Universidad
Francisco de Vitoria
UFV Madrid

Inteligencia Artificial II

Tema 5:
Deep Learning

Departamento de Ingeniería Informática
Universidad Francisco de Vitoria

Álvaro José García Tejedor
Curso 2022/23

Objetivos del tema

- Ubicación
  - Bloque II: **COMPUTACION NEURONAL**
    - Tema 2: *Aprendizaje no supervisado: Aprendizaje competitivo*
    - Tema 3: *Aprendizaje supervisado: Modelos lineales*
    - Tema 4: *Aprendizaje supervisado: Modelos no lineales*
    - Tema 5: ***Deep Learning***

- Objetivos
  - Entender el Deep Learning y las arquitecturas DNN, su uso y aplicación
  - Comprender cómo es el aprendizaje en las DNN y los problemas que presenta
  - Conocer las soluciones al problema del entrenamiento de capas profundas.

Contenido

1. Introducción
2. Deep Learning
    1. Definición
    2. Historia
    3. Arquitecturas DNN
    4. Tipos de aprendizaje
3. Entrenamiento
    1. El problema
    2. Soluciones basadas en modelos
    3. Cambios en el entrenamiento
    4. Regularización

1. Introducción
2. Deep Learning
    1. Definición
    2. Historia
    3. Arquitecturas DNN
    4. Tipos de aprendizaje
3. Entrenamiento
    1. El problema
    2. Soluciones basadas en modelos
    3. Cambios en el entrenamiento
    4. Regularización

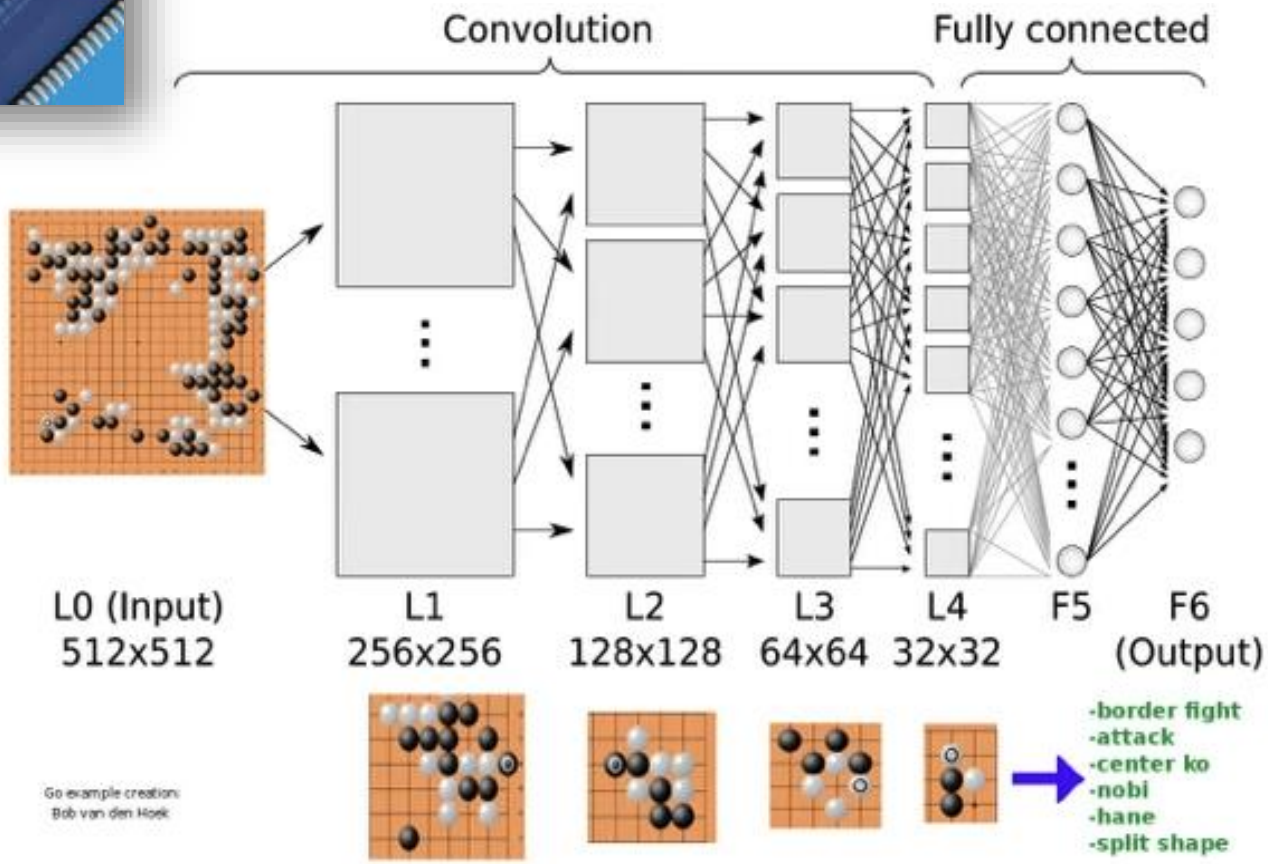
1. Introducción

- Un MLP con más de 2 capas ocultas ya es un modelo profundo

- Problema de los algoritmos de retropropagación: el error se va diluyendo de forma exponencial a medida que atraviesa capas hacia la capa de entrada
  - En una red muy profunda sólo las últimas capas se entrenan mientras que las primeras apenas sufren cambios.
  - A veces compensa utilizar redes con pocas capas ocultas que contengan muchas neuronas en lugar de redes con muchas capas ocultas que contengan pocas neuronas.

- Esto era así hasta que se desarrollaron los algoritmos y los modelos neuronales de *Deep Learning* (DL) que están formados por múltiples capas

1. Introducción



1. Introducción
2. Deep Learning
   1. Definición
   2. Historia
   3. Arquitecturas DNN
   4. Tipos de aprendizaje
3. Entrenamiento
   1. El problema
   2. Soluciones basadas en modelos
   3. Cambios en el entrenamiento
   4. Regularización

2.1 Definición de Deep Learning

Aprendizaje Profundo o **Deep Learning**

Área del *Machine Learning* que utiliza diferentes algoritmos de aprendizaje automático para modelar abstracciones de datos de alto nivel usando arquitecturas jerárquicas llamadas Redes Neuronales Profundas (*DNN*).

- Concepto de *profundidad*
  - Es la profundidad del grafo o dibujo correspondiente a una arquitectura, es decir, el camino mas largo desde un nodo de entrada hasta un nodo de salida.
  - Cuando el conjunto de elementos computacionales son neuronas, la profundidad corresponde al numero de capas de la red neuronal.

