env_sample_generation.py

`add_noise(state, terminated, bound)`

This function adds noise to a state to simulate environmental variability. It ensures the noisy state remains within defined bounds:

Parameters:

- `state`: Current state of the environment.
- `terminated`: Boolean indicating if the episode has terminated.
- `bound`: List of bounds for each state variable.

Returns:

- `noisy_state`: The noisy version of the input state.

`generate_episodes(num_samples, max_episode_length, bound)`

This function generates a specified number of episodes, each consisting of state transitions, actions, and rewards:

Parameters:

- `num_samples`: Number of samples to generate.
- `max_episode_length`: Maximum length of each episode.
- `bound`: List of bounds for each state variable.

Returns:

- `episode_inputs`: Tensor containing input states and actions.
- `episode_outputs`: Tensor containing next states, rewards, and termination flags.
- `df`: DataFrame containing the collected samples.

Saving Data

- **Training and Test Datasets**: Saved as pickle files for later use.
- **Samples DataFrame**: Saved as a CSV file for analysis.

Files Saved:

- `env_name_k_train_dataset.pkl`: Training dataset for the environment.
- `env_name_k_test_dataset.pkl`: Testing dataset for the environment.
- `env_name_k_sample.csv`: CSV file containing raw samples data.

---

state_var_graph.py

For a particular environment, this notebook plots an individual graph of state variable vs time for each model.

---

create_models.py

This notebook trains PETS, Monte Carlo Dropout and Bayesian neural network models.

1. Gaussian and Ensemble Gaussian Network (PETS)

This code defines two classes for neural networks used in probabilistic modeling: `GaussianNetwork` and `EnsembleGaussianNetwork`. Both classes use PyTorch and are designed for tasks where the output is a Gaussian distribution characterized by its mean and standard deviation.

GaussianNetwork Class

Overview

`GaussianNetwork` is a neural network that outputs the parameters of a Gaussian distribution (mean and standard deviation) given an input tensor.

Parameters:

- `input_dim` (int): The dimension of the input features.

- `hidden_dim` (int): The dimension of the hidden layers.
- `output_dim` (int): The dimension of the output features (mean and standard deviation).

Forward Method:

Parameters:

- `x` (torch.Tensor): Input tensor.

Returns:

- `mean` (torch.Tensor): The predicted mean of the Gaussian distribution.
- `std` (torch.Tensor): The predicted standard deviation of the Gaussian distribution, clamped to ensure numerical stability.

EnsembleGaussianNetwork Class

Overview

`EnsembleGaussianNetwork` is a neural network ensemble consisting of multiple `GaussianNetwork` instances. It combines the outputs of these models to provide an ensemble prediction.

Parameters:

- `num_models` (int): The number of `GaussianNetwork` models in the ensemble.
- `input_dim` (int): The dimension of the input features.
- `hidden_dim` (int): The dimension of the hidden layers.
- `output_dim` (int): The dimension of the output features (mean and standard deviation).

Forward Method

Parameters:

- `x` (torch.Tensor): Input tensor.

Returns:

- `means` (torch.Tensor): Stacked means from all models in the ensemble.
- `stds` (torch.Tensor): Stacked standard deviations from all models in the ensemble.

2. MC_Dropout_Net Class

Parameters

- **input_size** (int): The size of the input layer.
- **output_size** (int): The size of the output layer.
- **num_hidden_layers** (int): The number of hidden layers in the network.
- **hidden_layer_nodes** (int): The number of nodes in each hidden layer.
- **activation** (nn.Module): The activation function to be used in the hidden layers.
- **dropout_prob** (float): The dropout probability for regularization.
- **num_network** (int): The number of sub-networks to be used for Monte Carlo Dropout.

3. BayesianNN Class

Parameters

- **input_size** (int): The size of the input layer.
- **output_size** (int): The size of the output layer.
- **num_hidden_layers** (int): The number of hidden layers in the network.
- **hidden_layer_nodes** (int): The number of nodes in each hidden layer.
- **activation** (nn.Module): The activation function to be used in the hidden layers.

---

env_sample_generation.py

Sample Class

The `Sample` class generates samples from a given model and environment, storing them in a DataFrame and saving them to a CSV file.

Parameters

- **model** (nn.Module): The neural network model used to predict the next state given the current state and action.
- **env** (gym.Env): The OpenAI Gym environment to interact with.

- **df_data** (pd.DataFrame): A DataFrame to store the generated samples.

Methods

`__init__`

The constructor method initializes the `Sample` class with the specified model and environment.

`generateSample`

The method generates samples from the environment using the model, saves the samples in a DataFrame, and writes the DataFrame to a CSV file.

