Lecture11
Introducción a las GANs
(Generative Adversarial Networks)

Statistics > Machine Learning

[Submitted on 10 Jun 2014]

Generative Adversarial Networks

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio

We propose a new framework for estimating generative models via an adversarial process, in which we simultaneously train two models: a generative model G that captures the data distribution, and a discriminative model D that estimates the probability that a sample came from the training data rather than G. The training procedure for G is to maximize the probability of D making a mistake. This framework corresponds to a minimax two-player game. In the space of arbitrary functions G and D, a unique solution exists, with G recovering the training data distribution and D equal to 1/2 everywhere. In the case where G and D are defined by multilayer perceptrons, the entire system can be trained with backpropagation. There is no need for any Markov chains or unrolled approximate inference networks during either training or generation of samples. Experiments demonstrate the potential of the framework through qualitative and quantitative evaluation of the generated samples.

https://arxiv.org/abs/1406.2661

https://thiscatdoesnotexist.com/

https://thispersondoesnotexist.com/

https://thisstartupdoesnotexist.com/

https://thisponydoesnotexist.net/

Introducción a las GANs

1. Idea principal detrás de las GANs
2. Objetivo de las GAN
3. Modificación de la función de costo en la práctica
4. Ejemplo: GAN para generación de dígitos escritos a mano en PyTorch
5. Consejos y trucos para hacer que las GAN funcionen
6. Una DCGAN para generar imágenes faciales en PyTorch

Dejar que dos redes neuronales compitan entre sí

1. **Idea principal detrás de las GANs**
2. Objetivo de las GAN
3. Modificación de la función de costo en la práctica
4. Ejemplo: GAN para generación de dígitos escritos a mano en PyTorch
5. Consejos y trucos para hacer que las GAN funcionen
6. Una DCGAN para generar imágenes faciales en PyTorch

Redes generativas antagónicas (GAN)

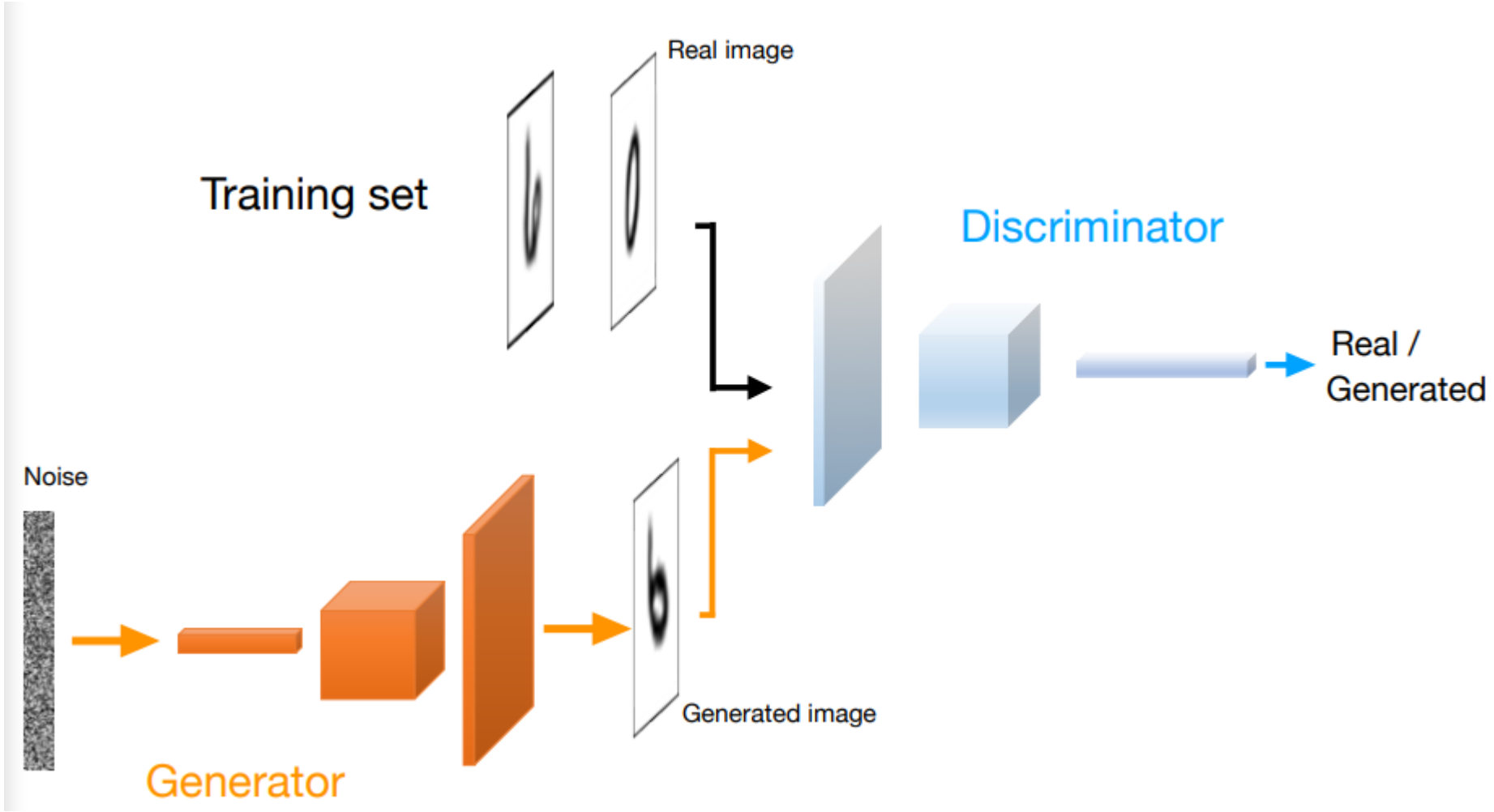
- El propósito de los modelos generativos es generar nuevos datos.

