Supplementary materials for Population-scale Genomic Data Augmentation Based on Conditional Generative Adversarial Networks

Junjie Chen
Temple University
Philadelphia, PA, USA
junjie.chen2019@temple.edu

Mohammad Erfan Mowlaei
Temple University
Philadelphia, PA, USA
erfan.molaei@gmail.com

Xinghua Shi*
Temple University
Philadelphia, PA, USA
mindyshi@temple.edu

ACM Reference Format:
Junjie Chen, Mohammad Erfan Mowlaei, and Xinghua Shi*. 2020. Supplementary materials for Population-scale Genomic Data Augmentation Based on Conditional Generative Adversarial Networks. In *Proceedings of the 11th ACM International Conference on Bioinformatics, Computational Biology and Health Informatics (BCB '20), September 21–24, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3388440.3412475

S.1 THE VANILLA GAN MODEL

The vanilla GAN [3] consists of two neural networks, a generator (G) and a discriminator (D), shown in **Figure S.1 (a)**. G learns its distribution $\mathbb{P}_g$ over real data distribution $\mathbb{P}_r$ by mapping a random noise vector $z \sim \mathbb{P}_z$ to a sample $x \sim \mathbb{P}_g$. D receives a sample x and outputs the probability that x belongs to $\mathbb{P}_r$ rather than $\mathbb{P}_g$. G and D are trained in adversarial positions by min-maxing the value function:

$$\min_G \max_D E_{x\sim\mathbb{P}_r}[\log D(x)] + E_{z\sim\mathbb{P}_z}[\log(1 - D(G(z)))] \quad (1)$$

During training, D will be trained to maximize that probability, and G will be trained to minimize the $log(1 - D(G(z)))$. G and D compete with each other while making each other stronger at the same time until reaching the Nash equilibrium [1, 5], such that G learns to generate more and more realistic samples while D learns to accurately distinguish more and more samples.

However, population structure is one major contributing factor to human genomic data, which should be taken in consideration when augmenting or analyzing genomic data. Taking account of population labels is also important in that we can augment the genomic data by generating more data for underrepresented populations to solve the challenges of data imbalance or bias in genomics. However, a model based on vanilla GAN for genomic data augmentation can not generate data for a specific population. We thus seek for a variant GAN model called conditional GAN **Figure S.1 (b)**.

BCB '20, September 21–24, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-7964-9/20/09…$15.00
https://doi.org/10.1145/3388440.3412475

S.2 EXPERIMENT SETUP

A noise $z \sim \mathbb{P}_z$ is a 100 dimension vector drawn from a standard Gaussian distribution. Here, x is a genotype sample encoded with one-hot. Each genotype x belongs to one of the five super-populations (EUR, EAS, AFR, AMR, SAS), where these super-population labels are used as the condition information y.

The generator first accepts a super-population label as a condition, and then embeds it to the same dimension with a noise vector in order to join it with the noise input by multiplication. Consequently, two consecutive 1-dimensional CNNs with respective filter sizes of 256 and 128, each followed by Up Sampling, Batch Normalization, and Leaky ReLU activation, are applied to the reshaped tensors from the multiply layer. The final layer in the generator is a CNN with a 'softmax' activation function and a filter size equal to the number of genotype coding. Similarly, the discriminator first accepts a super-population label as a condition, and then embeds it to the same dimension with genotypes in order to join it with the genotype input by multiplication. Subsequently, two CNN layers with a stride of 2 and filter sizes of 64 and 128 are applied to the output of a reshape layer. Each CNN is followed by a Leaky ReLU activation function and a Dropout with drop probability of 0.3. Finally, the output of Dropout is flattened, and connected by a Dense layer with linear activation function as output.

