Tutorial on deep transformer

Loc Nguyen
Loc Nguyen's Academic Network, Vietnam
Email: ng_phloc@yahoo.com
Homepage: www.locnguyen.net

Abstract
Development of transformer is a far progressive step in the long journeys of both generative artificial intelligence (GenAI) and statistical translation machine (STM) with support of deep neural network (DNN), in which STM can be known as interesting result of GenAI because of encoder-decoder mechanism for sequence generation built in transformer. But why is transformer being preeminent in GenAI and STM? Firstly, transformer has a so-called self-attention mechanism that discovers contextual meaning of every token in sequence, which contributes to reduce ambiguousness. Secondly, transformer does not concern ordering of tokens in sequence, which allows to train transformer from many parts of sequences in parallel. Thirdly, the third reason which is result of the two previous reasons is that transformer can be trained from large corpus with high accuracy as well as highly computational performance. Moreover, transformer is implemented by DNN which is one of important and effective approaches in artificial intelligence (AI) in recent time. Although transformer is preeminent because of its good consistency, it is not easily understandable. Therefore, this technical report aims to describe transformer with explanations which are as easily understandable as possible.
Keywords: transformer, generative artificial intelligence, statistical translation machine, sequence generation, self-attention.

1. Introduction
Artificial intelligence (*AI*) is recent trend in technological world, especially in computer science, in which artificial neural network (ANN, NN) is one of important subjects of AI. Essentially, *ANN* models or implements a complicated function $\boldsymbol{y} = f(\boldsymbol{x})$ where $\boldsymbol{x} = (x_1, x_2, \ldots, x_m)^T$ and $\boldsymbol{y} = (y_1, y_2, \ldots, y_n)^T$ are vectors so that $\boldsymbol{x}$ and $\boldsymbol{y}$ are imitated by input layer and output layer of ANN, respectively with note that each layer is composed of units called neurons x_i, y_i. The complication degree of function $\boldsymbol{y} = f(\boldsymbol{x})$ is realized by hidden layers of ANN which are intermediated layers between input layer and output layer. We denote:

$$\boldsymbol{y} = f(\boldsymbol{x}|\Theta)$$

Where Θ denotes parameters of ANN which are often weights and biases. Because $f(\boldsymbol{x} \mid \Theta)$ is essentially vector-by-vector function whose input and output are vectors, it should have denoted as $\boldsymbol{f}(\boldsymbol{x} \mid \Theta)$ but it is still denoted as $f(\boldsymbol{x} \mid \Theta)$ for convenience and moreover, input $\boldsymbol{x}$ and output $\boldsymbol{y}$ will be matrices if their elements x_i and y_i are vectors. If there are many enough hidden layers, ANN becomes a so-called deep neural network (*DNN*) such that DNN is cornerstone of the main subject of this report which is transformer because transformer, as its name implies, is the highly abstract and complicated version of function $\boldsymbol{y} = f(\boldsymbol{x})$. In other words, a transformer will make the transformation between complex and different objects if it is implemented by DNN or set of DNNs according to viewpoint of DNN. Although transformer can be applied into many areas, especially machine translation and computer vision, this report focuses on statistical machine translation (*STM*) because complex and different objects $\boldsymbol{x}$ and $\boldsymbol{y}$ in STM transformer are two sentences in two different languages where $\boldsymbol{x}$ is source language sentence and $\boldsymbol{y}$ is target language sentence. If

ordering of elements x_i / y_i in vector $\boldsymbol{x}$ / $\boldsymbol{y}$ specifying sentence is concerned as ordering of words x_i / y_i in a sentence, transformer will relate to *sequence generation*. Therefore, transformer as well as STM are inspired from sequence generation which, in turn, relates to recurrent neural network (*RNN*) as well as long short-term memory (*LSTM*) because sequence generation models are often implemented by RNN or LSTM. The most standard ANN/DNN called feedforward network (*FFN*) follows the one-way direction from input layer to hidden layers to output layer without reverse direction, which means that there is neither connections from output layer to hidden layers nor connections from hidden layers to input layers. In other words, there is no cycle in FFN, which cause the side-effect that it is difficult to model a sequence vector $\boldsymbol{x} = (x_1, x_2,\ldots, x_m)^T$ like a sentence in natural language processing (NLP) because elements / words / terms / tokens x_i in such sequence/sentence vector have the same structure and every connection $x_i \rightarrow x_{i+1}$ of two successive words x_i and x_{i+1} is, actually, a cycle. This is the reason that recurrent neural network (RNN) is better than FFN to generate sequence. Therefore, we research transformer after researching sequence generation which is concerned after RNN is concerned. Note, sequence and sentence are two exchangeable concepts in this research.

Suppose entire FNN is reduced into a state in RNN and RNN is ordered list of neurons called sequence of neurons and moreover, output of previous neuron x_{i-1} contributes to input of current neuron x_i. Namely, for formal definition, given T time points $t = 1, 2,\ldots, T$, then RNN is ordered sequence of T states and each state is modeled by triple (x_t, h_t, o_t) called state (x_t, h_t, o_t) where x_t, h_t, and o_t represent input layer, hidden layer, and output layer, respectively. Without loss of generality, let x_t, h_t, and o_t represent input neuron, hidden neuron, and output neuron, respectively when a layer is represented by one of its neurons. Please pay attention that x_t, h_t, and o_t are represented vectors of the t^{th} word in sentence $\boldsymbol{x} = (x_1, x_2,\ldots, x_m)^T$ modeled by RNN in context of NLP because a word is modeled by a numeric vector in NLP. Therefore, the aforementioned sentence $\boldsymbol{x} = (x_1, x_2,\ldots, x_m)^T$ is a matrix indeed but $\boldsymbol{x}$ is mentioned as a vector. Exactly, $\boldsymbol{x}$ is vector of vectors, which leads to the convention that its elements are denoted by bold letter such as $\boldsymbol{x}_i$ or $\boldsymbol{x}_t$ because such elements are variable vectors representing words. Note, a word in NLP can be mentioned as term or token.

$$\boldsymbol{x} = \boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m)^T$$
$$\boldsymbol{y} = \boldsymbol{Y} = (\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_n)^T$$
$$\boldsymbol{y} = \boldsymbol{Y} = f(\boldsymbol{x}|\Theta) = f(\boldsymbol{X}|\Theta)$$

Note, the subscript "T" denotes vector/matrix transposition operator. Whether the sentence / sequence is denoted as vector notation $\boldsymbol{x}$ or matrix notation $\boldsymbol{X}$ belongs to contextual explanations. Recall that transformer as well as STM are inspired from sequence generation which, in turn, is related to recurrent neural network (RNN) as well as long short-term memory (LSTM) because sequence generation models are often implemented by RNN or LSTM. Function $\boldsymbol{y} = f(\boldsymbol{x} \mid \Theta)$ implemented by DNNs such as RNN and LSTM is also called generator because it is sequence generation model indeed. Therefore, although transformer is different from RNN and LSTM, all of them are denoted by generator $\boldsymbol{y} = f(\boldsymbol{x} \mid \Theta)$ because they are sequence generation models indeed.

The t^{th} element/word in sequence/sentence $\boldsymbol{x} = (\boldsymbol{x}_1, \boldsymbol{x}_2,\ldots, \boldsymbol{x}_m)^T$ is represented by the t^{th} state $(\boldsymbol{x}_t, \boldsymbol{h}_t, \boldsymbol{o}_t)$ of RNN where $\boldsymbol{x}_t$ is the t^{th} input word and $\boldsymbol{o}_t$ is the t^{th} output word. If RNN models $\boldsymbol{x} = (\boldsymbol{x}_1, \boldsymbol{x}_2,\ldots, \boldsymbol{x}_m)^T$, then $T = m$ and so, if RNN models $\boldsymbol{y} = (\boldsymbol{y}_1, \boldsymbol{y}_2,\ldots, \boldsymbol{y}_n)^T$, then $T = n$. By a convention, word and sentence are mentioned as token and sequence, respectively. Moreover, $\boldsymbol{x}$ is called source sequence and $\boldsymbol{y}$ is called target sequence or generated sequence. Mathematical equation to update RNN is specified as follows (Wikipedia, Recurrent neural network, 2005):

$$\boldsymbol{h}_t = \sigma_{\boldsymbol{h}}(\boldsymbol{W}_{\boldsymbol{h}}\boldsymbol{x}_t + \boldsymbol{U}_{\boldsymbol{h}}\boldsymbol{h}_{t-1} + \boldsymbol{b}_{\boldsymbol{h}})$$
$$\boldsymbol{o}_t = \sigma_{\boldsymbol{o}}(\boldsymbol{W}_{\boldsymbol{o}}\boldsymbol{h}_t + \boldsymbol{b}_{\boldsymbol{o}})$$

Where $\boldsymbol{W}_{\boldsymbol{h}}$, $\boldsymbol{U}_{\boldsymbol{h}}$, and $\boldsymbol{W}_{\boldsymbol{o}}$ are weight matrices of current hidden neuron $\boldsymbol{h}_t$, previous hidden neuron $\boldsymbol{h}_{t-1}$, and current output neuron $\boldsymbol{o}_t$, respectively whereas $\boldsymbol{b}_{\boldsymbol{h}}$ and $\boldsymbol{b}_{\boldsymbol{o}}$ are bias vectors of $\boldsymbol{h}_t$ and $\boldsymbol{o}_t$, respectively. Moreover, $\sigma_{\boldsymbol{h}}(.)$ and $\sigma_{\boldsymbol{o}}(.)$ are activation functions of $\boldsymbol{h}_t$ and $\boldsymbol{o}_t$, respectively, which are vector-by-vector functions.

RNN copes with the problem of vanishing gradient when learning a long RNN of many states and so, long short-term memory (LSTM) is proposed to restrict the problem of vanishing gradient. State in RNN becomes cell in LSTM and so, given T time points $t = 1, 2, \ldots, T$, let the pair $(\boldsymbol{c}_t, \boldsymbol{h}_t)$ denote LSTM cell at current time point t where $\boldsymbol{c}_t$ represents real information stored in memory and $\boldsymbol{h}_t$ represents clear-cut information that propagates through next time points. A cell $(\boldsymbol{c}_t, \boldsymbol{h}_t)$ has four gates such as forget gate $\boldsymbol{f}_t$, input gate $\boldsymbol{i}_t$, output gate $\boldsymbol{o}_t$, and cell gate $\boldsymbol{g}_t$. At every time point t or every iteration t, cell $(\boldsymbol{c}_t, \boldsymbol{h}_t)$ updates its information based on these gates as follows:

$$\boldsymbol{f}_t = \sigma_{\boldsymbol{f}}\left(\boldsymbol{W}_{\boldsymbol{f}}\boldsymbol{x}_t + \boldsymbol{U}_{\boldsymbol{f}}\boldsymbol{h}_{t-1} + \boldsymbol{b}_{\boldsymbol{f}}\right)$$
$$\boldsymbol{i}_t = \sigma_{\boldsymbol{i}}(\boldsymbol{W}_{\boldsymbol{i}}\boldsymbol{x}_t + \boldsymbol{U}_{\boldsymbol{i}}\boldsymbol{h}_{t-1} + \boldsymbol{b}_{\boldsymbol{i}})$$
$$\boldsymbol{o}_t = \sigma_{\boldsymbol{o}}(\boldsymbol{W}_{\boldsymbol{o}}\boldsymbol{x}_t + \boldsymbol{U}_{\boldsymbol{o}}\boldsymbol{h}_{t-1} + \boldsymbol{b}_{\boldsymbol{o}})$$
$$\boldsymbol{g}_t = \sigma_{\boldsymbol{g}}\left(\boldsymbol{W}_{\boldsymbol{g}}\boldsymbol{x}_t + \boldsymbol{U}_{\boldsymbol{g}}\boldsymbol{h}_{t-1} + \boldsymbol{b}_{\boldsymbol{g}}\right)$$
$$\boldsymbol{c}_t = \boldsymbol{f}_t \otimes \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \otimes \boldsymbol{g}_t$$
$$\boldsymbol{h}_t = \boldsymbol{o}_t \otimes \sigma_{\boldsymbol{h}}(\boldsymbol{c}_t)$$

Note, $\boldsymbol{W}_{(.)}$ and $\boldsymbol{U}_{(.)}$ are weight matrices whereas $\boldsymbol{b}_{(.)}$ are bias vectors, which are parameters. Because core information of cell $(\boldsymbol{c}_t, \boldsymbol{h}_t)$ including $\boldsymbol{c}_t$ and $\boldsymbol{h}_t$ is calculated without any parameters, the problem of vanishing gradient can be alleviated when such gradient is calculated with regard to parameters such as weight matrices and bias vectors.

In general, when a sequence is modeled by a RNN or a LSTM, it is possible to generate a new sequence after RNN or LSTM is trained by backpropagation algorithm associated with stochastic gradient descent (*SGD*) algorithm. In other words, RNN and LSTM are important generation models although transformer is the main subject in this report because STM is, essentially, a sequence generation model that generates a new sentence in target language from a sentence in source language when sentence in NLP is represented by sequence. Because RNN and LSTM have the same methodological ideology, RNN is mentioned rather than LSTM because RNN is simpler one but they can be applied by exchangeable manner. For instance, given simplest case that source sequence $\boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m)^T$ and target sequence also called generated sequence $\boldsymbol{Y} = (\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_n)^T$ have the same length $m = n$.

$$\boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n)^T$$
$$\boldsymbol{Y} = (\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_n)^T = f(\boldsymbol{X}|\Theta)$$

Generation model $f(\boldsymbol{x} \mid \Theta)$ is implemented by a RNN of n states $(\boldsymbol{x}_t, \boldsymbol{h}_t, \boldsymbol{o}_t)$ so that $\boldsymbol{o}_t = \boldsymbol{y}_t$ for all t from 1 to n. After RNN was trained from sample by backpropagation algorithm associated with SGD, given source sequence $\boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n)^T$, target sequence $\boldsymbol{Y} = (\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_n)^T$ is generated easily by evaluating n states of RNN.

$$\begin{matrix} \boldsymbol{h}_t = \sigma_{\boldsymbol{h}}(\boldsymbol{W}_{\boldsymbol{h}}\boldsymbol{x}_t + \boldsymbol{U}_{\boldsymbol{h}}\boldsymbol{h}_{t-1} + \boldsymbol{b}_{\boldsymbol{h}}) \\ \boldsymbol{y}_t = \boldsymbol{o}_t = \sigma_{\boldsymbol{o}}(\boldsymbol{W}_{\boldsymbol{o}}\boldsymbol{h}_t + \boldsymbol{b}_{\boldsymbol{o}}) \end{matrix}, \forall t = \overline{1, n}$$

Such generation process with n-state RNN is depicted by following figure:

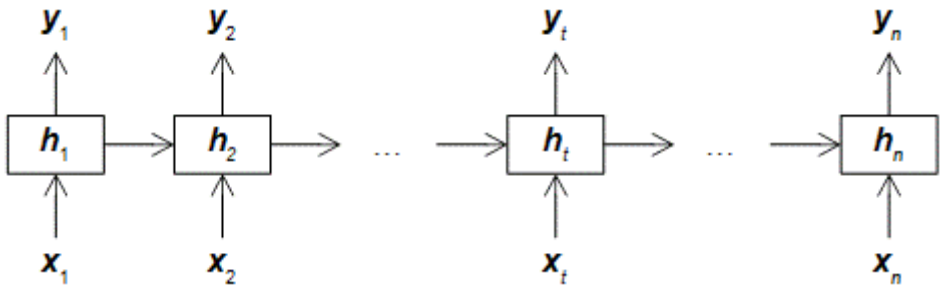

Figure 1.1. RNN generation model

The next section will focus on sequence generation and attention which is a mechanism that improves generation process.

