Intelligenza Artificiale

Anno Accademico 2022 - 2023

Esercizi in Python:

Algoritmi Genetici

(caso continuo)

Norme di utilizzo dei materiali didattici

La visione e l'utilizzo del presente materiale didattico è riservato agli utenti iscritti al corso al solo fine di studio e approfondimento didattico.

L'utente che accede alla sezione e-learning e che ne consulta i contenuti è tenuto a rispettare le disposizioni legislative che tutelano il diritto d'autore e pertanto è fatto espresso divieto di riprodurre, pubblicare o distribuire i materiali tratti dal presente sito, anche in forma parziale, fatta salva la possibilità di realizzare un'unica copia del materiale, in formato cartaceo e/o digitale, all'esclusivo fine di studio.

L'utente è responsabile per l'uso improprio o non autorizzato del materiale pubblicato effettuato in violazione delle suddette disposizioni.

Algoritmo Genetico
Caso di Funzione Continua

- Definiamo una funzione continua da minimizzare, ad esempio la seguente:

$$f(x, y) = x^2 + y^2$$

- Tale funzione ha il seguente minimo globale:

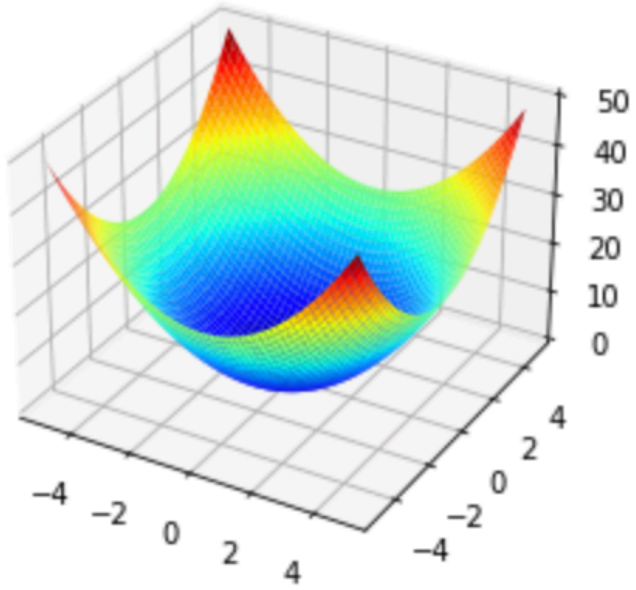
$$f(0, 0) = 0$$

- Possiamo definire questa funzione obiettivo come segue:

```
def objective(x):
    return x[0]**2.0 + x[1]**2.0
```

Algoritmo Genetico
Caso di Funzione Continua

- Rappresentazione grafica della funzione:



Algoritmo Genetico
Caso di Funzione Continua

- Vogliamo minimizzare questa funzione con un algoritmo genetico.

- Innanzitutto, dobbiamo definire i limiti di ciascuna variabile di input della funzione ***f***:

$$-5 \leq x \leq 5$$

$$-5 \leq y \leq 5$$

- In Python:

```
bounds = [[-5.0, 5.0], [-5.0, 5.0]]
```

Algoritmo Genetico
Definizione Iperparametri

- Definiamo un iperparametro **`n_bits`** che indica il numero di bit per ciascuna variabile di input nella funzione obiettivo. Per il nostro esempio impostiamolo a 16 bit:

```
n_bits = 16
```

- Ciò significa che la nostra stringa di bit effettiva avrà (16 × 2) = 32 bit, date le due variabili di input.

