mrainet Documentation

Release 0.1.1a

Wouter M. Kouw

Dec 18, 2019

Contents

MRAI-net is a set of neural networks designed to learn MRI-scanner acquisition-invariant representations. In other words, it extracts lower-dimensional feature vectors from two sets of images, such that there is minimal variance between them outside of tissue variation.

CHAPTER 1

Installation

mrainet is registered on PyPI and can be installed through:

```
pip install mrainet
```

1.1 Environment

Pip takes care of all dependencies, but the addition of these dependencies can mess up your current python environment. To ensure a clean install, it is recommended to set up a virtual environment using conda or virtualenv. To ease this set up, an environment file is provided, which can be run through:

```
conda env create -f environment.yml
source activate mrainet
```

For more information on getting started, see the Examples section.

CHAPTER 2

Classifiers

This page contains information on MRAI neural networks.

2.1 MRAI Convolutional Neural Network

class mrainet.mraicnn.**MRAIConvolutionalNeuralNetwork**(*patch_size=(31, 31), classes=[1, 2, 3], num_draw=10, num_kernels=[8], kernel_size=[(3, 3)], dense_size=[16, 8], strides=(1, 1), dropout=0.1, l2=0.001, margin=1, optimizer='rmsprop', batch_size=32, num_epochs=1*)

Network for MRI-scanner acquisition-invariant representation learning.

Class of convolutional neural networks that aim to map patches of two datasets from different MRI-scanners Methods include image processing operations, pair sampling and Siamese loss minimization.

Methods

compile_net(self)	Compile network architecture.
contrastive_loss(self, label, distance)	Contrastive Siamese loss.
extract_random_patches(self, X, Y)	Extract a random set of patches from image.
feedforward(self, patches, scan_ID)	Feed a set of patches forward through the network.
gen_index_combs(self, x)	Generate combinations of two index arrays.
index2patch(self, X, index)	Slice patches from an image at given indices.

Continued on next page

Table 1 – continued from previous page

l1_norm(self, x)	l1-norm for loss layer.
l2_norm(self, x)	l2-norm for loss layer.
load_model(self, model_fn, weights_fn)	Load model from filename.
matrix2sparse(self, X[, edge, remove_nans])	Map matrix to a sparse array format.
sample_pairs(self, X, y, Z, u[, num_draw])	Sample a set of pairs of patches from two images.
save_model(self, model_fn, weights_fn)	Save model to filename.
segment_image(self, X, model[, feed, ...])	Segment a new image using the trained network.
subsample_rows(self, X[, num_draw])	Take a random subsample of rows from X.
train(self, X, Y, Z, U[, num_targets])	Train the network using pairs of patches from the images.

compile_net(*self*)
Compile network architecture.

contrastive_loss(*self*, *label*, *distance*)
Contrastive Siamese loss.

For similar pairs, it consists of the squared Lp-distance. For dissimilar pairs, it consists of a hinge loss with respect to a margin parameter.

Parameters

label [int] Similarity label, 1=similar and 0=dissimilar

distance: float Lp-norm between pairs of patches mapped through the network.

Returns

float Loss value for current pair of patches.

extract_random_patches(*self*, *X*, *Y*)
Extract a random set of patches from image.

Parameters

X [array] Input image to sample patches from.

Y [array] Label image corresponding to X.

Returns

patches [array] Patches array, num patches by patch height by patch width by 1.

labels [array] Tissue label array corresponding to patches array.

feedforward(*self*, *patches*, *scan_ID*)
Feed a set of patches forward through the network.

Parameters

patches [array] Contains patches in form of number of patches by patch height by patch width by 1.

scan_ID [int] Scanner identification variable, indicating from which MRI-scanner these patches came from.

Returns

array Final layer representation of patches fed forward through the network.

gen_index_combs(*self*, *x*)
Generate combinations of two index arrays.

index2patch(*self*, *X*, *index*)
Slice patches from an image at given indices.

Parameters

X [array] input image

index [array] Row and column indices for the provided image.

Returns

patches [array] Number of patches by patch height by patch width by 1.

l1_norm(*self*, *x*)
l1-norm for loss layer.

l2_norm(*self*, *x*)
l2-norm for loss layer.

load_model(*self*, *model_fn*, *weights_fn*)
Load model from filename.

Parameters

model_fn [str] Filename of saved model.

weights_fn [str] Filename of saved weight matrix.

Returns

None

matrix2sparse(*self*, *X*, *edge=(0, 0)*, *remove_nans=False*)
Map matrix to a sparse array format.

Parameters

X [array] Matrix that should be mapped to sparse array format, may contain NaN's.

edge [tuple(int, int)] Dimensions of edge to ignore.

remove_nans [bool] Whether to remove NaN's as tissue labels.

Returns

sX [array] Original matrix mapped to (i,j,v) format where i corresponds to the i-th row of X, j to the j-column of X and v of the value at position (i,j) of X.

sample_pairs(*self*, *X*, *y*, *Z*, *u*, *num_draw=(10, 1)*)
Sample a set of pairs of patches from two images.

Parameters

X [array] slice from source MRI-scanner

y [array] source tissue index sparse array; where each row i,j,k consists of the pixel's row index i, the pixel's column index j and the pixel's tissue k.

Z [array] slice from target MRI-scanner

u [array] target tissue index sparse array; where each row i,j,k consists of the pixel's row index i, the pixel's column index j and the pixel's tissue k.

num_draw [tuple(int, int)] maximum number of patches to draw from (source, target)

Returns

P [list[A, B, a, b]] contains pairs of patches and scanner identifications

S [array] contains similarity labels between pairs

save_model (*self*, *model_fn*, *weights_fn*)
Save model to filename.

Parameters

model_fn [str] Filename to save model to.

weights_fn [str] Filename to save weight matrix to.

Returns

None

segment_image (*self*, *X*, *model*, *feed=True*, *mapost=False*, *scan_ID=1*)
Segment a new image using the trained network.

Parameters

X [array] new image that needs to be segmented.

model [sklearn-model] Trained classifier from scikit-learn, needs to have a predict method.

feed [bool] whether the extracted patches should be fed through the network, a value of False is for experimental purposes (def: True)

mapost [bool] whether to map the predictions to a maximum a posteriori form. (def: False)

scan_ID [int] scanner identification of new image.

Returns

preds [array] Label image of same size as input image, containing predictions made by the provided trained classifier.

subsample_rows (*self*, *X*, *num_draw=1*)
Take a random subsample of rows from X.

Parameters

X [array] Array to subsample from.

num_draw [int] Number of rows to subsample.

replace [bool] Whether to replace sampled rows.

Returns

array Smaller array.

train (*self*, *X*, *Y*, *Z*, *U*, *num_targets=1*)
Train the network using pairs of patches from the images.

Parameters

X [array] source scans, slices by height by width

Y [array] source labels, slices by height by width

Z [array] target scans, slices by height by width

U [array] target labels, slices by height by width, contains NaN's at unknown labels

num_targets [int] How many target labels to use.

Returns

None

2.2 MRAI Dense Neural Network

class mrainet.mraidnn.**MRAIDenseNeuralNetwork**(*patch_size=(31, 31), classes=[1, 2, 3], num_draw=10, dense_size=[16, 8], dropout=0.1, l2=0.001, margin=1, optimizer='rmsprop', batch_size=32, num_epochs=2*)

Network for MRI-scanner acquisition-invariant representation learning.

Class of fully-connected neural networks that aim to map patches of two datasets from different MRI-scanners Methods include image processing operations, pair sampling and Siamese loss minimization.

