Experiments on constructing seismic using generative adversarial network

Zhan Niu and Daniel Trad

ABSTRACT

Supervised machine learning attracts great attention in all areas of science. In Geophysics, however, supervised learning has the problem that available labelled data is often insufficient, limiting the chance of converging during training and harming model generality. As a solution, researchers explore ways to generate synthetic data for use in training. In this report, we explore the methodology of generating 1D data with a generative adversarial model. Both the generator and discriminator are convolutional, and the noise vectors are fed along the channel dimension to the generator. The networks are successfully trained via Wasserstein loss with gradient penalty and careful hypermeter tuning. We evaluate the trained networks quantitatively and qualitatively. We attempt to find the optimal stopping point for the training, however, the conclusion cannot be made during the training and part of it remains subjective.

INTRODUCTION

Machine learning has become a popular topic in most sciences and geophysical applications are not an exception. In Geophysics, many successful applications of supervised machine learning have been published, in particular in the area of image segmentation, for example facies recognition, salt body segmentation, relative geological time picking, etc.

Although applications have become more robust in the image recognition field, we still face a significant challenge that does not exist in the broader machine learning society: the lack of abundant public labelled data. The abundance of data is perhaps more crucial and needed to solve geophysical problems than in other areas like image classification because seismic data interpretation for example relies in subtle details with complex relations between physics and geology. Therefore, in order to solve problems involving a complex theory by using machine learning, geophysical research injects theoretical knowledge through the use of complex network architectures or applies physics guided regularization to compensate for the gaps in information during the learning process. As a partial solution, researchers have tried to generate synthetic data with satisfying quality for use in training and improve model convergence and model generality. For example, Wu et al. (2019) successfully trained a relatively ordinary U-Net with synthetic fault images. The images are very carefully generated, so the trained model can provide accurate results even on real data and easily adapt to other scenarios without harm in accuracy using transfer learning. However, we do not always have abundant or precise knowledge to model the data. Generative adversarial networks (Goodfellow et al., 2014, GANs) is capable of this kind of tasks. A successfully trained generator can produce artificial data in a given data distribution.

This report is our first attempt to test GANs in synthetic data modeling. We will use very simple cases to generate seismic data. At this stage, the main goal is to understand the characteristics of GAN and its behaviour during training and opening the door for further

research.

THEORY

Generative adversarial network

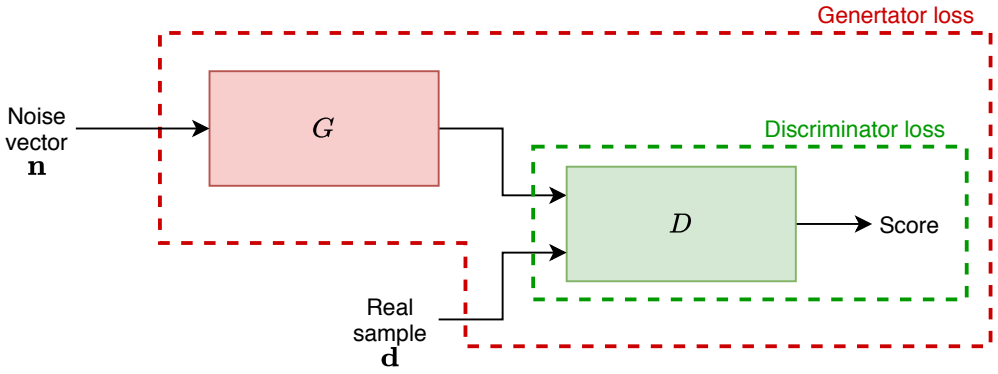

FIG. 1. A typical structure of unconditional GAN

Figure 1 shows a typical structure of a GAN. There are two sub-networks in a GAN, the generator (G) and the discriminator (D, which is sometimes called the critic depending the form of the output). G generates some samples in the form of random vectors $\boldsymbol{n}$, while D distinguishes the generated samples from real samples $\boldsymbol{d}$. The two networks compete against each other and try to improve themselves during the training. G will try to learn how to trick D by generating more realistic predictions, while D will try to become more aware of the differences between the generated and real samples, which will be changing as G improves.

