Deep Learning with PyTorch

이동민

삼성전자 서울대 공동연구소

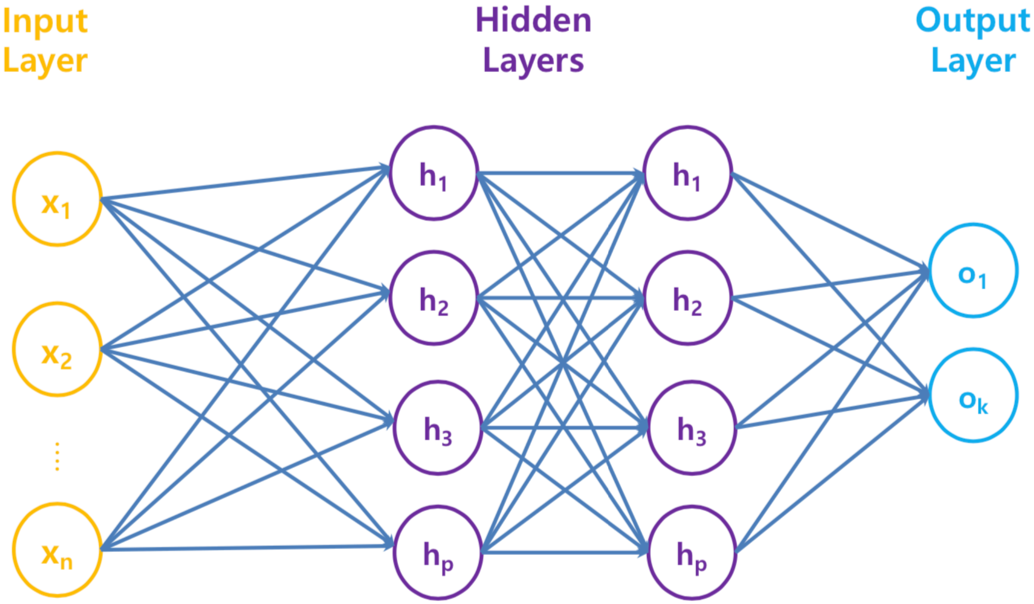
Jul 15, 2019

Outline

- Deep Learning with PyTorch
  - Tensor Manipulation
  - Deep Neural Network
  - Make Model with PyTorch

PyTorch

- Deep Neural Network도 numpy로 표현할 수 있을까?
  - Deep Neural Network (DNN)



- 신경망이 깊어질수록 numpy로 표현하는 것이 점점 더 어려워짐

  → 따라서 딥러닝 프레임워크를 통해 손쉽게 표현하고자 함

PyTorch

Facebook AI Research

Caffe2

PYTORCH

Yann LeCun

DIRECTOR OF AI RESEARCH

Facebook AI Research (FAIR)

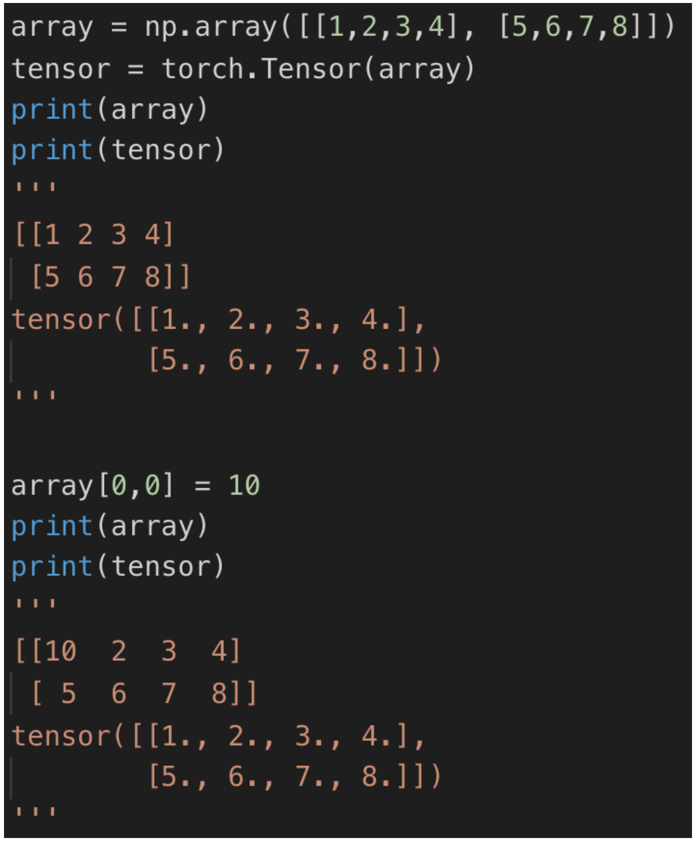
Soumith Chintala

RESEARCH ENGINEER

Facebook AI Research (FAIR)

Tensor Manipulation

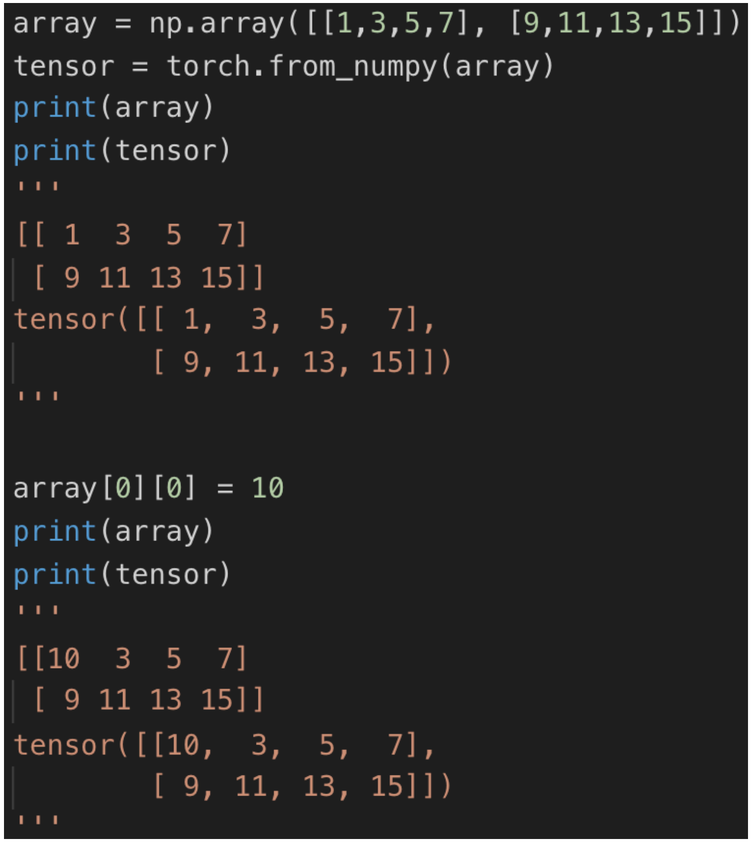
- Numpy array → Torch tensor
  - torch.Tensor - Constructs a tensor with data



```
array = np.array([[1,2,3,4], [5,6,7,8]])
tensor = torch.Tensor(array)
print(array)
print(tensor)
'''
[[1 2 3 4]
 [5 6 7 8]]
tensor([[1., 2., 3., 4.],
        [5., 6., 7., 8.]])
'''

array[0,0] = 10
print(array)
print(tensor)
'''
[[10  2  3  4]
 [ 5  6  7  8]]
tensor([[1., 2., 3., 4.],
        [5., 6., 7., 8.]])
'''
```

Tensor Manipulation

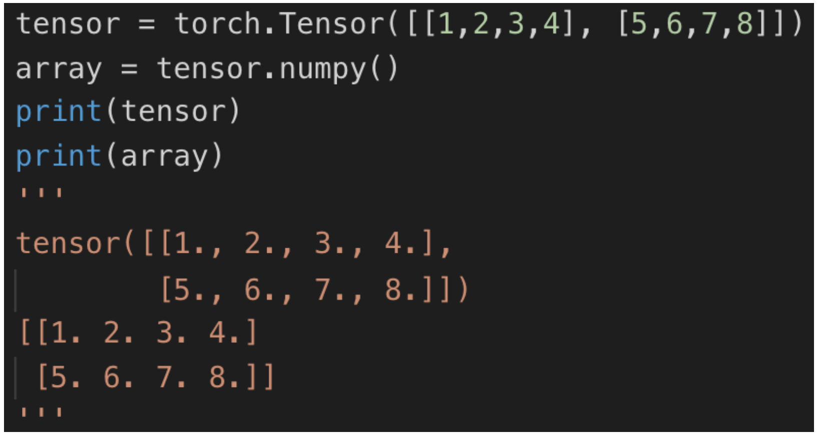
- Numpy array → Torch tensor
  - torch.from_numpy - Creates a Tensor from a numpy.ndarray



```
array = np.array([[1,3,5,7], [9,11,13,15]])
tensor = torch.from_numpy(array)
print(array)
print(tensor)
'''
[[ 1  3  5  7]
 [ 9 11 13 15]]
tensor([[ 1,  3,  5,  7],
        [ 9, 11, 13, 15]])
'''


array[0][0] = 10
print(array)
print(tensor)
'''
[[10  3  5  7]
 [ 9 11 13 15]]
tensor([[10,  3,  5,  7],
        [ 9, 11, 13, 15]])
'''
```

Tensor Manipulation

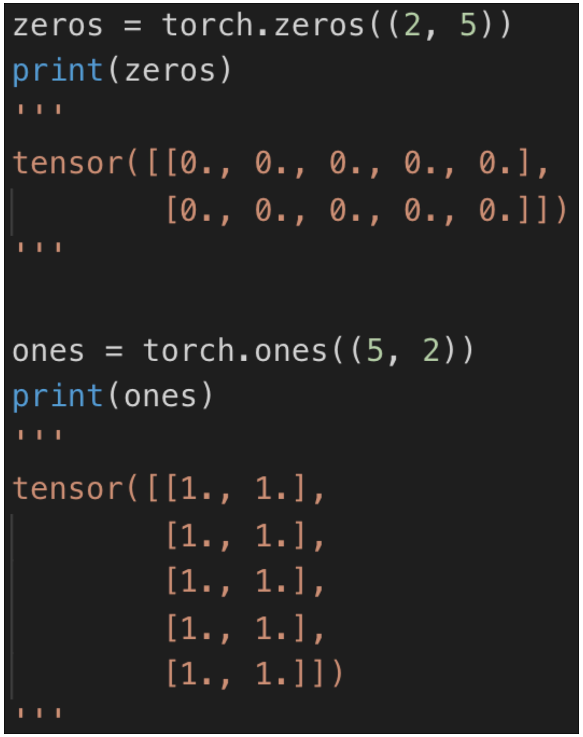
- Torch tensor → Numpy array
    - numpy() - Returns self tensor as a numpy ndarray



```
tensor = torch.Tensor([[1,2,3,4], [5,6,7,8]])
array = tensor.numpy()
print(tensor)
print(array)
'''
tensor([[1., 2., 3., 4.],
        [5., 6., 7., 8.]])
[[1. 2. 3. 4.]
 [5. 6. 7. 8.]]
'''
```

Tensor Manipulation

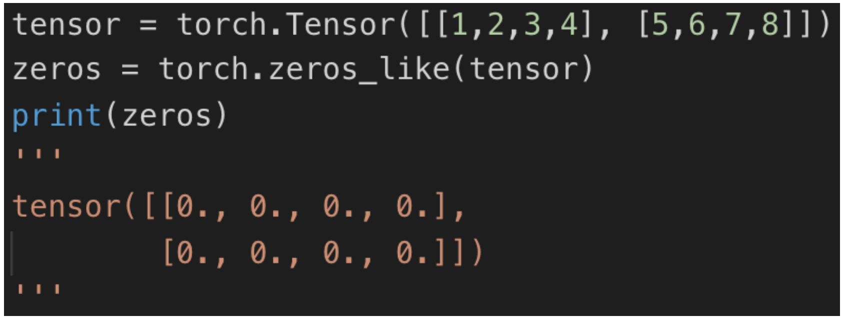
- Creating functions
  - Zeros & Ones



```
zeros = torch.zeros((2, 5))
print(zeros)
'''
tensor([[0., 0., 0., 0., 0.],
        [0., 0., 0., 0., 0.]])
'''


ones = torch.ones((5, 2))
print(ones)
'''
tensor([[1., 1.],
        [1., 1.],
        [1., 1.],
        [1., 1.],
        [1., 1.]])
'''
```

Tensor Manipulation

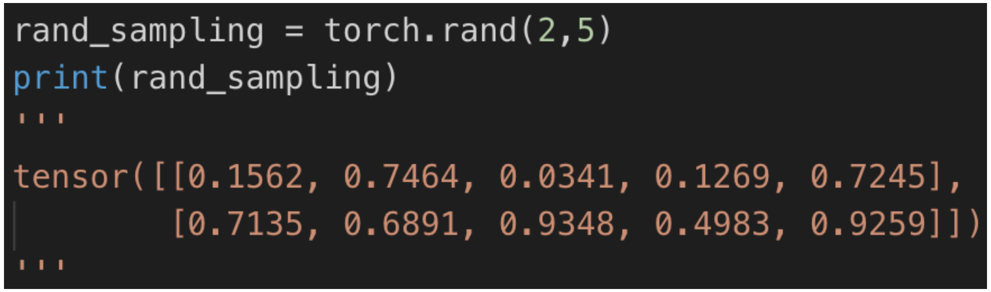
- Creating functions
    - Something like



```
tensor = torch.Tensor([[1,2,3,4], [5,6,7,8]])
zeros = torch.zeros_like(tensor)
print(zeros)
'''
tensor([[0., 0., 0., 0.],
        [0., 0., 0., 0.]])
'''
```

