2 | Introducing Sequential Models with applications in Natural Language Processing

2.1 The Hidden Markov Model

Hidden Markov Models (HMMs) are powerful graphical models used to describe sequential data. They were first introduced in a series of statistical papers by Leonard E. Baum (Baum & Petrie, 1966) and other authors in the second half of the 1960s. Besides their rich mathematical structure, HMMs work very well in a wide range of real world applications. Due to the successful results in Speech Recognition (Rabiner & Juang, 1986), the model has become increasingly popular and has been applied to several other areas such as natural language modeling (Manning & Schutze, 1999). There are also many applications in finance : (S. Kim, Shephard, & Chib, 1998) used HMMs for the analysis of stochastic volatility, in comparison with the traditional GARCH model. In (Nystrup, Madsen, & Lindström, 2017), the authors used the model to infer the hidden states associated with the daily returns in financial markets. Other applications include handwriting recognition (Nag, Wong, & Fallside, 1986) and bioinformatics (Krogh, Brown, Mian, Sjölander, & Haussler, 1994), (Durbin, Eddy, Krogh, & Mitchison, 1998), (Baldi & Brunak, 2001), (Manogaran et al., 2018), (Testa, Hane, Ellwood, & Oliver, 2015).

HMMs can be viewed as a dynamical mixture model. Indeed, the model is a simple generalization of the mixture model. It assumes the existence of latent variables called "hidden states", which are responsible of the observable data. the HMM framework no longer assumes that the hidden states are chosen independently as in vanilla mixture models. It rather assumes the latent states form an unobservable Markov chain.

2.2 Introducing Recurrent Neural Networks

2.2.1 From Hidden Markov Models to Recurrent Neural Networks

Hidden Markov Models (HMM) have tackled the problem of modeling sequences for decades, especially in the context of Speech Recognition (Rabiner & Juang, 1986).

HMM models were extremely popular in the 1980s for their rich mathematical structure and their good performances in a wide range of applications. However, they rely on a Markov assumption on the temporal structure of their hidden states, making the model inefficient for modeling long range dependencies.

On the other hand, Recurrent Neural Networks, with their significantly richer memory and computational

capacity, have attained state of the art performances in applications such as Speech Recognition (Graves, Mohamed, & Hinton, 2013). The following section will introduce Recurrent Neural Networks.

2.2.2 Vanilla Recurrent Neural Networks

Feed-forward neural networks make the assumption that the data is independent and identically distributed (i.i.d). Recurrent Neural Networks (RNNs) (Rumelhart, Hinton, & Williams, 1986) are a family of neural networks capable of processing the data in a sequential way. Figure 2.1 is representation of the vanilla RNN architecture.

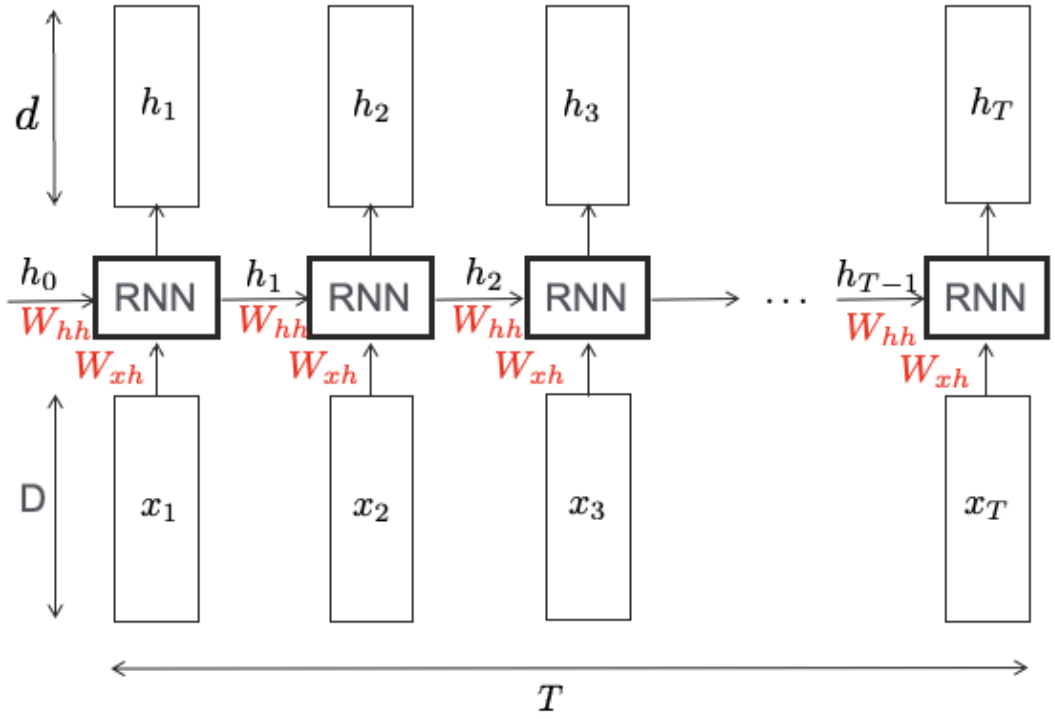


FIGURE 2.1 – Vanilla RNN

The objective is to process an input sequence of D-dimensional vectors $x_1, \ldots, x_T$ sequentially in order to generate a sequence of d-dimensional hidden states $h_1, \ldots, h_T$. The model is parameterized by the two matrices $W_{xh} \in \mathbb{R}^{D \times d}$ and $W_{hh} \in \mathbb{R}^{d \times d}$

At each time step $t \in \{1, \ldots, T\}$, there are two inputs to the hidden layer : the previous hidden state $h_{t-1} \in \mathbb{R}^d$, and the input vector at that time step x_t. The former input vector is processed by the weight matrix W_{hh} and the latter by the weight matrix W_{xh} in order to produce the next hidden state h_t, as shown in equation 2.2.1

$$\forall t \in \{1, \dots, T\} \quad h_t = \tanh\left(W_{hh}^T h_{t-1} + W_{xh}^T x_t\right) \tag{2.2.1}$$

Unfortunately, it is difficult to access information from many steps back if T is too large due to problems like vanishing or exploding gradients (Bengio, Simard, & Frasconi, 1994).

— **The exploding gradient problem :** This problem refers to the large increase in the gradient norm during training. In order to overcome this issue, Thomas Mikolov introduced a simple heuristic solution called **gradient clipping** in his PhD thesis in 2012. It consists in rescaling the gradient norm, whenever it goes over a threshold. The theoretical justification was then proposed in (Pascanu et al., 2013).

— **The vanishing gradient problem :** (Pascanu et al., 2013) showed that gradient signal from far away is lost because it's a lot smaller than the gradient signal from close-by.

To get the intuition of this issue, let us consider the example of a classification problem using an RNN.

We recall the equation that describes the hidden states transition,

$$\forall t \in \{1, \dots, T\} \quad h_t = \sigma\left(W_{hh}^T h_{t-1} + W_{xh}^T x_t\right) \tag{2.2.2}$$

Let us consider an output layer for the classification task with K possible outputs :

$$\hat{y}_T = \text{Softmax}\left(W_o^T h_T + b_o\right) \tag{2.2.3}$$

Where $W_o \in \mathbb{R}^{d \times K}$ and $b_o \in \mathbb{R}^K$

Hence, the loss function J we wish to optimize is the cross entropy between the final prediction $\hat{y}_T$ and the true output y.

In order to optimize the loss function, we will use Backpropagation through time as explained in (Werbos, 1990).

For that, we need to calculate $\frac{\partial J(\theta)}{\partial h_t}$ for all $t \in \{1, \dots, T\}$ using the Jacobian matrices $(\frac{\partial h_t}{\partial h_{t-1}})_{2 \leq t \leq T}$ as follows :

$$\forall t \in \{1, \dots, T\} \quad \frac{\partial J(\theta)}{\partial h_t} = \frac{\partial J(\theta)}{\partial h_T} \prod_{t < t' \leq T} \frac{\partial h_{t'}}{\partial h_{t'-1}}$$

We would like to explain the vanishing gradient problem for a simple case (a linear activation function). The proof in the general case can be found in (Pascanu et al., 2013).

1. **Simple case : If σ is the identity function**

   In that case,

$$\begin{aligned} \forall t \in \{1, \dots, T\} \quad \frac{\partial h_t}{\partial h_{t-1}} &= \operatorname{diag}\left(\sigma'\left(W_{hh}^T h_{t-1} + W_{xh}^T x_t\right)\right) W_{hh} \\ &= I W_{hh} \\ &= W_{hh} \end{aligned}$$

   And consequently, by taking the gradient of the objective function J with respect to the t-th hidden state, we get :

$$\begin{aligned} \forall t \in \{1, \dots, T\} \quad \frac{\partial J(\theta)}{\partial h_t} &= \frac{\partial J(\theta)}{\partial h_T} \prod_{t < t' \leq T} \frac{\partial h_{t'}}{\partial h_{t'-1}} \\ &= \frac{\partial J(\theta)}{\partial h_T} W_{hh}^{T-t} \end{aligned}$$

   The vanishing gradient problem occurs when the matrix W_{hh} is too "small". Let us suppose for example that the eigen values of W_{hh}, denoted $\lambda_1, \dots, \lambda_M$, verify : $\forall i \in \{1, \dots, M\} \quad |\lambda_i| < 1$.

   Then,

$$\forall t \in \{1, \dots, T\} \quad \frac{\partial J(\theta)}{\partial h_t} = \sum_{m=1}^{M} c_m \lambda_m^{T-t} q_m \quad (\text{with } c_1, \dots, c_M \in \mathbb{R})$$

   When $T - t$ is too large, λ_m^{T-t} approaches 0, so the gradient **vanishes**.

2. **Generalization : If σ is a non linear activation function**

   In (Pascanu et al., 2013), the authors generalize the result for the nonlinear activation function σ. They get the same result with a stronger assumption : The module of the eigen values of W_{hh} should be smaller than a γ value, which depends on the dimensionality of the vectors and the activation function σ.

In order to address the vanishing gradient problem, (Pascanu et al., 2013) used a regularization term to ensure that the back-propagated gradients neither increase or decrease in magnitude. The regularization term forces the Jacobian matrices to preserve the norm only in relevant directions.

Beyond the solutions discussed so far, new architectures based on gated activation function have been proposed to capture long-term dependencies. The first architecture of that kind, called Long Short-Term

Memory, was proposed by Hochreiter and Schmidhuber in 1997 (Hochreiter & Schmidhuber, 1997) and then refined in (Gers et al., 2000). Hochreiter proposed an architecture using two gates, namely an input gate and an output gate in the original LSTM paper (Hochreiter & Schmidhuber, 1997), while Gers added a forget gate in (Gers et al., 2000). More recently, Gated recurrent units (Cho et al., 2014) were introduced in the context of Neural Machine Translation using two gates to control the information flow from the previous steps.

2.2.3 Long Short-Term Memory (LSTM)

Before transformers, LSTMs achieved state-of-the-art results in a wide range of tasks, including machine translation (Sutskever, Vinyals, & Le, 2014), (Cho et al., 2014) and (Bahdanau, Cho, & Bengio, 2015), language modeling (Sundermeyer, Schlüter, & Ney, 2012), model identification (Z. Wang et al., 2018), time series prediction (Y. Li & Cao, 2018) and Robot Reinforcement Learning (Bakker et al., 2001)

So far, we have introduced Recurrent Neural Networks, which transition from the hidden state h_{t-1} to h_t using a linear transformation and a point-wise non-linearity, as described in equation 2.2.1.

Here, on step t, there is a **hidden state** h^t and a **cell state** c^t that stores long-term information. The core intuition behind LSTMs is to regulate the information that is removed or added to the cell state through different **gates**.

Let us dive into the concept of gate, as shown in figure 2.2a :

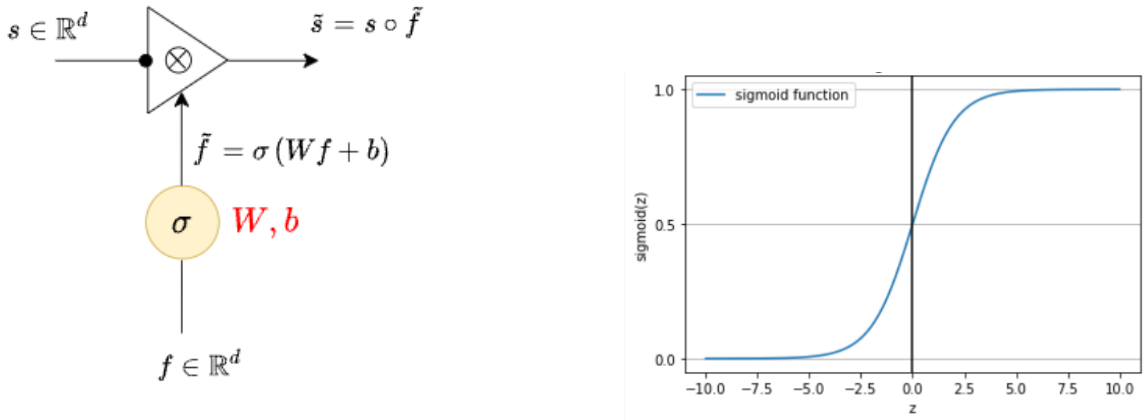
— The signal vector $s \in \mathbb{R}^d$ represents the information that we want to filter using the processing of the vector $f \in \mathbb{R}^d$.

— The processing of the vector f is done through a sigmoid layer, which consists in a linear transformation parameterized by (W, b) and a point-wise non-linearity (using the sigmoid activation function). The output of this layer is the **gate vector** $\tilde{f} := \sigma\left(Wf + b\right)$.

— Figure 2.2b is a representation of the sigmoid function $\sigma : z \mapsto \frac{1}{1+e^{-z}}$. So, the gate vector $\tilde{f}$ is composed of numbers close to zero or close to one, or somewhere in between, describing how much of each dimension (among the d dimensions) we would like to let through.

— The final step is a point-wise multiplication between the signal s and the gate vector $\tilde{f}$. As a result, some dimensions of s are going to be multiplied by values close to 1 in $\tilde{f}$, these dimensions are the ones we would like to keep. On the other hand, some dimensions of s are going to be multiplied by values close to 0, which are the dimensions that need to be updated.

— To get more intuition about this concept, let us suppose that the s signal represents the "memory vector" encoding dimensions like "gender" in the context of language modeling. The gate vector $\tilde{f}$,



(a) Filtering a signal using a sigmoid function and a neural network

(b) The sigmoid function

FIGURE 2.2 – The concept of gate used in LSTMs

by processing a new word, might detect a new "gender". Thus, it will filter the "gender" dimension of the s signal so that it could be replaced with a representation of the new "gender".

As shown in figure 2.3, the LSTM architecture controls the information flow of the cell state using three gates. Let us explain, step by step, how we transition from the couple of the previous hidden and cell states (h^{t-1}, C^{t-1}) to the new couple of hidden and cell states (h^t, C^t) using the "fresh" information x^t.

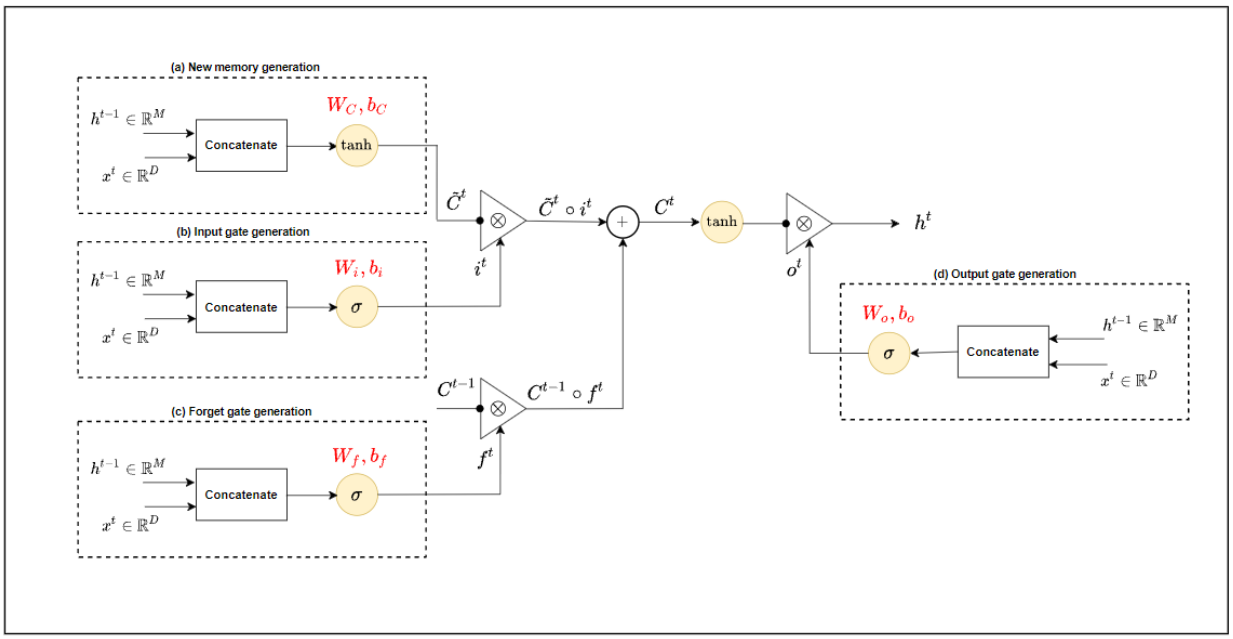


FIGURE 2.3 – The Long Short-Term Memory architecture

— **Step (1) : Generating the new memory candidate**

During this stage, we essentially combine the newly observed vector x^t and the previous hidden

state h^{t-1} to generate a new vector $\tilde{C}^t$ (eq 2.2.4), as shown in figure 2.3 part(a). Therefore, the vector $\tilde{C}^t$ can be seen as a representation of the fresh vector x^t in light of the contextual past. The parameters involved are $W_C \in \mathbb{R}^{(D+M)\times M},\ b_C \in \mathbb{R}^M$. The activation function ($\tanh$) pushes the values between -1 and 1.

$$\tilde{C}^t = \tanh\left(W_C[h^{t-1}, x^t] + b_C\right) \tag{2.2.4}$$

where $[h^{t-1}, x^t] \in \mathbb{R}^{D+M}$ is the concatenation of h^{t-1} and x^t

— **Step (2) : Generating the input gate vector**

As shown in figure 2.3 part(b), we use the x^t vector and h^{t-1} to generate the input gate vector i^t (eq 2.2.5) in order to determine which dimensions of the generated $\tilde{C}^t$ are worth preserving. The parameters involved are $W_i \in \mathbb{R}^{(D+M)\times M},\ b_i \in \mathbb{R}^M$.

$$i^t = \sigma\left(W_i[h^{t-1}, x^t] + b_i\right) \tag{2.2.5}$$

where σ stands for the sigmoid function

— **Step (3) : Filtering the new candidate using the input gate**

We use the input gate vector i^t to filter the new candidate $\tilde{C}^t$, which results in the point-wise multiplication $\tilde{C}^t \circ i^t$ (where $\circ$ stands for the Hadamard product).

— **Step (4) : Generating the forget gate vector**

Similarly to the input gate vector, the vectors h^{t-1} and x^t are used to generate the forget gate vector f^t (eq 2.2.6). This time, instead of assessing the usefulness of the new generated vector $\tilde{C}^t$ dimensions, the forget gate vector is used to filter the dimensions of the previous memory vector C^{t-1} that need to be updated.

$$f^t = \sigma\left(W_f[h^{t-1}, x^t] + b_f\right) \tag{2.2.6}$$

where σ stands for the sigmoid function

— **Step (5) : Filtering the previous cell state using the forget gate**

Similarly to Step (3), we filter the previous cell state C^{t-1} using the forget fate vector f^t, which results in the point-wise multiplication $C^{t-1} \circ f^t$.

— **Step (6) : Getting the final memory state**

At this stage, we combine the advice of the forget gate vector f^t filtering the past memory C^{t-1}, and the advice of the input gate vector i^t filtering the new generated memory $\tilde{C}^t$. The sum of both results produce the final cell state C^t (eq 2.2.7).

$$C^t = i^t \circ \tilde{C}^t + f^t \circ C^{t-1} \tag{2.2.7}$$

— **Step (7) : Generating the output gate vector**

As the hidden states are used as an input to every single gate (forget, input and output gates), the output gate vector o^t makes the assessment regarding what parts of the cell state (the memory state) C^t need to be exposed in the hidden state h^t. This final step decides what parts of the memory C^t need to be present in the hidden states h^t.

$$o^t = \sigma\left(W_o[h^{t-1}, x^t] + b_o\right) \tag{2.2.8}$$

where σ stands for the sigmoid function

— **Step (8) : Getting the final hidden state**

At this stage, we use the o^t vector to gate the point-wise $\tanh$ of the memory vector C^t, which results in the final hidden state h^t (eq 2.2.9). Moreover, at each time step t, we have the possibility to output the hidden state h^t to represent the processing of the first t input vectors $x^1, \ldots, x^t$.

$$h^t = o^t \circ \tanh(C^t) \tag{2.2.9}$$

To sum up, the LSTM architecture can **write**, **erase** or **read** information from the cell state. For each of these actions, the LSTM makes use of gates to decide which dimensions to modify. All the equations involved in the LSTM architecture are then summarized as follows :

— **Erasing** the content of the previous cell state using the **forget gate** $f^t = \sigma\left(W_f[h^{t-1}, x^t] + b_f\right)$, which results in $f^t \circ C^{t-1}$.

— **Writing** new content to the cell state, using the **input gate** $i^t = \sigma\left(W_i[h^{t-1}, x^t] + b_i\right)$, which results in $i^t \circ \tanh(\tilde{C}^t)$

— **Reading** some content from the cell state $C^t = f^t \circ C^{t-1} + i^t \circ \tanh(\tilde{C}^t)$ using the **output gate** $o^t = \sigma\left(W_o[h^{t-1}, x^t] + b_o\right)$, which results in the hidden state $h^t = o^t \circ \tanh(C^t)$

It's worth noticing that all the weight matrices W_C, W_f, W_o, W_i are applied to the concatenation of h^{t-1} and x^t.

Let us consider a couple of parameters $(W, b) \in \{(W_C, b_C), (W_i, b_i), (W_f, b_f), (W_o, b_o)\}$ and an activation function σ (sigmoid or $\tanh$). As $(W, b) \in \mathbb{R}^{(M+D)\times M} \times \mathbb{R}^M$, we can rewrite W as $\begin{bmatrix} U & V \end{bmatrix}$ with $U \in \mathbb{R}^{M\times M}$ and $V \in \mathbb{R}^{D\times M}$, which leads to the following equation :

$$\sigma\left(W[h^{t-1}, x^t] + b\right) = \sigma\left(Uh^{t-1} + Vx^t + b\right) \tag{2.2.10}$$

The LSTM architecture doesn't guarantee to solve the vanishing or exploding gradient problems. However, it ensures the model to learn long-distance dependencies by preserving the information in the cell state. Indeed, the information in a particular dimension is preserved if the forget gate vector and the input gate vector are respectively set to 1 and 0 for this particular dimension. It's much harder for a vanilla RNN to learn weight matrices W_{hh} and W_{xh} that preserve the information in any dimension of the hidden states.

2.2.4 Gated Recurrent Units

The Gated Recurrent Units was introduced in 2014 (Cho et al., 2014) by Kyunghyun Cho. The GRU structure is similar to the LSTM one, except that there isn't any cell state and it only contains 2 gates.

