EvoAug

Release 0.1

KooLab

Mar 24, 2023

CONTENTS

EvoAug is a Python library to train PyTorch models on regulatory genomics data with evolution-inspired data augmentations.

Check out the *User Guide* for further information, including how to install the project.

Note: This project is under active development. Source code can be found here.

CHAPTER

ONE

CONTENTS

1.1 User Guide

1.1.1 Installation

To use EvoAug, first install it using pip:

```
pip install evoaug
```

1.1.2 Example

Import evoaug:

```
from evoaug import evoaug, augment
import pytorch_lightning as pl
```

Define PyTorch model and modeling choices:

```
model = "DEFINE PYTORCH MODEL"
loss = "DEFINE PYTORCH LOSS"
optimizer_dict = "DEFINE OPTIMIZER OR OPTIMIZER DICT"
ckpt_aug_path = "path-to-aug-checkpoint.ckpt"
ckpt_finetune_path = "path-to-finetune-checkpoint.ckpt"
```

Train model with augmentations:

```
augment_list = [
   augment.RandomDeletion(delete_min=0, delete_max=20),
   augment.RandomRC(rc_prob=0.5),
   augment.RandomInsertion(insert_min=0, insert_max=20),
   augment.RandomTranslocation(shift_min=0, shift_max=20),
   augment.RandomMutation(mut_frac=0.05),
   augment.RandomNoise(noise_mean=0, noise_std=0.2),
]

robust_model = evoaug.RobustModel(
   model,
   criterion=loss,
   optimizer=optimizer_dict,
```

(continues on next page)

(continued from previous page)

```
    augment_list=augment_list,
    max_augs_per_seq=2,  # maximum number of augmentations per sequence
    hard_aug=True,  # use max_augs_per_seq, otherwise sample randomly up to max
    inference_aug=False,  # if true, keep augmentations on during inference time
)

# set up callback
callback_topmodel = pl.callbacks.ModelCheckpoint(
    monitor="val_loss", save_top_k=1, dirpath=output_dir, filename=ckpt_aug_path
)

# train model
trainer = pl.Trainer(
    gpus=1,
    max_epochs=100,
    auto_select_gpus=True,
    logger=None,
    callbacks=["ADD CALLBACKS", "callback_topmodel"],
)

# pre-train model with augmentations
trainer.fit(robust_model, datamodule=data_module)

# load best model
robust_model = evoaug.load_model_from_checkpoint(robust_model, ckpt_aug_path)
```

Fine-tune model without augmentations:

```
# set up fine-tuning
robust_model.finetune = True
robust_model.optimizer = "set up optimizer for fine-tuning"

# set up callback
callback_topmodel = pl.callbacks.ModelCheckpoint(
    monitor="val_loss",
    save_top_k=1,
    dirpath=output_dir,
    filename=ckpt_finetune_path,
)

# set up pytorch lightning trainer
trainer = pl.Trainer(
    gpus=1,
    max_epochs=100,
    auto_select_gpus=True,
    logger=None,
    callbacks=["ADD CALLBACKS", "callback_topmodel"],
)

# fine-tune model
trainer.fit(robust_model, datamodule=data_module)
```

(continues on next page)

(continued from previous page)

```
# load best fine-tuned model
robust_model = evoaug.load_model_from_checkpoint(robust_model, ckpt_finetune_path)
```

1.1.3 Examples on Google Colab

DeepSTARR analysis:

```
https://colab.research.google.com/drive/1a2fiRPBd1xvoJf0WNiMUgTYiLTs1XETf?usp=sharing
```

ChIP-seq analysis:

```
https://colab.research.google.com/drive/1GZ8v4Tq3LQMZI30qvdhF7ZW6Kf5GDyKX?usp=sharing
```

1.2 API Reference

This page contains auto-generated API reference documentation[1].

1.2.1 evoaug

EvoAug is a PyTorch package to pretrain sequence-based deep learning models for regulatory genomics data with evolution-inspired data augmentations followed by a finetuning on the original, unperturbed sequence data.

Submodules

evoaug.augment

Library of data augmentations for genomic sequence data.