The main goal of a GAN is to find a transform from a random distributed variable to a given data distribution. The input of a GAN is usually a uniformly randomized noise vector $\boldsymbol{n}$ and the output are not treated as the "ground truth" as we do in the supervised learning. One individual input is never learnt to be bonded to a label.

GANs are notoriously famous for their difficulty to be properly trained. There are two critical aspects of designing a GAN. First, the accuracy of the network mostly depends on the robustness of the discriminator due to the way the value function is defined. The discriminator must be capable of its job and able to be trained with general approaches. Second, the success is based on balancing the training of the generator and the discriminator. If the discriminator is too strong or learns too fast compared to the generator, it cannot provide useful feedback for the generator to continue the learning. In this case, the discriminator will always reject the model no matter how the generator modifies its parameters, so the generator is likely to get trapped in local minima and fail to escape from it because of unhelpful gradients. On the other hand, if we have a generator that is much superior to the discriminator, the generated example will always fool the discriminator. Because the "same thing" is labelled to be both right and wrong from the perspective of the discriminator, it will confuse and refuse to improve itself.

Because of the reasons above, training a GAN is where science forgets its modern role and becomes alchemy (Chollet, 2018). Many empirical tricks need to be applied to the model, and they may not be suitable in other cases.

Wasserstein GAN with gradient penalty

Here we use the value function from WGAN (Arjovsky et al., 2017) for more stable training. The value function is defined as

$$V_W = \min_G \max_D \mathbb{E}[D(\boldsymbol{d})] - \mathbb{E}\left[D(G(\boldsymbol{n}))\right], \tag{1}$$

where $\boldsymbol{n}$ refers to the random latent vector and $\boldsymbol{d}$ refers to real data. In practice, the expectations are simply replaced by the mean value of the current mini-batch. Equation 1 defines a min-max game, in which we want to find a $D(\cdot)$ that maximizes its expected score on real examples while minimizes its expected score on the generated ones. Meanwhile, we find a $G(\cdot)$ that maximizes its expected score from the discriminator. Gulrajani et al. (2017) propose a gradient penalty as a regularization term in addition to Equation 1 to enforce 1-Lipschitz constraint. The term is defined as

$$V_P = \left(\left\| \frac{\partial D(\boldsymbol{m})}{\partial \boldsymbol{m}} \right\|_2 - 1 \right)^2, \text{where } \boldsymbol{m} = \epsilon \boldsymbol{d} + (1-\epsilon) G(\boldsymbol{n}). \tag{2}$$

Here $\boldsymbol{m}$ refers to the mixing of real and generated samples, which is controlled by a random scalar ratio ϵ. The value ϵ is drawn from $U(0,1)$ at each discriminator update to lower the change of being stuck in some local minima by introducing more stochasticity. The partial derivative can be calculated with auto differentiation. Minimizing Equation 2 will favour the discriminator gradients with a unitary norm, therefore clipping larger unstable gradients and guiding the model to avoid small updates.

By combining Equation 1 and 2, we obtain the following loss functions required for updating the generator and discriminator parameters:

$$L_G = -\mathbb{E}\left[D(G(\boldsymbol{n}))\right], \tag{3}$$

$$L_D = \mathbb{E}\left[D(G(\boldsymbol{n}))\right] - \mathbb{E}[D(\boldsymbol{d})] + \lambda V_P. \tag{4}$$

Note the negative sign in L_G since the two losses are opposing each other. The min-max problem then becomes two optimization problems where the two networks are updated according to the losses in an alternating fashion. The trainable parameters in the discriminator are temporarily frozen when updating the generator using Equation 3, and the parameters in the generator are frozen when updating with Equation 4.

METHOD

Architecture

Since the problem is relatively simple, we designed two small networks from scratch. We use PyTorch (Paszke et al., 2019) as the machine learning framework. Both the generator and the discriminator are constructed using sequential 1D convolutional layers. The noise vectors $\boldsymbol{n}$ with a length of 100 are fed into the generator via the channel dimension with a size of 1 in the spatial dimension, which is gradually increased by undergoing a sequence of 1D transposed convolutional layers with proper kernel sizes and strides. The first layer in the sequence has 256 filters. The number of filters is halved multiple times in the successive

layers. At the last layer, the number of channels is reduced to 1, and the spatial dimension is expanded to 499 to match the length of traces $\boldsymbol{d}$ from the forward modelling.

