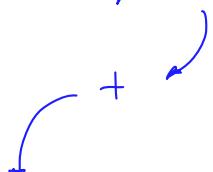


Percepción ~ regresión lineal



\downarrow

activaciones \rightarrow parte no lineal

} modelo capaz de "aprender"

¿Por qué se retroalimenta la IA?

- Potencia computacional \rightarrow mejor HW
- Datos \rightarrow un poñón de información

$$\text{Output} = f(b + \sum_i x_i \cdot w_i)$$

$$\phi = f(b + x_1 w_1 + x_2 w_2)$$

$$f(x) = \begin{cases} 1 & \text{si } x > 0 \\ 0 & \text{resto.} \end{cases}$$

tomo $w_1 = 1$ y $w_2 = 2$

$$\hookrightarrow \phi = f(b + x_1 + 2x_2)$$

x_1, x_2

$$0, 0 \quad f(b) = 0 \Rightarrow b < 0$$

$$0, 1 \quad f(b+2) = 0 \Rightarrow b+2 < 0 \Rightarrow b < -2$$

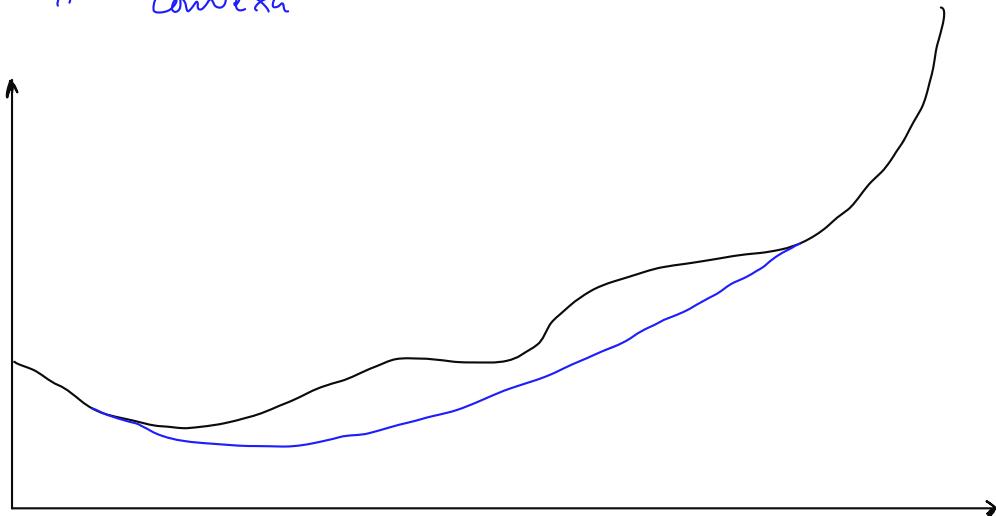
$$1, 0 \quad f(b+1) = 0 \Rightarrow b+1 < 0 \Rightarrow b < -1$$

$$1, 1 \quad f(b+1-1) = 1 \Rightarrow b+1+2 > 0 \Rightarrow b > -3$$

Percepción simple \rightarrow ¿Hay un mínimo global?

Depende de los pesos y la activación, se puede sobreparametrizar

- función no convexa
- .. convexa



2023 - 09 - 25

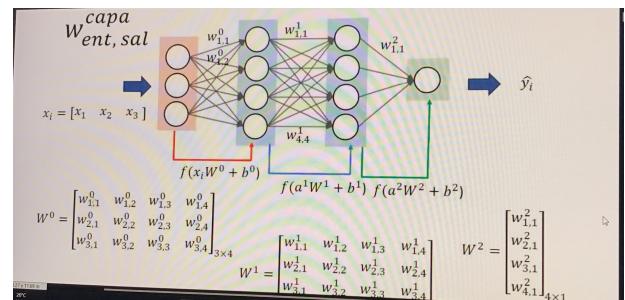
Sesión 2

2:31 PM

- Estructura de una red neuronal. \rightarrow pdf: ForwardBackprop.pdf.

$w_{k_1, k_2}^l \rightarrow$ número de capa
 $w_{k_1, k_2}^l \rightarrow$ neurona capa siguiente
 \hookrightarrow neurona capa actual

$$f(x_i W^0 + b^0) \quad \text{notación} \quad \longrightarrow$$



idea: $x^1 = x^0 - \eta \cdot \frac{\partial f}{\partial x} \Big|_{x^0}$: $\eta \in \text{learning rate}$

$$\underline{\underline{\delta_{a_1^2}}} = \frac{\partial E}{\partial a_1^2} \cdot \frac{\partial a_1^2}{\partial z_1^2}$$

Pasos

1. Forward
2. Backward
3. Step \rightarrow actualizamos pesos

Sistemas multi-task

Hiperparámetros \rightarrow se ajustan de forma empírica

Parámetros \rightarrow " " automáticamente.

Funciones de activación:

- identity
- step \rightarrow ya no se usa pues no es derivable
mejor \downarrow sigmoid.
- softmax \rightarrow cuando hay mas de una clase \Rightarrow probabilidad de cada neurona de salida
- Sigmoidide {
• tanh } sobre todo en capa final pues satura
- ReLU \rightarrow mejor, pero aún así, que la derivada sea 0 para valores negativos realentiza aprendizaje
- ELU
- Leaky ReLU
- ...

A la salida

Regresión

- identity
- sigmoidide \rightarrow normalizar entre 0 y 1

Clasificación

- sigmoidide
- tanh.
- softmax \rightarrow multiclase

Optimizadores:

- RMSprop *
- Momentum
- NAG
- Adadelta *
- Adagrad
- Adam **
- AdamW** (weighted)

* avanzados \rightarrow buenos

** muy buenos.

07MIAR – Redes neuronales y deep learning



Universidad
Internacional
de Valencia

Tareas avanzadas de *computer vision*
empleando aprendizaje profundo

01

Introducción

Tareas avanzadas de *computer vision* empleando aprendizaje profundo

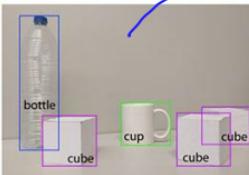
25/10/2021

Tareas avanzadas CV

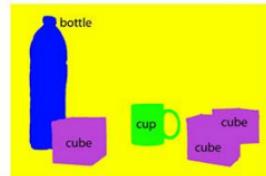
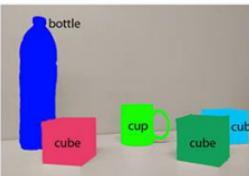
- Dentro del campo de la **visión por computador** existen diversas **tareas de interés** que tienen como denominador común el **tratamiento con los objetos** de una **escena** tanto **estática** como en **movimiento**.



Clasificación de imagen



Localización de objetos

Segmentación
semánticaSegmentación de
instancia

multitask
sigmoid



Tracking o localización dinámica

Generación sintética de imágenes, denoising,
compresión, detección de anomalías, etc.



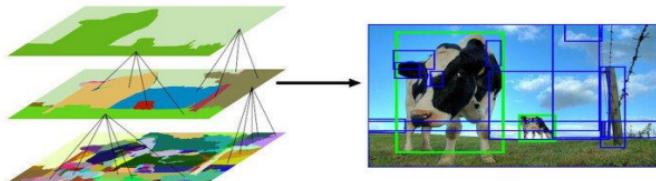
02

Detección de objetos

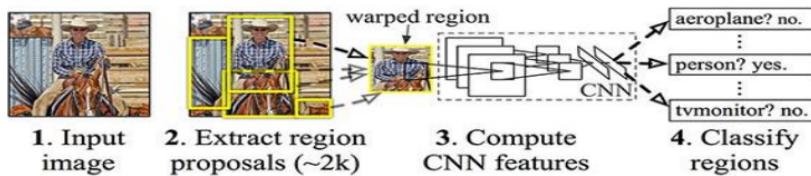
Tareas avanzadas de *computer vision* empleando aprendizaje profundo

R-CNN: Regiones con características CNN

- Método propuesto por Ross Girshick en **2013**. Se compone de los siguientes pasos:
 1. **Extracción de regiones candidatas** mediante el **algoritmo** de búsqueda selectiva. Agrupación jerárquica de regiones similares basada en color, textura, tamaño y forma.



2. **Uso** de técnicas de *transfer learning* (extracción de características) sobre las regiones candidatas empleando una **arquitectura pre-entrenada** (AlexNet).
3. **Clasificación** de cada región candidata empleando las **características extraídas** y Support Vector Machine (**SVM**).



Fast R-CNN: Regiones con características CNN

- Gran **desventaja R-CNN**: Tiene que **clasificar todas las regiones candidatas** que se extraen del método de búsqueda selectiva (i.e. **cuello de botella**).
- Girshick et al. (2015) proponen una **mejora de R-CNN** que trata de disminuir el alto coste computacional de dicho método. Proponen un **módulo de pooling sobre el mapa de activación** que obtienen al pasar la imagen por la red pre-entrenada (**proyecta ROIs**).
- El proceso de **clasificación** pasan de hacerlo con SVM a emplear un **perceptrón multicapa** que **predice la clase y la bounding box (offset)**. Consiguen un **aproximación entrenable end to end**.

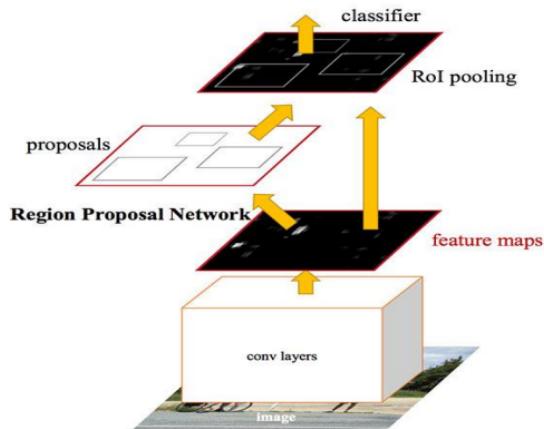
Fast-CNN



- El **performance** del proceso de **inferencia** (i.e. fase de predicción) sigue sufriendo dramáticamente debido a la **dependencia** en el **algoritmo** de **búsqueda selectiva**.

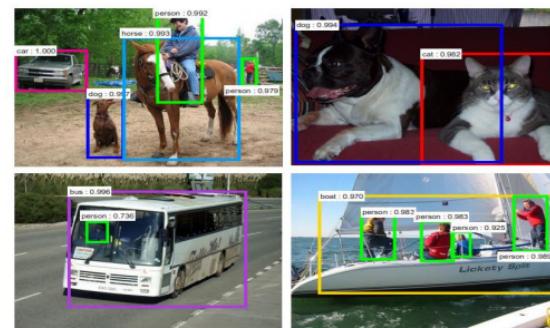
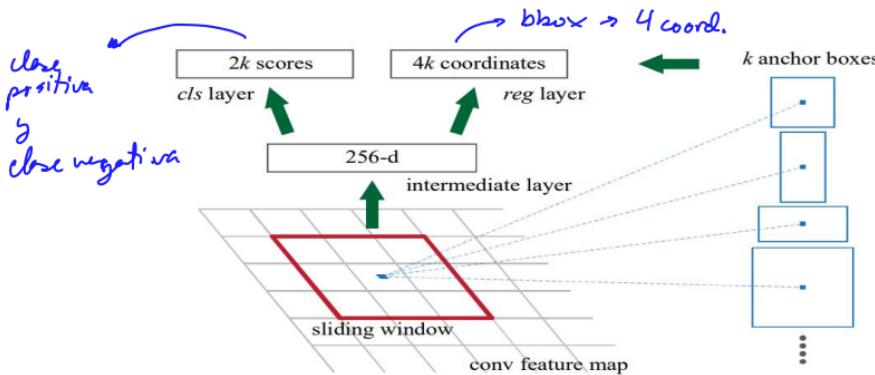
Faster R-CNN: Regiones con características CNN

- En el artículo donde Girshick et al. (2015) proponen **Faster R-CNN**, presentan la **Region Proposal Network (RPN)** que introduce la propuesta de regiones directamente en la arquitectura (**sustituyendo** el algoritmo de **búsqueda selectiva**).
- Supone una contribución muy importante porque la **Faster R-CNN** es capaz de **predecir 7-10 FPS** haciendo posible la detección de objetos en **tiempo real**.



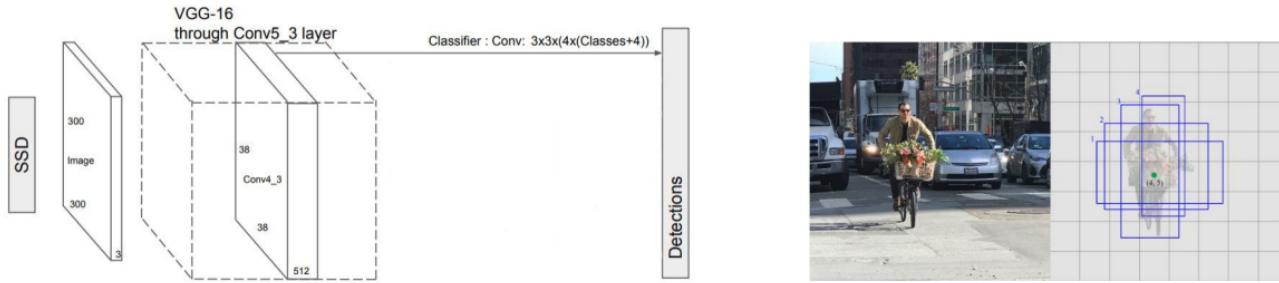
Faster R-CNN: Regiones con características CNN

- Sobre el mapa de activación se pasa una **ventana deslizante** y se generan k candidatos o **anchors**. Dichos candidatos **se evalúan simultáneamente** en la **RPN**.
- La **RPN** es un **perceptrón multicapa** con **dos fully-connected de salida**. Una de ellas se encarga de dar una **predicción** de si la **región** contiene o no **contiene objeto** mientras que la otra se encarga de predecir los **bordes de la región**.
- En el artículo emplean **3 escalas** y **3 relaciones de aspecto** para generar $k=9$ anchors a partir de una **ventana deslizante** de tamaño 3×3 .



SSD: Single Shot Multibox Detector

- Método propuesto por **Wei Liu** en **2016**. **Elimina** la necesidad de una **red generadora de candidatos**. La red se compone de dos niveles:
 - Extracción de mapas de activación (características) mediante red pre-entrenada (VGG16)
 - Aplicar **filtros convolucionales** (3x3) sobre el mapa de activación para detectar objetos.
- Para cada una de las **38 x 38 celdas** o localizaciones se obtienen **4 predicciones de objeto (4 filtros)** siguiendo una serie de **bounding box predefinidas (a conciencia)**.

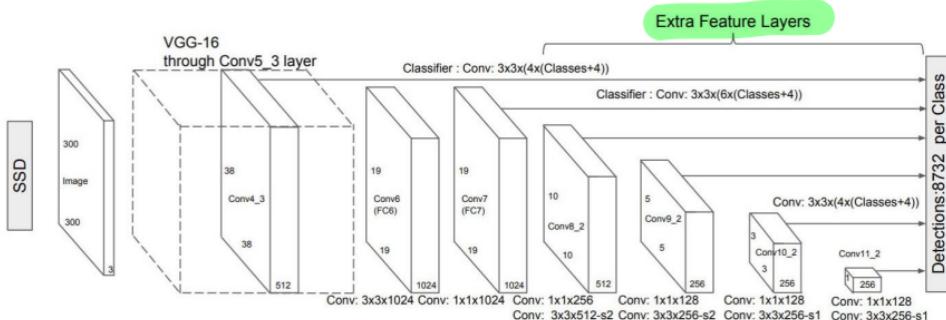


- Cada predicción está compuesta por una **bounding box (offset)** y **21 scores** de pertenencia a clase (**20 clases + fondo**).

→ desventaja o problema: resolución espacial, [↑] más gasto de memoria con el mapa de caract. de salida de VGG → pierde precisión

SSD: Single Shot Multibox Detector

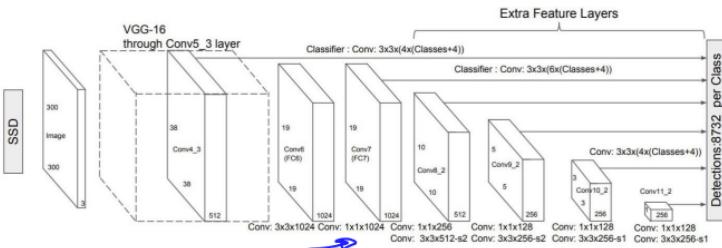
- La gran **potencia** de la red **SSD** reside en el uso de **múltiples capas** con el objetivo de realizar una **detección multiescala**.
- Concretamente **SSD añade 6 capas convolucionales** sobre el mapa de activación que obtiene de la red pre-entrenada y **sobre cinco** de ellas aplica la **detección de objetos** tal y como hemos visto anteriormente (**en 3 de ellas** hace **6 predicciones** por celda en vez de 4). Un total de **8732 predicciones**.



