

Deep Learning with PyTorch

이동민

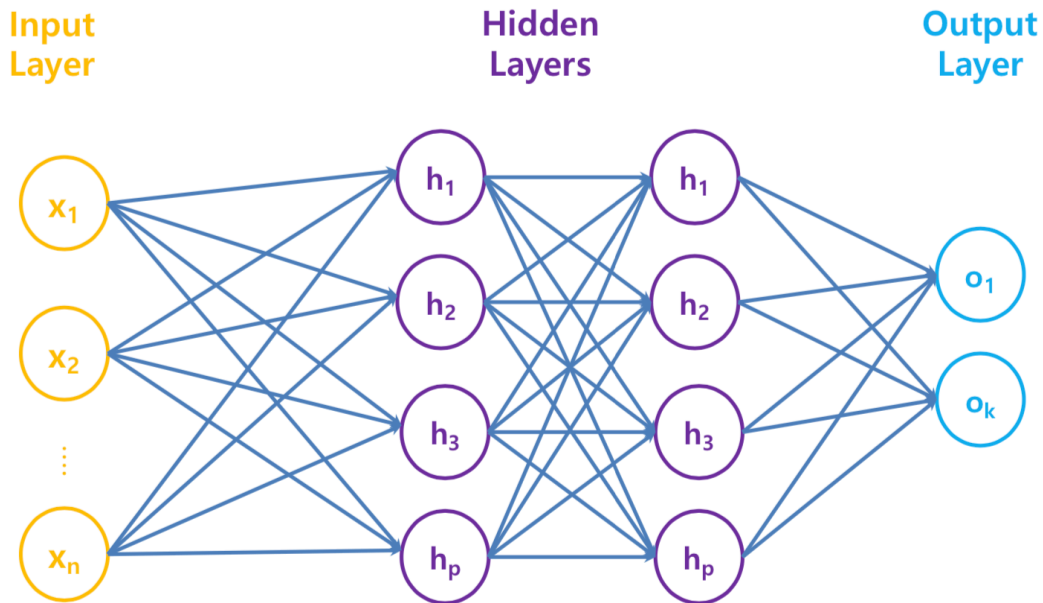
삼성전자 서울대 공동연구소
Jul 15, 2019

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



Yann LeCun
DIRECTOR OF AI RESEARCH
Facebook AI Research (FAIR)



P Y T O R C H



Soumith Chintala
RESEARCH ENGINEER
Facebook AI Research (FAIR)



CORE
Control + Optimization Research Lab

Tensor Manipulation

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

```
10  ### torch.Tensor
11  array = np.array([[1,2,3,4], [5,6,7,8]])
12  tensor = torch.Tensor(array)
13  print(array)
14  print(tensor)
15  '''
16  [[1 2 3 4]
17   [5 6 7 8]]
18  tensor([[1., 2., 3., 4.],
19         [5., 6., 7., 8.]])
20  '''
21
22  array[0,0] = 10
23  print(array)
24  print(tensor)
25  '''
26  [[10  2  3  4]
27   [ 5  6  7  8]]
28  tensor([[1., 2., 3., 4.],
29         [5., 6., 7., 8.]])
30  '''
```



Tensor Manipulation

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

```
32     ### torch.from_numpy
33     array = np.array([[1,3,5,7], [9,11,13,15]])
34     tensor = torch.from_numpy(array)
35     print(array)
36     print(tensor)
37     '''
38     [[ 1  3  5  7]
39     |
39     [ 9 11 13 15]]
40     tensor([[ 1,  3,  5,  7],
41     |
41             [ 9, 11, 13, 15]])
42     '''
43
44     array[0][0] = 10
45     print(array)
46     print(tensor)
47     '''
48     [[10  3  5  7]
49     |
49     [ 9 11 13 15]]
50     tensor([[10,  3,  5,  7],
51     |
51             [ 9, 11, 13, 15]])
52     '''
```



