WASSERSTEIN AUTO-ENCODERS

A PREPRINT

Ivan Barabanau
`Ivan.Barabanau@skoltech.ru`

Albert Matveev
`Albert.Matveev@skoltech.ru`

Anton Zhevnerchuk
`Anton.Zhevnerchuk@skoltech.ru`

October 26, 2018

ABSTRACT

We based our project on the paper [1] by Tolstikhin et al. We first define general ideas and concepts from [1] to set up the model we implemented. Then we describe some details of our implementation [2]. Finally, we make an overview of extensions of [1] we tried (and partially succeeded) to make.

1 Model overview

We have dataset $X \subset \mathcal{X}$ distributed according to some unknown distribution P_X. Wasserstein Auto-Encoders (WAEs) solve two tasks: to sample from P_X and to encode points from $\mathcal{X}$ into some low-dimensional representation, which can be efficiently decoded back. To achieve this goals a latent space $\mathcal{Z}$ together with some prior distribution P_Z are introduced. We learn two neural networks (NNs) E (encoder) and G (decoder). Encoder maps an element from $\mathcal{X}$ to an element from $\mathcal{Z}$, decoder implements an inverse mapping. To sample X from P_X we sample Z from P_Z and then apply decoder.

Let us describe model in more details. Sampling process described above implements sampling from $P_G = G(P_Z)$ – prior in $\mathcal{Z}$ transformed by decoder G. We use this distribution to simulate unknown P_X. So, our objective is to minimize some distance between P_X and P_G. We optimize so-called *Wasserstein distance*.

Definition 1.1 *Let $c : X \times X \to \mathbb{R}_+$ be some cost function. Wasserstein distance between $P_X(X)$ and $P_G(X)$ is given by:*

$$W_c(P_X, P_G) = \mathbb{E}_{X_1 \sim P_X(X_1), X_2 \sim P_G(X_2)}[c(X_1, X_2)].$$

Optimizing such an objective does not seem to be an easy task (we can note that there is no dependence on encoder E at all). It occurs that this objective has another form:

Theorem 1.1

$$\inf_{P_X, P_G} W_c(P_X, P_G) = \inf_{Q : Q_Z = P_Z} \mathbb{E}_{P_X} \mathbb{E}_{Q(Z|X)}[c(X, G(Z))],$$

where Q is a distribution over pairs (X, Z) and Q_Z is its marginal distribution on Z.

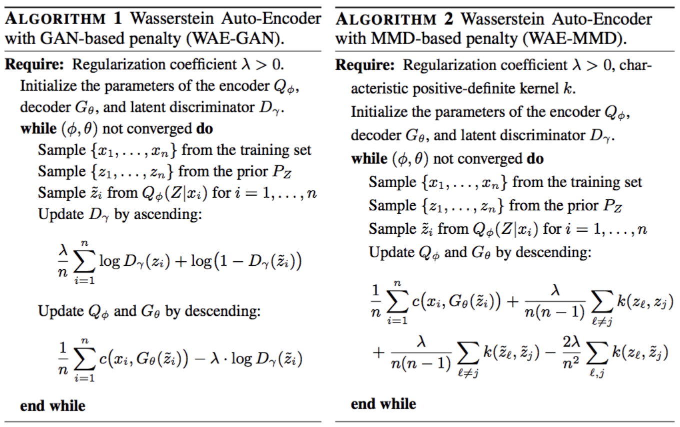
This transformation of objective allows us to conduct optimization over encoder weights in more proper way. The problem here is that such an equality holds only than marginal distribution Q_Z is equal to the prior P_Z. In order to satisfy this condition, regularization term $D_Z(Q_Z, P_Z)$, where Q_Z is a distribution induced on $\mathcal{Z}$ by P_Z and encoder E and D_Z is some divergence. We end up with the following objective:

$$\mathbb{E}_{P_X} \mathbb{E}_{Q(Z|X)}[c(X, G(Z))] + \lambda \mathcal{D}_Z(Q_Z, P_Z) \to \min.$$

It is still unclear how to optimize (and even evaluate) this function. The first term can be efficiently estimated using Monte-Carlo methods. In [1] two approaches to estimating the second term are proposed. The first one is to simulate divergence by another neural network (discriminator). This approach is called $WAE - GAN$. The second way is to use maximum mean discrepancy.

