4-1-2021

Variable Autoencoders for Biosensor Data Augmentation

Solomon Kim
Andrews University, ksolomon@andrews.edu

J. N. Andrews Honors Program
Andrews University

HONS 497
Honors Thesis

Variable Autoencoders for Biosensor Data Augmentation

Solomon Kim

04/01/2021

Advisor: Rodney Summerscales

Primary Advisor Signature: 

Department: Computing

Variable Autoencoders for Biosensor Data Augmentation

Solomon Kim

Abstract — **Over the past decade machine learning and artificial intelligence's resurgence spawned the desire to mimic human creative ability. Initially attempts to create images, music, and text flooded the community, though little has been learned regarding constrained, one-dimensional data generation. This paper demonstrates a variational autoencoder approach to this problem. By modeling biosensor current and concentration data we aim to augment the existing dataset. In training a multi-layer neural network based encoder and decoder we were able to generate realistic, original samples. These results demonstrate the ability to realistically augment datasets, improving training of machine learning models designed to predict concentration from input signals.**

I. INTRODUCTION

Machine learning provides a gateway to applicable artificial intelligence with one main drawback, the need for big data. Given that most data is difficult to collect, many projects require some form of data augmentation. While many examples of 2-dimensional data augmentation exist, using generative adversarial neural networks, multi-input, variable dependent 1-dimensional data augmentation is still in its infancy.

More specifically, the biofuel industry encounters a predictive analysis issue. Phenolic compounds inhibit the fermentation process in many crops, such as corn, when producing ethanol. Unfortunately, there is no effective way to efficiently monitor the concentration of phenolic compounds. Currently, the industry uses electrochemiluminescence (ECL) sensors, however, these sensors are bulky, expensive, and difficult to deploy in mass.

In order to combat this issue this project aims to create a smartphone ECL sensor that processes images or current data to predict the concentration of phenolic compounds. By using machine learning algorithms as a predictive model, the smartphone will ensure a more efficient and affordable solution. However, real data collection takes time so the training set for the predictive model is sparse, leading to an inflexible model. In this paper I will describe a variational autoencoder (VAE) approach for augmenting the training dataset, merging created samples and real samples, improving the predictive model, and providing a basis for other variable dependent methods of data augmentation.

II. RELATED WORK

A. Deep Learning for Data Creation

Many different methods for data creation have been explored, but the most popular and effective has been deep learning [1]. Deep learning attempts to generate data using unsupervised learning. When learning the model observes the data distribution in the given training set and then predicts what another sample in that distribution would be. The two most common types of generative models are VAEs [2] and a generative adversarial network (GAN) [3]. A variational autoencoder will generally use an encoder and decoder network separately. It uses these two networks in order to determine the probability of generated data being in the original dataset. At the end of training the decoder model will be able to generate realistic data. In a GAN there is a generator and discriminator. The generator will create samples and the discriminator will determine if those samples are real or fake by comparing them with real samples. The discriminator will provide feedback to the generator until the generator learns to fool the discriminator. At the end of training the generator is left to create realistic samples. Both of these methods have gained popularity over the years, though we will be using VAEs to solve our specific problem.

B. Variational Autoencoders for Data Augmentation

VAEs have been used for both 1-dimensional and 2-dimensional data augmentation for many years now. Recently there has been a surge in the need for this technology given the increase in machine learning. More specifically VAEs with conditional, or variabalized inputs have become increasingly popular. Lee et. al outlined a base conditional VAE model for attribute conditioned image generation [4]. This allowed them to generate faces from a latent space, using different descriptive variables, such as "smile" or "girl", to dictate the created image. While their base model was a VAE, they also investigated other types of models. While their work was an example of 2-dimensional data augmentation with descriptive variables there was only one paper that dealt with 1-dimensional data creation. Luo et. al investigated data augmentation for EEG-based emotion recognition, using a VAE model [5]. This paper compared the results of a VAE and multiple GAN models when generating

data. This paper uses a consistent evidence lower bound (ELBO) loss function, which was utilized in our research as well.

