

Deep Learning Toolkits II

PyTorch

Tong XIAO

A graph is created on the fly

```
from torch.autograd import Variable

x = Variable(torch.randn(1, 10))
prev_h = Variable(torch.randn(1, 20))
W_h = Variable(torch.randn(20, 20))
W_x = Variable(torch.randn(20, 10))
```



Tensors and Dynamic neural networks in Python with strong GPU acceleration.

PyTorch is a deep learning framework that puts Python first.

We are in an early-release Beta. Expect some adventures.

[Learn More](#)

Tensor computation (like numpy) with GPU

Deep learning with automatic differentiation

Developers

facebook



ParisTech
INSTITUT DES SCIENCES ET TECHNOLOGIE
PARIS INSTITUTE OF TECHNOLOGY

Carnegie
Mellon
University



Digital
Reasoning

Stanford
University



Inria



Outlines

- Examples
 - PyTorch as a fast calculator
 - Train a CNN with PyTorch
- Basic concepts
- Write new models
- HOWTOs

Example: A fast calculator

```
1 import numpy as np
2 import torch
3
4 # Task: compute matrix multiplication C = AB
5 d = 3000
6
7 # using numpy
8 A = np.random.rand(d, d).astype(np.float32)
9 B = np.random.rand(d, d).astype(np.float32)
10 C = A.dot(B)
11
12 # using torch with gpu
13 A = torch.rand(d, d).cuda()
14 B = torch.rand(d, d).cuda()
15 C = torch.mm(A, B)
```



350 ms



0.1 ms

Example: Auto Differentiate

```
1 import torch
2 from torch.autograd import Variable
3
4 # Task: compute  $d(||x||^2)/dx$ 
5 x = Variable(torch.range(1, 5), requires_grad=True)
6 print(x.data) # x.data = [1, 2, 3, 4, 5]
7
8 f = x.dot(x)
9 print(f.data) # f.data = 55
10
11 f.backward()
12 print(x.grad) # x.grad = [2, 4, 6, 8, 10]
```

