PyTorch Tutorials

Zhuangwei Zhuang

Southern **A**rtificial **I**ntelligence **L**aboratory

South **C**hina **U**niversity of **T**echnology

Nov 15, 2017

Background

Deep learning has achieved great performance on image classification, object detection and speech recognition

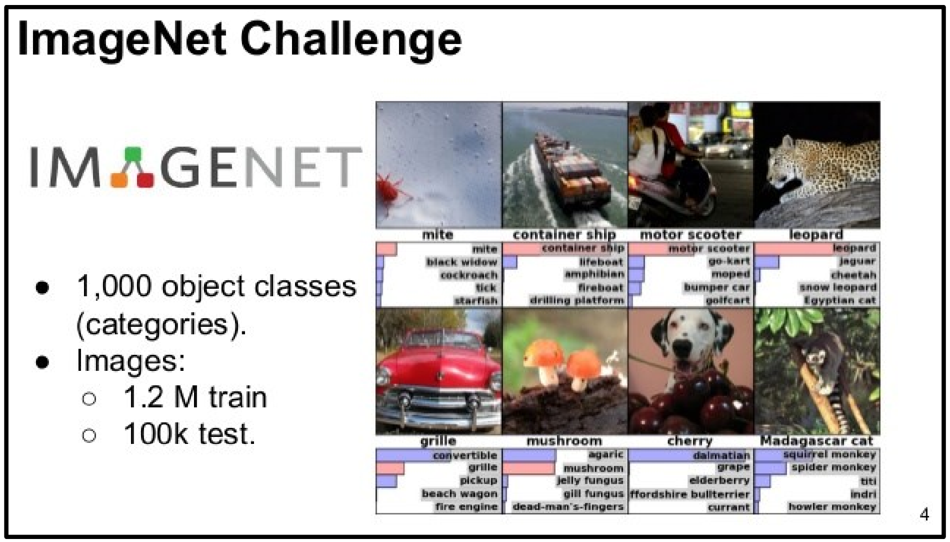
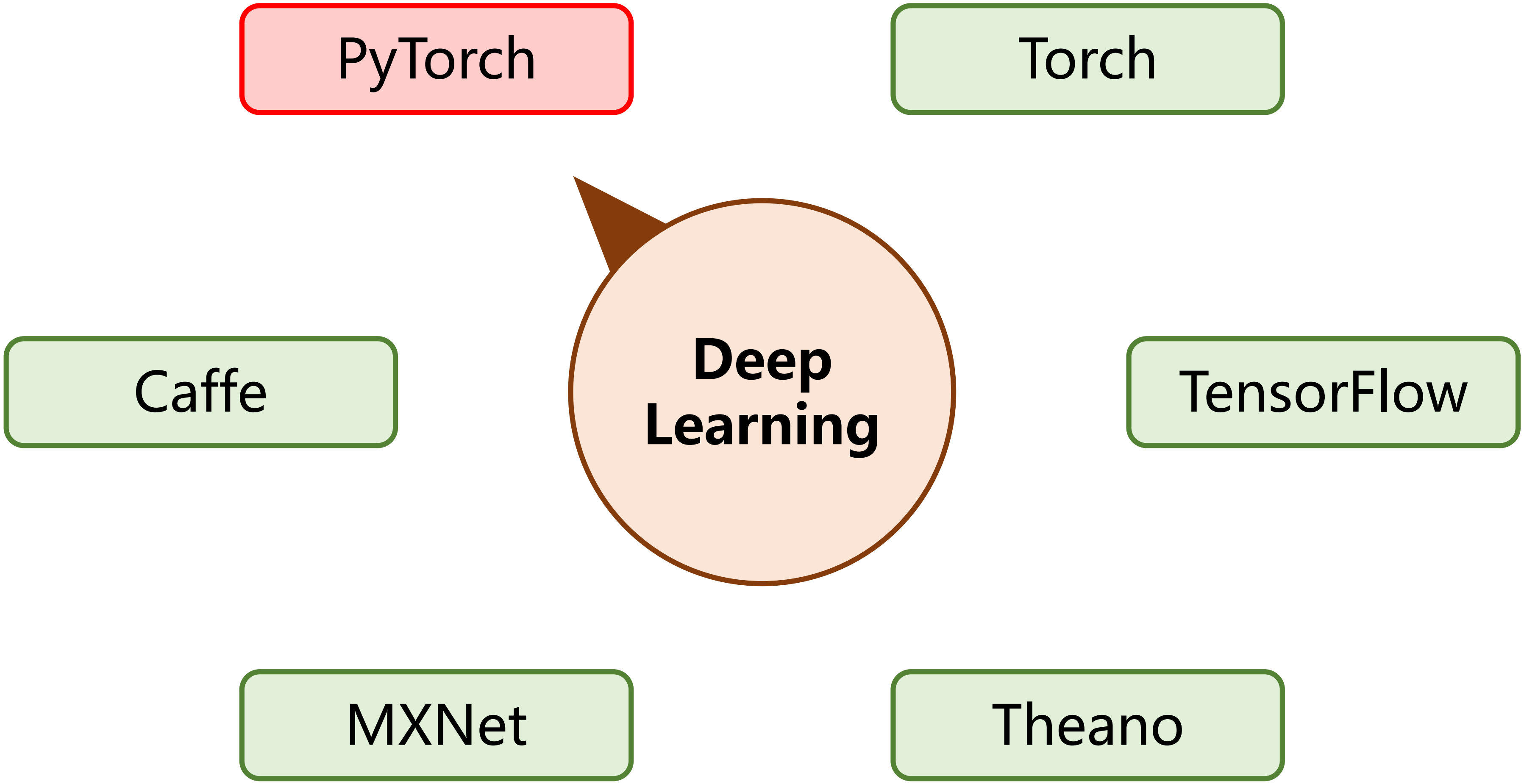


Image classification

Object detection

Platforms for Deep Learning



Outline

- **Introduction**
- **Use PyTorch on PC**
- **Use PyTorch for computation**
  - Tensor
  - Variable
  - GPU supported
- **Deep learning in PyTorch**
  - Data loader
  - Optimizer
  - Model
  - GPU supported
  - CNN example

Introduction

What is PyTorch

PyTorch is a python package for **tensor computation** and **deep neural networks** (DNNs). One can use PyTorch as:

- A replacement for numpy to use power of GPU
- A deep learning platform with maximum flexibility and speed

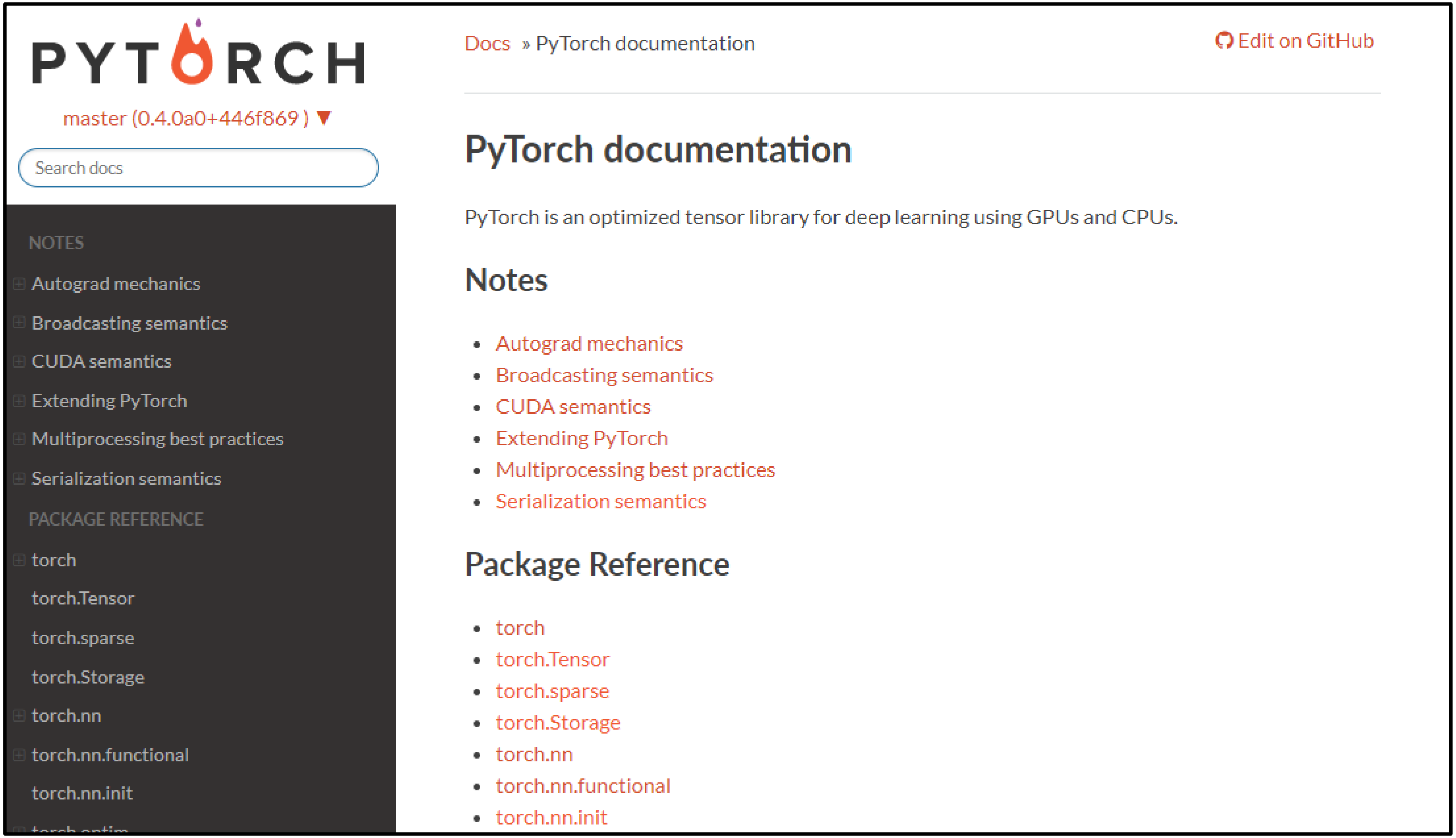
Official Websites

Get help from the following websites:

- **Source codes:** https://github.com/pytorch/pytorch
- **Official website:** http://pytorch.org/
- **Official tutorials:** http://pytorch.org/tutorials/
- **Official documentation:** (Provide detail description of all packages): http://pytorch.org/docs/
- **PyTorch forums:** (Platform for discussion) https://discuss.pytorch.org/
- **PyTorch examples:** https://github.com/pytorch/examples

