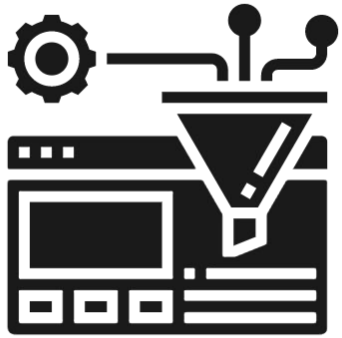
deeplearning.ai

Seq2Seq model for NMT

Outline

- Introduction to Neural Machine Translation
- Seq2Seq model and its shortcomings
- Solution for the information bottleneck



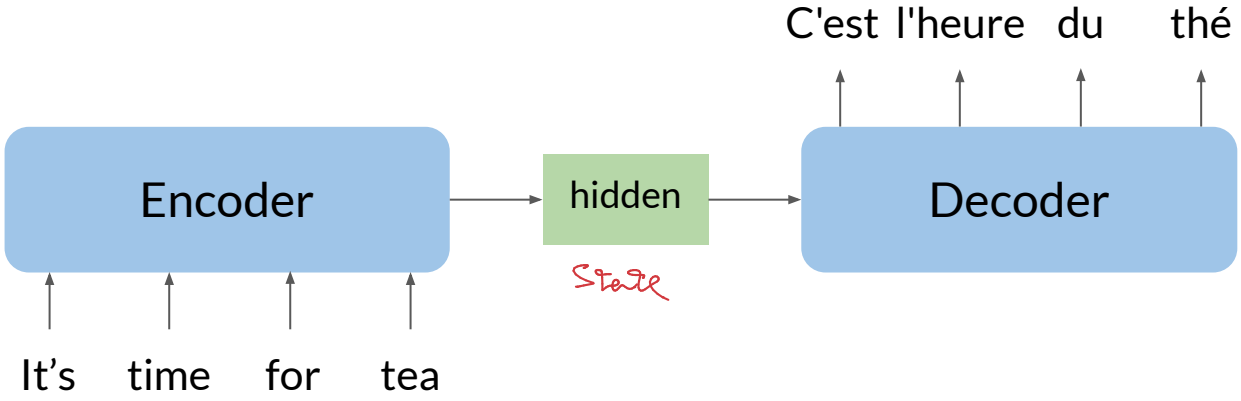
Neural Machine Translation

It’s time for tea → C'est l'heure du thé

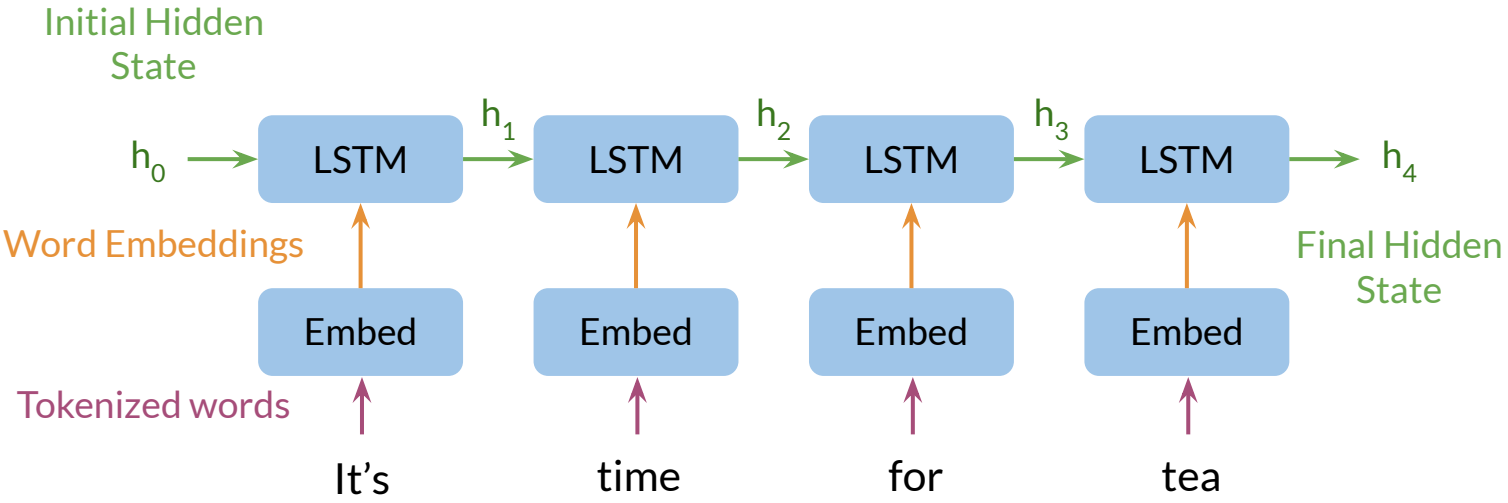
Seq2Seq model

- Introduced by Google in 2014
- Maps variable-length sequences to fixed-length memory → encode
- Inputs and outputs can have different lengths
- LSTMs and GRUs to avoid vanishing and exploding gradient problems

Seq2Seq model

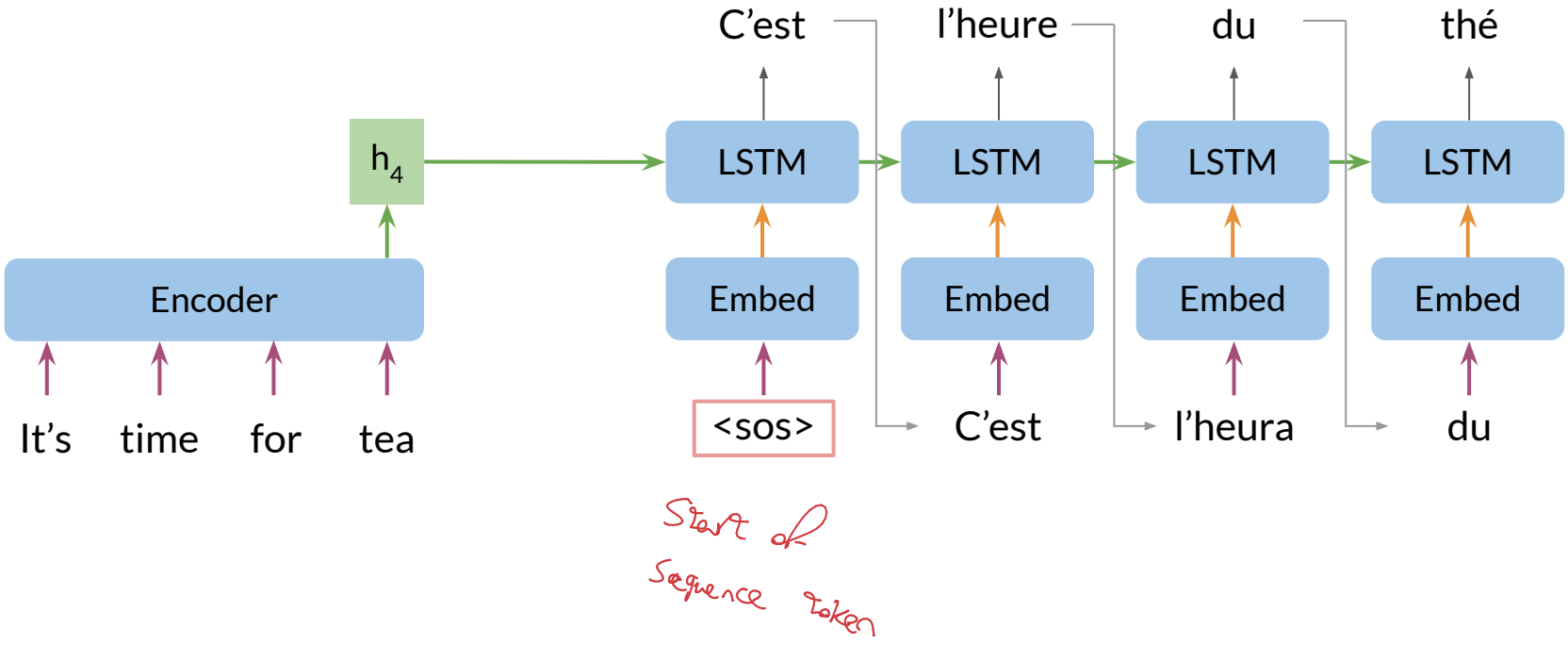


Seq2Seq encoder

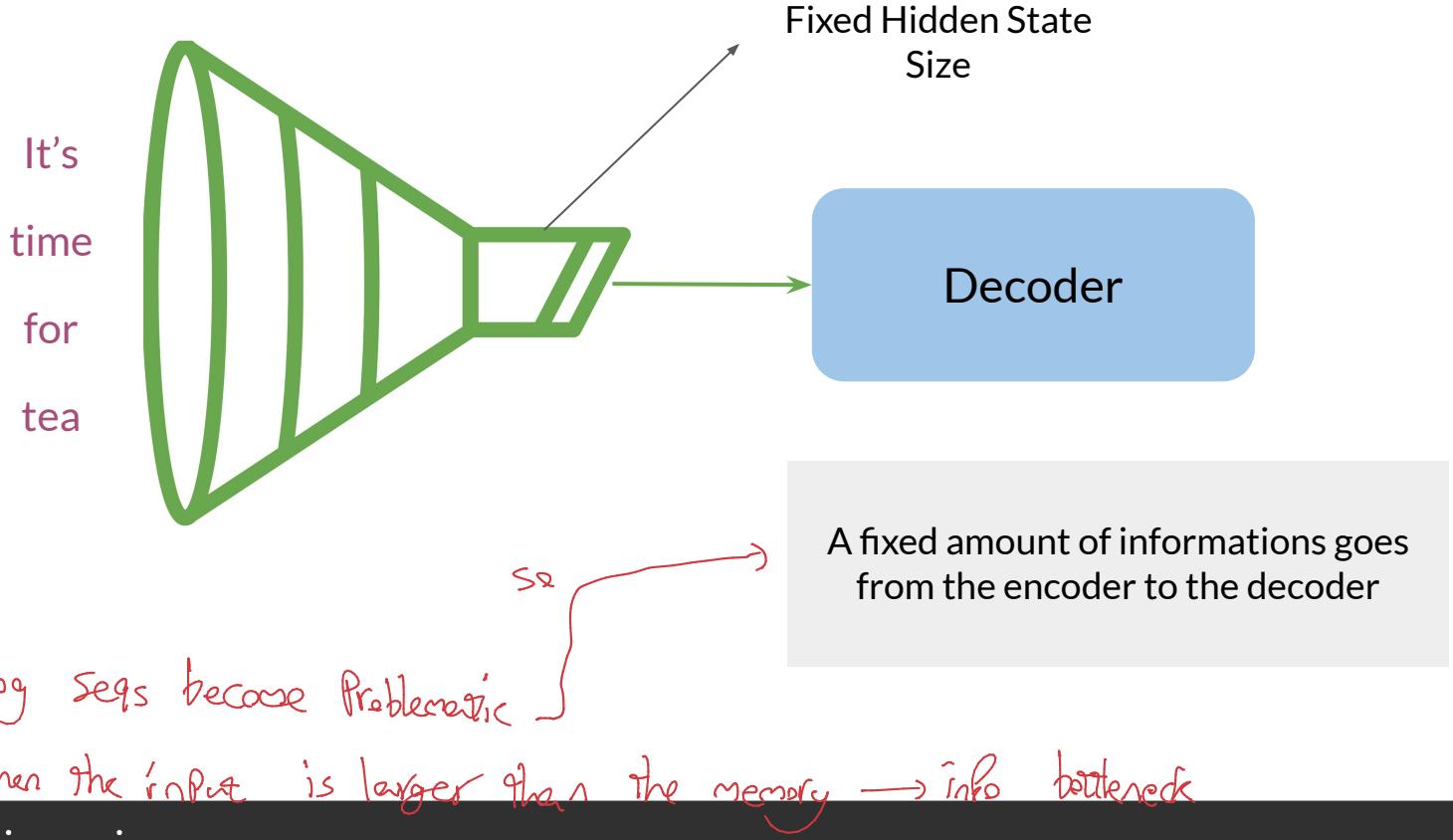


Encodes the overall meaning of the sentence

Seq2Seq decoder



The information bottleneck



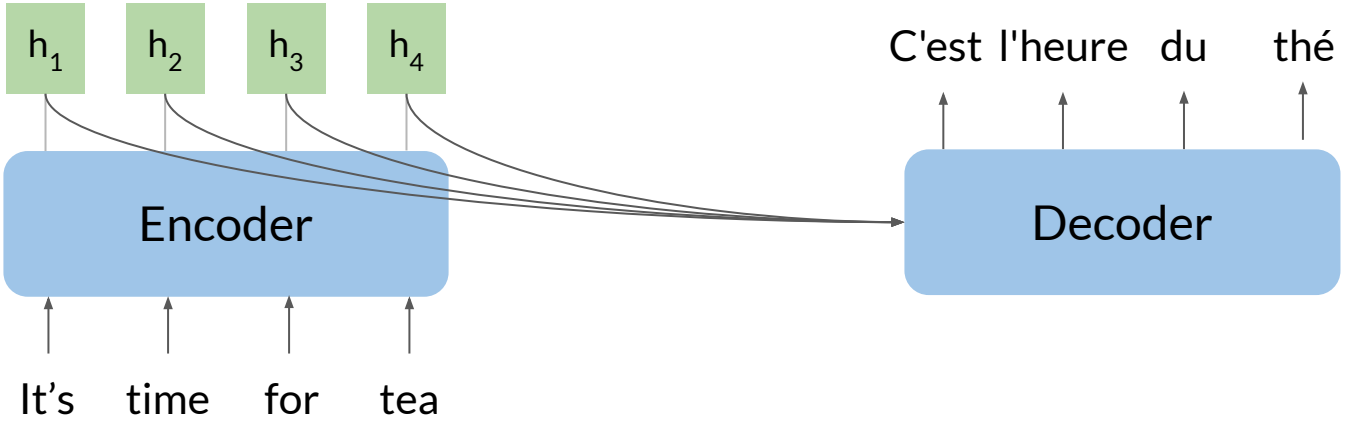
Seq2Seq shortcomings

- Variable-length sentences + fixed-length memory =

- As sequence size increases, model performance decreases

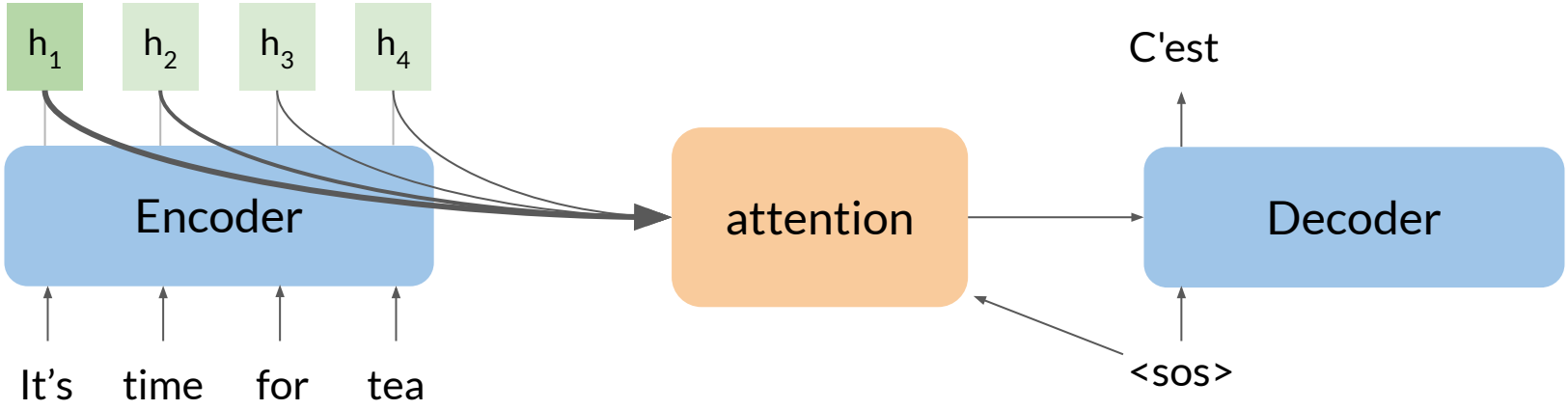
→ It struggle to compress longer seqs → ends up throttling itself and punishing the decoder

Use all the encoder hidden states?



would have flaws with memory & context

Solution: focus attention in the right place



The model can focus on specific hidden states at every step

deeplearning.ai

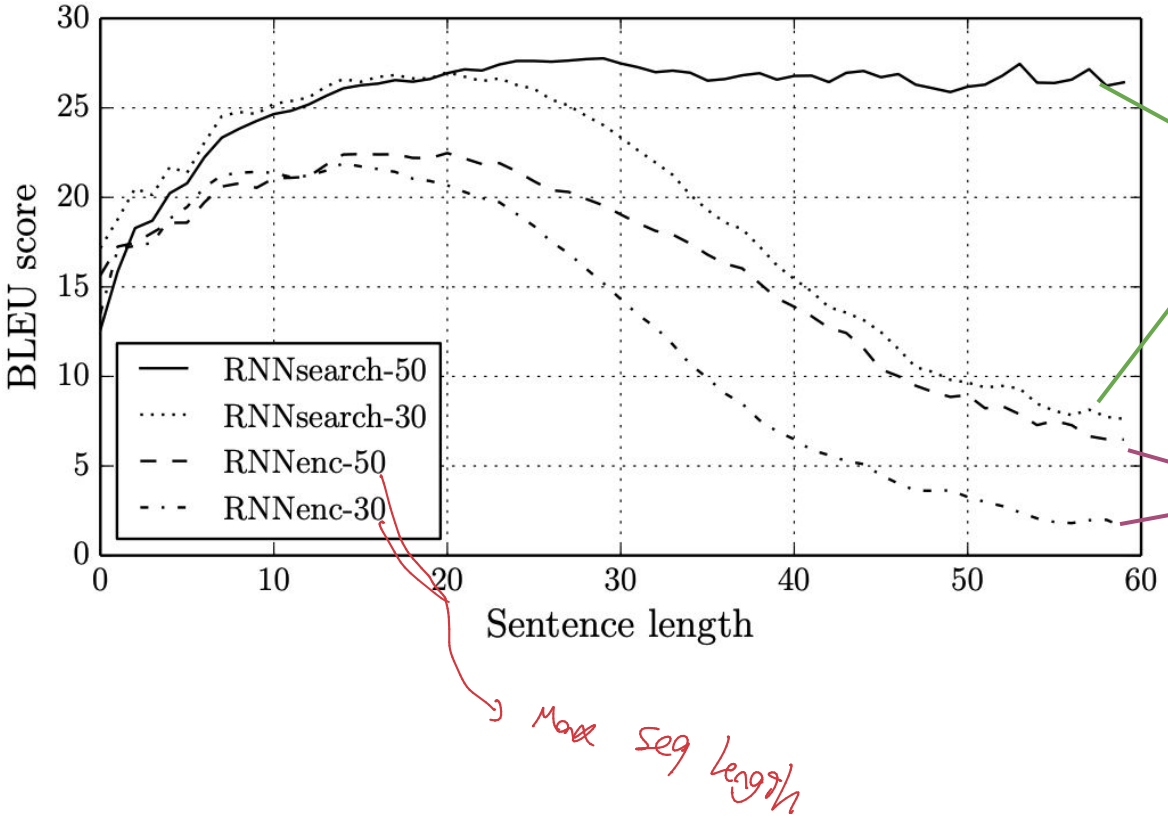
Seq2Seq model with attention

NEURAL MACHINE TRANSLATION BY JOINTLY LEARNING TO ALIGN AND TRANSLATE

Dzmitry Bahdanau
Jacobs University Bremen, Germany

KyungHyun Cho **Yoshua Bengio***
Université de Montréal

Performance



(bidirectional encoder & decoder)

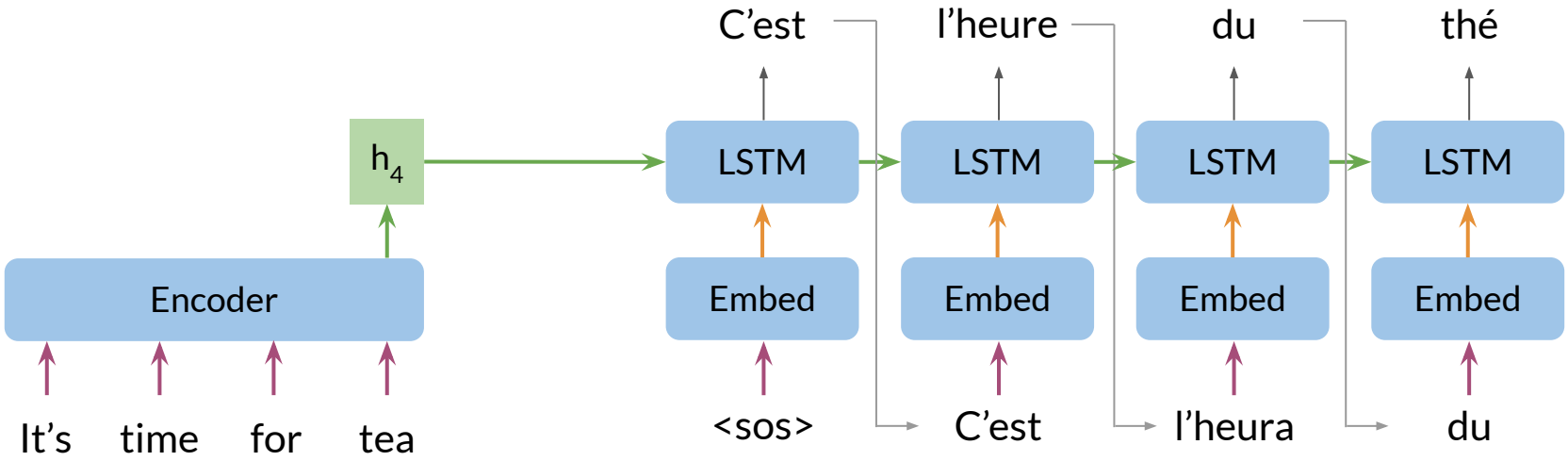
Seq2Seq with Attention

Traditional Seq2Seq Models

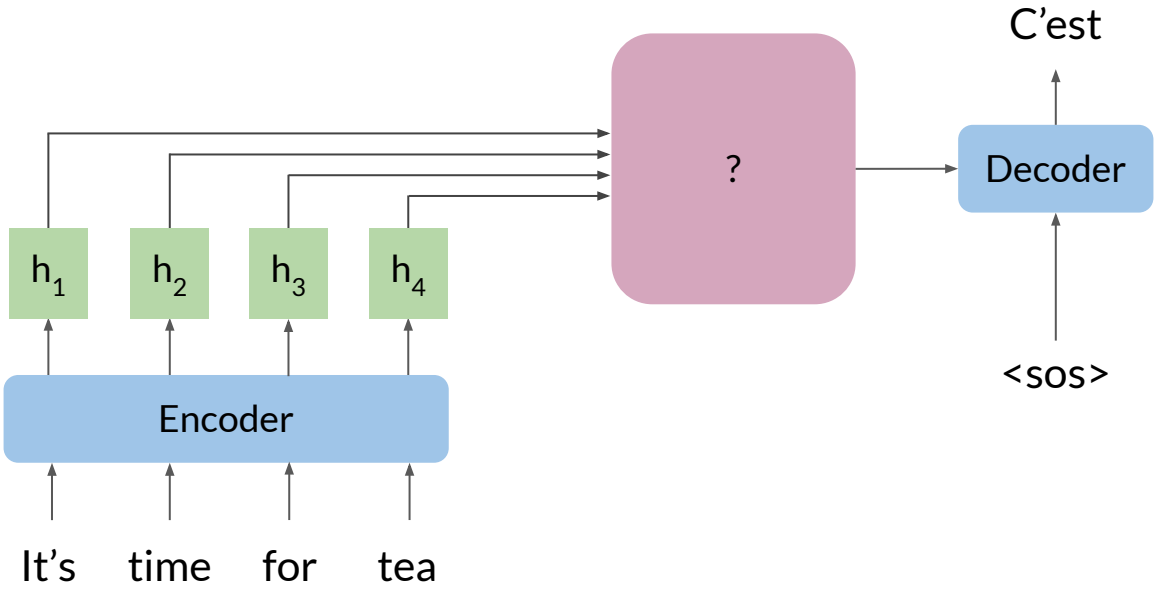
Max Seq length

Greater BLEU is better

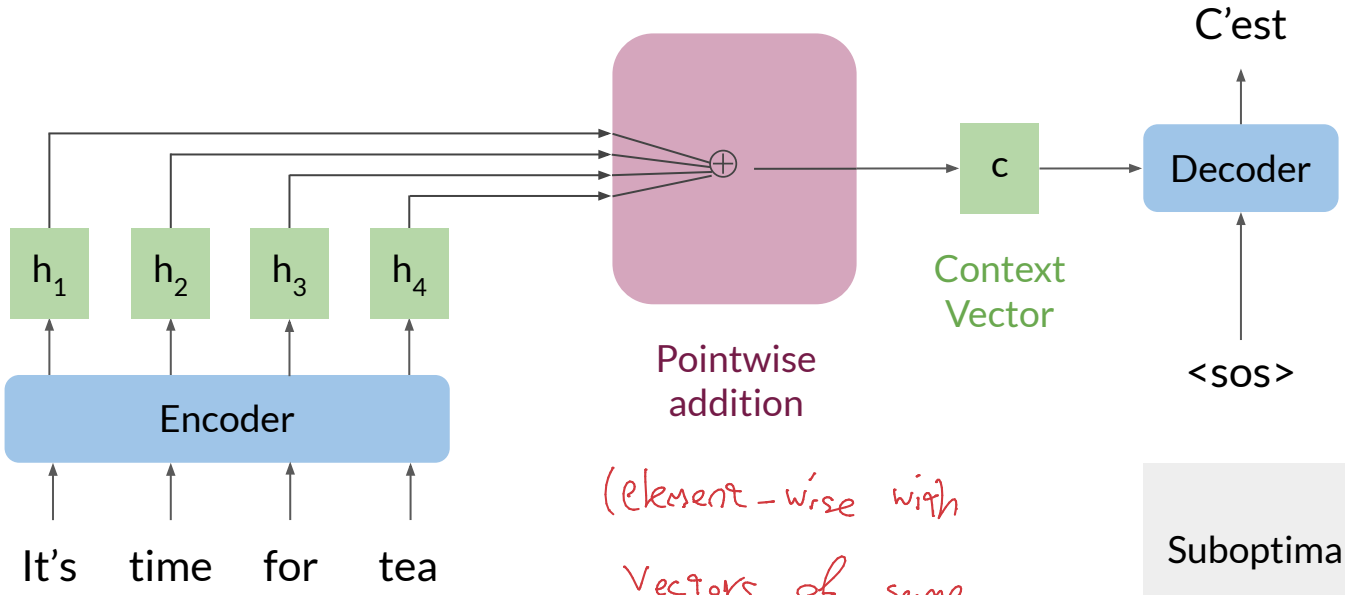
Traditional seq2seq models



How to use all the hidden states?



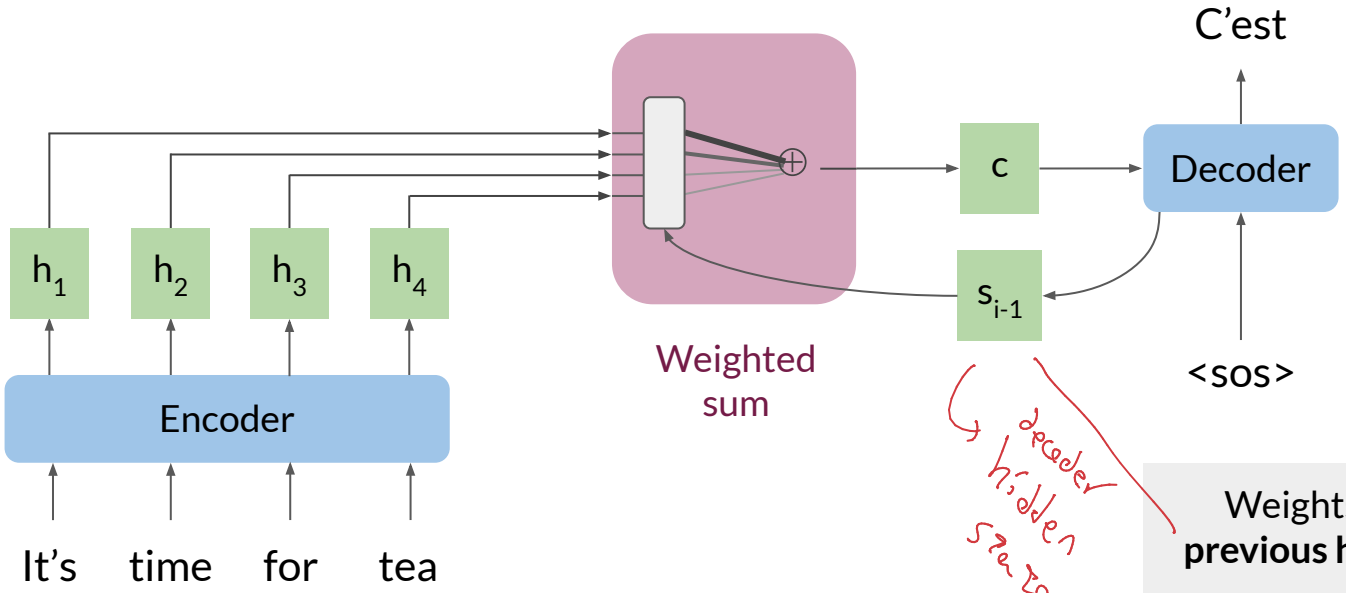
How to use all the hidden states?



(element-wise with vectors of same size)

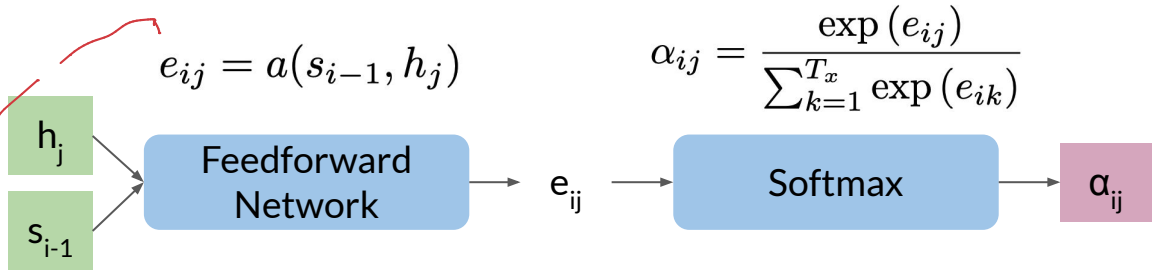
Suboptimal use of information

How to use all the hidden states?



Weights depend on the **previous hidden state** in the decoder

The attention layer in more depth



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

Context Vec ← c_i = Expected Value along word alignments

$\alpha_{i1}h_1 + \alpha_{i2}h_2 + \alpha_{i3}h_3 + \cdots + \alpha_{iM}h_M \rightarrow c_i$

Context Vector is an expected value

→ Score of how well the inputs around j match the expected output

at i more match → ↑ score

Recurrent models typically take in a sequence in the order it is written and use that to output a sequence. Each element in the sequence is associated with its step in computation time t. (i.e. if a word is in the third element, it will be computed at t_3). These models generate a sequence of hidden states h_t, as a function of the previous hidden state h_{t-1} and the input for position t.

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. In other words, 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, then 1) it will take a long time for you to process it and 2) you will lose a good amount of information mentioned earlier in the text as you approach the end.

Therefore, attention mechanisms have become critical for sequence modeling in various tasks, allowing modeling of dependencies without caring too much about their distance in the input or output sequences.

PS: Here's an article from The Atlantic that discusses the famous JFK speech containing the words "Ich bin ein Berliner," the example you saw in the Alignment video.

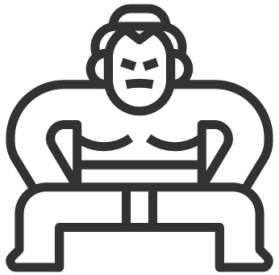
https://www.theatlantic.com/magazine/archive/2013/08/the-real-meaning-of-ich-bin-ein-berliner/309500/ (Putnam, 2013)

deeplearning.ai

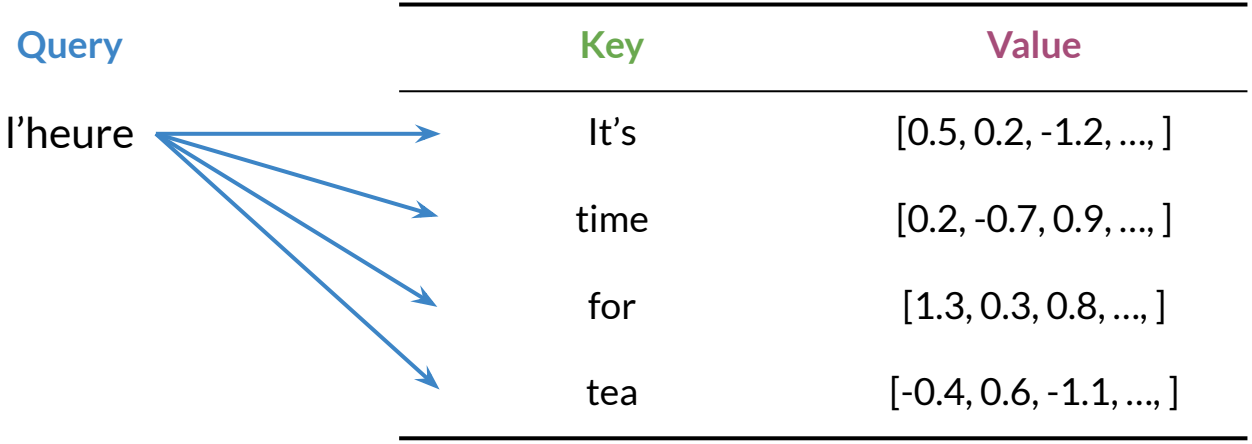
Queries, Keys, Values and Attention

Outline

- Queries, Keys, and Values
- Alignment

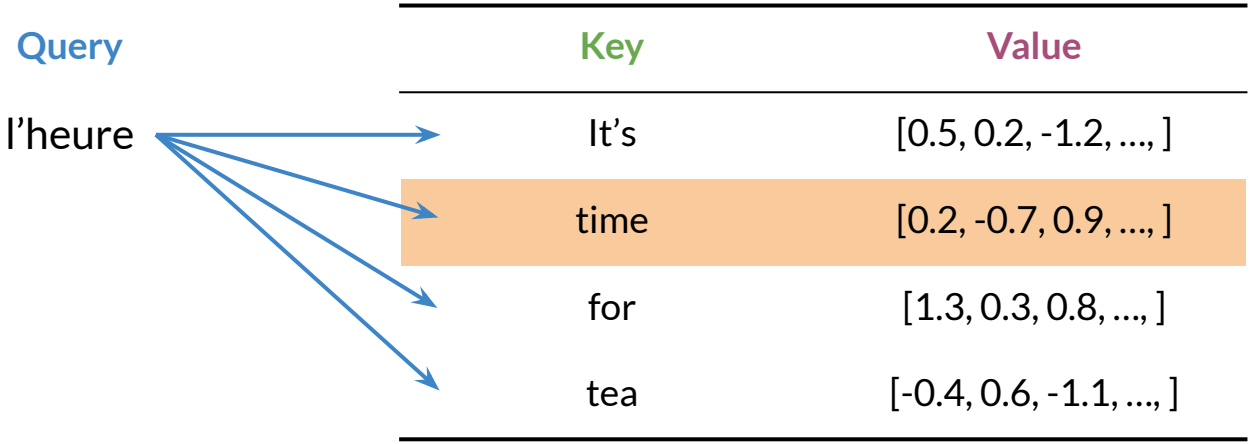


Queries, Keys, Values



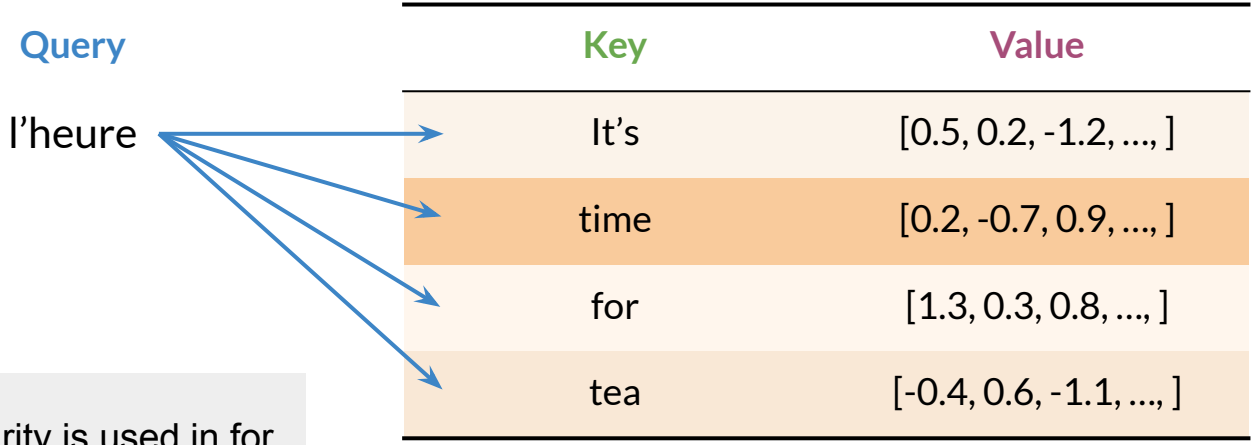
Query is matched to a key and the Value associated with that key is returned

Queries, Keys, Values



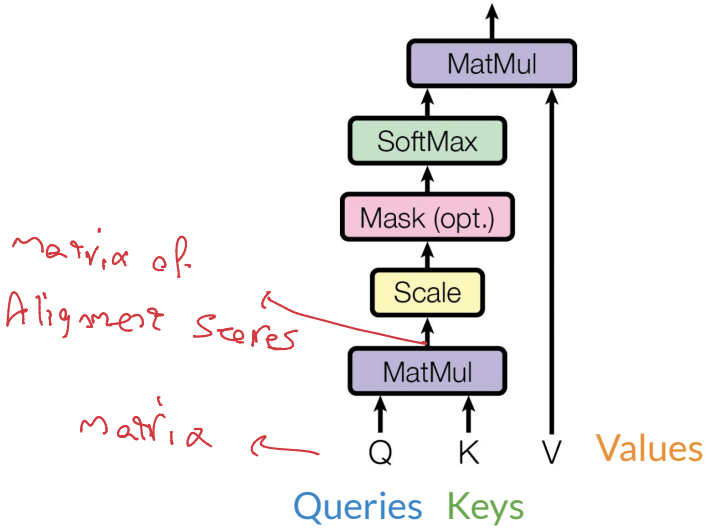
Queries, Keys, Values

the similarities between words is called alignment



Similarity is used in for weighted sum

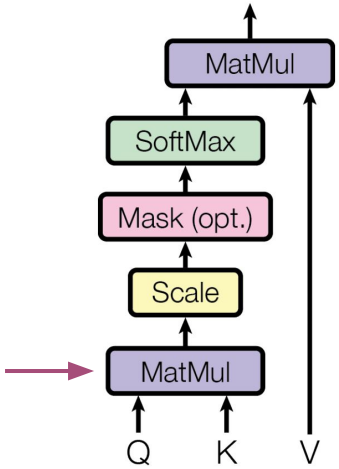
Scaled dot-product attention



(Vaswani et al., 2017)

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

Scaled dot-product attention

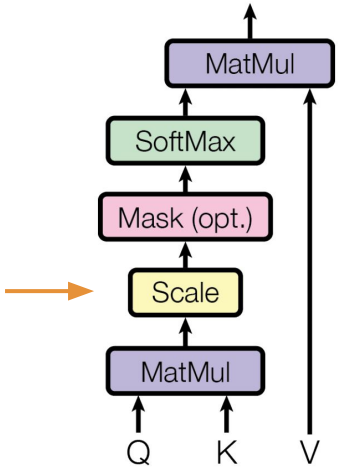


(Vaswani et al., 2017)

Alignment Scores = Similarity Between Q and K

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

Scaled dot-product attention



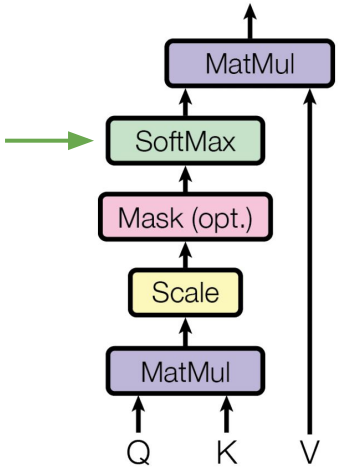
(Vaswani et al., 2017)

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

← Scale using the root of the key vector size

Improve model Performance of larger model sizes and could be seen as a regularization constant

Scaled dot-product attention



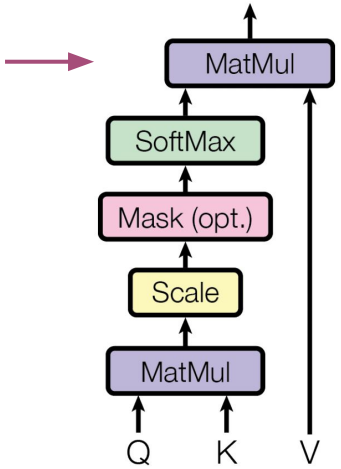
(Vaswani et al., 2017)

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

Weights for the weighted sum

Scaled dot-product attention → No NN & 2 MatMul

highly optimized



(Vaswani et al., 2017)

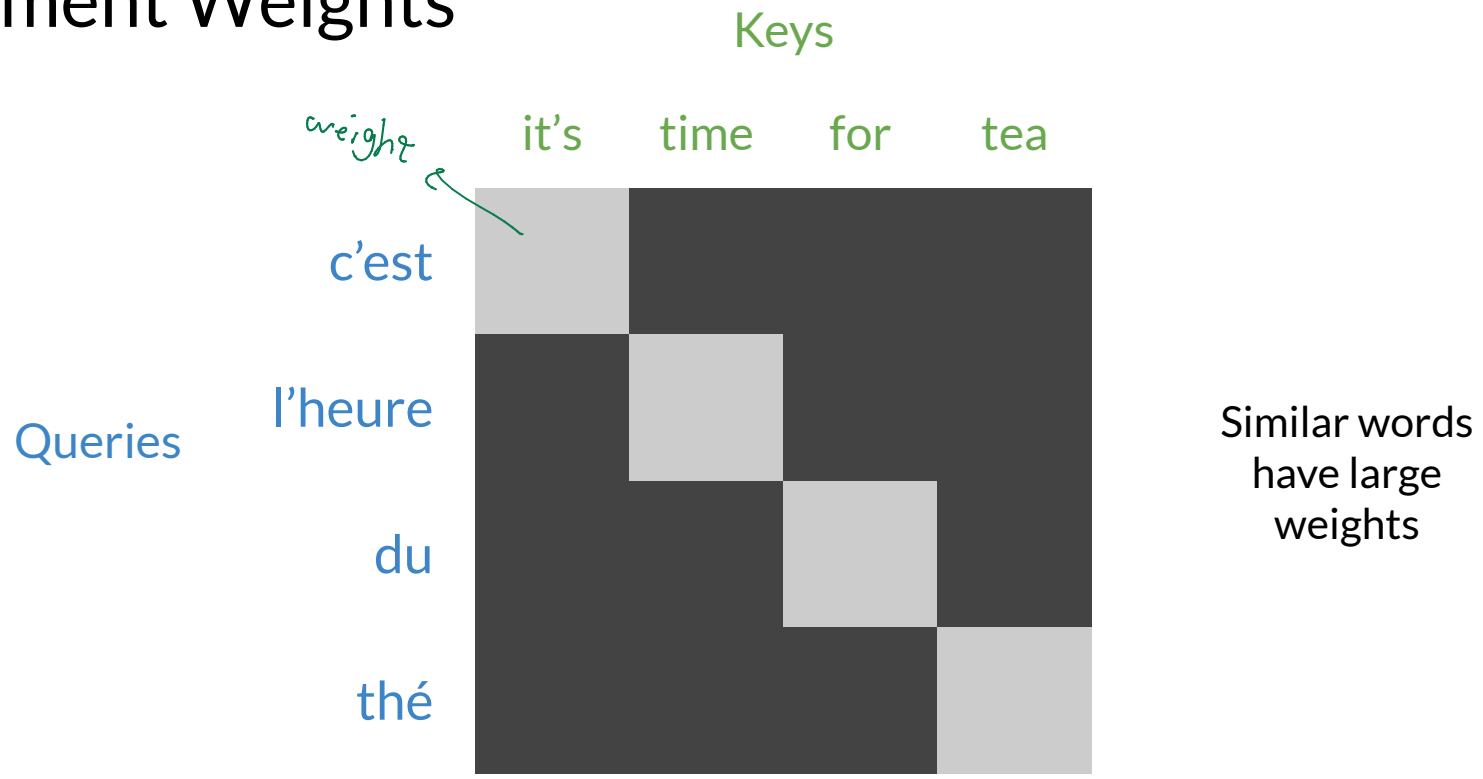
$$\text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

Weighted sum of values V

Attention vecs for each query

Just two matrix multiplications and a Softmax!

Alignment Weights

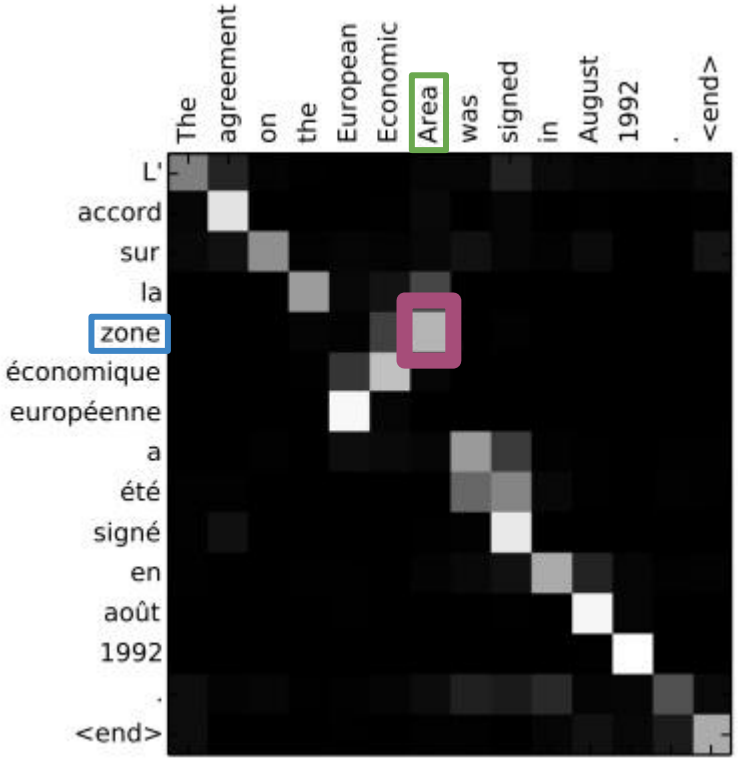


Similar words have large weights

Flexible attention

Works for languages with different grammar structures!

Zone & area are at dif Position but having same meaning



Bahdanau et al., 2015

Summary

- Attention is a layer that lets a model focus on what's important
- Queries, Values, and Keys are used for information retrieval inside the Attention layer
- Works for languages with very different grammatical structures

deeplearning.ai

Setup for machine translation

Data in machine translation

English	French
I am hungry!	J'ai faim!
...	...
I watched the soccer game.	J'ai regardé le match de football.

Attention! (pun intended) Assignment dataset is not as squeaky-clean as this example and contains some Spanish translations.

Machine translation setup

- Use pre-trained vector embeddings
- Otherwise, initially represent words with a one-hot vectors
- Keep track of index mappings with word2ind and ind2word dictionaries
- Add end of sequence tokens: <EOS>
- Pad the token vectors with zeros

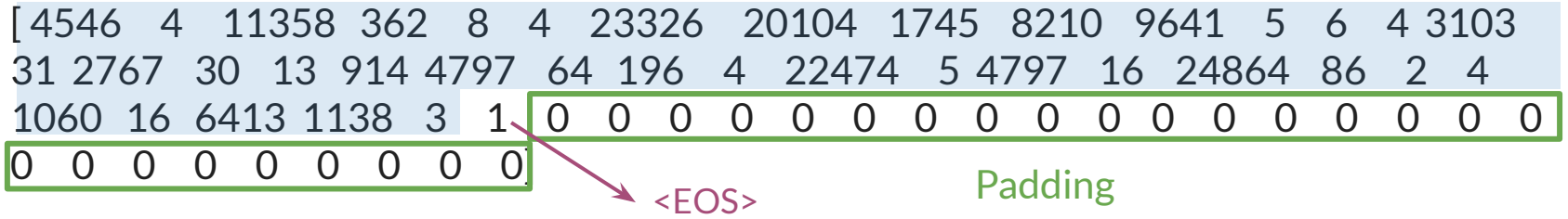
To match the length of the longest seq

Preparing to Translate to English

ENGLISH SENTENCE:

Both the ballpoint and the mechanical pencil in the series are equipped with a special mechanism: when the twist mechanism is activated, the lead is pushed forward.

TOKENIZED VERSION OF THE ENGLISH SENTENCE:



English to French

FRENCH TRANSLATION:

Le stylo à bille et le porte-mine de la série sont équipés d'un mécanisme spécial: lorsque le mécanisme de torsion est activé, le plomb est poussé vers l'avant.