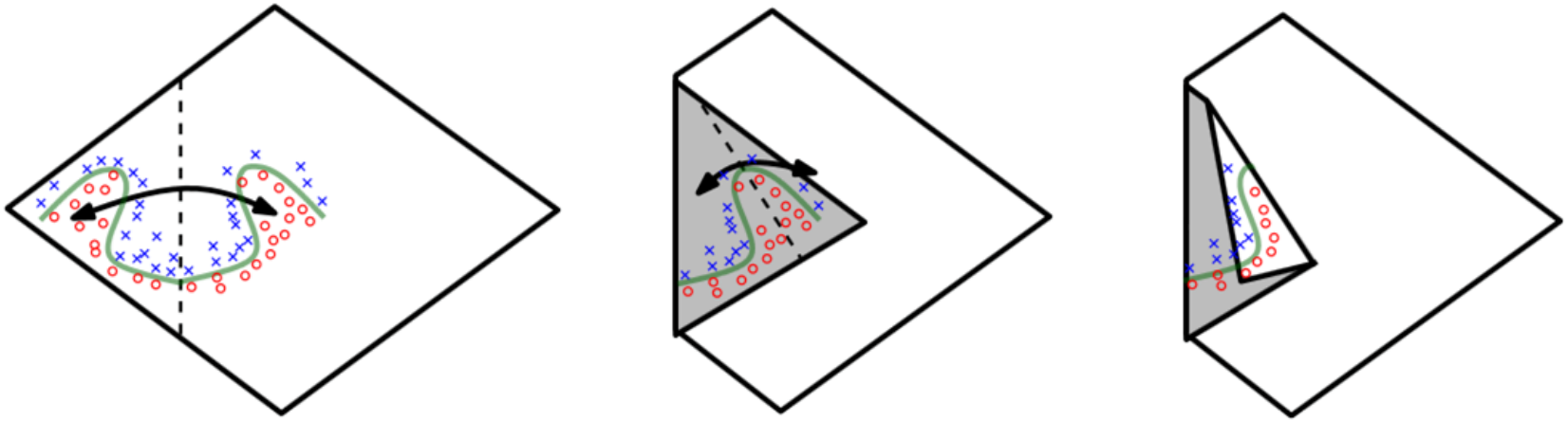
2.1 Definición de Deep Learning

- El Teorema de Aproximación Universal dice que *una única capa oculta es suficiente para aproximar* (que no aprender) *cualquier función con el grado de precisión deseado.*
  - Es decir, puede encontrar la relación entre los pares de entrada/salida del conjunto de entrenamiento
  - Pero no se garantiza que lo pueda hacer para pares nuevos (no garantiza que generalice)

- Entonces… ¿Por qué DL?
  1. Las redes generalizan mejor cuanto mayor es la profundidad. Una red neuronal con una sola capa oculta no es capaz de aprender representaciones jerárquicas.
  2. Las redes superficiales necesitan (exponencialmente) más anchura (más neuronas en la única capa oculta) para alcanzar la misma precisión.
  3. Las redes superficiales se sobreentrenan más fácilmente

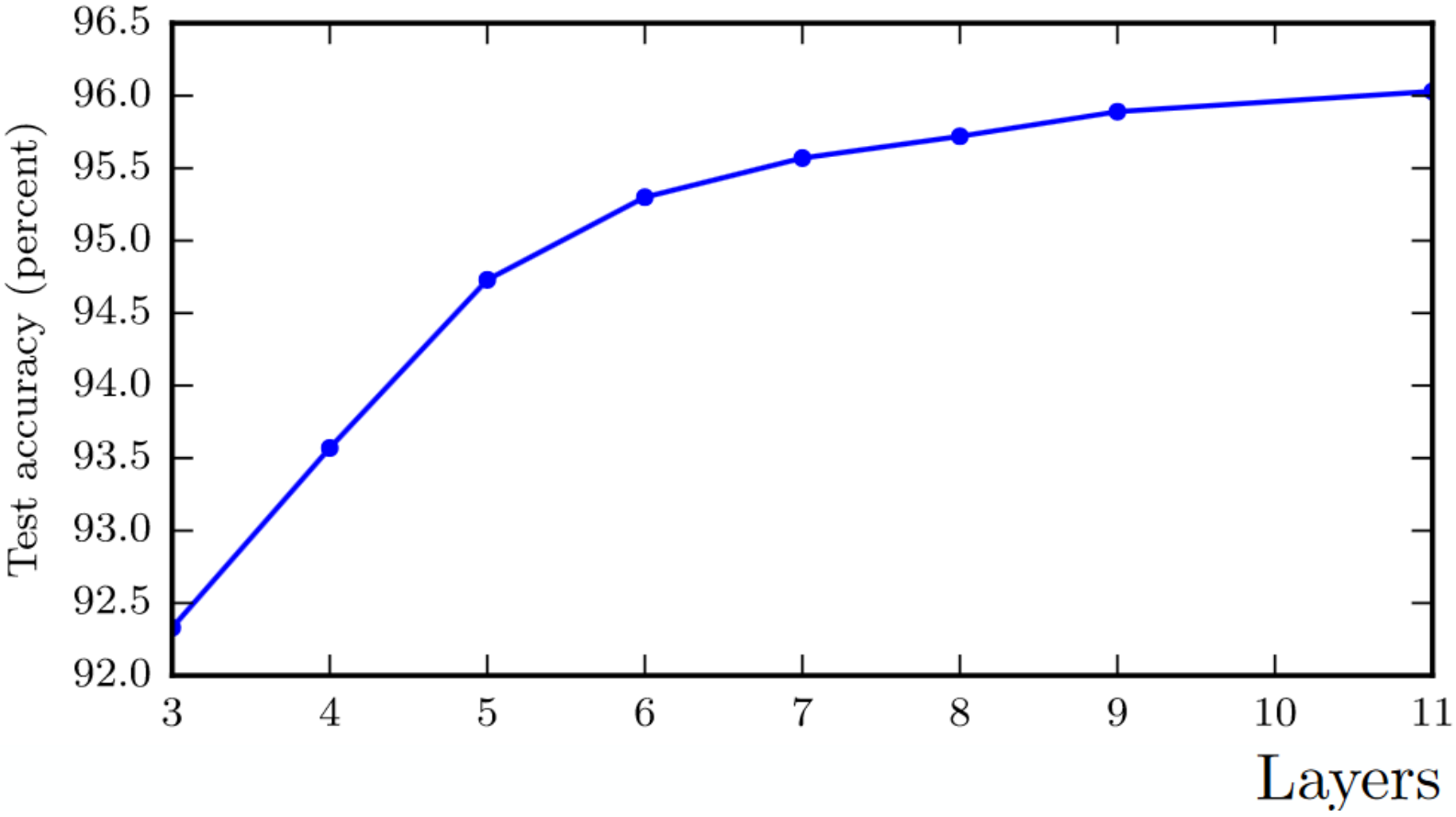
2.1 Definición de Deep Learning

- Generalización en modelos DNN
  - Múltiples capas de funciones lineales calcularían solo una función lineal de la entrada
  - Múltiples capas de funciones NO lineales permite computar características mucho más complejas de las entradas
    - Cada capa oculta computa una transformación no lineal de la salida de la capa anterior
    - Cada capa oculta aprende a representar datos con los conceptos aprendidos de las anteriores
    - Por lo tanto, un DNN tiene más capacidad de representación que una superficial (puede aprender funciones más complejas)



