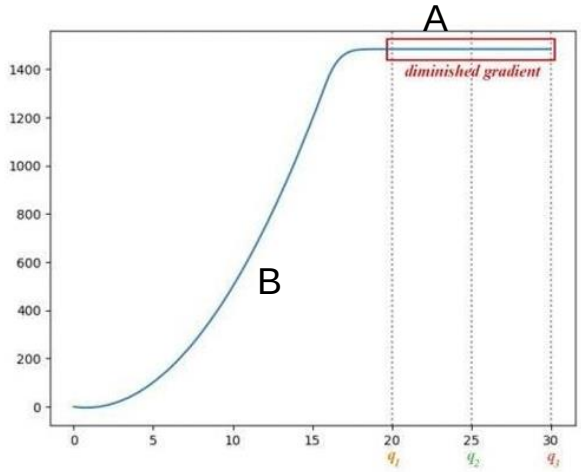
Wasserstein GANs

Wasserstein GANs

- Motivadas por la complejidad del entrenamiento tradicional de las GANs (juego de suma cero con equilibrio inestable).
- La función de pérdida penaliza el modelo proporcionalmente a la distancia entre la distribución de probabilidad predicha y la distribución de probabilidad esperada para un dato determinado.
- La función de pérdida entropía cruzada binaria es apta para entrenar al discriminador, pero puede ser poco útil para entrenar el generador.
    - Problema de desaparición del gradiente si las muestras generadas son muy lejanas a los datos reales

Wasserstein GANs

- La entropía cruzada binaria se corresponde con la divergencia de Kullback–Leibier para comparar datos reales y generados.
- D_{KL} en función de la distancia entre distribuciones
  - Cuanto más lejos está P de Q, D_{KL} crece hasta un valor máximo donde los gradientes desaparecen
  - En B los gradientes aportan información para el entrenamiento, en A no.



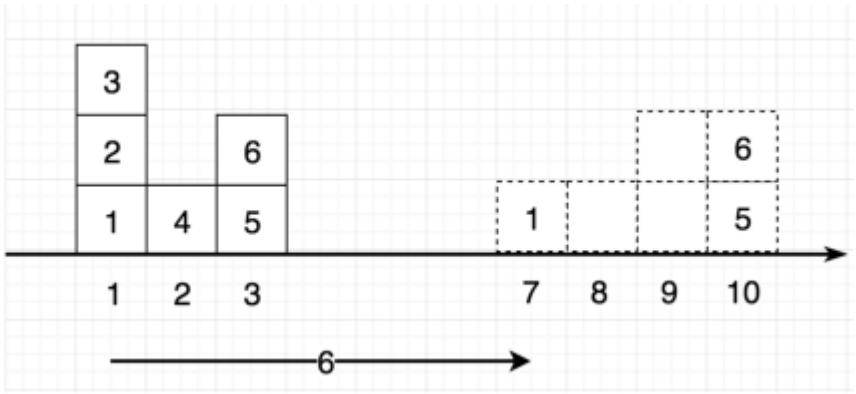
- Las WGAN proponen un modelo alternativo para el entrenamiento del generador, intentando lograr una mejor aproximación de la distribución de datos observada en cada conjunto de entrenamiento.

Wasserstein GANs

- En lugar de utilizar un discriminador para clasificar o predecir la probabilidad de que los datos generados sean reales o falsos, una WGAN reemplaza el discriminador con un crítico, una ANN que puntúa la realidad (o falsedad) de un dato determinado.
- Modifica la métrica de distancia original de las GANs (divergencia de Kullback–Leibier o Jensen-Shannon) por la distancia de Wasserstein (función más suave).
- La diferencia fundamental de la distancia de Wasserstein es su impacto en la convergencia de las series de distribuciones de probabilidad de los datos generados.

