In [1]:
```
try:
    import torch as t
    import torch.nn as tnn
except ImportError:
    print("Colab users: pytorch comes preinstalled. Select Change Ru")
    print("Local users: Please install pytorch for your hardware using instr
    print("ACG users: Please follow instructions here: https://vikasdhiman.i

    raise
```

In [2]:
```
def wget(url, filename):
    """
    Download files using requests package.
    Better than wget command line because this is cross platform.
    """
    try:
        import requests
    except ImportError:
        import subprocess
        subprocess.call("pip install requests".split())
        import requests
    r = requests.get(url)
    with open(filename, 'wb') as fd:
        for chunk in r.iter_content():
            fd.write(chunk)
```

In [3]:
```
## Doing it the Pytorch way without using our custom feature extraction
DEVICE='cuda:0'
DTYPE=t.float32
import torch
import torch.nn
import torch.optim
import torchvision
from torchvision.transforms import ToTensor
from torch.utils.data import DataLoader

torch.manual_seed(17)
```

These are special classes that are subclassed from DataLoader/Dataset

```
# Getting the dataset, the Pytorch way
all_training_data = torchvision.datasets.MNIST(
    root="data",
    train=True,
    download=True,
    transform=ToTensor()
)
test_data = torchvision.datasets.MNIST(
    root="data",
    train=False,
    download=True,
    transform=ToTensor()
)
```

where to store data downloaded data

train or test

complain or download if the dataset is not in the root directory

what to do with data

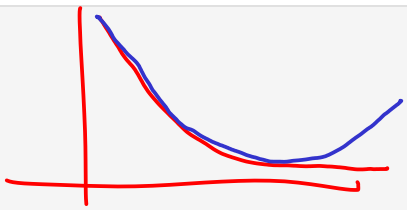
ToTensor() : intializes your data as tensor object

get test data

~~80%, 20%~~ all the training 90% 10%

In [4]:
```
training_data, validation_data = torch.utils.data.random_split(all_training_
```

In [8]:
```
# Hyper parameters
learning_rate = 1e-3 # controls how fast the
batch_size = 64
epochs = 5
momentum = 0.9

training_dataloader = DataLoader(training_data, shuffle=True, batch_size=bat
validation_dataloader = DataLoader(validation_data,  batch_size=batch_size)
test_dataloader = DataLoader(test_data,  batch_size=batch_size)


loss = torch.nn.CrossEntropyLoss()
# TODO:
# Define model = ?
class MLPNetwork(torch.nn.Module):
    def __init__(self, hidden_size=10, nclasses=10, input_size=28*28):
        super().__init__()
        self._layers = torch.nn.ModuleList([torch.nn.Flatten(),
            tnn.Linear(input_size, hidden_size),
            tnn.ReLU(),
            tnn.Linear(hidden_size, nclasses)])
    def forward(self, x):
        for l in self._layers:
            xnext = l(x) # call the layers in sequence
            x = xnext
        return x
model = MLPNetwork()

# alternatively you can also
# hidden_size=10
# nclasses=10
# input_size=28*28
# model = torch.nn.Sequential(torch.nn.Flatten(),
#             tnn.Linear(input_size, hidden_size),
#             tnn.ReLU(),
#             tnn.Linear(hidden_size, nclasses))
#

# Define optimizer
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate, momentum=m

def loss_and_accuracy(model, loss, validation_dataloader, device=DEVICE):
    # Validation loop
    validation_size = len(validation_dataloader.dataset)
    num_batches = len(validation_dataloader)
    test_loss, correct = 0, 0

    with torch.no_grad():
        for X, y in validation_dataloader:
            X = X.to(device)
            y = y.to(device)
            pred = model(X)
```



avoid overfitting →

nn = neural network

__setattr__ checks for instance of tnn.Module

input

self.l1 = tnn.Linear

when the weights are initialize

input

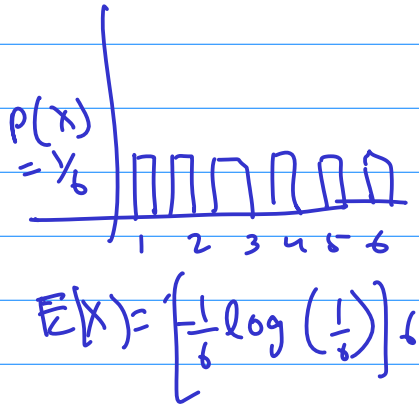
out

SGD → Stochastic Gradient descent

Entropy (Shanon's): measure of randomness

$$E[X] = \sum_{x \in X} -p(X=x) \log_2\left(p(X=x)\right)$$

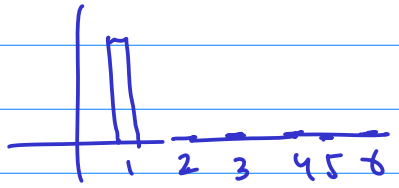
bits ← bits required to capture information in X