Official Websites

➢ PyTorch documentation



Official Websites

➢ PyTorch forums

PYTORCH

Sign Up | Log In

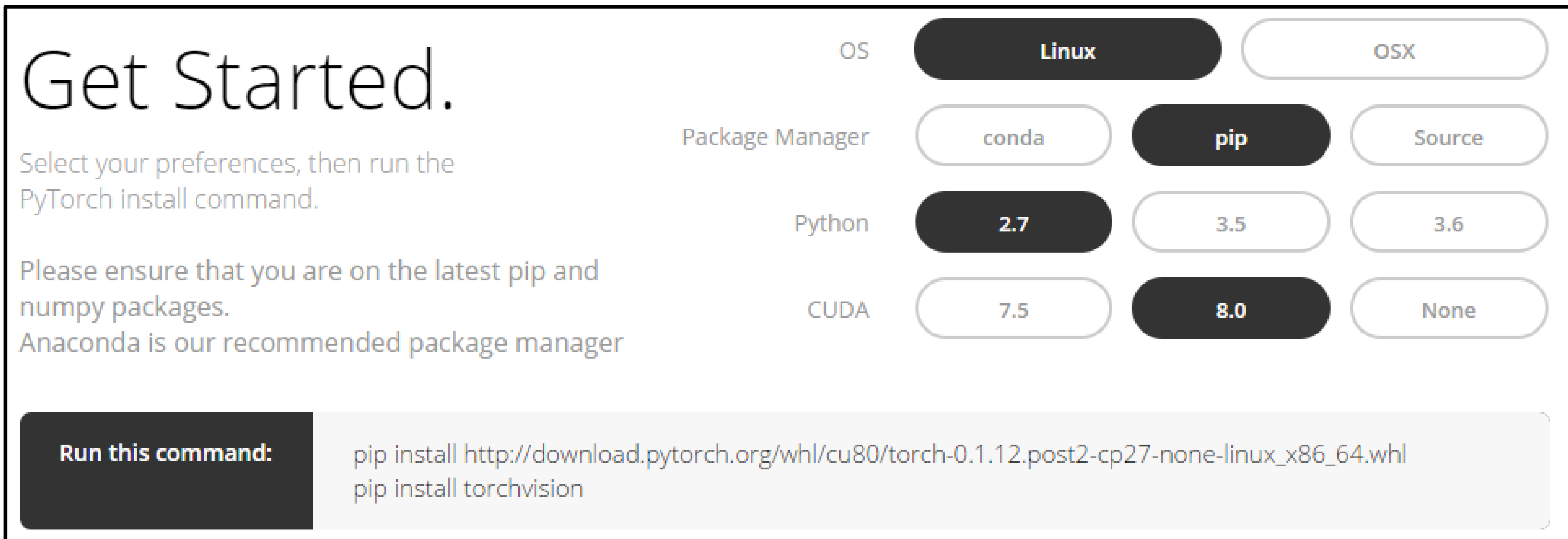
all categories | Latest | Top | Categories

Topic	Category	Users	Replies	Views	Activity
How to print a model after load it?			3	19	1m
Loss value is not reducing(stuck at 1.5 approx), is there something wrong in this network?			0	4	8m
Using word embedding with loss function			2	12	12m
Is there a way to use torch.cuda.comm.gather() with different tensor size like reduce_add_coalesced()?			1	4	20m
How to add new loss functions	autograd		1	6	37m
CNN training doesn't work because of tensor type			1	11	1h
Unsure about input type of convulution layer	reinforcement-learning		1	7	2h
Calculating loss on sequences with variable lengths			2	15	2h
☑ Mask selection with expand			10	41	2h

Installation

PyTorch only supports **Linux** and **OSX**. One can install PyTorch with following methods:

- **Binaries(recommended)**
- From source
- Docker image



Installation

➢ Install PyTorch from binaries:

- **Settings:** Linux, Python-2.7, Cuda-8.0
- Run the following commands on shell:

```
pip install http://download.pytorch.org/whl/cu80/torch-0.1.12.post2-cp27-none-linux_x86_64.whl
pip install torchvision [1]
```

[1] torchvision is a package that contains deep learning models and data sets for computer vision tasks.

PyTorch Packages

PyTorch consists of the following components:

- **torch:** a tensor library with strong GPU support
- **torch.autograd:** a automatic dierentiation library
- **torch.nn:** a neural networks library
- **torch.multiprocessing:** a python multiprocessing library with memory sharing of Tensors across processes.
- **torch.utils:** including utility functions.
- **torch.legacy:** legacy code ported over from Torch

Use PyTorch on PC

Software

- **IDE:** PyCharm Community Edition
  Visual Studio Code

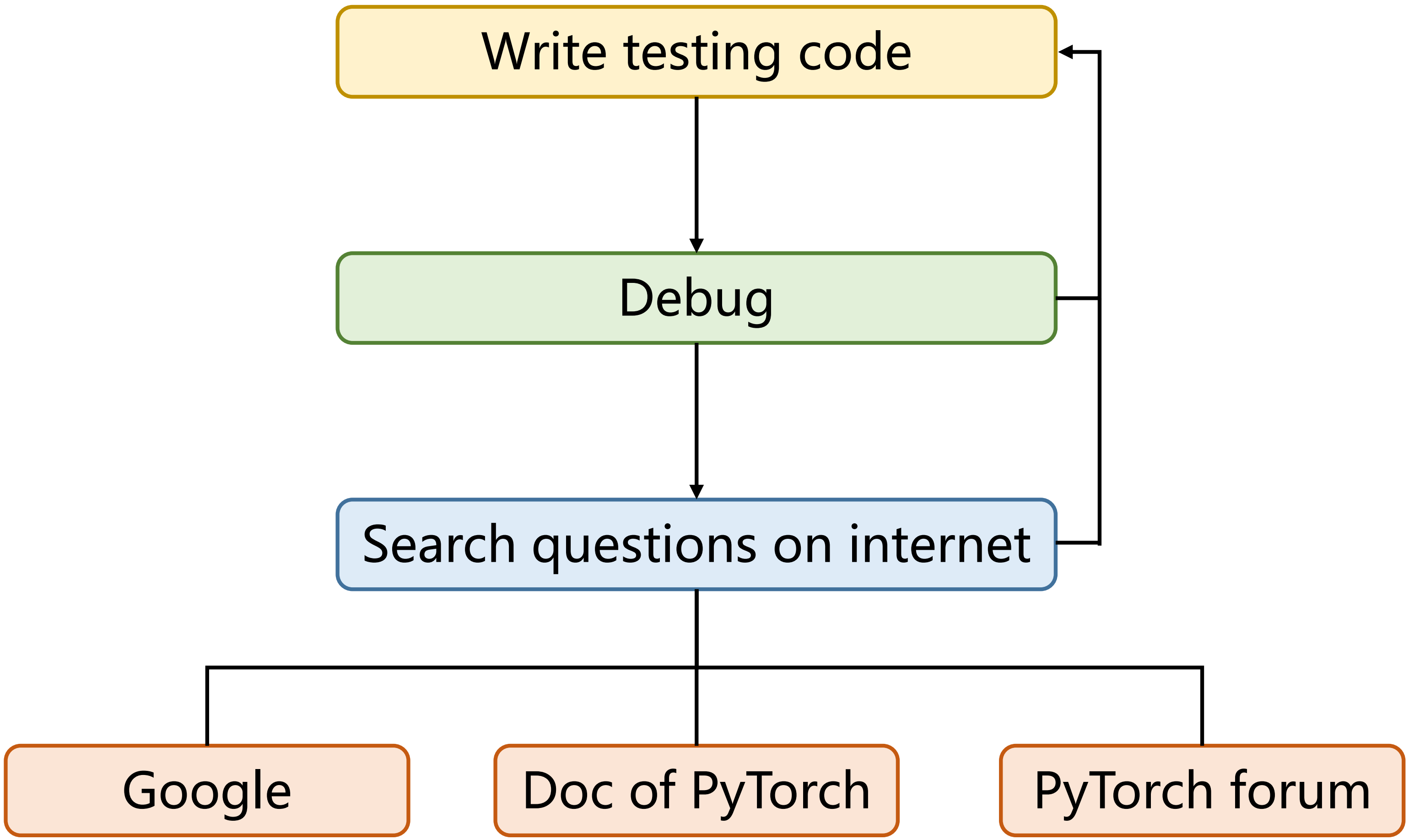
- **Python package:** Anaconda (python-2.7 recommended)

