Introduction to Deep Learning (I2DL)

Exercise 7: Pytorch

Deep Learning Frameworks

The two big ones

- Tensorflow - Google
  - As well as Keras
- Pytorch – Facebook

Other examples

- CNTK – Microsoft
- Mxnet - Apache

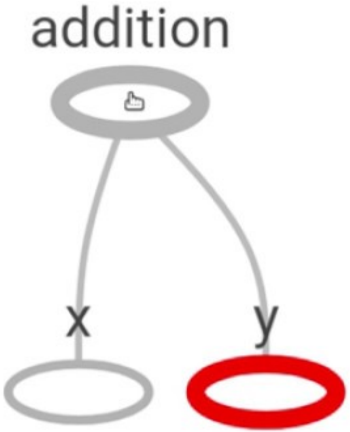
Static Framework: Tensorflow

Network Declaration

```
N, D, H = 64, 1000, 100

x = tf.placeholder(tf.float32, shape=(N, D))
y = tf.placeholder(tf.float32, shape=(N, D))
w1 = tf.Variable(tf.random_normal((D, H)))
w2 = tf.Variable(tf.random_normal((H, D)))

h = tf.maximum(tf.matmul(x, w1), 0)
y_pred = tf.matmul(h, w2)
diff = y_pred - y
loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, a
grad_w1, grad_w2 = tf.gradients(loss, [w1, w2])

learning_rate = 1e-5
new_w1 = w1.assign(w1 - learning_rate * grad_w1)
new_w2 = w2.assign(w2 - learning_rate * grad_w2)
updates = tf.group(new_w1, new_w2)
```

Iteration

```
with tf.Session() as sess:
  sess.run(tf.global_variables_initializer())
  values = {x: np.random.randn(N, D),
            y: np.random.randn(N, D),}
  losses = []
  for t in range(50):
    loss_val, _ = sess.run([loss, updates], feed_dict=values)
```



What if you want to change the learning rate?

Dynamic Framework: Pytorch

Network Declaration

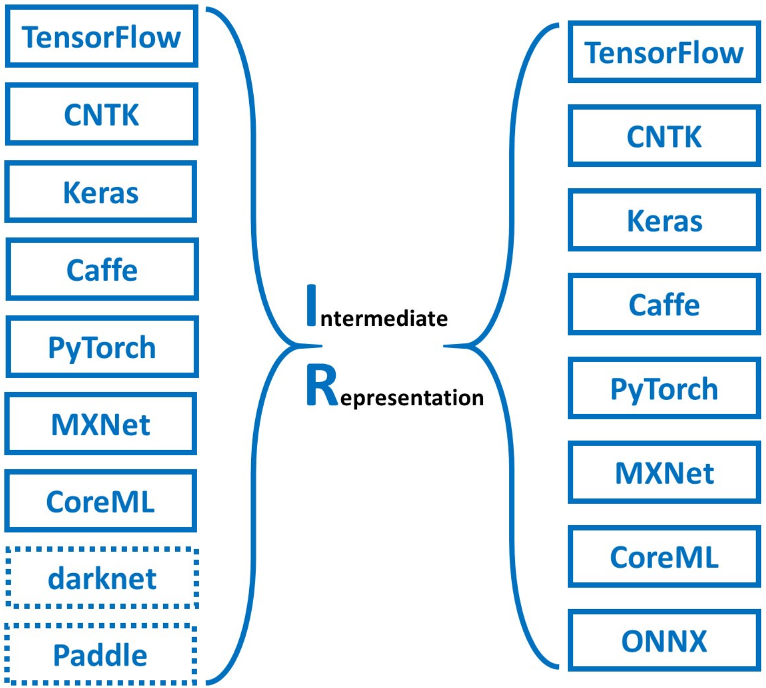
```
from torch.autograd import Variable

N, D_in, H, D_out = 64, 1000, 100, 10

x = Variable(torch.randn(N, D_in).cuda(),
y = Variable(torch.randn(N, D_out).cuda(),
w1 = Variable(torch.randn(D_in, H).cuda(),
w2 = Variable(torch.randn(H, D_out).cuda()
```