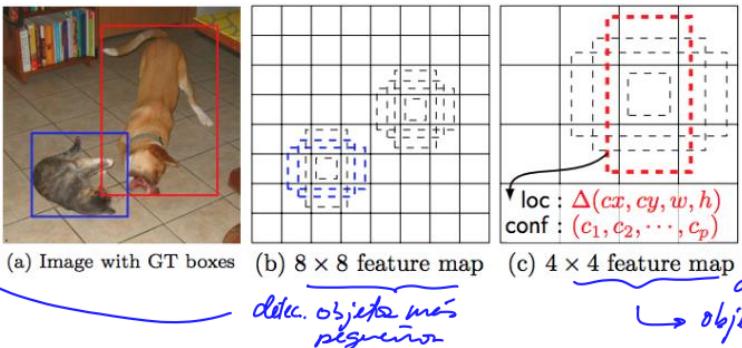
- Estas **capas convolucionales** van disminuyendo las dimensiones del mapa de características gradualmente (*stride* ≠ 1).

>

SSD: Single Shot Multibox Detector

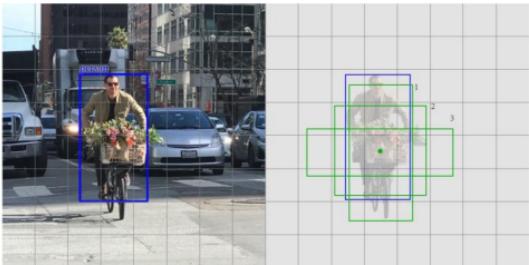


- SSD usa las **resoluciones bajas** para detectar **objetos grandes** mientras que los **objetos pequeños** son detectados en los **primeros mapas** de activación.



SSD: Single Shot Multibox Detector

- En la **fase de entrenamiento**, el coste para **optimizar la localización** solo se calcula sobre las **coincidencias positivas**. Una coincidencia positiva se define por tener una **IoU > 0.5 entre la BB por defecto y la del GT**.



- Una vez identificadas las **coincidencias positivas**, estas se emplean para **calcular el error** con respecto a la **BB predecida**.
- En la **fase de test**, SSD utiliza la técnica **non-máximo suppression** (algoritmo basado en el **nivel de confiancia** y en la **IoU entre predicciones**) para **eliminar predicciones duplicadas** sobre un mismo objeto.



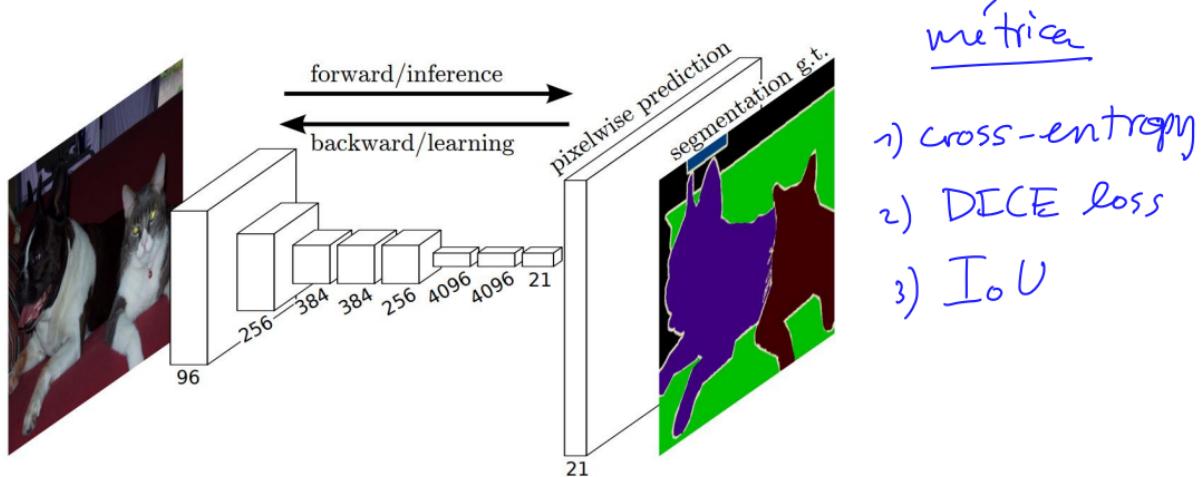
03

Segmentación semántica y de instancia

Tareas avanzadas de *computer vision* empleando aprendizaje profundo

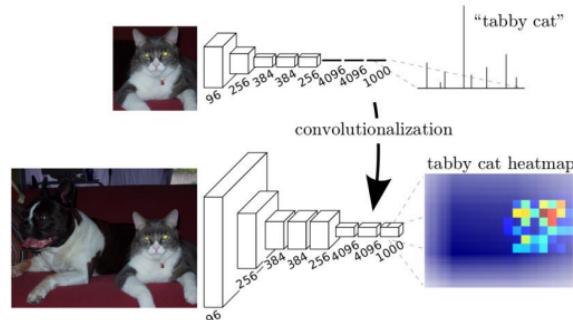
FCNN para la segmentación semántica

- J. Long et al. (2015) fueron los primeros en utilizar una **arquitectura** basada exclusivamente en **capas convolucionales** y de **pooling** para realizar tareas de **segmentación semántica**.
- Este tipo de red recibe como **entrada** una **imagen** de un **determinado tamaño** y como **salida** devuelve la **imagen segmentada** del **mismo tamaño**.



FCNN para la segmentación semántica

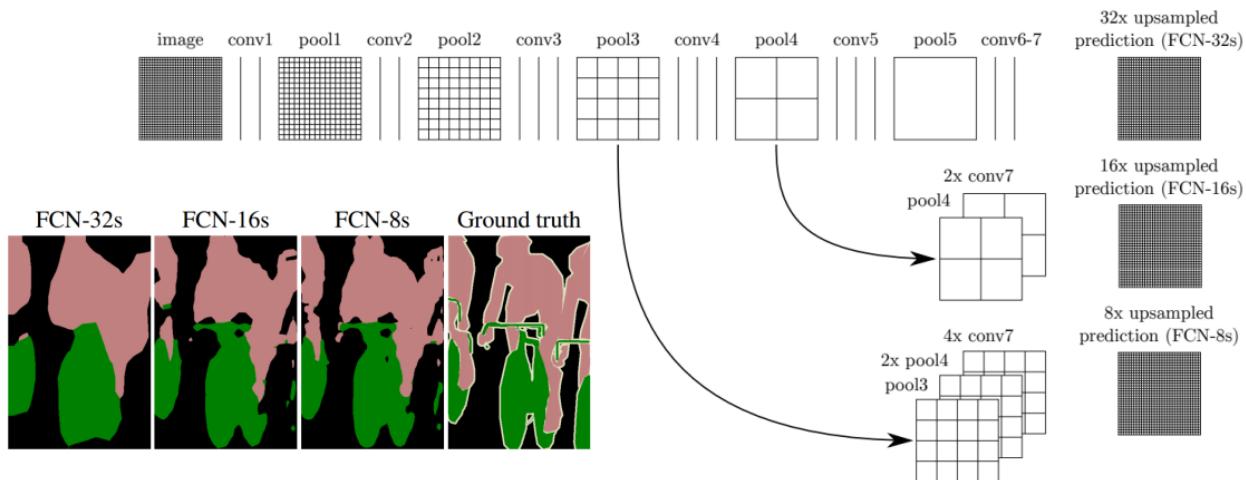
- Los autores **modifican** redes conocidas como **AlexNet**, **VGG16** o **GoogleNet** eliminando el **top model** (destinado a clasificación) y **reemplazándolo** por más **bloques convolucionales** produciendo **pequeños mapas** de características con representaciones **densas**.



- Cuando se llega al final de la red, **al último mapa** se le debe aplicar un **upsampling** para llevarlo a las **dimensiones espaciales originales** de la imagen.
- Se emplea una **capa convolucional** con un valor de **stride = 1/f** consiguiendo **ampliar** el mapa de activación **por un factor f**. Esta técnica es conocida como **deconvolución**. Los **filtros** aprendidos en estas capas constituyen las **bases** para **reconstruir la forma**.

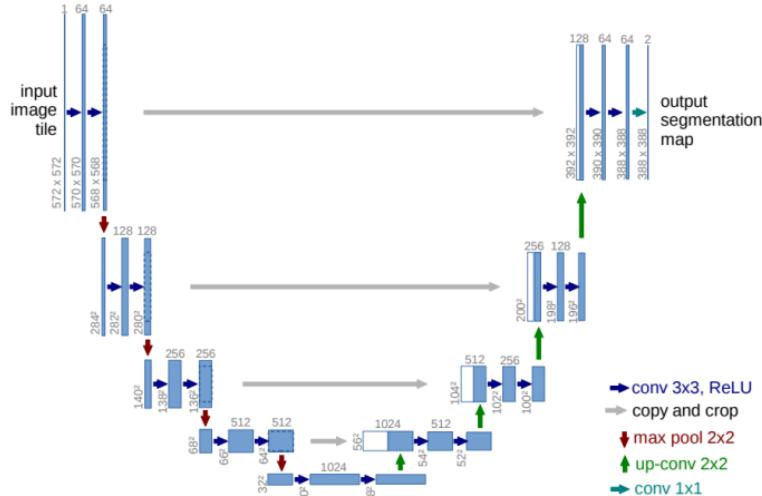
FCNN para la segmentación semántica

- La red es entrenada empleando una función de pérdidas a nivel de pixel (por ejemplo **Dice**).
- Además, los autores introducen **skip connections** en la red con el objetivo de **combinar mapas de características** que contienen **representaciones de alto y bajo nivel** que posteriormente fusionan dando lugar a la segmentación final.



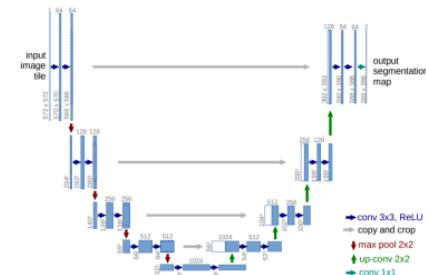
Segmentación semántica: U-net

- O. Ronneberger et al. (2015) extienden el trabajo anterior a imágenes de microscopía. Los autores crean U-net que se compone de **dos partes**: el **contracting path** encargado de codificar la **información relevante** de la imagen y el **expansive path** que localiza **espacialmente** los **patrones relevantes** en la imagen.



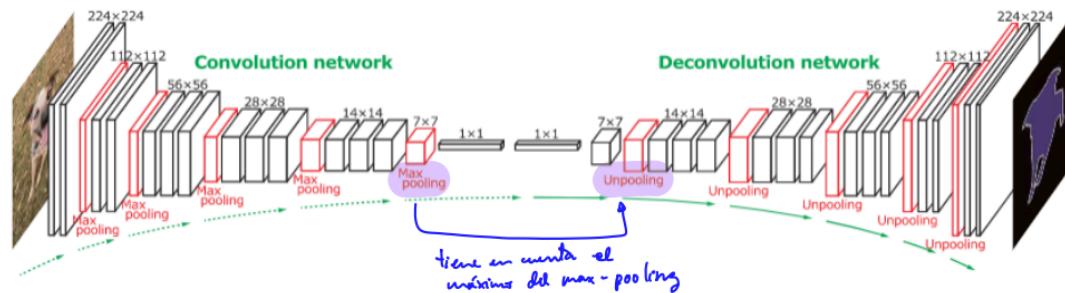
Segmentación semántica: U-net

- El **downsampling** o **contracting path** esta basado en una FCN con **tamaño de filtros constante** (i.e. 3x3).
- El **upsampling** o **expansive path** emplea **up-convolutions** (i.e. deconvoluciones, convoluciones transpuestas) reduciendo el número de mapas de características a la vez que aumenta el tamaño espacial.
- Una importante contribución es que para evitar pérdida de información, copian los mapas de cada bloque convolucional del **encoder** en su correspondiente mapa en el **decoder**. Esta información se concatena en la **tercera dimensión** gracias a las **skip connections**.
- Por último, una **capa convolucional 1x1** mapea las dimensiones del **último mapa de activación** a tantos **mapas 2D** como **clases** se tengan. Posteriormente, aplicando la función **softmax** se **categoriza** cada uno de los **pixels**, obteniendo la segmentación semántica.



Segmentación de instancia: Redes Conv-Deconv

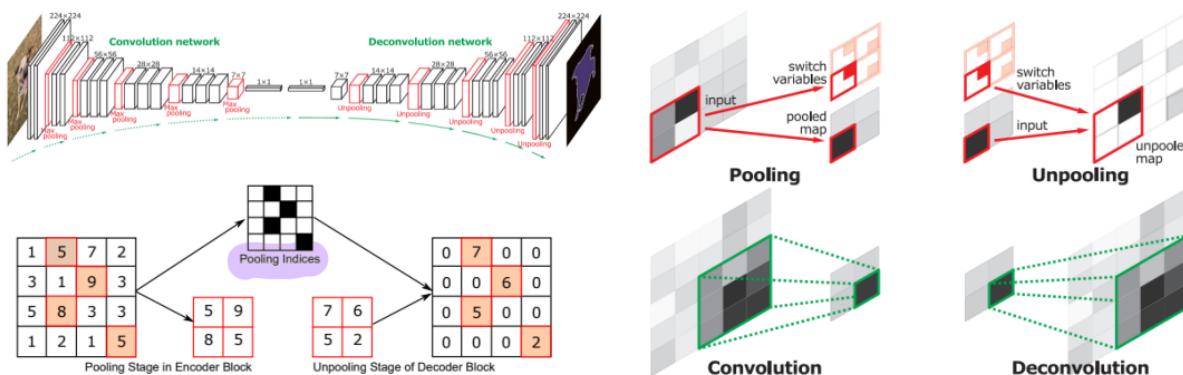
- H. Noh et al. (2015) diseñan una **red end to end** compuesta por dos niveles para realizar segmentación de instancia. La primera de ellas es un módulo detector de objetos mediante VGG16.
- La red que proponen para la **segmentación** se compone de **dos fases**, la de **convolución** y la **deconvolución**.



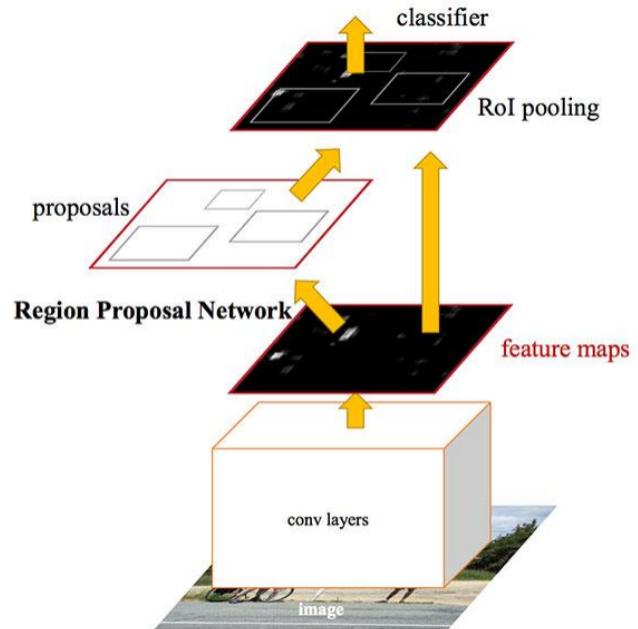
- Una **región candidata** es procesada por el **encoder** generando un **vector de características**.

Segmentación de instancia: Redes Conv-Deconv

- El **decoder** toma dicho vector de características y **genera** un **mapa a nivel de pixel** con la **probabilidad de pertenencia** a clase.
- La **subred de deconvolución** emplea la operación ***unpooling*** localizando las activaciones máximas para mantener su posición original en el mapa de características que va ampliando. **Tras la capa *unpooling*** se insertan varias **capas deconvolucionales** para **mantener** la **densidad** de la **información**.



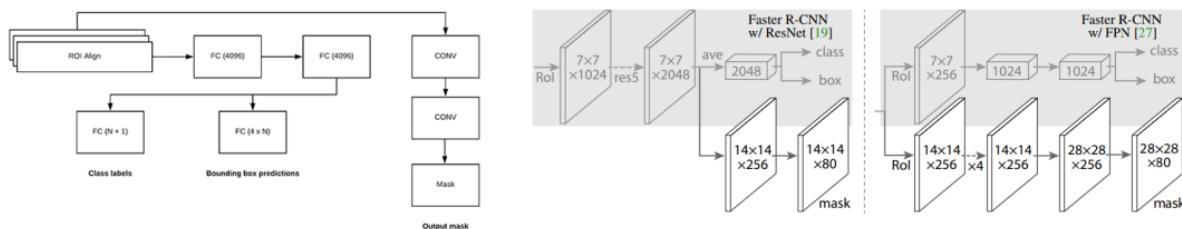
Faster R-CNN: Regiones con características CNN



Mask R-CNN para la segmentación de instancia

- La **Mask R-CNN** es una arquitectura para la **segmentación de instancia** fue propuesta por He et al. (2018) y nace a partir de la Faster R-CNN:
 1. Se reemplaza el módulo **ROI pooling** por el módulo **ROI align** mucho **más preciso** para el propósito de **segmentación**.
 2. Se **inserta** una **rama adicional** a la salida del nuevo módulo que **realiza la segmentación**.

