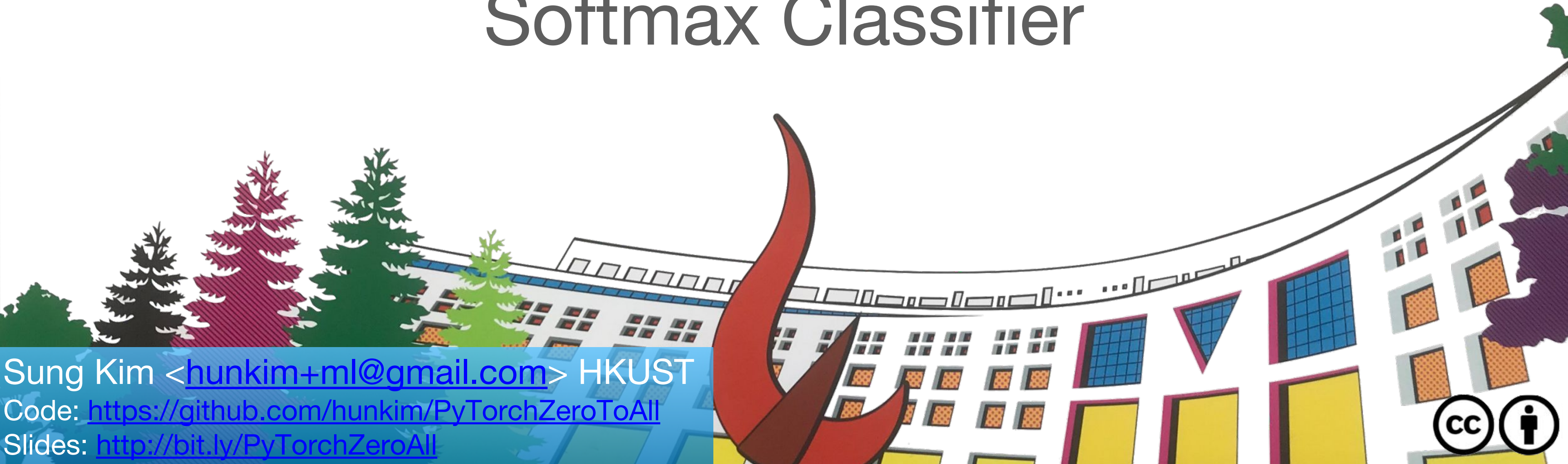
ML/DL for Everyone with PYTORCH

Lecture 9:
Softmax Classifier



Sung Kim <hunkim+ml@gmail.com> HKUST
Code: https://github.com/hunkim/PyTorchZeroToAll
Slides: http://bit.ly/PyTorchZeroAll

Call for Comments

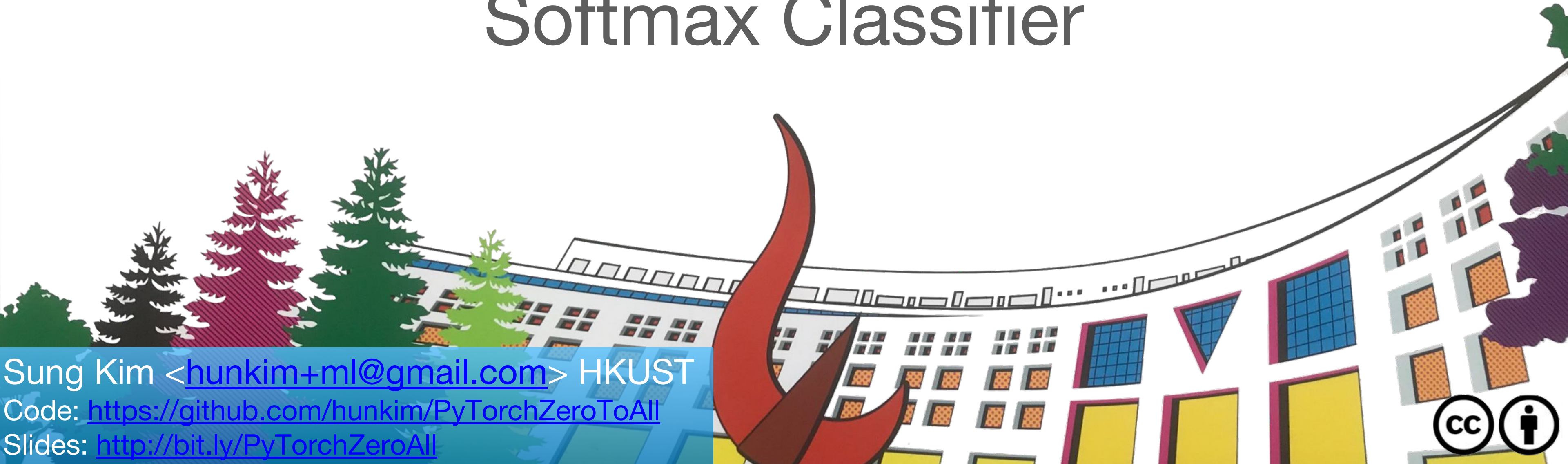
Please feel free to add comments directly on these slides.

Other slides: http://bit.ly/PyTorchZeroAll

Picture from http://www.tssablog.org/archives/3280

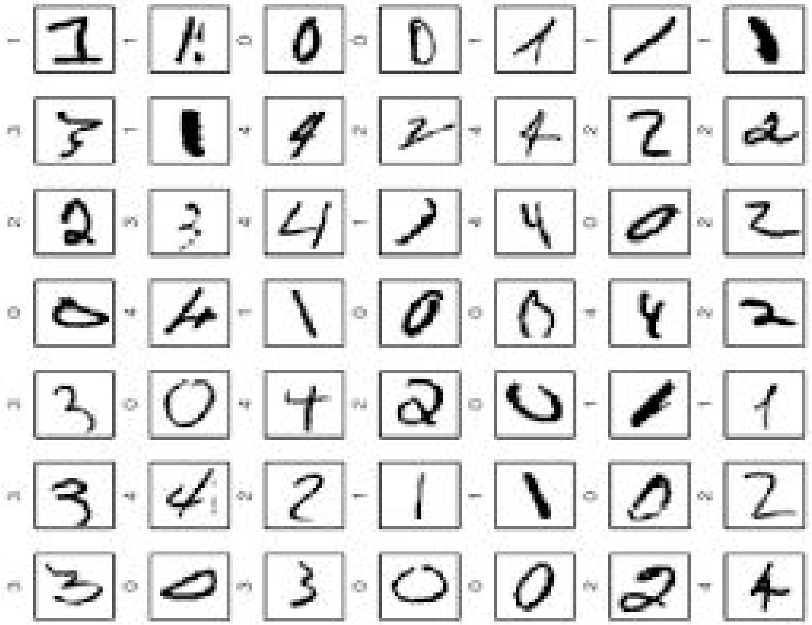
ML/DL for Everyone with PYTORCH

Lecture 9:
Softmax Classifier

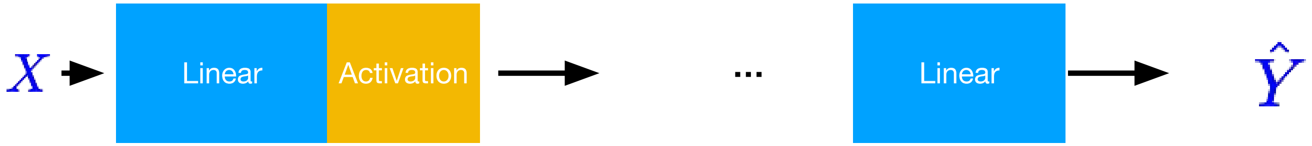


Sung Kim <hunkim+ml@gmail.com> HKUST
Code: https://github.com/hunkim/PyTorchZeroToAll
Slides: http://bit.ly/PyTorchZeroAll

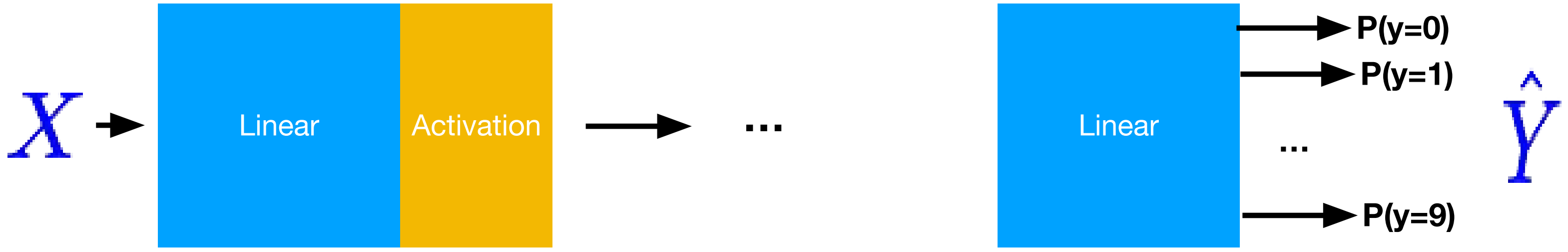
MNIST: 10 labels



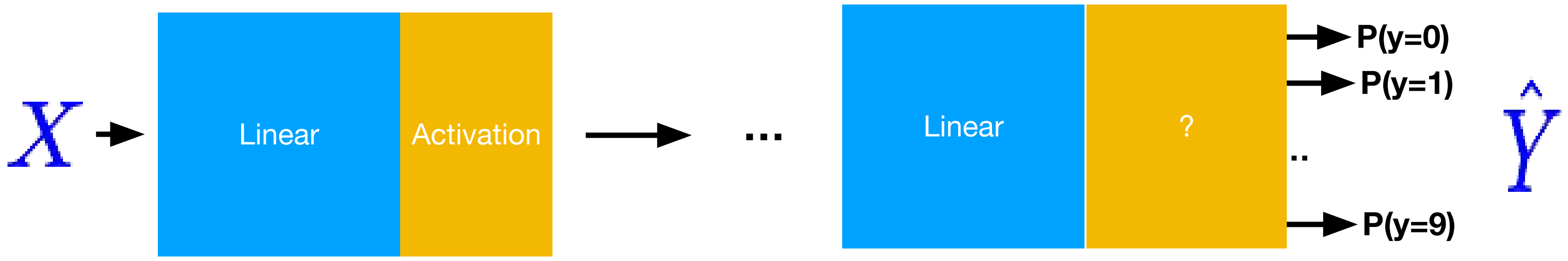
10 labels: 10 outputs



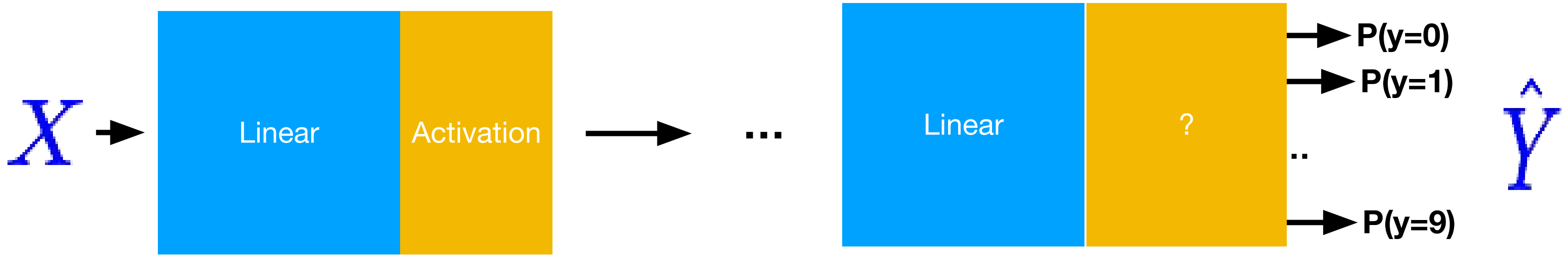
10 labels: 10 outputs



10 outputs

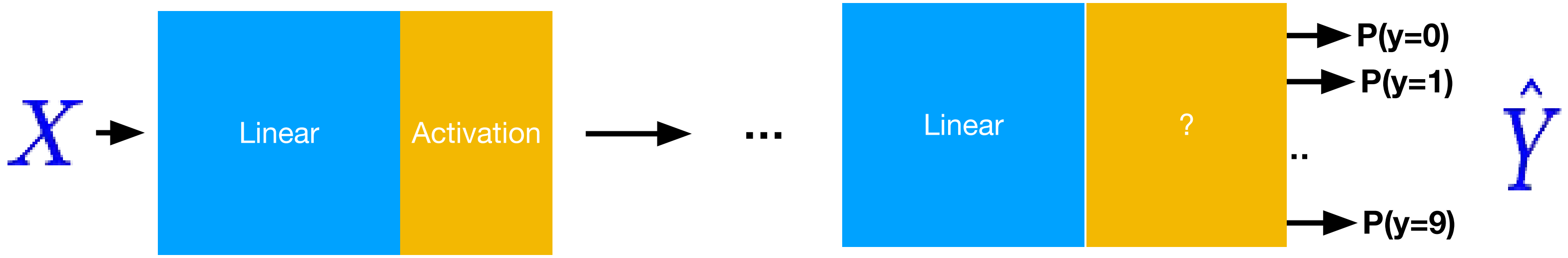


10 outputs



$$\underbrace{\begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \\ \cdots & \cdots \\ a_n & b_n \end{bmatrix}}_{x\in\mathbb{R}^{N\times 2}} \underbrace{\begin{bmatrix} w_1 \\ w_2 \end{bmatrix}}_{w\in\mathbb{R}^{2\times 1}} = \underbrace{\begin{bmatrix} y_1 \\ y_2 \\ \cdot\,\cdot \\ y_n \end{bmatrix}}_{y\in\mathbb{R}^{N\times 1}}$$

10 outputs

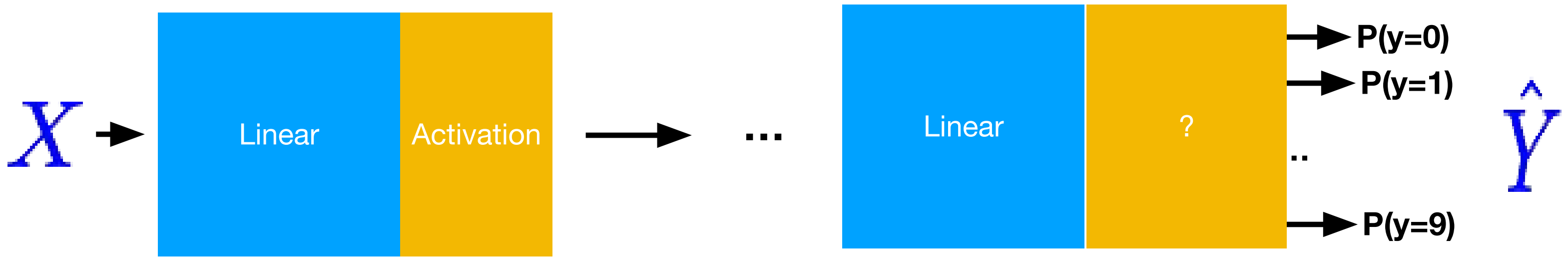


$$\underbrace{\begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \\ \cdots & \cdots \\ a_n & b_n \end{bmatrix}}_{x \in \mathbb{R}^{N\times 2}} \underbrace{\begin{bmatrix} w_1 \\ w_2 \end{bmatrix}}_{w \in \mathbb{R}^{2\times 1}} = \underbrace{\begin{bmatrix} y_1 \\ y_2 \\ \cdot\cdot \\ y_n \end{bmatrix}}_{y \in \mathbb{R}^{N\times 1}}$$

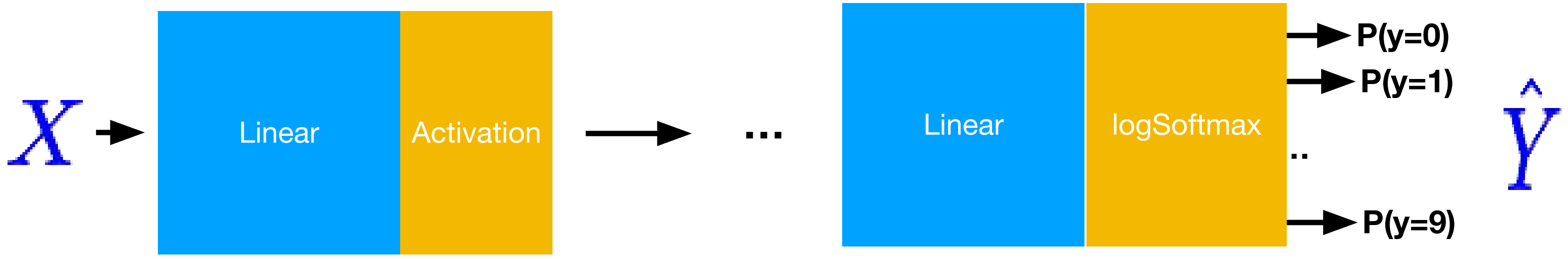
$$\underbrace{\begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \\ \cdots & \cdots \\ a_n & b_n \end{bmatrix}}_{x \in \mathbb{R}^{N\times 2}} \begin{bmatrix} ? \end{bmatrix} = y \in R^{N\times 10}$$

