Boring but important disclaimers:

- If you are not getting this from the GitHub repository or the associated Canvas page (e.g. CourseHero, Chegg etc.), you are probably getting the substandard version of these slides Don't pay money for those, because you can get the most updated version for free at

  `https://github.com/julianmak/OCES5303_ML_ocean`

  The repository principally contains the compiled products rather than the source for size reasons.

- Associated Python code (as Jupyter notebooks mostly) will be held on the same repository. The source data however might be big, so I am going to be naughty and possibly just refer you to where you might get the data if that is the case (e.g. JRA-55 data). I know I should make properly reproducible binders etc., but I didn't...
- I do not claim the compiled products and/or code are completely mistake free (e.g. I know I don't write Pythonic code). Use the material however you like, but use it at your own risk.
- As said on the repository, I have tried to honestly use content that is self made, open source or explicitly open for fair use, and citations should be there. If however you are the copyright holder and you want the material taken down, please flag up the issue accordingly and I will happily try and swap out the relevant material.

OCES 5303 :
ML methods in Ocean Sciences

Session 8: GANs

Outline

- Generative Adversarial Networks (GANs)

  → training via generator vs. discriminator

  → issues of mode collapse

  → conditional GAN (cGAN)

  → Deep Convolutional GAN (DCGAN)

Figure: Gans of different varieties.

Oceanic application

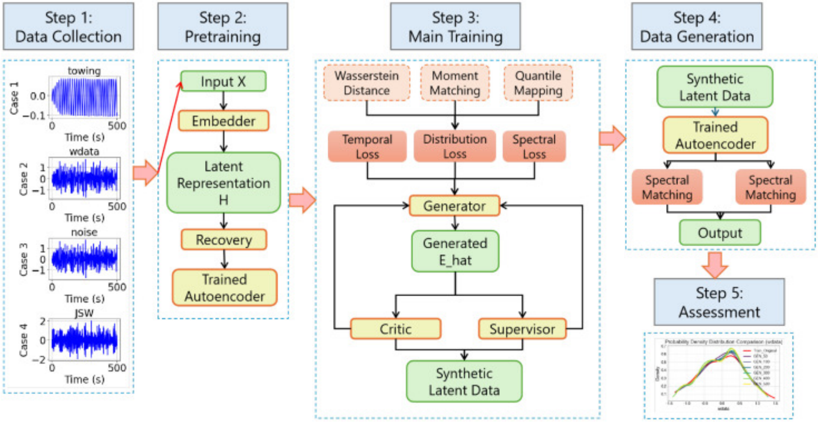

Figure: Schematic for wave reconstruction with a GAN. From Fig. 1 of Chen *et al.* (2026).

Oceanic application

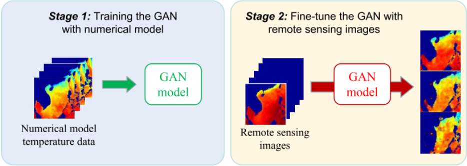

Figure: Reconstruction of satellite images based on GAN trained on numerical data. From Fig. 1 of Meng *et al.* (2023).

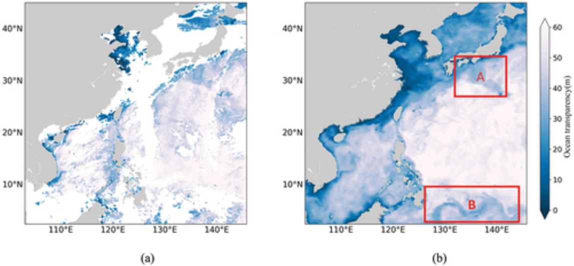

Figure: GAN based filling out of satellite images. From Fig. 11 of Zhou *et al.* (2023).

Oceanic application

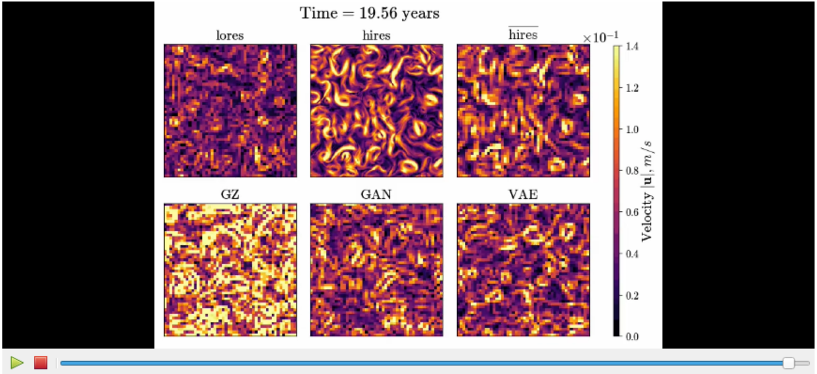

Figure: Visualisation of a low resolution model with parameterised eddies generated based on GAN. From Perezhogin *et al.* (2023).