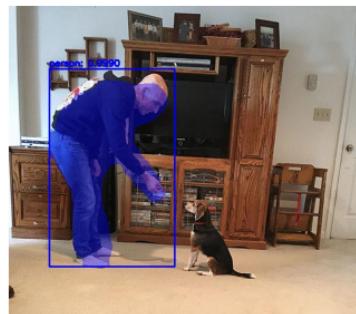
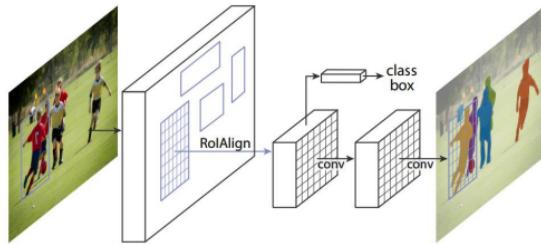
Mask R-CNN



- La **salida** de la **rama convolucional** es la **máscara** de la segmentación del **objeto** previamente detectado (objeto vs fondo), se obtiene aplicando la función **sigmoide** a cada **pixel** del **último mapa** de activación.

Mask R-CNN para la segmentación de instancia

- La **Mask R-CNN** es una arquitectura para la **segmentación de instancia** fue propuesta por **He et al.** (2018) y nace a partir de la Faster R-CNN:
 1. Se reemplaza el módulo ***ROI pooling*** por el módulo ***ROI align*** mucho más preciso para el propósito de **segmentación**.
 2. Se inserta una **rama adicional** a la salida del nuevo módulo que realiza la **segmentación**.



- La **salida de la rama convolucional** es la **máscara** de la segmentación del objeto previamente detectado (objeto vs fondo), se obtiene aplicando la función **sigmoide** a cada **pixel** del **último mapa** de activación.



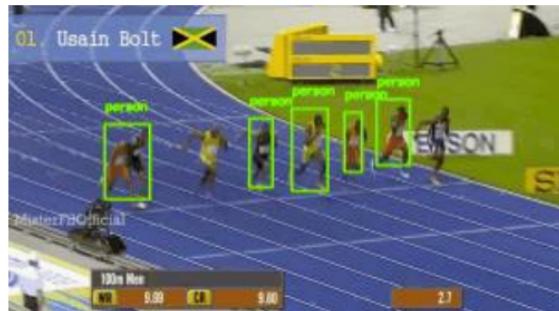
04

Tracking de objetos

Tareas avanzadas de *computer vision* empleando aprendizaje profundo

Introducción

- La tarea de **tracking** se basa en aplicar la técnica de **detección de objetos** (DO) durante una serie de **frames** consecutivos. La problemática reside en el **tiempo de procesado** del algoritmo de DO para realizar dicha tarea en **tiempo real**.



Faster R-CNN vs SSD

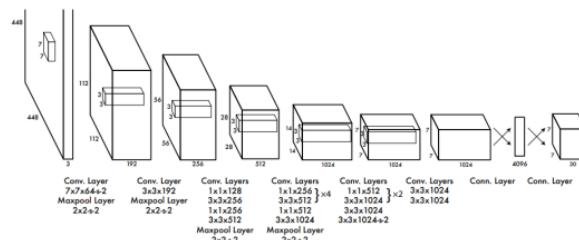
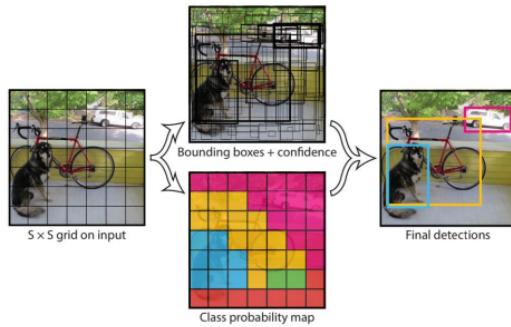
- Si rescatamos las dos metodologías basadas en aprendizaje profundo más comunes de la literatura (i.e. **Faster R-CNN** y **SSD** en sus múltiples versiones) y las **analizamos** desde el punto de vista tanto de **precisión** como de **frames por segundo (FPS)** procesados se puede observar que **Faster R-CNN** es un **método de detección de objetos estático** o que únicamente se podría aplicar a tracking en **videos time-lapse**.

Method	mAP	FPS	batch size	# Boxes	Input resolution
Faster R-CNN (VGG16)	73.2	7	1	~ 6000	~ 1000 × 600
SSD300	74.3	46	1	8732	300 × 300
SSD512	76.8	19	1	24564	512 × 512
SSD300	74.3	59	8	8732	300 × 300
SSD512	76.8	22	8	24564	512 × 512

- **Compromiso** entre **precisión** en la detección y **FPS** procesados. Por este motivo nace YOLO (You Only Look Once).

YOLO: You Only Look Once

- Introducida por **Redmon et al.** (2015) se basa en una **estrategia single-stage** (i.e. una única red para todo el proceso, al igual que SSD).
- División de la **imagen** en un **grid $S \times S$** . Para cada celda **se predicen B bounding boxes** y un **nivel de confidencia** (i.e. si hay o no objeto y que objeto de todas las clases es).
- **CNN para extraer las características**, las predicciones se realizan mediante **dos FC layers** después del último bloque convolucional.



YOLO: You Only Look Once

YOLO es mejor sólo si vamos en real time

- Los autores afirman que **YOLO** realiza una detección de objetos **super real-time** obteniendo **45 FPS** en una GPU. Desarrollan también una **variante** más **ligera** que corre a **155 FPS**.
- Existen YOLOv2 (2016) y YOLOv3 (2018) que **mejoran la precisión** en la detección de la primera versión ya que se entranen en **datasets más grandes** (i.e. COCO).

Real-Time Detectors	Train	mAP	FPS
100Hz DPM [31]	2007	16.0	100
30Hz DPM [31]	2007	26.1	30
Fast YOLO	2007+2012	52.7	155
YOLO	2007+2012	63.4	45

Less Than Real-Time			
	Train	mAP	FPS
Fastest DPM [38]	2007	30.4	15
R-CNN Minus R [20]	2007	53.5	6
Fast R-CNN [14]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[28]	2007+2012	73.2	7
Faster R-CNN ZF [28]	2007+2012	62.1	18
YOLO VGG-16	2007+2012	66.4	21

Backbone	Top-1	Top-5	Bn Ops	BFLOP/s	FPS
Darknet-19 [15]	74.1	91.8	7.29	1246	171
ResNet-101[5]	77.1	93.7	19.7	1039	53
ResNet-152 [5]	77.6	93.8	29.4	1090	37
Darknet-53	77.2	93.8	18.7	1457	78

Type	Filters	Size/Stride	Output
Convolutional	32	3 × 3	224 × 224
Maxpool		2 × 2/2	112 × 112
Convolutional	64	3 × 3	112 × 112
Maxpool		2 × 2/2	56 × 56
Convolutional	128	3 × 3	56 × 56
Convolutional	64	1 × 1	56 × 56
Convolutional	128	3 × 3	56 × 56
Maxpool		2 × 2/2	28 × 28
Convolutional	256	3 × 3	28 × 28
Convolutional	128	1 × 1	28 × 28
Convolutional	256	3 × 3	28 × 28
Maxpool		2 × 2/2	14 × 14
Convolutional	512	3 × 3	14 × 14
Convolutional	256	1 × 1	14 × 14
Convolutional	512	3 × 3	14 × 14
Convolutional	256	1 × 1	14 × 14
Convolutional	512	3 × 3	14 × 14
Maxpool		2 × 2/2	7 × 7
Convolutional	1024	3 × 3	7 × 7
Convolutional	512	1 × 1	7 × 7
Convolutional	1024	3 × 3	7 × 7
Convolutional	512	1 × 1	7 × 7
Convolutional	1024	3 × 3	7 × 7
Convolutional	1000	1 × 1	7 × 7
Avgpool		Global	1000

DarkNet-19 en YOLOv2

Type	Filters	Size	Output
Convolutional	32	3 × 3	256 × 256
Convolutional	64	3 × 3 / 2	128 × 128
Convolutional	32	1 × 1	
Convolutional	64	3 × 3	
Residual			128 × 128
Convolutional	128	3 × 3 / 2	64 × 64
Convolutional	64	1 × 1	
Convolutional	128	3 × 3	
Residual			64 × 64
Convolutional	256	3 × 3 / 2	32 × 32
Convolutional	128	1 × 1	
Convolutional	256	3 × 3	
Residual			32 × 32
Convolutional	512	3 × 3 / 2	16 × 16
Convolutional	256	1 × 1	
Convolutional	512	3 × 3	
Residual			16 × 16
Convolutional	1024	3 × 3 / 2	8 × 8
Convolutional	512	1 × 1	
Convolutional	1024	3 × 3	
Residual			8 × 8
Connected		Global	1000
Softmax			

DarkNet-53 en YOLOv3



05

Otras tareas

Tareas avanzadas de *computer vision* empleando aprendizaje profundo

Autoencoders

- Los **autoencoders** son un tipo de red neuronal no supervisada que tienen como objetivo comprimir los datos de entrada en una representación denominada **espacio latente**.
- El **espacio latente** se caracteriza por una **dimensionalidad mucho menor** que la dimensionalidad de los **datos de entrada**.
- La idea de un **autoencoder es reconstruir los datos** a partir del espacio latente entrenado minimizando cierta función de error (MSE, MAE, etc.).



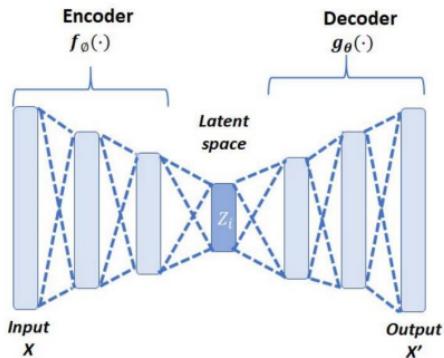
Aplicaciones autoencoders

- **Reducción de dimensionalidad:** Mapeo **no lineal** de los datos de entrada a un nuevo espacio vectorial que describe gran parte de la variabilidad del set de datos de entrada.
- **Denoising:** Eliminar ruido de un set de imágenes a la entrada. El **encoder** caracteriza la **distribución del ruido** y el **decoder** es capaz de generar **muestras sin dicho ruido** gracias al espacio latente generado.
- **Compresión de datos:** Generar nuevas representaciones de datos reducidas a partir del espacio latente.
- **Detección de anomalías/outliers:** Detectar **datos clasificados erróneamente** o detectar cuando un **dato** a la entrada **no sigue la distribución** típica de la población.
- Sistemas **Content Based Image Retrieval (CBIR)**: Creación de sistemas para la recuperación automática de información. Basado en **similitudes entre una Query y un diccionario de representaciones**.
↳ nueva muestra
- **Natural Language Processing**: Comprensión de texto, construcción de Word embeddings, o resumen de textos.

Autoencoder convencional

Una red **autoencoder** esta caracterizada por **dos niveles o subredes** (al igual que las arquitecturas para segmentación de imagen):

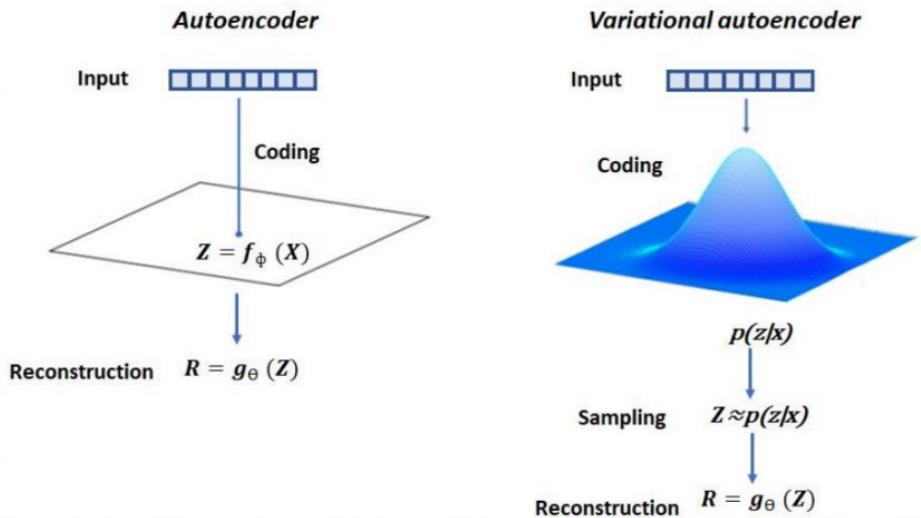
- **Encoder** - $f_\theta(\cdot)$: Comprime los datos de entrada en un espacio latente (Z_i) mediante $Z_i = f_\theta(X)$.
- **Decoder** - $g_\theta(\cdot)$: Tiene como entrada el espacio latente (Z_i) y se encarga de reconstruir una imagen de salida a partir de este según $g_\theta(Z_i)$.
- El **proceso completo** de un **autoencoder** queda definido por $g_\theta(f_\theta(X))$ y el proceso de optimización se basa en **minimizar el error** entre los datos reconstruidos y los originales.



$$\min_{\theta, \phi} L_{rec} = \min \frac{1}{n} \sum_{i=1}^n \|x_i - g_\theta(f_\phi(x_i))\|^2$$

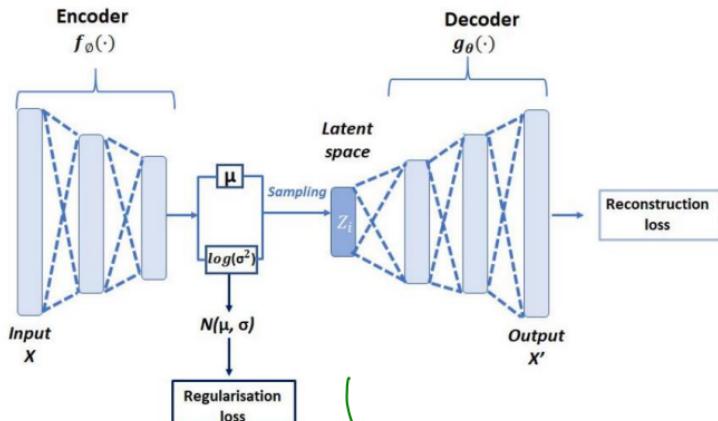
Autoencoder variacional

- La versión variacional de un autoencoder (**VAE** del inglés) introduce una **regularización** en el espacio latente para mejorar sus propiedades. VAE codifica los datos de entrada como una **distribución normal multivariante** alrededor de un punto en el espacio latente.



Autoencoder variacional

- El **encoder** asigna cada muestra de entrada a un vector de medias y otro de varianzas.
- Necesidad de **regularizar** tanto el logaritmo de la varianza como la media de la distribución que devuelve el **encoder** → **Match** entre **distribución** que saca el **encoder** y una **distribución normal estándar** (media cero y desviación unidad).
- **Sampling** de la **distribución multivariante** para reconstruir los datos originales.



$$Z \approx p(z|x) = \mu + \sigma \cdot \epsilon$$

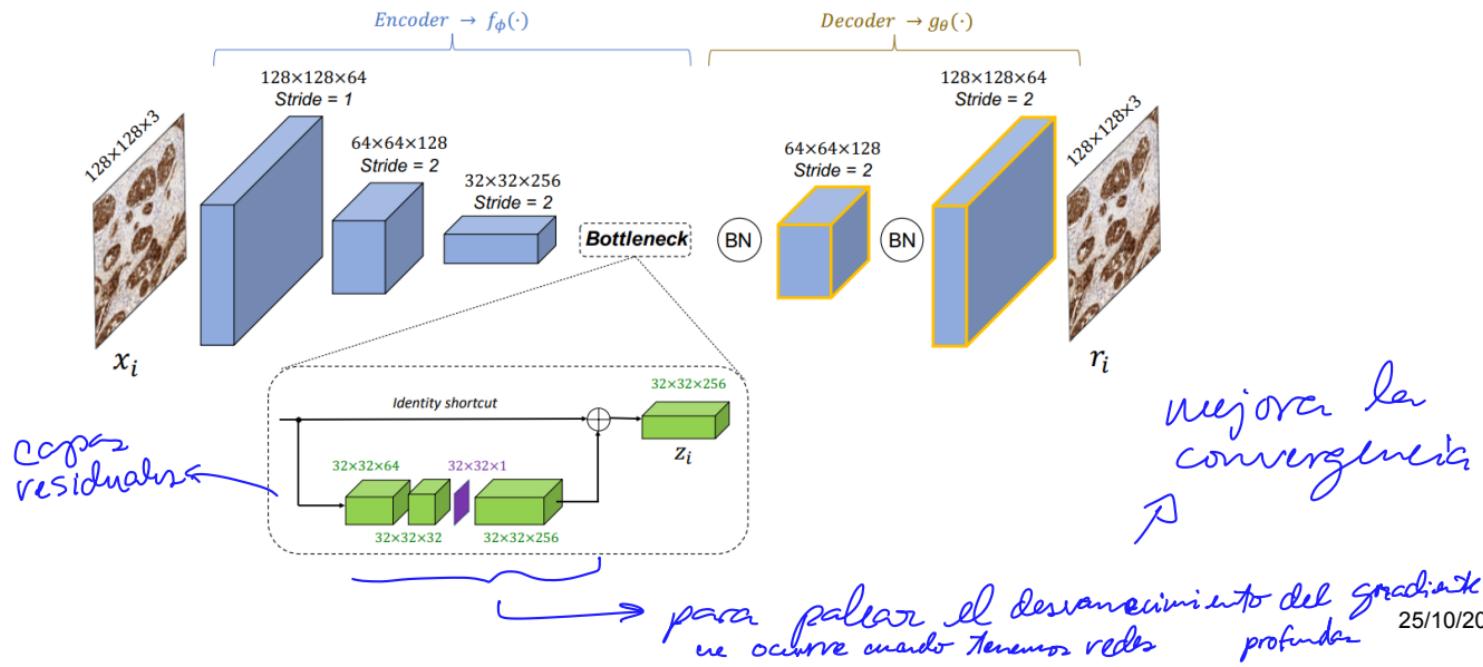
$$\min_{\theta, \phi} L_{rec} = \min \frac{1}{n} \sum_{i=1}^n \|x_i - g_\theta(f_\phi(x_i))\|^2$$

$$D_{KL}[N(\mu, \sigma) || N(0, 1)] = \frac{1}{2} \sum (1 + \log(\sigma^2) - \mu^2 - \sigma^2)$$

reparameterization tree

Autoencoder convolucional

- La arquitectura de los **autoencoders** varían según el caso de uso, más concretamente según el tipo de datos a la entrada.



07MIAR – Redes Neuronales y Deep Learning



Universidad
Internacional
de Valencia

Deep Learning para texto y secuencias lógicas

De:

Planeta Formación y Universidades

Contenidos

1. Procesamiento del Lenguaje Natural (NLP)
2. De texto a representaciones numéricas
3. Word2Vec
4. Redes neuronales recurrentes
5. Introducción a transformers

Contenidos

- 1. Procesamiento del Lenguaje Natural (NLP)**
2. De texto a representaciones numéricas
3. Word2Vec
4. Redes neuronales recurrentes
5. Introducción a *transformers*

Procesamiento del Lenguaje Natural (NLP)

- El **Natural Language Processing** (NLP) es la **intersección** entre los campos de las **ciencias de la computación**, la **inteligencia artificial** y la **lingüística**. El NLP estudia las **interacciones** entre las **computadoras** y el **lenguaje humano**
- Se ocupa de la formulación e investigación de mecanismos eficaces computacionalmente para la **comunicación** entre **personas** y **máquinas** por medio del lenguaje natural, es decir, de las **lenguas del mundo**
- Dentro del NLP se identifican una serie de **tareas** a resolver:

- Speech Recognition
- Speech to text
- Machine Translation
- Text classification
- Sentiment analysis
- Text generation (e.g. Q&A)
- Text recognition (OCR)
- Image/Video understanding
- Text summarization

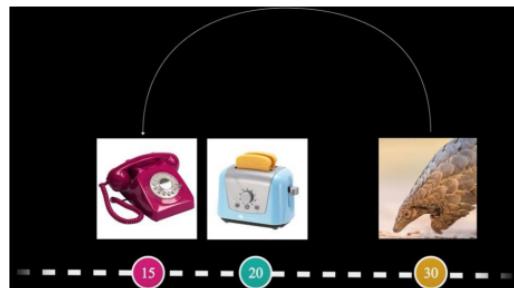


Contenidos

1. Procesamiento del Lenguaje Natural (NLP)
- 2. De texto a representaciones numéricas**
3. Word2Vec
4. Redes neuronales recurrentes
5. Introducción a *transformers*

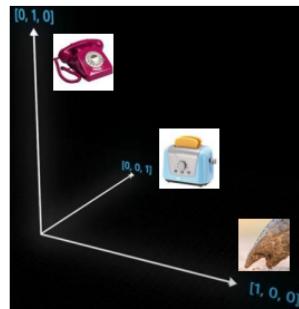
De texto a representaciones numéricicas

- Tal y como hemos visto a lo largo del curso, una **red neuronal** (independientemente de su arquitectura) viene **definida por** una serie de **parámetros numéricos** a optimizar. Dichos parámetros **restringen el tipo de** datos a la **entrada**. Una red neuronal no sabe manejar texto a la entrada
- La **solución más intuitiva** es **asignar un número entero** (etiqueta categórica) **a cada** una de las **unidades de texto** con las que se trabaje (carácter/palabra)
- Esta **solución no** es **válida** desde el punto de vista de una red neuronal puesto que la **red** intrínsecamente extrae **patrones de ordenación** de los **datos numéricos** mientras que el **texto** sigue una **secuencia lógica gramatical**.



De texto a representaciones numéricas

- Una solución para dotar de **independencia** a la codificación texto – dato numérico es la codificación **one-hot encoding**. Mediante esta codificación podemos establecer **una dimensión a cada carácter/palabra**
- Como sabemos esta codificación se compone de un **vector** de tantos **ceros** como palabras distintas se quieran codificar, indicando **con un 1** cada palabra diferente



dog = [0 0 0 0 0 1 0 0 0 0 0 0 ... 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

cat = [0 0 0 0 0 0 0 0 0 0 0 0 ... 0 0 0 0 0 0 1 0 0 0 0 0]

De texto a representaciones numéricas

- El problema de este tipo de codificación es que en **NLP** vamos a manejar **grandes vocabularios** de texto (**Corpus**) para el entrenamiento de modelos

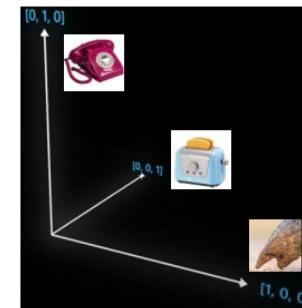
dog = [0 0 0 0 0 0 **1** 0 0 0 0 0 0 ... 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

cat = [0 0 0 0 0 0 0 0 0 0 0 0 ... 0 0 0 0 0 0 0 0 0 0 0 0 0 **1** 0 0 0 0 0]



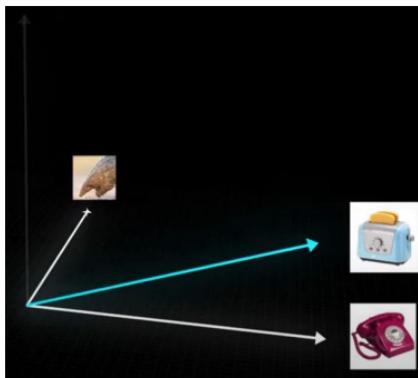
Vocabulary size

- Otro inconveniente es que **palabras similares no tienen** porque estar representadas por **vectores cercanos**
- **Tantas dimensiones como palabras** a codificar
- **Vectores muy dispersos** (todo ceros menos un uno)



Word vectors/embeddings

- ¿Cómo podemos evitar las limitaciones anteriores?
 - IDEA: Mapear palabras en un nuevo **espacio vectorial** más **denso** o concentrado (**Word embeddings**)
 - OBJETIVO: Obtener una representación de palabras que obtenga provecho de la “**similitud semántica**” de las mismas
 - ¿CÓMO?: Resolviendo un problema de **aprendizaje Supervisado** *auto-supervisado*
 - VENTAJAS: **Vectores** más pequeños y **compactos**



Contenidos

1. Procesamiento del Lenguaje Natural (NLP)
2. De texto a representaciones numéricas
- 3. Word2Vec**
4. Redes neuronales recurrentes
5. Introducción a *transformers*

Word2Vec

- ¿Cómo podemos obtener estas representaciones densas de palabras (**Word embeddings**)?
- El **algoritmo Word2Vec** es el más popular en la literatura para entrenar *Word embeddings* con propósitos generalistas
- Se basa en **predecir el vecindario de palabras** para cada una de las palabras que componen un texto
- Existen **dos versiones** de este algoritmo:
 - **Continuous bag of words (CBOW)**
 - **Modelo Skip-gram**



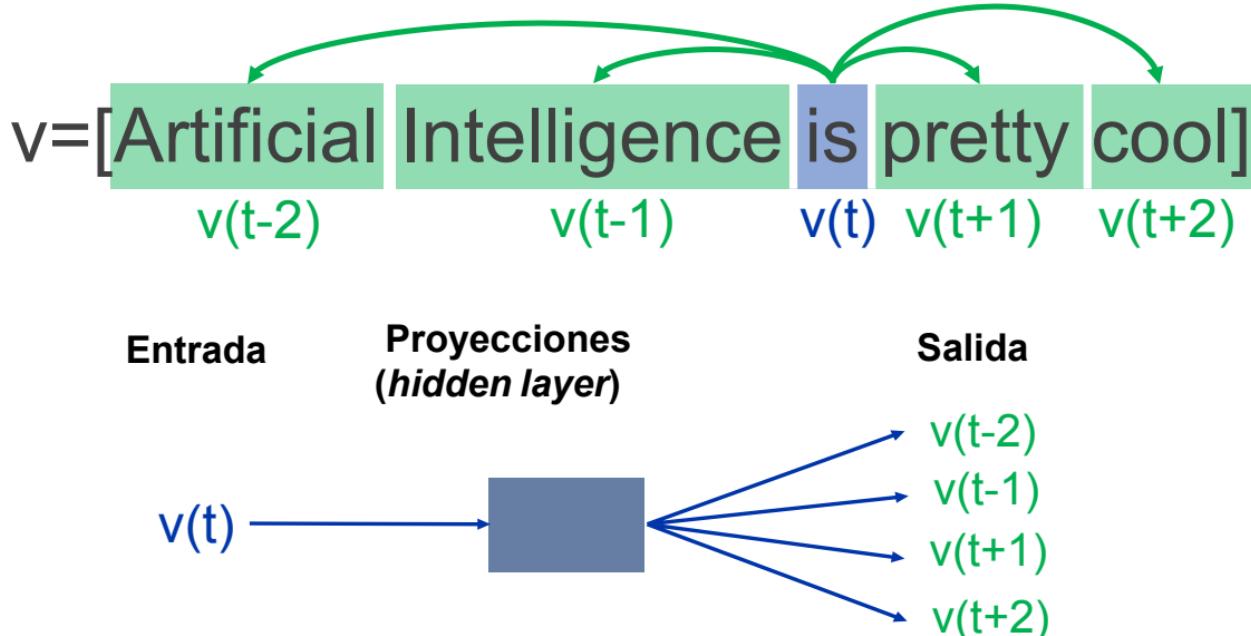
Tomáš Mikolov

Facebook member
Google ex-member

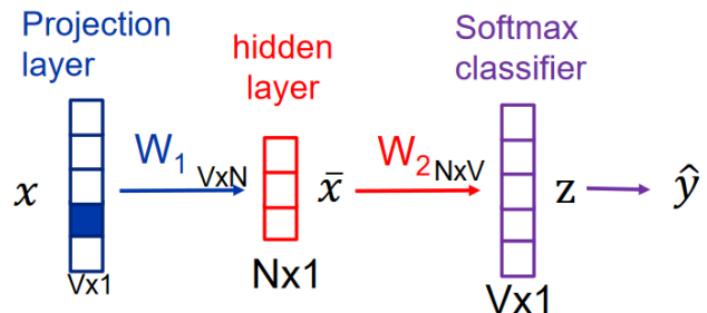
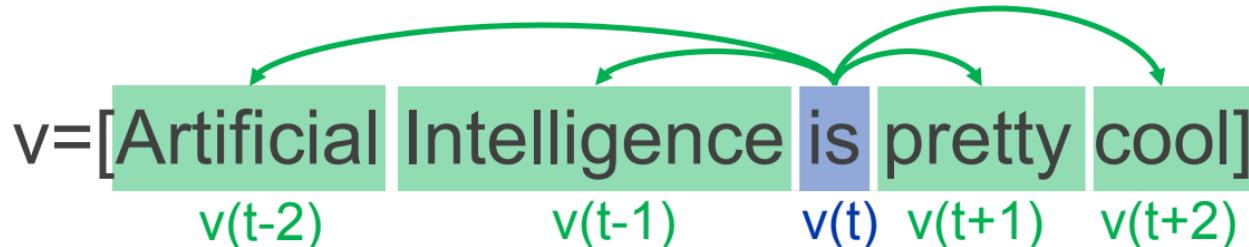
Mikolov et al. *Efficient Estimation of Word Representations in Vector Space*

Mikolov et al. *Distributed Representations of Words and Phrases and their Compositionality*

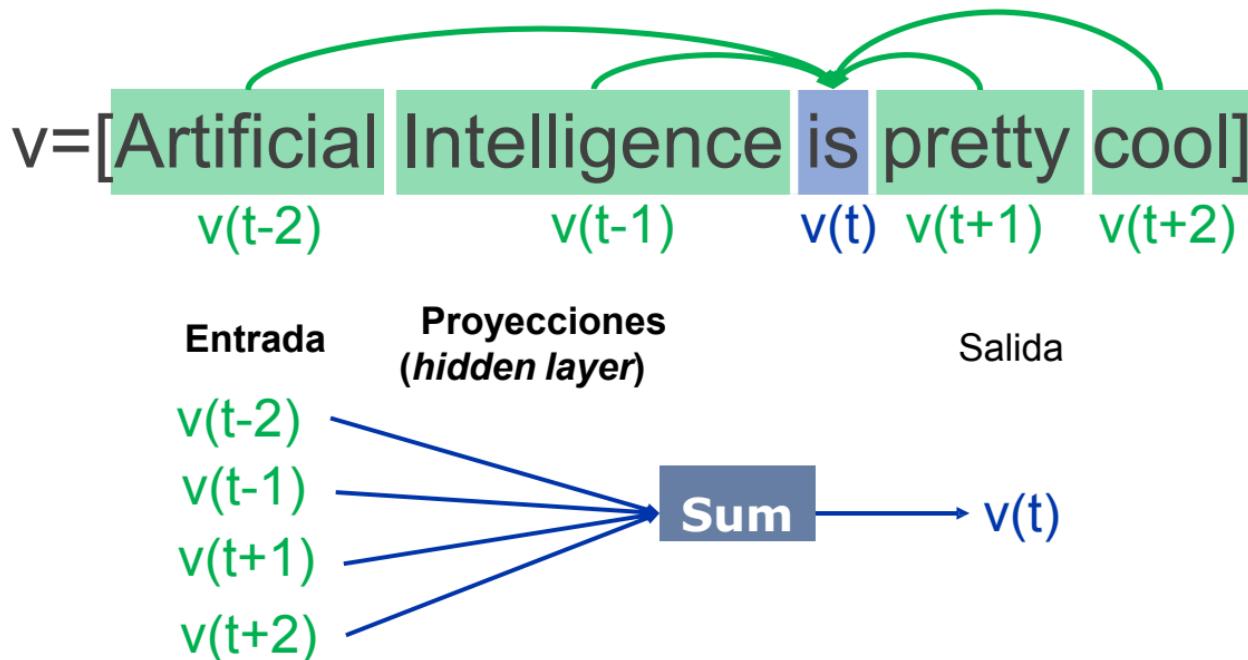
Modelo Skip-gram



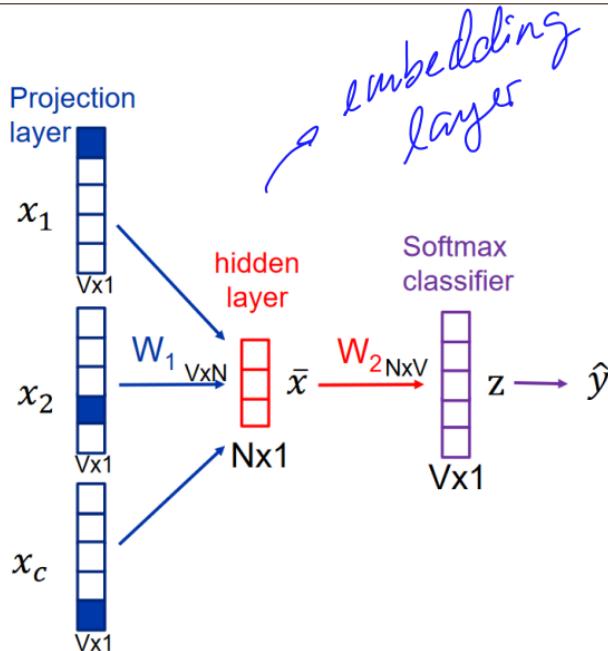
Modelo Skip-gram



Continuous Bag-of-Words



Continuous Bag-of-Words



$$\bar{x} = \frac{1}{2c} \sum_i^c W_1 \cdot x_i$$

$$z = W_2 \cdot \bar{x}$$

$$\hat{y}_i = softmax(z)$$

Cost function

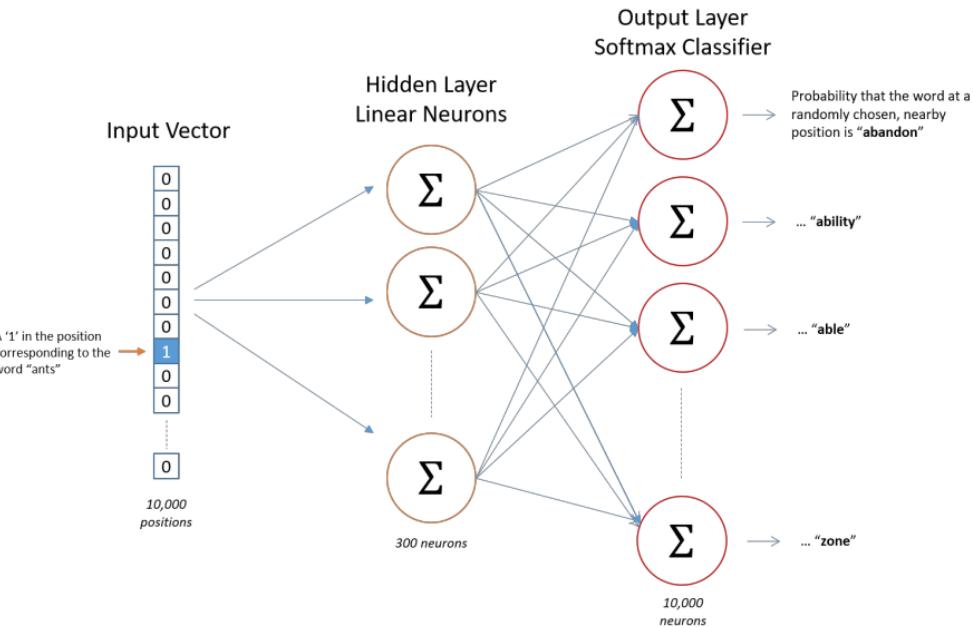
$$\text{argmin } H(\hat{y}, y) = -y_i \log(\hat{y}_i)$$

entropía cruzada

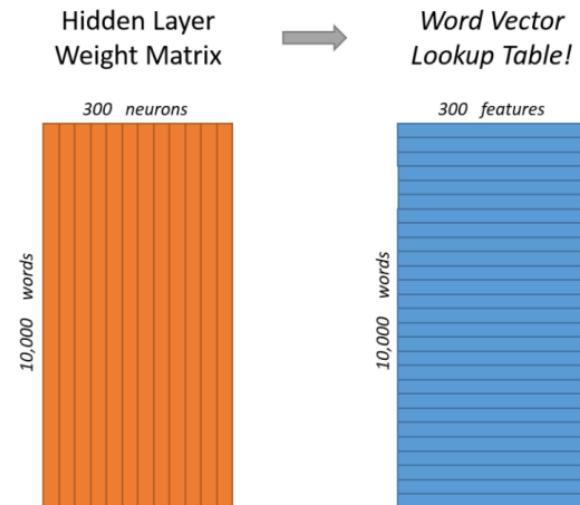
Word2Vec

Source Text	Training Samples
The quick brown fox jumps over the lazy dog. →	(the, quick) (the, brown)
The quick brown fox jumps over the lazy dog. →	(quick, the) (quick, brown) (quick, fox)
The quick brown fox jumps over the lazy dog. →	(brown, the) (brown, quick) (brown, fox) (brown, jumps)
The quick brown fox jumps over the lazy dog. →	(fox, quick) (fox, brown) (fox, jumps) (fox, over)