- Inicialmente para generar nuevas imágenes, pero aplicable a una amplia gama de dominios

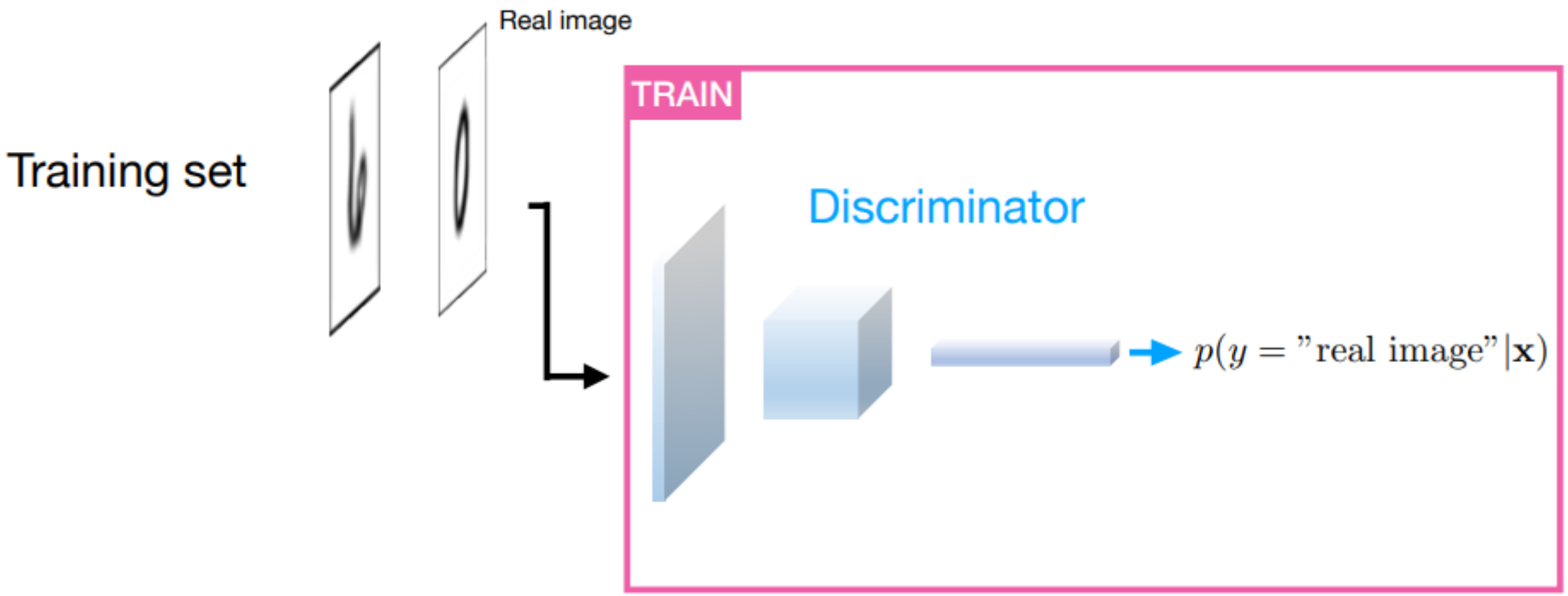
- Aprende la distribución del conjunto de entrenamiento y puede generar nuevas imágenes que nunca antes se habían visto.

- Similar a VAE, y a diferencia de, por ejemplo, los modelos autorregresivos o RNNs (que generan una palabra a la vez), las GAN generan toda la salida de una vez.

