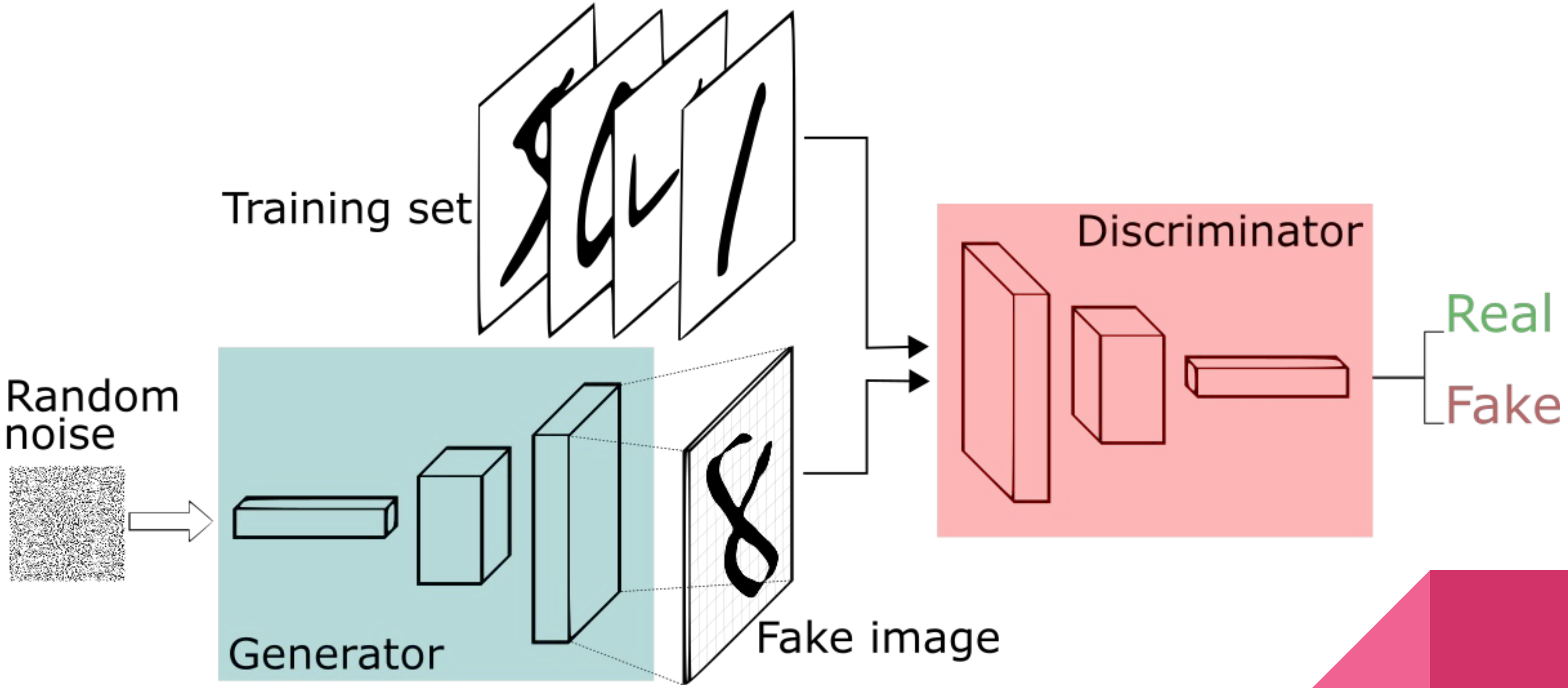
Applications of GAN on scRNA-seq data

Halil Ibrahim Bilgin
Sumeet Pal Singh

Outline

Generative adversarial networks (Goodfellow, 2014)



Taken from [deeplearning4j.org]()

Generative adversarial networks (GANs)

$$\min_G \max_D L(D,G) = \mathbb{E}_{x\sim p_r(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))]$$

The loss function of a classical GAN. (Goodfellow, 2014)

D(x): Output of discriminator -> whether the input is fake or not
G(z): Output of generator -> generates a fake output (e.g image) with a random noise prior (z)

Approximates $JSD(p_{\text{data}} \| p_g)$ in ideal conditions.

Wasserstein GANs (Arjovsky, 2017)

$$L(p_r, p_g) = W(p_r, p_g) = \max_{w \in W} \mathbb{E}_{x \sim p_r}[f_w(x)] - \mathbb{E}_{z \sim p_r(z)}[f_w(g_\theta(z))]$$

Uses Wasserstein Distance (Earth Mover's distance) as a GAN loss function. (Arjovsky, 2017)

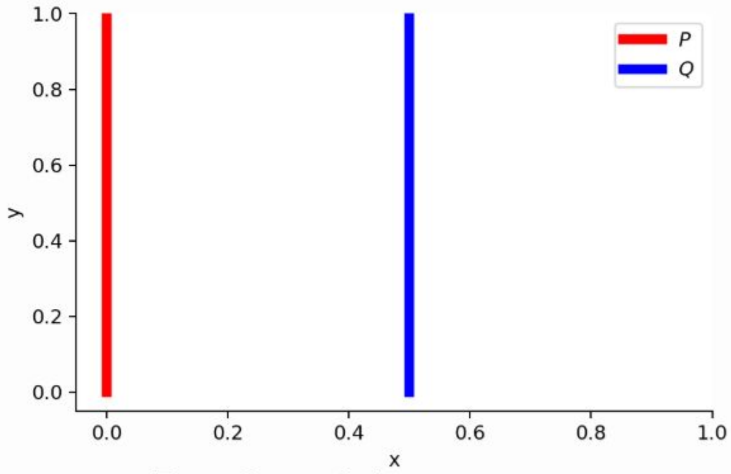
Author shows that wasserstein metric is continuous and smooth even two distributions are disjoint. In contrast, KL gives infinity and JS might have sudden jumps and not differentiable at some points.

A continuous and smooth loss function is important for stable learning process.

Wasserstein GANs (Arjovsky, 2017)

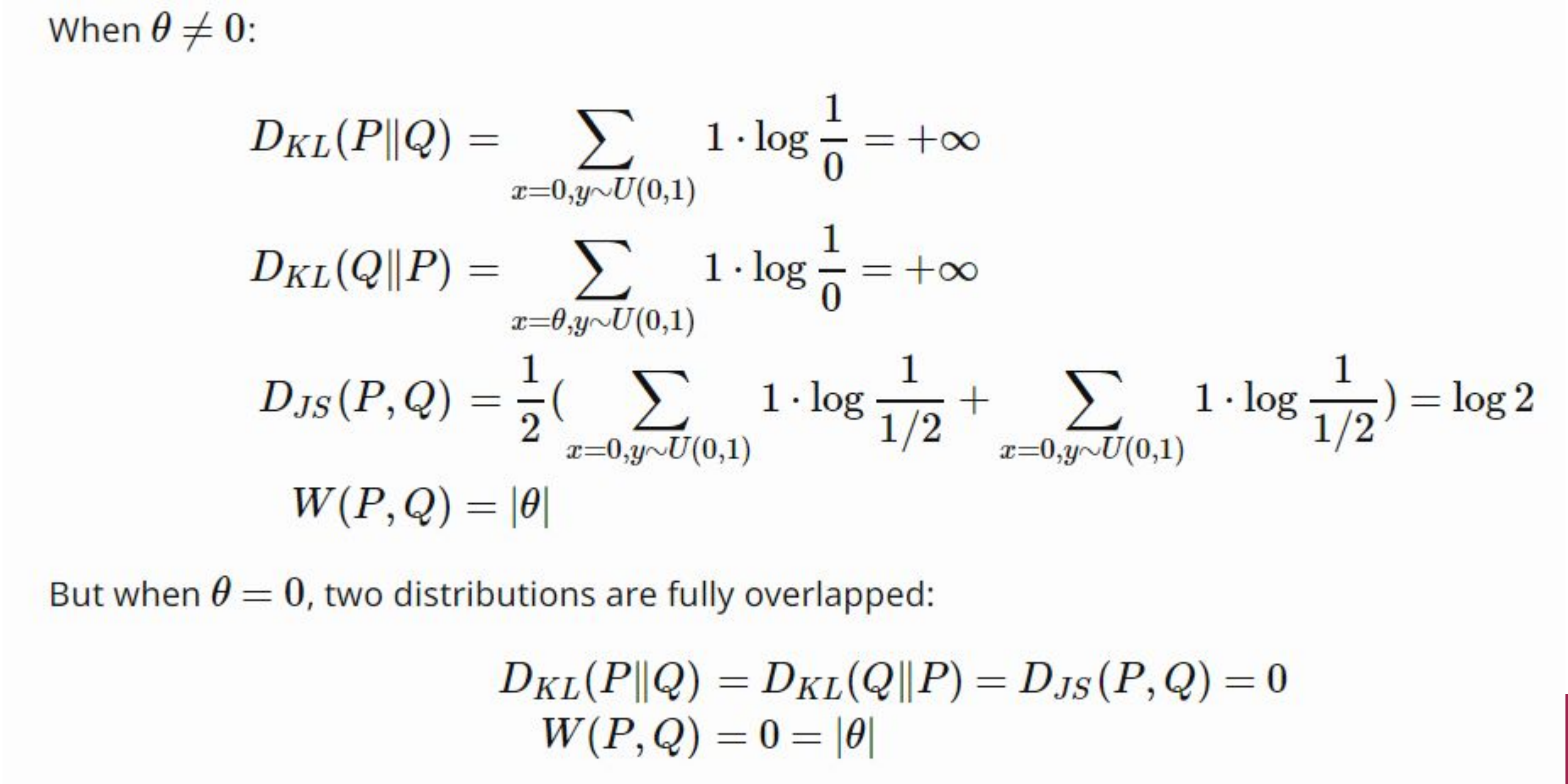
Suppose we have two probability distributions, P and Q.

$$\forall (x, y) \in P, x = 0 \text{ and } y \sim U(0,1)$$
$$\forall (x, y) \in Q, x = \theta, 0 \leq \theta \leq 1 \text{ and } y \sim U(0,1)$$



There is no overlap between P and Q when $\theta \neq 0$.

Taken from [From GAN to WGAN]



When $\theta \neq 0$:

$$D_{KL}(P\|Q) = \sum_{x=0, y\sim U(0,1)} 1 \cdot \log \frac{1}{0} = +\infty$$

$$D_{KL}(Q\|P) = \sum_{x=\theta, y\sim U(0,1)} 1 \cdot \log \frac{1}{0} = +\infty$$

$$D_{JS}(P, Q) = \frac{1}{2}\left(\sum_{x=0, y\sim U(0,1)} 1 \cdot \log \frac{1}{1/2} + \sum_{x=0, y\sim U(0,1)} 1 \cdot \log \frac{1}{1/2}\right) = \log 2$$

$$W(P, Q) = |\theta|$$

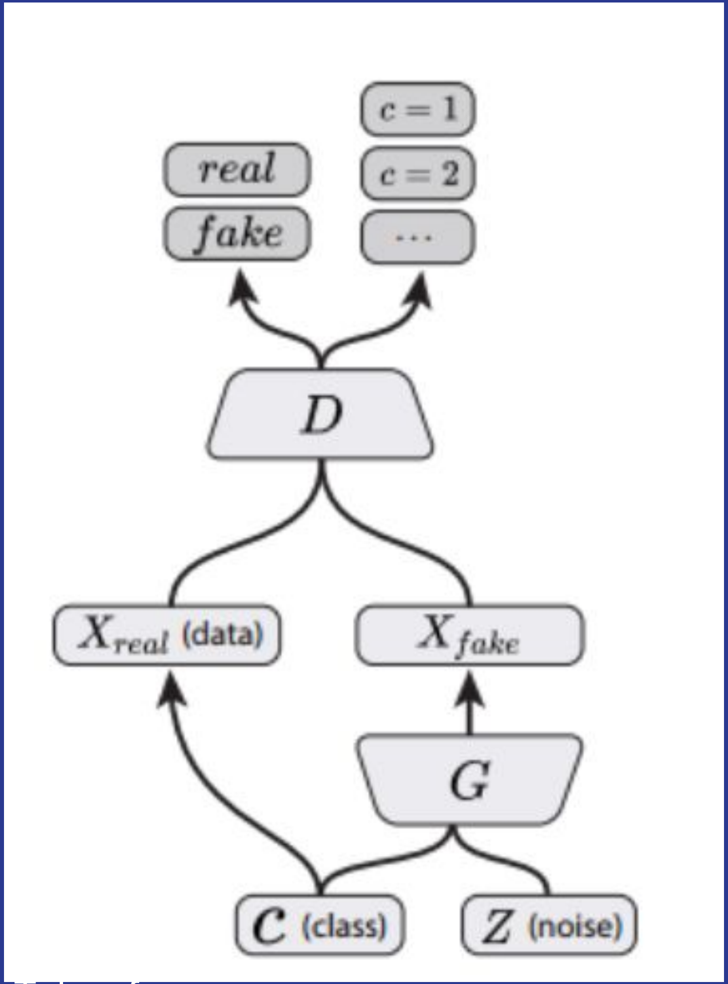
But when $\theta = 0$, two distributions are fully overlapped:

$$D_{KL}(P\|Q) = D_{KL}(Q\|P) = D_{JS}(P, Q) = 0$$
$$W(P, Q) = 0 = |\theta|$$

Taken from From GAN to WGAN

GANs with auxiliary classifier (ACGAN)
(Odena, 2016)

+ Makes use of class labels of samples
+ Generator can generate conditional output



Taken from
Github.com/ACGAN-PyTorch

GANs with auxiliary classifier (ACGAN) (Odena, 2016)

$$L_S = E[\log P(S = real \mid X_{real})] + E[\log P(S = fake \mid X_{fake})]$$

$$L_C = E[\log P(C = c \mid X_{real})] + E[\log P(C = c \mid X_{fake})]$$

D is trained to maximize $L_S + L_C$ while G is trained to maximize $L_C - L_S$ (Odena, 2016).

Can be used with Wasserstein loss as well.

Configuration of Generative Adversarial Networks

- Architecture (of discriminator and generator)
  - Activation function
  - Number of hidden layers
  - Number of nodes in each hidden layer
- Optimizer (e.g Adam, RMSProp)
- Learning rate
- Mini-batch size
- Noise
  - Dropouts,
  - Label noise
- Wasserstein GAN, or classical GAN?
- Scale the data or not?

High dimensional hyperparameters space

Challenges in training GANs

- Training is not stable. (Early stopping is not feasible)
- The balance between Generator and Discriminator should be maintained for training to converge.
- Mode collapse is quite typical.
  - Generator produces a limited diversity of samples or sometimes one single sample regardless of input z.
  - Might be related to loss function, unbalance in the learning and etc.
- Need to log generated samples over periods to keep track the failures
- There are lots of hyperparameters to try.

Single cell RNA sequencing

scRNA-seq is a new technology, first publication by (Tang et al. 2009). Gained widespread popularity after ~2014.

Computes each gene’s distribution of expression levels across a population of cells.

Allows to study new biological questions in which cell-specific changes in transcriptome are important, e.g. cell type identification, heterogeneity of cell responses, stochasticity of gene expression, inference of gene regulatory networks across the cells.

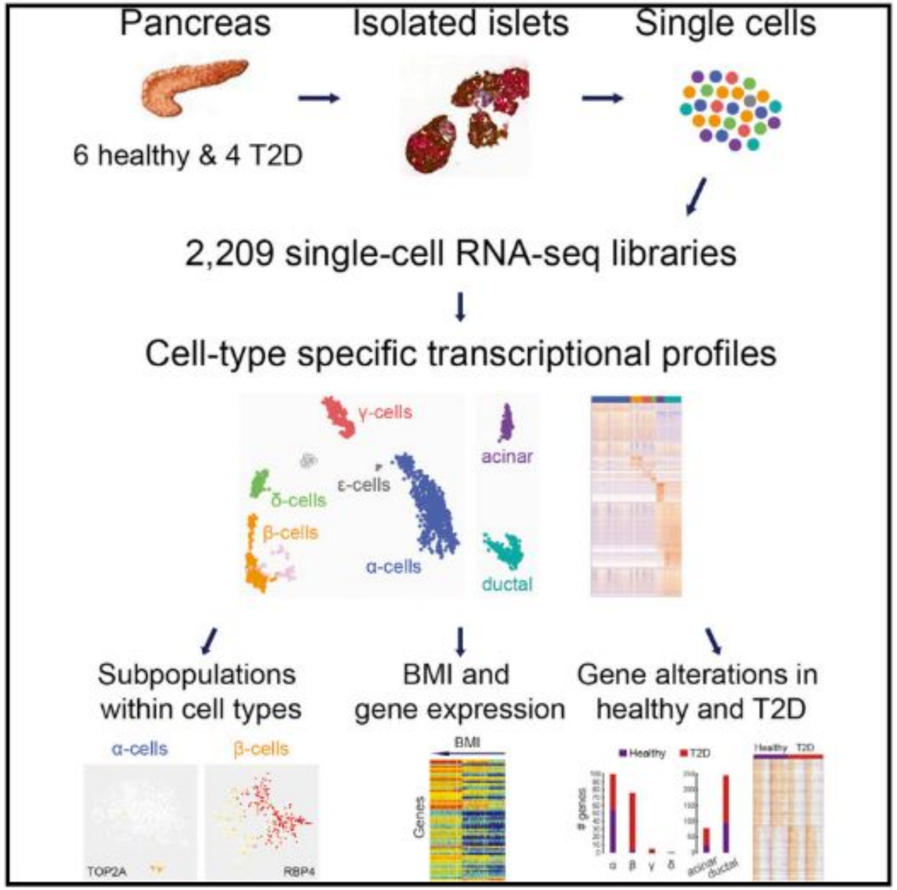
Computational challenges

- Each sample in scRNA-seq dataset contains large number of features (5000-25000).
- Statistical analysis (e.g differential gene expression analysis) for scRNA-seq data might require large number of samples for obtaining reasonable statistical power.
- Machine learning models might need larger training set due to the large number of dimensions to be trained.
- scRNA-seq costs are min. $653 per 10,000 cells (Macosko et al., 2015)

scRNA-seq for Pancreatic cells

GANs would be useful for:

- Identifying gene alterations across samples from healthy and Type-2 diabetic (T2D) subjects.
- Associating diabetes with known genes that cause different diseases (e.g obesity)
- Improve the accuracy of machine learning models designed for scRNA-seq data classification (e.g a cell is a coming from T2D or not)



A workflow of scRNA-seq (Segerstolpe, 2016)

Collecting and preprocessing the data

Collected the raw data published in the studies: Segerstolpe et al. (2016), Xin et al. (2016), Lawlor et al. (2017) and Wang et al. (2016), which consist of pancreatic cells of healthy people and t2d patients.

Processed the raw data (FastQC, Trimming reads, Demultiplexing, Aligning reads, Construction of expression matrix)

Did batch correction to avoid batch effects, which occur because measurements are affected by laboratory conditions, reagent lots, and personnel differences Using the method proposed in Haghverdi et al. (2017)

Collecting and preprocessing the data

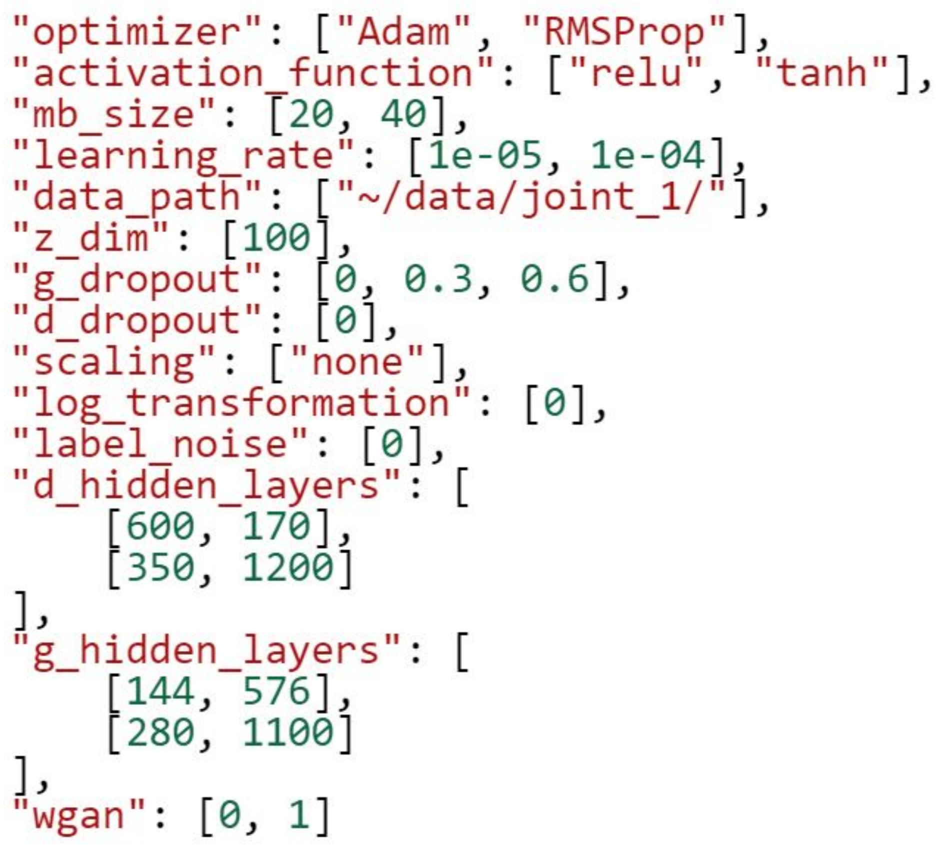
Constructed datasets using most varied 500, 1500 and 5000 genes, respectively and samples only from alpha and beta cells since other cells don’t have enough samples.