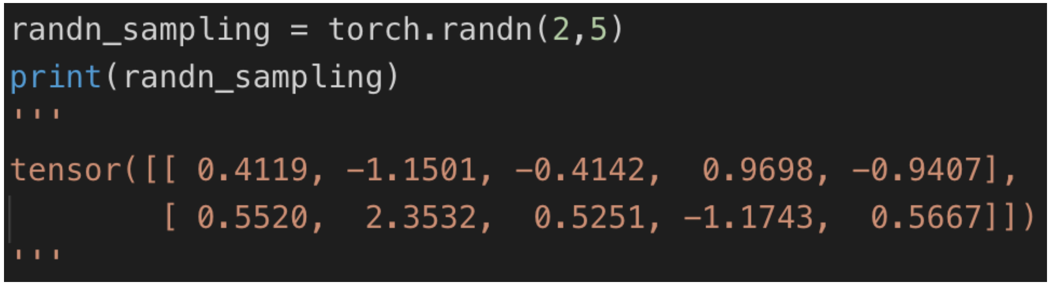
Tensor Manipulation

- Creating functions
    - Rand - Uniform distribution over $[0, 1)$



```python
rand_sampling = torch.rand(2,5)
print(rand_sampling)
'''
tensor([[0.1562, 0.7464, 0.0341, 0.1269, 0.7245],
        [0.7135, 0.6891, 0.9348, 0.4983, 0.9259]])
'''
```

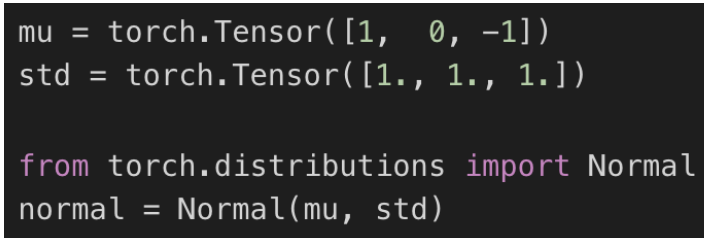
    - Randn - Standard normal(gaussian) distribution of mean 0 and variance 1



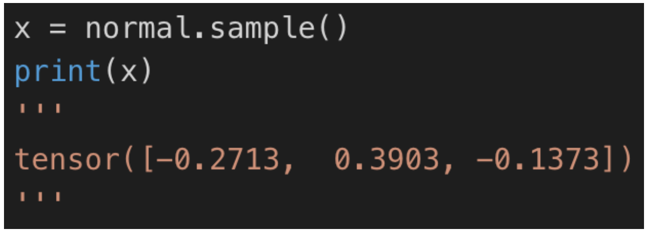
```python
randn_sampling = torch.randn(2,5)
print(randn_sampling)
'''
tensor([[ 0.4119, -1.1501, -0.4142,  0.9698, -0.9407],
        [ 0.5520,  2.3532,  0.5251, -1.1743,  0.5667]])
'''
```

Tensor Manipulation

- Creating functions
  - Normal(Gaussian) distribution



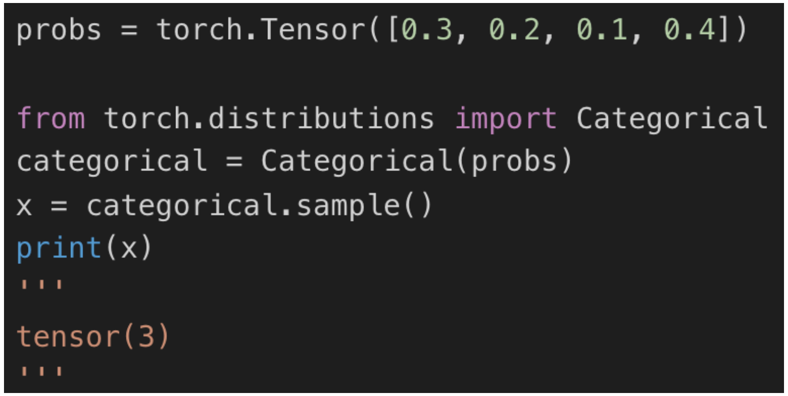
```
mu = torch.Tensor([1,  0, -1])
std = torch.Tensor([1., 1., 1.])

from torch.distributions import Normal
normal = Normal(mu, std)
```



```
x = normal.sample()
print(x)
'''
tensor([-0.2713,  0.3903, -0.1373])
'''
```

Tensor Manipulation

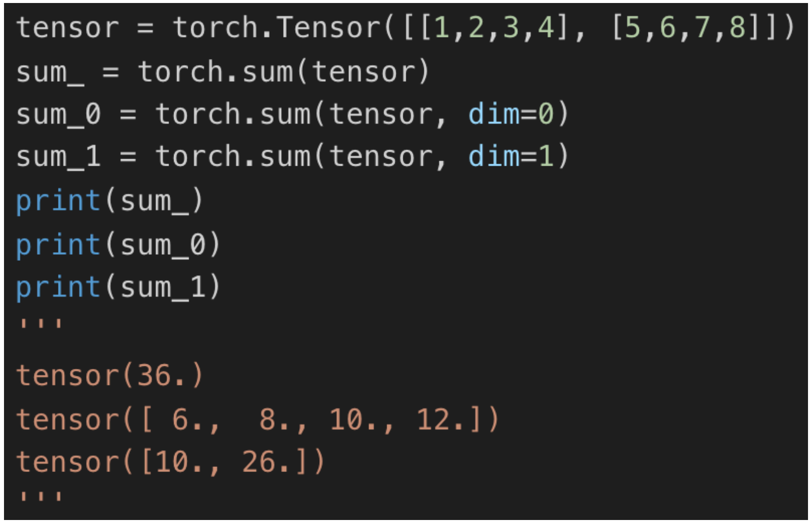
- Creating functions
  - Categorical distribution



```
probs = torch.Tensor([0.3, 0.2, 0.1, 0.4])

from torch.distributions import Categorical
categorical = Categorical(probs)
x = categorical.sample()
print(x)
'''
tensor(3)
'''
```

