Introduction to PyTorch

Heni Ben Amor, Ph.D.
Assistant Professor
Arizona State University

Ira A. Fulton Schools of
Engineering
Arizona State University

Neural Networks with PyTorch

- PyTorch is an open source machine learning library for Python
- Wide range of networks and training algorithms
- Allows for dynamic networks
- Accessible

Defining a Simple Network in PyTorch

architecture

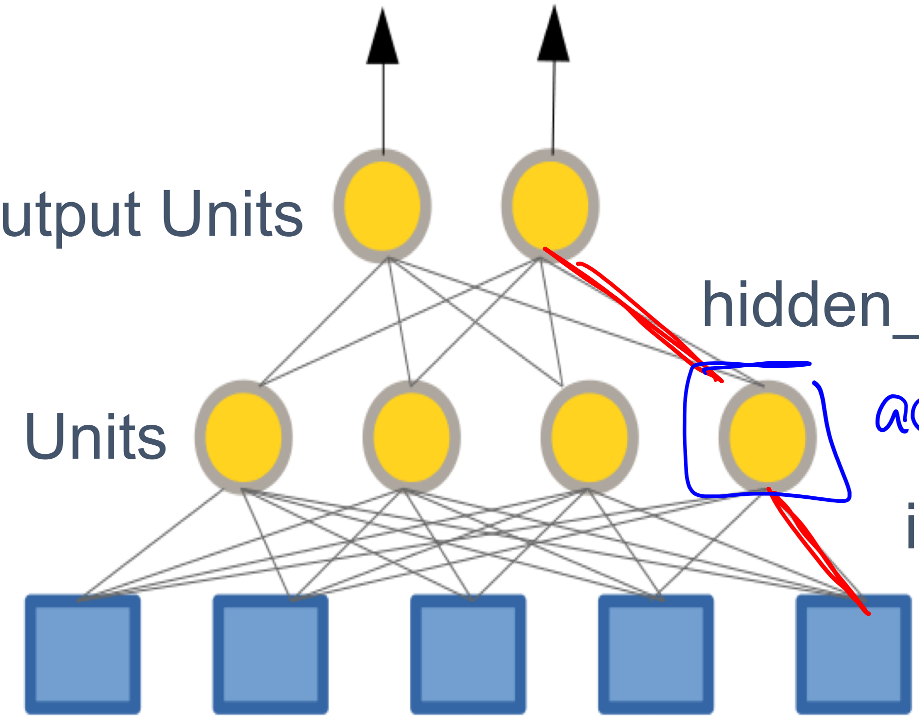
```
import torch.nn as nn
class Basic_Network(nn.Module):
    def __init__(self,input_size=5,hidden_size=4,output_size=2):
        super(Basic_Network, self).__init__()
        self.input_to_hidden = nn.Linear(input_size, hidden_size)
        self.nonlinear_activation = nn.Sigmoid()
        self.hidden_to_output = nn.Linear(hidden_size,
                                          output_size)
```

Computation

```
    def forward(self, network_input):
        hidden = self.input_to_hidden(network_input)
        hidden = self.nonlinear_activation(hidden)
        network_output = self.hidden_to_output(hidden)
        return network_output
model = Basic_Network(input_size, hidden_size, output_size)
...
network_output = model(network_input)
```

x

Simple Network



Output Units

hidden_to_output(...)

activation function

σ

Hidden Units

input_to_hidden(...)

Input Units

The Loss Function

(Training)

- A loss function takes the (output, target) pair and computes a measure which indicates how far away the output is from the target
- There are several loss function that can be used.
  - Let us use **nn.MSELoss()**

$\hat{y}$ y

```
loss_function = nn.MSELoss()
loss = loss_function(network_output, target_output)
print(loss.item())

loss.backward()
```

- When we call loss.backward(), the whole graph is differentiated with respect to the loss, and all Variables in the graph will have their .grad Variable accumulated with the gradient

Training the Weights of a Network

- The most frequent update rule used in practice is Stochastic Gradient Descent (SGD)
- Many sophisticated learning methods are also implemented: Nesterov-SGD, Adam, RMSProp
- Torch.optim allows you to change learning method

```
learning_rate = 0.01
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)

optimizer.zero_grad()
output = model(network_input)
loss = loss_function(network_output, target_output)
loss.backward()
optimizer.step()
```

← gradient descent

- optimizer.zero_grad() - zeros the gradient buffer and optimizer.step() - updates the weights

Summary

- We introduced PyTorch
- Easy specification of neural networks
- Wide range of functionality provided
- Fast setup of neural networks