Totally ~3500 samples with 4 different classes (alpha_healthy, beta_healthy, alpha_t2d and beta_t2d).

A repository for training GANs

Scripts that automatically

- Create experiments for given configuration file
- Save the generated data and model params. over periods
- Do analysis and save the plots



```
"optimizer": ["Adam", "RMSProp"],
"activation_function": ["relu", "tanh"],
"mb_size": [20, 40],
"learning_rate": [1e-05, 1e-04],
"data_path": ["~/data/joint_1/"],
"z_dim": [100],
"g_dropout": [0, 0.3, 0.6],
"d_dropout": [0],
"scaling": ["none"],
"log_transformation": [0],
"label_noise": [0],
"d_hidden_layers": [
    [600, 170],
    [350, 1200]
],
"g_hidden_layers": [
    [144, 576],
    [280, 1100]
],
"wgan": [0, 1]
```

Evaluation of GANs

- Evaluating the marker gene expression:

  Beta-cell index = non-beta INS / beta INS

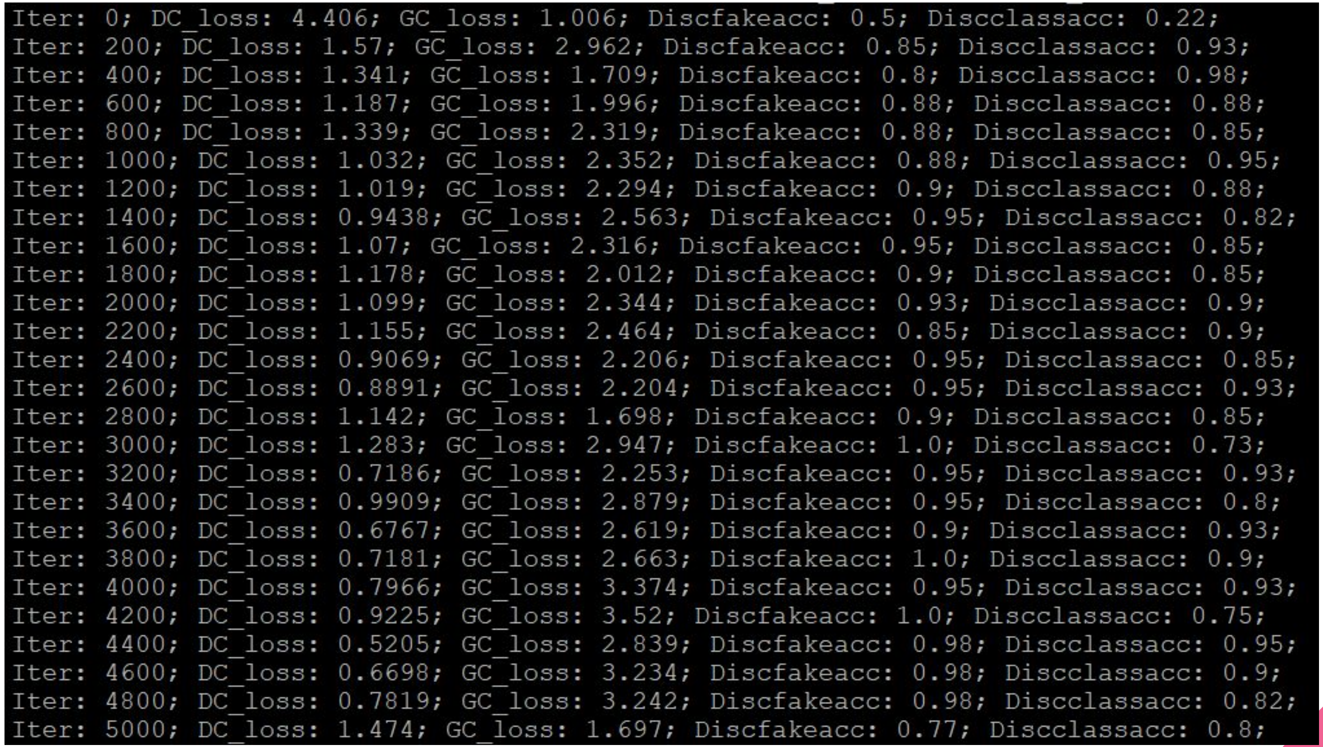
  Alpha-cell index = non-alpha GCG / alpha GCG

- Visualization

  PCA

  Marker expression over epochs

What about using loss in evaluation ?



```
Iter: 0; DC_loss: 4.406; GC_loss: 1.006; Discfakeacc: 0.5; Discclassacc: 0.22;
Iter: 200; DC_loss: 1.57; GC_loss: 2.962; Discfakeacc: 0.85; Discclassacc: 0.93;
Iter: 400; DC_loss: 1.341; GC_loss: 1.709; Discfakeacc: 0.8; Discclassacc: 0.98;
Iter: 600; DC_loss: 1.187; GC_loss: 1.996; Discfakeacc: 0.88; Discclassacc: 0.88;
Iter: 800; DC_loss: 1.339; GC_loss: 2.319; Discfakeacc: 0.88; Discclassacc: 0.85;
Iter: 1000; DC_loss: 1.032; GC_loss: 2.352; Discfakeacc: 0.88; Discclassacc: 0.95;
Iter: 1200; DC_loss: 1.019; GC_loss: 2.294; Discfakeacc: 0.9; Discclassacc: 0.88;
Iter: 1400; DC_loss: 0.9438; GC_loss: 2.563; Discfakeacc: 0.95; Discclassacc: 0.82;
Iter: 1600; DC_loss: 1.07; GC_loss: 2.316; Discfakeacc: 0.95; Discclassacc: 0.85;
Iter: 1800; DC_loss: 1.178; GC_loss: 2.012; Discfakeacc: 0.9; Discclassacc: 0.85;
Iter: 2000; DC_loss: 1.099; GC_loss: 2.344; Discfakeacc: 0.93; Discclassacc: 0.9;
Iter: 2200; DC_loss: 1.155; GC_loss: 2.464; Discfakeacc: 0.85; Discclassacc: 0.9;
Iter: 2400; DC_loss: 0.9069; GC_loss: 2.206; Discfakeacc: 0.95; Discclassacc: 0.85;
Iter: 2600; DC_loss: 0.8891; GC_loss: 2.204; Discfakeacc: 0.95; Discclassacc: 0.93;
Iter: 2800; DC_loss: 1.142; GC_loss: 1.698; Discfakeacc: 0.9; Discclassacc: 0.85;
Iter: 3000; DC_loss: 1.283; GC_loss: 2.947; Discfakeacc: 1.0; Discclassacc: 0.73;
Iter: 3200; DC_loss: 0.7186; GC_loss: 2.253; Discfakeacc: 0.95; Discclassacc: 0.93;
Iter: 3400; DC_loss: 0.9909; GC_loss: 2.879; Discfakeacc: 0.95; Discclassacc: 0.8;
Iter: 3600; DC_loss: 0.6767; GC_loss: 2.619; Discfakeacc: 0.9; Discclassacc: 0.93;
Iter: 3800; DC_loss: 0.7181; GC_loss: 2.663; Discfakeacc: 1.0; Discclassacc: 0.9;
Iter: 4000; DC_loss: 0.7966; GC_loss: 3.374; Discfakeacc: 0.95; Discclassacc: 0.93;
Iter: 4200; DC_loss: 0.9225; GC_loss: 3.52; Discfakeacc: 1.0; Discclassacc: 0.75;
Iter: 4400; DC_loss: 0.5205; GC_loss: 2.839; Discfakeacc: 0.98; Discclassacc: 0.95;
Iter: 4600; DC_loss: 0.6698; GC_loss: 3.234; Discfakeacc: 0.98; Discclassacc: 0.9;
Iter: 4800; DC_loss: 0.7819; GC_loss: 3.242; Discfakeacc: 0.98; Discclassacc: 0.82;
Iter: 5000; DC_loss: 1.474; GC_loss: 1.697; Discfakeacc: 0.77; Discclassacc: 0.8;
```

Best trial for one layer joint experiment in page 25

Discriminator Loss appears to be reduce over time from 4 to 0.5

Generator is stable

Running Experiments

- Use AC-GAN with WGAN and GAN loss functions
- Run every experiment 4 times with specifying same 4 seeds in all experiments for:
  - reproducibility
  - one model is less likely to perform better due to the order of training samples
- Train 30 epochs, Save 500 generated samples every 5 epochs
- Calculate index scores and plots using all the generated samples produced in 4 different run of the same experiment.

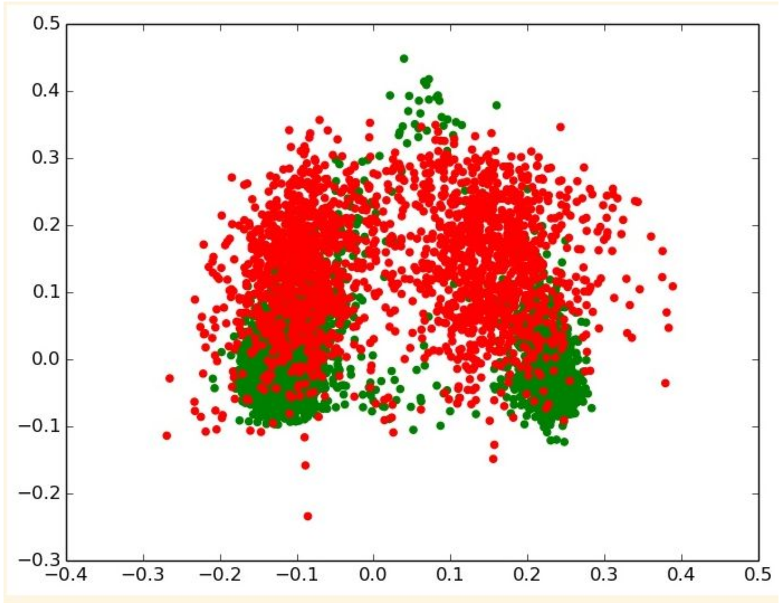
Experiments - Best optimizer

Tried Adam, Adagrad, Adadelta, Ftrl, GradientDescent and RMSProp for one and two layer arch.