- Dobbiamo definire di conseguenza il nostro tasso di mutazione:

```
r_mut = 1.0 / (float(n_bits) * len(bounds))
```

Algoritmo Genetico
Creazione popolazione iniziale

- Successivamente, dobbiamo assicurarci che la popolazione iniziale crei stringhe di bit casuali della giusta lunghezza:

```
pop = [randint(0, 2, n_bits*len(bounds)).tolist() for _ in range(n_pop)]
```

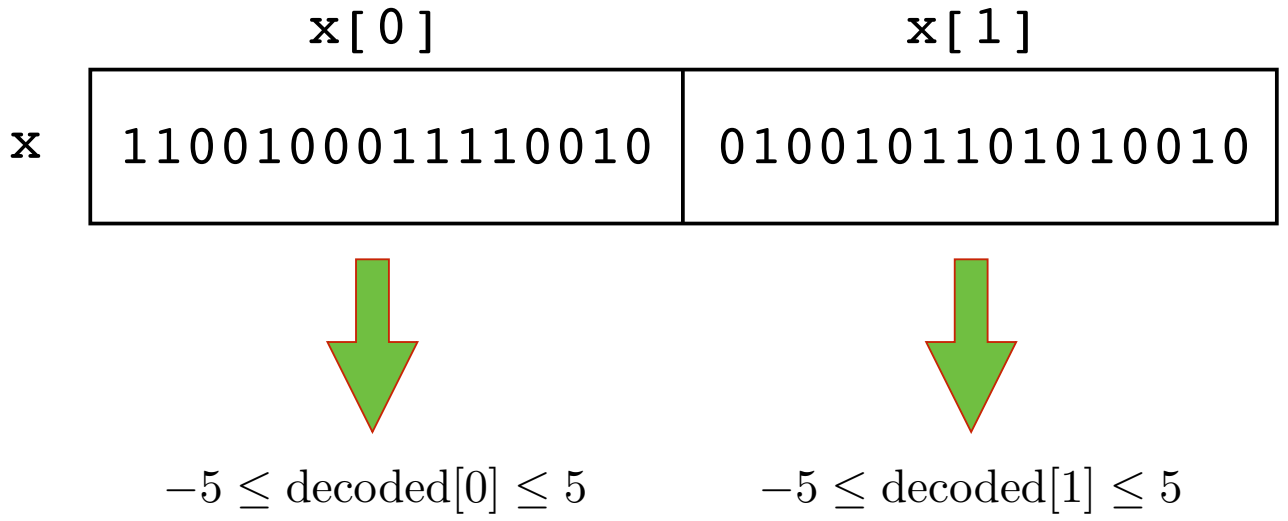
- Esempio:

In [10]:
```
bounds = [[-5.0, 5.0], [-5.0, 5.0]]
n_bits = 12
n_pop = 10
pop = [randint(0, 2, n_bits*len(bounds)).tolist() for _ in range(n_pop)]
pop
```

Out[10]:
```
[[0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1],
 [1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0],
 [0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0],
 [1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0],
 [1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1],
 [1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1],
 [0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1],
 [0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0],
 [0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1],
 [0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1]]
```

