15.4.2019
Itai Gat, Dor Cohen

DL - 097200 - Course project report

Introduction

We have experimented with different autoencoder architectures, our interest was to find if autoencoder keeps the context when samples dimensions are reduced. Our method was to measure if samples that were close to each other in original dimensions were still close once reduced. Our benchmark was MNIST and we have used the whole dataset which consists of 60,000 images where each has 28*28 = 784 pixels.

Theory

An autoencoder is a neural network which is trained to copy its input to its output, it has two parts internally: an encoder function **h=f(x)**, and a decoder which produces the reconstruction of the output **r=g(h)**.

Experimental setting

Firstly we have measured the pairwise distance of original samples, and kept the 200 most close samples for each. Then we have trained 4 different architectures for autoencoder, consisting of linear and non-linear activation function, with loss functions L1 and MSE. Finally we used the trained encoder functions to measure encoded samples distances in the same manner. We have also kept the mapping between original samples indices and labels, then practically we had for each sample the 200 most close samples indices, labels and distances values.

Model architectures

Our models were based on the following architectures:

Linear:
encoder: input(784) -> fc(784,128) -> fc(128, 64) -> fc(64,12) -> fc(12,2)
decoder: fc(2,12) -> fc(12, 64) -> fc(64,128) -> fc(128,784)

Non-Linear:
encoder: input(784) -> fc(784,128) -> relu -> fc(128, 64) -> relu -> fc(64,12) -> relu -> fc(12,2)
decoder: fc(2,12) -> relu -> fc(12, 64) -> relu -> fc(64,128) -> relu -> fc(128,784) -> tanh

The naming conventions we have used are: *linear_l1* and *linear_mse* for training the Linear architecture with L1 or MSE loss functions, while *relu_l1* and *relu_mse* were used to note the non-linear architectures.

Training procedure

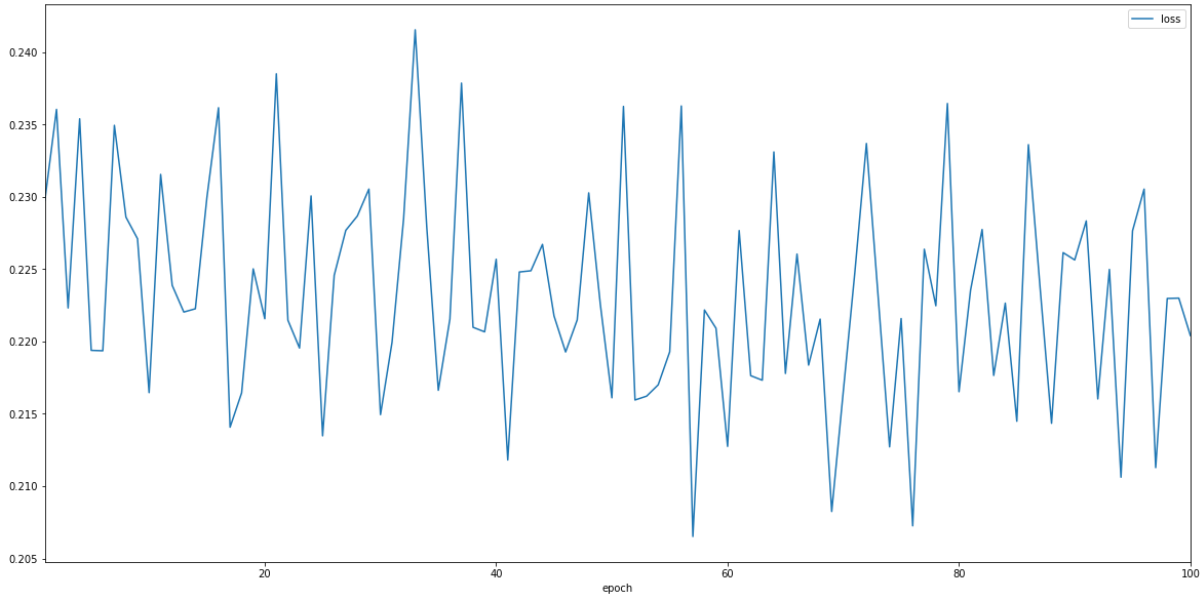
To optimize the network parameters we used Adam optimizer with initial learning rate of 1e-3 and weight decay of 1e-3, betas and epsilon values were Pytorch defaults. Fed data by batches of size 100 for 100 epochs.