Definition 1.2 *Given some positive-definite kernel $k : Z \times Z \to \mathbb{R}_+$ maximum mean discrepancy between two distribution P_Z, Q_Z is given by*

$$MMD_k(P_Z, Q_Z) = \|\mathbb{E}_{z \sim P_Z}[k(\cdot, z)] - \mathbb{E}_{z \sim Q_Z}[k(\cdot, z)]\|,$$



ALGORITHM 1 Wasserstein Auto-Encoder with GAN-based penalty (WAE-GAN).

```
Require: Regularization coefficient $\lambda > 0$.
  Initialize the parameters of the encoder $Q_\phi$,
  decoder $G_\theta$, and latent discriminator $D_\gamma$.
  while $(\phi, \theta)$ not converged do
    Sample $\{x_1, \dots, x_n\}$ from the training set
    Sample $\{z_1, \dots, z_n\}$ from the prior $P_Z$
    Sample $\tilde{z}_i$ from $Q_\phi(Z|x_i)$ for $i = 1, \dots, n$
    Update $D_\gamma$ by ascending:
        $\frac{\lambda}{n} \sum_{i=1}^{n} \log D_\gamma(z_i) + \log\big(1 - D_\gamma(\tilde{z}_i)\big)$
    Update $Q_\phi$ and $G_\theta$ by descending:
        $\frac{1}{n} \sum_{i=1}^{n} c\big(x_i, G_\theta(\tilde{z}_i)\big) - \lambda \cdot \log D_\gamma(\tilde{z}_i)$
  end while
```

ALGORITHM 2 Wasserstein Auto-Encoder with MMD-based penalty (WAE-MMD).

```
Require: Regularization coefficient $\lambda > 0$, characteristic positive-definite kernel $k$.
  Initialize the parameters of the encoder $Q_\phi$,
  decoder $G_\theta$, and latent discriminator $D_\gamma$.
  while $(\phi, \theta)$ not converged do
    Sample $\{x_1, \dots, x_n\}$ from the training set
    Sample $\{z_1, \dots, z_n\}$ from the prior $P_Z$
    Sample $\tilde{z}_i$ from $Q_\phi(Z|x_i)$ for $i = 1, \dots, n$
    Update $Q_\phi$ and $G_\theta$ by descending:
        $\frac{1}{n} \sum_{i=1}^{n} c\big(x_i, G_\theta(\tilde{z}_i)\big) + \frac{\lambda}{n(n-1)} \sum_{\ell \neq j} k(z_\ell, z_j)$
        $+ \frac{\lambda}{n(n-1)} \sum_{\ell \neq j} k(\tilde{z}_\ell, \tilde{z}_j) - \frac{2\lambda}{n^2} \sum_{\ell, j} k(z_\ell, \tilde{z}_j)$
  end while
```

Figure 1: Pseudo-code

where norm is taken in the space of real-valued functions on Z.

Such a distance between distributions is nice because it is proven to have pretty simple unbiased estimator. In order to make this report self-containing, we provide pseudo-code for this two approaches from [1] at Figure 1.

2 Implementation Details

During the project we implemented the models setups, described in the original paper, as well as estimated the possibility of WGAN application in the related 2D and 3D tasks. To provide maximal flexibility during the training process, we developed a base class-trainer, which accepts different types of divergences, kernels, optimizers and networks architectures as well. The primal aim was to reproduce the author's results on the provided MNIST and Celeba datssets. According to out estimations, we did not find a significant divergency in reconstruction and sampling performance between the GAN and MMD based WAE implementations, as claimed in a paper, despite simultaneously varying networks architectures and parameters values. Nonetheless, we provide with the best obtained examples of WAE performance. All the experiments were conducted in a Pytorch framework.