GANs: tl;dr

Figure: Structure of a GAN in a nutshell.

GANs

1. a generator takes input and **generates** “fake” data
   $\rightarrow$ input is usually some random noise
   $\rightarrow$ output can be numbers, images, sound, whatever

GANs

1. a generator takes input and **generates** "fake" data
   $\rightarrow$ input is usually some random noise
   $\rightarrow$ output can be numbers, images, sound, whatever

2. a discriminator takes the data and classifies them as whether they are "real" or "fake"
   $\rightarrow$ input is the numbers/images/sound/whatever
   $\rightarrow$ "real" is training data, "fake" is output from **generator**
   $\rightarrow$ output is a **number** in $[0, 1]$
   $\rightarrow$ usually $0 =$ fake and $1 =$ real

GANs

3. **discriminator** tries to maximise the score

   → i.e. being good at telling between "real" and "fake"s

4. **generator** tries to minimise the score

   → i.e. by generating very good "fakes"s

GANs

3. **discriminator** tries to maximise the score

   → i.e. being good at telling between "real" and "fake"s

4. **generator** tries to minimise the score

   → i.e. by generating very good "fakes"s

5. results in minimax problem, where they "fight" each other and both get "better" in due course

   → cf. **evolutionary arms race** in evolution biology

   → original GAN paper assumes zero sum games, and ordering of moves doesn't matter

   → **zero-sum** = one's gain is another's loss

   → universality as generator/discriminator are NN based

   → in practice discriminator moves first

vanilla GAN

- use penguins data as a quick test
  → aim to generate four numbers for some input noise
  → the `species` labels are not used for now

- see notebook for implementation of GAN
  → use PyTorch here (I wasn't comfortable with my Keras implementation, see notebook)
  → generator and discriminator are just MLPs
  → loss uses binary cross entropy
  → optimizer is Adam as usual

vanilla GAN

- use penguins data as a quick test
  → aim to generate four numbers for some input noise
  → the `species` labels are not used for now

- see notebook for implementation of GAN
  → use PyTorch here (I wasn't comfortable with my Keras implementation, see notebook)
  → generator and discriminator are just MLPs
  → loss uses binary cross entropy
  → optimizer is Adam as usual

! subtleties with the training loop
  → did manual batching partly because of that
  → need to be careful with where loss and gradients are computed, and which models are updated where

vanilla GAN

1. the discriminator moves first

   → generate outputs from noise, label as 0s, compute loss, train

   → take real data, label as 1s, compute loss, train (!!!)

   → generator is <u>fixed</u> at this substep

vanilla GAN

1. the discriminator moves first

   $\rightarrow$ generate outputs from noise, label as 0s, compute loss, train

   $\rightarrow$ take real data, label as 1s, compute loss, train (!!!)

   $\rightarrow$ generator is <u>fixed</u> at this substep

2. the generator moves second

   $\rightarrow$ generate outputs from noise, label as 1s

   $\rightarrow$ let the discriminator do its thing

   $\rightarrow$ compute loss based on discriminator output, train

   $\rightarrow$ discriminator is <u>fixed</u> at this substep

vanilla GAN

1. the discriminator moves first

   → generate outputs from noise, label as 0s, compute loss, train

   → take real data, label as 1s, compute loss, train (!!!)

   → generator is <u>fixed</u> at this substep
2. the generator moves second

   → generate outputs from noise, label as 1s

   → let the discriminator do its thing

   → compute loss based on discriminator output, train

   → discriminator is <u>fixed</u> at this substep
3. above is one epoch, repeat as necessary

!!! otherwise possible to have loss reduction simply by having a <u>weak</u> discriminator (because of zero-sum game nature)

vanilla GAN

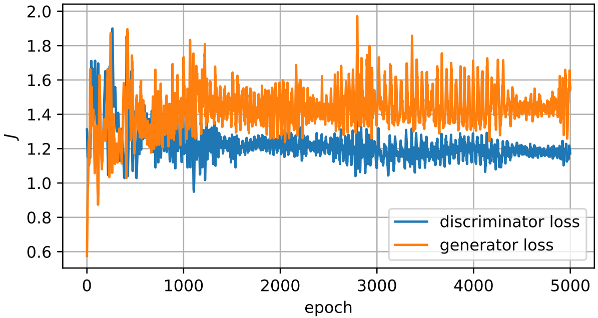

Figure: Generator and discriminator loss as a function of epochs for the penguins data.

