Universidad Autónoma de Yucatán

Maestría en Ciencias de la Computación

Redes Neuronales Convolucionales

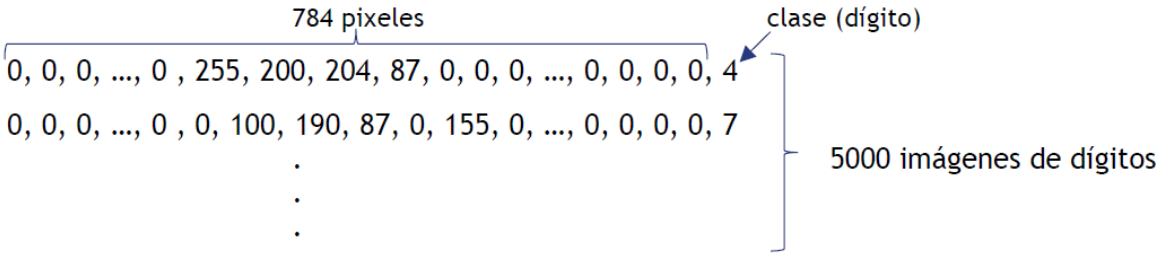
Práctica 4

Autor: Mario Herrera Almira

20 de febrero del 2023

Artificial Neural Networks
Implementar una Red Neuronal Artificial Multiclase para la clasificación de imágenes de dígitos del 0 al 9 escritos a mano (*handwriting digit recognition*).

La base de datos (mnist.txt) contiene 5000 muestras de imágenes de dígitos escritos a mano:



Cada renglón de la base de datos consiste en 785 elementos separados por comas, donde los primeros 784 valores identifican los pixeles de la imágen del dígito, y el último valor del renglón indica la clase a la que pertenece esa muestra. Los renglones están separados por un “Enter”.

El vector renglón de 784 valores representan una imagen de 28 x 28 pixeles. Los primeros 28 valores del vector,representan la primera columna de pixeles de la imagen, los siguientes 28 valores representan la segunda columna de pixeles, y así sucesivamente:

El objetivo principal es implementar una red neuronal (NN) artificial multiclase de retro propagación para clasificar las imágenes de dígitos del 0 al 9 escritos a mano. Las características de entrada de la NN son los 784 pixeles de la imagen. Para este trabajo sólo se utilizarán las primeras 1000 muestras de imágenes de la base de datos. Para entrenar a la NN (ajustar parámetros o pesos) utilizar las primeras 900 imágenes y los 100 restantes serán para probar el desempeño de la red. Calcular el error de la función de costo para cada época.

Presentar en un reporte la descripción del problema, el método utilizado (incluyendo arquitectura de la NN, su función de costo, gradientes, etc.), resultados y comentarios. Asimismo, entregar el código fuente para ser compilado y evaluado.
Como resultados, mostrar el porcentaje de reconocimiento final de la NN, esto es el porcentaje de aciertos de clasificación para las 100 nuevas muestras. Mencionar algunos de los errores de clasificación indicando la clase obtenida por la NN comparada con su verdadera clase. Graficar, con un graficador de su preferencia, los valores de costo obtenidos en cada época para observar el descenso del error durante el entrenamiento.

Presentar en el reporte la porción del código correspondiente a la función de costo y la actualización de pesos.

Al ejecutar el código fuente, debe cargar el modelo NN entrenado y realizar el proceso de prueba con las 100 nuevas muestras. Desplegar en pantalla: el porcentaje de reconocimiento de la red neuronal artificial en la prueba.

Respuesta:

El método utilizado para resolver este problema fue construir una red neuronal multicapas capaz de reconocer pequeñas imágenes de dígitos escritos a mano. Para ello se construyó una red neuronal que posee una capa de entrada con 784 neuronas una para cada píxel de la imagen de entrada (tamaño de 28 x 28), una capa oculta de 200 neuronas y una capa de salida de 10 neuronas (una por cada dígito).

