

# Xarxes generatives antagonistes

Iloc: [Institut d'Ensenyaments a Distància de les Illes Balears](#)  
Curs: Sistemes d'aprenentatge automàtic  
Llibre: Xarxes generatives antagonistes

Imprès per: Carlos Sanchez Recio  
Data: dimarts, 25 de març 2025, 07:24

# Taula de continguts

## 1. Introducció

## 2. Models generatius

- 2.1. Modelització de probabilitats
- 2.2. Dificultat dels models generatius

## 3. Estructura de les GAN

## 4. Discriminador

- 4.1. Entrenament

## 5. Generador

- 5.1. Entrada aleatòria

## 6. Entrenament de les GAN

- 6.1. Entrenament alternatiu
- 6.2. Convergència

## 7. Funcions de pèrdua

- 7.1. Pèrdua minimax
- 7.2. Pèrdua de Wasserstein

## 8. Problemes a la pràctica

- 8.1. Gradients evanescents
- 8.2. Collapse de mode
- 8.3. Convergència fallida

## 9. Variants de GAN

## 10. DCGAN

- 10.1. Llibreries i dades
- 10.2. Generador
- 10.3. Discriminador
- 10.4. Funcions de pèrdues
- 10.5. Entrenament
- 10.6. Resultats

## 11. Conclusió

## 1. Introducció

Les xarxes generatives antagonistes (*Generative Adversarial Networks*, **GAN**) són una innovació recent interessant en l'aprenentatge automàtic. Els GAN són models generatius: creen noves instàncies de dades que s'assemblen a les vostres dades d'entrenament. Per exemple, els GAN poden crear imatges que semblen fotografies de cares humans, encara que les cares no pertanyin a cap persona real. Aquestes imatges van ser creades per una GAN:



Les GAN aconsegueixen aquest nivell de realisme emparellant un sistema generador, que aprèn a produir la sortida objectiu, amb un sistema discriminador, que aprèn a distingir les dades reals de la sortida del generador. El generador intenta enganyar el discriminador, i el discriminador intenta evitar ser enganyat.

## 2. Models generatius

Què significa "generativa" en el nom de "Xarxa adversària generativa"? "Generatiu" descriu una classe de models estadístics que contrasta amb els models discriminatius.

Informalment:

- Els models generatius poden generar noves instàncies de dades.
- Els models discriminatius discriminen entre diferents tipus d'instàncies de dades.

Un model generatiu podria generar noves fotos d'animals que semblen animals reals, mentre que un model discriminatiu podria distingir un ca d'un moix. Els GAN són només un tipus de model generatiu.

Més formalment, donat un conjunt d'instàncies de dades  $X$  i un conjunt d'etiquetes  $Y$ :

Els models generatius capturen la probabilitat conjunta  $p(X, Y)$ , o només  $p(X)$  si no hi ha etiquetes

Els models discriminatius capturen la probabilitat condicional  $p(Y|X)$ .

Un model generatiu inclou la distribució de les dades en si i ens indica la probabilitat d'un exemple determinat. Per exemple, els models que prediuen la paraula següent en una seqüència solen ser models generatius (normalment molt més simples que les GAN) perquè poden assignar una probabilitat a una seqüència de paraules.

Un model discriminatiu ignora la qüestió de si una instància determinada és probable i només ens indica la probabilitat que s'apliqui una etiqueta a la instància.

Hem de tenir en compte que aquesta és una definició molt general. Hi ha molts de tipus de models generatius, les GAN en són un.

## 2.1. Modelització de probabilitats

Cap d'aquests dos tipus de model no ha de retornar un nombre que representi una probabilitat. Podem modelitzar la distribució de dades imitant aquesta distribució.

Per exemple, un classificador discriminatiu com un arbre de decisió pot etiquetar una instància sense assignar una probabilitat a aquesta etiqueta. Aquest classificador encara seria un model perquè la distribució de totes les etiquetes previstes modelitzaria la distribució real de les etiquetes a les dades.

De la mateixa manera, un model generatiu pot modelitzar una distribució produint dades "falses" però convinents que semblen extrems d'aquesta distribució.

## 2.2. Dificultat dels models generatius

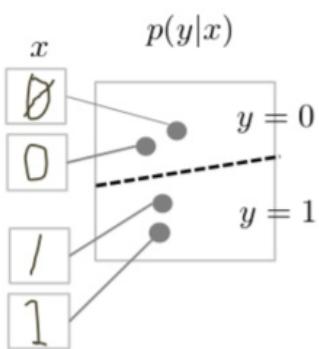
Els models generatius encaren una tasca més difícil que els models discriminatius anàlegs. Els models generatius han de modelitzar més.

Un model generatiu d'imatges podria capturar correlacions com "les coses que semblen vaixells probablement apareixeran a prop de coses que semblen aigua" i "és poc probable que els ulls apareguin al front". Són distribucions molt complicades.

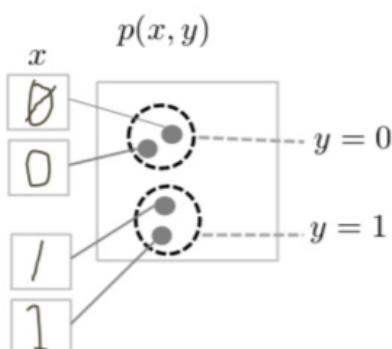
En canvi, un model discriminatiu podria aprendre la diferència entre "veler" o "no veler" només cercant alguns patrons reveladors. Podria ignorar moltes de les correlacions que el model generatiu ha d'encertar.

Els models discriminatius intenten traçar límits a l'espai de les dades, mentre que els models generatius intenten modelitzar com es colloquen les dades a tot l'espai. Per exemple, el diagrama següent mostra models discriminatius i generatius de díigits escrits a mà:

- Discriminative Model



- Generative Model



El model discriminatiu intenta distingir entre els 0 i els 1 escrits a mà dibuixant una línia a l'espai de dades. Si encerta la línia, pot distingir els 0 dels 1 sense haver de modelitzar exactament on es colloquen les instàncies a l'espai de dades a banda i banda de la línia.

En canvi, el model generatiu intenta produir 1 i 0 convincents generant díigits que es troben a prop dels seus homòlegs reals a l'espai de dades. Ha de modelitzar la distribució per tot l'espai de dades.

Els GAN ofereixen una manera eficaç d'entrenar models tan rics que s'assemblen a una distribució real. Per entendre com funcionen, haurem d'entendre l'estructura bàsica d'una GAN.

### 3. Estructura de les GAN

Una xarxa antagonista generativa (GAN) té dues parts:

El generador aprèn a generar dades plausibles. Les instàncies generades esdevenen exemples d'entrenament negatius per al discriminador.

El discriminador aprèn a distingir les dades falses del generador de les dades reals. El discriminador penalitza el generador per produir resultats poc plausibles.

Quan comença l'entrenament, el generador produeix dades òbviament falses i el discriminador aprèn ràpidament a dir que són falses:



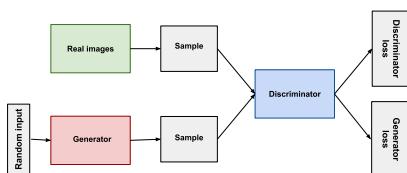
A mesura que avança l'entrenament, el generador s'acosta a produir una sortida que pot enganyar el discriminador:



Finalment, si l'entrenament del generador va bé, el discriminador empitjora a l'hora de dir la diferència entre real i fals. Comença a classificar les dades falses com a reals i la seva precisió disminueix.



Aquí tenim una imatge de tot el sistema:

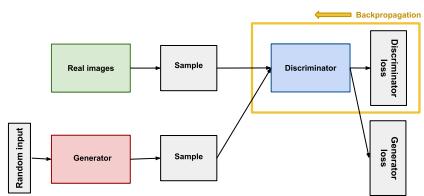


Tant el generador com el discriminador són xarxes neuronals. La sortida del generador està connectada directament a l'entrada del discriminador. Mitjançant la retropropagació, la [classificació](#) del discriminador proporciona un senyal que el generador utilitza per actualitzar els seus pesos.

Expliquem les peces d'aquest sistema amb més detall.

## 4. Discriminador

El discriminador en un GAN és simplement un classificador. Intenta distingir les dades reals de les dades creades pel generador. Podria utilitzar qualsevol arquitectura de xarxa adequada al tipus de dades que està classificant.