2. Sequence generation and attention

Recall that transformer as well as statistical machine translation (STM) are inspired from sequence generation which, in turn, is related to recurrent neural network (RNN) as well as long short-term memory (LSTM) because sequence generation models are often implemented by RNN or LSTM. Function $\boldsymbol{y} = f(\boldsymbol{x} \mid \Theta)$ implemented by DNNs such as RNN and LSTM is also called generator because it is sequence generation model indeed. Because RNN and LSTM have the same methodological ideology, RNN is mentioned rather than LSTM.

$$\boldsymbol{x} = \boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_m)^T$$
$$\boldsymbol{y} = \boldsymbol{Y} = (\boldsymbol{y}_1, \boldsymbol{y}_2, \dots, \boldsymbol{y}_n)^T$$
$$\boldsymbol{y} = \boldsymbol{Y} = f(\boldsymbol{x}|\Theta) = f(\boldsymbol{X}|\Theta)$$

Note, Θ denotes parameters of ANN which are often weights and biases whereas sequence is denoted as vector notation $\boldsymbol{x}$ or matrix notation $\boldsymbol{X}$ belongs to contextual explanations. This section focuses on sequence generation models such as RNN and LSTM before mentioning advanced concepts of transformer because, anyhow, transformer is next evolutional step of sequence generation models, especially in STM and natural language processing (NLP).

Given simplest case aforementioned that source sequence $\boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_m)^T$ and target sequence also called generated sequence $\boldsymbol{Y} = (\boldsymbol{y}_1, \boldsymbol{y}_2, \dots, \boldsymbol{y}_n)^T$ have the same length $m = n$.

$$\boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_n)^T$$
$$\boldsymbol{Y} = (\boldsymbol{y}_1, \boldsymbol{y}_2, \dots, \boldsymbol{y}_n)^T = f(\boldsymbol{X}|\Theta)$$

Generation model $f(\boldsymbol{X} \mid \Theta)$ is implemented by a RNN of n states $(\boldsymbol{x}_t, \boldsymbol{h}_t, \boldsymbol{o}_t)$ so that $\boldsymbol{o}_t = \boldsymbol{y}_t$ for all t from 1 to n. After RNN was trained from sample by backpropagation algorithm associated with stochastic gradient descent (SGD) algorithm, given source sequence $\boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_n)^T$, target sequence $\boldsymbol{Y} = (\boldsymbol{y}_1, \boldsymbol{y}_2, \dots, \boldsymbol{y}_n)^T$ is generated easily by evaluating n states of RNN.

$$\boldsymbol{h}_t = \sigma_{\boldsymbol{h}}(\boldsymbol{W}_{\boldsymbol{h}}\boldsymbol{x}_t + \boldsymbol{U}_{\boldsymbol{h}}\boldsymbol{h}_{t-1} + \boldsymbol{b}_{\boldsymbol{h}})$$
$$\boldsymbol{y}_t = \boldsymbol{o}_t = \sigma_{\boldsymbol{o}}(\boldsymbol{W}_{\boldsymbol{o}}\boldsymbol{h}_t + \boldsymbol{b}_{\boldsymbol{o}})$$

The simplest RNN generation needs to be extended if source sequence $\boldsymbol{X}$ is incomplete, for example, $\boldsymbol{X}$ has k token vectors $\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_k$ where $k < n$. When $\boldsymbol{X}$ is incomplete, without loss of generality, given current output $\boldsymbol{y}_t$, it is necessary to predict the next output $\boldsymbol{x}_{t+1}$ (with suppose $t > k$). The prediction process, proposed by Graves (Graves, 2014), is based on estimating the predictive probability $P(\boldsymbol{x}_{t+1} \mid \boldsymbol{y}_t)$ which is conditional probability of next input $\boldsymbol{x}_{t+1}$ given current output $\boldsymbol{y}_t$. As a result, RNN generation model is extended as follows (Graves, 2014, p. 4):

$$\boldsymbol{h}_t = \sigma_{\boldsymbol{h}}(\boldsymbol{W}_{\boldsymbol{h}}\boldsymbol{x}_t + \boldsymbol{U}_{\boldsymbol{h}}\boldsymbol{h}_{t-1} + \boldsymbol{b}_{\boldsymbol{h}})$$
$$\boldsymbol{y}_t = \sigma_{\boldsymbol{o}}(\boldsymbol{W}_{\boldsymbol{o}}\boldsymbol{h}_t + \boldsymbol{b}_{\boldsymbol{o}})$$
$$P(\boldsymbol{x}_{t+1}|\boldsymbol{y}_t)$$

Following figure depicts the prediction model proposed by Graves (Graves, 2014, p. 3):

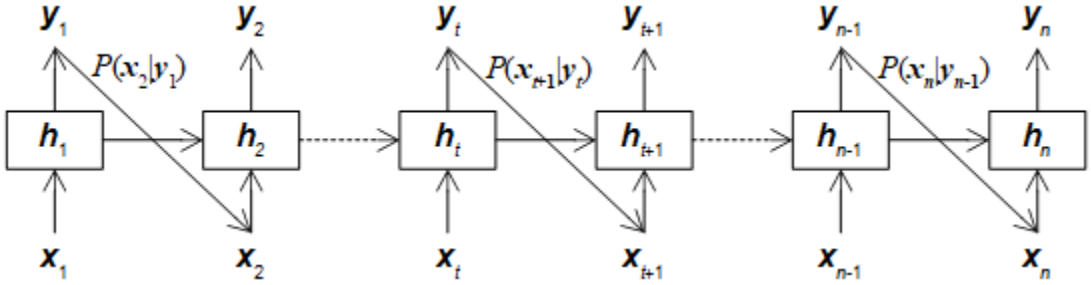

Figure 2.1. RNN prediction model

The problem here is how to specify predictive probability $P(\boldsymbol{x}_{t+1} \mid \boldsymbol{y}_t)$. In the most general form, suppose joint probability $P(\boldsymbol{x}_{t+1}, \boldsymbol{y}_t)$ is parameterized by multivariate normal distribution with mean vector μ and covariance matrix Σ.

$$P(\boldsymbol{x}_{t+1}, \boldsymbol{y}_t) = \mathcal{N}(\boldsymbol{x}_{t+1}, \boldsymbol{y}_t | \mu, \Sigma)$$

$$\mu = \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix}$$

$$\Sigma = \begin{pmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{pmatrix}$$

It is easy to estimate μ and Σ to determine $P(\boldsymbol{x}_{t+1}, \boldsymbol{y}_t)$ from sample by maximum likelihood estimation (MLE) method, for instance. Consequently, predictive probability $P(\boldsymbol{x}_{t+1} \mid \boldsymbol{y}_t)$ is determined based on joint probability $P(\boldsymbol{x}_{t+1}, \boldsymbol{y}_t)$ as multivariate normal distribution with mean vector μ_{12} and covariance matric Σ_{12} specified as follows (Hardle & Simar, 2013, p. 157):

$$P(\boldsymbol{x}_{t+1} | \boldsymbol{y}_t) = \mathcal{N}(\boldsymbol{x}_{t+1} | \mu_{12}, \Sigma_{12})$$

$$\mu_{12} = \mu_1 + \Sigma_{12}\Sigma_{22}^{-1}(\boldsymbol{y}_t - \mu_2)$$

$$\Sigma_{12} = \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21}$$

Because predictive probability $P(\boldsymbol{x}_{t+1} \mid \boldsymbol{y}_t)$ gets highest at the mean μ_{12}, it is possible to estimate $\boldsymbol{x}_{t+1}$ given $\boldsymbol{y}_t$ by μ_{12}.

$$\boldsymbol{h}_t = \sigma_{\boldsymbol{h}}(\boldsymbol{W}_{\boldsymbol{h}}\boldsymbol{x}_t + \boldsymbol{U}_{\boldsymbol{h}}\boldsymbol{h}_{t-1} + \boldsymbol{b}_{\boldsymbol{h}})$$

$$\boldsymbol{y}_t = \sigma_{\boldsymbol{o}}(\boldsymbol{W}_{\boldsymbol{o}}\boldsymbol{h}_t + \boldsymbol{b}_{\boldsymbol{o}})$$

$$\boldsymbol{x}_{t+1} = \mu_{12} = \mu_1 + \Sigma_{12}\Sigma_{22}^{-1}(\boldsymbol{y}_t - \mu_2)$$

The generation model above has only one RNN because source sequence $\boldsymbol{X}$ and target sequence $\boldsymbol{Y}$ have the same length. Some real applications, especially STM applications, require that lengths of $\boldsymbol{X}$ and $\boldsymbol{Y}$ are different, $m \neq n$. This problem is called different-length problem.

$$\boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_m)^T$$

$$\boldsymbol{Y} = (\boldsymbol{y}_1, \boldsymbol{y}_2, \dots, \boldsymbol{y}_n)^T$$

Solution for different-length problem is to specify two RNNs: a RNN called *encoder* for $\boldsymbol{X}$ generation and the other one called *decoder* for $\boldsymbol{Y}$ generation. Intermediate vector $\boldsymbol{a}$ is proposed to connect encoder and decoder, which is called context vector in literature (Cho, et al., 2014, p. 2). The encoder-decoder mechanism is an important progressive step in STM as well as generative artificial intelligence (GenAI) because there is no requirement of mapping token-by-token between two sequences $\boldsymbol{X}$ and $\boldsymbol{Y}$, which is much more important than solving the different-length problem. On the other hand, sequence generation as well as its advanced development – transformer can also be classified into domain of GenAI.

According to Cho et al. (Cho, et al., 2014), context variable $\boldsymbol{a}$, which is last output of encoder, becomes input of decoder. Following figure depicts encoder-decoder model proposed by Cho et al. (Cho, et al., 2014, p. 2) with note that context vector $\boldsymbol{a}$ has fixed length.

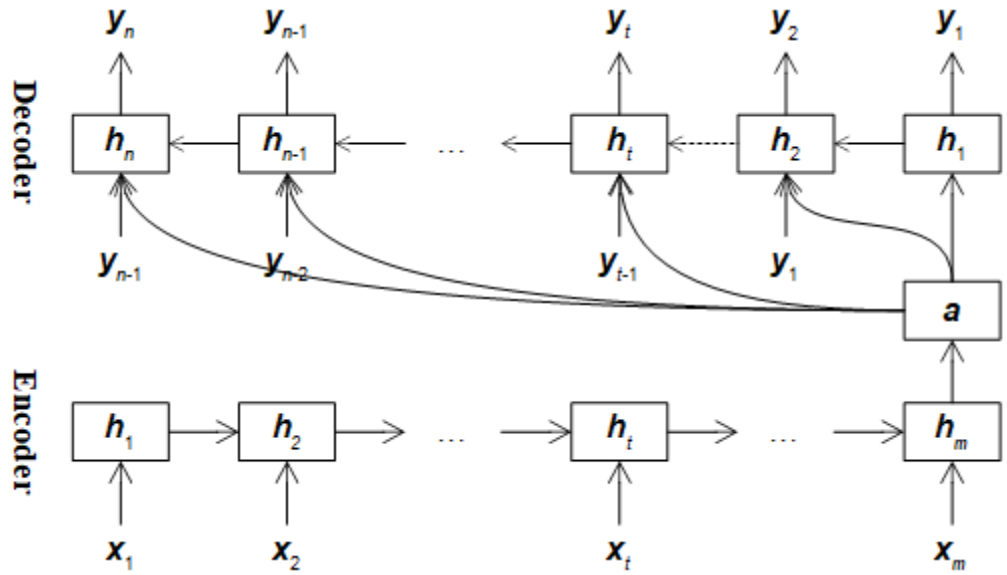

Figure 2.2. Encoder-decoder model with fixed-length context

Note, both context and current token t are inputs of next token t+1. Moreover, there is an assignment $\boldsymbol{y}_{t+1} = \boldsymbol{o}_t$. Therefore, each t^{th} state of decoder is modified as follows:

$$\boldsymbol{h}_t = \sigma_{\boldsymbol{h}}(\boldsymbol{W}_{\boldsymbol{h}}\boldsymbol{y}_t + \boldsymbol{U}_{\boldsymbol{h}}\boldsymbol{h}_{t-1} + \boldsymbol{V}_{\boldsymbol{h}}\boldsymbol{a} + \boldsymbol{b}_{\boldsymbol{h}})$$

$$\boldsymbol{y}_{t+1} = \hat{\boldsymbol{y}}_{t+1} = \boldsymbol{o}_t = \sigma_{\boldsymbol{o}}(\boldsymbol{W}_{\boldsymbol{o}}\boldsymbol{h}_t + \boldsymbol{b}_{\boldsymbol{o}})$$

Where $\boldsymbol{V}_{\boldsymbol{h}}$ is weight matrix for context variable $\boldsymbol{a}$. Moreover, it may be not required to calculate output for each t^{th} state of encoder. It may be only necessary to calculate hidden value of encoder.

$$\boldsymbol{h}_t^{\text{encoder}} = \sigma_{\boldsymbol{h}}^{\text{encoder}}\left(\boldsymbol{W}_{\boldsymbol{h}}^{\text{encoder}}\boldsymbol{x}_t + \boldsymbol{U}_{\boldsymbol{h}}^{\text{encoder}}\boldsymbol{h}_{t-1}^{\text{encoder}} + \boldsymbol{b}_{\boldsymbol{h}}^{\text{encoder}}\right)$$

In STM, given source sequence $\boldsymbol{X}$ and t target tokens $\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_t$, it is necessary to predict the next target token $\boldsymbol{y}_{t+1}$. In other words, predictive probability $P(\boldsymbol{y}_{t+1} \mid \Theta, \boldsymbol{X}, \boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_t)$ needs to be maximized so as to obtain $\boldsymbol{y}_{t+1}$. Predictive probability $P(\boldsymbol{y}_{t+1} \mid \Theta, \boldsymbol{X}, \boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_t)$ is called *likelihood* at the t^{th} state of decoder. Consequently, parameter Θ of encoder-decoder model is maximizer of such likelihood.

$$\hat{\Theta} = \underset{\Theta}{\operatorname{argmax}}\, P(\boldsymbol{y}_{t+1} | \Theta, \boldsymbol{X}, \boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_t)$$

Note, parameter Θ represents weight matrices and biases of RNN. By support of RNN and context vector $\boldsymbol{a}$ with implication of Markov property, likelihood $P(\boldsymbol{y}_{t+1} \mid \Theta, \boldsymbol{X}, \boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_t)$ can become simpler:

$$P(\boldsymbol{y}_{t+1} | \Theta, \boldsymbol{X}, \boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_t) = P(\boldsymbol{y}_{t+1} | \Theta, \boldsymbol{X}, \boldsymbol{y}_{\leq t}) = P(\boldsymbol{y}_{t+1} | \Theta, \boldsymbol{a}, \boldsymbol{y}_t) = P(\boldsymbol{o}_t | \Theta, \boldsymbol{a}, \boldsymbol{y}_t)$$

Likelihood $P(\boldsymbol{y}_{t+1} \mid \Theta, \boldsymbol{X}, \boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_t)$, which represents statistical language model, is object of maximum likelihood estimation (MLE) method for training encoder-decoder model (Cho, et al., 2014, p. 2). For example, the likelihood can be approximated by standard normal distribution, which is equivalent to square error function, as follows:

$$P(\boldsymbol{y}_{t+1} | \Theta, \boldsymbol{X}, \boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_t) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{(f(\boldsymbol{X}, \boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_t | \Theta) - \boldsymbol{y}_{t+1})^2}{2}\right)$$

Where $f(\boldsymbol{X}, \boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_t \mid \Theta)$ denotes encoder-decoder chain.

$$f(\boldsymbol{X}, \boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_t | \Theta) = \hat{\boldsymbol{y}}_{t+1} = \boldsymbol{o}_t = \sigma_{\boldsymbol{o}}(\boldsymbol{W}_{\boldsymbol{o}}\boldsymbol{h}_t + \boldsymbol{b}_{\boldsymbol{o}})$$

Therefore, training encoder-decoder model begins with MLE associated with backpropagation algorithm and SGD from decoder back to encoder.

$$\hat{\Theta} = \underset{\Theta}{\operatorname{argmax}} \log\left(P(\boldsymbol{y}_{t+1} | \Theta, \boldsymbol{X}, \boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_t)\right) = \underset{\Theta}{\operatorname{argmin}} (f(\boldsymbol{X}, \boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_t | \Theta) - \boldsymbol{y}_{t+1})^2$$

Alternately, in STM with predefined word vocabulary, a simple but effective way to train encoder-decoder model is to replace likelihood $P(\boldsymbol{y}_{t+1} \mid \Theta, \boldsymbol{X}, \boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_t)$ by a so-called linear component which is a feedforward network (FFN). Exactly, FNN maps the $(t+1)^{\text{th}}$ target token specified by token vector $\boldsymbol{y}_{t+1}$ to a weight vector $\boldsymbol{w}$ whose each element w_i ($0 \leq w_i \leq 1$) is weight of i^{th} token (Alammar, 2018).

$$\boldsymbol{w} = \left(w_1, w_2, \ldots, w_{|\Omega|}\right)^T = \text{linear}(\boldsymbol{y}_{t+1}) = \text{FFN}(\boldsymbol{y}_{t+1}) = \text{FFN}(\boldsymbol{o}_t)$$

Length of weight vector $\boldsymbol{w}$ is the cardinality $|\Omega|$ where Ω is the vocabulary containing all tokens. After token weight vector $\boldsymbol{w}$ is determined, it is easily converted into output probability vector $\boldsymbol{p} = (p_1, p_2, \ldots, p_{|\Omega|})^T$ where each element p_i is probability of the i^{th} token in vocabulary given the $(t+1)^{\text{th}}$ target token (Alammar, 2018).

$$\boldsymbol{p} = \text{softmax}(\boldsymbol{w}) = \left(p_1, p_2, \ldots, p_{|\Omega|}\right)^T$$

$$p_i = \frac{\exp(w_i)}{\sum_{l=1}^{|\Omega|} \exp(w_l)}$$

Following figure depicts linear component.

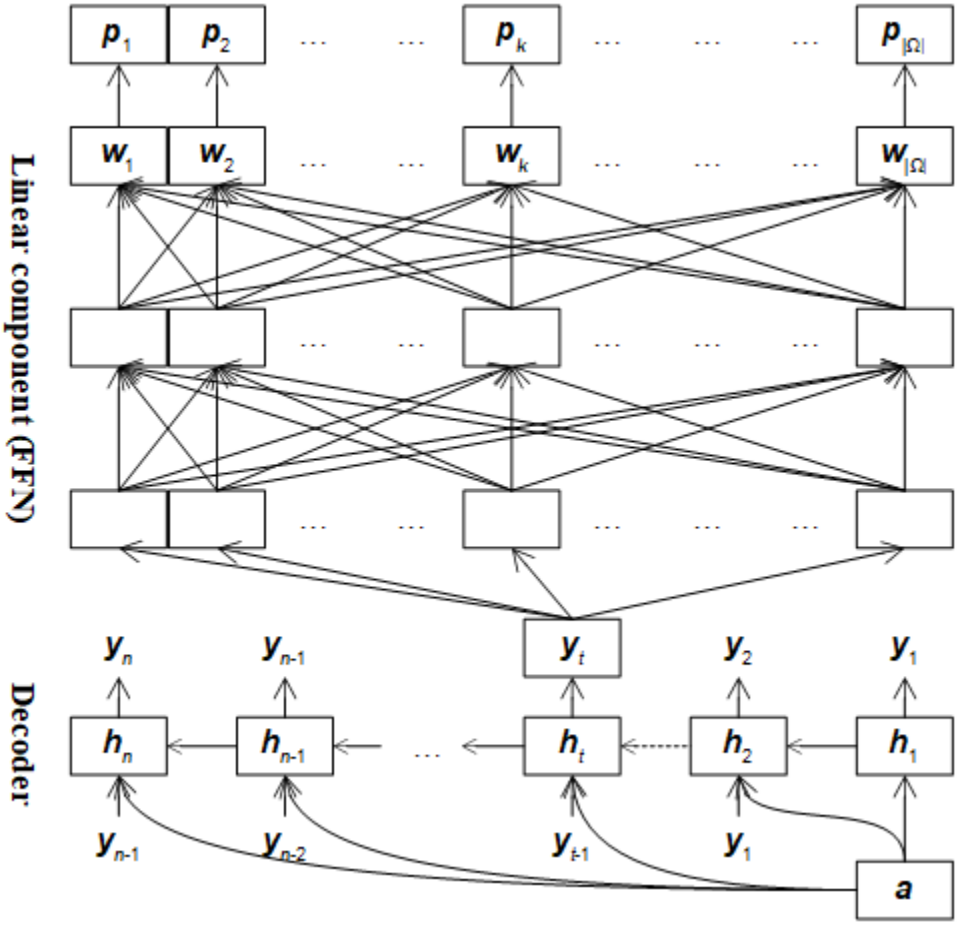

Figure 2.3. Linear component of encoder-decoder model

It is interesting that likelihood $P(\boldsymbol{y}_{t+1} \mid \Theta, \boldsymbol{X}, \boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_t)$ can be defined as output probability vector $\boldsymbol{p} = (p_1, p_2, \ldots, p_{|\Omega|})^T$. If the i^{th} token is issued, its probability p_t is 1 and other probabilities are 0.

$$P(\boldsymbol{y}_{t+1} | \Theta, \boldsymbol{X}, \boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_t) \stackrel{\text{def}}{=} \left(p_1, p_2, \ldots, p_{|\Omega|}\right)^T$$

Consequently, training encoder-decoder model begins with training linear component FFN($\boldsymbol{y}_{t+1}$) back to training decoder back to training encoder, which follows backpropagation algorithm

associated stochastic gradient descent (SGD) method. Concretely, the following *cross-entropy* $L(\boldsymbol{p} \mid \Theta)$ is minimized so as to train FFN($\boldsymbol{y}_{t+1}$).

$$L(\boldsymbol{p}|\Theta) = -\sum_{i=1}^{|\Omega|} q_i \log(p_i)$$

Where Θ is parameter of FFN($\boldsymbol{y}_{t+1}$) and the vector $\boldsymbol{q} = (q_1, q_2, \ldots, q_{|\Omega|})^T$ is binary vector from sample whose each element q_i has binary values $\{0, 1\}$ indicating whether the i^{th} token/word exists. For example, give sequence/sentence ("I", "am", "a", "student")^T, if there is only one token/word "I" in sample sentence, the binary vector will be $\boldsymbol{q} = (1, 0, 0, 0)^T$. If three words "I", "am", and "student" are mutually existent, the binary vector will be $\boldsymbol{q} = (1, 1, 0, 1)^T$. When SGD is applied into minimizing the cross-entropy, partial gradient of $L(\boldsymbol{p} \mid \Theta)$ with regard to w_j is:

$$\frac{\partial L(\boldsymbol{p}|\Theta)}{\partial w_j} = -\sum_{i=1}^{|\Omega|} q_i \left(\delta_{ij} - p_j\right)$$

Where,

$$\delta_{ij} = \begin{cases} 1 \text{ if } i = j \\ 0 \text{ if } i \neq j \end{cases}$$

Proof,

$$\frac{\partial L(\boldsymbol{p}|\Theta)}{\partial w_j} = -\sum_{i=1}^{|\Omega|} q_i \frac{\partial \log(p_i)}{\partial w_j} = -\sum_{i=1}^{|\Omega|} q_i \frac{d\log(p_i)}{dp_i} \frac{\partial p_i}{\partial w_j} = -\sum_{i=1}^{|\Omega|} q_i \frac{1}{p_i} \frac{\partial p_i}{\partial w_j}$$

Due to:

$$\frac{\partial p_i}{\partial w_j} = \frac{\partial}{\partial w_j}\left(\frac{\exp(w_i)}{\sum_{l=1}^{|\Omega|} \exp(w_l)}\right) = p_i\left(\delta_{ij} - p_j\right)$$

We obtain:

$$\frac{\partial L(\boldsymbol{p}|\Theta)}{\partial w_j} = -\sum_{i=1}^{|\Omega|} q_i \left(\delta_{ij} - p_j\right) \blacksquare$$

So that gradient of $L(\boldsymbol{p} \mid \Theta)$ with regard to $\boldsymbol{w}$ is:

$$\frac{dL(\boldsymbol{p}|\Theta)}{d\boldsymbol{w}} = \left(\frac{\partial L(\boldsymbol{p}|\Theta)}{\partial w_1}, \frac{\partial L(\boldsymbol{p}|\Theta)}{\partial w_2}, \ldots, \frac{\partial L(\boldsymbol{p}|\Theta)}{\partial w_{|\Omega|}}\right)^T$$

Therefore, parameter Θ is updated according to SGD associated with backpropagation algorithm:

$$\Theta = \Theta - \gamma \frac{dL(\boldsymbol{p}|\Theta)}{d\boldsymbol{w}}$$

Where γ ($0 < \gamma \leq 1$) is *learning rate*. Please pay attention that ordering of source tokens is set from the end token back to the beginning token so that null tokens specified by zero vectors are always in the opening of sequence.

When encoder-decoder model is developed, context vector $\boldsymbol{a}$ becomes a so-called *attention*. The main difference between context vector and attention vector is that attention vector is calculated dynamically (customized) for each decoder state. Moreover, that context vector has fixed length restricts its prospect. Anyhow, attention mechanism fosters target sequence to pay attention to source sequence. In general, attention of a decoder state (token) is weighted sum of all encoder states (tokens) with regard to such decoder state. Suppose encoder RNN is denoted as follows:

$$\boldsymbol{h}_t^{\text{encoder}} = \sigma_{\boldsymbol{h}}^{\text{encoder}}\left(\boldsymbol{W}_{\boldsymbol{h}}^{\text{encoder}} \boldsymbol{x}_t + \boldsymbol{U}_{\boldsymbol{h}}^{\text{encoder}} \boldsymbol{h}_{t-1}^{\text{encoder}} + \boldsymbol{b}_{\boldsymbol{h}}^{\text{encoder}}\right)$$

$$\boldsymbol{o}_t^{\text{encoder}} = \sigma_{\boldsymbol{o}}^{\text{encoder}}\left(\boldsymbol{W}_{\boldsymbol{o}}^{\text{encoder}}\boldsymbol{h}_t^{\text{encoder}} + \boldsymbol{b}_{\boldsymbol{o}}^{\text{encoder}}\right)$$

For convenience, let $\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_m$ denote m outputs of encoder such that:

$$\boldsymbol{s}_1 = \boldsymbol{o}_1^{\text{encoder}}$$
$$\boldsymbol{s}_2 = \boldsymbol{o}_2^{\text{encoder}}$$
$$\vdots$$
$$\boldsymbol{s}_m = \boldsymbol{o}_m^{\text{encoder}}$$

Let score($\boldsymbol{s}_i$, $\boldsymbol{h}_t$) be score of encoder output $\boldsymbol{s}_i$ and decoder hidden $\boldsymbol{h}_t$ where score($\boldsymbol{s}_i$, $\boldsymbol{h}_t$) measures how much the i^{th} token of source sequence modeled by encoder is close to the t^{th} token of target sequence modeled by decoder. As usual, score($\boldsymbol{s}_i$, $\boldsymbol{h}_t$) is defined as dot product of $\boldsymbol{s}_i$ and $\boldsymbol{h}_t$ (Voita, 2023).

$$\text{score}(\boldsymbol{s}_i, \boldsymbol{h}_t) = \boldsymbol{s}_i^T \boldsymbol{h}_t$$

Where decoder hidden $\boldsymbol{h}_t$ is:

$$\boldsymbol{h}_t = \sigma_{\boldsymbol{h}}(\boldsymbol{W}_{\boldsymbol{h}}\boldsymbol{y}_t + \boldsymbol{U}_{\boldsymbol{h}}\boldsymbol{h}_{t-1} + \boldsymbol{b}_{\boldsymbol{h}})$$

Let weight($\boldsymbol{s}_i$, $\boldsymbol{h}_t$) be weight of encoder output $\boldsymbol{s}_i$ and decoder hidden $\boldsymbol{h}_t$ over m states of encoder, which is calculated based on soft-max function (Voita, 2023):

$$\text{weight}(\boldsymbol{s}_i, \boldsymbol{h}_t) = \frac{\exp\left(\text{score}(\boldsymbol{s}_i, \boldsymbol{h}_t)\right)}{\sum_{j=1}^{m} \exp\left(\text{score}\left(\boldsymbol{s}_j, \boldsymbol{h}_t\right)\right)}$$

As a result, let $\boldsymbol{a}_t$ be attention of source sequence $\boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n)^T$ with regard to the t^{th} token of target sequence $\boldsymbol{Y} = (\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_n)^T$, which is weighted sum of all encoder outputs with regard to such t^{th} target token (Voita, 2023).

$$\boldsymbol{a}_t = \sum_{i=1}^{m} \text{weight}(\boldsymbol{s}_i, \boldsymbol{h}_t)\boldsymbol{s}_i$$

Obviously, $\boldsymbol{a}_t$ becomes one of inputs of the t^{th} token of target sequence $\boldsymbol{Y} = (\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_n)^T$ such that:

$$\boldsymbol{o}_t = \sigma_{\boldsymbol{o}}(\boldsymbol{W}_{\boldsymbol{o}}\boldsymbol{h}_t + \boldsymbol{V}_{\boldsymbol{o}}\boldsymbol{a}_t + \boldsymbol{b}_{\boldsymbol{o}})$$

Where $\boldsymbol{V}_{\boldsymbol{o}}$ is weight matrix of attention $\boldsymbol{a}_t$. In general, decoder RNN associated with the attention mechanism called Luong attention (Voita, 2023) is specified as follows:

$$\boldsymbol{h}_t = \sigma_{\boldsymbol{h}}(\boldsymbol{W}_{\boldsymbol{h}}\boldsymbol{y}_t + \boldsymbol{U}_{\boldsymbol{h}}\boldsymbol{h}_{t-1} + \boldsymbol{b}_{\boldsymbol{h}})$$
$$\boldsymbol{a}_t = \sum_{i=1}^{m} \text{weight}(\boldsymbol{s}_i, \boldsymbol{h}_t)\boldsymbol{s}_i$$
$$\boldsymbol{y}_{t+1} = \widehat{\boldsymbol{y}}_{t+1} = \boldsymbol{o}_t = \sigma_{\boldsymbol{o}}(\boldsymbol{W}_{\boldsymbol{o}}\boldsymbol{h}_t + \boldsymbol{V}_{\boldsymbol{o}}\boldsymbol{a}_t + \boldsymbol{b}_{\boldsymbol{o}})$$