$$w \in R^{N\times ?}$$

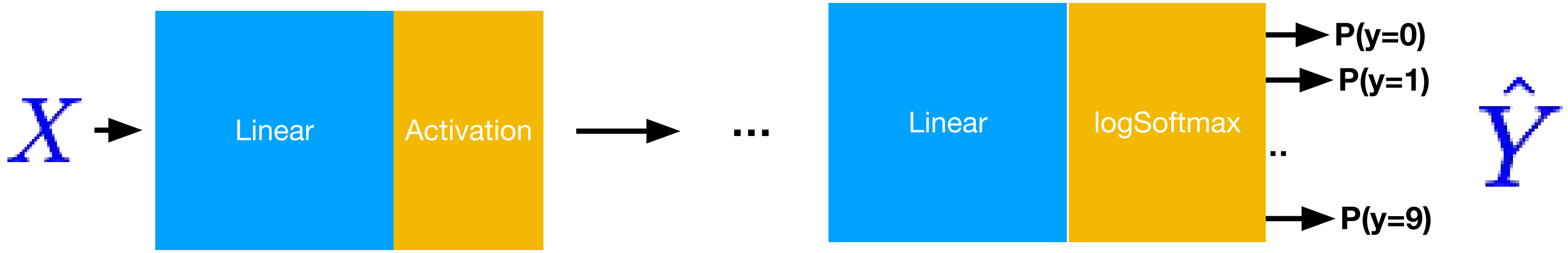
Probability

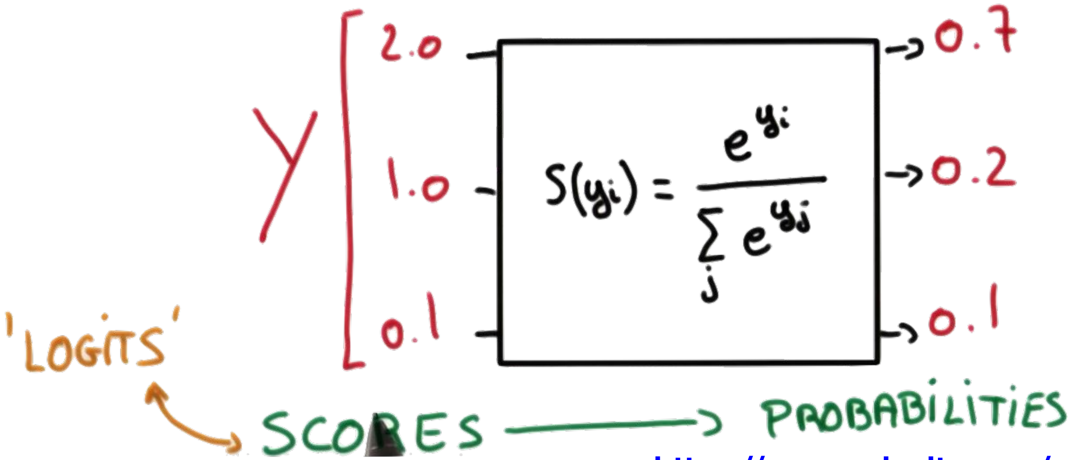


(log)Softmax



Softmax





https://www.udacity.com/course/deep-learning--ud730

LogSoftmax with NLLLoss

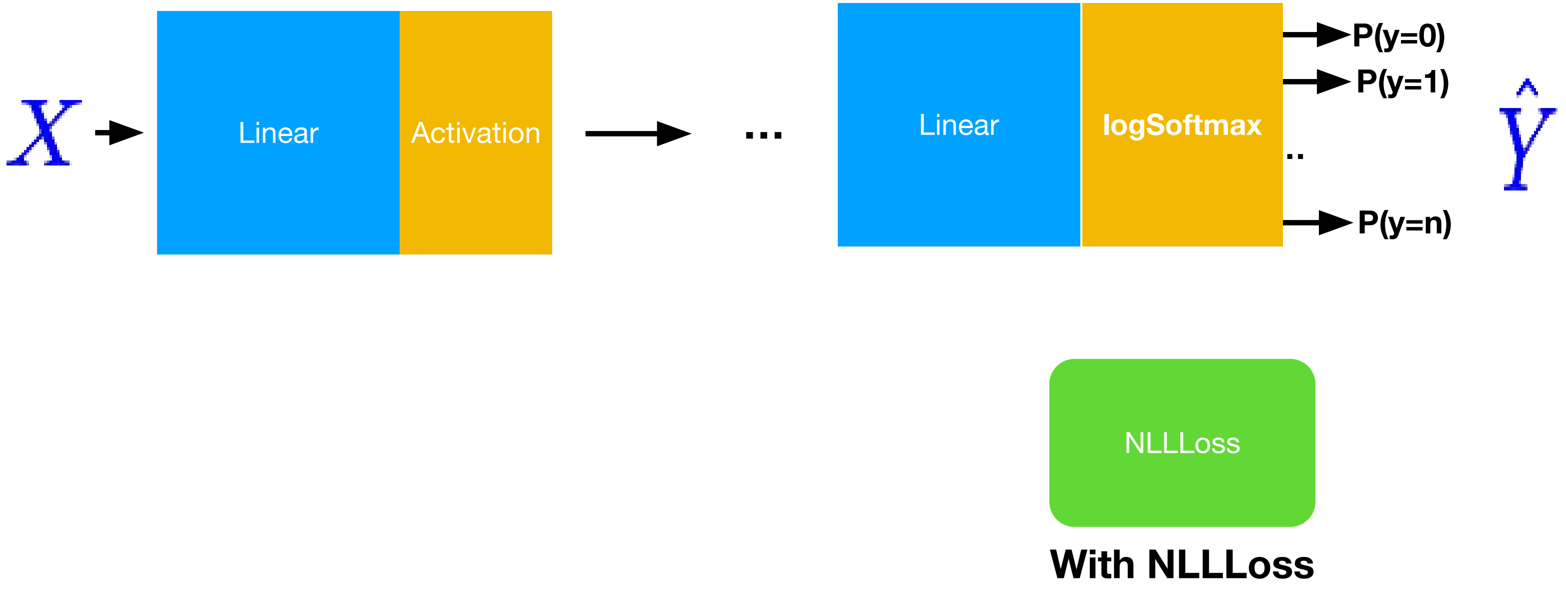
```
# http://pytorch.org/docs/master/nn.html#nllloss
logsm = nn.LogSoftmax()
loss = nn.NLLLoss()

# input is of size nBatch x nClasses = 3 x 5
input = Variable(torch.randn(3, 5), requires_grad=True)
logsm_out = logsm(input)

# target is of size nBatch
# each element in target has to have 0 <= value < nclasses
target = Variable(torch.LongTensor([1, 0, 4]))

l = loss(logsm_out, target)
l.backward()

print(input.size(), target.size(), l.size())
```