GAN convolucional profunda (DCGAN o simplemente GAN)

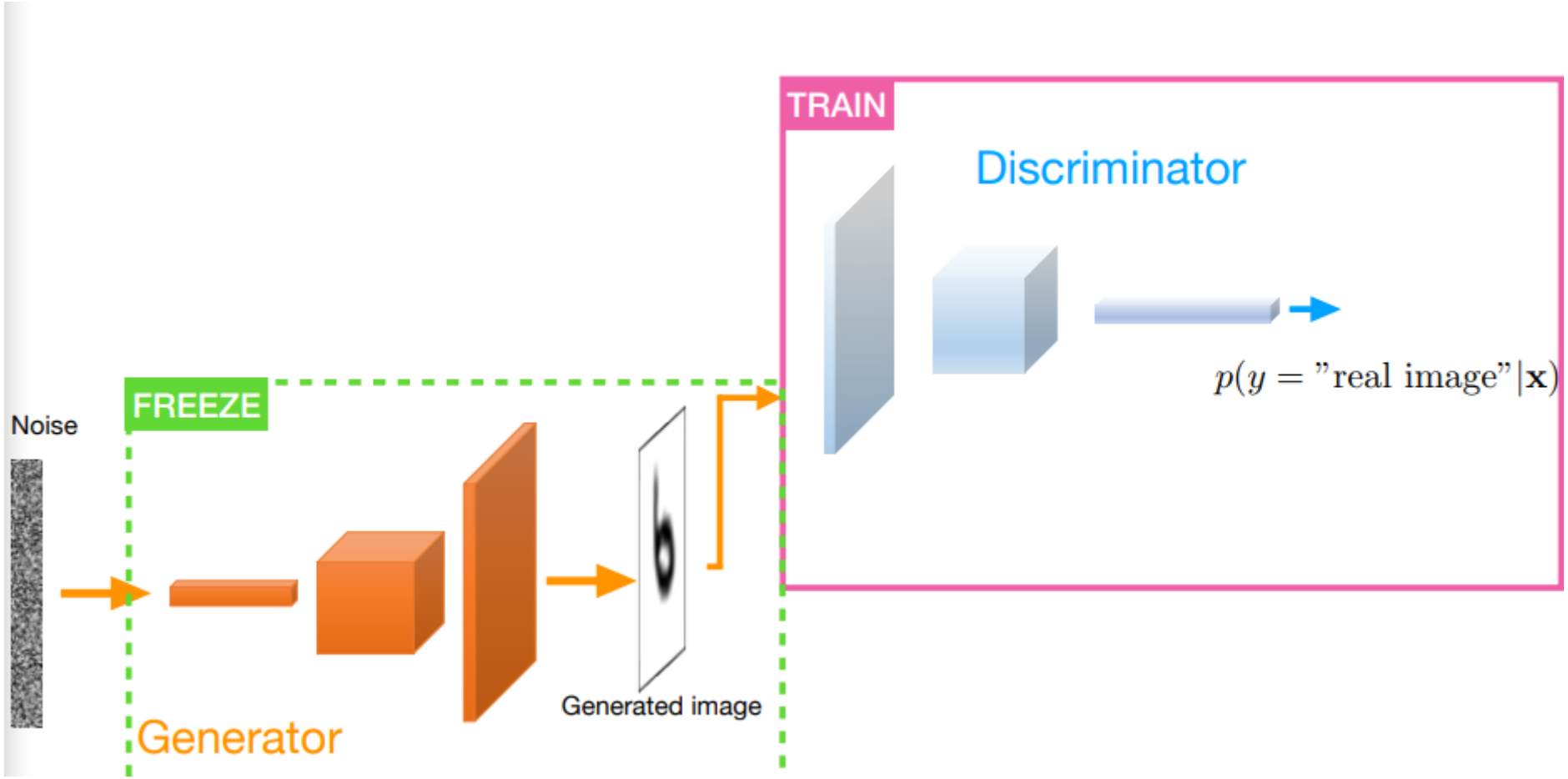


Paso 1.1: Discriminador



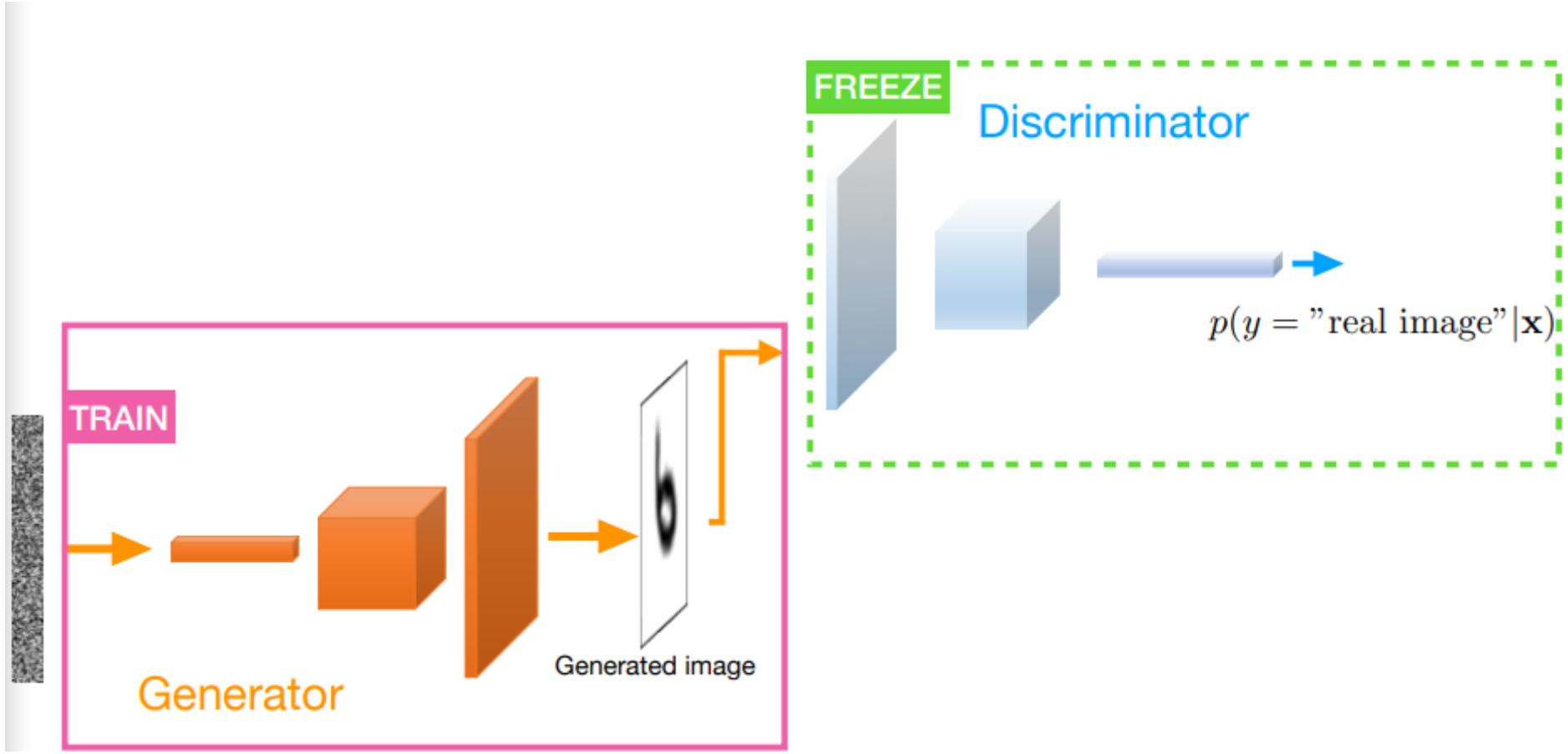
Entrena para predecir que la imagen real es real

Paso 1.2: Discriminador



