

# EST-25134: Aprendizaje Estadístico

**Profesor:** Alfredo Garbuno Iñigo — Primavera, 2022 — Redes Neuronales.

**Objetivo:** En esta sección estudiaremos modelos basados en redes neuronales. El campo de *deep learning* es muy extenso. Así que esta sección del curso busca dar una versión global del tema y mencionar puntos importantes de dichos temas. Un tratamiento más extenso se puede encontrar en cursos especializados. Por ejemplo, el curso de Aprendizaje Profundo en el ITAM.

**Lectura recomendada:** Capítulo 10 de [1] y algunos pasajes de [4]. *Disclaimer:* Todas las figuras han sido tomadas de [1].

## 1. INTRODUCCIÓN

- La primera publicación relacionada es la de Rosenblatt [2].
- Las redes neuronales se volvieron populares en los 80s (con impacto limitado).
- A partir de ahí, se vio un auge en otros modelos predictivos.
- A partir de 2010, surge *Deep Learning* con resultados impresionantes.
- Mejora en poder de cómputo, *software* y datos.

	1970	1980	1990	2000	2010	2020
Data (samples)	$10^2$ (e.g. iris)	$10^3$	$10^4$ OCR	$10^{7-8}$ web	$10^{10}$ advertising	$10^{12}$ social nets
RAM	1kB	100kB	10MB	100MB	1GB	100GB
CPU	100kF (8080)	1MF (80186)	10MF (80486)	1GF (Intel Core)	100GF NVIDIA	>1PF (8xP3 Volta)

FIGURA 1. Imagen tomada de [4].

- El avance y adopción se debe a diversos investigadores en varios ámbitos de lo que rodea *Deep Learning*.
- Por ejemplo ([time-line de eventos importantes](#)):
  - Rumelhart et al. [3]: Redescubren la regla de la cadena (*backpropagation*).

- Yann LeCun, **Corinna Cortes** y Christopher Burges hacen pública la base de datos de **MNIST** (1998).
  - Un equipo de investigadores liderado por **Fei-Fei Li** publica la base de **imageNet** para concurso (2009).
  - Un equipo de investigadores en **Google Deepmind** logra vencer a jugadores profesionales en Go (2016).
- Otros perfiles de gente impresionante haciendo investigación en el área se puede encontrar [aquí](#).

## 2. PRECURSOR: PERCEPTRÓN

- Objetivo: resolver el problema de clasificación binaria por medio de separaciones lineales.
- Es decir, poder encontrar una  $\omega$  tal que

$$\langle \omega, x \rangle \geq 0, \quad \text{si } y = 1, \quad (1)$$

$$\langle \omega, x \rangle < 0, \quad \text{si } y = -1. \quad (2)$$

- Por lo tanto, lo que queremos es un predictor de la forma

$$\hat{y} = \text{signo}(\langle \omega, x \rangle). \quad (3)$$

### 2.1. Algoritmo:

- Empezamos con  $\omega^{(1)} = 0$ .
- Para cada iteración  $t = 1, \dots, T$ :
  - Buscamos un elemento mal clasificado:

$$y_i \cdot \langle \omega^{(t)}, x_i \rangle < 0, \quad (4)$$

dentro de observaciones.

- Actualizamos por medio de:

$$\omega^{(t+1)} = \omega^{(t)} + y_i \cdot x_i. \quad (5)$$

- Nos detenemos cuando todas las observaciones están bien clasificadas.

### 2.2. Observaciones

- El perceptrón funciona cuando las clases son separables.
- El perceptrón no convergerá cuando las clases no son separables.
- La pérdida asociada a este algoritmo considera términos individuales

$$\max[0, -y\langle \omega, x \rangle]. \quad (6)$$

- La idea que utilizaremos: el perceptrón emite una señal si  $\langle \omega, x \rangle \geq 0$ , ¿qué tal que utilizamos un conjunto de perceptrones y los combinamos linealmente?

### 3. RED NEURONAL DE UNA CAPA

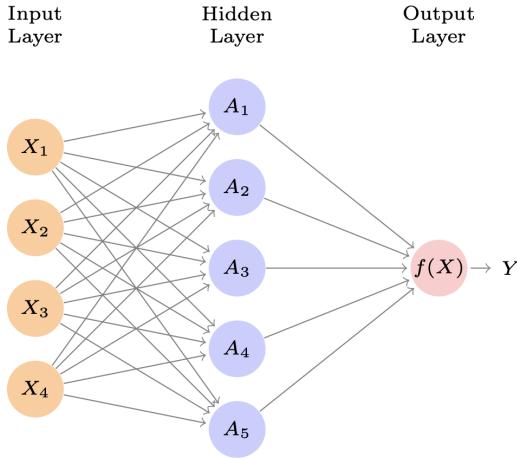
Consideramos el modelo predictivo de la forma

$$f(X) = \beta_0 + \sum_{k=1}^K \beta_k h_k(X), \quad (7)$$

donde los términos  $h_k$  son **transformaciones no-lineales de combinaciones lineales de los atributos**. Es decir,

$$h_k(X) = g \left( \omega_{k0} + \sum_{j=1}^p \omega_{kj} X_j \right), \quad (8)$$

donde  $g(\cdot)$  es una transformación no-lineal de  $\mathbb{R}$  a  $\mathbb{R}$ .



