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

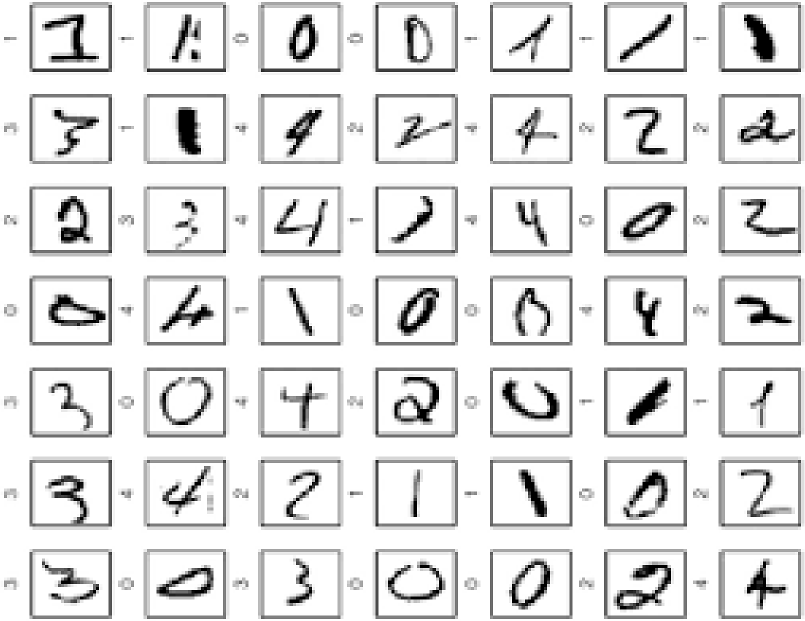
Picture from http://www.tssablog.org/archives/3280

ML/DL for Everyone with PYTORCH

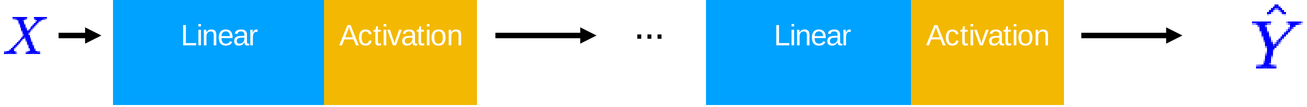
Lecture 9:
Softmax Classifier

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

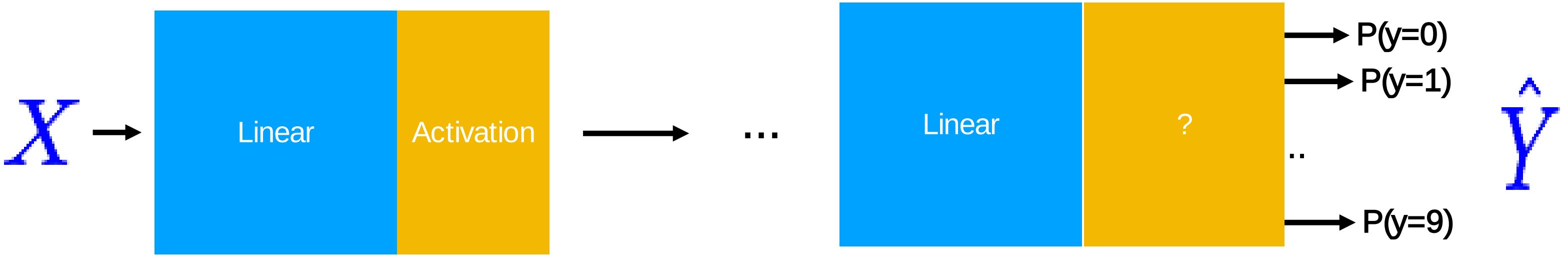
MNIST: 10 labels



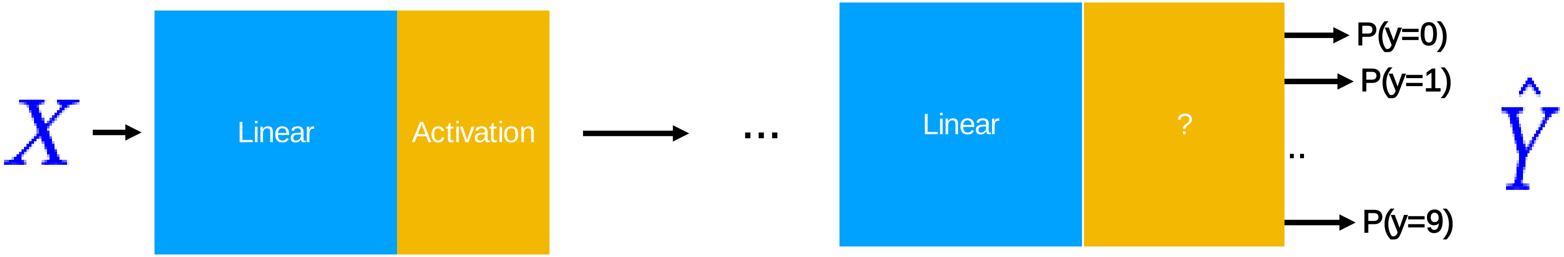
10 labels: 10 outputs



10 labels: 10 outputs

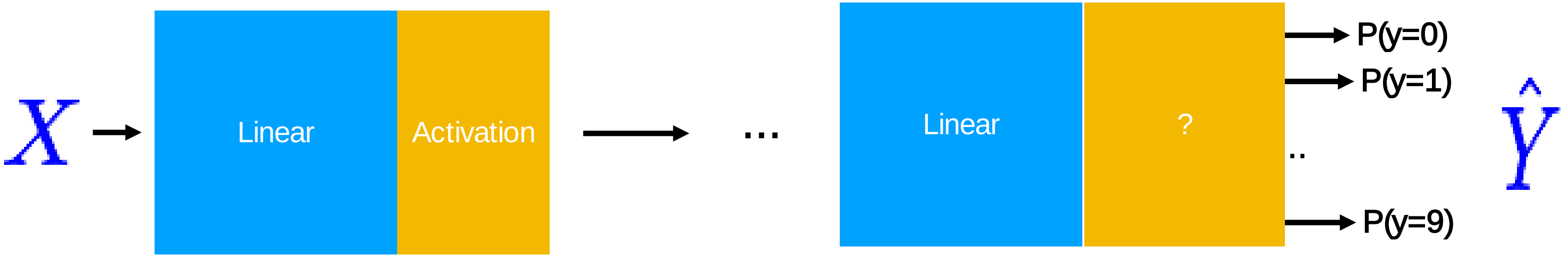


10 outputs



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

10 outputs

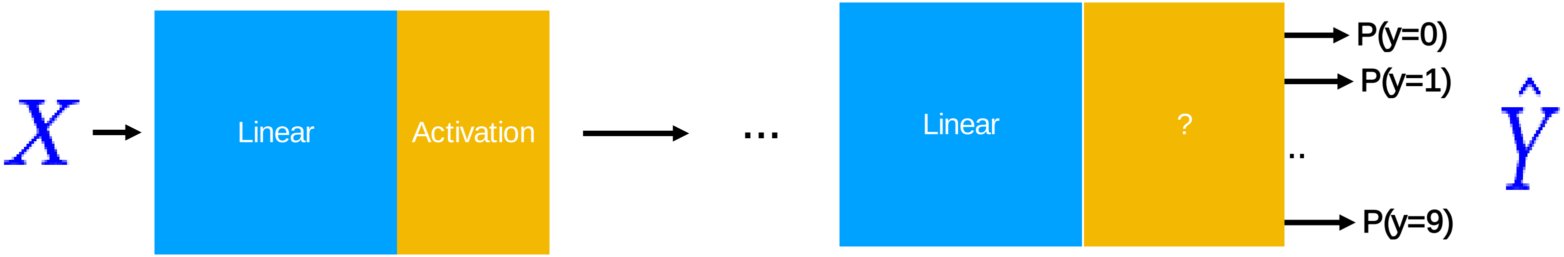


$$\underbrace{\begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \\ \cdots & \cdots \\ a_n & b_n \end{bmatrix}}_{x \in \mathbb{R}^{N \times 2}} \underbrace{\begin{bmatrix} w_1 \\ w_2 \end{bmatrix}}_{w \in \mathbb{R}^{2 \times 1}} = \underbrace{\begin{bmatrix} y_1 \\ y_2 \\ \cdot\ \cdot \\ y_n \end{bmatrix}}_{y \in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

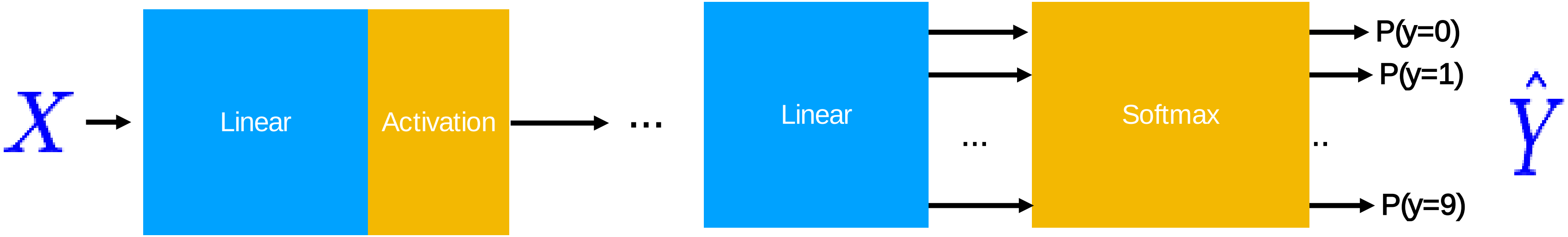
$$\underbrace{\begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \\ \cdots & \cdots \\ a_n & b_n \end{bmatrix}}_{x \in \mathbb{R}^{N \times 2}} \begin{bmatrix} ? \end{bmatrix} = y \in R^{N \times 10}$$

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

Probability

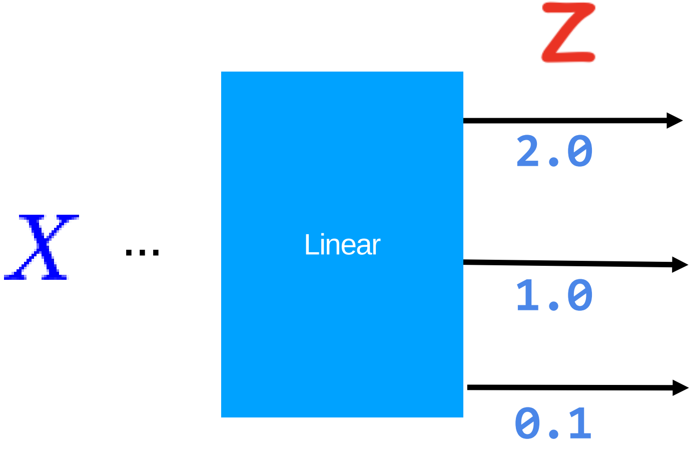


Softmax



Meet Softmax

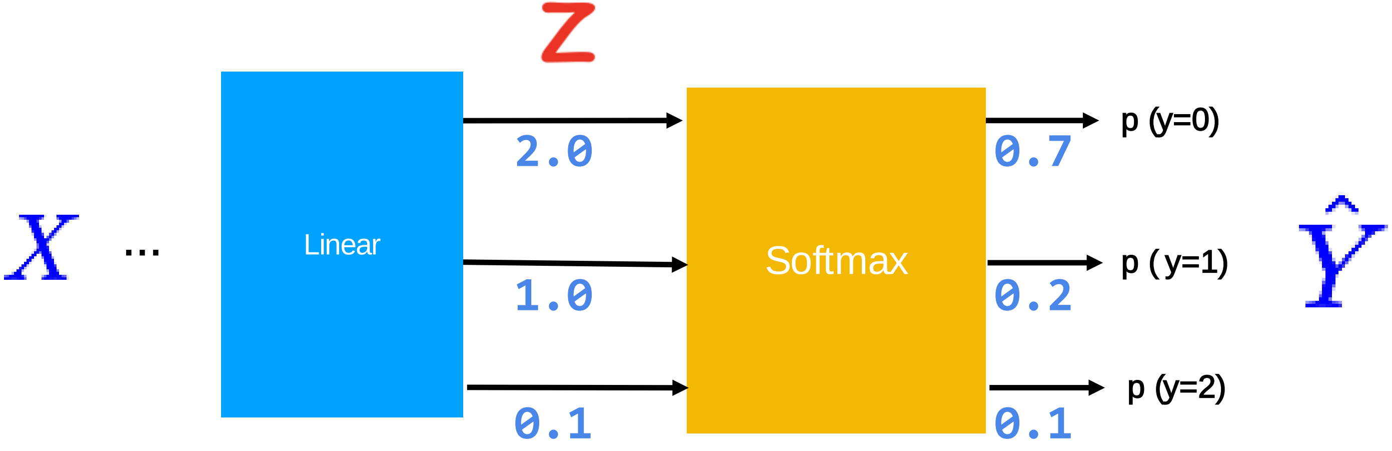
$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$



Scores (Logits)

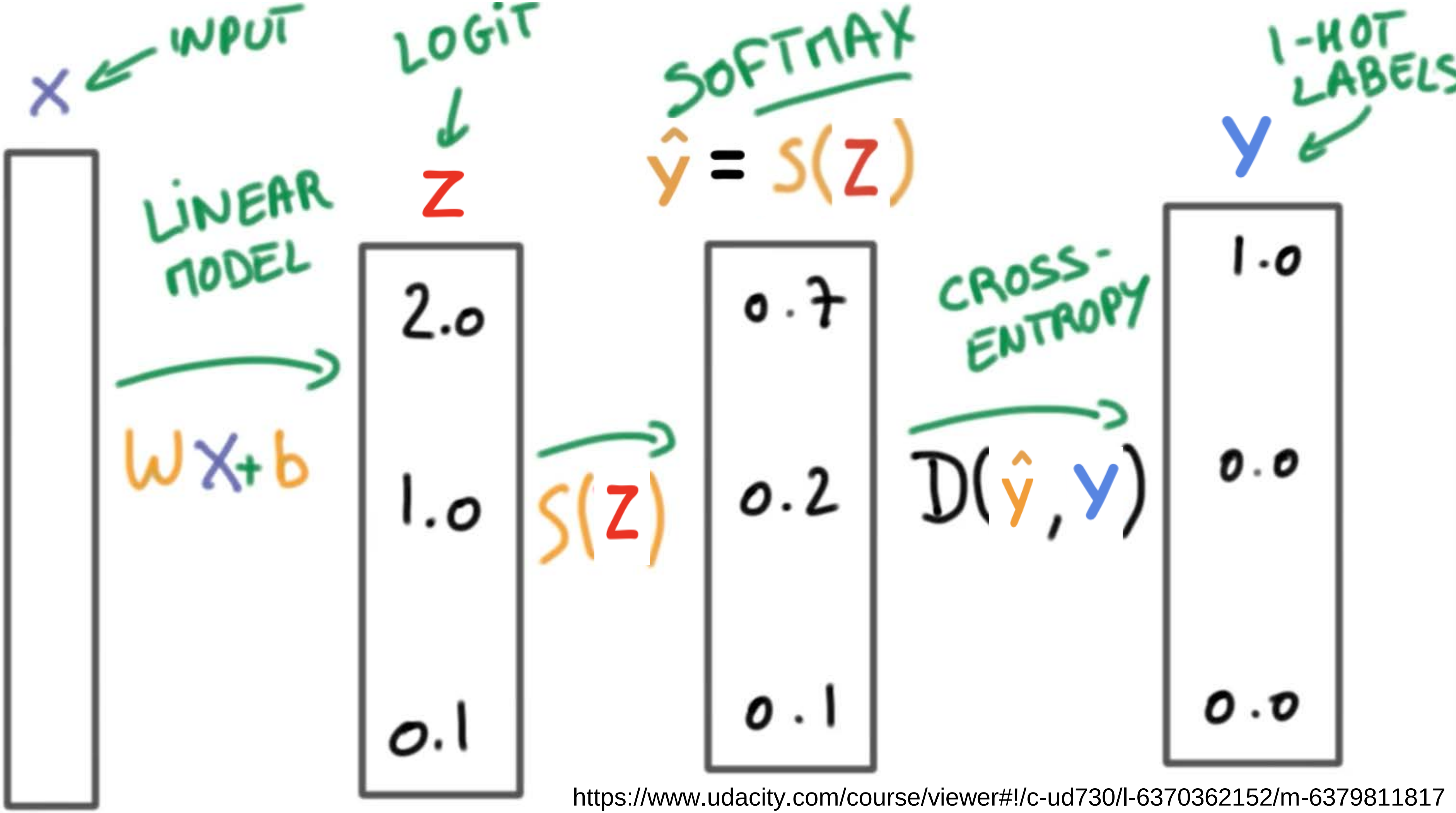
Meet Softmax

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



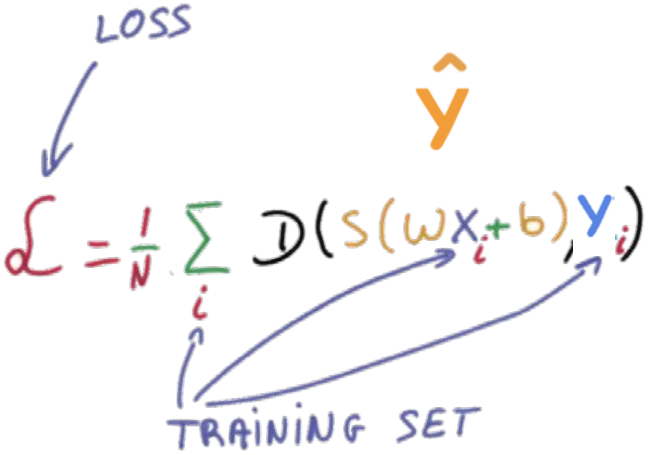
Scores (Logits)

Probabilities



https://www.udacity.com/course/viewer#!/c-ud730/l-6370362152/m-6379811817

Cost function: cross entropy



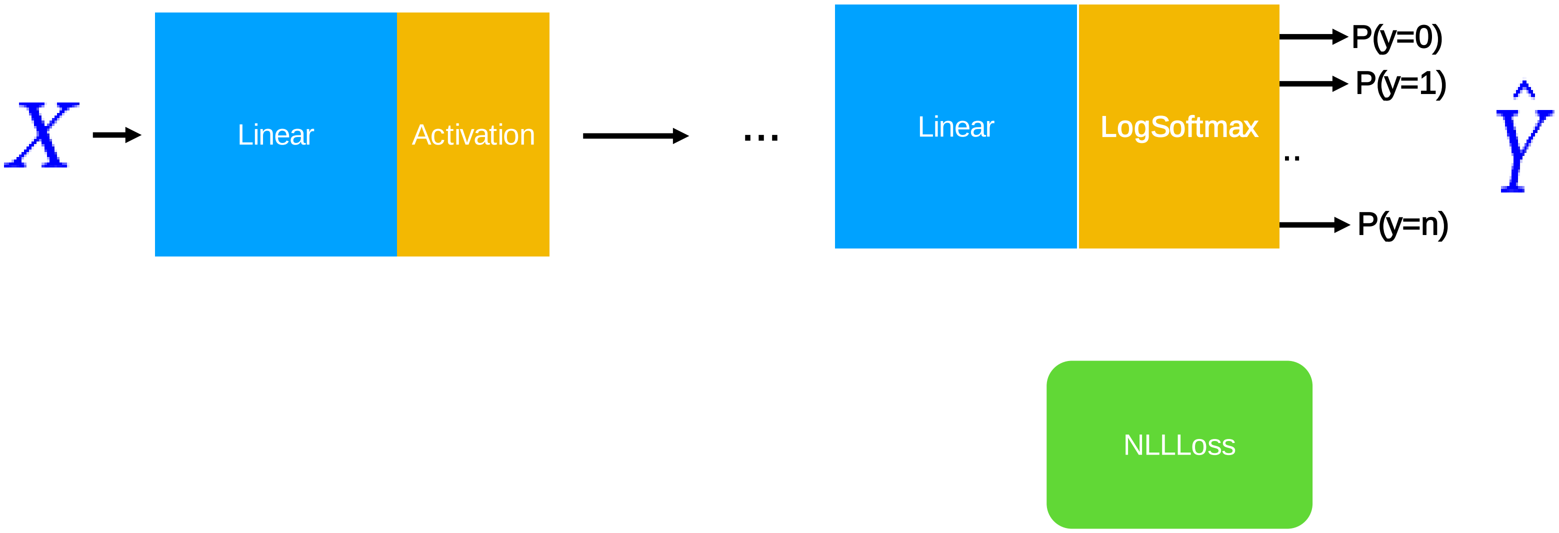
$$L(\hat{y}, y) = -\sum_{k}^{K} y^{(k)} \log \hat{y}^{(k)}$$

https://www.udacity.com/course/viewer#!/c-ud730/l-6370362152/m-6379811817

Exercise 9-1: CrossEntropyLoss VS NLLLoss

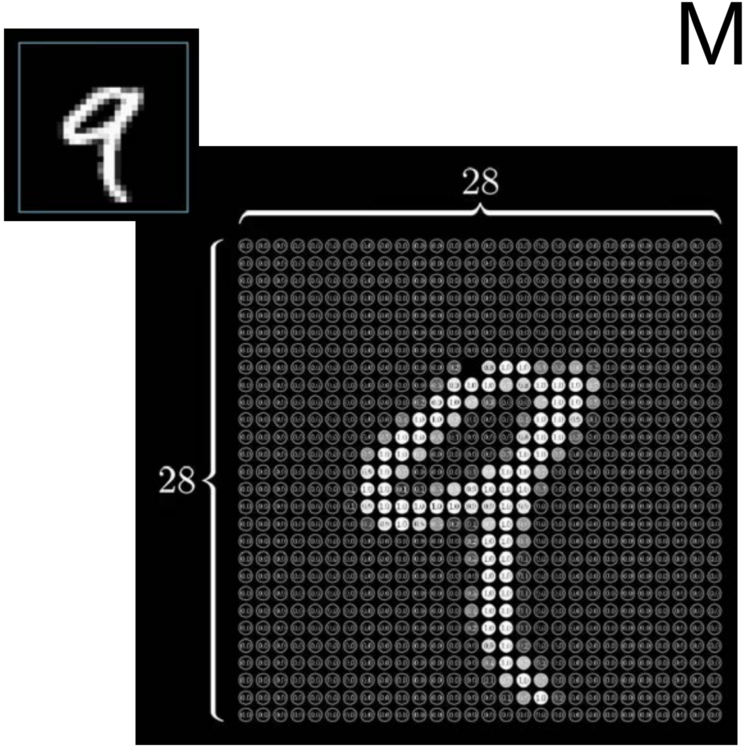
- What are the differences?
- Check out
  - http://pytorch.org/docs/master/nn.html#nllloss
  - http://pytorch.org/docs/master/nn.html#crossentropyloss
- Minimizing the Negative Log-Likelihood, in English http://willwolf.io/2017/05/18/minimizing_the_negative_log_likelihood_in_english/

(log)Softmax + NLLLoss



With NLLLoss

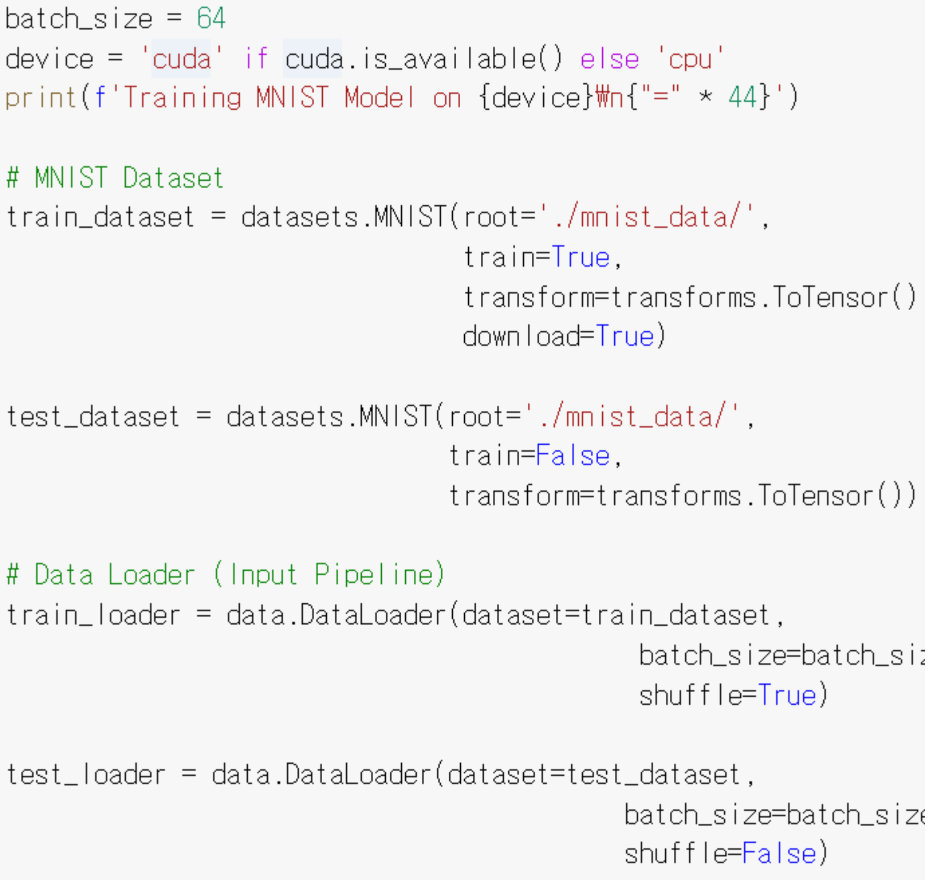
MNIST input



28x28 pixels = 784

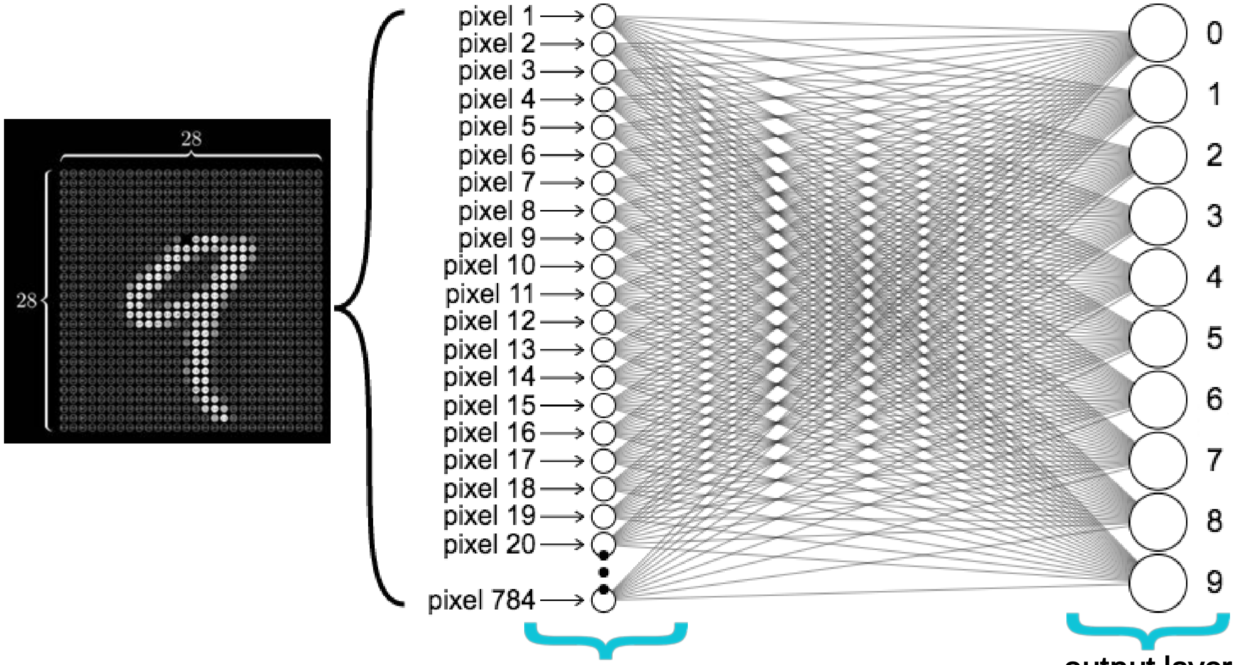
https://www.youtube.com/watch?v=aircAruvnKk&t=1s

