SGAN: An Alternative Training of Generative Adversarial Networks – Supplementary Material –

Tatjana Chavdarova and François Fleuret
Machine Learning group, Idiap Research Institute & École Polytechnique Fédérale de Lausanne
{firstname.lastname}@idiap.ch

The implementation details, as well as additional results of experiments on toy and realistic datasets, are given in Sections 1 and 2. In Section 3 we discuss two viewpoints of SGAN. We use notations as in the paper (see § 4).

1. Experiments on toy datasets

1.1. Details on the implementation

For experiments conducted on toy datasets, we used separate $2{\cdot}(N{+}1)$ networks. The architecture and the hyperparameters are as follows. Each network is a multilayer perceptron (MLP) of 4 fully connected layers and LeakyReLU non-linearity [6] with the PyTorch's default value for the negative slope of 0.01 [7]. The number of hidden units for each of the layers is 512, whereas the dimension of the input noise vector for the generator network is 100.

We use learning rate of $1{\cdot}10^{-5}$, as well as the *Adam* optimization method [5]. Using *RMSProp* [12] as optimization method did not give obvious improvements in our conducted experiments.

1.2. Experiments

Figure 1 depicts several image pairs, of: (i) samples generated by the local generators (left); and (ii) samples from the global one (right). The illustrated contours are obtained with GMM Kernel Density Estimation (KDE) [10], whose bandwidth is cross-validated. We used sample of p_g of size 500 (in Figure 1, N denotes the sample size). *C* in Figure 1 denotes the *Coverage* metric [13].

Figure 4 depicts experiment in which the parameters of the global pair are updated after each update of a local pair.

2. Experiments on real-world datasets

2.1. Details on the implementation

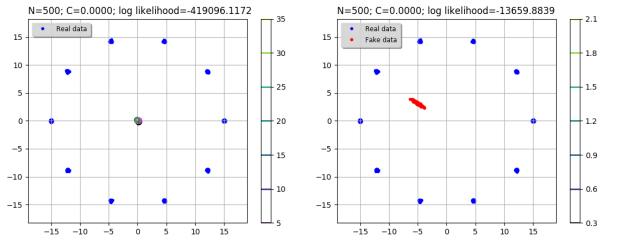
Regarding experiments on real-world applications, we considered: (i) using separate $2{\cdot}(N{+}1)$ networks; as well as (ii) using parameter sharing of the networks. In the latter, approximately half of the parameters of each network are shared among the corresponding other N networks (discriminators or generators). To distinguish the two, in the sequel we denote the former and the latter case as **N-S-** and **N-SW-**, respectively. For *DCGAN, we recommend using separate networks*, as sharing parameters makes the generators to produce similar samples, thus the performance gain of SGAN can be marginal.

We used learning rate of $1{\cdot}10^{-5}$, and a batch size of 50 and 64 for (FASHION)MNIST and the rest of the datasets, respectively. Unless otherwise stated, we used the *Adam* optimizer [5] whose hyperparameters (one parameter used for computing running averages of gradient and another for its square) we fixed to 0.5 and 0.999, as in [9].

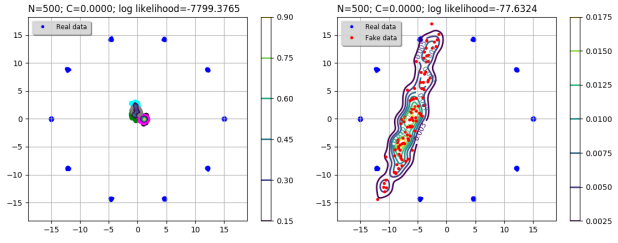
Implementation of the experiments on image datasets. For **MNIST** we did experiments using both MLPs and CNNs for the generators and the discriminators. In the former case, the architectures were almost identical to those used for the toy experiments, except that the first layer was adjusted for input space of $28{\times}28$. In the latter case, we used input space of $28{\times}28$ and we started with the DCGAN implementation [9] and changed it accordingly to the input space. In particular, we reduced the number of 2D transposed convolution layers from 5 to 4 and adjusted the hidden layers' sizes accordingly to the dimensions used for the real data space.

For **CIFAR10** unless otherwise emphasized, we used $32{\times}32$ image space. For the rest of the image datasets– unless otherwise stated, we used $64{\times}64$ input space and the original DCGAN [9] architecture, as provided by the authors. The implementation of DCGAN [9] uses Batch Normalization layers [4].

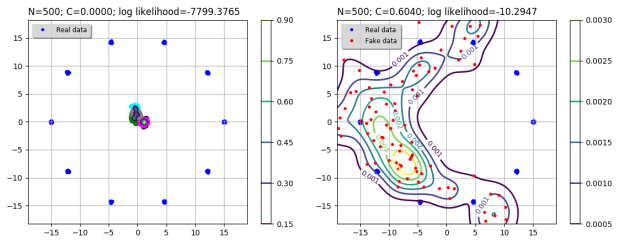
Implementation of the experiments on one Billion Word Benchmark. We started from the provided implementation of [2] and implemented our method. In particular, the character-level generative language model is implemented as a $1D$ CNN using 4 ResNet blocks [3], which network maps a latent vector into a sequence of one-hot character



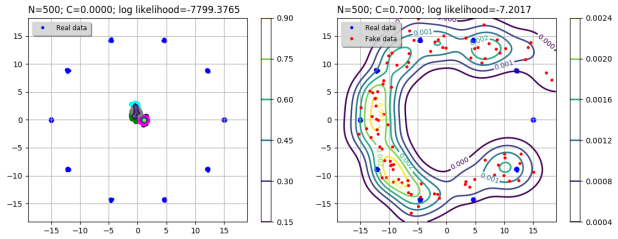
Iteration 1



Iteration 8



Iteration 40



Iteration 125

Figure 1: **5-S-WGAN** on the **10-GMM** dataset. Samples from the five local generators and from the global generator, are shown on the left (in separate color) and on the right (in red color), respectively. See § 1.2.

vectors of dimension 32. The discriminator is also a $1D$ CNN, that takes as input sequences of such character embeddings of size 32.

As optimization method we used *RMSProp* [12].

Separate networks. In Figure 3 we show the Inception score [11] (using its original implementation in TensorFlow [1]), of the global generator and the local generators.

In Figure 2 we show samples of **5-S-DCGAN** on **FASHION-MNIST** (on the right), as well as of **DCGAN** (on the left). Figures 5 & 6 depict samples using **DRAGAN** and **DCGAN**, respectively. We see that the global generator converges much earlier then the local ones.

Figure 2: Samples of **DCGAN** and **5-S-DCGAN** on **FASHION-MNIST** taken at the 6000-*th* iteration, on the left and right, respectively. The input dimension is 28×28.

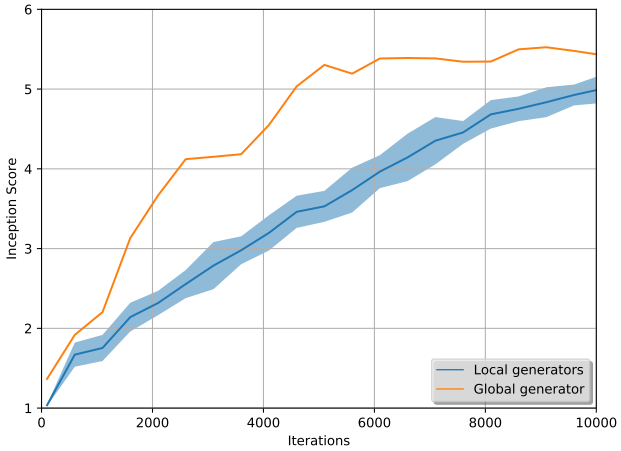


Figure 3: **10-S-DCGAN**, on **CIFAR10** (best seen in color). We plot the Inception Score [11] of the global generator (orange) as well as the scores of the local generators (blue). The input dimension is 32×32.

Shared parameters. In Figure 9 we show samples when training **DRAGAN** and **5-SW-DRAGAN** on **LSUN-bedroom** with input dimension of 64×64. Finally, in Figure 10 we show samples when training on the **Billion Word** dataset.

3. Different viewpoints of SGAN

Connecting SGAN to Actor-critic methods. In [8] the authors argue that at an abstract level GANs find similarities with actor-critic (AC) methods, which are widely used in reinforcement learning. Namely, the two have a feed-forward

