Training Generative Adversarial Networks with Binary Neurons by End-to-end Backpropagation

Hao-Wen Dong, Yi-Hsuan Yang
Research Center for IT Innovation, Academia Sinica, Taipei, Taiwan
{salu133445,yang}@citi.sinica.edu.tw

Abstract

We propose the BinaryGAN, a novel generative adversarial network (GAN) that uses binary neurons at the output layer of the generator. We employ the sigmoid-adjusted straight-through estimators to estimate the gradients for the binary neurons and train the whole network by end-to-end backpropogation. The proposed model is able to directly generate binary-valued predictions at test time. We implement such a model to generate binarized MNIST digits and experimentally compare the performance for different types of binary neurons, GAN objectives and network architectures. Although the results are still preliminary, we show that it is possible to train a GAN that has binary neurons and that the use of gradient estimators can be a promising direction for modeling discrete distributions with GANs. For reproducibility, the source code is available at https://github.com/salu133445/binarygan.

1 Introduction

Generative adversarial networks (GAN) [7] have enjoyed great success in modeling continuous distributions. However, applying GANs to discrete data have been shown nontrivial in that it is difficult to optimize the model distribution toward the target data distribution in a high-dimensional discrete space. Approaches adopted in the literature to utilize GANs on discrete data can roughly be divided into three directions.

One direction is to replace the target discrete outputs with continuous relaxations. Kusner *et al.* [13] proposed to use the continuous Gumbel-Softmax distribution to approximate a categorical distribution and generate sequences of discrete elements using one-hot encoding. Using the Wasserstein GANs [2], Gulrajani *et al.* [8] and Subramanian *et al.* [16] have developed in parallel models that can handle discrete data by simply passing the continuous, probabilistic outputs (i.e., softmax relaxation) of the generator to the discriminator.

The second direction is to view the generator as an agent in reinforcement learning (RL) and introduce RL-based training strategies. Yu *et al.* [19] considered the generator as a stochastic parametrized policy and trained the generator via policy gradient and Monte Carlo search. Hjelm *et al.* [10] used estimated difference measure from the discriminator to compute importance weights for generated samples, which provides a policy gradient for training the generator. More examples can be seen in natural language processing (NLP), including dialogue generation [15] and machine translation [18].

The third direction is to introduce gradient estimators to estimate the gradients for the nondifferentiable discretization operations for the generator. Since the discretization operations are used in the forward pass, the generator is able to provide discrete predictions to the discriminator during the training and directly generate discrete predictions at test time without any further post-processing. Moreover, it can support conditional computation graphs [3, 4] that allow the system to make discrete decisions for more advanced designs. However, to the best of our knowledge and as pointed out by [10], none of these has yet been shown to work with GANs.

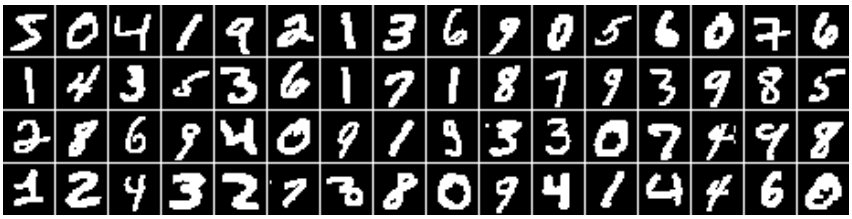


Figure 1: Sample binarized MNIST digits seen in our training data.

In our previous work on generating binary-valued music pianorolls [6], we proposed to append to the generator a refiner network that learns to binarize the generator's outputs to binary ones. However, the whole network was trained in a two-stage setting. It remains unclear whether and how we can train a GAN that has binary neurons in an end-to-end manner. We intend to study this issue in this paper and consider here the generation of binarized MNIST handwritten digits (see Figure 1) as a case study, assuming that people are more familiar with the MNIST digits than the pianorolls.

In this paper, we employ either stochastic or deterministic binary neurons at the output layer of the generator. In order to train the whole network by end-to-end backpropagation, we use the sigmoid-adjusted straight-through estimators [1, 3, 9] to estimate the gradients for the binary neurons. We experimentally compare the performance of the proposed model, which we dub BinaryGAN, using different types of binary neurons, GAN objectives and network architectures.