2.1 Definición de Deep Learning

- Mejor generalización con mayor profundidad



2.2 Historia

- Las técnicas DL existen desde hace bastante tiempo aunque no se consideraban viables por:
  1. La no disponibilidad de grandes cantidades de datos para entrenar a los modelos.
  2. No tener algoritmos de entrenamiento que permitan entrenar las capas más profundas.
  3. No tener HW en los ordenadores lo suficientemente potente como para la ejecución de los modelos.

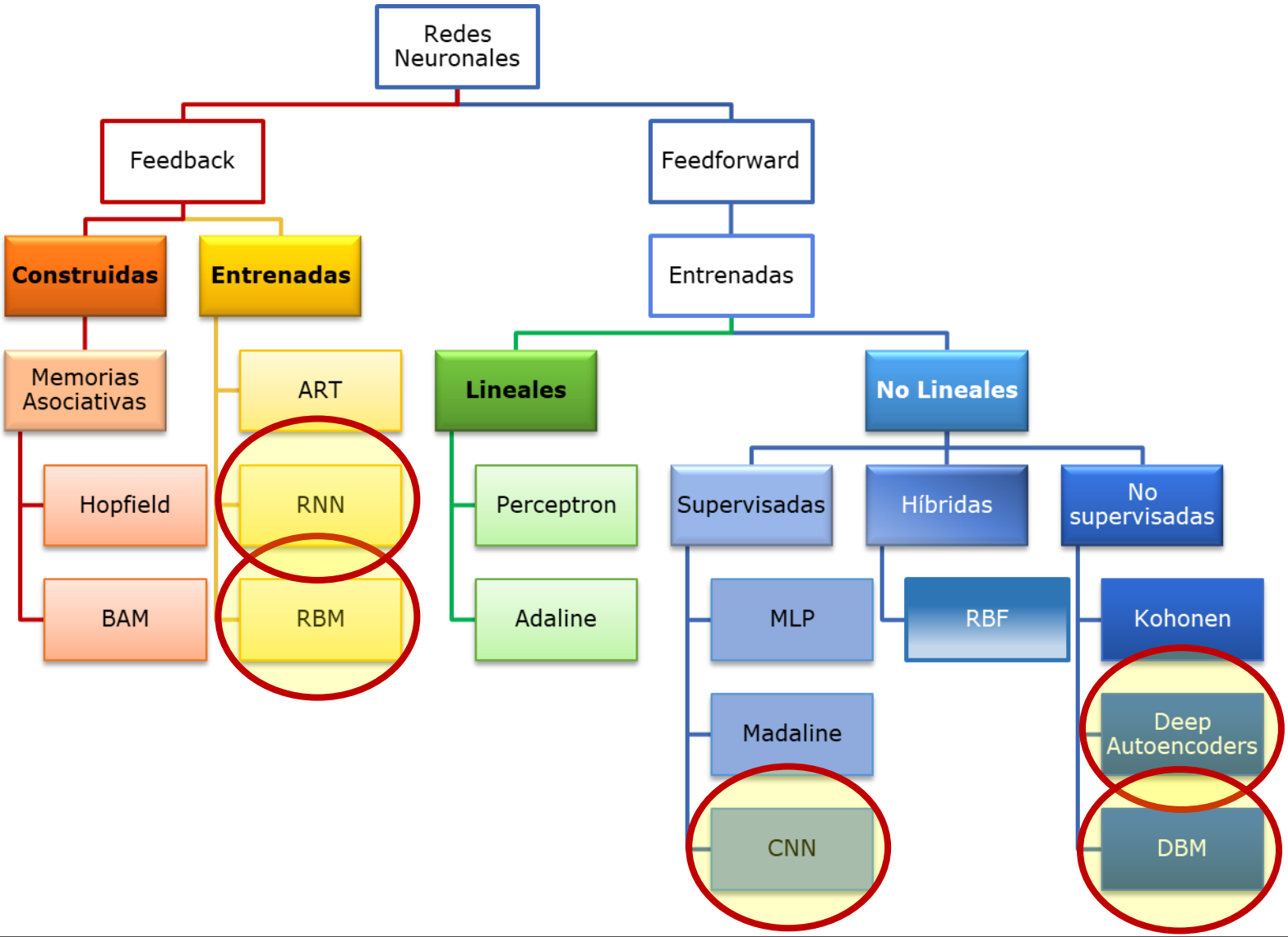
En los últimos años estos problemas se han visto solventados

- El primer éxito en el uso del aprendizaje profundo se debe a Geoffrey Hinton (2006) quien introdujo las *Redes de Creencia Profunda* (DBN) utilizando en cada capa de la red una *Maquina de Boltzmann Restringida* (RBM) para la asignación inicial de los pesos sinápticos

2.2 Historia

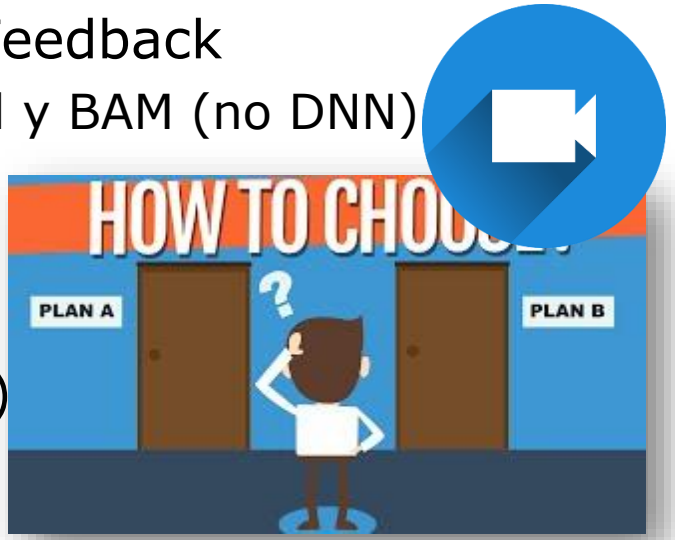
- Posteriormente se han desarrollado varias técnicas que permiten el aprendizaje en redes profundas, basadas en el *descenso estocástico de gradiente* + *backpropagation*
  - Estas técnicas permiten el entrenamiento de redes muy profundas (no hay problema en 5-10 capas ocultas)
  - Estas redes profundas se comportan mucho mejor que los redes poco profundas (MLP con una sola capa oculta)