#### 3.1. Detalles

- En la figura anterior tenemos que  $A_k = h_k(X) = g \left( \omega_0 + \sum_{j=1}^p \omega_{kj} X_j \right)$ .
- La función  $g(\cdot)$  se denomina **función de activación**.
- Las opciones mas populares son: **ReLU** o **sigmoide**.
- Si no utilizamos funciones de activación no-lineales, entonces el modelo seguiría siendo lineal.
- La salida de las funciones de activación son interpretadas como atributos .
- El modelo se entrena (en regresión) minimizando

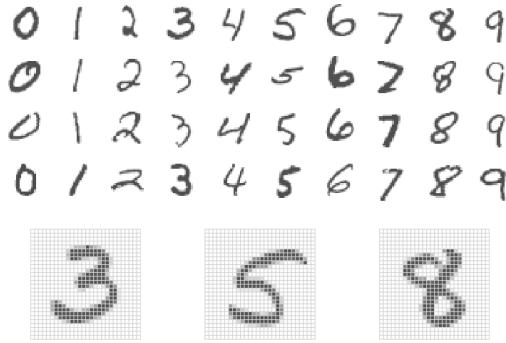
$$\sum_{i=1}^n (y_i - f(x_i))^2. \quad (9)$$

- La solución aprende representaciones de los atributos que pueden servir para predecir.

#### 3.2. Ejemplo: clasificación multi-clase (MNIST)

Tenemos imágenes de  $28 \times 28$  pixeles en escala de grises. Tenemos  $60K$  datos de entrenamiento y  $10K$  datos de validación. Podemos pensar que cada imagen es un vector de 784 dimensiones. Las etiquetas son los dígitos del 0 al 9.

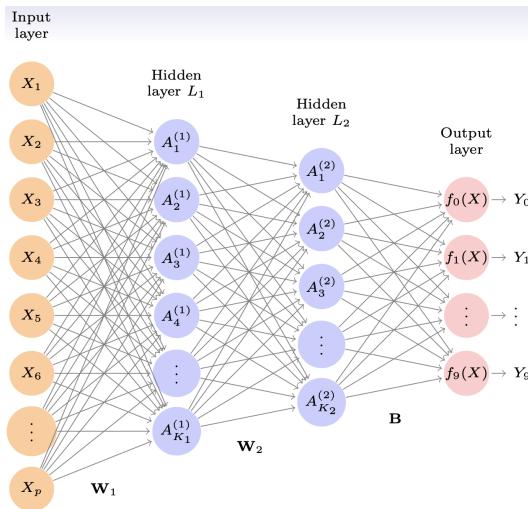
**Objetivo:** Predecir la clase de la imagen basada en los valores de los pixeles.



### 3.3. El modelo

Se utiliza una red neuronal de dos capas. La estructura (arquitectura) es 256 unidades en la primera capa, 128 unidades en la capa intermedia y 10 unidades de salida.

#### 3.3.1. ¿Cuántos parámetros tiene este modelo?



### 3.4. La capa de salida

- Denotemos por

$$Z_m = \beta_{m0} + \sum_{\ell=1}^{K_2} \beta_{m\ell} A_\ell^{(2)}, \quad (10)$$

las  $m$  combinaciones lineales de las unidades que salen de la segunda capa.

- Denotamos por  $m$  es el número de unidades en la capa de salida.
- Para obtener *probabilidades* usamos la función **softmax** como función de activación en la última capa

$$f_m(X) = \mathbb{P}(Y = m|X) = \frac{\exp(Z_m)}{\sum_{\ell=0}^9 \exp(Z_\ell)}, \quad (11)$$

donde entrenamos el modelo minimizando

$$-\sum_{i=1}^n \sum_{m=0}^9 y_{im} \log(f_m(x_i)), \quad (12)$$

la cual llamamos **entropía cruzada**.

- $y_{im}$  tomará el valor de 1 en la clase que a la que pertenezca la observación  $i$ ésima. Todos los demás valores son 0 (**one-hot encoding**).

### 3.5. Detalles

- La pérdida de **entropía relativa** corresponde a un modelo multinomial de  $K$  clases:

$$\mathbb{P}(y|x) = \prod_{k=1}^K p_k(x)^{y_k}, \quad (13)$$

- Considerando la función de **softmax** entonces la función de pérdida (individual) queda

$$\ell(y, \hat{y}) = - \left[ \sum_{k=1}^K y_k \log \left( \frac{\exp(z_k)}{\sum_{j=1}^K \exp(z_j)} \right) \right]. \quad (14)$$

- La cual se puede simplificar

$$\ell(y, \hat{y}) = - \sum_{k=1}^K y_k z_k + \log \left( \sum_{k=1}^K \exp(z_k) \right). \quad (15)$$