Predefined MNIST dataloader



```
batch_size = 64
device = 'cuda' if cuda.is_available() else 'cpu'
print(f'Training MNIST Model on {device}₩n{"=" * 44}')

# MNIST Dataset
train_dataset = datasets.MNIST(root='./mnist_data/',
                               train=True,
                               transform=transforms.ToTensor()
                               download=True)

test_dataset = datasets.MNIST(root='./mnist_data/',
                              train=False,
                              transform=transforms.ToTensor())

# Data Loader (Input Pipeline)
train_loader = data.DataLoader(dataset=train_dataset,
                                           batch_size=batch_si
                                           shuffle=True)

test_loader = data.DataLoader(dataset=test_dataset,
                                          batch_size=batch_siz
                                          shuffle=False)
```

MNIST Network

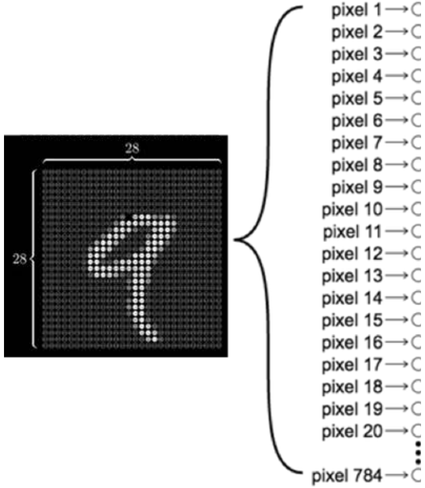


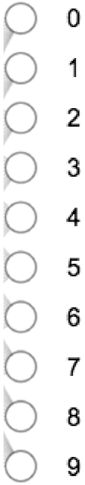
Input layer 784

output layer
10 (labels)

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

MNIST Network





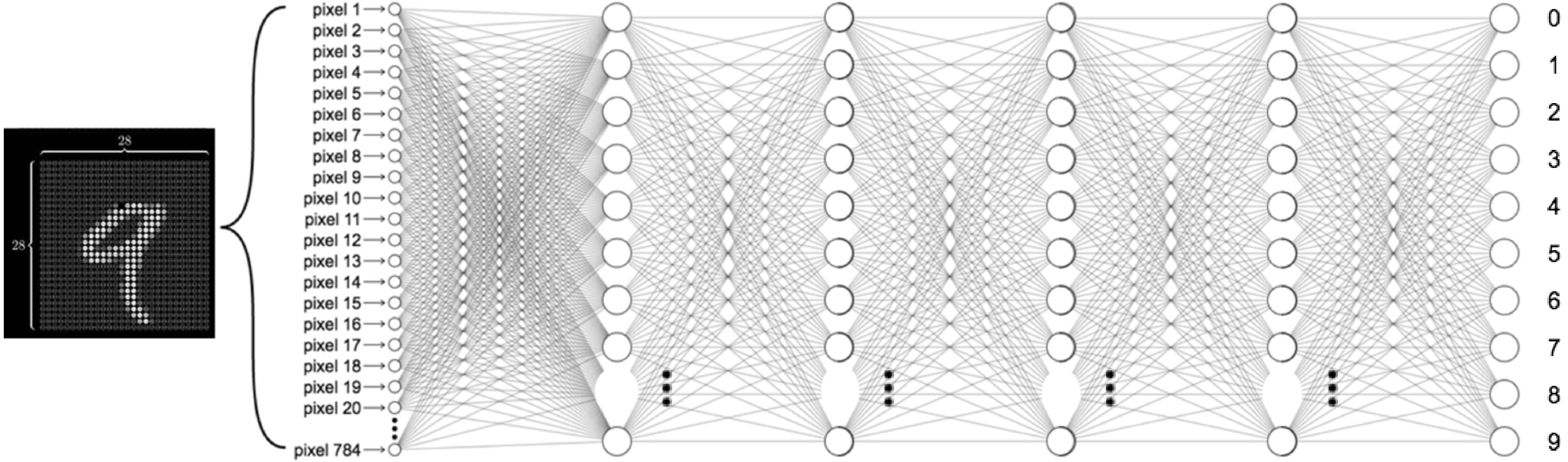
Input layer 784

Hidden layers

output layer
10 (labels)

