Tarea 1
Optimización ciega

Mario Emilio Jiménez Vizcaíno
A01173359@itesm.mx

1 INTRODUCCIÓN

Durante esta década la cantidad de datos generados por las personas y capturados por los dispositivos que usamos diariamente ha crecido exponencialmente, más rápido que la velocidad de los procesadores. Es por eso que los métodos de aprendizaje automático han tomado un papel fundamental en el procesamiento de estos, ya que estos están limitados por el tiempo de cálculo y no por el tamaño de la muestra de datos.

Los algoritmos de optimización como el gradiente descendente estocástico muestran un rendimiento sorprendente para los problemas a gran escala, ya que en promedio tienen un tiempo de ejecución mucho menor que otros algoritmos.

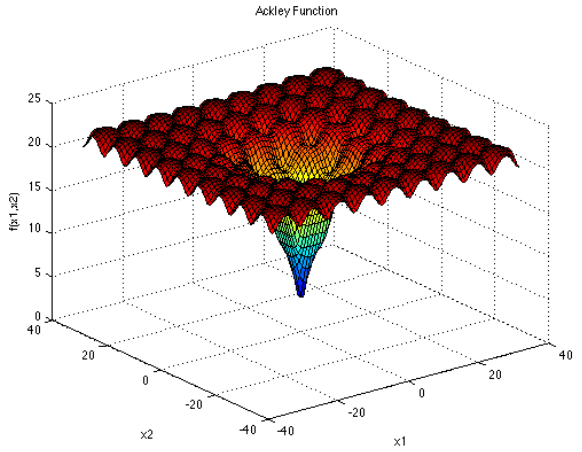
En esta práctica se demuestran 3 tipos variaciones de algoritmos de optimización ciega, además de sus respectivas implementaciones en Python.

2 METODOLOGÍA

Para la demostración de los diferentes algoritmos de mutación se utilizó la búsqueda aleatoria del mínimo en la función Ackley, cuya fórmula general es:

$$f(x) = -a\exp\left(-b\sqrt{\frac{1}{d}\sum_{i=1}^{d}x_i^2}\right) - \exp\left(\frac{1}{d}\sum_{i=1}^{d}\cos(cx_i)\right) + a + \exp(1)$$

Esta función es usada frecuentemente para probar algoritmos de optimización ya que alrededor del punto mínimo global (la coordenada (0, 0)) la gráfica presenta muchos puntos mínimos locales, por lo que los algoritmos de optimización corren el riesgo de quedarse atrapados en uno de estos. En la siguiente imagen se presenta el comportamiento de esta función en dos dimensiones:



En este caso se usó la fórmula con dos dimensiones:

$$f(x) = -a\exp\left(-b\sqrt{\frac{x_1^2 + x_2^2}{d}}\right) - \exp\left(\frac{\cos(cx_1) + \cos(cx_2)}{d}\right) + a + \exp(1)$$

Esta función fue evaluada en el rango $x_1, x_2 \epsilon [-32.768, -32.768]$ con los parámetros $a = 20$, $b = 0.2$ y $c = 2\pi$, utizando tres algoritmos de mutación, descritos a continuación.

La implementación de la fórmula puede ser encontrada en el apéndice B.

2.1 Un padre, varios hijos

La primer variante de algoritmos de mutación se llama "un padre, varios hijos" ya que en cada iteración se preserva sólo un punto (el padre), el cual se muta una cantidad predeterminada de veces para producir los hijos, de los cuales sólo el que tiene la mejor evaluación es conservado para la siguiente iteración del algoritmo.

La implementación de este algoritmo se encuentra en el apéndice C.

2.2 Varios padres, varios hijos

La segunda variante de mutación cambia en el número de puntos que se conservan después de cada iteración, ya que en este se conservan dos o más. Durante la mutación de padres para generar nuevos hijos se seleccionan m veces un padre al azar, para mutarlo y crear un nuevo hijo. Al final, de la lista de m hijos generados se seleccionan los nuevos padres para la siguiente iteración.

La implementación de este algoritmo se encuentra en el apéndice D.

2.3 Varios padres, varios hijos, con traslape generacional

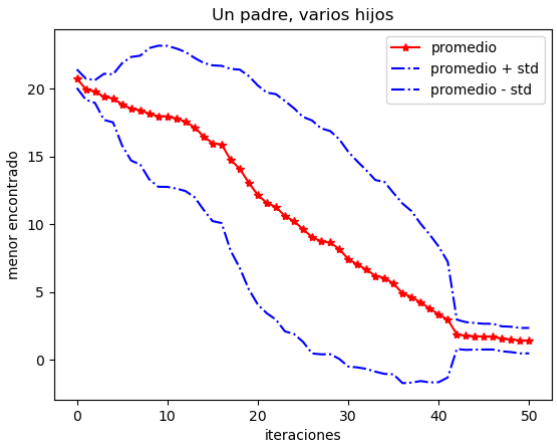
Finalmente, esta última variante del algoritmo de mutación es muy parecida a la anterior, pero cambia en que en el último paso, la selección de nuevos padres para la siguiente iteración, se seleccionan padres de la lista de padres e hijos.

