L'éco-système du Machine Learning

Ludovic Denoyer et Nicolas Baskiotis

1er semestre 2016-2017

Ce document a pour vocation d'aider les étudiants du M2 DAC à appréhender les différentes plateformes de l'apprentissage statistique auxquels ils seront confrontés. Il est en cours de rédaction et le restera pour un certain temps [1]

1 Python et sklearn

2 LUA/Torch

Torch (`http://torch.ch/`) est un framework basé sur le langage LuaJIT (`http://luajit.org/`), une implémentation de Lua. Il propose plein de modules orientés apprentissage et optimisation avec un support GPU, et plus spécifiquement une implémentation efficace de tenseurs (matrices multi-dimensionnelles) et de réseaux de neuronnes très modulaire.

2.1 LUA

Lua est écrit en C, c'est un langage dynamiquement typé, utilisé principalement comme extension ou langage de script. Lua (enfin surtout LuaJit) est bien plus rapide que python.

Quelques particularités :

- une seule structure de données native : le tableau associatif ;
- closure : une fonction contient son propre environnement lexical (toutes les variables locales ou non sont passées à la fonction). Attention, c'est vraiment très particulier ! une variable globale peut être changée par un appel de fonction ...
- fonction first-class : une fonction est une variable comme une autre (peut être retournée, passée en paramètre, ...) ;
- multi-task simple par le biais des coroutines
- et plein d'autres trucs sympas : ramasse-miette, module dynamique, tail calls,...

Documents :

1. Un des auteurs découvre une partie de l'éco-système, il se peut donc qu'il y ait des inexactitudes voir de grossières erreurs. Pour toute remarque, correction, complément, envoyez un email aux auteurs, ce sera grandement apprécié.

— le bouquin : `https://www.lua.org/pil/contents.html` (peut être trouvé en ebook)
— le tuto : `http://tylerneylon.com/a/learn-lua/`
— avancé : `http://www.lua.org/gems/`
— le manuel officiel : `http://www.lua.org/manual/`
— des tonnes de tutoriels : `http://lua-users.org/wiki/TutorialDirectory`

2.2 Torch

Torch est un framework développé sur LuaJit. En gros, il vise à simplifier la vie des machinelearneurs qui ont envie de faire des réseaux de neurones bien profonds sur GPUs, parallèlisables, flexibles, complexes, embarqués, ... Mais pas seulement : beaucoup de packages pour l'image, la vidéo, la visualisation, ...

Documents :
— le tuto : `http://torch.ch/docs/getting-started.html` et `http://torch.ch/docs/five-simple-examples.html`
— le tuto de 60min : `https://github.com/soumith/cvpr2015/blob/master/Deep%20Learning%20with%20Torch.ipynb`
— Tutos : `https://github.com/torch/torch7/wiki/Cheatsheet`
— Torch for python users : `https://github.com/torch/torch7/wiki/Torch-for-Numpy-users`
— Torch for matlab users : `https://github.com/atamahjoubfar/Torch-for-Matlab-users/blob/master/Torch_for_Matlab_users.pdf`

2.2.1 Les bases

gestion de packages
`luarocks [install paquet | search paquet | list]` permet d'installer, chercher et lister les paquets installés.

Creation et manipulation de tenseurs
Rappel : un tenseur est un tableau multi-dimensionnel. C'est l'objet de base en torch.

```
t = torch.Tensor(2,3)  -- creation d'un tenseur 2x3
u = torch.Tensor({{1,2,3},{4,5,6}})
v = torch.randn(2,3) -- matrice aleatoire
torch.range(0,9)  --
torch.linspace(1,4,6)
t:zero() -- remise a zero
t:fill(2.3) -- remplir avec la meme valeur
u:type() t:type() -- attention aux differences de type
u:typeAs(t) u:type(t:type()) -- pour convertir
u:nDimension() u:size() u:size(1) -- connaitre les dimensions
t:reshape(3,2)
```

```
 -- acceder aux elements/colonnes/etc
 -- attention ! indice de depart 1!!
 -- attention !  pour toutes ces fonctions pas de nouveau tenseur,
 -- tous partage la meme memoire !!!
u[1][2] u[{1,2}]
u[{1,{1,3}}]
u[{{},{1,3}}]
u:narrow(1,1,1) -- retourne tenseur extrait dim x index x size
u:sub(1,2,1,2) -- debutxfinx debutxfin ...
u:select(1,2)  -- dim x index
u:t() u:transpose(1,2) -- transpose
u:permute(2,1) -- permutation

-- copie d'un tenseur
u:copy() u:clone()

torch.cat(u,v,1) -- concatenation selon 1 dimension
```

Opération sur les tenseurs

```
u:nonzero() -- retourne les indices non zero
u:sort() u:sort(2) -- retourne triage, indices
t:trace()
t:sum() t:mean() t:std()
t:min() t:max() t:median()
t:cumsum() t:cumprod()
t:apply(math.sin)  -- appliquer une fonction
t:map(u, function(a,b) return a+b end)
tensor.equal(u,t)
tensor.add(u,t)
t:add(u) t:add(1) -- attention ! modifie le tenseur t!! utiliser torch.add(t,u) sinon
t:sub(u) t:csub(2) -- pareil !!
t:abs() t:sign()   -- pareil !!
t:mul(2) t:cmul(u) -- pareil !!
t:cmax(u) t:cmin(u)-- pareil !!
t:dot(u:t()) t*u:t() -- produit matriciel
t:norm(1) t:norm(1,2) -- norme 1 et norme 1 selon la 2eme dimension
-- comparaison 2 a 2 < <= > >= = et si tout est vrai
t:lt(u) t:le(u) t:gt(u) t:ge(u) t:eq(u) t:all()
```

Génération et chargement de données

```
torch.manualSeed(0) -- fixer la generation aleatoire pour debugage
torch.random() -- entier aleatoire
```

```
torch.uniform() torch.normal(1,0.5) torch.bernouilli(0.2)
torch.rand(10,5) -- uniform
torch.randn(10,5) -- gaussien
-- generation gaussienne (mu = {mu_1, mu_2,..}, sigma = {sig_1,sig_2,...})
gen_gauss = function(nbpoints, mu,sigma)
  local d = #mu
  local X = torch.randn(nbpoints,d)
  for i = 1,d do  X[{{},i}]=X[{{},i}]*sigma[i]+mu[i] end
  return X
end

-- deux gaussiennes et les labels
gen_bigauss = function(nbpoints, mu1, mu2, sigma1,sigma2)
  X=torch.cat(gen_gauss(nbpoints/2,mu1,sigma1),gen_gauss(nbpoints/2,mu2,sigma2),1)
  Y = torch.cat(torch.ones(nbpoints/2,1),-1*torch.ones(nbpoints/2,1))
  idx = torch.randperm(nbpoints):long()
  return X:index(1,idx),Y:index(1,idx)

X, Y = gen_bigauss(200,{1,1},{-1,-1},{1,1},{1,1})

-- serialiser des objets
torch.save("mydataset.dat",{x:X,y:Y})
dataset = torch.load("mydataset.dat")
dataset.x dataset.y

-- charger un fichier texte  (pas efficient du tout)
load_csv = function(fn)
  local csvfile = io.open(fn,'r')
  local header = csvfile.read()
  i = 0
  for line in csvfile:lines('*l') do
    i=i+1
    l = line:split(',')
    for k,v in ipairs(l) do
      data[i][k] = val
    end
  end
```

ML

Le package principal est `nn`, dont l'objet abstrait principal est le `Module`, sorte de fonction modulaire qui à partir d'entrées (sous forme de tenseur) calcule une sortie (tenseur également). Il reprend la terminologie des réseaux de neurones pour l'optimisation : **forward** pour le calcul de la sortie, **backward** pour l'optimisation à partir d'une erreur de sortie, ... Il contient :

— deux variables d'états : **output** et **gradInput** ;
— les fonctions [**output**] **forward(input)** et [**gradInput**] **backward(input, gradOutpout)** (prediction et backpropagation) auxquelles il ne faut pas toucher de préférence, qui sont appelées par les fonctions ci-dessous ;
— **output updateOutput(input)** et **gradInput updateGradInput(input,gradOutpout)** et **accGradParameters(input, gradOutput,scale)** qui calculent respectivement la sortie en fonction de **input**, le gradient en fonction de l'entré et du gradient à la sortie.

Un autre object d'intérêt est **Criterion** qui implémente un critère générique selon la même convention :

— variables d'état **output** et **gradIntput** résultats de l'appel des deux fonctions suivantes ;
— [**output**] **forward(input,target)** : en fonction d'une entrée et de la cible, calcule le coût associé (scalaire en général) ;
— [**gradInput**] **backward(input,target)** : calcule le gradient du coût (tenseur en général).

Le module **Logger** très utile pour logger des informations :

```
logger = optim.Logger('acc.log') logger:setNames{'Training acc','Test acc'}
logger:add({x,y}
logger:plot()
```

Enfin, le module **optim** implémente toute une série d'algorithmes d'optimisation, gradient ou non.

3 TensorFlow