TOKENIZED VERSION OF THE FRENCH TRANSLATION:

[7 29587 9 18240 8 7 420 5 3440 2 6 156 39 7941 14 19 5548 2648
562 7 5548 2 23194 18 20114 1 7 5695 18 8865 149 12 137 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

Padding

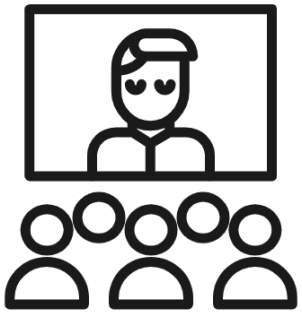
<EOS>

deeplearning.ai

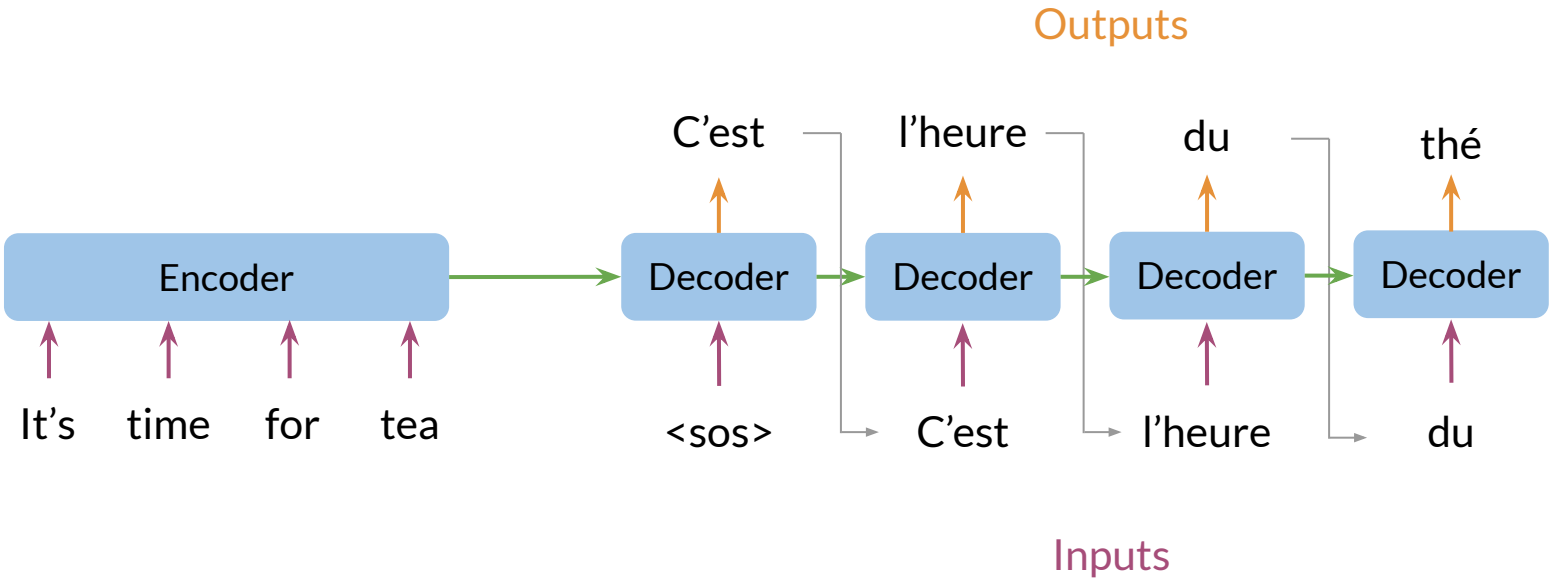
Teacher Forcing

Outline

- Training for NMT
- Teacher forcing

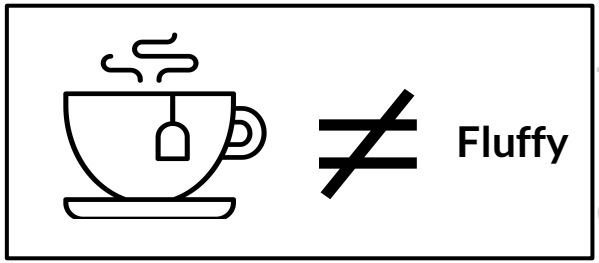


Traditional seq2seq models



Training seq2seq models

Sum the loss to get total loss (1)



≠ Fluffy

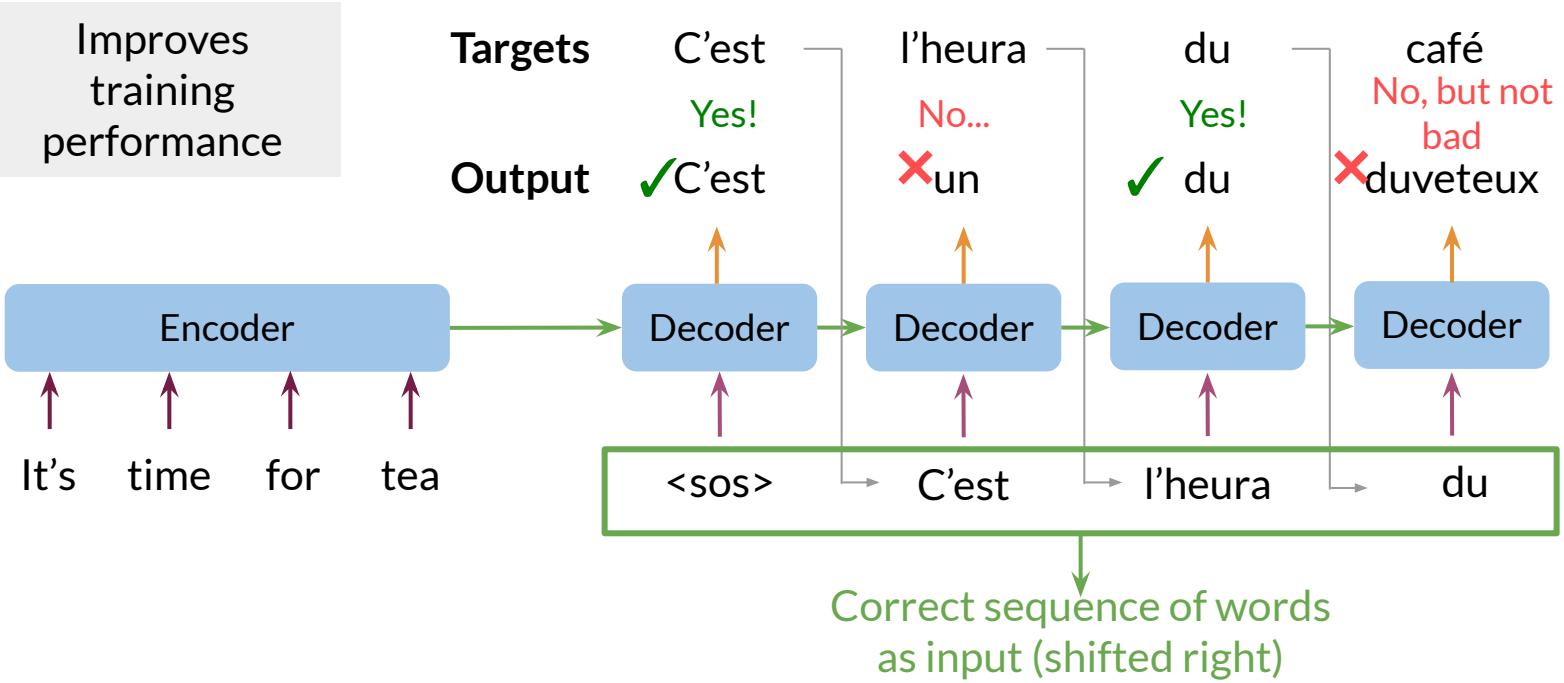
Errors from early steps propagate

(1) this doesn't work well: in early stages of training the model is naive. It'll make wrong prediction early in the seq

this Problem compounds as the model keeps making wrong Predictions and the Translated seq gets further & further from the target seq

Teacher Forcing → makes training faster

Improves training performance

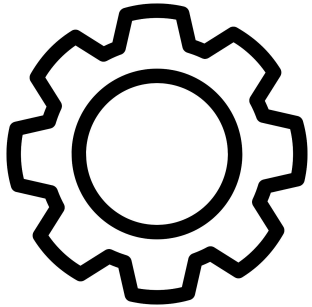


deeplearning.ai

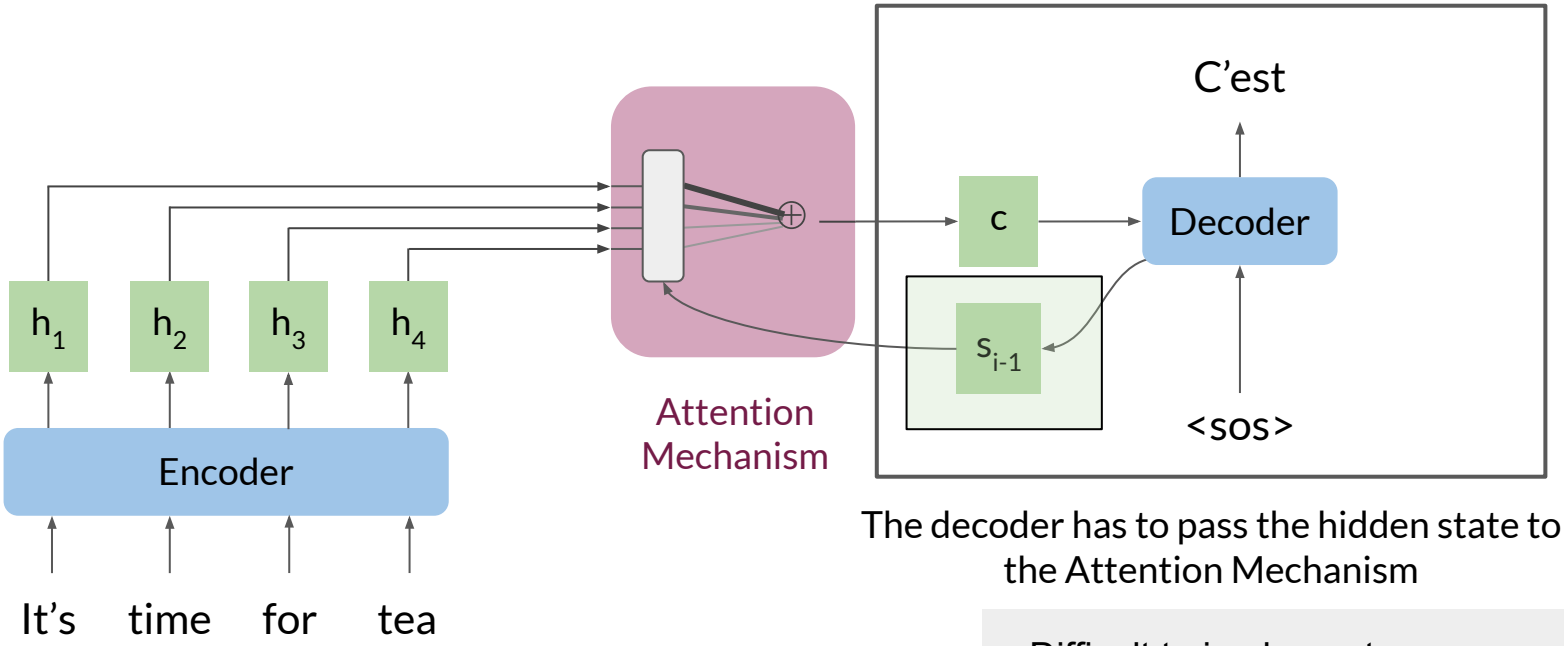
NMT Model with Attention

Outline

- How everything fits together
- NMT model in detail



NMT Model

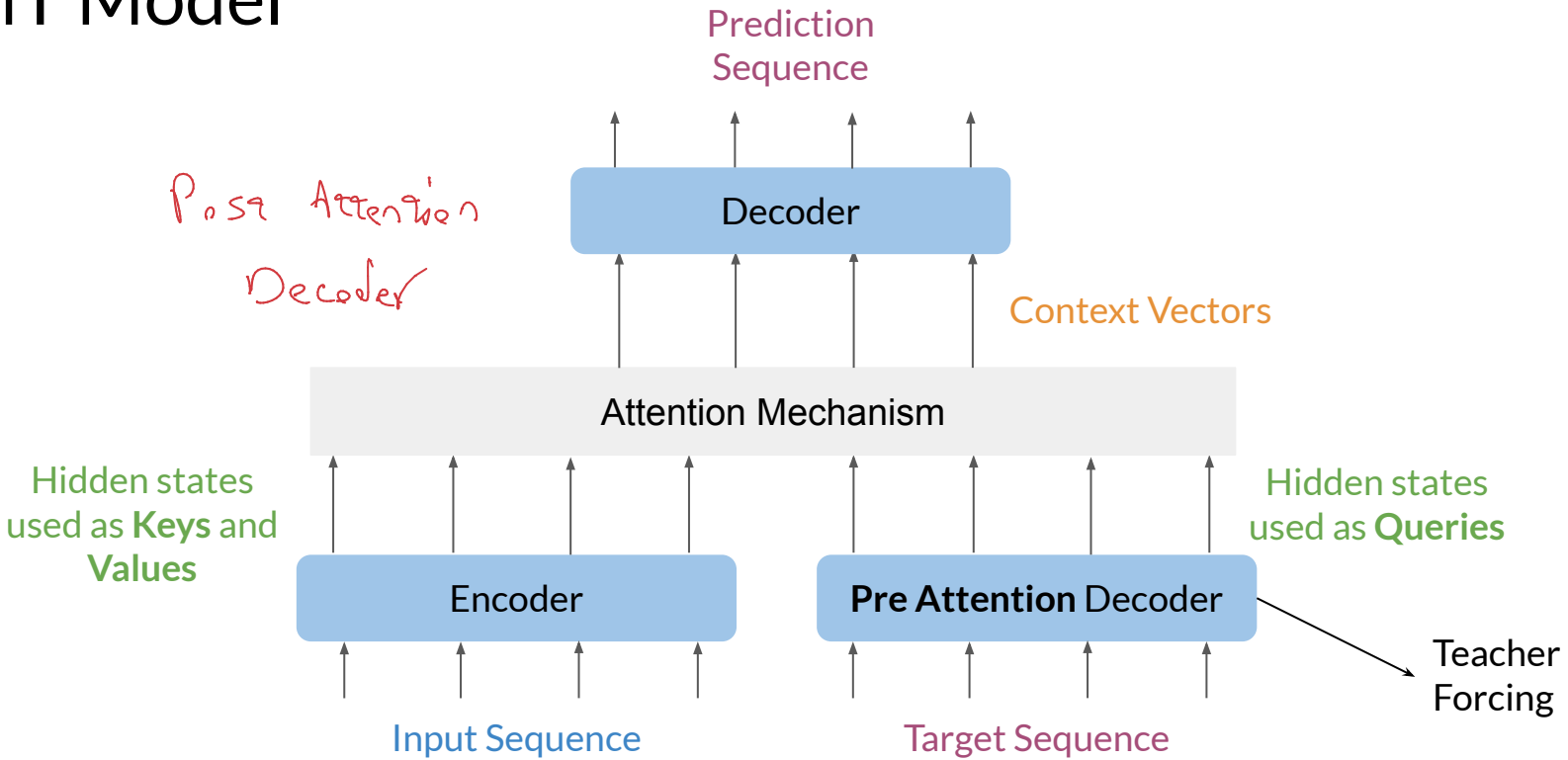


The decoder has to pass the hidden state to the Attention Mechanism

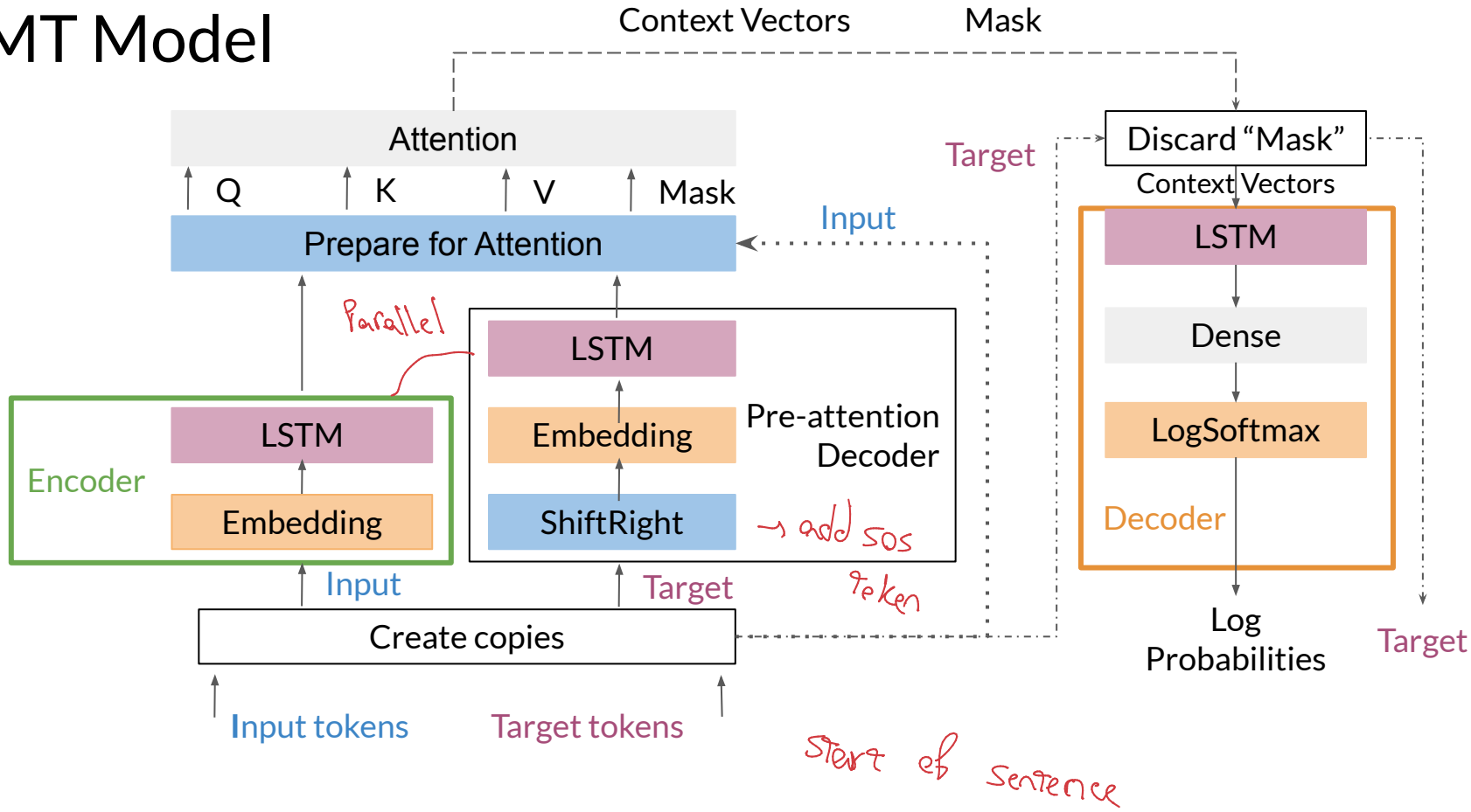
Difficult to implement, so a **pre-attention decoder** is introduced.

to Provide hidden state

NMT Model


Prediction Sequence
Post Attention Decoder
Decoder
Context Vectors
Attention Mechanism
Hidden states used as **Keys** and **Values**
Hidden states used as **Queries**
Encoder
Pre Attention Decoder
Teacher Forcing
Input Sequence
Target Sequence

NMT Model


Context Vectors
Mask
Attention
Q
K
V
Mask
Prepare for Attention
Parallel
LSTM
Embedding
ShiftRight
Pre-attention Decoder
→ add SOS token
Encoder
LSTM
Embedding
Input
Target
Create copies
Input tokens
Target tokens
Start of Sentence
Target
Discard "Mask"
Context Vectors
Input
LSTM
Dense
LogSoftmax
Decoder
Log Probabilities
Target

deeplearning.ai

BLEU Score

BLEU Score

BiL**i**ngual **E**valuation **U**nderstudy

Compares candidate translations to reference (human) translations

The closer to **1**, the better

BLEU Score

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

How many words from the candidate appear in the reference translations?

BLEU Score

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

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

# of words in candidate

A model that always outputs common words will do great!

BLEU Score (Modified)

: After you find a word from the Candidates in one or more of the References, stop considering that word from the reference for following word in the candidates

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!

or delete

(Exhaust the words in the references after you match them with a word in the Candidates)

BLEU score is great, but...

Consider the following:

- BLEU doesn't consider semantic meaning
- BLEU doesn't consider sentence structure:

"Ate I was hungry because!"

↳ Perfect BLEU score

deeplearning.ai

ROUGE-N Score

ROUGE

Recall-**O**riented **U**nderstudy for **G**isting **E**valuation

Compares candidates with reference (human) translations

Multiple versions for this metric

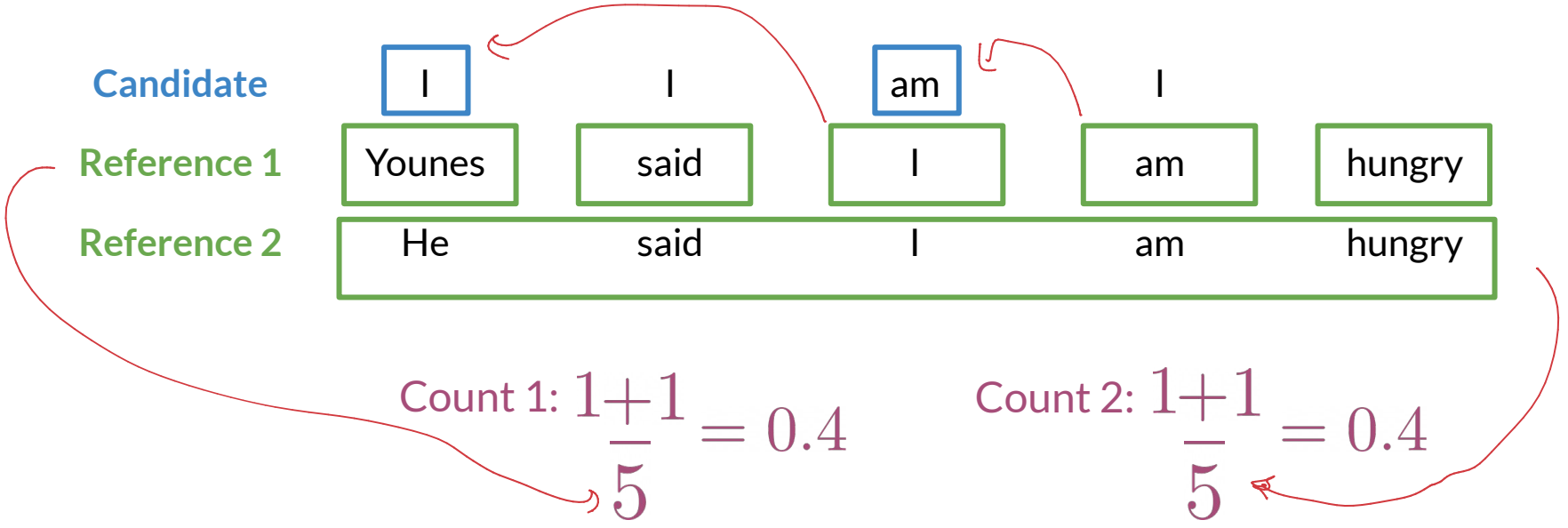
Initially developed to eval the quality of the machine summarized texts

ROUGE-N : get the counts of N-gram overlaps between the candidates and the reference translations

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

How many words from the reference appear in the candidate translations?

ROUGE-N


Candidate
I
I
am
I
Reference 1
Younes
said
I
am
hungry
Reference 2
He
said
I
am
hungry
Count 1: $\frac{1+1}{5} = 0.4$
Count 2: $\frac{1+1}{5} = 0.4$

ROUGE-N, BLEU and F1 score → don't consider the sentence structure and semantics

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

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

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

deeplearning.ai

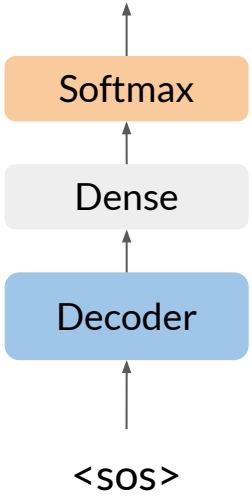
Sampling and Decoding

Outline

- Random sampling
- Temperature in sampling
- Greedy decoding

Seq2Seq model

Words	de	la	le	et	à	...
$P(w_i)$	0.02	0.04	0.1	0.005	0.08	...



Probability distribution over words in target language vocab and symbols

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

am	full	hungry	I	the
0.05	0.3	0.15	0.25	0.25

Often a little too random for accurate translation!

Solution: Assign more weight to more probable words, and less weight to less probable words.

Temperature [0-1]

Can control for more or less randomness in predictions

Lower temperature setting : More confident, conservative network

Higher temperature setting : More excited, random network → more mistakes

deeplearning.ai

Beam Search

Beam search decoding

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

Calculate probability of multiple possible sequences

Beam search

Beam search decoding → use a lot of memory

Probability of multiple possible sequences at each step given the outputs of the previous time step

Beam width B determines number of sequences you keep

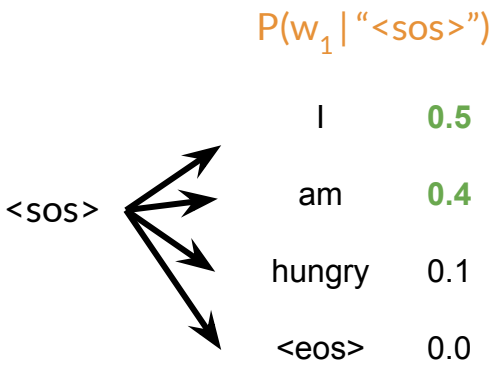
↳ B most probable seqs and drop others

Until all B most probable sequences end with <EOS>

Beam search with **B=1** is **greedy decoding**.

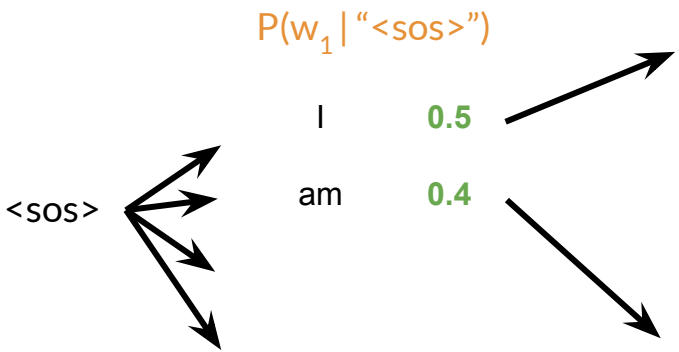
Beam search example

B = 2



Beam search example

B = 2

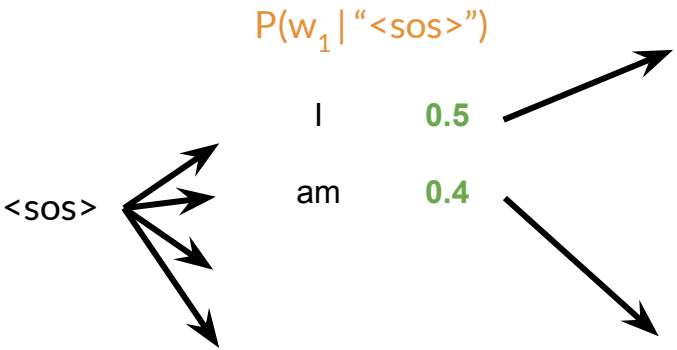


$P(w_2 \mid \text{“I”})$		$P(w_2 \mid \text{“I”})P(\text{“I”})$	
I	0.1	I I	0.05
am	0.5	I am	**0.25**
hungry	0.3	I hungry	0.15
<eos>	0.1	I <eos>	0.05

$P(w_2 \mid \text{“am”})$		$P(w_2 \mid \text{“am”})P(\text{“am”})$	
I	0.7	am I	**0.28**
am	0.05	am am	0.02
hungry	0.2	am hungry	0.08
<eos>	0.05	am <eos>	0.02

Beam search example

B = 2

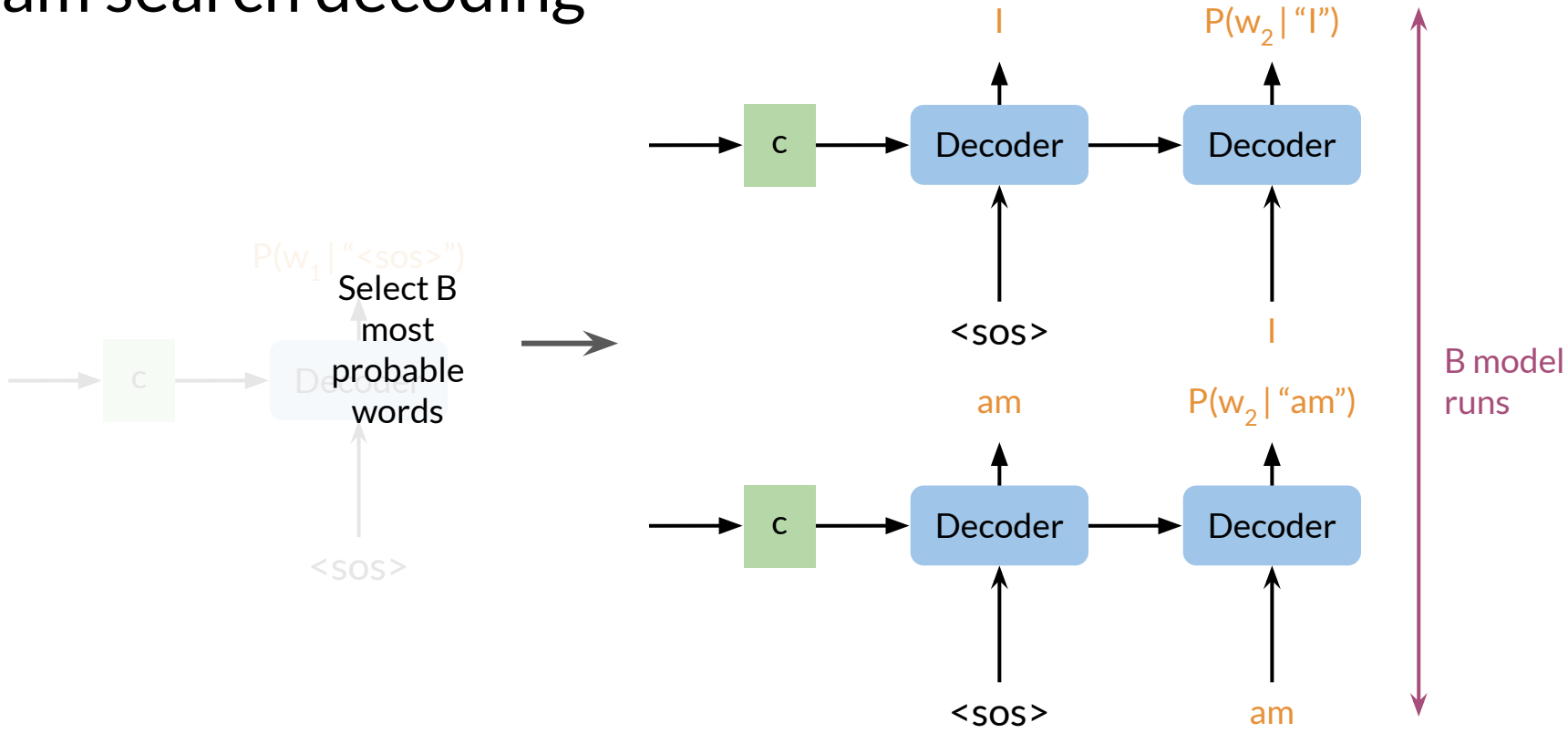


$P(w_2 \mid \text{“I”})$		$P(w_2 \mid \text{“I”})P(\text{“I”})$		$P(w_3 \mid \text{“I am”})$	
I	0.1	I I	0.05	I	0.1
am	0.5	I am	**0.25**	am	0.05
hungry	0.3	I hungry	0.15	hungry	0.6
<eos>	0.1	I <eos>	0.05	<eos>	0.3

$P(w_2 \mid \text{“am”})$		$P(w_2 \mid \text{“am”})P(\text{“am”})$		$P(w_3 \mid \text{“Am I”})$	
I	0.7	am I	**0.28**	I	0.15
am	0.05	am am	0.02	am	0.15
hungry	0.2	am hungry	0.08	hungry	0.5
<eos>	0.05	am <eos>	0.02	<eos>	0.2

Process steps when the model predicts an <EOS> token for all B most probable seqs - at end output = seq with highest prob

Beam search decoding



Problems with beam search

Penalizes long sequences, so you should normalize by the sentence length

Computationally expensive and consumes a lot of memory

deeplearning.ai

Minimum Bayes Risk

Minimum Bayes Risk (MBR)

- Generate several candidate translations
- Assign a similarity to every pair using a similarity score (such as ROUGE!)

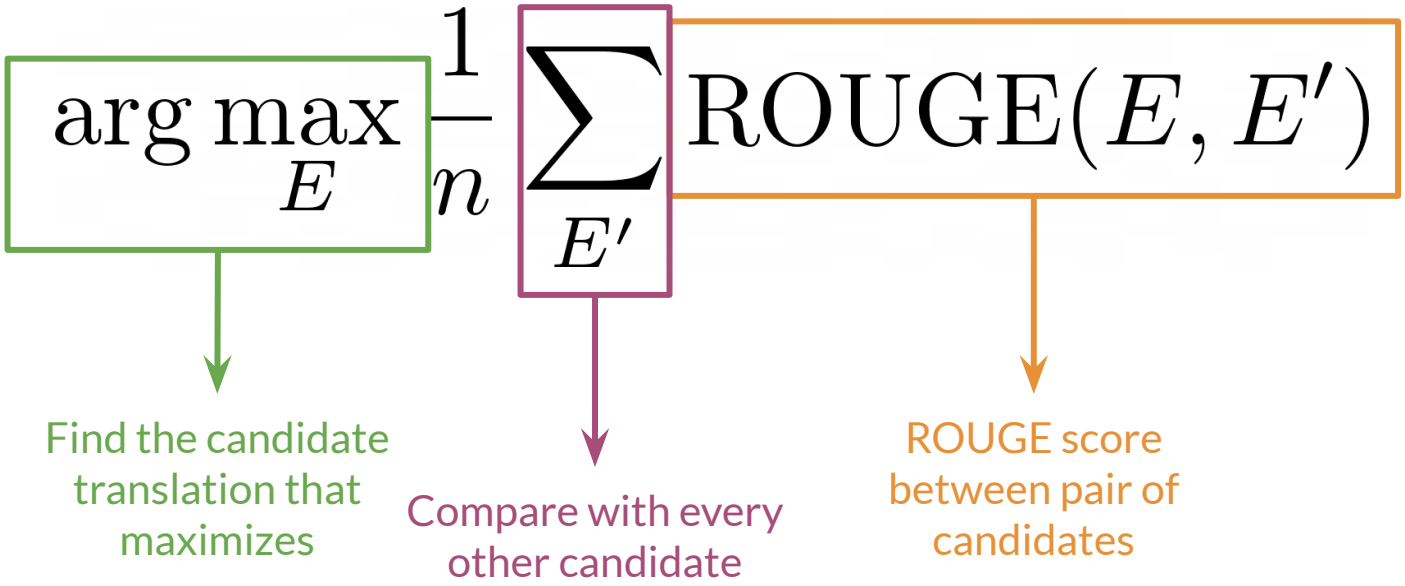
we can use loss func as well

- Select the sample with the highest average similarity

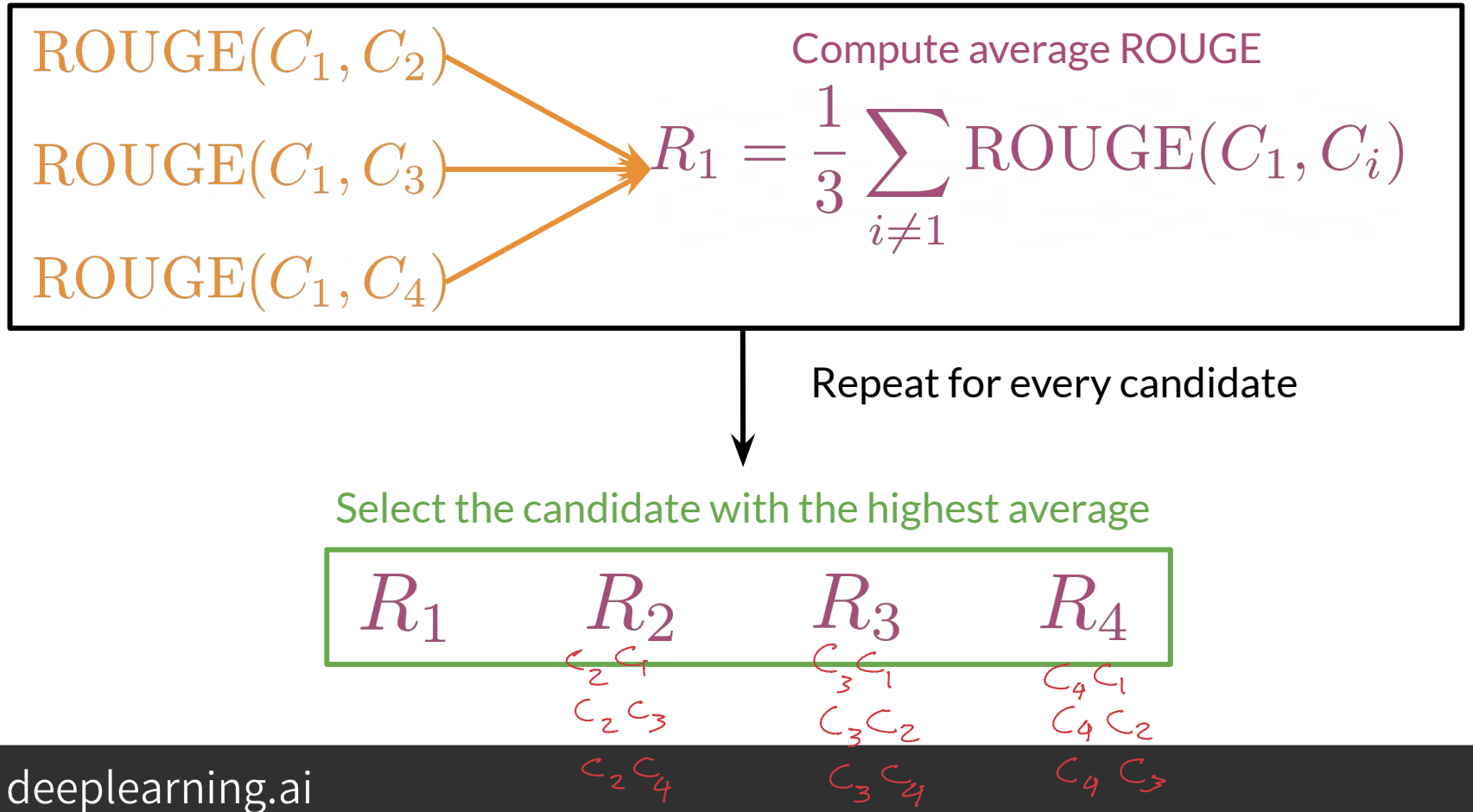
or lowest loss

* finding a consensus between all candidate translations

Minimum Bayes Risk (MBR)



Example: MBR Sampling



Summary

- Compare several candidate translations
- Choose candidate with highest average similarity
- Better performance than random sampling and greedy decoding

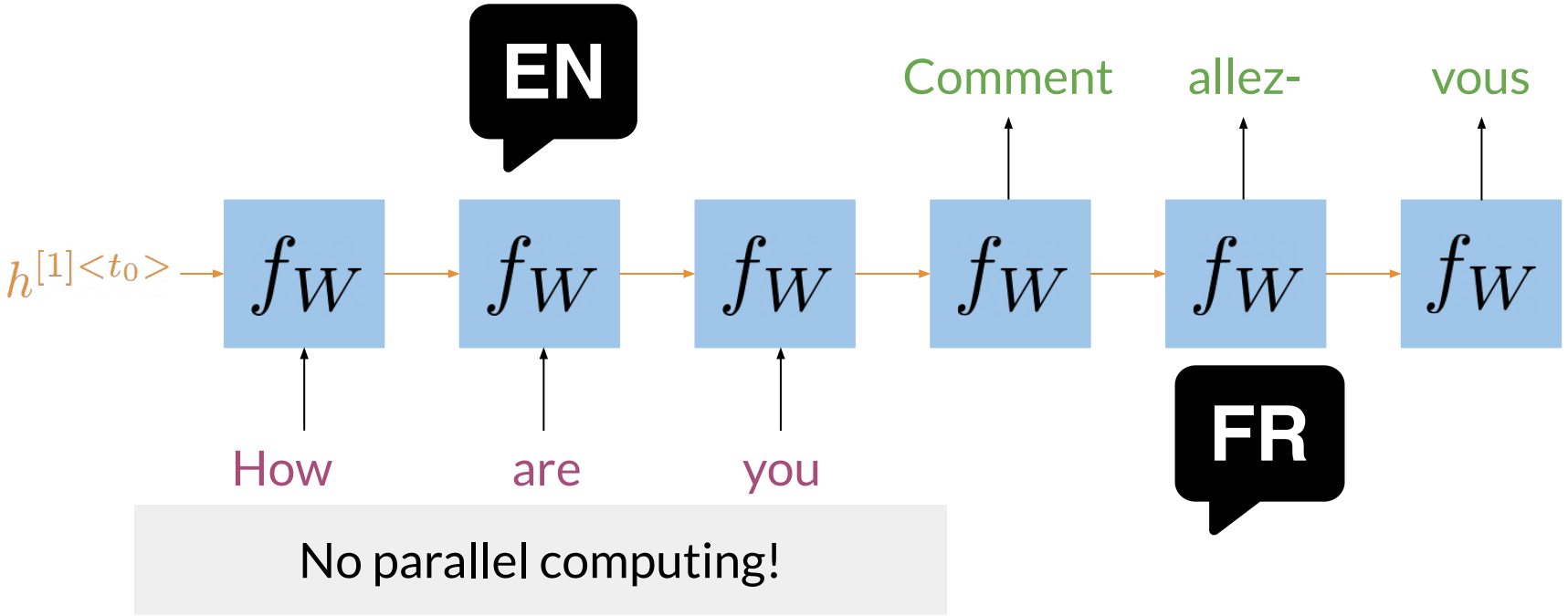
deeplearning.ai

Transformers vs RNNs

Outline

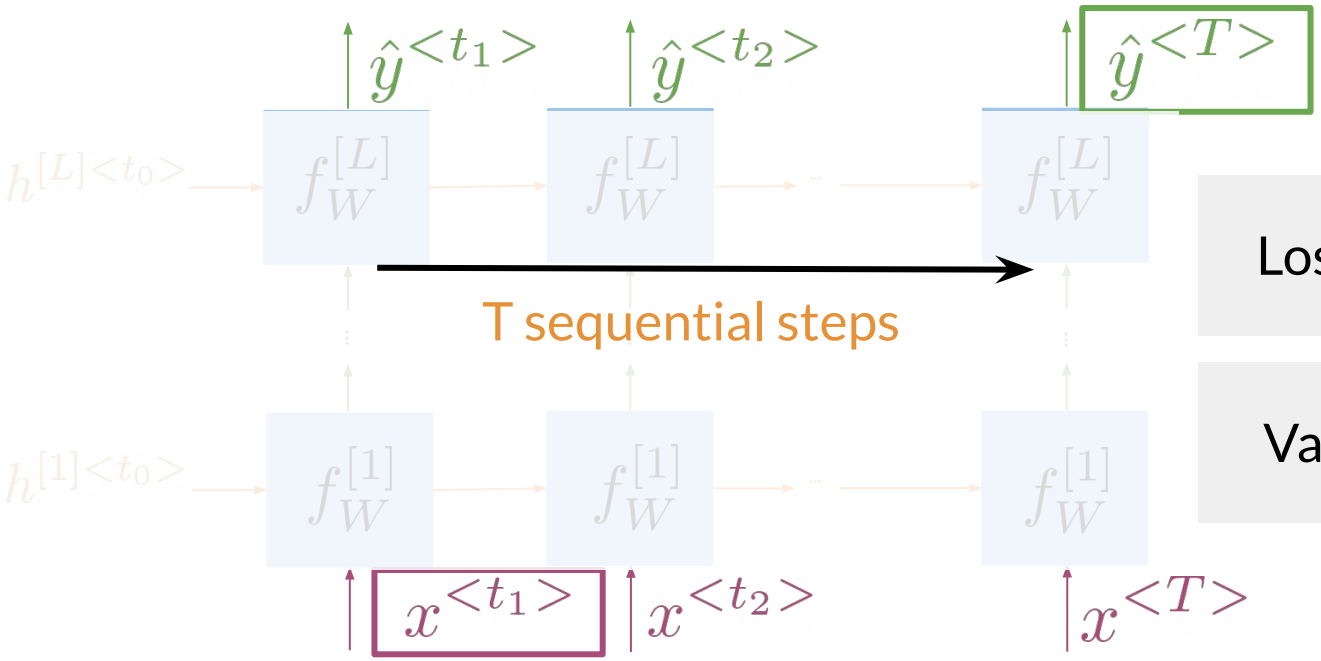
- Issues with RNNs
- Comparison with Transformers

Neural Machine Translation



No parallel computing!

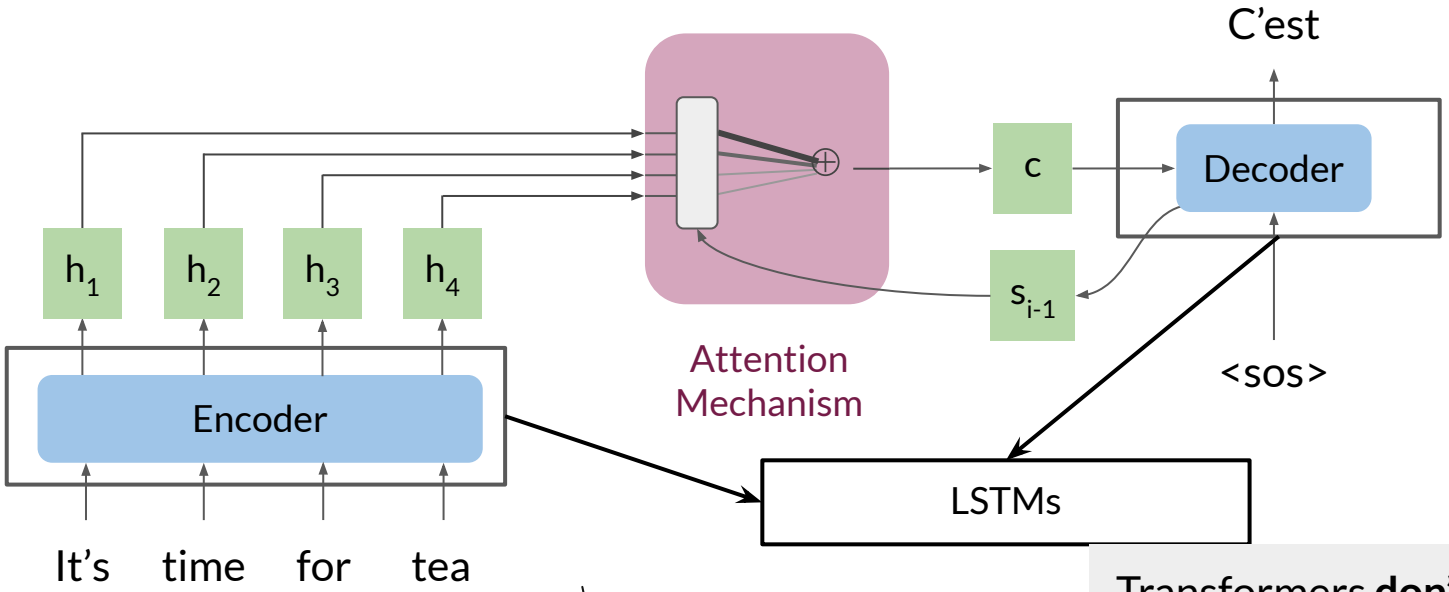
Seq2Seq Architectures



Loss of information

Vanishing gradient

RNNs vs Transformer: Encoder-Decoder

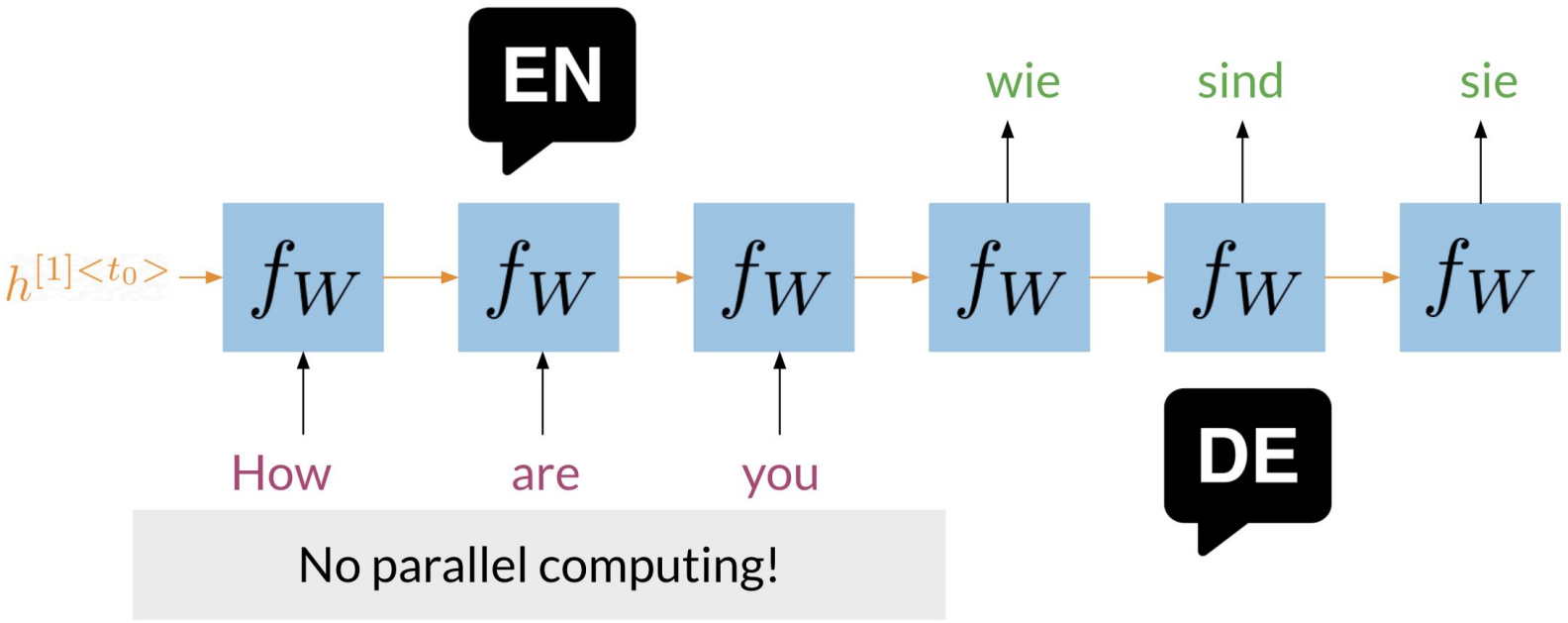


We used lstm for encoder & decoder last week

but

Attention is all you need

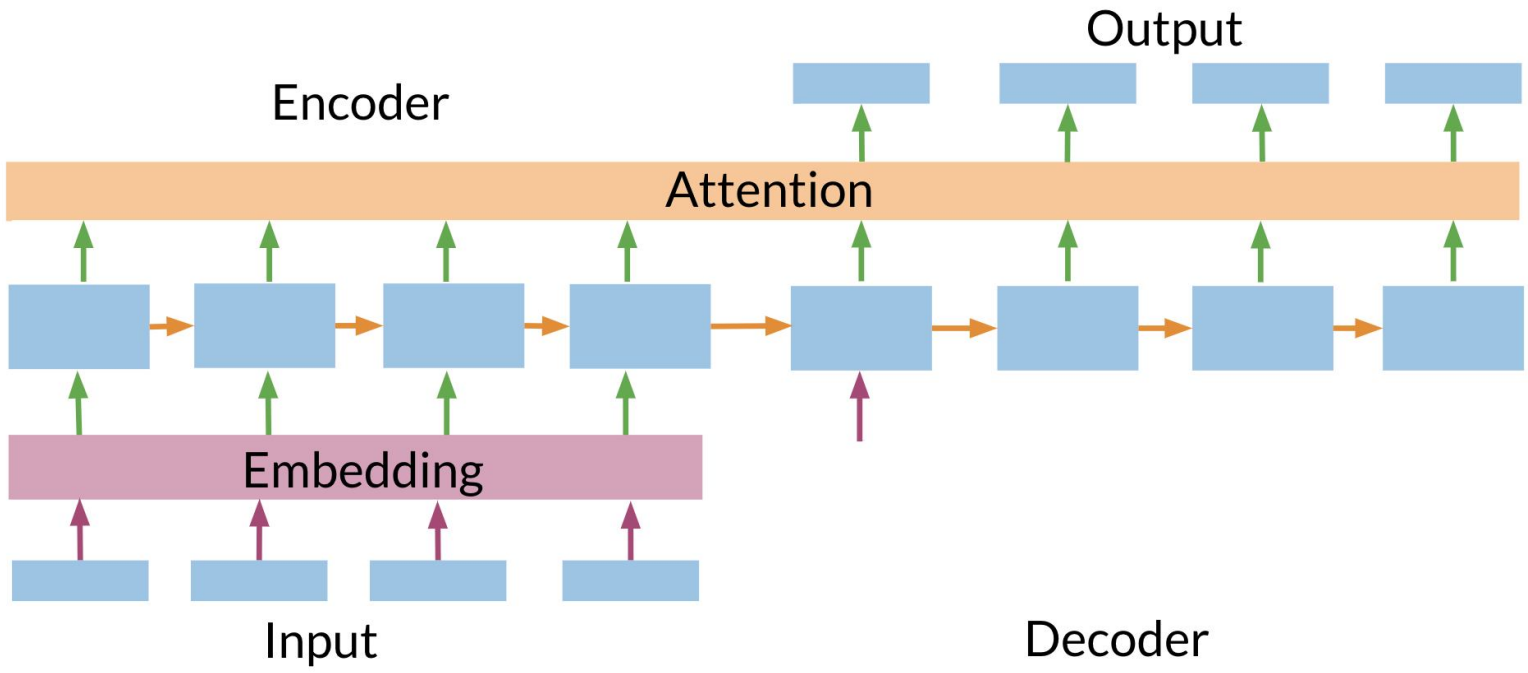
Transformers **don't** use RNNs, such as LSTMs or GRUs



In the image above, you can see a typical RNN that is used to translate the English sentence "How are you?" to its German equivalent, "Wie sind Sie?". One of the biggest issues with these RNNs, is that they make use of sequential computation. That means, in order for your code to process the word "you", it has to first go through "are" and then "you". Two other issues with RNNs are the:

- Loss of information: For example, it is harder to keep track of whether the subject is singular or plural as you move further away from the subject.
- Vanishing Gradient: when you back-propagate, the gradients can become really small and as a result, your model will not be learning much.