Tensor Manipulation

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

```
58 tensor = torch.Tensor([[1,2,3,4], [5,6,7,8]])
59 array = tensor.numpy()
60 print(tensor)
61 print(array)
62 '''
63 tensor([[1., 2., 3., 4.],
64         [5., 6., 7., 8.]])
65 [[1. 2. 3. 4.]
66  [5. 6. 7. 8.]]
67 '''
```

Tensor Manipulation

- Creating functions
 - Zeros & Ones

```
73  ### Zeros & Ones
74  zeros = torch.zeros((2, 5))
75  print(zeros)
76  '''
77  tensor([[0., 0., 0., 0., 0.],
78          |
79          [0., 0., 0., 0., 0.]])
80
81  ones = torch.ones((5, 2))
82  print(ones)
83  '''
84  tensor([[1., 1.],
85          |
86          [1., 1.],
87          |
88          [1., 1.],
89          |
89          [1., 1.]])
```



Tensor Manipulation

- Creating functions
 - Something like

```
91  ### Something_like
92  tensor = torch.Tensor([[1,2,3,4], [5,6,7,8]])
93  zeros = torch.zeros_like(tensor)
94  print(zeros)
95  '''
96  tensor([[0., 0., 0., 0.],
97         [0., 0., 0., 0.]])
98  '''
```



Tensor Manipulation

- Creating functions

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

```
100  ### Rand - Uniform distribution over [0, 1)
101  rand_sampling = torch.rand(2,5)
102  print(rand_sampling)
103  '''
104  tensor([[0.1562, 0.7464, 0.0341, 0.1269, 0.7245],
105         [0.7135, 0.6891, 0.9348, 0.4983, 0.9259]])
106  '''
```

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

```
108  ### Randn - Standard normal(gaussian) distribution of mean 0 and variance 1
109  randn_sampling = torch.randn(2,5)
110  print(randn_sampling)
111  '''
112  tensor([[ 0.4119, -1.1501, -0.4142,  0.9698, -0.9407],
113         [ 0.5520,  2.3532,  0.5251, -1.1743,  0.5667]])
114  '''
```



Tensor Manipulation

- Operation functions

- Sum

```
120     ### Sum
121     tensor = torch.Tensor([[1,2,3,4], [5,6,7,8]])
122     sum_ = torch.sum(tensor)
123     sum_0 = torch.sum(tensor, dim=0)
124     sum_1 = torch.sum(tensor, dim=1)
125     print(sum_)
126     print(sum_0)
127     print(sum_1)
128     '''
129     tensor(36.)
130     tensor([ 6.,  8., 10., 12.])
131     tensor([10., 26.])
132     '''
```



Tensor Manipulation

- Operation functions
 - Max

```
134  ### Max
135  tensor = torch.Tensor([[1,2], [3,4], [5,6], [7,8]])
136
137  max_ = torch.max(tensor)
138  print(max_)
139  '''
140  tensor(8.)
141  '''
142
143  max_0 = torch.max(tensor, dim=0)
144  value, index = torch.max(tensor, dim=0)
145  max_0_0 = torch.max(tensor, dim=0)[0]
146  max_0_1 = torch.max(tensor, dim=0)[1]
147  print(max_0)
148  print(value)
149  print(index)
150  print(max_0_0)
151  print(max_0_1)
152  '''
153  (tensor([7., 8.]), tensor([3, 3]))
154  tensor([7., 8.])
155  tensor([3, 3])
156  tensor([7., 8.])
157  tensor([3, 3])
158  '''
```

```
160  max_1 = torch.max(tensor, dim=1)
161  value, index = torch.max(tensor, dim=1)
162  max_1_0 = torch.max(tensor, dim=1)[0]
163  max_1_1 = torch.max(tensor, dim=1)[1]
164  print(max_1)
165  print(value)
166  print(index)
167  print(max_1_0)
168  print(max_1_1)
169  '''
170  (tensor([2., 4., 6., 8.]), tensor([1, 1, 1, 1]))
171  tensor([2., 4., 6., 8.])
172  tensor([1, 1, 1, 1])
173  tensor([2., 4., 6., 8.])
174  tensor([1, 1, 1, 1])
175  '''
```



