Deep-learning avec {torch} AIDE-MÉMOIRE

Intro

{torch} est basé sur PyTorch, un environnement de Deep-learning utilisé à grande échelle par les chercheurs.

{torch} profite de l'accélération matérielle de la carte GPU avec une interface conviviale, et couvre un grand nombre de cas d'usage, et pas seulement d'apprentissage profond, grâce à sa flexibilité et l'accès possible à l'interface de bas-niveau.

Il est au centre d'un écosystem de paquets logiciels prévus pour l'interface avec des types de donnée spécifiques comme {torchaudio} pour les données temporelles, {torchvision} pour les images, et {tabnet} pour les données tabulaires. Enfin {luz} offre une interface de programmation de plus haut niveau.

Définir
- Des modules NN
- Les instancier en un réseau

→ **Ajouter des paramètres**
- Optimiseur
- Fonction de coût
- Métrique

→ **Boucle d'entraînement**
model$train()
- Taille de batch
- Epochs
- Séparation de la Validation

→ **Évaluation**
model$eval()
- Erreur sur la Validation
- Courbe d'entraînement

→ **Inference**
- modalité
- probabilité

https://torch.mlverse.org/

https://mlverse.shinyapps.io/torch-tour/

INSTALLATION

Le package {torch} s'appuie sur la librairie C++ libtorch. Les prérequis s'installent entièrement depuis R

https://torch.mlverse.org/docs/articles/installation.html

```
install.packages("torch")
library(torch)
install_torch()
```

Voir ?install_torch pour les instructions avec GPU

La manipulation des modèles

DEFINIR UN MODULE NN

```
dense ← nn_module(
  "couche_dense_sans_biais",
  initialize = function(in_f, out_f) {
    self$w ← nn_parameter(torch_randn(in_f, out_f))
  },
  forward = function(x) {
    torch_mm(x, self$w)
  }
)
```

Crée un nn module nommé couche_dense_sans_biais

ASSEMBLER DES MODULES EN RESEAU

model ← dense(4, 3)
Instancie un réseau avec une seule couche

```
model ← nn_sequential(
    dense(4,3), nn_relu(), nn_dropout(0.4),
    dense(3,1), nn_sigmoid())
```

Instancie un réseau avec une sequence de couches

ENTRAINER UN MODÈLE

model$train()
Active la mise a jour des gradients

```
with_enable_grad({
  y_pred ← model(trainset)
  loss ← (y_pred - y)$pow(2)$mean()
  loss$backward()
})
```

Code détaillé de la boucle d'entrainement (alternatif)

EVALUER UN MODEL

```
model$eval()
```
Ou
```
with_no_grad({
  model(validationset)
})
```
Passe d'inférence du modèle sans mise à jour de gradient

OPTIMISATION

optim_sgd()
Optimiseur de descente de gradient stochastique

optim_adam()
Optimiseur ADAM

FONCTION DE COÛT POUR LA CLASSIFICATION

nn_cross_entropy_loss()
nn_bce_loss()
nn_bce_with_logits_loss()
(Binaire) fonction de coût d'entropie croisée
nn_nll_loss()
Fonction de coût de log-vraisemblance négative
nn_margin_ranking_loss()
nn_hinge_embedding_loss()
nn_multi_margin_loss()
nn_multilabel_margin_loss()
(Multinomial) (multi etiquette) fonction de coût de Hinge

FONCTION DE COÛT POUR LA REGRESSION

nn_l1_loss()
Fonction de coût L1
nn_mse_loss()
MSE loss nn_ctc_loss()
Connectionist Temporal Classification loss
nn_cosine_embedding_loss()
Fonction de coût de plongement cosinus
nn_kl_div_loss()
Fonction de coût de divergence de Kullback-Leibler
nn_poisson_nll_loss()
Fonction de coût de log-vraisemblance negative de Poisson

OTHER MODEL OPERATIONS

summary() Résume le modèle

torch_save(); torch_load() enregistre/ restaure le modèle dans un fichier

load_state_dict()
Restaure un modèle enregistré depuis python

Couches de réseaux de neurones

COUCHE PRINCIPALES

nn_linear()
Ajoute une couche de transformation linéaire à une entrée

nn_bilinear() à deux entrées

nn_sigmoid(), nn_relu()
Applique une fonction d'activation à une sortie

nn_dropout()
nn_dropout2d()
nn_dropout3d()
Applique une élimination de connexions à l'entrée

nn_batch_norm1d()
nn_batch_norm2d()
nn_batch_norm3d()
Applique une normalisation par batch aux paramètres

COUCHES DE CONVOLUTION

nn_conv1d() 1D, e.g. convolution temporelle

nn_conv_transpose2d() convolution 2D transposée (déconvolution)

nn_conv2d() 2D, e.g. convolution spatiale sur les images

nn_conv_transpose3d() convolution 3D transposée (déconvolution)
nn_conv3d() 3D, e.g. convolution spatiale sur les volumes

nnf_pad()
Couche de remplissage avec des zero

COUCHES D'ACTIVATION

nn_leaky_relu()
Activation lineaire rectifiée avec fuite

nn_relu6()
Activation lineaire rectifiée avec limitation à 6

nn_rrelu()
Activation lineaire rectifiée avec fuite, aléatoire

nn_elu(), nn_selu()
Activation lineaire exponentielle, idem mise a l'échelle

COUCHES D'EXTRACTION

nn_max_pool1d()
nn_max_pool2d()
nn_max_pool3d()
Extraction des maximum de 1D à 3D

nn_avg_pool1d()
nn_avg_pool2d()
nn_avg_pool3d()
Extraction des moyennes de 1D à 3D

nn_adaptive_max_pool1d()
nn_adaptive_max_pool2d()
nn_adaptive_max_pool3d()
Extraction adaptative des maximum

nn_adaptive_avg_pool1d()
nn_adaptive_avg_pool2d()
nn_adaptive_avg_pool3d()
Extraction adaptative des moyennes

COUCHES RÉCURENTES

nn_rnn()
Reseau récurent de noeuds entièrement connectés avec sortie réinjectée à l'entrée

nn_gru()
Unité récurrente à porte- Cho et al

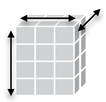
nn_lstm()
Unité à mémoire longue-courte - Hochreiter 1997

Manipulation des tenseurs

CRÉATION DE TENSEUR

tt <- torch_rand(4,3,2) distrib. uniforme
tt ← torch_randn(4,3,2) distrib. normale unitaire
tt ← torch_randint(1,7,c(4,3,2)) distrib. uniforme d'entiers dans **[**1,7**)**
Création d'un tenseur de la dimension voulue.

tt ← torch_ones(4,3,2)
torch_ones_like(a)
Création d'un tenseur unitaire de la dimension voulue ou de la dimension de 'a'. Voire **torch_zeros, torch_full, torch_arange,…**



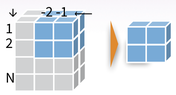
tt$shape **tt$ndim** **tt$dtype**
[1] 4 3 2 [1] 3 torch_Float
tt$requires_grad **tt$device**
[1] FALSE torch_device(type='cpu')
Dimension et attributes d'un tenseur

tt$stride() Saut de valeurs nécessaire entre deux dimensions du tenseur
[1] 6 2 1

tt ← torch_tensor(a,
dtype=torch_float(), device= "cuda")
a ← as.matrix(tt$to(device="cpu")
Copie une matrice 'a' de R dans un tenseur de reels en GPU et vice-versa

EXTRACTION DE VALEURS

tt[1:2, -2:-1,]
Extrait un tenseur 3D
tt[5:N, -2:-1, ..]
Extrait un tenseur 3D ou plus, N le 'dernier'



tt[1:2, -2:-1, 1:1]
tt[1:2, -2:-1, 1, keep=TRUE]
Extrait un tenseur 3D en conservant la dimension unitaire.

tt[1:2, -2:-1, 1]
La dimension unitaire est supprimée par défaut.

tt[tt > 3.1]
Filtrage booléen (le résultat est plat)

CONCATENATION DE TENSEURS

torch_stack()
Pile de tenseurs

torch_cat()
Assemble les tenseurs

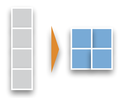
torch_split(2)
Découpe en sections de taille 2

torch_split(c(1,3,1))
Découpe en sections de taille explicites

OPÉRATION SUR LES DIMENSIONS

tt$unsqueeze(1) **torch_unsqueeze(tt,1)**
Ajoute une premiere dimension unitaire à "tt"

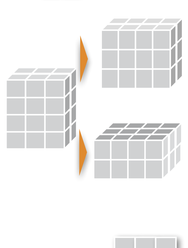
tt$squeeze(1) **torch_squeeze(t,1)**
Supprime la premiere dimension unitaire de "tt"



torch_reshape() **$view()**
Change les dimensions du tenseur
Avec copie ou (potentiellement) sans

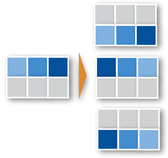
torch_flatten()
aplatit le tenseur

torch_transpose()



torch_movedim(c(1,2))
Inverse la dimension 1 avec 2

torch_movedim(c(1,2,3), c(3,1,2))
deplace dim 1 en 3, dim 2 en 1, dim 3 en 2
torch_permute(c(3,1,2))
Idem uniquement sur la base de la destination



torch_flip(1) retourne les valeurs de la dim 1

torch_flip(2) 2

torch_flip(c(1,2)) des deux dim

OPÉRATION SUR LES VALEURS DU TENSEUR

+, - , *
Operations entre deux tenseurs

$pow(2), $log(), $exp(),
$abs(), $floor(), $round(), $cos(),
$fmod(3), $fmax(1), $fmin(3)
torch_clamp(tt, min=0.1, max=0.7)
Operations sur chaque élément

$eq(), $ge(), $le()
Comparaison sur chaque élément

$to(dtype = torch_long())
Transformation des types

$sum(dim=1), $mean(), $max()
Fonctions d'agrégation sur un tenseur
$amax()

torch_repeat_interleave()
Répétition d'un tenseur n fois

torch luz torchaudio torchvision

UN MODELE DE RECONNAISANCE D'IMAGE SUR MNIST 5041

Mon premier modèle avec {torch}

Modèles pré-entraînés

Les modèles pré-entraînés sont des réseaux d'architecture classiques mis à disposition avec leurs paramètres pré-entraînés. On peut les utiliser pour une prediction directe, une extraction de vecteur de plongement, ou l'entraînement fin.

MODÈLES R NATIFS

library(torchvision)
resnet34 ← model_resnet34(pretrained=TRUE)
Modèles de classification d'image Resnet

resnet34_headless ← nn_prune_head(resnet34, 1)
Élagage de la dernière couche d'un modèle

IMPORT DE MODÈLES PYTORCH

{torchvisionlib} permet d'importer un modèle PyTorch sans avoir à en recoller les modules nn en R.
On le fait en deux étapes:

1- Dans python, instancier , scripter, et sauvegarder:

```
import torch
import torchvision

model = torchvision.models.segmentation.\
    fcn_resnet50(pretrained = True)
model.eval()

scripted_model = torch.jit.script(model)
torch.jit.save(scripted_model, "fcn_resnet50.pt")
```

2- charger le fichier de modèle dans R:

```
library(torchvisionlib)
model ← torch::jit_load("fcn_resnet50.pt")
```

Résolution de problème

HELPERS

with_detect_anomaly()
Fournit une vue interne du comportement du nn_module()

Callbacks

An callback est une ensemble de fonction à appliquer à un moment précis de la boucle d'entraînement du modèle.
Ils permettent par exemple une vue des états interne ou une collecte de statistiques sur le modele au cours de l'entraînement

```
# Datasets d'images MNIST
library(torchvision)
train_ds ← mnist_dataset(  root = " ~/.cache" ,
    download = TRUE,
    transform = torchvision::transform_to_tensor
)
test_ds ← mnist_dataset(   root = " ~/.cache",
    train = FALSE,
    transform = torchvision::transform_to_tensor
)
train_dl ← dataloader(train_ds, batch_size = 32,
    shuffle = TRUE)
test_dl ← dataloader(test_ds, batch_size = 32)

# definition du modèle et de ses couches
net ← nn_module(
 "Net",
 initialize = function() {
  self$fc1 ← nn_linear(784, 128)
  self$fc2 ← nn_linear(128, 10)
 },
 forward = function(x) {
  x %>% torch_flatten(start_dim = 2) %>%
   self$fc1() %>% nnf_relu() %>%
   self$fc2() %>% nnf_log_softmax(dim = 1)
 }
)
model ← net()
# definition de l'optimizer
optimizer ← optim_sgd(model$parameters, lr = 0.01)
# boucle d'entraînement (fit)
for (epoch in 1:10) {
 train_losses ← c()
 test_losses ← c()
 for (b in enumerate(train_dl)) {
  optimizer$zero_grad()
  output ← model(b[[1]]$to(device = device))
  loss ← nnf_nll_loss(output, b[[2]]$to(device = device))
  loss$backward()
  optimizer$step()
  train_losses ← c(train_losses, loss$item())
 }
 for (b in enumerate(test_dl)) {
  model$eval()
  output ← model(b[[1]]$to(device = device))
  loss ← nnf_nll_loss(output, b[[2]]$to(device = device))
  test_losses ← c(test_losses, loss$item())
  model$train()
 }
}
```