Introducción Keras
&
Funciones de Activación

U-TAD

Introducción a Keras

- Librería de Deep Learning de alto nivel que permite construir, entrenar, evaluar y ejecutar todo tipo de redes neuronales artificiales (profundas) de manera sencilla
- Desarrollada por Francois Chollet y publicada como open source en 2015
- Su API está inspirada en otras librerías como Scikit-Learn
- Históricamente, Keras ha soportado diferentes backends para la construcción a bajo nivel de las redes neuronales: Tensorflow, Microsoft Cognitive o Theano
- Con el lanzamiento de TensorFlow 2.0 el multi-backend de Keras ha sido dicontinuado. En la actualidad, Keras utiliza como único backend TensorFlow.

Keras APIs

- **Sequential API**: Un modelo secuencial es apropiado para una simple pila de capas donde cada capa tiene exactamente un tensor de entrada uno de salida (recibe una matriz de datos de entrada y devuelve otra matriz de datos, 95% de los casos y es la que vamos a ver)
- La Sequential API **no** es apropiada cuando:
  - El modelo tiene múltiples entradas o múltiples salidas (múltiples datasets)
  - Cualquiera de sus capas tiene múltiples entradas o múltiples salidas
  - Necesitas compartir las capas
  - Quieres una topología no lineal (por ejemplo, una conexión residual, un modelo de varias ramas: neuronas que hacen conexiones de feedbkack)
- Para los casos en los que se requiera alguna de las características anteriores, se utiliza la **Functional API** de Keras

Caso Práctico: detección de números

Implementando una RNA con Keras.ypynb

Pasos en la implementación:

1. Define tu conjunto de datos de entrenamiento: vectores de entrada y de salida (tensores)
2. Define la arquitectura de la Red Neuronal Artificial (hiper-parámetros)
3. Configura el proceso de aprendizaje mediante la selección de una función de error, una función de optimización y diferentes métricas para monitorizar el proceso
4. Entrena la RNA con tu conjunto de datos de entrenamiento mediante el uso del método **fit()**

Caso Práctico: Detección de transacciones fraudulentas

Límite de decisión de una RNA profunda.ipynb

El objetivo no es resolverlo, que podría hacerse con técnicas clásicas de ML, sino comprobar cómo se construye el límite de decisión.

En DL, normalmente utilizaremos el 90% del conjunto de datos para entrenamiento, el 5% para test y el otro 5% para validación.

Caso Práctico: Regresión con Keras

Regresión con Keras.ipynb

Al ser un dataset pequeño, sería más interesante hacerlo con ML, pero nos sirve para ver un ejemplo de regresión.

Carga el dataset con:

```
from tensorflow.keras import datasets

bostos_housing = datasets.boston_housing
```

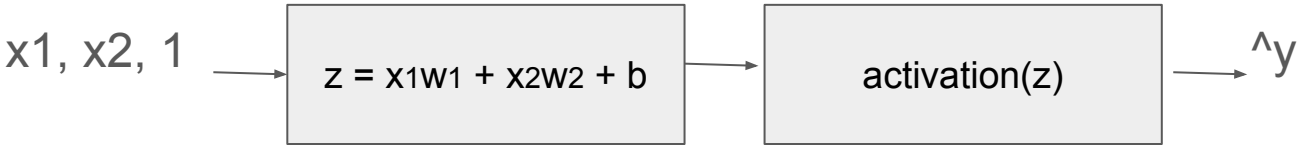
Ejercicio Práctico: Clasificación de audio con Keras

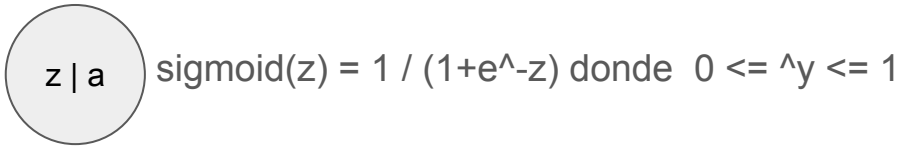
Ejercicio BB: Clasificación de sonido con Keras

