REPRODUCTION OF ADVERSARIAL FEATURE LEARNING

Michal Stypulkowski
Faculty of Mathematics and Computer Science
University of Wroclaw
Wroclaw, Poland
michal.stypulkowski96@gmail.com

ABSTRACT

Bidirectional Generative Adversarial Network (BiGAN) is extended version of Generative Adversarial Network (GAN). BiGAN learns not only to map from simple latent distribution to complex data distribution as GANs does, but it is able to learn inverse mapping as well. Learned features representation can be used for supervised tasks. In this work I will present results of *Adversarial Feature Learning* ICRL 2017 paper reproduction.

1 INTRODUCTION

Generative Adversarial Networks (GANs) are state-of-the-art methods used for generating new samples from complex data distributions. They were introduced by Goodfellow et al. (2014). From that time many researchers have been developing new models to improve results in data generation.

One of the well-known GANs is BiGAN (Donahue et al., 2017). Main difference between these two models is that BiGAN in addition to standard GAN learns inverse mapping by adding encoder to its architecture, which extract latent features in space of dimension lower than input data.

2 BIGAN

BiGAN has three components: generator G, discriminator D and encoder E.

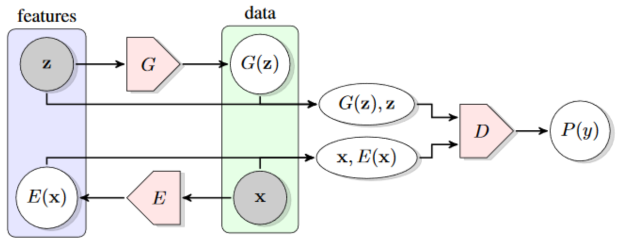

Figure 1: The structure of BiGAN.

Generator (G) takes random noise $\mathbf{z}$ and generate new samples $G(\mathbf{z})$. We want these samples to come from the same distribution as training data $\boldsymbol{x}$.

Encoder (E) maps data $\boldsymbol{x}$ into lower-dimensional latent feature space $E(\boldsymbol{x})$. The goal of E is to learn to invert the generator G.

Discriminator (D) tries to distinguish generated samples $G(\mathbf{z})$ from the real ones. Generator and Discriminator are competing with each other. Main difficulty in training GANs is to remain good trade-off between these to components. We don't want D to be too good at its task, because then it would be impossible for G to learn efficient data generation (it would not know how to mimic the real distribution). In the other hand, D cannot be too weak. G would have too much freedom in generating new samples. In BiGAN, D compares not only $\boldsymbol{x}$ and $G(\mathbf{z})$ but $\mathbf{z}$ and $E(\boldsymbol{x})$ as well.

Training objective of BiGAN is to find E, G, D that satisfy

$$\min_{E,G} \max_{D} V(D, E, G) \tag{1}$$

where $V(D,E,G) = \mathbb{E}_{\boldsymbol{x}} \sim p_{\boldsymbol{x}} \big[\log D(\boldsymbol{x}, E(\boldsymbol{x}))\big] + \mathbb{E}_{\mathbf{z}} \sim p_{\mathbf{z}} \big[\log(1 - D(G(\mathbf{z}), \mathbf{z}))\big]$.

3 Implementation details

Reconstruction was written in PyTorch and run on single Nvidia GeForce GTX 1060 TI. Code is available on `https://github.com/MStypulkowski/BiGAN`.

3.1 Architecture

To implement E, G and D, we use three neural networks. Each of them contains two linear layers with 1024 neurons. First one is followed by non-linearity and second one by batch normalization and non-linearity. Final layer is linear followed by activation function. In E and D non-linear function is LeakyReLU and in G we use ReLU. Final activation functions in G and D are both Sigmoid, while there is none in E.

3.2 Loss function

In practice instead of strictly satisfying condition (1) we define two loss functions, which we wish to minimize:

$$L_D = -\sum_{i=1}^{m} \log D(\boldsymbol{x}, E(\boldsymbol{x})) + \log(1 - D(G(\mathbf{z}), \mathbf{z}))$$

$$L_{E,G} = -\sum_{i=1}^{m} \log D(G(\mathbf{z}), \mathbf{z}) + \log(1 - D(\boldsymbol{x}, E(\boldsymbol{x})))$$

where m is size of single mini-batch.

3.3 Optimization

As the optimizer Adam with betas equal 0.5 and 0.999 was used. Donahue et al. (2017) suggests to use learning rate 2×10^{-4} for the first 200 epochs and decay it exponentially through another 200 epochs to 2×10^{-6} and to use ℓ_2 weight decay with parameter 2.5×10^{-5}. However it was found that it is more efficient not to use learning rate decay and to use weight decay with parameter 10^{-5}. Quantitative results (4) were up to 12% lower with exact author's suggestions. Multiplicative weights are initialized from normal distribution with mean 0 and standard deviation 0.02 while biases with zeros. Random vector $\mathbf{z}$ comes from 50-dimensional uniform distribution $U([-1,1])^{50}$.

4 Results

As the dataset for experiments we use MNIST. Figure 2 shows examples of new generated images $G(\mathbf{z})$ and reconstructions of real ones $G(E(\boldsymbol{x}))$ obtained on trained BiGAN model. Most of the generated digits are undistinguished from the real ones. Reconstructions are correct for most samples. In both cases we get sharp shapes with some blurriness.

To check performance of encoder and its ability of mapping images into latent space, we perform t-SNE on test set in order to visualize the structure of $E(\boldsymbol{x})$. Results are shown in figure 3. We may notice that classes seem to be clustered but not separated from each other in all cases. Results differ after each retraining of BiGAN and running t-SNE.

As the last test we perform 1NN algorithm on test set using labeled learned latent space of training set. Best quantitative result observed was roughly 92% using architecture mentioned in 3.3.

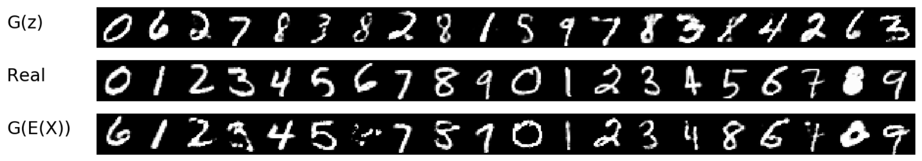

Figure 2: Generated random samples (top row), real images (middle row) and corresponding reconstructed images (bottom row).

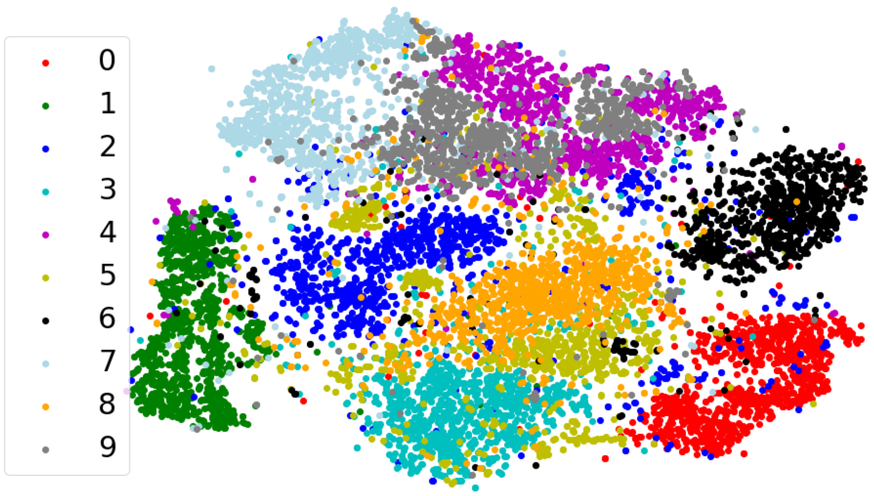

Figure 3: Two dimensional t-SNE mapping of latent space of the MNIST test set encoded by BiGAN.

5 Conditional BiGAN

Let's now consider extended version of BiGAN where we have access to labels of data. One of the generative supervised models is Conditional GAN (CGAN) (Mirza & Osindero, 2014). Here we try to combine these two models in supervised generative task with ability to learn latent distribution of data. For the purpose of this work let's denote this mixed model by CBiGAN.

Assume we know labels $\boldsymbol{c}$ of each image. We concatenate $\boldsymbol{c}$ with random vector $\mathbf{z}$ and use it as an input to the generator G. Also, instead of passing pairs $(\boldsymbol{x}, E(\boldsymbol{x}))$ and $(G(\mathbf{z}), \mathbf{z})$, we pass triples $(\boldsymbol{x}, E(\boldsymbol{x}), \boldsymbol{c})$ and $(G(\mathbf{z}, \boldsymbol{c}), \mathbf{z}, \boldsymbol{c})$. This architecture helps model to learn to generate random samples from given class. It is different from standard BiGAN where we are only able to generate random samples from all classes. Now we have control over which digit we want to see.

From now we will distinguish two types of CBiGANs: simple and extended. Simple has the same architecture as BiGAN (with additions described above), while in extended version we add one linear layer with non-linear activation function for each $\mathbf{z}$, $\boldsymbol{c}$ and $\boldsymbol{x}$ before concatenating them as inputs to core layers of generator and discriminator.

Figure 4 shows results of generating and reconstructing samples from MNIST dataset using both CBiGANs. Images reconstructed by simple CBiGAN seem to be less blurry then in BiGAN. We may also notice that less samples are being reconstructed as objects from different classes (after manual exploration of some outputs the number is 0). Generated digits are also looking well and, more importantly, they are consistent with desired classes. In extended CBiGAN reconstructed and generated samples have same properties as in simple CBiGAN, but their quality is worse. Images are more blurry.

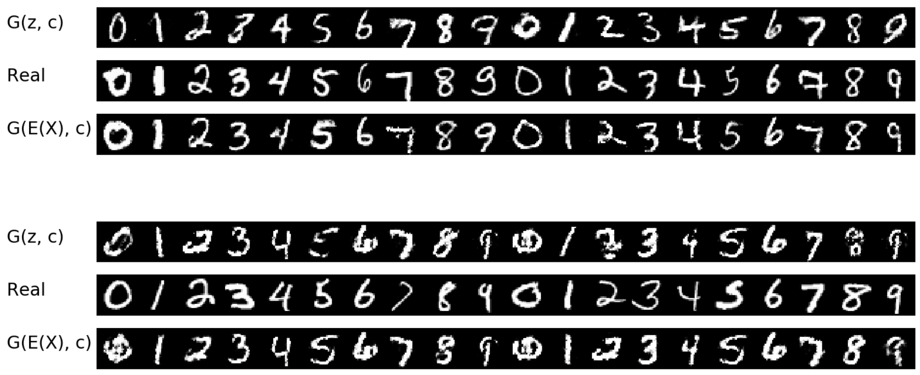

Figure 4: Results for simple CBiGAN (top part) and extended CBiGAN (bottom part). Real images (middle rows), corresponding reconstructed images (bottom rows) and generated random samples from the same classes as corresponding real images (top rows).

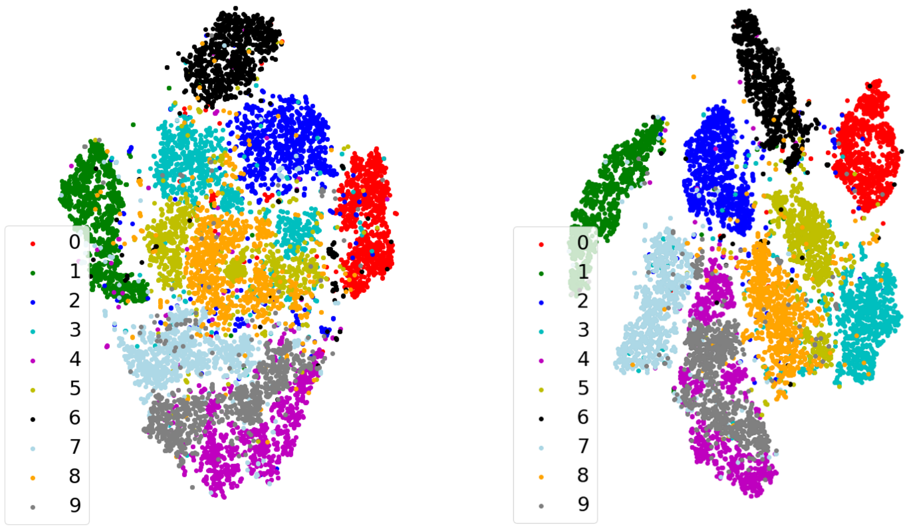

Figure 5: Two dimensional t-SNE mapping of latent space of the MNIST test set encoded by simple CBiGAN (left side) and extended CBiGAN (right side).

As in 4 t-SNE visualization (figure 5) and 1NN algorithm were run. Latent space clusters found in simple CBiGAN look similar to BiGAN's. In the extended model classes are separated much better. The only exceptions are $4s$ and $9s$. Samples from these two classes are mixed together. It can be explained by similar structure of handwritten 4 and 9 digits.

Result in 1NN test got worsen to 91% in simple CBiGAN, while in extended version we get 95% accuracy. We get control over classes of generated images and both visually better and more accurate reconstructions, losing less than 1% in classification task using simple CBiGAN. In extended CBiGAN we have more accuracy in classification losing some quality of new and reconstructed samples. Results of extended version lead to conclusion that encoder learned to map images to latent space very well, while generator seems to underfit. Additional changes in its architecture and choosing appropriate hyperparameters might solve this problem.

References

Jeff Donahue, Phillip Krahenbuhl, and Trevor Darrell. Adversarial feature learning. *In ICLR*, 2017.

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *In NIPS*, 2014.

Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. 2014.