Lecture 4 Generative Adversarial Networks

Shiwei Lan[1]

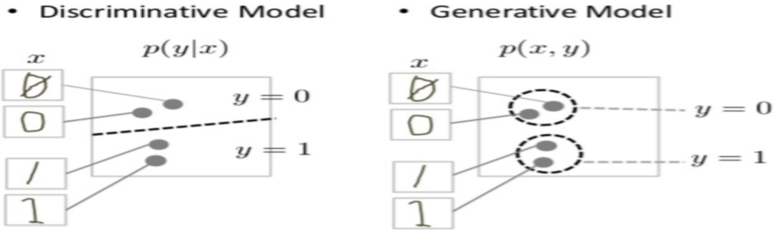
[1]School of Mathematical and Statistical Sciences
Arizona State University

STP598 Advanced Deep Learning Models
Spring 2026

Table of Contents

Generative Models

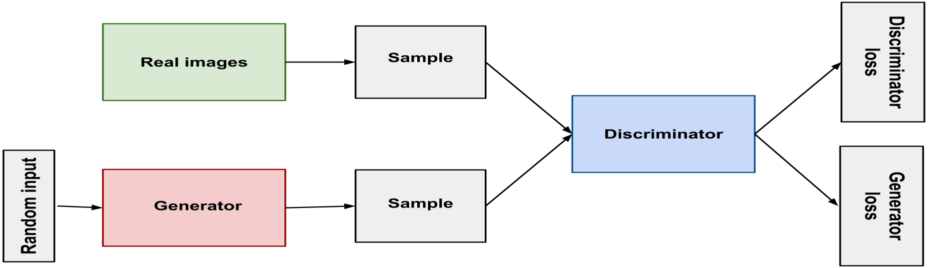
- Deep learning is great at recognition (*discriminative*), but what about creation (*generative*)?
- What do they mean? Given a set of data instances X and a set of labels Y:
  - Discriminative models discriminate between different kinds of data instances.
  - Generative models can generate new data instances.



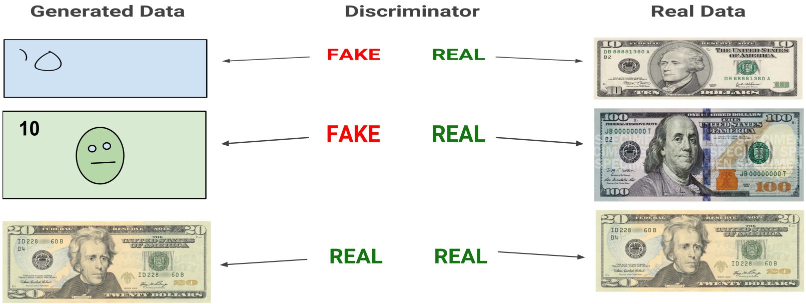
- (Deep) generative models like Boltzmann Machines, Deep Belief Networks, and Variational Autoencoders (VAEs) suffer from complex likelihood approximations, reliance on Markov chains, etc.

Overview of Generative Adversarial Networks (GANs)

- Generative Adversarial Networks (GANs) are an exciting recent innovation in generative models that create fake photos like real human.
- GANs achieve this level of realism by pairing
  - a generator, which learns to produce the target output, with
  - a discriminator, which learns to distinguish true data from the output of the generator.



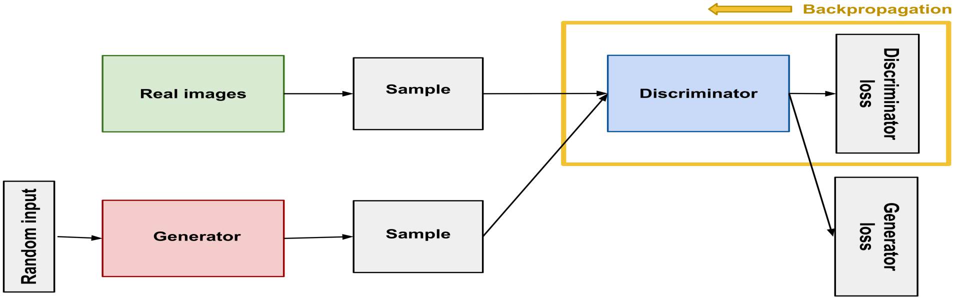
Overview of GAN



- Generator (G): counterfeiter -tries to produce fake currency that looks real.
- Discriminator (D): police - tries to distinguish between real currency and the counterfeiter's fakes.
- Competition drives both to improve until the fakes are indistinguishable from the real thing.

