AlphaZeroChomp

Release 1.0.0

Alessandro Canzonieri

Jun 15, 2024

CONTENTS:

Contents:

CHAPTER

ONE

ALPHAZEROCHOMP

class main.**CustomArgumentParser**(*prog=None*, *usage=None*, *description=None*, *epilog=None*, *parents=[]*, *formatter_class=<class 'argparse.HelpFormatter'>*, *prefix_chars='-'*, *fromfile_prefix_chars=None*, *argument_default=None*, *conflict_handler='error'*, *add_help=True*, *allow_abbrev=True*, *exit_on_error=True*)

error(*message: string*)

Prints a usage message incorporating the message to stderr and exits.

If you override this in a subclass, it should not return – it should either exit or raise an exception.

class Resnet.**ResBlock**(*num_hidden*)

A Residual Block for a neural network.

Parameters

num_hidden (*int*) – Number of hidden units in the convolutional layers.

forward(*x*)

Performs the forward pass of the residual block.

forward(*x*)

The *forward* function defines the forward pass of the ResNet block.

class Resnet.**ResNet**(*args*)

create_coordinate_matrix()

Creates a symmetric matrix with unique diagonal elements and symmetric off-diagonal elements.

The matrix is of size *(max_size, max_size)* with diagonal elements from *0* to *max_size - 1* and symmetric off-diagonal elements that encode positional symmetries.

Returns

A symmetric matrix with unique diagonal elements and symmetric off-diagonal elements.

Return type

torch.Tensor

forward(*x_input*, *current_actions=None*, *opponent_actions=None*)

Define the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

mask_and_renorm_NoSoftmax(*x_input*, *unrenorm_policy*)

Masks and renormalizes the policy without using the Softmax function.

This method applies a mask to the policy to exclude invalid moves and renormalizes the policy without using the Softmax function. It works best with args['MCTS_set_equal_prior'] = True and allows faster convergence of the ResNet network. This function can handle both batched and unbatched inputs.

Parameters

- **x_input** (*torch.Tensor*) – Input tensor representing the state, either batched (3D) or unbatched (2D).
- **unrenorm_policy** (*torch.Tensor*) – Unrenormalized policy tensor to be masked and renormalized.

Returns
Masked and renormalized policy tensor.

Return type
torch.Tensor

mask_and_renormalize(*x_input*, *unrenorm_policy*)

Masks and renormalizes the policy using the Softmax function.

This method applies a mask to the policy to exclude invalid moves, then renormalizes the policy using the Softmax function. It can handle both batched and unbatched inputs.

Parameters

- **x_input** (*torch.Tensor*) – Input tensor representing the state, either batched (3D) or unbatched (2D).
- **unrenorm_policy** (*torch.Tensor*) – Unrenormalized policy tensor to be masked and renormalized.

Returns
Masked and renormalized policy tensor.

Return type
torch.Tensor

prepare_data(*x_input*, *current_actions=None*, *opponent_actions=None*)

Prepares input data for a neural network model by incorporating current and opponent actions.

Parameters

- **x_input** (*torch.Tensor*) – The state of the game.
- **current_actions** (*list or torch.Tensor, optional*) – Actions taken by the current player. For self-play, it is a list of tuples with coordinates of actions. For training, it is a matrix of dimensions (max_size, max_size) with +1 at action coordinates.
- **opponent_actions** (*list or torch.Tensor, optional*) – Actions taken by the opponent player. For self-play, it is a list of tuples with coordinates of actions. For training, it is a matrix of dimensions (max_size, max_size) with -1 at action coordinates.

Returns
The modified input data after processing, which includes the current player's actions, opponent's actions, the original input state, and a coordinates matrix stacked together.

Return type
torch.Tensor

`set_device()`

Set the device for the PyTorch model based on the specified model device and availability.

This function updates the model to use 'mps' if specified and available, otherwise 'cuda' if specified and available, and defaults to 'cpu' if neither is available.

Parameters
`None`

Return type
None

`set_seed()`

Set all possible seeds to make the experiment reproducibol

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX