Machine Learning and Finance
Solution to the Final Exam - Session 1 - (2 hours)

The exam is marked out 100

The exam is composed of three independent problems:

- **Credit Risk Prediction** (40 marks)
- **Building Context-Based Embedding Vectors** (35 marks)
- **A Sequential Neural Network** (25 marks)

1 Credit Risk Prediction [40 marks]

We wish to create a model to assess the quality of a loan. We build a machine learning algorithm that predicts how likely the loan will be paid based on several features. There are three different possible labels:

- **Category A** if the likelihood of paying the loan is very high.
- **Category B** if the likelihood is neither high nor low.
- **Category C** if the likelihood is very low.

Convention: The category A is mapped to the index 0, the category B is mapped to the index 1 and the category C is mapped to the index 2.

To train the models, we use a training input dataset composed of N samples, each sample is a vector of size $D = 20$.

Let X be the training tensor containing all the samples. X is then of shape (N, D). Let T be the tensor of the targets after the one hot encoding process.

As shown in the Figure 1, the model we want to use is a neural network composed of one input layer with D passthrough neurons, followed by one hidden layer with $M = 10$ neurons, and finally one output layer with $K = 3$ neurons.

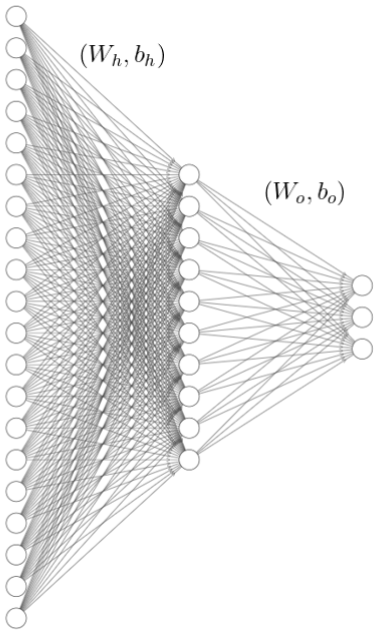


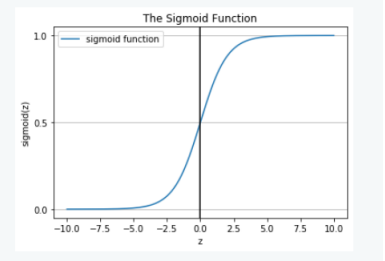
Figure 1: The Shallow Neural Network

We split the training input tensor X into several batches of size N_b. Let $\tilde{X}$ be the part of the training ïnput tensor containing the first N_b samples and $\tilde{T}$ the corresonding target tensor of shape (N_b, K).

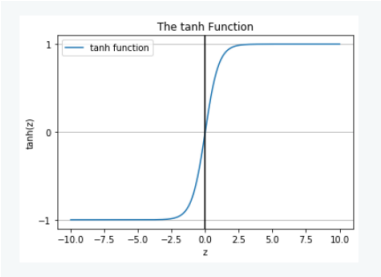
- **Q1: Name three popular activations functions and draw them.** *[3 marks]*

We can name for instance:

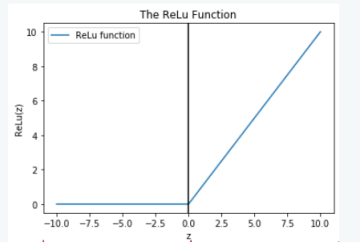
- The sigmoid function $z \mapsto \frac{1}{1+e^{-z}}$



- The tanh function $z \mapsto \frac{e^{2z}-1}{e^{2z}+1}$



- The ReLu function $z \mapsto \max(0, z)$



- **Q2: Which activation function would you use for the last layer. Justify your answer.** *[3 marks]*

As we are dealing with multiclass classification, we want the output of the neural network to be a distribution. Therefore, we should use the **softmax** activation.

- **Q3: What are the shapes of the hidden layer's weight vector W_h and its bias vector b_h?** *[3 marks]*

$$W_h \in \mathbb{R}^{D \times M}, \quad b_h \in \mathbb{R}^M$$

- **Q4: What are the shapes of the output layer's weight vector W_o and its bias vector b_o?** *[3 marks]*

$$W_o \in \mathbb{R}^{M \times K}, \quad b_o \in \mathbb{R}^K$$

- **Q5: What is the shape of the network's output matrix P if we perform the forward propagation on $\tilde{X}$** *[3 marks]*

$$P \in \mathbb{R}^{N_b \times K}$$

- **Q6: Write the equation that computes the network's output matrix P as a function of $\tilde{X}, W_h, b_h, W_o, b_o$.** *[4 marks]*

Let σ_h be the activation function used for the hidden layer and σ_o the one used for the output layer.

$$P = \sigma_o\left(\sigma_h(\tilde{X}W_h + b_h)W_o + b_o\right)$$

- **Q7: Write the loss function associated with this classification problem for the batch $\tilde{X}$ as a function of $P, \tilde{T}$** *[4 marks]*

Let $\mathcal{L}$ be the loss function for the batch $\tilde{X}$. We have then:

$$\mathcal{L} = -\frac{1}{N_b}\sum_{i=1}^{N_b}\sum_{k=1}^{K}\tilde{T}_{ik}\log(P_{ik})$$

- **Q8: What is backpropagation and how does it work?** *[5 marks]*

Backpropagation is a technique used to train the neural network. It consists in calculating the gradients of the loss function with regard to every model paramater (weights and biases of the hidden layer and the ouput layer in this case) using the chain rule. Then it performs a Gradient Descent step using these gradients.

- **Q9: List all the hyperparameters you can tweak in this model?** *[6 marks]*

Here is a list of the hyperparameters we can tweak in this model:
- The number of neurons in each hidden layer
- The number of hidden layers
- The activation functions
- The learning rate in the Gradient Descent.
- The batch size and the number of epochs.

- **Q10: If the model overfits the training data, how could you solve the problem? (List three methods)** *[6 marks]*

We can use several techniques to reduce the overfitting such as:
- **Weight regularization** which consists in adding to the loss function a cost associated with having large weights.
- **Dropout** which consists in randomly dropping out a number of output features during the training. The dropout rate will be a new hyperparameter.
- **Reducing the complexity** by decreasing the number of neurons for instance in the hidden layer.

2 Building Context-Based Embedding Vectors [35 marks]

2.1 A Context-free embedding model

2.1.1 The Word2vec/GloVe models

Q11: Describe the process of getting the embedding vectors using one of the two following models: Word2vec or GloVe. Make sure to specify the following elements: *[6 marks]*

- **How to prepare the dataset from a large corpus**
- **How to train the model**
- **How to extract the trained embedding vectors**

The GloVe approach.

- From the corpus, we get the matrix of co-occurence using the following algorithm:

Algorithm 1 Getting the co-occurence matrix

Input: sequences (list of lists of integers), context_size
Output: X (the co-occurence matrix)

```
Initialize the matrix X ∈ M_{V,V}(R) with zeros.
for sequence in sequences do
    for word w[i] of index i in sequence do
        for word w[j] of index j in the context of w[i] do
            X[w[i], w[j]] ⟵ X[w[i], w[j]] + 1
        end for
    end for
end for
```

- Let $\log X$ be the logarithm of the co-occurence matrix. The model consists in factorizing $\log X$ as follows:

$$\forall (i,j) \in \{1,\dots,V\}^2 \quad \log X_{ij} \approx W_i^T \tilde{W}_j + b_i + \tilde{b}_j$$

  The parameters of the regression model are:
  - A first **embedding matrix** and a bias term associated with it:

$$W = \begin{pmatrix} - & W_1 & - \\ \vdots & \vdots & \vdots \\ - & W_V & - \end{pmatrix} \in \mathcal{M}_{V,D}(\mathbb{R}), \quad b = \begin{pmatrix} b_1 \\ \vdots \\ b_V \end{pmatrix} \in \mathbb{R}^V$$

  - A second **embedding matrix** and a bias term associated with it:

$$\tilde{W} = \begin{pmatrix} - & \tilde{W}_1 & - \\ \vdots & \vdots & \vdots \\ - & \tilde{W}_V & - \end{pmatrix} \in \mathcal{M}_{V,D}(\mathbb{R}), \quad \tilde{b} = \begin{pmatrix} \tilde{b}_1 \\ \vdots \\ \tilde{b}_V \end{pmatrix} \in \mathbb{R}^V$$

- Instead of equal-weighting all the co-occurences, we introduce a **weighting function** $f(X_{ij})$ defined as follows:

$$\forall x \in \mathbb{R}_+ \quad f(x) = \begin{cases} (x/x_{\max})^\alpha & \text{if } x < x_{\max} \\ 1 & \text{otherwise} \end{cases}$$

- The **cost function** can then be written as follows:

$$J = \sum_{i=1}^{V} \sum_{j=1}^{V} f(X_{ij})(\log X_{ij} - W_i^T \tilde{W}_j - b_i - \tilde{b}_j)^2$$

- To train the algorithm, we can use one of the two options:

- Alternating least squares: By setting the gradients to zero, we get the following update equations:

$$\nabla_{W_i} J(W_i) = 0 \iff W_i = \left(\sum_{j'=1}^{V} f(X_{ij'}) \tilde{W}_{j'} \tilde{W}_{j'}^T \right)^{-1} \left(\sum_{j'=1}^{V} f(X_{ij'})(\log X_{ij'} - b_i - \tilde{b}_{j'}) \tilde{W}_{j'} \right)$$

$$\nabla_{\tilde{W}_j} J(\tilde{W}_j) = 0 \iff \tilde{W}_j = \left(\sum_{i'=1}^{V} f(X_{i'j}) W_{i'} W_{i'}^T \right)^{-1} \left(\sum_{i'=1}^{V} f(X_{i'j})(\log X_{i'j} - b_{i'} - \tilde{b}_j) W_{i'} \right)$$

$$\nabla_{b_i} J(b_i) = 0 \iff b_i = \left(\sum_{j'=1}^{V} f(X_{ij'}) \right)^{-1} \left(\sum_{j'=1}^{V} f(X_{ij'})(\log X_{ij'} - W_i^T \tilde{W}_{j'} - \tilde{b}_{j'}) \right)$$

$$\nabla_{\tilde{b}_j} J(\tilde{b}_j) = 0 \iff \tilde{b}_j = \left(\sum_{i'=1}^{V} f(X_{i'j}) \right)^{-1} \left(\sum_{i'=1}^{V} f(X_{i'j})(\log X_{i'j} - W_{i'}^T \tilde{W}_j - b_{i'}) \right)$$

- By using the Gradient Descent algorithm:

Algorithm 4 Training using gradient descent

Input: $\log X, f(X), \eta, N_{\text{epochs}}$
Output: $W^{(N_{\text{epochs}}-1)}, \tilde{W}^{(N_{\text{epochs}}-1)}, b^{(N_{\text{epochs}}-1)}, \tilde{b}^{(N_{\text{epochs}}-1)}$ (The trained paramaters)

Initialize randomly the parameters $W^{(0)}, \tilde{W}^{(0)}, b^{(0)}, \tilde{b}^{(0)}$
costs = [ ]
for $t \leftarrow 0, \ldots, N_{\text{epochs}} - 2$ **do**
    Calculate the cost as a function of $W^{(t)}, \tilde{W}^{(t)}, b^{(t)}, \tilde{b}^{(t)}$ and append the list costs
    for $i \leftarrow 0, \ldots, V-1$ **do**
        $W_i^{(t+1)} \longleftarrow W_i^{(t)} - \eta \nabla_{W_i} J(W_i^{(t)})$
    end for
    for $j \leftarrow 0, \ldots, V-1$ **do**
        $\tilde{W}_j^{(t+1)} \longleftarrow \tilde{W}_j^{(t)} - \eta \nabla_{W_j} J(\tilde{W}_j^{(t)})$
    end for
    for $i \leftarrow 0, \ldots, V-1$ **do**
        $b_i^{(t+1)} \longleftarrow b_i^{(t)} - \eta \nabla_{b_i} J(b_i^{(t)})$
    end for
    for $j \leftarrow 0, \ldots, V-1$ **do**
        $\tilde{b}_j^{(t+1)} \longleftarrow \tilde{b}_j^{(t)} - \eta \nabla_{\tilde{b}_j} J(\tilde{b}_j^{(t)})$
    end for
end for

- Once the model is trained, we can use the matrix W or $\tilde{W}$ as an embedding matrix.

2.1.2 Using pre-trained embedding vectors in a classification problem

Consider the problem of predicting the next word using the architecture in Figure 2.

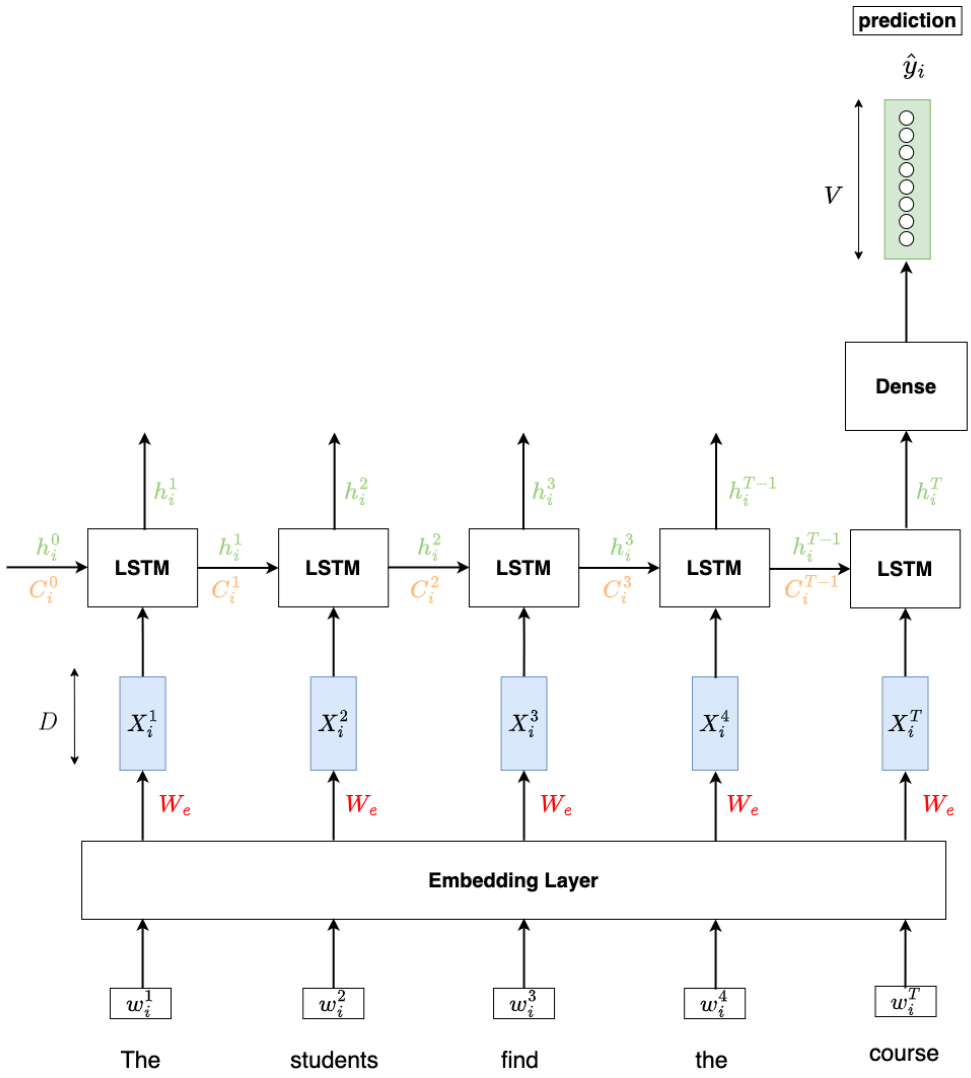


Figure 2: Predicting the next word

Let V be the vocabulary size. We would like to map the sequence of tokens $(w_i^1, \ldots, w_i^T)$ associated with the sentence "The students find the course" to the next word:

- We use an embedding layer, parameterized by the matrix W_e, which gives the sequence of embedding vectors $(X_i^1, \ldots, X_i^T)$ of dimension D.
- We use an LSTM layer with hidden states $(h_i^t, C_i^t)_{1 \leq t \leq T}$ of size M.
- The last hidden state h_i^T is then mapped to the prediction vector $\hat{y}_i \in \mathbb{R}^V$ using a Dense layer parameterized by (W_d, b_d).
- The prediction vector $\hat{y}_i$ is then compared to the true prediction $\tilde{y}_i \in \{0, 1\}^V$

Q12: List all the parameters of the architecture *[6 marks]*

The Layer	The Parameters	The number of parameters
The Embedding Layer	$W_e \in \mathbb{R}^{V \times D}$	$V \times D$
The LSTM Layer	$(W_k, b_k) \in \mathbb{R}^{(D+M) \times M} \times \mathbb{R}^M$ for $k \in \{f, i, o, C\}$	$4\left((D+M)M + M\right)$
The Dense Layer	$(W_d, b_d) \in \mathbb{R}^{(M \times V)} \times \mathbb{R}^V$	$MV + V$

Q13: Choose reasonable values of V, D, M. What would be the total number of trainable parameters if we let the model learn the embedding matrix ? *[4 marks]*

Let's take $V = 10000$, $D = 100$ and $M = 16$. As the embedding matrix is trainable, the total number of parameters is: $V * D + 4\left((D+M)M + M\right) + MV + V = 1000000 + 7488 + 170000 = 1177488$

Q14: With the same hyperparameters, what would be the total number of trainable parameters if we use pre-trained embedding vectors *[4 marks]*

As the embedding matrix is frozen, the embedding layer is not trainable.
Therefore, the total number of parameters is: $4\left((D+M)M+M\right)+MV+V = 7488+170000 = 177488$

2.1.3 Limitations of the context-free embedding models

Consider the following two sentences:

- Sentence A: "**Python** is a famous programming language"
- Sentence B: "**Python** is one of the largest snake species"

Q15: Based on the embedding of the word "Python" in both sentences, explain why using a context-free embedding model such as Word2vec or GloVe is suboptimal to represent the meaning of the word "Python". *[4 marks]*

By reading the preceding two sentences, we can understand that the meaning of the word 'Python' is different in both sentences. In sentence A, the word 'Python' refers to the programming language while in sentence B,the word 'Python' refers to the snake.
If we get embeddings for the word 'Python' in the preceding two sentences using an embedding model such as word2vec or GloVe, the embedding of the word 'Python' would be the same in both sentences, and so this renders the meaning of the word 'Python' the same in both sentences. This is because Word2vec/ GloVe are context-free models. So, they will ignore the context and always give the same embedding for the word 'Python' irrespective of the context.

2.2 The Scaled Dot Product Attention Layer

We would like to create context-based representations of the embedding vectors $(X^1, \ldots, X^T)$ using a self attention layer, as shown in figure 3

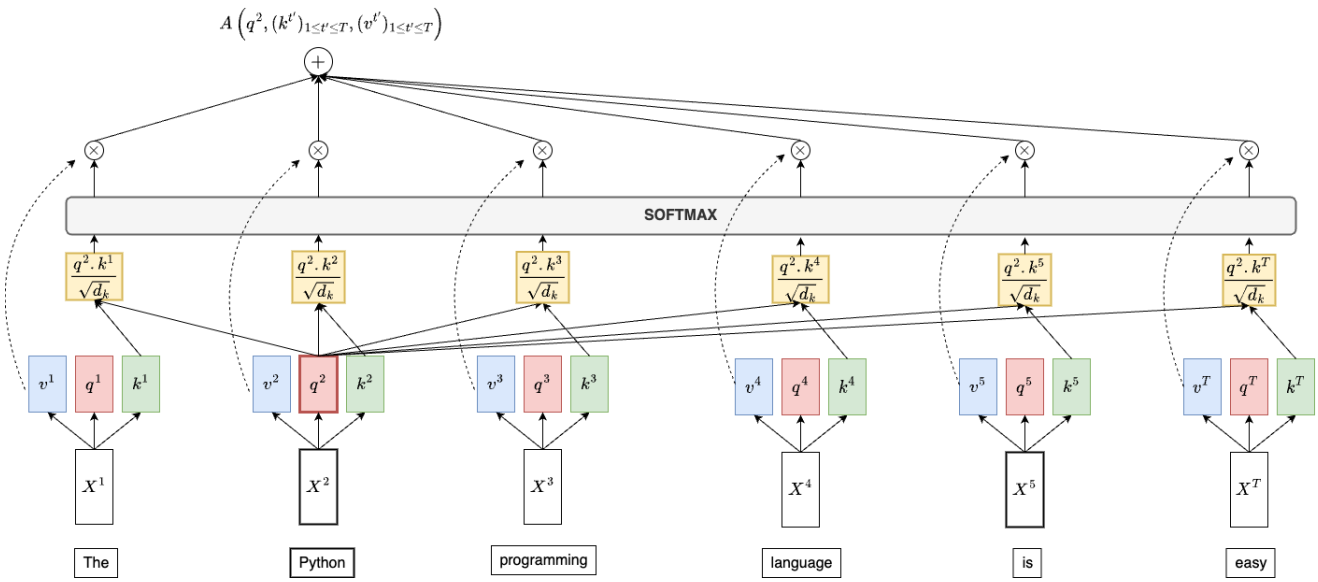


Figure 3: The Self Attention Layer

For all $t \in \{1, \ldots, T\}$, 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 as follows:

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

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 $A\left(q^2, (k^{t'})_{1\leq t'\leq T}, (v^{t'})_{1\leq t'\leq T}\right)$ be the context-based representation of the embedding vector X^2.

Q16: What is the expression of $A\left(q^2, (k^{t'})_{1\leq t'\leq T}, (v^{t'})_{1\leq t'\leq T}\right)$? *[4 marks]*

$$A\left(q^2, (k^{t'})_{1\leq t'\leq T}, (v^{t'})_{1\leq t'\leq T}\right) = \sum_{t'=1}^{T} \frac{\exp(\frac{q^2.k^{t'}}{\sqrt{d_k}})}{\sum_{t''=1}^{T}\exp(\frac{q^2.k^{t''}}{\sqrt{d_k}})} v^{t'}$$

Q17: Does the context-based representation $A\left(q^2, (k^{t'})_{1\leq t'\leq T}, (v^{t'})_{1\leq t'\leq T}\right)$ depend on the order of the tokens in the sentence "The Python programming language is easy ? *[2 marks]*

No, a permutation of the tokens will result in the same context-based representation.

We can generalize the way to create the context-based embedding $A\left(q^2, (k^{t'})_{1\leq t'\leq T}, (v^{t'})_{1\leq t'\leq T}\right)$ associated with the embedding X^2 to all the embedding vectors $(X^{t'})_{1\leq t'\leq T}$.

We consider the following matrices:

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

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.

Q18: Show that: *[5 marks]*

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

Notations:

$$\forall t, t' \in \{1, \dots, T\}\ e^{<t,t'>} = \frac{q^t.k^{t'}}{\sqrt{d_k}} \quad \text{and} \quad \alpha^{<t,t'>} = \frac{\exp(e^{<t,t'>})}{\sum_{t''=1}^{T} \exp(e^{<t,t''>})}$$

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 context-based embedding vector $A\left(q^t, (k^{t'})_{1\leq t'\leq T}, (v^{t'})_{1\leq t'\leq T}\right)$ associated with the embedding vector X^t.

3 A Sequential Neural Network [25 marks]

In this section, we are dealing with a long sequence $X_1, X_2, \ldots, X_T$ of T continuous observations in $\mathbb{R}$. We wish to create a sequential neural network to predict the next observation based on the previous τ observations. Obviously $\tau < T$

- **Q19: How would you derive the training and validation data (both features and targets) from the long sequence $X_1, \ldots, X_T$? *[5 marks]***

  – From the long sequence $X_1, \ldots, X_T$ we can derive the following $T - \tau$ sequences of length τ and the corresponding targets:

  $$\begin{array}{llll} \text{Input:} & X_1, \ldots X_\tau & \text{Target:} & X_{\tau+1} \\ \text{Input:} & X_2, \ldots X_{\tau+1} & \text{Target:} & X_{\tau+2} \\ \vdots & \vdots & \vdots & \vdots \\ \text{Input:} & X_{T-\tau}, \ldots X_{T-1} & \text{Target:} & X_T \end{array}$$

  Therefore, we obtain a tensor of inputs of shape $(T - \tau, \tau)$ and a tensor of targets of shape $(T - \tau, 1)$. We can then split these two tensors into training and validation tensors by taking the first 70% sequences as training data and the last 30% as testing data.

- **Q20: What are the main difficulties when training Recurrent Neural Networks? How can you handle them?*[5 marks]***

  Recurrent Neural Networks (RNNs) aren't capable of learning "long term dependencies" due to the vanishing gradient problem. We usually prefer to use a specific type of RNNs called the Long Short Term Memory (LSTM) networks

- **Q21: Describe the model you would use by specifying how the shape of the data is changing after each layer transformation. *[7 marks]***

  Let N_t be the number of sequences in the training data and N_v be the number of sequences in the testing set. Obviously $N_v + N_t = T - \tau$.
  We want to use an LSTM and a Dense layer.
  – First, the input tensor of shape $(N_t, \tau, 1)$ is fed into the LSTM layer. We specify d the dimension of the output vectors and we only keep the last one as it contains the information of the whole sequence.
  – Hence the $(N_t, \tau, 1)$ tensor will be transformed into an (N_t, d) tensor.
  – We use a Dense layer with only one output neuron and no activation function (since it's a regression problem).
  – So, the (N_t, d) will be transformed into an $(N_t, 1)$ tensor.

- **Q22: What would be your loss function? *[3 marks]***

  As we are dealing with a regression problem, we should use **the mean squared error** as a loss function.

- **Q23: Describe the algorithm you would use for training your neural network. *[5 marks]***

  For training the algorithm, we can use Stochastic Gradient Descent. We first split the input data into several batches of size M. For each batch, we calculate the loss after the forward propagation. We then deduce the gradients corresponding to each parameter of the neural network. Lastly, we perform one step of Gradient Descent. We repeat the process several times (corresponding to the number of epochs N_{epochs}).
  The pseudo code:
  – We initialize randomly the paramaters of the network: θ_0
  – We repeat N_{epochs} times:
    * Splitting the input data into batches.
    * For each batch:
      · We Calculate the loss $\mathcal{L}_{\text{batch}}$ for the specific batch
      · We Update the weights using a gradient descent step:

  $$\theta_{k+1} \leftarrow \theta_k - \eta \nabla_\theta \mathcal{L}_{\text{batch}}(\theta_k)$$