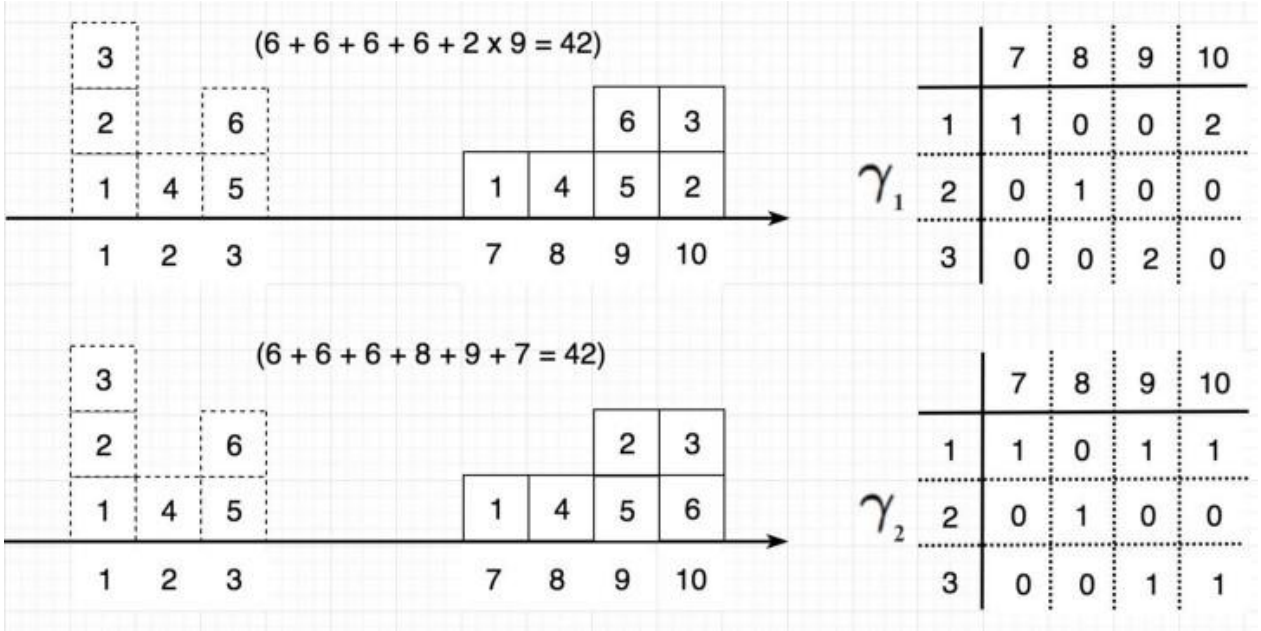
Wasserstein GANs

- Idea: utilizar una métrica específica (distancia de Wasserstein) para evaluar las diferencias entre distribuciones de probabilidad.
- La métrica también se conoce como “la distancia del movimiento de tierra” (Earth mover's distance) o distancia de Kantorovich-Rubinstein.
- Mínimo costo de transformar una pila (distribución) en otra: el costo es la cantidad de tierra (volumen) movida por la distancia que se mueve.



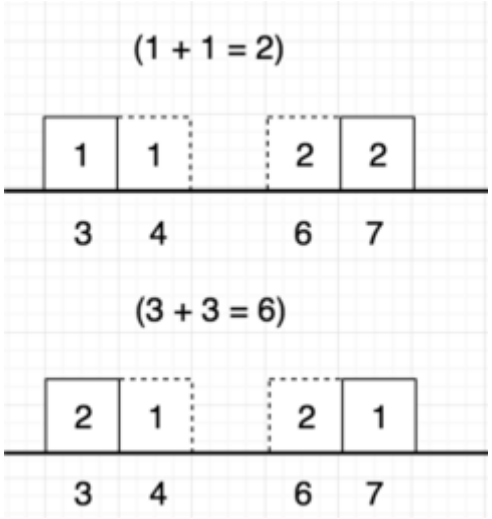
Earth mover’s distance

- Hay muchos planes posibles para la transformación.
- No todos los planes tienen el mismo costo.
- Diferentes planes pueden tener el mismo costo.



Earth mover’s distance

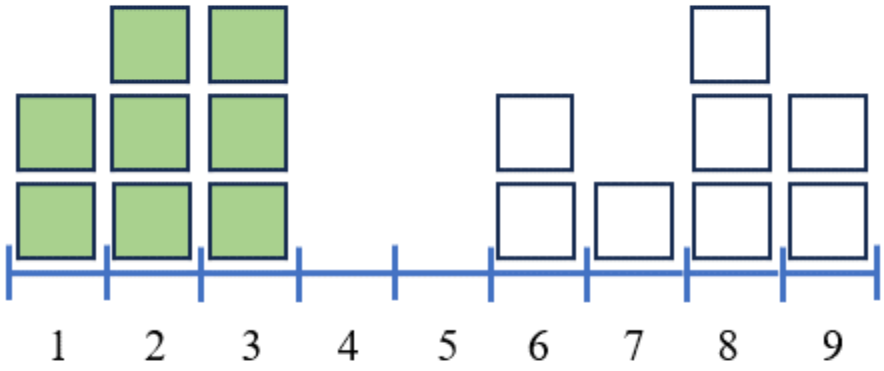
- Hay muchos planes posibles para la transformación.
- No todos los planes tienen el mismo costo.
- La distancia de Wasserstein es el mínimo costo de todos los planes.

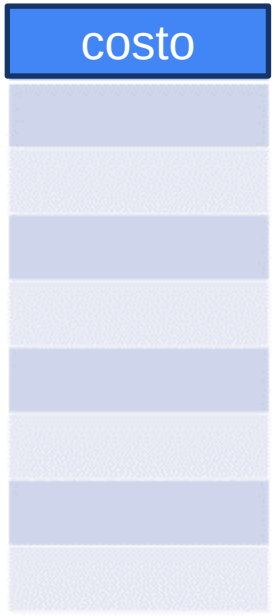


Distancia de Wasserstein

- No todos los planes tienen el mismo costo.
- La distancia de Wasserstein es el mínimo costo de todos los planes

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}\left[\|x - y\| \right]$$





Distancia de Wasserstein

- La distancia de Wasserstein es el mínimo costo de todos los planes

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}\left[\, \|x - y\| \,\right]$$

- En las WGAN la distancia de Wasserstein se utiliza para evaluar la distancia entre la distribución de los datos reales y la distribución de los datos sintéticos generados
- Se aplica la dualidad de Kantorovich–Rubinstein

$$W(\mathbb{P}_r, \mathbb{P}_\theta) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x\sim\mathbb{P}_r}[f(x)] - \mathbb{E}_{x\sim\mathbb{P}_\theta}[f(x)]$$

Distancia de Wasserstein

$$W(\mathbb{P}_r, \mathbb{P}_\theta) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x \sim \mathbb{P}_r}[f(x)] - \mathbb{E}_{x \sim \mathbb{P}_\theta}[f(x)]$$

- $\mathbb{P}_\theta$ es una densidad paramétrica
- En lugar de estimar la densidad de $\mathbb{P}_g$ se trabaja con una variable aleatoria Z con distribución $p(z)$ que se pasa a una función paramétrica $g_\theta : Z \to X$ que genera muestras de la distribución $\mathbb{P}_\theta$
- La idea es variar el parámetro θ para intentar aproximarse lo más posible a $\mathbb{P}_r$
- Típicamente g_θ es una ANN, como en las GAN (el generador)

Distancia de Wasserstein

- Es una función continua y casi diferenciable (en todos los puntos del dominio), que permite realizar entrenamientos más cercanos al óptimo.
- Es una métrica relevante: converge a 0 a medida que las distribuciones se acercan entre sí y diverge a medida que se alejan.
- Es más estable como función objetivo que la divergencia tradicional (Jensen-Shannon), que se satura localmente a medida que mejora el discriminador, causando la desaparición de gradientes.
- También ayuda a mitigar problemas de colapso de modo.

