```
In [1]: import numpy as np
```

CHEM 277B: Homework 3 - Meta-Heuristic Algorithms

1. Genetic Algorithms

Use GA to determine the maximum of the solution $f(x) = -x^2 + 8x + 15$ over the discrete range of x-values: [0,15], where $f(x)$ is the fitness function. Consider the following two possible encodings.

Encoding A					
Solution	**Fitness**	**Vector**	**Solution**	**Fitness**	**Vector**
0	15	1011	8	15	0100
1	22	0011	9	6	1100
2	27	1001	10	-5	0101
3	30	1000	11	-18	0110
4	31	0010	12	-33	0111
5	30	0001	13	-50	1101
6	27	0000	14	-69	1110
7	22	1010	15	-90	1111

Encoding B					
Solution	**Fitness**	**Vector**	**Solution**	**Fitness**	**Vector**
0	15	0001	8	15	1010
1	22	0010	9	6	0110
2	27	0100	10	-5	0111
3	30	1101	11	-18	0011
4	31	1011	12	-33	1000
5	30	1111	13	-50	1110
6	27	1001	14	-69	0000
7	22	1100	15	-90	0101

(a) List all good solutions as those with $f(x) > 27$, and write out individual solutions for each encodings. Define schema by looking for perfect conservation along each column; if perfect conservation holds, give that value for that position, else represent that column with a *. What is length and order of the two schema? Which encoding will you choose?

Encoding A

Solution	Fitness	Vector
3	30	1000
4	31	0010
5	30	0001

Encoding B

Solution	Fitness	Vector
3	30	1101
4	31	1011
5	30	1111

Schema

Schema	Order	Length
$*0**$	1	0
$1**1$	2	3

I will choose encoding A.

(b) Assume we draw candidate solutions $x = 10, x = 1, x = 15, x = 6, x = 0, x = 9$ as our initial population for GA optimization. Using your chosen encoding, list encoded solutions and their fitness. Pair the population so that the fittest members are paired with the least fit.

Encoding A:

Pair	Solution	Fitness	Vector
A	6	27	0000
A	15	-90	1111
B	1	22	0011
B	10	-5	0101
C	0	15	1011
C	9	6	1100

(c) Use the cross-over operator by defining the cross-over point between the 1st and 2nd element. Exchange the last three elements between the pairs of strings. Do we have new solutions, and if so, what is their fitness? Have we increased fitness of population as a whole? What is the best solution?

Cross-over Between 1st and 2nd elements of Same Pair

Pair	Old Solution	Old Fitness	Old Vector	New Vector	New Solution	New Fitness
A	6	27	0000	0111	12	-33
A	15	-90	1111	1000	3	30
B	1	22	0011	0101	10	-5
B	10	-5	0101	0011	1	22
C	0	15	1011	1100	9	6
C	9	6	1100	1011	0	15

New solutions

Vector	Solution	Fitness
0111	12	-33
1000	3	30

In [2]:
```
print(f'old = {np.mean([27, -90, 22, -5, 15, 6])}')
print(f'new = {np.mean([-33, 30, -5, 22, 6, 15])}')
```

```
old = -4.166666666666667
new = 5.833333333333333
```

We have increased our average fitness from -4.17 to 5.83. The best solution is from new vector 1000 (solution 3: fitness 30).

(d) Using population generated in (c), mutate 3rd element. Do we have new solutions, and if so, what are their fitness? Does mutation increase fitness of population as a whole? Did we find a better solution?

Mutate 3rd Element

Pair	Old Vector	Old Solution	Old Fitness	New Vector	New Solution	New Fitness
A	0111	12	-33	0101	10	-5
A	1000	3	30	1010	7	22
B	0101	10	-5	0111	12	-33
B	0011	1	22	0001	5	30
C	1100	9	6	1110	14	-69
C	1011	0	15	1001	2	27

New solutions

Vector	Solution	Fitness
1010	7	22
0001	5	30
1110	14	-69
1001	2	27

In [3]:
```
print(f'old = {np.mean([-33, 30, -5, 22, 6, 15])}')
print(f'new = {np.mean([-5, 22, -33, 30, -69, 27])}')
```

```
old = 5.833333333333333
new = -4.666666666666667
```

The overall fitness decreased and a better solution was not found compared to the original population.

e) Using population generated in (d), eliminate least fit member, and replace with cloned best member. Do 2-point cross-over by exchanging inner two elements between pairs. Do we have new solutions and what are their fitness? Have we increased fitness of population as a whole? Did we find a better solution?

