Improving Time Series Forecasting with Mixup Data Augmentation

Yun Zhou[1], Liwen You[2], Wenzhen Zhu[2], and Panpan Xu[2]

[1] Amazon Generative AI Innovation Center, USA
[2] Amazon Titan Lab, USA
{yunzzhou, yoliwen, wenzhu, xupanpan}@amazon.com

Abstract. *Mixup* is a domain-agnostic approach for data augmentation, originally proposed for training Deep Neural Networks (DNNs) for image classification. It obtains additional data for training by sampling from linear interpolations of model inputs and their labels. While proven to be effective for computer vision (CV) and natural language processing (NLP) tasks, it remains unknown if *mixup* can bring performance improvement for DNNs developed for forecasting tasks. We propose several different approaches for applying mixup to train neural forecasters and evaluate them on benchmark datasets. The study highlights the capability of *mixup* in enhancing model performance across a wide range of hyper-parameter settings in specific datasets, and paves the way for *mixup*'s potential success in broader time series forecast use cases.

Keywords: time series forecast · neural network forecaster · mixup · data augmentation.

1 Introduction

Recent advances in neural forecasting have brought state-of-the-art performance for many practical forecasting tasks [2]. However, training performant neural forecasters relies on the availability of large-scale historical data, which may not be possible in many real-world applications. Typical approaches used in deep learning to address low data resource problems include using pretrained models and adding data augmentations, both of which are highly effective for computer vision (CV) and natural language processing (NLP) tasks. In this work, we focus on developing data augmentation techniques for neural forecasters, in particular, the *mixup* technique.

First proposed by Zhang et al. [13], *mixup* is originally developed for image classification tasks in CV. It augments the original training data by linearly interpolating between two training inputs (blending two images) and their corresponding labels. The method is simple to implement and DNNs trained with *mixup* have obtained noticeable performance improvement on image classification benchmarks. Recently, researchers also applied *mixup* to NLP tasks by interpolating word embeddings instead of raw inputs [6]. It seems straightforward that we can apply *mixup* to train neural forecasters as well. However, there are some fundamental differences in the problem setting. First of all, most of the research on *mixup* focuses on supporting classification tasks instead of regression tasks. Second, many neural forecasters take in a mixture of numerical and categorical variables, which is different from both CV and NLP applications. Therefore these two questions arise: 1) is *mixup* an effective data augmentation strategy for training neural forecasters? 2) how to apply *mixup* when the input contains heterogeneous data types with both numerical and categorical variables?

In this work we aim to answer these questions via empirical study using a set of benchmark datasets. We evaluate several alternative ways of adding *mixup* in a forecasting setting, in order to find the most effective combination: 1) *mixup* can be applied directly to the original time series, or in the embedding space and 2) *mixup* can be applied directly to the forecast targets, or the losses computed separately using the original targets. We limit the current study to the Transformer architecture as it demonstrates state-of-the-art performance on many benchmark datasets [14]. However the methodology proposed in the paper can be applied to other model architectures such as CNN-QR [12] and DeepAR [10] as well, which we leave as future work. Our contributions are summarized as follows:

- We systematically investigated a variety of *mixup* strategies for training neural forecasters and identified the most effective combinations. To the best of our knowledge, this the first work that explores the effectiveness of *mixup* in training neural forecasters.
- We demonstrated that *mixup* has the capability to improve neural forecaster performance, while acknowledging the extent of the improvement is highly data dependent. *Mixup*'s data dependency has been observed in previous work and remains an active research field [4]. As *mixup* is straightforward to implement, our work recommends adding it as part of the toolbox for developing custom neural forecasting models. Our work may also inspire more researchers to explore the relation between *mixup* and data dependent characteristics.

2 Problem setting and notation

Time series forecasting aims to predict values of targets $y_{i,t+1:t+k}$ for time series $y_{i,\cdot}$, $i \in [1, n]$ at a multi-step time window $[t+1, t+k]$. Here, we focus on multi-step forecasting where $k > 1$ and uni-variate targets $y_{i,t} \in R^1$: $\hat{y}_{i,t+1:t+k} = f(y_{i,t-w+1:t}, x_{i,t-w+1:t}, m_i)$. Here, f is the forecasting model, w is the lookback or context window, and x is the covariate. m_i includes static meta-data for the i-th time series, such as product category in sales forecasting problems. For neural forecasters, each training sample contains a sliding window of length $w + k$.

The classic *mixup* routine constructs new training data points using a convex combination of two randomly drawn original samples u_i and u_j, and their corresponding labels v_i and v_j:

$$\tilde{u} = \lambda u_i + (1-\lambda)u_j, \tilde{v} = \lambda v_i + (1-\lambda)v_j, \tag{1}$$

where $(\tilde{u}, \tilde{v})$ is the new sample for training. λ can be a fixed number in $(0, 1)$ or randomly drawn from a beta distribution $Beta(\beta_1, \beta_2)$ where $\beta \in (0, \infty)$ [13].

In the problem setting of time series forecasting, $u_i = (y_{l,t-w+1:t}, x_{l,t-w+1:t}, m_l)$ and $v_i = y_{l,t+1:t+k}$, where $l \in [1, n]$. Since $x_{l,t-w+1:t}$ and m_l could contain categorical variables, we cannot directly apply Equation. 1 to obtain new training sample (u, v). We introduce two different variations of *mixup* training strategy to address this challenge in the next section.

As a first step of the investigation we focus on applying *mixup* to Transformer models, which recently has achieved state-of-the-art performance on benchmark datasets[9,8,14]. In particular, we built on the Informer model proposed in [14]. We modified the original architecture with an additional embedding layer to take in categorical variable inputs.

3 Mixup data augmentation

Several different ways to do data augmentation with *mixup* techniques are introduced in e.g., [13,5,3,11]. In this section, we describe the *mixup* strategies implemented in the paper, and how they are applied to time series forecasting problems. We note that the *mixup* here are not exhaustive, and the results may be further improved by incorporating other strategies as well.

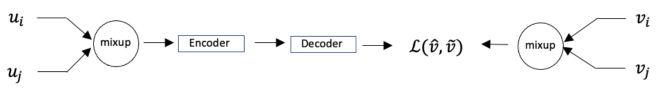
3.1 Static vs. dynamic mixup

The original *mixup* linearly interpolates data points and corresponding labels such that the trained neural networks have better generalization [13]. The scale and diversity of newly augmented data is thus of the first consideration.

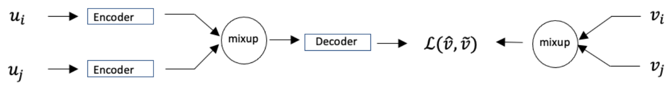
One way is to generate a fixed number of *mixup*-augmented data before training starts, e.g., 200% of the original data size n. Specifically, a convex combination of two original time series is (randomly) generated, then repeated $2n$ times. Each training epoch uses the same set of $2n$ augmented samples for training. We refer to it as the ***static mixup***. Its advantages are easy implementation and high reproducibility. However, the diversity of augmented data is limited as there will only be a fixed number of combinations of the original data, potentially restraining regularization effects on the model.

We adopt a different approach, termed ***dynamic mixup***, to alleviate the diversity issue by generating augmented data samples on-the-fly during the entire training process. Specifically, it generates new data points for every batch and epoch during training, using freshly drawn mixing weights $\lambda \sim Beta(\beta_1, \beta_2)$ and training data pair $(u_i, v_i), (u_j, v_j)$. (u, v) pairs can be drawn from any sliding windows in all training time series, e.g., $u_i = (x_{l,t-w+1:t}, y_{l,t-w+1:t}, m_l)$, $v_i = (y_{l,t+1:t+k})$, while $u_j = (x_{l',t-w+1:t}, y_{l',t-w+1:t}, m'_l)$, $v_j = (y_{l',t+1:t+k})$, $\forall l \neq l' \in$

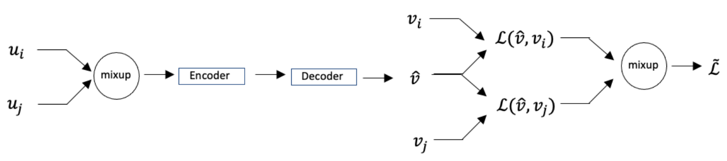
$[1, n]$. The diversity of the newly generated data is determined by the total number of data fed into model training, and can be easily adjusted to obtain richer augmentation. Note that if there is categorical features, *mixup* by Equation 1 can not be directly applied and we need to first transform categorical feature into numerical embedding first.