Tensor Manipulation

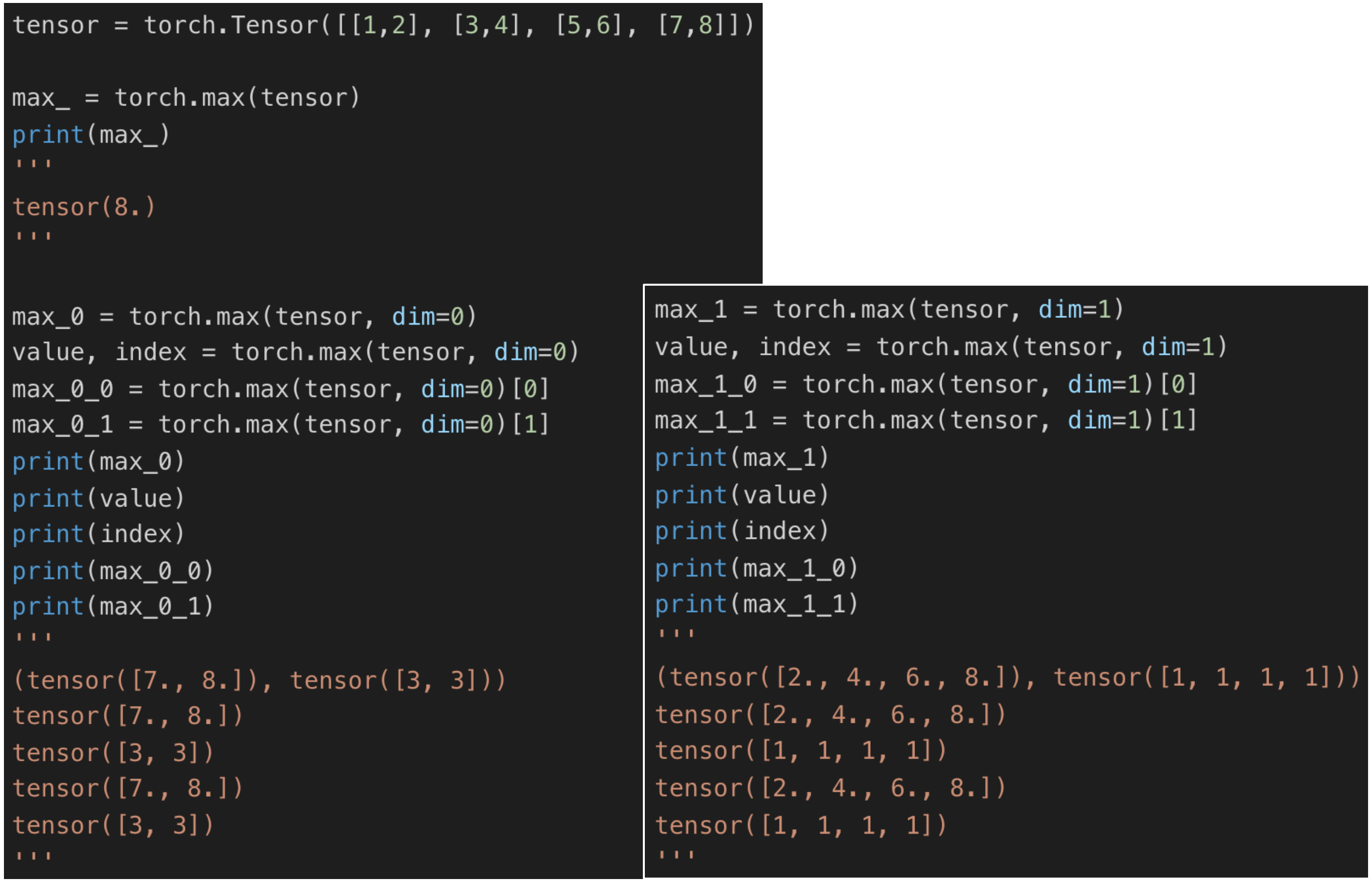
- Operation functions
  - Sum



```
tensor = torch.Tensor([[1,2,3,4], [5,6,7,8]])
sum_ = torch.sum(tensor)
sum_0 = torch.sum(tensor, dim=0)
sum_1 = torch.sum(tensor, dim=1)
print(sum_)
print(sum_0)
print(sum_1)
'''
tensor(36.)
tensor([ 6.,  8., 10., 12.])
tensor([10., 26.])
'''
```

Tensor Manipulation

- Operation functions
  - Max

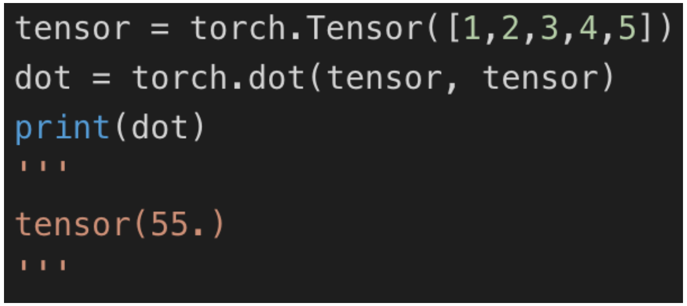


```
tensor = torch.Tensor([[1,2], [3,4], [5,6], [7,8]])

max_ = torch.max(tensor)
print(max_)
'''
tensor(8.)
'''

max_0 = torch.max(tensor, dim=0)
value, index = torch.max(tensor, dim=0)
max_0_0 = torch.max(tensor, dim=0)[0]
max_0_1 = torch.max(tensor, dim=0)[1]
print(max_0)
print(value)
print(index)
print(max_0_0)
print(max_0_1)
'''
(tensor([7., 8.]), tensor([3, 3]))
tensor([7., 8.])
tensor([3, 3])
tensor([7., 8.])
tensor([3, 3])
'''
```

```
max_1 = torch.max(tensor, dim=1)
value, index = torch.max(tensor, dim=1)
max_1_0 = torch.max(tensor, dim=1)[0]
max_1_1 = torch.max(tensor, dim=1)[1]
print(max_1)
print(value)
print(index)
print(max_1_0)
print(max_1_1)
'''
(tensor([2., 4., 6., 8.]), tensor([1, 1, 1, 1]))
tensor([2., 4., 6., 8.])
tensor([1, 1, 1, 1])
tensor([2., 4., 6., 8.])
tensor([1, 1, 1, 1])
'''
```

Tensor Manipulation

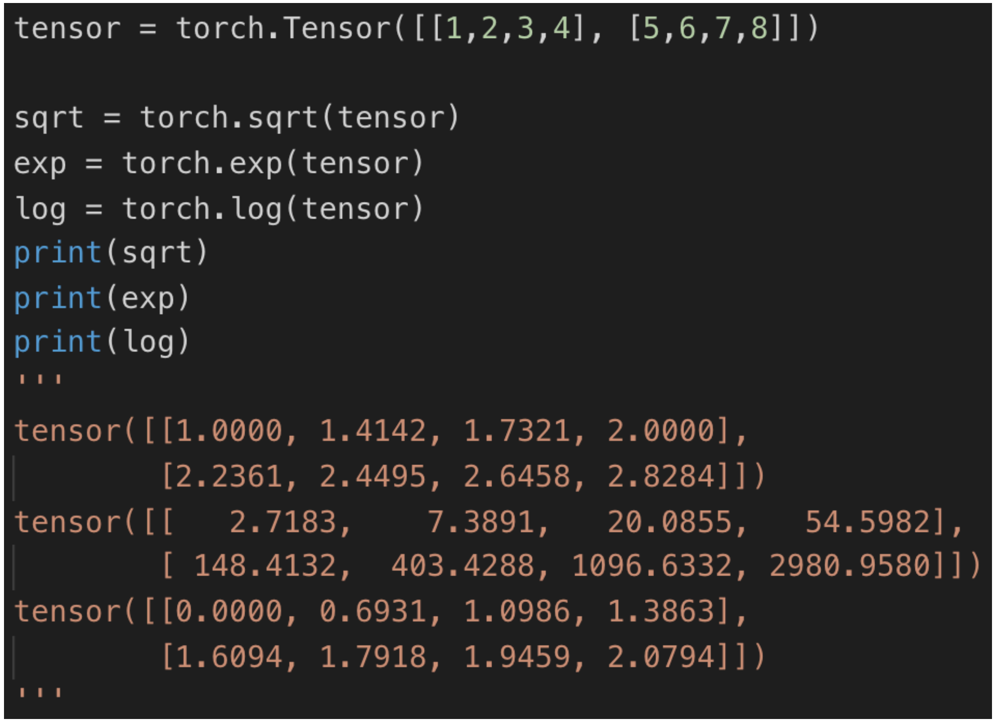
- Operation functions
  - Dot product
    - torch.dot - Computes the dot product of two tensors (1-Dimension)



```
tensor = torch.Tensor([1,2,3,4,5])
dot = torch.dot(tensor, tensor)
print(dot)
'''
tensor(55.)
'''
```

