Lab 5: Time Series and Transformers

Javier Porras-Valenzuela and Alejandro Ribeiro*

June 2, 2025

1 Time Series

We define a time series $\mathbf{X}$ as a collection of $T+1$ vectors $\mathbf{x}_t \in \mathbb{R}^n$ indexed by a time index $t = 0, 1, \dots T$. There are several tasks that we may want to perform in a time series, but the prototypical example is the prediction of the entry $\mathbf{x}_T$ at time T when given the history of the series between times 0 and $T-1$,

$$\mathbf{X}_T = \mathbf{x}_{0:T-1} = \big[\, \mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_{T-1} \,\big]. \tag{1}$$

This task is illustrated in Figure 1 for $T = 10$. The time series is observed between times $t = 0$ and $t = T-1 = 9$. The value at time $T = 10$ is unobserved. Our goal is to predict it.

This is a goal that we can formulate as a machine learning task. Given the history of the time series between times 0 and $T-1$, we introduce a learning parameterization $\mathcal{H}$ to produce estimates of the time series at time T,

$$\hat{\mathbf{x}}_T = \Phi\big(\, \mathbf{X}_T, \, \mathcal{H} \,\big). \tag{2}$$

These estimates can be compared to the true value of the time series $\mathbf{x}_T$ to formulate a training cost that we then optimize to find the optimal set of parameters. That is, we go through the usual steps of: (i) Acquiring data for several time series. This yields a set of U histories $\mathbf{X}_u$ and corresponding time T values $\mathbf{x}_{uT}$. (ii) Introducing a loss function $\ell(\hat{\mathbf{x}}_T, \mathbf{x}_T)$

*In alphabetical order.

Figure 1. Time Series Prediction. A time series is a collection of $T+1$ vectors $\mathbf{x}_t \in \mathbb{R}^n$ indexed by a time index t. The prototypical task in time series is the prediction of the entry at time T when given the history of the series between times $t = 0$ and $t = T-1$. In the figure, $T = 10$ and we want to predict the unobserved value of $\mathbf{x}_T = \mathbf{x}_{10}$ based on the observed values of $\mathbf{x}_{0:T-1} = \mathbf{x}_{0:9}$.

measuring the fit between the time series value $\mathbf{x}_T$ and its prediction $\hat{\mathbf{x}}_T$. (iii) Formulating the empirical risk minimization (ERM) problem,

$$\mathcal{H}^* \;=\; \operatorname*{argmin}_{\mathcal{H}} \; \frac{1}{U} \sum_{u=0}^{U-1} \ell\Big(\Phi\big(\mathbf{X}_{uT},\, \mathcal{H} \big),\, \mathbf{x}_{uT} \Big). \tag{3}$$

In (3), the index u denotes several different time series. This is not quite how time series work. In reality, we are given a single time series that extends for $T+U$ units of time and the "different" time series are actually different windows of the same time series,

$$\mathbf{X}_{uT} = \mathbf{x}_{u:u+T-1} = \big[\, \mathbf{x}_u, \mathbf{x}_{u+1}, \ldots, \mathbf{x}_{u+T-1} \,\big], \qquad \mathbf{x}_{uT} = \mathbf{x}_{u+T}. \tag{4}$$

Thus, out of a single time series we extract a number of training samples that consider time u as the starting point of a new sequence of length T out of which we want to predict the value of the sequence at time $u+T$. Our first task is to construct the dataset in (4) when given a time series.

Task 1 In this lab we work with weather data. We are given a time series with $T + U = 52,696$ entries each of which has various descriptors of the weather at different times of different days. The entries in the time series are twelve weather indicators such as humidity, atmospheric pressure, and temperature.

Load the data from the lab's page and plot component "T (degC)" of the time series as a function of time. This is the average temperature during each time interval.

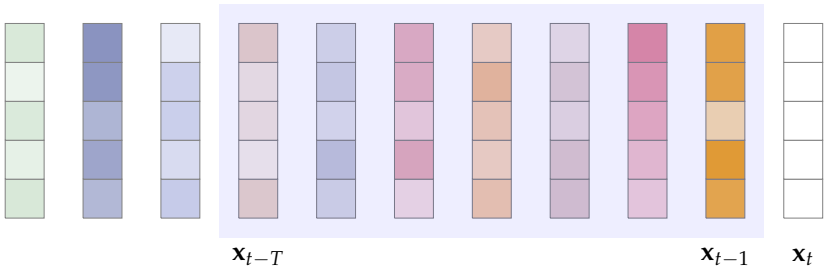

Figure 2. Time Series Prediction. It is more common to define time series as sets of vectors $\mathbf{x}_t \in \mathbb{R}^n$ that extend to infinity. In this context the prediction problem illustrated in Figure 1 morphs into the problem of estimating $\mathbf{x}_t$ given a window of T previous values $\mathbf{X}_t = \mathbf{x}_{t-T:t-1}$. They are mathematically equivalent problems.

Separate this time series into two parts. The first part contains 70% of the values and the second part contains the remaining 30%. Use these two time series to extract $U = 36,787$ samples of the form in (4) for a training set and to extract $U = 15,709$ samples of the form in (4) for a test set. In both cases, use $T = 100$.

1.1 A More Precise Definition of Time Series

We begun Section 1 describing time series as a collection of $T + 1$ vectors $\mathbf{x}_t \in \mathbb{R}^n$. A more common definition of a time series is that of a set of vectors $\mathbf{x}_t \in \mathbb{R}^n$ that extends from time $t = 0$ to infinity. At any point in time t our goal is to predict the value of $\mathbf{x}_t$ given the *whole* process's history $\mathbf{x}_{0:t-1}$. In practice, values in the distant past are considered irrelevant for the estimation of $\mathbf{x}_t$. We therefore introduce a window of length T and consider the history of the time series starting at time $t - T$. Formally, we define the windowed history

$$\mathbf{X}_t = \mathbf{x}_{t-T:t-1} = \big[\, \mathbf{x}_{t-T}, \mathbf{x}_{t-T+1}, \ldots, \mathbf{x}_{t-1} \,\big], \tag{5}$$

and consider a learning parameterization that maps $\mathbf{X}_t$ to predictions

$$\hat{\mathbf{x}}_t = \Phi\big(\, \mathbf{X}_t,\, \mathcal{H} \,\big). \tag{6}$$

This is an equivalent description of the history and parameterization in (1) and (2). It is just that instead of starting at time $t = 0$ to predict at time T as in (1) and (2) we start at arbitrary time t to predict at time $t + T$.

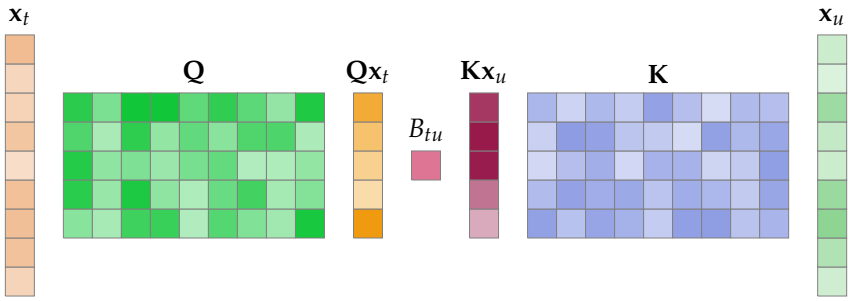

Figure 3. Attention. For vectors $\mathbf{x}_t$ and $\mathbf{x}_u$ we compute attention coefficient B_{tu} to measure their similarity. To compute these similarities the vector $\mathbf{x}_t$ is multiplied by the *query* matrix $\mathbf{Q}$ and the vector $\mathbf{x}_u$ is multiplied by the *key* matrix $\mathbf{K}$. The attention coefficient B_{tu} is the inner products between queries $\mathbf{Q}\mathbf{x}_t$ and keys $\mathbf{K}\mathbf{x}_u$.

This more accurate description of a time series is important during execution. The trained model $\Phi(\mathbf{X}, \mathcal{H}^*)$ is executed on a rolling basis. At any time t we make predictions by executing the model $\Phi(\mathbf{X}_t, \mathcal{H}^*)$ with the history window $\mathbf{X}_t$ as defined in (5). After observing $\mathbf{x}_t$ – at which point the problem of predicting $\mathbf{x}_t$ becomes moot – we advance time to $t+1$, update the history window and execute the model $\Phi(\mathbf{X}_{t+1}, \mathcal{H}^*)$ to make a prediction of the value of the time series at time $t+1$.

Henceforth, we work with the definition of a time series as a sequence of T vectors $\mathbf{X} = \mathbf{x}_{0:T-1}$ with the goal of predicting $\mathbf{x}_T$. There are less indexes involved and the notation is less cumbersome. But we keep in mind that out trained models are to be executed on a rolling basis on an indefinite time series.

2 Attention Layers

Attention layers create representations of the entries $\mathbf{x}_t$ of a time series $\mathbf{X} = [\mathbf{x}_0, \ldots, \mathbf{x}_T]$ that depend on context. This is done by constructing vectors $\mathbf{y}_t$ that are linear combinations of all of the entries of the time series weighted by importance coefficients. I.e., for a certain matrix $\mathbf{M}$

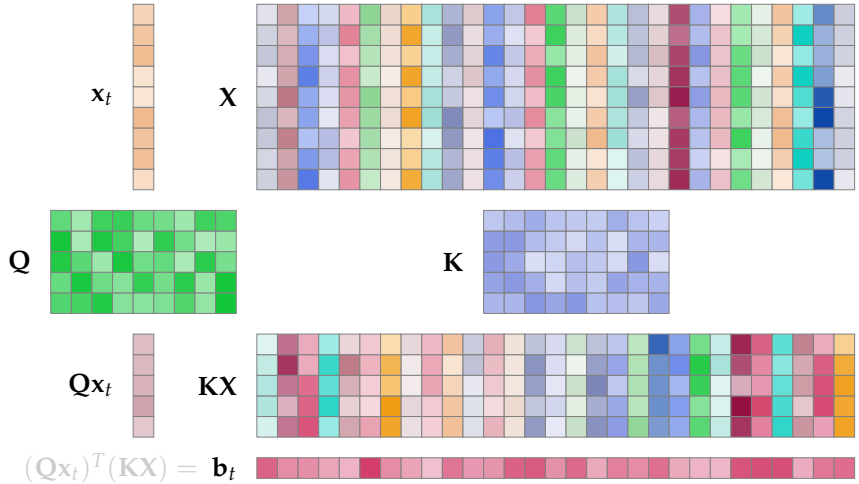

Figure 4. Attention vectors. For a given vector $\mathbf{x}_t$, we compute a vector of attention coefficients $\mathbf{b}_t$. These coefficients measure the similarity between $\mathbf{x}_t$ and all other vectors in the time series. To compute these similarities the vector $\mathbf{x}_t$ is multiplied by the *query* matrix $\mathbf{Q}$ and the series $\mathbf{X}$ is multiplied by the *key* matrix $\mathbf{K}$. The attention vector $\mathbf{b}_t$ is the inner product between the query $\mathbf{Q}\mathbf{x}_t$ and the keys $\mathbf{K}\mathbf{x}_u$.

and similarity function $\mathrm{d}(\cdot,\cdot)$, we compute the vector

$$\mathbf{y}_t \;=\; \sum_{u=0}^{T} \mathrm{d}(\mathbf{x}_t, \mathbf{x}_u)\mathbf{M}\mathbf{x}_u. \tag{7}$$

The collection of vectors $\mathbf{y}_t$ forms another time series $\mathbf{Y} = [\mathbf{y}_0, \ldots, \mathbf{y}_T]$. The construction of the time series $\mathbf{Y}$ is such that its entries $\mathbf{y}_t$ depend on all other entries of the time series. For this reason we call it a contextual representation. The purpose of the importance coefficients $\mathrm{d}(\mathbf{x}_t, \mathbf{x}_u)$ is for the representation $\mathbf{y}_t$ to be most affected by the time series vectors $\mathbf{x}_u$ that are deemed most relevant to $\mathbf{x}_t$.

The importance coefficients $\mathrm{d}(\mathbf{x}_t, \mathbf{x}_u)$ are called attention coefficients.

2.1 Attention Coefficients

To accomplish this we rely on the attention coefficients

$$B_{tu} \;=\; \langle \mathbf{Q}\mathbf{x}_t, \mathbf{K}\mathbf{x}_u \rangle \;=\; (\mathbf{Q}\mathbf{x}_t)^T (\mathbf{K}\mathbf{x}_u). \tag{8}$$

Attention is just a way of measuring the similarity between the components $\mathbf{x}_t$ and $\mathbf{x}_u$ of the time series. Indeed, if we make $\mathbf{Q} = \mathbf{K} = \mathbf{I}$ the attention coefficient reduces to the inner product between the time series's components, $B_{tu} = \langle \mathbf{x}_t, \mathbf{x}_u \rangle$. This inner product is a standard measure of similarity between vectors.

The incorporation of $\mathbf{Q}$ and $\mathbf{K}$ in (8) introduces learnable coefficients that may yield more relevant measures of similarity. The matrices $\mathbf{Q}$ and $\mathbf{K}$ have n columns – which is the number of entries of each of the time series vectors $\mathbf{x}_t$ – and m rows. In general, $m \ll n$ because we know that inner products are more meaningful in low dimensional spaces.

The coefficients B_{tu} can be arranged into *row* vectors $\mathbf{b}_t$ that include all of the attention coefficients associated with time t. It follows from (8) and the definition of the time series matrix $\mathbf{X} = [\mathbf{x}_0, \ldots, \mathbf{x}_T]$ that this vector of attention coefficients can be computed as

$$\mathbf{b}_t \;=\; (\mathbf{Q}\mathbf{x}_t)^T (\mathbf{K}\mathbf{X}). \tag{9}$$

The computation of the attention vector $\mathbf{b}_t$ is represented in Figure 4. We begin with the time series represented in its matrix form $\mathbf{X}$ and isolate a specific time index t. This is the vector $\mathbf{x}_t$ in Figure 4. To compute attention coefficients we multiply $\mathbf{x}_t$ by the *query* matrix $\mathbf{Q}$. This multiplication yields the query vectors $\mathbf{Q}\mathbf{x}_t$ for this particular component of the time series. In parallel, each of the vectors $\mathbf{x}_u$ of the time series is multiplied by the *key* matrix $\mathbf{K}$. This results in the calculation of the key vectors $\mathbf{K}\mathbf{x}_u$ which are the columns of the key matrix $\mathbf{K}\mathbf{X}$. Although not required, the number of rows of the query and key vectors are (much) smaller than the number of rows of the time series. The attention coefficients in (8) are the result of computing the inner product between the query vector and the key matrix.

The attention coefficients in (8) can be further grouped into an attention matrix $\mathbf{B}$. Operating from the definition of the attention vectors in (9) we can see that this matrix is given by

$$\mathbf{B} \;=\; (\mathbf{Q}\mathbf{X})^T (\mathbf{K}\mathbf{X}). \tag{10}$$

This computation is illustrated in Figure 5. The time series matrix $\mathbf{X}$ is multiplied by the query matrix $\mathbf{Q}$ and the key matrix $\mathbf{K}$. These multiplications result in the computation of the queries $\mathbf{QX}$ and the keys $\mathbf{KX}$. The attention matrix $\mathbf{B}$ is the outer product $(\mathbf{QX})^T(\mathbf{KX})$ between queries and keys.

Notice that there are a large number of attention coefficients but they are generated by a relatively small number of parameters. Indeed, there are at total of $(T+1)^2$ attention coefficients when we operate with a time series with $T+1$ vectors. However, the query and key matrices have $m \times n$ coefficients.

As is the case with convolutions in time, graphs, and images, the matrix expression in (10) is the one that we use for implementations. The scalar and vector expressions in (8) and (9) are valuable to understand attention but not used in implementations.

Task 2 Implement a Pytorch module for the similarity operation in (10). The query and key matrices are attributes of this class. The forward method should compute the attention matrix $\mathbf{B}$.

2.2 Nonlinear Attention

The similarity coefficients in (8) are what we call a *linear* attention mechanism. Nonlinear attention mechanisms post process linear attention coefficients with a nonlinear function.

The most common choice of nonlinearity is a function we call a softmax. For a given vector $\mathbf{b} \in \mathbb{R}^{T+1}$ the softmax is the vector, $\mathbf{a} = \text{sm}(\mathbf{b})$ with components,

$$a_u = \frac{\exp(b_u)}{\sum_{u'=1}^{T+1} \exp(b_{u'})} \quad \Leftrightarrow \quad \text{sm}(\mathbf{a}) = \frac{\exp(\mathbf{b})}{\mathbf{1}^T \exp(\mathbf{b})}, \tag{11}$$

where in the second equality we define the vector of exponentials $\exp(\mathbf{b}) := [\exp(b_0); \ldots; \exp(b_T)$ and the vector of all ones $\mathbf{1} := [1; \ldots; 1]$. As per (11), the softmax entry a_u is the ratio between the exponential $\exp(b_u)$ of Component u of the vector $\mathbf{b}$ normalized to the sum of the exponentials $\exp(b_{u'})$ of all components of $\mathbf{b}$. We point out that the definition is simi-

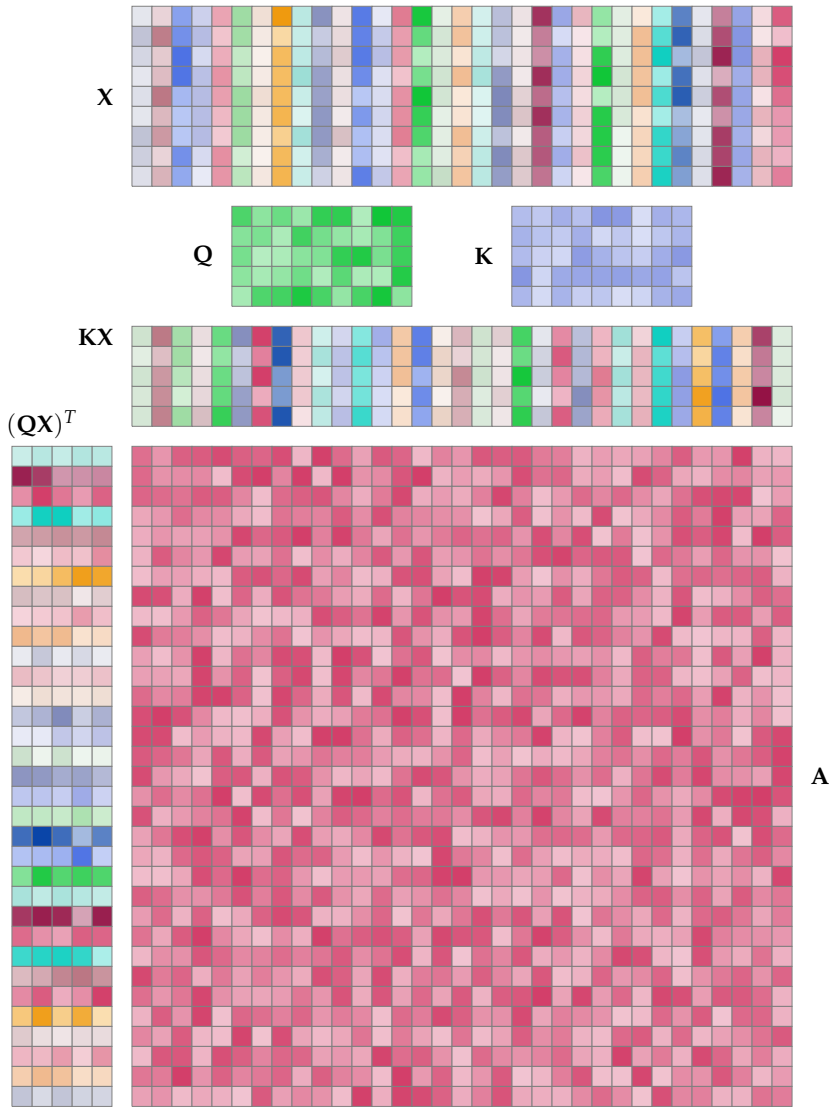

Figure 5. Attention Matrix. The time series matrix **X** is multiplied by the query matrix **Q** and the key matrix **K**. These multiplications result in the computation of the queries **QX** and the keys **KX**. The attention matrix **B** is the outer product $(\mathbf{QX})^T(\mathbf{KX})$ between queries and keys. There are a large number of attention coefficients but they are generated by a small number of parameters.

lar but not identical to the definition of the softmax function we used to introduce the cross entropy loss In Lab 2C.

With this definition we can now define softmax similarity coefficients as the application of the softmax function in (11) to the linear similarity vector in (9).

$$\mathbf{a}_t \;=\; \text{sm}\Big(\,\mathbf{b}_t\,\Big) \;=\; \text{sm}\Big(\,(\mathbf{Q}\mathbf{x}_t)^T(\mathbf{K}\mathbf{x}_u)\,\Big) \;=\; \frac{\exp\Big(\,(\mathbf{Q}\mathbf{x}_t)^T(\mathbf{K}\mathbf{X})\,\Big)}{\mathbf{1}^T\exp\Big(\,(\mathbf{Q}\mathbf{x}_t)^T(\mathbf{K}\mathbf{X})\,\Big)}\,. \tag{12}$$

Observe that the definition of the softmax in (11) is such that the sum of the entries of the softmax vector is normalized to, $\mathbf{1}^T\text{sm}(\mathbf{a}) = 1$. As a particular case, the sum of the similarity coefficients $\mathbf{a}_t$ in (12) is $\mathbf{1}^T\mathbf{a}_t = 1$. We can then think of the softmax similarity coefficients in (12) as a nonlinear normalization of the attention coefficients in (9).