Entrena para predecir que la imagen falsa es falsa

Paso 2: Generador



Entrena para generar imágenes similares a las imágenes de entrenamiento

Juego Adversario

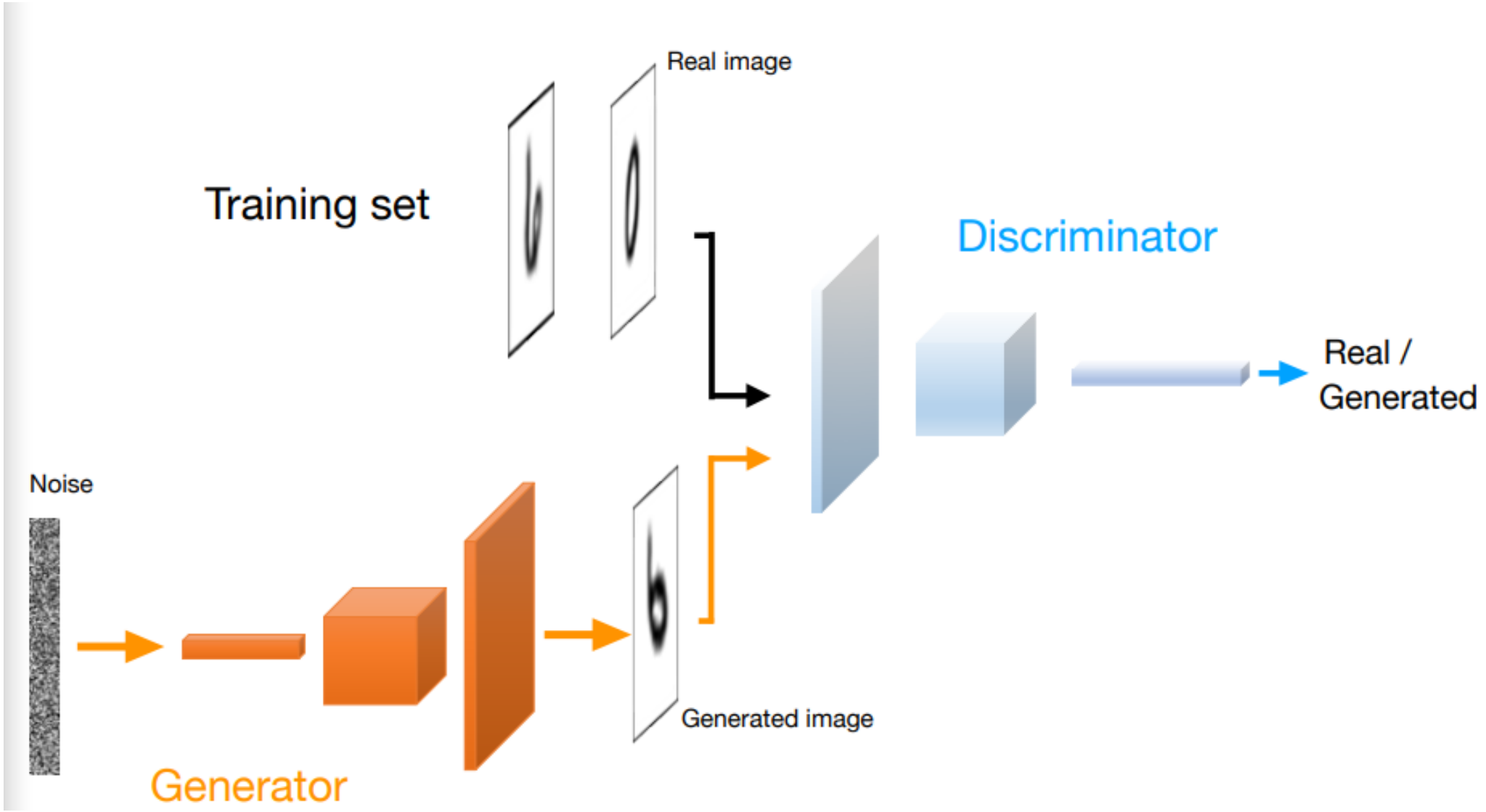
Discriminador: Aprende a ser mejor para distinguir imágenes reales de las generadas.