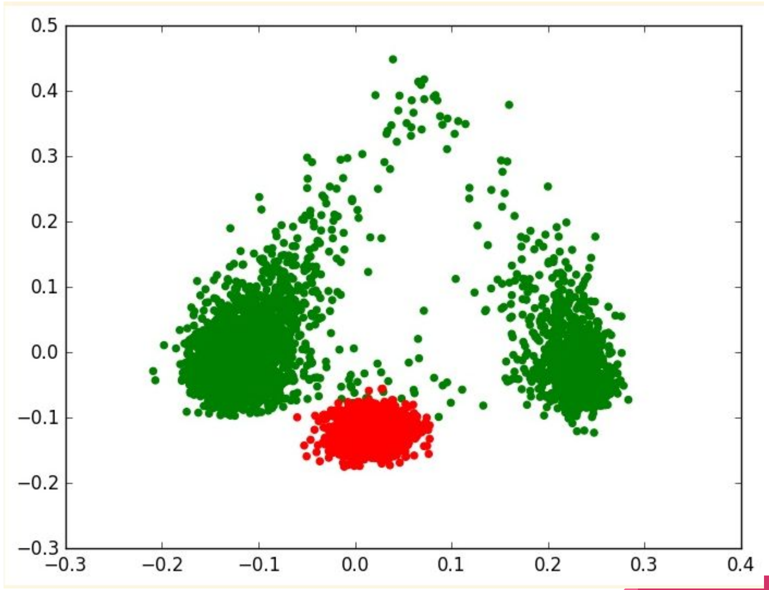
meanindexscores	**adam**	adagrad	adadelta	ftrl	GradientDescent	**rmsprop**
1layer-gan	**0.0524**	0.671	0.701	0.705	0.33	0.148
1layer-wgan	0.18	0.677	0.692	0.70	0.35	**0.079**
2layer-gan	**0.02**	0.656	0.66	0.705	0.923	0.035
2layer-wgan	0.03	0.666	0.663	0.71	0.895	**0.02**

Adam and RMSProp works best.

Experiments - Best optimizer

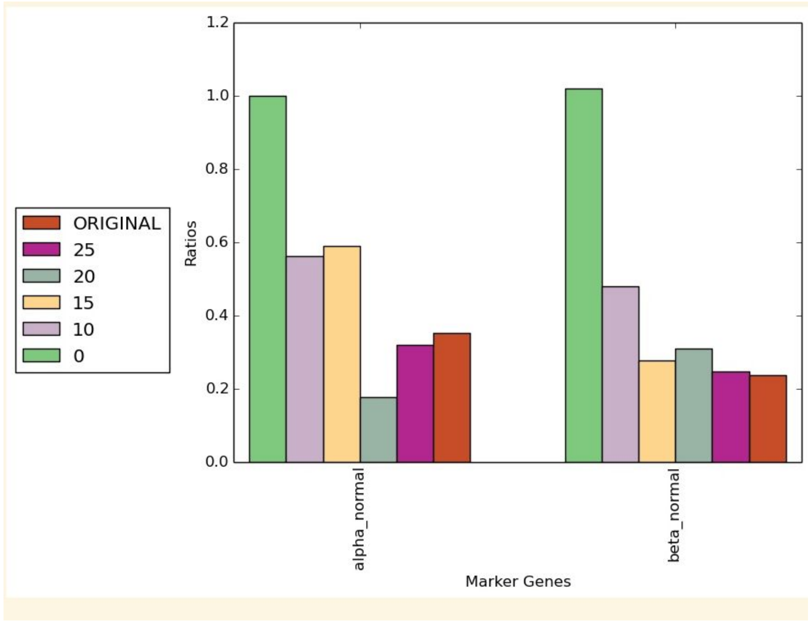


Good model -> Adam two layer

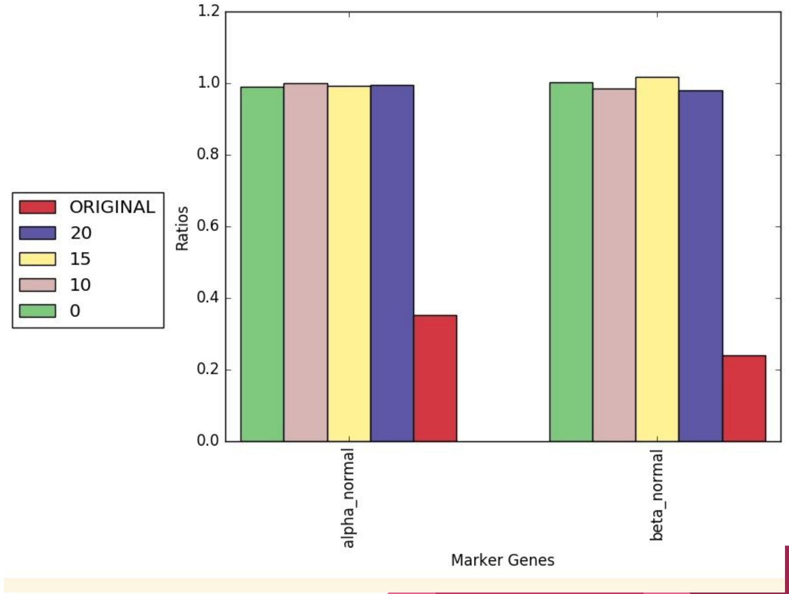


Bad model -> Adadelta one layer

Experiments - Best optimizer

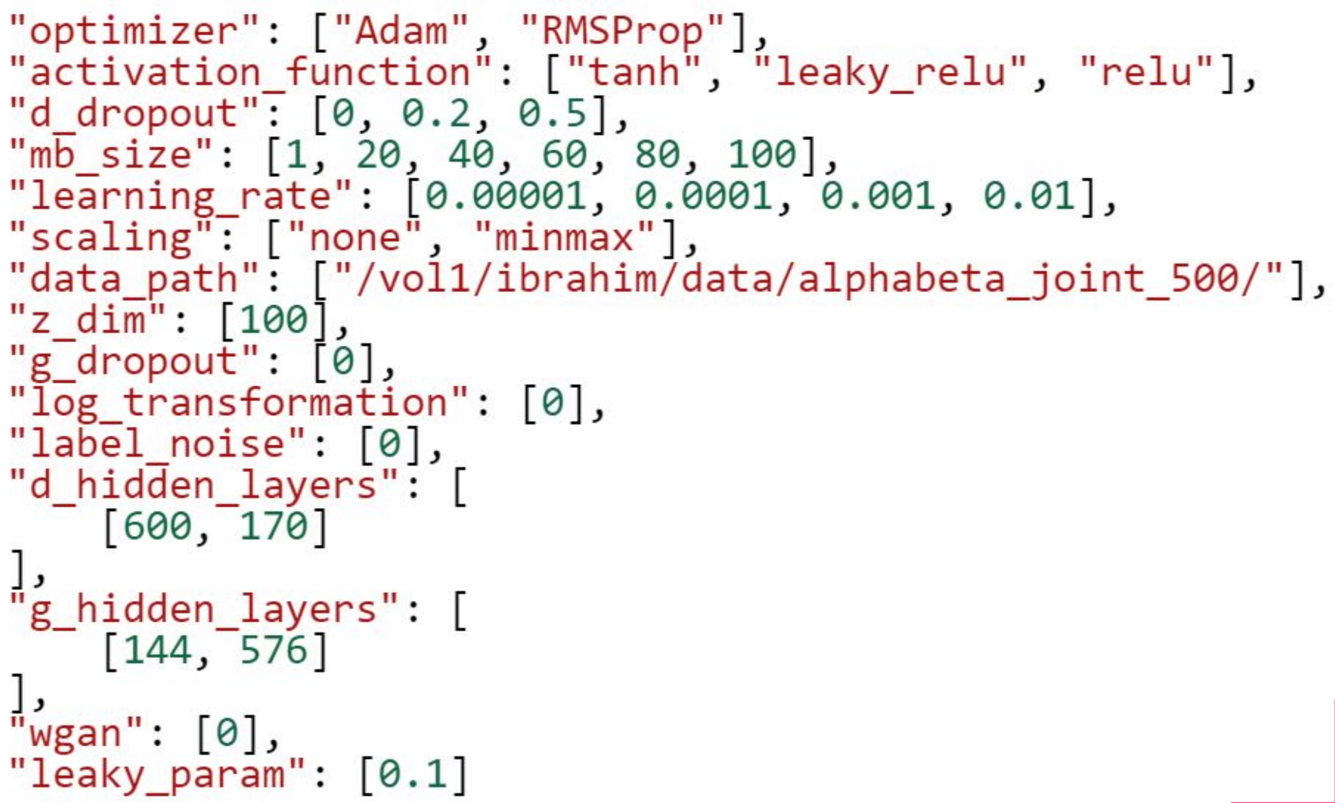


Good model -> Adam two layer



Bad model -> Adagrad one layer

Experiments - Scaling Minibatch size & activation



```
"optimizer": ["Adam", "RMSProp"],
"activation_function": ["tanh", "leaky_relu", "relu"],
"d_dropout": [0, 0.2, 0.5],
"mb_size": [1, 20, 40, 60, 80, 100],
"learning_rate": [0.00001, 0.0001, 0.001, 0.01],
"scaling": ["none", "minmax"],
"data_path": ["/vol1/ibrahim/data/alphabeta_joint_500/"],
"z_dim": [100],
"g_dropout": [0],
"log_transformation": [0],
"label_noise": [0],
"d_hidden_layers": [
    [600, 170]
],
"g_hidden_layers": [
    [144, 576]
],
"wgan": [0],
"leaky_param": [0.1]
```

Experiments - Scaling Minibatch size & activation

Experiment with min-max scaling (-1,1) & tanh in the output of generator

optimizer	mb_size	learning_rate	activation(in hidden layers)	meanindexscore	Best epoch
Adam	60	0.001	tanh	**0.02**	25
RMSProp	40	0.00001	tanh	0.033	20
RMSProp	40	0.00001	tanh	0.037	30
RMSProp	60	0.00001	tanh	0.041	30

Best 4 experiments
Mini batch size 60 works best.
Tanh is the best

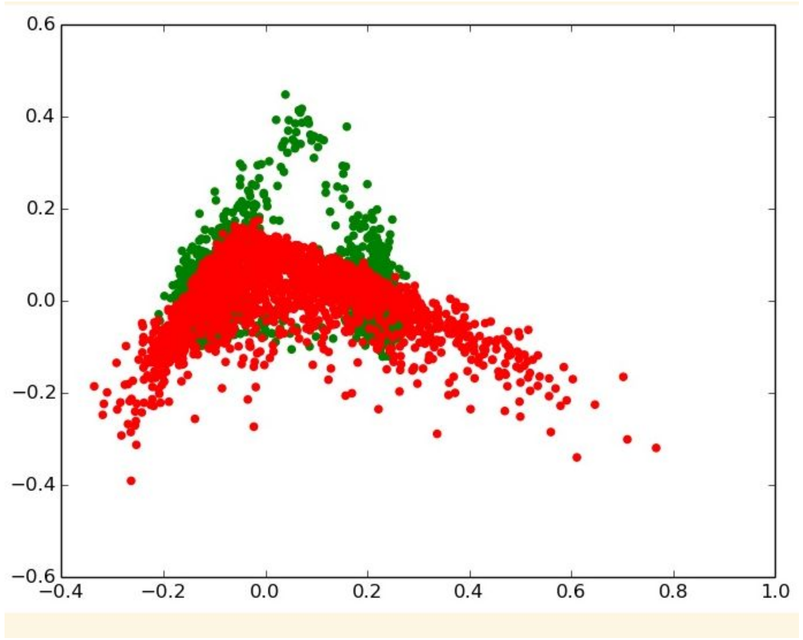
Experiments - Scaling Minibatch size & activation

Experiment with no input scaling

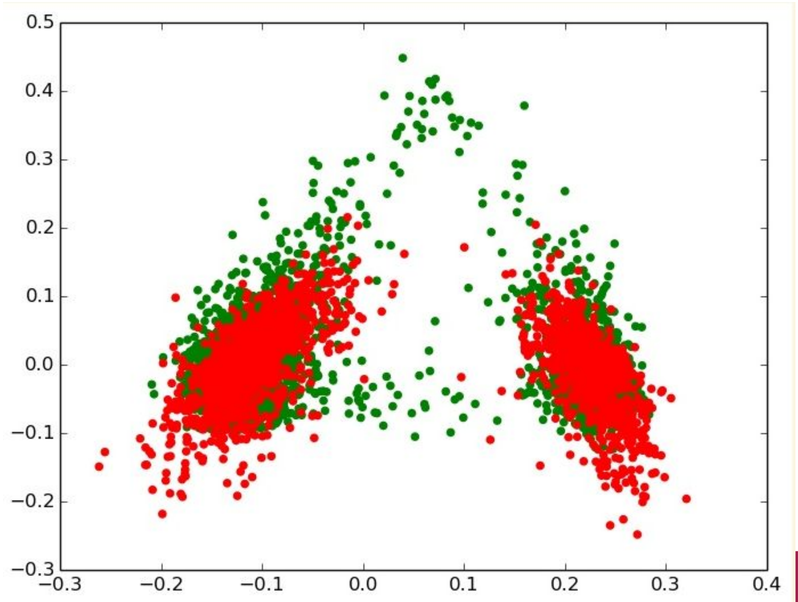
optimizer	mb_size	learning_rate	activation(in hidden layers)	meanindexscore	Best epoch
RMSProp	60	0.01	relu	**0.0005**	10
Adam	40	0.00001	leaky_relu	0.002	30
RMSProp	20	0.00001	leaky_relu	0.004	15
RMSProp	40	0.00001	relu	0.004	20

Best 4 experiments
Mini batch size 60 works best.
Relu and leaky_relu works best

Experiments - Scaling Minibatch size & activation

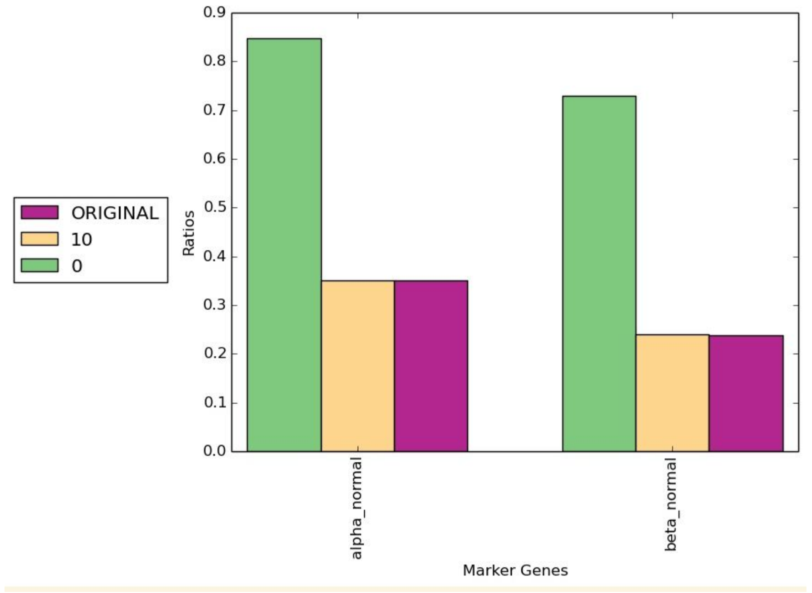


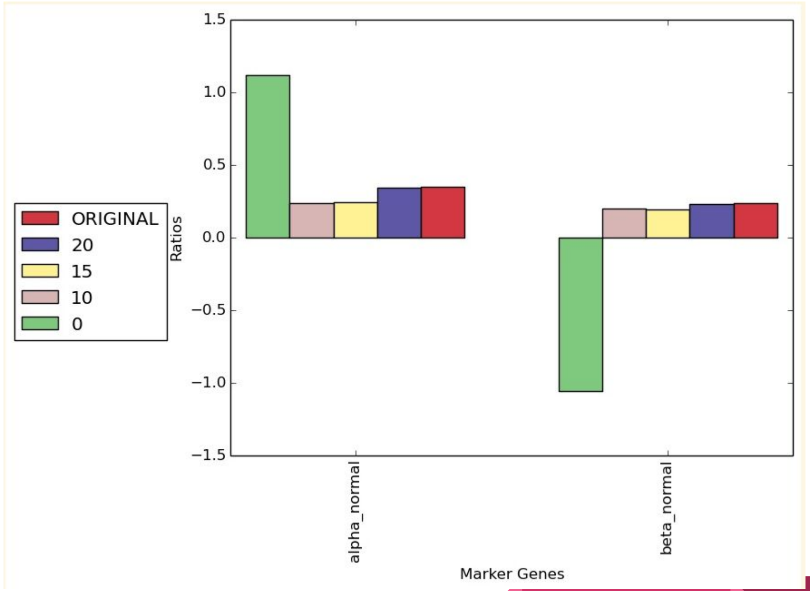
Best index score - not very plausible



4th best index score - better

Experiments - Scaling Minibatch size & activation





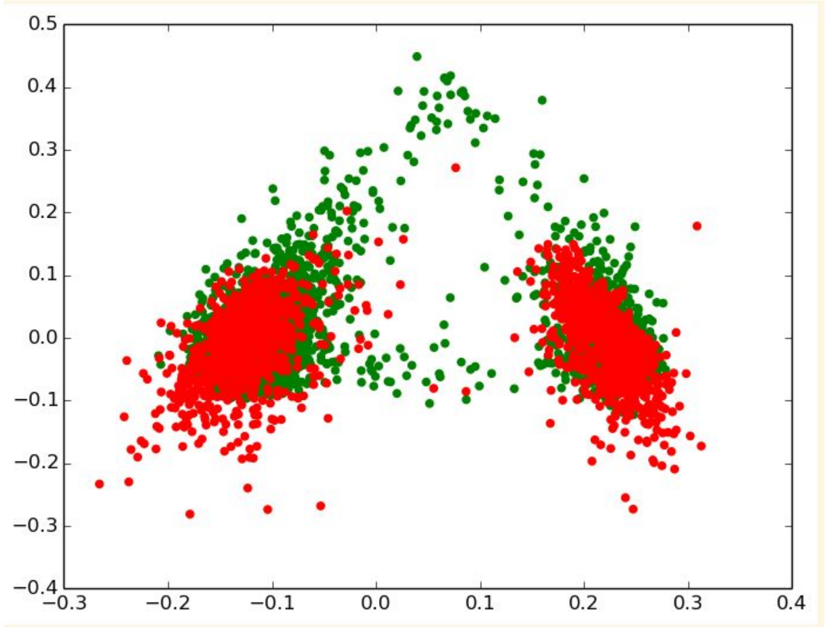
Best index score - not very plausible

4th best index score

Experiments - Scaling Minibatch size & activation

- The algorithm chooses the epoch of which generated samples that performs the best score over 30 epochs.
- PCA of the experiment with the best index score is plotted using the samples of 10th epoch.
- Even though the model learned the marker gene expression in 10 epochs, it apparently could not learn all expressions.