Example: Train a CNN

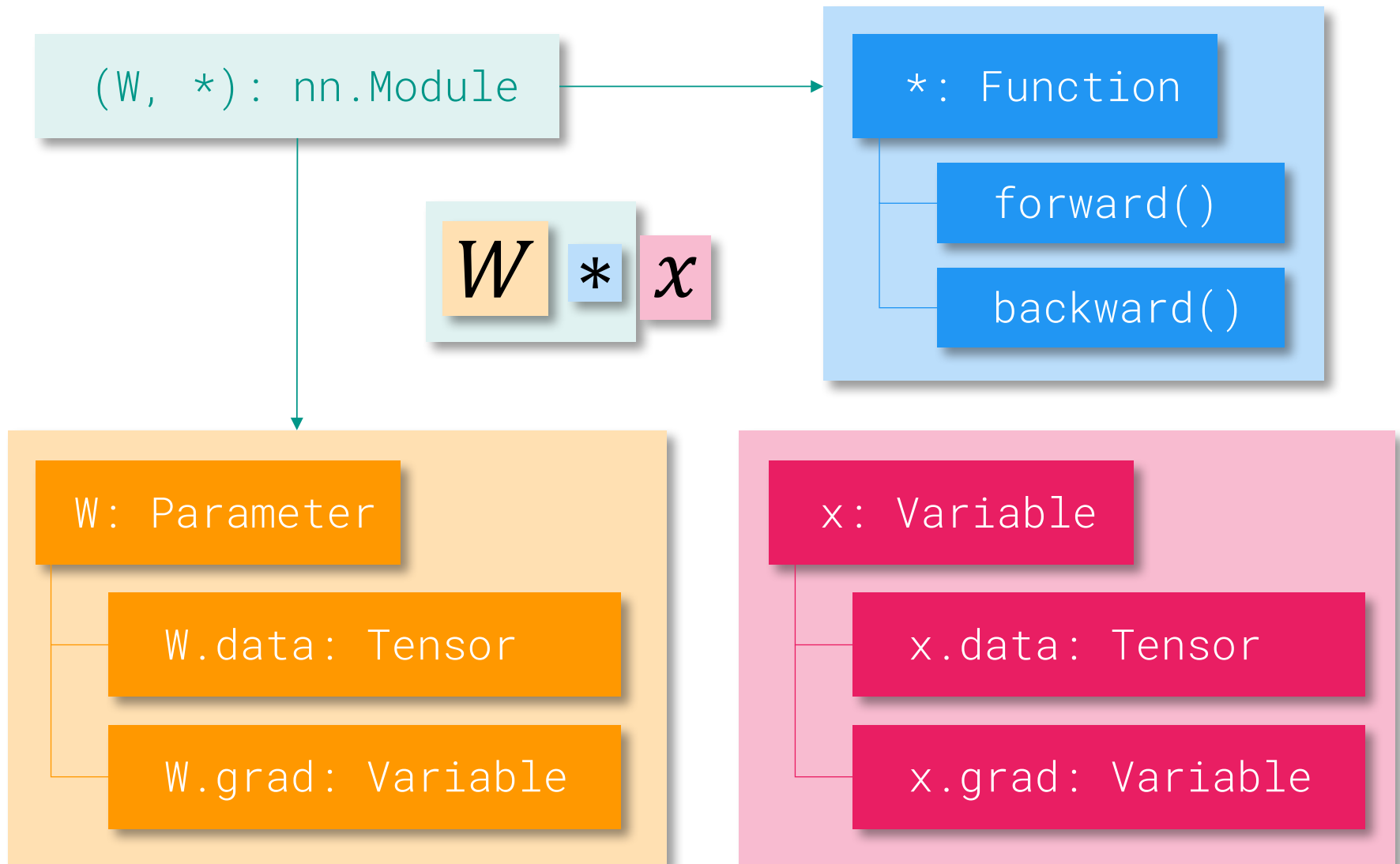
- See our notebook example

<http://nbviewer.jupyter.org/gist/Cysu/320960298b6ccaedb778a2a4de2dd2db>

Basic Concepts

- **torch.Tensor** – similar to `numpy.array`, with GPU
- **autograd.Variable** – wraps a Tensor and enables auto differentiation
- **autograd.Function** – operate on Variables. Implement forward and backward.
- **nn.Parameter** – a special Variable
- **nn.Module** – contain Parameters and define functions on input Variables

Basic Concepts



Write New Models

```
1 from torch import nn
2
3 class SimpleCNN(nn.Module):
4     def __init__(self):
5         super(SimpleCNN, self).__init__()
6
7         self.block = nn.Sequential(
8             nn.Conv2d(3, 16, kernel_size=3, padding=1),
9             nn.BatchNorm2d(16),
10            nn.ReLU(),
11            nn.MaxPool2d(kernel_size=4, stride=4))
12        self.fc = nn.Linear(64*16, 10)
13
14    def forward(self, x):
15        x = self.block(x)
16        x = x.view(x.size(0), -1)
17        x = self.fc(x)
18        return x
```

Group into
nn.Sequential

Write New Models

```
1 from torch import nn
2 import torch.nn.functional as F
3
4 class SimpleCNN(nn.Module):
5     def __init__(self):
6         super(SimpleCNN, self).__init__()
7         self.conv = nn.Conv2d(3, 16, kernel_size=3, padding=1)
8         self.bn = BatchNorm2d(16)
9         self.fc = nn.Linear(64*16, 10)
10
11     def forward(self, x):
12         x = self.conv(x)
13         x = F.relu(x)
14         x = F.max_pool2d(x, 4, stride=4)
15         x = x.view(x.size(0), -1)
16         x = self.fc(x)
17         return x
```

Functional
interface

Not recommended, but
sometimes useful

Write New Models

- Combine high-level `nn.Module` and low-level math operations on `Variable`
- Use functional interfaces smartly, e.g., convolve two feature maps using `F.conv2d`
- See <http://pytorch.org/docs/notes/extending.html> for more details

