Documentation: cnns4qspr

version

Orion Dollar, David Juergens, Nisarg Joshi, Saransh Jain

March 16, 2020

Contents

cnns4qspr: equivariant protein featurization

Guide

Welcome to the cnns4qspr package documentation! Below, you will find documentation for our four modules:

1. **loader.py** - a module for generation of CNN input features directly from pdb (protein data bank) files.
2. **visualizer.py** - a module for easy visualization of structural protein features, both before and within the deep CNN.
3. **featureizer.py** - a module for sending loaded features through a deep CNN, optimized to extract rich structural features into a single output vector.
4. **trainer.py** - a module for creating, training, and saving VAE models for regression/classification tasks using features generated by loader and featurizer.

The purpose of cnns4qspr is to enable anyone to perform high quality machine learning tasks with protein structure data.

loader.py

loader is acecssed via

```
from cnns4qspr import loader

import cnns4qspr.loader as loader
```

This module contains functions for loading a pdb file and calculating the atomic density fields for different atom types. The fields can then be used for plotting, or to send into the convolutional neural network.

`cnns4qspr.loader.`**`atoms_from_residues`**(`protein_dict`, `residue_list`)

This function finds all the atoms in a protein that are members of any residues in a list of residues.

Parameters:
- **protein_dict** (*dict, required*) – The dictionary of the protein, returned from load_pdb()
- **residue_list** (*list-like, required*) – The list of residues whose atoms we are finding coordinates for

`cnns4qspr.loader.`**`check_channel`**(`channel`, `filter_set`)

This function checks to see if a channel the user is asking to make a field

for is an allowed channel to ask for.

Parameters:
- **channel** (*str, required*) – The atomic channel being requested
- **filter_set** (*dict, required*) – The set of defined atomic filters

Returns: indicator for if the channel is allowed

Return type: boolean

`cnns4qspr.loader.`**`find_channel_atoms`**(`channel`, `protein_dict`, `filter_set`)

This function finds the coordinates of all relevant atoms in a channel.

It uses the filter set to constrcut the atomic channel (i.e., a channel can

be composed of multiple filters).

Parameters:
- **channel** (*str, required*) – The atomic channel being constructed
- **protein_dict** (*dict, required*) – The dictionary of the protein, returned from load_pdb()
- **filter_set** (*dict, required*) – The set of available filters to construct channels with

Returns: array containing the coordinates of each atom that is relevantto the channel

Return type: numpy array

`cnns4qspr.loader.`**`grid_positions`**(`grid_array`)

This function returns the 3D meshgrids of x, y and z positions. These cubes will be flattened, then used as a reference coordinate system to place the actual channel densities into.

Parameters: **grid_positions** (*pytorch tensor*) – lineraly spaced grid

Returns: meshgrid array of the x, y and z positions.

Return type: array

`cnns4qspr.loader.`**`load_pdb`**`(path)`

Loads all of the atomic positioning/type arrays from a pdb file.

The arrays can then be transformed into density (or "field") tensors before

being sent through the neural network.

Parameters: **path** (*str, required*) – The full path to the pdb file being voxelized.

Returns: A dictionary containing the following arrays fromthe pdb file: num_atoms, atom_types, positions, atom_type_set, xcoords, ycoords, zcoords, residues, residue_set

Return type: dictionary

`cnns4qspr.loader.`**`make_fields`**`(protein_dict, channels=['CA'], bin_size=2.0, num_bins=50)`

This function takes a protein dict (from load_pdb function) and outputs a

large tensor containing many atomic "fields" for the protein.

The fields describe the atomic "density" (an exponentially decaying function

of number of atoms in a voxel) of any particular atom type.

Parameters:

- **protein_dict** (*dict, requred*) – dictionary from the load_pdb function
- **channels** (*list-like, optional*) – the different atomic densities we want fields for theoretically these different fields provide different chemical information full list of available channels is in protein_dict['atom_type_set']
- **bin_size** (*float, optional*) – the side-length (angstrom) of a given voxel in the box that atomic densities are placed in
- **num_bins** (*int, optional*) – how big is the cubic field tensor side length (i.e., num_bins is box side length)

Returns: A list of atomic density tensors (50x50x50), one for eachchannel in channels

Return type: dictionary

`cnns4qspr.loader.`**`shift_coords`**`(protein_dict)`

This function shifts the coordinates of a protein so that it's coordinates are in the center of the field tensor.

Parameters: **protein_dict** (*dict*) – A dictionary of information from the first part of the load_pdb function.

Returns: The original protein dict but with an added value containingthe coordinates of the protein shifted to the origin.

Return type: dictionary

`cnns4qspr.loader.`**`voxelize`**`(path, channels=['CA'])`

This function creates a dictionary of tensor fields directly from a pdb file.

These tensor fields can be plotted, or sent directly into the cnn for

plotting internals, or sent all the way through a cnn/vae to be used for training.

