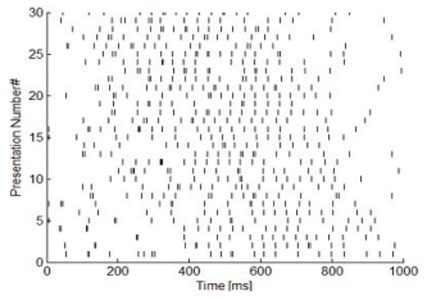
Modular toolkit for Data Processing (MDP)

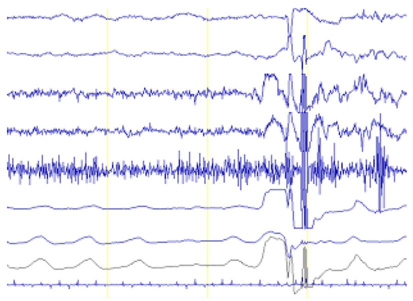
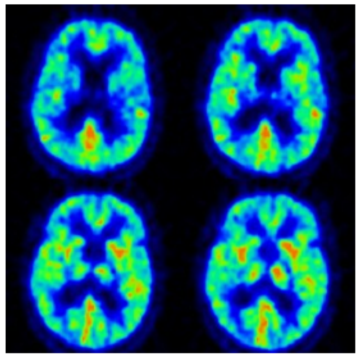
Tiziano Zito, Pietro Berkes, Niko Wilbert

Background

Open Source library (LGPL)
first release 2004
10k+ downloads, available in Debian and Python(x,y)

originated in computational neuroscience
(research group of Laurenz Wiskott)



. . . but used in other areas as well

Talk Overview

1. Introducing the basic building blocks of MDP
2. Extending MDP, Parallelization, Example
3. Outlook

Building blocks: Node

Node: fundamental data processing element, interface methods:

train (optional)
support for multiple phases, batch, online, chunks, supervised, unsupervised

execute
map n dimensional input to m dimensional output

inverse (optional)
inverse of execute method

data format: 2d numpy arrays
(1st index for samples, 2nd index for channels)
Nodes do automatic checks and conversions (dimensions, dtype).

Building blocks: Node

Example: Principal Component Analysis (PCA)
reduce dimension of data from 10 to 5:

```
>>> import mdp
>>> import numpy as np
>>> data = np.random.random((50,10))  # 50 data points
>>> node = mdp.nodes.PCANode(output_dim=5,
...                          dtype='float32')
>>> node.train(data)
>>> proj_data = node.execute(data)
```

shortcut:

```
>>> import mdp
>>> import numpy as np
>>> data = np.random.random((50,10))  # 50 data points
>>> proj_data = mdp.pca(data, output_dim=5, dtype='float32')
```

Building blocks: Node

Some available nodes:

PCA (standard, NIPALS)
ICA (FastICA, CuBICA, JADE, TDSEP)
Locally Linear Embedding
Hessian Locally Linear Embedding
Fisher Discriminant Analysis
Slow Feature Analysis
Independent Slow Feature Analysis
Restricted Boltzmann Machine
Growing Neural Gas
Factor Analysis
Gaussian Classifiers
Polynomial Expansion
Time Frames
Hit Parades
Noise
...

Or write your own node (and contribute it :-).

Building blocks: Flow

Combine nodes in a **Flow**:

```
>>> flow = PCANode() + SFANode() + FastICANode()
>>> flow.train(train_data)
>>> test_result = flow.execute(test_data)
>>> rec_test_data = flow.invert(test_result)
>>> flow += HitParadeNode()
```

