Redes Neuronales y Deep Learning

Inteligencia Artificial e Ingeniería del Conocimiento

Constantino Antonio García Martínez

Universidad San Pablo Ceu

Introducción a Redes Neuronales y Deep Learning

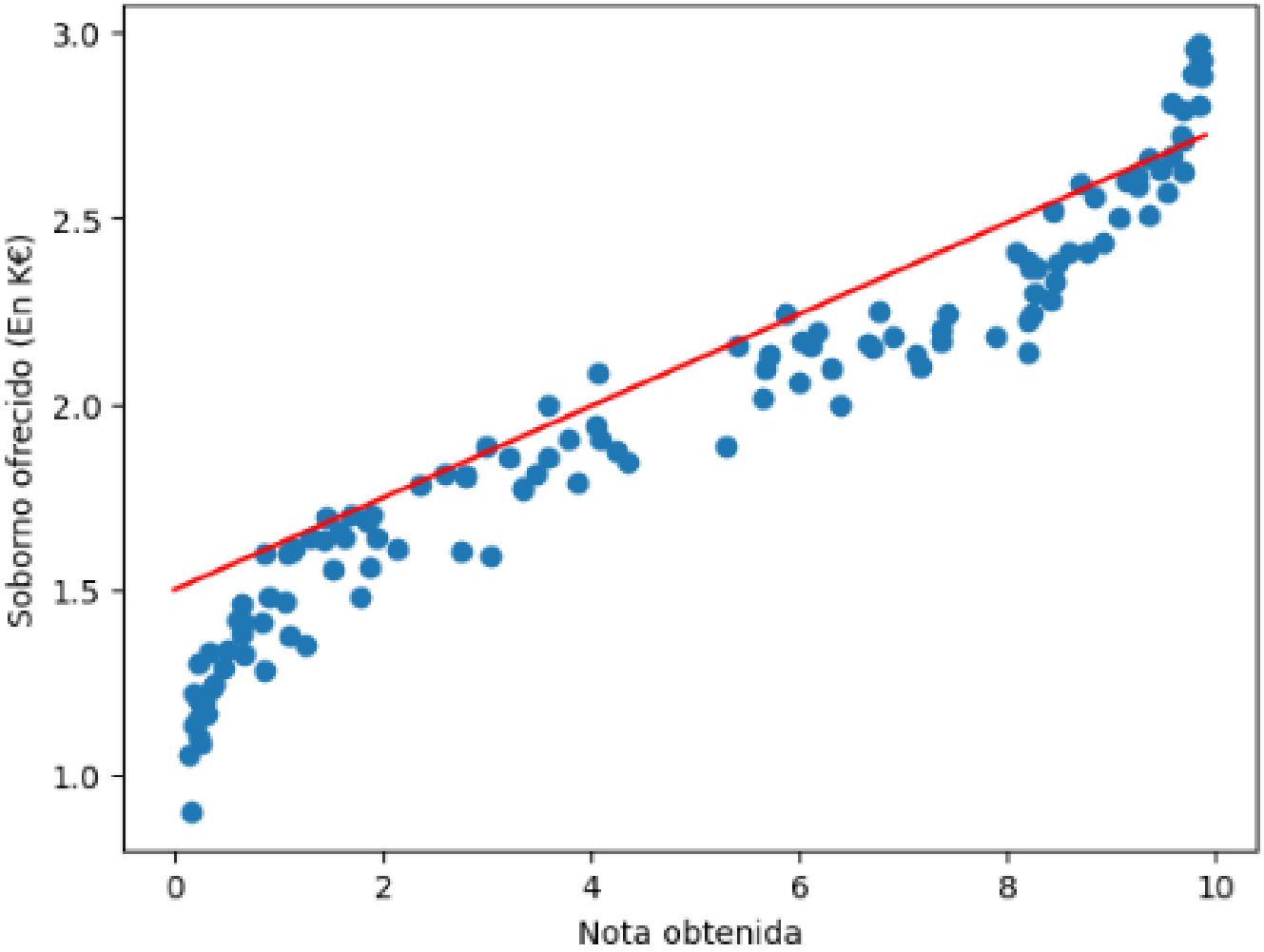
Introducción a Redes Neuronales y Deep Learning

Redes Neuronales

Example: Sobornos

¿Cómo podríamos mejorar la salida del modelo lineal del problema de los sobornos?

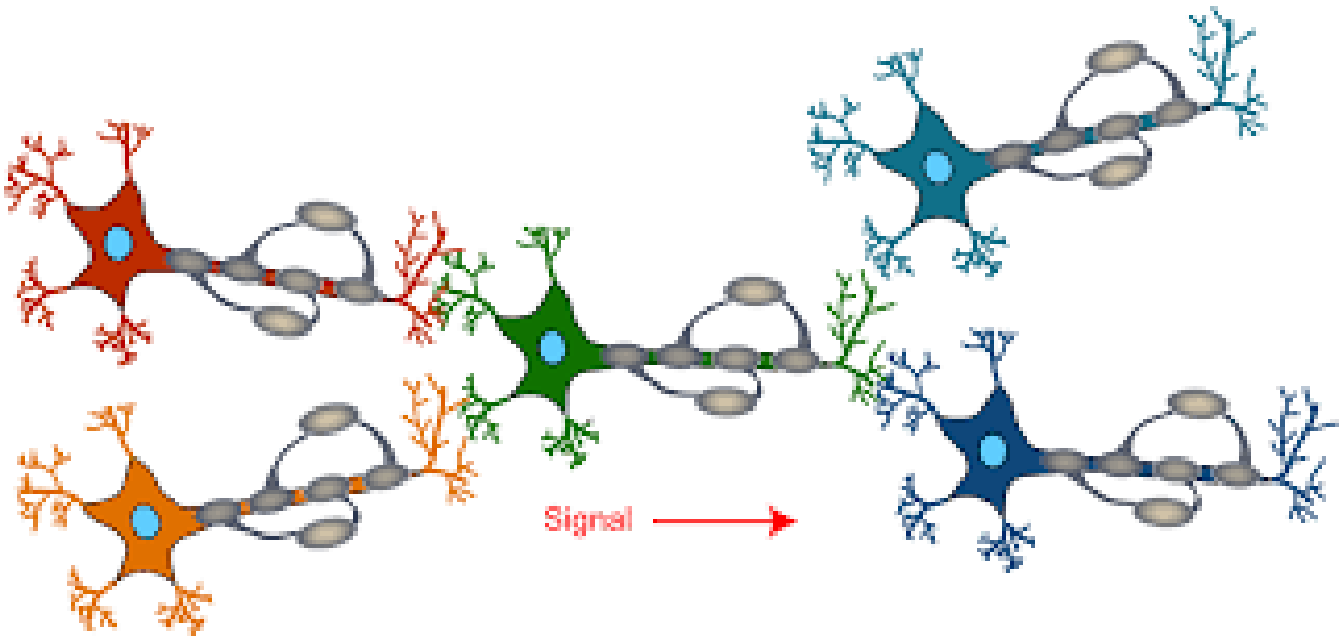
Redes Neuronales



La limitación del modelo surge del hecho de que los **modelos lineales solo producen hiperplanos como salidas**.