III. METHODOLOGY

A. Data Cleaning

The entire modeling and data cleaning approaches were done using Python 3.6 in a Google Collaboratory notebook. The data that we were modeling was ECL sensor data in the form of current occurring over time. Each sample run represented a single curve of 1000 points that we were attempting to replicate. We trimmed this down to the first 200 points, which were the most unique. For every curve there was a corresponding concentration that was taken into account. There were also two sets of data relating to the two different acids measured, vanillic and p-Coumaric. The data was split into these two acidic groups and sorted by concentration. We then normalized the data based on a control testing on the corresponding electrode. Simply dividing each sample by the maximum value of the control group normalized the data and the data was cleaned.

B. VAE Model Architecture

For our model we started from a generic VAE structure with an encoder feeding into the decoder. In Figure 1 we see the general visualization of the model architecture.

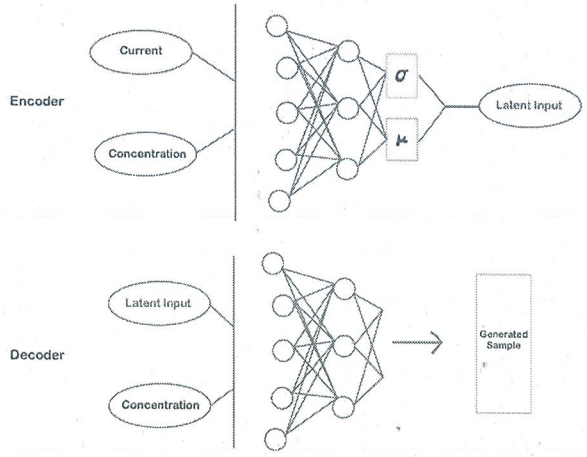

Figure 1: Illustration of The General Model Architecture

The encoder takes in the array of current samples, 200 points on the curve, as well as a single value for the corresponding concentration. The encoder consists of two branches, the current and concentration branch. A more detailed visualization of the encoder is shown in Figure 2.

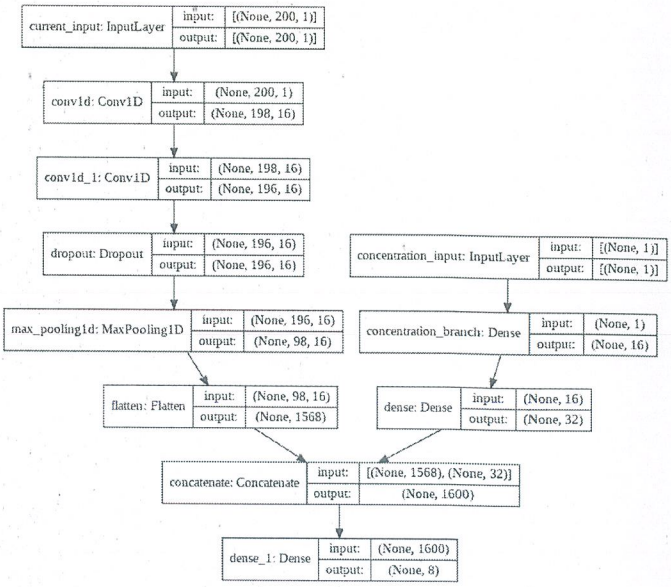

Figure 2: Detailed Encoder Architecture

The current branch begins with two Conv1D layers with 16 filters, relu activation, and a kernel size of 3. There is then a dropout layer, followed by a MaxPooling1D layer, and finally a flattening layer. The concentration branch on the other hand had only two relu activated dense layers. These branches were concatenated and combined with an output corresponding to the latent dimension we had chosen. For this project we decided to have a latent dimension of 4. This latent dimension would later be split into a mean and variance using tf.split. The decoder consists of two branches as well, the latent and concentration branch. A more detailed visualization is shown in Figure 3.

