PyTorch Tutorial

朱书琦　zhusq22@mails.tsinghua.edu.cn

You need...

- 比较熟练使用包管理软件，推荐conda
  - 隔离性
  - 自带包依赖性分析功能
  - 避免版本冲突
- 比较熟练使用Python编程
  - 基本语句
  - 查询API
- 了解神经网络架构和基本原理（数理基础）
  - MLP、CNN、RNN
  - Back Propagation
  - Stochastic Gradient Descent

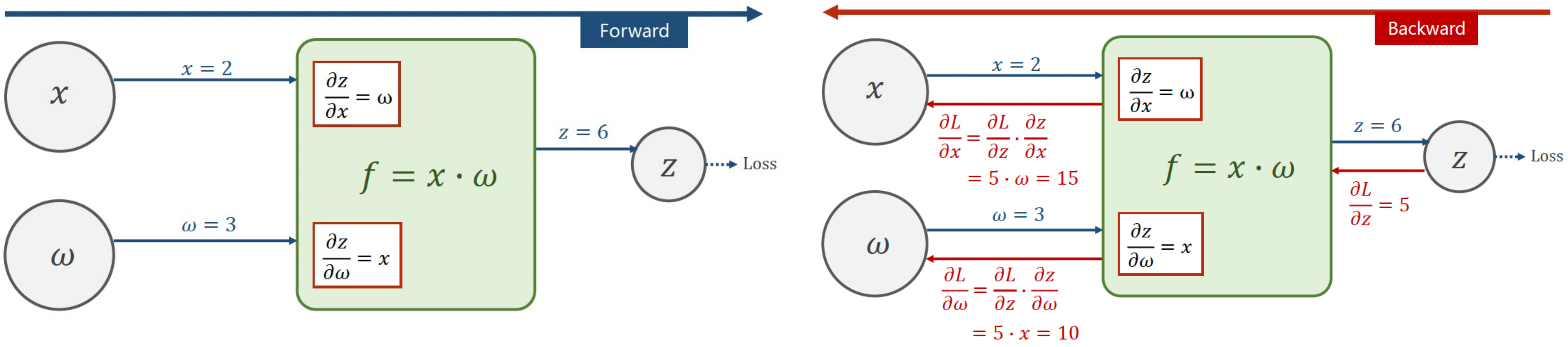
Installation

PyTorch Build	Stable (2.2.2)		Preview (Nightly)	
Your OS	Linux	Mac	Windows	
Package	Conda	Pip	LibTorch	Source
Language	Python		C++ / Java	
Compute Platform	~~CUDA 11.8~~	~~CUDA 12.1~~	~~ROCm 5.7~~	Default
Run this Command:	`pip3 install torch torchvision torchaudio`			

Installation

```
zsq — python — python — python — 80×24
Last login: Wed Mar 27 14:23:37 on ttys001
(base)
# zsq @ ZSQs-MacBook-Pro in ~ [14:20:11]
$ conda activate pytorch
(pytorch)
# zsq @ ZSQs-MacBook-Pro in ~ [14:20:21]
$ python
Python 3.9.19 | packaged by conda-forge | (main, Mar 20 2024, 12:53:33)
[Clang 16.0.6 ] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import torch
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2 (Intel(R) S
SE4.2) enabled only processors has been deprecated. Intel oneAPI Math Kernel Lib
rary 2025.0 will require Intel(R) Advanced Vector Extensions (Intel(R) AVX) inst
ructions.
Intel MKL WARNING: Support of Intel(R) Streaming SIMD Extensions 4.2 (Intel(R) S
SE4.2) enabled only processors has been deprecated. Intel oneAPI Math Kernel Lib
rary 2025.0 will require Intel(R) Advanced Vector Extensions (Intel(R) AVX) inst
ructions.
>>> print(torch.__version__)
2.2.2
>>>
```