Generador: Aprende a generar mejores imágenes para engañar al discriminador

¿Cómo se ven las funciones de pérdida?

1. Idea principal detrás de las GANs
2. **Objetivo de las GAN**
3. Modificación de la función de costo en la práctica
4. Ejemplo: GAN para generación de dígitos escritos a mano en PyTorch
5. Consejos y trucos para hacer que las GAN funcionen
6. Una DCGAN para generar imágenes faciales en PyTorch

¿Cuándo converge una GAN?



Objetivo de las GAN

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

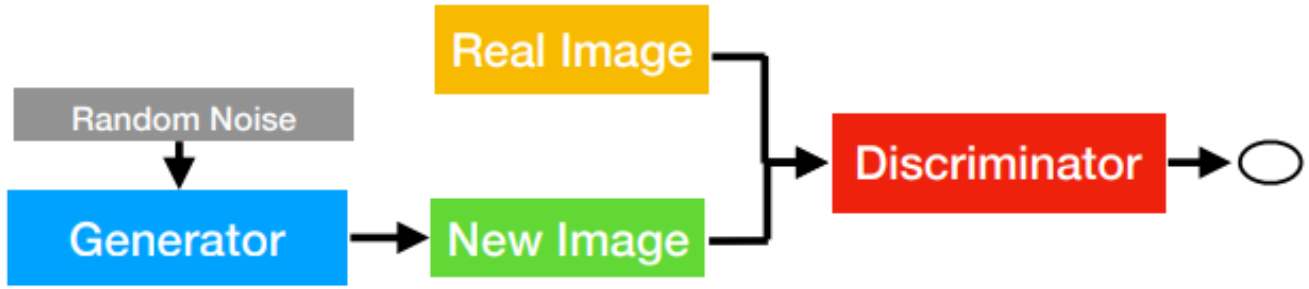
$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}\ (\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

Gradiente discriminador para actualización (ascenso de gradiente):

predecir bien en imágenes reales
=> quiere probabilidad cercana a 1

predecir bien en imágenes falsas
=> quiere probabilidad cercana a 0

$$\nabla_{\mathbf{w}_D} \frac{1}{n} \sum_{i=1}^{n} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right]$$

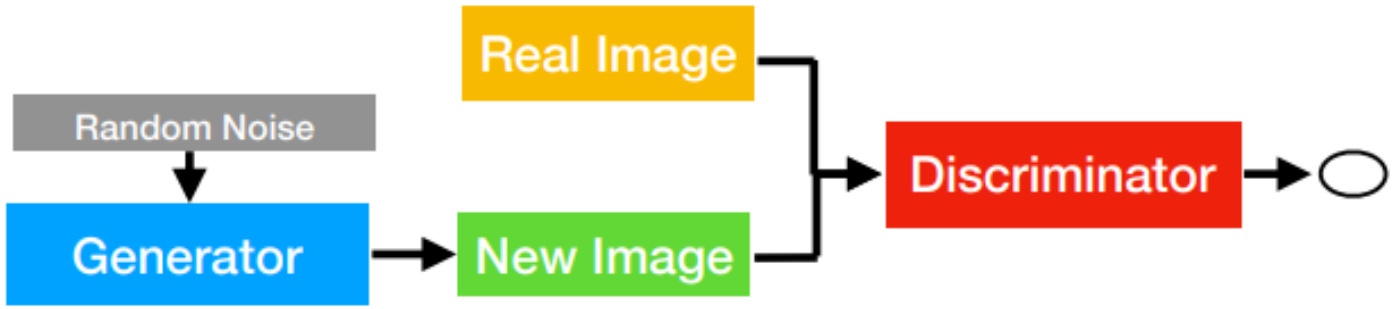


$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}\ (\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

