Pytorch

Bibliothèque

- Issue de `torch` initialement écrite en `LuaJIT` encapsulant une implémentation en C, créé à l'EPFL.
- Propose une couche objet dont les instances peuvent être sérialisées.
- La version `Pytorch` est un framework en `Python` incluant toutes les fonctionnalités de `torch`.
- Une des différences majeure avec `TensorFlow` est le calcul dynamique de graphe en plus de la correction de certains bugs.
- Se substitue à `numpy` et permet le calcul sur GPU de facon assez transparente.
- Accès et stockage des données grâce à la fonction `DataLoader`

Eléments de base

Tensor

- La notion de `tensor` remplace les `ndarrays` de numpy.
- Création d'un `tensor` :

$$torch.empty(5,3) \qquad torch.zeros(5,3)$$

- Accès à un élément : $t[0,0].item()$
- Addition de deux tensors : $t1 + t2$
- Ajoute $t2$ à $t1$ en modifiant $t1$ sur place : $t1.add_(t2)$
- Ajout d'une constante à tous les éléments : $t1 + 2$
- Convesion `nmupy` vers `pytorch` et inversement :

$$t = torch.from\_numpy(ar)$$

$$ar = t.numpy()$$

Définir un modèle de Réseau

Définir la classe

Pytorch tutoriel sur CIFAR10

```
# Define model
 class Net(nn.Module):
  def __init__(self):
    super().__init__()
    self.conv1 = nn.Conv2d(3, 6, 5)
    self.pool = nn.MaxPool2d(2, 2)
    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)
```

Définir un modèle de Réseau

Définir le déplacement des données dans le réseau

```
    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        # flatten all dimensions except batch
        x = torch.flatten(x, 1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
net =Net()
print(net)
```

Définir un modèle de Réseau

Fonction de perte

```
#Choisir la fonction de perte
criterion = nn.CrossEntropyLoss()
```

Optimiseur

```
#Choisir l'optimiseur
optimizer = optim.SGD(net.parameters(), lr=1e-3)
```

Définir un modèle de Réseau

Fonction d'apprentissage

```
for epoch in range(2):  # loop over the dataset multiple time

    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
```

Définir un modèle de Réseau

Sauver un modèle

```
PATH = './cifar_net.pth'
torch.save(net.state_dict(), PATH)
```

Tester un modèle

```
dataiter = iter(testloader)
images, labels = next(dataiter)
# print images
imshow(torchvision.utils.make_grid(images))
print('GroundTruth: ', ' '.join(f'{classes[labels[j]]:5s}
```

Définir un modèle de Réseau

Charger des données

```
transform = transforms.Compose(
            [transforms.ToTensor(),
              transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

batch_size = 4

trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size,
                                shuffle=True, num_workers=2)

testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=batch_size,
                                         shuffle=False, num_workers=2)

 classes = ('plane', 'car', 'bird', 'cat',
         'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
```

Définir un modèle de Réseau

Visualiser

```
def imshow(img):
  img = img / 2 + 0.5     # unnormalize
  npimg = img.numpy()
  plt.imshow(np.transpose(npimg, (1, 2, 0)))
  plt.show()


# get some random training images
dataiter = iter(trainloader)
images, labels = next(dataiter)

# show images
imshow(torchvision.utils.make_grid(images))
# print labels
print(' '.join(f'{classes[labels[j]]:5s}'for j in range(batch_size)))
```

Définir un modèle de Réseau

GPU ou CPU

```
device = torch.device('cuda:0' if torch.cuda.is_available()
#chargement du modèle
net.to(device)
#chargement des données
inputs, labels = data[0].to(device), data[1].to(device)
```