HyPyML

Release 1.0

Manaswin Oddiraju

May 24, 2024

CONTENTS

1 API Reference 1

1.1 Ensemble Model . . . 1

1.2 Models . . . 1

1.3 Configs . . . 3

1.4 Training Utils . . . 4

2 Indices and tables 7

Python Module Index 9

Index 11

CHAPTER

ONE

API REFERENCE

1.1 Ensemble Model

class hypyml.ensemble.**HybridModel**(*config:* HybridConfig)

Torch Module for Serial Hybrid Physics Models.

Parameters

- **models_mlp** (*Torch module list of all MLP modules.*)
- **models_cnn** (*Torch module list of all CNN modules.*)
- **models_physics** (*Torch module list of all Physics modules.*)
- **unmodified_inputs** (*Indices of the inputs that are to be passed directly to the*)
- **model.** (*model. These are appended to the outputs of the previous*)
- **architecture** (*Dict with key corresponding to model name and value being a model*)
- **config.**

forward(*x*, *phy_args=None*)

Function to run inference on the hybrid model.

1.2 Models

class hypyml.models.**MLP**(*config:* MLPConfig)

Multilayer Perceptron (MLP) neural network model.

config

Type

Instance of MLPConfig dataclass.

Note: This class implements a Multilayer Perceptron (MLP) neural network model. It takes a configuration dictionary with parameters such as hidden layer size, input and output dimensions, and the number of hidden layers.

forward(*x*)

Forward pass of the MLP model.

Parameters

x (*torch.Tensor*) – Input tensor.

Returns

Output tensor.

Return type

torch.Tensor

class hypyml.models.**Physics**(**args*, ***kwargs*)

Custom Autograd function to enable backpropagation on Custom Physics Models.

Attributes: config: Instance of PhysicsConfig.

static backward(*ctx*, *grad_output*)

Define a formula for differentiating the operation with backward mode automatic differentiation.

This function is to be overridden by all subclasses. (Defining this function is equivalent to defining the `vjp` function.)

It must accept a context `ctx` as the first argument, followed by as many outputs as the *forward()* returned (None will be passed in for non tensor outputs of the forward function), and it should return as many tensors, as there were inputs to *forward()*. Each argument is the gradient w.r.t the given output, and each returned value should be the gradient w.r.t. the corresponding input. If an input is not a Tensor or is a Tensor not requiring grads, you can just pass None as a gradient for that input.

The context can be used to retrieve tensors saved during the forward pass. It also has an attribute `ctx.needs_input_grad` as a tuple of booleans representing whether each input needs gradient. E.g., *backward()* will have `ctx.needs_input_grad[0] = True` if the first input to *forward()* needs gradient computed w.r.t. the output.

static forward(*ctx*, *x*, *forward_fun*, *jacobian_fun*, *args=None*)

Define the forward of the custom autograd Function.

This function is to be overridden by all subclasses. There are two ways to define forward:

Usage 1 (Combined forward and ctx):

```
@staticmethod
def forward(ctx: Any, *args: Any, **kwargs: Any) -> Any:
    pass
```

- It must accept a context ctx as the first argument, followed by any number of arguments (tensors or other types).
- See combining-forward-context for more details

Usage 2 (Separate forward and ctx):

```
@staticmethod
def forward(*args: Any, **kwargs: Any) -> Any:
    pass

@staticmethod
def setup_context(ctx: Any, inputs: Tuple[Any, ...], output: Any) -> None:
    pass
```

- The forward no longer accepts a ctx argument.
- Instead, you must also override the `torch.autograd.Function.setup_context()` staticmethod to handle setting up the `ctx` object. `output` is the output of the forward, `inputs` are a Tuple of inputs to the forward.
- See extending-autograd for more details

The context can be used to store arbitrary data that can be then retrieved during the backward pass. Tensors should not be stored directly on *ctx* (though this is not currently enforced for backward compatibility). Instead, tensors should be saved either with `ctx.save_for_backward()` if they are intended to be used in `backward` (equivalently, `vjp`) or `ctx.save_for_forward()` if they are intended to be used for in `jvp`.

1.3 Configs

`class hypyml.configs.`**`HybridConfig`**(*models: dict[str,* MLPConfig | PhysicsConfig | *Module]*, *model_inputs: dict[str, dict[str, list[int] | None]] | None = None*)

Config for Ensemble Models.

`models`

Contains Modelname as keys and an instance of ModelConfigs as values.

