Neural Network from Scratch

Rodriguez, Jose

jose.shamm@unitec.edu 21441215

Universidad Tecnológica Centroamericana

28 Junio, 2018

Resumen

Esta es una implementación de una red neuronal para la clase de Concurrencia de Sistemas Logicos. Es una de las implementaciones mas simples pero el tema es toda una ciencia. Tomo bastante investigación de mi parte, como podemos ver en la bibliografia. Con exito la implementacion se puede llevar en serie o en paralelo a traves de la manera en la cual entrenamos la red o las redes.

Neural Network from Scratch

Índice

Introducción

Como Aprenden las Maquinas

Tomando inspiracion de la naturaleza. Las maquinas como las conocemos son un conjunto de cables y electricidad sin pensamiento. En el campo de Ciencias de la Computación uno de los grandes avances de nuestros tiempos es la inteligencia artificial. En nuestra búsqueda a crear una maquina inteligente hemos intentado recrear el cerebro humano. De esta inspiración nacen las redes neuronales. La idea básica de una red es detectar errores, la pregunta es como hacerlo. Aquí vienen los componentes básicos pero críticos de una red, los neuronas, y sus conexiones. Cada neuron tiene una conexión y cada conexión tiene un peso que influye la decision de los neuronas para escoger esa ruta. Estos pesos combinado con nuestra step function es como estaremos decidiendo si tomar un camino resulto ser error o correcto.

Marco Teórico

Deep Learning

Funciones de Activación. Una función de activación tiene como propósito introducir un elemento no lineal al output de un neuron. Esto es importante por que queremos que nuestros neuronas aprendan de una manera no lineal. Toda función de activación toma un solo numero y realiza una operación matemática para darnos un output. Una de las funciones mas populares es la función de Sigmoid, es utilizada para tomar un numero y hacer un squash para que nos de un valor entre 0 y 1.

Las Neuronas de una Red. La estructura de una red neuronal es bien muy interesante. Tiene 3 capas que son, neuronas de entrada, neuronas escondidas y finalmente neuronas de salida. Las neuronas de entrada proporcionan información sobre el mundo exterior a nuestra red. También contiene un node conocido como el bias node que por default tiene un valor de 1. Nuestras neuronas escondidas tienen como propósito realizar

cálculos y transfieren información desde los nodos de entrada a los nodos de salida. Los nodos de salida, finalmente, realizan pocos cálculos y mas que nada muestran el resultado de nuestra red al mundo exterior.

Gradient Descent, Backpropagation y feedforwarding. Empecemos hablando sobre que es el propósito de gradient descent en nuestra red. El punto de el algoritmo de gradient descent es minimizar el error de nuestra red. Como vamos a hacer esto? Pues para nuestra sigmoid function, es tan simple como calcular su derivada. Backpropagation utilizara nuestra gradiente, multiplicada por el error del neuron y seteara el delta peso de nuestra conexión. Finalmente, feedforwarding se refiere a la manera en la que una red propaga y calcula su output. Feedforward es cuando el flujo de información se mueve en un solo sentido. Osea que el error se calcula inmediatamente en cada neuron y no necesitan propagar la información a otros neuronas cada una de las neuronas calculan de parte de lo que les dice la función de sigmoid.

Trabajo Realizado

Clase Conexión

```
class Connection:
  def __init__(self, connectedNeuron):
    self.connectedNeuron= connectedNeuron
    self.weight= numpy.random.normal()
    self.dWeight= 0.0
```

. La clase conexión es corta pero es la mas útil de todas nuestras clases. Esta clase es extremadamente simple y en retrospectiva debería llamarse dendron. Aquí lo único que hacemos es crear 3 miembros de instancia, la neurona conectada, su pedo y el peso delta. La neurona conectada la recibe en su constructor y de ahí le damos un peso aleatorio a nuestra conexion.

```
self.weight= numpy.random.normal()
```

Por que Peso Aleatorio?. Bueno, cada unidad oculta obtiene la suma de entradas multiplicada por el peso correspondiente. Ahora imagine que inicializa todos los pesos con el mismo valor (por ejemplo, cero o uno). En este caso, cada unidad oculta obtendrá exactamente la misma señal. P.ej. si todos los pesos se inicializan en 1, cada unidad recibe una señal igual a la suma de las entradas (y las salidas sigmoideas (suma (entradas))). Si todos los pesos son ceros, lo que es aún peor, cada unidad oculta obtendrá señal cero. No importa cuál fuera la entrada: si todos los pesos son iguales, todas las unidades en la capa oculta serÃąn las mismas también.

Clase Neuron

```
class Neuron:
  eta= 0.001
  alpha= 0.01
  def __init__(self, layer):
    self.dendrons= []
    self.error= 0.0
    self.gradient= 0.0
    self.output= 0.0
    if layer is None:
      pass
    else:
      for neuron in layer:
        con= Connection(neuron)
        self.dendrons.append(con)
```

. En nuestra clase neuron tenemos varios modulos, cubriremos los mas importantes empezando con el constructor de la clase. Aquí una neurona se inicializa con su

error,gradiente y su valor de salida en 0 y definimos una lista ***de instancia*** de conexiones. En nuestro for loop hacemos que el neuron tome su lista de conexiones y que cree nuevas conexiones y las agregue a su lista.