Back Propagation

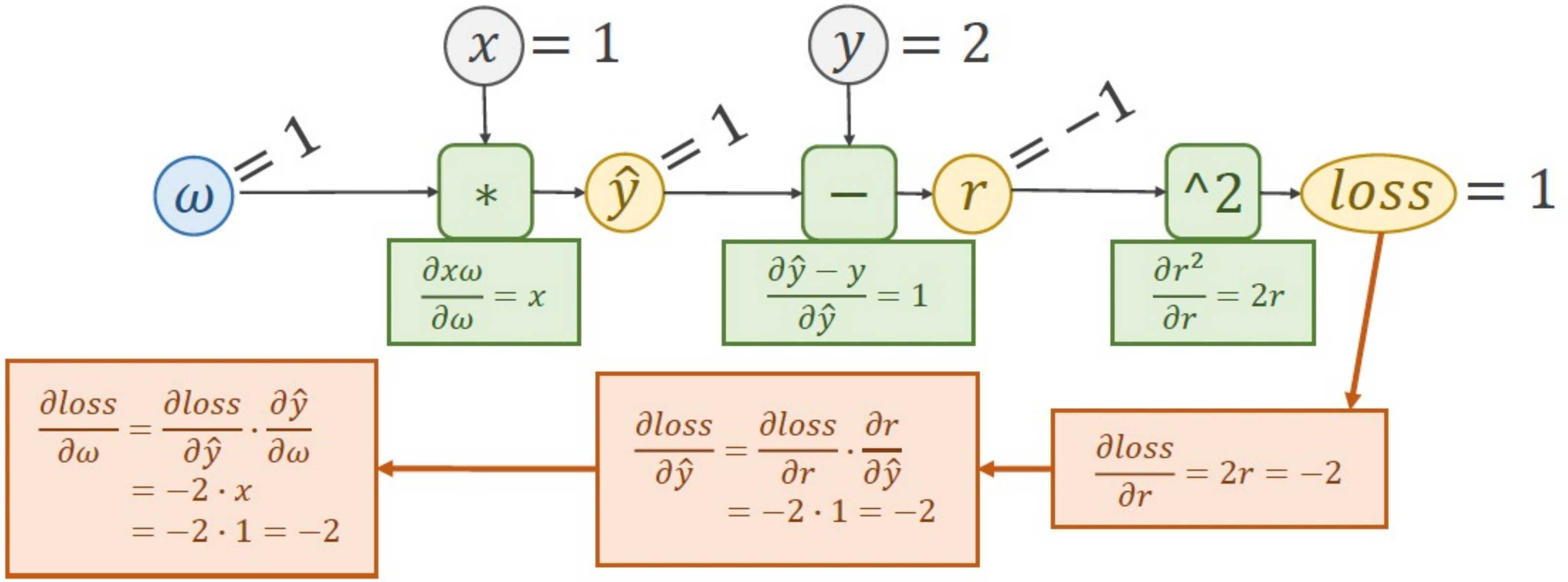


Basic Linear

Linear Model
$\hat{y} = x * \omega$

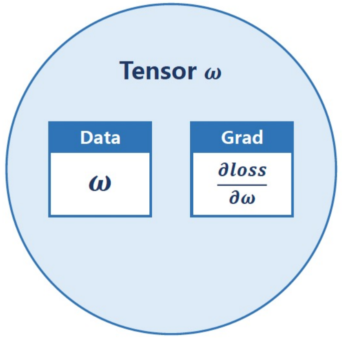
Loss Function
$loss = (\hat{y} - y)^2 = (x \cdot \omega - y)^2$

x (hours)	y (points)
1	2
2	4
3	6
4	?



Basic Linear

- Tensor
  - **data** and **grad**




```
x_data = [1.0, 2.0, 3.0]
y_data = [2.0, 4.0, 6.0]

w = torch.Tensor([1.0])
w.requires_grad = True
```

  - AutoGrad mechanics
  - strong GPU acceleration
- Model Definition

Linear Model

$$\hat{y} = x * \omega$$

Loss Function

$$loss = (\hat{y} - y)^2 = (x \cdot \omega - y)^2$$

```
def forward(x):
    return x * w


def loss(x, y):
    y_pred = forward(x)
    return (y_pred - y) ** 2
```

Basic Linear

- Train
  - forward to compute the loss
  - backward to compute grad
  - update weight
  - **RESET** grad!!!

```
print("predict (before training)", 4, forward(4).item())
for epoch in range(100):
    for x, y in zip(x_data, y_data):
        l = loss(x, y)
        l.backward()
        print("\tgrad:", x, y, w.grad.item())
        w.data = w.data - 0.01 * w.grad.data
        w.grad.data.zero_()
    print("epoch:", epoch, "loss:", l.item())
print("predict (after training)", 4, forward(4).item())
```

