A Brief Introduction to PyTorch

(a deep learning library)

Huaipeng Zhao

pytorch.org

Outline

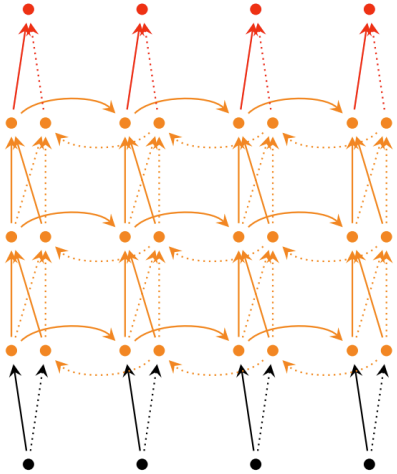
- ☐ **Why Deep Learning Libraries and Why PyTorch ?**
- ☐ **A Brief Introduction to PyTorch**
- ☐ **PyTorch in action**

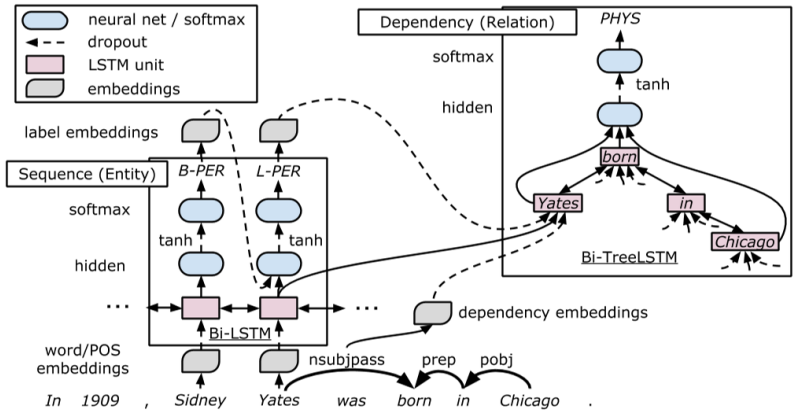
Outline

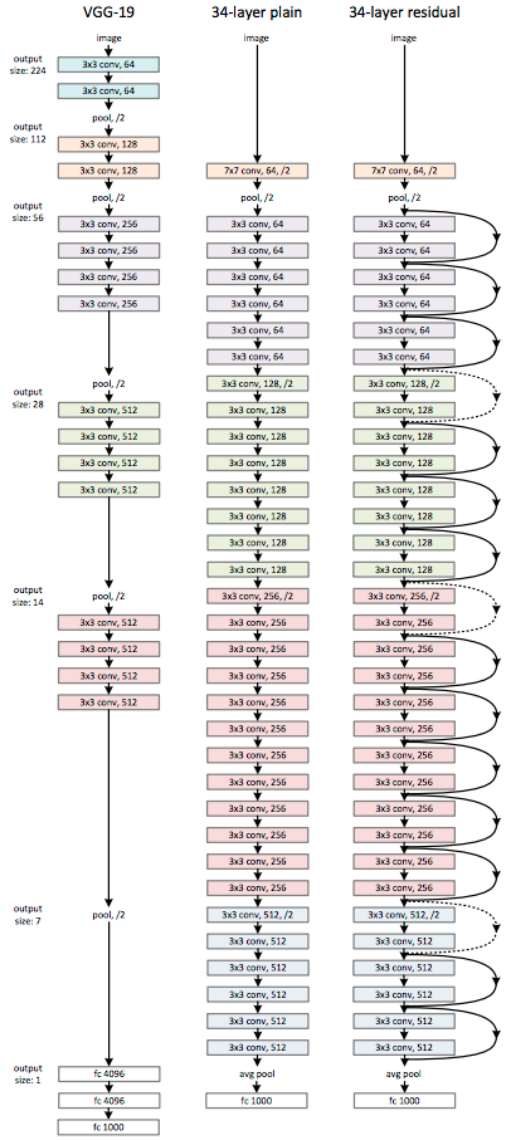
- ☐ **Why Deep Learning Libraries and Why PyTorch ?**
- ☐ **A Brief Introduction to PyTorch**
- ☐ **PyTorch in action**

Why a DL Library is Necessary?

- Complicated DL architectures
  - Easily build big computational graphs
  - Easily compute gradients

