Name: **Zahra Soukhtedel** Assignment Number: **3**
Course: **Deep Learning** Date: May 2, 2023

Problem 1

1.1

In an LSTM, we use gate weights to control the flow of information into and out of the cell state. The input gate decides which values from the input and previous hidden state should be added to the cell state. The forget gate decides which values in the current cell state should be discarded, and the output gate decides which values from the updated cell state should be included in the output.
To compute the cell state c_t and hidden state h_t at each time step t, we first compute the activations of the input gate i_t, forget gate f_t, and output gate o_t as:

$$i_t = \sigma(w_{xi}x_t + w_{hi}h_{t-1} + b_i)$$

$$f_t = \sigma(w_{xf}x_t + w_{hf}h_{t-1} + b_f)$$

$$o_t = \sigma(w_{xo}x_t + w_{ho}h_{t-1} + b_o)$$

where x_t is the input at time step t, h_{t-1} is the previous hidden state, and b_i, b_f, and b_o are bias terms.

$$g_t = \tanh(w_{xg}x_t + w_{hg}h_{t-1} + b_g)$$

Where g_t is the candidate update. Then, we update the cell state by combining the candidate update with the old cell state c_{t-1}, weighted by the forget gate:

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$

Finally, we compute the hidden state by applying the output gate to the updated cell state:

$$h_t = o_t * \tanh(c_t)$$

1.2

Here are the correct calculations for the gradients of the LSTM cell parameters:
1. w_{xi}, w_{xg}, b_i, w_{hi}, w_g, b_g (input gate weights):

$$\frac{\partial E}{\partial w_{xi}} = E_{delta} \cdot i_t \cdot (1 - i_t) \cdot g_t \cdot \frac{\partial \tanh(c_t)}{\partial w_{xi}}$$

$$\frac{\partial E}{\partial w_{xg}} = E_{delta} \cdot g_t \cdot (1 - \tilde{c}_t^2) \cdot i_t \cdot (1 - i_t) \cdot \frac{\partial \tanh(c_t)}{\partial w_{xg}}$$

$$\frac{\partial E}{\partial b_i} = E_{delta} \cdot i_t \cdot (1 - i_t) \cdot g_t \cdot \frac{\partial \tanh(c_t)}{\partial b_i}$$

$$\frac{\partial E}{\partial w_{hi}} = E_{delta} \cdot i_t \cdot (1 - i_t) \cdot g_t \cdot \frac{\partial \tanh(c_t)}{\partial w_{hi}}$$

$$\frac{\partial E}{\partial w_g} = E_{delta} \cdot i_t \cdot (1 - i_t) \cdot g_t \cdot \frac{\partial \tanh(c_t)}{\partial w_g}$$

$$\frac{\partial E}{\partial b_g} = E_{delta} \cdot i_t \cdot (1 - i_t) \cdot g_t \cdot \frac{\partial \tanh(c_t)}{\partial b_g}$$

2. w_{xf}, b_f, w_{hf} (forget gate weights):

$$\frac{\partial E}{\partial w_{xf}} = E_{delta} \cdot f_t \cdot (1 - f_t) \cdot c_{t-1} \cdot \frac{\partial \tanh(c_t)}{\partial w_{xf}}$$

$$\frac{\partial E}{\partial b_f} = E_{delta} \cdot f_t \cdot (1 - f_t) \cdot c_{t-1} \cdot \frac{\partial \tanh(c_t)}{\partial b_f}$$

$$\frac{\partial E}{\partial w_{hf}} = E_{delta} \cdot f_t \cdot (1 - f_t) \cdot c_{t-1} \cdot \frac{\partial \tanh(c_t)}{\partial w_{hf}}$$

3. w_{xo}, b_o, w_{ho} (output gate weights):

$$\frac{\partial E}{\partial w_{xo}} = E_{delta} \cdot o_t \cdot (1 - o_t) \cdot \tanh(c_t) \cdot \frac{\partial h_t}{\partial w_{xo}}$$

$$\frac{\partial E}{\partial b_o} = E_{delta} \cdot o_t \cdot (1 - o_t) \cdot \tanh(c_t) \cdot \frac{\partial h_t}{\partial b_o}$$

$$\frac{\partial E}{\partial w_{ho}} = E_{delta} \cdot o_t \cdot (1 - o_t) \cdot \tanh(c_t) \cdot \frac{\partial h_t}{\partial w_{ho}}$$

Note that the derivative of the hyperbolic tangent function $\tanh(x)$ with respect to its input x is $1 - \tanh^2(x)$. Therefore, we can simplify the above equations as follows: 1. w_{xi}, w_{xg}, b_i, w_{hi}, w_g, b_g (input gate weights):

$$\frac{\partial E}{\partial w_{xi}} = E_{delta} \cdot i_t \cdot (1 - i_t) \cdot g_t \cdot (1 - \tanh^2(c_t)) \cdot x_t$$

$$\frac{\partial E}{\partial w_{xg}} = E_{delta} \cdot i_t \cdot (1 - i_t) \cdot g_t \cdot (1 - \tilde{c}_t^2) \cdot h_{t-1}$$

$$\frac{\partial E}{\partial b_i} = E_{delta} \cdot i_t \cdot (1 - i_t) \cdot g_t \cdot (1 - \tanh^2(c_t))$$

$$\frac{\partial E}{\partial w_{hi}} = E_{delta} \cdot i_t \cdot (1 - i_t) \cdot g_t \cdot (1 - \tanh^2(c_t)) \cdot h_{t-1}$$

$$\frac{\partial E}{\partial w_g} = E_{delta} \cdot i_t \cdot (1 - i_t) \cdot g_t \cdot (1 - \tanh^2(c_t)) \cdot h_{t-1}$$

$$\frac{\partial E}{\partial b_g} = E_{delta} \cdot i_t \cdot (1 - i_t) \cdot g_t \cdot (1 - \tanh^2(c_t))$$

2. w_{xf}, b_f, w_{hf} (forget gate weights):

$$\frac{\partial E}{\partial w_{xf}} = E_{delta} \cdot f_t \cdot (1 - f_t) \cdot c_{t-1} \cdot (1 - \tanh^2(c_t)) \cdot x_t$$

$$\frac{\partial E}{\partial b_f} = E_{delta} \cdot f_t \cdot (1 - f_t) \cdot c_{t-1} \cdot (1 - \tanh^2(c_t))$$

$$\frac{\partial E}{\partial w_{hf}} = E_{delta} \cdot f_t \cdot (1 - f_t) \cdot c_{t-1} \cdot (1 - \tanh^2(c_t)) \cdot h_{t-1}$$

3. w_{xo}, b_o, w_{ho} (output gate weights):

$$\frac{\partial E}{\partial w_{xo}} = E_{delta} \cdot o_t \cdot (1 - o_t) \cdot \tanh(c_t) \cdot h_t$$

$$\frac{\partial E}{\partial b_o} = E_{delta} \cdot o_t \cdot (1 - o_t) \cdot \tanh(c_t)$$

$$\frac{\partial E}{\partial w_{ho}} = E_{delta} \cdot o_t \cdot (1 - o_t) \cdot \tanh(c_t) \cdot h_{t-1}$$

These equations give us the gradients for each of the parameters of the LSTM cell.

Problem 2

In the Scaled Dot-Product Attention mechanism, the scaling factor is used to prevent the dot products from becoming too large, which could lead to vanishing gradients during backpropagation. This leads to having more stable gradients.The scaling factor is the square root of the dimension of the key/query vectors (d_k). We suspect that for large values of d_k, the dot products grow large in magnitude, pushing the softmax function into regions where it has extremely small gradients. To counteract this effect, we scale the dot products byd_k.
We can compute the expected value and variance of the dot product based on assumptions. we have:

$$\text{Expected value:} \quad E[q{\cdot}k] = E\left[\sum_i (q_i * k_i)\right] = \sum_i (E[q_i]{*}E[k_i]) = 0, \quad \text{since} \quad E[q_i] = E[k_i] = 0.$$

$$\text{Variance:} \quad \mathrm{Var}(q \cdot k) = E[(q \cdot k)^2] - (E[q \cdot k])^2 = E[(q \cdot k)^2], \quad \text{since} \quad E[q \cdot k] = 0.$$

$$E[(q \cdot k)^2] = E\left[\sum_i (q_i * k_i)^2\right] = \sum_i E[(q_i * k_i)^2].$$

$$\text{Since} \quad q_i \quad \text{and} \quad k_i \quad \text{are independent,} \quad E[(q_i * k_i)^2] = E[q_i^2] * E[k_i^2] = 1 * 1 = 1.$$

$$\text{So,} \quad E[(q \cdot k)^2] = \sum_i E[(q_i * k_i)^2] = d_k.$$

$$\Rightarrow E[q \cdot k] = 0, \quad \mathrm{Var}(q \cdot k) = d_k$$

The variance of the dot product is d_k, which means that the dot product's magnitude grows with the dimensionality of the vectors. To counteract this effect, we scale the dot product by dividing it by the square root of d_k:

$$\text{Scaled dot product:} \quad \frac{q \cdot k}{\sqrt{dk}}$$

This scaling factor helps maintain the dot products' magnitudes within a reasonable range, preventing vanishing gradients and improving the stability of the attention mechanism during training.

Problem 3

3.1

3.1.1

α can be interpreted as a categorical probability distribution because they represent the relative importance of each input token when generating the output token. The attention mechanism computes a score for each input token, each score is between 0 and 1 and can be interpreted as a probability, then passed through a softmax function. The softmax function normalizes the scores into a probability distribution, ensuring that the sum of all attention weights equals 1.

3.1.2

This can happen when the attention mechanism identifies a highly informative input token that has a significant impact on the overall output of the model.(when the input sequence contains one or more tokens that are highly informative and contain key information that is relevant to the task at hand.) Or when the key values k_j compared to other key values are large then the dot product between the key and the query will be large.

3.1.3

α_j value will have the most weight thus c will be similar to vj. If the attention weights are diffuse, the output c may be a weighted combination of all the input elements, with no single input element playing a dominant role in determining the output. This can lead to a more balanced and nuanced output that takes into account the contributions of multiple input elements.

3.1.4

If dot product between some j^{th} word's key and a query is very large compared to others, means that α_j will have the most weights, thus c will be similar to one value v_j and something like copying v_j would happen.

3.2

3.2.1

suppose that A is a matrix of concatenated basis vectors $\{a_1, ..., a_m\}$, And assume that there exist some $c_1, c_2, ..., c_m$ and $d_1, d2, ..., d_p$ such that $v_a = \sum_{i=1}^{m} c_i a_i$ and $v_b = \sum_{i=1}^{p} d_i b_i$. (based on assumptions on basis vectors being orthogonal, it can easily concluded that these coefficients exists and their values are as follows $c_i =< v_a \cdot a_i >, d_i =< v_b \cdot b_i >$).
Now we show that for $M = A^T$ we can extract v_a from the sum vector, we will show that Ms is equal to C(a vector in $\mathbb{R}^d$ contains coefficients c_i), so we can compute v_a from C the result of Ms. Lets calculate Ms, based on assumptions of a_i, b_i being orthogonal and normalized we have:

$$Ms = M(v_a + v_b) = Mv_a + Mv_b, \quad M = A^T$$

$$\Rightarrow Ms = A^T v_a + A^T v_b = \begin{bmatrix} {a_1}^T \\ {a_2}^T \\ \cdot \\ \cdot \\ \cdot \\ {a_m}^T \end{bmatrix} v_a + \begin{bmatrix} {a_1}^T \\ {a_2}^T \\ \cdot \\ \cdot \\ \cdot \\ {a_m}^T \end{bmatrix} v_b = \begin{bmatrix} < a_1 \cdot v_a > \\ < a_2 \cdot v_a > \\ \cdot \\ \cdot \\ \cdot \\ < a_m \cdot v_a > \end{bmatrix} + \begin{bmatrix} < a_1 \cdot v_b > \\ < a_2 \cdot v_b > \\ \cdot \\ \cdot \\ \cdot \\ < a_m \cdot v_b > \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \\ \cdot \\ \cdot \\ \cdot \\ c_m \end{bmatrix} + 0 = C$$

$$\Rightarrow Ms = C$$

And we know that in terms of R^d (not in terms of A and B), v_a is just a collection of constants C. Thus M = A^T

3.2.2

Assuming c to be equal to $c = \frac{1}{2}(v_a + v_b)$, and we know that $c = \sum_{j=1}^{n} \alpha_i v_i$, so we must have $\alpha_a \approx \alpha_b >> \alpha_i (1 \leq i \leq n, i \neq a, b)$. Therefor we have that:

$$ {k_a}^T q \approx {k_b}^T q >> {k_i}^T q $$

From previous question, if the dot product is big, the probability mass will also be big and we want a balanced mass between α_a and α_b. q will be largest for k_a and k_b when it is a large multiplicative of a vector that contains a component in k_a direction and in k_b direction, so lets express q as follows:

$$ q = c(k_a + k_b), \quad c >> 0 $$

We will show that this expression works, first based on assumption of k_i's being orthogonal with their norm equal to 1 we have:

$$ \forall 1 \leq i \leq n, i \neq a, b : \quad {k_i}^T q = {k_i}^T c(k_a + k_b) = 0 $$

$$ {k_a}^T q = {k_a}^T c(k_a + k_b) = c, \quad {k_b}^T q = c $$

To calculate attention weights we have that $exp({k_i}^T q) = exp(0)$ will be insignificant to the probability mass and $exp({k_a}^T q) = exp({k_b}^T q) = exp(c)$, so we have:

$$ \alpha_a = \alpha_b = \frac{exp(c)}{\sum_{j=1}^{n} exp({k_j}^T q)} = \frac{exp(c)}{(n-2) * 1 + 2 * exp(c)} \approx \frac{1}{2} $$

3.3

3.3.1

Due to the small variances (values on the diagonal of covariance matrices) for all $i \in \{1, 2, ..., n\}$, we can infer that each key vector is in proximity to its mean vector, implying $k_i \approx \mu_i$. As the mean vectors are orthogonal, the problem can be simplified to the scenario where all keys were orthogonal. Thus, we can express q as:

$$ q = r(\mu_a + \mu_b), \quad r >> 0 $$

3.3.2

Assuming that ${\mu_i}^T \mu_i = 1$, we can observe that the value of k_a fluctuates between $(\alpha_+ 0.5)\mu_a$ and $(\alpha_+ 1.5)\mu_a$. However, k_a does not vary considerably except when $i \neq a$, and α is exceedingly small. We can estimate both k_a and k_i as:

$$k_a \approx \beta\mu_a, \quad where \quad \beta \simeq N(1, 0.5), \quad k_i \approx \beta\mu_i, \quad whenever \quad i \neq a$$

As all key vectors are orthogonal, and the direction of q aligns with directions k_a and k_b, we assume that the dot product between q and any other key vector is 0. There are two cases to be considered:

$${k_a}^T q \approx \beta{\mu_a}^T r(\mu_a + \mu_b) \approx r\beta, \quad {k_b}^T q \approx \beta{\mu_b}^T r(\mu_a + \mu_b) \approx r, \quad where \quad r >> 0$$

We can solve for coefficients α_a and α_b, and observe that for large values of r, exp(0) is insignificant, resulting in:

$$\alpha_a \approx softmax(r\beta) \approx \frac{1}{1 + exp(r(1-\beta))}, \quad \alpha_b \approx softmax(r) \approx \frac{1}{1 + exp(r(\beta - 1))}$$

As β ranges from 0.5 to 1.5, and $r >> 0$, we can observe that:

$$(\alpha_a \approx 0, \quad \alpha_b \approx 1, \quad when \quad \beta = 0.5), \quad (\alpha_a \approx 1, \quad \alpha_b \approx 0, \quad when \quad \beta = 1.5)$$

In conclusion, the output oscillates between v_a and v_b, where $c = \alpha_a v_a + \alpha_b v_b$ for large values of r. Any other terms are insignificant.

$$(c \approx v_b \quad when \quad \beta \to 0.5), \quad (c \approx v_a \quad when \quad \beta \to 1.5)$$

3.4

3.4.1

Assuming the same conditions as previously stated, and because all keys are similar to their averages and in accordance with the information given in question we can create two queries, q1 and q2, with one copying v_a and the other copying v_b. We represent the queries as $q_1 = r\mu_a, q_2 = r\mu_b$ for $r >> 0$, so we have that $c_1 \approx v_a$ and $c_2 \approx v_b$. Considering $c = \frac{1}{2}(c_1 + c_2)$, we have that $c \approx \frac{1}{2}(v_a + v_b)$.

3.4.2

Based on what we had in section 3.3.2, we would have that other dot products expect a, b, will be insignificant when r is large, and for a, b we have that:

$${k_a}^T q_1 = \beta{\mu_a}^T r\mu_a = r\beta, \quad {k_b}^T q_2 \approx \beta{\mu_b}^T rmu_b \approx r, \quad where \quad r >> 0.$$

For attention weights we have: $\alpha_{a1} \approx softmax(r\beta) \approx 1, \quad \alpha_{b2} = softmax(r) \approx 1$. And other attention weights would be around zero, so we can say that $c_1 \approx v_a, c_2 \approx v_b$, thus we have:

$$c \approx \frac{1}{2}(v_a + v_b)$$

Problem 4

The input image size can pose challenges in Vision Transformer (ViT) and MLPMixer models due to fixed-size patch embedding and the architecture's dependence on specific input dimensions. These models split the input image into non-overlapping patches, which are then embedded into a fixed-size vector. The subsequent layers work with these fixed-size vectors, so altering the input image size may cause a discrepancy in the expected input dimensions for the model.
To resolve this issue, there are various solutions available. One solution is to use an adaptive input size approach, where the model is created to handle different input sizes using convolutional layers or other techniques that can adapt to varying input sizes. Another solution is resizing the input image to match the expected size for the model using various image resizing techniques; however, this approach may result in information loss or introduce artifacts in the resized image. Alternatively, instead of resizing the entire image, you can resize just the extracted patches from the input image to maintain the aspect ratio of the original image while still allowing the model to process the input. Finally, data augmentation techniques such as random cropping, flipping, and rotation can be used to create multiple versions of the input image with different sizes, helping the model learn to generalize better and handle varying input sizes during fine-tuning.
The ViT and MLPMixer models employ several methods, including patch embedding, where the input image is divided into non-overlapping patches, and each patch is linearly embedded into a fixed-size vector. In the ViT model, patch embeddings are processed by a series of transformer layers designed to capture both local and global contextual information. In contrast, in the MLPMixer model, patch embeddings are processed through a sequence of MLP layers designed to capture local and global contextual information via token-mixing and channel-mixing layers. Ultimately, the final output of the model is processed by a classification head, usually a linear layer followed by a softmax activation function to produce class probabilities. It is crucial to adjust the learning rate and other hyperparameters accordingly when fine-tuning these models on larger image sizes to achieve optimal performance.

Problem 5

First, let's consider the matrix $E_{t,:}$ for a given position t. This matrix has d_{model} rows and 1 column, and each row is defined as a concatenation of sine and cosine functions with different frequencies. The frequencies are determined by the formula $\frac{1}{\lambda_m} = 10000^{\frac{2m}{d_{model}}}$, where m is the row index.
Now, let's consider the matrix $E_{t+k,:}$ for a position deviation of k. This matrix is also a concatenation of sine and cosine functions with different frequencies, but shifted by k positions compared to $E_{t,:}$.
To find the linear transformation $T^{(k)}$ that maps $E_{t,:}$ to $E_{t+k,:}$, we can use the fact that the sine and cosine functions are periodic with period 2π. Therefore, we can express $E_{t+k,:}$ as a linear combination of the rows of $E_{t,:}$, where each row is shifted by a certain phase angle. Specifically,

we can write:

$$E_{t+k,:} = \begin{bmatrix} sin(\frac{t+k}{f_1}) \\ cos(\frac{t+k}{f_1}) \\ sin(\frac{t+k}{f_2}) \\ cos(\frac{t+k}{f_2}) \\ . \\ . \\ . \\ sin(\frac{t+k}{f_{\frac{d_{model}}{2}}}) \\ cos(\frac{t+k}{f_{\frac{d_{model}}{2}}}) \end{bmatrix} = \sum_{m=1}^{d_{model}/2} \begin{bmatrix} sin(\frac{km}{f_1}) \\ cos(\frac{km}{f_1}) \\ sin(\frac{km}{f_2}) \\ cos(\frac{km}{f_2}) \\ . \\ . \\ . \\ sin(\frac{km}{f_{\frac{d_{model}}{2}}}) \\ cos(\frac{km}{f_{\frac{d_{model}}{2}}}) \end{bmatrix} \begin{bmatrix} sin(\frac{t}{f_m}) \\ cos(\frac{t}{f_m}) \end{bmatrix}$$

We can express this equation in matrix form as:

$$E_{t+k,:} = \sum_{m=1}^{d_{model}/2} A_{k,m} E_{t,:}$$

where $A_{k,m}$ is a 2×2 rotation matrix that depends on the frequency f_m and the position deviation k. Specifically, we have:

$$A_{k,m} = \begin{bmatrix} cos(\frac{km}{f_m}) & -sin(\frac{km}{f_m}) \\ sin(\frac{km}{f_m}) & cos(\frac{km}{f_m}) \end{bmatrix}$$

Therefore, the linear transformation $T^{(k)}$ that maps $E_{t,:}$ to $E_{t+k,:}$ is given by the matrix $T^{(k)} = \sum_{m=1}^{d_{model}/2} A_{k,m}$. This matrix is a $d_{model} \times d_{model}$ matrix that depends on the position deviation k, and it satisfies the equation $T^{(k)} E_{t,:} = E_{t+k,:}$ for any valid position $t \in \{1, ..., n-k\}$ in the sequence.

another answer:

To show that there exists a linear transformation $T^{(k)}$ such that $T^{(k)} E_{t,:} = E_{t+k,:}$, we need to find the matrix $T^{(k)}$ that satisfies this equation.

Let's first consider $k = 1$, i.e., we want to find the transformation matrix $T^{(1)}$ that maps each vector $E_{t,:}$ to the subsequent vector $E_{t+1,:}$. Let's denote by T_m the m-th row of $T^{(1)}$ and let's express $E_{t+1,:}$ in terms of $E_{t,:}$:

$$E_{t+1,:} = \begin{bmatrix} sin(\frac{t+1}{f_1}) \\ cos(\frac{t+1}{f_1}) \\ sin(\frac{t+1}{f_2}) \\ cos(\frac{t+1}{f_2}) \\ \vdots \\ sin(\frac{t+1}{f_{d_{model}/2}}) \\ cos(\frac{t+1}{f_{d_{model}/2}}) \end{bmatrix} = \begin{bmatrix} sin(\frac{t}{f_1} + \frac{1}{f_1}) \\ cos(\frac{t}{f_1} + \frac{1}{f_1}) \\ sin(\frac{t}{f_2} + \frac{1}{f_2}) \\ cos(\frac{t}{f_2} + \frac{1}{f_2}) \\ \vdots \\ sin(\frac{t}{f_{d_{model}/2}} + \frac{1}{f_{d_{model}/2}}) \\ cos(\frac{t}{f_{d_{model}/2}} + \frac{1}{f_{d_{model}/2}}) \end{bmatrix}$$

$$= \begin{bmatrix} sin(\frac{t}{f_1}) & cos(\frac{t}{f_1}) & 0 & 0 & \cdots & 0 & 0 \\ -cos(\frac{t}{f_1})\frac{1}{f_1} & sin(\frac{t}{f_1})\frac{1}{f_1} & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & sin(\frac{t}{f_2}) & cos(\frac{t}{f_2}) & \cdots & 0 & 0 \\ 0 & 0 & -cos(\frac{t}{f_2})\frac{1}{f_2} & sin(\frac{t}{f_2})\frac{1}{f_2} & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & sin(\frac{t}{f_{d_{model}/2}}) & cos(\frac{t}{f_{d_{model}/2}}) \\ 0 & 0 & 0 & 0 & \cdots & -cos(\frac{t}{f_{d_{model}/2}})\frac{1}{f_{d_{model}/2}} & sin(\frac{t}{f_{d_{model}/2}})\frac{1}{f_{d_{model}/2}} \end{bmatrix}$$

$$* \begin{bmatrix} sin(\frac{t}{f_1}) \\ cos(\frac{t}{f_1}) \\ sin(\frac{t}{f_2}) \\ cos(\frac{t}{f_2}) \\ \vdots \\ sin(\frac{t}{f_{d_{model}/2}}) \\ cos(\frac{t}{f_{d_{model}/2}}) \end{bmatrix} = TE_{t,:},$$

where T is the $d_{model} \times d_{model}$ matrix given above.
Now, we can extend this to any position deviation k by noting that we need to shift the input sequence by k positions to get the output sequence. Therefore, we can use the transformation matrix $T^{(k)} = T^k$, which applies the linear transformation T repeatedly k times. That is,

$$T^{(k)}E_{t,:} = T^kE_{t,:} = \underbrace{T \cdots T}_{k \text{ times}} E_{t,:} = E_{t+k,:},$$

where the last equality follows from the fact that each application of T shifts the frequency components of the input vector by one position in the sequence.
Therefore, we have shown that there exists a linear transformation matrix $T^{(k)}$ that maps $E_{t,:}$ to $E_{t+k,:}$ for any valid position t and any position deviation k.