Koç University
COMP547 Deep Unsupervised Learning
Lecture Notes*

Seher Özçelik

April 30, 2021

10.1 Autoregressive models

Start with a reminder that some autoregressive models that we have seen so far are PixelRNN, PixelCNN, PixelCNN++ and PixelSNAIL. As an overview, first remember the Masked Autoencoder Distribution Estimation MADE(2015) approach. MADE creates an autoencoder where the latent space logits will give you the conditional probabilities of each input and the product of them will give you the joint probability of all inputs as seen in Fig.1a. Then, we have seen PixelRNN/CNN(2016) approach as in Fig.1b where a pixel's probability is conditioned on previous surrounding pixels and these surrounding is determined by a kernel with binary mask. Hence this is a convolutional approach to masked autoencoders. This simplifies the design of the masked autoencoder approach to pictures and add efficiency. Shortcoming of this model, namely blind spots, is dealed with by using two different stacks as in the Fig.1b. Yet, another autoregressive model WaveNet(2016) has come for speech recognition Fig.1c. Here, by 1D kernel/mask and appropriate dilation, output sees many previous input in an effective manner. Moreover, PixcelCNN idea is extended to video generation process as in Video Pixel Networks (2017) Fig.1d. We also saw Subscale Pixcel Networks (2018), which generates picture in smaller resolution or in smaller scale then by conditioning on these pixels continue to generate pixels between them. We also mentioned Hierarchical Autoregressive Image Models with Auxiliary Decoders (2019) and Scaling Autoregressive Video Models (2020). Note that Autoregressive models are mostly used in language models from the start. Please check OpenAI GPT-3.

Basically, in autoregressive models, you get the logits of the data by a network coming from the input like models in Fig.1. Then you find the probabilities by softmax or some probability function and train them by using cross entropy or maximum likelihood loss respectively. Check the simple equations Eq.1. Note that in the second equation, you have an ordering.

$$
\begin{aligned}
log\ p_\theta(x) &= \sum_{i=1}^{d} log\ p_\theta(x_i|parents(x_i)) \\
log\ p_\theta(x) &= \sum_{i=1}^{d} log\ p_\theta(x_i|x_{i:i-1})
\end{aligned}
\tag{1}
$$

Huge advances of these models due to:

- Larger batch sizes
- More hidden units
- More layers (Note that GPT-3 has 175 billion parameters)
- Clever ways to condition on auxiliary variables (e.g. Subscale Pixel Networks)
- Preprocessing
- Computer power
- Several days / weeks of training
- Fewer assumptions

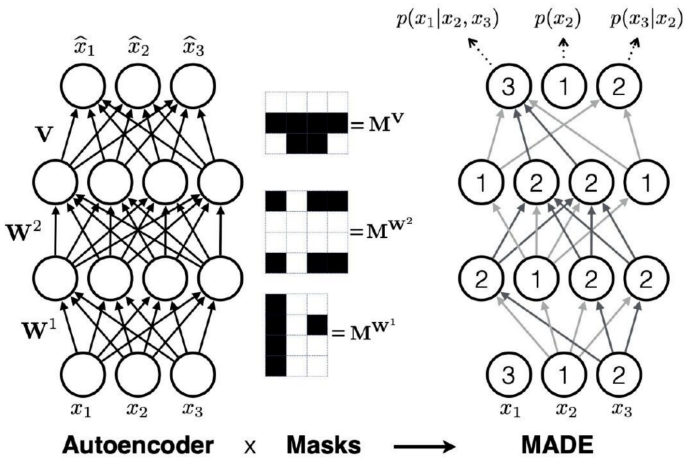
*Instructions borrowed from Andreas Geiger from his notes for his Deep Learning class.

