Summary Of Several Autoencoder models

Presentor: Ji Gao

Department of Computer Science, University of Virginia

https://qdata.github.io/deep2Read/

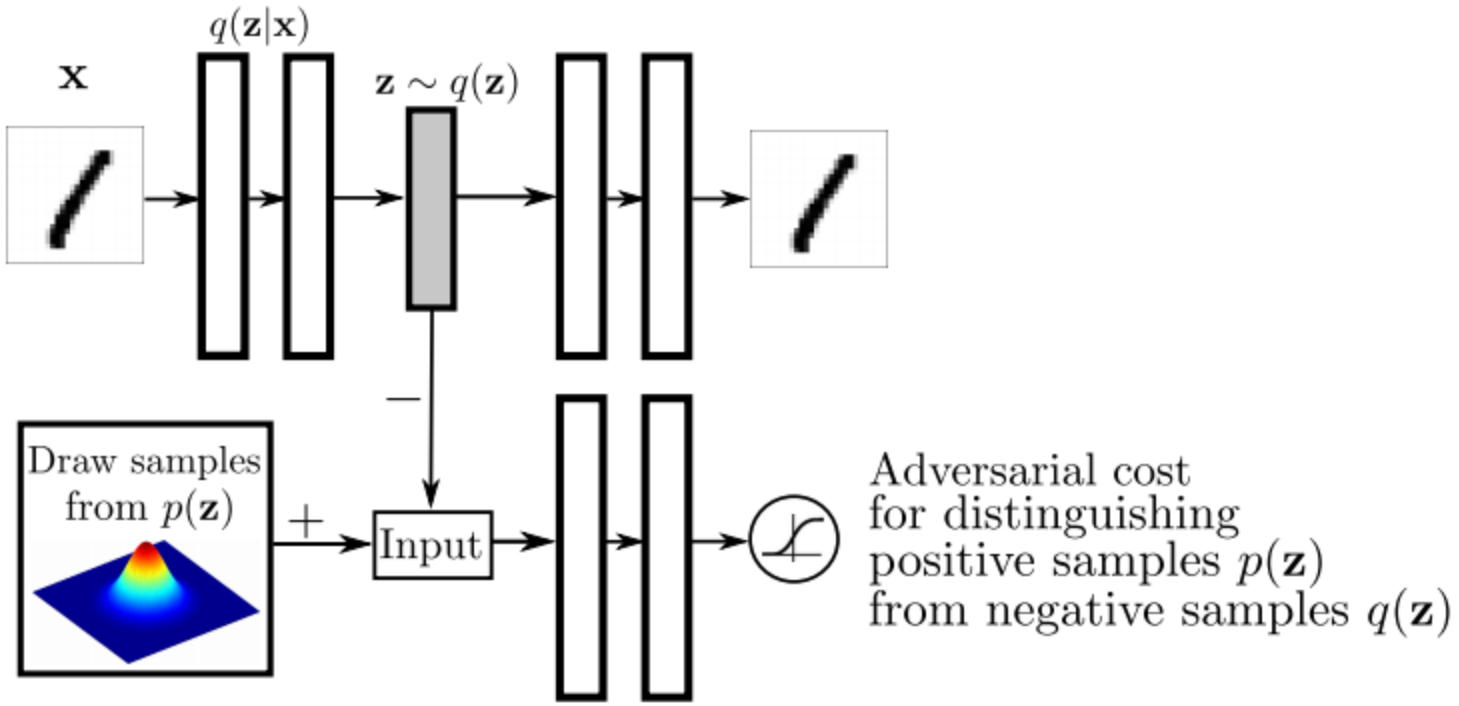
List

- Adversarial Autoencoders
- PixelGAN Autoencoders
- Generating and designing DNA with deep generative models
- Feedback GAN (FBGAN) for DNA: a Novel Feedback-Loop Architecture for Optimizing Protein Functions
- Autoregressive Generative Adversarial Networks

Adversarial autoencoders

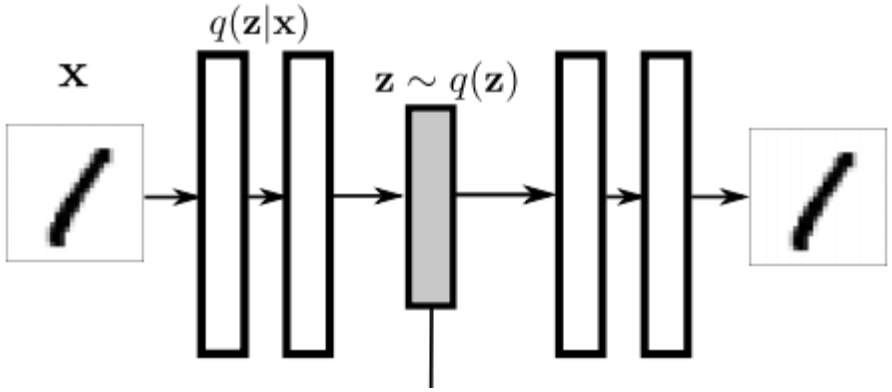
Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, Ian Goodfellow, Brendan Frey

- Use adversarial learning in training autoencoders

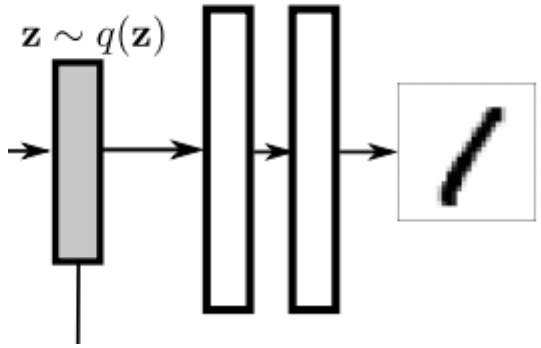


Autoencoders

- Autoencoder



- Decoder = Generator: Start from a prior (often normal distribution), produce sample



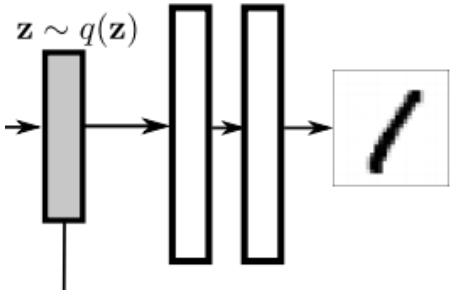
Autoencoders

Why Decoder work?

- Any distribution in d dimension can be generated by a sufficiently complicated function on d normally distributional variables.

ɪsion can be generated by a sufficiently
normally distributional variables.

Why not directly optimize decoder?



- If directly optimize decoder via sampling, it will take exponentially number of samples(And also exponentially parameters)
- A lot of the sampling are useless, for a X, we only need the part of z that are likely to produce X
- Find most likely z to produce X can save huge amount of time and make the process tractable

Variational Autoencoder

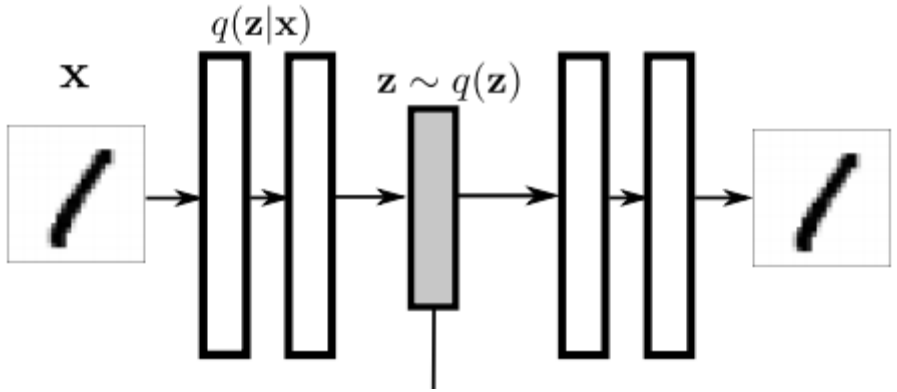
Latent variable $z \sim P(z)$ If we sample $Q(z)$ to approximate $P(x)$, we have

$$D_{KL}(Q(z)||P(z|x)) = E_{z\sim Q}[\log Q(z) - \log P(z|x)]$$

$$= E_{z\sim Q}[\log Q(z) - \log P(x|z) - \log P(z) + \log P(x)] \quad \text{Bayesian}$$

$$\log P(x) - D_{KL}(Q(z)||P(z|x)) = E_{z\sim Q}[\log P(z|x)] - D_{KL}(Q(z)||P(z))$$

Reasonable to let $Q(z)$ conditioned on x.