Tensor Manipulation

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

$$\langle [x_1, \dots, x_n], [y_1, \dots, y_n] \rangle = x^T y = \sum_{i=1}^n x_i y_i = x_1 y_1 + \dots + x_n y_n$$

```
177     ### Dot product
178     tensor = torch.Tensor([1,2,3,4,5])
179     dot = torch.dot(tensor, tensor)
180     print(dot)
181     '''
182     tensor(55.)
183     '''
```



Tensor Manipulation

- Operation functions
 - Mathematical functions
 - `torch.sqrt` : \sqrt{x}
 - `torch.exp` : e^x
 - `torch.log` : $\log_e x$

```
185     ### Mathematical functions
186     tensor = torch.Tensor([[1,2,3,4], [5,6,7,8]])
187
188     sqrt = torch.sqrt(tensor)
189     exp = torch.exp(tensor)
190     log = torch.log(tensor)
191     print(sqrt)
192     print(exp)
193     print(log)
194     '''
195     tensor([[1.0000, 1.4142, 1.7321, 2.0000],
196            [2.2361, 2.4495, 2.6458, 2.8284]])
197     tensor([[ 2.7183,  7.3891, 20.0855, 54.5982],
198            [148.4132, 403.4288, 1096.6332, 2980.9580]])
199     tensor([[0.0000, 0.6931, 1.0986, 1.3863],
200            [1.6094, 1.7918, 1.9459, 2.0794]])
201     '''
```



Tensor Manipulation

- Operation functions
 - Concatenate

```
203  ### Concatenate
204  tensor_a = torch.Tensor([[1,2,3,4], [5,6,7,8]])
205  tensor_b = torch.Tensor([[1,3,5,7], [2,4,6,8]])
206
207  cat = torch.cat([tensor_a, tensor_b]) # vstack
208  print(cat)
209  '''
210  tensor([[1., 2., 3., 4.],
211          [5., 6., 7., 8.],
212          [1., 3., 5., 7.],
213          [2., 4., 6., 8.]])
214  '''
215
216  cat_0 = torch.cat([tensor_a, tensor_b], dim=0) # vstack
217  print(cat_0)
218  '''
219  tensor([[1., 2., 3., 4.],
220          [5., 6., 7., 8.],
221          [1., 3., 5., 7.],
222          [2., 4., 6., 8.]])
223  '''
224
225  cat_1 = torch.cat([tensor_a, tensor_b], dim=1) # hstack
226  print(cat_1)
227  '''
228  tensor([[1., 2., 3., 4., 1., 3., 5., 7.],
229          [5., 6., 7., 8., 2., 4., 6., 8.]])
230  '''
```



Tensor Manipulation

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

```
232     ### View
233     tensor_a = torch.Tensor([[1,3,5,7], [2,4,6,8]])
234
235     tensor_b = tensor_a.view(8)
236     print(tensor_b.shape)
237     '''
238     torch.Size([8])
239     '''
240
241     tensor_c = tensor_a.view(-1, 2)
242     print(tensor_c.shape)
243     '''
244     torch.Size([4, 2])
245     '''
246
247     tensor_d = tensor_a.view(-1)
248     print(tensor_d.shape)
249     '''
250     torch.Size([8])
251     '''
```



Tensor Manipulation

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

```
253     ### Squeeze & Unsqueeze
254     tensor = torch.zeros(2, 1, 1, 5)
255
256     squ_0 = torch.squeeze(tensor)
257     print(squ_0.shape)
258     '''
259     torch.Size([2, 5])
260     '''
261
262     squ_1 = torch.squeeze(tensor, 1)
263     print(squ_1.shape)
264     print(tensor.squeeze(1).shape)
265     '''
266     torch.Size([2, 1, 5])
267     torch.Size([2, 1, 5])
268     '''
```