PyCharm

VS Code

Anaconda

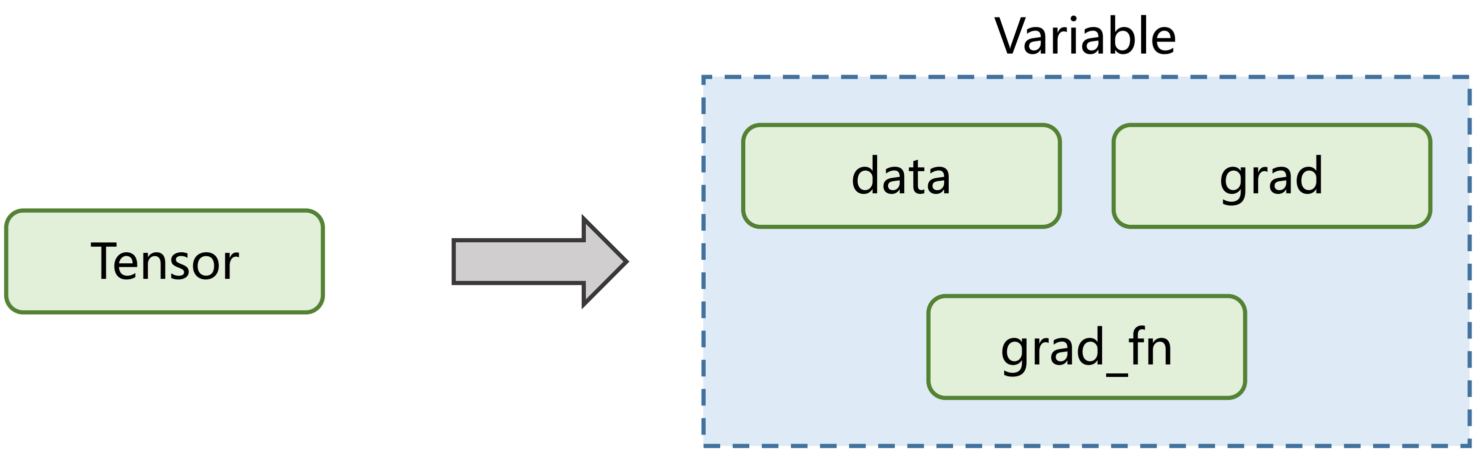
Learning PyTorch in Yourself



Use PyTorch for Computation

Data Type

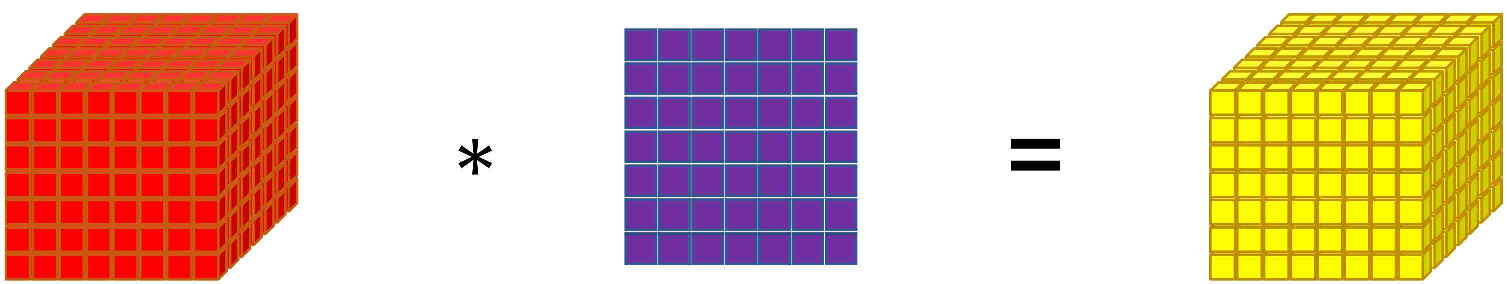
- **Tensor:** an n-dimensional array like numpy array
- **Variable:** used for automatic differentiation. Almost all operations of Tensor can be performed on Variable
- **Parameter:** A kind of Variable that is to be considered a module parameter



Tensor

PyTorch defines seven CPU tensor types and eight GPU tensor types

Data type	CPU tensor	GPU tensor
16-bit floating point	-	torch.cuda.HalfTensor
32-bit floating point	**torch.FloatTensor**	**torch.cuda.FloatTensor**
64-bit floating point	torch.DoubleTensor	torch.cuda.DoubleTensor
8-bit integer (unsigned)	torch.ByteTensor	torch.cuda.ByteTensor
8-bit integer (signed)	torch.CharTensor	torch.cuda.CharTensor
16-bit integer (signed)	torch.ShortTensor	torch.cuda.ShortTensor
32-bit integer (signed)	torch.IntTensor	torch.cuda.IntTensor
64-bit integer (signed)	torch.LongTensor	torch.cuda.LongTensor



Tensor

A tensor can be constructed through two methods:

- From Python list or sequence
- By specifying tensor size

```
>>>import torch
>>>torch.FloatTensor([[1, 2, 3], [1, 2, 3]])

1 2 3
1 2 3
[torch.FloatTensor of size 2x3]

>>>torch.FloatTensor(2, 4).fill_(1)

1 1 1 1
1 1 1 1
[torch.FloatTensor of size 2x4]
```

Creation Operations

- torch.eye(n, m=None, out=None)
- torch.from_numpy(ndarray)
- torch.linspace(start, end, steps=100, out=None)
- torch.logspace(start, end, steps=100, out=None)
- torch.ones(*sizes, out=None)
- torch.rand(*sizes, out=None)
- torch.randn(*sizes, out=None)
- torch.randperm(n, out=None)
- torch.arange(start, end, step=1, out=None)
- torch.range(start, end, step=1, out=None)
- torch.zeros(*sizes, out=None)

Creation Operations

torch.zeros(*sizes, out=None)

returns a tensor filled with the scalar value 0, with the shape defined by the varargs sizes

```
>>>import torch
>>>torch.zeros(4, 5)

0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
[torch.FloatTensor of size 4x5]
```

Indexing, Slicing, Joining and Mutating

- torch.cat(seq, dim=0, out=None)
- torch.chunck(tensor, chunks, dim=0)
- torch.gather(input, dim, index, out=None)
- torch.index_select(input, dim, index, out=None)
- torch.masked_select(input, mask, out=None)
- torch.nonezero(input, out=None)
- torch.split(tensor, split_size, dim=0)
- torch.squeeze(input, dim=None, out=None)
- torch.stack(sequence, dim=0, out=None)
- torch.t(input, out=None)
- torch.transpose(input, dim0, dim1, out=None)
- torch.unbind(tensor, dim=0)
- torch.unsqueeze(input, dim, out=None)

Indexing, Slicing, Joining and Mutating

torch.squeeze(input, dim=None, out=None)

returns a tensor with all the dimentions of input of size 1 removed

```
>>>import torch
>>>x = torch.zeros(2, 1, 2, 1, 2)
>>>x.size()
(2L, 1L, 2L, 1L, 2L)
>>>
>>>y = torch.squeeze(x)
>>>y.size()
(2L, 2L, 2L)
>>>
>>>y = torch.squeeze(x, 0)
>>>y.size()
(2L, 1L, 2L, 1L, 2L)
```

Math Operations

PyTorch provides many math functions:

- Pointwise Operations
  - add(), ceil(), clamp(), div(), cos(), abs(), ...
- Reduction Operations
  - dist(), mean(), std(), norm(), sum(), var(), ...
- Comparison Operations
  - eq(), equal(), ge(), max(), ne(), sorted(), ...
- Other Operations
  - diag(), trace(), cross(), ...
- BLAS and LAPACK Operations
  - addbmm(), admm(), addmv(), ...

Pointwise Operations

torch.add(input, value, out=None)

Adds the scalar value to each element of the input tensor and returns a new resulting tensor

$$out = tensor + value$$

```
>>> import torch
>>> a = torch.FloatTensor(2).fill_(1)
>>> a
1
1
[torch.FloatTensor of size 2]
>>>
>>> torch.add(a, 20)
21
21
[torch.FloatTensor of size 4]
```

Reduction Operations

torch.mean(input)

Returns the mean value of all elements in the input tensor

```
>>> import torch
>>> x = torch.randn(1, 3)
>>> x

-0.2946 -0.9143 2.1809
[torch.FloatTensor of size 1x3]

>>>torch.mean(x)
0.32398951053619385
```

Comparison Operations

torch.eq(input, other, out=None)

Computes element-wise equality and returns a **torch.ByteTensor** containing a 1 at each location where the tensors are equal and a 0 at every other location

```
>>> x = torch.Tensor([[1, 2], [3, 4]])

1 2
3 4
[torch.FloatTensor of size 2x2]

>>>x.eq(2)

0 1
0 0
[torch.ByteTensor of size 2x2]
```

Variable

class torch.autograd.Variable

- data (any tensor class): tensor to wrap
- requires_grad(bool): indicating whether the Variable has been created by a subgraph containing any Variable. Can be changed only on leaf Variables
- volatile(bool): indicating whether the Variable should be used in inference mode

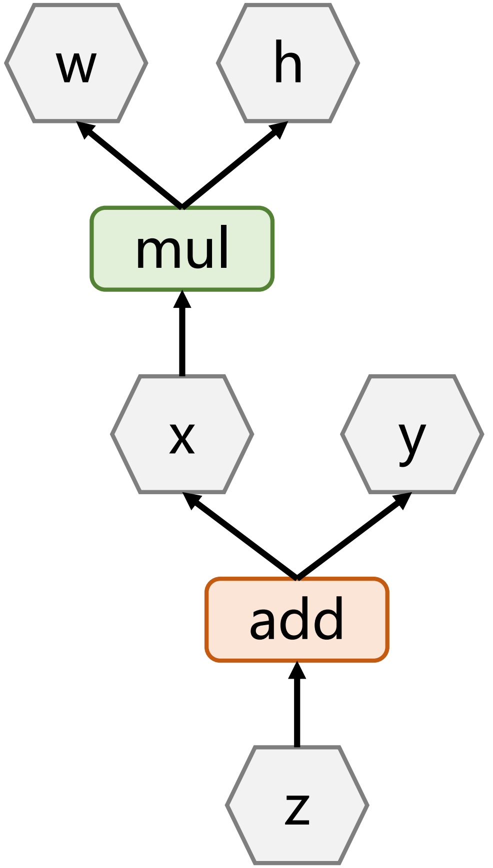
```
>>>import torch
>>>from torch.autograd import Variable
>>>x = Variable(torch.randn(3, 3), volatile=False)
```

Variable

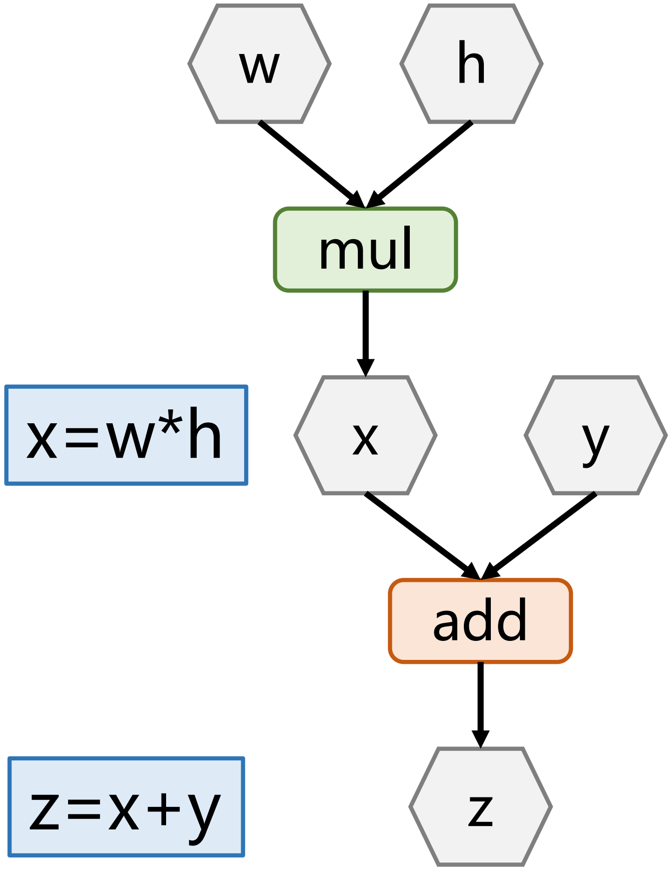
Variable API is nearly the same as regular Tensor API. In most cases tensors can be safely replaced with Variables

