In [51]:
```
import numpy as np
import sys
```

In [52]:
```
class Neural_network(object):
    def __init__(self, structure=None, init_weight=None):
        self.af = Activation_fcn()
        if structure:
            self.create_network(structure, init_weight)

    def create_network(self, structure, init_weight):
        self.nnetwork = [structure[0]]

        if init_weight == 'zero':
            for i in range(1, len(structure)):
                new_layer = {
                    # Tu należy wykonać odpowiedniej modyfikacje
                    'weights': np.zeros((structure[i]['units'], structure[i-1]['uni
                    'activation_function': structure[i]['activation_function'],
                    'activation_potential': None,
                    'delta': None,
                    'output': None}
                self.nnetwork.append(new_layer)

        if init_weight == 'one':
            for i in range(1, len(structure)):
                new_layer = {
                    # Tu należy wykonać odpowiedniej modyfikacje
                    'weights': np.ones((structure[i]['units'], structure[i-1]['unit
                    'activation_function': structure[i]['activation_function'],
                    'activation_potential': None,
                    'delta': None,
                    'output': None}
                self.nnetwork.append(new_layer)
        if init_weight == 'rand':
            for i in range(1, len(structure)):
                new_layer = {
                    # Tu należy wykonać odpowiedniej modyfikacje
                    'weights': np.random.randn(structure[i]['units'], structure[i-1
                    'activation_function': structure[i]['activation_function'],
                    'activation_potential': None,
                    'delta': None,
                    'output': None}
                self.nnetwork.append(new_layer)
        return self.nnetwork

    def forward_propagate(self, inputs):
        inp = inputs.flatten()
        for i in range(1, len(self.nnetwork)):
            layer = self.nnetwork[i]

            layer['activation_potential'] = np.matmul(layer['weights'], inp)

            layer['output'] = self.af.output(layer['activation_potential'], layer['
            inp = layer['output']
        return inp

    def predict(self, nnetwork, inputs):
        out = []
        for input in inputs:
            output = self.forward_propagate(input)
```

```
            out.append(output)
        return out
```

In [58]:
```
class Activation_fcn(object):
    def __init__(self):
        self.functions = {
            'linear': self.linear,
            'sigmoid': self.logistic,
            'logistic': self.logistic,
            'tanh': self.tanh,
            'ReLu': self.relu
        }

    def output(self, activation_potential, name):
        if name in self.functions:
            return self.functions[name](activation_potential)
        else:
            sys.exit(f"Error: Activation function '{name}' not found.")

    def linear(self, x):
        return x  # Zwraca bezpośrednio potencjał aktywacji

    def logistic(self, x):
        return 1 / (1 + np.exp(-x))  # Użycie bezpośredniego potencjału

    def tanh(self, x):
        return np.tanh(x)  # Użycie bezpośredniego potencjału

    def relu(self, x):
        return np.maximum(0, x)  # Użycie bezpośredniego potencjału
```

In [64]:
```
if __name__ == "__main__":
    structure = [{'type': 'input', 'units': 1},
                 {'type': 'dense', 'units': 8, 'activation_function': 'tanh'},
                 {'type': 'dense', 'units': 8, 'activation_function': 'tanh'},
                 {'type': 'dense', 'units': 1, 'activation_function': 'linear'}]

    model = Neural_network(structure, 'rand')
    #print(model.nnetwork)

    model_zero = Neural_network(structure, 'zero')
    #print(model_zero.nnetwork)

    model_one = Neural_network(structure, 'one')
    #print(model_one.nnetwork)
```

In [77]:
```
if __name__ == "__main__":
    structure = [{'type': 'input', 'units': 1},
                 {'type': 'dense', 'units': 2, 'activation_function': 'linear'},
                 {'type': 'dense', 'units': 1, 'activation_function': 'linear'}]
    network = model.create_network(structure, "zero")
    n = 1
    X = np.linspace(-5, 5, n).reshape(-1, 1)

    predicted1 = model.predict(network, X)
    print(predicted1)
```

```
[array([0.])]
```

In [82]:
```
if __name__ == "__main__":
    structure = [{'type': 'input', 'units': 1},
                 {'type': 'dense', 'units': 2, 'activation_function': 'tanh'},
                 {'type': 'dense', 'units': 2, 'activation_function': 'tanh'},
```

```
                {'type': 'dense', 'units': 1, 'activation_function': 'tanh'}]
    network = model.create_network(structure, "one")
    n = 1
    X = np.linspace(-5, 5, n).reshape(-1, 1)
    predicted2 = model.predict(network, X)
    print(predicted2)
```

```
[array([-0.95857383])]
```

In [85]:
```
if __name__ == "__main__":
    structure = [{'type': 'input', 'units': 1},
                 {'type': 'dense', 'units': 4, 'activation_function': 'ReLu'},
                 {'type': 'dense', 'units': 4, 'activation_function': 'ReLu'},
                 {'type': 'dense', 'units': 1, 'activation_function': 'ReLu'}]
    network = model.create_network(structure, "rand")
    n = 1
    X = np.linspace(-5, 5, n).reshape(-1, 1)
    predicted3 = model.predict(network, X)
    print(predicted3)
```

```
[array([6.22695315])]
```

In [ ]:
```

```