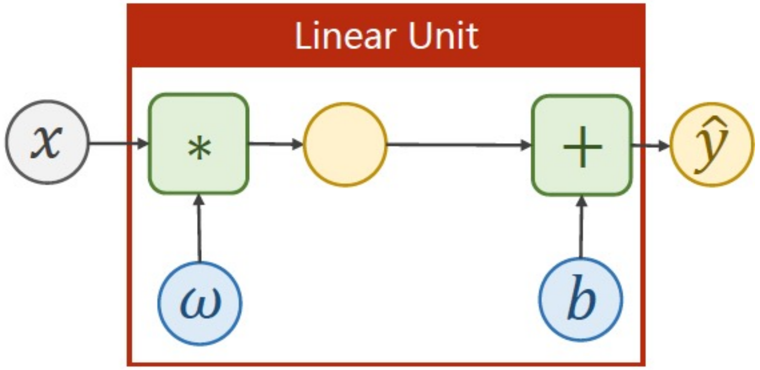
Basic Linear

```
predict (before training) 4 4.0
    grad: 1.0 2.0 -2.0
    grad: 2.0 4.0 -7.840000152587891
    grad: 3.0 6.0 -16.228801727294922
epoch: 0 loss: 7.315943717956543
    grad: 1.0 2.0 -1.478623867034912
    grad: 2.0 4.0 -5.796205520629883
    grad: 3.0 6.0 -11.998146057128906
epoch: 1 loss: 3.9987640380859375
    grad: 1.0 2.0 -1.0931644439697266
    grad: 2.0 4.0 -4.285204887390137
    grad: 3.0 6.0 -8.870372772216797
epoch: 2 loss: 2.1856532096862793
    grad: 1.0 2.0 -0.8081896305084229
    grad: 2.0 4.0 -3.1681032180786133
    grad: 3.0 6.0 -6.557973861694336
epoch: 3 loss: 1.1946394443511963
```

```
epoch: 95 loss: 9.094947017729282e-13
    grad: 1.0 2.0 -7.152557373046875e-07
    grad: 2.0 4.0 -2.86102294921875e-06
    grad: 3.0 6.0 -5.7220458984375e-06
epoch: 96 loss: 9.094947017729282e-13
    grad: 1.0 2.0 -7.152557373046875e-07
    grad: 2.0 4.0 -2.86102294921875e-06
    grad: 3.0 6.0 -5.7220458984375e-06
epoch: 97 loss: 9.094947017729282e-13
    grad: 1.0 2.0 -7.152557373046875e-07
    grad: 2.0 4.0 -2.86102294921875e-06
    grad: 3.0 6.0 -5.7220458984375e-06
epoch: 98 loss: 9.094947017729282e-13
    grad: 1.0 2.0 -7.152557373046875e-07
    grad: 2.0 4.0 -2.86102294921875e-06
    grad: 3.0 6.0 -5.7220458984375e-06
epoch: 99 loss: 9.094947017729282e-13
predict (after training) 4 7.999998569488525
```

Linear Model

- nn.Module
  - base class for all neural network modules
  - must implement these two functions
- nn.Linear




```python
class LinearModel(torch.nn.Module):
    def __init__(self):
        super(LinearModel, self).__init__()
        self.linear = torch.nn.Linear( in_features: 1,  out_features: 1)

    def forward(self, x):
        y_pred = self.linear(x)
        return y_pred
```

阅读器模式

```python
class Linear(Module):
    r"""Applies a linear transformation to the incoming data: :math:`y = xA^T + b`.

    This module supports :ref:`TensorFloat32<tf32_on_ampere>`.

    On certain ROCm devices, when using float16 inputs this module will use :ref:`different precision<fp16_on_mi200>

    Args:
        in_features: size of each input sample
        out_features: size of each output sample
        bias: If set to ``False``, the layer will not learn an additive bias.
            Default: ``True``

    Shape:
        - Input: :math:`(*, H_{in})` where :math:`*` means any number of
          dimensions including none and :math:`H_{in} = \text{in\_features}`.
        - Output: :math:`(*, H_{out})` where all but the last dimension
          are the same shape as the input and :math:`H_{out} = \text{out\_features}`.

    Attributes:
        weight: the learnable weights of the module of shape
            :math:`(\text{out\_features}, \text{in\_features})`. The values are
            initialized from :math:`\mathcal{U}(-\sqrt{k}, \sqrt{k})`, where
            :math:`k = \frac{1}{\text{in\_features}}`
        bias:   the learnable bias of the module of shape :math:`(\text{out\_features})`.
                If :attr:`bias` is ``True``, the values are initialized from
                :math:`\mathcal{U}(-\sqrt{k}, \sqrt{k})` where
                :math:`k = \frac{1}{\text{in\_features}}`

    Examples::

        >>> m = nn.Linear(20, 30)
        >>> input = torch.randn(128, 20)
        >>> output = m(input)
        >>> print(output.size())
        torch.Size([128, 30])
    """
```