3 Experiments and Extensions

3.1 Reproducing original paper

In accordance with authors' claims, subnetworks architectures were implemented in a VGG-style, i.e the stack of the Convolutional, Batch Normalization and Activation layers. For the MNIST dataset the amount of hidden latent units was set to 8, whilst for more diverse Celeba dataset - to 64. Apart from varying neural networks architectures, we also experimented with kernel functions. For instance, RBF and inverse multiquadric kernel were used, without significant outperformance in both cases. The best results of image sampling, from out point of view, are presented at Figure 2.

3.2 Completing MNIST images

Encoder-decoder architecture can be used not only to reconstruct the original data from the latent representation. Probably the most famous example of using such an approach for other tasks is so-called Conditional Variational Auto-Encoders (CVAEs), which can be used for reconstructing the lower part of image from its upper part. An overview on CVAEs can be found in [3].

The initial objective of WAE can be easily reformulated in this manner by conditioning all the output distributions on the input data: we minimize Wasserstein distance between $P_Y(Y|X)$ and $P_G(Y|X)$. We did not validate that all the steps from the initial WAE objective to the final can be easily modified for this setting, now it is still a topic for further research. However, straightforward adaptation of the final WAE objective to the setting with different input and output leads to the pretty nice results on MNIST demonstrated on Figure 3.

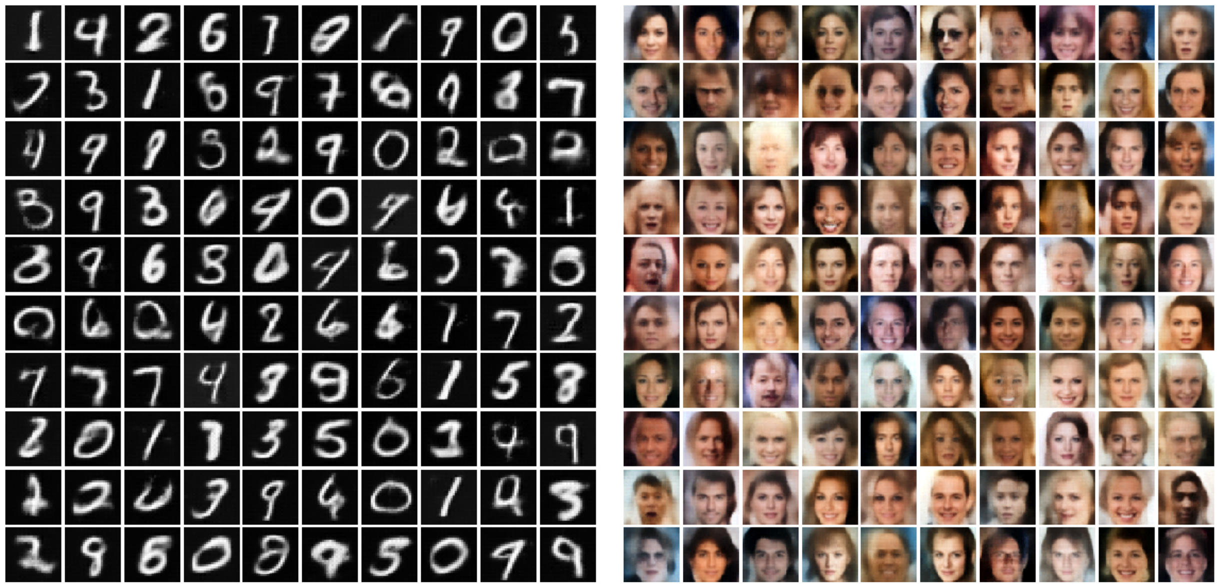


Figure 2: Generated samples from MNIST dataset and Celeba dataset

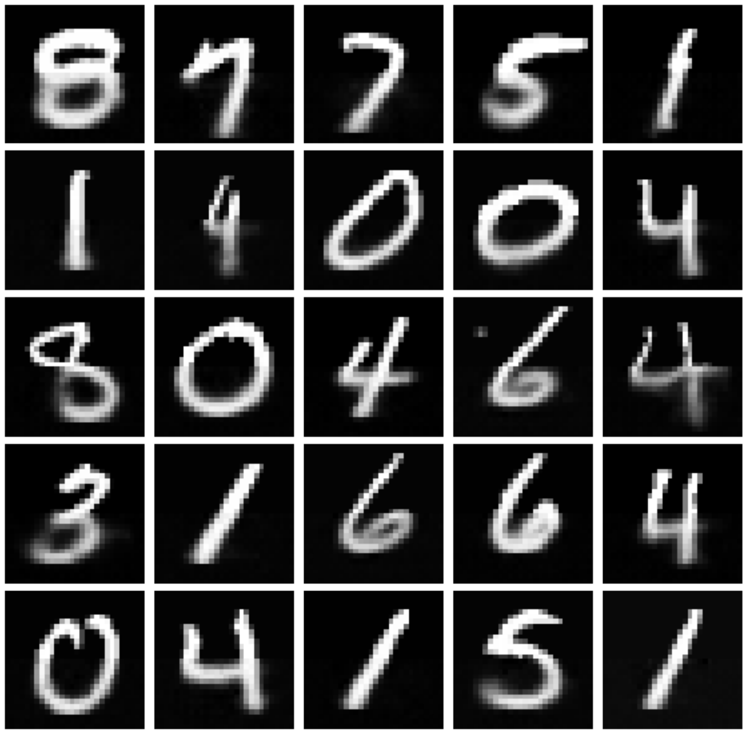


Figure 3: Lower half of images are generated by our algorithm from the upper half

3.3 3D datasets

Another experiment we conducted was adapting WAE architecture for point clouds. This is a complicated problem, and a lot of research papers today are devoted to generative models for 3D datasets. Main desiderata to such approaches would be the following: invariance of the output with respect to geometric transformations (scaling, translations, rotations), and ability to process unordered sets of points. The most natural way to construct autoencoding architecture for 3D point sets is to use one of existing neural networks specifically designed for this input.

Among others, one of the earliest approaches and one of the most popular ones is PointNet [4]. This architecture allows learning on point clouds, extracting hierarchical representations for points, and forming multilevel abstractions. This neural network was specifically designed to fulfill the requirements for such an approach. Basic building blocks for PointNet are multilayer perceptrons and max pooling layers, allowing easy to implement structure. We adapted segmentation architecture such that it would generate euclidean coordinates of the same amount of points as input data.

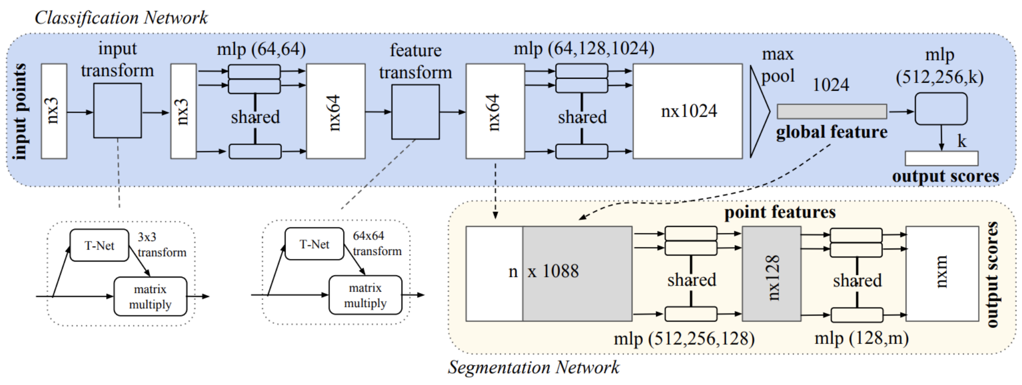


Figure 2. **PointNet Architecture.** The classification network takes n points as input, applies input and feature transformations, and then aggregates point features by max pooling. The output is classification scores for k classes. The segmentation network is an extension to the classification net. It concatenates global and local features and outputs per point scores. "mlp" stands for multi-layer perceptron, numbers in bracket are layer sizes. Batchnorm is used for all layers with ReLU. Dropout layers are used for the last mlp in classification net.