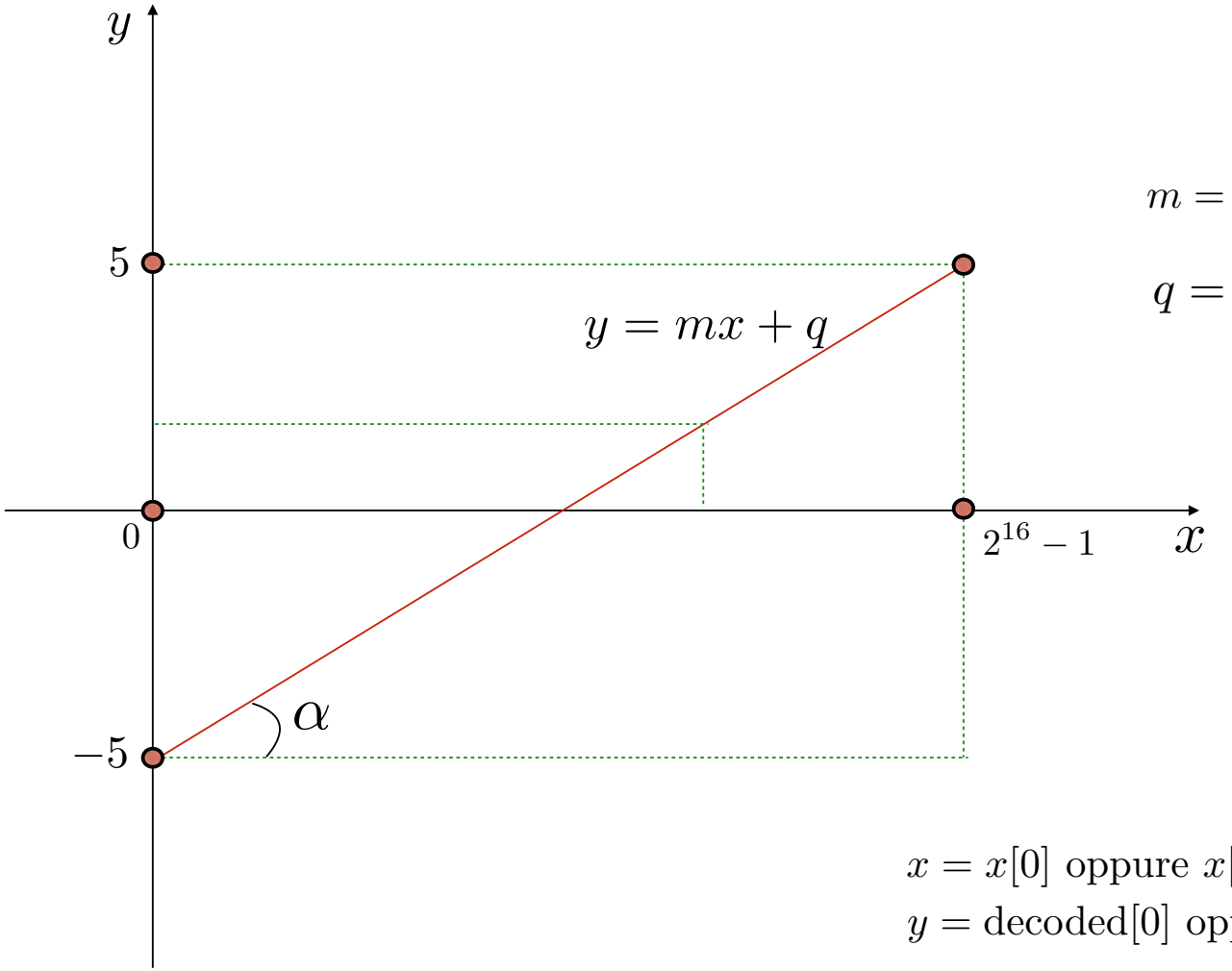
Algoritmo Genetico
Decodifica

- Per il calcolo della funzione occorre decodificare da binario a reale le due sottostringhe:



Algoritmo Genetico

Decodifica



$$m = \frac{10}{2^{16}-1}$$

$$q = -5$$

$x = x[0]$ oppure $x[1]$

$y = \text{decoded}[0]$ oppure $\text{decoded}[1]$

Algoritmo Genetico
Decodifica

- La funzione **`decode()`** effettua le suddette elaborazioni:

In [ ]:
```
def decode(bounds, n_bits, bitstring):
    decoded = list()
    largest = 2**n_bits - 1
    for i in range(len(bounds)):
        # extract the substring
        start, end = i * n_bits, (i * n_bits)+n_bits
        substring = bitstring[start:end]
        # convert bitstring to a string of chars
        chars = ''.join([str(s) for s in substring])
        # convert string to integer
        integer = int(chars, 2)
        # scale integer to desired range
        value = bounds[i][0] + (integer/largest) * (bounds[i][1] - bounds[i][0])
        # store
        decoded.append(value)
    return decoded
```

