Lossy Compression using Neural Networks

Divam Gupta
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213, USA
divamg@andrew.cmu.edu

Viraj Parimi
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213, USA
vparimi@andrew.cmu.edu

1 Introduction

Learning compact representations of data in an unsupervised setting is one of the key challenges in machine learning. In the past, extensive work has been done in the domain of continuous latent space representations of the input data using Variational Autoencoders(VAEs) which are capable of modelling the data distribution as a non-linear transformation of the unobserved latent variables in a generative setting. In this project, we intend to pursue an alternative chain of thought which works with discrete latent space representations without significant performance implications with respect to the continuous counterparts.

We first start with a vanilla autoencoder model with a continuous latent output. As we need floating point to store the continuous vectors, the compression ratio is not very good. To improve the compression ratio, we try a multiple techniques. Firstly we try a naive quantization technique. Then we tried incorporate quantization with the loss function to make the outputs closer to the quantized numbers and finally we also tried a training time quantization technique.

We implement and experiment these compression techniques in three models - Multilayer Perceptron(MLP) model, Convolutional Neural Network(CNN) model and Long Short Term Memory networks(LSTM) model. We show empirical results on the MNIST [5], CIFAR10 [6] and ANKI [7] text dataset. Details about these datasets can be found in Appendix.

The code of our implementations can be found at `https://github.com/divamgupta/dnn_lossy_compression`

2 DNN based Compression Methods

2.1 Autoencoders

Autoencoders are one of the most popular approaches that are used to learn continuous latent space representations of the input data in an unsupervised setting. These models are comprised of two separate functions, the encoder ("recognition") function that performs a non-linear transformation on the input data to learn the latent variables and a decoder function that tries to reconstruct the input data from the learned representations. Additionally sequence autoencoders are just standard autoencoders which have been adapted to work with sequence data by incorporating LSTM RNN cells. Doing this allows the model to learn the temporal ordering of the sequence data allowing the model to learn better compressed representation.

2.2 Naive Quantized Autoencoder

In this technique we first train a vanilla autoencoder with sigmoid activations in the bottleneck representation layer. By adding sigmoid we ensure that the values are between 0 and 1, hence doing a threshold quantization should not change it drastically. While training no thresholding is done, but at

Preprint. Under review.

inference time the sigmoid output is thresholded. If the output of the sigmoid is more than 0.5, it is replaced by 1, else it is replaced by 0. In our experiments we found out that by using this technique there is a big drop in quality of the reconstructions.

2.3 Quantized AutoEncoder with fence loss

There is a big performance drop if we naively quantized the encoder outputs of the autoencoder. This is because many values of the latent space are near 0.5 ("sitting on the fence"). Hence, to circumvent that we try a loss function which penalizes the encoder for not having the values close to 0 or 1. Such loss function can be defined as follows,

$$\text{Fence Loss} = \frac{1}{N}\sum_{i} \min(x_i, 1 - x_i)$$

where N is the size of latent space. Using this technique the distribution of the latent vector during training time is close to the distribution of the quantized latent vector.

2.4 Training time Quantized Autoencoder with commit loss

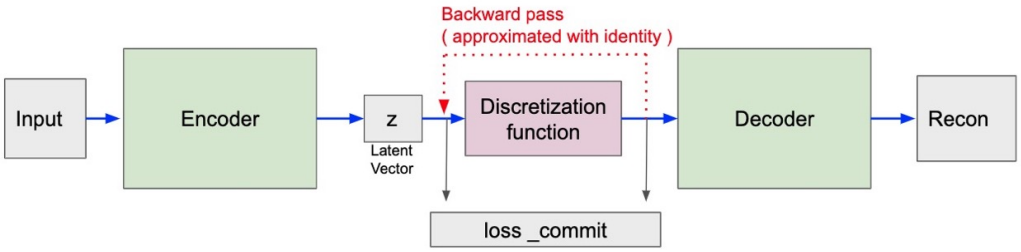

Figure 1: Training time Quantized Autoencoder

This model is similar to the naïve quantized model, but we do the quantization of the bottleneck representation layer at training time itself. We use the same threshold based quantization - if output is more than 0.5, it is replaced by 1, else it is replaced by 0. This is not a differentiable operation hence during back propagation, the quantization is skipped. This is equivalent to approximating the gradient by the identity operation.

$$\text{Commit Loss} = \frac{1}{N}\sum_{i}(h_a^i - h_b^i)^2$$

where N is the size of the latent space,h_a is the latent space after discretization and h_b is the latent space after discretization. This discretization can be represented as,

$$\text{Binarize}(h) = \begin{cases} 1 & h > 0.5 \\ 0 & \text{otherwise} \end{cases}$$

Similar to VQVAEs, this is not stable during training, hence we have to add a commitment loss. Here we minimize the L2 distance between the latent space before and after quantization. We have to ensure that the output of the encoder before discretization is similar to the output after thresholding. Example, the outputs before thresholding should be either close to 0 or close to 1. By this the encoder learns to assign discrete codes for given inputs.

3 Models

We implemented the above discussed ideas and incorporated them in different types of autoencoders where each model is designed to target certain type of input domain.

3.1 Multilayered Perceptron

These models make use of the MLP to create dense connections between the input and hidden latent space. This interconnection forms the encoder. Further it also creates dense connection between the hidden latent space back to the original input space which forms the decoder. Even though these dense connections are memory intensive, they help in capturing the latent space more thoroughly.

3.2 LSTM

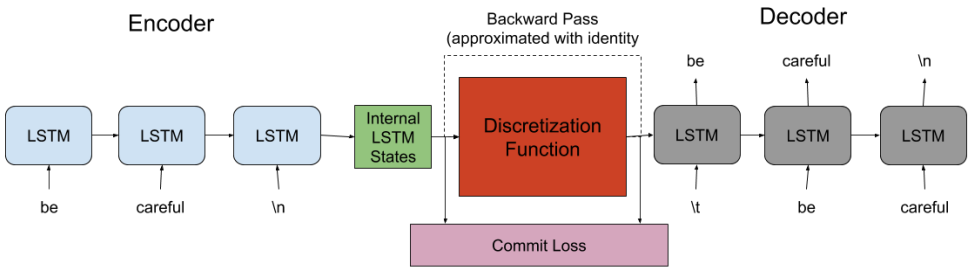

Figure 2: Training time Quantized LSTM Autoencoder with Commit Loss

This model is an extension of the widely used seq2seq [8] models. An RNN layer that acts as an encoder processes the input sequence and passes on its hidden states to another RNN layers which acts a decoder. We used a character level model where the decoder uses the hidden states of the encoder as its initial state and predicts a character till the end of sequence. Such models are equipped to handle textual datasets.

3.3 Convolutional Neural Networks

These models make use of the highly popular CNNs which are more robust at capturing image features. We first pass the input through a encoder convolutional layer which is them downsampled using maxpooling. This is then passed through a decoder convolutional layer which after upsampling returns back to the input space. Such models are more adept at retreiving more information while being efficient compared to MLP.

4 Metrics

In order to quantify the performance of the methods we have to use two different set of metrics which are suitable for either the textual domain or image domain. For images we use PSNR, SSIM, MSE, MAE and CR. For texts we use Levenshtien, Cosine and Jaccard distance. Additionally we also use BLEU score. Details of these metrics can be found in the appendix.

5 Experimental Results

We implement and experiment with all the models described above. To get better insights of the performance of each model, we experiment with multiple model sizes. By increasing the model size, the compression ratio decreases but the quality of the compression increases. We can observe this trade-off for all the models.

5.1 Multilayered Perceptron

For the MLP network we perform experiments on the MNIST dataset. The input size of the model is a vector of length 784. We experiment with model of hidden size 10, 50 and 100. The quantitative results are mentioned in Table 1. For model with 50 and 100 hidden units the MAE of the model with

commit loss is better. Visually (Figure 3) we can see that the reconstructions for models with small hidden units are very bad. The results look more noisy for model with naive quantization.

Table 1: Multilayered Perceptron based Autoencoder results on MNIST

Model/Metric	PSNR	SSIM	CR	MSE	MAE
Vanilla Autoencoder 10	31.2751	0.3861	19.6	48.8886	119.2946
Naive Quantized Autoencoder 10	31.8633	0.4221	627.2	42.9833	96.4650
Commit Loss Quantized Autoencoder 10	31.7965	0.4180	627.2	44.8423	99.6676
Vanilla Autoencoder 50	32.3890	0.6910	1.96	38.4118	94.0038
Naive Quantized Autoencoder 50	33.0452	0.5230	62.72	33.2511	74.9453
Commit Loss Quantized Autoencoder 50	32.9476	0.6812	62.72	34.6837	70.0960
Vanilla Autoencoder 100	33.8850	**0.8651**	0.196	27.7169	71.5690
Naive Quantized Autoencoder 100	**34.2288**	0.6527	6.272	**25.5917**	64.0222
Commit Loss Quantized Autoencoder 100	33.5612	0.7730	6.272	30.2342	**59.7807**

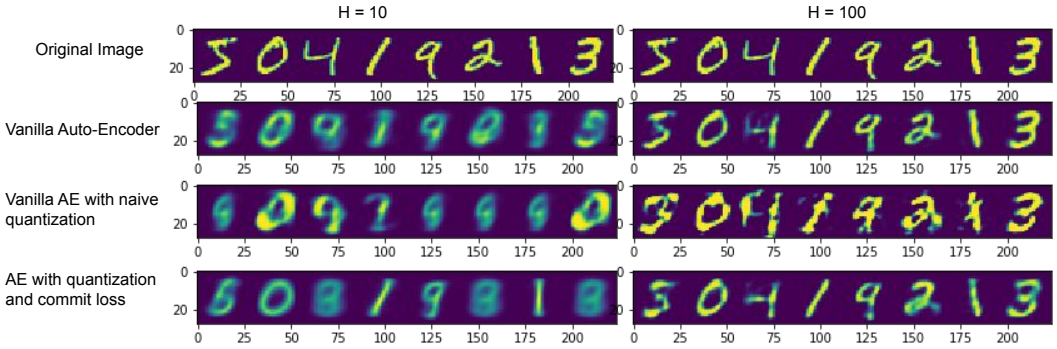

Figure 3: Qualitative results of MLP model on the MNIST dataset. We see that the quality decreases if we naively do quantization on the autoencoder model. We also see that the quantized model with commit loss looks pretty much like the non-quantized vanilla model.

5.2 Convolutional Neural Networks

For the CNNs network we perform experiments on the CIFAR10 dataset. The input size of the model is a image of size 32x32x3. We experiment with model of different size. The latent dimension of the small, medium and large are 4x4x16 , 4x4x32 and 4x4x48 respectively.

The quantitative results are mentioned in Table 2. For all the sizes we can observe an improvement of model with commit loss over the naive quantization model. The fence loss does not show any improvement. Visually (Figure 4) we can see that the reconstructions for models with naive quantization are very bad. The reconstruction of the discrete model with commit loss looks very similar to the continuous vanilla autoencoder.

5.3 LSTM

For the LSTM network we perform experiments on the ANKI dataset. The input size of the model is of size 22x38 . We experiment with model of hidden state sizes - 32, 64, 128, 256.

The quantitative results are mentioned in Table 3. We observe that the commit loss model performs better on the Levenshtien and BLEU score. Whereas the naive quantized model performs better on Cosine and Jaccard distances. Visual details of the models are presented in the appendix.

Table 2: Convolutional Neural Network based Autoencoder results on CIFAR10

Model/Metric	CR	PSNR	SSIM	MSE
Vanilla CNN Autoencoder Small	3	29.0800	0.9013	80.3677
Naive Quantized CNN Autoencoder Small	96	27.9230	0.5404	104.9021
Commit Loss Quantized CNN Autoencoder Small	96	**28.6237**	**0.8413**	**89.2711**
Fence Loss Quantized CNN Autoencoder Small	96	27.9546	0.5562	104.1400
Vanilla CNN Autoencoder Medium	2	29.4380	0.9323	74.0077
Naive Quantized CNN Autoencoder Medium	48	27.8886	0.6400	105.7349
Commit Loss Quantized CNN Autoencoder Medium	48	**28.8654**	**0.8842**	**84.4380**
Fence Loss Quantized CNN Autoencoder Medium	48	28.0009	0.6909	103.0374
Vanilla CNN Autoencoder Large	1	29.5720	0.9425	71.7605
Naive Quantized CNN Autoencoder Large	32	28.1532	0.7706	99.4850
Commit Loss Quantized CNN Autoencoder Large	32	**28.8930**	**0.9006**	**83.9042**
Fence Loss Quantized CNN Autoencoder Large	32	28.0028	0.7566	102.9923

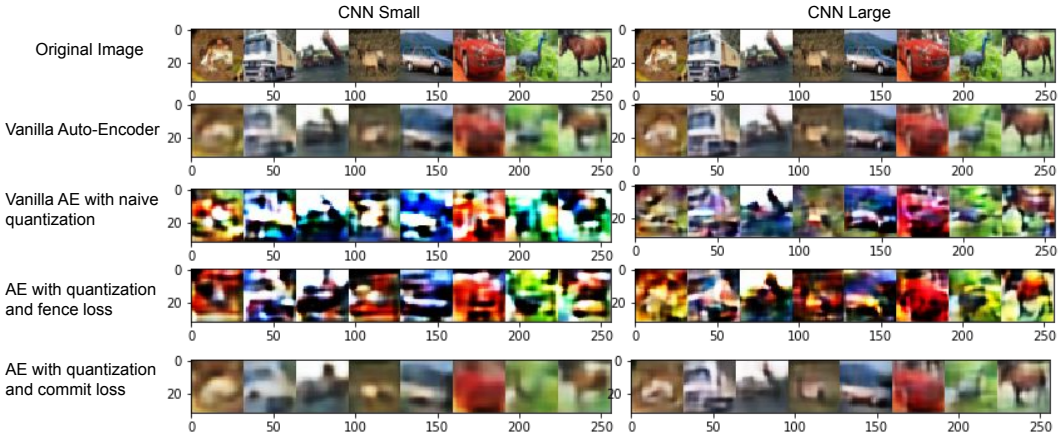

Figure 4: Qualitative results of CNN model on the CIFAR10 dataset. We see that the quality decreases if we naively do quantization on the auto-encoder model. We also see that the quantized model with commit loss looks pretty much like the non-quantized vanilla model.

Table 3: LSTM based Autoencoder on ANKI

Model/Metric	Levenshtien	Cosine	Jaccard	BLEU
Vanilla LSTM 32	10.36	0.1066	0.236	0.7762
Naive Quantized LSTM 32	15.2	**0.2004**	0.4705	0.8503
Commit Loss Quantized LSTM 32	**13.29**	0.2376	0.4843	**0.8295**
Fence Loss Quantized LSTM 32	16.14	0.2161	**0.4455**	0.8909
Vanilla LSTM 64	3.57	0.0379	0.0878	**0.4447**
Naive Quantized LSTM 64	15.07	**0.1686**	**0.3495**	0.8902
Commit Loss Quantized LSTM 64	**13.07**	0.2028	0.4327	**0.8023**
Fence Loss Quantized LSTM 64	14.78	0.1971	0.4306	0.9003
Vanilla LSTM 128	2.25	0.0136	0.0165	0.2151
Naive Quantized LSTM 128	12.2	**0.1347**	**0.3147**	0.8209
Commit Loss Quantized LSTM 128	13.21	0.2233	0.4706	0.8123
Fence Loss Quantized LSTM 128	**11.39**	0.1879	0.4044	**0.7665**
Vanilla LSTM 256	3.38	0.0284	0.0627	0.3392
Naive Quantized LSTM 256	10.04	0.1771	0.3589	0.7165
Commit Loss Quantized LSTM 256	10.6	**0.1643**	**0.3502**	0.7212
Fence Loss Quantized LSTM 256	**9.86**	0.1799	0.3859	**0.7163**

References

[1] Bowman, Samuel R. et al. "Generating Sentences from a Continuous Space." Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning (2016): n. pag. Crossref. Web.

[2] Zhou Wang, A. C. Bovik, H. R. Sheikh and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," in IEEE Transactions on Image Processing, vol. 13, no. 4, pp. 600-612, April 2004.

[3] Wikipedia contributors. "Peak signal-to-noise ratio." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 13 Apr. 2020. Web. 4 May. 2020.

[4] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: a method for automatic evaluation of machine translation. In Proceedings of the 40th Annual Meeting on Association for Computational Linguistics (ACL '02). Association for Computational Linguistics, USA, 311–318. DOI:https://doi.org/10.3115/1073083.1073135

[5] LeCun, Y. & Cortes, C. (2010), 'MNIST handwritten digit database', .