La implementación de este algoritmo se encuentra en el apéndice E.

3 RESULTADOS

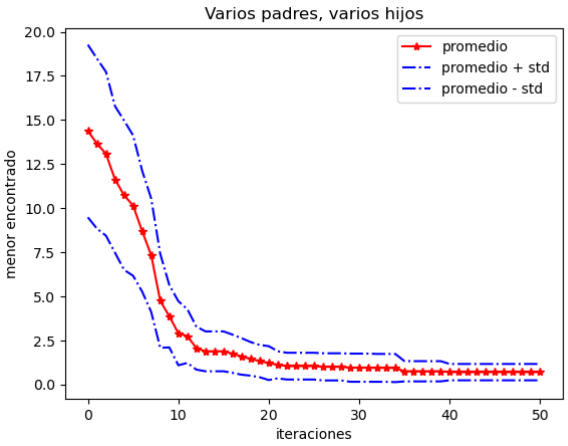
Para cada variante del algoritmo de mutación se creó una clase que iteraba 50 veces, generando 10 hijos en el algoritmo de un padre, varios hijos, y conservando 10 hijos en los algoritmos de varios padres, varios hijos. El código de inicio ejecutó el experimento 10 veces para calcular un promedio y una desviación estándar de estas iteraciones, para finalmente crear una gráfica del progreso del algoritmo al encontrar el punto mínimo global.

3.1 Un padre, varios hijos



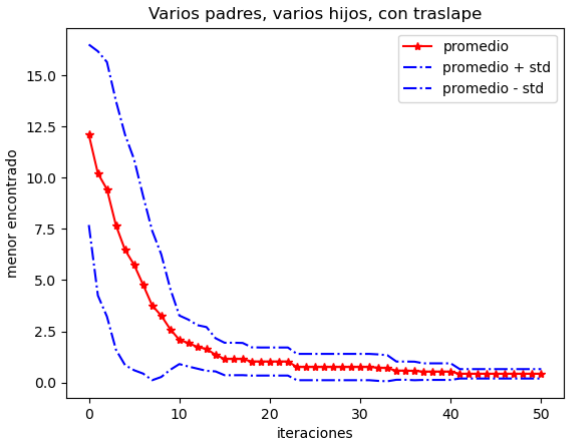
En esta primera gráfica se puede observar que muchos de los experimentos variaron demasiado entre las iteraciones 10 y 30, pero después la mayoría encontró el mínimo global de la función.

3.2 Varios padres, varios hijos



En el segundo algoritmo, muchos de los experimentos encontraron rápidamente el mínimo global, aunque probablemente un par de ellos no haya llegado tan rápido, provocando el aumento de la desviación estándar hasta el final.

3.3 Varios padres, varios hijos, con traslape generacional



Finalmente, este algoritmo, a pesar de que debería ser un poco mejor que el anterior tuvo un rendimiento comparable con éste, al igual conservando un poco de desviación estándar hasta el final de las 50 iteraciones.

4 CONCLUSIÓN

En esta práctica se demostró la utilidad y una implementación de algunos de los algoritmos de mutación, además de los resultados que podríamos esperar al seleccionar uno de estos. También se mostró que los algoritmos con traslape generacional, a pesar de tener buenos fundamentos teóricos ya que permiten que los padres que tuvieron una muy buena evaluación sean seleccionados para la siguiente iteración, a veces puede ser preferible un algoritmo de mutación sin traslape para así generar más hijos que implementen mutaciones que probablemente sean beneficiosas para avanzar en la optimización del problema.