$$E[X] = \left[\frac{1}{6}\log\left(\frac{1}{6}\right)\right]_6$$

Cross entropy

$$H[Y, \hat{Y}] = \sum_{y \in Y} -\overbrace{p(y)}^{\text{true}} \log\left(\overbrace{p(\hat{y})}^{\text{predicted}}\right)$$

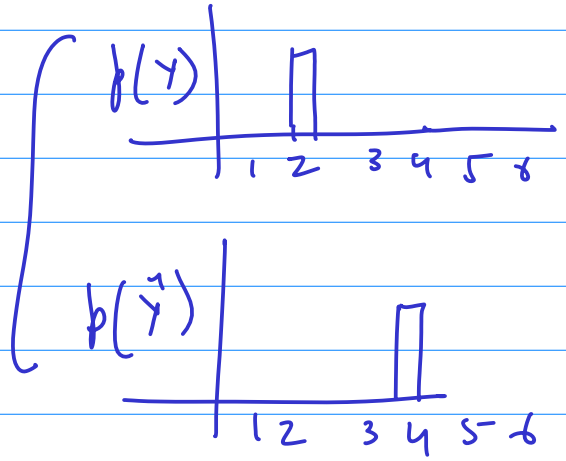


$$E[X] = -1\log(1) + -0\log(0) + -0\log(0) + \dots \text{ 5 times} = 0$$

Cross entropy is a measure of information provided by Y about $\hat{Y}$ and vice versa

- is maximum when the probabilities mismatch
- is minimum when the prob. match

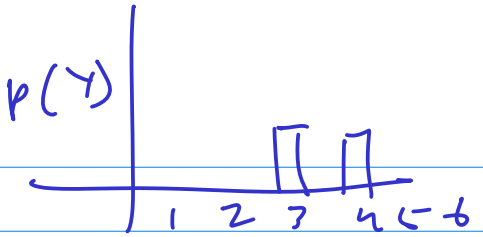
$$H[Y, \hat{Y}] = -0\log 0 + -1\log(0) + -0\log 0 + -0\log 1 + \dots = \infty$$

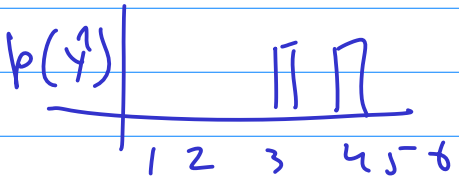


$\log(0^+) \to -\infty$

$$H[Y,\hat{Y}] = +0\log(0) + -1\log(1) + 0\log(0) + \cdots$$

$$= 0.$$





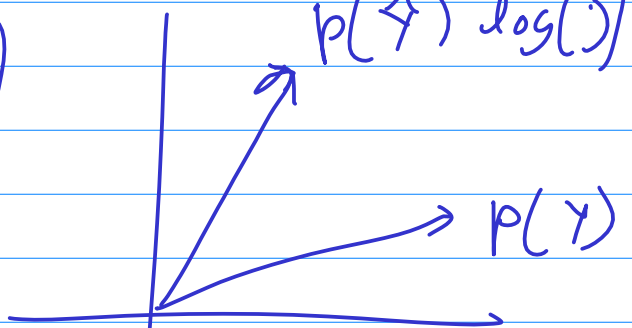
$$H[Y,\hat{Y}] = -0.5\log(0.5) - 0.5\log(0.5) \qquad \log(1) = 0$$

