Augmenting Imbalanced Time-series Data via Adversarial Perturbation in Latent Space

Beomsoo Kim BOMBSKIM@KAIST.AC.KR
KAIST, Daejeon, Republic of Korea

Jang-Ho Choi JANGHOCHOI@ETRI.RE.KR
ETRI, Daejeon, Republic of Korea

Jaegul Choo JCHOO@KAIST.AC.KR
KAIST, Daejeon, Republic of Korea

Editors: Vineeth N Balasubramanian and Ivor Tsang

Abstract

Success of training deep learning models largely depends on the amount and quality of training data. Although numerous data augmentation techniques have already been proposed for certain domains such as computer vision where simple schemes such as rotation and flipping have been shown to be effective, other domains such as time-series data have a relatively smaller set of augmentation techniques readily available. Data imbalance is a phenomenon often observed in real-world data. However, a simple oversampling technique may make a model vulnerable to overfitting, so a proper data augmentation is desired. To tackle these problems, we propose a novel data augmentation method that utilizes the latent vectors of an autoencoder in a novel way. When input data are perturbed in its latent space, their reconstructed data retain properties similar to the original one. In contrast, adversarial augmentation is a technique to train robust deep neural networks against unforeseen data shifts or corruptions by providing a downstream model with samples that are difficult to predict. Our method adversarially perturbs input data in its latent space so that the augmented data is diverse and conducive to reducing test error of a downstream model. The experimental results demonstrated that our method achieves the right balance, significantly modifying the input data to help generalization while retaining their realism.

Keywords: time-series data augmentation, adversarial augmentation, autoencoders, data imbalance.

1. Introduction

Most successful applications of deep learning models are backed by a large amounts of quality data. Large datasets help models to learn more diverse features and decrease differences the between training and test error by keeping models from overfitting. As a general tendency, the performance of a model is improved by increasing the number of sample data. For example, in one experiment a ResNet model trained on billions of images outperformed the other models that had the same architecture, but were trained on smaller datasets Yalniz et al. (2019).

However, in many cases, acquiring quality data is not a cheap process. While more and more large and refined datasets are being made publicly available to promote research in various domains, quality datasets still largely fall short when it comes to solving existing or new practical problems. In particular, if one tries to train a model in a supervised manner, the problem becomes worse. This is because acquiring labeled data is more costly than acquiring unlabeled data. Most data is originally stored in an unlabeled format, unless it has been gathered in order to compose a labeled dataset in the first place. Also, even after labeling data, a dataset tends to exhibit other problems such as class imbalance. For example, weather data collected from a warm region is much more likely to contain an observation of rain than snow.

One popular approach to alleviate this problem is data augmentation, which is a way of increasing the amount of data by leveraging the existing dataset. Numerous data augmentation methods have been proposed for training deep learning models. For instance, random cropping, resizing, and flipping methods can be applied to most models that receive images as input. However, such domain-specific transformations cannot easily be applied to other domains, such as speech recognition or natural language processing.

Recently, a domain-agnostic data augmentation method using autoencoders was invented DeVries and Taylor (2017). The method does not rely on domain specific transformation but simply perturbs, interpolates, or extrapolates between given samples in the latent space of an autoencoder. The method was applied to five different datasets from different domains and helped enhance model accuracies.

Another recent approach for data augmentation is adversarial augmentation Peng et al. (2018); Zhang et al. (2019). Adversarial augmentation can be used to find the best augmentation policy to perform image transformation along with the training process, rather than generate new images from scratch. However, this approach requires a predetermined set of domain-specific functions and cannot be easily transferred to other domains. Note that without transformation functions that can guarantee the preservation of certain properties in an existing sample, the adversarial augmentation process may infinitely increase the training loss and the target may not converge.

In this paper, we propose a novel method that combines the two aforementioned approaches, namely, data perturbation in latent space and adversarial data augmentation. First, we convert an input vector into a latent vector using the encoder of an autoencoder. Then we adversarially perturb the latent vector. The perturbed vector then goes through a decoder and this results in a similar vector that is expected to give a larger loss value than the original input vector when given to a target model. The method can be used for vastly different datasets with little modifications. We apply this method to time-series data, where there are fewer number of readily applicable data augmentation functions compared to image data. Our method performed consistently better in the experiments than random perturbation, as was suggested by DeVries and Taylor (2017).

