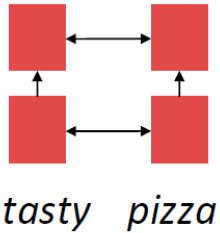
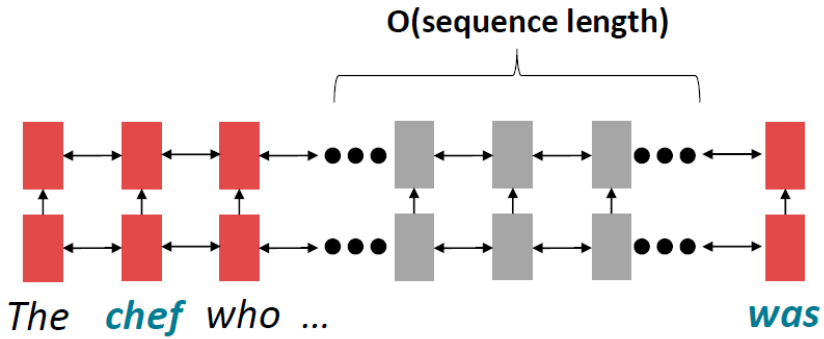
RNN –Issues

Issues with recurrent models: **Linear interaction distance**

- RNNs are unrolled "left-to-right".
- This encodes linear locality: a useful heuristic!
  - Nearby words often affect each other's meanings



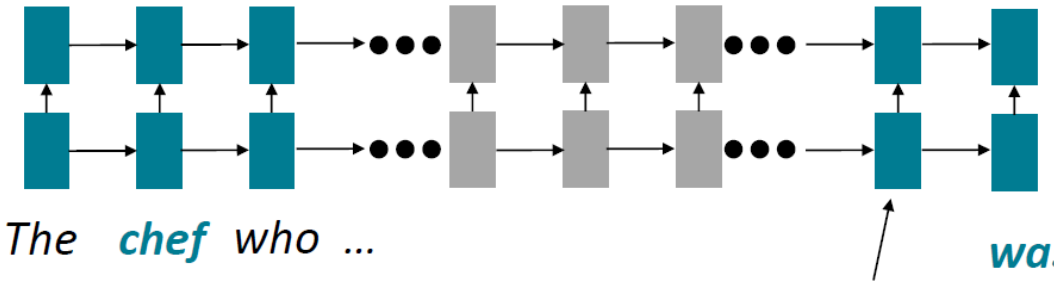
- **Problem:** RNNs take **O(sequence length)** steps for distant word pairs to interact.



https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1214/slides/cs224n-2021-lecture09-transformers.pdf

Issues with recurrent models: **Linear interaction distance**

- **O(sequence length)** steps for distant word pairs to interact means:
  - Hard to learn long-distance dependencies (because gradient problems!)
  - Linear order of words is "baked in"; we already know linear order isn't the right way to think about sentences…

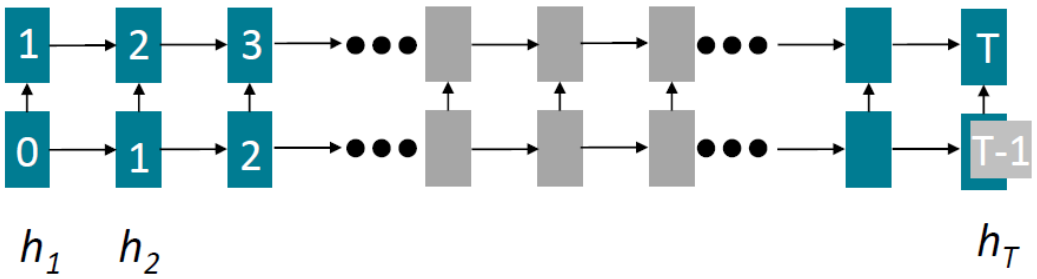


Info of ***chef*** has gone through O(sequence length) many layers!

https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1214/slides/cs224n-2021-lecture09-transformers.pdf

Issues with recurrent models: **Lack of parallelizability**

- Forward and backward passes have **O(sequence length)** unparallelizable operations
  - GPUs can perform a bunch of independent computations at once!
  - But future RNN hidden states can't be computed in full before past RNN hidden states have been computed
  - Inhibits training on very large datasets!



Numbers indicate min # of steps before a state can be computed

https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1214/slides/cs224n-2021-lecture09-transformers.pdf

Issues with Recurrent Attention

- Scalability issues
  - Performance degrades as the distance between words increases
- Parallelization limitations
  - Recurrent processes lacks ability to be parallelized
- Memory constraints
  - RNNs have limited memory and struggle with long-range dependencies
  - Diluted impact of earlier elements on output as sequence progresses

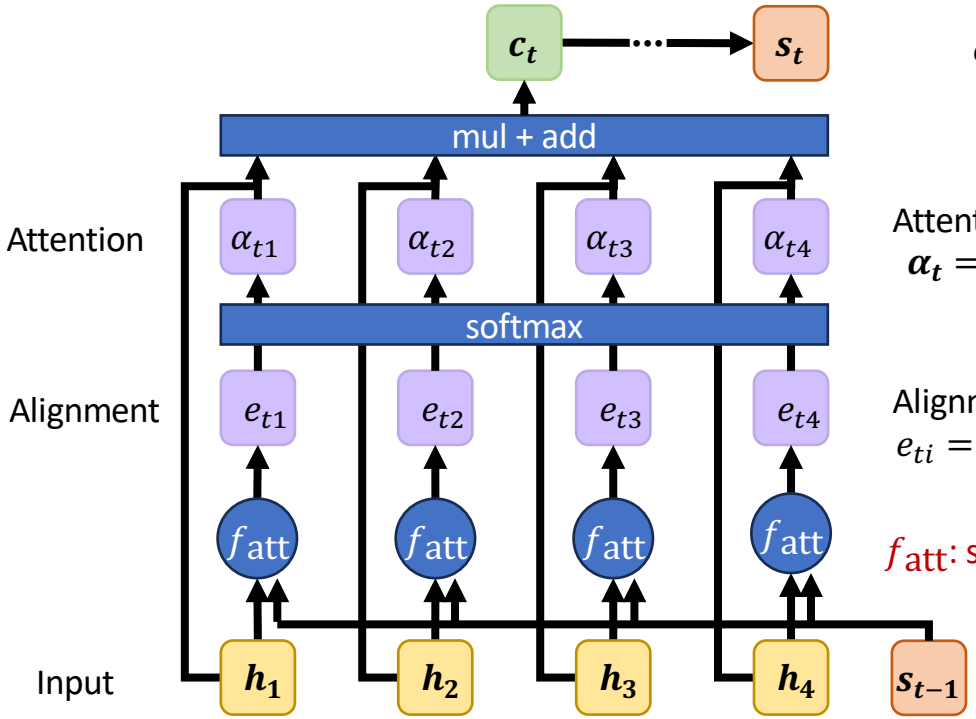
- **Potential solution: decouple attention from RNNs**
  - **How? Separate the attention mechanism into smaller, self-contained components**

Slide Credit : https://cs.uwaterloo.ca/~wenhuche/teaching/cs886/

Self -ATTENTION

Attention we've seen so far

Now known as **"additive" recurrent attention** (type of encoder-decoder attention)



Context vector

$$c_t = \sum_i \alpha_{ti} h_i$$

Attention weights

$$\alpha_t = \text{softmax}(e_t)$$

Alignment scores

$$e_{ti} = f_{\text{att}}(s_{t-1}, h_i)$$

f_{att}: simple feedforward network (e.g. MLP)

Slide Credit : https://cs.uwaterloo.ca/~wenhuche/teaching/cs886/

Decoupling from RNNs

- **Recall:** attention determines the **importance of elements** to be passed forward in the model.
  - These weights lets the model pay **attention** to the **most significant parts**

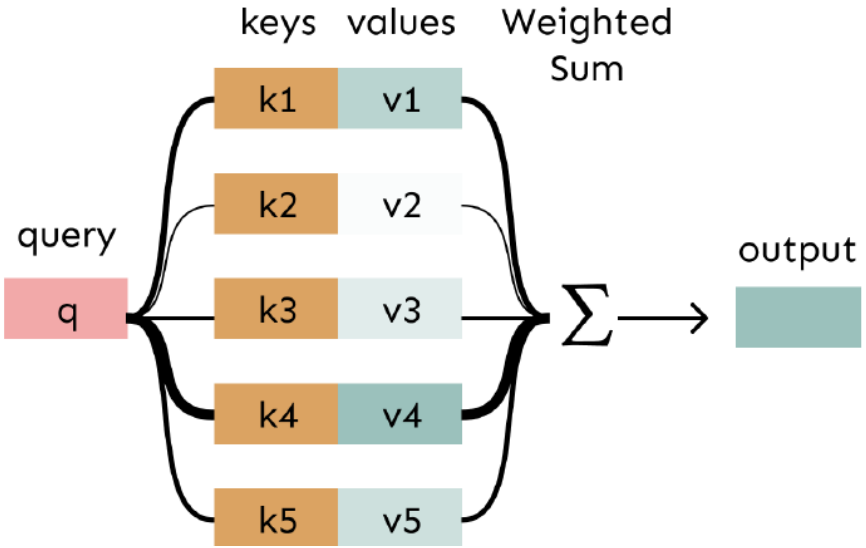
- **Objective**: a more general attention mechanism not confined to RNNs
  - We need a modified procedure to:
    1. Determine weights based on context that indicate the elements to attend to
    2. Apply these weights to enhance attended features

Slide Credit : https://cs.uwaterloo.ca/~wenhuche/teaching/cs886/

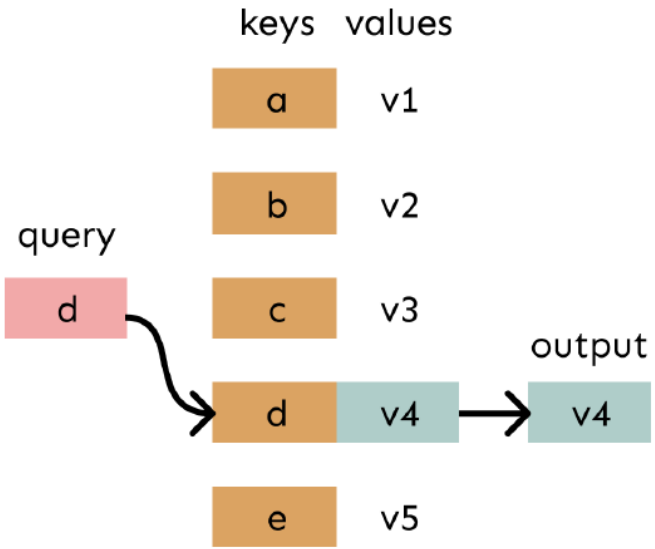
Decoupling from RNNs

Attention is just a **weighted** average – this is very powerful if the weights are learned!

In **attention**, the query matches all keys *softly*, to a weight between 0 and 1. The keys' values are multiplied by the weights and summed.



In a **lookup table**, we have a table of keys that map to values. The query matches one of the keys, returning its value.



Decoupling from RNNs

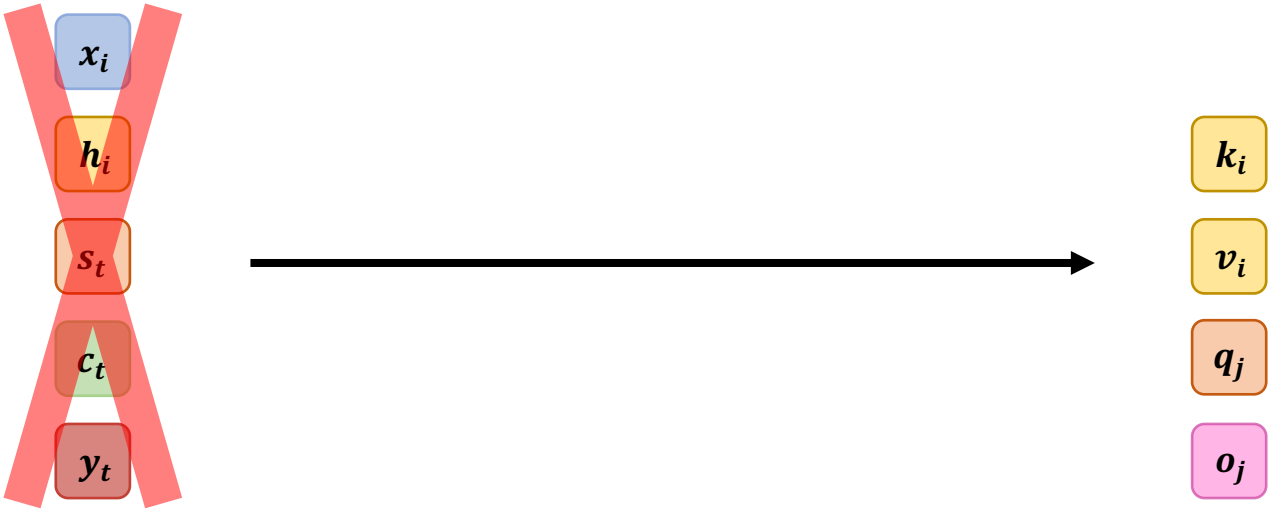
- RNN Notation

x_i	Input for position i in source sequence
h_i	Hidden states for position i in source sequence
s_t	Hidden states for position t in target sequence
c_t	Context vector for position t in target sequence
y_t	Output for position t in target sequence

Slide Credit : https://cs.uwaterloo.ca/~wenhuche/teaching/cs886/

Decoupling from RNNs

- New Notation



Slide Credit : https://cs.uwaterloo.ca/~wenhuche/teaching/cs886/

Decoupling from RNNs

- New Notation

k_i **Key** vector for position i in an arbitrary sequence

v_i **Value** vector for position i in an arbitrary sequence

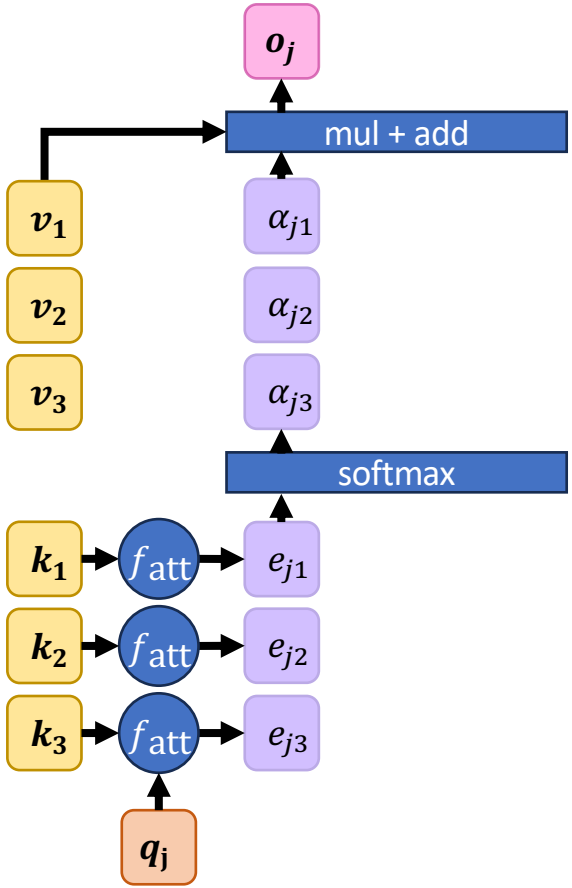
q_j **Query** vector for position j in a (same/different) arbitrary sequence

o_j **Output** vector corresponding to position j

Slide Credit : https://cs.uwaterloo.ca/~wenhuche/teaching/cs886/

A more general attention

$Attention(\boldsymbol{q}, \boldsymbol{k}, \boldsymbol{v})$



Output vectors

$$\boldsymbol{o_j} = \sum_i \alpha_{ji} \boldsymbol{v_i} \quad \in \mathbb{R}^{d_k}$$

Attention weights

$$\boldsymbol{\alpha_j} = \text{softmax}(\boldsymbol{e_j})$$

Alignment scores

$$e_{ij} = f_{\text{att}}(\boldsymbol{q_j}, \boldsymbol{k_i})$$

Goal: find the "alignment" or "compatibility" of keys with a query to scale values

$f_{\text{att}}(\cdot)$

Keys: $\boldsymbol{k_i} \in \mathbb{R}^{d_k}$

Values: $\boldsymbol{v_i} \in \mathbb{R}^{d_k}$

Query: $\boldsymbol{q_j} \in \mathbb{R}^{d_q}$

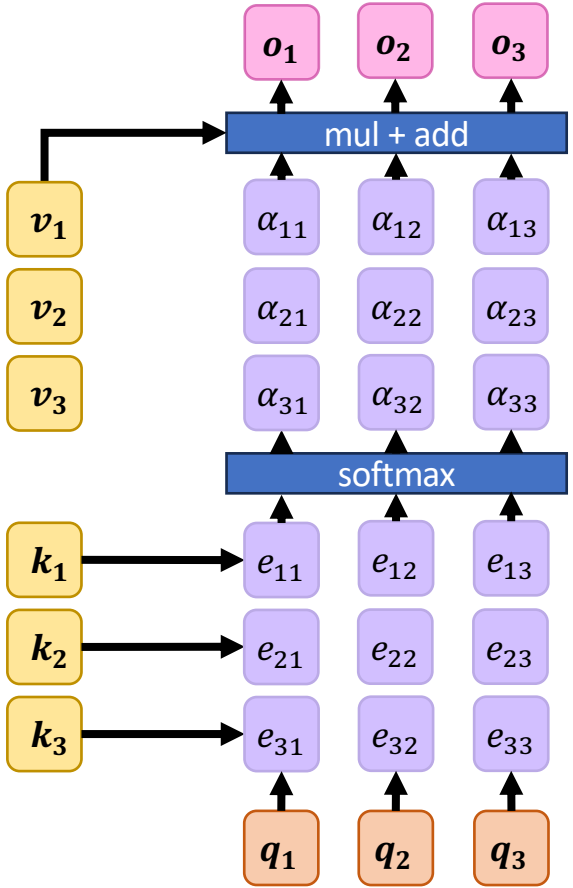
$i \in \{1, \ldots, N\}$

$j \in \{1, \ldots, T\}$

Slide Credit : https://cs.uwaterloo.ca/~wenhuche/teaching/cs886/

A more general attention

$Attention(\boldsymbol{q}, \boldsymbol{k}, \boldsymbol{v})$



Attention

Alignment

Output vectors
$\boldsymbol{o_j} \in \mathbb{R}^{d_k}$,
for j in $\{1, \ldots, T\}$

Attention weights
$\boldsymbol{\alpha_j} = \text{softmax}(\boldsymbol{e_j})$

Alignment scores
$e_{ij} = f_{\text{att}}(\boldsymbol{q_j}, \boldsymbol{k_i})$

$f_{\text{att}}(\cdot)$

Keys: $\boldsymbol{k_i} \in \mathbb{R}^{d_k}$
Values: $\boldsymbol{v_i} \in \mathbb{R}^{d_k}$
Query: $\boldsymbol{q_j} \in \mathbb{R}^{d_q}$

$i \in \{1, \ldots, N\}$
$j \in \{1, \ldots, T\}$

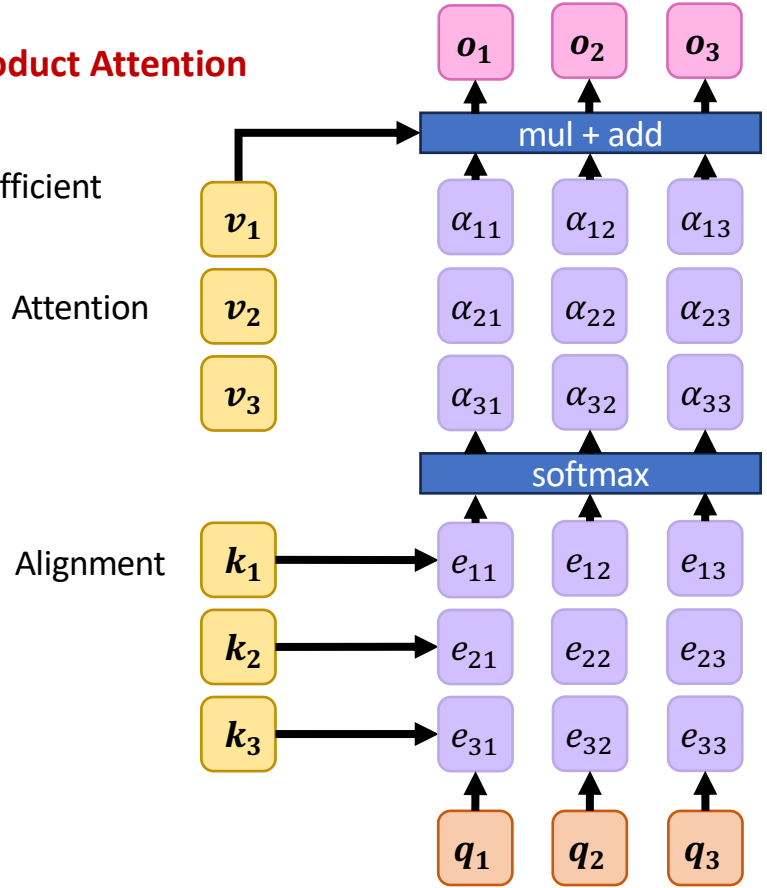
Goal: find the "alignment" or "compatibility" between keys and queries to scale values

Slide Credit : https://cs.uwaterloo.ca/~wenhuche/teaching/cs886/

Attention in Transformer

Scaled Dot-Product Attention

- Faster
- More space-efficient



Output vectors

$$o_j \in \mathbb{R}^{d_k}$$

Attention weights

$$\alpha_j = \text{softmax}(e_j)$$

Alignment scores

$$e_{ij} = f_{\text{att}}(q_j, k_i) = \frac{q_j \cdot k_i}{\sqrt{d_k}}$$

$f_{\text{att}}(\cdot)$ = scaled dot-product

Keys: $k_i \in \mathbb{R}^{d_k}$
Values: $v_i \in \mathbb{R}^{d_k}$
Query: $q_j \in \mathbb{R}^{d_k}$

} Share the same dimension

Slide Credit : https://cs.uwaterloo.ca/~wenhuche/teaching/cs886/

Attention is a *general* Deep Learning technique

- We've seen that attention is a great way to improve the sequence-to-sequence model for Machine Translation.
- However: You can use attention in many architectures (not just seq2seq) and many tasks (not just MT)

- **More general definition of attention**:
  - Given a set of vector *values*, and a vector *query*, **attention** is a technique to compute a weighted sum of the values, dependent on the query.