- Architectural advances
  - Masked / Causal Convolutions
  - Dilated Convolutions
  - Transformers
- Loss functions
  - Relying heavily on well-behaved cross-entropy loss

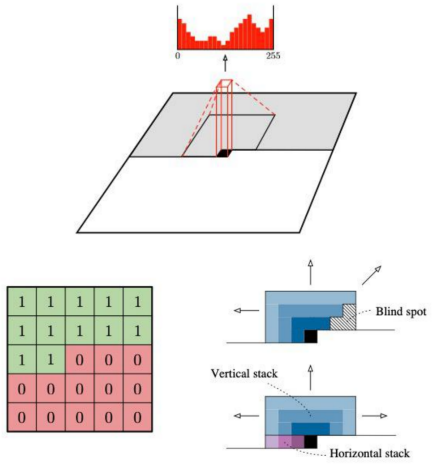
Drawbacks of these models:

- No single layer of learned representation
- Currently, sampling time is slow for practical deployment.
- Not directly usable for downstream tasks. (This is mostly for visual domain, for example GPT-3 could be used for many downstream tasks even without any fine tuning.)
- No interpolations.

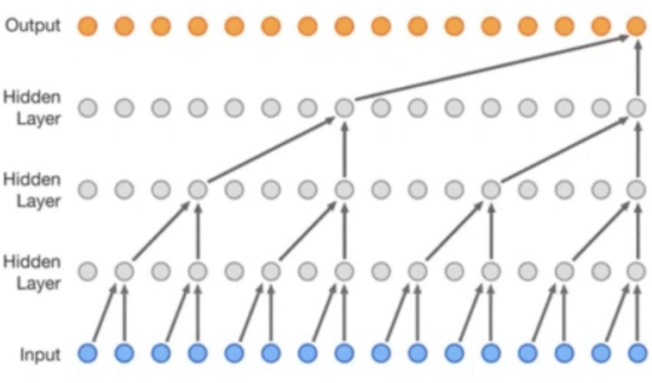
Please check the lecture slides for possible future directions of autoregressive models.



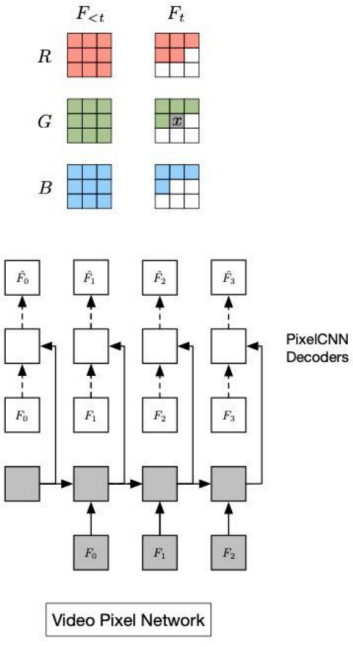
(a) **MADE**



(b) **PixelCNN/RNN**



(c) **WaveNet**



(d) **Video Pixel Networks**

Figure 1: **Autoregressive Models**

10.2 Flow models

As a reminder, some flow models that we saw earlier are NICE(2014), RealNVP(2016), Autoregressive Flows, Inverse Autoregressive Flows, Glow(2018) and Flow++(2019). See the Fig.2 that perfectly shows the spirit of the Flow models. Here, we do not know the true distribution of the data, for that reason, we take a latent space distribution which is accessible to us e.g. Uniform distribution. Then we generate a reversible mapping from data space to latent space and we learn this mapping variables to use it to generate the data with reverse mapping. We can simply write the equations of this kind of transformation as an example. Let x_1 and x_2 are two dimensional data coming from a mixture of gaussians $x_i \sim \sum_{j=1}^{5} \pi_j \mathcal{N}(\mu_j, \sigma_j)$ and we want to learn these parameters to sample new data. Here the latent space Z is in uniform distribution and we can simply go there by cumulative distribution function CDF. To adjust the volume between these two random variables as transforming one to another we will need $\left|det(\frac{dz_i}{dx_i})\right|$ multiplication. Note that derivative of CDF will give the probability density function PDF. Here, you get log likelihood of x_1 and you get the probability variables of x_2 conditioned on x_1 by using multi later perceptron MLP transfer. See equations Eg.2 and notice that you'll maximize log likelihood of joint probability of x_1 and x_2.

$$
\begin{aligned}
p(x_1) &= \sum_{j=1}^{5} \pi_{1_j} \mathcal{N}(\mu_{1_j}, \sigma_{1_j}) \\
z_1 &= f_1(x_1) = CDF(x_1), \quad z_1 \sim U[0,1] \\
log\ p_\theta(x_1) &= log\ p_{z_1}(z_1) + log\ \left|det(\frac{dz_1}{dx_1})\right| \\
log\ p_{\mu_1,\sigma_1}(x_1) &= log\ U(z_1) + log\ PDF(x_1) \\
log\ p_{\mu_1,\sigma_1}(x_1) &= log\ U(z_1) + log\ p(x_1) \\
\mu_{x_2}, \sigma_{x_2}, \pi_{x_2} &\sim MLP(x_1) \\
p(z_2; x_1, x_2) &= f_2(x_2; x_1) = CDF(x_2; \mu_{x_2}(x_1), \sigma_{x_2}(x_1), \pi_{x_2}(x_1)) = CDF(\sum_{j=1}^{5} \pi_{2_j} \mathcal{N}(\mu_{2_j}, \sigma_{2_j})) \\
log\ p_{\mu_2,\sigma_2}(x_2; x_1) &= log\ U(z_2; x_1, x_2) + log\ p(x_2; x_1) \\
log\ p(x_1, x_2) &= log\ p(x_1) + log\ p(x_2; x_1) \\
\mathcal{J} &= -\frac{1}{batch\ size} \sum_{k=1}^{batch\ size} log\ p(x_1^k, x_2^k)
\end{aligned}
\tag{2}
$$

Note that, when this is for pictures, you can take previous pixels as PixelCNN and by conditioned on them you can create mixture of gaussians parameters of current pixel. Then use this for maximizing likelihood of that pixel.

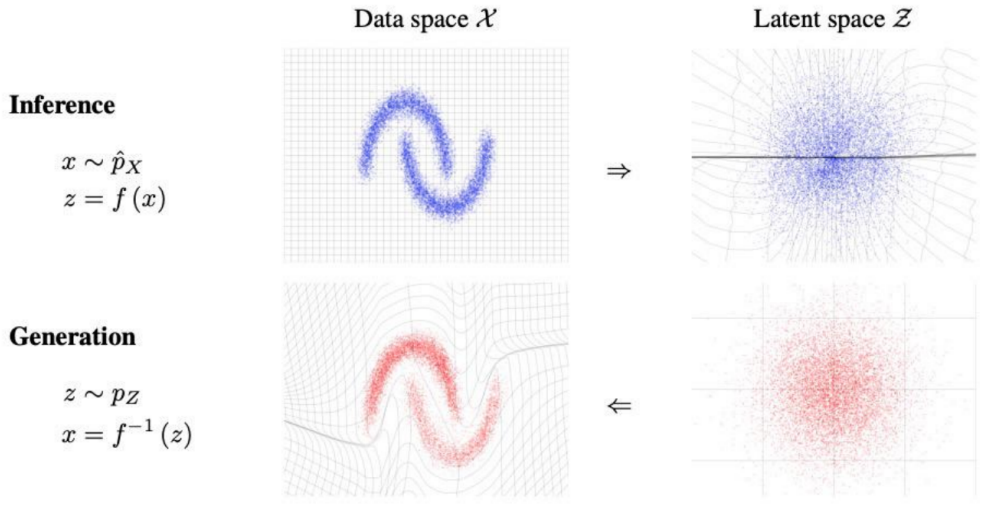
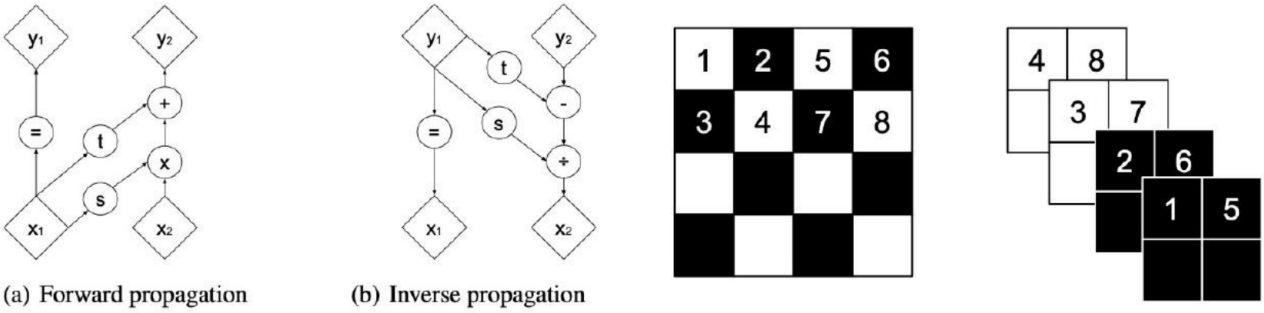


Figure 2: **Flow Models**

Let's look closer to RealNVP (Non Volume Preserving flow, 2016) model. Please see Equations Eg.3 for 2D

data. Note that, when this is for pictures, you can use the checkerboard approach to condition one pixcel on to next/previous one. See Fig.3b

$$
\begin{aligned}
z_i &\sim \mathcal{N}(0,1) \\
z_1 &= x_1 \\
logSigma &= scalePar * tanh(logSigma(MLP(x_1))) + scaleShiftPar \\
z_2 &= exp(logSigma) * x_2 + \mu(MLP(x_1)) \\
same\ for\ z_2 &= x_2 \ \ and\ reverses\ (for\ sampling) \\
\frac{dz}{dx} &= \begin{bmatrix} 1 & 0 \\ \mu(MLP(X_1))' & exp(logSigma) \end{bmatrix} \\
(z_1, z_2) &= (f_{\theta,x1} \circ f_{\theta,x2} \circ f_{\theta,x1} \circ f_{\theta,x2} \circ f_{\theta,x1} \circ f_{\theta,x2} \circ f_{\theta,x1} \circ f_{\theta,x2})\ composition\ of\ MLP's \\
\log p_\theta(x) &= log\ p(z) + log\ \left| det(\frac{dz}{dx}) \right| \ \ remember\ for\ composition\ det(A\dot{B}) = det(A)det(B)
\end{aligned}
\tag{3}
$$



(a) **Computation graph** (b) **Checkerboards**

Figure 3: **RealNVP**

Advantage of the flow models:

- Very stable training process and containing fixed set of engineering practices.

Drawbacks of these modes:

- z is as big as x. Models end up becoming big.
- As of now, no notion of lower dimensional embedding.
- Careful initialization (not really a negative)

For the possible future improvements of the flow models, please check the lecture slides.

10.3 Latent Variable Models

In latent variable models, we basically assume that we have a latent space z, which could generate our data. See Fig.4a. This latent space dimension can be much less that the input space dimension. Moreover, there will be meaningful directions in this latent space that you can generate different samples between two generated samples by interpolation. See Fig.4b

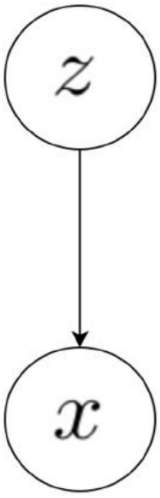
To understand the model calculations, let we have 2D data like Fig.5. Now, let's write the legitimate calculations to train this model as simple VAE.

- After MLP from input space to latent space (encoder step):

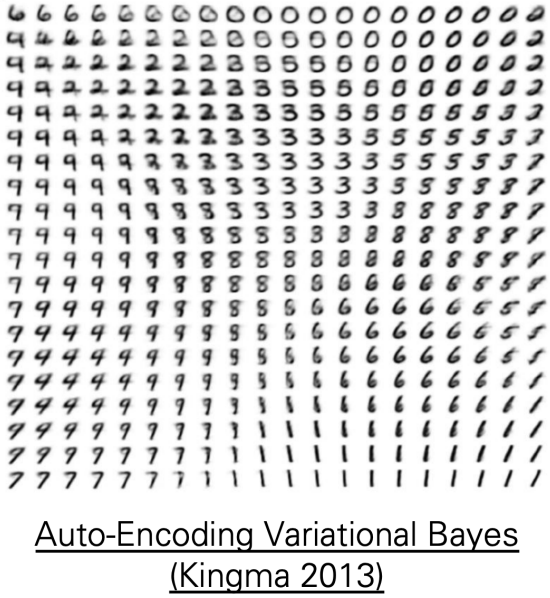
$$[x_1, x_2] -> [\mu_{z_1}, \mu_{z_2}, \sigma_{z_1}, \sigma_{z_2}]$$

- Then sample from standart normal : ϵ_1, ϵ_2

$$[z_1, z_2] = [\sigma_{z_1}\epsilon_1 + \mu_{z_1}, \sigma_{z_2}\epsilon_2 + \mu_{z_2}]$$



(a) **Basically latent variable models**



(b) **Interpolations**

Figure 4: **Latent Variable Models**

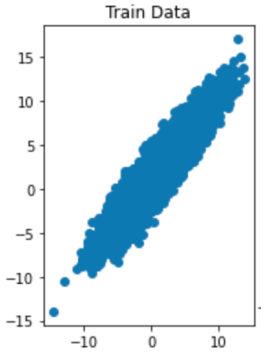


Figure 5: **2D train data**

- After MLP from latent space to input space for reconstruction (decoder step):

$$[z_1, z_2] -> [\mu_{x_1}, \mu_{x_2}, \sigma_{x_1}, \sigma_{x_2}]$$

- (same for x_1, x_2) Find reconstruction loss:

$$\begin{aligned} \mathbb{E}_{x_1}\mathbb{E}_{q_\phi(z_1|x_1)}\left[-log\ p_\theta(x_1|z_1)\right] &= \mathbb{E}_{x_1}\left[-log\ \left(\frac{1}{\sqrt{2\pi}\sigma_{x_1}} e^{-\frac{(x_1-\mu_{x_1})^2}{2\sigma_{x_1}^2}}\right)\right] \\ &= \frac{1}{\#batch}\sum_{x_1}\left(\frac{log(2\pi)}{2} + log\ \sigma_{x_1} + \frac{(x_1-\mu_{x_1})^2}{2\sigma_{x_1}^2}\right) \end{aligned} \tag{4}$$

- (same for x_1, x_2) Find KL (regularization) term for ELBO:

$$
\begin{aligned}
\mathbb{E}_{x_1} KL(q_\phi(z_1|x_1)||p(z_1)) &= \mathbb{E}_{x_1}\mathbb{E}_{q_\phi(z_1|x_1)}\left[log\frac{q_\phi(z_1|x_1)}{p(z_1)}\right] = \mathbb{E}_{x_1}\mathbb{E}_{q_\phi(z_1|x_1)}\left[log\frac{\frac{1}{\sqrt{2\pi}\sigma_{z_1}}e^{-\frac{(z_1-\mu_{z_1})^2}{2\sigma_{z_1}^2}}}{\frac{1}{\sqrt{2\pi}}e^{-\frac{z_1^2}{2}}}\right] \\
&= \mathbb{E}_{x_1}\mathbb{E}_{q_\phi(z_1|x_1)}\left[log\frac{e^{\frac{z_1^2}{2}-\frac{(z_1-\mu_{z_1})^2}{2\sigma_{z_1}^2}}}{\sigma_{z_1}}\right] \\
&= \mathbb{E}_{x_1}\mathbb{E}_{q_\phi(z_1|x_1)}\left[-log\ \sigma_{z_1} + \frac{z_1^2}{2} - \frac{(z_1-\mu_{z_1})^2}{2\sigma_{z_1}^2}\right] \\
&= \mathbb{E}_{x_1}\left[-log\ \sigma_{z_1} + \mathbb{E}_{q_\phi(z_1|x_1)}\left[\frac{z_1^2}{2}\right] - \mathbb{E}_{q_\phi(z_1|x_1)}\left[\frac{(z_1-\mu_{z_1})^2}{2\sigma_{z_1}^2}\right]\right] \\
&= \mathbb{E}_{x_1}\left[-log\ \sigma_{z_1} + \mathbb{E}_{q_\phi(z_1|x_1)}\left[\frac{(z_1-\mu_{z_1}+\mu_{z_1})^2}{2}\right] - \frac{1}{2}\right] \\
&= \mathbb{E}_{x_1}\left[-log\ \sigma_{z_1} + \mathbb{E}_{q_\phi(z_1|x_1)}\left[\frac{(z_1-\mu_{z_1})^2+(\mu_{z_1})^2}{2} + z_1\mu_{z_1} - \mu_{z_1}^2\right] - \frac{1}{2}\right] \\
&= \mathbb{E}_{x_1}\left[-log\ \sigma_{z_1} + \frac{\sigma_{z_1}^2}{2} + \frac{\mu_{z_1}^2}{2} - \frac{1}{2}\right] \\
&= \frac{1}{\#batch}\sum_{x_1}\left(-log\ \sigma_{z_1} + \frac{\sigma_{z_1}^2}{2} + \frac{\mu_{z_1}^2}{2} - \frac{1}{2}\right)
\end{aligned} \tag{5}
$$

- Sampling:

$$[z_1, z_2] \sim \mathcal{N}(0,1) \ \ then \ \ decoder[z_1, z_2] -> use\ [\mu_{x_1}, \mu_{x_2}] = [x_1, x_2]_{sample}$$

or by adding standart gaussian noise ϵ_1, ϵ_2,

$$[z_1, z_2] \sim \mathcal{N}(0,1) \ \ then \ \ decoder[z_1, z_2] -> use\ [\sigma_{x_1}\epsilon_1 + \mu_{x_1}, \sigma_{x_2}\epsilon_2 + \mu_{x_2}] = [x_1, x_2]_{sample}$$

Note that, when it is for, say 32x32 images, computations are as follows;
Let latent space dim is 16. Inputs are 32x32 images.

- After some CONV2D layers, encoder outputs:

$$x -> [\mu_{z_1}...\mu_{z_{16}}, \sigma_{z_1}...\sigma_{z_{16}}]$$

- Then sample from standart normal : $\epsilon_1 ... \ \epsilon_{16}$

$$z = [\sigma_{z_1}\epsilon_1 + \mu_{z_1}, ..., \sigma_{z_{16}}\epsilon_{16} + \mu_{z_{16}}]$$

- After some TRANSPOSE_CONV2D layers (decoder step):

$$z -> reconstruction(x)$$

- Find reconstraction MSE loss:

$$MSE(x, reconstruction(x))$$

- Find KL (regularization) term for ELBO:

  Eq.5

- Sampling:

$$[z_1, ..., z_{16}] \sim \mathcal{N}(0,1) \ \ then \ \ decoder(z) -> x_{sample}$$

Finally note that, we mentioned PixelVAE(2016), Bidirectional-Inference Variational Autoencoder BIVA (2019), VQ-VAE (2017), VQ-VAE 2.0 (2019) in our lectures. Moreover, you may want to check the well known VAE applications like Sketch-RNN(2017), World Models(2018), β-VAE(2017), SCAN:Learning Hierarchical Compositional Visual Concepts Model(2017) which uses β-VAE with additional text input and Generative Query Networks(2018).

Advantages of VAEs:

- Notion of "compressed" representation learning
- Also gives you approximate log-likelihood
- Interpolations, retrospective analysis of what the model learns
- Disentangled representations
- You will have all Generative Model, Density Model, Latent Variables, Dimensionality Reduction

Drawbacks of these models:

- Blurry samples
- Factorized gaussian posterior or decoder assumptions maybe too limiting
- Not much success on large scale is still on-going work
- Encouraging disentanglement with the KL term still only shown on relatively toy domains
- There maybe other ways to learn better representations or to get better samples or get better density estimates (basically, not the best at any one thing but gives you all together)

For possible future advances in VAEs, please check the lecture slides.

10.4 Implicit models

Namely GANs, Generative Adversarial Networks. The basic difference of GANs is, they do not have explicitly written probability function. First paper came in 2014, original starter of GANs. Then DCGAN(2015), StyleGAN(2019), BigGAN(2019) and VQGAN(2020). Through these years sample quality increased very much. To understand the basic structure of GANs, please see the Fig.6[1] Then follow the following equations. Here D stands for Discriminator and G for Generator. Remember that, training procedure tries to maximize the discriminator accuracy while generator tries to fool it. Let's have one dim data x.

$$min_G max_D \mathbb{E}_{x\sim p_{data}}[\log D(x)] + \mathbb{E}_{z\sim p(z)}[\log(1 - D(G(z)))]$$

$$\implies min_G max_D \mathbb{E}_{x\sim p_{data}}[\log\ \sigma(MLP_D(x))] + \mathbb{E}_{z\sim p(z)}[\log\ (1 - \sigma(MLP_D(MLP_G(z))))]\ ; \quad \sigma : sigmoid$$

Take batch x and generate fake data with $MLP_G(z)$ where $z \sim \mathcal{N}(0, 1)$. Then,

$$minimize\ by\ (-discriminator)\ (n\ times)$$

$$-> \ -\frac{1}{\#batch}\sum_x log\ \sigma(MLP_D(x)) - \frac{1}{\#batch}\sum_z log\ (1 - \sigma(MLP_D(MLP_G(z))))$$

Generate fake data with $MLP_G(z)$ where $z \sim \mathcal{N}(0, 1)$ again

$$minimize\ by\ generator -> \ \frac{1}{\#batch}\sum_z log\ (1 - \sigma(MLP_D(MLP_G(z))))$$

Advantages of GANs:

- Amazing samples
- Need less computer power
- More easy to train compared to others to get good samples
- Powerful interpolation and conditional generation

Drawbacks of GANs:

- Plenty of varying engineering tricks and details

[1] https://developers.google.com/machine-learning/gan/gan$_s tructure$

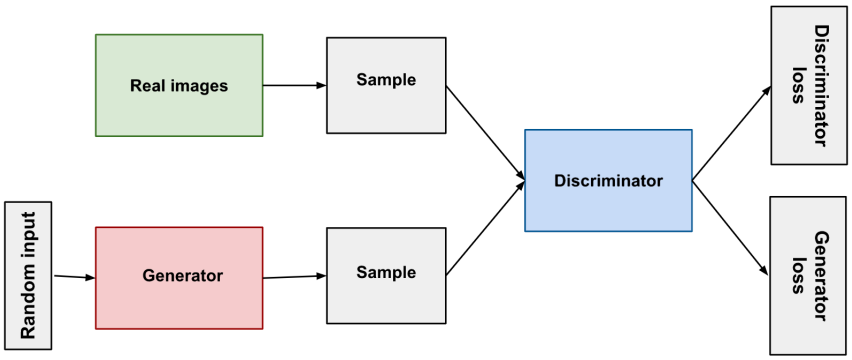


Figure 6: **Gan Structure**

- No clear theoretical background yet
- Hard to know which piece is significantly helping push the cutting edge results
- Ablations for large scale datasets are time-consuming
- Unconditional GANs - not capturing all sample diversity (or mode dropping behavior)
- Evaluation metrics can't capture real generalization capacity

For possible future directions of GANs, please revisit the lecture slides.