Backpropagation. Las funciones en nuestra clase son cortas pero es intuitivo entender lo que esta pasando por ellas. Veremos la mas importante y menos entendible a simple vista, pero si nuestras nueronas van a tener algunas otras funciones como la funcion de sigmoid, set error, get output, etc. Todas son funciones fácilmente entendibles en lo que hacen.

```
def backPropagate(self):
  self.gradient= self.error * self.dSigmoid(self.output)
  for dendron in self.dendrons:
    dendron.dWeight = Neuron.eta * (
    dendron.connectedNeuron.output * self.gradient) + self.alpha *
    dendron.dWeight
    dendron.weight = dendron.weight + dendron.dWeight
    dendron.connectedNeuron.addError(dendron.weight * self.
   gradient)
  self.error = 0
```

. Ahora en backprogation utilizaremos la formula $\delta weight = \eta \times gradient \times output of connected neuron + \alpha \times previous \delta weight$ para calcular el peso de la conexión como podemos ver aquí utilizamos el gradient calculado por nuestra función de sigmoid. Después es simple el peso es el error mas el delta peso que hemos calculado y finalmente le añadimos eso, multiplicado por el gradiente para minimizar el resultado a nuestro error en el dendron. Basado en el calculo de este peso es que feedforwarding nos da el output de nuestras neuronas.

Clase Network

```
class Network:
  def __init__(self, topology):
    #topology is a list containing [input_layer,hidden_layers,
   output_layer]
     self.layers= []
     for numNeuron in topology:
       layer= []
       for i in range(numNeuron):
         #we got no layers then
         if not self.layers:
           layer.append(Neuron(None))
         else:
           layer.append(Neuron(self.layers[-1]))
       layer.append(Neuron(None)) #this is our bias neuron
       layer[-1].setOutput(1)     #our bias will be of 1
       self.layers.append(layer)
```

. Para finalizar esta la clase que trae a nuestros componentes juntos. La clase Network, su aspecto mas complicado es su constructor. Veamos paso por paso. Primero recibimos topology, esto nos define nuestros input layers, hidden layers y la output layer. Decimos que por cada neuron en la topología vamos a darle append en la layer, ahora lo que pasa es que tenemos 1 espacio reservado para nuestro neuron de bias.

```
  def feedForward(self):
    for layer in self.layers[1:]:
      for neuron in layer:
        neuron.feedForward()

  def backPropagate(self, target):
```

```
    for i in range(len(target)):
      self.layers[-1][i].setError(target[i] - self.layers[-1][i].
    getOutput())
    for layer in self.layers[::-1]: #extended slice reverses the
    order of self.layers
      for neuron in layer:
        neuron.backPropagate()
```

Finalmente. Como podemos ver feedforward y backpropagate simplemente llaman a las funciones respectivas de sus neuronas. Para concluir, esta es una red que puede resolver problemas de 0 y 1 por la funcion getThResults que esta definida en networks(El código fuente completo puede ser encontrado en la bibliografia). Las pruebas corridas han sido con adders de 1 bit, and de 1 bit y XOR.

Estrategia de Paralelizacion

```
for i in range(2):
  t= threading.Thread(target=TrainNet, args=(inputs[i], [nets[i]],
    outputs[i], i, result_list))
  threads.append(t)
  t.start()
for i in range(2):
  threads[i].join()
net1= result_list[0]
net2= result_list[1]
```

Implementacion en Paralelo. Lo que he creado es una implementacion de una libreria de una red neuronal. Ahora en el main.py tenemos una muestra de como implementarlo, y el codigo que mostramos arriba es una implementacion en paralelo. Basicamente la estrategia es crear 2 redes a partir de nuestra entrada y separar la mitad

del entendimiento entre ellas. En esencia es tener 2 cerebros mas simples y mas pequeños que son equivalentes al 1 cerebro que entrenamos en el main. Y a partir de la entrada utilizar cualquiera de nuestras 2 redes.

Referencias

A Quick Introduction to Neural Networks. (s.f.). https://ujjwalkarn.me/2016/08/09/quick-intro-neural-networks/. Published: 08-09-2016.

Codigo Fuente. (s.f.). https://github.com/Jose-R-Rodriguez/NeuralNetworkImplementation/. Published: 06-28-2018.

Deep Learning: Feedforward Neural Network. (s.f.). https://towardsdatascience.com/deep-learning-feedforward-neural-network-26a6705dbdc7/. Published: 01-05-2017.

Understanding Activation Functions in Neural Networks. (s.f.). https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0. Published: 03-30-2017.

Understanding Gradient Descent. (s.f.). https://eli.thegreenplace.net/2016/understanding-gradient-descent/. Published: 08-05-2016.

Why should weights of Neural Networks be initialized to random numbers? (s.f.). https://stackoverflow.com/questions/20027598/why-should-weights-of-neural-networks-be-initialized-to-random-numbers. Published: 12-30-2017.