

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>

Videos: <http://bit.ly/PyTorchVideo>



Call for Comments

Please feel free to add comments directly on these slides.

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



ML/DL for Everyone with PYTORCH

Lecture 9: Softmax Classifier

Sung Kim <hunkim+ml@gmail.com> HKUST

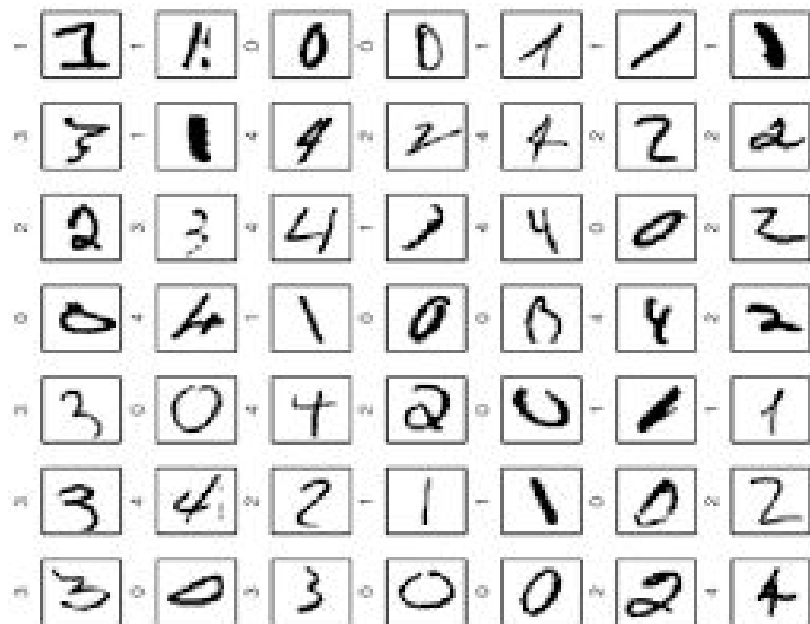
Code: <https://github.com/hunkim/PyTorchZeroToAll>