HOWTOs

- Initialize parameters
- Fine-tuning
- Extract features
- Two-branch modules

Initialize Parameters

```
1 class SimpleCNN(nn.Module):
2     def __init__(self):
3         # Create some modules here
4         ...
5         for m in self.modules():
6             if isinstance(m, nn.Conv2d):
7                 n = m.kernel_size[0] * m.kernel_size[1] * m.out_channels
8                 m.weight.data.normal_(0, math.sqrt(2. / n))
9                 if m.bias is not None:
10                    m.bias.data.zero_()
11            elif isinstance(m, nn.BatchNorm2d):
12                m.weight.data.fill_(1)
13                m.bias.data.zero_()
14            elif isinstance(m, nn.Linear):
15                m.weight.data.normal_(0, 0.001)
16                if m.bias is not None:
17                    m.bias.data.zero_()
```

Any operation that changes a tensor in-place is post-fixed with an _

Fine-tuning

```
1 model = torchvision.models.resnet18(pretrained=True)
2 model.fc = nn.Linear(512, 100)
3
4 new_param_ids = set(id(p) for p in model.fc.parameters())
5 base_params = [p for p in model.parameters() if id(p) not in new_param_ids]
6
7 optimizer = optim.SGD([
8     {'params': base_params},
9     {'params': model.fc.parameters(), 'lr': 0.1}],
10    lr=0.01, momentum=0.9)
```

Replace the classifier

Different lr for pretrained layers and new layers

Extract Features

- Make a new class `FeatureExtractor` that holds a pretrained model
- Redefine the forward function, call the layers in the pretrained model explicitly, save the features we want for output
- Use `model._modules`

Extract Features – model._modules

```
1 OrderedDict([
2   ('conv1', Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)),
3   ('bn1', BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True)),
4   ('relu', ReLU (inplace)),
5   ('maxpool', MaxPool2d (size=(3, 3), stride=(2, 2), padding=(1, 1), dilation=(1, 1))),
6   ('layer1', Sequential (
7     (0): BasicBlock (
8       (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
9       (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True)
10      (relu): ReLU (inplace)
11      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
12      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True)
13    )
14    (1): BasicBlock (
15      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
16      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True)
17      (relu): ReLU (inplace)
18      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
19      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True)
20    )
21  )),
22  ...
```

Extract Features

```
1 class FeatureExtractor(nn.Module):
2     def __init__(self, model, layer_names):
3         super(FeatureExtractor, self).__init__()
4         self._model = model
5         self._layer_names = set(layer_names)
6
7     def forward(self, x):
8         outs = {}
9         for name, module in self._model._modules.items():
10             if isinstance(module, nn.Linear):
11                 x = x.view(x.size(0), -1)
12                 x = module(x)
13                 if name in self._layer_names:
14                     outs[name] = x
15         return outs
16
```

Not included in the `_modules`.
So call it explicitly.