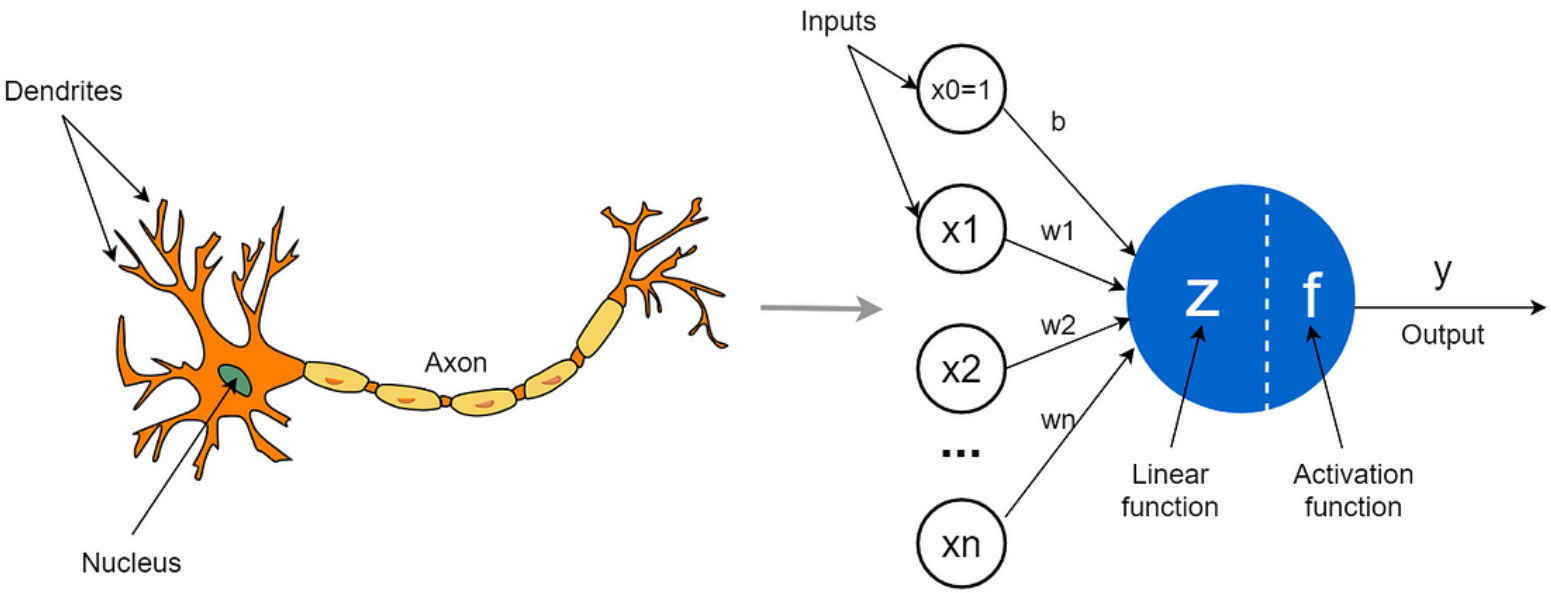
Inspiración biológica: neuronas

Un sistema no lineal interesante es la neurona:

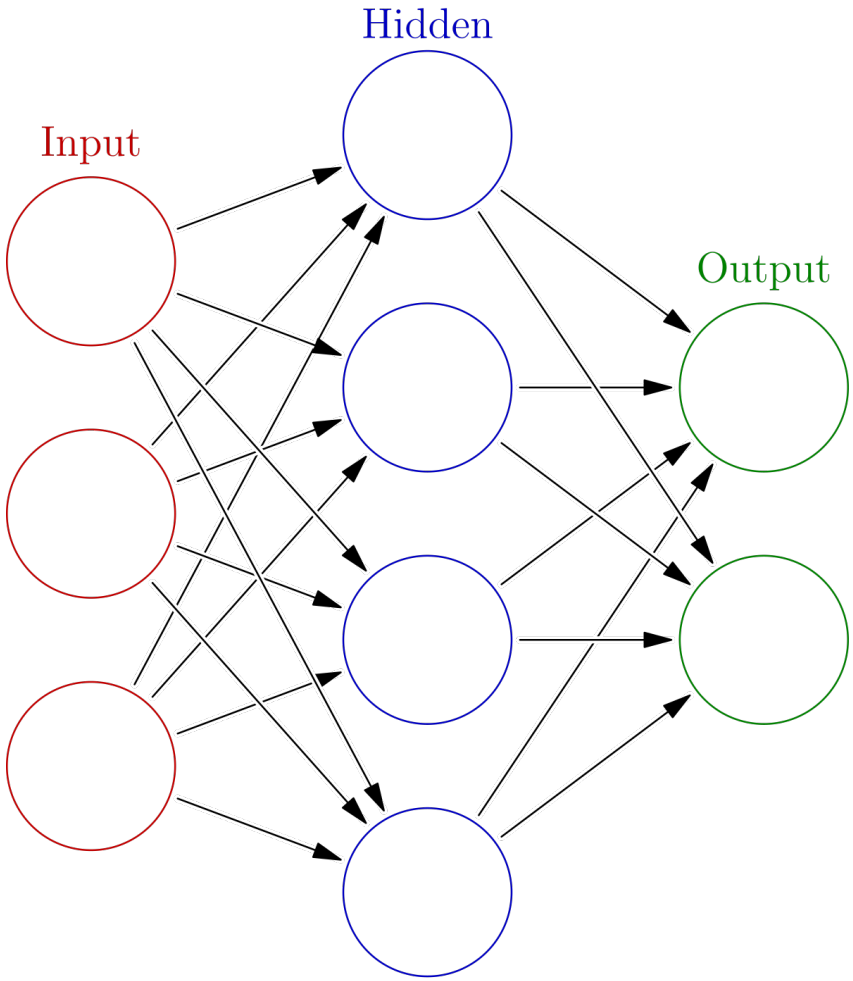


¡Observa la **activación no lineal**!