https://www.kaggle.com/kongaevans/speaker-recognition-dataset/

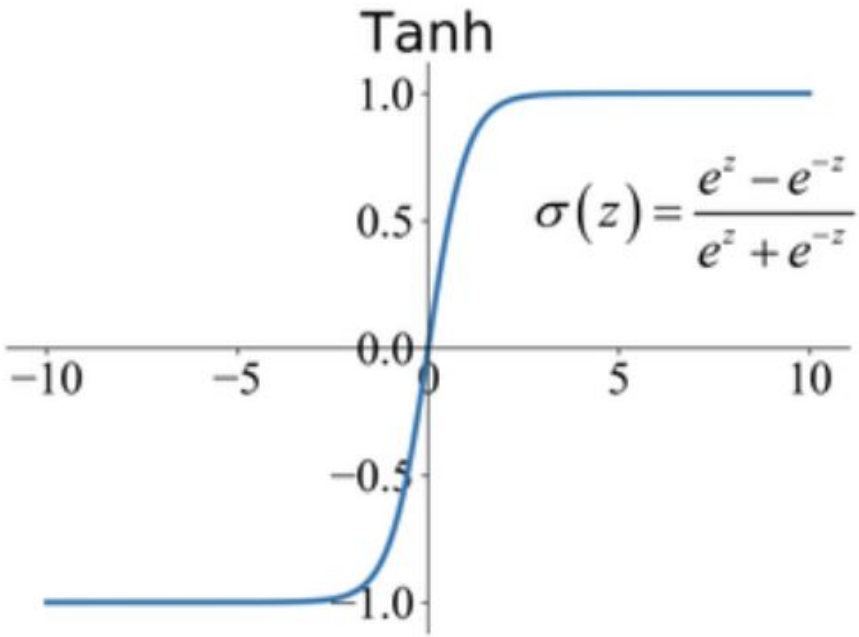
Funciones de activación: introducción

Además de las vistas anteriormente, como sigmoid o softmax, profundizaremos en las funciones de activación más modernas y que son las que se usan actualmente.





Función de activación tanh

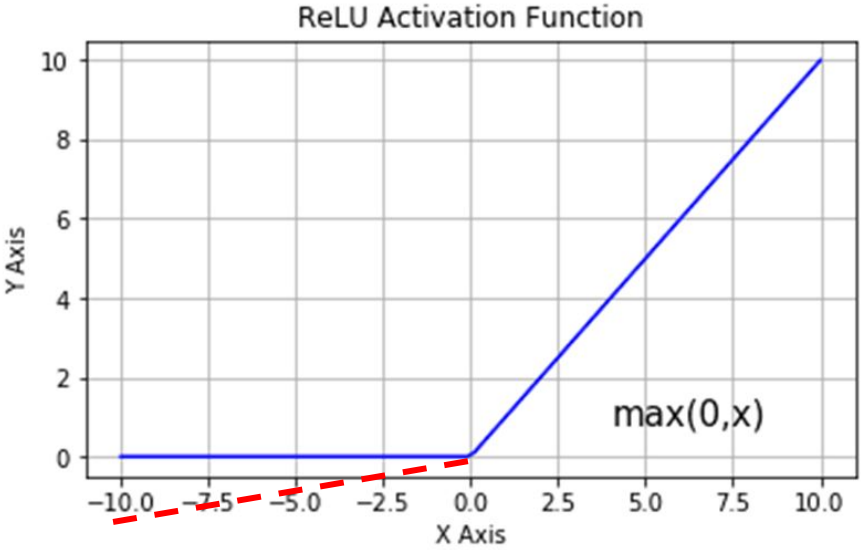


La diferencia fundamental es que vamos a poder proporcionar valores negativos: *-1 <= tanh(z) <= 1 (y corta en 0 en vez de en 0.5)*

La usaremos en las hidden layers, pero seguiremos usando la sigmoide para la output layer (para clasificación binaria)

Como desventaja de sigmoid y tanh:
Si Z es muy grande (o muy pequeño), se aproximará a 1 sin llegar a 1, por lo que la pendiente será muy pequeña provocando que al computar Gradient Descent (w = w - n*dJ/dw), esta derivada de error será un valor muy pequeño, lo que implica que la actualización será muy lenta hasta que la pendiente sea más grande. Lo que implica que el entrenamiento sea lento mientras sea Z grande o pequeño. Se resuelve con ReLU

Función de activación: ReLU(z)



Si z <= 0 entonces ^y = 0,

y si z > 0 entonces y = z

Ahora la pendiente se de 1 para z>0, pero si z<=0 entonces no tenemos pdte, por lo que la derivada será siempre 0.

Por lo que se puede aplicar una variación tal que tenga una pequeña pendiente.

leaky relu = max(0.01*z, z)

Como las hidden layers suelen devolver valores positivos, tampoco impacta mucho usar relu o leaky relu.

Norma general:

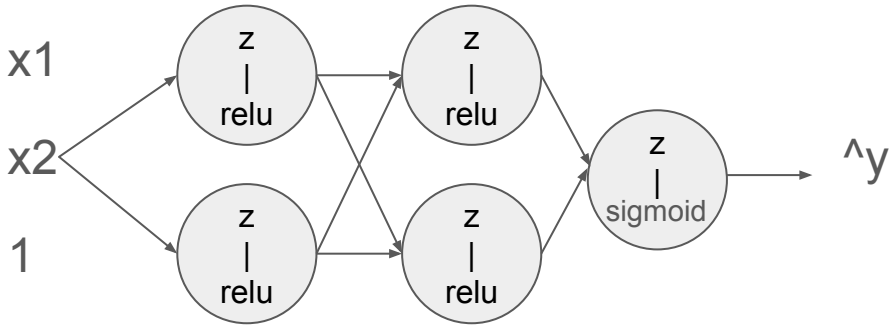
Clasif. binaria: ReLU (o leaky Relu) -> hidden layers
Sigmoid -> output layer

Clasif. multiclase: ReLU -> hidden layers
Softmax -> output layer

Entrenamiento con diferentes funciones de activación

Forward Propagation y Backward Propagation

- El cambio de la función de activación afecta al proceso de entrenamiento de la red neuronal dado que iremos derivando respecto a las func. de activación



Solo cambia la ecuación ***da***, pero el proceso de cálculo es el mismo que vimos.

Caso práctico: Visualizando las funciones de activación

Visualización de las funciones de activación.ipynb

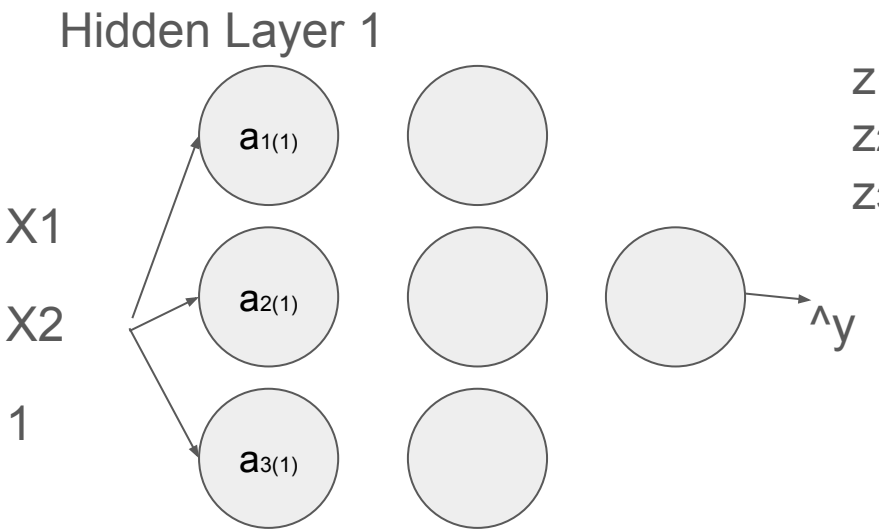
Probad el entrenamiento con “sigmoid”, “tanh” y “relu” para las hidden layers:

Si usáramos la función sigmoide, cuando se tienen varias capas, o tarda mucho en converger (encontrar el valor óptimo de los parámetros) o directamente no lo encuentra. Por lo que no se usa en redes profundas para las hidden layers (se usa tanh o ReLU)

Con tanh, además de converger más rápido, consigue un límite de decisión más preciso.

Con relu (recomendada), es todavía más rápida minimizando el error y consigue mayor precisión. ReLU sortea un núcleo de ejemplos negativos (legítimos) construyendo una función más compleja.

¿Por qué utilizar una función de activación?



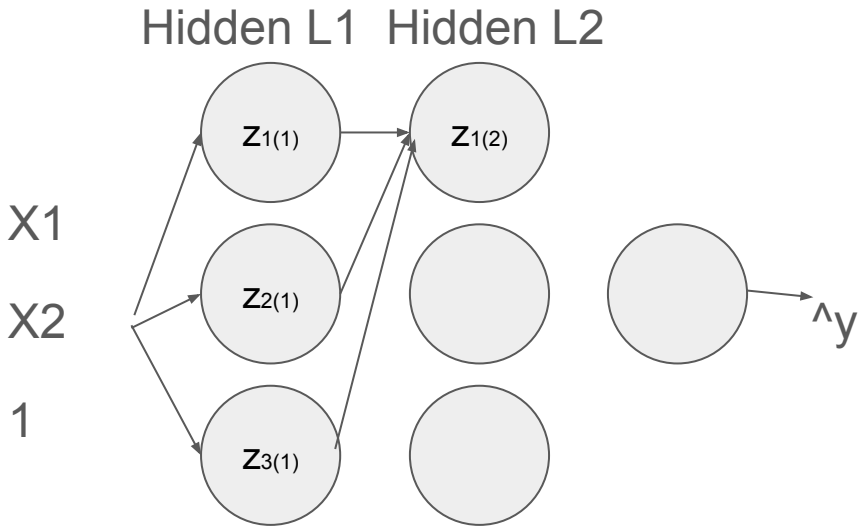
$z_{1(1)}=x_1w_{11(1)}+x_2w_{21(1)}+b_{1(1)}$ $a_{1(1)}=\text{sigmoid}(z_{1(1)})$

$z_{2(1)}=x_1w_{12(1)}+x_2w_{22(1)}+b_{2(1)}$ $a_{2(1)}=\text{sigmoid}(z_{2(1)})$

$z_{3(1)}=x_1w_{13(1)}+x_2w_{23(1)}+b_{3(1)}$ $a_{3(1)}=\text{sigmoid}(z_{3(1)})$

Aplicamos “z”, que es una función lineal, y después “a” que no lo es, como sigmoide (o tanh o ReLU). Si eliminamos “a” tal que el output de “z” sería el input de las neuronas de la siguiente capa, lo que sucede es que la función final que nosotros vamos a computar va a ser una función lineal, construiría un límite de decisión lineal y, por tanto, no vamos a poder resolver problemas que no sean linealmente separables.

¿Por qué utilizar una función de activación?



$$z_{1(1)} = x_1 w_{11(1)} + x_2 w_{21(1)} + b_{1(1)}$$
$$z_{2(1)} = x_1 w_{12(1)} + x_2 w_{22(1)} + b_{2(1)}$$
$$z_{3(1)} = x_1 w_{13(1)} + x_2 w_{23(1)} + b_{3(1)}$$

$$z_{1(2)} = z_{1(1)} w_{11(2)} + z_{2(1)} w_{21(2)} + z_{3(1)} w_{31(2)} + b_{1(2)} =$$
$$= w_{11(2)}(x_1 w_{11(1)} + x_2 w_{21(1)} + b_{1(1)}) + w_{21(2)}(\ldots)$$
$$= x_1 w_{11(1)} w_{11(2)} + x_2 w_{21(1)} w_{11(2)} + b_{1(1)} w_{11(2)} +$$
$$= \quad x_1 w_1' \quad + \quad x_2 w_2' \quad + \quad b'$$

Básicamente, cuando componemos un conjunto de funciones lineales, lo que obtenemos es otra función lineal. Lo que provoca que la función que obtengamos al final sea una función lineal y, por tanto, solo construya una decisión lineal, lo que no nos permitiría resolver problemas complejos. Por esto usaremos funciones de activación no lineales.

Caso práctico: ¿Por qué utilizar una función de activación?