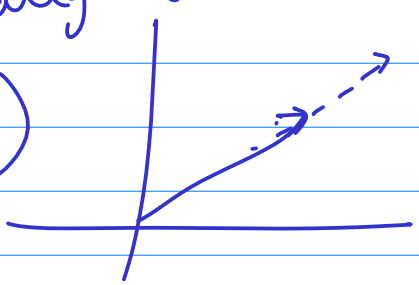
$$\underline{p}(Y) = \begin{bmatrix} p(Y=1) \\ p(Y=2) \\ \vdots \\ p(Y=6) \end{bmatrix}_{6\times 1}$$

$$\underline{p}(\hat{Y}) = \begin{bmatrix} p(\hat{Y}=1) \\ \vdots \\ p(\hat{Y}=6) \end{bmatrix}_{6\times 1} \longrightarrow \log(p(\hat{Y})) = \begin{bmatrix} \log(\,) \\ \vdots \\ \log(\,) \end{bmatrix}$$

$$H[Y,\hat{Y}] = -\underline{p}(Y)^T \left[\log(\underline{p}(\hat{Y}))\right]$$

minimum when $\underline{p}(Y)$ is aligned with $\log(\underline{p}(\hat{Y}))$





$\vec{a}\cdot\vec{b} = |a|\cdot|b|\cos\theta$

max when $\theta = 0$

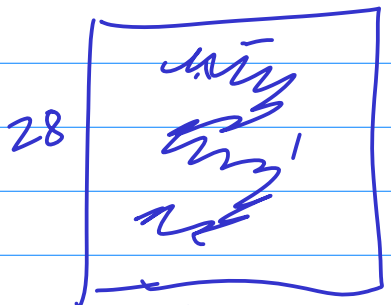
Classification → Binary :- (two classes)
Hinge loss

