Introduction to practical aspects of Deep Learning

Yuping Luo

Introduction to ~~practical aspects of Deep Learning~~ PyTorch

Yuping Luo

Why using a framework?

- Performance;
- Fewer bugs;
- Code reuse (backpropagation, convolution, etc.);
- Community;
- ...

Basic Pipeline

1. (Installation and Import)
2. Data Loading
3. Network Architecture
4. Optimization(train)
5. Evaluation

```python
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.utils.data import DataLoader
from torchvision import datasets, transforms

train_loader = torch.utils.data.DataLoader(
    datasets.MNIST('../data', train=True, download=True,
        transform=transforms.Compose([transforms.ToTensor(),
transforms.Normalize((0.1307,), (0.3081,))])),
    batch_size=32, shuffle=True)
test_loader = torch.utils.data.DataLoader(
    datasets.MNIST('../data', train=False,
        transform=transforms.Compose([transforms.ToTensor(),
transforms.Normalize((0.1307,), (0.3081,))])),
    batch_size=32)

device = torch.device("cpu") # "cuda"

model = nn.Sequential(
    nn.Linear(784, 200),
    nn.ReLU(),
    nn.Linear(200, 10),
    nn.LogSoftmax(dim=-1),
).to(device)

optimizer = optim.SGD(model.parameters(), lr=0.01)

def train():
    model.train()
    for data, target in train_loader:
        data, target = data.to(device).view(-1, 28 * 28), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()

def test():
    model.eval()
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device).view(-1, 28 * 28), target.to(device)
            output = model(data)
            pred = output.max(1, keepdim=True)[1]
            correct += (pred == target.view_as(pred)).sum().item()
    print('accuracy', correct / len(test_loader.dataset))

for epoch in range(100):
    train()
    test()
```

Installation and Import

```
$ pip install torch torchvision
```

```python
import torch        # Basic Tensor operations
import torch.nn as nn      # Modules and layers (class style)
import torch.nn.functional as F      # Modules and layers (function style)
import torch.optim as optim      # Optimizers
from torch.utils.data import DataLoader
from torchvision import datasets, transforms
```

Data Loading

Data preprocessing

```
train_loader = torch.utils.data.DataLoader(
    datasets.MNIST('../data', train=True, download=True,
        transform=transforms.Compose([transforms.ToTensor(),
transforms.Normalize((0.1307,), (0.3081,))])),
    batch_size=32, shuffle=True)
test_loader = torch.utils.data.DataLoader(
    datasets.MNIST('../data', train=False,
        transform=transforms.Compose([transforms.ToTensor(),
transforms.Normalize((0.1307,), (0.3081,))])),
    batch_size=32)
```

Network Architecture and Optimizer

```python
device = torch.device("cpu") # "cuda"

model = nn.Sequential(
    nn.Linear(784, 200),
    nn.ReLU(),
    nn.Linear(200, 10),
    nn.LogSoftmax(dim=-1),
).to(device)

optimizer = optim.SGD(model.parameters(), lr=0.01)
```

Move parameters to `device` (cpu or cuda)

Training

```python
def train():
    model.train()
    for data, target in train_loader:
        data, target = data.to(device).view(-1, 28 * 28), target.to(device)
        optimizer.zero_grad()
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()
```

Use train mode: Dropout/BatchNorm/etc.

compute gradient

apply gradient

Evaluation

+ Average over multiple random seeds!

```python
def test():
    model.eval()
    correct = 0
    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device).view(-1, 28 * 28), target.to(device)
            output = model(data)
            pred = output.max(1, keepdim=True)[1]
            correct += (pred == target.view_as(pred)).sum().item()
    print('accuracy', correct / len(test_loader.dataset))
```

Use train mode: Dropout/BatchNorm/etc.

don’t need gradient

Main Loop

```
for epoch in range(100):
    train()
    test()
```

Run!

nn.Module: Flexible Network Architecture

A container of parameters.

Nested module

```
class MyLinear(nn.Module):
    def __init__(self, n_in, n_out):
        super().__init__()
        self.fc = nn.Linear(n_in, n_out, bias=False)
        self.bias = nn.Parameter(torch.zeros(n_out))

    def forward(self, x):
        return self.fc(x) + torch.exp(self.bias)
```

nn.Module: Flexible Network Architecture

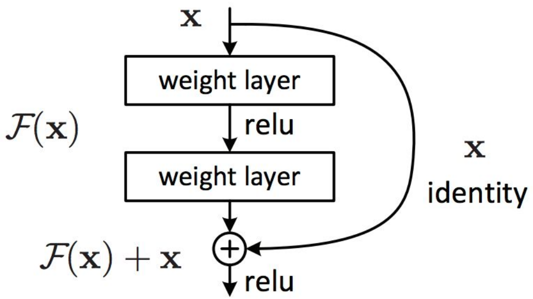


Figure 2. Residual learning: a building block.

He, Kaiming, et al.

```python
class ResBlock(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(100, 100)
        self.fc2 = nn.Linear(100, 100)

    def forward(self, x):
        y = self.fc1(x)
        y = F.relu(y)
        y = self.fc2(y)
        return F.relu(y + x)

net = nn.Sequential(
    # ...
    ResBlock(),
    ResBlock(),
    # ...
)
```

Inplace Operations

Do NOT use inplace operations if you require grads.

Live examples!

GPU

1. GPU/CPU interaction is slow.
2. Large batch size.
   a. Data parallel (mostly used)
   b. Async gradient update (ASGD, etc.)
3. Floating point: 32-bit (`float`) vs 64-bit (`double`) vs 16-bit (`half`)
4. `nvidia-smi`
5. Async preprocessing (by CPU).

Reproducibility

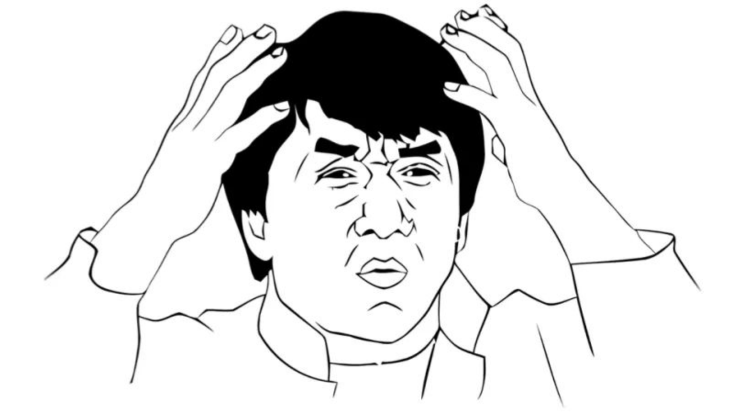
1. Easier to debug.
2. Fix random seed!

```
np.random.seed(seed)
tf.set_random_seed(np.random.randint(2**30))
torch.manual_seed(np.random.randint(2**30))
random.seed(np.random.randint(2**30))
torch.cuda.manual_seed_all(np.random.randint(2**30))
```

3. Make a copy of source code / command lines.

...TensorFlow

1. Fewer calls to `sess.run` due to large overhead in TensorFlow.
2. Debug?
   a. `tf.Print`
   b. `sess.run('Add:0')`
   c. ...or eager mode

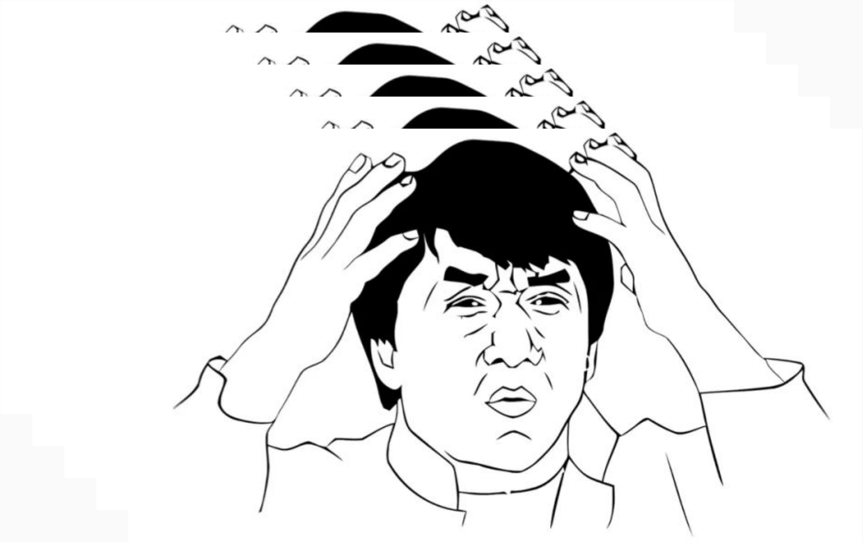


Overfitting

1. Regularization (L2, etc.);
2. Dropout;
3. Data augmentation;
4. Smaller network;
5. Early stop;
6. ...

Hyperparameter Tuning

1. Coordinate Descent;
2. Grid Search;
3. Random Search;
4. ...



Advanced Techniques

Hessian-vector product

- Why not storing the whole matrix?

- Quadratic Form $f(x) \approx f(x_0) + (x - x_0)^\top \nabla f(x)|_{x_0} + \frac{1}{2}(x - x_0)^\top \nabla^2 f(x)|_{x_0} (x - x_0)$
  - Minimizer $x = H^{-1} \nabla f(x)|_{x_0}$
  - Conjugate Gradient only requires to compute Hv

Hessian-vector product

$$\mathbf{H}v = \frac{\partial}{\partial x}\left(v^{\mathsf{T}}\boxed{\frac{\partial f}{\partial x}}\right)$$

```python
def hessian_vec_prod(f, params, v: torch.Tensor) -> torch.Tensor:
    grads = torch.autograd.grad(f, params, create_graph=True)

    dot = nn.utils.parameters_to_vector(grads).mul(v).sum()
    grads = [g.contiguous() for g in torch.autograd.grad(dot, params,
retain_graph=True)]
    return nn.utils.parameters_to_vector(grads)
```

symbolic

Hessian-vector product

$$\mathbf{H}v = \frac{\partial}{\partial x}\left(\boxed{v^{\top}\frac{\partial f}{\partial x}}\right)$$

```python
def hessian_vec_prod(f, params, v: torch.Tensor) -> torch.Tensor:
    grads = torch.autograd.grad(f, params, create_graph=True)

    dot = nn.utils.parameters_to_vector(grads).mul(v).sum()
    grads = [g.contiguous() for g in torch.autograd.grad(dot, params, retain_graph=True)]
    return nn.utils.parameters_to_vector(grads)
```

Hessian-vector product

$$\mathbf{H}v = \boxed{\frac{\partial}{\partial x}\left(v^{\mathsf{T}}\frac{\partial f}{\partial x}\right)}$$

```python
def hessian_vec_prod(f, params, v: torch.Tensor) -> torch.Tensor:
    grads = torch.autograd.grad(f, params, create_graph=True)

    dot = nn.utils.parameters_to_vector(grads).mul(v).sum()
    grads = [g.contiguous() for g in torch.autograd.grad(dot, params, retain_graph=True)]
    return nn.utils.parameters_to_vector(grads)
```

Gradient Checkpointing

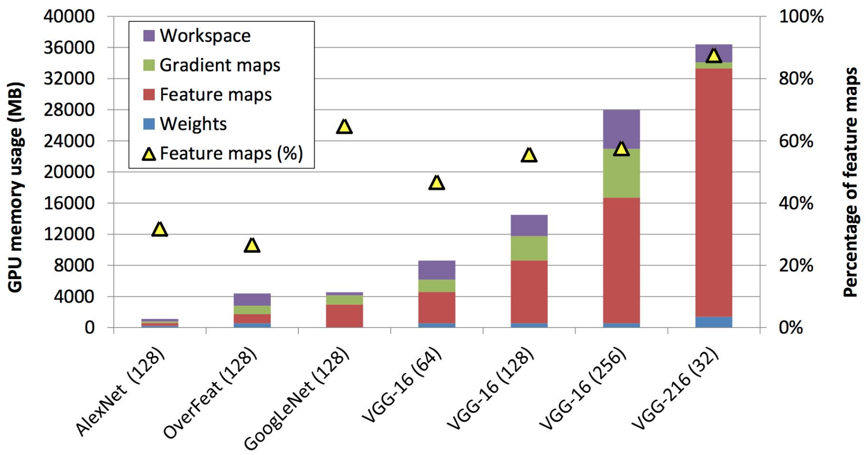
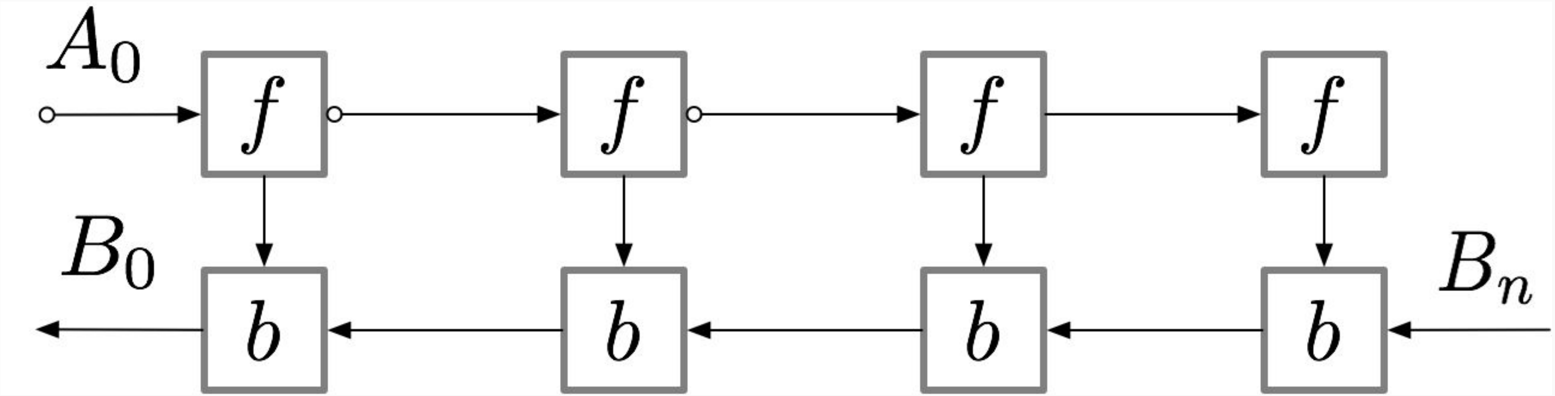


Fig. 4: Breakdown of GPU memory usage based on its functionality (left axis). The right axis shows the fraction of allocated memory consumed by feature maps.

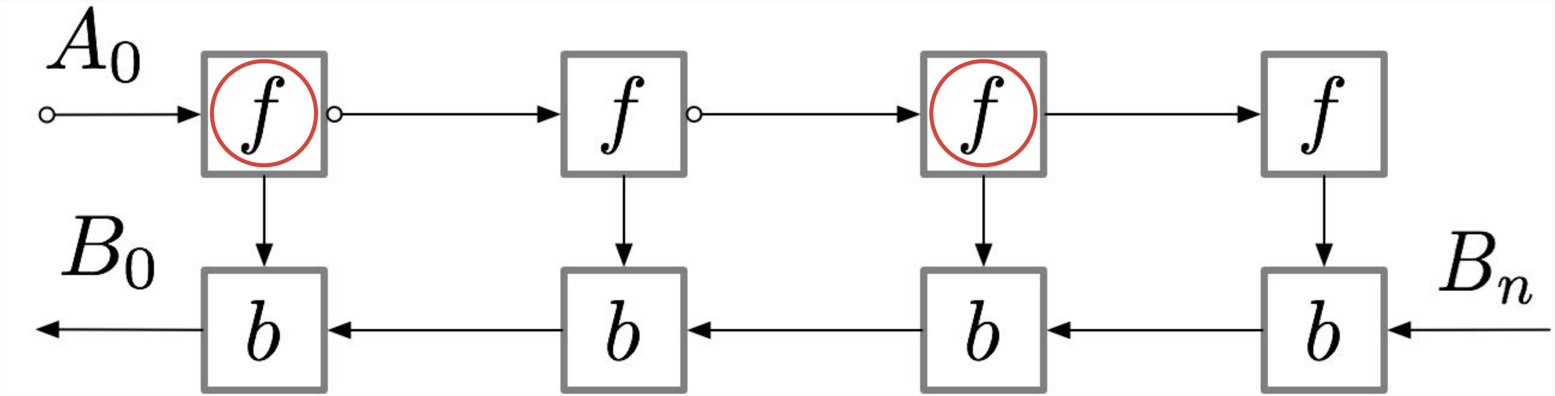
From [Rhu, Minsoo, et al.]