Les dades d'entrenament del discriminador provenen de dues fonts:

Instàncies de dades reals, com ara imatges reals de persones. El discriminador utilitza aquests casos com a exemples positius durant l'entrenament.

Instàncies de dades falses creades pel generador. El discriminador utilitza aquests casos com a exemples negatius durant l'entrenament.

A la figura, els dos blocs "Sample" representen aquestes dues fonts de dades que alimenten el discriminador. Durant l'entrenament del discriminador, el generador no s'entrena. Els seus pesos es mantenen constants mentre produeix exemples perquè el discriminador pugui entrenar.

## 4.1. Entrenament

El discriminador connecta dues funcions de pèrdua. Durant l'entrenament del discriminador, el discriminador ignora la pèrdua del generador i només utilitza la pèrdua del discriminador. Utilitzem la pèrdua del generador durant l'entrenament del generador, tal com es descriu a la secció següent.

Durant l'entrenament del discriminador:

El discriminador classifica tant les dades reals com les falses del generador.

La pèrdua del discriminador penalitza el discriminador per classificar erròniament una instància real com a falsa o una instància falsa com a real.

El discriminador actualitza els seus pesos mitjançant la retropropagació de la pèrdua del discriminador a través de la xarxa del discriminador.

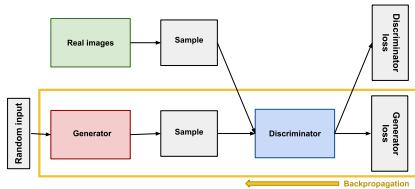
A la següent secció veurem per què la pèrdua del generador es connecta amb el discriminador.

## 5. Generador

La part generadora d'un GAN aprèn a crear dades falses incorporant la retroalimentació del discriminador. Aprèn a fer que el discriminador classifiqui la seva sortida com a real.

L'entrenament del generador requereix una integració més estreta entre el generador i el discriminador de la que requereix la formació del discriminador. La part del GAN que entrena el generador inclou:

- entrada aleatòria
- xarxa generadora, que transforma l'entrada aleatòria en una instància de dades
- xarxa discriminadora, que classifica les dades generades
- sortida del discriminador
- pèrdua del generador, que penalitza el generador per no aconseguir enganyar el discriminador



## 5.1. Entrada aleatòria

Les xarxes neuronals necessiten algun tipus d'entrada. Normalment introduïm dades amb les quals volem fer alguna cosa, com ara una instància sobre la qual volem classificar o fer una predicció. Però, què feim servir com a entrada per a una xarxa que produeix instàncies de dades completament noves?

En la seva forma més bàsica, una GAN pren soroll aleatori com a entrada. Aleshores, el generador transforma aquest soroll en una sortida significativa. Mitjançant la introducció de soroll, podem aconseguir que el GAN produueixi una gran varietat de dades, mostrejant des de diferents llocs de la distribució objectiu.

Els experiments suggereixen que la distribució del soroll no importa gaire, de manera que podem triar alguna cosa que sigui fàcil de mostrejar, com ara una distribució uniforme. Per comoditat, l'espai des del qual es mostreja el soroll sol ser de dimensió més petita que la dimensionalitat de l'espai de sortida.

## 6. Entrenament de les GAN

Com que una GAN conté dues xarxes entrenades per separat, el seu algorisme d'entrenament ha de fer front a dues complicacions:

- Els GAN han de fer malabars amb dos tipus diferents d'entrenament (generador i discriminador).
- La convergència a les GAN és difícil d'identificar.

## 6.1. Entrenament alternatiu

El generador i el discriminador tenen diferents processos d'entrenament. Llavors, com entrenam la GAN en conjunt?

L'entrenament de la GAN es desenvolupa en períodes alternatius:

1. El discriminador s'entrena per a una o més èpoques.
2. El generador s'entrena durant una o més èpoques.
3. Es repeteixen els passos 1 i 2 per continuar entrenant les xarxes generadora i discriminadora

Durant la fase d'entrenament del discriminador el generador es manté constant. A mesura que l'entrenament del discriminador intenta esbrinar com distingir les dades reals de les falses, ha d'aprendre a reconèixer els defectes del generador. Aquest és un problema diferent per a un generador ben entrenat que per a un generador no entrenat que produeix una sortida aleatòria.