- Lo cual es muy útil para métodos iterativos de optimización

$$\frac{\partial \ell}{\partial z_j} = \text{softmax}(z_j) - y_j. \quad (16)$$

### 3.6. Regularización

- Con tantos parámetros en los modelos resulta indispensable *regularizar* nuestro problema de entrenamiento.
- Consideremos el problema de clasificar imágenes de perros y gatos.
- Las imágenes son tomadas con nuestras cámaras (12Mp) lo cual se traduce en  $12 \times 10^6$  píxeles.
- Un modelo de una capa con mil unidades tiene entonces (apróx.)  $36 \times 10^9$  parámetros.
- Según una búsqueda en Google (datos de 2019), tenemos una población de 471M perros y 373M gatos.
  - Esto es (apróx)  $0.844 \times 10^9$  imágenes.
- Necesitaríamos  $36/.844 \approx 42.65$  más datos para tener una relación 1 a 1 de parámetros con datos.

Los métodos usuales de regularización son (mas adelante veremos detalles de esto):

1. Regularización en coeficientes matrices  $W_k$ .
2. Regularización *dropout*.
  - Resultados en MNIST son:
  - A la fecha, los mejores resultados reportan un error de generalización de menos del 0.5 %
  - El error de personas en este conjunto de datos es de 0.2 %,

Method	Test Error
Neural Network + Ridge Regularization	2.3%
Neural Network + Dropout Regularization	1.8%
Multinomial Logistic Regression	7.2%
Linear Discriminant Analysis	12.7%

FIGURA 2. Resultados de generalización obtenidos por distintos modelos en el conjunto de datos de MNIST, fuente: [1].

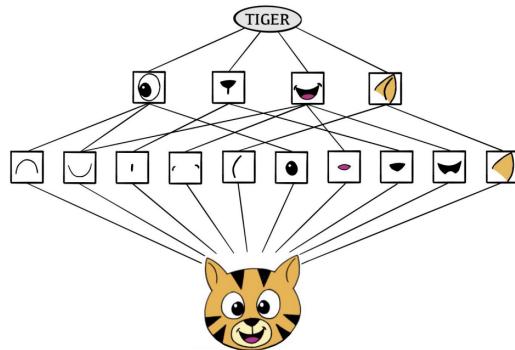
#### 4. MODELOS CONVOLUCIONALES



- Historia de éxito para problemas de visión por computadora.
- La base de ImageNet está compuesta de mas de 20,000 categorías con imágenes de  $256 \times 256$
- El conjunto de datos son 14M anotadas utilizando Amazon Mechanical Turk.
- Cada imagen la podemos pensar como una matriz con tres canales de color RGB.
- Por lo tanto una imagen es un arreglo tridimensional de  $256 \times 256 \times 3$  de números de 8 bits.



### 4.1. Ideas generales



- Las CNN construyen una imagen de manera jerárquica.
- Primero se **extraen** características globales (por medio de filtros) y se combinan para crear **atributos** específicos de la imagen.
- La extracción y combinación de atributos se logran con capas de **convolución** y **pooling**,

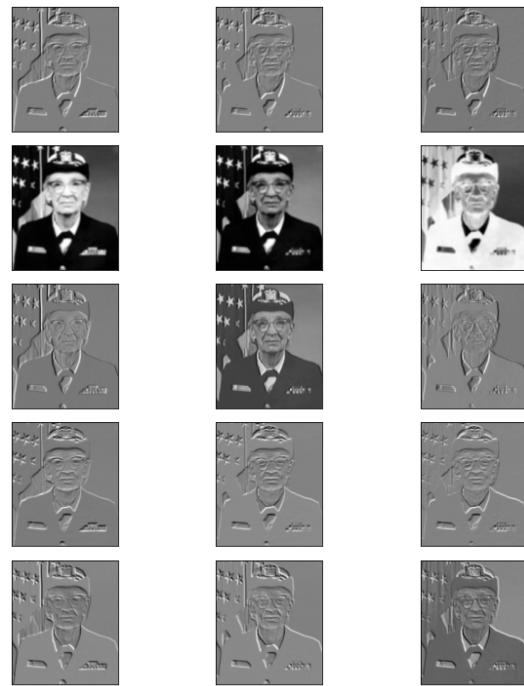
### 4.2. Capas de convolución

$$\text{Input Image} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \\ j & k & l \end{bmatrix} \quad \text{Convolution Filter} = \begin{bmatrix} \alpha & \beta \\ \gamma & \delta \end{bmatrix}.$$

$$\text{Convolved Image} = \begin{bmatrix} a\alpha + b\beta + d\gamma + e\delta & b\alpha + c\beta + e\gamma + f\delta \\ d\alpha + e\beta + g\gamma + h\delta & e\alpha + f\beta + h\gamma + i\delta \\ g\alpha + h\beta + j\gamma + k\delta & h\alpha + i\beta + k\gamma + l\delta \end{bmatrix}$$

- El filtro es una imagen que representa un patrón en la imagen original.
- El filtro se arrastra a lo largo de la imagen y se registran los *scores*.
- Los *scores* no son mas que un producto interior.
- Los filtros se aprenden durante el ajuste del modelo.