Algoritmo Genetico
Decodifica

- Possiamo quindi chiamare questo all'inizio di ogni ciclo dell'algoritmo per decodificare la popolazione, per poi valutare la funzione:

```
# decode population
decoded = [decode(bounds, n_bits, p) for p in pop]
# evaluate all candidates in the population
scores = [objective(d) for d in decoded]
```

Algoritmo Genetico
Tournament Selection

In [ ]:
```python
# tournament selection
def selection(pop, scores, k=3):
    # first random selection
    selection_ix = randint(len(pop))
    for ix in randint(0, len(pop), k-1):
        # check if better (e.g. perform a tournament)
        if scores[ix] < scores[selection_ix]:
            selection_ix = ix
    return pop[selection_ix]
```

Algoritmo Genetico
Crossover

In [ ]:
```
def crossover(p1, p2, r_cross):
    # children are copies of parents by default
    c1, c2 = p1.copy(), p2.copy()
    # check for recombination
    if rand() < r_cross:
        # select crossover point that is not on the end of the string
        pt = randint(1, len(p1)-2)
        # perform crossover
        c1 = p1[:pt] + p2[pt:]
        c2 = p2[:pt] + p1[pt:]
    return [c1, c2]
```

Algoritmo Genetico
Mutation

In [ ]:
```
# mutation operator
def mutation(bitstring, r_mut):
    for i in range(len(bitstring)):
        # check for a mutation
        if rand() < r_mut:
            # flip the bit
            bitstring[i] = 1 - bitstring[i]
```

Algoritmo Genetico
Codice (1a parte)

In [ ]:
```
def genetic_algorithm(objective, bounds, n_bits, n_iter, n_pop, r_cross, r_mut):
    # initial population of random bitstring
    pop = [randint(0, 2, n_bits*len(bounds)).tolist() for _ in range(n_pop)]
    # keep track of best solution
    best, best_eval = 0, objective(decode(bounds, n_bits, pop[0]))
```

Algoritmo Genetico
Codice (2a parte)

```
# enumerate generations
for gen in range(n_iter):
    # decode population
    decoded = [decode(bounds, n_bits, p) for p in pop]
    # evaluate all candidates in the population
    scores = [objective(d) for d in decoded]
    # check for new best solution
    for i in range(n_pop):
        if scores[i] < best_eval:
            best, best_eval = pop[i], scores[i]
            print(">%d, new best f(%s) = %.3f" % (gen,  decoded[i], scores[i]))
    # select parents
    selected = [selection(pop, scores) for _ in range(n_pop)]
```

Algoritmo Genetico
Codice (3a parte)

```
        # create the next generation
        children = list()
        for i in range(0, n_pop, 2):
            # get selected parents in pairs
            p1, p2 = selected[i], selected[i+1]
            # crossover and mutation
            for c in crossover(p1, p2, r_cross):
                # mutation
                mutation(c, r_mut)
                # store for next generation
                children.append(c)
        # replace population
        pop = children
    return [best, best_eval]
```

Problema con Funzione Continua
Inizializzazione Iperparametri

- Definiamo gli iperparametri come segue:

In [ ]:
```
# define range for input
bounds = [[-5.0, 5.0], [-5.0, 5.0]]
# define the total iterations
n_iter = 100
# bits
n_bits = 16
# define the population size
n_pop = 100
# crossover rate
r_cross = 0.9
# mutation rate
r_mut = 1.0 / (float(n_bits) * len(bounds))
```

Problema con Funzione Continua
Esecuzione dell'Algoritmo

- Eseguiamo l'algoritmo, passandogli la funzione obiettivo **`objective`** che abbiamo definita:

In [ ]:
```
best, score = genetic_algorithm(objective, bounds, n_bits, n_iter, n_pop, r_cross, r_mut)
print('Done!')
decoded = decode(bounds, n_bits, best)
print('f(%s) = %f' % (decoded, score))
```