Tensor Manipulation

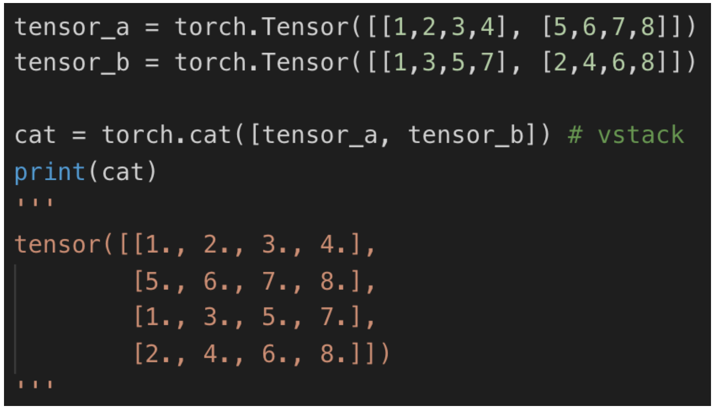
- Operation functions
  - Mathematical functions



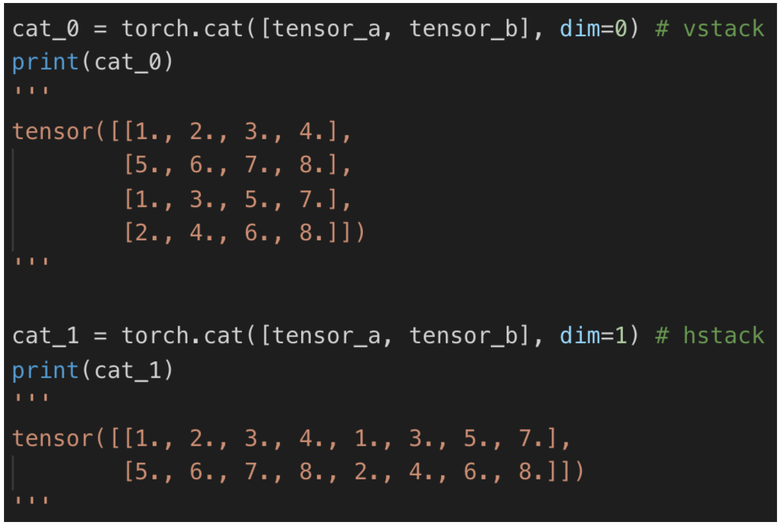
```
tensor = torch.Tensor([[1,2,3,4], [5,6,7,8]])

sqrt = torch.sqrt(tensor)
exp = torch.exp(tensor)
log = torch.log(tensor)
print(sqrt)
print(exp)
print(log)
'''
tensor([[1.0000, 1.4142, 1.7321, 2.0000],
        [2.2361, 2.4495, 2.6458, 2.8284]])
tensor([[   2.7183,    7.3891,   20.0855,   54.5982],
        [ 148.4132,  403.4288, 1096.6332, 2980.9580]])
tensor([[0.0000, 0.6931, 1.0986, 1.3863],
        [1.6094, 1.7918, 1.9459, 2.0794]])
'''
```

Tensor Manipulation

- Operation functions
  - Concatenate



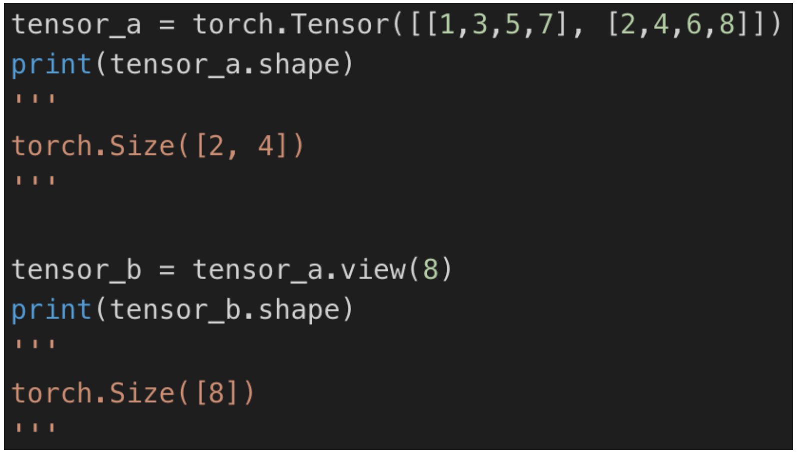
```python
tensor_a = torch.Tensor([[1,2,3,4], [5,6,7,8]])
tensor_b = torch.Tensor([[1,3,5,7], [2,4,6,8]])

cat = torch.cat([tensor_a, tensor_b]) # vstack
print(cat)
'''
tensor([[1., 2., 3., 4.],
        [5., 6., 7., 8.],
        [1., 3., 5., 7.],
        [2., 4., 6., 8.]])
'''
```



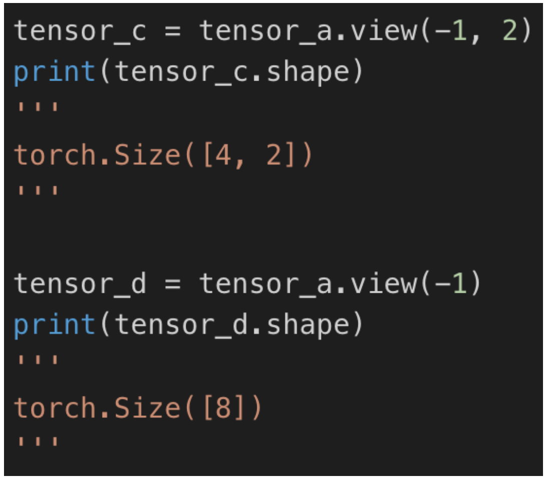
```python
cat_0 = torch.cat([tensor_a, tensor_b], dim=0) # vstack
print(cat_0)
'''
tensor([[1., 2., 3., 4.],
        [5., 6., 7., 8.],
        [1., 3., 5., 7.],
        [2., 4., 6., 8.]])
'''

cat_1 = torch.cat([tensor_a, tensor_b], dim=1) # hstack
print(cat_1)
'''
tensor([[1., 2., 3., 4., 1., 3., 5., 7.],
        [5., 6., 7., 8., 2., 4., 6., 8.]])
'''
```

Tensor Manipulation

- Operation functions
  - View
    - torch.view - Returns a new tensor with the same data as the self tensor but of a different shape



