Introduction to Deep Learning & Neural Networks with keras

Week 2

Training a Neural Networks

1. Gradient Descent

El gradiente descendiente es un algoritmo de optimización iterativo para encontrar el mínimo de una función. Para encontrar el mínimo de una función usando el descenso de gradiente, empezamos con un valor inicial aleatorio de w. Vamos a llamarlo w0, suponemos que es igual a 0.2, y tenemos que movernos hacia el punto verde que es el mínimo de la función parábola, que en este caso es w=2.

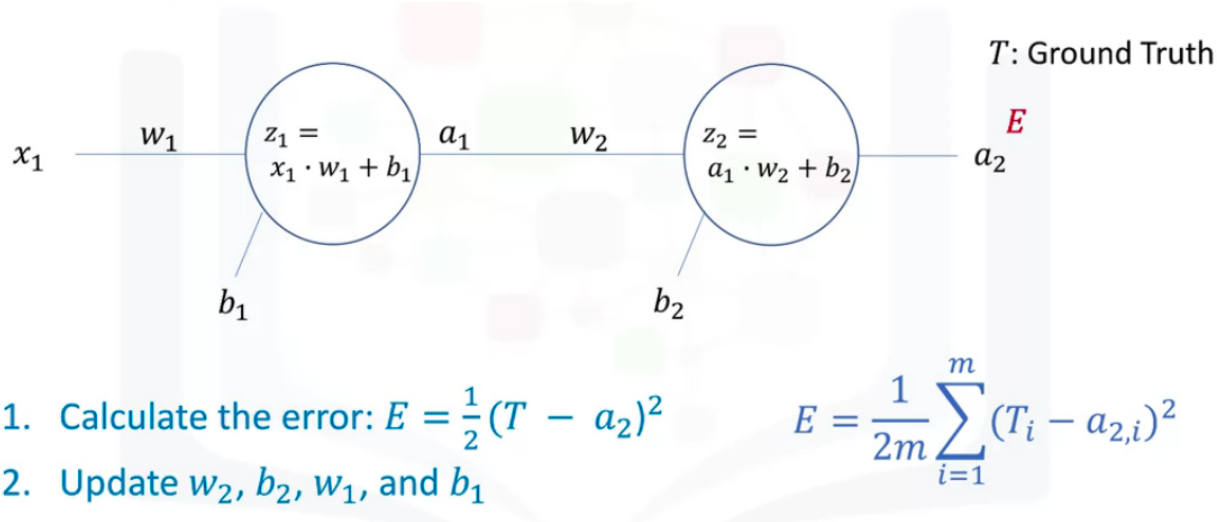
Para determinar en qué dirección moverse, calculamos el gradiente de la función de pérdida en el valor actual de w, que es 0.2. El gradiente está dado por la pendiente de la tangente en w = 0.2, y luego la magnitud del paso es controlada por un parámetro llamado la tasa de aprendizaje.

Si la tasa de aprendizaje es alta podemos perder el mínimo. Mientras que si es pequeña nos puede llevar mucho tiempo encontrar el valor mínimo.

2. Backpropagation

El algoritmo de Backpropagation lo podemos resumir en los siguientes puntos:
Inicializamos los pesos y sesgos a valores aleatorios. Luego, repetimos iterativamente los siguientes pasos:

1. Calcular la salida de red utilizando la propagación directa.
2. Calcular el error entre el ground truth y la salida estimada o pronosticada de la red.
3. Actualizar los pesos y los sesgos a través de la Backpropagation. Repetir los pasos anteriores hasta que se alcance el número mínimo de interaciones o ephocs, o bien el error entre el ground truth y la salida calculada esté por debajo del umbral predefinido.



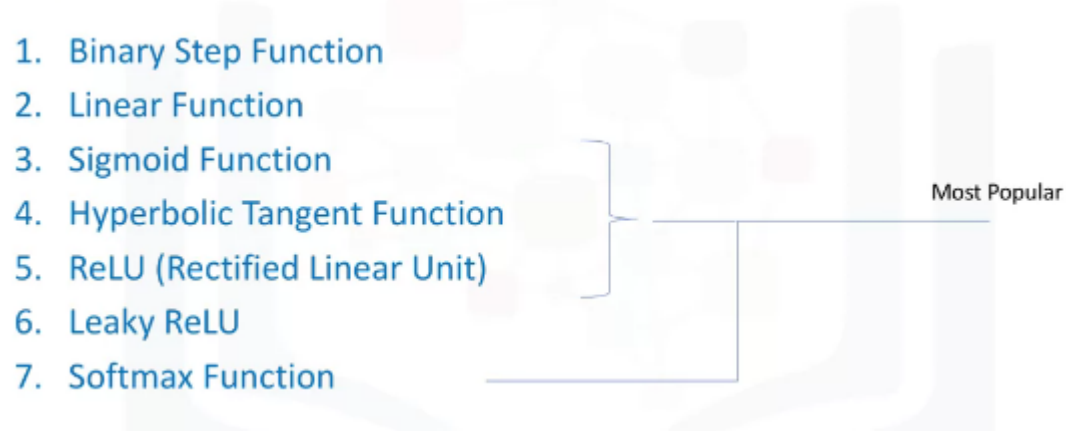
3. Vanishing Gradient

Al utilizar la función sigmoide todos los valores intermedios están entre 0 y 1. Por tanto, cuando se utiliza Backpropagation, multiplicamos factores que son menores a 1, y entonces sus gradientes tienden a hacerse más pequeños a medida que seguimos moviéndonos hacia atrás en la red. Esto significa que las capas de neuronas anterioes aprenden lentamente en

comparación con las capas de neuronas posteriores. Las capas anteriores de la red son las más lentas de entrenar. El resultados es un proceso de entrenamiento lento y una predicción que se ve comprometida. En consecuencia, esta es la razón por la que no usamos la función sigmoide o funciones similares como funciones de activación, porque son propensas al vanishing gradient.

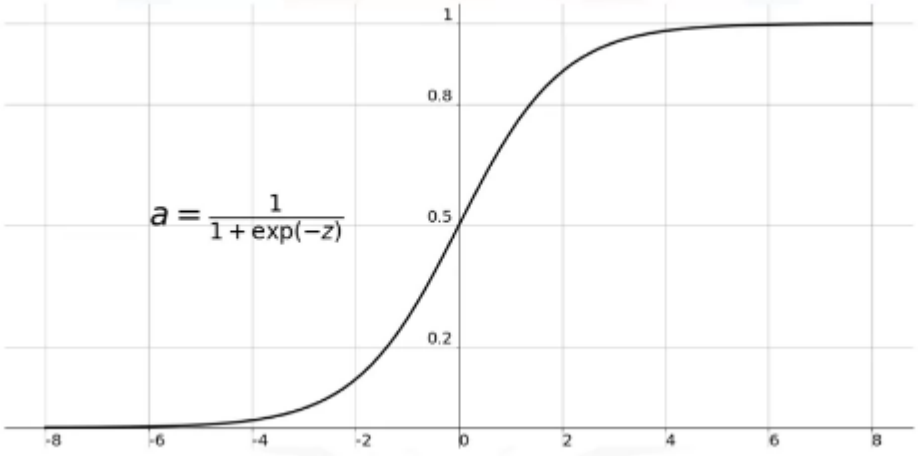
4. Activation Function

Hay siete tipos de funciones de activación que se pueden usar al construir una red neuronal.



A continuación se detallan las más utilizadas:

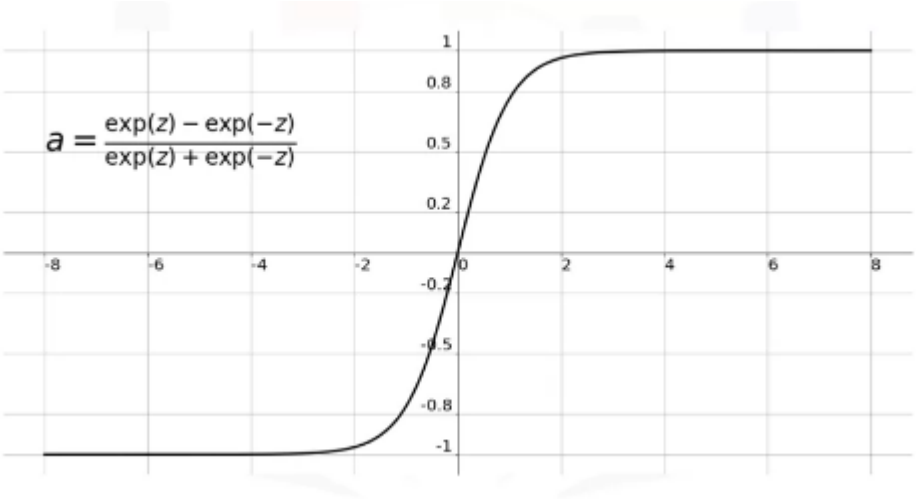
- Función sigmoide:



En z = 0, a es igual a 0.5 y cuando z es un número positivo muy grande, a es cerca de 1, y cuando z es un número muy grande negativo, a está cerca de cero. Las funciones sigmoides solían ser ampliamente utilizadas como funciones de activación en las capas ocultas de una red neuronal. Sin embargo, como puede ver la función es bastante plana más allá de la región +3 y -3. Esto significa que una vez que la función cae en esa región, los gradientes
se vuelven muy pequeños. Esto da como resultado el problema de gradiente de fuga **y a medida que los gradientes se acercan a 0, la red realmente no aprende**

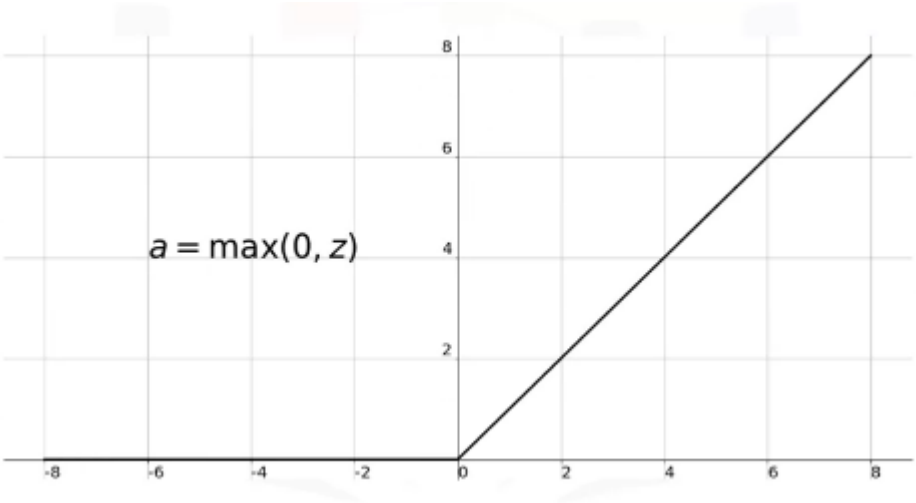
Otro problema con la función sigmoide es que los valores solo varían de 0 a 1. Esto significa que la función sigmoide no es simétrica alrededor del origen

- Función hiperbólica tangente.



En realidad sólo es una versión escalada de la función sigmoide, pero a diferencia de la función sigmoide, es simétrica sobre el origen.
Va de -1 a +1. Sin embargo, aunque supera la falta de simetría de la función sigmoide, también conduce al problema del gradiente de fuga en redes neuronales muy profundas.

- Función rectificada de unidad lineal, o RelU



Es la función de activación más utilizada al diseñar redes hoy en día. Además de ser no lineal, la principal ventaja de usarla, es que no activa
todas las neuronas al mismo tiempo. De acuerdo con la trama aquí, si la entrada es negativa se convertirá a 0, y la neurona no se activa.
Esto significa que a la vez, sólo unas pocas neuronas se activan, haciendo que la red sea escasa y muy eficiente. Además, la función ReLU
fue uno de los principales avances en el campo del aprendizaje profundo que condujo a superar el problema de gradiente de fuga.

- Función softmax.

$$a_i = \frac{e^{z_i}}{\sum_{k=1}^{m} e^{z_k}}$$

Es un tipo de función sigmoide, pero es útil cuando estamos tratando de manejar problemas de clasificación.
La función softmax se utiliza idealmente en la capa de salida del clasificador donde estamos tratando de obtener las probabilidades para definir la clase de cada entrada. Por lo tanto, si una red con 3 neuronas en la capa de salida tiene de salidas [1.6, 0.55, 0.98] entonces con una función de activación softmax, las salidas se convierten a [0.51, 0.18, 0.31]. De esta manera, es más fácil para nosotros clasificar un punto de datos dado y determinar a qué categoría pertenece.

- En conclusión, el sigmoide y las funciones tangente se evitan en muchas aplicaciones hoy en día ya que pueden conducir al problema del gradiente de fuga.
- La función ReLU es la función que es ampliamente utilizada hoy en día, y es importante tener en cuenta que sólo se usa en las capas ocultas.
- Finalmente, al construir un modelo, puede comenzar con la función ReLU y luego puede cambiar a otras funciones de activación si fuera necesario, porque la función ReLu no condujera a un buen rendimiento.