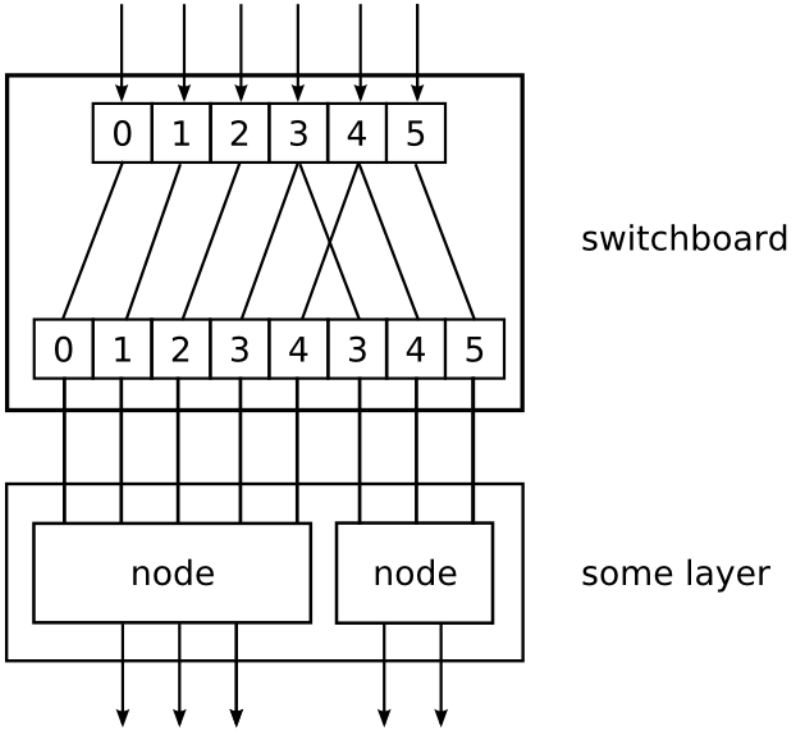
- automatic organization: training, execution, inversion
- automatic checks: dimensions and data formats
- use arrays or iterators
- crash recovery, checkpoints

Building blocks: Network

`mdp.hinet` package for hierarchical networks

Layer (combine nodes horizontally in parallel)
Switchboard (routing between layers)
FlowNode (combine nodes into a “supernode”)

All these classes are nodes, combine them as you want.

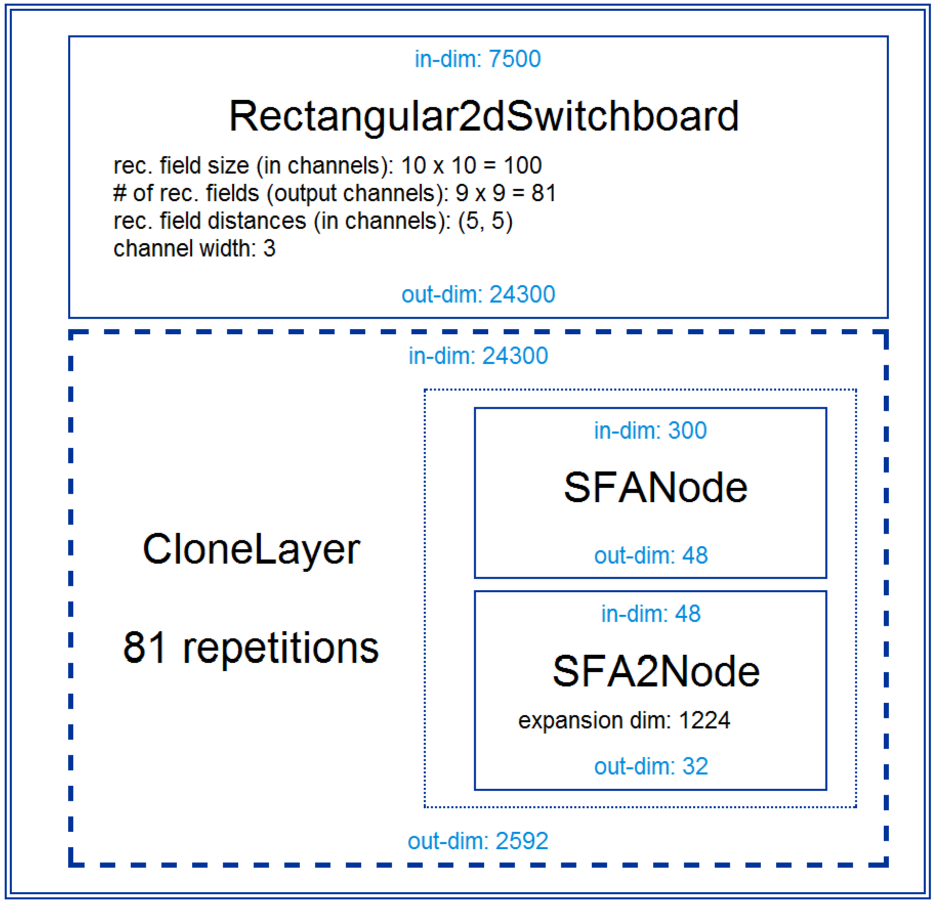


Building blocks: Network

HTML representation of your network:

```
>>> mdp.hinet.show_flow(flow)
```

Use this in your reports or GUI.