Slides: <http://bit.ly/PyTorchZeroAll>

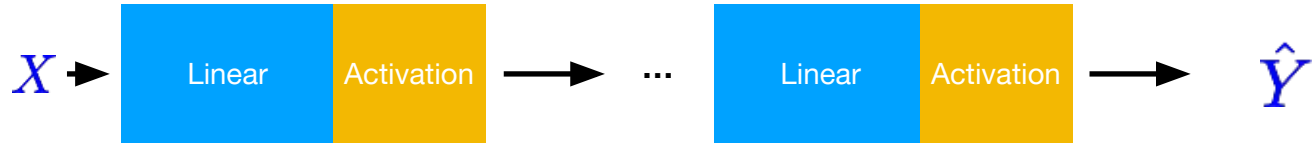
Videos: <http://bit.ly/PyTorchVideo>



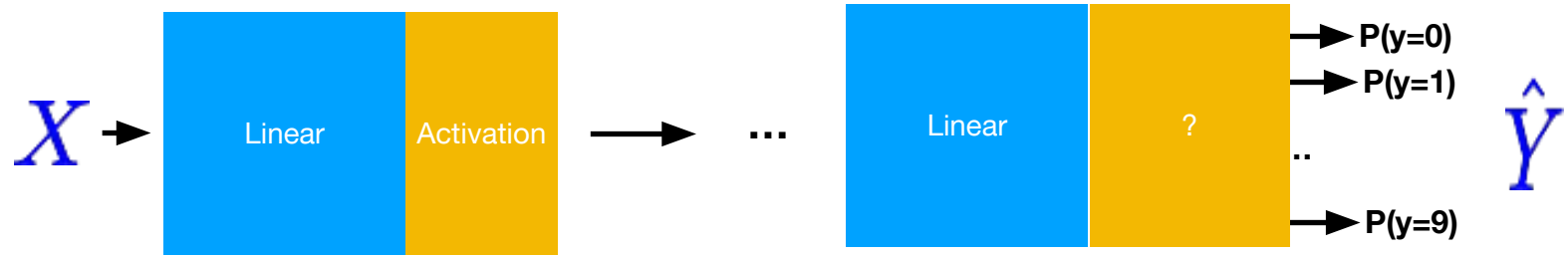
MNIST: 10 labels



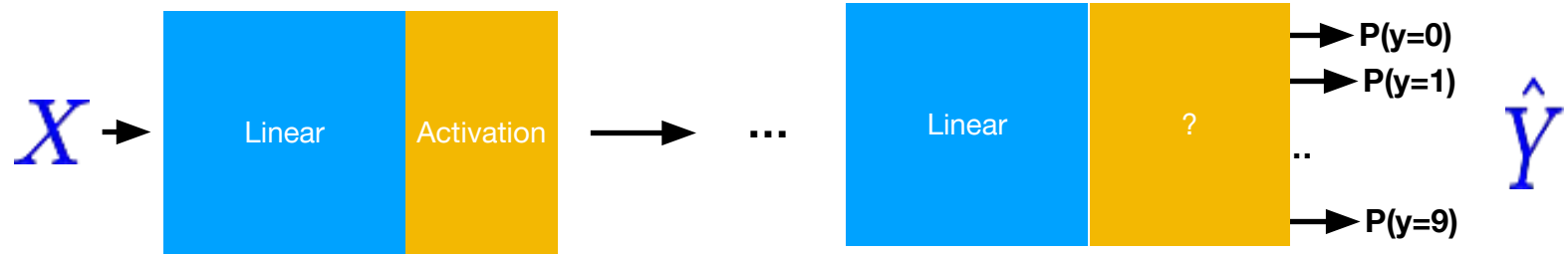
10 labels: 10 outputs



10 labels: 10 outputs

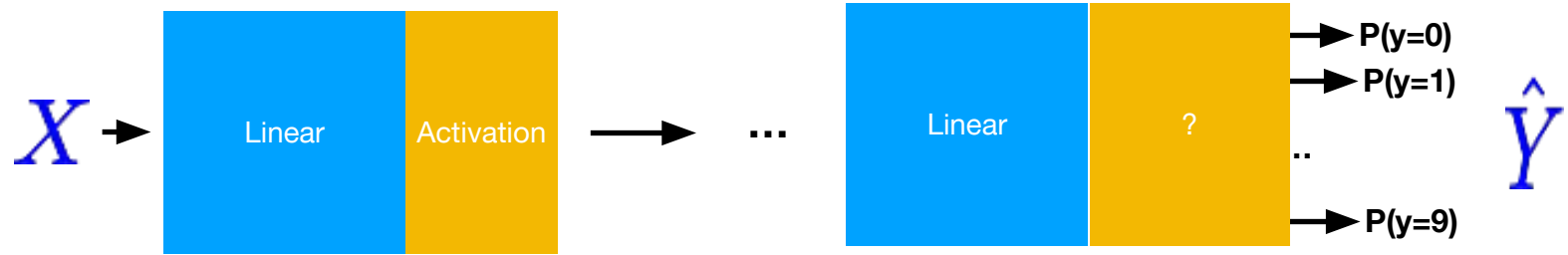


10 outputs



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

10 outputs

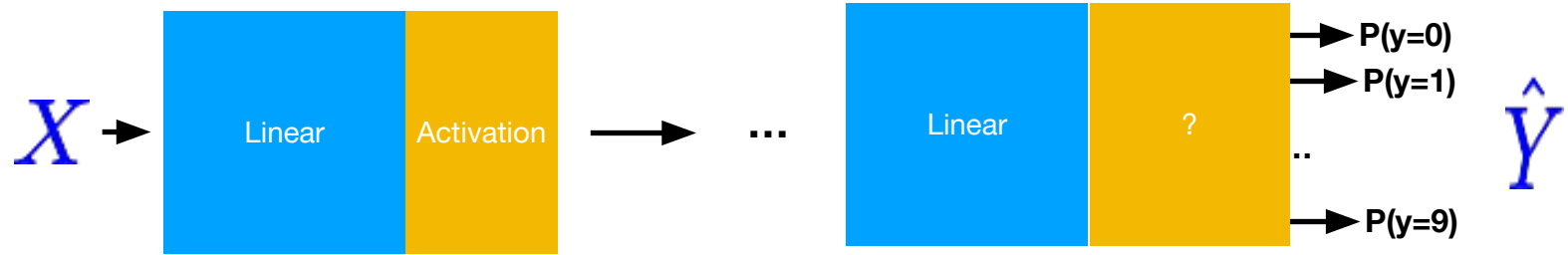


$$\underbrace{\begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \\ \dots & \dots \\ a_n & b_n \end{bmatrix}}_{x \in \mathbb{R}^{N \times 2}} \underbrace{\begin{bmatrix} w_1 \\ w_2 \end{bmatrix}}_{w \in \mathbb{R}^{2 \times 1}} = \underbrace{\begin{bmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{bmatrix}}_{y \in \mathbb{R}^{N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

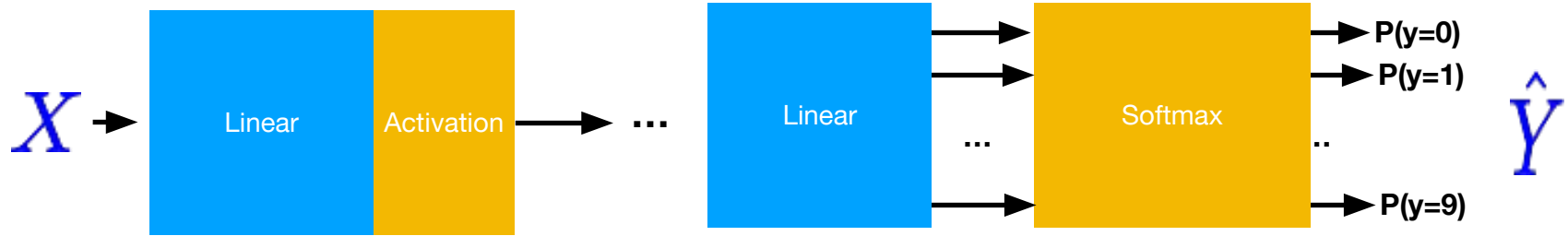
$$\underbrace{\begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \\ \dots & \dots \\ a_n & b_n \end{bmatrix}}_{x \in \mathbb{R}^{N \times 2}} \begin{bmatrix} ? \end{bmatrix} = y \in \mathbb{R}^{N \times 10}$$

$w \in \mathbb{R}^{2 \times ?}$

Probability

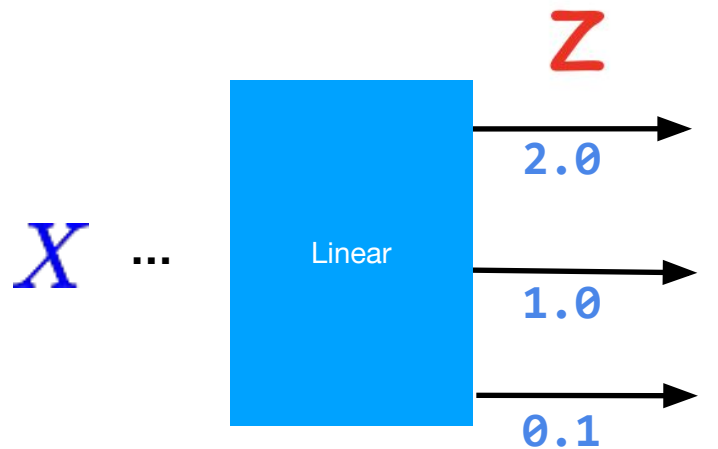


Softmax



Meet Softmax

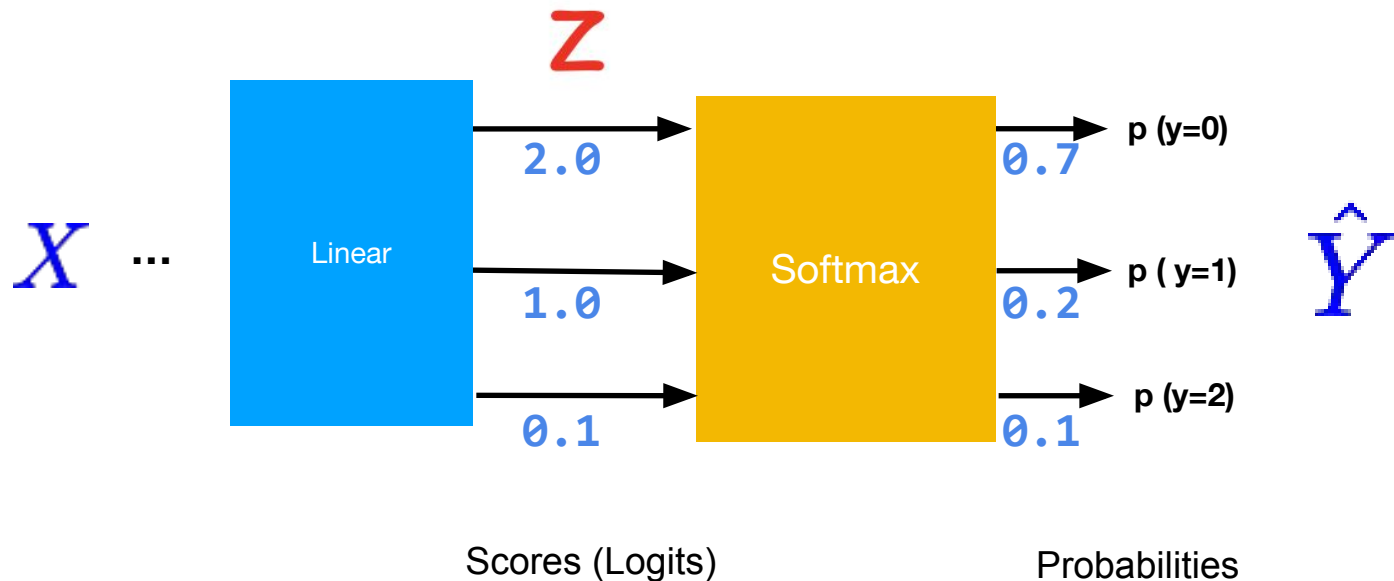
$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}} \quad \text{for } j = 1, \dots, K.$$