Let us dive into the GRU architecture, represented in figure 2.4

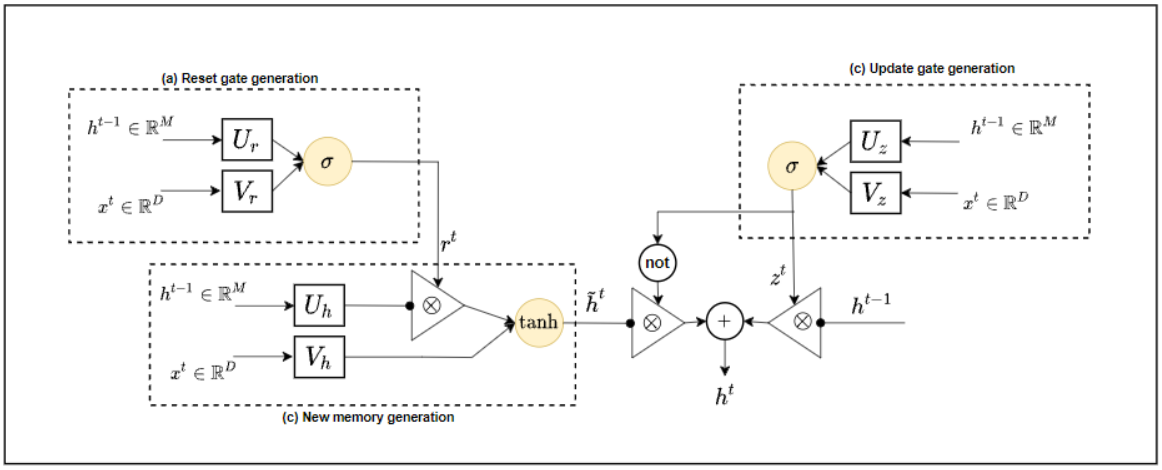


FIGURE 2.4 – The Gated Recurrent Units architecture

— **Step (1) : Generating the reset gate vector**

The objective of the reset gate vector (eq 2.2.11) is to determine which dimensions of the previous

hidden state h^{t-1} are relevant to the computation of the new memory candidate. Similarly to the LSTM gates, it is computed by processing the previous hidden state h^{t-1} and the newly observed vector x^t, in the reduced way described in equation 2.2.10.

$$r^t = \sigma\left(U_r h^{t-1} + V_r x^t\right) \tag{2.2.11}$$

where σ stands for the sigmoid function

— **Step(2) : Generating the new memory candidate**

This stage is analogous to the memory generation in the LSTM architecture. The new candidate $\tilde{h}^t$ is obtained by combining the information from the newly observed vector x^t and the filtered version of the previous hidden state using the reset gate vector r^t. A point-wise $\tanh$ is then applied to push the values between -1 and 1 (eq 2.2.12).

$$\tilde{h}^t = \tanh\left(r^t \circ U_h h^{t-1} + V_h x^t\right) \tag{2.2.12}$$

where $\circ$ stands for the Hadamard product.

— **Step (3) : Generating the update gate vector**

The update gate vector z^t is responsible for balancing between the information from the previous hidden state h^{t-1} and the information from the new memory candidate $\tilde{h}^t$. The vector is calculated using the previous hidden state h^{t-1} and the newly observed vector x^t using the parameters U_z, V_z (eq 2.2.13)

$$z^t = \sigma\left(U_z h^{t-1} + V_z x^t\right) \tag{2.2.13}$$

where σ stands for the sigmoid function

— **Step (4) : Getting the final hidden state**

The final hidden state h^t is generated using the update gate vector z^t. As showed in equation 2.2.14, if a particular dimension of z^t is close to 1, it means that almost all the information from the previous hidden state is copied out to h^t. Conversely, if it's close to 0, it means that the information from $\tilde{h}^t$ should be carried forward to h^t.

$$h^t = z^t \circ h^{t-1} + (1 - z^t) \circ \tilde{h}^t \tag{2.2.14}$$

2.2.5 Different applications of RNNs

With the LSTM or the GRU model, we have discussed how to learn a mapping from the input space $\mathcal{X}$ into the hidden space $\mathcal{H}$.

This framework can support three types of applications, corresponding to the fact that at least one of the spaces $\mathcal{X}$ and $\mathcal{H}$ encodes sequences.

— The **One to Many** application : It consists in learning mapping functions of the form $\Phi_\theta : X \in \mathcal{X} = \mathbb{R}^D \mapsto (h_i^1, \ldots, h_i^T) \in \mathcal{H} = \mathbb{R}^{T \times d}$ using the LSTM/GRU model $\mathcal{E}_\theta$ as shown in figure 2.5. Image captioning (Vinyals, Toshev, Bengio, & Erhan, 2015) is a typical example, where the description of an image is generated. An image is mapped into a feature vector, which in turn becomes the input for an LSTM architecture.

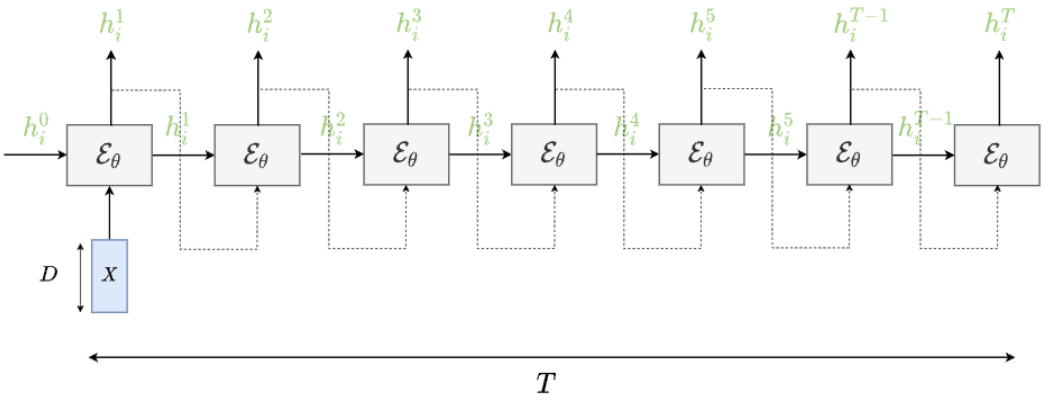


FIGURE 2.5 – The Vector to Sequence framework

— The **Many to One** application : It consists in learning mapping functions of the form $\Phi_\theta : (X_i^1, \ldots, X_i^T) \in \mathcal{X} = \mathbb{R}^{T \times D} \mapsto h_i^T \in \mathcal{H} = \mathbb{R}^d$ using the LSTM/GRU model $\mathcal{E}_\theta$ as shown in figure 2.6. Sentiment Analysis (Murthy, Allu, Andhavarapu, Bagadi, & Belusonti, 2020) is a typical example, where each sentence is mapped to the last hidden hidden state of the LSTM layer. A final Dense layer is then used to predict the sentiment of the sentence (positive, negative or neutral for instance).

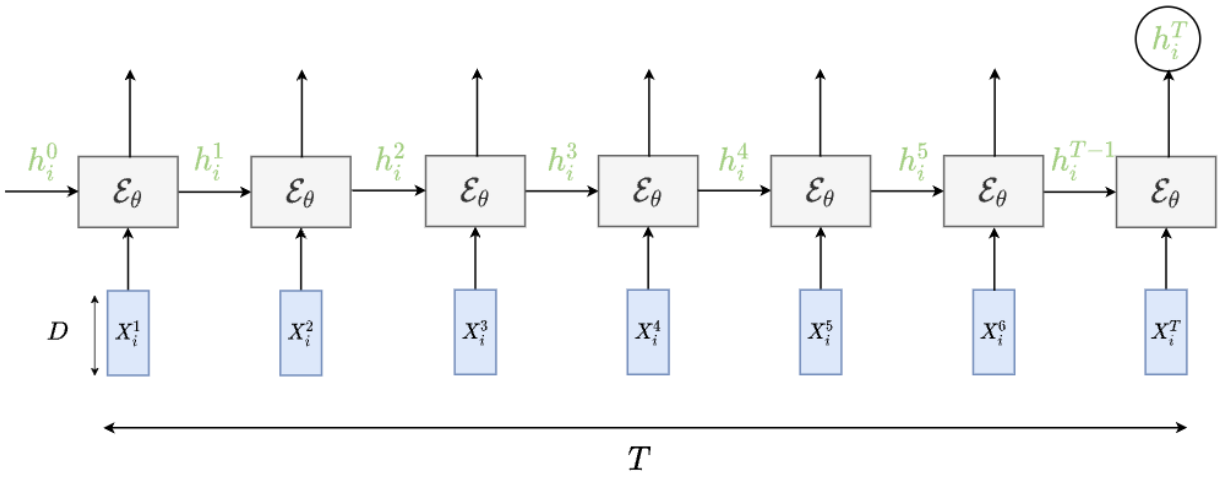


FIGURE 2.6 – The sequence to sequence framework

— The **Many to Many** application : It consists in learning mapping functions of the form $\Phi_\theta : (X_i^1, \ldots, X_i^T) \in \mathcal{X} = \mathbb{R}^{T \times D} \mapsto (h_i^1, \ldots, h_i^T) \in \mathcal{H} = \mathbb{R}^{T \times d}$ using the LSTM/GRU model $\mathcal{E}_\theta$ as shown in figure 2.7. Part of Speech Tagging (P. Wang, Qian, Soong, He, & Zhao, 2015) is a typical example, where the objective is to tag each word of a sentence with its "Part-of-Speech" tag.

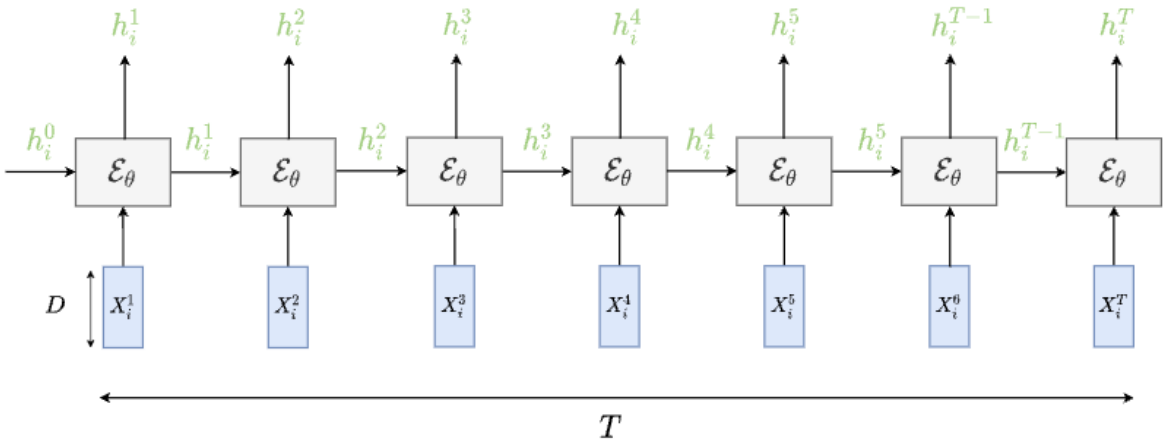


FIGURE 2.7 – The sequence to sequence framework

2.2.6 Applying the RNN/LSTM Model to predict the next word

Word Embeddings

The English Vocabulary contains approximately $V = 13 \times 10^6$ words (or tokens).

The first step in any NLP problem is to map the V tokens into a D-dimensional [1] space encoding all semantics of the language. Each dimension is responsible of some meaning like the gender, the tense,

1. In general, D is between 50 and 300

etc.

We can first represent the tokens as numbers in $\{1, \ldots, V\}$. Each token is then associated with a unique integer $\in \{1, \ldots, V\}$.

Rather than representing the tokens by their indexes, the equivalent representation would be to represent them as vectors of size V with 1 at the index position and zeroes in all the other positions. These vectors are called the **one hot vectors** associated with the tokens.

Predicting the next word using LSTM architectures

For that, we consider a corpus composed of several documents in some language containing V possible words. Using an unsupervised model, we can map each word with a D dimensional vector called **embedding**.

We would like for instance to predict the next word based on the previous T words. Therefore, we organise the dataset in sequences of words that we would like to map to the next words.

We denote each sequence $w_i = (w_i^1, \ldots, w_i^T) \in [V]^T$ (where $[V]$ stands for $\{1, \ldots, V\}$) and the label associated with it $y_i \in [V]$. So, the training data is $\mathcal{T} = \{(w_i, y_i)\}_{i=1}^N$.

The model aims at predicting the next word. Therefore, it's going to map each sequence w_i to a probability distribution $\hat{y}_i \in \Sigma_V$ over all the possible V words.

Where $\Sigma_n := \{y = (y_1, \ldots, y_n) \in [0,1]^n \quad \text{such that} \quad \sum_{i=1}^{n} y_i = 1\}$.

Figure 2.8 shows an example of a sequence w_i "*The students find the course*", processed with a Recurrent Neural Network in order to predict the next word "*interesting*".

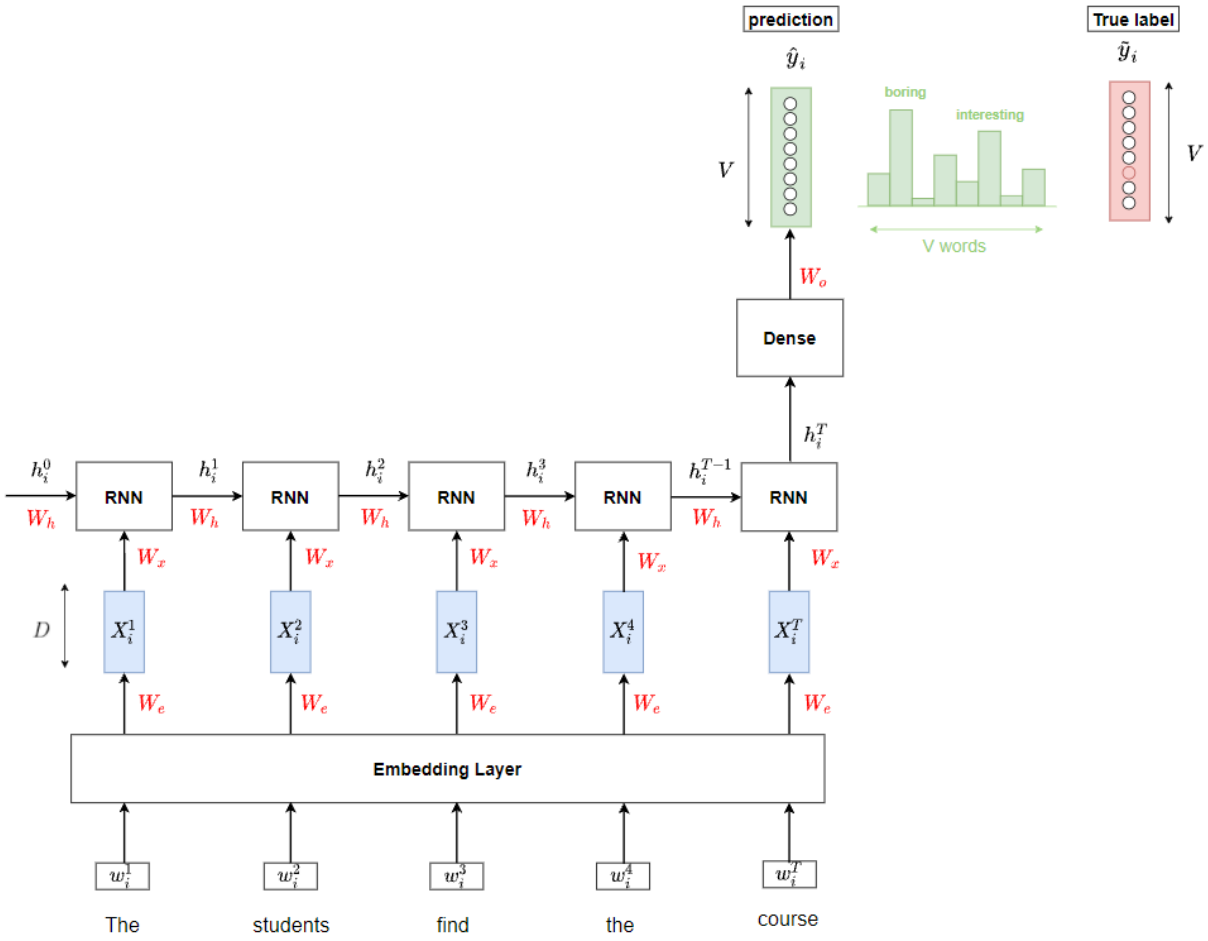


FIGURE 2.8 – RNN for predicting the next word

Below are the details of the prediction process in figure 2.8 :

— First, the input sequence "*The students find the course*" is transformed into a sequence of integers $(w_i^1, \dots, w_i^T)$, via a dictionary that maps each of the V possible words into an index in $[V]$.

— The embedding layer is then used to turn each integer $w_i^t \in [V]$ into a D-dimensional vector representation x_i^t. This is done using an **embedding matrix** W_e of shape (V, D) where each row i represents the embedding of the word of index i. There are different ways of dealing with the embedding matrix :

  — We can consider it as part of the parameters to be trained by backpropagation.

  — We can choose to freeze it by using pretrained word vectors. Such embedding vectors are trained separately on a large corpus of data using a language model.

— The sequence of hidden states $(h_i^1, \dots, h_i^T)$ is then computed sequentially as follows :

$$\forall t \in [T] \quad h_i^t = \sigma\left(W_h h_i^{t-1} + W_x x_i^t\right) \tag{2.2.15}$$

- $x_i^t \in \mathbb{R}^D$ represents the embedding of the new word w_i^t.
- $h_i^{t-1} \in \mathbb{R}^M$ represents the previous hidden state. The size M of the hidden states is a hyper-parameter. We initialize the hidden states with vector of zeros ($h_i^0 = 0_{\mathbb{R}^M}$).
- The RNN layer is parameterized with two matrices : The first weight matrix $W_h \in \mathbb{R}^{M \times M}$ is used to condition the previous hidden state h_i^{t-1}, and the second weight matrix $W_x \in \mathbb{R}^{M \times D}$ is used to condition the new embedding x_i^t.
- σ stands for the non linear activation function. (like the sigmoid, tanh, etc).
- The choice of the activation function is also a hyperparameter.
- It's worth noticing that the **same** matrices W_h and W_x are used throughout the sequential processing.
- The final hidden state $h_i^T \in \mathbb{R}^M$ summarizes the information of the whole sequence in M dimensions.

- The vector h_i^T is then passed into a dense layer, parameterized with a weight matrix $W_o \in \mathbb{R}^{V \times M}$, with a softmax activation function to get the final prediction $\hat{y}_i = (\hat{y}_i^1, \dots, \hat{y}_i^V)$ as follows :

$$\forall v \in [V] \quad \hat{y}_i^v = \frac{e^{[W_o h_i^T]_v}}{\sum_{v'=1}^{V} e^{[W_o h_i^T]_{v'}}}$$

  Where $[W_o h_i^T]_v$ represents the v-th dimension of the V-dimensional vector $W_o h_i^T$
- The final prediction $\hat{y}_i$ is then compared to the true label y_i.
- We usually use **one hot encoding** to represent the discrete random variable Y_i. It consists in encoding Y_i with a random variable $\tilde{Y}_i = (\tilde{Y}_i^1, \dots, \tilde{Y}_i^V)^T$ such that :

$$\forall v \in [V] \quad \tilde{Y}_i^v = 1_{\{Y_i = v\}}$$

We summarize the notations for an element (w_i, y_i) of the training data $\mathcal{T}$ as follows :

Tensor	Space	Definition
w_i^t	$[V]$	t-th word of the input sequence w_i
X_i^t	$\mathbb{R}^D$	t-th embedding vector of word w_i^t
W_h	$\mathbb{R}^{M\times M}$	First weight matrix of the RNN layer
W_x	$\mathbb{R}^{M\times D}$	Second weight matrix of the RNN layer
h_i^t	$\mathbb{R}^M$	t-th hidden state of the RNN layer
W_o	$\mathbb{R}^{V\times M}$	Weight matrix for the dense layer
$\hat{y}_i$	Σ_V	Model prediction
y_i	$[V]$	The true label
$\tilde{y}_i$	$\{0,1\}^V$	The one-hot vector associated with the true label y_i

The whole architecture can be written as follows :

— Let us denote $\theta = (W_e, W_h, W_x, W_o)$ the parameters of the model and f_θ the function that maps each sequence w_i to the prediction $\hat{y}_i$.

— Thus,

$$\hat{y}_i = f_\theta(w_i)$$

— As a result, the distribution of the label Y conditioned on the input sequence W is :

$$Y \mid W = w_i \sim \mathcal{M}\left(1, [f_\theta(w_i)]_1, \ldots, [f_\theta(w_i)]_V\right)$$

Where :

— $\mathcal{M}(1, \pi_1, \ldots, \pi_V)$ stands for the Multinomial distribution, parameterized by $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_V)$

— $[f_\theta(w_i)]_v$ represents the v-th dimension of the V-dimensional vector $\hat{y}_i = f_\theta(w_i)$

The loss function can be derived from the likelihood as follows :

— The likelihood $\mathcal{L}(\theta)$ can then be written as follows :

$$\begin{aligned}
\mathcal{L}(\theta) &= \prod_{i=1}^{N} \mathbb{P}\left(Y = y_i \mid W = w_i\right) \quad \text{(since } (w_i, y_i)_i \text{ are i.i.d)} \\
&= \prod_{i=1}^{N} \mathbb{P}\left(\tilde{Y} = \tilde{y}_i \mid W = w_i\right) \\
&= \prod_{i=1}^{N} \prod_{v=1}^{V} \left[[f_\theta(w_i)]_v\right]^{\tilde{y}_i^v}
\end{aligned}$$

— Instead of maximizing the likelihood, we can minimize the scaled negative log likelihood. Which gives the following loss function :

$$J(\theta) := -\frac{1}{N}\log\left(\mathcal{L}(\theta)\right)$$
$$= -\frac{1}{N}\sum_{i=1}^{N}\sum_{v=1}^{V}\tilde{y}_i^v \log\left([f_\theta(w_i)]_v\right)$$

We can use the exact same framework to predict the next word by replacing the RNN layer with the LSTM layer, as shown in figure 2.9.

