Transformer-PhysX

Release 0.0.1

May 11, 2022

Getting Started:

Welcome to the Transformer-PhysX documentation. Transformer-PhysX is an actively developed project for using transformer models to predict physical systems.

CHAPTER 1

Installation

Transformer-PhysX is currently a pure Python 3 module, meaning that installation should be universal across all platforms. Depending on your platform, manual installation of PyTorch may be beneficial to ensure cuda version is correct.

1.1 Install from PyPI (recommended)

```
pip install trphysx
```

PyPI homepage and previous versions.

1.2 Install from Source

```
git clone https://github.com/zabaras/transformer-physx.git

cd transformer-physx/

pip install -e .
```

If you want to change the module's source code or want the latest pushed commit.

1.3 Dependencies

For the most up-to-date list of dependencies, please see the setup.py. The general list is:

- torch >= 1.7.0
- filelock >= 3.0.0
- h5py >= 2.9.0

- numpy >= 1.15.0 (should already be installed with torch)
- matplotlib >= 3.0.0 (for visualizations)

1.4 Verify Installation

```
python -c 'import trphysx; print(trphysx.__version__)'
```

CHAPTER 2

Quick Start

The most reliable way to familiarize yourself with Transformer-PhysX package is through the Google Colab notebooks. Several examples (built in and custom) are provided to allow users to quickly start running transformers to model physical systems!

Since the framework requires two phases, training the embedding and the transformer model, separate Colab notebooks are provided for each.

2.1 Current Collab Notebooks

System	Training Embedding	Training Transformer	Built-in
Lorenz	Open	Open	True
Rossler	Open	Open	False
Cylinder	Open	Open	True
Gray-Scott	Not available	Coming Soon	True

2.2 Example Information

(coming soon)

CHAPTER 3

trphysx.config

3.1 trphysx.config.arg_parser

class trphysx.config.arg_parser.**DataClass**(**args*, ***kwargs*)
Bases: typing.Protocol

class trphysx.config.arg_parser.**HfArgumentParser**(*dataclass_types: Union[trphysx.config.arg_parser.DataClass, Iterable[trphysx.config.arg_parser.DataClass]], **kwargs*)
Bases: argparse.ArgumentParser

This subclass of *argparse.ArgumentParser* uses type hints on dataclasses to generate arguments. The class is designed to play well with the native argparse. In particular, you can add more (non-dataclass backed) arguments to the parser after initialization and you'll get the output back after parsing as an additional namespace.

Code originally from the Huggingface Transformers repository: https://github.com/huggingface/transformers/blob/master/src/transformers/hf_argparser.py

Parameters

- **dataclass_types** (*Union[DataClass, Iterable[DataClass]]*) – Dataclass type, or list of dataclass types for which we will "fill" instances with the parsed args.
- **kwargs** (*optional*) – Passed to *argparse.ArgumentParser()* in the regular way.

parse_args_into_dataclasses(*args: Iterable[str] = None*, *return_remaining_strings: bool = False*, *look_for_args_file: bool = True*, *args_filename: str = None*) → Tuple[trphysx.config.arg_parser.DataClass]
Parse command-line args into instances of the specified dataclass types.

Parameters

- **args** (*Iterable[str]*) – List of strings to parse. The default is taken from sys.argv. (same as argparse.ArgumentParser)
- **return_remaining_strings** (*bool*) – If true, also return a list of remaining argument strings.

- **look_for_args_file** (*bool*) – If true, will look for a ".args" file with the same base name as the entry point script for this process, and will append its potential content to the command line args.
- **args_filename** (*str*) – If not None, will uses this file instead of the ".args" file specified in the previous argument.

Returns

- the dataclass instances in the same order as they were passed to the initializer.abspath
- if applicable, an additional namespace for more (non-dataclass backed) arguments added to the parser after initialization.
- The potential list of remaining argument strings. (same as argparse.ArgumentParser.parse_known_args)

Return type (Tuple[*DataClass*])

3.2 trphysx.config.args

class trphysx.config.args.**ModelArguments**(*init_name: str = 'lorenz'*, *model_name: str = None*, *config_name: str = None*, *embedding_name: str = None*, *embedding_file_or_path: str = None*, *transformer_file_or_path: str = None*, *viz_name: str = None*)

Bases: object

Arguments pertaining to which model/config/tokenizer we are going to fine-tune, or train from scratch.

init_name = 'lorenz'

model_name = None

config_name = None

embedding_name = None

embedding_file_or_path = None

transformer_file_or_path = None

viz_name = None

class trphysx.config.args.**DataArguments**(*n_train: int = 2048*, *n_eval: int = 256*, *stride: int = 32*, *training_h5_file: str = None*, *eval_h5_file: str = None*, *overwrite_cache: bool = False*, *cache_path: str = None*)

Bases: object

Arguments pertaining to training and evaluation data.

n_train = 2048

n_eval = 256

stride = 32

training_h5_file = None

eval_h5_file = None

overwrite_cache = False

cache_path = None

class trphysx.config.args.**TrainingArguments**(*block_size: int = -1*, *exp_dir: str = None*, *ckpt_dir: str = None*, *plot_dir: str = None*, *save_steps: int = 25*, *eval_steps: int = 25*, *plot_max: int = 3*, *epoch_start: int = 0*, *epochs: int = 200*, *lr: float = 0.001*, *max_grad_norm: float = 0.1*, *dataloader_drop_last: bool = True*, *gradient_accumulation_steps: int = 1*, *train_batch_size: int = 256*, *eval_batch_size: int = 16*, *local_rank: int = -1*, *n_gpu: int = 1*, *seed: int = 12345*, *notes: str = None*)

Bases: object

Arguments pertaining to what data we are going to input our model for training and eval.

block_size = -1

exp_dir = None

ckpt_dir = None

plot_dir = None

save_steps = 25

eval_steps = 25

plot_max = 3

epoch_start = 0

epochs = 200

lr = 0.001

max_grad_norm = 0.1

dataloader_drop_last = True

gradient_accumulation_steps = 1

train_batch_size = 256

eval_batch_size = 16

local_rank = -1

n_gpu = 1

seed = 12345

notes = None

class trphysx.config.args.**ArgUtils**

Bases: object

Argument utility class for modifying particular arguments after initialization

classmethod config(*modelArgs: trphysx.config.args.ModelArguments*, *dataArgs: trphysx.config.args.DataArguments*, *trainingArgs: trphysx.config.args.TrainingArguments*, *create_paths: bool = True*) → Tuple[trphysx.config.args.ModelArguments, trphysx.config.args.DataArguments, trphysx.config.args.TrainingArguments]

Runs additional runtime configuration updates for argument instances

Parameters

- **modelArgs** (ModelArguments) – Transformer model arguments
- **dataArgs** (DataArguments) – Data loader/ data set arguments
- **trainingArgs** (TrainingArguments) – Training arguments
- **create_paths** (*bool, optional*) – Create training/testing folders. Defaults to True.

Returns Updated argument instances

Return type Tuple[*ModelArguments*, *DataArguments*, *TrainingArguments*]

classmethod configModelNames(*modelArgs: trphysx.config.args.ModelArguments*) → trphysx.config.args.ModelArguments

classmethod configPaths(*modelArgs: trphysx.config.args.ModelArguments*, *dataArgs: trphysx.config.args.DataArguments*, *trainingArgs: trphysx.config.args.TrainingArguments*) → Tuple[trphysx.config.args.ModelArguments, trphysx.config.args.DataArguments, trphysx.config.args.TrainingArguments]

Sets up various folder path parameters

Parameters

- **modelArgs** (ModelArguments) – Transformer model arguments
- **dataArgs** (DataArguments) – Data loader/ data set arguments
- **trainingArgs** (TrainingArguments) – Training arguments

Returns Updated argument instances

Return type Tuple[*ModelArguments*, *DataArguments*, *TrainingArguments*]

classmethod configTorchDevices(*args: trphysx.config.args.TrainingArguments*) → trphysx.config.args.TrainingArguments

Sets up device ids for training

Parameters **args** (TrainingArguments) – Training arguments

Returns Updated argument instance

Return type *TrainingArguments*

3.3 trphysx.config.configuration_auto

class trphysx.config.configuration_auto.**AutoPhysConfig**

Bases: object

Helper class for creating configurations for different built in examples

Raises **EnvironmentError** – If direct initialization of this class is attempted.

classmethod load_config(*model_name_or_path*, ***kwargs*) → trphysx.config.configuration_phys.PhysConfig

Creates a configuration object for a transformer model. Predefined configs currently support: “lorenz”, “cylinder”, “grayscott”

Parameters **model_name_or_path** (*str*) – Name of model or path to save config JSON file

Returns Configuration of transformer

Return type (*PhysConfig*)

classmethod from_json_file(*json_file: str*) → Dict[KT, VT]
Reads a json file and loads it into a dictionary.

Parameters **json_file** (*string*) – Path to the JSON file containing the parameters.

