# *Assignment 2*

### 1. Input the training data

```
import torch
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])
```

### 2. create a network model

```
import torch.nn as nn
class Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.layer1 = nn.Linear(2, 2)
        self.layer1.weight.data = torch.tensor([[0.48, -0.51], [-0.43, -0.48]])
        self.layer1.bias.data = torch.tensor([0.23, 0.05])
        self.activation1 = nn.Sigmoid()
        self.layer2 = nn.Linear(2, 3)
        self.layer2.weight.data = torch.tensor([[-0.99, -0.66], [0.36, 0.34], [-0.75,
0.66]])
        self.layer2.bias.data = torch.tensor([0.32, -0.44, 0.7])
        self.activation2 = nn.Softmax(dim=1)

    def forward(self, X):
        X = self.activation1(self.layer1(X))
        X = self.activation2(self.layer2(X))
        return X
```

### 3.the predictions workflow

```
device = torch.device("cpu")
model = Model()
model.to(device)
prediction_result = model.forward(X)
print(f"prediction : {prediction_result}")
```

```
prediction : 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>)
```

### 4.the loss using categorical cross entropy

```
loss_fn = nn.CrossEntropyLoss()
loss = loss_fn(prediction_result, y)
print(loss)
```

```
tensor(1.0681, grad_fn=<NllLossBackward0>)
```

### 5.the gradient of the loss with respect to the weights and biases

```
loss.backward()
print(f"layer1.weight.grad : {model.layer1.weight.grad}")
print(f"layer1.bias.grad : {model.layer1.bias.grad}")
print(f"layer2.weight.grad : {model.layer2.weight.grad}")
print(f"layer2.bias.grad : {model.layer2.bias.grad}")
```

```
layer1.weight.grad : tensor([[ 0.0057,  0.0067],
        [-0.0017,  0.0058]])
layer1.bias.grad : tensor([0.0167, 0.0001])
layer2.weight.grad : tensor([[-0.0059, -0.0053],
        [ 0.0323,  0.0252],
        [-0.0264, -0.0199]])
layer2.bias.grad : tensor([-0.0157,  0.0579, -0.0422])
```

# *Assignment 3*

## *Q1.*

For a tensor X of arbitrary shape, does len(X) always correspond to the length of a certain axis of X? What is that axis?

**Answer:** len(X) is always the length of the first axis of X.

```
import torch
import random
dim1 = random.randint(1, 10)
dim2 = random.randint(1, 10)
dim3 = random.randint(1, 10)
tensor = torch.rand(dim1 , dim2, dim3)
print(f"dim1: {dim1}, dim2: {dim2}, dim3: {dim3}, len(tensor): {len(tensor)}")
```

```
torch.Size([10, 3, 4])
dim1: 10, dim2: 3, dim3: 4, len(tensor): 10
```

## *Q2.*

Run A / A.sum(axis=1) and see what happens. Can you analyze the reason?

**Answer:** If the length of second axis is not equal to the first axis. There will be an error.

```
A = torch.tensor([[1, 2, 3], [4, 5, 6]])
print(A / A.sum(axis=1))
```

```
---------------------------------------------------------------------------

RuntimeError                              Traceback (most recent call last)

<ipython-input-16-ffde4c540022> in <module>
      1 A = torch.tensor([[1, 2, 3], [4, 5, 6]])
----> 2 print(A / A.sum(axis=1))
```

```
RuntimeError: The size of tensor a (3) must match the size of tensor b (2) at non-
singleton dimension 1
```

```
A = torch.tensor([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(f"A.sum(axis=1) is {A.sum(axis=1)}")
print(A / A.sum(axis=1))
```

```
A.sum(axis=1) is tensor([ 6, 15, 24])
tensor([[0.1667, 0.1333, 0.1250],
        [0.6667, 0.3333, 0.2500],
        [1.1667, 0.5333, 0.3750]])
```

However, if the length of second axis is equal to the first axis, the result will be a matrix with the same shape as A. The value of each element is the corresponding element of A divided by the sum of the corresponding row.

## *Q3.*

When traveling between two points in Manhattan, what is the distance that you need to cover in terms of the coordinates, i.e., in terms of avenues and streets? Can you travel diagonally?

**Answer:** Firstly, you can't travel diagonally. Because there are buildings on the diagonal.

You can't cross them directly. This problem is classic. The distance between two points on the Manhattan grid is the sum of the absolute values of the differences of the coordinates.

For example, the distance between (x1, y1) and (x2, y2) is |x1-x2| + |y1-y2|. This distance is called the Manhattan distance.

## *Q4.*

Find the gradient of the function $f(x) = 3x_1^2 + 5e^{x_2}$

**Answer:** The gradient vector of $f(x)$ is $\nabla f(x) = \begin{bmatrix} 6x_1 \\ 5e^{x_2} \end{bmatrix}$

## *Q5.*

Can you write out the chain rule for the case where u = f(x, y, z) and x = x(a, b), y = y(a, b), and z = z(a, b)?

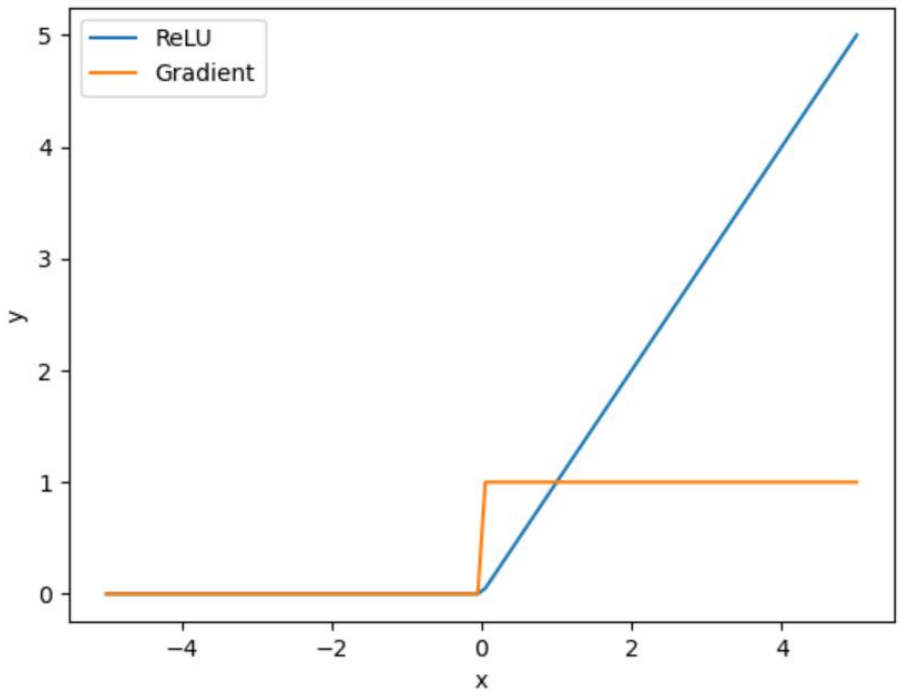
**Answer:**

$$\frac{\partial u}{\partial a} = \frac{\partial u}{\partial x}\frac{\partial x}{\partial a} + \frac{\partial u}{\partial y}\frac{\partial y}{\partial a} + \frac{\partial u}{\partial z}\frac{\partial z}{\partial a}$$

$$\frac{\partial u}{\partial b} = \frac{\partial u}{\partial x}\frac{\partial x}{\partial b} + \frac{\partial u}{\partial y}\frac{\partial y}{\partial b} + \frac{\partial u}{\partial z}\frac{\partial z}{\partial b}$$

这里请注意求偏导的符号！

## *Q6.*




这里的完美答案为设计一个带控制语句的函数，并且可视化它的梯度（尤其是间断点）。

## *Q7.*

A coin with 40% chance for heads is tossed 10 times. Let X be the variable representing the number of heads.

The probabilities for X, P(X), are shown in the table below. What is the value of the variance?

**Answer:**

The mean distribution of X is $X = 4$.

