Tarea 1

Javier Espinoza
2 de septiembre de 2017

CC5114-1 Redes Neuronales y Programación Genética
Departamento de Ciencias de la Computación
Facultad de Ciencias Físicas y Matemáticas
https://github.com/jaevespinoza/neural-network

1. Cómo correr el análisis

La red desarrollada corresponde a una con 4 layers, 9 neurones de inputs, 7 y 5 neurones en las hidden layers, y 2 en el layer de outputs, debido que solo hay 2 clases: si ganó X o si no ganó.

El dataset analizado corresponde a jugadas de TicTacToe, y se quiere predecir si dado un input (un juego), si ganó el jugador X o no.

Para correr el análisis de los datos, se debe ejecutar el programa *TicTacToe.py*, y tener el archivo *tictactoe.txt* en la misma carpeta. Lo que hace el programa es leer el dataset, y comienza a leer línea por línea y entrenar una Network aleatoria para poder aprender y luego predecir el resultado de los siguientes inputs. Lo que el programa imprime es el porcentaje de inputs predecidos incorrectamente. Hay 2 formas de correr el programa: La primera corresponde a crear siempre una red aleatoria y entrenarla una cantidad i de epochs a cada una, mientras i va aumentando progresivamente. La segunda forma es tener una red fija, y entrenarla con una cantidad i de epochs, aumentando i. Luego, se corre *Plot.py* con el archivo de los resultados, y se grafica dependiendo de cuántas líneas tiene el archivo *result.txt*.

La Figura 1 corresponde a correr el programa de la primera manera, mientras que la Figura 2 es el resultado de correr *TicTacToe.py* de la segunda manera.

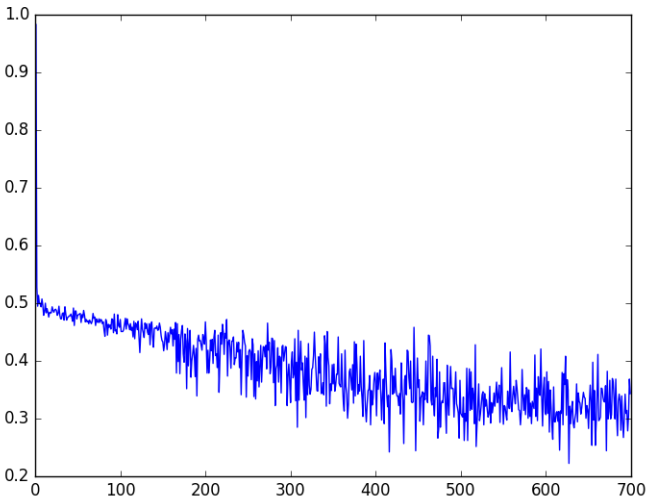

Figura 1: Resultado de correr TicTacToe.py creando una red aleatoria por cada iteración y entrenarla con i epochs. Se puede ver el decremento del error, pero no baja la cantidad suficiente, es decir, sigue habiendo un error. EL máximo de epochs usados fueron 700.

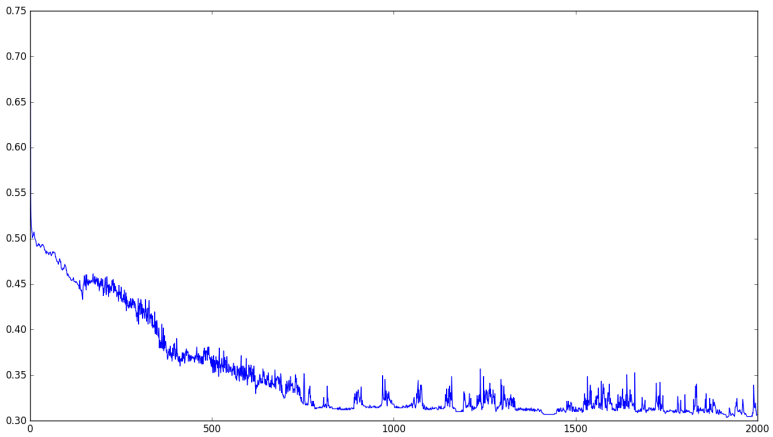

Figura 2: Resultado de usar el segundo método. Se ve un decremento del error más estable, pero al igual que el primer método, no baja completamente a un error 0. EL máximo de epochs usados fueron 2000.

2. Cómo el número de layers tiene impacto en el aprendizaje?

Tener más hidden layers no necesariamente dará un mejor resultado. Esto puede verse en el siguiente gráfico:

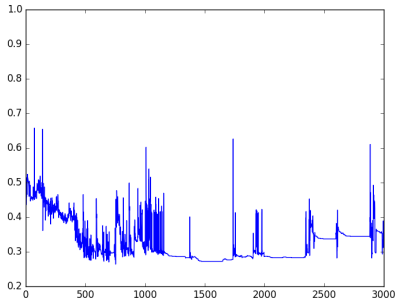

Figura 3: Resultado de usar 5 layers (3 hidden, 1 de input y otra de output). Se puede ver que los errores son más grandes de vez en cuando, con más variaciones a medida que se aumentan los epochs.

El número de hidden nodes/layers depende de una relación compleja entre: EL número de inputs y outputs, la cantidad de data disponible para entrenar, la complejidad de la función que se está tratando de aprender y el algoritmo de entrenamiento.

Para minimizar el error a lo más pequeño posible, se debe escoger un número óptimo de hidden layers y nodos para cada layer. Si hay muy pocos nodos, nos dará un error muy grande para el sistema pues los factores predictivos pueden ser muy complejos para un número pequeño de nodos. Si hay muchos nodos, se no generalizarán correctamente los resultados.

3. Cuál es la velocidad de procesamiento?

Depende de cuantas layers hayan dentro de la red, y del formato de la data que se quiere usar. En el caso usado, entrenar con 1000 epochs demora alrededor de 3 minutos al tener que leer toda la data del archivo por cada iteración.

4. Efecto de diferentes learning rates

A mayor learning rate, se tiene que los inputs/deltas tienen más influencias al hacer back propagation y al actualizar los pesos. Pero eso hace que haya menos precisión y que los pesos varíen más. A menor learning rate, hay una mayor

precisión al hacer los dos procesos mencionados anteriormente, pero se necesitan más inputs/epochs para poder llegar a una red que prediga correctamente. Las figuras 4 y 5 corresponden a usar diferentes learning rates para el segundo método de aprendizaje.

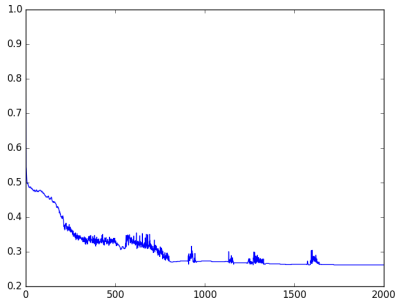

Figura 4: Resultado de usar un learning rate de 0.3

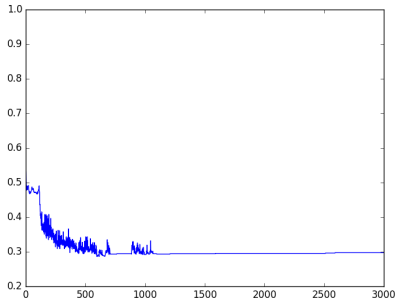

Figura 5: Resultade de usar un learning rate de 0.8

5. El orden del data set importa?

Importa por lo menos para esta implementación. Esto se debe a que el dataset con el que se trabajó tenía todos los inputs que daban a una cierta clase ordenados, y luego poseía los otros inputs pertenecientes a la otra clase. Ejemplo:

Debido a esto, se tuvo que modificar el programa *TicTacToe.py* para que pueda leer una linea de una clase e inmediatamente lea un input de la otra clase. Así, la predicción tiene un porcentaje de resultados incorrectos mucho más bajo.

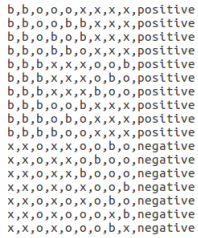

Figura 6: Ejemplo de formato de archivo. Se ve que se tiene 1 clase muchas veces, y luego se tienen los inputs de la otra. Esto hace que el aprendizaje sea incorrecto.

6. Qué neurones cambiaron durante la fase de entrenamiento?

Para ver esto, se desarrolló una función que imprima la diferencia de pesos según layers y nodos. Y dentro de los nodos se guardó la lista con los pesos iniciales. Solamente se puede ver el cambio a través de prints, y lo que se vio fue que el nodo 4 de la capa 1 cambió demasiado.

Cuadro 1: Cambio de pesos del nodo dicho

Peso antes de entrenar	Peso después
-1	-19.2827283852
0	-9.52050271986
1	-2.03916289636
0	0.332919023321
-3	1.2850556759
-2	-1.1675094821
3	-0.99034666309
2	0.280018629594
2	-0.795972669326