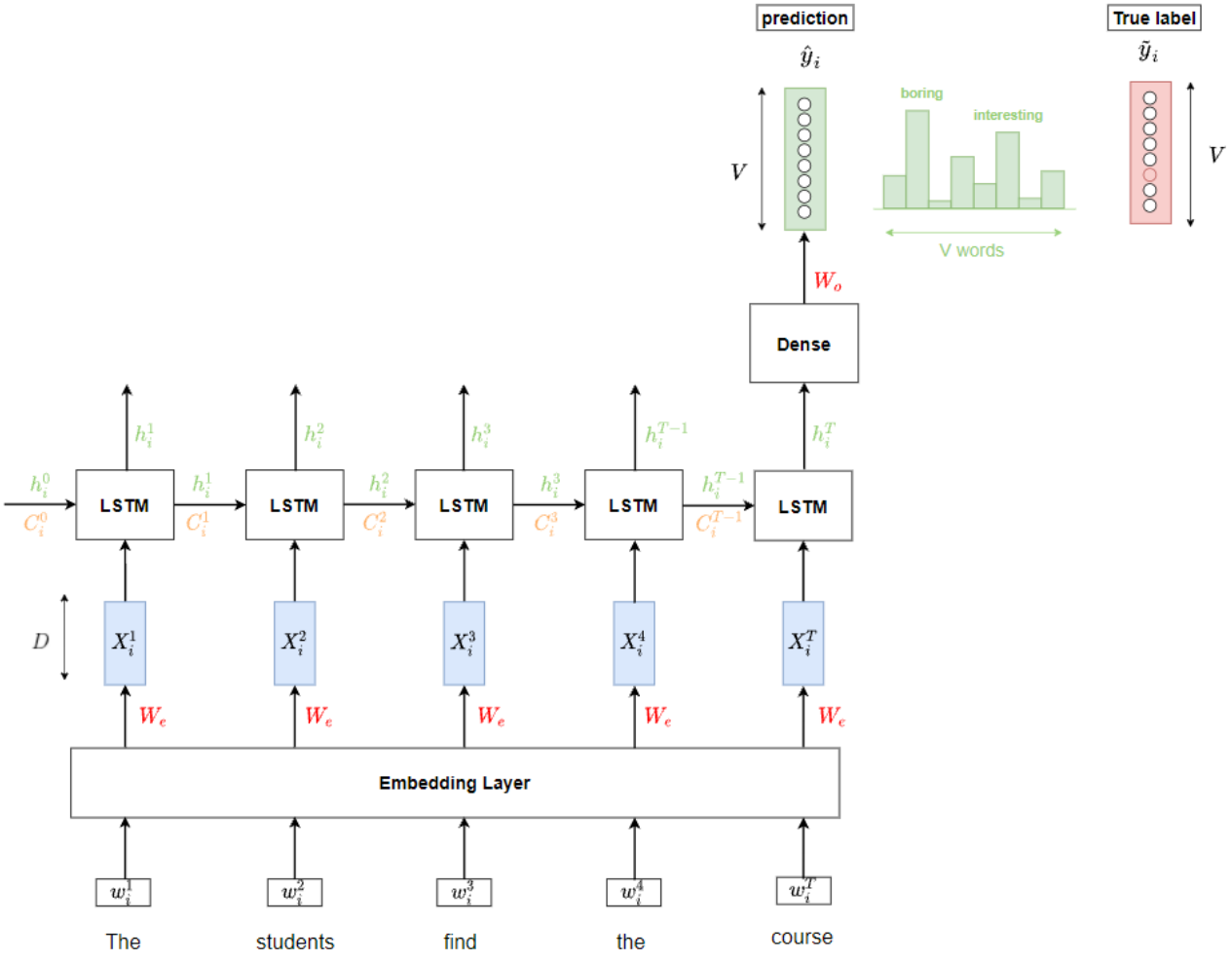


FIGURE 2.9 – Predicting the next word using the LSTM architecture

2.3 The Sequence to Sequence Framework

For Many to Many applications, the LSTM/GRU models can only be applied if the input and the output sequences are of the same length, which can be useful for applications such as POS tagging (P. Wang et al., 2015). However, if we want to learn a mapping Φ_θ from a sequence of input vectors of length T_x into a sequence of output vectors of length T_y (where $T_x \neq T_y$), we need to introduce a new framework, composed of two steps, represented in figure 2.10.

— An encoder layer $\mathcal{E}_{\theta_e}$, parameterized by θ_e learns to map the input sequence $(X_i^1, \ldots, X_i^{T_x}) \in \mathbb{R}^{T_x \times D_x}$ into the sequence of hidden states $h_i^1, \ldots, h_i^{T_x}$.

— The last hidden state $h_i^{T_x}$ is fed into the decoder layer $\mathcal{D}_{\theta_d}$, parameterized by θ_d, as an initial hidden state.

— Let $s_i^0, \ldots, s_i^{T_y}$ be the decoder hidden states. Thus, we have $s_i^0 = h_i^{T_x}$.

— The decoder layer produces the output vectors $s_i^1, \ldots, s_i^{T_y}$.

In the sequence to sequence framework, the encoder and the decoder can be any sequential model such as vanilla RNNs (Kombrink, Mikolov, Karafiát, & Burget, 2011) or LSTMs (Hochreiter & Schmidhuber, 1997).

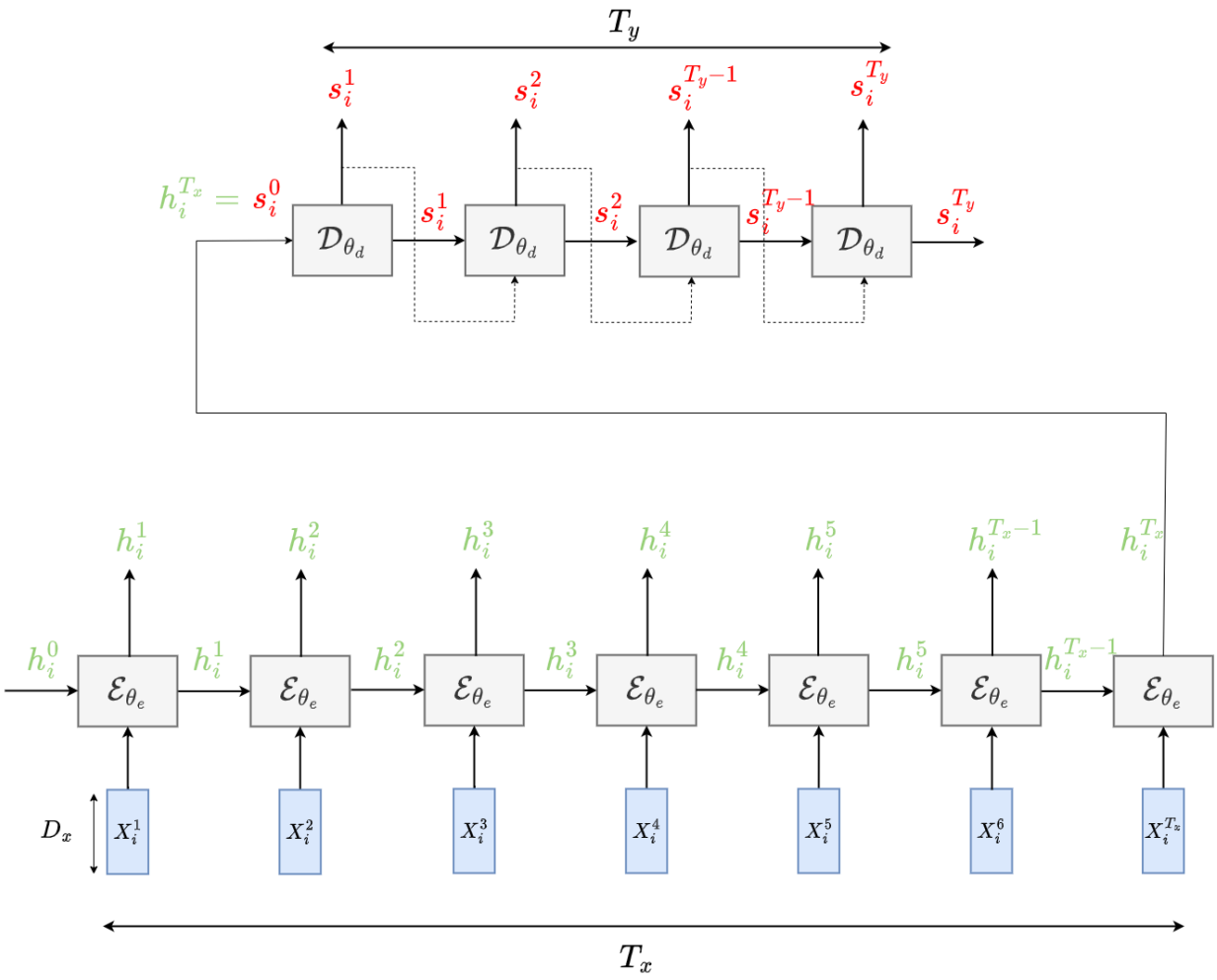


FIGURE 2.10 – The sequence to sequence framework

2.3.1 Applying the Sequence to Sequence Model for Neural Machine Translation

In the previous section, we introduced how we can process a sequence of feature vectors in order to predict a probability distribution in a classification context. The problem we are dealing with is slightly different, in the sense that our objective is to map a sequence of feature vectors into another related sequence. In Natural Language Processing, that would typically be the case for a translation model.

Indeed, the sequence to sequence model was first used for English-French translation (Sutskever et al., 2014).

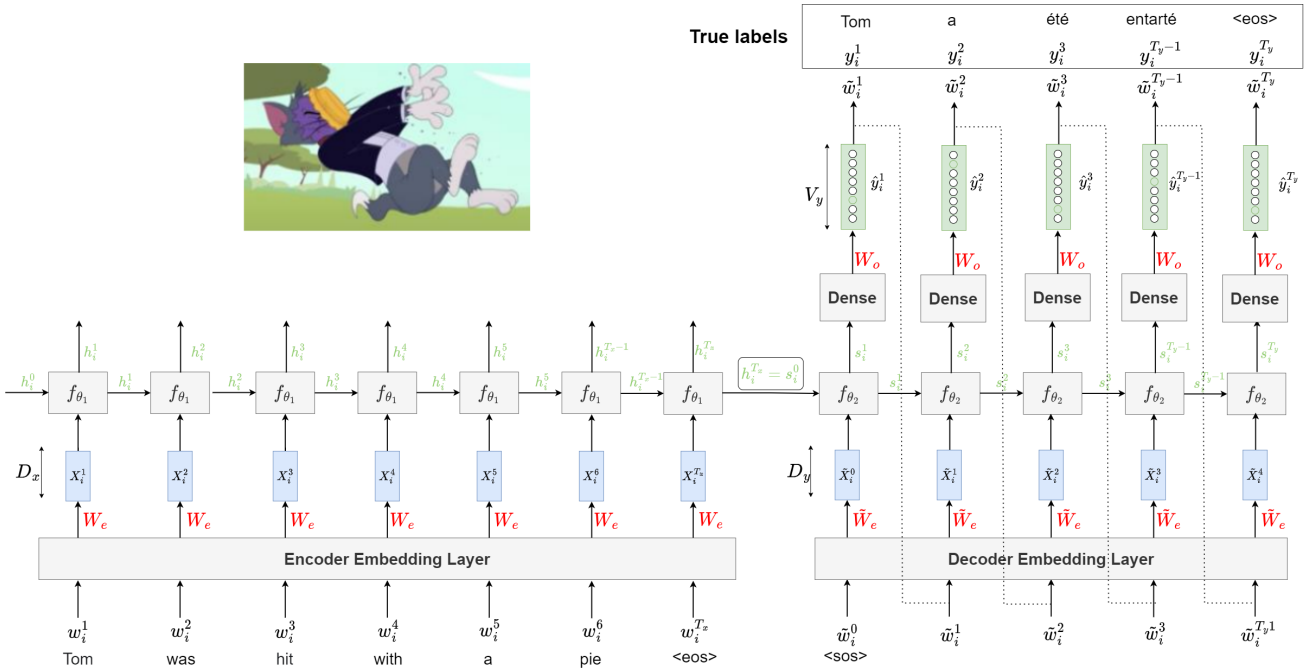


FIGURE 2.11 – The sequence to sequence architecture for machine translation

Figure 2.11 is a representation of such a model.

Basically, the input is a sequence of English words like "Tom was hit with a pie", that we want to translate into the French sentence "Tom a été entarté" using an encoder decoder architecture.

The input data is then a sequence of English words $w_i = (w_i^1, \dots, w_i^{T_x})$ of length T_x, and the target associated with it is a sequence of French words $y_i = (y_i^1, \dots, y_i^{T_y})$ of length T_y.

Such a sequence to sequence model is typically composed of :

— **The encoder** :

The encoder compresses all the information of the input sequence into a fixed length vector as follows :

— As usual, the first step is to map the input words $(w_i^1, \dots, w_i^{T_x})$ into the embedding space of size D_x via the embedding matrix W_e associated with the English vocabulary.

— The embedding vectors $X_i^1, \dots, X_i^{T_x}$ associated with the input words $(w_i^1, \dots, w_i^{T_x})$ are processed using an RNN model f_{θ_1} (the GRU model for instance).

— Let $h_i^t \in \mathbb{R}^M$ represents the hidden state of the encoder at time t :

$$h_i^t = f_{\theta_1}\left(h_i^{t-1}, X_i^t\right)$$

— As the final hidden state $h_i^{T_x}$ encodes all the information in the input sequence, it is going to be used to initialize the hidden states of the decoder.

— **The decoder** :

The decoder is also an RNN which takes the vector $h_i^{T_x}$ and generates an output sequence $(\tilde{w}_i^1, \dots, \tilde{w}_i^{T_y})$.

— Let s_i^t represent the hidden state of the decoder at time t.

— The first hidden state of the decoder s_i^0 is the last hidden state of the encoder $h_i^{T_x}$.

— The first input vector $\tilde{X}_i^0$ is the embedding vector of size D_y associated with the token of index $\tilde{w}_i^0$ "<sos>" (i.e, start of sequence). For that, we use the embedding matrix $\tilde{W}_e$ associated with the French vocabulary.

— Each hidden state s_i^t, at a particular time step $t \in \{1, \dots, T_y\}$, is transformed into a V_y-dimensional discrete distribution $\hat{y}_i^t$ (where V_y is the size of the French vocabulary) via a feed forward neural network parameterized by W_o. Therefore, $\tilde{w}_i^t$ the predicted word at time t is the word whose index corresponds to the highest probability in $\hat{y}_i^t \in V_y$.

— The hidden state s_i^t of the decoder at time $t \in \{2, \dots, T_y\}$ is obtained by processing the previous hidden state s_i^{t-1} and the embedding $\tilde{X}_i^{t-1}$ of the previous predicted word $\tilde{w}_i^{t-1}$

— Let us denote θ the parameters of the model and for all $t \in \{1, \dots, T_y\}$ $\tilde{y}_i^t$ the one hot encoding vector associated with the integer y_i^t.

— Since we have a multiclass classification problem (over V_y possible categories) at each time step $t \in \{1, \dots, T_y\}$, the global loss function $J_i(\theta)$ associated with an input-ouput pair (w_i, y_i) is the average of all the losses $J_i^t(\theta)$ associated with each time step $t \in \{1, \dots, T_y\}$. Consequently :

$$J_i(\theta) = \frac{1}{T_y} \sum_{t=1}^{T_y} \underbrace{\sum_{v=1}^{V_y} \tilde{y}_i^t[v] \log\left(\hat{y}_i^t[v]\right)}_{J_i^t(\theta)} \tag{2.3.16}$$

Where :

— For all $t \in \{1, \dots, T_y\}$ and $v \in \{1, \dots, V_y\}$ $\tilde{y}_i^t[v]$ stands for the v-th dimension of $\tilde{y}_i^t$.

— Similarly, for all $t \in \{1, \dots, T_y\}$ and $v \in \{1, \dots, V_y\}$ $\hat{y}_i^t[v]$ stands for the v-th dimension of $\hat{y}_i^t$.

— **The teacher forcing strategy**

As the decoder predictions are fed back into itself, the model has to predict the whole sequence $\hat{y}_i^1, \dots, \hat{y}_i^{T_y}$ before it compares it to the true sequence $\tilde{y}_i^1, \dots, \tilde{y}_i^{T_y}$ using the loss 2.3.16.

This recursive output-as-input process can result in slow convergence since a bad prediction that occurs at the beginning of the predicted sequence is very likey to affect the rest of the predicted sequence.

An interesting technique that is frequently used in temporal supervised learning tasks (Williams & Zipser, 1989) to overcome this issue is to replace, during the training process, the output vector $\tilde{w}_i^t$ that is fed into the decoder at the next iteration with the true prediction y_i^t as shown in figure 2.12. This technique is called *teacher forcing* since it corrects the predictions at each time step to maximize the chances of producing better next predictions, the same way a teacher would correct the answer of a student one step at a time.

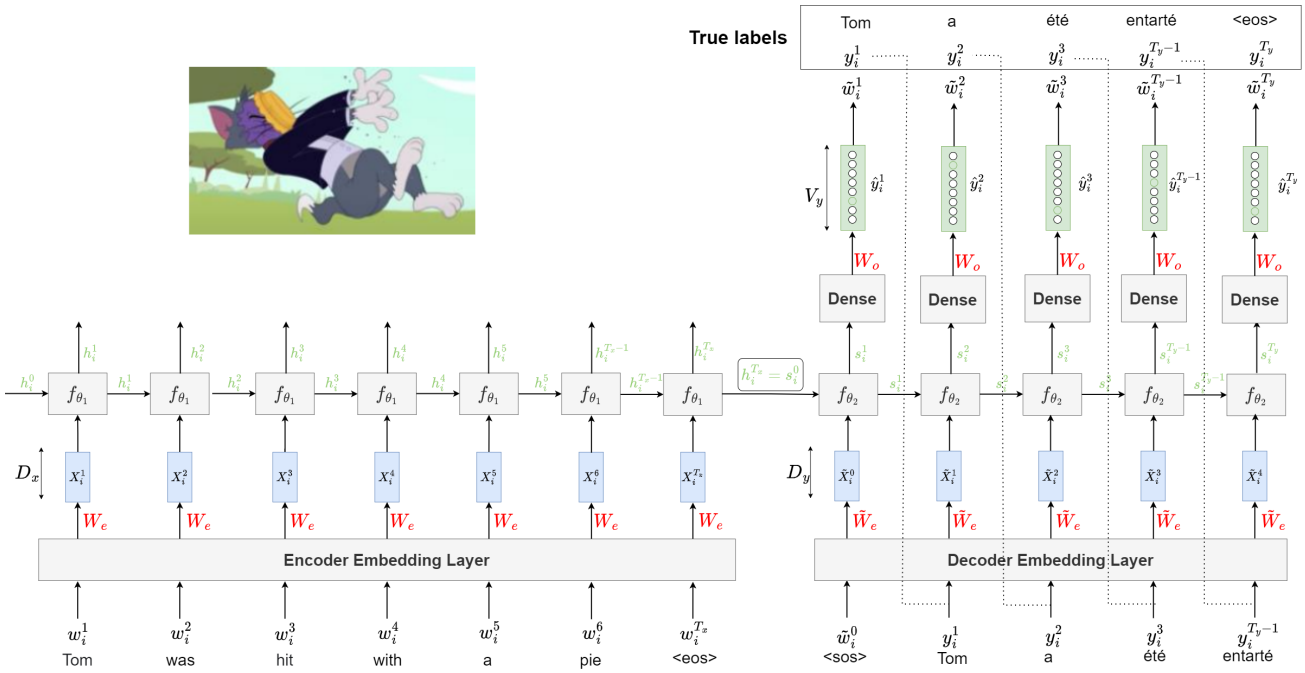


FIGURE 2.12 – The sequence to sequence with teacher forcing architecture for machine translation

2.4 Limitations of classical models

Although Recurrent Neural networks (RNNs) and convolutional neural networks (CNNs) have been sucessfully applied to capture non trivial relationships in complex systems, classic models only perform the task of perception, which consists in learning a mapping between inputs and outputs. They do not carry out sequential reasoning.

Moreover, there are two main challenges with the sequence to sequence framework using RNNs. First, by feeding a single fixed length vector to the decoder, the encoder has to compress all the input information in few dimensions, which leads to a loss of information. Due to this limitation, the performance of the sequence to sequence model degrades rapidly as the length of the input sequence increases.

Second, this architecture doesn't allow **model alignment** between the input and the output sequences. We would like each output sequence to selectively focus on relevant parts of the input sequence.

Let us consider a mapping from a sequence $(X_i^1, \ldots, X_i^{T_x})$ into a sequence $(Y_i^1, \ldots, Y_i^{T_y})$. The intuition of alignment is represented in figure 2.13, it shows how much of each input vector $X_i^{t'}$ should be considered when generating an output vector Y_i^t, for all $(t, t') \in \{1, \ldots, T_y\} \times \{1, \ldots, T_x\}$.

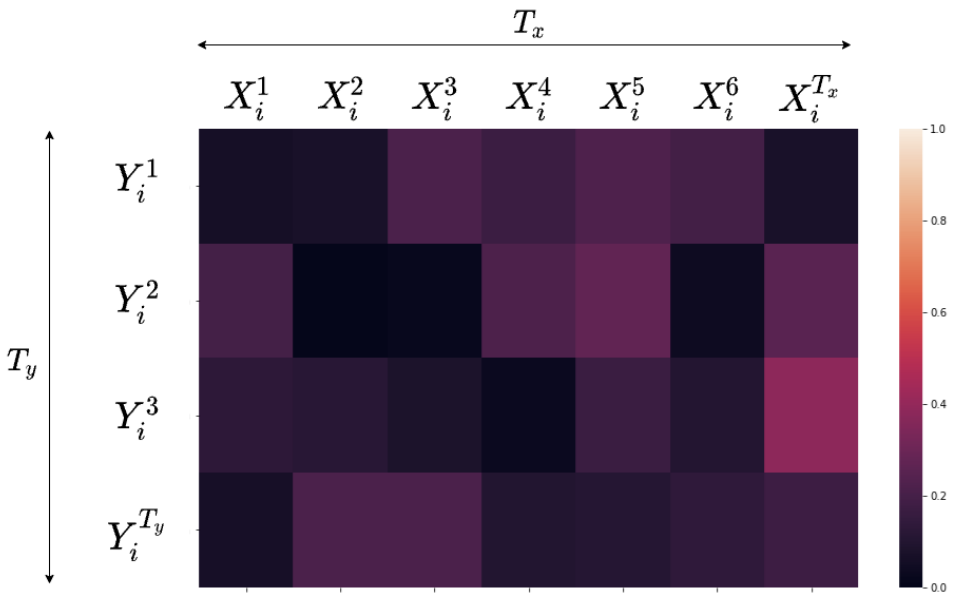


FIGURE 2.13 – Matrix of alignment scores

To sum up, the sequence to sequence framework is not well adapted to modeling long time dependencies and capturing the relevance between the input and the output sequences. Attention mechanisms aim at addressing the aforementioned crucial challenges, especially in applications such as machine translation or time series prediction.

The process of reasoning consists in combining the perception with a selective memory guiding the process of reasoning by focusing on relevant parts of the input or the memory. That's why neuroscientists (Dehaene, 2012), (Deco & Rolls, 2005), (Lindsay, 2020) consider attention as a pillar of learning and reasoning.

2.5 Introducing the Attention Mechanisms in Machine Learning

2.5.1 Query-Retrieval Modeling

Attention mechanisms originate from database Query-Retrieval Problems. Consider the database represented in figure 2.14 where a query is searched through the keys in order to retrieve a value (Garcia-Molina, Ullman, & Widom, 2000).

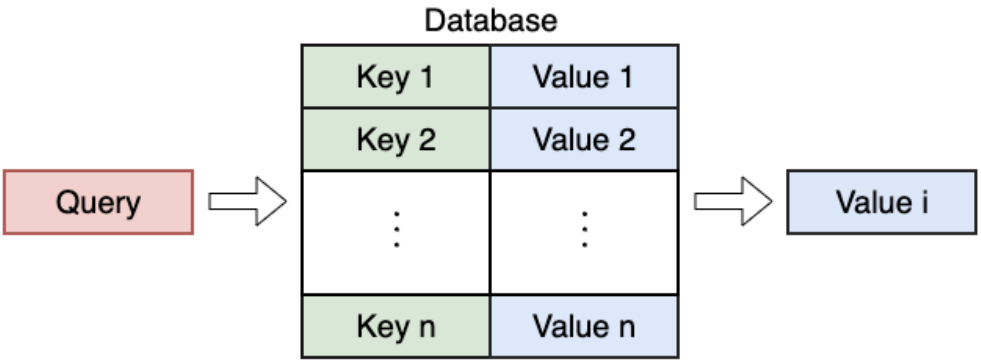


FIGURE 2.14 – Hard Query Retrieval Problem

Attention mechanisms can be viewed as a soft query retrieval process, where multiple keys can correspond to the query, rather than only one. As a result, we need to calculate the similarity between the query and all the keys. The sum of the values, weighted by the computed similarities is the soft-query retrieval vector, or the **attention** vector.