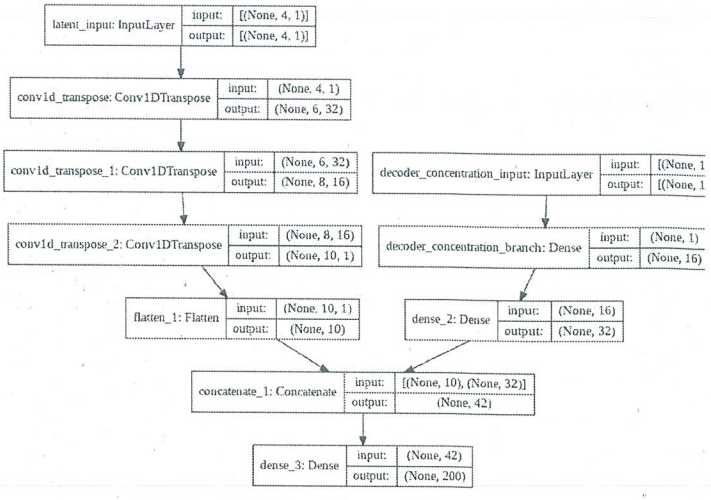

Figure 3 - Detailed Architecture of Decoder

The latent branch has three Conv1DTranspose transpose layers that lead to a flattening layer. The concentration branch has two dense layers. Both of these branches are concatenated and then fed into a dense output layer with dimensionality of 200. This output dimension matches the number of samples in the original curve.

C. Training Strategy

We had to create two separate, identical models; one for vanillic acid and one for p-Coumaric acid. Each of these models were trained in the same way. The loss function that we decided to use was a variation of evidence lower bound (ELBO). Our goal is to maximize the ELBO on the marginal log-likelihood. Considering that our VAE will encode our inputs x into a latent representation z, our goal is to find the probability of values in z being present in x. Originally our ELBO function is represented by Equation 1.

$$\log p(x) \geq \text{ELBO} = \mathbb{E}_{q(z|x)}\left[\log \frac{p(x,z)}{q(z|x)}\right]$$

Equation 1 - ELBO

However, given the complexity of this equation we decompose it into Equation 2.

$$\log p(x|z) + \log p(z) - \log q(z|x)$$

Equation 2 - Monte Carlo Estimate

This decomposition is more formally known as the Monte Carlo estimate, a much quicker and easier value to calculate. We could also use the Kullback-Leibler divergence, which has been well documented, however, for simplicity's sake we decided to stay with the Monte Carlo estimate. In order to calculate the three parts of Equation 2 we used two different strategies. To calculate the second and third terms we use a log normal probability density function. One note is that we use a reparameterization trick in order to backpropagate gradients in the encoder [6]. If we were to just create a sample z from the decoder using the latent distribution defined by the mean and variance given by the encoder we would create a bottleneck given we would be learning on a random node. To avoid this problem we must reparameterize the sample z using the encoder results as well as a defined constant. Since we have a defined function to generate z we can now backpropagate through these nodes. We sample our reparameterized z from the distribution to obtain the second term and sample the mean and variance as well as the reparameterized z in order to find the third term. The first term, however, takes a bit more work. Figure 4 visualizes sigmoid cross entropy.

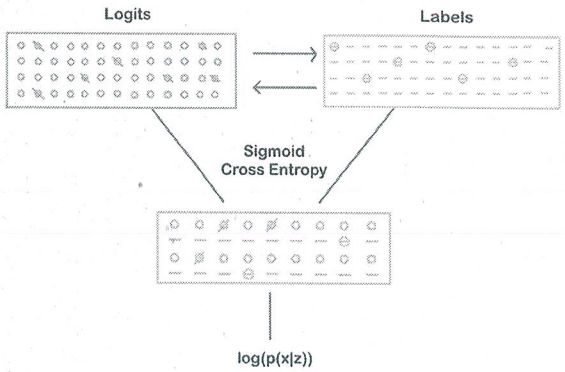

Figure 4 - Illustration of Sigmoid Cross Entropy

The logits are represented by our generated sample and the labels are our original sample. Sigmoid cross entropy will measure the probability error between the two samples. Essentially it will compare the two and check the probability that a value from the logits, or generated sample will be in a real sample. This allows us to have all three parts of Equation 2, giving us a measurable loss.

