Universidad Autónoma de Baja California

Ingeniería en Computación

Facultad de Ciencias Químicas e Ingeniería

Inteligencia Artificial

Algoritmo NADAM

Alumnos:

Huertas Villegas Cesar 1273328

Urias Vega Juan Daniel 1267333

Zavala Roman Irvin Eduardo 1270771

Grupo: 561

08/04/2022

Periodo 2022-1

Historia

El algoritmo Nadam incorporado en el año 2015 por el científico Timothy Dozat fue creado a partir de la idea de optimizar el algoritmo Adam, combinando cosas de este junto a otro algoritmo de nombre Nesterov, de ahí el nombre “Nadam”.

La idea sale a partir de querer mejorar los dos componentes principales de Adam; el componente de impulso y el componente de la tasa de aprendizaje. El principal problema es que el impulso que tiene Adam es muy inferior al que otros algoritmos poseen actualmente.

Por lo cual se decidió implementar los conocimientos que se tienen del componente de impulso de Nesterov creando así una versión mejorada de este, el cual fue implementado en el algoritmo Adam mejorando así la velocidad de convergencia y la calidad de aprendizaje y creando el algoritmo Nadam.

Explicación

```
import numpy as np
import matplotlib.pyplot as plt
def funcion(x):
    # Extended Rosenbrock function
    # Minima -> f=0 at (1,.....,1)
    n = len(x) # n even
    fvec = np.zeros((n,1))
    idx1 = np.array(range(0,n,2)) # odd index
    idx2 = np.array(range(1,n,2)) # even index
    fvec[idx1]=10.0*(x[idx2]-(x[idx1])**2.0)
    fvec[idx2]=1.0-x[idx1]
    f = fvec.T @ fvec
    return f[0,0]
def gradiente(x):
    # Extended Rosenbrock gradient function
    n = len(x) # n even
    Jf = np.zeros((n,n))
    fvec = np.zeros((n,1))
    idx1 = np.array(range(0,n,2)) # odd index
    idx2 = np.array(range(1,n,2)) # even index

    fvec[idx1]=10.0*(x[idx2]-(x[idx1])**2.0)
    fvec[idx2]=1.0-x[idx1]
    for i in range(n//2):
        Jf[2*i,2*i]     = -20.0*x[2*i]
        Jf[2*i,2*i+1]   = 10.0
        Jf[2*i+1,2*i]   = -1.0
```

```
    gX = np.matmul(2.0*Jf.T,fvec)
    return gX

def nadam(beta_1, beta_2, alpha, tmax, goal, mingrad,eps, n):


    wt = np.zeros((n,1))+50
    mt = np.zeros((n,1))
    vt = np.zeros((n,1))
    mt_gorrito = np.zeros((n,1))
    vt_gorrito = np.zeros((n,1))

    #Para graficar
    t_arreglo = np.array([])
    goal_a = np.array([])
    perf_a = np.array([])

    for t in range(tmax):
        gd = gradiente(wt)
        #vectores anteriores
        mt_gorrito_anterior = mt_gorrito
        #Algoritmo
        mt = beta_1*mt+(1-beta_1)*gd
        vt = beta_2*vt+(1-beta_2)*gd**2
        mt_gorrito = mt/(1-beta_1**(t+1))
        vt_gorrito = vt/(1-beta_2**(t+1))
        wt = wt -
(alpha/(np.sqrt(vt_gorrito)+eps))*(beta_1*mt_gorrito_anterior+((1-beta_
1)/(1-beta_1**(t+1)))*gd)
        perf = funcion(wt)
        if(perf <= goal):
               print("En la iteracion ",t," se alcanzo la precision de
",goal)
            t_arreglo = np.array(range(0,t,1))
            goal_a = np.zeros(t)+goal
            break
        elif(np.linalg.norm(gd) < mingrad):
             print("En la iteracion ",t," se alcanzo el gradiente minimo
de ",mingrad)
            t_arreglo = np.array(range(0,t,1))
            goal_a = np.zeros(t)+goal
            break
        elif(t == tmax-1):
               print("Se alcanzo la maxima cantidad de iteraciones, la
meta no se consiguio :(")
            t_arreglo = np.array(range(0,t,1))
            goal_a = np.zeros(t)+goal
            break
        perf_a = np.append(perf_a, perf)

    plt.yscale("log")
    plt.plot(t_arreglo, goal_a)
    plt.plot(t_arreglo, perf_a)
    plt.show()
    return wt
beta_1 = 0.9
beta_2 = 0.999
alpha = 0.1
```

```
tmax = 10000
goal = 10**-8
mingrad = 10**-10
eps = 10**-8
n = 100

wt = nadam(beta_1, beta_2, alpha, tmax, goal, mingrad,eps, n)
print(wt[0])
print("f(x) final:\n",funcion(wt))
```

Primeramente se tiene las importaciones de librerías, tales como numpy y matplotlib, numpy para poder crear arreglos que servirán como entrada al algoritmo y matplotlib para poder graficar el comportamiento del algoritmo.

Posteriormente se tiene una función para poder retornar la multiplicación de la función por su transpuesta, luego tenemos otra función que es para obtener el gradiente de la función, donde el gradiente es un arreglo de derivadas de las ecuaciones correspondientes.

Luego tenemos una función para el algoritmo de nadam, al cual se le pasan distintos parámetros tales como (beta1, beta2, alpha, tmax, goal, mingrad, eps, n), donde beta1 y beta2, son valores que nos ayudaran para acelerar el algoritmo y que este puede llegar lo más rápido posible a la meta (goal), alpha es el paso inicial (La tasa de aprendizaje), tmax, son las iteraciones máximas que realizará el algoritmo, gola, es el valor al cual se espera llegar cuando ese valor llegue el programa termina, mingrad, es un valor muy pequeño que se utiliza en el algoritmo, eps un valor muy pequeño que se utiliza para evitar las divisiones entre 0, y n es el número variables que tendrá el arreglo para el algoritmo.

Una vez explicados los parámetros se explicara el código, donde primeramente se tiene que inicializar las variables wt, mt, vt, que serán arreglos de n variables y lo mismo para mt_gorrito y vt_gorrito. También se inicializan variables donde se almacenarán los resultados para luego ser graficados.

Luego se realiza un ciclo for, para que se realice el algoritmo las veces necesarias hasta que se cumpla la meta o se lleguen a las interacciones máximas, en donde se tiene que mt es beta1 multiplicado por mt mas 1-beta1 multiplicado por el gradiente

donde mt es representa el momento de la función, luego se tiene vt, que se representa de la misma manera, pero está elevando al cuadrado el gradiente. Luego tenemos mt_gorrito y vt_gorrto que son variables donde nos dice el mínimo de una función y nos ayudarán a acelerar el algoritmo, ambas tienen la misma ecuación pero con sus respectivas variables, para mt_gorrito se divide mt entre 1-beta1 donde beta 1 estará elevado a t+1, donde t es la iteración actual, esto debido a que las iteraciones en python comienzan desde el 0 y esto para que se puede tener tener un valor elevado a la potencia 1, y para vt_gorrito sería la misma fórmula cambiando mt por vt y beta1 por beta2.

$$\hat{V}_t = \frac{V_t}{1-\beta_1^t}$$
$$\hat{S}_t = \frac{S_t}{1-\beta_2^t}$$
$$V_t = \beta_1 V_{t-1} + (1-\beta_1)\frac{\partial L}{\partial w_t}$$
$$S_t = \beta_2 S_{t-1} + (1-\beta_2)\left[\frac{\partial L}{\partial w_t}\right]^2$$

Por último para calcular wt que es un arreglo de n variables que serán modificadas con ayuda de las demás intentando hacer que se acerquen lo más posible al 1 ya sea por la derecha o por la izquierda, es decir un vector de resultados.

Para obtener wt se aplica la siguiente fórmula

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{\hat{S}_t} + \epsilon}\left(\beta_1 \hat{V}_{t-1} + \frac{1-\beta_1}{1-\beta_1^t} \cdot \frac{\partial L}{\partial w_t}\right)$$

donde los valores que previamente se calcularon, se utilizarán para ayudar a ajustar los valores y tratar de aproximarse al 1.

Luego se tiene unas ciertas condiciones que ayudan a saber si se llegó a la meta, máximo de iteraciones o si la norma del gradiente es menor que el mínimo gradiente, todas estas ayudan a detener el ciclo para que este no utilice todas las iteraciones ya que pueden llegar a ser demasiadas cuando ya se llego a un resultado y seria minimo el cambio al realizar dichas iteraciones restantes.

Al final del código se tiene una parte para poder graficar cómo fue el comportamiento del código durante las iteraciones

Conclusión

De todos los algoritmos de optimización basados en gradiente descendiente, NADAM se posiciona como un término medio, sin llegar a la rapidez del Levenberg Marquardt y siendo mejor que el gradiente descendiente original. También se puede observar que este algoritmo es bastante reciente, con una idea bastante sencilla cómo combinar 2 métodos ya conocidos para crear algo que dé mejores resultados, si de por si ADAM se puede ver como una combinación de RMSprop y momentum, NADAM se puede ver como una combinación de ADAM y Nesterov.

Aunque la idea es buena, no se está aplicado lo suficiente en la industria como para marcar NADAM como un algoritmo fundamental o estándar a comparación de ADAM, pero nos demuestra que en los algoritmos de optimización no está todo escrito y existe margen de mejora e innovación a futuro.

Bibliografía

Martínez, J. (2020). *Cómo Entrenar una CNN Usando Nadam en Keras*.

DataSmarts.

https://datasmarts.net/es/como-entrenar-una-cnn-usando-nadam-en-keras/

Ruder, S. (2016). *An overview of gradient descent optimization algorithms*.

Sebastian Ruder.

https://ruder.io/optimizing-gradient-descent/