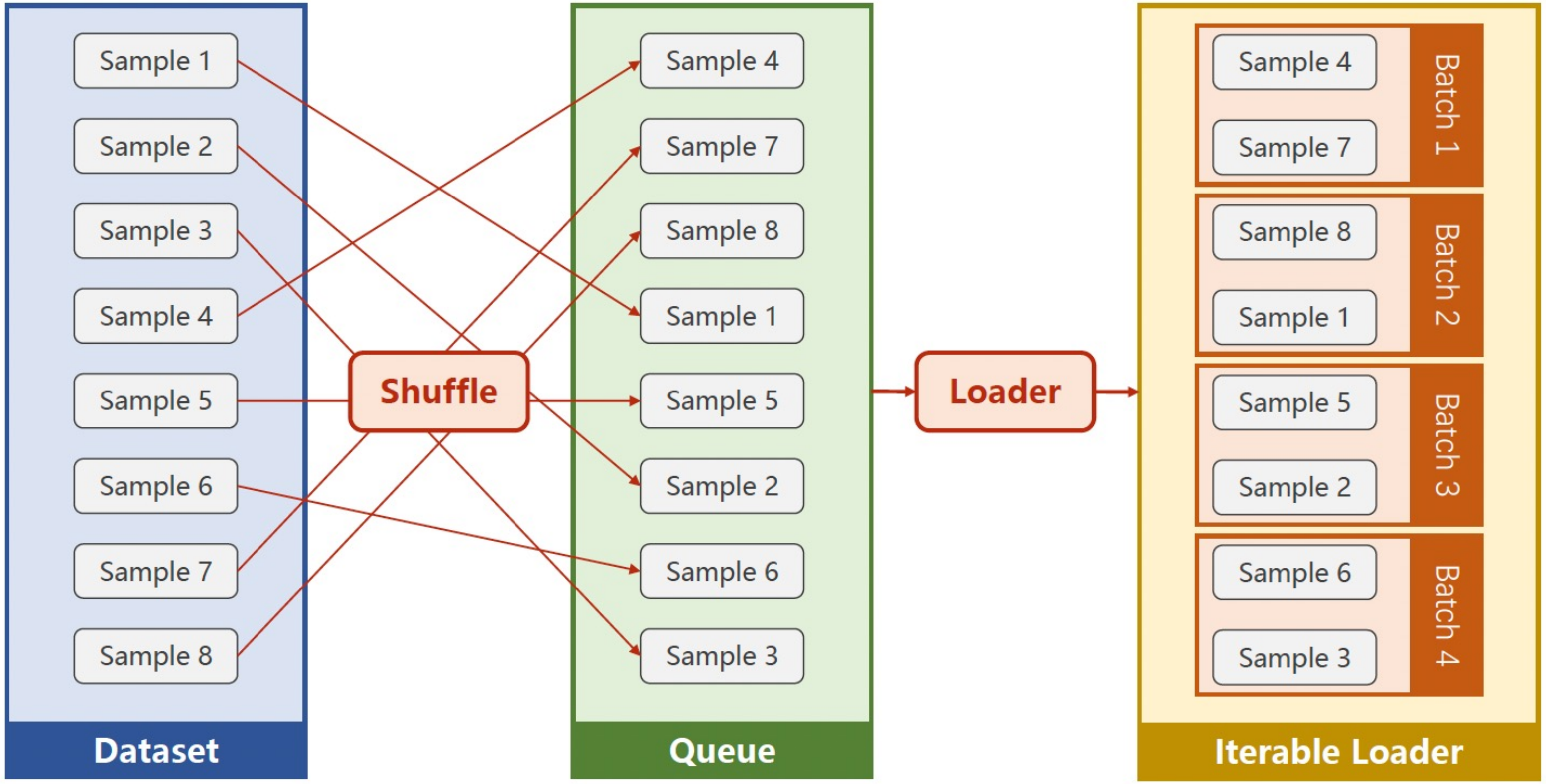
Linear Model

- Train
  - prepare dataset
  - construct model and create an instance
  - choose loss and optimizer
  - **forward to compute the loss**
  - **RESET grad!!!**
  - **backward to compute grad**
  - **update weight**

```
x_data = torch.Tensor([[1.0], [2.0], [3.0]])
y_data = torch.Tensor([[2.0], [4.0], [6.0]])
model = LinearModel()

criterion = torch.nn.MSELoss(size_average=False)
optimizer = torch.optim.SGD(model.parameters(), lr=0.01)
for epoch in range(100):
    y_pred = model(x_data)
    loss = criterion(y_pred, y_data)
    print("epoch:", epoch, "loss:", loss.item())
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
print("w:", model.linear.weight.item())
print("b:", model.linear.bias.item())
y_test = model(torch.Tensor([4.0]))
print("y_test:", y_test.item())
```

