Artificial Neural Networks with PyBrain

Inteligencia Artificial en los Sistemas de Control Autónomo
Máster en Ciencia y Tecnología desde el Espacio

Departamento de Automática

Universidad de Alcalá

ISG

Table of Contents

Introduction

PyBrain: The most popular Python ANN library

- Actually, PyBrain is a ML library
- Built on `Scikit`
- Scikit is the most popular Python Machine Learning library
- Multiple networks
- Multiple learning algorithms
- Blackbox optimization

(Slides from PyBrain documentation)

Introduction

Installation (I)

Method 1:

1. Install Numpy: `apt-get install python-numpy`
2. Install Scikit: `apt-get install python-scikit`
3. Install PyBrain: `apt-get install python-pybrain`
4. (Install pip: `apt-get install python-pip`)
5. Install dateutils: `pip install dateutils`

Introduction

Installation (II)

Method 2 (ROS VM):

1. Install pip: `apt-get install python-pip`
2. Install dateutils: `pip install dateutils`
3. Install Numpy: `pip install numpy`
4. Install Scikit: `apt-get install python-scipy`
5. Install PyBrain: `pip install pybrain`

PyBrain basics

Building a network (I)

Method: `buildNetwork(*layers, **options)`: Multilayer network

- `layers`: Array with the number of neurons per layer
- `` `bias': True ``: Biased or not
- `` `hiddenclass': SigmoidLayer ``: Activation function in hidden layer
- `` `outclass': LinearLayer ``: Activation function in hidden layer (recommended `SoftmaxLayer`)
- `` `recurrent': False ``: Feed forward network by default

Class: `FeedForwardNetwork`

- `activate(inpt)`: Feed network with `inpt`

PyBrain basics

Building a network (II)

One input neuron
Two hidden neurons
One output neuron

```
>>> from pybrain.tools.shortcuts import
    buildNetwork
>>> net = buildNetwork(2, 3, 1)
>>> net.activate([2, 1])
array([-0.98646726])
```

```
>>> from pybrain.structure import
    SoftmaxLayer
>>> net = buildNetwork(2, 3, 2, outclass=
    SoftmaxLayer, bias=True)
>>> net.activate((2, 3))
array([ 0.6656323,  0.3343677])
```

Two input neurons
Three hidden neurons
One output neuron
Softmax in output layer
Tanh in input layer
With bias

PyBrain basics

Building a dataset (I)

Class `ClassificationDataSet`

- Constructor: `(target=1, nb_classes=0, class_labels=None)`
- Interesting fields
  - `input`: Array of arrays with input data
  - `target`: Array of arrays with class
- Interesting methods
  - `load_matlab(cls, fname)`
  - `calculateStatistics()`
  - `_convertToOneOfMany(bounds=(0, 1))`
  - `addSample(inp, target)`
  - `splitWithProportion(proportion = 0.5)`

PyBrain basics

Building a dataset (II)

Two inputs
One output

```
>>> from pybrain.datasets import ClassificationDataSet
>>> ds = ClassificationDataSet(2, 1)
```

```
>>> ds.addSample((0, 0), (0,))
>>> ds.addSample((0, 1), (1,))
>>> ds.addSample((1, 0), (1,))
>>> ds.addSample((1, 1), (0,))
```

XOR samples

Examine dataset

```
>>> ds['input']
array([[ 0.,  0.],
       [ 0.,  1.],
       [ 1.,  0.],
       [ 1.,  1.]])
```

```
>>> ds['target']
array([[0],
       [1],
       [1],
       [0]])
```

PyBrain basics

Training a network (I)

Class: `BackpropTrainer`

- Constructor: `(network, dataset=None, learningrate=0.01, lrdecay=1.0, momentum=0., verbose=False, batchlearning=False, weightdecay=0.)`
- `train()`: Train just for one epoch
- `testOnData(dataset=None, verbose=False)`: Compute MSE
- `testOnClassData(self, dataset=None, verbose=False, return_targets=False`
- `trainUntilConvergence(self, dataset=None, maxEpochs=None, verbose=None, continueEpochs=10, validationProportion=0.25, trainingData=None, validationData=None, convergence_threshold=10)`

PyBrain basics

Training a network (II)

```
>>> from pybrain.supervised.trainers import BackpropTrainer
>>> net = buildNetwork(2, 3, 1, bias=True)
>>> trainer = BackpropTrainer(net, ds)
>>> trainer.train()
0.31516384514375834
>>> trainer.trainUntilConvergence()
```

Advanced features

Network storage and recovery

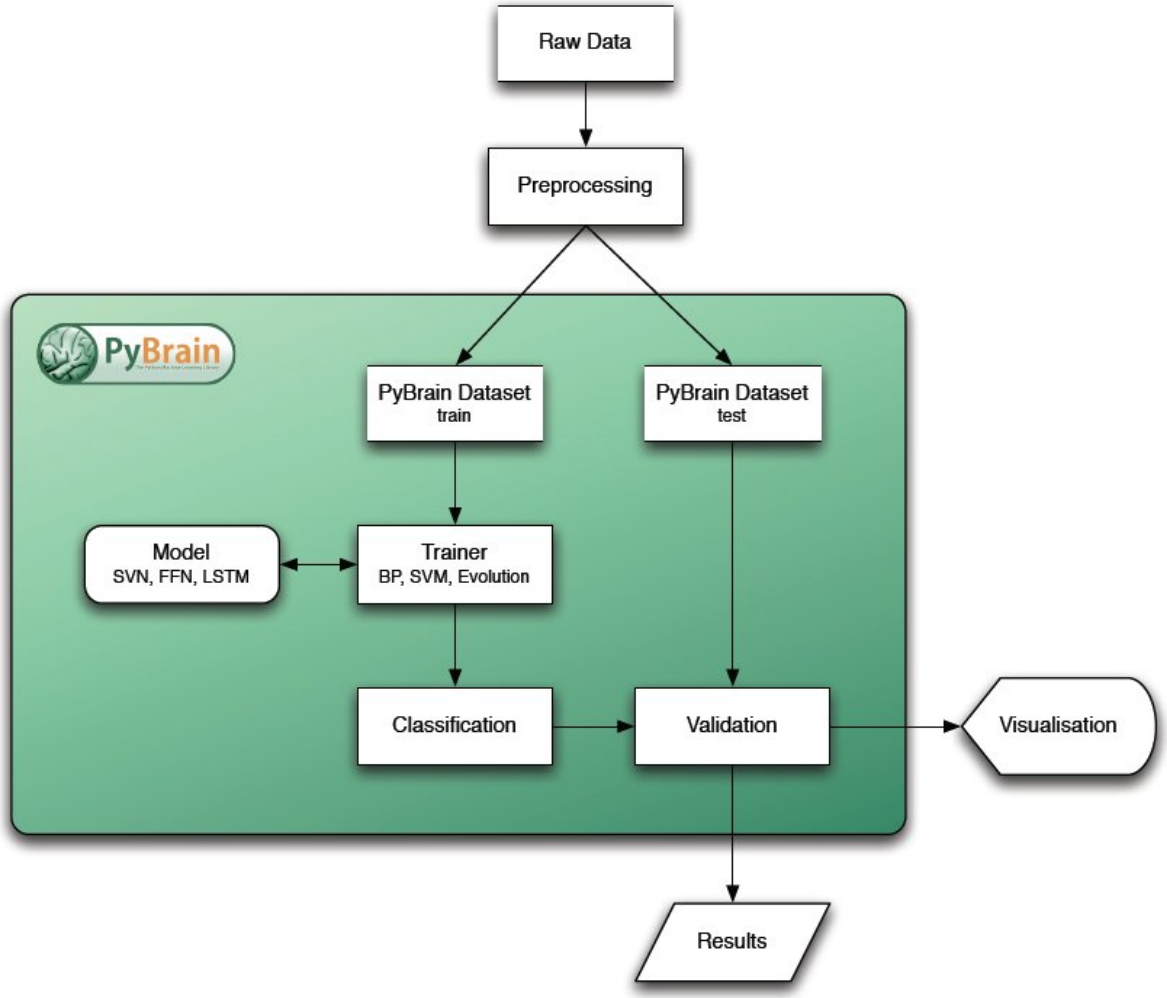
We usually need to store the trained network to embed it in the robot

```
from pybrain.tools.shortcuts import buildNetwork
from pybrain.tools.xml.networkwriter import NetworkWriter
from pybrain.tools.xml.networkreader import NetworkReader

net = buildNetwork(2,4,1)

NetworkWriter.writeToFile(net, 'filename.xml')
net = NetworkReader.readFrom('filename.xml')
```

(Source)

Model validation

Machine Learning workflow



Model validation

Training and validation data sets

```
alldata = ...

tstdata, trndata = alldata.splitWithProportion(0.25)

trndata._convertToOneOfMany()
tstdata._convertToOneOfMany()

fnn = buildNetwork(trndata.indim, 5, trndata.outdim, outclass=
    SoftmaxLayer)
trainer = BackpropTrainer(fnn, dataset=trndata, momentum=0.1,
    verbose=True, weightdecay=0.01)
trnresult = percentError(trainer.testOnClassData(), trndata['
    class'])
tstresult = percentError(trainer.testOnClassData(dataset=tstdata
    ), tstdata['class'])
```

Study case

Face recognition (I)

The dataset:

- 40 people
- 10 images for each person
- 64x64 pixels
- 40 x 10 = 400 samples

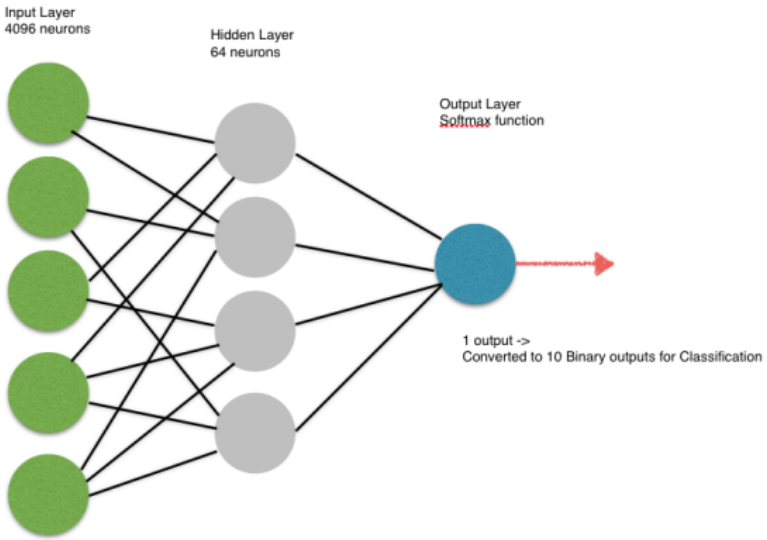
(Source)

Study case

Face recognition (II)

The solution:

- Multilayer perceptron
- Input layer: 64 x 64 = 4096 neurons
- Hidden layer: 64 neurons, sigmoid
- Output layer: 10 neurons, softmax
- 40 x 10 = 400 samples



Study case

Face recognition (III)

```
from sklearn import datasets
olivetti = datasets.fetch_olivetti_faces()
X, y = olivetti.data, olivetti.target

from pybrain.datasets import ClassificationDataSet
from pybrain.utilities import percentError
from pybrain.tools.shortcuts import buildNetwork
from pybrain.supervised.trainers import BackpropTrainer
from pybrain.structure.modules import SoftmaxLayer

ds = ClassificationDataSet(4096, 1 , nb_classes=40)
for k in xrange(len(X)):
    ds.addSample(X[k], y[k])

tstdata, trndata = ds.splitWithProportion(0.25)
trndata._convertToOneOfMany()
tstdata._convertToOneOfMany()
fnn = buildNetwork( trndata.indim, 64, trndata.outdim, outclass=
    SoftmaxLayer)
trainer = BackpropTrainer(fnn, dataset=trndata, momentum=0.1,
    learningrate=0.01, verbose=True, weightdecay=0.01)

trainer.trainEpochs(50)
print 'Percent Error on Test dataset: ', percentError(trainer.
    testOnClassData(dataset=tstdata), tstdata['class'])
```

Exercises

Introductory exercises

Based on the template code given in the next slide, do the following tasks:

1. Substitute X by the proper values to train and validate a single neuron network with AND
   - Which is the effect of parameters `learningrate` and `epochs`?
2. Train and validate a two neurons network with XOR
   - Visualize weights (search the solution in Internet)
3. Train and validate a single neuron network with XOR
   - What is happening?
4. Train and validate a 10 neuron network with XOR
   - Use a `SoftmaxLayer` output layer. Analyze the result
5. Train and validate a 20 neuron network with XOR
   - Increasing the network size improves the result?