Gradiente discriminador para actualización (descenso de gradiente):

predecir mal en imágenes falsas
=> quiere probabilidad cercana a 1

$$\nabla_{\mathbf{w}_G} \frac{1}{n} \sum_{i=1}^{n} \log \overbrace{\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)}$$



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 $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ 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 $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ 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.

Goodfellow, Ian, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. "Generative Adversarial Nets." In Advances in Neural Information Processing Systems, pp. 2672-2680. 2014.

http://papers.nips.cc/paper/5423-generative-adversarial-nets

Convergencia GAN

- Converge cuando se alcanza el equilibrio de Nash (concepto de teoría de juegos) en el juego minmax (suma cero)

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

- El equilibrio de Nash en la teoría de juegos se alcanza cuando las acciones de un jugador no cambian dependiendo de las acciones del oponente.

- Aquí, esto significa que la GAN produce imágenes realistas y el discriminador genera predicciones aleatorias (probabilidades cercanas a 0.5)

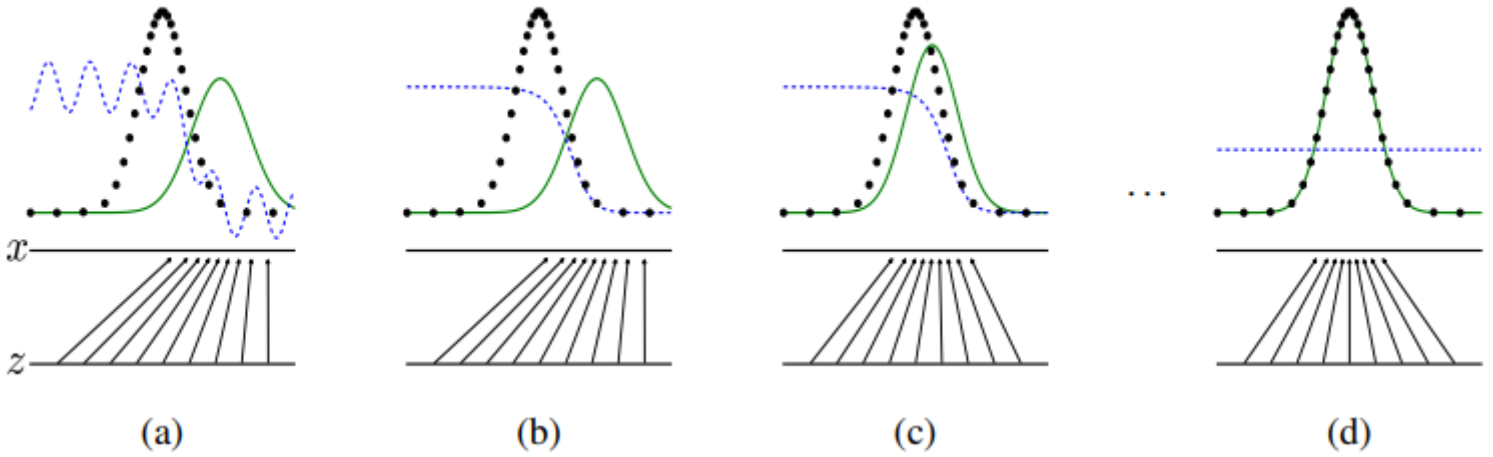


Figure 1: Generative adversarial nets are trained by simultaneously updating the **d**iscriminative distribution (D, blue, dashed line) so that it discriminates between samples from the data generating distribution (black, dotted line) $p_{\boldsymbol{x}}$ from those of the **g**enerative distribution p_g (G) (green, solid line). The lower horizontal line is the domain from which $\boldsymbol{z}$ is sampled, in this case uniformly. The horizontal line above is part of the domain of $\boldsymbol{x}$. The upward arrows show how the mapping $\boldsymbol{x} = G(\boldsymbol{z})$ imposes the non-uniform distribution p_g on transformed samples. G contracts in regions of high density and expands in regions of low density of p_g. (a) Consider an adversarial pair near convergence: p_g is similar to p_{data} and D is a partially accurate classifier. (b) In the inner loop of the algorithm D is trained to discriminate samples from data, converging to $D^*(\boldsymbol{x}) = \frac{p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_g(\boldsymbol{x})}$. (c) After an update to G, gradient of D has guided $G(\boldsymbol{z})$ to flow to regions that are more likely to be classified as data. (d) After several steps of training, if G and D have enough capacity, they will reach a point at which both cannot improve because $p_g = p_{\text{data}}$. The discriminator is unable to differentiate between the two distributions, i.e. $D(\boldsymbol{x}) = \frac{1}{2}$.