Returns Dictionary of parsed JSON

Return type Dict

3.4 trphysx.config.configuration_cylinder

class trphysx.config.configuration_cylinder.**CylinderConfig**(*n_ctx=16, n_embd=128, n_layer=6, n_head=4, state_dims=[3, 64, 128], activation_function='gelu_new', **kwargs*)

Bases: *trphysx.config.configuration_phys.PhysConfig*

This is the configuration class for the modeling of the flow around a cylinder system.

model_type = 'cylinder'

hidden_size

num_attention_heads

num_hidden_layers

3.5 trphysx.config.configuration_grayscott

class trphysx.config.configuration_grayscott.**GrayScottConfig**(*n_ctx=128, n_embd=512, n_layer=2, n_head=32, state_dims=[2, 32, 32, 32], activation_function='gelu_new', **kwargs*)

Bases: *trphysx.config.configuration_phys.PhysConfig*

This is the configuration class for the modeling of the Gray-scott system.

model_type = 'cylinder'

hidden_size

num_attention_heads

num_hidden_layers

3.6 trphysx.config.configuration_lorenz

class trphysx.config.configuration_lorenz.**LorenzConfig**(*n_ctx=64, n_embd=32, n_layer=4, n_head=4, state_dims=[3], activation_function='gelu_new', initializer_range=0.05, **kwargs*)

Bases: *trphysx.config.configuration_phys.PhysConfig*

This is the configuration class for the modeling of the Lorenz system.

model_type = 'lorenz'

hidden_size

num_attention_heads

num_hidden_layers

3.7 trphysx.config.configuration_phys

class trphysx.config.configuration_phys.**PhysConfig**(***kwargs*)

Bases: object

Parent class for physical transformer configuration. This is a slimmed version of the pretrainedconfig from the Hugging Face repository.

Parameters

- **n_ctx** (*int*) – Context window of transformer model.
- **n_embd** (*int*) – Dimensionality of the embeddings and hidden states.
- **n_layer** (*int*) – Number of hidden layers in the transformer.
- **n_head** (*int*) – Number of self-attention heads in each layer.
- **state_dims** (*List*) – List of physical state dimensionality. Used in embedding models.
- **activation_function** (*str, optional*) – Activation function. Defaults to "gelu_new".
- **resid_pdrop** (*float, optional*) – The dropout probability for all fully connected layers in the transformer. Defaults to 0.0.
- **embd_pdrop** (*float, optional*) – The dropout ratio for the embeddings. Defaults to 0.0.
- **attn_pdrop** (*float, optional*) – The dropout ratio for the multi-head attention. Defaults to 0.0.
- **layer_norm_epsilon** (*float, optional*) – The epsilon to use in the layer normalization layers. Defaults to 1e-5.
- **initializer_range** (*float, optional*) – The standard deviation for initializing all weight matrices. Defaults to 0.02.
- **output_hidden_states** (*bool, optional*) – Output embeddeding states from transformer. Defaults to False.

- **output_attentions** (*bool, optional*) – Output attention values from transformer. Defaults to False.
- **use_cache** (*bool, optional*) – Store transformers internal state for rapid predictions. Defaults to True.

Raises **AssertionError** – If provided parameter is not a config parameter

model_type = ''

save_pretrained(*save_directory: str*) → None
Save a configuration object to JSON file.

Parameters **save_directory** (*str*) – Directory where the configuration JSON file will be saved.

Raises **AssertionError** – If provided directory does not exist.

classmethod from_dict(*config_dict: Dict[str, any], **kwargs*) → transphysx.config.configuration_phys.PhysConfig
Constructs a config from a Python dictionary of parameters.

Parameters

- **config_dict** (*Dict[str, any]*) – Dictionary of parameters.
- **kwargs** (*Dict[str, any]*) – Additional parameters from which to initialize the configuration object.

Returns An instance of a configuration object

Return type (*PhysConfig*)

to_dict() → Dict[str, any]
Serializes this instance to a Python dictionary.

Returns Dictionary of config attributes

Return type (Dict[str, any])

to_json_string() → str
Serializes this instance to a JSON string.

Returns String of configuration instance in JSON format.

Return type (str)

to_json_file(*json_file_path: str*) → None
Save config instance to JSON file.

Parameters **json_file_path** (*str*) – Path to the JSON file in which this configuration instance's parameters will be saved.

update(*config_dict: Dict[KT, VT]*) → None
Updates attributes of this class with attributes from provided dictionary.

Parameters **config_dict** (*Dict*) – Dictionary of attributes that shall be updated for this class.

CHAPTER 4

trphysx.data_utils

4.1 trphysx.data_utils.data_utils

class trphysx.data_utils.data_utils.**DataCollator**
Bases: object

Data collator used for training datasets. Combines examples in a minibatch into one tensor.

Parameters

- **examples** (*List[Dict[str, Tensor]]*) – List of training examples. An example should be a dictionary of tensors from the dataset.
- **Returns** – Dict[str, Tensor]: Minibatch dictionary of combined example data tensors

4.2 trphysx.data_utils.dataset_auto

class trphysx.data_utils.dataset_auto.**AutoDataset**
Bases: object

Helper class for creating training data-sets for different numerical examples

Raises **EnvironmentError** – If direct initialization of this class is attempted.

classmethod **create_dataset**(*dataset_name: str*, **args*, ***kwargs*) → trphysx.data_utils.dataset_phys.PhysicalDataset
Creates a data-set for testing or validation Currently supports: “lorenz”, “cylinder”, “grayscott”

Parameters **dataset_name** (*str*) – Keyword/name of the data-set needed

Raises **KeyError** – If dataset_name is not a supported model type

Returns Initialized data-set

Return type (*PhysicalDataset*)

4.3 trphysx.data_utils.dataset_cylinder

class trphysx.data_utils.dataset_cylinder.**CylinderDataset**(*embedder: trphysx.embedding.embedding_model.Embedding*, *file_path: str*, *block_size: int*, *stride: int = 1*, *ndata: int = -1*, *eval: bool = False*, *overwrite_cache: bool = False*, *cache_path: str = None*, ***kwargs*)

Bases: *trphysx.data_utils.dataset_phys.PhysicalDataset*

Dataset for 2D flow around a cylinder numerical example

embed_data(*h5_file: h5py._hl.files.File*, *embedder: trphysx.embedding.embedding_model.EmbeddingModel*) → None

Embeds cylinder flow data into a 1D vector representation for the transformer.

Parameters

- **h5_file** (*h5py.File*) – HDF5 file object of raw data
- **embedder** (EmbeddingModel) – Embedding neural network

4.4 trphysx.data_utils.dataset_grayscott

class trphysx.data_utils.dataset_grayscott.**GrayscottDataset**(*embedder: trphysx.embedding.embedding_model.Embeddi*, *file_path: str*, *block_size: int*, *stride: int = 1*, *ndata: int = -1*, *eval: bool = False*, *overwrite_cache: bool = False*, *cache_path: str = None*, ***kwargs*)

Bases: *trphysx.data_utils.dataset_phys.PhysicalDataset*

Dataset class for the Gray-scott numerical example.

embed_data(*h5_file: h5py._hl.files.File*, *embedder: trphysx.embedding.embedding_model.EmbeddingModel*)

Embeds gray-scott data into a 1D vector representation for the transformer.

TODO: Clean up and remove custom positions

Parameters

- **h5_file** (*h5py.File*) – HDF5 file object of raw data
- **embedder** (EmbeddingModel) – Embedding neural network

class trphysx.data_utils.dataset_grayscott.**GrayscottPredictDataset**(*embedder: trphysx.embedding.embedding_mode file_path: str, block_size: int, neval: int = 16, overwrite_cache: bool = False, cache_path: str = None*)

Bases: *trphysx.data_utils.dataset_grayscott.GrayscottDataset*

Prediction data-set for the flow around a cylinder numerical example. Used during testing/validation since this data-set will store the embedding model and target states.

TODO: Remove this and have an overloaded trainer class for gray-scott

Parameters

- **embedder** (*trphysx.embedding.embedding_model.EmbeddingModel*) – Embedding neural network
- **file_path** (*str*) – Path to hdf5 raw data file
- **block_size** (*int*) – Length of time-series blocks for training
- **stride** (*int, optional*) – Stride interval to sample blocks from the raw time-series. Defaults to 1.
- **neval** (*int, optional*) – Number of time-series from the HDF5 file to use for testing. Defaults to 16.
- **overwrite_cache** (*bool, optional*) – Overwrite cache file if it exists, i.e. embeded the raw data from file. Defaults to False.
- **cache_path** (*str, optional*) – Path to save the cached embeddings at. Defaults to None.

recover(*x0*, *mb_size: int = 96*)

Recovers the physical state variables from an embedded vector

Parameters

- **x0** (*torch.Tensor*) – [B, config.n_embd] Time-series of embedded vectors
- **mb_size** (*int, optional*) – Mini-batch size for recovering the state variables

Returns [B, 2, H, W, D] physical state variable tensor

Return type (torch.Tensor)

4.5 trphysx.data_utils.dataset_lorenz

class trphysx.data_utils.dataset_lorenz.**LorenzDataset**(*embedder: trphysx.embedding.embedding_model.EmbeddingModel*, *file_path: str*, *block_size: int*, *stride: int = 1*, *ndata: int = -1*, *eval: bool = False*, *overwrite_cache: bool = False*, *cache_path: str = None*, ***kwargs*)

Bases: *trphysx.data_utils.dataset_phys.PhysicalDataset*

Dataset for the Lorenz numerical example

embed_data(*h5_file: h5py._hl.files.File*, *embedder: trphysx.embedding.embedding_model.EmbeddingModel*) → None

Embeds lorenz data into a 1D vector representation for the transformer.

