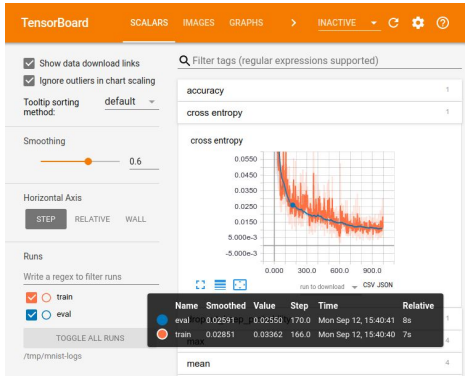
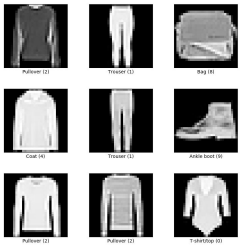
Introduction to Deep Learning (I2DL)

Exercise 7: Pytorch

Today's Outline

- Exercise 6 Recap
  - Or: why are you bad?
- Pytorch
  - And other libraries
- "Optional Submission"
  - It's good for you <3
- Organization



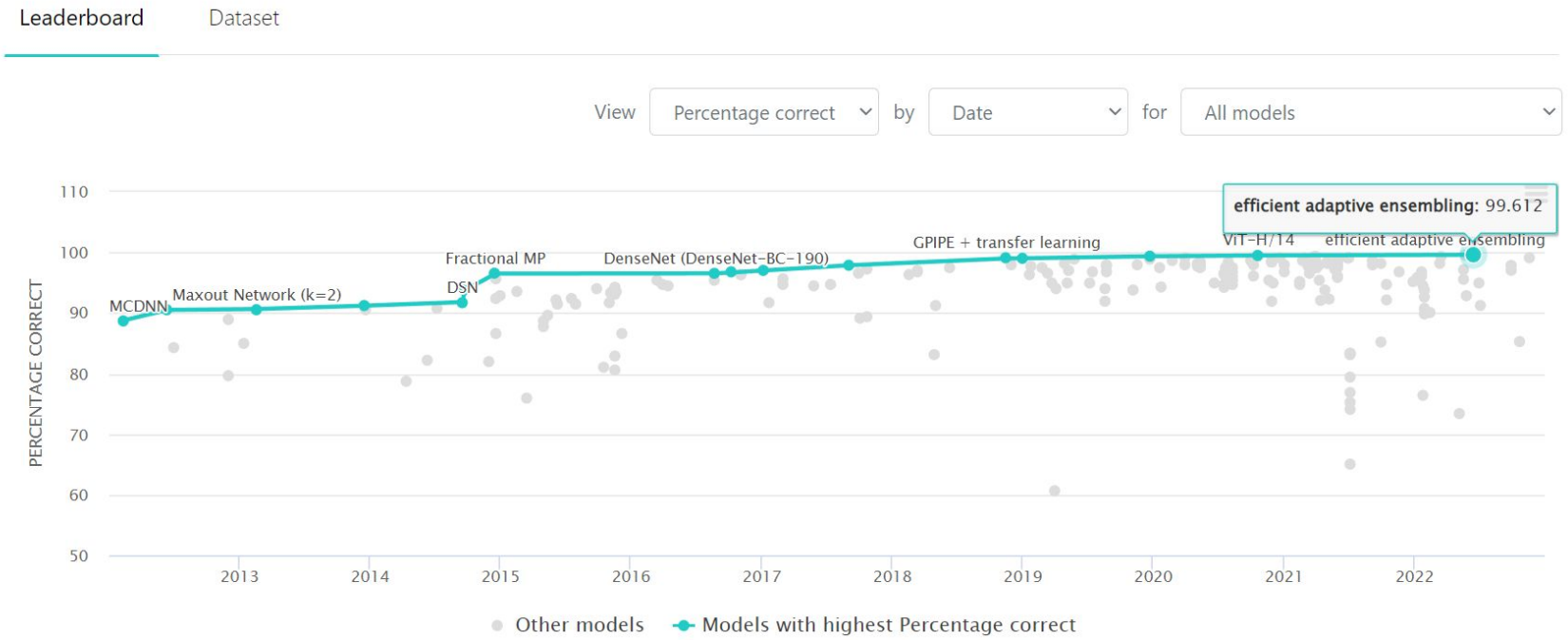


Exercise 6 Recap

Our Leaderboard

#	User	Score
1	u0093	62.11
2	u0573	60.67
3	u0808	60.36
4	u0120	60.11
5	u1163	58.34
6	u1558	57.37
7	u1068	57.02
8	u0049	56.96
9	u1378	56.80
10	u1147	56.77