Goodfellow, Ian, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. "Generative Adversarial Nets." In Advances in Neural Information Processing Systems, pp. 2672-2680. 2014.

http://papers.nips.cc/paper/5423-generative-adversarial-nets

¿Cómo se ven las funciones de pérdida?

1. Idea principal detrás de las GANs
2. Objetivo de las GAN
3. **Modificación de la función de costo en la práctica**
4. Ejemplo: GAN para generación de dígitos escritos a mano en PyTorch
5. Consejos y trucos para hacer que las GAN funcionen
6. Una DCGAN para generar imágenes faciales en PyTorch

Problemas de entrenamiento de las GAN

- Oscilación entre la pérdida del generador y el discriminador

- Colapso de modo (el generador solo produce ejemplos de un tipo particular)

- El discriminador es demasiado fuerte, de modo que el gradiente del generador desaparece deja de aprender.

- El discriminador es demasiado débil y el generador produce imágenes no realistas que lo engañan con demasiada facilidad (aunque es un problema poco común)

Problemas de entrenamiento de las GAN

- El discriminador es demasiado fuerte, de modo que el gradiente del generador desaparece.

- Puede arreglarse de la siguiente manera:

En lugar de descenso de gradiente con

$$\nabla_{\mathbf{w}_G} \frac{1}{n} \sum_{i=1}^{n} \log \left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)$$

Haga ascenso de gradiente con

$$\nabla_{\mathbf{w}_G} \frac{1}{n} \sum_{i=1}^{n} \log \left(D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)$$

Función de pérdida de GAN en la práctica

Discriminador

- Maximiza la probabilidad de predicción de clasificar lo real como real y lo falso como falso
- Recuerde que maximizar la probabilidad logarítmica es lo mismo que minimizar la probabilidad logarítmica negativa (es decir, minimizar la entropía cruzada)

Generador

- Minimizar la probabilidad de que el discriminador haga predicciones correctas (predecir lo falso como falso; lo real como real), lo que se puede lograr maximizando la entropía cruzada.
- Sin embargo, esto no funciona bien en la práctica debido a problemas de gradiente pequeños
- Mejor: Voltee las etiquetas y minimice la entropía cruzada (obligue al discriminador a generar una alta probabilidad real si una imagen es falsa)

Ascenso de gradiente

predecir bien en imágenes reales
=> quiere probabilidad cercana a 1

predecir bien en imágenes falsas
=> quiere probabilidad cercana a 0

$$\nabla_{\mathbf{w}_D} \frac{1}{n} \sum_{i=1}^{n} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right]$$

<u>Objetivo discriminador</u> en negativo. Perspectiva de probabilidad logarítmica (entropía cruzada binaria):

Real images, $y = 1$

$$\mathcal{L}(\mathbf{w}) = -y^{(i)} \log\left(\hat{y}^{(i)}\right) - \left(1 - y^{(i)}\right) \log\left(1 - \hat{y}^{(i)}\right)$$

Want, $\hat{y} = 1$

Fake images, $y = 0$

$$\mathcal{L}(\mathbf{w}) = -y^{(i)} \log\left(\hat{y}^{(i)}\right) - \left(1 - y^{(i)}\right) \log\left(1 - \hat{y}^{(i)}\right)$$

Want, $\hat{y} = 0$

Gradiente descendente con

$$\nabla_{\mathbf{w}_G} \frac{1}{n} \sum_{i=1}^{n} \log \left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)$$

<u>Objetivo generador</u> en negativo. Perspectiva de probabilidad logarítmica (entropía cruzada binaria):

Fake images, $y = 0$

$$\mathcal{L}(\mathbf{w}) = -y^{(i)} \log\left(\hat{y}^{(i)}\right) - \left(1 - y^{(i)}\right) \log\left(1 - \hat{y}^{(i)}\right)$$

Want, $\hat{y} = 0$

Cambie el signo a "+" para que se convierta en "want $\hat{y} = 1$"

Es mejor dar la vuelta a las etiquetas en lugar del signo.

gradient descent with

$$\nabla \mathbf{w}_G \frac{1}{n} \sum_{i=1}^{n} \log \left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)$$

Generator objective in the neg. log-likelihood (binary cross entropy) perspective:

Fake images, $y = 0$

$$\mathcal{L}(\mathbf{w}) = -y^{(i)} \log\left(\hat{y}^{(i)}\right) - \left(1 - y^{(i)}\right) \log\left(1 - \hat{y}^{(i)}\right)$$

Want, $\hat{y} = 0$

Flip sign to "+" so that it turns into "want $\hat{y} = 1$"

Haga Gradiente ascendente con

$$\nabla_{\mathbf{w}_G} \frac{1}{n} \sum_{i=1}^{n} \log \left(D \left(G \left(\boldsymbol{z}^{(i)} \right) \right) \right)$$

Y cambie las etiquetas

<u>Objetivo generador</u> en negativo. Perspectiva de probabilidad logarítmica (entropía cruzada binaria):

etiqueta de imagen falsa volteada -> etiqueta de imagen real, y = 1

$$\mathcal{L}(\mathbf{w}) = -y^{(i)} \log \left(\hat{y}^{(i)} \right) - \left(1 - y^{(i)} \right) \log \left(1 - \hat{y}^{(i)} \right)$$

Want, $\hat{y} = 1$

Implementando nuestra primera GAN

1. Idea principal detrás de las GANs
2. Objetivo de las GAN
3. Modificación de la función de costo en la práctica
4. **Ejemplo: GAN para generación de dígitos escritos a mano en PyTorch**
5. Consejos y trucos para hacer que las GAN funcionen
6. Una DCGAN para generar imágenes faciales en PyTorch

Implementando nuestra primera GAN

1. Idea principal detrás de las GANs
2. Objetivo de las GAN
3. Modificación de la función de costo en la práctica
4. Ejemplo: GAN para generación de dígitos escritos a mano en PyTorch
5. **Consejos y trucos para hacer que las GAN funcionen**
6. Una DCGAN para generar imágenes faciales en PyTorch

https://github.com/soumith/ganhacks

¿Cómo se ven las funciones de pérdida?

1. Idea principal detrás de las GANs
2. Objetivo de las GAN
3. Modificación de la función de costo en la práctica
4. Ejemplo: GAN para generación de dígitos escritos a mano en PyTorch
5. Consejos y trucos para hacer que las GAN funcionen
6. Una DCGAN para generar imágenes faciales en PyTorch

GAN convolucional profundo

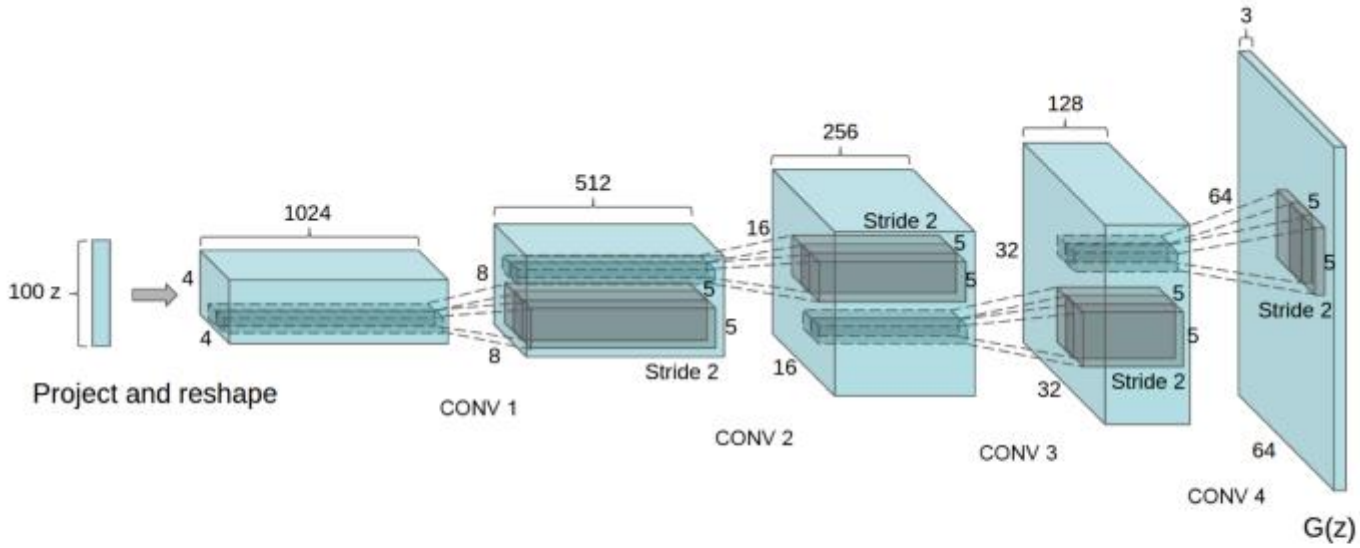


Figure 1: DCGAN generator used for LSUN scene modeling. A 100 dimensional uniform distribution Z is projected to a small spatial extent convolutional representation with many feature maps. A series of four fractionally-strided convolutions (in some recent papers, these are wrongly called deconvolutions) then convert this high level representation into a 64×64 pixel image. Notably, no fully connected or pooling layers are used.

Radford, A., Metz, L., & Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv preprint arXiv:1511.06434.

https://arxiv.org/abs/1511.06434