Figure 2.15 represents the different steps involved in calculating the attention vector from a query $q \in \mathbb{R}^{d_q}$, a list of keys $(k_i)_{1 \leq i \leq n} \in \mathbb{R}^{n \times d_k}$ and a list of values $(v_i)_{1 \leq i \leq n} \in \mathbb{R}^{n \times d_v}$

— Using an alignment function a, we calculate the similarity between the query and all the keys as follows :

$$\forall i \in \{1, \ldots, n\} \quad a_i = a(q, k_i)$$

— Several alignment functions have been proposed in the literature in order to get the alignment scores $(a_i)_{1 \leq i \leq n}$, as shown in table 4.1 :

Function	Equation	References
Dot Product	$a(q, k_i) = q^T k_i$	(Luong, Pham, & Manning, s. d.)
Scaled Dot Product	$a(q, k_i) = \frac{q^T k_i}{\sqrt{d_k}}$	(Vaswani et al., 2017)
Luong's Multiplicative al.	$a(q, k_i) = q^T W k_i$	(Luong et al., s. d.)
Bahdanau's Additive al.	$a(q, k_i) = v_a^T \tanh(W_1 q + W_2 k_i)$	(Bahdanau et al., 2017)
Feature-based	$a(q, k_i) = W_{imp}^T \text{act}(W_1 \phi_1(k_i) + W_2 \phi_2(q) + b)$	(Y. Li, Kaiser, Bengio, & Si, 2019)
Kernel Method	$a(q, k_i) = \phi(q)^T \phi(k_i)$	(Yuan et al., 2021)

TABLE 2.1 – Alignment Functions

— The distribution function is used to map the alignment scores $(a_i)_{1 \leq i \leq n}$ to the attention weights $(\alpha_i)_{1 \leq i \leq n}$. The function ensures that $\forall i \in \{1, \ldots, n\} \quad \alpha_i \geq 0$ and $\sum_{i=1}^{n} \alpha_i = 1$.

— We usually use the softmax distribution function in order to get dense alignments as follows :

$$\forall i \in \{1, \ldots, n\} \quad \alpha_i = \frac{e^{a_i}}{\sum_{j=1}^{n} e^{a_j}} \tag{2.5.17}$$

— Sparse alignment [2] can be obtained by using the **sparsemax** (Martins & Astudillo, 2016) or the **sparse entmax** (Martins et al., 2020) distribution functions.

— The attention vector $A\left(q, (k_i)_{1 \leq i \leq n}, (v_i)_{1 \leq i \leq n}\right)$ is then calculated as follows :

$$A\left(q, (k_i)_{1 \leq i \leq n}, (v_i)_{1 \leq i \leq n}\right) = \sum_{i=1}^{n} \alpha_i v_i$$

2. Non zero probabilities are assigned to only a few number of values

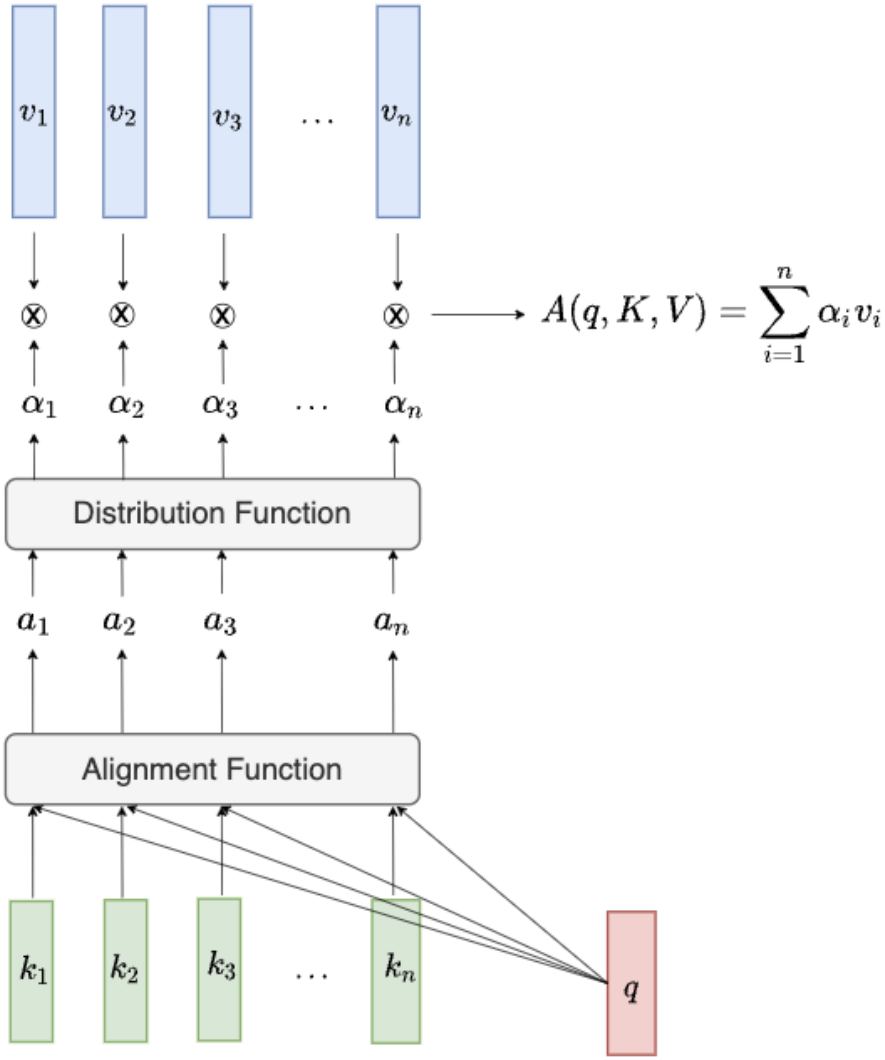


FIGURE 2.15 – Soft-Query Retrieval

2.5.2 Introducing Attention Mechanisms to the Sequence to Sequence framework

The idea of attention mechanisms was first introduced in (Bahdanau et al., 2015) and consists in introducing an attention layer between the encoder and the decoder in order to enable the decoder to decide which part of the encoder outputs are relevant to the generation of the next output.

Formally, let us suppose that our objective is to learn a mapping function Φ_θ from the space of the input sequences $\mathcal{X}$ to the space of the output sequences $\mathcal{Y}$.

Let D_x be the dimensionality of the input vectors and T_x be the length of the input sequences. Let D_y be the dimensionality of the output vectors and T_y be the length of the output sequences.

The training dataset is composed of N input sequences $(X_i^1, \ldots, X_i^{T_x})_{1 \leq i \leq N}$ and N output sequences $(Y_i^1, \ldots, Y_i^{T_y})_{1 \leq i \leq N}$.

We would like to describe the mapping function Φ_θ which transforms an element from the input space $(X_i^1, \ldots, X_i^{T_x}) \in \mathcal{X}$ into an element of the output space $(\hat{Y}_i^1, \ldots, \hat{Y}_i^{T_y}) \in \mathcal{Y}$.

Hence,

$$\mathcal{Y} \ni (\hat{Y}_i^1, \ldots, \hat{Y}_i^{T_y}) = \Phi_\theta(X_i^1, \ldots, X_i^{T_x})$$

The mapping function Φ_θ, described in figure 2.17, can be decomposed into the following transformations :

— An encoder layer $\mathcal{E}_{\theta_e}$ parameterized by θ_e learns to map the input sequence $(X_i^1, \ldots, X_i^{T_x})$ into the sequence of encoder hidden states $h_i^1, \ldots, h_i^{T_x}$.

— A decoder layer $\mathcal{D}_{\theta_d}$ parameterized by θ_d learns to map the encoder hidden states $h_i^1, \ldots, h_i^{T_x}$ to the output sequence $\hat{Y}_i^1, \ldots, \hat{Y}_i^{T_y}$.

— Let $s_i^1, \ldots, s_i^{T_y}$ be the decoder hidden states. An attention layer $\mathcal{A}_{\theta_a}$ parameterized by θ_a learns to assign attention weights $\alpha_i^{<t_y,1>}, \ldots, \alpha_i^{<t_y,T_x>}$ to the encoder hidden states $h_i^1, \ldots, h_i^{T_x}$ when generating the decoder hidden state $s_i^{t_y}$ for all $t_y \in \{1, \ldots, T_y\}$.

— Each decoder hidden state $s_i^{t_y}$ is then mapped into the prediction vector $\hat{Y}_i^{t_y}$ for all $t_y \in \{1, \ldots, T_y\}$ using the final layer $\mathcal{F}_{\theta_f}$.

1. **The Encoder :**

   The encoder $\mathcal{E}_{\theta_e}$ is a GRU model 2.4, parameterized by the following parameters : $U_r \in \mathbb{R}^{M \times M}, V_r \in \mathbb{R}^{D \times M}, U_h \in \mathbb{R}^{M \times M}, V_h \in \mathbb{R}^{D \times M}, U_z \in \mathbb{R}^{M \times M}, V_z \in \mathbb{R}^{D \times M}$

   The GRU maps the input sequence $(X_i^1, \ldots, X_i^{T_x})$ into the sequence $(h_i^1, \ldots, h_i^{T_x})$.

   Let us consider $t_x \in \{1, \ldots, T_x\}$. Equations 2.5.18, 2.5.19 , 2.5.20 , 2.5.21 describe how the encoder $\mathcal{E}_{\theta_e}$ generates the hidden state $h_i^{t_x}$ from $h_i^{t_x-1}$ and $X_i^{t_x}$:

$$r_i^{t_x} = \sigma\left(U_r h_i^{t_x-1} + V_r X_i^{t_x}\right) \quad \text{(Generating the reset gate vector)} \tag{2.5.18}$$

$$\tilde{h}_i^{t_x} = \tanh\left(r_i^{t_x} \circ U_h h_i^{t_x-1} + V_h X_i^{t_x}\right) \quad \text{(Generating the new memory candidate)} \tag{2.5.19}$$

$$z_i^{t_x} = \sigma\left(U_z h_i^{t_x-1} + V_z X_i^{t_x}\right) \quad \text{(Generating the update gate vector)} \tag{2.5.20}$$

$$h_i^{t_x} = (1 - z_i^{t_x}) \circ \tilde{h}_i^{t_x} + z_i^{t_x} \circ h_i^{t_x-1} \quad \text{(Generating the new hidden state)} \tag{2.5.21}$$

2. **The Decoder :**

Let us now consider $t_y \in \{1, \dots, T_y\}$. We would like to generate the decoder hidden state $s_i^{t_y}$ from the previous hidden state $s_i^{t_y-1}$ and a context vector $c_i^{t_y}$ resulting from an attention mechanism applied on the encoder hidden states $h_i^1, \dots, h_i^{T_x}$.

3. **The Attention Layer :**

   The attention layer $\mathcal{A}_{\theta_a}$, represented in figure 2.16, assigns a weight to each encoder hidden state in order to output the final **context vector** c_i^t as follows :

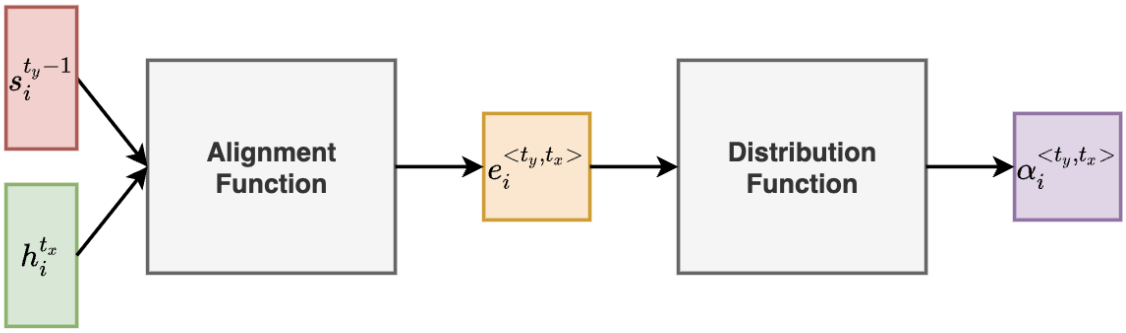


FIGURE 2.16 – The attention layer

— A Feed Forward neural network is applied to calculate an alignment score between the decoder previous hidden state $s_i^{t_y-1}$ and each encoder hidden state $h_i^{t_x} \in \{h_i^1, \dots, h_i^{T_x}\}$. There are multiple alignment functions available, including the additive alignment function of Bahdanau (Bahdanau et al., 2017), as well as the multiplicative alignment function of Luong (Luong et al., s. d.).

$$e_i^{<t_y,t_x>} = \begin{cases} v_a^T \tanh\left(W_1 s_i^{t_y-1} + W_2 h_i^{t_x}\right) & \text{Bahdanau's additive alignment function} \\ s_i^{t_y-1^T} W h_i^{t_x} & \text{Luong's multiplicative alignment function} \end{cases}$$

— The resulting alignment score $e_i^{<t_y,t_x>}$ can then be turned into the attention weight $\alpha_i^{<t_y,t_x>}$ using a distribution function (such as the softmax, the sparsemax (Martins & Astudillo, 2016) or the sparse entmax (Martins et al., 2020) distribution functions).

— The attention vector (also called the context vector) is then calculated as follows :

$$c_i^{t_y} = \sum_{t_x=1}^{T_x} \alpha_i^{<t_y,t_x>} h_i^{t_x}$$

— As a result, the context vector can be seen as the attention vector associated with the query

$q = s_i^{t_y-1}$, the keys $(k_t)_{1\leq t\leq T_x} = (h_i^{t_x})_{1\leq t_x\leq T_x}$ and the values $(v_t)_{1\leq t\leq T_x} = (h_i^{t_x})_{1\leq t_x\leq T_x}$.

$$A\left(q, (k_i)_{1\leq i\leq n}, (v_i)_{1\leq i\leq n}\right) = \sum_{t_x=1}^{T_x} \alpha_i^{<t_y,t_x>} h_i^{t_x} = c_i^{t_y}$$

4. **The Final Layer :**

   The final layer $\mathcal{F}_{\theta_f}$ is then applied on each decoder hidden state $s_i^{t_y}$ in order to generate the prediction $\hat{Y}_i^{t_y}$. The nature of the final layer depends on the kind of application we are dealing with. For instance, in a sentiment analysis problem, the final layer is a basic Dense layer parameterized by $\theta_f = (W_f, b_f)$ with a softmax activation function :

$$\hat{Y}_i^{t_y} = \text{softmax}\left(W_f^T s_i^{t_y} + b_f\right)$$

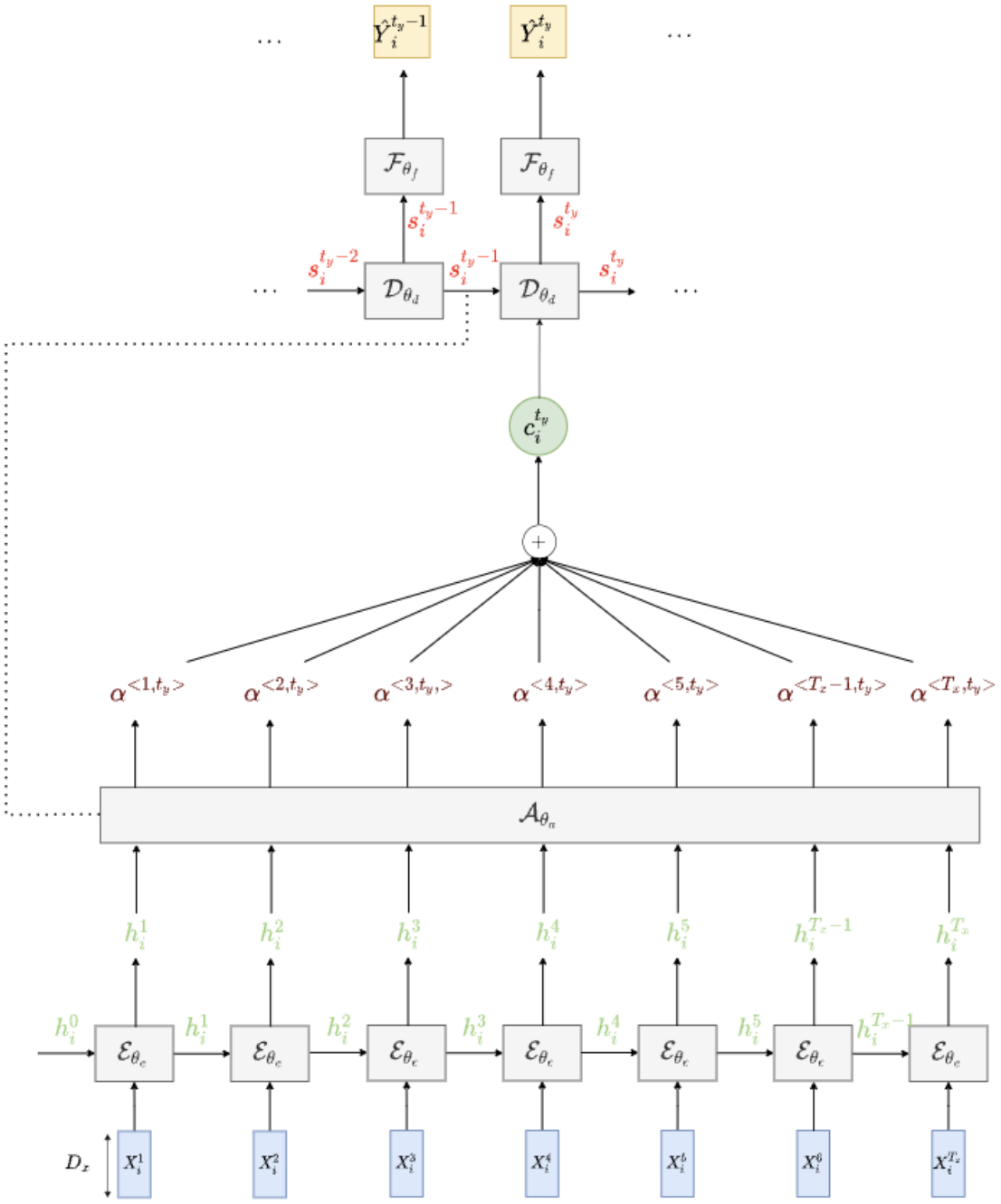


FIGURE 2.17 – Sequence to sequence model with attention

2.5.3 Applying the Sequence to Sequence Model for Neural Machine Translation

By replacing the sequence to sequence architecture represented in 2.11 with the sequence to sequence with attention mechanisms framework defined in 2.18, we obtain the following architecture for Neural Machine Translation.

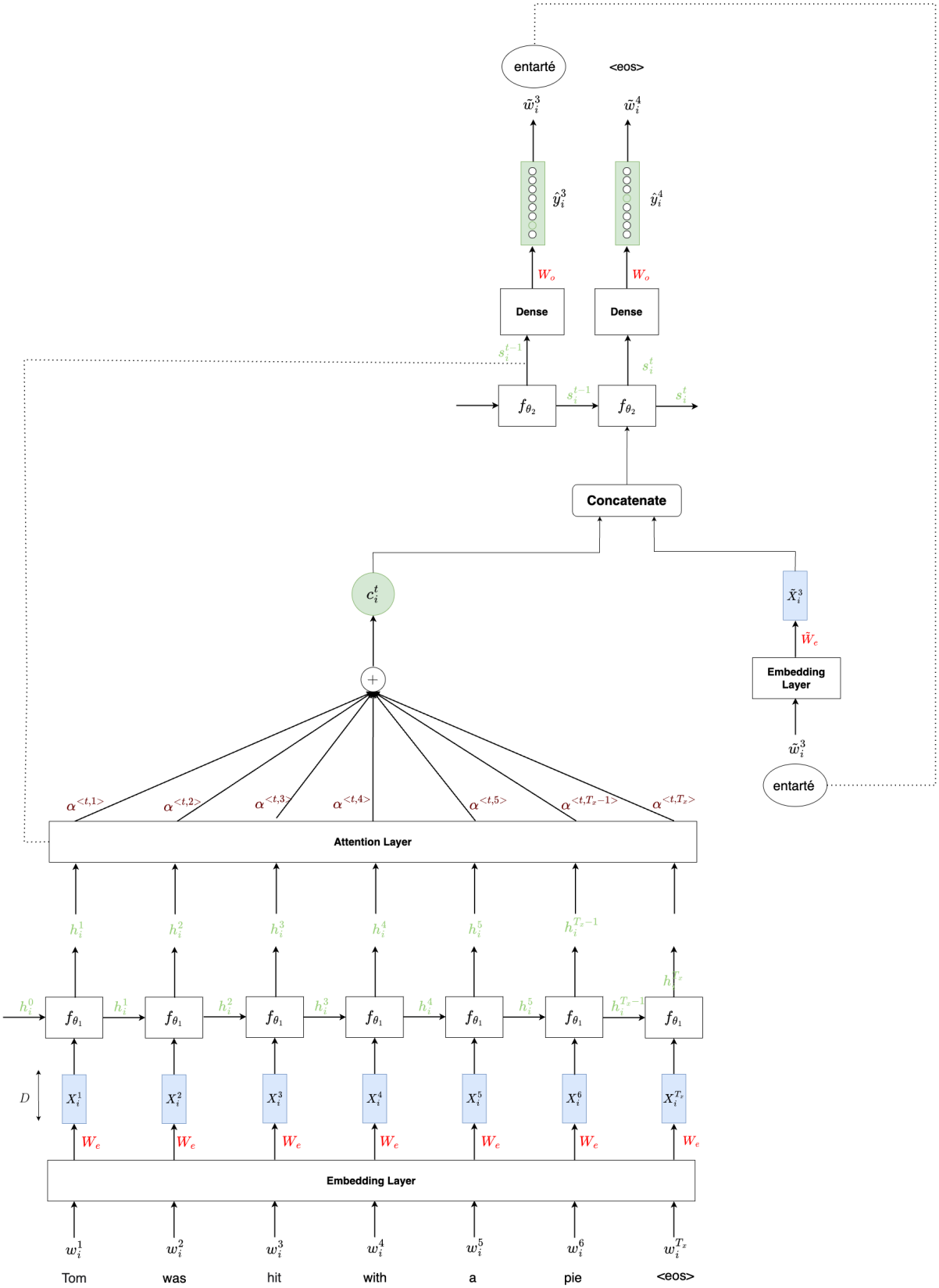


FIGURE 2.18 – The sequence to sequence architecture with attention mechanisms

2.6 The Transformer architecture

2.6.1 Introduction

"Attention is All You Need" (Vaswani et al., 2017) stands out among the most important and interesting papers of the recent years. It presented several improvements to the soft attention algorithm and made it possible to perform the sequence to sequence modeling without having to use recurrent network units.

Therefore, the **Transformer** model proposed in (Vaswani et al., 2017) and represented in the figure 2.19 is entirely based on self-attention mechanisms without using sequence-aligned recurrent architectures.

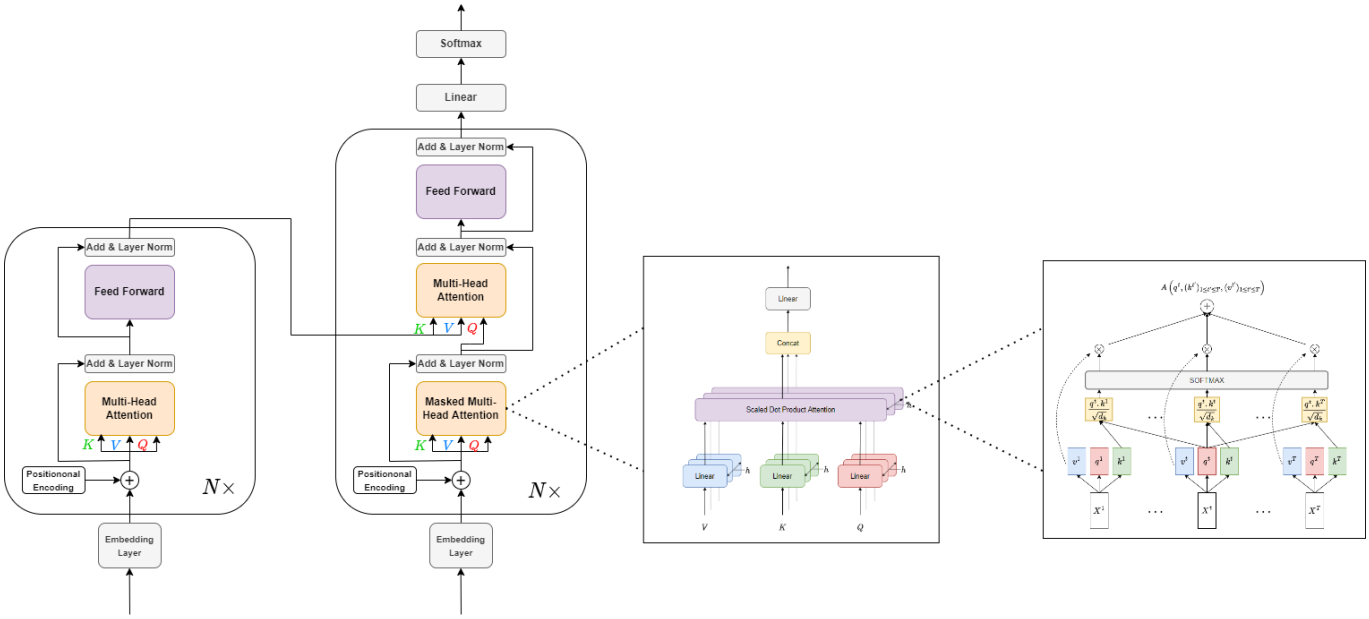


FIGURE 2.19 – The Transformer Architecture

2.6.2 Creating a contextual embedding with Self Attention

Let us consider a sequence of D-dimensional input vectors $(X^t)_{1\leq t\leq T}$. In order to use the attention mechanism, we define the projections of the embeddings X^t onto the d_q-dimensional query space, d_k-dimensional key space and d_v-dimensional value space :

$$\mathbb{R}^{d_q} \ni q^t = W_Q^T X^t$$
$$\mathbb{R}^{d_k} \ni k^t = W_K^T X^t$$
$$\mathbb{R}^{d_v} \ni v^t = W_V^T X^t$$

Where $W_Q \in \mathbb{R}^{D\times d_q}$, $W_K \in \mathbb{R}^{D\times d_k}$ and $W_V \in \mathbb{R}^{D\times d_v}$ are the projection matrices onto the low dimensional query, key and value spaces, respectively. We also need $d_q = d_k$.

Let us consider a query $q^t \in \{q^1, \ldots, q^T\}$. The Self Attention transformation, represented in figure 2.20 consists in creating a contextual embedding $A\left(q^t, (k^{t'})_{1\leq t'\leq T}, (v^{t'})_{1\leq t'\leq T}\right)$ associated with the embedding vector X^t.

To that end, the scaled dot product alignment function (Vaswani et al., 2017) is used to calculate the similarity $e^{<t,t'>}$ between the query q^t and the keys $(k^{t'})_{1\leq t'\leq T}$ as follows :

$$e^{<t,t'>} = \frac{q^t.k^{t'}}{\sqrt{d_k}} \tag{2.6.22}$$

A Softmax distribution function is then used to turn the similarity scores $e^{<t,t'>}$ into attention weights $\alpha^{<t,t'>}$ representing the contribution of the embedding $X^{t'}$ in the process of generating the contextual embedding $A\left(q^t, (k^{t'})_{1\leq t'\leq T}, (v^{t'})_{1\leq t'\leq T}\right)$.

$$\alpha^{<t,t'>} = \frac{e^{<t,t'>}}{\sum_{s=1}^{T} e^{<t,s>}} \tag{2.6.23}$$

The contextual embedding $A\left(q^t, (k^{t'})_{1\leq t'\leq T}, (v^{t'})_{1\leq t'\leq T}\right)$ can then be computed as follows :

$$A\left(q^t, (k^{t'})_{1\leq t'\leq T}, (v^{t'})_{1\leq t'\leq T}\right) = \sum_{t'=1}^{T} \alpha^{<t,t'>} v^{t'} \tag{2.6.24}$$

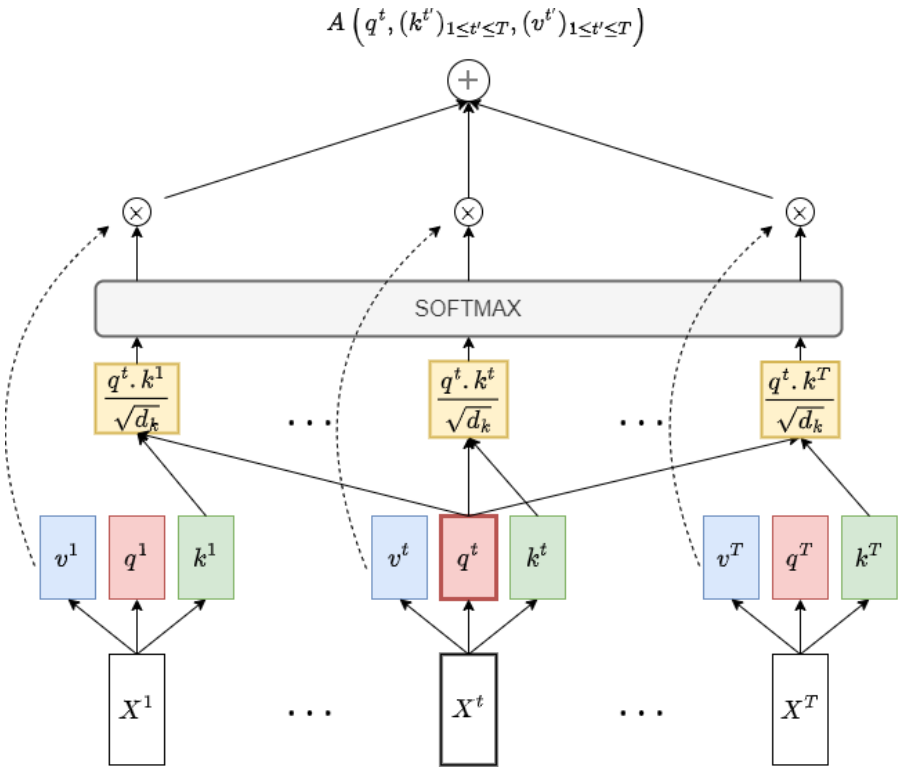


FIGURE 2.20 – Creating the contextual embedding with Self Attention

2.6.3 The Matrix of contextual embeddings

We can generalize the way to create the contextual embedding $A\left(q^t,(k^{t'})_{1\leq t'\leq T},(v^{t'})_{1\leq t'\leq T}\right)$ associated with the embedding X^t to all the embedding vectors $(X^{t'})_{1\leq t'\leq T}$.

We consider the following matrix notations :

$$Q=\begin{bmatrix}— & q^1 & — \\ \vdots & \vdots & \vdots \\ — & q^T & —\end{bmatrix}\in\mathbb{R}^{T\times d_q},\quad K=\begin{bmatrix}— & k^1 & — \\ \vdots & \vdots & \vdots \\ — & k^T & —\end{bmatrix}\in\mathbb{R}^{T\times d_k},\quad V=\begin{bmatrix}— & v^1 & — \\ \vdots & \vdots & \vdots \\ — & v^T & —\end{bmatrix}\in\mathbb{R}^{T\times d_v}$$

We define the scaled dot product attention matrix, denoted $A(Q,K,V)$, as follows :

$$A(Q,K,V):=\text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Where the notation Softmax(M) for a matrix $M\in\mathbb{R}^{T\times d}$ refers to the Softmax applied to each row of the matrix M.

We have :

$$\begin{aligned}
\text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V &= \text{Softmax}\left(\left[\frac{q^t.k^{t'}}{\sqrt{d_k}}\right]_{\substack{1\leq t\leq T\\1\leq t'\leq T}}\right)V \\
&= \text{Softmax}\left(\left[e^{<t,t'>}\right]_{\substack{1\leq t\leq T\\1\leq t'\leq T}}\right)V \\
&= \left[\alpha^{<t,t'>}\right]_{\substack{1\leq t\leq T\\1\leq t'\leq T}}\begin{bmatrix} — & v^1 & — \\ \vdots & \vdots & \vdots \\ — & v^T & — \end{bmatrix} \\
&= \begin{bmatrix} — & \sum_{t'=1}^{T}\alpha^{<1,t'>}v^{t'} & — \\ \vdots & \vdots & \vdots \\ — & \sum_{t'=1}^{T}\alpha^{<t,t'>}v^{t'} & — \\ \vdots & \vdots & \vdots \\ — & \sum_{t'=1}^{T}\alpha^{<T,t'>}v^{t'} & — \end{bmatrix} \\
&= \begin{bmatrix} — & A\left(q^1,(k^{t'})_{1\leq t'\leq T},(v^{t'})_{1\leq t'\leq T}\right) & — \\ \vdots & \vdots & \vdots \\ — & A\left(q^t,(k^{t'})_{1\leq t'\leq T},(v^{t'})_{1\leq t'\leq T}\right) & — \\ \vdots & \vdots & \vdots \\ — & A\left(q^T,(k^{t'})_{1\leq t'\leq T},(v^{t'})_{1\leq t'\leq T}\right) & — \end{bmatrix}
\end{aligned}$$

Therefore :

$$A(Q,K,V) = \begin{bmatrix} — & A\left(q^1,(k^{t'})_{1\leq t'\leq T},(v^{t'})_{1\leq t'\leq T}\right) & — \\ \vdots & \vdots & \vdots \\ — & A\left(q^t,(k^{t'})_{1\leq t'\leq T},(v^{t'})_{1\leq t'\leq T}\right) & — \\ \vdots & \vdots & \vdots \\ — & A\left(q^T,(k^{t'})_{1\leq t'\leq T},(v^{t'})_{1\leq t'\leq T}\right) & — \end{bmatrix}$$

In other words, the t-th row of the scaled dot product attention matrix $A(Q,K,V)$ is the contextual embedding vectors $A\left(q^t,(k^{t'})_{1\leq t'\leq T},(v^{t'})_{1\leq t'\leq T}\right)$ associated with the embedding vector X^t.

2.6.4 MultiHead Attention with the Scaled Dot Product Attention

The scaled dot product attention can be generalized to any query $Q \in \mathbb{R}^{T_q \times d_q}$, key $K \in \mathbb{R}^{T_k \times d_k}$ and value $V \in \mathbb{R}^{T_v \times d_v}$ matrices such that $d_q = d_k$ and $T_k = T_v = T'$.

$$Q = \begin{bmatrix} — & q^1 & — \\ \vdots & \vdots & \vdots \\ — & q^{T_q} & — \end{bmatrix} \in \mathbb{R}^{T_q \times d_q}, \quad K = \begin{bmatrix} — & k^1 & — \\ \vdots & \vdots & \vdots \\ — & k^{T_k} & — \end{bmatrix} \in \mathbb{R}^{T_k \times d_k}, \quad V = \begin{bmatrix} — & v^1 & — \\ \vdots & \vdots & \vdots \\ — & v^{T_v} & — \end{bmatrix} \in \mathbb{R}^{T_v \times d_v}$$

The scaled dot product attention matrix is then defined as follows :

$$A(Q, K, V) = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V = \begin{bmatrix} — & A\left(q^1, (k^{t'})_{1 \leq t' \leq T'}, (v^{t'})_{1 \leq t' \leq T'}\right) & — \\ \vdots & \vdots & \vdots \\ — & A\left(q^t, (k^{t'})_{1 \leq t' \leq T'}, (v^{t'})_{1 \leq t' \leq T'}\right) & — \\ \vdots & \vdots & \vdots \\ — & A\left(q^{T_q}, (k^{t'})_{1 \leq t' \leq T'}, (v^{t'})_{1 \leq t' \leq T'}\right) & — \end{bmatrix} \in \mathbb{R}^{T_q \times d_v}$$

The MultiHead Attention module, represented in figure 2.21, consists in applying the attention mechanism defined in the previous section h times in order to capture different notions of similarity.

Hence, for each head $h' \in \{1 \ldots, h\}$, let $W_Q^{h'} \in \mathbb{R}^{d_q \times p_q}$, $W_K^{h'} \in \mathbb{R}^{d_k \times p_k}$ and $W_V^{h'} \in \mathbb{R}^{d_v \times p_v}$ be the h'-th projection matrices of Q, K and V onto the low dimensional key, query and value spaces of size p_q, p_k and p_v, respectively. (We also have $p_q = p_k$).

The h'-th attention head is then defined as $A\left(QW_Q^{h'}, KW_K^{h'}, VW_V^{h'}\right)$.

The concatenation layer aims at stacking all the attention heads together to make a new flatten vector, which is then projected onto $\mathbb{R}^{p_o}$ using the projection matrix $W_o \in \mathbb{R}^{hp_v \times p_o}$. The output of the MultiHead Attention layer P is then computed as follows :

$$P = \text{concat}\left(A\left(QW_Q^1, KW_K^1, VW_V^1\right), \ldots, A\left(QW_Q^h, KW_K^h, VW_V^h\right)\right) W_o \in \mathbb{R}^{T_q \times p_o}$$

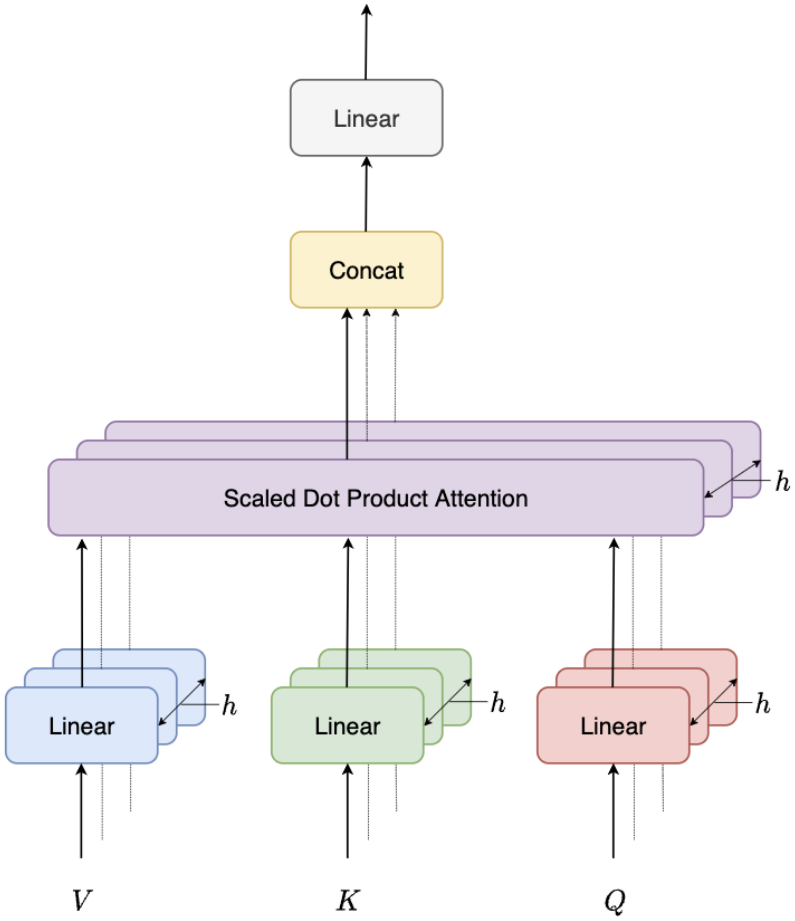


FIGURE 2.21 – MultiHead Attention with h heads

2.6.5 Positional encoding

The attention mechanism is permutation invariant. In order to account for the order of the embedding vectors $X^1, \ldots, X^T \in \mathbb{R}^D$, we use positional encoding vectors.

The positional encoding vectors $p^1, \ldots, p^T \in \mathbb{R}^D$ have the same dimension as the input embeddings $X^1, \ldots, X^T$.

— For each time step $t \in \{1, \ldots, T\}$, there is a unique positional encoding vector.

— The distance between two steps should be consistent across sentences with different lengths.

The method used in the original paper (Vaswani et al., 2017) is a linear transformation $T^{(k)} \in \mathbb{R}^{D \times D}$ such that

$$T^{(k)} p^t = p^{t+k}$$

It is defined as follows : For $d \in \{1, \dots, \frac{D}{2}\}$:

$$w_i = \frac{1}{100000^{\frac{2i}{D}}} \quad \text{and} \quad p^t = \begin{pmatrix} \sin(w_1 t) \\ \cos(w_1 t) \\ \vdots \\ \sin(w_d t) \\ \cos(w_d t) \\ \vdots \\ \sin(w_{\frac{D}{2}} t) \\ \cos(w_{\frac{D}{2}} t) \end{pmatrix}$$

We define for a $d \in \{1, \dots, \frac{D}{2}\}$ and for all $t \in \{1 \dots, T\}$:

$$e_d^t = \begin{pmatrix} \sin w_d t \\ \cos w_d t \end{pmatrix}$$

Therefore,

$$p^t = \begin{pmatrix} e_1^t \\ \vdots \\ e_d^t \\ \vdots \\ e_{\frac{D}{2}}^t \end{pmatrix}$$

By defining :

$$\mathcal{T}^{(k)} = \begin{pmatrix} \Phi_1^{(k)} & & & & \\ & \ddots & & & \\ & & \Phi_d^{(k)} & & \\ & & & \ddots & \\ & & & & \Phi_{\frac{D}{2}}^{(k)} \end{pmatrix} \quad \text{where} \quad \Phi_d^{(k)} = \begin{pmatrix} \cos(w_d k) & \sin(w_d k) \\ -\sin(w_d k) & \cos(w_d k) \end{pmatrix}$$

We get :

$$\mathcal{T}^{(k)}p^t = \begin{pmatrix} \Phi_1^{(k)} & & & & & \\ & \ddots & & & & \\ & & \Phi_d^{(k)} & & & \\ & & & \ddots & & \\ & & & & \Phi_{\frac{D}{2}}^{(k)} \end{pmatrix} \begin{pmatrix} e_1^t \\ \vdots \\ e_d^t \\ \vdots \\ e_{\frac{D}{2}}^t \end{pmatrix}$$

$$= \begin{pmatrix} \Phi_1^{(k)} e_1^t \\ \vdots \\ \Phi_d^{(k)} e_d^t \\ \vdots \\ \Phi_{\frac{D}{2}}^{(k)} e_{\frac{D}{2}}^t \end{pmatrix}$$

We have for all d in $\{1, \ldots, \frac{D}{2}\}$:

$$\begin{aligned} \Phi_d^{(k)} e_d^t &= \begin{pmatrix} \cos(w_d k) & \sin(w_d k) \\ -\sin(w_d k) & \cos(w_d k) \end{pmatrix} \begin{pmatrix} \sin w_d t \\ \cos w_d t \end{pmatrix} \\ &= \begin{pmatrix} \cos(w_d k)\sin w_d t + \sin(w_d k)\cos w_d t \\ -\sin(w_d k)\sin w_d t + \cos(w_d k)\cos w_d t \end{pmatrix} \\ &= \begin{pmatrix} \sin\left(w_d(t+k)\right) \\ \cos\left(w_d(t+k)\right) \end{pmatrix} \\ &= e_d^{t+k} \end{aligned}$$

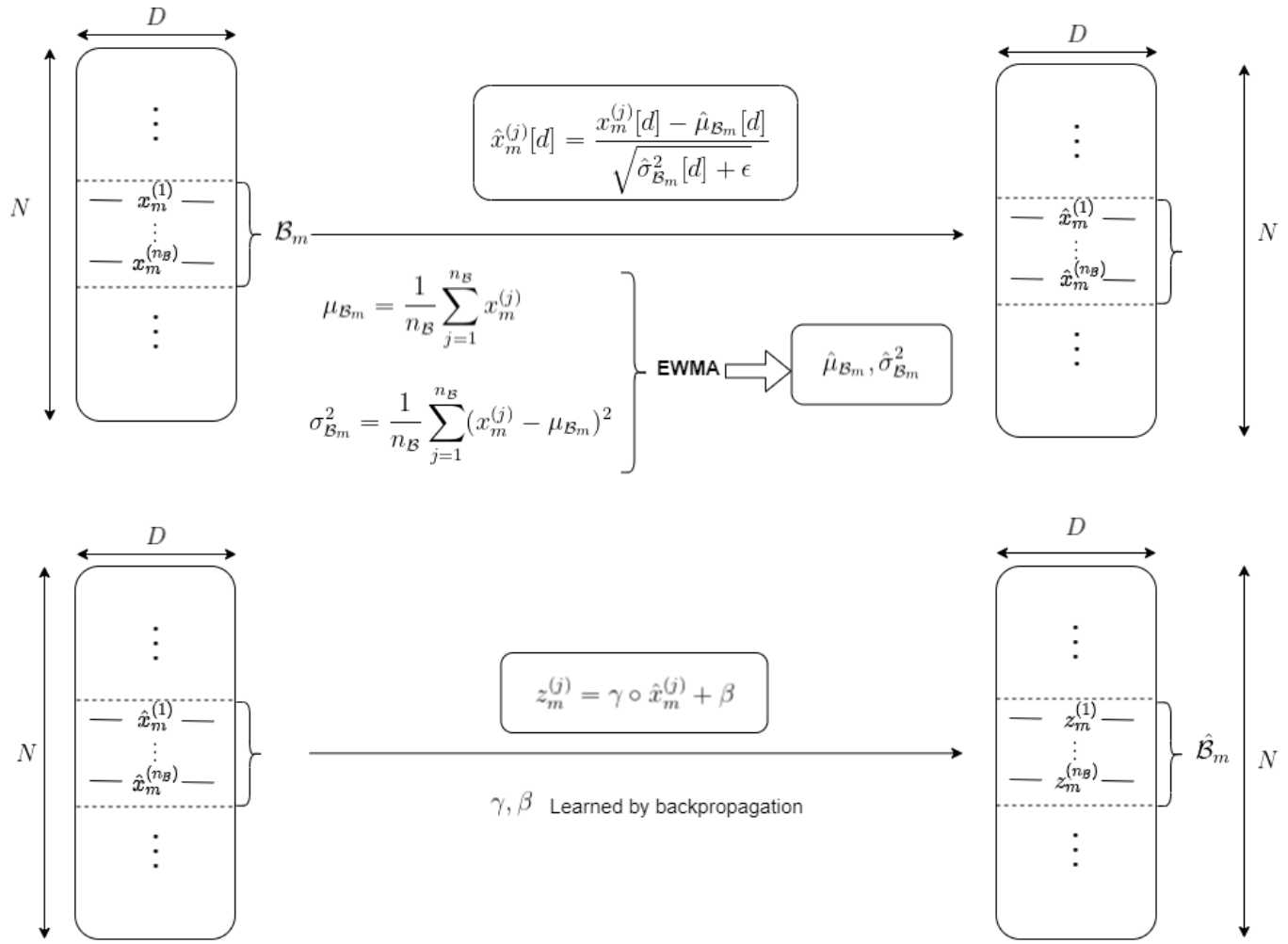
Therefore,

$$\mathcal{T}^{(k)}p^t = \begin{pmatrix} \Phi_1^{(k)} e_1^t \\ \vdots \\ \Phi_d^{(k)} e_d^t \\ \vdots \\ \Phi_{\frac{D}{2}}^{(k)} e_{\frac{D}{2}}^t \end{pmatrix} = \begin{pmatrix} e_1^{t+k} \\ \vdots \\ e_d^{t+k} \\ \vdots \\ e_{\frac{D}{2}}^{t+k} \end{pmatrix} = p^{t+k}$$

2.6.6 The Normalization Process

The Batch Normalization

As explained in (Ioffe & Szegedy, 2015), the Batch Normalization process described in Algorithm 1 considerably helps overcome the vanishing gradients problem by acting like a regularizer.