Extract Features

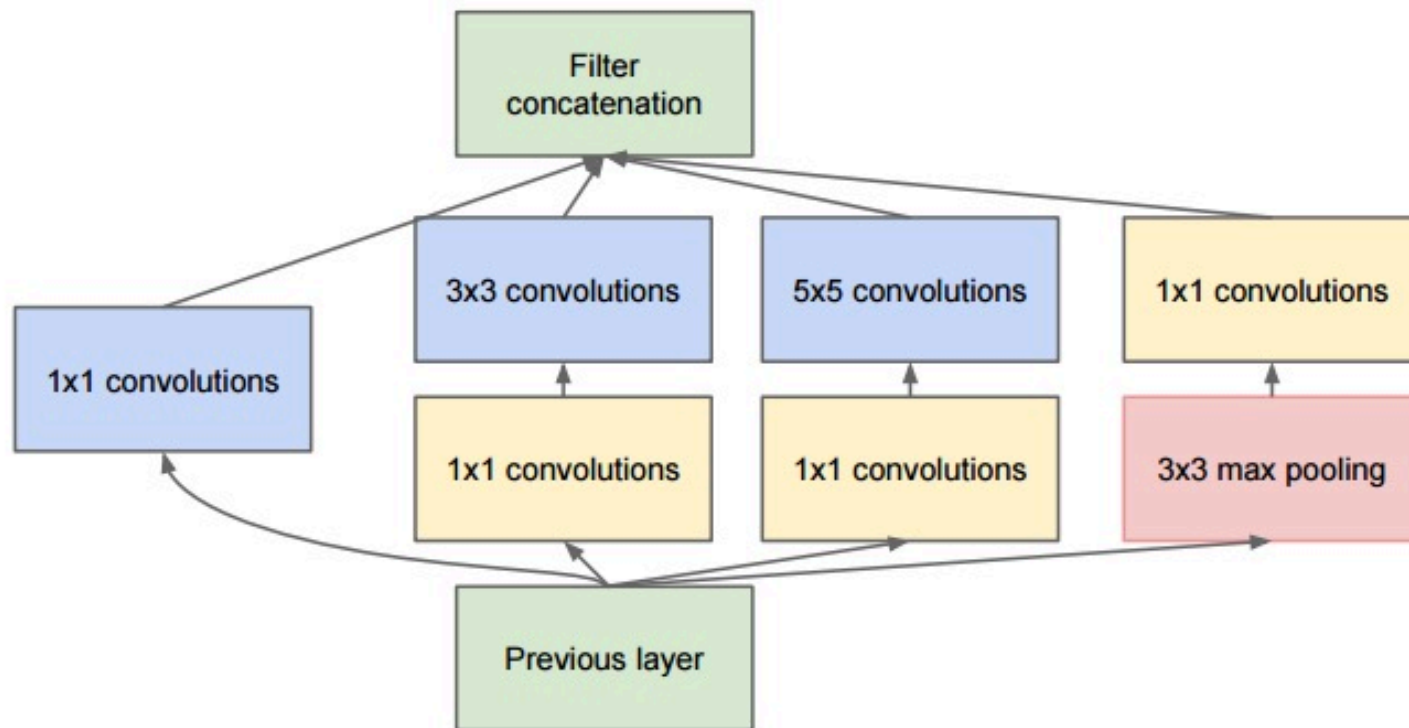
Remember to call `model.eval()`.
Affects BN, Dropout, etc.

```
17 model = torchvision.models.resnet18(pretrained=True).cuda().eval()
18 extractor = FeatureExtractor(model, ['avgpool', 'fc'])
19
20 x = Variable(torch.randn(1, 3, 224, 224).cuda(), volatile=True)
21 outs = extractor.forward(x)
```

Don't call `extractor(x)`
directly

Reduces memory cost significantly!
All non-output Tensors could be corrupted.

Two-branch Modules



Two-branch Modules

```
1 class InceptionBlock(nn.Module):
2     def __init__(self, num_in, num_out):
3         super(InceptionBlock, self).__init__()
4         self.branches = [
5             nn.Sequential(
6                 nn.Conv2d(num_in, num_out, kernel_size=1),
7                 nn.ReLU()),
8             nn.Sequential(
9                 nn.Conv2d(num_in, num_out, kernel_size=1),
10                nn.ReLU(),
11                nn.Conv2d(num_out, num_out, kernel_size=3, padding=1),
12                nn.ReLU()),
13            ...
14        ]
15        for i, branch in enumerate(self.branches):
16            self.add_module(str(i), branch)
17
18    def forward(self, x):
19        # Concatenate branch results along channels
20        return torch.cat([b(x) for b in self.branches], 1)
```

Need to call `add_module` to register the branches!

Summary

- Examples
- Basic concepts
 - torch – Tensor
 - autograd – Variable, Function
 - nn – Module, Parameter
- Write new models
- HOWTOs

More Adventures

- Tutorials
 - official <https://github.com/pytorch/tutorials>
 - jcjohnson <https://github.com/jcjohnson/pytorch-examples>
 - RNN and NLP <https://github.com/spro/practical-pytorch>
- Examples <https://github.com/pytorch/examples>
- Docs <http://pytorch.org/docs/>
- Discussions <https://discuss.pytorch.org/>

Thanks to all the developers!

Tong XIAO



Adam Paszke



Soumith Chintala



Sam Gross

And many others!