Neuronas artificiales



Redes Neuronales



El problema de los sobornos y redes neuronales

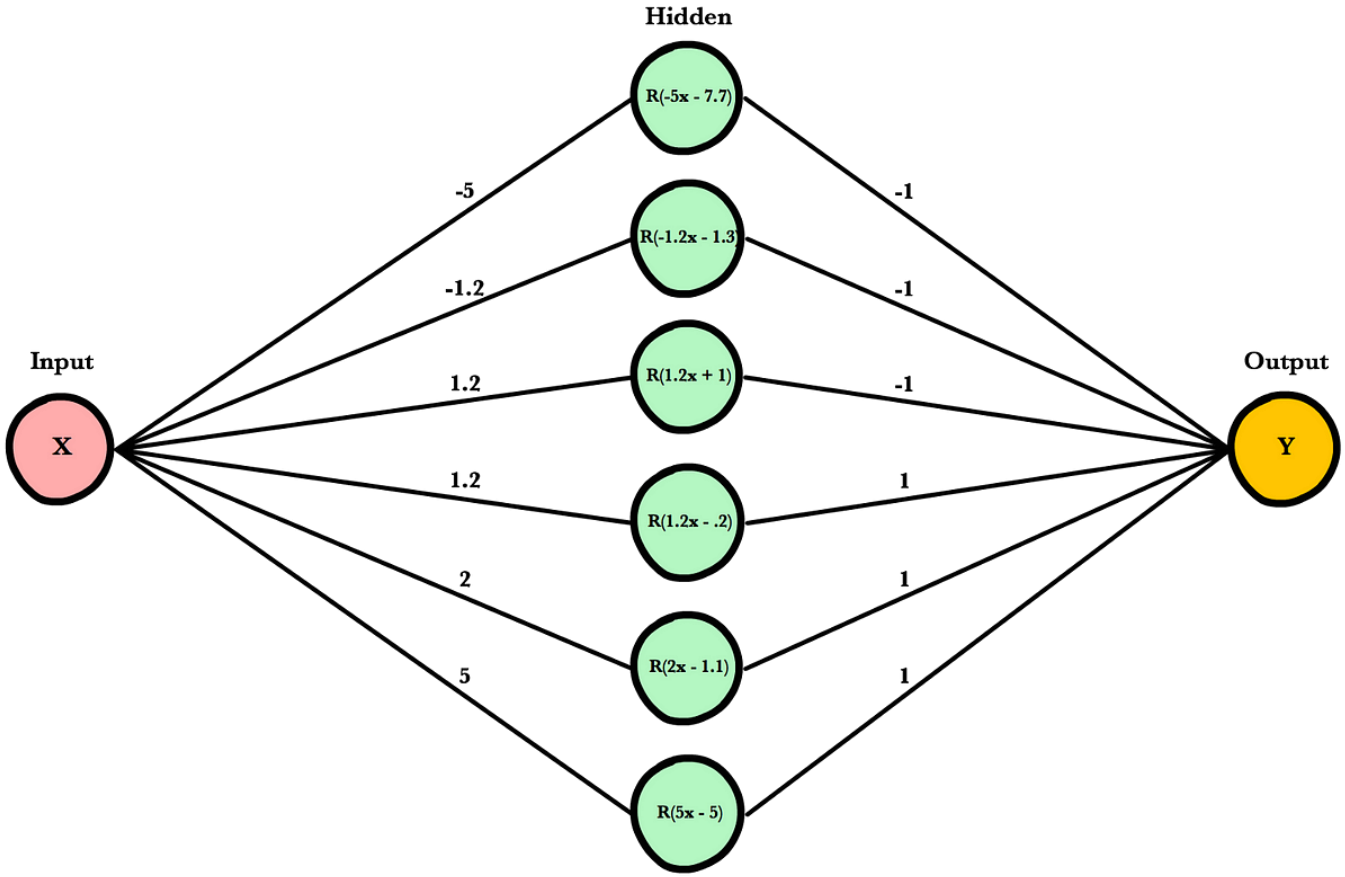
Example: Sobornos

¿Cuál es la arquitectura de una red neuronal con una única **capa oculta** en el problema de los sobornos?

El problema de los sobornos y redes neuronales

Example: Sobornos

¿Cuál es la arquitectura de una red neuronal con una única **capa oculta** en el problema de los sobornos?



El problema de los sobornos y redes neuronales

Example: Sobornos

Para aplicar una red neuronal a nuestro problema de sobornos (versión de una variable) necesitamos resolver dos problemas:

1. ¿Cómo mapear nuestra entrada única a h capas ocultas?
2. ¿Cómo "comprimir" la salida de las neuronas para simular la activación no lineal?

El problema de los sobornos y redes neuronales

Example: Sobornos

Para aplicar una red neuronal a nuestro problema de sobornos (versión de una variable) necesitamos resolver dos problemas:

1. ¿Cómo mapear nuestra entrada única a h capas ocultas?
2. ¿Cómo "comprimir" la salida de las neuronas para simular la activación no lineal?

Example: Sobornos

Para aplicar una red neuronal a nuestro problema de sobornos (versión de una variable) necesitamos resolver dos problemas:

1. ¿Cómo mapear nuestra entrada única a h capas ocultas?
2. ¿Cómo "comprimir" la salida de las neuronas para simular la activación no lineal?

1. El primer problema se resuelve mediante multiplicación matricial:

$$\underset{n\times h}{\mathrm{H}} = \underset{n\times 1}{X} \times \underset{1\times h}{\beta}$$

Example: Sobornos

Para aplicar una red neuronal a nuestro problema de sobornos (versión de una variable) necesitamos resolver dos problemas:

1. ¿Cómo mapear nuestra entrada única a h capas ocultas?
2. ¿Cómo "comprimir" la salida de las neuronas para simular la activación no lineal?

1. El primer problema se resuelve mediante multiplicación matricial:

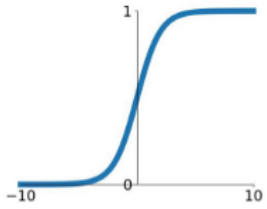
$$\underset{n\times h}{\mathrm{H}} = \underset{n\times 1}{X} \times \underset{1\times h}{\beta}$$

2. Simplemente aplicamos cualquier función no lineal a la salida de cada neurona. Esto se conoce como **función de activación**. Un ejemplo simple es la función sigmoide.

Activation Functions

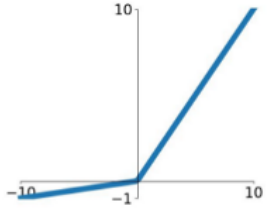
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



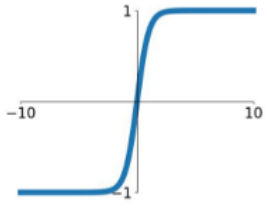
Leaky ReLU

$$\max(0.1x, x)$$



tanh

$$\tanh(x)$$

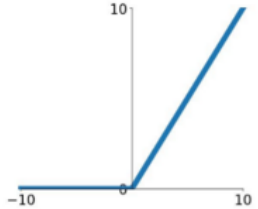


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

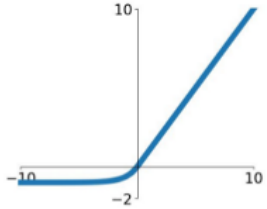
ReLU

$$\max(0, x)$$



ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Una función de activación que suele funcionar bien es **ReLU**.

El problema de los sobornos y redes neuronales

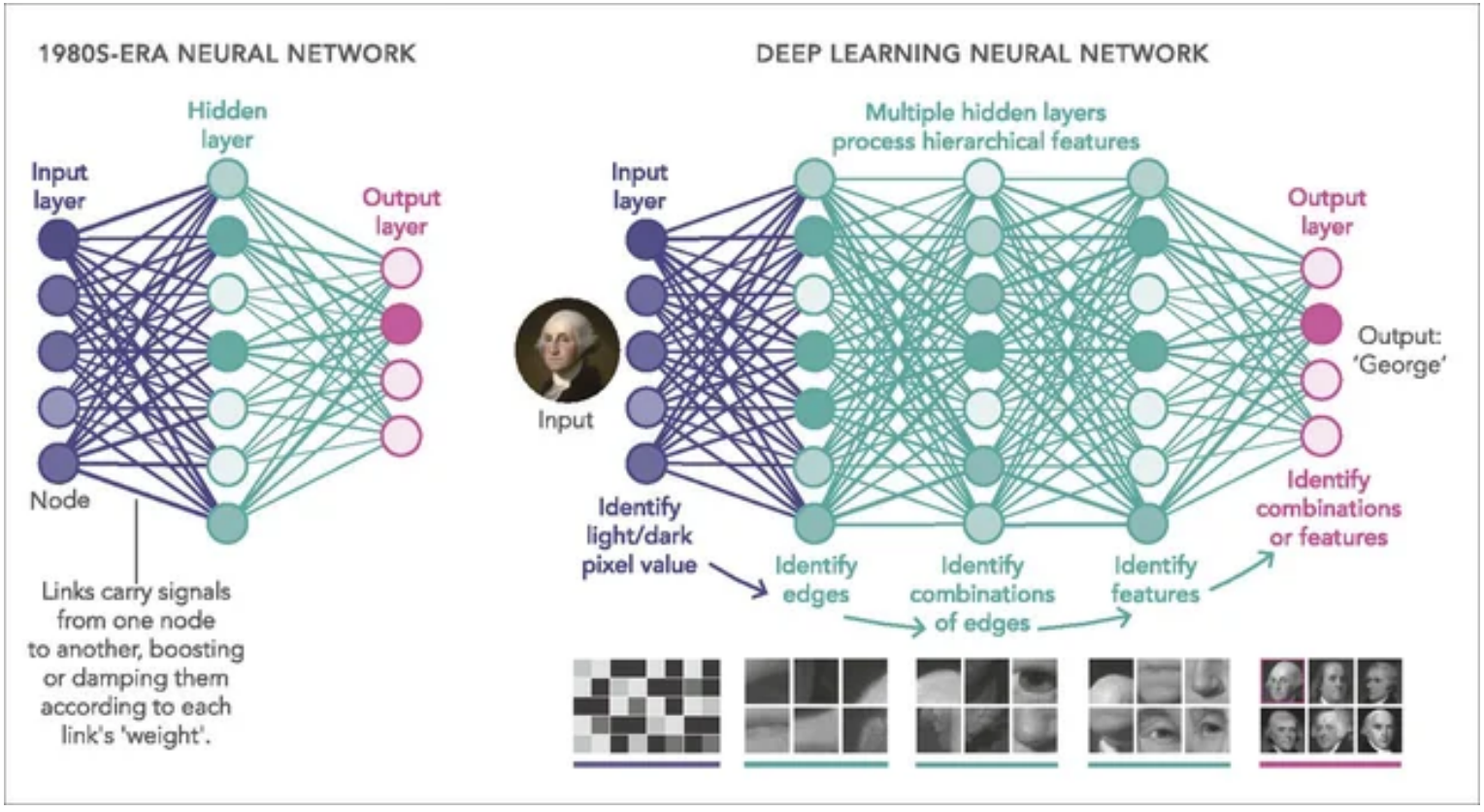
Code Example: NNet con Keras

```
model = keras.Sequential([
  keras.layers.InputLayer(shape=(1,)),
  keras.layers.Dense(512, activation='leaky_relu'),
  keras.layers.Dense(1, activation=None),
])
```

Introducción a Redes Neuronales y Deep Learning

Deep Learning

Redes Neuronales vs. Deep Learning



Típicamente, las redes neuronales (y la mayoría de algoritmos de ML) requieren ingeniería de características para funcionar bien. ¡Deep Learning automatiza la extracción de características!

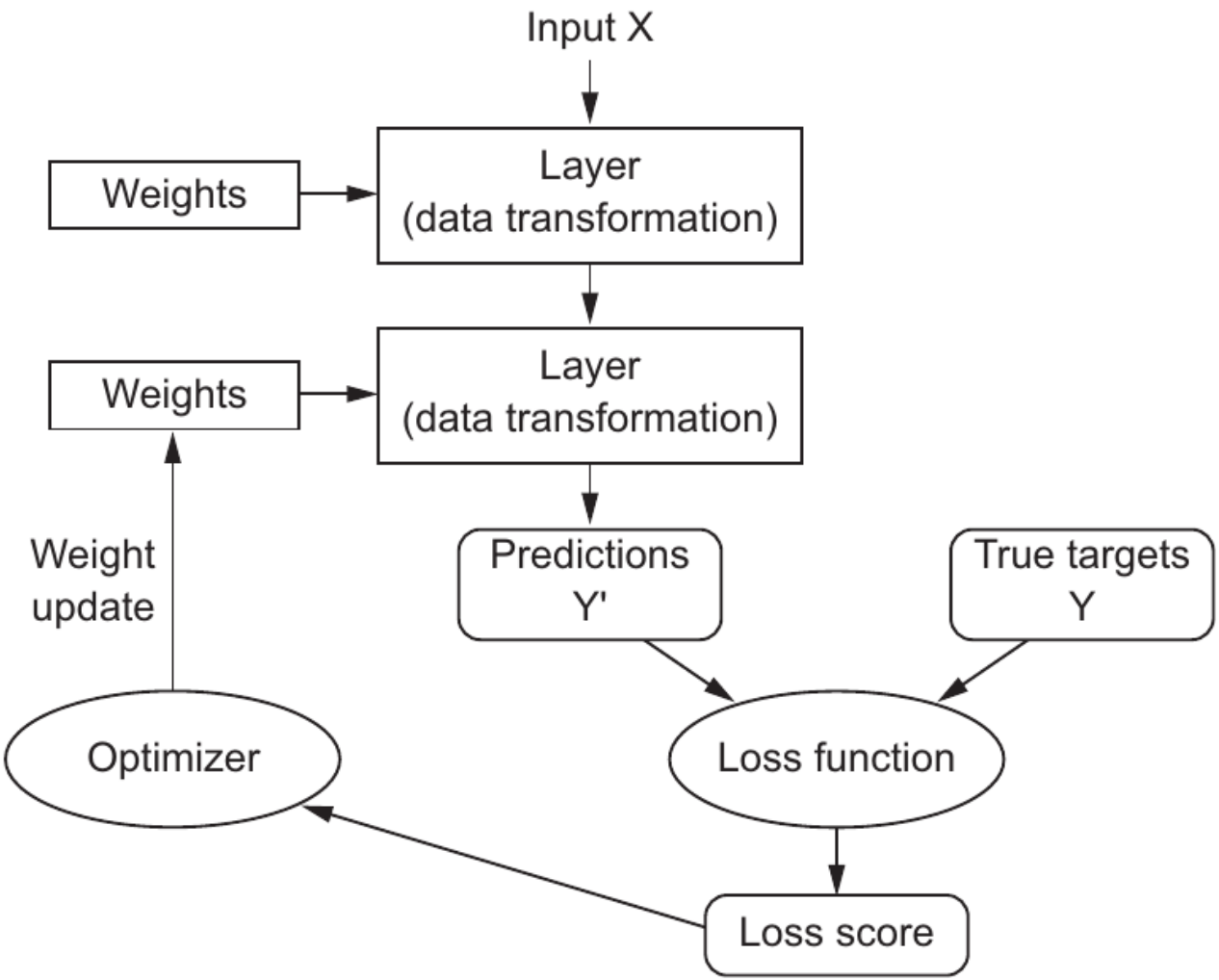
Arquitecturas y Capas Comunes

- **Capas Convolucionales (ConvNets o CNNs):** Componente clave en CNNs para datos de imagen, aprende jerarquías espaciales.
- **Capas Recurrentes (LSTM, GRU):** Usadas en RNNs para datos secuenciales, con celdas de memoria para retener información.
- **Capas Completamente Conectadas (Dense):** Capa de propósito general donde cada neurona se conecta con todas las neuronas de la capa anterior.
- **Arquitecturas Populares:**
  - CNNs para imágenes y datos espaciales (ej., ResNet, VGG).
  - RNNs/LSTMs para datos secuenciales (ej., texto, series temporales).
  - Transformers para self-attention y modelos de lenguaje grandes (LLMs)

Introducción a Redes Neuronales y Deep Learning

Ingredientes Clave del Deep Learning

Ingredientes Clave del Deep Learning



Función de Pérdida y Diseño de la Capa de Salida

- **Regresión:**
  - **Dimensión de salida es 1**.
  - **Sin Activación:** La salida coincide directamente con los valores finales del modelo.
  - **Error Cuadrático Medio (MSE):**

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

Función de Pérdida y Diseño de la Capa de Salida

- **Clasificación Binaria:**
  - **Dimensión de salida es 1**.
  - **Activación Sigmoide:** Mapea salidas a probabilidades entre 0 y 1:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

  - **Entropía Cruzada Binaria:**

$$\text{BCE} = -\frac{1}{N} \sum_{i=1}^{N} (y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i))$$

- **Clasificación Multiclase:**
  - **Dimensión de salida es C**.
  - **Activación Softmax:** Normaliza salidas a probabilidades de clase:

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

  - **Entropía Cruzada Categórica:**

$$\text{CCE} = -\frac{1}{N} \sum_{i=1}^{N} \sum_{c=1}^{C} y_{i,c} \log(\hat{y}_{i,c})$$

Monitorización de la convergencia

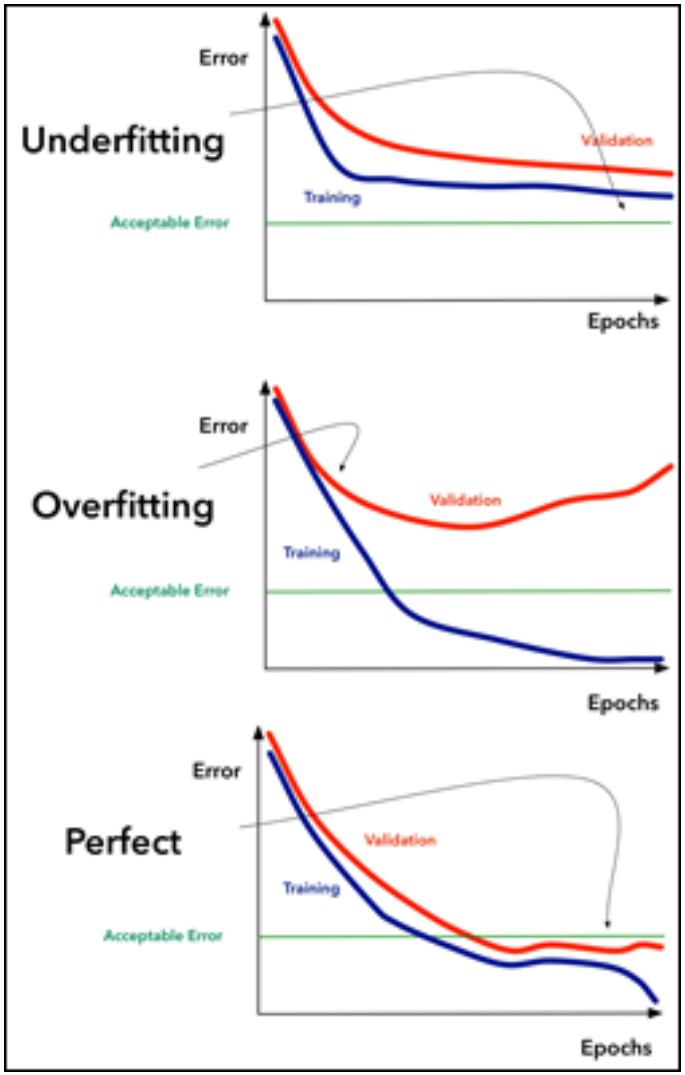


Imagen de https://histalk2.com/2018/12/05/machine-learning-primer-for-clinicians-part-7/

Ingredientes Clave del Deep Learning

Code Example: Problema de sobornos con redes neuronales

Entrenamiento de redes neuronales

Entrenamiento de redes neuronales

Grafos computacionales, AD y Backprop

Motivación: regresión con red neuronal de cuatro capas

Pérdida en un ejemplo:

$$\text{Loss}\left(x, y, \mathsf{V}_1, \mathsf{V}_2, \mathsf{V}_3, \mathsf{w}\right) = \left(\mathsf{w} \cdot \sigma\left(\mathsf{V}_3 \sigma\left(\mathsf{V}_2 \sigma\left(\mathsf{V}_1 \phi(x)\right)\right)\right) - y\right)^2$$

Descenso de gradiente:

$$\begin{aligned}
\mathsf{V}_1 &\leftarrow \mathsf{V}_1 - \eta \nabla_{\mathbf{V_1}} \text{Loss}\left(x, y, \mathsf{V}_1, \mathsf{V}_2, \mathsf{V}_3, \mathsf{w}\right) \\
\mathsf{V}_2 &\leftarrow \mathsf{V}_2 - \eta \nabla_{\mathbf{V_2}} \text{Loss}\left(x, y, \mathsf{V}_1, \mathsf{V}_2, \mathsf{V}_3, \mathsf{w}\right) \\
\mathsf{V}_3 &\leftarrow \mathsf{V}_3 - \eta \nabla_{\mathbf{V_3}} \text{Loss}\left(x, y, \mathsf{V}_1, \mathsf{V}_2, \mathsf{V}_3, \mathsf{w}\right) \\
\mathsf{w} &\leftarrow \mathsf{w} - \eta \nabla_{\mathsf{w}} \text{Loss}\left(x, y, \mathsf{V}_1, \mathsf{V}_2, \mathsf{V}_3, \mathsf{w}\right)
\end{aligned}$$

¿Cómo obtener el gradiente sin hacer el trabajo manual?

Grafos computacionales

$$\text{Loss}\left(x, y, \mathsf{V}_1, \mathsf{V}_2, \mathsf{V}_3, \mathsf{w}\right) = \left(\mathsf{w} \cdot \sigma\left(\mathsf{V}_3 \sigma\left(\mathsf{V}_2 \sigma\left(\mathsf{V}_1 \phi(x)\right)\right)\right) - y\right)^2$$

Definición: grafo computacional
Un grafo acíclico dirigido cuyo nodo raíz representa la expresión matemática final y cada nodo representa subexpresiones intermedias.

Objetivo: Calcular automáticamente gradientes mediante el algoritmo general de **backpropagation** (así funcionan PyTorch, TensorFlow y Jax).

Diferenciación Automática

```python
import torch
from torch.autograd import Variable

# Let's demonstrate the calculation of a simple derivative: y = x^2
x = Variable(torch.tensor([3.0]), requires_grad=True)
y = x**2
y.backward()

# The derivative of y with respect to x is dy/dx = 2x
print(f"Theoretical gradient: {2 * x.data}")
# Output: Theoretical gradient: tensor([6.])
print(f"Gradient from autograd: {x.grad}")
# Output: Gradient from autograd: tensor([6.])
```

Grafos Computacionales

Exercise: Construye el grafo computacional para...

- a y b son parámetros aprendibles.
- $c = a + b$
- $d = b + 1$
- $e = c \cdot d$

Grafos Computacionales

Exercise: Construye el grafo computacional para...

- a y b son parámetros aprendibles.
- $c = a + b$
- $d = b + 1$
- $e = c \cdot d$

Exercise: Evalúa el grafo si a=2 y b=1

Grafos Computacionales

Exercise: Construye el grafo computacional para...

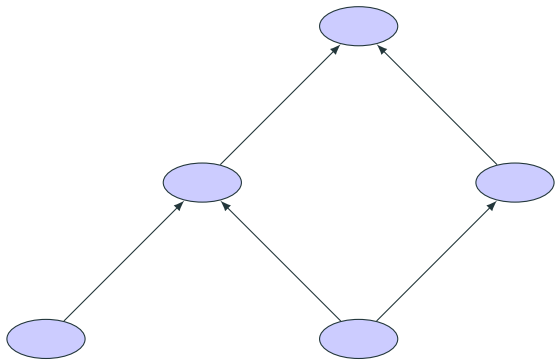
- a y b son parámetros aprendibles.
- $c = a + b$
- $d = b + 1$
- $e = c \cdot d$

Exercise: Evalúa el grafo si a=2 y b=1

Podemos evaluar expresiones en un **forward pass** pero, ¿cómo calculamos eficientemente las derivadas?

Diferenciación forward- y backward-mode

La **diferenciación forward-mode** y la **diferenciación backward-mode** (o backward-mode) son algoritmos para calcular derivadas eficientemente:



- Backward-mode es el algoritmo comúnmente usado en redes neuronales.
- **Backpropagation** es una aplicación especializada de backward-mode AD, adaptada para arquitecturas de redes neuronales:
  - Backprop calcula las derivadas de la función de pérdida respecto a todos los parámetros.
  - Backward-mode AD puede calcular derivadas de cualquier proceso computacional.

¿Por qué Backward-mode?

Exercise: Evalúa las derivadas usando forward-mode

Exercise: Evalúa las derivadas usando backward-mode

¿Por qué Backward-mode?

Exercise: Evalúa las derivadas usando forward-mode

Exercise: Evalúa las derivadas usando backward-mode

- En **forward-mode** obtenemos la derivada de **todas las salidas respecto a una entrada**.
- En **backward-mode** obtenemos la derivada de **una salida respecto a todas las entradas**.
  - ¡Por eso backward-mode se usa típicamente en ML!
  - En el **forward pass** calculamos los valores de los nodos, y en el **backward pass** calculamos las derivadas.

Backprop en redes neuronales de dos capas

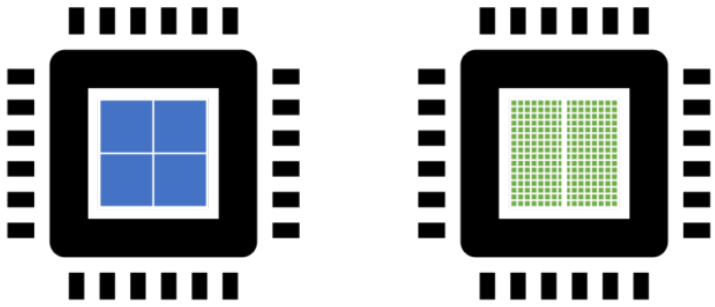
Exercise: Red neuronal de dos capas

Supongamos un único ejemplo de entrenamiento en un problema de regresión. La función de pérdida es MSE y la red neuronal se define como $y_{out} = \mathbf{w} \cdot \sigma\left(\mathbf{V}\phi(x)\right)$. Dibuja el grafo computacional y calcula los gradientes usando backprop. Pista: $\frac{d\sigma(x)}{dx} = \sigma(x)\left(1 - \sigma(x)\right)$

Entrenamiento de redes neuronales

Infraestructura Moderna de Entrenamiento: GPUs

CPUs vs. GPUs



CPU	GPU
Central Processing Unit	Graphics Processing Unit
4-8 Cores	100s or 1000s of Cores
Low Latency	High Throughput
Good for Serial Processing	Good for Parallel Processing
Quickly Process Tasks That Require Interactivity	Breaks Jobs Into Separate Tasks To Process Simultaneously
Traditional Programming Are Written For CPU Sequential Execution	Requires Additional Software To Convert CPU Functions to GPU Functions for Parallel Execution

Uso de GPUs y CUDA en redes neuronales

- **¿Por qué GPUs?**
  - **Alto Paralelismo:** Las redes neuronales involucran grandes multiplicaciones matriciales, que pueden paralelizarse efectivamente.
  - **Alto Rendimiento:** Las GPUs procesan miles de cálculos simultáneamente, ideal para las necesidades computacionales del deep learning.
  - **Hardware Especializado:** Las GPUs están diseñadas para manejar operaciones matemáticas repetitivas más eficientemente que las CPUs.
- **CUDA (Compute Unified Device Architecture):**
  Plataforma de NVIDIA que permite el control directo de operaciones GPU, usando *kernels personalizados* y *gestión eficiente de memoria*.
- **Aceleración con CUDA:**
  Bibliotecas como `cuDNN` proporcionan rutinas optimizadas (ej., convoluciones) para deep learning, reduciendo significativamente el tiempo de entrenamiento.
- **Beneficios:**
  Entrenamiento más rápido, menor latencia en inferencia y escalabilidad para grandes conjuntos de datos.

Entrenamiento de redes neuronales

Mini-batch SGD y Optimizadores

Mini-batch Stochastic Gradient Descent

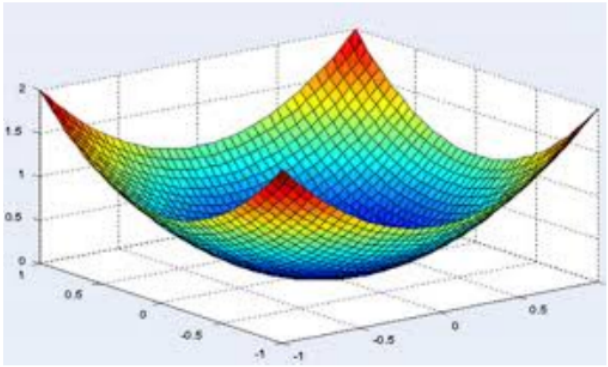
- Restricciones de memoria en redes neuronales/Deep Learning:
  - Parámetros del modelo
  - Activaciones (calculadas en el forward pass y necesarias para backprop)
  - Gradientes (backward pass)
- Enfoques de entrenamiento:
  - **Gradient Descent (GD, también llamado Batch GD)**: Todos los datos a la vez $\Rightarrow$ Imposible con datos grandes y redes grandes.
  - **Stochastic GD (SGD)**: Una muestra a la vez $\Rightarrow$ reduce el uso de memoria pero aumenta la varianza en las estimaciones del gradiente.
  - **Mini-batch SGD**: Lo mejor de ambos mundos.
    - Procesa un subconjunto (mini-batch) de datos en cada paso, permitiendo actualizaciones más estables que SGD usando menos memoria que GD completo.
    - Tamaños típicos de batch son 32, 64, 128, o hasta 256, dependiendo del tamaño del modelo y la memoria disponible.

Entrenamiento de redes neuronales

Inicialización de Pesos, Optimizadores

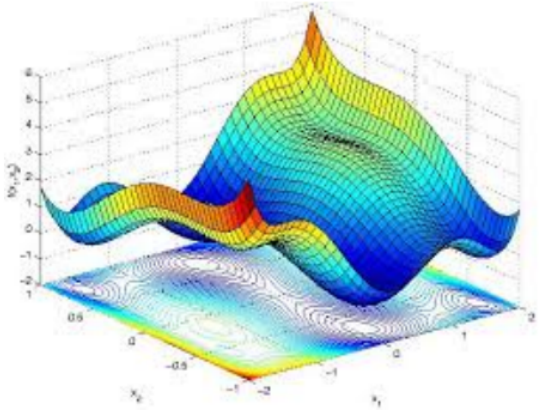
La Optimización de redes neuronales es Difícil

Linear predictors



(convex)

Neural networks



(non-convex)

La optimización es difícil:

- La optimización de redes neuronales es no convexa, lo que lleva a múltiples mínimos locales y puntos de silla.
- **Gradientes que Desaparecen o Explotan:** Los gradientes pueden volverse demasiado pequeños (desaparecen) o demasiado grandes (explotan), ralentizando o desestabilizando el entrenamiento.

La Optimización de redes neuronales es Difícil

Las redes neuronales fueron abandonadas por un tiempo hasta que se desarrollaron varias técnicas para entrenar modelos profundos:

1. **Mejor Inicialización**
   - Xavier/Glorot
   - He initialization
2. **Optimizadores Avanzados**
   - SGD con Momentum
   - Métodos adaptativos como **Adam**

La Optimización de redes neuronales es Difícil

Exercise: Inicialización en Keras

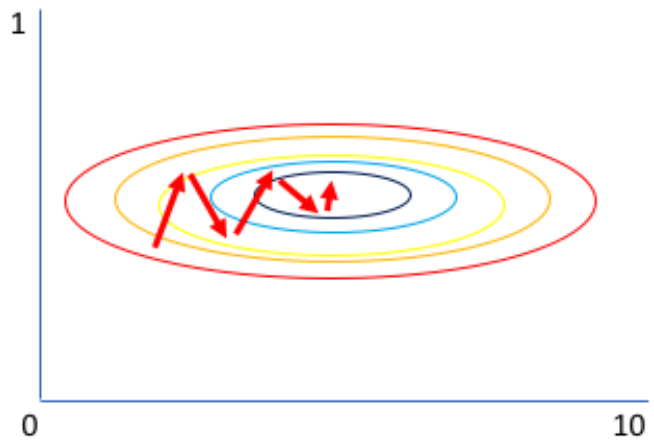
Lee la API de Keras para aprender cómo usar diferentes métodos de inicialización en las capas Dense. ¿Cuál es el método por defecto?

Code Exercise: Cambiando arquitectura, optimizador e hiperparámetros en el problema de los sobornos

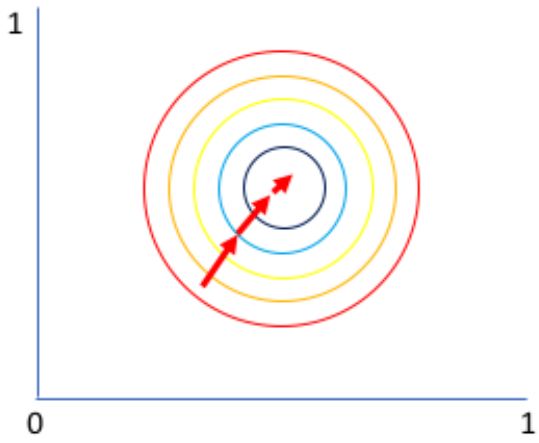
Entrenamiento de redes neuronales

Regularización y Estabilidad

Why normalize?



Gradient of larger parameter dominates the update



Both parameters can be updated in equal proportions

Técnicas de estabilización: Batch Normalization

- Normaliza la entrada a una capa de la red.
- **Beneficios:**
  - Acelera el entrenamiento y mejora la convergencia del modelo.
  - Permite tasas de aprendizaje más altas, mitigando problemas como gradientes que desaparecen/explotan.
- Dos pasos principales:
  1. **Normalización:** Para una entrada x:

     $$\hat{x} = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

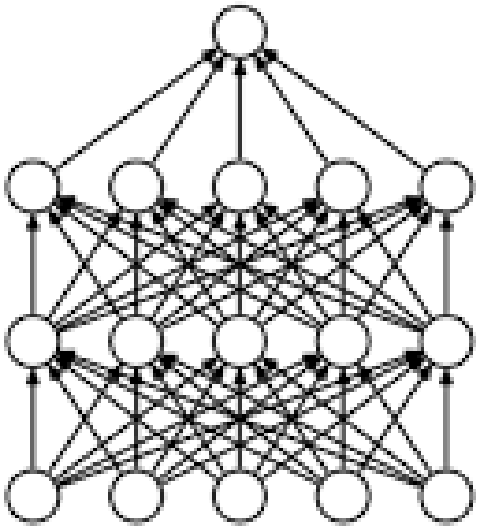
     donde: μ es la media del mini-batch, σ^2 es la varianza del mini-batch, ϵ es una pequeña constante para estabilidad numérica.
  2. **Escalado y Desplazamiento:** Después de la normalización, las salidas se escalan y desplazan para permitir que el modelo aprenda la representación óptima:

     $$y = \gamma\hat{x} + \beta$$

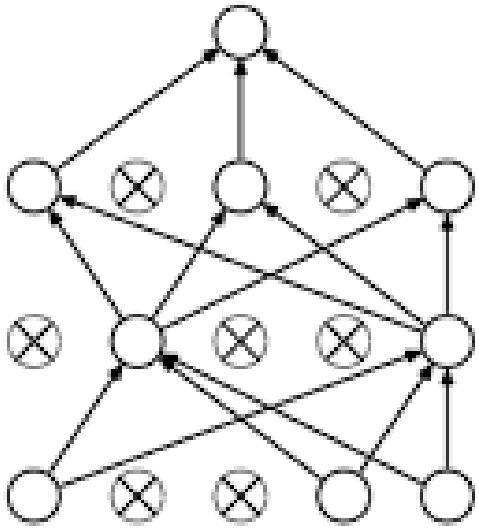
     donde: γ es el parámetro de escala, β es el parámetro de desplazamiento.

Técnicas de Regularización: Dropout

- **Regularización de Pesos:**
  - **Regularización L1:** Fomenta la dispersión penalizando la suma absoluta de pesos.
  - **Regularización L2:** Añade una penalización proporcional al cuadrado de los pesos, fomentando pesos más pequeños (**weight decay**).
- **Dropout:**
  - Desactiva aleatoriamente neuronas durante el entrenamiento para reducir el sobreajuste.
  - Ayuda a prevenir la co-adaptación de neuronas, haciendo la red más robusta.



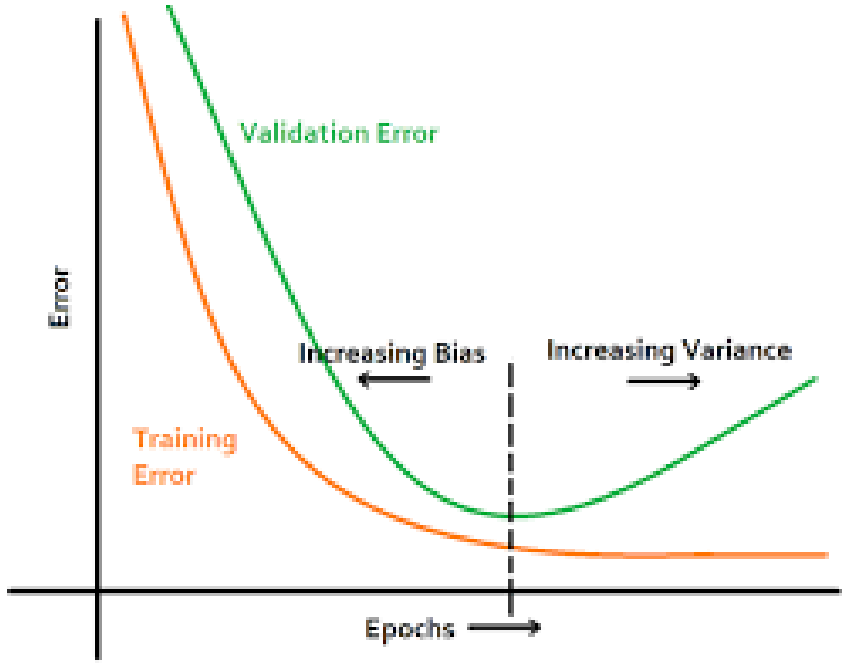
(a) Standard Neural Net



(b) After applying dropout.

Técnicas de Regularización: Early Stopping

- **Early Stopping:**
  - Monitoriza el rendimiento de validación para detener el entrenamiento cuando comienza el sobreajuste.
  - Ahorra tiempo de cómputo y previene la degradación del rendimiento del modelo.



Ejercicios con Problemas Reales

Code Exercise: PetFinder.my

Code Exercise: MNIST