### 4.3. Ejemplo de un filtro



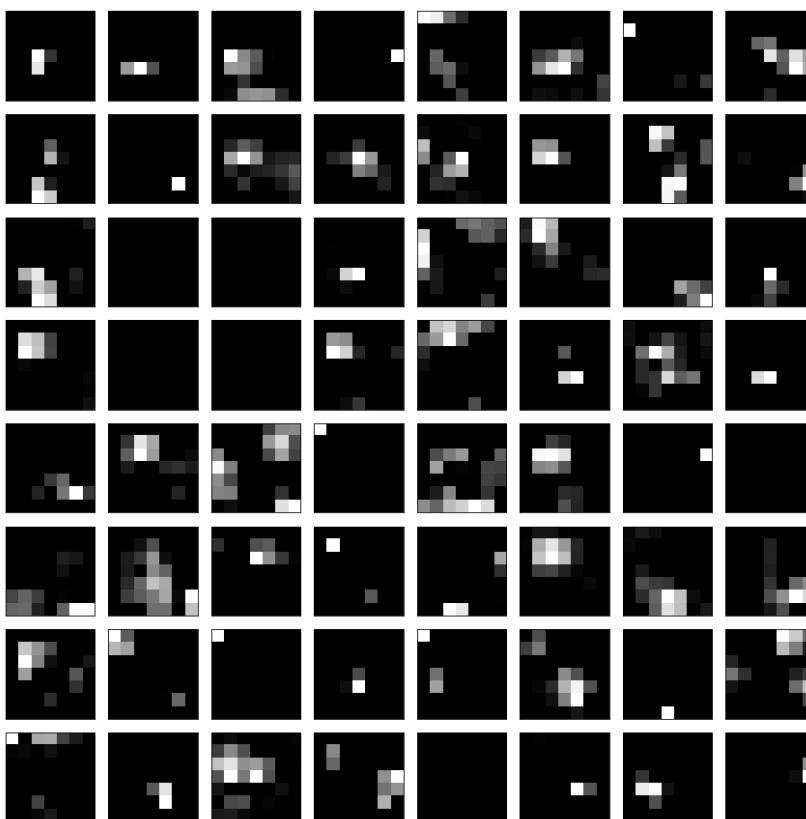
- Los filtros resaltan características particulares de la imagen.
- Con los resultados de los filtros se crean *nuevas* características en capas intermedias.



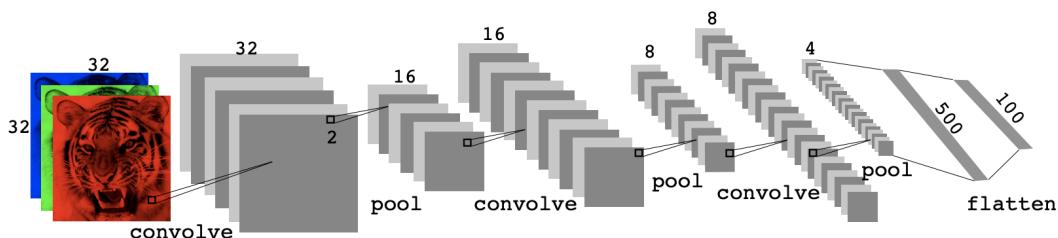
#### 4.4. Capas de resumen (pooling)

$$\text{Max pool} \quad \begin{bmatrix} 1 & 2 & 5 & 3 \\ 3 & 0 & 1 & 2 \\ 2 & 1 & 3 & 4 \\ 1 & 1 & 2 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} 3 & 5 \\ 2 & 4 \end{bmatrix}$$

- Las capas de resumen se utilizan para generar representaciones de menor resolución.
- Ayuda a enfocar y mejorar la identificación de atributos.
- Permiten que el clasificador sea **localmente invariante**.
- Reduce la dimensión de los atributos.



#### 4.5. Arquitectura de una CNN



- El enfoque es sobre los bloques: capas de convolución o resúmenes.
- Los filtros usualmente son pequeños  $3 \times 3$ .
- Cada filtro crea un nuevo canal en la capa.

- Por un lado reducimos el tamaño de las representaciones. Por otro, aumentamos el número de filtros.

#### 4.6. Aplicación de CNN

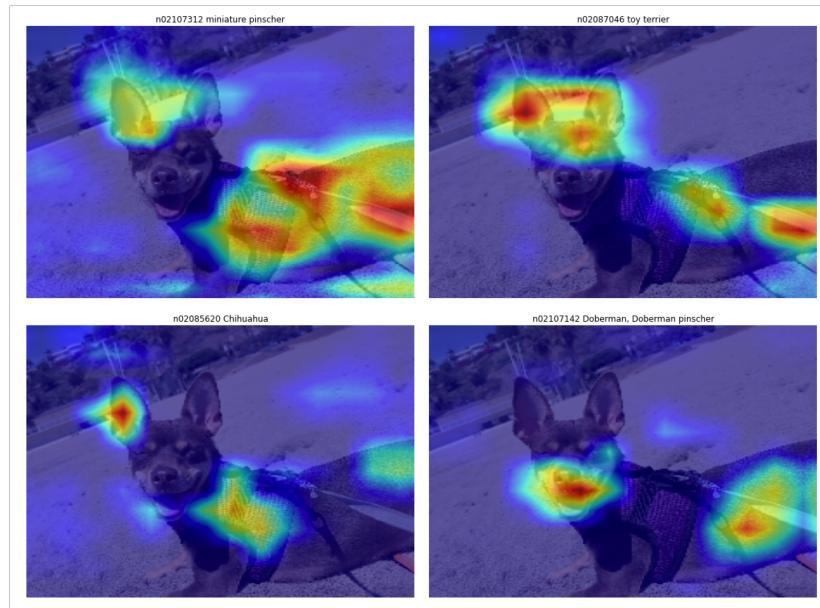
- El procedimiento de entrenamiento es computacionalmente caro.
- En la práctica no se tienen los recursos para poder entrenarlos (FANG si).
- Se ha volcado a una estrategia de compartir los pesos de los modelos entrenados.
- Amazon y otros proveedores están capitalizando en estas ideas (Amazon Rekognition).

### Available models

Model	Size (MB)	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth	Time (ms) per inference step (CPU)	Time (ms) per inference step (GPU)
Xception	88	79.0%	94.5%	22.9M	81	109.4	8.1
VGG16	528	71.3%	90.1%	138.4M	16	69.5	4.2
VGG19	549	71.3%	90.0%	143.7M	19	84.8	4.4
ResNet50	98	74.9%	92.1%	25.6M	107	58.2	4.6
ResNet50V2	98	76.0%	93.0%	25.6M	103	45.6	4.4
ResNet101	171	76.4%	92.8%	44.7M	209	89.6	5.2
ResNet101V2	171	77.2%	93.8%	44.7M	205	72.7	5.4
ResNet152	232	76.6%	93.1%	60.4M	311	127.4	6.5
ResNet152V2	232	78.0%	94.2%	60.4M	307	107.5	6.6
InceptionV3	92	77.9%	93.7%	23.9M	189	42.2	6.9

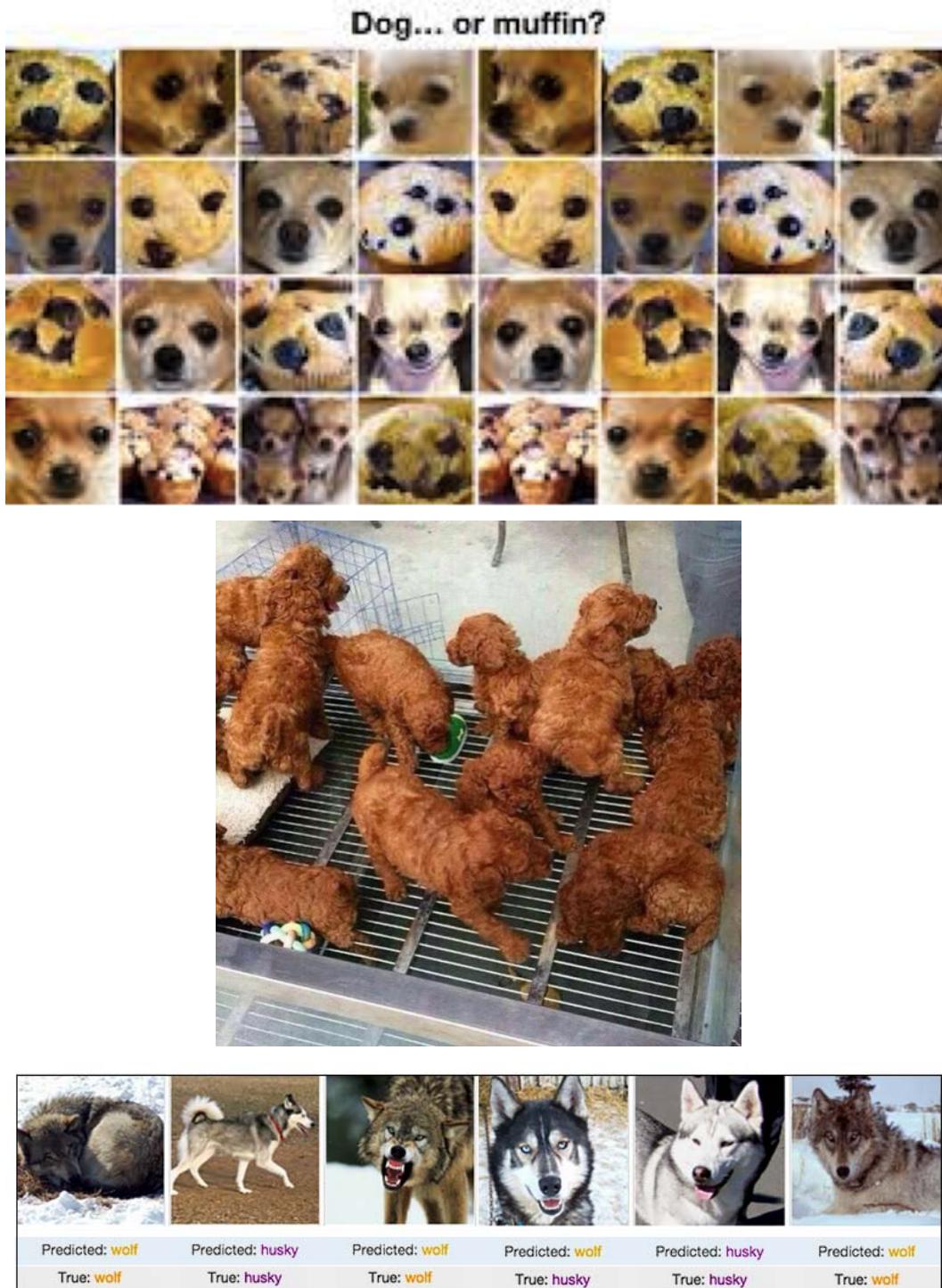
FIGURA 3. Modelos disponibles en Keras.

Nos permite clasificar imágenes propias en estos modelos.



#### 4.7. Comentarios

- Los filtros que se aprenden de manera automática corresponden con filtros que los expertos en visión por computadora han utilizado.
- No hay garantía de un aprendizaje en sentido generalizable (inteligencia artificial).



## 5. MODELOS RECURRENTES

- Otro caso de éxito de las redes neuronales.
- Aplicación en finanzas y análisis de texto.

### 5.1. Motivación

- Supongamos que tenemos las reseñas de algún producto.
- Por ejemplo, la base de IMBD que tiene los *ratings* de películas.
- El objetivo es poder identificar el sentimiento de la reseña: **positiva o negativa**.
- Esto es un problema de **clasificación binaria**.

This has to be one of the worst films of the 1990s. When my friends & I were watching this film (being the target audience it was aimed at) we just sat & watched the first half an hour with our jaws touching the floor at how bad it really was. The rest of the time, everyone else in the theater just started talking to each other, leaving or generally crying into their popcorn ...

### 5.2. Vectorización de texto

- Pregunta clave: ¿cómo representamos los textos en vectores de atributos?
- Se escoge un diccionario de palabras, por ejemplo las  $10K$  más frecuentes.
- Cada reseña se representa con un vector de longitud  $10K$  donde tiene un 1 en las posiciones de las palabras que ocurren.
- Tenemos una matriz de diseño de tamaño  $n \times 10K$ .
- James et al. [1] comparan una regresión logística (con regularización) y una red neuronal con dos capas.
- Pudimos haber escogido *vectorizar* identificando los pares de palabras más frecuentes (**bigramas**).

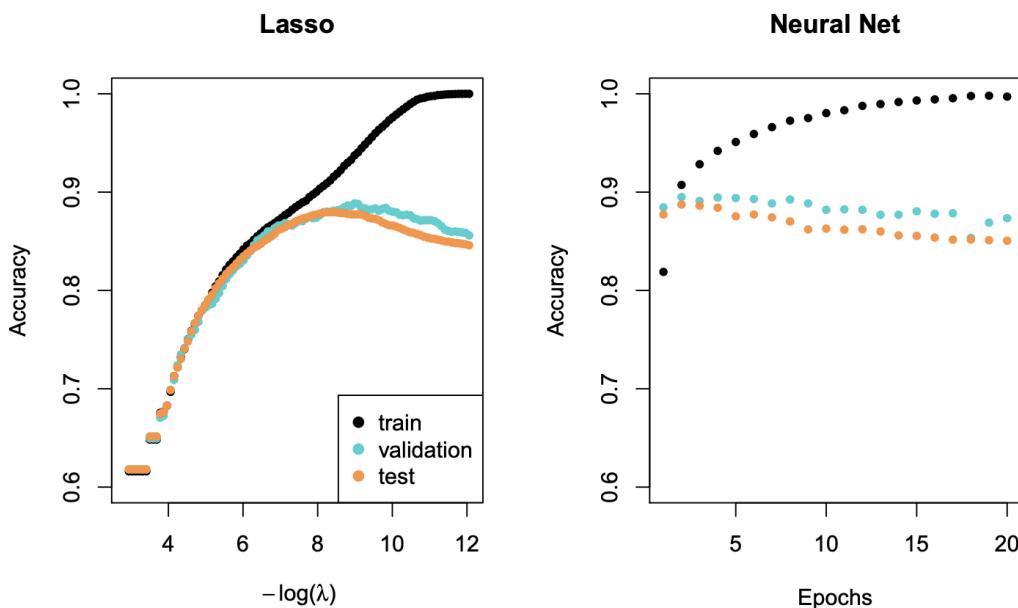


FIGURA 4. Resultado del experimento.