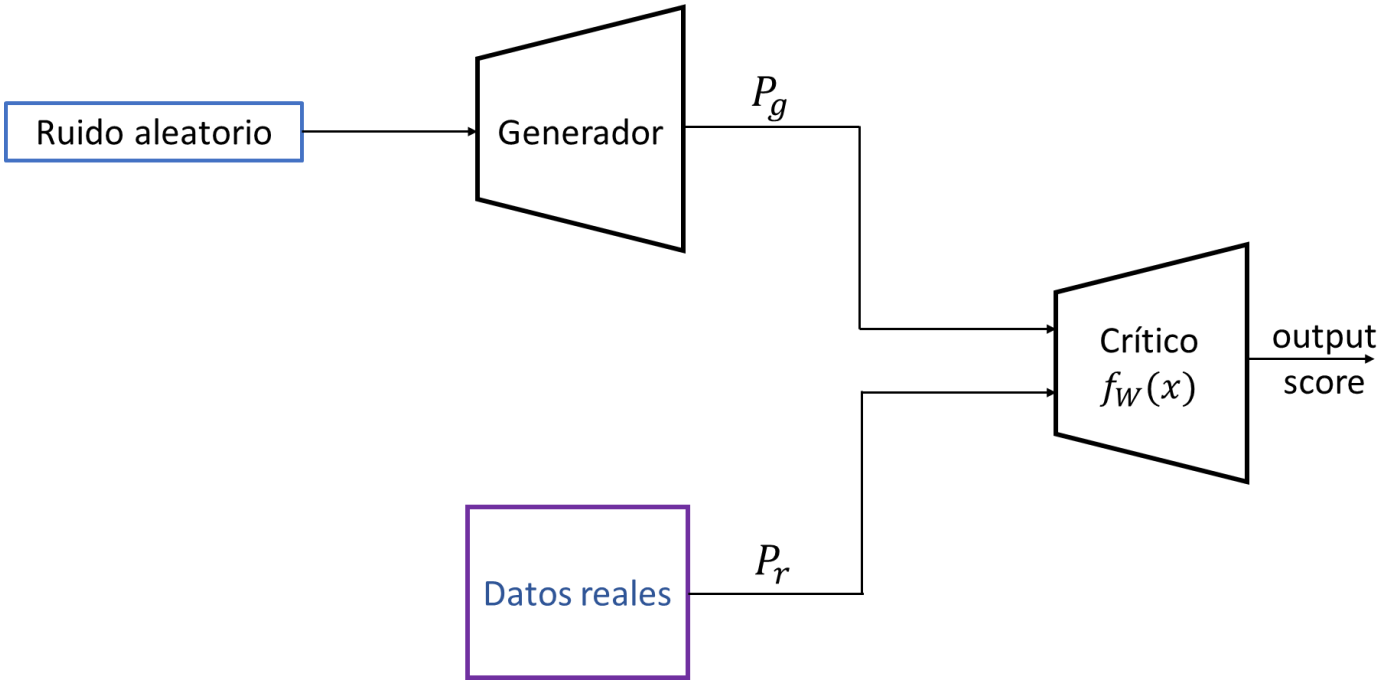
Wasserstein GANs

- Crítico (critical) en lugar de discriminador
- El crítico trata de determinar una función de ajuste (*fitting*) $f_W(x)$ que permite aproximar la distribución de los datos reales (valor esperado del 'output score')
  - $f_W(x)$ se aplica a batches de datos reales, para calcular el 'output score' de cada muestra. Con los resultados se estima la distribución de datos reales
  - El mismo procedimiento se aplica para los datos sintéticos generados
- El objetivo es maximizar la distancia de Wasserstein entre las distribuciones empíricas calculadas

$$L(P_r, P_g) = \mathbb{E}_{x \in P_r}[f_w(x)] - \mathbb{E}_{z \in P_y(z)}[f_w(G_\theta(z))]$$