2.3 Arquitecturas DNN



2.3 Arquitecturas DNN

- Redes Neuronales Convolucionales (CNN)
- Deep Auto-Encoders (DAE)
  - Sparse autoencoders (SAE)
  - Denoising autoencoders (DAE)
  - Contractive autoencoders (CAE)
- Máquinas de Boltzmann Restringidas (RBM)
  - Deep Belief Networks (DBN)
  - Máquinas de Boltzmann Profundas (DBM)
- Redes Neuronales Recurrentes (RNN) Feedback
  - Más de 13 modelos, incluyendo Hopfield y BAM (no DNN)
  - Fully recurrent (Basic RNN)
  - Long Short-Term Memory (LSTM)
  - Gate Recurrent Unit (GRU)
- Generative Adversarial Networks (GAN)



2.4 Tipos de aprendizaje

Aprendizaje no supervisado

- Cuando se entrena con datos sin etiquetar, cada capa de neuronas en una red profunda aprende características automáticamente al tratar de reconstruir la entrada de la que extrae sus muestras, minimizando la diferencia entre las conjeturas de la red y la distribución de probabilidad de los datos de entrada en sí.
- En el proceso, estas redes aprenden a reconocer las correlaciones entre ciertas características relevantes y los resultados óptimos.
- Tipos
  - Autoencoders
  - GAN
  - RBM

2.4 Tipos de aprendizaje

Aprendizaje supervisado

- El entrenamiento con datos etiquetados se puede aplicar a datos no estructurados, lo que le da acceso a mucha más información.
  - Conclusión: con cuantos más datos se pueda entrenar una red, más precisa será

    malos modelos entrenados en una gran cantidad de datos pueden superar a buenos modelos poco entrenados
- Una técnica muy habitual en *Deep Learning* consiste en empezar el entrenamiento supervisado, en lugar de empezar con pesos al azar, con pesos pre-entrenados, especialmente para las primeras capas.
- Tipos
  - CNN
  - RNN

2.4 Tipos de aprendizaje

Aprendizaje por refuerzo

- Modelos orientados a objetivos: aprenden cómo alcanzar un objetivo complejo o maximizar una dimensión particular a lo largo de muchos pasos
  - Ejemplo: maximizar los puntos ganados en un juego a lo largo de muchos movimientos.
  - Al igual que un niño, estos algoritmos se penalizan cuando toman las decisiones equivocadas y se los recompensa cuando toman las correctas.
- Los algoritmos de refuerzo que incorporan el aprendizaje profundo han vencido a campeones humanos
  - Juegos complejos como el Go o el ajedrez (AlphaGo, AlphaGo Zero, AlphaZero)
  - Videojuegos (videojuegos de Atari o Starcraft: AlphaStar).

1. Introducción
2. Deep Learning
    1. Definición
    2. Historia
    3. Arquitecturas DNN
    4. Tipos de aprendizaje
3. Entrenamiento
    1. El problema
    2. Soluciones basadas en modelos
    3. Cambios en el entrenamiento
    4. Regularización

3.1 El problema

- A pesar de los beneficios de las DNN, es muy difícil entrenar arquitecturas profundas y hasta hace muy poco no se había conseguido entrenarlas satisfactoriamente

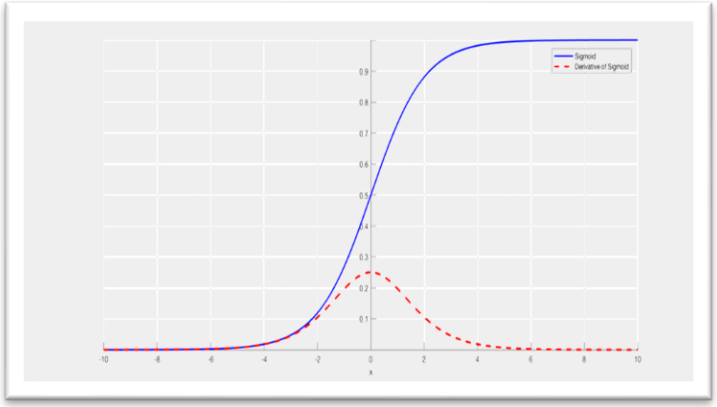
- El método clásico de descenso de gradiente para arquitecturas supervisadas no funciona por tres motivos:
  - *Disponibilidad de datos*
    - Datos etiquetados son difíciles de obtener
    - Dada el alto poder de expresividad, entrenar con datos insuficientes produce overfitting
  - *Óptimos locales*
    - El aprendizaje supervisado implica resolver un problema de optimización altamente no convexo (minimizar el error de entrenamiento en función del peso de la red)
    - En una red profunda, este problema se complica al existir muchos más óptimos locales, y el entrenamiento con descenso de gradiente (o el de gradiente conjugado) ya no funcionan bien.

3.1 El problema

- *Difusión de gradientes (vanishing/exploding gradients)*
  - La sensibilidad de una neurona de la última capa oculta es

$$\delta_h^p = f_h'(S_h^p) \sum_{k=1}^{m} W_{hk}\, \delta_k^p$$

contiene el producto de las derivadas de dos funciones de activación: $f_h'(S_h^p)$ y δ_k^p.



Si $f = sigmoide \rightarrow f'_{\max} = 0{,}25 \rightarrow f' * f' \ll 1$

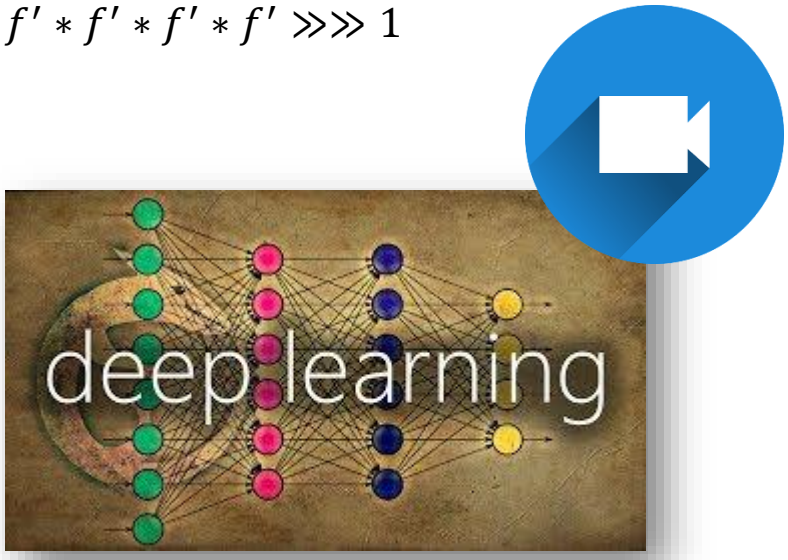
  - Para capas más profundas la derivada del error con respecto a los pesos (sensibilidad) es más pequeña todavía. Los pesos de las capas anteriores cambian lentamente y las capas anteriores no aprenden mucho.
    Si $f'_{\max} = 0{,}25 \rightarrow f' * f' * f' * f' \lll 1$

  - Como también $\eta << 1$ $w^{t+1} \approx w^t$

3.1 El problema

- Además, si las últimas capas tienen suficiente número de neuronas serán capaces de generalizar sin ayuda de las anteriores capas, lo que equivale a entrenar un MLP menos profundo pero cuyos datos de entrada están corruptos (salida de las primeras capas sin entrenar).

- También puede ocurrir lo contrario (*exploding gradient*) cuando la derivada es mucho mayor que 1
  Si $f'_{\max} = 5 \rightarrow f' * f' * f' * f' \ggg 1$



3.2 Soluciones basadas en modelos

Uso de RBM

- Hinton (2006) propuso una estrategia en dos pasos para entrenar Deep Belief Networks (DBN) que pueden verse como una composición de redes simples no supervisadas (Restricted Boltzmann Machines - RBM):
  1. Cada capa de la red es entrenada utilizando un algoritmo de pre-entrenamiento no supervisado comenzando con la capa de entrada a la red
     - Se pueden guiar sus parámetros hacia mejores regiones dentro del hiperespacio de parámetros
  2. Después de pre-entrenar todas las capas, la red puede ser ajustada utilizando un entrenamiento supervisado:

descenso estocástico de gradiente (SGD) + backpropagación

3.2 Soluciones basadas en modelos

Entrenamiento de Autoencoders

- Misma estrategia en 2 pasos para entrenar autoencoders apilados (Deep Autoencoders - DAE):
  1. Entrenar cada capa encoder de forma no supervisada, especialmente las primeras.
     - El objetivo es reconstruir la entrada en la salida usando en medio una capa oculta con menos neuronas
     - De esta manera se ajustan los pesos de esa capa oculta
  2. A continuación se usa esa capa oculta como capa de entrada y se repite el proceso.
  3. Por último se combinan y se entrena todo la red de forma supervisada para hacer únicamente un ajuste fino de los pesos.
     - El entrenamiento supervisado no empieza con pesos al azar sino con pesos útiles especialmente para las primeras capas.

3.3 Cambios en el entrenamiento

Uso de Funciones de Activación que no saturen

- Funciones que mantienen el gradiente constante en cada capa de la red.
  - ReLU $f(x) = \max(0, x)$

    La pendiente en la parte positiva es siempre = 1 (ni se desvanece ni explota)

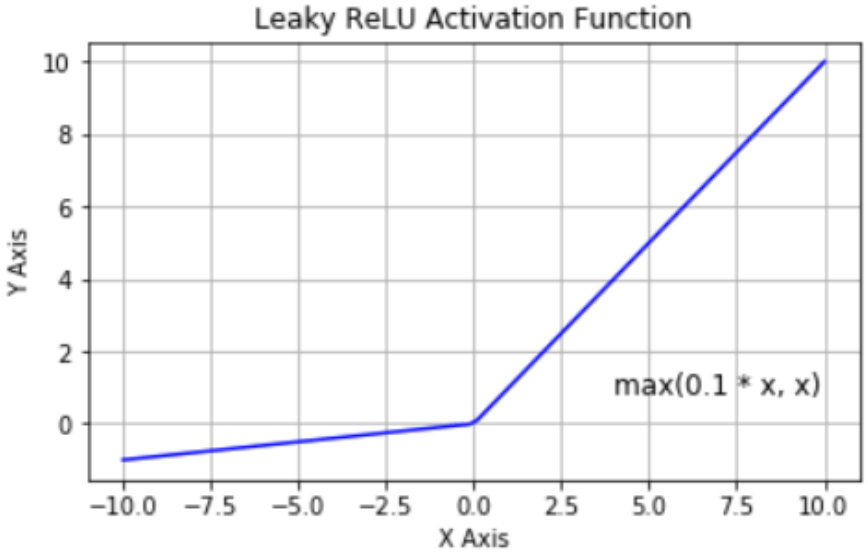
    La pendiente en la parte negativa es siempre = 0 (problema)
  - LeakyReLU $f(x) = \max(\alpha x, x)$

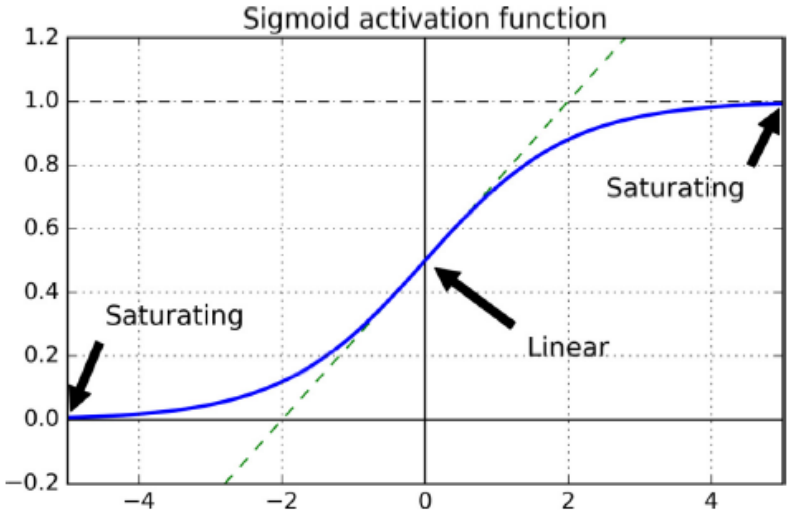
    En la parte negativa hay siempre una pequeña pendiente ($\alpha \ll 1$)