→ Multi-class
Cross entropy loss

MNIST – 10 digits

$y = 0$
$y = 1$
$y = 2$
$\vdots$
$y = 9$
} 9 possible classes

for every ima



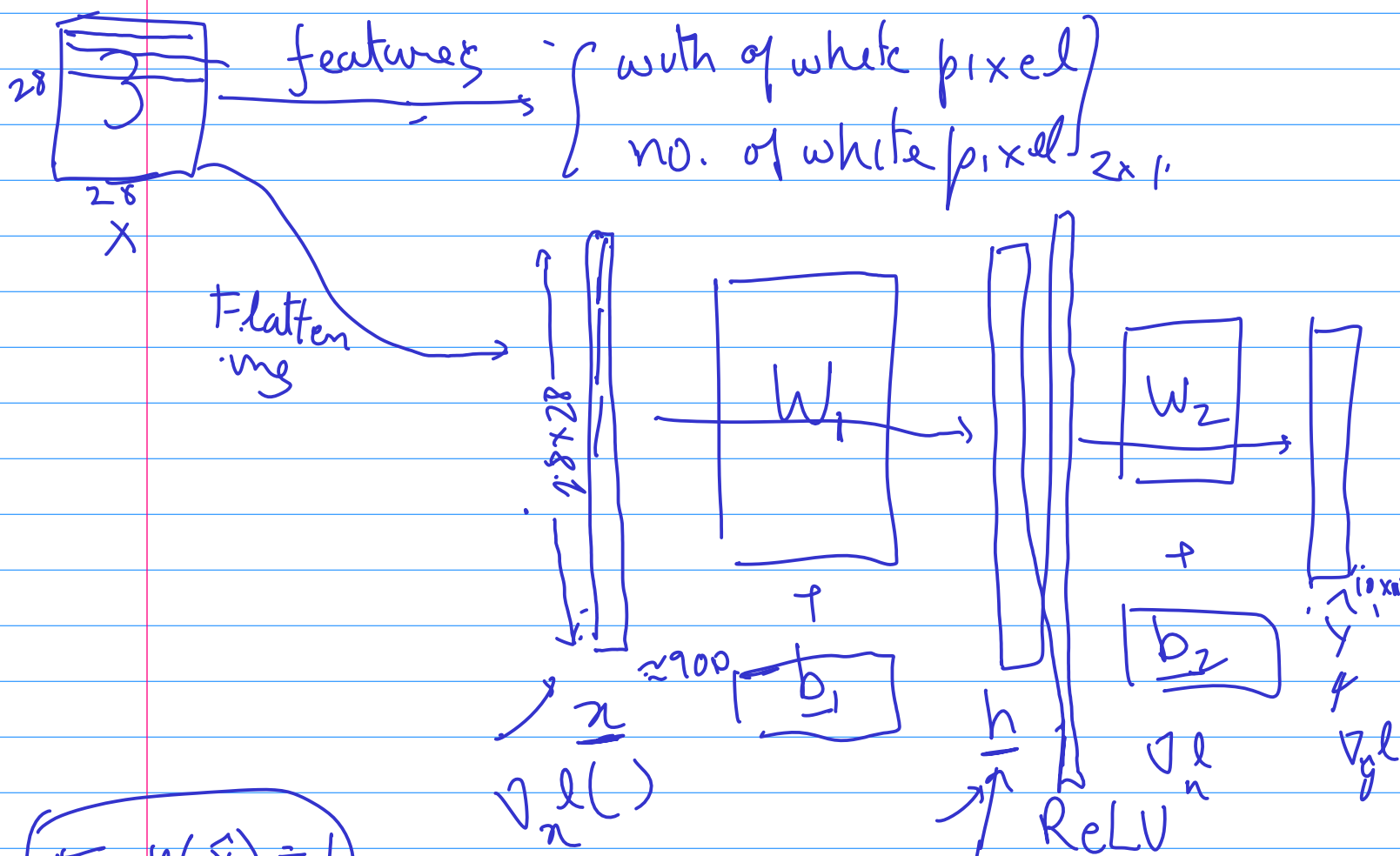
→ label $Y = 3$

↓ convert label to probabilities

$P(Y=0) = 0$
$\vdots$
$P(Y=3) = 1$
$\vdots$
$P(Y=9) = 0$
} one-hot vector

model $f(X;\theta) \rightarrow \hat{Y} = f(X;\theta)$

MLP : Multi layer perceptron



$\sum_{y \in Y} P(y) = 1$

$x \in [0,1]$

$h \in \mathbb{R}^{5\times 1}$

$$\hat{P}(y) = MLP(x; W_2, b_2, W_1, b_1) = \underbrace{b_2 + W_2}_{\text{Linear}}\Big(ReLU\Big(\underbrace{W_1 x + b_1}_{\text{Linear}}\Big)\Big)$$

$\hat{P}(y) \in \mathbb{R}^{10\times 1}$

$\mathbb{R} = float32 =$ 4 bytes

$h_{5\times 1}$

$x \in \mathbb{R}^{900\times 1}$

$W_1 \in \mathbb{R}^{h\times 900\times 4}$ $\quad b_1 \in \mathbb{R}^{h\times 1\times 4}$

$W_2 \in \mathbb{R}^{10\times h\times 4}$ $\quad b_2 \in \mathbb{R}^{10\times 1\times 4}$

Total bytes $= 5\times 900\times 4 + 5\times 4 + 10\times 5\times 4 + 10\times 4$

$$\underline{z} \in \underbrace{(-\infty, \infty)}_{\mathbb{R}}^{n\times 1} \xrightarrow{\boxed{\text{softmax}}} \underbrace{(0,1)^{n\times 1}}_{\underline{p}(\underline{z}) \;\; \sum \underline{p}(z) = 1}$$

Softmax

$$p(z_i) = \frac{\exp(z_i)}{\sum_{i=1}^{n} \exp(z_i)}$$

$$\begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix} \xrightarrow{\text{softmax}} \underbrace{\begin{bmatrix} \frac{\exp(z_1)}{\sum \exp(z_i)} \\ \frac{\exp(z_2)}{\sum \exp(z_i)} \\ \vdots \end{bmatrix}}_{p(\underline{z})}$$

$$\exp(\underline{z}) = \begin{bmatrix} \exp(z_1) \\ \vdots \\ \exp(z_n) \end{bmatrix}$$

$$\boxed{\text{softmax}(\underline{z}) = \exp(\underline{z}) \,/\, \exp(\underline{z})^T \mathbb{1}_{n\times 1}}$$

$p(\hat{y}) = \text{Softmax}(\text{MLP}(x_i)) \in \mathbb{R}^{10\times 1}$

$p(y) =$ one-hot vector

$\text{Loss}(y, \hat{y}) = -\sum_{y\in Y} p(y) \log(\hat{y})$

if correct label = 3

$p(y=3) = 1$

$= -\log(p(\hat{y}=3))$

$= -\log\left(\frac{\exp(\hat{y}_3)}{\sum_i \exp(\hat{y}_i)}\right)$

Gradient descent

$D_T = \{(x_1, y_1) \dots$

$l(y_i, \hat{y}_i) \qquad \hat{y}_i = f(x_i; W)$

$L(D, W) = \sum_{i\in D} l(y_i, \hat{y}_i)$] Loss over the entire dataset

1-step

$W_{t+1} = W_t - \alpha \nabla_W \sum_{i\in D} l(y_i, \hat{y}_i)$ × not feasible for big data sets

$$w_{t+1} = w_t - \alpha \sum_{i \in D} \left(\nabla_w \ell(y_i, \hat{y}_i; w_{t,0}) \right)$$