A CÓDIGO DE ENTRADA PARA LOS TRES ALGORITMOS

```python
import logging
import sys

import matplotlib.pyplot as plt
import pandas as pd

from AckleyFunction import ackley_fun
from UP_VH import UnPadreVariosHijos
from VP_VH import VariosPadresVariosHijos
from VP_VH_T import VariosPadresVariosHijosTraslape


def seleccionar_modelo():
    if len(sys.argv) < 2:
        return None
    modeloIdx = int(sys.argv[1])
    if modeloIdx == 1:
        return UnPadreVariosHijos
    if modeloIdx == 2:
        return VariosPadresVariosHijos
    if modeloIdx == 3:
        return VariosPadresVariosHijosTraslape
    return None


modelo = seleccionar_modelo()
if modelo is None:
    print(
        "Introduzca un argumento {1, 2, 3} para seleccionar un algoritmo de mutación")
    exit(1)
dfExperimentos = pd.DataFrame()
experimentos = 10

logging.basicConfig(level=logging.ERROR,
                    format='%(asctime)s - %(levelname)s - %(funcName)s - %(message)s')
logger = logging.getLogger()

for experimento in range(1, experimentos + 1):
    algoritmo = modelo(ackley_fun)
    mejores, evaluaciones = algoritmo.run()

    cantidad = len(mejores)

    logger.info('Experimento %d', experimento)
    logger.info('Cantidad de soluciones %d', cantidad)
    logger.info('Cantidad de evaluaciones %d', len(evaluaciones))

    df = pd.DataFrame()
    df['algoritmo'] = [algoritmo.__class__.__name__] * cantidad
    df['experimento'] = [experimento] * cantidad
    df['iteracion'] = list(range(cantidad))
    df['x'] = mejores
    df['evaluacion'] = evaluaciones
    df.at[0, 'menor'] = df.loc[0]['evaluacion']
    for rowidx in range(1, df.shape[0]):
        df.at[rowidx, 'menor'] = min(
            df.loc[rowidx]['evaluacion'], df.iloc[rowidx - 1].menor)
    dfExperimentos = dfExperimentos.append(df)

dfExperimentos.reset_index(drop=True, inplace=True)

resultados = dfExperimentos.groupby(
    'iteracion').agg({'menor': ['mean', 'std']})
promedios = resultados['menor']['mean'].values
std = resultados['menor']['std'].values
plt.plot(range(cantidad), promedios, color='red', marker='*')
plt.plot(range(cantidad), promedios + std, color='b', linestyle='-.')
plt.plot(range(cantidad), promedios - std, color='b', linestyle='-.')
plt.xlabel('iteraciones')
plt.ylabel('menor encontrado')
plt.legend(['promedio', 'promedio + std', 'promedio - std'])

if modelo is UnPadreVariosHijos:
    plt.title('Un padre, varios hijos')
    plt.savefig('upvh.png')
elif modelo is VariosPadresVariosHijos:
    plt.title('Varios padres, varios hijos')
    plt.savefig('vpvh.png')
elif modelo is VariosPadresVariosHijosTraslape:
    plt.title('Varios padres, varios hijos, con traslape')
    plt.savefig('vpvht.png')
```

B IMPLEMENTACIÓN DE LA FUNCIÓN ACKLEY PARA UNA LISTA $[x_1, x_2]$

```python
import math


def ackley_fun(x):
    a = 20
    b = 0.2
    c = 2 * math.pi
    d = 2
    term1 = -a * math.exp(-b * math.sqrt((x[0] ** 2 + x[1] ** 2) / d))
    term2 = -math.exp((math.cos(c * x[0]) + math.cos(c * x[1])) / d)
    return term1 + term2 + a + math.e
```

C IMPLEMENTACIÓN DEL ALGORITMO DE MUTACIÓN "UN PADRE, VARIOS HIJOS"

```python
import logging

import numpy as np


class UnPadreVariosHijos:

    def __init__(self, eval_func, iterations=50, nChildren=10):
        self.eval_func = eval_func
        self.iter = iterations
        self.nChildren = nChildren
        self.parent = np.random.uniform(-32.768, 32.768, (2))
        self.best_iter = [self.parent]
        self.evaluations = [self.eval_func(self.parent)]
        self.logger = logging.getLogger()
        self.logger.info('Initializing algorithm with parent {} and {} children'.format(
            self.parent, self.nChildren))

    def evaluate(self, children):
        self.logger.debug('Evaluando hijos: {}'.format(children))
        return list(map(self.eval_func, children))

    def mutation(self):
        children = self.parent + \
            np.random.normal(scale=2, size=(self.nChildren, 2))
        return np.clip(children, -32.768, 32.768)

    def run(self):
        for _ in range(self.iter):
            children = self.mutation()
            self.logger.debug('Children are {}'.format(children))
            evaluations = self.evaluate(children)
            self.logger.debug('Evaluations are {}'.format(evaluations))
            self.parent = children[np.argmin(evaluations)]
            self.best_iter.append(self.parent)
            self.evaluations.append(self.eval_func(self.parent))
            self.logger.info('New parent is {} with evaluation of {}'.format(
                self.parent, self.evaluations[-1]))

        return self.best_iter, self.evaluations


if __name__ == '__main__':
    from AckleyFunction import ackley_fun
    logging.basicConfig(level=logging.INFO,
                        format='%(asctime)s - %(levelname)s - %(funcName)s - %(message)s')
    algoritmo = UnPadreVariosHijos(ackley_fun)
    algoritmo.run()
```