- Sigmoides simétricas (tanh) convergen más rápidamente que sigmoide
  - El orden preferente de uso de las funciones de activación es: *LeakyRelu > ReLU > Tanh > Sigm*
  - LeCun propone como función de activación

    f(x) = 1,7159 tanh (2x/3)

3.3 Cambios en el entrenamiento



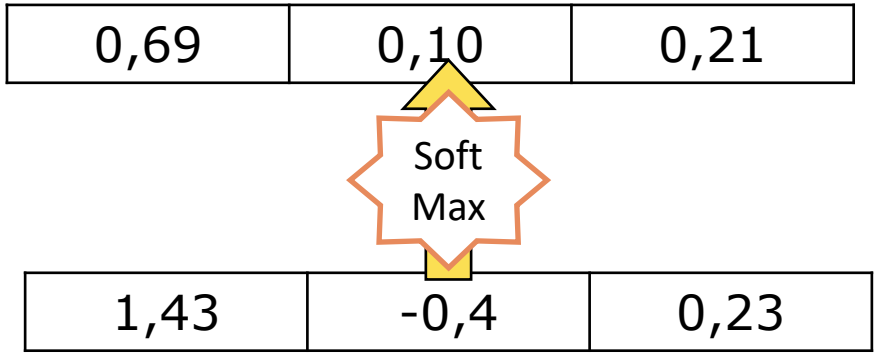


3.3 Cambios en el entrenamiento

- Uso de la función SoftMax para problemas de clasificación
  - Este modelo generaliza la regresión logística (sigmoidea).
  - Transforma un vector R^N de valores arbitrarios en un vector R^N de valores en el rango (0,1).

$$f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

  - Tiene más pendiente ➔ el gradiente no se desvanece tan fácilmente como con funciones lineales o sigmoideas.
  - Como $\sum_{i=1}^{n} f(x_i) = 1$ la salida de SoftMax se puede interpretar como una *distribución de probabilidad*.



3.3 Cambios en el entrenamiento

Uso de otras *Loss Function*

- *Loss Function (Función de coste)* es la magnitud que se minimiza durante el entrenamiento. Representa cuan buena es una red neuronal para un problema dado.
  - RMS es sólo una de las posibles Loss Functions, por lo que ya no se puede hablar propiamente de "*error*", sino de "*coste*"
- En problemas de clasificación, la opción más habitual es reemplazar el error cuadrático por la función *categorical cross-entropy loss*

$$CE = -\frac{1}{c}\sum_{i=1}^{c} d_i \, log(y_i)$$

  - d_i es el valor de la salida deseado (1 para la clase correcta, 0 para las demás)
  - y_i es la probabilidad de pertenencia predicha por la red
  - c es el número de clases

3.3 Cambios en el entrenamiento

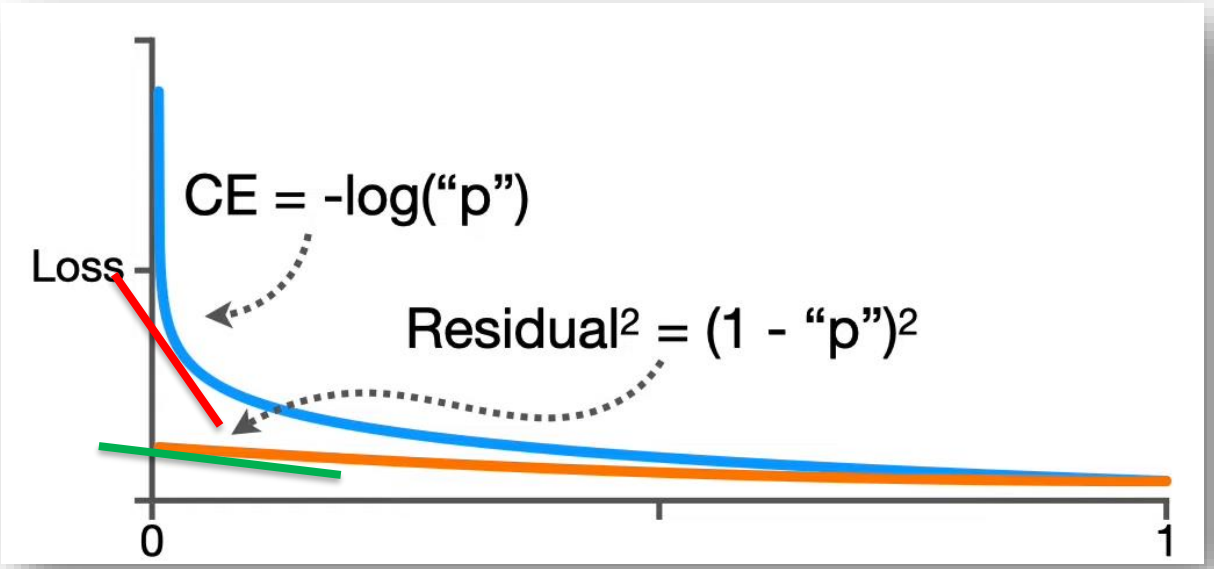
	Clase A	Clase B	Clase C
real	1	0	0
predicho	0,57	0,20	0,23

$$CE = -\sum_{i=1}^{c} d_i \log(y_i) = -d_A \log(y_A) - d_B \log(y_B) - d_C \log(y_C)$$

$$CE = -1 \log(0{,}57) - 0 \log(0{,}20) - 0 \log(0{,}23) = \mathbf{0,56}$$

- Si los valores predichos se calculan con la función SoftMax, entonces CE
  - Tiende a 0 para valores de y_i muy próximos a 1
  - Tiende a infinito para valores de y_i muy próximos a 0
- CE penaliza solo las predicciones correctas y mucho más fuertemente que el RMS y además su derivada presenta un gradiente mayor, lo que evita la saturación.

