Micro tuto Pytorch

Mathieu Lefort

26 novembre 2020

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

Les tenseurs

Hyperparamètres

```
batch_size = 5, nb_epochs = 10, eta = 0.00001
```

Modèle

```
w = torch.empty((data.shape[1],label.shape[1]))
b = torch.empty((1,label.shape[1]))
torch.nn.init.uniform_(w,-0.001,0.001)
torch.nn.init.uniform_(b,-0.001,0.001)
```

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
```

Modèle

```
w = torch.empty((data.shape[1],label.shape[1]))
b = torch.empty((1,label.shape[1]))
torch.nn.init.uniform_(w,-0.001,0.001)
torch.nn.init.uniform_(b,-0.001,0.001)
```

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
```

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,-0.001,0.001)
torch.nn.init.uniform_(b,-0.001,0.001)
```

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
```

Modèle

```
model = torch.nn.Linear(data_train.shape[1],label_train.shape[1])
torch.nn.init.uniform_(model.weight,-0.001,0.001)
```

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
```

Modèle

```
model = torch.nn.Linear(data_train.shape[1],label_train.shape[1])
torch.nn.init.uniform_(model.weight,-0.001,0.001)
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()
```