http://pytorch.org/docs/master/nn.html#nllloss

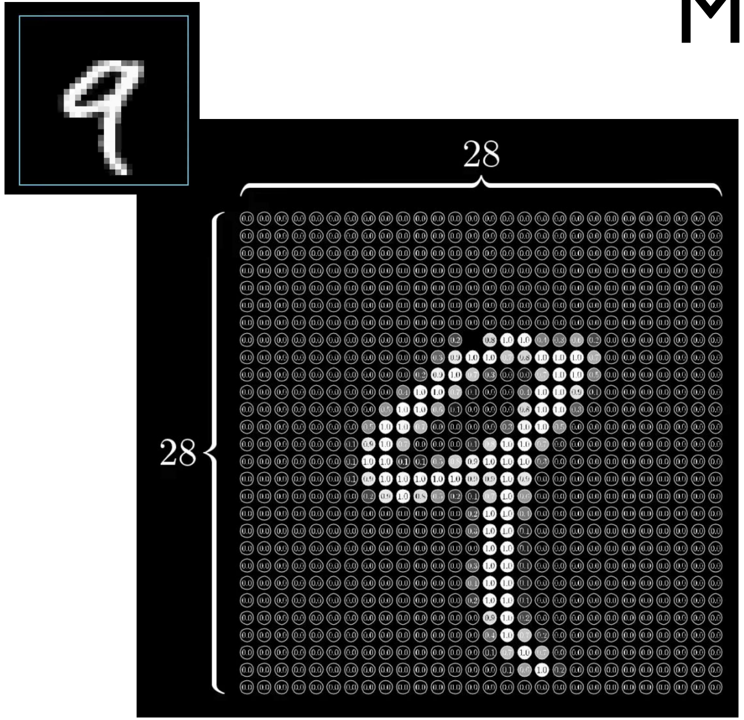
Exercise 9-1: CorssEntropyLoss VS NLLLoss

- What are the differences?
- Check out
  - http://pytorch.org/docs/master/nn.html#nllloss
  - http://pytorch.org/docs/master/nn.html#crossentropyloss
- Minimizing the Negative Log-Likelihood, in English http://willwolf.io/2017/05/18/minimizing_the_negative_log_likelihood_in_english/