Table 1 shows the details on the output dimension after each transposed convolutional layer in the generator. Without zero padding and dilation, the output dimension of the ith layer can be calculated as

$$l_i = l_{i-1}s_i + k_i \tag{5}$$

where s_i and k_i refers to the stride and kernel size of the ith layer, respectively.

Table 1. The detailed structure of the genertator

layer	channel	length	kernel size	stride	# filters
1	100	1	3	2	256
2	256	3	4	1	256
3	256	6	4	2	128
4	128	14	4	2	128
5	128	30	3	2	32
6	32	61	4	2	32
7	32	124	3	2	16
8	16	249	3	2	1
output	1	499			

The discriminator is fully convolutional. It takes input that has 1 channel with length of 499 and make it through 4 convolutional layers with kernel size of 3. Then the length and channel dimensions of the output are switched. Finally, the output goes through two 1×1 convolution layer to be packed to a scaler score for each samples. The details are summarized in Table 2.

Table 2. The detailed structure of the discriminator

layer	channel	length	kernel size	# filters
1	1	499	3	64
2	64	499	3	64
3	64	499	3	64
4	64	499	3	1
5	499	1	1	250
6	250	1	1	1
output	1	1		

We use leaky ReLU instead of ReLU as inter-layer activation function in both generator and discriminator. We also introduce batch normalizations before each convolutional layers in the generator only, since the discriminator remains more stable during training compared to the generator.

THE DATASET

The data is generated by 1D forward modelling with the direct arrival removed by subtraction. We use simple velocity models with four horizontal layers with random interval velocity and thickness. As a source wavelet we use a Gaussian function at shallow locations. The model has a free-surface boundary condition and absorbing boundary at depth. Since the source position is shallow, the primary wave overlaps with the ghost wave from the surface boundary and forms a unique waveform (see orange traces in Figure 2) 10000 traces are generated in total to ensure continuous distribution. Each trace has 2000 timesteps and is later resampled and trimmed to 499 to make the generator training-friendly.

The data are divided by 10 times the global mean for normalization. No bias is removed from the data to avoid shifting the origin. The number 10 was obtained empirically, which is a bit unfortunate because it is crucial for convergence. This number is bounded to the initialization of trainable parameters in both networks. Three traces after normalization are shown in Figure 2 as orange lines.

TRAINING DETAILS AND WORKFLOW

Since the GAN consists of two networks, we have to define two separate optimizers, one for each of them. Both networks use an Adam optimizer (Kingma and Ba, 2017) with a learning rate of 1×10^{-4} and a $\beta_1 = 0.5$ lower than the default value. This ensures that the model updated is more influenced by the current gradient than by the momentum part. Based on experiments, it is crucial to use additional methods to stabilize the training since the value function mentioned in the previous section will react more wildly than a common loss function like binary cross-entropy or mean square error. The convolutional kernels in both networks are initialized with a standard deviation of 0.2, which is smaller than PyTorch's default, to avoid huge predictions at early stages. The λ in Equation 4 is set to 10.

We train the GAN for 300 epochs. We load the data with a batch size of 64 on each $16\,\mathrm{GB}$ graphic card. Moreover, we trained the generator once but the discriminator twice at each iteration to balance the power of the two networks during training. The training workflow is shown as Algorithm 1.

RESULTS AND DISCUSSIONS

Manual inspection

One major issue of evaluating results from GAN is the lack of proper metrics. There are quantitative measurements that check if the generated examples are in the same distribution as the provided data. However, there is still no clear metrics that directly show us when to stop the training. One intuitive and still efficient way of verifying is to check the generated samples manually. From inspection, the generator stops improving efficiently after the 100th epoch, despite oscillations continuing on the lost functions. Figure 2 shows the result after training for 100 epochs.

In Figure 2, we can see that the generated traces look like the real data. The number on the upper-left refers to the scores from the discriminator. Note that the negative signs do not

Algorithm 1 Training workflow.