Image Classification on CIFAR-10



https://paperswithcode.com/sota/image-classification-on-cifar-10

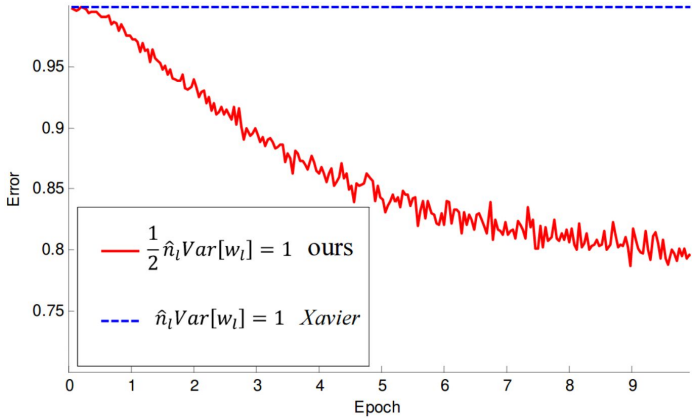
Some Limiting Factors

- Computational power and/or time → Pytorch -> GPU support
- Specialized architectures → CNNs -> Lecture 9
- More knowledge e.g., proper initialization → Lectures ☺️

Lecture Recap: Initialization

Lecture

- Network weights shouldn't only be randomly initialized
- They should be tailored to our activation function



We in exercise 6 ^^

```
self.params = {'W1': std * np.random.randn(input_size, hidden_size),
               'b1': np.zeros(hidden_size)}
```

Pytorch

Exercise Overview

Exercise 01: Organization Exercise 02: Math Recap	Intro
Exercise 03: Dataset and Dataloader Exercise 04: Solver and Linear Regression Exercise 05: Neural Networks Exercise 06: Hyperparameter Tuning	Numpy (Reinvent the wheel)
Exercise 07: Introduction to Pytorch Exercise 08: Autoencoder	Pytorch/Tensorboard
Exercise 09: Convolutional Neural Networks Exercise 10: Semantic Segmentation Exercise 11: Recurrent Neural Networks	Applications (Hands-off)

Deep Learning Frameworks

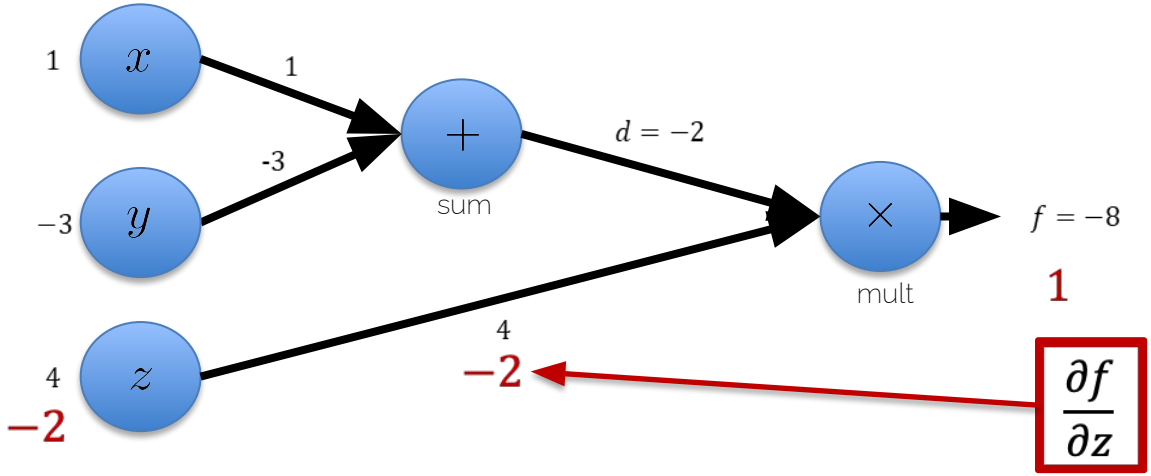
The two big ones

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

Other examples

- CNTK – Microsoft
- Mxnet – Apache
- Jax – Google
- ...

Different Paradigms

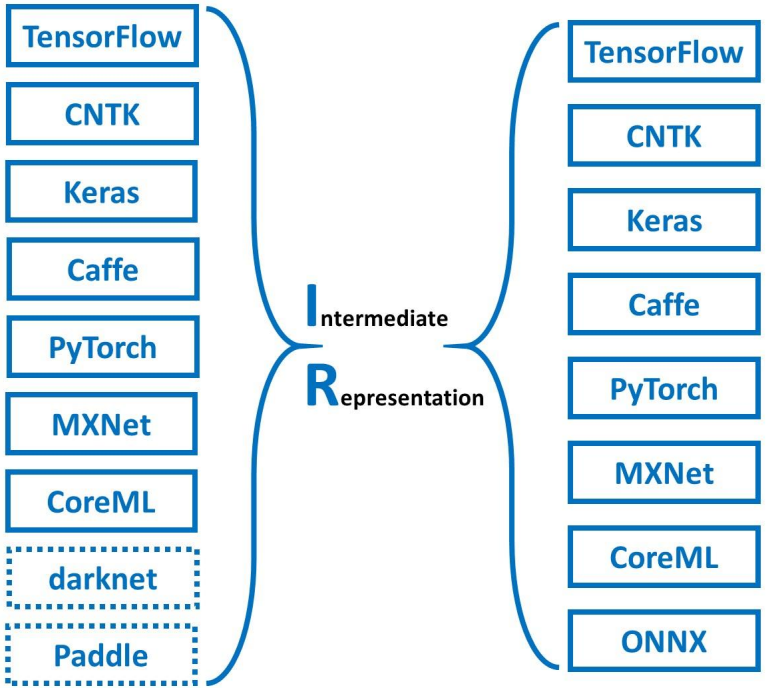


	Tensorflow	Pytorch
Graph Creation	Static/Eager	Dynamic/On Runtime
Similar to	C	Python

Framework Conversion

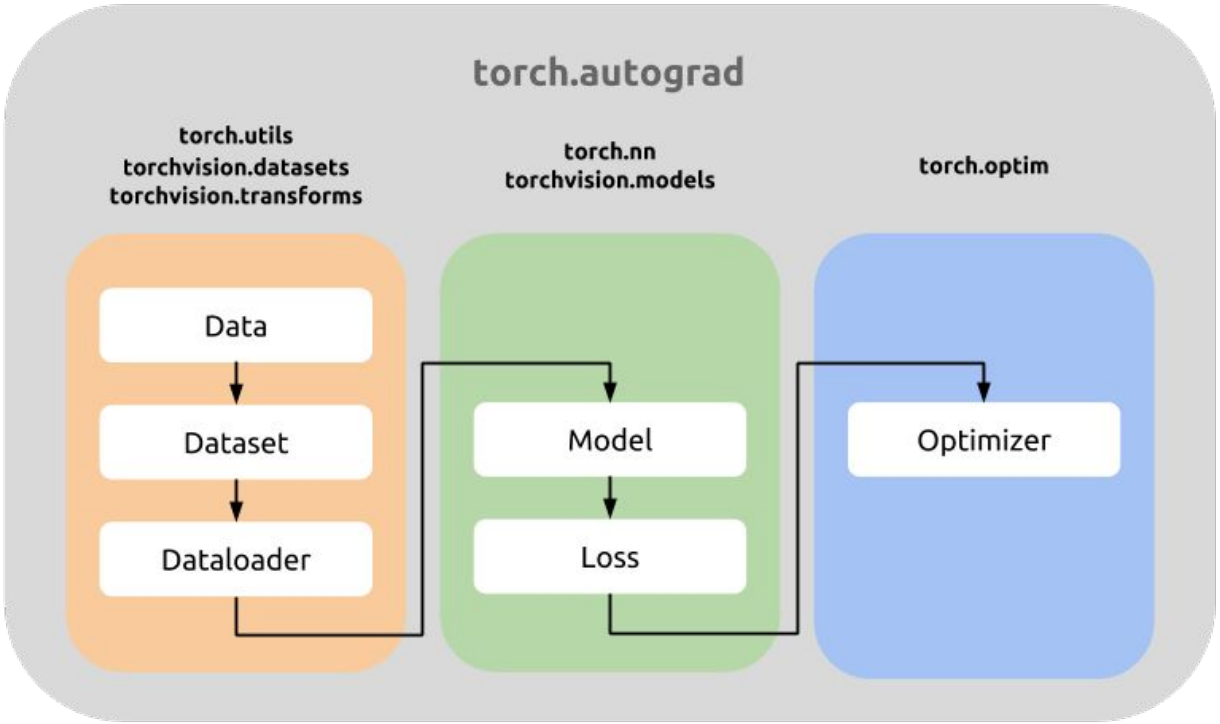
Usual workflow:

- Develop and train network in your favourite framework
- Convert and optimize in target framework for production



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

Pytorch: Overview

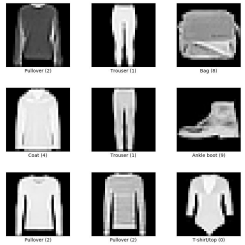


Some key features

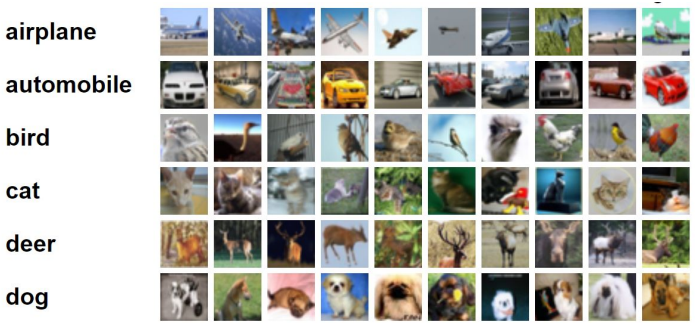
- Simple device management

```
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
```



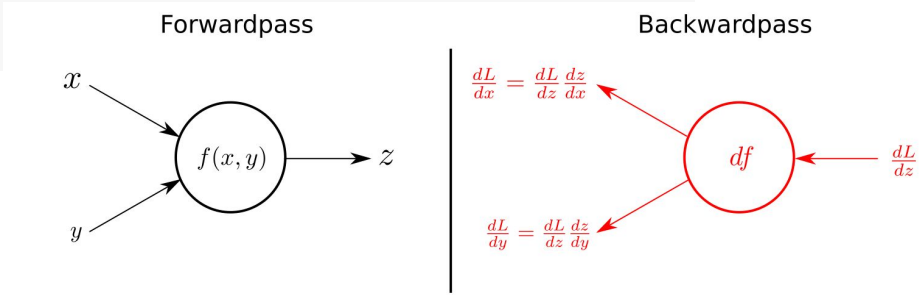
- Implementations of:
  - Optimizers, etc.
  - Datasets
  - Automatic gradients



Easy network creation

```python
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_size)

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

Where is the backward pass?



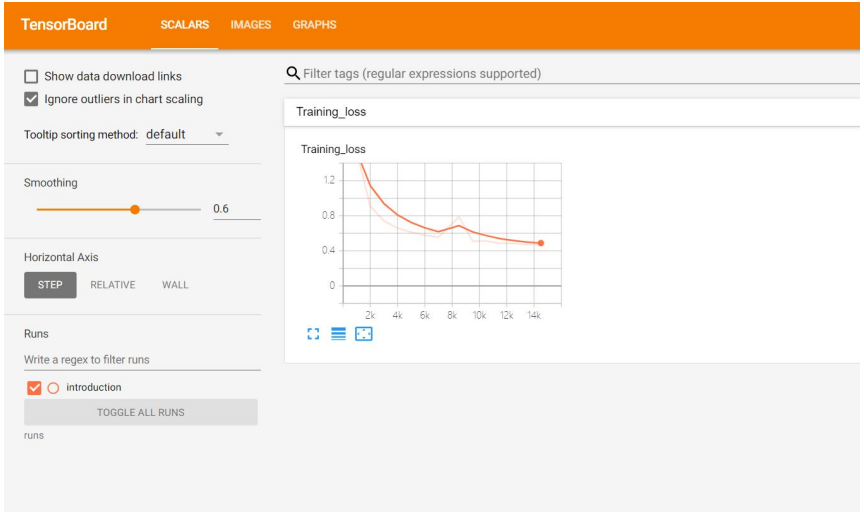
References on Pytorch

- Repository: https://github.com/pytorch/pytorch
- Examples (recommendation): https://github.com/pytorch/examples
- PyTorch for NumPy users: https://github.com/wkentaro/pytorch-for-numpy-users
- Look up your own and share! ☺️

Tensorboard (also in Pytorch)

- Directly access tensorboard in your training loop

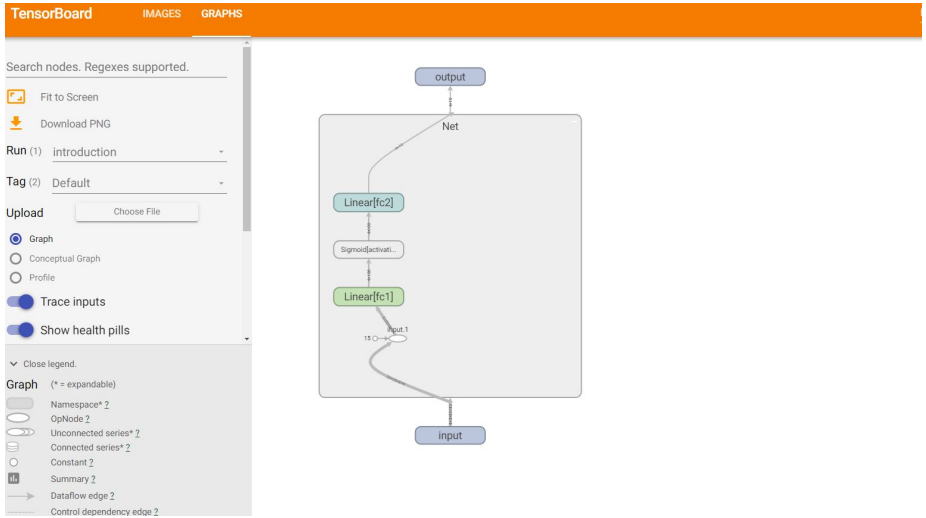
- Tensorboard generates the graph/timestamps etc. for you



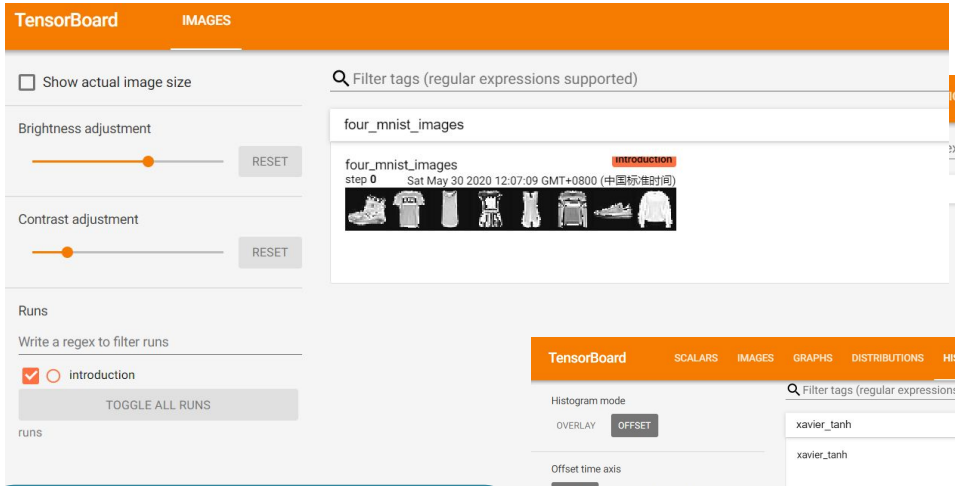
Visualize Networks

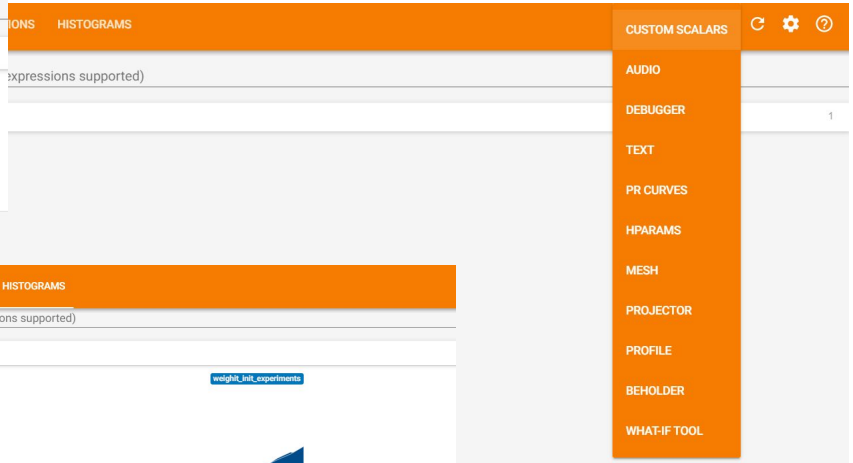
- Using a single forward pass, tensorboard can map and display your network graph

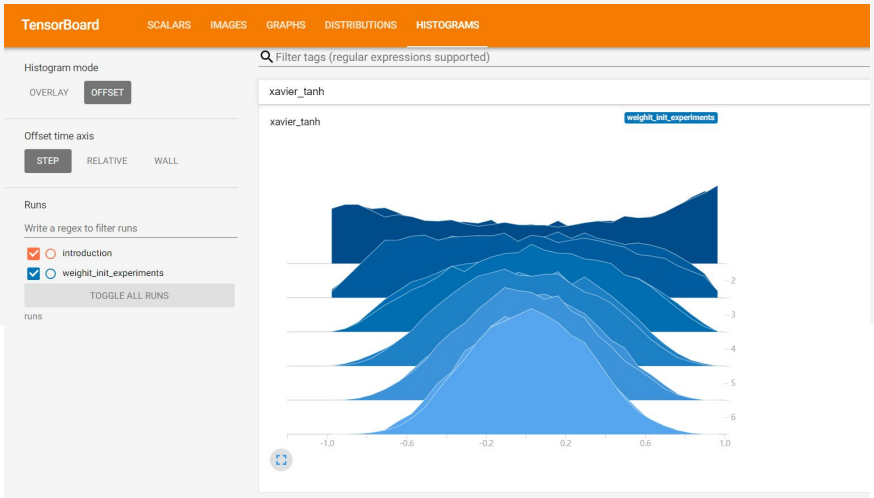
Graph creation needs network & one batch!



In short: document everything!



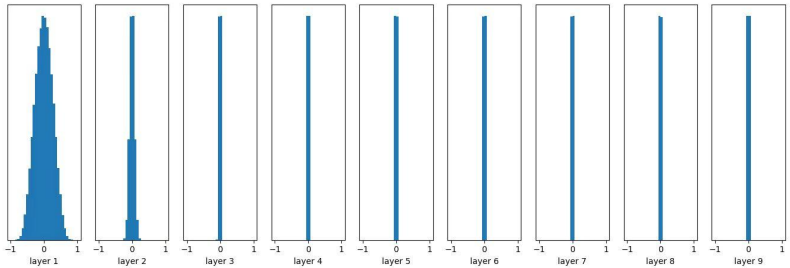


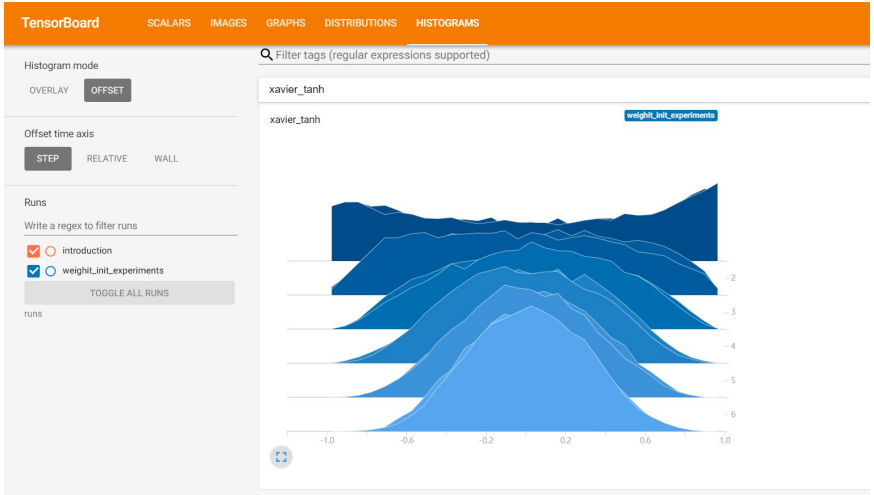


Everything is saved "automatically"!

Example: Weight Initialization

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





More Abstraction: 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}'.format(
              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
- configure_optimizers

Lightning Trainer

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}'.format(
              epoch, batch_idx * len(data), len(train_loader.dataset),
              100. * batch_idx / len(train_loader), loss.item()))
```

PyTorch Lightning

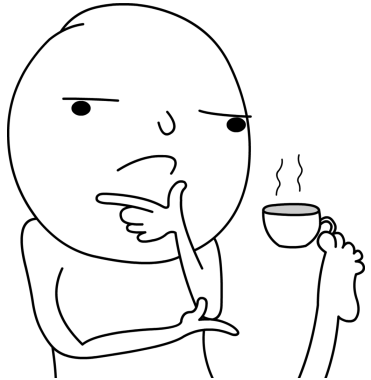
```
# model
class Net(LightningModule):
  def __init__(self):
      self.layer_1 = torch.nn.Linear(28 * 28, 128)
      self.layer_2 = torch.nn.Linear(128, 10)

  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

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 ☺️

What to use? Your call!

- Advantages
  - Better overview of the relevant code
  - Nice debugging features
  - Many automated options, like logging

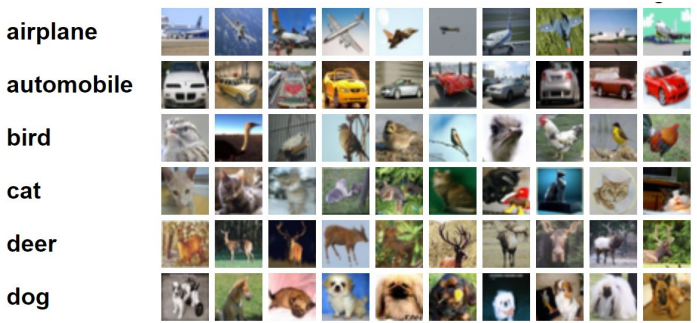


- Potential Problems
  - Can have issues like any stock library…
  - Not always straightforward to add features yourself

“Optional” Submission

CIFAR10… Again…

- Task: CIFAR10 classification (but now in Pytorch)
- New:
  - More knowledge from lecture 7
  - Can use everything but no: convolutional layers/transformers/ pre-trained networks
  - Filesize and parameter limit



So... Tuning again?

- Make sure
  - To get into pytorch
    (read docs and source code!)
  - To improve upon your previous submission
  - How can you select good hyperparameters?
  - Discuss with fellow students
    -> Code sharing on campuswire allowed!

Organization

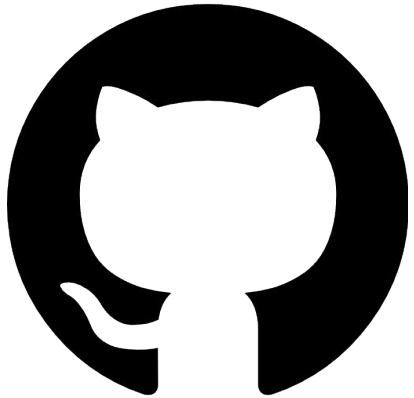
Post Deadline Submissions

Post Deadline Exercises

The following entries are identical to the ones above, but allow you to submit solutions after the deadline of the respective exercise has passed.
Submitting solutions to these "Post Deadline" exercises does not count towards the bonus nor can they substitute a missed exercise!

- **Exercise 3 (Post Deadline) – Dataset and Dataloader [Optional]**
- **Exercise 4 (Post Deadline) – Solver and Linear Regression [Optional]**
- **Exercise 5 (Post Deadline) – Neural Networks [Optional]**
- **Exercise 6 (Post Deadline) – Hyperparameter Tuning [Optional]**
- **Exercise 7 (Post Deadline) – Intro to Pytorch [Optional]**
- **Exercise 8 (Post Deadline) – Autoencoder [Optional]**
- **Exercise 9 (Post Deadline) – Convolutional Neural Networks [Optional]**
- **Exercise 10 (Post Deadline) – Semantic Segmentation [Optional]**
- **Exercise 11 (Post Deadline) – Recurrent Neural Networks [Optional]**

Solutions on Github



- Please don't upload solutions
  - You only hurt future students progression
  - We will issue take-downs!
  - No future employer cares

- What can you do else?
  - Choose an exercise of 7, 9, 10, or any other task/paper
  - Document your whole journey
  - Create: visualizations, ablations
  - Outline: key changes, maybe a story
  - Share: your documents with students

See you next week