Dataset

- DataLoader: batch_size=2, shuffle=True



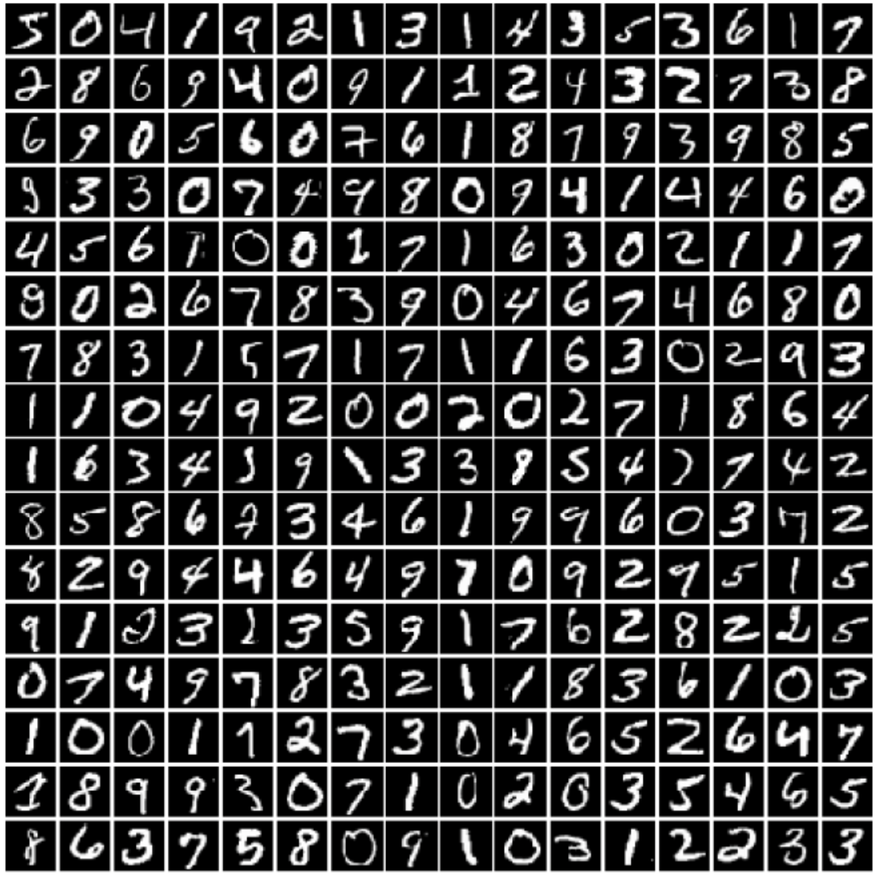
Dataset

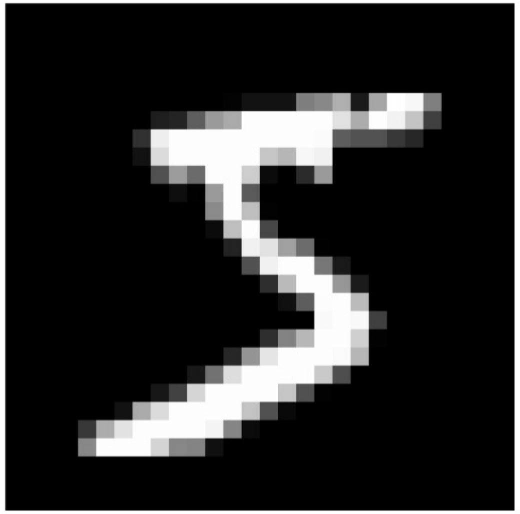
- Dataset
  - abstract class
  - must implement these three functions
  - use with Dataloader

```python
import torch
from torch.utils.data import Dataset
from torch.utils.data import DataLoader


1 个用法
class RandomDataset(Dataset):
    def __init__(self):
        self.x = torch.randn((128, 64))
        self.y = torch.randn((128, 1))

    def __getitem__(self, idx):
        return self.x[idx], self.y[idx]

    def __len__(self):
        return self.x.shape[0]


if __name__ == '__main__':
    dataset = RandomDataset()
    train_loader = DataLoader(dataset=dataset, batch_size=32, shuffle=True)
    for epoch in range(100):
        for i, (x, y) in enumerate(train_loader):
            pass
```

