DSP Tutorial
Introduction to Pytorch

TA: Timmy S. T. Wan 萬世澤

Outline

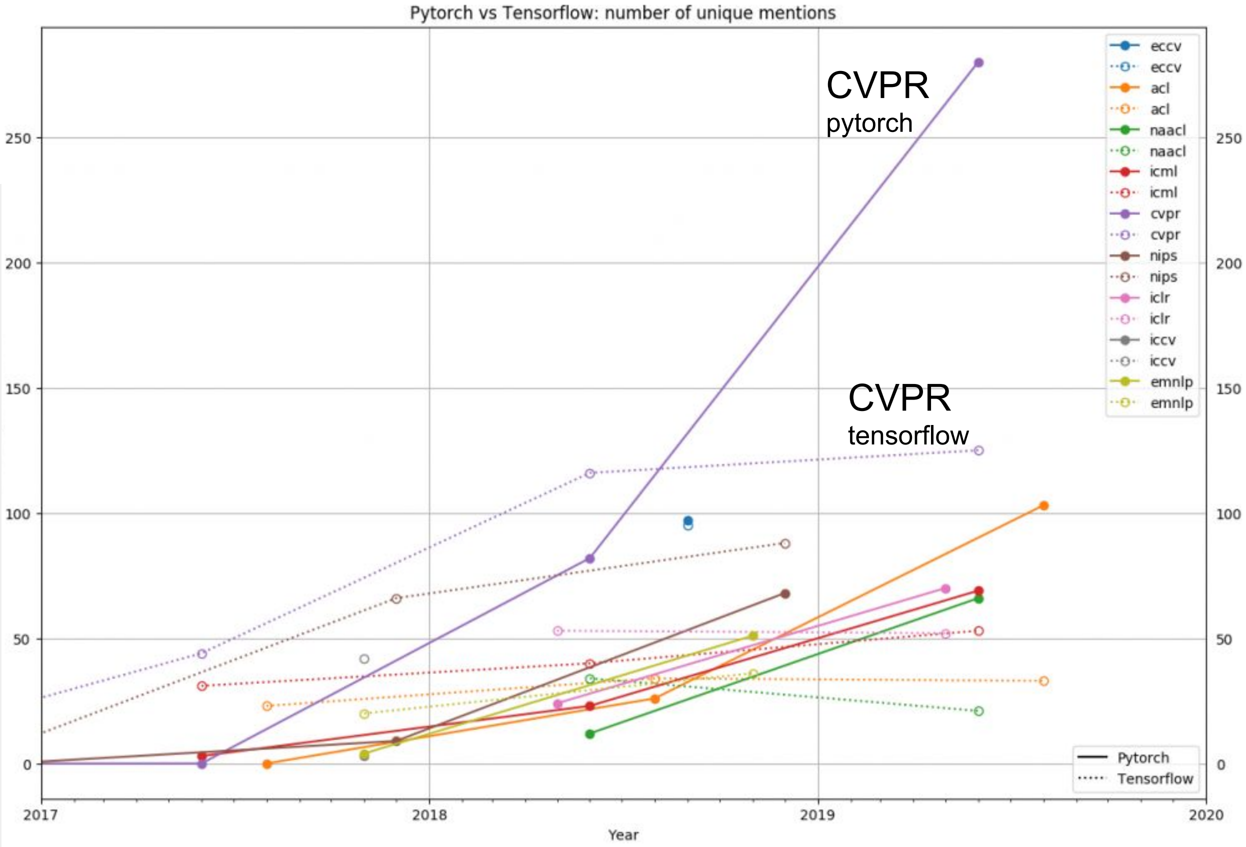
1. Why Pytorch?
2. Neural Network in Brief
3. Implement a linear autoencoder
4. Homework: Inpainting
5. Reference

Related materials can be downloaded here.

Why Pytorch?

