Compress 4 variables into 3 using Autoencoders

Rushabh Vasani[1]

[1]*Department of Information Technology,*
Chandubhai S. Patel Institute of Technology, Charusat University,
Changa, Gujarat, India

[1]vasanirushabh24@gmail.com

***Abstract* - Compressing the data is a big concern nowadays in order to optimize hard disk usages. In this paper we have tried to explain my approach to compress (encoding) the data of 4 variables in 3, storing them on the disk and then decompress(decoding) the encoded variables and get the original data back. This can be considered as a smaller version of image compression as images are also a combination of variables. And the main motivation for this is to try and apply similar approaches to image compression in the future.**

Keywords: *Variable encoding, Machine Learning, Autoencoder, Singular value decomposition.*

I. INTRODUCTION

Here we have used Autoencoders[1] to compress the data. An Autoencoder[1] is a neural network[2] that learns to reduce it's input and then to reconstruct the original input from the reduced version (encoded).

The autoencoder consists of 2 internal neural Architectures. One is the encoder part and the other is the decoder.

1. **Encoder:** The layers that encode the data into code (encoded /compressed data).
2. **Decoder:** The layers that reconstruct the original data (decoded/decompressed data) from the code.

And we have used some techniques like singular value decomposition, one cycle training approach, etc in order to make the model learn better.

II. DETAILS

The entire procedure can be divided into 3 portions:

1. **Preprocessing**: To extract useful information from the given data in order to model learn better.
2. **Model definition**: Deciding and defining the perfect model architecture for the task.
3. **Training:** training the model with the gradient descent approach.

III. PREPROCESSING

This step refers to manipulate the input data to make the training process smoother. My preprocessing was a 2 step process.

a. **Normalization**[3]: Make the inputs having a mean of 0 and a standard deviation of 1. This fixes the outliers of the data and makes the learning process much easier, smoother, and faster for the computer.
b. **Adding Singular values**: singular value decomposition (SVD)[4] is a mathematical process (basically matrix operations) to factorize a matrix into a lower dimension of our choice. I first constructed a (4 x 4) diagonal matrix from those 4 variables. And then computed 3 singular values of that matrix.

IV. MODEL DEFINITION

I have used a lot of nn.Linear, nn.BatchNorm1d[5] and nn.Tanh layers. Basically nn.Tanh is there to add a non-linearity[6].

non-linearity:

Because without non-linearity the model would become a bunch of linear functions(layers here) and a combination of more than 1 linear function is again a linear function.

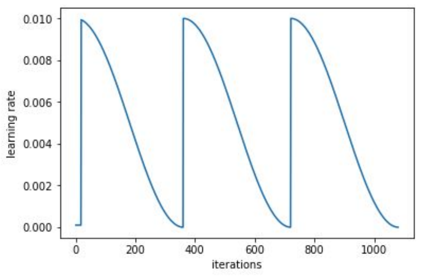
There are other non-linear functions like nn.ReLU as well but nn.Tanh turned out to work better in this case when we tested.

BatchNorm:

As we normalize our input data to make the learning process faster, Batchnorm does the same thing for the hidden layers. It normalizes the activations to make learning faster and easier.

V. TRAINING

I used the Cyclical Learning Rates[7] (CLR) approach. Which eliminates the need to experimentally find the best values for learning rates. Instead of decreasing the learning rate, this method lets the learning rate cyclically vary between reasonable boundaries.



I have used Mean Squared Error (MSE) as my loss function[8] because it best describes how close our reconstructed tensor (matrix) and our target tensor (matrix) are that is the reason why we have to choose to mean squared error loss.

Otherwise, there are other loss functions like cross-entropy loss, cross-entropy with logits loss, mean squared error with logits loss. Root mean squared loss. But all of those have their own use cases. Root mean squared is very similar to the mean squared error even it's just a rooting square and then mean squared error loss but the mean squared error turned out to be performing better as it compares the data better than every other loss function.

VI. FUTURE SCOPE

Similar kinds of Autoencoder (with convolutional layers) can be used for Image compression to save memory. This is mostly needed to store the satellite images as they are very large in size. So if we could compress them and then decode them whenever needed then it can become very useful for organizations like ATLAS, NASA, ISRO to store the satellite images.

VII. RESULTS

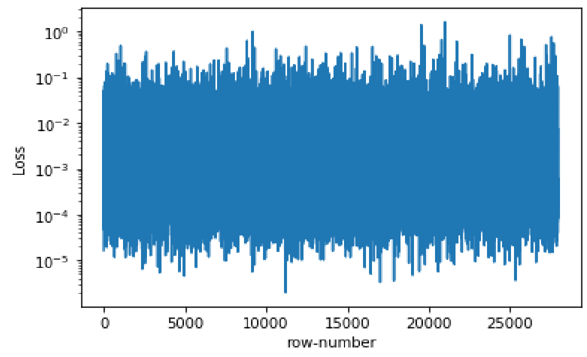
Even though CNNs[9] tend to perform better in general for similar tasks, Linear NN results were better for the given task.

A. Performance of the different approaches

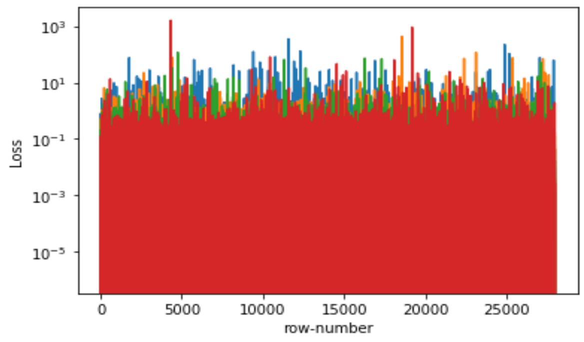
Architecture	**MSE loss**
CNN	~0.053
NN with only Linear and Tanh layers	~0.021

NN with BatchNorm layers	0.0161
NN with BatchNorm layers and with singular values added in the data	0.0073

B. Validation Losses plotted on the graph



plots the row-wise average Mean Squared Error loss for each data entry in the test set. The average loss is 0.0073.



- The above graph plots the row-wise *[(expected-result) / expected]* values for each data entry in the test set for every column separately.
- Here the range is up to 10^3 because of dividing *(expected - result)* by smaller numbers.

VIII. Conclusion

Successfully built a 4 to 3 variable autoencoder with 0.0073 Mean Squared Error Loss. This still can be further enhanced to give better results. In the future, an image compression model can be built with similar kinds of technologies and algorithms and provided the experience of this variable Autoencoder project.

References

[1] L. N. Smith, A research paper on cyclic learning rates [April 2017]

[2] Following links referred:
1. https://en.wikipedia.org/wiki/Autoencoder
2. https://en.wikipedia.org/wiki/Neural_network
3. https://en.wikipedia.org/wiki/Batch_normalization
4. https://en.wikipedia.org/wiki/Activation_function
5. https://arxiv.org/abs/1506.01186
6. https://en.wikipedia.org/wiki/Normalization_(statistics)
7. https://en.wikipedia.org/wiki/Singular_value_decomposition
8. https://en.wikipedia.org/wiki/Loss_function
9. https://en.wikipedia.org/wiki/Convolutional_neural_network

GitHub repository of this project can be found here.