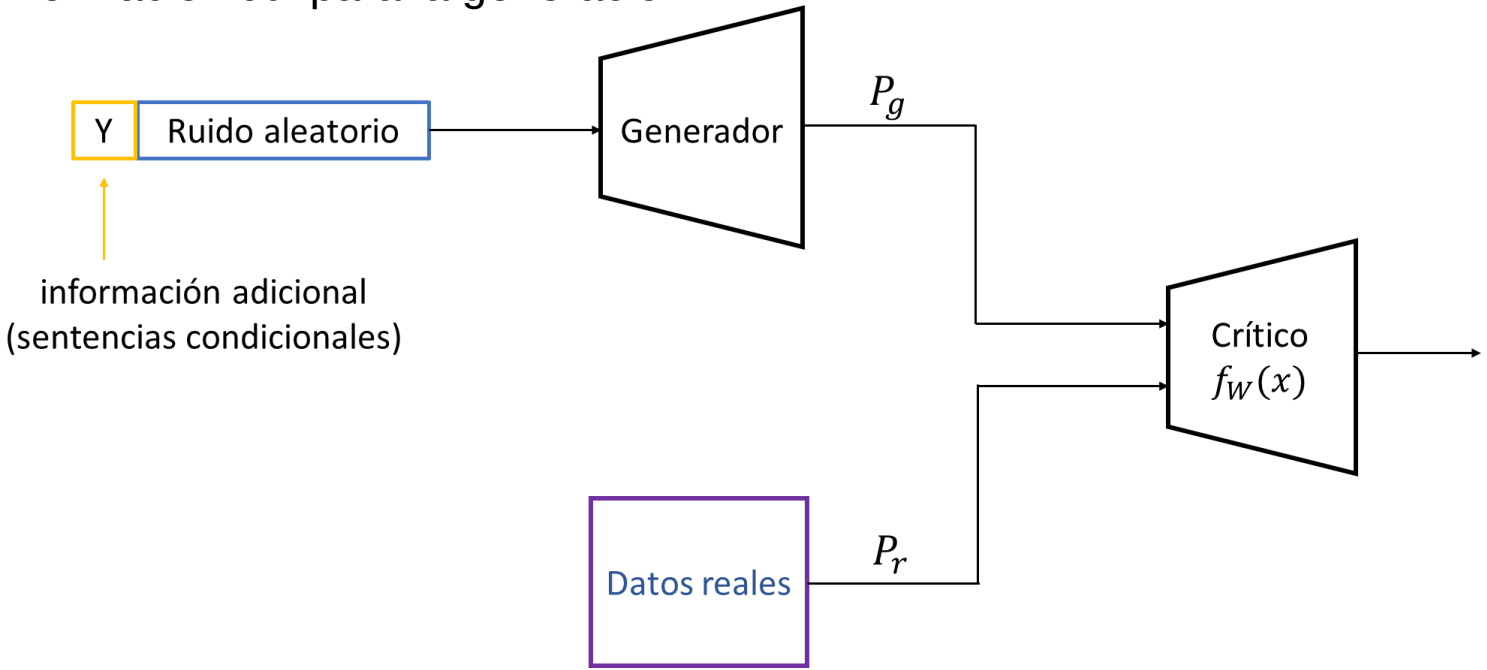
Wasserstein GANs

- Si $W(P_g - P_r) < \varepsilon$, las distribuciones pueden considerarse iguales y los datos generados seguirán la misma distribución que los reales.



Wasserstein GANs

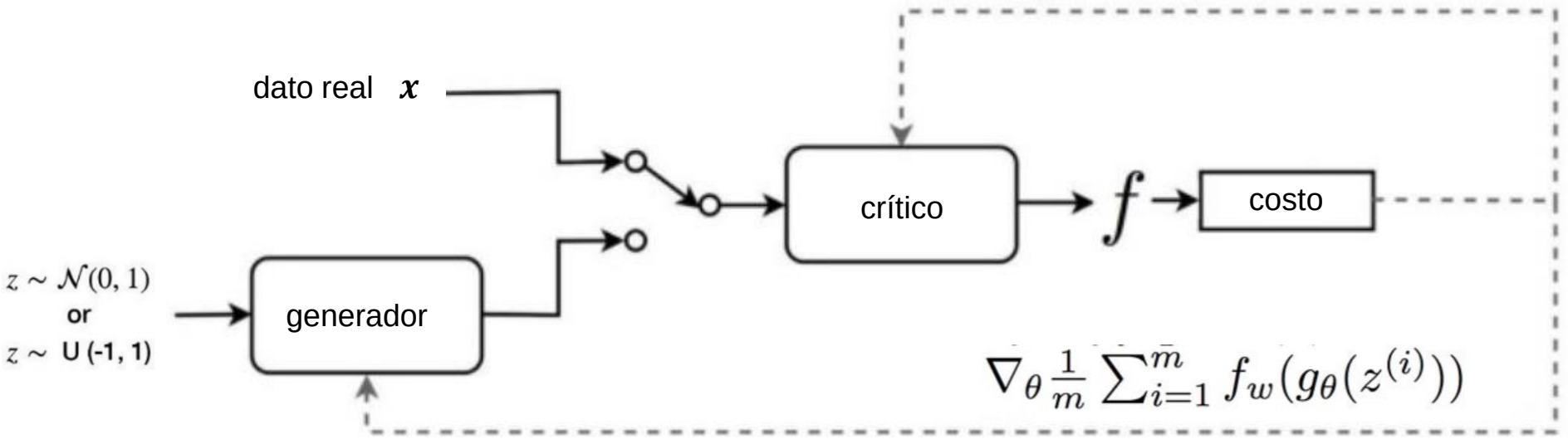
- El modelo admite incluir condiciones, para trabajar con WGANs condicionales
  - Por ejemplo, puntos con determinadas propiedades que proporcionen información útil para la generación



Esquema general del entrenamiento de una Wasserstein GAN

Loss = diferencia de average score entre datos reales y sintéticos

$$\nabla_w \left[\frac{1}{m} \sum_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m} \sum_{i=1}^{m} f_w(g_\theta(z^{(i)})) \right]$$



Loss = average score en datos sintéticos

WGAN: entrenamiento del crítico

- El objetivo del crítico es estimar la distancia de Wasserstein, resolviendo el problema de optimización para hallar los valores de w que definen la función de ajuste $f_W(x)$ que aproxima la distribución de los datos reales

$$L_{crit}(w) = \max_{w \in W} \mathbb{E}_{x \in P_r}[f_w(x)] - \mathbb{E}_{z \in P_y(z)}[f_w(G_\theta(z))]$$

  - Para asegurar la condición de Lipschitz, se restringen los valores de w a un espacio compacto, por ejemplo, un rango pequeño, mediante recorte (clipping)
- La principal diferencia entre el crítico de una WGAN y el discriminador estándar de una GAN es que el discriminador estándar se entrena para identificar/discriminar las muestras reales de las sintéticas, y el crítico de una WGAN estima la distancia de Wasserstein entre las distribuciones Pg y $\Pr$.