2. Related Work

2.1. Basic Augmentations for Time-series Data

This section introduces popular transformation functions for time-series data that do not rely on deep learning models. Window warping transforms a given sequence by randomly selecting a part of the sequence and speeding it up or down Le Guennec et al. (2016). Normally, the data is warped using a spline interpolation of an order of three or greater. This method can be seen as a one-dimensional variant for cropping and resizing augmentation for image data. Although it is only a simple function, many time-series models have shown improved performances after applying this method.

Another augmentation method involves injecting small noises that follow certain distributions into the input data. For example, Gaussian random noise or step-like trend noise can be added to input data to synthesize new data from existing ones Wen et al. (2020).

SMOTE (Synthetic Minority Over-sampling Technique) is a widely used augmentation method for imbalanced data Chawla et al. (2002). In SMOTE, two samples from a minority class are interpolated to generate a new sample. This method is widely used in many domains other than time-series. Note that this method is not always applicable to times-series data. For example, averaging two sine waves with different periods does not in general result in a sine wave. Also, the dataset to be augmented has to be labeled.

2.2. Model-based Augmentations

More advanced augmentation techniques that rely on deep learning models have been proposed for time-series data. Recently, a domain-agnostic data augmentation method was proposed by DeVries and Taylor (2017). First, the encoder of an autoencoder transforms an input sequence into a latent vector. Then the latent vector is changed with either simple random perturbation, interpolation, or extrapolation between given samples. As mentioned earlier for SMOTE, interpolating two sequences involves some risk that the resulting sequence does not preserve some important properties of the original sequences. This problem can be significantly alleviated if the interpolation is done in latent space not in input space, since the decoder of an autoencoder tends to restore the desired property of the original input sequence even after a significant perturbation.

On the other hand, a method called adversarial augmentation has been applied in many situations. Adversarial augmentation is a broad term that refers to perturbing training examples such that a downstream model gives a higher loss when fed with the perturbed example. There are several motivations for applying adversarial augmentation in training. Most notably, adversarial augmentation can be used to defend an adversarial attack Tramer and Boneh (2019). An adversarial attack means applying small but critical perturbations to input data, so that the perturbed input results in the model giving an incorrect output with high confidence Goodfellow et al. (2014).

However, defense against an adversarial attack is not the only use case for adversarial augmentation and there are numerous different motivations for using adversarial augmentation for training. For example, adversarial augmentation can be used for improved generalization with respect to unseen domains Volpi et al. (2018). Even more impressively, it has been shown that adversarial augmentation can reduce test time error if applied properly Peng et al. (2018); Zhang et al. (2019). One may think the idea of improving test error by synthesizing data that causes larger loss in training seems contradictory. One necessary assumption for this usage of adversarial augmentation is that the adversarially perturbed data still is still realistic enough to be used as input for a downstream model.

Our method is mainly inspired by the adversarial augmentation method by Peng et al. (2018); Zhang et al. (2019) among many different applications of adversarial augmentation techniques. In this paper we use the adversarial augmentation to refer to such specific approaches. In their method, an augmentation model is jointly trained with a target training model. The objective of an adversarial augmentation model is simply to increase the training error of a target training model so that the target model does not overfit to the training data, and generalizes to the test data well. The augmentation model performs image transformation along with the training process, rather than generating new images from scratch. In this setting, existing samples are perturbed using a given set of transformation functions. The hyperparameters that control the functions are optimized to increase the training error of a target model in the hope that it will decrease test errors at the end. Since this approach require a predetermined set of domain-specific functions, one augmentation model cannot be commonly used for vastly different domains. As stated earlier, one important obligation is to keep input data from diverging to a completely different data after augmentation, since the adversarial augmentation model will constantly try to increase training loss. Therefore, a careful selection of the underlying transformation functions is required.

It is important to distinguish between adversarial augmentation and GAN. Although adversarial augmentation was first developed as a defense mechanism against adversarial attack, which is, in turn, inspired by GAN, GAN architecture is essential in adversarial augmentation. In particular, a discriminator, a GAN component, is not needed. Also note that adversarial augmentation is sometimes called, simply, adversarial training. We will stick to the term adversarial augmentation in this paper so that adversarial augmentation and GAN can be clearly distinguished.

2.3. Other Related Work

It is known that combining a discriminator from the GAN architecture can improve the reconstruction performance of an autoencoder. For example, Pathak et al. (2016) reports that an autoencoder tends to output relatively blurry images compared to an original image when trained to minimize only reconstruction loss. This is because the autoencoder prefers to output a safer value without considering consistency between pixels. To prevent this, adversarial loss can be added as another optimization target. With a discriminator, an autoencoder tends to reconstruct high frequency details better.

3. Method

3.1. Autoencoder for Augmentation

Typically, autoencoders are used to extract a more abstract representation of a set of data by reducing the dimension. There are a few points that make autoencoders a suitable ingredient for data augmentation.

The fact that the autoencoder architecture comes with decoders as well as encoders by definition makes them suitable for our method, in contrast to other representation learning techniques. Perturbing input data in its latent space without reconstructing it in the original space cannot strictly be seen as a data augmentation technique. It also makes qualitatively assessing the augmented data hard in practice, since typically it is harder to analyze input data in its latent space.

Also, autoencoders can be used to augment both labeled and unlabeled data. One can adopt a policy of simply attaching the same label to the augmented sample by assuming that it is not drastically different from the original one, so that the class needs to be changed. This is contrasted with other augmentation techniques such as SMOTE, which is only applicable to a labeled dataset.

There are also cases where doing perturbation in the latent vector space of an autoencoder is safer than in the input space. For example, adding independent random noises for each time dimension of a time series data ignores the correlation between different times. However, this problem does not happen when perturbing a latent vector if the autoencoder is designed such that its latent space does not contain the time dimension of the input data.

When selecting an autoencoder architecture, it is important not to choose an architecture with too complex a model with respect to a given set of data. This is because an autoencoder may learn the copying task without actually learning useful representation of the data and generalize poorly when encoding and reconstructing unseen data. Theoretically, an autoencoder with a one-dimensional latent vector can do exact copying of an arbitrary number of samples when given a nonlinear encoder and a decoder with enough capacity Goodfellow et al. (2016). If an autoencoder is only capable of exact copying, the perturbation in the latent space results in completely unrelated data, and therefore cannot be used for our data augmentation method.

3.2. Adversarial Loss to Capture High Frequency Details

As explained in Section 2.3, an autoencoder tends to produce only a rough outline of the predicted object and fails to capture any high frequency detail if it is trained solely to optimize a reconstruction objective Pathak et al. (2016). Depending on the domain, this can be a serious hurdle to augmenting existing data. For example, a picture with a high resolution contains significantly more information than a corresponding lower resolution one, in the sense of both entropy and file size on a physical device. We adopted this architecture in our method.

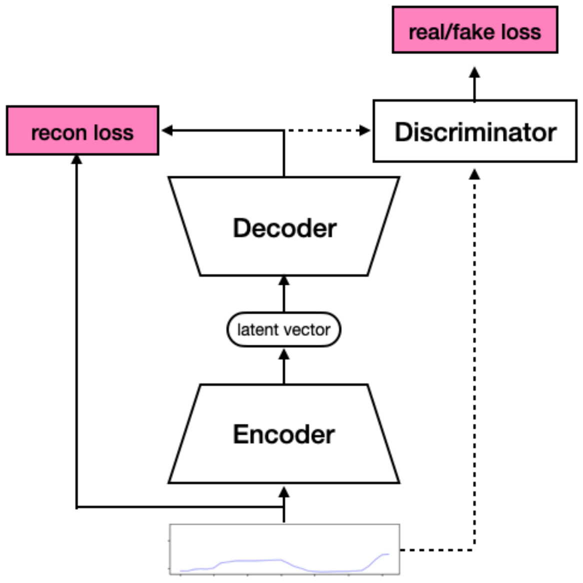

Figure 1: An autoencoder with a discriminator.

Since our method uses an extended architecture with a discriminator, our method can be considered a GAN variant. In this case, the autoencoder corresponds to a generator of a GAN. Therefore, several optimization techniques developed for GAN training can be applied to our method as well.

The two time-scales update rule (TTUR) is a technique for applying different learning rates to the generator (in our case, the autoencoder) and the discriminator of a GAN Heusel et al. (2017). Typically, a smaller learning rate is used for the generator and a larger learning rate is used for the discriminator.

Also, we used a cost-sensitive penalty for the discriminator. We associated a large cost for classifying a real sample as a fake one, compared to classifying a fake sample as a real one. No cost was associated for correct classifications.

Settings	AE recon	AE adv	Disc real	Disc fake
AE	0.0798	0.2661	0.2784	0.2402
Enhanced	**0.0484**	**0.0114**	**0.0150**	0.8055
No TTUR	0.0635	0.0154	0.0268	0.8019
No cost-sensitivty	0.0545	0.2536	0.2807	**0.2546**

Table 1: Autoencoder and discriminator losses

We conducted a detailed experiment to evaluate the effect of applying each technique. Table 1 shows loss changes after applying GAN enhancement techniques. Here, row enhanced refers to an autoencoder enhanced with TTUR and cost-sensitivity settings. The following rows provide ablation study results. For example, 'No TTUR' in the table refers to an autoencoder trained using only using cost-sensitivity. All of the ablation studies ended up worsening the autoencoder reconstruction performance, which suggests that every technique helped improve the autoencoder. The dataset and model settings were the same as the ones for *adjacent* problem in the experiment section.

3.3. Adversarial Augmentation

After an autoencoder with a discriminator finishes training we use the model as a module for adversarial augmentation. In the adversarial augmentation phase an input sample is augmented by perturbing its latent representation. We refer to the module that is in charge with perturbation of latent vectors perturbation module. The perturbation module has learnable parameters, whose objective is to maximize the training loss of the downstream model. The parameters are jointly updated with the parameters of the downstream model. Note the perturbation module is not used at all in autoencoder and discriminator training phase. On the other hand, the autoencoder is not trained in adversarial augmentation phase.

The perturbation module consists of an affine layer, a standardization layer, and rescale layer. The affine layer transforms given a latent vector and an additional randomly generated vector. The output dimension of affine layer should be the same as that of the latent vector, such that the resulting noise vector can be added element-wise.

The output of the affine layer is standardized by the standardization layer so that the resulting value does not grow or shrink indefinitely. In other words, the standardization

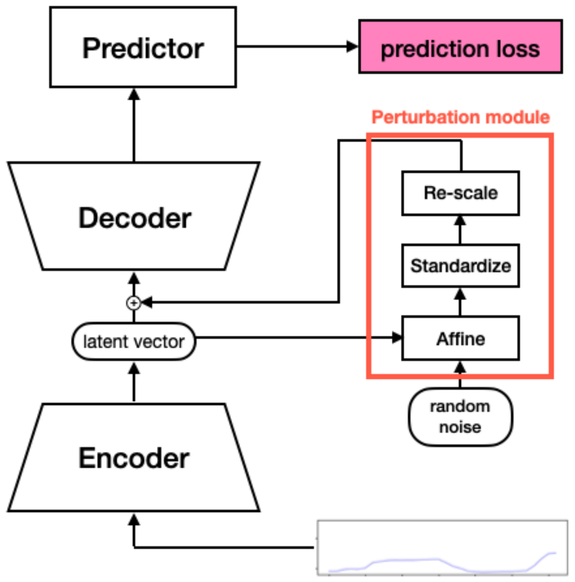

Figure 2: Adversarial augmentation using autoencoder and perturbation module.

layer makes the parameters of affine layer scale-invariant. We used the layer normalization Ba et al. (2016) without including learnable parameters for the standardization in our experiments.

The standardized perturbation vector has a limitation, in that the scale of each element does not reflect the statistics of the latent vector to which it will be added. For example, the absolute average value of the first element in the latent space is ten times as large as that of the second element. In this case, letting each element in the perturbation vector has the same scale causes the perturbations in certain positions of the latent vector to become relatively insignificant. To tackle this issue we re-scale the standardized perturbation vector using the standard deviation of each element of the latent vector. That is to say, the scale of each element in the perturbation vector is proportional to that of the latent vector.

Although using standardization when generating a perturbation vector guarantees the parameter of the affine layer will be scale-invariant, the parameters still can grow excessively. In a typical neural network training, each parameter is naturally bounded to a reasonably small number as the training goes on. This is not always the case for adversarial objective training. In this case, weight normalization Salimans and Kingma (2016) may be needed to prevent floating-point overflow or underflow. Note that weight normalization was not originally developed for solving such problems, but rather for the decoupled optimization of direction and scale parameters.

Our adversarial augmentation method has an uncommon characteristic that is not often observed in other augmentation methods. Since the perturbation module and the downstream model, which is named Predictor in the illustration above, are jointly trained, the augmen- tation function is nonstationary as the training goes on. Clearly, it is hard to guarantee that the most effective augmentation function in an early training phase will also be the most effective one in a later training phase. Dynamically changing the augmentation function opens up new possibilities that have not been actively explored before in the data augmentation literature.

3.4. Architectural Details

The encoder of the autoencoder we used in our experiments is composed of 1-dimensional convolution layers. The convolutional layers have kernel size four, stride two, and padding size one. The encoder consists of four convolutional layers and the output channels for hidden dimensions are 64, 64, 128, and 256. On all layers except for the first one, batch normalization is followed. For an input with 32 timesteps, the corresponding latent vector has two timesteps and 128 channels. The decoder consists of five transposed convolutional layers. The first transposed convolution has kernel size one and its output channel size is 500. The four following transposed convolutional layers have kernel size 4, stride two, and padding size one, which corresponds to the settings for four encoder layers. The channel sizes for the latter four layers are 256, 128, 64, and the same channel size of the input data, which in our experiment was six. Like the encoders, all the decoder layers except for the first layer are followed by batch normalizations. The discriminator for the autoencoder consists of four affine layers. Between each affine transformations dropout with 0.4 probability was applied Srivastava et al. (2014).

In the affine layer of the perturbation module, we composed two smaller affine functions without adding an intermediate non-linear transformation. The output dimension of the first small affine function is smaller than the input dimension. The reason for using two affine functions rather than a single large affine function is to prevent overfitting by reducing the number of parameters. In our experiments, latent vectors have 512 elements and random noises have 100 dimensions. The hidden dimensions between the two small affine transformations is 100. For the re-scaling layer, a running variance of the latent vectors was used.

4. Experiment

4.1. Problem Formulation

We used the KDD CUP 2018 dataset, a public air quality dataset that comes from the KDD CUP Challenge 2018 [1]. We selected six common air quality features, PM2.5, PM10, NO_2, CO, O_3, and SO_2 for our experiments. The observations are hourly and come from 35 different observatories. Each observatory owns the records, which were observed every one hour from January 1, 2017 to January 31, 2018. Observatory information was not included, and observations from all the observatories were treated in the same way. The first 80% of the observations in time were used for training and the latter ones were used for testing.

With this setting we formulated two regression problems. First is what we named *adjacent*. In this formulation, an input is 32 consecutive hourly observations and the corresponding prediction target is the mean PM 2.5 value for the next six hours from the last observation. Similarly, we defined another problem we named *daily*. In this formulation, a prediction target value is also the mean PM 2.5 value for the next six hours from the last observation. An input consists of five chunks and each chunk consists of six consecutive

1. http://www.kdd.org/kdd2018/

Method	MSE-macro	MAE-macro
AE anti-adversarial scale 0.3	0.4601	0.4753
AE anti-adversarial scale 0.1	0.4197	0.4322
AE anti-adversarial scale 0.03	0.3645	0.3999
AE random scale 0.3	0.3548	0.3955
AE random scale 0.1	0.3533	0.3939
AE random scale 0.03	0.3531	0.3937
AE adversarial scale 0.3	0.4435	0.4713
AE adversarial scale 0.1	**0.3339**	**0.3767**
AE adversarial scale 0.03	0.3422	0.3860

Table 2: Comparison of autoencoder-based adversarial augmentations and other schemes

hourly observations. The first observations of the chunks are spaced part by 24 hours, hence the name *daily*.

Although the problems we formulated are not classification problems, we labeled each sample to measure the degree of dataset imbalance and macro-averaged performances. We classified a sample whose target PM2.5 (ug/m^3) value was larger than 150 as a minor class. The ratios of minor classes in *adjacent* and *daily* problems were 5.8% and 5.0% respectively.

When measuring performances, we associated higher weights for errors in the minor class data, to avoid under-representation of the minor class due to its small sample size. In other words, we first separately measured errors such as root-mean-square error for the major and minor classes and then macro-averaged them. This simulates a situation where the number of minor class and major class samples are equal.

4.2. Prediction Model

To measure the effectiveness of different augmentation settings, a downstream prediction model was picked and commonly used. We designed a 2-layered GRU model with 15 hidden dimensions. Although it had a simple architecture, it showed superior performances compared to the 1-dimensional ResNet, which is a commonly used architecture for time series research Wang et al. (2017).

4.3. Results

Note that all the results provided here are averages of ten runs with different random seeds.

First, we confirmed the effect of applying an adversarial objective in a perturbation module. To this end, we set different objectives for the perturbation module. We conducted experiments on *adjacent* problem where we augmented 80% of the minor class data that prediction model received as input.

Table 2 shows the performances of augmentation schemes with different objectives on *adjacent* problem. The top three rows in Table 2 had anti-adversarial objectives. In other words, the goal of the perturbation module was to reduce training error as much as possible. On the other hand, the middle three rows were not trained at all and simply had gaussian random noises added. Note that, in this case, only the re-scale layer of perturbation module was applied. Scales in Table 2 are hyperparameters that were applied after the re-scale

Method	RMSE-macro	RMSE-major	RMSE-minor	MAE-macro	MAE-major	MAE-minor
no oversample	32.3337	19.3603	41.4260	22.2224	11.4704	32.9697
oversample 4	29.0988	21.1376	35.3084	19.7390	12.5932	26.8800
translation no oversample	32.2180	19.4070	41.2206	22.2082	11.5418	32.8745
translation oversample 4	29.4237	20.9871	35.9280	19.9388	12.2506	27.6270
warping no oversample	32.4036	19.2136	41.6005	22.4365	11.4799	33.3931
warping oversample 4	29.2733	20.9871	35.6846	19.7390	12.3363	27.1417
AE adversarial no oversample	31.9889	19.4419	40.8511	21.9941	11.4561	32.5320
AE adversarial oversample 4	**28.1781**	22.0033	33.2245	**18.8160**	12.8881	24.7392

Table 3: Augmentation experiment on *adjacent* problem.

Method	RMSE-macro	RMSE-major	RMSE-minor	MAE-macro	MAE-major	MAE-minor
no oversample	25.8272	16.4430	32.6147	17.7445	10.0685	25.4254
oversample 4	25.3613	17.9637	31.0396	17.4189	10.9516	23.8813
translation no oversample	25.4619	17.1458	31.6606	17.3647	10.3793	24.3500
translation oversample 4	25.2844	18.3059	30.7165	17.3055	11.4548	23.1562
warping no oversample	27.7443	18.7329	34.4756	19.8806	12.1997	27.5615
warping oversample 4	32.7451	26.7302	37.8118	23.7383	18.0158	29.4558
AE adversarial no oversample	25.0572	16.8234	31.1921	16.9256	10.3843	23.4620
AE adversarial oversample 4	**25.0184**	18.3059	30.2776	**16.9207**	11.1489	22.6974

Table 4: Augmentation experiment on *daily* problem

layer. Clearly, the adversarial augmentation outperformed anti-adversarial augmentation or random perturbation.

We compared adversarial augmentation with four different augmentation functions. The first augmentation, which is designated translation in Table 3 and Table 4, adds the same amount of value for each input feature across time.The amount of the addition is randomly decided. Window warping was also compared. For adversarial augmentation we used the best hyperparameter found in the previous. For completeness, we conducted experiments with and without oversampling for the minor class. When oversampling was applied, the minor class sample was sampled four times as frequently as before. Bold numbers indicate the best macro performances. Table 3 and Table 4 report that our adversarial augmentation performed the best in all the settings.

5. Conclusion

Time series data is one of the most prevalent data types in both practice and academia. However, it has a relatively small number of readily available augmentation methods. In this paper we proposed a novel augmentation method using an autoencoder and an adversarial augmentation technique. We demonstrated in our experiments that that our method achieves superior performance compared to other commonly used augmentation functions for times series.

Until recently, adversarial augmentation has only been seen as a remedy for preventing adversarial attacks. Recently Peng et al. (2018); Zhang et al. (2019) demonstrated that it can actually be used to reduce test time error, although the applicability of the adversarial augmentation in different domains remains largely to be explored.

One requirement for successfully applying adversarial augmentation using a neural network model is that the realism of the data should be preserved for however long the model is trained, to generate harder samples to predict. We tackled this critical problem by properly design- ing an autoencoder and perturbation module. The experimental results suggest that our method can be effective for similar imbalanced time-series problems, which are frequently encountered in practice.

We applied our novel augmentation method in an imbalanced time-series domain where no adversarial augmentation had been previously applied. We believe our method can be applied to other domains as well after a decent amount of adaptations, and advance the data augmentation literature in general.

Acknowledgments

This work was supported by the Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No. 2018-0- 00219, Space-time complex artificial intelligence blue-green algae prediction technology based on direct-readable water quality complex sensor and hyperspectral image, 90%, and No.2019-0-00075, Artificial Intelligence Graduate School Program(KAIST), 10%).

References

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16: 321–357, 2002.

Terrance DeVries and Graham W Taylor. Dataset augmentation in feature space. *arXiv preprint arXiv:1702.05538*, 2017.

Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in neural information processing systems*, pages 6626–6637, 2017.

Arthur Le Guennec, Simon Malinowski, and Romain Tavenard. Data Augmentation for Time Series Classification using Convolutional Neural Networks. In *ECML/PKDD Workshop on Advanced Analytics and Learning on Temporal Data*, Riva Del Garda, Italy, September 2016. URL https://halshs.archives-ouvertes.fr/halshs-01357973.

Deepak Pathak, Philipp Krahenbuhl, Jeff Donahue, Trevor Darrell, and Alexei A Efros. Context encoders: Feature learning by inpainting. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2536–2544, 2016.

Xi Peng, Zhiqiang Tang, Fei Yang, Rogerio S. Feris, and Dimitris Metaxas. Jointly optimize data augmentation and network training: Adversarial data augmentation in human pose estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

Tim Salimans and Diederik P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *arXiv preprint arXiv:1602.07868*, 2016.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

Florian Tramer and Dan Boneh. Adversarial training and robustness for multiple perturbations. In *Advances in Neural Information Processing Systems*, pages 5866–5876, 2019.

Riccardo Volpi, Hongseok Namkoong, Ozan Sener, John C Duchi, Vittorio Murino, and Silvio Savarese. Generalizing to unseen domains via adversarial data augmentation. In *Advances in neural information processing systems*, pages 5334–5344, 2018.

Zhiguang Wang, Weizhong Yan, and Tim Oates. Time series classification from scratch with deep neural networks: A strong baseline. In *2017 International joint conference on neural networks (IJCNN)*, pages 1578–1585. IEEE, 2017.

Qingsong Wen, Liang Sun, Xiaomin Song, Jingkun Gao, Xue Wang, and Huan Xu. Time series data augmentation for deep learning: A survey. *arXiv preprint arXiv:2002.12478*, 2020.

I Zeki Yalniz, Hervé Jégou, Kan Chen, Manohar Paluri, and Dhruv Mahajan. Billion-scale semi-supervised learning for image classification. *arXiv preprint arXiv:1905.00546*, 2019.

Xinyu Zhang, Qiang Wang, Jian Zhang, and Zhao Zhong. Adversarial autoaugment. *arXiv preprint arXiv:1912.11188*, 2019.