Experiments - Scaling Minibatch size & activation



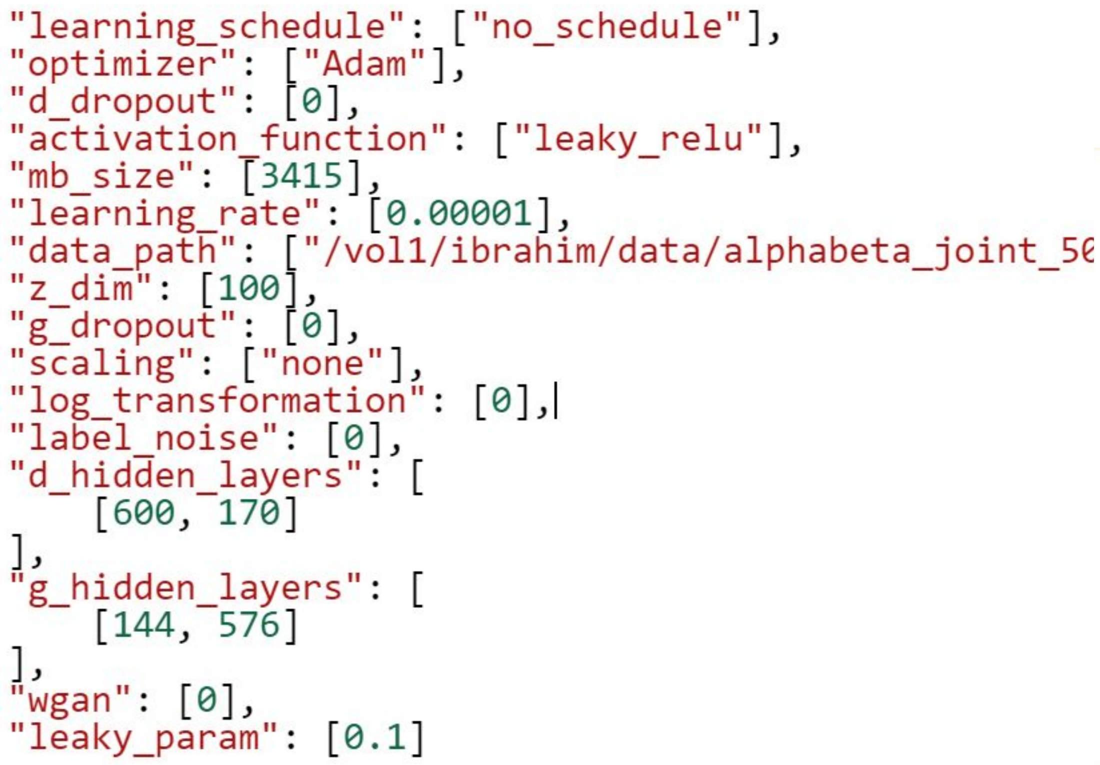
Plot of the samples drawn at 30th epoch for the experiment with the best index score in 10th epoch

More plausible :)

Finding

Even though index score is useful, it might be better to evaluate the PCA plots drawn with the generated samples of later epochs

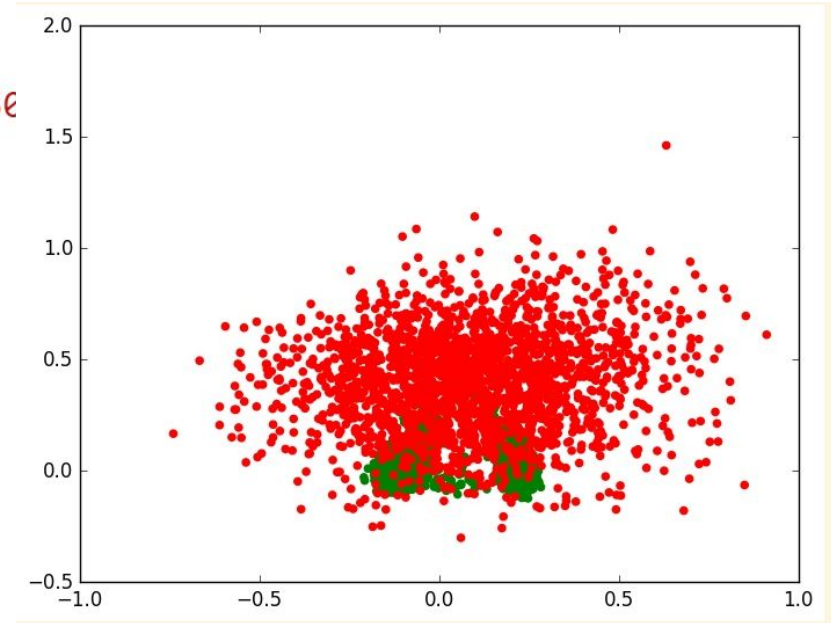
Experiments - Batch learning



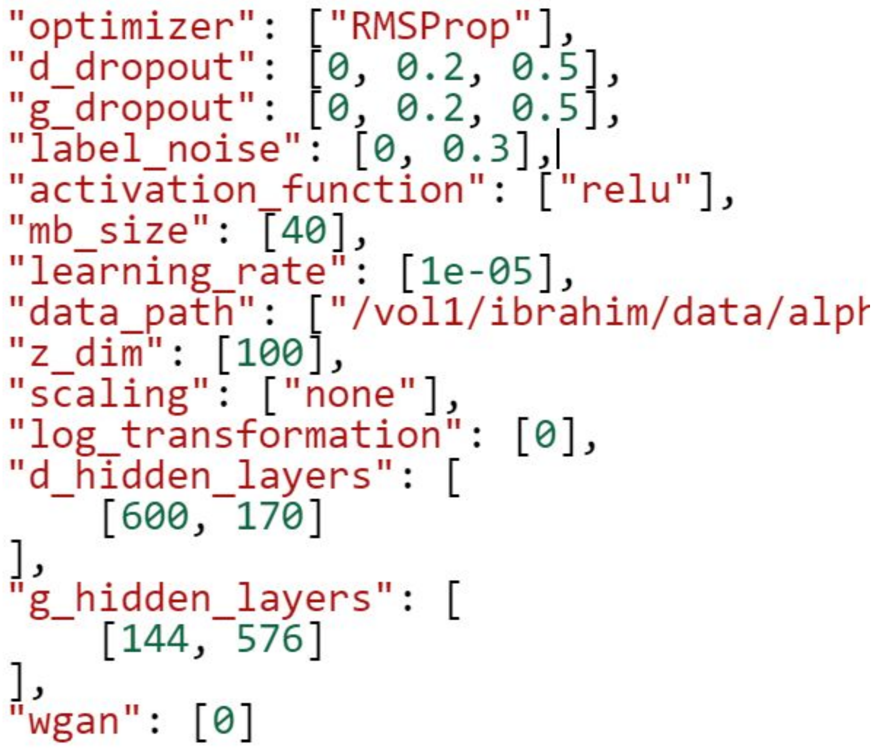
```
"learning_schedule": ["no_schedule"],
"optimizer": ["Adam"],
"d_dropout": [0],
"activation_function": ["leaky_relu"],
"mb_size": [3415],
"learning_rate": [0.00001],
"data_path": ["/vol1/ibrahim/data/alphabeta_joint_5
"z_dim": [100],
"g_dropout": [0],
"scaling": ["none"],
"log_transformation": [0],
"label_noise": [0],
"d_hidden_layers": [
    [600, 170]
],
"g_hidden_layers": [
    [144, 576]
],
"wgan": [0],
"leaky_param": [0.1]
```

Index score 0.123 -> Not good

PCA



Experiments - Regularization(Noise)

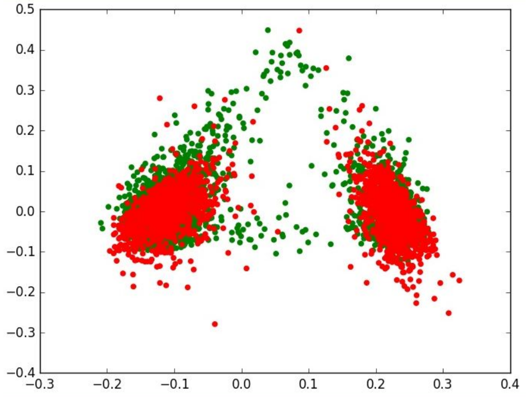


```
"optimizer": ["RMSProp"],
"d_dropout": [0, 0.2, 0.5],
"g_dropout": [0, 0.2, 0.5],
"label_noise": [0, 0.3],
"activation_function": ["relu"],
"mb_size": [40],
"learning_rate": [1e-05],
"data_path": ["/vol1/ibrahim/data/alph
"z_dim": [100],
"scaling": ["none"],
"log_transformation": [0],
"d_hidden_layers": [
    [600, 170]
],
"g_hidden_layers": [
    [144, 576]
],
"wgan": [0]
```

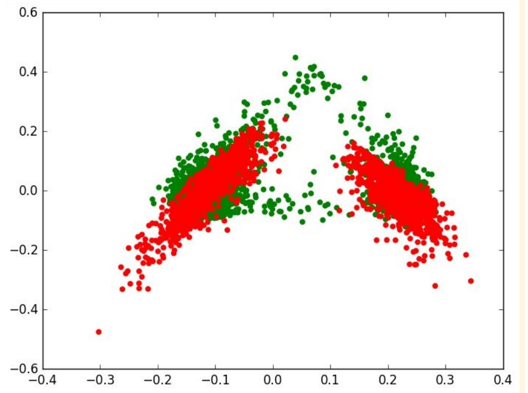
g_dropout	d_dropout	label_noise	meanindexscore	best_epoch
0	0	0	0.009	25
0	0.5	0	0.012	20
0	0	0.3	0.016	20
0	0.5	0.3	0.021	30

Table shows best 4 setting.
There appears no big difference
Regularization might not be the main bottleneck

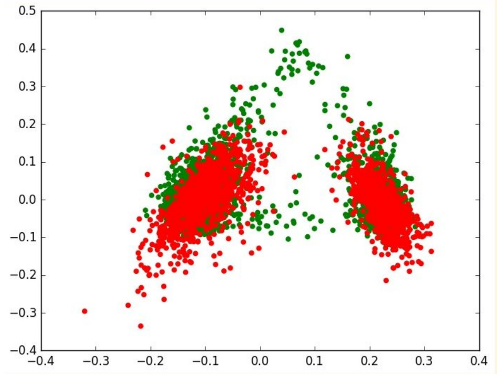
Experiments - Regularization(Noise)



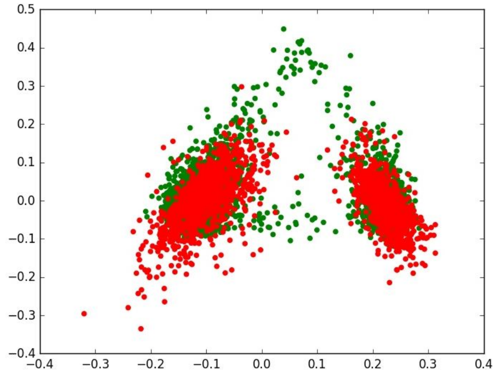
best



second

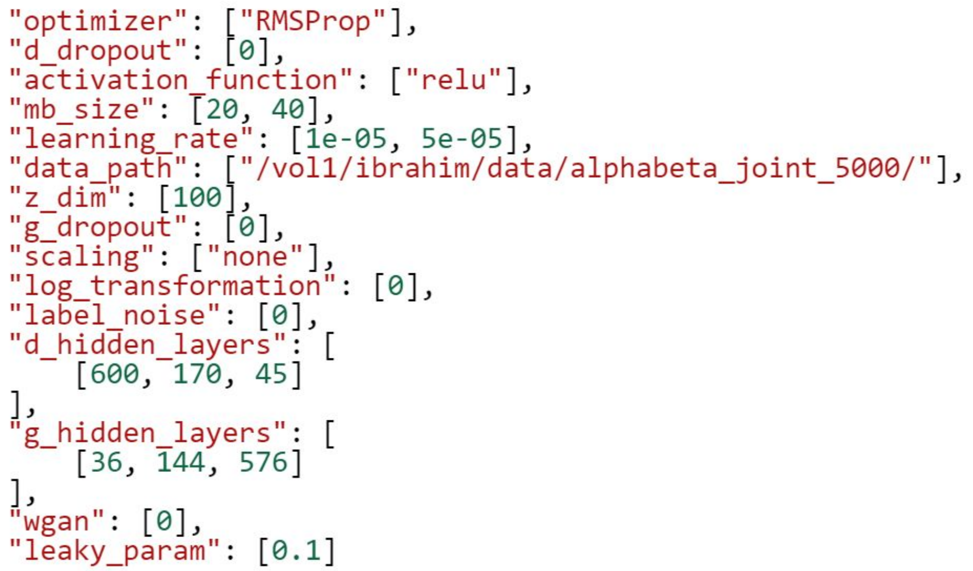


third



fourth

Experiments - 3 layers



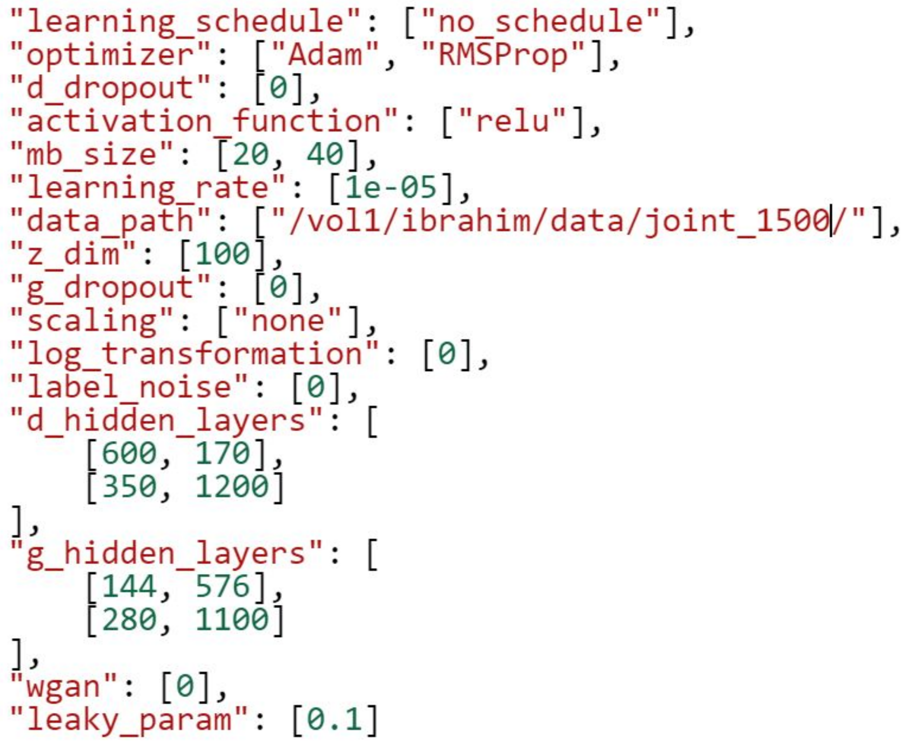
```
"optimizer": ["RMSProp"],
"d_dropout": [0],
"activation_function": ["relu"],
"mb_size": [20, 40],
"learning_rate": [1e-05, 5e-05],
"data_path": ["/vol1/ibrahim/data/alphabeta_joint_5000/"],
"z_dim": [100],
"g_dropout": [0],
"scaling": ["none"],
"log_transformation": [0],
"label_noise": [0],
"d_hidden_layers": [
    [600, 170, 45]
],
"g_hidden_layers": [
    [36, 144, 576]
],
"wgan": [0],
"leaky_param": [0.1]
```

Best score -> 0.125

Not working well

Experiment - Generating more genes (1500)

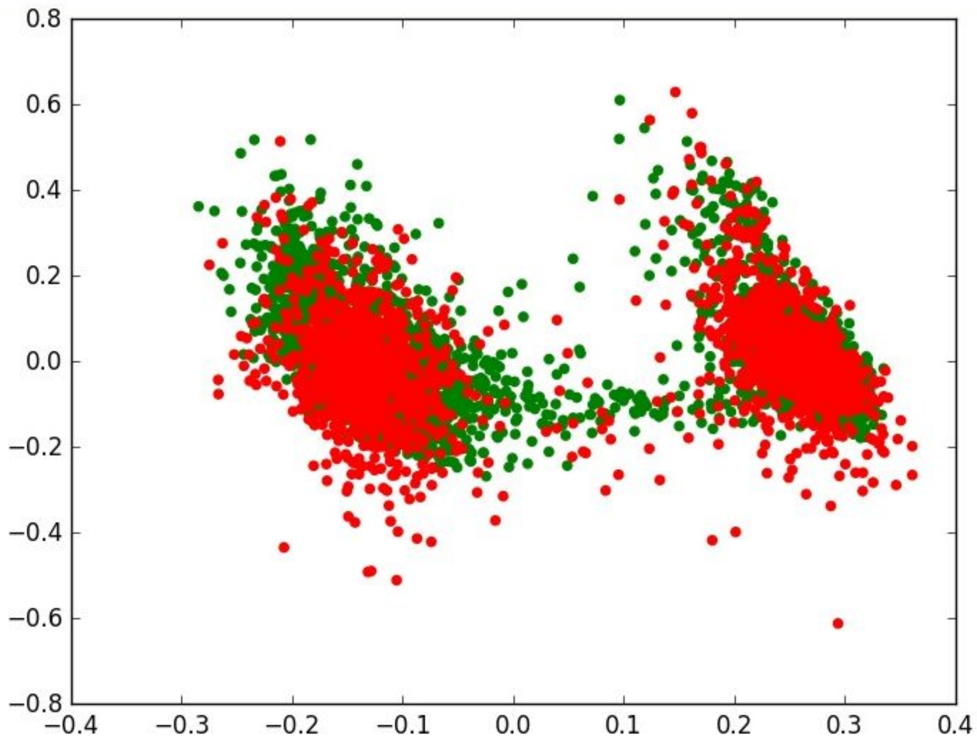
Run experiments for 1500 genes using the hyperparams that worked well for the most variable 500 genes



```
"learning_schedule": ["no_schedule"],
"optimizer": ["Adam", "RMSProp"],
"d_dropout": [0],
"activation_function": ["relu"],
"mb_size": [20, 40],
"learning_rate": [1e-05],
"data_path": ["/vol1/ibrahim/data/joint_1500/"],
"z_dim": [100],
"g_dropout": [0],
"scaling": ["none"],
"log_transformation": [0],
"label_noise": [0],
"d_hidden_layers": [
    [600, 170],
    [350, 1200]
],
"g_hidden_layers": [
    [144, 576],
    [280, 1100]
],
"wgan": [0],
"leaky_param": [0.1]
```

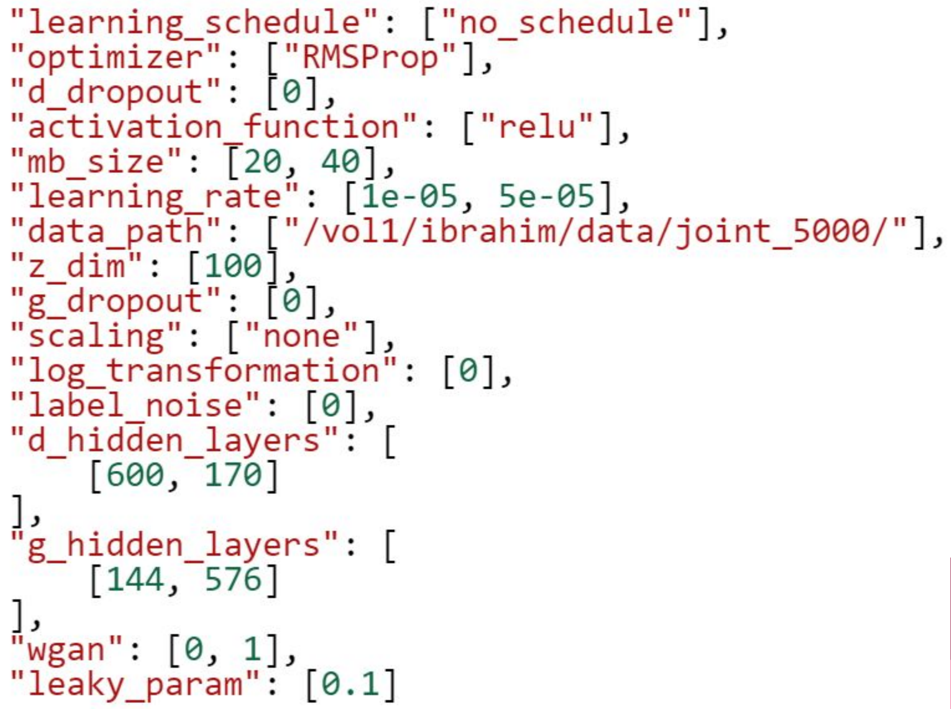
Experiment - Generating more genes (1500)

Best index score: 0.004



Experiment - Generating more genes (5000)

Run experiments for 5000 genes using the hyperparams that worked best for the most variable 1500 genes