It is notoriously difficult to design an effective GAN model, since the optimization of a GAN model requires a careful tuning of several hyper-parameters with numerous combinations [5, 8]. It usually requires endless experiments to improve the performance of GAN which is computational intensive and time consuming. There were a few empirical observations we identified when developing our PG-cGAN model that had considerable impact on the results. For example, as advised in [4], we trained the discriminator (for five iterations) and the generator (for one iteration) in turn at each step. For both components of our model, we used RMSProp [2] as the optimizer with a learning rate of 1e-5, which is more stable than the Adam optimizer [6]. Although many studies reported that Conv2DTranpose could produce high-quality images [7], when we test Conv1DTranpose on genotypes' 1D tensors, it performs worse than the combination of Conv1D and Upsampling layers. We also noticed that increasing the number of convolutional layers results in downgrading the performance of our model. On the human HLA genotype dataset, the best results were obtained using two CNNs as hidden layers of the generator and two CNNs as the preliminary layers of the discriminator.

S.3 TABLES

Table S.1: Number of individuals in each super-population in the 1000 Genomes Project.

Super-population	Number of individuals
AMR	504
AFR	504
EAS	504
EUR	503
SAS	489
Total	2,504

Table S.2: Layers and corresponding output shapes of the generator and discriminator in PG-cGAN.

Generator		Discriminator	
Layers	Output shape	Layers	Output shape
Input1 (population label y)	(None, 1)	Input1 (population label y)	(None, 1)
Embedding	(None, 1, 100)	Embedding	(None, 1, 28640)
Flatten	(None, 100)	Flatten1	(None, 28640)
Input2 (Noise z)	(None, 100)	Input2 (Genotypes x)	(None, 7160, 4)
Multiply (connected to Input2 and Flatten)	(None, 100)	Flatten2	(None, 28640)
Dense	(None, 458240)	Multiply (connected to Flatten1 and Flatten2)	(None, 28640)
Reshape	(None, 1790, 256)	Reshape	(None, 7160, 4)
BatchNormalization	(None, 1790, 256)	LeakyReLU	(None, 7160, 4)
LeakyReLU	(None, 1790, 256)	Dropout (0.3)	(None, 7160, 4)
Conv1D	(None, 1790, 256)	Conv1D	(None, 3580, 64)
UpSampling1D	(None, 3580, 256)	LeakyReLU	(None, 3580, 64)
BatchNormalization	(None, 3580, 256)	Dropout (0.3)	(None, 3580, 64)
LeakyReLU	(None, 3580, 256)	Conv1D	(None, 1790, 128)
Conv1D	(None, 3580, 128)	LeakyReLU	(None, 1790, 128)
UpSampling1D	(None, 7160, 128)	Dropout (0.3)	(None, 1790, 128)
BatchNormalization	(None, 7160, 128)	Flatten	(None, 229120)
LeakyReLU	(None, 7160, 128)	Dense (activation = linear)	(None, 1)
Conv1D (activation = softmax)	(None, 7160, 4)		

Table S.3: Number of individuals in each super-population in the synthetic HLA genotypes generated by our PG-cGAN model. We generate a similar amount of samples for each super-population to test our model.

Super population	# of individuals
AMR	461
AFR	502
EAS	492
EUR	551
SAS	494
sum	2,500

S.4 FIGURES

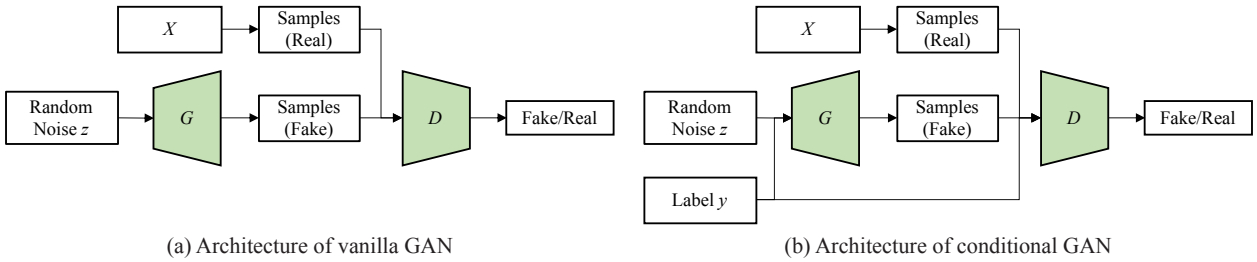

Figure S.1: Architectures of vanilla GAN and cGAN. The vanilla GAN consists of two neural networks, a generator (G) and a discriminator (D). G approximates the real data distribution, while D distinguishes the probability that one sample belongs to real data rather than fake data. G and D play a min-maxing game, trained in adversarial positions. The cGAN extends a vanilla GAN by adding auxiliary information (e.g. a class label y) as an condition on both the generator and discriminator.