D
N
$x_m^{(1)}$
$x_m^{(n_\mathcal{B})}$
$\mathcal{B}_m$
$\hat{x}_m^{(j)}[d] = \frac{x_m^{(j)}[d] - \hat{\mu}_{\mathcal{B}_m}[d]}{\sqrt{\hat{\sigma}^2_{\mathcal{B}_m}[d] + \epsilon}}$
$\mu_{\mathcal{B}_m} = \frac{1}{n_\mathcal{B}} \sum_{j=1}^{n_\mathcal{B}} x_m^{(j)}$
$\sigma^2_{\mathcal{B}_m} = \frac{1}{n_\mathcal{B}} \sum_{j=1}^{n_\mathcal{B}} (x_m^{(j)} - \mu_{\mathcal{B}_m})^2$
EWMA
$\hat{\mu}_{\mathcal{B}_m}, \hat{\sigma}^2_{\mathcal{B}_m}$
$\hat{x}_m^{(1)}$
$\hat{x}_m^{(n_\mathcal{B})}$
$z_m^{(j)} = \gamma \circ \hat{x}_m^{(j)} + \beta$
γ, β Learned by backpropagation
$z_m^{(1)}$
$z_m^{(n_\mathcal{B})}$
$\hat{\mathcal{B}}_m$

Algorithm 1 The Batch Normalization

Input:

Batches $\mathcal{B}_1, \ldots, \mathcal{B}_M$ of size $n_{\mathcal{B}} : \forall m \in \{1, \ldots, M\} \quad \mathcal{B}_m = \{x_m^{(1)}, \ldots, x_m^{(n_{\mathcal{B}})}\}$

Output: Batches $\hat{\mathcal{B}}_1, \ldots, \hat{\mathcal{B}}_M$ of size $n_{\mathcal{B}} : \forall m \in \{1, \ldots, M\} \quad \mathcal{B}_m = \{z_m^{(1)}, \ldots, z_m^{(n_{\mathcal{B}})}\}$

1: Set $m_0 = 0, v_0 = 0$

2: **for** $m = 1$ **to** M **do**

3: Calculate the mini-batch mean $\mu_{\mathcal{B}_m}$ and the mini-batch variance $\sigma^2_{\mathcal{B}_m}$ as follows :

$$\mu_{\mathcal{B}_m} = \frac{1}{n_{\mathcal{B}}} \sum_{j=1}^{n_{\mathcal{B}}} x_m^{(j)} \tag{2.6.25}$$

$$\sigma^2_{\mathcal{B}_m} = \frac{1}{n_{\mathcal{B}}} \sum_{j=1}^{n_{\mathcal{B}}} (x_m^{(j)} - \mu_{\mathcal{B}_m}) \circ (x_m^{(j)} - \mu_{\mathcal{B}_m}) \tag{2.6.26}$$

4: Update the mini-batch mean and the mini-batch using EWMA to get $\hat{\mu}_{\mathcal{B}_m}$ and $\hat{\sigma}^2_{\mathcal{B}_m}$ as follows :

$$\hat{\mu}_{\mathcal{B}_m} = \lambda_\mu \circ \hat{\mu}_{\mathcal{B}_{m-1}} + (1 - \lambda_\mu) \circ \mu_{\mathcal{B}_m} \tag{2.6.27}$$

$$\hat{\sigma}^2_{\mathcal{B}_m} = \lambda_\sigma \circ \hat{\sigma}^2_{\mathcal{B}_{m-1}} + (1 - \lambda_\sigma) \circ \sigma^2_{\mathcal{B}_m} \tag{2.6.28}$$

5: Normalize the batch m :

$$\hat{x}_m^{(j)}[d] = \frac{x_m^{(j)}[d] - \hat{\mu}_{\mathcal{B}_m}[d]}{\sqrt{\hat{\sigma}^2_{\mathcal{B}_m}[d] + \epsilon}} \quad (\text{for all } d \in \{1, \ldots, D\} \text{ for all } j \in \{1, \ldots, n_{\mathcal{B}}\}) \tag{2.6.29}$$

with $\epsilon \approx 10^{-5}$ is just a smoothing parameter to avoid dividing by zero.

6: Scale and shift the previous output using two parameters γ and β estimated during the training

$$z_m^{(j)} = \gamma \circ \hat{x}_m^{(j)} + \beta \quad (\text{for all } j \in \{1, \ldots, n_{\mathcal{B}}\}) \tag{2.6.30}$$

Algorithm 1 ensure the distribution of the activations within a layer has zero mean and unit variance across a minibatch.

The Batch Normalization also makes the optimization significantly smoother, as explained in (Santurkar, Tsipras, Ilyas, & Madry, 2018).

The Layer Normalization

The batch normalization process recenters and rescales across the examples within a minibatch. As a result, the batch normalization method can suffer from bad estimates of the mean and the variance parameters when the batch size is too small.

Geoffrey Hinton and his team proposed **Layer Normalization** (Ba, Kiros, & Hinton, 2016). The new algorithm overcomes the cons of Batch Normalization by normalizing the activations across the feature dimension instead of mini-batch directions, which makes is easier to apply for RNNs as well.

2.6.7 The Final Architecture

The Transformer architecture is a sequence to sequence architecture, which relies mainly on the attention mechanism.

The main layers used in the model are the following :

— The Multi-Head Attention layer introduced in 2.6.4.

— The pointwise Feed Forward layer.

— The Normalization Layer 2.6.6

Let us introduce the different steps of both the encoder and the decoder in the transformer.

Consider a sequence $(w_i^1, \ldots, w_i^{T_x})$ of indices representing the sequence of words processed using the word2idx dictionary.

The Encoder Layer

The Encoder, represented in figure 2.23, generates an attention based representation able to focus on a particular piece of information from a large context. It is composed of a stack of $N = 6$ identical layers. Each layer is composed of the following sub-layers :

— The first sub-layer is a Multi-Head self Attention 2.6.4, with a residual connection and a normalization layer .

— The Multi-Head attention can be seen as a way of re-averaging the value vectors in order to create contextual embeddings without introducing non-linearities. The second sub-layer is a fully connected feed-forward layer, with a residual connection and a normalization layer.

In the original paper (Vaswani et al., 2017), all the layers output data of the same dimension $d_{\text{model}} = 512$.

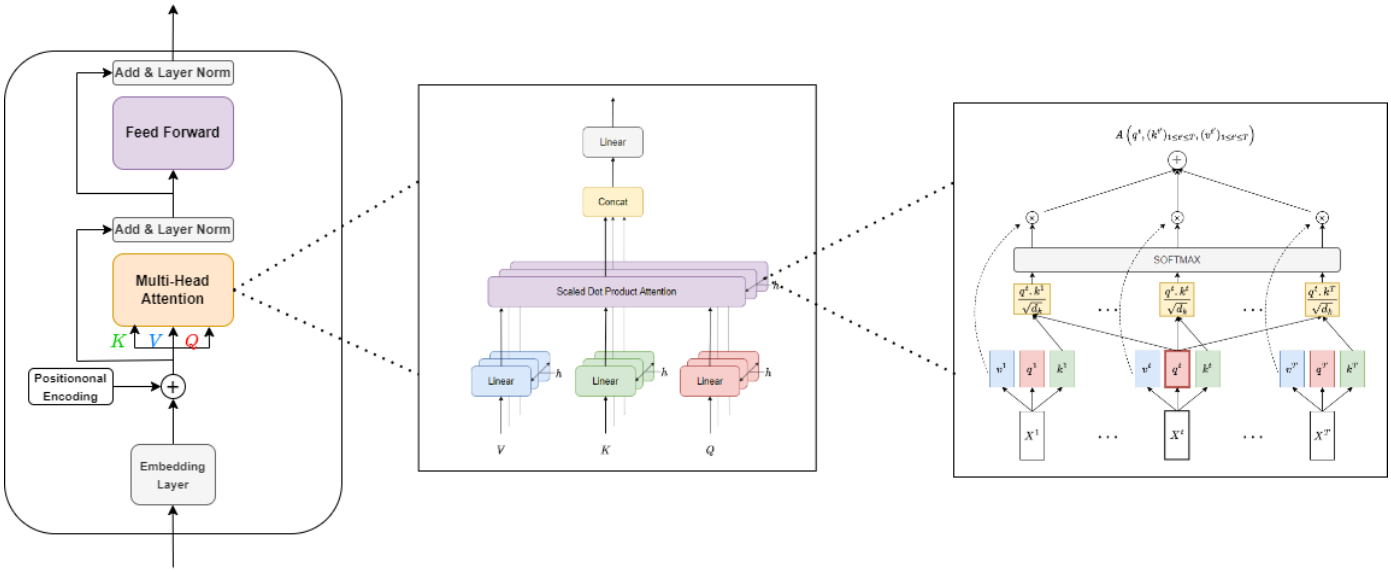


FIGURE 2.23 – The Encoder layer in the Transformer

The Decoder Layer

The decoder, represented in figure 2.24 is responsible of retrieving the information from the encoded representation through the matrices K and V.

It is composed of a stack of $N = 6$ identical layers. Each layer can be divided into the following steps :

— Similarly to the encoder, we combine the multi-head attention layers with a fully connected feed forward layer.

— The first multi-head attention layer is slightly modified to the one introduced in 2.6.4. In order to avoid look-ahead bias we introduce the notion of mask. Consider the following decoder queries, keys and values :

$$Q = \begin{bmatrix} — & q^1 & — \\ \vdots & \vdots & \vdots \\ — & q^T & — \end{bmatrix} \in \mathbb{R}^{T\times d}, \quad K = \begin{bmatrix} — & k^1 & — \\ \vdots & \vdots & \vdots \\ — & k^T & — \end{bmatrix} \in \mathbb{R}^{T\times d}, \quad V = \begin{bmatrix} — & v^1 & — \\ \vdots & \vdots & \vdots \\ — & v^T & — \end{bmatrix} \in \mathbb{R}^{T\times d}$$

For $t \in \{1, \dots, T\}$, we don't want the contextual embedding associated with the query q^t, defined in 2.6.24, to depend on the pairs $(k_{t'}, v_{t'})_{t\leq t'\leq T}$

Hence, we "mask" the contributions $\alpha^{<t,t'>}$ for all $t' > t$ by setting them to zero.

So, we set $e^{<t,t'>}$, defined in 2.6.22, to $-\infty$ for all $t' > t$ as shown in the following table :

	1	2	…	t’	…	T-1	T
1	$e^{<1,1>}$	-∞	…	-∞	…	-∞	-∞
2	$e^{<2,1>}$	$e^{<2,2>}$	…	-∞	…	-∞	-∞
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
t	$e^{<t,1>}$	$e^{<t,2>}$	…	$e^{<t,t'>}$	…	-∞	-∞
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
T-1	$e^{<T-1,1>}$	$e^{<T-1,2>}$	…	$e^{<T-1,t'>}$	…	$e^{<T-1,T-1>}$	-∞
T	$e^{<T,1>}$	$e^{<T,2>}$	…	$e^{<T,t'>}$	…	$e^{<T,T-1>}$	$e^{<T,T>}$

After applying the softmax 2.6.23, we get the masked attention weights :

	1	2	…	t’	…	T-1	T
1	$\alpha^{<1,1>}$	0	…	0	…	0	0
2	$\alpha^{<2,1>}$	$\alpha^{<2,2>}$	…	0	…	0	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
t	$\alpha^{<t,1>}$	$\alpha^{<t,2>}$	…	$\alpha^{<t,t'>}$	…	0	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
T-1	$\alpha^{<T-1,1>}$	$\alpha^{<T-1,2>}$	…	$\alpha^{<T-1,t'>}$	…	$\alpha^{<T-1,T-1>}$	0
T	$\alpha^{<T,1>}$	$\alpha^{<T,2>}$	…	$\alpha^{<T,t'>}$	…	$\alpha^{<T,T-1>}$	$\alpha^{<T,T>}$

— Additionally, we add a residual connection and a normalization layer.

— Another multi-head attention layer, where the queries are the output of the first multi-head attention and the keys and values are the encoder outputs.

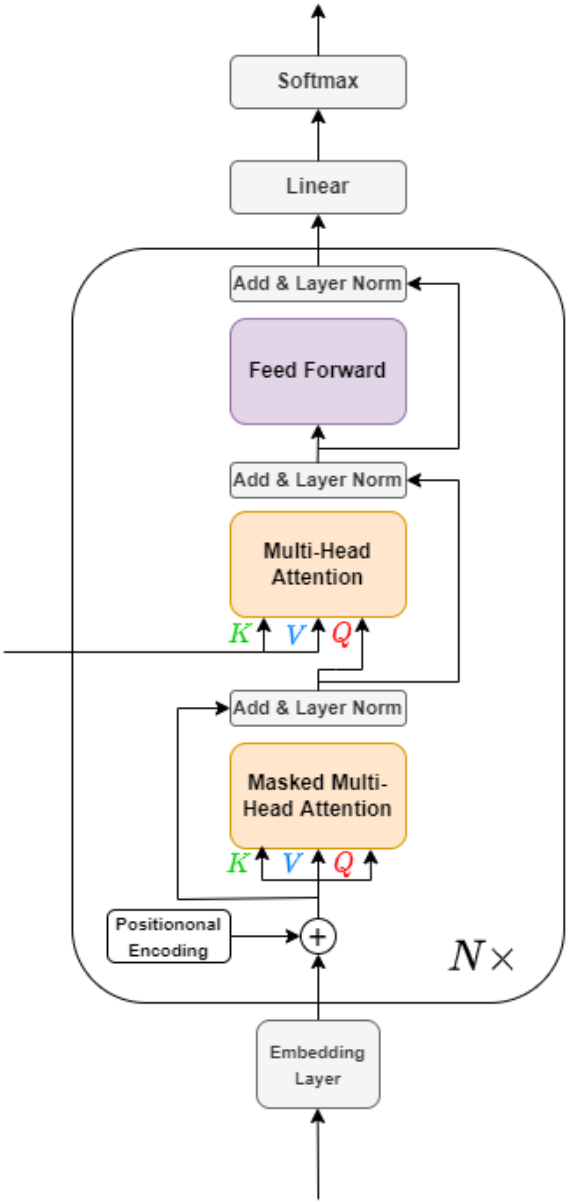


FIGURE 2.24 – The Decoder layer in the Transformer

The Transformer architecture

Finally, by putting it all together, we get the final transformer architecture, represented in 2.25.

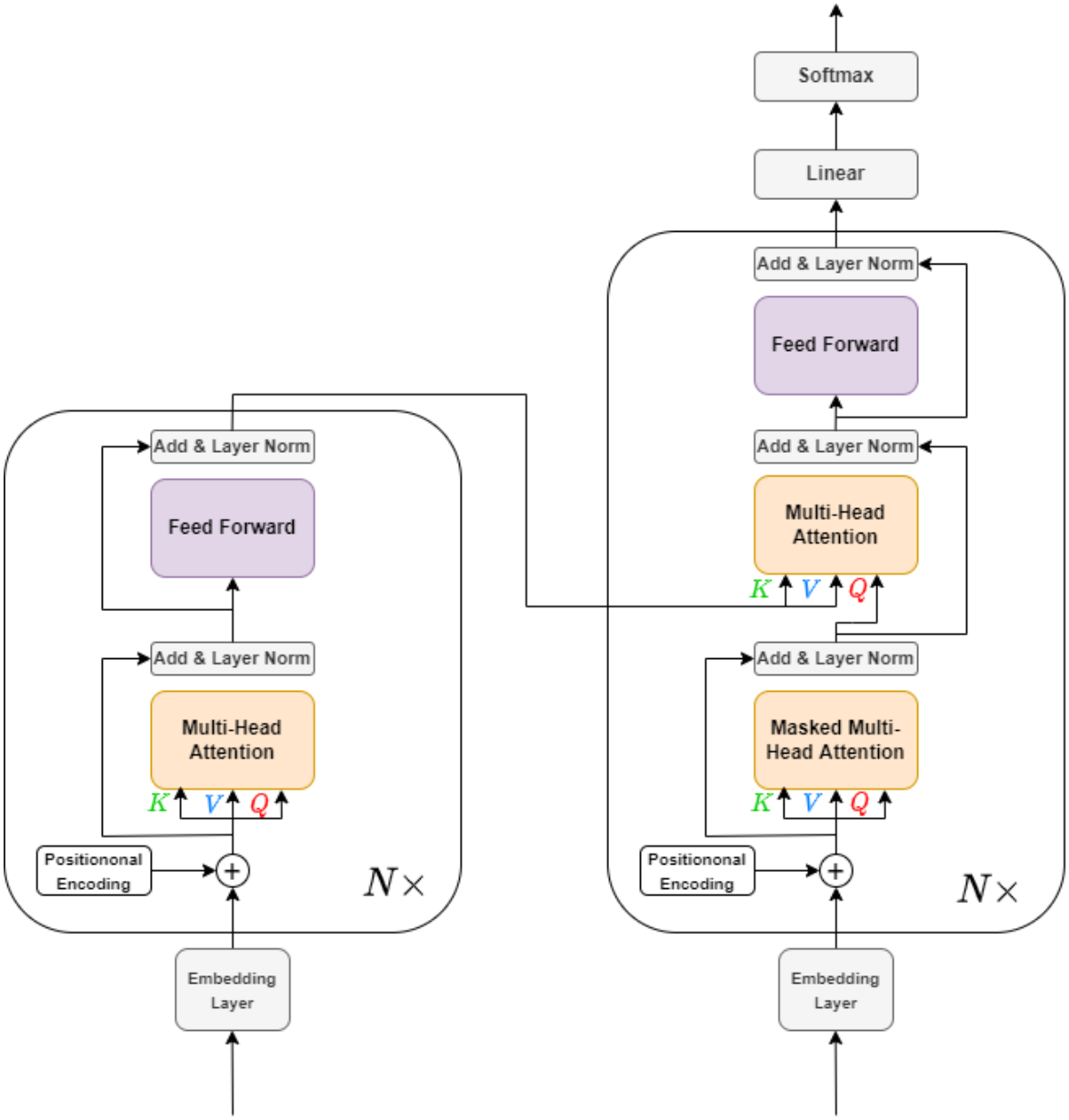


FIGURE 2.25 – The Transformer architecture

2.7 The Optimization algorithm

2.7.1 Position of the problem

In this section, we study the optimization problem, which can be written as follows :

$$\min_{\theta \in \mathbb{R}^d} f(\theta) \quad \text{where} \quad f(\theta) := \mathbb{E}_{s \sim \mathbb{P}}[\mathcal{L}(\theta, s)], \tag{2.7.31}$$

where :

— The function f is called **the objective function**.

— The function $\mathcal{L}$ is the loss function.

— $\mathbb{P}$ is the unknown data distribution on the domain $\mathcal{S}$

— θ is the set of parameters we wish to optimize.

In the following sections, we are going to focus on the Adam optimizer. Section 2.7.2 is a brief introduction to the history of the Adam algorithm. Section 2.7.3 is a description of the algorithm. In section 2.7.4, we analyze the convergence behavior of the algorithm in the nonconvex setting.

2.7.2 Brief history of the Adam optimizer

The Adam algorithm was first introduced in 2015 (Kingma & Ba, 2015). The authors proposed a proof of convergence which was found to have problems. In 2018, (S. J. Reddi et al., 2018) clarified the inconsistency of the previous paper and fixed the proof in the convex setting. In (S. Reddi et al., 2018), the authors conducted the proof for the non convex case under some useful parameter settings.

In the following sections, we provide a detailed version of the proof provided in the original paper (S. Reddi et al., 2018).