Type

dict

`model_inputs`

By default, the Ensemble model operates sequentially, using the output of the preceding model as input for the next. Setting this dict to a non-empty value overrides that behavior. Keys are model names; values are dicts specifying input customization. Each inner dict holds model names as keys and specifies how to stack inputs: - 'None' stacks the entire tensor. - A list of ints stacks only specified dimensions. Use "Input" if the input to this model matches the hybrid model's original input.

Type

dict

`class hypyml.configs.`**`MLPConfig`**(*num_input_dim: int*, *num_hidden_dim: int*, *num_output_dim: int*, *num_hidden_layers: int*, *activation_functions: str*)

Configuration class for the Multilayer Perceptron (MLP) model.

`layers`

Total number of layers in the MLP (including hidden and output layers).

Type

int

`num_input_dim`

Number of input dimensions to the MLP.

Type

int

`num_hidden_dim`

Number of hidden dimensions in each hidden layer.

Type

int

num_output_dim

Number of output dimensions from the MLP.

Type

int

num_hidden_layers

Number of hidden layers in the MLP.

Type

int

activation_functions

String representation of the activation functions used in the MLP.

Type

str

class hypyml.configs.**PhysicsConfig**(*forward_func: Callable[[Tensor], Tensor]*, *jacobian_func: Callable[[Tensor], Tensor]*)

Configuration class for physics-related functions.

forward_func

Forward function of the physics model.

Type

Callable[[torch.Tensor], torch.Tensor]

jacobian_func

Function to compute the Jacobian of the physics model.

Type

Callable[[torch.Tensor], torch.Tensor]

1.4 Training Utils

hypyml.train_utils.**train**(*model*, *train_loader*, *test_loader*, *optimizer*, *loss_fn*, *scheduler*, *filename*, *epochs*, *print_training_loss=True*, *save_frequency=50*)

Training Function.

Parameters

- **train_loader** (*torch.Torch_Dataloader*) – Torch Dataloader with training samples.
- **test_loader** (*torch.Torch_Dataloader*) – Torch Dataloader with validation samples.
- **optimizer** (*torch.optim.Optimizer*) – Initialized Torch Optimizer.
- **loss_fn** (*callable*) – Loss function for training.
- **scheduler** (*torch.optim.lr_scheduler*) – Learning rate scheduler.
- **filename** (*str*) – File name for saving the trained model.
- **epochs** (*int*) – Number of training epochs.
- **print_training_loss** (*bool*) – Option to toggle printing epoch loss.
- **save_frequency** (*int*) – Number of epochs per which to save the model parameters to disk.

Return type

Trained Hybrid Model.

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

h

hypyml.configs, 3
hypyml.ensemble, 1
hypyml.models, 1
hypyml.train_utils, 4

INDEX

A

activation_functions (*hypyml.configs.MLPConfig attribute*), 4

B

backward() (*hypyml.models.Physics static method*), 2

C

config (*hypyml.models.MLP attribute*), 1

F

forward() (*hypyml.ensemble.HybridModel method*), 1
forward() (*hypyml.models.MLP method*), 1
forward() (*hypyml.models.Physics static method*), 2
forward_func (*hypyml.configs.PhysicsConfig attribute*), 4

H

HybridConfig (*class in hypyml.configs*), 3
HybridModel (*class in hypyml.ensemble*), 1
hypyml.configs
    module, 3
hypyml.ensemble
    module, 1
hypyml.models
    module, 1
hypyml.train_utils
    module, 4

J

jacobian_func (*hypyml.configs.PhysicsConfig attribute*), 4

L

layers (*hypyml.configs.MLPConfig attribute*), 3

M

MLP (*class in hypyml.models*), 1
MLPConfig (*class in hypyml.configs*), 3
model_inputs (*hypyml.configs.HybridConfig attribute*), 3
models (*hypyml.configs.HybridConfig attribute*), 3
module
    hypyml.configs, 3
    hypyml.ensemble, 1
    hypyml.models, 1
    hypyml.train_utils, 4

N

num_hidden_dim (*hypyml.configs.MLPConfig attribute*), 3
num_hidden_layers (*hypyml.configs.MLPConfig attribute*), 4
num_input_dim (*hypyml.configs.MLPConfig attribute*), 3
num_output_dim (*hypyml.configs.MLPConfig attribute*), 3

P

Physics (*class in hypyml.models*), 2
PhysicsConfig (*class in hypyml.configs*), 4

T

train() (*in module hypyml.train_utils*), 4