Where,

$$\boldsymbol{s}_i = \boldsymbol{o}_i^{\text{encoder}}$$
$$\text{score}(\boldsymbol{s}_i, \boldsymbol{h}_t) = \boldsymbol{s}_i^T \boldsymbol{h}_t$$
$$\text{weight}(\boldsymbol{s}_i, \boldsymbol{h}_t) = \frac{\exp\left(\text{score}(\boldsymbol{s}_i, \boldsymbol{h}_t)\right)}{\sum_{j=1}^{m} \exp\left(\text{score}\left(\boldsymbol{s}_j, \boldsymbol{h}_t\right)\right)}$$

Following figure depicts encoder-decoder model with attention (Voita, 2023):

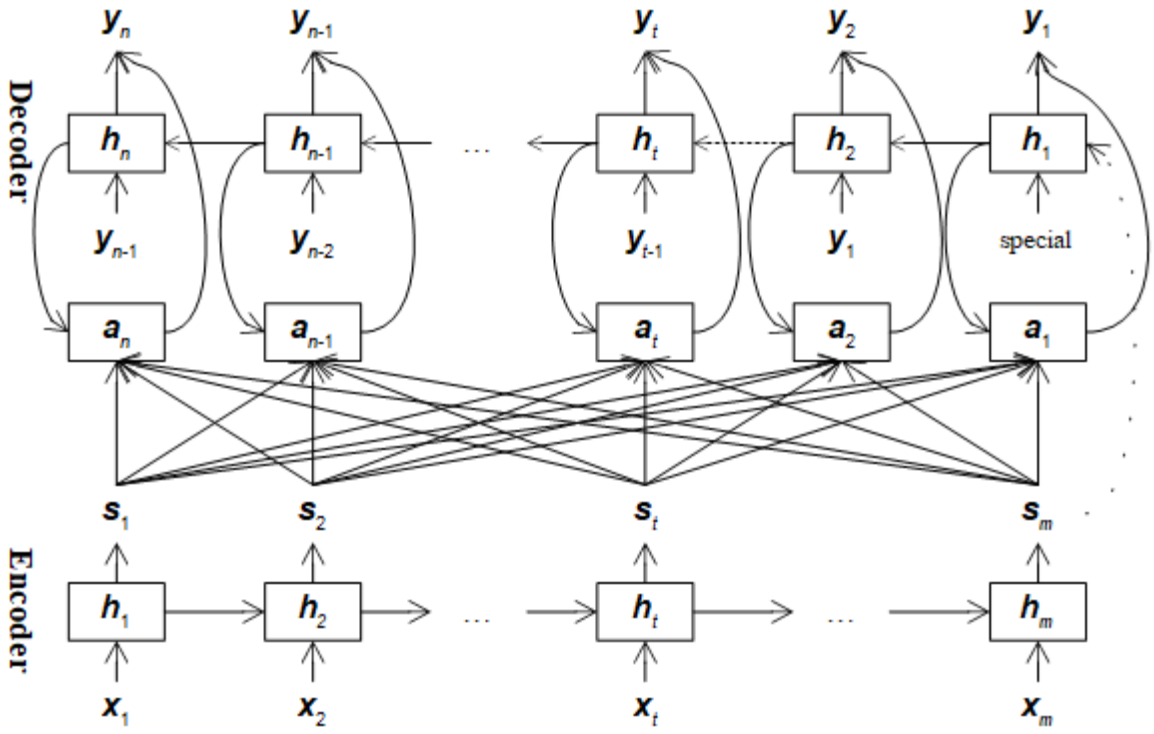

Figure 2.4. Encoder-decoder model with attention

Training encoder-decoder model with support attention is still based on likelihood maximization or linear component aforementioned. Attention mechanism mentioned here does not ever concern internal meaning of every token, which only fosters target sequence to pay attention at source sequence. The attention that concerns internal meanings of tokens is called self-attention which is an advancement of attention. In other words, self-attention fosters source sequence to pay attention to itself. Transformer mentioned in the next section will implement self-attention.

3. Transformer

Transformer, developed by Vaswani et al. (Vaswani, et al., 2017) in the famous paper "Attention Is All You Need", has also attention mechanism and encoder-decoder mechanism like the aforementioned generation model that applies recurrent neural network (RNN) and short-term memory (LSTM) but transformer does not require to process successively tokens of sequence in token-by-token ordering, which improves translation speed. Moreover, another strong point of transformer is that it has *self-attention* which is the special attention that concerns internal meanings of its own tokens. Transformer supports both attention and self-attention, which fosters target sequence to pay attention to both source sequence and target sequence and also fosters source sequence to pay attention to itself. Besides, transformer does not apply RNN / LSTM. Note that word and sentence in natural language processing (NLP) are mentioned as token and sequence, respectively by a convention, so that source sequence $\boldsymbol{X}$ is fed to encoder and target sequence $\boldsymbol{Y}$ is fed to decoder where $\boldsymbol{X}$ and $\boldsymbol{Y}$ are concerned exactly as matrices.

$$\boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m)^T$$

$$\boldsymbol{Y} = (\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_n)^T$$

Each encoder as well as each decoder in transformer are composed of some identical layers. The number of layer which is developed by Vaswani et al. (Vaswani, et al., 2017, p. 3) is 6. Each encoder layer has two sublayers which are *multi-head attention* sublayer and feedforward sublayer

whereas each decoder layer has three sublayers which are masked multi-head attention sublayer, multi-head attention sublayer, and feedforward sublayer. Every sublayer is followed by association of *residual mechanism* and *layer normalization*, denoted as Add & Norm = LayerNorm($\boldsymbol{X}$ + Sublayer($\boldsymbol{X}$)). The residual mechanism means that sublayer Sublayer($\boldsymbol{X}$) is added with its input as the sum $\boldsymbol{X}$ + Sublayer($\boldsymbol{X}$). Note, Sublayer($\boldsymbol{X}$) can be attention sublayer or feedforward sublayer. The layer normalization is to normalize such sum. Following figure summarizes transformer developed by Vaswani et al. (Vaswani, et al., 2017, p. 3).

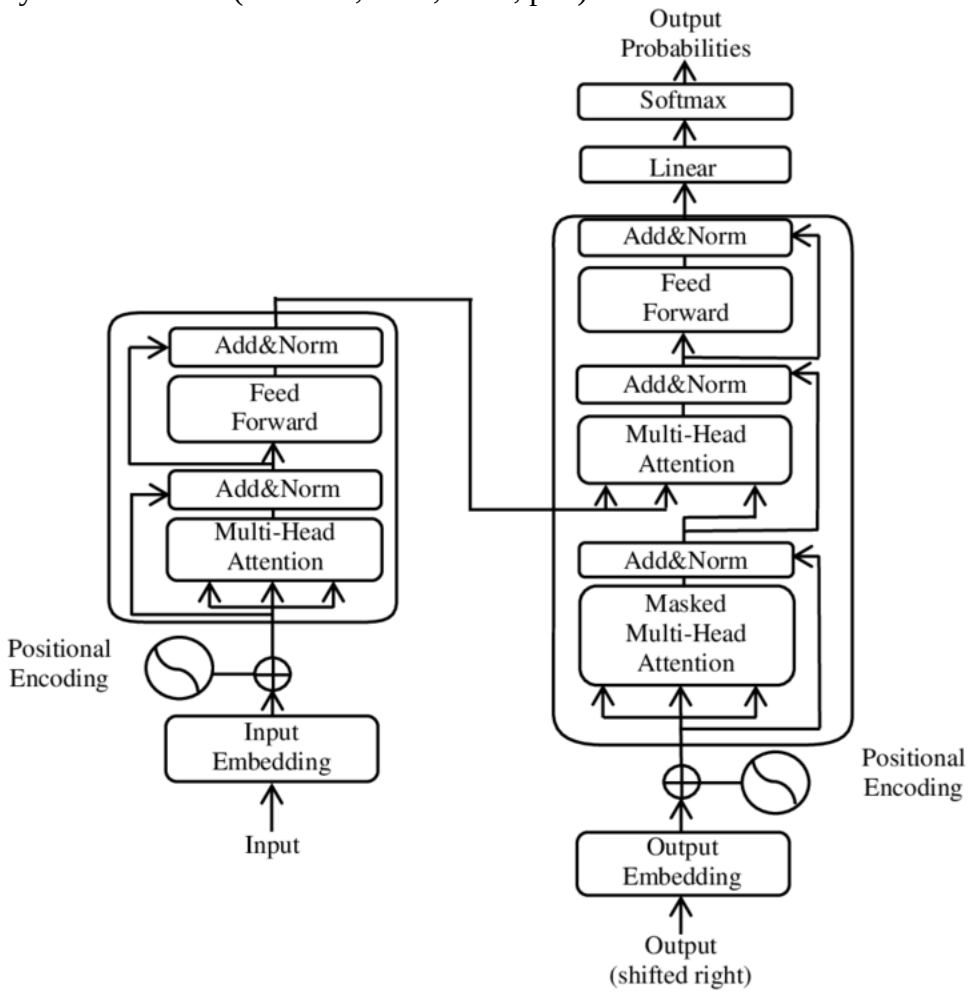

Figure 3.1. Architecture of transformer

Feedforward sublayer also called feedforward network (FNN) aims to fine-tune attention by increasing degree of complication.

Encoder and its attention are described firstly when multi-head attention is derived from basic concept of attention. Attention (self-attention) proposed by Vaswani et al. (Vaswani, et al., 2017) is based on three important matrices such as query matrix Q, key matrix K, and value matrix V. The number of rows of these matrices is m which is the number of tokens in sequence matrix $\boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m)^T$ but the number of columns of query matrix Q and key matrix K is d_k whereas the number of columns of value matrix V is d_v. The number m of token is set according to concrete applications, which is often the number of words of the longest sentence. In literature (Vaswani, et al., 2017), d_k and d_v are called key dimension and value dimension, respectively. Dimensions of matrices Q, K, and V are m x d_k, m x d_k, and m x d_v, respectively (Vaswani, et al., 2017), (Wikipedia, Transformer (deep learning architecture), 2019).

$$Q = \begin{pmatrix} \boldsymbol{q}_1 \\ \boldsymbol{q}_2 \\ \vdots \\ \boldsymbol{q}_m \end{pmatrix} = \begin{pmatrix} q_{11} & q_{12} & \cdots & q_{1d_k} \\ q_{21} & q_{22} & \cdots & q_{2d_k} \\ \vdots & \vdots & \ddots & \vdots \\ q_{m1} & q_{m2} & \cdots & q_{md_k} \end{pmatrix}, K = \begin{pmatrix} \boldsymbol{k}_1 \\ \boldsymbol{k}_2 \\ \vdots \\ \boldsymbol{k}_m \end{pmatrix} = \begin{pmatrix} k_{11} & k_{12} & \cdots & k_{1d_k} \\ k_{21} & k_{22} & \cdots & k_{2d_k} \\ \vdots & \vdots & \ddots & \vdots \\ k_{m1} & k_{m2} & \cdots & k_{md_k} \end{pmatrix}$$

$$V = \begin{pmatrix} \boldsymbol{v}_1 \\ \boldsymbol{v}_2 \\ \vdots \\ \boldsymbol{v}_m \end{pmatrix} = \begin{pmatrix} v_{11} & v_{12} & \cdots & v_{1d_v} \\ v_{21} & v_{22} & \cdots & v_{2d_v} \\ \vdots & \vdots & \ddots & \vdots \\ v_{m1} & v_{m2} & \cdots & v_{md_v} \end{pmatrix}$$

Where,

$$\boldsymbol{q}_i = \left(q_{i1}, q_{i2}, \ldots, q_{id_k}\right)^T, \boldsymbol{k}_i = \left(k_{i1}, k_{i2}, \ldots, k_{id_k}\right)^T$$
$$\boldsymbol{v}_i = \left(v_{i1}, v_{i2}, \ldots, v_{id_v}\right)^T$$

Suppose every token vector $\boldsymbol{x}_i$ in sequence matrix $\boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m)^T$ has d_m elements such that d_m is called model dimension which is often 512 in NLP.

$$\boldsymbol{x}_i = \left(x_{i1}, x_{i2}, \ldots, x_{id_m}\right)^T$$

Query matrix Q, key matrix K, and value matrix V are determined by products of sequence matrix $\boldsymbol{X}$ and query weight matrix W^Q, key weight matrix W^K, value weight matrix W^V.

$$Q = \boldsymbol{X}W^Q$$
$$K = \boldsymbol{X}W^K$$
$$V = \boldsymbol{X}W^V$$

Of course, dimensions of weight matrices W^Q, W^K, and W^V are d_m x d_k, d_m x d_k, and d_m x d_v, respectively. All of them have d_m rows. Matrices W^Q and W^K have d_k columns whereas matrix W^V have d_v columns.

$$W^Q = \begin{pmatrix} w_{11}^Q & w_{11}^Q & \cdots & w_{1d_k}^Q \\ w_{21}^Q & w_{22}^Q & \cdots & w_{2d_k}^Q \\ \vdots & \vdots & \ddots & \vdots \\ w_{d_m1}^Q & w_{d_m2}^Q & \cdots & w_{d_md_k}^Q \end{pmatrix}, W^K = \begin{pmatrix} w_{11}^K & w_{11}^K & \cdots & w_{1d_k}^K \\ w_{21}^K & w_{22}^K & \cdots & w_{2d_k}^K \\ \vdots & \vdots & \ddots & \vdots \\ w_{d_m1}^K & w_{d_m2}^K & \cdots & w_{d_md_k}^K \end{pmatrix}$$

$$W^V = \begin{pmatrix} w_{11}^V & w_{11}^V & \cdots & w_{1d_v}^V \\ w_{21}^V & w_{22}^V & \cdots & w_{2d_k}^V \\ \vdots & \vdots & \ddots & \vdots \\ w_{d_m1}^V & w_{d_m2}^V & \cdots & w_{d_md_k}^V \end{pmatrix}$$

Attention is calculated based on scaled product of query matrix Q, key matrix K, and value matrix V in order to make effects on value matrix V specifying real sequence by probabilities and moreover, these probabilities are calculated by matching query matrix Q specifying query sequence and key matrix K specifying key sequence, which is similar to searching mechanism. These probabilities are also based on soft-max function, which implies weights too. Moreover, attention focuses on all tokens of sequence, which improves meaningful context of sentence in NLP. Given matrices Q, K, and V, attention of Q, K, and V is specified as follows:

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

Note, the subscript "T" denotes vector/matrix transposition operator. It is easy to recognize this attention is self-attention of only one sequence $\boldsymbol{X}$ via Q, K, and V which are essentially calculated from $\boldsymbol{X}$ and weight matrices W^Q, W^K, and W^V. Note, self-attention concerns internal meanings of

its own tokens. Transformer here fosters source sequence to pay attention to itself. The reason of dividing product QK^T by the scaling factor $\sqrt{d_k}$ is to improve convergence speed in training transformer. Before explaining how to calculate weight / probability matrix $\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)$, it is necessary to skim the product QK^T of query matrix Q and key matrix K which aims to match query sequence and key sequence.

$$QK^T = \begin{pmatrix} \frac{\boldsymbol{q}_1\boldsymbol{k}_1^T}{\sqrt{d_k}} & \frac{\boldsymbol{q}_1\boldsymbol{k}_2^T}{\sqrt{d_k}} & \cdots & \frac{\boldsymbol{q}_1\boldsymbol{k}_m^T}{\sqrt{d_k}} \\ \frac{\boldsymbol{q}_2\boldsymbol{k}_1^T}{\sqrt{d_k}} & \frac{\boldsymbol{q}_2\boldsymbol{k}_2^T}{\sqrt{d_k}} & \cdots & \frac{\boldsymbol{q}_2\boldsymbol{k}_m^T}{\sqrt{d_k}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\boldsymbol{q}_m\boldsymbol{k}_1^T}{\sqrt{d_k}} & \frac{\boldsymbol{q}_m\boldsymbol{k}_2^T}{\sqrt{d_k}} & \cdots & \frac{\boldsymbol{q}_m\boldsymbol{k}_m^T}{\sqrt{d_k}} \end{pmatrix}$$

The dot product $\boldsymbol{q}_i\boldsymbol{k}_j^T$ which indicates how much the query vector $\boldsymbol{q}_i$ matches or attends mutually the key vector $\boldsymbol{k}_j$ is specified as follows:

$$\boldsymbol{q}_i\boldsymbol{k}_j^T = \sum_{l=1}^{d_k} q_{il}k_{jl}$$

Probability matrix $\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)$ is specified as follows:

$$\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) = \begin{pmatrix} \boldsymbol{p}_1 \\ \boldsymbol{p}_2 \\ \vdots \\ \boldsymbol{p}_m \end{pmatrix} = \begin{pmatrix} \text{softmax}\left(\frac{\boldsymbol{q}_1\boldsymbol{k}_1^T}{\sqrt{d_k}}, \frac{\boldsymbol{q}_1\boldsymbol{k}_2^T}{\sqrt{d_k}}, \ldots, \frac{\boldsymbol{q}_1\boldsymbol{k}_m^T}{\sqrt{d_k}}\right) \\ \text{softmax}\left(\frac{\boldsymbol{q}_2\boldsymbol{k}_1^T}{\sqrt{d_k}}, \frac{\boldsymbol{q}_2\boldsymbol{k}_2^T}{\sqrt{d_k}}, \ldots, \frac{\boldsymbol{q}_2\boldsymbol{k}_m^T}{\sqrt{d_k}}\right) \\ \vdots \\ \text{softmax}\left(\frac{\boldsymbol{q}_m\boldsymbol{k}_1^T}{\sqrt{d_k}}, \frac{\boldsymbol{q}_m\boldsymbol{k}_2^T}{\sqrt{d_k}}, \ldots, \frac{\boldsymbol{q}_m\boldsymbol{k}_m^T}{\sqrt{d_k}}\right) \end{pmatrix}$$

The i^{th} row of probability matrix $\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)$ includes weights / probabilities that the i^{th} token is associated with all tokens including itself with note that $\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)$ is m x m matrix, specified by weight/probability vector $\boldsymbol{p}_i$. It is necessary to explain the i^{th} row of probability matrix $\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)$ which is the following row vector:

$$\text{softmax}\left(\frac{\boldsymbol{q}_i\boldsymbol{k}_1^T}{\sqrt{d_k}}, \frac{\boldsymbol{q}_i\boldsymbol{k}_2^T}{\sqrt{d_k}}, \ldots, \frac{\boldsymbol{q}_i\boldsymbol{k}_m^T}{\sqrt{d_k}}\right) = \boldsymbol{p}_i = (p_{i1}, p_{i2}, \ldots, p_{im})$$

Each probability p_{ij}, which is weight indeed, is calculated by soft-max function as follows:

$$p_{ij} = \frac{\exp\left(\frac{\boldsymbol{q}_i\boldsymbol{k}_j^T}{\sqrt{d_k}}\right)}{\sum_{l=1}^{m}\exp\left(\frac{\boldsymbol{q}_i\boldsymbol{k}_l^T}{\sqrt{d_k}}\right)}, \forall j = \overline{1,m}$$

Where exp(.) is natural exponential function. Therefore, probability matrix $\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)$ is totally determined:

$$\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) = \begin{pmatrix} \boldsymbol{p}_1 \\ \boldsymbol{p}_2 \\ \vdots \\ \boldsymbol{p}_m \end{pmatrix} = \begin{pmatrix} p_{11} & p_{12} & \cdots & p_{1m} \\ p_{21} & p_{22} & \cdots & p_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ p_{m1} & p_{m2} & \cdots & p_{mm} \end{pmatrix}$$

Where,

$$\boldsymbol{p}_i = (p_{i1}, p_{i2}, \ldots, p_{im})^T, \forall i = \overline{1, m}$$

Self-attention of *Q*, *K*, and *V* is totally determined as follows:

$$\text{Attention}(\boldsymbol{X}) = \text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V = \begin{pmatrix} \boldsymbol{p}_1 \\ \boldsymbol{p}_2 \\ \vdots \\ \boldsymbol{p}_m \end{pmatrix}\left(\overline{\boldsymbol{v}}_1, \overline{\boldsymbol{v}}_2, \ldots, \overline{\boldsymbol{v}}_{d_v}\right)$$

$$= \begin{pmatrix} \boldsymbol{p}_1\overline{\boldsymbol{v}}_1^T & \boldsymbol{p}_1\overline{\boldsymbol{v}}_2^T & \cdots & \boldsymbol{p}_1\overline{\boldsymbol{v}}_{d_v}^T \\ \boldsymbol{p}_2\overline{\boldsymbol{v}}_1^T & \boldsymbol{p}_2\overline{\boldsymbol{v}}_2^T & \cdots & \boldsymbol{p}_2\overline{\boldsymbol{v}}_{d_v}^T \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{p}_m\overline{\boldsymbol{v}}_1^T & \boldsymbol{p}_m\overline{\boldsymbol{v}}_2^T & \cdots & \boldsymbol{p}_m\overline{\boldsymbol{v}}_{d_v}^T \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1d_v} \\ a_{21} & a_{22} & \cdots & a_{2d_v} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{md_v} \end{pmatrix}$$

Where,

$$\overline{\boldsymbol{v}}_j = \begin{pmatrix} v_{1j} \\ v_{2j} \\ \vdots \\ v_{mj} \end{pmatrix}$$

$$a_{ij} = \boldsymbol{p}_i\overline{\boldsymbol{v}}_j^T = \sum_{k=1}^{m} p_{ik}v_{kj}$$

Note, $\overline{\boldsymbol{v}}_j$ is the j^{th} column vector of value matrix *V*. Of course, dimension of self-attention Attention(*Q*, *K*, *V*) is *m* x d_v having *m* rows and d_v columns. Attention Attention(*Q*, *K*, *V*) is also called scaled dot product attention because of dot product $\boldsymbol{q}_i\boldsymbol{k}_j^T$ and scaling factor $\sqrt{d_k}$. Each row $\boldsymbol{a}_i = (a_{i1}, a_{i2}, \ldots, a_{1d_v})^T$ of Attention(*Q*, *K*, *V*), which is a d_v-length vector, is self-attention of the i^{th} token which is contributed by all tokens via scaled dot products QK^T. Therefore, the preeminence of self-attention is that self-attention concerns all tokens in detail instead of concerning only sequence and the self-attention $\boldsymbol{a}_i = (a_{i1}, a_{i2}, \ldots, a_{1d_v})^T$ of the i^{th} token is attended by all tokens. For example, given sentence "Jack is now asleep, because he is tired.", the word "he" is strongly implied to the word "Jack" by self-attention of the word "he" although the word "he" is ambiguous. Following figure (Han, et al., 2021, p. 231) illustrates the self-attention of the word "he" in which each strength of implication of another word (accept itself "he") to the word "he" is indicated by strong degree of connection color.

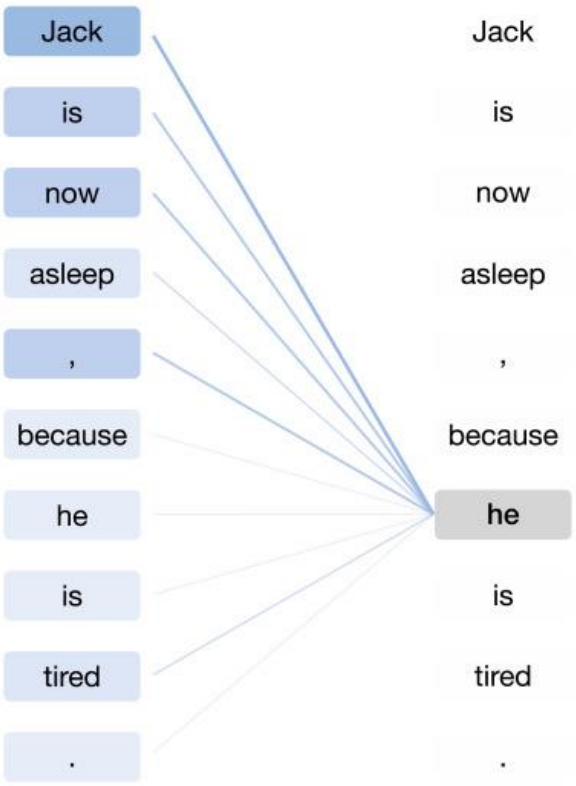

Figure 3.2. Self-attention example

Vaswani et al. (Vaswani, et al., 2017) proposed an improvement of attention called multi-head attention which is concatenation of many attentions. The existence of many attentions aims to discover as much as possible different meanings under attentions and the concatenation mechanism aims to unify different attentions into one self-attention. Following equation specifies multi-head attention with note that the multi-head attention here is self-attention.

$$\text{MultiheadAttention}(\boldsymbol{X}) = \text{concatenate}(\text{head}_1, \text{head}_2, \dots, \text{head}_h)W^O$$

Where,

$$\text{head}_i = \text{Attention}(Q_i, K_i, V_i)$$
$$Q_i = \boldsymbol{X}W_i^Q$$
$$K_i = \boldsymbol{X}W_i^K$$
$$V_i = \boldsymbol{X}W_i^V$$

Of course, W_i^Q, W_i^K, and W_i^V are query weight matrix, key weight matrix, and value weight matrix for the i^{th} head, respectively whereas W^O is the entire weight matrix whose dimension is often set as hd_v x d_m so that multi-head attention MultiheadAttention($\boldsymbol{X}$) is m x d_m matrix which is the same to dimension of input sequence matrix $\boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_m)^T$. Note that the concatenation mechanism follows horizontal direction so that the concatenation concatenate(head_1, head_2,…, head_h) is m x hd_v matrix when each head head_i = Attention(Q_i, K_i, V_i) is m x d_v matrix. There are h heads (attentions) in the equation above. In practice, h is set so that $hd_v = d_m$ which is model dimension. Recall that d_m is often 512 in NLP. For easy illustration, the concatenation of h attentions is represented as m x hd_v as follows:

$$\text{concatenate}(\text{head}_1, \dots, \text{head}_i, \dots, \text{head}_h)$$
$$= \begin{pmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \cdots & a_{1d_v}^{(1)} & \cdots & a_{11}^{(i)} & a_{12}^{(i)} & \cdots & a_{1d_v}^{(i)} & \cdots & a_{11}^{(h)} & a_{12}^{(h)} & \cdots & a_{1d_v}^{(h)} \\ a_{21}^{(1)} & a_{22}^{(1)} & \cdots & a_{2d_v}^{(1)} & \cdots & a_{21}^{(i)} & a_{22}^{(i)} & \cdots & a_{2d_v}^{(i)} & \cdots & a_{2d_v}^{(h)} & a_{21}^{(h)} & \cdots & a_{2d_v}^{(h)} \\ \vdots & \vdots & \ddots & \vdots & \cdots & \vdots & \vdots & \ddots & \vdots & \cdots & \vdots & \vdots & \ddots & \vdots \\ a_{m1}^{(1)} & a_{m2}^{(1)} & \cdots & a_{md_v}^{(1)} & \cdots & a_{m1}^{(i)} & a_{m2}^{(i)} & \cdots & a_{md_v}^{(i)} & \cdots & a_{m1}^{(h)} & a_{m2}^{(h)} & \cdots & a_{md_v}^{(h)} \end{pmatrix}$$

Obviously, weight matrix W^O is hd_v x d_m matrix so that multi-head attention MultiheadAttention($\boldsymbol{X}$) is m x d_m matrix, as follows:

$$W^O = \begin{pmatrix} \omega_{11}^{(1)} & \omega_{12}^{(1)} & \cdots & \omega_{1d_m}^{(1)} \\ \omega_{21}^{(1)} & \omega_{21}^{(1)} & \cdots & \omega_{2d_m}^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ \omega_{d_v1}^{(1)} & \omega_{d_v2}^{(1)} & \cdots & \omega_{d_vd_m}^{(1)} \\ \vdots & \vdots & \vdots & \vdots \\ \omega_{11}^{(i)} & \omega_{12}^{(i)} & \cdots & \omega_{1d_m}^{(i)} \\ \omega_{21}^{(i)} & \omega_{21}^{(i)} & \cdots & \omega_{2d_m}^{(i)} \\ \vdots & \vdots & \ddots & \vdots \\ \omega_{d_v1}^{(i)} & \omega_{d_v2}^{(i)} & \cdots & \omega_{d_vd_m}^{(i)} \\ \vdots & \vdots & \vdots & \vdots \\ \omega_{11}^{(h)} & \omega_{12}^{(h)} & \cdots & \omega_{1d_m}^{(h)} \\ \omega_{21}^{(h)} & \omega_{21}^{(h)} & \cdots & \omega_{2d_m}^{(h)} \\ \vdots & \vdots & \ddots & \vdots \\ \omega_{d_v1}^{(h)} & \omega_{d_v2}^{(h)} & \cdots & \omega_{d_vd_m}^{(h)} \end{pmatrix}$$

After multi-head attention goes through residual mechanism and layer normalization of attention sublayer, it is fed to feedforward sublayer or feedforward network (FFN) to finish the processing of encoder. Let EncoderAttention($\boldsymbol{X}$) be output of encoder which is considered as attention:

$$\text{MultiheadAttention}(\boldsymbol{X}) = \text{LayerNorm}\big(\boldsymbol{X} + \text{MultiheadAttention}(\boldsymbol{X})\big)$$

$$\text{EncoderAttention}(\boldsymbol{X}) = \text{LayerNorm}\Big(\text{MultiheadAttention}(\boldsymbol{X}) + \text{FFN}\big(\text{MultiheadAttention}(\boldsymbol{X})\big)\Big)$$

If there is a stack of N encoders, the process above is repeated N times. In literature (Vaswani, et al., 2017), N is set to be 6. Without loss of generality, we can consider $N = 1$ as simplest case for easy explanations.

Now it is essential to survey how decoder applies encoder attention EncoderAttention($\boldsymbol{X}$) into its encoding task. Essentially, decoder has two multi-head attentions such as masked multi-head attention and multi-head attention whereas encoder has only one multi-head attention. Their attentions are similar to encoder's attention but there is a slight difference. Firstly, decoder input sequence $\boldsymbol{Y} = (\boldsymbol{y}_1, \boldsymbol{y}_2, \dots, \boldsymbol{y}_n)^T$ is fed to masked multi-head attention sublayer with note that $\boldsymbol{Y}$ is n x d_m matrix with support that model dimension d_m, which is often set to be 512 in natural language processing (NLP), may not be changed with regard to decoder. Because masked multi-head attention is composed by concatenation of masked head attentions by the same way of encoder, we should concern masked head attention. Sequence $\boldsymbol{Y}$ should have $n = m$ tokens like sequence $\boldsymbol{X}$ in practice. This is necessary because the length $m = n$ is the largest number of possible tokens in

any sequence. For shorter sentences in NLP, redundant tokens are represented by zeros. Moreover, most of parameters (weight matrices) of encoder and decoder are independent from m and n, especially in the case $m = n$.

There is a principle that a token $\boldsymbol{y}_i$ in sequence $\boldsymbol{Y}$ does not know its successive tokens $\boldsymbol{y}_{i+1}$, $\boldsymbol{y}_{i+2}, \ldots, \boldsymbol{y}_n$ with note that these tokens are called unknown tokens for token $\boldsymbol{y}_i$, which causes that soft-max function needs to be added a mask matrix M whose unknown positions are removed by setting them to be negative infinites because evaluation of negative infinite by exponential function is zero. *Masked attention* is self-attention too.

$$\text{MaskedAttention}(\boldsymbol{Y}) = \text{MaskedAttention}(Q, K, V) = \text{softmax}\left(M + \frac{QK^T}{\sqrt{d_k}}\right)V$$

Where masked matrix M is triangle matrix with negative infinites on upper part and zeros on lower part as follows:

$$M = \begin{pmatrix} 0 & -\infty & -\infty & \ldots & -\infty \\ 0 & 0 & -\infty & \cdots & -\infty \\ 0 & 0 & 0 & \cdots & -\infty \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 0 \end{pmatrix}$$

Note,

$$Q = \boldsymbol{Y}W^Q$$
$$K = \boldsymbol{Y}W^K$$
$$V = \boldsymbol{Y}W^V$$

Where W^Q, W^K, and W^V are weight matrices with note that they are different from the ones of encoder. Dimensions of weight matrices W^Q, W^K, and W^V are d_m x d_k, d_m x d_k, and d_m x d_v, respectively. Dimensions of matrices Q, K, and V are n x d_k, n x d_k, and n x d_v, respectively whereas dimension of masked matrix M is n x d_m. We have QK^T is n x n matrix:

$$QK^T = \begin{pmatrix} \frac{\boldsymbol{q}_1\boldsymbol{k}_1^T}{\sqrt{d_k}} & -\infty & -\infty & \ldots & -\infty \\ \frac{\boldsymbol{q}_2\boldsymbol{k}_1^T}{\sqrt{d_k}} & \frac{\boldsymbol{q}_2\boldsymbol{k}_2^T}{\sqrt{d_k}} & -\infty & \cdots & -\infty \\ \frac{\boldsymbol{q}_3\boldsymbol{k}_1^T}{\sqrt{d_k}} & \frac{\boldsymbol{q}_3\boldsymbol{k}_2^T}{\sqrt{d_k}} & \frac{\boldsymbol{q}_3\boldsymbol{k}_3^T}{\sqrt{d_k}} & \cdots & -\infty \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\boldsymbol{q}_n\boldsymbol{k}_1^T}{\sqrt{d_k}} & \frac{\boldsymbol{q}_n\boldsymbol{k}_2^T}{\sqrt{d_k}} & \frac{\boldsymbol{q}_n\boldsymbol{k}_3^T}{\sqrt{d_k}} & \cdots & \frac{\boldsymbol{q}_n\boldsymbol{k}_n^T}{\sqrt{d_k}} \end{pmatrix}$$

Recall that the purpose of masked matrix M is to remove the affections of current token from its after tokens such that:

$$\text{softmax}\left(M + \frac{QK^T}{\sqrt{d_k}}\right) = \begin{pmatrix} \boldsymbol{p}_1 \\ \boldsymbol{p}_2 \\ \boldsymbol{p}_3 \\ \vdots \\ \boldsymbol{p}_n \end{pmatrix} = \begin{pmatrix} p_{11} & 0 & 0 & \ldots & 0 \\ p_{21} & p_{22} & 0 & \cdots & 0 \\ p_{31} & p_{32} & p_{33} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ p_{n1} & p_{n2} & p_{n3} & \cdots & p_{nn} \end{pmatrix}$$

Where,

$$p_{ij} = \frac{\exp\left(\frac{\boldsymbol{q}_i \boldsymbol{k}_j^T}{\sqrt{d_k}}\right)}{\sum_{l=1}^{n} \exp\left(\frac{\boldsymbol{q}_i \boldsymbol{k}_l^T}{\sqrt{d_k}}\right)}, \forall i \geq j$$

$$p_{ij} = \frac{\exp(-\infty)}{\sum_{l=1}^{n}(\ldots)} = 0, \forall i < j$$

Therefore, masked attention is determined as follows:

$$\text{MaskedAttention}(\boldsymbol{Y}) = \text{MaskedAttention}(Q, K, V) = \text{softmax}\left(M + \frac{QK^T}{\sqrt{d_k}}\right)V$$

$$= \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1d_v} \\ a_{21} & a_{22} & \cdots & a_{2d_v} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nd_v} \end{pmatrix}$$

Where attention element a_{ij} is calculated by the aforementioned way:

$$a_{ij} = \boldsymbol{p}_i \overline{\boldsymbol{v}}_j^T = \sum_{k=1}^{n} p_{ik} v_{kj}$$

Dimension of masked attention MaskedAttention($\boldsymbol{Y}$) is n x d_v having n rows and d_v columns. Following equation specifies masked multi-head attention which is concatenation of some masked attentions.

$$\text{MaskedMultiheadAttention}(\boldsymbol{Y}) = \text{concatenate}(\text{head}_1, \text{head}_2, \ldots, \text{head}_h)W^O$$

Where,

$$\text{head}_i = \text{MaskedAttention}(Q_i, K_i, V_i)$$
$$Q_i = \boldsymbol{Y}W_i^Q$$
$$K_i = \boldsymbol{Y}W_i^K$$
$$V_i = \boldsymbol{Y}W_i^V$$

Please pay attention that weights matrices W_i^Q, W_i^K, W_i^V, and W^O are different from the ones of encoder. Dimensions of W_i^Q, W_i^K, W_i^V, and W^O are d_m x d_k, d_m x d_k, d_m x d_v, and hd_v x d_m so that dimension of masked multi-head attention MaskedMultiheadAttention($\boldsymbol{Y}$) is n x d_m. Residual mechanism and layer normalization are applied into masked multi-head attention too:

$$\text{MaskedMultiheadAttention}(\boldsymbol{Y}) = \text{LayerNorm}\big(\boldsymbol{Y} + \text{MaskedMultiheadAttention}(\boldsymbol{Y})\big)$$

Because mechanism of multi-head attention of decoder is relatively special, it is called complex multi-head attention for convention. Because complex multi-head attention is composed by concatenation of some complex attentions by the same way of encoder, we should concern complex attention.

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

Query matrix Q and key matrix K of complex attention are products of encoder attention EncoderAttention($\boldsymbol{X}$) and query weight matrix U^Q and key weight matrix U^K, respectively.

$$Q = T\text{EncoderAttention}(\boldsymbol{X})U^Q$$
$$K = T\text{EncoderAttention}(\boldsymbol{X})U^K$$

Where T is transformation matrix whose dimension is n x m. If $n = m$, matrix T will be removed. Value matrix V of complex attention is product of masked multi-head attention and value weight matrix U^V.

$$V = \text{MaskedMultiheadAttention}(\boldsymbol{Y})U^V$$

Dimensions of weight matrices U^Q, U^K, and U^V are d_m x d_k, d_m x d_k, and d_m x d_v, respectively. Following figure depicts Attention($\boldsymbol{X}$, $\boldsymbol{Y}$) in general view.

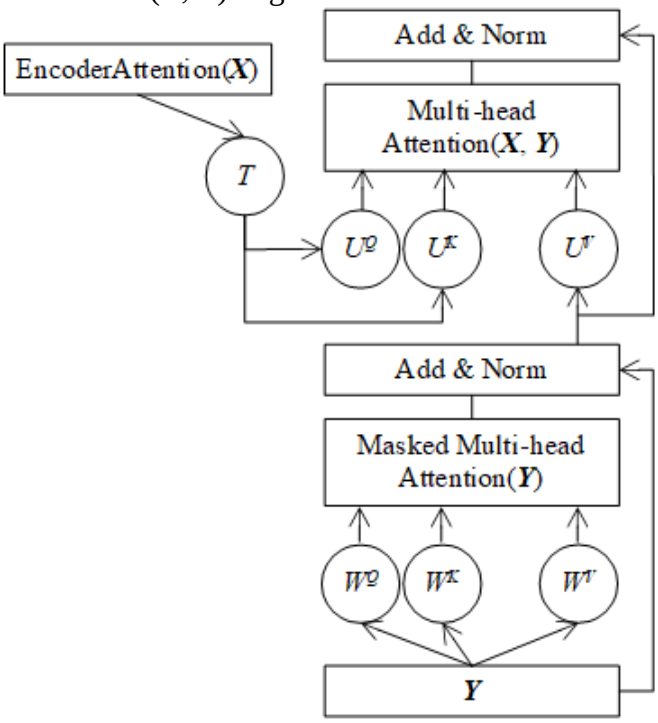

Figure 3.3. Decoder attention Attention($\boldsymbol{X}$, $\boldsymbol{Y}$) in general view

Transformer here fosters target sequence to pay attention to itself and source sequence by masked self-attention and encoder attention. Of course, after complex attention is calculated, multi-head attention of decoder (complex multi-head attention) is totally determined.

$$\text{MultiheadAttention}(\boldsymbol{X}, \boldsymbol{Y}) = \text{concatenate}(\text{head}_1, \text{head}_2, \dots, \text{head}_h)U^O$$

Where,

$$\text{head}_i = \text{ComplexAttention}(Q_i, K_i, V_i)$$
$$Q_i = T\text{EncoderAttention}(\boldsymbol{X})U_i^Q$$
$$K_i = T\text{EncoderAttention}(\boldsymbol{X})U_i^K$$
$$V_i = \text{MaskedMultiheadAttention}(\boldsymbol{Y})U^V$$

Of course, U_i^Q, U_i^K, and U_i^V are query weight matrix, key weight matrix, and value weight matrix of the i^{th} head, respectively whereas U^O is entire weight matrix and T is transformation matrix. Because encoder attention EncoderAttention($\boldsymbol{X}$) is m x d_m matrix, dimension of transformation matrix T is n x m. If $n = m$, matrix T will be removed. In practice, it is necessary to set $n = m$. Dimensions of U_i^Q, U_i^K, U_i^V, and U^O are d_m x d_k, d_m x d_k, d_m x d_v, and hd_v x d_m so that dimension of multi-head attention MultiheadAttention($\boldsymbol{X}$, $\boldsymbol{Y}$) is n x d_m. Residual mechanism and layer normalization are applied into decoder multi-head attention too:

$$\text{MultiheadAttention}(\boldsymbol{X}, \boldsymbol{Y})$$
$$= \text{LayerNorm}\big(\text{MaskedMultiheadAttention}(\boldsymbol{Y}) + \text{MultiheadAttention}(\boldsymbol{X}, \boldsymbol{Y})\big)$$

Let $\boldsymbol{Z}$ be output of decoder which is decoder attention too, we obtain:

$$\boldsymbol{Z} = \text{DecoderAttention}(\boldsymbol{X}, \boldsymbol{Y})$$
$$= \text{LayerNorm}\Big(\text{MultiheadAttention}(\boldsymbol{X}, \boldsymbol{Y}) + \text{FFN}\big(\text{MultiheadAttention}(\boldsymbol{X}, \boldsymbol{Y})\big)\Big)$$

Where FFN denotes feedforward network or feedforward sublayer. If there is a stack of N decoders, the process above is repeated N times. In literature (Vaswani, et al., 2017), N is set to be 6. Without loss of generality, we can consider $N = 1$ as simplest case for easy explanations. Note, dimension of $\boldsymbol{Z}$ is n x d_m. Model dimension d_m is often set to be 512 in NLP.

In context of statistical translation machine (STM), it is necessary to calculate probabilities of words (tokens) in vocabulary Ω. Because these probabilities are calculated based on soft-max function, it is first to map decoder output matrix $\boldsymbol{Z}$ into weight vector $\boldsymbol{w} = (w_1, w_2, \ldots, w_{|\Omega|})^T$ where every element w_i of vector $\boldsymbol{w}$ is weight of the i^{th} word in vocabulary Ω. The mapping is implemented by a feedforward network (FNN) called linear component in literature (Vaswani, et al., 2017, p. 3). In other words, input of linear component is sequence matrix $\boldsymbol{Z}$ whereas its output is weight vector $\boldsymbol{w}$ (Alammar, 2018). Please pay attention that the length of $\boldsymbol{w}$ is the number of words (tokens) in vocabulary Ω and so, $\boldsymbol{w}$ is also called token/word weight vector.

$$\boldsymbol{w} = \big(w_1, w_2, \ldots, w_{|\Omega|}\big)^T = \text{linear}(\boldsymbol{Z}) = \text{FFN}(\boldsymbol{Z})$$

In practice, $\boldsymbol{Z}$ is flattened into long vector because $\boldsymbol{w}$ is vector too so that FNN can be implemented. After token weight vector $\boldsymbol{w}$ is determined, it is easily converted into output probability vector $\boldsymbol{p} = (p_1, p_2, \ldots, p_{|\Omega|})^T$ where each element p_i is probability of the i^{th} word/token in vocabulary when sentence/sequence $\boldsymbol{Z}$ is raised (Alammar, 2018). If the t^{th} word is issued, its probability p_t is 1 and other probabilities are 0.

$$\boldsymbol{p} = \text{softmax}(\boldsymbol{w}) = \big(p_1, p_2, \ldots, p_{|\Omega|}\big)^T$$
$$p_i = \frac{\exp(w_i)}{\sum_{l=1}^{|\Omega|} \exp(w_l)}$$

Consequently, the next token which is predicted in STM for example is the one whose probability is highest, which means that the largest element in $\boldsymbol{p}$ need to be found for STM translation after linear component $\boldsymbol{w}$ and output probability $\boldsymbol{p}$ are evaluated given $\boldsymbol{Z}$ which in turn determined based on source sequence $\boldsymbol{X}$ and target sequence $\boldsymbol{Y}$ via mechanism encoder/decoder and attention.

