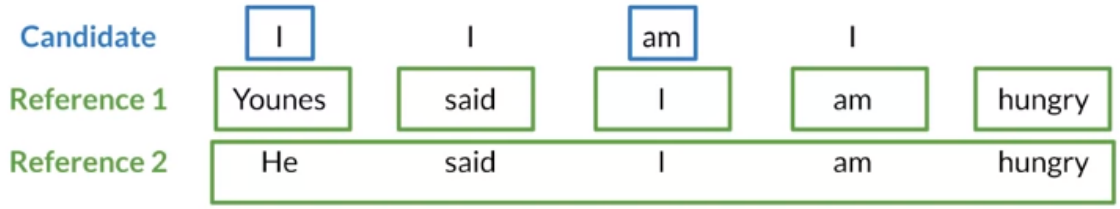
Count: $\frac{1+1}{4} = 0.5$

Better than the previous implementation version!

ROUGE-N Score (Recall-Oriented Understudy for Gisting Evaluation)

How many words from reference appear in the candidate translations

ROUGE-N



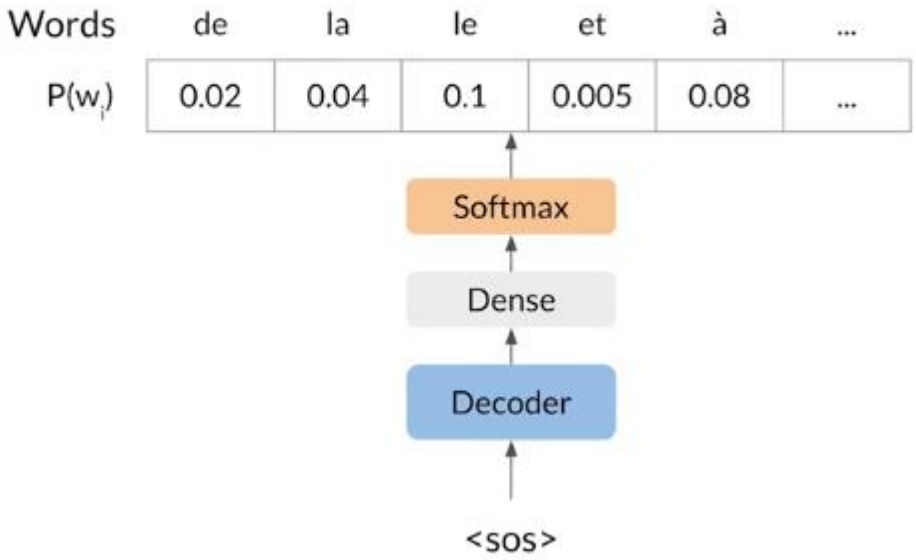
Count 1: $\frac{1+1}{5} = 0.4$ Count 2: $\frac{1+1}{5} = 0.4$

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \longrightarrow F1 = 2 \times \frac{\text{BLEU} \times \text{ROUGE-N}}{\text{BLEU} + \text{ROUGE-N}}$$

$$F1 = 2 \times \frac{0.5 \times 0.4}{0.5 + 0.4} = \frac{4}{9} \approx 0.44$$

Sampling and decoding

Seq2Seq model



Greedy decoding: selects the most probable word at each step, but the best word at each step may not be the best for longer sequences, can be fine for shorter sequences, but limited by inability to look further down the sequence

J'ai faim.

I am hungry .

I am, am, am, am...

Random sampling

Often a little too random for accurate translation

Solution: assign more weight to more probable words, and less weight to less probable words

Temperature:

- Scale 0-1
- Can control for more or less randomness in predictions
- Lower temperatures settings: more confident, conservative network
- Higher temperatures settings: more excited, random network

Beam Search

Most probable translation is not the one with the most probable word at each step

Solution: calculate probability of multiple possible sequences

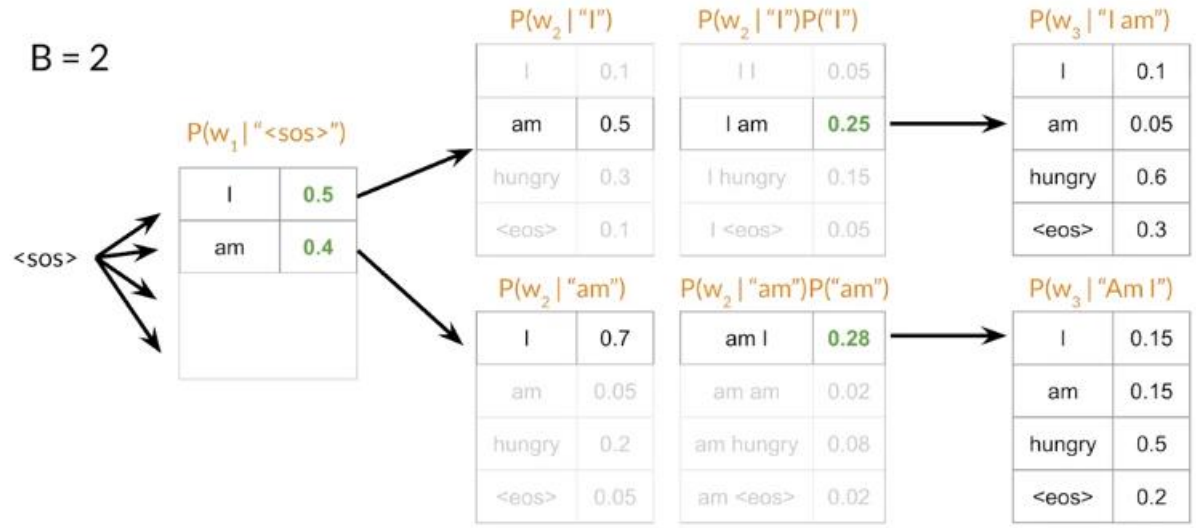
Beam width B determines number of sequences you keep

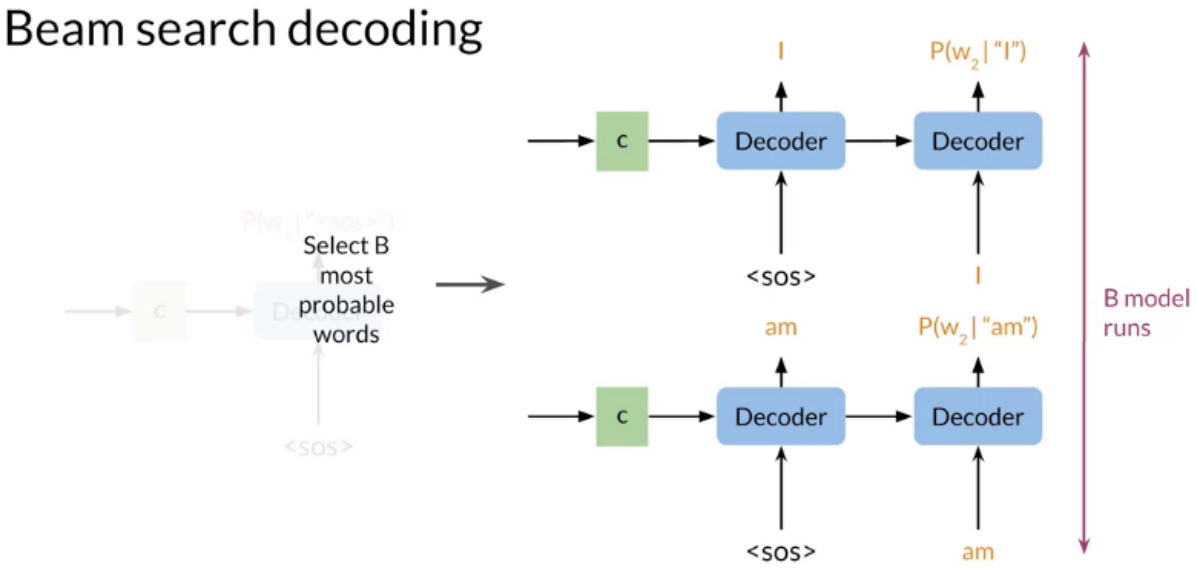
Until all B most probable sequences end with <EOS>

Beam search with B=1 is greedy decoding

Problems:

- Penalizes long sequences, so you should normalize by the sentence length
- Computationally expensive and consumes a lot of memory

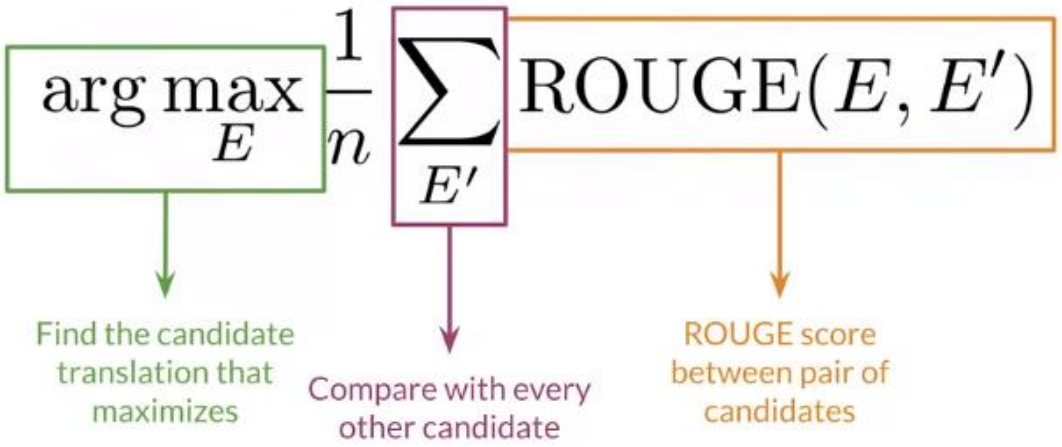




Minimum Bayes Risk (MBR)

- Generate several candidate translations
- Assign a similarity to every pair using a similarity score (such as ROUGE)
- Select the sample with the highest average similarity
- Better performance than random sampling and greedy decoding

Minimum Bayes Risk (MBR)



Week 2:

Transformers vs RNNs

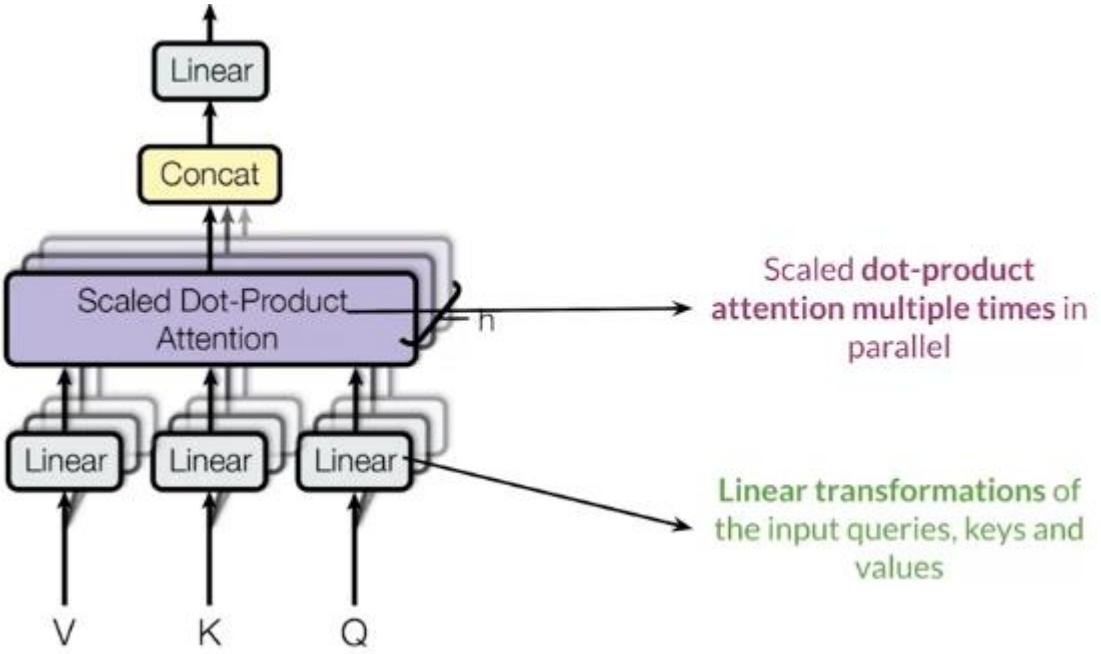
RNNs:

- No room for parallel
- Loss of information
- Vanishing gradient

Transformers:

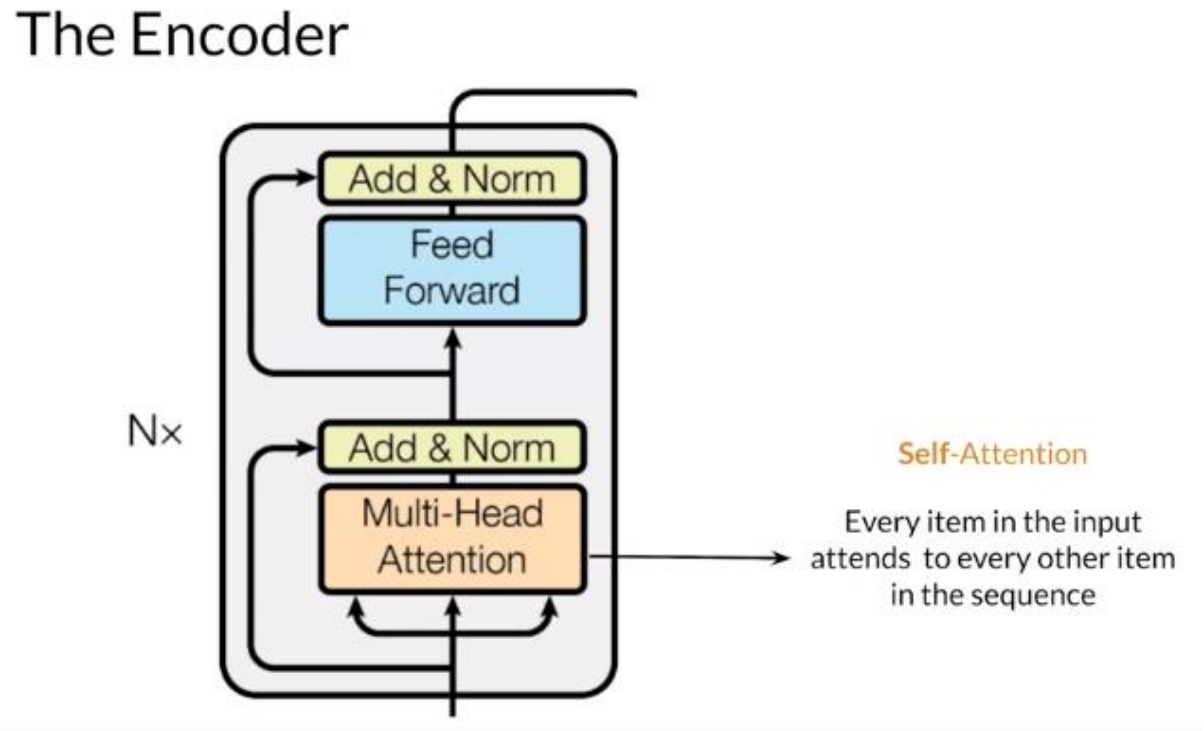
- Based on attention
- Don't require any sequential computation per layer, only a single step is needed
- Gradient steps that need to be taken from the last output to the first input in a transformer is just one.

Multi-Head Attention

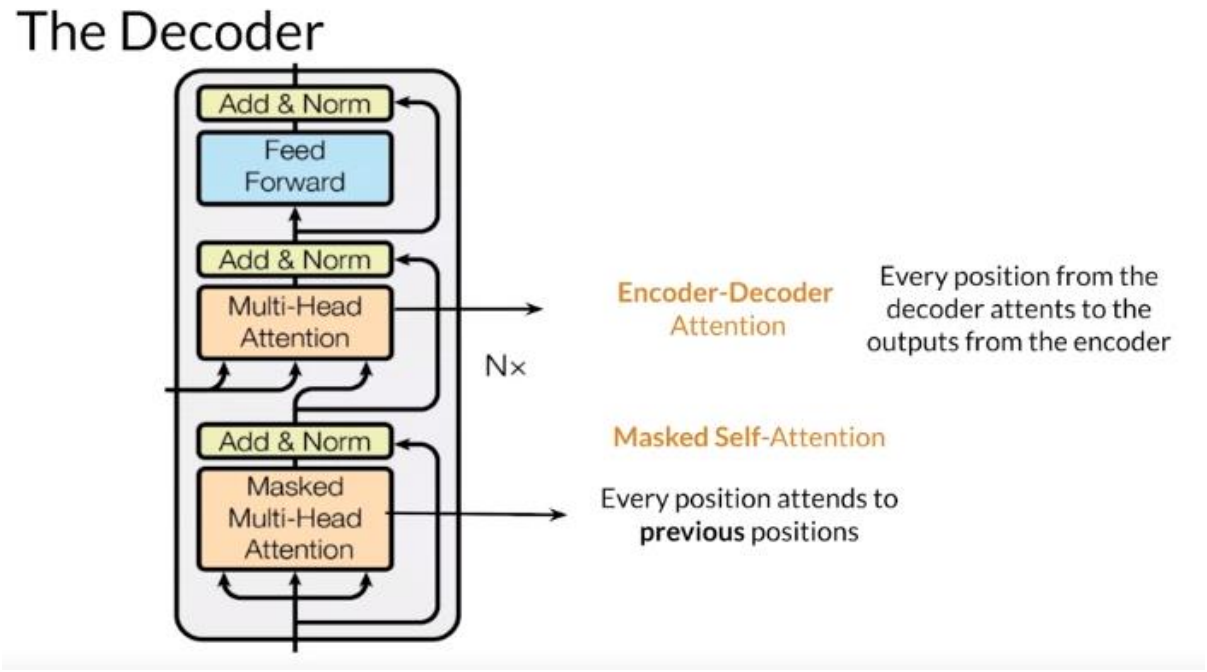


Linear transformations are learnable parameters

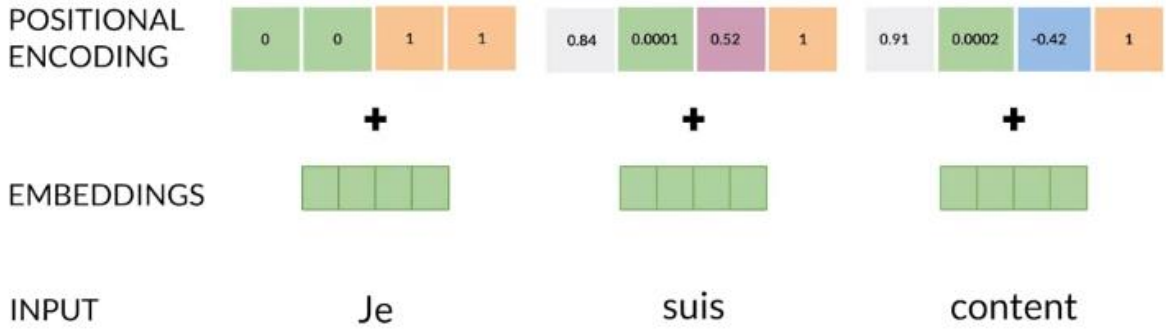
Self-Attention: every item in the input attends to every other item in the sequence



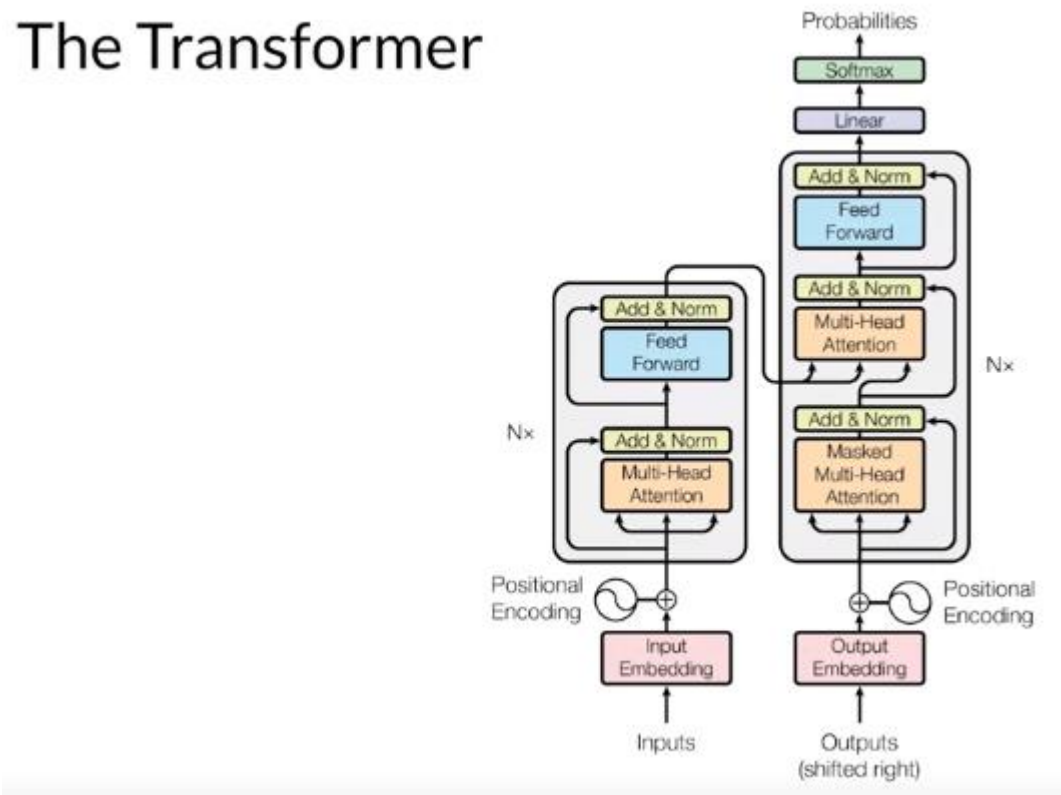
Masked Self-Attention: every position attends to previous positions



Positional Encoding



The Transformer

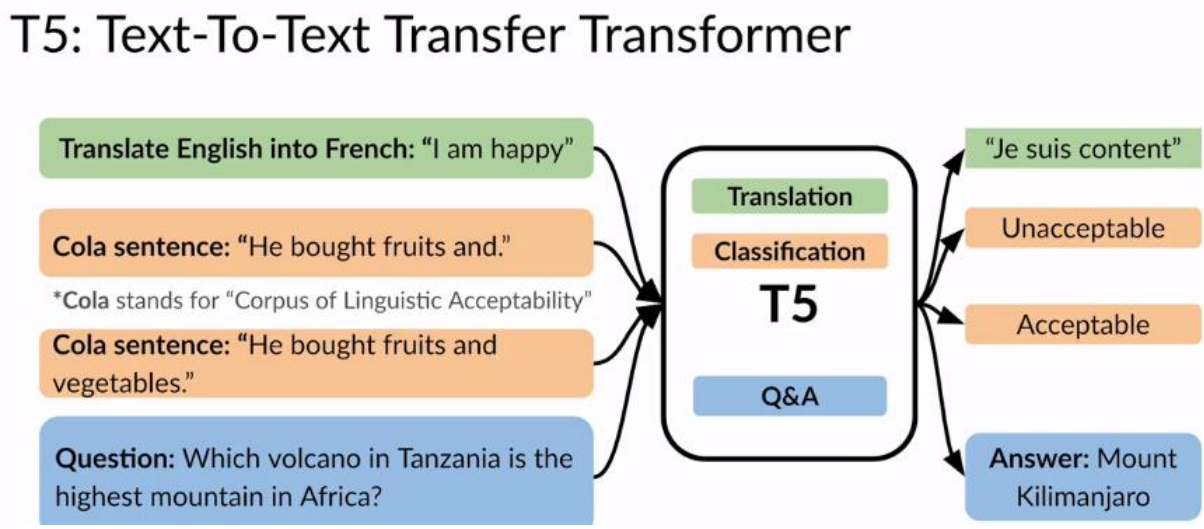


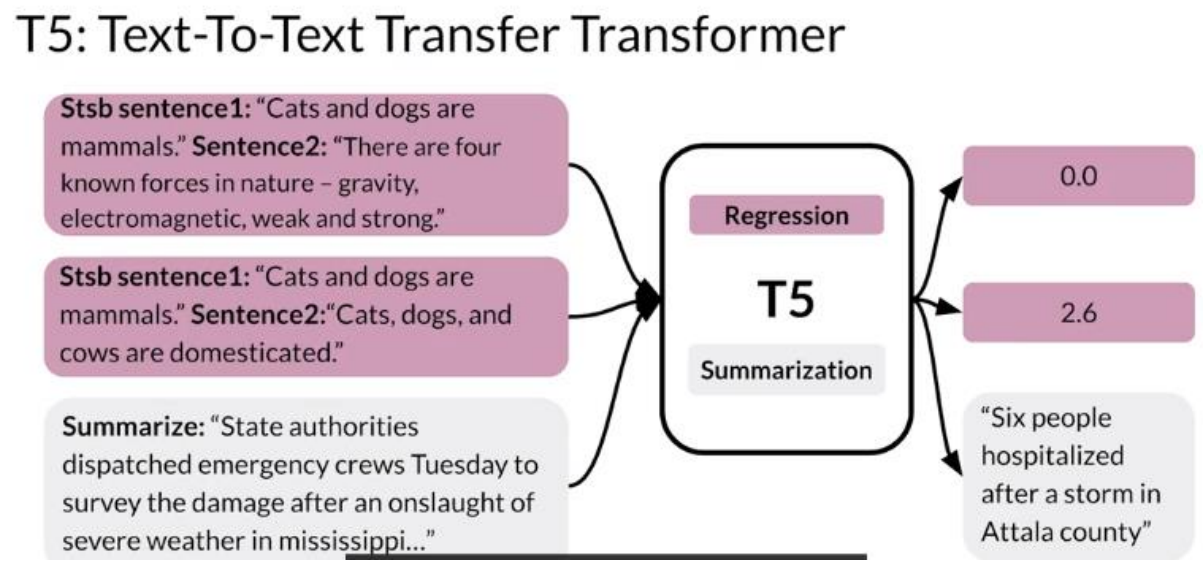
Transformer Application:

- Text summarization
- Auto-complete
- Named entity recognition (NER)
- Question answering (Q&A)
- Translation
- Chat-bots
- Other NLP tasks
- Sentiment Analysis
- Market Intelligence
- Text Classification
- Character Recognition
- Speel Checking