Parameters

- **h5_file** (*h5py.File*) – HDF5 file object of raw data
- **embedder** (EmbeddingModel) – Embedding neural network

4.6 trphysx.data_utils.dataset_phys

class trphysx.data_utils.dataset_phys.**PhysicalDataset**(*embedder: trphysx.embedding.embedding_model.EmbeddingModel*, *file_path: str*, *block_size: int*, *stride: int = 1*, *ndata: int = -1*, *eval: bool = False*, *overwrite_cache: bool = False*, *cache_path: str = None*, ***kwargs*)

Bases: torch.utils.data.dataset.Dataset

Parent class for training and evaluation datasets for physical transformers. The caching of the dataset is based on the Hugging Face implementation.

Parameters

- **embedder** (EmbeddingModel) – Embedding neural network
- **file_path** (*str*) – Path to hdf5 raw data file
- **block_size** (*int*) – Length of time-series blocks for training
- **stride** (*int, optional*) – Stride interval to sample blocks from the raw time-series. Defaults to 1.
- **ndata** (*int, optional*) – Number of time-series from the HDF5 file to block. Will use all if negative. Defaults to -1.
- **eval** (*bool, optional*) – If this is a eval data-set, which will provide target states. Defaults to False.
- **overwrite_cache** (*bool, optional*) – Overwrite cache file if it exists, i.e. embed the raw data from file. Defaults to False.

- **cache_path** (*str, optional*) – Path to save the cached embeddings at. Defaults to None.

read_cache(*cached_features_file: str*) → None
Default method to read cache file into object.

Parameters **cached_features_file** (*str*) – Cache file path

write_cache(*cached_features_file: str*) → None
Default method to write cache file .

Parameters **cached_features_file** (*str*) – Cache file path

embed_data(*h5_file: h5py._hl.files.File*, *embedder: trphysx.embedding.embedding_model.EmbeddingModel*)
Embeds raw physical data into a 1D vector representation for the transformer. This is problem specific and thus must be overridden.

Parameters

- **h5_file** (*h5py.File*) – HDF5 file object to read raw data from
- **embedder** (EmbeddingModel) – Embedding neural network

Raises **NotImplementedError** – If function has not been overridden by a child dataset class.

CHAPTER 5

trphysx.embedding

5.1 Subpackages

5.1.1 trphysx.embedding.training

trphysx.embedding.training.enn_args

class trphysx.embedding.training.enn_args.**EmbeddingParser**
Bases: argparse.ArgumentParser

Arguments for training embedding models

mkdirs (**directories*) → None
Makes a directory if it does not exist

Parameters **directories** (*str...*) – a sequence of directories to create

Raises **OSError** – if directory cannot be created

parse (*args: List[T] = None*, *dirs: bool = True*) → None
Parse program arguments

Parameters

- **args** (*List, optional*) – Explicit list of arguments. Defaults to None.
- **dirs** (*bool, optional*) – Make experiment directories. Defaults to True.

trphysx.embedding.training.enn_data_handler

class trphysx.embedding.training.enn_data_handler.**EmbeddingDataHandler**
Bases: object

Base class for embedding data handlers. Data handlers are used to create the training and testing datasets.

mu = None

std = None

norm_params
Get normalization parameters

Raises **ValueError** – If normalization parameters have not been initialized

Returns mean and standard deviation

Return type (Tuple)

createTrainingLoader(**args*, ***kwargs*)

createTestingLoader(**args*, ***kwargs*)

class trphysx.embedding.training.enn_data_handler.**LorenzDataHandler**
Bases: *trphysx.embedding.training.enn_data_handler.EmbeddingDataHandler*

Built in embedding data handler for Lorenz system

class LorenzDataset(*examples: List[T]*)
Bases: torch.utils.data.dataset.Dataset

Dataset for training Lorenz embedding model.

Parameters **examples** (*List*) – list of training/testing examples

class LorenzDataCollator
Bases: object

Data collator for lorenz embedding problem

createTrainingLoader(*file_path: str*, *block_size: int*, *stride: int = 1*, *ndata: int = -1*, *batch_size: int = 32*, *shuffle: bool = True*) → torch.utils.data.dataloader.DataLoader
Creating training data loader for Lorenz system. For a single training simulation, the total time-series is sub-chunked into smaller blocks for training.

Parameters

- **file_path** (*str*) – Path to HDF5 file with training data
- **block_size** (*int*) – The length of time-series blocks
- **stride** (*int*) – Stride of each time-series block
- **ndata** (*int, optional*) – Number of training time-series. If negative, all of the provided
- **will be used. Defaults to -1.** (*data*) –
- **batch_size** (*int, optional*) – Training batch size. Defaults to 32.
- **shuffle** (*bool, optional*) – Turn on mini-batch shuffling in dataloader. Defaults to True.

Returns Training loader

Return type (DataLoader)

createTestingLoader(*file_path: str*, *block_size: int*, *ndata: int = -1*, *batch_size: int = 32*, *shuffle: bool = False*) → torch.utils.data.dataloader.DataLoader
Creating testing/validation data loader for Lorenz system. For a data case with time-steps [0,T], this method extract a smaller time-series to be used for testing [0, S], s.t. S < T.

Parameters

- **file_path** (*str*) – Path to HDF5 file with testing data

- **block_size** (*int*) – The length of testing time-series
- **ndata** (*int, optional*) – Number of testing time-series. If negative, all of the provided
- **will be used. Defaults to -1.** (*data*) –
- **batch_size** (*int, optional*) – Testing batch size. Defaults to 32.
- **shuffle** (*bool, optional*) – Turn on mini-batch shuffling in dataloader. Defaults to False.

Returns Testing/validation data loader

Return type (DataLoader)

class trphysx.embedding.training.enn_data_handler.**CylinderDataHandler**

Bases: *trphysx.embedding.training.enn_data_handler.EmbeddingDataHandler*

Built in embedding data handler for flow around a cylinder system

class CylinderDataset(*examples: List[T], visc: List[T]*)

Bases: torch.utils.data.dataset.Dataset

Dataset for training flow around a cylinder embedding model

Parameters

- **examples** (*List*) – list of training/testing example flow fields
- **visc** (*List*) – list of training/testing example viscosities

class CylinderDataCollator

Bases: object

Data collator for flow around a cylinder embedding problem

createTrainingLoader(*file_path: str*, *block_size: int*, *stride: int = 1*, *ndata: int = -1*, *batch_size: int = 32*, *shuffle: bool = True*) → torch.utils.data.dataloader.DataLoader

Creating training data loader for the flow around a cylinder system. For a single training simulation, the total time-series is sub-chunked into smaller blocks for training.

Parameters

- **file_path** (*str*) – Path to HDF5 file with training data
- **block_size** (*int*) – The length of time-series blocks
- **stride** (*int*) – Stride of each time-series block
- **ndata** (*int, optional*) – Number of training time-series. If negative, all of the provided
- **will be used. Defaults to -1.** (*data*) –
- **batch_size** (*int, optional*) – Training batch size. Defaults to 32.
- **shuffle** (*bool, optional*) – Turn on mini-batch shuffling in dataloader. Defaults to True.

Returns Training loader

Return type (DataLoader)

createTestingLoader(*file_path: str*, *block_size: int*, *ndata: int = -1*, *batch_size: int = 32*, *shuffle: bool = False*) → torch.utils.data.dataloader.DataLoader

Creating testing/validation data loader for the flow around a cylinder system. For a data case with time-steps [0,T], this method extract a smaller time-series to be used for testing [0, S], s.t. S < T.

Parameters

- **file_path** (*str*) – Path to HDF5 file with testing data
- **block_size** (*int*) – The length of testing time-series
- **ndata** (*int, optional*) – Number of testing time-series. If negative, all of the provided
- **will be used. Defaults to -1.** (*data*) –
- **batch_size** (*int, optional*) – Testing batch size. Defaults to 32.
- **shuffle** (*bool, optional*) – Turn on mini-batch shuffling in dataloader. Defaults to False.

Returns Testing/validation data loader

Return type (DataLoader)

class trphysx.embedding.training.enn_data_handler.**GrayScottDataHandler**

Bases: *trphysx.embedding.training.enn_data_handler.EmbeddingDataHandler*

Built in embedding data handler for the Gray-Scott system

class GrayScottDataset(*h5_file: str, keys: List[T], indices: List[T], block_size: int = 1*)

Bases: torch.utils.data.dataset.Dataset

Dataset for Gray-Scott system. Dynamically loads data from file each mini-batch since loading an entire data-set would be way too large. This data-set support the loading of sub-chunked time-series.

Parameters

- **h5_file** (*str*) – Path to hdf5 file with raw data
- **keys** (*List*) – List of keys corresponding to each example
- **indices** (*List*) – List of start indices for each time-series block
- **block_size** (*int, optional*) – List to time-series block sizes for each example. Defaults to 1.

class GrayScottDataCollator

Bases: object

Data collator for the Gray-scott embedding problem

createTrainingLoader(*file_path: str*, *block_size: int*, *stride: int = 1*, *ndata: int = -1*, *batch_size: int = 32*, *shuffle: bool = True*, *mpi_rank: int = -1*, *mpi_size: int = 1*) → torch.utils.data.dataloader.DataLoader

Creating training data loader for the Gray-Scott system. For a single training simulation, the total time-series is sub-chunked into smaller blocks for training. This particular dataloader support splitting the dataset between GPU processes for parallel training if needed.

Parameters

- **file_path** (*str*) – Path to HDF5 file with training data
- **block_size** (*int*) – The length of time-series blocks
- **stride** (*int*) – Stride of each time-series block
- **ndata** (*int, optional*) – Number of training time-series. If negative, all of the provided
- **will be used. Defaults to -1.** (*data*) –
- **batch_size** (*int, optional*) – Training batch size. Defaults to 32.

- **shuffle** (*bool, optional*) – Turn on mini-batch shuffling in dataloader. Defaults to True.
- **mpi_rank** (*int, optional*) – Rank of current MPI process. Defaults to -1.
- **mpi_size** (*int, optional*) – Number of training processes. Set to 1 for serial training. Defaults to 1.

Returns Training loader

Return type (DataLoader)

createTestingLoader(*file_path: str*, *block_size: int*, *ndata: int = -1*, *batch_size: int = 32*, *shuffle: bool = False*) → torch.utils.data.dataloader.DataLoader

Creating testing/validation data loader for the Gray-Scott system. For a data case with time-steps [0,T], this method extract a smaller time-series to be used for testing [0, S], s.t. S < T.

Parameters

- **file_path** (*str*) – Path to HDF5 file with testing data
- **block_size** (*int*) – The length of testing time-series
- **ndata** (*int, optional*) – Number of testing time-series. If negative, all of the provided
- **will be used. Defaults to -1.** (*data*) –
- **batch_size** (*int, optional*) – Testing batch size. Defaults to 32.
- **shuffle** (*bool, optional*) – Turn on mini-batch shuffling in dataloader. Defaults to False.

Returns Testing/validation data loader

Return type (DataLoader)

class trphysx.embedding.training.enn_data_handler.**AutoDataHandler**

Bases: object

Helper class for intializing different built in data-handlers for embedding training

classmethod load_data_handler(*model_name: str*, ***kwargs*) → trphysx.embedding.training.enn_data_handler.EmbeddingDataHandler

Gets built-in data handler. Currently supports: “lorenz”, “cylinder”, “grayscott”

Parameters **model_name** (*str*) – Model name

Raises **KeyError** – If model_name is not a supported model type

Returns Embedding data handler

Return type (*EmbeddingDataHandler*)

trphysx.embedding.training.enn_trainer

trphysx.embedding.training.enn_trainer.**set_seed**(*seed: int*) → None

Set random seed

Parameters **seed** (*int*) – random seed

class trphysx.embedding.training.enn_trainer.**EmbeddingTrainer**(*model: trphysx.embedding.embedding_model.Embe args: argparse.ArgumentParser, optimizers: Tuple[torch.optim.optimizer.Optimizer, torch.optim.lr_scheduler._LRScheduler], viz: trphysx.viz.viz_model.Viz = None*)

Bases: object

Trainer for Koopman embedding model

Parameters

- **model** (EmbeddingTrainingHead) – Embedding training model
- **args** (TrainingArguments) – Training arguments
- **optimizers** (*Tuple[Optimizer, Scheduler]*) – Tuple of Pytorch optimizer and lr scheduler.
- **viz** (Viz, *optional*) – Visualization class. Defaults to None.

train(*training_loader: torch.utils.data.dataloader.DataLoader*, *eval_dataloader: torch.utils.data.dataloader.DataLoader*) → None

Training loop for the embedding model

Parameters

- **training_loader** (*DataLoader*) – Training dataloader
- **eval_dataloader** (*DataLoader*) – Evaluation dataloader

evaluate(*eval_dataloader: torch.utils.data.dataloader.DataLoader*, *epoch: int = 0*) → Dict[str, float]

Run evaluation, plot prediction and return metrics.

Parameters

- **eval_dataset** (*Dataset*) – Evaluation dataloader
- **epoch** (*int, optional*) – Current epoch, used for naming figures. Defaults to 0.

Returns Dictionary of prediction metrics

Return type Dict[str, float]

5.2 trphysx.embedding.embedding_auto

class trphysx.embedding.embedding_auto.**AutoEmbeddingModel**

Bases: object

Helper class for initializing of loading various embedding models.

Raises **EnvironmentError** – If direct initialization of this class is attempted.

classmethod init_model(*model_name: str*, *config: trphysx.config.configuration_phys.PhysConfig*) → trphysx.embedding.embedding_model.EmbeddingModel

Initialize embedding model. Currently supports: “lorenz”, “cylinder”, “grayscott”

Parameters

- **model_name** (*str*) – Keyword/name of embedding model
- **config** (PhysConfig) – Transformer configuration class

Raises **ValueError** – If model_name is not a supported model type

Returns Initialized embedding model

Return type (*EmbeddingModel*)

classmethod init_trainer(*model_name: str*, *config: trphysx.config.configuration_phys.PhysConfig*) → trphysx.embedding.embedding_model.EmbeddingTrainingHead

Initialize embedding model with a training head. Currently supports: "lorenz", "cylinder", "grayscott"

Parameters

- **model_name** (*str*) – Keyword/name of embedding model
- **config** (PhysConfig) – Transformer configuration class

Raises **KeyError** – If model_name is not a supported trainer model types

Returns Initialized embedding model trainer

Return type (*EmbeddingTrainer*)

classmethod load_model(*model_name: str*, *config: trphysx.config.configuration_phys.PhysConfig*, *file_or_path_directory: Optional[str] = None*, *epoch: int = 0*) → trphysx.embedding.embedding_model.EmbeddingModel

Initialize and load embedding model from memory. Currently supports: "lorenz", "cylinder", "grayscott"

Parameters

- **model_name** (*str*) – Keyword/name of embedding model
- **config** (PhysConfig) – Transformer configuration class
- **file_or_path_directory** (*str, optional*) – embedding model file or directory path
- **epoch** (*int, optional*) – Epoch to load model from, only used if function is provided a directory

Raises **ValueError** – If model_name is not a supported model type

Returns Initialized embedding model with loaded weights

Return type (*EmbeddingModel*)

5.3 trphysx.embedding.embedding_cylinder

class trphysx.embedding.embedding_cylinder.**CylinderEmbedding**(*config: trphysx.config.configuration_phys.PhysConfig*)

Bases: *trphysx.embedding.embedding_model.EmbeddingModel*

Embedding Koopman model for the 2D flow around a cylinder system

Parameters **config** (PhysConfig) – Configuration class with transformer/embedding parameters

model_name = 'embedding_cylinder'

forward(*x: torch.Tensor*, *visc: torch.Tensor*) → Tuple[torch.Tensor]

Forward pass

Parameters

- **x** (*Tensor*) – [B, 3, H, W] Input feature tensor
- **visc** (*Tensor*) – [B] Viscosities of the fluid in the mini-batch

Returns

Tuple containing:

(Tensor): [B, config.n_embd] Koopman observables
(Tensor): [B, 3, H, W] Recovered feature tensor

Return type (TensorTuple)

embed(*x: torch.Tensor*, *visc: torch.Tensor*) → torch.Tensor
Embeds tensor of state variables to Koopman observables

Parameters

- **x** (*Tensor*) – [B, 3, H, W] Input feature tensor
- **visc** (*Tensor*) – [B] Viscosities of the fluid in the mini-batch

Returns [B, config.n_embd] Koopman observables

Return type (Tensor)

recover(*g: torch.Tensor*) → torch.Tensor
Recovers feature tensor from Koopman observables

Parameters **g** (*Tensor*) – [B, config.n_embd] Koopman observables

Returns [B, 3, H, W] Physical feature tensor

Return type (Tensor)

koopmanOperation(*g: torch.Tensor*, *visc: torch.Tensor*) → torch.Tensor
Applies the learned Koopman operator on the given observables

Parameters

- **g** (*Tensor*) – [B, config.n_embd] Koopman observables
- **visc** (*Tensor*) – [B] Viscosities of the fluid in the mini-batch

Returns [B, config.n_embd] Koopman observables at the next time-step

Return type Tensor

koopmanOperator
Current Koopman operator

Parameters **requires_grad** (*bool, optional*) – If to return with gradient storage. Defaults to True

Returns Full Koopman operator tensor

Return type Tensor

koopmanDiag

class trphysx.embedding.embedding_cylinder.**CylinderEmbeddingTrainer**(*config: trphysx.config.configuration_phys.*

Bases: *trphysx.embedding.embedding_model.EmbeddingTrainingHead*

Training head for the Lorenz embedding model

Parameters **config** (PhysConfig) – Configuration class with transformer/embedding parameters

forward(*states: torch.Tensor*, *viscosity: torch.Tensor*) → Tuple[float]
Trains model for a single epoch

Parameters

- **states** (*Tensor*) – [B, T, 3, H, W] Time-series feature tensor
- **viscosity** (*Tensor*) – [B] Viscosities of the fluid in the mini-batch

Returns

Tuple containing:

(float): Koopman based loss of current epoch
(float): Reconstruction loss

Return type FloatTuple

evaluate(*states: torch.Tensor*, *viscosity: torch.Tensor*) → Tuple[float, torch.Tensor, torch.Tensor]
Evaluates the embedding models reconstruction error and returns its predictions.

Parameters

- **states** (*Tensor*) – [B, T, 3, H, W] Time-series feature tensor
- **viscosity** (*Tensor*) – [B] Viscosities of the fluid in the mini-batch

Returns Test error, Predicted states, Target states

Return type Tuple[Float, Tensor, Tensor]

5.4 trphysx.embedding.embedding_grayscott

class trphysx.embedding.embedding_grayscott.**GrayScottEmbedding**(*config: trphysx.config.configuration_phys.PhysCo*

Bases: *trphysx.embedding.embedding_model.EmbeddingModel*

Embedding Koopman model for the 3D Gray-Scott system

Parameters **config** (PhysConfig) – Configuration class with transformer/embedding parameters

---

Note: For more information on the Gray-Scott model see "Complex Patterns in a Simple System" by John E. Pearson; https://doi.org/10.1126/science.261.5118.189

---

model_name = 'embedding_grayscott'

forward(*x: torch.Tensor*) → Tuple[torch.Tensor]
Forward pass

Parameters **x** (*Tensor*) – [B, 2, H, W, D] Input feature tensor

Returns

Tuple containing:

(Tensor): [B, config.n_embd] Koopman observables
(Tensor): [B, 2, H, W, D] Recovered feature tensor

Return type TensorTuple

embed(*x: torch.Tensor*) → torch.Tensor
Embeds tensor of state variables to Koopman observables

Parameters **x** (*Tensor*) – [B, 2, H, W, D] Input feature tensor

Returns [B, config.n_embd] Koopman observables

Return type Tensor

recover(*g: torch.Tensor*) → torch.Tensor
Recovers feature tensor from Koopman observables

Parameters **g** (*Tensor*) – [B, config.n_embd] Koopman observables

Returns [B, 2, H, W, D] Physical feature tensor

Return type (Tensor)

koopmanOperation(*g: torch.Tensor*) → torch.Tensor
Applies the learned Koopman operator on the given observables

Parameters **g** (*Tensor*) – [B, config.n_embd] Koopman observables

Returns [B, config.n_embd] Koopman observables at the next time-step

Return type (Tensor)

koopmanOperator
Current Koopman operator

Parameters **requires_grad** (*bool, optional*) – If to return with gradient storage. Defaults to True

Returns Full Koopman operator tensor

Return type Tensor

koopmanDiag

class trphysx.embedding.embedding_grayscott.**GrayScottEmbeddingTrainer**(*config: trphysx.config.configuration_ph*

Bases: *trphysx.embedding.embedding_model.EmbeddingTrainingHead*

Training head for the Gray-Scott embedding model

Parameters **config** (PhysConfig) – Configuration class with transformer/embedding parameters

forward(*states: torch.Tensor*) → Tuple[float]
Trains model for a single epoch

Parameters **states** (*Tensor*) – [B, T, 3, H, W] Time-series feature tensor

Returns

Tuple containing:

(float): Koopman based loss of current epoch
(float): Reconstruction loss

Return type FloatTuple

5.5 trphysx.embedding.embedding_lorenz

class trphysx.embedding.embedding_lorenz.**LorenzEmbedding**(*config: trphysx.config.configuration_phys.PhysConfig*)

Bases: *trphysx.embedding.embedding_model.EmbeddingModel*

Embedding Koopman model for the Lorenz ODE system

Parameters **config** (PhysConfig) – Configuration class with transformer/embedding parameters

model_name = 'embedding_lorenz'

forward(*x: torch.Tensor*) → Tuple[torch.Tensor]
Forward pass

Parameters **x** (*Tensor*) – [B, 3] Input feature tensor

Returns

Tuple containing:

(Tensor): [B, config.n_embd] Koopman observables
(Tensor): [B, 3] Recovered feature tensor

Return type TensorTuple

embed(*x: torch.Tensor*) → torch.Tensor
Embeds tensor of state variables to Koopman observables

Parameters **x** (*Tensor*) – [B, 3] Input feature tensor

Returns [B, config.n_embd] Koopman observables

Return type Tensor

recover(*g: torch.Tensor*) → torch.Tensor
Recovers feature tensor from Koopman observables

Parameters **g** (*Tensor*) – [B, config.n_embd] Koopman observables

Returns [B, 3] Physical feature tensor

Return type Tensor

koopmanOperation(*g: torch.Tensor*) → torch.Tensor
Applies the learned Koopman operator on the given observables

Parameters **g** (*Tensor*) – [B, config.n_embd] Koopman observables

Returns [B, config.n_embd] Koopman observables at the next time-step

Return type (Tensor)

koopmanOperator
Current Koopman operator

Parameters **requires_grad** (*bool, optional*) – If to return with gradient storage. Defaults to True

Returns Full Koopman operator tensor

Return type (Tensor)

koopmanDiag

class trphysx.embedding.embedding_lorenz.**LorenzEmbeddingTrainer**(*config: trphysx.config.configuration_phys.PhysC*

Bases: *trphysx.embedding.embedding_model.EmbeddingTrainingHead*

Training head for the Lorenz embedding model

Parameters **config** (PhysConfig) – Configuration class with transformer/embedding parameters

forward(*states: torch.Tensor*) → Tuple[float]
Trains model for a single epoch

Parameters **states** (*Tensor*) – [B, T, 3] Time-series feature tensor

Returns

Tuple containing:

(float): Koopman based loss of current epoch
(float): Reconstruction loss

Return type FloatTuple

evaluate(*states: torch.Tensor*) → Tuple[float, torch.Tensor, torch.Tensor]
Evaluates the embedding models reconstruction error and returns its predictions.

Parameters **states** (*Tensor*) – [B, T, 3] Time-series feature tensor

Returns Test error, Predicted states, Target states

Return type Tuple[Float, Tensor, Tensor]

5.6 trphysx.embedding.embedding_model

class trphysx.embedding.embedding_model.**EmbeddingModel**(*config: trphysx.config.configuration_phys.PhysConfig*)

Bases: torch.nn.modules.module.Module

Parent class for embedding models that handle the projection of the physical systems states into a vector representation

Parameters **config** (PhysConfig) – Configuration class with transformer/embedding parameters

model_name = 'embedding_model'

embed(*x*)

recover(*x*)

koopmanOperator

koopmanDiag

input_dims

embed_dims

num_parameters
Get number of learnable parameters in model

devices
Get list of unique device(s) model exists on

save_model (*save_directory: str*, *epoch: int = 0*) → None
Saves embedding model to the specified directory.

Parameters

- **save_directory** (*str*) – Folder directory to save state dictionary to.
- **epoch** (*int, optional*) – Epoch of current model for file name. Defaults to 0.

Raises **FileNotFoundError** – If provided path is a file

load_model (*file_or_path_directory: str*, *epoch: int = 0*) → None
Load a embedding model from the specified file or path

Parameters

- **file_or_path_directory** (*str*) – File or folder path to load state dictionary from.
- **epoch** (*int, optional*) – Epoch of current model for file name, used if folder path is provided. Defaults to 0.

Raises **FileNotFoundError** – If provided file or directory could not be found.

class trphysx.embedding.embedding_model.**EmbeddingTrainingHead**
Bases: torch.nn.modules.module.Module

Parent class for training head for embedding models

forward (**args*, ***kwargs*)
Defines 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.

---

evaluate (**args*, ***kwargs*)

save_model (**args*, ***kwargs*)
Saves the embedding model

load_model (**args*, ***kwargs*)
Load the embedding model

CHAPTER 6

trphysx.transformer

6.1 trphysx.transformer.attention

class trphysx.transformer.attention.**MaskedAttention**(*nx: int*, *n_ctx: int*, *config: trphysx.config.configuration_phys.PhysConfig*, *scale: bool = False*, *mask: str = 'tril'*)

Bases: torch.nn.modules.module.Module

Masked self-attention module based on the Hugging face implementation https://github.com/huggingface/transformers/blob/master/src/transformers/modeling_gpt2.py

Parameters

- **nx** (*int*) – Dimensionality of feature vector
- **n_ctx** (*int*) – Context length of the attention (TODO: Not needed with config object?)
- **config** (PhysConfig) – Transformer config object
- **scale** (*bool, optional*) – Scale the attention scores. Defaults to False.
- **mask** (*str, optional*) – Attention mask type. Defaults to 'tril'.

Raises **ValueError** – Invalid mask type

merge_heads(*x: torch.Tensor*) → torch.Tensor

Merge attention heads

Parameters **x** (*Tensor*) – [batch, head, seq_length, head_features] Input tensor

Returns [batch, seq_length, head * head_features] Concatenated output tensor

Return type Tensor

split_heads(*x*, *k: bool = False*) → torch.Tensor

Splits key, query or value tensor into separate heads. Dimensionality of output depends if tensor is a key.

Parameters

- **x** (*Tensor*) – [batch, seq_length, nx] Input tensor
- **k** (*bool*) – If input tensor is a key tensor

Returns [batch, head, seq_length, head_features] Split features for query and value, [batch, head, seq_length, head_features] split feature for key

Return type Tensor

forward(*x: torch.Tensor*, *layer_past: List[torch.Tensor] = None*, *attention_mask: torch.Tensor = None*, *head_mask: torch.Tensor = None*, *use_cache: bool = False*, *output_attentions: bool = False*) → List[torch.Tensor]

Masked attention forward pass

Parameters

- **x** (*Tensor*) – [batch, seq_length, nx] Input feature.
- **layer_past** (*Tensor, optional*) – [2, batch, n_head, seq_length, nx] Precomputed self-attention vectors. Defaults to None.
- **attention_mask** (*Tensor, optional*) – Optional defined attention mask. Applied before soft mask. Defaults to None.
- **head_mask** (*Tensor, optional*) – Optional attention value mask. Applied after softmax Defaults to None.
- **use_cache** (*bool, optional*) – Return calculated key values or faster generation. Defaults to False.
- **output_attentions** (*bool, optional*) – Return attention matrix. Defaults to False.

Returns Output consisting of output feature, key values (if requested), attention tensor (if requested)

Return type List[Tensor]

6.2 trphysx.transformer.generate_utils

class trphysx.transformer.generate_utils.**GenerationMixin**

Bases: object

Class containing generative functions for transformers

prepare_inputs_for_generation(*inputs_embeds: torch.Tensor*, *position_ids: torch.Tensor = None*, *prop_embeds: torch.Tensor = None*, ***kwargs*) → Dict[str, torch.Tensor]

Prepares input features for prediction

Parameters

- **inputs_features** (*Dict[str, Tensor]*) – Input feature tensors
- **are being generated.** (*that*) –

Returns Dictionary of model inputs

Return type Dict[str, Tensor]

generate(*inputs_embeds: torch.Tensor*, *position_ids: torch.Tensor = None*, *prop_embeds: torch.Tensor = None*, *max_length: int = None*, *attention_mask: torch.LongTensor = None*, *use_cache: bool = False*, ***model_specific_kwargs*) → Tuple[torch.Tensor]

Generated a predicted sequence of features

Parameters

- **inputs_embeds** (*Tensor*) – [batch, seq, n_embed] Input feature tensor
- **position_ids** (*Tensor, optional*) – [seq, n_embed] Position tensor. Defaults to None.
- **prop_embeds** (*Tensor, optional*) – [batch, seq, n_embed] Property tensor. Defaults to None.
- **max_length** (*int, optional*) – Length of time series to predict. Defaults to None.
- **attention_mask** (*LongTensor, optional*) – Manual attention mask. Defaults to None.
- **use_cache** (*bool, optional*) – Cache past transformer states for faster generation. Defaults to False.

Returns [batch, max_length, n_embed] Predicted feature tensor, additional optional transformer outputs.

Return type Tuple[Tensor]

6.3 trphysx.transformer.phys_transformer_base

class trphysx.transformer.phys_transformer_base.**PhysformerBase**(*config*, **inputs*, ***kwargs*)

Bases: torch.nn.modules.module.Module

Parent class for physical transformers

model_name = 'transformer_model'

forward()

Defines 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.

---

generate()

get_input_embeddings()

set_input_embeddings(*new_embeddings*)

tie_weights()

Tie the weights between the input embeddings and the output embeddings. If the *torchscript* flag is set in the configuration, can't handle parameter sharing so we are cloning the weights instead.

save_model(*save_directory: str*, *epoch: int = 0*) → None

Saves transformer model to the specified directory.

Parameters

- **save_directory** (*str*) – Folder to save file at
- **epoch** (*int, optional*) – Epoch number to name model file. Defaults to 0.

Raises **AssertionError** – If provided directory is not valid.

load_model (*file_or_path_directory: str*, *epoch: int = 0*) → None
Load a transformer model from the specified file or path

Parameters

- **file_or_path_directory** (*str*) – File or folder path to load state dictionary from.
- **epoch** (*int, optional*) – Epoch of current model for file name, used if folder path is provided. Defaults to 0.

Raises **FileNotFoundError** – If provided file or directory could not be found.

6.4 trphysx.transformer.phys_transformer_gpt2

class trphysx.transformer.phys_transformer_gpt2.**MLP**(*n_state: int*, *config: trphysx.config.configuration_phys.PhysConfig*)

Bases: torch.nn.modules.module.Module

Simple fully connected neural network layer. Includes activations function and dropout.

Parameters

- **n_state** (*int*) – dimensionality of input features
- **config** (PhysConfig) – Phys-transformer config object

forward(*x: torch.Tensor*) → torch.Tensor
Forward pass

Parameters **x** (*Tensor*) – [B, T, n_state] input features

Returns Output features

Return type Tensor

class trphysx.transformer.phys_transformer_gpt2.**Block**(*n_ctx: int*, *config: trphysx.config.configuration_phys.PhysConfig*, *scale: bool = False*)

Bases: torch.nn.modules.module.Module

Transformer decoder block consisting of layer norm, masked self-attention, layer norm and fully connected layer.

Parameters

- **n_ctx** (*int*) – contex length of block
- **config** (PhysConfig) – Phys-transformer config object
- **scale** (*bool, optional*) – Scaled self-attention calculation. Defaults to False.

forward(*x: torch.Tensor*, *layer_past: List[torch.Tensor] = None*, *attention_mask: torch.LongTensor = None*, *head_mask: torch.LongTensor = None*, *use_cache: bool = False*, *output_attentions: bool = False*) → List[torch.Tensor]
Forward pass

Parameters

- **x** (*Tensor*) – [B, T, n_state] input features

- **layer_past** (*[type], optional*) – Past self-attention calculation. Defaults to None.
- **attention_mask** (*LongTensor, optional*) – Attention mask. Defaults to None.
- **head_mask** (*LongTensor, optional*) – Attention value. Defaults to None.
- **use_cache** (*bool, optional*) – Store attention state (key values). Defaults to False.
- **output_attentions** (*bool, optional*) – Return attention values. Defaults to False.

Returns List of output tensors

Return type List[Tensor]

class trphysx.transformer.phys_transformer_gpt2.**PhysformerGPT2**(*config: trphysx.config.configuration_phys.PhysCo model_name: str = None*)

Bases: *trphysx.transformer.generate_utils.GenerationMixin*, *trphysx.transformer.phys_transformer_base.PhysformerBase*

Transformer decoder model for modeling physics

Parameters

- **config** (PhysConfig) – Phys-transformer config object
- **model_name** (*str, optional*) – Model name. Defaults to None.

forward(*inputs_embeds: torch.Tensor*, *position_ids: torch.Tensor = None*, *prop_embeds: torch.Tensor = None*, *past: List[List[torch.Tensor]] = None*, *attention_mask: torch.LongTensor = None*, *head_mask: torch.LongTensor = None*, *use_cache: bool = True*, *output_attentions: bool = False*) → List[torch.Tensor]

Forward pass

Note: Attention masks are not properly implemented presently and will likely not work.

Parameters

- **inputs_embeds** (*Tensor*) – [B, T, n_embed] Input features
- **position_ids** (*Tensor, optional*) – [T, n_embed] Manually specify position ids. Defaults to None.
- **prop_embeds** (*Tensor, optional*) – [B, T, n_embed] Optional property feature. Defaults to None.
- **past** (*List[List[Tensor]], optional*) – Transformer past state. Defaults to None.
- **attention_mask** (*LongTensor, optional*) – [B, T] Sequence attention mask. Defaults to None.
- **head_mask** (*LongTensor, optional*) – Attention value mask. Defaults to None.
- **use_cache** (*bool, optional*) – Return attention states (keys). Defaults to True.
- **output_attentions** (*bool, optional*) – Return attention scores. Defaults to False.

Returns Output features, attention state (if requested), hidden states of all layers (if requested), attention tensor (if requested)

Return type List[Tensor]

6.5 trphysx.transformer.phys_transformer_helpers

class trphysx.transformer.phys_transformer_helpers.**PhysformerTrain**(*config: trphysx.config.configuration_phys.Ph*, *transformer_model: trphysx.transformer.phys_transforme* = *None*)

Bases: *trphysx.transformer.phys_transformer_base.PhysformerBase*

Model head for training the physics transformer base.

Parameters

- **config** (PhysConfig) – Phys-transformer config object
- **transformer_model** (PhysformerBase) – Initialized transformer model

forward(*inputs_embeds: torch.Tensor*, *labels_embeds: torch.Tensor*, ***kwargs*) → Tuple[Union[float, torch.Tensor]]

Forward method for this head calculates the MSE between the predicted time-series and target embeddings This head allows for easy distribution to multiple GPUs and CPUs. See transformer

Parameters

- **inputs_embeds** (*Tensor*) – [B, T, n_embed] Input features
- **labels_embeds** (*Tensor*) – [B, T, n_embed] Target output features
- ****kwargs** (*optional*) – Additional tensformer forward pass arguments

Returns mse loss, last hidden state, (present attention state), (all hidden_states), (attention scores)

Return type Tuple[Union[float, Tensor]]

evaluate(*inputs_embeds: torch.Tensor*, *labels_embeds: torch.Tensor*, ***kwargs*) → Tuple[Union[float, torch.Tensor]]

Generate a time-series prediction using the transformer and calc MSE error.

Parameters

- **inputs_embeds** (*Tensor*) – [B, 1, n_embed] Starting input feature(s)
- **labels_embeds** (*Tensor*) – [B, T, n_embed] Target output features
- ****kwargs** (*optional*) – Additional tensformer forward pass arguments

Returns mse loss, last hidden state, (present attention state), (all hidden_states), (attention scores)

Return type Tuple[Union[float, Tensor]]

generate(**args*, ***kwargs*)

Generate call is just the forward call of the transformer

save_model(**args*, ***kwargs*)

Saves physformer model

load_model(**args*, ***kwargs*)

Load a physformer model

6.6 trphysx.transformer.utils

class trphysx.transformer.utils.**Conv1D**(*nf: int*, *nx: int*)
Bases: torch.nn.modules.module.Module

1D-convolutional layer (eqv to FCN) as defined by Radford et al. for OpenAI GPT (and also used in GPT-2). Basically works like a linear layer but the weights are transposed.

Note: Code adopted from: https://github.com/huggingface/transformers/blob/master/src/transformers/modeling_utils.py

---

Parameters

- **nf** (*int*) – The number of output features.
- **nx** (*int*) – The number of input features.

forward(*x: torch.Tensor*) → torch.Tensor
Forward pass

Parameters **x** (*Tensor*) – [..., nx] input features

Returns [..., nf] output features

Return type Tensor

trphysx.transformer.utils.**gelu_new**(*x: torch.Tensor*) → torch.Tensor
Implementation of the GELU activation function currently in Google BERT repo (identical to OpenAI GPT).

trphysx.transformer.utils.**gelu_fast**(*x*)
Faster approximate form of GELU activation function

trphysx.transformer.utils.**mish**(*x: torch.Tensor*) → torch.Tensor
Mish activation function

trphysx.transformer.utils.**linear_act**(*x: torch.Tensor*) → torch.Tensor
Linear activate function

trphysx.transformer.utils.**get_activation**(*activation_string: str*) → Callable
Gets a activation function

Parameters **activation_string** (*str*) – Name of activate function

Raises **KeyError** – Not a valid activation function

Returns activate function

Return type Callable

CHAPTER 7

trphysx.utils

7.1 trphysx.utils.metrics

class trphysx.utils.metrics.**Metrics**(*file_path: str = '.', file_name: str = 'log_metrics.h5'*)

Bases: object

Data class for storing training errors

Parameters

- **file_path** (*str, optional*) – Path to write logging files
- **file_name** (*str, optional*) – Log file name

file_path = '.'

file_name = 'log_metrics.h5'

push(***kwargs*) → None

Pushes elements in kwargs into the attributes of this class

Parameters ****kwargs** – Attributes to save

writeToHDF5(*file_name: str = None*) → None

Write the classes attributes to HDF5 file

Parameters **file_name** (*str, optional*) – File name to write to

appendToHDF5(*file_name: str = None*) → None

Appends the classes attributes to HDF5 file

Parameters **file_name** (*str, optional*) – File name to write to

delHDF5(*file_name: str = None*) → None

Deletes hdf5 file if it exists

Parameters **file_name** (*str, optional*) – File name

7.2 trphysx.utils.trainer

trphysx.utils.trainer.**set_seed**(*seed: int*) → None
Set random seed

Parameters **seed** (*int*) – random seed

class trphysx.utils.trainer.**Trainer**(*model: trphysx.transformer.phys_transformer_helpers.PhysformerTrain, args: trphysx.config.args.TrainingArguments, optimizers: Tuple[torch.optim.optimizer.Optimizer, torch.optim.lr_scheduler._LRScheduler], train_dataset: torch.utils.data.dataset.Dataset = None, eval_dataset: torch.utils.data.dataset.Dataset = None, embedding_model: trphysx.embedding.embedding_model.EmbeddingModel = None, viz: trphysx.viz.viz_model.Viz = None*)
Bases: object

Generalized trainer for physics transformer models

Parameters

- **model** (PhysformerTrain) – Transformer with training head
- **args** (TrainingArguments) – Training arguments
- **optimizers** (*Tuple[Optimizer, Scheduler], optional*) – Tuple of Pytorch optimizer and lr scheduler.
- **train_dataset** (*Dataset, optional*) – Training dataset. Defaults to None.
- **eval_dataset** (*Dataset, optional*) – Eval/Validation dataset. Defaults to None.
- **embedding_model** (EmbeddingModel, *optional*) – Embedding model. Used for recovering states during state evaluation of the model. Defaults to None.
- **viz** (Viz, *optional*) – Visualization class. Defaults to None.

get_train_dataloader(*train_dataset: torch.utils.data.dataset.Dataset = None*) → torch.utils.data.dataloader.DataLoader
Creates a training dataloader. Overload for unusual training cases.

Parameters **train_dataset** (*Dataset, optional*) – Optional training dataset. If none is provided, the class training data will be used. Defaults to None.

Raises **ValueError** – If both the dataset parameter and class dataset have not been provided

Returns Training dataloader

Return type DataLoader

get_eval_dataloader(*eval_dataset: torch.utils.data.dataset.Dataset = None*) → torch.utils.data.dataloader.DataLoader
Creates a evaluation dataloader used for validation or testing of model.

Parameters **eval_dataset** (*Dataset, optional*) – Optional eval dataset. If none is provided, the class eval data will be used. Defaults to None.

Raises **ValueError** – If both the dataset parameter and class dataset have not been provided

Returns Evaluation dataloader

Return type DataLoader

train() → None
Trains the transformer model

training_step(*model: trphysx.transformer.phys_transformer_helpers.PhysformerTrain, inputs: Dict[str, Any]*) → Tuple[float, torch.Tensor, torch.Tensor]
Calls a forward pass of the training model and backprops for a single time-step

Parameters

- **model** (PhysformerTrain) – Transformer model with training head, could be
- **inputs** (*Dict[str, Any]*) – Dictionary of model inputs for forward pass

Returns

Tuple containing: loss value, hidden states of transformer, attention states of the transformer.

Return type Tuple[float, Tensor, Tensor]

evaluate(*epoch: int = None*) → Dict[str, float]
Run evaluation and return metrics.

Parameters **epoch** (*int, optional*) – Current epoch, used for naming figures. Defaults to None.

Returns Dictionary of prediction metrics

Return type Dict[str, float]

eval_step(*model: trphysx.transformer.phys_transformer_helpers.PhysformerTrain, inputs: Dict[str, Any]*) → Tuple[float, torch.Tensor, torch.Tensor]
Calls a eval pass of the training model.

Parameters

- **model** (PhysformerTrain) – Transformer model with training head
- **inputs** (*Dict[str, Any]*) – Dictionary of model inputs for forward pass

Returns

Tuple containing: prediction error value, time-step error, predicted embeddings.

Return type Tuple[float, Tensor, Tensor]

eval_states(*pred_embeds: torch.Tensor*, *states: Any*, *epoch: int = None*, *plot_id: int = 0*, *plot: bool = True*) → float
Evaluates the predicted states by recovering the state space from the predicted embedding vectors. Can be overloaded for cases with special methods for recovering the state field.

Parameters

- **pred_embeds** (*Tensor*) – [B, T, n_embed] Predicted embedded vectors
- **states** (*Any*) – Target states / data for recovery
- **epoch** (*int, optional*) – Current epoch, used for naming figures. Defaults to None.
- **plot_id** (*int, optional*) – Secondary plotting id to distinguish between numerical cases. Defaults to 0.
- **plot** (*bool, optional*) – Plot models states. Defaults to True.

Returns Predicted state MSE error

Return type float

CHAPTER 8

trphysx.viz

8.1 trphysx.viz.viz_auto

class trphysx.viz.viz_auto.**AutoViz**
Bases: object

Helper class for initializing visualization classes.

Raises **EnvironmentError** – If direct initialization of this class is attempted.

classmethod load_viz(*viz_name: str*, **args*, ***kwargs*) → trphysx.viz.viz_model.Viz
Loads built in visualization class. Currently supports: “lorenz”, “cylinder”, “grayscott”

Parameters **viz_name** (*str*) – Keyword/name of visualization class

Raises **KeyError** – If viz_name is not a supported visualization type

Returns Initialized viz class

Return type (*Viz*)

8.2 trphysx.viz.viz_cylinder

class trphysx.viz.viz_cylinder.**CylinderViz**(*plot_dir: str = None*)
Bases: *trphysx.viz.viz_model.Viz*

Visualization class for flow around a cylinder

Parameters **plot_dir** (*str, optional*) – Directory to save visualizations in. Defaults to None.

plotPrediction(*y_pred: torch.Tensor*, *y_target: torch.Tensor*, *plot_dir: str = None*, *epoch: int = None*, *pid: int = 0*, *nsteps: int = 10*, *stride: int = 20*) → None
Plots the predicted x-velocity, y-velocity and pressure field contours

Parameters

- **y_pred** (*Tensor*) – [T, 3, H, W] Prediction tensor.
- **y_target** (*Tensor*) – [T, 3, H, W] Target tensor.
- **plot_dir** (*str, optional*) – Directory to save figure, overrides plot_dir one if provided. Defaults to None.
- **epoch** (*int, optional*) – Current epoch, used for file name. Defaults to None.
- **pid** (*int, optional*) – Optional plotting id for indexing file name manually. Defaults to 0.
- **nsteps** (*int, optional*) – Number of timesteps to plot. Defaults to 10.
- **stride** (*int, optional*) – Number of timesteps in between plots. Defaults to 10.

plotPredictionVorticity(*y_pred: torch.Tensor*, *y_target: torch.Tensor*, *plot_dir: str = None*, *epoch: int = None*, *pid: int = 0*, *nsteps: int = 10*, *stride: int = 10*) → None

Plots vorticity contours of flow around a cylinder at several time-steps. Vorticity gradients are calculated using standard smoothed central finite difference.

Parameters

- **y_pred** (*Tensor*) – [T, 3, H, W] Prediction tensor.
- **y_target** (*Tensor*) – [T, 3, H, W] Target tensor.
- **plot_dir** (*str, optional*) – Directory to save figure, overrides class plot_dir if provided. Defaults to None.
- **epoch** (*int, optional*) – Current epoch, used for file name. Defaults to None.
- **pid** (*int, optional*) – Optional plotting id for indexing file name manually. Defaults to 0.
- **nsteps** (*int, optional*) – Number of timesteps to plot. Defaults to 10.
- **stride** (*int, optional*) – Number of timesteps in between plots. Defaults to 5.

plotEmbeddingPrediction(*y_pred: torch.Tensor*, *y_target: torch.Tensor*, *plot_dir: str = None*, *epoch: int = None*, *bidx: int = None*, *tidx: int = None*, *pid: int = 0*) → None

Plots the predicted x-velocity, y-velocity and pressure field contours

Parameters

- **y_pred** (*Tensor*) – [B, T, 3, H, W] Prediction tensor.
- **y_target** (*Tensor*) – [B, T, 3, H, W] Target tensor.
- **plot_dir** (*str, optional*) – Directory to save figure, overrides plot_dir one if provided. Defaults to None.
- **epoch** (*int, optional*) – Current epoch, used for file name. Defaults to None.
- **bidx** (*int, optional*) – Batch index to plot. Defaults to None (plot random example in batch).
- **tidx** (*int, optional*) – Timestep index to plot. Defaults to None (plot random timestep).
- **pid** (*int, optional*) – Optional plotting id for indexing file name manually. Defaults to 0.

8.3 trphysx.viz.viz_grayscott

class trphysx.viz.viz_grayscott.**GrayScottViz**(*plot_dir: str = None*)
Bases: *trphysx.viz.viz_model.Viz*

Visualization class for the 3D Gray-scott system.

Parameters **plot_dir** (*str, optional*) – Directory to save visualizations in. Defaults to None.

plotPrediction(*y_pred: torch.Tensor*, *y_target: torch.Tensor*, *plot_dir: str = None*, *epoch: int = None*, *pid: int = 0*, *nsteps: int = 10*, *stride: int = 5*) → None
Plots z-slice of Gray-Scott prediction along the z-axis and saves to file

Parameters

- **y_pred** (*torch.Tensor*) – [T, 2, H, W, D] prediction time-series of states
- **y_target** (*torch.Tensor*) – [T, 2, H, W, D] target time-series of states
- **plot_dir** (*str, optional*) – Directory to save figure, overrides class plot_dir if provided. Defaults to None.
- **epoch** (*int, optional*) – Current epoch, used for file name. Defaults to None.
- **pid** (*int, optional*) – Optional plotting id for indexing file name manually. Defaults to 0.
- **nsteps** (*int, optional*) – Number of timesteps to plot. Defaults to 10.
- **stride** (*int, optional*) – Number of timesteps in between plots. Defaults to 5.

8.4 trphysx.viz.viz_lorenz

class trphysx.viz.viz_lorenz.**HandlerColormap**(*cmap: <module 'matplotlib.cm' from '/home/docs/checkouts/readthedocs.org/user_builds/transformer-physx/envs/latest/lib/python3.8/site-packages/matplotlib/cm.py'>*, *num_stripes: int = 8*, ***kw*)
Bases: matplotlib.legend_handler.HandlerBase

Class for creating colormap legend rectangles

Parameters

- **cmap** (*matplotlib.cm*) – Matplotlib colormap
- **num_stripes** (*int*) – Number of countour levels (strips) in rectangle

create_artists(*legend*, *orig_handle*, *xdescent*, *ydescent*, *width*, *height*, *fontsize*, *trans*)

class trphysx.viz.viz_lorenz.**LorenzViz**(*plot_dir: str = None*)
Bases: *trphysx.viz.viz_model.Viz*

Visualization class for Lorenz ODE

Parameters **plot_dir** (*str, optional*) – Directory to save visualizations in. Defaults to None.

plotPrediction(*y_pred: torch.Tensor*, *y_target: torch.Tensor*, *plot_dir: str = None*, *epoch: int = None*, *pid: int = 0*) → None
Plots a 3D line of a single Lorenz prediction

Parameters

- **y_pred** (*Tensor*) – [T, 3] Prediction tensor.
- **y_target** (*Tensor*) – [T, 3] Target tensor.
- **plot_dir** (*str, optional*) – Directory to save figure, overrides plot_dir one if provided. Defaults to None.
- **epoch** (*int, optional*) – Current epoch, used for file name. Defaults to None.
- **pid** (*int, optional*) – Optional plotting id for indexing file name manually. Defaults to 0.

plotMultiPrediction(*y_pred: torch.Tensor*, *y_target: torch.Tensor*, *plot_dir: str = None*, *epoch: int = None*, *pid: int = 0*, *nplots: int = 2*) → None

Plots the 3D lines of multiple Lorenz predictions

Parameters

- **y_pred** (*Tensor*) – [B, T, 3] Prediction tensor.
- **y_target** (*Tensor*) – [B, T, 3] Target tensor.
- **plot_dir** (*str, optional*) – Directory to save figure, overrides plot_dir one if provided. Defaults to None.
- **epoch** (*int, optional*) – Current epoch, used for file name. Defaults to None.
- **pid** (*int, optional*) – Optional plotting id for indexing file name manually. Defaults to 0.
- **nplots** (*int, optional*) – Number of cases to plot. Defaults to 2.

plotPredictionScatter(*y_pred: torch.Tensor*, *plot_dir: str = None*, *epoch: int = None*, *pid: int = 0*) → None

Plots scatter plots of a Lorenz prediction contoured based on distance from the basins. This will only contour correctly for the parameters s=10, r=28, b=2.667

Parameters

- **y_pred** (*Tensor*) – [T, 3] Prediction tensor.
- **plot_dir** (*str, optional*) – Directory to save figure, overrides plot_dir one if provided. Defaults to None.
- **epoch** (*int, optional*) – Current epoch, used for file name. Defaults to None.
- **pid** (*int, optional*) – Optional plotting id for indexing file name manually. Defaults to 0.

8.5 trphysx.viz.viz_model

class trphysx.viz.viz_model.**Viz**(*plot_dir: str = None*)

Bases: object

Parent class for visualization

Parameters **plot_dir** (*str, optional*) – Directory to save visualizations in. Defaults to None.

plotPrediction(*y_pred: torch.Tensor*, *y_target: torch.Tensor*, *plot_dir: str = None*, ***kwargs*) → None

Plots model prediction and target values

Parameters

- **y_pred** (*Tensor*) – prediction tensor
- **y_target** (*Tensor*) – target tensor
- **plot_dir** (*str, optional*) – Directory to save plot at. Defaults to None.
- ****kwargs** – Additional keyword arguments.

Raises **NotImplementedError** – If function has not been overridden by a child dataset class.

plotEmbeddingPrediction(*y_pred: torch.Tensor*, *y_target: torch.Tensor*, *plot_dir: str = None*, ***kwargs*) → None

Plots model prediction and target values during the embedding training

Parameters

- **y_pred** (*Tensor*) – mini-batch of prediction tensor
- **y_target** (*Tensor*) – mini-batch target tensor
- **plot_dir** (*str, optional*) – Directory to save plot at. Defaults to None.
- ****kwargs** – Additional keyword arguments.

Raises **NotImplementedError** – If function has not been overridden by a child dataset class.

saveFigure(*plot_dir: str = None*, *file_name: str = 'plot'*, *savepng: bool = True*, *savepdf: bool = False*) → None

Saves active matplotlib figure to file

Parameters

- **plot_dir** (*str, optional*) – Directory to save plot at, will use class plot_dir if none provided. Defaults to None.
- **file_name** (*str, optional*) – File name of the saved figure. Defaults to 'plot'.
- **savepng** (*bool, optional*) – Save figure in png format. Defaults to True.
- **savepdf** (*bool, optional*) – Save figure in pdf format. Defaults to False.

CHAPTER 9

Contact

Have a question, issue or found a bug? Best way to get into contact is through an issue on our github page!

CHAPTER 10

License

MIT License

Copyright (c) 2020 Nicholas Geneva

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

CHAPTER 11

Indices and Tables

- genindex
- modindex
- search

Python Module Index

t

Index

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W