

Escuela Técnica Superior de Ingeniería Informática

Máster en Lógica, Computación e Inteligencia Artificial

TRABAJO FIN DE MÁSTER

Implementación de modelo de reconocimiento de expresiones faciales

AUTOR:

Ariel Batista Valdez

TUTOR:

José Luís Ruiz Reina

JUNIO DE 2020

Introducción

Los modelos de reconocimiento de imágenes son parte fundamental del día a día en muchas aplicaciones inteligentes. Con este tipo de modelos se pueden automatizar muchas funciones y tareas que de otra forma seria imposible.

En este trabajo de fin de máster vamos a abordar un modelo de clasificación de imágenes, específicamente uno de distinga entre distintas expresiones faciales, nos vamos a centrar en un único algoritmo: El perceptrón multicapa que a su vez es un tipo de red neuronal.

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 cómo 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 réplica exacta de este sistema, debido a que aun hoy no se sabe cómo funciona el cerebro biológico.

Entonces, ¿que son las redes neuronales artificiales? 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 del perceptrón multicapa en detalle e implementaremos su funcionamiento en el lenguaje de programación C#, explicando cada uno de sus componentes y de cómo interactúan entre sí, aplicándolo a la creación del modelo de clasificación de imágenes específicamente imágenes para el reconocimiento de las expresiones faciales.

Objetivo general

El objetivo de este trabajo es estudiar el de crear un modelo de reconocimiento de expresiones faciales utilizando un algoritmo de perceptrón multicapa, se incluirá la importancia y los posibles usos de este modelo. Es un trabajo practico, pero se incluirá teorías relevantes para el mismo.

Objetivos específicos

Estudiar el algoritmo del perceptrón multicapa.

Estudiar los diferentes componentes del perceptrón multicapa.

Implementar el algoritmo en el lenguaje de programación C#.

Crear varios modelos de clasificación de expresiones faciales.

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

Justificación de la importancia de este modelo.

Capítulo 1. Marco teórico

Modelo de Clasificación:

Modelo de Clasificación de Imágenes:

Modelo de Clasificación Binaria:

Modelo de Clasificación Multiclase:

Modelo de Clasificación Multi-Etiqueta:

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 está 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 sería 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 dé como resultado el valor de salida esperado en cada caso.

La fórmula del perceptrón simple seria:

Y’ = Activación (∑W \* X)

donde:

W= vector de pesos

X= vector de datos de entrada

Activación = función de activación

Y’ = valor predicho

Red Neuronal Artificial:

Red Neuronal Convolucional:

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, debe ser usada para casos de regresión

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 más 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:

E= perdida(y, y’)

Donde:

E= 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 número 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 más adelante cuando explique el de retro-propagación 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

Al igual que en modelo de regresión lineal en las redes neuronales se utiliza el descenso por gradiente para encontrar las matrices de pesos W que minimicen la perdida.

Una vez obtenido el error de la capa de salida, es necesario encontrar la matriz deltaW, esta matriz se sumará a W para así obtener la matriz de pesos que se necesita.

Para calcular la matriz deltaW se necesita propagar el error y calcular por cada nodo su error correspondiente, en caso de que la función de activación sea derivable se multiplica también y por último se multiplica por la tasa de aprendizaje.

DeltaWo = ( Eo . WoT)\* dActivacion \* tasa de aprendizaje

Wo = Wo + deltaWo

Donde:

Wo = matriz de la última capa

Eo= perdida del lote o batch

dActivacion = pendiente de la función de activación

Una vez calculado el error a la capa de salida, se debe proceder a calcular el error a la capa previa y así hasta la última primera capa, de ahí el nombre de retro-propagación porque la optimización comienza desde la última capa hacia la primera.

Para las calcular las capas ocultas seria todo muy parecido, simplemente se necesita calcular el error de dicha capa, así por ejemplo para calcular la matriz DeltaW de la penúltima capa seria:

DeltaWh = (Eh . Wht) \* dActivacion \* tasa de aprendizaje

Wh = Wh + DeltaWh

Donde:

Eh = ( Eo . Wot)

Wh = matriz de la penúltima capa oculta

dActivacion = derivada de la función de activación, si tiene.

Para las demás capas solo sería cuestión de calcular dicho error de la misma forma, Eh sería igual al error de la capa previa multiplicado la matriz transpuesta de los pesos de la capa actual.

Optimizadores

La optimización de la red neuronal consiste en encontrar las matrices de pesos W que minimicen el valor de la función de perdida, el algoritmo de la retro propagación sirve a este propósito, pero dentro de este se incluyen diferentes variaciones, los optimizadores, dependiendo del elegido y de los parámetros suministrados se encontrara el mínimo más o menos mínimo de la función de perdida en más o menos tiempo.

La tasa de aprendizaje, esta es un valor constante, multiplicado por la matriz DeltaW se minimiza los valores de esta y permite cambiar el tamaño del paso dado. Pero que sea un valor constante no es útil pues a medida que se aproxime al mínimo es necesario disminuir el avance, para no pasarse, una de las formas es reducir a la tasa de aprendizaje a ritmo constante otra seria utilizar el optimizador ADAM.

Los optimizadores implementados en este trabajo son:

SGD: Stocastic Gradient Decent o Descenso por gradiente estocástico, en este se multiplica la matriz Grandiente por la tasa de aprendizaje.

DeltaW = Gradiente \* Tasa de aprendizaje.

Donde:

Gradiente = E \* WT \*dActivación

ADAM: Adaptive Moment Estimation o Estimación de momento adaptativo, es un optimizador de que calcula una tasa de aprendizaje variable para iniciar con un valor alto al inicio y decae en cada interacción, este utiliza 2 parámetros que son beta\_1 y beta\_2 valores que pueden ser modificados, pero en la mayoría de los casos siempre se presentan con los mismos valores beta\_1 = 0.9 y beta\_2= 0.999.

Por cada Matriz de pesos W de cada capa se calcula 2 matrices de variaciones llamada momentos M y V, estas matrices se inicializan en 0, y son utilizadas a partir de la segunda interacción, adicionalmente se utiliza el numero de la interacción actual.

M = \_beta1 \* Manterior + (1 - \_beta1) \* Gradiente;

V = \_beta2 \* Vanterior + (1 - \_beta2) \* Gradiente 2

Tanto la matriz M como la V se almacenan para poder ser utilizada en la siguiente iteración, por eso M y V actuales serán menor que M y V de la interacción anterior. Con los valores de M y V calculados se procede a calcular las matrices M’ V’ y la diferencia de M’ entre V’ elevado al cuadrado conformaran la nueva matriz gradiente.

M’ = M / (1 – beta1epoch +1)

M’ = M / (1 – beta2epoch +1)

Nuevo Gradiente = M’/ V2

Donde:

Epoch= número de epoch actual, como este comienza en 0 se le añade 1.

Finalmente se actualizan los pesos.

DeltaW = nuevo Gradiente \* tasa de aprendizaje.