This observation is important because it makes it plain that (12) is similar to a pointwise nonlinearity. Indeed, if we use A_{tu} to denote the entries of $\mathbf{a}_t$, it follows from the definitions in (8), (9) and (12) that

$$A_{tu} \;=\; \frac{\exp(B_{tu})}{\sum_{u'=1}^{T+1}\exp(B_{tu})} \tag{13}$$

Thus, the similarity coefficient A_{tu} is obtained by applying an exponential pointwise nonlinearity to the linear similarity coefficient B_{tu} followed by a normalization. The purpose of the exponential nonlinearity is to magnify the difference between different similarity coefficients.

Similarly to (10), we can group all attention coefficients into a matrix

$$\mathbf{A} = \text{sm}\Big(\,(\mathbf{Q}\mathbf{X})^T(\mathbf{K}\mathbf{X})\,\Big), \tag{14}$$

where the softmax function implements normalizing along the rows of $(\mathbf{Q}\mathbf{X})^T(\mathbf{K}\mathbf{X})$. I.e., the rows of the similarity matrix $\mathbf{A}$ are the vectors $\mathbf{a}_t$ defined in (12).

Task 3 Implement a Pytorch module for the similarity operation in (14). The query and key matrices are attributes of this class. The forward method should compute the attention matrix $\mathbf{A}$.

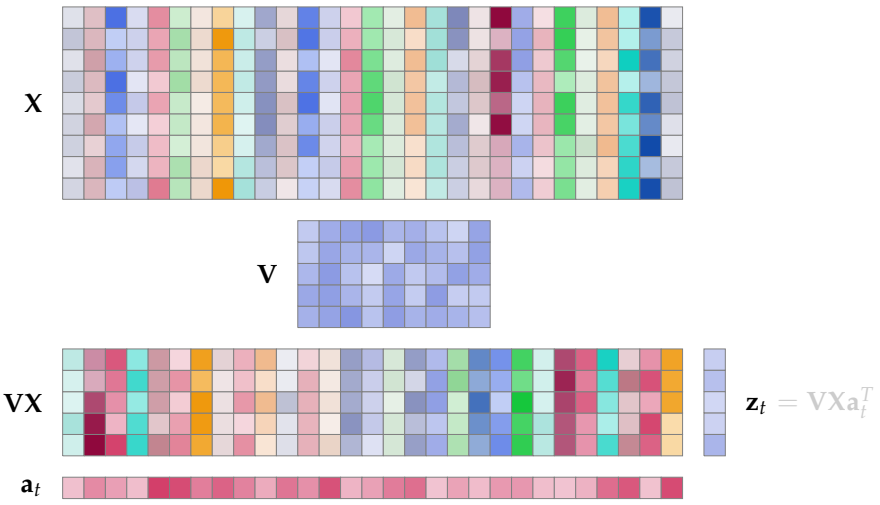

Figure 6. Contextual Representations of Reduced Dimension. Contextual representations are created by multiplying the time series matrix by a matrix $\mathbf{V}$ with m rows and n columns. Since $m \ll n$, this creates a representation $\mathbf{VX}$ of the time series in a lower dimensional space. Each of the vectors in $\mathbf{VX}$ is multiplied by the attention coefficient A_{tu} [cf. (12)-(14)]. The sum of the result over all u is the contextual representation $\mathbf{z}_t$ of dimension m.

2.3 Contextual Representations

The similarity coefficients in (12)-(14) are used to create a contextual representation of $\mathbf{x}_t$,

$$\mathbf{z}_t = \sum_{u=0}^{T} (\mathbf{V}\mathbf{x}_u) A_{tu} \tag{15}$$

This contextual representation is a linear combination of all the vectors in the time series multiplied by a matrix $\mathbf{V}$ and scaled by the similarity coefficients A_{tu}. The matrix $\mathbf{V} \in \mathbb{R}^{m \times n}$ is a projection in a lower dimensional space. The dimensions of $\mathbf{V}$ are the same as the dimensions of $\mathbf{Q}$ and $\mathbf{K}$.

It is ready to see that the the expression in (15) is equivalent to

$$\mathbf{z}_t = \mathbf{VX}\mathbf{a}_t^T. \tag{16}$$

This operation is explained in Figure 6. The time series $\mathbf{X}$ is multiplied