- data: tensor
- grad: gradient
- grad_fn: gradient function graph trace

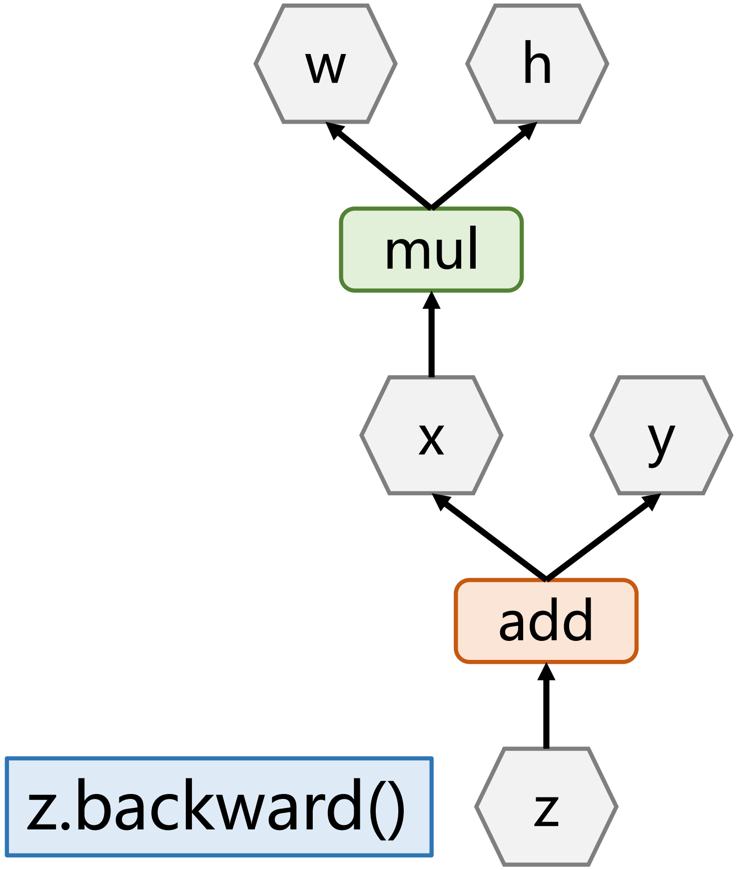
```
>>>import torch
>>>from torch.autograd import Variable
>>>
>>>w = Variable(torch.randn(3, 3))
>>>h = Variable(torch.randn(3, 3))
>>>x = w*h
>>>
>>>y = Variable(torch.randn(3, 3))
>>>z = x+y
```



Variable



Forward



Backward

Variable

- backward(gradient=None, retain_graph=None, create_graph=None, retain_variables=None)

  computes the gradient of current variable w.r.t graph leaves
- detach()

  returns a new Variable, detached from the current graph
- detach_()

  detaches the Variable from the graph that created it
- register_hook(hook)

  registers a backward hook
- reinforce(reward)

  registers a reward obtained as a result of a stochastic process

Variable

backward(gradient, retain_graph, create_graph)

Computes the gradient of current variable w.r.t. graph leaves.

- gradient (Tensor, Variable, None): gradient w.r.t. the variable
- retain_graph (bool): used for double backward
- create_graph (bool): used for higher order derivative products

```
>>>import torch
>>>from torch.autograd import Variable
>>>v = Variable(torch.Tensor([0, 0, 0]), requires_grad=True)
>>>
>>>v.backward(torch.Tensor([1, 1, 1]))
>>>
>>>v.grad.data
1 1 1 [torch.FloatTensor of size 3]
>>>
```

Parameters

torch.nn.Parameter(data, requires_grad=True)

A kind of Variable that is to be considered a module parameter

- data(Tensor): parameter tensor
- requires_grad(bool): if True, the parameter requires gradient

Parameter v.s Variable

Parameters are Variable subclasses, that have very special property when used with Modules – when they are assigned as Module attributes they are automatically added to the list of its parameters.

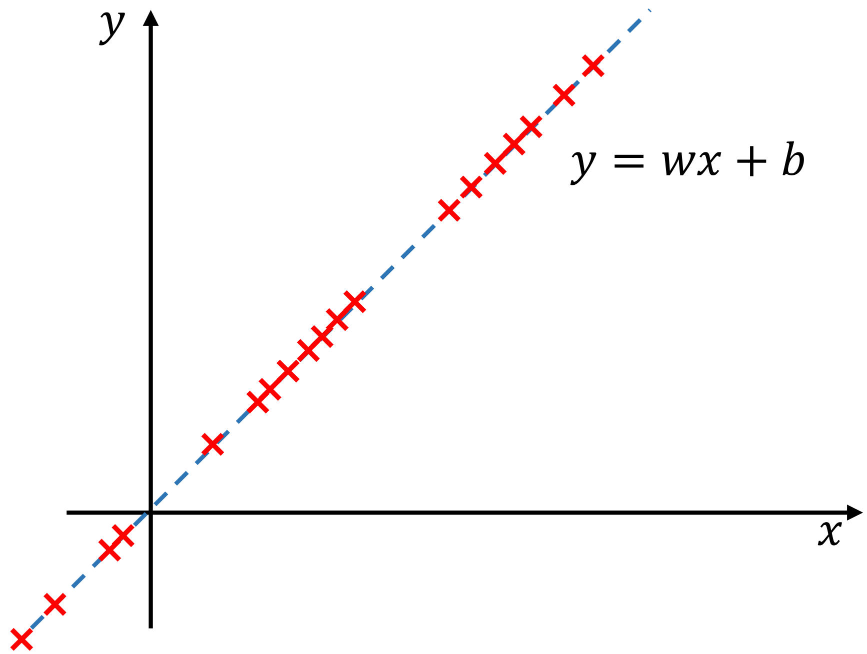
GPU Supported

All operations can be conducted on GPU if the tensor is converted to cuda.tensor

```
>>>import torch
>>>x = torch.randn(3, 3)
>>>x
1.1980 -1.2213  0.5500
0.0943 -0.0436 -1.9253
0.6731  0.4803 -1.5076
[torch.FloatTensor of size 3x3]

>>>x.cuda()
1.1980 -1.2213  0.5500
0.0943 -0.0436 -1.9253
0.6731  0.4803 -1.5076
[torch.cuda.FloatTensor of size 3x3 (GPU 0)]
```

Example

➢ Solve simple regression problem



Prediction: $y = wx + b$

Ground truth: $\hat{y} = x$

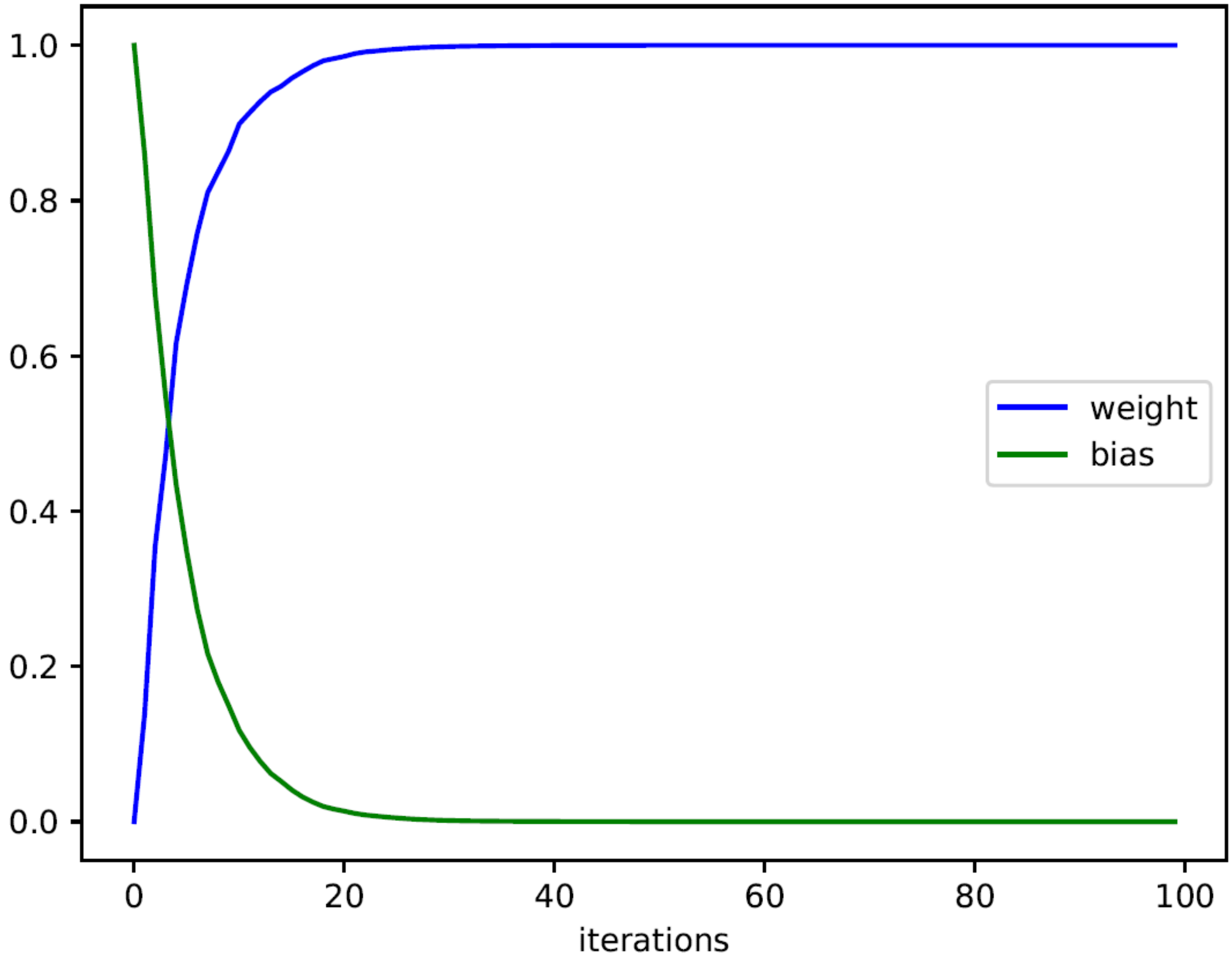
Optimization problem: $\min_{w,b} \|\hat{\mathbf{y}} - \mathbf{y}\|^2$