Tensor Manipulation

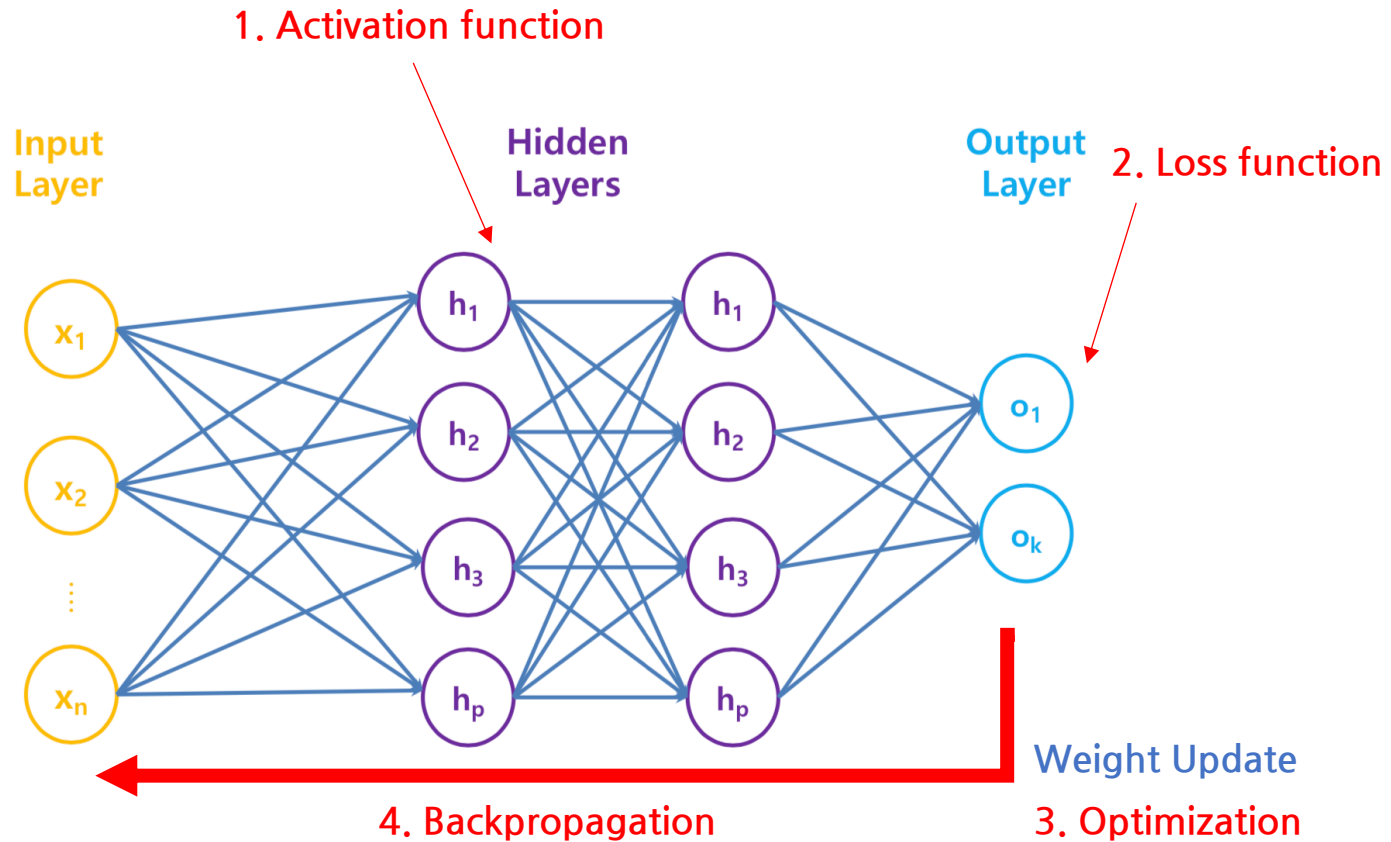
- Operation functions
 - Unsqueeze - `torch.unsqueeze(input, dim)`
 - Returns a new tensor with a dimension of size one inserted at the specified position

```
270 unsqu_0 = torch.unsqueeze(tensor, 0)
271 print(unsqu_0.shape)
272 '''
273 torch.Size([1, 2, 1, 1, 5])
274 '''
275
276 unsqu_1 = torch.unsqueeze(tensor, 1)
277 print(unsqu_1.shape)
278 print(tensor.unsqueeze(1).shape)
279 '''
280 torch.Size([2, 1, 1, 1, 5])
281 torch.Size([2, 1, 1, 1, 5])
282 '''
```