3.3 Cambios en el entrenamiento

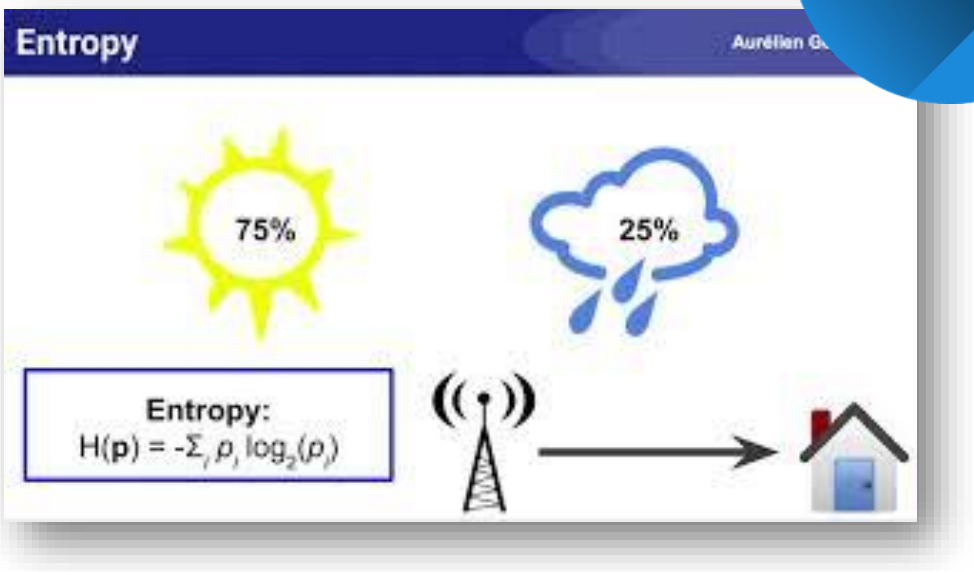


- *Combinando distintas funciones de activación con distintas funciones de coste podemos evitar el vanishing gradient*

Tipo de salida	Función de activación	Loss function
Binaria	Sigmoidea	Binary cross-entropy
Múltiples clases	SoftMax	Categorical cross-entropy
Numérica	Lineal	MSE

3.3 Cambios en el entrenamiento

AVANZADO



3.3 Cambios en el entrenamiento

Inicialización de los pesos

- Enfoque tradicional
  - Inicialización aleatoria con distribución normal N(0,1): media 0 y desviación 1.
    - Es aleatoria para romper la simetría de la red y evitar que todas las neuronas de una capa acaben aprendiendo lo mismo (dos neuronas con los mismos pesos siempre tendrán el mismo gradiente, por lo que no aprenderán características diferentes).
  - Problema: Si la función de activación es sigmoidea la varianza de las salidas en cada capa sea mayor que la varianza de sus entradas.
    - ➔La varianza aumenta hasta que se saturan las neuronas de las capas finales, lo que dificulta el aprendizaje
    - Cuando la derivada de la función de activación es cercana a 0, nos quedamos sin gradiente del error que propagar hacia atrás

3.3 Cambios en el entrenamiento

- Enfoque moderno (Deep Learning)
  - Si una neurona tiene un *fan-in* (n) elevado, pequeños cambios en muchos de sus pesos de entrada pueden influir mucho.
    - Cuanto mayor el fan-in ➔ pesos más pequeños
  - LeCun (1990): Inicialización aleatoria con distribución normal
    - $N(0,1/\sqrt{n})$ media 0 y desviación $1/\sqrt{n}$ si se usa su función de activación
    - $N(0,2/\sqrt{n})$ media 0 y desviación $2/\sqrt{n}$ si se usa ReLU
  - Glorot y Bengio (2010) Inicialización Xavier: Inicialización aleatoria con distribución normal $N(0,1/\sqrt{m})$ media 0 y desviación $1/\sqrt{m}$ donde m es la media del fan-in y el fan-out.
    - Usada con Factivación Lineal, Tanh, Sigmoidea y Softmax
    - Es la que usa Keras por defecto

3.3 Cambios en el entrenamiento

Uso de Optimizers

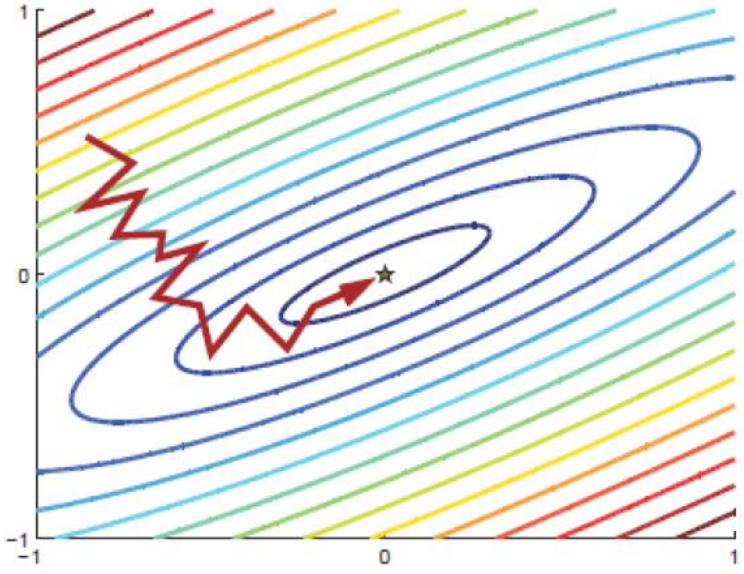
- Determinan cómo se optimizan los pesos de la red en función de la *Loss Function*.

- *GD* (*Batch Gradient Descend*) usado por defecto en BackProp
  - MSE es el promedio de la suma de los errores del training set completo (*batch gradient descend*).
  - Muy ineficiente porque en cada iteración del algoritmo hay que recorrer todo el training set para actualizar los pesos.
  - La dirección de máxima pendiente del gradiente no apunta necesariamente al mínimo salvo que las líneas de nivel sean circulares.

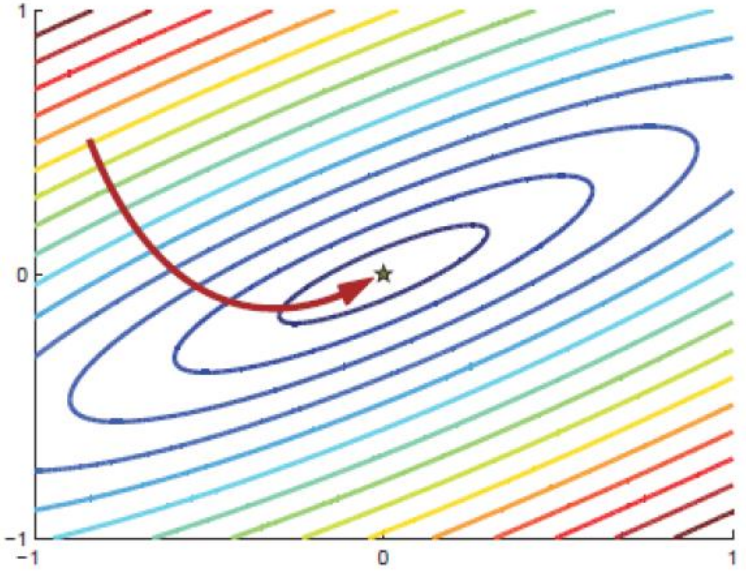
3.3 Cambios en el entrenamiento

- Alternativas
  - *Mini-batch Gradient Descent*: Se aproxima la derivada de un pequeño mini lote del conjunto de datos y usaremos esa derivada para actualizar los pesos.
    - No se garantiza que dé los pasos en la dirección óptima.
    - Los lotes tienen que estar balanceados.
  - *SGD (Stochastic Gradient Descent)*: Versión extrema del *Mini-batch Gradient Descent* con tamaño de batch=1
    - Aproximación estocástica del descenso de gradiente.
    - Indica la dirección en la que la función tiene el ratio de aumento más pronunciada aunque no indica hasta donde se debe avanzar en esa dirección.
    - Se mejora aún más con el uso del Momento