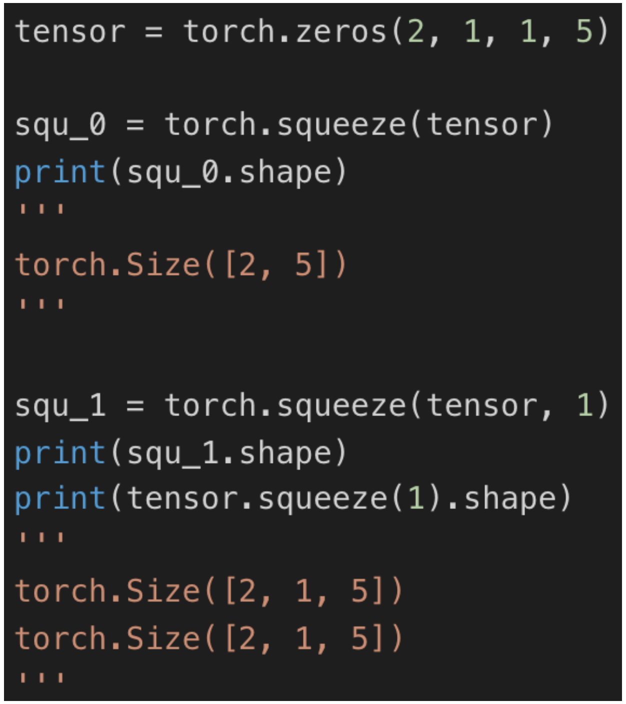
```python
tensor_a = torch.Tensor([[1,3,5,7], [2,4,6,8]])
print(tensor_a.shape)
'''
torch.Size([2, 4])
'''

tensor_b = tensor_a.view(8)
print(tensor_b.shape)
'''
torch.Size([8])
'''
```



```python
tensor_c = tensor_a.view(-1, 2)
print(tensor_c.shape)
'''
torch.Size([4, 2])
'''

tensor_d = tensor_a.view(-1)
print(tensor_d.shape)
'''
torch.Size([8])
'''
```

Tensor Manipulation

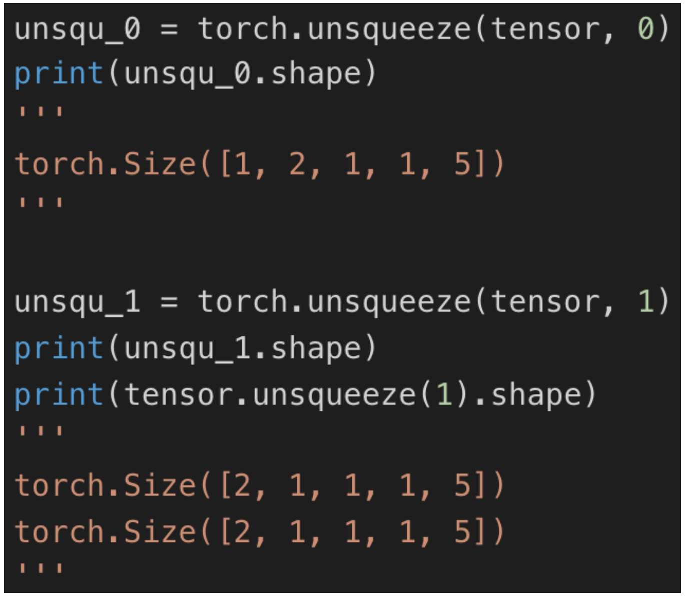
- Operation functions
  - Squeeze - torch.squeeze(input, dim=None)
    - Returns a tensor with all the dimensions of input of size 1 removed



```
tensor = torch.zeros(2, 1, 1, 5)

squ_0 = torch.squeeze(tensor)
print(squ_0.shape)
'''
torch.Size([2, 5])
'''


squ_1 = torch.squeeze(tensor, 1)
print(squ_1.shape)
print(tensor.squeeze(1).shape)
'''
torch.Size([2, 1, 5])
torch.Size([2, 1, 5])
'''
```

Tensor Manipulation

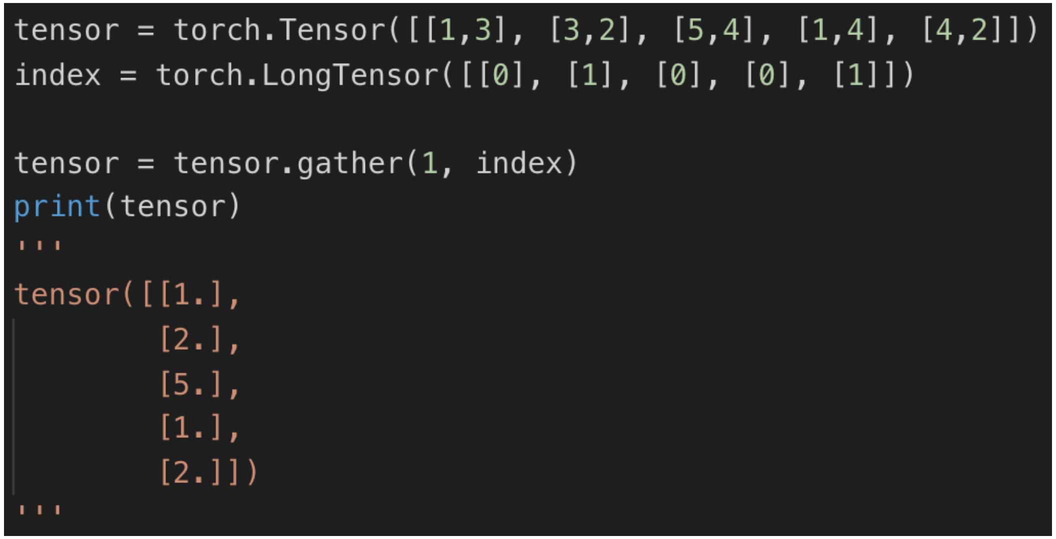
- Operation functions
  - Unsqueeze - torch.unsqueeze(input, dim)
    - Returns a new tensor with a dimension of size one inserted at the specified position



```
unsqu_0 = torch.unsqueeze(tensor, 0)
print(unsqu_0.shape)
'''
torch.Size([1, 2, 1, 1, 5])
'''


unsqu_1 = torch.unsqueeze(tensor, 1)
print(unsqu_1.shape)
print(tensor.unsqueeze(1).shape)
'''
torch.Size([2, 1, 1, 1, 5])
torch.Size([2, 1, 1, 1, 5])
'''
```