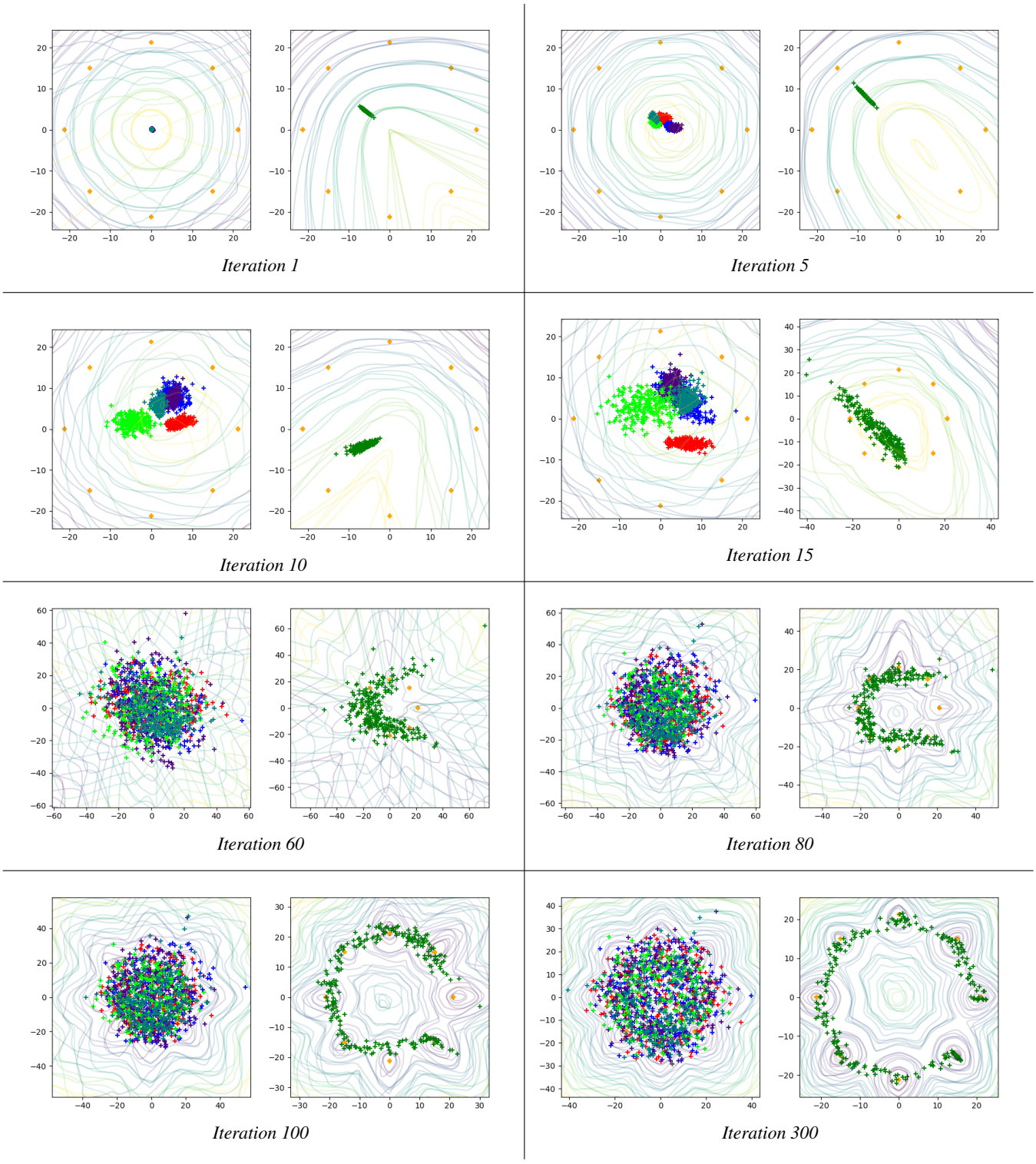


Figure 4: **5-S-WGAN** experiment on the **8-GMM** toy dataset (best seen in color). Real data samples are illustrated in orange. In each image pair, we illustrate samples from the five local generators and from the global generator, on the left (in separate color) and on the right (in green), respectively. The displayed contours represent the level sets of the discriminators $\mathcal{D}$ and $\mathcal{D}^{msg}$–illustrated on the left and right of each image pair, respectively, where yellow is low and purple is high.

Figure 5: **5-S-DRAGAN** on **CIFAR10** at $40{\cdot}10^3$-*th* iteration, and $32{\times}32$ real data space.

Figure 6: **5-S-DCGAN** on **CelebA** at $1{\cdot}10^3$-*th* iteration, and $32{\times}32$ real data space.

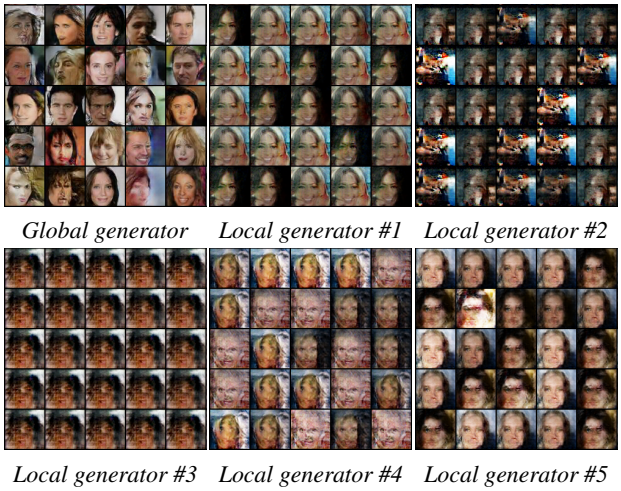
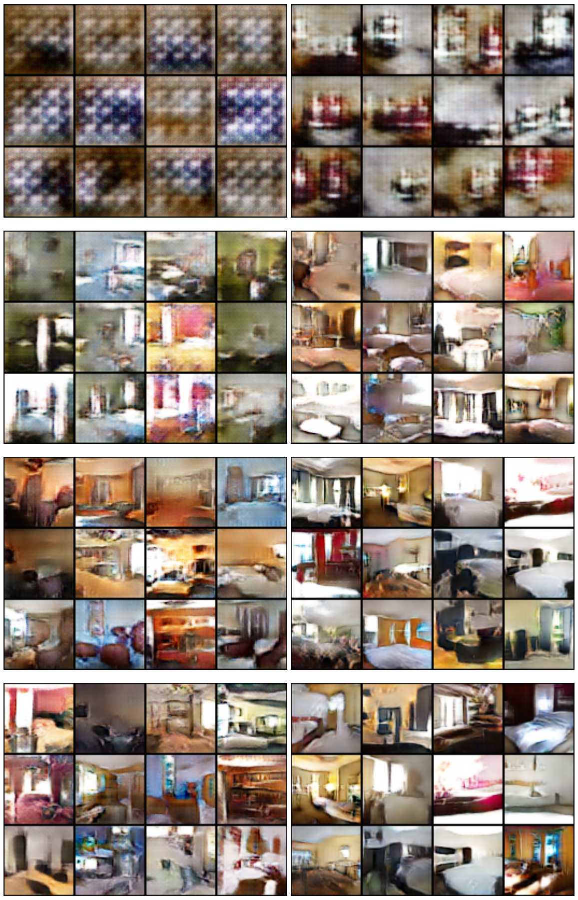


Figure 7: Samples of the generators of **5-S-DRAGAN** on the **CelebA** dataset at the $50{\cdot}10^3$-*th* iteration. The input dimension is $64{\times}64$.

Figure 8: Samples of the generators of **5-S-DCGAN** on the **LSUN-bedroom** dataset at the $100{\cdot}10^3$-*th* iteration. The input dimension is $64{\times}64$.

model which either takes an action (AC) or generates a sample (GAN). This acting/generating model is trained using a second one. The latter model is the only one that has direct access to information from the environment (AC) or the real data (GAN), whereas the former has to learn based on the signals from the latter. We refer the interested reader to [8] which further elaborates the differences and finds connections that both the methods encounter difficulties in training.

We make use of the graphical illustration proposed in [8] of the structre of the GAN algorithm illustrated in Figure 11a, and we extend it to illustrate how SGAN works,



DRAGAN **5-SW-DRAGAN**

Figure 9: **DRAGAN** and **5-SW-DRAGAN** on **LSUN-bedroom** at the 1000-*th*, $5{\cdot}10^3$-*th*, $10{\cdot}10^3$-*th* and $14{\cdot}10^3$-*th* iteration, from top to bottom row, respectively. Using $64{\times}64$ real data space.

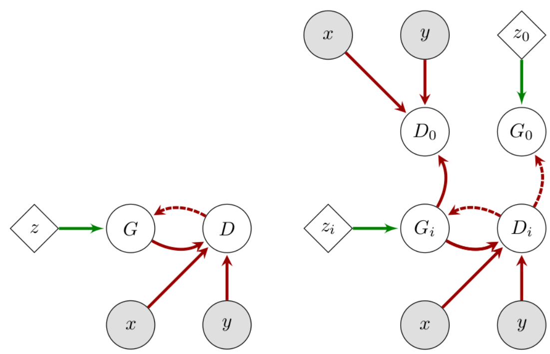
```
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
```

```
The Larntare F bucnt 1h setirder
The tielirc indian on orabo stir
We dola wan Fobkbomn hrcas and 8
Tod letrocsix r telt car rntr ce
Then on s indent on leWand ghis
Ja candt conteltirt in ald do ce
Gaid nochir Weabsilan of ansany
The ticosgose arc on Mesinntelat
More wucarcanosnped rochusroe t`
They derato raEyand soalceatecst
De Ths chsrc s aeareP thel ea t
```

```
" S4vvvoFlnls anr ans ffrcinns s
Thon taa forinint ssroso siwfd f
Hothst bffld ’nvlonyoiar" cov sh
Woisi’n fof Monisg dhak N‘f fnv
ThD fas ong fafpn so n wns is of
D" wiay wyd alvriMbnlor M nld ff
DDd4y vooc onl vocfay w s offo f
" c4Df co  noy soonlono ans war
ln wns bfrfncorfiw Thofv lnnd fo
Dy wad Dld at N fovl dcy fot aor
ThD doDn bacd d vffnonlo anfofin
Th’ toits isg thid st hsgo ffffs
" coracgod Mfopf lny thisg  aff
```

```
The conareed same ming tay spid
Then the gioncolly the can id co
She  beme lant arecong nelode .
It taecanting the they trehos so
In the later asterol antarlist n
Sion ot tndy ttin an os pomcerer
The pither canned Sblets castery
They BastitiBented tome man angu
I lant taot suncedrthet prourpli
And Biecon eels acecount tre Car
This tain Datertlals comegel yan
Is the Woilg  ate costort thab f
Thin Incin is Inlasar cumand mot
```

Figure 10: Snippets from **WGAN** (left) and **5-SW-WGAN** (right) on the **One Billion Word Benchmark**, taken at the 700-*th* and 2500-*th* iteration (top and bottom row, respectively).



(a) GAN training [8] (b) SGAN training

Figure 11: Graphical representations [8] of the information flow structures of GAN and SGAN training. See § 3.

Figure 11b, where nodes with index i can be multiple. Empty circles represent models with a distinct loss function. Filled circles represent information from the environment. Diamonds represent fixed functions, both deterministic and stochastic. Solid lines represent the flow of information, while dotted lines represent the flow of gradients used by another model. In SGAN, D_0 is being trained with samples from the multiple generators whose input is in the real-data space. For clarity, we omited D^{msg} in the illustration–used to train G_0, as the arrows already indicate that these two "global" models do not affect the ensemble.

Game theoretic interpretation. We can define a game that describes the training of G_0 and D_0 in the SGAN framework as follows. Let us consider a tuple $(\mathcal{P}, \mathcal{A}, u)$, where $\mathcal{P} = \{G, D\}$ is the set of new players that we introduce. Let us assume that G and D, at each iteration can select among the elements of $\mathcal{D}$ and $\mathcal{G}$, respectively. Hence, $\mathcal{A} = (A_g, A_d)$ have a finite set of N actions.

Such "top level players" in SGAN assign uniform distribution over their actions, more precisely both G and D sample from the elements of $\mathcal{D}$ and $\mathcal{G}$ respectively, with uniform probability. To connect to classical training, let us assume that G and D fix their choice to one element of $\mathcal{D}$ and $\mathcal{G}$ respectively, *i.e.* with probability one they sample from a single generator/discriminator. The trained networks G_0 and G_i, as well as D_0 and D_j, with i and j being the selected choice of G and D respectively, are identical in expectation. Finally, rather than predefining the uniform sampling in SGAN, incorporating estimations of the actions' pay-off $u = (u_g, u_d)$ could prove useful for training (G_0, D_0).

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org. 2

[2] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville. Improved training of Wasserstein GANs. In *Advances in Neural Information Processing Systems 30*, pages 5767–5777. 2017. 1

[3] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015. 1

[4] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 448–456. JMLR.org, 2015. 1

[5] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. 1

[6] A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier nonlinearities improve neural network acoustic models. 2013. 1

[7] A. Paszke, S. Gross, S. Chintala, and G. Chanan. PyTorch. https://github.com/pytorch/pytorch, 2017. 1

[8] D. Pfau and O. Vinyals. Connecting generative adversarial networks and actor-critic methods. *CoRR*, abs/1610.01945, 2016. 2, 4, 5

[9] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434, 2015. 1

[10] M. Rosenblatt. Remarks on some nonparametric estimates of a density function. *Ann. Math. Statist.*, 27(3):832–837, 09 1956. 1

[11] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, X. Chen, and X. Chen. Improved techniques for training GANs. In *Advances in Neural Information Processing Systems 29*, pages 2234–2242, 2016. 2

[12] T. Tieleman and G. Hinton. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012. 1, 2

[13] I. O. Tolstikhin, S. Gelly, O. Bousquet, C.-J. Simon-Gabriel, and B. Schölkopf. AdaGAN: Boosting generative models. In *Advances in Neural Information Processing Systems 30*. 2017. 1