Ignacio David Vázquez Pérez 218292866

Práctica SGD

Introducción

En el aprendizaje automático, el descenso de gradiente es una técnica de optimización utilizada para calcular los parámetros del modelo (coeficientes y sesgo) para algoritmos como la regresión lineal, la regresión logística, las redes neuronales, etc. En esta técnica, iteramos repetidamente a través del conjunto de entrenamiento y actualizamos el modelo. parámetros de acuerdo con el gradiente del error con respecto al conjunto de entrenamiento.

Ejecutar el descenso de gradiente por lotes con un gran conjunto de datos puede ser muy costoso porque necesitamos reevaluar todo el conjunto de datos de entrenamiento en cada paso de tiempo.

Para obtener resultados precisos a través del descenso de gradiente estocástico, es importante presentarlos con datos en un orden aleatorio, por lo que queremos barajar el conjunto de entrenamiento para cada época para evitar ciclos.

Estos son los pasos de SGD en pseudocódigo:

1. Elija un vector inicial de parámetros y tasa de aprendizaje.
2. Repetir hasta obtener un mínimo aproximado:
    a. Mezcla aleatoriamente ejemplos en el conjunto de entrenamiento.
    b. Para i=1,2,...,n hacer:

        $w := w - \eta \nabla j_i(w)$

        Dónde j es una funcion de costo

En Stochastic Gradient Descent (SGD), no tenemos que esperar para actualizar el parámetro del modelo después de iterar todos los puntos de datos en nuestro conjunto de entrenamiento, sino que simplemente actualizamos los parámetros del modelo después de iterar cada uno de los puntos de datos en nuestro entrenamiento. establecer.

Dado que actualizamos los parámetros del modelo en SGD después de iterar cada punto de datos, aprenderá el parámetro óptimo del modelo más rápido, por lo tanto, una convergencia más rápida, y esto también minimizará el tiempo de entrenamiento.

Desarrollo

1. Importa numpy que sirven para utilizar algoritmos necesarios para reentrenar la red neuronal, operaciones matriciales respectivamente

```
import numpy as np
```

2. Se define la clase

```
# Descenso de gradiente estocástico

class SGD(object):
        def __init__(self, rate = 0.01, niter = 10, shuffle=True):
        self.rate = rate
        self.niter = niter
        self.weight_initialized = False
        # Si es verdadero, baraja los datos de entrenamiento cada época
        self.shuffle = shuffle
```

3. Se define la funcion sigmoide

```
def sigmoid(self, x):
        return 1.0/(1.0 + np.exp(-x))
```

4. De define la funcion para entenar el mdelo, recibe los atributes a predecir y sus etiquetas

```
def fit(self, X, y):
        """
        Ajustar datos de entrenamiento
        X : Vectores de entrenamiento, X.shape : [#muestras, #características]
        y : valores objetivo, forma y : [#muestras]
        """
        # inicializar pesos sinapticos
        self.initialize_weights(X.shape[1])
        # inicializar función de costo
        self.cost = []
```

```
        for i in range(self.niter):
            if self.shuffle:
            X, y = self.shuffle_set(X, y)
            cost = []
            for xi, target in zip(X, y):
            cost.append(self.SGD_method(xi, target))
            avg_cost = sum(cost)/len(y)
            self.cost.append(avg_cost)
        return self
```

5. Se define la función que mezcla el dataset de manera aleatoria

```
def shuffle_set(self, X, y):
        """Mezclar datos de entrenamiento"""
        r = np.random.permutation(len(y))
return X[r], y[r]
```

6. Se inicializan los vectores de los pesos sinápticos con el bias

```
def initialize_weights(self, m):
        """Inicializar pesos a cero"""
        self.weight = np.zeros(1 + m)
        self.weight_initialized = True
```

7. Se genera el metodo del algortimo SGD, donde se pasa de parametro las variables a predecir y sus etiquetas. En la variable “weight” se encuentran los ds pess sinapticos que se actualizan cada vez que se entrena el modelo y se actualiza el costo.

```
def SGD_method(self, xi, target):
        """Aplicar la regla de aprendizaje SGD para actualizar sus pesos sinapticos"""
        output = self.net_input(xi)
        error = (target - output)
        self.weight[1:] += self.rate * xi.dot(error)
        self.weight[0] += self.rate * error
        cost = 0.5 * error**2
        return cost
```

8. En la función predict se realiza la operazion de orodcuto punto y desues se manda a llamar la funcion sigmoide donde se convierten los valoresa un rango de 0 o 1.

```
def predict(self, X):
        Y_pred = []
        for x in X:
            y_pred = self.sigmoid(self.activation(x))
```

```
            Y_pred.append(y_pred)
    return np.array(Y_pred)
```

Conclusiones

Este es un método sencillo que puede resultar beneficioso si se puede mezclar con otros métodos de optimización. Debido a su flexibilidad de implementación, se puede generar diversos tipos de entrenamiento, ya sea por lotes ó mini lotes que ayuden a alcanzar una función de costo de los valores mínimos locales para alcanzar un mejor rendimiento. Las ventajas se pueden atribuir cuando se realizar las operaciones de manera distribuida y paralelizada, de esta manera se puede disminuir e costo computacional ya que de una manera más eficiente se generan lotes y se entrenen modelos al mismo tiempo.

Créditos

- singh, R. (2019, June 5). *Sigmoid Neuron model, Gradient Descent with sample code*. Medium; Analytics Vidhya. https://medium.com/analytics-vidhya/sigmoid-neuron-model-gradient-descent-with-sample-code-4919bfc9d4c4
- Singh, H. (2021, March 15). *Variants of Gradient Descent Algorithm | Types of Gradient Descent*. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/03/variants-of-gradient-descent-algorithm/