WGAN: entrenamiento del crítico

```
for i in n_critic_steps:
  opt_critic.zero_grad()
  real_images = data[0].float().to(device)

  # Generar imágenes
  noise = sample_noise()
  fake_images = netGenerator(noise)

  # Evaluarlas con el crítico
  real_log = netCritic(real_images)
  fake_log = netCritic(fake_images)

  # Calcular L = E{x~P_X}[fW(x)] - E_{Z~P_Z}[f(G(z))]
  loss = -(real_log.mean() - fake_log.mean())

  loss.backward(retain_graph=True)
  opt_critic.step()

  # Clippling del gradiente
  for p in netCritic.parameters():
      p.data.clamp_(-self.c, self.c)
```

WGAN: entrenamiento del generador

- El objetivo del generador es minimizar la distancia de Wasserstein entre Pg y $\Pr$.
- El generador intenta encontrar θ* que minimice la distancia de Wasserstein entre Pg y $\Pr$.

$$L_{gen}(w) = \min_{\theta} - \mathbb{E}_{z \sim Z}[f_w(g_\theta(z))]$$

- La principal diferencia entre el generador de una WGAN y el generador estándar es que el generador WGAN intenta minimizar la distancia de Wasserstein entre Pg y $\Pr$, y por su parte el generador de una GAN estándar intenta engañar al discriminador con las imágenes generadas.

WGAN: entrenamiento del generador

```
opt_gen.zero_grad()
noise = sample_noise()
fake_images = netGenerator(noise)

# Evaluar las imágenes generadas
fake_log = netCritic(fake_images)

# * - E_{Z~P_Z}[fW(g(z))]
loss = -fake_log.mean().view(-1)

loss.backward()
opt_gen.step()
```

Wasserstein GAN: algoritmo

Require: : α, the learning rate. c, the clipping parameter. m, the batch size. n_{critic}, the number of iterations of the critic per generator iteration.

Require: : w_0, initial critic parameters. θ_0, initial generator's parameters.

while θ has not converged **do**
  for $t = 0, ..., n_{\text{critic}}$ **do**
    Sample $\{x^{(i)}\}_{i=1}^{m} \sim \mathbb{P}_r$ a batch from the real data.
    Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
    $g_w \leftarrow \nabla_w \left[\frac{1}{m}\sum_{i=1}^{m} f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))\right]$
    $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
    $w \leftarrow \text{clip}(w, -c, c)$
  end for
  Sample $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$ a batch of prior samples.
  $g_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^{m} f_w(g_\theta(z^{(i)}))$
  $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
end while

WGAN: particularidades

- Función de activación lineal en la capa de salida del crítico

- A diferencia de un discriminador estándar, que utiliza activación sigmoide para predecir la verosimilitud de que una imagen sea real.

  model.add(Dense(1, activation='linear'))

- La función de activación lineal es la función por defecto: se puede dejar la función de activación indeterminada y se obtendrá el mismo resultado

  model.add(Dense(1))

WGAN: particularidades

- Etiquetas de clase para datos reales y generados

- Una GAN usa 0 como etiqueta de clase para datos generados y 1 para datos reales. Las etiquetas se usan para entrenar la GAN (objetivo del discriminador).
- Una WGAN no tiene etiquetas específicas para el crítico. Solo promueve que l crítico retorne diferentes scores para datos reales y generados.
- Se utilizan etiquetas positivas y negativas para las clases (-1 para datos reales y +1 para datos generados).
- Generar etiquetas de clase para datos reales:
  `y = -ones((n_samples, 1))`
- Generar etiquetas de clase para datos generados:
  `y = ones((n_samples, 1))`

WGAN: particularidades

- Función de pérdida de Wasserstein
- Función específica que calcula el score promedio para imágenes reales y generadas.
- Como SGD es un método de minimización, puede multiplicarse la etiqueta de clase por el score promedio para minimizar la pérdida de generador y crítico:

```
def wasserstein_loss(y_true, y_pred):
    return backend.mean(y_true * y_pred)
```

- En Keras, se usa la función de pérdida definida al compilar el modelo:

```
model.compile(loss=wasserstein_loss, ...)
```

- En PyTorch, se calcula la función de pérdida dentro del ciclo de entrenamiento

WGAN: particularidades

- Clip de pesos en el crítico

- En Keras: extension de la clase Constraint.
- Definir la función __call__() para aplicar la operación, get_config() para obtener una configuración y opcionalmente __init__() para definer una configuración inicial (simétrica, por ejemplo el hipercubo de lado ±0.01).

```
class ClipConstraint(Constraint):
    def __init__(self, clip_value):
        self.clip_value = clip_value
    def __call__(self, weights):
        return backend.clip(weights, -self.clip_value, self.clip_value)
    def get_config(self):
        return {'clip_value': self.clip_value}
```

WGAN: particularidades

- Clip de pesos en el crítico
- En Keras: uso de la clase Constraint definida.

  const = ClipConstraint(0.01)
- Usarla en la definición de una capa

  model.add(Conv2D(..., kernel_constraint=const))
- En PyTorch, se usa la función clamp

  p.data.clamp_(-0.01,0.01)

WGAN: particularidades

- Actualizar el crítico con más frecuencia que el generador

- En una GAN, generador y discriminador usualmente se actualizan en cada paso.
- En una WGAN se actualiza el crítico con más frecuencia, para proporcionar mejor información sobre la distancia de Wasserstein.
- Nuevo ciclo en el entrenamiento (n_critic iteraciones)

```
for _ in range(n_critic):                    # entrenar y actualizar el discriminador
    X_real, y_real = generate_real_samples(dataset, half_batch)
    c_loss1 = c_model.train_on_batch(X_real, y_real)
    X_fake, y_fake = generate_fake_samples(g_model, latent_dim, half_batch)
    c_loss2 = c_model.train_on_batch(X_fake, y_fake)
# entrenar y actualizar el generador
```

WGAN: particularidades

- Usar la versión RMSProp de SGD

- Una GAN suele usar Adam con learning rate bajo y momento modesto.
- En WGAN se recomienda usar RMSProp, con learning rate muy bajo (0.00005).
- Evitar usar momento para responder a nuevas características identificadas por el generador/discriminador, evitar inestbilidades y colapso de modo.
- En Keras:

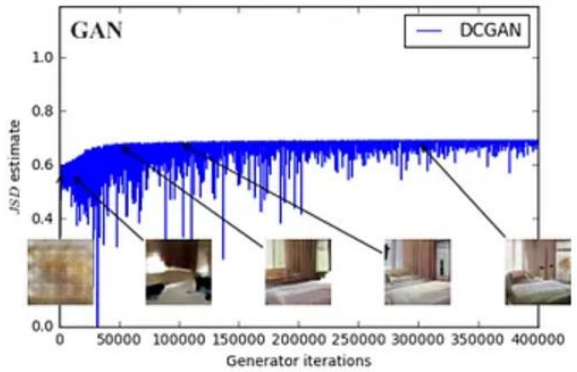
```
opt = RMSprop(lr=0.00005)
model.compile(loss=wasserstein_loss, optimizer=opt)
```

- En PyTorch:

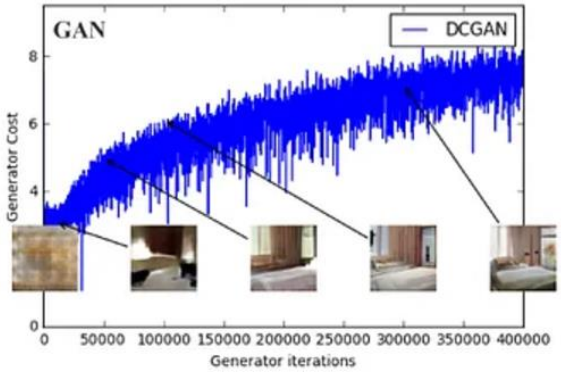
```
C_opt = torch.optim.RMSprop(C.parameters(), lr=0.0005)
G_opt = torch.optim.RMSprop(G.parameters(), lr=0.0005)
```

Wasserstein GAN: función de pérdida y calidad de datos

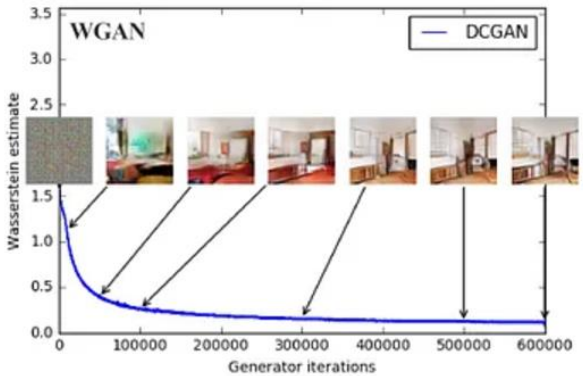
- En una GAN tradicional, la función de pérdida mide qué tan bien el generador engaña al discriminador, pero no evalúa la calidad de los datos generados.
- La pérdida del generador no disminuye aunque la calidad del dato generado mejore (no se puede inferir calidad de los valores de la función de pérdida).
- En una WGAN, la función de pérdida refleja la calidad de los datos generados.



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



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



$$\frac{1}{m}\sum_{i=1}^{m} f_w\left(x^{(i)}\right)-\frac{1}{m}\sum_{i=1}^{m} f_w\left(g_\theta\left(z^{(i)}\right)\right)$$

Wasserstein GAN: recorte de pesos

- Recortar los pesos es una forma “terrible” de imponer una restricción de Lipschitz (Arjovsky et al., 2017).
  - Si recorte es grande, los pesos pueden tardar mucho tiempo en alcanzar su límite, dificultando el entrenamiento del crítico hasta el óptimo.
  - Si el recorte es pequeño, los gradientes pueden desaparecer cuando el número de capas es grande.
- Arjovsky et al., experimentaron con variantes simples (como proyectar los pesos en una esfera) con poca diferencia y mantuvieron la estrategia de recortar los pesos debido a su simplicidad y su buen rendimiento.
- Dejaron el estudio de mejores estrategias para asegurar la condición de Lipschitz como trabajo futuro y “alentaron activamente a los investigadores interesados a mejorar el método propuesto”.

WGAN-GP: penalización de gradientes en Wasserstein GANs

- Para asegurar continuidad de Lipschitz de la función de pérdida, se limitan (recortan) los pesos a un rango [-c,c]
  - El recorte puede generar fallos de gradiente en el entrenamiento y limita la clase de funciones f_W que se pueden aprender.
- En una WGAN-GP se suma una penalización del gradiente a la función de pérdida, para asegurar que entrene apropiadamente en un espacio continuo

$$L(P_r, P_g) = \mathbb{E}_{x \in P_r}[f_w(x)] - \mathbb{E}_{z \in P_y(z)}[f_w(G_\theta(z))] + k \,.\, \mathbb{E}_{x \in P_y}[(||\nabla f_w(x)||_2 - 1)^2]$$

  - Py es la distribución obtenida al muestrear uniformemente a lo largo de líneas rectas entre puntos de las distribuciones real (Pr) y generada (Pg).

WGAN-GP: penalización de gradientes en Wasserstein GANs

- WGAN-GP converge más rápido y puede generar muestras de mayor calidad que al aplicar la estrategia de poda de pesos.
- Para WGAN-GP se suele evitar el uso de normalización [de batches} para el crítico, porque crea correlaciones entre las muestras y reduce la eficacia de la penalización del gradiente.

Wasserstein GAN-GP: algoritmo

Require: The gradient penalty coefficient λ, the number of critic iterations per generator iteration n_{critic}, the batch size m, Adam hyperparameters α, β_1, β_2.

Require: initial critic parameters w_0, initial generator parameters θ_0.

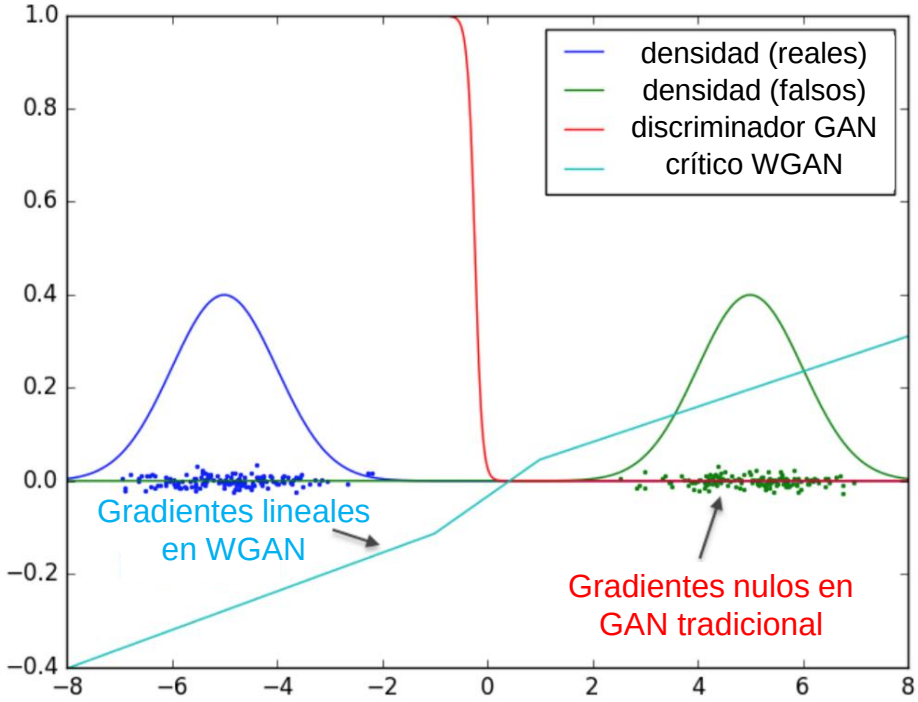
while θ has not converged **do**
  for $t = 1, ..., n_{\text{critic}}$ **do**
    for $i = 1, ..., m$ **do**
      Sample real data $\boldsymbol{x} \sim \mathbb{P}_r$, latent variable $\boldsymbol{z} \sim p(\boldsymbol{z})$, a random number $\epsilon \sim U[0, 1]$.
      $\tilde{\boldsymbol{x}} \leftarrow G_\theta(\boldsymbol{z})$
      $\hat{\boldsymbol{x}} \leftarrow \epsilon \boldsymbol{x} + (1 - \epsilon)\tilde{\boldsymbol{x}}$
      $L^{(i)} \leftarrow D_w(\tilde{\boldsymbol{x}}) - D_w(\boldsymbol{x}) + \lambda(\|\nabla_{\hat{\boldsymbol{x}}} D_w(\hat{\boldsymbol{x}})\|_2 - 1)^2$
    end for
    $w \leftarrow \text{Adam}(\nabla_w \frac{1}{m} \sum_{i=1}^{m} L^{(i)}, w, \alpha, \beta_1, \beta_2)$
  end for
  Sample a batch of latent variables $\{\boldsymbol{z}^{(i)}\}_{i=1}^{m} \sim p(\boldsymbol{z})$.
  $\theta \leftarrow \text{Adam}(\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} -D_w(G_\theta(\boldsymbol{z})), \theta, \alpha, \beta_1, \beta_2)$
end while

Wasserstein GANs: ventajas

- El crítico es capaz de aportar información valiosa para el entrenamiento del generador, aún cuando esté muy entrenado (idealmente, hasta el óptimo).
  - Es una ventaja fundamental sobre las GAN tradicionales, en las cuales el discriminador, una vez optimizado, no es capaz de proporcionar información útil (del gradiente) para el entrenamiento del generador.
- El crítico no se satura: converge a una función lineal con gradientes definidos en todos los puntos.
- El entrenamiento es más estable, menos sensible a la configuración de la arquitectura y de los hiperparámetros.
- El entrenamiento no requiere mantener el balance entre discriminador y generador.

Wasserstein GANs: ventajas

- Utilizar la distancia de Wasserstein permite obtener mejores resultados, incluso si las distribuciones $\Pr$ y P_g son muy diferentes.
- El proceso de aprendizaje es más estable, evita el colapso de modo y mejora el aprendizaje de clases.
- Una WGAN puede seguir aprendiendo aún cuando el crítico no lo haga.



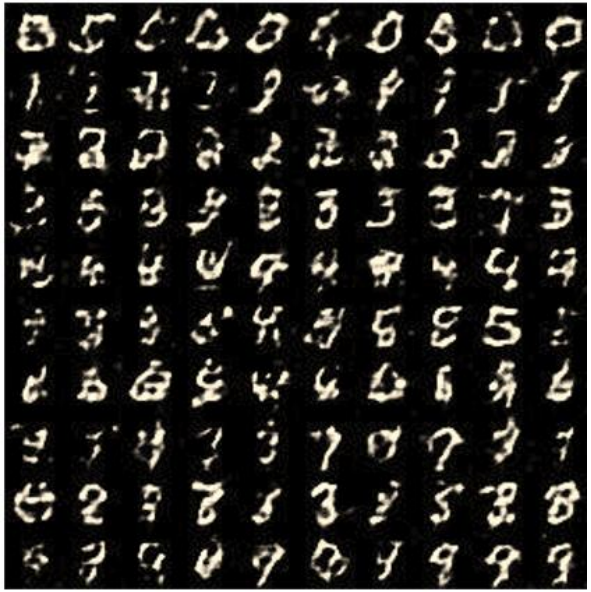
Wasserstein GANs: implementación

Cambios menores sobre una implementación de GAN tradicional.

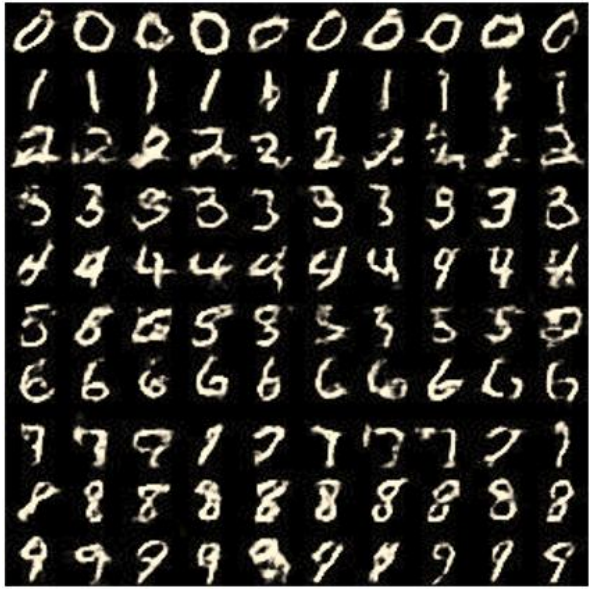
1. Se debe usar la distancia de Wasserstein como función de pérdida para entrenar el crítico y el generador.
2. Se debe usar una función de activación lineal en la capa de salida del crítico (en lugar de sigmoide en el discriminador).
3. Se debe aplicar algún criterio para asegurar que f es Liptchitziana:
    - restringir los pesos luego de actualizar cada batch (clamp), e.g., al rango [-0.01,0.01].
    - Penalización de gradientes, en la función de pérdida
4. Actualizar el crítico más veces que el generador (e.g., 5 veces).
5. Usar la variante RMSProp de SGD con learning rate baja y sin momento.

Wasserstein GAN: ejemplo MNIST

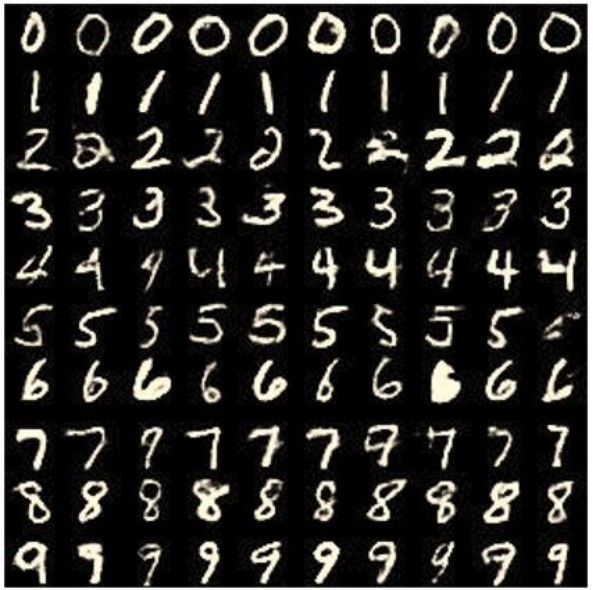
6 épocas de entrenamiento



20 épocas de entrenamiento

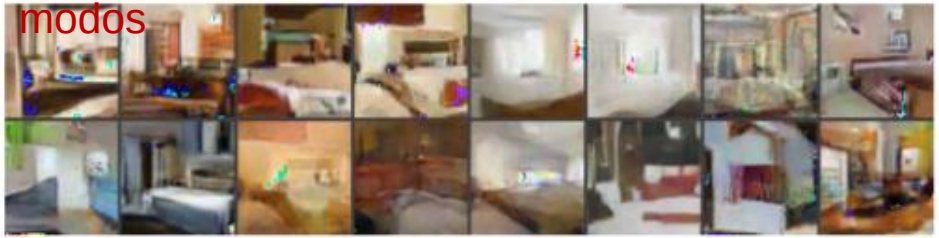


40 épocas de entrenamiento



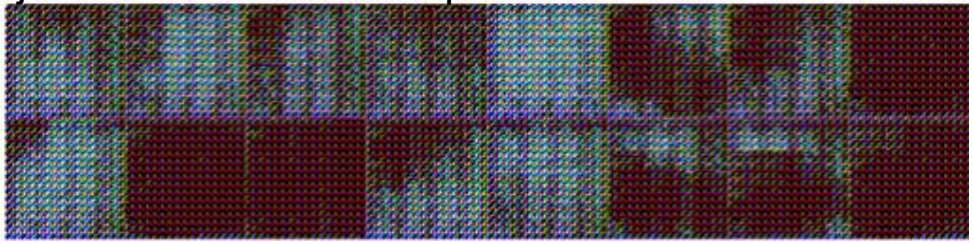
Wasserstein GAN: ejemplos vs. GAN

vs. MLP de cuatro capas: WGAN mejor diversidad, GAN colapsa a pocos modos



vs. DCGAN, cuatro capas: WGAN imágenes de mejor calidad

vs. GAN sin normalización de batches y número fijo de filtros en cada capa: GAN falla



Wasserstein GAN: códigos de ejemplo

Disponibles en https://github.com/nesmachnow/Curso-GANs (Clase 5)

- Wasserstein GAN
  colab.research.google.com/drive/1ilRzKzb5WbzHE_Da2z1Nzd-bAPt9d9R2
- Wasserstein GAN con penalización de gradientes
  colab.research.google.com/drive/1Lrk6urqFjX3YEnLkCwipx7_9gz7QBABo
- Wasserstein GAN para modelar una serie temporal financiera
  https://github.com/CasperHogenboom/WGAN_financial_time-series