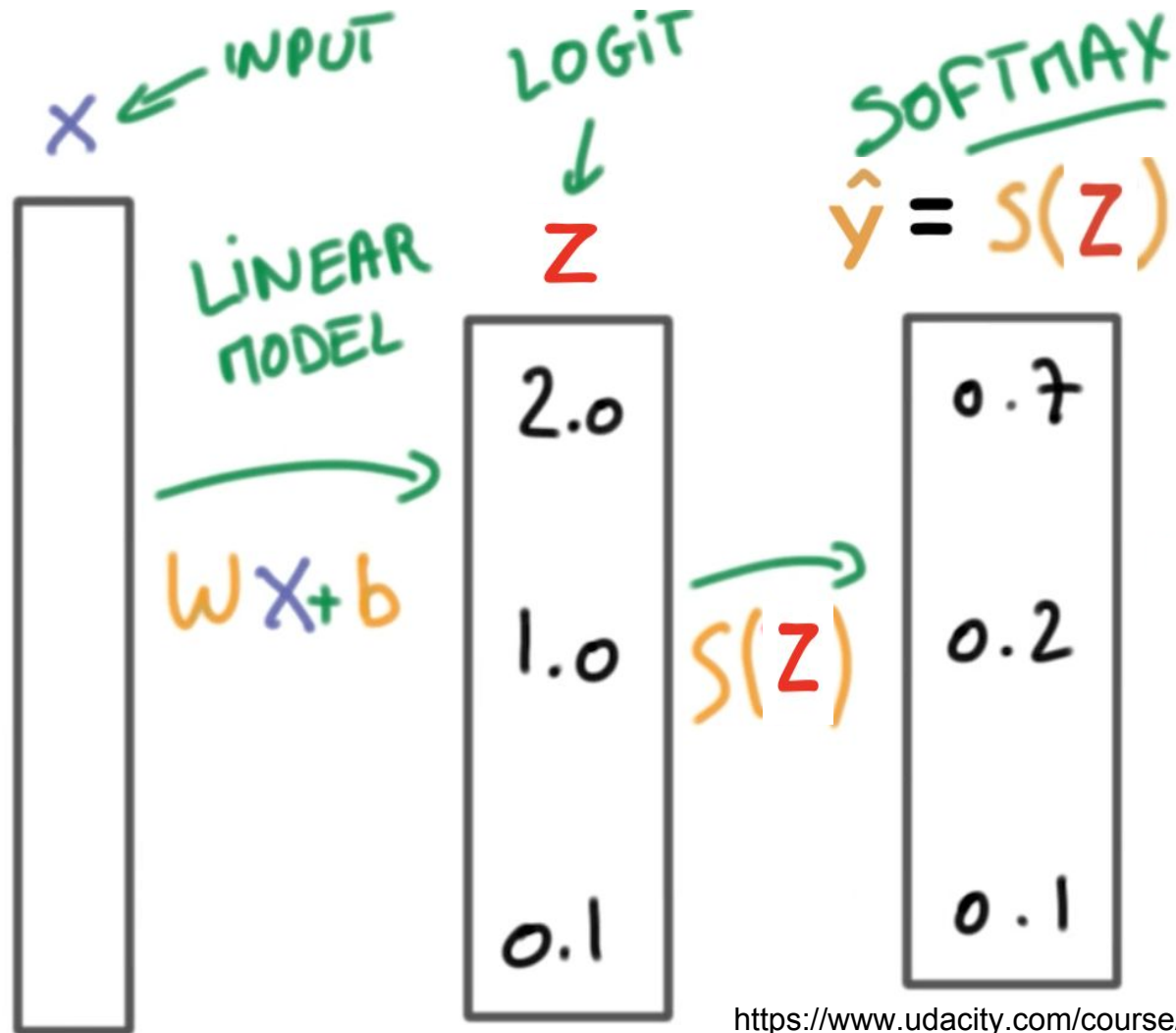


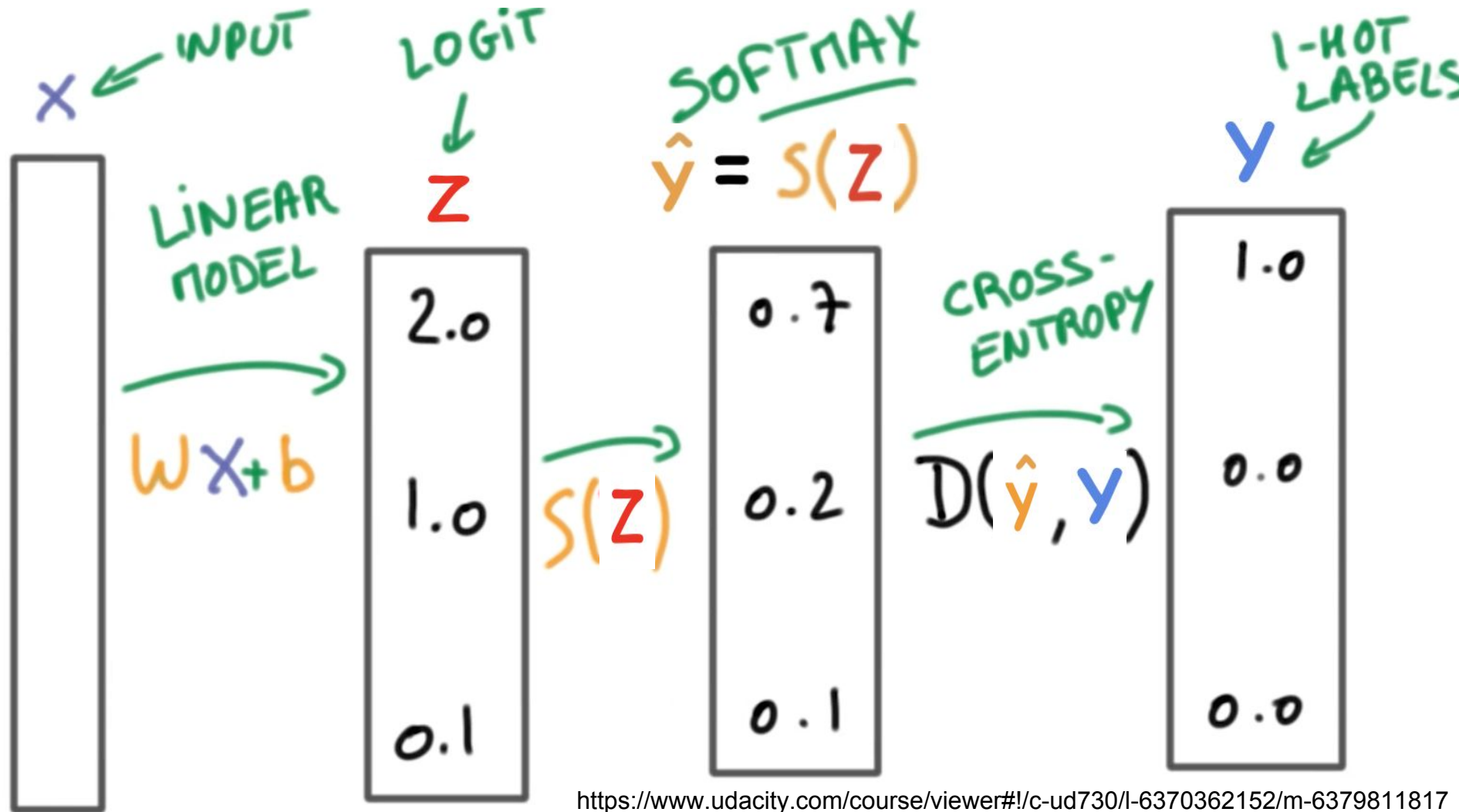
Scores (Logits)

Meet Softmax

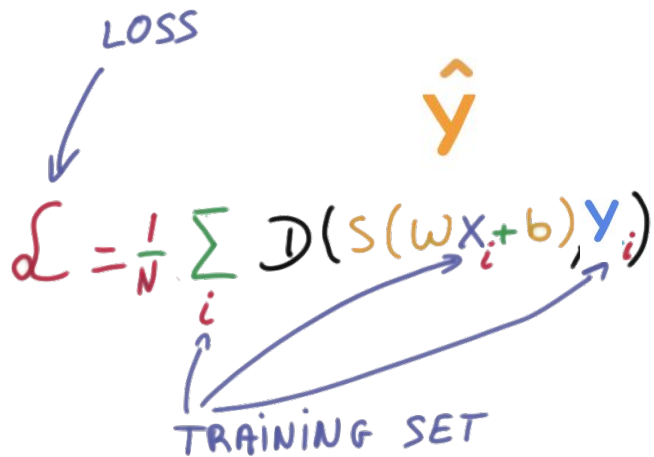
$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}} \quad \text{for } j = 1, \dots, K.$$







Cost function: cross entropy

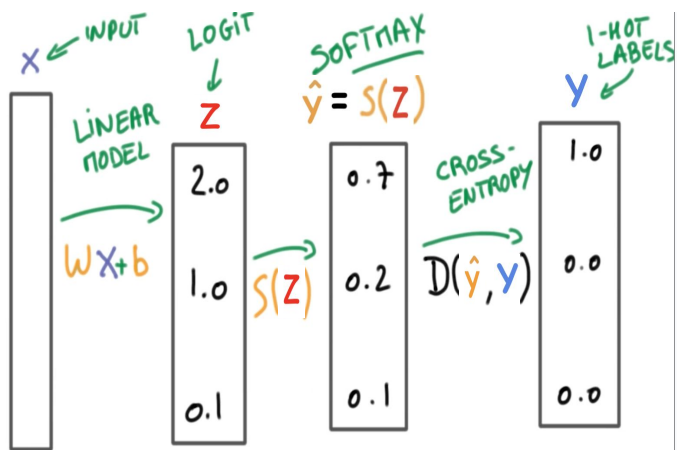


A handwritten diagram of the cross-entropy loss formula. The formula is $\mathcal{L} = \frac{1}{N} \sum \mathcal{D}(s(w x_i + b), y_i)$. Annotations include: a blue arrow labeled "LOSS" pointing to the \mathcal{L} term; a blue arrow labeled "TRAINING SET" pointing to the index i in x_i and y_i ; and an orange \hat{y} symbol above the $s(w x_i + b)$ term. The terms $s(w x_i + b)$ and y_i are colored orange and blue respectively.

$$\mathcal{L} = \frac{1}{N} \sum \mathcal{D}(s(w x_i + b), y_i)$$

$$D(\hat{Y}, Y) = -Y \log \hat{Y}$$

Cost function: cross entropy



$$D(\hat{Y}, Y) = -Y \log \hat{Y}$$

Cross entropy example

```
import numpy as np
```

One hot

0: 1 0 0 0

1: 0 1 0 0

2: 0 0 1 0

```
Y = np.array([1, 0, 0])
```

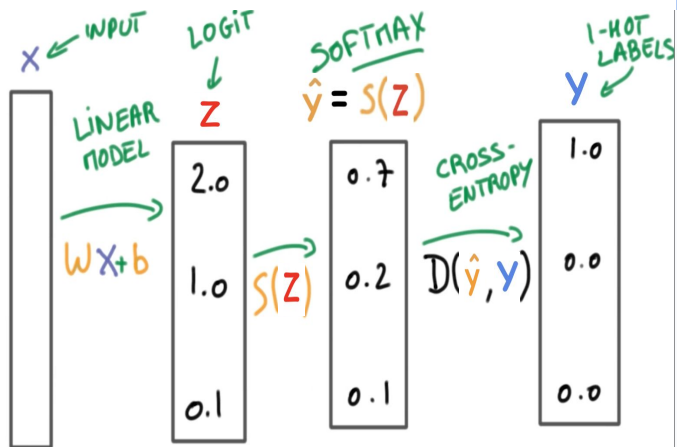
```
Y_pred1 = np.array([0.7, 0.2, 0.1])
```

```
Y_pred2 = np.array([0.1, 0.3, 0.6])
```

```
print("loss1 = ", np.sum(-Y * np.log(Y_pred1)))
```

```
print("loss2 = ", np.sum(-Y * np.log(Y_pred2)))
```