- note the loss values are not small
  $\rightarrow$ not an issue as such
- quite a bit of fluctuation
  $\rightarrow$ competition between generator and discriminator via minimax/zero-sum game

GAN: mode collapse

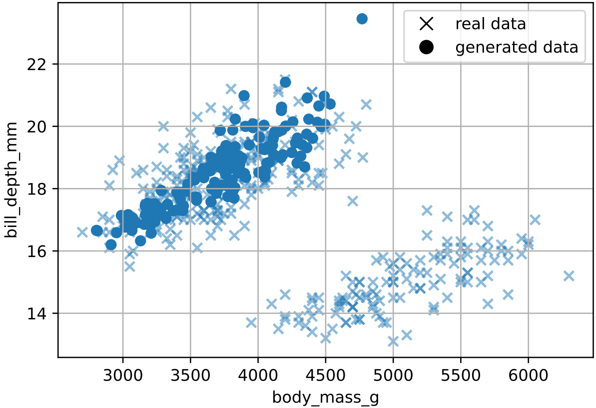

Figure: Outputs from the generator for the vanilla GAN.

- model exposed to Gentoo data but none being generated?

GAN: mode collapse

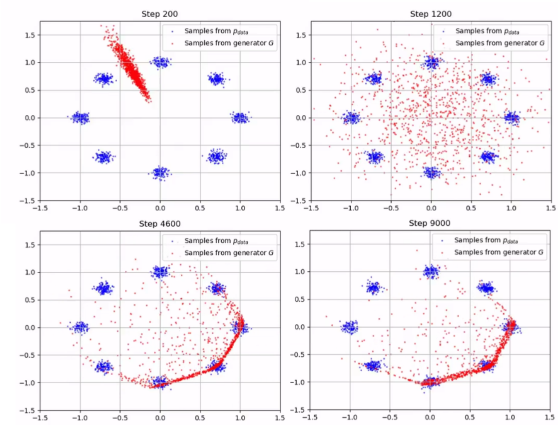

Figure: Visualisation of mode collapse via generator output as a function of training epoch. From Christoph Reich's GitHub page.

GAN: mode collapse

- mode collapse refers to generator specialising too early

  $\rightarrow$ reduces diversity of resulting output

  $\rightarrow$ generator "plays it too safe"

- one way to avoid is to use a different loss such as the Wasserstein metric

  $\rightarrow$ amount of "work" to move one pdf to another, measures distances between pdfs

  $\rightarrow$ deviation from data pdf is then penalised

  $\rightarrow$ not standard loss in PyTorch at the time of writing

- other methods (e.g. unrolling) is possible, but going to do something easier...

conditional GAN (cGAN)

- we could just make the GAN use the label information
  $\rightarrow$ GAN **conditioned** on the label information
  $\rightarrow$ e.g. a **prompt** when you get AI to generate an image

- side-steps but does not avoid mode collapse
  $\rightarrow$ produced samples could still be lacking in diversity

conditional GAN (cGAN)

- we could just make the GAN use the label information
  $\rightarrow$ GAN **conditioned** on the label information
  $\rightarrow$ e.g. a **prompt** when you get AI to generate an image

- side-steps but does not avoid mode collapse
  $\rightarrow$ produced samples could still be lacking in diversity

- setting basically exactly the same, just need some trickery (!!!) to merge the label information in
  $\rightarrow$ label information usually in one-hot form, i.e.

$$(\text{Adelie}, \text{Chinstrap}, \text{Gentoo}) \quad \rightarrow \quad ([1,0,0],[0,1,0],[0,0,1])$$

  $\rightarrow$ bulk up input dimension by three here

conditional GAN (cGAN)

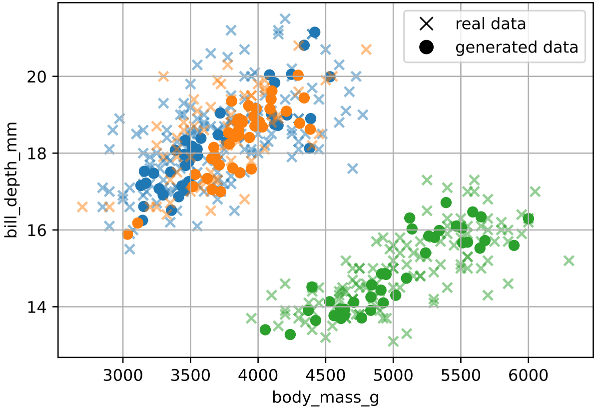

Figure: Outputs from the generator for the cGAN.