$Var = \sum_{i=1} (x_i - X)^2 * P(x_i) = 2.3997$

```
p = [0.006,0.0403,0.1209,0.215,0.2508,0.2007,0.1115,0.0425,0.0106,0.0016,0.0001]
x = [i for i in range(11)]
mean = 4
var = 0
for i in range(11):
    var += p[i] * (x[i] - mean)**2
print(f"variance is {var}")
```

```
variance is 2.3997
```

# *Assignment 4*

满分答案包括模型定义、训练过程、测试结果、文字说明四部分。

### 1.1 Write a program of a perceptron to learn an OR function using pytorch

[1]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
```

```
/home/zhaoxy/anaconda3/envs/comhedges/lib/python3.10/site-
packages/tqdm/auto.py:22: TqdmWarning: IProgress not found. Please update
jupyter and ipywidgets. See
https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

### 1.2 Define the input and output

[2]:
```
x = torch.tensor([[0,0],[0,1],[1,0],[1,1]], dtype=torch.float)
y = torch.tensor([[0],[1],[1],[1]], dtype=torch.float)
```

### 1.3 Define the model

[3]:
```
class Or_Net(nn.Module):
    def __init__(self):
        super(Or_Net, self).__init__()
        self.fc1 = nn.Linear(2, 1)
    def forward(self, x):
        x = self.fc1(x)
        x = torch.sigmoid(x)
        return x
```

### 1.4 Initialize the perceptron model

[4]:
```
net = Or_Net()
```

### 1.5 Define the loss function and optimizer

[5]:
```
criterion = torch.nn.MSELoss()
optimizer = torch.optim.Adam(net.parameters(), lr=0.1)
```

### 1.6 Train the model

[6]:
```
for epoch in range(20000):
    optimizer.zero_grad()
    output = net(x)
    loss = criterion(output, y)
    loss.backward()
    optimizer.step()
    if epoch % 1000 == 0:
        print('epoch: {}, loss: {}'.format(epoch, loss.item()))
```

```
epoch: 0, loss: 0.3693433105945587
epoch: 1000, loss: 0.00031523327925242484
epoch: 2000, loss: 9.040788427228108e-05
epoch: 3000, loss: 3.947518052882515e-05
epoch: 4000, loss: 2.0148338080616668e-05
epoch: 5000, loss: 1.1084875040978659e-05
epoch: 6000, loss: 6.348990609694738e-06
epoch: 7000, loss: 3.7203694773779716e-06
epoch: 8000, loss: 2.2094313862908166e-06
epoch: 9000, loss: 1.3226831470092293e-06
epoch: 10000, loss: 7.956527952046599e-07
epoch: 11000, loss: 4.800610895472346e-07
epoch: 12000, loss: 2.902128244386404e-07
epoch: 13000, loss: 1.7561167453550297e-07
epoch: 14000, loss: 1.0637280922765058e-07
epoch: 15000, loss: 6.445011990763305e-08
epoch: 16000, loss: 3.910562540454521e-08
epoch: 17000, loss: 2.3740749810485795e-08
epoch: 18000, loss: 1.4405565629260764e-08
epoch: 19000, loss: 8.751704250187231e-09
```

### 1.7 Test the model

[7]:
```
print('test the model')
print('input: {}, output: {}'.format(x[0], net(x[0]).data))
print('input: {}, output: {}'.format(x[1], net(x[1]).data))
print('input: {}, output: {}'.format(x[2], net(x[2]).data))
print('input: {}, output: {}'.format(x[3], net(x[3]).data))
print('the weight of the model: {}'.format(net.fc1.weight.data))
print('the bias of the model: {}'.format(net.fc1.bias.data))
```

```
test the model


input: tensor([0., 0.]), output: tensor([0.0001])
input: tensor([0., 1.]), output: tensor([0.9999])
input: tensor([1., 0.]), output: tensor([0.9999])
input: tensor([1., 1.]), output: tensor([1.])
the weight of the model: tensor([[18.7141, 18.7076]])
the bias of the model: tensor([-9.1212])
```

## *Assignment 5*

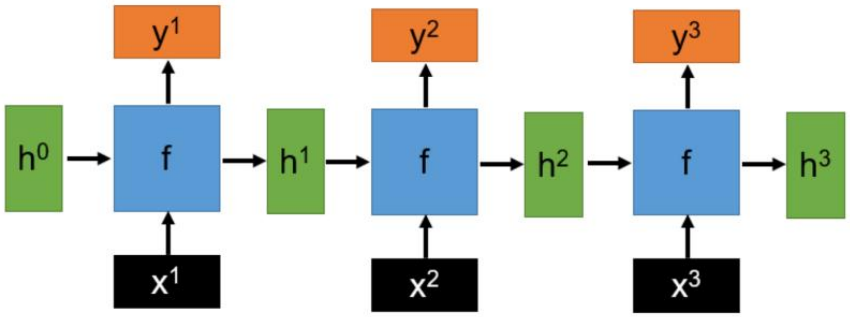
**满分答案报告规范、代码清晰完备、包含训练和测试结果、必要的注释或文字说明。上传李雪琪同学的作业以供参考。**