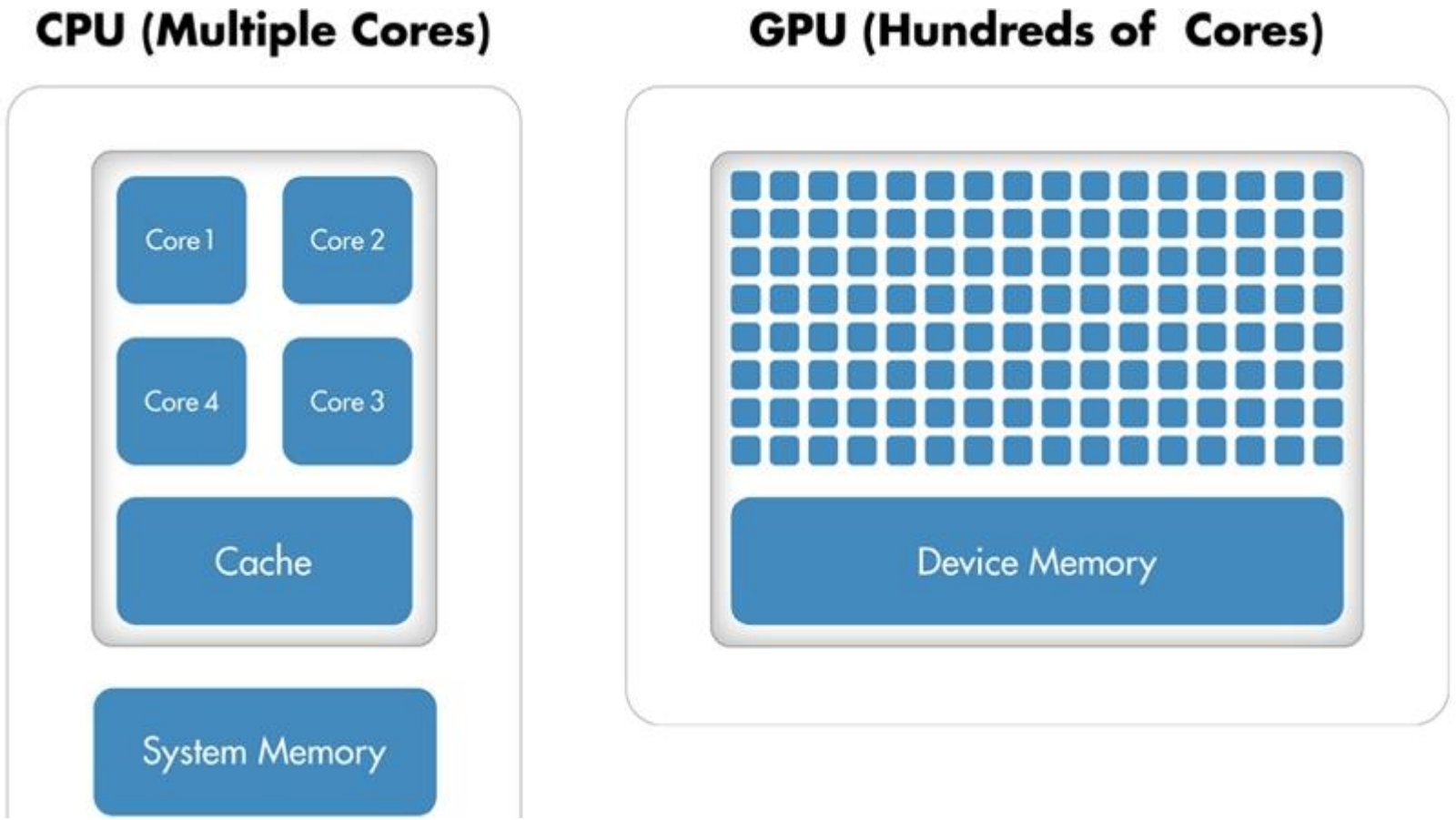




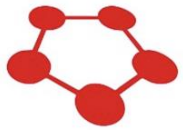


Why a DL Library is Necessary?

- Run it all efficiently on GPU



Popular Deep Learning Libraries



Why PYTORCH

Life is short, you need PYTORCH

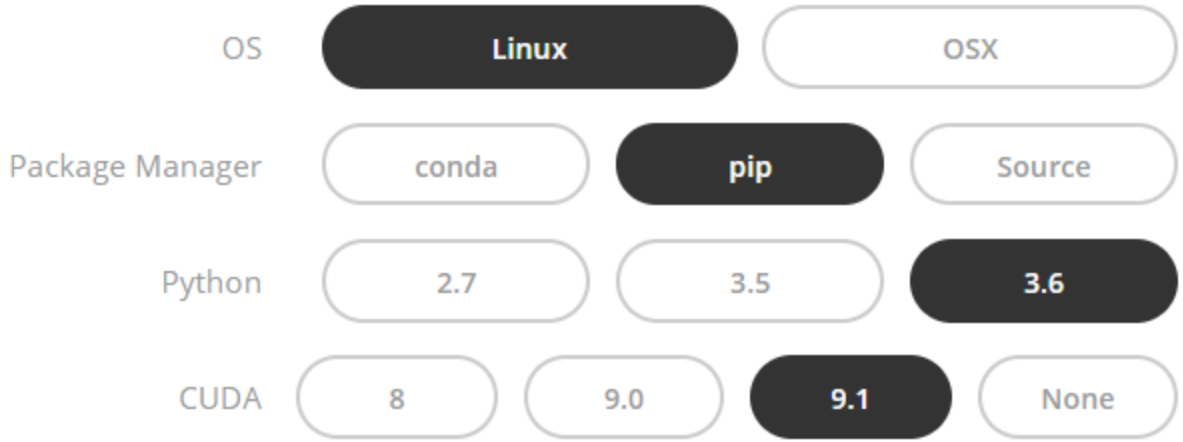
Why PYTORCH

- Support Python
- Dynamic Graph
- Friendly API for research
- Easily debug
- Love all things [*Pythonic*]

Outline

- ☐ Why Deep Learning Libraries and Why PyTorch ?
- ☐ **A Brief Introduction to PyTorch**
- ☐ PyTorch in action

Installing PYTORCH



Run this command:

```
pip3 install http://download.pytorch.org/whl/cu91/torch-0.3.1-cp36-cp36m-linux_x86_64.whl
pip3 install torchvision
```

Three Levels of Abstraction

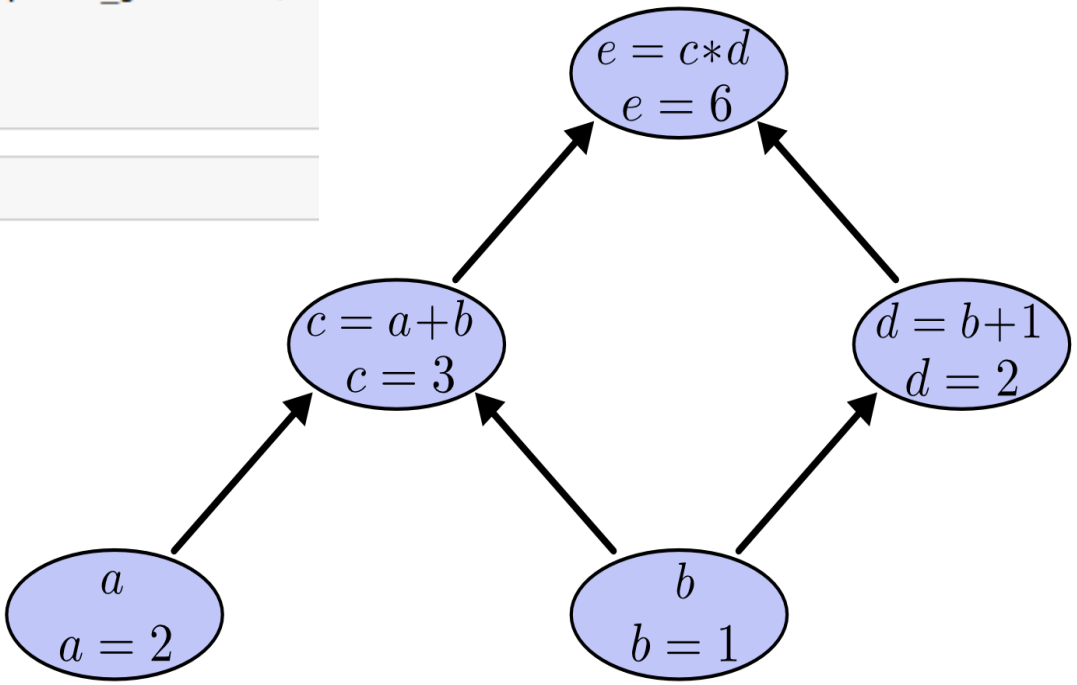
- **Tensor:** Imperative ndarray
- **Variable:** Node in a computational graph (data,grad)
- **Module:** A neural network layer

```
In [1]: import torch
        x = torch.FloatTensor([[1.0,2.0],[3.0,4.0]])
        y = torch.FloatTensor(torch.randn(2,2))
        z = torch.randn(2,2).type(torch.FloatTensor)

In [2]: var = torch.autograd.Variable(x)

In [3]: fc = torch.nn.Linear(2,2)
```

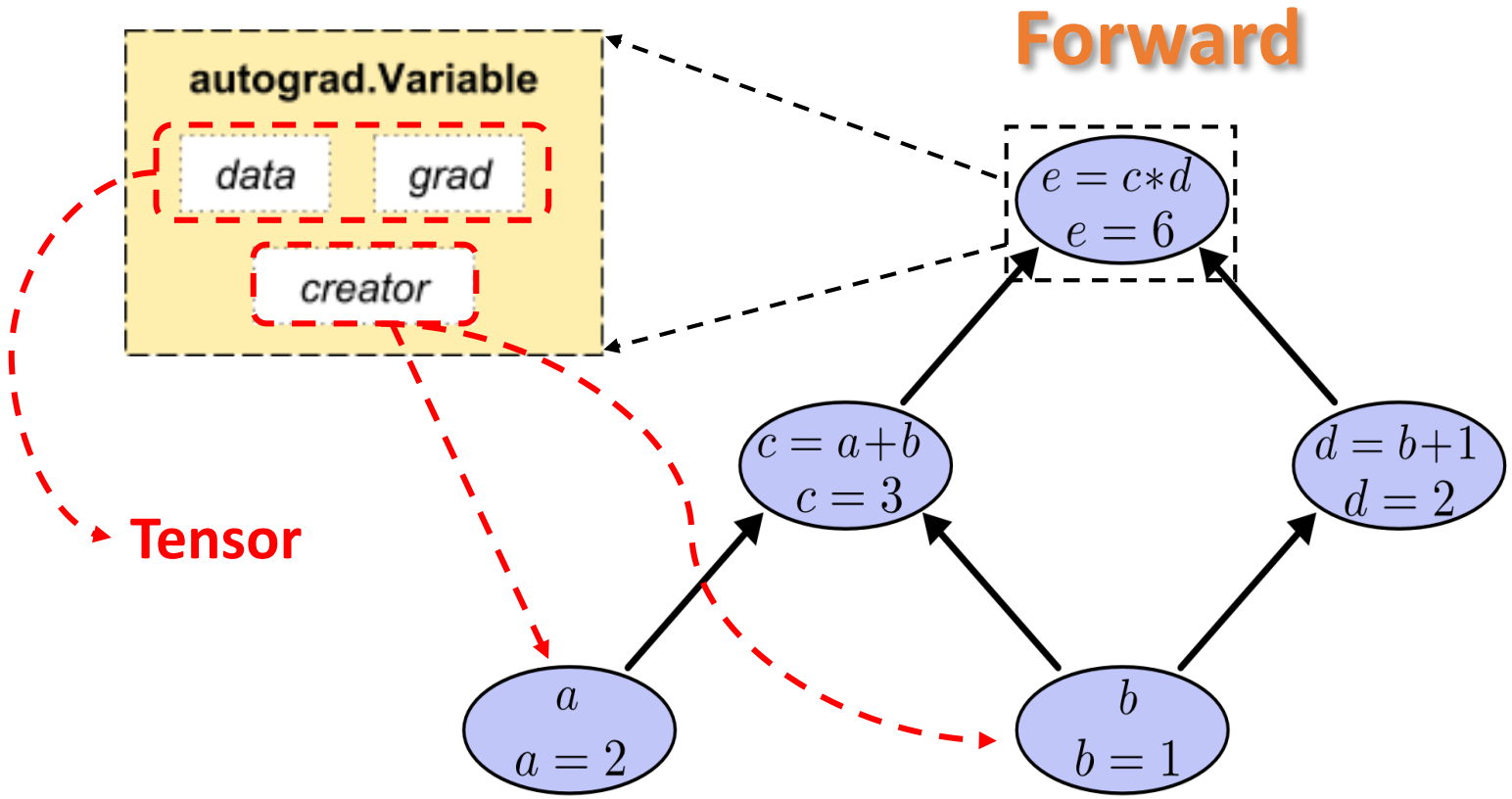
A Toy Neural Network

```
In [1]: import torch
        from torch.autograd import Variable

In [2]: a = Variable(torch.LongTensor([2]),requires_grad=True)
        b = Variable(torch.LongTensor([1]),requires_grad=True)
        c = a + b
        d = b + 1
        e = c * d

In [3]: print a,b,c,d,e
```

