Domain Decomposition and Generative Adversarial Networks for modelling fluid flow

Release 0.2

Jón Atli Tómasson

CONTENTS:

1 The ddgan class	1
2 Indices and tables	9
Python Module Index	11
Index	13

CHAPTER

ONE

THE DDGAN CLASS

```
class ddgan.GAN(nsteps: int = 10, ndims: int = 10, batch_size: int = 20, batches: int = 10, seed: int = 143, epochs: int = 500, nLatent: int = 10, n_critic: int = 5, lmbda: int = 10, gen_learning_rate: float = 0.0001, disc_learning_rate: float = 0.0001, noise: bool = False, logs_location: str = './logs/gradient_tape/', model_location: str = 'models/', noise_level: float = 0.001, n_gradient_ascent: int = 500)
```

Bases: object

Class for the predictive WGAN.

Inspired by: - Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky,

Vincent Dumoulin, and Aaron Courville. Improved Training of Wasserstein GANs, 2017

- Céesar Quilodráan-Casas, Vinicius Santos Silva, Rossella Arcucci, Claire E. Heaney, Yike Guo, and Christopher C. Pain. Digital twins based on bidirectional LSTM and GAN for modelling the COVID-19 pandemic, 2021.
- D. Xiao, C.E. Heaney, F. Fang, L. Mottet, R. Hu, D.A. Bistrian,

E. Aristodemou, I.M. Navon, and C.C. Pain. A domain decomposition non-intrusive reduced order model for turbulent flows. Computers & Fluids, 182:15{27, 2019.

```
batch_size: int = 20
batches: int = 10
```

d_loss = None

d_summary_writer = None

disc_learning_rate: float = 0.0001

discriminator = None

 $discriminator_loss(d_real: numpy.ndarray, d_fake: numpy.ndarray) \rightarrow float$

Calculate the loss for the discriminator as the sum of the reduced real and fake discriminator losses

Parameters

- **d_real** (*float*) Discriminator loss form classifying real data
- **d_fake** (*float*) Discriminator loss form classifying fake data

Returns Discriminator loss

Return type float

discriminator_opt = None

epochs: int = 500

```
g_loss = None
g_summary_writer = None
gen_learning_rate: float = 0.0001
generator = None
generator_loss(d fake: numpy.ndarray) \rightarrow float
     Calculate the loss of the generator as the negative reduced fake discriminator loss. The generator has the
     task of fooling the discriminator.
         Parameters d_fake (float) – Discriminator loss from classifying fake data
         Returns Generator loss
         Return type float
generator_opt = None
initializer = <tensorflow.python.keras.initializers.initializers_v2.RandomNormal</pre>
object>
learn_hypersurface_from_POD_coeffs(training_data: numpy.ndarray)
     Umbrella function that makes logs and begins training
         Parameters training_data (np.ndarray) - [description]
         Returns Fitted data and future predictions
         Return type (tensor)
lmbda: int = 10
logs_location: str = './logs/gradient_tape/'
make\_GAN(find\_old\_model=False) \rightarrow None
     Searching for an existing model, creating one from scratch if not found.
         Parameters find_old_model (bool, optional) - To search for an existing model at
            self.model location Defaults to False.
make\_discriminator() \rightarrow None
     Create the discriminator network
make\_generator() \rightarrow None
    Create the generator network
make_logs() \rightarrow None
     Printing summaries for generator, discriminator and w in self.logs.location
model_location: str = 'models/'
nLatent: int = 10
n_critic: int = 5
n_gradient_ascent: int = 500
ndims: int = 10
noise: bool = False
noise_generator = None
```

```
noise_level: float = 0.001
     nsteps: int = 10
     print_loss() \rightarrow None
           Printing the loss results
     random_generator(size=None, random_state=None)
     resetting_states() \rightarrow None
           Resetting model loss states
     resume_training(training_data, start_epoch)
           Umbrella function for resuming training from starting epoch.
               Parameters
                    • training_data (np.ndarray) - Structured training data in 1d array
                    • start_epoch (int) – epoch to resume from
     save\_gan(epoch: int) \rightarrow None
           Saving a trained model
               Parameters epoch (int) – Epoch number
     seed: int = 143
     setup(find \ old \ model = False) \rightarrow None
           Setting up the neccecary values for the GAN class
               Parameters find_old_model (bool) – whether to look for a ready model in stead of building
                   one from scratch`
     train(training_data: numpy.ndarray) → None
           Training the GAN
               Parameters training_data (np.ndarray) – Actual values for comparison
     w_loss = None
     w_summary_writer = None
     write_summary(epoch: int) \rightarrow None
           Writing a summary from the current model state
               Parameters epoch (int) – Current epoch
class ddgan. Optimize(start from: int = 100, nPOD: int = 10, nLatent: int = 10, dt: int = 1, npredictions: int =
                          20, optimizer epochs: int = 5000, evaluated subdomains: int = 2, cycles: int = 3, gan:
                          Optional[ddgan.src.Train.GAN] = None, nOptimized: int = 50, debug: bool = False,
                          initial_values: str = 'Past', disturb: bool = False)
     Bases: object
     Finding position and orienting within the latent space to predict in time.
     Inspired by: - Céesar Quilodráan-Casas, Vinicius Santos Silva,
           Rossella Arcucci, Claire E. Heaney, Yike Guo, and Christopher C. Pain. Digital twins based on
           bidirectional LSTM and GAN for modelling the COVID-19 pandemic, 2021.
     add_bc(the input, i: int, boundrary condition: numpy.ndarray)
           Adding boundrary conditions to left and rightmost domains
```

```
Parameters
```

- **the_input** (*tensor*) Current iteration tensor
- i (int) Iteration number
- boundrary_condition (np.ndarray) Boundrary values

Returns Updated iteration tensor

Return type tensor

communicate(the_input, prediction: numpy.ndarray, j: int, domains: numpy.ndarray)

Communicate with neighbouring subdomains

Parameters

- **the_input** (*tensor*) Current iteration guess
- **prediction** (*np.ndarray*) Latent values
- **j** (*int*) Domain number
- **domains** (*np.ndarray*) Iteration ordering of domains

Returns Updated iteration tensor

```
Return type tensor
```

```
cumulative_steps = None
```

cycles: int = 3

debug: bool = False

dim_steps = None

disturb: bool = False

dt: int = 1

evaluated_subdomains: int = 2

gan: ddgan.src.Train.GAN = None

initial_guess(the_input)

Adding initial guesses for next iteration

Parameters the_input (tensor) – Current iteration tensor

Returns Updated iteration tensor

Return type tensor

initial_values: str = 'Past'

mse = <tensorflow.python.keras.losses.MeanSquaredError object>

mse_loss(input, output)

Mean square error loss function

Parameters

- **input** (*tensor*) Predicted values
- output (tensor) Actual value

Returns mse loss value

Return type int

```
nLatent: int = 10
nOptimized: int = 50
nPOD: int = 10
```

npredictions: int = 20

opt_latent_var(*latent_var*: *tensorflow.python.ops.variables.Variable*, *output*: *numpy.ndarray*)

Main input optimization loop optimizing the latent variable based on mse

Parameters

- latent_var (tf.variable) Variable to be optimized
- **output** (*np.ndarray*) Actual output

Returns loss variable float: norm of the latent variables

Return type float

```
optimizer = <tensorflow.python.keras.optimizer_v2.adam.Adam object>
optimizer epochs: int = 5000
```

predict(*training_data: numpy.ndarray*, *scaling=None*) → numpy.ndarray Communicator with the optimization scripts

Parameters

- training_data (np.ndarray) Data used in the training of the GAN
- **scaling** (*sklearn.preprocessing.MinMaxScaler*, *optional*) Scaling used to normalize training data. Defaults to None.

Returns predictions

Return type np.ndarray

predictDD(*training_data: numpy.ndarray, boundrary_conditions: numpy.ndarray, dim_steps=None*) → list Prediction script if Domain Decomposition is applied

Parameters

- training_data (np.ndarray) -
- boundrary_condition (np.ndarray) values for leftmost and rightmost domains
- dim_steps (np.ndarray) number of samples in each dimension of gan

start from: int = 100

timestep_loop(real output: numpy.ndarray, prev latent: numpy.ndarray)

Optimizes inputs either from a previous timestep or from new randomly initialized inputs

Parameters

- real_output (np.ndarray) Actual values
- **prev_latent** (*np.ndarray*) Latent values from previous iteration

Returns Updated values list: Loss values np.ndarray: Converged values np.ndarray: Initial z values list: Norm of latent variables

Return type np.ndarray

timestep_loopDD(*real_output: numpy.ndarray*, *prev_latent: numpy.ndarray*) → numpy.ndarray Optimizes inputs either from a previous timestep or from new randomly initialized inputs

Parameters

- real_output (np.ndarray) Actual values
- prev_latent (np.ndarray) Latent values from previous iteration

Returns Updated values

Return type np.ndarray

timesteps(*initial: numpy.ndarray*, *inn: numpy.ndarray*) → numpy.ndarray Outermost loop. Collecting the predicted points and iterating through predictions

Parameters

- **initial** (*np.ndarray*) Initial value array
- inn (np.ndarray) Gan input array

Returns Predicted points

Return type np.ndarray

timestepsDD(*initial: numpy.ndarray*, *inn: numpy.ndarray*, *boundrary_condition: numpy.ndarray*) \rightarrow list Outermost loop. Collecting the predicted points and iterating through predictions

Parameters

- initial (np.ndarray) Initial value array
- **inn** (*np.ndarray*) Gan input array
- boundrary_condition (np.ndarray) values for leftmost and rightmost domains

Returns Predicted points

Return type np.ndarray

update_updated(tmp: numpy.ndarray, updated, j: int)

Update latent values

Parameters

- tmp (np.ndarray) Values for current timestep
- **updated** (tensor) Values for previous timestep
- j (int) Domain number

Returns Updated tensor

Return type tensor

ddgan.set_seed(seed)

Sets seed for random, numpy and tensorflow

Parameters seed (int) – Random number generator seed

 $\label{eq:ddgan.train_step} ddgan. \ train_step(\textit{gan, noise: numpy.ndarray, real: numpy.ndarray, reverse_step: bool = False}) \rightarrow None \\ Training the gan for a single step$

Parameters

- gan (GAN) Model object
- noise (np.ndarray) Gaussian noise input
- real (np.ndarray) Actual values
- reverse_step (bool) Whether to make the discriminator take a step back

ddgan.truncated_normal(mean=0.0, sd=1.0, low=-4.0, upp=4.0)

Generating a trunicated scipy random number generator

Parameters

- mean (float, optional) mean of the distribution. Defaults to 0.
- **sd** (*float*, *optional*) standard deviation. Defaults to 1.
- low (float, optional) lower bound. Defaults to -5.
- upp (float, optional) upper bound. Defaults to 5.

Returns trunicated normal distribution rng

Return type scipy.stats obj

Domain Decomposition and Generative Adversarial Networks for modelling fluid flow, Release 0.2			

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

Domain Decomposition and Generative Adversarial Networks for modelling fluid flow, Release 0.2

PYTHON MODULE INDEX

d ddgan, 1



Python Module Index

INDEX

A	initial_values (ddgan.Optimize attribute), 4
<pre>add_bc() (ddgan.Optimize method), 3</pre>	initializer (ddgan.GAN attribute), 2
В	L
batch_size (ddgan.GAN attribute), 1 batches (ddgan.GAN attribute), 1	<pre>learn_hypersurface_from_POD_coeffs() (ddgan.GAN method), 2 lmbda (ddgan.GAN attribute), 2</pre>
C	logs_location (ddgan.GAN attribute), 2
<pre>communicate() (ddgan.Optimize method), 4 cumulative_steps (ddgan.Optimize attribute), 4 cycles (ddgan.Optimize attribute), 4</pre>	M make_discriminator() (ddgan.GAN method), 2
D	<pre>make_GAN() (ddgan.GAN method), 2 make_generator() (ddgan.GAN method), 2</pre>
d_loss (ddgan.GAN attribute), 1	make_logs() (ddgan.GAN method), 2
d_summary_writer (ddgan.GAN attribute), 1 ddgan module, 1	<pre>model_location (ddgan.GAN attribute), 2 module ddgan, 1</pre>
debug (ddgan.Optimize attribute), 4	mse (ddgan.Optimize attribute), 4
dim_steps (ddgan.Optimize attribute), 4	mse_loss() (ddgan.Optimize method), 4
disc_learning_rate (ddgan.GAN attribute), 1 discriminator (ddgan.GAN attribute), 1	N
discriminator_loss() (ddgan.GAN method), 1 discriminator_opt (ddgan.GAN attribute), 1 disturb (ddgan.Optimize attribute), 4 dt (ddgan.Optimize attribute), 4	n_critic (ddgan.GAN attribute), 2 n_gradient_ascent (ddgan.GAN attribute), 2 ndims (ddgan.GAN attribute), 2 nLatent (ddgan.GAN attribute), 2
E	nLatent (ddgan.Optimize attribute), 4 noise (ddgan.GAN attribute), 2
epochs (ddgan.GAN attribute), 1	noise_generator (ddgan.GAN attribute), 2
evaluated_subdomains (ddgan.Optimize attribute), 4	noise_level (ddgan.GAN attribute), 2
G	nOptimized (ddgan.Optimize attribute), 5 nPOD (ddgan.Optimize attribute), 5
g_loss (ddgan.GAN attribute), 1 g_summary_writer (ddgan.GAN attribute), 2	npredictions (ddgan.Optimize attribute), 5 nsteps (ddgan.GAN attribute), 3
GAN (class in ddgan), 1 gan (ddgan.Optimize attribute), 4	0
gen_learning_rate (ddgan.GAN attribute), 2 generator (ddgan.GAN attribute), 2 generator_loss() (ddgan.GAN method), 2 generator_opt (ddgan.GAN attribute), 2	opt_latent_var() (ddgan.Optimize method), 5 Optimize (class in ddgan), 3 optimizer (ddgan.Optimize attribute), 5 optimizer_epochs (ddgan.Optimize attribute), 5
1	Р
initial_guess() (ddgan.Optimize method), 4	<pre>predict() (ddgan.Optimize method), 5</pre>
initial_gaess() (augun.opimize memou),	predict() (adgan.Optimize memod), 5

```
predictDD() (ddgan.Optimize method), 5
print_loss() (ddgan.GAN method), 3
R
random_generator() (ddgan.GAN method), 3
resetting_states() (ddgan.GAN method), 3
resume_training() (ddgan.GAN method), 3
S
save_gan() (ddgan.GAN method), 3
seed (ddgan.GAN attribute), 3
set_seed() (in module ddgan), 6
setup() (ddgan.GAN method), 3
start_from (ddgan.Optimize attribute), 5
Т
timestep_loop() (ddgan.Optimize method), 5
timestep_loopDD() (ddgan.Optimize method), 5
timesteps() (ddgan.Optimize method), 6
timestepsDD() (ddgan.Optimize method), 6
train() (ddgan.GAN method), 3
train_step() (in module ddgan), 6
truncated_normal() (in module ddgan), 6
U
update_updated() (ddgan.Optimize method), 6
W
w_loss (ddgan.GAN attribute), 3
w_summary_writer (ddgan.GAN attribute), 3
write_summary() (ddgan.GAN method), 3
```

14 Index