Performance Analysis of an augmentation algorithms for tabular data

Krunal Patel *Masters Computer Science Lakehead University T*hunder Bay, Ontario, Canada.

krunalp2102@gmail.com

***Abstract*—Machine performance is highly dependant on the amount and quality of data. More the amount of data more the accuracy model will get, it is the simple rule of machine-learning. For these reason, every model needs to be feed with at-least decent amount of data to be more accurate. Data Augmentation plays vital role to achieve more data, from less data available. There are so many augmentation algorithms out there for image classification, but very few algorithms can be applied for tabular augmentation. These paper focuses on few already applied successful algorithm such as Generative Adversarial Network and an Augmentation using noisy data copies. As well as, it also contains few experiments with the algorithms[references] which were developed and used only on an image augmentation tasks until now. At the end of the paper, it compares performances of those algorithms in accordance with their mean square errors. According to the study, Generative Adversarial Network out-performs all of those available algorithms in-terms of a performance, but it is way more expensive than the other algorithms when it comes to the training time. Whereas AUTOELM takes negligible amount of time to get trained. In order to decrease the time complexity a bit, I have conducted a experiment combining AUTOELM with GAN, Which supposedly getting the same accuracy in less time than GAN alone. However, it was helpful in decreasing training time to almost half of the time taken by GAN alone and the accuracy was also decent but not up-to the mark.Which can always be improved by making certain changes in the future, expectantly in the Generator function and to the generated input samples .**

***Index Terms*—Generative Adversarial Network, Data Augmentation,**

I. Introduction

Machine Learning algorithms require large amount of data in-order to train a model. In real world scenario, having large amount of data available for training a model is not so easy in every situations. Many automated car makers [references] and e-commerce sites[references] are the best examples, who seeks more data in-order to accurately predict the condition. That is when data augmentation comes into picture. Data augmentation has been very popular and serves with the ability to train a model even with less amount of training data amongst image classification tasks over last few years. We can actually train a model from a small number of images by applying some transformation techniques[references] on to it. But it is totally different scenario when it comes to tabular data augmentation. It is hard to get the meaning and relevance out of a tabular data, especially for time series forecasting. When you apply a normal data augmentation techniques[references], there are much higher chances of losing the meaning of the data and the data set can no longer be of useful as it should be. Different techniques and approaches for tabular data augmentation have been already published and used by industry experts lately.

Back in 2013-14, Variational auto-encoders (VAE) has been introduced, which are deep latent Gaussian models applied with stochastic variational inference, giving the scalability to the models equal to large data-sets (Welling and Kingma 2013, Wierstra 2014). Since then, many variations of the VAE model have been explored in the natural language domain. Fabius and van Amersfoort were the first to introduce variational recurrent auto-encoders (VRAE). Generative adversarial networks (GAN) are one of the class of latent variable models with implicit latent distribution (Goodfellow et al. 2014). Yu et al. and Fedus,Goodfellow, and Dai made an advancement in applying the GAN model for text generation. In 2018, one popular study showed the work to generate new data by translating sentences into French and back into English [References]. Other work displayed the use of data noise as smoothing [Xie et al., 2017]. When it comes to tabular data, basic flow of adding noise to it is to generate random points and adding those to the points available in the data-set[[**?**]] These techniques are not often used because of their high computation and an implementation cost relative to performance. In 2019, Jason Wei and Kai Zou came up with an universal augmentation technique for NLP called Easy Data Augmentation. They synthetically evaluated EDA on five benchmark classification tasks[[**?**]]. Recently, the tasks of controllable generation and style transfer has been drawing an attention using variational models, which was successfully explored in (Hu et al. 2017 and Shen et al. 2017).

For time-series prediction, there are two techniques that are published and being in practice, those are Generative Adversarial Network [Rocca, Joseph. "Understanding Generative Adversarial Networks (GANs)." Medium, Towards Data Science, 25 Aug. 2019, towardsdatascience.com/understanding-generative-adversarial-networks-gans-cd6e4651a29.] and Gated Recurrent Unit [@inproceedingsinproceedings, author = Le, Thi-Thu-Huong and Kim, Jihyun and Kim, Howon, year = 2016,month = 07,pages =105-110,title = Classification performance using gated recurrent unit recurrent neural network on energy disaggregation, doi = 10.1109/ICMLC.2016.7860885]. Although these techniques has been proven efficient in classifying images, they are really good in classification

of tabular data also. But, those do not inherit dimension reduction and feature extraction, which are proven to be very effective in terms of training accuracy and time complexity. In July 2018, Yimin Yang, Q.M Jonathan Wu and Yaonan Wang developed a new multi layer architecture for an Auto Encoder for dimension reduction and an image reconstruction [Y. Yang, Q. M. J. Wu and Y. Wang, "Autoencoder With Invertible Functions for Dimension Reduction and Image Reconstruction," in IEEE Transactions on Systems, Man, and Cybernetics: Systems, vol. 48, no. 7, pp. 1065-1079, July 2018.]. Provided new architecture was only tested on an image data sets and performed really well. So, i was wondering if i can apply the same architecture on augmenting tabular data as well. These architecture inherits the functionality of feature selection and dimension reduction. Dimension reduction and feature selection plays vital role in reducing features to be fed into the model, which leads to an easy and efficient processing of data. By using that, we can reduce a required time to its extreme without compromising the results.

What these new architecture has to offer : 1) New auto-encoder with multi-layer approach, claiming to be with the fastest training speed in compared to the relevant methods. 2) Weights for the encoding layer are replaced by the previous decoding layer unlike other architectures and are correlated with input data [(section reference)].

In this paper, I have included my study for all the above stated augmentation algorithms for tabular data and their performance comparisons with this new architecture [Y. Yang, Q. M. J. Wu and Y. Wang, "Autoencoder With Invertible Functions for Dimension Reduction and Image Reconstruction," in IEEE Transactions on Systems, Man, and Cybernetics: Systems, vol. 48, no. 7, pp. 1065-1079, July 2018.].

II. Data Augmentation and Regularization

Appropriate regularization is necessary for the high performance on many tasks for models. Widely accepted techniques to prevent model over-fitting model regularizers such as batch normalization and dropout are extremely popular and accepted.

Data augmentation (DA) is a regularization method class that creates artificial training data to gain better models. Most data augmentation techniques can be categorized into generative or transformative methods. Transformative DA is used in computer vision problems, for example images are altered randomly with linear transformations (shifting, rotating etc.) to enhance the performance of computer vision models (Steinkraus, Platt and Simard 2003). Generative DA alters the generative capability of latent variable models to create convincing data samples. With the advancement in generative models like GANs, the potential to use them for data augmentation has gained much attention nowadays. However, available resources and lack of published researches on tabular data augmentation was the major hurdle, as per my knowledge my work will be first to explore different DAs on a tabular data-set, comparing their efficiency.

III. Notation

The sets of real numbers are denoted by **R**. For N samples

$$(x_i)_{i=1}^{N} \tag{1}$$

x represents the original input data. For generated fake samples

$$(x'_j)M_{j=1} \tag{2}$$

x' represents newly generated input data, M is the number of total fake data to be generated.

IV. Experiments

Augmentation on a classification problems are more explored than the time series prediction. My goal was to see how those augmentation algorithms perform on a time-series problems. I have considered different approaches to perform performance comparison for Data Augmentation on tabular data Which are listed below:
1) Noisy copies of the data-set
2) Using beta values from AUOTELM
3) Generative Adversarial Network (GAN)
4) Combined structure of an AUTOELM with GAN

A. Noisy copies

Simplest way to augment the time series data is to add noisy random but relevant values, to the original data.

Algorithm 1 : Noisy copy generation and data manipulation

Input : Target data points P (y`|x`)
Data manipulation function R (x,y | D)
Training set D

1: Initialize model parameter and manipulation parameter .
2: making 50 duplicate copies of the training set D.
3: repeat:
4: generate random maximum and minimun threshold noise considering each data point according to equation (1).
5: until end of the data points

Output : Augmented random data points having the same range than of the input.
Augmented data points P (y`|x`) and manipulation R (y,x|D)

Fig. 1. Algorithm for noisy copy augmentation technique

After creating 50 copies of the input data, each tabular cell gets manipulated by the random threshold produced from the equation 3.

$$X + (15 * z - 15) + (2 * z(X) - 1) \tag{3}$$

where, (15 * z - 15) is a global offset with a large scalar and (2 * z(X) - 1) is small element wise offset.

B. Using beta values from AUOTELM - base algorithm

According to paper[references], Initially, the data transformation is carried out to transfer the inputs to a random feature space followed by encoding the layer updates through a replacement learning method according to Fig 2. After the successful initialization of encoding layer, input data is them mapped to feature dimension space(low-dimensional). According to Fig. 2, output data of a network f can be represented as

$$f = G(a_n, b_n, G(a_f, b_f, x)) \tag{4}$$

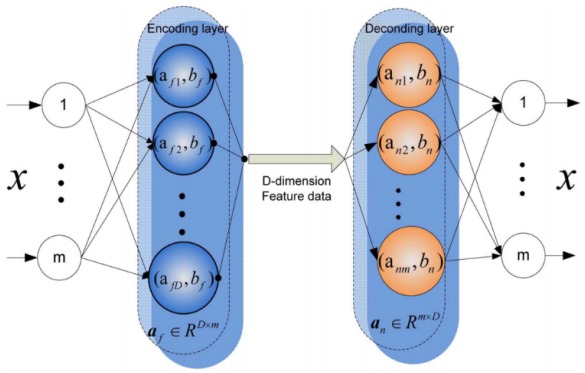


Fig. 2. Structure of two layer AUTOELM [references]

Algorithm 2 : Using Beta values from an AUTOELM

Initialization : Given a training set $\{(x_i,y_i)\}_{i=1} \subset R^m \times R^m$, an invertible activation function G(•). maximum loop number L and j = 1

Initial Training Phase :

1 : orthogonal random: generate the intial nodes a_f and b_f according to equation (4).
2: **Repeat**
3: Obtain hidden nodes a_n and b_n in decoding layer based on equation (5-6)
4: j = j + 1 and obtain hidden nodes a_f b_f in encoding layer based on equation (7).
5: **until j < L**

Sequential Training Phase :

Input : a_n, b_n and extracted features
1: finding inverse activation function and beta according to equation (8-9).

Output: Beta values, which can be used to alter input data

Fig. 3. Algorithm for AUTOELM [references]

- Beta weight calculation

The learning process is divided into two consecutive steps i) Initial training phase ii) Sequential training phase.
Step 1: The initial parameters for the encoding layer are being calculated by orthogonal random process

$$H_f = G(a_f, b_f, x), (a_f)^T * a_f = I, (b_f)^T * b_f = 1 \tag{5}$$

where af is the orthogonal random weights and biases of the encoding layer. Hf is current feature data. G denotes any general hidden neuron G, in our case it is sigmoid function, used in the decoding layer. ii) Given an invertible function G, obtain the parameters in the encoding layer article

$$a_n = \{^{arcsin(y)*(H_f)^\dagger}_{-log(\frac{1}{y}-1)*(H_f)^\dagger} \tag{6}$$

$$b_n = \sqrt{mse(a_n * H_f - G^{-1}(y))} \tag{7}$$

$$G^{-1}(.) = \{^{arcsin(.) if G(.)=sin(.)}_{-log(\frac{1}{(.)}) if G(.)=1/(1+e^{(.)})} \tag{8}$$

where mse means the mean squared errors and $H^\dagger$is the Moore-Penrose generalized inverse of matrix H, which can be found as, $H^\dagger = (H^T)$, if $H^T H$ is non-singular, or $H^\dagger = H^T(H^T H)^{-1}$ if HH^T is singular. iii) Update a_f and b_f by

$$a_f = (a_n)^T, b_f = b_n, b_f \epsilon R \tag{9}$$

and update the feature data $H_f = G(a_f, b_f, x)$ iV) repeat steps ii and iii multiple times. These parameters are then passed to the sequential phase of the network where beta weight is being calculated according to the formula

$$b_f = \sqrt{mse(H * A_n - arcsin(y))}$$

where $A_n is\ A_n + ((K^{-1} * H^T) * (arcsin(y) - (H * a_n))$ and a_n is an initial weight (10)

After three consecutive steps we can use b_f from equation 10. Which is clearly visible in the Fig 4 Which can be our potential Beta weight for the rest of the alteration required in the augmentation. After three consecutive training of both phases, obtained beta weights can be used in augmenting training data-set as per the following formula.

$$X + (5 * B_w - 5) + (2 * z(X) - 1) \tag{11}$$

Where, (-5) and (5) helps keeping the data points to be in the same range as of the input data points, that is [-5,5].

C. Generative Adversarial Network

Generative Adversarial Networks are made up of two neural networks i) Generator and ii) Discriminator [Goodfellow et al., 2014], which got a recognition as a two-player zero-sum mini-max game later on.

- The Discriminator: is a simple classifier, which tries to distinguish real data from the given set of fake data by the generator. Discriminator requires two inputs at a time, Real and fake data. It treats real examples as positive examples and the other as negative during training. Discriminator training can be summarized in following steps :
  - The Discriminator classifies both real fake input data from the generator

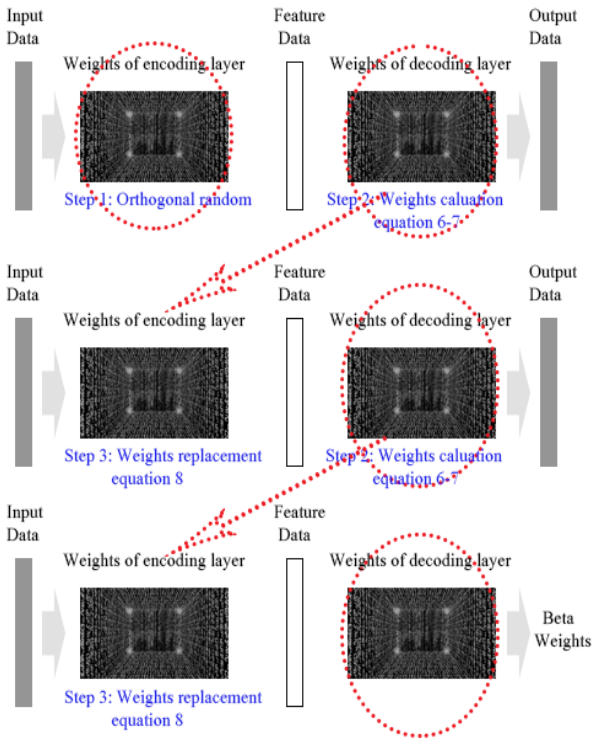


Fig. 4. Beta weight calculation for AUTOELM [references]

- The discriminator loss penalized the discriminator for miss-classifying a real instance as fake or vice versa.
- The discriminator updates its weights through back propagation from the discriminator loss through th network.

In Figure , two boxes represent two data sources feed ing into the discriminator. During discriminator trainin generator stays idle and the weights for it remains static while it produces examples for the discriminator.

- The generator: is a part of GAN learns to create mor and more accurate fake data by using feedback(gradients from the discriminator. Generator training require tigh integration between the generator and the discriminato Components needed to train the Generator :
  - random input
  - generator network (transforms random inputs to data instance)
  - discriminator network (classifies the generated data,
  - discriminator input
  - generator loss, which penalizes the generator for failing to fool the discriminator.

In its most basic form, GAN takes random noise as its input, and then transforms this noise into a meaningful output. Generator training can be summed up with the following steps:

- Sample random noise.
- Produce generator output from given random noise sample.
- Find the discriminator classification(Real or Fake).
- loss calculation from the discriminator classification.
- Obtain gradients from both discriminator and generator through back-propagation.
- Use gradients to change weights(Generator only).

$$min_G max_D V(G,D) = E_{x\ p_{data}(x)}[logD(x)] + E_{z\ p_z[log(1-D(G(z))]} \quad (12)$$

Where V(G,D) represents the discriminator. p_data is the probability distribution of the real data-set, z is the input noise of the generator. G(z) is the generator function and G(x) is a discriminator function which specifies the probability that a discriminator input x is coming from the real data-set.

Most commonly, GANs have been modified to synthesize EHR(electronic health records) in [Baowaly et al.,2018 and Choi et al.,2017]. But, they were only able to synthesize discrete values in patient records and cannot be used directly for the tables containing other types. Recently, Table GAN has been introduced to generate new fake table based on the base table provided [Park et al.,2018], But, could only be used effectively for the classification problems. DCGAN [Radford et al.,2015] created a benchmark image generation framework which can be used for tables after minor customization [Park et al. 2018].

I have tried creating 1D GAN structure for the study of time-series analysis, which is shown in below figure.

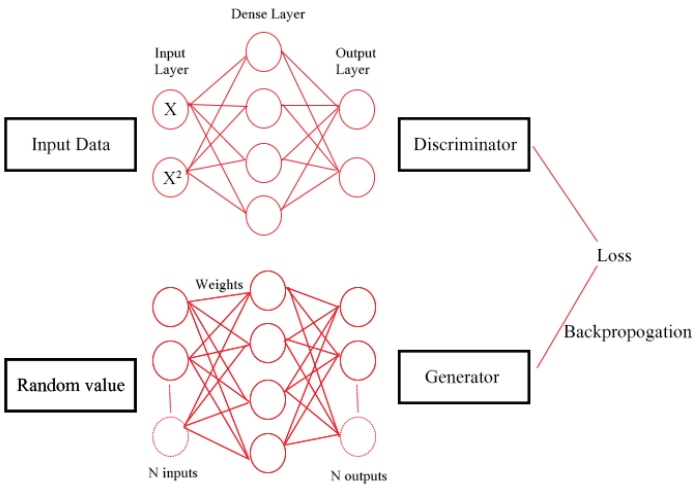


Fig. 5. GAN Structure

Inputs for the discriminator training can be described as (13)

$$Input = (X, X^2) \quad (13)$$

Where X is input data from the data-set and X^2 is the squared input of the same data-set. Because Generator requires two inputs.

Whereas, Generator generates random floating points based on the gradients passed on by the loss function. Generator can be denoted by (14)

$$Output = RAND(-5, 5) \tag{14}$$

Where -5 and 5 represents the boundaries for the points to be generated.

And the loss function is represented by (**??**)

$$E_x[log(D(x))] + E_z[log(1 - D(G(z)))] \tag{15}$$

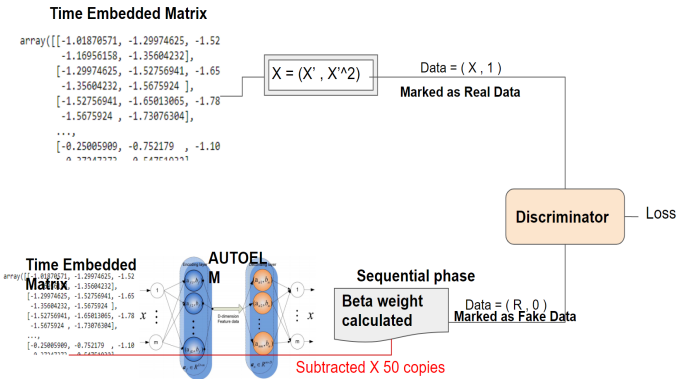
Where, D(x) is the discriminator's probability estimation, that the x is real data, E_x is the expected value, G(z) is the generator's output on the given noise z, D(G(z)) is the discriminator's estimation probability of that the fake data is real and E_z is the expected value over all random inputs to the generator

D. AUTOELM with GAN

Results generated by GAN is outstanding, but the time complexity of GAN is also way more expensive. It can take up-to 36 hours on extremely small tabular data-set, which will be even drastic for data-sets containing images.

So, the main concept behind this experiment was to reduce the time consumption for the training of a tabular data. In the experiment as of the above description on GAN, Generator is supposed to generate a random data points between given interval and then passed to the neural network created with an input, Dense and an output layer. Based on the gradients passed by the loss function, neural network tries to adjust those points that are more relevant to the input data points to get the best possible outcome. Instead of generating totally new data points randomly, I took the modified data-points using the beta values from an AUTOELM (explained in section 2). instead of generating the points based on the gradients, algorithm is fetching random relevant values from the generated points, which helps reducing the overall time consumption.

Structure for the experiment looks like below.



The less the distance between two provided points the less time it takes to fool the discriminator, as it only require a few steps of weight modification. By using (Section using Beta Weight from AUTOELM) method we can have data points gathered under even tighter plane space. So that the points generated takes less time to get to the points of the real data line. Below figure shows the shape of the real data and the fake data.

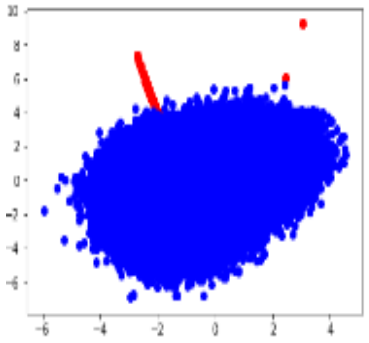


Fig. 7. Shape of a real and fake data

Where, red dotted line shows the actual U shape for the input data in 1D environment setup and blue scattered points represents the uneven shape of the generated fake data. After some sort of training, those blue dotted lines should be nearly equal to in the shape of the original data point shape, which is U. For GAN, those fake points are generated randomly and have more distance from the original data points, which leads to more training time. In this case, new experiments(using beta values from AUTOELM from section 2) leads to less distance between those real and fake data points that is on an average 0.4345 which was 0.8912 in the case of normal GAN. Distance is measured by the following formula:

$$D = \sqrt{X^2 + Y^2} \tag{16}$$

Where, D is distance, X is real data point and Y represents fake data point.

Loss functions play a vital role in adjusting those weights for the Generator, to generate new data points that a Discriminator cannot differentiate between a fake and original data-points. There are two loss functions i have tried on the experimental model, which are as follows:

$$\frac{1}{m}\sum_{i=1}^{m}[logD(x^i) + log(1 - D(G(z^i)))] \tag{17}$$

$$\frac{1}{m}\sum_{i=1}^{m}[log(1 - D(G(z^i))) \tag{18}$$

Where D is the discriminator weights and G is the Generator weights.

Equation (17) and (18) are to change the weights of a discriminator and Generator by implementing those in the below equations of weight calculation.

$$\theta = (1 - \alpha\lambda)\theta - \alpha\frac{1}{b}\sum_{k=i}^{i+b-1}\frac{\sigma E}{\sigma\theta}(x^k, y^k, \theta) \tag{19}$$

Above equation is a weight updation formula. Where E= the error measure

$\theta = weights$

$\alpha = learningrate$

$1 - \alpha\lambda = weightdecay$

$b = batchsize$

$x = variables$

Where new Weights can be used as a beta weight in the sequential phase of an AUTOELM algorithm.

E. Performances

a) Performance graphs for using Beta values of AUTOELM: :

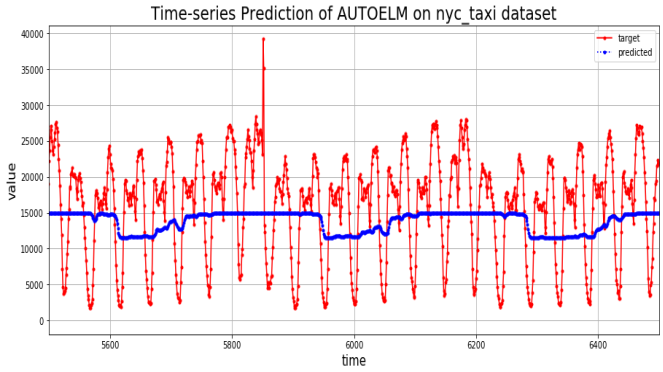


Fig. 8. First iteration

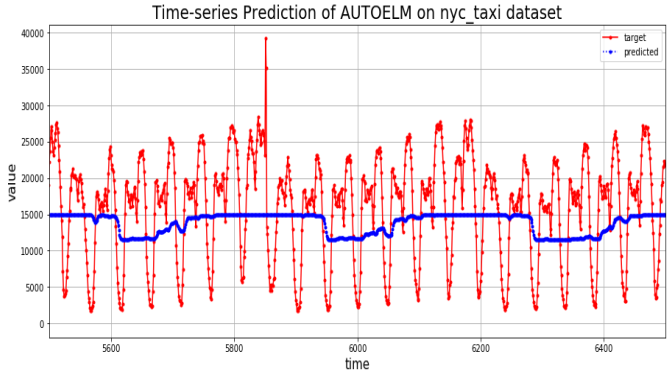


Fig. 9. Second iteration

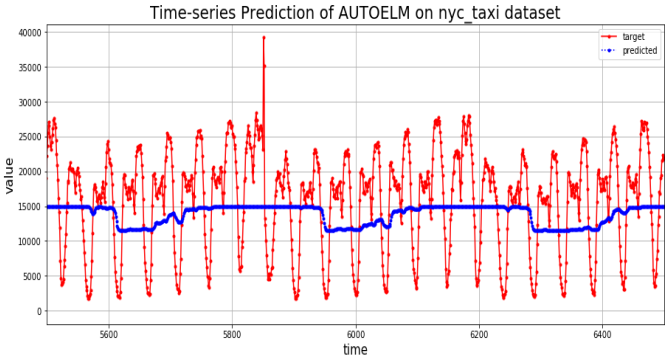


Fig. 10. Third iteration

b) Error prediction for using Beta values from AUTOELM: :

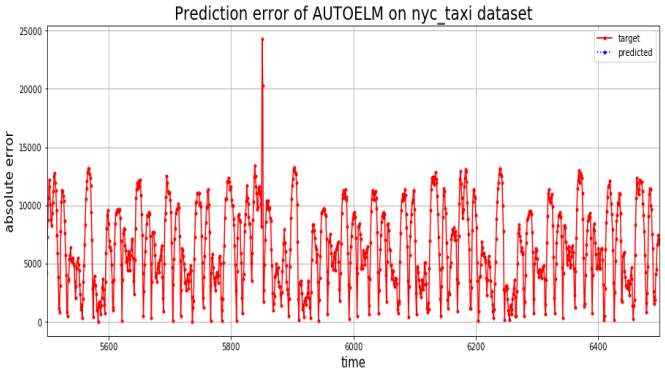


Fig. 11. First iteration

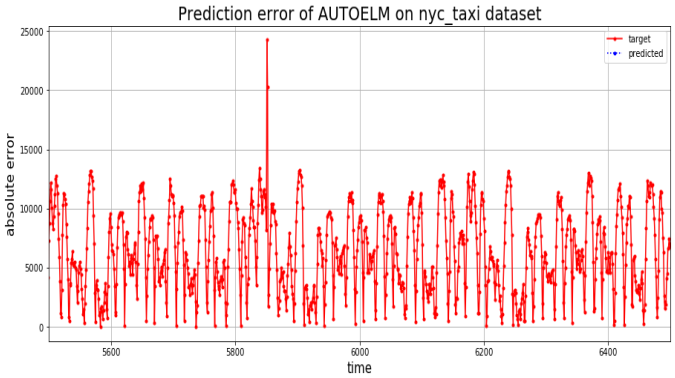


Fig. 12. Second iteration

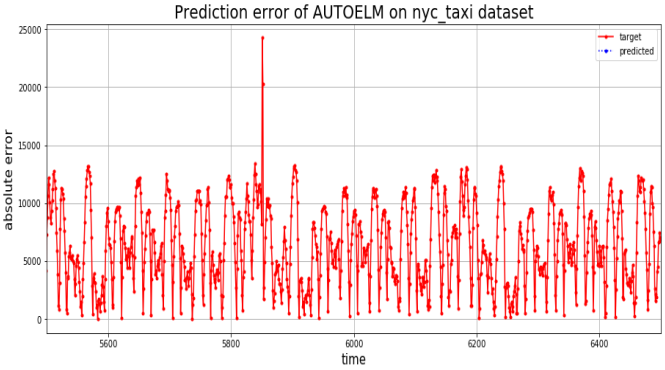


Fig. 13. Third iteration

c) Performance graph for using noisy values: :

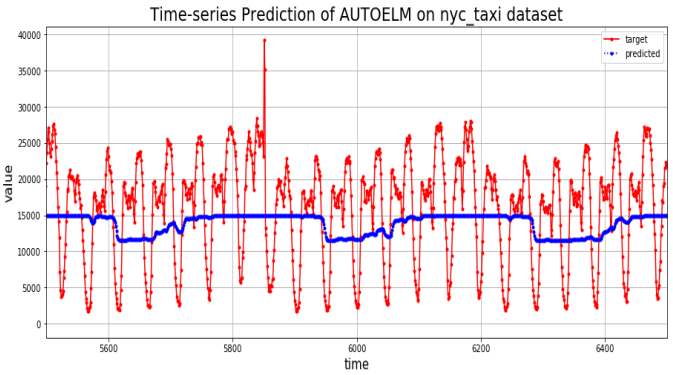


Fig. 14. First iteration

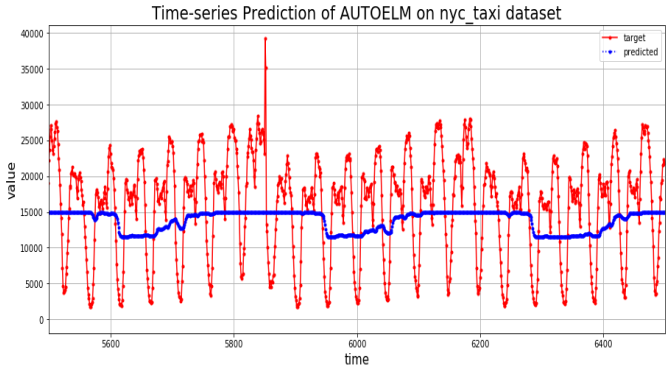


Fig. 15. Second iteration

d) Error prediction for using noisy values: :

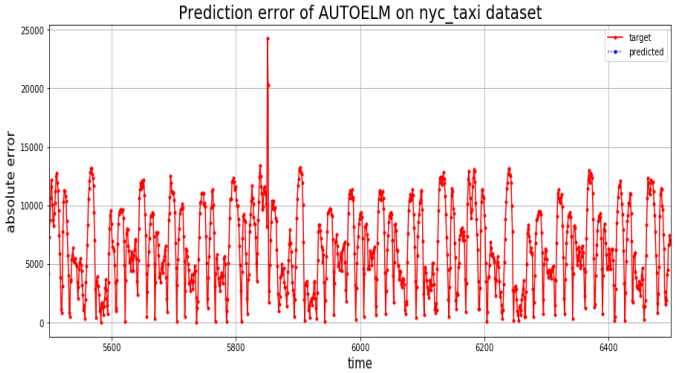


Fig. 16. First iteration

e) Performance measure: : Performance measured for different methods have been analysed based on the Mean Squared Error found with the data-set trained with different algorithms.

Below performance measures are taken on the RTX 2080 GPU with 6 GB graphics and 16 GB DDR6 RAM with python 3+ in Jupyter Notebook environment. Which can vary depending on the system configurations.

TABLE I
PERFORMANCE ANALYSIS

Augmentationn Methods	**Mean Squared Error *AUTOELM***
Noisy copy	5.1282
Beta values of AUTOELM	4.7023
GAN	2.657
GAN + AUTOELM	3.564

TABLE II
TIME CONSUMPTION FOR TRAINING

Augmentationn Methods	**Time consumption *AUTOELM***
Noisy copy	7 minutes
Beta values of AUTOELM	12 minutes
GAN	36 hours
GAN + AUTOELM	19.3 hours

REFERENCES

[1] @inproceedingsinproceedings, author = Arslan, Mehmet and Güzel, Metehan and Demirci, Mehmet and Ozdemir, Suat, year = 2019, month = 09, pages = , title = SMOTE and Gaussian Noise Based Sensor Data Augmentation, doi = 10.1109/UBMK.2019.8907003

[2] https://github.com/jasonwei20/eda$_nlp$ $Rocca, Joseph. \backslash Understanding Generative Adversarial Networks (GANs)."$ $generative-adversarial-networks-gans-cd6e4651a29.$

[3] @inproceedingsinproceedings, author = Le, Thi-Thu-Huong and Kim, Jihyun and Kim, Howon, year = 2016,month = 07,pages =105-110,title = Classification performance using gated recurrent unit recurrent neural network on energy disaggregation, doi = 10.1109/ICMLC.2016.7860885

[4] Fadaee, M.; Bisazza, A.; and Monz, C. 2017. Data augmentation for low-resource neural machine translation. In ACL, volume 2, 567–573

[5] Mesnil, G.; Dauphin, Y.; Yao, K.; Bengio, Y.; Deng, L.; Hakkani-Tur, D.; He, X.; Heck, L.; Tur, G.; Yu, D.; and Zweig, G. 2015. Using recurrent neural networks for slot filling in spoken language understanding. IEEE/ACM TASLP 23:530–539.

[6] Hemphill, C. T.; Godfrey, J. J.; and Doddington, G. R. 1990. The atis spoken language systems pilot corpus. In The Workshop on Speech and Natural Language, 96–101.

[7] Kafle, K.; Yousefhussien, M.; and Kanan, C. 2017. Data augmentation for visual question answering. In INLG, 198– 202.

[8] Yao, K.; Peng, B.; Zhang, Y.; Yu, D.; Zweig, G.; and Shi, Y. 2014. Spoken language understanding using long short-term memory neural networks. In SLT Workshop, 189–194.

[9] Zoph, B.; Yuret, D.; May, J.; and Knight, K. 2016. Transfer learning for low-resource neural machine translation. In EMNLP, 1568–1575.

[10] Simard, P.; Steinkraus, D.; and Platt, J. 2003. Best practices for convolutional neural networks applied to visual document analysis. In ICDAR, 958.

[11] Liu, B., and Lane, I. 2016. Attention-based recurrent neural network models for joint intent detection and slot filling. In INTERSPEECH, 685–689.

[12] H.-S. Chang, E. Learned-Miller, and A. McCallum. Active bias: Training more accurate neural networks by emphasizing high variance samples. In NeurIPS, pages 1002–1012, 2017.

[13] E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, and Q. V. Le. Autoaugment: Learning augmentation policies from data. In CVPR, 2019.

[14] Y. Fan, F. Tian, T. Qin, X.-Y. Li, and T.-Y. Liu. Learning to teach. In ICLR, 2018.

[15] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. Journal of computer and system sciences, 55(1):119–139, 1997.
[16] G. E. Hinton, P. Dayan, B. J. Frey, and R. M. Neal. The" wake-sleep" algorithm for unsupervised neural networks. Science, 268(5214):1158, 1995.
[17] Z. Hu, Z. Yang, X. Liang, R. Salakhutdinov, and E. P. Xing. Toward controlled generation of text. In ICML, 2017.
[18] L. Jiang, Z. Zhou, T. Leung, L.-J. Li, and L. Fei-Fei. Mentornet: Learning data-driven curriculum for very deep neural networks on corrupted labels. In ICML, 2018.
[19] T. Ko, V. Peddinti, D. Povey, and S. Khudanpur. Audio augmentation for speech recognition. In INTERSPEECH, 2015.
[20] D. S. Park, W. Chan, Y. Zhang, C.-C. Chiu, B. Zoph, E. D. Cubuk, and Q. V. Le. Specaugment: A simple data augmentation method for automatic speech recognition. arXiv preprint arXiv 1904.08779.
[21] M. Ren, W. Zeng, B. Yang, and R. Urtasun. Learning to reweight examples for robust deep learning. In ICML, 2018.
[22] P. Y. Simard, Y. A. LeCun, J. S. Denker, and B. Victorri. Transformation invariance in pattern recognition—tangent distance and tangent propagation. In Neural networks: tricks of the trade, pages 239–274. Springer, 1998.
[23] X. Wu, S. Lv, L. Zang, J. Han, and S. Hu. Conditional BERT contextual augmentation. arXiv preprint arXiv:1812.06705, 2018.