It is not difficult to learn linear component FFN($\boldsymbol{Z}$) by backpropagation algorithm associated stochastic gradient descent (SGD) method. Concretely, the following cross-entropy $L(\boldsymbol{p} \mid \Theta)$ is minimized so as to train FFN($\boldsymbol{Z}$).

$$L(\boldsymbol{p}|\Theta) = -\sum_{i=1}^{|\Omega|} q_i \log(p_i)$$

Where Θ is parameter of FFN($\boldsymbol{Z}$) and the vector $\boldsymbol{q} = (q_1, q_2, \ldots, q_{|\Omega|})^T$ is binary vector from sample whose each element q_i has binary values {0, 1} indicating whether the i^{th} token/word exists. For example, give sequence/sentence ("I", "am", "a", "student")^T, if there is only one token/word "I" in sample sentence, the binary vector will be $\boldsymbol{q} = (1, 0, 0, 0)^T$. If three words "I", "am", and "student" are mutually existent, the binary vector will be $\boldsymbol{q} = (1, 1, 0, 1)^T$. When SGD is applied into minimizing the cross-entropy, partial gradient of $L(\boldsymbol{p} \mid \Theta)$ with regard to w_j is:

$$\frac{\partial L(\boldsymbol{p}|\Theta)}{\partial w_j} = -\sum_{i=1}^{|\Omega|} q_i\big(\delta_{ij} - p_j\big)$$

Where,

$$\delta_{ij} = \begin{cases} 1 \text{ if } i = j \\ 0 \text{ if } i \neq j \end{cases}$$

Proof,

$$\frac{\partial L(\boldsymbol{p}|\Theta)}{\partial w_j} = -\sum_{i=1}^{|\Omega|} q_i \frac{\partial \log(p_i)}{\partial w_j} = -\sum_{i=1}^{|\Omega|} q_i \frac{d\log(p_i)}{dp_i}\frac{\partial p_i}{\partial w_j} = -\sum_{i=1}^{|\Omega|} q_i \frac{1}{p_i}\frac{\partial p_i}{\partial w_j}$$

Due to:

$$\frac{\partial p_i}{\partial w_j} = \frac{\partial}{\partial w_j}\left(\frac{\exp(w_i)}{\sum_{l=1}^{|\Omega|}\exp(w_l)}\right) = p_i(\delta_{ij} - p_j)$$

We obtain:

$$\frac{\partial L(\boldsymbol{p}|\Theta)}{\partial w_j} = -\sum_{i=1}^{|\Omega|} q_i(\delta_{ij} - p_j) \blacksquare$$

So that gradient of $L(\boldsymbol{p} \mid \Theta)$ with regard to $\boldsymbol{w}$ is:

$$\frac{dL(\boldsymbol{p}|\Theta)}{d\boldsymbol{w}} = \left(\frac{\partial L(\boldsymbol{p}|\Theta)}{\partial w_1}, \frac{\partial L(\boldsymbol{p}|\Theta)}{\partial w_2}, \ldots, \frac{\partial L(\boldsymbol{p}|\Theta)}{\partial w_{|\Omega|}}\right)^T$$

Therefore, parameter Θ is updated according to SGD associated with backpropagation algorithm:

$$\Theta = \Theta - \gamma \frac{dL(\boldsymbol{p}|\Theta)}{d\boldsymbol{w}}$$

Where γ ($0 < \gamma \leq 1$) is learning rate.

For STM example, given French source sentence "Je suis étudiant" (Alammar, 2018) is translated into English target sentence "I am a student" (Alammar, 2018) by transformer which is trained with corpus before (transformer was determined), which goes through following rounds:

Round 1:

- French source sentence "Je suis étudiant" coded by sentence/sequence matrix $\boldsymbol{X} = (\boldsymbol{x}_1 = c(\text{"<bos>"}), \boldsymbol{x}_2 = c(\text{"je"}), \boldsymbol{x}_3 = c(\text{"suis"}), \boldsymbol{x}_4 = c(\text{"étudiant"}), \boldsymbol{x}_5 = c(\text{"<eos>"}))^T$ where $c(.)$ is embedding numeric vector of given word with note that words "<bos>" and "<eos>" are special predefined words indicating the beginning of sentence and the end of sentence, respectively. As a convention, $c(.)$ is called word/token vector whose dimension can be d_m=512. If predefined sentence length is longer, redundant word vectors are set to be zeros, for example, let $\boldsymbol{x}_6 = \boldsymbol{0}, \boldsymbol{x}_7 = \boldsymbol{0}, \ldots, \boldsymbol{x}_{100} = \boldsymbol{0}$ given the maximum number words in sentence is 100. These zero vectors do not affect decoder evaluation and training parameters.
- Source sequence $\boldsymbol{X}$ is fed to encoder so as to produce encoder attention EncoderAttention($\boldsymbol{X}$).

Round 2:

- English target sentence is coded by sequence/matrix $\boldsymbol{Y} = (\boldsymbol{y}_1 = c(\text{"<bos>"}))^T$. If predefined sentence length is longer, redundant word vectors are set to be zeros.
- Target sequence $\boldsymbol{Y} = (\boldsymbol{y}_1 = c(\text{"<bos>"}))^T$ and encoder attention EncoderAttention($\boldsymbol{X}$) are fed to decoder so as to produce decode output $\boldsymbol{Z}$.
- Output $\boldsymbol{Z}$ goes through linear component $\boldsymbol{w}$ = linear($\boldsymbol{Z}$) and soft-max function component $\boldsymbol{p}$ = softmax($\boldsymbol{w}$) so as to find out the maximum probability p_i so that the i^{th} associated word in vocabulary is "i". As a result, the embedding numeric vector of the word "i" is added to target sequence so that we obtain $\boldsymbol{Y} = (\boldsymbol{y}_1 = c(\text{"<bos>"}), \boldsymbol{y}_2 = c(\text{"i"}))^T$.

Round 3:

- Both target sequence $\boldsymbol{Y}$ = ($\boldsymbol{y}_1$ = c("<bos>"), $\boldsymbol{y}_2$ = c("i"))^T and encoder attention EncoderAttention($\boldsymbol{X}$) are fed to decoder so as to produce decode output $\boldsymbol{Z}$.
- Output $\boldsymbol{Z}$ goes through linear component $\boldsymbol{w}$ = linear($\boldsymbol{Z}$) and soft-max function component $\boldsymbol{p}$ = softmax($\boldsymbol{w}$) so as to find out the maximum probability p_i so that the i^{th} associated word in vocabulary is "am". As a result, the embedding numeric vector of the word "am" is added to target sequence so that we obtain $\boldsymbol{Y}$ = ($\boldsymbol{y}_1$ = c("<bos>"), $\boldsymbol{y}_2$ = c("i") , $\boldsymbol{y}_3$ = c("am"))^T.

Similarly, rounds 4, 5, and 6 are processed by the same way so as to obtain final target sequence $\boldsymbol{Y}$ = ($\boldsymbol{y}_1$ = c("<bos>"), $\boldsymbol{y}_2$ = c("i"), $\boldsymbol{y}_3$ = c("am") , $\boldsymbol{y}_4$ = c("a"), $\boldsymbol{y}_5$ = c("student") , $\boldsymbol{y}_6$ = c("<eos>"))^T which is the English sentence "I am a student" translated from the French sentence "Je suis étudiant". Note, the translation process is stopped when the end-of-sentence word "<eos>" is met.

Main ideas of transformer were described but there are two improvements such as positional encoding and normalization. Firstly, *positional encoding* is that sequences $\boldsymbol{X}$ and $\boldsymbol{Y}$ were added by their corresponding position vectors:

$$\boldsymbol{X} = \boldsymbol{X} + \left(\text{pos}(\boldsymbol{x}_1), \text{pos}(\boldsymbol{x}_2), \dots, \text{pos}(\boldsymbol{x}_m)\right)^T$$

$$\boldsymbol{Y} = \boldsymbol{Y} + \left(\text{pos}(\boldsymbol{y}_1), \text{pos}(\boldsymbol{y}_2), \dots, \text{pos}(\boldsymbol{y}_n)\right)^T$$

Without loss of generality, let POS($\boldsymbol{X}$) = (pos($\boldsymbol{x}_1$), pos($\boldsymbol{x}_2$),…, pos($\boldsymbol{x}_m$))^T be position vector whose each element is position pos($\boldsymbol{x}_i$) of token $\boldsymbol{x}_i$. It is necessary to survey pos($\boldsymbol{x}_i$).

$$\text{pos}(\boldsymbol{x}_i) = \text{pos}\left(\left(x_{i1}, x_{i2}, \dots, x_{ij}, \dots, x_{id_m}\right)^T\right)$$

$$= \left(\text{pos}(x_{i1}), \text{pos}(x_{i2}), \dots, \text{pos}\left(x_{ij}\right), \dots, \text{pos}\left(x_{id_m}\right)\right)^T$$

This implies how to calculate position vector POS($\boldsymbol{X}$) is how to calculate position pos(x_{ij}) where i is position of the i^{th} token and j is position of the j^{th} numeric value of such token vector. We have:

$$\text{pos}\left(x_{ij}\right) = \text{pos}(i, j)$$

Suppose two successive numeric values such as j^{th} numeric value and $(j+1)^{\text{th}}$ numeric value such that $j = 2k$ and $j+1 = 2k+1$, we need to calculate two kinds of positions as follows:

$$\text{pos}(i, 2k)$$

$$\text{pos}(i, 2k+1)$$

Fortunately, these positions are easily calculated by sine function and cosine function as follows (Vaswani, et al., 2017, p. 6):

$$\text{pos}(i, 2k) = \sin\left(\frac{i}{10000^{2k/d_m}}\right)$$

$$\text{pos}(i, 2k+1) = \cos\left(\frac{i}{10000^{2k/d_m}}\right)$$

Recall that d_m is model dimension which is the length of token vector $\boldsymbol{x}_i$. It is often set to be 512 in NLP. As a result, we have:

$$\boldsymbol{X} = \boldsymbol{X} + \text{POS}(\boldsymbol{X})$$

$$\text{POS}(\boldsymbol{X}) = \left(\text{pos}(\boldsymbol{x}_1), \text{pos}(\boldsymbol{x}_2), \dots, \text{pos}(\boldsymbol{x}_m)\right)^T$$

$$\text{pos}(\boldsymbol{x}_i) = \left(\text{pos}(i, 1), \text{pos}(i, 2), \dots, \text{pos}(i, d_m)\right)^T$$

$$\text{pos}(i, j) = \begin{cases} \sin\left(\frac{i}{10000^{2k/d_m}}\right) \text{ if } j = 2k \\ \cos\left(\frac{i}{10000^{2k/d_m}}\right) \text{ if } j = 2k+1 \end{cases}$$

Please pay attention that target sequence $\boldsymbol{Y}$ is added by position vector POS($\boldsymbol{Y}$) by the same way too. There may be a question that why sequences $\boldsymbol{X}$ and $\boldsymbol{Y}$ are added by their position vectors before

they are fed into encoder/decoder when tokens in a sequence have their own orders because a sequence is an ordered list of tokens indeed. The answer depends on computational effectiveness as well as flexibility. For example, when sequences are added by their position vectors, transformer can be trained by incomplete French source sequence "<bos> Je suis" and incomplete English target sequence "a student <eos>" because there is no requirement of token ordering. Moreover, sequences can be split into many parts and these parts are trained parallel. This improvement is necessary in case of training huge corpus.

The second improvement is layer (network) normalization:

$$\text{LayerNorm}(\boldsymbol{X} + \text{Sublayer}(\boldsymbol{X}))$$

$$\text{LayerNorm}(\boldsymbol{Y} + \text{Sublayer}(\boldsymbol{Y}))$$

Because residual mechanism is implemented by the sum $\boldsymbol{X}$ + Sublayer($\boldsymbol{X}$) or $\boldsymbol{Y}$ + Sublayer($\boldsymbol{Y}$), it is necessary to survey the following normalization without loss of generality:

$$\text{LayerNorm}(\boldsymbol{x})$$

Where $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)^T$ is layer of n neuron x_i with note that each neuron x_i is represented by a number. Suppose $\boldsymbol{x}$ as a sample conforms normal distribution, its sample mean and variance are calculated as follows:

$$\overline{\boldsymbol{x}} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{\boldsymbol{x}})^2$$

As a result, layer normalization is distribution normalization:

$$\text{LayerNorm}(\boldsymbol{x}) = \left(\text{norm}(x_1), \text{norm}(x_2), \ldots, \text{norm}(x_n)\right)^T$$

$$\text{norm}(x_i) = \frac{x_i - \overline{\boldsymbol{x}}}{\sqrt{s^2}}$$

In literature, layer normalization aims to improve convergence speed in training.

It is not difficult to train transformer from corpus which can be a huge set of pairs of source/target sequences. Backpropagation algorithm associated with stochastic gradient descent (SGD) is a simple and effective choice. Feedforward sublayer represented by feedforward network (FFN) is easily trained by backpropagation algorithm associated SGD, besides attention sublayers can be trained by backpropagation algorithm associated SGD too. For instance, attention parameters for encoder such as weight matrices W_i^Q, W_i^K, W_i^V, and W^O can be learned by backpropagation algorithm associated with SGD. Attention parameters for decoder such as weight matrices W_i^Q, W_i^K, W_i^V, W^O, T, U_i^Q, U_i^K, U_i^V, and U^O can be learned by backpropagation algorithm associated SGD too. Note, starting point for backpropagation algorithm to train transformer is to make comparison of target sequence (for example, the English target sentence "I am a student" given the French source sentence "Je suis étudiant") and evaluated sequence (for example, the English evaluated sentence "We are scholars" given the same French source sentence "Je suis étudiant") at decoder, which goes backward encoder. Moreover, please pay attention that zero vectors representing redundant tokens do not affect updating these weight matrices when training transformer.

4. Pre-trained model

AI models cope with two problems of model learning: 1) it is impossible to preprocess or annotate (label) huge data so as to make the huge data better for training, and 2) huge data is often come

with data stream rather than data scratch. Note, the first problem is most important. *Transfer learning* (Han, et al., 2021, pp. 226-227) can solve the two problems by separating the training process by two stages: 1) *pre-training stage* aims to draw valuable knowledge from data stream / data scratch, and 2) *fine-tuning stage* later will take advantages of knowledge from pre-training stage so as to apply the knowledge into solving task-specific problem just by fewer samples or smaller data. As its name hints, transfer learning draws knowledge from pre-training stage and then transfers such knowledge to fine-tuning stage for doing some specific task. Capturing knowledge in pre-training stage is known as source task and doing some specific task is known as target task (Han, et al., 2021, p. 227). Source task and target task may be essentially similar like GPT model and BERT model for token generation mentioned later but these tasks can be different or slightly different. The fine-tuning stage is dependent on concrete application and so, pre-training stage is focused in this section. The purpose of pre-training stage is to build a *large-scale pre-trained model* called *PTM* which must have ability to process huge data or large-scale data. If large-scale data is come from data stream called downstream data, PTM will need to reach the strong point that is parallel computation. If large-scale is too huge, PTM will need to reach the strong point that is efficient computation. When efficient computation can be reached by good implementation, parallel computation requires an improvement of methodology. In order to catch knowledge inside data without human interference with restriction that such knowledge represented by label, annotation, context, meaning, etc. is better than cluster and group, self-supervised learning is often accepted as a good methodology for PTM (Han, et al., 2021, pp. 227-229). Essentially, *self-supervised learning* tries to draw pseudo-supervised information from unannotated/unlabeled data so that such pseudo-supervised information plays the role of supervised information like annotation and label that fine-tuning stage applies into supervised learning tasks for solving specific problem with limited data. The pseudo-supervised information is often relationships and contexts inside data structure. Anyhow, self-supervised learning is often associated with transfer learning because, simply, annotating entirely huge data is impossible. Self-supervised learning associated with pre-training stage is called self-supervised pre-training. Although self-supervised pre-training is preeminent, pre-training stage can apply other learning approaches such as supervised learning and unsupervised learning.