2.7.3 Description of the algorithm

The Adam algorithm defined in (S. Reddi et al., 2018) is summarized in Algorithm 2

Algorithm 2 The Adam Optimizer

Input:
Initial parameter value : $\theta_1 \in \mathbb{R}^d$
Learning rates : $\{\eta_t\}_{t=1}^T$
Decay parameters : $0 \leq \beta_1, \beta_2 \leq 1$
Stability parameter : $\epsilon > 0$
Output: ϵ-First Order Stationary Point θ_{T+1}

Set $\boldsymbol{m}_0 = 0, \boldsymbol{v}_0 = 0$
for $t = 1$ **to** T **do**
  Draw a batch $(s_t^i)_{i \in \mathcal{B}_t}$ from $\mathbb{P}$
  Compute $\boldsymbol{g}_t = \frac{1}{|\mathcal{B}_t|} \sum_{s \in \mathcal{B}_t} \nabla \mathcal{L}(\theta_t, s)$
  Update $\boldsymbol{m}_t = \beta_1 \boldsymbol{m}_{t-1} + (1 - \beta_1)\, \boldsymbol{g}_t$
  Update $\boldsymbol{v}_t = \boldsymbol{v}_{t-1} - (1 - \beta_2)\, (\boldsymbol{v}_{t-1} - \boldsymbol{g}_t \circ \boldsymbol{g}_t)$
  Update $\theta_{t+1} = \theta_t - \eta_t \frac{\boldsymbol{m}_t}{\sqrt{\boldsymbol{v}_t + \epsilon}}$

2.7.4 The convergence behavior of the Adam optimizer

Preliminaries

We are fetching for First Order Stationary Points. We would like to prove that under some assumptions on the loss function (not necessarily convex), we can have :

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\|\nabla f\left(\boldsymbol{\theta}_t\right)\|_2^2\right] \leq h(T) \quad \text{with} \quad \lim_{T\to+\infty} h(T) = 0$$

Where T is the number of batches

Assumptions

— $(\mathcal{A}_1)$: We assume the loss function $\mathcal{L}$ to be L-smooth, which means that :

$$\forall \theta_1, \theta_2 \in \mathbb{R}^d \; \forall s \in \mathcal{S} \quad \|\nabla\mathcal{L}(\theta_2, s) - \nabla\mathcal{L}(\theta_1, s)\|_2 \leq L\|\theta_2 - \theta_1\|_2 \tag{2.7.32}$$

— $(\mathcal{A}_2)$: We assume the loss function $\mathcal{L}$ to have bounded gradient : i.e,

$$\exists G \in \mathbb{R}_+ \; \forall \theta \in \mathbb{R}^d \; \forall s \in \mathcal{S} \quad \|\nabla\mathcal{L}(\theta, s)\|_2 \leq G \tag{2.7.33}$$

— $(\mathcal{A}_3)$: We assume the variance of the loss function $\mathcal{L}$ to be bounded : i.e,

$$\forall \theta \in \mathbb{R}^d \quad \mathbb{E}\left[\|\nabla\mathcal{L}(\boldsymbol{\theta}; \xi) - \nabla\mathcal{L}(\boldsymbol{\theta})\|_2^2 | \mathcal{F}_t\right] \leq \sigma^2 \tag{2.7.34}$$

where the sigma-algebra $\mathcal{F}_t$ represents *the information known at time t*

The convergence theorem

Theorem 2.7.1 (*Convergence of the Adam Algorithm*)

Let $\eta_t = \eta$ for all $t \in [T]$. Furthermore, assume that ϵ, β_2 and η are chosen such that the following conditions are satisfied :

$$\eta \leq \frac{2G\sqrt{1-\beta_2}}{L} \tag{2.7.35}$$

$$1 - \beta_2 \leq \frac{\epsilon^4}{16G^2(G+\epsilon)^2} \tag{2.7.36}$$

Then, for $(\theta_t)_t$ generated using ADAM (Algorithm 2) , we have the following inequality :

1. *If the batch size b_t is fixed (i.e, $b_t = b_0$ for all t). Then,*

$$\exists c_1, c_2 \in \mathbb{R}_+ \quad \frac{1}{T}\sum_{t=1}^{T} \mathbb{E}\left[\|\nabla f(\theta_t)\|_2^2\right] \leq \frac{c_1}{T} + c_2 \tag{2.7.37}$$

2. *If the batch size $b_t = b_0 T$ for all t. Then,*

$$\frac{1}{T}\sum_{t=1}^{T} \mathbb{E}\left[\|\nabla f(\boldsymbol{\theta}_t)\|_2^2\right] = O\left(\frac{1}{T}\right) \tag{2.7.38}$$

3. *If the batch size in linear in time (i.e, $b_t = b_0 t$ for all t). Then,*

$$\frac{1}{T}\sum_{t=1}^{T} \mathbb{E}\left[\|\nabla f(\boldsymbol{\theta}_t)\|_2^2\right] = O\left(\frac{\ln(T)}{T}\right) \tag{2.7.39}$$

4. *If the batch size is of the form $b_t = \lceil b_0 t^\gamma \rceil$ for all t (with $0 < \gamma < 1$). Then,*

$$\frac{1}{T}\sum_{t=1}^{T} \mathbb{E}\left[\|\nabla f(\boldsymbol{\theta}_t)\|_2^2\right] = O\left(\frac{1}{T^\gamma}\right) \tag{2.7.40}$$

Démonstration : We would like to understand the change in function value between two successive iterations of the algorithm 2. In the whole proof, we will consider $\beta_1 = 0$

1. **Showing that the objective function f is L-smooth**

   For all $\theta_1, \theta_2 \in \mathbb{R}^d$

$$\begin{aligned}
\|\nabla f(\boldsymbol{\theta}_1) - \nabla f(\boldsymbol{\theta}_2)\|_2 &= \|\nabla \mathbb{E}_{s\sim\mathbb{P}}[\mathcal{L}(\boldsymbol{\theta}_1; s)] - \nabla \mathbb{E}_{s\sim\mathbb{P}}[\mathcal{L}(\boldsymbol{\theta}_2; s)]\|_2 \quad \text{(from the definition 4.4.21)} \\
&= \|\mathbb{E}_{s\sim\mathbb{P}}[\nabla\mathcal{L}(\boldsymbol{\theta}_1; s)] - \mathbb{E}_{s\sim\mathbb{P}}[\nabla\mathcal{L}(\boldsymbol{\theta}_2; s)]\|_2 \\
&= \|\mathbb{E}_{s\sim\mathbb{P}}[\nabla\mathcal{L}(\boldsymbol{\theta}_1; s) - \nabla\mathcal{L}(\boldsymbol{\theta}_2; s)]\|_2 \\
&\leq \mathbb{E}_{s\sim\mathbb{P}}[\|\nabla\mathcal{L}(\boldsymbol{\theta}_1; s) - \nabla\mathcal{L}(\boldsymbol{\theta}_2; s)\|_2] \\
&\leq \mathbb{E}_{s\sim\mathbb{P}}[L\|\boldsymbol{\theta}_2 - \boldsymbol{\theta}_1\|_2] \quad \text{(from the assumption 2.7.32)} \\
&= L\|\theta_2 - \theta_1\|_2
\end{aligned}$$

   Therefore, f is L-smooth

2. **Deducing the change in the objective value between two successive iterations**.

   Let us consider $t \in [T]$. As f is L-smooth, we can deduce that :

$$f(\boldsymbol{\theta}_{t+1}) \leq f(\boldsymbol{\theta}_t) + \nabla^\top f(\boldsymbol{\theta}_t)(\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t) + \frac{L}{2}\|\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t\|_2^2 \tag{2.7.41}$$

From the update equations in algorithm 2 we have :

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta_t \frac{\mathbf{g}_t}{\left(\sqrt{\mathbf{v}_t} + \epsilon\right)}$$

Which can be expressed component-wise as follows :

$$\forall i \in [d] \quad \boldsymbol{\theta}_{i,t+1} = \boldsymbol{\theta}_{i,t} - \eta_t \frac{\mathbf{g}_{i,t}}{\left(\sqrt{v_{i,t}} + \epsilon\right)} \tag{2.7.42}$$

From 2.7.41 and 2.7.42, we deduce the following inequality :

$$f\left(\boldsymbol{\theta}_{t+1}\right) \leq f\left(\boldsymbol{\theta}_t\right) - \eta_t \sum_{i=1}^{d} \left(\left[\nabla f\left(\boldsymbol{\theta}_t\right)\right]_i \times \frac{\mathbf{g}_{i,t}}{\sqrt{\mathbf{v}_{i,t}} + \epsilon} \right) + \frac{L\eta_t^2}{2} \sum_{i=1}^{d} \frac{\mathbf{g}_{i,t}^2}{\left(\sqrt{\mathbf{v}_{i,t}} + \epsilon\right)^2}$$

Let us introduce the following notations for each time t' :

b'_t be the size of $\mathcal{B}_{t'}$.

The sigma-algebra $\mathcal{F}_{t'}$ represents *the information known at time t'*.

Consequently :

$$\boxed{\begin{aligned}\mathbb{E}\left[f\left(\boldsymbol{\theta}_{t+1}\right) \mid \mathcal{F}_t\right] \leq f\left(\boldsymbol{\theta}_t\right) & \underbrace{-\eta_t \sum_{i=1}^{d} \left(\left[\nabla f\left(\boldsymbol{\theta}_t\right)\right]_i \times \mathbb{E}\left[\frac{\mathbf{g}_{i,t}}{\sqrt{\mathbf{v}_{i,t}} + \epsilon} \mid \mathcal{F}_t\right] \right)}_{\text{(a)}} \\ & \underbrace{+ \frac{L\eta_t^2}{2} \sum_{i=1}^{d} \mathbb{E}\left[\frac{\mathbf{g}_{i,t}^2}{\left(\sqrt{\mathbf{v}_{i,t}} + \epsilon\right)^2} \mid \mathcal{F}_t\right]}_{(b)}\end{aligned}} \tag{2.7.43}$$

3. **Bounding the first term (a) in 2.7.43**

We have :

$$\begin{aligned}\mathbb{E}\left[\frac{\mathbf{g}_{i,t}}{\sqrt{\mathbf{v}_{i,t}} + \epsilon} \mid \mathcal{F}_t\right] &= \mathbb{E}\left[\frac{\boldsymbol{g}_{i,t}}{\sqrt{\boldsymbol{v}_{i,t}} + \epsilon} - \frac{\boldsymbol{g}_{i,t}}{\sqrt{\beta_2 \boldsymbol{v}_{i,t-1}} + \epsilon} + \frac{\boldsymbol{g}_{i,t}}{\sqrt{\beta_2 \boldsymbol{v}_{i,t-1}} + \epsilon} \mid \mathcal{F}_t\right] \\ &= \mathbb{E}\left[\frac{\mathbf{g}_{i,t}}{\sqrt{\mathbf{v}_{i,t}} + \epsilon} - \frac{\mathbf{g}_{i,t}}{\sqrt{\beta_2 \mathbf{v}_{i,t-1}} + \epsilon} \mid \mathcal{F}_t\right] + \underbrace{\mathbb{E}\left[\frac{\mathbf{g}_{i,t}}{\sqrt{\beta_2 \mathbf{v}_{i,t-1}} + \epsilon} \mid \mathcal{F}_t\right]} \\ &= \mathbb{E}\left[\frac{\mathbf{g}_{i,t}}{\sqrt{\mathbf{v}_{i,t}} + \epsilon} - \frac{\mathbf{g}_{i,t}}{\sqrt{\beta_2 \mathbf{v}_{i,t-1}} + \epsilon} \mid \mathcal{F}_t\right] + \frac{[\nabla f(\boldsymbol{\theta})]_i}{\sqrt{\beta_2 \mathbf{v}_{i,t-1}} + \epsilon}\end{aligned}$$

Which enables us to rewrite (a) defined in 2.7.43 as follows :

$$\begin{aligned}(a) &= -\eta_t \sum_{i=1}^{d} \left(\left[\nabla f\left(\boldsymbol{\theta}_t\right)\right]_i \times \left[\frac{\left[\nabla f\left(\boldsymbol{\theta}_t\right)\right]_i}{\sqrt{\beta_2 \boldsymbol{v}_{i,t-1}} + \epsilon} + \mathbb{E}\left[\frac{\boldsymbol{g}_{i,t}}{\sqrt{\boldsymbol{v}_{i,t}} + \epsilon} - \frac{\boldsymbol{g}_{i,t}}{\sqrt{\beta_2 \boldsymbol{v}_{i,t-1}} + \epsilon} \mid \mathcal{F}_t\right] \right] \right) \\ &= -\eta_t \sum_{i=1}^{d} \frac{\left[\nabla f\left(\boldsymbol{\theta}_t\right)\right]_i^2}{\sqrt{\beta_2 \boldsymbol{v}_{i,t-1}} + \epsilon} \underbrace{-\eta_t \sum_{i=1}^{d} \left[\nabla f\left(\boldsymbol{\theta}_t\right)\right]_i \times \mathbb{E}\left[\frac{\boldsymbol{g}_{i,t}}{\sqrt{\boldsymbol{v}_{i,t}} + \epsilon} - \frac{\boldsymbol{g}_{i,t}}{\sqrt{\beta_2 \boldsymbol{v}_{i,t-1}} + \epsilon} \mid \mathcal{F}_t\right]}_{(a_1)}\end{aligned} \tag{2.7.44}$$

Let us bound the term (a_1) in 2.7.44 :

$$
\begin{aligned}
-\eta_t \sum_{i=1}^{d} \left[\nabla f\left(\boldsymbol{\theta}_t\right)\right]_i \times \mathbb{E}\left[\frac{\boldsymbol{g}_{i,t}}{\sqrt{\boldsymbol{v}_{i,t}}+\epsilon} - \frac{\boldsymbol{g}_{i,t}}{\sqrt{\beta_2 \boldsymbol{v}_{i,t-1}}+\epsilon} \mid \mathcal{F}_t\right] & \\
\leq \left|\eta_t \sum_{i=1}^{d} \left[\nabla f\left(\boldsymbol{\theta}_t\right)\right]_i \times \mathbb{E}\left[\frac{\boldsymbol{g}_{i,t}}{\sqrt{\boldsymbol{v}_{i,t}}+\epsilon} - \frac{\boldsymbol{g}_{i,t}}{\sqrt{\beta_2 \boldsymbol{v}_{i,t-1}}+\epsilon} \mid \mathcal{F}_t\right]\right| & \\
\leq \eta_t \sum_{i=1}^{d} \left|\left[\nabla f\left(\boldsymbol{\theta}_t\right)\right]_i\right| \left| \times \mathbb{E}\left[\frac{\boldsymbol{g}_{i,t}}{\sqrt{\boldsymbol{v}_{i,t}}+\epsilon} - \frac{\boldsymbol{g}_{i,t}}{\sqrt{\beta_2 \boldsymbol{v}_{i,t-1}}+\epsilon} \mid \mathcal{F}_t\right]\right| & \\
\leq \eta_t \sum_{i=1}^{d} \left|\left[\nabla f\left(\boldsymbol{\theta}_t\right)\right]_i\right| \times \mathbb{E}\left[\underbrace{\left|\frac{\boldsymbol{g}_{i,t}}{\sqrt{\boldsymbol{v}_{i,t}}+\epsilon} - \frac{\boldsymbol{g}_{i,t}}{\sqrt{\beta_2 \boldsymbol{v}_{i,t-1}}+\epsilon}\right|}_{(a_2)} \mid \mathcal{F}_t\right] &
\end{aligned}
\tag{2.7.45}
$$

By using the update rule $\mathbf{v}_{i,t} = \beta_2 \mathbf{v}_{i,t-1} + (1-\beta_2)\,\mathbf{g}_{i,t}^2$ from algorithm 2, we can bound the term (a_2) in 2.7.45 :

$$
\begin{aligned}
(a_2) &= \left|\boldsymbol{g}_{i,t}\right| \left|\frac{1}{\sqrt{\boldsymbol{v}_{i,t}}+\epsilon} - \frac{1}{\sqrt{\beta_2 \boldsymbol{v}_{i,t-1}}+\epsilon}\right| \\
&= \frac{\left|\boldsymbol{g}_{i,t}\right|}{\left(\sqrt{\boldsymbol{v}_{i,t}}+\epsilon\right)\left(\sqrt{\beta_2 \boldsymbol{v}_{i,t-1}}+\epsilon\right)} \left|\sqrt{\boldsymbol{v}_{i,t}} - \sqrt{\beta_2 \boldsymbol{v}_{i,t-1}}\right| \\
&= \frac{\left|\boldsymbol{g}_{i,t}\right|}{\left(\sqrt{\boldsymbol{v}_{i,t}}+\epsilon\right)\left(\sqrt{\beta_2 \boldsymbol{v}_{i,t-1}}+\epsilon\right)} \frac{\left|\boldsymbol{v}_{i,t} - \beta_2 \boldsymbol{v}_{i,t-1}\right|}{\sqrt{\boldsymbol{v}_{i,t}} + \sqrt{\beta_2 \boldsymbol{v}_{i,t-1}}} \\
&= \frac{\left|\boldsymbol{g}_{i,t}\right|}{\left(\sqrt{\boldsymbol{v}_{i,t}}+\epsilon\right)\left(\sqrt{\beta_2 \boldsymbol{v}_{i,t-1}}+\epsilon\right)} \frac{(1-\beta_2)\,\mathbf{g}_{i,t}^2}{\sqrt{\beta_2 \mathbf{v}_{i,t-1} + (1-\beta_2)\,\mathbf{g}_{i,t}^2} + \sqrt{\beta_2 \mathbf{v}_{i,t-1}}} \quad \text{(by using the update rule)} \\
&\leq \frac{\left|\boldsymbol{g}_{i,t}\right|}{\left(\sqrt{\boldsymbol{v}_{i,t}}+\epsilon\right)\left(\sqrt{\beta_2 \boldsymbol{v}_{i,t-1}}+\epsilon\right)} \frac{(1-\beta_2)\,\mathbf{g}_{i,t}^2}{\sqrt{\beta_2 \mathbf{v}_{i,t-1} + (1-\beta_2)\,\mathbf{g}_{i,t}^2}} \quad \text{(since } \sqrt{\beta_2 \mathbf{v}_{i,t-1}} \geq 0\text{)} \\
&\leq \frac{\left|\boldsymbol{g}_{i,t}\right|}{\epsilon\left(\sqrt{\beta_2 \boldsymbol{v}_{i,t-1}}+\epsilon\right)} \frac{(1-\beta_2)\,\mathbf{g}_{i,t}^2}{\sqrt{(1-\beta_2)\,\mathbf{g}_{i,t}^2}} \quad \text{(since } \sqrt{v_{i,t}} \geq 0 \text{ and } \beta_2 \mathbf{v}_{i,t-1} \geq 0\text{)} \\
&= \frac{\sqrt{1-\beta_2}\mathbf{g}_{i,t}^2}{\epsilon\left(\sqrt{\beta_2 \boldsymbol{v}_{i,t-1}}+\epsilon\right)}
\end{aligned}
\tag{2.7.46}
$$

From 2.7.44, 2.7.45 and 2.7.46, we deduce the following inequality :

$$
(a_1) \leq \eta_t \sum_{i=1}^{d} \left(\left|\left[\nabla f\left(\boldsymbol{\theta}_t\right)\right]_i\right| \frac{\sqrt{1-\beta_2}}{\epsilon} \mathbb{E}\left[\frac{\mathbf{g}_{i,t}^2}{\sqrt{\beta_2 \mathbf{v}_{i,t-1}}+\epsilon} \mid \mathcal{F}_t\right]\right) \tag{2.7.47}
$$

Therefore, we deduce a bound for (a) from 2.7.44 and 2.7.47 :

$$
(a) \leq -\eta_t \sum_{i=1}^{d} \frac{\left[\nabla f\left(\boldsymbol{\theta}_t\right)\right]_i^2}{\sqrt{\beta_2 \boldsymbol{v}_{i,t-1}}+\epsilon} + \eta_t \sum_{i=1}^{d} \left(\left|\left[\nabla f\left(\boldsymbol{\theta}_t\right)\right]_i\right| \frac{\sqrt{1-\beta_2}}{\epsilon} \mathbb{E}\left[\frac{\mathbf{g}_{i,t}^2}{\sqrt{\beta_2 \mathbf{v}_{i,t-1}}+\epsilon} \mid \mathcal{F}_t\right]\right) \tag{2.7.48}
$$

By using the assumption 2.7.33, we can also bound the term $|\left[\nabla f\left(\boldsymbol{\theta}_t\right)\right]_i$ for all $i \in [d]$.

Indeed,

$$\begin{aligned}\forall i \in [d] \quad |[\nabla f\left(\boldsymbol{\theta}_t\right)]_i| &\leq \|\nabla f(\boldsymbol{\theta_t})\|_2 \\ &:= \|\mathbb{E}_{s\sim\mathbb{P}}[\mathcal{L}(\theta_t, s)]\|_2 \\ &\leq \mathbb{E}_{s\sim\mathbb{P}}[\|\nabla\mathcal{L}(\boldsymbol{\theta_t}; s)\|] \\ &\leq G \quad \text{(from assumption2.7.33)}\end{aligned}$$

So,

$$\forall i \in [d] \quad |[\nabla f\left(\boldsymbol{\theta}_t\right)]_i| \leq G \tag{2.7.49}$$

From 2.7.48 and 2.7.49 we deduce :

$$\boxed{(a) \leq -\eta_t \sum_{i=1}^{d} \frac{[\nabla f\left(\boldsymbol{\theta}_t\right)]_i^2}{\sqrt{\beta_2 \boldsymbol{v}_{i,t-1}} + \epsilon} + \frac{\eta_t G\sqrt{1-\beta_2}}{\epsilon} \sum_{i=1}^{d} \mathbb{E}\left[\frac{\mathbf{g}_{i,t}^2}{\sqrt{\beta_2 \mathbf{v}_{i,t-1}} + \epsilon} \mid \mathcal{F}_t\right]} \tag{2.7.50}$$

4. **Bounding the second term (b) in 2.7.43**

By using the update rule $\mathbf{v}_{i,t} = \beta_2 \mathbf{v}_{i,t-1} + (1-\beta_2)\,\mathbf{g}_{i,t}^2$ from algorithm 2 in the expression (b), we get :

$$\begin{aligned}(b) &:= \frac{L\eta_t^2}{2} \sum_{i=1}^{d} \mathbb{E}\left[\frac{\mathbf{g}_{i,t}^2}{\left(\sqrt{\mathbf{v}_{i,t}} + \epsilon\right)^2} \mid \mathcal{F}_t\right] \\ &= \frac{L\eta_t^2}{2} \sum_{i=1}^{d} \mathbb{E}\left[\frac{\boldsymbol{g}_{i,t}^2}{\left(\sqrt{\beta_2 \boldsymbol{v}_{i,t-1} + (1-\beta_2)\,\boldsymbol{g}_{i,t}^2} + \epsilon\right)^2} \mid \mathcal{F}_t\right] \\ &\leq \frac{L\eta_t^2}{2} \sum_{i=1}^{d} \mathbb{E}\left[\frac{\boldsymbol{g}_{i,t}^2}{\left(\sqrt{\beta_2 \boldsymbol{v}_{i,t-1}} + \epsilon\right)^2} \mid \mathcal{F}_t\right] \quad (\text{since } (1-\beta_2)\,\boldsymbol{g}_{i,t}^2 \geq 0) \\ &\leq \frac{L\eta_t^2}{2\epsilon} \sum_{i=1}^{d} \mathbb{E}\left[\frac{g_{i,t}^2}{\sqrt{\beta_2 \mathbf{v}_{i,t-1}} + \epsilon} \mid \mathcal{F}_t\right] \quad (\text{since} \sqrt{\beta_2 \mathbf{v}_{i,t-1}} \geq 0)\end{aligned}$$

So,

$$\boxed{(b) \leq \frac{L\eta_t^2}{2\epsilon} \sum_{i=1}^{d} \mathbb{E}\left[\frac{g_{i,t}^2}{\sqrt{\beta_2 \mathbf{v}_{i,t-1}} + \epsilon} \mid \mathcal{F}_t\right]} \tag{2.7.51}$$

5. **Combining the upper bounds on (a) and (b) defined in 2.7.43**

By combining the upper bounds 2.7.50 and 2.7.51, we get the following inequality :

$$\mathbb{E}\left[f\left(\boldsymbol{\theta}_{t+1}\right) \mid \mathcal{F}_t\right] \leq f\left(\boldsymbol{\theta}_t\right) \underbrace{-\eta_t \sum_{i=1}^{d} \frac{\left[\nabla f\left(\boldsymbol{\theta}_t\right)\right]_i^2}{\sqrt{\beta_2 \boldsymbol{v}_{i,t-1}} + \epsilon}}_{(c)} + \underbrace{\frac{\eta_t G \sqrt{1-\beta_2}}{\epsilon} \sum_{i=1}^{d} \mathbb{E}\left[\frac{\mathbf{g}_{i,t}^2}{\sqrt{\beta_2 \mathbf{v}_{i,t-1}} + \epsilon} \mid \mathcal{F}_t\right]}_{(d)} + \underbrace{\frac{L\eta_t^2}{2\epsilon} \sum_{i=1}^{d} \mathbb{E}\left[\frac{g_{i,t}^2}{\sqrt{\beta_2 \mathbf{v}_{i,t-1}} + \epsilon} \mid \mathcal{F}_t\right]}_{(e)} \tag{2.7.52}$$

— Bounding the sum of (d) and (e) defined in 2.7.52 :

We have :

$$\begin{aligned}(d) + (e) &= \left(\frac{\eta_t G \sqrt{1-\beta_2}}{\epsilon} + \frac{L\eta_t^2}{2\epsilon}\right) \sum_{i=1}^{d} \mathbb{E}\left[\frac{\mathbf{g}_{i,t}^2}{\sqrt{\beta_2 \mathbf{v}_{i,t-1}} + \epsilon} \mid \mathcal{F}_t\right] \\ &\leq \frac{1}{\epsilon}\left(\frac{\eta_t G \sqrt{1-\beta_2}}{\epsilon} + \frac{L\eta_t^2}{2\epsilon}\right) \sum_{i=1}^{d} \mathbb{E}\left[\mathbf{g}_{i,t}^2 \mid \mathcal{F}_t\right] \quad (\text{since } \sqrt{\beta_2 \mathbf{v}_{i,t-1}} \geq 0)\end{aligned} \tag{2.7.53}$$

— Bounding the expression (c) defined in 2.7.52 :

To that end, we first need to prove that $\forall i \in [d]\ \forall t' \in [T]\ v_{i,t'} \leq G^2$. We can do it by induction on t'.