Require: $G(\cdot)$, $D(\cdot)$, $\boldsymbol{d}$
  for each epoch **do**
    for each mini-batch **do**
      for counting 2 **do**
        generate noise vectors $\boldsymbol{n}$
        $\hat{\boldsymbol{d}} \leftarrow G(\boldsymbol{n})$ ▷ generate fake data
        $S_{\text{fake}} \leftarrow D(\hat{\boldsymbol{d}})$ ▷ get the score of fake data
        $S_{\text{real}} \leftarrow D(\boldsymbol{d})$ ▷ get the score of real data
        $\boldsymbol{m} \leftarrow \epsilon \boldsymbol{d} + (1-\epsilon)\hat{\boldsymbol{d}}$
        $V_p \leftarrow \dfrac{\partial D(\boldsymbol{m})}{\partial \boldsymbol{m}}$ ▷ gradient panalty
        update $D(\cdot)$ based on Equation 4 ▷ back-propagate and apply Adam
      generate another noise vectors $\boldsymbol{n}$
      $\hat{\boldsymbol{d}} \leftarrow G(\boldsymbol{n})$ ▷ generate another fake data
      $S_{\text{fake}} \leftarrow D(\hat{\boldsymbol{d}})$ ▷ get score on fake data
      update $G(\cdot)$ based on Equation 3
    visualize $\hat{d}$ and save $D(\cdot)$ and $G(\cdot)$ regularly

have physical meaning since the score is not bounded and only relative difference matters. We can see that the generated traces achieved similar scores as the real data, which means the discriminator treats them as the same. Specifically, the zero response of the discriminator is -10.824, which means the trained discriminator still cannot distinguish zero traces and traces with reflections. However, the discriminator responses to white noises range from -45 to -38, which can be safely considered as "different".

One common problem of a GAN is mode collapse, where the generator learns only one style presented by the data. In our case, the generator may end up producing similar traces all the time. The model collapse is less likely to have happened in our case since the result in Figure 2 shows great divergence. Besides, most of the examples can reproduce the unique waveform mentioned earlier in both normal and reversed polarity.

Figure 3 shows the loss curves for the first 100 epochs and Figure 4shows the evolution of generated examples during the process. Since the two losses are competing with each other, we can see the loss curves are not guaranteed to drop all the time. In general, the curves are in a mirror relationship. Most severe competition happens during the first epochs. Before the 10th epoch, the discriminator loss decreases drastically. This is because the discriminator's job at the early stage is to distinguish white noise generated by the generator (Figure 4a) and physically meaningful real data (Figure 4f), which is considerably easy. In the meanwhile, this is also a corresponding steep increase in the generator loss. Although the generator loss is increasing, the generator is much improved (Figure 4b) because the discriminator's feedback is useful. As the generator generates more reasonable results, the discriminator's job is not easy anymore. After the 20th epoch, the generator loss starts to decrease, which indicates the discriminator learns slower compared to the generator. By comparing Figure 4d, 4e and 4f, we conclude that the two networks reach equilibrium and

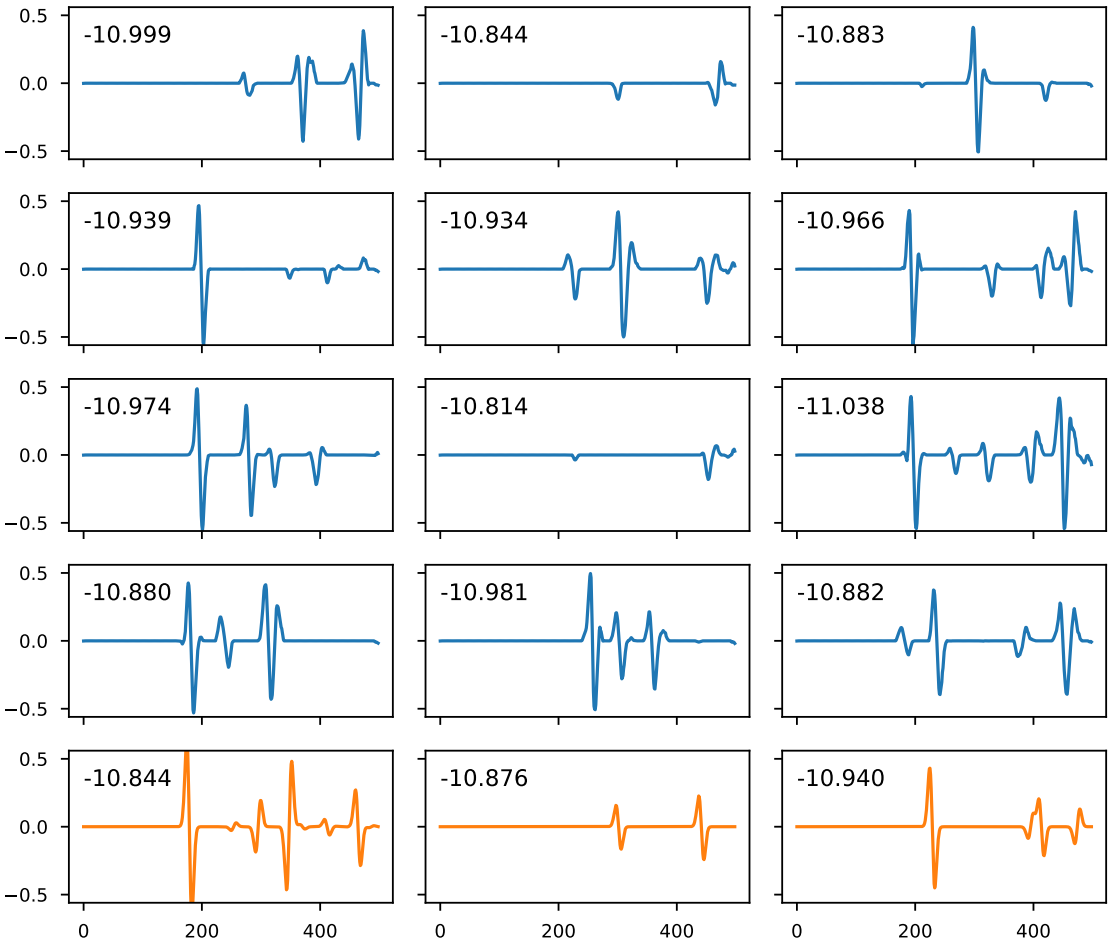

FIG. 2. The results from the trained generator. The blue curves refer to the generated traces while the orange traces are from the data. The number on the upper right in each subplot refers to the scores obtained from the discriminator. The higher the score, the better it looks from the perspective of the discriminator.

can hardly be improved.

Quantitative analysis

Figure 5 shows the distribution of the scores on the trained generator from the trained discriminator after 100 epochs. The real data score distribution is shown in blue, which can be assumed to be Gaussian. The mean discriminator score of the real data is -10.940, and that of the generated samples is -10.932. The means are close and the mean from generated samples is slightly higher than the real data distribution. This indicates the discriminator may get confused and stops improving itself. Figure 6 shows the mean generated score using the discriminator at the 100th epoch. Note that the generator stops improving from the perspective of the 100th discriminator, even though its gradients comes from discriminator at later epochs. We can infer that both the generator and the discriminator stops improving at around 100th epoch, which roughly agree with our observations using manual inspection. Therefore, we chose the models at the 100th epochs to be the best model. However, this

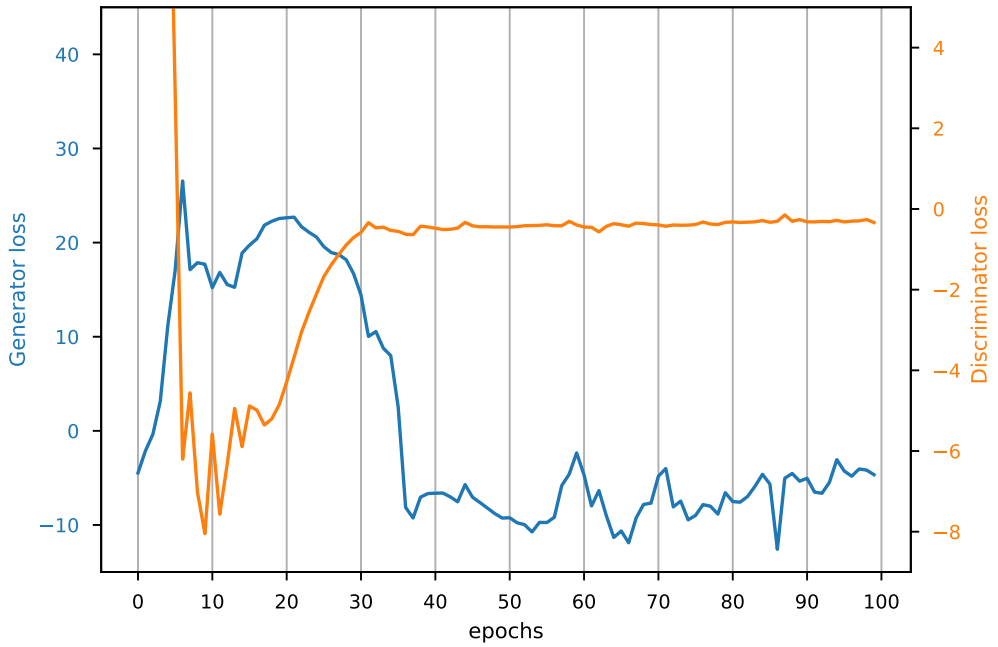

FIG. 3. GAN loss curves. The blue and the orange lines refer to the losses of generator and discriminator, respectively. The losses are defined by Equation 3 and 4.

conclusion is subjective and made after the training process. There is no clear metrics indicating the stopping point during the training and further study is needed on this topic.

CONCLUSIONS

In this report, we explore a way of generating 1D seismic traces using WGAN. The trained generator successfully transform uniformly distributed noise vectors to data distribution generated by the forward modelling. The two models reach equilibrium at around 100 epochs and hardly improve each other afterwards. The generated samples from the trained model preserve the unique waveform of real data, despite of the discriminator still lacks the ability to distinguish empty traces from real examples. The future work will be expanding the same model architecture to 2D and applying conditions to the noise vector to gain more control over the generation process.

ACKNOWLEDGEMENTS

We thank the sponsors of CREWES for continued support. This work was funded by CREWES industrial sponsors, NSERC (Natural Science and Engineering Research Council of Canada) through the grants CRDPJ 461179-13 and CRDPJ 543578-19.

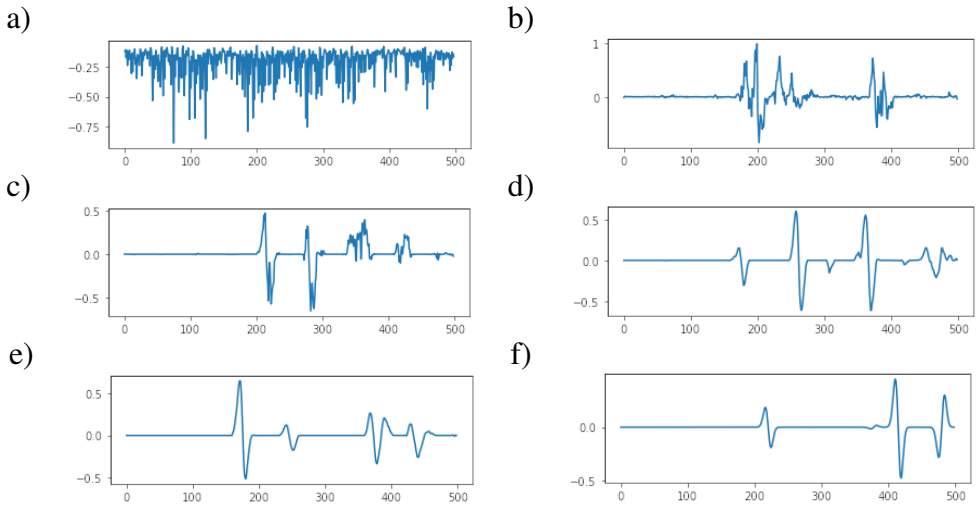

FIG. 4. Predictions from the generator at **a)** 1st epoch; **b)** 5th epoch; **c)** 20th epoch; **d)** 47th epoch; **e)** 100th epoch and **f)** refers to a sample from real data for comparison.