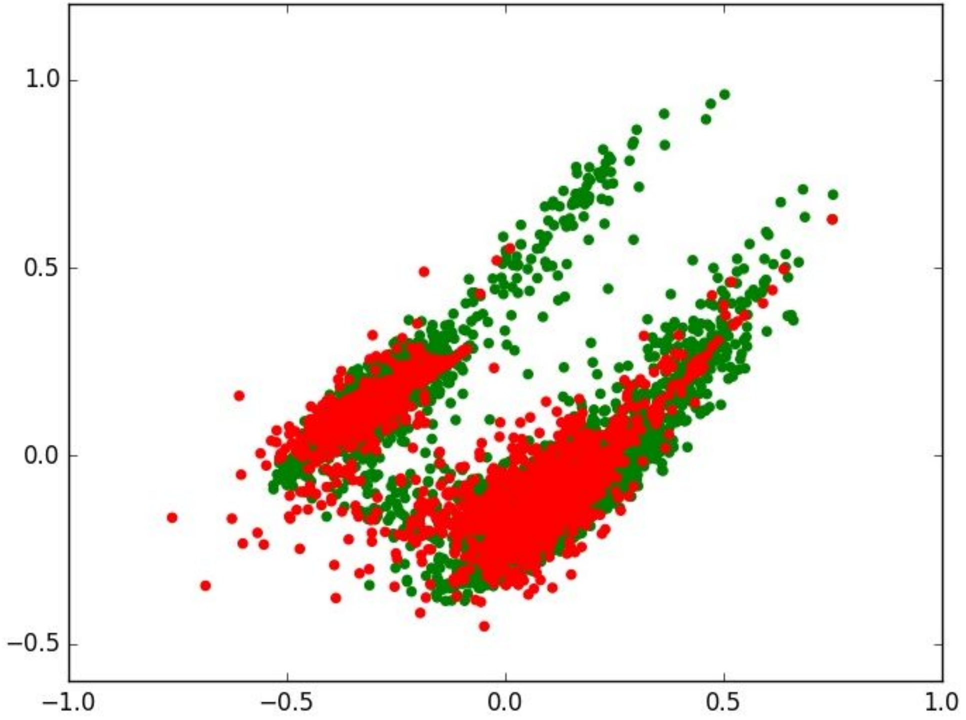
```
"learning_schedule": ["no_schedule"],
"optimizer": ["RMSProp"],
"d_dropout": [0],
"activation_function": ["relu"],
"mb_size": [20, 40],
"learning_rate": [1e-05, 5e-05],
"data_path": ["/vol1/ibrahim/data/joint_5000/"],
"z_dim": [100],
"g_dropout": [0],
"scaling": ["none"],
"log_transformation": [0],
"label_noise": [0],
"d_hidden_layers": [
    [600, 170]
],
"g_hidden_layers": [
    [144, 576]
],
"wgan": [0, 1],
"leaky_param": [0.1]
```

Experiment - Generating more genes (5000)

Best index score: 0.009

Learning rate: 1e-05

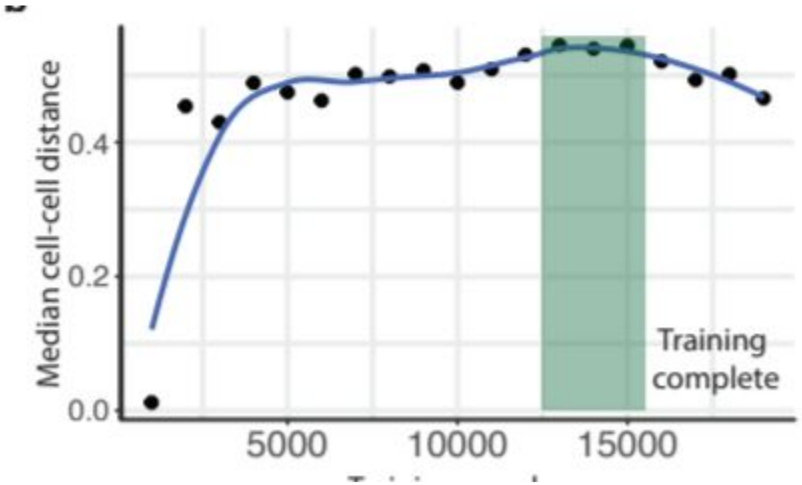
Mini batch size: 40



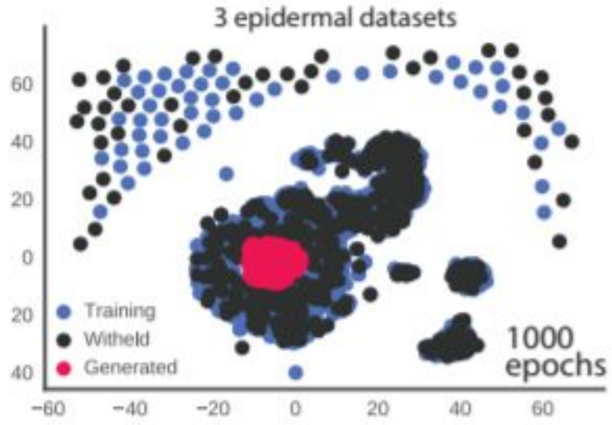
Paper published on this topic

“Generative adversarial networks uncover epidermal regulators and predict single cell perturbations”, February 2018

Evaluation methods they use:

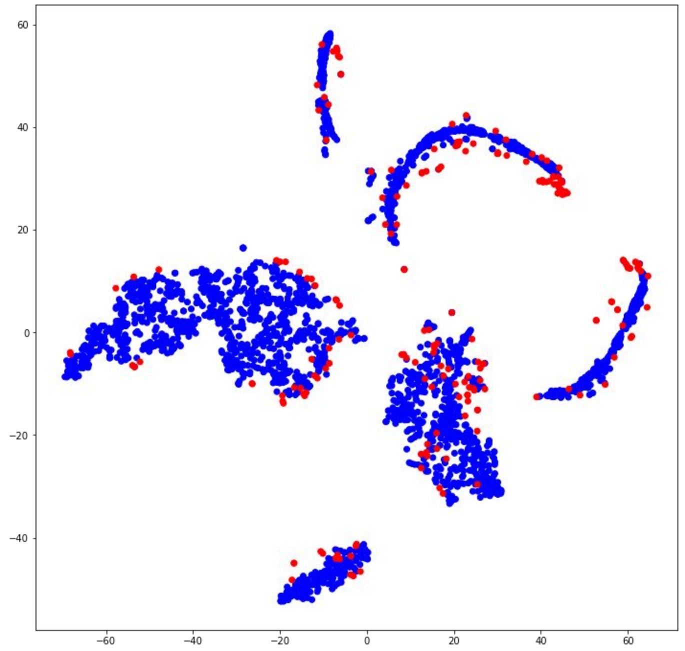


Median cell-cell distance (Ghahramani, 2018)



t-SNE plot(Ghahramani, 2018)

Using t-SNE in our models



t-SNE plot of one of our good performing experiments

Disadvantages of using it for evaluation:

- t-SNE is Stochastic damages reproducibility
- Slow!
- Distribution of training data in the plot will change in every experiment

Findings & Discussion

- Wasserstein GAN and classical GAN formulation gives similar results
- Two layer works better than one layer. Three layer model does not learn marker gene expression
- Tanh -> one hidden layer, RELUs -> two hidden layer
- Best learning rate proportional to the minibatch size
- Dropout and label noise not helping.
- Adam and RMSProp performs much better than other optimizers.
- Minmax scaling worsen the performance
- Index score is consistent with PCA and differential gene expression and also is quick measurement tool for evaluation of models.

Findings & Discussion

In differential gene expression analysis ~30% of the genes are differentially expressed in most of the best result runs. => Not sufficient. Thus,

- Need better evaluation metrics that quantify the performance of all genes
  - Median cell-cell distance between generated samples and true data gives inconsistent results with PCA and index scores and **slow!**
  - Correlation is also inconsistent with index scores

For more successful models

- Need a better architecture (smarter than fully connected layers)
- Need more data to train GANs

Conclusion

Created repository which implements ACGAN with GAN and WGAN-GP loss and does grid search, plot desired analysis and compare the analysis

Used ACGAN to generate gene expressions of specific cell types.

Developed a measurement to evaluate GANs

Performed lots of hyperparameter tuning to obtain the best results

References

References

Ghahramani, Arsham, et al. “Generative Adversarial Networks Uncover Epidermal Regulators and Predict Single Cell Perturbations.” BioRxiv, Cold Spring Harbor Laboratory, 1 Jan. 2018, www.biorxiv.org/content/early/2018/02/08/262501.

Tang, Fuchou, Catalin Barbacioru, Yangzhou Wang, Ellen Nordman, Clarence Lee, Nanlan Xu, Xiaohui Wang, et al. 2009. “mRNA-Seq Whole-Transcriptome Analysis of a Single Cell.” Nat Meth 6 (5). Springer Nature: 377–82. doi:10.1038/nmeth.1315.

Arjovsky, Martin, Soumith Chintala, and Léon Bottou. 2017. “Wasserstein GAN.” http://arxiv.org/abs/1701.07875.

Goodfellow, Ian J., Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. “Generative Adversarial Networks.” http://arxiv.org/abs/1406.2661.

References

Wang, Yue J., Jonathan Schug, Kyoung-Jae Won, Chengyang Liu, Ali Naji, Dana Avrahami, Maria L. Golson, and Klaus H. Kaestner. 2016. “Single Cell Transcriptomics of the Human Endocrine Pancreas.” Diabetes, June. American Diabetes Association, db160405.

Xin Y, Et al. n.d. “RNA Sequencing of Single Human Islet Cells Reveals Type 2 Diabetes Genes. - PubMed - NCBI.” Accessed June 5, 2018. https://www.ncbi.nlm.nih.gov/pubmed/27667665.

Haghverdi, Laleh, Aaron T. L. Lun, Michael D. Morgan, and John C. Marioni. 2018. “Batch Effects in Single-Cell RNA-Sequencing Data Are Corrected by Matching Mutual Nearest Neighbors.” Nature Biotechnology 36 (5). Nature Publishing Group: 421.

Lawlor, Nathan, Joshy George, Mohan Bolisetty, Romy Kursawe, Lili Sun, V. Sivakamasundari, Ina Kycia, Paul Robson, and Michael L. Stitzel. 2017. “Single-Cell Transcriptomes Identify Human Islet Cell Signatures and Reveal Cell-Type–specific Expression Changes in Type 2 Diabetes.” Genome Research 27 (2). Cold Spring Harbor Laboratory Press: 208.

Odena, Augustus, Christopher Olah, and Jonathon Shlens. 2016. “Conditional Image Synthesis With Auxiliary Classifier GANs.” http://arxiv.org/abs/1610.09585.