Outils : THEANO

Bibliothèque logicielle en Python utilisée dans le Deep Learning.
Développée par le MILA (Institut des algorithmes d'apprentissage de Montréal – UdeM - McGill)

http://www.deeplearning.net/software/theano/

Première version : 2007
Licence logiciel : BSD
Open Source : oui
Plate-forme : Multiplatforme
Ecrit en : Python
Interface : Python (keras)
Support Open Multi-Processing : oui (interface de programmation pour le calcul parallèle sur architecture à mémoire partagée)
Support Open Computing Language : en développement (programmer des systèmes parallèles hétérogènes comprenant par exemple à la fois un CPU multi-cœur et un GPU)
Support CUDA (Compute Unified Device Architecture): oui (technologie de GPGPU (General-Purpose Computing on Graphics Processing Units), c'est-à-dire utilisant un GPU pour exécuter des calculs généraux à la place du CPU)
Différenciation automatique : oui
A des modèles pré-entraînés : A travers le zoo modèle de Lasagne (lien GitHub)
Réseaux récurrents : oui
Réseaux convolutifs : oui
Restricted Boltzmann machine (RBM)/Deep belief network (DBN) : oui (méthodes non supervisées)
Exécution parallèle (multinoeud) : oui
Développé activement : non

- Construit des graphiques symbolique (calcul des gradients et opérations mathématiques)
- Tensor (matrice, vecteur,scalaire)
- Shared Variable
- Déclaration des calculs séparée de leur compilation
- Stabilisation des expressions numériques instables
- Capable d'utiliser la GPU pour optimiser les calculs matriciels
- Applications :
    - Deep learning (grâce au GPU)
    - Permet aussi Machine Learning et analyse mathématique

- **Performances**

Theano utilise plusieurs techniques pour augmenter les performances comme la génération de code C, la pré-allocation de mémoire temporaire ou la fusion des boucles. Il est également possible d'optimiser les opérations sur les matrices.

De plus, Theano fonctionne sur le principe d'opérations mathématiques symboliques (aussi appelées « graph »). Le but est de définir une structure pour un ensemble d'opérations mathématiques qui seront ensuite compilées de manière optimale par Theano en s'appuyant sur diverses librairies. Tensorflow fonctionne également de la même manière. La compilation de ces graph peut prendre un peu de temps avant de s'exécuter.

Enfin, Theano permet d'exécuter de manière transparente un code de machine learning en tirant profit de la GPU du système. Les cartes graphiques permettent de réaliser des calculs matriciels avec une grande rapidité et ainsi optimiser le temps de calcul comparé à un CPU classique. Pour utiliser la GPU, il faut préalablement avoir installé libgpuarray et préciser device=cuda.

- **Domaines d'application**

Theano est une librairie conçue à la base pour faire du deep learning. Elle est capable d'utiliser des GPU qui sont le moyen le plus rapide d'entrainer et d'exécuter des réseaux de neurones et est donc parfaitement adaptée à ce domaine.

Theano permet également de faire du machine learning plus traditionnel et orienté analyse mathématique des données (Régression logistique, …) mais n'est pas la meilleure librairie dans ce domaine.

Finalement, il faut aussi noter que Theano est une librairie qui est aujourd'hui beaucoup moins populaire et utilisée qu'il y a 1 ou 2 ans car l'équipe en charge de l'outil a annoncé la fin du support et du développement du projet, laissant émerger d'autres frameworks comme Tensorflow ou Caffe.

- **Support mathématique**

Theano peut supporter n'importe quel type d'objet Python mais est désigné pour manipuler des expressions symboliques. Plus précisément, l'objet de base manipulé est une matrice que l'on appelle un Tensor. Ces Tensors possèdent un certain type (scalar, matrix, array …) et peuvent être manipulés avec des opérations mathématique simples (Addition, Multiplication, …) ou complexes (Dérivées, Gradient, …).

Theano s'occupera d'optimiser, de compiler et de faire les calculs que l'on aura préalablement déclarés sur les tensors. Ces opérations s'appuient sur les librairies numpy et scipy ainsi que sur BLAS pour optimiser les calculs vectoriels. La vitesse de calcul peut notamment dépasser le C grâce à l'utilisation de GPU et Theano peut reconnaitre quelques expressions numériques instables pour les exécuter d'une manière plus stable avec d'autres algorithmes.

- **Limitations**

Les boucles *While-* ou *for-Loops* à l'intérieur d'un graphique d'expression sont prises en charge, mais uniquement via l'opération theano.scan() (qui impose des restrictions sur la façon dont le corps de boucle peut interagir avec le reste du graphique).

Ni *goto* ni *recursion* ne sont supportés ou planifiés dans les graphes d'expression.

Réseaux de neurones assez gros mettent du temps à compiler.

- **Avantages :**
  - Différenciation automatique.
  - Librairie bas niveau donc très modulable
  - Outils Blocks Lasagne K peuvent être utilisés au dessus(/avec) de Theano
  - Optimisation de la compilation des modèles mathématiques

- **Inconvénients :**
  - Peut être déployé sur un seul GPU.
  - Les grands modèles peuvent exiger de longs temps de compilation
  - Librairie bas niveau donc plus difficilement utilisable
  - N'est plus développé par MILA
  - « Successeur » : TensorFlow (Yan Goodfellow)

INSTALLATION

Sont requis pour une bonne fonctionnalité :
- Python 2.7* or (>= 3.4 and < 3.6)
- NumPy >= 1.9.1 <= 1.12
- SciPy >= 0.14 < 0.17.1
- BLAS installation (with Level 3 functionality)

Pour **Python 2.7**, la commande d'installation est sudo apt-get install python2.7
Python est un langage de programmation.

Pour **Theano**, la commande d'installation est pip install Theano

```
laura@laura-VirtualBox:~/Bureau$ pip install Theano
Collecting Theano
Collecting six>=1.9.0 (from Theano)
  Using cached https://files.pythonhosted.org/packages/73/fb/00a976f728d0d1fecfe898238ce23f502a721c0ac0ecfedb80e0d88c64e9/six-1.12.0-py2.py3-none-any.whl
Collecting numpy>=1.9.1 (from Theano)
  Using cached https://files.pythonhosted.org/packages/c4/33/8ec8dcdb4ede5d453047bbdbd01916dbaccdb63e98bba60989718f5f0876/numpy-1.16.2-cp27-cp27mu-manylinux1_
Collecting scipy>=0.14 (from Theano)
  Using cached https://files.pythonhosted.org/packages/81/39/f1457091d0a45a84a2bd7815e2cf6bd45d4fe240728e9ed567cbb17c8abe/scipy-1.2.1-cp27-cp27mu-manylinux1_x
Installing collected packages: six, numpy, scipy, Theano
Successfully installed Theano-1.0.4 numpy-1.16.2 scipy-1.2.1 six-1.12.0
```

Theano collecte **Six**, **NumPy** et **SciPy**.
- Six est une bibliothèque de compatibilité python 2 et 3, ses fonctions vont aplanir les différences entre les versions Python pour écrire du code Python compatible avec les deux versions de Python.
- Numpy est une librairie destinée à manipuler de grands ensembles de nombres, matrices, tableaux multidimentionnels et fonctions mathématiques opérant sur ces tableaux.
- SciPy est un projet visant à unifier et fédérer un ensemble de bibliothèques Python à usage scientifique. Scipy utilise les tableaux et matrices du module NumPy.

Pour **BLAS**, la commande d'installation est : sudo apt install libopenblas-dev
BLAS (Basic Linear Algebra Subprograms) est un ensemble de fonction standardisées pour faire de l'algèbre linéaire (vecteurs, matrices, produits scalaires ...) et sert notamment à optimiser les calculs fait par NumP² y ou SciPy.

TEST

Dans un premier temps, un exemple (très) simple, adding 2 scalars :
http://deeplearning.net/software/theano/tutorial/adding.html#adding-two-scalars
Pour vérifier que Theano est installé et fonctionne.

```
import os
import sys
import numpy
import theano.tensor as T

from theano import function
x = T.dscalar('x')
y = T.dscalar('y')
z = x + y
f = function([x,y], z)

f(2,3)
print "somme = %d" % f(2,3)
```

Puis d'autres exemples plus complexes :

XOR : https://www.simonho.ca/machine-learning/xor-logic-gate-neural-networks/

```
import numpy as np
import theano
import theano.tensor as T
import time

# Set inputs and correct output values
inputs = [[0,0], [1,1], [0,1], [1,0]]
outputs = [0, 0, 1, 1]

# Set training parameters
alpha = 0.01 # Learning rate
training_iterations = 50000
hidden_layer_nodes = 3

# Define tensors, poids et biais
x = T.matrix("x")
y = T.vector("y")
b1 = theano.shared(value=1.0, name='b1')
b2 = theano.shared(value=1.0, name='b2')

# Set random seed
rng = np.random.RandomState(2345)

# Initialize weights
w1_array = np.asarray(rng.uniform(low=-1, high=1, size=(2, hidden_layer_nodes)),
                      dtype=theano.config.floatX) # Force type to 32bit float for GPU
w1 = theano.shared(value=w1_array, name='w1')

w2_array = np.asarray(rng.uniform(low=-1, high=1, size=(hidden_layer_nodes, 1)),
                      dtype=theano.config.floatX) # Force type to 32bit float for GPU
w2 = theano.shared(value=w2_array, name='w2')

a1 = T.nnet.sigmoid(T.dot(x, w1) + b1)  # Input -> Hidden #produit scalaire
a2 = T.nnet.sigmoid(T.dot(a1, w2) + b2) # Hidden -> Output
hypothesis = T.flatten(a2)
# Il faut laplatir pour que les hypotheses (matrice) et y (vecteur) aient la meme forme.

cost = T.sum((y - hypothesis) ** 2) # calcul de lerreur quadratic

updates_rules = [ #mise a jour des poids/biais du reseau
    (w1, w1 - alpha * T.grad(cost, wrt=w1)),
    (w2, w2 - alpha * T.grad(cost, wrt=w2)),
    (b1, b1 - alpha * T.grad(cost, wrt=b1)),
    (b2, b2 - alpha * T.grad(cost, wrt=b2))
]
```

```
# Theano compiled functions, execution
train = theano.function(inputs=[x, y], outputs=[hypothesis, cost], updates=updates_rules)
predict = theano.function(inputs=[x], outputs=[hypothesis])

# Training
cost_history = []

start = time.time()
for i in range(training_iterations):
    if (i+1) % 5000 == 0:
        print "Iteration #%s: " % str(i+1)
        print "Cost: %s" % str(cost)
    h, cost = train(inputs, outputs)
    cost_history.append(cost)
end = time.time()

test_data = [[0,0], [1,1], [0,1], [1,0]]
predictions = predict(test_data)
print np.round(predictions)

print('Time (s):', end - start)
```

Logistic Regression : http://deeplearning.net/software/theano/tutorial/examples.html#a-real-example-logistic-regression

```
import numpy
import theano
import theano.tensor as T #tensor element de base
rng = numpy.random

#trouver relations logistique entre 2 variables. faire un modele statistique.
#declare 2 constantes
N = 400                                   # training sample size
feats = 784                               # number of input variables

# generate a dataset: D = (input_values, target_class)
D = (rng.randn(N, feats), rng.randint(size=N, low=0, high=2))
                    #random matrix N et feats et prends vector de longueur N, val entre 0 et 2 dedans
training_steps = 10000          #nb detape pour lentrainement

# Declare Theano symbolic variables, donc des tensor
x = T.dmatrix("x")      #carre
y = T.dvector("y")          #ligne

# initialize the weight vector w randomly
#
# this and the following bias variable b
# are shared so they keep their values
# between training iterations (updates)
w = theano.shared(rng.randn(feats), name="w")  # shared variable partagée, utilise dans plusieurs fct
                    #variables partagées, 784 valeurs au pif entiendra W

# initialize the bias term
b = theano.shared(0., name="b")
#et B contiendra 0

print("Initial model:")
print(w.get_value())    #784 au pif
print(b.get_value())    #val 0

# Construct Theano expression graph
p_1 = 1 / (1 + T.exp(-T.dot(x, w) - b))   # Probability that target = 1  #
prediction = p_1 > 0.5                    # The prediction thresholded sup à 0.5
                      #vrai sinon faux
xent = -y * T.log(p_1) - (1-y) * T.log(1-p_1) # erreur perte loss function
                    #formule de lerreur, on utilie y comme vrai valeur de sortie
                    #tensor de sortie qui est un vecteur, on calcul lerreur
cost = xent.mean() + 0.01 * (w ** 2).sum()# The cost to minimize
                    #on calcule le cout, qu'on cherche à minimiser
gw, gb = T.grad(cost, [w, b])             # Compute the gradient of the cost
                    #gradient, faire gradient à partir du cost, gradient/w et /b
                                          # w.r.t weight vector w and
                                          # bias term b
                                          # (we shall return to this in a
                                          # following section of this tutorial)


# Compile
# Pour le training, on demande a utiliser une fonction qui prend
    #nos tensor x et y en parametre (qu'on a cree plus haut)
# En sortie, il nous faut le resultat de chaque phase de training et l'erreur associee
# Et on precise que ce que l'on veut mettre a jour a chaque iteration c'est w et b,
    #avec gw et gb qui sont les gradients qui vont dans une certaine direction
train = theano.function(
          inputs=[x,y],
          outputs=[prediction, xent],
          updates=((w, w - 0.1 * gw), (b, b - 0.1 * gb)))

#Pour la phase de prediction, on a juste besoin de passer notre tensor x
    #et le seul resultat que l'on veut est la prediction
#(On demande au modele d'evaluer une entree : "Quel chiffre est-ce ?" "C'est un 1".
predict = theano.function(inputs=[x], outputs=prediction)

# Train
#On fait 10 000 iterations pour entrainer notre modele
#D est notre dataset, on lui passe donc D[0] en tant que x (qui est bien une matrice)
    #et D[1] en tant que y (qui est bien un vecteur)
for i in range(training_steps):
    pred, err = train(D[0], D[1])

#On affiche les coeff w et b finaux :
print("Final model:")
print(w.get_value())
print(b.get_value())
print("target values for D:")
print(D[1])

#On affiche les predictions
print("prediction on D:")
print(predict(D[0]))
#true si supp à 0.5
```