To contribute a custom augmentation, use the following syntax:

```
class CustomAugmentation(AugmentBase):
    def __init__(self, param1, param2):
        self.param1 = param1
        self.param2 = param2

    def __call__(self, x: torch.Tensor) -> torch.Tensor:
        # Perform augmentation
        return x_aug
```

[1] Created with sphinx-autoapi

Module Contents

Classes

AugmentBase	Base class for EvoAug augmentations for genomic sequences.
RandomDeletion	Randomly deletes a contiguous stretch of nucleotides from sequences in a training
RandomInsertion	Randomly inserts a contiguous stretch of nucleotides from sequences in a training
RandomTranslocation	Randomly cuts sequence in two pieces and shifts the order for each in a training
RandomInversion	Randomly inverts a contiguous stretch of nucleotides from sequences in a training
RandomMutation	Randomly mutates sequences in a training batch according to a user-defined
RandomRC	Randomly applies a reverse-complement transformation to each sequence in a training
RandomNoise	Randomly add Gaussian noise to a batch of sequences with according to a user-defined

class evoaug.augment.**AugmentBase**

Base class for EvoAug augmentations for genomic sequences.

abstract __call__(*x*)

Return an augmented version of *x*.

Parameters
x (*torch.Tensor*) – Batch of one-hot sequences (shape: (N, A, L)).

Returns
Batch of one-hot sequences with random augmentation applied.

Return type
torch.Tensor

class evoaug.augment.**RandomDeletion**(*delete_min=0*, *delete_max=20*)

Bases: *AugmentBase*

Randomly deletes a contiguous stretch of nucleotides from sequences in a training batch according to a random number between a user-defined delete_min and delete_max. A different deletion is applied to each sequence.

Parameters

- **delete_min** (*int, optional*) – Minimum size for random deletion (defaults to 0).
- **delete_max** (*int, optional*) – Maximum size for random deletion (defaults to 20).

__call__(*x*)

Randomly delete segments in a set of one-hot DNA sequences.

Parameters
x (*torch.Tensor*) – Batch of one-hot sequences (shape: (N, A, L)).

Returns
Sequences with randomly deleted segments (padded to correct shape with random DNA)

Return type
torch.Tensor

class evoaug.augment.**RandomInsertion**(*insert_min=0*, *insert_max=20*)

Bases: *AugmentBase*

Randomly inserts a contiguous stretch of nucleotides from sequences in a training batch according to a random number between a user-defined insert_min and insert_max. A different insertions is applied to each sequence. Each sequence is padded with random DNA to ensure same shapes.

Parameters

- **insert_min** (*int, optional*) – Minimum size for random insertion, defaults to 0
- **insert_max** (*int, optional*) – Maximum size for random insertion, defaults to 20

__call__(*x*)

Randomly inserts segments of random DNA to a set of DNA sequences.

Parameters
x (*torch.Tensor*) – Batch of one-hot sequences (shape: (N, A, L)).

Returns
Sequences with randomly inserts segments of random DNA. All sequences are padded with random DNA to ensure same shape.

Return type
torch.Tensor

class evoaug.augment.**RandomTranslocation**(*shift_min=0*, *shift_max=20*)

Bases: *AugmentBase*

Randomly cuts sequence in two pieces and shifts the order for each in a training batch. This is implemented with a roll transformation with a user-defined shift_min and shift_max. A different roll (positive or negative) is applied to each sequence. Each sequence is padded with random DNA to ensure same shapes.

Parameters

- **shift_min** (*int, optional*) – Minimum size for random shift, defaults to 0.
- **shift_max** (*int, optional*) – Maximum size for random shift, defaults to 20.

__call__(*x*)

Randomly shifts sequences in a batch, x.

Parameters
x (*torch.Tensor*) – Batch of one-hot sequences (shape: (N, A, L)).

Returns
Sequences with random translocations.

Return type
torch.Tensor

class evoaug.augment.**RandomInversion**(*invert_min=0*, *invert_max=20*)

Bases: *AugmentBase*

Randomly inverts a contiguous stretch of nucleotides from sequences in a training batch according to a user-defined invert_min and invert_max. A different insertions is applied to each sequence. Each sequence is padded with random DNA to ensure same shapes.

Parameters

- **invert_min** (*int, optional*) – Minimum size for random insertion, defaults to 0.

- **invert_max** (*int, optional*) – Maximum size for random insertion, defaults to 20.

__call__(*x*)

Randomly inverts segments of random DNA to a set of one-hot DNA sequences.

Parameters
x (*torch.Tensor*) – Batch of one-hot sequences (shape: (N, A, L)).

Returns
Sequences with randomly inverted segments of random DNA.

Return type
torch.Tensor

class evoaug.augment.**RandomMutation**(*mutate_frac=0.05*)

Bases: *AugmentBase*

Randomly mutates sequences in a training batch according to a user-defined mutate_frac. A different set of mutations is applied to each sequence.

Parameters
mutate_frac (*float, optional*) – Probability of mutation for each nucleotide, defaults to 0.05.

__call__(*x*)

Randomly introduces mutations to a set of one-hot DNA sequences.

Parameters
x (*torch.Tensor*) – Batch of one-hot sequences (shape: (N, A, L)).

Returns
Sequences with randomly mutated DNA.

Return type
torch.Tensor

class evoaug.augment.**RandomRC**(*rc_prob=0.5*)

Bases: *AugmentBase*

Randomly applies a reverse-complement transformation to each sequence in a training batch according to a user-defined probability, rc_prob. This is applied to each sequence independently.

Parameters
rc_prob (*float, optional*) – Probability to apply a reverse-complement transformation, defaults to 0.5.

__call__(*x*)

Randomly transforms sequences in a batch with a reverse-complement transformation.

Parameters
x (*torch.Tensor*) – Batch of one-hot sequences (shape: (N, A, L)).

Returns
Sequences with random reverse-complements applied.

Return type
torch.Tensor

class evoaug.augment.**RandomNoise**(*noise_mean=0.0*, *noise_std=0.2*)

Bases: *AugmentBase*

Randomly add Gaussian noise to a batch of sequences with according to a user-defined noise_mean and noise_std. A different set of noise is applied to each sequence.

Parameters

- **noise_mean** (*float, optional*) – Mean of the Gaussian noise, defaults to 0.0.
- **noise_std** (*float, optional*) – Standard deviation of the Gaussian noise, defaults to 0.2.

__call__(*x*)

Randomly adds Gaussian noise to a set of one-hot DNA sequences.

Parameters
x (*torch.Tensor*) – Batch of one-hot sequences (shape: (N, A, L)).

Returns
Sequences with random noise.

Return type
torch.Tensor

evoaug.evoaug

Model (implemented in Pytorch Lightning) demonstrating how to use augmentations during training.

Module Contents

Classes

RobustModel	PyTorch Lightning module to specify how augmentation should be applied to a model.

Functions

load_model_from_checkpoint(model, checkpoint_path)	Load PyTorch lightning model from checkpoint.
augment_max_len(augment_list)	Determine whether insertions are applied to determine the insert_max,

class evoaug.evoaug.**RobustModel**(*model*, *criterion*, *optimizer*, *augment_list=[]*, *max_augs_per_seq=0*, *hard_aug=True*, *finetune=False*, *inference_aug=False*)

Bases: pytorch_lightning.LightningModule

PyTorch Lightning module to specify how augmentation should be applied to a model.

Parameters

- **model** (*torch.nn.Module*) – PyTorch model.
- **criterion** (*callable*) – PyTorch loss function
- **optimizer** (*torch.optim.Optimizer or dict*) – PyTorch optimizer as a class or dictionary
- **augment_list** (*list*) – List of data augmentations, each a callable class from augment.py. Default is empty list – no augmentations.

- **max_augs_per_seq** (*int*) – Maximum number of augmentations to apply to each sequence. Value is superceded by the number of augmentations in augment_list.
- **hard_aug** (*bool*) – Flag to set a hard number of augmentations, otherwise the number of augmentations is set randomly up to max_augs_per_seq, default is True.
- **finetune** (*bool*) – Flag to turn off augmentations during training, default is False.
- **inference_aug** (*bool*) – Flag to turn on augmentations during inference, default is False.

forward(*x*)

Standard forward pass.

configure_optimizers()

Standard optimizer configuration.

training_step(*batch*, *batch_idx*)

Training step with augmentations.

validation_step(*batch*, *batch_idx*)

Validation step without (or with) augmentations.

test_step(*batch*, *batch_idx*)

Test step without (or with) augmentations.

predict_step(*batch*, *batch_idx*)

Prediction step without (or with) augmentations.

_sample_aug_combos(*batch_size*)

Set the number of augmentations and randomly select augmentations to apply to each sequence.

_apply_augment(*x*)

Apply augmentations to each sequence in batch, x.

_pad_end(*x*)

Add random DNA padding of length insert_max to the end of each sequence in batch.

finetune_mode(*optimizer=None*)

Turn on finetune flag – no augmentations during training.

evoaug.evoaug.**load_model_from_checkpoint**(*model*, *checkpoint_path*)

Load PyTorch lightning model from checkpoint.

Parameters

- **model** (RobustModel) – RobustModel instance.
- **checkpoint_path** (*str*) – path to checkpoint of model weights

Returns

Object with weights and config loaded from checkpoint.

Return type

RobustModel

evoaug.evoaug.**augment_max_len**(*augment_list*)

Determine whether insertions are applied to determine the insert_max, which will be applied to pad other sequences with random DNA.

Parameters

augment_list (*list*) – List of augmentations.

Returns
Value for insert max.

Return type
int

PYTHON MODULE INDEX

e

INDEX

Symbols

A

C

E

F

L

M

P

R

T

V