Word2Vec

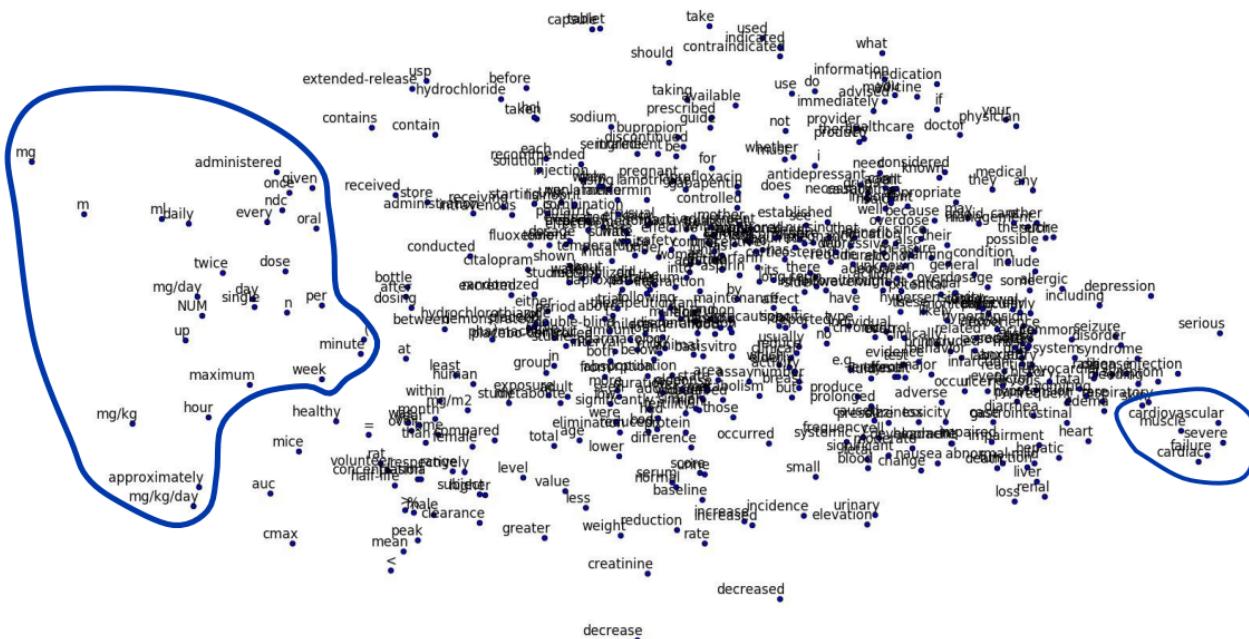


Word2Vec



$$\begin{bmatrix} 0 & 0 & 0 & \boxed{1} & 0 \end{bmatrix} \times \begin{bmatrix} 17 & 24 & 1 \\ 23 & 5 & 7 \\ 4 & 6 & 13 \\ \boxed{10} & 12 & \boxed{19} \\ 11 & 18 & 25 \end{bmatrix} = [10 \ 12 \ 19]$$

Word2Vec

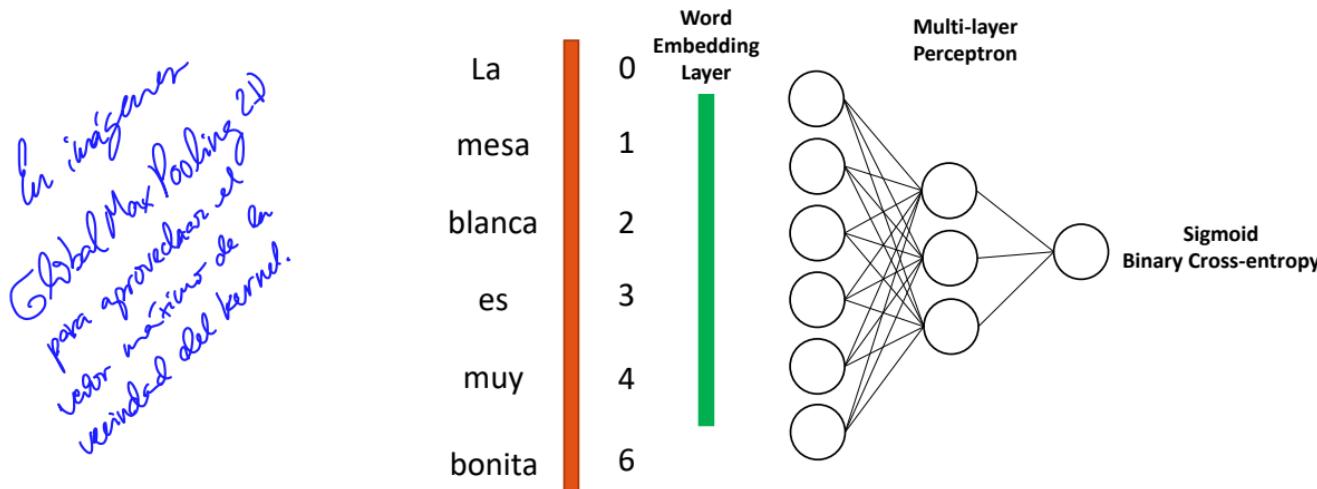


Contenidos

1. Procesamiento del Lenguaje Natural (NLP)
2. De texto a representaciones numéricas
3. Word2Vec
- 4. Redes neuronales recurrentes**
5. Introducción a *transformers*

MLP en análisis de texto

- ¿Es posible llevar a cabo tareas de **Natural Language Processing** empleando un **Perceptrón Multicapa**? ¿Es la arquitectura de red más adecuada? Imaginemos la tarea de clasificación binaria de secuencias de texto:

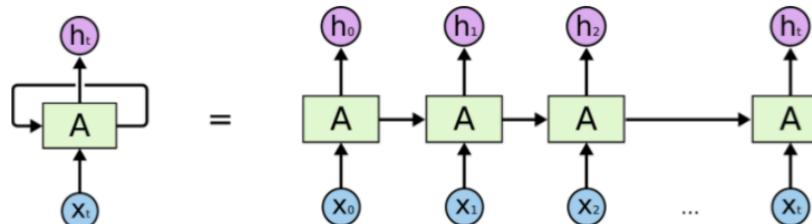


- Es posible emplear un **MLP** para tareas de NLP pero **no es la arquitectura más adecuada** ya que se pierde la “**secuencialidad gramatical**” de las palabras

Redes Neuronales Recurrentes

- Un **MLP** es **incapaz de retener dependencias entre muestras** y estamos ante un tipo de dato puramente secuencial
- Las **unidades básicas** que componen el **texto** (caracteres o palabras) son totalmente **dependientes** unas de otras, i.e. “secuencialidad gramatical”
- Las redes neuronales recurrentes o ***recurrent neural networks*** (RNN) surgen para superar esta limitación
- Necesidad de que la **información** de entrada persista entre muestras (Loop)

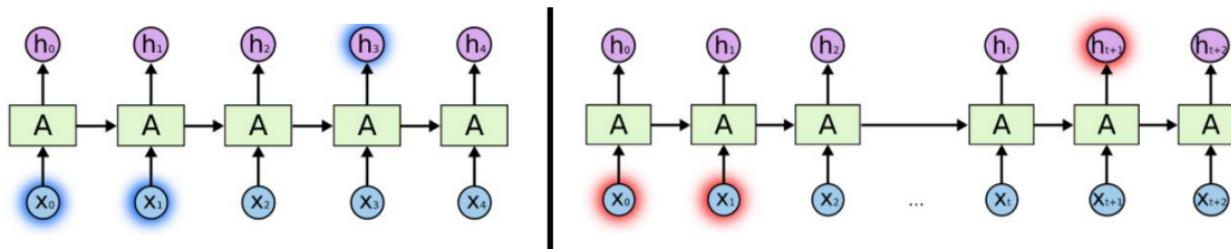
El tío de María nació en Francia. Recuerda ciertas palabras en _____



Redes Neuronales Recurrentes

- Podemos pensar en **una RNN** como **múltiples copias de la misma red**, cada una **pasándole el mensaje a la siguiente**
- No solo se emplean en texto, son la **mejor opción** siempre que se trabaje con **datos** en los que exista una **componente secuencial** (e.g. series temporales)
- ¿Pueden las RNN retener la información relevante de **muchos instantes atrás** en la secuencia? Conforme **aumenta el “gap”** las **RNN pierden** esta **propiedad**
Las nubes están en el *cielo*

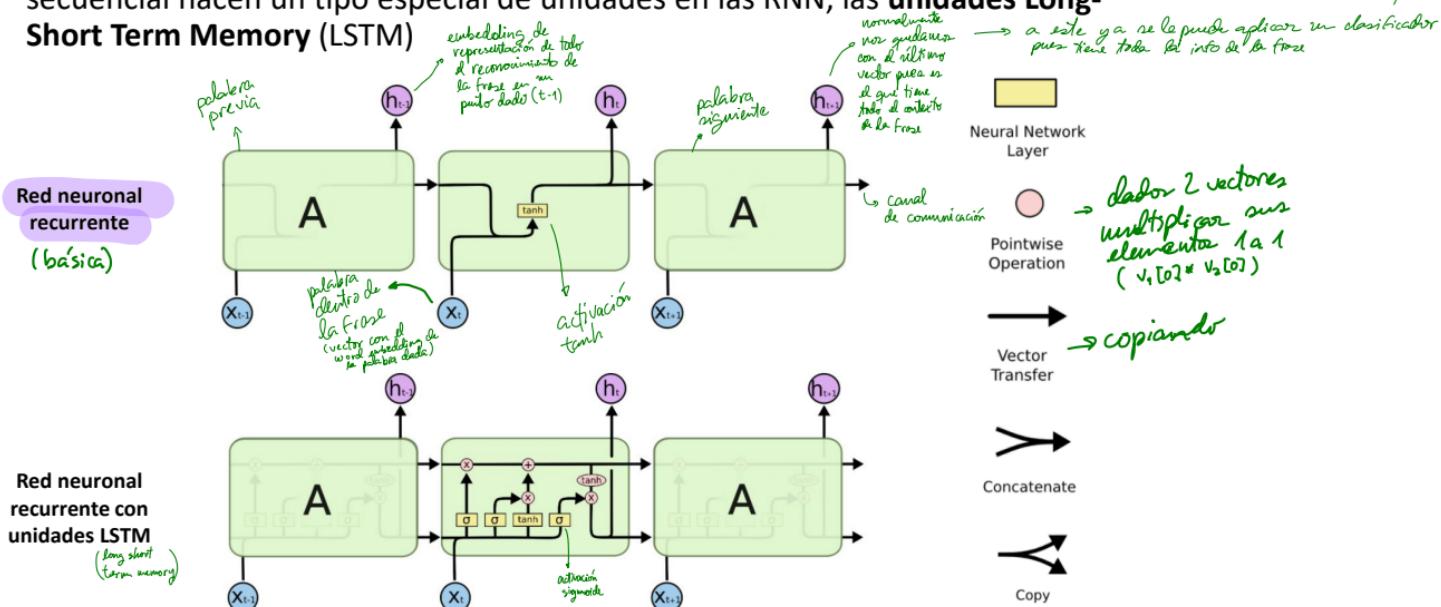
Yo crecí en Alemania... Puedo hablar *Alemán* de manera flúida



Unidades Long-Short Term Memory

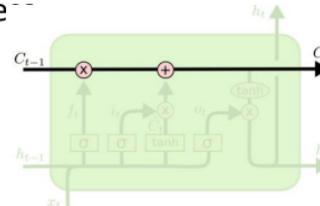
"Celdas GRU" → audio

- Con el objetivo de preservar el máximo número de instantes la información secuencial nacen un tipo especial de unidades en las RNN, las **unidades Long-Short Term Memory (LSTM)**

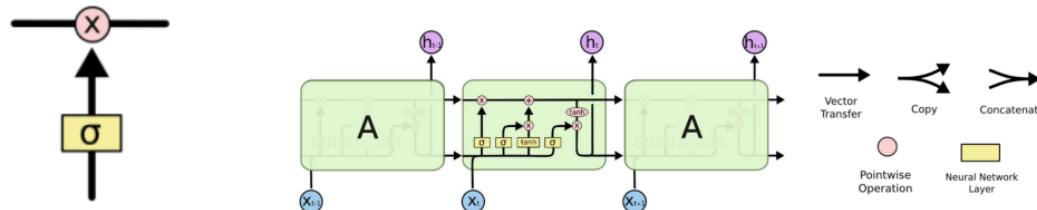


Unidades Long-Short Term Memory

- La clave de la **unidad LSTM** es la celda de estado (**cell state**), el camino recto superior por el que **fluyen los datos** a lo largo de los instantes **secuenciales** y van siendo alterados según intere



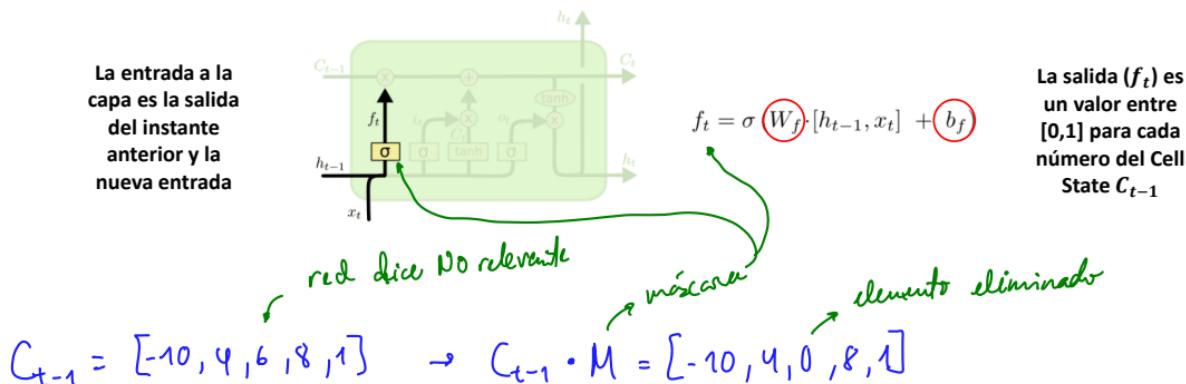
- El resto de unidad LSTM tiene la capacidad de **eliminar o añadir información sobre este camino** a partir de unas estructuras denominadas **compuertas** gestionadas por una capa sigmoide [0-1]