Parameters:

- **path** (*str, required*) – path to a .pdb file
- **channels** (*list of strings, optional*) – The list of atomic channels to be included in the output dictionary, one field for every channel. Any channels from points 1-4 below may be combined in any order. i.e., one could call voxelize with the channels parameter as channels=[‘charged’, ‘CB’, ‘GLY’, ‘polar’, …etc]. Note that voxelization for channels containing more atoms will take longer. any of the following atom types [‘C’ ‘CA’ ‘CB’ ‘CD’ ‘CD1’ ‘CD2’ ‘CE’ ‘CE1’ ‘CE2’ ‘CE3’ ‘CG’ ‘CG1’ ‘CG2’ ‘CH2’ ‘CZ’ ‘CZ2’ ‘CZ3’ ‘N’ ‘ND1’ ‘ND2’ ‘NE’ ‘NE1’ ‘NE2’ ‘NH1’ ‘NH2’ ‘NZ’ ‘O’ ‘OD1’ ‘OD2’ ‘OE1’ ‘OE2’ ‘OG’ ‘OG1’ ‘OH’ ‘OXT’ ‘SD’ ‘SG’] Any canonical residue in the protein, using the three letter residue code, all caps (NOTE: the residue must actually exist in the protein) e.g., [‘LYS’, ‘LEU’, ‘ALA’] The ‘other’ channel options: ‘backbone’, ‘sidechains’ There are 6 channels corresponding to specific types of residues: ’charged’, ‘polar’, ‘nonpolar’, ‘amphipathic’, ‘acidic’, ‘basic’

Returns: a dictionary containing a voxelized atomic fields, one for each channel requested. Each field has shape = ([1, 1, 50, 50, 50])

Return type: dictioanry

visualizer.py

visualizer is accessed via

```
from cnns4qspr import visualizer

import cnns4qspr.visualizer as visualizer
```

This module contains functions to plot atomic density fields before they go into a model, as well as what the density fields have been transformed into at certain points within the model.

`cnns4qspr.visualizer.`**`plot_field`**`(field, color='deep', threshold=0.2, alpha=0.7, show=True)`

This function takes a tensorfield and plots the field density in 3D space. The field describes an atomic “density” at each voxel.

Parameters:

- **field** (*pytorch tensor, required*) – A field from a field dictionary that was output by either *voxelize* or *make_fields*.
- **color** (*str, optional*) – The color scheme to plot the field. Any of the Plotly continuous color schemes. ‘deep’ and ‘ice_r’ are recommended as good baselines.
- **threshold** (*float, optional*) – The threshold intensity that a voxel must have in order to be included in the plot.
- **alpha** (*float, optional*) – Amount of transparency to use in plotted marks.
- **show** (*boolean, optional*) – Whether to show the plot. If false, the plotly fig object is returned.

Returns: If show=False, a plotly figure object is returned

Return type: plotly figure object

`cnns4qspr.visualizer.`**`plot_internals`**`(model, field, block=0, channel=0, threshold_on=True, feature_on=False)`

This function enables visualization of the output of various convolutional layers inside the model.

Parameters:

- **model** (*pytorch neural network, required*) – The CNN model that data can be sent through. The model object can be constructed by using featurizer.load_cnn()
- **field** (*pytorch tensor, required*) – A field from a field dictionary (see make_fields or voxelize)
- **block** (*int, optional*) – Any of the 5 major blocks contained in the model object. This integer determines at which stage of the CNN the data will be plotted from.
- **channel** (*int, optional*) –
- **threshold_on** (*boolean, optional*) – If threshold_on=True, plot will be constructed with the default threshold value from plot_fields (0.2). If threshold_on=False, plot is constructed with threshold 0.0001
- **feature_on** (*boolean, optional*) – Whether or not to return the feature vector of the field that was sent through the model to make this plot.

Returns: the feature vector that results from the particular field being sent all the way through the network. Only returns if feature_on=True

Return type: pytorch tensor

featurizer.py

featurizer is accessed via

```
from cnns4qspr import featurizer

import cnns4qspr.featurizer as featurizer
```

This module contains functions for loading a pre-trained convolutional neural network and getting convolutional vector from the tensor fields which can be used for VAE or for plotting the internals.

`cnns4qspr.featurizer.`**`featurize`**`(field, channels='all')`

Takes a dictionary of voxelized tensor fields (50x50x50) and convolutes to a single 256 element scalar feature vector

Parameters: **field** (*dict, required*) – Dictionary storing channel type as key and voxelized input tensor as value

Returns: Dictionary storing channel type as key anddense structural feature vector as value

Return type: feature_vec (dict)

`cnns4qspr.featurizer.`**`gen_feature_set`**`(pdb_path, channels=['CA'], save=False, save_fn='feature_set.npy', save_path='./')`

This function takes a directory of pdbs and either saves them or returns an nx256 np array where n is number of pdbs.

Parameters:

- **pdb_path** (*path, required*) – path of the pdb file
- **channels** (*list-like, optional*) – for the specific channels we want the arrays for.

Returns: returns the array of nx256 where nis the number of pdbs.