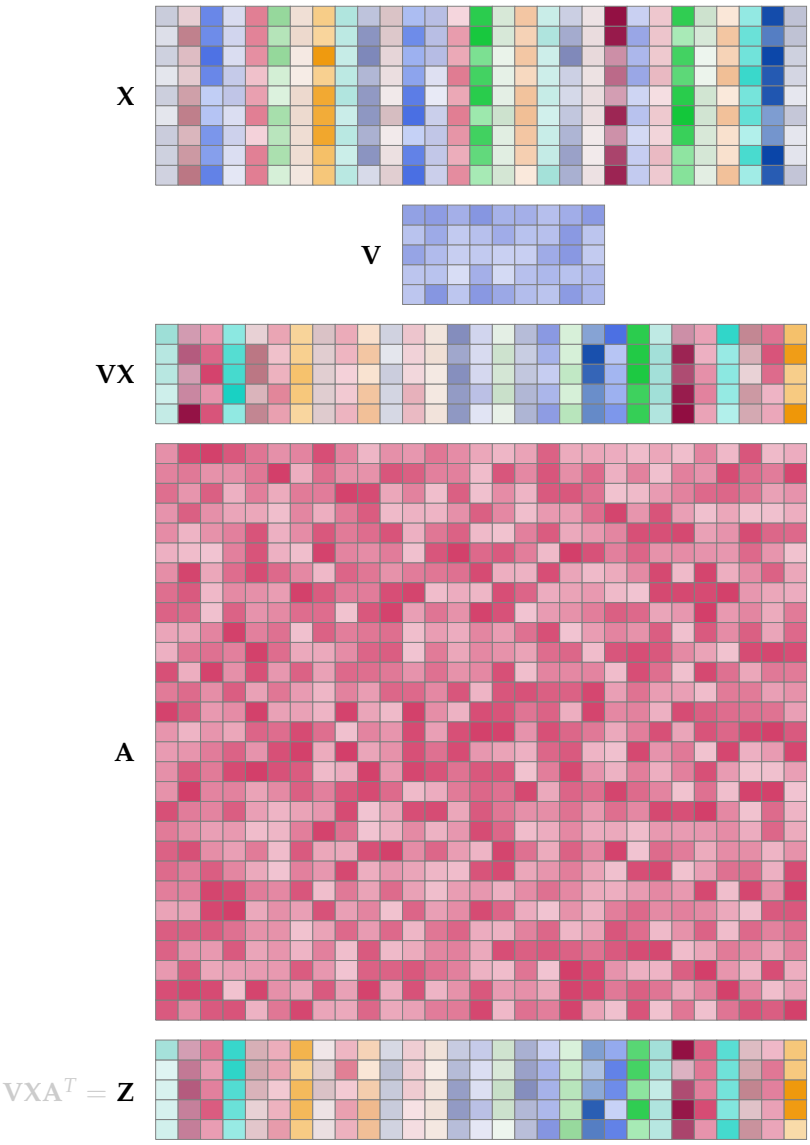

Figure 7. Contextual Representations of Reduced Dimension. Operations in Figure 6 shown in matrix form. The product **VX** represents the time series in a space of lower dimension $m \ll n$. Multiplication on the right with the attention matrix $\mathbf{A}^T$ produces the contextual representation of the time series, $\mathbf{Z} \in \mathbb{R}^m$.

by the value matrix $\mathbf{V}$. This produces an alternative representation of the time series given by the product $\mathbf{VX}$. This representation is of lower dimensionality. Instead of having vectors $\mathbf{x}_u \in \mathbb{R}^n$ associated with each point in time u we have vectors $\mathbf{V}\mathbf{x}_u \in \mathbb{R}^m$. We choose $m \ll n$. The low dimensional contextual representation $\mathbf{z}_t$ is obtained by weighting each vector $\mathbf{V}\mathbf{x}_u$ by the attention coefficient A_{tu} and summing over all times u. Equivalently, we obtain $\mathbf{z}_t$ as the product $\mathbf{VX}\mathbf{a}_t^T$ shown in (16). We say that this representation is contextual because $\mathbf{z}_t$ depends on vectors $\mathbf{x}_u$ that have deemed similar to $\mathbf{x}_t$ by the attention coefficient A_{tu}.

The operation in (16) can also be represented in matrix form. The matrix $\mathbf{Z}$ with columns $\mathbf{z}_t$ is given by

$$\mathbf{Z} = \mathbf{VXA}^T \tag{17}$$

This operation is represented in Figure 7. The top part of Figure 7 is the same as the top part of Figure 6. We are constructing a lower dimensional representation $\mathbf{VX}$ of the time series. We then produce the contextual representation $\mathbf{Z}$ by multiplying $\mathbf{VX}$ with the attention matrix transpose $\mathbf{A}^T$.

The representations in (17) are of dimension $m \ll n$. We complete an attention layer with a dimensional recovery step. This is done by multiplication with the transpose of a matrix $\mathbf{W} \in \mathbb{R}^{m \times n}$. We can write this operation in terms of individual contextual vectors $\mathbf{z}_t$,

$$\mathbf{y}_t \;=\; \mathbf{W}^T\mathbf{z}_t \;=\; \mathbf{W}^T\mathbf{VX}\mathbf{a}_t^T. \tag{18}$$

or in terms of the matrix $\mathbf{Z}$ with all of the contextual vectors

$$\mathbf{Y} \;=\; \mathbf{W}^T\mathbf{Z} \;=\; \mathbf{W}^T\mathbf{VXA}^T \tag{19}$$

The vectors $\mathbf{y}_t$ are contextual representations of the time series that have same dimensionality as the components of the time series. The operations in (18) and (19) are illustrated in Figures 8 and 9.

The representation $\mathbf{Y}$ is the output of an attention layer.

2.4 Softmax Attention Layers

As it follows from the discussions in Sections 2.2 and 2.3 a softmax attention layer has two distinct operations. The first operation is the computa-

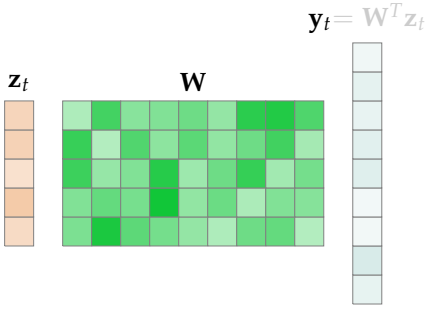

Figure 8. Contextual Representations of Original Dimension. We recover representations with the same initial dimension by multiplying the representation of reduced dimensionality by the transposed matrix $\mathbf{W}^T$. The vector $\mathbf{y}_t$ is the output of the attention layer. It is a linear combination of all the vectors in the time series weighted by their relevance to time t entry [cf. (18)].

tion of the softmax attention coefficients,

$$\mathbf{A} \;=\; \mathrm{sm}\Big(\,(\mathbf{QX})^T(\mathbf{KX})\,\Big). \tag{20}$$

This is Equation (14) repeated here for reference. The second operation is the computation of the contextual representation,

$$\mathbf{Y} \;=\; \mathbf{W}^T\mathbf{V}\mathbf{X}\,\mathbf{A}^T \;=\; \mathbf{W}^T\mathbf{V}\mathbf{X}\,\Big[\,\mathrm{sm}\Big(\,(\mathbf{Q}\,\mathbf{X})^T(\mathbf{K}\,\mathbf{X})\,\Big)\,\Big]^T. \tag{21}$$

This is Equation (19) repeated here for reference.

The parameters of the attention layer are the matrices $\mathbf{Q}$, $\mathbf{K}$, $\mathbf{V}$ and $\mathbf{W}$. All of these matrices have m rows and n columns with $m \ll n$. Having intermediate representations of smaller dimension is important.

The expressions in (14) and (21) are what you should use to implement and analyze attention layers. However, it is sometimes instructive to keep in mind the definition of the attention vectors

$$\mathbf{a}_t \;=\; \mathrm{sm}\Big(\,(\mathbf{Q}\mathbf{x}_t)^T(\mathbf{K}\mathbf{x}_u)\,\Big), \tag{22}$$

and the expanded expression for the computation of the contextual representation

$$\mathbf{y}_t \;=\; \mathbf{W}^T\sum_{u=0}^{T} A_{tu}\mathbf{V}\mathbf{x}_u. \tag{23}$$

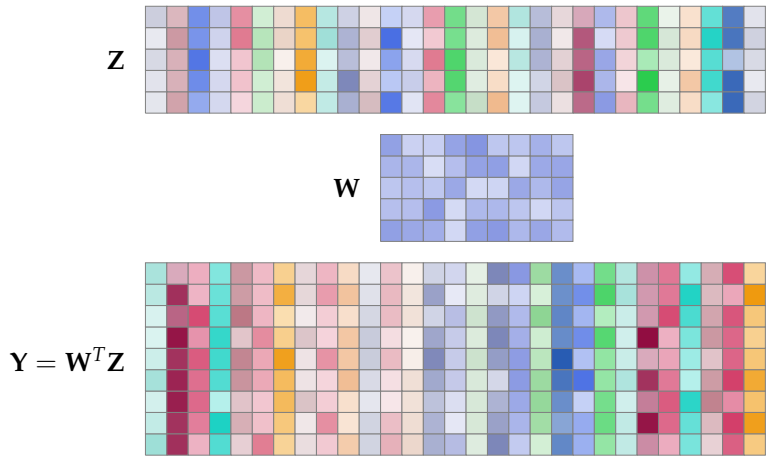

Figure 9. Contextual Representations of Original Dimension. Operations in Figure 8 shown in matrix form. The reduced dimensionality contextual representation $\mathbf{Z}$ is multiplied by the transposed matrix $\mathbf{W}^T$ [cf. (19)]. This is the output of the attention layer.

Equation (22) is a repetition of (12) and equation (23) is a combination of (18) and (15). Notice that the expression in (23) has the same form of the conceptual expression in (7) with $\mathrm{d}(\mathbf{x}_t, \mathbf{x}_u) = A_{tu}$ and $\mathbf{M} = \mathbf{W}^T\mathbf{V}$.

Task 4 Code a Pytorch module that implements an attention layer. The matrices $\mathbf{Q}$, $\mathbf{K}$, $\mathbf{V}$ and $\mathbf{W}$ are parameters of this module. The forward method of this module takes a time series $\mathbf{X}$ as an input and produces the time series $\mathbf{Y}$ as an output. The module receives m and n as initialization parameters.

It is common to postprocess the contextual representation further, but to do so without further mixing of different time components. The simplest we can do is add a pointwise nonlinearity to (21) so that the output is

$$\mathbf{Y}_\sigma \;=\; \sigma\Big(\,\mathbf{Y}\,\Big) \;=\; \sigma\Big(\,\mathbf{W}^T\mathbf{V}\mathbf{X}\mathbf{A}^T\,\Big). \tag{24}$$

It is also not uncommon to postprocess each $\mathbf{y}_t$ with a fully connected neural network (FCNN). This is not unwise because the dimensionality of $\mathbf{y}_t$ is not too large and we will use the same FCNN for all times t. We will not do this here.

Task 5 Modify the module of Task 4 to include a pointwise nonlinear operation. You can choose your favorite nonlinearity here, but we suggest you implement a relu.

3 Transformers

A transformer is a layered architecture where each layer is an attention layer. Formally, this is a composition of operations defined by the recursion

$$\mathbf{A}_\ell \;=\; \mathrm{sm}\Big((\mathbf{Q}_\ell \mathbf{X}_{\ell-1})^T (\mathbf{K}_\ell \mathbf{X}_{\ell-1}) \Big), \tag{25}$$

$$\mathbf{Y}_\ell \;=\; \mathbf{W}_\ell^{h^T} \mathbf{V}_\ell \mathbf{X}_{\ell-1} \mathbf{A}_\ell^T, \tag{26}$$

$$\mathbf{X}_\ell \;=\; \sigma\Big(\mathbf{Y}_\ell\Big). \tag{27}$$

This recursion is initialed with $\mathbf{X}_0 = \mathbf{X}$ and is repeated L times, where L is the number of layers of the transformer. This is analogous to the composition of layers in convolutional and graph neural networks.

For future reference, define the tensor $\mathcal{A} = [\mathbf{Q}_\ell, \mathbf{K}_\ell, \mathbf{V}_\ell, \mathbf{W}_\ell]$ grouping the query, key, value, and dimension recovery matrices of all layers. With this definition we write the output of a transformer as

$$\Phi(\mathbf{X}, \mathcal{A}) \;=\; \mathbf{X}_L \tag{28}$$

In (28), $\mathbf{X}$ is the time series that we input to the transformer and $\mathcal{A}$ is the trainable parameter. The output $\Phi(\mathbf{X}, \mathcal{A})$ is another time series with the same number of time components $T + 1$ and vectors with the same dimension n. The vectors $\mathbf{y}_t$ are representations of time t that depend on the context of the whole time series.

Task 6 Code a Pytorch module to implement a Transformer as specified by (25)-(27). This implementation can leverage the implementation of the attention layer in Task 5.

4 Time Series Prediction

We use a transformer to predict the next entry of a time series; Section 1. To do so observe that the output of the transformer $\mathbf{Y}$ is an $n \times (T+1)$ matrix representing the time series $\mathbf{X}$ which is also an $n \times (T+1)$ matrix. This is mismatched to a problem in which the input is a time series with T components $[\mathbf{x}_0, \ldots, \mathbf{x}_{T-1}]$ and the output is a prediction $\hat{\mathbf{x}}_T$ of the value at time T. To sort out this mismatch consider the average $\bar{\mathbf{x}} = \mathbf{X}\mathbf{1}/T$ of the time series values and define the input of the transformer as the time series

$$\tilde{\mathbf{X}} = \Big[\, \mathbf{X},\, \bar{\mathbf{x}} \,\Big]. \tag{29}$$

In the time series $\tilde{\mathbf{X}}_t$ we append the mean $\bar{\mathbf{x}}$ to the given time series $\mathbf{X}_t$. The idea is that $\bar{\mathbf{x}}$ is a naive prediction of the time series entry for time $T+1$.

We can now use a transformer to refine this estimate. We do that by reading the transformer output at time $T+1$ and declaring it to be our estimate of the weather data,

$$\hat{\mathbf{x}}_T = \Big[\, \Phi(\tilde{\mathbf{X}}, \mathcal{A}) \,\Big]_T. \tag{30}$$

An alternative approach is to process the time series $\mathbf{X}$ without appending the naive estimate $\bar{\mathbf{x}}$. This gives as an output a time series with T components. In this case we declare that the estimate $\hat{\mathbf{x}}_T$ is the average of the outputs of the transformer for all times,

$$\hat{\mathbf{x}}_T = \Big[\, \Phi(\mathbf{X}, \mathcal{A}) \,\Big]\mathbf{1}. \tag{31}$$

Notice that (29)-(30) and (31) are similar approaches. In (29)-(30) we compute an average before running the transformer and in (31) we compute an average after running the transformer.

Task 7 Use the data in Task 1 to train a transformer for weather prediction. You can choose either of the approaches in (29)-(30) or (31). The parameters of the transformer are your choice. We suggest that you use $L = 3$ layers and $m = 3$ for your intermediate representations. Use a mean squared loss and evaluate train and test performance.

5 Multihead Attention

As we did with CNNs and GNNs we also incorporate multiple features. Features in transformers are called heads. Now we have H heads, each with its own query, key, value, and dimension recovery matrices:

$$\mathbf{A}_\ell^h = \text{sm}\Big((\mathbf{Q}_\ell^h \mathbf{X}_{\ell-1})^T (\mathbf{K}_\ell^h \mathbf{X}_{\ell-1}) \Big), \tag{32}$$

$$\mathbf{Y}_\ell^h = \mathbf{W}_\ell^T \mathbf{V}_\ell^h \mathbf{X}_{\ell-1} \mathbf{A}_\ell^{h\,T}, \tag{33}$$

$$\mathbf{X}_\ell = \mathbf{X}_{\ell-1} + \sigma\Big(\sum_{h=1}^{H} \mathbf{Y}_\ell^h \Big). \tag{34}$$

A (minor) difference between multihead transformers and neural networks with multiple features is that the outputs of attention layers always have a single feature. The multiple features $\mathbf{Y}_\ell^h$ generated by different heads are added at the output of each layer to produce the layer's output $\mathbf{X}_\ell$.

In (34), we introduce another change. We add $\mathbf{X}_{\ell-1}$ to the output of the pointwise nonlinearity. This is called a residual connection. It is just a trick that prevents gradients from becoming too small when stacking multiple attention layers during training.

Task 8 Code a Pytorch module to implement a multihead transformer as specified by (32)-(34). This implementation can leverage the implementation of the attention layer in Task 5.

Task 9 Use the data in Task 1 to train a multihead transformer for weather prediction. You can choose either of the approaches in (29)-(30) or (31). The parameters of the transformer are your choice. We suggest that you use $L = 3$ layers, $m = 3$ for your intermediate representations, and $H = 4$ for the number of heads. Use a mean squared loss and evaluate train and test performance.

6 Report

Do not take much time to prepare a lab report. We do not want you to report your code and we don't want you to report your work. Just give us answers to questions we ask. Specifically give us the following:

Question	**Report deliverable**
Task 1	Do not report
Task 2	Do not report
Task 3	Do not report
Task 4	Do not report
Task 5	Do not report
Task 6	Do not report
Task 7	Train MSE and Test MSE
Task 8	Do not report
Task 9	Train MSE and Test MSE

Number of images in the train and test set. Training loss Test loss

We will check that your answers are correct. If they are not, we will get back to you and ask you to correct them. As long as you submit responses, you get an A for the assignment. It counts for 8 points total.