De la mateixa manera, mantenim el discriminador constant durant la fase d'entrenament del generador. En cas contrari, el generador intentaria colpejar un objectiu en moviment i no podria convergir-hi mai.

És això d'anar i tornar el que permet a les GAN resoldre problemes generatius insolubles d'una altra manera. Començam a resoldre el problema generatiu amb un de molt més senzill, un problema de [classificació](#). Per contra, si no podem entrenar un classificador per saber la diferència entre les dades reals i les generades, fins i tot per a la sortida inicial del generador aleatori, no podem ni tan sols iniciar l'entrenament de la GAN.

## 6.2. Convergència

A mesura que el generador millora amb l'entrenament, el rendiment del discriminador empitjora perquè el discriminador no pot distingir fàcilment entre real i fals. Si el generador funciona perfectament, aleshores el discriminador té una precisió del 50%. De fet, el discriminador tira una moneda per fer la seva predicció.

Aquesta progressió planteja un problema per a la convergència de la GAN en el seu conjunt: la retroalimentació del discriminador es fa més poc significativa amb el temps. Si la GAN continua entrenant més enllà del punt en què el discriminador està donant una retroalimentació completament aleatòria, aleshores el generador començà a entrenar amb la retroalimentació no desitjada i la seva pròpia qualitat pot col·lapsar-se.

En el cas d'una GAN, la convergència és sovint un estat fugaç, més que no estable.

## 7. Funcions de pèrdua

Les GAN intenten replicar una distribució de probabilitat. Per tant, haurien d'utilitzar funcions de pèrdua que reflecteixin la distància entre la distribució de les dades generades per la GAN i la distribució de les dades reals.

Com es capture la diferència entre dues distribucions a les funcions de pèrdua GAN? Aquesta qüestió és una àrea de recerca activa i s'hi han proposat molts d'enfocaments. Abordarem dues funcions comunes de pèrdua de GAN aquí, totes dues implementades a TF-GAN:

minimax loss: la funció de pèrdua utilitzada en el document que va introduir les GAN.

- Pèrdua de Wasserstein: la funció de pèrdua predeterminada per als estimadors TF-GAN. Es descriu per primera vegada en un article del 2017.
- TF-GAN també implementa moltes altres funcions de pèrdua.

### Una funció de pèrdua o dues?

Una GAN pot tenir dues funcions de pèrdua: una per a l'entrenament del generador i una altra per a l'entrenament del discriminador. Com poden treballar plegades dues funcions de pèrdua per reflectir una mesura de distància entre distribucions de probabilitat?

En els esquemes de pèrdues que veurem aquí, les pèrdudes del generador i del discriminador es deriven d'una única mesura de distància entre distribucions de probabilitat. En tots dos esquemes, però, el generador només pot afectar un terme en la mesura de distància: el terme que reflecteix la distribució de les dades falses. Així, durant l'entrenament del generador, deixam anar l'altre terme, que reflecteix la distribució de les dades reals.

Les pèrdudes del generador i del discriminador semblen diferents al final, tot i que deriven d'una fórmula única.

## 7.1. Pèrdua minimax

A l'article que va introduir les GAN, el generador intenta minimitzar la funció següent mentre que el discriminador intenta maximitzar-la:

$$E_x[\log(D(x))] + E_z[\log(1 - D(G(z)))]$$

En aquesta funció:

$D(x)$  és l'estimació del discriminador de la probabilitat que la instància real de dades  $x$  sigui real.

$E_x$  és el valor esperat de totes les instàncies de dades reals.

$G(z)$  és la sortida del generador quan es dóna el soroll  $z$ .

$D(G(z))$  és l'estimació del discriminador de la probabilitat que una instància falsa sigui real.

$E_z$  és el valor esperat de totes les entrades aleatòries al generador (en efecte, el valor esperat de totes les instàncies falses generades  $G(z)$ ).

La fórmula deriva de l'entropia creuada entre la distribució real i la generada.

El generador no pot afectar directament el terme  $\log(D(x))$  de la funció, de manera que, per al generador, minimitzar la pèrdua equival a minimitzar  $\log(1 - D(G(z)))$ .

A TF-GAN, vegeu `minimax_discriminator_loss` i `minimax_generator_loss` per obtenir una implementació d'aquesta funció de pèrdua.

### Pèrdua Minimax modificada

El document GAN original assenyala que la funció de pèrdua minimax anterior pot fer que el GAN quedí encallat en les primeres etapes de la formació GAN quan la feina del discriminador és molt fàcil. Per tant, el document suggerix modificar la pèrdua del generador de manera que el generador intenti maximitzar el log  $D(G(z))$ .

A TF-GAN, vegeu `modified_generator_loss` per obtenir una implementació d'aquesta modificació.

## 7.2. Pèrdua de Wasserstein

Per defecte, TF-GAN utilitza la pèrdua de Wasserstein.

Aquesta funció de pèrdua depèn d'una modificació de l'esquema GAN (anomenada "Wasserstein GAN" o "WGAN") en què el discriminador no classifica realment les instàncies. Per a cada instància emet un número. Aquest nombre no ha de ser necessàriament inferior a 1 ni superior a 0, de manera que no podem utilitzar 0.5 com a llindar per decidir si una instància és real o falsa. L'entrenament del discriminador només intenta fer que la sortida sigui més gran per a instàncies reals que per a instàncies falses.

Com que realment no pot discriminar entre real i fals, el discriminador WGAN s'anomena realment "crític" en lloc de "discriminador". Aquesta distinció té importància teòrica, però per a finalitats pràctiques podem tractar-la com un reconeixement que les entrades a les funcions de pèrdua no han de ser probabilitats.

Les funcions de pèrdua en si són enganyosament senzilles:

Pèrdua crítica:  $D(x) - D(G(z))$

El discriminador intenta maximitzar aquesta funció. En altres paraules, intenta maximitzar la diferència entre la seva sortida en instàncies reals i la seva sortida en instàncies falses.

Pèrdua del generador:  $D(G(z))$

El generador intenta maximitzar aquesta funció. En altres paraules, intenta maximitzar la sortida del discriminador per a les seves instàncies falses.

En aquestes funcions:

$D(x)$  és la sortida de la crítica per a una instància real.

$G(z)$  és la sortida del generador quan es dóna el soroll  $z$ .

$D(G(z))$  és la sortida del crític per a una instància falsa.

La sortida de la crítica  $D$  no ha d'estar entre 1 i 0.

Les fòrmules es deriven de la distància del moviment de terra ([Earth Mover's Distance](#), EMD) entre la distribució real i la generada.

A TF-GAN, vegeu `wasserstein_generator_loss` i `wasserstein_discriminator_loss` per a les implementacions.

### Requeriments

La justificació teòrica del Wasserstein GAN (o WGAN) requereix que els pesos de tot el GAN es retallin de manera que es mantinguin dins d'un rang restringit.

### Avantatges

Les GAN de Wasserstein són més poc vulnerables a quedar-se encallades que els GAN basats en minimax i eviten problemes amb els gradients de desaparició. L'EMD també té l'avantatge de ser una veritable mètrica: una mesura de distància en un espai de distribucions de probabilitat. L'entropia creuada no és una mètrica en aquest sentit.

## 8. Problemes a la pràctica

Els GAN tenen diversos modes de fallada comuns. Tots aquests problemes comuns són àrees de recerca activa. Tot i que no s'ha resolt completament cap d'aquests problemes, esmentarem algunes de les solucions que s'hi han assajat.

## 8.1. Gradients evanescents

La recerca ha suggerit que si el discriminador és massa bo, l'entrenament del generador pot fallar a causa dels gradients que desapareixen. En efecte, un discriminador òptim no proporciona prou informació perquè el generador avanci.

### Intents de remei

- Pèrdua de Wasserstein: la pèrdua de Wasserstein està dissenyada per evitar que els gradients desapareguin fins i tot quan s'entrena el discriminador per ser òptim.
- Pèrdua minimax modificada: el document GAN original proposava una modificació de la pèrdua minimax per fer front als gradients que s'esvaeixen.

## 8.2. Collapse de mode

Normalment volem que la GAN produueixi una gran varietat de sortides. Volem, per exemple, una cara diferent per a cada entrada aleatòria a un generador de cares.