[6] Alex Krizhevsky https://www.cs.toronto.edu/ kriz/cifar.html

[7] ANKI http://www.manythings.org/anki/

[8] Keras Blog https://blog.keras.io/a-ten-minute-introduction-to-sequence-to-sequence-learning-in-keras.html

6 Appendix

6.1 Datasets

- MNIST - It is a handwritten digits database where each grayscale image is 28 $\times$ 28 pixels. It has 60k training images and 20k testing images.
- CIFAR10 - It is a collection of color images with 10 classes where each image is 32 $\times$ 32 pixels. It contains 60k training images where each class contains 6k samples.
- ANKI - It is a text based dataset which contains a plethora of bilingual sentence pair files. This was originated as a teaching tool to learn new languages. Each file contains a list of sentence pairs which get more complex as the file progresses.

6.2 Metric Details

6.2.1 Image

For the image based models we used the following metrics that are widely used for this purpose.

- Peak Signal to Noise Ration (PSNR) [3] - This metric is designed specifically for images and is defined as follows,

$$PSNR = 10 \cdot \log_{10} \left(\frac{MAX_I^2}{MSE} \right)$$

  where MAX_I is the maximum possible intensity value of the input image and MSE represents the mean squared error between the input image and its noisy approximation. The value is represented in decibels and higher is better.

- Structural Similarity (SSIM) Index [2] - This metric was again designed to quantify the similarity between two images. It is an improvement over $PSNR$ and MSE as it considers the idea that pixels have strong inter-dependencies especially when they are close. The measure is defined over two windows (generally gaussian) with common sizes.
- Compression Ratio (CR) - It is defined as follows,

$$CR = \frac{\text{Size of uncompressed data}}{\text{Size of compressed data}} \tag{1}$$

  The metric allows us to quantify the extent of data savings that we gain from a particular method.

- Mean Squared Error (MSE) - It is also a widely used metric for quantifying compression effectiveness. Ideally MSE should be lower for better approaches.
- Mean Absolute Error (MAE) - This is one variant of MSE which allows one to similarly quantify the differences between two images.

6.2.2 Text

For the text based models we utilized the following metrics to evaluate the quality of the reconstructed text.

- Levenshtien Distance - This is a string metric that is defined to measure the difference between two sequences. It can quantified as the minimum number of single-character edits (i.e. insertions, deletions, or substitutions) required to change one sequence into the another sequence.
- Cosine Distance - Cosine Similarity between two sequences is just the cosine of angle between these sequences. This distance hence can be calculated by subtracting it from 1. This metric is advantageous as even if the two input sequences are far apart in terms of euclidean distance, they may still have the same orientation.
- Jaccard Distance - This metric also known as the intersection over union is defined as the size of the intersection divided by the size of the union of two sets. We first convert the input sentences into two different sets using lemmatization and then performing the necessary operation. The distance then can be calculated by subtracting this from 1. One downside of this metric is that as the length of the input sequences increases, the size of the intersection set tends to increase as well.
- BLEU Score [4] - This metric was initially created to compare the quality of machine translated sentences when given a set of quality reference solutions. For our purpose we used the original input sentence as our reference solution and then compare the reconstructed text against that reference using BLEU algorithm. Since BLEU was defined for 4-grams, we use the smoothed out version which allows us to calculate the score even if the input sentence is smaller than 4 words. Again to get the distance we simply subtract this quantity from 1.

6.3 LSTM Autoencoder Results

Method	Example 1	Example 2
Original Text	wheres your daughter	whats your full name
Vanilla Auto-Encoder	where beas you spees	what look for leared
Vanilla AE with naive quantization	were seere bar aw yough	whot fore frerelingry
AE with quantization and fence loss	wherd youre deed that	show deve youre feally
AE with quantization and commit loss	wheres your so there	whats your to starte

Figure 5: Qualitative results of LSTM model on the test dataset. We see that the quality decreases if we naively do quantization on the auto-encoder model. Due to the small size of the bottleneck layer, the quality of the compressed text is bad. But we see that the model with commit loss is better than tan than the niave quantization model. The fence loss also seems to help to a small extent.