GD

$\Updownarrow$ x

1 epoch

```
for i in D:
```

$$w_{t+1,i} = w_{t,i} - \alpha \nabla_w \ell(y_i, \hat{y}_i; w_{t,i})$$

one sample at a time

Stochastic GD

Batch SGD batch size

1 epoch

```
for b in range(D/batch_size)
```

$$w_{t+1,b} = w_{t,b} - \alpha \nabla_w \sum_{\text{batchsize}} \ell(y_b, \hat{y}_b)$$

one batch at time

SGD with momentum (Next time)

```
            test_loss += loss(pred, y).item()
            correct += (pred.argmax(dim=-1) == y).type(DTYPE).sum().item()

    test_loss /= num_batches
    correct /= validation_size
    return test_loss, correct

def train(model, loss, training_dataloader, validation_dataloader, device=DE
    model.to(device)
    train_losses = []
    valid_losses = []
    for t in range(epochs):
        # Train loop
        training_size = len(training_dataloader.dataset)
        for batch, (X, y) in enumerate(training_dataloader):
            X = X.to(device)
            y = y.to(device)
            # Compute prediction and loss
            pred = model(X)
            loss_t = loss(pred, y)

            # Backpropagation
            optimizer.zero_grad()
            loss_t.backward()
            optimizer.step()

            if batch % 100 == 0:
                loss_t, current = loss_t.item(), (batch + 1) * len(X)
                print(f"loss: {loss_t:>7f}  [{current:>5d}/{training_size:>5
                train_losses.append(loss_t)
                valid_loss, correct = loss_and_accuracy(model, loss, validat
                valid_losses.append(valid_loss)
                print(f"Validation Error: \n Accuracy: {(100*correct):>0.1f}
    return model, train_losses, valid_losses

trained_model, train_losses, valid_losses = train(model, loss, training_data

test_loss, correct = loss_and_accuracy(model, loss, test_dataloader)
print(f"Test Error: \n Accuracy: {(100*correct):>0.1f}%, Avg loss: {test_los
```