Unidades Long-Short Term Memory

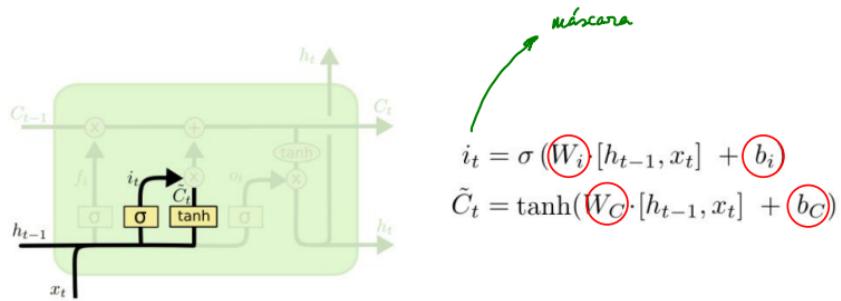
Nivelar la importancia de las palabras

- El primer paso dentro de la celda LSTM es decidir que **información del cell state quiero preservar para el nuevo estado**. Esto se lleva a cabo mediante la compuerta de olvido (**forget gate layer**)



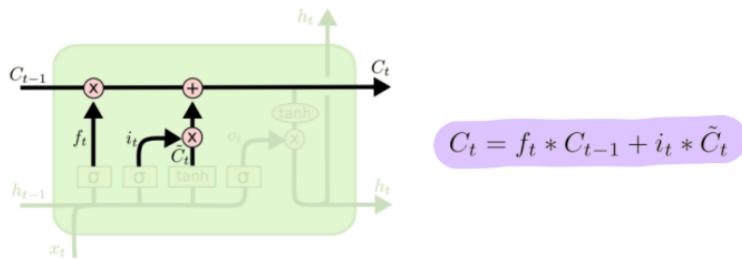
Unidades Long-Short Term Memory

- El siguiente paso consiste en decidir que nueva información va a hacer modificar el **cell state** del estado anterior. Este paso se divide en tres partes:
 - La compuerta de entrada (**input gate layer**) decide que valores actualizará i_t
 - Una capa activada con **tanh** crea un vector de nuevos candidatos \tilde{C}_t
 - Se **combinan** ambos para para **actualizar el cell state**



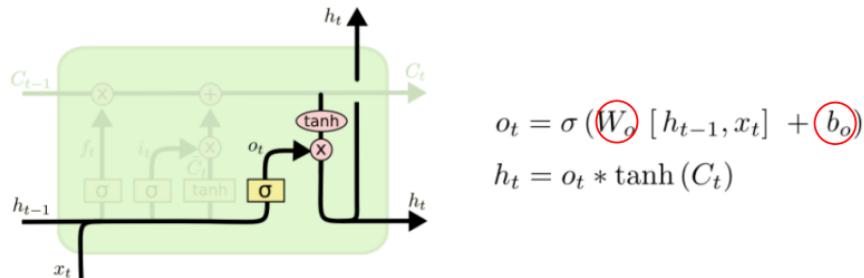
Unidades Long-Short Term Memory

- Ahora es momento de actualizar el **cell state** de la unidad anterior (C_{t-1}) según lo que marcan la **forget gate layer** (f_t), la **input gate layer** (i_t) y el vector \tilde{C}_t de nuevos candidatos, dando lugar al nuevo **cell state** C_t :
 - Se multiplica el estado viejo $C_{t-1} * f_t$ eliminando lo que no interesa del estado anterior
 - Se suma al C_{t-1} la nueva información ($i_t * \tilde{C}_t$) dada por los nuevos valores candidatos escalados por el peso que tendrán en el nuevo estado C_t



Unidades Long-Short Term Memory

- Finalmente, debemos decidir cuál va a ser la salida (h_t) del estado actual. Esta salida será nuestro nuevo **cell state** (C_t) pero en versión filtrada
- Concretamente, mediante una compuerta de salida o **output gate layer** se regulan las partes del **cell state** (C_t) que se quieren sacar como salida
- Mediante una **tanh** se normalizan los **valores de C_t** al rango [-1,1] y se lleva a cabo el **producto** por o_t que define la **salida final** de la celda (h_t)



LSTMs en Keras: ¿Qué necesito saber?

difícil de
parallelizar
(dependen de
etapas anteriores)

- Necesitaré entrenar una **capa de Embedding** según el **problema a resolver**. También se pueden **reentrenar embeddings** más generalistas (**word2Vec**) a partir del conocimiento sobre **grandes Corpus** (similar a **transfer learning** en imágenes)
- Instanciar **capas LSTM**, escogiendo el **número de unidades** (hiperparámetro)

Embedding class

```
tf.keras.layers.Embedding(  
    input_dim,  
    output_dim,  
    embeddings_initializer="uniform",  
    embeddings_regularizer=None,  
    activity_regularizer=None,  
    embeddings_constraint=None,  
    mask_zero=False,  
    input_length=None,  
    **kwargs  
)
```

LSTM class

```
tf.keras.layers.LSTM(  
    units,  
    activation="tanh",  
    recurrent_activation="sigmoid",  
    use_bias=True,  
    kernel_initializer="glorot_uniform",  
    recurrent_initializer="orthogonal",  
    bias_initializer="zeros",  
    unit_forget_bias=True,  
    kernel_regularizer=None,  
    recurrent_regularizer=None,  
    bias_regularizer=None,  
    activity_regularizer=None,  
    kernel_constraint=None,  
    recurrent_constraint=None,  
    bias_constraint=None,  
    dropout=0.0,  
    recurrent_dropout=0.0,  
    return_sequences=False,  
    return_state=False,  
    go_backwards=False,  
    stateful=False,  
    time_major=False,  
    unroll=False,  
    **kwargs  
)
```

usarán nuestro propio Embedding
para atornos más específicos, ej.
vocabulario más técnico

usamos
Word2Vec
para entornos
generalistas

devuelve los "h" intermedios.

Contenidos

1. Procesamiento del Lenguaje Natural (NLP)
2. De texto a representaciones numéricas
3. Word2Vec
4. Redes neuronales recurrentes
5. Introducción a *transformers*

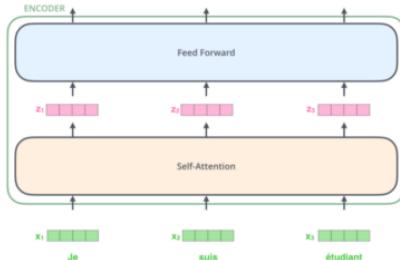
Introducción a transformers

"Vision transformers"

- A pesar de que las **LSTMs** han sido **muy exitosas** en la mayoría de **tareas** que envuelven el **NLP** poseen algunas desventajas
- Su **performance decrece** conforme **aumenta** la **longitud** de la **oración**. La probabilidad de mantener el contexto de una palabra lejana para predecir la actual, decrece exponencialmente con la distancia a ella
- Es **difícil** de **parallelizar** el **entrenamiento** de una LSTM ya que procesan secuencialmente las palabras y la salida de una celda la necesita la siguiente

Introducción a transformers

- Recientemente se presenta una nueva arquitectura capaz de procesar todas las palabras en paralelo y relacionarlas entre sí mediante un **módulo de atención**



- Se compone por **seis encoders y seis decoders**

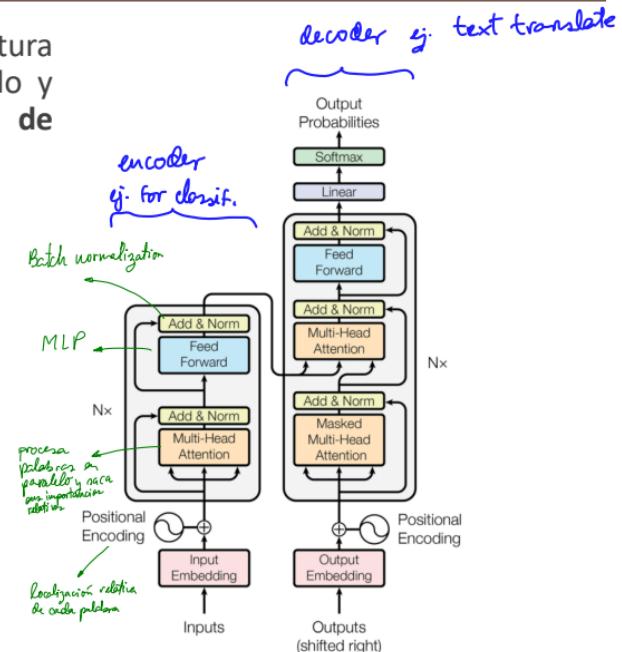
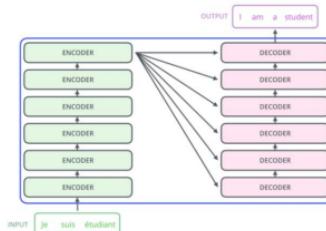


Figure 1: The Transformer - model architecture.

07MIAR – Redes Neuronales y Deep Learning

Close 23/10/2023

Min 01:43:58 \rightarrow returned_sequences = ~~True~~
~~False~~

07MIAR – Redes neuronales y deep learning



Universidad
Internacional
de Valencia

Generative Adversarial Networks para la
síntesis de imagen

01

Introducción

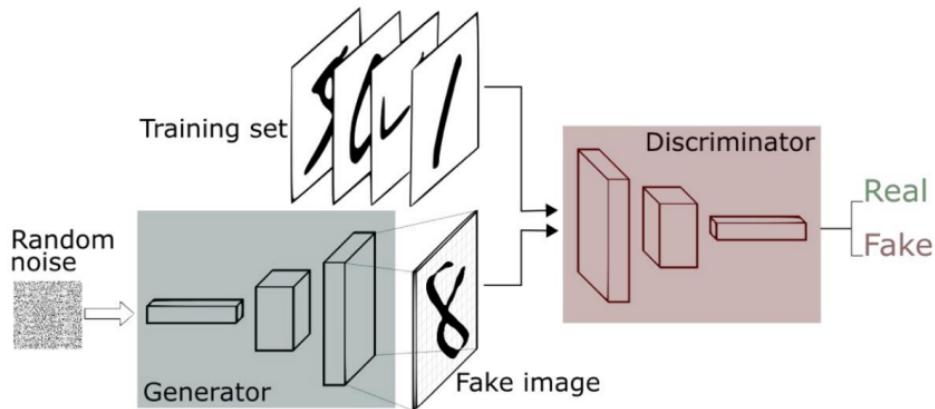
Generative Adversarial Network: Contexto de la arquitectura

- Una **Generative Neural Network** (GAN) es un tipo de arquitectura de red neuronal para llevar a cabo tareas de **modelado generativo** propuesta por **Goodfellow et al.** en **2014**.
- El modelado generativo implica el uso de un modelo que tiene como objetivo **generar nuevos ejemplos** que probablemente vengan de una distribución existente de muestras pero a su vez serán distintos de una población de instancias existente.
- Una GAN se entrena a partir de **dos modelos de arquitectura de red**. Un **generador** que aprende a generar nuevas muestras y un **discriminador** que aprende a identificar instancias generadas por el generador de instancias reales.
- El **objetivo** de una GAN es que el nuevo **contenido generado** a partir de ruido a la entrada (inicialización) sea tan **realista** (tras ciertas épocas) como para **confundir** al **discriminador**. Mapeo entre vector o matriz de elementos aleatorios a contenido realista.
- Tras el entrenamiento, el modelo generativo será capaz de **crear nuevas muestras sintéticas a demanda**.
- Muy utilizadas en el ámbito de la **visión por computador** pero no exclusivamente

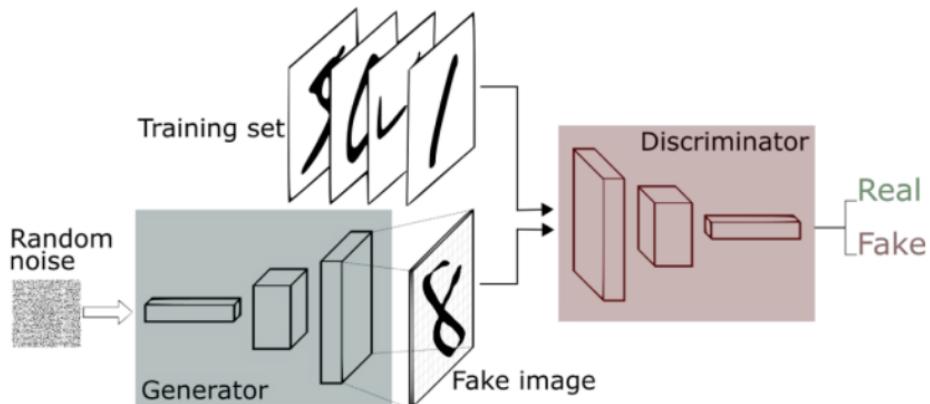
* Perceptual losses → ver si lo generado es real o no

Generative Adversarial Network: Funcionamiento

- El **generador** (“falsificador”) trata de generar contenido suficientemente realista como para **engañosar al discriminador** (“policía”)
- El objetivo del **discriminador** es **discernir** el contenido **real** del **falso** (creado por el generador)
- Se trata de un juego **min-max** en el que compiten entre ellos, i.e. **tarea adversarial**



Generative Adversarial Network: Funcionamiento



$$\mathcal{L}_{adv} = \min_G \max_D V(D, G) = \underbrace{\mathbb{E}_{x \sim p_{data}(x)} [\log D(x)]}_{\text{probabilidad de que } D(x) \text{ prediga que los datos reales son verdaderos}} + \underbrace{\mathbb{E}_{z \sim p_z(z)} [1 - \log D(G(z))]}_{\text{probabilidad de que } D(x) \text{ prediga que los datos generados, } x=G(z), \text{ no son verdaderos}}$$

Generative Adversarial Network: Entrenamiento

- El **entrenamiento** de una GAN se lleva a cabo en dos fases:

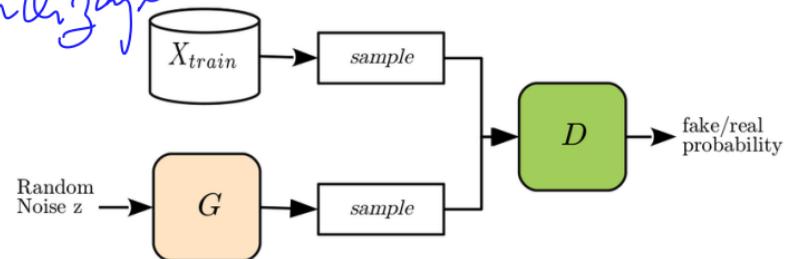
1. El **discriminador** recibe imágenes **reales** (etiqueta **1**) y **falsas** (etiqueta **0**) y debe aprender a **discernir** correctamente **ambos tipos de imagen**. Cuando **se equivoca**, se produce un **error** y sus **pesos** se **actualizan** convenientemente.

2. Con los **pesos del discriminador congelados**, el **generador** introduce imágenes **generadas (fake)** con **etiqueta "real" (1)** al **discriminador**. Cuando el **discriminador** predice que la **imagen es falsa**, al no coincidir la etiqueta con la predicción, se genera un **error** que permite **actualizar los pesos** del generador para que aprenda a sintetizar imágenes cada vez más realistas

* Ratio de aprendizaje
del discriminador

menor

↳ aprendizaje
más rápido



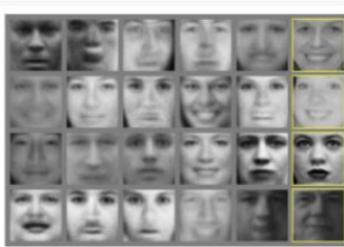
Implementaciones

- Keras: <https://github.com/eriklindernoren/Keras-GAN/blob/master/cyclegan/cyclegan.py>
- Pytorch: <https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix>

Aplicaciones GAN: Generación de imagen

7	3	9	3	9	9	9
1	1	0	6	0	0	
0	1	9	1	2	2	
6	3	2	0	8	8	

a)



b)

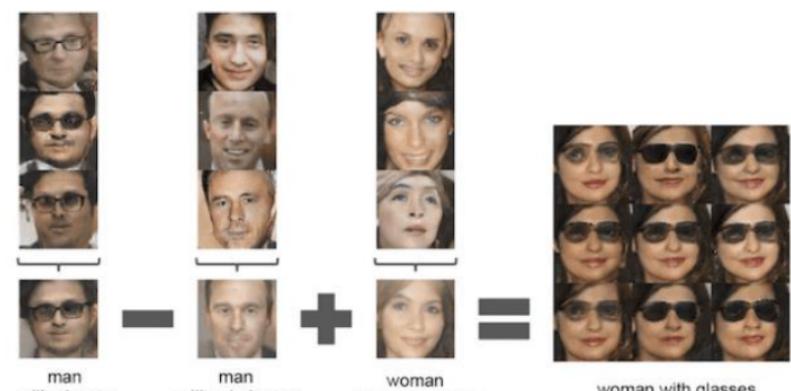


Examples of GANs used to Generate New Plausible Examples for Image Datasets. Taken from Generative Adversarial Nets, 2014.



Example of GAN-Generated Photographs of Bedrooms. Taken from Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks, 2015.

Aplicaciones GAN: Generación de imagen



Example of Vector Arithmetic for GAN-Generated Faces. Taken from Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks, 2015.



Examples of Photorealistic GAN-Generated Faces. Taken from Progressive Growing of GANs for Improved Quality, Stability, and Variation, 2017.

Aplicaciones GAN: Generación de imagen



Example of Photorealistic GAN-Generated Objects and Scenes Taken from Progressive Growing of GANs for Improved Quality, Stability, and Variation, 2017.



2014

2015

2016

2017

Example of the Progression in the Capabilities of GANs from 2014 to 2017. Taken from The Malicious Use of Artificial Intelligence: Forecasting, Prevention, and Mitigation, 2018.

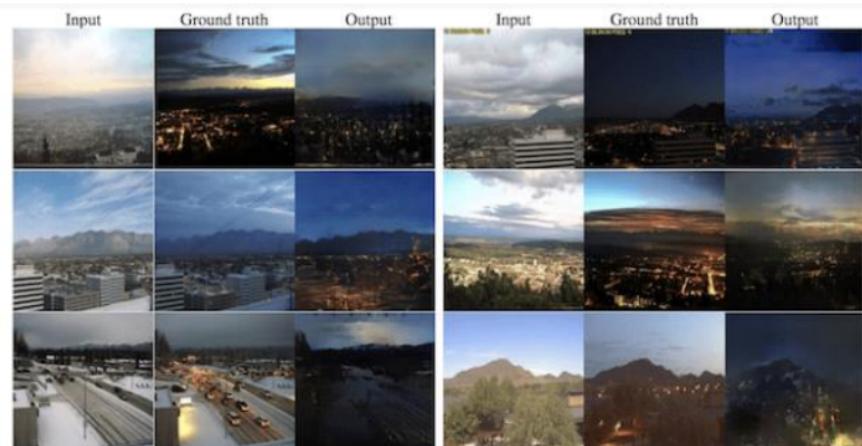
Aplicaciones GAN: Generación de imagen



Example of Realistic Synthetic Photographs Generated with BigGANTaken from Large Scale GAN Training for High Fidelity Natural Image Synthesis, 2018.

Aplicaciones GAN: Image-to-Image Traslation

Problema, depende mucho del emparejamiento en el dataset

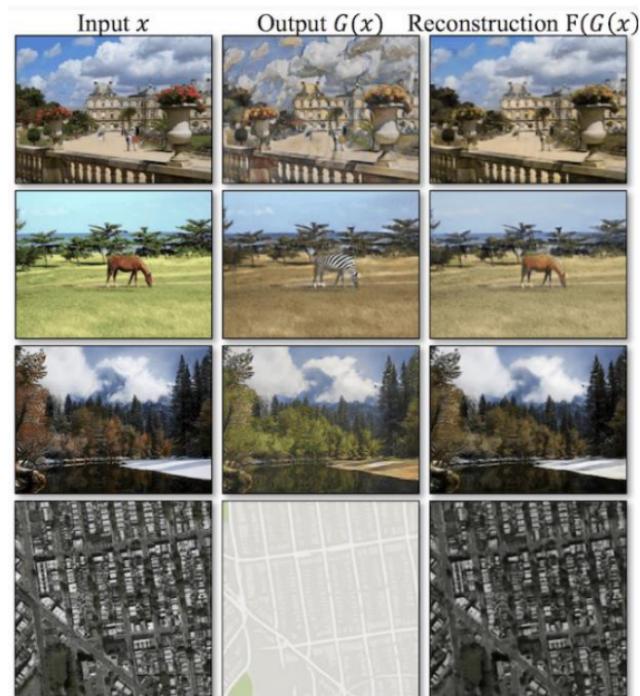


Example of Photographs of Daytime Cityscapes to Nighttime With pix2pix. Taken from Image-to-Image Translation with Conditional Adversarial Networks, 2016.



Example of Sketches to Color Photographs With pix2pix. Taken from Image-to-Image Translation with Conditional Adversarial Networks, 2016.

Aplicaciones GAN: Image-to-Image Traslation



Example of Four Image-to-Image Translations Performed With CycleGANTaken from Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks, 2017.

Ya no hacen falta imágenes emparejadas.



Example of Translation from Paintings to Photographs With CycleGAN.Taken from Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks, 2017.

Problema, un modelo específico para cada cambio de dominio/estilo. Si quiero un dominio nuevo de tipo de pintura, tengo que entrenar un nuevo modelo.

Aplicaciones GAN: Text-to-Image Translation

Ya no hay dos dominios emparejados oíno que ahora el condicionante es el Texto

The small bird has a red head with feathers that fade from red to gray from head to tail

Stage-I images



Stage-II images



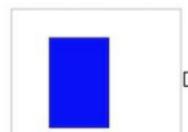
This bird is black with green and has a very short beak

Stage-I images



Stage-II images

Example of Textual Descriptions and GAN-Generated Photographs of Birds Taken from StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks, 2016.



This bird is completely black.



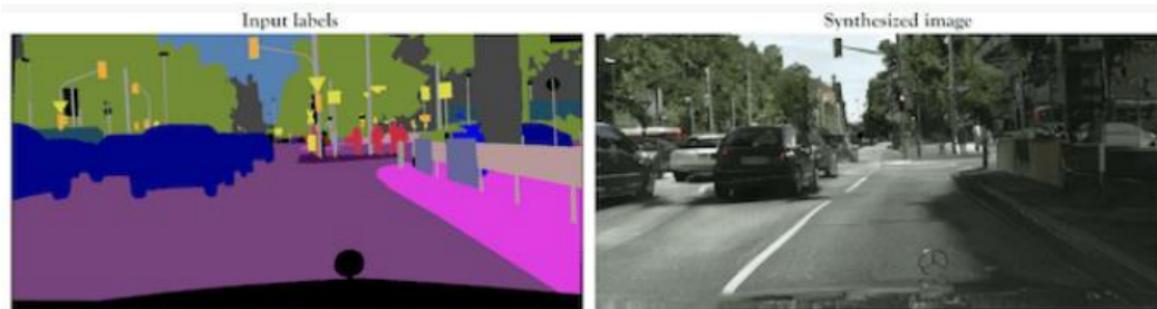
This bird is bright blue.



a man in an orange jacket, black pants and a black cap wearing sunglasses skiing

Example of Photos of Object Generated From Text and Position Hints With a GAN.Taken from Learning What and Where to Draw, 2016.

Aplicaciones GAN: Semantic-Image-to-Photo Translation



Example of Semantic Image and GAN-Generated Cityscape Photograph. Taken from High-Resolution Image Synthesis and Semantic Manipulation with Conditional GANs, 2017.

Aplicaciones GAN: Generating new human poses

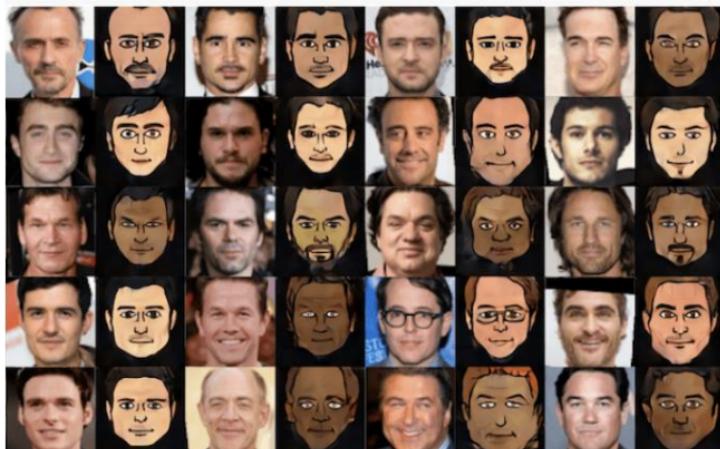


Example of GAN-based Face Frontal View Photo Generation Taken from Beyond Face Rotation: Global and Local Perception GAN for Photorealistic and Identity Preserving Frontal View Synthesis, 2017.



Example of GAN-Generated Photographs of Human Poses Taken from Pose Guided Person Image Generation, 2017.

Aplicaciones GAN: Photograph editing



Example of Celebrity Photographs and GAN-Generated Emojis. Taken from Unsupervised Cross-Domain Image Generation, 2016.

Real image

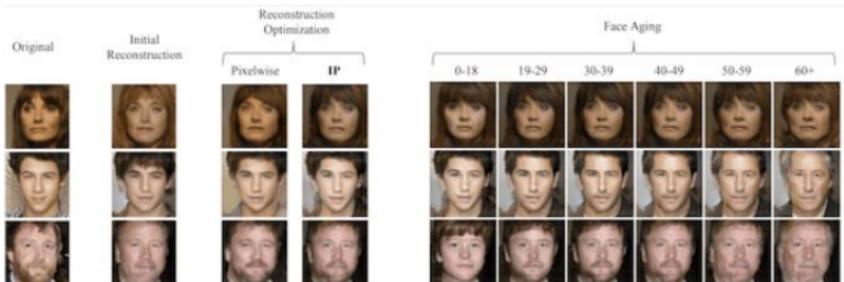


Reconstructed images



Blonde ↑ Bangs ↑ Smile ↑ Male ↑

Example of Face Photo Editing with IcGAN. Taken from Invertible Conditional GANs For Image Editing, 2016.



Example of Photographs of Faces Generated With a GAN With Different Apparent Ages. Taken from Face Aging With Conditional Generative Adversarial Networks, 2017.

Aplicaciones GAN: Photo blending & inpainting



(a)



(b)



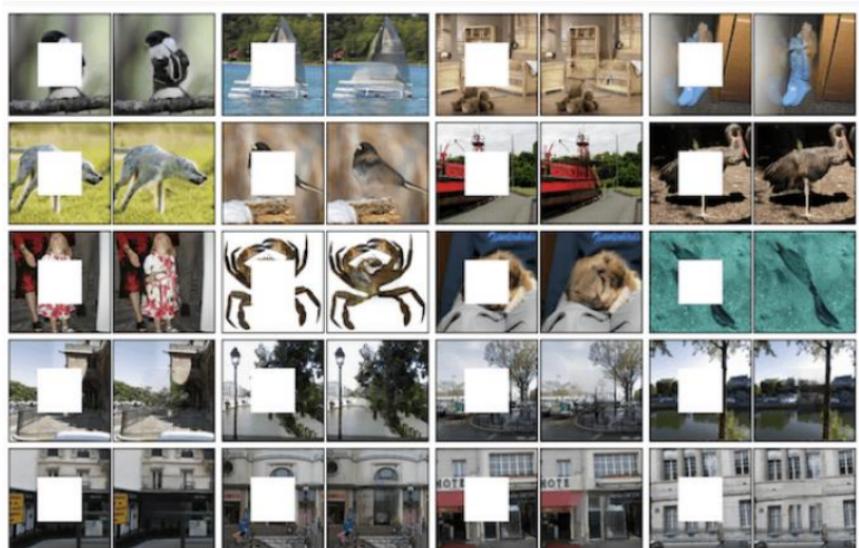
(c)



(d)



Example of GAN-based Photograph Blending.Taken from GP-GAN: Towards Realistic High-Resolution Image Blending, 2017.



Example of GAN-Generated Photograph Inpainting Using Context Encoders.Taken from Context Encoders: Feature Learning by Inpainting describe the use of GANs, specifically Context Encoders, 2016.

Aplicaciones GAN: SuperResolution



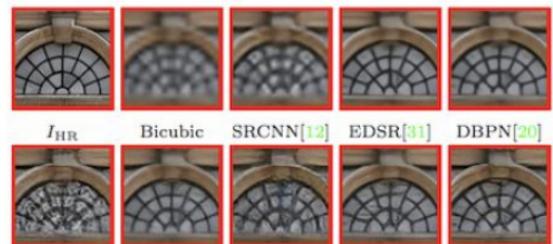
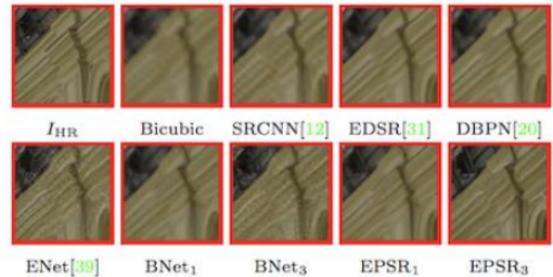
Example of GAN-Generated Images With Super Resolution. Taken from Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network, 2016.



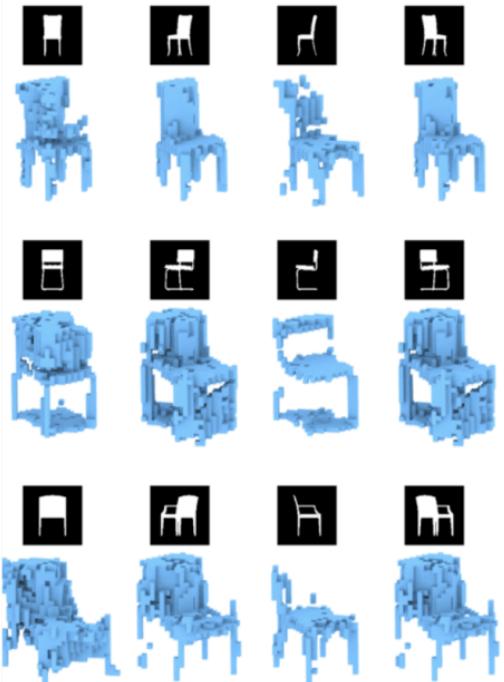
008 from Urban100



Example of High-Resolution GAN-Generated Photographs of Buildings. Taken from Analyzing Perception-Distortion Tradeoff using Enhanced Perceptual Super-resolution Network, 2018.



Aplicaciones GAN: Video Prediction & 3D Object Generation



Example of Video Frames Generated With a GAN. Taken from Generating Videos with Scene Dynamics, 2016.



02

Conditional GAN

Generative Adversarial Networks para la síntesis de imagen

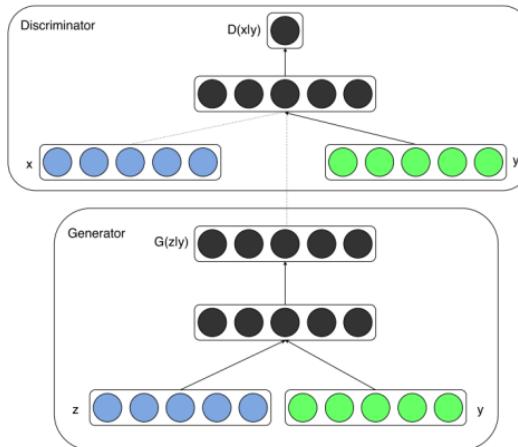
Introducción

* Proxy → condición de entrada

- Propuestas en **2014** por **Mirza et al.** las conditional GAN (**cGAN**) abordan la **problemática del escaso control del contenido de la versión vanilla**.
- Su objetivo es el de **dotar de estructura** al **espacio latente** de entrada al **generador**
- **Introducir información externa** (adicionalmente a las *inputs* que requiere la GAN básica) que “**guía**” el proceso de generación de imágenes:
 - Etiquetas de clase
 - Descripciones de texto
 - Mapas semánticos
 - Imágenes condicionales
 - Máscaras de objetos
 - Mapas de atención

Funcionamiento

- La información externa que “guía” el proceso de generación de imágenes se le proporciona tanto al generador como al discriminador.
 - Las tareas del generador y discriminador siguen siendo idénticas a las de la GAN convencional.
 - Tanto en el generador como en el discriminador, el espacio latente y la condición se combinan mediante una representación oculta (i.e. MLP).



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

Fig 2 shows some of the generated samples. Each row is conditioned on one label and each column is a different generated sample.

000000090000000000000000
111111111111111111111111
222222222222222222222222
323333333333333333333333
444444444444444444444444
555555555555555555555555
666666666666666666666666
777777777777777777777777
888888888888888888888888
989999999999999999999999

Implementaciones

- cGAN: <https://github.com/eriklindernoren/Keras-GAN/tree/master/cgan>

conditional
GAN

- DCGAN: <https://github.com/eriklindernoren/Keras-GAN/blob/master/dcgan>

→ deep convolutional
GAN

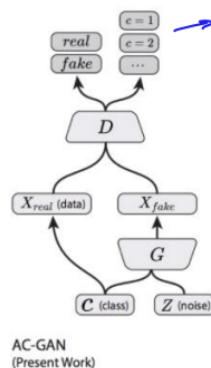
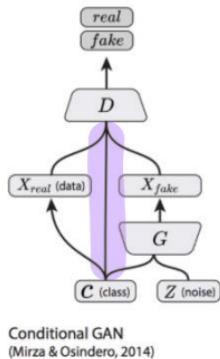
Variante: Auxiliary Classifier GAN

- Como hemos visto la cGAN proporciona al generador tanto un punto en el **espacio latente** como una **etiqueta de clase** con el objetivo de generar una imagen para dicha clase.
- El **discriminador** es alimentado por una **imagen** y una **etiqueta de clase** y su objetivo es el de clasificar si dicha imagen es real o falsa.
- La auxiliary classifier GAN (**AC-GAN**) solo le **proporciona** al **discriminador** una **imagen de entrada** pero no le dota de información de clase. De hecho es objetivo del discriminador estimarla (adicionalmente a la tarea real vs fake)

Ventaja:

- Mejora el generador, pues se le fuerza al generador a afinar con la clase (dependencia entre etiqueta y clase)

???



