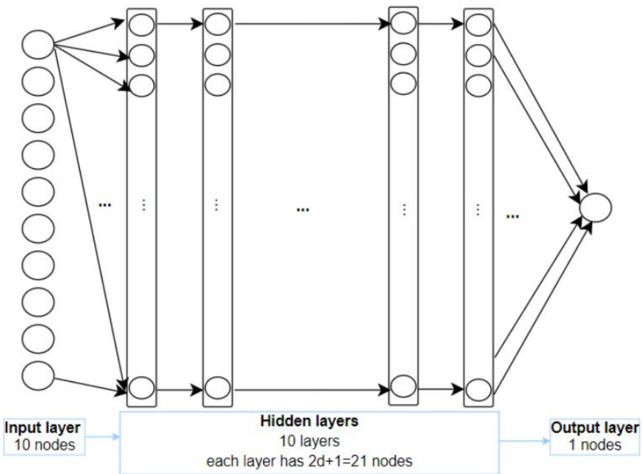
BS 6207 ASSIGNMENT 1 - *Tian Siqi, G2101015G*

Github link(codes,output files and a more detailed report):
https://github.com/SiqiT/Assigmnment1

Question 1

Here we set *k=10, d=10*(10 hidden layers, each layer has 21 nodes)



Input layer: X.shape(1,10)-->(batch_size=1,10 nodes)
Output layer: y.shape(1,1)-->(batch_szie=1,1 node)

```
X = Variable(torch.rand(batch_size, d), requires_grad=True)
y = torch.tensor([[torch.sum(X**2)/d]])
```

Activation function *ReLu* and its derivative:

```
def ReLu_d(x):
    return np.where(x < 0, 0, 1)
def ReLu(x):
    return np.where(x < 0, 0, x)
```

Loss function:
```
loss = (y_pred - y)**2
```

(1) Forward Propagation

- Autograd

I use the *nn_module* class to define the network structure and achieve the forward propagation for one time. *'nn.Linear'* is used for full_connected layer, *'nn.ReLU'* is used for the activation function.

- My_grad

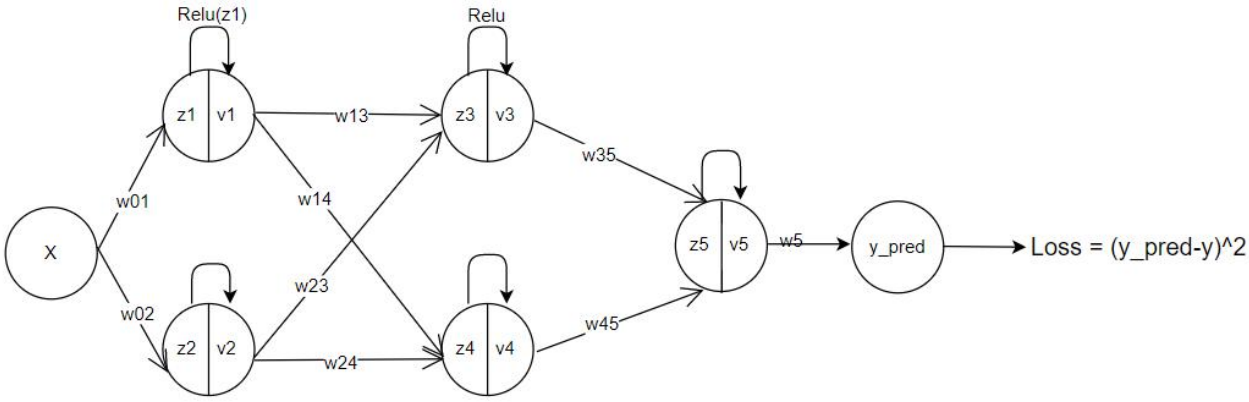
Based on the same original weight and bias in the autograd, I use *np.dot* to achieve forward propagation in the function feedforward and save the node value per layer in a tensor variable.

(2) Backward Propagation

- Autograd

loss.backward() can computes dloss/dx for every parameter, then use *param.grad* to get grad for each variable.

- My_grad



Activation function:

$$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases} \qquad f'(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$$

Loss:

$$\frac{\mathrm{dL}}{\mathrm{dy\_pred}} = 2*(\mathrm{y\_pred} - \mathrm{y})$$

Output layer:

$$\mathrm{y\_pred} = \mathrm{v5} * \mathrm{w5} + \mathrm{b5} \qquad \mathrm{v5} = \delta(z_5)$$

$$\frac{\partial L}{\partial \mathrm{w}_5} = \frac{\partial L}{\partial \mathrm{y\_pred}} \frac{\partial \mathrm{y\_pred}}{\partial \mathrm{w}_5} = dL * \delta'(\mathrm{z}_5) * v_5 = 2(y\_pred - y) * \delta'(\mathrm{z}_5) * v_5$$

$$\frac{\partial L}{\partial \mathrm{b}_5} = \frac{\partial L}{\partial \mathrm{y\_pred}} \frac{\partial \mathrm{y\_pred}}{\partial \mathrm{b}_5} = dL * \delta'(\mathrm{z}_5) = 2(y\_pred - y) * \delta'(\mathrm{z}_5)$$

The same, Hidden layer:

$$\frac{\partial L}{\partial \mathrm{w}_4} = \frac{\partial L}{\partial \mathrm{z}_5} \frac{\partial \mathrm{z}_5}{\partial \mathrm{w}_4} = \frac{\partial L}{\partial \mathrm{z}_5} * \delta'(\mathrm{z}_4) * w_{45} * v_4$$

$$\frac{\partial L}{\partial \mathrm{b}_4} = \frac{\partial L}{\partial \mathrm{z}_5} \frac{\partial \mathrm{z}_5}{\partial b_5} = \frac{\partial L}{\partial \mathrm{z}_5} * \delta'(\mathrm{z}_4) * w_{45}$$

Input layer: change v to x;

Finally, Compare the two files torch_autograd.dat and my_autograd.dat and show that they give the same values for up to 5 significant numbers:

```
diff_w=0
for i in range(len(d_w)):
    diff_w = diff_w+ np.absolute(sum(sum(d_w[i]-auto_w[i])))
```

```
diff_w
```

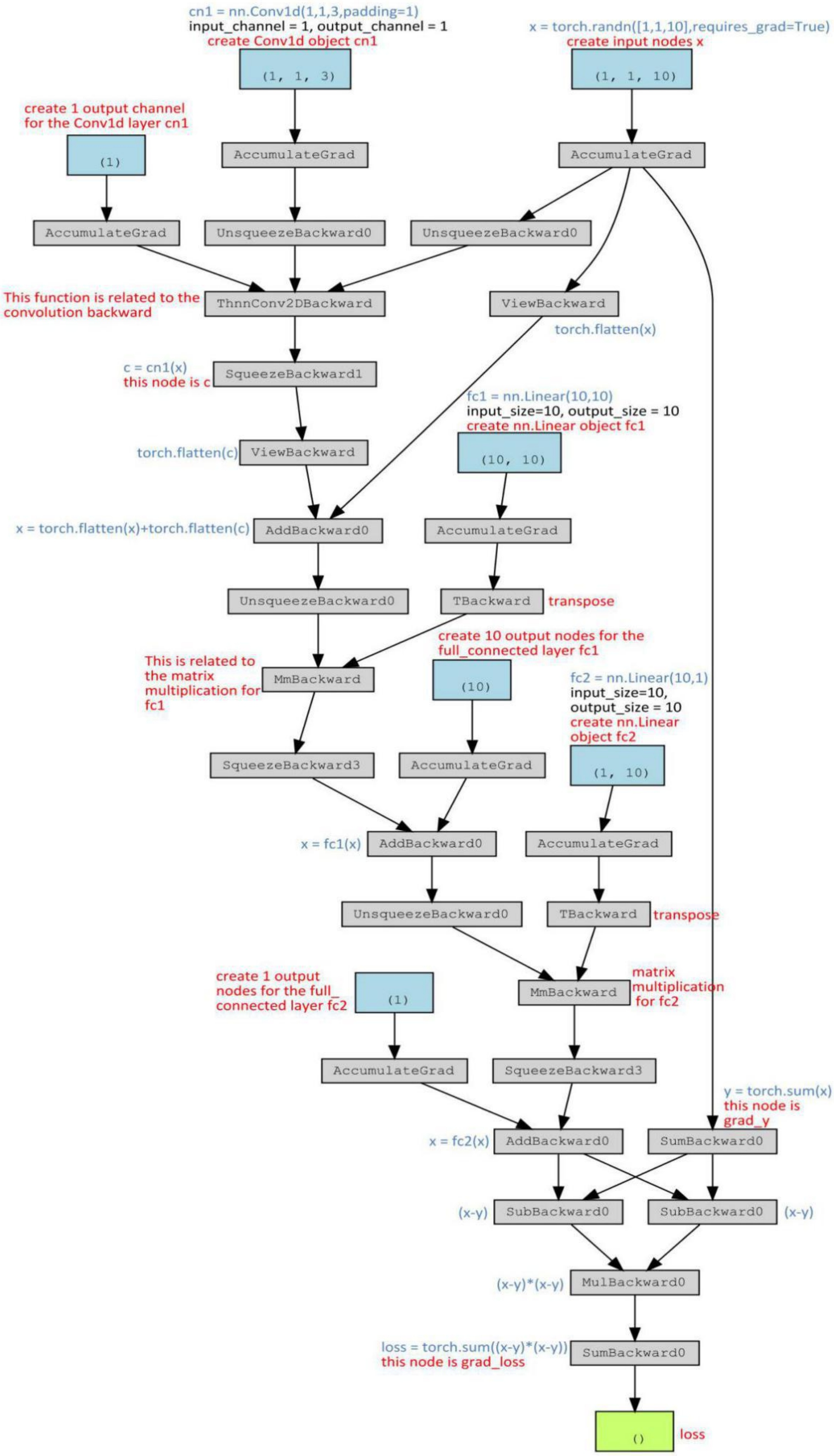
```
0.0
```

```
diff_b=0
for i in range(len(d_z)):
    diff_b = diff_b+ np.absolute(sum(sum(d_z[i]-auto_b[i])))
diff_b
```

```
0.0
```

The gradient values from those 2 ways are the same.

Question 2



cn1 = nn.Conv1d(1,1,3,padding=1)
input_channel = 1, output_channel = 1
create Conv1d object cn1
(1, 1, 3)
x = torch.randn([1,1,10],requires_grad=True)
create input nodes x
(1, 1, 10)
create 1 output channel for the Conv1d layer cn1
(1)
AccumulateGrad
AccumulateGrad
AccumulateGrad
UnsqueezeBackward0
UnsqueezeBackward0
This function is related to the convolution backward
ThnnConv2DBackward
ViewBackward
torch.flatten(x)
c = cn1(x)
this node is c
SqueezeBackward1
fc1 = nn.Linear(10,10)
input_size=10, output_size = 10
create nn.Linear object fc1
torch.flatten(c)
ViewBackward
(10, 10)
x = torch.flatten(x)+torch.flatten(c)
AddBackward0
AccumulateGrad
UnsqueezeBackward0
TBackward
transpose
create 10 output nodes for the full_connected layer fc1
This is related to the matrix multiplication for fc1
MmBackward
(10)
fc2 = nn.Linear(10,1)
input_size=10,
output_size = 10
create nn.Linear object fc2
SqueezeBackward3
AccumulateGrad
(1, 10)
x = fc1(x)
AddBackward0
AccumulateGrad
UnsqueezeBackward0
TBackward
transpose
create 1 output nodes for the full_connected layer fc2
(1)
MmBackward
matrix multiplication for fc2
AccumulateGrad
SqueezeBackward3
y = torch.sum(x)
this node is grad_y
x = fc2(x)
AddBackward0
SumBackward0
(x-y)
SubBackward0
SubBackward0
(x-y)
(x-y)*(x-y)
MulBackward0
loss = torch.sum((x-y)*(x-y))
this node is grad_loss
SumBackward0
()
loss

This is the computational graph for the above scripts, and there are explanations for nodes in the graph above.
For different types of tensor variables, the autograd backward functions for calculating gradients would be different:
torch.sum() —> grad_fn=<*SumBackward*>
nn.Conv1d(x) —> grad_fn=<*SqueezeBackward*>
torch.flatten() —> grad_fn=<*ViewBackward*>
a+b —> grad_fn=<*AddBackward*>
a-b —> grad_fn=<*SubBackward*>
a*b —> grad_fn=<*MulBackward*>
nn.Linear(x) —> grad_fn=<*AddBackward*>

For other nodes that are not marked in the graph:
‘*AccumulateGrad*’ represents leaf nodes, also as known as the end point of the computational graph for BP. It accumulates all backward gradient information for the leaf nodes.

‘*UnsqueezeBackward*’,‘*SqueezeBackward*’ are working with the tensor variable’s dimension and shape, to add a dimension or remove a dimension for a variable.