# Contents

1 Quickstart guide to PADDLE
   1.1 Installation ................................................. 3
   1.2 A Quick Example ....................................... 3
   1.3 Where to go from here ................................. 6

2 Command-line Applications
   2.1 oneimage.py ............................................. 7

3 PADDLE algorithm (paddle.dual)
   3.1 Main Functions .......................................... 9
   3.2 Inner Optimizations .................................. 11

4 PADDLE-TF algorithm (paddle.tight)
   4.1 Main Functions .......................................... 15
   4.2 Inner Optimizations .................................. 17

5 Common and low-level functions (paddle.common)
   5.1 Evaluation of dictionaries ............................. 19
   5.2 Misc ...................................................... 20

6 Data-handling functions (paddle.data) ............................................. 23

7 Examples
   7.1 Experiments from [Basso10] ........................... 25

Bibliography ................................................................. 27

Index ........................................................................... 29
PADDLE is a Python package for learning dictionaries with frame-like properties, as well as achieving sparse coding of the training data. In particular, it provides algorithms for

- learning a dictionary together with a dual of itself, and
- learning a dictionary close to a tight-frame.

PADDLE’s license is free for both commercial and non-commercial use, under the BSD terms.
1.1 Installation

First of all, you need Python. If you do not have it installed yet, shame on you!

The easiest way to install PADDLE is using either the command easy_install, provided by setupools:

$ easy_install -U paddle

or using pip:

$ pip install -U paddle

Alternatively, you can download the latest source package, uncompress it and install directly from the sources (you may have to run the last command as administrator):

$ tar xvzf PADDLE-latest.tar.gz
$ cd paddle-X.Y.Z
$ python setup.py install

To check the installation, run:

$ python -c "import paddle"

If no message is issued, PADDLE and its dependencies have been installed correctly.

1.2 A Quick Example

As an introduction to the use of PADDLE, we briefly describe an example learning a dictionary from a single image.

Here we focus on the essentials. The complete script, oneimage.py, can be found in the examples directory of the source distribution.

First, we load the image and possibly convert it to gray-scale:

```python
# load the image
img = sp.misc.imread(sys.argv[1])

# possibly convert to grayscale
if img.ndim == 3:
    assert img.shape[2] <= 3, img.shape
    img = sp.mean(img, 2)
```
For testing, you can use the example image `lena_std.png`, in the `examples` directory of the source distribution:

Then, we extract $N_{\text{max}}$ image patches of size 12 by 12:

```python
# extract the patches
h, w = opts.patchsize, opts.patchsize
X = paddle.common.img2patches(img, size=(h, w), Nmax=opts.nmax)
assert X.shape[0] == (h*w), X.shape

# standardize the patches
X -= sp.mean(X, 0).reshape((1, -1))
```

Note that in the last line we get rid of the offset. Here is a subset of sampled patches (as you see, many of them are mostly homogeneous):
Finally, we run PADDLE on the patches:

```python
# call paddle
K = opts.dictsize
D0, C0, U0 = paddle.dual.init(X, K)
D, C, U, full_out = paddle.dual.learn(X, D0, C0, U0, tau=opts.tau, rtol=1.e-8)
```

Here are the results of a run on 10k patches. First the atoms of the dictionary:

And then the filters (rows) of the dual:

That’s it.
1.3 Where to go from here

Now, you may want to:

• try out some of the ready-to-use command line applications distributed with this package, see section *Command-line Applications*;

• repeat some of the experiments presented in [Basso10], see section *Experiments from [Basso10]*;

• develop some Python code using this package, see ...
2.1 oneimage.py

This is the script already shown in the brief introductory tutorial, the file oneimage.py can be found in the examples directory of the source distribution.

Run the script with `-h` to get a list of possible options:

```
$ python paddle/examples/oneimage.py -h
Usage: oneimage.py imagefile [options]
```

Options:

- `-h, --help`  show this help message and exit
- `n INT`  limit the number of training patches to INT
- `p INT`  set the size of the patches to INTxINT (default 12)
- `k INT`  set the size of the dictionary to INT (default 200)
- `--tau=FLOAT`  set the weight of sparsity penalty to FLOAT (default 0.5)
- `--npzfile=PATH`  save the results to PATH in npz format
- `--outpath=PATH`  save to PATH three images with a sample of the training patches, the dictionary atoms and their dual filters, all arranged in tables
- `--nrows=INT`  set the number of rows of the tables to INT (default 10)
- `--ncols=INT`  set the number of columns of the tables to INT (default 20)
This module implements the PADDLE algorithm for learning a dictionary and its dual.