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

MNIST Network



Input layer 784

hidden 1: 520

hidden 2: 320

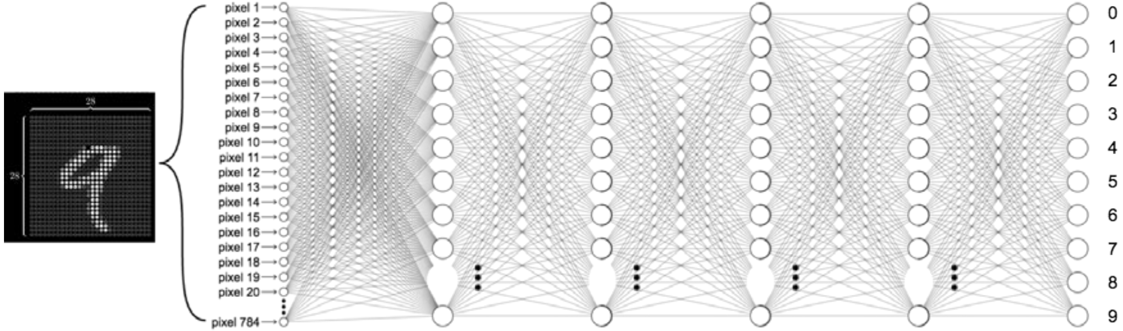
hidden 3: 240

hidden 4: 120

output layer
10 (labels)

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

MNIST Network

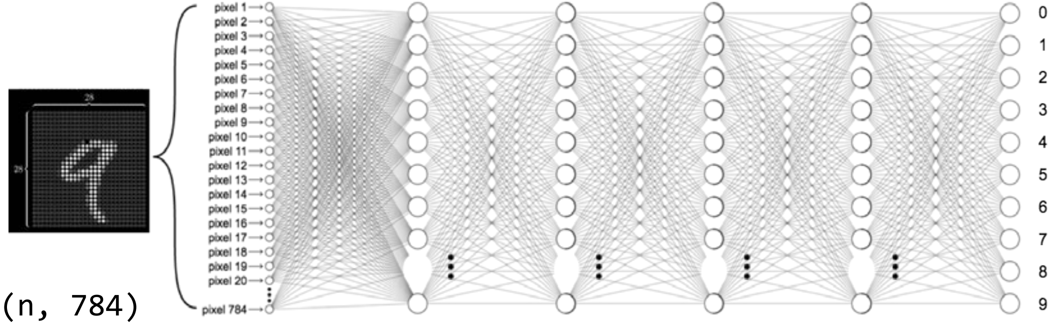


```
self.l1 = nn.Linear(784, 520)
        self.l2 = nn.Linear(520, 320)
                self.l3 = nn.Linear(320, 240)
                       self.l4 = nn.Linear(240, 120)
                              self.l5 = nn.Linear(120, 10)
```

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



input layer 784

hidden 1: 520

hidden 2: 320

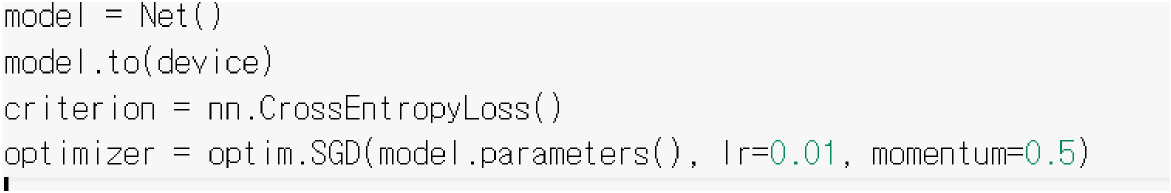
hidden 3: 240

hidden 3: 120

output layer 10 (labels)

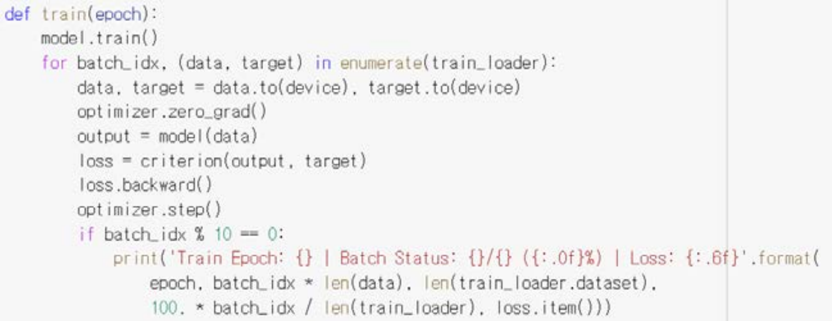
Softmax & NLL loss

```
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
```

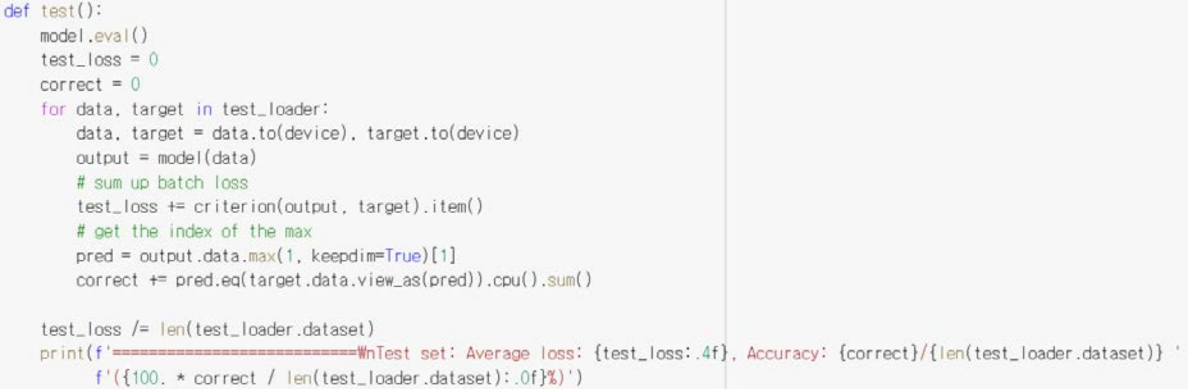


```
model = Net()
model.to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.5)
```