(log)Softmax + NLLLoss



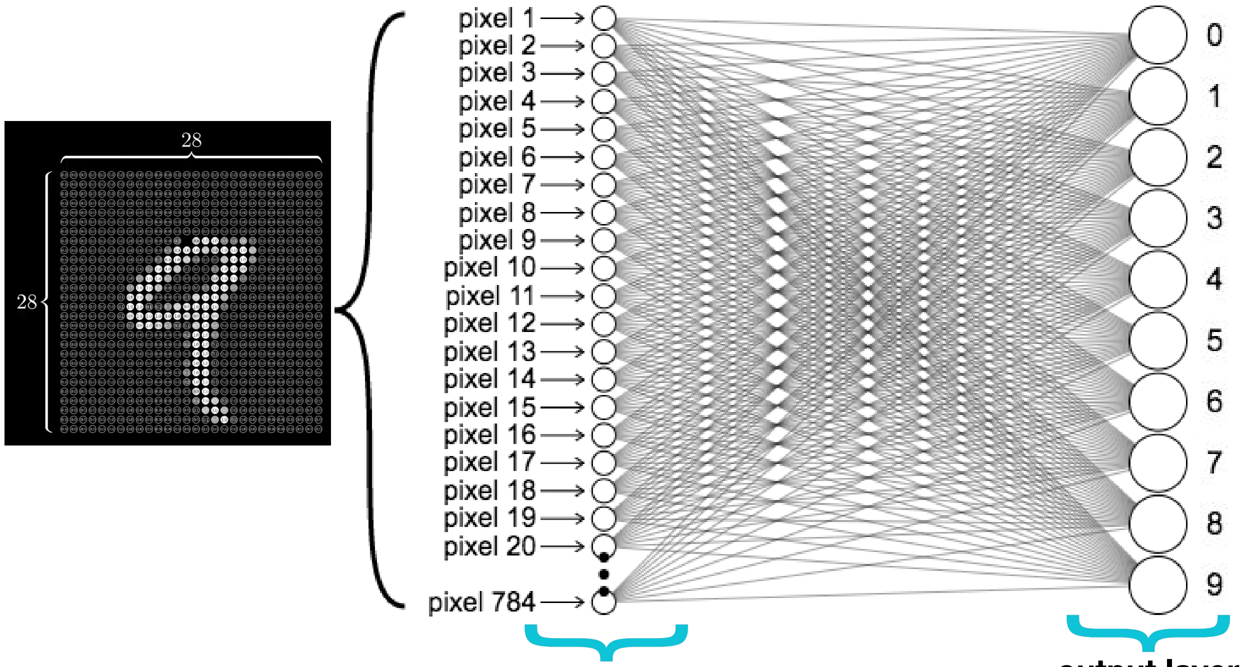
MNIST input



28x28 pixels = 748

https://www.youtube.com/watch?v=aircAruvnKk&t=1s

MNIST Network

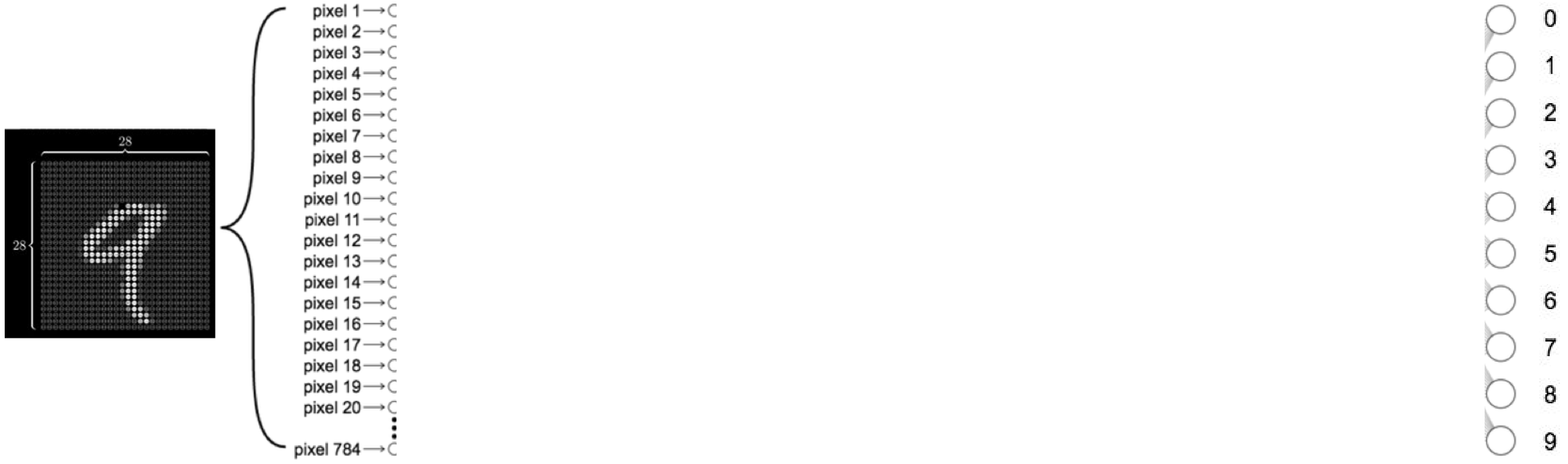


Input layer 784

output layer 10 (labels)

https://ml4a.github.io/ml4a/looking_inside_neural_nets

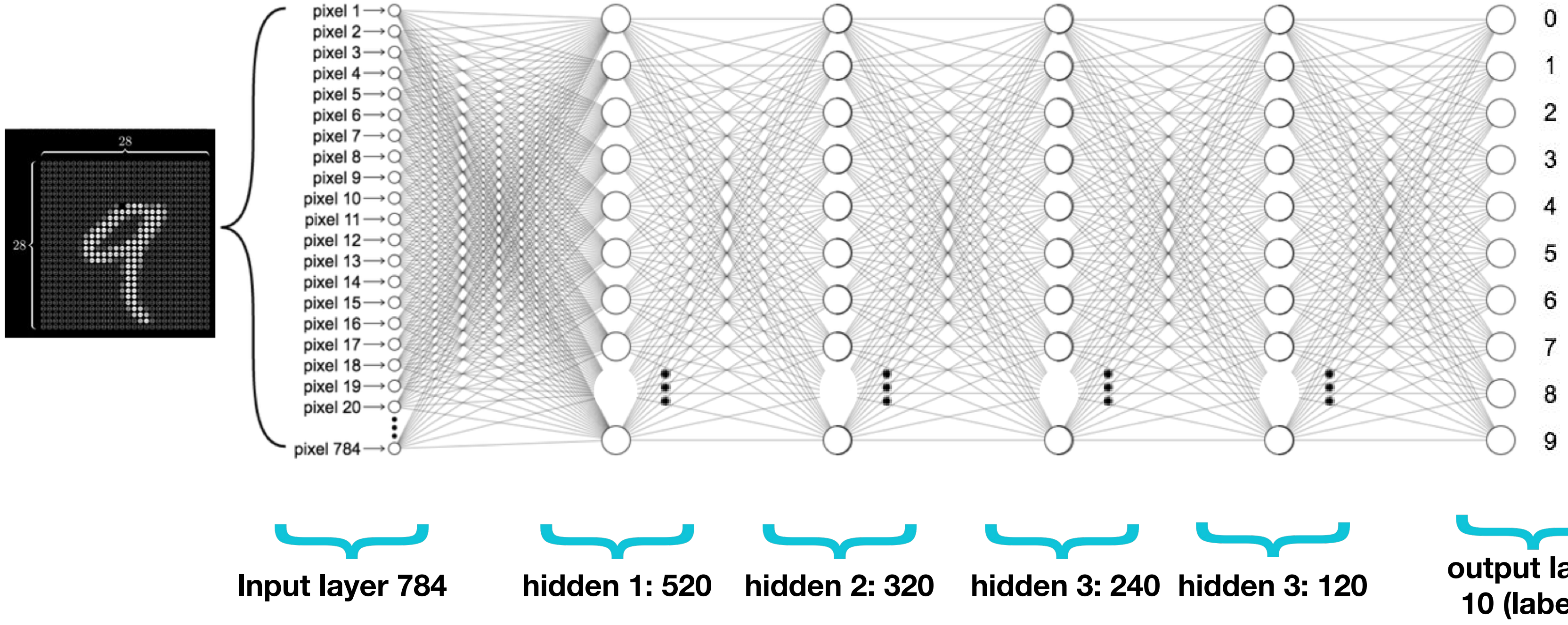
MNIST Network



$\underbrace{\hspace{3em}}_{\textbf{Input layer 784}}$ $\underbrace{\hspace{6em}}_{\textbf{Hidden layers}}$ $\underbrace{\hspace{3em}}_{\textbf{output layer 10 (labels)}}$

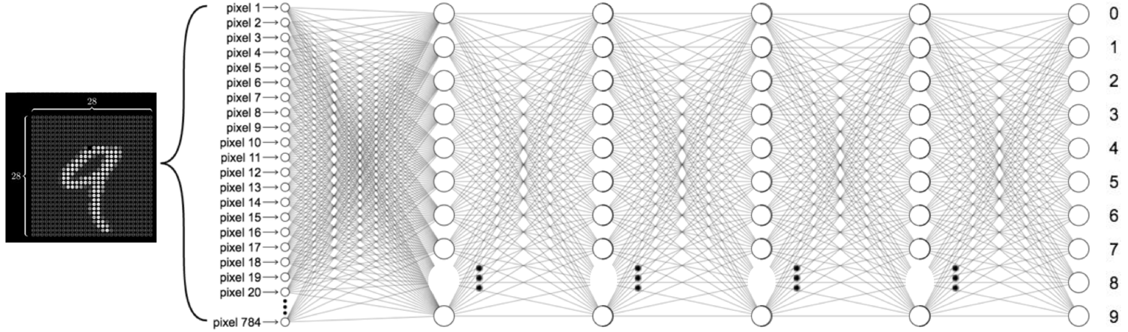
https://ml4a.github.io/ml4a/looking_inside_neural_nets