There are two main functions:

- `paddle.dual.init`, for initializing the unknowns, and
- `paddle.dual.learn`, for running the actual PADDLE algorithm.

The algorithm uses other routines, in particular two driving the three inner optimizations:

- `paddle.dual._ist`, optimization of $U$ by iterative soft-thresholding
- `paddle.dual._pgd`, optimization of $D$ and $C$ by projected gradient descent

Both uses FISTA acceleration. Also non-accelerated version, using fixed stepsizes, are available, though not used by `paddle.dual.learn`:

- `paddle.dual._ist_fixed`
- `paddle.dual._pgd_fixed`

### 3.1 Main Functions

**paddle.dual.init**(X, K, det=False, rnd=False)

Initializes the variables.

Initializes the matrices $D$, $C$ and $U$ from the data matrix $X$. The dimension of the dictionary is $K$. If `det` is True the first $K$ columns of $X$ are chosen as atoms (deterministic initialization), otherwise they are picked randomly. The atoms are normalized to one. The matrix $U$ is chosen as to minimize the reconstruction error. The matrix $C$ is chosen as the pseudo-inverse of $D$, with rows normalized to one.

**Parameters**

- **X**: (d, N) ndarray
  - Input data
- **K**: integer
  - Size of the dictionary
- **det**: bool
  - If True, the choice of atoms is deterministic
- **rnd**: bool
  - If True, the atoms are not sampled from the examples, but have random values
Returns

\[ D_0 \] : (d, K) ndarray
The initial dictionary

\[ C_0 \] : (K, d) ndarray
The initial dual

\[ U_0 \] : (K, N) ndarray
The initial encodings

\[ \text{paddle.dual.} \text{learn} \ (X, D_0, C_0, U_0, callable=None, **kwargs) \]
Runs the PADDLE algorithm.

The function takes as input the data matrix \( X \), the initial values for the three unknowns, \( D_0, C_0 \) and \( U_0 \), and a dictionary of parameters.

The optional parameters are passed as keyword arguments.

Parameters

\[ X \] : (d, N) ndarray
Input data

\[ D_0 \] : (d, K) ndarray
The initial dictionary

\[ C_0 \] : (K, d) ndarray
The initial dual

\[ U_0 \] : (K, N) ndarray
The initial encodings

\[ \text{callable} \] : function of type foo(D, C, U, X)
If not None, it gets called at each iteration and the result is appended in an item of full_output

\[ \tau \] : float, optional
Weight of the sparsity penalty (default 0.1)

\[ \eta \] : float, optional
Weight of the coding error (default 1.0)

\[ \mu \] : float, optional
Weight of the l2 penalty on the coefficients (default 0.0)

\[ \text{nnU} \] : bool, optional
Adds a non-negativity constraint on \( U \) (default False)

\[ \text{nnD} \] : bool, optional
Adds a non-negativity constraint on \( U \) (default False)

\[ \text{maxiter} \] : int, optional
Maximum number of outer iterations (default 10)

\[ \text{minused} \] : integer, optional
Minimum number of times an atom as to be used (default 1)

\[ \text{verbose} \] : bool, optional
Enables verbose output (default False)

\[ \text{rtol} \] : float, optional
Relative tolerance checking convergence (default 1.e-4)

**save_dict**: bool, optional
- If true, the dictionary is saved after each outer iteration (default False)

**save_path**: str, optional
- The path in which to save the dictionary (relevant only if save_dict is True, default "/")

**save_sorted**: bool, optional
- If true and if save_dict is also True, the atoms of dictionary are sorted wrt the usage in the sparse coding before being displayed and saved (default False)

**save_shape**: integer pair, optional
- Numbers of (rows, cols) used to display the atoms of the dictionary (default (10, 10))

**Returns**
- \(D\): (d, K) ndarray
  - The final dictionary
- \(C\): (K, d) ndarray
  - The final dual
- \(U\): (K, N) ndarray
  - The final encodings
- **full_out**: dict
  - Full output

### 3.2 Inner Optimizations

**paddle.dual._ist** \((X, U0, D, C, pars, maxiter=1000)\)
- Iterative soft-thresholding with FISTA acceleration.

Minimization of \(\frac{1}{2}\|X - DU\|_F^2 + \eta K\|U - CX\|_F^2 + \frac{2\tau}{K}\|U\|_1\) wrt \(U\). When \(\eta = 0\) the functional reduces to the well-known LASSO.