→ se le pide al discriminador la etiqueta de la clase

Problema:

- necesitas anotaciones en tu dataset o un buen text encoding

uso de
modelos de texto
preentrenados

03

CycleGAN

Generative Adversarial Networks para la síntesis de imagen

Tenemos un dataset de un dominio y otro de otro dominio y queremos un modelo que nos lleve de uno a otro

Introducción

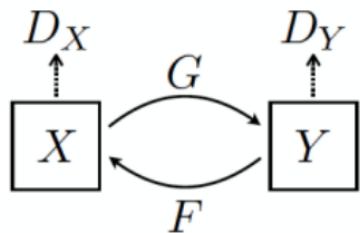
- La tarea de ***image-to-image translation*** tiene como objetivo el generar una **imagen sintética** (cierta modificación) a partir de una **imagen dada**, e.g. verano a invierno.
- El **entrenamiento** de un modelo para resolver este tipo de tarea requiere de un gran **dataset pareado de ejemplos**. En muchas **ocasiones** resulta **imposible**, e.g. style transfer en pintura.
- Con el objetivo de solventar esta limitación en **2017** nace **CycleGAN** (Jun-Yan Zhu). CycleGAN posibilita la **resolución** de **tareas** de traslación ***image-to-image sin necesidad*** de disponer muestras **pareadas** y en sentido bidireccional.
- Los modelos son **entrenados** de manera **no supervisada** a partir de una **gran colección** del **dominio fuente** y el **dominio destino** (sin relación entre ellas).



[...] we present a method that can learn to [capture] special characteristics of one image collection and figuring out how these characteristics could be translated into the other image collection, all in the absence of any paired training examples.

Funcionamiento

- Una **CycleGAN** requiere de un **entrenamiento simultáneo** de dos modelos **generadores** (G y F) y **dos** modelos **discriminadores** (D_X, D_Y).
- G transforma las imágenes **del dominio X al Y** mientras que F transforma imágenes **del dominio Y al X** .
- D_Y trata de **discernir** si la imagen de entrada es una **imagen real o ficticia del dominio Y** . De forma análoga trabaja D_X en el **dominio X** .



Entrenamiento generadores

- El proceso de entrenamiento de una CycleGAN viene definido por tres términos claramente diferenciados:

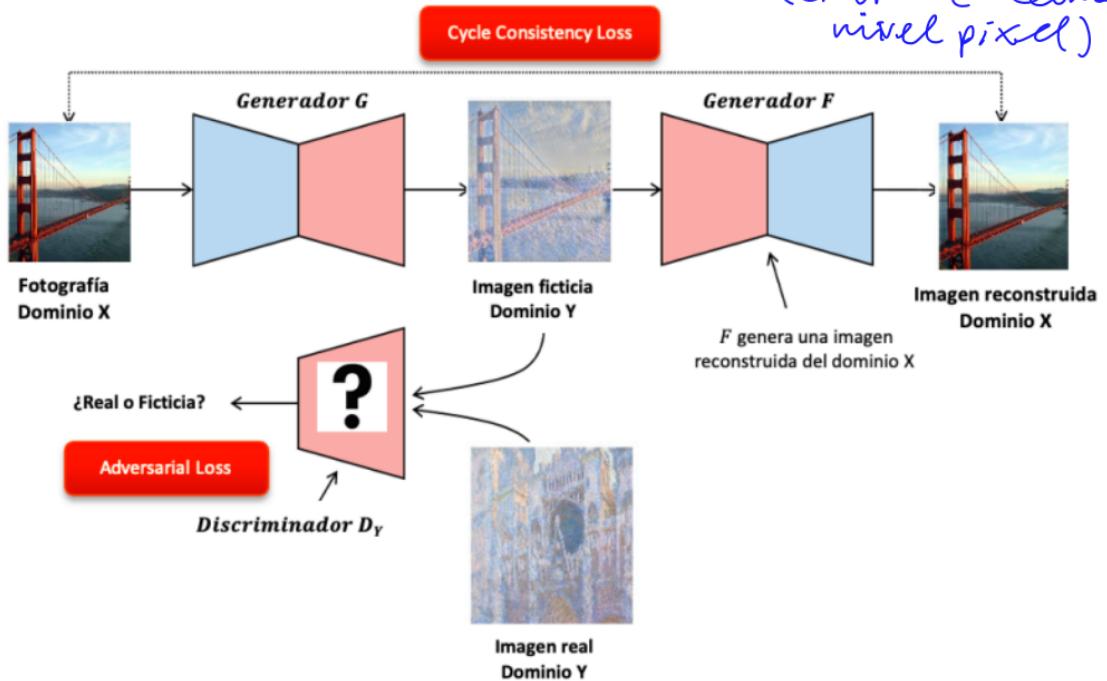
- Término adversarial (\mathcal{L}_{adv}). Mide la **capacidad** de los **generadores** de **crear imágenes** que se asemejan a los **dominios opuestos**. Se corresponde al término de ***binary cross-entropy*** de la versión vanilla.
- Pérdida de los generadores o ***cycle consistency loss*** (\mathcal{L}_{cc}). Surge del concepto de ciclo. Una **imagen** del **dominio X** pasa por G creando una **imagen** del **dominio Y** que le **entra a F** devolviendo dicha imagen al **dominio X**. Esta última imagen reconstruida debe ser muy similar a la imagen de partida. El grado de disimilitud (***cycle consistency loss***) es el criterio a minimizar, se suele emplear **MAE** o **L1**.
- Pérdida de identidad o ***identity loss*** (\mathcal{L}_{id}). Si a G le **entra** una **imagen** del **dominio Y**, G debería dejar **inalterada la imagen** puesto que ya pertenece al dominio **Y** que es capaz de generar G . De manera **análoga** ocurre con el generador F y el **dominio X**. Cuanto más modifique un generador una imagen de entrada de su dominio mayor será la pérdida registrada (mediante **MAE**).

Optimización
no convexa

α, λ, δ balancean la
importancia de cada uno de
los términos durante el
entrenamiento

$$\mathcal{L}_{G,F} = \underbrace{\alpha(\mathcal{L}_{adv}^G + \mathcal{L}_{adv}^F)}_{\text{generadores}} + \underbrace{\lambda(\mathcal{L}_{cc}^G + \mathcal{L}_{cc}^F)}_{\text{cycle consistency}} + \underbrace{\delta(\mathcal{L}_{id}^G + \mathcal{L}_{id}^F)}_{\text{identidad}}$$

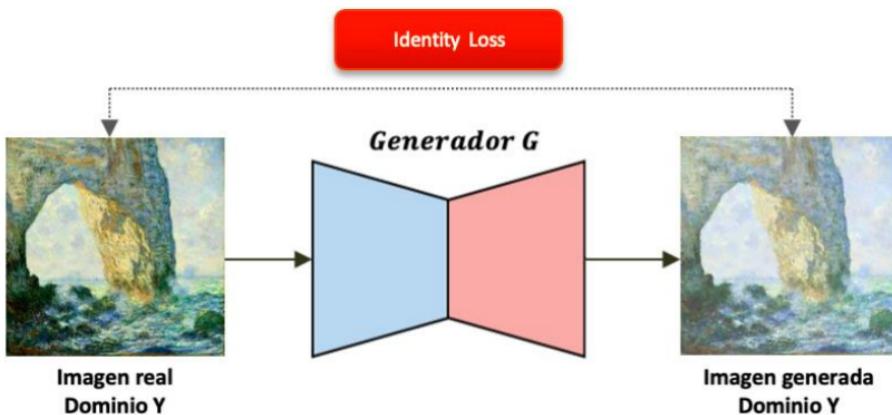
Cycle Consistency Loss



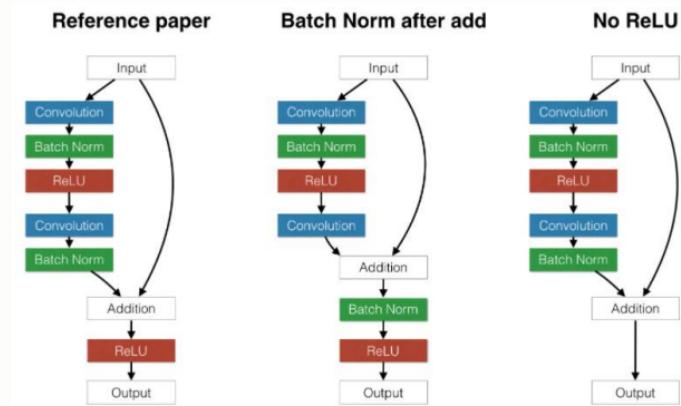
tipo Auto Encoder
(error de reconstrucción a
nivel pixel)

Identity Loss

consistencia a nivel de identidad

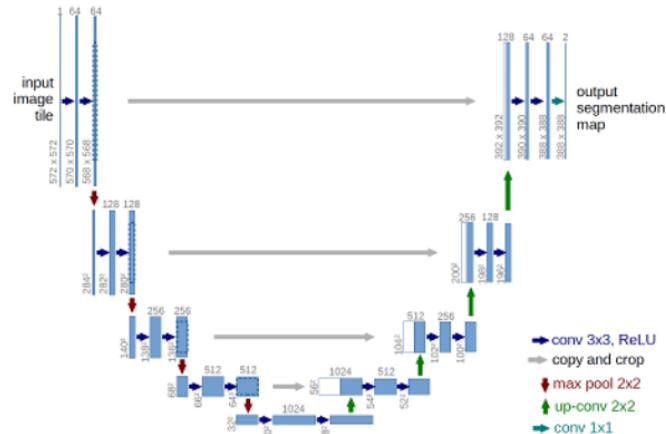


Arquitecturas generadores



<https://arxiv.org/pdf/1603.08155.pdf>

<https://github.com/facebookarchive/fb.resnet.torch>



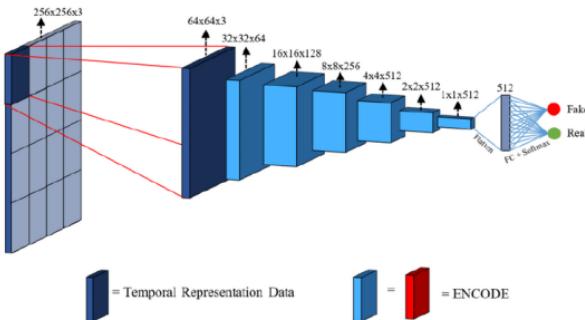
<https://github.com/zhixuhao/unet>

Entrenamiento discriminadores

Idea: el fondo ha de ser consistente
también

- Para entrenar D_Y debemos pasarle como input a la red discriminadora simultáneamente una **imagen** que realmente pertenezca al **dominio Y**, y una **imagen sintética** creada por el generador G . De manera **análoga** para entrenar F a partir de imágenes reales y *fake* del dominio X .
- La **pérdida** de los **discriminadores** vendrá dada por el cálculo de la **entropía cruzada binaria** obtenida a partir de las **imágenes reales** y **sintéticas**.
- Discriminador** basado en **PatchGAN**, evalúa (**real/fake**) el contenido **por parches** y mediante **promedio** emite el **veredicto global** de la **imagen**. Así el discriminador focaliza en el estilo y no en el contenido.

<https://arxiv.org/abs/1611.07004v3>



Implementaciones

- Keras: <https://github.com/eriklindernoren/Keras-GAN/blob/master/cyclegan/cyclegan.py>
- Pytorch: <https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix>



04

StyleGAN

Generative Adversarial Networks para la síntesis de imagen

Introducción

- StyleGAN presentada por Tero Karras et al. (NVIDIA) en 2018 focaliza sus esfuerzos en mejorar el generador de la vanilla GAN con el objetivo de sintetizar imágenes de alta calidad.
- Incluyen el uso de una red de mapeo del espacio latente en uno intermedio que controla el estilo en diferentes puntos del generador. Adicionalmente se introduce una fuente de ruido que ejerce de variación sobre cada punto del modelo generador.
- El resultado de estas mejoras se traduce en la generación de imagen sintética de alta calidad y fidelidad. Además, ofrece control sobre el estilo de las imágenes generadas a distintos nivel de detalle variando los vectores de estilo y ruido.



Our generator starts from a learned constant input and adjusts the “style” of the image at each convolution layer based on the latent code, therefore directly controlling the strength of image features at different scales

Funcionamiento

- El generador de **StyleGAN** no toma un punto del espacio latente si no que emplea **dos nuevas fuentes de aleatoriedad** para generar una imagen sintética: una **red de mapeo** independiente y **capas de ruido**.
- La salida de la red de mapeo es un **vector** que define el **estilo a integrar** en cada punto de la **red generadora** vía una nueva capa denominada Adaptive Instance Normalization (**AdaIN**). Esta nueva capa controla el estilo de las imágenes generadas.
- En cada punto del modelo generador se introduce una **variación estocástica** mediante la adición de **ruido** (sobre los mapas de activación) con el objetivo de dotar al modelo de **alta capacidad para interpretar el estilo**.
- Esta incorporación de **vectores de estilo y ruido** permite identificar el estilo y variación estocástica a **nivel local** para un nivel de detalle dado.

Diferencias

- pasamos la 'z' por un MLP que nos da un vector latente con muchos más dimensiones
 - ↳ tener más control del estilo a distintos niveles de la red mediante la capa AdaIN

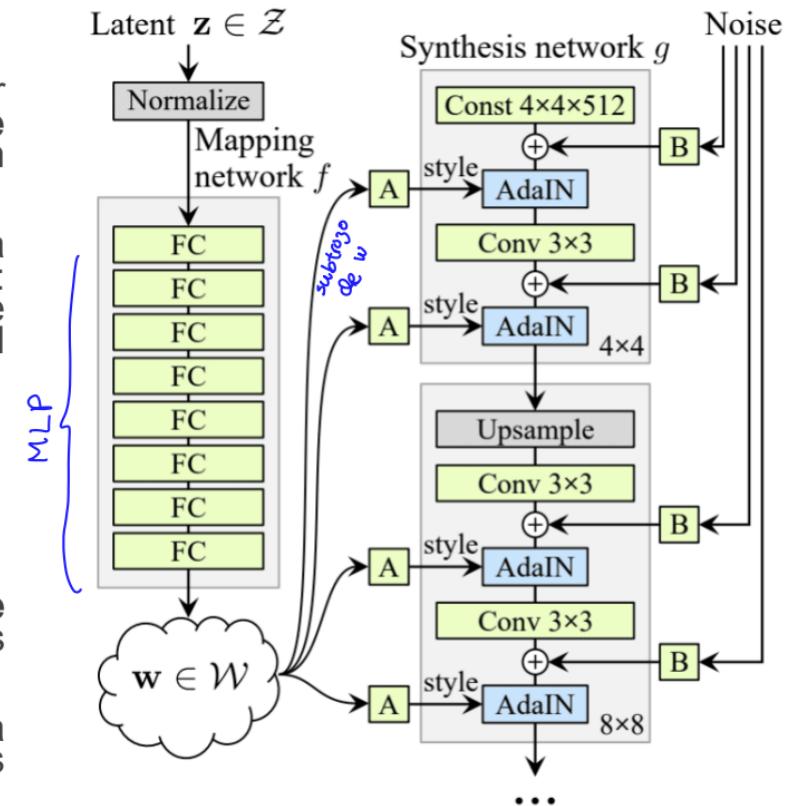
Arquitectura generador StyleGAN

- La red de mapeo es un **MLP** compuesto por **ocho capas densas**. Tanto el **espacio latente inicial** como el **intermedio** tienen dimensionalidad **512**.
- El **vector de estilo** se transforma y se incorpora a **cada bloque** del modelo generador vía **AdaIN**:
 a) **Estandarización** del mapa de activación de salida a una Gaussiana standard.
 b) Añadir el **style vector** como bias.

para que genere la mayor variedad posible

$$\text{AdaIN}(\mathbf{x}_i, \mathbf{y}) = \mathbf{y}_{s,i} \frac{\mathbf{x}_i - \mu(\mathbf{x}_i)}{\sigma(\mathbf{x}_i)} + \mathbf{y}_{b,i},$$

- ↑
- Se añade **ruido gaussiano** a cada **mapa de activación** (salida de los bloques convolucionales) **antes** de aplicar **AdaIN**.
 - Distintos muestreros de ruido** se generan para cada **bloque**. Este ruido introduce variaciones del estilo para un nivel de detalle dado.



Detalles de entrenamiento e implementación

- A diferencia de la **vanilla GAN**, el **entrenamiento** se lleva a cabo **de manera progresiva** (hasta la convergencia) con **resoluciones** de **imagen** potencia de dos que va **en aumento**. Se van expandiendo generador y discriminador.
- Este entrenamiento progresivo propicia el **cambio** a las ***upsampling layers*** (UpSampling2D + Conv2D) en vez de emplear las capas de convoluciones transpuestas (Conv2DTranspose).
- **Eliminar el espacio latente** como **entrada**. En su defecto se establece una **dimensionalidad** de entrada **constante** para cada resolución de imagen y es el espacio latente intermedio el que introduce la información.
- Realmente se generan **dos style vectors** de la red de mapeo y **se combinan** en el modelo generador a partir de un **split point** que **marca** que **vector de estilo** emplea **AdaIN**.

• Keras: <https://github.com/manicman1999/StyleGAN-Keras>

• PyTorch: https://github.com/tomguluson92/StyleGAN_PyTorch

- primero se entra el 1er bloque (primera resolución)
- se entra el discriminador y se congela ese bloque y se entra el siguiente bloque
- y así de forma progresiva para mejorar la convergencia y la resolución espacial.

w ~
cte.
para cada
una de las
resoluciones



viu

Universidad
Internacional
de Valencia

universidadviu.com

De:
 Planeta Formación y Universidades