Exercises

Introductory exercises (II)

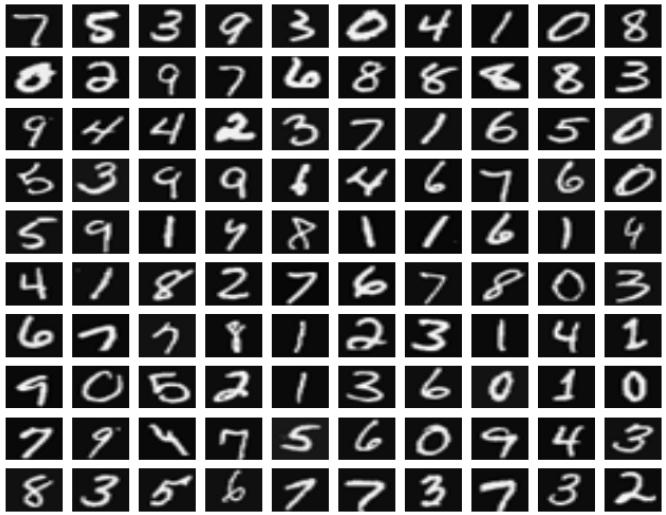
```
#!/usr/bin/python

from pybrain.datasets import SupervisedDataSet
from pybrain.tools.shortcuts import buildNetwork
from pybrain.supervised.trainers import BackpropTrainer

dataModel = [
    [(0,0), (X,)],
    [(0,1), (X,)],
    [(1,0), (X,)],
    [(1,1), (X,)],
]

ds = SupervisedDataSet(X, X)
for input, target in dataModel:
    ds.addSample(input, target)

net = buildNetwork(X, X, X, bias=True)

trainer = BackpropTrainer(net, ds, learningrate = X, momentum =
    X)
trainer.trainEpochs(epochs=X)

print '0,0->', net.activate([0,0])
print '0,1->', net.activate([0,1])
print '1,0->', net.activate([1,0])
print '1,1->', net.activate([1,1])
```

Exercises

Handwritting character recognition (I)

Handwritting recognition

1. Download the dataset from `https://atc1.aut.uah.es/~david/ex3data1.mat`
2. Load dataset using the code shown in the next slide
3. Complete the code to train and validate the neural network



(Source)

Exercises

Handwritting character recognition (II)

```
#!/usr/bin/python

import numpy as np
import scipy.io
import math

print "Loading MATLAB data..."
data = scipy.io.loadmat("ex3data1.mat")
X = data["X"]
y = data["y"]
y[y == 10] = 0 # '0' is encoded as '10' in data, fix it
```

Exercises

ANN integration with ROS (I)

1. Install Hector quadrotor:
   - `sudo apt-get install ros-indigo-hector-*`
   - Close the terminal and open a new one
2. Run simulation
   - `roslaunch hector_quadrotor_demo outdoor_flight_gazebo.launch`
3. Run teleoperation (with joystick)
   - `roslaunch hector_quadrotor_teleop xbox_controller.launch`

Exercises

ANN integration with ROS (II)

Implement a **simple** altitude control in a UAV to keep it at $1{,}5 \pm 0{,}5$m over the ground

1. Locate the topic where sonar and an altimeter publish their data
2. Identify which message types use the sonar and altimeter
3. Locate the topic that controls the UAV motion
4. Identify which message type is used to control the UAV motion
5. Implement a node for altitude control

Exercises

ANN integration with ROS (III)

Implement a **neural** altitude control in a UAV to keep it at $1{,}5 \pm 0{,}5$m over the ground

1. Implement a node that capture data from the sonar and altimeter
2. Build a dataset with the captured data, formatting and labeling it
   - Hint: Use stdout redirection
3. Train an ANN with the layout that you prefere
4. Validate the ANN
   - Hint: Write a script that loads a network, reads the input form the keyboard and gives the ANN output
5. Integrate it in a ROS node
6. Test the result