State of the Art Transformers

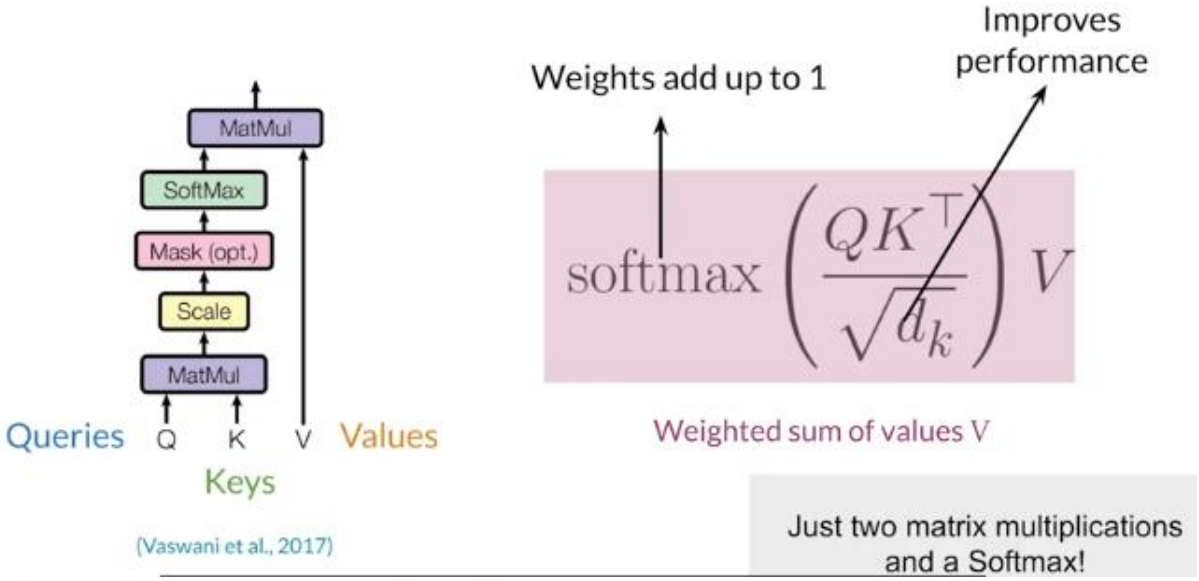
- GPT-2: Generative pre-training for Transformer
- BERT: Bidirectional Encoder Representation from Transformers
- T5: Text-to-text transfer transformer

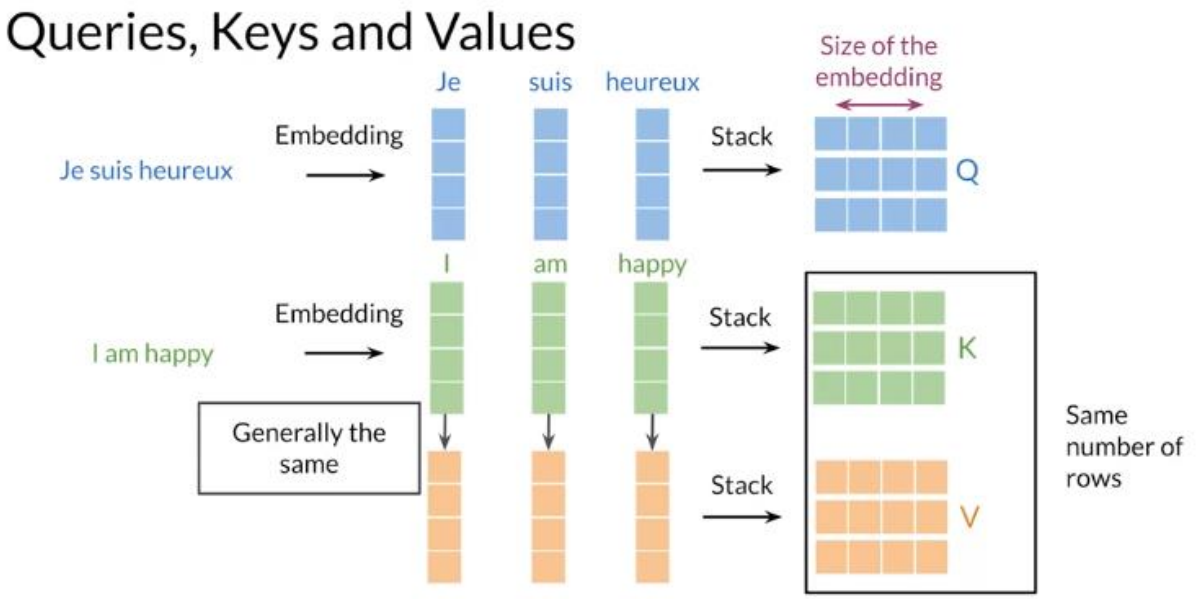


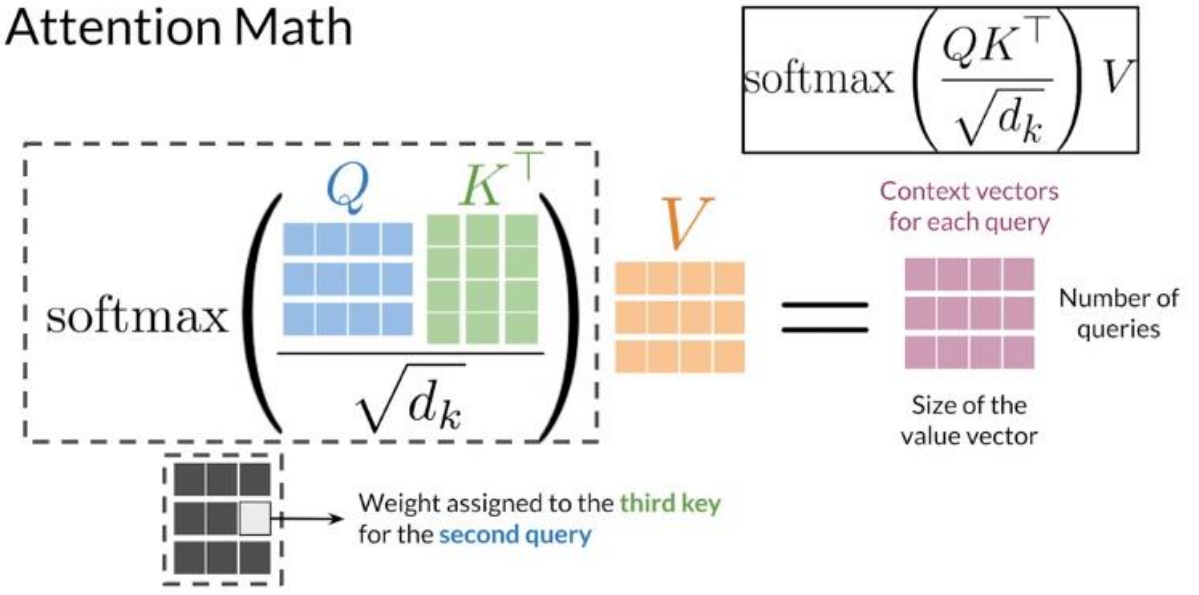


Scaled and Dot-Product Attention

- Essential for Transformer
- The input to Attention is queries, keys and values
- GPUs and TPUs

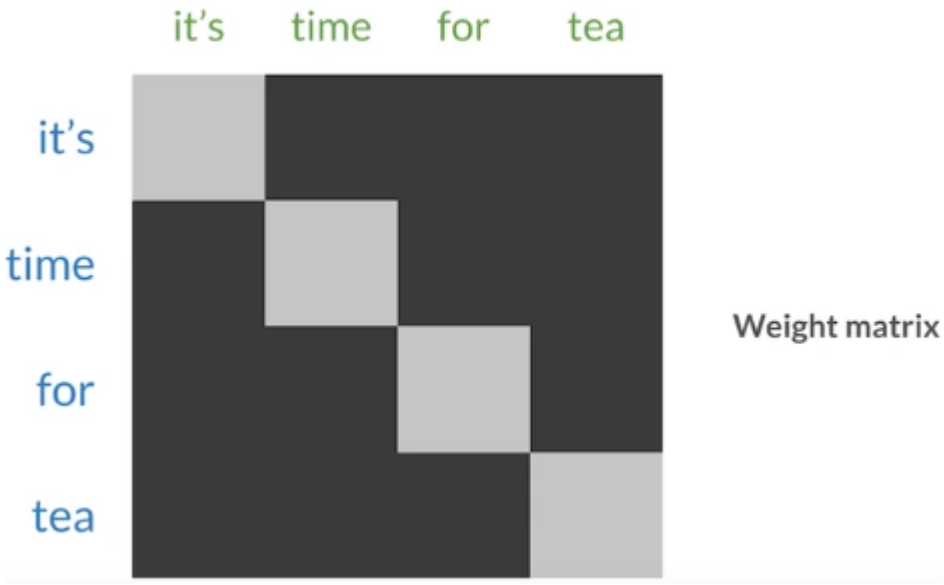




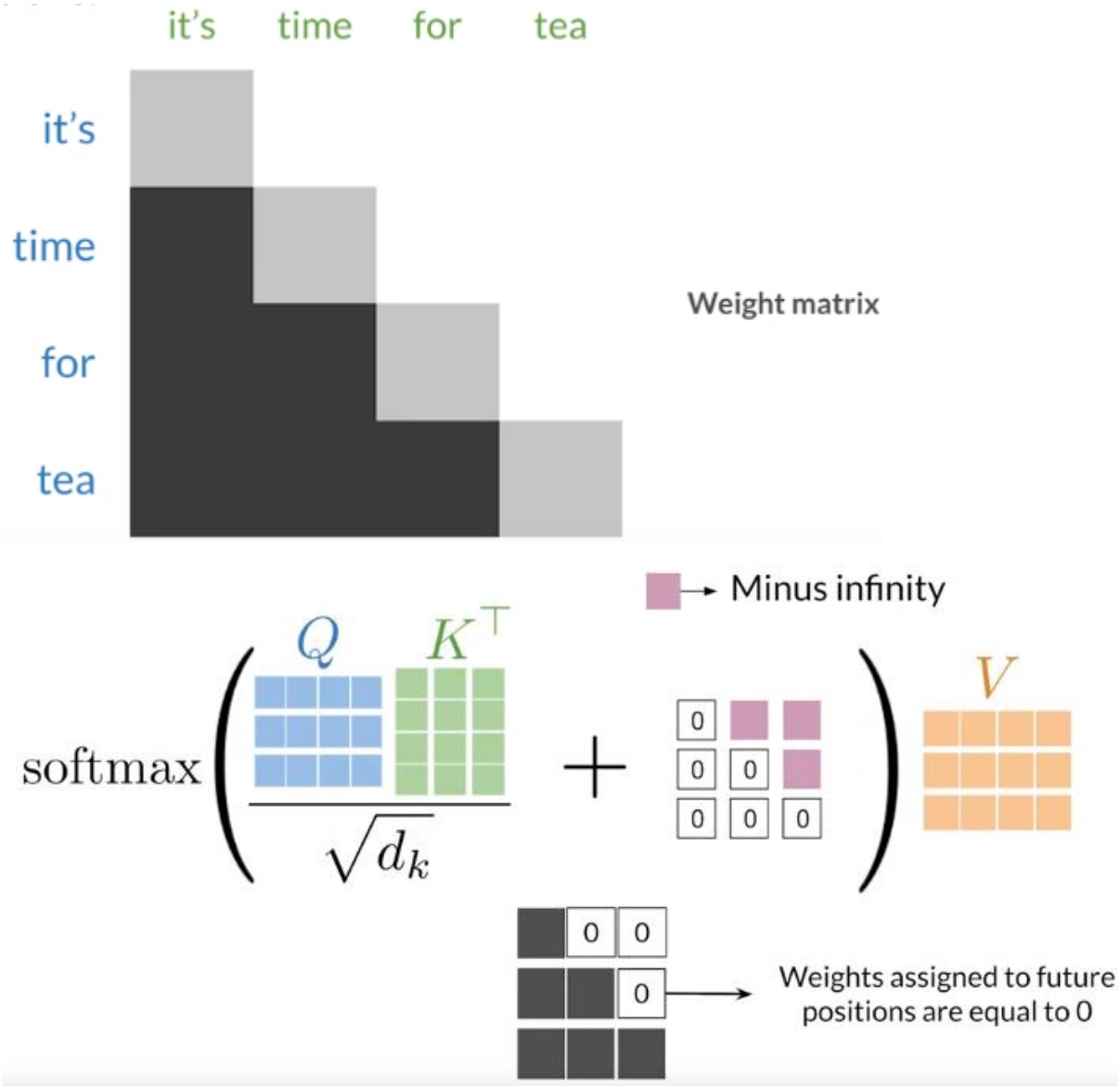


Encoder-Decoder Attention: Queries from one sentence, keys and values from another

Self-Attention: Queries, keys and values come from the same sentences



Masked Self-Attention: Queries, keys and values come from the same sentence. Queries don't attend to future positions

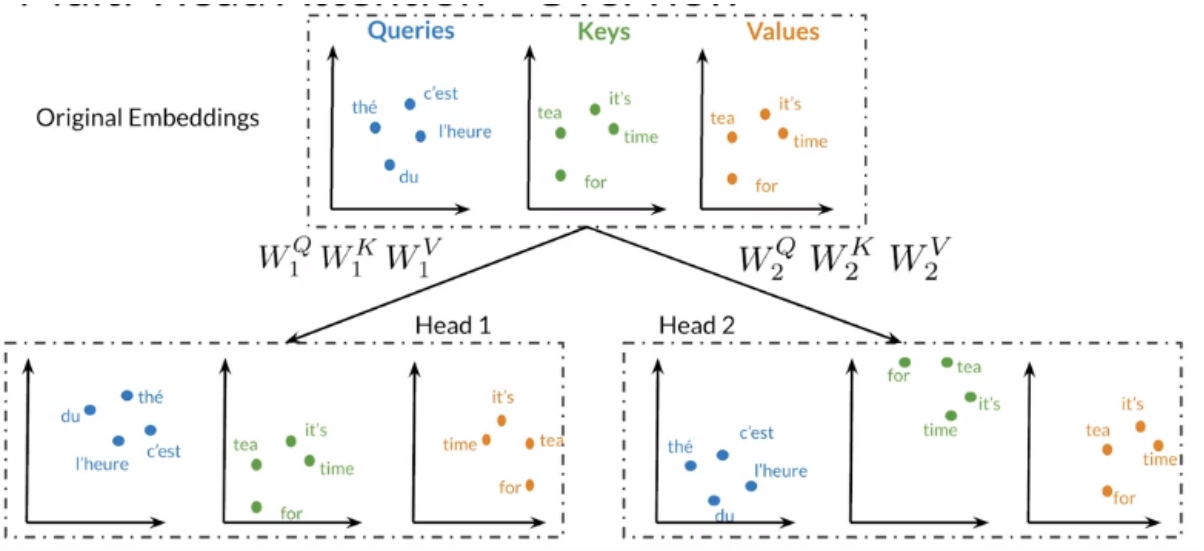


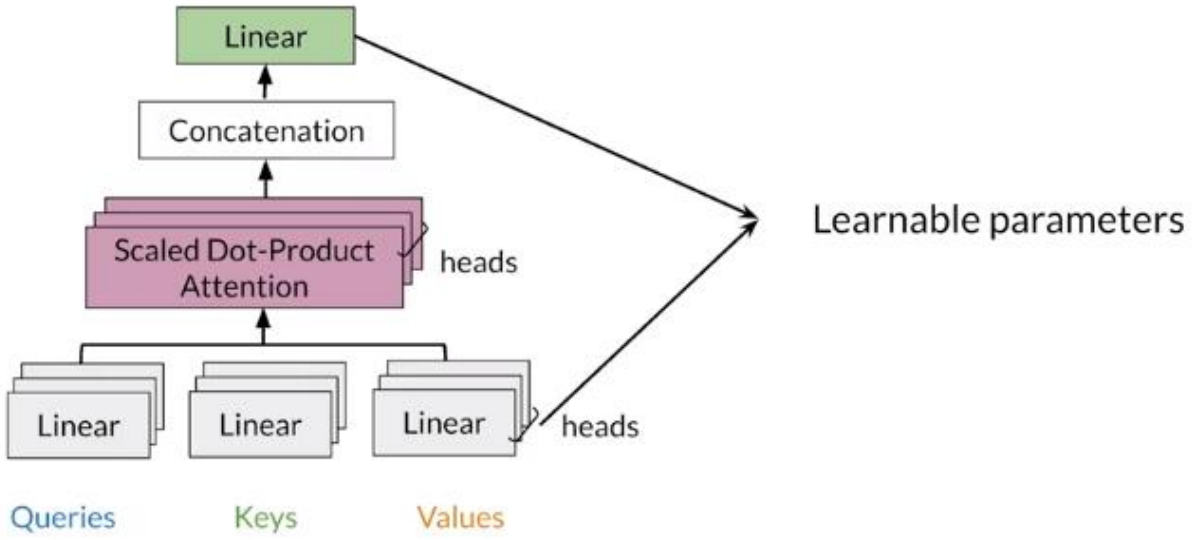
Multi-Head Attention

Attend to information from different representation

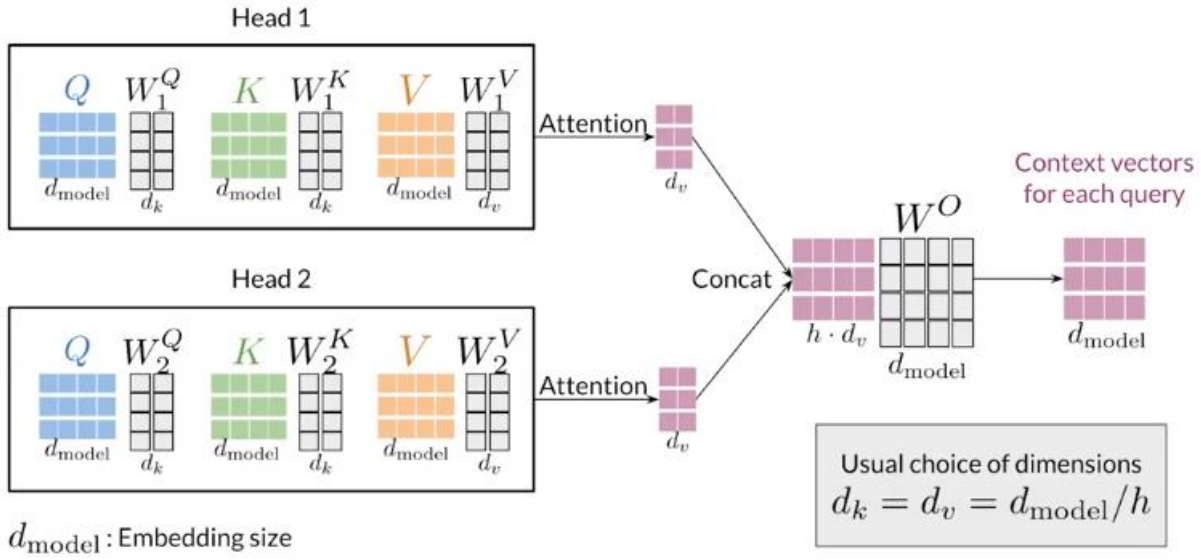
Parallel computations

Similar computational cost to single-head attention



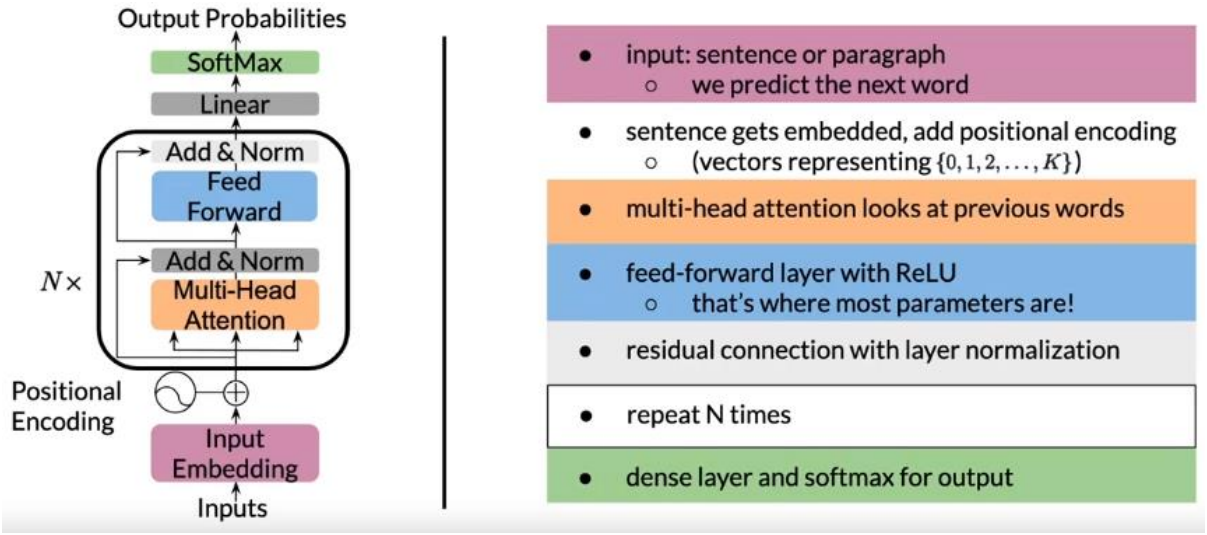


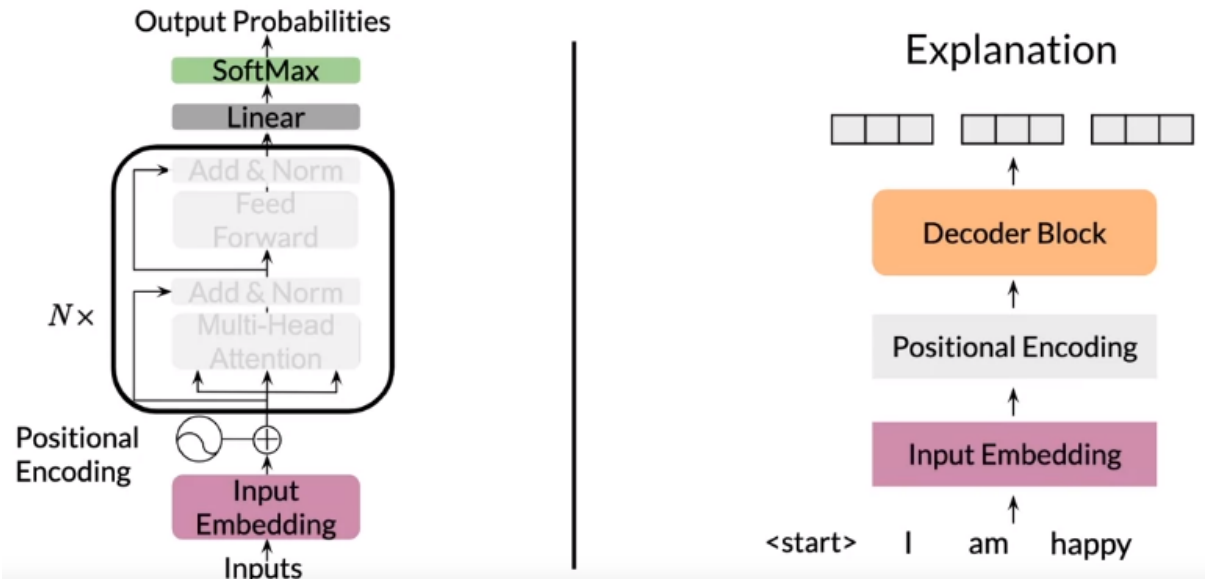
h = number of head

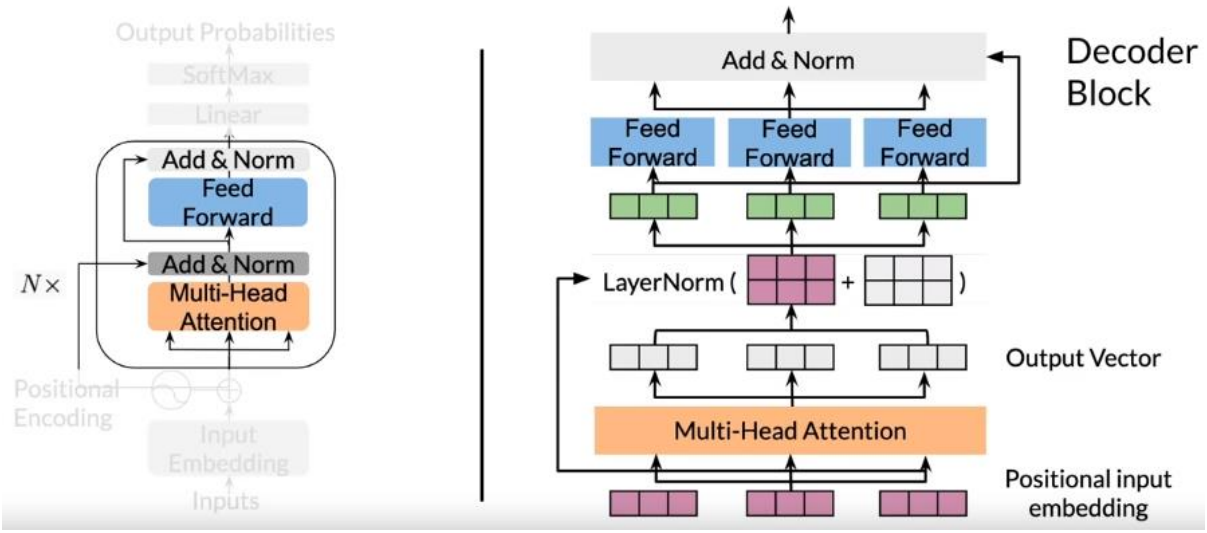


Transformer Decoder

- Mainly consists of three layers
- **Decoder** and Feed-forward blocks (in decoder) are the core of this model
- It also includes a module to calculate the cross-entropy loss

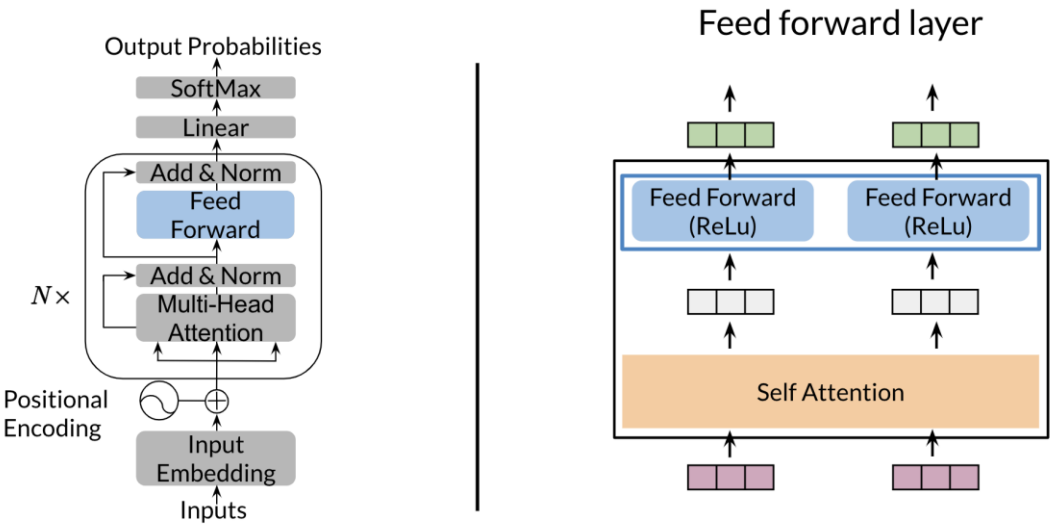






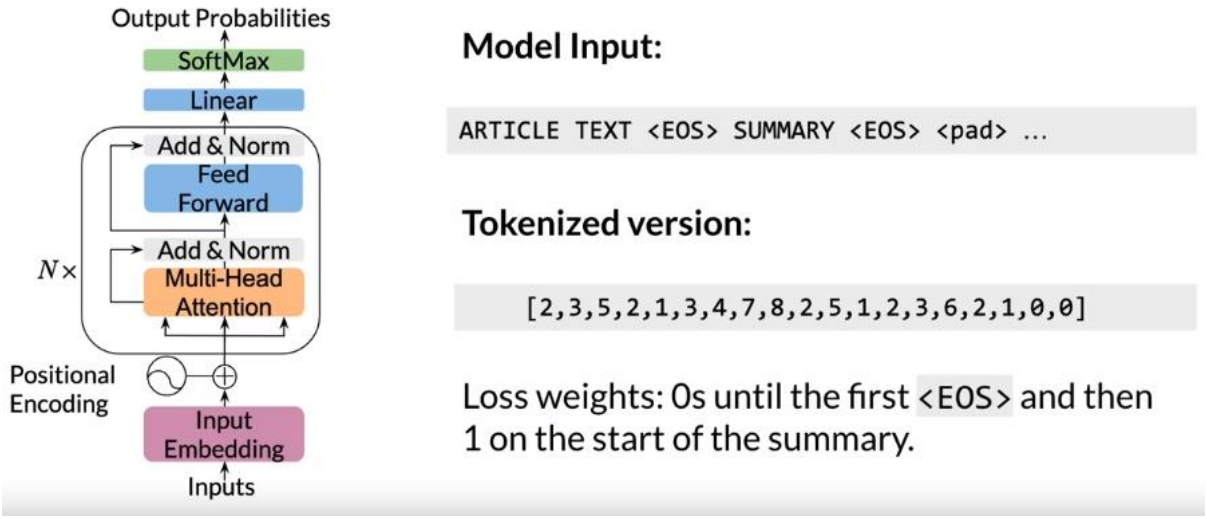
Feed Forward:

- ReLu
- Shared parameters for efficiency
- Replace the hidden states of RNN decoder

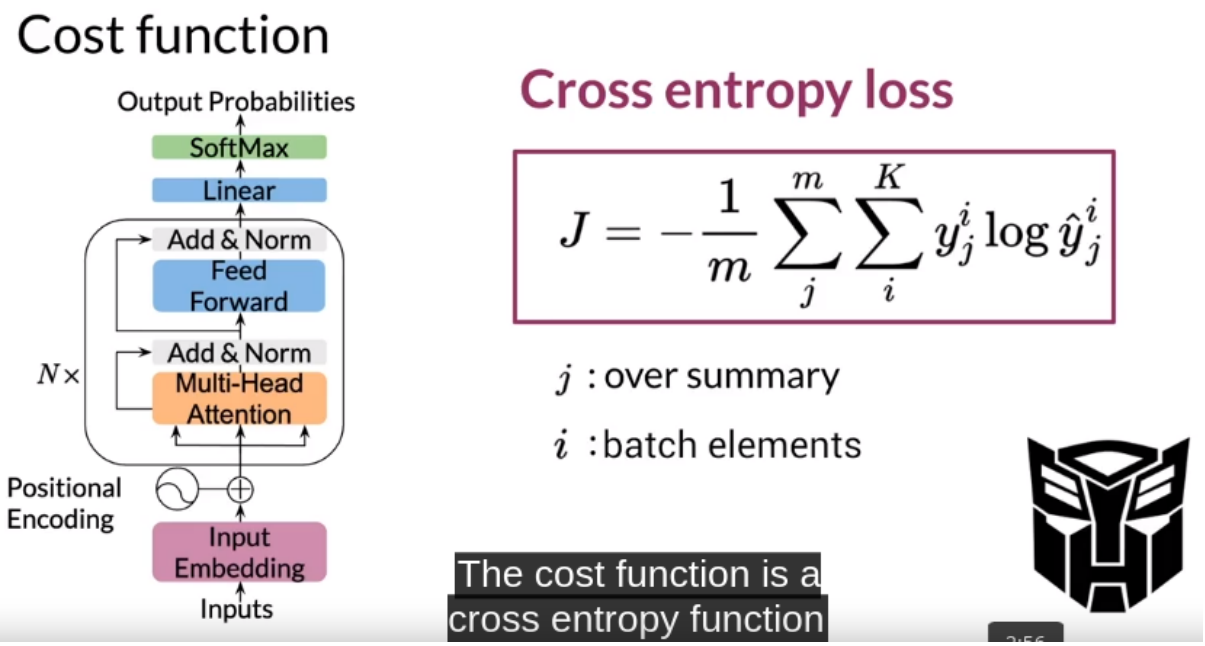


Transformer for summarization

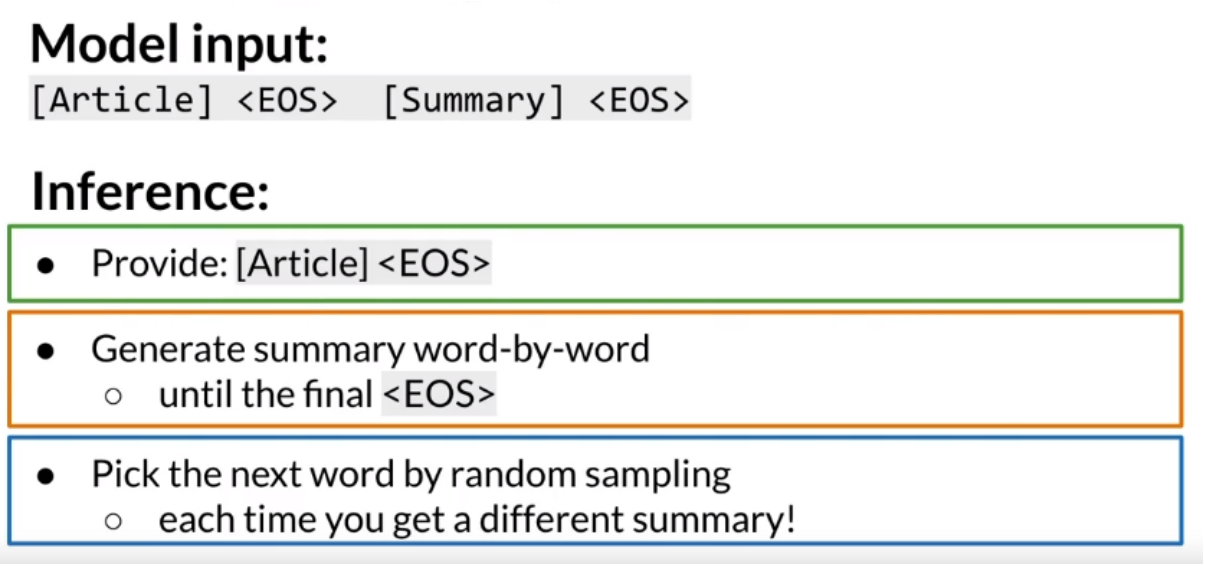
- Transformer only takes text as input and predicts the next word
- Loss weights: 0s until the first <EOS> and then 1 on the start of the summary
- When there is little data for the summaries, it helps to waste the article loss with non zero numbers, ex 0.2 or 0.5 or 1



Cross entropy loss ignores the words from the article to be summarized



Inference with a Language Model



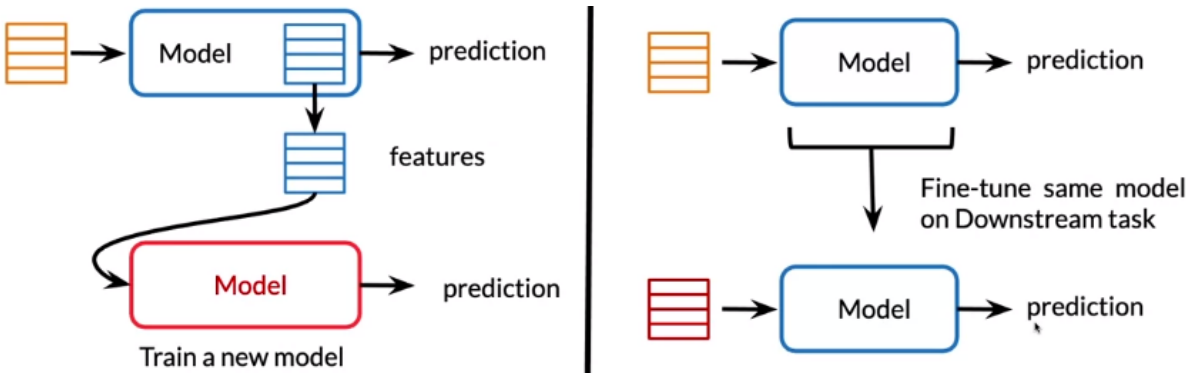
Week 3

Question Answering

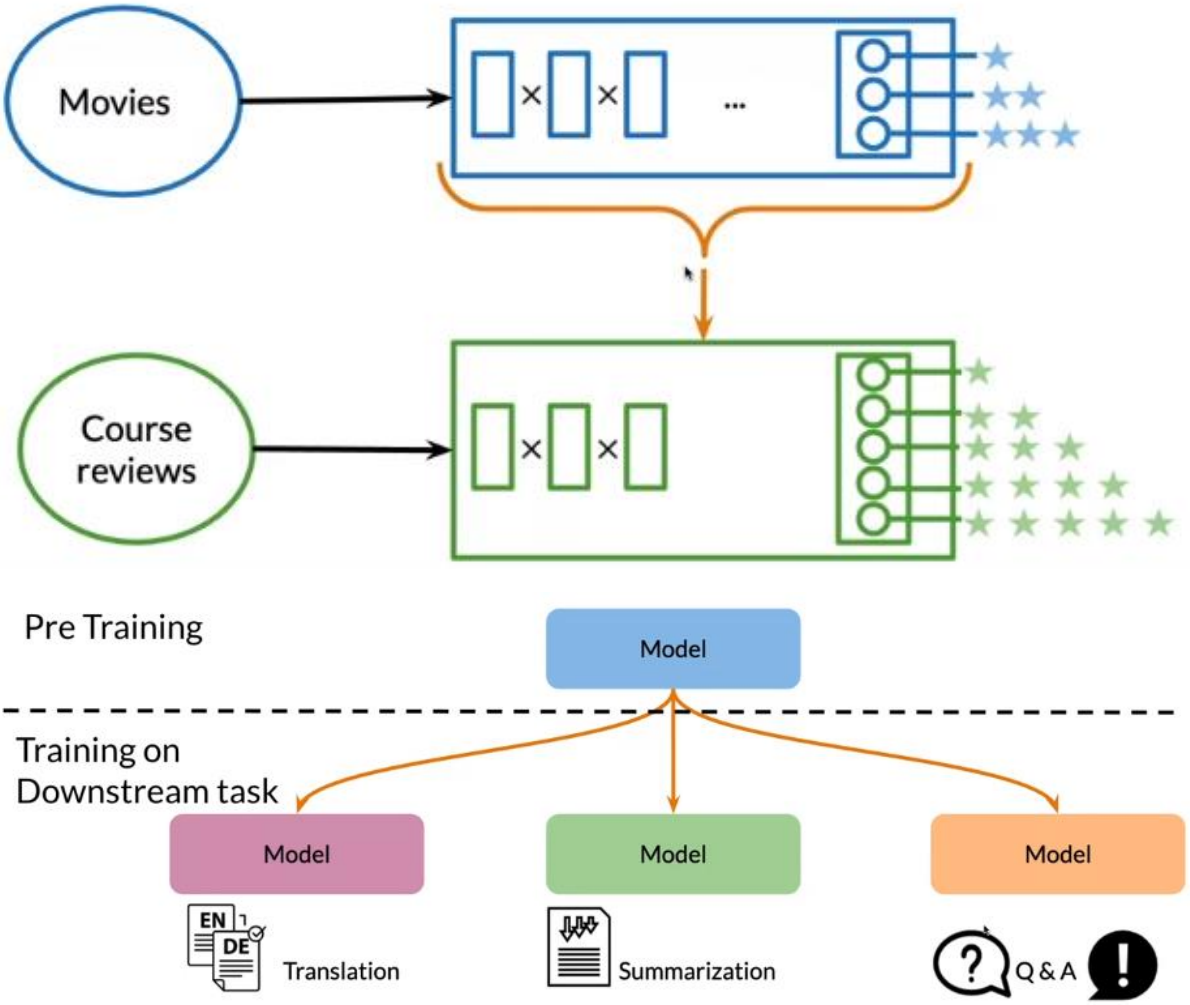
- Context-based
- Closed book

Transfer:

- Feature-based:
  - ex. word vectors being learnt
  - Taking existing weights from a deeplearning model, and then using those weights in another model as they are without changing them

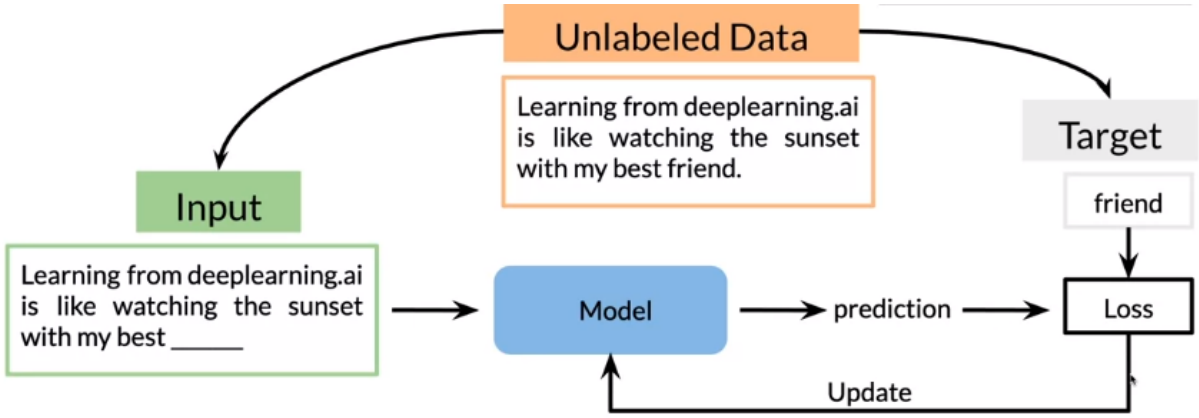


- Fine-tuning a model for each downstream task: tweak model to make sure the model work on the specific task



Pre-train data:

- Labelled
- Unlabelled: self-supervised tasks



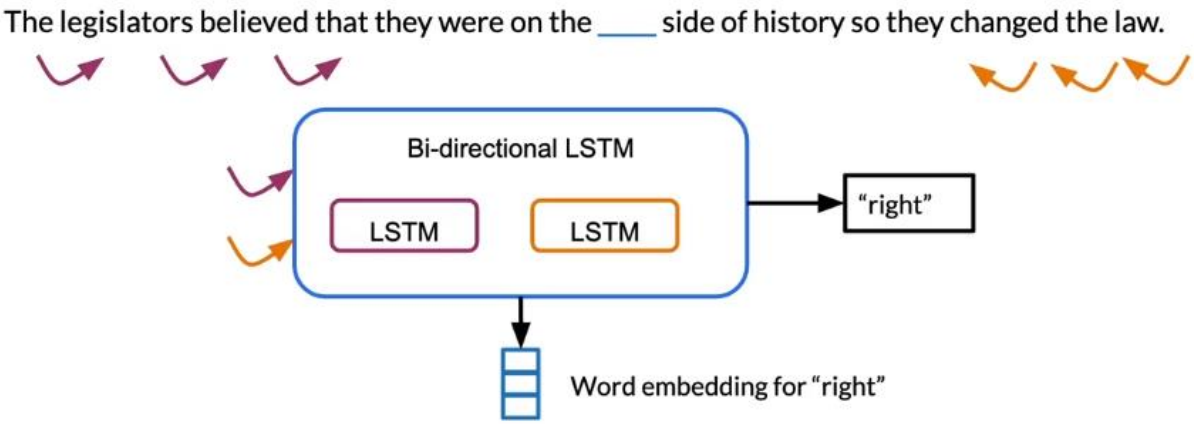
Pre-training task

- Language modelling
- Masked words
- Next sentence

Continuous Bag of Words: fixed window

ELMo:

- Full context using RNN
- Bi-directional LSTM



GPT:

- Transformer: Decoder
- Uni-directional Context

BERT:

- Transformer: Encoder
- Bi-directional Context
- Multi-Mask
- Next Sentence Prediction

T5:

- Transformer: Encoder – Decoder
- Bi-directional Context
- Multi-Task

Bidirectional Encoder Representations from Transformer (BERT)

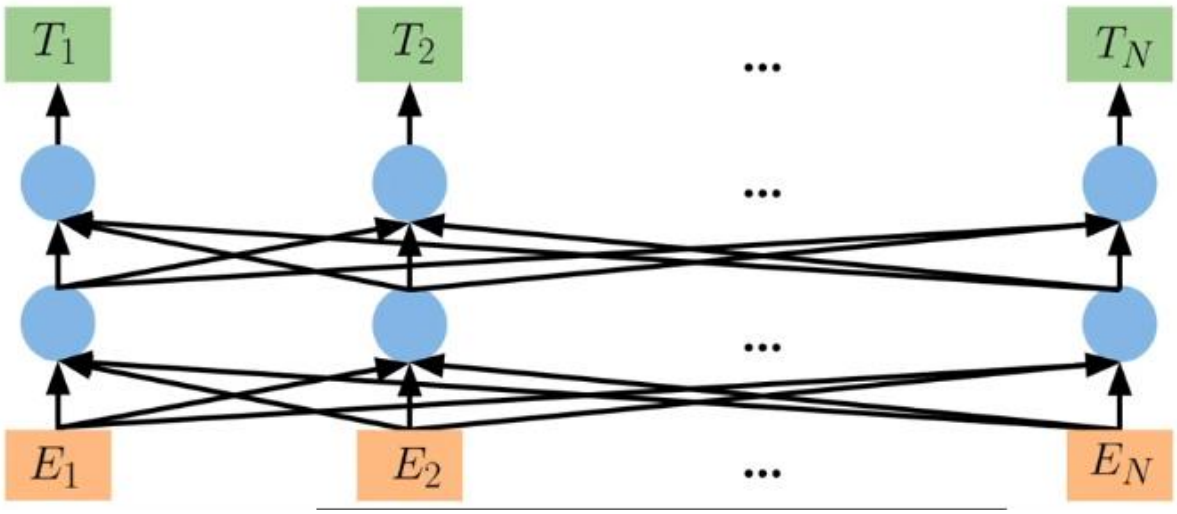
Makes use of transfer learning/pre-training

A multi-layer bidirectional transformer

Positional embeddings

BERT_base:

- 12 layers (12 transformer blocks)
- 12 attentions heads
- 110 million parameters



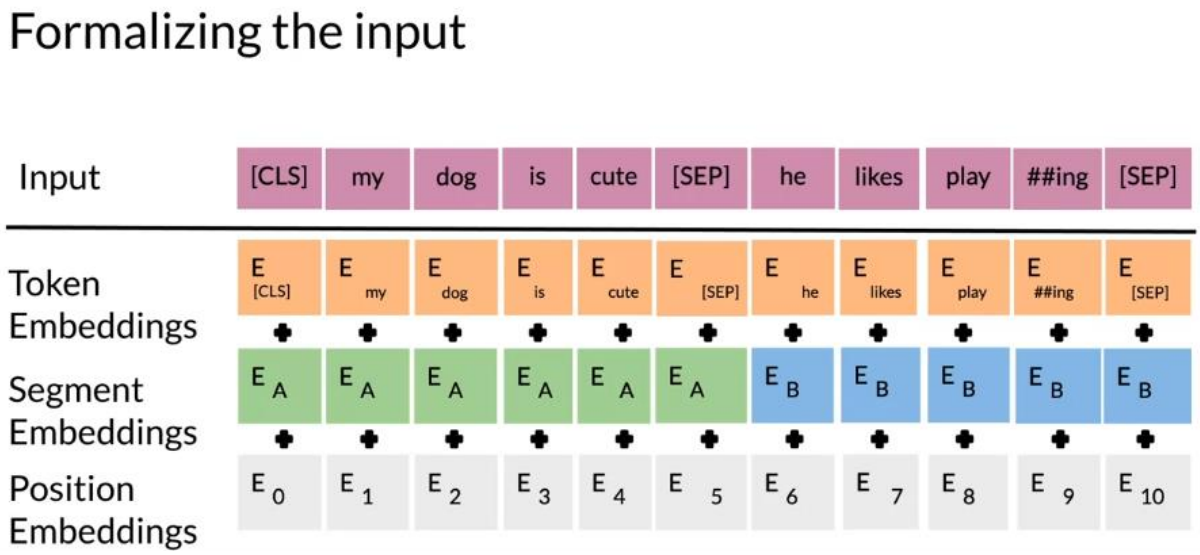
2 steps:

- pre-training: unlabelled data over different pre-training tasks
- fine-tuning: first initialized with a pre-trained parameter, and all the parameters are fine-tuned using labeled data from the downstream tasks

BERT pre-training

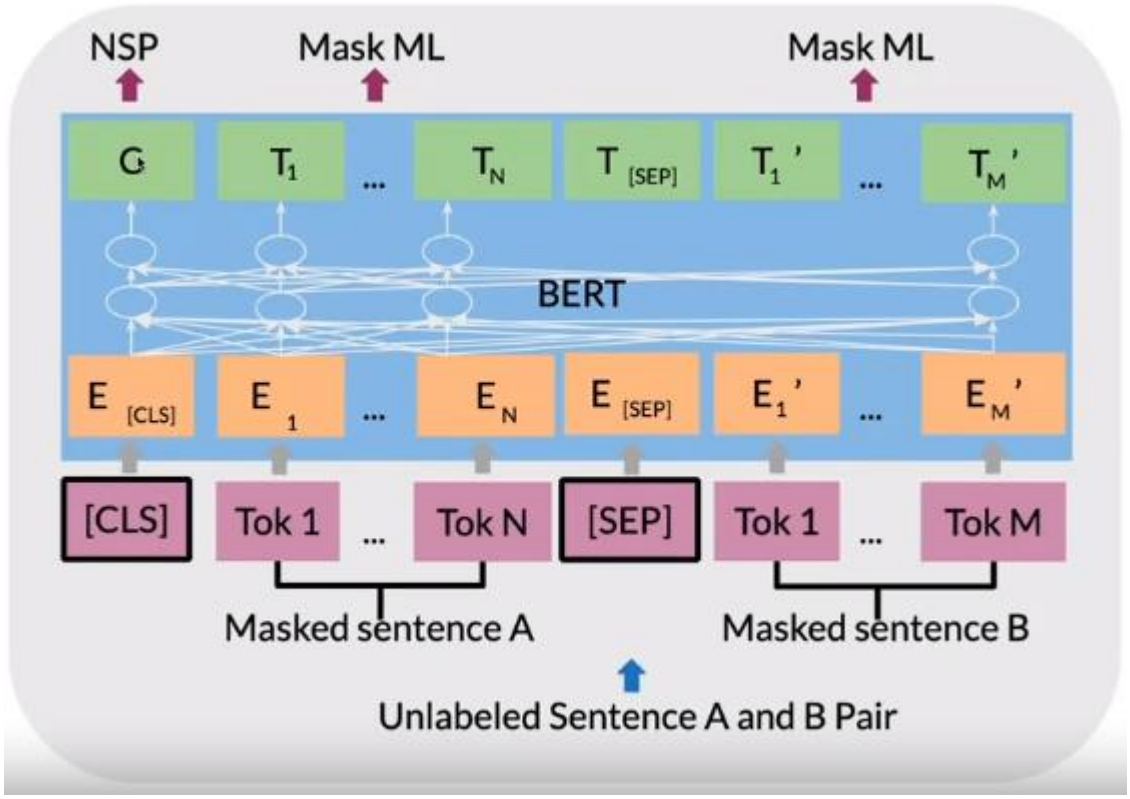
- Masked language modelling
- There could be multiple masked spans in a sentence

- Choose 15% of the tokens at random: mask them 80% of the time, replace them with a random token 10% of the time, or keep as is 10% of the time.
- Next sentence prediction is also used when pre-training. Give two sentences if it's true, it's meaning the 2 sentences follow one another

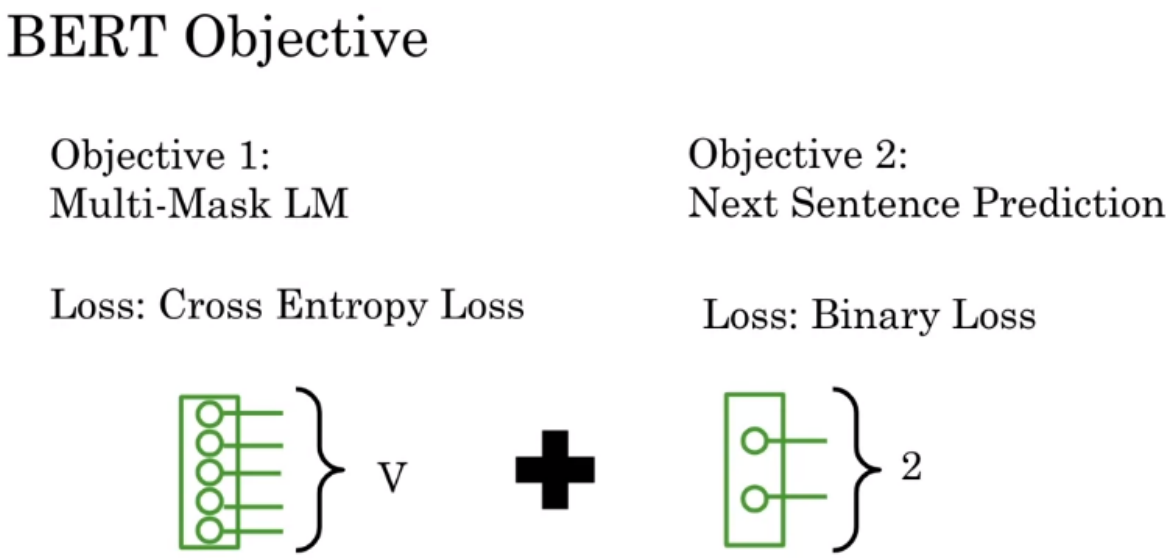


[CLS]: a special classification symbol added in front of every input

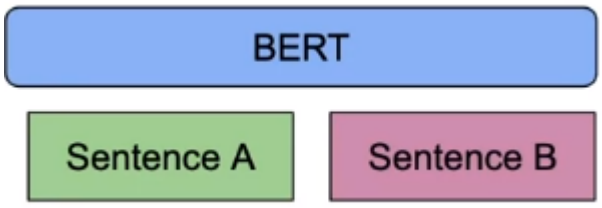
[SEP]: a special separator token



The C token in the image above could be used for classification purposes



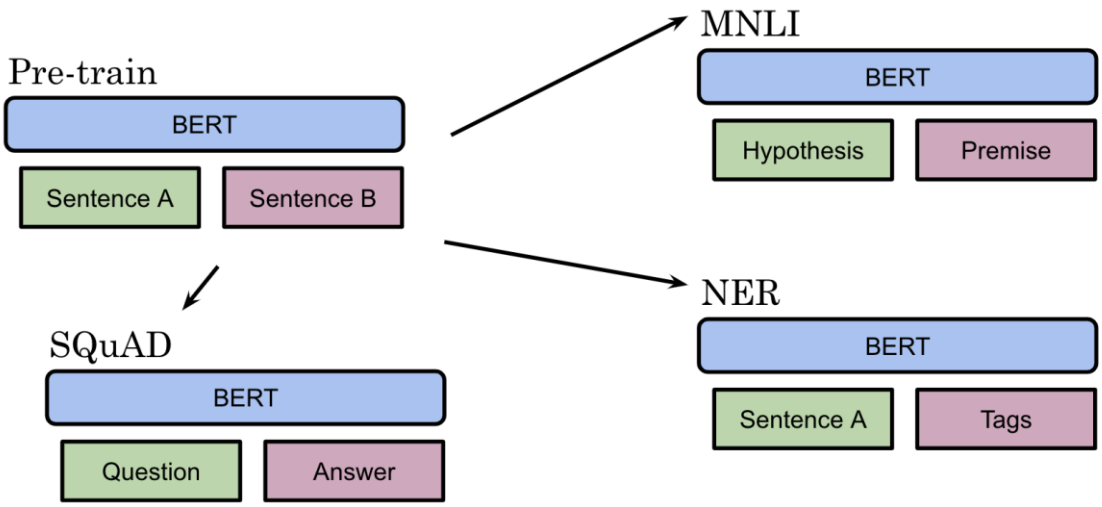
Fine tuning BERT

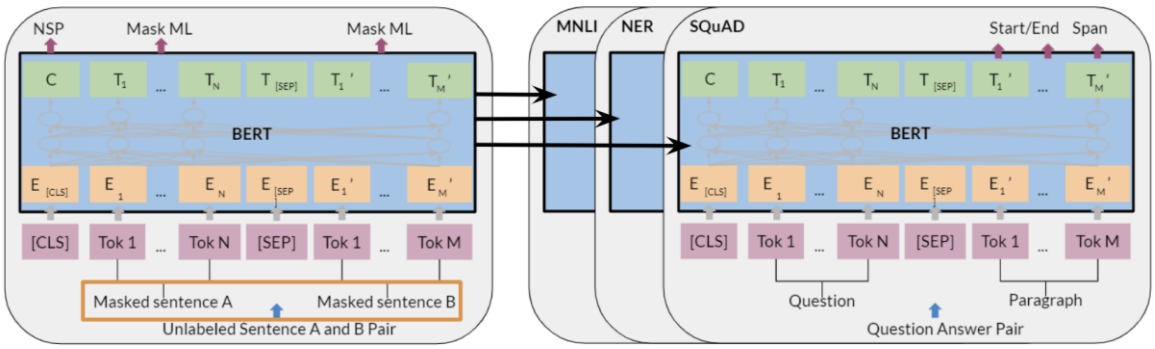


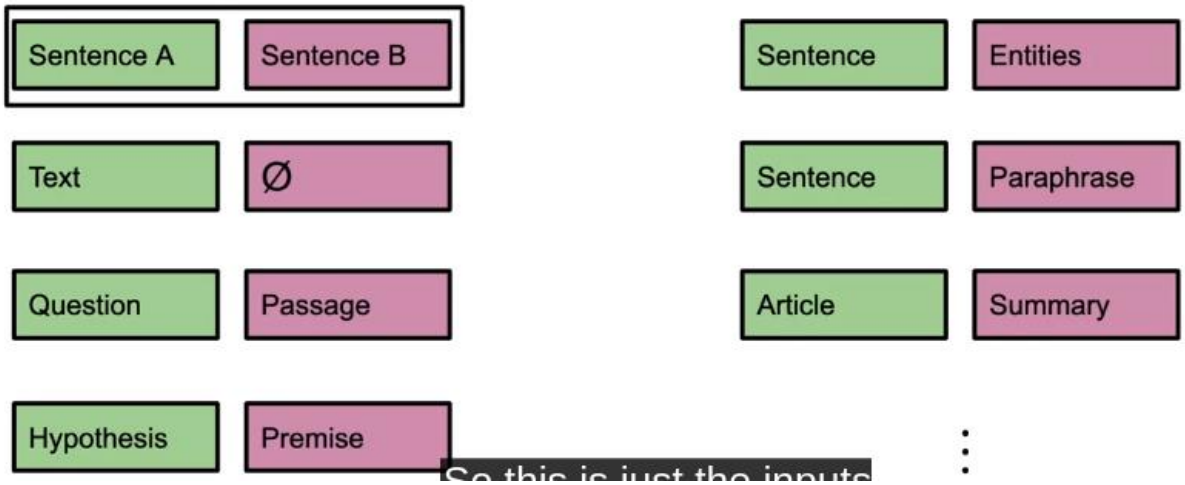
MNLI (Multi-Genre Natural Language Inference)

Premise: “The cat sat on the windowsill, basking in the sunlight.”

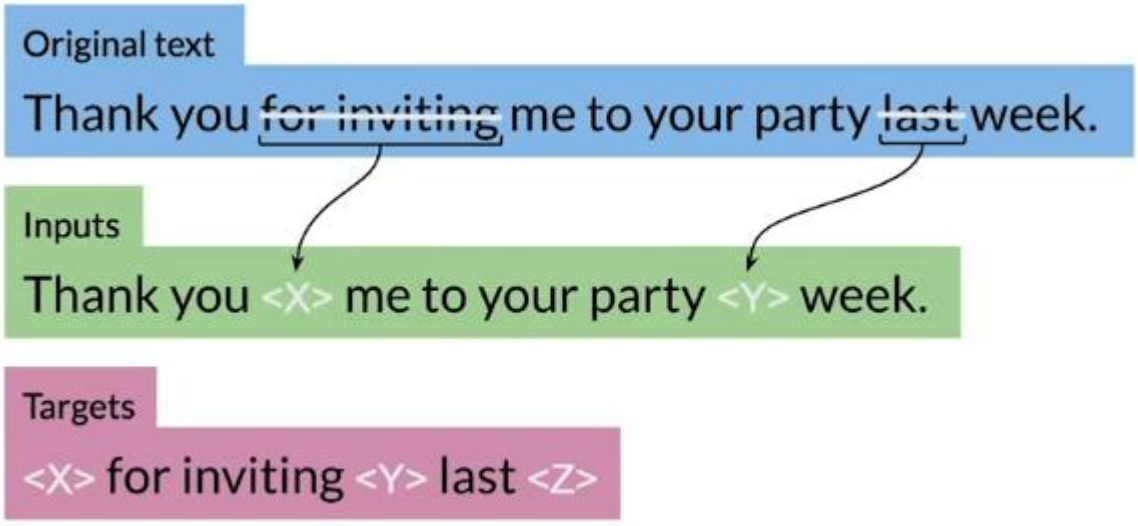
Hypothesis: “The cat is enjoying the warmth of the sun.”

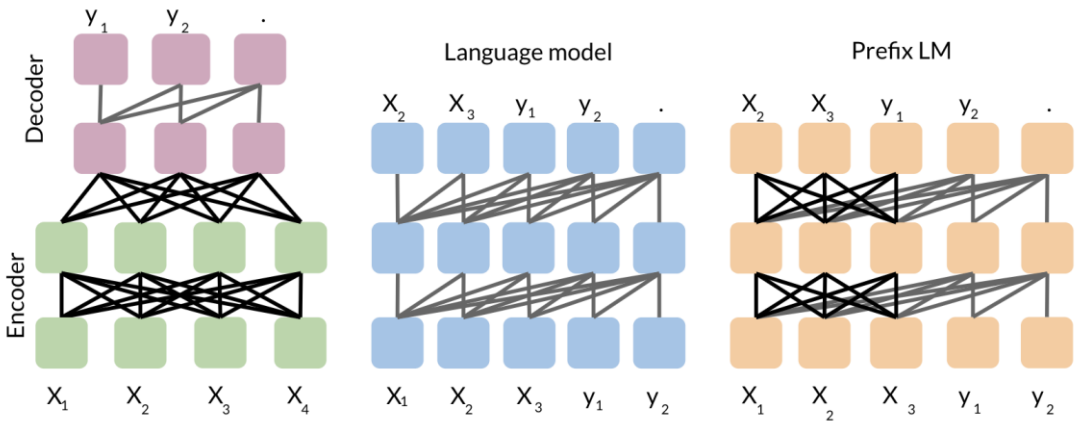






Transformer: T5





Multi-Task Training Strategies

Machine translation:

- Translate English to German: That is good

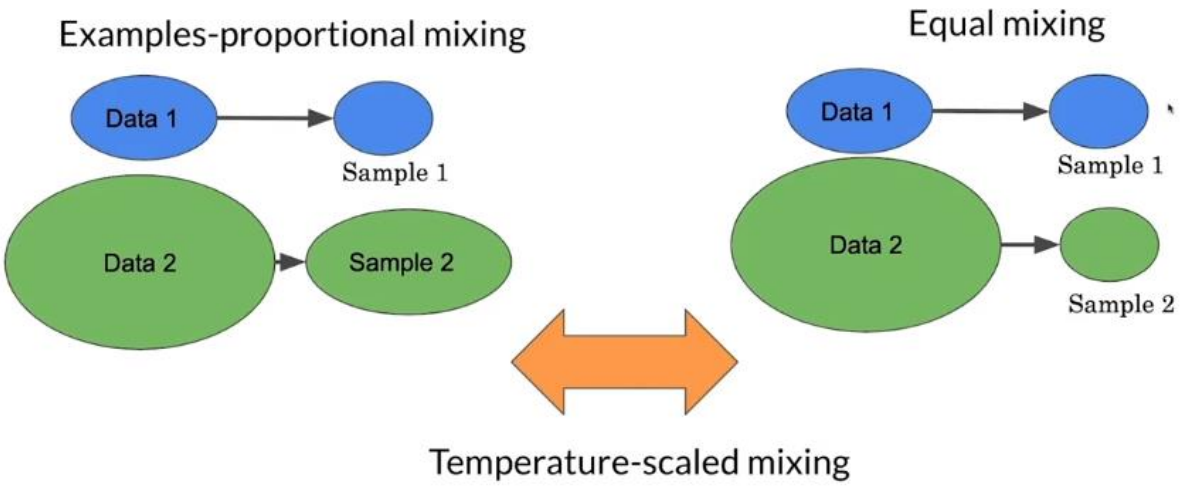
Predict entailment, contradiction, or neutral

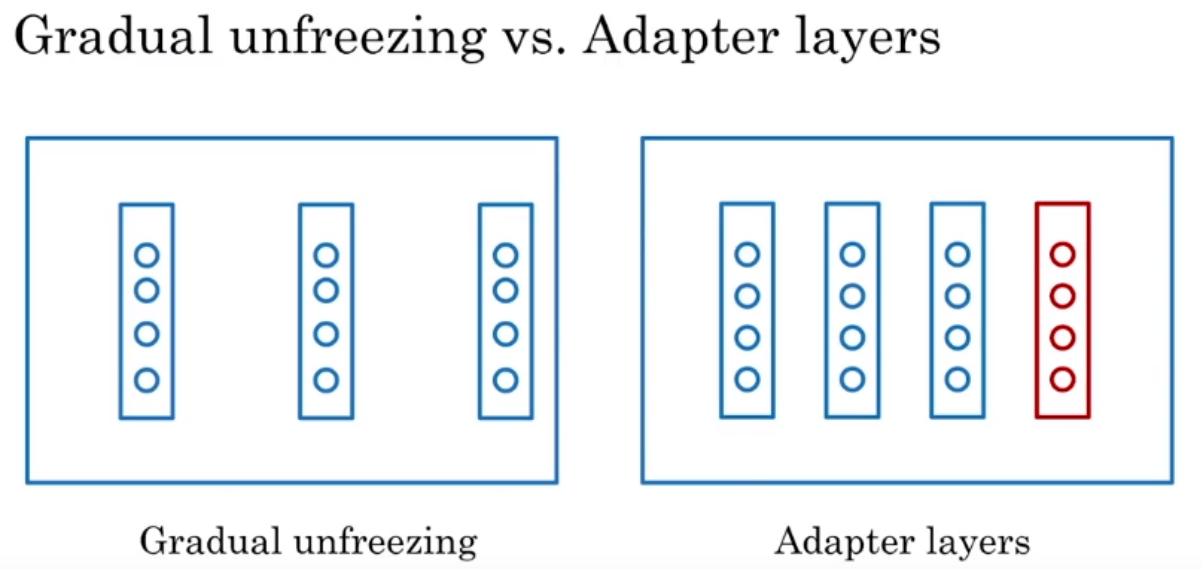
- mnli premise: I hate pigeons hypothesis: My feelings towards pigeons are filled with animosity. Target: entailment

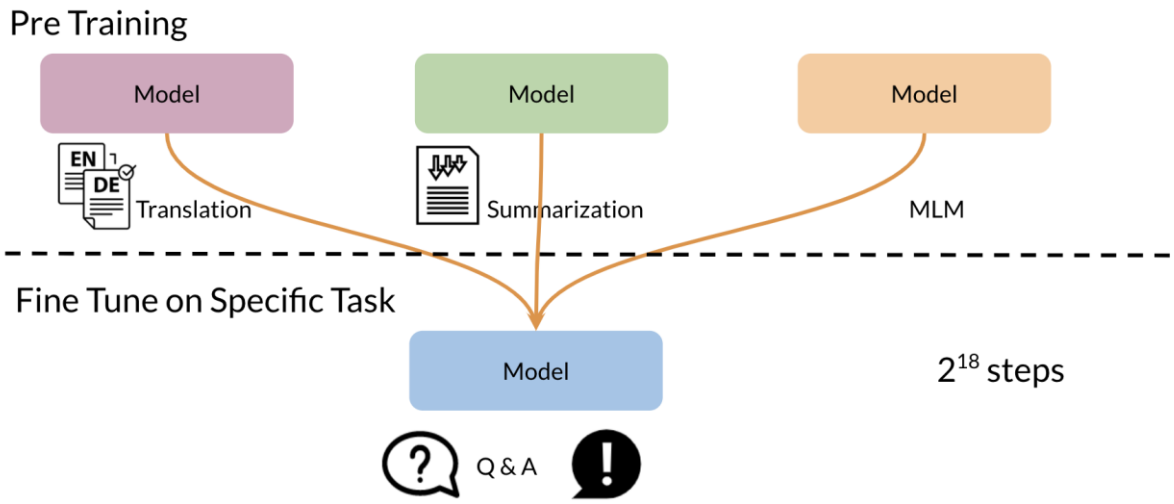
Winograd schema

- The city councilmen refused the demonstrators a permit because *they* feared violence

Data Training Strategies







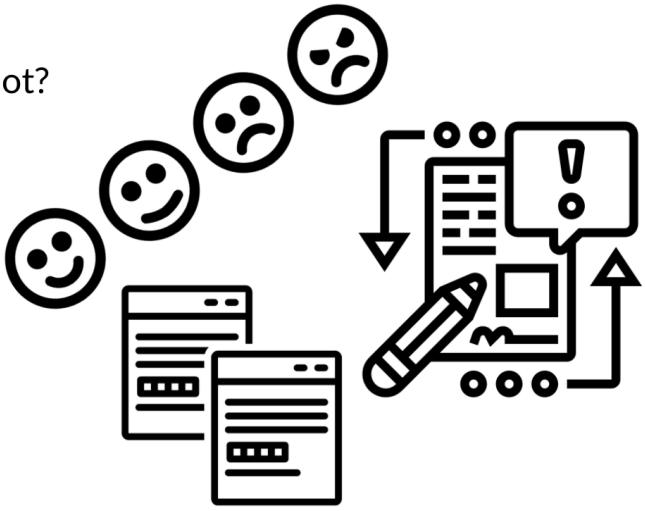
BLUE (General Language Understanding Evaluation) Benchmark

- A collection used to train, evaluate, analyze natural language understanding systems
- Datasets with different genres, and of different sizes and difficulties
- Leaderboard
- This GLUE benchmark is used for research purposes, it is model agnostic, and relies on models that make use of transfer learning.

- Mark as completed

Tasks Evaluated on

- Sentence grammatical or not?
- Sentiment
- Paraphrase
- Similarity
- Questions duplicates
- Answerable
- Contradiction
- Entailment
- Winograd (co-ref)



Sentence Piece

```
eaccent = '\u00E9'
e_accent = '\u0065\u0301'
print(f'{eaccent} = {e_accent} : {eaccent == e_accent}')
```

```
é = é : False
```

SentencePiece uses the Unicode standard normalization form, NFKC, so this isn't an issue. Looking at the example from above but with normalization:

```
from unicodedata import normalize

norm_eaccent = normalize('NFKC', '\u00E9')
norm_e_accent = normalize('NFKC', '\u0065\u0301')
print(f'{norm_eaccent} = {norm_e_accent} : {norm_eaccent == norm_e_accent}')
```

```
é = é : True
```

Normalization has actually changed the unicode code point (unicode unique id) for one of these two characters.

```
def get_hex_encoding(s):
    return ' '.join(hex(ord(c)) for c in s)

def print_string_and_encoding(s):
    print(f'{s} : {get_hex_encoding(s)}')
```

```
for s in [eaccent, e_accent, norm_eaccent, norm_e_accent]:
    print_string_and_encoding(s)
```

```
é : 0xe9
é : 0x65 0x301
é : 0xe9
é : 0xe9
```

BPE (Byte Pair Encoding) Algorithm

Byte Pair Encoding (BPE) is a simple yet effective algorithm used for data compression and particularly popular in natural language processing tasks like machine translation and text generation. It was originally proposed by Sennrich, Haddow, and Birch in 2016.

Here's a simplified explanation of how the BPE algorithm works:

Initialization: BPE starts with initializing a vocabulary that contains all the characters or tokens in the training data. Each character or token is initially considered as a separate symbol.

Tokenization: The input text is tokenized into individual characters or tokens (e.g., words or subwords).

Frequency Counting: BPE counts the frequency of each pair of adjacent symbols (characters or tokens) in the training data. It keeps track of the frequency of all symbol pairs.

Merging: BPE iteratively merges the most frequent pair of symbols into a single new symbol. The most frequent pair is determined based on the frequency counts obtained in the previous step. After merging, the new symbol is added to the vocabulary.

Updating Vocabulary: After merging, the vocabulary is updated to reflect the newly merged symbols.

Repeat: Steps 3-5 are repeated for a fixed number of iterations or until a certain condition is met (e.g., reaching a predefined vocabulary size).

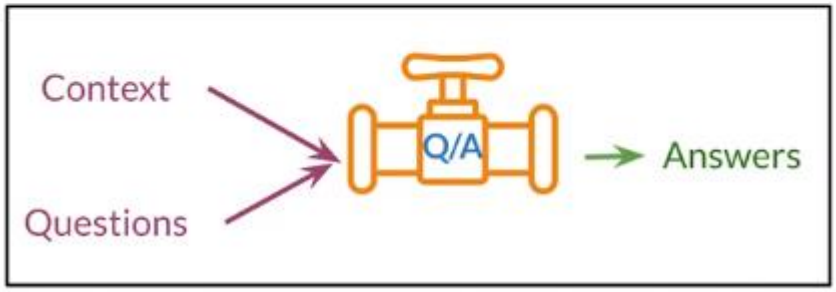
End: Once the iterations are complete, the final vocabulary is obtained, which consists of a set of symbols that represent the most frequent character sequences in the training data.

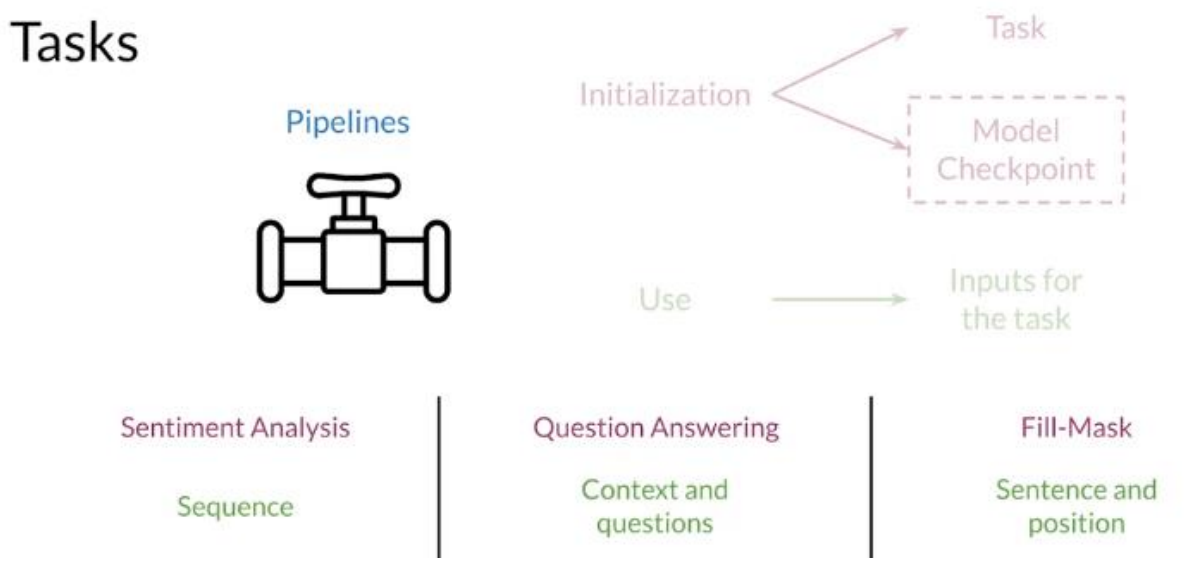
The resulting vocabulary can then be used to encode the input text by replacing the frequent character sequences with their corresponding symbols from the vocabulary. This compression helps in reducing the size of the data representation while preserving the essential information.

BPE is particularly useful in tasks where the vocabulary size needs to be limited, such as in neural machine translation models, where a smaller vocabulary size can improve computational efficiency and generalization. Additionally, BPE can handle out-of-vocabulary words by breaking them down into subword units, enabling the model to process unseen words effectively.

Hugging Face

Using Transformers by Pipelines

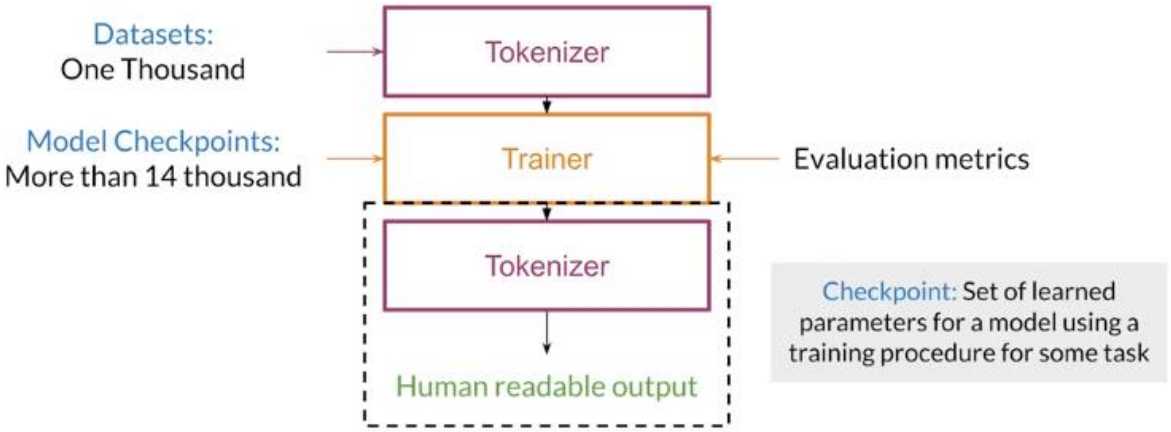




Fine-Tuning Transformers

- Datasets
  - load them using just one function
  - Optimized to work with massive amounts of data
- Tokenizer: associated with every checkpoint to pre-process the data, translating your text to readable inputs for the model. Also, can help you transform into a human readable text.

- Trainer
  - Trainer object lets you define the training procedure
  - Number of epochs
  - Warm-up steps
  - Weight decay
  - Train using one line of code
- Checkpoints:
  - Huge number of model checkpoints that you can use in your pipelines
  - But beware, not every checkpoint would be suitable for your task
  - Upload the architecture and weights with 1 line of code
- Evaluation metrics:
  - predefined metrics that you can use to evaluate your model's performance which can integrate into the trainer objects
  - Like BLEU and ROUGE



Model Hub

- Hub containing models that you can use in your pipelines according to the task you need: https://hunggingface.co/models
- Model card shows a description of your selected model and useful information such as code snippet examples