2 Backgrounds

2.1 Generative Adversarial Networks

A generative adversarial network (GAN) [7] is a generative model that can learn a data distribution in an unsupervised manner. It contains two components: a *generator* G and a *discriminator* D. The generator takes as input a random vector $\mathbf{z}$ sampled from a prior distribution $p_{\mathbf{z}}$ and outputs a fake sample. The discriminator take as input either a real sample drawn from the data distribution p_d or a fake sample generated by and outputs a scalar representing the genuineness of that sample.

The training is formulated as a two-player game: the discriminator aims to tell the fake data from the real ones, while the generator aims to fool the discriminator. Note that in this paper we refer to GAN as its non-saturating version, which can provide stronger gradients in the early stage of the training as suggested by [7]. The objectives for the generator and the discriminator are

$$\textbf{(GAN)} \quad \min_G -\mathbf{E}_{\mathbf{z}\sim p_{\mathbf{z}}}[\log D(G(\mathbf{z})), \tag{1}$$

$$\textbf{(GAN)} \quad \max_D \mathbf{E}_{\mathbf{x}\sim p_d}[\log D(\mathbf{x})] + \mathbf{E}_{\mathbf{z}\sim p_{\mathbf{z}}}[\log(1 - D(G(\mathbf{z})))], \tag{2}$$

Another form called WGAN [2] was later proposed with the intuition to estimate the Wasserstein distance between the real and the model distributions by a deep neural network and use it as a critic for the generator. It can be formulated as

$$\textbf{(WGAN)} \quad \min_G \max_D \mathbf{E}_{\mathbf{x}\sim p_d}[D(\mathbf{x})] - \mathbf{E}_{\mathbf{z}\sim p_{\mathbf{z}}}[D(G(\mathbf{z}))]\,, \tag{3}$$

where the discriminator D must be Lipschitz continuous.

In [2], the Lipschitz constraint on the discriminator is imposed by *weight clipping*. However, this can lead to undesired behaviors as discussed in [8]. A *gradient penalty* (GP) term that punishes the discriminator when it violates the Lipschitz constraint is then proposed in [8]. The objectives become

$$\textbf{(WGAN-GP)} \quad \min_G \mathbf{E}_{\mathbf{x}\sim p_d}[D(\mathbf{x})] - \mathbf{E}_{\mathbf{z}\sim p_{\mathbf{z}}}[D(G(\mathbf{z}))]\,, \tag{4}$$

$$\textbf{(WGAN-GP)} \quad \max_D \mathbf{E}_{\mathbf{x}\sim p_d}[D(\mathbf{x})] - \mathbf{E}_{\mathbf{z}\sim p_{\mathbf{z}}}[D(G(\mathbf{z}))] - \mathbf{E}_{\hat{\mathbf{x}}\sim p_{\hat{\mathbf{x}}}}[(\nabla_{\hat{\mathbf{x}}}\|\hat{\mathbf{x}}\| - 1)^2]\,, \tag{5}$$

where $p_{\hat{\mathbf{x}}}$ is defined sampling uniformly along straight lines between pairs of points sampled from p_d and p_g, the model distribution.

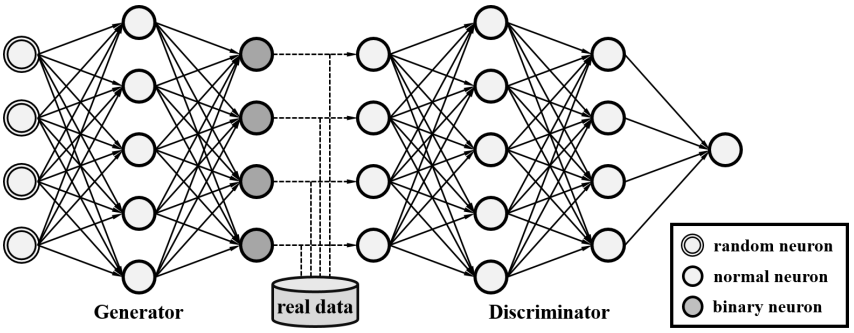


Figure 2: System diagram of the proposed model implemented by MLPs. Note that binary neurons are only used at the output layer of the generator.

2.2 Deterministic and Stochastic Binary Neurons

Binary neurons are neurons that output binary-valued predictions. In this work, we consider two types of them. A *deterministic binary neuron* (DBN) acts like a neuron with the hard thresholding function as its activation function. The output of a DBN for a real-valued input x is defined as

$$DBN(x) = u(\sigma(x) - 0.5)\,, \tag{6}$$

where $u(\cdot)$ denotes the unit step function and $\sigma(\cdot)$ is the sigmoid function.