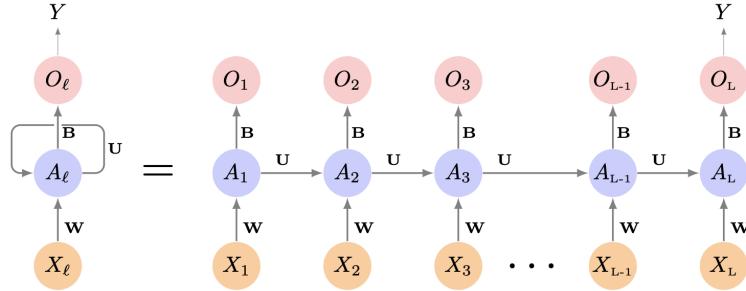
### 5.3. Redes neuronal recurrentes (RNN)

- A veces tenemos datos que vienen en secuencias.
- La arquitectura de las RNN toman en cuenta la naturaleza secuencial de los datos.
- Los atributos de *una observación* es una **secuencia** de vectores:

$$X = \{X_1, X_2, \dots, X_L\}. \quad (17)$$

- Las respuestas *pueden* ser las usuales en problemas de clasificación.

### 5.4. Arquitectura



- Una capa intermedia tiene una secuencia de vectores  $A_\ell$ , los cuales reciben como entradas los atributos en dicho paso  $X_\ell$  y el estado latente de  $A_{\ell-1}$ . Cada paso produce resultados intermedios  $O_\ell$ .
- Se utilizan los mismos pesos  $W$ ,  $U$  y  $B$  en cada paso de la secuencia (recurrente).
- Pensemos en  $A_\ell$  como una secuencia de un modelo que cambia su respuesta después de actualizar con cada elemento  $X_\ell$ .

### 5.5. Detalles de RNN

El cálculo de cada unidad en las capas intermedias  $A_\ell$  se puede escribir como sigue. Supongamos que  $X_\ell \in \mathbb{R}^p$  y que  $A_\ell \in \mathbb{R}^K$ . Entonces con las matrices de pesos  $W \in \mathbb{R}^{p \times K}$ ,  $U \in \mathbb{R}^{K \times K}$  y los coeficientes  $\beta_0 \in \mathbb{R}$  y  $\omega_0, \beta \in \mathbb{R}^K$  con  $\ell = 1, \dots, L$  las pasos intermedios y las salidas se calcular por medio de

$$A_\ell = g\left(W^\top x_\ell + U^\top A_{\ell-1} + \omega_0\right), \quad (18)$$

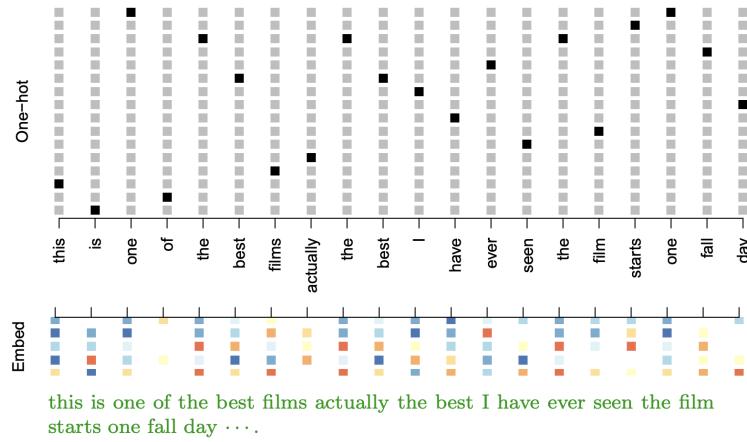
$$O_\ell = \beta_0 + \beta^\top A_\ell. \quad (19)$$

Usualmente nos interesa media la salida en la última capa para contrastarla con la predicción

$$\sum_{i=1}^n (y_i - O_{iL})^2. \quad (20)$$

### 5.6. Regresando a las reseñas

- Los documentos son nuestras observaciones como secuencias de palabras  $\{\mathcal{W}_\ell\}_{\ell=1}^L$ .
- Cada palabra  $\mathcal{W}_\ell$  esta representada como un vector de binarios  $X_\ell$  de longitud  $10K$ .
- Se pueden usar *word embeddings* (representaciones vectoriales de palabras) utilizando una matriz  $E \in \mathbb{R}^{m \times 10K}$ .
- Usualmente,  $m \ll 10K$ .



Los *embeddings* se pueden aprender con algún procedimiento de pre-entrenamiento. Los mas comunes son `word2vec` o `GloVe` (puedes consultar [4] para mas detalles).

## 5.7. Conclusiones

- Sólo tocamos el modelo mas sencillo de RNN hay mas variaciones con arquitecturas mas complejas (LSTM o GRU).
- Se pueden combinar con CNN para el ajuste. Aplicando filtros a lo largo de la secuencia.
- La construcción de modelos mas complejos utilizan la noción de bloques: un bloque de convolución, un bloque de secuencias recurrentes, etc.
- Se puede extender a secuencias. Los modelos seq2seq donde se aprenden secuencias como salidas (traducción de textos).

## 6. CASOS DE USO

- CNN para visión por computadora.
- RNN para modelos de lenguaje o predicciones secuenciales (series de tiempo).
- ¿Siempre son las mejores soluciones?