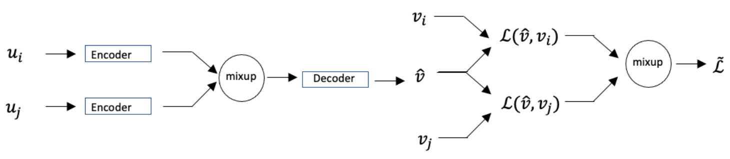
(a) input + label mixup



(b) latent space + label mixup



(c) input + loss mixup



(d) latent space + loss mixup

Fig. 1: **Mixup strategy illustration.** We illustrate four different mixup strategies for forecasting applied to a Transformer with encoder and decoder blocks. **(a)** shows the input and label mixup combination; **(b)** shows the latent space and label mixup combination; **(c)** shows the input and loss mixup combination and **(d)** shows the latent space and loss mixup combination.

3.2 Input mixup vs. latent space mixup

Neural forecasters take in both numerical features and categorical features. It is natural to *mixup* numerical features on either space. However for categorical feature, we cannot simply add them together based on equation 1. Therefore, for categorical data, we perform *mixup* on categorical features' embeddings. This is commonly seen in areas such as NLP where word embeddings are used for mixup training [11]. The numerical and categorical input mixup results are combined and fed into the model together. We refer to this as ***input mixup***, as illustrated in Figure 1 (a) and (c).

An alternative approach which we refer to as ***latent space mixup*** directly mixup Transformer encoder outputs, as illustrated in Figure 1 (b) and (d). Formally, it can be denoted as: $\tilde{g}(u) = \lambda \cdot g(u_i) + (1-\lambda) \cdot g(u_j)$, $\tilde{v} = \lambda v_i + (1-\lambda) v_j$, where $g(u_i)$ and $g(u_j)$ are the transformer encoder outputs. This method can naturally handle heterogeneous input data types. By applying mixup on the latent space, it is an end-to-end training process, where the latent space constantly changes during the training process.

3.3 Label mixup vs. Loss mixup

Besides mixup the labels (forecast targets in the paper), an alternative approach applies mixup on the losses calculated separately using the original pair of labels [3], as illustrated in Figure. 1 (c) and (d). In this paper, we experimented with ***loss mixup***, which can be formally denoted as: $\tilde{\mathcal{L}} = \lambda \mathcal{L}(\hat{v}, v_i) + (1-\lambda)\mathcal{L}(\hat{v}, v_j)$, where $\hat{v}$ is the forecast results from the output of decoder, and v_i and v_j are the original forecast targets. $\mathcal{L}$ could be any loss function commonly used for training neural forecasters. In our experiments, we used weighted *quantile loss* as $\mathcal{L}$. In contrast, the loss function directly using ***label mixup*** can be denoted as: $\tilde{\mathcal{L}} = \mathcal{L}(\hat{v}, \lambda v_i + (1-\lambda) v_j)$, as illustrated in Figure. 1 (a) and (b).

4 Experiments

4.1 Settings

In dynamic *mixup*, we may control the frequency the model sees *mixup* augmented data instead of the original data. To make comparable experiments, we set the number of training samples in a vanilla training (no *mixup*) as 1x. Then we tried training with (1) 1x original data plus 1x *mixup* data (total 2x), or (2) 1x original data plus 3x *mixup* data (total 4x). We also experimented different values of (β_1, β_2) as the Beta distribution of mixing weight λ is significantly data dependent based on discussions in previous work such as [3]. Lastly, we combined input/latent space *mixup* and label/loss *mixup*, using the four configurations illustrated in Figure. 1.