Replace least fit member with cloned best member

Pair	Vector	Solution	Fitness
A	0101	10	-5
A	1010	7	22
B	0111	12	-33
B	0001	5	30
C	~1110~ 0001	~14~ 5	~-69~ 30
C	1001	2	27

2-Point Cross-Over of Inner Two Elements of Pair

Pair	Old Vector	Old Solution	Old Fitness	New Vector	New Solution	New Fitness
A	0101	10	-5	0011	1	22
A	1010	7	22	1100	9	6
B	0111	12	-33	0001	5	30
B	0001	5	30	0111	12	-33
C	0001	5	30	0001	5	30
C	1001	2	27	1001	2	27

In [4]:
```
print(f'old = {np.mean([-5, 22, -33, 30, -69, 27])}')
print(f'new = {np.mean([22, 6, 30, -33, 30, 27])}')
```

```
old = -4.666666666666667
new = 13.666666666666666
```

The average fitness of the population increased to 13.67. The best solution is still 5, so a solution with better fitness was not found.

(f) Using population generated in (e), eliminate least fit member, and replace with cloned best member. Do cross-over between 3rd and 4th elements and exchange first three elements between pairs. Do we have new solutions, and what are their fitness? Have we increased the fitness of population as a whole? Did we find a better solution?

Replace least fit member with cloned best member

Pair	Vector	Solution	Fitness
A	0011	1	22
A	1100	9	6
B	0001	5	30
B	~0111~ 0001	~12~ 5	~-33~ 30
C	0001	5	30
C	1001	2	27

Cross-over Between 3rd and 4th Elements

Pair	Old Vector	Old Solution	Old Fitness	New Vector	New Solution	New Fitness
A	0011	1	22	1101	13	-50
A	1100	9	6	0010	4	31
B	0001	5	30	0001	5	30
B	0001	5	30	0001	5	30
C	0001	5	30	1001	2	27
C	1001	2	27	0001	5	30

New solutions

Vector	Solution	Fitness
1101	13	-50
0010	4	31

In [5]:
```
print(f'old = {np.mean([22, 6, 30, -33, 30, 27])}')
print(f'new = {np.mean([-50, 31, 30, 30, 27, 30])}')
```

```
old = 13.666666666666666
new = 16.333333333333332
```

The fitness of the population increased as a whole, and a better solution was found. Solution 4 has a fitness of 31, the highest fitness of all of the solutions.

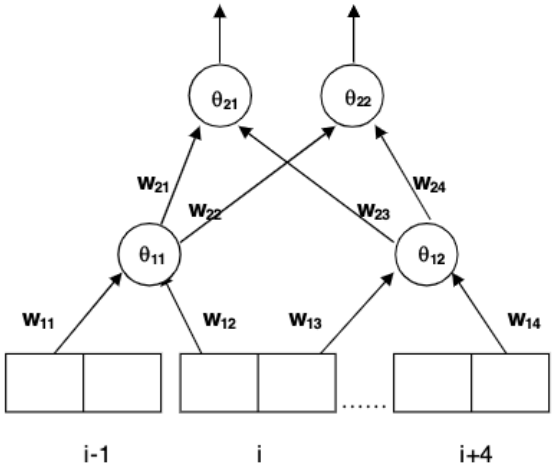
(g) Do you think that the encoding of the solution space was adequate? Why?

I think Encoding A is adequate because it increased the average fitness of the group after most of the genetic changes (cross-over and mutation), and was eventually able to produce the true maximum which was 31, solution 4.