- We sometimes say that the query *attends to* the values.
- For example, in the seq2seq + attention model, each decoder hidden state (query) *attends to* all the encoder hidden states (values).

Attention is a *general* Deep Learning technique

- More general definition of attention:
  - Given a set of vector *values*, and a vector *query*, attention is a technique to compute a weighted sum of the values, dependent on the query.

Intuition:

- The weighted sum is a *selective summary* of the information contained in the values, where the query determines which values to focus on.
- Attention is a way to obtain a *fixed-size representation of an arbitrary set of representations* (the values), dependent on some other representation (the query).

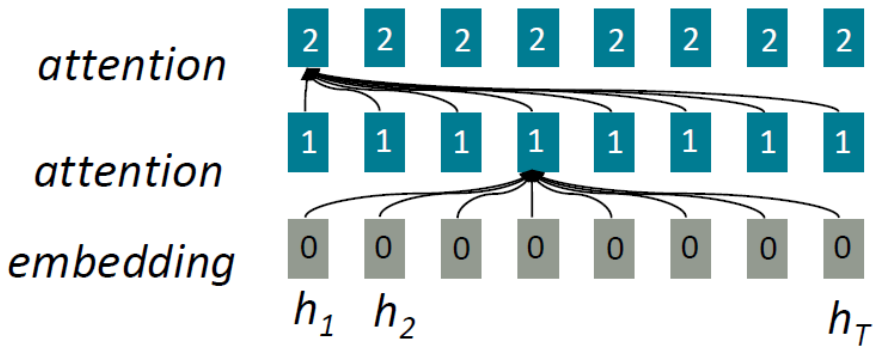
Upshot:

- Attention has become the powerful, flexible, general way pointer and memory manipulation in all deep learning models. A new idea from after 2010! From NMT!

https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1214/slides/cs224n-2021-lecture09-transformers.pdf

If not recurrence, then what? **How about attention?**

- **Attention** treats each word's representation as a **query** to access and incorporate information from **a set of values.**
  - We saw attention from the **decoder** to the **encoder**; today we'll think about attention **within a single sentence**.
- Number of unparallelizable operations does not increase sequence length.
- Maximum interaction distance: O(1), since all words interact at every layer!



All words attend to all words in previous layer; most arrows here are omitted

https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1214/slides/cs224n-2021-lecture09-transformers.pdf

Self-Attention

- Recall: Attention operates on **queries**, **keys**, and **values.**
  - We have some **queries** $q_1, q_2, \ldots, q_T$. Each query is $q_i \in \mathbb{R}^d$
  - We have some **keys** $k_1, k_2, \ldots, k_T$. Each key is $k_i \in \mathbb{R}^d$
  - We have some **values** $v_1, v_2, \ldots, v_T$. Each value is $v_i \in \mathbb{R}^d$
- In **self-attention**, the queries, keys, and values are drawn from the same source.
  - For example, if the output of the previous layer is $x_1, \ldots, x_T$, (one vec per word) we could let $v_i = k_i = q_i = x_i$ (that is, use the same vectors for all of them!)
- The (dot product) self-attention operation is as follows:

The number of queries can differ from the number of keys and values in practice.

$$e_{ij} = q_i^\top k_j$$

Compute **key-query** affinities

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})}$$

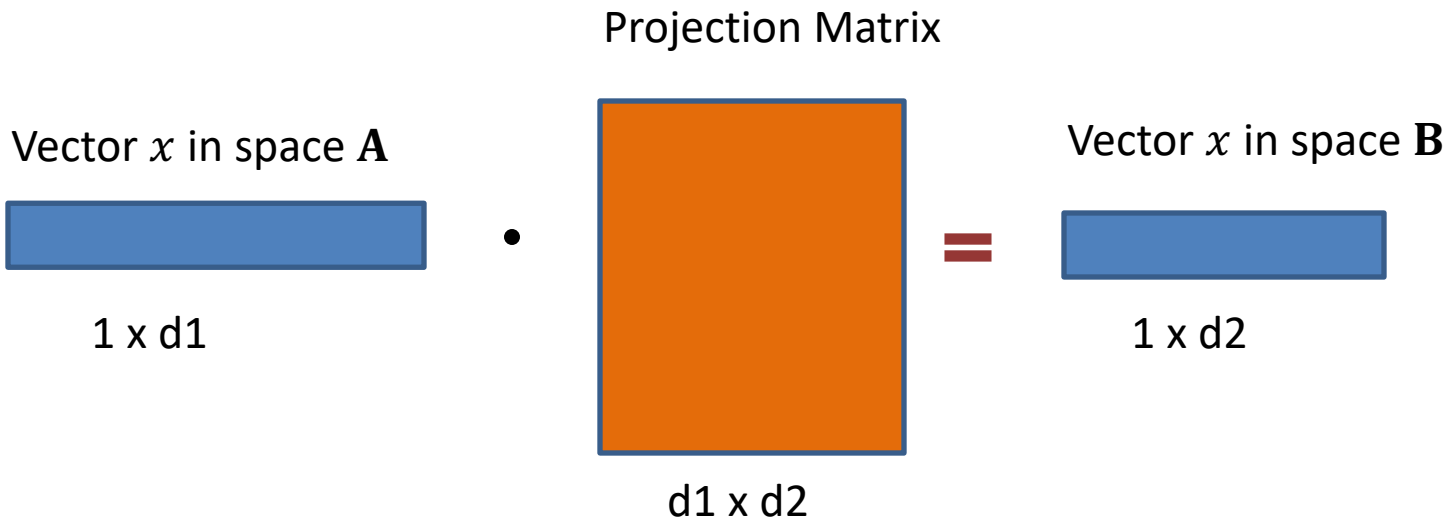
Compute attention weights from affinities (softmax)

$$\text{output}_i = \sum_j \alpha_{ij} v_j$$

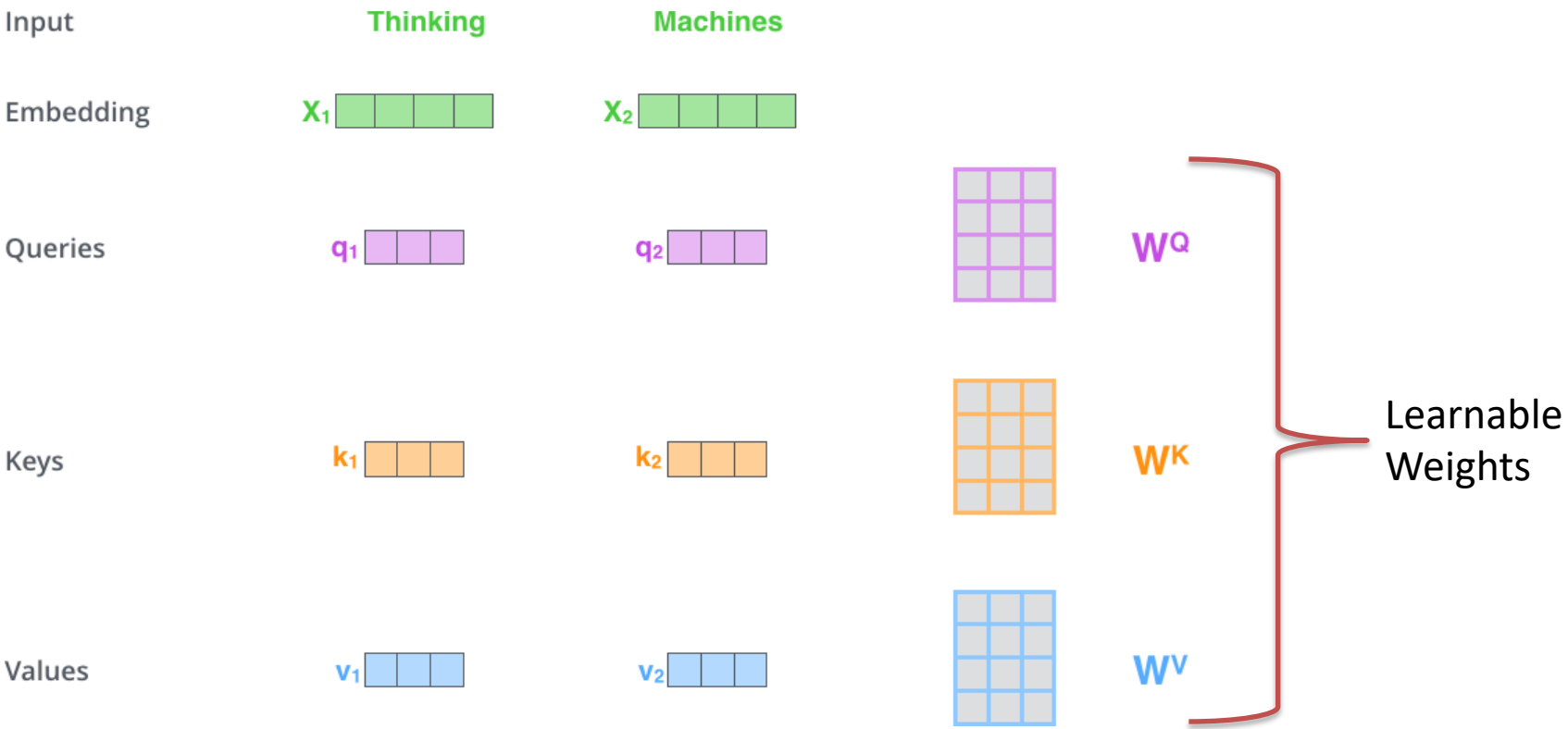
Compute outputs as weighted sum of **values**

https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1214/slides/cs224n-2021-lecture09-transformers.pdf

Projection

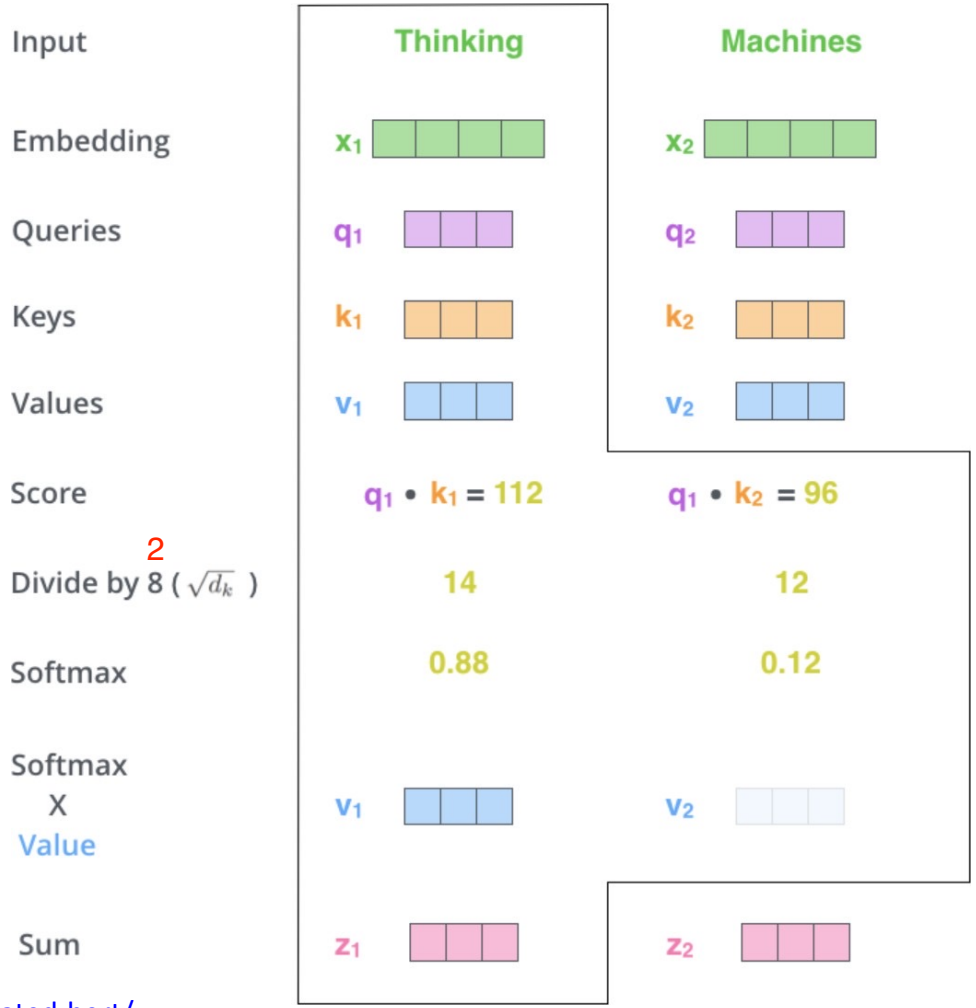

Projection Matrix
Vector x in space $\mathbf{A}$
Vector x in space $\mathbf{B}$
1 x d1
1 x d2
d1 x d2

Self-attention

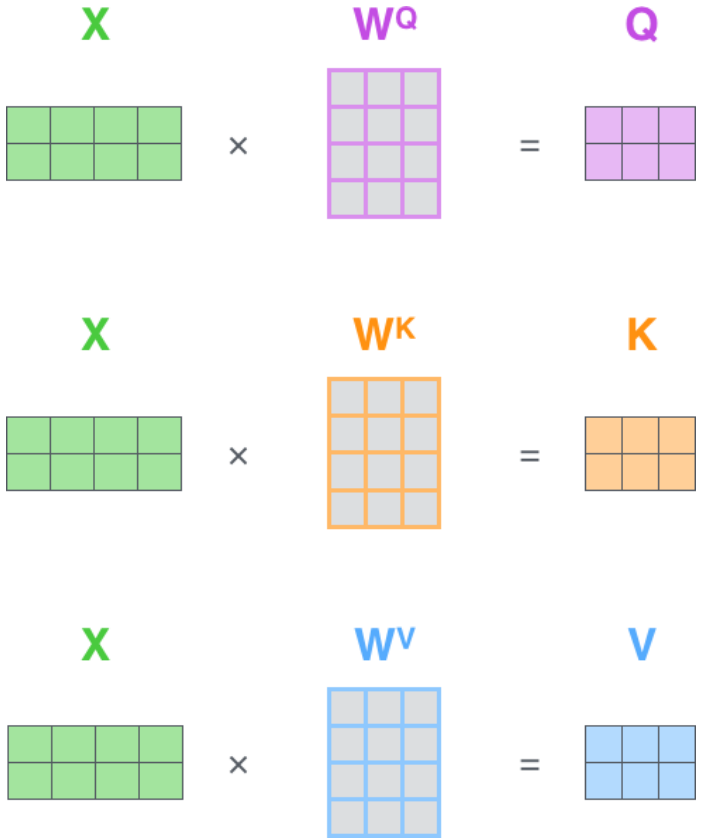


Acknowledgement to Figure from http://jalammar.github.io/illustrated-bert/

Self-attention



Acknowledgement to Figure from http://jalammar.github.io/illustrated-bert/



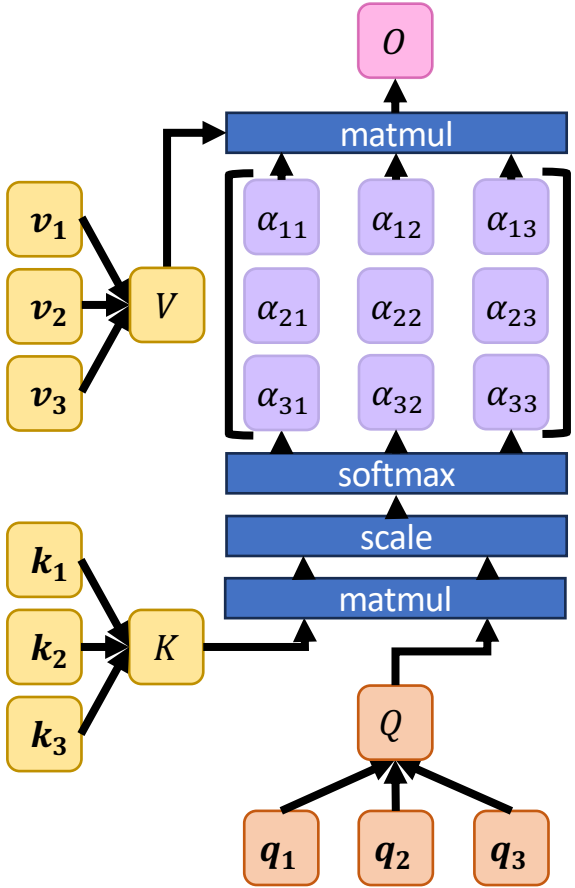
$$\text{softmax}\left(\frac{Q \times K^T}{\sqrt{d_k}}\right) V = Z$$

Acknowledgement to Figure from http://jalammar.github.io/illustrated-bert/

Attention in Transformer

Calculate dot-products in **parallel** with **matrix multiplication**

- High concurrency on modern hardware (GPUs)
- Independently calculates each query



Matrix of outputs $O \in \mathbb{R}^{T \times d_k}$

Matrix containing attention weights

$$A[j,i] = \text{softmax}\left(\frac{Q \cdot K^T}{\sqrt{d_k}}\right)$$

Alignment scores

$e_{ji} = f_{\text{att}}(\boldsymbol{q_j}, \boldsymbol{k_i}) = \frac{Q \cdot K^T}{\sqrt{d_k}}$ [j, i]

Alignment

$f_{\text{att}}(\cdot) =$ scaled dot-product

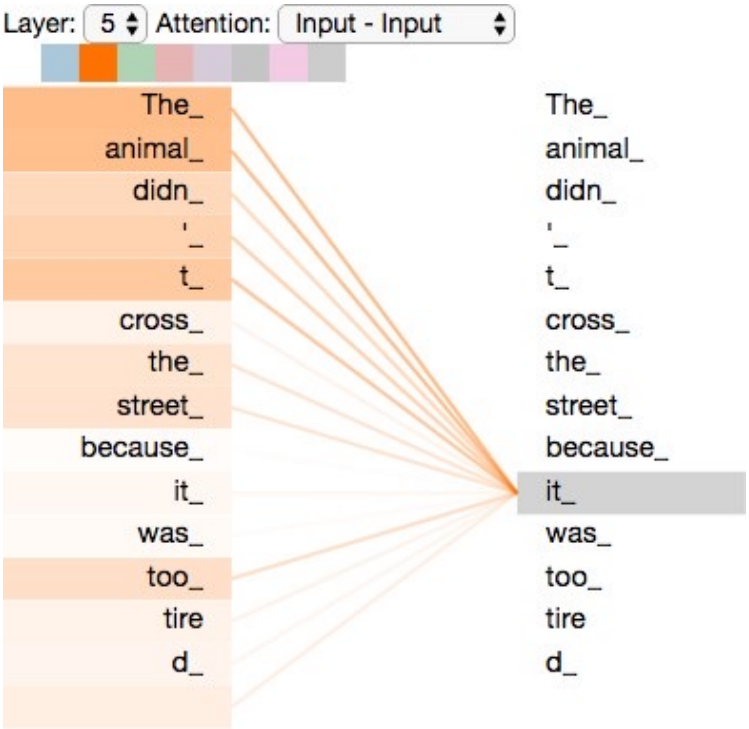
Keys: $K \in \mathbb{R}^{N \times d_k}$

Values: $V \in \mathbb{R}^{N \times d_k}$

Query: $Q \in \mathbb{R}^{T \times d_k}$

Slide Credit : https://cs.uwaterloo.ca/~wenhuche/teaching/cs886/

Self-attention

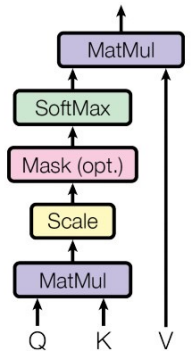


The **animal** didn't cross the street because **it** was too tired

The animal didn't cross the **street** because **it** was too wide

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

Scaled Dot-Product Attention



Equation and Figure in (Vaswani et al., 2017)

Acknowledgement to Figure from http://jalammar.github.io/illustrated-bert/

Learning Transformer Attention

Self-Attention

Cross-Attention

** X, Y are matrices of arbitrary sequences

Slide Credit : https://cs.uwaterloo.ca/~wenhuche/teaching/cs886/

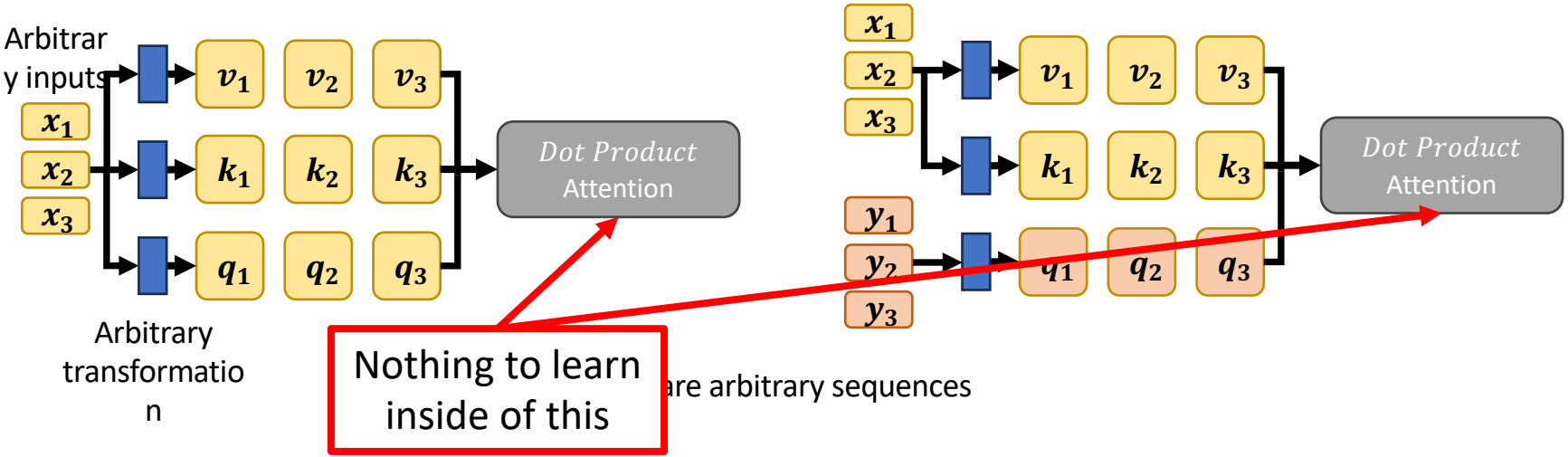
Learning Transformer Attention

Self-Attention

- Keys, values, and queries are all derived from the same source

Cross-Attention

- Keys-values and queries are derived from separate sources



Slide Credit : https://cs.uwaterloo.ca/~wenhuche/teaching/cs886/

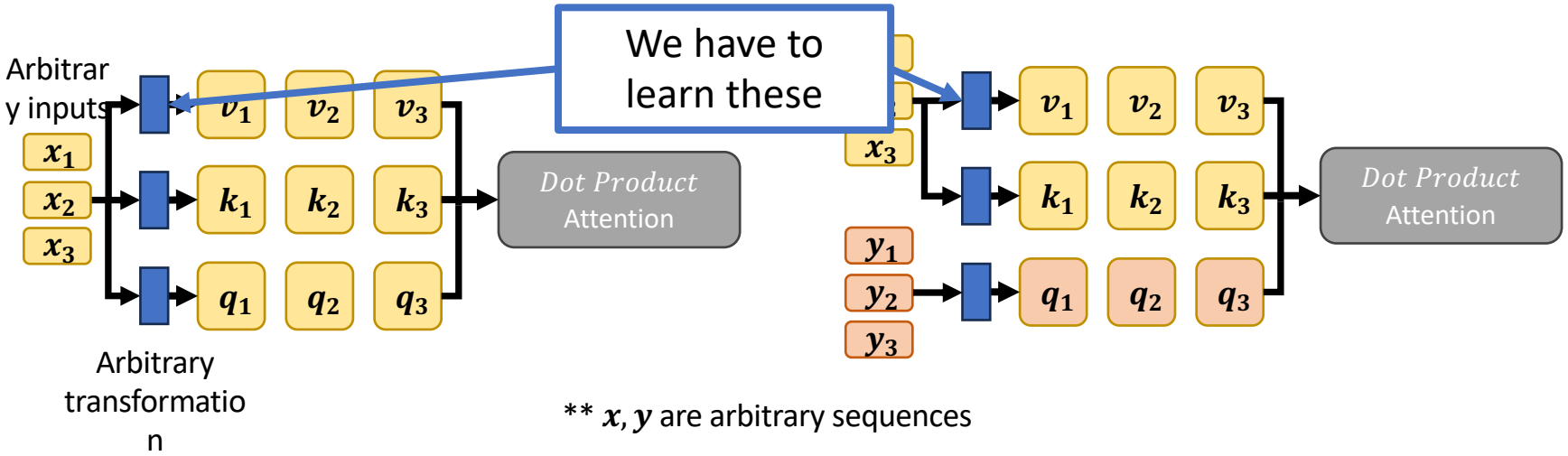
Learning Transformer Attention

Self-Attention

- Keys, values, and queries are all derived from the same source

Cross-Attention

- Keys-values and queries are derived from separate sources



** x, y are arbitrary sequences

Slide Credit : https://cs.uwaterloo.ca/~wenhuche/teaching/cs886/

Multi-Head Attention

- Builds on Scaled Dot-Product Attention
- Extension of generalized attention mentioned outlined previously
- Leverages multiple heads to attend to different things

Slide Credit : https://cs.uwaterloo.ca/~wenhuche/teaching/cs886/

Learning Multi-Head Attention

Why do we need multiple heads?

Output vectors

$$\boldsymbol{o_j} = \sum_i \alpha_{ji} \boldsymbol{v_i} \quad \in \mathbb{R}^{d_k}$$

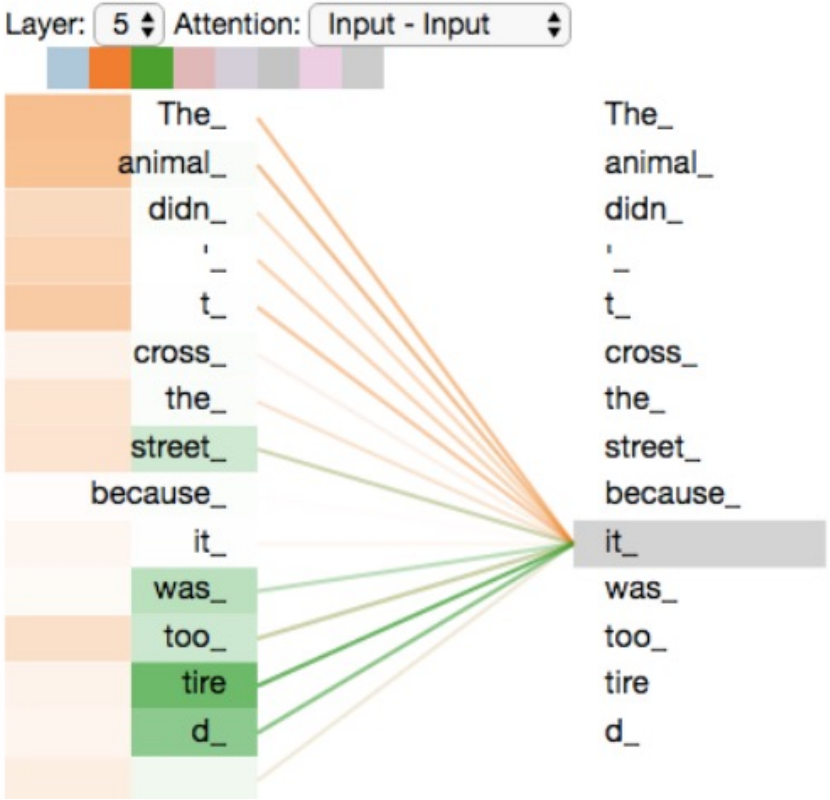
Attention

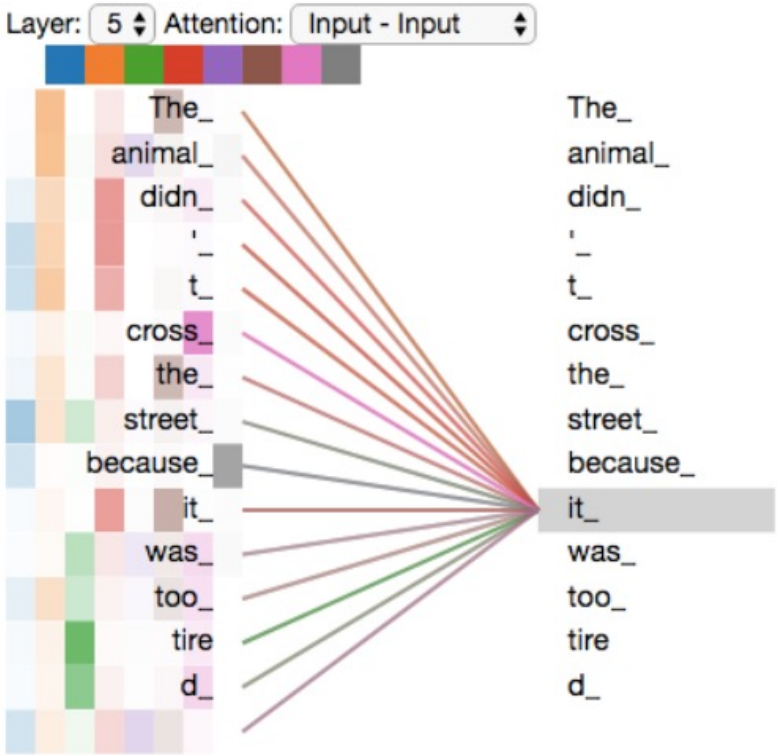
o_j

Since we've summed through the n positions, we lose resolution in our representation

Slide Credit : https://cs.uwaterloo.ca/~wenhuche/teaching/cs886/

Multi-Head Attention



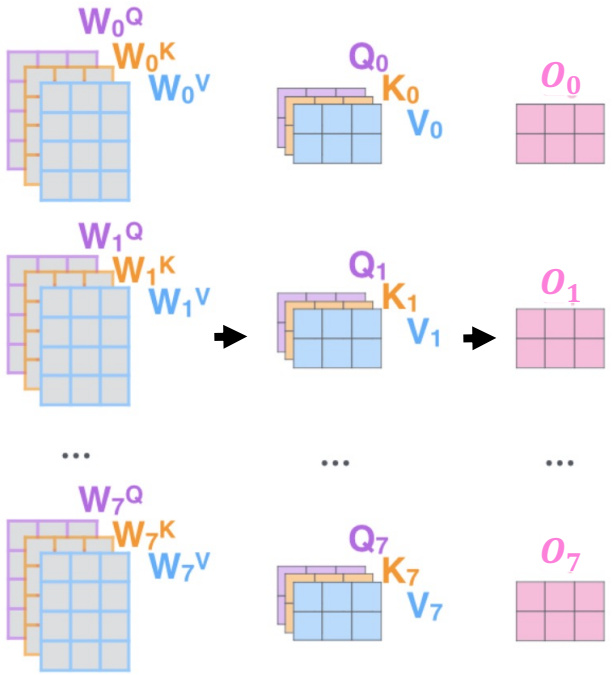


Slide Credit : https://cs.uwaterloo.ca/~wenhuche/teaching/cs886/

Learning Multi-Head Attention

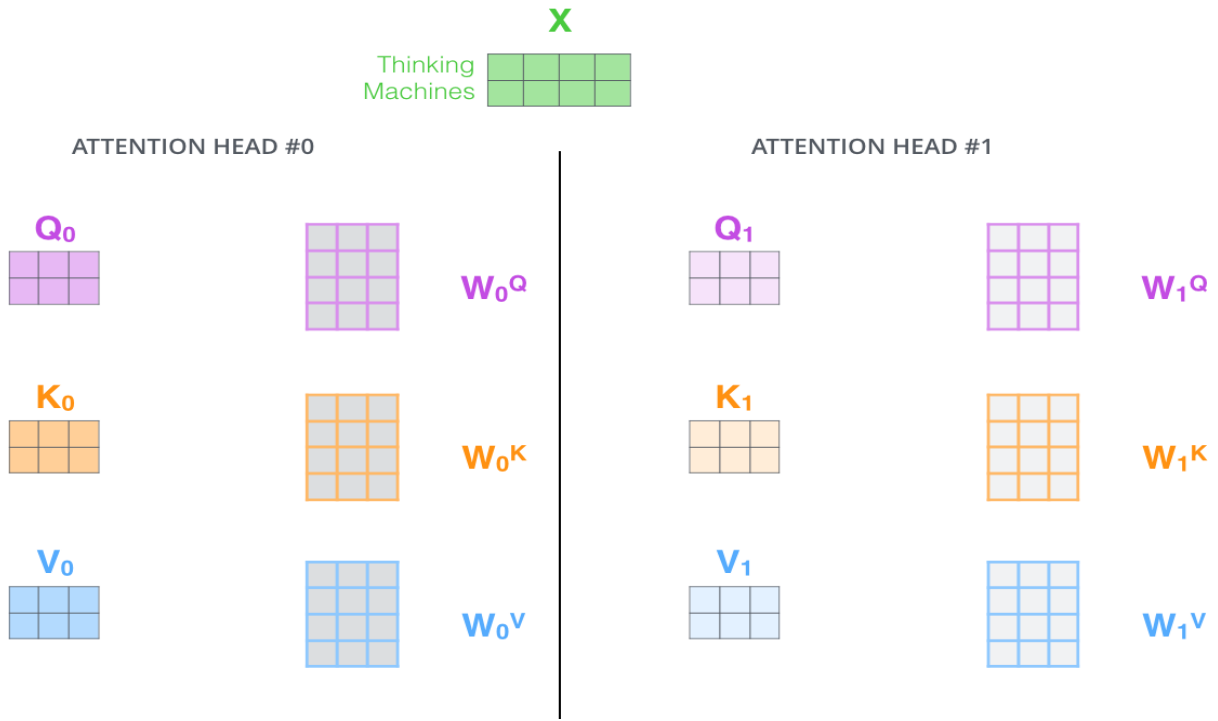
- Main idea:
  - Learn multiple sets of weights matrices to attend to different things
  - Preserve resolution since more heads increases chance that the information is maintained

- Allows model to jointly attend to information from different representation subspaces (like ensembling)



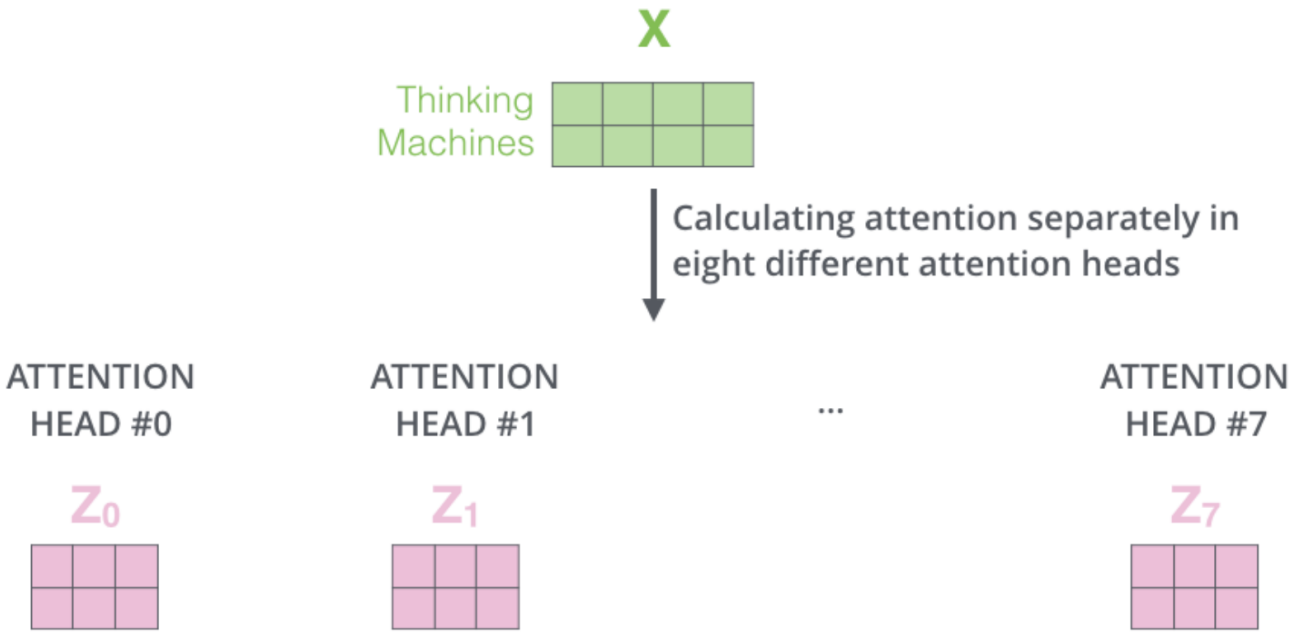
Slide Credit : https://cs.uwaterloo.ca/~wenhuche/teaching/cs886/

Multi-head attention



Acknowledgement to Figure from http://jalammar.github.io/illustrated-bert/

Multi-head Attention



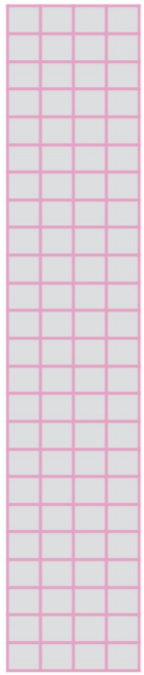
Acknowledgement to Figure from http://jalammar.github.io/illustrated-bert/

Multi-head Attention

1) Concatenate all the attention heads

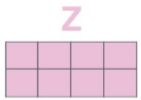
2) Multiply with a weight matrix W^O that was trained jointly with the model

X



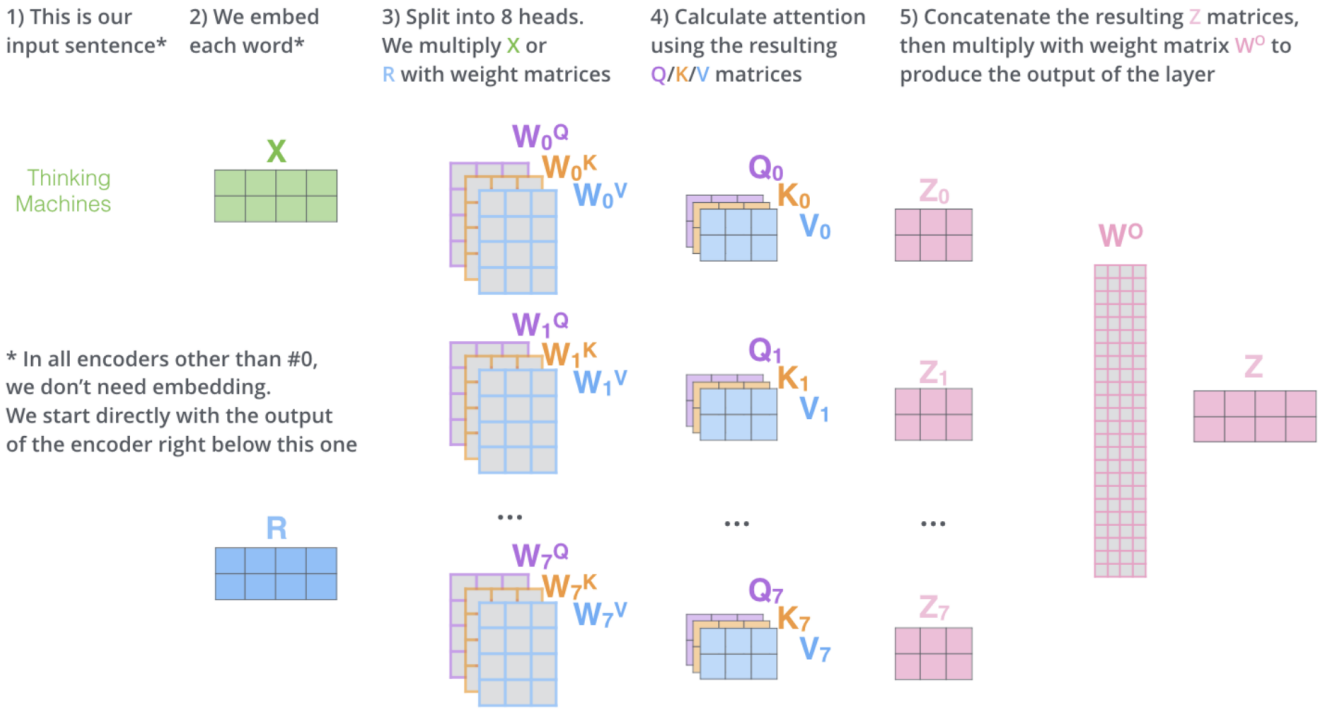
3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN

=



Acknowledgement to Figure from http://jalammar.github.io/illustrated-bert/

Multi-head Attention



- Single multiplication per layer ⇒ efficiency on GPU/TPU

Acknowledgement to Figure from http://jalammar.github.io/illustrated-bert/

Learning Multi-Head Attention

- To make computation efficient, weight matrices project to subspaces

$$W_{h_i}^{Q} \in \mathbb{R}^{d_{model} \times d_k} \rightarrow Q = XW_{h_i}^{Q} \in \mathbb{R}^{t \times d_k},$$
$$W_{h_i}^{K} \in \mathbb{R}^{d_{model} \times d_k} \rightarrow K = XW_{h_i}^{K} \in \mathbb{R}^{n \times d_k},$$
$$W_{h_i}^{V} \in \mathbb{R}^{d_{model} \times d_k} \rightarrow V = XW_{h_i}^{V} \in \mathbb{R}^{n \times d_k},$$

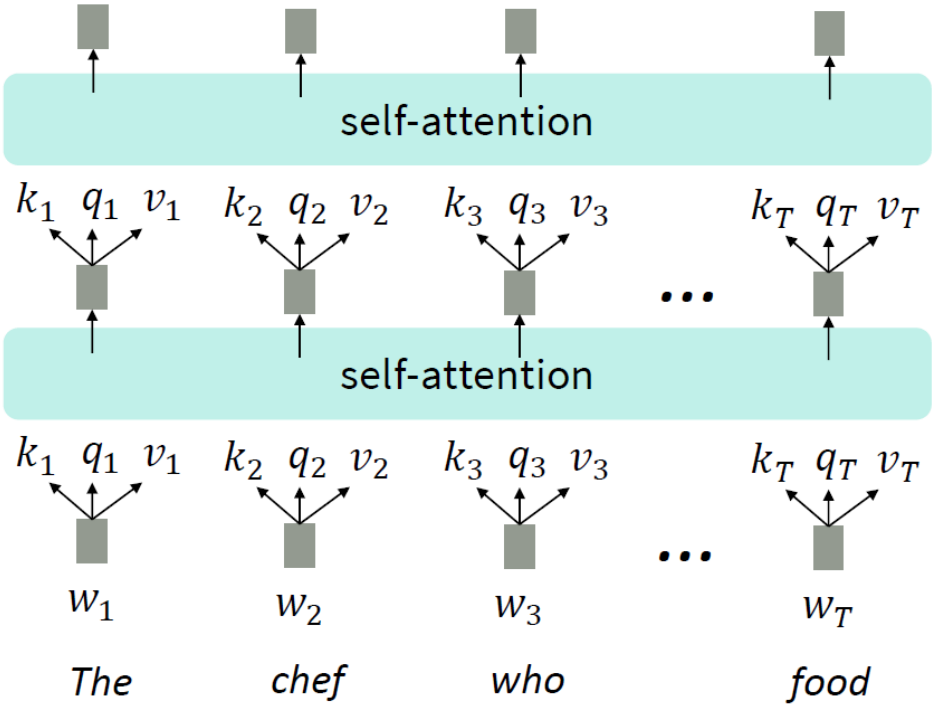
where $d_k = d_{model}/h$ (512/8 = 64 in paper)

- Together all heads take roughly the same computational time as one fully dimensioned attention head

Slide Credit : https://cs.uwaterloo.ca/~wenhuche/teaching/cs886/

Self-attention as an NLP building block

- In the diagram at the right, we have stacked self-attention blocks, like we might stack LSTM layers.
- Can self-attention be a drop-in replacement for recurrence?
- No. It has a few issues, which we'll go through.
- First, self-attention is an operation on **sets**. It has no inherent notion of order.



Self-attention doesn't know the order of its inputs.

https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1214/slides/cs224n-2021-lecture09-transformers.pdf

Barriers and solutions for Self-Attention as a building block

Barriers	Solutions
• Doesn't have an inherent notion of order!	

https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1214/slides/cs224n-2021-lecture09-transformers.pdf

Fixing the first self-attention problem: **sequence order**

- Since self-attention doesn't build in order information, we need to encode the order of the sentence in our keys, queries, and values.
- Consider representing each **sequence index** as a **vector**

$$p_i \in \mathbb{R}^d, \text{ for } i \in \{1,2,\dots,T\} \text{ are position vectors}$$

- Don't worry about what the p_i are made of yet!
- Easy to incorporate this info into our self-attention block: just add the p_i to our inputs!
- Let $\tilde{v}_i\ \tilde{k}_i, \tilde{q}_i$ be our old values, keys, and queries.

$$\begin{aligned} v_i &= \tilde{v}_i + p_i \\ q_i &= \tilde{q}_i + p_i \\ k_i &= \tilde{k}_i + p_i \end{aligned}$$

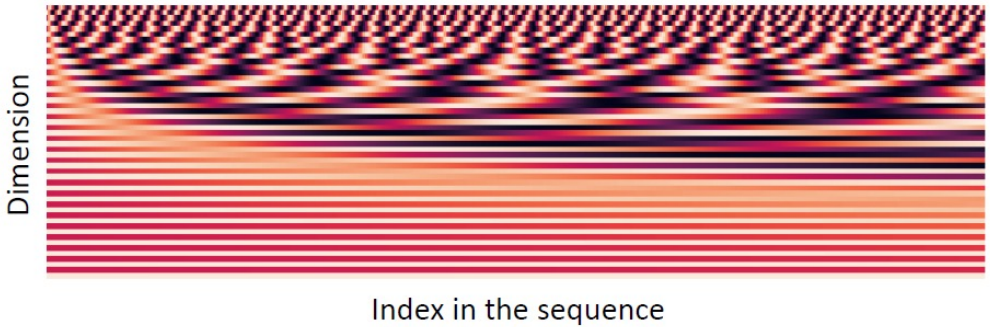
In deep self-attention networks, we do this at the first layer! You could concatenate them as well, but people mostly just add…

https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1214/slides/cs224n-2021-lecture09-transformers.pdf

Position representation vectors through sinusoids

- **Sinusoidal position representations:** concatenate sinusoidal functions of varying periods:

$$p_i = \begin{pmatrix} \sin(i/10000^{2*1/d}) \\ \cos(i/10000^{2*1/d}) \\ \vdots \\ \sin(i/10000^{2*\frac{d}{2}/d}) \\ \cos(i/10000^{2*\frac{d}{2}/d}) \end{pmatrix}$$



- Pros:
  - Periodicity indicates that maybe "absolute position" isn't as important
  - Maybe can extrapolate to longer sequences as periods restart!
- Cons:
  - Not learnable; also the extrapolation doesn't really work!

Image: https://timodenk.com/blog/linear-relationships-in-the-transformers-positional-encoding/

https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1214/slides/cs224n-2021-lecture09-transformers.pdf

Position representation vectors learned from scratch

- **Learned absolute position representations:** Let all p_i be learnable parameters!
  Learn a matrix $p \in \mathbb{R}^{d \times T}$, *and let each* p_i *be a column of that matrix!*

- Pros:
  - Flexibility: each position gets to be learned to fit the data
- Cons:
  - Definitely can't extrapolate to indices outside $1, \ldots, T$.
- Most systems use this!

- Sometimes people try more flexible representations of position:
  - Relative linear position attention [Shaw et al., 2018]
  - Dependency syntax-based position [Wang et al., 2019]

https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1214/slides/cs224n-2021-lecture09-transformers.pdf

Barriers and solutions for Self-Attention as a building block

Barriers	Solutions
• Doesn't have an inherent notion of order!	• Add position representations to the inputs
• No nonlinearities for deep learning! It's all just weighted averages	

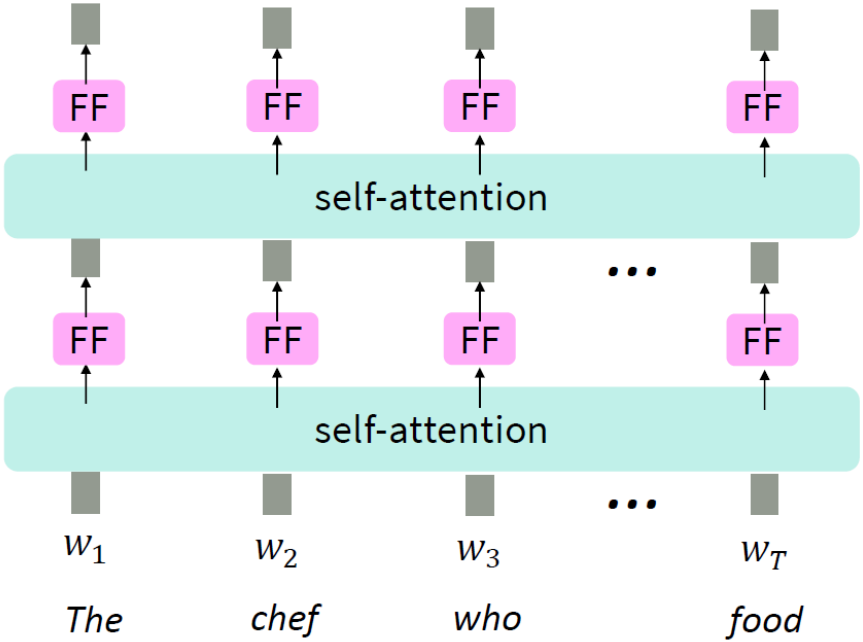
https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1214/slides/cs224n-2021-lecture09-transformers.pdf

Adding nonlinearities in self-attention

- Note that there are no elementwise nonlinearities in self-attention; stacking more self-attention layers just re-averages **value** vectors

- Easy fix: add a **feed-forward network** to post-process each output vector.

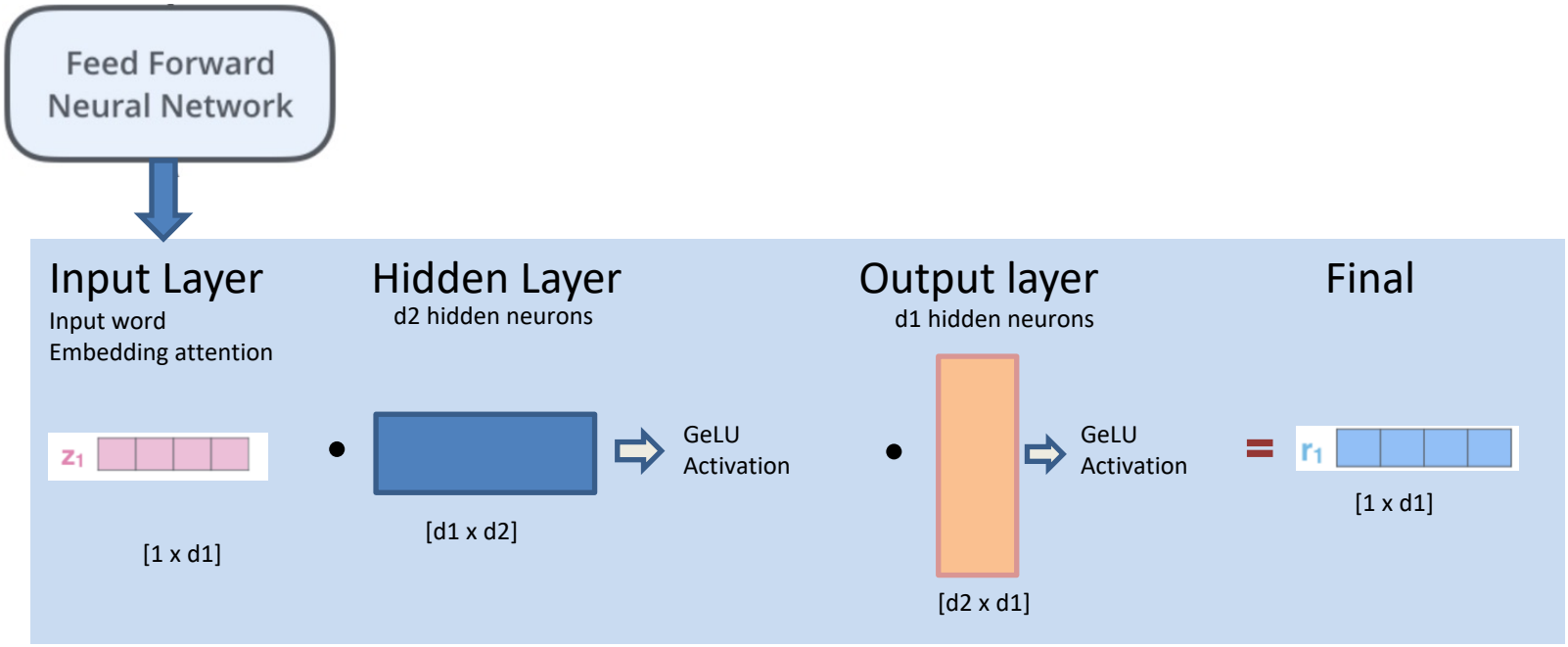
$$m_i = MLP(\text{output}_i)$$
$$= W_2 * \text{ReLU}(W_1 \times \text{output}_i + b_1) + b_2$$



Intuition: the FF network processes the result of attention

https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1214/slides/cs224n-2021-lecture09-transformers.pdf

Feed Forward Neural Network



Barriers and solutions for Self-Attention as a building block

Barriers	Solutions
• Doesn't have an inherent notion of order!	• Add position representations to the inputs
• No nonlinearities for deep learning magic! It's all just weighted averages	• Easy fix: apply the same feedforward network to each self-attention output.

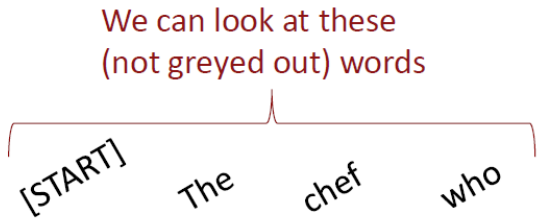
• Need to ensure we don't "look at the future" when predicting a sequence
  • Like in machine translation
  • Or language modeling	

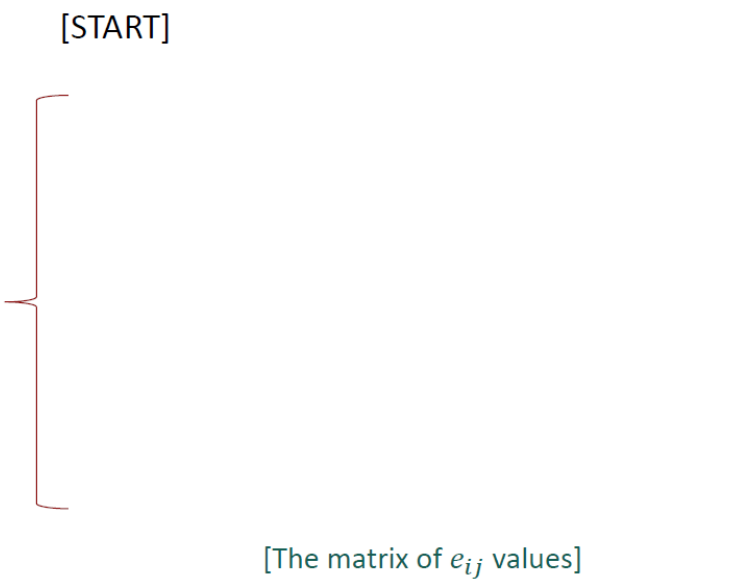
https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1214/slides/cs224n-2021-lecture09-transformers.pdf

Masking the future in self-attention

- To use self-attention in **decoders**, we need to ensure we can't peek at the future.
- At every timestep, we could change the set of **keys and queries** to include only past words. (Inefficient!)
- To enable parallelization, we **mask out attention** to future words by setting attention scores to $-\infty$.

$$e_{ij} = \begin{cases} q_i^\top k_j, j < i \\ -\infty, j \geq i \end{cases}$$



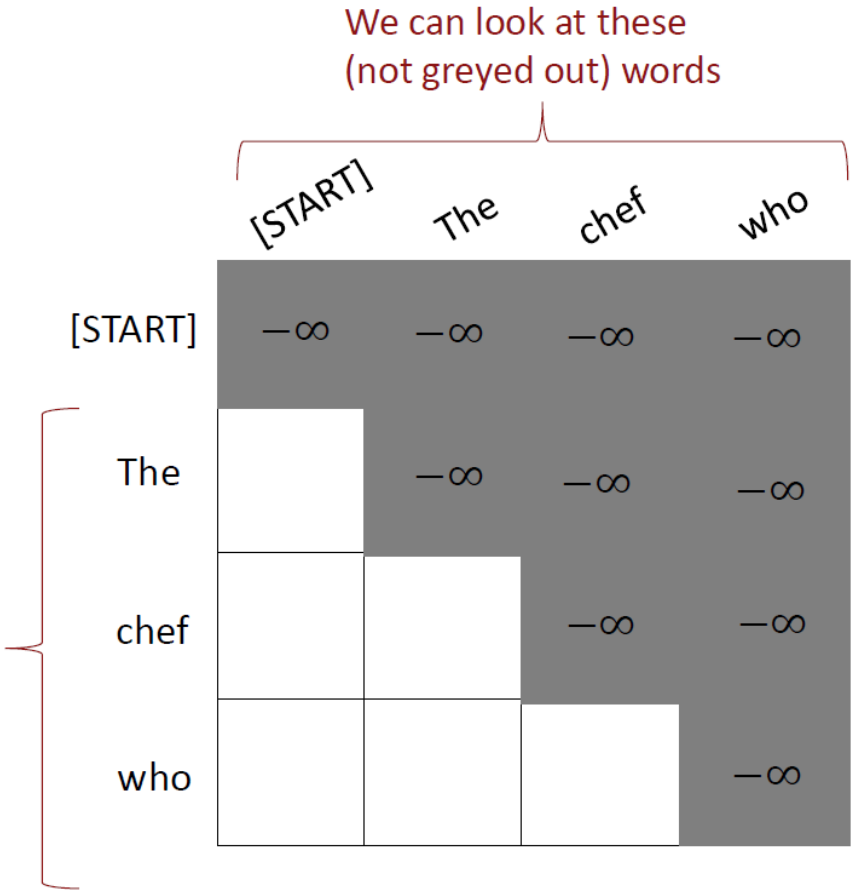


https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1214/slides/cs224n-2021-lecture09-transformers.pdf

Masking the future in self-attention

- To use self-attention in **decoders**, we need to ensure we can't peek at the future.
- At every timestep, we could change the set of **keys and queries** to include only past words. (Inefficient!)
- To enable parallelization, we **mask out attention** to future words by setting attention scores to $-\infty$.

$$e_{ij} = \begin{cases} q_i^\top k_j, j < i \\ -\infty, j \geq i \end{cases}$$



https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1214/slides/cs224n-2021-lecture09-transformers.pdf

Barriers and solutions for Self-Attention as a building block

Barriers	Solutions
• Doesn't have an inherent notion of order!	• Add position representations to the inputs
• No nonlinearities for deep learning magic! It's all just weighted averages	• Easy fix: apply the same feedforward network to each self-attention output.

• Need to ensure we don't "look at the future" when predicting a sequence
  • Like in machine translation
  • Or language modeling
	• Mask out the future by artificially setting attention weights to 0!

https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1214/slides/cs224n-2021-lecture09-transformers.pdf

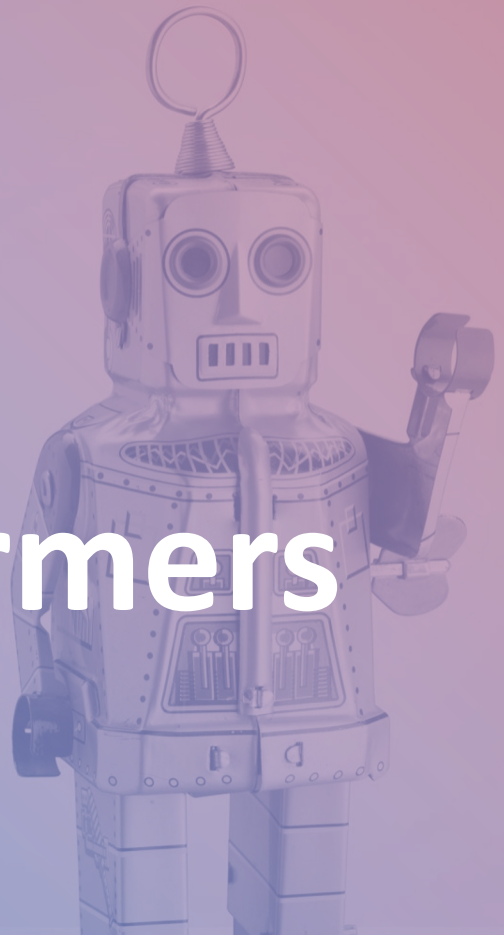
Necessities for a self-attention building block:

- **Self-attention**:
  - the basis of the method.
- **Position representations**:
  - Specify the sequence order, since self-attention is an unordered function of its inputs.
- **Nonlinearities**:
  - At the output of the self-attention block
  - Frequently implemented as a simple feed-forward network.
- **Masking**:
  - In order to parallelize operations while not looking at the future.
  - Keeps information about the future from "leaking" to the past.

- That's it! But this is not the **Transformer** model we've been hearing about.

https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1214/slides/cs224n-2021-lecture09-transformers.pdf

Transformers



Attention is All you Need (2017)

- **Key Idea:**
  - Decouple attention from RNNs
  - Use self-attention to make this efficient

- **Contributions:**
  - Multi-head attention
  - Transformer architecture

- Highly impactful (as we'll touch on later)

Transformer Architecture

Ways attention is used in the transformer:

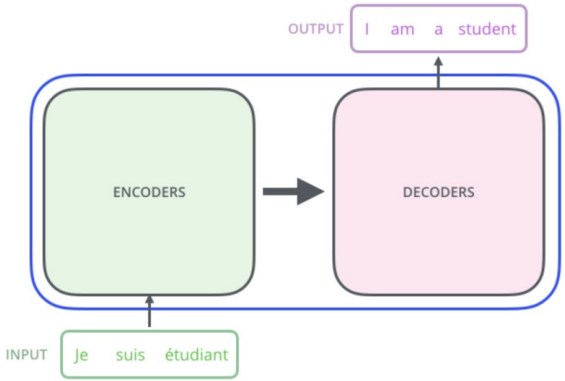
- **Self-attention in the encoder**
  - Allows the model to attend to all positions in the previous encoder layer
  - Embeds context about how elements in the sequence relate to one another

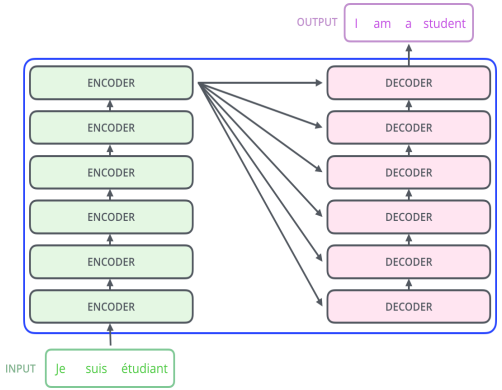
- **Masked self-attention in the decoder**
  - Allows the model to attend to all positions in the previous decoder layer up to and including the current position (during auto-regressive process)
  - Prevents forward looking bias by stopping leftward information flow during training
  - Also embeds context about how elements in the sequence relate to one another

- **Encoder-decoder cross-attention**
  - Allows decoder layers to attend all parts of the latent representation produced by the encoder
  - Pulls context from the encoder sequence over to the decoder”

A. Vaswani *et al.*, “Attention is All you Need,” in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

Transformer: a seq2seq model





Residual connection & Layer normalisation

$N = 6$

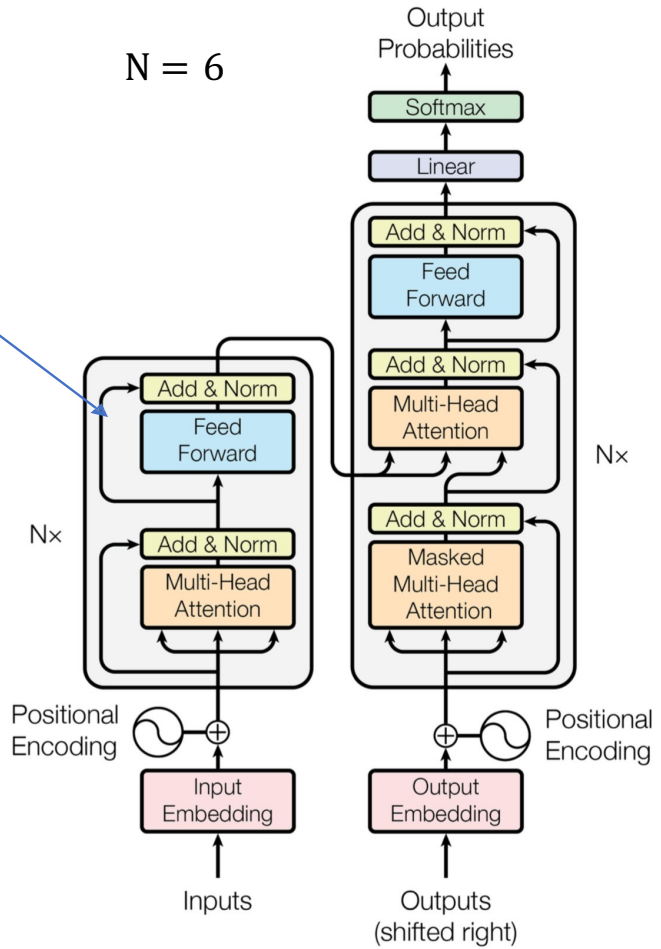


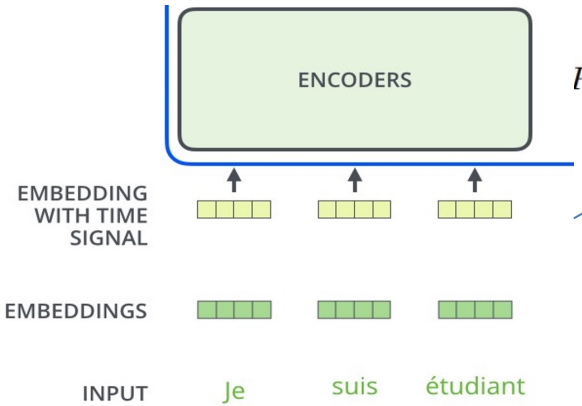
Figure 1: The Transformer - model architecture.

Figure in (Vaswani et al., 2017)

Acknowledgement to Figure from http://jalammar.github.io/illustrated-bert/

Transformer: a seq2seq model

N = 6



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

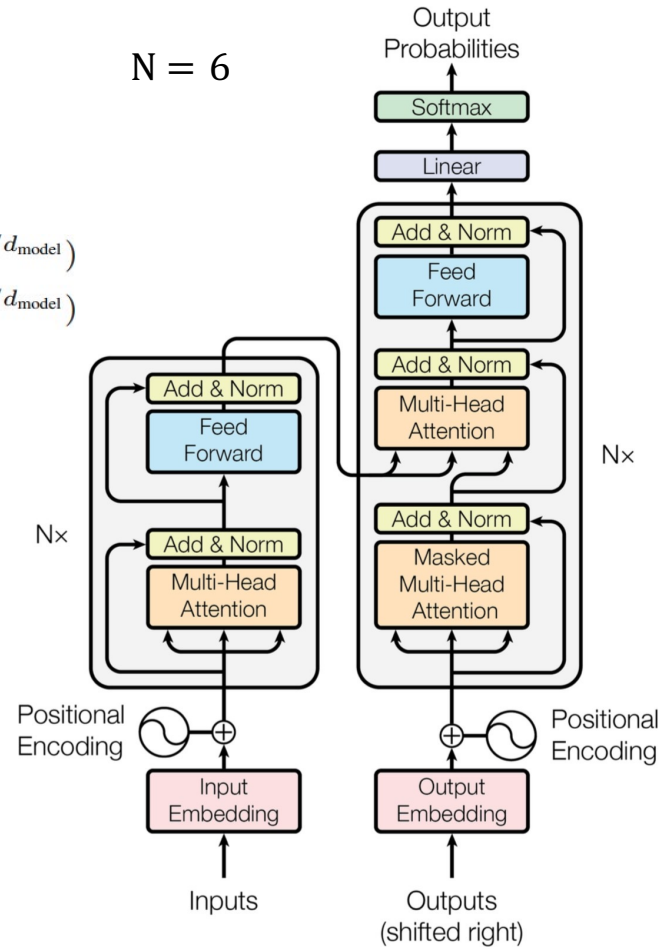
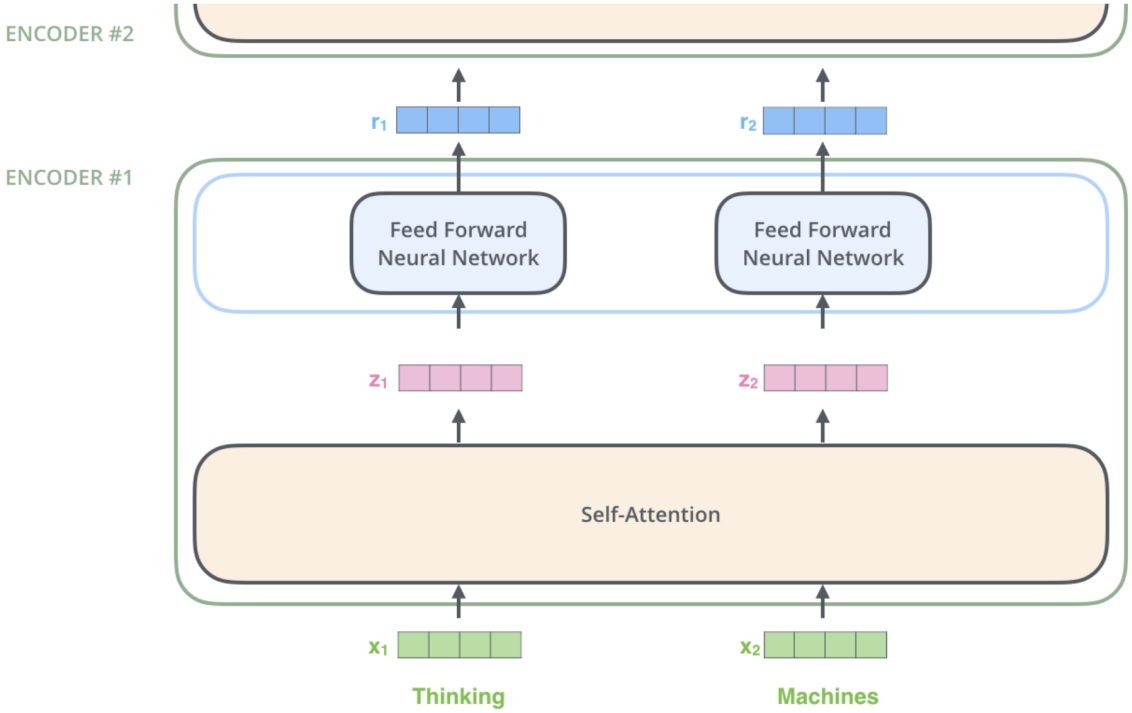


Figure 1: The Transformer - model architecture.

Figure in (Vaswani et al., 2017)

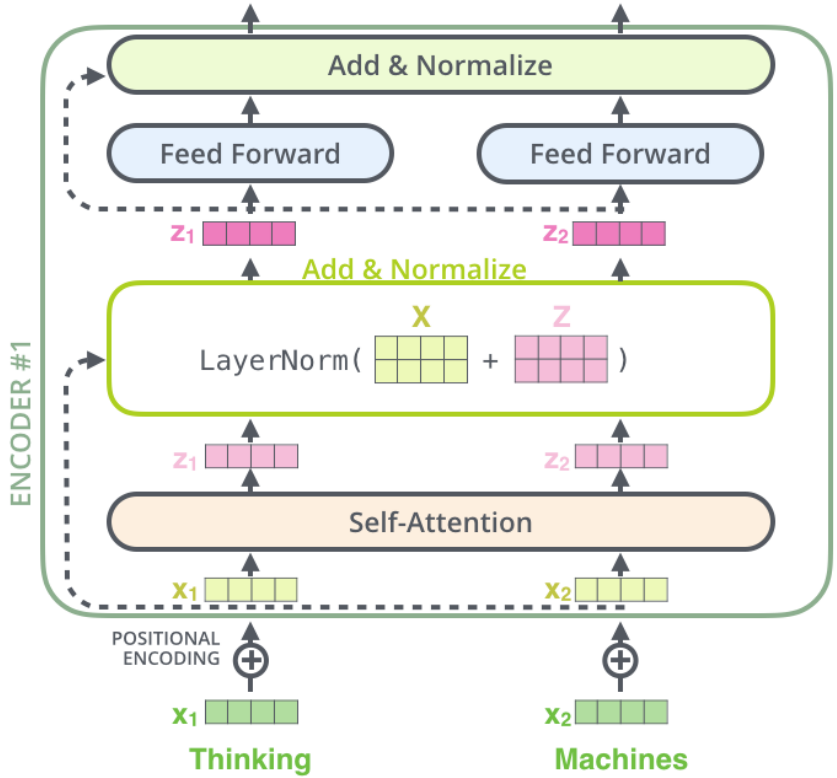
Acknowledgement to Figure from http://jalammar.github.io/illustrated-bert/

Encoding



Acknowledgement to Figure from http://jalammar.github.io/illustrated-bert/

Add and Normalize



Layer Normalization by Ba, Kiros and Hinton, https://arxiv.org/pdf/1607.06450.pdf

Acknowledgement to Figure from http://jalammar.github.io/illustrated-bert/

Layer Norm vs. Batch Norm

- Sentence 1: $[a_1, a_2, a_3]$
  - $a_1 = [1, 2, 3, 4]$
  - $a_2 = [2, 3, 4, 5]$
  - $a_3 = [3, 4, 5, 6]$
- Sentence 2: $[b_1, b_2, b_3]$
  - $b_1 = [4, 5, 6, 7]$
  - $b_2 = [5, 6, 7, 8]$
  - $b_3 = [6, 7, 8, 9]$

Batch of Two sentences

Layer Norm

For a_1

- $\mu = \frac{(1+2+3+4)}{4} = 2.5$
- $\sigma^2 = \frac{(1-2.5)^2+(2-2.5)^2+(3-2.5)^2+(4-2.5)^2}{4} = 1.25$
- Layer Normed $a_1 = \frac{[1,2,3,4]-2.5}{\sqrt{1.25+0.001}}$

For a_2, a_3, b_1, b_2, and b_3

- The calculations would be similar to those for a_1.

Batch Norm

For Feature 1

- Values: $[1, 2, 3, 4, 5, 6]$
- $\mu = \frac{(1+2+3+4+5+6)}{6} = 3.5$
- $\sigma^2 = \frac{(1-3.5)^2+(2-3.5)^2+\cdots+(6-3.5)^2}{6} = 2.917$
- Batch Normed Feature 1 = $\frac{[1,2,3,4,5,6]-3.5}{\sqrt{2.917+0.001}}$

For Features 2, 3, and 4

- The calculations would be similar to those for Feature 1.

Three Multi-Head attention blocks

- Encoder Multi-Head Attention (left)
  - Keys, values and queries are the output of the previous layer in the encoder.
  - Multiple word-word alignments.
- Decoder Masked Multi-Head Attention (lower right)
  - Set the word-word attention weights for the connections to illegal “future” words to $-\infty$.
- Encoder-Decoder Multi-Head Attention (upper right)
  - Keys and values from the output of the encoder, queries from the previous decoder layer.

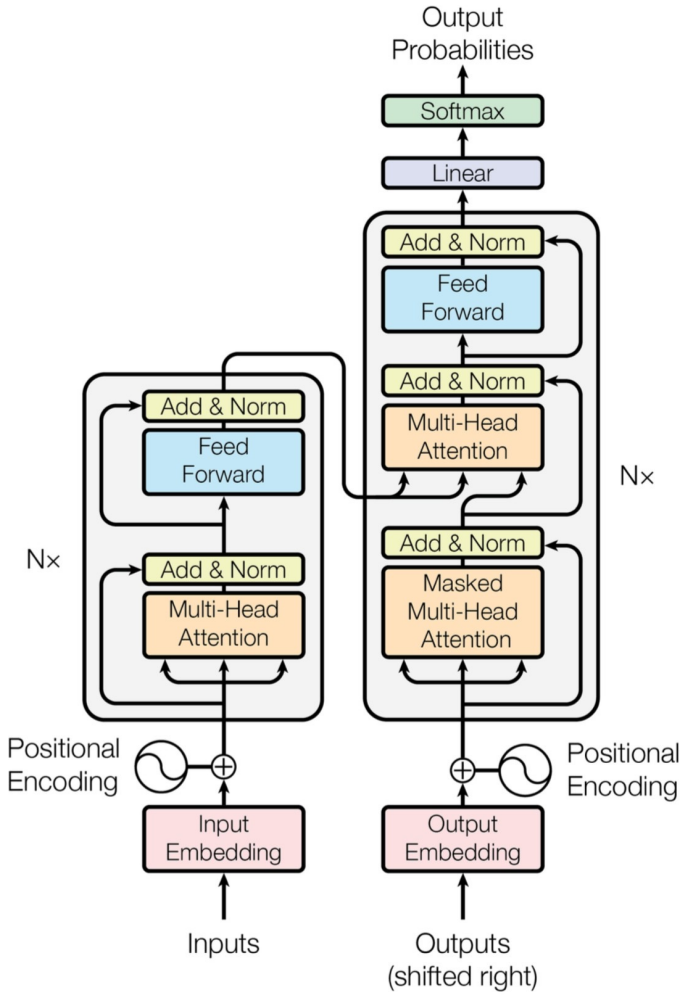
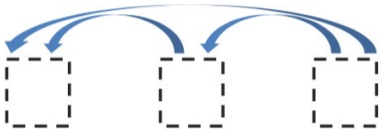


Figure 1: The Transformer - model architecture.

Figure in (Vaswani et al., 2017)

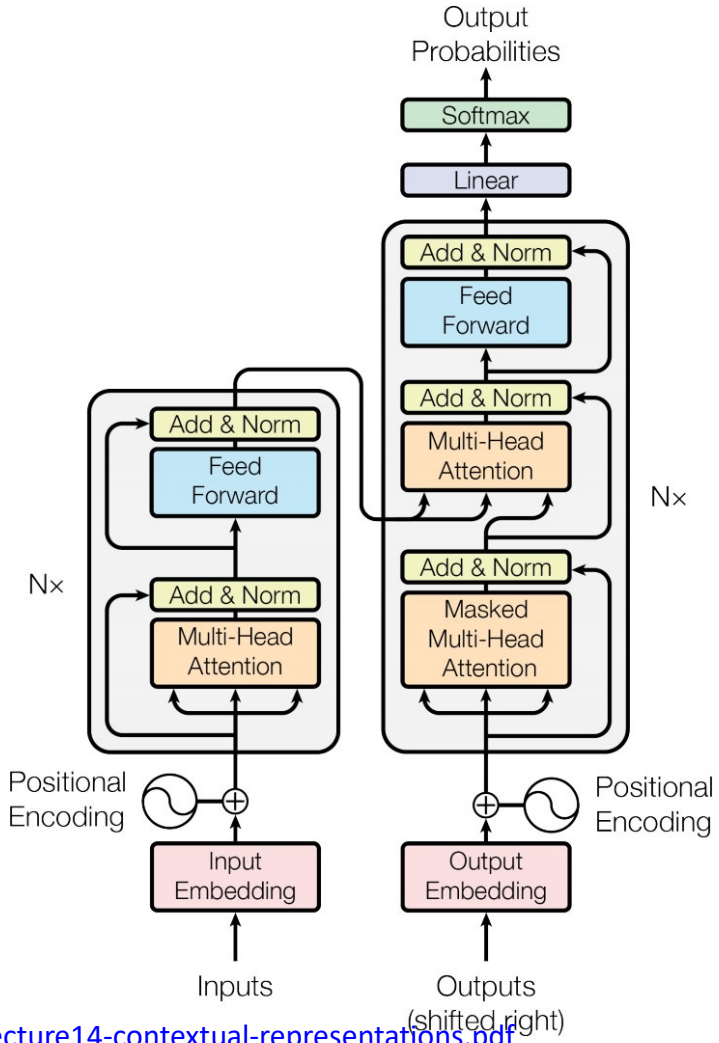
Transformer Decoder

- 2 sublayer changes in decoder
- Masked decoder self-attention on previously generated outputs:



- Encoder-Decoder Attention, where queries come from previous decoder layer and keys and values come from output of encoder

Blocks repeated 6 times also



Source : http://web.stanford.edu/class/cs224n/slides/cs224n-2020-lecture14-contextual-representations.pdf

Pre-Training

Pre-Trained Embeddings

- Train from scratch

What they will know:

May be not enough to learn relationships between words

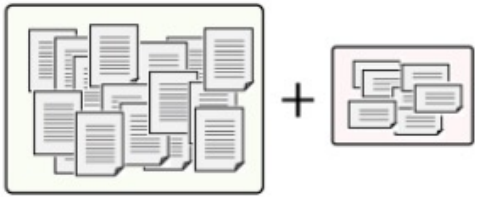
- Take pretrained (Word2Vec, GloVe)

What they will know:

Know relationships between words, but are **not** specific to the task

- Initialize with pretrained, then fine-tune

What they will know:

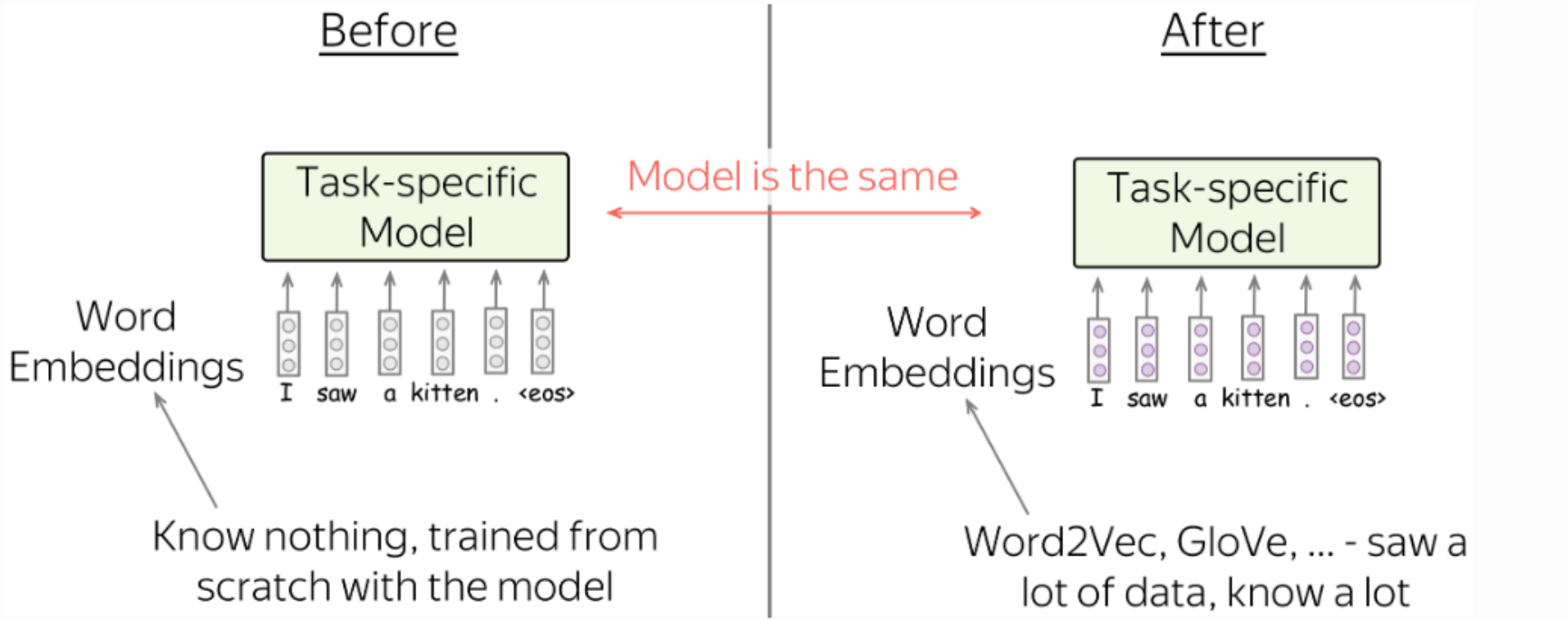


Know relationships between words and adapted for the task

"**Transfer**" knowledge from a huge unlabeled corpus to your task-specific model

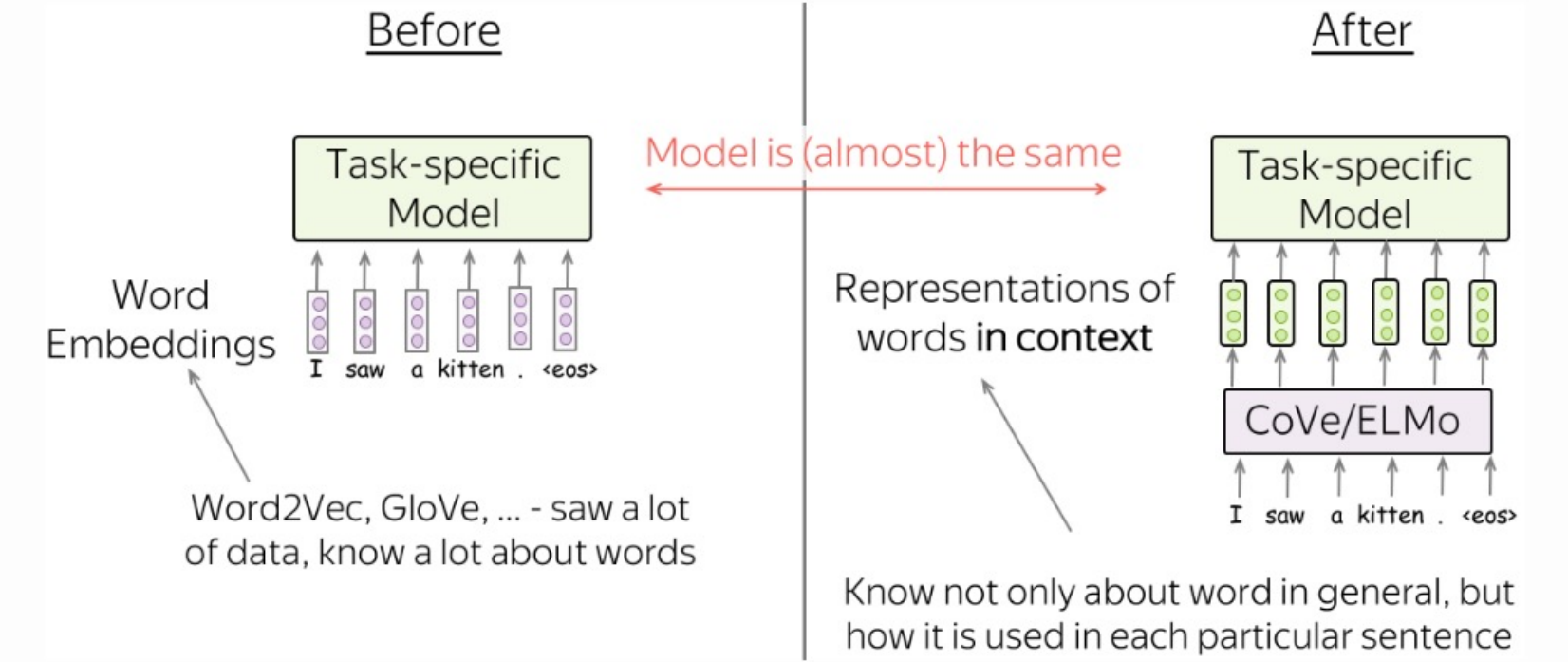
Source: https://lena-voita.github.io/nlp_course/transfer_learning.html

Pre-Trained Embeddings



Source: https://lena-voita.github.io/nlp_course/transfer_learning.html

Embeddings from Language/Translation Models



Source: https://lena-voita.github.io/nlp_course/transfer_learning.html

Embeddings from Language/Translation Models

Source: https://lena-voita.github.io/nlp_course/transfer_learning.html

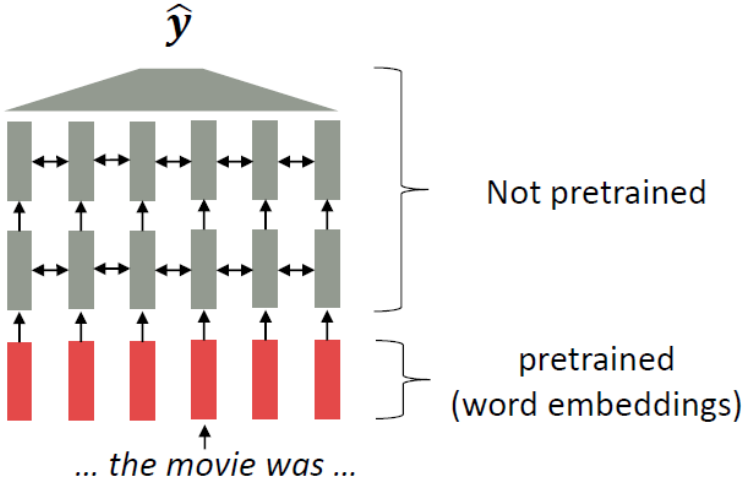
Where we were: **pretrained word embeddings**

Circa 2017:

- Start with pretrained word embeddings (no context!)
- Learn how to incorporate context in an LSTM or Transformer while training on the task.

Some issues to think about:

- The training data we have for our **downstream task** (like question answering) must be sufficient to teach all contextual aspects of language.
- Most of the parameters in our network are randomly initialized!



[Recall, *movie* gets the same word embedding, no matter what sentence it shows up in]

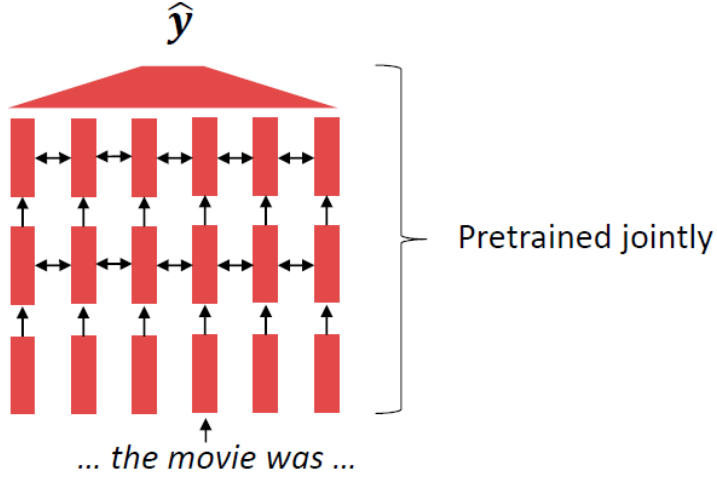
https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1214/slides/cs224n-2021-lecture09-transformers.pdf

Where we're going: **pretraining whole models**

In modern NLP:

- All (or almost all) parameters in NLP networks are initialized via **pretraining**.
- Pretraining methods hide parts of the input from the model, and train the model to reconstruct those parts.

- This has been exceptionally effective at building strong:
  - **representations of language**
  - **parameter initializations** for strong NLP models.
  - **Probability distributions** over language that we can sample from

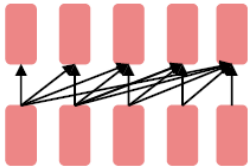


[This model has learned how to represent entire sentences through pretraining]

 https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1214/slides/cs224n-2021-lecture09-transformers.pdf

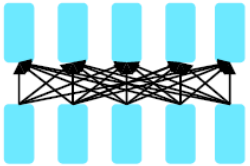
Pretraining for three types of architectures

The neural architecture influences the type of pretraining, and natural use cases.



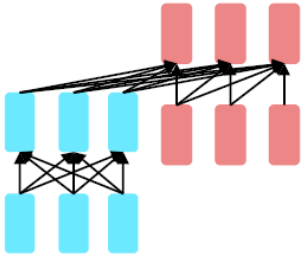
Decoders

- Language models! What we've seen so far.
- Nice to generate from; can't condition on future words



Encoders

- Gets bidirectional context – can condition on future!
- Wait, how do we pretrain them?



Encoder-Decoders

- Good parts of decoders and encoders?
- What's the best way to pretrain them?

https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1214/slides/cs224n-2021-lecture09-transformers.pdf

Transformer Models

All of these models are Transformer models

ELMo
Oct 2017
Training:
800M words
42 GPU days

GPT
June 2018
Training
800M words
240 GPU days

BERT
Oct 2018
Training
3.3B words
256 TPU days
~320–560
GPU days

GPT-2
Feb 2019
Training
40B words
~2048 TPU v3
days according to
a reddit thread

XL-Net,
ERNIE,
Grover
RoBERTa, T5
July 2019—

Source : http://web.stanford.edu/class/cs224n/slides/cs224n-2020-lecture14-contextual-representations.pdf

Pre-Training through Language Models – Decoder models

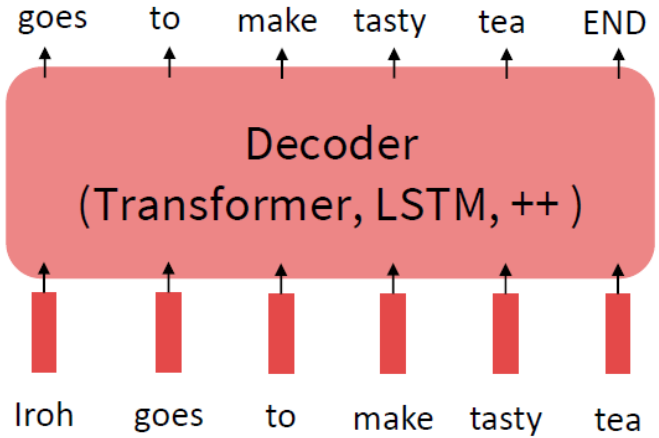
Pretraining through language modeling [Dai and Le, 2015]

Recall the **language modeling** task:

- Model $p_\theta(w_t|w_{1:t-1})$, the probability distribution over words given their past contexts.
- There's lots of data for this! (In English.)

Pretraining through language modeling:

- Train a neural network to perform language modeling on a large amount of text.
- Save the network parameters.

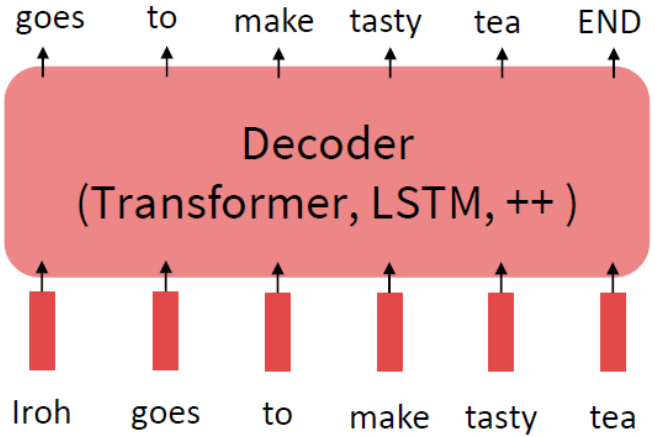


The Pretraining / Finetuning Paradigm

Pretraining can improve NLP applications by serving as parameter initialization.

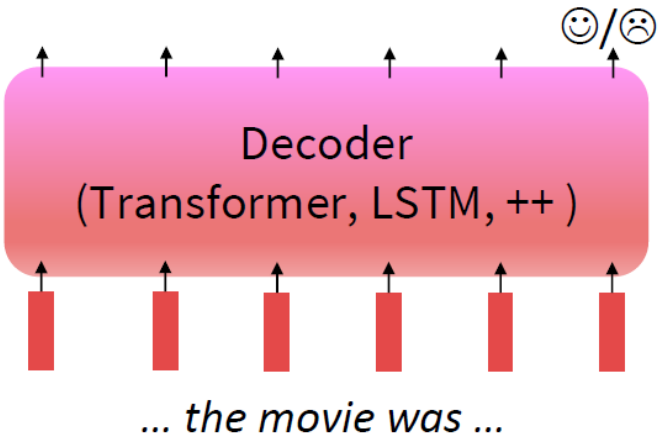
Step 1: Pretrain (on language modeling)

Lots of text; learn general things!



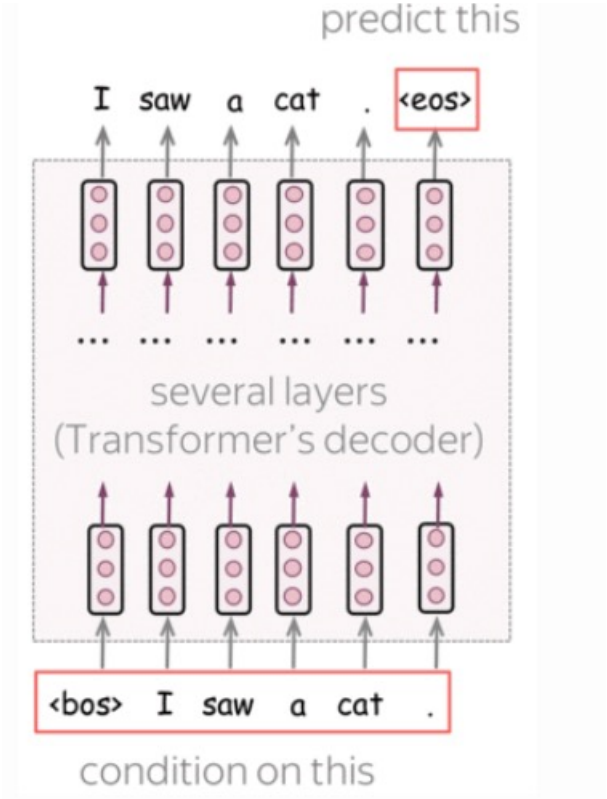
Step 2: Finetune (on your task)

Not many labels; adapt to the task!



https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1214/slides/cs224n-2021-lecture09-transformers.pdf

GPT: Generative Pre-Training for Language Understanding

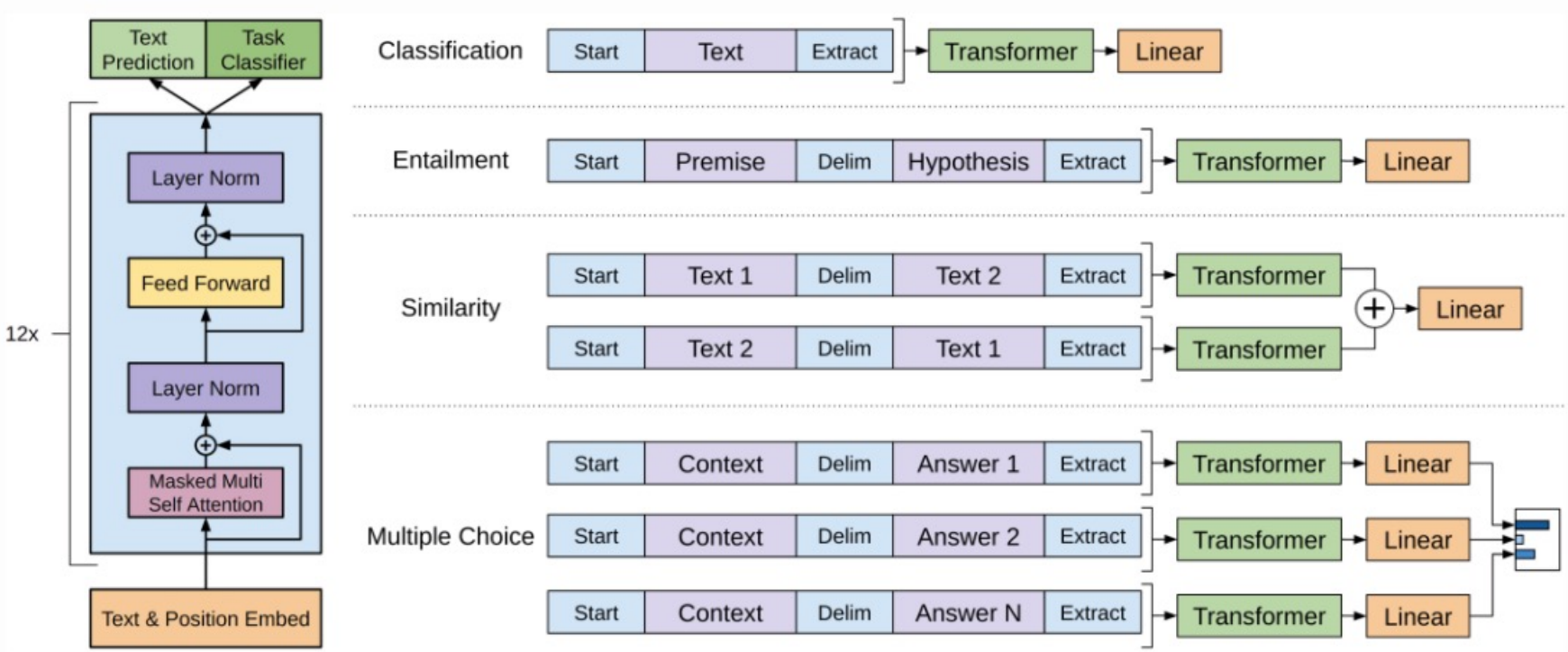


$$L_{xent} = -\sum_{t=1}^{n} \log(p(y_t|y_{<t})).$$

Decoder of Transformer Model

Source: https://lena-voita.github.io/nlp_course/transfer_learning.html

Fine Tune GPT



The figure is from the original GPT paper.

Source: https://lena-voita.github.io/nlp_course/transfer_learning.html

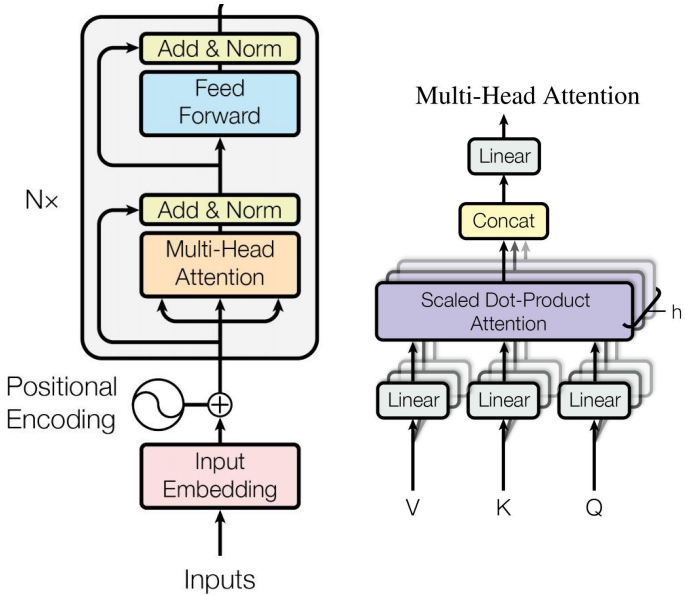
Pre-Training through Masked Language Models – Encoder models

BERT

Model Architecture

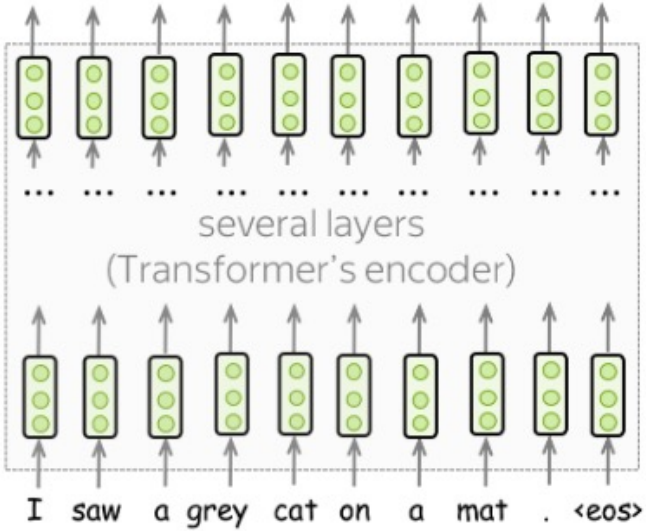
Transformer encoder

- Multi-headed self attention
  - Models context
- Feed-forward layers
  - Computes non-linear hierarchical features
- Layer norm and residuals
  - Makes training deep networks healthy
- Positional embeddings
  - Allows model to learn relative positioning



Source : http://web.stanford.edu/class/cs224n/slides/Jacob_Devlin_BERT.pdf

Model Architecture



Model architecture:

- Transformer's encoder

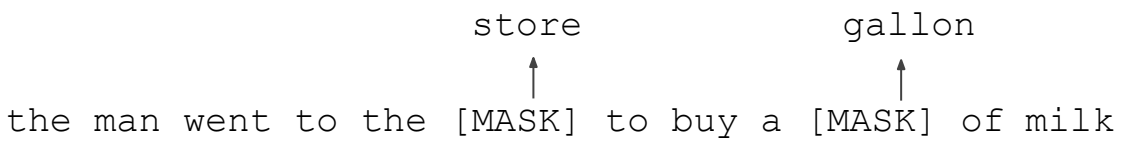
What is special about it:

- Training objectives
  - MLM: Masked language modeling
  - NSP: Next sentence prediction
- The way it is used
  - No task-specific models

Source: https://lena-voita.github.io/nlp_course/transfer_learning.html

Masked LM

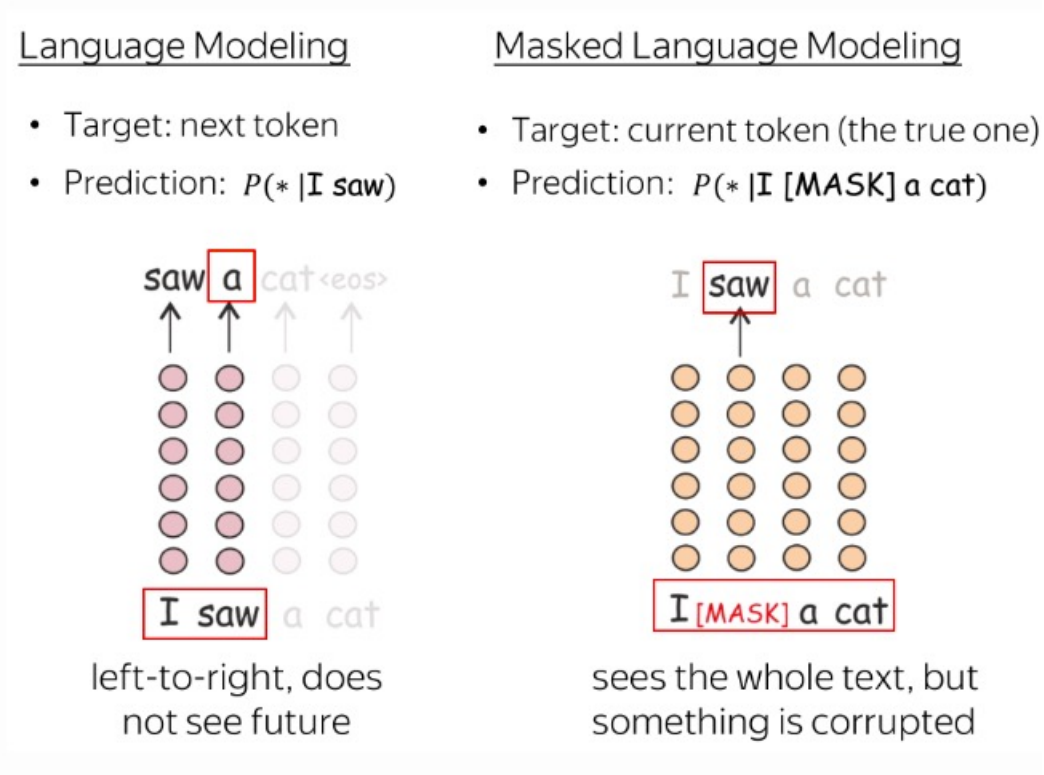
- Mask out k% of the input words, and then predict the masked words
  - We always use $k = 15\%$



- Too little masking: Too expensive to train
- Too much masking: Not enough context

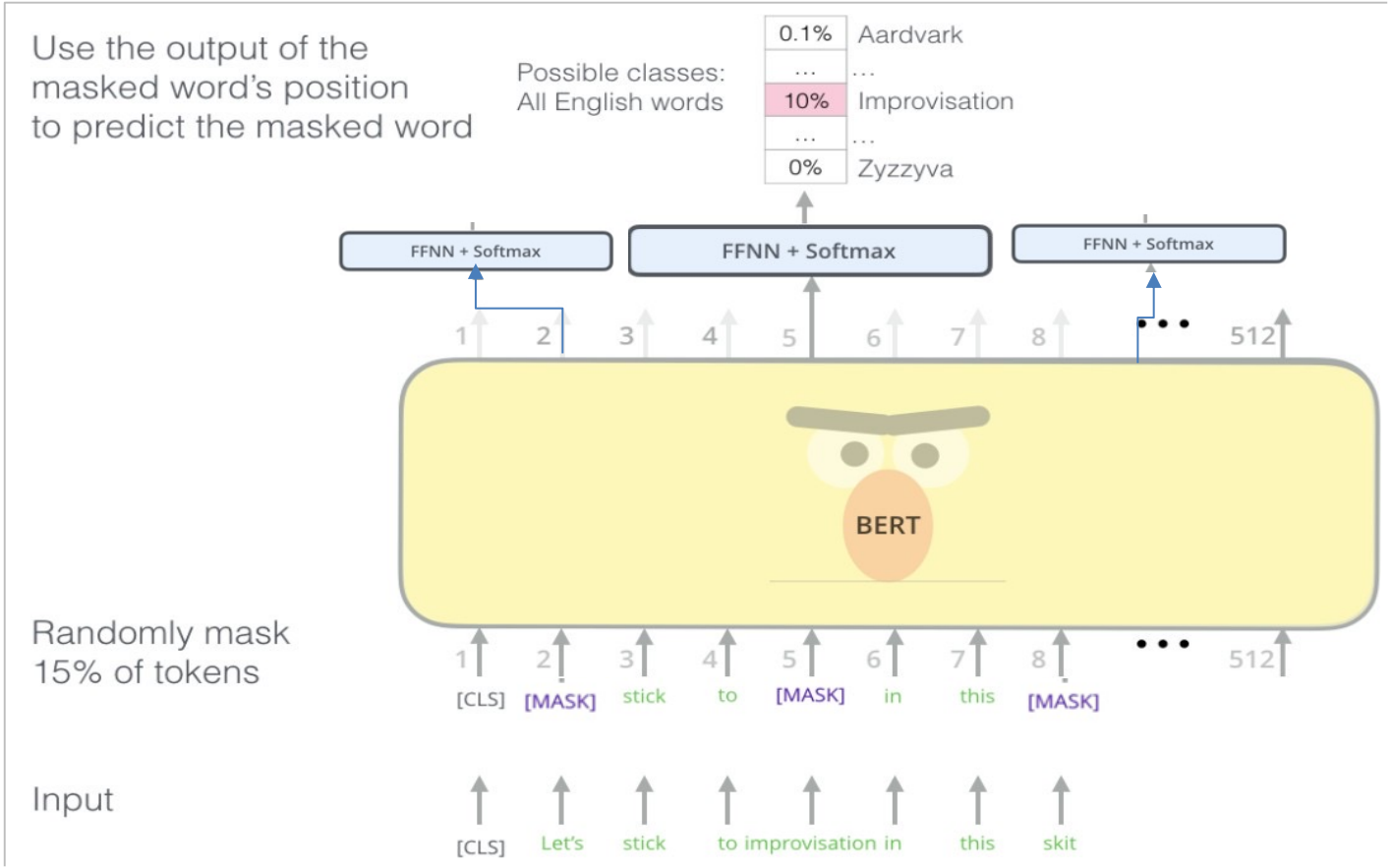
Source : http://web.stanford.edu/class/cs224n/slides/Jacob_Devlin_BERT.pdf