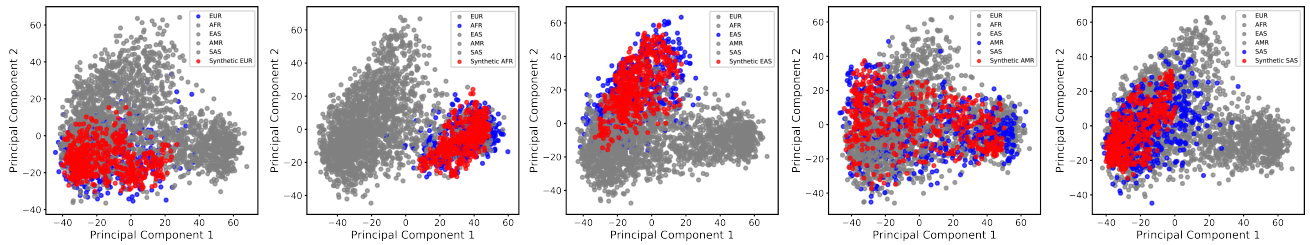

Figure S.2: PCA visualization on each super-population. Each subplot highlights one super-population of real (in blue) and corresponding synthetic (in red) HLA genotypes. Grey dots are the genotypes for the other populations.

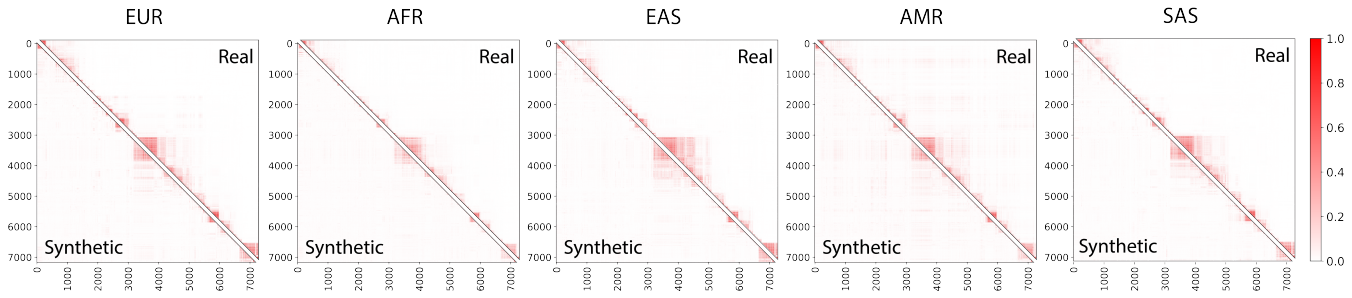

Figure S.3: LD visualization on each super population.

REFERENCES

[1] Sanjeev Arora, Rong Ge, Yingyu Liang, Tengyu Ma, and Yi Zhang. 2017. Generalization and equilibrium in generative adversarial nets (gans). In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 224–232.

[2] Yoshua Bengio. 2015. Rmsprop and equilibrated adaptive learning rates for nonconvex optimization. *corr abs/1502.04390* (2015).

[3] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in neural information processing systems*. 2672–2680.

[4] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. 2017. Improved training of wasserstein gans. In *Advances in neural information processing systems*. 5767–5777.

[5] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. 2017. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in neural information processing systems*. 6626–6637.

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

[7] Alec Radford, Luke Metz, and Soumith Chintala. 2015. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434* (2015).

[8] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. 2016. Improved techniques for training gans. In *Advances in neural information processing systems*. 2234–2242.