ENTRENAMIENTO DE UNA RED NEURONAL

VISIÓN GENERAL

El aprendizaje en una red neuronal, consiste en aprender sus parámetros (pesos y sesgos) observando muchas muestras y su etiqueta correspondiente durante el proceso de entrenamiento. Recordemos que una capa aplica a sus datos de entrada los pesos y sesgos que tienen guardadas sus neuronas, por lo tanto es clave encontrar los valores adecuados de estos parámetros, este es el objetivo del entrenamiento.

Los parámetros de inicializan con valores aleatorios(también hay otras formas de inicialización) a partir de aquí su valor se va ajustando gradualmente.
Para conseguir esto, entran en juego diferentes actores, como la ***loss function*** *(función de pérdida), el objetivo de esta función es medir cuán lejos están las predicciones del valor real, es decir la función de pérdida coge las predicciones de la red y las compara con el valor verdadero de la etiqueta y calcula el error cometido.*

Este valor es la clave de todo el proceso, el objetivo es aprovechar esta medida de error como señal de retroalimentación para ajustar el valor de los parámetros en una dirección que disminuya el error, aquí es donde entra en juego ***el optimizador****.*
El optimizador implementa la retropropagación del error para ajustar los parámetros.

El resumen que podemos hacer de la visión general es que,en general los pesos se inicializan con valores aleatorios, con lo cual el error es alto porque la red empieza haciendo estimaciones aleatorios, poco a poco, el optimizador va retropropagando el error y ajustando los pesos para ir disminuyendo el error cometido.
Este es a alto nivel, el proceso de entrenamiento que repetido un número suficiente de veces.produce valores de pesos que minimizan el error cometido.

FORWARD AND BACKWARD PROPAGATION

La primera fase, forward propagation se da cuando los datos de entrenamiento cruzan toda la red neuronal para calcular sus predicciones, es decir los datos de entrada pasan a través de toda la red neuronal, de tal manera que todas las neuronas apliquen sus transformaciones a la información que reciben de las neuronas de la capa anterior y la propaguen a las neuronas de la capa siguiente, cuando los datos han cruzado todas las capas y todas las neuronas han aplicado sus transformaciones se obtiene una predicción.
A continuación con la loss function, compararemos la predicción con el valor real y calcularemos el error que hemos cometido, como el objetivo es minimizar este error se va a ir propagando hacia atrás para que las neuronas vayan ajustando sus pesos y sesgos, esta propagación hacia atrás se conoce como Backward propagation, veamos en más detalle en qué consiste.

Inicialmente, partiendo desde la capa de salida, la información de cuánto error se ha cometido se propaga hacia todas las neuronas de la capa anterior (si la capa de salida es la n, propagamos el error a las neuronas de la capa n-1).Sin embargo, las neuronas de esta capa oculta (la n-1), solo reciben una fracción de la señal total del error cometido, que se basa en la contribución relativa que

haya aportado cada neurona a la salida original, de acuerdo con los pesos de esta, este proceso se repite hasta que llegamos a la primera capa de la red.
Una vez que la señal de error ha sido propagada por toda la red, podemos ajustar los pesos, para ello usaremos una técnica llamada descenso de gradiente, esta técnica va cambiando los pesos en pequeños incrementos, con la ayuda del cálculo de la derivada(o gradiente) de la función de pérdida, lo cual nos permite ver en qué dirección "descender" hacia el mínimo global. Esto lo va haciendo en general por lotes de datos (después veremos la importancia de que el proceso se haga por lotes de datos de entrada y no individualmente) en las sucesivas iteraciones.

DESCENSO DE GRADIENTE

Es un algoritmo de optimización genérico capaz de encontrar soluciones óptimas para una amplia gama de problemas. La idea general del descenso de gradiente, es ajustar parámetros de forma iterativa para minimizar una función de pérdida.
El descenso de gradiente, se aprovecha del hecho de que todas las operaciones utilizadas en la red neuronal son derivables y calculan el gradiente de la función de pérdida con respecto a los parámetros de la red neuronal. Este hecho permite mover los parámetros en la dirección opuesta al gradiente, disminuyendo así el error, ya que el gradiente por defecto apunta hacia el sentido en el que se incrementa el valor de la función de pérdida, por lo tanto si se usa el negativo del gradiente podemos conseguir el sentido en el que tendemos a reducir la función de pérdida.

El descenso de gradiente, usa la primera derivada(gradiente) de la función de pérdida.
Recordemos que el gradiente nos da la pendiente de una función en ese punto. El proceso consiste en encadenar la derivada de la función de pérdida con las derivadas de cada capa de la red neuronal, es decir aplicar la regla de la cadena.
En realidad, el uso de la regla de la cadena para calcular los valores del gradiente de una red neuronal, es lo que da lugar al algoritmo de backpropagation.
La retropropagación, comienza con el valor de pérdida final y avanza hacia atrás, de las capas superiores (las más cercanas a la salida) a las capas inferiores, aplicando la regla de la cadena, para calcular la contribución que cada parámetro tuvo en el valor de pérdida, es lo que se conoce como diferenciación simbólica.
En cada iteración ,una vez que todas las neuronas disponen del valor del gradiente de la función de pérdida que les corresponde, se actualizan los valores de los parámetros en sentido contrario al que indica el gradiente

TIPOS DE DESCENSO DE GRADIENTE

SGD(Descenso de gradiente estocástico): Se estima el gradiente a partir del error observado para cada muestra del entrenamiento,el término estocástico se refiere al hecho de que cada dato se extrae al azar.
Descenso de Gradiente en lotes : Se da cuando usamos todo el conjunto de datos de entrenamiento en cada paso del algoritmo de optimización.

Típicamente el primer caso nos da mejores resultados, pero existen motivos que justifican el segundo caso, porque muchas técnicas de optimización solo funcionan cuando se aplica el proceso a un conjunto de puntos en vez de a uno solo.

Pero si los datos de entrenamiento están bien distribuidos, un pequeño subconjunto de ellos nos debería bastar.Por esto a menudo se usa una tercera opción,llamada descenso de gradiente en minilotes, esta opción suele ser mejor que el descenso de gradiente estocástico y se requieren menos cálculos para actualizar los parámetros de la red neuronal.

***Observación** : En el algoritmo descenso de gradiente en lotes, nos referimos por lotes a todo el conjunto de entrenamiento,pero muchas veces cuando nos referimos a lotes, nos referimos a lo que aquí hemos llamado mini lotes, es decir, una parte de los datos de entrada, por eso muchas veces cuando nos referimos a lotes, nos referimos a un subconjunto de los datos de entrada.*

La mayoría de las veces se usa un SGD, con un subconjunto de muestras del conjunto de muestras de entrenamiento, que llamaremos lote. Para asegurarse que se usan todos los datos, recordemos que en SGD se usaban todas las muestras, lo que hacemos es particionar los datos de entrenamiento en varios lotes de un tamaño determinado. Entonces cogemos el primer lote, lo pasamos por la red, se calcula el gradiente de la función de pérdida y se actualizan los parámetros de la red neuronal, esto seguiría hasta el último lote.

FUNCIÓN DE PÉRDIDA

Recordemos que el objetivo de esta función es guiar el proceso de aprendizaje de la red, mediante el cálculo de la diferencia entre el valor predicho y el valor real, ese valor, la perdía ,se propaga al optimizar el cual intentará ajustar los parámetros para minimizar dicho error .

La siguiente tabla nos muestra los diferentes tipos que podemos encontrar :

Tipo de Problema	*Función de Activación*	*Función de pérdida*
Clasificación múltiple con one-hot(ejemplo de los números)	*softmax*	*categorical_crossentropy*
Clasificación múltiple con índice categoría	*softmax*	*sparse_categorical_crossentropy*
Regresión	*ninguna*	*mse*
Clasificación binaria	*sigmoid*	*binary_crossentropy*