MNIST Train

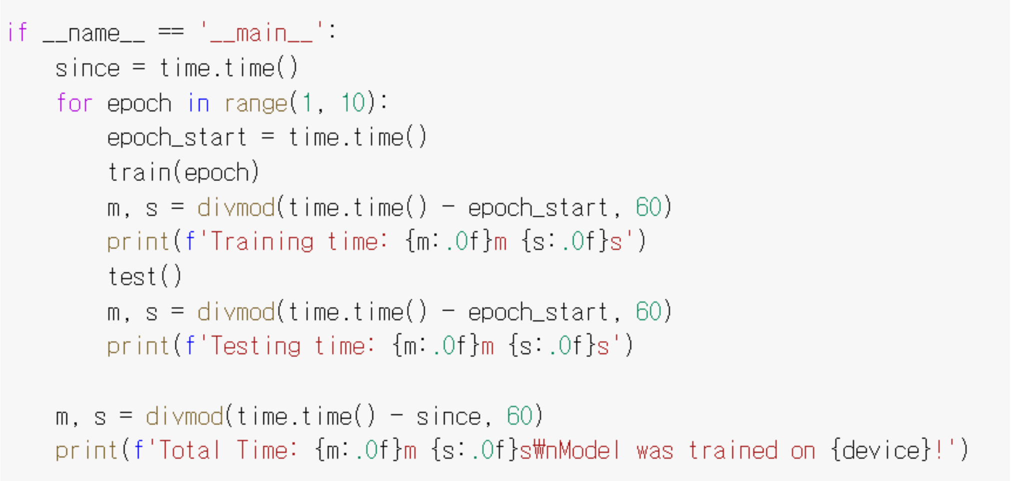


```
def train(epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = criterion(output, target)
        loss.backward()
        optimizer.step()
        if batch_idx % 10 == 0:
            print('Train Epoch: {} | Batch Status: {}/{} ({:.0f}%) | Loss: {:.6f}'.format(
                epoch, batch_idx * len(data), len(train_loader.dataset),
                100. * batch_idx / len(train_loader), loss.item()))
```



```
def test():
    model.eval()
    test_loss = 0
    correct = 0
    for data, target in test_loader:
        data, target = data.to(device), target.to(device)
        output = model(data)
        # sum up batch loss
        test_loss += criterion(output, target).item()
        # get the index of the max
        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(f'===========================WnTest set: Average loss: {test_loss:.4f}, Accuracy: {correct}/{len(test_loader.dataset)} '
          f'({100. * correct / len(test_loader.dataset):.0f}%)')
```

Accuracy?

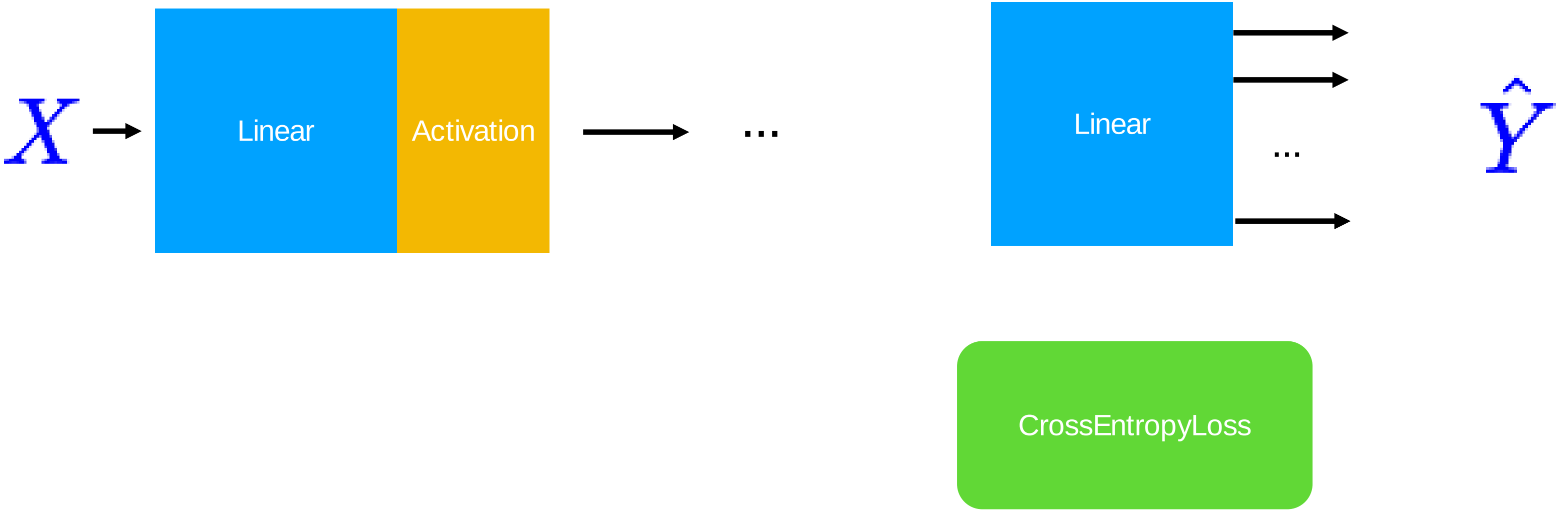


```
if __name__ == '__main__':
    since = time.time()
    for epoch in range(1, 10):
        epoch_start = time.time()
        train(epoch)
        m, s = divmod(time.time() - epoch_start, 60)
        print(f'Training time: {m:.0f}m {s:.0f}s')
        test()
        m, s = divmod(time.time() - epoch_start, 60)
        print(f'Testing time: {m:.0f}m {s:.0f}s')

    m, s = divmod(time.time() - since, 60)
    print(f'Total Time: {m:.0f}m {s:.0f}s₩nModel was trained on {device}!')
```

```
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!



With 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

WHAT
NEXT

Lecture 10: CNN