Cross entropy in PyTorch



$$D(\hat{Y}, Y) = -Y \log \hat{Y}$$

```
# Softmax + CrossEntropy (LogSoftmax + NLLLoss)
loss = nn.CrossEntropyLoss()

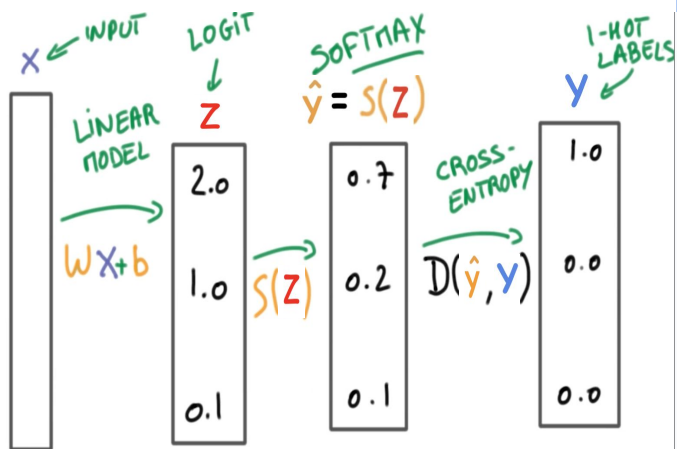
# target is of size nBatch
# each element in target has to have 0 <= value < nClasses (0-2)
# Input is class, not one-hot
Y = Variable(torch.LongTensor([0]), requires_grad=False)

# input is of size nBatch x nClasses = 1 x 4
# Y_pred are logits (not softmax)
Y_pred1 = Variable(torch.Tensor([[2.0, 1.0, 0.1]]))
Y_pred2 = Variable(torch.Tensor([[0.5, 2.0, 0.3]]))

l1 = loss(Y_pred1, Y)
l2 = loss(Y_pred2, Y)

print("PyTorch Loss1 = ", l1.data,
      "\nPyTorch Loss2=", l2.data)
```

Cross entropy in PyTorch



$$D(\hat{Y}, Y) = -Y \log \hat{Y}$$

```
# Softmax + CrossEntropy (LogSoftmax + NLLLoss)
loss = nn.CrossEntropyLoss()

# target is of size nBatch
# element in target has to have 0 <= value < nClasses (0-2)
# Input is class, not one-hot
Y = Variable(torch.LongTensor([2, 0, 1]), requires_grad=False)

# input is of size nBatch x nClasses = 2 x 4
# Y_pred are logits (not softmax)
Y_pred1 = Variable(torch.Tensor([[0.1, 0.2, 0.9],
                                   [1.1, 0.1, 0.2],
                                   [0.2, 2.1, 0.1]]))

Y_pred2 = Variable(torch.Tensor([[0.8, 0.2, 0.3],
                                   [0.2, 0.3, 0.5],
                                   [0.2, 0.2, 0.5]]))

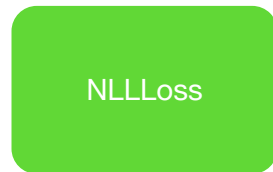
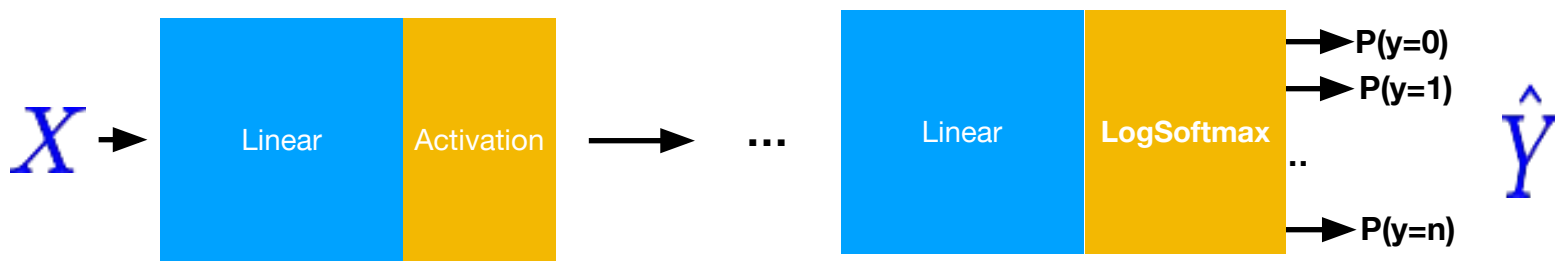
l1 = loss(Y_pred1, Y)
l2 = loss(Y_pred2, Y)

print("Batch Loss1 = ", l1.data, "\nBatch Loss2=", l2.data)
```

Exercise 9-1: CrossEntropyLoss VS NLLLoss

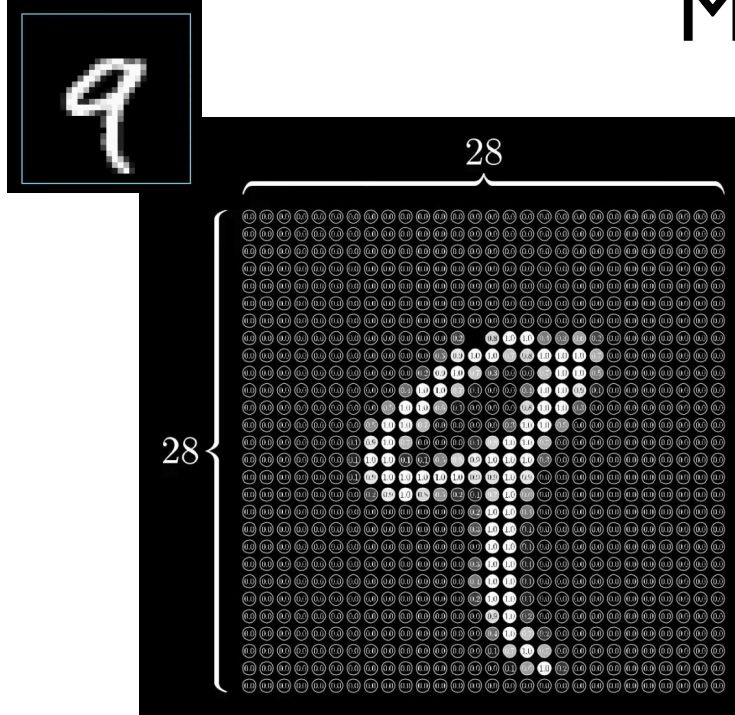
- What are the differences?
- Check out
 - <http://pytorch.org/docs/master/nn.html#nllloss>
 - <http://pytorch.org/docs/master/nn.html#crossentropyloss>
- Minimizing the Negative Log-Likelihood, in English
http://willwolf.io/2017/05/18/minimizing_the_negative_log_likelihood_in_english/