3.3 Cambios en el entrenamiento



SGD



BATCH

3.3 Cambios en el entrenamiento

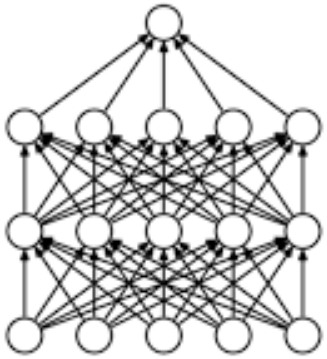
- Algoritmos de optimización adaptativos sobre el SGD
  - *RMSProp* (*Root Mean Square Propagation*):
    - Adapta la tasa de aprendizaje para cada parámetro por separado.
    - Usa el signo del gradiente para ver la dirección y lo combina con adaptar el tamaño del paso ya que los gradientes pueden variar mucho en magnitud y dirección.
  - *Adam* (*Adaptive Moment Estimation*):
    - Modificación del RMSProp que lo combina con el momento.
    - Adapta el ratio de aprendizaje en función de la media y varianza (cómo estén distribuidos los parámetros) del gradiente y la media de lso pasos anteriores.
    - Ahora mismo es el optimizador predeterminado.

3.4 Regularización

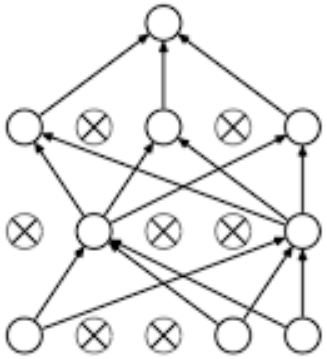
Evitan el *overfitting* y mejoran la *capacidad de generalización* compensando la exagerada cantidad de parámetros en la red.

Regularización por Dropout

- Técnica que desconecta al azar algunas neuronas de la capa completamente conectada durante el entrenamiento
  - Cada vez que presentamos un patrón desconectamos al azar cada neurona oculta con una probabilidad de, digamos, 50%
  - Estamos entrenando al azar con 2^h arquitecturas que comparten los pesos entre ellas
- Obliga a las capas totalmente conectadas a aprender el mismo concepto de diferentes maneras
- También se puede aplicar el dropout a la capa de entrada pero con menor probabilidad



(a) Standard Neural Net



(b) After applying dropout.

3.4 Regularización

Regularización L2 (weight decay)

- La forma más común de regularización.
- En la *Loss Function* agrega un término a cada peso que penaliza el valor al cuadrado de todos los parámetros que se optimizan.

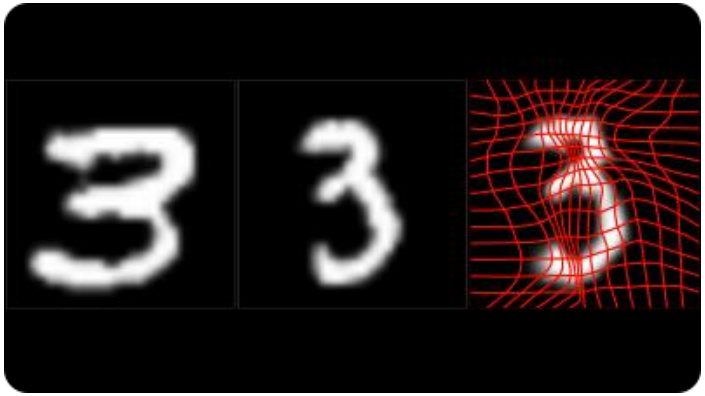
$$L2 = \frac{\lambda}{2}\left(\sum_{j=1}^{n} \|w_j\|^2\right)$$

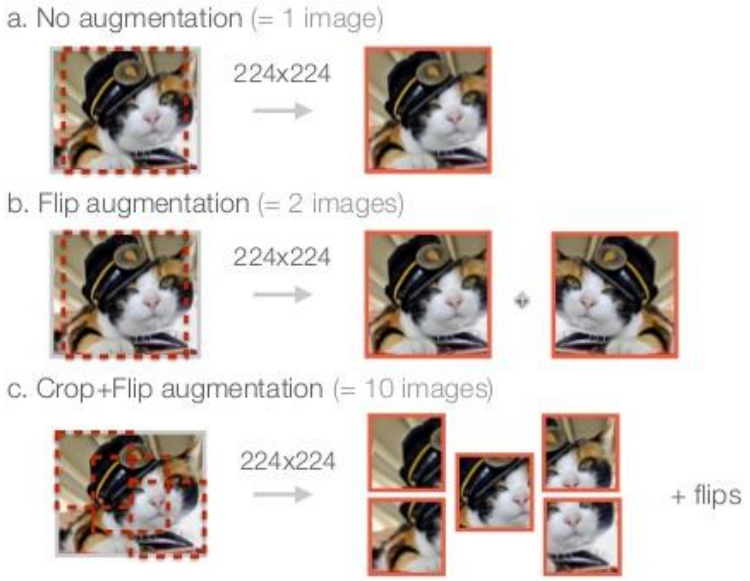
  - n es el número de capas
  - λ es el factor de regularización
  - Los pesos de muy alto valor son penalizados fuertemente.
  - El modelo tiende a preferir que todas las entradas a una capa se usen un poco en lugar de que algunas entradas se usen mucho.
  - El modelo se utilizará al máximo y habrá menos pesos no utilizados.

3.4 Regularización

Magnificación de datos (data augmentation)

- Crear sintéticamente nuevos datos de entrenamiento aplicando algunas transformaciones en los datos de entrada.
  - Desplazamiento de imágenes
  - Valores aleatorios de RGB.
- En ImageNet 2012, *Alex Krizhevesky* (Alexnet) usó un aumento de datos de un factor de 2048
  - El conjunto de datos utilizado para entrenar su modelo era 2048 veces mayor que al inicio





3.4 Regularización