In [2]:

```
import pandas as pd

import numpy as np
import torch
import torch.nn
```

In [3]:

```
model1 = torch.nn.Sequential( torch.nn.Linear(10, 2, bias=False), torch.
```

In [4]:

```
optimizer = torch.optim.SGD(model1.parameters(), 1e-2)
```

In [5]:

```
def training_loop(n_epochs, optimizer, model, loss_fun, input, output):
        for epoch in range(1, n_epochs + 1):
            p = model(input)
            loss = loss_fun(p, output)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            if epoch == 1 or epoch % 1000 == 0:
                     print('Epoch {}, Loss {}'.format(epoch, float(loss)
```

In [7]:

```
data1 = pd.read_csv("bdata1.csv")
data2 = data1.to_numpy()
data3 = torch.tensor(data2, dtype=torch.float32)
```

In [10]:

```
training_loop(
       n_epochs = 5000,
optimizer = optimizer, model = model1,
loss_fun = torch.nn.MSELoss(), input = data3,
       output = data3)
```

```
Epoch 1, Loss 0.516442596912384
Epoch 1000, Loss 0.11687449365854263
Epoch 2000, Loss 0.09645391255617142
Epoch 3000, Loss 0.09391627460718155
Epoch 4000, Loss 0.09342242777347565
Epoch 5000, Loss 0.09324485063552856
```

In [11]:

```
print()
print(list(model1.parameters()))
```

```
[Parameter containing:
tensor([[-0.2511, -0.2623, -0.0915, -0.2339, -0.2447, -0.0549, -0.2969
, -0.4366,
         -0.2054, -0.1461],
        [ 0.2766,  0.3744,  0.4700,  0.1556,  0.2882, -0.1334, -0.0069
, -0.1935,
         -0.1312, -0.2977]], requires_grad=True), Parameter containing
:
tensor([[-0.3337,  0.3798],
        [-0.2370,  0.4226],
        [-0.1723,  0.5251],
        [-0.1739,  0.5223],
        [-0.4939,  0.3955],
        [-0.6612, -0.3852],
        [-0.5854, -0.5473],
        [-0.6621, -0.3868],
        [-0.5263, -0.2721],
        [-0.4786, -0.4410]], requires_grad=True)]
```

In [12]:

```
data1.head
```

Out[12]:

```
<bound method NDFrame.head of    0  1  2  3  4  5  6  7  8  9
0  0  1  1  1  1  1  0  1  0  0
1  1  0  1  1  1  0  0  0  0  0
2  1  1  1  1  1  0  0  0  0  0
3  1  1  0  0  1  1  0  1  0  0
4  1  1  1  1  1  0  0  0  1  0
5  1  0  0  0  1  1  1  1  1  1
6  0  0  0  0  0  1  1  1  1  1
7  0  0  0  0  0  1  1  1  1  0
8  0  0  0  0  0  1  1  1  0  1
9  0  0  0  0  1  0  1  0  1  1>
```

In [13]:

```
m0 = model1[0]
```

In [14]:

```
m0
```

Out[14]:

```
Linear(in_features=10, out_features=2, bias=False)
```

Which of the two nodes in the code reveals the two types of inputs?

Hidden layer reveals two types of inputs. _2*10 will be your hidden layer._

Can you use the encoder matrix to compute the code for each of the ten rows as input?

yes. we can compute the code for each of the ten rows using encoder matrix.

In [17]:

```
Model_parameter = np.array([( -0.2511, -0.2623, -0.0915, -0.2339, -0.244
         -0.2054, -0.1461),
        (0.2766,  0.3744,  0.4700,  0.1556,  0.2882, -0.1334, -0.0069, -
         -0.1312, -0.2977)])
Model_parameter.shape
```

Out[17]:

```
(2, 10)
```

```
(10*10) = (10*2)*(2*10) Therefor, we can represent ten rows using
encoder matrix.
```