D IMPLEMENTACIÓN DEL ALGORITMO DE MUTACIÓN ”VARIOS PADRES, VARIOS HIJOS”

```python
import logging

import numpy as np


class VariosPadresVariosHijos:

    def __init__(self, eval_func, iterations=50, nChildren=10):
        self.eval_func = eval_func
        self.iter = iterations
        self.nChildren = nChildren
        self.parents = np.random.uniform(-32.768, 32.768, (nChildren, 2))

        self.logger = logging.getLogger()
        self.logger.info('Initializing algorithm with parents {} and {} children'.format(
            self.parents, self.nChildren))

        evaluations = self.evaluate(self.parents)
        self.best_iter = [self.parents[np.argmin(evaluations)]]
        self.evaluations = [evaluations[np.argmin(evaluations)]]

    def evaluate(self, children):
        self.logger.debug('Evaluando hijos: {}'.format(children))
        return list(map(self.eval_func, children))

    def mutation(self):
        def get_random_parent():
            return self.parents[np.random.randint(0, len(self.parents))]

        def mutate(p):
            return p + np.random.normal(scale=2, size=(1, 2))
        childrenList = [mutate(get_random_parent())
                        for _ in range(2 * len(self.parents))]
        children = np.concatenate(childrenList, axis=0)
        return np.clip(children, -32.768, 32.768)

    def run(self):
        for _ in range(self.iter):
            children = self.mutation()
            self.logger.debug('Children are {}'.format(children))
            evaluations = self.evaluate(children)
            self.logger.debug('Evaluations are {}'.format(evaluations))
            sortedIdxs = np.argsort(evaluations)
            self.parents = children[sortedIdxs][0:self.nChildren]
            self.best_iter.append(children[sortedIdxs[0]])
            self.evaluations.append(evaluations[sortedIdxs[0]])
            self.logger.info('New parents are {} with best evaluation of {}'.format(
                self.parents, self.evaluations[-1]))

        return self.best_iter, self.evaluations


if __name__ == '__main__':
    from AckleyFunction import ackley_fun
    logging.basicConfig(level=logging.INFO,
                        format='%(asctime)s - %(levelname)s - %(funcName)s - %(message)s')
    algoritmo = VariosPadresVariosHijos(ackley_fun)
    algoritmo.run()
```

E IMPLEMENTACIÓN DEL ALGORITMO DE MUTACIÓN ”VARIOS PADRES, VARIOS HIJOS, CON TRASLAPE GENERACIONAL”

```python
import logging

import numpy as np


class VariosPadresVariosHijosTraslape:

    def __init__(self, eval_func, iterations=50, nChildren=10):
        self.eval_func = eval_func
        self.iter = iterations
        self.nChildren = nChildren
        self.parents = np.random.uniform(-32.768, 32.768, (nChildren, 2))

        self.logger = logging.getLogger()
        self.logger.info('Initializing algorithm with parents {} and {} children'.format(
            self.parents, self.nChildren))

        evaluations = self.evaluate(self.parents)
        self.best_iter = [self.parents[np.argmin(evaluations)]]
        self.evaluations = [evaluations[np.argmin(evaluations)]]

    def evaluate(self, children):
        self.logger.debug('Evaluando hijos: {}'.format(children))
        return list(map(self.eval_func, children))

    def mutation(self):
        def get_random_parent():
            return self.parents[np.random.randint(0, len(self.parents))]

        def mutate(p):
            return p + np.random.normal(scale=2, size=(1, 2))
        childrenList = [mutate(get_random_parent())
                        for _ in range(2 * len(self.parents))]
        children = np.concatenate(childrenList, axis=0)
        return np.clip(children, -32.768, 32.768)

    def run(self):
        for _ in range(self.iter):
            children = self.mutation()
            self.logger.debug('Children are {}'.format(children))
            points = np.concatenate((children, self.parents))
            evaluations = self.evaluate(points)
            self.logger.debug('Evaluations are {}'.format(evaluations))
            sortedIdxs = np.argsort(evaluations)
            self.parents = points[sortedIdxs][0:self.nChildren]
            self.best_iter.append(points[sortedIdxs[0]])
            self.evaluations.append(evaluations[sortedIdxs[0]])
            self.logger.info('New parents are {} with best evaluation of {}'.format(
                self.parents, self.evaluations[-1]))

        return self.best_iter, self.evaluations


if __name__ == '__main__':
    from AckleyFunction import ackley_fun
    logging.basicConfig(level=logging.INFO,
                        format='%(asctime)s - %(levelname)s - %(funcName)s - %(message)s')
    algoritmo = VariosPadresVariosHijosTraslape(ackley_fun)
    algoritmo.run()
```