Image Compression with Autoencoders

Huseyin Can Ercan

1 January, 2018

Abstract: Autoencoder usage for image compression, dimension reduction of datasets. Tensorflow api implementation for compression.

1 Introduction

Neural networks can provide diverse functionality for different applications. Classification or regression are possible with different architectures. Except classical problems neural networks also can solve generation or correction problems. One basic architecture is autoencoders for similar problems. Autoencoders are primitive network models which generates the input in output layer, simply repeats the input. This function can decrease the noise in data or complete a missing part. Autoencoders extracts the attributes of data and uses same attributes for reconstruction phase. Extracted attributes can be used for reconstruction on an identical network in another location. We can use extracted attributes to compress data.

1.1 Network Model and Training

Neural networks initialized randomly. Bias values and weights of the network selected randomly from small value ranges. Training algorithms updates the network weights and by this progress network reduces it's error. In training progress model can face serious problems. In some cases simply network architecture cannot model the data, or transfer functions can be unsuitable for data. These cases are hyper parameter relates problems. One major problem is the vanishing gradient for neural networks because this problem is related to network depth and transfer function. Deeper networks mean smaller gradients and slower learning or nor learning. Deep learning concept can solve this problem.

In our case we are going to use simple architectures for model. Because our data is MNIST handwritten numbers. MNIST data consists of 60,000 training examples, and 10,000 test examples, every example is a 28×28 2d array. Modelling the MNIST data with simple autoencoders is possible if we need to use wider and multiple channel images more complex models are required like convolutional neural networks.

Fundemnetal idea of the neural network training is minimizing the error. In our case error is the root mean square error of the input and output vectors. We need to train network to reconstruct input images. Forcing the minimize distance between input and output will provide required training. At this stage we face the gradient decent optimization problem. User selected hyper parameters and algorithm effects the learning process and model success. In out case we are using RMS prop. Selected algorithm and learning rate values effect the learning speed and minimum error point. For an autoencoder selecting an acceptable error limit will benefit the model. Higher iteration numbers or extreme error limits will harm the model because they will lead lower success on test set because of memorizing not learning.

Neural net training phase is a repetitive process and for every individual example training algorithm updates the internal parameters of the network.

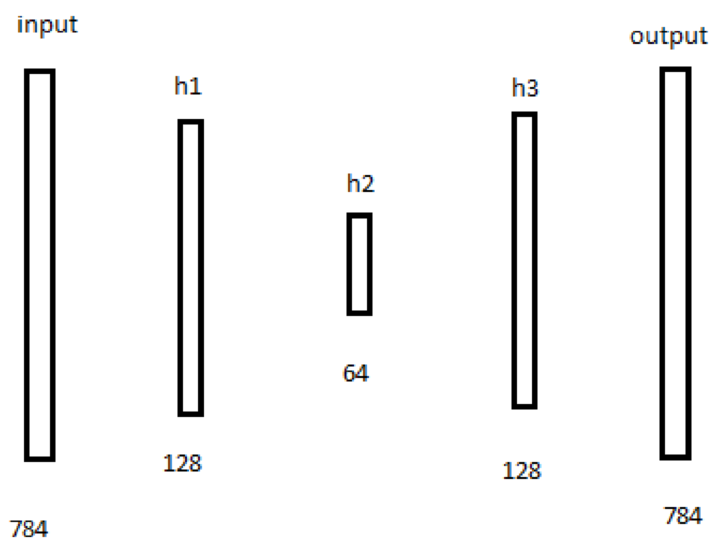

Illustration 1: Basic network model for MNIST dataset with 3 hidden layers

784×128	128×64	64×128	128×784

For every example training algorithm calculates the outputs then calculates the weight changes of the network. For our model we are using sigmoid transfer function. Transfer functions are important for extracting meaningful information from output later. Sigmoid function squeezes the layer values in an acceptable range. If we do not use a transfer function we just can get the sum values of the previous layers. 5 layer simple model requires 217088 individual weight update for a single example training. This process creates overload for limited number core hardware. CPU based computation takes too long for network training. Modern machine learning frameworks like TensorFlow can use GPU instead CPU.

1.2 Implementation Details

Tensorflow framework provides wide variety for customization for network models. User can add stacks of network layer to an input layer then can request any layer output from Tensorflow session. Our model uses sigmoid transfer function for every layer, also initialization process needs random weights, bias values.

For compression model we need the middle layer outputs. These values forms the compressed data. At this point quantization problem emerges. Middle layer output of the network is very valuable and precise for reconstruction performance. Small changes can lead influential output errors. To minimize this problem precision guarded with 19 digit and exponentiation number. Network sigmoid transfer function generates outputs between 0.0 and 1.0. Python numpy.ndarray (n dimensional array) provides required function to transfer data to text files. Binary file recovery can create problem between big endian and little endian machines.