Example

```python
import torch
from torch.autograd import Variable
from torch.nn import Parameter

weight = Parameter(torch.zeros(1))
bias = Parameter(torch.ones(1))
params_list = [{"params": weight}, {"params": bias}]

for i in range(100):
    x = torch.randn(100)
    x_var = Variable(x)
    prediction = x_var * weight + bias
    # compute loss
    loss = (x_var - prediction).pow(2).sum()

    # backward and update parameters
    optimizer = torch.optim.SGD(params=params_list, lr=0.001)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

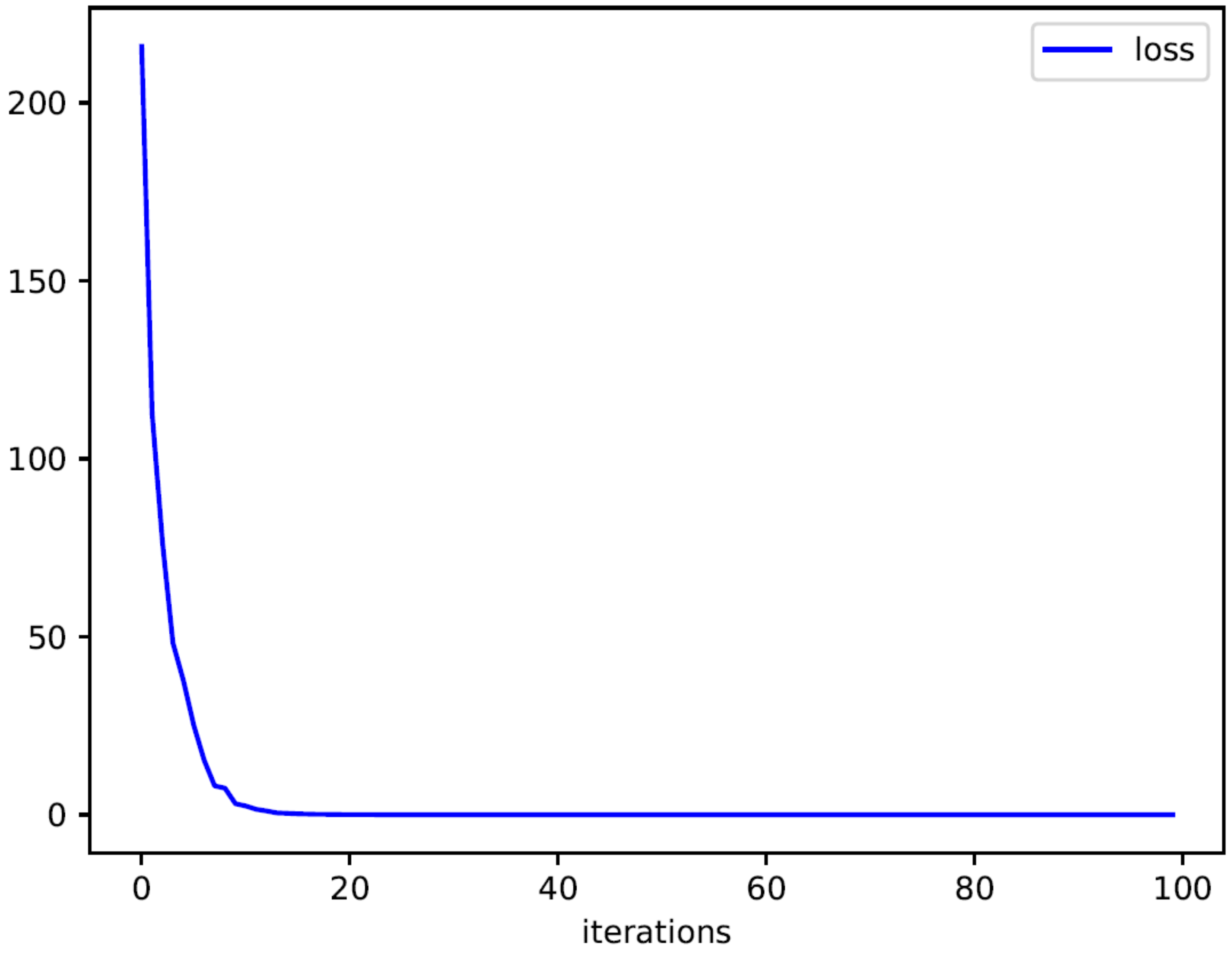
Example

- Weight and bias



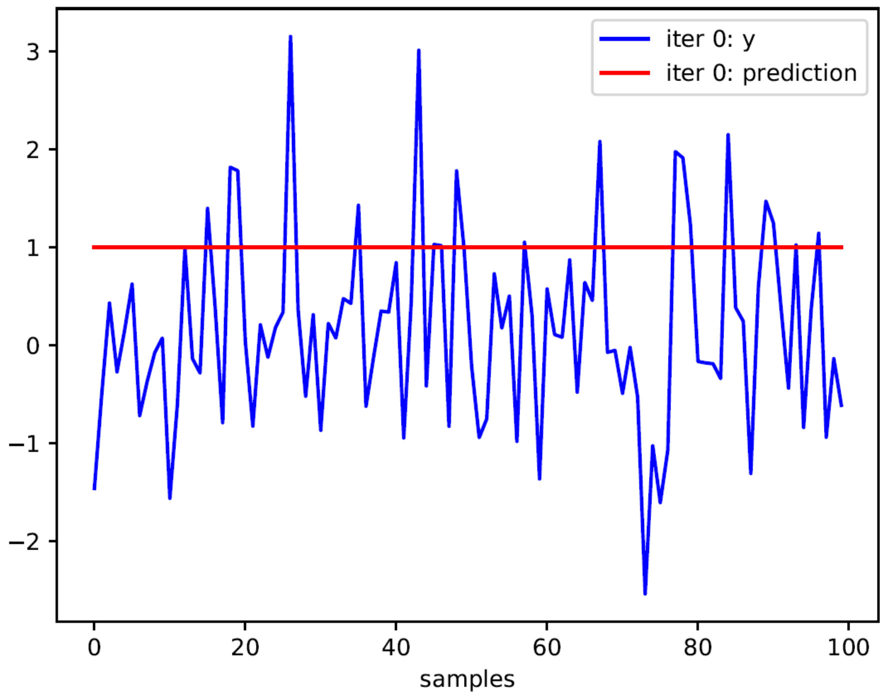
Example

➢ Loss

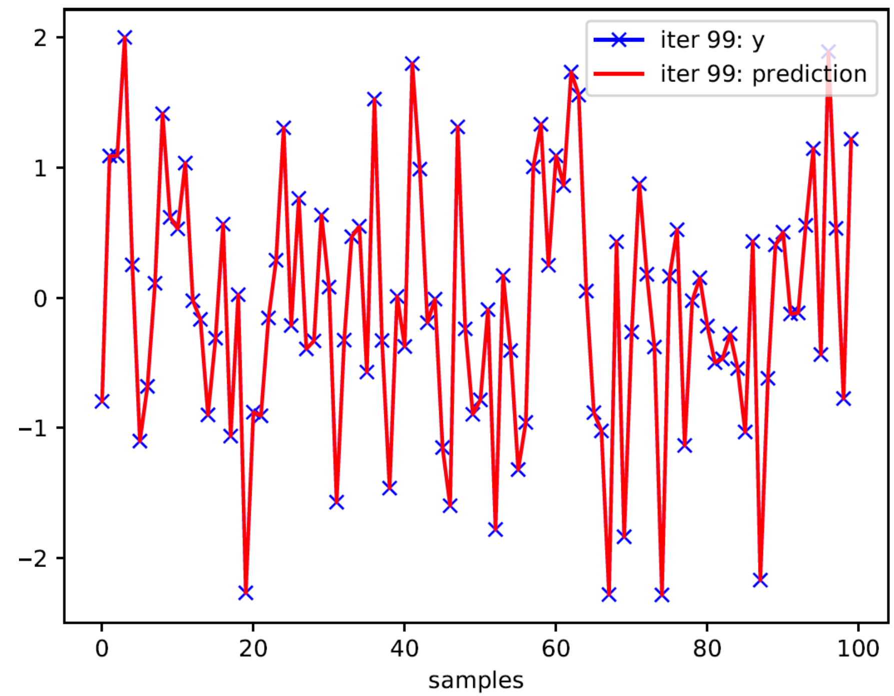


Example

➢ Prediction



Iter 0

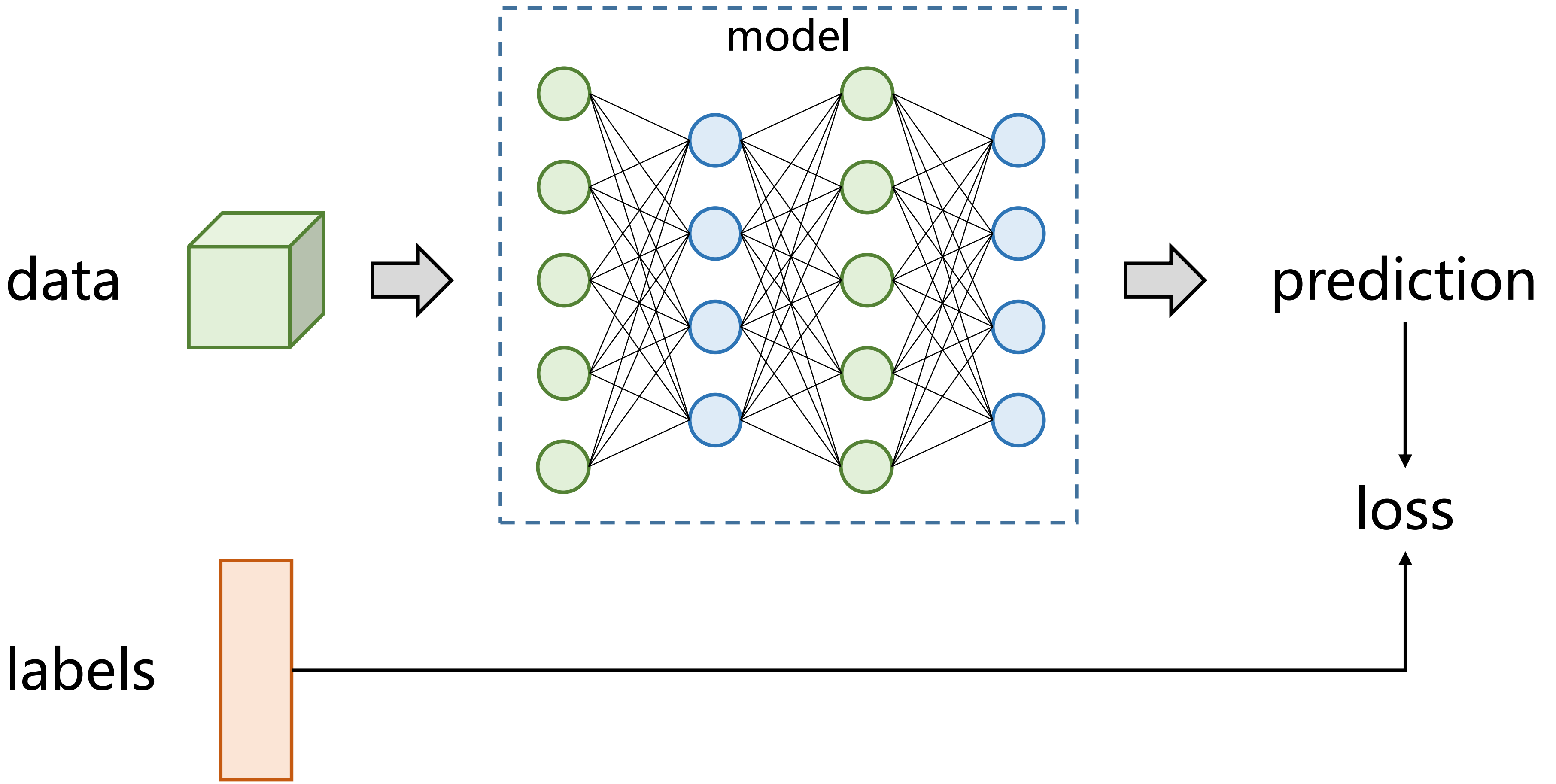


Iter 99

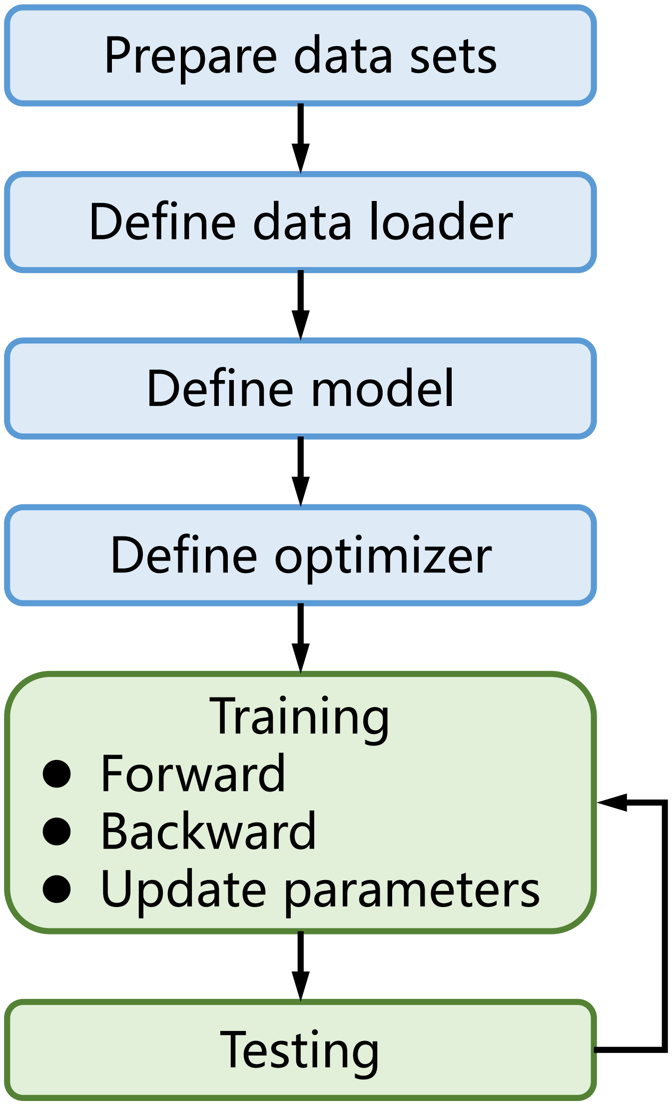
Q & A?

Deep Learning In PyTorch

Framework



Pipeline



➢ Important concepts:

- **Data loader:**

  loading and preprocessing data

- **Model:**

  descripting network structure

- **Optimizer:**

  including SGD, Adam, RMSprop etc.

Data Loader

Define a data loader object for loading and preprocessing data sets from disk or cache by using **torchversion** and **torch.utils.data.DataLoader**

- **torch.utils.data.DataLoader:** class for loading data
- **Torchversion:** package for computer vision tasks, contents state-of-the-art models, data sets, data transformers

Torchvision

The torchvision package consists of popular data sets, model architectures and common image transformations for computer vision

- torchvision.datasets
  - MNIST
  - COCO
  - LSUN
  - Imagenet-12
  - CIFAR
  - STL10
  - **ImageFolder**
- torchvision.transforms
  - Compose
  - CenterCrop
  - RandomCrop
  - RandomHorizontalFlip
  - Normalize
  - ToTensor
  - ...

Data Sets

torchvision.datasets.MNIST(…)

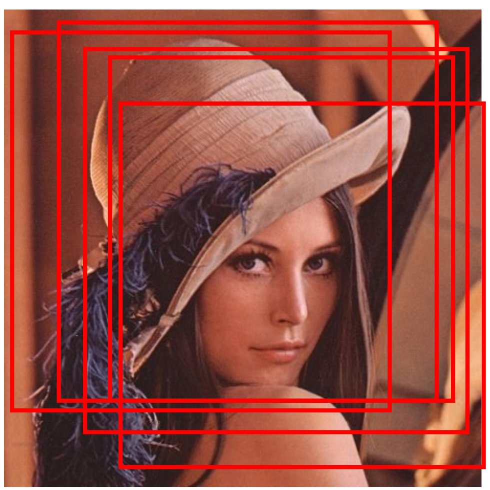
- root(string): root directory of data set
- train(bool): if True, creates data set for training, otherwise for testing
- download(bool): if True, downloads the data set from internet
- transform(callable): function for data transformation
- target_transform: function for target transformation

```
import torchvision.datasets as dset
import torchvision.transforms as transforms

mnist_train = dset.MNIST(("/home/dataset/mnist  “, train=True,
                          download=True,
                          transform=transforms.ToTensor())
```

Data Augmentation

Origin

Horizontal Flip



Random Crop

Center Crop

Data Loader

torch.utils.data.DataLoader(...)

- dataset(Dataset): dataset from which to load data
- batch_size(int): number of samples per batch
- shuffle(bool): if True, reshuffle data at every epoch
- num_workers(int): number of subprocesses to used for data loading
- pin_memory(bool): if True, the data loader will copy tensors into CUDA pinned memory before returning them
- ...

Data Loader

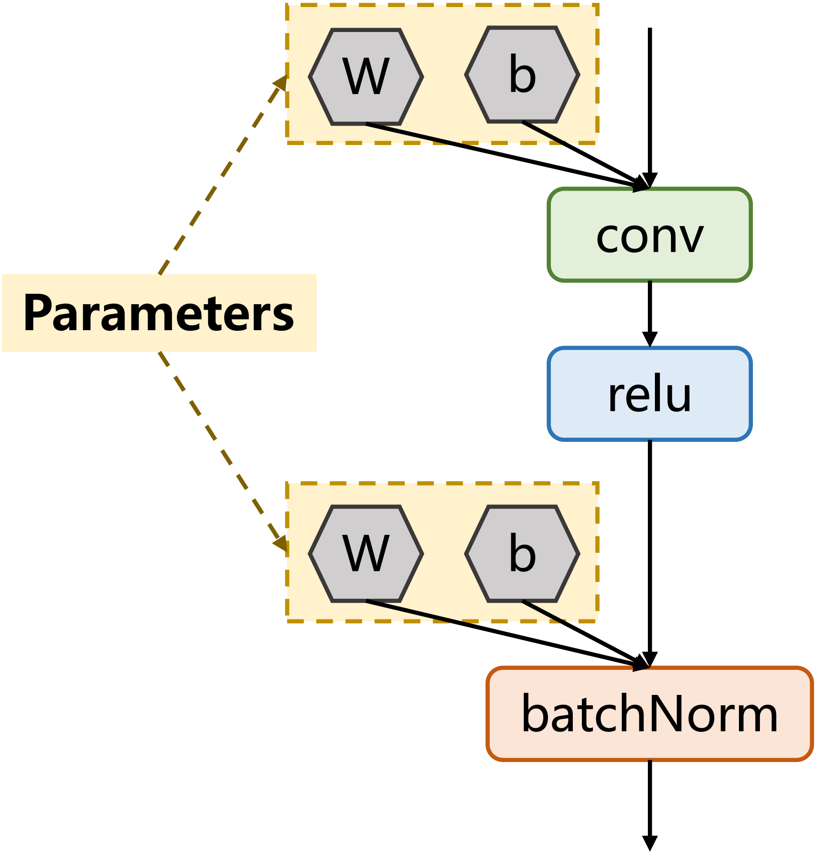
Example of creating data loader for training

```
import torchvision.datasets as dset
import torchvision.transforms as transforms
import torch

train_loader = torch.utils.data.DataLoader(
    dset.MNIST("/home/dataset/mnist", train=True, download=True,
            transform=transforms.Compose([
                transforms.ToTensor(),
                transforms.Normalize(norm_mean, norm_std)
            ])),
    batch_size=self.train_batch_size, shuffle=True,
    num_workers=self.n_threads, pin_memory=False
)
```

Optimizer

torch.optim is a package implementing various optimization algorithm, including SGD, Adam, RMSprop, etc.

- torch.optim.Optimizer
- torch.optim.SGD
- torch.optim.Adadelta
- torch.optim.Adagrad
- torch.optim.Adam
- torch.optim.Adamax
- torch.optim.ASGD
- torch.optim.LBFGS
- torch.optim.RMSprop
- torch.optm.Rprop



Optimizer

torch.optim.Optimizer(params, defaults)

Base class for all optimizers

- params(iterable): an iterable of Variables or dicts. Specifies what Variables should be optimized
- defaults(dict): containing values of optimization options

Functions of optimizer:

- load_state_dict(state_dict): loads the optimizer state
- state_dict(): returns the state of the optimizer as a dict
- step(closure): perform a single optimization step
- zero_grad(): clears the gradients of all optimized Variables

Optimizer

torch.optim.SGD(params, lr, momentum=0, dampening=0, weight_decay=0, nesterov=False)

```
import torch

optimzer = torch.optim.SGD(params=model.parameters(),
                           lr=0.01,
                           momentum=0.9,
                           weight_decay=0.0001,
                           nesterov=True)
```

Optimizer

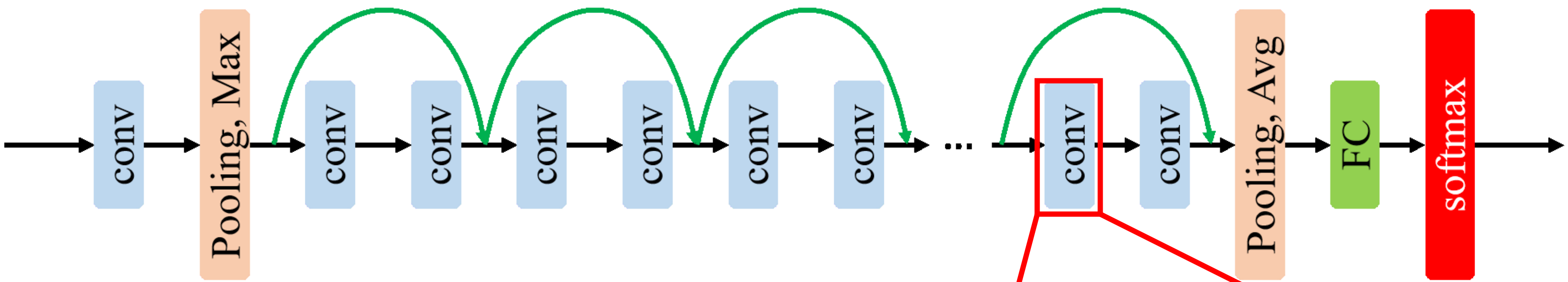
update parameters of model

```
optimizer.zero_grad() # set gradient of parameters to zero
...# backward
optimizer.step() # update parameters
```

update learning rate of optimizer

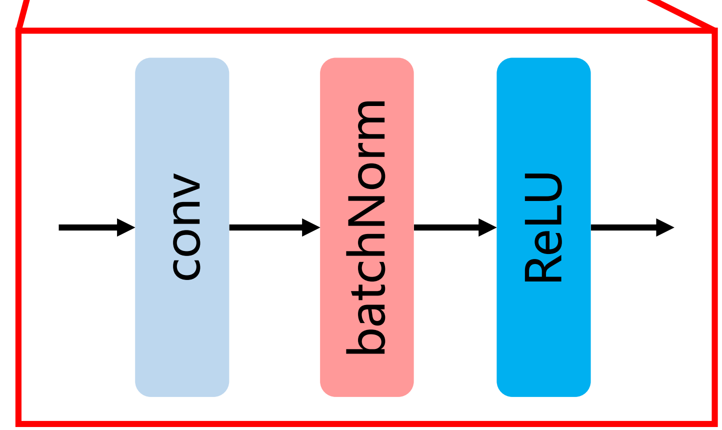
```
for param_group in optimzer.param_groups:
    param_group['lr'] = 0.001
```

Model



Architecture of Residual Network

conv	Convolutional layer
batchNorm	Batch Normalization
ReLU	Activation layer
Pooling	Pooling layer
softmax	Softmax layer
FC	Fully connected layer



Model

torch.nn: package for deep learning

- Parameters
- Containers
- Convolutional layers
- Pooling layers
- Normalization layers
- Linear layers
- Loss functions
- Non-linear Activations
- Recurrent layers
- ...

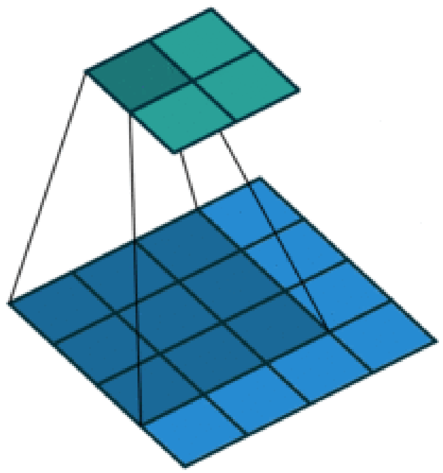
Convolutional Layer

torch.nn.Conv2d(...)

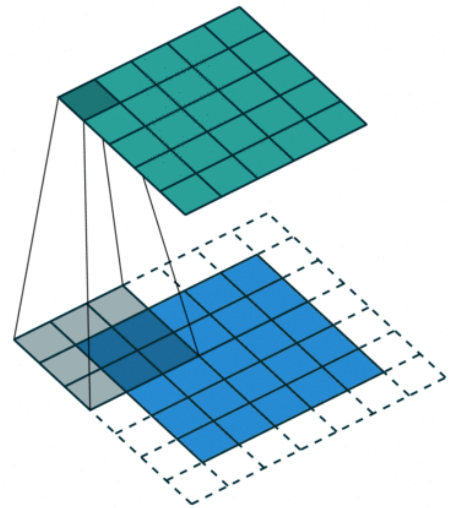
Applies a 2D convolution over an input signal composed of several input planes

- in_channels(int): number of channels in the input image
- out_channels(int): number of channels in the output features
- kernel_size(int or tuple): size of the convolving kernel
- stride(int or tuple): stride of the convolution
- padding(int or tuple): zero-padding added to the input
- bias(bool): if True, adds a learnable bias to the output
- ...

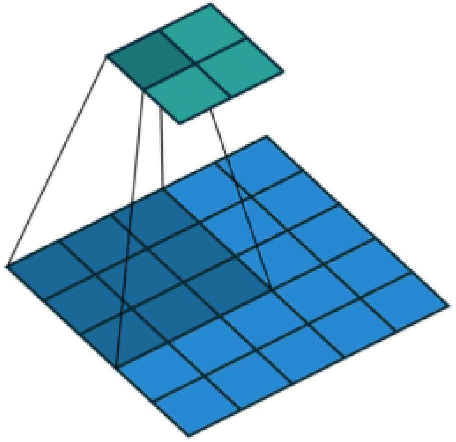
Convolutional Layers



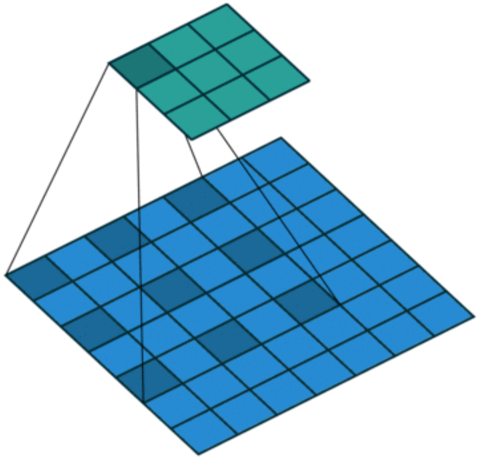
No padding, no strides



Padding = 1



Stride = 2



Dilation = 1

Those pictures are from https://github.com/vdumoulin/conv_arithmetic

Convolutional Layers

input: $(N, C_{in}, H_{in}, W_{in})$ **output:** $(N, C_{in}, H_{in}, W_{in})$

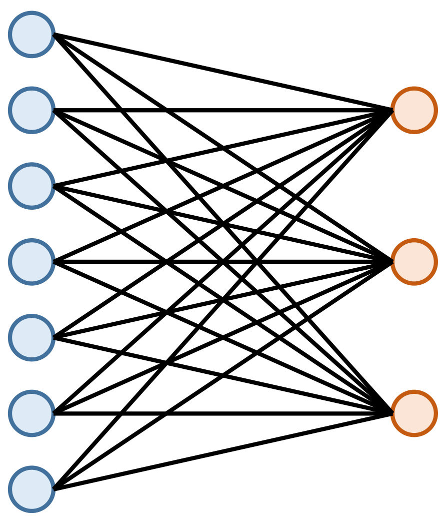
```
>>> import torch.nn as nn
>>> from torch import autograd
>>>
>>> # With square kernels and equal stride
>>> m = nn.Conv2d(16, 33, 3, stride=2)
>>>
>>> # non-square kernels and unequal stride and with padding
>>> m = nn.Conv2d(16, 33, (3, 5), stride=(2, 1), padding=(4, 2))
>>>
>>> input = autograd.Variable(torch.randn(20, 16, 50, 100))
>>> output = m(input)
```

Linear Layers

torch.nn.Linear(in_features, out_features, bias=True)

Applies a linear transformation to the in coming data

- in_features(int): size of each input sample
- out_features(int): size of each output sample
- bias(bool): if True, the layer will learn an additive bias



Linear Layers

input: $(N, in\_features)$ **output:** $(N, out\_features)$

$$\mathbf{Y} = \mathbf{WX} + \boldsymbol{b}$$

```
>>> import torch.nn as nn
>>> from torch import autograd
>>> m = nn.Linear( 20, 30)
>>>
>>> input = autograd.Variable(torch.randn(128, 20))
>>> output = m(input)
>>> print(output.size())
```

Containers

- torch.nn.Module
- torch.nn.Sequential
- torch.nn.ModuleList
- torch.nn.ParameterList

Module

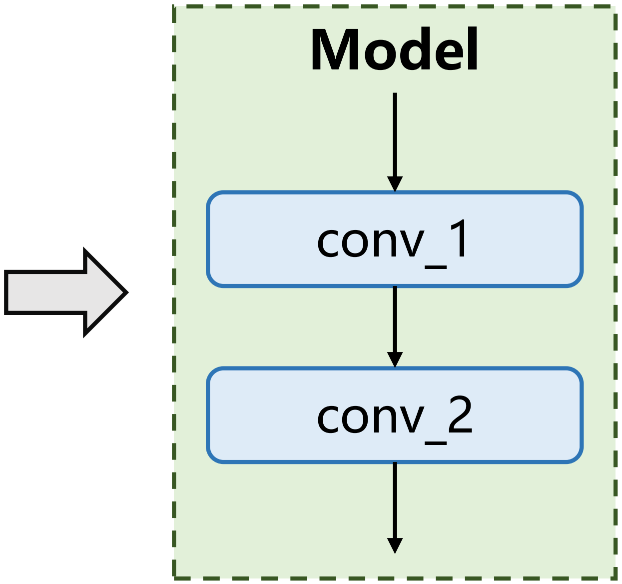
torch.nn.Module

Base class for all neural network modules

- cpu(device_id=None): moves module to the CPU
- cuda(device_id=None): moves module to the GPU
- eval(): sets the module in evaluation mode
- train(mode=True): sets the module in training mode
- zero_grad(): sets gradients of all model parameters to zero
- load_state_dict(state_dict): copies parameters and buffers from state_dict into this module and its descendants
- state_dict(destination=None, prefix=''): returns a dictionary containing a whole state of the module

Module