15.4.2019
Itai Gat, Dor Cohen

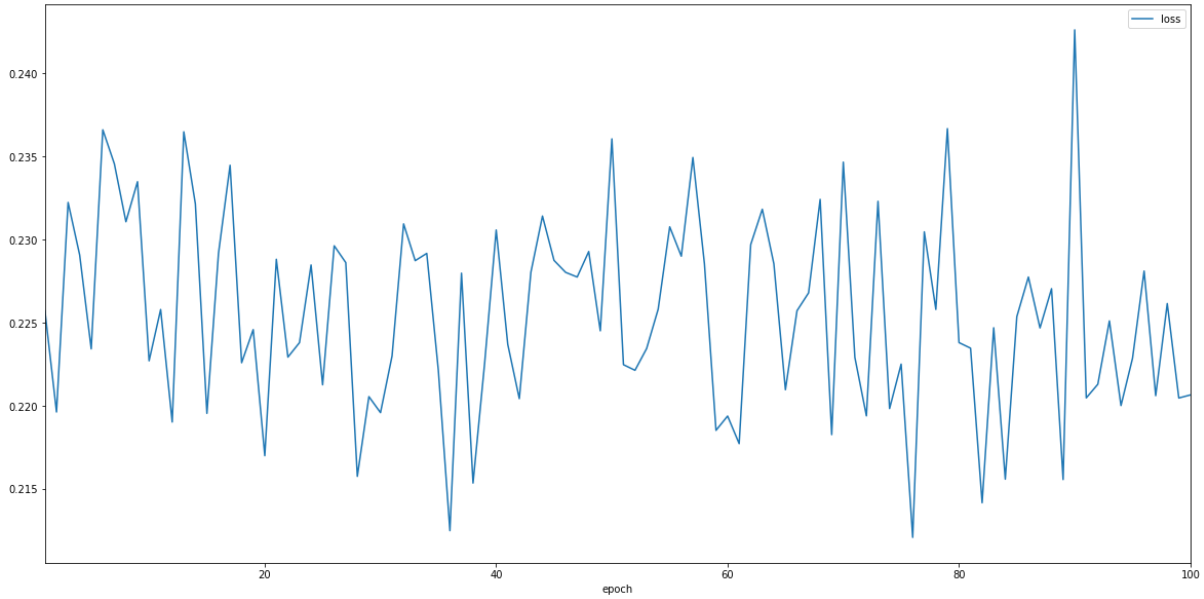
Results

We start by presenting the training loss curves for each model:

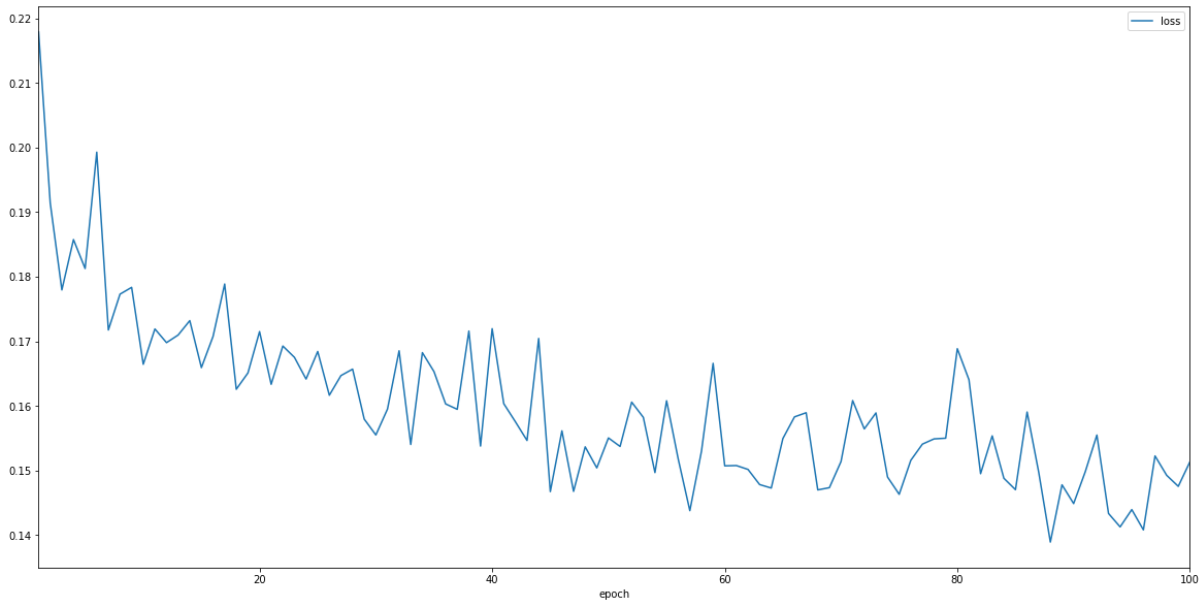
linear l1



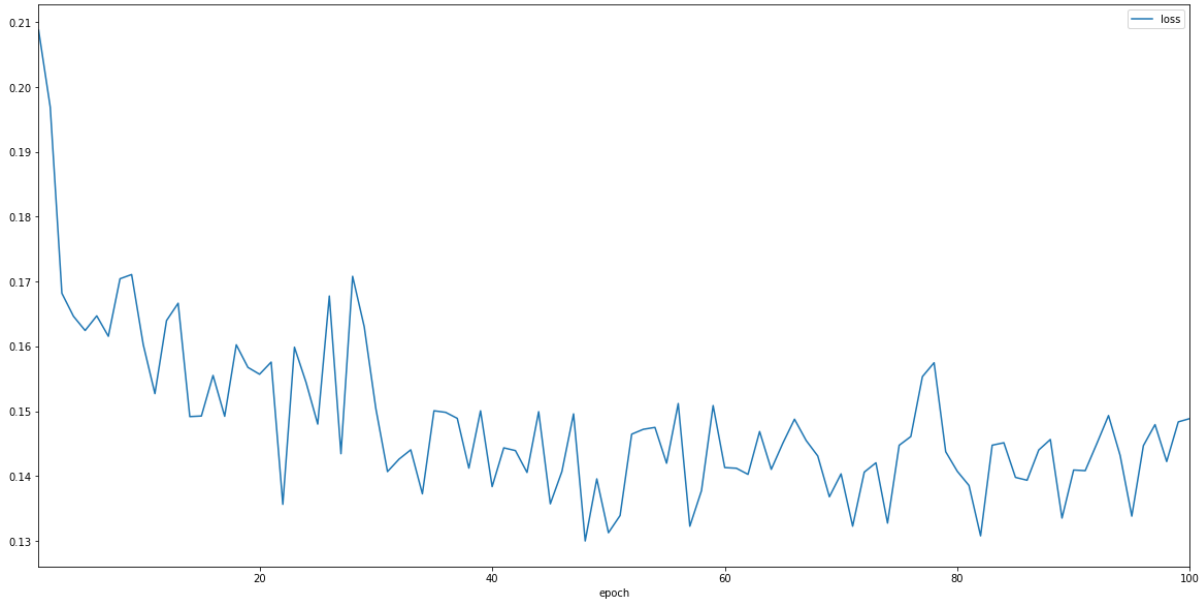
linear mse



relu l1



relu mse

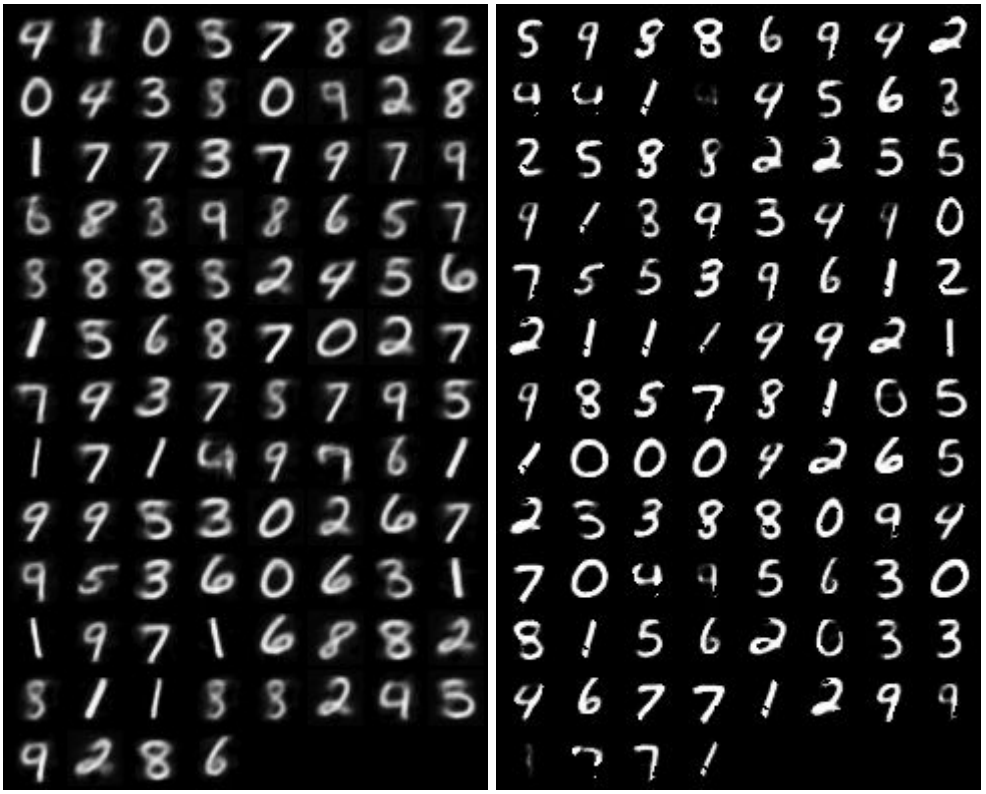


We can clearly see how the nonlinearity improves the network representation.

Let us also compare samples which were passed forward through the trained autoencoder after 90 epochs, it should also convince us with quality of the latent representation:

linear l1 vs. mse

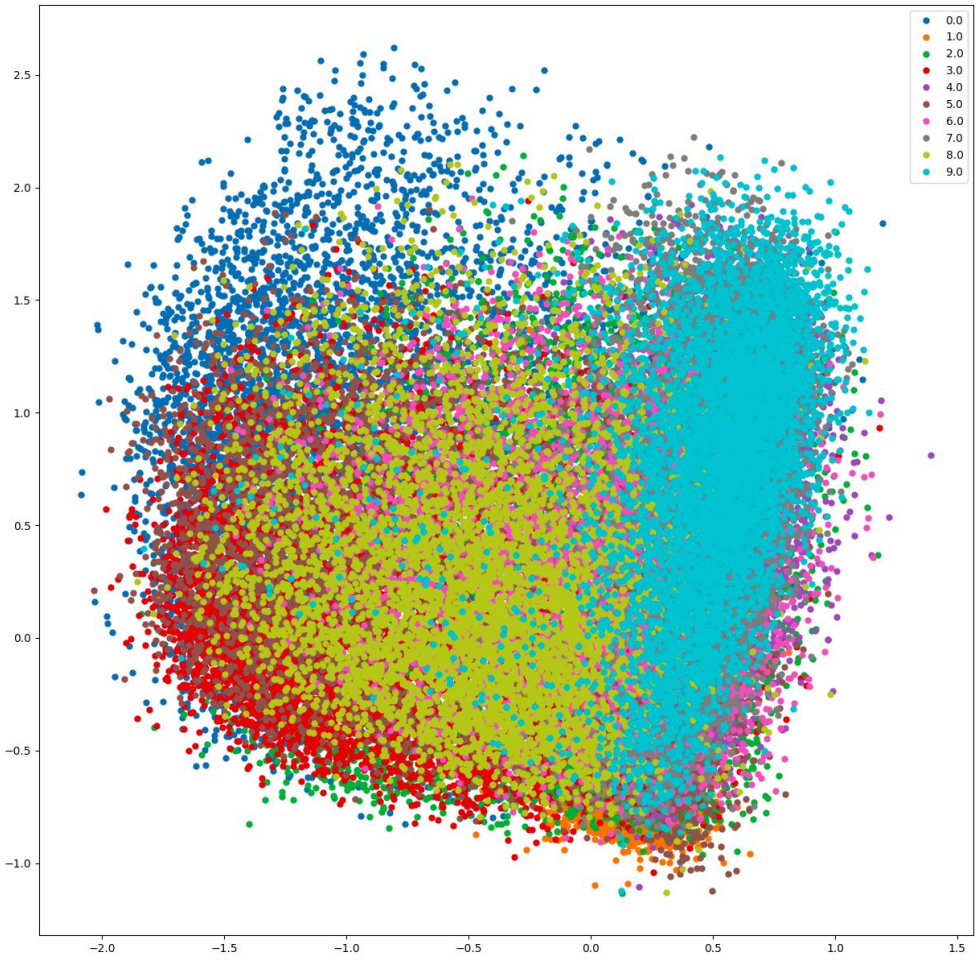
relu l1 vs. mse



It is also clear visually that Relu based autoencoder is superior to non-linear one.

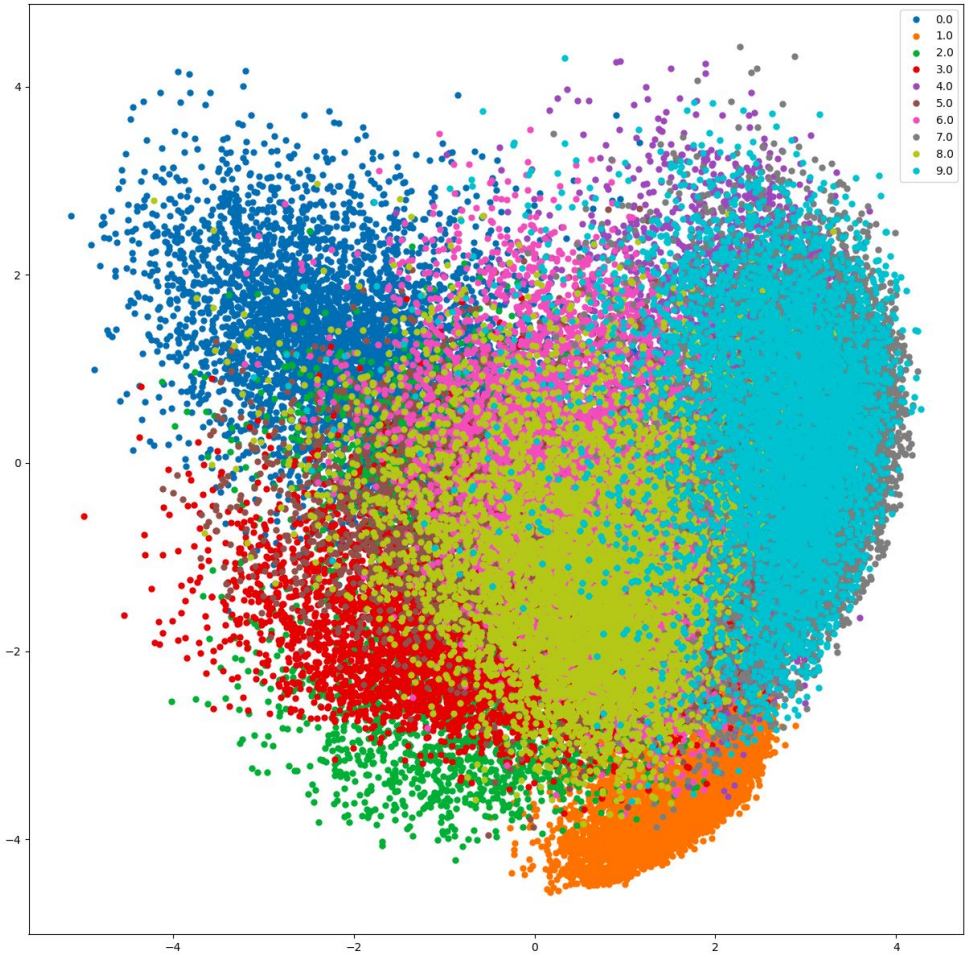
Next we have plotted the encoded representation of original samples, where each has its own color according to its label:

linear l1



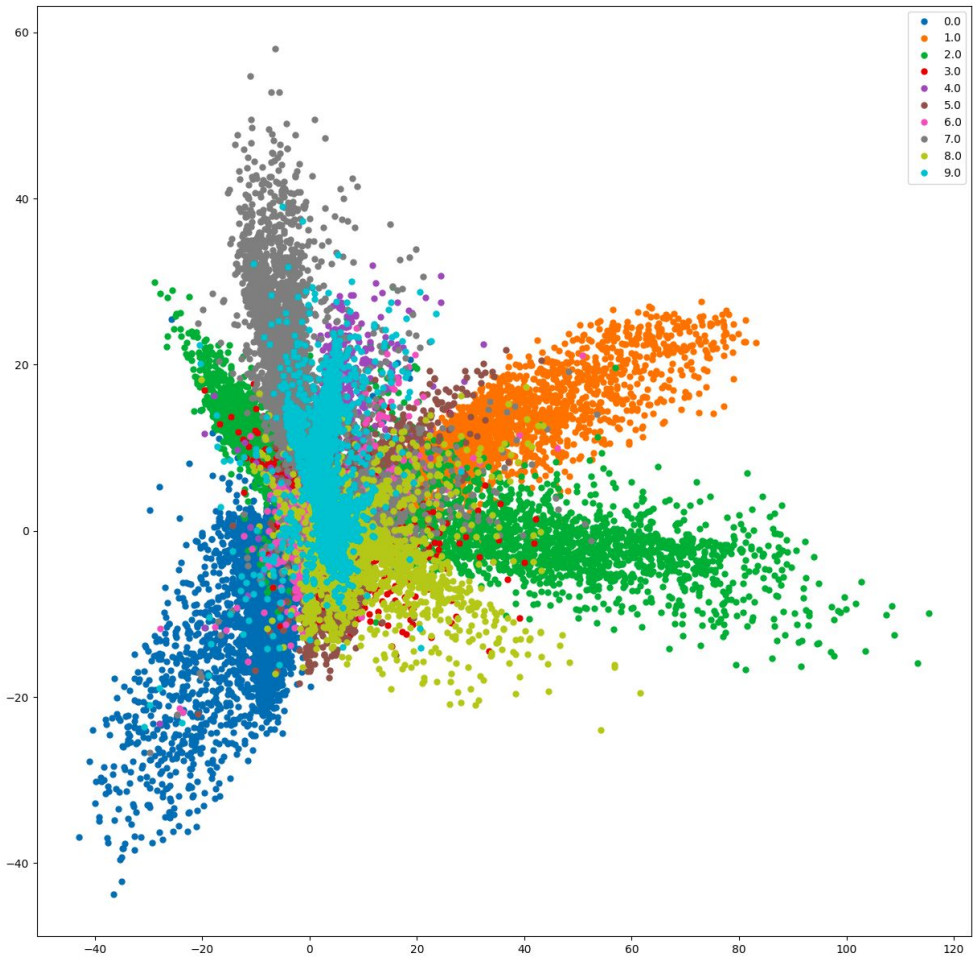
15.4.2019
Itai Gat, Dor Cohen

linear mse



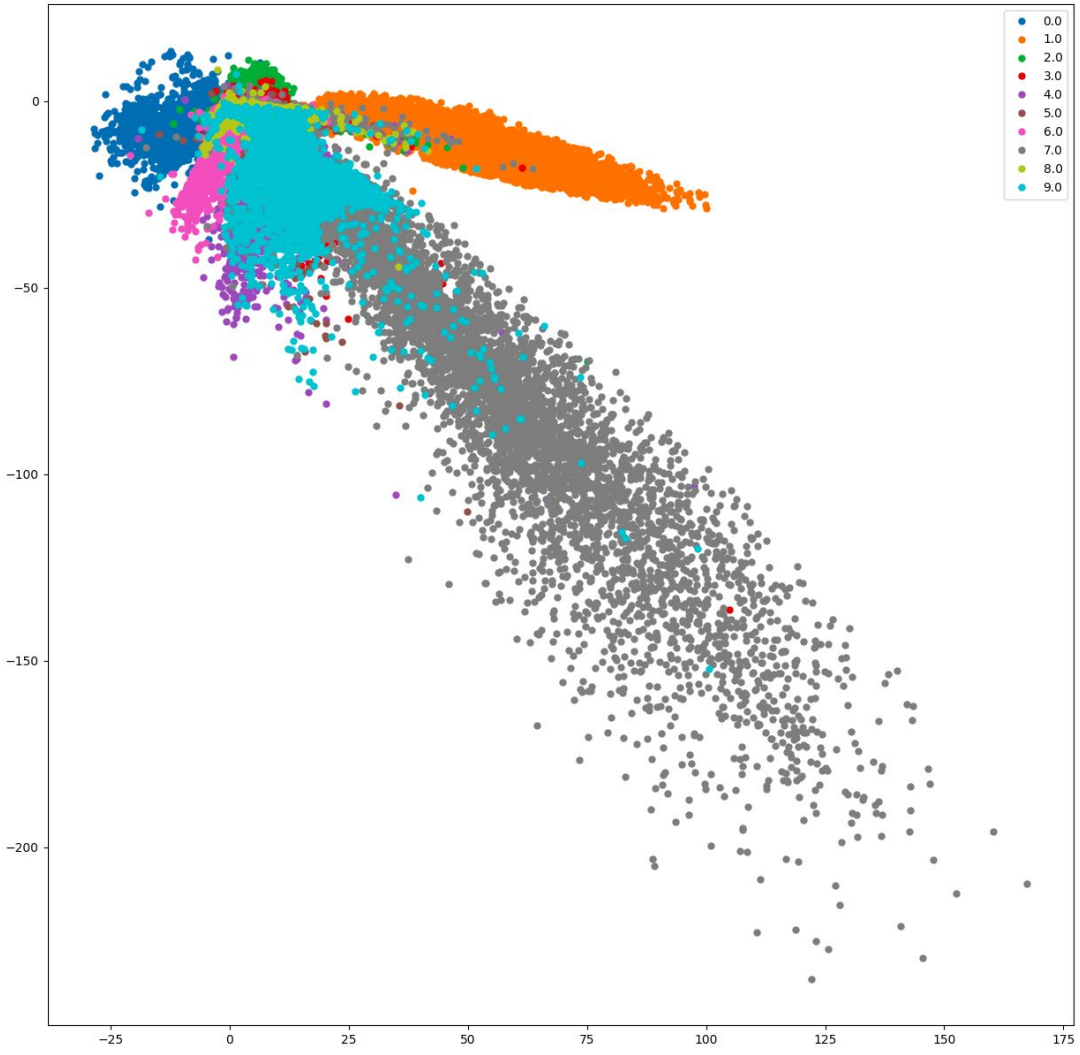
15.4.2019
Itai Gat, Dor Cohen

relu l1



15.4.2019
Itai Gat, Dor Cohen

relu mse



The scatter plots shows that particularly labels ‘1’ and ‘7’ get relatively isolated clusters, while for other samples it is hard to see that.