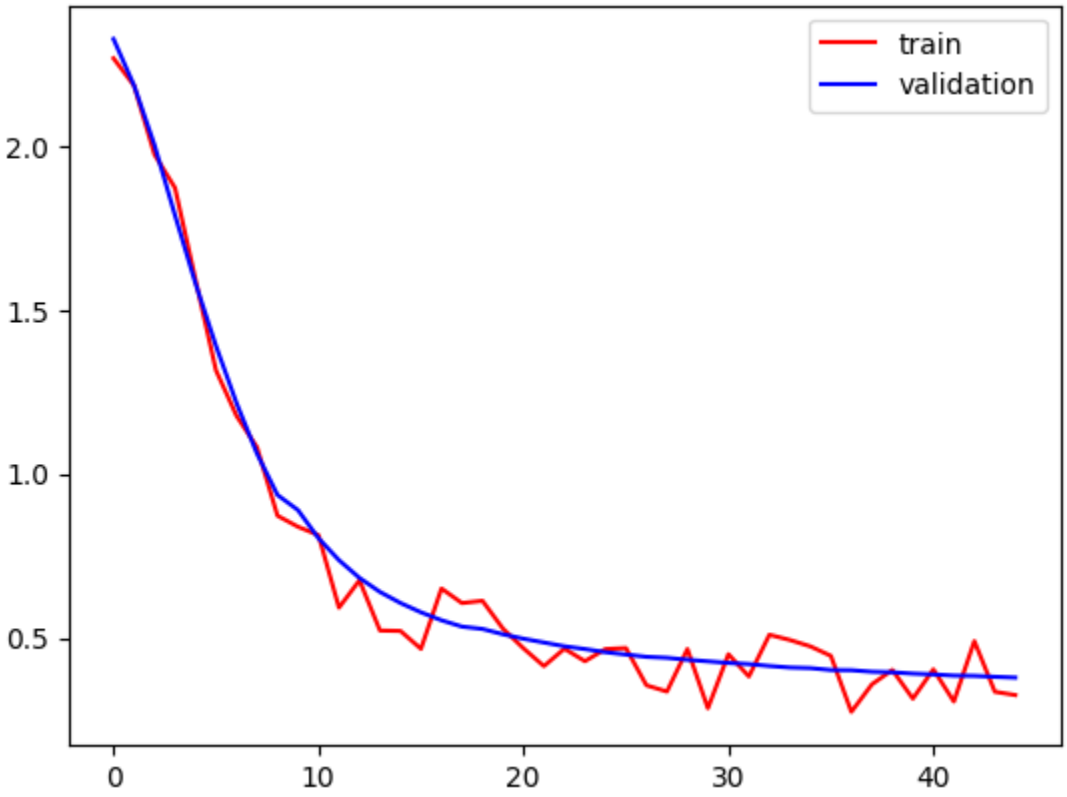
```
Validation Error:   64/54000]
 Accuracy: 10.9%, Avg loss: 2.325170

Validation Error: 6464/54000]
 Accuracy: 13.6%, Avg loss: 2.183869

Validation Error: 2864/54000]
 Accuracy: 34.1%, Avg loss: 2.001862

Validation Error: 9264/54000]
 Accuracy: 46.8%, Avg loss: 1.787784

Validation Error: 5664/54000]
 Accuracy: 54.8%, Avg loss: 1.580385

Validation Error: 2064/54000]
 Accuracy: 62.2%, Avg loss: 1.389772

Validation Error: 8464/54000]
 Accuracy: 68.5%, Avg loss: 1.216087

Validation Error: 4864/54000]
 Accuracy: 75.8%, Avg loss: 1.061541

Validation Error: 1264/54000]
 Accuracy: 79.9%, Avg loss: 0.935546

Validation Error:   64/54000]
 Accuracy: 80.2%, Avg loss: 0.889401

Validation Error: 6464/54000]
 Accuracy: 82.1%, Avg loss: 0.803695

Validation Error: 2864/54000]
 Accuracy: 82.9%, Avg loss: 0.736784

Validation Error: 9264/54000]
 Accuracy: 83.7%, Avg loss: 0.682686

Validation Error: 5664/54000]
 Accuracy: 84.5%, Avg loss: 0.639954

Validation Error: 2064/54000]
 Accuracy: 85.1%, Avg loss: 0.606284

Validation Error: 8464/54000]
 Accuracy: 85.5%, Avg loss: 0.577947

Validation Error: 4864/54000]
 Accuracy: 86.0%, Avg loss: 0.553612

Validation Error: 1264/54000]
 Accuracy: 86.4%, Avg loss: 0.534344

Validation Error:   64/54000]
 Accuracy: 86.7%, Avg loss: 0.527205
```

```
Validation Error: 6464/54000]
 Accuracy: 86.8%, Avg loss: 0.510988

Validation Error: 2864/54000]
 Accuracy: 87.1%, Avg loss: 0.497435

Validation Error: 9264/54000]
 Accuracy: 87.0%, Avg loss: 0.485881

Validation Error: 5664/54000]
 Accuracy: 87.4%, Avg loss: 0.474297

Validation Error: 2064/54000]
 Accuracy: 87.4%, Avg loss: 0.465911

Validation Error: 8464/54000]
 Accuracy: 87.7%, Avg loss: 0.456387

Validation Error: 4864/54000]
 Accuracy: 87.9%, Avg loss: 0.449242

Validation Error: 1264/54000]
 Accuracy: 87.9%, Avg loss: 0.442204

Validation Error:   64/54000]
 Accuracy: 88.2%, Avg loss: 0.439258

Validation Error: 6464/54000]
 Accuracy: 88.1%, Avg loss: 0.433261

Validation Error: 2864/54000]
 Accuracy: 88.0%, Avg loss: 0.428486

Validation Error: 9264/54000]
 Accuracy: 88.3%, Avg loss: 0.423615

Validation Error: 5664/54000]
 Accuracy: 88.5%, Avg loss: 0.420128

Validation Error: 2064/54000]
 Accuracy: 88.5%, Avg loss: 0.414353

Validation Error: 8464/54000]
 Accuracy: 88.4%, Avg loss: 0.409636

Validation Error: 4864/54000]
 Accuracy: 88.7%, Avg loss: 0.407865

Validation Error: 1264/54000]
 Accuracy: 88.8%, Avg loss: 0.401992

Validation Error:   64/54000]
 Accuracy: 88.9%, Avg loss: 0.401415

Validation Error: 6464/54000]
```

```
 Accuracy: 88.9%, Avg loss: 0.396768

Validation Error: 2864/54000]
 Accuracy: 88.9%, Avg loss: 0.395058

Validation Error: 9264/54000]
 Accuracy: 89.0%, Avg loss: 0.391086

Validation Error: 5664/54000]
 Accuracy: 88.9%, Avg loss: 0.388569

Validation Error: 2064/54000]
 Accuracy: 89.0%, Avg loss: 0.385012

Validation Error: 8464/54000]
 Accuracy: 89.0%, Avg loss: 0.384191

Validation Error: 4864/54000]
 Accuracy: 89.3%, Avg loss: 0.381349

Validation Error: 1264/54000]
 Accuracy: 89.3%, Avg loss: 0.378932

Test Error:
 Accuracy: 90.0%, Avg loss: 0.351287
```