MNIST Network



https://ml4a.github.io/ml4a/looking_inside_neural_nets

MNIST Network

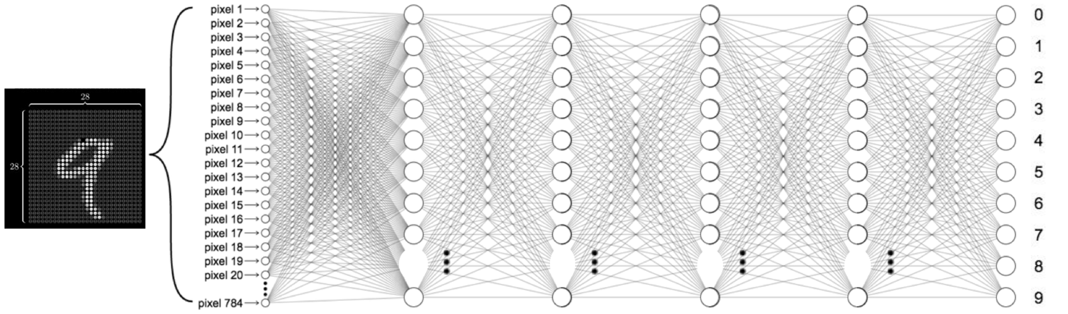


```
self.l1 = nn.Linear(784, 520)
        self.l2 = nn.Linear(520, 320)
               self.l3 = nn.Linear(320, 240)
                      self.l4 = nn.Linear(240, 120)
                             self.l5 = nn.Linear(120, 10)
```

https://ml4a.github.io/ml4a/looking_inside_neural_nets

Softmax & NLL loss

```python
class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        self.l1 = nn.Linear(784, 520)
        self.l2 = nn.Linear(520, 320)
        self.l3 = nn.Linear(320, 240)
        self.l4 = nn.Linear(240, 120)
        self.l5 = nn.Linear(120, 10)

    def forward(self, x):
        # Flatten the data (n, 1, 28, 28)-> (n, 784)
        x = x.view(-1, 784)
        x = F.relu(self.l1(x))
        x = F.relu(self.l2(x))
        x = F.relu(self.l3(x))
        x = F.relu(self.l4(x))
        x = F.relu(self.l5(x))
        return F.log_softmax(x)
```



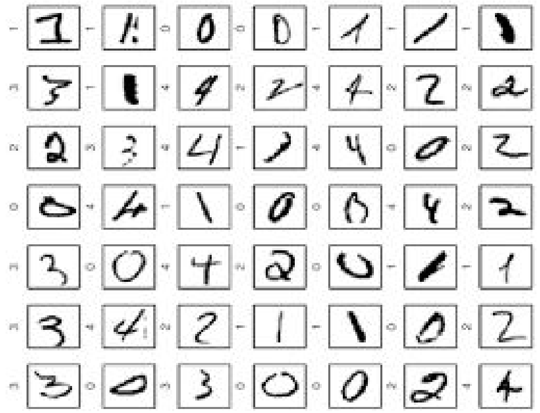
Softmax & NLL loss

```python
class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        self.l1 = nn.Linear(784, 520)
        self.l2 = nn.Linear(520, 320)
        self.l3 = nn.Linear(320, 240)
        self.l4 = nn.Linear(240, 120)
        self.l5 = nn.Linear(120, 10)

    def forward(self, x):
        # Flatten the data (n, 1, 28, 28)-> (n, 784)
        x = x.view(-1, 784)
        x = F.relu(self.l1(x))
        x = F.relu(self.l2(x))
        x = F.relu(self.l3(x))
        x = F.relu(self.l4(x))
        x = F.relu(self.l5(x))
        return F.log_softmax(x)
```

```python
for batch_idx, (data, target) in enumerate(train_loader):
    data, target = Variable(data), Variable(target)
    optimizer.zero_grad()
    output = model(data)
    loss = F.nll_loss(output, target)
    loss.backward()
    optimizer.step()
```

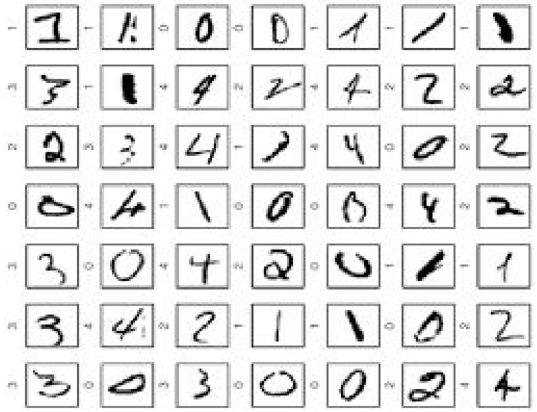
MNIST Softmax

```python
# Training settings
batch_size = 64
train_loader = torch.utils.data.DataLoader(
    datasets.MNIST('../data', train=True, download=True, transform=transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.1307,), (0.3081,))])),
    batch_size=batch_size, shuffle=True)
test_loader = torch.utils.data.DataLoader(
    datasets.MNIST('../data', train=False, transform=transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.1307,), (0.3081,))])),
    batch_size=batch_size, shuffle=True)

class Net(nn.Module):
   def __init__(self):
        super(Net, self).__init__()
        self.l1 = nn.Linear(784, 520)
        self.l2 = nn.Linear(520, 320)
        self.l3 = nn.Linear(320, 240)
        self.l4 = nn.Linear(240, 120)
        self.l5 = nn.Linear(120, 10)

    def forward(self, x):
        x = x.view(-1, 784)  # Flatten the data (n, 1, 28, 28)-> (n, 784)
        x = F.relu(self.l1(x))
        x = F.relu(self.l2(x))
        x = F.relu(self.l3(x))
        x = F.relu(self.l4(x))
        x = F.relu(self.l5(x))
        return F.log_softmax(x)

model = Net()
optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.5)

def train(epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = Variable(data), Variable(target)
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()
        if batch_idx % 10 == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.data[0]))
```



```python
# Training settings
batch_size = 64
train_loader = torch.utils.data.DataLoader(
    datasets.MNIST('../data', train=True, download=True, transform=transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.1307,), (0.3081,))])),
    batch_size=batch_size, shuffle=True)
test_loader = torch.utils.data.DataLoader(
    datasets.MNIST('../data', train=False, transform=transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.1307,), (0.3081,))])),
    batch_size=batch_size, shuffle=True)

class Net(nn.Module):
   def __init__(self):
        super(Net, self).__init__()
        self.l1 = nn.Linear(784, 520)
        self.l2 = nn.Linear(520, 320)
        self.l3 = nn.Linear(320, 240)
        self.l4 = nn.Linear(240, 120)
        self.l5 = nn.Linear(120, 10)

    def forward(self, x):
        x = x.view(-1, 784)  # Flatten the data (n, 1, 28, 28)-> (n, 784)
        x = F.relu(self.l1(x))
        x = F.relu(self.l2(x))
        x = F.relu(self.l3(x))
        x = F.relu(self.l4(x))
        x = F.relu(self.l5(x))
        return F.log_softmax(x)

model = Net()
optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.5)

def train(epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = Variable(data), Variable(target)
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()
        if batch_idx % 10 == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.data[0]))
```