Forward



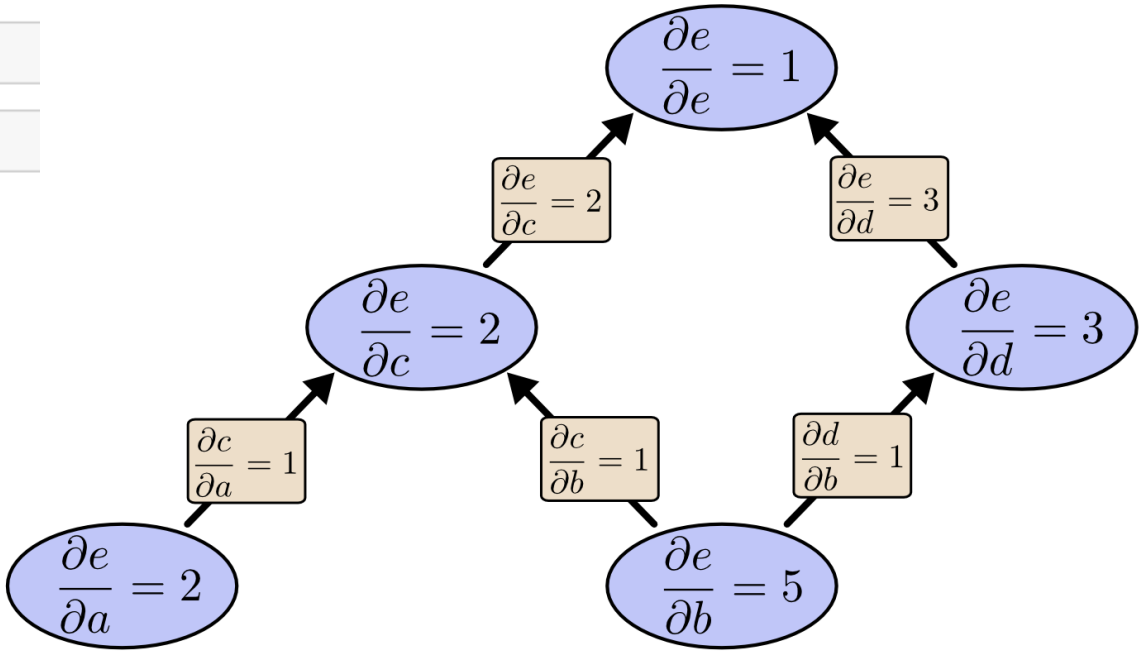
A Toy Neural Network



A Toy Neural Network

```
In [4]: e.backward()

In [5]: print a.grad,b.grad
```

Backward



A Toy Neural Network

In [1]:
```
import tensorflow as tf

a = tf.placeholder(tf.float32)
b = tf.placeholder(tf.float32)

c = a + b
d = b + 1
e = c * d

grad_a,grad_b = tf.gradients(e,[a,b])

with tf.Session() as sess:
    grad_a,grad_b = sess.run([grad_a,grad_b],feed_dict={a:2,b:1})
```

In [1]:
```
import torch
from torch.autograd import Variable

a = Variable(torch.LongTensor([2]),requires_grad=True)
b = Variable(torch.LongTensor([1]),requires_grad=True)
c = a + b
d = b + 1
e = c * d

e.backward()

print a.grad.data,b.grad.data
```

Major Components of PyTorch

Package	Description
torch	a Tensor library like NumPy, with strong GPU support
torch.autograd	a tape based automatic differentiation library that supports all differentiable Tensor operations in torch
torch.nn	a neural networks library deeply integrated with autograd designed for maximum flexibility
torch.optim	an optimization package to be used with torch.nn with standard optimization methods such as SGD, RMSProp, LBFGS, Adam etc.
torch.utils	DataLoader, Dataset and other utility functions for convenience

Torch Module:PyTorch as A Tensor Library

- Tensor operations: slicing, indexing, math operations, linear algebra, reductions
  - CPU & GPU
  - Fast! (comparison on speed of **matrix multiplication**)

$$\boldsymbol{M} * \boldsymbol{M} * \boldsymbol{M} \quad M \in \mathbb{R}^{1000\times1000}$$

Numpy

```
In [2]: M = numpy.random.randn(1000,1000)

In [3]: timeit -n 500 M.dot(M).dot(M)
500 loops, best of 3: 30.7 ms per loop
```

PyTorch

```
In [4]: N = torch.randn(1000,1000).cuda()

In [5]: timeit -n 500 N.mm(N).mm(N)
500 loops, best of 3: 474 µs per loop
```

Torch.nn : a neural networks library

Containers	Module, Sequential,ModuleList,ParameterList
Convolution Layers	Conv1d,Conv2d,Conv3d…
Recurrent Layers	RNN,LSTM,GRU,RNNCell…
Linear Layers	Linear, Bilinear
Non-linear Activations	ReLU,Sigmoid,Tanh,LeakyReLU…
Loss Functions	NLLLoss,BCELoss,CrossEntropyLoss…

Torch.nn.functional

Torch.nn

```
In [1]: import torch
        import torch.nn as nn
        from torch.autograd import Variable

        relu = nn.ReLU(inplace=True)

        x = Variable(torch.randn(10,128))
        x = relu(x)
```

Torch.nn.functional

```
In [1]: import torch
        import torch.nn as nn
        import torch.nn.functional as F
        from torch.autograd import Variable

        x = Variable(torch.randn(10,128))
        x = F.relu(x)
```

How to load dataset?

In [7]:
```
class MyData(Dataset):
    def __init__(self):
        super(MyData,self).__init__()
        ex1 = {'sent':'Nice to meet you','label':1}
        ex2 = {'sent':'I hate you','label':0}
        self.examples = [ex1,ex2]
    def __len__(self):
        return len(self.examples)
    def __getitem__(self,index):
        return self.examples[index]
```

In [8]:
```
dataset = MyData()
data_iter = DataLoader(dataset=dataset,batch_size=2,shuffle=True)
for batch in data_iter:
    print batch
```

```
{'sent': ['I hate you', 'Nice to meet you'], 'label':
 0
 1
[torch.LongTensor of size 2]
}
```

How to build a model?

In [9]:
```
import torch
import torch.nn as nn
```

In [10]:
```
class Net(nn.Module):
    def __init__(self):
        super(Net,self).__init__()
        self.fc = nn.Linear(10,2)
    def forward(self,x):
        return self.fc(x)
```

In [11]:
```
net = Net()
x = torch.autograd.Variable(torch.randn(2,10))
print net(x)
```

```
Variable containing:
-0.4123 -0.6326
 0.7484 -0.2521
[torch.FloatTensor of size 2x2]
```

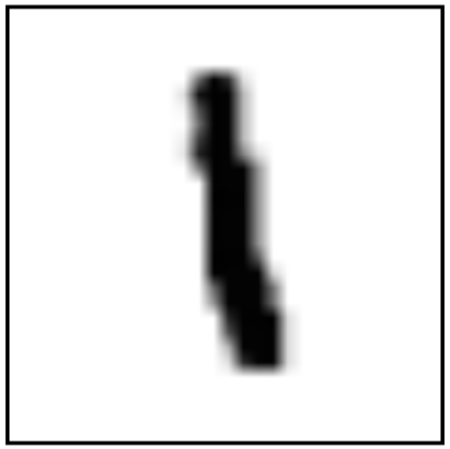
How to train a model?

