UnConfused Terminator

Instituto Tecnológico de Costa Rica
Ingeniería en Computación
Josué Fernández Díaz
2013033195
Email: josue.0795@gmail.com

18 de Mayo del 2018

Contents

Abstract

Se presenta el problema común de las redes neuronales, donde se presenta un ejemplo simple con la serie de datos MNIST, donde se puede notar las perdidas, que indican el aprendizaje de las redes, de forma que con cada iteración, o epoch, se acerca mas a su destino, o como se puede ver tiene comportamientos extraños dependiendo de los datos ingresado al inicio.

1 Introducción

Las redes neuronales son una poderosa herramienta capaz de incontables funcionalidades, y en esta ocasión se puede ver un método de inteligencia artificial supervisado por medio de redes neuronales con el set de datos MNIST de Yann Lecun de 1989, donde se tiene una cantidad de 70 mil datos, donde cada uno es una imagen de 28x28 que representa un numero del 0 al 9, donde por supuesto trae su etiqueta/label respectiva para poder identificar y enseñar a la red. Se usa sigmoid como funcion de activación, y softmax/cross entropy como perdida, mostrando varios resultados interesantes donde se puede ver el poder de las redes, y en otros cosas "alocadas". Posteriormente se3 cambió softmax por sigmoid, ya que presenta mejores resultados, además de que el material[2] de sigmoid es mucho mayor en internet, por lo que seguir esta rutina es mas sencillo.

2 Metodología

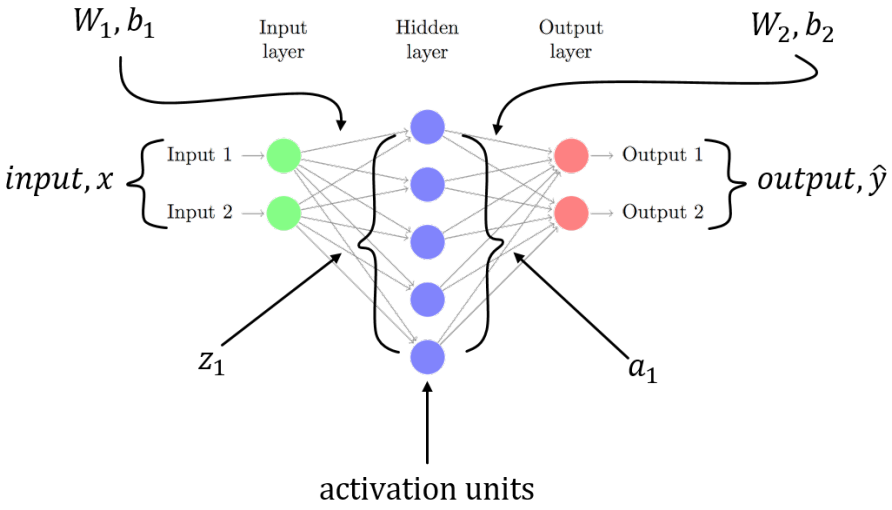
De funcion de activación se usó sigmoid[2], ya que ReLU presentaba problema, el cual era que después de obtener la primera perdida, inmediatamente tiraba valores NaN, y por supuesto continuaba imprimiendo estos valores. Por otro lado, el sigmoid con el mismo set de datos, el mismo tamaño de batch y la misma funcion de perdida, realizaba el proceso de forma correcta, presentando un valor de perdida razonable. Esta fórmula presenta la forma:

$$S(x) = \frac{1}{1 + e^{-x}} \tag{1}$$

La funcion de perdida de aplicó softmax[1], como se especificó en la tarea, formula la cual presenta la formula:

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \text{for j} = 1, \ldots, \text{K}. \tag{2}$$

Basándose en el proyecto, se usó mayormente lo que se aplica en estas imágenes, agregando por su puesto las funciones de activación, y la perdida que no se in-



cluyen.

$$z_1 = xW_1 + b_1 \qquad \delta_3 = \hat{y} - y, \delta_2 = \delta_3 W_2^T$$

$$a_1 = z_1 \qquad \frac{\partial L}{\partial W_2} = a_1^T \delta_3 \qquad \frac{\partial L}{\partial b_2} = \delta_3$$

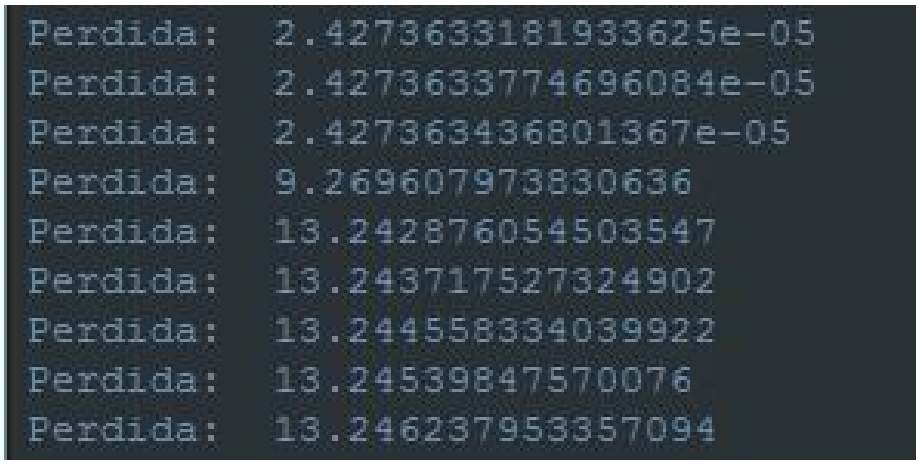
$$z_2 = a_1 W_2 + b_2$$

$$a_2 = \hat{y} = softmax(z_2) \qquad \frac{\partial L}{\partial W_1} = x^T \delta_2 \qquad \frac{\partial L}{\partial b_1} = \delta_2$$

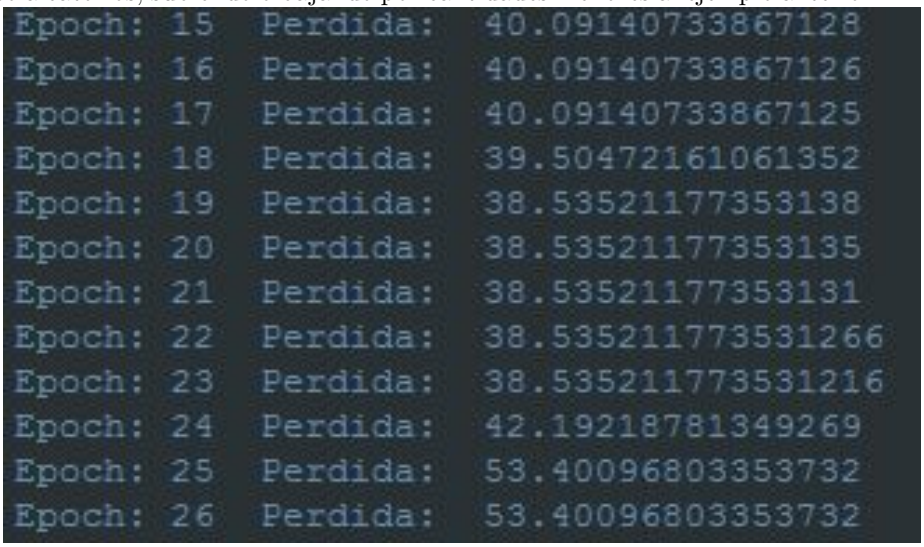
3 Experimentos y Resultados

Pruebas usando una sola capa: Se tuvieron resultados a grandes saltos, como por ejemplo este, que se mantenía en una zona estable hasta que en un punto saltaba completamente, ya sea aumentando o disminuyendo, por ejemplo de 14 bajaba a 0.000045 o viceversa. El siguiente ejemplo es el explicado, con un batch de tamaño 10.

Como experimento propio se intentó cambiar y elevar la cantidad de datos por epoch, dando como resultado que mientras menos datos, el error general es menor mientras menor sea el batch. Con un batch de 10 se puede llegar a una perdida menor a 1, mientras que subiendo, batch de 100 sube a 20 y en 1000 sube a 120. Estos por supuesto son diferentes pesos generalizados después de varias ejecuciones y no siempre se obtienen los resultados requeridos.



Personalmente creo que esto fue por los datos, al estar ordenados primero aparecen todos los 0 y de pronto saltan a los 1 y por eso el cambio drástica en el valor de perdida. Con un batch 1000, presenta un comportamiento extraño, ya que a pesar de que no realiza saltos como el anterior, si realiza movimientos un poco aleatorios, subiendo o bajando por cantidades menores al ejemplo anterior.



4 Conclusiones

Se puede ver como la red presenta varias comportamientos de forma que en algunas situaciones es eficaz, mientras que en otras no muestra dicho comportamiento, sino que se presenta de forma traviesa y realiza algunos cálculos de forma errónea.

References

[1] Sasam Shandasani. *Build a flexible Neural Network with Backpropagation in Python*. 2017. URL: https://dev.to/shamdasani/build-a-flexible-neural-network-with-backpropagation-in-python.

[2] weiji14. *nn from scratch*. 2017. URL: https://github.com/dennybritz/nn-from-scratch/blob/master/nn_from_scratch.py#L66.