In contrast, transformers are based on attention and don't require any sequential computation per layer, only a single step is needed. Additionally, the gradient steps that need to be taken from the last output to the first input in a transformer is just one. For RNNs, the number of steps increases with longer sequences. Finally, transformers don't suffer from vanishing gradients problems that are related to the length of the sequences. Here is an image that might help you visualize it.



We are going to talk more about how the attention component works with transformers. So don't worry about it for now :)

deeplearning.ai

Transformers Overview

The Transformer Model

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

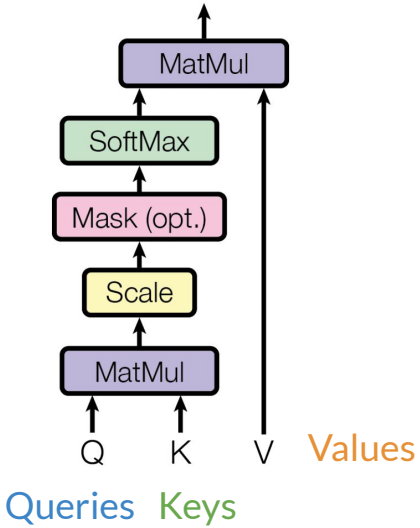
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

https://arxiv.org/abs/1706.03762

Scaled Dot-Product Attention

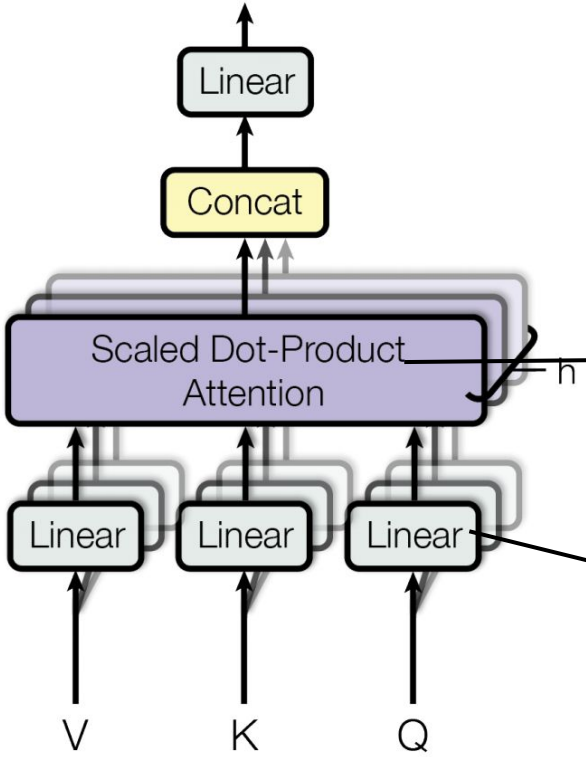


$$\text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

(Vaswani et al., 2017)

efficient in terms of computation and memory

Multi-Head Attention

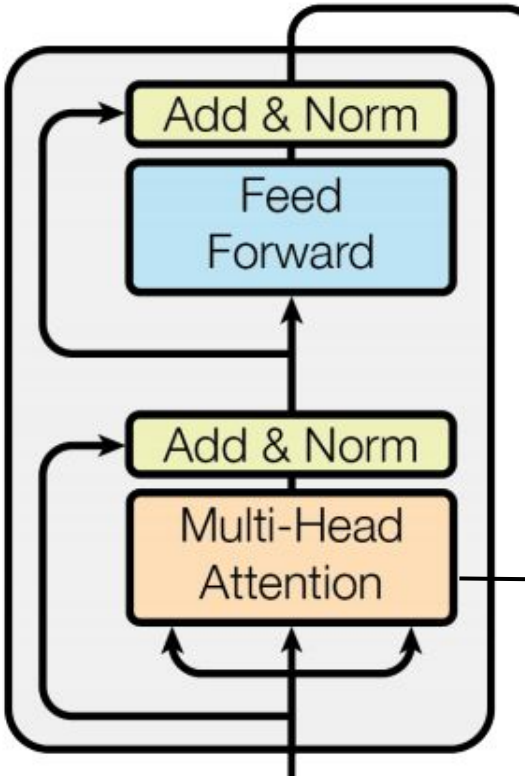


Scaled **dot-product attention multiple times** in parallel

Linear transformations of the input queries, keys and values

learnable Parameters

The Encoder



← Nx

N encoder layer

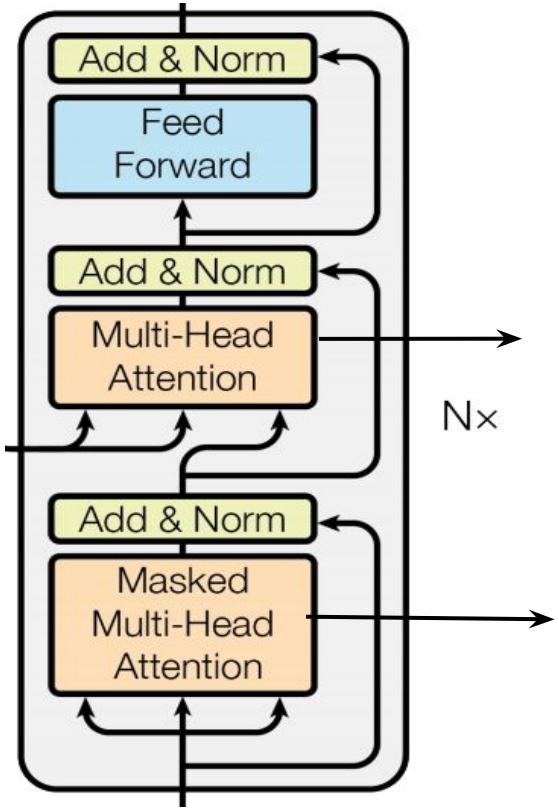
One encoder layer

Provides contextual representation of each item in the input sequence

Self-Attention

Every item in the input attends to every other item in the sequence (input)

The Decoder



Encoder-Decoder Attention

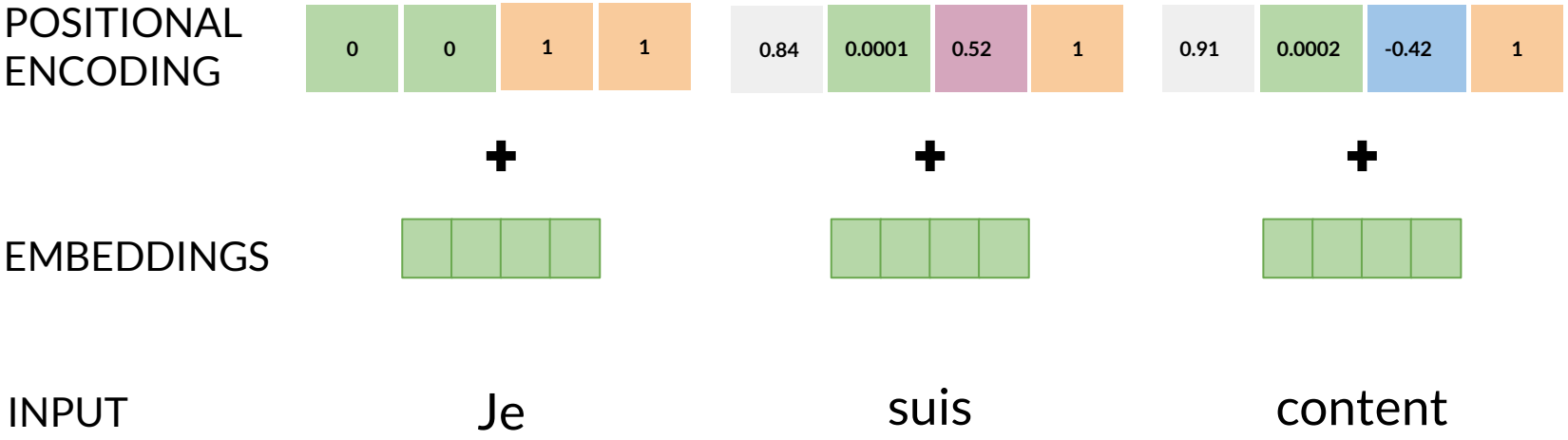
Every position from the decoder attents to the outputs from the encoder

Masked Self-Attention

Every position attends to **previous** positions

RNNs vs Transformer: Positional Encoding

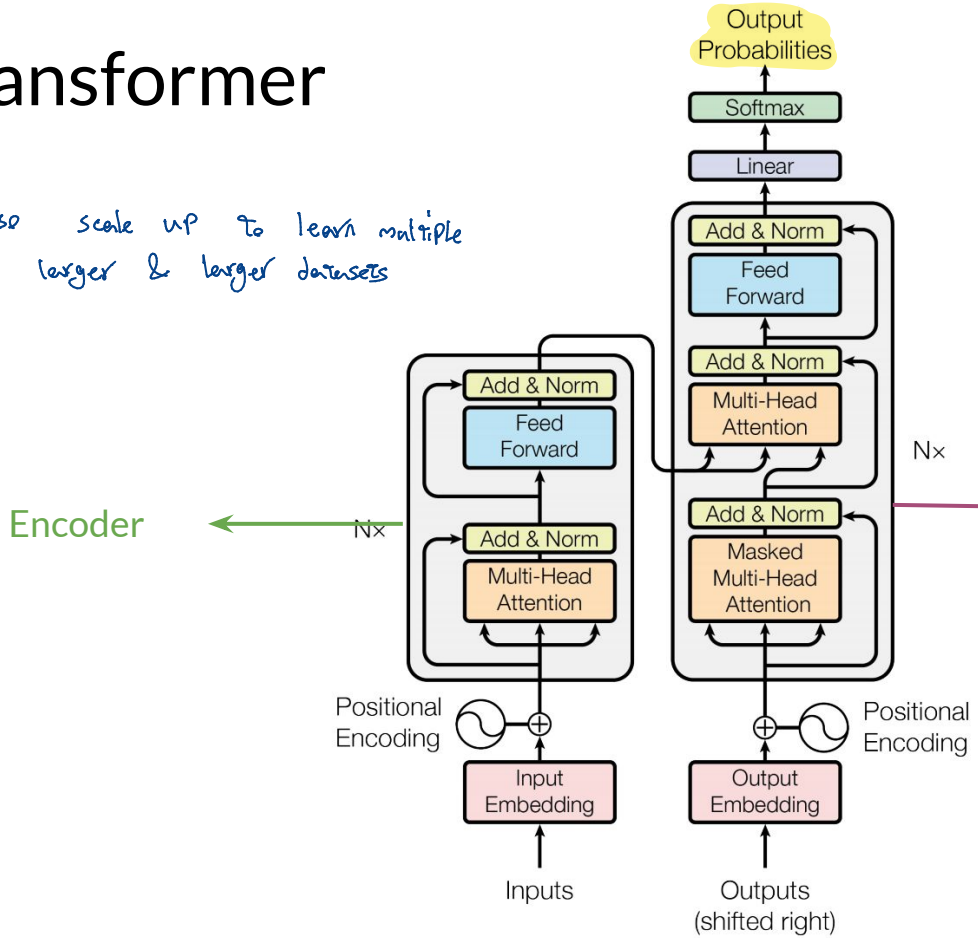
Encodes each input's position in the sequence



Pos encoding + Embedding → for every input word, we have info about its order and position

The Transformer

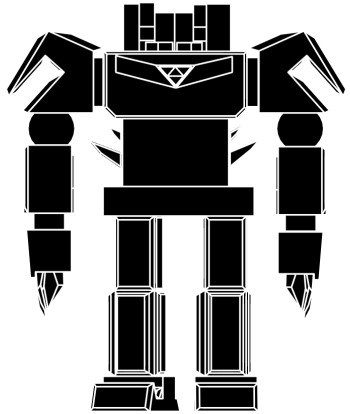
☆ It can also scale up to learn multiple tasks on larger & larger datasets



Easy to parallelize!

Summary

- In RNNs parallel computing is difficult to implement
- For long sequences in RNNs there is loss of information
- In RNNs there is the problem of vanishing gradient
- Transformers help with all of the above

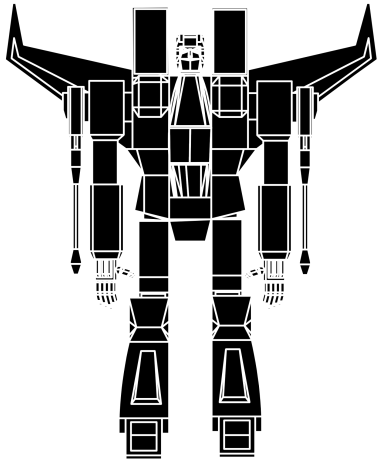


deeplearning.ai

Transformer Applications

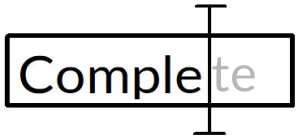
Outline

- Transformers applications in NLP
- Some Transformers
- Introduction to T5



Transformer NLP applications

Text summarization



Auto-Complete

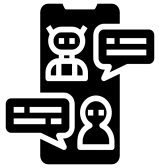
The	wind	blows	hard
Article	Noun	Verb	Adiective

Named entity recognition (NER)

Question answering (Q&A)

Translation

Chat-bots



Other NLP tasks

- Sentiment Analysis
- Market Intelligence
- Text Classification
- Character Recognition
- Spell Checking

State of the Art Transformers

Radford, A., et al. (2018)
Open AI

GPT-2: Generative Pre-training for Transformer

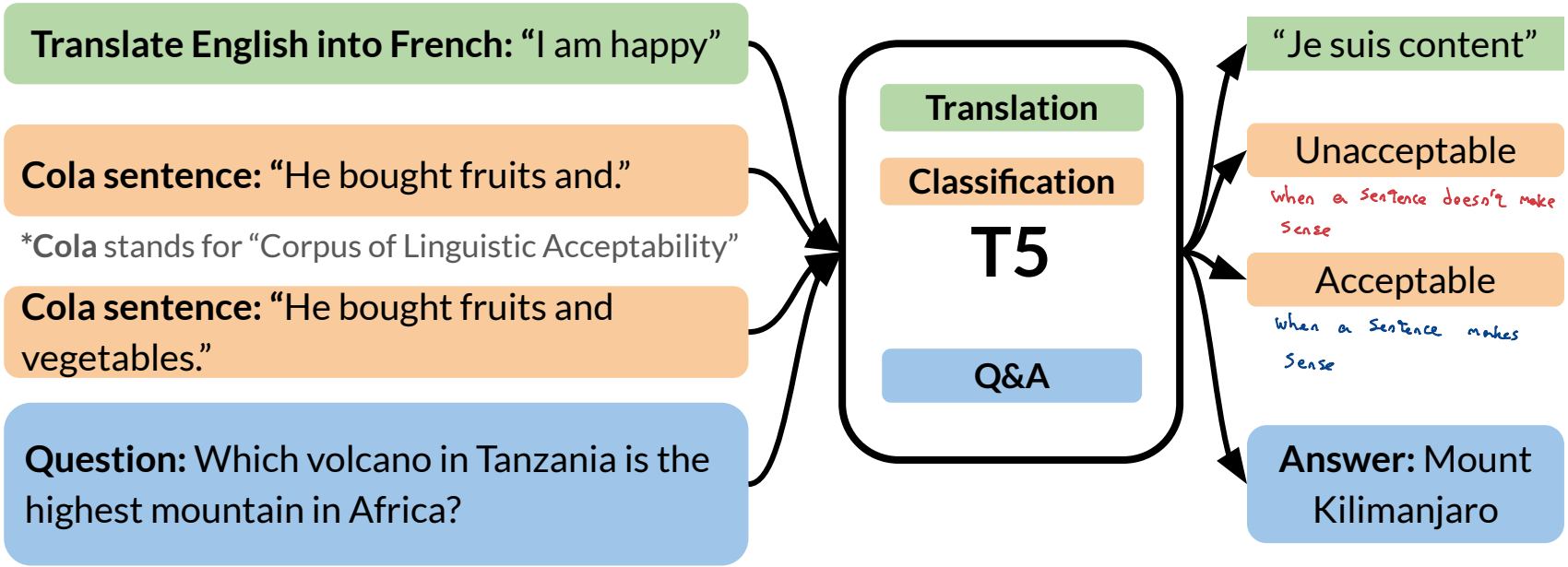
Devlin, J., et al. (2018)
Google AI Language

BERT:Bidirectional Encoder Representations from Transformers

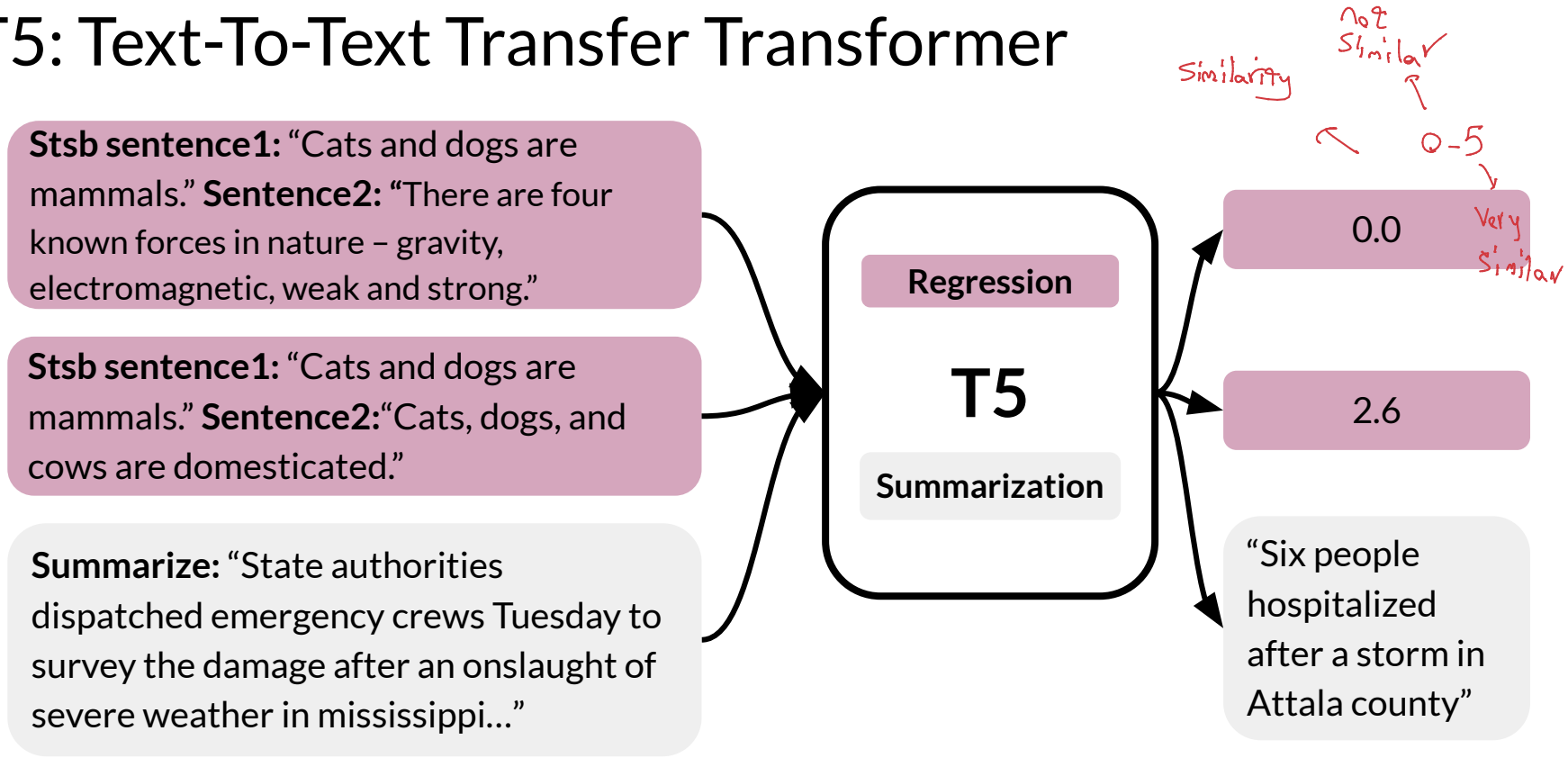
Colin, R., et al. (2019)
Google

T5: Text-to-text transfer transformer

T5: Text-To-Text Transfer Transformer



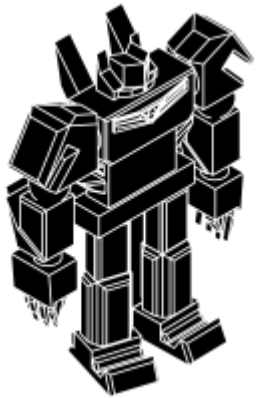
T5: Text-To-Text Transfer Transformer



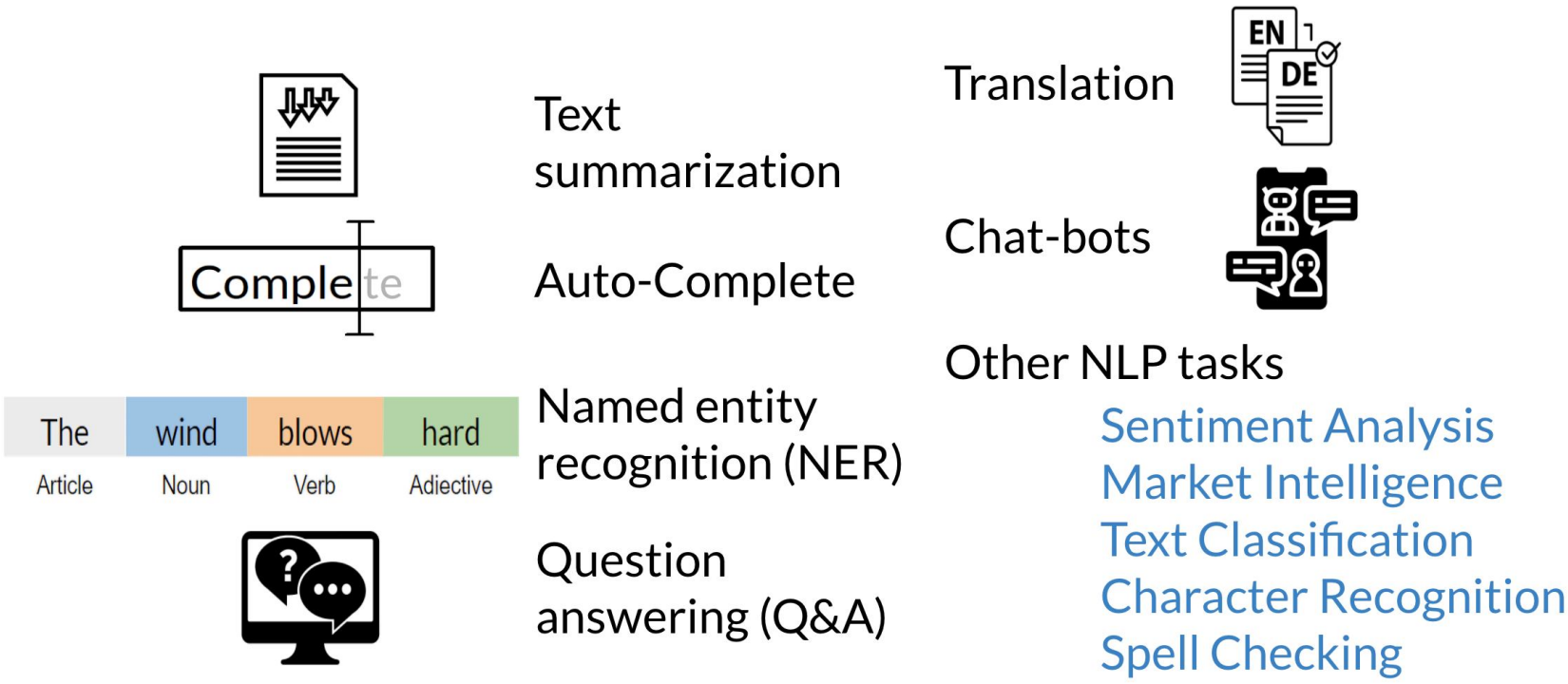
T5: Demo

Summary

- Transformers are suitable for a wide range of NLP applications
- Some transformers include GPT, BERT and T5
- T5 is a powerful multi-task transformer

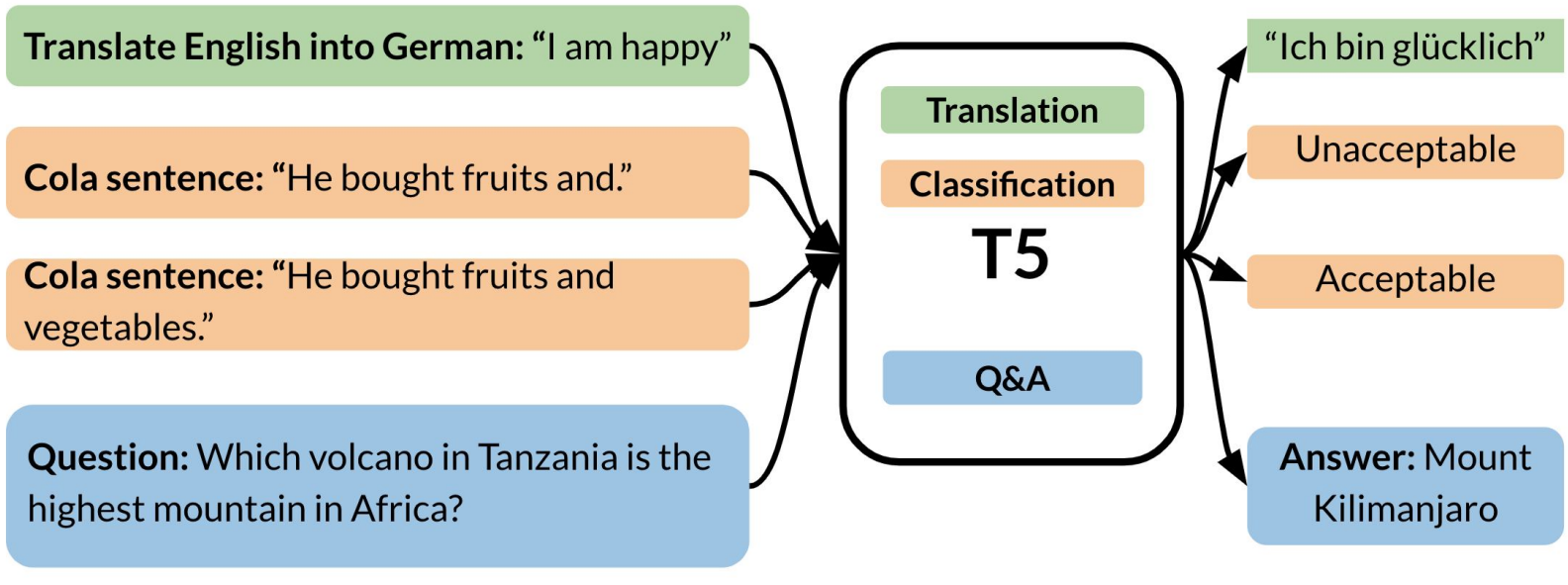


Here is a brief summary of all the different applications you can build using transformers:



It would be really cool if you can actually just go ahead and play trivia against a transformer: https://t5-trivia.glitch.me/ .

Another exciting area of research is the use of transfer learning with transformers. For example, to train a model that will translate English to German, you can just prepend the text "translate English to German" to the inputs that you are about to feed the model. You can then keep that same model to detect sentiment by prepending another tag. The following image summarizes the T5 model which uses this concept:



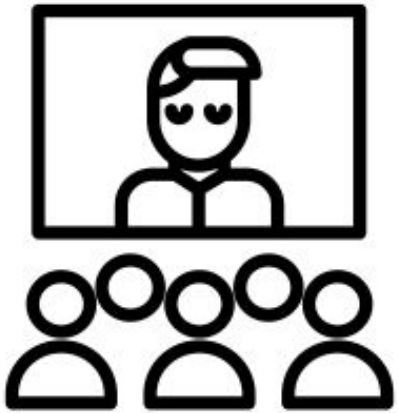
GPT-2, BERT, and T5 are some of the latest transformer models.

deeplearning.ai

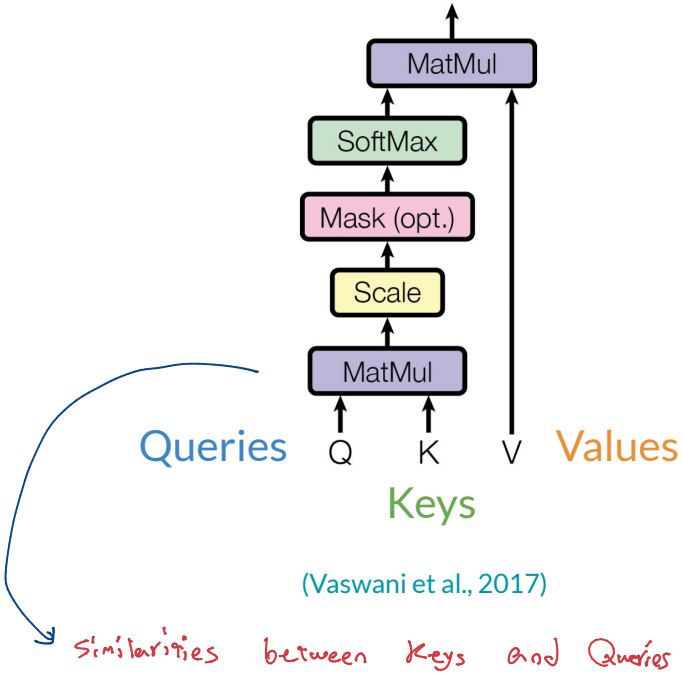
Scaled Dot-Product Attention

Outline

- Revisit scaled dot product attention
- Mathematics behind Attention



Scaled dot-product attention



(Vaswani et al., 2017)

Weights add up to 1

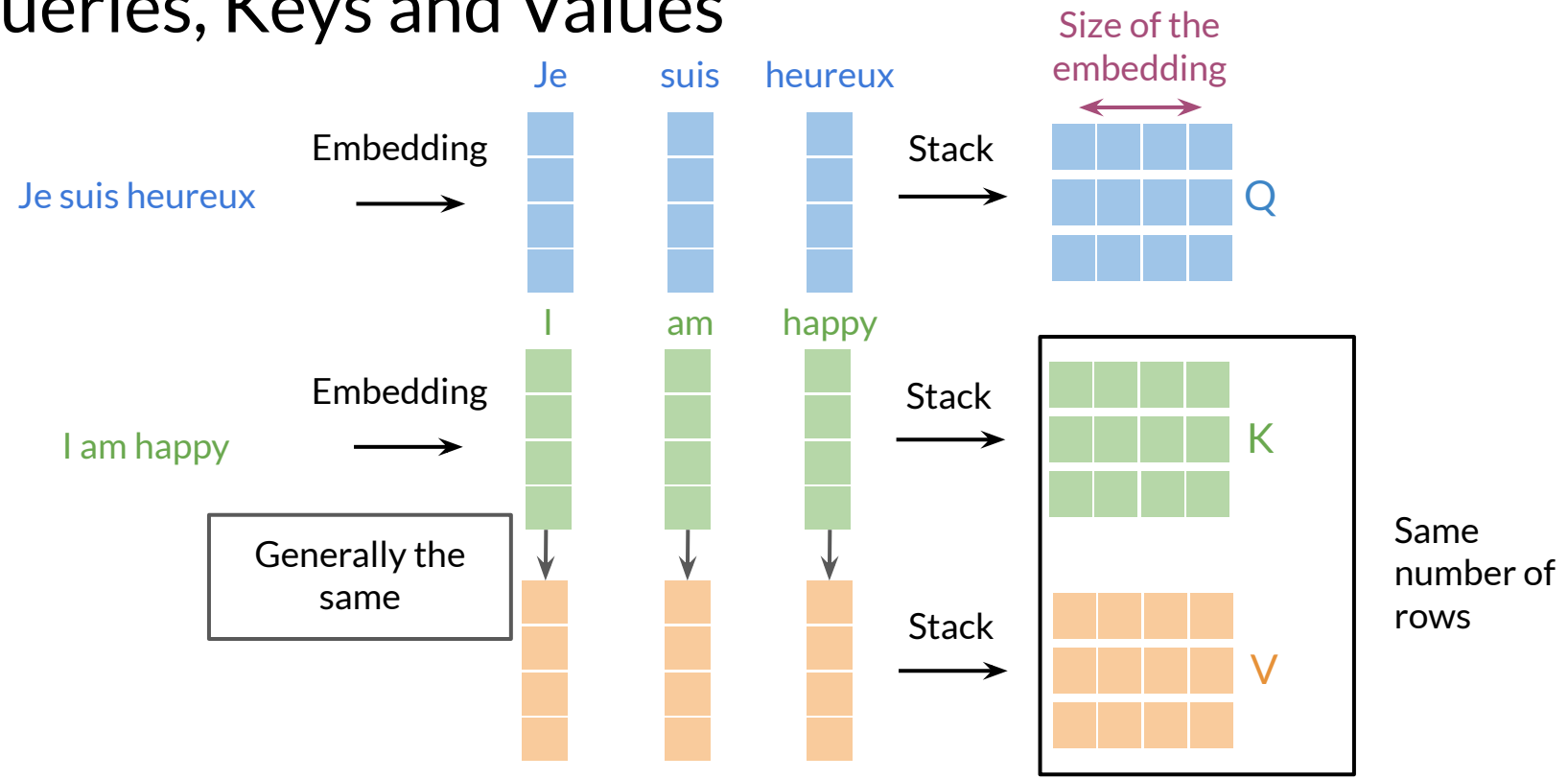
Improves performance

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

Weighted sum of values V

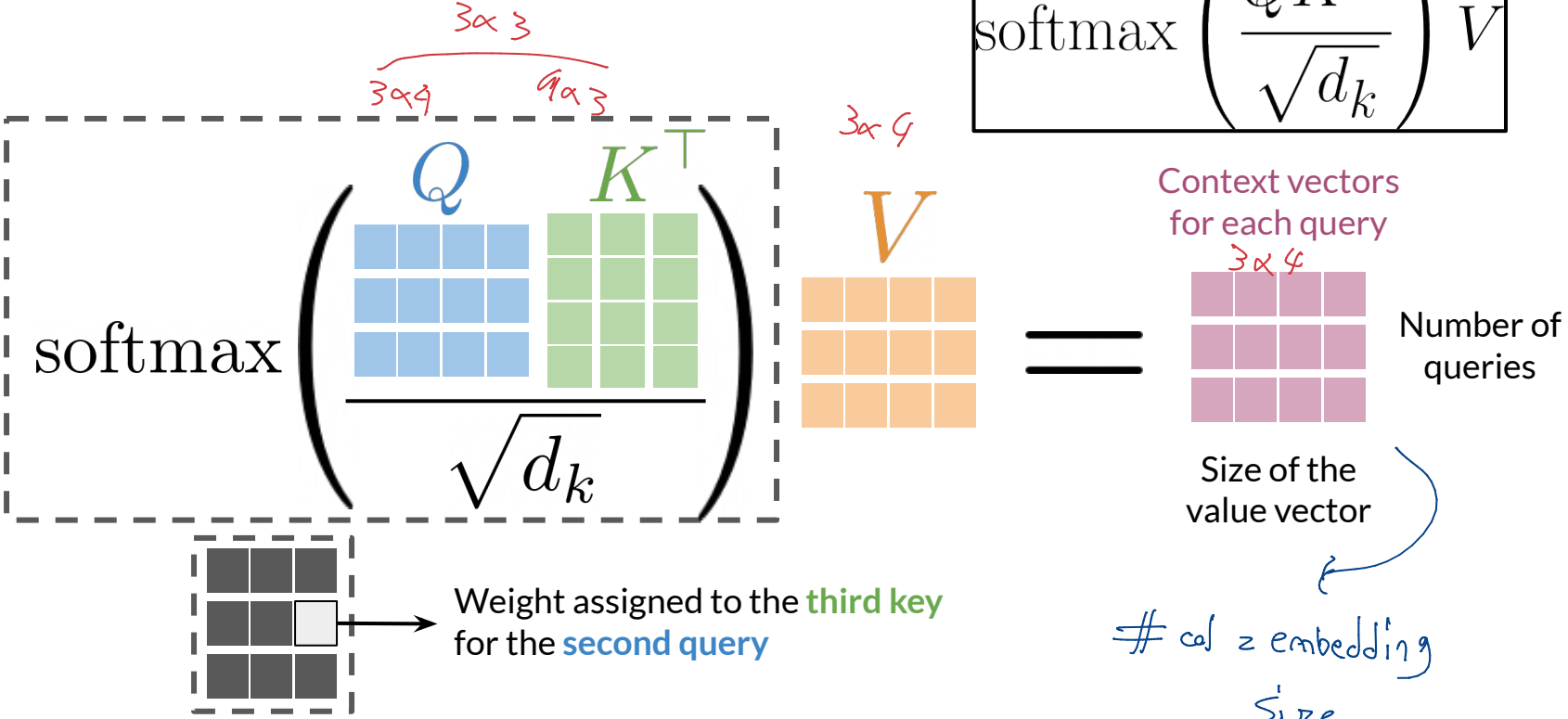
Just two matrix multiplications and a Softmax!

Queries, Keys and Values



Attention Math

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



Summary

- Scaled Dot-product Attention is essential for Transformer
- The input to Attention are queries, keys, and values
- GPUs and TPUs

deeplearning.ai

Masked Self-Attention

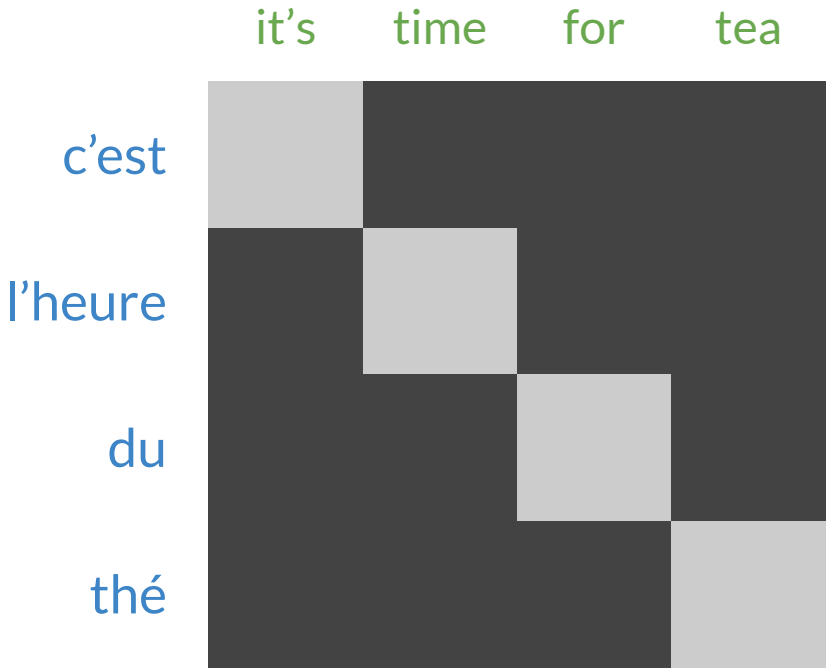
(Aka - Causal Attention)

Outline

- Ways of Attention
- Overview of masked Self-Attention

Encoder-Decoder Attention

Queries from one sentence, keys and values from another

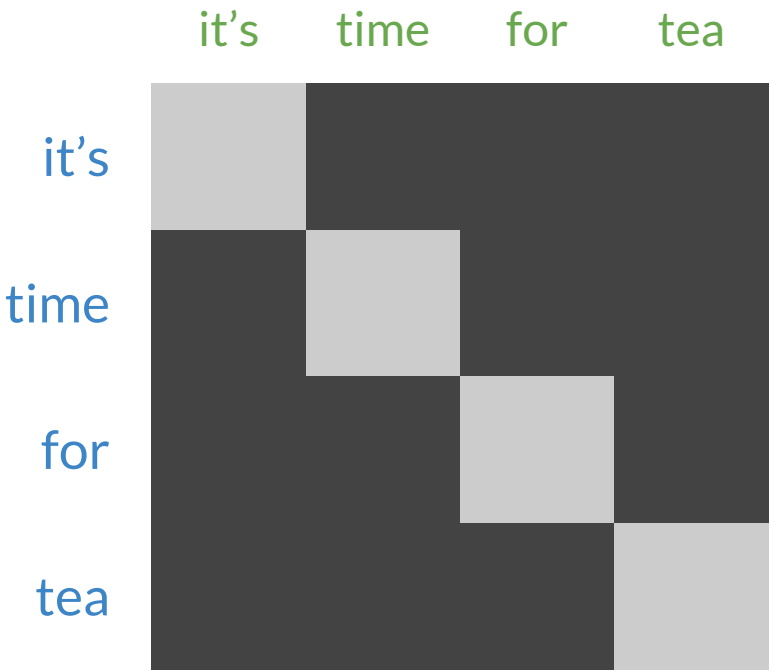


Weight matrix

Self-Attention

Queries, keys and values come from the **same sentence**

☆ Self-Attention gives you a representation of the meaning of each word within the sentence



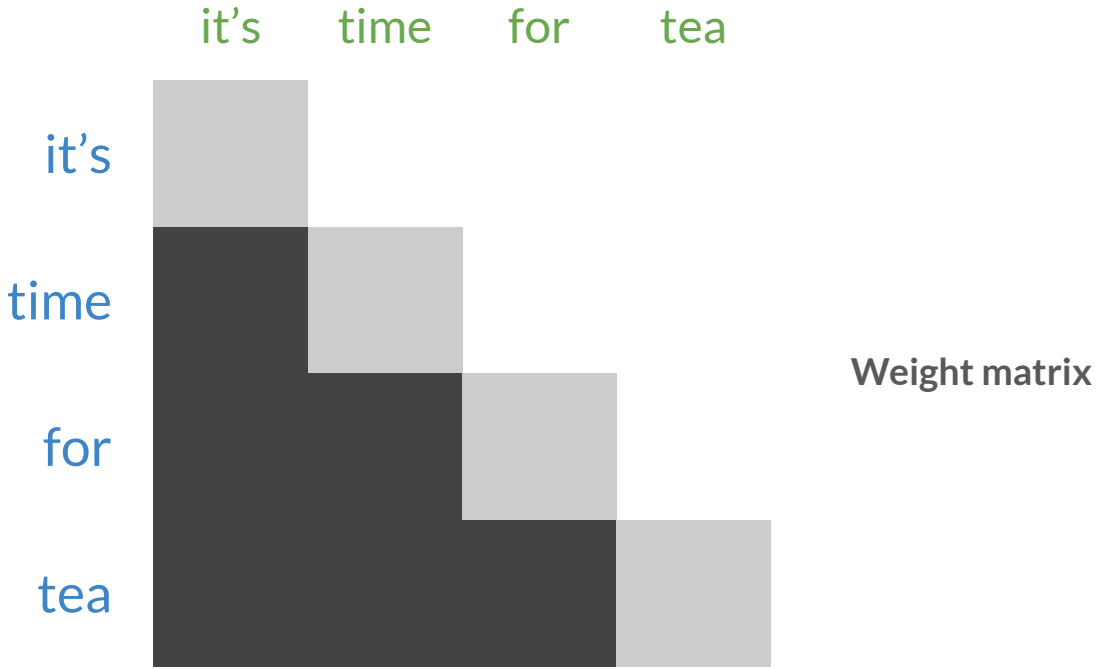
Weight matrix

Meaning of each word **within** the sentence

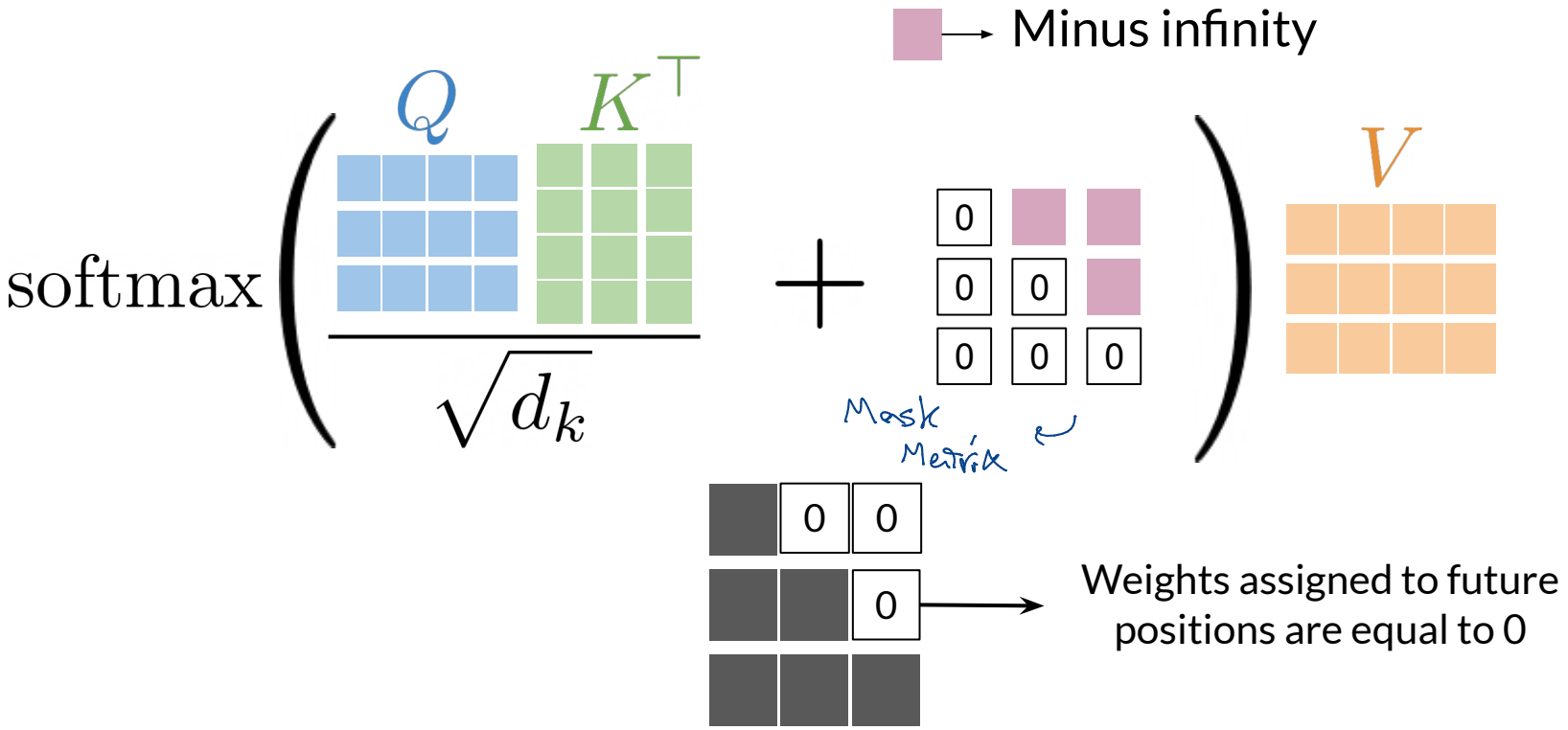
Masked Self-Attention

Queries, keys and values come from the **same sentence**. Queries don't attend to future positions.

* Used in decoder and ensure that Predictions at each Position depend only on the known outputs



Masked self-attention math



Summary

- There are three main ways of Attention: Encoder/Decoder, self-attention and masked self-attention.
- In self-attention, queries and keys come from the same sentence
- In masked self-attention queries cannot attend to the future

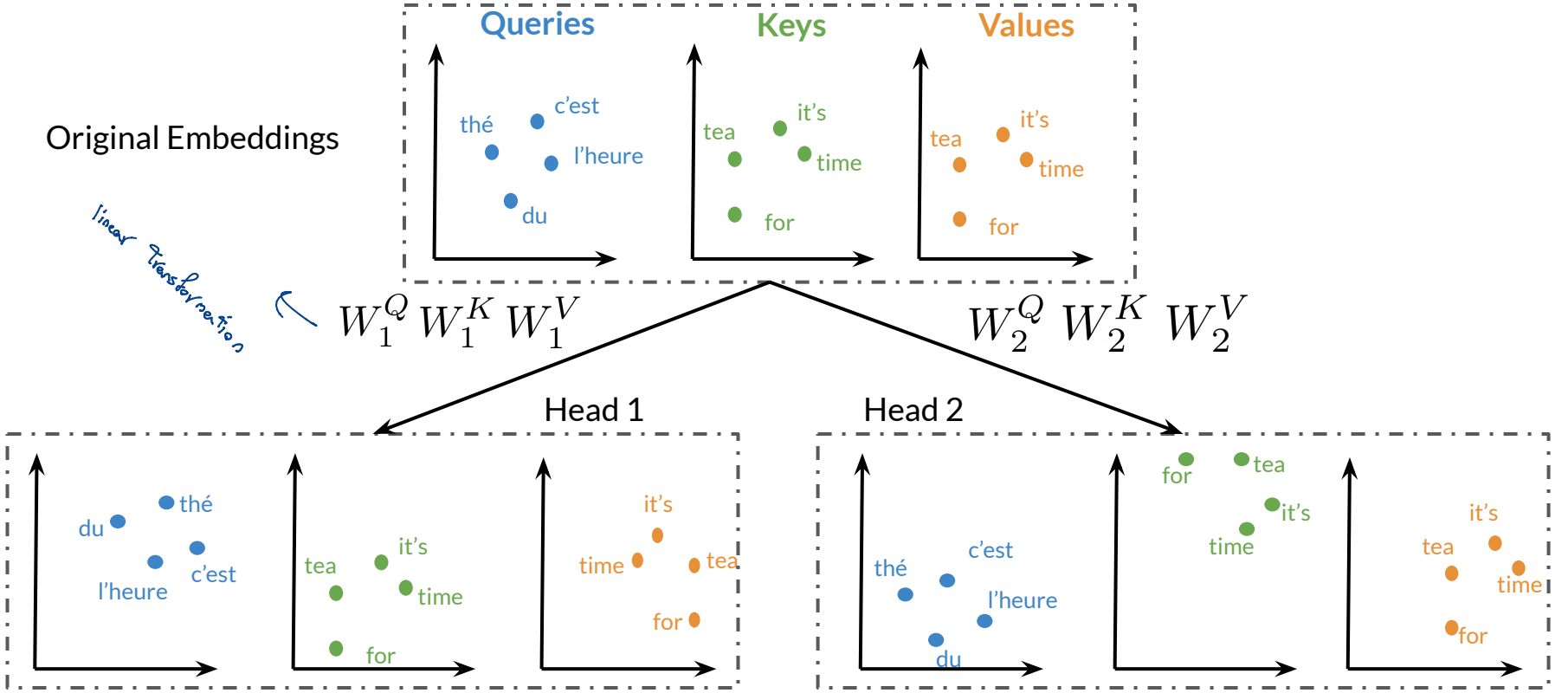
deeplearning.ai

Multi-head Attention

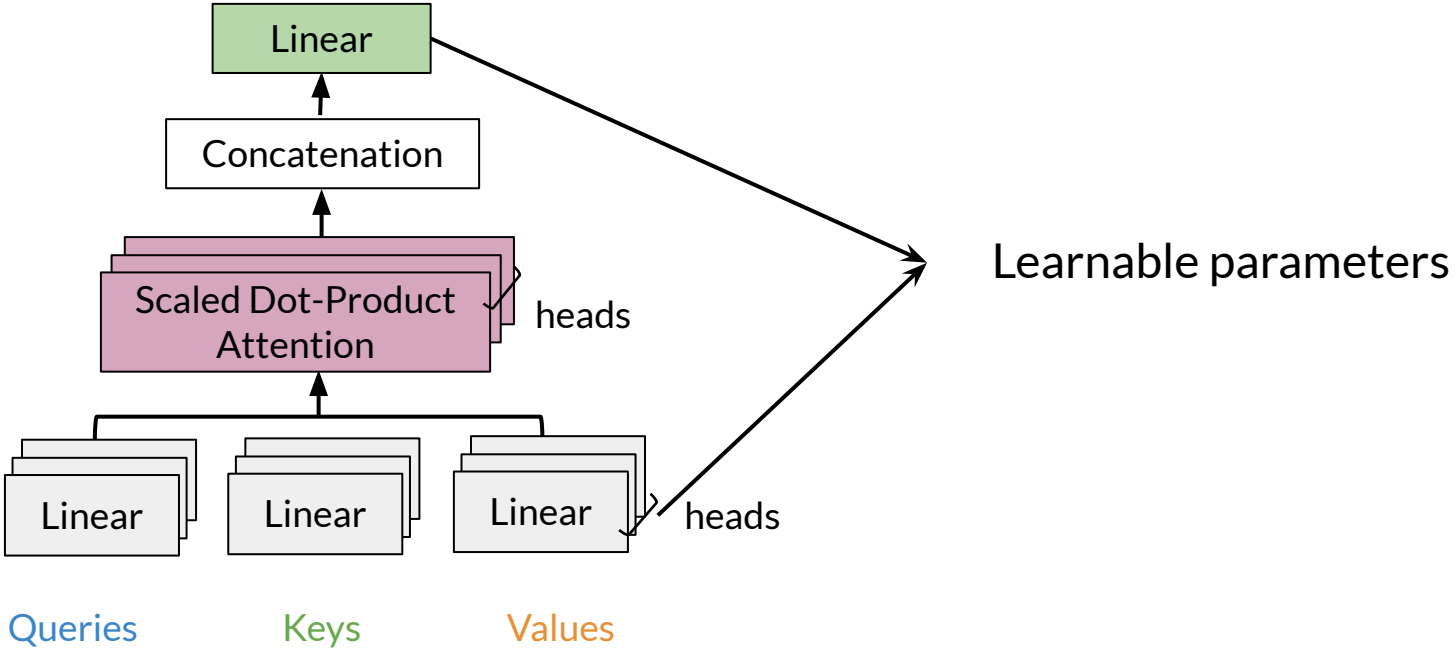
Outline

- Intuition Multi-Head Attention
- Math of Multi-Head Attention

Multi-Head Attention - Overview

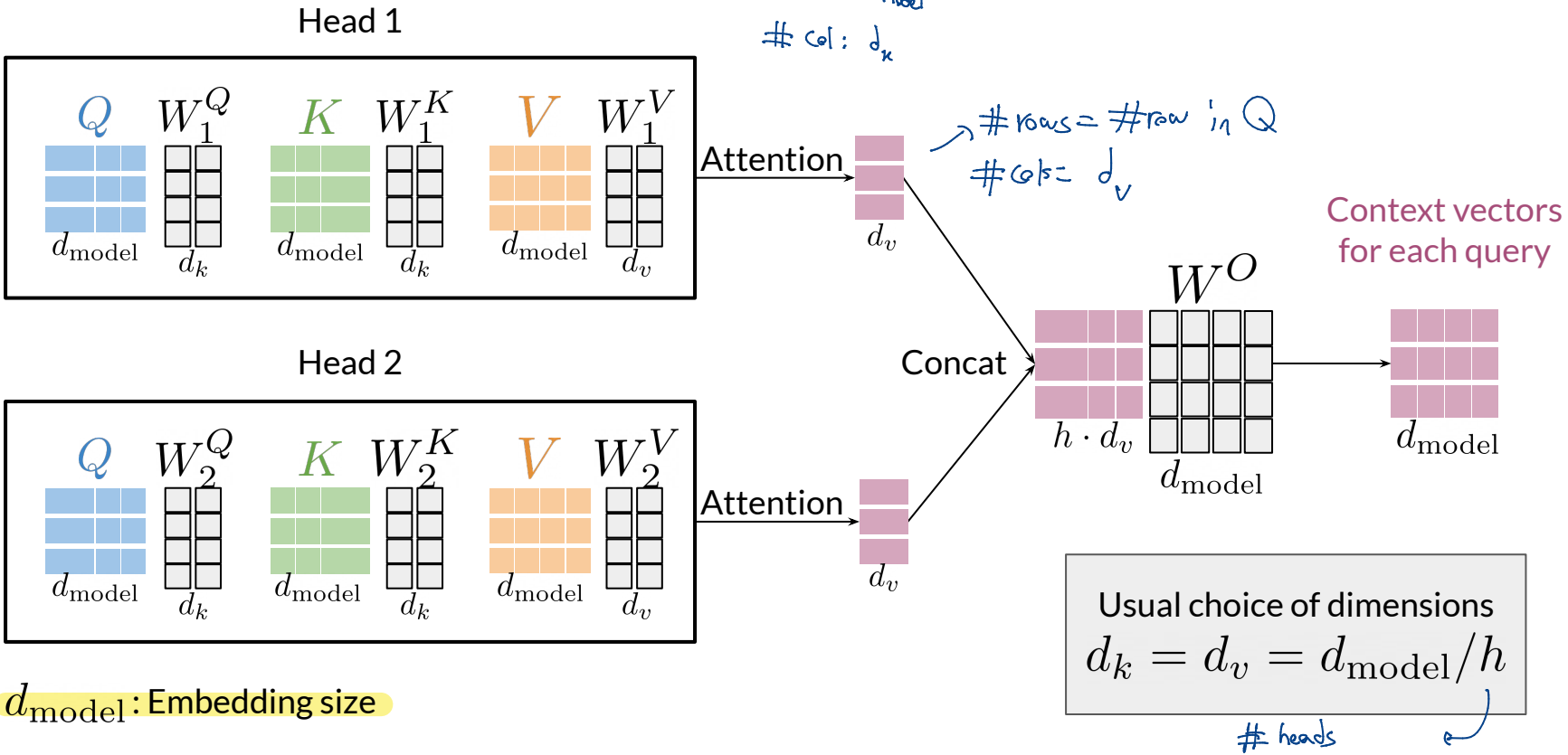


Multi-Head Attention - Overview



Multi-Head Attention

In Transformation matrices:
rows: d_{model}
col: d_k



d_{model} : Embedding size

Usual choice of dimensions

$$d_k = d_v = d_{model}/h$$

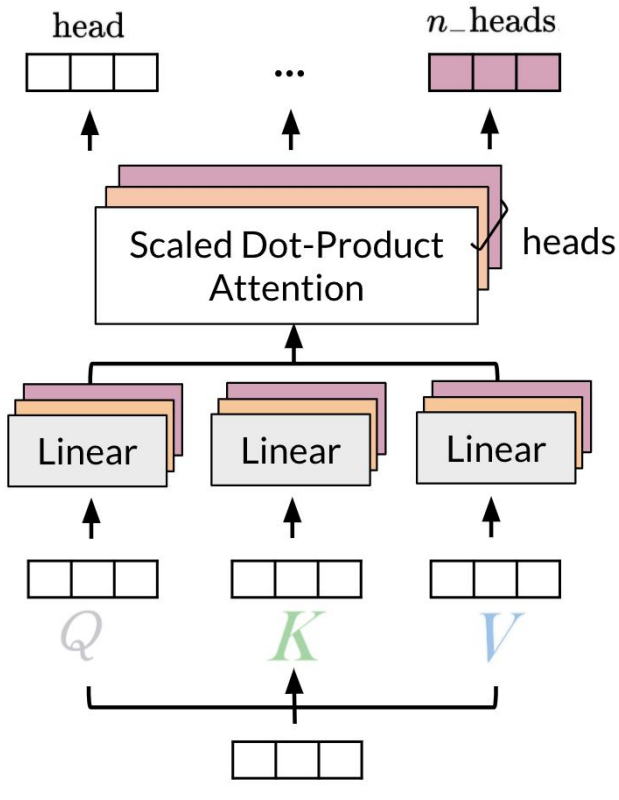
heads

Summary

- Multi-Headed models attend to information from different representations
- Parallel computations
- Similar computational cost to single-head attention

In this reading, I will summarize the intuition behind multi-head attention and scaled dot product attention.

- Each head uses different linear transformations to represent words
- Different heads can learn different relationships between words



Given a word, you take its embedding then you multiply it by the Q, K, V matrix to get the corresponding queries, keys and values. When you use multi-head attention, a head can learn different relationships between words from another head.

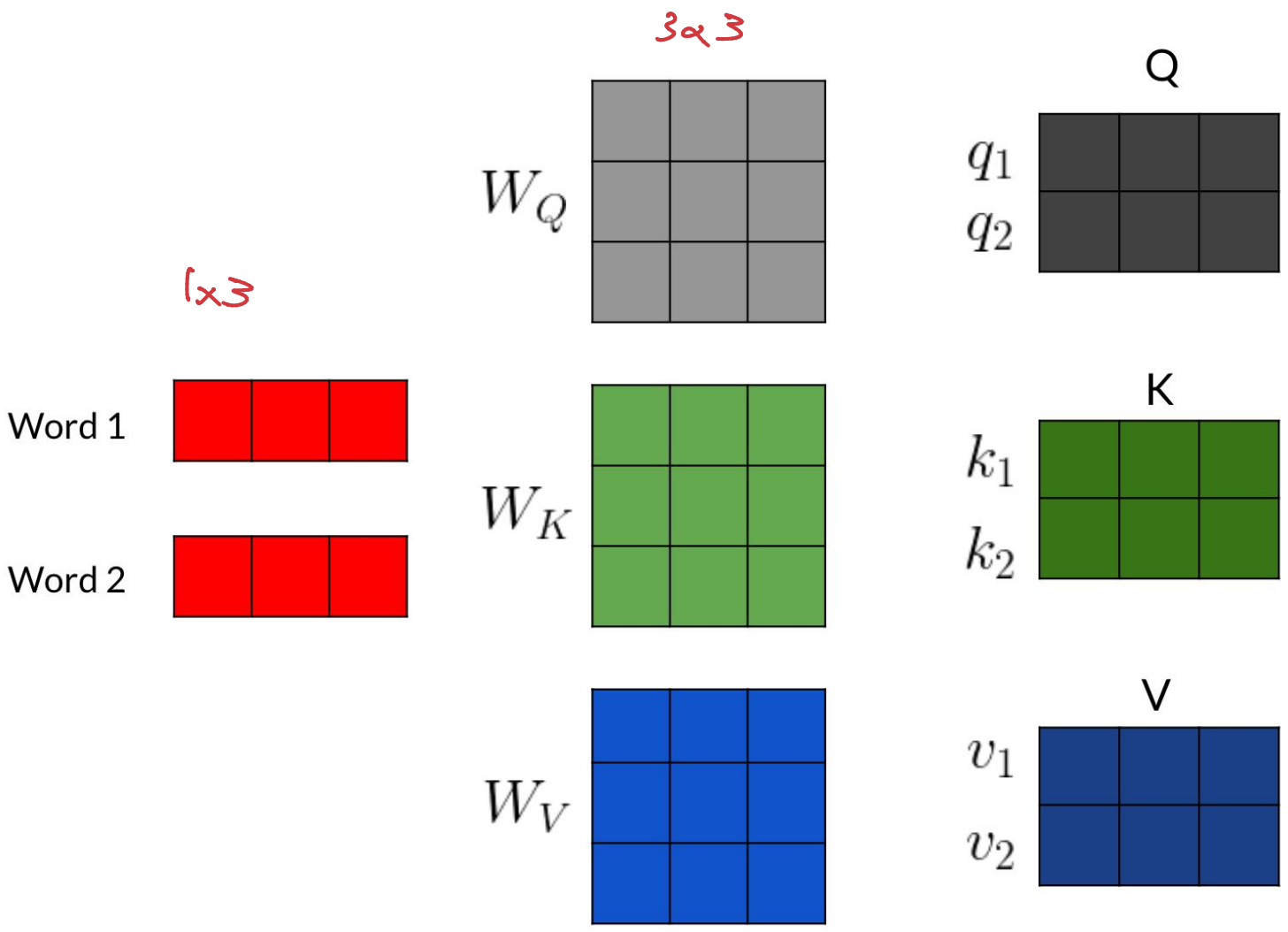
Here's one way to look at it:

- First, imagine that you have an embedding for a word. You multiply that embedding with Q to get q_1, K to get k_1, and V to get v_1.

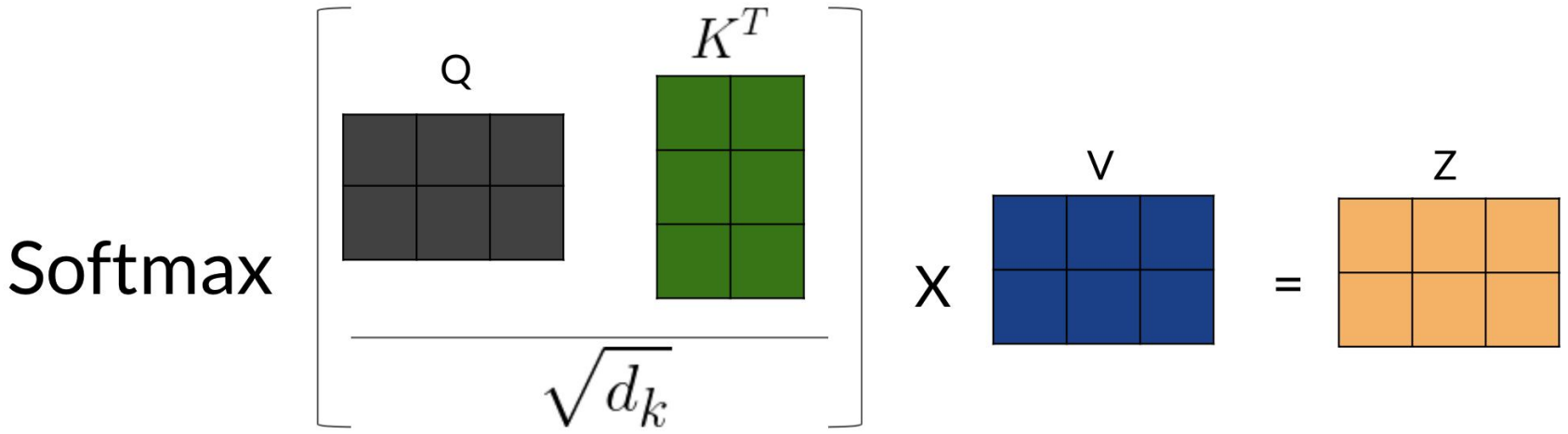
- Next, you feed it to the linear layer, once you go through the linear layer for each word, you need to calculate a score. After that, you end up having an embedding for each word. But you still need to get the score for how much of each word you are going to use. For example, this will tell you how similar two words are q_1 and k_1 or even q_1 and k_2 by doing a simple $q_1 \dot{k}_1$. You can take the softmax of those scores (the paper mentions that you have to divide by $\sqrt{(d)}$ to get a probability and then you multiply that by the value. That gives you the new representation of the word.

If you have many heads, you can concatenate them and then multiply again by a matrix that is of dimension (dim of each head by num heads - dim of each head) to get one final vector corresponding to each word.

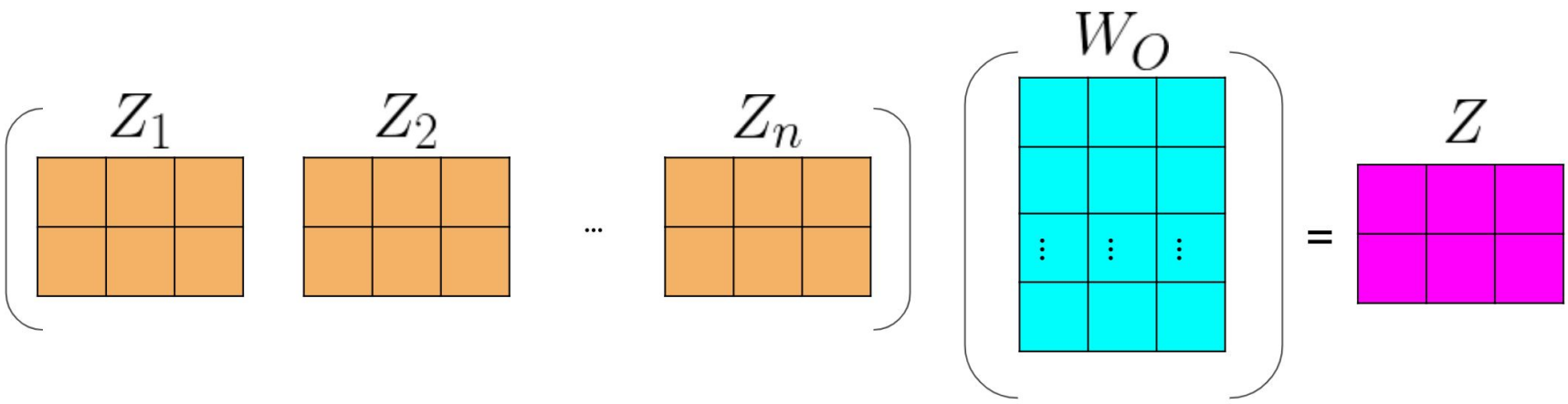
Here is step by step guide, first you get the Q, K, V matrices:



For each word, you multiply it by the corresponding W_Q, W_K, W_V matrices to get the corresponding word embedding. Then you have to calculate scores with those embedding as follows:



Note that the computation above was done for one head. If you have several heads, concretely n, then you will have $Z_1, Z_2, \ldots, Z_n$. In which case, you can just concatenate them and multiply by a W_O matrix as follows:



Hence, the more heads you have, the more Zs you will end up concatenating and as a result, that will change the inner dimension of W_O, which will then project the combined embeddings into one final embedding.

deeplearning.ai

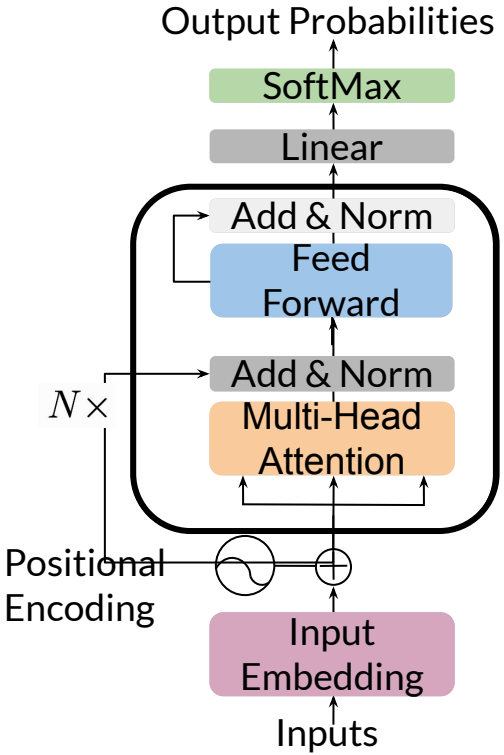
Transformer decoder

GPT-2

Outline

- Overview of Transformer decoder
- Implementation (decoder and feed-forward block)

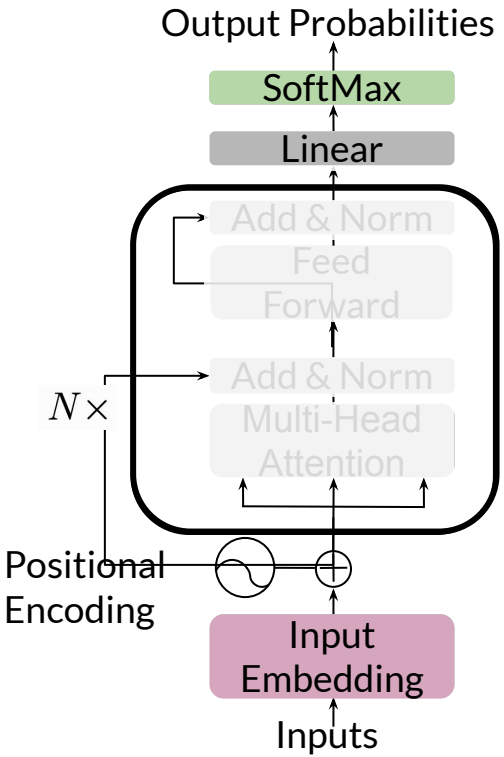
Transformer decoder



Overview

- input: sentence or paragraph
  - we predict the next word
- sentence gets embedded, add positional encoding
  - (vectors representing $\{0, 1, 2, \ldots, K\}$)
- multi-head attention looks at previous words
- feed-forward layer with ReLU
  - that's where most parameters are!
- residual connection with layer normalization
- repeat N times
- dense layer and softmax for output

Transformer decoder



d_{model} = Size of embeddings (512 - 1024 - -)

(Batch x length x Vocab size)

Explanation

N decoder blocks

Decoder Block

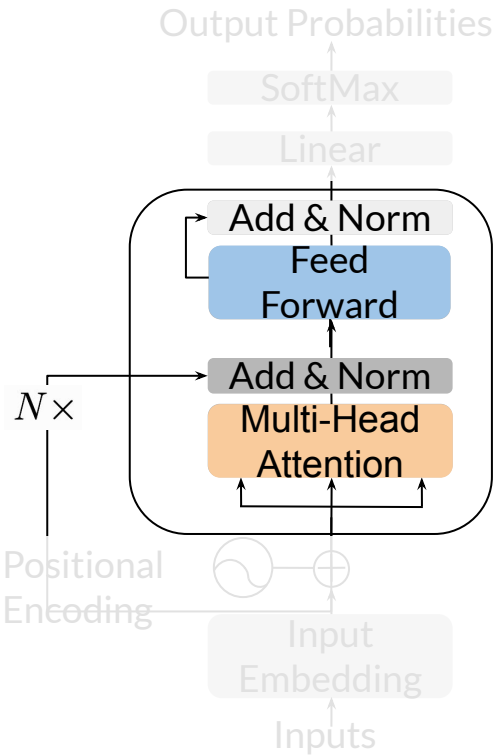
Positional Encoding

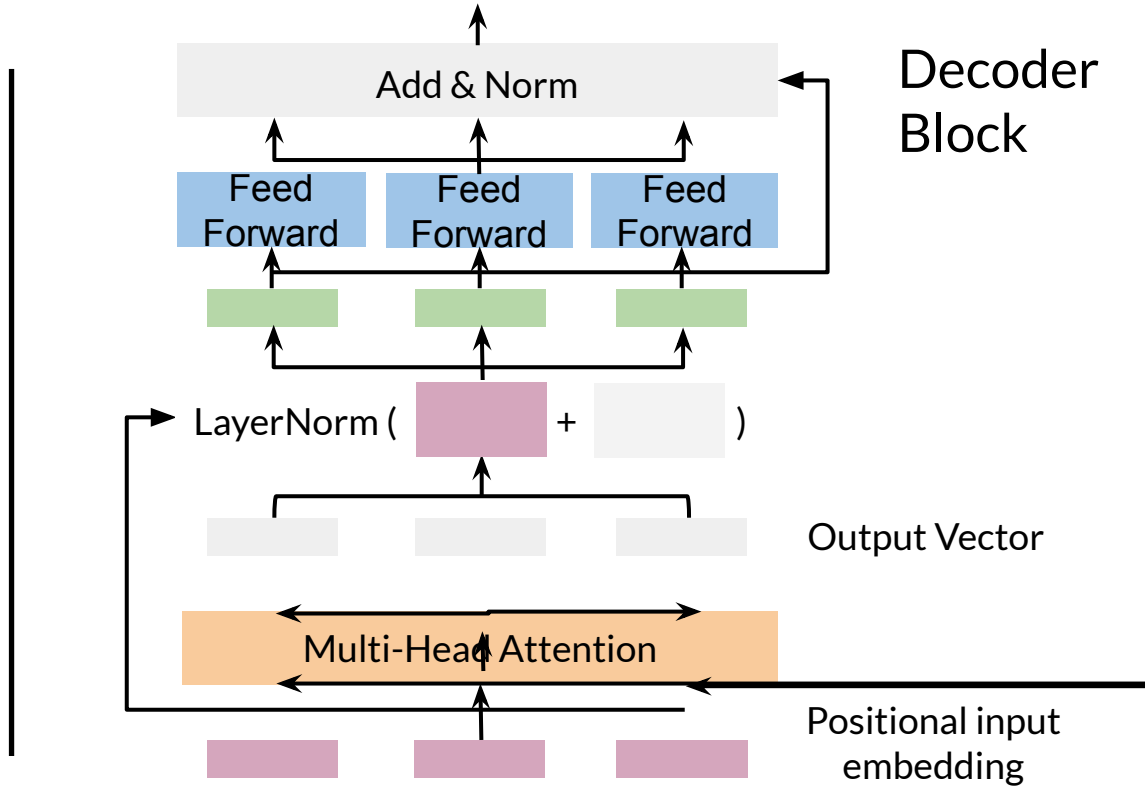
(Batch x length x d_{model})

Input Embedding

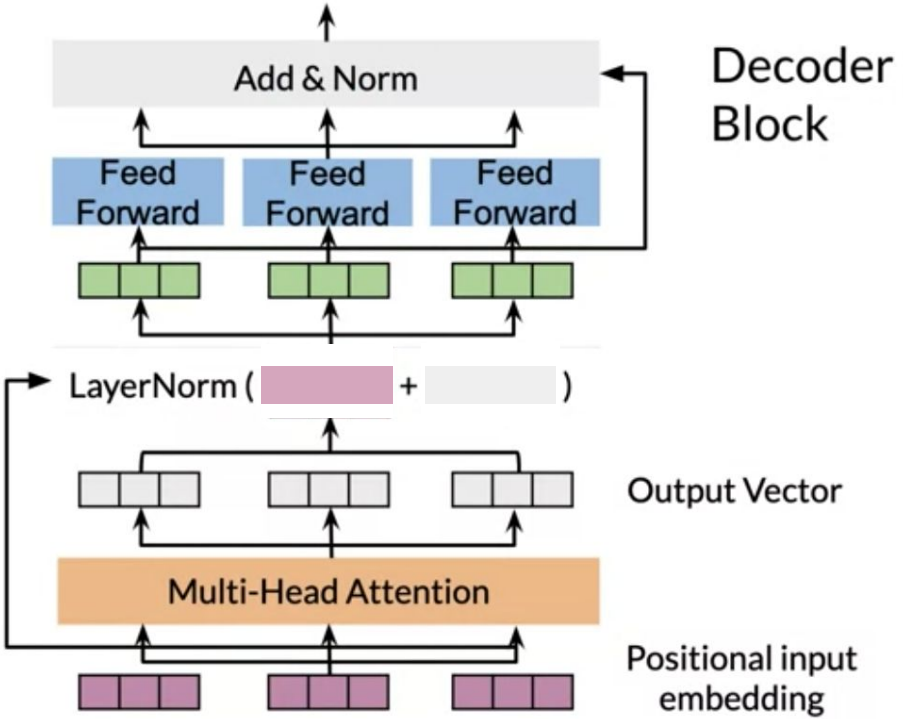
(Batch x length) <start> I am happy

The Transformer decoder





Decoder Block

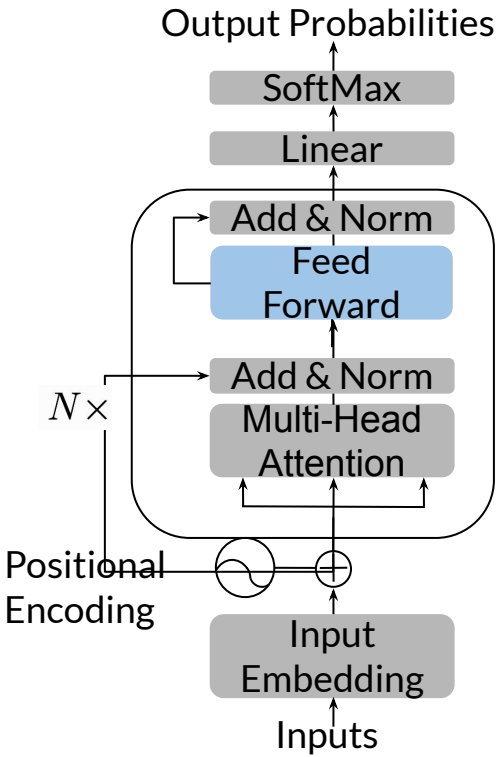


Decoder Block

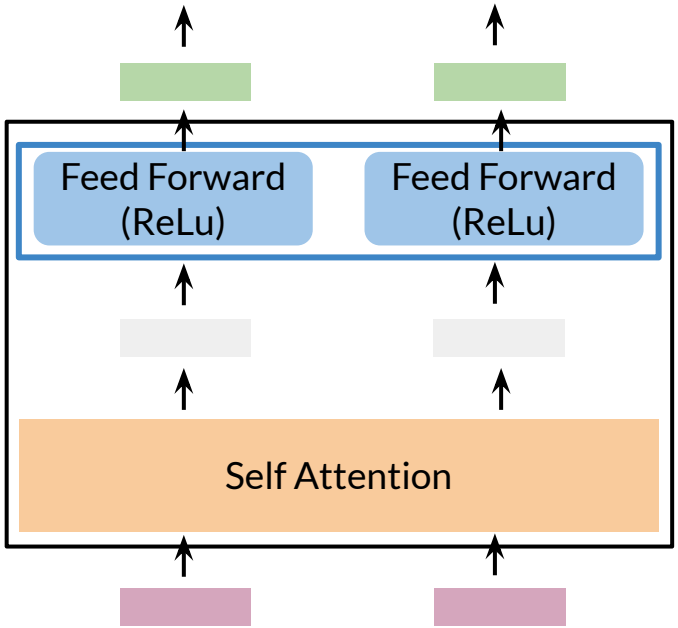
Output Vector

Positional input embedding

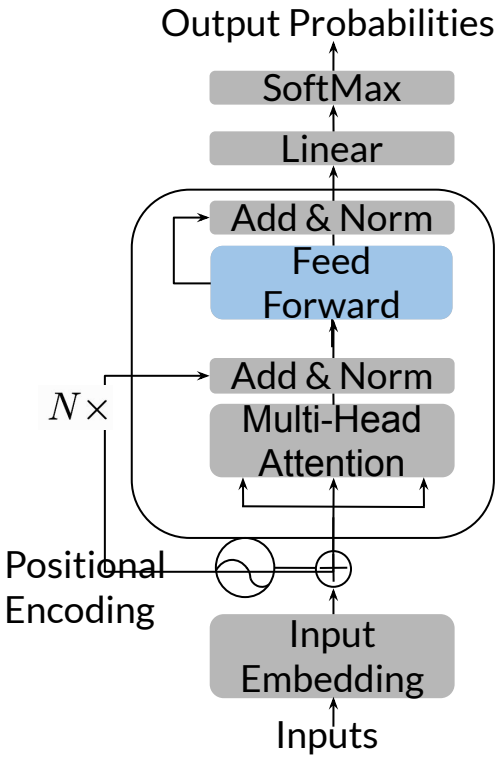
The Transformer decoder



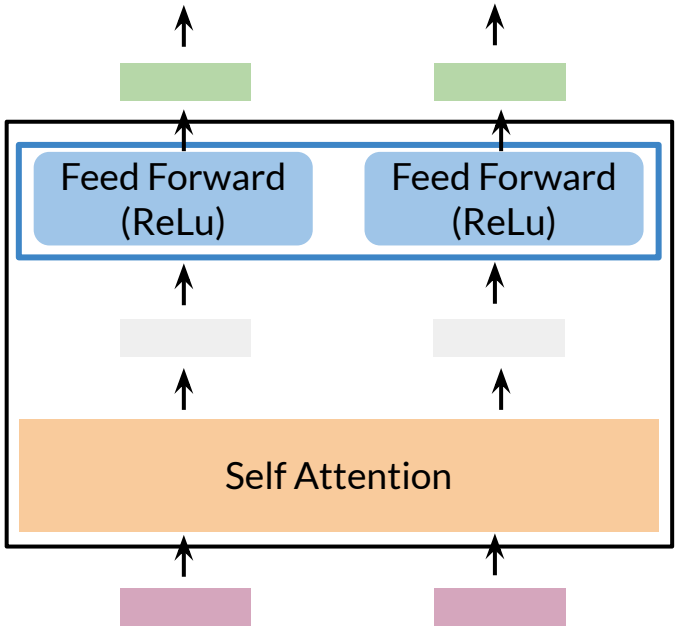
Feed forward layer



The Transformer decoder



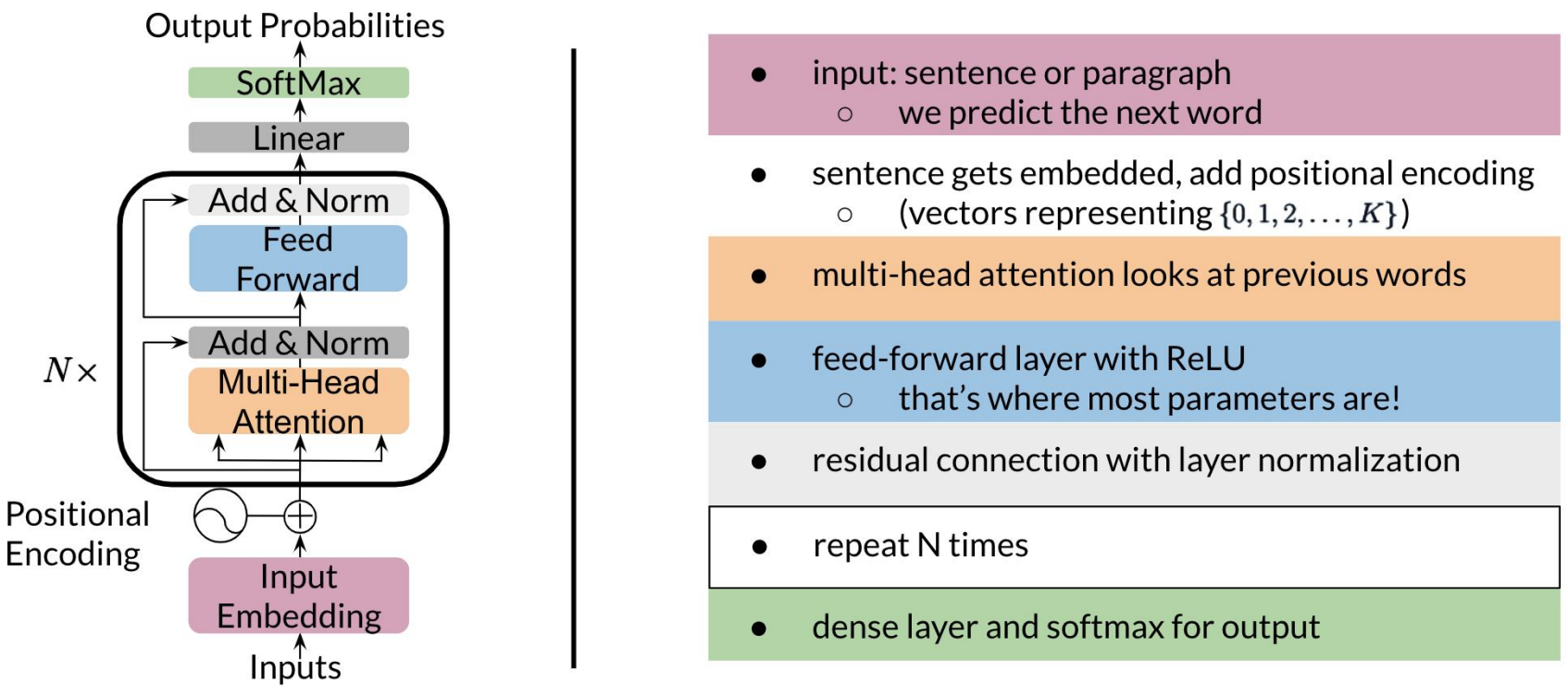
Feed forward layer



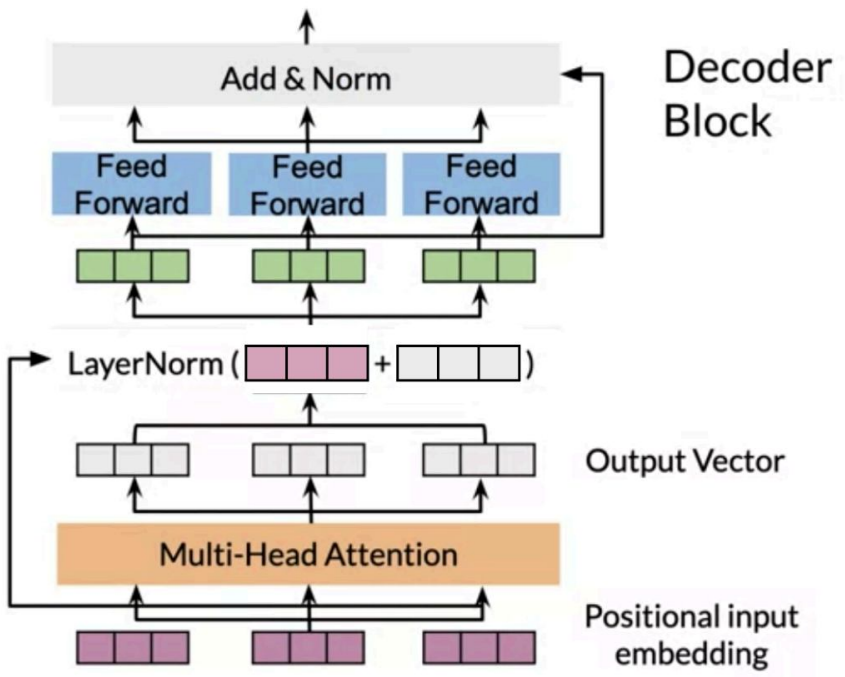
Summary

- Transformer decoder mainly consists of three layers
- Decoder and feed-forward blocks are the core of this model code
- It also includes a module to calculate the cross-entropy loss

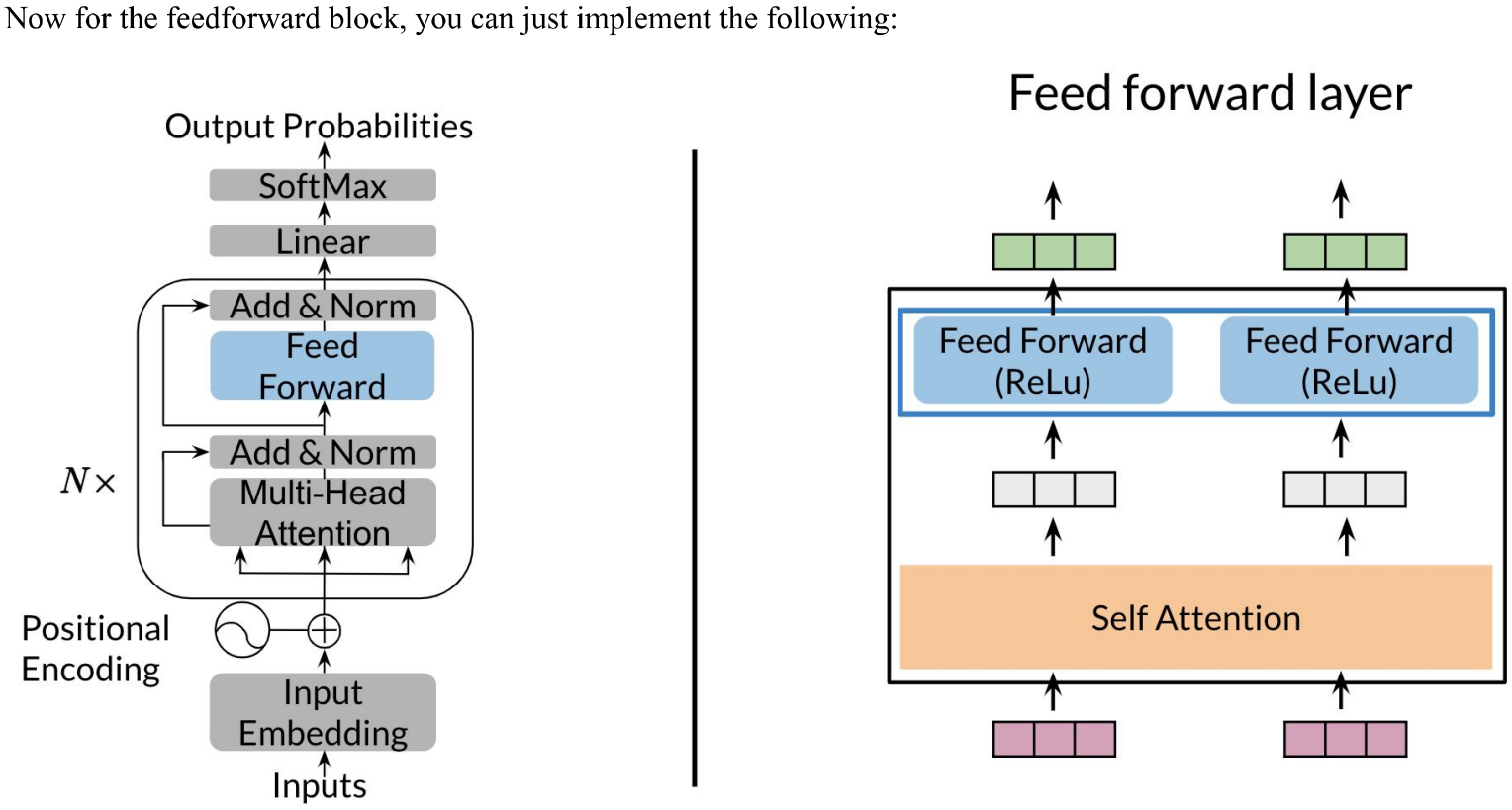
That was a lot of information! Don't worry if you did not understand everything, we will go over everything step by step.