Problema con Funzione Continua
Esecuzione Algoritmo

```
>0, new best f([-0.87982177734375, 3.613128662109375]) = 13.829
>0, new best f([0.746307373046875, -3.32366943359375]) = 11.604
>0, new best f([0.07110595703125, 0.619659423828125]) = 0.389
>0, new best f([-0.084381103515625, 0.124359130859375]) = 0.023
>5, new best f([0.02044677734375, 0.09063720703125]) = 0.009
>6, new best f([0.016937255859375, -0.046081542968750]) = 0.002
>9, new best f([0.016937255859375, 0.0146484375]) = 0.001
>11, new best f([0.000152587890625, 0.0152587890625]) = 0.000
>12, new best f([0.000152587890625, 0.009765625]) = 0.000
>12, new best f([0.000152587890625, 0.0]) = 0.000
>34, new best f([0.0, 0.0]) = 0.000
Done!
f([0.0, 0.0]) = 0.000000
```

Problema con Funzione Continua

Altra Esecuzione Algoritmo

```
>0, new best f([1.071014404296875, 0.44403076171875]) = 1.344
>0, new best f([0.687103271484375, -0.24169921875]) = 0.531
>0, new best f([0.609588623046875, -0.219268798828125]) = 0.420
>1, new best f([-0.476226806640625, -0.32318115234375]) = 0.331
>1, new best f([0.407257080078125, 0.089569091796875]) = 0.174
>2, new best f([0.279541015625, 0.13580322265625]) = 0.097
>2, new best f([0.1361083984375, 0.24932861328125]) = 0.081
>2, new best f([-0.054473876953125, -0.219268798828125]) = 0.051
>5, new best f([-0.0531005859375, 0.08941650390625]) = 0.011
>5, new best f([-0.053558349609375, 0.088043212890625]) = 0.011
>6, new best f([0.023345947265625, 0.088043212890625]) = 0.008
>6, new best f([-0.0537109375, 0.05523681640625]) = 0.006
>9, new best f([0.000457763671875, -0.071563720703125]) = 0.005
>9, new best f([0.023956298828125, 0.045623779296875]) = 0.003
>11, new best f([0.024261474609375, 0.035858154296875]) = 0.002
>12, new best f([0.028839111328125, 0.02655029296875]) = 0.002
>13, new best f([0.023040771484375, 0.013427734375]) = 0.001
>14, new best f([0.023345947265625, 0.008544921875]) = 0.001
>14, new best f([0.0079345703125, 0.01129150390625]) = 0.000
>15, new best f([0.0079345703125, 0.01007080078125]) = 0.000
>15, new best f([0.007476806640625, 0.008544921875]) = 0.000
>16, new best f([0.003662109375, 0.01007080078125]) = 0.000
>17, new best f([0.004119873046875, 0.006103515625]) = 0.000
>18, new best f([0.00335693359375, 0.003814697265625]) = 0.000
>19, new best f([0.003662109375, 0.002288818359375]) = 0.000
>19, new best f([0.001983642578125, 0.00091552734375]) = 0.000
>22, new best f([0.000762939453125, 0.001068115234375]) = 0.000
>24, new best f([0.000762939453125, 0.0]) = 0.000
>26, new best f([0.0, 0.00030517578125]) = 0.000
>34, new best f([0.0, 0.000152587890625]) = 0.000
>37, new best f([0.0, 0.0]) = 0.000
Done!
f([0.0, 0.0]) = 0.000000
```

Riferimenti

Wirsansky, E. *Hands-On Genetic Algorithms with Python*, Packt, 2020.

Brownlee, J. *Optimization for Machine Learning - Finding Function Optima with Python*, Machine Learning Mastery, 2021.

Gridin, I. *Learning Genetic Algorithms with Python*, BPB Publications, 2021.

Luke, S. *Essentials of Metaheuristics*, Second Edition, 2013.