Mini tuto PyTorch

Mathieu Lefort

9 septembre 2024

Framework de deep learning

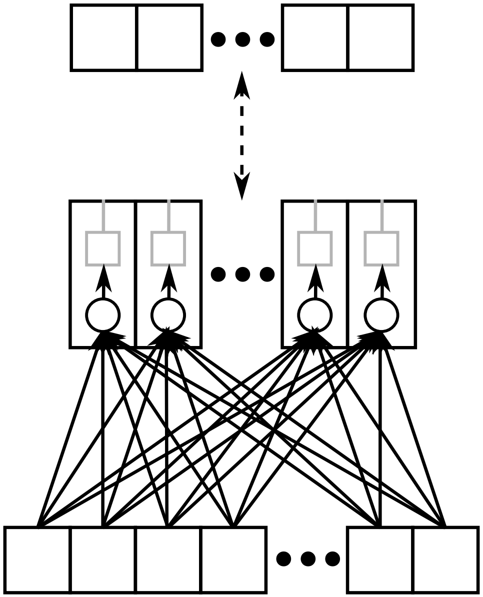
Intérêts

- différentiation automatique (sur arbre de dépendance de variables)
- facilité de développement
- partage du code

Existants

- Theano (Université de Montréal)
- PyTorch (Facebook)
- Tensorflow (Google)
- Keras

Modèle du perceptron



labels

fonction de coût

couche de neurones

poids + biais

entrées

Les tenseurs

Hyperparamètres

```
batch_size = 5, nb_epochs = 10, eta = 0.00001, w_min = -0.001, w_max = 0.001
```

Modèle

```
w = torch.empty((data.shape[1],label.shape[1]))
b = torch.empty((1,label.shape[1]))
torch.nn.init.uniform_(w,w_min,w_max)
torch.nn.init.uniform_(b,w_min,w_max)
```

Données

```
(data,label) = torch.load('mnist.pkl'))
indices = numpy.arange(data.shape[0],step=batch_size)
for n in range(nb_epochs):
    numpy.random.shuffle(indices)
    for i in indices:
        x = data[i:i+batch_size]
        t = label[i:i+batch_size]
```

Activité

```
        y = torch.mm(x,w)+b
```

Apprentissage

```
        grad = (t-y)
        w += eta * torch.mm(x.T,grad)
        b += eta * grad.sum(axis=0)
```

Les loaders

Hyperparamètres

```
batch_size = 5, nb_epochs = 10, eta = 0.00001, w_min = -0.001, w_max = 0.001
```

Modèle

```
w = torch.empty((data.shape[1],label.shape[1]))
b = torch.empty((1,label.shape[1]))
torch.nn.init.uniform_(w,w_min,w_max)
torch.nn.init.uniform_(b,w_min,w_max)
```

Données

```
(data,label) = torch.load('mnist.pkl'))
dataset = torch.utils.data.TensorDataset(data,label)
loader = torch.utils.data.DataLoader(dataset, batch_size, shuffle=True)
for n in range(nb_epochs):
    for x,t in train_loader:
```

Activité

```
        y = torch.mm(x,w)+b
```

Apprentissage

```
        grad = (t-y)
        w += eta * torch.mm(x.T,grad)
        b += eta * grad.sum(axis=0)
```

La différentiation automatique

Hyperparamètres

```
batch_size = 5, nb_epochs = 10, eta = 0.00001, w_min = -0.001, w_max = 0.001
```

Modèle

```
w = torch.empty((data.shape[1],label.shape[1]),requires_grad=True)
b = torch.empty((1,label.shape[1]),requires_grad=True)
torch.nn.init.uniform_(w,w_min,w_max)
torch.nn.init.uniform_(b,w_min,w_max)
```

Données

```
(data,label) = torch.load('mnist.pkl'))
dataset = torch.utils.data.TensorDataset(data,label)
loader = torch.utils.data.DataLoader(dataset, batch_size, shuffle=True)
for n in range(nb_epochs):
    for x,t in train_loader:
```

Activité

```
        y = torch.mm(x,w)+b
```

Apprentissage

```
        loss = ((t-y).pow(2)).sum()
        loss.backward()
        with torch.no_grad():
            w -= eta*w.grad
            b -= eta*b.grad
            w.grad.zero_()
            b.grad.zero_()
```

Les couches

Hyperparamètres

```
batch_size = 5, nb_epochs = 10, eta = 0.00001, w_min = -0.001, w_max = 0.001
```

Modèle

```
model = torch.nn.Linear(data_train.shape[1],label_train.shape[1])
torch.nn.init.uniform_(model.weight,w_min,w_max)
```

Données

```
(data,label) = torch.load('mnist.pkl'))
dataset = torch.utils.data.TensorDataset(data,label)
loader = torch.utils.data.DataLoader(dataset, batch_size, shuffle=True)
for n in range(nb_epochs):
    for x,t in train_loader:
```

Activité

```
        y = model(x)
```

Apprentissage

```
        loss = ((t-y).pow(2)).sum()
        loss.backward()
        with torch.no_grad():
            w -= eta*w.grad
            b -= eta*b.grad
            w.grad.zero_()
            b.grad.zero_()
```

Les optimiseurs

Hyperparamètres

```
batch_size = 5, nb_epochs = 10, eta = 0.00001, w_min = -0.001, w_max = 0.001
```

Modèle

```
model = torch.nn.Linear(data_train.shape[1],label_train.shape[1])
torch.nn.init.uniform_(model.weight,w_min,w_max)
loss_func = torch.nn.MSELoss(reduction='sum')
optim = torch.optim.SGD(model.parameters(), lr=eta)
```

Données

```
(data,label) = torch.load('mnist.pkl'))
dataset = torch.utils.data.TensorDataset(data,label)
loader = torch.utils.data.DataLoader(dataset, batch_size, shuffle=True)
for n in range(nb_epochs):
    for x,t in train_loader:
```

Activité

```
        y = model(x)
```

Apprentissage

```
        loss = loss_func(t,y)
        loss.backward()
        optim.step()
        optim.zero_grad()
```

Et pour les CNNs

Couches de convolution

```
torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride, padding,
bias=True)
```

Couche de *max pooling*

```
torch.nn.MaxPool2d(kernel_size, stride, padding)
```

Ou fonction de *max pooling*

```
torch.nn.functional.max_pool2d(input, kernel_size, stride, padding)
```

Les modules

```
import torch
import torch.nn as nn
import torch.nn.functional as F

class MonModele(nn.Module):
```

Constructeur

```
    def __init__(self):
        super(MonModele, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, 5)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)
```

Constructeur

```
    def forward(self, x):
        x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = torch.flatten(x, 1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

```
model = MonModele()
```