2. Artificial Neural Networks

In lecture we saw that ANN can be used for secondary structure prediction of alpha-helix, beta-sheet, or random coil given the amino acid sequence as input. Each amino acid is represented by two numbers, the first representing its propensity to be hydrophobic (+1) or hydrophilic (-1), while the second is its propensity to form helix (+1) or not helix

(-1). A helix is predicted by the network if the output is (1,-1), β−sheet if output is (-1,1), and coil if (-1,-1). We could design two simple Boolean functions as part of a bigger network with the following connectivity:



Please code up your own little neural network using NumPy! If you choose to do it with code, to make life easier you can also treat this architecture as a fully connected network of shape (6,2,2).

(a) Initialize the weights to random values between 0 and 1.0

In [6]:
```
# Neural Network Class
class NN():
    def __init__(self, architecture, learning_rate, activation):
        # initialize the model
        self.arch = architecture # architecture of the network
        self.activation = activation # activation functions
        self.learning_rate = learning_rate
        self.depth = len(self.arch) # depth of the network


    # initialize the weights and biases
    def init_weights(self):
        self.weights = []
        self.biases = []
        for i in range(self.depth - 1):
            prev_layer = self.arch[i]
            current_layer = self.arch[i + 1]
            self.weights.append(np.random.rand(prev_layer, current_layer))
            self.biases.append(np.random.rand(current_layer))


    # feedforward through the network with matrix multiplication
    def feed_forward(self, x):
        self.z_s = []
        self.a_s = [x]
        for i in range(self.depth - 1):
            z_l = self.a_s[i].dot(self.weights[i]) + self.biases[i]
            self.z_s.append(z_l)
            a_l = self.activation(z_l)
            self.a_s.append(a_l)
        return self.a_s[-1]


    # backpropagation
    def back_prop(self):
        for i in range(self.depth - i):
            self.weights[i] -= self.learning_rate * self.weights_grad[i]
            self.biases[i] -= self.learning_rate * self.biases_grad[i]


    # fit the model
    def fit(self, x, y, activation_grad):
        self.feed_forward(x)
        self.calc_error(y, activation_grad)
```

```
        self.calc_grad()
        self.back_prop()

    # predict the output
    def predict(self, x):
        return self.feed_forward(x)

    def calc_error(self, y, activation_grad):
        # activation_grad is the gradient of the activation function
        self.errors = [y - self.a_s[-1] * activation_grad(self.z_s[-1])]

    def calc_grad(self):
        self.weights_grad = []
        self.biases_grad = []
        error = self.errors[0]
        for i in range(self.depth - 2, -1, -1):
            self.weights_grad.append(self.a_s[i].T.dot(error))
            self.biases_grad.append(np.sum(error, axis=0))
            error = error.dot(self.weights[i].T) * self.activation(self.z_s[i], derivative=True)
        self.weights_grad.reverse()
        self.biases_grad.reverse()
```

In [7]:
```
np.random.seed(0)
nn = NN([6, 2, 2], 0.1, activation=np.tanh)
nn.init_weights()
nn.weights
```

Out[7]:
```
[array([[0.5488135 , 0.71518937],
        [0.60276338, 0.54488318],
        [0.4236548 , 0.64589411],
        [0.43758721, 0.891773  ],
        [0.96366276, 0.38344152],
        [0.79172504, 0.52889492]]),
 array([[0.07103606, 0.0871293 ],
        [0.0202184 , 0.83261985]])]
```

(b) Given the following input values for one pattern: i-1=(-1,1) i=(-1,-1) i+4=(1,-1)

In [8]:
```
x = np.array([-1, 1, 1, -1, 1, -1])
nn.predict(x)
```

Out[8]:
```
array([0.68131947, 0.83883013])
```

(c) The actual observed output for this one pattern is (-1,-1). Define the error and calculate it for all nodes that are not in the input layer.

In [9]:
```
# Gradient of the activation function
def tanh_grad(x):
    return 1 - np.tanh(x) ** 2
```

In [ ]:
```
nn.fit(x, np.array([[-1, -1]]), tanh_grad)
```