Masked LM



Source: https://lena-voita.github.io/nlp_course/transfer_learning.html

Training tasks (1) - Masked Language Model



Source : http://web.stanford.edu/class/cs224n/slides/Jacob_Devlin_BERT.pdf

Masked LM

- Problem: Mask token never seen at fine-tuning
- Solution: 15% of the words to predict, but don't replace with [MASK] 100% of the time. Instead:
- 80% of the time, replace with [MASK]

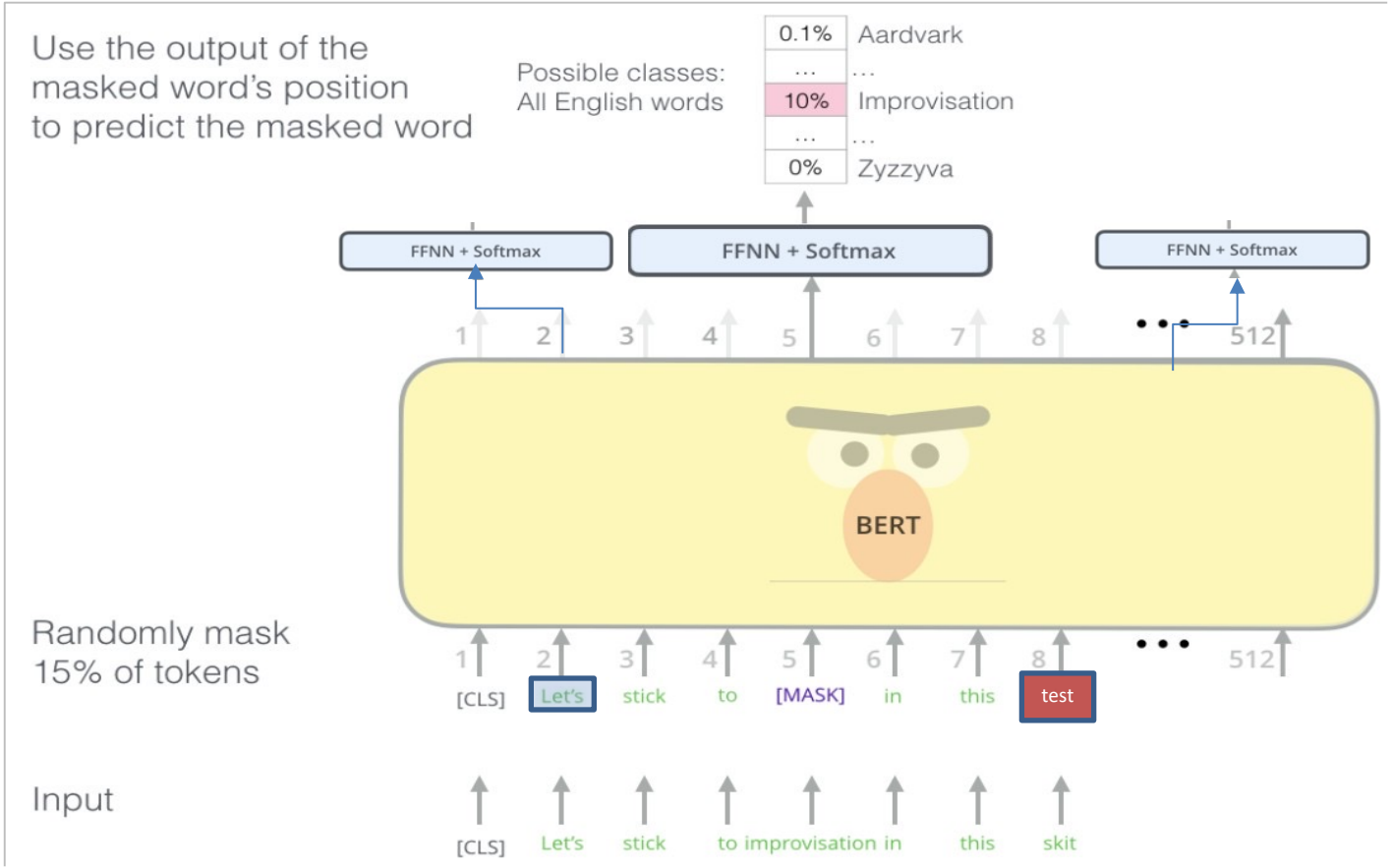
  went to the store → went to the [MASK]
- 10% of the time, replace random word

  went to the store → went to the running
- 10% of the time, keep same

  went to the store → went to the store

Source : http://web.stanford.edu/class/cs224n/slides/Jacob_Devlin_BERT.pdf

Training tasks (1) - Masked Language Model



Acknowledgement to the Figure adapted from http://jalammar.github.io/illustrated-bert/

Next Sentence Prediction

- To learn *relationships* between sentences, predict whether Sentence B is actual sentence that proceeds Sentence A, or a random sentence

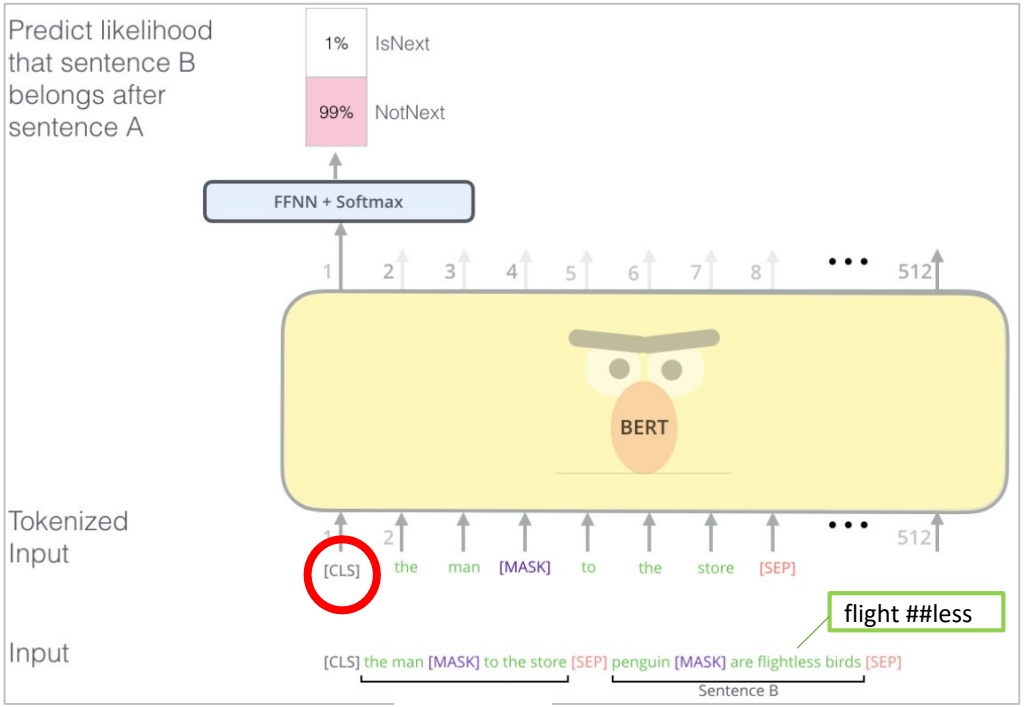
Sentence A = `The man went to the store.`
Sentence B = `He bought a gallon of milk.`
Label = `IsNextSentence`

Sentence A = `The man went to the store.`
Sentence B = `Penguins are flightless.`
Label = `NotNextSentence`

Source : http://web.stanford.edu/class/cs224n/slides/Jacob_Devlin_BERT.pdf

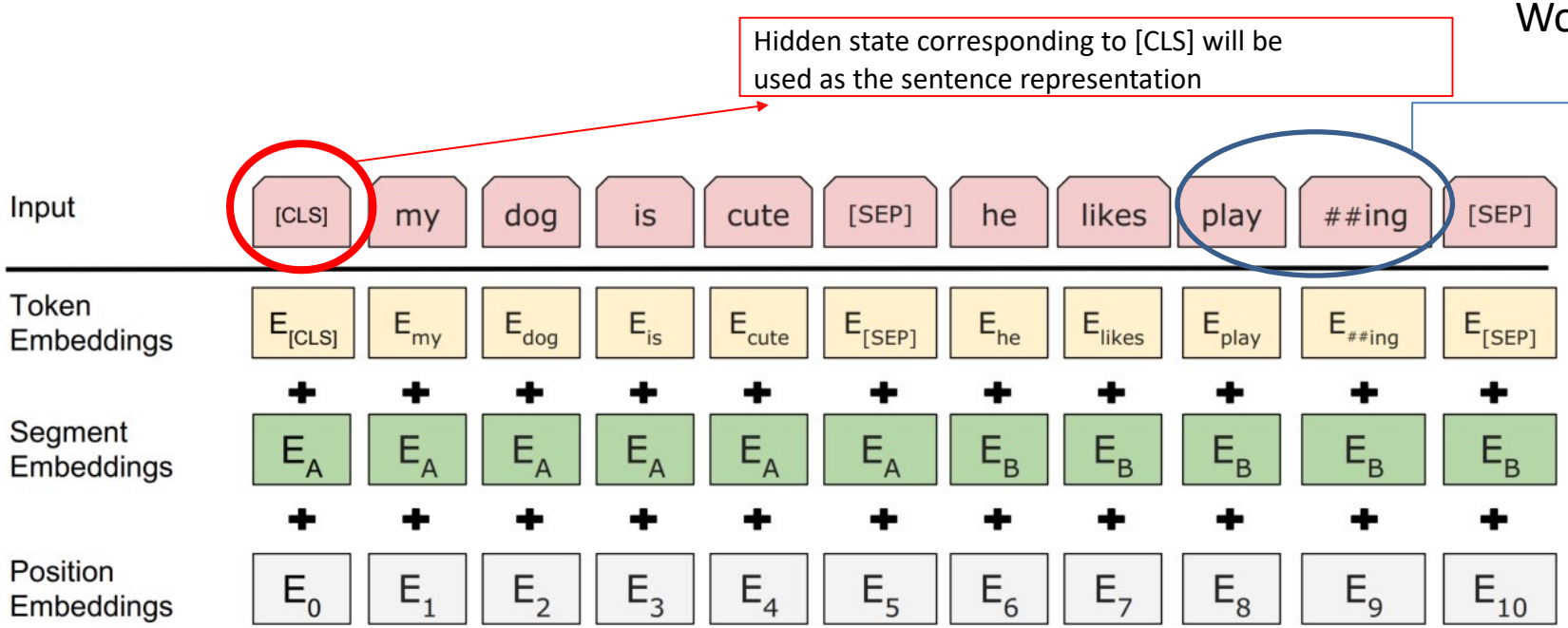
Training tasks (2) – Next Sentence Prediction

- Next sentence prediction –
  Binary classification
- For every input document as a sentence-token 2D list:
  - Randomly select a split over sentences:
    - Store the segment A
    - For 50% of the time:
      - Sample random sentence split from another document as segment B.
    - For 50% of the time:
      - Use the actual sentences as segment B.



Acknowledgement to the Figure adapted from http://jalammar.github.io/illustrated-bert/

Input Representation



- Token Embeddings: WordPiece embedding (Wu et al., 2016)
- Segment Embeddings: randomly initialized and learned; single sentence input only adds EA
- Position embeddings: randomly initialized and learned

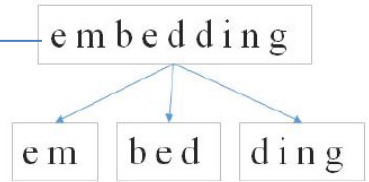


Figure in (Devlin et al., 2018)

Pre-Training datasets and details

- Training loss L is the sum of the mean masked LM likelihood and mean next sentence prediction likelihood.
- Data: Wikipedia (2.5B words) + BookCorpus (800M words)
- Batch Size: 131,072 words (1024 sequences * 128 length or 256 sequences * 512 length)
- Training Time: 1M steps (~40 epochs)
- Optimizer: AdamW, 1e-4 learning rate, linear decay
- BERT-Base: 12-layer, 768-hidden, 12-head
- BERT-Large: 24-layer, 1024-hidden, 16-head
- Trained on 4x4 or 8x8 TPU slice for 4 days

Source : http://web.stanford.edu/class/cs224n/slides/Jacob_Devlin_BERT.pdf

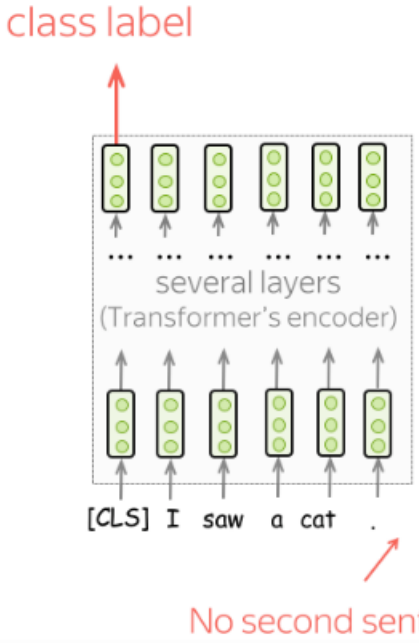
Fine-tuning with BERT

Single sentence classification

To classify individual sentences, feed the data as shown on the illustration and predict the label from the final representation of the [CLS] token.

Examples of tasks:

- SST-2 - binary sentiment classification (the one we saw in the Text Classification lecture);
- CoLA (Corpus of Linguistic Acceptability) - say whether a sentence is linguistically acceptable.



Source: https://lena-voita.github.io/nlp_course/transfer_learning.html

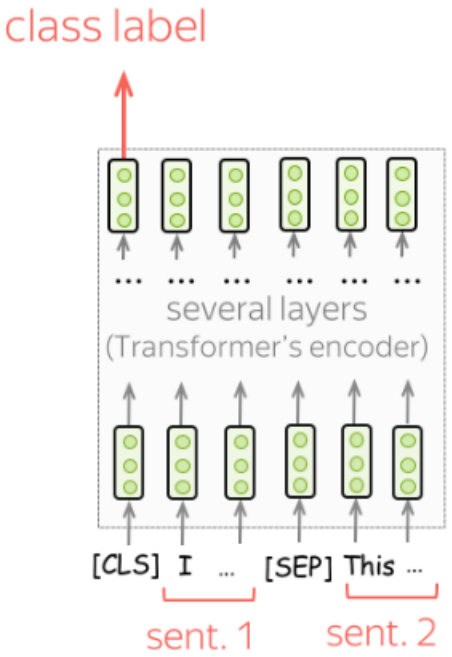
Fine-tuning with BERT

Sentence Pair Classification

To classify pairs of sentences, feed the data as you did in training. Similar to the single sentence classification, predict the label from the final representation of the [CLS] token.

Examples of tasks:

- SNLI - entailment classification. Given a pair of sentences, say if the second is an **entailment**, **contradiction**, or **neutral**);
- QQP (Quora Question Pairs) - given two questions, say if they are semantically equivalent;
- STS-B - given two sentences, return a similarity score from 1 to 5.



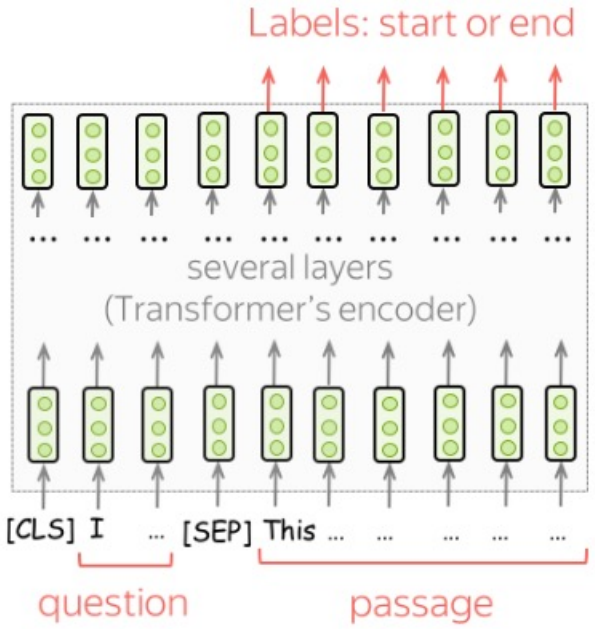
Source: https://lena-voita.github.io/nlp_course/transfer_learning.html

Fine-tuning with BERT

Question Answering

For QA, the BERT authors used only one dataset, SQuAD v1.1. In this task, you are given a text passage and a question. The answer to this question is always a part of the passage, and the task is to find the correct segment of the passage.

To use BERT for this task, feed a question and a passage as shown in the illustration. Then, for each token in the passage use final BERT representations to predict whether this token is the start or the end of the correct segment.

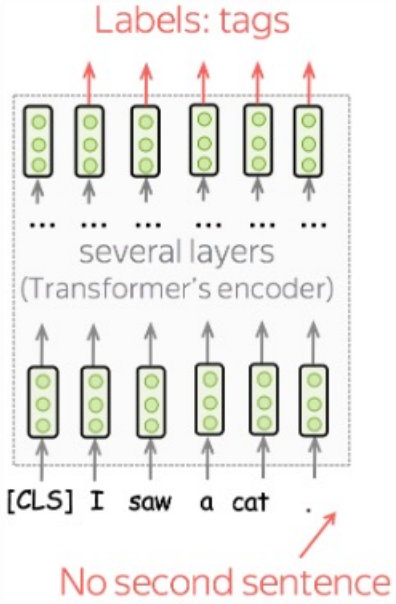


Source: https://lena-voita.github.io/nlp_course/transfer_learning.html

Fine-tuning with BERT

Single sentence tagging

In tagging tasks, you have to predict tags for each token. For example, in Named Entity Recognition (NER), you have to predict if a word is a named entity and its type (e.g., **location**, **person**, etc).

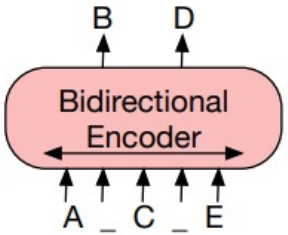


Source: https://lena-voita.github.io/nlp_course/transfer_learning.html

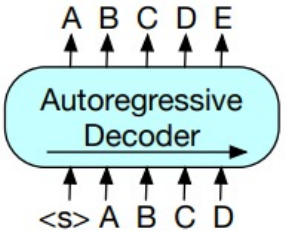
Pre-Training through Encoder-Decoder Models

BART

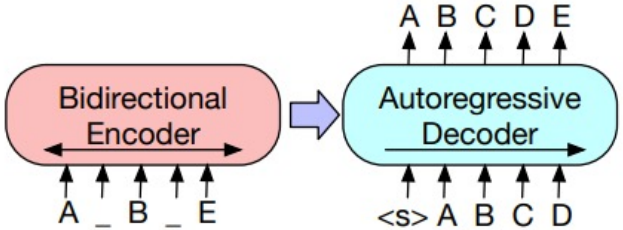
BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension



(a) BERT: Random tokens are replaced with masks, and the document is encoded bidirectionally. Missing tokens are predicted independently, so BERT cannot easily be used for generation.



(b) GPT: Tokens are predicted auto-regressively, meaning GPT can be used for generation. However words can only condition on leftward context, so it cannot learn bidirectional interactions.



(c) BART: Inputs to the encoder need not be aligned with decoder outputs, allowing arbitary noise transformations. Here, a document has been corrupted by replacing spans of text with a mask symbols. The corrupted document (left) is encoded with a bidirectional model, and then the likelihood of the original document (right) is calculated with an autoregressive decoder. For fine-tuning, an uncorrupted document is input to both the encoder and decoder, and we use representations from the final hidden state of the decoder.

https://arxiv.org/abs/1910.13461

BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension

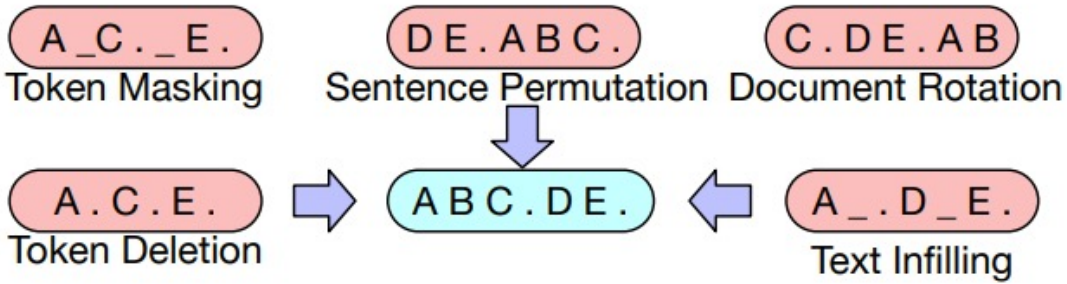
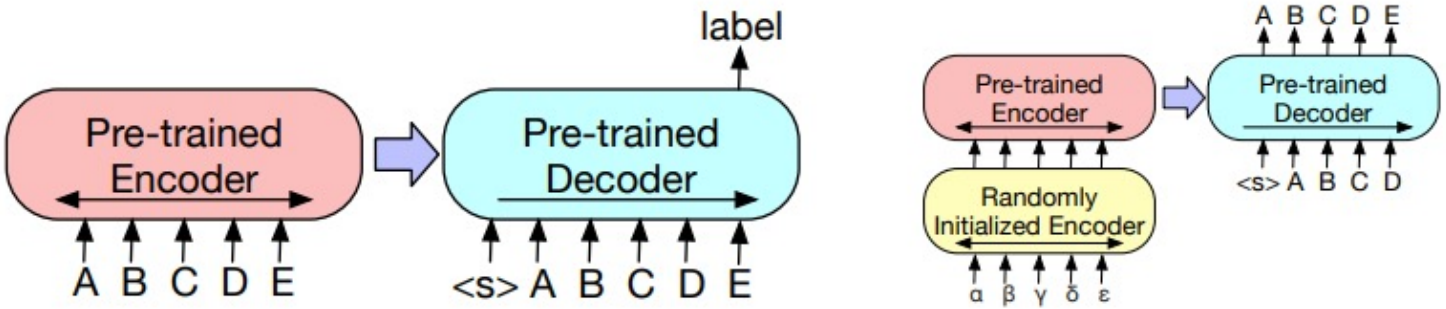


Figure 2: Transformations for noising the input that we experiment with. These transformations can be composed.

https://arxiv.org/abs/1910.13461

BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension



(a) To use BART for classification problems, the same input is fed into the encoder and decoder, and the representation from the final output is used.

(b) For machine translation, we learn a small additional encoder that replaces the word embeddings in BART. The new encoder can use a disjoint vocabulary.

Figure 3: Fine tuning BART for classification and translation.

https://arxiv.org/abs/1910.13461

Transformer Models

Encoder Models

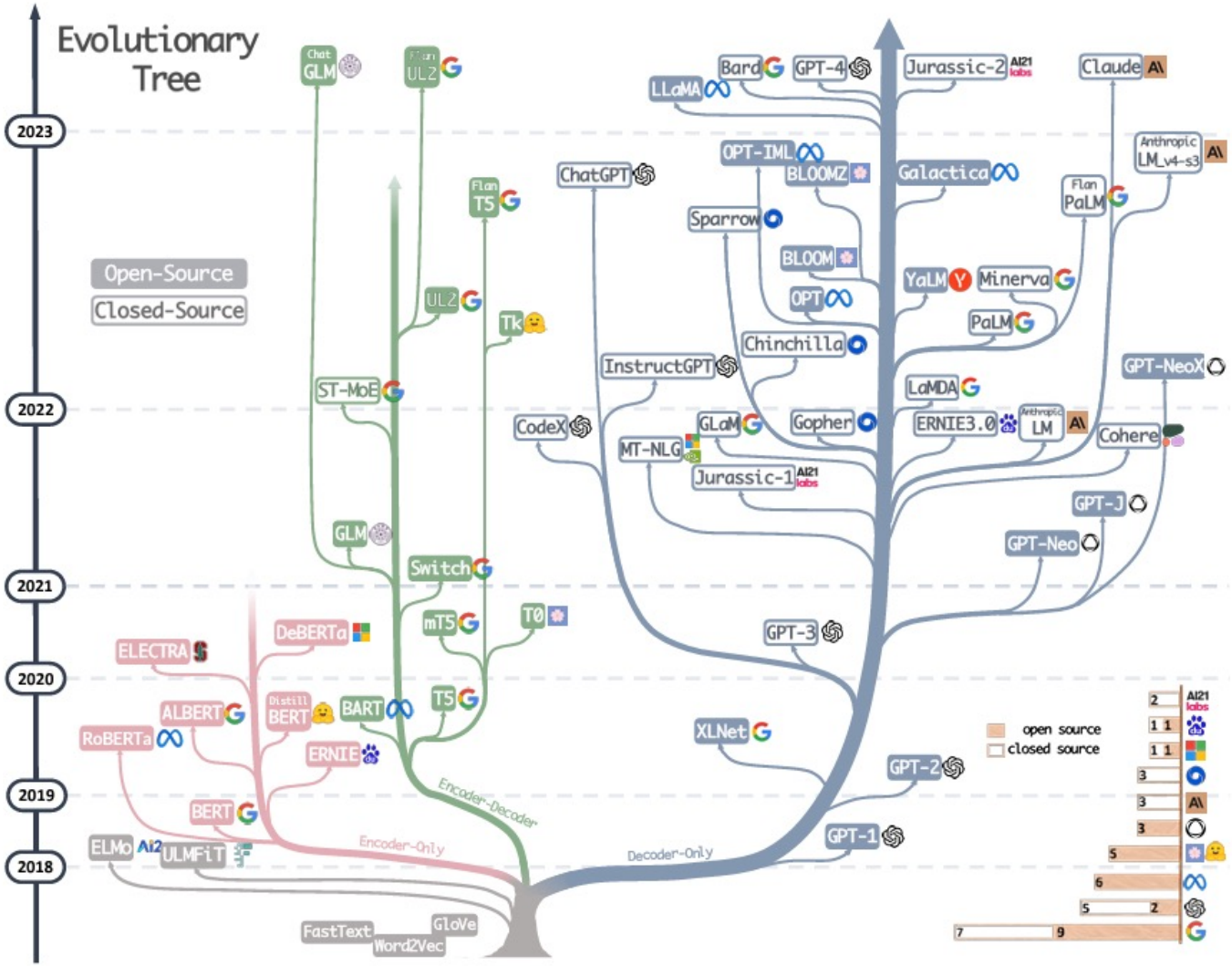
(ALBERT, **BERT**, XLNET, DistilBERT, ELECTRA, RoBERTa) - Sentence Classification, named entity recognition, extractive question answering, Part of Speech tagging, Topic Modelling

Decoder Models

CTRL, GPT, **GPT-2**, Transformer XL -Text Generation

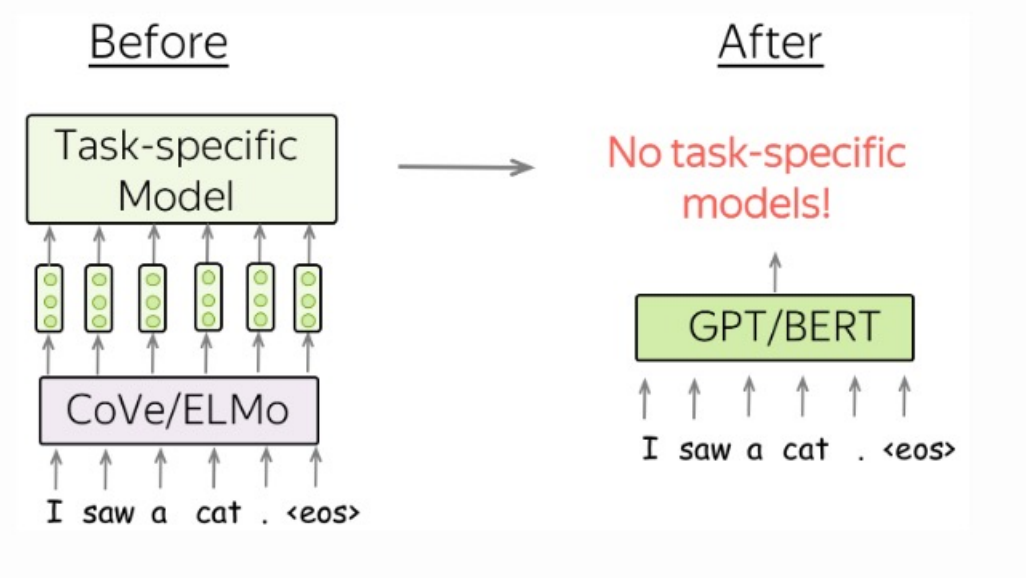
Encoder-Decoder Models

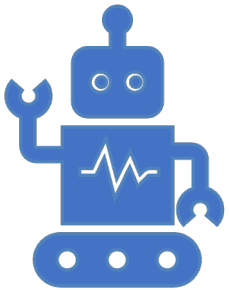
BART, T5, Marian, mBART - Summarization, translation, generative question answering



https://arxiv.org/abs/2304.13712

Current State of NLP

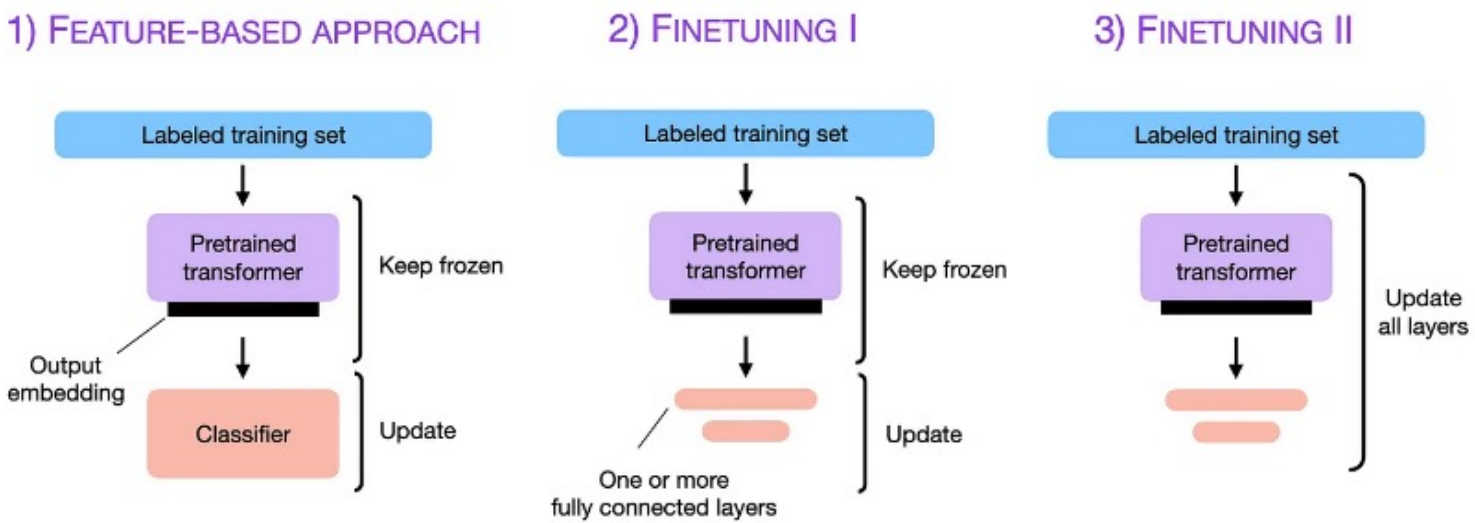




Source: https://lena-voita.github.io/nlp_course/transfer_learning.html

Fine-Tuning

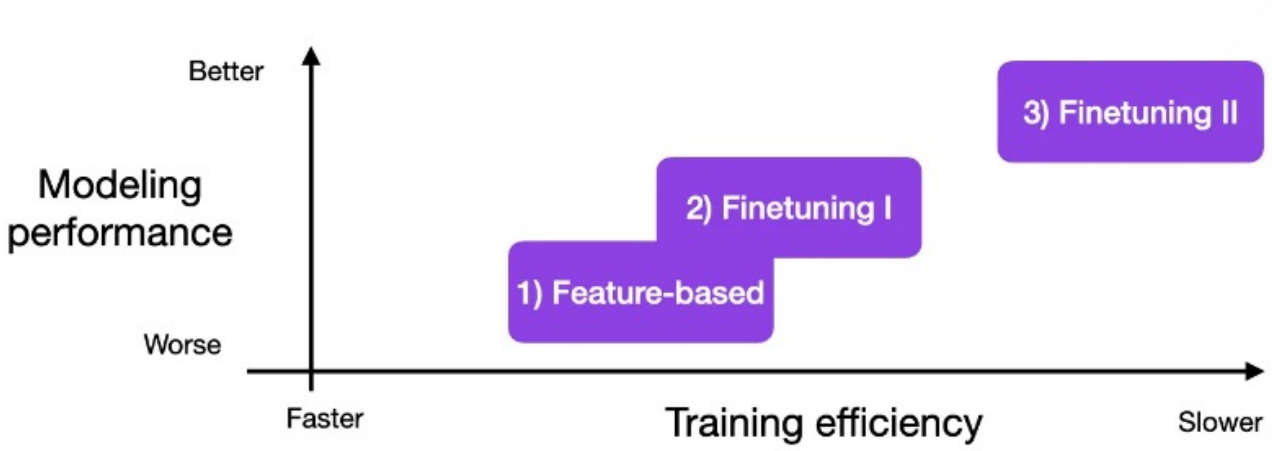
Fine-Tuning - Conventional



The 3 conventional feature-based and finetuning approaches.

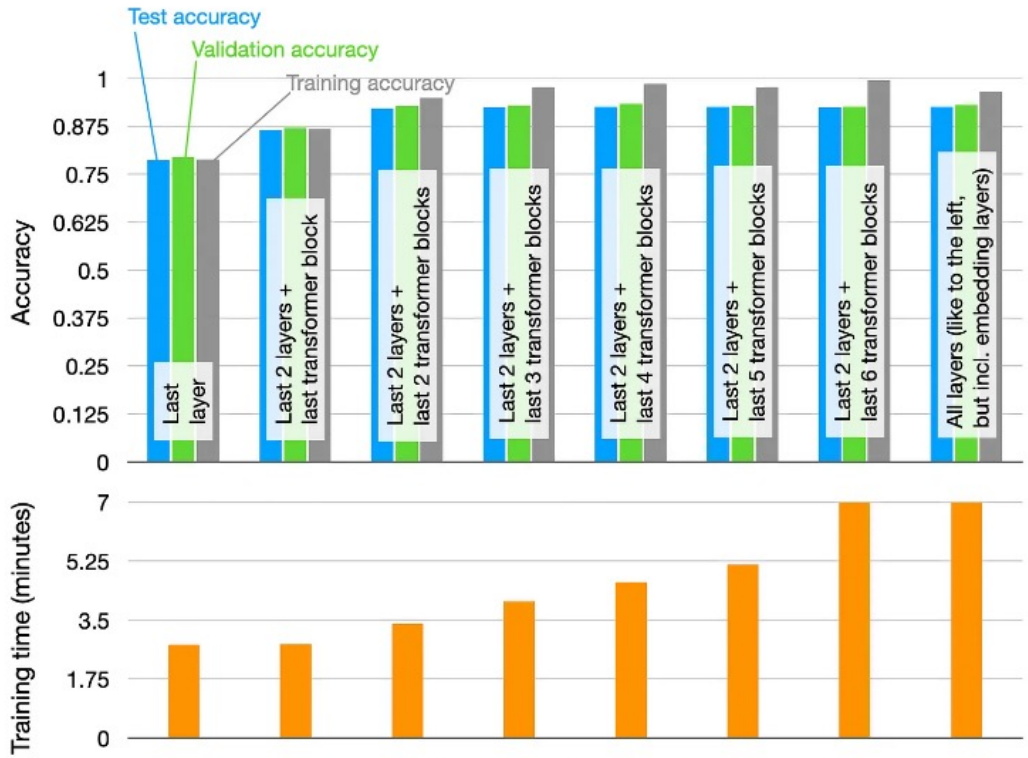
https://magazine.sebastianraschka.com/p/finetuning-large-language-models?utm_source=profile&utm_medium=reader2

Fine-Tuning - Conventional



https://magazine.sebastianraschka.com/p/finetuning-large-language-models?utm_source=profile&utm_medium=reader2

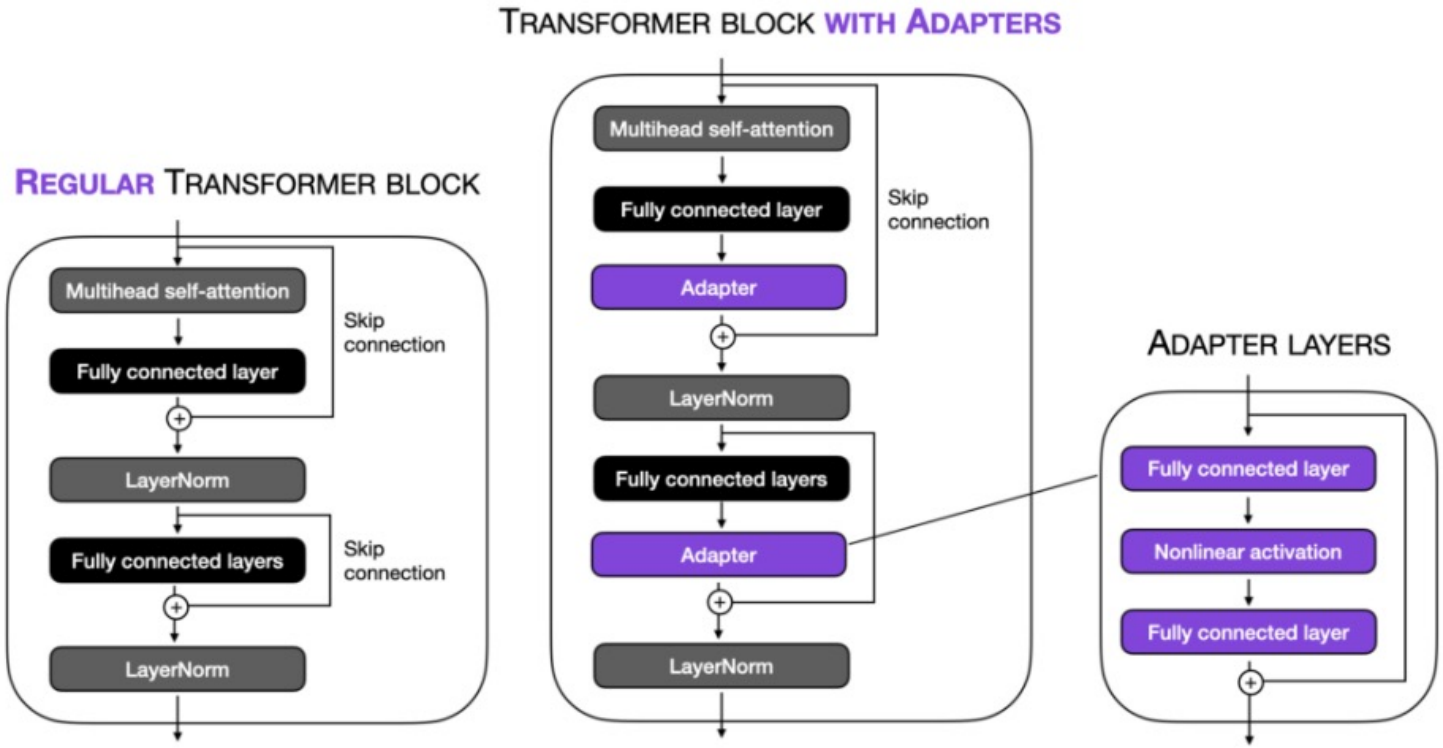
Fine-Tuning - Conventional



Performance of pretrained DistilBERT models finetuned on the IMDB Movie review dataset. The code can be found here on GitHub.

https://magazine.sebastianraschka.com/p/finetuning-large-language-models?utm_source=profile&utm_medium=reader2

Parameter Efficient Fine Tuning - Adapters



https://magazine.sebastianraschka.com/p/finetuning-large-language-models?utm_source=profile&utm_medium=reader2

Tokenization

Tokenization Overview

- What is tokenization?
  - Process of breaking down text into individual tokens (words, sub-words, or characters)
  - Fundamental step in natural language processing (NLP) and machine learning (ML)
- Importance of tokenization in NLP
  - Simplifies text processing and analysis
  - Facilitates text representation for ML algorithms
  - Affects the quality of downstream NLP tasks (e.g., sentiment analysis, translation)
- Common tokenization techniques
  - Whitespace-based tokenization
  - Character-based tokenization
  - Rule-based tokenization (e.g., using regular expressions)
  - **Sub-word tokenization (e.g., BPE, WordPiece, SentencePiece)**

Byte Pair Encoding (BPE) - Introduction

- What is Byte Pair Encoding (BPE)?
  - A data compression algorithm
  - Originally designed for lossless compression of binary data
  - Adapted for use in NLP as a sub-word tokenization technique

- BPE in NLP
  - Learns a fixed-size sub-word vocabulary based on the frequency of character pairs
  - Iteratively merges the most frequent pairs to create new sub-words
  - Enables efficient representation of large and diverse text corpora

- Why use BPE for tokenization?
  - Reduces the impact of out-of-vocabulary words
  - Balances computational efficiency and linguistic information capture
  - Provides a flexible and scalable approach to tokenization
  - Widely used in modern NLP models like BERT and GPT

BPE - Algorithm

BPE algorithm steps:

1. Initialize vocabulary with individual characters and their frequencies in the corpus

2. Iteratively merge the most frequent character pairs and update their frequencies

3. Stop when the desired vocabulary size is reached or no more frequent pairs can be merged

BPE - How It Works - Training

```
("hug", 10), ("pug", 5), ("pun", 12), ("bun", 4), ("hugs", 5)
```

```
Vocabulary: ["b", "g", "h", "n", "p", "s", "u",
```

```
("h" "u" "g", 10), ("p" "u" "g", 5), ("p" "u" "n", 12), ("b" "u" "n", 4), ("h" "u" "g" "s", 5)
```

```
Vocabulary: ["b", "g", "h", "n", "p", "s", "u", "ug"]
Corpus: ("h" "ug", 10), ("p" "ug", 5), ("p" "u" "n", 12), ("b" "u" "n", 4), ("h" "ug" "s", 5)
```

```
Vocabulary: ["b", "g", "h", "n", "p", "s", "u", "ug", "un"]
Corpus: ("h" "ug", 10), ("p" "ug", 5), ("p" "un", 12), ("b" "un", 4), ("h" "ug" "s", 5)
```

```
Vocabulary: ["b", "g", "h", "n", "p", "s", "u", "ug", "un", "hug"]
Corpus: ("hug", 10), ("p" "ug", 5), ("p" "un", 12), ("b" "un", 4), ("hug" "s", 5)
```

Source: https://huggingface.co/learn/nlp-course/chapter6/5?fw=pt

BPE - Test

```
Vocabulary: ["b", "g", "h", "n", "p", "s", "u", "ug", "un", "hug"]
```

Merge Rules

```
("u", "g") -> "ug"
("u", "n") -> "un"
("h", "ug") -> "hug"
```

“bug” -> [“b”, “ug”]
“mug” -> [“[UNK]”, “ug”]

- It is very rare to get <UNK> token with BPE
- Most tokenizers include all the possible bytes in the initial vocabulary

Source: https://huggingface.co/learn/nlp-course/chapter6/5?fw=pt

Reading for Exam I

Recommended Readings

All the slides and notebooks covered till today

Word Embeddings: https://lena-voita.github.io/nlp_course/word_embeddings.html

Text Classification: https://lena-voita.github.io/nlp_course/text_classification.html
Start from Text classification with NN (skip CNNs)

Language Modeling: https://lena-voita.github.io/nlp_course/language_modeling.html
Skip - CNNs

Seq2Seq and Attention: https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html

Transfer Learning: https://lena-voita.github.io/nlp_course/transfer_learning.html
Transformer: https://jalammar.github.io/illustrated-transformer/

Training NN:
https://d2l.ai/chapter_multilayer-perceptrons/numerical-stability-and-init.html
https://d2l.ai/chapter_multilayer-perceptrons/dropout.html