In order to train we had to iterate over each concentration, making sure that we are changing our descriptive variable. To do this we ran an exterior for loop that iterates over each concentration present in the dataset. If there are samples for that concentration we choose the first curve of 200 points and run it through our VAE with the concentration. We return our loss and backpropagate using an Adam optimizer with a learning rate of $1e^{-4}$. We had to use tensorflow's gradient tape in order to accomplish this given we have multiple branches in our functional model. After back propagation we move on to the next sample for our concentration and do the same. Once we have gone through each sample for a specified concentration we then iterate through for a specified number of epochs. For the purposes of our project we had 25 epochs for this inner loop. Then we move to the next concentration and repeat. Since the model at the time the first concentrations are seen doesn't have much exposure to the data we then run this outer for loop 4 times, an outer epoch. In all we train the model on each concentration and sample within that concentration 100 times.

D. Testing Strategy

In order to test the results of our generation technique we had to look towards our predictive model. We trained the predictive model with differing amounts of real and

generated data. The training and test data for the predictive model were split using a stratified shuffle split. We ran tests with no generated data, 10% generated data, and then increasing in 10% increments until it was training on an entirely generated dataset. We then measured the MAE and RMSE of the predictive model at each of these increments in order to determine the success of our VAE generative model. We repeated this process of splitting our training data, training the model for 10 epochs, recording the average MAE and RMSE, and then increasing the amount of generated data added by 10% until we had used 100% of the generated data. This was done 10 separate times, the learning curves for MAE and RMSE were then averaged and standard deviation was found for each percentage of generated data added. Another factor that was considered was the visual similarity between the generated and real curves. Note that there were two predictive models tested, one for each of the acids, with the corresponding generative models producing data.

IV. RESULTS

The two different predictive models for the two acids yielded different results when using the generated data. Figure 5 shows the average MAE learning curve for p-Coumaric acid.

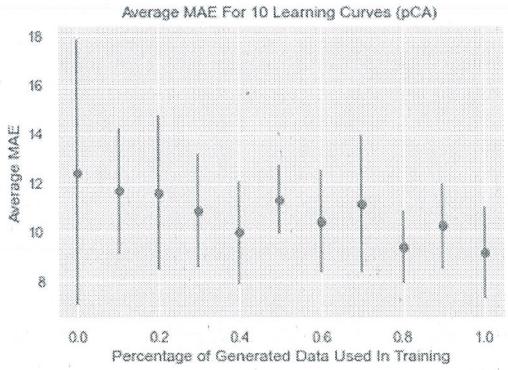

Figure 5 - MAE Learning Curve Amalgamation (pCA)

While there is a large amount of standard deviation in the first initial error, as we add generated samples not only does the standard deviation decrease, but the average MAE decreases as well. This indicates that adding more generated data improves the model's accuracy using this metric. For these tests we used 250 samples as the 100% mark, anything above that proved to plateau. Similar to the MAE results, the RMSE results showed consistent improvement. Figure 6 shows the average RMSE for 10 learning curves for p-Coumaric acid. The RMSE was a little bit higher than the MAE for this acid, which is understandable. Given that RMSE is a quadratic function the value is expected to be larger than MAE, and more sensitive to large deviations from the norm.

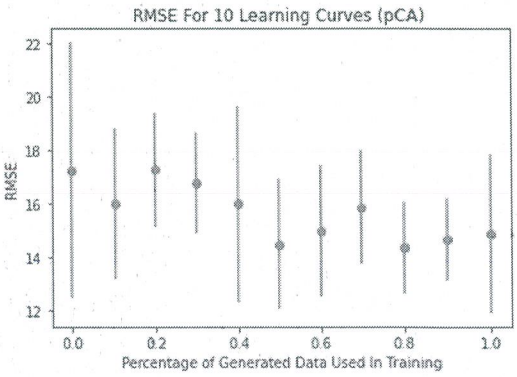

Figure 6 - RMSE Learning Curve Amalgamation (pCA)

Again, we see a large standard deviation in the first sample, but a progressive downward trend leading to more than a full point of improvement. In general the VAE model improved the performance of our predictive model.