In [10]:
```
import matplotlib.pyplot as plt
plt.plot(train_losses, 'r', label='train')
plt.plot(valid_losses, 'b', label='validation')
plt.legend()
```

Out[10]:
```
<matplotlib.legend.Legend at 0x7f92d52b5900>
```



In [11]:
```
torch.nn.__file__
```

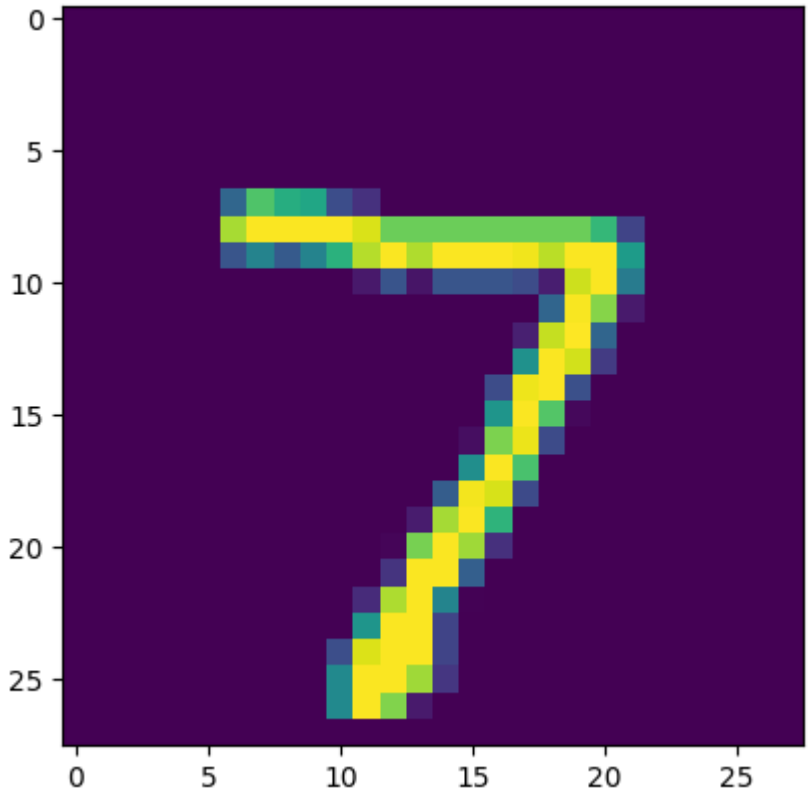
Out[11]:
```
'/home/vdhiman/.local/share/virtualenvs/nbgrader-notebooks-_16a_jDm/lib/python3.10/site-packages/torch/nn/__init__.py'
```

In [12]:
```
X, _ = next(iter(test_dataloader))
X.shape
```

Out[12]:
```
torch.Size([64, 1, 28, 28])
```

In [13]:
```
import matplotlib.pyplot as plt
plt.imshow(X[0, 0])
```

Out[13]:
```
<matplotlib.image.AxesImage at 0x7f92d02888b0>
```



In [14]:
```
print("The predicted image label is ", model(X.to(DEVICE)).argmax(dim=-1)[0]
```

```
The predicted image label is  7
```

In [ ]: