Attention Mechanism, Transformers, BERT, and GPT

Deep Learning (ENGG*6600*01)

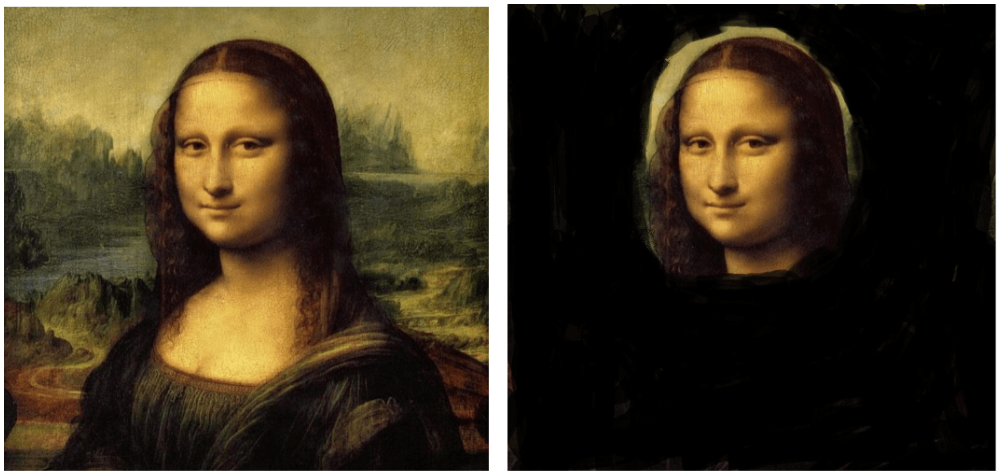
School of Engineering,
University of Guelph, ON, Canada

Course Instructor: Benyamin Ghojogh
Summer 2023

Attention Mechanism

Introduction

- When looking at a **scene or picture**, our visual system, so as a machine learning model [1], focuses on or attends to some specific parts of the scene/image with more information and importance and ignores the less informative or less important parts.
- Moreover, when reading a **text**, especially when we want to try fast reading, one technique is skimming [2] in which our visual system or a model skims the data with high pacing and only attends to more informative words of sentences [3].
- The concept of attention can be modeled in machine learning where attention is a simple **weighting** of data. In the attention mechanism, the more informative or more important parts of data are given larger weights for the sake of more attention.



(a)

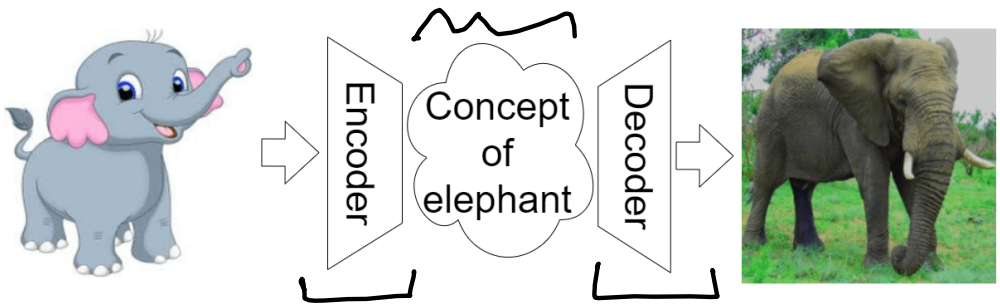
Mona Lisa is a portrait painted by Leonardo da Vinci.

Mona Lisa is a portrait painted by Leonardo da Vinci.

(b)

Autoencoder and the Context Vector

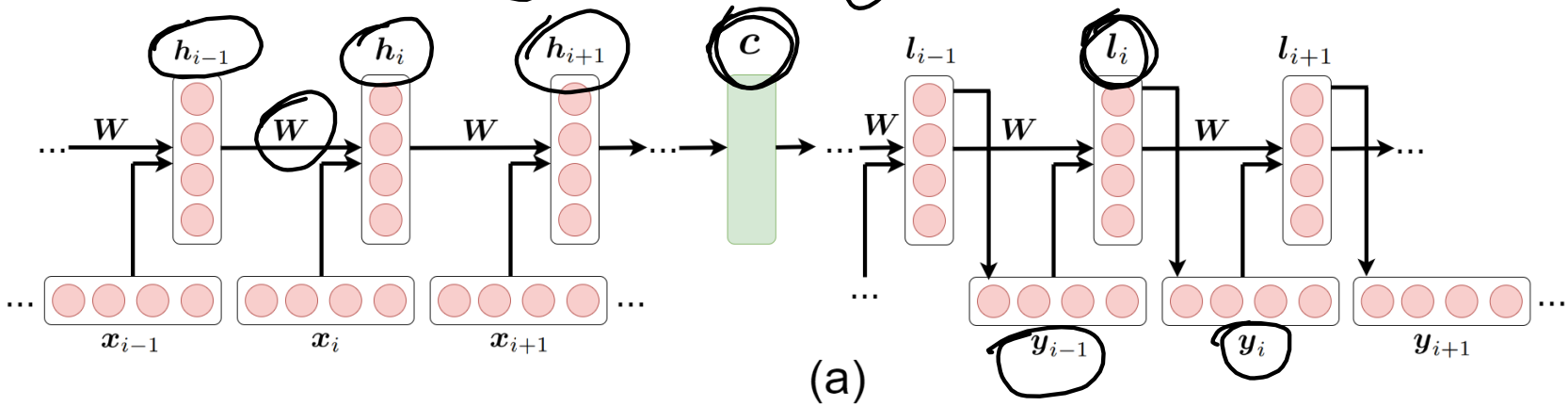
- Consider an autoencoder with encoder and decoder parts where the encoder gets an input and converts it to a *context vector* and the decoder gets the context vector and converts to an output.
- The output is related to the input through the context vector in the so-called **hidden space**.
- An autoencoder has an **encoder** (left part of autoencoder), a **hidden space** (in the middle), and a **decoder** (right part of autoencoder).
- For example, the input can be a sentence or a word in English and the output is the same sentence or word but in French. Assume the word "elephant" in English is fed to the encoder and the word "l'éléphant" in French is output. The context vector models the **concept of elephant** which also exists in the mind of human when thinking to elephant. This context is abstract in mind and can be referred to any fat, thin, huge, or small elephant [4].
- Another example for transformer is transforming a **cartoon image** of elephant to picture of a **real elephant**.
- As the autoencoder is transforming data from a domain to a hidden space and then to another domain, it can be used for **domain transformation** [5], **domain adaptation** [6], and **domain generalization** [7].



The Sequence-to-Sequence Model

- The context is modeled as a vector in the hidden space. Let the **context vector** be denoted by $\boldsymbol{c} \in \mathbb{R}^p$ in the p-dimensional hidden space.
- Consider a sequence of ordered tokens, e.g., a sequence of words which make a sentence. We want to transform this sequence to another related sequence. For example, we want to take a sentence in English and translate it to the same sentence in French. This model which **transforms a sequence to another related sequence** is named the **sequence-to-sequence model** [8].
- Suppose the number of words in a document or considered sentence be n. Let the ordered **input tokens or words** of the sequence be denoted by $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{i-1}, \boldsymbol{x}_i, \ldots, \boldsymbol{x}_n\}$ and the **output sequence** be denoted by $\{\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_{i-1}, \boldsymbol{y}_i, \ldots, \boldsymbol{y}_n\}$. As this figure illustrates, there exist **latent vectors**, denoted by $\{\boldsymbol{l}_i\}_{i=1}^n$, in the decoder part for every word.
- In the sequence-to-sequence model, the probability of generation of the i-th word conditioning on all the previous words is determined by a function $g(.)$ whose inputs are the immediate previous word $\boldsymbol{y}_{i-1}$, the i-th latent vector $\boldsymbol{l}_i$, and the context vector $\boldsymbol{c}$:

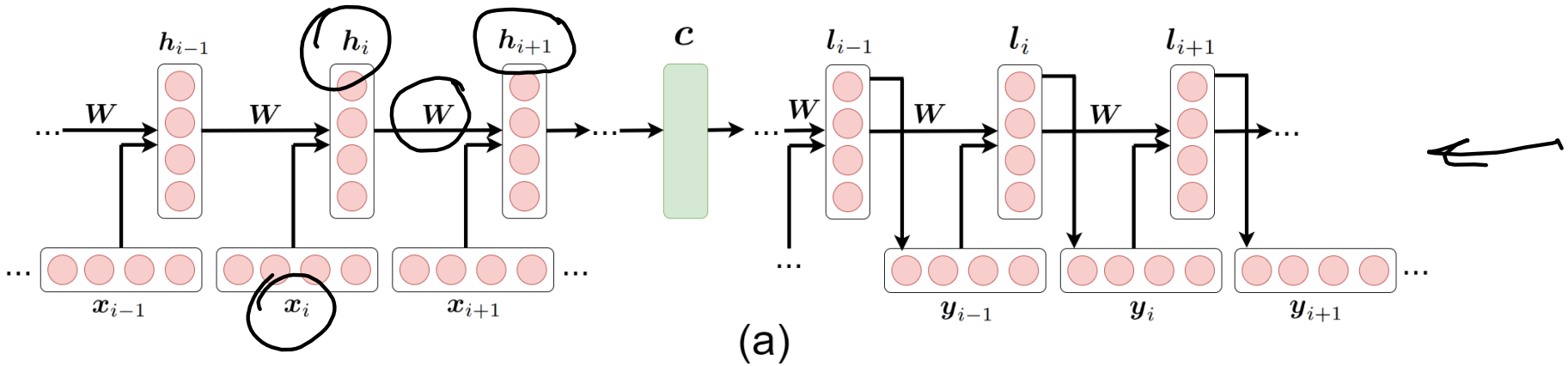
$$\mathbb{P}(\boldsymbol{y}_i | \boldsymbol{y}_1, \ldots, \boldsymbol{y}_{i-1}) = g(\boldsymbol{y}_{i-1}, \boldsymbol{l}_i, \boldsymbol{c}). \tag{1}$$



(a)

The Sequence-to-Sequence Model

- In the sequence-to-sequence model, every word $\boldsymbol{x}_i$ produces a hidden vector $\boldsymbol{h}_i$ in the encoder part of the autoencoder. The hidden vector of every word, $\boldsymbol{h}_i$, is fed to the next hidden vector, $\boldsymbol{h}_{i+1}$, by a projection matrix $\boldsymbol{W}$.
- In this model, for the whole sequence, there is **only one context vector $\boldsymbol{c}$** which is equal to the last hidden vector of the encoder, i.e., $\boldsymbol{c} = \boldsymbol{h}_n$.
- Note that the encoder and decoder in the sequence-to-sequence model can be any sequential model such as RNN [9] or LSTM [10].



(a)

The Sequence-to-Sequence Model with Attention

- In the sequence-to-sequence model **with attention** [11, 12], the probability of generation of the i-th word is determined as [11]:

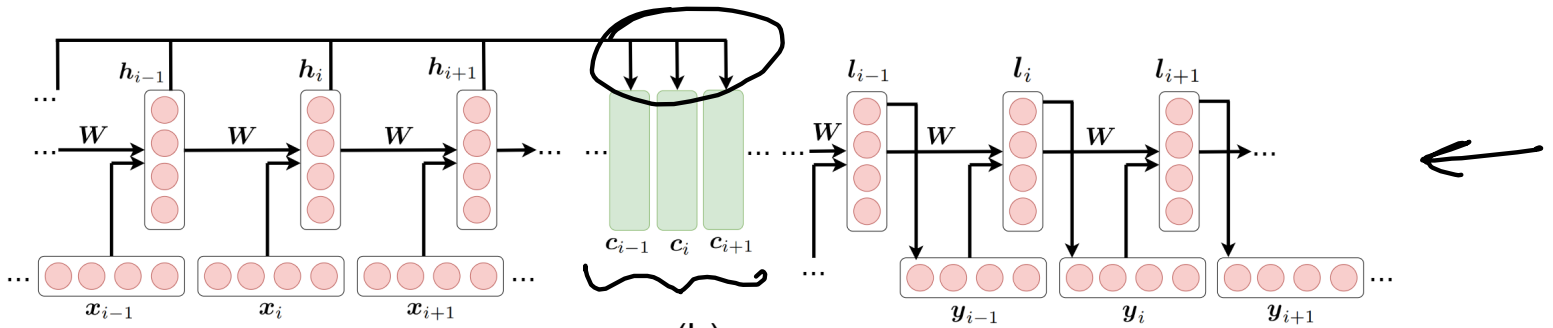
$$\mathbb{P}(\boldsymbol{y}_i|\boldsymbol{y}_1,\ldots,\boldsymbol{y}_{i-1}) = g(\boldsymbol{y}_{i-1},\boldsymbol{l}_i,\boldsymbol{c}_i). \tag{2}$$

- In this model, in contrast to the sequence-to-sequence model which has only one context vector for the whole sequence, this model has **a context vector for every word**.
- The context vector of every word is a linear combination, or **weighted sum**, of all the hidden vectors; hence, the i-th context vector is:

$$\boldsymbol{c}_i = \sum_{j=1}^{n} a_{ij}\boldsymbol{h}_j, \tag{3}$$

where $a_{ij} \geq 0$ is the weight of $\boldsymbol{h}_j$ for the i-th context vector.