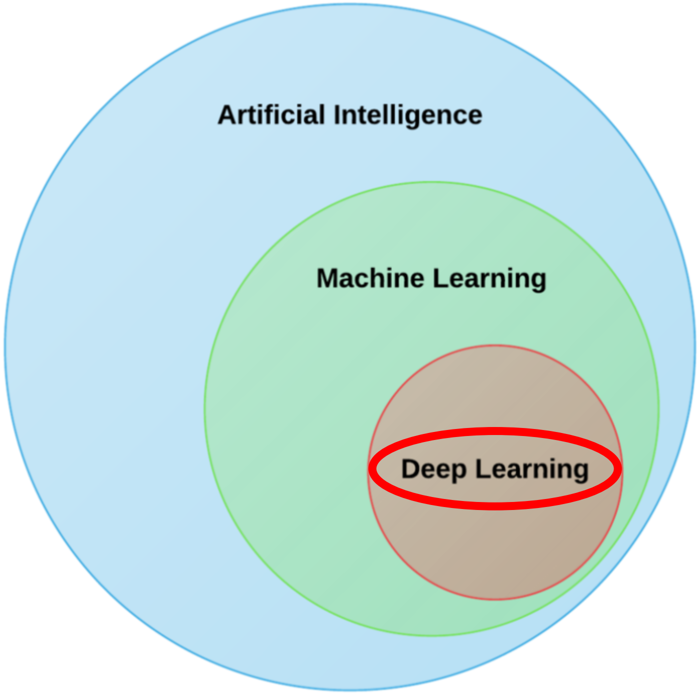
Tensor Manipulation

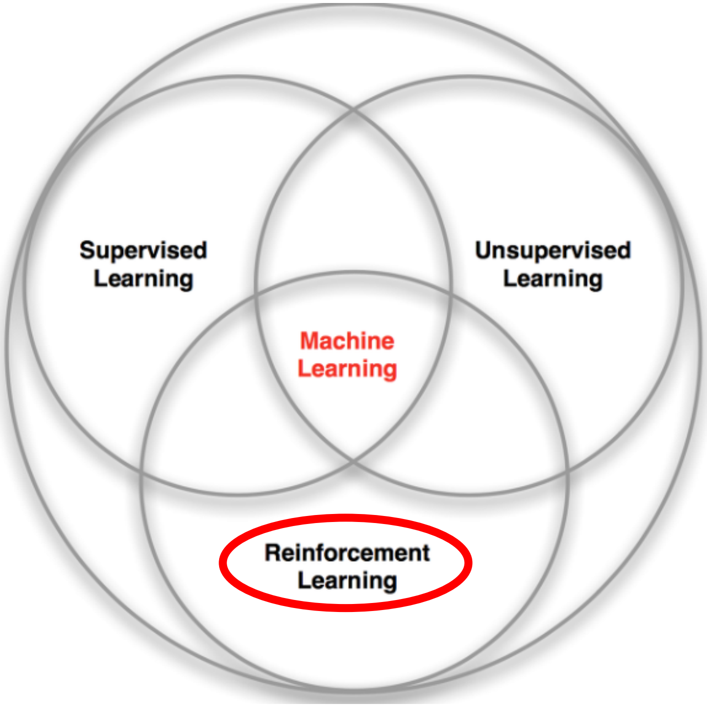
- Operation functions
  - Gather - gather(dim, index)



```
tensor = torch.Tensor([[1,3], [3,2], [5,4], [1,4], [4,2]])
index = torch.LongTensor([[0], [1], [0], [0], [1]])

tensor = tensor.gather(1, index)
print(tensor)
'''
tensor([[1.],
        [2.],
        [5.],
        [1.],
        [2.]])
'''
```

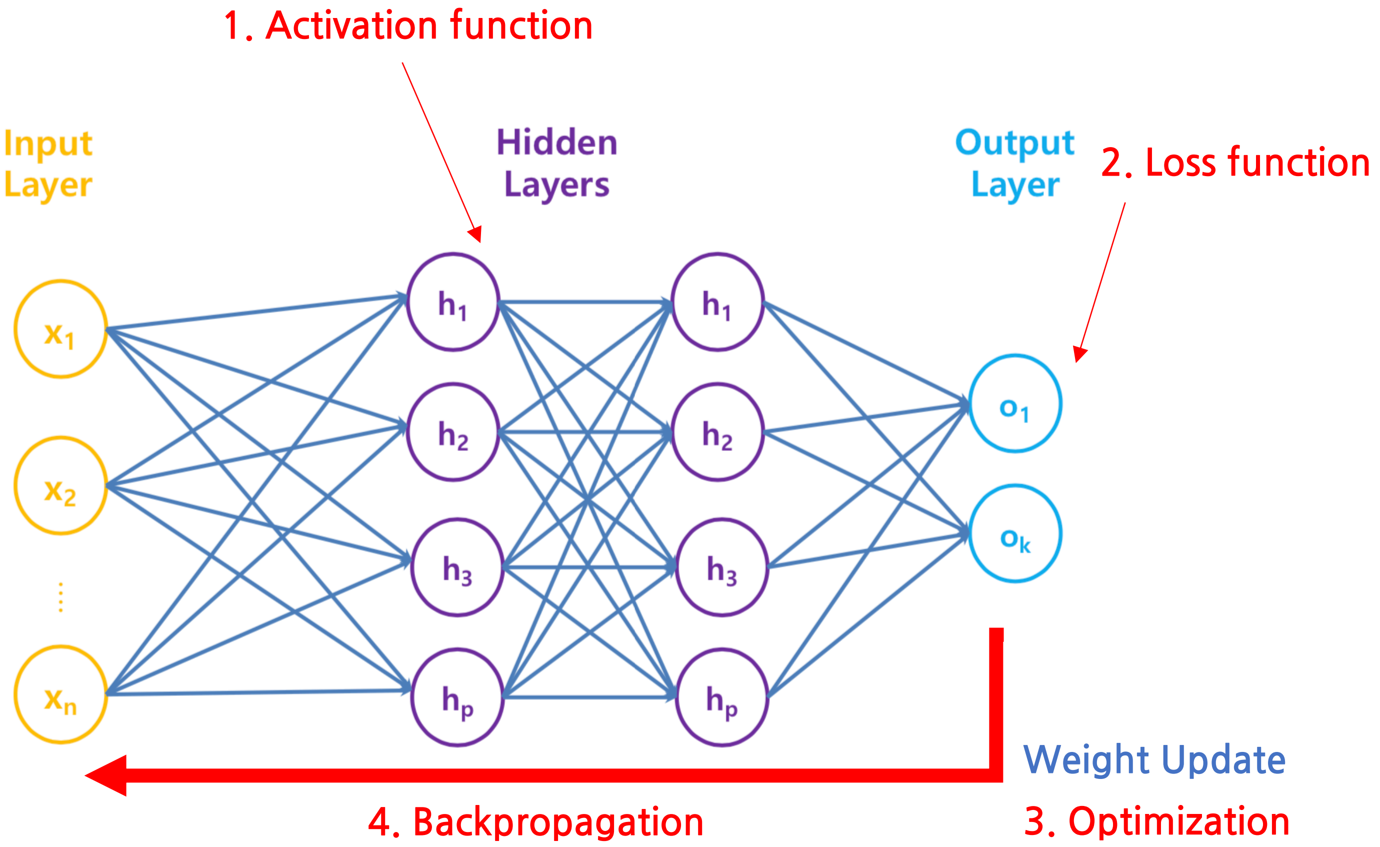
Deep Neural Network





Deep Neural Network

- Deep Neural Network (DNN) process



1. Activation function

2. Loss function

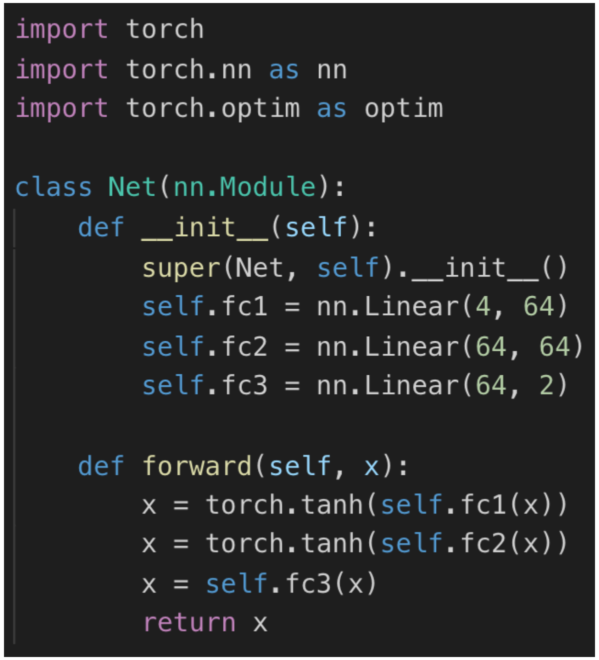
Weight Update

3. Optimization

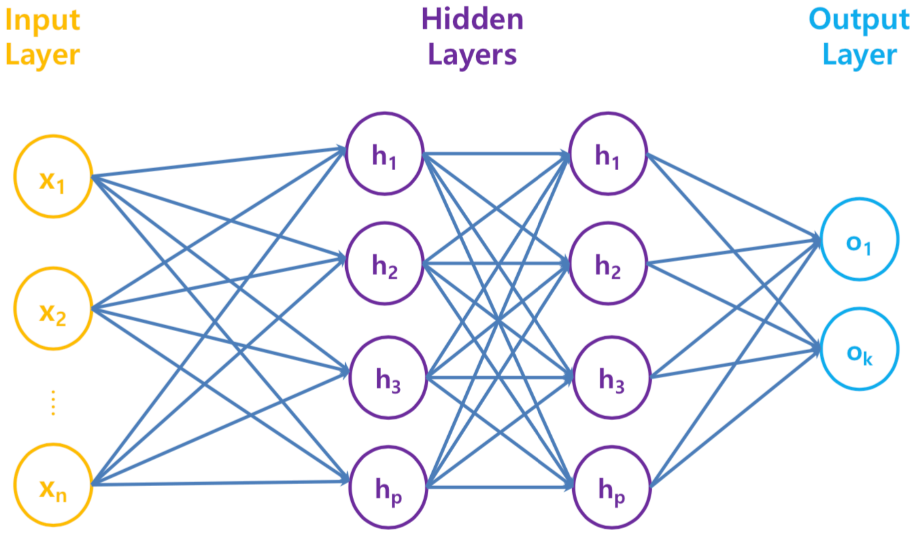
4. Backpropagation

Make Model with PyTorch

- PyTorch example for Deep Neural Network (DNN)



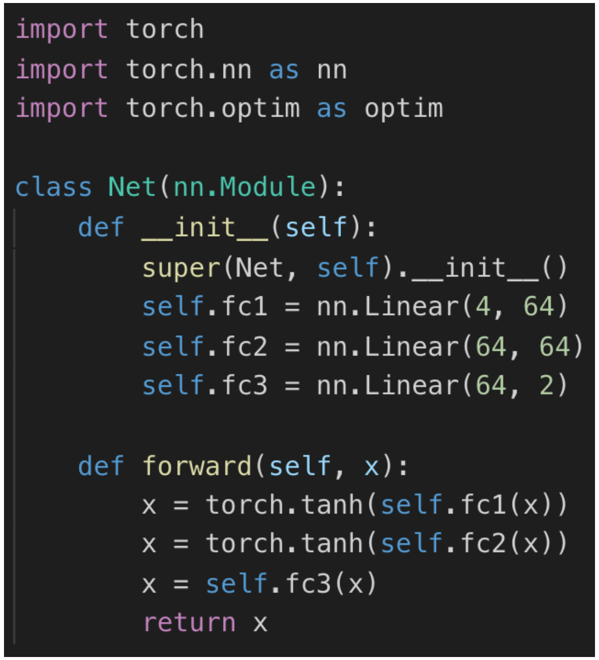
```
import torch
import torch.nn as nn
import torch.optim as optim

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(4, 64)
        self.fc2 = nn.Linear(64, 64)
        self.fc3 = nn.Linear(64, 2)

    def forward(self, x):
        x = torch.tanh(self.fc1(x))
        x = torch.tanh(self.fc2(x))
        x = self.fc3(x)
        return x
```



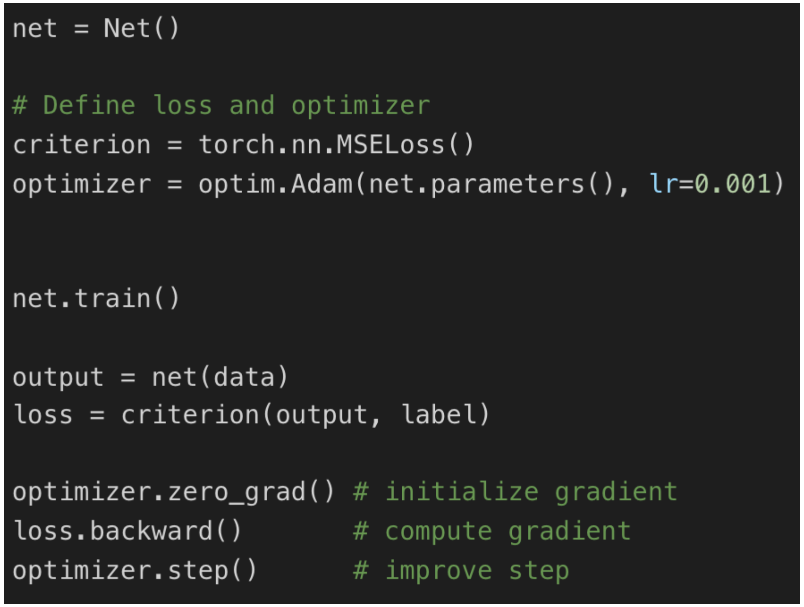
$$o_1 = \sum_{i=1}^{p} w_i \phi\left(\sum_{j=1}^{p} w_j' \phi\left(\sum_{k=1}^{n} w_k'' x_k\right)\right)$$

Make Model with PyTorch

- PyTorch example for Deep Neural Network (DNN)



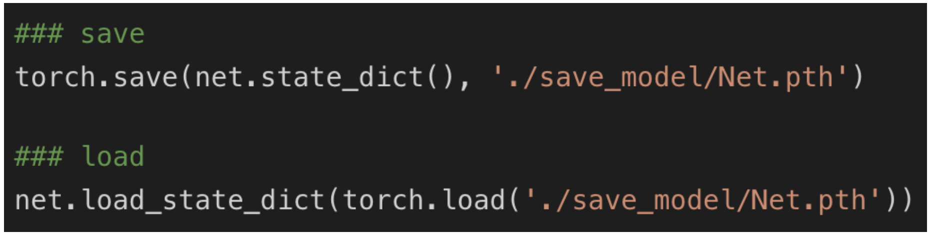
```python
import torch
import torch.nn as nn
import torch.optim as optim

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(4, 64)
        self.fc2 = nn.Linear(64, 64)
        self.fc3 = nn.Linear(64, 2)

    def forward(self, x):
        x = torch.tanh(self.fc1(x))
        x = torch.tanh(self.fc2(x))
        x = self.fc3(x)
        return x
```



```python
net = Net()

# Define loss and optimizer
criterion = torch.nn.MSELoss()
optimizer = optim.Adam(net.parameters(), lr=0.001)


net.train()

output = net(data)
loss = criterion(output, label)

optimizer.zero_grad() # initialize gradient
loss.backward()       # compute gradient
optimizer.step()      # improve step
```

Make Model with PyTorch

- Save & Load



```
### save
torch.save(net.state_dict(), './save_model/Net.pth')

### load
net.load_state_dict(torch.load('./save_model/Net.pth'))
```

Thank you