Tecnológico de Monterrey

Programación multinúcleo

Octavio Navarro

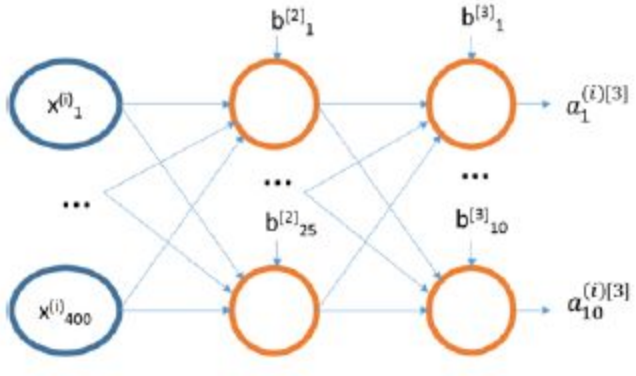
Detección de dígitos escritos a mano

Cynthia Castillo Millán A01374530

Pablo Macías Landa A01229743

Descripción del proyecto
Se implementó una red neuronal que logre clasificar dígitos escritos a mano. Cada dígito de ejemplo es de 20 x 20 pixeles en escala de grises, cada uno representado por un número en el rango de [-1, 1].
El dataset está conformado por 5000 ejemplos, cada uno en una línea donde el último número es la etiqueta del número del que se trata.
Se realizó una Red Neuronal con Backpropagation sin bias ni regularización

Solución en GPU
El tipo de red neuronal que se implementó fue de la siguiente forma:



Donde el input son los 400 valores de los píxeles, 25 nodos en una capa oculta y 10 valores de salida. La salida está contemplada para ser un vector[10] de ceros donde se espera que sólo el índice que corresponde a su dígito contenga un 1.

Se definieron las siguientes matrices:

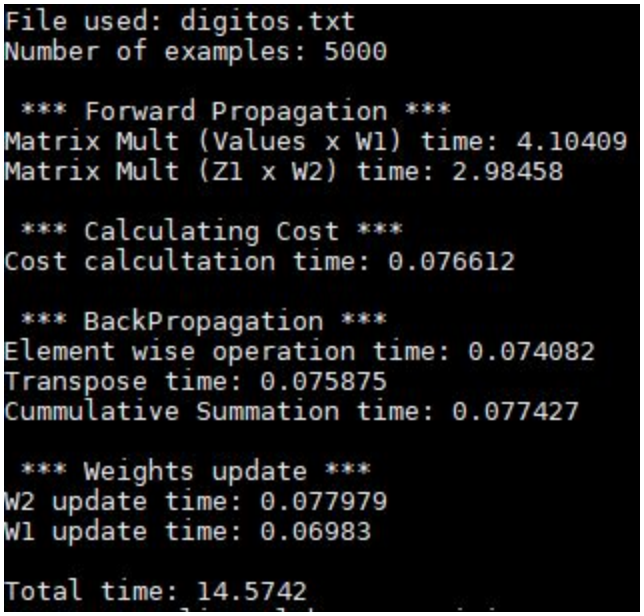
Nombre	**Dimensiones**	**Descripción**
values	5000 x 400	todos los píxeles de todas las imágenes
outputs	5000 x 10	etiqueta de cada imagen
w1	400 x 25	pesos entre la entrada y la capa oculta
w2	25 x 10	pesos entre la capa oculta y la salida
z1	5000 x 25	w1 * values
z2	5000 x 10	z1 * w2
delta3	5000 x 10	resultado obtenido menos resultado esperado (z2 - outputs)

delta2	5000 x 25	Permite actualizar los pesos de W2 w2^T * delta3
delta1	5000 x 400	Permite actualizar los pesos de W1 w1^T*delta2

Se crearon 5 kernels para realizar las operaciones entre matrices:

Kernel	**Función**
matrixMult	Multiplicación entre matrices
Transpose	Transposición de una matriz
ElemWise	aplicar operaciones a cada elemento de dos matrices
CostFunc	Obtener el costo mediante Softmax
cumulativeSum	Obtener la suma acumulada de un vector bidimensional respecto a su dimensión X

Tiempos Obtenidos por Kernel y total:



Tiempo promedio: 13.7 s

Solución en CPU

Para la implementación en cpu se utilizó python por su facilidad con librerías externas para mejorar el entrenamiento, se hizo una prueba en c++ también pero la versión de python obtuvo mejores resultados.

Primero se crea la matriz de pesos w1 y w2, se llena con valores random para empezar el entrenamiento. Utilizamos la librería de optimize.minimize dentro de scipy que utiliza nuestra función para calcular los costos que funciona igual que la versión de GPU multiplicando el sigmoidal de z3 de la diferencia entre los resultados obtenidos contra los resultados esperados para obtener delta3 y delta 2 que se obtiene multiplicando el sigmoidal de z2 por la multiplicación de delta3 y la transpuesta de w2. Con delta2 y delta3 obtenemos los nuevos pesos multiplicandolos por la transpuesta de la matriz de entrada y por la transpuesta de a2 respectivamente. La librería hace esta operación en 200 iteraciones en aproximadamente 3 horas.

GPU	**CPU**	**Speedup**
13.7 s	10800 s	0.00126851851

Referencias

1. **Stanford - wiki “Softmax Regression”** http://ufldl.stanford.edu/wiki/index.php/Softmax_Regression
2. **Welch Labs - Demystifying Neural Networks** https://www.youtube.com/watch?v=bxe2T-V8XRs
3. **Stanford - wiki “Softmax Regression”** http://ufldl.stanford.edu/wiki/index.php/Softmax_Regression
4. **A. Ng. Machine Learning. Curso de Coursera-Stanford (2011).**