Iteration

```
learning_rate = 1e-6
for t in range(500):
    y_pred = x.mm(w1).clamp(min=0).mm(w2)
    loss = (y_pred - y).pow(2).sum()

    if w1.grad: w1.grad.data.zero_()
    if w2.grad: w2.grad.data.zero_()
    loss.backward()

    w1.data -= learning_rate * w1.grad.data
    w2.data -= learning_rate * w2.grad.data
```

-> Advantages/Disadvantages

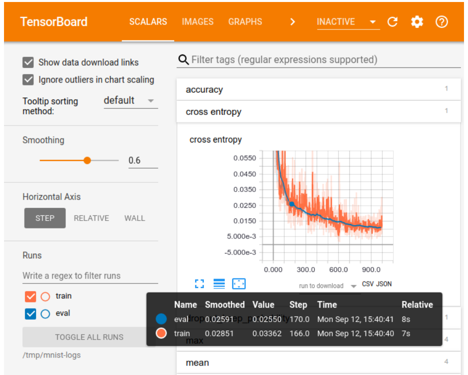
Framework Conversion



See: https://github.com/microsoft/MMdnn

Today's Contents

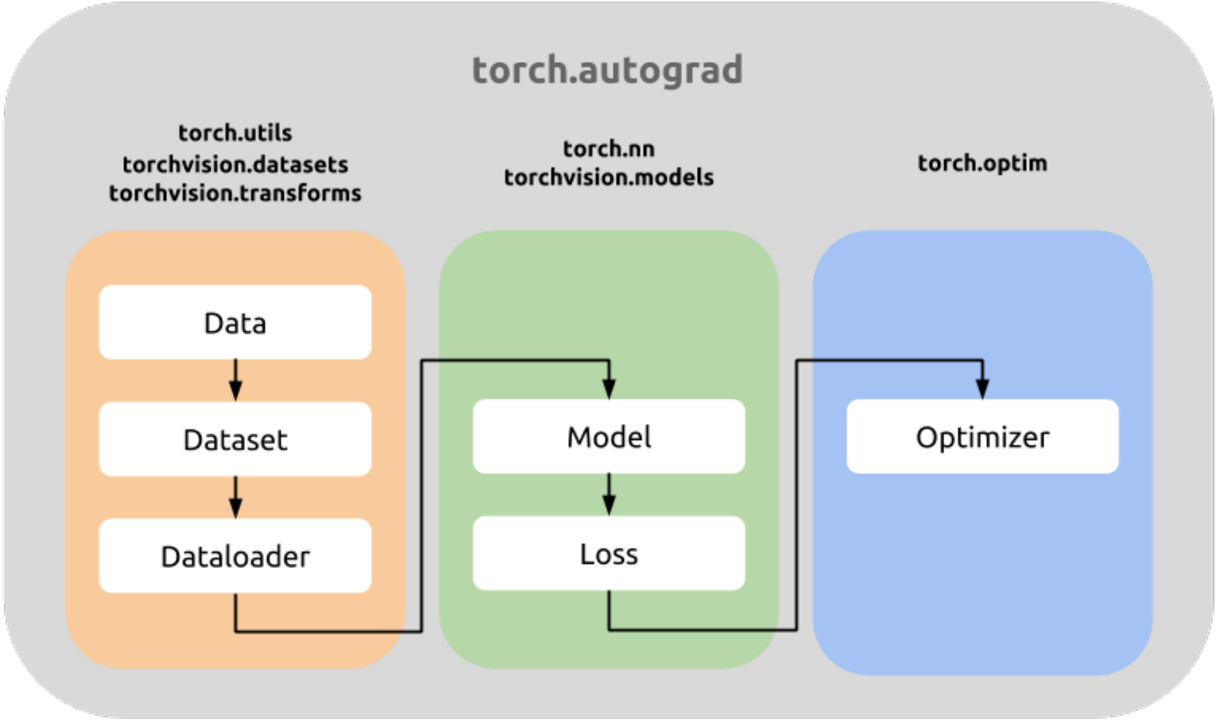
- Pytorch
  - Networks & Data
- Tensorboard
  - Easy visualization on the fly
  - Part of Tensorflow, but also accessible in Pytorch
- Pytorch Lightning
  - Solver



No submission
;-)

Pytorch

Pytorch: Overview



PyTorch vs. NumPy

- PyTorch shares lots of similarities with NumPy.
  - np.array vs. torch.Tensor

```python
a_np = np.array([[1,2,3],[5,6,7]]) #NumPy array
a_ts = torch.tensor([[1,2,3],[4,5,6]]) # Tensor
print(a_np)
print(a_ts)
```

```
[[1 2 3]
 [5 6 7]]
tensor([[1, 2, 3],
        [4, 5, 6]])
```

PyTorch vs. NumPy

- Similar Operations
  - Indexing

```
# indexing
b = a_ts[:2, :2] # use numpy type indexing
b[:, 0] = 0 # For assignment
b
```

```
tensor([[0, 2],
        [0, 5]])
```

  - Mathematical operations
  - Etc.

Easy Device Assignment

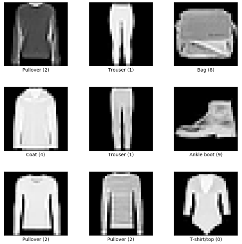
```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print(device)

print(f"Original device: {x.device}") # "cpu", integer

tensor = x.to(device)
print(f"Current device: {x.device}") #"cpu" or "cuda", double
```

```
cpu
Original device: cpu
Current device: cpu
```

Datasets: Torchvision

- Torchvision
  - torchvision.datasets contains many datasets, such as ImageNet, FashionMNIST, etc.

```
fashion_mnist_dataset = torchvision.datasets.FashionMNIST(root='../datasets',
                                                          train=True,
                                                          download=True,
                                                          transform=transform)

fashion_mnist_dataloader = DataLoader(fashion_mnist_dataset, batch_size=8)
```

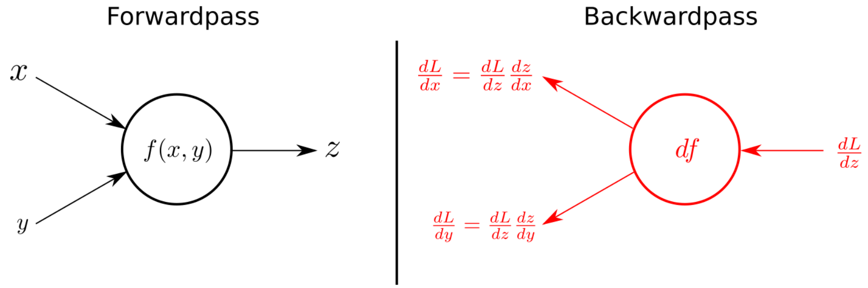


Source: https://www.tensorflow.org/datasets/catalog/fashion_mnist

Easy Network Creation

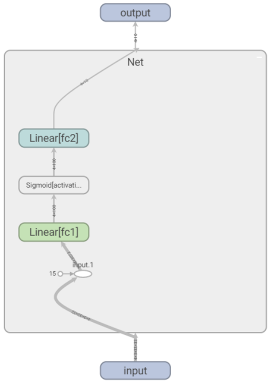
```
import torch.nn as nn
# defining the model
class Net(nn.Module):
    def __init__(self, input_size=1*28*28, output_size=100):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(input_size, output_siz

    def forward(self, x):
        x = self.fc1(x)
        return x
net = Net()
net = net.to(device)
```

Where is the backward pass?



Training Loop

zero_grad()	zero the gradient
y_pred = net(X)	complete a forward pass
loss = criterion(y_pred, y)	calculate the loss
loss.backward()	backpropagation
optimizer.step()	apply gradients




```
# zero the parameter gradients
optimizer.zero_grad()

# forward + backward + optimize
y_pred = net(X) # input x and predict based on x
loss = criterion(y_pred, y) # calculate the loss
loss.backward()  # backpropagation, compute gradients
optimizer.step()  # apply gradients
```

References on Pytorch

- Documentation(1.4.0): https://tutorials.pytorch.kr/beginner/nlp/pytorch_tutorial.html
- Repository: https://github.com/pytorch/pytorch
- Examples (very nice): https://github.com/pytorch/examples
- PyTorch for NumPy users: https://github.com/wkentaro/pytorch-for-numpy-users

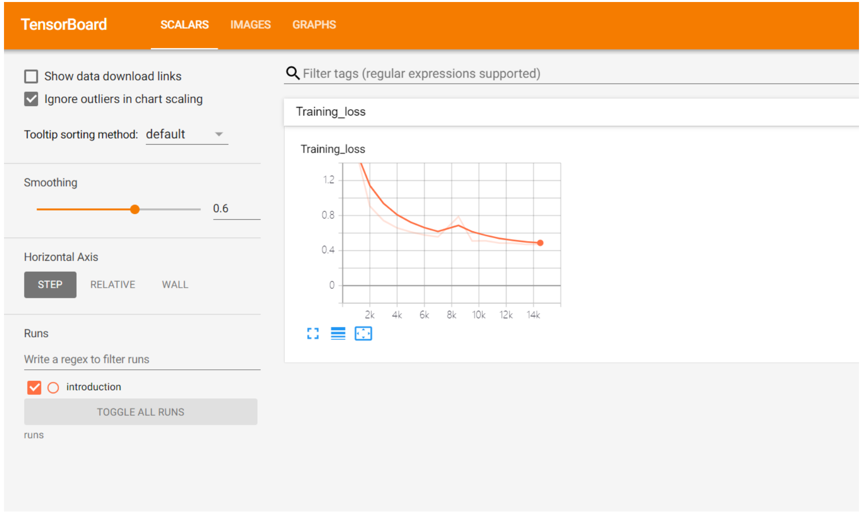
Tensorboard

Simple Logging

- Directly access tensorboard in your training loop

```python
# ...log the running loss
writer.add_scalar('Training loss',
                  running_loss / 1000,
                  epoch * len(trainloader) + i)
```

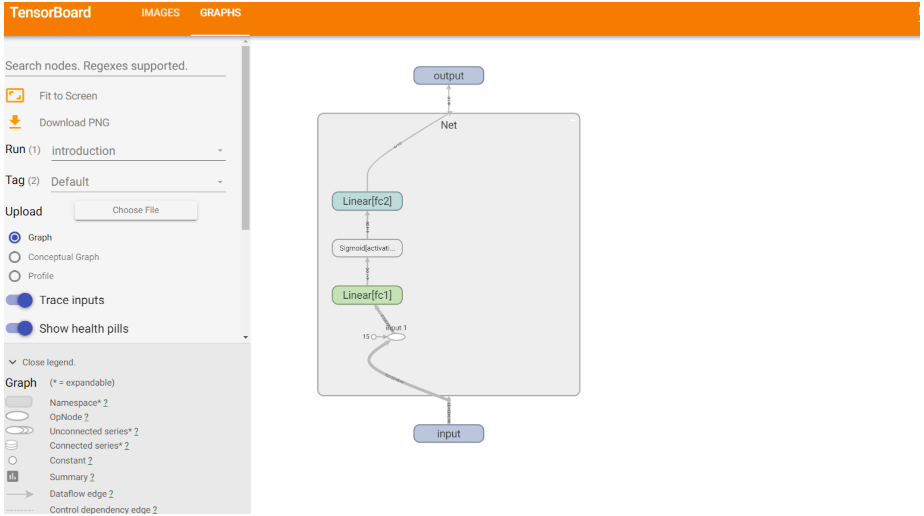
- Tensorboard generates the graph/timestamps etc. for you



Visualize Network Architectures

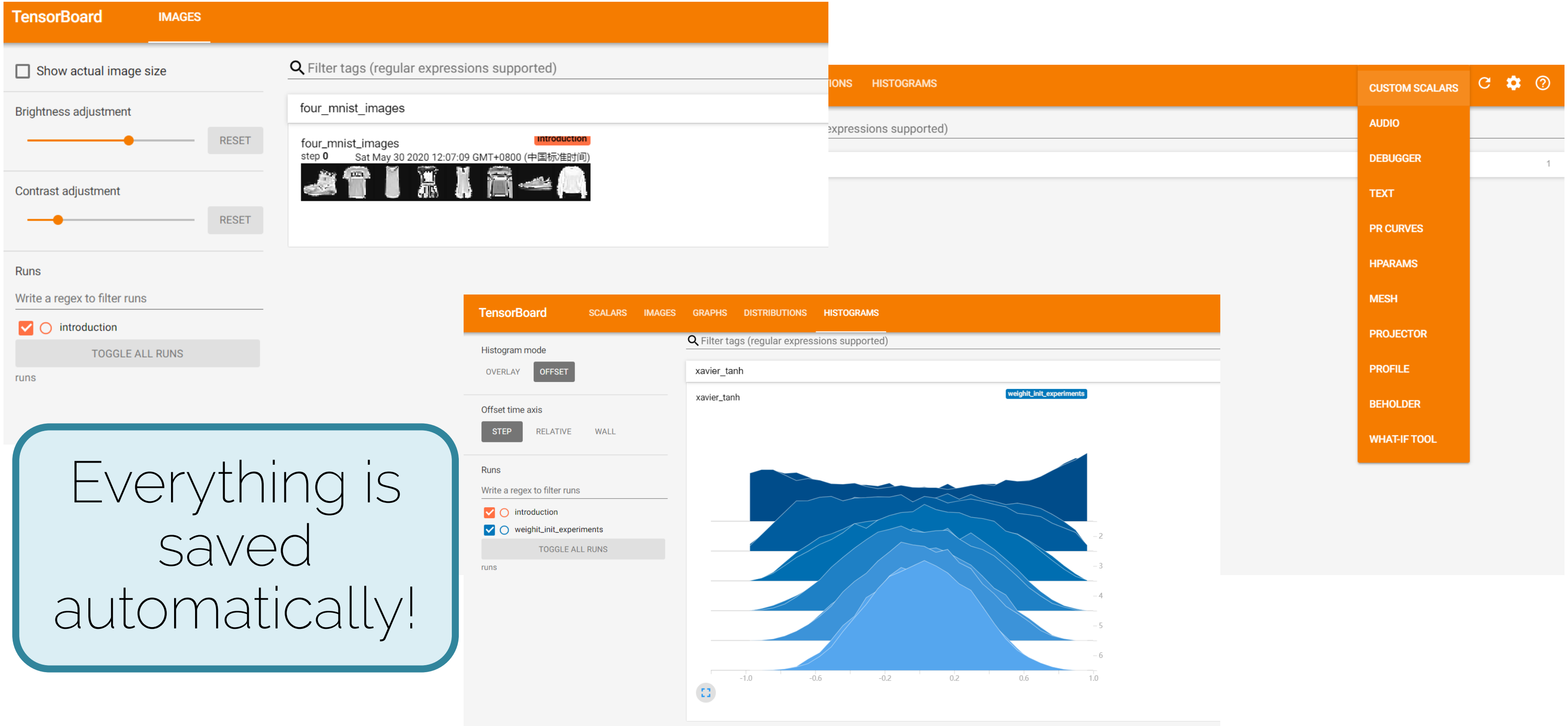
- Using a single forward pass, tensorflow can map and display your network graph

```python
# Initialize your model
net = Net()

# Visualize its architecture in TensorBoard
writer.add_graph(net, images)
```



Graph creation needs network & one batch!

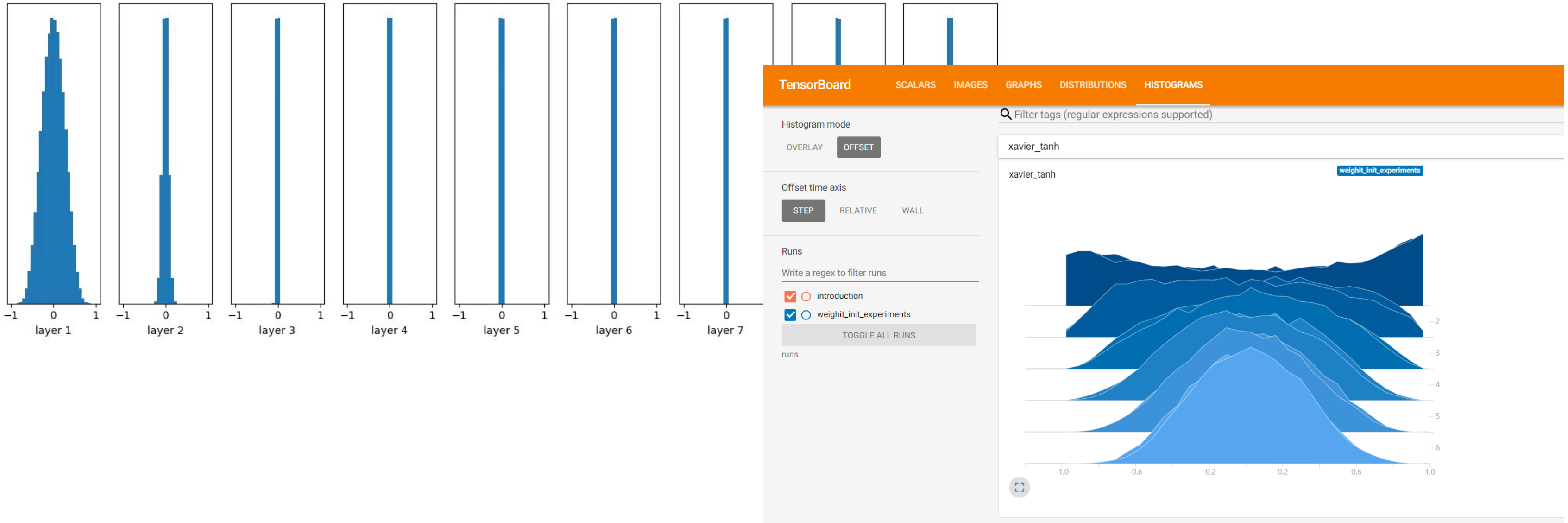
Document Everything!



Everything is saved automatically!

Weight Initialization

- Histogram visualization for layer outputs can show off effects of weight initialization as shown in the lecture



PyTorch Lightning

Idea Behind PyTorch Lightning

Classify our code into three categories

1. **Research** code (the exciting part!, changes with new tasks, models etc.) → LightningModule
2. **Engineering** code (the same for all projects and models) → Trainer
3. **Non-essential** code (logging, organizing runs → Callbacks

Lightning Module

PyTorch

```
# model
class Net(nn.Module):
  def __init__(self):
      self.layer_1 = torch.nn.Linear(28 * 28, 128)
      self.layer_2 = torch.nn.Linear(128, 10)

  def forward(self, x):
    x = x.view(x.size(0), -1)
    x = self.layer_1(x)
    x = F.relu(x)
    x = self.layer_2(x)
    return x

# train loader
mnist_train = MNIST(os.getcwd(), train=True, download=True,
                    transform=transforms.ToTensor())
mnist_train = DataLoader(mnist_train, batch_size=64)

net = Net()

# optimizer + scheduler
optimizer = torch.optim.Adam(net.parameters(), lr=1e-3)
scheduler = StepLR(optimizer, step_size=1)

# train
for epoch in range(1, 100):
   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 = F.nll_loss(output, target)

      loss.backward()
      optimizer.step()
      if batch_idx % args.log_interval == 0:
          print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}
              epoch, batch_idx * len(data), len(train_loader.dataset),
              100. * batch_idx / len(train_loader), loss.item()))
```

Methods that need to be implemented

- __init__
- forward
- training_step
- training_dataloader
- configure_optimizers

Some other methods

- validation_step
- validation_end
- prepare_data

Trainer

PyTorch

```python
# model
class Net(nn.Module):
    def __init__(self):
        self.layer_1 = torch.nn.Linear(28 * 28, 128)
        self.layer_2 = torch.nn.Linear(128, 10)

    def forward(self, x):
      x = x.view(x.size(0), -1)
      x = self.layer_1(x)
      x = F.relu(x)
      x = self.layer_2(x)
      return x

# train loader
mnist_train = MNIST(os.getcwd(), train=True, download=True,
                    transform=transforms.ToTensor())
mnist_train = DataLoader(mnist_train, batch_size=64)

net = Net()

# optimizer + scheduler
optimizer = torch.optim.Adam(net.parameters(), lr=1e-3)
scheduler = StepLR(optimizer, step_size=1)

# train
for epoch in range(1, 100):
   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 = F.nll_loss(output, target)
      loss.backward()
      optimizer.step()
      if batch_idx % args.log_interval == 0:
          print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(
              epoch, batch_idx * len(data), len(train_loader.dataset),
              100. * batch_idx / len(train_loader), loss.item()))
```

PyTorch Lightning

```python
# model
class Net(LightningModule):
  def __init__(self):
    self.layer_1 = torch.nn.Linear(28 * 28, 128)
    self.layer_2 = torch.nn.Linear(128, 10)

  def forward(self, x):
    x = x.view(x.size(0), -1)
    x = self.layer_1(x)
    x = F.relu(x)
    x = self.layer_2(x)
    return x

  def train_dataloader(self):
    mnist_train = MNIST(os.getcwd(), train=True, download=True,
                        transform=transforms.ToTensor())
    return DataLoader(mnist_train, batch_size=64)

  def configure_optimizers(self):
    optimizer = torch.optim.Adam(self.parameters(), lr=1e-3)
    scheduler = StepLR(optimizer, step_size=1)
    return optimizer, scheduler

  def training_step(self, batch, batch_idx):
    data, target = batch
    output = self.forward(data)
    loss = F.nll_loss(output, target)
    return loss
```

```python
if __name__ == '__main__':
  net = Net()

  trainer = Trainer()
  trainer.fit(net)
```

1. Initialize the model with hyperparamers for training (e.g. as a dictionary)
2. Trainer contains all code relevant for training our neural networks
3. Call the method .fit() for training the network

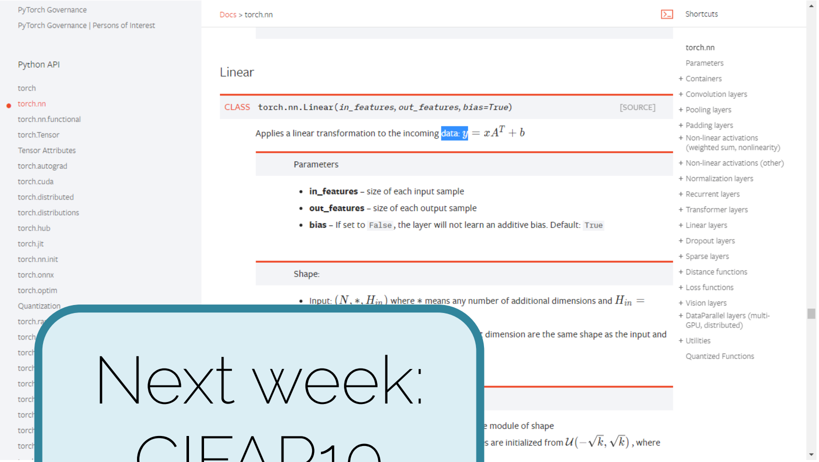
That's all you need to train you model ☺

Advantages

- Better overview of the relevant code
- Nice debugging features
- Many automated options, for example
  - Saving hyperparameters
  - Logging training/ validation results
  - Logging images with tensorboard
  - Saving models and reloading models from checkpoints

Your task for this week

- Go over all notebooks
  - There is no submission or code to implement
- Experiment yourself!
  - House prices
    1. Pytorch with own solver
    2. Pytorch lightning
  - Try out visualizations
- Check out documentations



Next week: CIFAR10 (again)

See you next week!