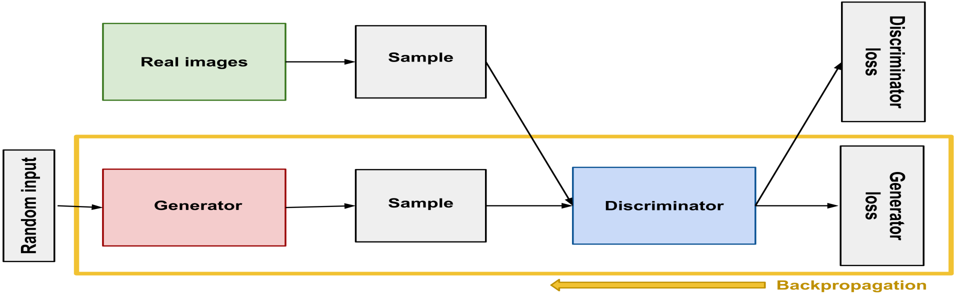
Table of Contents

Discriminator



- The discriminator, $D(x)$, is simply a classifier to distinguish real data from the data created by the generator.
- Any appropriate network can be trained based on:
  - Real data instances, such as real pictures of people, used as positive examples.
  - Fake data instances created by the generator, used as negative examples.
- The discriminator loss penalizes the discriminator for misclassifying a real instance as fake or a fake instance as real.

Generator



- The generator, $G(z)$, learns to fake data by incorporating feedback from the discriminator. It aims to fool the discriminator to classify its output as real.
- Generator training requires tighter integration between the generator and the discriminator, which needs:
  - random input $z \sim p_g(z)$, e.g. $N(0, I)$
  - generator network, which transforms the random input into a data instance
  - discriminator network, which classifies the generated data
  - discriminator output
  - generator loss, which penalizes the generator for failing to fool the discriminator

Table of Contents

Minimax Loss

- GAN trains D to maximize the probability of assigning the correct label to both training examples and samples from G; and also train G to minimize error rate of fake data.
- Goodfellow et al (2014) proposed the cross-entropy loss between the real and generated distributions:

$$V(D,G) = \mathrm{E}_{x\sim p_{\mathrm{data}}(x)}[\log D(x)] + \mathrm{E}_{z\sim p_{\mathrm{z}}(z)}[\log(1-D(G(z)))]$$

- GAN is trained in a mini-max game, i.e. $\min_G \max_D V(D,G)$.

Minimax Loss

Proposition

For given G, the optimal discriminator D is

$$D_G^*(x) = \frac{p_{\mathrm{data}}(x)}{p_{\mathrm{data}}(x) + p_{\mathrm{g}}(x)}$$

Proof.

For given generator G, we maximize the loss $V(D, G)$ which can be rewritten

$$\begin{aligned} V(G, D) &= \int_x p_{\mathrm{data}}(x) \log(D(x)) dx + \int_z p_z(z) \log(1 - D(G(z))) dz \\ &= \int_x [p_{\mathrm{data}}(x) \log(D(x)) + p_{\mathrm{g}}(x) \log(1 - D(x))] dx \end{aligned}$$

Note $f(y) = a \log(y) + b \log(1 - y)$ attains its maximum at $\frac{a}{a+b}$ for $a, b > 0$. $\square$

Minimax Loss

Theorem

The global minimum of the virtual training criterion $V(D_G^*, G)$ *is achieved iff* $p_{\mathrm{g}} = p_{\mathrm{data}}$ *with minimal value* $-\log 4$*.*

Proof.