Cuanto tienes un martillo en la mano, a todo le ves cara de clavo. —Dicho popular.

- Son usualmente exitosas cuando la razón señal - ruido es alta.
- Para aplicaciones donde debemos de modelar el componente aleatorio, modelos mas sencillos son preferibles.

## 7. AJUSTE Y REGULARIZACIÓN

El ajuste de una red neuronal se realiza buscando la configuración del modelo que tenga el menor error posible en el conjunto de entrenamiento. Es decir,

$$\min_{\theta} \frac{1}{2} \sum_{i=1}^n (y_i - f_{\theta}(x_i))^2, \quad (21)$$

donde  $\theta$  son los parámetros a ajustar. Por ejemplo, para una red con una capa intermedia

$$f_{\theta}(x_i) = \beta_0 + \sum_{k=1}^K \beta_k g \left( \omega_{k0} + \sum_{j=1}^p \omega_{kj} x_{ij} \right). \quad (22)$$

### 7.1. Descenso en gradiente

- Denotemos por  $R(\theta)$  la función de pérdida para el ajuste de una red neuronal.
- Descenso en gradiente es un algoritmo iterativo que busca el mínimo de la función  $R(\theta)$ .
- El procedimiento opera de tal forma que  $R(\theta^{t+1}) < R(\theta^t)$ . Buscando actualizar

$$\theta^{t+1} = \theta^t - \rho \nabla R(\theta^t). \quad (23)$$

- El parámetro,  $\rho$  es la tasa de aprendizaje.

### 7.2. Gradientes y backpropagation

- La función  $R(\theta) = \sum_{i=1}^n R_i(\theta)$  es una suma. El gradiente, es una suma.
- Consideremos

$$f_\theta(x_i) = \beta_0 + \sum_{k=1}^K \beta_k g\left(\omega_{k0} + \sum_{j=1}^p \omega_{kj} x_{ij}\right). \quad (24)$$

- Denotemos  $z_{ik} = \omega_{k0} + \sum_{j=1}^p \omega_{kj} x_{ij}$ .
- Entonces usamos la regla de la cadena

$$\frac{\partial R_i(\theta)}{\partial \beta_k} = \frac{\partial R_i(\theta)}{\partial f_\theta(x_i)} \cdot \frac{\partial f_\theta(x_i)}{\partial \beta_k} \quad (25)$$

$$= -(y_i - f_\theta(x_i)) \cdot g'(z_{ik}), \quad (26)$$

y también tendríamos

$$\frac{\partial R_i(\theta)}{\partial \omega_{kj}} = \frac{\partial R_i(\theta)}{\partial f_\theta(x_i)} \cdot \frac{\partial f_\theta(x_i)}{\partial g(z_{ik})} \cdot \frac{\partial g(z_{ik})}{\partial z_{ik}} \cdot \frac{\partial z_{ik}}{\partial \omega_{kj}} \quad (27)$$

$$= -(y_i - f_\theta(x_i)) \cdot \beta_k \cdot g'(z_{ik}) \cdot x_{ij}, \quad (28)$$

### 7.3. Observaciones

- El procedimiento puede ser lento. Sin embargo, podemos utilizar early stopping como un mecanismo de regularización.
- Podemos utilizar descenso en gradiente estocástico (SGD) para acelerar el cómputo y mejorar el avance. Usar minibatches de 128 observaciones.
- Una época (epoch) es el número de iteraciones que se necesitan para usar todos los datos si procesamos los datos en minibatches.
- Podemos usar regularización en los pesos de la red. Pero también podemos usar dropout o aumento de datos.

### 7.4. Regularización: Dropout



- Con regularización con *dropout* se busca que en cada iteración de descenso en gradiente se desactiven algunas unidades con probabilidad  $\phi$ .
- Los pesos de las unidades que quedan se rescalan por  $1/(1 - \phi)$ .
- En regresión se puede probar que es equivalente a ridge.
- Los pesos tienden a agruparse.
- Es equivalente a quitar variables en el curso del entrenamiento (como en bosques aleatorios).

### 7.5. Regularización: Aumento de datos



- Especialmente útil con SGD en CNN.
- Se buscan transformaciones naturales de las imágenes.
- La categoría no cambia.
- Mejora la capacidad predictiva.

## 8. SOFTWARE

- El avance en *software* ha sido impresionante con Tensorflow y Pytorch.
- El paquete de keras es una colección de métodos que hacen la definición de una red neuronal más sencilla.
- El paquete de torch de R es un paquete que permite el ajuste de modelos sin usar la interfase a python.

## REFERENCIAS

- [1] G. James, D. Witten, T. Hastie, and R. Tibshirani. *An Introduction to Statistical Learning: With Applications in R*. Springer Texts in Statistics. Springer US, New York, NY, 2021. [1](#), [6](#), [12](#)
- [2] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958. ISSN 1939-1471. . [1](#)
- [3] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, oct 1986. ISSN 1476-4687. . [1](#)
- [4] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola. *Dive into Deep Learning*. In print, 2021. [1](#), [14](#)