Extending MDP: Writing Nodes

Write your own node class:

```
>>> class MyNode(Node):
...     def _train(self, x):
...         ... training code ...
...     def _execute(self, x):
...         ... execution code ...
...
>>> flow = PCANode() + MyNode()
```

- integrate with the existing library
- benefit from automatic checks and conversions
- contribute your node to make it available to a broader audience

Parallelization

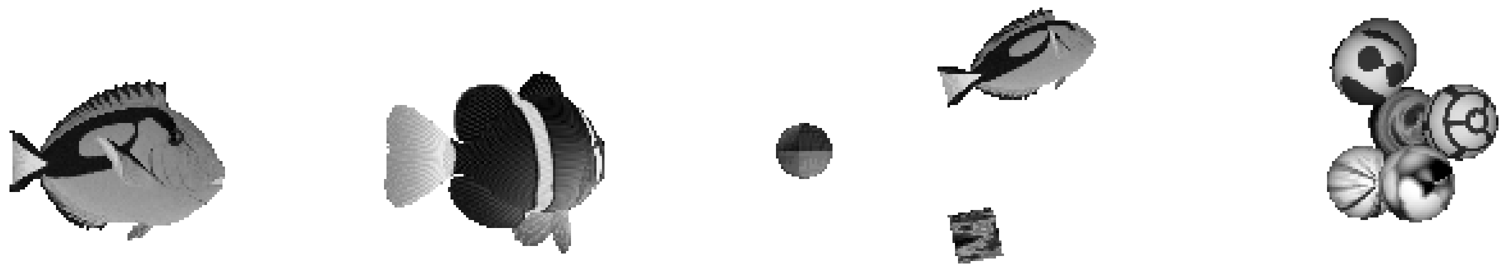
- for "embarrassingly parallel" problems
- use multiple cores or multiple machines
  (experimental support for parallel python library)
- uses abstract scheduler API (easy to write adaptor)
- easy to implement for your own nodes
  (implement `_fork` and `_join` methods)

Example:

```
>>> flow = PCANode() + SFANode()
>>> scheduler = mdp.parallel.ProcessScheduler(n_processes=4)
>>> pflow = mdp.parallel.make_flow_parallel(flow)
>>> pflow.train(data, scheduler)
```

Real World Example

- object recognition system,
  working on 155x155 pixel image sequences
- several GB of training data for each training phase.
- hierarchical network with nested nodes,
  900 “supernodes” on lowest layer
- training is distributed over network, takes multiple hours



[Franzius, M., Wilbert, N., and Wiskott, L., 2008]

Upcoming: BiNet package

`mdp.binet` package will allow data flow in both directions, enabling for example error backpropagation and loops.

compatible with both the `mpd.parallel` and `mdp.hinet` packages.

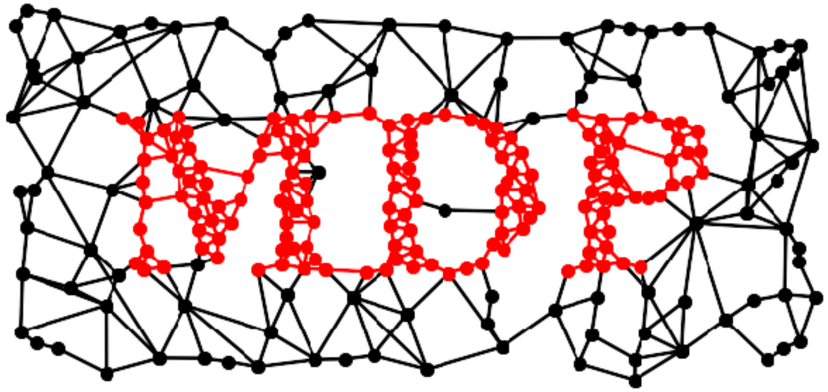
HTML+JS based inspector for debugging and analysing

scheduled for inclusion in MDP 3.0 (maybe end of 2009)

Embedding / Using MDP

- comprehensive documentation:
  tutorial covering basic and advanced usage,
  detailed doc-strings,
  PEP8 compliant, commented, and pylint-clean code
- API is stable and designed for straightforward embedding
- unittest coverage (390+ unit tests)
- minimal dependencies: Python + NumPy
- used by:
  PyMCA (X-ray fluorescence mapping),
  PyMVPA (ML framework for neuroimaging data analysis),
  Chandler (personal organizer application)

Thank you!



mdp-toolkit.sourceforge.net