Methods

compile_net(self)	Compile network architecture.
contrastive_loss(self, label, distance)	Contrastive Siamese loss.
extract_random_patches(self, X, Y)	Extract a random set of patches from image.
feedforward(self, patches, scan_ID)	Feed a set of patches forward through the network.
gen_index_combs(self, x)	Generate combinations of two index arrays.
index2patch(self, X, index)	Slice patches from an image at given indices.
l1_norm(self, x)	l1-norm for loss layer.
l2_norm(self, x)	l2-norm for loss layer.
load_model(self, model_fn, weights_fn)	Load model from filename.
matrix2sparse(self, X[, edge, remove_nans])	Map matrix to a sparse array format.
sample_pairs(self, X, y, Z, u[, num_draw])	Sample a set of pairs of patches from two images.
save_model(self, model_fn, weights_fn)	Save model to filename.
segment_image(self, X, model[, feed, ...])	Segment a new image using the trained network.
subsample_rows(self, X[, num_draw])	Take a random subsample of rows from X.
train(self, X, Y, Z, U[, num_targets])	Train the network using pairs of patches from the images.

compile_net(*self*)

Compile network architecture.

contrastive_loss(*self*, *label*, *distance*)

Contrastive Siamese loss.

For similar pairs, it consists of the squared Lp-distance. For dissimilar pairs, it consists of a hinge loss with respect to a margin parameter.

Parameters

label [int] Similarity label, 1=similar and 0=dissimilar

distance: float Lp-norm between pairs of patches mapped through the network.

Returns

float Loss value for current pair of patches.

extract_random_patches(*self*, *X*, *Y*)

Extract a random set of patches from image.

Parameters

X [array] Input image to sample patches from.

Y [array] Label image corresponding to X.

Returns

patches [array] Patches array, num patches by patch height by patch width by 1.

labels [array] Tissue label array corresponding to patches array.

feedforward (*self*, *patches*, *scan_ID*)
Feed a set of patches forward through the network.

Parameters

patches [array] Contains patches in form of number of patches by patch height by patch width by 1.

scan_ID [int] Scanner identification variable, indicating from which MRI-scanner these patches came from.

Returns

array Final layer representation of patches fed forward through the network.

gen_index_combs (*self*, *x*)
Generate combinations of two index arrays.

index2patch (*self*, *X*, *index*)
Slice patches from an image at given indices.

Parameters

X [array] input image

index [array] Row and column indices for the provided image.

Returns

patches [array] Number of patches by patch height by patch width by 1.

l1_norm (*self*, *x*)
l1-norm for loss layer.

l2_norm (*self*, *x*)
l2-norm for loss layer.

load_model (*self*, *model_fn*, *weights_fn*)
Load model from filename.

Parameters

model_fn [str] Filename of saved model.

weights_fn [str] Filename of saved weight matrix.

Returns

None

matrix2sparse (*self*, *X*, *edge=(0, 0)*, *remove_nans=False*)
Map matrix to a sparse array format.

Parameters

X [array] Matrix that should be mapped to sparse array format, may contain NaN's.

edge [tuple(int, int)] Dimensions of edge to ignore.

remove_nans [bool] Whether to remove NaN's as tissue labels.

Returns

sX [array] Original matrix mapped to (i,j,v) format where i corresponds to the i-th row of X, j to the j-column of X and v of the value at position (i,j) of X.

sample_pairs (*self*, *X*, *y*, *Z*, *u*, *num_draw=(10, 1)*)
Sample a set of pairs of patches from two images.

Parameters

X [array] slice from source MRI-scanner

y [array] source tissue index sparse array; where each row i,j,k consists of the pixel's row index i, the pixel's column index j and the pixel's tissue k.

Z [array] slice from target MRI-scanner

u [array] target tissue index sparse array; where each row i,j,k consists of the pixel's row index i, the pixel's column index j and the pixel's tissue k.

num_draw [tuple(int, int)] maximum number of patches to draw from (source, target)

Returns

P [list[A, B, a, b]] contains pairs of patches and scanner identifications

S [array] contains similarity labels between pairs

save_model (*self*, *model_fn*, *weights_fn*)
Save model to filename.

Parameters

model_fn [str] Filename to save model to.

weights_fn [str] Filename to save weight matrix to.

Returns

None

segment_image (*self*, *X*, *model*, *feed=True*, *mapost=False*, *scan_ID=1*)
Segment a new image using the trained network.

Parameters

X [array] new image that needs to be segmented.

model [sklearn-model] Trained classifier from scikit-learn, needs to have a predict method.

feed [bool] whether the extracted patches should be fed through the network, a value of False is for experimental purposes (def: True)

mapost [bool] whether to map the predictions to a maximum a posteriori form. (def: False)

scan_ID [int] scanner identification of new image.

Returns

preds [array] Label image of same size as input image, containing predictions made by the provided trained classifier.

subsample_rows (*self*, *X*, *num_draw=1*)
Take a random subsample of rows from X.

Parameters

X [array] Array to subsample from.

num_draw [int] Number of rows to subsample.

replace [bool] Whether to replace sampled rows.

Returns

array Smaller array.

train(*self*, *X*, *Y*, *Z*, *U*, *num_targets=1*)

Train the network using pairs of patches from the images.

Parameters

X [array] source scans, slices by height by width

Y [array] source labels, slices by height by width

Z [array] target scans, slices by height by width

U [array] target labels, slices by height by width, contains NaN's at unknown labels

num_targets [int] How many target labels to use.

Returns

None

CHAPTER 3

Examples

In the /demos folder, there are a number of example scripts. These show potential use cases.

Here we walk through a simple version. First, make sure to import some necessary modules:

```
import numpy as np
import matplotlib.pyplot as plt

from mrainet.mraicnn import MRAIConvolutionalNeuralNetwork
from mrainet.util import extract_all_patches
from mrainet.viz import viz_embedding
```

Next, we should load some data. The folder *mrainet/demos/data/* contains a source MRI-scan and its segmentation as well as a target MRI-scan with an imcomplete segmentation.

```
# Load source MRI-scan and corresponding segmentation
X = np.load('./demos/data/subject01_GE2D_1.5T.npy')
Y = np.load('./demos/data/subject01_segmentation.npy')

# Load target MRI-scan and corresponding segmentation
Z = np.load('./demos/data/subject02_GE2D_3.0T.npy')
U = np.load('./demos/data/subject02_segmentation.npy')

# Note that U is missing a lot of labels
print('Proportion missing labels = ' + str(np.mean(~np.isnan(U.ravel()))))
```

Now, it's time to initialize and compile the network.

```
# Initialize and compile a small neural network
N = MRAIConvolutionalNeuralNetwork(patch_size=(31, 31),
                                   num_kernels=[8],
                                   kernel_size=[(3, 3)],
                                   dense_size=[16, 8],
                                   batch_size=128,
                                   num_epochs=4,
```

(continues on next page)

(continued from previous page)

```
                                        num_draw=10,
                                        margin=10)
```

Note that these options will result in a training set of 220 000 samples, and training might be quite expensive on a CPU laptop.

Now we'll call the training procedure, which automatically handles the pair sampling procedure.

```
# Call training procedure on source and target data
N.train(X, Y, Z, U, num_targets=1)
```

After training, we'll map all source and target patches extracted from the images to MRAI's learned representation.

```
# Extract all source patches and feed them through network.
PX = extract_all_patches(X[0], patch_size=(31, 31), edge=(15, 15), add_4d=True)
HX = N.feedforward(PX, scan_ID=0)

# Map label image to sparse array format
sY = N.matrix2sparse(Y[0], edge=(15, 15))

# Extract all target patches and feed them through network.
PZ = extract_all_patches(Z[0], patch_size=(31, 31), edge=(15, 15), add_4d=True)
HZ = N.feedforward(PZ, scan_ID=1)

# Map label image to sparse array format
sU = N.matrix2sparse(U[0], edge=(15, 15), remove_nans=False)

# Filter out missing target labels
HZ = HZ[~np.isnan(sU[:, 2]), :]
sU = sU[~np.isnan(sU[:, 2]), :]
```

Given 2-dimensional feature vectors for each patch, we can visualize them using a scatter plot:

```
# Create figure
fig, ax = plt.subplots(figsize=(15, 10))

# Call visualizer
viz_embedding(HX, sY[:, 2], marker='o', ax=ax)
viz_embedding(HZ, sU[:, 2], marker='x', ax=ax)
```

CHAPTER 4

Contact

Any comments, questions, or general feedback can be submitted to the repository's issues tracker.

Python Module Index

m

Index

T