FROM TRANSFORMERS TO PROBABILISTIC TRANSFORMERS [*]

Campbell Rankine
Canberra, ACT
`campbellrankine@gmail.com`

***K*eywords** Probablistic Transformers · Time Series Analysis · Linear Transformers · Regression Analysis

1 Transformer Introduction

Compared to other time series forecasting methods like CNN's and LSTM networks, Transformers have proven to be more effective at considering global patterns in the data. Remembering previous patterns is essential for time series forecasting and this global view makes transformers the top choice for high dimension time series analysis. Linear Transformers learn using the following components.

1.1 Cosine Embeddings

Cosine embeddings encode temporal data into our input so that temporal information is not lost when converting the input to the latent space. Cosine embeddings are defined as:

$$P(v_1, 2i) = sin\Big(\frac{v_1}{n^{\frac{2i}{D_L}}}\Big) = p_1 \tag{1}$$

$$P(v_1, 2i+1) = cos\Big(\frac{v_1}{n^{\frac{2i+1}{D_L}}}\Big) = p_1' \tag{2}$$

1.2 Add and Normalize Module

Known by the shortened name, "Add and Norm layer", these layers combine the input with the output from the attention cells to make sure that information from the input is not lost as we move through the transformer layers. These layers are defined usign some hyperparameter γ and take the input vector: v_i and the layer output vector: v_k.

$$v_i = v_0 + v_k \tag{3}$$

$$v_i = \gamma^{(i)}(v_0 + v_k) + \beta_i \tag{4}$$

1.3 Residual Connections

Given that a layer in our transformer will transform the input, often it's best to pass the input to a later layer of the transformer. This is because we don't want information about our input to be lost as we go through layers of the transformer. To do this we use skip connections that are denoted by arrows in figure 2.

[*]*Citation*: **Authors. Title. Pages.... DOI:000000/11111.**

1.4 Scaled Dot Product Attention

Introduce 3 learnable weight matrices: W^q, W^v, and W^k. The purpose is to learn different representations of the data depending on position and layer output. Now during training we have our Query, Key, Value matrices defined below:

$$Query = W^q x_i$$
$$Key = W^k x_i$$
$$Value = W^v x_i$$

Additionally to calculate $Attention(Q, K, V)$ we will use a softmax activation function to provide normalized outputs in relation to our query. The softmax function ($\sigma(x)$) is defined in equation 5:

$$\sigma(x) = \frac{Exp(x_i)}{\sum_{i \epsilon N}^{N} Exp(x_j)} \tag{5}$$

Now we follow intuition from vector similarity functions like cosine similarity and calculate a similarity value for the Query and Key vectors. Finally we calculate the dot product between the resultant vector and the Value vector to produce the attention value defined in Eq. 9. Additionally we normalize the query key dot product using some constant defined at training time: d_k.

$$Attention(Q, K, V) = \sigma\Big(\frac{QK^T}{\sqrt{d_k}}\Big)V \tag{6}$$

1.5 Multi-headed Attention

another learned dimensionality reduction for the Q, K, V matrices, and we calculate and concatenate attention of each of those reduced dimensionality inputs. This is more efficient than scaled dot product attention for our full dimensionality Q, K, V matrices as we can run these reduced dimensionality heads in parallel, and learn more representations of the data. Multiheaded attention can be calculated using Eq 7. W^0 is our final learnable matrix for attention. It projects the attention value back to the dimensionality of the model, using the concatenated reduced dimensionality attention vector of size h

$$MultiHead(Q, K, V) = Concat([head_1, head_2, ..., head_h])W^0 \tag{7}$$

Where $head_i$ can be calculated using equation 11. Each head value is just the attention of the reduced dimensionality projection of Q, K, and V.

$$head_i = Attention(QW_i^q, KW_i^k, VW_i^v) \tag{8}$$

2 Transformer Architecture

2.1 Attention Architecture

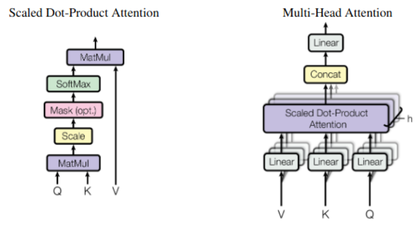

Figure 1: Attention Architecture

2.2 Transformer Architecture

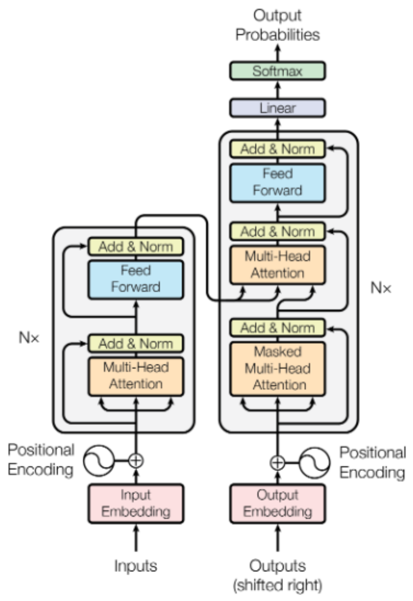

Figure 2: Transformer Architecture

3 Probabilistic Time Series Forecasting

3.1 Input Data

The probabilistic forecasting problem starts with a dataset $\mathcal{D}_{train}$ that satisfies the following condition: $\mid \mathcal{D}_{train} \mid \geq 1$. The dataset can be described as:

$$\mathcal{D}_{train} = \left[x^i_{1:T^i} \right]_{i=1}^{\mathcal{D}}$$

The term T^i above represents the length of the time series i. Given D_{train} we would like to predict some future values of length $P \geq 1$.

3.2 The Modelling Problem

We would like to model some unknown joint distribution of the future P values given its observed past. The past values have a context window of length t. Instead of passing the entire dataset to the model, we'd like to pass some fixed context length, denoted by C in Eq. 9, to the model. The data inside the context window will be denoted by $\mathbb{C}$ in Eq. 9. This assures we have consistent inputs to the model, instead of dynamic sizing. This unknown joint distribution can be described using Eq. 9. The distribution framework we use to model the future output is a parametric distribution, so we can use ϕ to denote the parameters of that distribution. Finally, we have model parameters from our neural network, which we will use Θ to denote. Eq. 9 is derived from the definition of joint distributions.

$$p_\phi \Big(x^i_{C+1:C+P} \mid x^i_{1:C}, \mathbb{C}^i_{1:C+P}; \Theta \Big) = \prod_{\mathtt{t=C+1}}^{\mathtt{C+P}} \Big(x^i_t \mid x^i_{1:t-1}, \mathbb{C}^i_{1:t-1}; \Theta \Big) \tag{9}$$

3.3 Lag Features

To construct a tokenized set of points from numerical time series data we construct a set of lagged features from the prior values of the time series. We denote this positive sorted set of indices using:

$$L_{ind} = \Big[1, 2, ..., L\Big]$$

We select $L_{ind}[i]$ by stepping up by the time change between samples in our time series data. Additionally, to pass extra temporal information to the model we pass a cyclic time based feature such as "second of minute" or "hour of day". This allows the model to implicitly make inferences about the amount of time that has passed between each sample step.

3.4 Standardizing the Model Input

To ensure our model is robust to outliers we must normalize our input (over the context window) by taking:

$$x_t := \frac{x_t - Med(x_t)}{IQR(x_{1:C})}$$

Where Med is the median function and IQR is a function that calculates Interquartile Range.

3.5 Value Scaling

To combat the scaling differences between time series sets, we can pass information about the scale to the model as a form of weight normalization. In this case we can pass the μ and σ for each time series. This functions as a form of input time summary statistics.

4 Probabilistic Transformers for Time Series Modelling

4.1 Decode Only Transformer Architecture

Robust temporal models such as chat-GPT and LLama3 use a network architecture known as a Decode Only Transformer. These models are similar to the Transformers described above however the instead of the encoder block we stack decoder blocks ontop of each other and we input a tokenized version of the input. In our case the tokenized input is given by the lag features. The architecture for Decode-Only Transformers can be found in Figure 3:

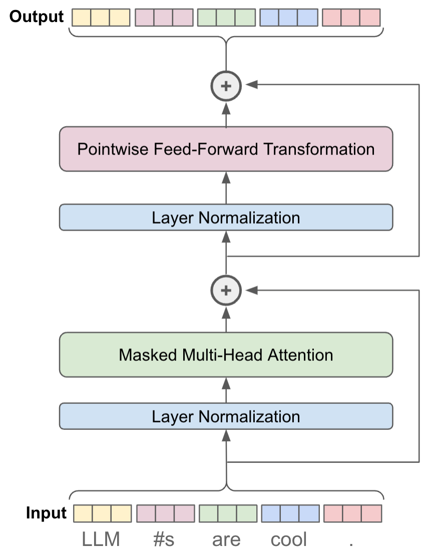
Output
Pointwise Feed-Forward Transformation
Layer Normalization
Masked Multi-Head Attention
Layer Normalization
Input
LLM
#s
are
cool
.

Figure 3: Attention Architecture

4.2 Distribution Heads

Distribution Heads are what project our linear output to a sampled probabilistic space. The specific distribution head used in the Lag-Llama paper is a student t distribution. The PDF for a student-t distribution is given by Equation 10 where Γ represents the gamma function found in other probability distributions.

$$f(t) = \frac{\Gamma(\frac{v+1}{2})}{\sqrt{v\pi}\Gamma(\frac{v}{2})}\left(1 + \frac{t^2}{v}\right)^{\frac{-v+1}{2}} \tag{10}$$

The model outputs 100 samples that we can use to fill the parameters of a sampled student t distribution. From these sample points we estimate the population mean, take the sample mean and degrees of freedom and we project these samples into the parameters for the student t distribution. The output graphs then involve plotting the mean, median, and standard deviation across PREDICTION-LENGTH timesteps.

4.3 Model Loss

Since we're now working in a probabilistic space we need to design a loss function that works in accordance with that. The default loss function used in the Lag-Llama paper is a weighted average of the negative log likelihood of the output samples when compared to the true population median (the future price). The equation for this loss function can be found in Equation 11 where α represents the output distribution from the distribution head:

$$\mathcal{L}(\alpha, \hat{y}_i) = -w_{y_i} log \frac{exp(\alpha_{x,y_n})}{\sum_{c=1}^{C} exp(\alpha_{n,c})} \tag{11}$$