Once you get the embeddings, you append to it the positional encoding, which you can think of just a learned number that tells you information about the position of the word. Then, you do mutli-head attention as explained in the previous video/reading. There is a feedforward layer with a ReLU after, a residual connection with layer normalization (repeat the block shown above N times), finally a linear layer with a softmax. Zoning into the block that gets repeated N times, you get the following:



Now for the feedforward block, you can just implement the following:



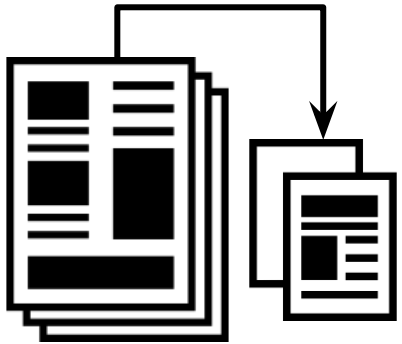
You get the input, (red vector) run it through self-attention and then a feedforward with ReLU. At the end of the decoder, you can just run a softmax.

deeplearning.ai

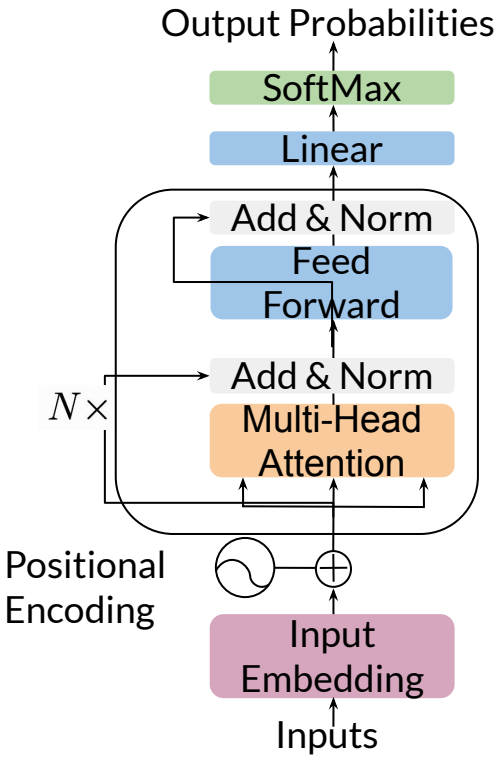
Transformer summarizer

Outline

- Overview of Transformer summarizer
- Technical details for data processing
- Inference with a Language Model

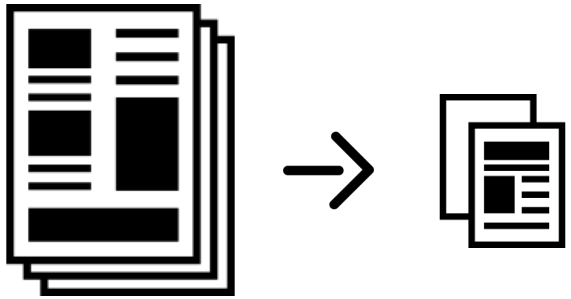


Transformer for summarization

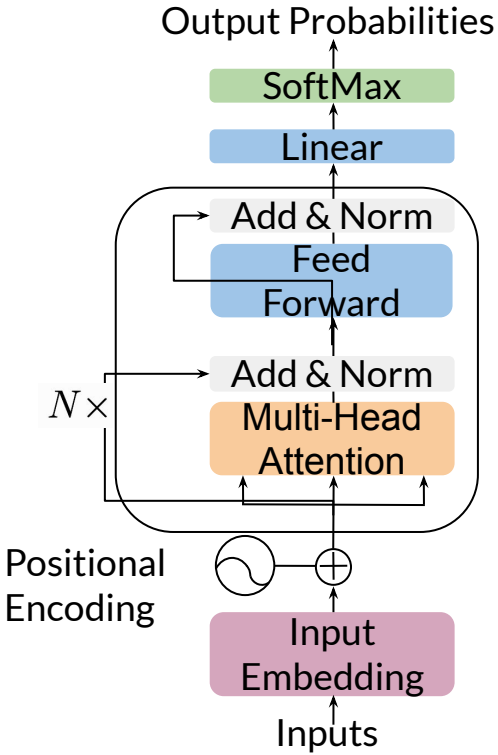


Input

Output: Summary



Technical details for data processing



Model Input:

```
ARTICLE TEXT <EOS> SUMMARY <EOS> <pad> ...
```

Tokenized version:

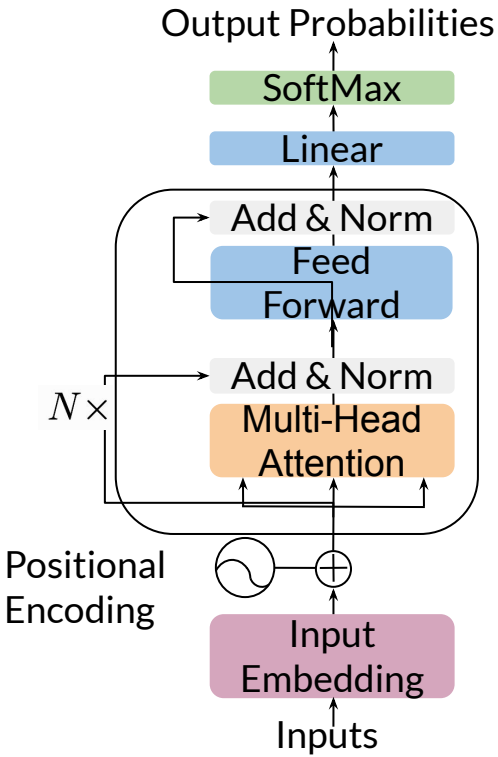
1 = Eos

0 = Padding

```
[2,3,5,2,1,3,4,7,8,2,5,1,2,3,6,2,1,0,0]
```

Loss weights: 0s until the first `<EOS>` and then 1 on the start of the summary.

Cost function



Cross entropy loss

$$J = -\frac{1}{m}\sum_{j}^{m}\sum_{i}^{K} y_j^i \log \hat{y}_j^i$$

j : over summary

i : bach elements

Inference with a Language Model

Model input:

`[Article] <EOS>  [Summary] <EOS>`

Inference:

- Provide: `[Article] <EOS>`

- Generate summary word-by-word
  - until the final `<EOS>`

- Pick the next word by random sampling
  - each time you get a different summary!

Summary

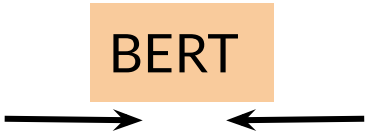
- For summarization, a weighted loss function is optimized
- Transformer Decoder summarizes predicting the next word using
- The transformer uses tokenized versions of the input

deeplearning.ai

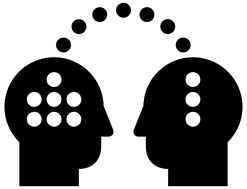
Week 3 Overview

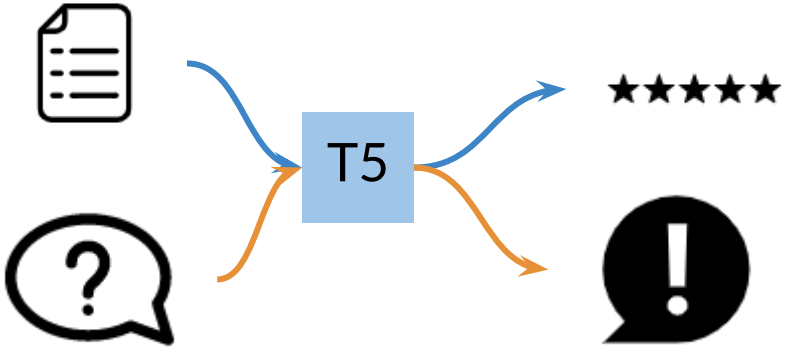
Week 3

Question Answering



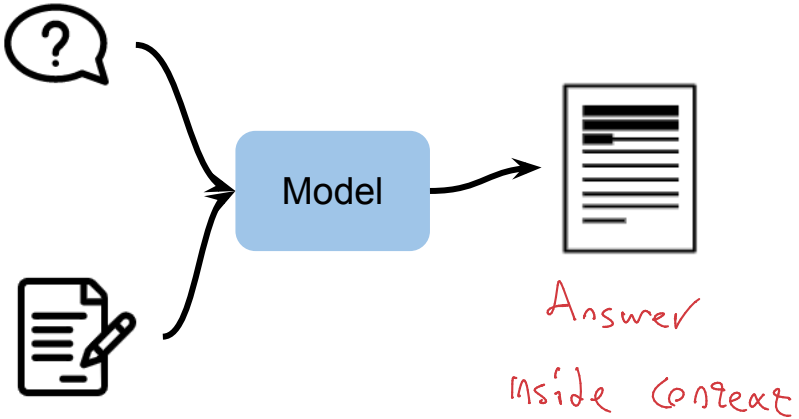
Transfer learning



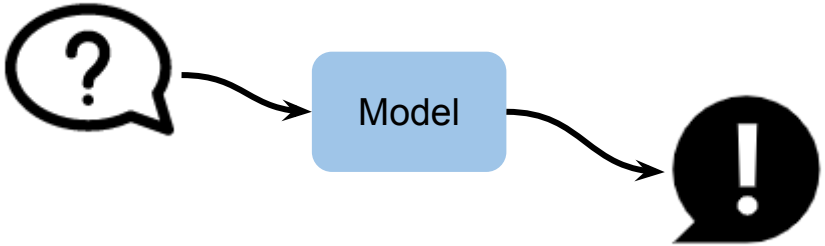


Question Answering

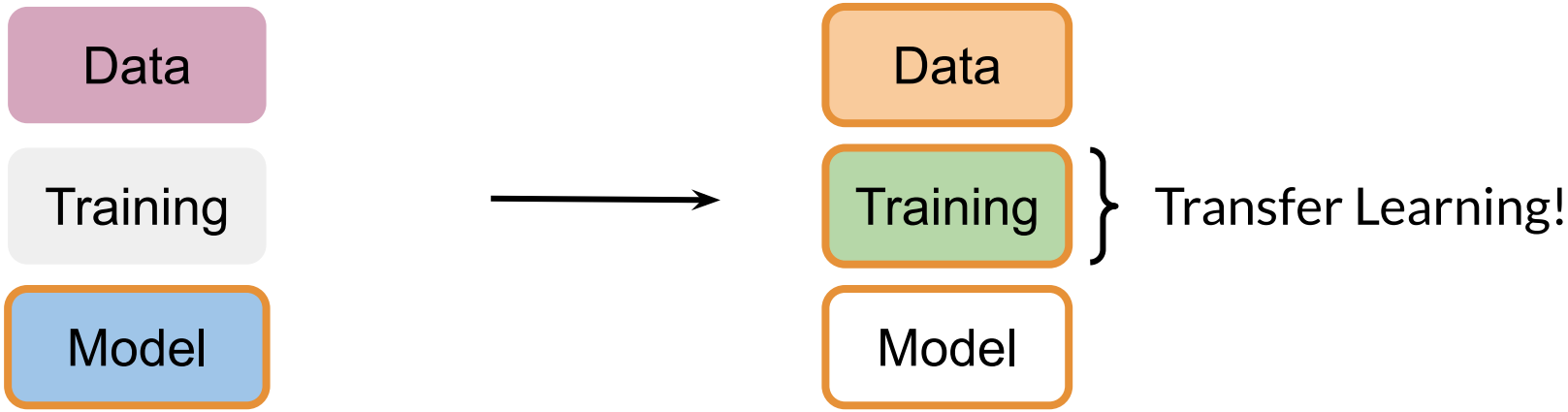
Context-based



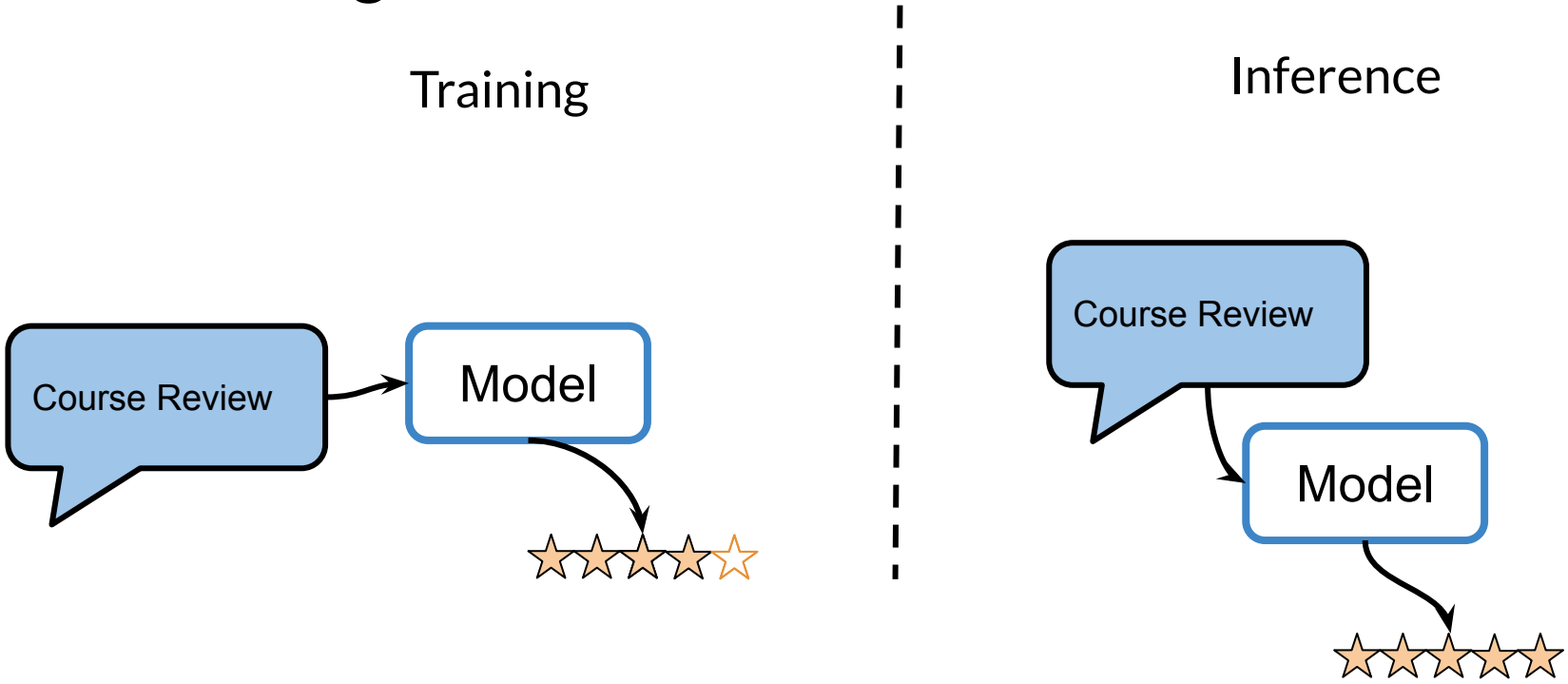
Closed book



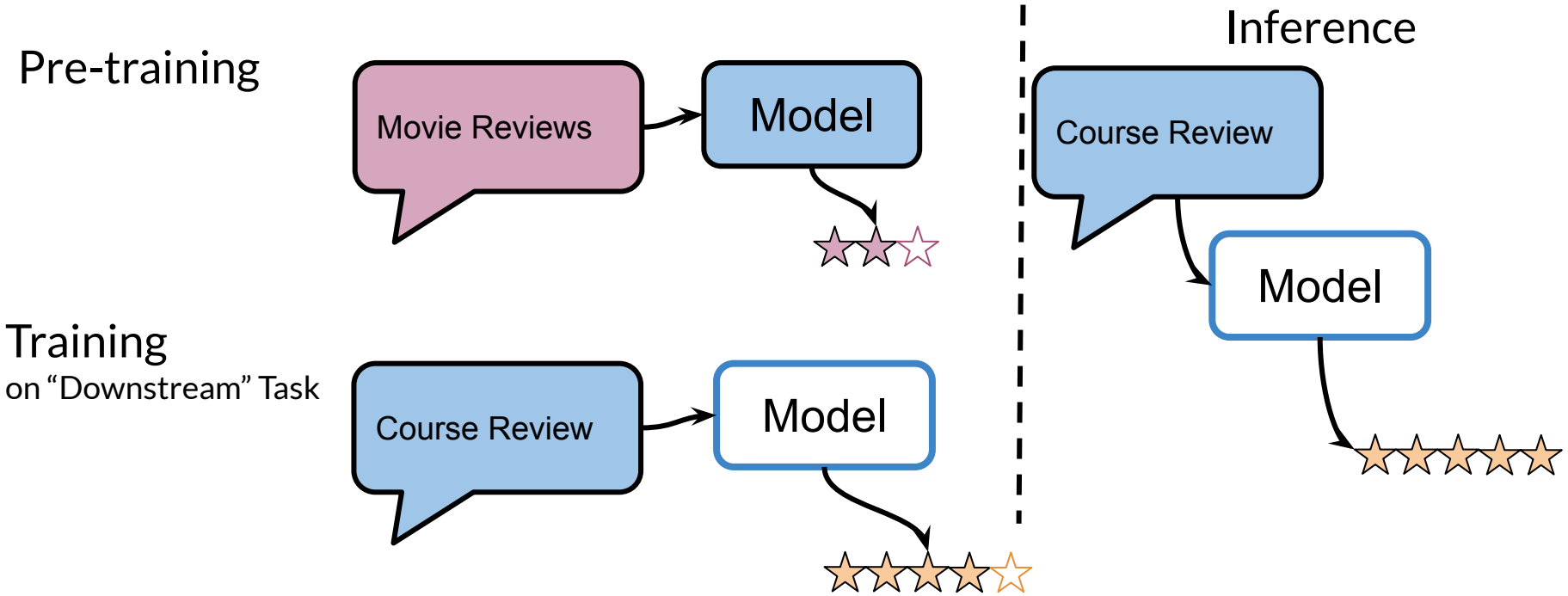
Not just the model



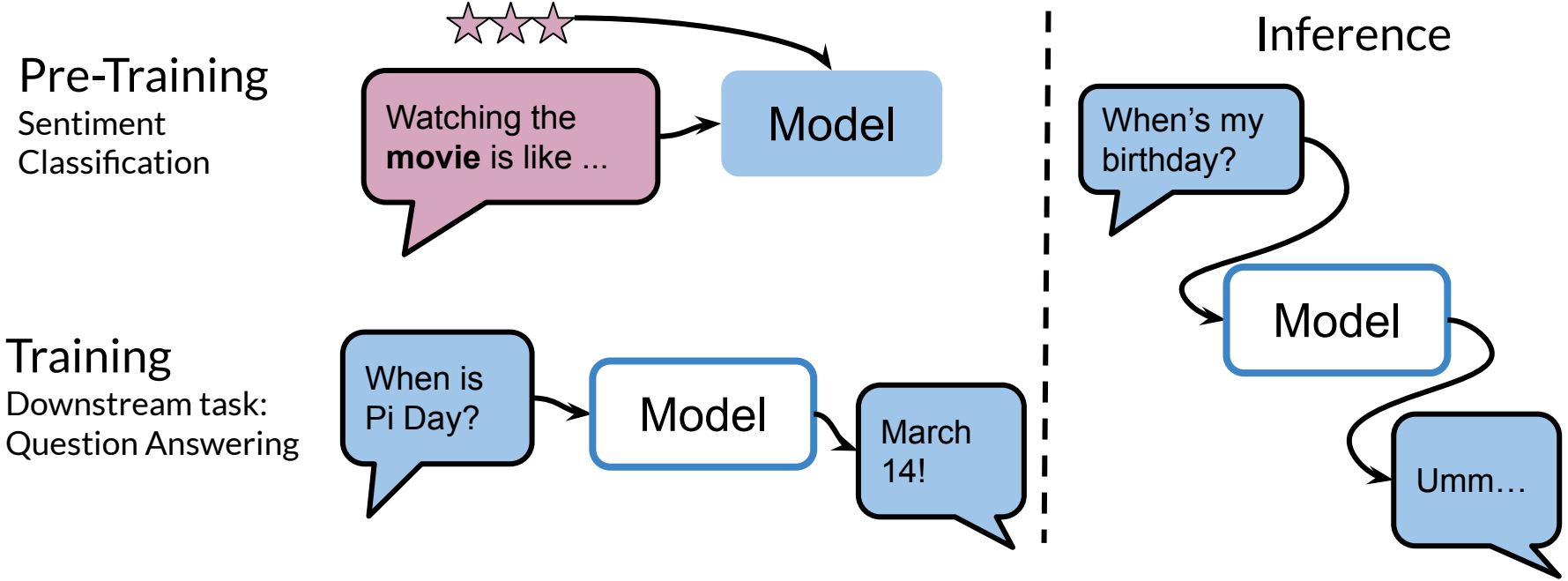
Classical training



Transfer learning

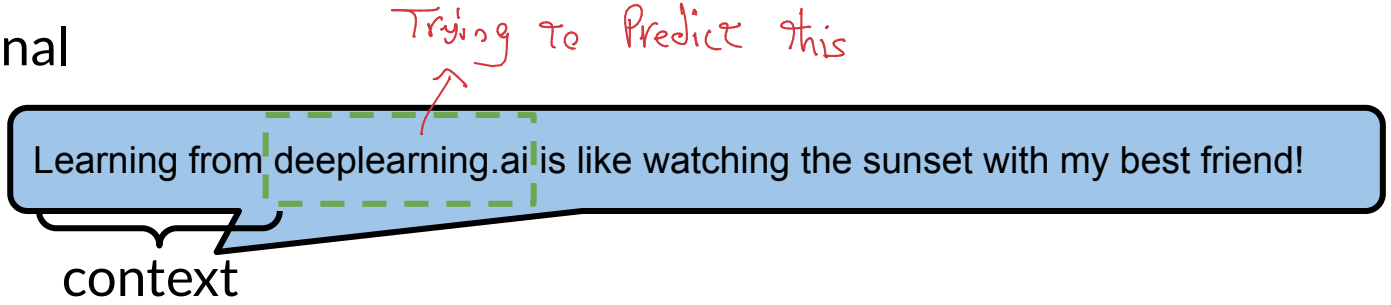


Transfer Learning: Different Tasks

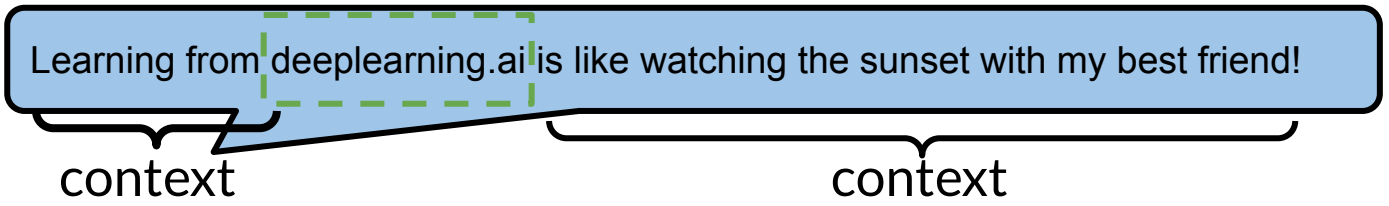


BERT: Bi-directional Context

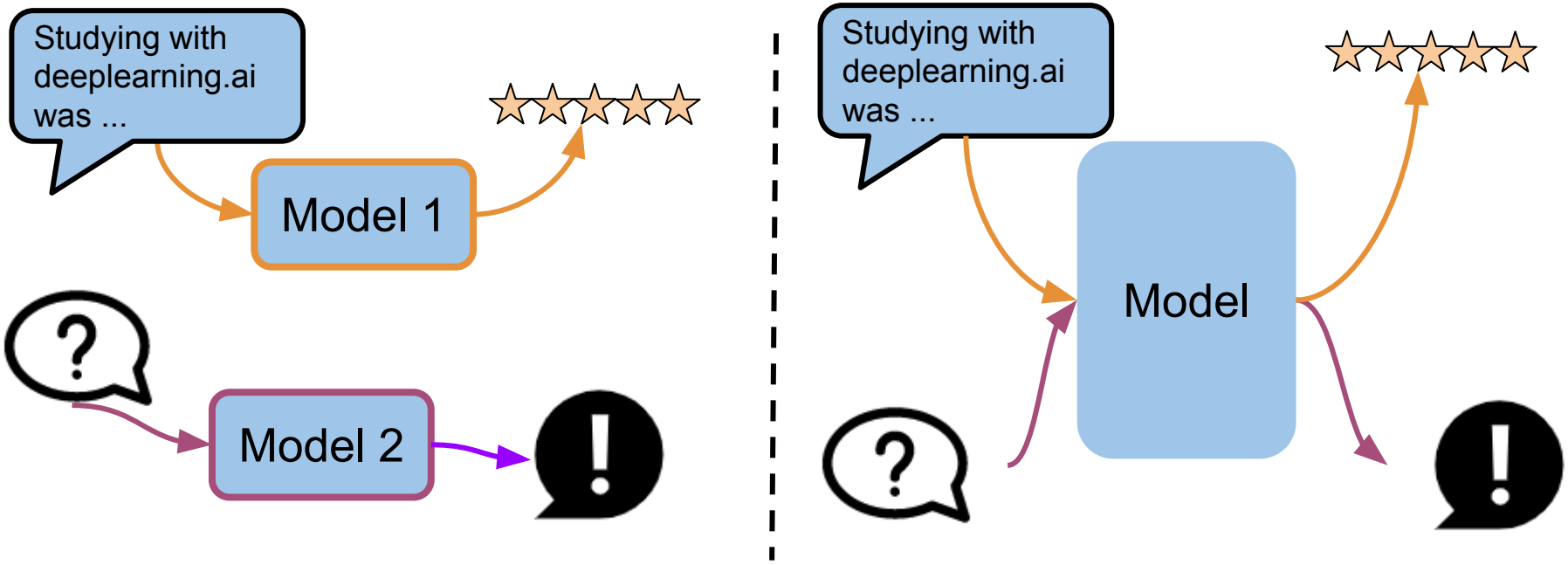
Uni-directional



Bi-directional

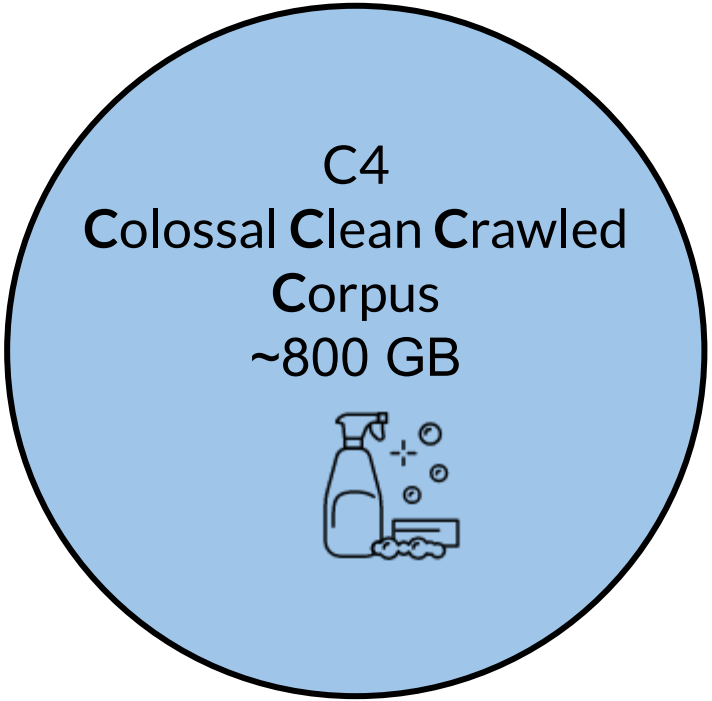


T5: Single task vs. Multi task



T5: more data, better performance

English wikipedia
~13 GB

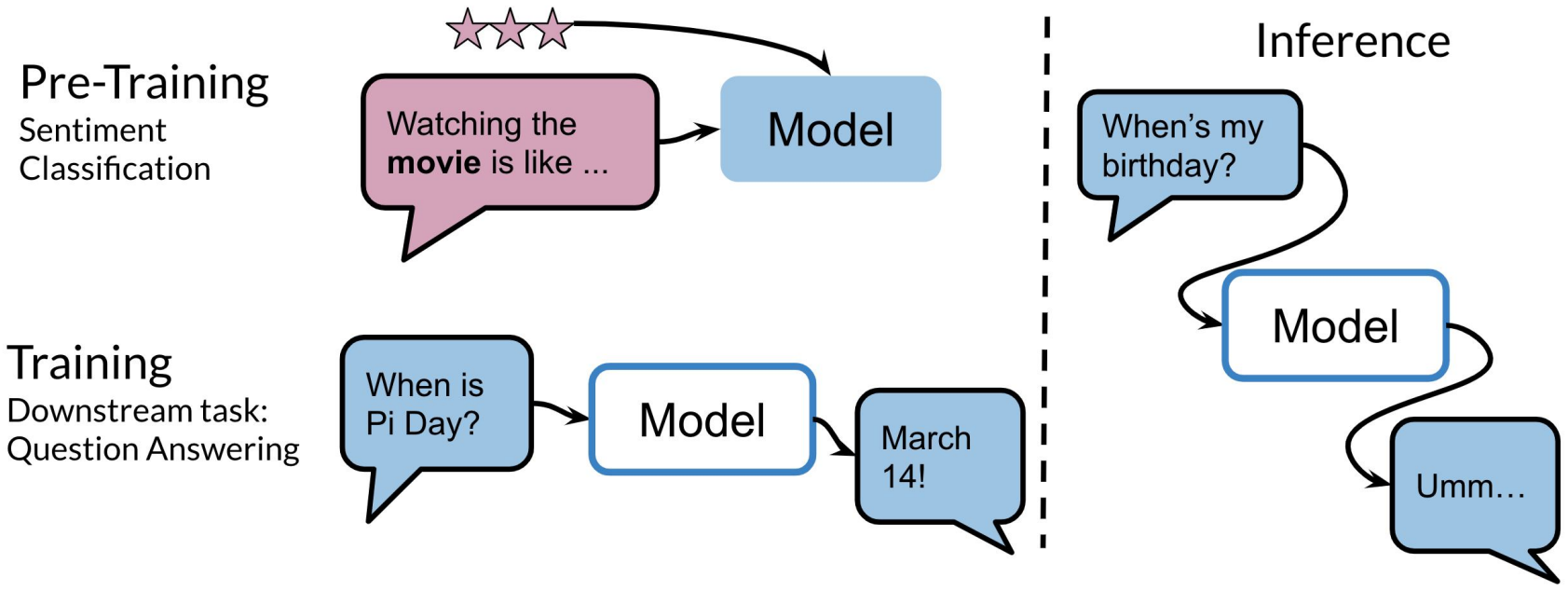


Desirable goals for Transfer learning

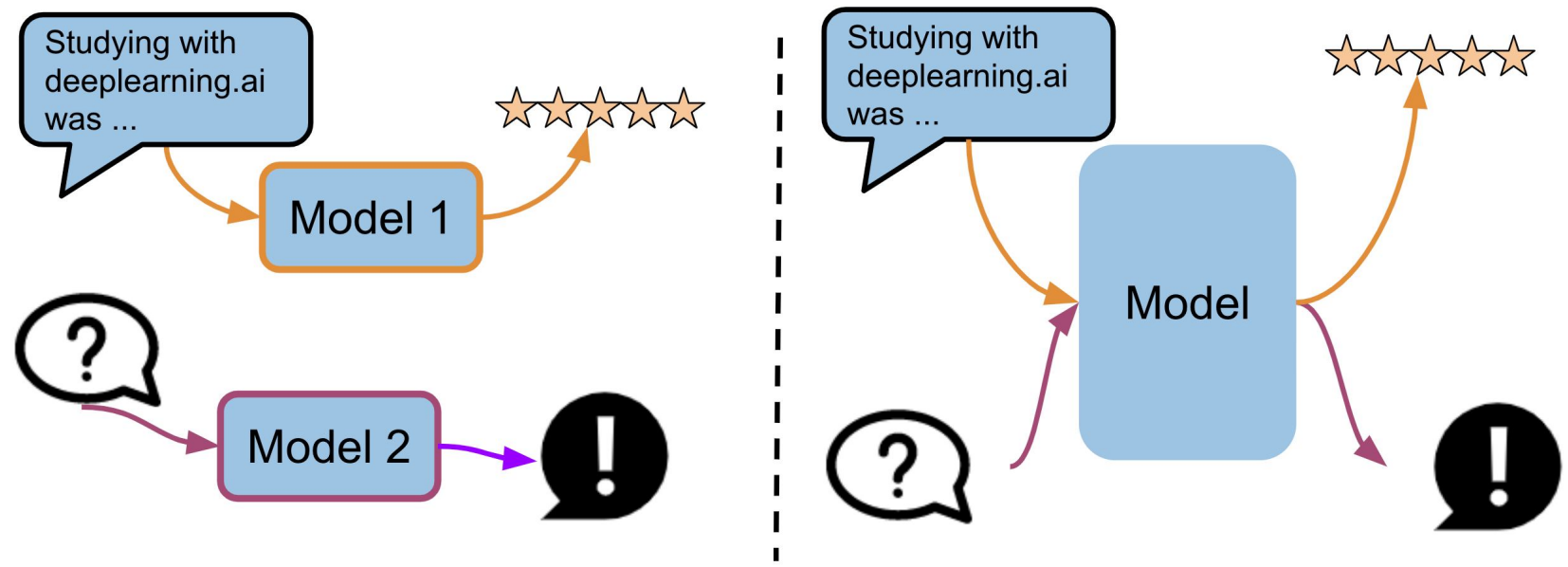
→ Reduce training time ⟹ Improve predictions
(converge faster)

→ Small datasets (already learned a lot from other tasks)

Welcome to Week 3! In this week you are going to learn about transfer learning and specifically you will understand how T5 and BERT actually work.



In the image above, you can see how a model initially trained on some type of sentiment classification, could now be used for question answering. One other model that has state of the art makes use of multi tasking. For example, the same model could be used for sentiment analysis, question answering, and many other things.



These new types of models make use of a lot of data. For example the C4 (colossal cleaned crawled corpus) is about 800 GB when all of the english wikipedia is just 13 GB!

deeplearning.ai

Transfer Learning in NLP

Desirable Goals

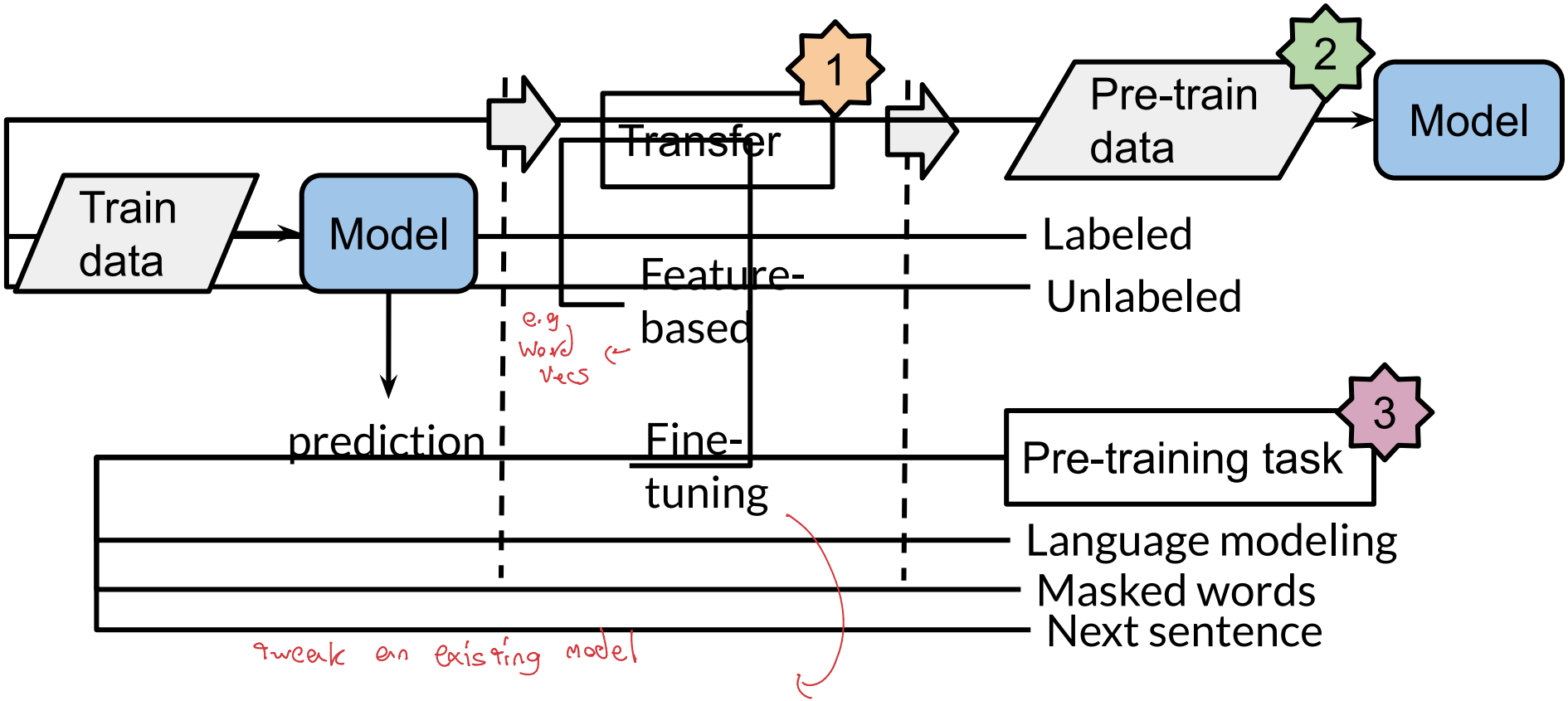
- Reduce training time

- Improve predictions

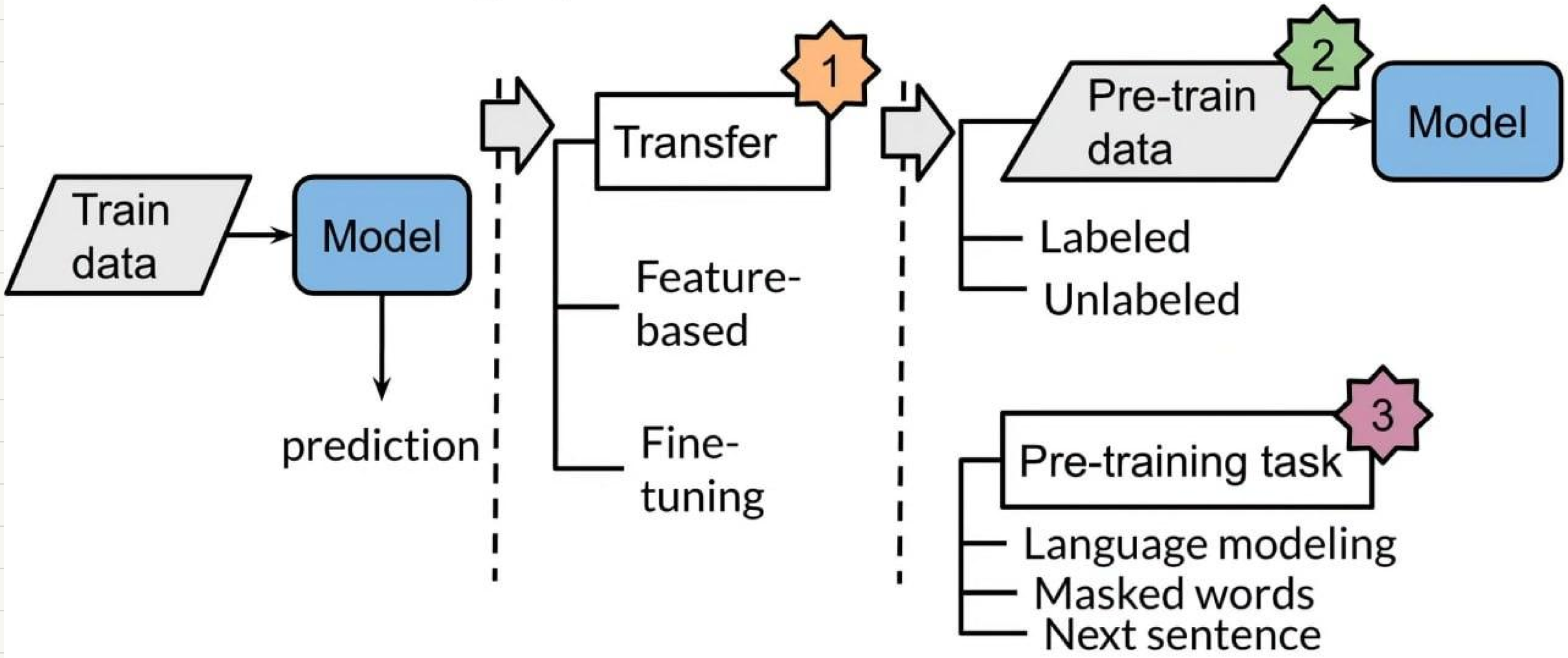
- Small datasets

Transfer Learning!

Transfer learning options

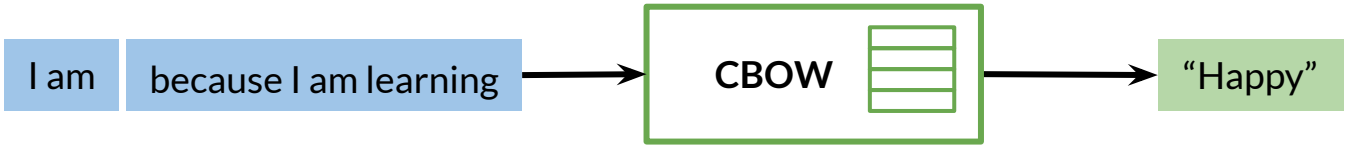


Transfer learning options

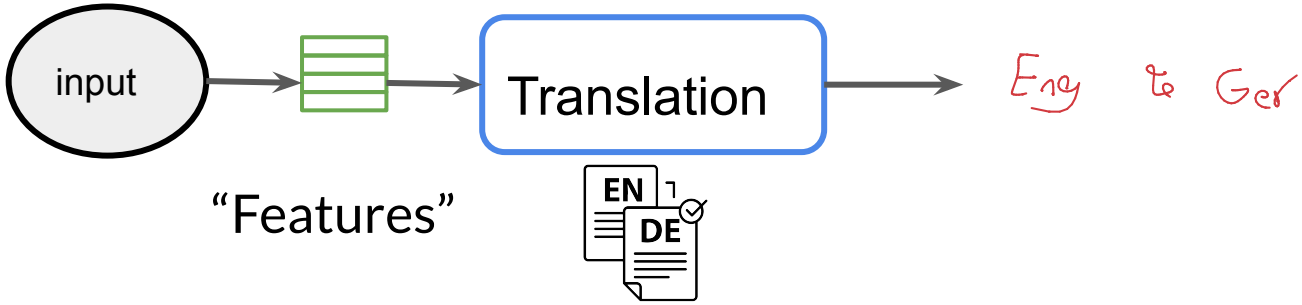


Transfer 1

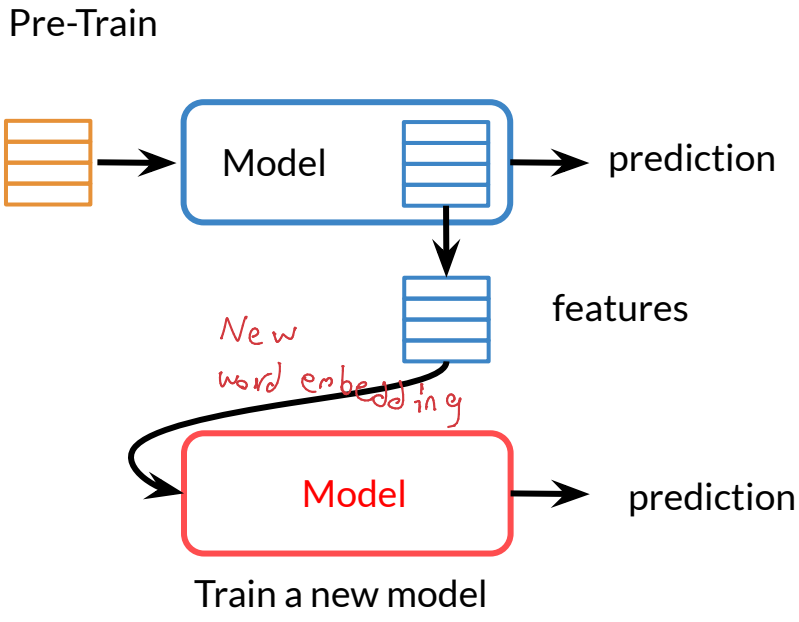
General purpose learning

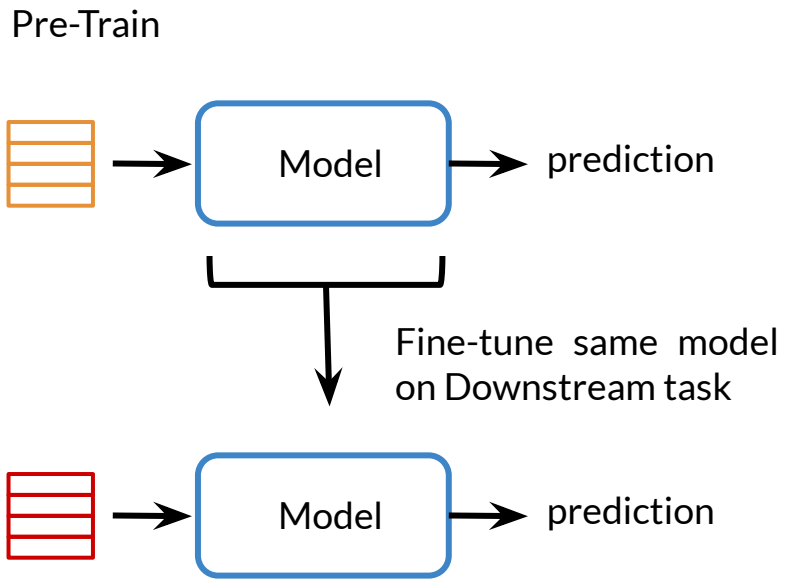


Word Embeddings



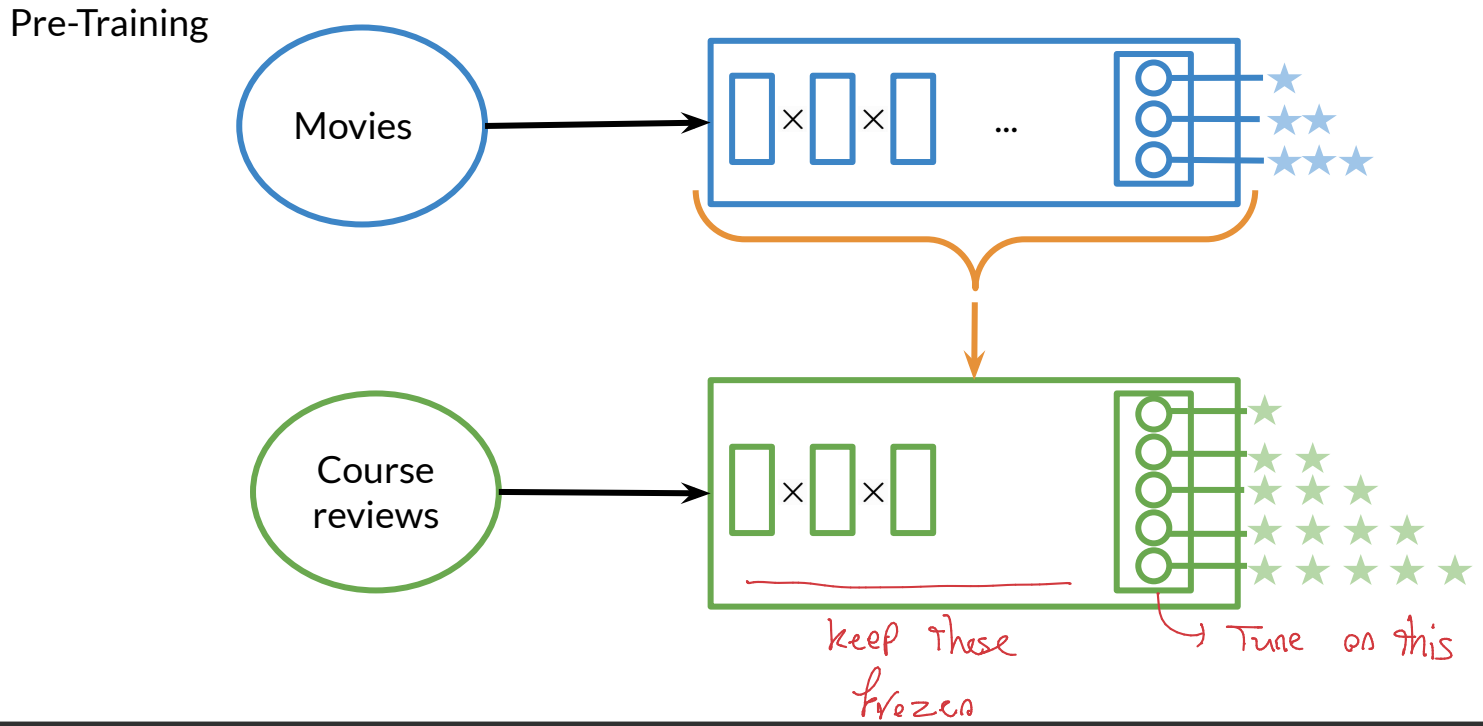
Feature-based vs. Fine-Tuning



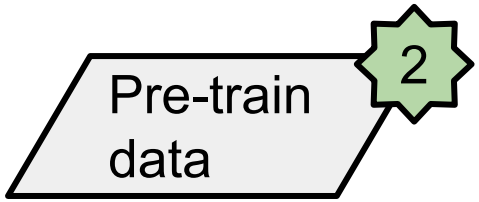


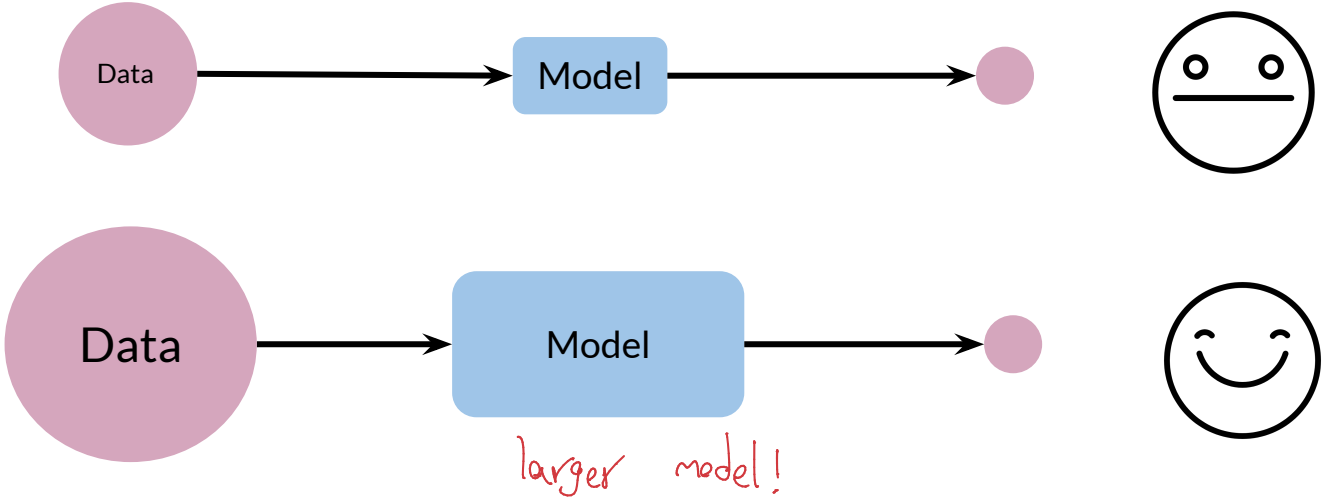
Fine-tune: adding a layer

Transfer 1

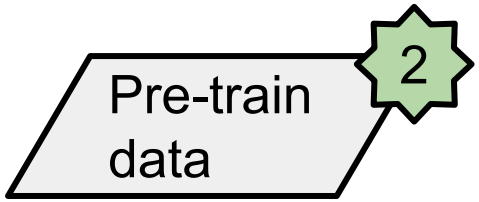


Data and performance



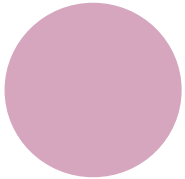


Labeled vs Unlabeled Data



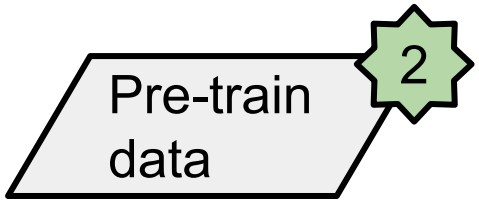
Labeled text data

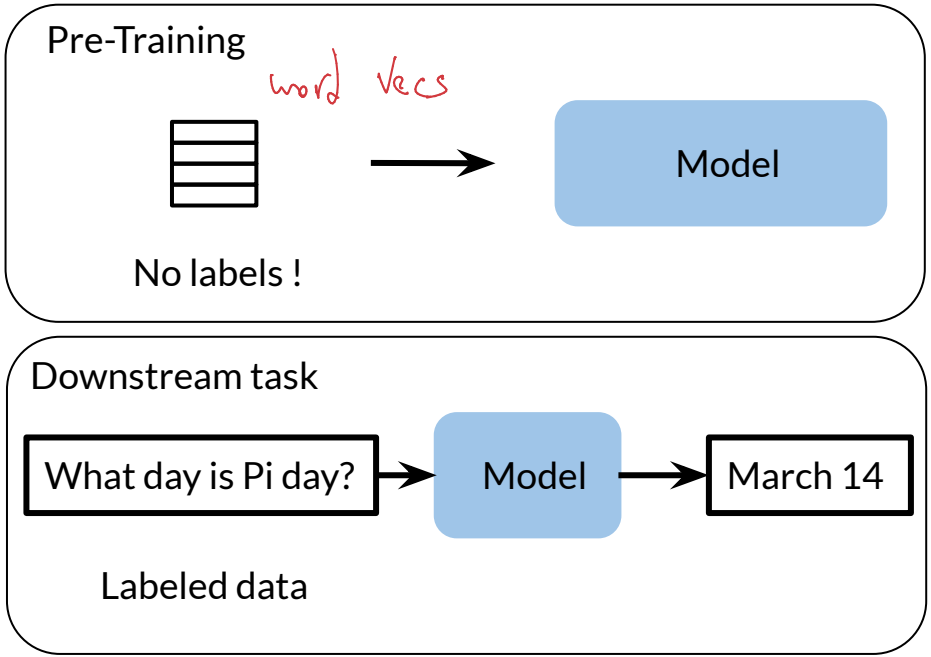
Unlabeled text data



you have way more unlabeled text data than labeled ones

Transfer learning with unlabeled data

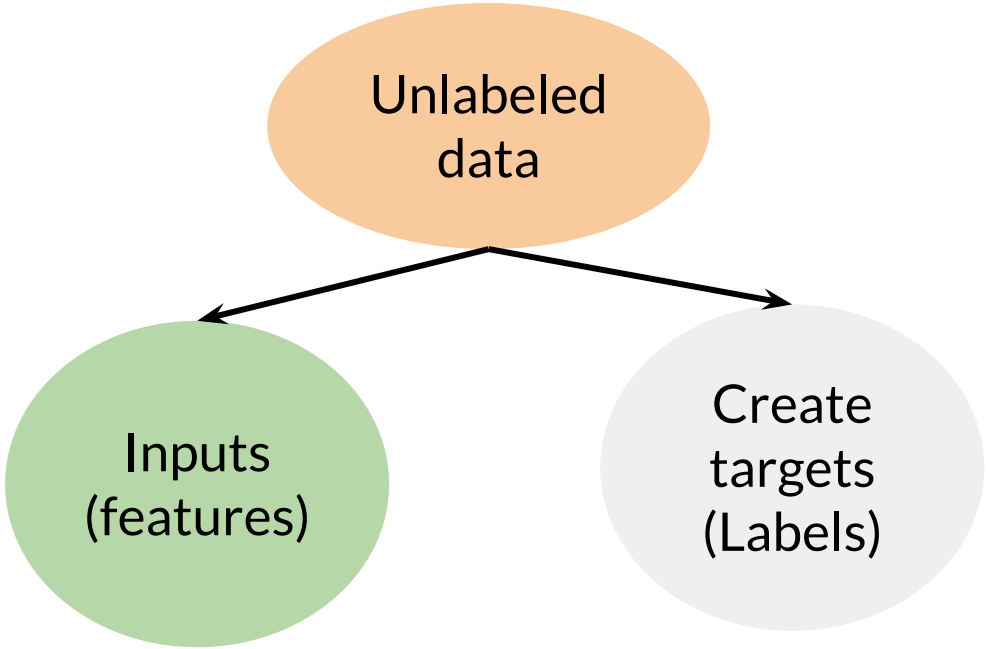




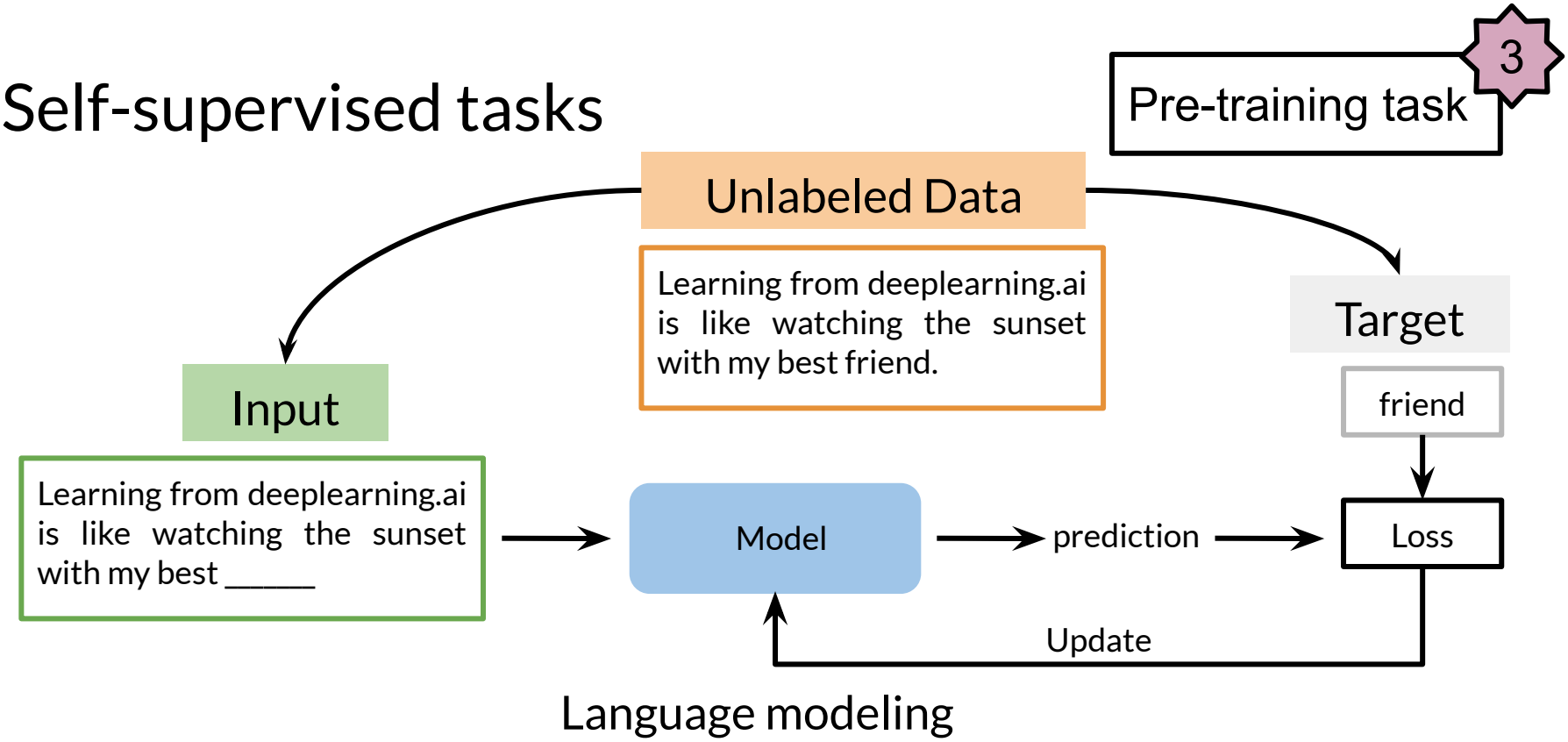
Which tasks work with **unlabeled** data?

Self-supervised task

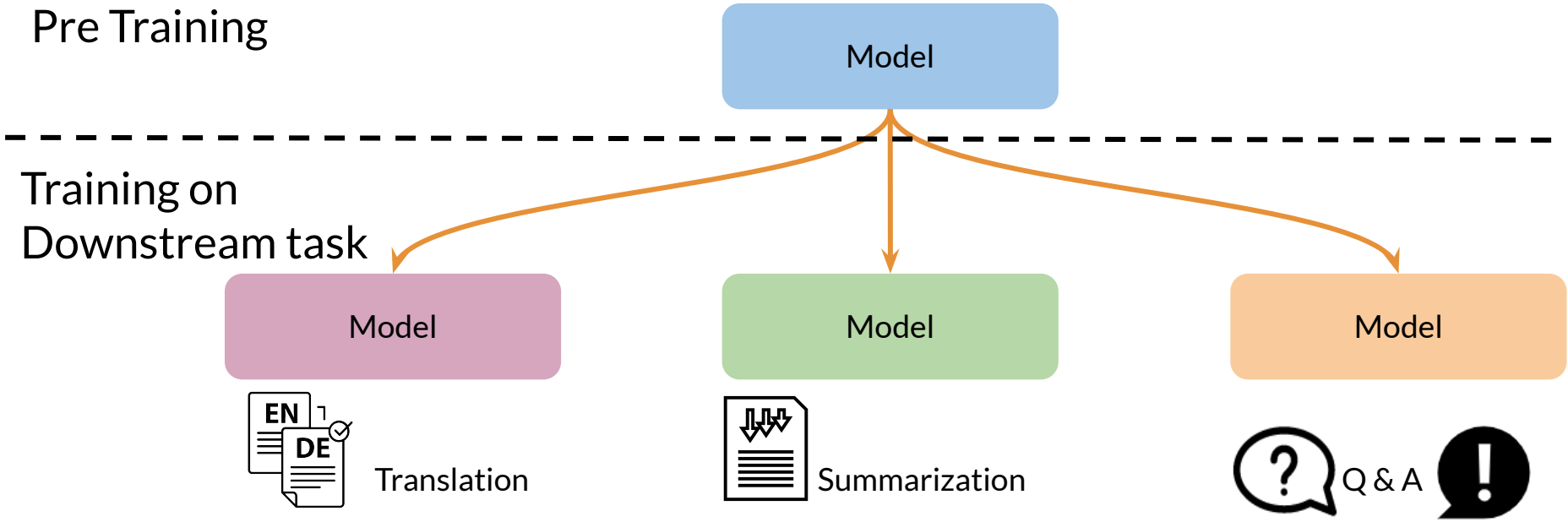
Pre-training task 3



Self-supervised tasks

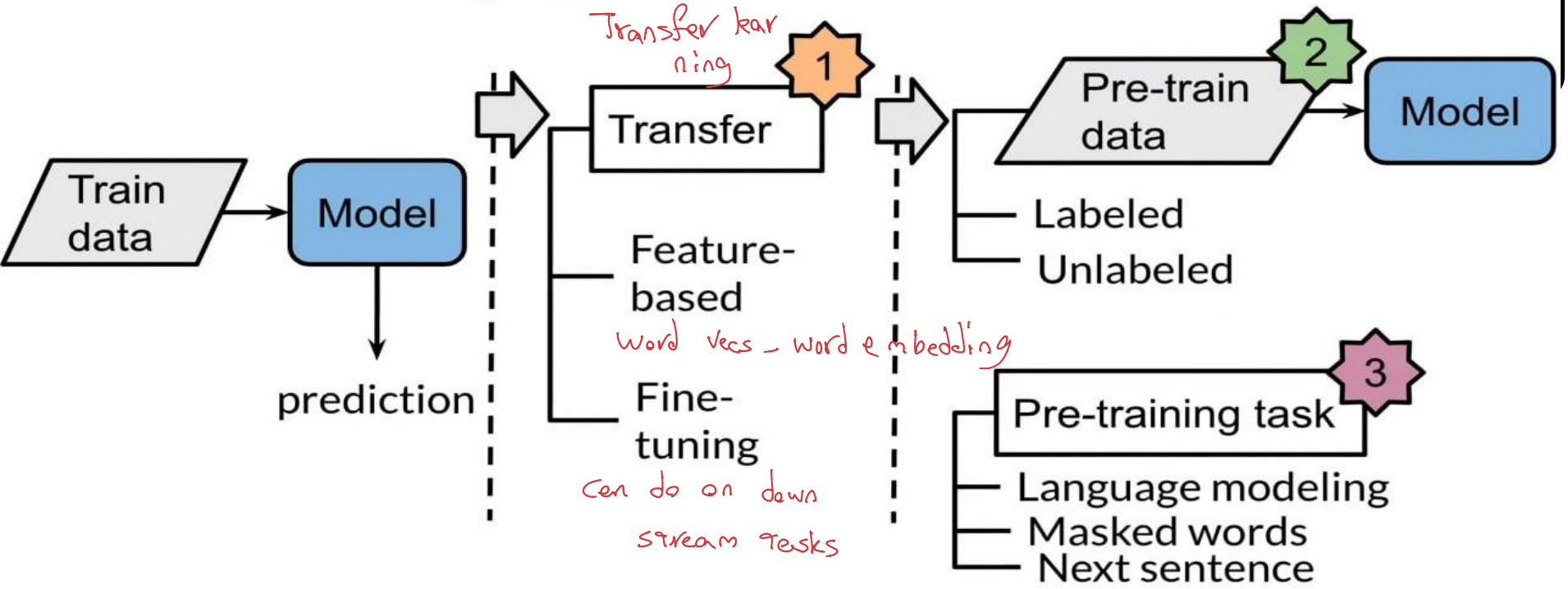


Fine-tune a model for each downstream task



Summary

Transfer learning options



Feature based: we obtain word embeddings from a pre-trained model and use these as inputs features for a different task

1. Pre-training with Labeled Data:

- **What it means:** In this case, you pre-train a model on a large labeled dataset, where each example has an associated label (supervised learning). The model learns to predict labels for input data.
- **Example:** Imagine you want to build a text classifier for medical documents, but you don't have a lot of labeled medical data. You can first pre-train a model on a large general-purpose labeled dataset, such as news articles labeled by topic. The model learns general features about text (sentence structures, common words, etc.), and then you fine-tune it on your specific medical document dataset.

2. Pre-training with Unlabeled Data:

- **What it means:** In this case, pre-training happens on a large dataset without labels (unsupervised learning). The goal is to learn useful representations (features) of the input data, like word embeddings or language models, that can be used for downstream tasks.
- **Example:** Pre-training models like BERT, GPT, or Word2Vec on a large amount of text from the internet is an example of using unlabeled data. These models learn patterns about language—like syntax, grammar, and the relationships between words—without needing labels for the text. Once pre-trained, the model can then be fine-tuned for tasks like sentiment analysis, machine translation, or question-answering, where you have labeled data.

Steps in Pre-training (unlabeled or labeled):

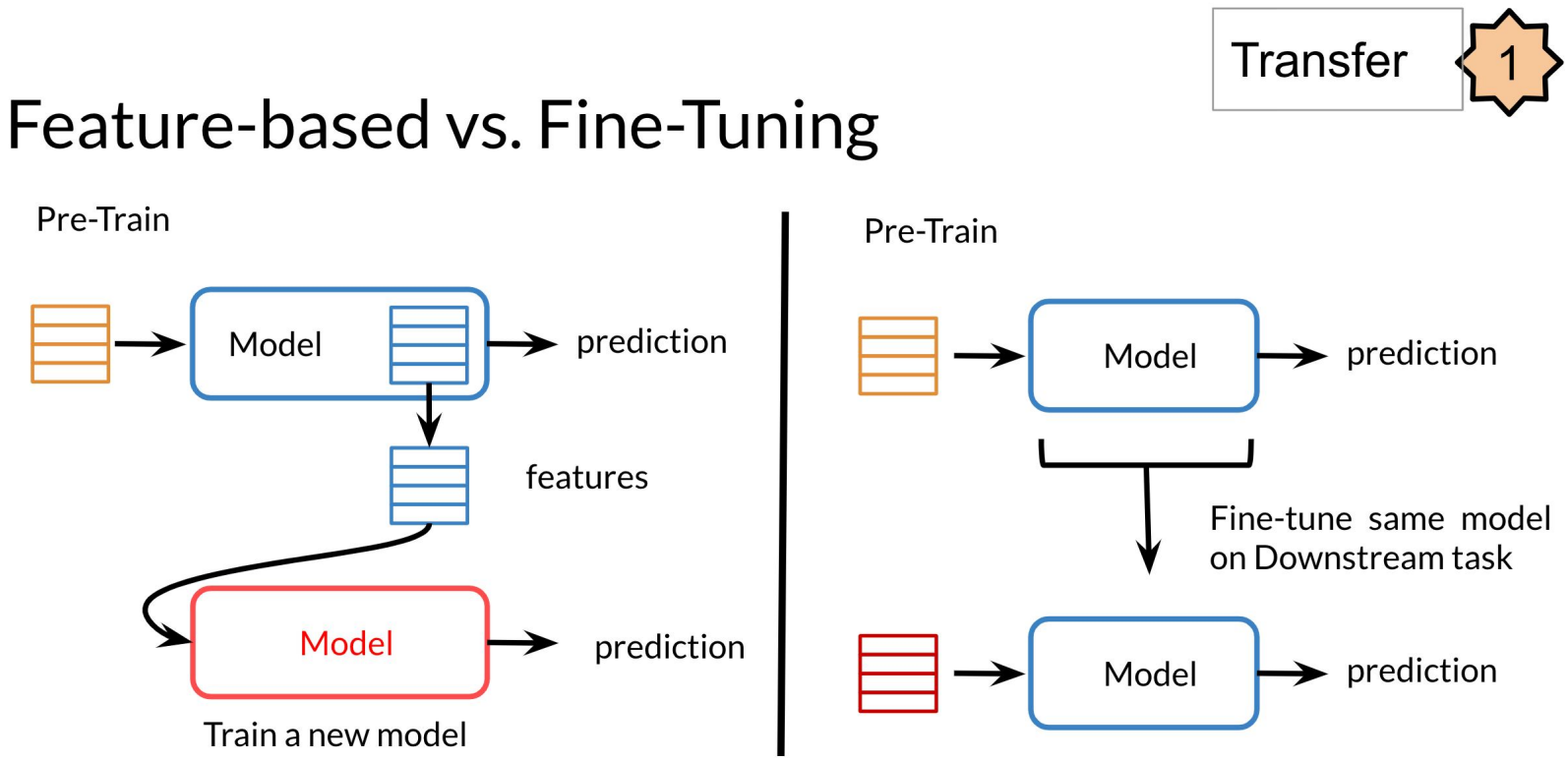
1. **Choose a Large Dataset:** Depending on your goal, select a dataset—either labeled or unlabeled—that's relevant to the problem.
    - For example, if you want to work with text, you might use Wikipedia, news articles, or a large corpus of general text (for unlabeled pre-training).
2. **Pre-train the Model:** Train the model on this dataset. For language tasks, this often involves learning representations of words (embeddings) or training a language model that can predict the next word, masked words, or sentence structure.
    - In **labeled data** pre-training, you'll train the model to predict labels (e.g., text classification).
    - In **unlabeled data** pre-training, you'll train the model to learn general features (e.g., word embeddings or language models like BERT).
3. **Fine-tune on Specific Task:** Once pre-trained, the model is adapted to the specific task using a smaller, task-specific dataset (which is usually labeled). You fine-tune the pre-trained model to make it more specialized.

In summary, **pre-training** is like teaching the model basic knowledge from a lot of data (either labeled or unlabeled) before teaching it more specialized knowledge for your specific task.

There are three main advantages to transfer learning:

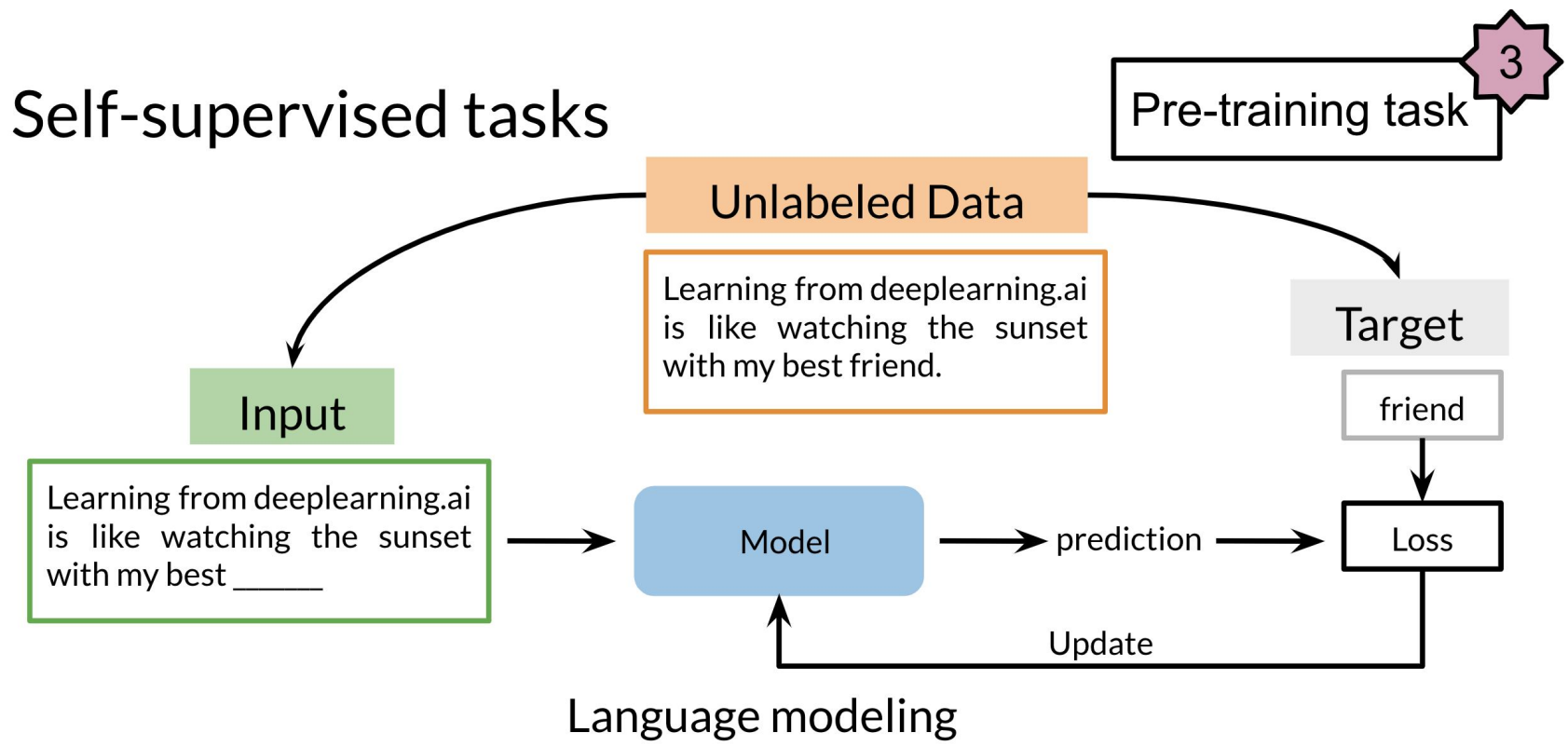
- Reduce training time
- Improve predictions
- Allows you to use smaller datasets

Two methods that you can use for transfer learning are the following:



In feature based, you can train word embeddings by running a different model and then using those features (i.e. word vectors) on a different task.

When fine tuning, you can use the exact same model and just run it on a different task. Sometimes when fine tuning, you can keep the model weights fixed and just add a new layer that you will train. Other times you can slowly unfreeze the layers one at a time. You can also use unlabelled data when pre-training, by masking words and trying to predict which word was masked.

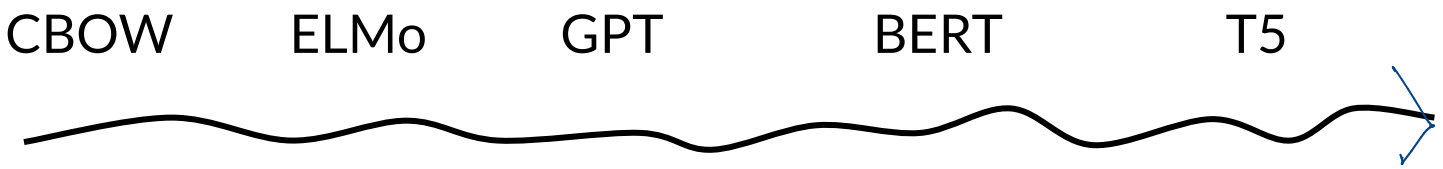


For example, in the drawing above we try to predict the word "friend". This allows your model to get a grasp of the overall structure of the data and to help the model learn some relationships within the words of a sentence.

deeplearning.ai

ELMo, GPT, BERT, T5

Outline



Context

… right …

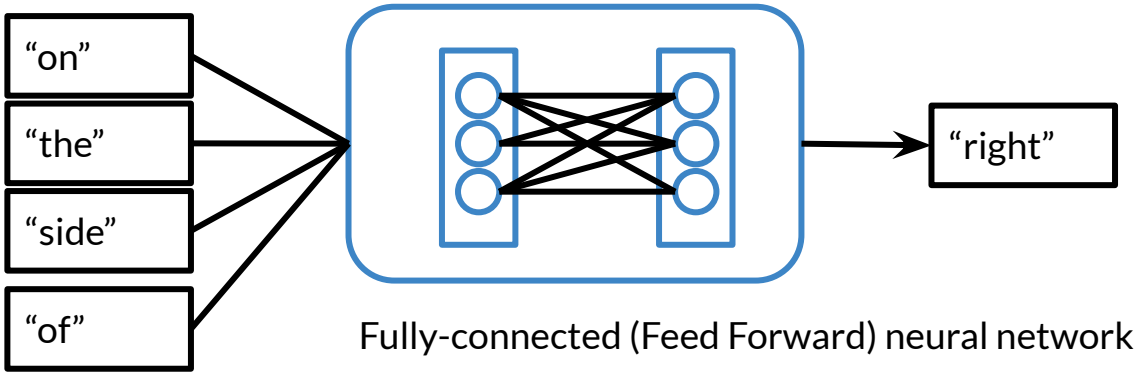
… they were on the right …

… they were on the right side of the street

Continuous Bag of Words

… they were on the right side of the street

Fixed window Fixed window



Fully-connected (Feed Forward) neural network

Need more context?

… they were on the right side of the street.

Fixed window Fixed window

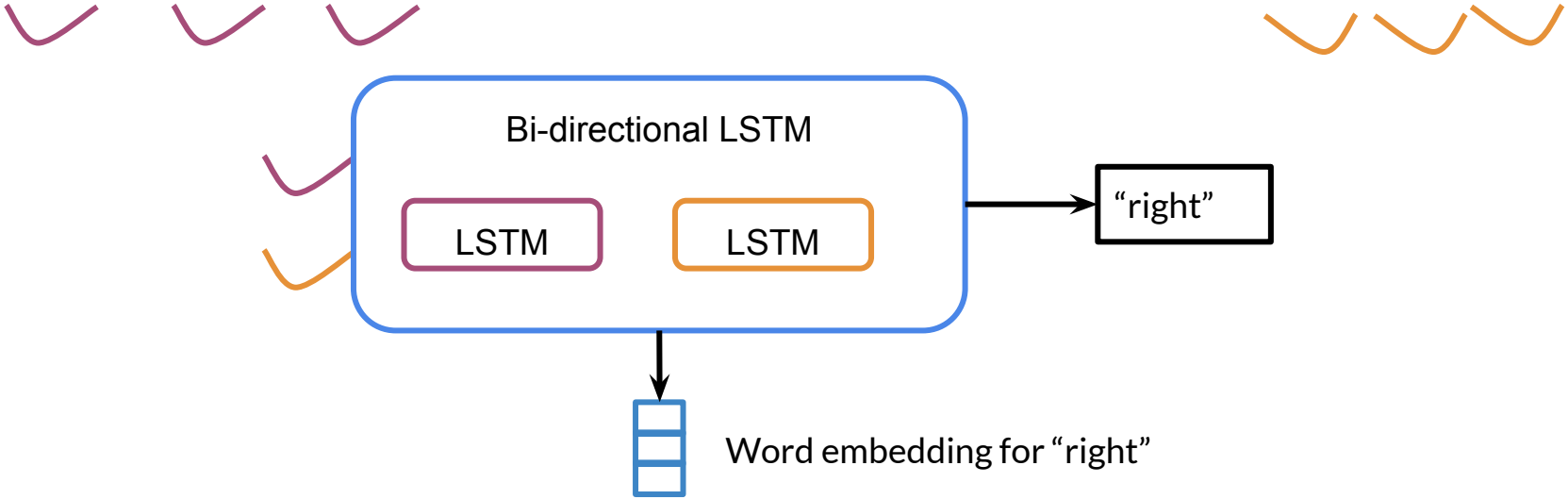
… they were on the right side of history.

Use all context words

The legislators believed that they were on the right side of history, so they changed the law.

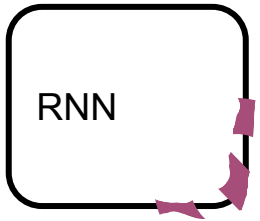
ELMo: Full context using RNN

The legislators believed that they were on the ____ side of history so they changed the law.

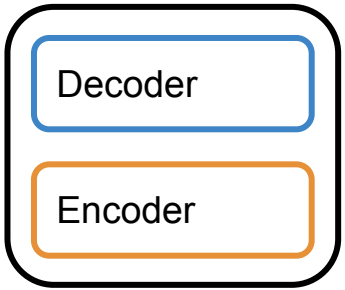


Open AI GPT

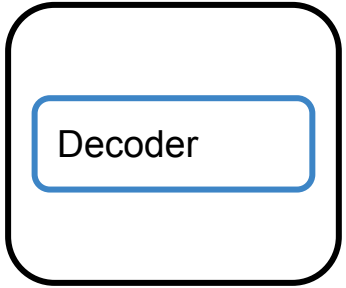
ELMo



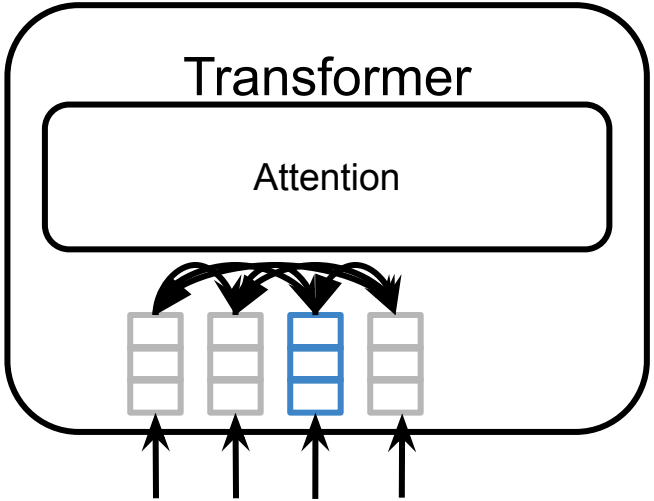
Transformer



GPT



Why not bi-directional?

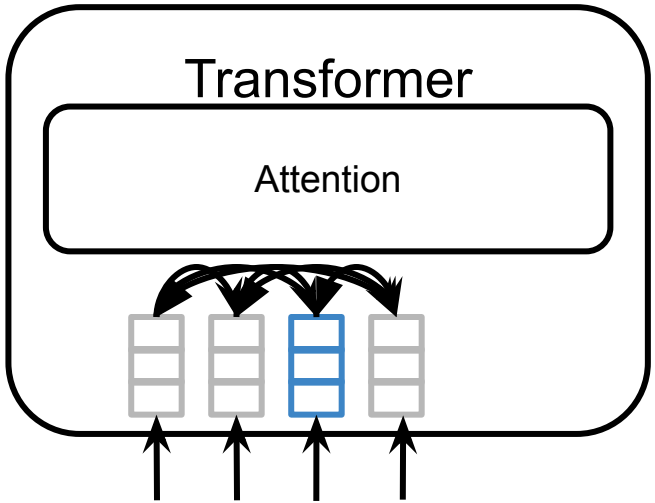


… on the right side...

Each word can peek at itself!

Model Type	Directionality	Description	Example Models	Foundation Models (FMs)
Encoder-Only	Bi-Directional	Processes the input text bidirectionally, capturing context from both left and right simultaneously.	BERT, RoBERTa, DistilBERT, Titan	-
Decoder-Only	Uni-Directional	Processes the input text in a single direction (usually left-to-right), generating each token based on previous tokens.	GPT-2, GPT-3, LLaMA	-
Encoder-Decoder	Bi-Directional (Encoder) + Uni-Directional (Decoder)	The encoder processes the input bidirectionally, while the decoder generates the output uni-directionally.	T5, BART, mT5, Titan	Amazon's Titan

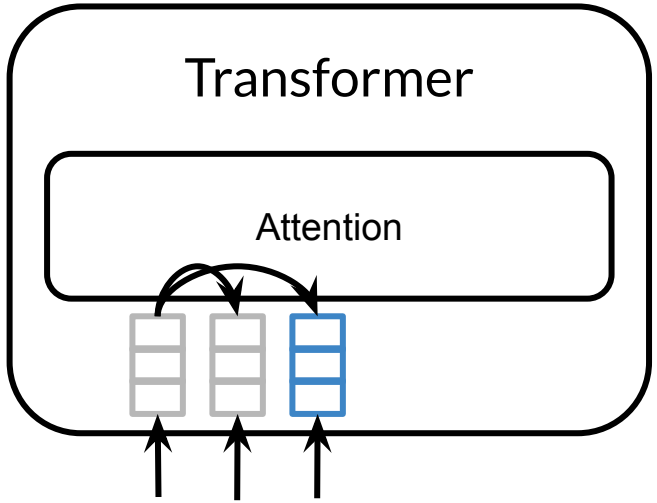
GPT: Uni-directional



... on the right side...

Each word can peek at itself!

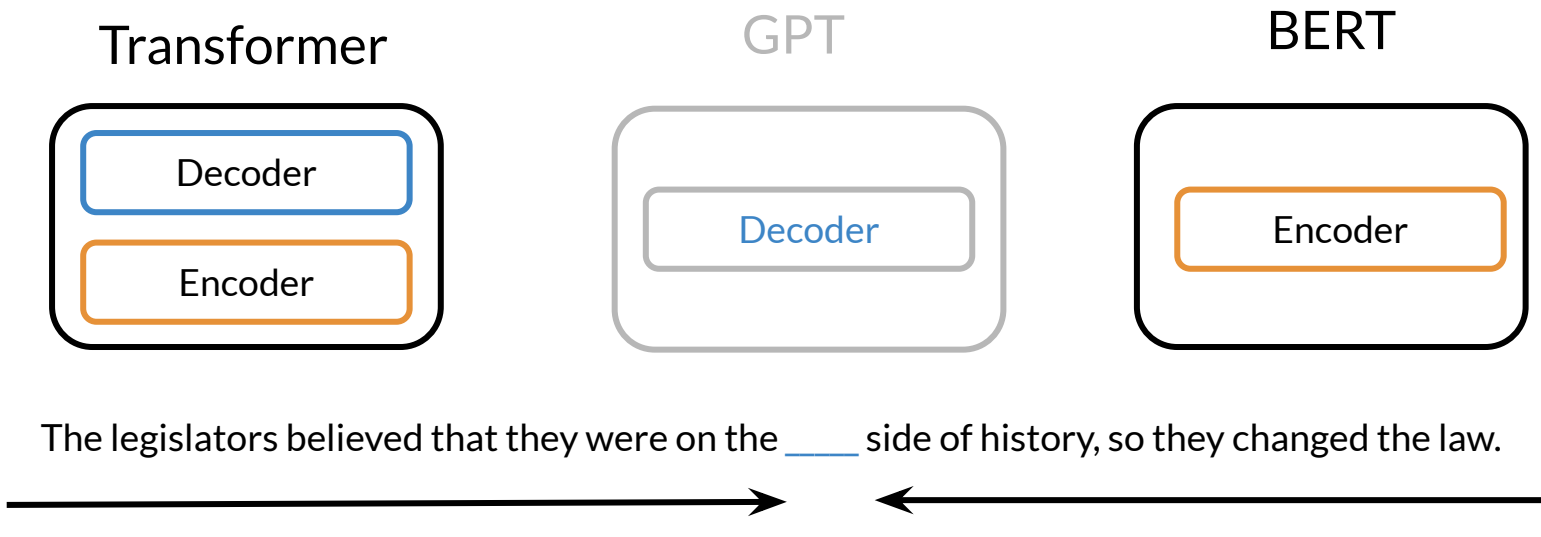
GPT



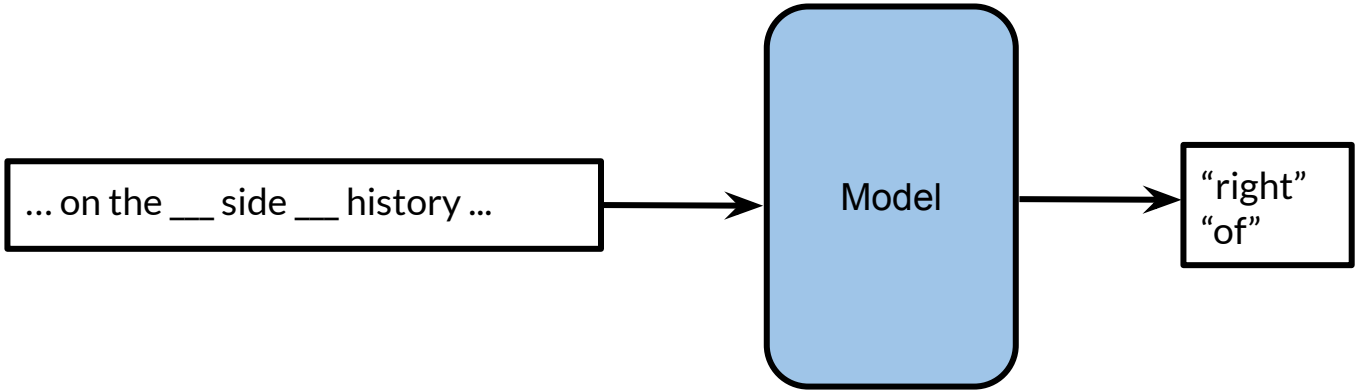
... on the right

No peeking!

BERT

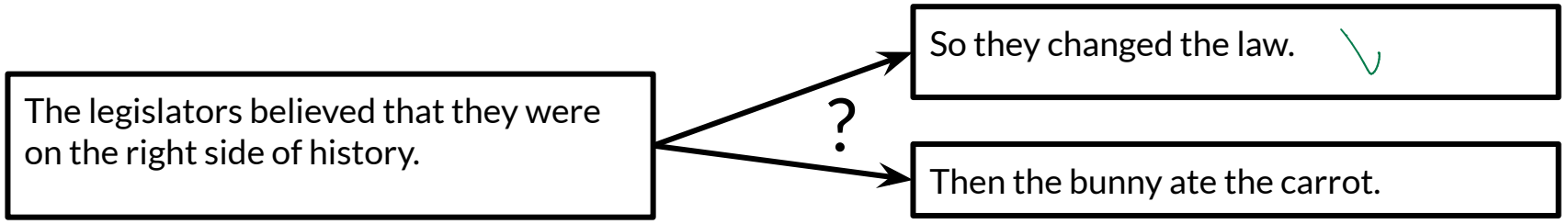

Transformer
Decoder
Encoder
GPT
Decoder
BERT
Encoder
The legislators believed that they were on the ____ side of history, so they changed the law.
Bi-directional

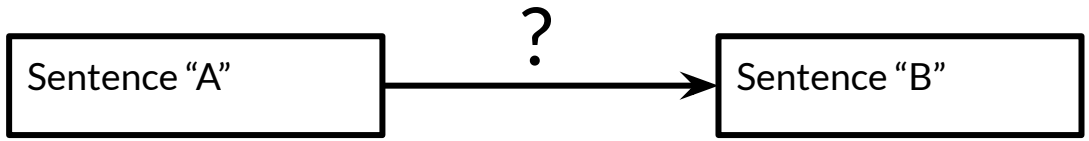
Transformer + Bi-directional Context



Multi-Mask Language Modeling

BERT: Words to Sentences

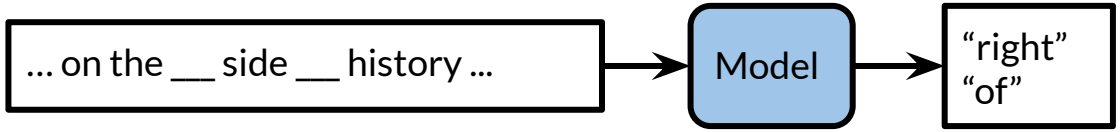




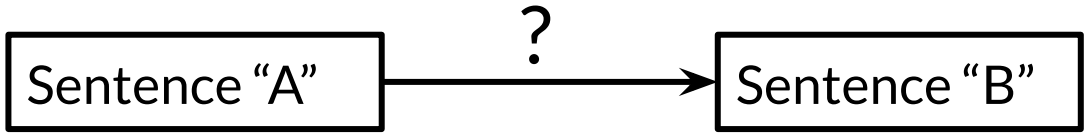
Next Sentence Prediction

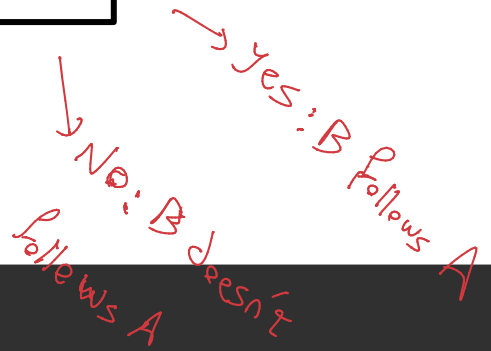
BERT Pre-training Tasks

Multi-Mask Language Modeling

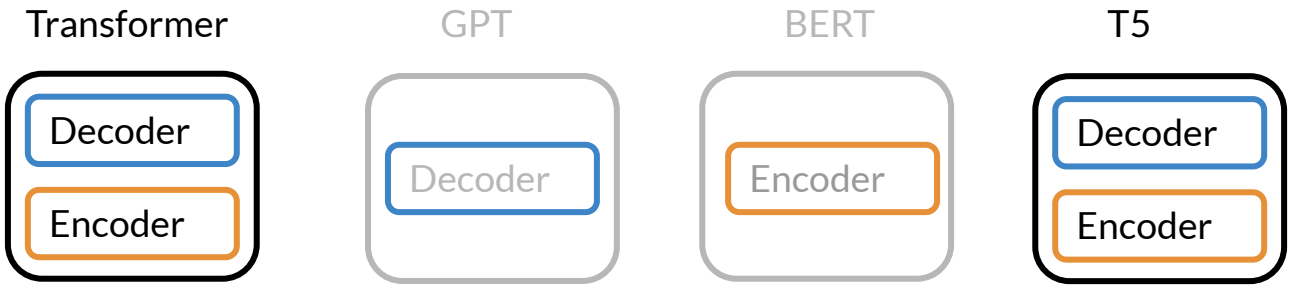


Next Sentence Prediction

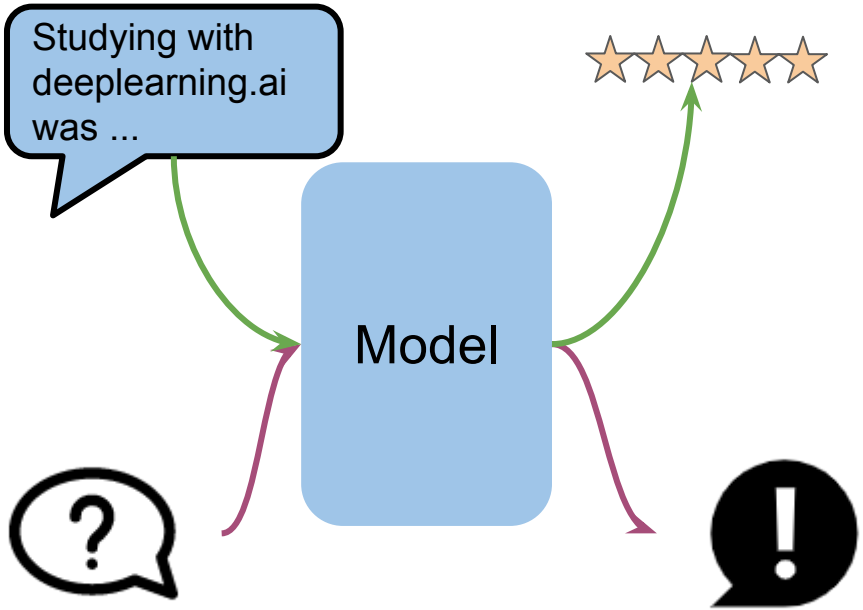




T5: Encoder vs. Encoder-Decoder

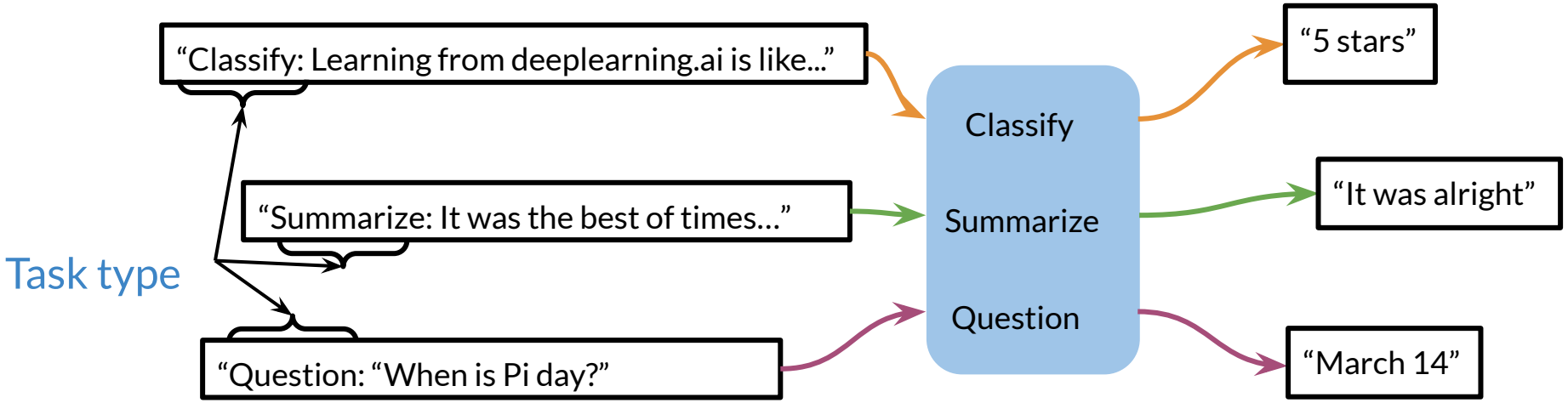


T5: Multi-task



How?

T5: Text-to-Text



Summary

More details next!

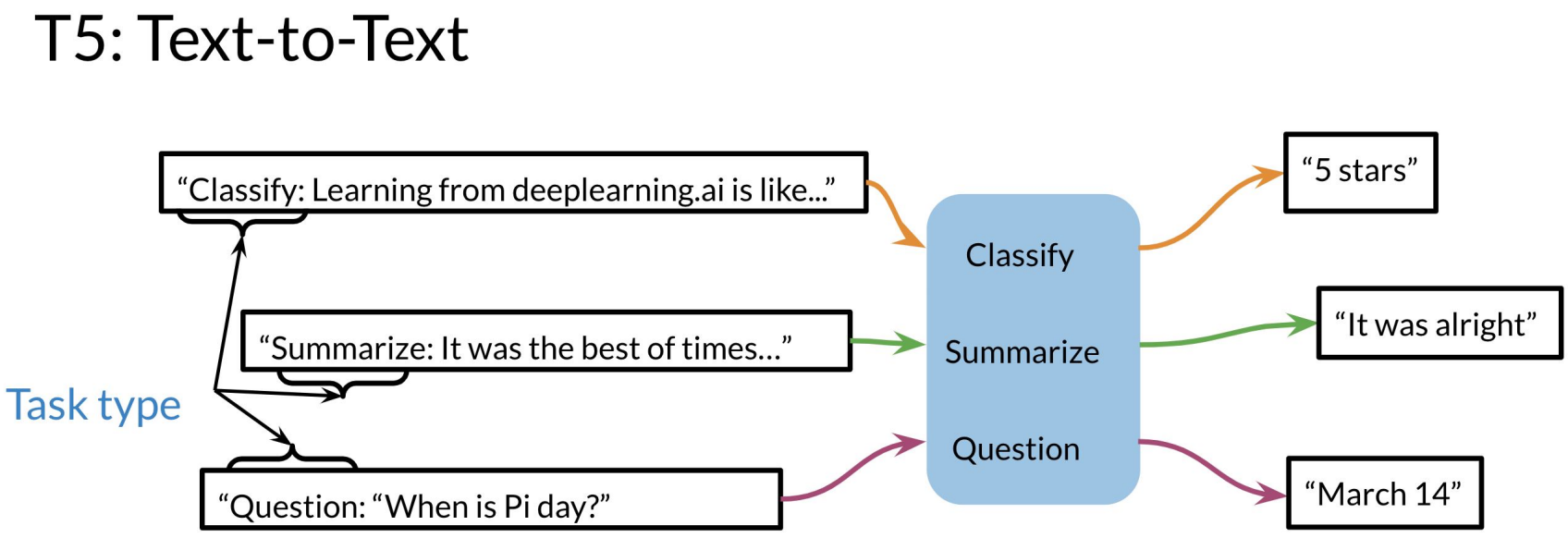
CBOW	ELMo	GPT	BERT	T5
Fixed Context window	Full sentence	Transformer: Decoder	Transformer: Encoder	Transformer: Encoder - Decoder
FFNN	Bi-directional Context	Uni-directional Context	Bi-directional Context	Bi-directional Context
	RNN		Multi-Mask	Multi-Task
			Next Sentence Prediction	

The models mentioned in the previous video were discovered in the following order.

In CBOW, you want to encode a word as a vector. To do this we used the context before the word and the context after the word and we use that model to learn and creates features for the word. CBOW however uses a fixed window C (for the context).

What's ElMo does, it uses a bi-directional LSTM, which is another version of an RNN and you have the inputs from the left and the right.

Then Open AI introduced GPT. GPT unfortunately is uni-directional but it makes use of transformers. Although ElMo was bi-directional, it suffered from some issues such as capturing longer-term dependencies. Transformers help with that, but since GPT was still unidirectional, BERT was then introduced which stands for the Bi-directional Encoder Representation from Transformers. Last but not least, T5 was introduced which makes use of transfer learning and uses the same model to predict on many tasks. Here is an illustration of how it works.



deeplearning.ai

Bidirectional Encoder Representations from Transformers (BERT)

Outline

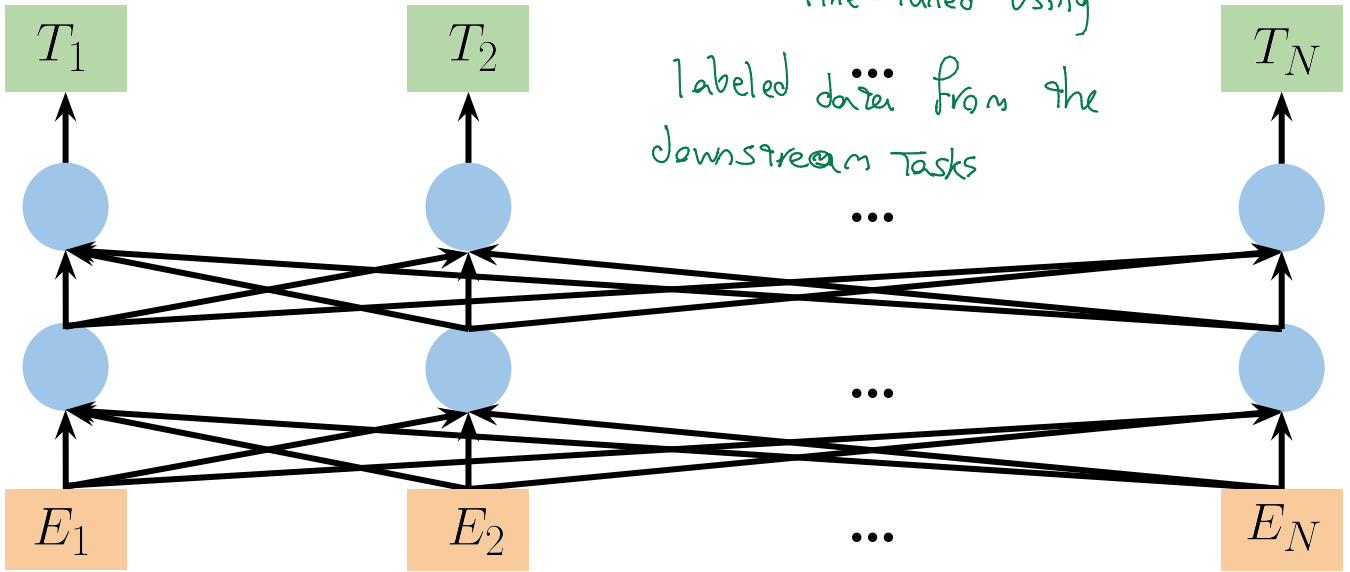
- Learn about the BERT architecture

- Understand how BERT pre-training works

BERT

2 steps { 1- Pre-Training: the model is trained on unlabeled data over dif Pre-Training task
2- Fine-Tuning: The bert model is first init with a Pre-Trained Params and all of Params are fine-Tuned using labeled datas from the downstream tasks

- Makes use of transfer learning/pre-training:



→ Transformer block

input Embedding

BERT

- A multi layer bidirectional transformer
- Positional embeddings
- BERT_base:
    - 12 layers (12 transformer blocks)
    - 12 attentions heads
    - 110 million parameters

BERT pre-training : Mask 15% of the words

After school Lukasz does his _____ in the library.

- Masked language modeling (MLM)

BERT pre-training

After school Lukasz does his homework in the library.

After school _____ his homework in the _____ .

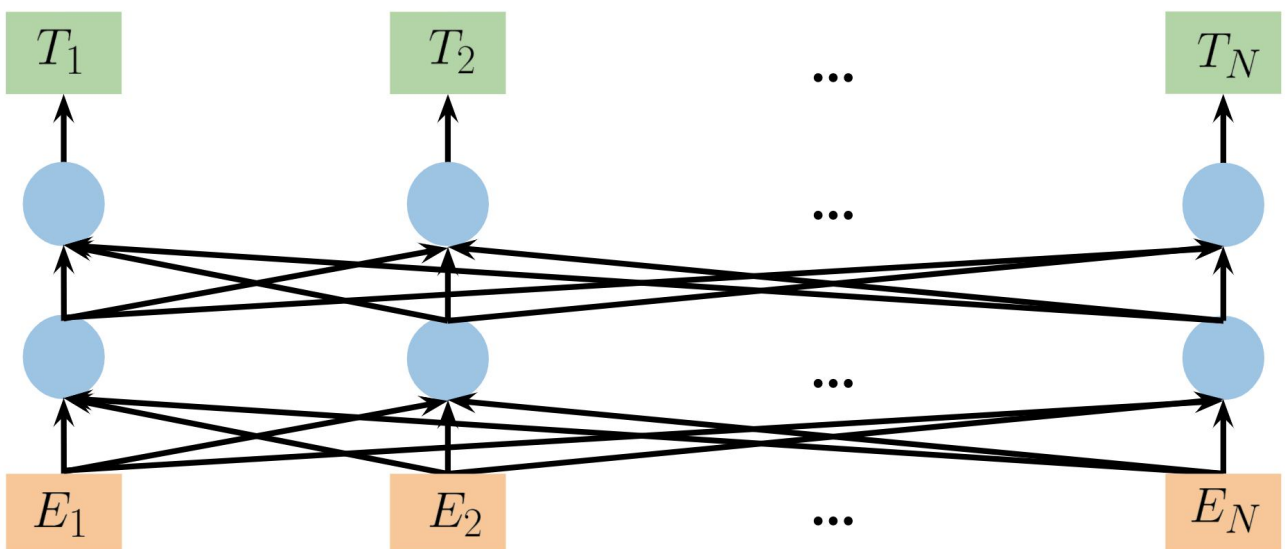
Summary

- 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.
- There could be multiple masked spans in a sentence

- Next sentence prediction is also used when pre-training.

You will now learn about the BERT architecture and understand how the pre-training works.

- Makes use of transfer learning/pre-training:



There are two steps in the BERT framework: pre-training and fine-tuning. During pre-training, the model is trained on unlabeled data over different pre-training tasks. For fine tuning, the BERT model is first initialized with the pre-trained parameters, and all of the parameters are fine-tuned using labeled data from the downstream tasks. For example, in the figure above, you get the corresponding embeddings for the input words, you run it through a few transformer blocks, and then you make the prediction at each time point T_i.

- 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.
- There could be multiple masked spans in a sentence
- Next sentence prediction is also used when pre-training.

The next video will talk about the BERT objective.

When BERT selects 15% of the tokens at random during pre-training:

- **80% of the time**: The selected token is replaced with the special `[MASK]` token.
- **10% of the time**: The selected token is replaced with a random token from the vocabulary.
- **10% of the time**: The selected token is left unchanged (kept as is).

Why does BERT do this?

This method adds some variability and difficulty to the task, ensuring the model doesn't overfit to always expect `[MASK]` tokens and learn shortcuts to predict the masked token. Let me break down the reasoning:

1. **Masking 80% of the time**: This is the core of the **masked language modeling (MLM)** objective. The model must predict the original word based on the context, which teaches it to deeply understand word relationships in both directions (left-to-right and right-to-left).
2. **Replacing with a random token 10% of the time**: By introducing a random token, the model learns to not only predict missing words but also to handle noise and understand that not everything will be perfect. This encourages robustness and prevents the model from relying too heavily on always seeing `[MASK]`.
3. **Keeping the token as is 10% of the time**: This prevents the model from becoming too reliant on knowing that certain tokens are always missing. It forces the model to maintain accurate predictions, even when no masking occurs and continue to learn the language relationships.

Why does BERT use this strategy?

- **Prevent overfitting**: If BERT replaced tokens with `[MASK]` 100% of the time, the model might overfit to predicting based on the presence of the `[MASK]` token. The 10% variability helps the model learn more flexible representations.
- **Encourage contextual understanding**: By sometimes keeping the original token, BERT is forced to understand both when words are missing and when they are not, reinforcing its bidirectional understanding of context.
- **Handle noise**: Random token replacement teaches the model to deal with imperfect inputs, making it more robust to real-world text, where there might be typos, incorrect words, or noise.

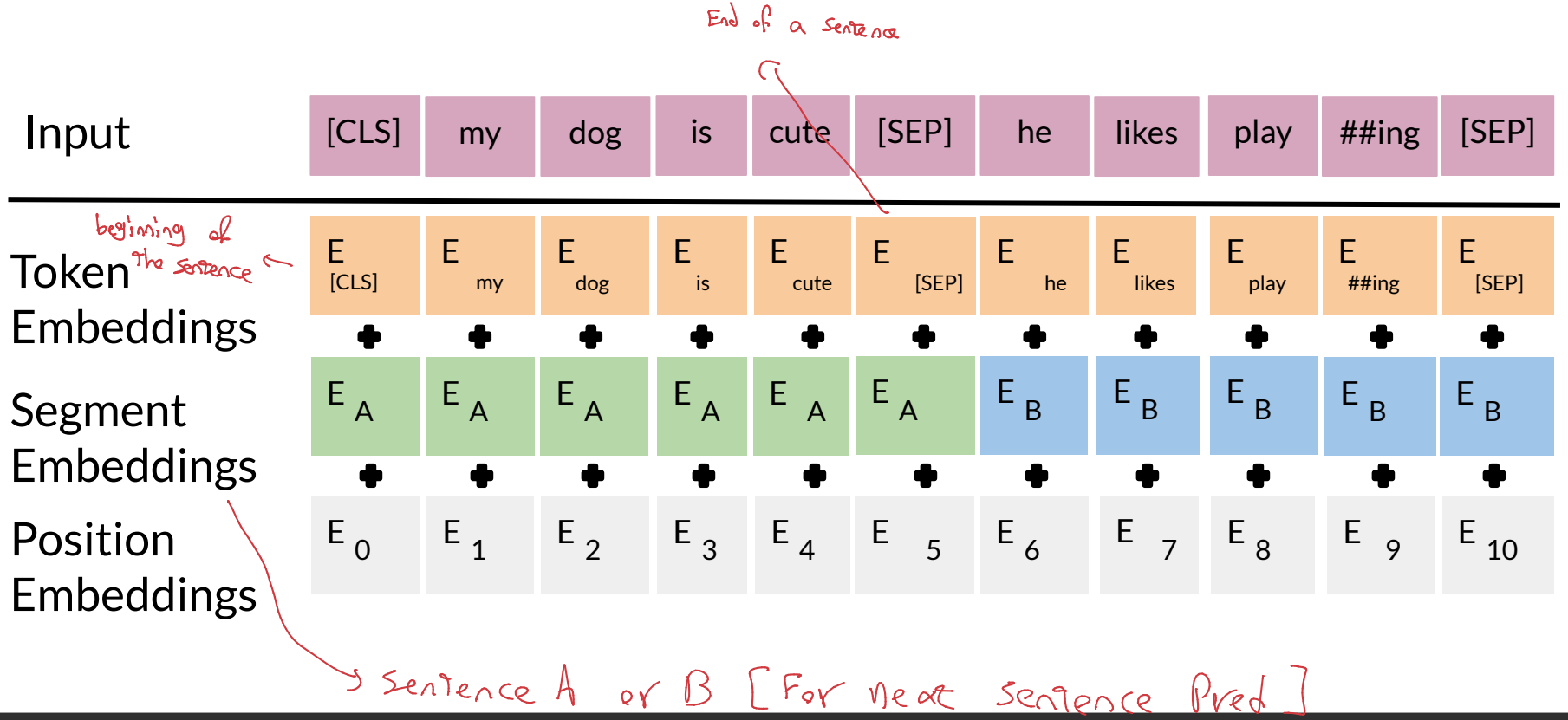
deeplearning.ai

BERT Objective

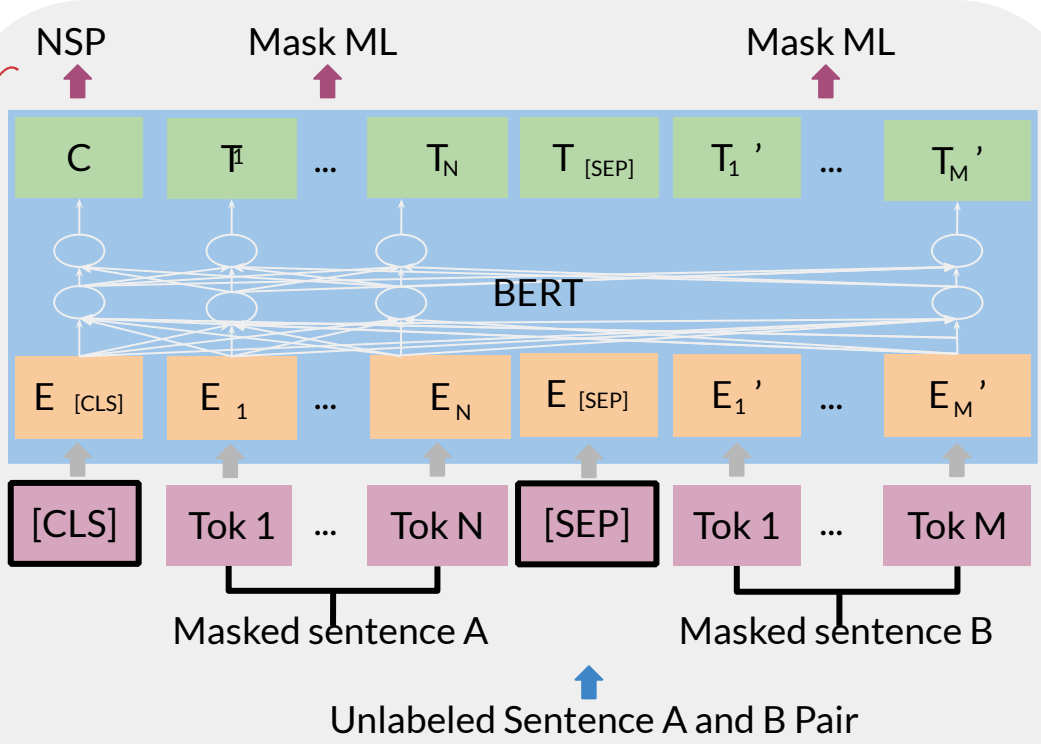
Outline

- Understand how BERT inputs are fed into the model
- Visualize the output
- Learn about the BERT objective

Formalizing the input



Visualizing the output



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

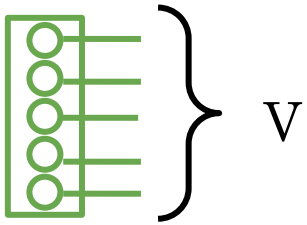
- [SEP]: a special separator token

BERT is open source!

BERT Objective

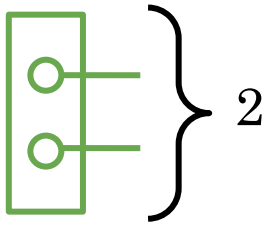
Objective 1:
Multi-Mask LM

Loss: Cross Entropy Loss



Objective 2:
Next Sentence Prediction

Loss: Binary Loss

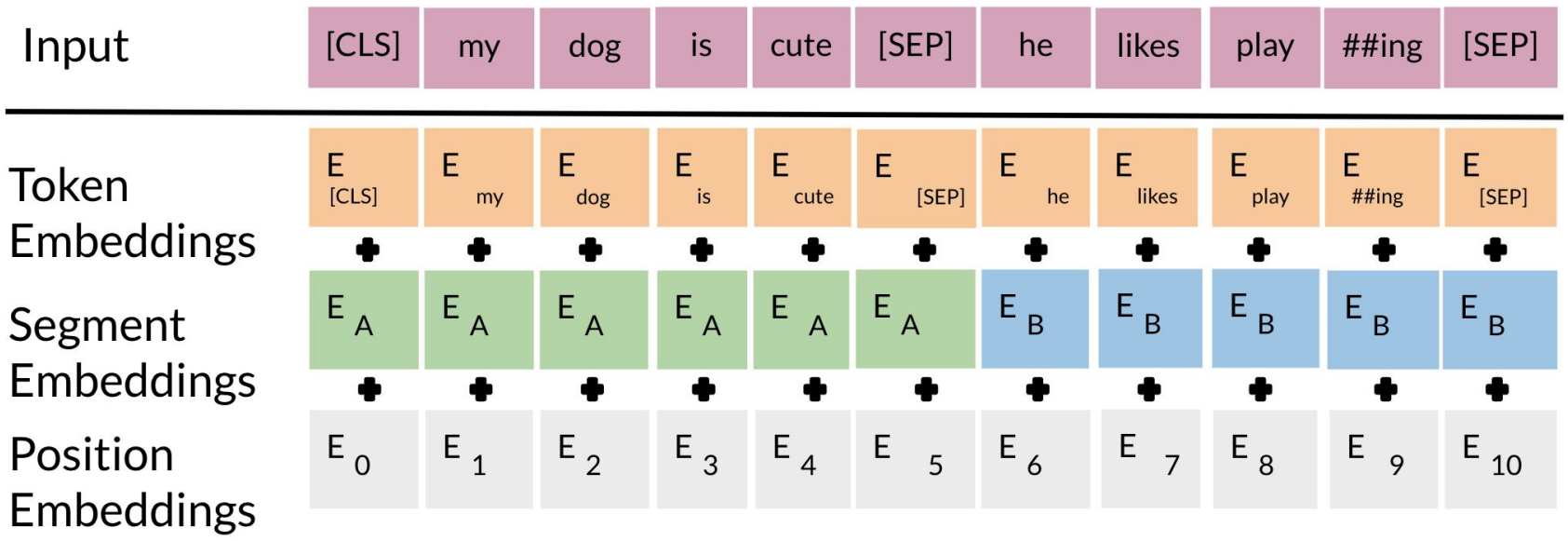


Summary

- BERT objective

- Model inputs/outputs

We will first start by visualizing the input.

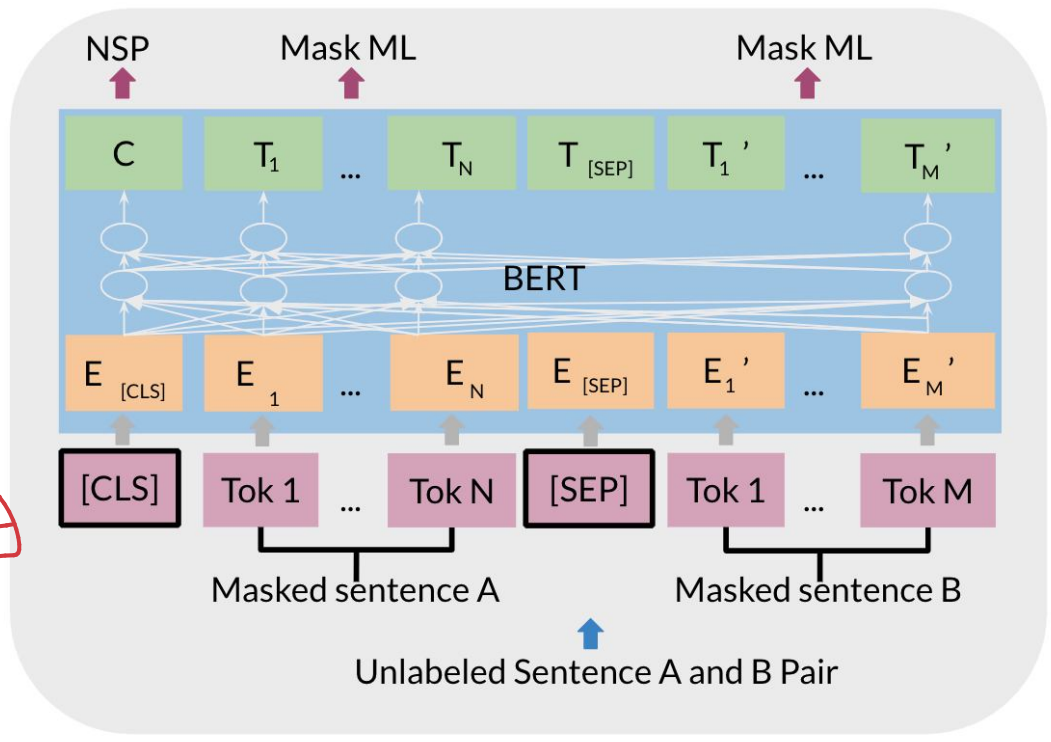


The input embeddings are the sum of the token embeddings, the segmentation embeddings and the position embeddings.

The input embeddings: you have a CLS token to indicate the beginning of the sentence and a sep to indicate the end of the sentence

The segment embeddings: allows you to indicate whether it is sentence a or b.

Positional embeddings: allows you to indicate the word's position in the sentence.

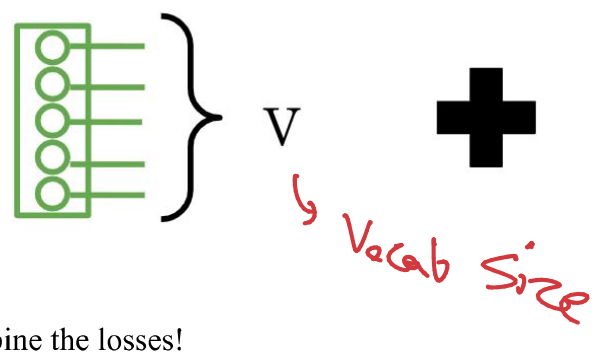


- [CLS]: a special classification symbol added in front of every input
- [SEP]: a special separator token

The C token in the image above could be used for classification purposes. The unlabeled sentence A/B pair will depend on what you are trying to predict, it could range from question answering to sentiment. (in which case the second sentence could be just empty). The BERT objective is defined as follows:

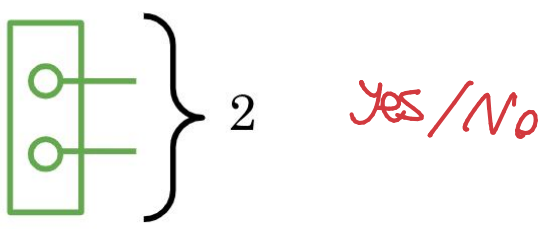
Objective 1:
Multi-Mask LM

Loss: Cross Entropy Loss



Objective 2:
Next Sentence Prediction

Loss: Binary Loss



You just combine the losses!

The "C" token in BERT's output, as shown in the second image, refers to the output corresponding to the special `[CLS]` token. This output is particularly useful for tasks like **Next Sentence Prediction (NSP)** during BERT's pre-training. Let me explain how it's used in detail.

How Next Sentence Prediction (NSP) works:

1. Input Setup:

- During pre-training, BERT receives two sentences (or segments) as input, which are concatenated together. These segments are separated by the special `[SEP]` token.
- The first sentence is called **sentence A**, and the second is **sentence B**.
- BERT adds the special `[CLS]` token at the beginning of the input sequence. This token will later hold the overall representation of the entire input.

2. Objective of NSP:

The task is to predict whether sentence B logically follows sentence A in the original text. BERT does this by classifying the relationship between the two sentences:

- **Label is 1 (True)** if sentence B follows sentence A in the original text.
- **Label is 0 (False)** if sentence B is a randomly selected sentence that does not follow sentence A.

3. Role of the `[CLS]` token:

- After the input passes through BERT's Transformer layers, the output corresponding to the `[CLS]` token (denoted as **C** in the image) holds a fixed-size, pooled representation of the entire input (sentences A and B).
- BERT uses this `[CLS]` token's output to make a prediction for the NSP task.

4. How the prediction works:

- The `[CLS]` token's output (represented as vector **C**) is fed into a classifier (typically a simple feedforward neural network with a softmax layer) to predict the probability of the two possible classes:
  - **Class 1**: Sentence B follows sentence A (True).
  - **Class 0**: Sentence B is a random sentence (False).

The classifier looks at the overall context of bo entences, which is encoded into the `[CLS]` token output.

In more detail:

- After processing the input sequence (with `[CLS]`, both sentences, and `[SEP]` tokens), BERT outputs a sequence of vectors, one for each token.
- The vector corresponding to the `[CLS]` token is special because it aggregates information about the entire sequence. This vector acts as the summary of both sentence A and sentence B.
- This summary vector (**C**) is then used by a binary classifier to predict whether sentence B is the true continuation of sentence A or if it's a random sentence.

Why is this useful?

During BERT's pre-training, NSP teaches the model not only to understand individual words and sentences but also to grasp relationships between sentences. This pre-training task helps BERT build a better contextual understanding, which improves its performance on downstream tasks like question answering, summarization, or sentence-pair classification.

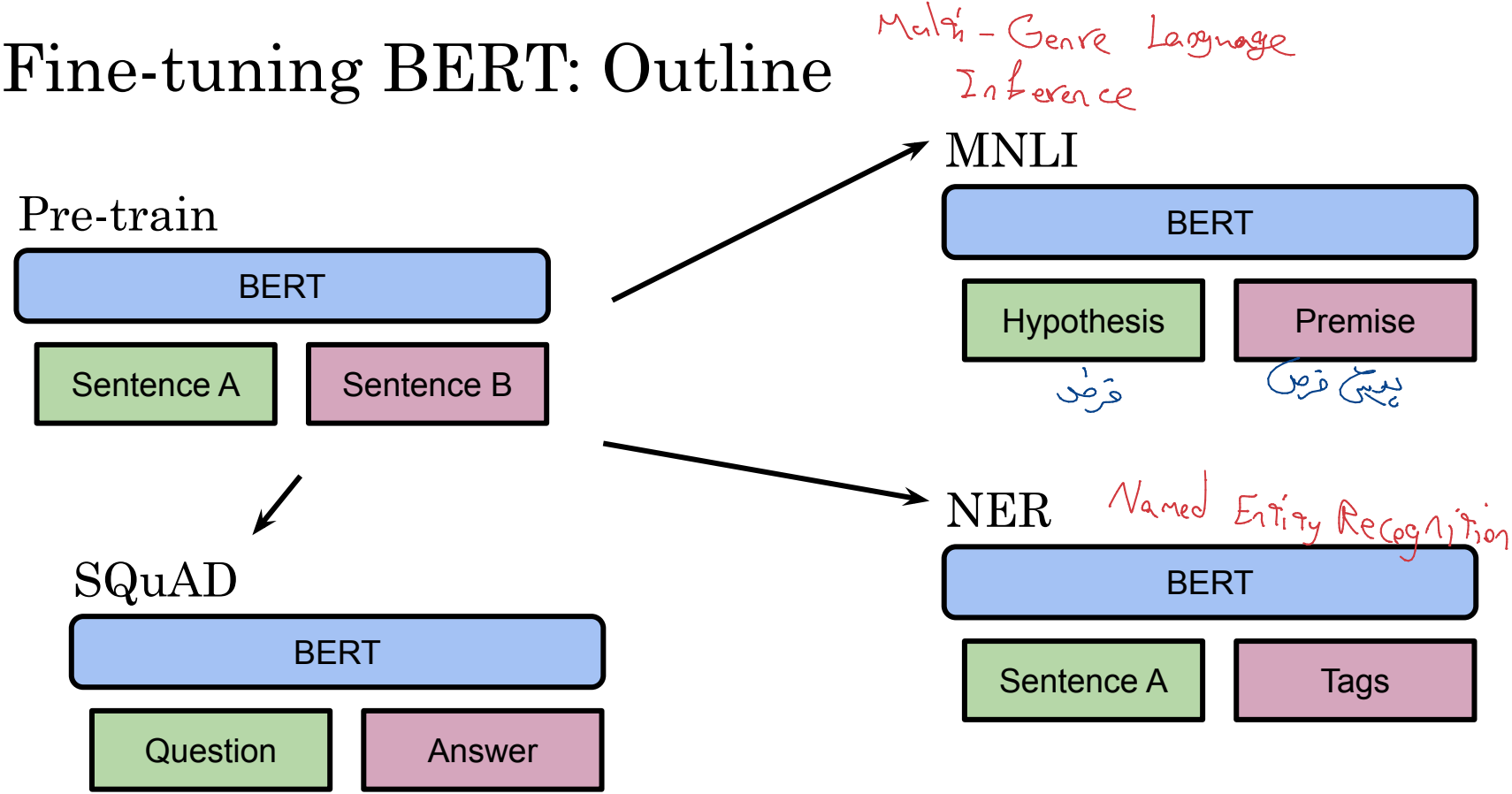
Summary:

- **[CLS] token output (C)**: Contains a pooled representation of both sentence A and sentence B.
- **NSP task**: Uses this `[CLS]` output to determine whether sentence B follows sentence A.
- **Classifier**: A small network (usually just a feedforward layer) predicts whether the sentence pair is coherent or randomly paired.

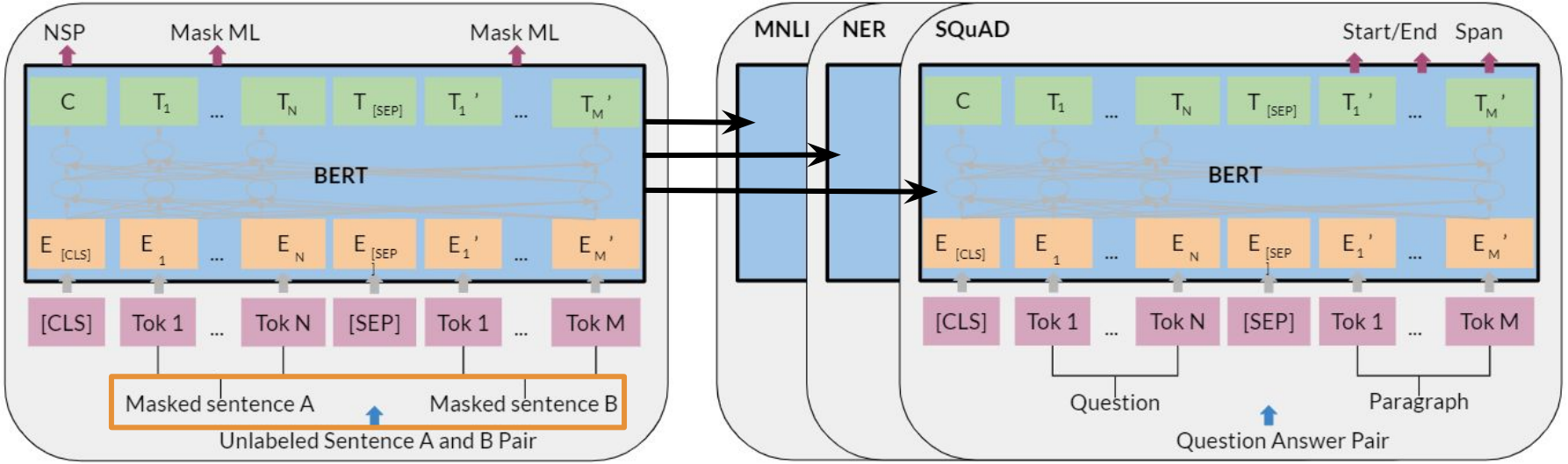
deeplearning.ai

Fine-tuning BERT

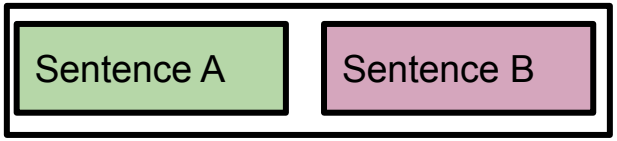
Fine-tuning BERT: Outline

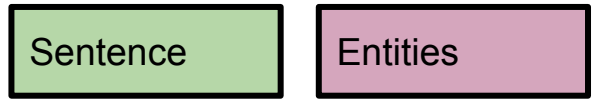


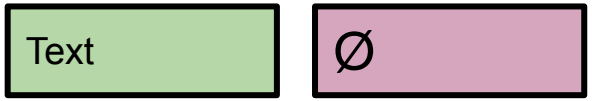
Inputs

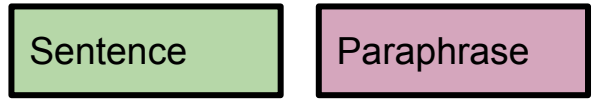

NSP
Mask ML
Mask ML
C
T1
...
TN
T [SEP]
T1'
...
TM'
BERT
E [CLS]
E 1
...
E N
E [SEP]
E1'
...
EM'
[CLS]
Tok 1
...
Tok N
[SEP]
Tok 1
...
Tok M
Masked sentence A
Masked sentence B
Unlabeled Sentence A and B Pair
MNLI
NER
SQuAD
Start/End
Span
Question
Paragraph
Question Answer Pair

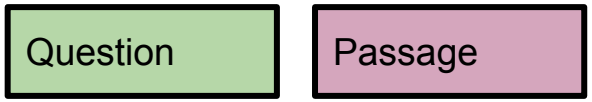
Summary

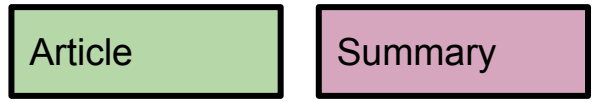

Sentence A
Sentence B

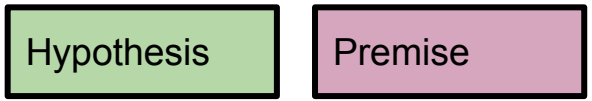

Sentence
Entities


Text
Ø

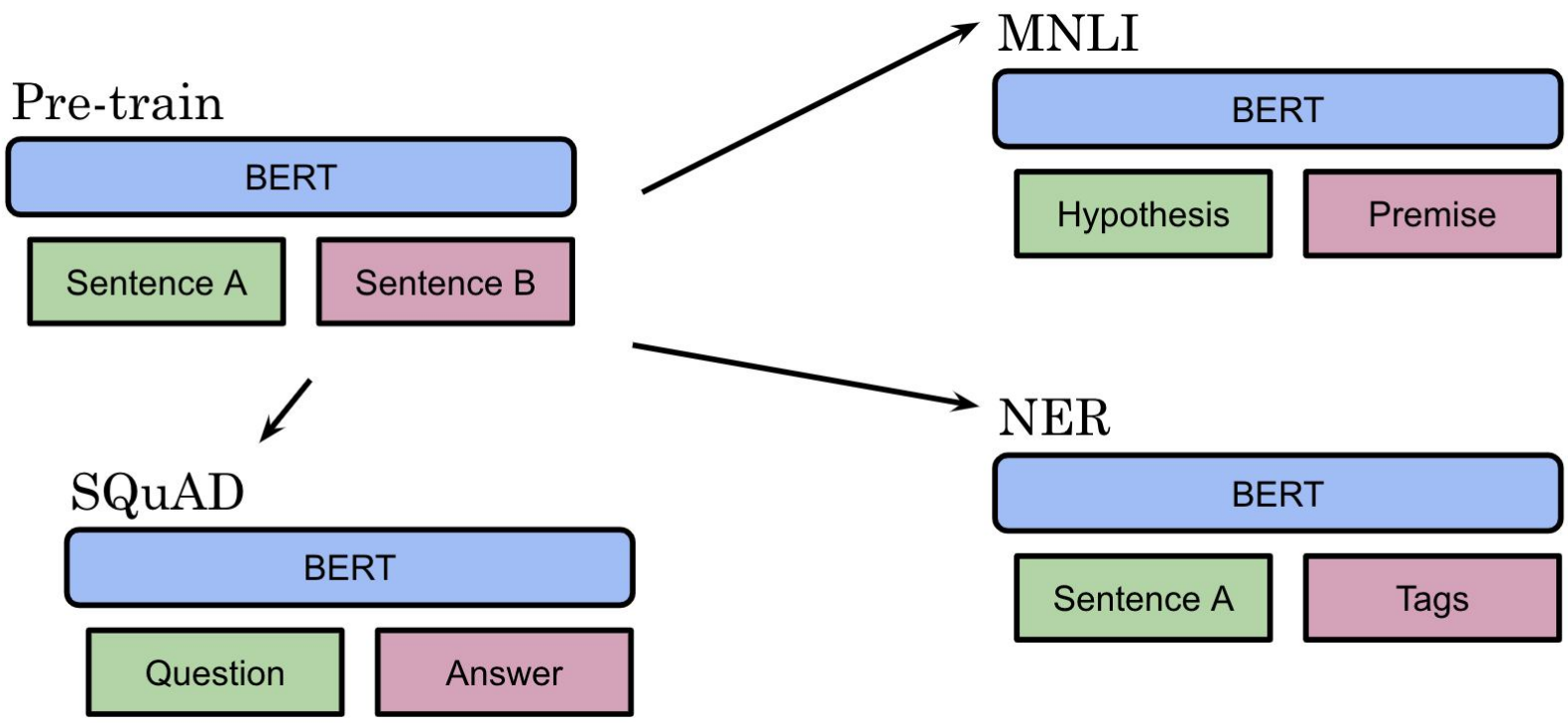

Sentence
Paraphrase


Question
Passage

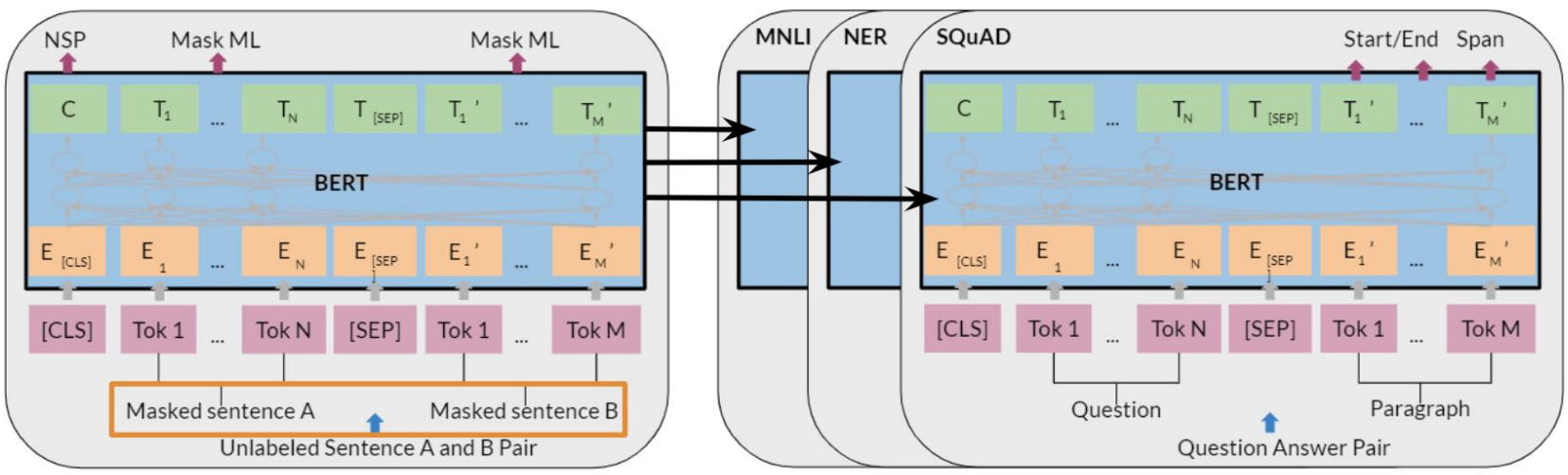

Article
Summary


Hypothesis
Premise

Once you have a pre-trained model, you can fine tune it on different tasks.



For example, given a hypothesis, you can identify the premise. Given a question, you can find the answer. You can also use it for named entity recognition. Here is a summary of the inputs.



- You can replace sentences A/B
- Paraphrase from sentence A
- Question/passage
- Hypothesis premise pairs in entailment
- Text and a Ø for classification/sequence tagging
- Output tokens are fed into a layer for token level tasks otherwise use [CLS] embedding as input.

In the context of the **Multi-Genre Natural Language Inference (MNLI)** task, the terms "hypothesis" and "premise" are used to describe the two sentences involved in the inference task:

1. **Premise**:
    - This is the first sentence or statement.
    - It represents a fact or observation. The premise provides the context or background that the model needs to understand the situation.
2. **Hypothesis**:
    - This is the second sentence or statement.
    - It is a statement whose truth is in question. The hypothesis is compared against the premise to determine whether it logically follows, contradicts, or is neutral with respect to the premise.

Task Overview:

In MNLI, the model is given a **premise** and a **hypothesis**, and it needs to predict the relationship between them. There are three possible labels:

- **Entailment**: The hypothesis logically follows from the premise. (The hypothesis is true if the premise is true.)
- **Contradiction**: The hypothesis contradicts the premise. (The hypothesis cannot be true if the premise is true.)
- **Neutral**: The hypothesis is neither entailed nor contradicted by the premise. (There's no clear relationship.)

Example:

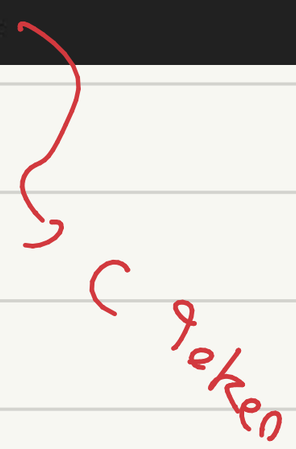
Premise: "A man is riding a bicycle down a busy street."

Hypothesis:

- Entailment: "Someone is riding a bicycle."
- Contradiction: "There is no one riding a bicycle."
- Neutral: "The man is wearing a helmet."

MNLI in BERT:

When fine-tuning BERT for MNLI, the premise and hypothesis are concatenated into a single sequence with special tokens like `[CLS]` and `[SEP]`. BERT then predicts the relationship between them based on the pooled output of the `[CLS]` token.



deeplearning.ai

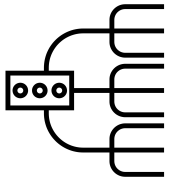
Transformer T5

Outline

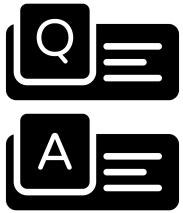
- Understand how T5 works
- Recognize the different types of attention used
- Overview of model architecture

Transformer - T5 Model

Text to Text



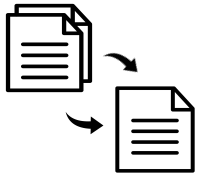
Classification



Question Answering (Q&A)

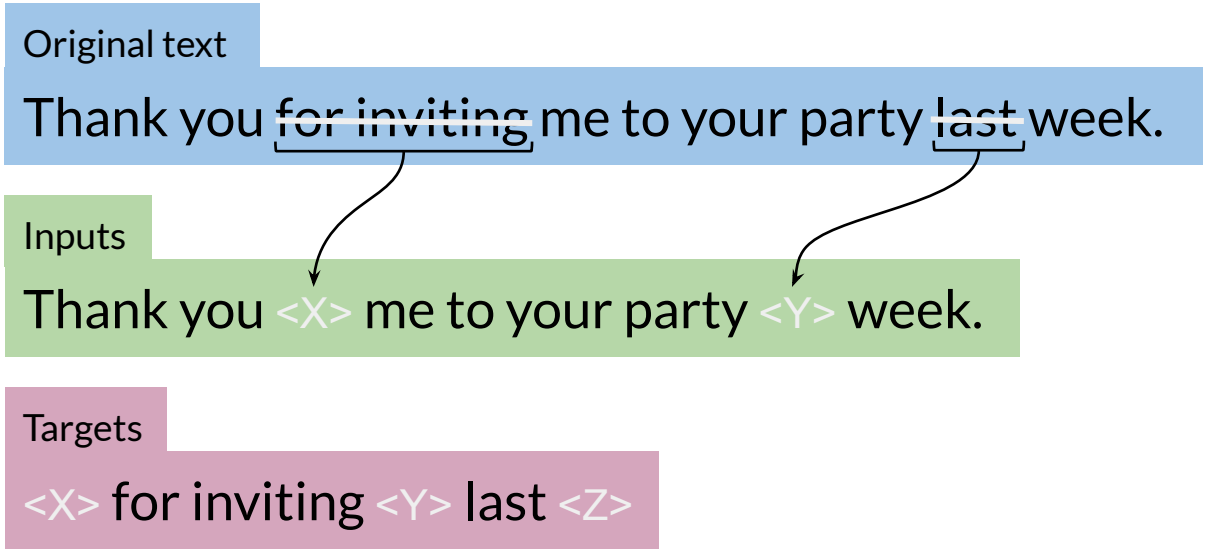
Machine Translation

Summarization

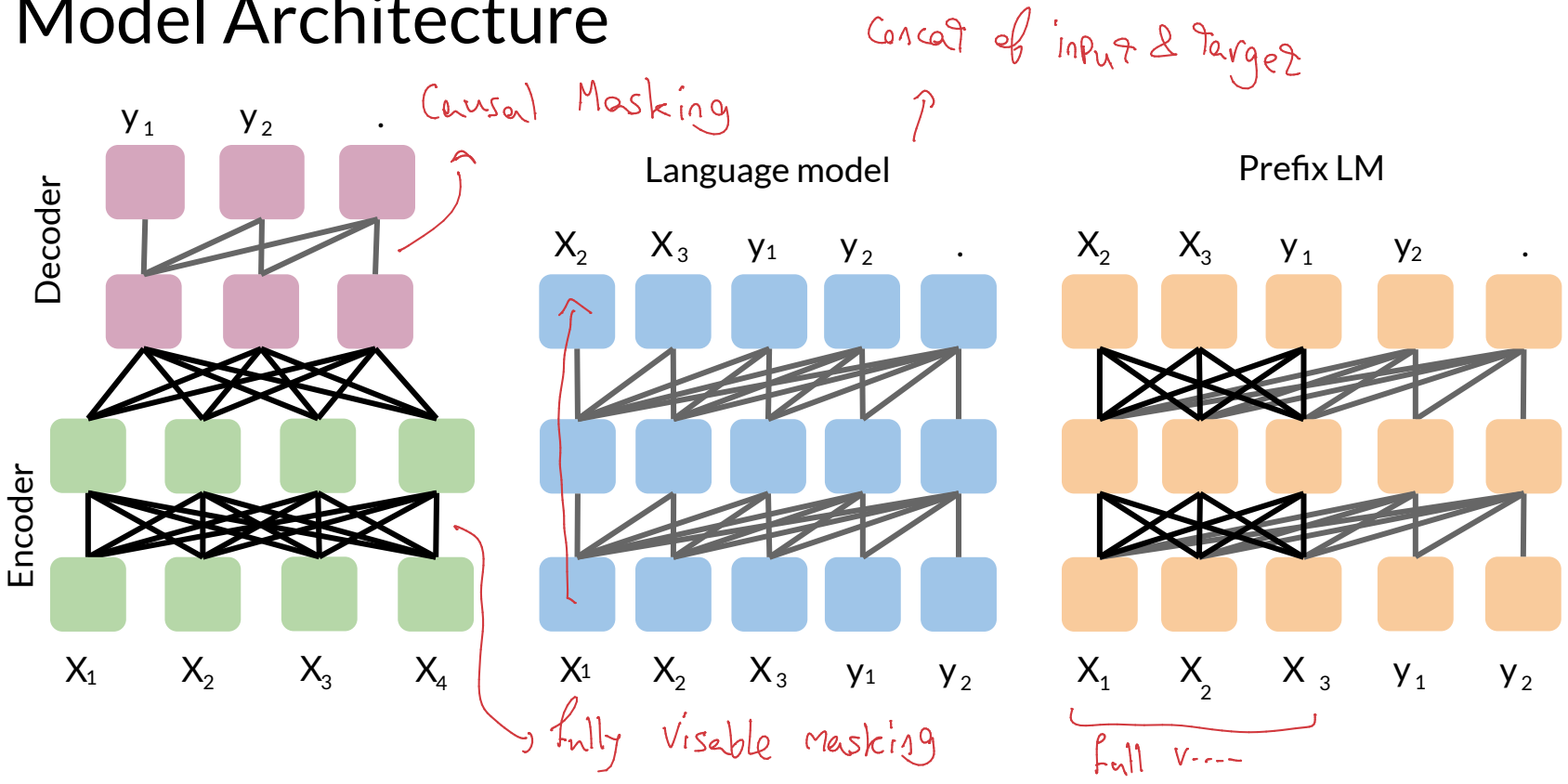


Sentiment

Transformer - T5 Model - Pre Training



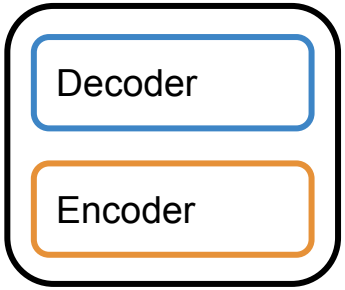
Model Architecture



Causal M... for rest

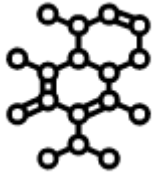
Model Architecture

- Encoder/decoder



- 12 transformer blocks each

- 220 million parameters

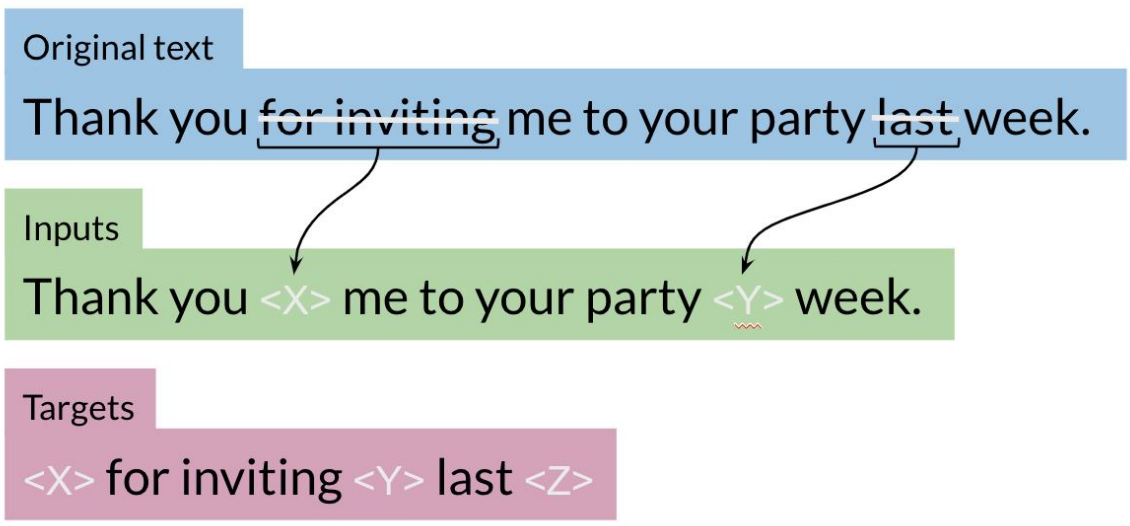


Summary

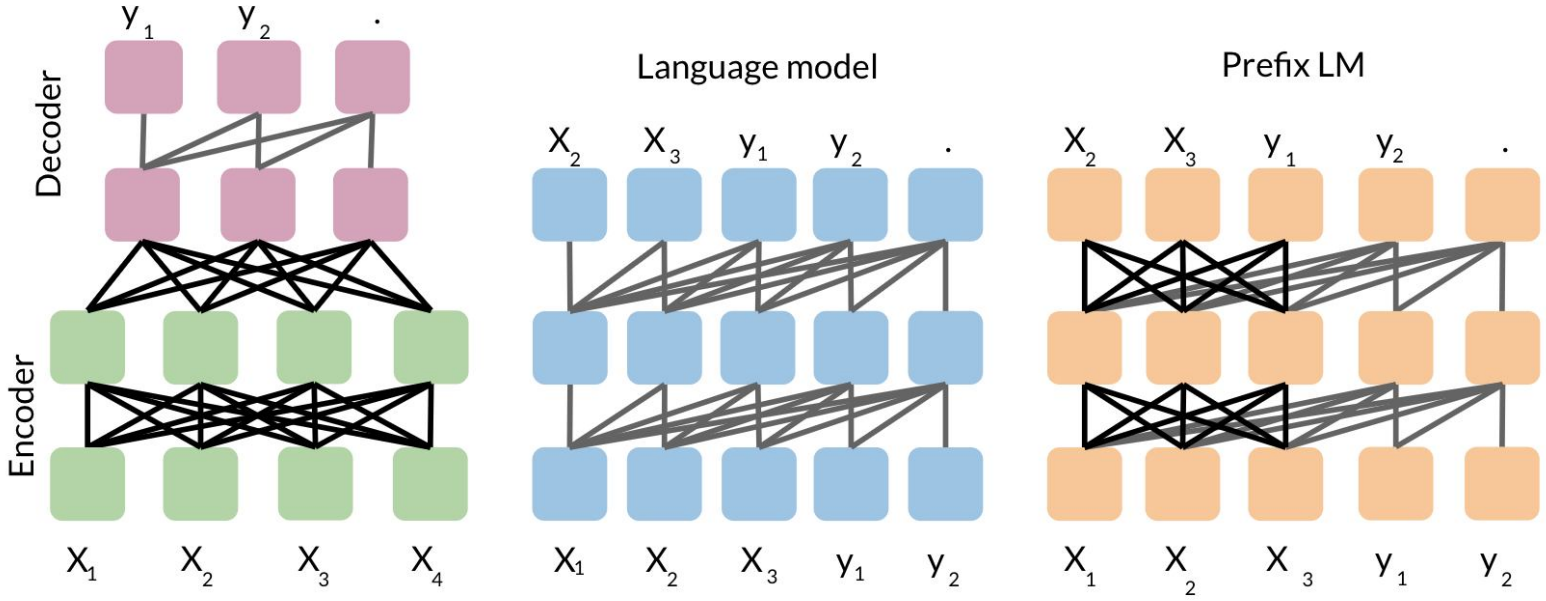
- Prefix LM attention
- Model architecture
- Pre-training T5 (MLM)

↳ Masked Lang Model

One of the major techniques that allowed the T5 model to reach state of the art is the concept of masking:



For example, you represent the “for inviting” with <X> and last with <Y> then the model predicts what the X should be and what the Y should be. This is exactly what we saw in the BERT loss. You can also mask out a few positions, not just one. The loss is only on the mask for BERT, for T5 it is on the target.



So we start with the basic encoder-decoder representation. There you have a fully visible attention in the encoder and then causal attention in the decoder. So light gray lines correspond to causal masking. And dark gray lines correspond to the fully visible masking.

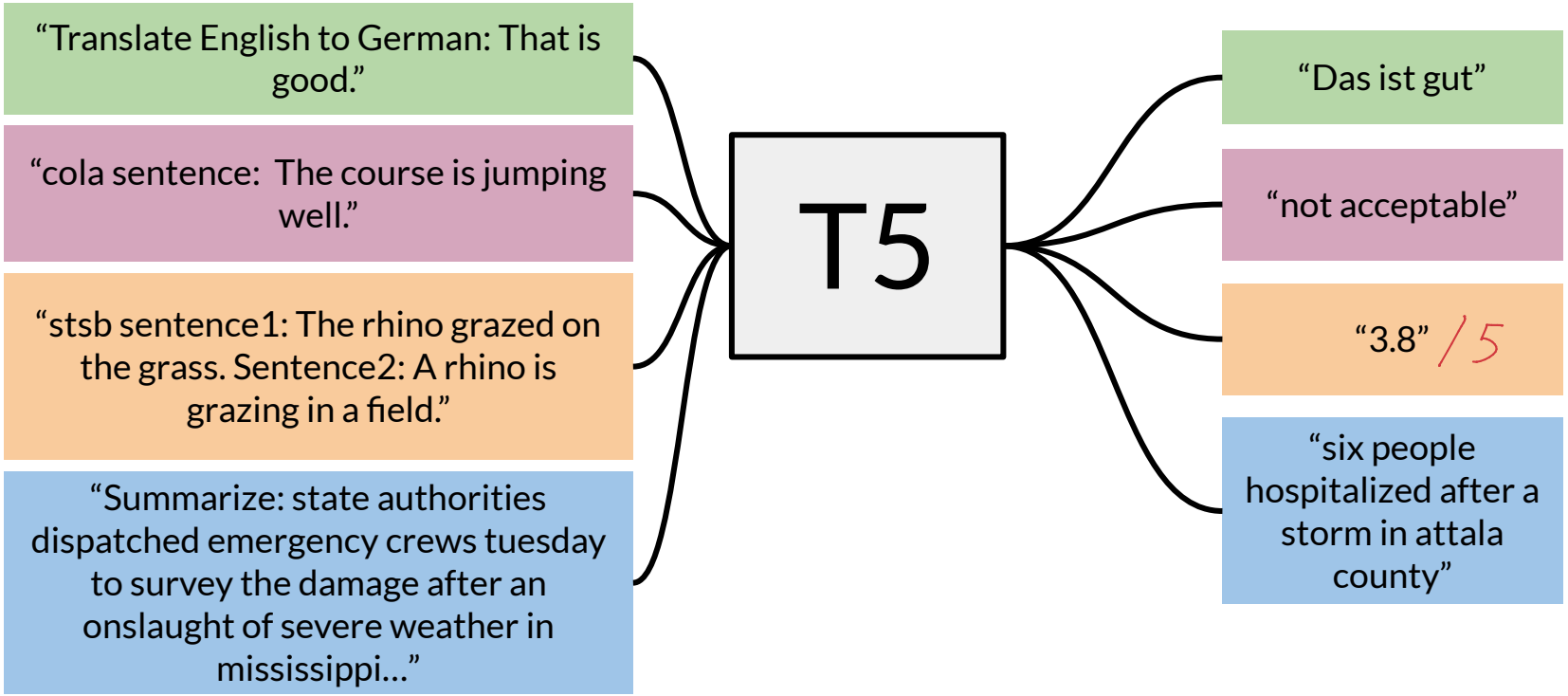
In the middle we have the language model which consists of a single transformer layer stack. And it's being fed the concatenation of the inputs and the target. So it uses causal masking throughout as you can see because they're all gray lines. And you have X1 going inside, you get X2, X2 goes into the model and you get X3 and so forth.

To the right, we have prefix language model which corresponds to allowing fully visible masking over the inputs as you can see with the dark arrows. And then causal masking in the rest.

deeplearning.ai

Multi-task Training Strategy

Multi-task training strategy



©Exploring the Limits of Transfer learning with a unified text to Text Transformer. Raffel et. al. 2020

Input and Output Format

Relations in NLI : 1) entailment
2) Contradiction 3) Neutral

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

↳ Predict they as (1)

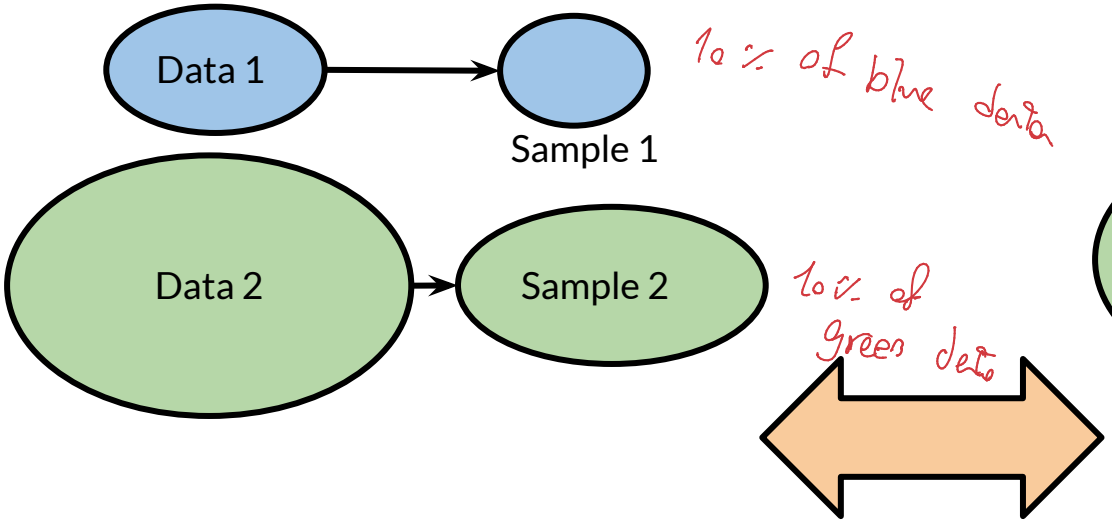
Multi-task Training Strategy

Fine-tuning method	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
* All parameters	**83.28**	**19.24**	**80.88**	**71.36**	**26.98**	**39.82**	**27.65**
Adapter layers, $d = 32$	80.52	15.08	79.32	60.40	13.84	17.88	15.54
Adapter layers, $d = 128$	81.51	16.62	79.47	63.03	19.83	27.50	22.63
Adapter layers, $d = 512$	81.54	17.78	79.18	64.30	23.45	33.98	25.81
Adapter layers, $d = 2048$	81.51	16.62	79.47	63.03	19.83	27.50	22.63
Gradual unfreezing	82.50	18.95	79.17	**70.79**	26.71	39.02	26.93

How much data from each task to train on?

Data Training Strategies

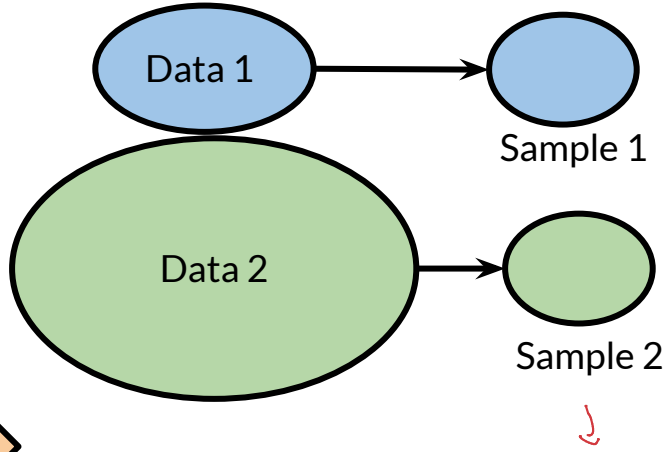
Examples-proportional mixing



10 % of blue data

10 % of green data

Equal mixing

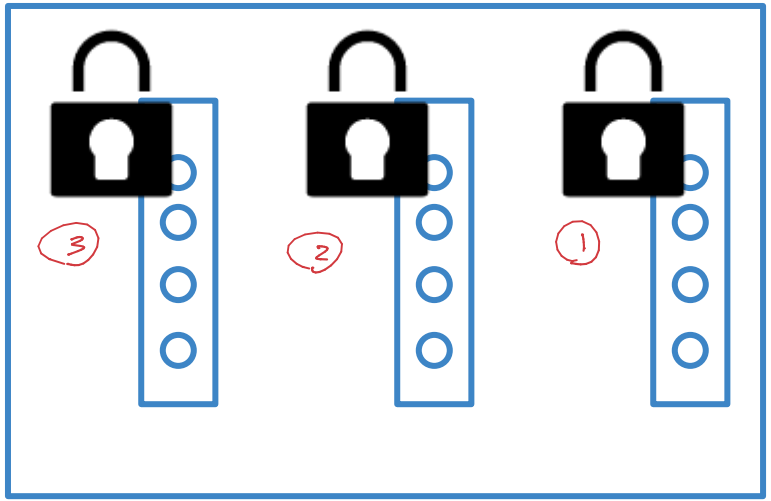


Regardless of size of each data, we take an equal sample

Temperature-scaled mixing

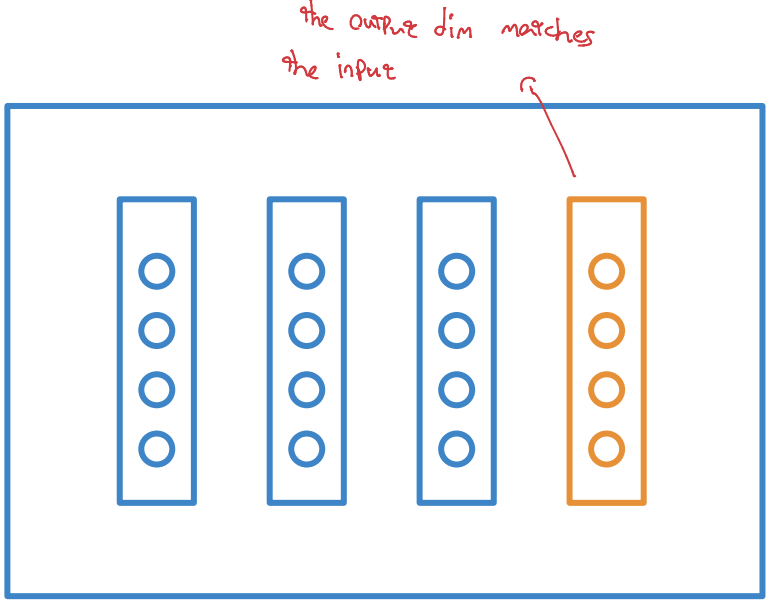
Play with Params α to get sth in between

Gradual unfreezing vs. Adapter layers



Unfreez ① then ② then ③

Gradual unfreezing



Adapter layers

Adapter layers are a technique used in transfer learning, particularly when fine-tuning pre-trained models. They add lightweight, trainable layers between the existing layers of a large pre-trained model.

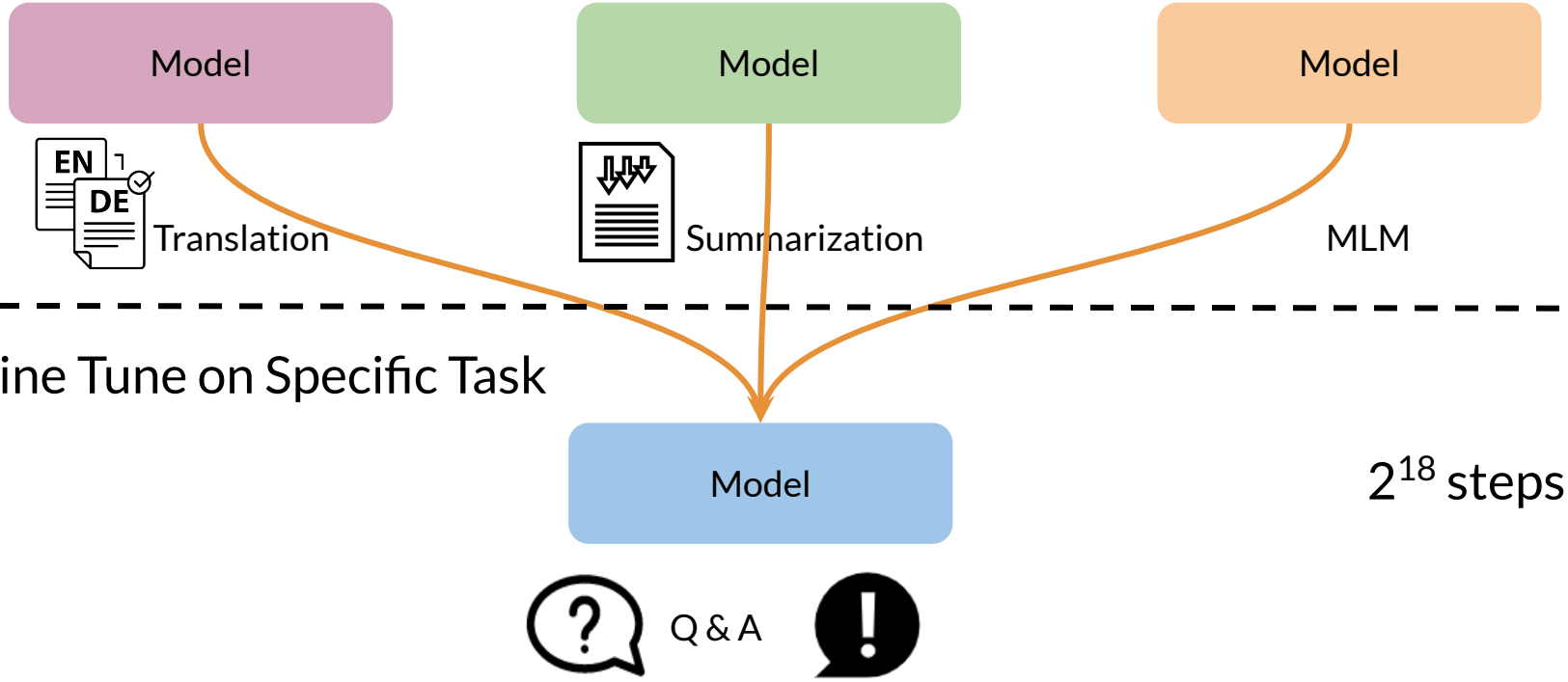
Here's how they work:

- Instead of updating all the parameters of a large pre-trained model during fine-tuning, only the small adapter layers are trained. The rest of the model remains frozen.
- This reduces the computational cost of fine-tuning because only a small portion of the model is being updated.
- The adapter layers act as intermediaries that adjust the model's knowledge to fit the new task.

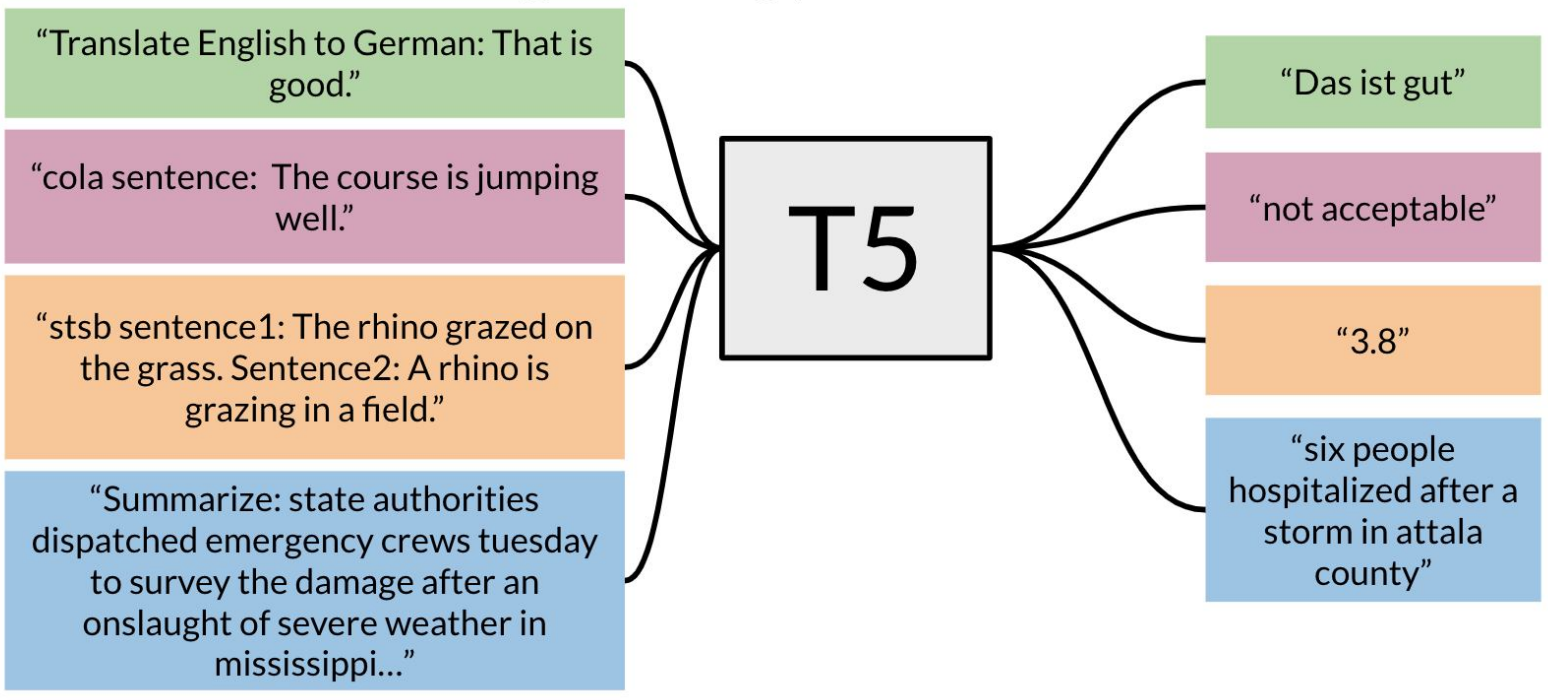
In contrast to **gradual unfreezing**, where layers are progressively unfrozen and fine-tuned, adapter layers allow the core model to remain mostly intact, while still adapting the model to the new task through the smaller trainable layers. This can be particularly useful in scenarios where computational resources are limited or where the pre-trained model's weights should largely remain unchanged.

Fine-tuning

Pre Training



This is a reminder of how the T5 model works:



You can see that you only have to add a small prefix to the input and the model as a result will solve the task for you. There are many tasks that the t5 model can do for you.

It is possible to formulate most NLP tasks in a "text-to-text" format – that is, a task where the model is fed some text for context or conditioning and is then asked to produce some output text. This framework provides a consistent training objective both for pre-training and fine-tuning. Specifically, the model is trained with a maximum likelihood objective (using "teacher forcing") regardless of the task.

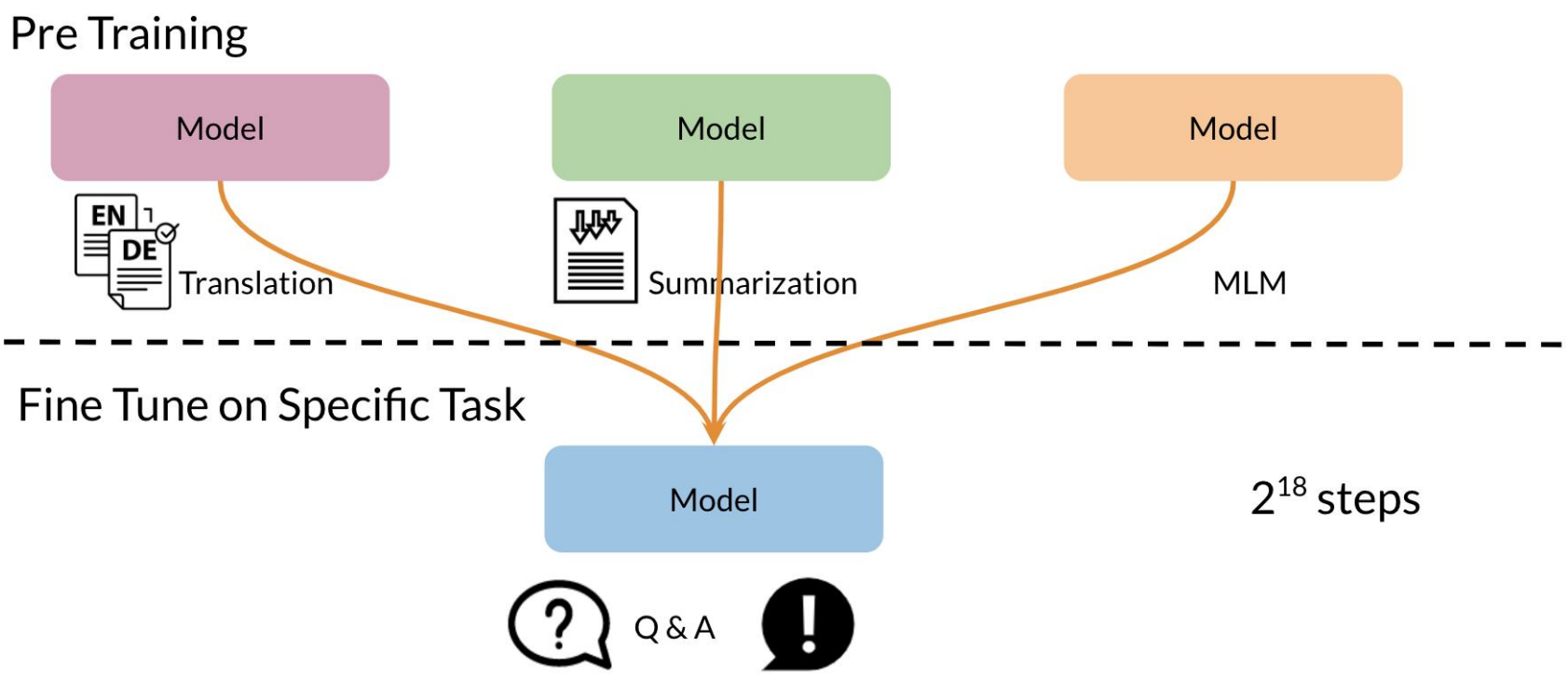
Training data strategies

Examples-proportional mixing: sample in proportion to the size of each task's dataset

Temperature scaled mixing: adjust the "temperature" of the mixing rates. This temperature parameter allows you to weight certain examples more than others. To implement temperature scaling with temperature T, we raise each task's mixing rate rm to the power of 1⁄T and renormalize the rates so that they sum to 1. When T = 1, this approach is equivalent to examples-proportional mixing and as T increases the proportions become closer to equal mixing

Equal mixing: In this case, you sample examples from each task with equal probability. Specifically, each example in each batch is sampled uniformly at random from one of the datasets you train on.

Fine tuning example



You can see above how fine tuning on a specific task could work even though you were pre-training on different tasks.

deeplearning.ai

GLUE Benchmark

General Language Understanding Evaluation

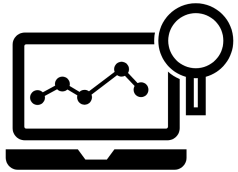
- A collection used to train, evaluate, analyze natural language understanding systems
- Datasets with different genres, and of different sizes and difficulties
- Leaderboard how well their models perform compare to others

Tasks Evaluated on

- Sentence grammatical or not?
- Sentiment
- Paraphrase
- Similarity
- Questions duplicates
- Answerable
- Contradiction
- Entailment
- Winograd (co-ref) → a Pronoun refer to a certain noun or another noun

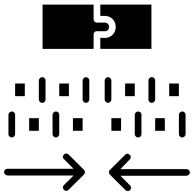
General Language Understanding Evaluation

- Drive research

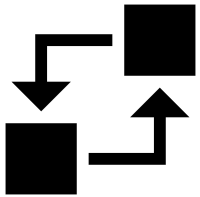


- Model agnostic

= independent from model

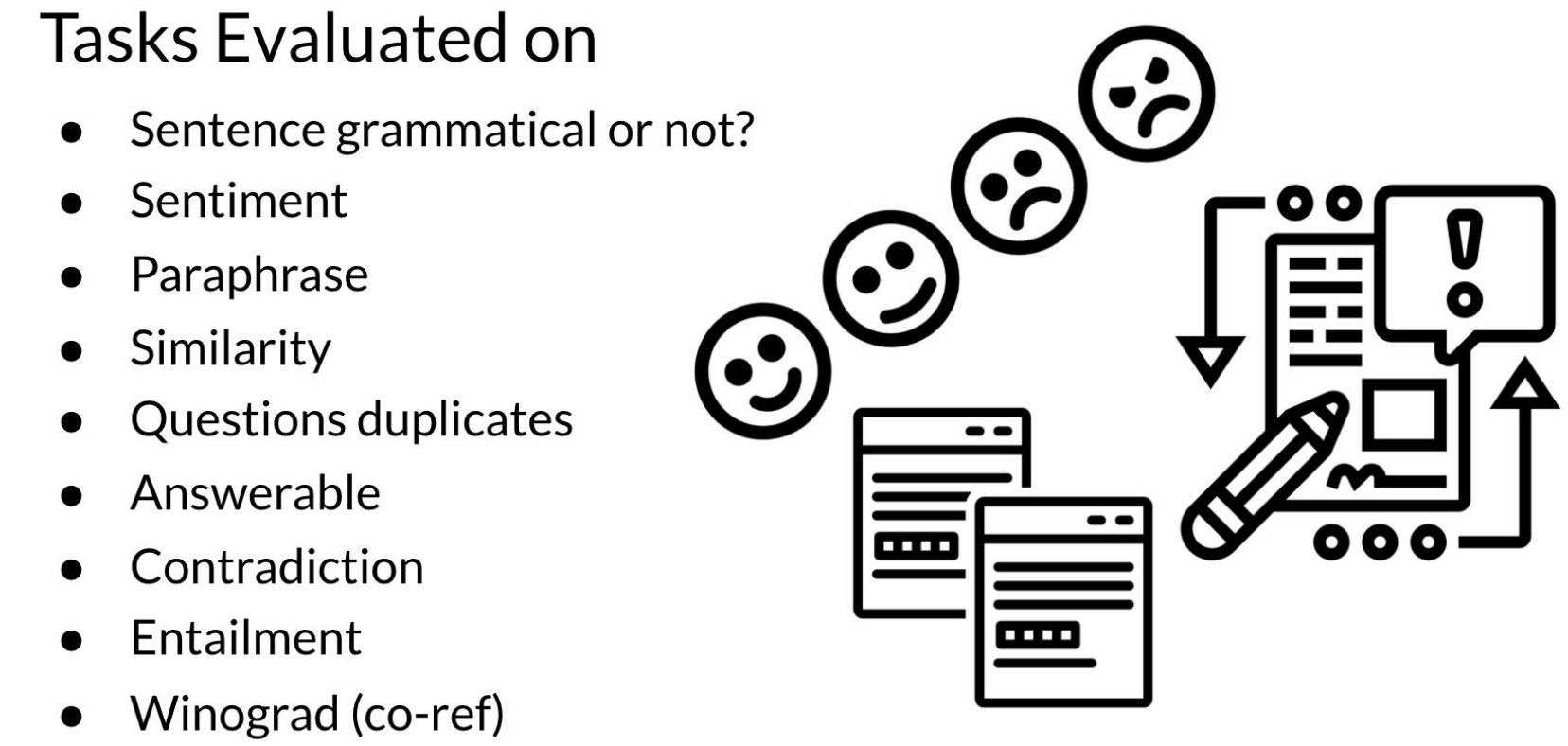


- Makes use of transfer learning



General Language Understanding Evaluation (GLUE) is contains:

- A collection used to train, evaluate, analyze natural language understanding systems
- Datasets with different genres, and of different sizes and difficulties
- Leaderboard

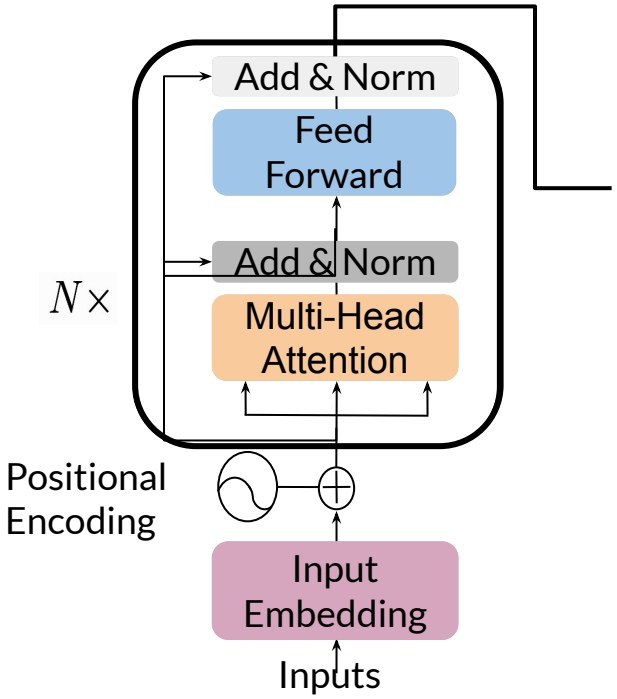


Currently T5 is state of the art according to this GLUE benchmark and you will be implementing it for homework this week! This GLUE bench mark is used for research purposes, it is model agnostic, and relies on models that make use of transfer learning.

deeplearning.ai

Question Answering

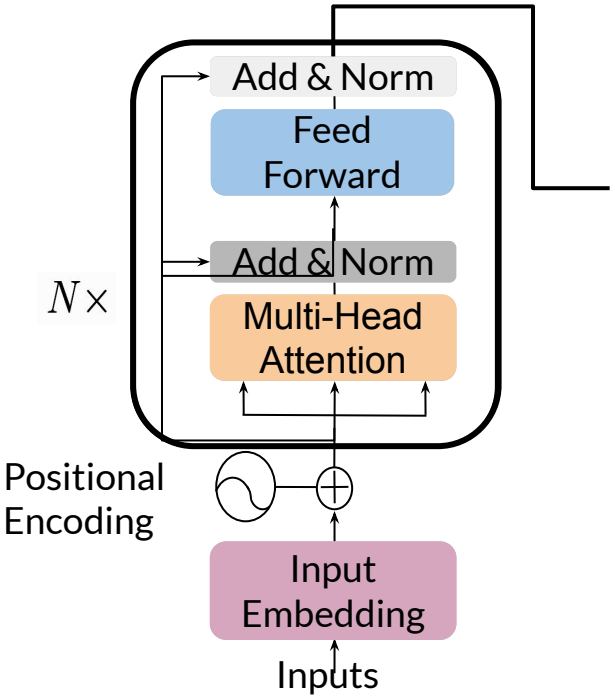
Transformer encoder



Feedforward:

```
[
      LayerNorm,
      dense,
      activation,
      dropout_middle,
      dense,
      dropout_final
 ]
```

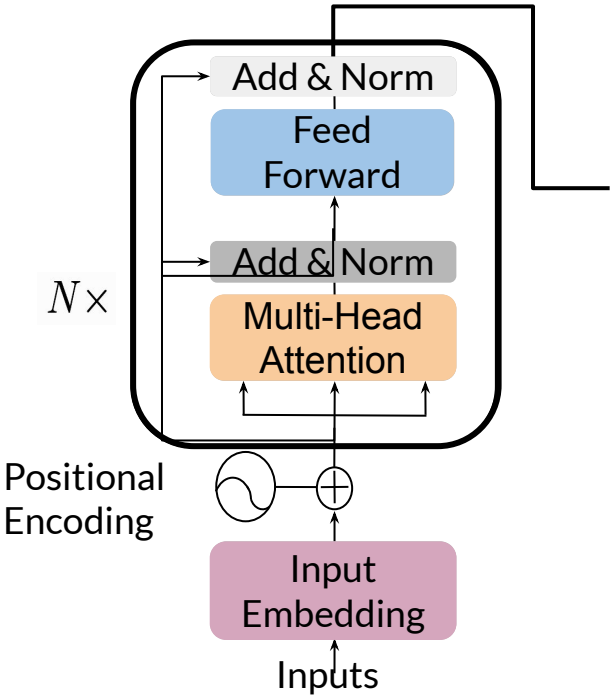
Transformer encoder



Encoder block:

```
[
      Residual(
          LayerNorm,
          attention,
          dropout_,
      ),
      Residual(
          feed_forward,
      ),
 ]
```

Transformer encoder



Feedforward:

```
[
      LayerNorm,
      dense,
      activation,
      dropout_middle,
      dense,
      dropout_final
 ]
```

Encoder block:

```
[
      Residual(
          LayerNorm,
          attention,
          dropout_,
      ),
      Residual(
          feed_forward,
      )
 ]
```

Data examples

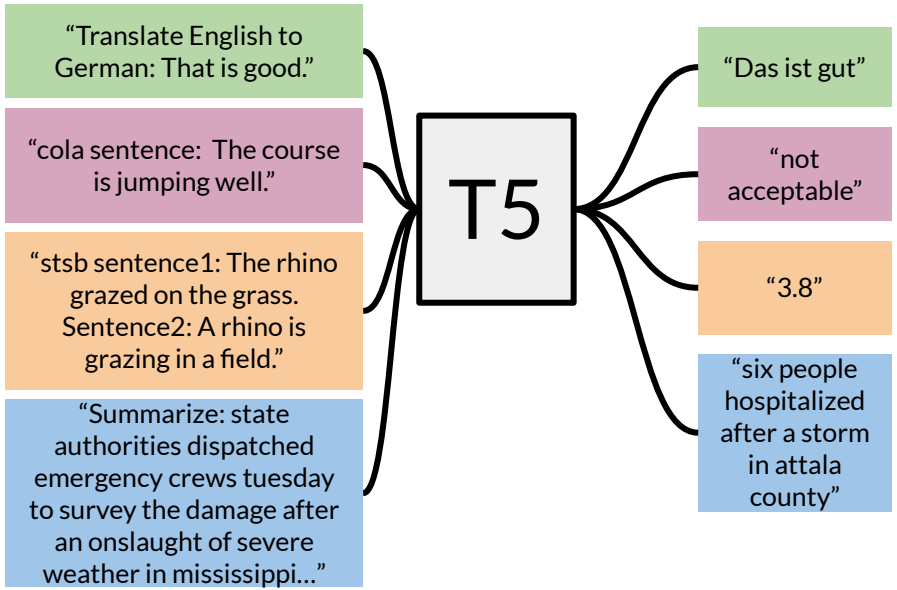
Question: What percentage of the French population today is non - European ?

Context: Since the end of the Second World War , France has become an ethnically diverse country . Today , **approximately five percent** of the French population is non - European and non - white . This does not approach the number of non - white citizens in the United States (roughly 28 – 37 % , depending on how Latinos are classified ; see Demographics of the United States) . Nevertheless , it amounts to at least three million people , and has forced the issues of ethnic diversity onto the French policy agenda . France has developed an approach to dealing with ethnic problems that stands in contrast to that of many advanced , industrialized countries . Unlike the United States , Britain , or even the Netherlands , France maintains a " color - blind " model of public policy . This means that it targets virtually no policies directly at racial or ethnic groups . Instead , it uses geographic or class criteria to address issues of social inequalities . It has , however , developed an extensive anti - racist policy repertoire since the early 1970s . Until recently , French policies focused primarily on issues of hate speech — going much further than their American counterparts — and relatively less on issues of discrimination in jobs , housing , and in provision of goods and services .

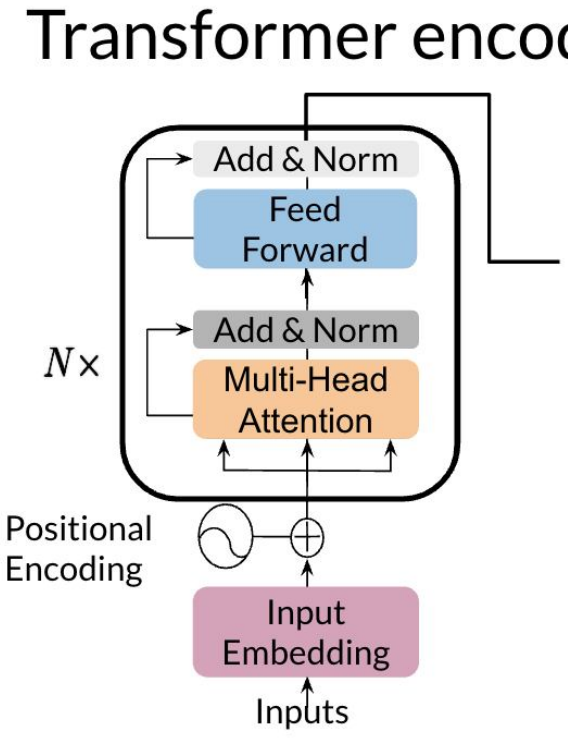
Target: Approximately five percent

Implementing Q&A with T5

- Load a pre-trained model
- Process data to get the required inputs and outputs: "question: Q context: C" as input and "A" as target
- Fine tune your model on the new task and input
- Predict using your own model



You will be implementing an encoder this week. Last week you implemented the decoder. So here it is:



Feedforward:

```
[
      LayerNorm,
      dense,
      activation,
      dropout_middle,
      dense,
      dropout_final
 ]
```

Encoder block:

```
[
      Residual(
          LayerNorm,
          attention,
          dropout_,
      ),
      Residual(
          feed_forward,
      )
 ]
```

You can see there is a feedforward and the encoder-block above. It makes use of two residual connections, layer normalization, and dropout.

The steps you will follow to implement it are:

- Load a pre-trained model
- Process data to get the required inputs and outputs: "question: Q context: C" as input and "A" as target
- Fine tune your model on the new task and input
- Predict using your own model

deeplearning.ai

Hugging Face: Introduction

Outline

- What is Hugging Face?
- How you can use the Hugging Face ecosystem

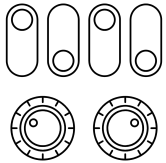
Hugging Face

Transformers library

Use it with

Use it for

Applying state of the art transformer models

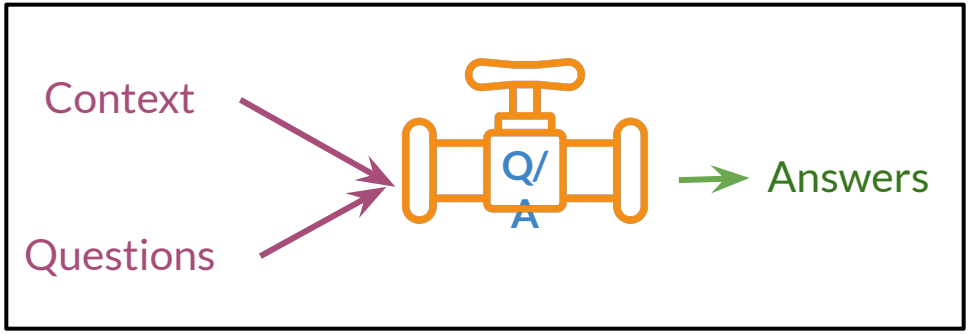


Fine-tuning pretrained transformer models

Hugging Face: Using Transformers

Pipelines

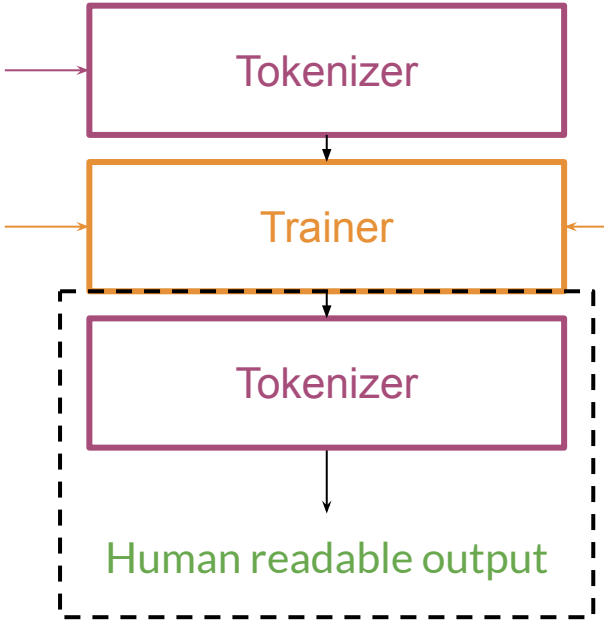
1. **Pre-processing** your inputs
2. **Running** the model
3. **Post-processing** the outputs



Hugging Face: Fine-Tuning Transformers

Datasets:
One Thousand

Model Checkpoints:
More than 14 thousand



Evaluation metrics

Checkpoint: Set of learned parameters for a model using a training procedure for some task

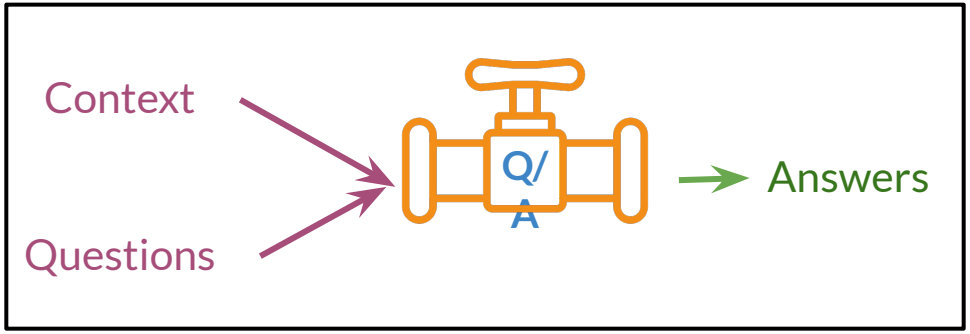
deeplearning.ai

Hugging Face: Using Transformers

Using Transformers

Pipelines

1. **Pre-processing** your inputs
2. **Running** the model
3. **Post-processing** the outputs



Tasks

obj: you build & for specific NLP tasks

Pipelines

Initialization → Task

Initialization → Model Checkpoint (optional)

Use → Inputs for the task

Sentiment Analysis	Question Answering	Fill-Mask
Sequence	Context and questions	Sentence and position

Output: Pos/Neg

Checkpoints

Huge number of model checkpoints that you can use in your pipelines.

But **beware**, not every checkpoint would be suitable for your task.

Model Hub

Hub containing models that you can use in your pipelines according to the task you need: https://huggingface.co/models

Model Card shows a description of your selected model and useful information such as code snippet examples.

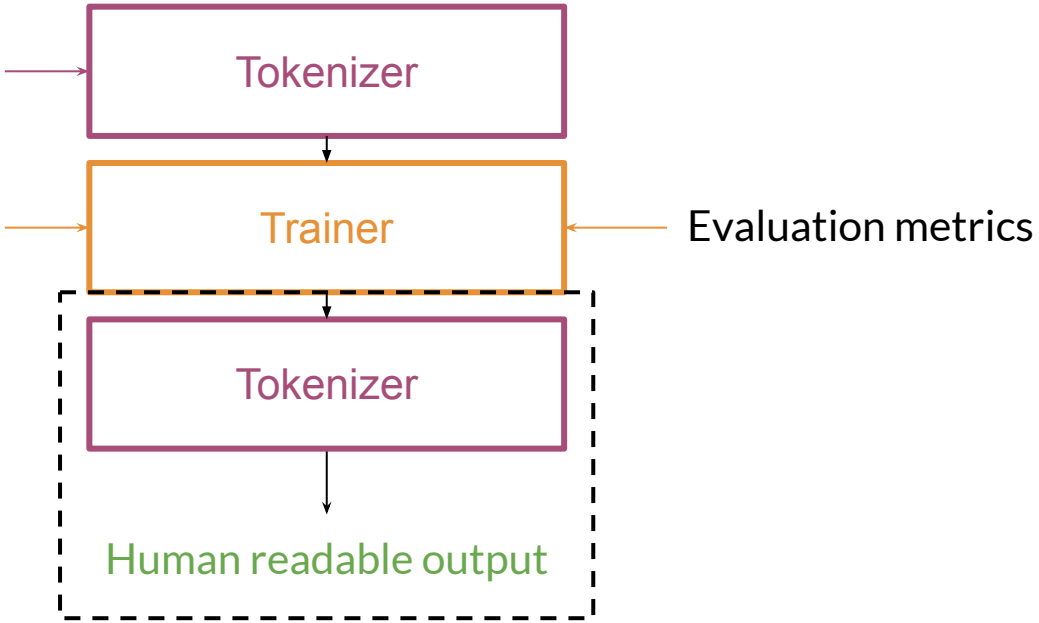
deeplearning.ai

Hugging Face: Fine-Tuning Transformers

Fine-Tuning Tools

Datasets:
One Thousand

Model Checkpoints:
More than 14 thousand



Model Checkpoints

Model Checkpoints:
More than 15 thousand
(and increasing)

Upload the architecture and weights with 1 line of code!

Model	Dataset	Name in 🤗
DistilBERT	Stanford Question Answering Dataset (SQuAD)	distilbert-base-cased-distilled-squad
BERT	Wikipedia and Book Corpus	bert-base-cased
...	...	...

Datasets

Datasets:
One Thousand

Load them using just one function

Optimized to work with massive amounts of data!

Tokenizers

"What well-known superheroes were introduced between 1939 and 1941 by Detective Comics?"

[101, 1327, 1218, 118, 1227, 18365, 1279, 1127, 2234, 1206, 3061, 1105, 3018, 1118, 9187, 7452, 136, 102]

Depending on the use case, you might need to run additional steps.

Trainer and Evaluation Metrics

Trainer object let's you define the training procedure

- Number of epochs
- Warm-up steps
- Weight decay
- ...

Train using one line of code!

Pre-defined evaluation metrics, like BLEU and ROUGE