- still some hints of mode collapse within specie?

convolutional GAN (DCGAN)

- example on the Mercator eddy data (see notebook 05)
  $\rightarrow$ already processed and standardised, just using SST images of cyclonic eddies here

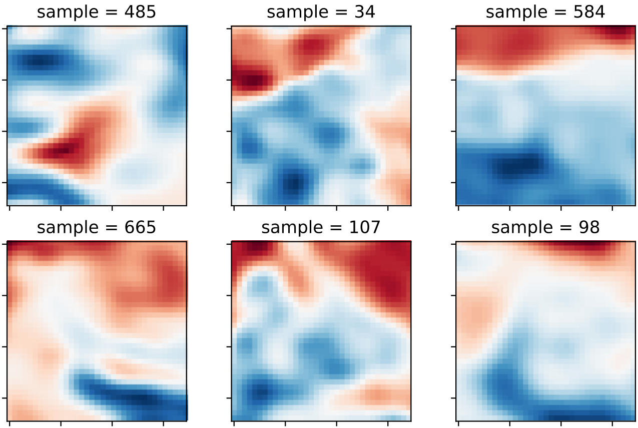

Figure: Some samples of the SSTs of cyclonic eddies from the Mercator eddy dataset.

convolutional GAN

- see notebook for the network structure

  $\rightarrow$ `Conv2d` and `ConvTranspose2d` again but in PyTorch

- some care is needed to make sure the dimensions at each stage are correct and intended

  $\rightarrow$ would suggest doing the **discriminator** first

  $\rightarrow$ possibly easier to figure how to go from

$$(\text{channel}, \text{height}, \text{width}) \quad \rightarrow \quad (1, 1, 1)$$

  via `Conv2d` layers than the other way round

  $\rightarrow$ once you have **discriminator**, could mirror the ordering of operations, swap out `Conv2d` for `ConvTranspose2d`, which then does

$$(\text{noise}, 1, 1) \quad \rightarrow \quad (\text{channel}, \text{height}, \text{width})$$

convolutional GAN

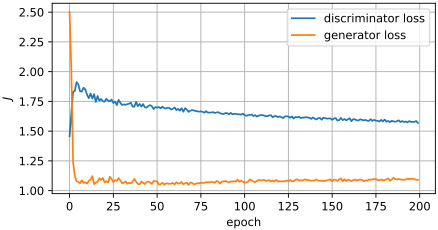

Figure: Generator and discriminator loss as function of epoch.

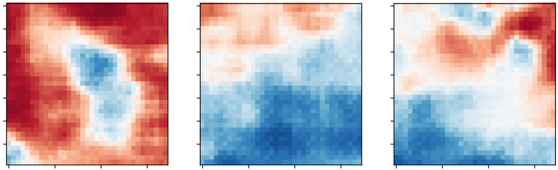

Figure: Visualisation of samples of generator output.

convolutional GAN

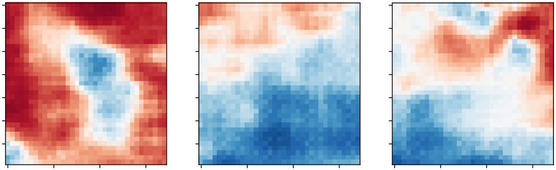

Figure: Visualisation of samples of generator output.

- is the above "good"?

  $\rightarrow$ some guiding principles to judge?

  $\rightarrow$ probably need to look into the full dataset quite a bit more

  $\rightarrow$ compute some statistics to be quantitative about it

Demonstration

Figure: Generated image from ‘*HKUST with sun setting over the sea*‘, from `https://visualgpt.io/`. Spot any issues here?

- some varieties of GANs

  → basics in constructing generator and discriminator

  → deal with issues of mode collapse?

  → how to judge whether something is “good” quantitatively?

  → Keras implementation?

  (see notebook)

Moving to a Jupyter notebook →