Learn As You Go

Apr 28, 2020

Contents:

Python implementation of the Learn As You Go algorithm described in arxiv:1506:01079 and arxiv:2004.11929.

The package defines a decorator that can be applied to functions to convert them to functions which learn outputs as they go and emulate the true function when expected errors are low. Two emulators are included: the *k*-nearest neighbors Monte Carlo accelerator described there, and a simple neural network.

The basic usage of the emulator code is something like this:

```
@emulate(CholeskyNnEmulator)
def loglike(x):
    """
    Your complex and expensive function here
    """
    return -np.dot(x,x)
```

This decorates the function `loglike` so that it is an instance of the `Learner` class. It can be used in the same way as the original function: just call it as `loglike(x)`.

The `__call__(x)` method now hides some extra complexity: it uses the Learn As You Go emulation scheme. It learns both the output of `loglike(x)` and the difference between the emulated and the true values of `loglike(x)` so that it can make a prediction of future the error residuals. We then put a cutoff on the amount of error that one will allow for any local evaluation of the target function. Any call to the emulator that has a too-large error will be discarded and the actual function `loglike(x)` defined above will be evaluated instead.

The logic for generating training sets and returning a value from either the true function or the emulated function are contained in the `Learner` class. The `Learner` class relies on an emulator class to do the actual emulation.

You can define you own emulator. Define a class that inherits from `BaseEmulator` and define two methods on it: `set_emul_func(self, x_train: np.ndarray, y_train: np.ndarray)` and `set_emul_error_func(self, x_train: np.ndarray, y_train: np.ndarray)` that set functions for, respectively, `self.emul_func` and `self.emul_error_func`. An example of this definition is provided in *examples/example_custom_emulator.py*.

See readthedocs.org for more documentation.

CHAPTER 1

Installation

pip

The package is available on pypi.org. Install it with

```
pip install layg
```

anaconda

If you use anaconda you can create an appropriate environment and install to your python path by running

```
conda env create --file environment.yml
pip install -e .
```

from this directory.

1.1 Examples

1.1.1 Basic Usage

An example of basic use of *layg*.

```
"""
An example use of the `layg` package
"""

import matplotlib.pyplot as plt  # type: ignore
import numpy as np  # type: ignore

# TODO: remove NOQA when isort is fixed
from layg import CholeskyNnEmulator as Emulator  # NOQA
from layg import emulate  # NOQA

```

(continues on next page)

(continued from previous page)

```
def main():

    ndim = 2

    #####################
    #####################
    # Toy likelihood
    @emulate(Emulator)
    def loglike(x):
        if x.ndim != 1:
            loglist = []
            for x0 in x:
                loglist.append(-np.dot(x0, x0))
            return np.array(loglist)
        else:
            return np.array([-np.dot(x, x)])

    #####################
    #####################

    # Make fake data
    def get_x(ndim):
        """
        Sample from a Gaussian with mean 0 and std 1
        """

        return np.random.normal(0.0, 1.0, size=ndim)

    if ndim == 1:
        Xtrain = np.array([get_x(ndim) for _ in range(1000)])
        xlist = np.array([np.linspace(-3.0, 3.0, 11)]).T

    elif ndim == 2:

        Xtrain = np.array([get_x(ndim) for _ in range(10000)])
        xlist = np.array([get_x(ndim) for _ in range(10)])

    else:
        raise RuntimeError(
            "This number of dimensions has not been implemented for testing yet."
        )

    Ytrain = np.array([loglike(X) for X in Xtrain])
    loglike.train(Xtrain, Ytrain)

    loglike.output_err = True
    for x in xlist:
        print("x", x)
        print("val, err", loglike(np.array(x)))
    loglike.output_err = False

    # Plot an example
    assert loglike.trained

    fig = plt.figure()
    ax = fig.add_subplot(111)
```

(continues on next page)

(continued from previous page)

```
    x_len = 100

    x_data_plot = np.zeros((x_len, ndim))
    for i in range(ndim):
        x_data_plot[:, i] = np.linspace(0, 1, x_len)

    y_true = np.array([loglike.true_func(x) for x in x_data_plot])
    y_emul = np.array([loglike(x) for x in x_data_plot])
    y_emul_raw = np.array([loglike.emulator.emul_func(x) for x in x_data_plot])

    ax.plot(x_data_plot[..., 0], y_true, label="true", color="black")
    ax.scatter(x_data_plot[..., 0], y_emul, label="emulated", marker="+")
    ax.scatter(
        x_data_plot[..., 0],
        y_emul_raw,
        label="emulated\n no error estimation",
        marker="+",
    )

    ax.legend()

    ax.set_xlabel("Input")
    ax.set_ylabel("Output")

    fig.savefig("check.png")


def test_main():
    main()


if __name__ == "__main__":
    main()
```

1.1.2 Use with emcee

An example using *layg* with the common Markov chain Monte Carlo sampler emcee.

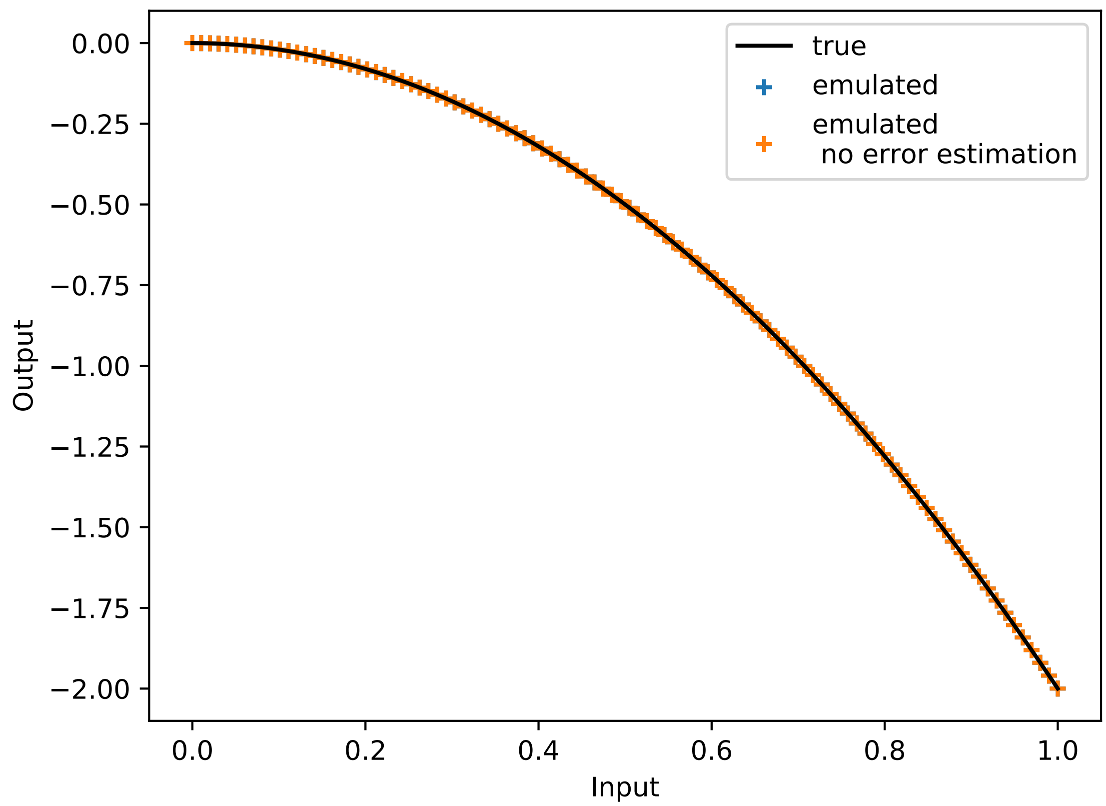
```
"""
An example use of the `learn_as_you_go` package with emcee
"""

import emcee  # type: ignore
import gif  # type: ignore
import matplotlib.pyplot as plt  # type: ignore
import numpy as np  # type: ignore

# TODO: remove NOQA when isort is fixed
from layg import CholeskyNnEmulator  # NOQA
from layg import emulate  # NOQA


def main():
```

(continues on next page)


true
emulated
emulated
no error estimation
0.00
−0.25
−0.50
−0.75
−1.00
−1.25
−1.50
−1.75
−2.00
Output
0.0
0.2
0.4
0.6
0.8
1.0
Input

(continued from previous page)

```
    ndim = 2

    nwalkers = 20
    niterations = 1000
    nthreads = 1

    np.random.seed(1234)

    # Toy likelihood
    @emulate(CholeskyNnEmulator)
    def loglike(x):
        return np.array([-np.dot(x, x) ** 1])

    loglike.output_err = True
    loglike.abs_err_local = 2

    # Starting points for walkers
    p0 = np.random.normal(-1.0, 1.0, size=(nwalkers, ndim))
    sampler = emcee.EnsembleSampler(nwalkers, ndim, loglike, threads=nthreads)

    # Sample with emcee
    with open("test.txt", "w") as f:
        for result in sampler.sample(p0, iterations=niterations, storechain=True):

            for pos, lnprob, err in zip(result[0], result[1], result[3]):
                for k in list(pos):
                    f.write("%s " % str(k))
                f.write("%s " % str(lnprob))
                f.write("%s " % str(err))
                f.write("\n")

    print("n exact evals:", loglike._nexact)
    print("n emul evals:", loglike._nemul)

    # Plot points sampled
    nframes = 50
    duration = 10
    frames = []
    lim = (-3, 3)

    for i in range(0, niterations * nwalkers, niterations * nwalkers // nframes):
        x = sampler.chain.reshape(niterations * nwalkers, ndim)[:i]
        y = np.array(sampler.blobs).reshape(niterations * nwalkers)[:i]
        frame = plot(x, y, lim)
        frames.append(frame)

    gif.save(frames, "mc.gif", duration=duration)


@gif.frame
def plot(x, err, lim):

    true = x[err == 0.0]
    emul = x[err != 0.0]

    plt.figure(figsize=(5, 5), dpi=100)
```

(continues on next page)

(continued from previous page)

```
    marker = "."
    alpha = 0.3

    plt.scatter(true[:, 0], true[:, 1], marker=marker, alpha=alpha, label="true")
    plt.scatter(emul[:, 0], emul[:, 1], marker=marker, alpha=alpha, label="emulated")

    plt.xlim(lim)
    plt.ylim(lim)

    legend = plt.legend(loc=1)
    for lh in legend.legendHandles:
        lh.set_alpha(1)


def test_main():
    main()


if __name__ == "__main__":
    main()
```

1.1.3 Custom Emulators

This example shows how to build a custom emulator by defining a subclass of *layg.emulator.BaseEmulator*.

The emulator simply learns the mean and standard deviation of the supplied training data.

In this example the emulated function is very simple: it returns real numbers drawn from a Gaussian distribution with some mean.

```
from typing import Callable

import matplotlib.pyplot as plt  # type: ignore
import numpy as np  # type: ignore

from layg import BaseEmulator, emulate  # NOQA


class MeanEmulator(BaseEmulator):
    """
    An emulator that returns the mean of the training values

    The error estimate is the standard deviation of the error in the cross validation␣
↪data.
    This emulator is not very useful other than as an example of how to write one.
    """

    def set_emul_func(self, x_train: np.ndarray, y_train: np.ndarray) -> None:
        self.emul_func: Callable[[np.ndarray], np.ndarray] = lambda x: np.mean(y_
↪train)

    def set_emul_error_func(self, x_cv: np.ndarray, y_cv_err: np.ndarray) -> None:
        self.emul_error: Callable[[np.ndarray], np.ndarray] = lambda x: y_cv_err.std()


MEAN = 2 + np.random.uniform(size=1)
```

(continues on next page)

(continued from previous page)

```
@emulate(MeanEmulator)
def noise(x: np.ndarray) -> np.ndarray:
    """
    Sample from a Gaussian distribution

    The scatter is small enough that the emulated value is always used.
    """

    return np.random.normal(loc=MEAN, scale=1e-2, size=1)


def main():
    """
    Plot some output from this emulator
    """

    NUM_TRAIN = noise.init_train_thresh
    NUM_TEST = 20
    XDIM = 1

    # Train the emulator
    x_train = np.random.uniform(size=(NUM_TRAIN, XDIM))
    y_train = np.array([noise(x) for x in x_train])

    # Output error estimates
    noise.output_err = True

    # Get values from the trained emulator
    x_emu = np.random.uniform(size=(NUM_TEST, XDIM))

    y_emu = np.zeros_like(x_emu)
    y_err = np.zeros_like(x_emu)

    for i, x in enumerate(x_emu):
        val, err = noise(x)
        y_emu[i] = val
        y_err[i] = err

    # Plot the results
    fig = plt.figure()
    ax = fig.add_subplot(111)

    ax.scatter(x_train[:, 0], y_train, marker="+", label="training values")
    ax.errorbar(
        x_emu,
        y_emu,
        yerr=y_err.flatten(),
        linestyle="None",
        marker="o",
        capsize=3,
        label="emulator",
        color="red",
    )

    ax.legend()
```

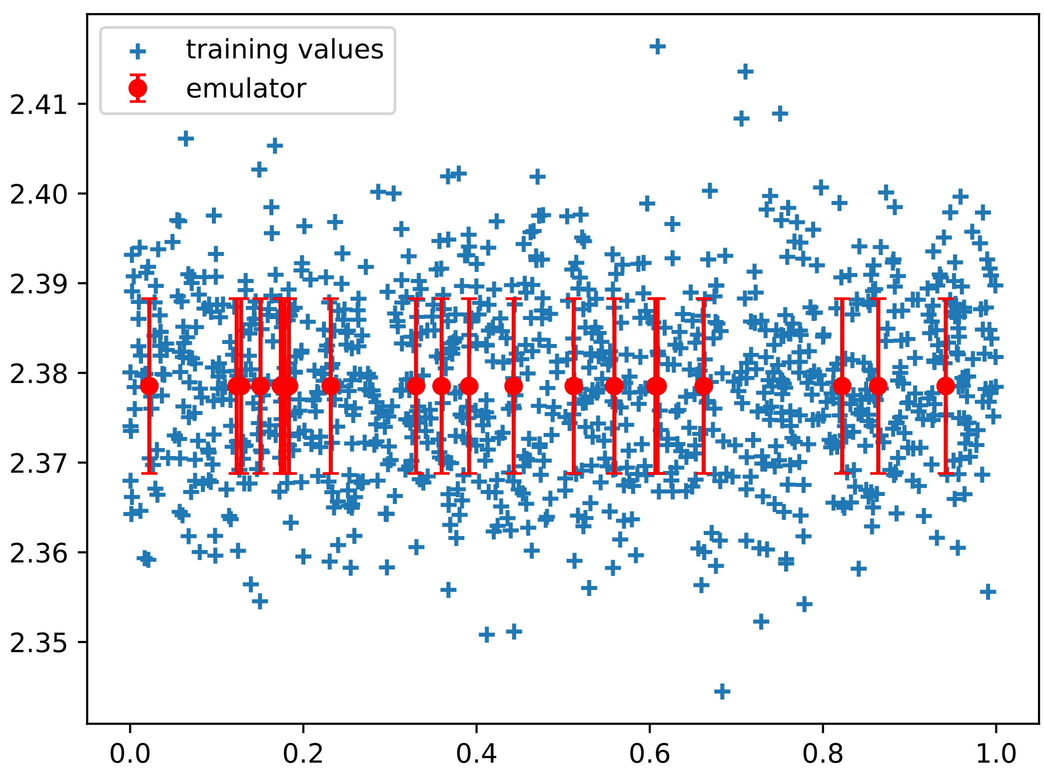
(continues on next page)

(continued from previous page)

```
    # `__file__` is undefined when running in sphinx
    try:
        fig.savefig(__file__ + ".png")
    except NameError:
        pass


def test_main():
    """
    Runs in pytest
    """
    main()


if __name__ == "__main__":
    main()
```



1.2 API

`layg.learner.Learner`(true_func, ...)	A class that contains logic for learning as you go
`layg.emulator.BaseEmulator`()	Base class from which emulators should inherit
`layg.emulator.cholesky_nn_emulator.CholeskyNnEmulator`()	An emulator based on Cholesky decomposition and nearest neighbours
`layg.emulator.torch_emulator.TorchEmulator`()	Class that uses pytorch to do emulation

1.2.1 layg.learner.Learner

class layg.learner.**Learner**(*true_func: Callable[[numpy.ndarray], numpy.ndarray], emulator_class*)

A class that contains logic for learning as you go

This class does not contain any emulation but should be constructed with an emulator containing emulation logic. The emulator must be a subclass of BaseEmulator, implementing two methods, *set_emul_func* and *set_emul_error_func*, that set the respective functions.

Attributes

emulator_class [BaseEmulator] The type of emulator used.

emulator [BaseEmulator] An instance of the class *emulator_class*. This is where the heavy lifting goes on.

true_func [Callable] The function which is emulated

frac_err_local [float] Maximum fractional error in emulated function. Calls to emulation function that exceed this error level are evaluated exactly instead. Default: 1.0

abs_err_local [float] Maximum absolute error allowed in emulated function. Calls to emulation function that exceed frac_err_local but are lower than abs_err_local are emulated, rather than exactly evaluated. FIXME: this doesn't happen Default: 0.05

output_err [bool] Whether to output an error estimate. Set to False if you do not want the error to be an output of the emulated function. Set to True if you do. Default: False

trained [bool] Whether the emulator has been trained

used_train_x [List[np.ndarray]]

used_train_y [List[np.ndarray]] Values from the true function that were used last time the emulator was trained

batch_train_x [List[np.ndarray]]

batch_train_y [List[np.ndarray]] Values from the true function that have not yet been used to train the emulator

init_train_thresh [int] Number of points to accumulate before training the emulator

frac_cv [float] Fraction of training set to use for error modelling The default value of 0.5 means that the prediction and the error are estimated off the same amount of data.

Methods

`__call__`(self, x)	The method that is executed when the wrapped function is called

Continued on next page

Table 2 – continued from previous page

`emulation_is_valid`(self, val, err)	Check if an emulated value is valid and likely accurate
`eval_true_func`(self, x)	Wrapper for evaluating true function
`split_CV`(self, xdata, ydata, frac_cv)	Splits a dataset into a cross-validation and training set.
`train`(self, x_train, y_train)	Train a ML algorithm to replace true_func: X –> Y.

__init__(*self, true_func: Callable[[numpy.ndarray], numpy.ndarray], emulator_class*)

Constructor for Learner class

Parameters

true_func [Callable] Function to be emulated

emulator_class [BaseEmulator] The emulator class to be used

Methods

`__init__`(self, true_func, numpy.ndarray], . . .)	Constructor for Learner class
`emulation_is_valid`(self, val, err)	Check if an emulated value is valid and likely accurate
`eval_true_func`(self, x)	Wrapper for evaluating true function
`split_CV`(self, xdata, ydata, frac_cv)	Splits a dataset into a cross-validation and training set.
`train`(self, x_train, y_train)	Train a ML algorithm to replace true_func: X –> Y.

1.2.2 layg.emulator.BaseEmulator

class `layg.emulator.`**BaseEmulator**

Base class from which emulators should inherit

This class is abstract. The child class must implement the marked methods.

Methods

`add_data`(self, x_train, y_train)	Add data to the training set on the fly

set_emul_error_func	
set_emul_func	

__init__(*self*)

Initialize self. See help(type(self)) for accurate signature.

Methods

`__init__`(self)	Initialize self.
`add_data`(self, x_train, y_train)	Add data to the training set on the fly

Continued on next page

Table 5 – continued from previous page

set_emul_error_func(self, x_cv, y_cv_err)	
set_emul_func(self, x_train, y_train)	

1.2.3 layg.emulator.cholesky_nn_emulator.CholeskyNnEmulator

class layg.emulator.cholesky_nn_emulator.**CholeskyNnEmulator**

An emulator based on Cholesky decomposition and nearest neighbours

This emulator described in detail in arXiv:1506.01079.

Methods

add_data(self, x_train, y_train)	Add data to the training set on the fly

set_emul_error_func	
set_emul_func	

__init__(*self*)

Initialize self. See help(type(self)) for accurate signature.

Methods

__init__(self)	Initialize self.
add_data(self, x_train, y_train)	Add data to the training set on the fly
set_emul_error_func(self, x_cv, y_cv_err)	
set_emul_func(self, x_train, y_train)	

1.2.4 layg.emulator.torch_emulator.TorchEmulator

class layg.emulator.torch_emulator.**TorchEmulator**

Class that uses pytorch to do emulation

The Universal Approximation Theorem says that any Lebesgue integrable function can be approximated by a feed-forward network with sufficient layers of sufficient width. It doesn't guarantee that we can train the network though.

Methods

add_data(self, x_train, y_train)	Add data to the training set on the fly
set_emul_error_func(self, x_cv, y_cv_err)	Fit a quadratic to the residuals and mean distance to nearby points

set_emul_func	

__init__(*self*)

Initialize self. See help(type(self)) for accurate signature.

Methods

`__init__`(self)	Initialize self.
`add_data`(self, x_train, y_train)	Add data to the training set on the fly
`set_emul_error_func`(self, x_cv, y_cv_err)	Fit a quadratic to the residuals and mean distance to nearby points
`set_emul_func`(self, x_train, y_train)	

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Index

Symbols

B

C

L

T