The function is used by **paddle.dual.learn** for the optimization wrt \(U\).

**Parameters**
- **X**: (d, N) ndarray
  - Data matrix
- **U0**: (K, N) ndarray
  - Initial value of the unknown
- **D**: (d, K) ndarray
  - Dictionary
- **C**: (K, d) ndarray
  - Dual of the dictionary
- **pars**: dict
  - Optional parameters
- **maxiter**: int
  - Maximum number of iterations allowed (default 500)
Returns

U : (K, N) ndarray
Optimal value of the unknown

full_out : dict
Full output

```python
paddle.dual._pgd(Y0, ABt, BBt, cost, maxiter=500, axis=0, bound=1, verbose=False, rtol=0.0001, nn=False)
```
Projected gradient descent with FISTA acceleration.

Minimization of \( \|A - YB\|_F^2 \) wrt \( Y \), under additional constraints on the norms of the columns (or the rows) of \( Y \). The minimization is performed by alternatively descending along the gradient direction \( AB^T - YBB^T \) and projecting the columns (rows) of \( Y \) on the ball with given radius.

The function is used by `paddle.dual.learn` for the optimization wrt \( D \) and \( C \). In the former case, \( A \) and \( B \) are \( X \) and \( U \), respectively, while in the latter the roles are swapped.

Parameters

- \( Y0 : (a1, a2) \) ndarray
  Initial value of the unknown
- \( ABt : (a1, a2) \) ndarray
  Part of the gradient
- \( BBt : (a2, a2) \) ndarray
  Part of the gradient
- \( cost : \) function of type `foo(Y)`
  Evaluates the cost function
- \( maxiter : \) int
  Maximum number of iterations allowed (default 500)
- \( axis : \) int
  Dimension of \( Y \) along which the constraint is active (0 for cols, 1 for rows, default is 0)
- \( bound : \) float
  Value of the constraint on the norms of the columns/rows of \( Y \) (default is 1)
- \( verbose : \) bool
  If True displays the value of the cost function at each iteration (default is False)
- \( rtol : \) float
  Relative tolerance for convergence criterion

Returns

- \( Y : () \) ndarray
  Optimal value of the unknown
- \( j : \) int
  Number of iterations performed

```python
paddle.dual._ist_fixed(X, U0, D, C, pars, maxiter=1000)
```
Iterative soft-thresholding with fixed step-size.

Minimization of \( \frac{1}{n}\|X - DU\|_F^2 + \frac{2}{n}\|U - CX\|_F^2 + \frac{2}{n}\|U\|_1 \) wrt \( U \). When \( \eta = 0 \) the functional reduces to the well-known LASSO.
This function is currently not used. The main function `paddle.dual.learn` uses its FISTA-accelerated counterpart `paddle.dual._pgd`.

**Parameters**

- `X` : (d, N) ndarray  
  Data matrix
- `U0` : (K, N) ndarray  
  Initial value of the unknown
- `D` : (d, K) ndarray  
  Dictionary
- `C` : (K, d) ndarray  
  Dual of the dictionary
- `pars` : dict  
  Optional parameters
  - `maxiter` : int  
    Maximum number of iterations allowed (default 500)

**Returns**

- `U` : (K, N) ndarray  
  Optimal value of the unknown
- `full_out` : dict  
  Full output

`paddle.dual._pgd_fixed(A0, X, U, B, G2, cost, maxiter=500, pars=None, sigma=None, axis=0, bound=1)`

Projected gradient descent with fixed stepsize.
This module implements the PADDLE algorithm for learning a dictionary approximating a tight frame.

As in `paddle.dual`, there are two main functions:

- `paddle.tight.init` for initializing the unknowns, and
- `paddle.tight.learn` for running the actual PADDLE-TF algorithm.

The algorithm uses other routines, in particular two driving the three inner optimizations:

- `paddle.tight._ist`, optimization of $U$ by iterative soft-thresholding
- `paddle.tight._pgd`, optimization of $D$ and $C$ by projected gradient descent

Both uses FISTA acceleration.

## 4.1 Main Functions

```python
paddle.tight.init(X, K, det=False)
```

Initializes the variables.

Initializes the matrices $D$ and $U$ from the data matrix $X$. The dimension of the dictionary is $K$. If `det` is True the first $K$ columns of $X$ are chosen as atoms (deterministic initialization), otherwise they are picked randomly. The atoms are normalized to one. The matrix $U$ is chosen as to minimize the reconstruction error.

**Parameters**

- $X$: (d, N) ndarray
  - Input data
- $K$: integer
  - Size of the dictionary
- $det$: bool
  - If True, the choice of atoms is deterministic

**Returns**

- $D0$: (d, K) ndarray
  - The initial dictionary
- $U0$: (K, N) ndarray
  - The initial dictionary
The initial encodings

```
paddle.tight.learn(X, D0, U0, callable=None, **kwargs)
```
Runs the PADDLE-TF algorithm.

The function takes as input the data matrix $X$, and the initial values for the unknowns $D_0$ and $U_0$.

A function that will be called at each iteration might also be passed as optional argument.

All other optional parameters are passed as keyword arguments.

**Parameters**
- $X$: (d, N) ndarray
  - Input data
- $D_0$: (d, K) ndarray
  - The initial dictionary
- $U_0$: (K, N) ndarray
  - The initial encodings
- `callable`: function of type `foo(D, U, X)`
  - If not None, it gets called at each iteration and the result is appended in an item of `full_output`
- $\tau$: float, optional
  - Weight of the sparsity penalty (default 0.1)
- $\eta$: float, optional
  - Weight of the frame potential (default 1.0)
- $\mu$: float, optional
  - Weight of the l2 penalty on the coefficients (default 0.0)
- $\text{maxiter}$: int, optional
  - Maximum number of outer iterations (default 10)
- $\text{minused}$: integer, optional
  - Minimum number of times an atom as to be used (default 1)
- $\text{verbose}$: bool, optional
  - Enables verbose output (default False)
- $\text{rtol}$: float, optional
  - Relative tolerance checking convergence (default 1.e-4)
- `save_dict`: bool, optional
  - If true, the dictionary is saved after each outer iteration (default False)
- `save_path`: str, optional
  - The path in which to save the dictionary (relevant only if `save_dict` is True, default "./"")
- `save_sorted`: bool, optional
  - If true and if `save_dict` is also True, the atoms of dictionary are sorted wrt the usage in the sparse coding before being displayed and saved (default False)
- `$\text{save_shape}$: integer pair, optional`
  - Numbers of (rows,cols) used to display the atoms of the dictionary (default (10,10))
Returns

\[ D : (d, K) \text{ ndarray} \]

The final dictionary

\[ U : (K, N) \text{ ndarray} \]

The final encodings

\text{full_out} : \text{dict}

Full output

4.2 Inner Optimizations

\text{paddle.tight.\_ist}(X, U0, D, pars, maxiter=1000)

Iterative soft-thresholding with FISTA acceleration.

Minimization of \[ \frac{1}{2} \| X - DU \|_F^2 + \frac{\tau}{K} \| U \|_1 \] wrt \( U \), that is the well-known LASSO.

Although nearly equivalent to calling \text{paddle.dual.\_ist} with \( \eta = 0 \), the cost function is different because of the (constant) term from the frame potential.

The function is used by \text{paddle.tight.learn} for the optimization wrt \( U \).

Parameters

\[ X : (d, N) \text{ ndarray} \]

Data matrix

\[ U0 : (K, N) \text{ ndarray} \]

Initial value of the unknown

\[ D : (d, K) \text{ ndarray} \]

Dictionary

\text{pars} : \text{dict}

Optional parameters

\text{maxiter} : \text{int}

Maximum number of iterations allowed (default 500)

Returns

\[ U : (K, N) \text{ ndarray} \]

Optimal value of the unknown

\text{full_out} : \text{dict}

Full output

\text{paddle.tight.\_pgd}(D0, X, U, XUt, UUt, pars, maxiter=500, bound=1)

Projected gradient descent with FISTA acceleration.

Minimization of \[ \| X - DU \|_F^2 + \eta \| DD^T - \frac{2K}{\tau} I \|_F^2 \] wrt \( D \), under additional constraints on the norms of the columns of \( D \). The minimization is performed by alternatively descending along the gradient direction and projecting the columns (rows) of \( D \) on the ball with given radius.

The function is used by \text{paddle.tight.learn} for the optimization wrt \( D \).

Parameters

\[ D0 : (d, K) \text{ ndarray} \]

Initial value of the unknown

\[ X : (d, N) \text{ ndarray} \]
Input data (fixed)

\( U : (K, N) \) ndarray

Current encodings (fixed)

\( XU_t : (d, K) \) ndarray

Part of the gradient

\( UU_t : (K, K) \) ndarray

Part of the gradient

**maxiter** : int

Maximum number of iterations allowed (default 500)

**bound** : float

Value of the constraint on the norms of the columns/rows of \( Y \) (default is 1)

**verbose** : bool

If True displays the value of the cost function at each iteration (default is False)

**rtol** : float

Relative tolerance for convergence criterion

**Returns**

\( D : (d, K) \) ndarray

Optimal value of the dictionary

\( j : \) int

Number of iterations performed
This module implements some common and low-level functions used both by paddle.dual and paddle.tight.

5.1 Evaluation of dictionaries

\texttt{paddle.common.print\_frame\_assessment}(D)

Prints an evaluation of D as a frame.

This function computes and prints the following information for D:

• the frame bounds (computed from the eigenvalues)
• the equivalent tight frame constant $\alpha$
• if there is a violation of the fundamental inequality
• the value of the frame potential and its theoretical lower bound
• the relative error between the frame operator $DD^T$ and $\alpha I$
• the mutual coherence

\textbf{Parameters}

\begin{itemize}
  \item D : (d, K) ndarray
    
    Frame matrix
\end{itemize}

\texttt{paddle.common.coherence}(D)

Compute the mutual coherence of D.

\textbf{Parameters}

\begin{itemize}
  \item D : (d, K) ndarray
    
    A dictionary.
\end{itemize}

\textbf{Returns}

\begin{itemize}
  \item c : float
    
    The mutual coherence.
\end{itemize}

\texttt{paddle.common.cumcoherence}(D, C=None)

Compute the cumulative mutual/cross coherence of D/(D,C).
Parameters

- \( D \): (d, K) ndarray
  - A dictionary.
- \( C \): (K, d) ndarray
  - A dual of the dictionary (optional).

Returns

- \( c \): (K-1,) ndarray
  - The cumulative coherence.

### 5.2 Misc

**paddle.common._cost_rec\((D, X, U, pars=None)\)**

Reconstruction error term.

Computes the normalized reconstruction error \( \|X - DU\|_F^2 / (d \times N) \).

**Parameters**

- \( D \): (d, K) ndarray
  - Dictionary.
- \( X \): (d, N) ndarray
  - Input data.
- \( U \): (K, N) ndarray
  - Encodings.
- \( pars \): not used

**Returns**

- \( rec\_err \): float
  - Reconstruction error.

**paddle.common._cost_cod\((C, X, U, pars)\)**

Coding error term.

Computes the normalized and weighted coding error \( \eta \|U - CX\|_F^2 / (K \times N) \).

**Parameters**

- \( C \): (K, d) ndarray
  - Dual Dictionary.
- \( X \): (d, N) ndarray
  - Input data.
- \( U \): (K, N) ndarray
  - Encodings.
- \( pars \): dict with at least key eta
  - \( pars[\"eta\"] \) is the \( \eta \) parameter weighting the coding error

**Returns**

- \( cod\_err \): float
  - Coding error.
paddle.common._replaceAtoms(X, U, D, replace)
Replaces some atoms.

Replaces the atoms of D listed in replace with the worst reconstructed examples of X. Only U is changed (and returned).

**Parameters**
- **X**: (d, N) ndarray
  - Input data
- **U**: (K, N) ndarray
  - Current encodings of the input data X
- **D**: (d, K) ndarray
  - Current dictionary
- **replace**: list of integer
  - List of indexes of the atoms to be replaced

**Returns**
- **U**: (K, N) ndarray
  - The updated encodings.

paddle.common._saveDict(D, U, Nrows=8, Ncols=25, path='./savedDict.png', sorted=False)
Saves a figure of a dictionary atoms.

Creates a table with Nrows x Ncols images of the atoms of D, drawn as square image patches. It assumes that the dimension d of the atoms is a perfect square.