## *Assignment 6*

### *Q1.*

1、题目要求提供论据来证明一个朴素的 RNN 在做反向传播时有一个梯度消失的问题。

首先我们要知道朴素 RNN 由于其结构的原因在处理长序列时，梯度可能会在进行反向传播的过程中变得很小，这就是所谓的朴素 RNN 所存在的梯度消失问题。因为 RNN 在时间步骤之间共享权重，每个序列共用一个 f，因此在计算梯度时，会连续相乘多个相同的矩阵。如果这些矩阵的特征值小于 1，梯度会在每个时间步骤减小，最终在很多时间步骤后变得非常小。隐藏状态（h）是通过输入（x）和前一个时间步的隐藏状态（h）计算得出的。在反向传播期间，我们需要计算每个时间步的梯度以更新模型参数。然而，由于 RNN 的结构，每个时间步的梯度都与上一个时间步的梯度相关。因此，当我们计算梯度时，梯度值会通过时间步不断传递。如果在这个传递过程中梯度值不断缩小，最终可能会变得非常小，甚至接近于零。




对于如图所示三个时间RNN单元，其中$y_i = w_0 h_i + b_2$，$h_i = w_x X_i + w_s h_{i-1} + b_1$

假设其损失函数为 L，我们反向传播实际上就是使得 L 为损失函数对$w_0$，$w_x$，$w_s$，$b_1$，$b_2$求偏导，这里以对权重求偏导为例子。

当 t = 3 时刻，L 对 Wo 求偏导为：

$$\frac{\partial L3}{\partial w_0} = \frac{\partial L3}{\partial y3}\frac{\partial y3}{\partial w_0}$$

对 Wx 求偏导可得：

$$\frac{\partial L3}{\partial w_x} = \frac{\partial L3}{\partial y3}\frac{\partial y3}{\partial h3}\frac{\partial h3}{\partial w_x} + \frac{\partial L3}{\partial y_3}\frac{\partial y3}{\partial h3}\frac{\partial h3}{\partial h2}\frac{\partial h2}{\partial w_x} + \frac{\partial L3}{\partial y3}\frac{\partial y3}{\partial h3}\frac{\partial h3}{\partial h2}\frac{\partial h2}{\partial h1}\frac{\partial h1}{\partial w_x}$$

对 Ws 求偏导同理，这里不赘述，可以看出对于 Wo 的偏导不会受到层数增加的影响，对于 Wx 和 Ws 可能会产生梯度消失。

扩展到任意时刻：

$$\frac{\partial L_t}{\partial w_x} = \sum_{k=0}^{t}\frac{\partial L_t}{\partial y_t}\frac{\partial y_t}{\partial h_t}\left(\prod_{j=k+1}^{t}\frac{\partial h_j}{\partial h_{j-1}}\right)\frac{\partial h_k}{\partial w_x}$$

对 Ws 求偏导同理，这里不赘述，随着层数的增加，$\prod_{j=k+1}^{t}\frac{\partial h_j}{\partial h_{j-1}}$累乘的单元数增加，而由于激活函数 tanh 的偏导数小于 1，所以随着 t 的增加这部分连乘会越来越小，导致上述公式的值越来越趋近于 0， 这就导致了梯度消失问题。

通过编写代码可以证明这一点，代码如下：

```
import torch.nn as nn
import torch

# 设置随机种子以确保可重复性
torch.manual_seed(50)

# 定义一个朴素 RNN 类
class NaiveRNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(NaiveRNN, self).__init__()
        self.hidden_size = hidden_size
        self.i2h = nn.Linear(input_size + hidden_size, hidden_size)
        self.i2o = nn.Linear(input_size + hidden_size, output_size)
        nn.init.uniform_(self.i2h.weight, a=0, b=1)
        nn.init.uniform_(self.i2o.weight, a=0, b=1)

    def forward(self, input, hidden):
        combined = torch.cat((input, hidden), 1)
        hidden = torch.tanh(self.i2h(combined))
        output = self.i2o(combined)
        return output, hidden
```

```
    def init_hidden(self):
        return torch.zeros(1, self.hidden_size)

# 设置超参数
input_size = 5
hidden_size = 10
output_size = 1
sequence_length = 20

# 创建模型
model = NaiveRNN(input_size, hidden_size, output_size)

# 生成随机输入数据和目标数据
inputs = [torch.randn(1, input_size) for _ in range(sequence_length)]
targets = torch.zeros(sequence_length, 1)

# 初始化隐藏状态
hidden = model.init_hidden()

# 通过 RNN 传递输入序列
outputs = []
for t in range(sequence_length):
    output, hidden = model(inputs[t], hidden)
    outputs.append(output)

# 计算损失
loss_fn = nn.MSELoss()
loss = loss_fn(torch.stack(outputs).squeeze(), targets)
```

```python
# 反向传播
loss.backward()


print("sequence_length = ", sequence_length)
# 检查梯度消失问题
for name, param in model.named_parameters():
    print(f"{name}.grad: {param.grad.norm().item()}")
```

当序列长度为 5 时：

```
sequence_length =  5
i2h.weight.grad: 17.681381225585938
i2h.bias.grad: 10.598565101623535
i2o.weight.grad: 20.075037002563477
i2o.bias.grad: 6.8015522956848145
```

隐藏状态权重偏导的 L2 范数大约 17.7 还是大的

```
sequence_length =  10
i2h.weight.grad: 8.89891529083252
i2h.bias.grad: 5.336695194244385
i2o.weight.grad: 31.642980575561523
i2o.bias.grad: 9.888933181762695
```

隐藏状态权重偏导的 L2 范数变小到 8.89

```
sequence_length =  20
i2h.weight.grad: 4.735414505004883
i2h.bias.grad: 3.0114617347717285
i2o.weight.grad: 30.16775131225586
i2o.bias.grad: 9.771400451660156
```

20 时进一步变小

```
sequence_length =  200
i2h.weight.grad: 0.7760611176490784
i2h.bias.grad: 0.3978833556175232
i2o.weight.grad: 38.87307357788086
i2o.bias.grad: 12.273470878601074
```

200 时接近于 0.77，可以看出随着序列增加，对于 Wx 和 Ws 的梯度不断变小，慢慢消失，而对于 Wo 的梯度没什么影响。

## *Q2.*

Let $h_d$ and $h_t$ denote the hidden states of depth dimension and time dimension, respectively.

Let $C_d$ and $C_t$ denote the cell states of depth dimension and time dimension, respectively.

Let $x_d$ and $x_t$ denote the inputs along two dimensions.

For simplicity, we denote $h_t$ by $[h_t, x_t]$ and $h_d$ by $[h_d, x_d]$. Note that, $[]$ denote the concat operation.

The new cell state and hidden state can be calculated as follows:

**For dimension $t$**:

$$f_t = \sigma(W_{f,t}[h_{t-1}, h_{d-1}] + b_{f,t})$$

$$i_t = \sigma(W_{i,t}[h_{t-1}, h_{d-1}] + b_{i,t})$$

$$o_t = \sigma(W_{o,t}[h_{t-1}, h_{d-1}] + b_{o,t})$$

$$\hat{C}_t = \tanh(W_{C,t}[h_{t-1}, h_{d-1}] + b_{C,t})$$

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \hat{C}_t$$

$$h_t = o_t \cdot \tanh C_t$$

**For dimension $d$**:

$$f_d = \sigma(W_{f,d}[h_{t-1}, h_{d-1}] + b_{f,d})$$

$$i_d = \sigma(W_{i,d}[h_{t-1}, h_{d-1}] + b_{i,d})$$

$$o_d = \sigma(W_{o,d}[h_{t-1}, h_{d-1}] + b_{o,d})$$

$$\hat{C}_d = \tanh(W_{C,d}[h_{t-1}, h_{d-1}] + b_{C,d})$$

$$C_d = f_d \cdot C_{d-1} + i_d \cdot \hat{C}_d$$

$$h_d = o_d \cdot \tanh C_d$$

表达形式不同，逻辑正确即可。