- This weighted sum for a specific word in the sequence determines **which words in the sequence have more effect on that word**. In other words, it determines which words this specific word "**attends**" to more. This notion of weighted impact, to see which parts have more impact, is called "**attention**".



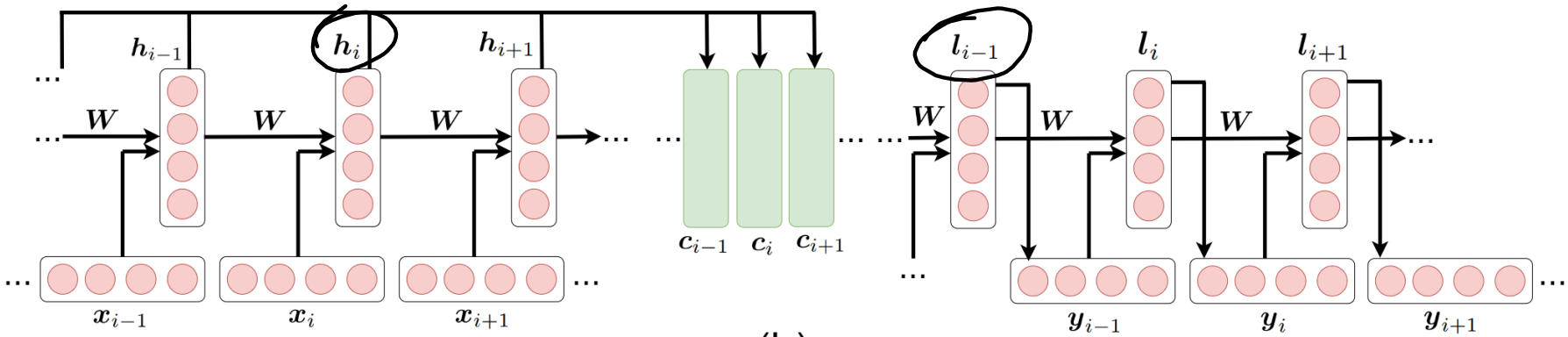
(b)

The Sequence-to-Sequence Model with Attention

- The original idea of arithmetic linear combination of vectors for the purpose of word embedding, similar to Eq. (3), was in the Word2Vec method [13, 14].
- The sequence-to-sequence model with attention considers a notion of **similarity** between the latent vector $\boldsymbol{l}_{i-1}$ of the decoder and the hidden vector $\boldsymbol{h}_j$ of the encoder [8]:

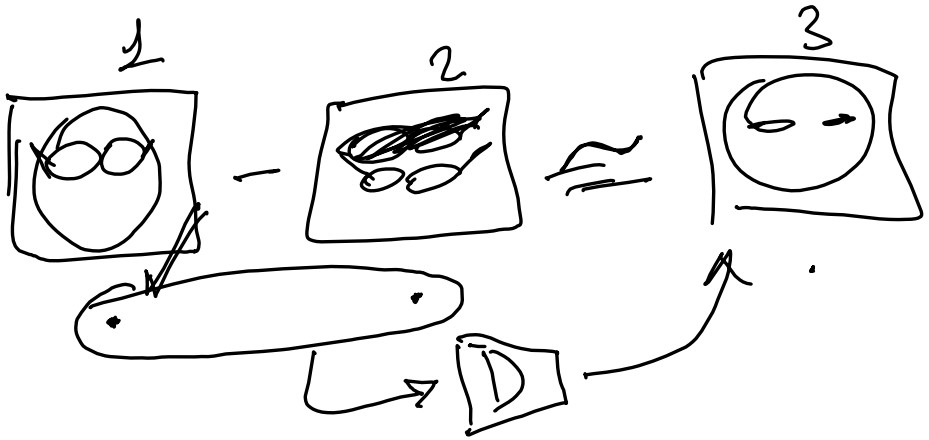
$$\mathbb{R} \ni s_{ij} := \text{similarity}(\boldsymbol{l}_{i-1}, \boldsymbol{h}_j). \tag{4}$$

- Intuition: The output word $\boldsymbol{y}_i$ depends on the previous latent vector $\boldsymbol{l}_{i-1}$ and the hidden vector $\boldsymbol{h}_j$ depends on the input word $\boldsymbol{x}_j$. Hence, this similarity score relates to the **impact of the input $\boldsymbol{x}_j$ on the output $\boldsymbol{y}_i$**. In this way, the **score s_{ij} shows the impact of the j-th word to generate the i-th word** in the sequence. This similarity notion can be a neural network learned by backpropagation [15].



(b)

king − man + woman ≈ queen

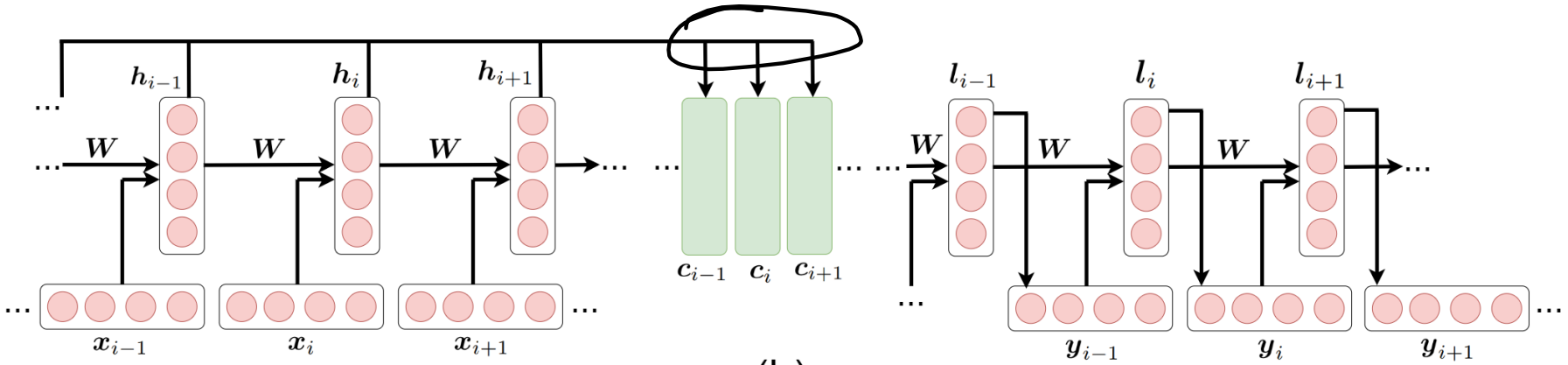


The Sequence-to-Sequence Model with Attention

- To make this score a **probability**, these scores should sum to one; hence, we make its softmax form as [11]:

$$\mathbb{R} \ni a_{ij} := \frac{e^{s_{ij}}}{\sum_{k=1}^{n} e^{s_{ik}}}. \tag{5}$$

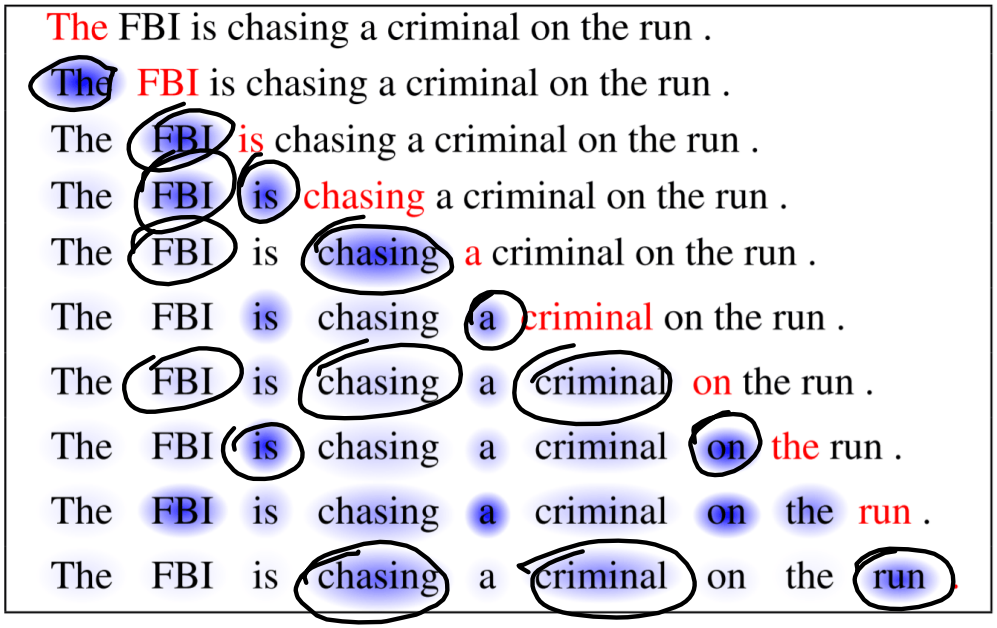
  In this way, the score vector $[a_{i1}, a_{i2}, \ldots, a_{in}]^\top$ behaves as a discrete probability distribution.
- Therefore, In Eq. (3), $\boldsymbol{c}_i = \sum_{j=1}^{n} a_{ij} \boldsymbol{h}_j$, the weights sum to one and the weights with higher values attend more to their corresponding hidden vectors.



(b)

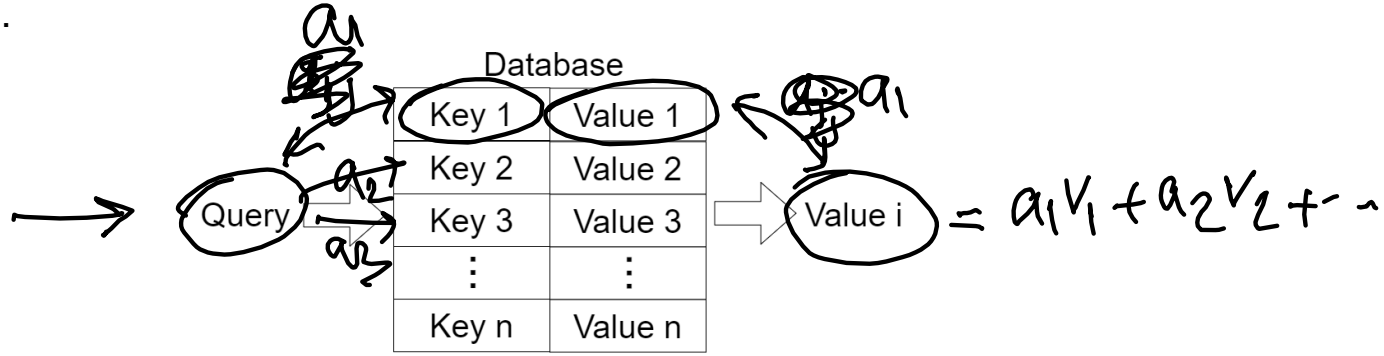
The Need for Composite Embedding

- Many of the previous methods for NLP, such as word2vec [13, 14, 16, 17] and GloVe [18], used to learn **a representation for every word**.
- However, for understanding how the words **relate to each other**, we can have a **composite embedding** where the compositions of words also have some embedding representation [19].
- For example, this figure shows a sentence which highlights the **relation of words**. This figure shows, when reading a word in a sentence, which previous words in the sentence we remember more. This relation of words shows that we need to have a composite embedding for natural language embedding.



Query-Retrieval Modeling

- Consider a **database** with keys and values where a query is searched through the keys to retrieve a value [20].



- We can generalize this hard definition of query-retrieval to a **soft query-retrieval** where several keys, rather than only one key, can be corresponded to the query. For this, we calculate the **similarity of the query with all the keys** to see which keys are more similar to the query:

$$\text{attention}(\boldsymbol{q}, \{\boldsymbol{k}_i\}_{i=1}^n, \{\boldsymbol{v}_i\}_{i=1}^n) := \sum_{i=1}^n a_i \boldsymbol{v}_i, \tag{6}$$

where:

$$\mathbb{R} \ni s_i := \text{similarity}(\boldsymbol{q}, \boldsymbol{k}_i), \tag{7}$$

$$\mathbb{R} \ni a_i := \text{softmax}(s_i) = \frac{e^{s_i}}{\sum_{k=1}^n e^{s_k}}, \tag{8}$$

and $\boldsymbol{q}$, $\{\boldsymbol{k}_i\}_{i=1}^n$, and $\{\boldsymbol{v}_i\}_{i=1}^n$ denote the query, keys, and values, respectively.

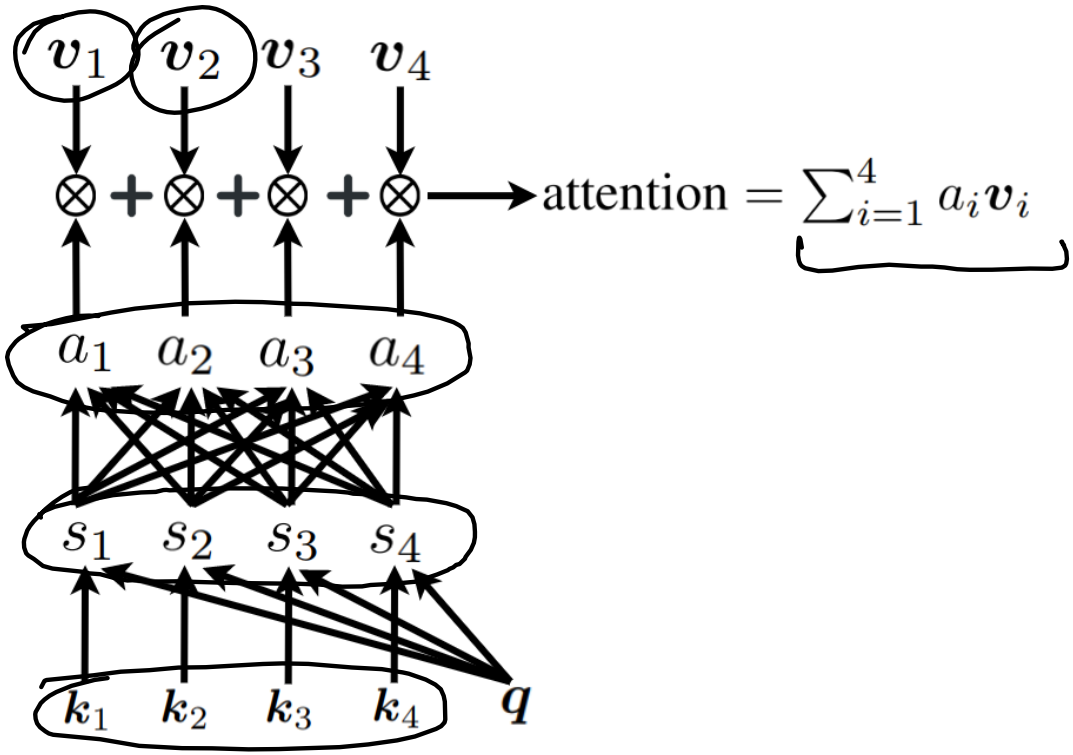
Query-Retrieval Modeling

- An illustration of Eqs. (6), (7), and (8) is shown in this figure.

$$\text{attention}(\boldsymbol{q}, \{\boldsymbol{k}_i\}_{i=1}^n, \{\boldsymbol{v}_i\}_{i=1}^n) := \sum_{i=1}^n a_i \boldsymbol{v}_i,$$

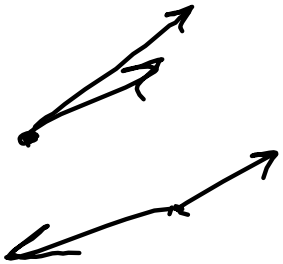
$$\mathbb{R} \ni s_i := \text{similarity}(\boldsymbol{q}, \boldsymbol{k}_i),$$

$$\mathbb{R} \ni a_i := \text{softmax}(s_i) = \frac{e^{s_i}}{\sum_{k=1}^n e^{s_k}}.$$



Query-Retrieval Modeling

- The similarity s_i can be any notion of similarity. Some of the well-known similarity measures are [21]:



$$\text{inner product: } s_i = \boldsymbol{q}^\top \boldsymbol{k}_i, \tag{9}$$

$$\text{scaled inner product: } s_i = \frac{\boldsymbol{q}^\top \boldsymbol{k}_i}{\sqrt{p}}, \tag{10}$$

$$\text{general inner product: } s_i = \boldsymbol{q}^\top \boldsymbol{W} \boldsymbol{k}_i, \tag{11}$$

$$\text{additive similarity: } s_i = \boldsymbol{w}_q^\top \boldsymbol{q} + \boldsymbol{w}_k^\top \boldsymbol{k}_i, \tag{12}$$

where $\boldsymbol{W} \in \mathbb{R}^{p \times p}$, $\boldsymbol{w}_q \in \mathbb{R}^p$, and $\boldsymbol{w}_k \in \mathbb{R}^p$ are some learnable matrices and vectors.

- Among these similarity measures, the scaled inner product is used most often.

Query-Retrieval Modeling

- The Eq. (6), attention$(\boldsymbol{q}, \{\boldsymbol{k}_i\}_{i=1}^n, \{\boldsymbol{v}_i\}_{i=1}^n) := \sum_{i=1}^n a_i \boldsymbol{v}_i$, calculates the attention of a **target word (or query)** with respect to **every other input word (or keys)** which are the previous and forthcoming words.
- Using Eq. (6), we see **how similar the other words** of the sequence are to that word. In other words, we see **how impactful** the other previous and forthcoming words are for generating a missing word in the sequence.
- Example:
  - Sentence: "I am a student".
  - We are processing the word "student".
  - Query: "student"
  - Values: "I", "am", "a"
  - Normalized similarity of the query and the values: weights 0.7, 0.2, and 0.1 for "I", "am", and "a", respectively.
  - The attention value for the word "student": $0.7\boldsymbol{v}_{\text{I}} + 0.2\boldsymbol{v}_{\text{am}} + 0.1\boldsymbol{v}_{\text{a}}$.

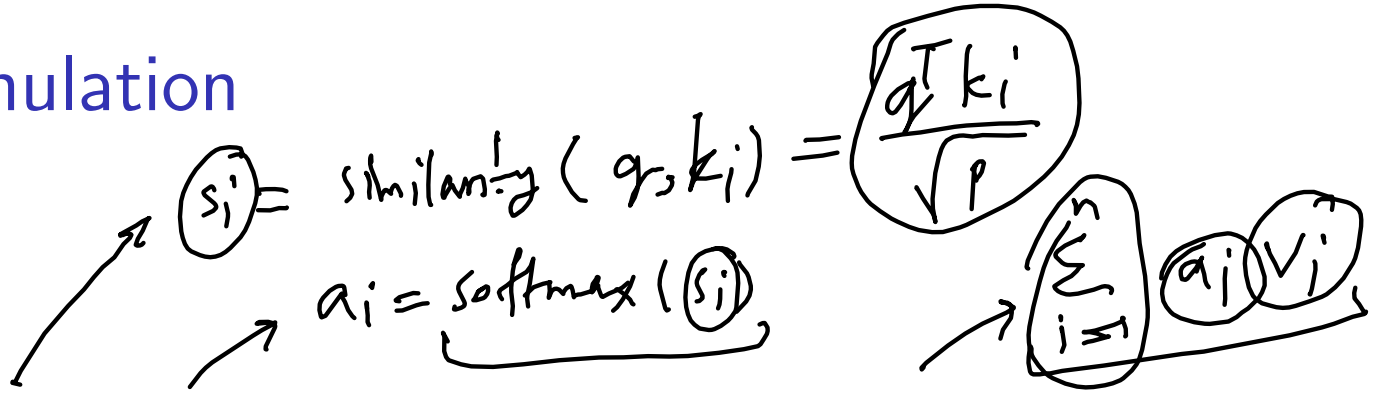
Attention Formulation

- Let the words of a sequence of words be in a d-dimensional space, i.e., the sequence is $\{\boldsymbol{x}_i \in \mathbb{R}^d\}_{i=1}^n$. This d-dimensional representation of words can be taken from any word-embedding NLP method such as word2vec [13, 14, 16, 17] or GloVe [18].
- The query, key, and value are projection of the words into p-dimensional, p-dimensional, and r-dimensional subspaces, respectively:

$$\begin{cases} \mathbb{R}^p \ni \boldsymbol{q}_i = \boldsymbol{W}_Q^\top \boldsymbol{x}_i, & (13) \\ \mathbb{R}^p \ni \boldsymbol{k}_i = \boldsymbol{W}_K^\top \boldsymbol{x}_i, & (14) \\ \mathbb{R}^r \ni \boldsymbol{v}_i = \boldsymbol{W}_V^\top \boldsymbol{x}_i, & (15) \end{cases}$$

  where $\boldsymbol{W}_Q \in \mathbb{R}^{d \times p}$, $\boldsymbol{W}_K \in \mathbb{R}^{d \times p}$, and $\boldsymbol{W}_V \in \mathbb{R}^{d \times r}$ are the projection matrices into the low-dimensional query, key, and value subspaces, respectively.
- Consider the words, queries, keys, and values in matrix forms as $\boldsymbol{X} := [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n] \in \mathbb{R}^{d \times n}$, $\boldsymbol{Q} := [\boldsymbol{q}_1, \ldots, \boldsymbol{q}_n] \in \mathbb{R}^{p \times n}$, $\boldsymbol{K} := [\boldsymbol{k}_1, \ldots, \boldsymbol{k}_n] \in \mathbb{R}^{p \times n}$, and $\boldsymbol{V} := [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n] \in \mathbb{R}^{r \times n}$, respectively.
- In the similarity measures, such as the scaled inner product, we have the inner product $\boldsymbol{q}^\top \boldsymbol{k}_i$. Hence, the similarity measures contain $\boldsymbol{q}^\top \boldsymbol{k}_i = \boldsymbol{x}_i^\top \boldsymbol{W}_Q \boldsymbol{W}_K^\top \boldsymbol{x}_i$. This is like a kernel matrix so its behaviour is similar to the kernel for measuring similarity.

Attention Formulation



- Considering Eqs. (7), and (8) and the above definitions, the Eq. (6) can be written in matrix form, for the whole sequence of n words, as:

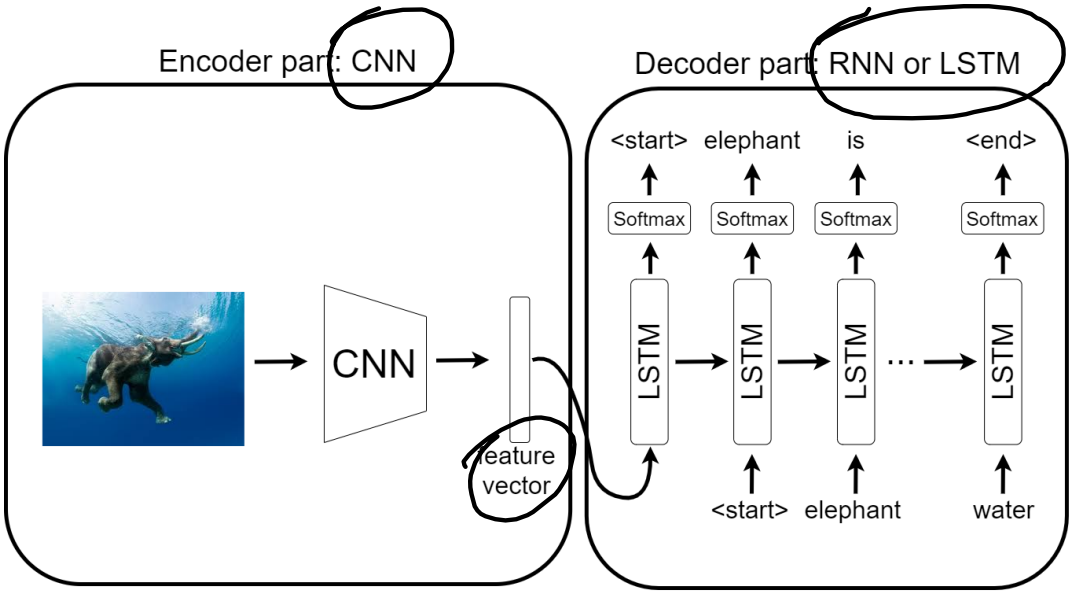
$$\mathbb{R}^{r\times n} \ni \boldsymbol{Z} := \text{attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \boldsymbol{V}\,\text{softmax}\left(\frac{1}{\sqrt{p}}\boldsymbol{Q}^\top \boldsymbol{K}\right), \tag{16}$$

where $\boldsymbol{Z} = [\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n]$ is the attention values, for all the words, which shows how much every word attends to its previous and forthcoming words. In Eq. (16), the softmax operator applies the softmax function on every row of its input matrix so that every row sums to one.

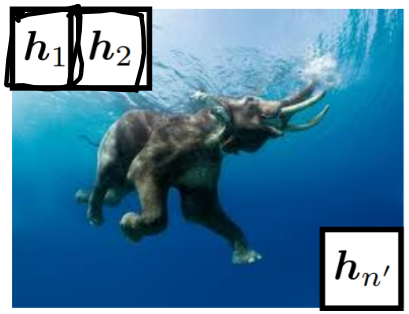
- As the queries, keys, and values are all from the same words in the sequence, this attention is referred to as the "**self-attention**" [19].

Attention in Other Fields Such as Vision and Speech

- The attention concept has widely been used in **NLP** [11, 12]. Attention can be used in the field of **computer vision** [22]. Attention in computer vision means attending to specific parts of image which are more important and informative.
- We can consider the extracted low-layer features, which are different **parts of image**, as the hidden vectors $\{\boldsymbol{h}_j\}_{j=1}^{n'}$ in Eq. (4), where n' is the number of features for extracted image partitions. **Similarity with latent vectors of decoder (LSTM)** is computed by Eq. (4) and the query-retrieval model of **attention** mechanism, introduced before, is used to learn a **self-attention on the images**. Note that, as the partitions of image are considered to be the hidden variables used for attention, the model **attends to important parts of input image**.



(a)



(b)

Attention in Other Fields Such as Vision and Speech

- Using attention in different fields of science is usually referred to as "**attend, tell, and do something...**".
- Some examples of applications of attention are caption generation for images (show, attend and tell) [22], caption generation for images with ownership protection (protect, show, attend and tell) [23], text reading from images containing a text (show, attend and read) [24], translation of one image to another related image (show, attend and translate) [25, 26], visual question answering (show, ask, attend, and answer) [27], human-robot social interaction (show, attend and interact) [28], and speech recognition (listen, attend and spell) [29, 30].

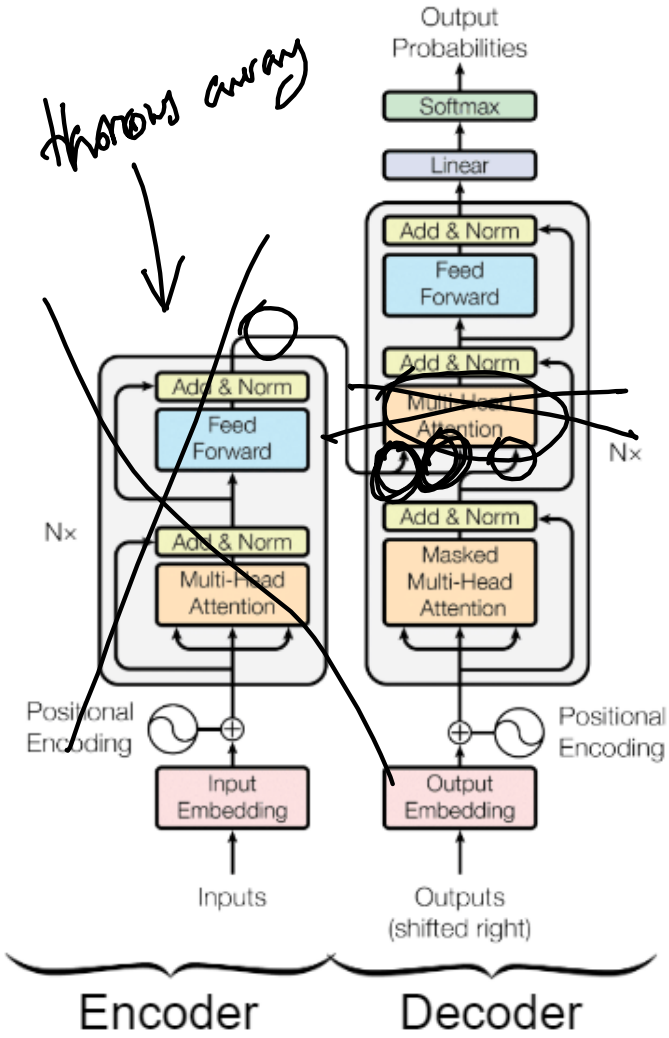
Transformer

The Concept of Transformation

- As explained before, we can have an **autoencoder** which takes an input data, embeds data to a context vector which simulates a concept in human's mind [4], and generates an output. The input and output are related to each other through the context vector. In other words, the autoencoder **transforms the input to a related output**.
- An example for transformation is **translating** a sentence from a language to the same sentence in another language. Another example for transformer is **image captioning** in which the image is transformed to its caption explaining the content of image.
- An autoencoder, named "**transformer**", is proposed in the literature for the task for **transformation** [21].

Transformer Structure

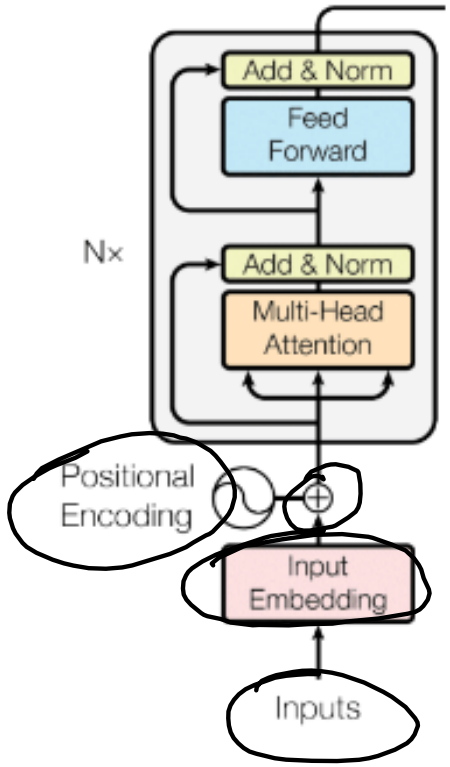
- The structure of transformer is depicted in this figure. A transformer is an **autoencoder** consisting of an **encoder** and a **decoder**.



Encoder of Transformer

Encoder of Transformer: Positional Encoding

- The **encoder** part of transformer embeds the **input** sequence of n words $\boldsymbol{X} \in \mathbb{R}^{d \times n}$ into **context vectors** with the **attention** mechanism.
- We will explain later that transformer does not have any recurrence and RNN or LSTM module. As there is **no recurrence** and no convolution, the model has no sense of **order** in sequence. As the order of words is important for meaning of sentences, we need a way to account for the order of tokens or words in the sequence. For this, we can **add a vector accounting for the position to each input word embedding**.



Encoder of Transformer: Positional Encoding

- Consider the embedding of the i-th word in the sequence, denoted by $\boldsymbol{x}_i \in \mathbb{R}^d$. For encoding the position of the i-th word in the sequence, the **position vector** $\boldsymbol{p}_i \in \mathbb{R}^d$ can be set as:

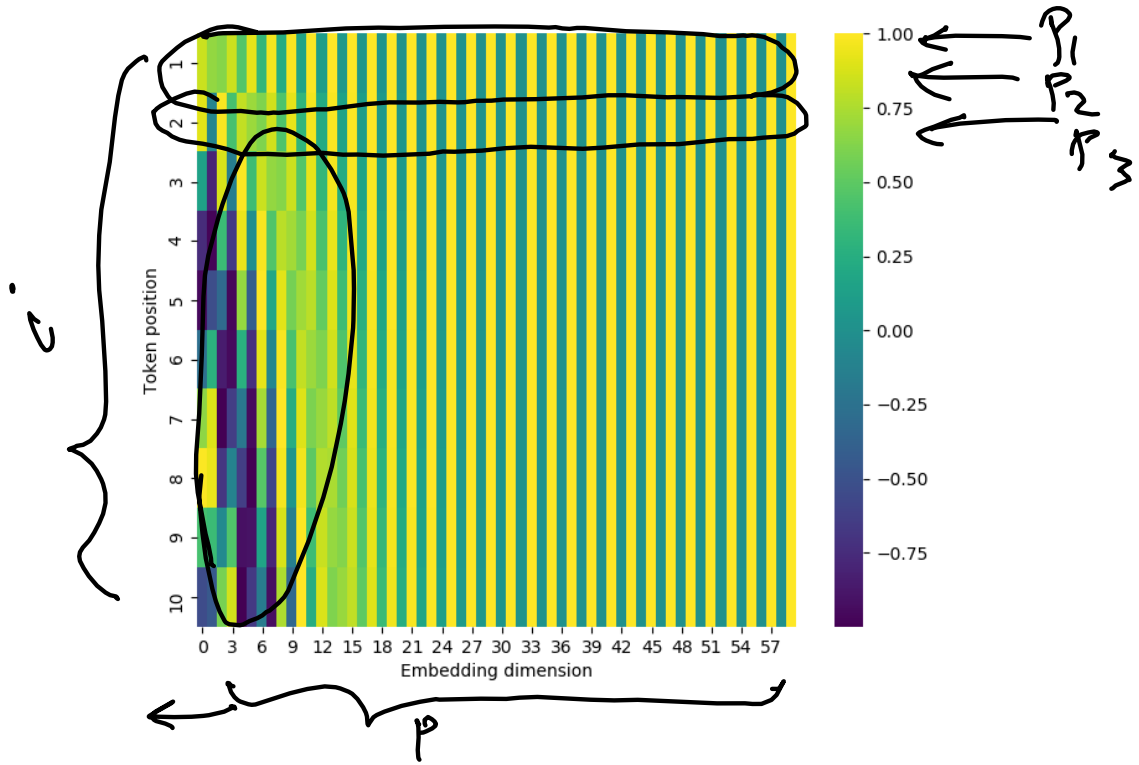
$$\begin{cases} \boldsymbol{p}_i(2j+1) := \cos\left(\dfrac{i}{10000^{\frac{2j}{p}}}\right), \\ \boldsymbol{p}_i(2j) := \sin\left(\dfrac{i}{10000^{\frac{2j}{p}}}\right), \end{cases} \tag{17}$$

for all $j \in \{0, 1, \ldots, \lfloor d/2 \rfloor\}$, where $\boldsymbol{p}_i(2j+1)$ and $\boldsymbol{p}_i(2j)$ denote the odd and even elements of $\boldsymbol{p}_i$, respectively.

Encoder of Transformer: Positional Encoding

- This figure illustrates the dimensions of the position vectors across different positions. The position vectors for different positions of words are **different** as expected. Moreover, this figure shows that the **difference of position vectors concentrate more on the initial dimensions** of vectors.
- For incorporating the information of position with data, we add the positional encoding to the input embedding:

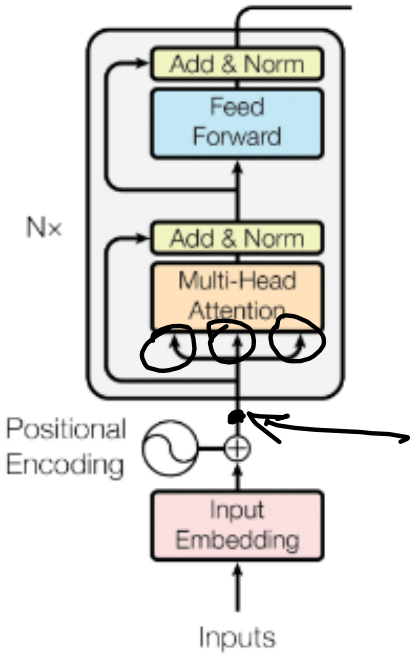
$$\boldsymbol{x}_i \leftarrow \boldsymbol{x}_i + \boldsymbol{p}_i. \tag{18}$$

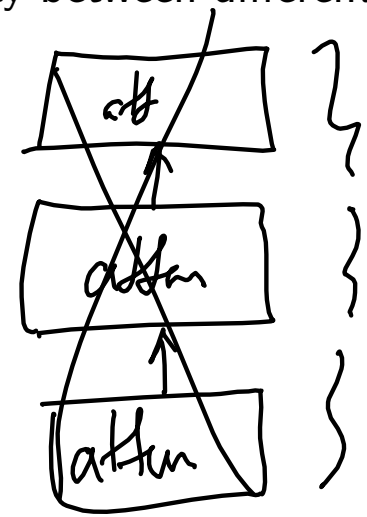


Multihead Attention with Self-Attention

- After positional encoding, data are fed to a **multihead attention** module with **self-attention**.
- This module applies the attention mechanism **for h times**.
- This several repeats of attention is for the reason explained here. The first attention determines how much every word attends to other words. The second repeat of attention calculates how much every pair of words attends to other pairs of words. Likewise, the third repeat of attention sees how much every pair of pairs of words attends to other pairs of pairs of words; and so on.
- This measure of attention or similarity between hierarchical pairs of words reminds us of the maximum mean discrepancy [31, 32] which measures similarity between different moments of data distributions.

$$\| \phi(x_i) - \phi(x_j) \|_2^2$$





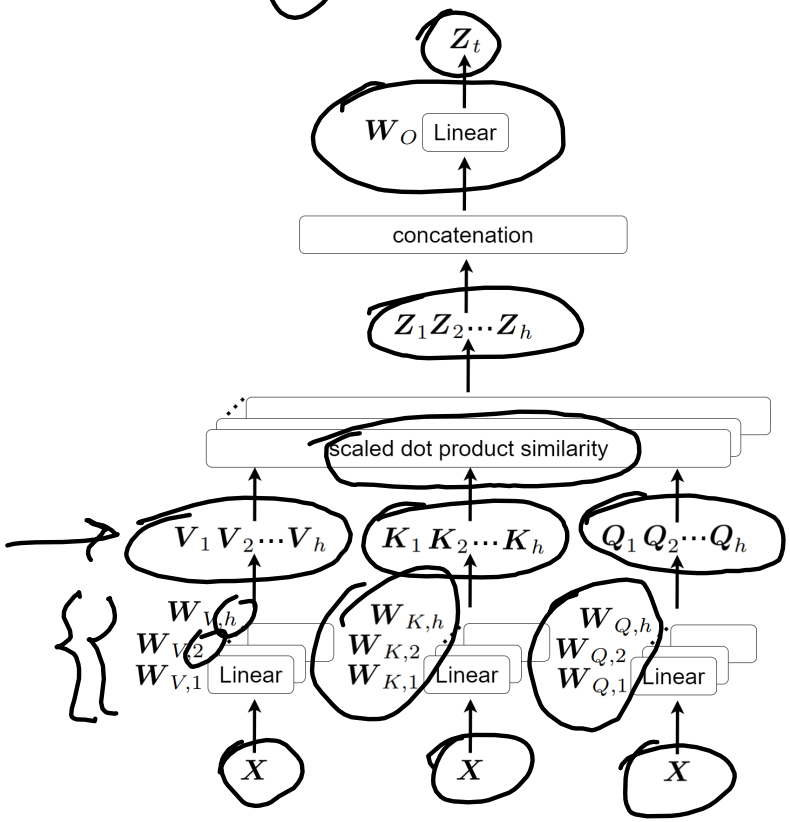
Multihead Attention with Self-Attention

- The data, which include positional encoding, are passed from linear layers for obtaining the queries, values, and keys. These linear layers model linear projections introduced in Eqs. (13), (15), and (14), respectively. We have h of these linear layers to generate h set of queries, values, and keys as:

$$\mathbb{R}^{p\times n} \ni \boldsymbol{Q}_i = \boldsymbol{W}_{Q,i}^\top \boldsymbol{X}, \quad \forall i \in \{1,\dots,h\}, \tag{19}$$

$$\mathbb{R}^{p\times n} \ni \boldsymbol{V}_i = \boldsymbol{W}_{V,i}^\top \boldsymbol{X}, \quad \forall i \in \{1,\dots,h\}, \tag{20}$$

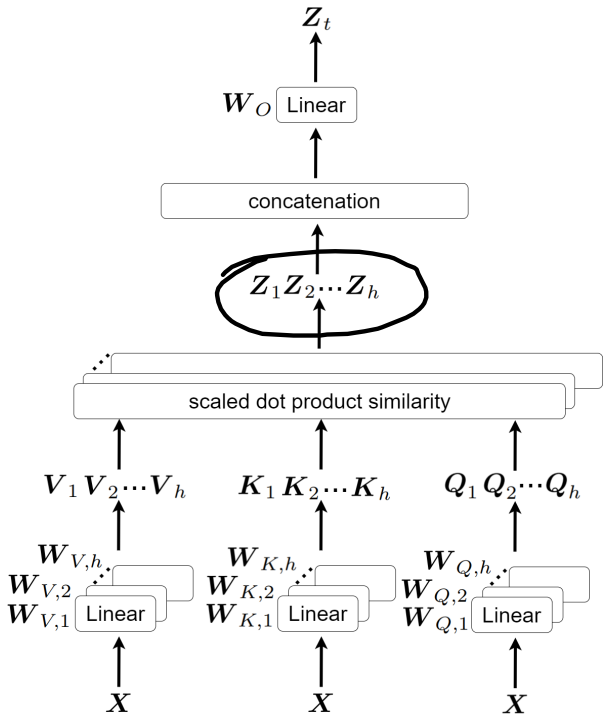
$$\mathbb{R}^{r\times n} \ni \boldsymbol{K}_i = \boldsymbol{W}_{K,i}^\top \boldsymbol{X}, \quad \forall i \in \{1,\dots,h\}. \tag{21}$$



Multihead Attention with Self-Attention

- Then, the scaled dot product similarity, defined in Eq. (10) or (16), is used to generate the h attention values $\{\boldsymbol{Z}_i\}_{i=1}^{h}$. These h attention values are concatenated to make a new long flattened vector. Then, by a linear layer, which is a linear projection, the total attention value, $\boldsymbol{Z}_t$, is obtained:

$$\boldsymbol{Z}_t = \boldsymbol{W}_O^{\top} \text{concat}(\boldsymbol{Z}_1, \boldsymbol{Z}_2, \ldots, \boldsymbol{Z}_h). \tag{22}$$



Layer Normalization

- The data (containing positional encoding) and the total attention value are added:

$$z'_t \leftarrow z_t + x. \tag{23}$$

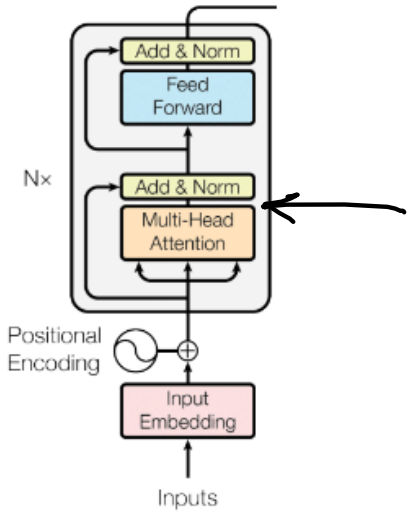
- This addition is inspired by the concept of residual introduced by ResNet [33].
- After this addition, a layer normalization is applied where for each hidden unit h_i we have:

$$h_i \leftarrow \frac{g}{\sigma}(h_i - \mu), \tag{24}$$

where μ and σ are the empirical mean and standard deviation over H hidden units:

$$\mu := \frac{1}{H}\sum_{i=1}^{H} h_i, \quad \sigma := \sqrt{\sum_{i=1}^{H}(h_i - \mu)^2}. \tag{25}$$

- This is a standardization which makes the mean zero and the variance one; it is closely related to batch normalization and reduces the covariate shift [34].

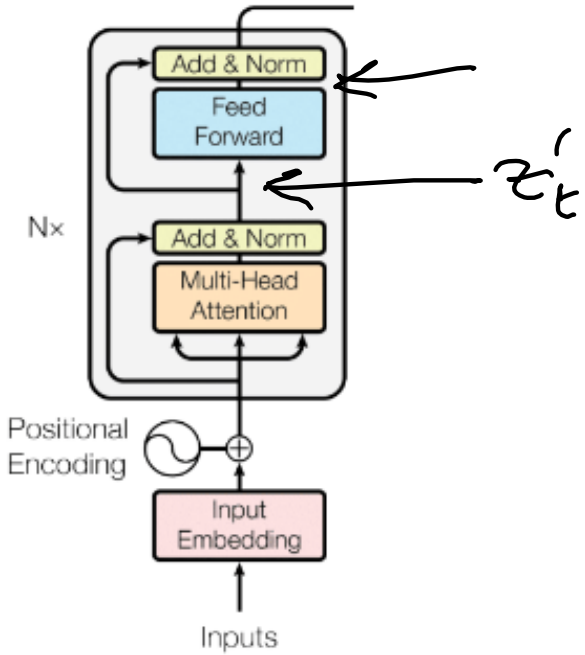


Feedforward Layer

- Henceforth, let $\boldsymbol{Z}'_t$ denote the total attention after both addition and layer normalization.
- We feed $\boldsymbol{Z}'_t$ to a feedforward network, having nonlinear activation functions, and then like before, we **add the input of feedforward network to its output**:

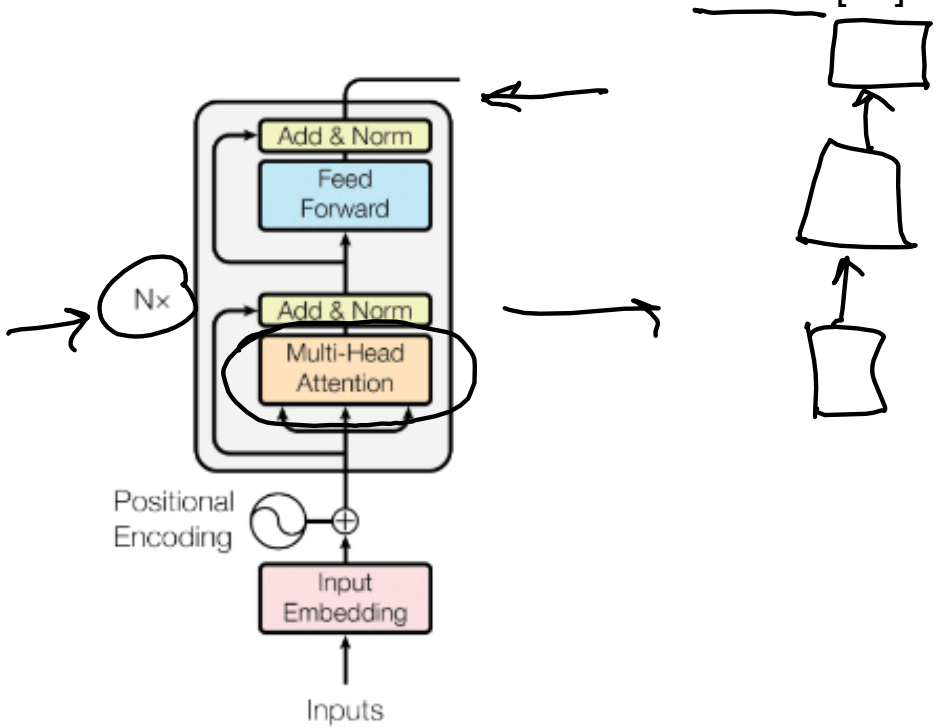
$$\boldsymbol{Z}''_t \leftarrow \boldsymbol{R} + \boldsymbol{Z}'_t, \tag{26}$$

  where $\boldsymbol{R}$ denotes the output of feedforward network.
- Again, **layer normalization** is applied and we, henceforth, denote the output of encoder by $\boldsymbol{Z}''_t$.
- This is the encoding for the whole input sequence or sentence having the information of attention of words and hierarchical pairs of words to each other.



Stacking

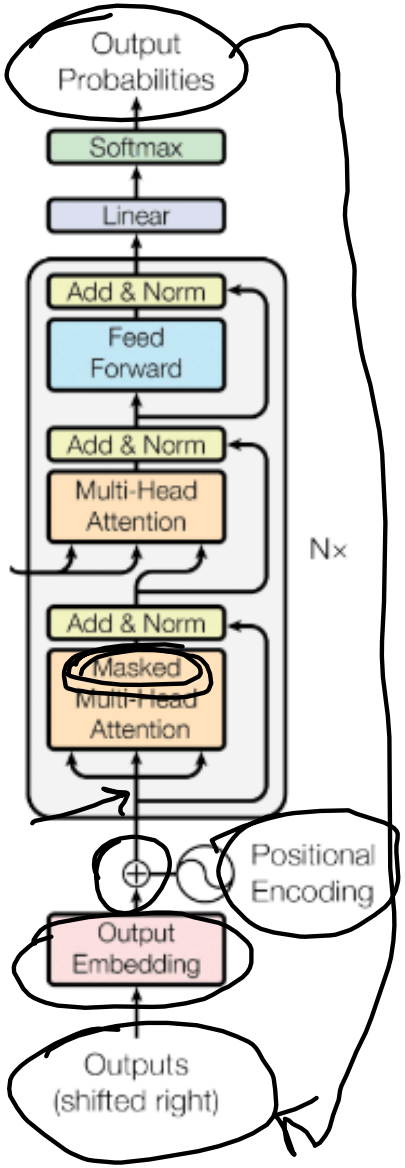
- The encoder is a **stack** of N identical layers.
- This stacking is for having **more learnable parameters** to have enough degree of freedom to learn the whole dictionary of words.
- Through experiments, a good number of stacks is found to be $N = 6$ [21].



Decoder of Transformer

Masked Multihead Attention with Self-Attention

- A part of decoder is the masked **multihead attention** module whose input is the output embeddings $\{\boldsymbol{y}_i\}_{i=1}^{n}$ shifted one word to the right.
- **Positional encoding** is also added to the output embeddings for including the information of their positions. For this, we use Eq. (18) where $\boldsymbol{x}_i$ is replaced by $\boldsymbol{y}_i$.
- The output embeddings added with the positional encodings are fed to the masked multihead attention module. This module is similar to the multihead attention module but **masks away the forthcoming words after a word**.
- Therefore, every output word only attends to its previous output words, every pair of output words attends to its previous pairs of output words, every pair of pairs of output words attends to its previous pairs of pairs of output words, and so on.
- The reason for using the masked version of multihead attention for the output embeddings is that when we are generating the output text, we **do not have the next words yet** because the next words are not generated yet.
- This masking imposes some idea of **sparsity** which was also introduced by the **dropout** technique [35] but in a stochastic manner.



Masked Multihead Attention with Self-Attention

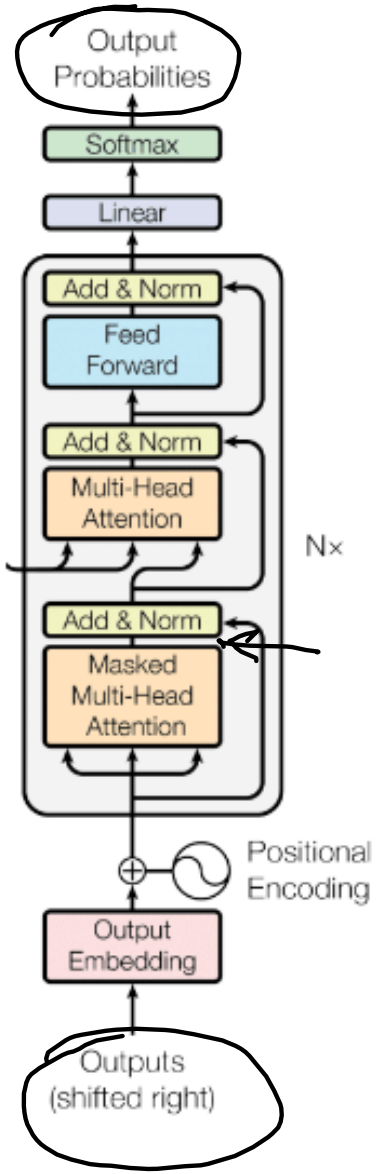
- Recall Eq. (16) which was used for multihead attention. The masked multihead attention is defined as:

$$\begin{aligned} \mathbb{R}^{r\times n} \ni \boldsymbol{Z}_m &:= \text{maskedAttention}(\boldsymbol{Q},\boldsymbol{K},\boldsymbol{V}) \\ &= \boldsymbol{V}\,\text{softmax}\left(\frac{1}{\sqrt{p}}(\boldsymbol{Q}^\top \boldsymbol{K} + \boldsymbol{M})\right), \end{aligned} \tag{27}$$

where the mask matrix $\boldsymbol{M} \in \mathbb{R}^{n\times n}$ is:

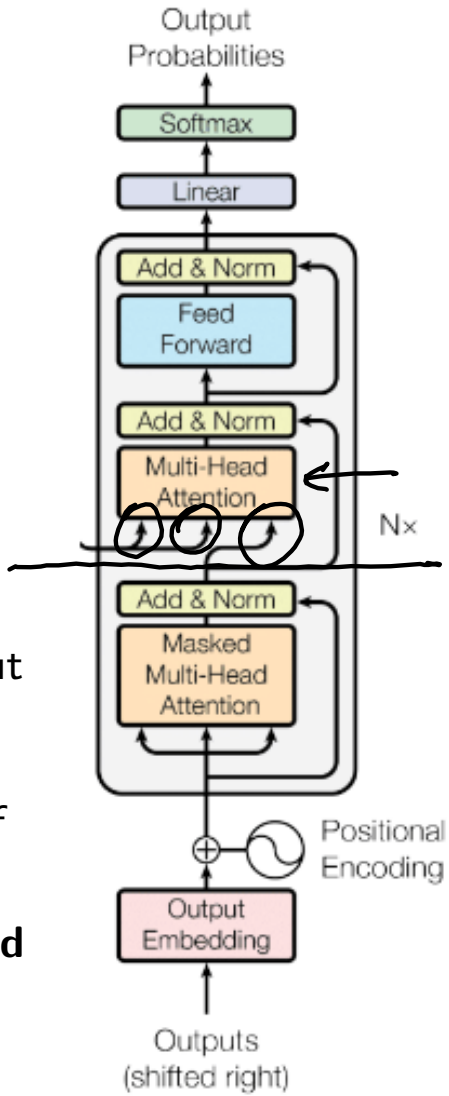
$$\boldsymbol{M}(i,j) := \begin{cases} 0 & \text{if } j \le i, \\ -\infty & \text{if } j > i. \end{cases} \tag{28}$$

- As the softmax function has exponential operator, the mask does not have any impact for $j \le i$ (because it is multiplied by $e^0 = 1$) and masks away for $j > i$ (because it is multiplied by $e^{-\infty} = 0$). Note that $j \le i$ and $j > i$ correspond to the previous and next words, respectively, in terms of position in the sequence.
- Similar to before, the output of masked multihead attention is **normalized** and then is **added to its input**.



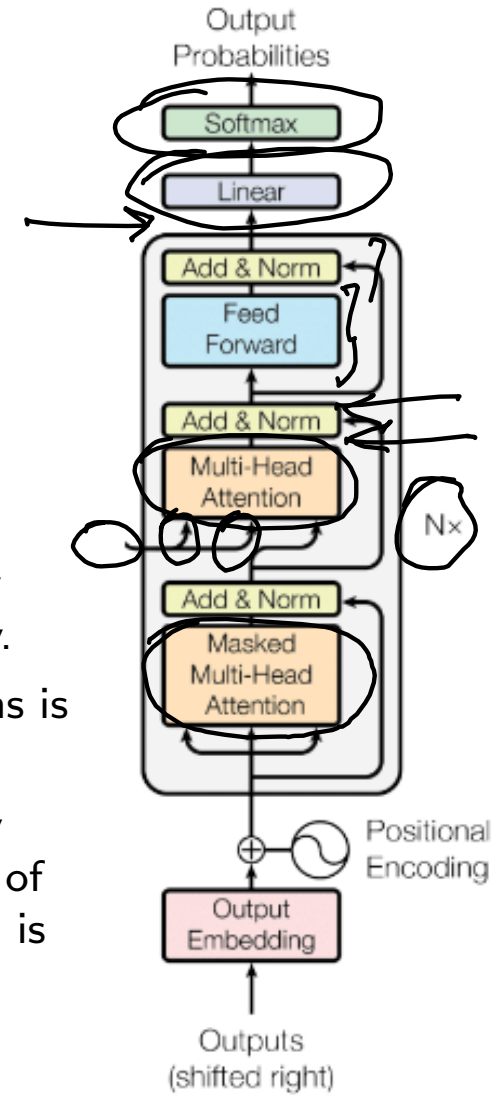
Multihead Attention with Cross-Attention

- The output of masked multihead attention module is fed to a **multihead attention** module with **cross-attention**.
- This module is not self-attention because all its values, keys, and queries are not from the same sequence but its **values and keys are from the output of encoder** and the **queries are from the output of the masked multihead attention module in the decoder**.
- In other words, the **values and keys come from the processed input embeddings** and the **queries are from the processed output embeddings**.
- The calculated multihead attention determines how much every output embedding attends to the input embeddings, how much every pair of output embeddings attends to the pairs of input embeddings, how much every pair of pairs of output embeddings attends to the pairs of pairs of input embeddings, and so on.
- This shows the **connection** between **input sequence** and the **generated output sequence**.



Feedforward Layer and Softmax Activation

- Again, the output of the multihead attention module with cross-attention is **normalized** and **added to its input**.
- Then, it is fed to a **feedforward** neural network with layer **normalization** and **added to its input** afterwards.
- The masked multihead attention, the multihead attention with cross-attention, and the feedforward network are **stacked** for $N = 6$ times.
- The output of feedforward network passes through a **linear layer** by linear projection and a **softmax** activation function is applied finally.
- The number of output neurons with the softmax activation functions is the **number of all words in the dictionary** which is a large number.
- The outputs of decoder sum to one and are the probability of every word in the dictionary to be the generated next word. For the sake of sequence generation, the token or word with the **largest probability** is the next word.



Complexity of Attention

Attention is All We Need!

- The output of decoder is fed to the masked multihead attention module of decoder with some shift.
- This is not a notion of recurrence. Hence, we see that there is not any recurrent module like RNN [9] and LSTM [10] in transformer. Therefore, **attention is all we need** to learn a sequence and there is no need to any recurrence module.
- The proposal of transformers [21] was a **breakthrough in NLP**; the state-of-the-art NLP methods are all based on transformers nowadays.

Complexity Comparison

- In self-attention, we learn attention of every word to every other word in the sequence of n **words**. Also, we learn a p-**dimensional embedding for every word**. Hence, the complexity of operations per layer is $\mathcal{O}(n^2 p)$. This is while the complexity per layer in recurrence is $\mathcal{O}(np^2)$.
- Although, the complexity per layer in self-attention is worse than recurrence, many of its operations can be performed in **parallel** because all the words of sequence are processed simultaneously, as also explained in the following. Hence, the $\mathcal{O}(n^2 p)$ is not very bad for being able to parallelize it. That is while the recurrence cannot be parallelized for its sequential nature.
- As for the number of sequential operations, the self-attention mechanism **processes all the n words simultaneously** so its sequential operations is in the order of $\mathcal{O}(1)$. As recurrence should process the words **sequentially**, the number of its sequential operations is of order $\mathcal{O}(n)$.
- As for the **maximum path length** between every two words, self-attention learns **attention between every two words**; hence, its maximum path length is of the order $\mathcal{O}(1)$. However, in recurrence, as every word requires a path with a length of a fraction of sequence (a length of n in the worst case) to reach the process of another word, its maximum path length is $\mathcal{O}(n)$.

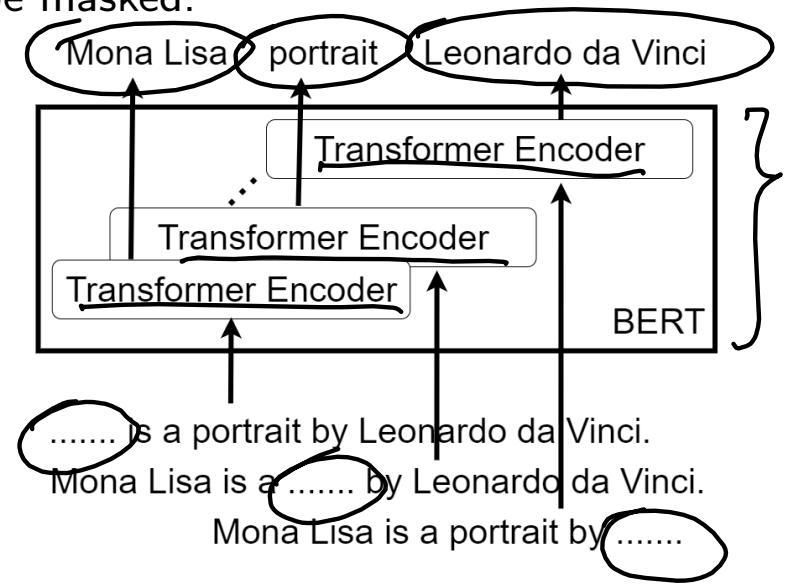
Table 1. Comparison of complexities between self-attention and recurrence (Vaswani et al., 2017).

	Complexity per layer	Sequential operations	Maximum path length
Self-Attention	$\mathcal{O}(n^2 p)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Recurrence	$\mathcal{O}(np^2)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$

BERT

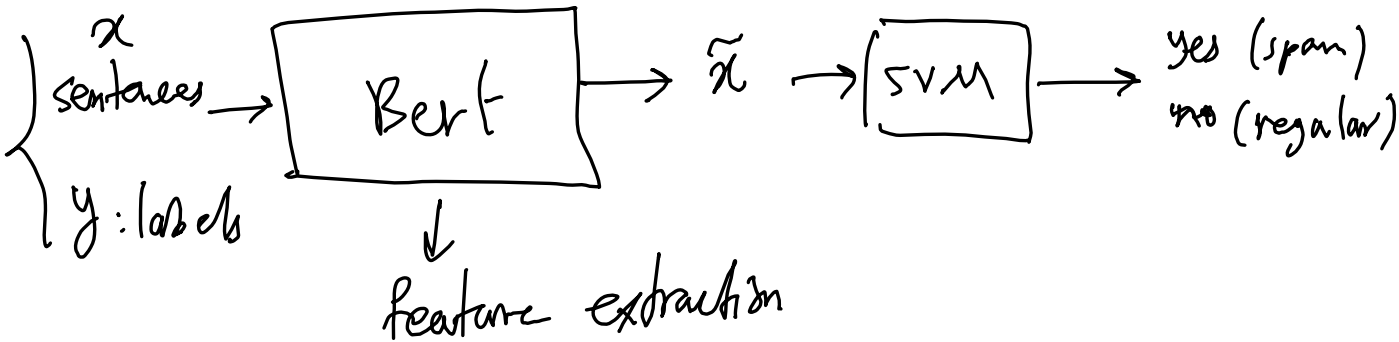
BERT

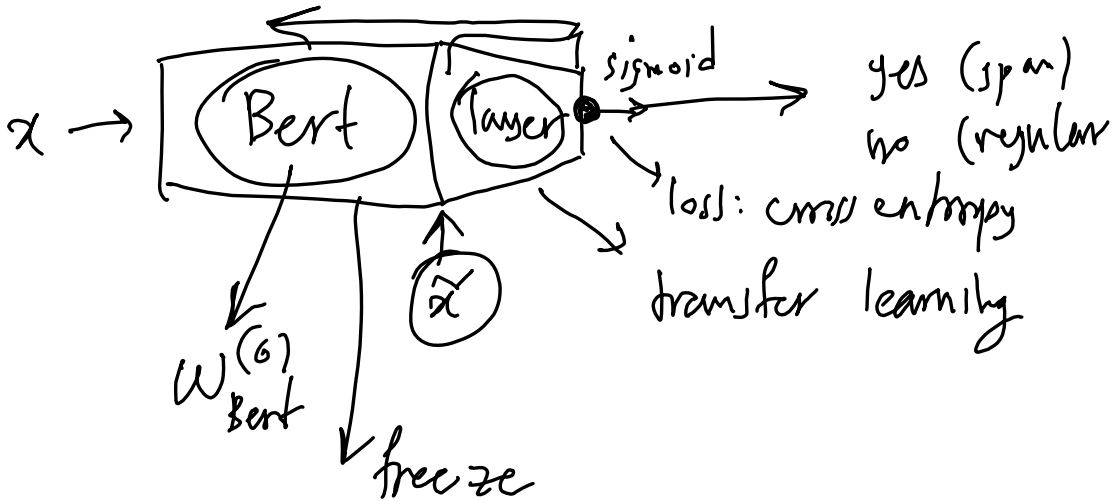
- **BERT (Bidirectional Encoder Representations from Transformers)** [36] is one of the state-of-the-art methods for NLP. It is a **stack of encoders of transformer**. In other words, it is built using **transformer encoder blocks**.
- Although some NLP methods such as XLNet [37] have slightly outperformed it, BERT is still one of the best models for different NLP tasks such as **question answering** [38], **natural language understanding** [39], **sentiment analysis**, and **language inference** [40].
- BERT uses the technique of **masked language modeling**. It **masks 15% of words** in the input document/corpus and asks the model to predict the missing words. A sentence with a **missing word** is given to **every transformer encoding block in the stack** and the block is supposed to **predict the missing word**.
- It is an **unsupervised** manner because any word can be masked in a sentence and the output is supposed to be that word. As it is unsupervised and does not require labels, the **huge text data of Internet** can be used for training the BERT model where words are randomly selected to be masked.



BERT

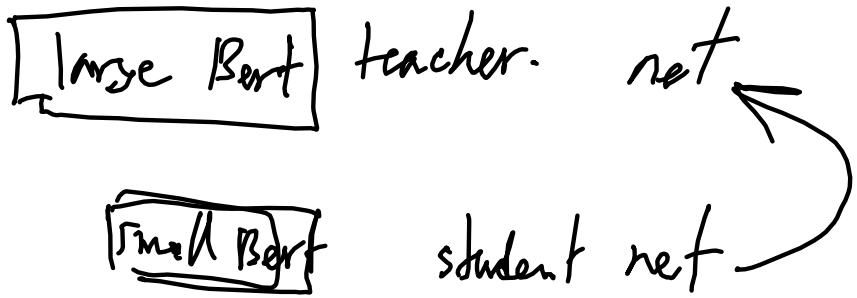
- BERT learns to predict the missing word based on attention to its previous and forthcoming words so it is **bidirectional**. Hence, BERT **jointly conditions on both left (previous) and right (forthcoming)** context of every word.
- Moreover, as the missing word is predicted based on the other words of sentence, BERT embeddings for words are **context-aware embeddings**. Therefore, in contrast to word2vec [13, 14, 16, 17] and GloVe [18] which provide a single embedding per each word, every word has **different BERT embeddings in various sentences**.
- The BERT embeddings of words differ in different sentences based on their context. For example, the word "bank" has different meanings and therefore different embeddings in the sentences "**Money is in the bank**" and "**Some plants grow in bank of rivers**".
- It is also noteworthy that, for an input sentence, BERT outputs **an embedding for the whole sentence** in addition to giving **embeddings for every word** of the sentence. This sentence embedding is not perfect but works well enough in applications. One can use the BERT sentence embeddings and train a classifier on them for the task of spam detection or sentiment analysis.
- The BERT model is usually not trained from the scratch as its training has been done in a long time on huge amount of Internet data. For using it in different NLP applications, such as sentiment analysis, researchers usually do **transfer learning** and **add one or several neural network layers** on top of a pre-trained BERT model and train the network for their own task. During training, one can either **freeze** the weights of the BERT model and just train the added layers or **also fine tune** BERT weights by backpropagation.


x
sentences
y : labels
Bert
feature extraction
$\tilde{x}$
SVM
yes (span)
no (regular)


x
Bert
layer
sigmoid
yes (span)
no (regular
loss: cross entropy
transfer learning
$\tilde{x}$
$W^{(6)}_{Bert}$
freeze

BERT

- The parameters of encoder in transformer [21] are **6 encoder layers**, **512 hidden layer units** in the fully connected network and **8 attention heads** ($h = 8$).
- This is while BERT [36] has **24 encoder layers**, **1024 hidden layer units** in the fully connected network and **16 attention heads** ($h = 16$).
- Usually, when we say BERT, we mean the **large BERT** [36] with the above-mentioned parameters.
- As the BERT model is huge and requires a lot of memory for saving the model, it cannot easily be used in **embedded systems**. Hence, many commercial smaller versions of BERT are proposed with less number of parameters and number of stacks. Some of these smaller versions of BERT are **small BERT** [41], **tiny BERT** [42], **DistilBERT** [43], and **Roberta BERT** [44].
- Some BERT models, such as **clinical BERT** [45] and **BioBERT** [46], have also been trained on **medical texts** for the biomedical applications.



GPT

GPT

- **GPT (Generative Pre-trained Transformer)**, or **GPT-1**, [47] is another state-of-the-art method for NLP.
- It is a stack of **decoders of transformer**. In other words, it is built using **transformer decoder blocks**.
- In GPT, the **multihead attention** module with **cross-attention** is **removed** from the decoder of transformer because there is **no encoder in GPT**.
- Hence, the decoder blocks used in GPT have only **positional encoding**, **masked multihead self-attention** module and **feedforward** network with their **adding**, **layer normalization**, and **activation** functions.
- As GPT uses the masked multihead self-attention, it considers attention of word, pairs of words, pairs of pairs of words, and so on, only on the previous (left) words, pairs of words, pairs of pairs of words, and so on. In other words, GPT is **not bidirectional** and **conditions only on the previous words** and not the forthcoming words.
- The objective of BERT was to **predict a masked word** in a sentence. However, GPT model is used for **language model** [48, 49, 50] whose objective is to **predict the next word**, in an incomplete sentence, given all of the previous words.
- The **predicted new word is then added to the sequence and is fed to the GPT as input again** and the other next word is predicted. This goes on until the sentences get complete with their next coming words.
- In other words, GPT model takes some document and **continues the text** in the best and related way. For example, if the input sentences are about psychology, the trained GPT model generates the next words and sentences also about psychology to complete the document.
- As any text without label can be used for predicting the next words in sentences, GPT is an **unsupervised** method making it possible to be trained on **huge amount of Internet**

GPT

- The successors of **GPT-1** [47] are **GPT-2** [51] and **GPT-3** [52]. GPT-2 and GPT-3 are extension of GPT-1 with **more number of stacks** of transformer decoder. Hence, they have **more learnable parameters** and can be trained with more data for better language modeling and inference.
- For example, **GPT-2** has **1.5 billion** parameters.
- GPT-2 and especially GPT-3 have been trained with **much more Internet data** with various general and academic subjects to be able to generate text in any subject and style of interest. For example, GPT-2 has been trained on **8 million web pages** which contain 40GB of Internet text data.
- GPT-2 is a quite large model and cannot be easily used in **embedded systems** because of requiring large memory. Hence, different sizes of GPT-2, like **small, medium, large, Xlarge, and DistilGPT-2**, are provided for usage in embedded systems, where the number of stacks and learnable parameters differ in these versions. These versions of GPT-2 can be found and used in the HuggingFace transformer Python package [53].
- GPT-2 has been used in many different applications such as **dialogue systems** [54], **patent claim generation** [55], and **medical text simplification** [56]. A combination of GPT-2 and BERT has been used for **question answering** [57].
- It is noteworthy that GPT can be seen as **few shot learning** [52]. A **comparison** of GPT and BERT can also be found in [58].

GPT

- GPT-3 is a very huge version of GPT with so many number of stacks and learnable parameters. For comparison, note that **GPT-2**, **NVIDIA Megatron** [59], **Microsoft Turing-NLG** [60], and **GPT-3** [52] have **1.5 billion**, **8 billion**, **17 billion**, and **175 billion** learnable parameters, respectively.
- This huge number of parameters allows GPT-3 to be trained on **very huge amount of Internet text data** with **various subjects** and topics.
- Hence, GPT-3 has been able to learn **almost all topics** of documents and even some people are discussing whether it can pass the **Turing's writer's test** [61, 62].
- Note that GPT-3 has **kind of memorized** the texts of all subjects but **not in a bad way, i.e., overfitting**, rather in a good way. This memorization is because of the **complexity of huge number of learnable parameters** [63] and not being overfitted is because of being trained by **big enough Internet data**.
- GPT-3 has had many different interesting applications such as **fiction and poetry generation** [64].

Acknowledgment

- Some slides of this slide deck are inspired by teachings of Prof. Ali Ghodsi (at University of Waterloo, Department of Statistics) and Prof. Pascal Poupart (at University of Waterloo, Department of Computer Science).
- This lecture is based on our tutorial paper: Benyamin Ghojogh, Ali Ghodsi, "Attention mechanism, transformers, BERT, and GPT: Tutorial and survey" [65]

References

[1] Y. Li, L. Kaiser, S. Bengio, and S. Si, "Area attention," in *International Conference on Machine Learning*, pp. 3846–3855, PMLR, 2019.

[2] J. Xu, "On the techniques of English fast-reading," in *Theory and Practice in Language Studies*, vol. 1, pp. 1416–1419, Academy Publisher, 2011.

[3] K. Yu, Y. Liu, A. G. Schwing, and J. Peng, "Fast and accurate text classification: Skimming, rereading and early stopping," in *International Conference on Learning Representations*, 2018.

[4] L. I. Perlovsky, "Toward physics of the mind: Concepts, emotions, consciousness, and symbols," *Physics of Life Reviews*, vol. 3, no. 1, pp. 23–55, 2006.

[5] Y. Wang, L. Wang, S. Shi, V. O. Li, and Z. Tu, "Go from the general to the particular: Multi-domain translation with domain transformation networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 9233–9241, 2020.

[6] S. Ben-David, J. Blitzer, K. Crammer, A. Kulesza, F. Pereira, and J. W. Vaughan, "A theory of learning from different domains," *Machine learning*, vol. 79, no. 1-2, pp. 151–175, 2010.

[7] Q. Dou, D. Coelho de Castro, K. Kamnitsas, and B. Glocker, "Domain generalization via model-agnostic learning of semantic features," *Advances in Neural Information Processing Systems*, vol. 32, pp. 6450–6461, 2019.

References (cont.)

[8] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *International Conference on Learning Representations*, 2015.

[9] S. Kombrink, T. Mikolov, M. Karafiát, and L. Burget, "Recurrent neural network based language modeling in meeting recognition," in *Twelfth annual conference of the international speech communication association*, 2011.

[10] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[11] J. Chorowski, D. Bahdanau, K. Cho, and Y. Bengio, "End-to-end continuous speech recognition using attention-based recurrent NN: First results," *arXiv preprint arXiv:1412.1602*, 2014.

[12] M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," *arXiv preprint arXiv:1508.04025*, 2015.

[13] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *International Conference on Learning Representations*, 2013.

[14] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, pp. 3111–3119, 2013.

References (cont.)

[15] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[16] Y. Goldberg and O. Levy, "word2vec explained: deriving Mikolov et al.'s negative-sampling word-embedding method," *arXiv preprint arXiv:1402.3722*, 2014.

[17] T. Mikolov, K. Chen, G. S. Corrado, and J. A. Dean, "Computing numeric representations of words in a high-dimensional space," May 19 2015.
US Patent 9,037,464.

[18] J. Pennington, R. Socher, and C. D. Manning, "GloVe: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing*, pp. 1532–1543, 2014.

[19] J. Cheng, L. Dong, and M. Lapata, "Long short-term memory-networks for machine reading," in *Conference on Empirical Methods in Natural Language Processing*, pp. 551–561, 2016.

[20] H. Garcia-Molina, J. D. Ullman, and J. Widom, *Database systems: The complete book*.
Prentice Hall, 1999.

[21] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, pp. 5998–6008, 2017.

References (cont.)

[22] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio, "Show, attend and tell: Neural image caption generation with visual attention," in *International conference on machine learning*, pp. 2048–2057, 2015.

[23] J. H. Lim, C. S. Chan, K. W. Ng, L. Fan, and Q. Yang, "Protect, show, attend and tell: Image captioning model with ownership protection," *arXiv preprint arXiv:2008.11009*, 2020.

[24] H. Li, P. Wang, C. Shen, and G. Zhang, "Show, attend and read: A simple and strong baseline for irregular text recognition," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 8610–8617, 2019.

[25] H. Zhang, W. Chen, J. Tian, Y. Wang, and Y. Jin, "Show, attend and translate: Unpaired multi-domain image-to-image translation with visual attention," *arXiv preprint arXiv:1811.07483*, 2018.

[26] C. Yang, T. Kim, R. Wang, H. Peng, and C.-C. J. Kuo, "Show, attend, and translate: Unsupervised image translation with self-regularization and attention," *IEEE Transactions on Image Processing*, vol. 28, no. 10, pp. 4845–4856, 2019.

[27] V. Kazemi and A. Elqursh, "Show, ask, attend, and answer: A strong baseline for visual question answering," *arXiv preprint arXiv:1704.03162*, 2017.

[28] A. H. Qureshi, Y. Nakamura, Y. Yoshikawa, and H. Ishiguro, "Show, attend and interact: Perceivable human-robot social interaction through neural attention q-network," in *2017 IEEE International Conference on Robotics and Automation*, pp. 1639–1645, IEEE, 2017.

References (cont.)

[29] W. Chan, N. Jaitly, Q. V. Le, and O. Vinyals, "Listen, attend and spell," *arXiv preprint arXiv:1508.01211*, 2015.

[30] W. Chan, N. Jaitly, Q. Le, and O. Vinyals, "Listen, attend and spell: A neural network for large vocabulary conversational speech recognition," in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 4960–4964, IEEE, 2016.

[31] A. Gretton, K. Borgwardt, M. Rasch, B. Schölkopf, and A. J. Smola, "A kernel method for the two-sample-problem," in *Advances in neural information processing systems*, pp. 513–520, 2007.

[32] A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf, and A. Smola, "A kernel two-sample test," *The Journal of Machine Learning Research*, vol. 13, no. 1, pp. 723–773, 2012.

[33] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[34] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[35] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

References (cont.)

[36] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[37] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le, "XLnet: Generalized autoregressive pretraining for language understanding," in *Advances in neural information processing systems*, pp. 5753–5763, 2019.

[38] C. Qu, L. Yang, M. Qiu, W. B. Croft, Y. Zhang, and M. Iyyer, "BERT with history answer embedding for conversational question answering," in *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 1133–1136, 2019.

[39] L. Dong, N. Yang, W. Wang, F. Wei, X. Liu, Y. Wang, J. Gao, M. Zhou, and H.-W. Hon, "Unified language model pre-training for natural language understanding and generation," in *Advances in Neural Information Processing Systems*, pp. 13063–13075, 2019.

[40] Y. Song, J. Wang, Z. Liang, Z. Liu, and T. Jiang, "Utilizing BERT intermediate layers for aspect based sentiment analysis and natural language inference," *arXiv preprint arXiv:2002.04815*, 2020.

[41] H. Tsai, J. Riesa, M. Johnson, N. Arivazhagan, X. Li, and A. Archer, "Small and practical BERT models for sequence labeling," *arXiv preprint arXiv:1909.00100*, 2019.

References (cont.)

[42] X. Jiao, Y. Yin, L. Shang, X. Jiang, X. Chen, L. Li, F. Wang, and Q. Liu, "TinyBERT: Distilling BERT for natural language understanding," *arXiv preprint arXiv:1909.10351*, 2019.

[43] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "DistilBERT, a distilled version of bert: smaller, faster, cheaper and lighter," *arXiv preprint arXiv:1910.01108*, 2019.

[44] I. Staliūnaitė and I. Iacobacci, "Compositional and lexical semantics in RoBERTa, BERT and DistilBERT: A case study on CoQA," *arXiv preprint arXiv:2009.08257*, 2020.

[45] E. Alsentzer, J. R. Murphy, W. Boag, W.-H. Weng, D. Jin, T. Naumann, and M. McDermott, "Publicly available clinical BERT embeddings," *arXiv preprint arXiv:1904.03323*, 2019.

[46] J. Lee, W. Yoon, S. Kim, D. Kim, S. Kim, C. H. So, and J. Kang, "BioBERT: a pre-trained biomedical language representation model for biomedical text mining," *Bioinformatics*, vol. 36, no. 4, pp. 1234–1240, 2020.

[47] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding by generative pre-training," tech. rep., OpenAI, 2018.

[48] R. Rosenfeld, "Two decades of statistical language modeling: Where do we go from here?," *Proceedings of the IEEE*, vol. 88, no. 8, pp. 1270–1278, 2000.

[49] R. Jozefowicz, O. Vinyals, M. Schuster, N. Shazeer, and Y. Wu, "Exploring the limits of language modeling," *arXiv preprint arXiv:1602.02410*, 2016.

References (cont.)

[50] K. Jing and J. Xu, "A survey on neural network language models," *arXiv preprint arXiv:1906.03591*, 2019.

[51] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[52] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, *et al.*, "Language models are few-shot learners," in *Advances in neural information processing systems*, 2020.

[53] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, *et al.*, "HuggingFace's transformers: State-of-the-art natural language processing," *arXiv preprint arXiv:1910.03771*, 2019.

[54] P. Budzianowski and I. Vulić, "Hello, it's GPT-2–how can I help you? Towards the use of pretrained language models for task-oriented dialogue systems," *arXiv preprint arXiv:1907.05774*, 2019.

[55] J.-S. Lee and J. Hsiang, "Patent claim generation by fine-tuning OpenAI GPT-2," *arXiv preprint arXiv:1907.02052*, 2019.

[56] H. Van, D. Kauchak, and G. Leroy, "AutoMeTS: The autocomplete for medical text simplification," *arXiv preprint arXiv:2010.10573*, 2020.

[57] T. Klein and M. Nabi, "Learning to answer by learning to ask: Getting the best of gpt-2 and bert worlds," *arXiv preprint arXiv:1911.02365*, 2019.

References (cont.)

[58] K. Ethayarajh, "How contextual are contextualized word representations? Comparing the geometry of BERT, ELMo, and GPT-2 embeddings," *arXiv preprint arXiv:1909.00512*, 2019.

[59] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-LM: Training multi-billion parameter language models using model parallelism," *arXiv preprint arXiv:1909.08053*, 2019.

[60] Microsoft, "Turing-NLG: A 17-billion-parameter language model by Microsoft." Microsoft Blog, 2020.

[61] K. Elkins and J. Chun, "Can GPT-3 pass a writerâs Turing test?," *Journal of Cultural Analytics*, vol. 2371, p. 4549, 2020.

[62] L. Floridi and M. Chiriatti, "GPT-3: Its nature, scope, limits, and consequences," *Minds and Machines*, pp. 1–14, 2020.

[63] D. Arpit, S. Jastrzebski, N. Ballas, D. Krueger, E. Bengio, M. S. Kanwal, T. Maharaj, A. Fischer, A. Courville, Y. Bengio, and S. Lacoste-Julien, "A closer look at memorization in deep networks," in *International Conference on Machine Learning*, 2017.

[64] G. Branwen, "GPT-3 creative fiction." `https://www.gwern.net/GPT-3`, 2020.

[65] B. Ghojogh and A. Ghodsi, "Attention mechanism, transformers, bert, and gpt: Tutorial and survey," 2020.