The following hyper-parameters are fixed the whole time. The batch size is 64, the context length in time series is 168, and the prediction length is 24. In the Informer architecture, the embedding dimension is 512, the start token length in decoder is 24, the number of encoder layer is 3, the number of decoder layer is 2. For training we used *quantile loss* and scaled each time series using its historical mean value before feeding it into the model and computing the training loss. The final forecasts are rescaled back to compute the accuracy metrics, using the *MeanScaler* implemented in GluonTS[1]. The input *mixup* is performed after normalization. For all the experiments, we report weighted *Quantile Loss* (wQL) at 10%, 50%, and 90%. Here, wQL is a commonly used metric in time series forecast which measures the accuracy of a model at any specified quantile. It is particularly useful when there are different panelties for under-predicting and over-predicting with adjustable quantiles.

4.2 Datasets

The experiments were applied to three public datasets, *traffic*, *electricity*, and *wiki-rolling* data [7], which are directly available from GluonTS [1]. *Traffic* contains the hourly occupancy rate measured by sensors at San Francisco bay area freeways with $n = 862$ entities. *Electricity* data contains the hourly energy assumption for $n = 321$ entities within year 2012 - 2014. *Wiki-rolling* contains $n = 9535$ entities of daily records starting from year 2012. In every dataset, the last sliding windows of each time series were used as test sets and the other sliding windows were used as training sets.

4.3 Results

Figure 2 plots the test error in the *traffic* dataset, and Table 1 provides more detailed numbers with comparison to the baseline results. To simplify the comparison between different *mixup* strategies, we show the results for the *traffic* dataset in Table 1 when mixing weights $\lambda \sim Beta(0.1, 0.1)$. Similar observation can be extended to other datasets and other *Beta* distributions.

Table 1: Comparing *mixup* results with baseline on *traffic* dataset, using weighted Quantile Loss (wQL). Result shows that *mixup* consistently improves the baseline forecasting performance for this dataset. Best metrics in each row are in bold (except for training loss which are not comparable).

Metric: median	Mixup probability		Mixup method				
	Augmentation	Beta	Input		Latent		No mixup (baseline)
			label mixup	loss mixup	label mixup	loss mixup	
Training Loss	2x	0.1,0.1	0.0610	0.0560	0.0570	0.0570	0.0580
	4x	0.1,0.1	0.0570	0.0560	0.0570	0.0540	
WQL10	2x	0.1,0.1	0.0836	0.0829	**0.0821**	0.0829	0.0842
	4x	0.1,0.1	0.0852	0.0825	**0.0808**	0.0822	
WQL50	2x	0.1,0.1	0.1424	0.1427	**0.1419**	0.1443	0.1450
	4x	0.1,0.1	0.1448	0.1414	**0.1388**	0.1404	
WQL90	2x	0.1,0.1	0.0798	**0.0728**	0.0747	0.0751	0.0779
	4x	0.1,0.1	0.0744	0.0720	**0.0719**	0.0777	

Compared to the baseline, combining latent embedding space *mixup* and label *mixup* in 4x augmentation data achieved the best test error, as measure by weighted quantile loss at 10%, 50%, and 90% (with a relative improvement over the baseline for 4%, 4.3%, and 7.7%, respectively). If compared within *mixup* strategies we experimented, combining latent embedding space *mixup* and label *mixup* is the routine that allows uniformly improvement over baseline, and is better than all other *mixup* routines except weighted quantile loss at 90% with 2x data augmentation. More data augmentation (from 2x to 4x) further improves the performance as we expected.