We have:

$$\log P(x) \geq E_{z\sim Q}[\log P(z|x)] - D_{KL}(Q(z|x)||P(z))$$

Variational bound

Variational Autoencoder

$\log P(x) \geq E_{z\sim Q}[\log P(z|x)] - D_{KL}(Q(z|x)||P(z))$

In VAE, we let $Q(z|x) = N(z|\mu(x;\theta), \Sigma(x;\theta))$

In this case:

$$E_{X\sim D}\left[\log P(X) - \mathcal{D}\left[Q(z|X)\|P(z|X)\right]\right] = E_{X\sim D}\left[E_{z\sim Q}\left[\log P(X|z)\right] - \mathcal{D}\left[Q(z|X)\|P(z)\right]\right]$$

Sample x and z, we have $\log P(X|z) - \mathcal{D}\left[Q(z|X)\|P(z)\right]$

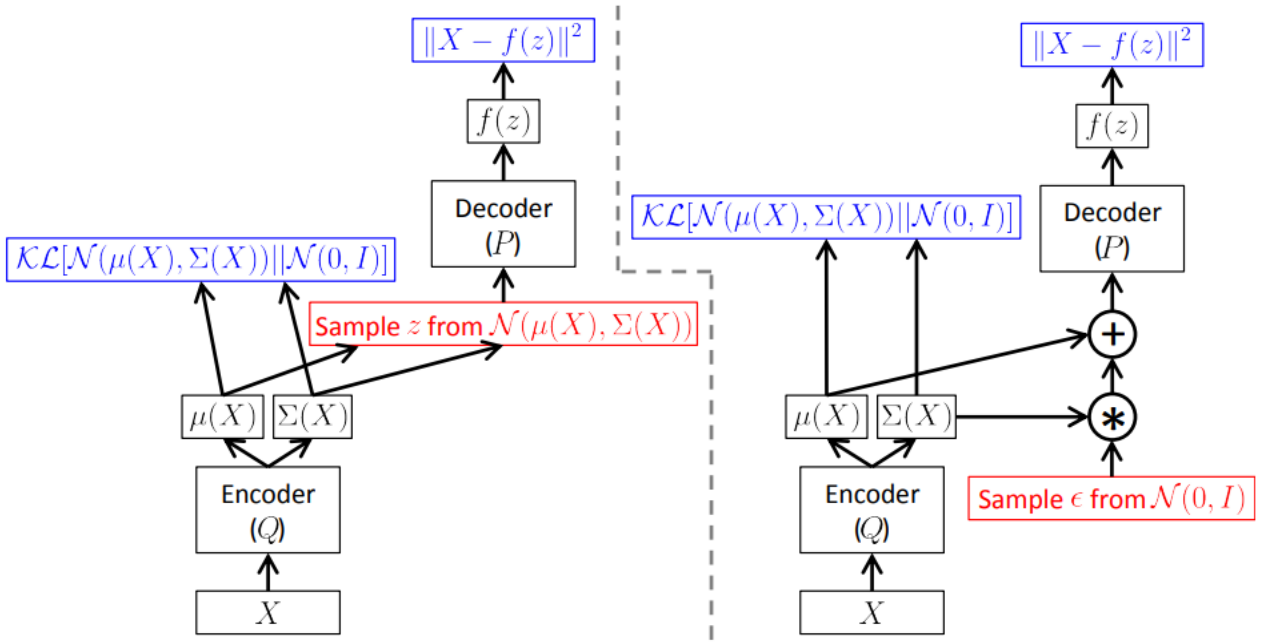
Posterier

- Gaussian Posterier

$$P(X|z;\theta) = \mathcal{N}(X|f(z;\theta), \sigma^2 * I).$$

Variational Autoencoder

From *Tutorial on Variational Autoencoders* https://arxiv.org/abs/1606.05908



$$z = \mu(X) + \Sigma^{1/2}(X) * \epsilon$$

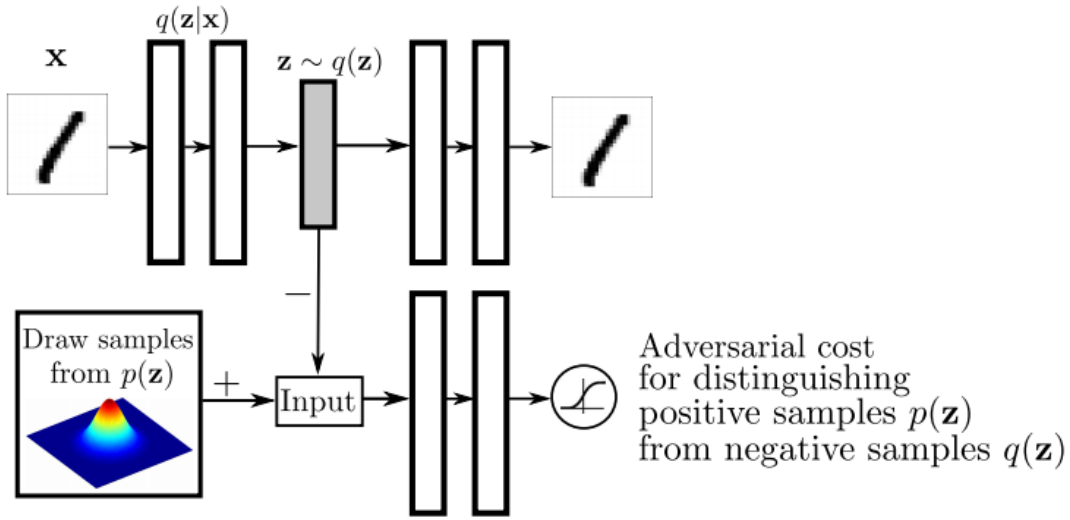
Adversarial autoencoder

- VAE works on

$$\log P(x) \geq E_{z\sim Q}[\log P(x|z)] - D_{KL}(Q(z|x)||P(z))$$

- $D_{KL}(Q(z|x)||P(z))$ term can be optimized in adversarial training
- Train repeatedly in two steps:

1. Maximize $E_{z\sim Q}[\log P(x|z)]$
2. Minimize the distance between $Q(z|x)$ and $P(z)$

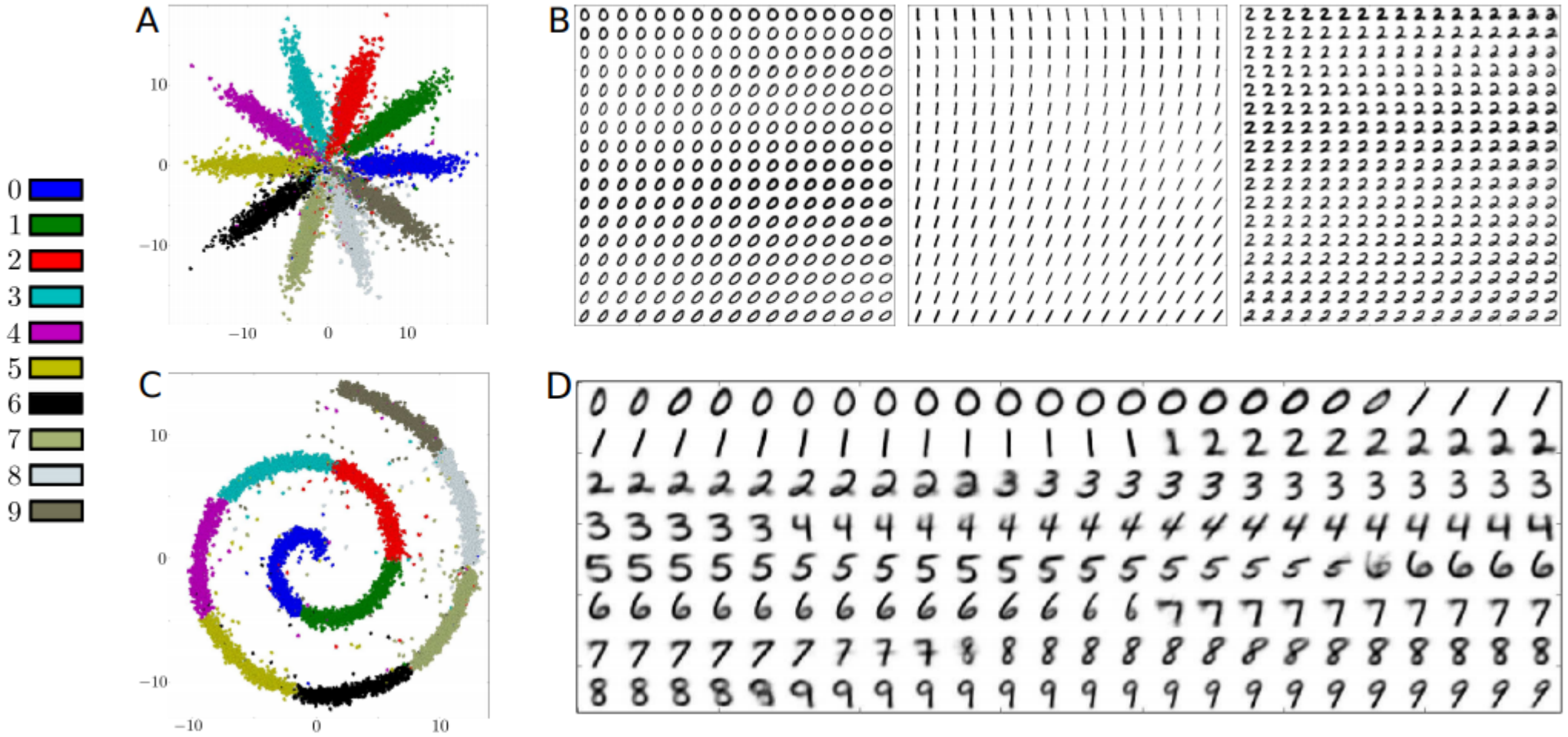


Freedom of choosing q()

- Compare to VAE, in this form it can be optimized using several different ways:
- 1. Deterministic: $q(z|x)$ is a deterministic function on x
- 2. Gaussian posterior: $Q(z|x) = N(z|\mu(x;\theta), \Sigma(x;\theta))$similar to VAE. Can use the same reparameterization
- 3. Universal approximator posterior, $q(z|x,\eta) = \delta(z - f(x,\eta))$

$$q(\mathbf{z}|\mathbf{x}) = \int_{\eta} q(\mathbf{z}|\mathbf{x},\eta)p_{\eta}(\eta)d\eta \quad \Rightarrow \quad q(\mathbf{z}) = \int_{\mathbf{x}}\int_{\eta} q(\mathbf{z}|\mathbf{x},\eta)p_{d}(\mathbf{x})p_{\eta}(\eta)d\eta d\mathbf{x}$$

Adversarial autoencoder performance

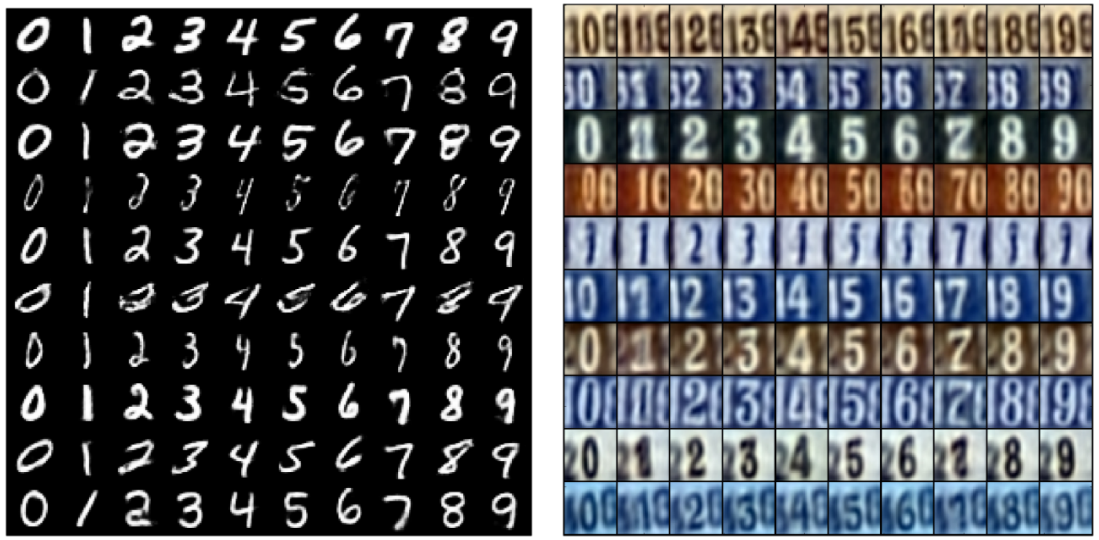


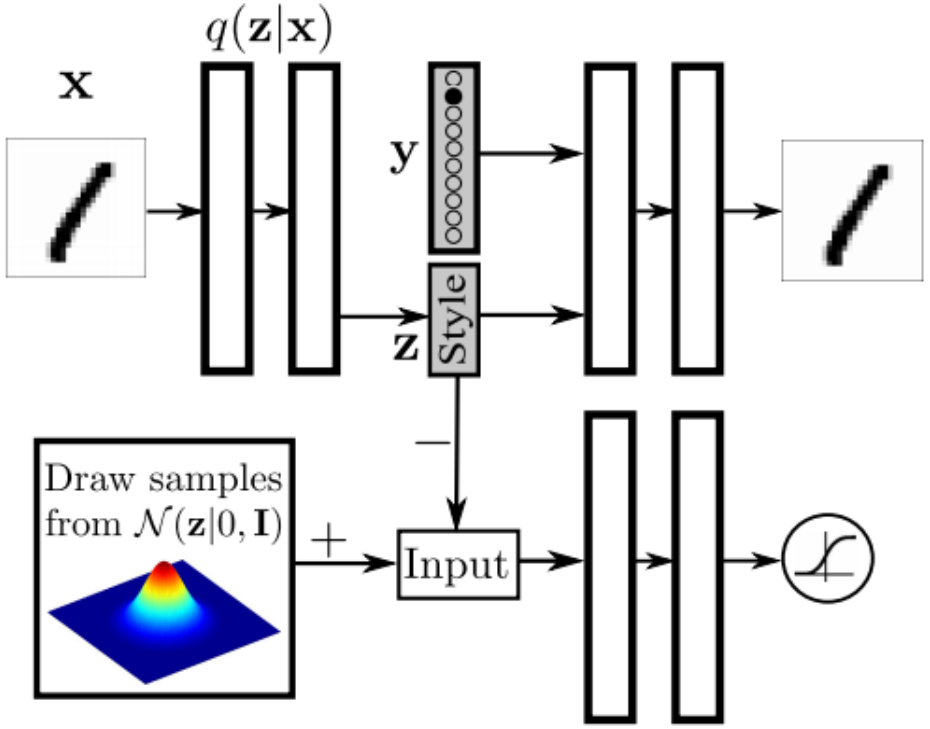
Log likelihood

	MNIST (10K)	MNIST (10M)	TFD (10K)	TFD (10M)
DBN [Hinton et al., 2006]	138 ± 2	-	1909 ± 66	-
Stacked CAE [Bengio et al., 2013]	121 ± 1.6	-	2110 ± 50	-
Deep GSN [Bengio et al., 2014]	214 ± 1.1	-	1890 ± 29	-
GAN [Goodfellow et al., 2014]	225 ± 2	386	2057 ± 26	-
GMMN + AE [Li et al., 2015]	282 ± 2	-	2204 ± 20	-
Adversarial Autoencoder	$\mathbf{340 \pm 2}$	**427**	$\mathbf{2252 \pm 16}$	**2522**

Supervised learning

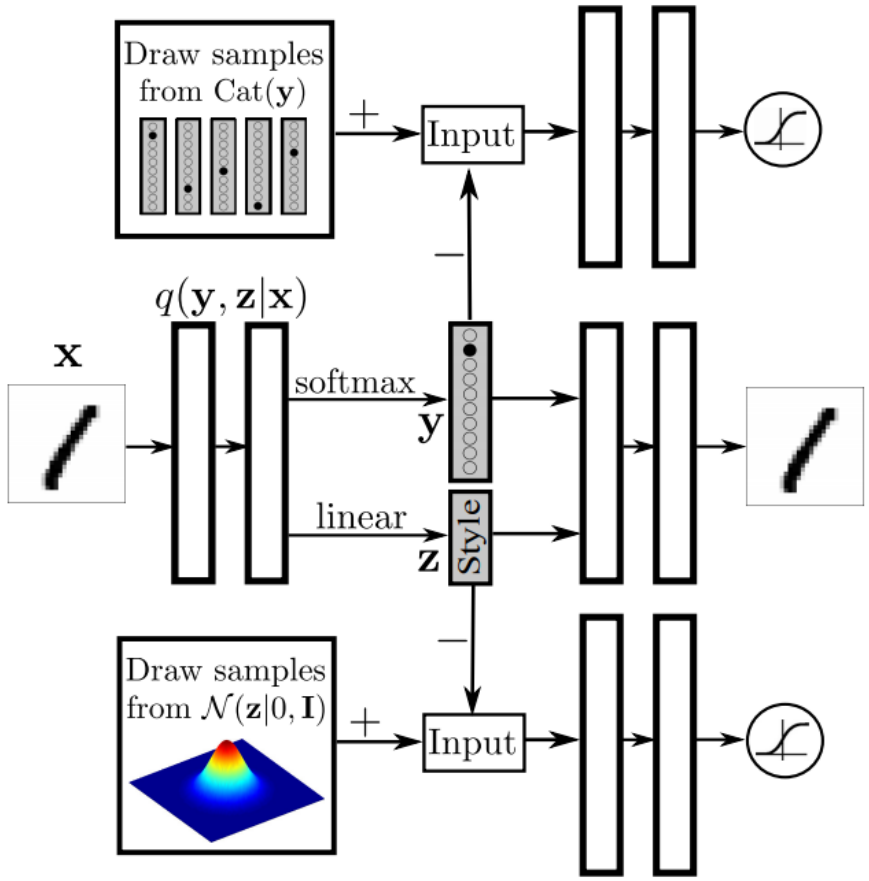
- Fully supervised learning to generate samples in a particular way





Semi-supervised learning

- 2 adversarial nets: One with categorical data
- Train in three phases:
- 1. Reconstruction phase
- 2. Regularization phase
- 3. Semi-supervised phase

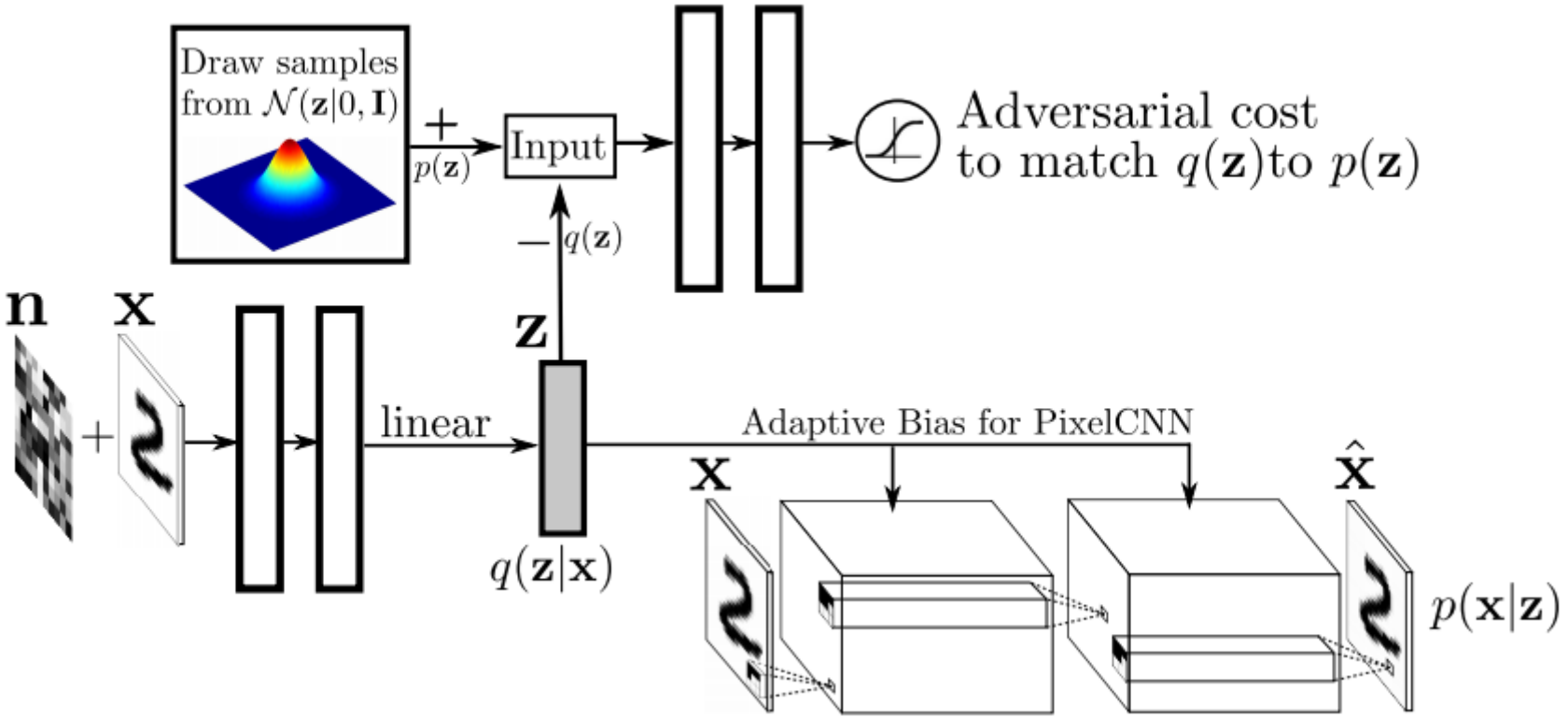


	MNIST (100)	MNIST (1000)	MNIST (All)	SVHN (1000)
NN Baseline	25.80	8.73	1.25	47.50
VAE (M1) + TSVM	11.82 (±0.25)	4.24 (±0.07)	-	55.33 (±0.11)
VAE (M2)	11.97 (±1.71)	3.60 (±0.56)	-	-
VAE (M1 + M2)	3.33 (±0.14)	2.40 (±0.02)	0.96	36.02 (±0.10)
VAT	2.33	1.36	0.64 (±0.04)	24.63
CatGAN	1.91 (±0.1)	1.73 (±0.18)	0.91	-
Ladder Networks	1.06 (±0.37)	0.84 (±0.08)	0.57 (±0.02)	-
ADGM	0.96 (±0.02)	-	-	16.61 (±0.24)
Adversarial Autoencoders	1.90 (±0.10)	1.60 (±0.08)	0.85 (±0.02)	17.70 (±0.30)

Table 2: Semi-supervised classification performance (error-rate) on MNIST and SVHN.

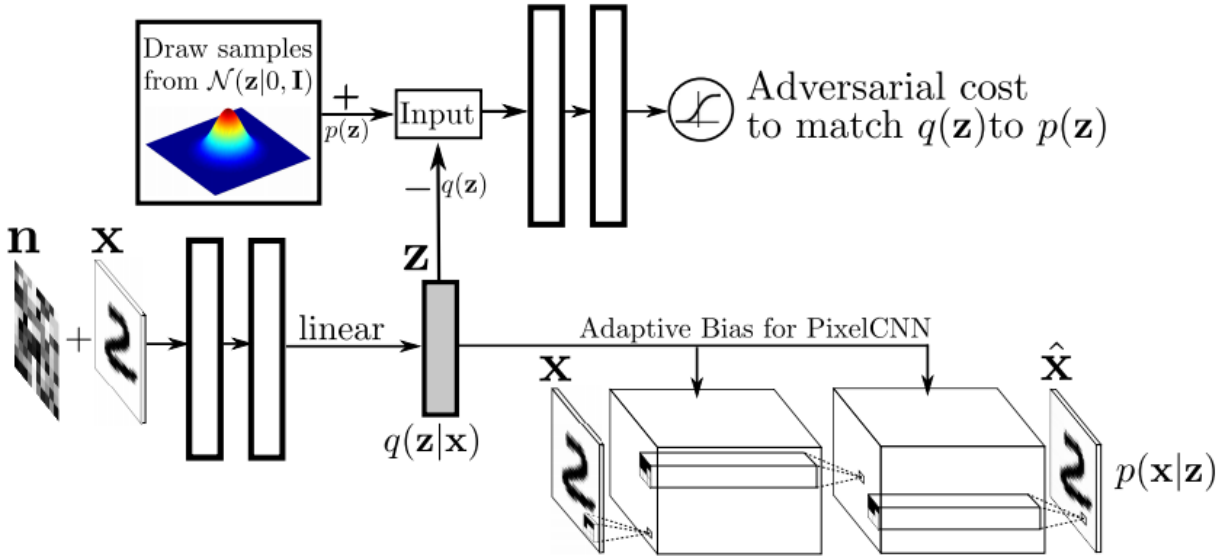
PixelGAN Autoencoders

Alireza Makhzani, Brendan Frey

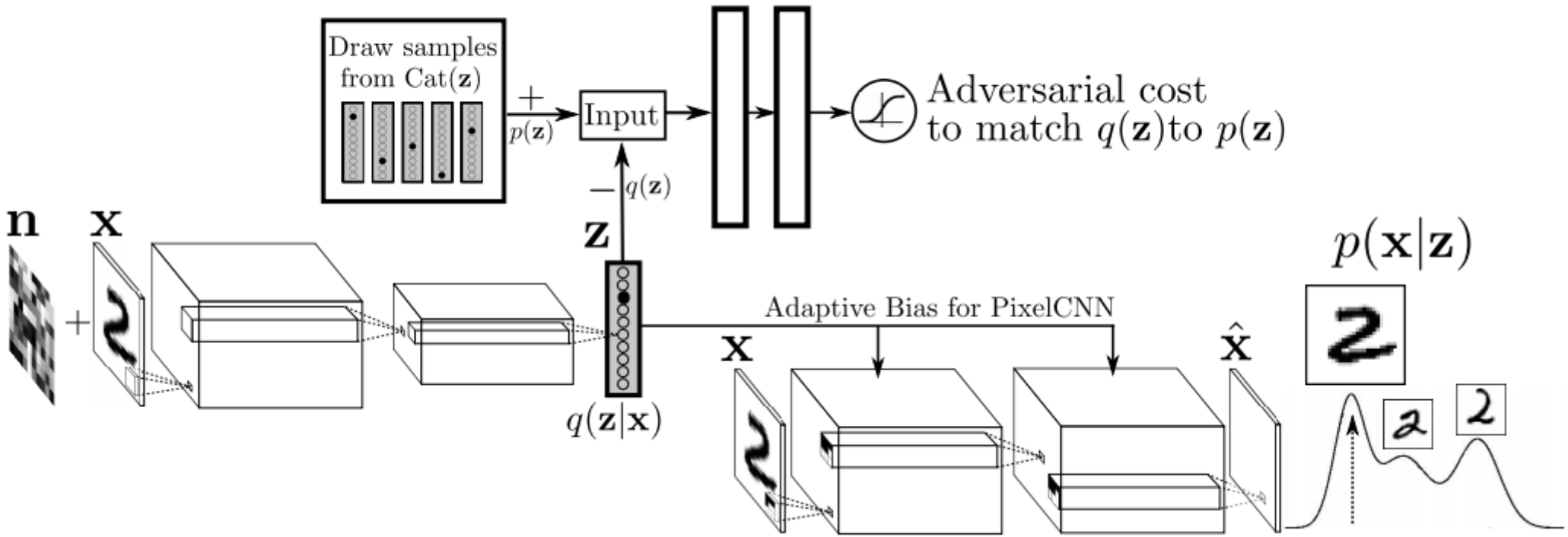


PixelGAN Autoencoders

- Use PixelCNN as the generative path
- PixelCNN conditioned on q(z|x)



Categorical prior



Experiment

	MNIST (Unsupervised)	MNIST (20 labels)	MNIST (50 labels)	MNIST (100 labels)	SVHN (500 labels)	SVHN (1000 labels)	NORB (1000 labels)
VAE [25]	-	-	-	3.33 (±0.14)	-	36.02 (±0.10)	18.79 (±0.05)
VAT [26]	-	-	-	2.33	-	24.63	9.88
ADGM [27]	-	-	-	0.96 (±0.02)	-	22.86	10.06 (±0.05)
SDGM [27]	-	-	-	1.32 (±0.07)	-	16.61 (±0.24)	9.40 (±0.04)
Adversarial Autoencoder [6]	4.10 (±1.13)	-	-	1.90 (±0.10)	-	17.70 (±0.30)	-
Ladder Networks [28]	-	-	-	0.89 (±0.50)	-	-	-
Convolutional CatGAN [24]	4.27	-	-	1.39 (±0.28)	-	-	-
InfoGAN [18]	5.00	-	-	-	-	-	-
Feature Matching GAN [29]	-	16.77 (±4.52)	2.21 (±1.36)	0.93 (±0.06)	18.44 (±4.80)	8.11 (±1.30)	-
Temporal Ensembling [30]	-	-	-	-	7.05 (±0.30)	5.43 (±0.25)	-
PixelGAN Autoencoders	5.27 (±1.81)	12.08 (±5.50)	1.16 (±0.17)	1.08 (±0.15)	10.47 (±1.80)	6.96 (±0.55)	8.90 (±1.0)

Table 1: Semi-supervised learning and clustering error-rate on MNIST, SVHN and NORB datasets.

Generating and designing DNA with deep generative models

Nathan Killoran, Leo J. Lee, Andrew Delong, David Duvenaud, Brendan J. Frey

- 2017
- Three approaches to generate DNA sequence:
- 1. GAN
- 2. Activation maximization(Deep Dream)
- 3. A joint of 1 and 2

GAN on discrete output

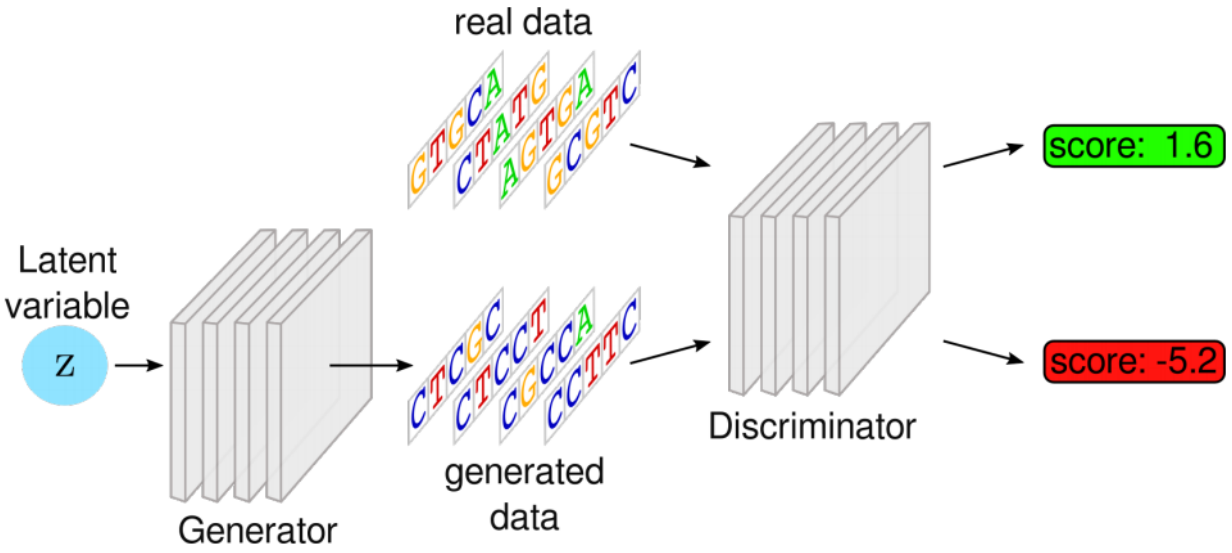
- DNA sequence is discrete, similar to NLP task
- WGAN-GP can generate the sequence in the direct way:

  Let GAN directly output one-hot character embeddings from a latent vector without any discrete sampling step. Softmax directly passed to critic.

```
Busino game camperate spent odea  Solice Norkedin pring in since
In the bankaway of smarling the   ThiS record ( 31. ) UBS ) and Ch
SingersMay , who kill that imvic  It was not the annuas were plogr
Keray Pents of the same Reagun D  This will be us , the ect of DAN
Manging include a tudancs shat "  These leaded as most-worsd p2 a0
His Zuith Dudget , the Denmbern   The time I paidOa South Cubry i
In during the Uitational questio  Dour Fraps higs it was these del
Divos from The ’ noth ronkies of  This year out howneed allowed lo
She like Monday , of macunsuer S  Kaulna Seto consficutes to repor
```

GAN on DNA

- Use such method on DNA:



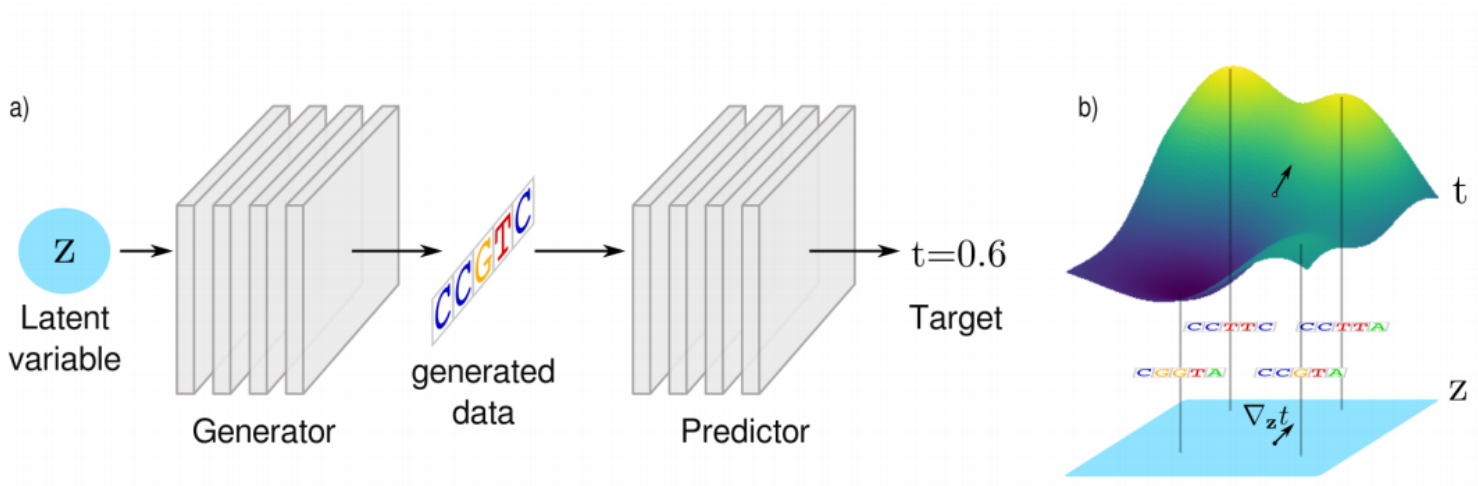
Activation Maximization

- The method is actually Deep Dream:
- Start from sample x, make it target at a certain property t(output)
- $x \rightarrow x + \epsilon \nabla_x t$
- Works on continuous case, so need to relax discrete symbols into continuous case

$$\mathbf{x}_{ij} = \frac{\exp(\mathbf{z}_{ij})}{\sum_{k=1}^{4} \exp(\mathbf{z}_{ik})}.$$

Joint method

- Use GAN to generate sample
- Use activation maximization to optimize a sample to certain properties



Experiment: Motif

- Sample sequences tuned to have a high predictor score

a)

b)

TGAGAGTGATGTATTGGAATTGATGCCTCACCTCTGCTTGCAGACTGTCA
GGAATGAACTGGGGAGACAGGCCCAGAGGAATTGAGAAAGTAATGAGCAC
GCCCTGGGTTTTAAGAAATACTGTTGCATCAGGGCAAATGTAAGATTTTG
TTTTGTTTGAGATCTGTGGGGTATGCTGGAATTAAAGTCTGGACTACCAC
CTGATACTGAATGCAGATTTGAAGAACAAAGGGTATTAAAACACATGCTT
GATCCCCAAGTGTGGAATTGAGAAGGAAGCTGGAGAATCCCCAAACTCTG
CAGCCACATCAGCTTACCTAAGGAAGTGATGTGTTTTAAAACCAGCTTTG
TAGAATTTTTCTTGGTATTAATGATGATCTAGGCTTACACAGGGACATCA
GACATTGCTTAGTCTGAGGGATACAGTGGGGAGTGGGTATTAAAATCTCC
ACATGCCTGAGACATTCCTGCTCTTGAATCTGAGGAATTATGCTTAATCC

Experiment

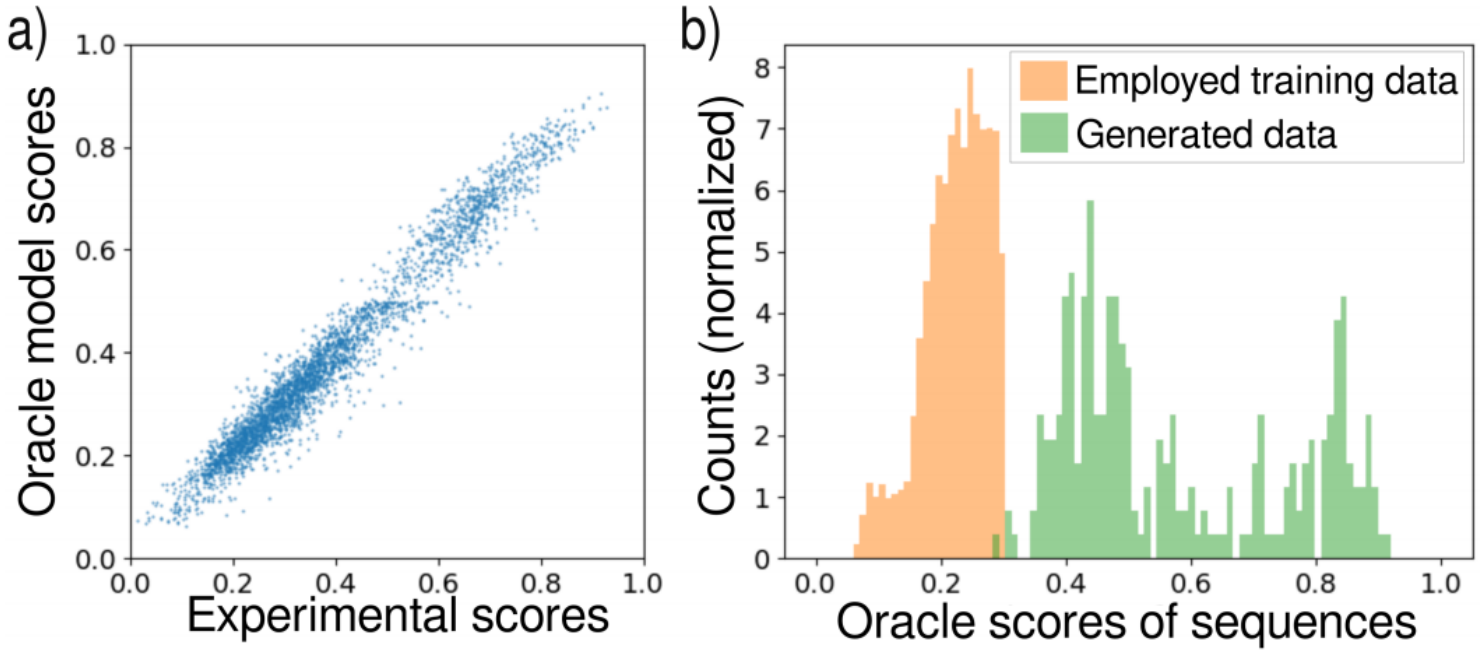


Figure 8: Protein binding optimization with a learned predictor model. a) Original experimental data contains sequences and measured binding scores (horizontal axis); we fit a model to this data (vertical axis) to serve as an oracle for scoring generated sequences. Plot shows scores on held-out test data (Spearman correlation 0.97). b) Data is restricted to sequences with oracle scores in the 40th percentile (orange distribution), then used to train a generator and predictor model. Generated sequences are optimized to have as high binding score as possible. These genererated samples are then scored with the oracle (green distribution). The design process has clearly picked up enough structure that it can generalize well beyond the training data.

Feedback GAN (FBGAN) for DNA: a Novel Feedback-Loop Architecture for Optimizing Protein Functions

Anvita Gupta, James Zou

- 2018
- Target: Design DNA automatically following some properties

Feedback GAN

- (a) WGAN-GP as generator
- (b) Analyzer: suppose to be any function
  - Rate the generated samples
  - Mark the top sorted samples as real samples
- (c) Feedback scheme
  - Send the top sorted sample back to the discriminator

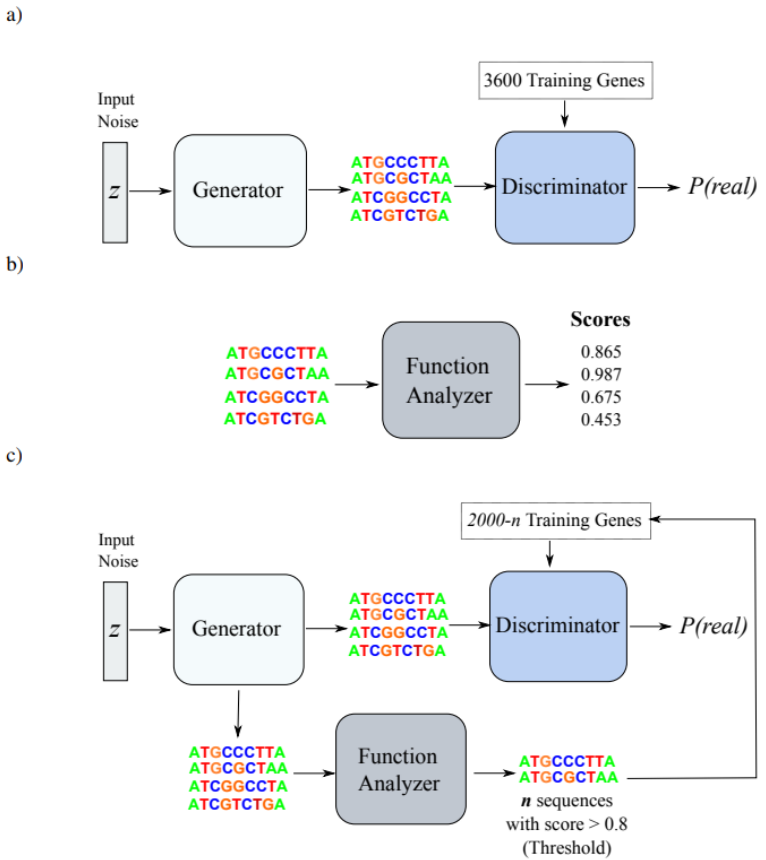
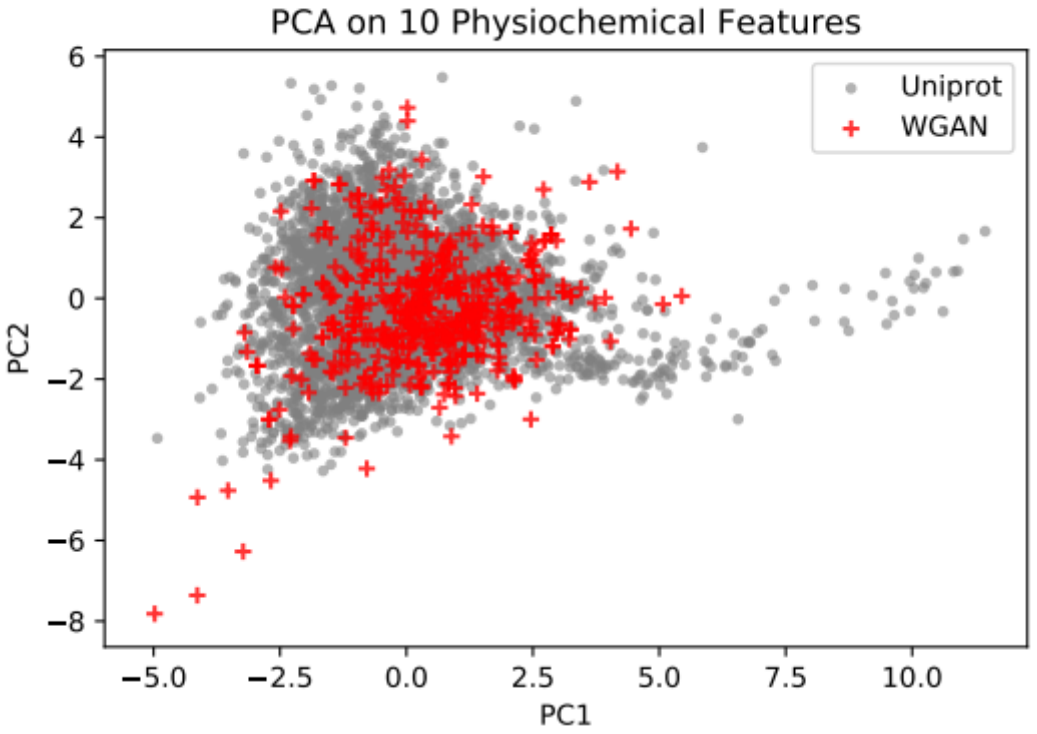


Figure 1: a) The general training mechanism of the GAN model used to produce gene sequences. The training set used was 3600 genes coding for Uniprot proteins under 50 amino acids in length. b) The general form of the function analyzer, which takes in a sequence and produces a score. The analyzer may be any model which fits this framework, from a deep-neural network to a lab. c) The novel feedback-loop training mechanism in FBGAN. At every epoch, several predictions are sampled from the generator and input into the analyzer. The analyzer gives a score to each sequence as demonstrated in b, and the highest scoring sequences are selected. These high scoring sequences are input back into the discriminator as "real" data. n selected sequences from the analyzer replace the n oldest sequences in the "real" training dataset of the discriminator. In this way, gradually the discriminator's set of "real" data is replaced by synthetic data receiving high scores from the analyzer.

Evaluation

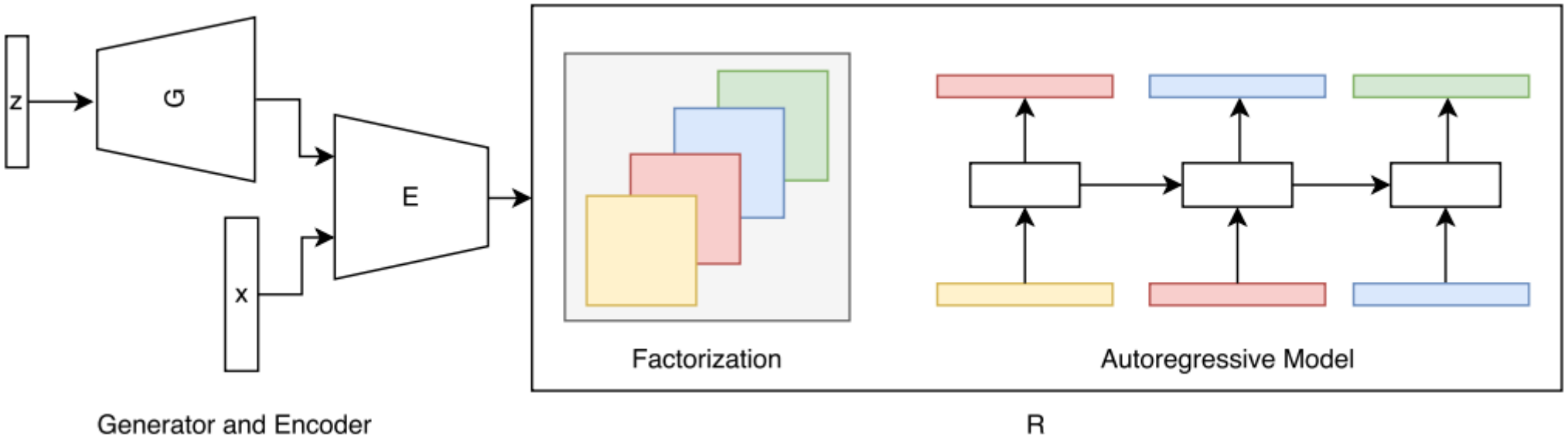
- Before training, 3.125% of sequences initially followed the correct gene structure
- After training, 77.08% of sampled sequences contained the correct gene structure



Autoregressive Generative Adversarial Networks

Yasin Yazici, Kim-Hui Yap, Stefan Winkler

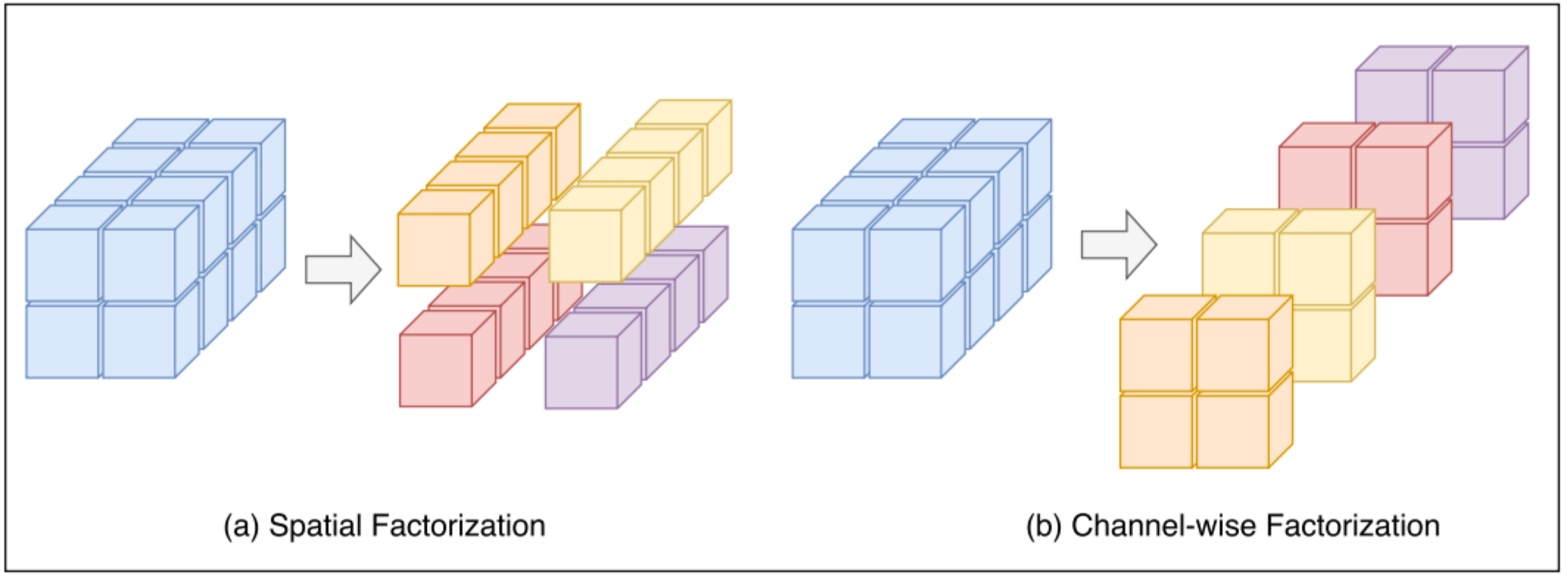
- ICLR 18 Workshop



ARGAN

- Replace discriminator into a CNN + Autoregressive model
- Motivation: an autoregressive model would model the feature distribution better than fully connected layers

S-ARGAN and C-ARGAN



Result

Model	CIFAR-10	STL-10
Real data	11.24 ± 0.16	26.08 ± 0.26
ALI (Dumoulin et al., 2016)	5.34 ± 0.05	-
BEGAN (Berthelot et al., 2017)	5.62	-
D2GAN (Dinh Nguyen et al., 2017)	7.15 ± 0.07	7.98
MGGAN (Hoang et al., 2017)	8.23	9.09
DFM (Warde-Farley & Bengio, 2017)	7.72 ± 0.13	8.51 ± 0.13
Improved-GAN (Salimans et al., 2016)	6.86 ± 0.06	-
DCGAN (Radford et al., 2015)	6.40 ± 0.05	7.54
EBGAN (Zhao et al., 2016)	6.74 ± 0.09	-
WGAN-GP (ResNet) (Gulrajani et al., 2017)	7.86 ± 0.07	9.05 ± 0.12
WGAN-GP (DCGAN) (Our implementation)	6.80	-
S-ARGAN (Proposed)	6.50	7.44
C-ARGAN (Proposed)	6.46	7.60
PARGAN (Proposed)	6.86	7.89