Return type: feature_set (numpy array)

`cnns4qspr.featurizer.`**`load_cnn`**`(checkpoint_fn, n_input=1)`

This function loads the pretrained SE3 Convolutional Neural Net to be used as a feature extractor from the users input pdb file.

Parameters: **checkpoint_fn** (*str, required*) – The filename of the ckpt file used to load the model.

Returns: A fully trained pytorch network objectwith 256 output nodes.

Return type: model (pytorch.network)

trainer.py

trainer is accessed via

```
from cnns4qspr import trainer

import cnns4qspr.trainer as trainer
```

This module contains a Trainer class for quickly and easily building VAE or FeedForward property predictors from the extracted structural protein features.

class cnns4qspr.trainer.**Trainer**(type='classifier', network_type='vae', output_size=3, input_size=256, latent_size=3, encoder=[128], decoder=[128], predictor=[128, 128])

Loads VAE or FeedForward pytorch network classes and allows the user to quickly define its architecture and train. This class has pre-defined options for both classifier and regressor model types. If classifier is chosen, the output size must match the number of labels being trained on.
Trained models can be saved as .ckpt files which can later be loaded for further model evaluation and/or training.

Parameters:

- **type** (*str*) – The target of the neural network (classifier or regressor)
- **network_type** (*str*) – The type of neural network (vae or feedforward)
- **output_size** (*int*) – Number of output nodes (defaults to 1 if regressor)
- **input_size** (*int*) – The number of input nodes (defaults to 256, the size of the featurized protein vectors)
- **latent_size** (*int*) – The number of latent nodes in VAE
- **encoder** (*list*) – A list of hidden layer sizes for encoder network
- **decoder** (*list*) – A list of hidden layer sizes for decoder network
- **predictor** (*list*) – A list of hidden layer sizes for predictor network (*note* if network_type is vae then predictor is appended to the latent space, if network_type is feedforward then predictor takes the structural feature vector as input)

type
The target of the neural network (classifier or regressor)

Type: str

network_type
The type of neural network (vae or feedforward)

Type: str

latent_size
The number of latent nodes in VAE

Type: int

network
Instantiation of VAE or FeedForward pytorch network class

Type: pytorch Module

architecture
Dictionary containing the size and number of all neural net layers

Type: dict

current_state
Dictionary containing the most recent values of neural net parameters (weights, biases, optimizer history, epoch #)

Type: dict

checkpoint

Dictionary containing the saved values of the neural net parameters at the networks' best validation performance (user defines based on accuracy or loss)

Type: dict

load_state
Indicates whether a previous model has been loaded

Type: bool

total_losses
Saved series of the total loss during both training and validation

Type: dict

vae_losses
Saved series of the VAE loss during both training and validation

Type: dict

predictor_losses
Saved series of the predictor loss during both training and validation

Type: dict

accuracies
Saved series of predictor accuracies during both training and validation

Type: dict

latent_means
Saved series of latent space mean values during both training and validation

Type: dict

label_history
Saved series of label values during both training and validation

Type: dict

n_epochs
Number of epochs the model has been trained for

Type: int

load (`checkpoint_path`**)**
Loads model from saved .ckpt checkpoint state.

Parameters: **checkpoint_path** (*str*) – Path to checkpoint file (must include .ckpt filename)

predict (`data`**)**
Predicts output given a set of input data for a fully trained model.

Parameters: **data** (*np.array*) – nxm feature matrix where n = # of samples and m = # of features
Returns: nx1 prediction matrix where n = # of samples
Return type: prediction (np.array)

save (`mode='best'`**,** `path='./'`**,** `fn='default'`**)**
Saves the model to .ckpt file. Mode can be chosen to save either the best performing model or the most recent state of the model.

Parameters:

- **mode** (*str*) – Indicates which model to save (best or current)
- **path** (*str*) – Path to directory where model will be stored
- **fn** (*str*) – Filename of saved model

train(data, labels, n_epochs, batch_size, val_split=0.2, optimizer='default', learning_rate=0.0001, store_best=True, criterion='acc', verbose=True**)**

Trains model. Feature data and labels (or targets) must be input separately. By default, 20% of training data is used for model validation and the Adam optimizer is used with a learning rate of 1e-4. Data and model states are stored as Trainer attributes (defined during instantiation).

Parameters:

- **data** (*np.array*) – nxm feature matrix where n = # of samples and m = # of features
- **labels** (*np.array*) – nx1 target matrix where n = # of samples. Only one target can be trained on at a time
- **n_epochs** (*int*) – Number of epochs to train the model
- **batch_size** (*int*) – Size of training and validation batches
- **val_split** (*float*) – Fraction of data to be used for model validation
- **optimizer** (*torch.optim*) – Model optimizer. Must be a pytorch optimizer object
- **learning_rate** (*float*) – Learning rate of optimizer
- **store_best** (*bool*) – Checks model performance and stores as an attribute if improved
- **criterion** (*str*) – Criterion for model improvement (acc or loss)
- **verbose** (*bool*) – Prints progress bar and metrics during training

Indices and tables

- **genindex**
- **modindex**
- **search**

Index

Python Module Index

c