MQTransformer: Multi-Horizon Forecasts with Context Dependent Attention and Optimal Bregman Volatility

Kevin C. Chen
kevnche@amazon.com
Amazon
New York, NY, USA

Lee Dicker
leehd@amazon.com
Amazon
New York, NY, USA

Carson Eisenach
ceisen@amazon.com
Amazon
New York, NY, USA

Dhruv Madeka
maded@amazon.com
Amazon
New York, NY, USA

ABSTRACT

In many forecasting applications (e.g. retail demand, electricity load, weather, finance, etc.), the forecasts must obey certain properties such as having certain context-dependent and time-varying seasonality patterns and avoiding excessive revision as new information becomes available. Here we propose a new forecasting neural net architecture that addresses some of these issues, MQ-Transformer, by incorporating three architectural improvements to the current state-of-the-art: 1) a novel decoder-encoder attention that aligns the historical and future time periods 2) a novel positional encoding that learns seasonality from the historical time series and 3) a novel decoder-self attention that allows the network to minimize the forecast volatility. We then define a new measure of forecast volatility, Bregman Volatility, to understand one major source of the improvement from our model. Bregman Volatility allows us to compute the optimal volatility of a sequence of forecasts in terms of the improvement in forecast accuracy over that time period. We show both theoretically and empirically that the decoder-self attention module optimizes Bregman volatility and thereby improves forecast accuracy as well.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**.

KEYWORDS

time series, neural networks, forecasting, volatility

ACM Reference Format:
Kevin C. Chen, Lee Dicker, Carson Eisenach, and Dhruv Madeka. 2022. MQ-Transformer: Multi-Horizon Forecasts with Context Dependent Attention and Optimal Bregman Volatility. In *Proceedings of 8th SIGKDD International Workshop on Mining and Learning from Time Series – Deep Forecasting: Models, Interpretability, and Applications (KDD '22).* ACM, New York, NY, USA, 11 pages. https://doi.org/XXXXXXX.XXXXXXX

KDD '22, August 15, 2022, Washington, DC

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Time series forecasting is a fundamental problem in machine learning with relevance to many applications including supply chain management, finance, healthcare, *etc.* Modern forecasting applications require predictions of many correlated time series over multiple horizons. In multi-horizon forecasting, the learning objective is to produce forecasts for multiple future horizons at each time-step. Beyond point estimation, decision making problems require a measure of uncertainty about the forecasted quantity. Access to the full distribution is often unnecessary and several quantiles are sufficient (e.g. many problems in Operations Research use the 50^{th} and 90^{th} quantiles).

As an example, consider a large e-commerce retailer with a system to produce forecasts of the demand distribution for a set of products at a target time T. Using these forecasts as an input, the retailer can then optimize buying and placement decisions. Accurate forecasts are important, but – perhaps less obviously – forecasts that don't exhibit excess volatility as a target date approaches minimize costly effects in a supply chain [4, 5].

Recent work applying deep learning to time-series forecasting focuses primarily on the use of recurrent and convolutional architectures [12, 20, 21, 30, 32][1]. These are Seq2Seq architectures [26] – which consist of an *encoder* that summarizes an input sequence into a fixed-length context vector, and a *decoder* which produces an output sequence. This line of work has led to major advances in forecast accuracy and real-world forecasting systems increasingly rely on neural nets. Accordingly, a need for black-box forecasting system diagnostics has arisen. In this paper, we develop a new notion called *Bregman Volatility* to quantify the amount of forecast volatility beyond the forecast changes required to incorporate new information and improve forecast accuracy. While tools such as Bregman Volatility can detect flaws in forecasts, the question of how to incorporate that information into model design is unexplored. Existing multi-horizon forecasting architectures do not explicitly handle excess variation, since forecasts on any particular date are not made aware of errors in the forecast for previous dates.

Another limitation in many existing architectures is the well known information bottleneck, where the encoder transmits information to the decoder via a single hidden state. To address this, Bahdanau et al. [1] introduce a method called *attention*, allowing

[1] For a complete overview see Benidis et al. [3]

the decoder to take as input a weighted combination of relevant latent encoder states at each output time step, rather than using a single context to produce all decoder outputs. While NLP is the predominant application of attention architectures to date, in this paper we use novel attention modules and positional embeddings to introduce the proper inductive biases for probabilistic time-series forecasting to the model architecture.

Summary of Contributions In this paper, we are concerned with both improving forecast accuracy *and* reducing excess forecast volatility. We present a set of novel architectures that seek to remedy some of inductive biases that are currently missing in state of the art MQ-Forecasters [30]. Our main contributions are

- **Positional Encoding from Event Indicators**: Current MQ-Forecasters use explicitly engineered holiday "distances" to provide the model with information about seasonality. We introduce a novel positional encoding mechanism that allows the network to learn a context-dependent seasonality function specific that is more general than conventional position encoding schemes.
- **Horizon-Specific Decoder-Encoder Attention**: Current MQ-Forecasters learn a single encoder representation for all future periods being forecasted. We present a novel horizon-specific decoder-encoder attention scheme that allows the network to learn a representation of the past that depends on the period being forecasted.
- **Decoder Self-Attention for Forecast Evolution**: To the best of our knowledge, this is *the first work* to consider the *impacts of network architecture design on forecast evolution*. Importantly, we accomplish this by using attention mechanisms to introduce the right inductive biases, and not by explicitly penalizing a measure of forecast variability so that we do not need to make trade-offs between accuracy and volatility.
- **Bregman Volatility**: We develop a new notion of forecast volatility called "Bregman Volatility" and show how our Decoder Self-Attention module optimizes Bregman Volatility. Bregman Volatility shows how forecast volatility and accuracy are closely linked and may be of independent interest in other contexts.

By providing MQ-Forecasters with the structure necessary to learn *context information dependent encodings*, we observe major increases in accuracy (5.5% in overall P90 quantile loss throughout the year, and **up to 33% during peak periods**) on our demand forecasting application. In terms of excess volatility, we see a reduction of 68% in Bregman Volatility for the mean forecast.

In addition, we apply MQTransformer to four public datasets and show parity with the state-of-the-art on simple univariate tasks. On a substantially more complex public dataset (retail forecasting) we show a **38% improvement** over the previously reported state-of-the-art, and a 5% improvement in P50 QL, 11% in P90 QL versus our baseline. Because our innovations are compatible with efficient training schemes, our architecture also achieves a significant speedup (several orders of magnitude greater throughput) over earlier transformer models for time-series forecasting.

2 BACKGROUND AND RELATED WORK

Time Series Forecasting We consider the high-dimensional regression problem

$$p(y_{t+1,i}, \ldots, y_{t+H,i} | \mathbf{y}_{:t,i}, \mathbf{x}^{(h)}_{:t,i}, \mathbf{x}^{(f)}_{:t,i}, \mathbf{x}^{(s)}_{i}), \tag{1}$$

where $y_{t+s,i}, \mathbf{y}_{:t,i}, \mathbf{x}^{(h)}_{:t,i}, \mathbf{x}^{(f)}_{t:,i}, \mathbf{x}^{(s)}_{i}$ denote future observations of the target time series i, observations of the target time series observed until time t, the past covariates, known future information, and static covariates, respectively.

For sequence modeling problems, Seq2Seq [26] is the canonical deep learning framework and it has been adapted to time series forecasting [12, 20, 21, 23, 29, 30, 32]. The MQ-Forecaster framework [30] solves (1) by treating each series i as a sample from a joint stochastic process and feeding into a neural network which predicts Q quantiles for each horizon. These types of models have limited contextual information available to the decoder – a drawback inherited from the Seq2Seq architecture – as it produces each estimate $y^q_{t+s,i}$, the q^{th} quantile of the distribution of the target at time $t+s$, $y_{t+s,i}$. Seq2Seq models rely on a single encoded context to produce forecasts for all horizons, imposing an information bottleneck and making it difficult to learn long term dependencies.

Our MQTransformer architecture uses the direct strategy: the model outputs the quantiles of interest directly, rather than the parameters of a distribution from which samples are to be generated. This has been shown [30] to outperform parametric models, like DeepAR [23], on a wide variety of tasks. Recently, Lim et al. [18] consider an application of attention to multi-horizon forecasting, but their method still produces a single context for all horizons and by using an RNN decoder, does not enjoy the same scaling properties as MQ-Forecaster models.

Attention Mechanisms Bahdanau et al. [1] introduced the attention mechanism to solve the information bottleneck and sequence alignment problems in Seq2Seq architectures for NMT. Recently, attention has enjoyed success in many applications including natural language processing (NLP), computer vision (CV) and time-series forecasting [7, 11, 15, 17, 18, 25, 31]. Many variants have been proposed including self-attention and dot-product attention [6, 9, 19, 28], and transformer architectures (end-to-end attention with no recurrent layers) achieve state-of-the-art performance on most NLP tasks.

Time series forecasting applications exhibit seasonality and the absolute position encodings commonly used in the literature cannot be applied. Our work differs from previous work on *relative position encodings* [8, 14, 24] in that we learn a representation from a time series of indicator variables which encode events relevant to the target application (e.g. holidays and promotions). Existing encoding schemes either involve feature engineering (e.g. sinusoidal encodings) or have a maximum input sequence length. See Appendix A for more background on attention mechanisms.

Notation We denote by H and Q the number of horizons and quantiles being forecast, respectively. Bolded characters are used to indicate vector and matrix values. The concatenation of two vectors $\mathbf{v}$ and $\mathbf{u}$ is denoted as $[\mathbf{u}; \mathbf{v}]$.

3 METHODOLOGY

Here we present our MQTransformer architecture, building on the MQ-Forecaster framework [30]. We extend the MQ-Forecaster family of models because it, unlike many other architectures in the literature, can be applied to millions of samples due to its use of *forking sequences* – a technique to dramatically increase throughput during training and avoid expensive data augmentation.

For ease of exposition, we reformulate the generic probabilistic forecasting problem in (1) as

$$p(y_{t+1,i}, \ldots, y_{t+H,i} | \mathbf{y}_{:t,i}, \mathbf{x}_{:t,i}, \mathbf{x}_i^{(l)}, \mathbf{x}^{(g)}, \mathbf{x}_i^{(s)}),$$

where $\mathbf{x}_{:t,i}$ are past observations of all covariates, $\mathbf{x}_i^{(l)} = \{\mathbf{x}_{s,i}^{(l)}\}_{s=1}^{\infty}$ are known covariates specific to time-series i, $\mathbf{x}^{(g)} = \{\mathbf{x}_s^{(g)}\}_{s=1}^{\infty}$ are the global, known covariates. Here known signifies that the model has access to (potentially noisy) observations of past and future values. Note that this formulation is equivalent to (1), and that known covariates can be included in the past covariates $\mathbf{x}_{:t}$. When it can be inferred from context, the time series index i is omitted.

Learning Objective We train a quantile regression model to minimize the quantile loss, summed over all forecast creation times (FCTs) and quantiles $\sum_t \sum_q \sum_k L_q \left(y_{t+k}, \widehat{y}_{t+k}^{(q)}\right)$, where $L_q(y, \widehat{y}) = q(y - \widehat{y})_+ + (1-q)(\widehat{y} - y)_+$, $(\cdot)_+$ is the positive part operator, q is a quantile, and k is the horizon.

3.1 Network Architecture

The design of the architecture is similar to MQ-(C)RNN [30], and consists of encoder, decoder and position encoding blocks. The position encoding outputs, for each time step t, are a representation of global position information, $\mathbf{r}_t^{(g)} = \text{PE}_t^{(g)}(\mathbf{x}_i^{(g)})$, as well as time-series specific context information, $\mathbf{r}_t^{(l)} = \text{PE}_t^{(l)}(\mathbf{x}_i^{(l)})$. Intuitively, $\mathbf{r}_t^{(g)}$ captures position information that is independent of the time-series i (such as holidays), whereas $\mathbf{r}_t^{(l)}$ encodes time-series specific context information (such as promotions). In both cases, the inputs are a time series of indicator variables *and require no feature-engineering or handcrafted functions.*

The encoder then summarizes past observations of the covariates into a sequence of hidden states $\mathbf{h}_t := \text{encoder}(\mathbf{y}_{:t}, \mathbf{x}_{:t}, \mathbf{r}_{:t}^{(g)}, \mathbf{r}_{:t}^{(l)}, \mathbf{s})$. Using these representations, the decoder produces an $H \times Q$ matrix of forecasts $\widehat{\mathbf{Y}}_t = \text{decoder}(\mathbf{h}_{:t}, \mathbf{r}^{(g)}, \mathbf{r}^{(l)})$. Note that in the decoder, the model has access to position encodings.

In this section we focus on our novel attention blocks and position encodings; the reader is directed to Appendix A.2 for additional architecture details.

MQTransformer Following the generic pattern given above, we present the MQTranformer architecture. First, define the combined position encoding as $\mathbf{r} := [\mathbf{r}^{(g)}; \mathbf{r}^{(l)}]$. In the encoder we use a stack of dilated temporal convolutions [27, 30] to encode historical time-series and a multi-layer perceptron to encode the static features as (2).

Our decoder incorporates our horizon specific and decoder self-attention blocks, and consists of two branches. The first (global) branch summarizes the encoded representations into horizon-specific

Table 1: MQTransformer encoder and decoder

Encoder	
$\mathbf{h}_t^1 = \text{TemporalConv}(\mathbf{y}_{:t}, \mathbf{x}_{:t}, \mathbf{r}_{:t})$	(2)
$\mathbf{h}_t^2 = \text{FeedForward}(\mathbf{s})$	
$\mathbf{h}_t = [\mathbf{h}_t^1; \mathbf{h}_t^2]$,	
Decoder Contexts	
$\mathbf{c}_{t,h} = \text{HSAttention}(\mathbf{h}_{:t}, \mathbf{r})$	(3)
$\mathbf{c}_t^a = \text{FeedForward}(\mathbf{h}_t, \mathbf{r})$	
$\mathbf{c}_t = [\mathbf{c}_{t,1}; \cdots; \mathbf{c}_{t,H}; \mathbf{c}_t^a]$	
$\widetilde{\mathbf{c}}_{t,h} = \text{DSAttention}(\mathbf{c}_{:t}, \mathbf{h}_{:t}, \mathbf{r})$,	

$(\mathbf{c}_{t,h})$ and horizon agnostic $(\mathbf{c}_t^a)$ contexts. Formally, the global branch $\mathbf{c}_t := m_G(\cdot)$ is given by (3).

The output branch consists of a self-attention block followed by a local MLP, which produces outputs using the same weights for each horizon. For FCT t and horizon h, the output is given by $(\widehat{y}_{t+h}^1, \ldots, \widehat{y}_{t+h}^Q) = m_L(\mathbf{c}_t^a, \mathbf{c}_{t,h}, \widetilde{\mathbf{c}}_{t,h}, \mathbf{r}_{t+h})$. Next we describe the specifics of our position encoding and attention blocks.

3.2 Learning Position and Context Representations from Event Indicators

Prior work typically uses a variant on one of two approaches to provide attention blocks with position information: (1) a handcrafted representation (such as sinusoidal encodings) or (2) a matrix $\mathbf{M} \in \mathbb{R}^{L \times d}$ of position encoding where L is the maximum sequence length and each row corresponds to the position encoding for time point.

In contrast, our novel encoding scheme maps sequences of indicator variables to a d-dimensional representations. For demand forecasting, this enables our model to learn an arbitrary function of events (like holidays and promotions) to encode position information. As noted above, our model includes two position encodings: $r_t^{(g)} := PE_t^{(g)}(\mathbf{x}^{(g)})$ and $r_t^{(l)} := PE_t^{(l)}(\mathbf{x}^{(l)})$, one that is shared among all time-series i and one that is specific. For the design we use in Section 4, $PE^{(g)}$ is implemented as a bidirectional (looking both forward and backward in time) 1-D convolution and $PE^{(l)}$ is an MLP applied separately at each time step. For reference, MQ-(C)RNN [30] uses linear holiday and promotion distances to represent position information.

By using 1-D convolutions and MLPs over a time-series of indicator variables, our approach is more flexible and can be used to learn a position representation that is context specific. In the demand forecasting application, consider two products during a holiday season where one has a promotion and the other does not. These two products need different position encodings, else the decoder-encoder attention (Section 3.3) will not be able to align target horizons with past contexts. In addition, note that the classical method of learning a matrix embedding $\mathbf{M}$ can be recovered as a special case of our approach. Consider a sequence of length L, and take $\mathbf{x}^{(g)} := [\mathbf{e}_1, \ldots, \mathbf{e}_L]$, where $\mathbf{e}_s$ is used to denote the vector in $\mathbb{R}^L$ with a 1 in the s^{th} position and 0s elsewhere. To recover the matrix embedding scheme, we define $\text{PE}_t^{\text{matrix}}(\mathbf{x}^{(g)}) := \mathbf{x}_t^{(g),\top}\mathbf{M}$.

3.3 Context Dependent and Feedback-Aware Attention

Horizon-Specific Decoder-Encoder Attention To motivate the horizon-specific attention unit, consider a retail demand forecasting task. At each FCT T, the forecaster produces forecasts for multiple target horizons, which may contain different events such as promotions or holidays. Prior work [30] incorporated horizon-specific contexts with the functional form $c_{t,h} = f(h_t, r_{t+h})$ for each FCT, target horizon. Because events like promotions or holidays – which are strong predictors of observed demand – are fairly sparse, this functional form is insufficient. Instead, a function of the form $c_{t,h} = f(h_1, r_1, \ldots, h_t, r_t, r_{t+h})$ allows the model to use information from many past periods, which is valuable due to the sparsity of relevant events. Using an attention mechanism to align target horizons enables this. Figure 1 depicts the difference between prior work and the horizon-specific attention in MQTransformer.

To do this, we introduce the horizon-specific attention mechanism, which can be viewed is a multi-headed attention mechanism where the projection weights are shared across all horizons. Each head corresponds to a different horizon. It differs from a traditional multi-headed attention mechanism in that its purpose is to attend over representations of past time points to produce a representation specific to the target period. In our architecture, the inputs to the block are the encoder hidden states and position encodings. Mathematically, for time s and horizon h, the attention weight for the value at time t is computed as (4).

Observe that there are two key differences between these attention scores and those in the vanilla transformer architecture: (a) projection weights are shared by all H heads, (b) the addition of the position encoding of the target horizon h to the query. The output of our horizon specific decoder-encoder attention block, $\mathbf{c}_{t,h}$, is obtained by taking a weighted sum of the encoder hidden contexts, up to a maximum look-back of L periods as in (5).

Decoder Self-Attention Our Bregman Volatility theory below shows a deep connection between accuracy and volatility. We leverage this connection to develop a novel decoder self-attention scheme for multi-horizon forecasting. To motivate the development, consider a model which forecasts values of 40, 60 when the demand has constantly been 50 units. We would consider this model to have excess volatility. Similarly, a model forecasting 40, 60 when demand jumps between 40 and 60 units would not be considered to have excess volatility. The first model fails to learn from its past forecasts.

To ameliorate this, we pass information of the previous forecast errors to the current forecast. For each FCT t and horizon h, the model attends on the previous forecasts using a query containing the demand information for that period. The attention mechanism has a separate head for each horizon. The demand information at time t is incorporated via the encoded context $\mathbf{h}_t$ and previous forecasts are represented via the corresponding horizon-specific context $\mathbf{c}_{s,r}$ – in the absence of decoder-self attention $\mathbf{c}_{s,r}$ would be passed through the local MLP to generate the forecasts. Formally, the attention scores are given by (6). The horizon-specific and feedback-aware outputs, $\widetilde{\mathbf{c}}_{t,h}$, are given by (7). Note how we sum only over previous forecasts of the same period. In Section 4 we demonstrate the importance of this choice by comparing to a more traditional decoder-self attention unit that attends over all past forecasts. See Figure 2 in Appendix A.2 for an illustration.

4 EMPIRICAL RESULTS

4.1 Large-Scale Demand Forecasting

First, we evaluate our architecture on a demand forecasting problem for a large-scale e-commerce retailer with the objective of producing multi-horizon forecasts for the next 52 weeks. We conduct our experiments on a subset of products ($\sim$ 2 million products) in the US store. Each model is trained using a single machine with 8 NVIDIA V100 Tensor Core GPUs, on three years of demand data (2015-2018); one year (2018-2019) is held out for back-testing.

To assess the effects of each innovation, we ablate by removing components one at a time. The architectures we compare are the baseline (MQ-CNN), MQTransformer (MQT), MQTransformer without decoder self-attention (MQT-NoDS), and MQ-Transformer with a decoder self-attention unit that attends over all past forecasts (MQT-All). MQ-CNN is selected as the baseline since prior work[2] demonstrate that MQ-CNN outperforms MQ-RNN and DeepAR on this dataset, and as can be seen in Table 4, MQ-CNN outperforms MQ-RNN and DeepAR on the public retail forecasting dataset. For additional details see Appendix A.6 (Table 5 describes the decoder-self attention unit used in MQT-All).

Forecast Accuracy Table 3 summarizes several key metrics that demonstrate the accuracy improvements achieved by adding our proposed attention mechanisms to the MQ-CNN architecture. We consider overall quantile loss, as well as quantile loss for specific target periods (seasonal peaks, promotions). We also measure post-peak ramp-down performance – models that suffer issues with alignment will continue to forecast high for target weeks after a seasonal peak. By including MQTransformer's attention mechanisms, we see up to 33% improvements for seasonal peaks and 24.9% improvements on promotions versus MQ-CNN. Further, the ablation analysis (against MQT-NoDS and MQT-All) shows that both novel attention units are important to improving accuracy, and that our decoder-self attention scheme (where we attend only over past forecasts for the same target period) introduces a powerful inductive bias. Table 8 in Appendix A.6 shows the metrics computed on only short horizons, where we see even greater gains in accuracy.

4.2 Publicly Available Datasets

Following Lim et al. [18], we consider applications to brick-and-mortar retail sales, electricity load, securities volatility and traffic forecasting. In each task, we report the quantile loss summed over all forecast creation times, normalized by the target values:

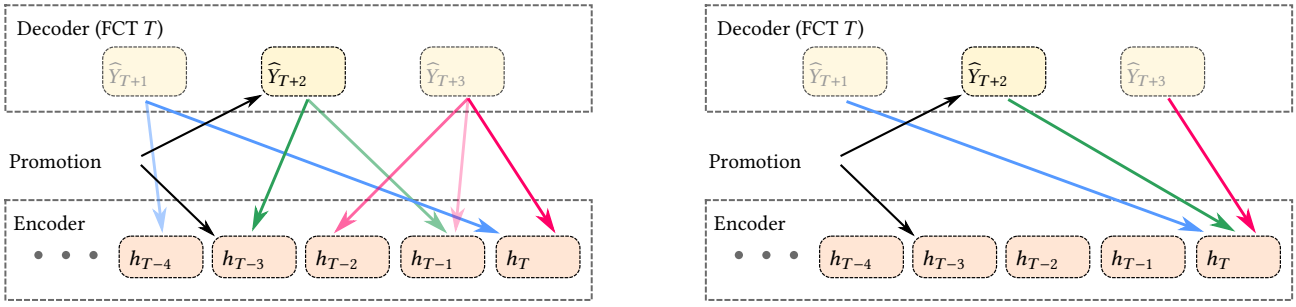
$$\frac{2 \sum_t \sum_k L_q\left(y_{t+k}, \widehat{y}^{(q)}_{t+k}\right)}{\sum_t \sum_k |y_{t+k}|}.$$

For the retail task, we predict the next 30 days of sales, given the previous 90 days of history. This dataset has a rich set of static, time series, and known features. At the other end of the spectrum, the electricity load dataset is univariate. See Section 6 for more information about these tasks. Table 4 compares MQTransformer's

[2]Wen et al. [30], Figure 3 shows MQ-CNN (labeled "MQ_CNN_wave") outperforms MQ-RNN (all variants) and DeepAR (labeled "Seq2SeqC") on the test set.

Table 2: Attention weight and output computations for blocks introduced in Section 3.3

Block	Attention Weights	Output
Decoder-Encoder Attention	$A^h_{t,s} = \mathbf{q}^{h,\top}_t \mathbf{W}^\top_q \mathbf{W}_k \mathbf{k}_s$ (4) $\mathbf{q}^h_t = [\mathbf{h}_t; \mathbf{r}_t; \mathbf{r}_{t+h}]$ $\mathbf{k}_s = [\mathbf{h}_s; \mathbf{r}_s]$ $\mathbf{v}_s = \mathbf{h}_s$	$\mathbf{c}_{t,h} = \sum_{s=t-L}^{t} A^h_{t,s} \mathbf{W}_v \mathbf{v}_s$ (5)
Decoder Self-Attention	$A^h_{t,s,r} = \mathbf{q}^\top_{t,h} \mathbf{W}^{h,\top}_q \mathbf{W}^h_k \mathbf{k}_{s,r}$ (6) $\mathbf{q}_{t,h} = [\mathbf{h}_t; \mathbf{c}_{t,h}; \mathbf{r}_t; \mathbf{r}_{t+h}]$ $\mathbf{k}_{s,r} = [\mathbf{c}_{s,r}; \mathbf{r}_s; \mathbf{r}_{s+r}]$ $\mathbf{v}_{s,r} = \mathbf{c}_{s,r}$	$\widetilde{\mathbf{c}}_{t,h} = \sum_{(s,r)\in\mathcal{H}(t,h)} A^h_{s,t,r} \mathbf{W}^h_v \mathbf{v}_{s,r}$, (7) $\mathcal{H}(t,h) := \{(s,r) \mid s + r = t + h\}$



(a) MQTransformer decoder at FCT T. **(b) MQCNN decoder at FCT T**

Figure 1: Example of a demand forecasting task where periods $T-3$ and $T+2$ both have promotions. The encoded context h_{T-3}, the last time the item had a promotion, contains useful information for forecasting for target periods that also have a promotion. The horizon-specific attention aligns past encoded contexts with the target horizon.

Table 3: P50 (50th percentile) and P90 (90th percentile) quantile loss on the backtest year along different dimensions. Values indicate relative performance versus the baseline.

	MQ-CNN		MQT		MQT-NoDS		MQT-All	
Dimension	P50	P90	P50	P90	P50	P90	P50	P90
Overall	1.000	1.000	**0.977**	**0.945**	0.983	0.963	**0.977**	0.957
Seasonal Peak 1	1.000	1.000	**0.865**	**0.667**	0.881	0.688	0.908	0.729
Seasonal Peak 2	1.000	1.000	0.957	**0.884**	0.984	0.946	**0.953**	0.931
Seasonal Peak 3	1.000	1.000	0.838	**0.845**	**0.837**	0.892	0.851	0.870
Post-Peak Rampdown	1.000	1.000	0.987	**0.978**	**0.985**	0.997	0.990	0.991
Promotion Type 1	1.000	1.000	**0.945**	**0.800**	0.969	0.824	0.953	0.842
Promotion Type 2	1.000	1.000	**0.868**	**0.751**	0.895	0.762	0.914	0.808
Promotion Type 3	1.000	1.000	**0.949**	**0.837**	1.075	1.004	0.960	0.911

performance with DeepAR [23], ConvTrans [17], MQ-RNN [30], and TFT [18] [3].

Our MQTransformer architecture is competitive with or beats the state-of-the-art on the electricity load, volatility and traffic prediction tasks, as shown in Table 4. On the most challenging task, it dramatically outperforms the previously reported state of the art by 38% and the MQ-CNN baseline by 5% at P50 and 11% at P90.

Because MQ-CNN and MQTransformer are trained using forking sequences, we can use the entire training population, rather than downsample as is required to train TFT [18]. To ascertain what portion of the gain is due to learning from more trajectories, versus our innovations alone, we retrain the optimal MQTransformer architecture using a random sub-sample of 450K trajectories (the same sampling procedure as TFT) and without using forking sequences – the results are indicated in parentheses in Table 4. We can observe that MQTransformer still dramatically outperforms

[3] Results for TFT, MQ-RNN, DeepAR and ConvTrans are from [18].

TFT, and its performance is similar to the MQ-CNN baseline trained on all trajectories. See Section 6 for more details and results on the Favorita forecasting task.

Computational Efficiency As an example take the Retail dataset. [18] was only able to use 450K out of 20M trajectories and the optimal TFT architecture required 13 minutes per epoch (minimum validation error at epoch 6)[4] using a single V100 GPU. Our innovations are compatible with forking sequences so our architecture can use *all available trajectories*. To train, MQTransformer requires only 5 minutes per epoch (on 20M trajectories) using a single V100 GPU (minimum validation error at 5 epochs).Some of the differences in runtime can be attributed to use of different deep learning frameworks, but it is clear that MQTransformer can be trained much more efficiently than models like TFT and DeepAR.

5 ABLATION STUDIES

In the ablation studies, the model MQT-All uses a standard multi-head attention applied to the concatenated contexts for all horizons at each period t. A separate head is used for each output horizon (52 heads on the private dataset). Table 6 gives the number of parameters in each trained model.

5.1 Experiment Setup

In this section we describe the details of the model architecture and training procedure used in the experiments on the large-scale demand forecasting application.

Training Procedure. Because we did not have enough history available to set aside a true holdout set, all models are trained for 100 epochs, and the final model is evaluated on the test set. For the same reason, no hyperparameter tuning was performed.

Architecture and Hyperparameters. The categorical variables consist of static features of the item, and the timeseries categorical variables are event indicators (e.g. holidays). The parameters are summarized in Table 7.

5.2 Results for shorter horizons

Table 8 shows results for horizons up to 6 weeks. Similar to Section 4, MQTransformer significantly outperforms the baseline.

6 EXPERIMENTS ON PUBLIC DATASETS

We describe the experiment setup used for the public datasets in Section 4. As mentioned , we display the baseline results published in Lim et al. [18]. For MQ-CNN and MQTransformer, we use their pre-processing and evaluation code to ensure parity.

6.1 Datasets

We evaluate our MQTransformer on four public datasets. We summarize the datasets and preprocessing logic below; the reader is referred to Lim et al. [18] for more details. Lim et al. [18] released their code under the Apache 2.0 License. Favorita's terms of use for their retail dataset allows for it to be used for scientific research. The Electricity and Traffic datasets are provided by the UCI machine learning repository [10]. The volatility dataset is sourced from the Oxford-Man Institute's realized library v0.3 [13].

Retail. This dataset is provided by the Favorita Corporacion (a major Ecuador Grocery) as part of a Kaggle[5] to predict sales for thousands of items at multiple brick-and-mortar locations. In total there are 135K items (item, store combinations are treated as distinct entities), and the dataset contains a variety of features including: local, regional and national holidays; static features about each item; total sales volume at each location. The task is to predict log-sales for each (item, store) combination over the next 30 days, using the previous 90 days of history. The training period is January 1 - December 1, 2015. The following 30 days are used as a validation set, and the 30 days after that as the test set. These 30 day windows correspond to a single FCT. While Lim et al. [18] extract only 450K samples from the histories during the train window, there are in fact 20M trajectories avalaible for training – because our models can produce forecasts for multiple trajectories (FCDs) simultaneously, we train using all available data from the training window.

Electricity. This dataset consists of time series for 370 customers of at an hourly grain. The univariate data is augmented with a day-of-week, hour-of-day and offset from a fixed time point. The task is to predict hourly load over the next 24 hours for each customer, given the past seven days of usage. From the training period (January 1, 2014 through September, 1 2019) 500K samples are extracted.

Traffic. This dataset consists of lane occupancy information for 440 San Francisco area freeways. The data is aggregated to an hourly grain, and the task is to predict the hourly occupancy over the next 24 hours given the past seven days. The training period consist of all data before 2008-06-15, with the final 7 days used as a validation set. The 7 days immediately following the training window is used for evaluation. The model takes as input lane occupancy, hour of day, day of week, hours from start and an entity identifier.

Volatility. The volatility dataset consists of 5 minute sub-sampled realized volatility measurements from 2000-01-03 to 2019-06-28. Using the past one year's worth of daily measurements, the goal is to predict the next week's (5 business days) volatility. The period ending on 2015-12-31 is used as the training set, 2016-2017 as the validation set, and 2018-01-01 through 2019-06-28 as the evaluation set. The region identifier is provided as a static covariate, along with time-varying covariates daily returns, day-of-week, week-of-year and month. A log transformation is applied to the target.

7 BREGMAN VOLATILITY DEFINITIONS

Assume that some number $X \in \mathbb{R}$ (e.g. customer demand) is observed at a given time point in the future, T_∞, and forecasts for X, denoted $\widehat{X}_t$, are created at time points $t = 1, 2, \ldots, T < T_\infty$. The accuracy of each forecast $\widehat{X}_t$ is measured by $c(\widehat{X}_t - X)$, where $c : \mathbb{R} \to \mathbb{R}$ is some convex cost function that is minimized at 0. We assume that c is differentiable on all of $\mathbb{R}$, except possibly at 0.

Let $\mathcal{F}_t$ denote the σ-field of information available at time t, so, in particular, $\mathcal{F}_t \subseteq \mathcal{F}_{t+1}$. Define the function $c_X : \mathbb{R} \to \mathbb{R}$ by $c_X(x) = c(x - X)$. Our goal is to find forecasts $\widehat{X}_t \in \mathcal{F}_t$ that are

[4]Timing results obtained by running the source code provided by Lim et al. [18]

[5]The original competition can be found here.

Table 4: Quantile loss metrics with the best results on each task emphasized. Results in parentheses correspond to training MQTransformer without forking sequences on 450K trajectories only.

	Task	DeepAR	ConvTrans	MQ-RNN	MQ-CNN	TFT	MQTransformer
P50 QL	Electricity	0.075	0.059	0.077	0.076	**0.055**	0.057
	Retail	0.574	0.429	0.379	0.269	0.354	**0.256** (0.2645)
	Volatility	0.050	0.047	0.042	0.042	**0.039**	**0.039**
	Traffic	0.161	0.122	0.117	0.115	**0.095**	0.101
	Task	DeepAR	ConvTrans	MQ-RNN	MQ-CNN	TFT	MQTransformer
P90 QL	Electricity	0.040	0.034	0.036	0.035	**0.027**	**0.027**
	Retail	0.230	0.192	0.152	0.118	0.147	**0.106** (0.109)
	Volatility	0.024	0.024	0.021	0.020	0.020	**0.019**
	Traffic	0.099	0.081	0.082	0.077	0.070	**0.068**

Table 5: Attention weight and output computations for MQT-All decoder self-attention

Attention Weights	Output
$A^h_{t,s} = \mathbf{q}_t^\top \mathbf{W}_q^{h,\top} \mathbf{W}_k^h \mathbf{k}_s$ $\mathbf{q}_t = [\mathbf{c}_{s,1}; \cdots; \mathbf{c}_{s,H}; \mathbf{h}_t; \mathbf{r}_t]$ $\mathbf{k}_s = [\mathbf{c}_{s,1}; \cdots; \mathbf{c}_{s,H}; \mathbf{r}_s]$ $\mathbf{v}_s = [\mathbf{c}_{s,1}; \cdots; \mathbf{c}_{s,H}]$	$\widetilde{\mathbf{c}}_{t,h} = \sum_{s=t-L}^{t} A^h_{t,s} \mathbf{W}_v^h \mathbf{v}_s$

Table 6: Parameter counts in trained models for the large scale demand-forecasting task.

Model	Number of parameters
MQ-CNN	9.17×10^5
MQT	9.25×10^5
MQT-NoDS	9.09×10^5
MQT-All	2.82×10^6

Table 7: Parameter settings for Large Scale Demand Forecasting Experiments

Parameter	Value
Encoder Convolution Dilation Rates	[1,2,4,8,16,32]
Position Encoding Dilation Rates	[1,2,4,8,16,20]
Static Categorical	One-Hot
Time-Series Categorical	One-Hot
Static Encoder Dimension	64
Convolution Filters	32
Attention Block Head Dimension	16
Dropout Rate	0.15
Activation Function	ReLU

optimal in the following sense:

$$\widehat{X}_t = \underset{X_t \in \mathcal{F}_t}{\operatorname{argmin}} \mathbb{E}[c_X(X_t) \mid \mathcal{F}_t]. \tag{8}$$

Observe from (8) the loss function c determines the form of the corresponding forecast $\widehat{X}_t$. For example, if the cost function is squared-error loss, then the forecasts are conditional means, while if cost function is quantile loss, then the forecasts are conditional quantile forecasts.

If the forecasts $\widehat{X}_t$ satisfy (8), this implies that the evolution of the sequence of forecasts $\widehat{X}_1, \widehat{X}_2, \ldots, \widehat{X}_T$ should satisfy properties determined by their Bregman volatility, which is defined in (9). In particular, the forecast accuracy improvement from time $t = 1$ to time $t = T$ should equal the Bregman volatility of the sequence of forecasts, in expectation. To our knowledge, Bregman volatility is a new concept, but it builds on Bregman divergence, an important tool from convex analysis for measuring the distance between points with respect to a specified convex function (e.g. [2]).

Definition 7.1. Define $D_{c_X} : \mathbb{R}^2 \to \mathbb{R}$ to be the Bregman divergence of c_X, which is given by $D_{c_X}(x, y) = c_X(x) - c_X(y) - c'_X(y)(x - y)$, for $x, y \in \mathbb{R}$. If c is not differentiable at 0, define $c'_X(X) = 0$ and observe that 0 is an element of the subgradient $\partial c'_X(X)$, because c is minimized at 0 by assumption.

Definition 7.2. Define the t-th Bregman volatility of a sequence of forecasts $\widehat{X}_1, \ldots, \widehat{X}_T$ to be

$$\mathrm{Vol}_{c_X}^{(t:T)} = \sum_{t'=t}^{T-1} D_{c_X}(\widehat{X}_{t'}, \widehat{X}_{t'+1}) \tag{9}$$

To connect Bregman volatility and forecast accuracy, observe that for $t < T$,

$$\mathbb{E}[c_X(\widehat{X}_t)] = \mathbb{E}[c_X(\widehat{X}_T)] + \sum_{t'=t}^{T-1} \mathbb{E}[c'_X(\widehat{X}_{t'+1})(\widehat{X}_{t'} - \widehat{X}_{t'+1})] + \mathbb{E}[\mathrm{Vol}_{c_X}^{(t:T)}] \tag{10}$$

If the forecasts $\widehat{X}_t$ satisfy (8) for $1 \le t \le T$ and if c is differentiable at 0 or $X \mid \mathcal{F}_{t'+1}$ is continuous with a differentiable probability density function, then $\mathbb{E}[c'_X(\widehat{X}_{t'+1}) \mid \mathcal{F}_{t'+1}] = 0$. Under these assumptions, it follows that

$$\begin{aligned} &\mathbb{E}[c'_X(\widehat{X}_{t'+1})(\widehat{X}_{t'} - \widehat{X}_{t'+1})] \\ &= \mathbb{E}[(\widehat{X}_{t'} - \widehat{X}_{t'+1})\mathbb{E}\{c'_X(\widehat{X}_{t'+1}) \mid \mathcal{F}_{t'+1}\}] \\ &= 0 \end{aligned} \tag{11}$$

Rearranging (10) gives our first key result.

Table 8: Quantile loss on the backtest year along different dimensions. Values are relative performance versus baseline, with the best result for each dimension emphasized.

	MQ-CNN		MQT		MQT-NoDS		MQT-All	
Dimension	P50	P90	P50	P90	P50	P90	P50	P90
Overall	1.000	1.000	**0.962**	**0.915**	0.983	0.963	0.977	0.957
Seasonal Peak 1	1.000	1.000	**0.865**	**0.667**	0.881	0.688	0.908	0.729
Seasonal Peak 2	1.000	1.000	**0.925**	**0.850**	0.972	0.910	0.932	0.886
Seasonal Peak 3	1.000	1.000	**0.798**	**0.804**	0.803	0.872	0.815	0.830
Post-Peak Rampdown	1.000	1.000	0.968	**0.956**	0.983	0.996	**0.967**	0.975
Promotion Type 1	1.000	1.000	**0.915**	**0.601**	0.929	0.616	0.921	0.628
Promotion Type 2	1.000	1.000	**0.764**	**0.537**	0.805	0.561	0.861	0.640
Promotion Type 3	1.000	1.000	**0.901**	**0.746**	1.075	0.942	0.941	0.848

Theorem 7.3. *Let $c : \mathbb{R} \to \mathbb{R}$ be a convex function minimized at 0, which is differentiable everywhere except possibly at 0. Assume that the sequence of forecasts $\widehat{X}_1, \dots \widehat{X}_T$ satisfies* (8)*. If c is differentiable at 0 or if the distribution $X \mid \mathcal{F}_t$ is continuous with a differentiable probability density function, then*

$$\mathbb{E}[\mathrm{Vol}_{c_X}^{(t:T)}] = \mathbb{E}[c_X(\widehat{X}_t)] - \mathbb{E}[c_X(\widehat{X}_T)] \tag{12}$$

In other words, Theorem 7.3 says that the expected Bregman Volatility is equal to the expected accuracy gain, if the forecasts are optimal in the sense of satisfying (8). To operationalize this result, we point out that if $\mathrm{Vol}_{c_X}^{(t:T)}$ systematically deviates from $c_X(\widehat{X}_t) - c_X(\widehat{X}_T)$, then this indicates that the forecasts are *not* optimal and can potentially be improved. This is addressed in Section 8.1.

Theorem 7.3 covers the case where c is differentiable at zero or $X \mid \mathcal{F}_t$ is continuous. However, in some important cases neither of these conditions hold, e.g. where X represents discrete customer demand or inventory units and c is quantile loss. In Appendix B.1, we derive an alternative version of Theorem 7.3 which applies to non-differentiable c and discrete X, where the equality in (12) is replaced with a bound depending on the probability that $X = \widehat{X}_t$.

7.1 Examples of Bregman Divergence

We work out the Bregman Volatility for some key examples.

Square Loss The Square Loss is defined by $c(x) = x^2$. Then $D_{c_X}(\widehat{X}_t, \widehat{X}_{t+1}) = (\widehat{X}_t - \widehat{X}_{t+1})^2$ and the corresponding Bregman volatility is the quadratic variation $\mathrm{Vol}_{c_X}^{(t:T)} = \sum_{t'=t}^{T-1} (\widehat{X}_{t'} - \widehat{X}_{t'+1})^2$.

Quantile Loss The Quantile Loss is defined by $c(x) = (1-q)xI\{x > 0\} - qxI\{x < 0\}$ where $0 \le q \le 1$ is fixed. The Bregman Divergence for quantile loss is derived in Appendix B.2. The Bregman Divergence for quantile loss is the same for all $0 \le q \le 1$. Unlike the square loss, the Bregman divergence for quantile loss depends on the final realized value X, which is not observed until time $t = T_\infty$ and not just the forecasts $\widehat{X}_t$. However, we can still compute $\mathbb{E}[D_{c_X}(\widehat{X}_t, \widehat{X}_{t+1}) \mid \mathcal{F}_{t+1}]$ as described in Appendix B.2.

8 FORECAST REGRESSION AND DECODER SELF-ATTENTION

8.1 A Procedure to Improve Accuracy and Volatility

The previous section gives us a metric, the Bregman volatility, which can be used to diagnose forecast optimality (i.e. for an optimal forecast, the Bregman volatility should equal the accuracy gain). Now we present a procedure – *Forecast Regression* – to adjust the forecast $\widehat{X}_{t+1}$ to account for previous forecasts $\widehat{X}_1, \dots, \widehat{X}_t$.

Theorem 8.1. *Suppose we have a sequence of adjusted forecasts $\widetilde{X}_1, \dots, \widetilde{X}_T$ defined by $\widetilde{X}_1 = \widehat{X}_1$ and $\widetilde{X}_t = \sum_{t'=1}^{t} \beta_{t',t} \widehat{X}_{t'}$, where – denoting $\boldsymbol{b}_t = (\beta_{1,t}, \dots, \beta_{t,t})$ –*

$$\boldsymbol{b}_t = \operatorname*{argmin}_{\boldsymbol{b} \in \mathbb{R}^t} \mathbb{E}\big[c_X\big(\sum_{t'=1}^{t} \beta_{t',t} \widehat{X}_{t'}\big) \mid \mathcal{F}_t\big] \tag{13}$$

for $1 < t \le T$. Then (11)–(12) *will hold with $\widetilde{X}_{t'}, \widetilde{X}_{t'+1}$ in place of $\widehat{X}_{t'}, \widehat{X}_{t'+1}$ and $\sum_{t=1}^{T} \mathbb{E}[c_X(\widetilde{X}_t)] \le \sum_{t=1}^{T} \mathbb{E}[c_X(\widehat{X}_t)]$.*

Theorem 8.1 follows immediately from the definition of $\boldsymbol{b}_t$ and implies that linear regression can improve both forecast accuracy and volatility. For quantile loss, Forecast Regression - using quantile regression to solve (13) - can still be utilized to improve accuracy and Bregman Volatility, but the analogues of (11)–(12) from Theorem 8.1 do not hold exactly due to the non-differentiability of quantile loss. Instead, the equalities in (11)–(12) must be replaced with bounds. This is worked-out in Appendix B.1.

8.2 Decoder Self-Attention Optimizes Bregman Volatility

Here we justify the claim that the Decoder Self-Attention module in MQ-Transformer optimizes Bregman Volatility. Specifically, we show that the Decoder Self-Attention module is a generalization of the Forecast Regresson procedure, Equation (13). We fix a specific FTD, so that all forecasts and demand are for that FTD and for clarity we drop references to the forecasting horizon and position encoding. We represent the query and key using f_q and f_k to represent general non-linear functions computed by neural nets. Then the query is f_q(forecast, demand) and the key is f_k(forecast).

Table 9: Bregman volatility ablation study

Model	Pct of MQCNN's excess Bregman Vol
MQCNN	100%
MQT-NoDS	202%
MQT-All	63%
MQT	32%

We write the forecasts in the same notation as (13), i.e. $\widehat{X}_{t'}$ is the forecast produced by MQ-Transformer *before* the Decoder Self-Attention module. It is not required for the Bregman Volatility results to use the previous demand so we write a simplified Decoder Self-Attention Weight matrix without the demand term as $A^{simple}_{t,t'} = f_{q,t}(\widehat{X}_t)^T W_q^T W_k f_{k,t'}(\widehat{X}_{t'})$, where $A^{simple}_{t,t'}$ is the attention weight between weeks t and t' in the output. The output of the Decoder Self-Attention block is a set of adjusted forecasts $\widetilde{X}_t = \sum_{t' \le t} A^{simple}_{t',t} \widehat{X}_{t'}$. Rewriting the simplified Decoder Self-Attention optimization problem in the notation of Equation (13):

$$\boldsymbol{b}_t = \underset{\boldsymbol{b} \in \mathbb{R}^N}{\operatorname{argmin}} \mathbb{E}\Big[c_{QL}\Big(\sum_{t'=1}^{t} A^{simple}_{t',t}(\boldsymbol{b})\widehat{X}_{t'}\Big) \mid \mathcal{F}_t\Big] \tag{14}$$

Here we write the Attention matrix $A^{simple}_{t',t}$ as $A^{simple}_{t',t}(\boldsymbol{b})$ to emphasize the Attention matrix is computed by a neural net with a large number N of parameters, and we reinterpret the $\boldsymbol{b}$ coefficients as neural net weights of MQ-Transformer. Comparing (14) with (13), Decoder Self-Attention trains a much larger set of N parameters as weights of the neural nets represented as $A^{simple}_{t',t}$ compared with linear regression. This suggests that Decoder Self-Attention generalizes the Forecast Regression procedure and therefore optimizes Bregman Volatility. In practice Bregman Volatility will only be approximately satisfied: 1) in practice the cost function is only approximately minimized 2) the result holds in expectation, whereas for real data we average over many data points 3) there will be generalization error out-of-sample

9 CONCLUSIONS AND FUTURE WORK

We present three novel architecture enhancements that improve the state of the art MQ-Forecasters. We also introduce a new notion of Bregman Volatility that connects forecast accuracy and volatility. To our knowledge, this is the first work to link model architecture and forecast volatility. Together, these innovations produced significant improvements in accuracy and volatility – there was no tradeoff between these two desiderata, but rather they are closely linked. On a public retail forecasting task MQTransformer outperformed the baseline by 5% at P50, 11% at P90 and the previous state-of-the-art (TFT) by 38%. Our model also achieves large increases in throughput compared to existing transformer architectures for forecasting.

REFERENCES

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. arXiv:1409.0473

[2] Arindam Banerjee, Srujana Merugu, Inderjit S Dhillon, Joydeep Ghosh, and John Lafferty. 2005. Clustering with Bregman divergences. *Journal of Machine Learning Research* 6, 10 (2005).

[3] Konstantinos Benidis, Syama Sundar Rangapuram, Valentin Flunkert, Bernie Wang, Danielle Maddix, Caner Turkmen, Jan Gasthaus, Michael Bohlke-Schneider, David Salinas, Lorenzo Stella, et al. 2020. Neural forecasting: Introduction and literature overview. arXiv:2004.10240

[4] Robert L. Bray and Haim Mendelson. 2012. Information Transmission and the Bullwhip Effect: An Empirical Investigation. *Management Science* 58, 5 (2012), 860–875.

[5] Frank Chen, Zvi Drezner, Jennifer K. Ryan, and Divid Simchi-Levi. 2000. Quantifying the Bullwhip Effect in a Simple Supply Chain: The Impact of Forecasting, Lead Times, and Information. *Management Science* 46, 3 (2000), 436–443.

[6] Jianpeng Cheng, Li Dong, and Mirella Lapata. 2016. Long short-term memory-networks for machine reading. arXiv:1601.06733

[7] Yagmur Gizem Cinar, Hamid Mirisaee, Parantapa Goswami, Eric Gaussier, Ali Ait-Bachir, and Vadim Strijov. 2017. Position-based content attention for time series forecasting with sequence-to-sequence rnns. In *ICONIP*.

[8] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. 2019. Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context. In *ACL*. arXiv:1901.02860

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL-HLT*.

[10] Dheeru Dua and Casey Graff. 2017. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml

[11] Andrea Galassi, Marco Lippi, and Paolo Torroni. 2019. Attention, please! a critical review of neural attention models in natural language processing. arXiv:1902.02181

[12] Alberto Gasparin, Slobodan Lukovic, and Cesare Alippi. 2019. Deep Learning for Time Series Forecasting: The Electric Load Case. arXiv:1907.09207

[13] Neil Shephard Gerd Heber, Asger Lunde and Kevin K. Sheppard. 2009. Oxford-Man Institute's Realized Library.

[14] Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Noam Shazeer, Ian Simon, Curtis Hawthorne, Andrew M. Dai, Matthew D. Hoffman, Monica Dinculescu, and Douglas Eck. 2018. Music Transformer: Generating Music with Long-Term Structure. arXiv:1809.04281

[15] Sangyeon Kim and Myungjoo Kang. 2019. Financial series prediction using Attention LSTM. arXiv:1902.10877

[16] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *ICLR*.

[17] Shiyang Li, Xiaoyong Jin, Yao Xuan, Xiyou Zhou, Wenhu Chen, Yu-Xiang Wang, and Xifeng Yan. 2019. Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. In *NIPS*.

[18] Bryan Lim, Sercan O. Arik, Nicolas Loeff, and Tomas Pfister. 2019. Temporal Fusion Transformers for Interpretable Multi-horizon Time Series Forecasting. arXiv:1912.09363

[19] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. arXiv:1508.04025

[20] Bhaskar Pratim Mukhoty, Vikas Maurya, and Sandeep Kumar Shukla. 2019. Sequence to sequence deep learning models for solar irradiation forecasting. In *IEEE Milan PowerTech*.

[21] Rafaela C. Nascimento, Yania M. Souto, Eduardo Ogasawara, Fabio Porto, and Eduardo Bezerra. 2019. STConvS2S: Spatiotemporal Convolutional Sequence to Sequence Network for weather forecasting. arXiv:1912.00134

[22] Georg Ostrovski, Will Dabney, and Rémi Munos. 2018. Autoregressive quantile networks for generative modeling. In *International Conference on Machine Learning*. PMLR, 3936–3945.

[23] David Salinas, Valentin Flunkert, Jan Gasthaus, and Tim Januschowski. 2020. DeepAR: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting* 36, 3 (2020), 1181–1191.

[24] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. 2018. Self-Attention with Relative Position Representations. In *NAACL*. arXiv:1803.02155

[25] Shun-Yao Shih and Fan-Keng Sun and Hung-yi Lee. 2019. Temporal pattern attention for multivariate time series forecasting. *Machine Learning* 108, 8–9 (2019), 1421–1441.

[26] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. In *NIPS*.

[27] Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alexander Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. 2016. WaveNet: A Generative Model for Raw Audio. arXiv:1609.03499

[28] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, , and Illia Polosukhin. 2017. Attention is all you need. In *NIPS*.

[29] Ruofeng Wen and Kari Torkkola. 2019. Deep Generative Quantile-Copula Models for Probabilistic Forecasting. In *ICML Time Series Workshop*.

[30] Ruofeng Wen, Kari Torkkola, B. Narayanaswamy, and Dhruv Madeka. 2017. A multi-horizon quantile recurrent forecaster. In *NIPS Time Series Workshop*.

[31] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. 2015. Show, attend and tell: Neural image caption generation with visual attention. In *ICML*.

[32] Rose Yu, Stephan Zheng, Anima Anandkumar, and Yisong Yue. 2017. Long-term Forecasting using Higher Order Tensor RNNs. arXiv:1711.00073

A MQ-TRANSFORMER DETAILS

A.1 Attention Mechanisms

Attention mechanisms can be viewed as a form of content based addressing, that computes an alignment between a set of *queries* and *keys* to extract a *value*. Formally, let $\mathbf{q}_1, \ldots, \mathbf{q}_t$, $\mathbf{k}_1, \ldots, \mathbf{k}_t$ and $\mathbf{v}_1, \ldots, \mathbf{v}_t$ be a series of queries, keys and values, respectively. The s^{th} attended value is defined as $\mathbf{c}_s = \sum_{i=1}^{t} \text{score}(\mathbf{q}_s, \mathbf{k}_t)\mathbf{v}_t$, where score is a scoring function – commonly $\text{score}(\mathbf{u}, \mathbf{v}) := \mathbf{u}^\top \mathbf{v}$. In the vanilla transformer model, $\mathbf{q}_s = \mathbf{k}_s = \mathbf{v}_s = \mathbf{h}_s$, where $\mathbf{h}_s$ is the hidden state at time s. Because attention mechanisms have no concept of absolute or relative position, some sort of position information must be provided. Vaswani et al. [28] uses a sinusoidal positional encoding added to the input to an attention block, providing each token's position in the input time series.

In the vanilla transformer [28], a sinusoidal position embedding is added to the network input and each encoder layer consists of a multi-headed attention block followed by a feed-forward sub-layer. For each head i, the attention score between query q_s and key k_t is defined as follows for the input layer

$$A^h_{s,t} = (\mathbf{x}_s + \mathbf{r}_s)^\top \mathbf{W}^{h,\top}_q \mathbf{W}^h_k (\mathbf{x}_t + \mathbf{r}_t) \tag{15}$$

where $\mathbf{x}_s$, $\mathbf{r}_s$ are the observation of the time series and the position encoding, respectively, at time s. In Section 3 we introduced attention mechanisms that differ in their treatment of the position dependent biases

A.2 MQTransformer Architecture

We describe in detail the layers in MQTransformer, which is based off of the MQ-Forecaster framework [30] and uses a wavenet encoder [27] for time-series covariates. On different datasets, we consider the following variations: choice of encoding for categorical variables, a tunable parameter d_h (dimension of hidden layers), dropout rate p_{drop}, a list of dilation rates for the wavenet encoder, and a list of dilation rates for the position encoding. The ReLU activation function is used throughout the network.

A.3 Input Embeddings and Position Encoding

Static categorical variables are encoded using either one-hot encoding or an embedding layer. Time-series categorical variables are one-hot encoded, and then passed through a single feed-forward layer of dimension d_h. The global position encoding module takes as input the known time-series covariates, and consist of a stack of dilated, bi-directional 1-D convolution layers with d_h filters. After each convolution is a ReLU activation, followed by a dropout layer with rate p_{drop}, and the local position encoding is implemented as a single dense layer of dimension d_h.

A.4 Encoder

After categorical encodings are applied, the inputs are passed through the encoder block. The encoder consists of two components: a single dense layer to encode the static features, and a stack of dilated, temporal convolutions. The time-series covariates are concatenated with the position encoding to form the input to the convolution stack. The output of the encoder block is produced by replicating the encoded static features across all time steps and concatenating with the output of the convolution.

A.5 Decoder

Table 1 describes the blocks in the decoder. The dimension of each head in both the horizon-specific and decoder self-attention blocks is $d_h/2$. The dense layer used to compute $\boldsymbol{c}^a_t$ has dimension $d_h/2$. The output block is two layer MLP with hidden layer dimension $d_h/2$, and weights are shared across time points and horizons. The output layer has one output per horizon, quantile pair.

A.6 Large Scale Demand Forecasting Experiments

A.7 Training Procedure

We only consider tuning two hyper-parameters, size of hidden layer $d_h \in \{32, 64, 128\}$ and learning rate $\alpha \in \{1\times10^{-2}, 1\times10^{-3}, 1\times10^{-4}\}$. The model is trained using ADAM [16] with parameters $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 1e-8$ and a minibatch size of 256, for a maximum of 100 epochs and an early stopping patience of 5 epochs. We train a model for each hyperparameter setting in the search grid (6 combinations), select the one with the minimal validation loss and report the selected model's test-set error in Table 4.

A.8 Architecture Details

Our MQTransformer architecture contains a single tune-able hyperparameter – hidden layer dimension d_h. Dataset specific settings are used for the dilation rates. For static categorical covariates we use an embedding layer with dimension d_h and use one-hot encoding for time-series covariates. A dropout rate of 0.15 and ReLU activations are used throughout the network. The only difference between this variant and the one used for the non-public large scale demand forecasting task is the use of an embedding layer for static, categorical covariates rather than one-hot encoding.

A.9 Reported Model Parameters

The optimal parameters for each task are given in Table 10.

B BREGMAN VOLATILITY DETAILS

B.1 Non-Differentiable Cost Functions

We state an alternative version of Theorem 7.3 for non-differentiable cost functions and discrete random variables X.

Theorem B.1. *Assume that $c : \mathbb{R} \to \mathbb{R}$ is a convex function, which is minimized at 0 and differentiable everywhere except possibly at 0. Assume that the subgradient of c at 0 is contained in the interval $[-L, U]$, i.e. $\partial c(0) \subseteq [-L, U]$, for constants $L, U \geq 0$. Further assume that for each $t = 1, \ldots, T$, the distribution of $X \mid \mathcal{F}_t$ is discrete and supported on the points $x_{1,t}, \ldots, x_{K_t,t} \in \mathbb{R}$, and $\Pr\{X = x_{k,t} \mid \mathcal{F}_t\} = p_{k,t}$ for $k = 1, \ldots, K_t$. Then*

$$\mathbb{E}[\text{Vol}^{(t:T)}_{c_X}] = \mathbb{E}[c_X(\widehat{X}_t)] - \mathbb{E}[c_X(\widehat{X}_T)] + \text{err}, \tag{16}$$

where

$$|\text{err}| \leq (L + U) \sum_{t=1}^{T} \mathbb{E}\left[|\widehat{X}_{t-1} - \widehat{X}_t| p_{\widehat{k}_t,t} \right]$$

and $p_{\widehat{k}_t,t} = p_{k,t}$, if $\widehat{X}_t = x_{k,t}$ for some $k = 1, \ldots, K_t$, and 0 otherwise.

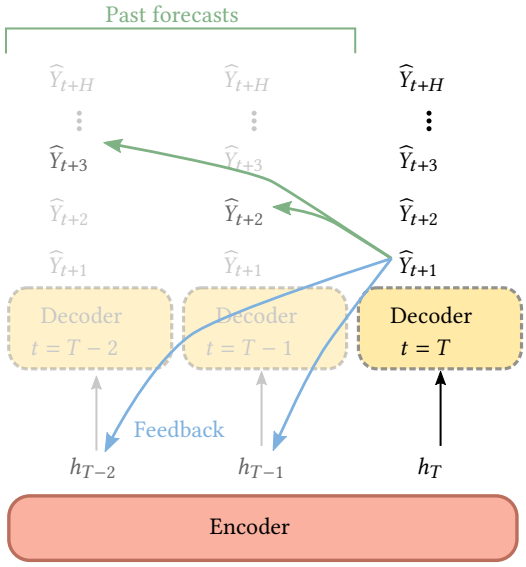


Figure 2: The decoder attends over past forecasts for the *same target horizon* – the context h_t contains feedback information (demand and other encoded signals), allowing the model to adjust forecasts that are too volatile or not volatile enough as the target date approaches.

Table 10: Parameter settings of reported MQTransformer model on each public dataset.

Name	d_h	α	Enc. Dilation Rates	Pos. Dilation Rates
Electricity	128	1×10^{-3}	[1,2,4,8,16,32]	[1,2,4,8,8]
Traffic	64	1×10^{-3}	[1,2,4,8,16,32]	[1,2,4,8,8]
Volatility	128	1×10^{-3}	[1,2,4,8,16,32,64]	[1,1,2]
Retail	64	1×10^{-4}	[1,2,4,8,16,32]	[1,2,4,8,14]

Next we work out how to apply Theorem B.1 for quantile loss and the Forecast Regression methodology, in the case where replicate observations and forecasts are available. Assume that c is quantile loss, i.e. $c(x) = (1-q)xI\{x > 0\} - qxI\{x < 0\}$ for some fixed $0 \le q \le 1$. Then c is non-differentiable at 0 and $\partial c(0) = [-q, 1-q]$. Assume further that we have replicates $X^{(i)}$, $i = 1, \ldots, N$ and corresponding forecasts $\widehat{X}_t^{(i)} \in \mathcal{F}_t$, $t = 1, \ldots, T$. As in Section 8.1, we consider the alternative forecasts $\widetilde{X}_t^{(i)} = \sum_{s=1}^{t} \beta_{t',t} \widehat{X}_{t'}^{(i)}$, where $\boldsymbol{b}_t = (\beta_{1,t}, \ldots, \beta_{t,t})$ satisfies

$$\boldsymbol{b}_t \in \operatorname*{argmin}_{\boldsymbol{b} \in \mathbb{R}^t} \frac{1}{N} \sum_{i=1}^{N} c_{X^{(i)}} \left(\sum_{t'=1}^{t} \beta_{t',t} \widehat{X}_{t'}^{(i)} \right).$$

Then, by (16),

$$\frac{1}{N} \sum_{i=1}^{N} \left\{ \mathrm{Vol}_{c_{X^{(i)}}}^{(1:T)} (\widetilde{X}_1^{(i)}, \ldots, \widetilde{X}_T^{(i)}) - [c_{X^{(i)}}(\widetilde{X}_1^{(i)}) - c_{X^{(i)}}(\widetilde{X}_T^{(i)})] \right\} = \mathrm{err},$$

where

$$|\mathrm{err}| \le \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T-1} |\widetilde{X}_t^{(i)} - \widetilde{X}_{t+1}^{(i)}| I\{\widetilde{X}_{t+1}^{(i)} = X^{(i)}\}.$$

B.2 Quantile Loss Bregman Divergence

We provide the result for Bregman divergence for quantile loss.

$$D_{c_X}(\widehat{X}_t, \widehat{X}_{t+1}) = \begin{cases} |\widehat{X}_t - X| & \text{if } \mathrm{sgn}(\widehat{X}_t - X) = -\mathrm{sgn}(\widehat{X}_{t+1} - X), \\ 0 & \text{if } \mathrm{sgn}(\widehat{X}_t - X) = \mathrm{sgn}(\widehat{X}_{t+1} - X), \\ c_X(\widehat{X}_t) & \text{if } \mathrm{sgn}(\widehat{X}_t - X)\mathrm{sgn}(\widehat{X}_{t+1} - X) = 0. \end{cases} \tag{17}$$

Now let F_t denote the cumulative distribution function of $X \mid \mathcal{F}_t$. The expected Bregman divergence is

$$\mathbb{E}[D_{c_X}(\widehat{X}_t, \widehat{X}_{t+1}) \mid \mathcal{F}_{t+1}] = \int_{\widehat{X}_{t+1}}^{\widehat{X}_t} F_{t+1}(u) - F_{t+1}(\widehat{X}_{t+1})\, du + (1-q)(\widehat{X}_t - \widehat{X}_{t+1}) \Pr\{X - \widehat{X}_{t+1} \mid \mathcal{F}_{t+1}\}$$

This shows that we can still compute $\mathbb{E}[D_{c_X}(\widehat{X}_t, \widehat{X}_{t+1}) \mid \mathcal{F}_{t+1}]$ even though for quantile loss the Bregman volatility depends on the final demand. Note that the quantity on the right-hand side, the expected Bregman divergence for quantile loss is similar to the quantile divergence defined in [22] and studied in [29] and elsewhere.