Parameters

- **filename** (str): The name of the file to save the generated samples.
- **num_samples** (int): The number of samples to generate.
- **max_episode_length** (int, optional): The maximum length of an episode. Defaults to 200.
- **noise** (float, optional): The noise to add to the next state. Defaults to 0.

---

mdp_construction_all_model_samples.py

MDP Class

Overview

The `MDP` class is designed to handle Markov Decision Processes (MDPs) by processing samples from various models, discretizing variables, constructing the MDPs, and providing visualization and comparison capabilities.

Class: `MDP`

`__init__(self, env_name, num_samples, action_discretize_bins=None, action_bound=None)`

Parameters:

- `env_name` (str): Name of the environment.
- `num_samples` (str): Number of samples to be processed.
- `action_discretize_bins` (int, optional): Number of bins for action discretization. Default is `None`.
- `action_bound` (list, optional): Bounds for action discretization. Default is `None`.

Description:

Initializes the MDP object by loading sample data for the original, PETS, Bayesian, and Monte Carlo Dropout models. It optionally discretizes actions if `action_discretize_bins` and `action_bound` are provided.

`discretize(self, bound, no_of_bins, include_out_of_bounds)`

Parameters:

- `bound` (list): Bounds for discretizing variables.
- `no_of_bins` (list): Number of bins for each variable.
- `include_out_of_bounds` (bool): Whether to include out-of-bound values in the discretization.

Description:

Discretizes the variables in the sample data according to the provided bounds and number of bins. It handles both within-bounds and out-of-bounds cases.

`discretize_variables(self, df, variable, nx_variable, bound, no_of_bins)`

Parameters:

- `df` (pd.DataFrame): DataFrame containing the data to be discretized.
- `variable` (str): Name of the variable to be discretized.
- `nx_variable` (str): Name of the next variable to be discretized.
- `bound` (list): Bounds for discretization.
- `no_of_bins` (int): Number of bins for discretization.

Description:

Discretizes a variable within specified bounds and number of bins. Optionally discretizes the next variable if provided.

`discretize_variables_within_bounds(self, df, variable, nx_variable, bound, no_of_bins)`

Parameters:

- `df` (pd.DataFrame): DataFrame containing the data to be discretized.
- `variable` (str): Name of the variable to be discretized.
- `nx_variable` (str): Name of the next variable to be discretized.
- `bound` (list): Bounds for discretization.
- `no_of_bins` (int): Number of bins for discretization.

Description:

Discretizes variables strictly within the provided bounds and number of bins. Removes rows with out-of-bound values.

`mdp_construct(self, bound, no_of_bins, include)`

Parameters:

- `bound` (list): Bounds for discretizing variables.
- `no_of_bins` (list): Number of bins for each variable.
- `include` (bool): Whether to include out-of-bound values in the discretization.

Description:

Constructs the MDP by discretizing variables, computing transition probabilities, and saving the results. It also generates visualizations of the constructed MDPs.

`prob_construct(self, df)`

Parameters:

- `df` (pd.DataFrame): DataFrame containing discretized data.

Description:

Constructs the probability transition matrix for the given discretized data.

`comparison(self, no_of_bins, include)`

Parameters:

- `no_of_bins` (int): Number of bins used for discretization.

- `include` (bool): Whether out-of-bound values were included in the discretization.

Description:

Compares the sizes of the constructed MDPs and calculates the KL divergence between different models. Saves the comparison results to a file.

`mdp_size(self, df)`

Parameters:

- `df` (pd.DataFrame): DataFrame containing the MDP data.

Description:

Calculates and returns the number of unique states in the MDP.

`kl_divergence(self, df1, df2)`

Parameters:

- `df1` (pd.DataFrame): First DataFrame containing MDP data.
- `df2` (pd.DataFrame): Second DataFrame containing MDP data.

Description:

Calculates and returns the KL divergence between the two MDPs.

`visualize_discreteMDP_networkx(self, df=None, save_path=None)`

Parameters:

- `df` (pd.DataFrame, optional): DataFrame containing the MDP data to be visualized. Default is `None`.
- `save_path` (str, optional): Path to save the visualization. Default is `None`.

Description:

Visualizes the MDP using NetworkX. If a save path is provided, the visualization is saved as a PNG file; otherwise, it is displayed.

---

onnx_to_constraints.py

ONNX to Constraints Conversion

This notebook demonstrates the process of converting an ONNX model to constraints, specifically for use with constraint solvers like Z3. The steps involve loading the ONNX model, extracting weights, marking specific activation layers, creating Z3 variables, and generating the corresponding SMT constraints.