As can be seen in Figure 2, all different *mixup* strategies generally improve the baseline results measured by wQL at 10%, 50%, and 90%, except for a few hyper-parameter settings. We'd like to point out that only one of the settings (relate to one point in Figure 2) will be deployed when using *mixup* in reality. We graphed all our experiments for comprehensive evaluation, and should not require every *mixup* setting to improve the baseline. Figure 2 indicates that the *mixup* data augmentation can be successfully employed in the time series forecasting problem to enhance model performance, in addition to its effectiveness in NLP and CV use cases. Our implemented *mixup* strategies show general improvement on 90% wQL in the *electricity* data and 10% wQL in the *wiki-rolling* data, as illustrated in Figure 3. Furthermore, under 2x data augmentation, combining input and label *mixup* provides improvement over all baseline metrics when $\lambda \sim Beta(5,5)$ in the *electricity* data, while input *mixup* alone can provide the universal improvement when $\lambda \sim Beta(0.1, 0.1)$ in the *wiki-rolling* data. It demonstrates that the effectiveness of *mixup* is also data dependent as noted in [3,4]. Further study on *mixup*'s relation to data dependent characteristics is needed in this field in order to universally improve all metrics, and we discuss this as the future work in the last section.

5 Conclusion

In this paper, we demonstrated that *mixup* has the capability to improve neural forecaster performance, acknowledging the extent of the improvement is data dependent, as observed in previous work on *mixup*. As *mixup* is straightforward to implement, we recommend considering adding it as part of the toolbox for developing custom neural forecasting models. We also hope our work can inspire more exploration on the relation between *mixup* and data dependent characteristics.

Discussion and future works Our work is an initial investigation on applying mixup for training neural forecasters and there are a lot of remaining future work. One important next step is to experiment with a wider range of datasets and neural forecaster models such as CNN-QR and DeepAR. Besides that, we also plan to investigate and develop systematic methods to select the best mixup strategy and hyperparameters (e.g. β), potentially through the use a backtest window.

As we dynamically generate *mixup* during each epoch, the training process of the network might not be stable. In order to make network parameter learning process more smooth, we could freeze the *mixup* data generation process, and let

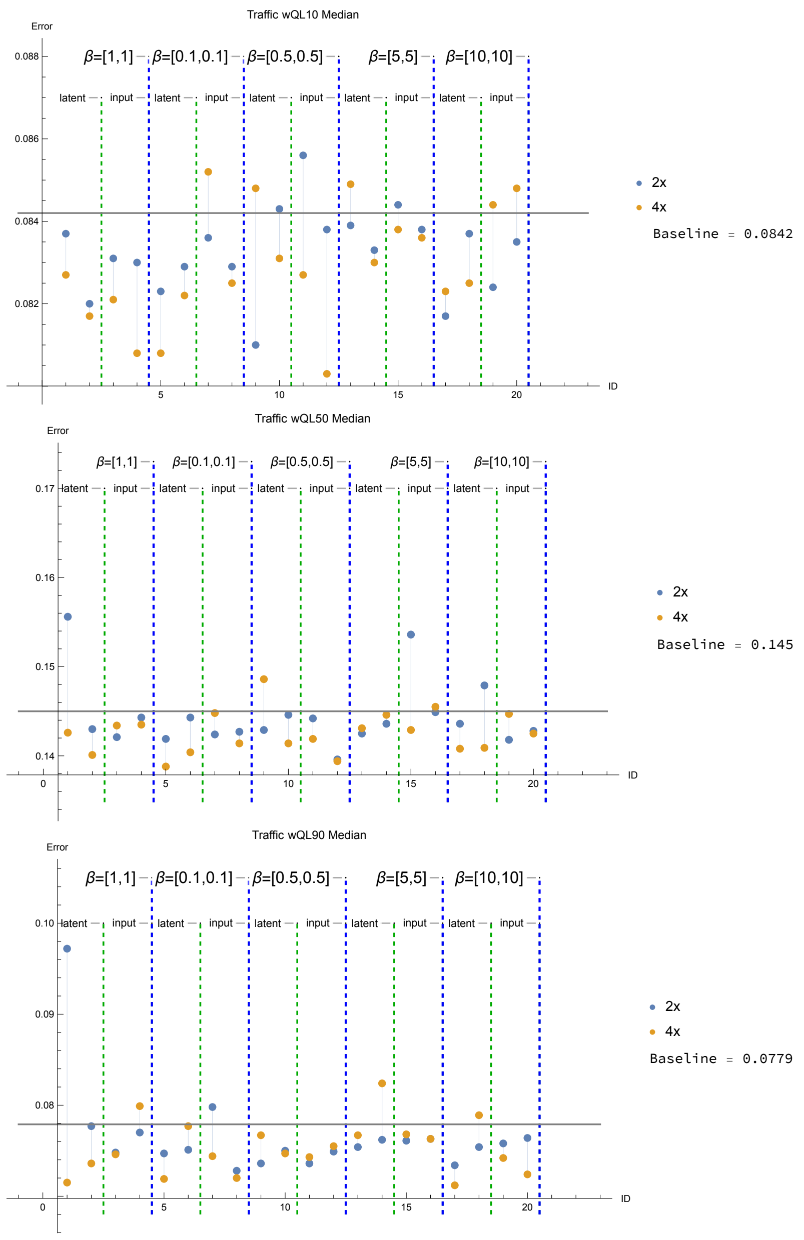


Fig. 2: **Model performance (measured using wQL) with different experimental settings on *traffic* data.** Median of three trials in each setting are shown. Number of augmented data samples are indicated using blue (2x) or yellow (4x) color. The blue dashed line separate trials with different β. The green dashed line indicated the *mixup* on input or latent space. Results from trials with label *mixup* are plotted on odd experiment ID, results from trials with loss *mixup* are plotted on even experiment ID. Baseline results with no mixup are shown in horizontal lines. A wide range of hyper-parameter settings were implemented, and almost all improve the baseline results, with little need to select or fine tune the *mixup* parameters.

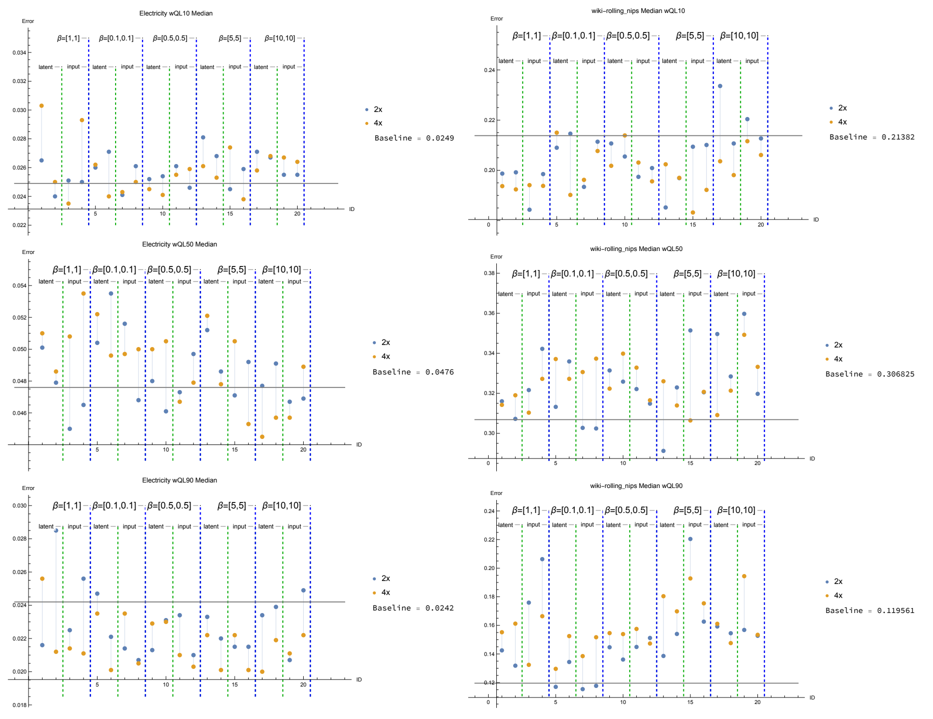


Fig. 3: **Model performance (measured using wQL) with different experimental settings on *electricity* (left) and *wiki-rolling* (right) datasets.** Median of three trials in each setting are shown. Number of augmented data samples are indicated using blue (2x) or yellow (4x) color. The blue dashed line separate trials with different β. The green dashed line indicated the *mixup* on input or latent space. Results from trials with label *mixup* are plotted on odd experiment ID, results from trials with loss *mixup* are plotted on even experiment ID. Baseline results with no mixup are shown in horizontal lines. A wide range of hyper-parameter settings were implemented, general improvement over baseline was seen at wQL90 in *electricity* dataset and wQL10 in *wiki-rolling* dataset. For other metrics, we observe multiple improvement by *mixup* strategies, but acknowledge the need to further select or fine tune the *mixup* hyper-parameters.

network train for a few epochs with the same data and then generate another set of *mixup* data iteratively.

Data augmentation increases sample size and makes model generalize better. However, data augmentation also introduces noise into the dataset, therefore we might need to use smaller learning rate and longer training epochs to train the model.

During *mixup*, we could select which two times series to *mixup* based on similarity of them instead of just randomly pick two times series at random time segments. For example, we can pick pairs from those incorrectly predicted with high probability. Furthermore, we do not need to use all *mixup* data to update model weight, instead only using those with high prediction confidence.

References

1. Alexandrov, A., Benidis, K., Bohlke-Schneider, M., Flunkert, V., Gasthaus, J., Januschowski, T., Maddix, D.C., Rangapuram, S., Salinas, D., Schulz, J., et al.: Gluonts: Probabilistic time series models in python. arXiv preprint arXiv:1906.05264 (2019)
2. Benidis, K., Rangapuram, S.S., Flunkert, V., Wang, B., Maddix, D., Turkmen, C., Gasthaus, J., Bohlke-Schneider, M., Salinas, D., Stella, L., et al.: Neural forecasting: Introduction and literature overview. arXiv preprint arXiv:2004.10240 (2020)
3. Chang, O., Tran, D.N., Koishida, K.: Single-channel speech enhancement using learnable loss mixup. Proc. Interspeech 2021 pp. 2696–2700 (2021)
4. Chidambaram, M., Wang, X., Hu, Y., Wu, C., Ge, R.: Towards understanding the data dependency of mixup-style training. arXiv preprint arXiv:2110.07647 (2021)
5. DeVries, T., Taylor, G.W.: Dataset augmentation in feature space. arXiv preprint arXiv:1702.05538 (2017)
6. Guo, H., Mao, Y., Zhang, R.: Augmenting data with mixup for sentence classification: An empirical study. arXiv preprint arXiv:1905.08941 (2019)
7. Lai, G., Chang, W.C., Yang, Y., Liu, H.: Modeling long-and short-term temporal patterns with deep neural networks. In: The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval. pp. 95–104 (2018)
8. Li, S., Jin, X., Xuan, Y., Zhou, X., Chen, W., Wang, Y.X., Yan, X.: Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. Advances in Neural Information Processing Systems **32** (2019)
9. Lim, B., Arık, S., Loeff, N., Pfister, T.: Temporal fusion transformers for interpretable multi-horizon time series forecasting. International Journal of Forecasting **37**(4), 1748–1764 (2021). https://doi.org/https://doi.org/10.1016/j.ijforecast.2021.03.012, `https://www.sciencedirect.com/science/article/pii/S0169207021000637`
10. Salinas, D., Flunkert, V., Gasthaus, J., Januschowski, T.: Deepar: Probabilistic forecasting with autoregressive recurrent networks. International Journal of Forecasting **36**(3), 1181–1191 (2020)
11. Sun, L., Xia, C., Yin, W., Liang, T., Yu, P.S., He, L.: Mixup-transformer: dynamic data augmentation for nlp tasks. arXiv preprint arXiv:2010.02394 (2020)
12. Wen, R., Torkkola, K., Narayanaswamy, B., Madeka, D.: A multi-horizon quantile recurrent forecaster. arXiv: Machine Learning (2017)
13. Zhang, H., Cisse, M., Dauphin, Y.N., Lopez-Paz, D.: mixup: Beyond empirical risk minimization. arXiv preprint arXiv:1710.09412 (2017)

14. Zhou, H., Zhang, S., Peng, J., Zhang, S., Li, J., Xiong, H., Zhang, W.: Informer: Beyond efficient transformer for long sequence time-series forecasting. In: Proceedings of AAAI (2021)