Para calcular la función de costo de la salida de la red neuronal se utilizaron las siguientes fórmulas:

Esta primera fórmula representa la salida o predicción del modelo en un *Forward Pass* convertido a probabilidades utilizando la función conocida como Función *Softmax*:

$$\hat{y} = \frac{e^{S_k}}{\sum_j e^{S_j}}$$

La siguiente fórmula es una primera representación de la función de costo donde se puede observar que el costo va a depender de la sumatoria de la multiplicación de la predicción esperada del modelo (y_j) que es el vector columna donde todos los valores son ceros excepto el valor que representa a la clase que debería haber predicho el modelo según la imagen de entrada por el logaritmo de lo que realmente arrojó el modelo al final del *Forward Pass* ($\ln(\hat{y}_j)$). Luego de realizar algunas operaciones aritméticas se puede determinar que esta sumatoria va a ser igual a menos el logaritmo natural de $\hat{y}$ como se muestra a continuación:

$$L_i = -\sum_{j=1}^{\#clases} y_j \ln(\hat{y}_j) = -\ln\left(\frac{e^{S_k}}{\sum_j e^{S_j}}\right)$$

Finalmente la formula anterior se puede calcular para cada clase y lo que se desea es un único valor promedio que indique que tan bien o mal están los valores de los pesos y los sesgos por lo que se realiza el promedio de la fórmula anterior quedando finalmente la **función de costo** de la manera siguiente:

$$L_{(w,b)} = \frac{1}{m}\sum_{j=1}^{m} -\ln\left(\frac{e^{S_k}}{\sum_j e^{S_j}}\right)$$

O lo que es lo mismo:

$$L_{(w,b)} = \frac{1}{m}\sum_{j=1}^{m} -\ln(\hat{y})$$

Para el cálculo de los gradientes se utilizó el algoritmo de retro propagación donde primero se calcula la derivada de la función de costo con respecto a Z que es la salida de la capa oculta de la red neuronal. Esta derivada se puede calcular de la forma siguiente:

$$\frac{\partial J}{\partial Z} = \hat{y} - y$$

Donde ya se había explicado anteriormente que $\hat{y}$ es la predicción que devuelve el modelo luego de hacer un *Forward Pass* y y es lo que debería haber devuelto si ya estuviera perfecta la respuesta que sería un vector donde todo sería cero menos el elemento que clasifica correctamente a la imagen de entrada.

Luego esta derivada se va propagando hacia atrás en la red neuronal aplicando la regla de la cadena, en cada neurona se calcula las derivadas parciales de la derivada que regresa de la neurona posterior con respecto a los pesos “W” y los sesgos “b”. Por ejemplo, en la última capa oculta la derivada que regresa es precisamente la derivada del costo con respecto a la salida que es la formula que se planteó hace unos momentos, entonces para una neurona de la capa oculta se calculas las derivadas parciales de “W” y “b” de la siguiente manera.

Para los pesos de las aristas:

$$\frac{\partial J}{\partial W} = \frac{\partial J}{\partial Z} * \frac{\partial Z}{\partial W}$$

Para los sesgos:

$$\frac{\partial J}{\partial b} = \frac{\partial J}{\partial Z} * \frac{\partial Z}{\partial b}$$

En el caso de los sesgos la derivada $\frac{\partial Z}{\partial b}$ es igual a 1 por lo que quedaría de la siguiente manera:

$$\frac{\partial J}{\partial b} = \frac{\partial J}{\partial Z}$$

Como al final todas estas derivadas en realidad van a ser equivalentes a matrices y $\frac{\partial Z}{\partial W}$ corresponde a la matriz columna que contiene los valores de entrada de la red neuronal entonces la fórmula de los pesos de las aristas se puede escribir de la siguiente manera:

$$\frac{\partial J}{\partial W} = \frac{\partial J}{\partial Z} * X^T$$

Donde X^T representa la traspuesta de la matriz columna que contiene los valores de entrada. Por cuestiones de simplicidad en el código todas estas derivas se representaron de la siguiente manera:

$$\frac{\partial J}{\partial Z} => dz$$

$$\frac{\partial J}{\partial W} => dw$$

$$\frac{\partial J}{\partial b} => db$$

Entonces en el código se pueden apreciar estos cálculos de la forma siguiente respectivamente:

$$dz = \hat{y} - y$$

$$dw = dz \,@\, X^T$$

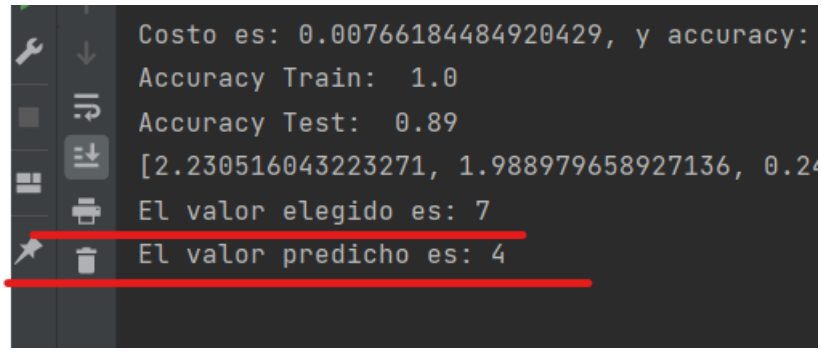
$$db = dz$$

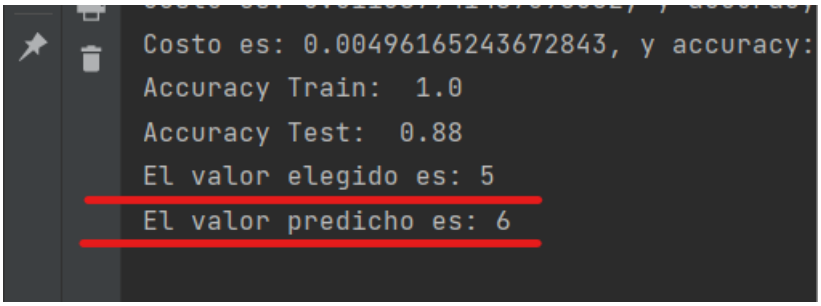
Donde @ es un operador para realizar el producto punto entre dos matrices.