In [12]:
```
net = Net()
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(net.parameters(),lr=1e-3)

for epoch in range(10):
    '''
    for x,y in data_loader:
        x,y = Variable(x),Variable(y)
        x = net(x)
        loss = criterion(x,y)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        ...

    '''
```

Outline

- ☐ Why Deep Learning Libraries and Why PyTorch ?
- ☐ A Brief Introduction to PyTorch
- ☐ **PyTorch in action**

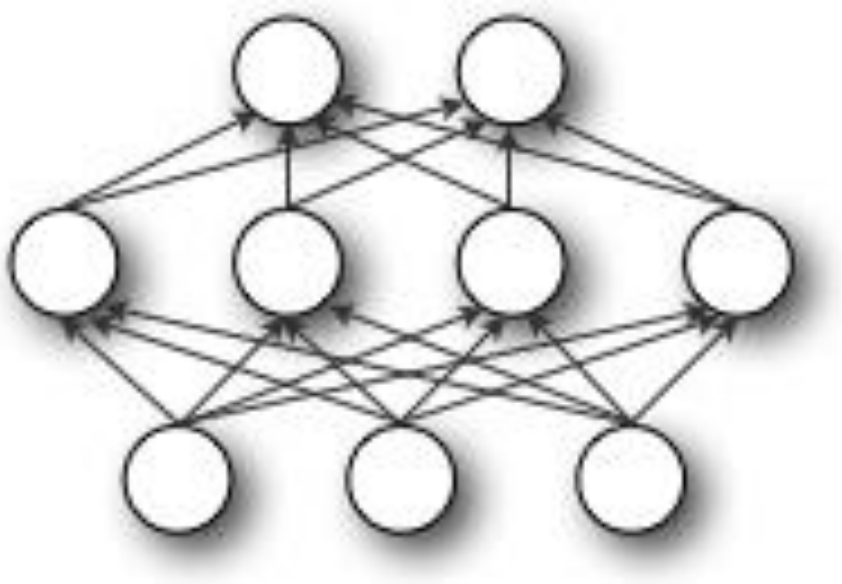
MNIST Dataset

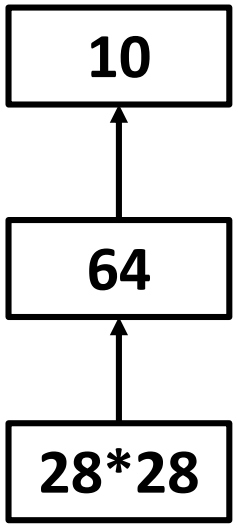


$\approx$

$$\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .6 & .8 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .7 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .7 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .5 & 1 & .4 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & .4 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & .4 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & .7 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & .9 & 1 & .1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & .3 & 1 & .1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

MLP for MNIST (0-d features)





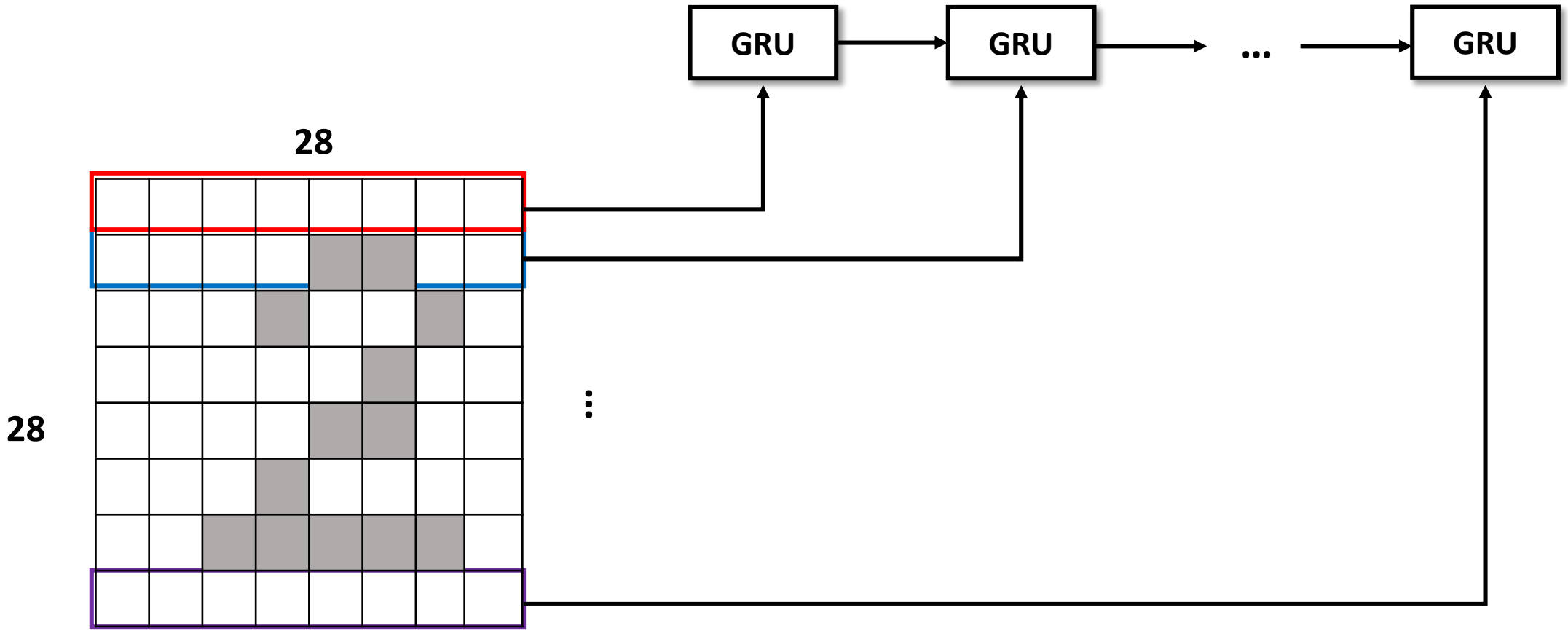
MLP for MNIST

In [3]:

```
class MLP(nn.Module):
    def __init__(self,n_class=10):
        super(MLP, self).__init__()

        self.fc = nn.Sequential(
            nn.Linear(28*28,64),
            nn.ReLU(inplace=True),
            nn.Linear(64,n_class)
        )

        """
        self.fc1 = nn.Linear(28*28,64)
        self.relu = nn.ReLU(inplace=True)
        slef.fc2 = nn.Linear(64,n_class)
        """

    def forward(self, x):
        x = x.view(-1,28*28)      # x:(batch_size,1,28,28) => x:(batch_size,28*28)
        logits = self.fc(x)
        return logits
```

Last Test Acc: 95.8%

RNN for MNIST (1-d features)

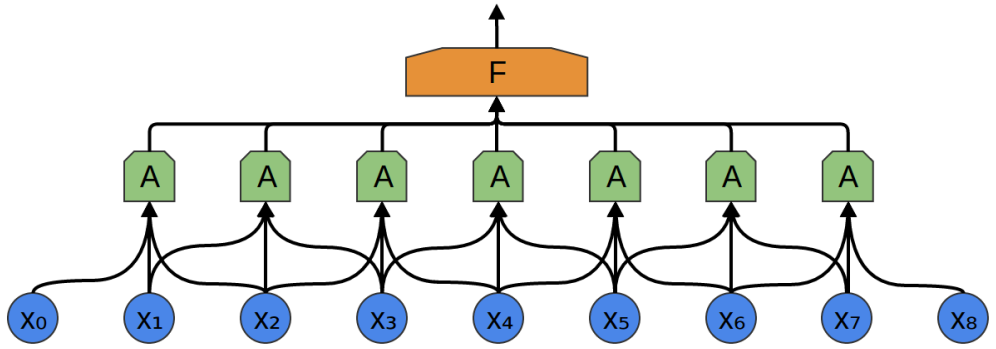


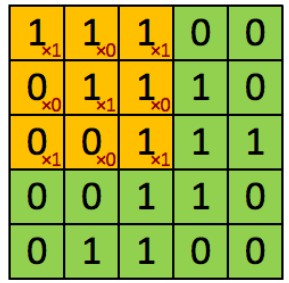
RNN for MNIST

```python
class RNN(nn.Module):
    def __init__(self,input_size=28,hidden_size=64,n_class=10):
        super(RNN, self).__init__()
        self.RNN = nn.GRU(
            input_size = input_size,
            hidden_size = hidden_size,
            batch_first = True
        )
        self.fc = nn.Linear(hidden_size,10)

    def forward(self, x):
        x = x.squeeze()        # x:(batch_size,1,28,28) => x:(batch_size,28,28)
        out, _ = self.RNN(x)   # x:(batch_size,28,28) => out:(batch_size,28,hidden_size)
        # get last hidden
        out = out[:, -1, :]    # out:(batch_size,hidden_size)
        logits = self.fc(out)
        return logits
```

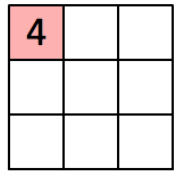
Last Test Acc: 97.7%

CNN for MNIST (2-d features)

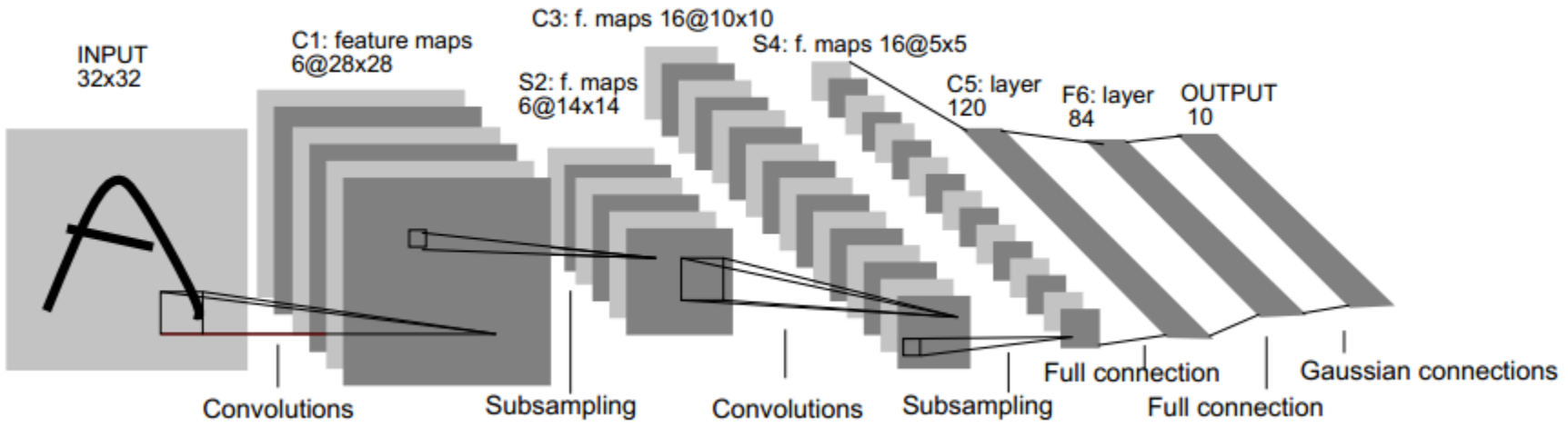




Image



Convolved Feature



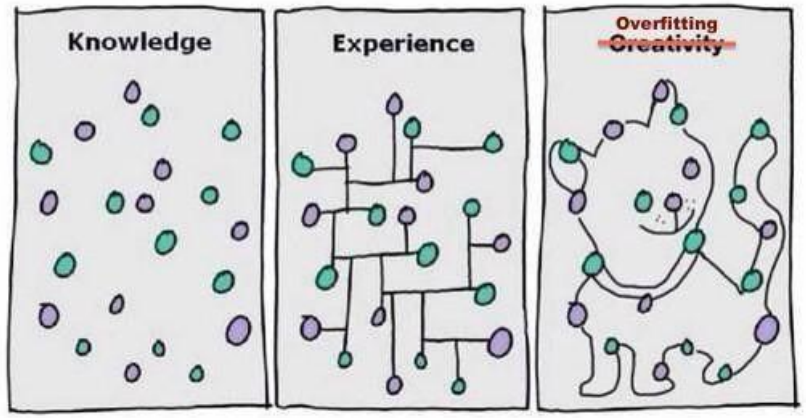
CNN for MNIST

```python
class LeNet(nn.Module):
    def __init__(self,n_class=10):
        super(LeNet, self).__init__()
        self.conv1 = nn.Conv2d(
            in_channels = 1,
            out_channels = 20,
            kernel_size = 5
        )
        self.conv2 = nn.Conv2d(
            in_channels = 20,
            out_channels = 50,
            kernel_size = 5
        )
        self.fc1 = nn.Linear(4*4*50, 500)
        self.fc2 = nn.Linear(500, n_class)
    def forward(self, x):
        x = F.relu(self.conv1(x))    # x:[batch_size,1,28,28] => x:[batch_size,20, 24, 24]
        x = F.max_pool2d(x, 2, 2)    # x:[batch_size,20,24,24] => x:[batch_size,20, 12, 12]
        x = F.relu(self.conv2(x))    # x:[batch_size,20,12,12] => x:[batch_size,50, 8, 8]
        x = F.max_pool2d(x, 2, 2)    # x:[batch_size,50,8,8] => x:[batch_size,50, 4, 4]
        x = x.view(-1, 4*4*50)       # x:[batch_size,50,4,4] => x:[batch_size,50*4*4]
        x = F.relu(self.fc1(x))      # x:[batch_size,50*4*4] => x:[batch_size,500]
        x = self.fc2(x)              # x:[batch_size,500] => x:[batch_size,10]
        return x
```

Last Test Acc: 99.2%

Assignments

1. Try more models (Bi-LSTM,ResNet…)
2. L2 Regularization
3. Dropout
4. Data Argumentation
5. Batch Normalization
6. Try other optimization algorithms(SGD…)
7. Try more activation function(Tanh,Sigmoid…)



Resources

- Official resources
  - Documentation http://pytorch.org/docs/master/
  - Tutorials http://pytorch.org/tutorials/index.html
  - Example projects https://github.com/pytorch/examples
- Github code
  - fairseq-py
  - OpenNMT-py
- Open courses and blogs
  - **Stanford CS231N 2017**, Lecture 8 “Deep Learning Software”