(log)Softmax + NLLLoss



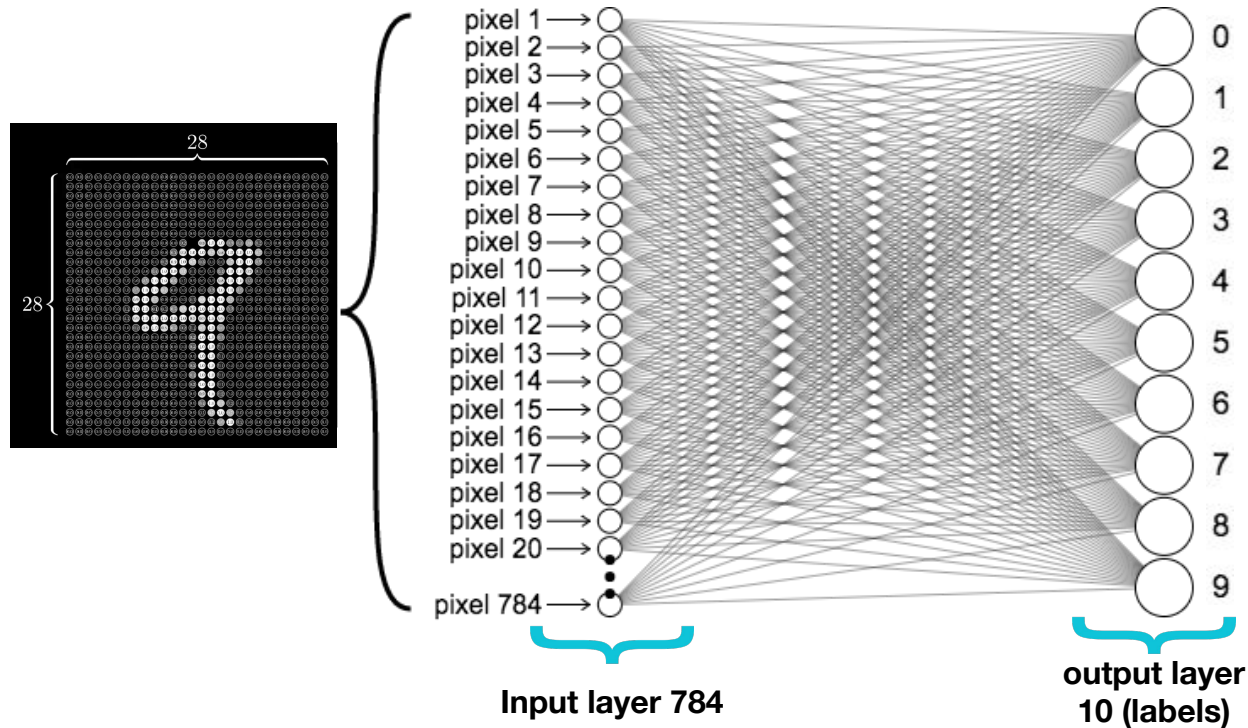
With NLLLoss

MNIST input

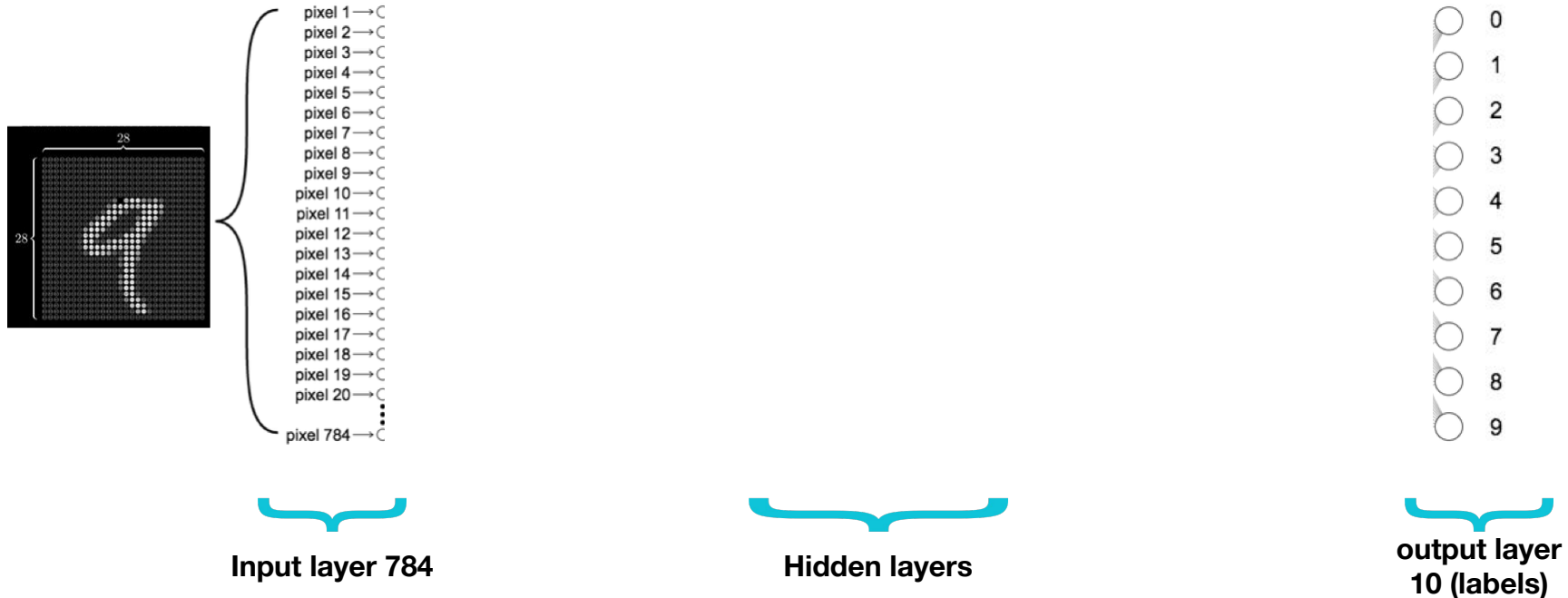


28x28 pixels = 784

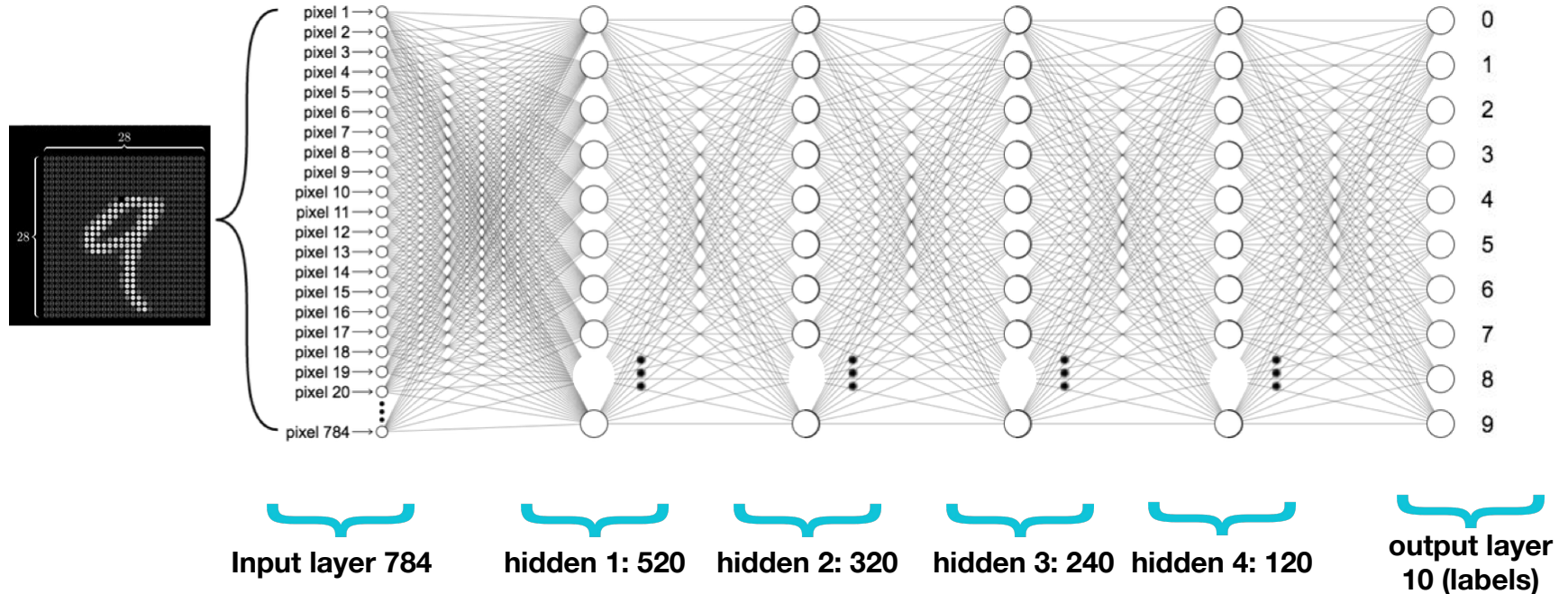
MNIST Network



MNIST Network

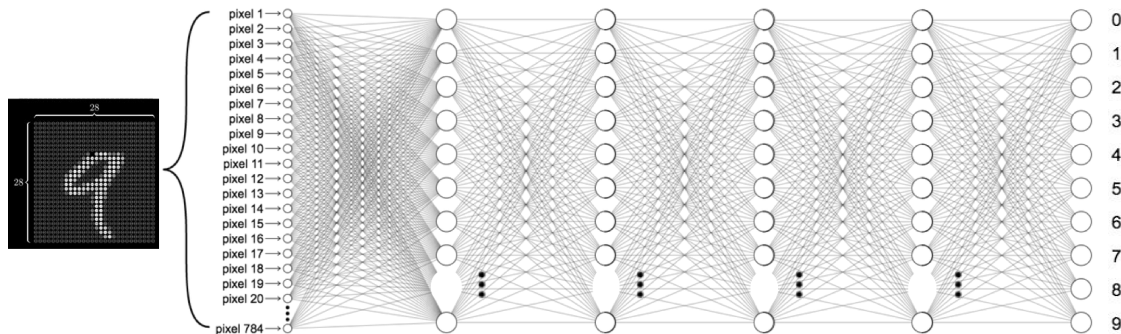


MNIST Network





MNIST Network



```
self.l1 = nn.Linear(784, 500)
self.l2 = nn.Linear(500, 320)
self.l3 = nn.Linear(320, 240)
self.l4 = nn.Linear(240, 120)
self.l5 = nn.Linear(120, 10)
```

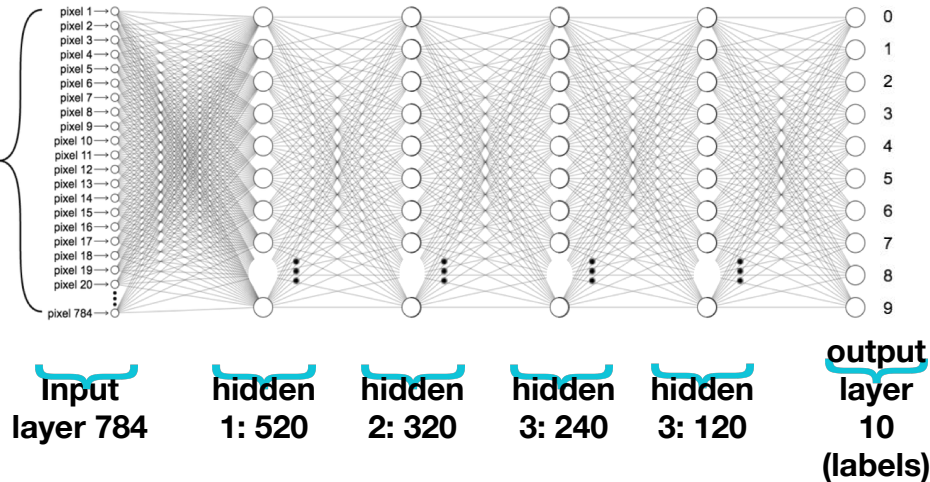
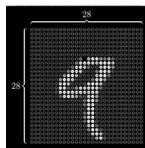
Softmax & NLL loss



```
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))
        return self.l5(x) # No need activation
```



Softmax & NLL loss



```
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))
        return self.l5(x) # No need activation
```

```
        criterion = nn.CrossEntropyLoss()
        ...
        for batch_idx, (data, target) in enumerate(train_loader):
            data, target = Variable(data), Variable(target)
            optimizer.zero_grad()
            output = model(data)
            loss = criterion(output, target)
            loss.backward()
            optimizer.step()
```

```
# 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)

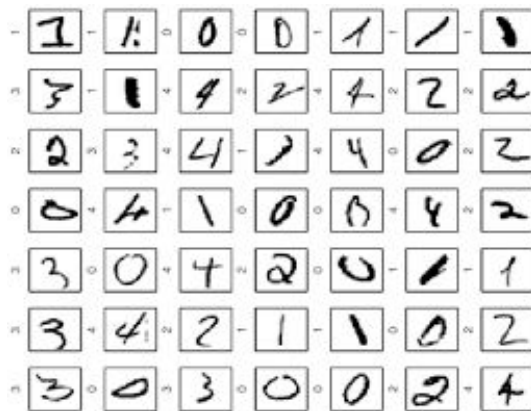
    def forward(self, x):
        x = x.view(-1, 784) # Flatten the data (n, 1, 28, 28)-> (n, 784)
        x = F.relu(self.l1(x))
        x = F.relu(self.l2(x))
        x = F.relu(self.l3(x))
        x = F.relu(self.l4(x))
        return self.l5(x)

model = Net()

criterion = nn.CrossEntropyLoss()
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 = criterion(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]))
```

MNIST Softmax



```
# 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))
        return self.l5(x)

model = Net()

criterion = nn.CrossEntropyLoss()
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 = criterion(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 += criterion(output, target, size_average=False).data[0]
        # get the index of the max log-probability
        pred = torch.max(output.data, 1)[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)))
```



```
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):
    ...
```

```
Y_pred_scores = Variable(
    torch.Tensor([[0.3, 0.2, 0.9, 0.1]]))

# torch.max returns both the max values and indices
Y_pred_val, Y_pred_idx =
    torch.max(Y_pred_scores.data, 1)

Y_pred_idx = torch.max(Y_pred_scores.data, 1)[1]
```

```
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 += criterion(output, target, size_average=False).data[0]
        # get the index of the max log-probability
        pred = torch.max(output.data, 1)[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)))
```



Accuracy?

```
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 = criterion(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 += criterion(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)))

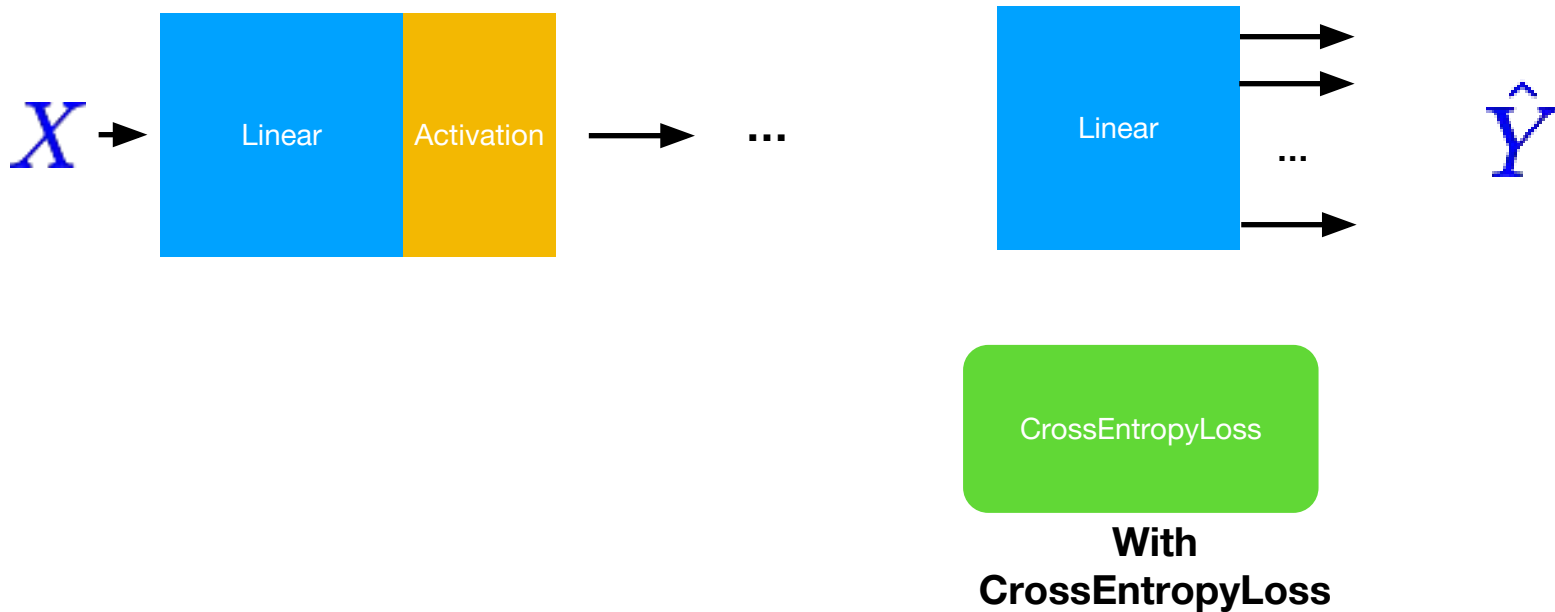
for epoch in range(1, 10):
    train(epoch)
    test()
```

Train Epoch: 1	[0/60000 (0%)]	Loss: 2.313209
Train Epoch: 1	[640/60000 (1%)]	Loss: 2.303560
Train Epoch: 1	[1280/60000 (2%)]	Loss: 2.296464
Train Epoch: 1	[1920/60000 (3%)]	Loss: 2.297758
Train Epoch: 1	[2560/60000 (4%)]	Loss: 2.308579
Train Epoch: 1	[3200/60000 (5%)]	Loss: 2.300100
Train Epoch: 1	[3840/60000 (6%)]	Loss: 2.300800
Train Epoch: 1	[4480/60000 (7%)]	Loss: 2.301295
Train Epoch: 1	[5120/60000 (9%)]	Loss: 2.295039
...		
Train Epoch: 9	[51200/60000 (85%)]	Loss: 0.069267
Train Epoch: 9	[51840/60000 (86%)]	Loss: 0.044378
Train Epoch: 9	[52480/60000 (87%)]	Loss: 0.163481
Train Epoch: 9	[53120/60000 (88%)]	Loss: 0.243676
Train Epoch: 9	[53760/60000 (90%)]	Loss: 0.045024
Train Epoch: 9	[54400/60000 (91%)]	Loss: 0.064958
Train Epoch: 9	[55040/60000 (92%)]	Loss: 0.071447
Train Epoch: 9	[55680/60000 (93%)]	Loss: 0.043712
Train Epoch: 9	[56320/60000 (94%)]	Loss: 0.099484
Train Epoch: 9	[56960/60000 (95%)]	Loss: 0.159727
Train Epoch: 9	[57600/60000 (96%)]	Loss: 0.109291
Train Epoch: 9	[58240/60000 (97%)]	Loss: 0.116370
Train Epoch: 9	[58880/60000 (98%)]	Loss: 0.127303
Train Epoch: 9	[59520/60000 (99%)]	Loss: 0.030254

Test set: Average loss: -12.1596, Accuracy: 9697/10000 (**97%**)

Multiple label prediction?

Just use `CrossEntropyLoss`!



Exercise 9-2

- 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



Lecture 10: CNN