Introducción

De las redes neuronales artificiales se dicen muchas cosas, ¿pero que son en realidad? Se dice que fueron inspiradas por el funcionamiento del cerebro humano, que las neuronas(biológicas) y su interconexión motivaron a los pioneros de la inteligencia artificial a pensar en como implementar un algoritmo similar para dotar a las maquinas la capacidad de aprender y pensar.

Si bien es cierto que esta fue la premisa inicial, sería muy difícil de implementar de una forma artificial una replica exacta de este sistema, debido a que aun hoy no se sabe como funciona el cerebro biológico.

Entonces, ¿que son las redes neuronales? Son un algoritmo computacional para aprendizaje automático supervisado, que utiliza nodos independientes e interconectados llamados perceptrones, para almacenar los valores aprendidos de los conjuntos de entrenamiento. Son utilizadas para problemas de clasificación y regresión. Su ventaja principal es que pueden incorporar múltiples pesos W para cada valor X permitiendo adaptarse bastante bien a la generalización.

En este trabajo vamos a abordar el algoritmo de las redes neuronales en detalle e implementaremos su funcionamiento en el lenguaje de programación C#, explicando cada uno de sus componentes y de cómo interactúan entre sí.

Objetivo general

El objetivo de este trabajo es estudiar el algoritmo básico de las redes neuronales y de como llevar a cabo su implementación, explicar por que funciona, como funciona y por que es importante su conocimiento y cuales problemas se pueden resolver con este sistema. Es un trabajo practico, pero se incluirá teorías relevantes para el mismo.

Objetivos específicos

Estudiar el algoritmo de las redes neuronales

Estudiar los diferentes componentes de las redes neuronales

Implementar el algoritmo en el lenguaje de programación C#

Realizar ejemplos de aprendizaje supervisado, regresión y clasificación con esta implementación

Comparar los resultados con la implementación de scikit-learn

Dar una breve explicación de problemas que se pueden resolver con este algoritmo y por qué.

Capitulo 1. Definiciones

Perceptrón simple

El perceptrón simple es un modelo lineal de aprendizaje supervisado, sirve para regresión y para clasificación, al añadirle la función de activación. Este esta limitado en que el set de datos debe ser linealmente separable.

Perceptrón multicapa

Es una combinación de varios perceptrones, en donde existe una hilera de perceptrones en paralelo que conforman una capa y estos a su vez se conectan a otra hilera de perceptrones en paralelo que seria otra capa, por esto el nombre de multicapa, y así la salida de una capa seria la entrada de otra capa, al igual que el perceptrón simple sirve para regresión y clasificación la ventaja de este es que el conjunto de datos puede no ser linealmente separable.

Tanto el perceptrón simple como multicapa funcionan asignado pesos W a las entradas X para obtener una suma ponderada que de como resultado el valor de salida esperado en cada caso.

La fórmula del perceptrón simple seria:

Activación (∑W \* X)

donde:

W= vector de pesos

X= vector de datos de entrada

Activación = función de activación

Función de activación

La función de activación es una función que toma como entrada la combinación lineal (∑W \* X) y devuelve el resultado de dicha función, se utiliza para modificar la salida.

Las funciones de activación usadas en este TFM son:

Ninguna: no aplica ninguna transformación. Activación(x) = x

Umbral: devuelve solo dos valores 1 o 0, se puede utilizar para hacer una clasificación binaria. Activación(x) = si x<0 entonces 0 de lo contrario 1.

Relu: omite los valores negativos. Activación(x)= si x<0 entonces 0 de lo contrario x

Sigmoide: función que devuelve la probabilidad de pertenecer a una clase u otra, se utiliza para clasificador binario, pero agrega la probabilidad. Activación(x) = 1/(1 + ex)

Tanh: función que devuelve números comprendido entre -1 y 1. Activación(x)= (ex – e-x) / (ex + e-x)

Softmax: a diferencia de las demás funciones esta toma como valor de entrada un vector y devuelve un vector del mismo tamaño, con números comprendidos entre 0 y 1, la suma de este vector dará siempre 1, cada uno de los elementos de este vector contendría la probabilidad de pertenecer a una clase, sirve para problemas de clasificación donde el número de clases será mayor o igual a 2.

Activación(Z)= ezj/∑kezk

Donde:

Z= vector de entrada

ezj= Euler elevado al componente del vector Z

kezk= Sumatoria de Euler elevado a cada componente del vector Z

Función de perdida

Para encontrar los pesos correspondientes del modelo de red neuronal se realiza mediante un proceso conocido como optimización, en este interviene la función de perdida.

Esta función es una comparación de valores que se realiza entre el valor predicho por la red y el valor real o esperado, mientras mas difiera este valor mayor seria la perdida, de ahí que la optimización consista en minimizar dicha perdida. A la perdida también se le conoce como error.

Las diferentes funciones de perdidas utilizadas en este trabajo tienen la siguiente firma:

L = perdida(y, y’)

Donde:

L= perdida del lote o batch

y= valor esperado

y’ valor predicho por la red.

Las funciones de pérdidas son:

MSE: Mean Square Error o error cuadrático medio, la perdida debe ser siempre positiva de lo contrario perdidas negativas compensarían a las perdidas positivas, en esta función se eleva al cuadrado las diferencias de (y, y’) y se divide entre el numero de ejemplos en el lote(n).

MSE(y, y’) = (y – y’)2/n

MAE: Mean Absolute Error o error absoluto medio, similar a la anterior, pero en vez de elevar al cuadrado se calcula el valor absoluto de las diferencias de (y, y’).

MAE(y, y’) = |y – y’|/n

MSLE: Mean Squared Logarithmic Error o error cuadrado logarítmico medio similar a la MSE pero agrega el logaritmo, esta es usado cuando no se quiere penalizar cantidades bajas.

MSLE(y, y’) = (log(1 + y) – log(1 + y’))2/n

Binary CrossEntropy: utilizada para modelos de clasificación de 2 clases donde se espera que y sea un numero comprendido entre 0 y 1, como la diferencia de estos valores máximo valdrán 1 mas adelante cuando explique el de retropropagación se vera la importancia de que la perdida sea más que 1.

BinaryCrossEntropy(y, y’) = (y \* log(y’) + (1 – y) \* log(1 – y’))/n

Multiclass CrossEntropy: similar a la entropía binaria, pero en problemas de clasificación multiclase se utiliza esta variante

MulticlassCrossEntropy(y, y’) = - y \* log(y’)

Algoritmo de retro propagación

Optimizadores

Capitulo 2. Algoritmos

Capitulo 3. Implementación en C#

Capitulo 4. Pruebas y comparación de resultados con scikit-learn

Conclusiones