The vanillic acid on the other hand was not as easy to interpret. Figure 7 shows the average MAE learning curve for vanillic acid.

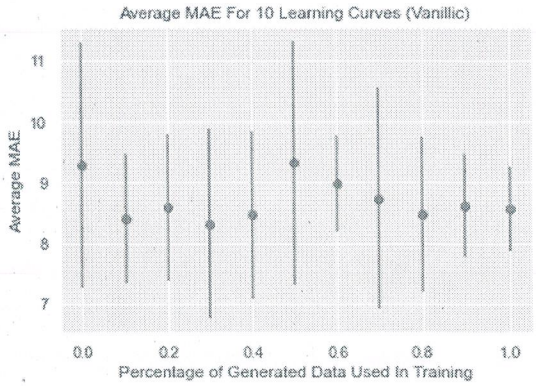

Figure 7 - MAE Learning Curve Amalgamation (Vanillic)

In comparison to Figure 5, the predictive model for vanillic acid did not benefit from a significant increase in accuracy like the p-Coumaric acid did. While there is still ultimately a small increase, the trend is weaker. Similarly, Figure 8 shows the RMSE for vanillic acid.

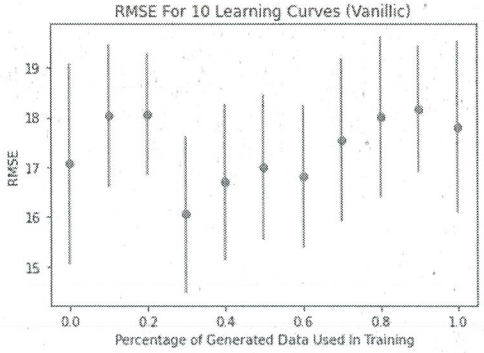

Figure 8 - RMSE Learning Curve Amalgamation (Vanillic)

The curve moves quite sporadically and there is no clear trend. This signals that there must be a high amount of standard deviation among the samples, learning to an inconsistent model. While these results are not ideal we can see that adding 30% of the augmented dataset allowed for some improvement to the vanillic predictive model.

In terms of visual results, Figure 9 shows a single generated curve from our vanillic generative model and Figure 10 shows a generated curve from our p-Coumaric generative model.

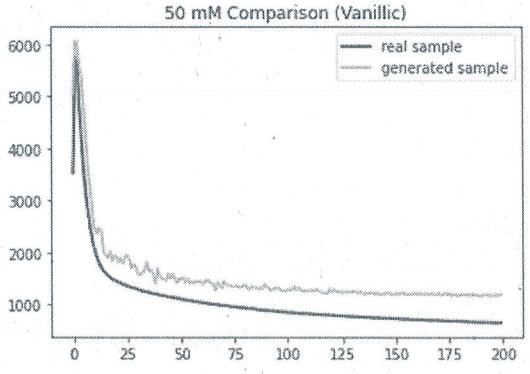

Figure 9 - 50 mM Generated Vanillic Sample Compared to Real Sample

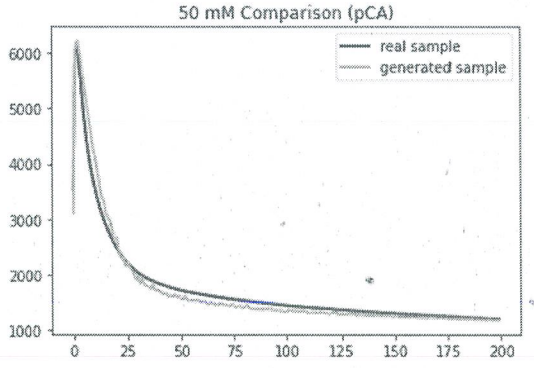

Figure 10 - 50 mM Generated p-Coumaric Acid Sample Compared to Real Sample

The yellow line represents the generated sample and the blue line represents the real sample. Visually speaking our VAE for both the vanillic and p-Coumaric acid models generated realistic looking curves. In combination of these results, the VAE generative model improved both the p-Coumaric and vanillic acid predictive models and generated realistic looking samples for generation.

V. DISCUSSION

While these results were promising there is much room for future work as well as some limiting factors to this research. Given the nature of this problem there was not much training data to begin with for either the VAE or the predictive model. Since we had less than 100 samples for the vanillic and p-Coumaric acid training the VAE was difficult. Usually 1D CNNs would require quite a bit more data to be reliable. Since this data is experimental there could be more data collected in the future. Another future improvement will be increasing the complexity of the VAE model and fine tuning the different layers. There is also room for improvement in the type of optimization function used. While we used the Adam optimizer, according to the general literature, there are many other options.

The generative model is inherently limited by the types of concentration that are in the training set, so there was no ability to create samples of other concentrations. This was not a problem since the predictive model did not have a varied set of concentrations. Another expansion of this research project would be to use other models for data generation. Originally we were going to use a GAN in order to generate the data, however, there was not enough data initially and the time limitation of an undergraduate research project limited the complexity of the model used. If this project is continued it would be beneficial to investigate other generative models.

Visually speaking the curves generated were fairly similar at the most important parts of the curve. The beginning of the curve, the first 25 samples, were considered the most important and unique. Figures 9 and 10 both show this area as being similar between the generated and real samples. There were examples of curves that did not detect the height of the peaks as well, but in general each concentration curve looked different and matched the corresponding peaks.

The predictive model posed many complications for this project, specifically in evaluating the success of our generative model. One major issue was the lack of data for our predictive model. There were only 60 total samples for

p-Coumaric acid and over 120 samples for vanillic acid. This caused differing performance between the two predictive models. This most likely caused the more significant improvement when adding generated data to the p-Coumaric model. Since it originally did not have many samples, any addition would allow for increased accuracy. While the vanillic acid had improvement, it was not as significant because it had more samples to work with. It is also worth noting that the vanillic dataset had more varied concentrations and therefore a larger range to predict, possibly leading to a lower MAE and less effective augmented dataset. Another issue with the predictive model was the lack of tuning. This was out of scope of the project and therefore the predictive model was not optimized. This means that there is future work to be done in this area, possibly improving the results of the generative VAE. Regardless, the visual comparisons were successful and there was still numerical improvement with the addition of generated data. This allows for future work to be done and plenty of potential for meaningful improvement.[1]

ACKNOWLEDGEMENTS

I would like to thank Rodney L. Summerscales, Ph.D., at the Andrews University departments of Engineering and Computing for mentoring and guiding this project as well as my learning. I would also like to thank Hyun Kwon, Ph.D., for providing leadership on this project, as well as Reise Campbell for assistance with preliminary data cleaning.

[1]https://github.com/solomonjkim/1D-Variational-Autoencoder

REFERENCES

[1] A. Mikołajczyk and M. Grochowski, "Data augmentation for improving deep learning in image classification problem," *2018 International Interdisciplinary PhD Workshop (IIPhDW)*, Świnoujście, Poland, 2018, pp. 117-122, doi: 10.1109/IIPHDW.2018.8388338.

[2] K. Team, "Keras documentation: Variational AutoEncoder," *Keras*. [Online]. Available: https://keras.io/examples/generative/vae/. [Accessed: 17-Mar-2021].

[3] "Generate Artificial Faces with CelebA Progressive GAN Model," *TensorFlow*. [Online]. Available: https://www.tensorflow.org/hub/tutorials/tf_hub_generative_image_module. [Accessed: 17-Mar-2021].

[4] X. Yan, J. Yang, K. Sohn, and H. Lee. Attribute2image: Conditional image generation from visual attributes. European Conference in Computer Vision, 2016.

[5] Y. Luo, L.-Z. Zhu, Z.-Y. Wan, and B.-L. Lu, "Data augmentation for enhancing EEG-based emotion recognition with deep generative models," 2020, arXiv:2006.05331. [Online]. Available: http://arxiv.org/abs/2006.05331

[6] "Convolutional Variational Autoencoder : TensorFlow Core," *TensorFlow*. [Online]. Available: https://www.tensorflow.org/tutorials/generative/cvae#reparameterization_trick. [Accessed: 17-Mar-2021].