Tanmateix, si un generador produueix una sortida especialment plausible, el generador pot aprendre a produir només aquesta sortida. De fet, el generador sempre intenta trobar la sortida que sembla més plausible per al discriminador.

Si el generador comença a produir la mateixa sortida (o un petit conjunt de sortides) una i altra vegada, la millor estratègia del discriminador és aprendre a rebutjar sempre aquesta sortida. Però si la següent generació de discriminadors s'encalla en un mínim local i no troba la millor estratègia, llavors és massa fàcil que la següent iteració del generador trobi la sortida més plausible per al discriminador actual.

Cada iteració del generador s'optimitza en excés per a un discriminador particular, i el discriminador mai aconsegueix aprendre la manera de sortir de la trampa. Com a resultat, els generadors giren a través d'un petit conjunt de tipus de sortida. Aquesta forma de fallada a les GAN s'anomena **collapse de mode** (*mode collapse*).

### Intents de remei

Els enfocaments següents intenten forçar el generador a ampliar el seu abast evitant que s'optimitzi per a un únic discriminador fix:

1. Pèrdua de Wasserstein: la pèrdua de Wasserstein alleuja el collapse del mode permetent-vos entrenar el discriminador a l'òptim sense preocupar-vos dels gradients que desapareixen. Si el discriminador no s'encalla en els mínims locals, aprèn a rebutjar les sortides sobre les quals s'estabilitza el generador. Així que el generador ha de provar alguna cosa nova.
2. GAN desenrotllats: els GAN desenrotllats utilitzen una funció de pèrdua de generador que incorpora no només les classificacions actuals del discriminador, sinó també les sortides de futures versions del discriminador. Per tant, el generador no pot optimitzar massa per a un únic discriminador.

### 8.3. Convergència fallida

Les GAN sovint no aconsegueixen convergir.

#### Intents de remei

Els investigadors han intentat utilitzar diverses formes de regularització per millorar la convergència GAN, com ara:

- Afegir soroll a les entrades del discriminador: vegeu, per exemple, Cap a mètodes bàsics per a l'entrenament de xarxes adversàries generatives.
- Penalització dels pesos del discriminador: Vegeu, per exemple, Estabilització de la formació de xarxes generatives adversàries mitjançant la regularització.

## 9. Variants de GAN

Els investigadors continuen trobant tècniques GAN millorades i nous usos per a les GAN. Aquí tenim una mostra de les variacions de GAN per donar una idea de les possibilitats d'aquesta tècnica.

### GANs progressives

En una GAN progressiva, les primeres capes del generador produueixen imatges de molt baixa resolució, i les capes posteriors afegeixen detalls. Aquesta tècnica permet a la GAN entrenar més ràpidament que les GAN no progressives comparables i produceix imatges de resolució més alta.

Per obtenir més informació, vegeu [Karras et al, 2017](#).

### GANs condicionals

Els GAN condicionals entrenen en un conjunt de dades etiquetat i us permeten especificar l'etiqueta per a cada instància generada. Per exemple, un MNIST GAN incondicional produiria dígits aleatoris, mentre que un MNIST GAN condicional us permetria especificar quin dígit hauria de generar el GAN.

En lloc de modelitzar la probabilitat conjunta  $P(X, Y)$ , les GAN condicionals modelen la probabilitat condicional  $P(X|Y)$ .

Per obtenir més informació sobre les GAN condicionals, vegeu [Mirza et al, 2014](#).

### Traducció d'imatge a imatge

Els GAN de traducció d'imatge a imatge prenen una imatge com a entrada i l'assignen a una imatge de sortida generada amb diferents propietats. Per exemple, podem prendre una imatge de màscara amb una taca de color en forma de cotxe, i el GAN pot omplir la forma amb detalls fotorealistes del cotxe.

De la mateixa manera, podem entrenar un GAN d'imatge a imatge per prendre esbossos de bosses de mà i convertir-los en imatges fotorealistes.



En aquests casos, la pèrdua és una combinació ponderada de la pèrdua habitual basada en el discriminador i una pèrdua píxel a píxel que penalitza el generador per apartar-se de la imatge d'origen.

Per a més informació, vegeu [Isola et al, 2016](#).

