# Machine Learning and Artificial Intelligence

## Assignment 1

ID: 12232418 Name: Jiang Yuchen

According to the question, we are requested to construct the neural network and perform forward pass and backward process. The environment requirements are: `torch==1.13`, `python=3.7`.

### Prepare data

Using random seed 4321 to prepare data.

In [1]:
```
# initialize
import torch

# Make data
torch.manual_seed(4321)

X = torch.rand(size=(8, 2))
y = torch.randint(low=0, high=3, size=(8,))

print(X)
print(y)
```

```
tensor([[0.1255, 0.5377],
        [0.6564, 0.0365],
        [0.5837, 0.7018],
        [0.3068, 0.9500],
        [0.4321, 0.2946],
        [0.6015, 0.1762],
        [0.9945, 0.3177],
        [0.9886, 0.3911]])
tensor([0, 2, 2, 0, 2, 2, 0, 1])
```

### Construct Network

According to the requirement, build two linear layers and two activation functions as the whole network. Using `copy_` function to initialize the weight and bias in two linear layers.

In [2]:
```
# Define network
import torch.nn as nn

class VanillaNet(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(in_features=2, out_features=2, bias=True)
        self.activation1 = nn.Sigmoid()
        self.fc2 = nn.Linear(in_features=2, out_features=3, bias=True)
        self.activation2 = nn.Softmax(dim=-1)

        # init_weight
        with torch.no_grad():
            self.fc1.weight.copy_(torch.tensor(([[0.48, -0.51], [-0.43, -0.48]])))
            self.fc1.bias.copy_(torch.tensor([0.23, 0.05]))
            self.fc2.weight.copy_(torch.tensor(([[-0.99, -0.66], [0.36, 0.34], [-0.75, 0.66]])))
            self.fc2.bias.copy_(torch.tensor([0.32, -0.44, 0.70]))

    def forward(self, x):
        x = self.activation1(self.fc1(x))
        x = self.activation2(self.fc2(x))
        return x
```

## Forward and Backward

Create instance of the model. Taking `X` as input and get the forward result which is the prediction.

Then, using `nn.CrossEntropyLoss()` to calculate categorical cross entropy.

Last, using `loss.backward()` to get the gradient of parameters in the model.

In [3]:
```
# Main and prediction
model = VanillaNet()
prediction = model(X)
prediction
```

Out[3]:
```
tensor([[0.1867, 0.2663, 0.5470],
        [0.1747, 0.2958, 0.5295],
        [0.1959, 0.2738, 0.5303],
        [0.2022, 0.2590, 0.5388],
        [0.1812, 0.2820, 0.5368],
        [0.1787, 0.2902, 0.5311],
        [0.1863, 0.2966, 0.5171],
        [0.1886, 0.2943, 0.5171]], grad_fn=<SoftmaxBackward0>)
```

In [4]:
```
# Loss
loss_fn = nn.CrossEntropyLoss()
loss = loss_fn(prediction, y)
loss
```

Out[4]:
```
tensor(1.0681, grad_fn=<NllLossBackward0>)
```

In [6]:
```
# gradient of loss
loss.backward()
for name, param in model.named_parameters():
    print(name, "'s gradient:",  param.grad)
```

```
fc1.weight 's gradient: tensor([[ 0.0057,  0.0067],
        [-0.0017,  0.0058]])
fc1.bias 's gradient: tensor([0.0167, 0.0001])
fc2.weight 's gradient: tensor([[-0.0059, -0.0053],
        [ 0.0323,  0.0252],
        [-0.0264, -0.0199]])
fc2.bias 's gradient: tensor([-0.0157,  0.0579, -0.0422])
```