MLP

- MINIST





$28 * 28 = 784$

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	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	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	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	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	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	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	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	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	0.1	0.1	0.1	0.5	0.5	0.7	0.1	0.7	1.0	1.0	0.5	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.1	0.4	0.6	0.7	1.0	1.0	1.0	1.0	1.0	0.9	0.7	1.0	1.0	0.8	0.3	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.2	0.9	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.4	0.3	0.3	0.2	0.2	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.9	1.0	1.0	1.0	1.0	1.0	0.8	0.7	1.0	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	0.0	0.0	0.0	0.3	0.6	0.4	1.0	1.0	0.8	0.0	0.0	0.2	0.6	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	0.0	0.0	0.1	0.0	0.6	1.0	0.4	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	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.6	1.0	0.8	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	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.8	1.0	0.3	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	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	1.0	0.9	0.6	0.4	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	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.3	0.9	1.0	1.0	0.5	0.1	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	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.2	0.7	1.0	1.0	0.6	0.1	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	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1	0.4	1.0	1.0	0.7	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	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	0.3	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	0.0	0.0	0.0	0.0	0.2	0.5	0.7	1.0	1.0	0.8	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	0.0	0.0	0.0	0.2	0.6	0.9	1.0	1.0	1.0	1.0	0.7	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	0.0	0.1	0.5	0.9	1.0	1.0	1.0	1.0	0.8	0.3	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	0.1	0.3	0.8	1.0	1.0	1.0	1.0	0.8	0.3	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	0.1	0.7	0.9	1.0	1.0	1.0	1.0	1.0	0.8	0.3	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	0.2	0.7	0.9	1.0	1.0	1.0	1.0	1.0	0.5	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	0.0	0.0	0.5	1.0	1.0	1.0	0.8	0.5	0.5	0.1	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
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	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	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	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	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

MLP

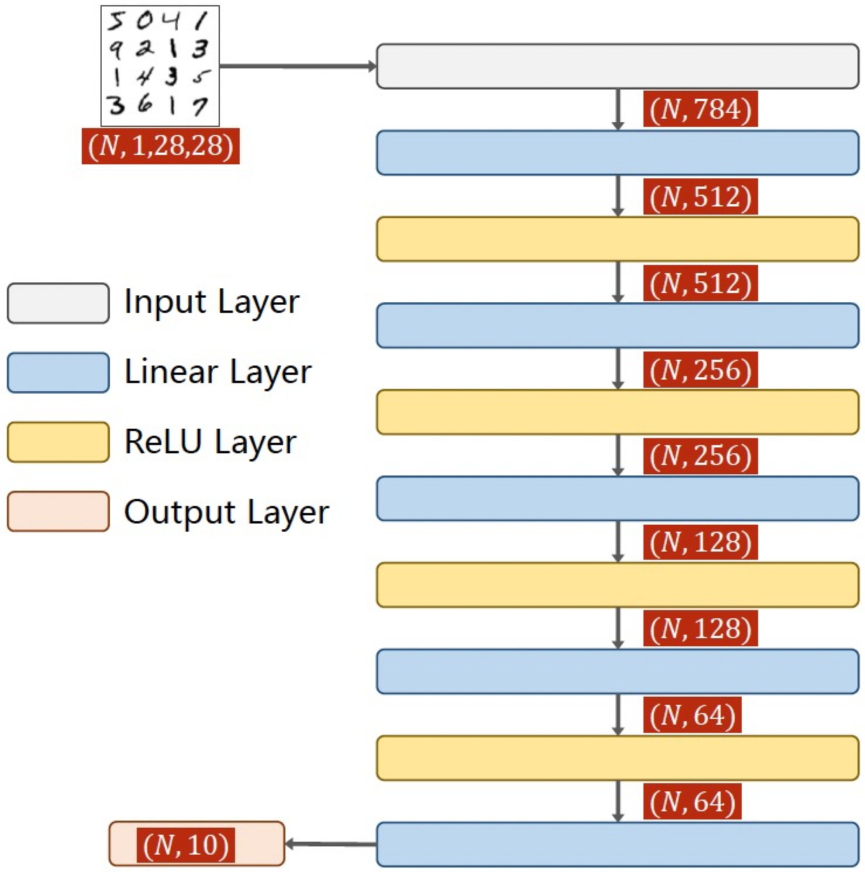
- Prepare dataset
    - auto download
    - preprocess
    - train 60000 and test 10000

```python
from torchvision import transforms
from torchvision import datasets
from torch.utils.data import DataLoader
batch_size = 64
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize( mean: (0.1307,), std: (0.3081,))])

train_dataset = datasets.MNIST(root='./dataset/minist', train=True, download=True, transform=transform)
train_loader = DataLoader(train_dataset, shuffle=True, batch_size=batch_size)
test_dataset = datasets.MNIST(root='./dataset/minist', train=False, download=True, transform=transform)
test_loader = DataLoader(test_dataset, shuffle=True, batch_size=batch_size)
```

MLP

- Construct MLP model
  - choose hyperparameters at will
  - output dimensions(10) for softmax

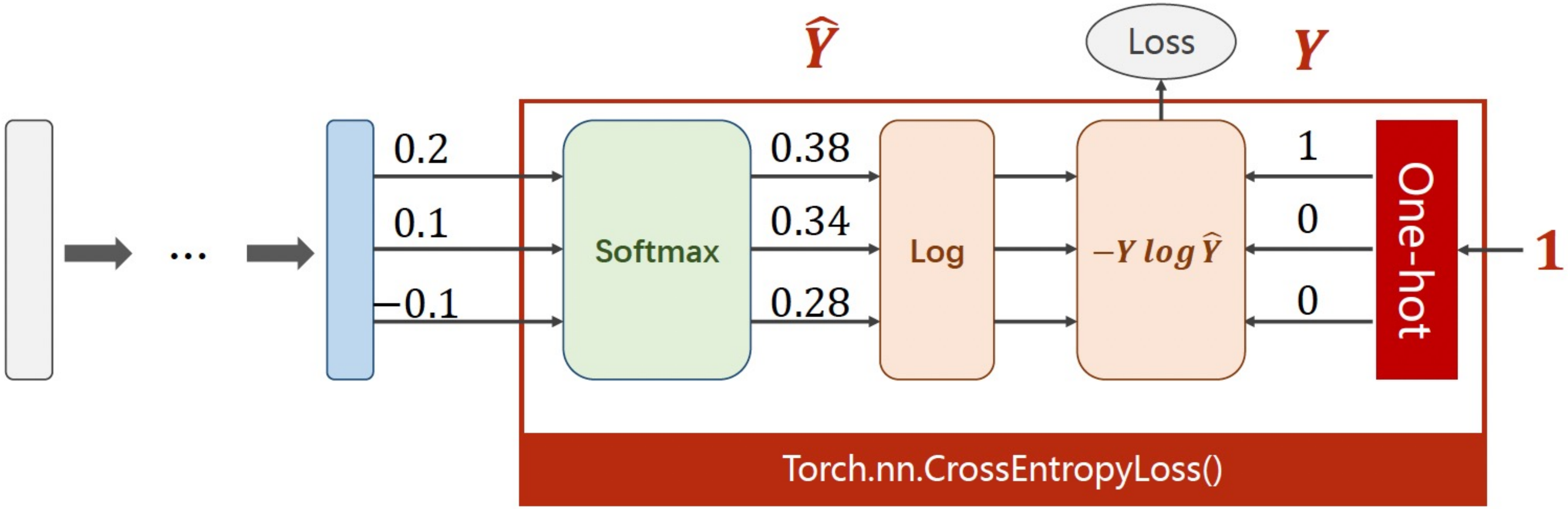
```
import torch.nn.functional as F
class MLP(torch.nn.Module):
    def __self__(self):
        super(MLP, self).__init__()
        self.l1 = torch.nn.Linear( in_features: 784,  out_features: 512)
        self.l2 = torch.nn.Linear( in_features: 512,  out_features: 256)
        self.l3 = torch.nn.Linear( in_features: 256,  out_features: 128)
        self.l4 = torch.nn.Linear( in_features: 128,  out_features: 64)
        self.l5 = torch.nn.Linear( in_features: 64,  out_features: 10)

    def forward(self, x):
        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)
```



MLP

- Create model instance
- choose loss and optimizer

```
model = MLP()
criterion = torch.nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.5)
```



MLP

- Train and test
  - 2 min for 10 epochs

```
[epoch 8, batch 299] loss: 0.37047654286026954
[epoch 8, batch 599] loss: 0.3695543259382248
[epoch 8, batch 899] loss: 0.35767048348983127
Accuracy on test set: 90.38%%
[epoch 9, batch 299] loss: 0.34093401715159416
[epoch 9, batch 599] loss: 0.3417566152910391
[epoch 9, batch 899] loss: 0.32475153950353464
Accuracy on test set: 91.05%%
```

```python
def train(epoch, train_loader, model, criterion, optimizer):
    running_loss = 0.0
    for batch_idx, (x, y) in enumerate(train_loader, 0):
        optimizer.zero_grad()
        outputs = model(x)
        loss = criterion(outputs, y)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
        if batch_idx % 300 == 299:
            print(f"[epoch {epoch}, batch {batch_idx}] loss: {running_loss / 300}")
            running_loss = 0.0
def test(test_loader, model):
    correct = 0
    total = 0
    with torch.no_grad():
        for (x, y) in test_loader:
            outputs = model(x)
            _, predicted = torch.max(outputs.data, dim=1)
            total += y.size(0)
            correct += (predicted == y).sum().item()
    print(f"Accuracy on test set: {100 * correct / total}%%")
```

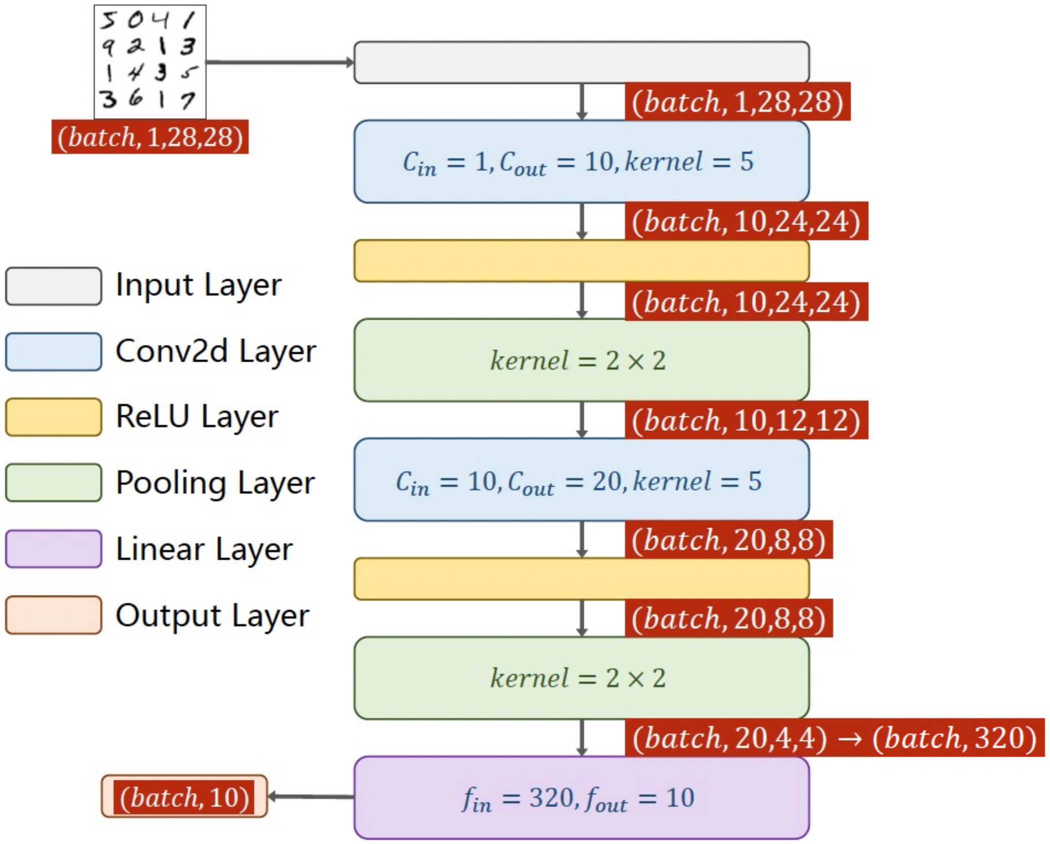
MLP

- Better coding habits
  - detailed comments and elegant output
  - decoupling
  - use tqdm to visualize

```python
if __name__ == '__main__':
    print("Loading data...")
    train_dataset = datasets.MNIST(root='./dataset/minist', train=True, download=True, transform=transform)
    train_loader = DataLoader(train_dataset, shuffle=True, batch_size=batch_size)
    test_dataset = datasets.MNIST(root='./dataset/minist', train=False, download=True, transform=transform)
    test_loader = DataLoader(test_dataset, shuffle=True, batch_size=batch_size)

    print("Building model...")
    model = MLP()
    criterion = torch.nn.CrossEntropyLoss()
    optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.5)

    print("Training...")
    for epoch in range(10):
        train(epoch, train_loader, model, criterion, optimizer)
        test(test_loader, model)
```

CNN

```python
class CNN(torch.nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = torch.nn.Conv2d( in_channels: 1,  out_channels: 10, kernel_size=5)
        self.conv2 = torch.nn.Conv2d( in_channels: 10,  out_channels: 20, kernel_size=5)
        self.pooling = torch.nn.MaxPool2d(2)
        self.fc = torch.nn.Linear( in_features: 320,  out_features: 10)

    def forward(self, x):
        # Flatten data from (batch_size, 1, 28, 28) to (batch_size, 784)
        batch_size = x.size(0)
        x = F.relu(self.pooling(self.conv1(x)))
        x = F.relu(self.pooling(self.conv2(x)))
        x = x.view(batch_size, -1)  # flatten
        x = self.fc(x)
        return x
```



Use GPU

- Define device
- Move model to GPU

```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print(device)
model = CNN().to(device)
```

- Send the inputs and targets at every step to the GPU.
  - x, y = x.to(device), y.to(device)

- In the cloud
  - run in Microsoft Learn
  - run in Google Colab

Thanks