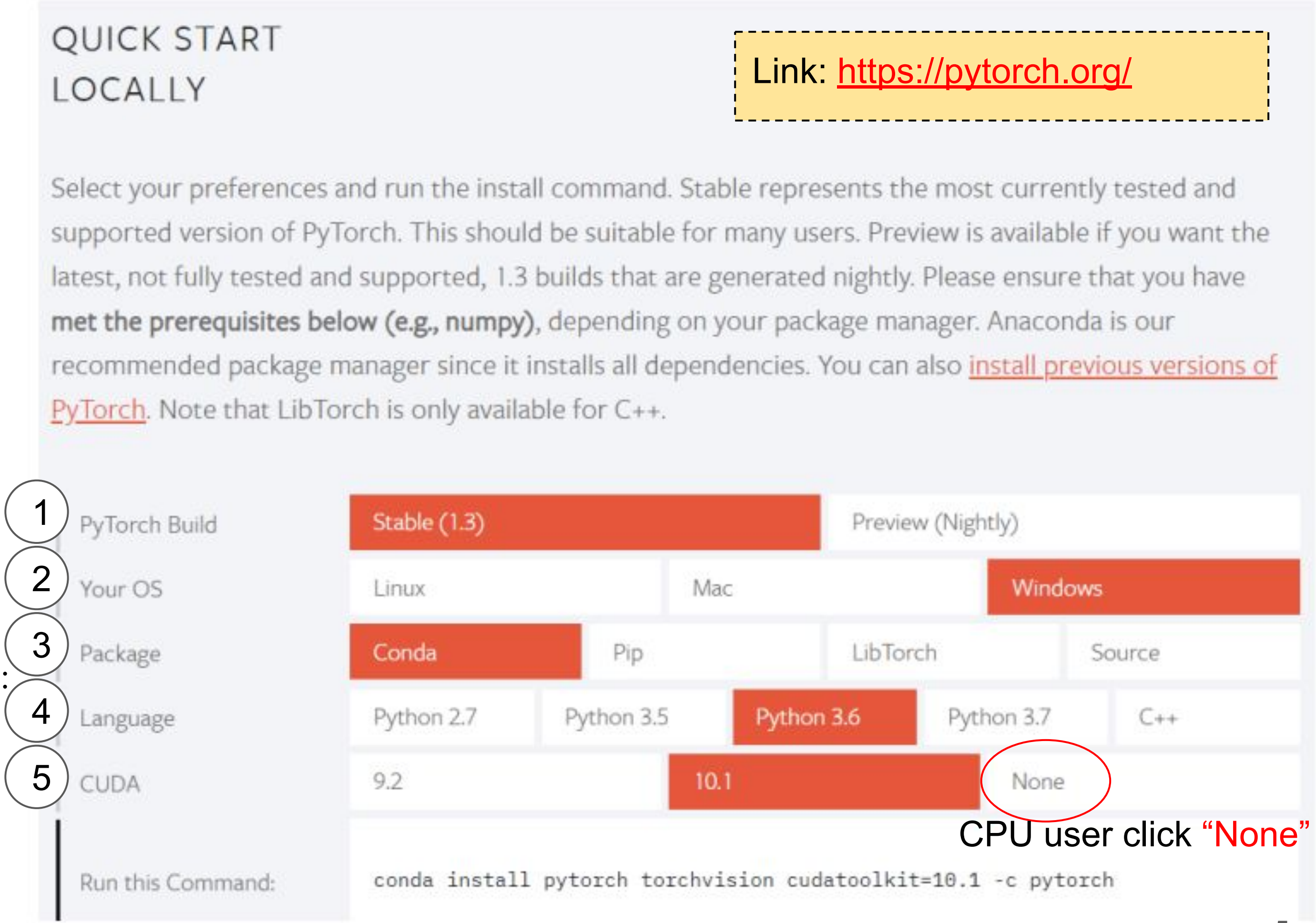
PYTORCH

1. A framework developed by Facebook
2. Speed up the prototyping the deep learning model
3. Widely used for research community
4. Python first
5. This tutorial is for **Pytorch 1.0.0+**



ref. https://blog.exxactcorp.com/pytorch-vs-tensorflow-in-2020-what-you-should-know-about-these-frameworks/

Installation



QUICK START LOCALLY

Link: https://pytorch.org/

Select your preferences and run the install command. Stable represents the most currently tested and supported version of PyTorch. This should be suitable for many users. Preview is available if you want the latest, not fully tested and supported, 1.3 builds that are generated nightly. Please ensure that you have **met the prerequisites below (e.g., numpy)**, depending on your package manager. Anaconda is our recommended package manager since it installs all dependencies. You can also install previous versions of PyTorch. Note that LibTorch is only available for C++.

1	PyTorch Build	[✓ Stable (1.3)]	Preview (Nightly)		
2	Your OS	Linux	Mac	[✓ Windows]	
3	Package	[✓ Conda]	Pip	LibTorch	Source
4	Language	Python 2.7	Python 3.5	[✓ Python 3.6]	Python 3.7	C++
5	CUDA	9.2	[✓ 10.1]	None	

CPU user click “None”

Run this Command:

```
conda install pytorch torchvision cudatoolkit=10.1 -c pytorch
```

If you have CPU only, the following script will be enough: “pip install torch torchvision”

Neural Network in Brief

The materials are modified from PyTorch Tutorial for NTU Machine Learning Course 2017

What we want to do using neural network?

Learning a function **F**, such that **F(x)** = **y** where function **F** is a neural network.

E.g.

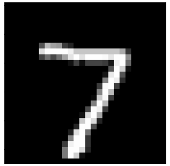
$$MSE = \frac{1}{n} \Sigma \left(y - y' \right)^2$$

Learn **F(.)**, such that **F(X)** = **Y**. The goal is to minimize the loss **L (e.g. MSE Loss)** which we actively optimize the model on.

Data X	Ground Truth Y	Prediction Y’
x1	y1	y1’
x2	y2	y2’
...	...	...

E.g.

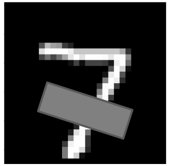
F(



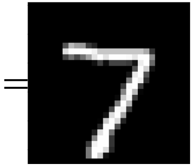
) = ‘7’

Classification: train f as a classifier

F(

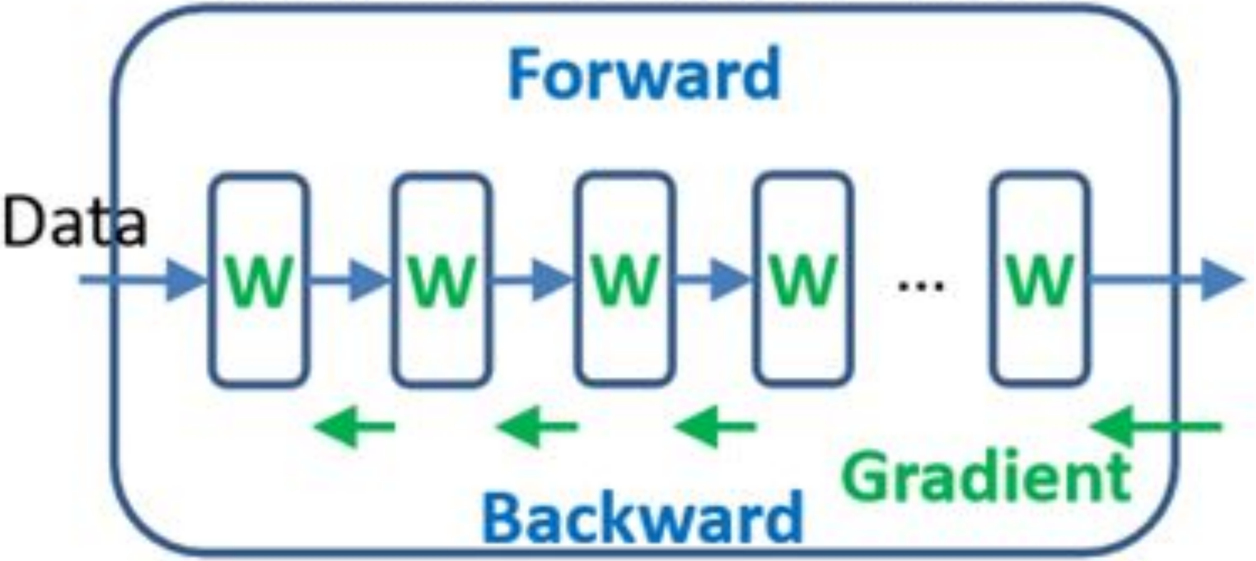


) =



Inpainting: f could be a autoencoder

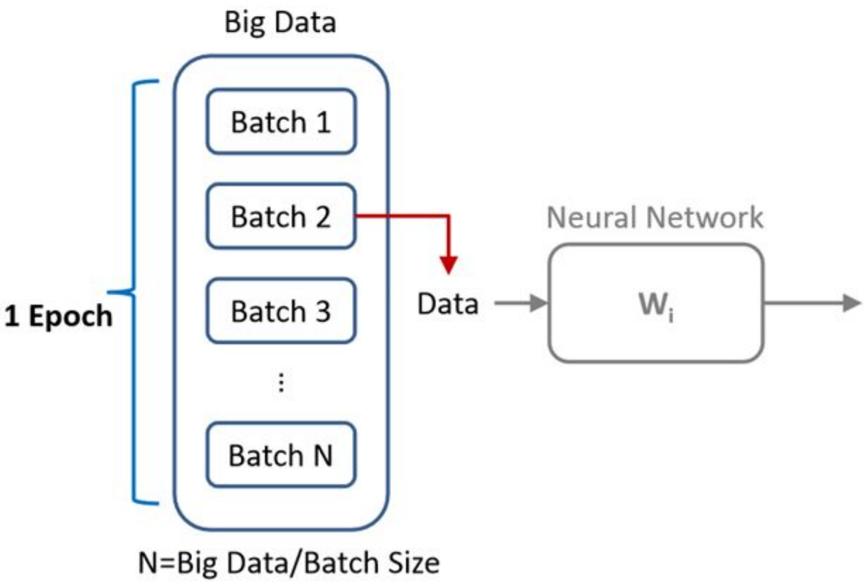
F is a Neural Network



Prediction
Y’

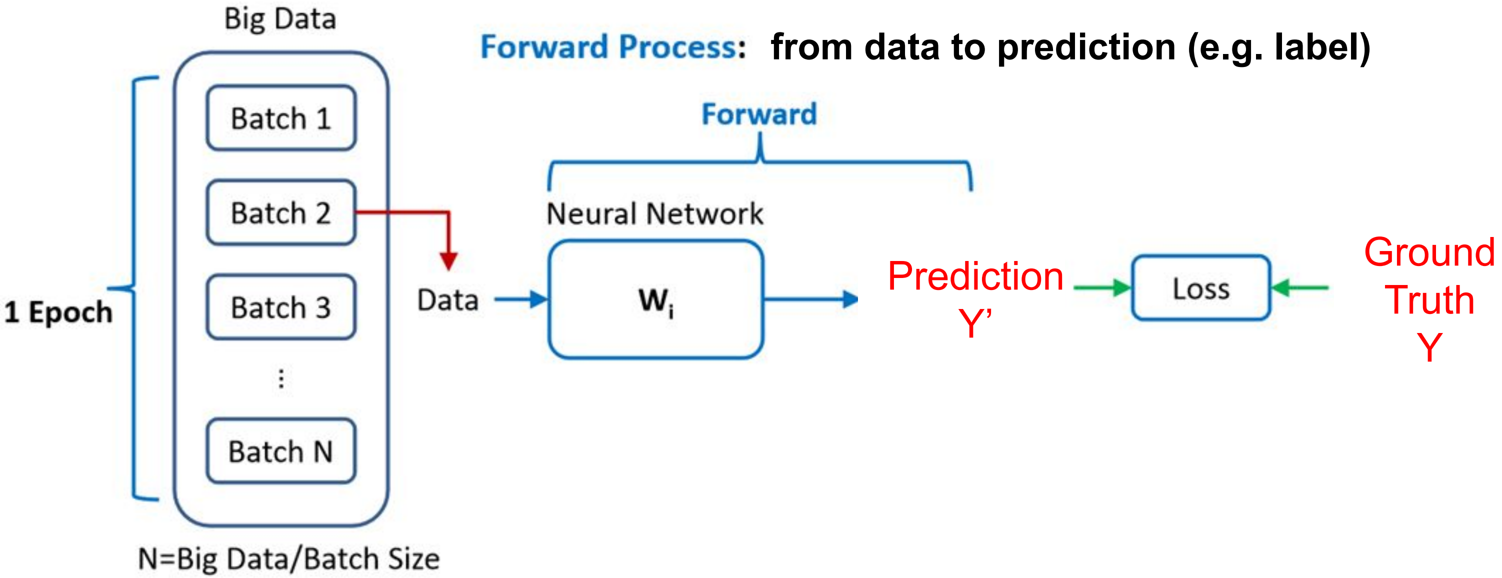
Each W is a layer weight!

Neural Network in Brief

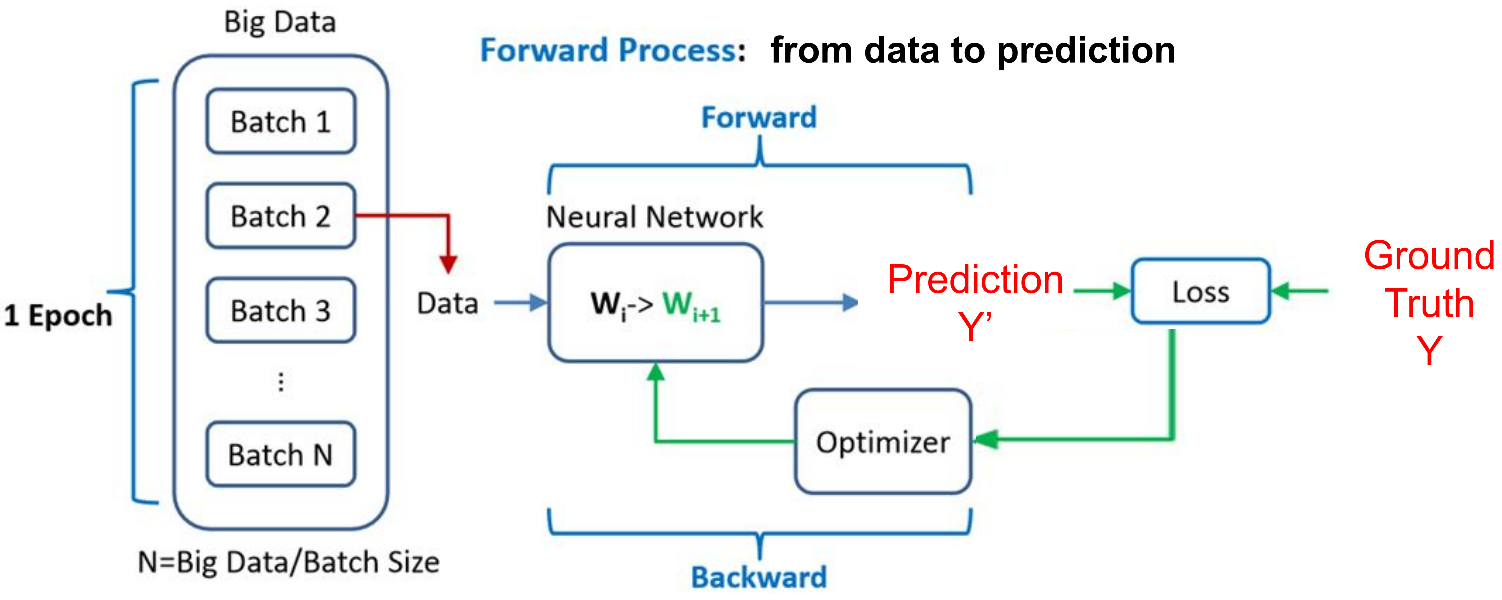


Neural Network in Brief

Forward Process: **from data to prediction (e.g. label)**



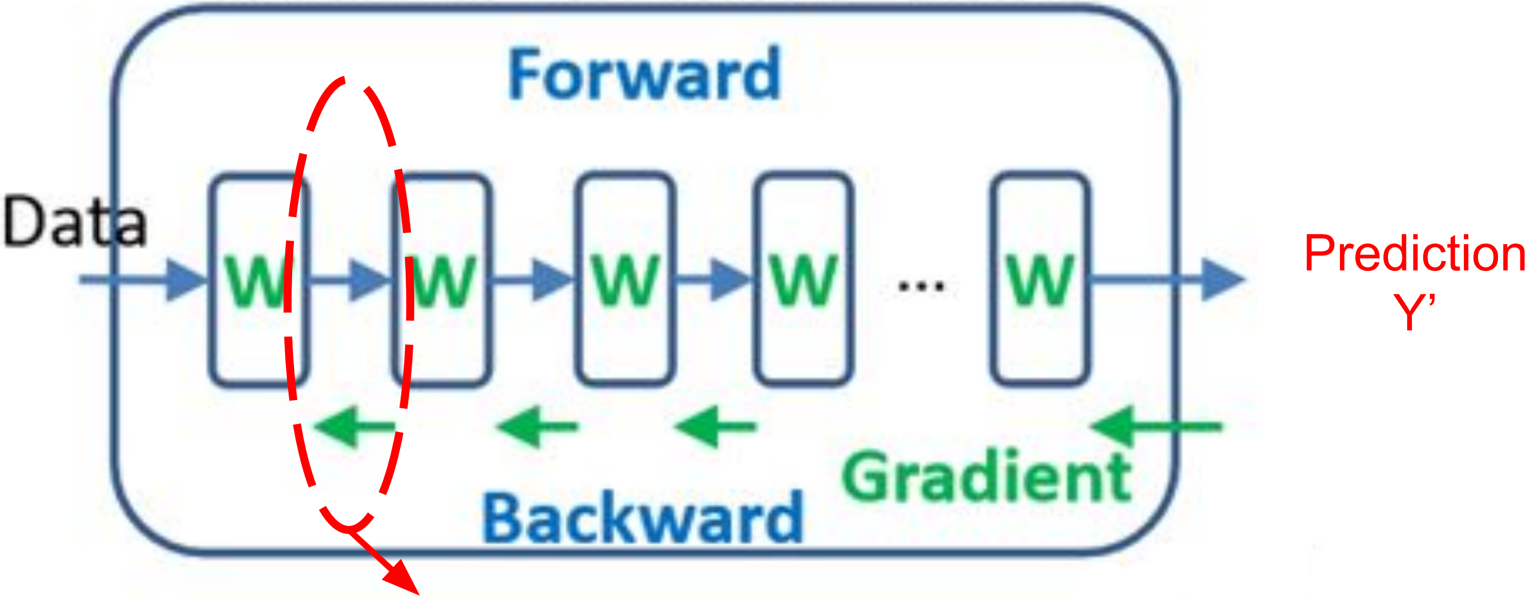
Neural Network in Brief



Forward Process: from data to prediction

Backward Process: update the parameters

Data flow inside the neural net



What goes inside a flow?

1. **Tensor (n-dim array)**
2. **Gradient of Functions**

Concepts of PyTorch

- Modules of PyTorch

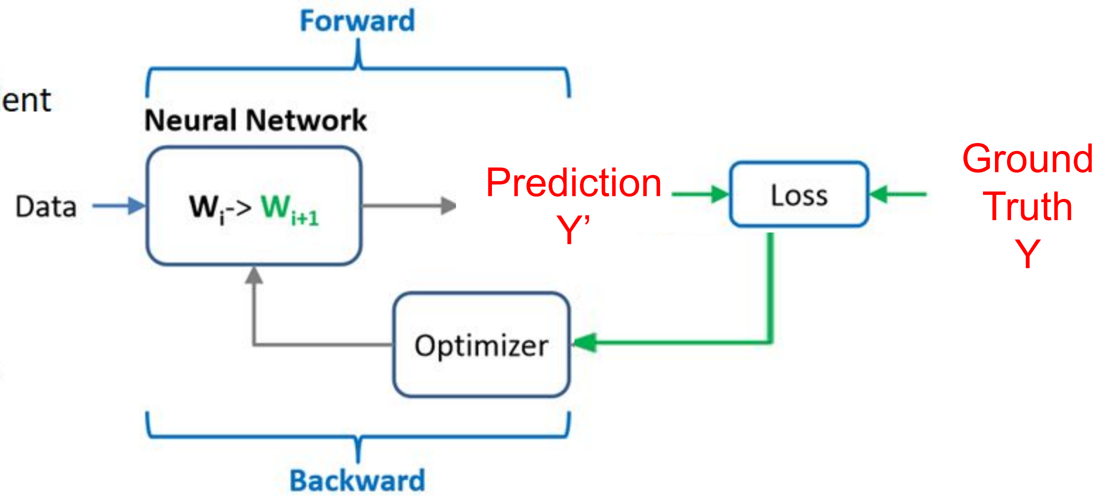
Data:

- Tensor

- Tensor with gradient

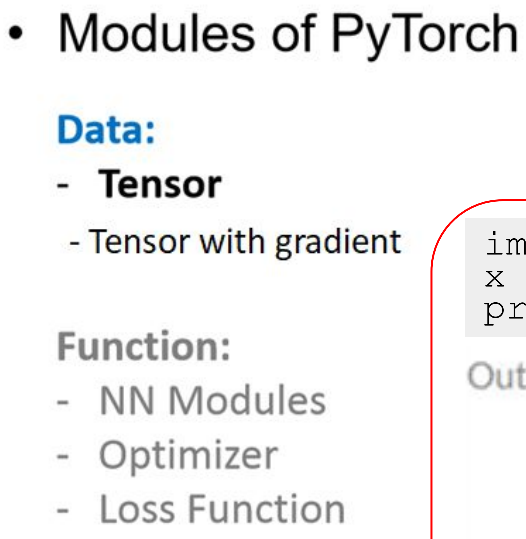
Function:

- NN Modules
- Optimizer
- Loss Function



Tensor is a basic unit in Pytorch

Tensor is a multi-dimensional matrix containing elements of a single data type, and PyTorch provides functions for operating on these Tensors.



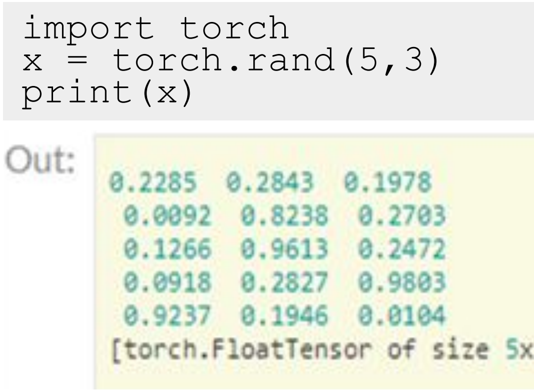
- Modules of PyTorch

Data:

- **Tensor**
  - Tensor with gradient

Function:

- NN Modules
- Optimizer
- Loss Function



```
import torch
x = torch.rand(5,3)
print(x)
```

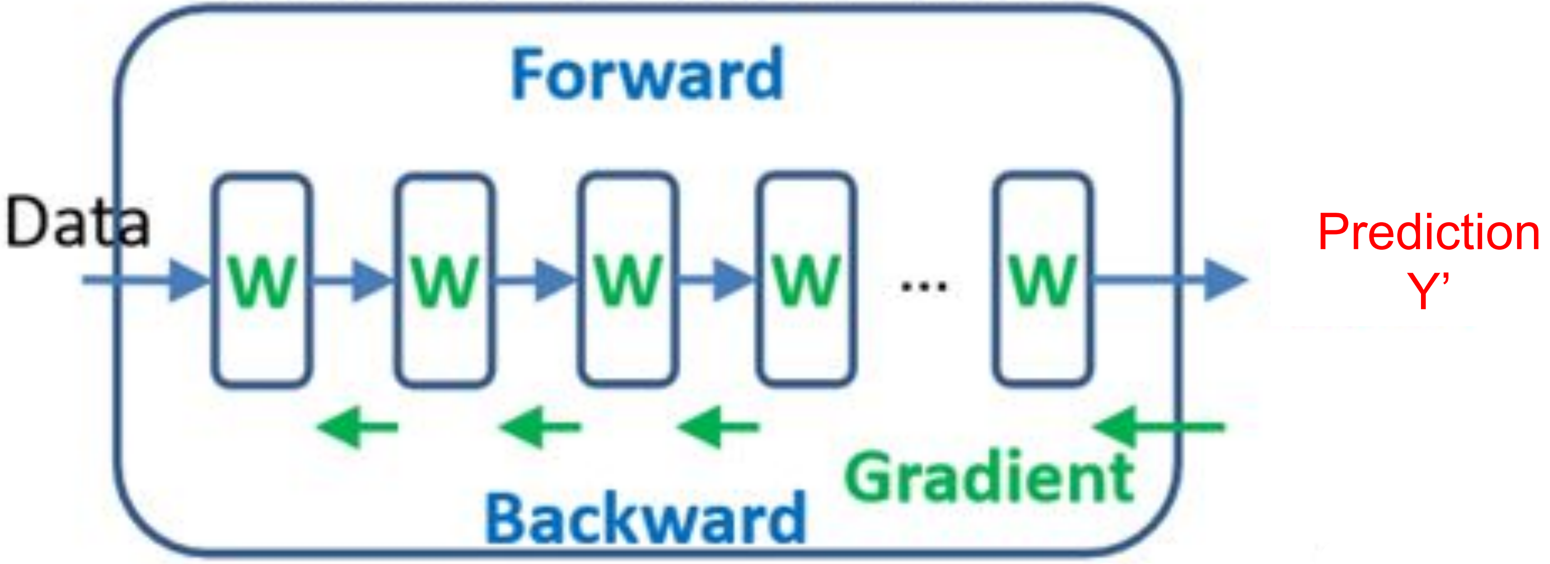
Out:

```
0.2285  0.2843  0.1978
 0.0092  0.8238  0.2703
 0.1266  0.9613  0.2472
 0.0918  0.2827  0.9803
 0.9237  0.1946  0.0104
[torch.FloatTensor of size 5x3]
```

- Operations
  - z=x+y
  - torch.add(x,y, out=z)
  - y.add_(x) # in-place

To learn more, please see link1 and link2 for more comprehensive understanding.

Tensor with gradients



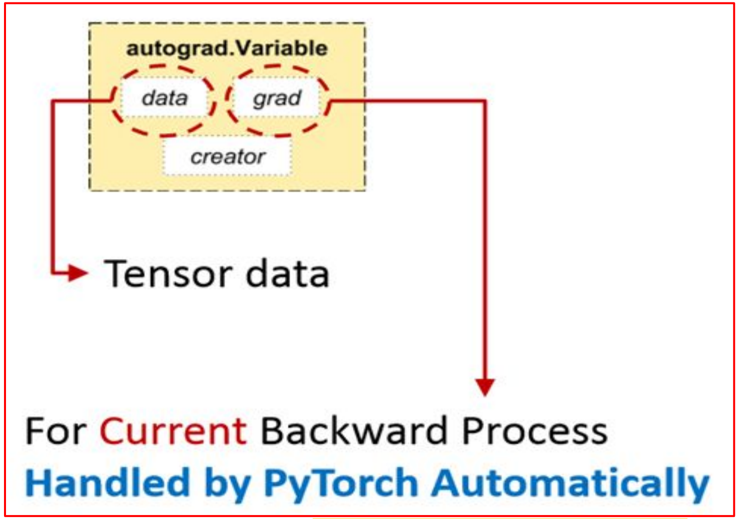
Require a Tensor with **Gradient** for backpropagation calculation!

E.g. Gradient Descent: $w_{t+1} = w_t - \eta \boxed{\nabla w_t}$

Tensor with gradients

Modules of Pytorch

- Data:
  - Tensor
  - **Tensor with gradients**
- **Function:**
  - NN Modules
  - Optimizer
  - Loss Function



For Current Backward Process

Handled by PyTorch Automatically

Tensor can be created with requires_grad=True so that torch.autograd records operations on them for automatic differentiation. For example:

```
import torch
updatable_tensor = torch.ones((2,2), requires_grad=True)
```

Note. requires_grad is set to False by default!

Example: Compute Gradients

- Modules of PyTorch

Data:

- Tensor

Tensor with gradients

To learn more about auto-differentiation, please see this article.
If you’re unfamiliar with the math behind neural net, I also recommend you to read this article.

```
from torch import ones
x = ones((2,2),requires_grad=True)
print(x)
```

```
1  1
1  1
```

$$\text{out} = \frac{1}{4}\sum z_i$$

$$z_i = 3y_i^2 = 3(x_i + 2)^2$$

$$\frac{\partial \text{out}}{\partial x_i} = \frac{3}{2}(x_i + 2) = \frac{9}{2}$$

```
y=x+2
z=y*y*3
out=z.mean()
out.backward()
print(x.grad)
```

```
4.5  4.5
4.5  4.5
```

NN Modules

- Modules of PyTorch

Data:

- Tensor
- Tensor with gradient

Function:

- **NN Modules**
- Optimizer
- Loss Function

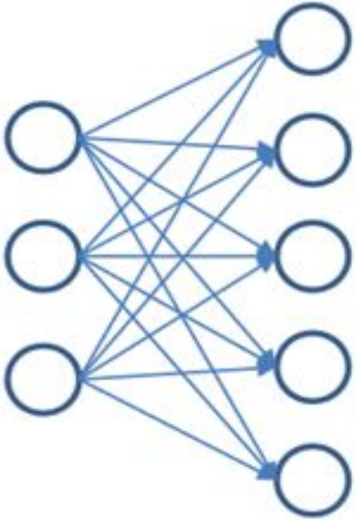
- NN Modules (torch.nn)
  - Gradient handled by PyTorch

- Common Modules
  - Convolution layers
  - Linear layers
  - Pooling layers
  - Dropout layers
  - Etc…

To build a linear autoencoder in this homework, we only use Linear layers.

NN Modules

- Linear Layer
  - torch.nn.Linear(in_features=3, out_features=5)
  - y=Ax+b



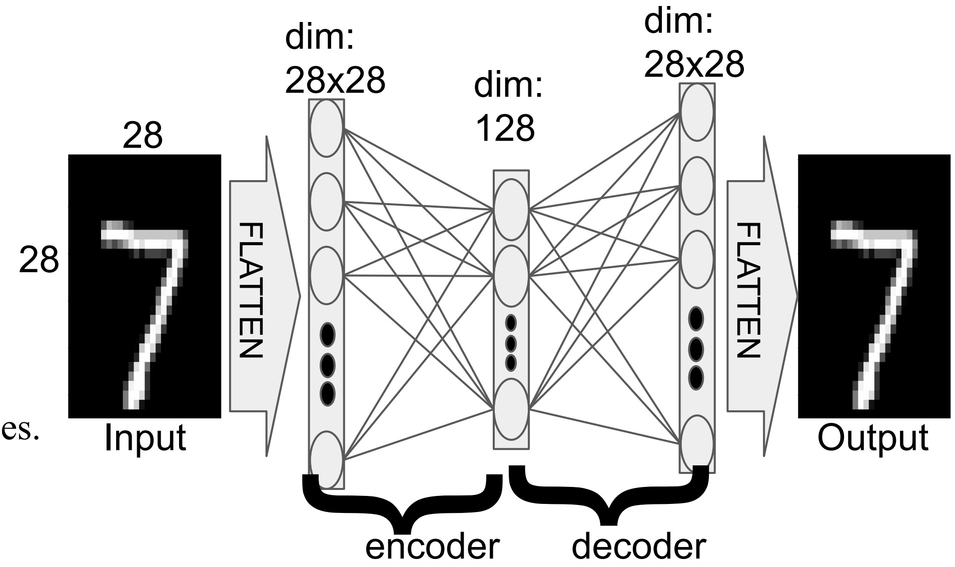
So far, we can implement a

linear autoencdoer

Linear Autoencoder

Define your network

```
import torch.nn as nn
class autoencoder(nn.Module):
  def __init__(self, latent_dim=128):
     # instantiate nn.Linear modules as member variables.
     super(autoencoder, self).__init__()
     self.encoder = nn.Linear(28 * 28, latent_dim)
     self.decoder = nn.Linear(latent_dim, 28 * 28)
  def forward(self, x):
     # Accept a input tensor x and return a output tensor y
     y = self.decoder(self.encoder(x))
     return y
```

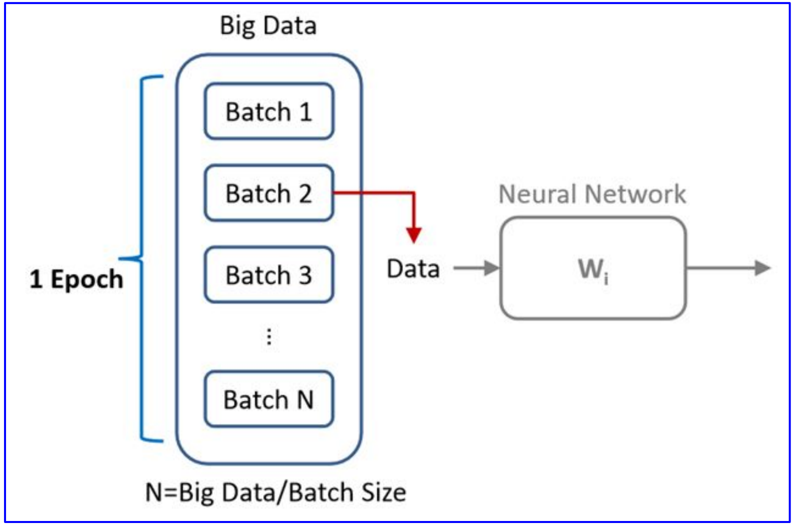


use your network

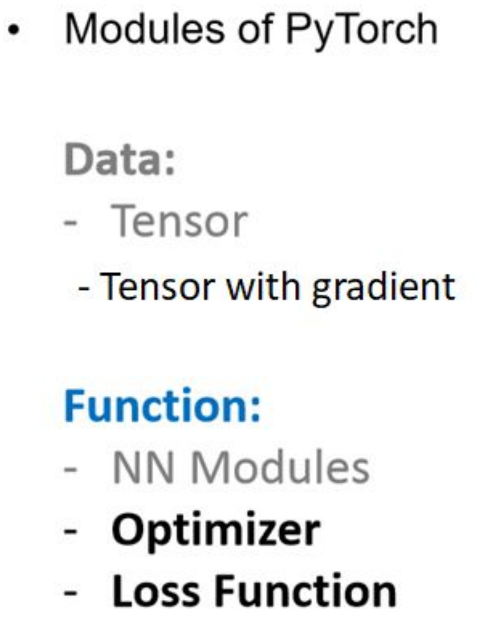
```
net = autoencoder(latent_dim=128)
input = torch.rand(28*28)
output = net(input)
print(output.shape) # 784
```

Train linear autoencoder with MNIST data

```
from torch.utils.data import DataLoader # helpful for iterating data
from torchvision.datasets import MNIST # helpful for MNIST data I/O
from torchvision.transforms import ToTensor # convert a matrix in range [0,255] to a FloatTensor in range [0.0,1.0]
trainset = MNIST('./data', download=True, train=True, transform=ToTensor())# load training set
trainloader = DataLoader(trainset,batch_size=64,shuffle=True)# get iterator over the training set
net = autoencoder() # initialize the neural network
net.train() # notify all layers that you are in training mode
loss_fn = … # define loss function (see next page)
optimizer = … # define optimizer (see next page)
for data in trainloader:
    x, _ = data # input x with shape=(64,1,28,28)
    x = x.view(x.size(0),-1) # flatten input x with shape=(64,784)
    ground_truth = x.clone() # clone the input as ground truth
    prediction = net(x) # call forward function
    # compute gradient of loss and update weights below
    …
```



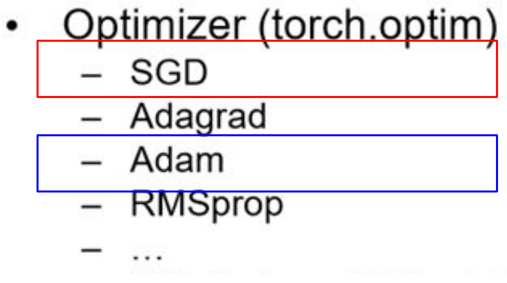
Optimizer and Loss Function



- Modules of PyTorch

Data:
- Tensor
- Tensor with gradient

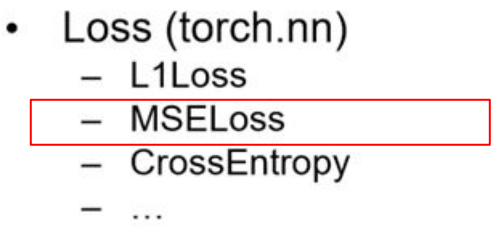
Function:
- NN Modules
- **Optimizer**
- **Loss Function**



- Optimizer (torch.optim)
  - SGD
  - Adagrad
  - Adam
  - RMSprop
  - …

Stochastic Gradient Descent

Adaptive Moment estimation (fast convergence!)

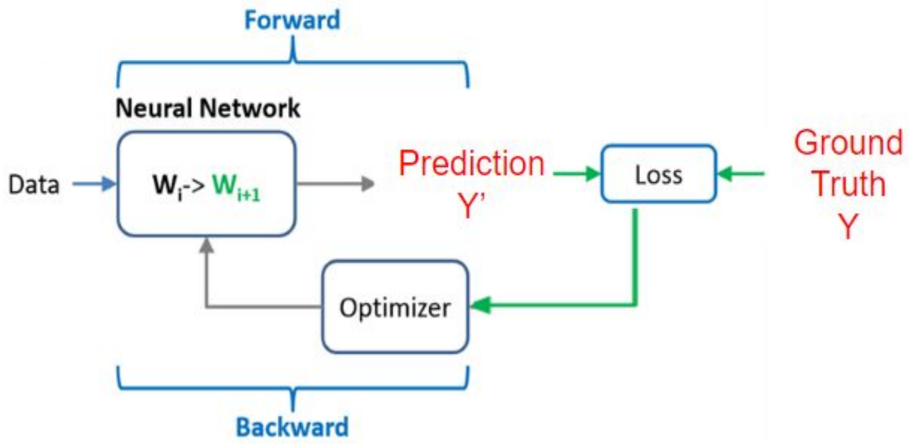


- Loss (torch.nn)
  - L1Loss
  - MSELoss
  - CrossEntropy
  - …

Mean Square Error Loss (L2-Loss)

Train linear autoencoder with MNIST data (Cont’d)

```
… … # import useful library
from torch.optim import SGD, Adam
from torch.nn import MSELoss
… … # define data loader and intialize the neural network
loss_fn = MSELoss()
optimizer = SGD(net.parameters(),lr=0.1)
for data in trainloader:
    x, _ = data # get input sample
    x= x.view(x.size(0),-1) # flatten input
    ground_truth = x.clone() # clone the sample as ground truth
    prediction = net(x) # forward
    # compute gradient of loss and update weights below
    loss = loss_fn(prediction, ground_truth) # compute loss
    optimizer.zero_grad() # clear gradient
    loss.backward() # compute gradient of loss w.r.t all the parameters in loss that have requires_grad = True
    optimizer.step() # update model parameters
```



Homework:
Image Inpainting using
Linear AutoEncoder

Reference

1. Pytorch Tutorial for ML course 2017
   1. https://www.slideshare.net/lymanblueLin/pytorch-tutorial-for-ntu-machine-learing-course-2017
2. Pytorch Official Tutorial
   1. https://pytorch.org/tutorials/index.html
3. Pytorch vs Tensorflow in 2020
   1. https://blog.exxactcorp.com/pytorch-vs-tensorflow-in-2020-what-you-should-know-about-these-frameworks/