```
# define your model
import torch.nn as nn
class Model(nn.Module):
    def __init__(self):
        super(Model, self).__init__()
        self.conv_1 = nn.Conv2d(1, 20, 5)
        self.conv_2 = nn.Conv2d(20, 20, 5)

    def forward(self, x):
        out = self.conv_1(x)
        out = self.conv_2(out)
        return out
```

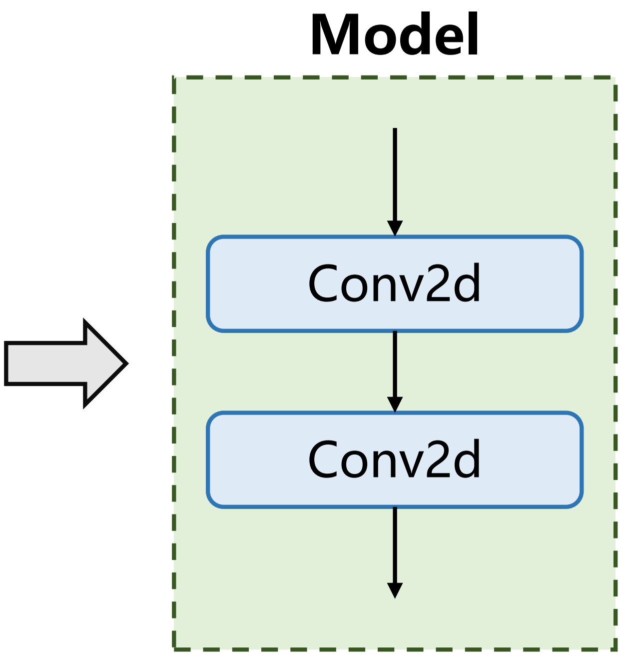


Sequential

torch.nn.Sequential(*args)

A sequential container. Modules will be added to it in the order they are passed in the constructor

```
# define your model
import torch.nn as nn

Model = nn.Sequential(
            nn.Conv2d(1, 20, 5),
            nn.Conv2d(20, 20, 5),
)
```



GPU Supported

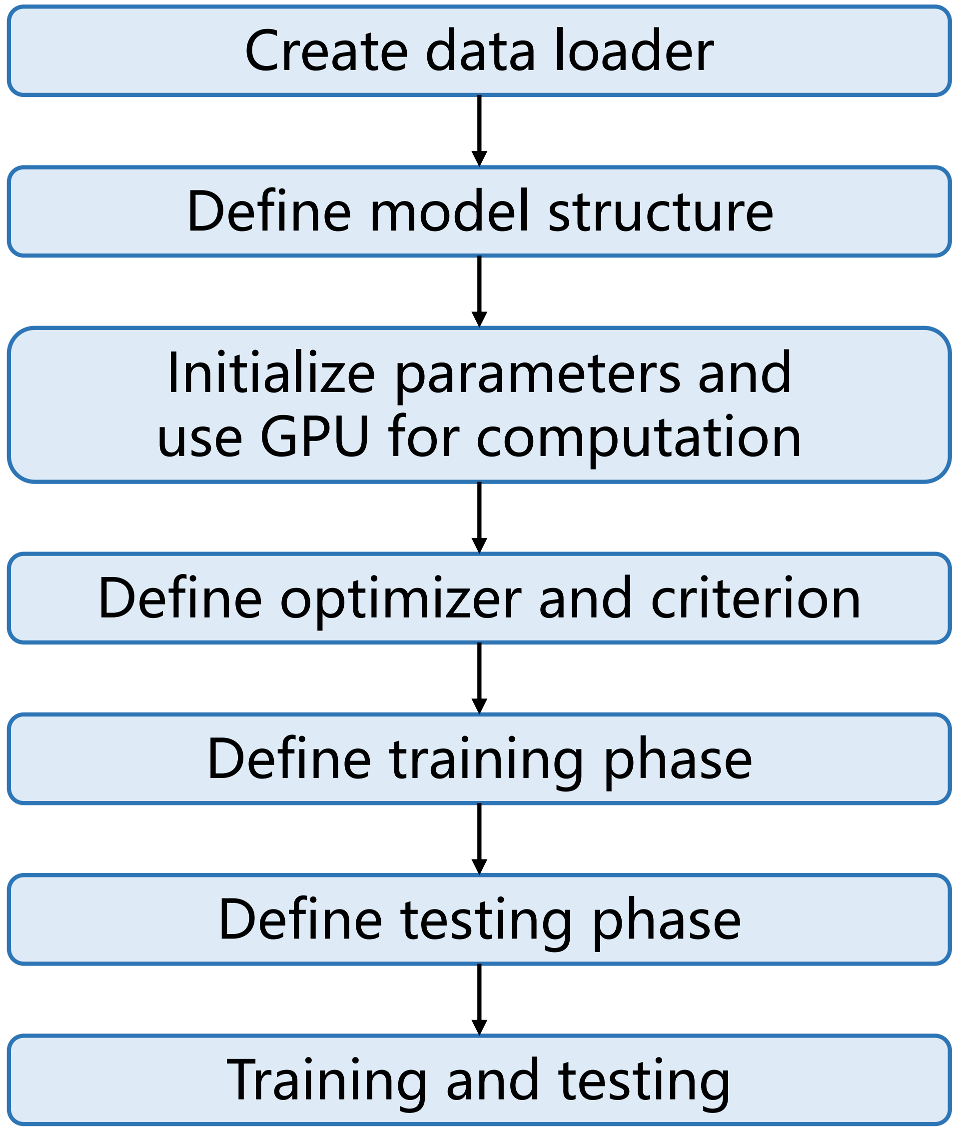
torch.nn.DataParallel(...)

Implements data parallelism at the module level

- module: module to be parallelized
- device_id: CUDA devices, default: all devices
- output_device: device location of output, default: device_id[0]

```
# parallelize your model
net = torch.nn.DataParallel(model, device_ids=[0, 1]).cuda()
output = net(input_var)
```

CNN Example


Create data loader
Define model structure
Initialize parameters and use GPU for computation
Define optimizer and criterion
Define training phase
Define testing phase
Training and testing

CNN Example

➢ Step 1: create data loader

```
import torch
import torch.nn as nn
from torch.autograd import Variable
from torchvision import datasets, transforms
import torch.nn.init as nnInit
train_loader = torch.utils.data.DataLoader(
        datasets.MNIST('../data/', train=True, download=True,
                        transform=transforms.Compose([
                            transforms.ToTensor(),
                            transforms.Normlize((0.1307,), (0.3081, ))
                        ])),
    batch_size=64, shuffle=True, num_workers=4, pin_memory=True)
```

CNN Example

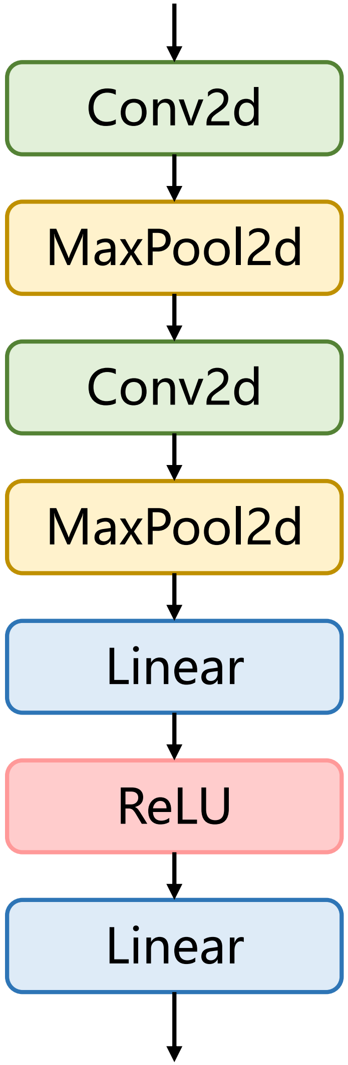
➢ Step 1: create data loader

```
test_loader = torch.utils.data.DataLoader(
        datasets.MNIST('../data/', train=False,
                        transform=transforms.Compose([
                            transforms.ToTensor(),
                            transforms.Normalize((0.1307,), (0.3081, ))
                        ])),
    batch_size=64, shuffle=False, num_workers=4, pin_memory=True)
```

CNN Example

➢ Step 2: define model structure

```
class LeNet5(nn.Module):
    def __init__(self):
        super(LeNet5, self).__init__()
        self.features = nn.Sequential(
                nn.Conv2d(1, 20, 5)
                nn.MaxPool2d(2, 2)
                nn.Conv2d(20, 50, 5)
                nn.MaxPool2d(2, 2))
        self.classifier = nn.Sequential(
                nn.Linear(800, 500)
                nn.ReLU(inplace=True)
                nn.Linear(500, 10))
    def forward(self, x):
        out = self.features(x)
        out = out.view(out.size(0), -1)
        out = self.classifier(out)
        return out
```



CNN Example

➤ Step 3: initialize parameters and use GPU for computation

```python
# create model instance
model = LeNet5()
# initialize weights and bias
for m in model.modules():
    if isinstance(m, nn.Linear) or isinstance(m, nn.Conv2d):
        nnInit.xavier_normal(m.weight)
        if m.bias is not None:
            m.bias.data.zero_()

model.cuda()
```

CNN Example

- Step 4: define optimizer and criterion

```
optimizer = torch.optim.SGD(model.parameters(),
                            lr=0.01,
                            momentum=0.9,
                            weight_decay=1e-4
                            nesterov=True)

criterion = nn.CrossEntropyLoss().cuda()
```

CNN Example

➢ Step 5: define training phase

```
def train(epoch):
    model.train()

    for batch_idx, (data, target) in enumerate(train_loader):
        data, target = Variable(data.cuda()), Variable(target.cuda())
        optimizer.zero_grad()
        output = model(data)
        loss = criterion(output, target)
        loss.backward()
        optimizer.step()
        if batch_idx % 100 == 0:
            print '[training] loss' , loss.data[0]
```

CNN Example

➢ Step 6: define testing phase

```
def test(epoch):
    model.eval ()
    correct = 0

    for batch_idx, (data, target) in enumerate(test_loader):
        data, target = Variable(data.cuda()), Variable(target.cuda())
        output = model(data)
        pred = output.data.max(1)[1]
        correct += pred.eq(target.data).cpu().sum()

    print '[testing] accuracy' : 100.0*correct/len(test_loader.dataset)
```

CNN Example

- Step 7: training and testing

```
for epoch in range(1, 11):
    train(epoch)
    test(epoch)
```

Q & A?

Thank You