Resultados y Comentarios

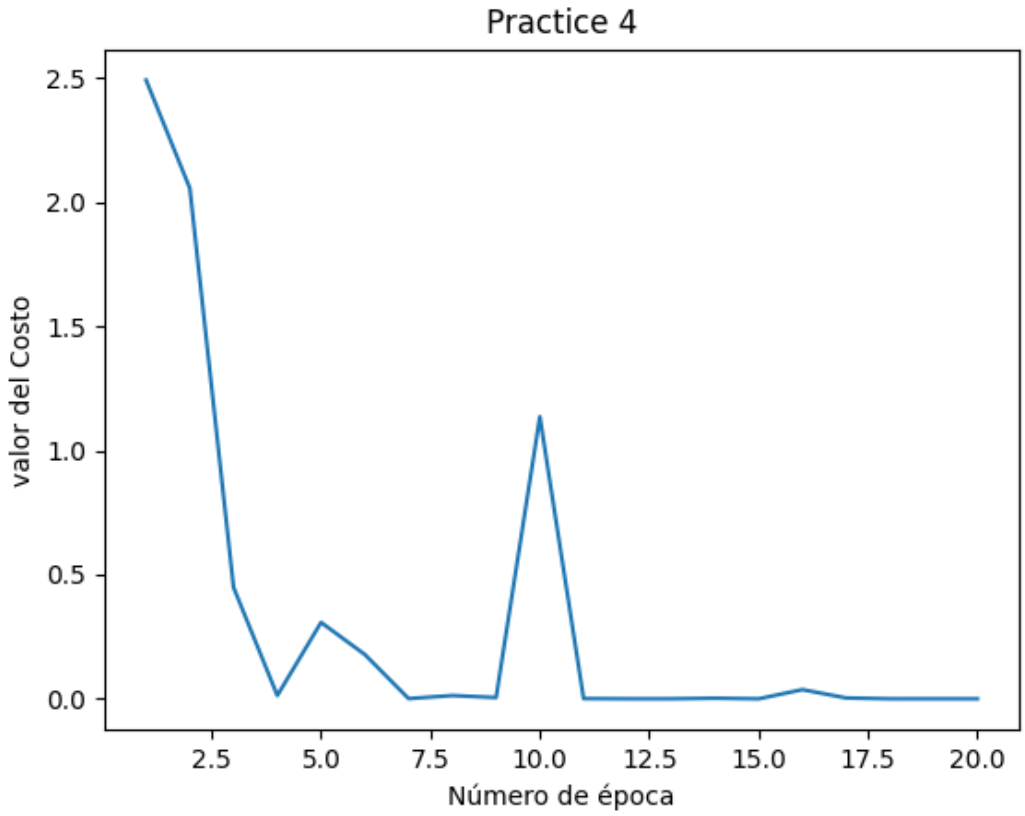
La red neuronal presentó unos resultados bastante buenos considerando que solo se utilizan 900 imágenes para entrenarla y 100 para probar su efectividad con imágenes que nunca antes haya visto. La efectividad obtenida con respecto al grupo de entrenamiento es de aproximadamente **100%** mientras que con el grupo de prueba arroja resultados entre **88%** y **90%**.

En las siguientes figuras se pueden observar algunos de los errores cometidos por la red neuronal a la hora de clasificar las imágenes en el conjunto de prueba. Para arrojar estos resultados se utilizó la función *predict* que permite utilizar los valores ya optimizados para clasificar una imagen seleccionada aleatoriamente del conjunto de prueba:





En la siguiente gráfica se puede observar claramente como disminuye el costo en cada época, aunque en algunas ocasiones este costo da unos saltos hacia arriba para luego continuar bajando:



Fragmentos de código requeridos

Función de costo

En este fragmento de código se puede observar el momento en que se calcula el costo de la predicción de la red neuronal utilizando la fórmula descrita anteriormente

$$L_{(w,b)} = \frac{1}{m}\sum_{j=1}^{m} -\ln(\hat{y})$$

```
def x_entropy(scores, y, batch_size):
    probs = softmax(scores)
    y_hat = probs[y.squeeze(), np.arange(batch_size)]
    cost = np.sum(-np.log(y_hat)) / batch_size

    return probs, cost
```

Actualización de los pesos

En este fragmento de código se puede observar el momento donde se actualizan los pesos de la red neuronal, así como los sesgos, en todos los casos utilizando la función general:

$$w = w - \alpha\left(\frac{dJ}{dw}\right)$$

```
def train(epochs, parameters, mb_size, learning_rate):
    for epoch in range(epochs):
        for i, (x, y) in enumerate(create_mini_batches(mb_size, x_train, y_train)):
            scores_2, z1, a1 = scores(x.T, parameters)
            y_hat, cost = x_entropy(scores_2, y, batch_size=len(x))
            grads = backward(y_hat, x, y, z1, a1, scores_2, parameters, batch_size=len(x))

            parameters['w1'] = parameters['w1'] - learning_rate * grads['w1']
            parameters['b1'] = parameters['b1'] - learning_rate * grads['b1']
            parameters['b2'] = parameters['b2'] - learning_rate * grads['b2']
            parameters['w2'] = parameters['w2'] - learning_rate * grads['w2']

        print(f'Costo es: {cost}, y accuracy: {accuracy(x_test, y_test, mb_size)}')
        cost_list.append(cost)

    return parameters
```