Finally we show our empirical question result, we have used few metrics and we shall describe each of them in the following part:

1. Percent of samples that were kept close originally vs encoded (for example score@5 means the percent of original vs. encoded samples for the top 5 closest samples, averaged on all samples)

model_name	distance_type	score@5	score@10	score@20	score@50
linear_mse	L1	0.004993	0.008343	0.013582	0.02558
linear_mse	L2	0.00512	0.008242	0.013722	0.025741
linear_l1	L1	0.002263	0.003748	0.0063	0.012188
linear_l1	L2	0.002233	0.003678	0.006278	0.012261
relu_mse	L1	0.027783	0.043683	0.066798	0.1113
relu_mse	L2	0.027767	0.043457	0.067046	**0.11222**
relu_l1	L1	0.023263	0.037283	0.058863	0.101125
relu_l1	L2	0.02234	0.036687	0.057701	0.100684

2. Same as (1) but clustered for each label, shows only score@20

model_name	**dist**	**label 0**	**label 1**	**label 2**	**label 3**	**label 4**	**label 5**	**label 6**	**label7**	**label 8**	**label 9**
linear_mse	L1	0.015963	0.047605	0.009114	0.009224	0.008268	0.004842	0.007325	0.014246	0.004256	0.009497395
linear_mse	L2	0.015955	0.04779	0.0095	0.009452	0.008388	0.005294	0.007452	0.014046	0.004298	0.009606657
linear_l1	L1	0.014207	0.009092	0.005623	0.005929	0.003817	0.00511	0.003819	0.005164	0.00423	0.005547151
linear_l1	L2	0.014174	0.009322	0.005438	0.005749	0.003706	0.005054	0.003768	0.005124	0.004495	0.005454698
relu_mse	L1	0.092124	0.098524	0.060255	0.051615	0.056804	0.058024	0.077585	0.065044	0.044095	0.059077156
relu_mse	L2	0.092529	0.095802	0.061522	0.053009	0.056967	0.059804	0.076943	0.065523	0.045197	0.058825013
relu_l1	L1	0.086907	**0.10907**	0.046165	0.03877	0.040842	0.05035	0.066492	**0.071381**	0.031781	0.038779627
relu_l1	L2	0.086839	0.102484	0.046215	0.038705	0.039601	0.052131	0.065149	0.068428	0.032106	0.038334174

Next we have looked at the most close samples labels, we have computed the ratio of identical labels among the most close samples.

3. Improvement metric: the average of **original_rate - encoded_rate** (for example -0.25 means after encoding 25% of the original samples are not close anymore)

	model_name	distance_type	score@5	score@10	score@20	score@50
0	linear_mse	L1	-0.57288	-0.561	-0.54615	-0.51974
1	linear_mse	L2	-0.58099	-0.56967	-0.55674	-0.53226
2	linear_l1	L1	-0.64869	-0.63726	-0.62226	-0.59509
3	linear_l1	L2	-0.65672	-0.64638	-0.63286	-0.60753
4	relu_mse	L1	-0.19797	-0.18754	-0.17229	**-0.14673**
5	relu_mse	L2	-0.20592	-0.19669	-0.18292	-0.15905
6	relu_l1	L1	-0.25102	-0.24066	-0.22585	-0.2001
7	relu_l1	L2	-0.25936	-0.24982	-0.23622	-0.21245

4. Similarity metric: the average of abs(**original_rate - encoded_rate**), lower values are indicate on better similarity after encoding.

	model_name	distance_type	score@5	score@10	score@20	score@50
0	linear_mse	L1	17165.18	16812.87	16357.78	15560.71
1	linear_mse	L2	17412.85	17070.36	16674.15	15936.71
2	linear_l1	L1	19448.89	19109.61	18647.23	17826.59
3	linear_l1	L2	19695.74	19380.14	18970.78	18202.64
4	relu_mse	L1	5985.549	5665.249	5193.369	**4417.016**
5	relu_mse	L2	6226.397	5932.247	5512.592	4787.709
6	relu_l1	L1	7564.998	7256.799	6800.114	6017.723
7	relu_l1	L2	7817.239	7527.116	7114.144	6392.121

Conclusions

In our experiments we have seen that autoencoder generally does not keep the context when reducing samples to lower dimension. However we have seen empirically that with specific samples we might keep some the context, for example with ‘1’ and ‘7’. It is possible that other architectures would lead to better results, moreover a custom loss function that keeps context could be used. Future work could use other datasets as CIFAR-10 or ImageNet.

Technical details

Optimization and distance computation were performed on Ubuntu 16.04 machine with 4GB of RAM and took 1 day for all models