In normal conditions a neural network output is generated uninterrupted. Layer outputs used as inputs in next layer as inputs in Tensorflow session. To implement a compression mechanism with specially Tensorflow framework implementation has to export middle layer values environment independent and with minimum quantization error. In this condition main purpose is not the reach minimum disk space, it is reaching the minimum dimension vector for the compressed data.

After reaching minimum dimension in compressed vector we can use other data compression methods to minimize actual physical space. In previous model we used 784 dimension vector as input and reached a 64 dimension vector after compression. Compression ratio is 0,081, compression performance is 91,8. Same performance is applied to all inputs. But loss value may vary between samples.

Recovery of the transferred vectors depends on the identical networks at the end of the communication lines. Encoding operation needs the first half of the network, decoding operation needs the second half of the network. For encoding operation receiving side needs at least the second half of the neural network. At this stage also implementation has to able to transfer neural network models. To answer this requirement used theTensorflow session save procedure. But storing the whole model is a costly necessity.

Model size: 2623 KB	Original data size: 22 KB	Compressed data size: 2KB

Table 1: Cost sizes of model(128 64 128), one original data sample and compressed data vector

Neural network model should transfer 131 samples to compensate model transfer cost for Table1 example. Network file sizes vary via network architecture.

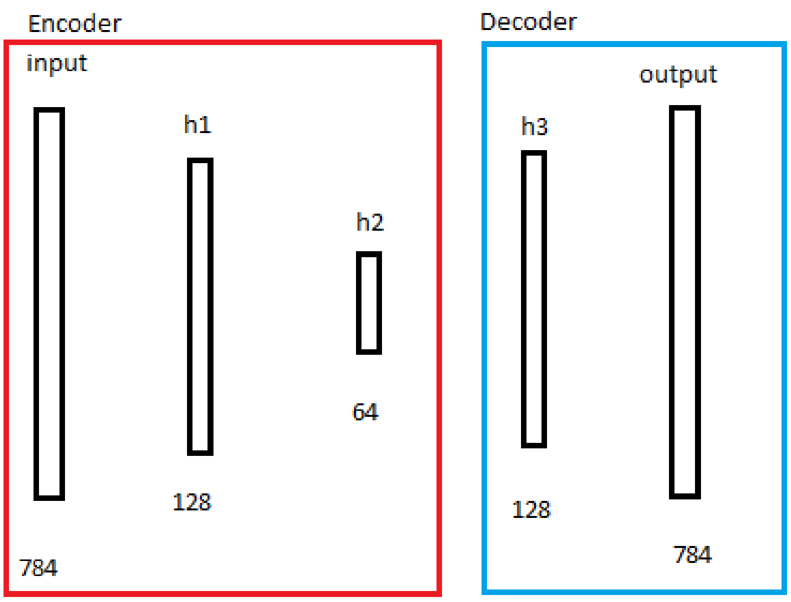

Illustration 2: Neural network encoder and decoder parts

Model\Iteration	1	50	100	200	Test
128-64-128	0.20	0.04	0.028	0.019	0.011
128-50-128	0.16	0.037	0.023	0.017	0.012
128-36-128	0.18	0.029	0.023	0.016	0.013
128-22-128	0.12	0.042	0.034	0.023	0.014

Table 2: Root mean square error values of the test set on different model architectures

Reducing the compression layer width may increase compression performance but increasing loss unavoidable.

1.2 Summary

For MNIST data set autoencoders can provide significant dimension reduction. We can reduce a 784 dimension vector to 64 dimension with 0.011 RMSE value. Existing data compression methods provides efficienct compression result for images or video data. These methods are well developed and tested. They are complex and result of an accumulated research time. In the other hand neural networks are versatile and with adequate data we can train them without knowing the complex model of the data. Thus, autoencoders and similar dimension reducing networks are useful for vector compression. Manipulation methods of the vectors may effect the real physical memory need but vector compression ratio remains unchanged with same neural network model.

1.3 Future Research

Convolutional networks for multi channel data may be more useful than one channel compressing neural nets. Dimensionally reduced data is not an image anymore, need to merge a lose less precision value transfer method for more efficient compressed vector transfer.

References

1. Lossy Image Compression With Compressive Autoencoders Lucas Theis, Wenzhe Shi, Andrew Cunningham& Ferenc Huszar´ ICLR 2017
2. Tensorflow Api Documents https://www.tensorflow.org/api_docs