Figure 4: PointNet architecture

As a training data, we chose ModelNet dataset. It comprises relatively simple 3D models of 40 classes, including planes, furniture, laptops, etc. We trained PointNet-based Wasserstein Autoencoder 500 epochs with MMD cost. The results are disappointing; our architecture was not able to recover the initial structure of the point cloud, generating blobs. The reason for that is hidden in the architecture. Unfortunately, segmentation and autoencoding problems are essentially different, and what was proposed to recover pointwise labels is not suitable for generating. Since we need to obtain a vector as hidden representation of point cloud so that it could be fed into regularizer, we loose all point cloud structure, and the information cannot be restored efficiently. Making this approach work would require either conceptually different regularizers, which are not applicable in WAE framework, or another encoding strategy, which we are not aware of.

References

[1] Ilya Tolstikhin, Olivier Bousquet, Sylvain Gelly, and Bernhard Scholkopf. Wasserstein auto-encoders. *https://openreview.net/pdf?id=HkL7n1-0b*, 2018.

[2] Ivan Barabanau, Albert Matveev, and Anton Zhevnerchuk. Wasserstein auto-encoders (pytorch). *https://github.com/zhevnerchuk/WAE*, 2018.

[3] Carl Doersch. Tutorial on variational autoencoders. *https://arxiv.org/abs/1606.05908*, 2016.

[4] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 652–660, 2017.

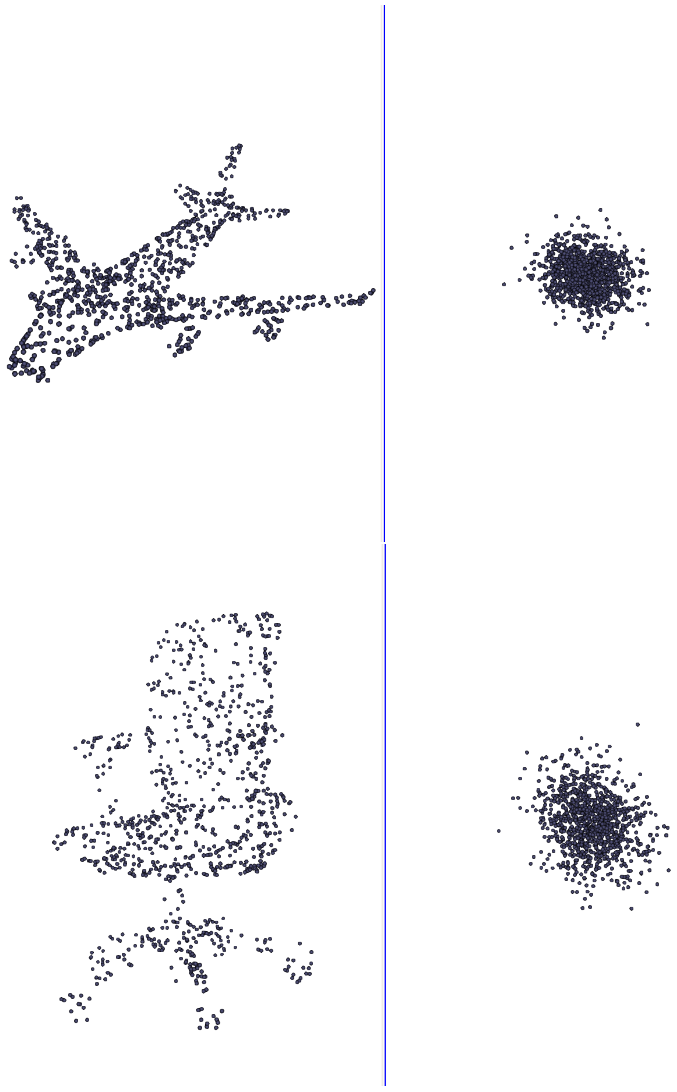


Figure 5: Left: ground truth point cloud, right: the output of WAE