### CycleGAN

Les CycleGAN aprenen a transformar imatges d'un conjunt en imatges que podrien pertànyer de manera plausible a un altre conjunt. Per exemple, un CycleGAN va produir la imatge de la dreta a continuació quan es va donar la imatge de l'esquerra com a entrada. Va agafar una imatge d'un cavall i la va convertir en una imatge d'una zebra.



Les dades d'entrenament per a la CycleGAN són simplement dos conjunts d'imatges (en aquest cas, un conjunt d'imatges de cavalls i un conjunt d'imatges de zebres). El sistema no requereix etiquetes ni correspondències per parelles entre imatges.

Per obtenir més informació, vegeu [Zhu et al., 2017](#), que il·lustra l'ús de CycleGAN per realitzar traduccions d'imatge a imatge sense dades aparellades.

## Síntesi de text a imatge

Les GAN de text a imatge prenen text com a entrada i produeixen imatges que són plausibles i descrites pel text. Per exemple, la imatge d'una flor de baix es va produir introduint una descripció de text a una GAN.



**Imatge:** Resposta de sortida d'una GAN al text d'entrada "this flower has petals that are yellow with shades of orange"

En aquest sistema la GAN només pot produir imatges dins un conjunt petit de classes.

Per a més informació, vegeu [Zhang et al., 2016](#).

## Superresolució

La superresolució incrementa la resolució de les imatges, afegint detall on cal per farcir les àrees borroses. Per exemple, la imatge de baix al mig és una versió submostrejada de la imatge original a l'esquerra. Donada la imatge borrosa, una GAN produí la imatge més detallada de la dreta.



La imatge generada per la GAN s'assembla molt a la imatge original, però si miram bé la diadema, veurem que la GAN no reproduceix el patró d'estrella de l'original. En canvi, va crear el seu propi patró plausible per substituir el patró esborrat pel mostratge inferior.

Per a més informació, vegeu [Ledig et al., 2017](#).

## Pintat de cares

Les GAN s'han utilitzat per a la tasca de pintat semàntic d'imatges. A la tasca d'*inpainting*, els trossos d'una imatge s'enfosqueixen i el sistema intenta omplir els trossos que falten.

Yeh et al, 2017 van utilitzar una GAN per superar altres tècniques per pintar imatges de cares:



### Conversió de text a veu

No totes les GAN produeixen imatges. Per exemple, també s'han fet servir GANs per produir veu sintètica a partir d'una entrada de text. Per a més informació, vegeu [Yang et al, 2017](#).

## 10. DCGAN

Al seu llibre **Python Deep Learning**, Jordi Torres ofereix el següent exemple de DCGAN.

<https://colab.research.google.com/drive/16Re1qUzPjRFE7uSzOqgoRER4tFXIK6lF?usp=sharing>

Vegem els blocs que conté als apartats següents.

- Llibreries i dades
- Generador
- Discriminador
- Funcions de pèrdues
- Entrenament
- Resultats

## 10.1. Llibreries i dades

```
import tensorflow as tf  
  
from tensorflow import keras  
print(tf.__version__)
```

```
!nvidia-smi
```

```
import imageio  
import matplotlib.pyplot as plt  
import numpy as np  
import os  
import time
```

```
(train_images, _) = tf.keras.datasets.mnist.load_data()  
  
train_images = train_images.reshape(train_images.shape[0], 28, 28, 1).astype('float32')
```

```
train_images = (train_images - 127.5)/127.5
```

```
BUFFER_SIZE = 60000  
BATCH_SIZE = 256  
  
train_dataset =  
tf.data.Dataset.from_tensor_slices(train_images).shuffle(BUFFER_SIZE).batch(BATCH_SIZE)
```

## 10.2. Generador

```
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense, Reshape, Conv2DTranspose, BatchNormalization,
LeakyReLU, Conv2D, Flatten

def make_generator_model():
    model = Sequential()
    model.add(Dense(7*7*256, use_bias=False, input_shape=(100,)))
    model.add(Reshape((7,7,256)))

    model.add(Conv2DTranspose(128,(5,5),strides=(2,2), padding='same'))
    model.add(BatchNormalization())
    model.add(LeakyReLU(alpha=0.01))

    model.add(Conv2DTranspose(64, (5,5), strides=(1,1), padding='same'))
    model.add(BatchNormalization())
    model.add(LeakyReLU(alpha=0.01))

    model.add(Conv2DTranspose(1, (5,5), strides=(2,2), padding='same', activation='tanh'))

    return model
```

```
generator = make_generator_model()

generator.summary()
```

```
noise_dim = 100

noise = tf.random.normal([1, noise_dim])

generated_image = generator(noise, training=False)

plt.imshow(generated_image[0, :, :, 0], cmap='gray')
```

## 10.3. Discriminador

```
def make_discriminator_model():
    model = Sequential()
    model.add(Conv2D(32, (5,5), strides = (2,2), padding='same', input_shape=[28,28,1]))
    model.add(LeakyReLU(alpha=0.01))

    model.add(Conv2D(64, (5,5), strides=(2,2), padding='same'))
    model.add(BatchNormalization())
    model.add(LeakyReLU(alpha=0.01))

    model.add(Conv2D(128, (5,5), strides=(2,2), padding='same'))
    model.add(BatchNormalization())
    model.add(LeakyReLU(alpha=0.01))

    model.add(Flatten())
    model.add(Dense(1,activation='sigmoid'))

    return model
```

```
discriminator = make_discriminator_model()

discriminator.summary()
```

## 10.4. Funcions de pèrdues

```
cross_entropy = tf.keras.losses.BinaryCrossentropy(from_logits=True)
```

```
def discriminator_loss(real_output, fake_output):
    real_loss = cross_entropy(tf.ones_like(real_output), real_output)
    fake_loss = cross_entropy(tf.zeros_like(fake_output), fake_output)
    total_loss = real_loss + fake_loss
    return total_loss
```

```
def generator_loss(fake_output):
    return cross_entropy(tf.ones_like(fake_output), fake_output)
```

```
generator_optimizer = tf.keras.optimizers.Adam(1e-4)
```

```
discriminator_optimizer = tf.keras.optimizers.Adam(1e-4)
```

## 10.5. Entrenament

```
@tf.function
def train_step(images):
    noise = tf.random.normal([BATCH_SIZE, noise_dim])

    with tf.GradientTape(persistent=True) as gen_tape, tf.GradientTape(persistent=True) as disc_tape:

        generated_images = generator(noise, training=True)

        real_output = discriminator(images, training=True)
        fake_output = discriminator(generated_images, training=True)

        gen_loss = generator_loss(fake_output)
        disc_loss = discriminator_loss(real_output, fake_output)

        gradients_of_generator = gen_tape.gradient(gen_loss, generator.trainable_variables)
        gradients_of_discriminator = gen_tape.gradient(disc_loss, discriminator.trainable_variables)

        generator_optimizer.apply_gradients(zip(gradients_of_generator,
                                                generator.trainable_variables))

        discriminator_optimizer.apply_gradients(zip(gradients_of_discriminator,
                                                    discriminator.trainable_variables))
```

## 10.6. Resultats

```
def generate_images(model, test_input):  
  
    predictions = model(test_input, training = False)  
  
    fig = plt.figure(figsize=(grid_size_x, grid_size_y))  
    for i in range(predictions.shape[0]):  
        plt.subplot(grid_size_x, grid_size_y, i+1)  
        plt.imshow(predictions[i, :, :, 0]*127.5+127.5, cmap='gray')  
        plt.axis('off')  
    plt.show()
```

```
grid_size_x = 10  
grid_size_y = 10  
seed = tf.random.normal([grid_size_x*grid_size_y, noise_dim])  
  
def train(dataset, epochs):  
    for epoch in range(epochs):  
        start = time.time()  
        for image_batch in dataset:  
            train_step(image_batch)  
        generate_images(generator, seed)  
        print('Time for epoch {} is {} sec'.format(epoch + 1, time.time()-start))  
    generate_images(generator, seed)
```

```
EPOCHS = 100  
train(train_dataset, EPOCHS)
```

## 11. Conclusió

En aquest lliurament hem vist:

- La diferència entre els models generatius i discriminatius
- Problemes que les GAN poden resoldre
- El paper del generador i del discriminador en un sistema GAN
- Avantatges i inconvenients de les funcions de pèrdues GAN habituals
- Possibles solucions als problemes habituals de l'entrenament de GANs

En aquest repositori podem veure més [exemples de TF-GAN](#).