Substituting $V(D_G^*, G) = \frac{p_{\mathrm{data}}(x)}{p_{\mathrm{data}}(x)+p_{\mathrm{g}}(x)}$ into $V(D, G)$ yields

$$
\begin{aligned}
V(D_G^*, G) &= \mathrm{E}_{x\sim p_{\mathrm{data}}}\left[\log \frac{p_{\mathrm{data}}(x)}{p_{\mathrm{data}}(x)+p_{\mathrm{g}}(x)}\right] + \mathrm{E}_{x\sim p_{\mathrm{g}}}\left[\log \frac{p_{\mathrm{g}}(x)}{p_{\mathrm{data}}(x)+p_{\mathrm{g}}(x)}\right] \\
&= -\log 4 + \mathrm{KL}\left(p_{\mathrm{data}} \middle\| \frac{p_{\mathrm{data}}+p_{\mathrm{g}}}{2}\right) + \mathrm{KL}\left(p_{\mathrm{g}} \middle\| \frac{p_{\mathrm{data}}+p_{\mathrm{g}}}{2}\right) \\
&= -\log 4 + 2\mathrm{JSD}(p_{\mathrm{data}} \| p_{\mathrm{g}}) \geq -\log 4
\end{aligned}
$$

where the Jensen–Shannon divergence JSD attains minimum 0 iff $p_{\mathrm{g}} = \mathrm{data}$. $\square$

Table of Contents

The GAN algorithm

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**
  for k steps **do**
  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)\right].$$

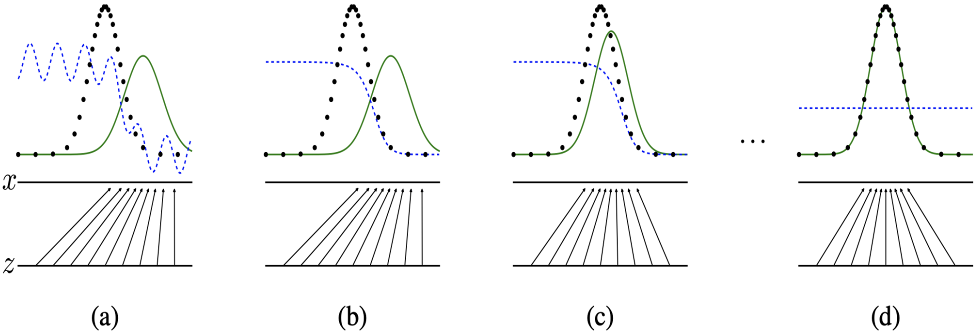
  end for
  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for
The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Training

- In theory, one should maximize the loss $V(D, G)$ w.r.t. D and then minimize the loss $V(D^*_G, G)$ w.r.t. G.
- In practice, we alternate between k steps of optimizing D and one step of optimizing G until it converges to an equilibrium (saddle point).
- Replace minimizing $\log(1 - D(G(z))$ with maximizing $\log(D(G(z))$ to train G to avoid saturation in early stage.



Quick quizzes

True or false: the discriminator network and generator network influence each other solely through the data produced by the generator and the labels produced by the discriminator. When it comes to backpropagation, they are separate networks.

- False
- True

Quick quizzes

True or false: the discriminator network and generator network influence each other solely through the data produced by the generator and the labels produced by the discriminator. When it comes to backpropagation, they are separate networks.

- False
- True

True or false: a typical GAN trains the generator and the discriminator simultaneously.

- False
- True

Quick quizzes

True or false: the discriminator network and generator network influence each other solely through the data produced by the generator and the labels produced by the discriminator. When it comes to backpropagation, they are separate networks.

- False
- True

True or false: a typical GAN trains the generator and the discriminator simultaneously.

- False
- True

True or false: a GAN always uses the same loss function for both discriminator and generator training.

- False
- True

Challenges

- **Vanishing Gradients**: if discriminator is too good, then generator training can fail due to vanishing gradients.
  - Wasserstein loss (Arjovsky et al 2017): designed to prevent vanishing gradients even when you train the discriminator to optimality
  - Modified minimax loss
- **Mode Collapse**: if a generator produces an especially plausible output, it may learn to produce *only* that output.
  - Wasserstein loss: alleviates mode collapse by letting you train the discriminator to optimality without worrying about vanishing gradients.
  - Unrolled GANs (Metz et al, 2017): uses a generator loss function that incorporates not only the current, but also future discriminator's classifications.
- **Failure to Converge**: as the generator improves with training, the discriminator deteriorates whose feedback gets less meaningful over time.
  - Adding noise to discriminator inputs: Arjovsky and Bottou (2017).
  - Penalizing discriminator weights: Roth et al (2017)

GAN Variations

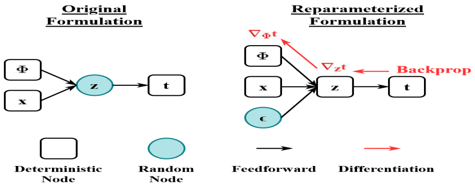
- **Progressive GANs** (Karras et al, 2017): the generator's first layers produce very low resolution images, and subsequent layers add details. Faster training for higher resolution images.
- **Conditional GANs** (Mirza et al, 2014): trains on a labeled data set and let you specify the label for each generated instance. Instead of modeling the joint probability $p(X, Y)$, it models the conditional probability $P(X|Y)$.
- **Image-to-Image Translation** (Isola et al, 2016): takes an image as input and map it to a generated output image with different properties.
- **CycleGAN** (Zhu et al, 2017): learns to transform images from one set into images that could plausibly belong to another set.
- **Text-to-Image Synthesis** (Zhang et al, 2016): takes text as input and produce images that are plausible and described by the text.
- **Super-resolution** (Ledig et al, 2017): increases the resolution of images, adding detail where necessary to fill in blurry areas.
- **Face Inpainting** (Yeh et al, 2017): chunks of an image are blacked out, and the system tries to fill in the missing chunks.

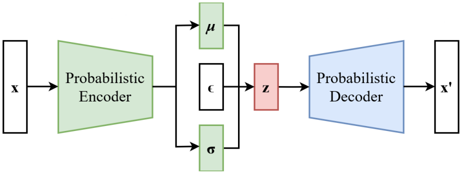
Table of Contents

VAE review

- What is variational auto-encoder (VAE, Kingma and Welling 2014)? How does it compare with GAN?
- VAE consists of *probabilistic*
  - **Encoder**: $x \to q_\phi(z|x) \sim N(E_\phi(x), \Sigma_\phi^2)$
  - **Decoder**: $z \to p_\theta(x|z) \sim N(D_\theta(z), I)$
- It approximates the posterior distribution $p_\theta(z|x)$ with variational distribution $q_\phi(z|x)$ to by optimizing the evidence lower bound (ELBO):

$$L_{\theta,\phi}(x) := \underbrace{\mathrm{E}_{z\sim q_\phi(\cdot|x)}[\log p_\theta(x|z)]}_{\text{reconstruction error}} - \underbrace{\mathrm{D}_{KL}(q_\phi(\cdot|x)||p_\theta(\cdot))}_{\text{regularization}}$$





GAN vs VAE

- Both are generative models, use latent variables, trained with neural networks, and aim at "implicit density learning".
- However, they differ philosophically:
  - VAE = probabilistic, likelihood-based
  - GAN = adversarial, game-based

Aspect	GAN	VAE
Training objective	Adversarial game	ELBO (likelihood-based)
Likelihood	Implicit / none	Explicit (approximate)
Sample quality	Very sharp	Often blurry
Training stability	Often unstable	Stable
Latent space	Often less structured	Structured, continuous
Mode coverage	Can miss modes	Good
Inference $x \to z$	Not naturally	Yes (encoder)

Implementation of GANs

- In PyTorch, there is a good tutorial on DCGAN trained on Celeb-A Faces dataset.
- Here is another GAN trained on MNIST data set https://www.geeksforgeeks.org/deep-learning/generative-adversarial-networks-gans-in-pytorch/.
- This is HiFi-GAN that synthesizes speech https://pytorch.org/hub/nvidia_deeplearningexamples_hifigan/.
- Kaggle has a good tutorial for beginner https://www.kaggle.com/code/songseungwon/pytorch-gan-basic-tutorial-for-beginner.
- We will demo a GAN on sol.