A *stochastic binary neuron* (SBN) acts like a neuron using Bernoulli sampling as its activation function and binarizes a real-valued input x according to probability, defined as

$$SBN(x) = u(\sigma(x) - v),\ v \sim U[0, 1]\,, \tag{7}$$

where $U[0, 1]$ denotes a uniform distribution. Note that in this work we refer to $\sigma(x)$ in (6) and (7) as the *preactivated outputs*.

2.3 Sigmoid-adjusted Straight-through Estimators

Backpropgating through binary neurons, however, is intractable. The reason is that, for a DBN, the threshold function is nondifferentiable and that, for an SBN, it requires the computation of the expected gradient averaged on all possible combinations of values taken by the binary neurons, where the number of such combinations is exponential to the total number of binary neurons.

The *straight-through estimator* was first proposed as a regularizer in [9]. It simply ignores the gradient of a binary neuron and treats it as an identity function in the backward pass. A variant called *sigmoid-adjusted straight-through estimator* [3] replaces the gradient of a binary neuron with the gradient of the sigmoid function instead. The latter is found to achieve better performance in a classification task presented in [1]. Hence, when training networks with binary neurons, we resort to the sigmoid-adjusted straight-through estimators to provide the gradients for the binary neurons.

3 BinaryGAN

We propose the BinaryGAN, a model that can generate binary-valued predictions without any further post-processing and can be trained by end-to-end backpropagation. The proposed model consists of a *generator* G and a *discriminator* D. The generator takes as input a random vector $\mathbf{z}$ drawn from a prior distribution $p_{\mathbf{z}}$ and generate a fake sample $G(\mathbf{z})$. The discriminator takes as input either a real sample drawn from the data distribution or a fake sample generated by the generator and outputs a scalar indicating the genuineness of that sample.

In order to handle binary data, we propose to use binary neurons, either deterministic or stochastic ones, at the *output layer* (i.e., the final layer) of the generator. Hence, the model space (i.e., the output space of the generator) is 2^N, where N is the number of visible binary neurons at the output layer. We employ the sigmoid-adjusted straight-through estimators to estimate the gradients for binary neurons and train the whole network by end-to-end backpropagation. Figure 2 shows the system diagram for the proposed model implemented by multilayer perceptrons (MLPs).

Layer	Number of nodes	Activation
dense	1024	ReLU
dense	784	sigmoid

Table 1: Network architecture of the generator for the MLP model.

Layer	Number of nodes	Activation
dense	512	LeakyReLU
dense	256	LeakyReLU
dense	1	sigmoid

Table 2: Network architecture of the discriminator for the MLP model.

4 Experiments and Results

4.1 Training Data—Binarized MNIST Database

In this work, we use the binarized version of the MNIST handwritten digit database [14]. Specifically, we convert pixels with nonzero intensities to ones and pixels with zero intensities to zeros. Figure 1 shows some sample binarized MNIST digits seen in our training data.

4.2 Implementation Details

- Both the generator and the discriminator are implemented as multilayer perceptrons (MLPs) (see Tables 1 and 2 for the network architectures). Note that other network architectures will be compared in Section 4.6.
- The batch size for all the experiments is set to 64.
- As suggested by [8], we apply batch normalization [11] only to the generator and omit batch normalization in the discriminator.
- We train the proposed model with the WGAN-GP objective. Note that other GAN objectives will be compared in Section 4.5.
- We use the Adam optimizer [12], as suggested by [8], with hyperparameters $\alpha = 0.0001, \beta_1 = 0.5$ and $\beta_2 = 0.9$.
- We apply the *slope annealing trick* [4]. Specifically, we multiply the slopes of the sigmoid functions in the sigmoid-adjusted straight-through estimators by 1.1 after each epoch.

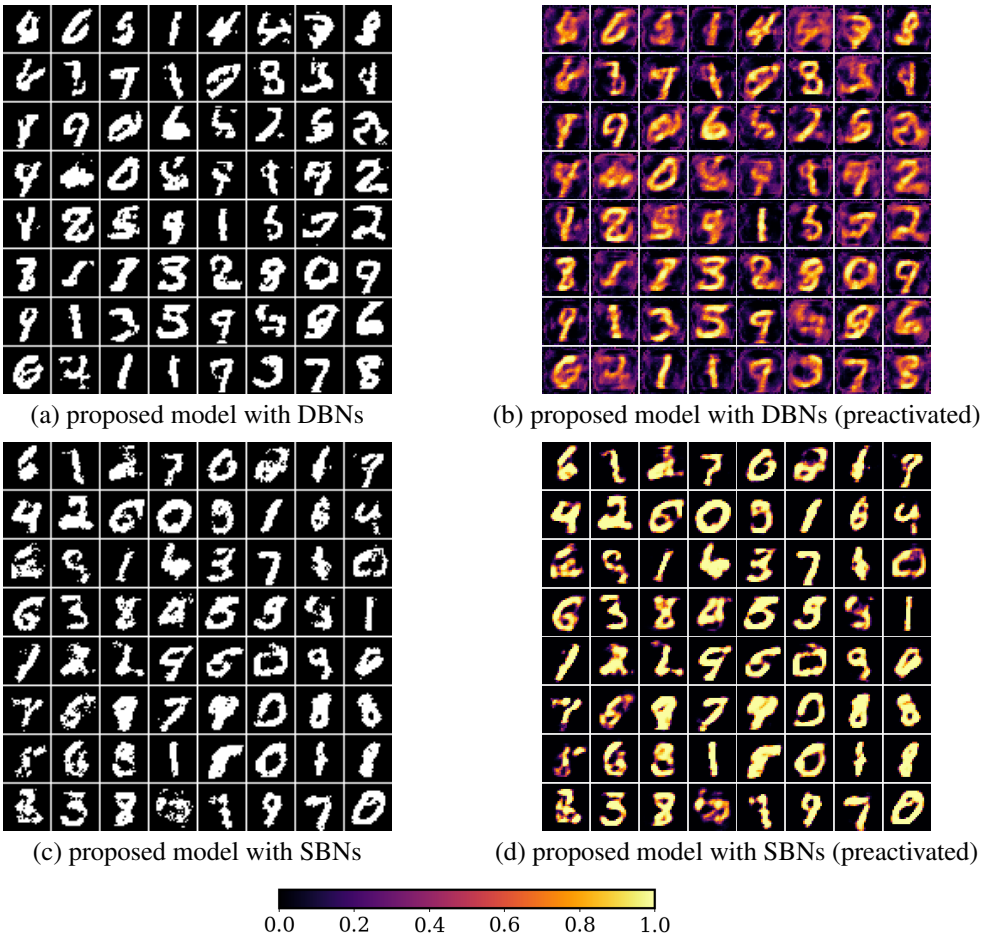
Our implementation of binary neurons are mostly based on the code provided in the blog post—"Binary Stochastic Neurons in Tensorflow"—on the R2RT blog [1].

4.3 Experiment I—Comparison of the proposed model using deterministic binary neurons and stochastic binary neurons

In the first experiment, we compare the performance of using **deterministic binary neurons** (**DBNs**) and **stochastic binary neurons** (**SBNs**) in the proposed model. We show in Figures 3(a) and 3(c) some sample generated digits for the two models. We can see that the proposed model with DBNs and SBNs can achieve similar qualities. However, from Figures 3(b) and 3(d) we can see that the *preactivated outputs* (i.e., the real-valued, intermediate values right before the binarization operation; see Section 2.2) for the two models show distinct characteristics. In order to see how DBNs and SBNs work differently, we compute the histograms of their preactivated outputs, as shown in Figure 4.

We can see from Figure 4 that the proposed model with DBNs outputs more preactivated values in the middle of zero and one, which results in a flatter histogram. We attribute this phenomenon to the fact that the output of a DBN is less sensitive to the absolute value for it depends only on whether the preactivated value is greater than the threshold. Moreover, we observe a notch around 0.5, the threshold value we use in our implementation. It seems that DBNs tend to avoid producing preactivated values around the decision boundary (i.e., the threshold).

In contrast, the proposed model with SBNs outputs more preactivated values close to zero and one, which we attribute to the fact that the output of an SBN is more sensitive to the absolute value for it relies on Bernoulli sampling (e.g., an SBN may fire even with a tiny preactivated value). As a result, in order to avoid false triggering, it seems that an SBN tends to produce a preactivated value closer to zero and one.



(a) proposed model with DBNs

(b) proposed model with DBNs (preactivated)

(c) proposed model with SBNs

(d) proposed model with SBNs (preactivated)

Figure 3: Sample generated digits and preactivated outputs (i.e., the real-valued, intermediate values right before the binarization operation; see Section 2.2) for the proposed model implemented by MLPs and trained with the WGAN-GP objective.

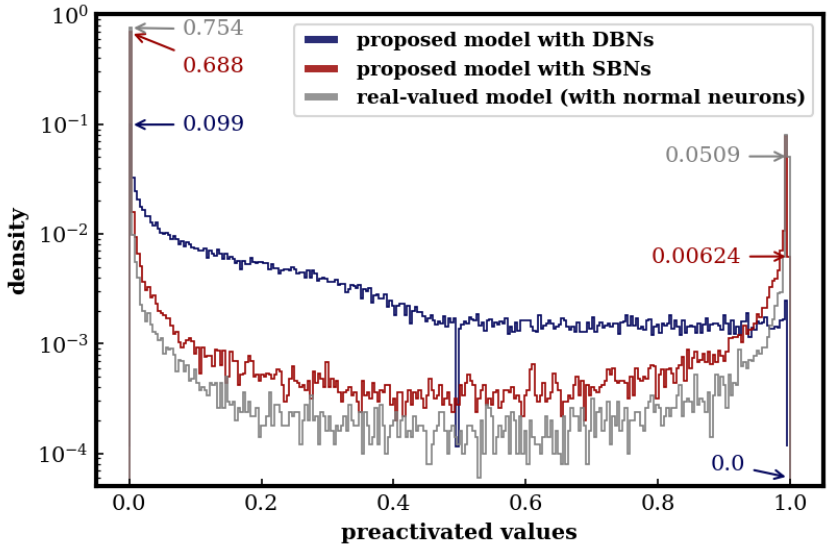


Figure 4: Histograms of the preactivated outputs for the proposed model and the probabilistic predictions for the real-valued model. The two models are both implemented by MLPs and trained with the WGAN-GP objective.

(a) raw predictions

(b) hard thresholding

(c) Bernoulli sampling

Figure 5: Sample raw predictions and binarized results for the real-valued model implemented by MLPs and trained with the WGAN-GP objective. (a) shows the raw, probabilistic outputs. (b) and (c) show the binarized results of applying a threshold of 0.5 and Bernoulli sampling, respectively, to the raw predictions shown in (a).

4.4 Experiment II—Comparison of the proposed model and the real-valued model

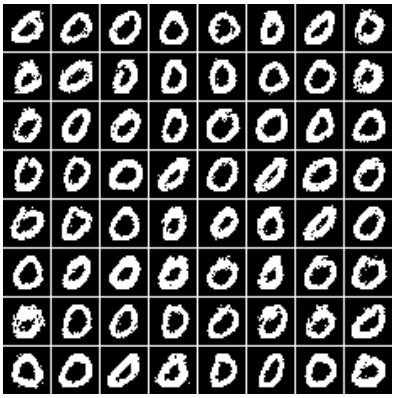
In the second experiment, we compare the proposed model with a variant that uses normal neurons at the output layer (with sigmoid functions as the activation functions).[1] We refer to this model as the **real-valued model**. Figure 5(a) shows some sample raw, probabilistic predictions of this model. Figures 5(b) and 5(c) show the final binarized results using two common post-processing strategies: *hard thresholding* and *Bernoulli sampling*, respectively.

We also show in Figure 4 the histogram of its probabilistic predictions. We can see that the histogram of this real-valued model is more U-shaped than that of the proposed model with SBNs. Moreover, there is no notch in the middle of the curve as compared to the proposed model with DBNs. From here we can see how different binarization strategies can shape the characteristics of the preactivated outputs of binary neurons. This also emphasizes the importance of including the binarization operations in the training so that the binarization operations themselves can also be optimized.

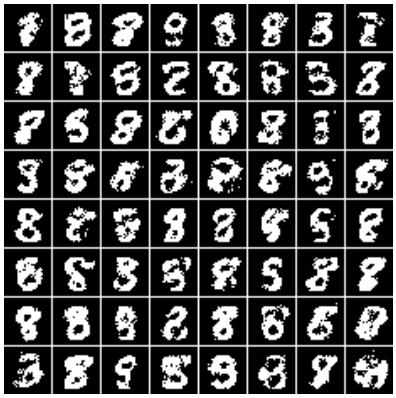
4.5 Experiment III—Comparison of the proposed model trained with the GAN, WGAN and WGAN-GP objectives

In the third experiment, we compare the proposed model trained by the **WGAN-GP** objective with that trained by the **GAN** objective and by the **WGAN** objective (using weight clipping). Implementation details are summarized as follows.

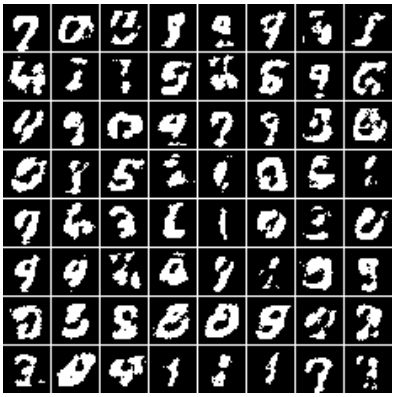
[1]This is how we train the MuseGAN model in [5]. After the training, we binarize the real-valued predictions with a threshold of 0.5 to obtain the final binary-valued results.



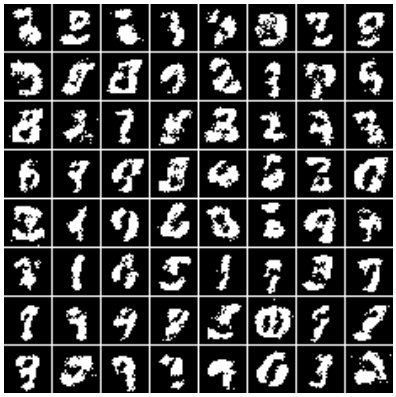
(a) GAN model (with DBNs)



(b) GAN model (with SBNs)



(c) WGAN model (with DBNs)



(d) WGAN model (with SBNs)

Figure 6: Sample generated digits for the GAN and WGAN models, both implemented by MLPs.

- For the GAN model, we apply batch normalization to the generator and the discriminator.
- For the WGAN model, we find it works better to apply batch normalization to the generator while omitting it for the discriminator.
- For the GAN model, we employ the Adam optimizer [12] with hyperparameters $\alpha = 0.0001, \beta_1 = 0.5$ and $\beta_2 = 0.9$.
- For the WGAN model, we use the RMSProp optimizer [17], as suggested by [2], with a learning rate of 0.0001 and a decay rate of 0.9.

As can be seen from Figure 6, the WGAN model is able to generate digits with similar qualities as the WGAN-GP model does, while the GAN model suffers from the so-called *mode collapse* issue.

4.6 Experiment IV—Comparison of the proposed model using multilayer perceptrons and convolutional neural networks

In the last experiment, we compare the performance of using **multilayer perceptrons** (**MLPs**) and **convolutional neural networks** (**CNNs**). For the CNN model, we implement both the generator and the discriminator as deep CNNs (see Tables 3 and 4 for the network architectures). Note that the number of trainable parameters for the MLP and CNN models are 0.53M and 1.4M, respectively.

We present in Figure 7 some sample generated digits by the proposed and the real-valued model implemented by CNNs. It can be clearly seen that the CNN model can better capture the characteristics of different digits and generate less artifacts even with a smaller number of trainable parameters as compared to the MLP model.

Layer	Number of filters	Kernel	Strides	Activation
transconv	128	2×2	$(1, 1)$	ReLU
transconv	64	4×4	$(2, 2)$	ReLU
transconv	32	3×3	$(2, 2)$	ReLU
transconv	1	4×4	$(2, 2)$	sigmoid

Table 3: Network architecture of the generator for the CNN model.

Layer	Number of filters	Kernel	Strides	Activation
conv	32	3×3	$(1, 1)$	LeakyReLU
maxpool		2×2	$(2, 2)$	
conv	64	3×3	$(1, 1)$	LeakyReLU
maxpool		2×2	$(2, 2)$	
flatten				
dense	128			LeakyReLU
dense	1			sigmoid*

*No activation for the WGAN and WGAN-GP models.

Table 4: Network architecture of the discriminator for the CNN model.