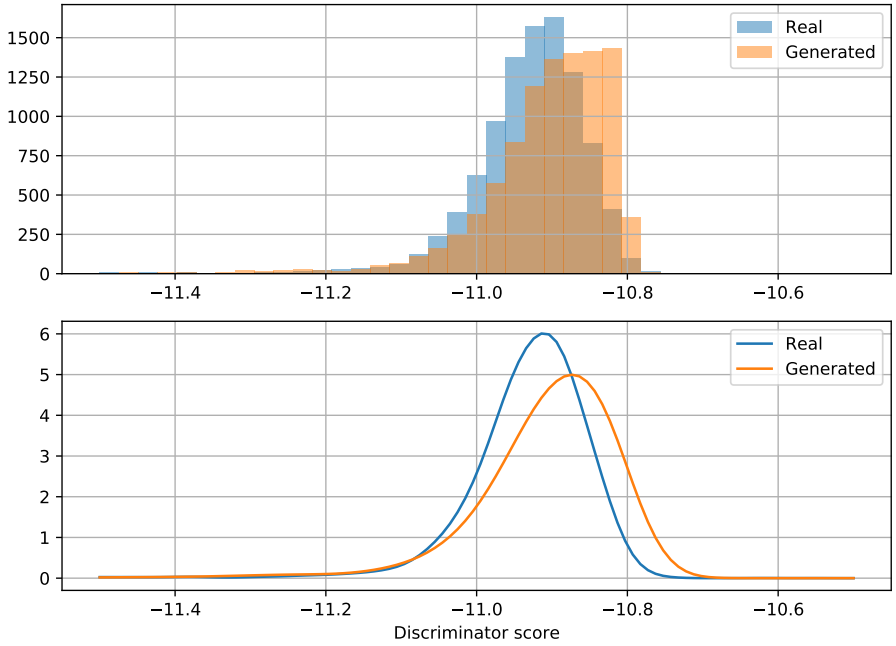

FIG. 5. Histogram and kernel density estimation of real data and generated samples. The parts in blue represent the results from real data distribution, while the orange parts represent the results from generated examples.

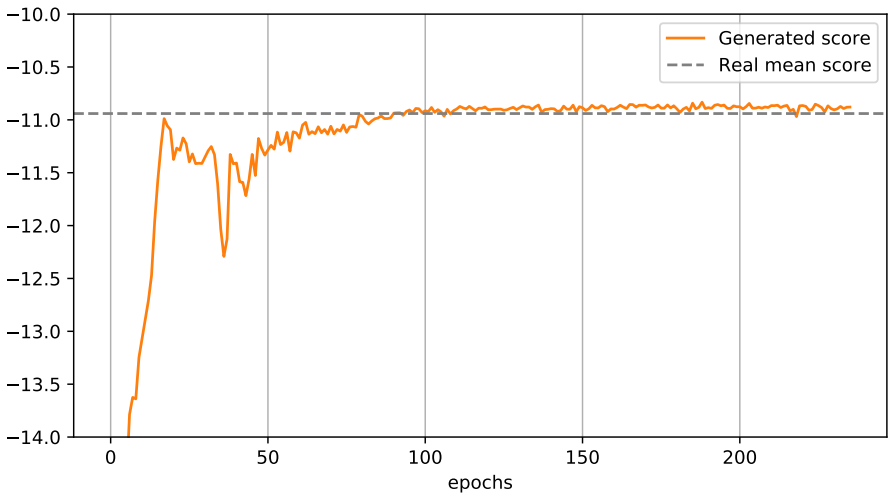

FIG. 6. Mean generator score using the discriminator from the 100th epoch. The grey dashed line refers to the mean real data score, which is -10.940.

REFERENCES

Arjovsky, M., Chintala, S., and Bottou, L., 2017, Wasserstein gan, 1701.07875.

Chollet, F., 2018, Deep Learning mit Python und Keras: Das Praxis-Handbuch vom Entwickler der Keras-Bibliothek: MITP-Verlags GmbH & Co. KG.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y., 2014, Generative adversarial networks, 1406.2661.

Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A., 2017, Improved training of wasserstein gans, 1704.00028.

Kingma, D. P., and Ba, J., 2017, Adam: A method for stochastic optimization, 1412.6980.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S., 2019, PyTorch: An Imperative Style, High-Performance Deep Learning Library: Curran Associates, Inc.

Wu, X., Liang, L., Shi, Y., and Fomel, S., 2019, Faultseg3d: Using synthetic data sets to train an end-to-end convolutional neural network for 3d seismic fault segmentation: Geophysics, **84**, No. 3, IM35–IM45.