Accuracy?



```
train_loader = torch.utils.data.DataLoader(
    datasets.MNIST('../data', train=True, download=True, transform=transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.1307,), (0.3081,))
    ])),
    batch_size=batch_size, shuffle=True)
test_loader = torch.utils.data.DataLoader(
    datasets.MNIST('../data', train=False, transform=transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize((0.1307,), (0.3081,))
    ])),
    batch_size=batch_size, shuffle=True)
```

```
def train(epoch):
    ...
```

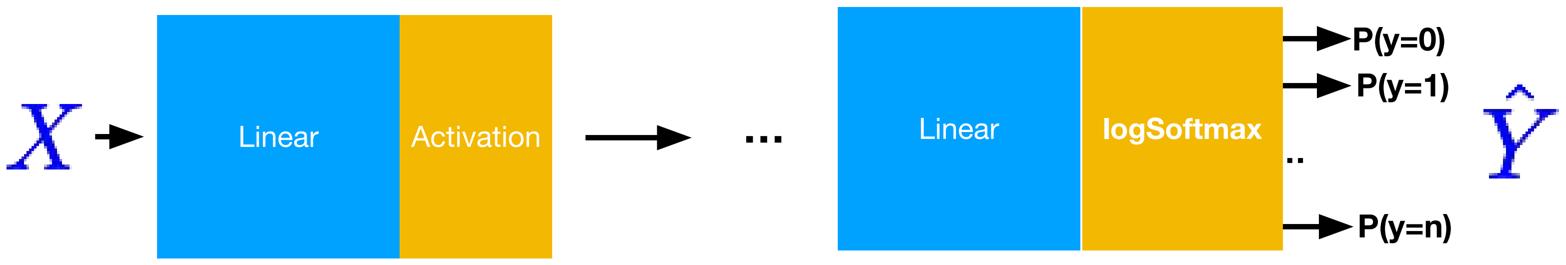
```
def test():
    model.eval()
    test_loss = 0
    correct = 0
    for data, target in test_loader:
        data, target = Variable(data, volatile=True), Variable(target)
        output = model(data)
        # sum up batch loss
        test_loss += F.nll_loss(output, target, size_average=False).data[0]
        # get the index of the max log-probability
        pred = output.data.max(1, keepdim=True)[1]
        correct += pred.eq(target.data.view_as(pred)).cpu().sum()

    test_loss /= len(test_loader.dataset)
    print('\nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(
        test_loss, correct, len(test_loader.dataset),
        100. * correct / len(test_loader.dataset)))
```

```python
def train(epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = Variable(data), Variable(target)
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()
        if batch_idx % 10 == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.data[0]))


def test():
    model.eval()
    test_loss = 0
    correct = 0
    for data, target in test_loader:
        data, target = Variable(data, volatile=True), Variable(target)
        output = model(data)
        # sum up batch loss
        test_loss += F.nll_loss(output, target, size_average=False).data[0]
        # get the index of the max log-probability
        pred = output.data.max(1, keepdim=True)[1]
        correct += pred.eq(target.data.view_as(pred)).cpu().sum()

    test_loss /= len(test_loader.dataset)
    print('\nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n’.
        format(test_loss, correct, len(test_loader.dataset),
        100. * correct / len(test_loader.dataset)))

for epoch in range(1, 10):
    train(epoch)
    test()
```

Accuracy?

```
Train Epoch: 9 [46720/60000 (78%)]        Loss: 0.790513
Train Epoch: 9 [47360/60000 (79%)]        Loss: 0.335216
Train Epoch: 9 [48000/60000 (80%)]        Loss: 0.675538
Train Epoch: 9 [48640/60000 (81%)]        Loss: 0.359488
Train Epoch: 9 [49280/60000 (82%)]        Loss: 0.276906
Train Epoch: 9 [49920/60000 (83%)]        Loss: 0.412109
Train Epoch: 9 [50560/60000 (84%)]        Loss: 0.556780
Train Epoch: 9 [51200/60000 (85%)]        Loss: 0.332712
Train Epoch: 9 [51840/60000 (86%)]        Loss: 0.514475
Train Epoch: 9 [52480/60000 (87%)]        Loss: 0.515686
Train Epoch: 9 [53120/60000 (88%)]        Loss: 0.462904
Train Epoch: 9 [53760/60000 (90%)]        Loss: 0.571690
Train Epoch: 9 [54400/60000 (91%)]        Loss: 0.446774
Train Epoch: 9 [55040/60000 (92%)]        Loss: 0.441682
Train Epoch: 9 [55680/60000 (93%)]        Loss: 0.438245
Train Epoch: 9 [56320/60000 (94%)]        Loss: 0.470004
Train Epoch: 9 [56960/60000 (95%)]        Loss: 0.474394
Train Epoch: 9 [57600/60000 (96%)]        Loss: 0.527718
Train Epoch: 9 [58240/60000 (97%)]        Loss: 0.614899
Train Epoch: 9 [58880/60000 (98%)]        Loss: 0.512663
Train Epoch: 9 [59520/60000 (99%)]        Loss: 0.474054

Test set: Average loss: 0.5403, Accuracy: 7820/10000 (78%)
```

Multiple label prediction?
No problem! Use logSoftmax + NLLLoss



With NLLLoss

Exercise 9-1

- Build a classifier for Otto Group Product
  - https://www.kaggle.com/c/otto-group-product-classification-challenge/data
  - Use train.csv.zip (1.59 MB)
- Use DataLoader

WHAT NEXT

Lecture 10: CNN