**Parameters**
- **D**: (d, K) ndarray
  - Dictionary
- **U**: (K, N) ndarray
  - Encodings
- **Nrows**: integer
  - Number of rows in the figure
- **Ncols**: integer
  - Number of columns in the figure
- **path**: string
  - Name of the file where the figure will be saved
- **sorted**: bool
  - If True, the atoms will be sorted according to their usage in U

paddle.common.img2patches(img, size, Nmax=0)
Extract patches from an image.

**Parameters**
- **img**: (rows, cols) or (rows, cols, channels) ndimage
  - Input image.
- **size**: 2-tuple
  - Size of the patches
- **Nmax**: int
  - If > 0, number of patches to sample randomly
paddle.common.sparsity(X, axis=0)

paddle.common.gendata(d, N, s, K, D=None, rnd=<built-in method normal of mtrand.RandomState object at 0x7f5481387660>, dtype=<type 'numpy.float32'>)

Generates N vectors in d dimensions. Each vector is the linear combinations of s atoms sampled from a dictionary of size K.

rnd is a function returning an array of randomly generated numbers. The shape of the array is passed as the keyword argument size.
This module implements some functions used to handle data.

```
paddle.data.checkBSD(path)
```
Checks that the is the root directory of the standard distribution of the Berkeley segmentation dataset, as downloaded from http://www.eecs.berkeley.edu/Research/Projects/CS/vision/bsd3/BSDS300-images.tgz. It returns the absolute paths of the test and training directories.

```
paddle.data.draw_sample(path, W, N)
```
Draws randomly $N$ patches with size $W \times W$ from the images in path. It returns an ndarray with shape $W^2 \times N$.

```
paddle.data.draw_patches(traindir, W, N)
```

```
paddle.data.loadMNIST(path)
```
Loads the MNIST dataset. It returns the absolute paths of the test and training directories.
All examples listed here can be found in the examples directory, either in the paddle source distribution or in the installation. To find out the exact path of the installation, you can (for instance) issue the following commands from the interpreter:

```python
$ python
Python 2.6.5 (r265:79063, Apr 16 2010, 13:57:41)
[GCC 4.4.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import paddle
>>> print paddle.__path__
['/usr/local/lib/python2.6/dist-packages/paddle-1.0.0-py2.6.egg/paddle']
```

This means that the examples will be in the directory `/usr/local/lib/python2.6/dist-packages/paddle-1.0.0-py2.6.egg/paddle/examples`.

### 7.1 Experiments from [Basso10]

Some of the examples are the scripts used for the experiments presented in [Basso10], that is:

- the two synthetic experiments of section 4.1, `experiment_synthetic.py`;  
- the experiment on the Berkeley Segmentation Dataset (BSD, available here), section 4.2, `experiment_BSD.py`;  
- the experiment on the MNIST dataset (download the dataset as a single file from here), section 4.2, `experiment_MNIST.py`.

All experiments can be run simply as:

```bash
$ python path_to_examples/experiment_XXXX.py [arguments if any]
```

The classification experiment of section 4.3 has not been included since it does involve third-parties software and its implementation is slightly more involved than a plain script.

References

Acknowledgments

We would like to thank Salvatore Masecchia and Annalisa Barla, authors of the L1L2Py package, who allowed us to use the L1L2Py website stylesheets.
Symbols

_cost_cod() (in module paddle.common), 20
_cost_rec() (in module paddle.common), 20
_ist() (in module paddle.dual), 11
_ist() (in module paddle.tight), 17
_ist_fixed() (in module paddle.dual), 12
_pgd() (in module paddle.dual), 12
_pgd() (in module paddle.tight), 17
_pgd_fixed() (in module paddle.dual), 13
_replaceAtoms() (in module paddle.common), 20
_saveDict() (in module paddle.common), 21

C

checkBSD() (in module paddle.data), 23
cohere() (in module paddle.common), 19
cumcohere() (in module paddle.common), 19

D

draw_patches() (in module paddle.data), 23
draw_sample() (in module paddle.data), 23

G

gendata() (in module paddle.common), 22

I

img2patches() (in module paddle.common), 21
init() (in module paddle.dual), 9
init() (in module paddle.tight), 15

L

learn() (in module paddle.dual), 10
learn() (in module paddle.tight), 16
loadMNIST() (in module paddle.data), 23

P

print_frame_assessment() (in module paddle.common), 19

S

sparsity() (in module paddle.common), 21