That the essentially strong point of transformer is self-attention makes transformer appropriate to be a good PTM when self-attention follows essentially ideology of self-supervised learning because self-attention mechanism tries to catch contextual meaning of every token inside its sequence. Moreover, transformer supports parallel computation based on its other aspect that transformer does not concern token ordering in sequence. Anyhow, transformer is suitable to PTM for transfer learning and so this section tries to explain large-scaled pre-trained model (PTM) via transformer as an example of PTM. Note, fine-tuning stage of transfer learning will take advantages of PTM for solving task-specific problem; in other words, fine-tuning stage will fine-tune or retrain PTM with downstream data, smaller data, or a smaller group of indications. When fine-tuning stage is not focused in description, PTM is known as transfer learning model which includes two stages such as pre-training stage and fine-tuning stage. In this case, source task and target task of transfer learning have the same model architecture (model backbone) which is the same PTM architecture. Large-scale PTM implies its huge number of parameters as well as huge data from which it is trained.

Generative Pre-trained Transformer (*GPT*), developed in 2018 with GPT-1 by OpenAI (www.openai.com) whose product is ChatGPT launched in 2022, is a PTM that applies only decoder of transformer into sequence generation. In pre-training stage, GPT trains its decoder from

huge data over internet and available sources so as to predict next word $\boldsymbol{y}_{t+1}$ from previous words $\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_t$ by maximizing likelihood $P(\boldsymbol{y}_{t+1} \mid \Theta, \boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_t)$ and taking advantages of self-attention mechanism aforementioned (Han, et al., 2021, p. 231). Maximization of likelihood $P(\boldsymbol{y}_{t+1} \mid \Theta, \boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_t)$ belongs to autoregressive language model.

$$\widehat{\Theta} = \underset{\Theta}{\operatorname{argmax}} \log\left(P(\boldsymbol{y}_{t+1} | \Theta, \boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_t)\right)$$

$$\widehat{\boldsymbol{y}}_{t+1} = \underset{\boldsymbol{y}_{n+1}}{\operatorname{argmax}} P\left(\boldsymbol{y}_{t+1} \middle| \widehat{\Theta}, \boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_t\right)$$

Where,

$$P(\boldsymbol{y}_{t+1} | \Theta, \boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_t) \stackrel{\text{def}}{=} \boldsymbol{p}$$

$$\boldsymbol{p} = \text{softmax}(\boldsymbol{w}) = \left(p_1, p_2, \ldots, p_{|\Omega|}\right)^T$$

$$p_i = \frac{\exp(w_i)}{\sum_{l=1}^{|\Omega|} \exp(w_l)}$$

$$\boldsymbol{w} = \left(w_1, w_2, \ldots, w_{|\Omega|}\right)^T = \text{linear}(\boldsymbol{Z}) = \text{FFN}(\boldsymbol{Z})$$

And,

$$\text{MultiheadAttention}(\boldsymbol{X}, \boldsymbol{Y}) = \text{LayerNorm}\left(\text{MaskedMultiheadAttention}(\boldsymbol{Y}) + \text{MultiheadAttention}(\boldsymbol{X}, \boldsymbol{Y})\right)$$

$$\boldsymbol{Z} = \text{DecoderAttention}(\boldsymbol{X}, \boldsymbol{Y}) = \text{LayerNorm}\left(\text{MultiheadAttention}(\boldsymbol{X}, \boldsymbol{Y}) + \text{FFN}\left(\text{MultiheadAttention}(\boldsymbol{X}, \boldsymbol{Y})\right)\right)$$

Because GPT has only one decoder, sequence $\boldsymbol{X}$ is null in GPT.

$$\text{MultiheadAttention}(\boldsymbol{X}, \boldsymbol{Y}) = \text{MultiheadAttention}(\boldsymbol{Y})$$

$$\text{DecoderAttention}(\boldsymbol{X}, \boldsymbol{Y}) = \text{DecoderAttention}(\boldsymbol{Y})$$

Likelihood $P(\boldsymbol{y}_{t+1} \mid \Theta, \boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_t)$ is simplified for easy explanation. Exactly, given sequence $\boldsymbol{Y} = (\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_{n+1})^T$, GPT aims to maximize log-likelihood $L(\Theta \mid \boldsymbol{Y})$ as follows (Han, et al., 2021, p. 231):

$$L(\Theta | \boldsymbol{Y}) = \sum_{i=1}^{n} \log\left(P(\boldsymbol{y}_{i+1} | \Theta, \boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_i)\right)$$

$$\widehat{\Theta} = \underset{\Theta}{\operatorname{argmax}}\, L(\Theta | \boldsymbol{Y})$$

Later on, GPT improves its pre-trained decoder in fine-tuning stage by re-training the decoder with annotated data, high-quality data, and domain-specific data so as to improve pre-trained parameters. Moreover, GPT adds extra presentation layers in fine-tuning stage (Han, et al., 2021, p. 231). Following figure (Han, et al., 2021, p. 232) depicts prediction process of GPT.

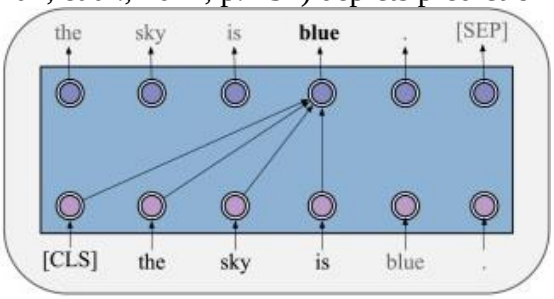

Figure 4.1. Prediction process of GPT

Bidirectional Encoder Representations from Transformers (*BERT*), developed in 2018 by Google, is a PTM that applies only encoder of transformer into sequence generation. In pre-training stage,

BERT trains its encoder from huge data over internet and available sources. Given (t+1)-length sequence $(\boldsymbol{x}_1, \boldsymbol{x}_2,\ldots, \boldsymbol{x}_{t+1})^T$, BERT applies masked language model to randomize an unknown token at random position denoted *masked* where the random index *masked* is randomized in t+1 indices $\{1, 2,\ldots, t+1\}$ with note that the randomization process can be repeated many times. Such unknown token, which is called masked token denoted $\boldsymbol{y}_{masked}$, will be predicted given t-length sequence $(\boldsymbol{x}_1, \boldsymbol{x}_2,\ldots, \boldsymbol{x}_t)^T$ without loss of generality. In order words, masked words $\boldsymbol{x}_{masked}$ is predicted from other words $\boldsymbol{x}_1, \boldsymbol{x}_2,\ldots, \boldsymbol{x}_t$ by maximizing likelihood $P(\boldsymbol{x}_{masked} \mid \Theta, \boldsymbol{x}_1, \boldsymbol{x}_2,\ldots, \boldsymbol{x}_t)$ and taking advantages of self-attention mechanism aforementioned (Han, et al., 2021, p. 232).

$$\widehat{\Theta} = \underset{\Theta}{\operatorname{argmax}} \log\left(P(\boldsymbol{x}_{masked}|\Theta, \boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_t)\right)$$

$$\widehat{\boldsymbol{y}}_{masked} = \underset{\boldsymbol{y}_{masked}}{\operatorname{argmax}} P\left(\boldsymbol{x}_{masked}\middle|\widehat{\Theta}, \boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_t\right)$$

Where,

$$P(\boldsymbol{x}_{masked}|\Theta, \boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_t) \stackrel{\text{def}}{=} \boldsymbol{p}$$

$$\boldsymbol{p} = \text{softmax}(\boldsymbol{w}) = \left(p_1, p_2, \ldots, p_{|\Omega|}\right)^T$$

$$p_i = \frac{\exp(w_i)}{\sum_{l=1}^{|\Omega|} \exp(w_l)}$$

$$\boldsymbol{w} = \left(w_1, w_2, \ldots, w_{|\Omega|}\right)^T = \text{linear}(\boldsymbol{Z}) = \text{FFN}(\boldsymbol{Z})$$

And,

$$\text{MultiheadAttention}(\boldsymbol{X}) = \text{LayerNorm}\left(\boldsymbol{X} + \text{MultiheadAttention}(\boldsymbol{X})\right)$$

$$\boldsymbol{Z} = \text{EncoderAttention}(\boldsymbol{X})$$

$$= \text{LayerNorm}\left(\text{MultiheadAttention}(\boldsymbol{X}) + \text{FFN}\left(\text{MultiheadAttention}(\boldsymbol{X})\right)\right)$$

Likelihood $P(\boldsymbol{y}_{masked} \mid \Theta, \boldsymbol{x}_1, \boldsymbol{x}_2,\ldots, \boldsymbol{x}_m)$ is simplified for easy explanation, thereby it is necessary to explain more how BERT defines and maximizes likelihood with support of masked language model. Given sequence $\boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2,\ldots, \boldsymbol{x}_m)^T$, let $R = \{r_1, r_2,\ldots, r_k\}$ be the set of indices whose respective tokens are initially masked, for instance, token $\boldsymbol{x}_{r_j}$ will be initially masked if r_j belongs to mask set R. Let $\boldsymbol{X}_{r_j}$ be the set of r_j–1 tokens which are unmasked later, for instance, the tokens $\boldsymbol{x}_{r_1}, \boldsymbol{x}_{r_2},\ldots, \boldsymbol{x}_{r_j-1}$ which were initially masked before are now unmasked (known) at current iteration. Note, the set R is called mask set or mask pattern and $\boldsymbol{X}_{r_j}$ does not include token $\boldsymbol{x}_{r_j}$. BERT randomizes k masked indices so as to establish mask set R. Let S be the set of indices whose tokens are always known, which is the complement of mask set R with regard to all indices so that union of R and S is $\{1, 2,\ldots, m\}$. Thereby, let $\boldsymbol{S}$ be the set of tokens whose indices are in S. In other words, $\boldsymbol{S}$ contains tokens which are always known. BERT aims to maximize log-likelihood $L(\Theta \mid \boldsymbol{X})$ as follows (Han, et al., 2021, p. 232):

$$L(\Theta|\boldsymbol{X}) = \sum_{r_j \in R} \log\left(P\left(\boldsymbol{x}_{r_j}\middle|\Theta, \boldsymbol{X}_{r_j}, \boldsymbol{S}\right)\right)$$

$$\widehat{\Theta} = \underset{\Theta}{\operatorname{argmax}}\, L(\Theta|\boldsymbol{X})$$

Later on, BERT improves its pre-trained encoder in fine-tuning stage by re-training the encoder with annotated data, high-quality data, and domain-specific data so as to improve pre-trained parameters. By support of masked language model (autoencoding language model) for masking tokens, BERT can predict a token at any position in two directions given a list of other tokens while GPT only predicts a token at next position given previous tokens. The name "BERT", which

is abbreviation of “Bidirectional Encoder Representations from Transformers”, hints that BERT can generate tokens/words in bidirectional way at any positions. Therefore, GPT is appropriate to language generation and BERT is appropriate to language understanding (Han, et al., 2021, p. 231). BERT also adds extra presentation layers in fine-tuning stage (Han, et al., 2021, p. 232). Following figure depicts prediction process of BERT.

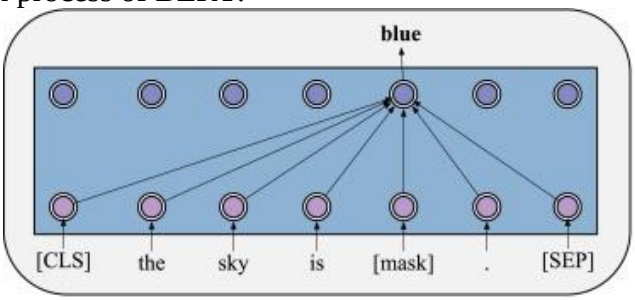

Figure 4.2. Prediction process of BERT

Recall that given a transfer model, capturing knowledge in pre-training stage is known as source task and doing some specific task is known as target task (Han, et al., 2021, p. 227), thereby there is a question that how source task transfers knowledge to target task or how PTM makes connection between source task and knowledge task. The answer is that there are two transferring approaches such as feature transferring and parameter transferring (Han, et al., 2021, p. 227). *Feature transferring* converts coarse data like unlabeled data into fine data like labeled data so that fine data considered as feature is fed to fine-tuning stage. *Parameter transferring* transfers parameters learned at pre-training stage to fine-tuning stage. If pre-training stage and fine-tuning stage share the same model architecture which is the same PTM architecture, parameter transferring will always occur in PTM. Both GPT and BERT apply parameter transferring because they will initialize or set up their models such as GPT decoder and BERT encoder by billions of parameters that were learned in pre-training stage with the same model architecture (model backbone) such as GPT decoder and BERT encoder before they perform fining-tuning task in fine-tuning stage. Self-supervised learning which trains unlabeled data is appropriate to pre-training stage because unlabeled data is much more popular than labeled data, thereby parameter transferring is often associated with self-supervised learning. Because transformer is suitable to self-supervised learning due to its self-attention mechanism, parameter transferring is suitable to PTMs like GPT and BERT. Moreover, if they apply transformer into annotating or creating task-specific data / fine data for improving their decoder and encoder in fine-tuning stage, they will apply feature transferring too. In general, within parameter transferring and same architecture, PTM itself is backbone for both pre-training stage and fine-tuning stage.

5. Conclusions

As the paper title “Attention is all you need” (Vaswani, et al., 2017) hints, attention-awarded transformer is the important framework for generative artificial intelligence and statistical translation machine whose applications are not only large but also highly potential. For instance, it is possible for transformer to generate media content like sound, image, video from texts, which is very potential for cartoon industry and movie making applications (film industry). The problem of difference in source data and target data, which can be that, for example, source sequence is text sentence and target sequence is raster data like sound and image, can be solved effectively and smoothly because of two aforementioned strong points of transformer such as self-attention and not concerning token ordering. Moreover, transformer’s methodology is succinct with support of

encoder-decoder mechanism and deep neural network. Therefore, it is possible to infer that applications of transformer can go beyond some recent pre-trained models and/or pre-trained models based on transformer can be improved more.

References

Alammar, J. (2018, June 27). *The Illustrated Transformer*. (GitHub) Retrieved June 2024, from Jay Alammar website: https://jalammar.github.io/illustrated-transformer

Cho, K., Merrienboer, B. v., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014, September 3). Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *arXiv preprint*, 1-15. doi:10.48550/arXiv.1406.1078

Graves, A. (2014, June 5). Generating Sequences With Recurrent Neural Networks. *arXiv preprint*, 1-43. doi:10.48550/arXiv.1308.0850

Han, X., Zhang, Z., Ding, N., Gu, Y., Liu, X., Huo, Y., . . . Zhu, J. (2021, August 26). Pre-trained models: Past, present and future. *AI Open, 2*(2021), 225-250. doi:10.1016/j.aiopen.2021.08.002

Hardle, W., & Simar, L. (2013). *Applied Multivariate Statistical Analysis.* Berlin, Germany: Research Data Center, School of Business and Economics, Humboldt University.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., . . . Polosukhin, I. (2017). Attention Is All You Need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, & S. Vishwanathan (Ed.), *Advances in Neural Information Processing Systems (NIPS 2017). 30.* Long Beach: NeurIPS. Retrieved from https://arxiv.org/abs/1706.03762

Voita, L. (2023, November 17). *Sequence to Sequence (seq2seq) and Attention*. (GitHub) Retrieved June 2024, from Elena (Lena) Voita website: https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html

Wikipedia. (2005, April 7). *Recurrent neural network.* (Wikimedia Foundation) Retrieved from Wikipedia website: https://en.wikipedia.org/wiki/Recurrent_neural_network

Wikipedia. (2019, August 25). *Transformer (deep learning architecture)*. (Wikimedia Foundation) Retrieved from Wikipedia website: https://en.wikipedia.org/wiki/Transformer_(deep_learning_architecture)