¿Por qué utilizar una función de activación?.ipynb

Probad con *activation=None* comprobando que crea un límite de decisión lineal

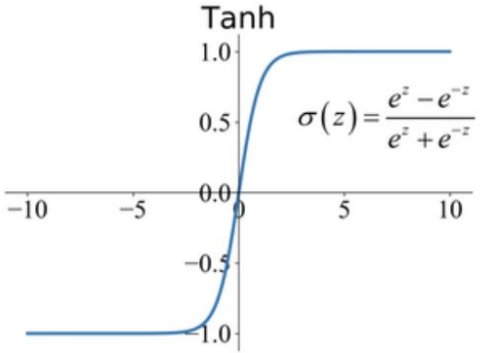
¿Por qué utilizar una función de activación diferenciable?

Donde diferenciable, para nosotros, sea únicamente derivable.

Todas las funciones de activación vistas anteriormente son diferenciables.

dtanh/dz = d/dz (e^z - e^-z / e^z + e^-z) =

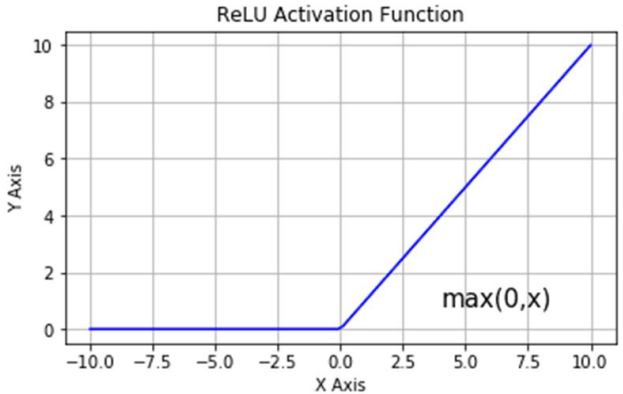
= 1 - tanh^2



ReLU(z) = max(0,z)

drelu/dz = d/dz(max(0,z)) = 0 si z < 0

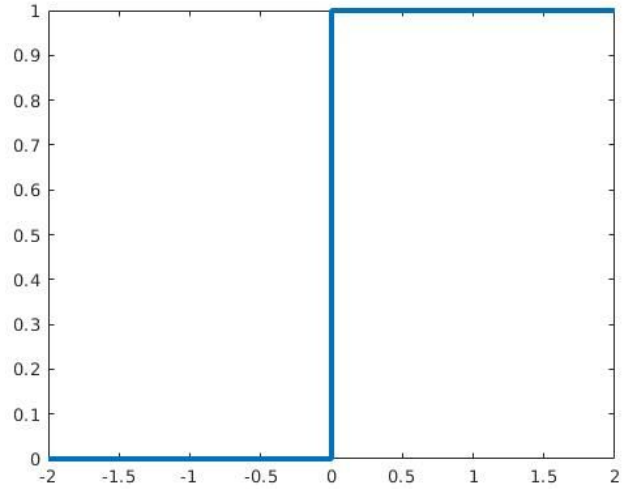
1 si z > 0



¿Por qué utilizar una función de activación diferenciable?

heaviside(z) = 0 si z < 0

= 1 si z >= 0



No es una función derivable (no hay pdte.)

dheaviside/dz = 0 si Z>0 y si Z<0

Si usáramos heaviside como función de activación,

durante el proceso de backpropagation:

$w_{1(1)} = w_{1(1)} - n\cancelto{0}{dL/dw_{1(1)}}$ Tendremos que calcular da/dz, lo cual sería 0 que multiplicará al resto de derivadas y obtendremos como resultado 0, por lo que la actualización de $w_{1(1)}$ no se realizará (siempre será igual y no funcionaría).

Ejercicio: Clasificación de sentimientos

Clasificación de sentimientos.ipynb

El objetivo es, a partir de un texto (reseña), debemos tratar de predecir el sentimiento que tenía esa persona cuando escribía el texto.

Recibimos 25k críticas de películas etiquetados por un sentimiento positivo o negativo (la entrada es texto)

Notas:

- Modifica el *learning rate* para obtener mejores resultados.
- Prueba con las funciones de activación tanh y relu para quedarte con la mejor