— It's true for $t' = 0$

— Let's consider $t' \in [T]$ such that $\forall i \in [d]\ v_{i,t'-1} \leq G^2$. We have :

$$\begin{aligned}\forall i \in [d]\ v_{i,t'} &= \beta_2 \mathbf{v}_{i,t-1} + (1-\beta_2)\, \mathbf{g}_{i,t}^2 \\ &\leq \beta_2 G^2 + (1-\beta_2)\, \mathbf{g}_{i,t}^2 \quad (\text{by induction hypothesis}) \\ &\leq \beta_2 G^2 + (1-\beta_2)\, \|\mathbf{g}_t\|_2^2 \\ &= \beta_2 G^2 + (1-\beta_2) \left|\left| \frac{1}{b_t} \sum_{s \in \mathcal{B}_t} \nabla \mathcal{L}(\theta_t, s) \right|\right|_2^2 \quad (\text{by definition of } g_t) \\ &\leq \beta_2 G^2 + (1-\beta_2)\, \frac{1}{b_t^2} \sum_{s \in \mathcal{B}_t} ||\nabla \mathcal{L}(\theta_t, s)||_2^2 \\ &\leq \beta_2 G^2 + (1-\beta_2)\, \frac{1}{b_t^2} \sum_{s \in \mathcal{B}_t} G^2 \quad (\text{by assumption 2.7.33}) \\ &= \beta_2 G^2 + (1-\beta_2)\, \frac{1}{b_t^2} b_t G^2 \\ &= \beta_2 G^2 + \frac{(1-\beta_2)}{b_t} G^2 \\ &\leq \beta_2 G^2 + (1-\beta_2)\, G^2 \quad (\text{since} b_t \geq 1) \\ &= G^2\end{aligned}$$

We conclude by induction that,

$$\forall i \in [d]\ \forall t' \in [T]\ v_{i,t'} \leq G^2 \tag{2.7.54}$$

Consequently,

$$\begin{aligned}(c) &:= -\eta_t \sum_{i=1}^{d} \frac{[\nabla f(\boldsymbol{\theta}_t)]_i^2}{\sqrt{\beta_2 \boldsymbol{v}_{i,t-1}} + \epsilon} \\ &\leq -\frac{\eta_t}{\sqrt{\beta_2} G + \epsilon} \sum_{i=1}^{d} [\nabla f(\boldsymbol{\theta}_t)]_i^2 \quad \text{(by using 2.7.54)} \\ &= -\frac{\eta_t}{\sqrt{\beta_2} G + \epsilon} \|\nabla f(\boldsymbol{\theta}_t)\|_2^2 \end{aligned} \tag{2.7.55}$$

— Combining the results :

By using the inequalities 2.7.53 and 2.7.55, the upper bound in 2.7.52 becomes :

$$\boxed{\begin{aligned}\mathbb{E}[f(\boldsymbol{\theta}_{t+1}) \mid \mathcal{F}_t] \leq f(\boldsymbol{\theta}_t) &- \frac{\eta_t}{\sqrt{\beta_2} G + \epsilon} \|\nabla f(\boldsymbol{\theta}_t)\|_2^2 \\ &+ \frac{1}{\epsilon} \left(\frac{\eta_t G \sqrt{1-\beta_2}}{\epsilon} + \frac{L \eta_t^2}{2\epsilon} \right) \underbrace{\mathbb{E}\left[\|\mathbf{g}_t\|_2^2 \mid \mathcal{F}_t \right]}_{(f)}\end{aligned}} \tag{2.7.56}$$

6. **Bounding the last term (f)**

Let us introduce the following notations :

$$\xi_t := \frac{1}{b_t} \sum_{s \in \mathcal{B}_t} (\nabla \mathcal{L}(\theta_t, s) - \nabla f(\theta_t)) \tag{2.7.57}$$

$$\forall s \in \mathcal{B}_t \quad Y_s := \nabla \mathcal{L}(\theta_t, s) - \nabla f(\theta_t) \tag{2.7.58}$$

$$Y := \sum_{s \in \mathcal{B}_t} Y_s \tag{2.7.59}$$

Then,

$$\xi_t := \frac{1}{b_t} Y \tag{2.7.60}$$

Consequently,

$$
\begin{aligned}
(f) &:= \mathbb{E}\left[\|\mathbf{g}_t\|_2^2 \mid \mathcal{F}_t\right] \\
&= \mathbb{E}\left[\left\|\frac{1}{b_t}\sum_{s\in\mathcal{B}_t}\nabla\mathcal{L}(\theta_t,s)\right\|_2^2 \mid \mathcal{F}_t\right] \quad \text{(by definition)} \\
&= \mathbb{E}\left[\left\|\frac{1}{b_t}\sum_{s\in\mathcal{B}_t}\left(\nabla\mathcal{L}(\theta_t,s)-\nabla f(\theta_t)+\nabla f(\theta_t)\right)\right\|_2^2 \mid \mathcal{F}_t\right] \\
&= \mathbb{E}\left[\left\|\left(\frac{1}{b_t}\sum_{s\in\mathcal{B}_t}\left(\nabla\mathcal{L}(\theta_t,s)-\nabla f(\theta_t)\right)\right)+\nabla f(\theta_t)\right\|_2^2 \mid \mathcal{F}_t\right] \\
&= \mathbb{E}\left[\|\xi_t+\nabla f(\theta_t)\|_2^2 \mid \mathcal{F}_t\right] \quad \text{(by definition 2.7.57)} \\
&= \mathbb{E}\left[\left(\xi_t+\nabla f(\theta_t)\right)^\top\left(\xi_t+\nabla f(\theta_t)\right) \mid \mathcal{F}_t\right] \\
&= \mathbb{E}\left[\|\xi_t\|_2^2 \mid \mathcal{F}_t\right] + \underbrace{\mathbb{E}\left[\xi_t \mid \mathcal{F}_t\right]^\top}_{=0}\nabla f(\theta_t) + \nabla f(\theta_t)^\top\underbrace{\mathbb{E}\left[\xi_t \mid \mathcal{F}_t\right]}_{=0} + \|\nabla f(\theta_t)\|_2^2 \\
&= \frac{1}{b_t^2}\mathbb{E}\left[\|Y\|_2^2 \mid \mathcal{F}_t\right] + \|\nabla f(\theta_t)\|_2^2 \quad \text{(using 2.7.60)} \\
&= \frac{1}{b_t^2}\mathbb{E}\left[\left\|\left(\sum_{s\in\mathcal{B}_t}Y_s\right)^\top\left(\sum_{s'\in\mathcal{B}_t}Y_{s'}\right)\right\|_2^2 \mid \mathcal{F}_t\right] + \|\nabla f(\theta_t)\|_2^2 \quad \text{(using 2.7.59)} \\
&= \frac{1}{b_t^2}\sum_{s,s'\in\mathcal{B}_t}\mathbb{E}\left[Y_s^\top Y_{s'} \mid \mathcal{F}_t\right] + \|\nabla f(\theta_t)\|_2^2 \\
&= \frac{1}{b_t^2}\sum_{\substack{s,s'\in\mathcal{B}_t\\ s\neq s'}}\mathbb{E}\left[Y_s^\top Y_{s'} \mid \mathcal{F}_t\right] + \frac{1}{b_t^2}\sum_{\substack{s,s'\in\mathcal{B}_t\\ s=s'}}\mathbb{E}\left[Y_s^\top Y_{s'} \mid \mathcal{F}_t\right] + \|\nabla f(\theta_t)\|_2^2 \\
&= \frac{1}{b_t^2}\sum_{\substack{s,s'\in\mathcal{B}_t\\ s\neq s'}}\mathbb{E}\left[Y_s \mid \mathcal{F}_t\right]^\top\underbrace{\mathbb{E}\left[Y_{s'} \mid \mathcal{F}_t\right]}_{=0} + \frac{1}{b_t^2}\sum_{s\in\mathcal{B}_t}\mathbb{E}\left[\|Y_s\|_2^2 \mid \mathcal{F}_t\right] + \|\nabla f(\theta_t)\|_2^2 \quad (\text{since} Y_s Y_{s'} \mid \mathcal{F}_t) \\
&= \frac{1}{b_t^2}\sum_{s\in\mathcal{B}_t}\mathbb{E}\left[\|\nabla\mathcal{L}(\theta_t,s)-\nabla f(\theta_t)\|_2^2 \mid \mathcal{F}_t\right] + \|\nabla f(\theta_t)\|_2^2 \quad \text{(by definition 2.7.58)} \\
&\leq \frac{1}{b_t^2}\sum_{s\in\mathcal{B}_t}\sigma^2 + \|\nabla f(\theta_t)\|_2^2 \quad \text{(using the assumption 2.7.34)} \\
&\leq \frac{\sigma^2}{b_t} + \|\nabla f(\theta_t)\|_2^2
\end{aligned}
$$

We conclude the following bound on the (f) term defined in 2.7.56 :

$$(f) \leq \frac{\sigma^2}{b_t} + \|\nabla f(\theta_t)\|_2^2 \tag{2.7.61}$$

By using the bound 2.7.61, the inequality 2.7.56 becomes :

$$\mathbb{E}\left[f\left(\boldsymbol{\theta}_{t+1}\right) \mid \mathcal{F}_t\right] \leq f\left(\boldsymbol{\theta}_t\right) - \frac{\eta_t}{\sqrt{\beta_2}G+\epsilon}\left\|\nabla f\left(\boldsymbol{\theta}_t\right)\right\|_2^2 + \frac{1}{\epsilon}\left(\frac{\eta_t G\sqrt{1-\beta_2}}{\epsilon} + \frac{L\eta_t^2}{2\epsilon}\right)\left(\frac{\sigma^2}{b_t} + \|\nabla f(\theta_t)\|_2^2\right)$$

Which results in the following inequality :

$$\boxed{\mathbb{E}\left[f\left(\boldsymbol{\theta}_{t+1}\right) \mid \mathcal{F}_t\right] \leq f\left(\boldsymbol{\theta}_t\right) + \left\|\nabla f\left(\boldsymbol{\theta}_t\right)\right\|_2^2 \underbrace{\left(\frac{-\eta_t}{\sqrt{\beta_2}G + \epsilon} + \frac{1}{\epsilon}\left(\frac{\eta_t G\sqrt{1-\beta_2}}{\epsilon} + \frac{L\eta_t^2}{2\epsilon}\right)\right)}_{(i)} + \underbrace{\frac{\eta_t \sigma^2}{\epsilon b_t}\left(\frac{G\sqrt{1-\beta_2}}{\epsilon} + \frac{L\eta_t}{2\epsilon}\right)}_{(ii)}} \tag{2.7.62}$$

7. **Incorporating the assumptions on the hyperparameters**

The final step of the proof is to use the conditions on the hyperparameters to bound (i) and (ii).

— Using the choice of η

Based on the condition 2.7.36, the learning rate is chosen to be fixed such that :

$$\eta \leq \frac{2G\sqrt{1-\beta_2}}{L}$$

Therefore,

$$\frac{L\eta}{2\epsilon} \leq \frac{G\sqrt{1-\beta_2}}{\epsilon} \tag{2.7.63}$$

We can then bound the first term (i) as follows :

$$\begin{aligned}
(i) &:= -\eta\left(\frac{1}{\sqrt{\beta_2}G + \epsilon} - \frac{1}{\epsilon}\left(\frac{G\sqrt{1-\beta_2}}{\epsilon} + \frac{L\eta_t}{2\epsilon}\right)\right) \\
&\leq -\eta\left(\frac{1}{\sqrt{\beta_2}G + \epsilon} - \frac{1}{\epsilon}\left(\frac{G\sqrt{1-\beta_2}}{\epsilon} + \frac{G\sqrt{1-\beta_2}}{\epsilon}\right)\right) \quad \text{(using 2.7.63)} \\
&= -\eta\left(\frac{1}{\sqrt{\beta_2}G + \epsilon} - \underbrace{\frac{2G\sqrt{1-\beta_2}}{\epsilon^2}}_{(iii)}\right)
\end{aligned} \tag{2.7.64}$$

We can use the inequality 2.7.63 to bound the second term (ii) as follows :

$$\begin{aligned}
(ii) &:= \frac{\eta_t \sigma^2}{\epsilon b_t}\left(\frac{G\sqrt{1-\beta_2}}{\epsilon} + \frac{L\eta_t}{2\epsilon}\right) \\
&\leq \frac{\eta_t \sigma^2}{\epsilon b_t}\left(\frac{G\sqrt{1-\beta_2}}{\epsilon} + \frac{G\sqrt{1-\beta_2}}{\epsilon}\right) \quad \text{(using 2.7.63)} \\
&= \frac{2\eta\sigma^2 G\sqrt{1-\beta_2}}{\epsilon^2 b_t}
\end{aligned} \tag{2.7.65}$$

— Using the choice of β_2 to bound (iii)

By using condition 2.7.36, we can bound (iii) :

$$\begin{aligned}\frac{2G\sqrt{1-\beta_2}}{\epsilon^2} &= \sqrt{\frac{4G^2(1-\beta_2)}{\epsilon^4}} \\ &\leq \sqrt{\frac{4G^2}{\epsilon^4}\frac{\epsilon^4}{16G^2(G+\epsilon)^2}} \quad \text{(using 2.7.36)} \\ &= \frac{1}{2}\frac{1}{G+\epsilon} \\ &\leq \frac{1}{2}\frac{1}{\sqrt{\beta_2}G+\epsilon} \quad \text{(since } \beta_2 \leq 1\text{)} \end{aligned} \tag{2.7.66}$$

— Deducing a new bound for (i)

From 2.7.66, we conclude that :

$$\frac{1}{\sqrt{\beta_2}G+\epsilon} - \frac{2G\sqrt{1-\beta_2}}{\epsilon^2} \geq \frac{1}{2}\frac{1}{\sqrt{\beta_2}G+\epsilon}$$

The inequality 2.7.64 becomes

$$(i) \leq -\frac{\eta}{2(\sqrt{\beta_2}G+\epsilon)} \tag{2.7.67}$$

Finally, by using 2.7.67 and 2.7.65, we get the following update to the inequality 2.7.62 :

$$\boxed{\mathbb{E}\left[f\left(\boldsymbol{\theta}_{t+1}\right) \mid \mathcal{F}_t\right] \leq f\left(\boldsymbol{\theta}_t\right) - \frac{\eta}{2(\sqrt{\beta_2}G+\epsilon)}\left\|\nabla\mathcal{L}\left(\boldsymbol{\theta}_t\right)\right\|_2^2 + \frac{2\eta\sigma^2G\sqrt{1-\beta_2}}{\epsilon^2 b_t}} \tag{2.7.68}$$

8. **Concluding according to the batch size**

Notations :

Let us define the following constants :

$$\begin{aligned}\Delta &= \frac{\eta}{2(\sqrt{\beta_2}G+\epsilon)} \\ \alpha &= \frac{2\eta\sigma^2G\sqrt{1-\beta_2}}{\epsilon^2}\end{aligned}$$

The inequality 2.7.68 can then be written as follows :

$$\mathbb{E}\left[f\left(\boldsymbol{\theta}_{t+1}\right) \mid \mathcal{F}_t\right] \leq f\left(\boldsymbol{\theta}_t\right) - \Delta\left\|\nabla\mathcal{L}\left(\boldsymbol{\theta}_t\right)\right\|_2^2 + \frac{\alpha}{b_t} \tag{2.7.69}$$

By taking the expected value of the inequality 2.7.69,

$$\mathbb{E}\left[f\left(\boldsymbol{\theta}_{t+1}\right)\right] \leq \mathbb{E}\left[f\left(\boldsymbol{\theta}_t\right)\right] - \Delta\mathbb{E}\left[\left\|\nabla\mathcal{L}\left(\boldsymbol{\theta}_t\right)\right\|_2^2\right] + \frac{\alpha}{b_t}$$

Which can be rearranged as follows :

$$\mathbb{E}\left[\|\nabla\mathcal{L}\left(\boldsymbol{\theta}_t\right)\|_2^2\right] \leq \frac{\mathbb{E}\left[f\left(\boldsymbol{\theta}_t\right)\right] - \mathbb{E}\left[f\left(\boldsymbol{\theta}_{t+1}\right)\right]}{\Delta} + \frac{\alpha}{\Delta b_t} \tag{2.7.70}$$

By summing 2.7.70 for all $t \in [T]$, we get :

$$\begin{aligned}
\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\|\nabla\mathcal{L}\left(\boldsymbol{\theta}_t\right)\|_2^2\right] &\leq \frac{1}{T\Delta}\sum_{t=1}^{T}\left(\mathbb{E}\left[f\left(\boldsymbol{\theta}_t\right)\right] - \mathbb{E}\left[f\left(\boldsymbol{\theta}_{t+1}\right)\right]\right) + \frac{\alpha}{T\Delta}\sum_{t=1}^{T}\frac{1}{b_t} \\
&= \frac{1}{T\Delta}\left(f\left(\boldsymbol{\theta}_1\right) - \mathbb{E}\left[f\left(\boldsymbol{\theta}_{T+1}\right)\right]\right) + \frac{\alpha}{T\Delta}\sum_{t=1}^{T}\frac{1}{b_t} \quad \text{(using telescoping sum)} \\
&\leq \frac{1}{T\Delta}\left(f\left(\boldsymbol{\theta}_1\right) - f\left(\boldsymbol{\theta}^*\right)\right) + \frac{\alpha}{T\Delta}\sum_{t=1}^{T}\frac{1}{b_t} \quad \left(\text{where } \theta^* := \arg\min_{\theta} f(\theta)\right)
\end{aligned}$$

We conclude that :

$$\boxed{\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\|\nabla\mathcal{L}\left(\boldsymbol{\theta}_t\right)\|_2^2\right] \leq \frac{f\left(\boldsymbol{\theta}_1\right) - f\left(\boldsymbol{\theta}^*\right)}{T\Delta} + \frac{\alpha}{T\Delta}\sum_{t=1}^{T}\frac{1}{b_t}} \tag{2.7.71}$$

— If the batch size is fixed : $b_t = b_0$ for all t :

Then, the inequality 2.7.71 becomes :

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\|\nabla\mathcal{L}\left(\boldsymbol{\theta}_t\right)\|_2^2\right] \leq \frac{f\left(\boldsymbol{\theta}_1\right) - f\left(\boldsymbol{\theta}^*\right)}{T\Delta} + \frac{\alpha}{\Delta b_0}$$

Let us denote $c_1 = \frac{f(\boldsymbol{\theta}_1) - f(\boldsymbol{\theta}^*)}{\Delta}$ and $c_2 = \frac{\alpha}{\Delta b_0}$, we conclude the first part 2.7.37 of the theorem :

$$\boxed{\exists c_1, c_2 \in \mathbb{R}_+ \quad \frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\|\nabla\mathcal{L}\left(\boldsymbol{\theta}_t\right)\|_2^2\right] \leq \frac{c_1}{T} + c_2}$$

— If the batch size $b_t = b_0 T$ for all t :

Then, the inequality 2.7.71 becomes :

$$\begin{aligned}
\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\|\nabla\mathcal{L}\left(\boldsymbol{\theta}_t\right)\|_2^2\right] &\leq \frac{c_1}{T} + \frac{c_2}{T}\sum_{t=1}^{T}\frac{1}{T} \\
&= \frac{c_1 + c_2}{T}
\end{aligned}$$

We conclude the part 2.7.38 of the theorem, i.e :

$$\boxed{\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\|\nabla\mathcal{L}\left(\boldsymbol{\theta}_t\right)\|_2^2\right] = O\left(\frac{1}{T}\right)}$$

— If the batch size in linear in time (i.e, $b_t = b_0 t$ for all t) :

Then, the inequality 2.7.71 becomes :

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\|\nabla\mathcal{L}\left(\boldsymbol{\theta}_t\right)\|_2^2\right] \leq \underbrace{\frac{c_1}{T}+\frac{c_2}{T}\sum_{t=1}^{T}\frac{1}{t}}_{(\mathcal{E}_1)} \tag{2.7.72}$$

We would like to find an equivalent to $(\mathcal{E}_1)$.

By applying the mean value theorem to the function $\Phi : t \mapsto \ln(t)$ between t and $t+1$ (for a fixed $t \in [T]$), there exists $c_t \underset{t\to+\infty}{\sim} t$ such that :

$$\Phi(t+1)-\Phi(t) = \frac{1}{c_t} \sim \frac{1}{t}$$

As, $\sum(\frac{1}{t})_t$ diverges, we conclude that :

$$\sum_{t=1}^{T}\left(\Phi(t+1)-\Phi(t)\right) \underset{t\to+\infty}{\sim} \sum_{t=1}^{T}\frac{1}{t}$$

By telescoping sum, it implies :

$$\ln(T) \underset{T\to+\infty}{\sim} \sum_{t=1}^{T}\frac{1}{t}$$

Which gives, by deviding by T :

$$\frac{1}{T}\sum_{t=1}^{T}\frac{1}{t} \underset{T\to+\infty}{\sim} \frac{\ln(T)}{T}$$

On the other hand,

$$\frac{1}{T} = o\left(\frac{\ln(T)}{T}\right)$$

Which gives an equivalent to the bound $(\mathcal{E}_1)$ in 2.7.72 :

$$(\mathcal{E}_1) \underset{T\to+\infty}{\sim} c_2\frac{\ln(T)}{T} \tag{2.7.73}$$

From 2.7.71 and 2.7.73 we conclude the part 2.7.39 of the theorem :

$$\boxed{\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\|\nabla f\left(\boldsymbol{\theta}_t\right)\|_2^2\right] = O\left(\frac{\ln(T)}{T}\right)}$$

— If the batch size is of the form $b_t = \lceil b_0 t^\gamma \rceil$ for all t (with $0 < \gamma < 1$) :

Then, the inequality 2.7.71 becomes :

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left[\|\nabla\mathcal{L}\left(\boldsymbol{\theta}_t\right)\|_2^2\right] \leq \underbrace{\frac{c_1}{T}+\frac{c_2}{T}\sum_{t=1}^{T}\frac{1}{t^\gamma}}_{(\mathcal{E}_2)} \tag{2.7.74}$$

We would like to find an equivalent to $(\mathcal{E}_2)$:

By applying the mean value theorem to the function $\Psi : t \mapsto \frac{1}{1-\gamma} t^{1-\gamma}$ between $t-1$ and t, there exists $c_t \sim t$ such that :

$$\Psi(t) - \Psi(t-1) = \frac{1}{c_t^{\gamma}} \underset{t \to +\infty}{\sim} \frac{1}{t^{\gamma}}$$

And the Riemann series $\sum_t \frac{1}{t^{\gamma}}$ diverges (since $\gamma < 1$), so we have :

$$\sum_{t=1}^{T} (\Psi(t) - \Psi(t-1)) \underset{t \to +\infty}{\sim} \sum_{t=1}^{T} \frac{1}{t^{\gamma}}$$

Hence, by telescoping sum :

$$\frac{T^{1-\gamma}}{1-\gamma} \underset{T \to +\infty}{\sim} \sum_{t=1}^{T} \frac{1}{t^{\gamma}}$$

Consequently :

$$\frac{c_2}{T} \sum_{t=1}^{T} \frac{1}{t^{\gamma}} \underset{T \to +\infty}{\sim} \frac{c_2}{(1-\gamma) T^{\gamma}}$$

And since :

$$\frac{1}{T} = o\left(\frac{1}{T^{\gamma}}\right)$$

We conclude the following equivalent to the bound $(\mathcal{E}_2)$:

$$(\mathcal{E}_2) \underset{T \to +\infty}{\sim} \frac{c_2}{(1-\gamma) T^{\gamma}} \tag{2.7.75}$$

From 2.7.71 and 2.7.75 we conclude the second part 2.7.39 of the theorem :

$$\boxed{\frac{1}{T} \sum_{t=1}^{T} \mathbb{E}\left[\left\| \nabla f\left(\boldsymbol{\theta}_t\right) \right\|_2^2 \right] = O\left(\frac{1}{T^{\gamma}}\right)}$$

■