Deep Neural Network

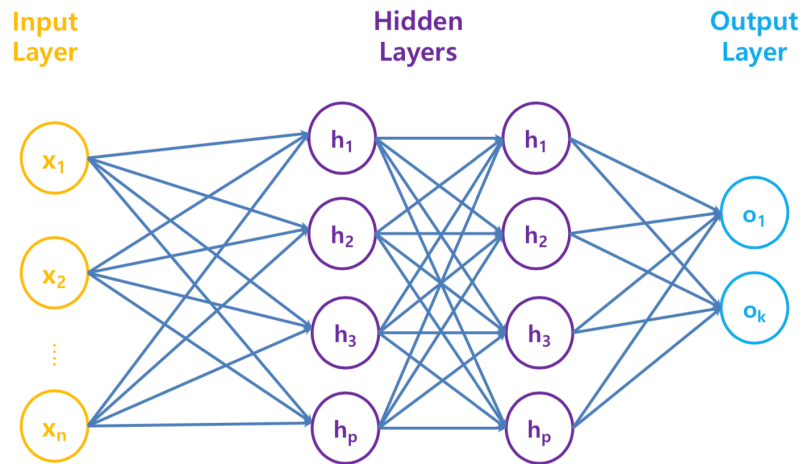
- Deep Neural Network (DNN) process



Make Model with PyTorch

- PyTorch example for Deep Neural Network (DNN)

```
294 import torch
295 import torch.nn as nn
296 import torch.optim as optim
297
298 class Net(nn.Module):
299     def __init__(self):
300         super(Net, self).__init__()
301         self.fc1 = nn.Linear(4, 64)
302         self.fc2 = nn.Linear(64, 64)
303         self.fc3 = nn.Linear(64, 2)
304
305     def forward(self, x):
306         x = torch.tanh(self.fc1(x))
307         x = torch.tanh(self.fc2(x))
308         x = self.fc3(x)
309         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)

```
294 import torch
295 import torch.nn as nn
296 import torch.optim as optim
297
298 class Net(nn.Module):
299     def __init__(self):
300         super(Net, self).__init__()
301         self.fc1 = nn.Linear(4, 64)
302         self.fc2 = nn.Linear(64, 64)
303         self.fc3 = nn.Linear(64, 2)
304
305     def forward(self, x):
306         x = torch.tanh(self.fc1(x))
307         x = torch.tanh(self.fc2(x))
308         x = self.fc3(x)
309         return x
```

```
311 net = Net()
312
313 # Define loss and optimizer
314 criterion = torch.nn.MSELoss()
315 optimizer = optim.Adam(net.parameters(), lr=0.001)
316
317
318 net.train()
319
320 hypothesis = net(inputs)
321 loss = criterion(hypothesis, labels)
322
323 optimizer.zero_grad() # initialize gradient
324 loss.backward()       # compute gradient
325 optimizer.step()      # improve step
```



Make Model with PyTorch

- Save & Load

```
331     ### save
332     torch.save(net.state_dict(), './save_model/model.pth')
333
334     ### load
335     net.load_state_dict(torch.load('./save_model/model.pth'))
```



Make Model with PyTorch

- MNIST example

Thank you