5 Discussions and Conclusions

We have presented a novel GAN-based model that can generate binary-valued predictions without any further post-processing and can be trained by end-to-end backpropagation. We have implemented such a model to generate binarized MNIST digits and experimentally compared the proposed model for different types of binary neurons, GAN objectives and network architectures. Although the results are still preliminary, we have shown that the use of gradient estimators can be a promising direction for modeling discrete distributions with GANs. A future direction is to examine the use of gradient estimators for training a GAN that has a conditional computation graph [3, 4], which allows the system to make binary decisions by binary neurons for more advanced designs.

References

[1] Binary stochastic neurons in tensorflow, 2016. Blog post on the R2RT blog. [Online] `https://r2rt.com/binary-stochastic-neurons-in-tensorflow`.

[2] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *Proc. ICML*, 2017.

[3] Yoshua Bengio, Nicholas Léonard, and Aaron C. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[4] Junyoung Chung, Sungjin Ahn, and Yoshua Bengio. Hierarchical multiscale recurrent neural networks. In *Proc. ICLR*, 2017.

[5] Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang. MuseGAN: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment. In *Proc. AAAI*, 2018.

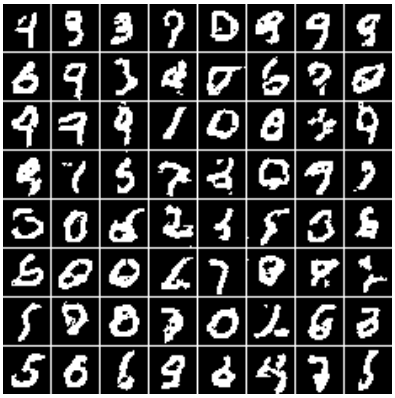
[6] Hao-Wen Dong and Yi-Hsuan Yang. Convolutional generative adversarial networks with binary neurons for polyphonic music generation. In *Proc. ISMIR*, 2018.

[7] Ian J. Goodfellow et al. Generative adversarial nets. In *Proc. NIPS*, 2014.

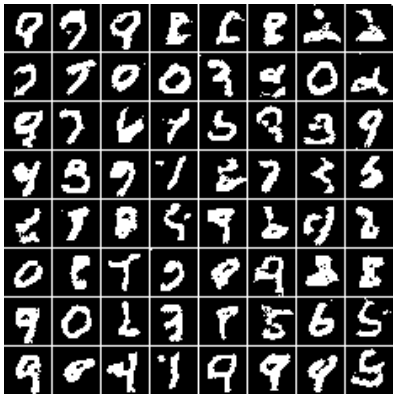
[8] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of Wasserstein GANs. In *Proc. NIPS*, 2017.

[9] Geoffrey Hinton. Neural networks for machine learning—Using noise as a regularizer (lecture 9c), 2012. Coursera, video lectures. [Online] `https://www.coursera.org/lecture/neural-networks/using-noise-as-a-regularizer-7-min-wbw7b`.

[10] R. Devon Hjelm et al. Boundary-seeking generative adversarial networks. In *Proc. ICLR*, 2018.



(a)proposed (with DBNs)



(b) proposed (with SBNs)

(c) hard thresholding

(d) Bernoulli sampling

Figure 7: Sample generated digits for the proposed and the real-valued models, both implemented by CNNs and trained with the WGAN-GP objective. (a) and (b) show the results for the proposed model with DBNs and SBNs, respectively. (c) and (d) show the binarized results of applying a threshold of 0.5 and Bernoulli sampling, respectively, to the raw predictions of the real-valued model.

[11] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proc. ICML*, 2015.

[12] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[13] Matt J. Kusner and José Miguel Hernández-Lobato. GANS for sequences of discrete elements with the Gumbel-softmax distribution. In *Proc. NIPS Workshop on Adversarial Training*, 2016.

[14] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86(11):2278–2324, 1998.

[15] Jiwei Li et al. Adversarial learning for neural dialogue generation. In *Proc. EMNLP*, 2017.

[16] Sai Rajeswar, Sandeep Subramanian, Francis Dutil, Christopher Pal, and Aaron Courville. Adversarial generation of natural language. In *Proc. ACL Workshop on Representation Learning for NLP*, 2017.

[17] Tijmen Tieleman and Geoffrey Hinton. Neural networks for machine learning—Rmsprop: Divide the gradient by a running average of its recent magnitude (lecture 6e), 2012. Coursera, video lectures. [Online] `https://www.coursera.org/lecture/neural-networks/rmsprop-divide-the-gradient-by-a-running-average-of-its-recent-magnitude-YQHki`.