Gradient Checkpointing

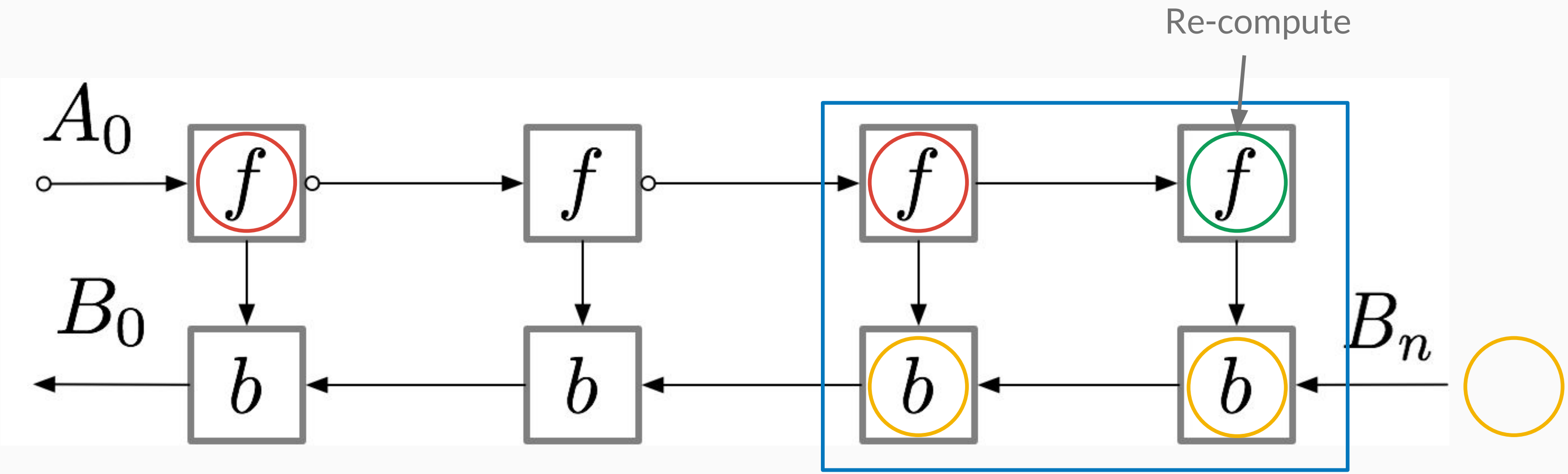


https://github.com/openai/gradient-checkpointing

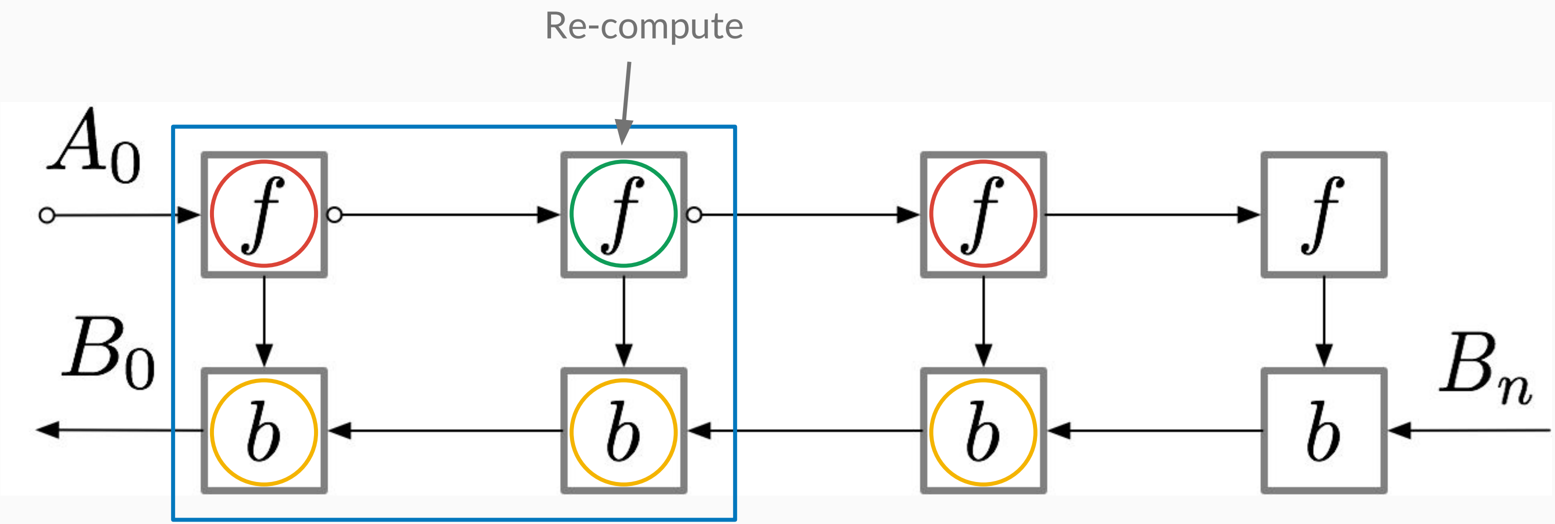
Gradient Checkpointing



Gradient Checkpointing



Gradient Checkpointing



Gradient Checkpointing