[18] Zhen Yang, Wei Chen, Feng Wang, and Bo Xu. Improving neural machine translation with conditional sequence generative adversarial nets. In *Proc. NAACL*, 2018.

[19] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. SeqGAN: Sequence generative adversarial nets with policy gradient. In *Proc. AAAI*, 2017.

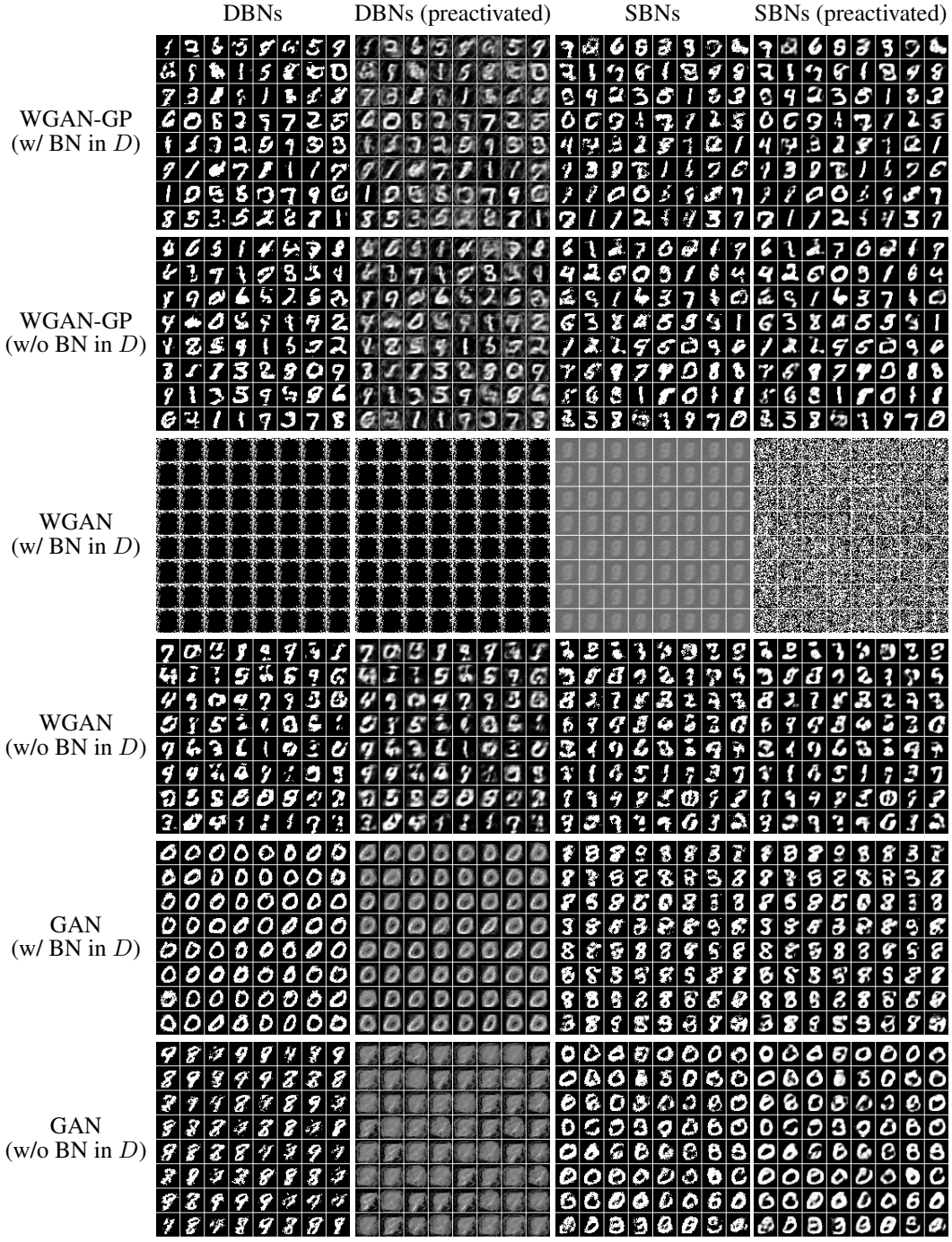


Figure 8: Sample generated digits and preactivated outputs (i.e., the real-valued, intermediate values right before the binarization operation; see Section 2.2) for the proposed model trained with different GAN objectives. Note that 'BN' stands for batch normalization.