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 7: RNNs

Outline

- Recurrent Neural Networks (RNNs)

  $\rightarrow$ hidden states

  $\rightarrow$ time-series prediction
- GRUs and LSTMs
- ConvLSTMs

  $\rightarrow$ sequential image prediction

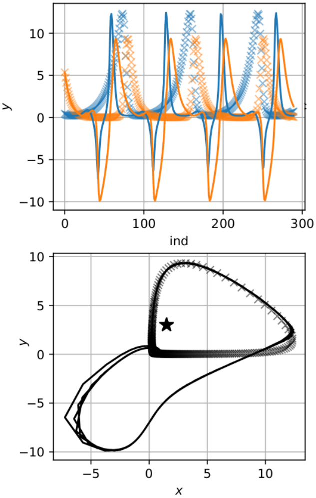

Figure: RNN doing insane things...

RNNs

- for **sequences** of data
  - $\rightarrow$ predictive text
  - $\rightarrow$ acoustic data
  - $\rightarrow$ speech data (e.g. translation)
  - $\rightarrow$ **time-series** data

- Recurrent Neural Network (RNNs)
  - $\rightarrow$ some sort of **memory** effect desired

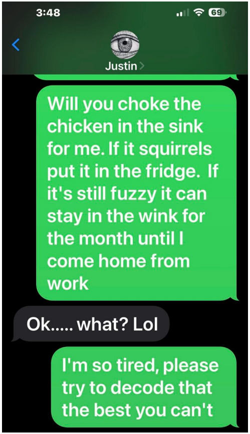

Figure: le wat?

RNNs

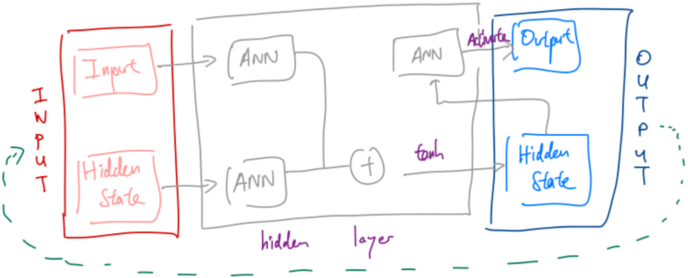

Figure: Demonstrative schematic of a RNN.

- introduction of a hidden state
  $\rightarrow$ part of the input/output
  $\rightarrow$ prediction depends on value(s) of hidden state
- for this schematic we have **three** neural networks blocks that are trained

RNNs: Lotka-Volterra

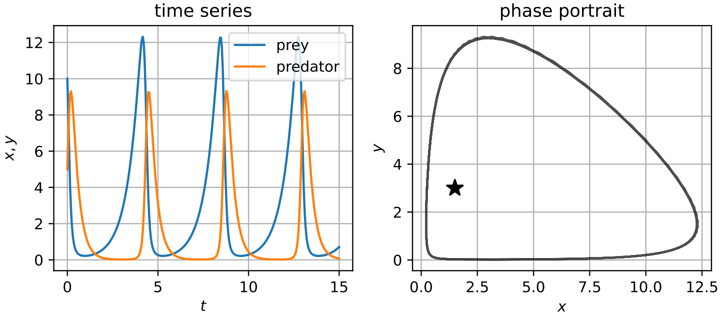

Figure: Time-series from Lotka-Volterra equation as time-series and **phase portrait**.

- artificially generate some data
  $\rightarrow$ Lotka-Volterra or predator-prey model here
- need to define inputs and outputs
  $\rightarrow$ inputs are **sequences** of two numbers (x, y)
  $\rightarrow$ outputs are two numbers (x, y)

RNNs: Lotka-Volterra

- use default RNN in keras

  $\rightarrow$ can make these more complex if you write them yourself...

```python
# keras wrap of a simple RNN (possibly stacked) with a linear layer output

def simple_rnn(input_size, output_size, seq_length=1, hidden_size=1, num_layers=1):

    # need to use the "Cell" variant to loop up
    rnn_cells = [layers.SimpleRNNCell(hidden_size) for _ in range(num_layers)]

    inputs = keras.Input(shape=(seq_length, input_size))
    x = layers.RNN(rnn_cells)(inputs)  # this is one block
    outputs = layers.Dense(output_size)(x)
    model = keras.Model(inputs, outputs, name="simple RNN")

    return model
```

Figure: Basic RNN done in keras.

RNNs: Lotka-Volterra

- train RNN to predict (x_{i+1}, y_{i+1}) from (x_i, y_i)
  $\rightarrow$ one-step prediction

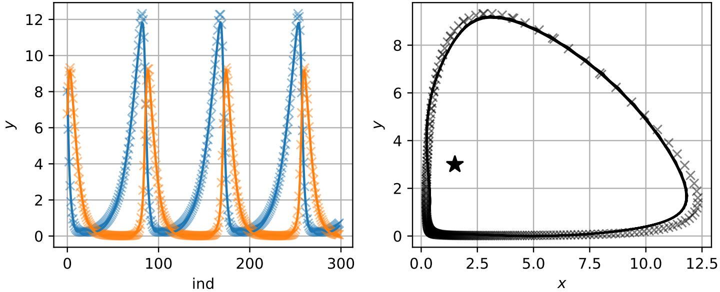

Figure: Easy one-step test. Actual data are markers, and RNN predictions are given as lines.

RNNs: Lotka-Volterra

- harder test: provide initial condition and keep applying the RNN
  $\rightarrow$ errors can (and will) accumulate
  $\rightarrow$ more relevant application, since that's what we want to use the RNN for in some ways

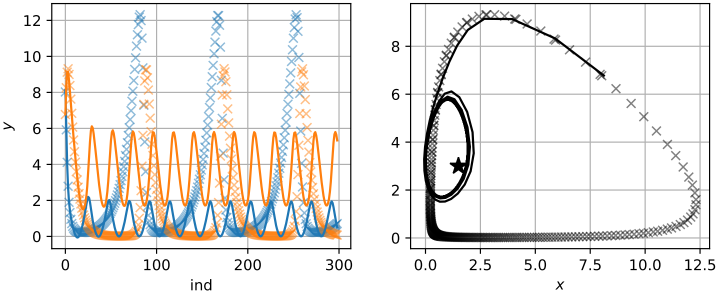

Figure: Harder sequential one-step test. Actual data are markers, and RNN predictions are given as lines.

RNNs: Lotka-Volterra

- example of an RNN doing insane things
  $\rightarrow$ **extinction** followed by **resurrection**!?
  $\rightarrow$ amplitude otherwise ok, phase errors though

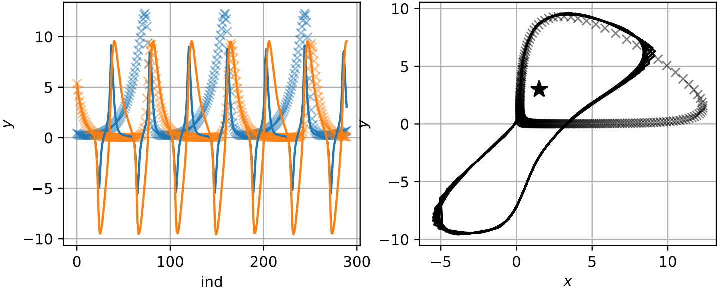

Figure: Harder sequential test with RNN trained on a sequence of ten time-steps. Actual data are markers, and RNN predictions are given as lines.

LSTMs and GRUs

- basic RNNs (as well as NNs with many layers) have known issues

  $\to$ notable one is the **exploding/vanishing gradient** issue
- suppose we represent mapping as $x_t = N(x_{t-1}, \theta)$

  $\to$ x_t is the input at time t

  $\to$ θ the model parameters

$$\Rightarrow \; \mathrm{d}x_t = \nabla_\theta N(x_{t-1}, \theta)\, \mathrm{d}\theta + \nabla_x N(x_{t-1}, \theta)\, \mathrm{d}x_{t-1}$$

$$\Rightarrow \; \mathrm{d}x_{t-1} = \nabla_\theta N(x_{t-2}, \theta)\, \mathrm{d}\theta + \nabla_x N(x_{t-2}, \theta)\, \mathrm{d}x_{t-2}$$

$$\Rightarrow \; [\nabla_\theta N(x_{t-1}, \theta) + \nabla_x N(x_{t-1}, \theta) \nabla_\theta N(x_{t-2}, \theta)]\, \mathrm{d}\theta$$

LSTMs and GRUs

$$\Rightarrow \left[\nabla_\theta N(x_{t-1},\theta) + \nabla_x N(x_{t-1},\theta)\nabla_\theta N(x_{t-2},\theta) + \ldots\right] \mathrm{d}\theta$$

- the more times you do this the longer the chain
- magnitude of these terms depend on N

  $\rightarrow$ if larger than 1, can blow up (model training crashes)

  $\rightarrow$ if smaller than 1, can 'stall' (doesn't really advance)

- vanishing gradient implies information further back in the chain smaller and contribute little to learning

  $\rightarrow$ not really learning the past in that case...

LSTMs and GRUs

- various proposed fixes, one is Long Short-term Memory
  (Hochreiter & Schmidhuber, 1995)
- consider the following example:

LSTMs and GRUs

- various proposed fixes, one is Long Short-term Memory (Hochreiter & Schmidhuber, 1995)
- consider the following example:

Julian, noted for his relentless questioning style, is not a welcome guest at the post-grad seminars.

→ `Julian` implies `his`, so you want to keep that information

→ `relentless` implies `not a welcome` probably

→ at `not a welcome` probably but don't need `Julian` anymore

LSTMs

- want to selectively ‘forget’ (ignore?) information
  → short chains (i.e. the **short-term** part)
  → in principle remember everything though (i.e. the **long** and **memory** part)

- introduce input, output and forget gates as a memory blow
  → each with its own recurrent part and NN (weights + bias) part
  → values of 0 to 1 (shut off to use everything, and in between)
  → ‘input’ decides what to remember
  → ‘output’ decides what to output (GRUs don’t have this)
  → ‘forget’ (or deactivate?) decides what information is ignored/deactivated

LSTMs

```python
# same game but for only for LSTMCell (GRUCell is done basically the same)

def simple_lstm(input_size, output_size, seq_length=1, hidden_size=1, num_layers=1):

    # need to use the "Cell" variant to loop up
    lstm_cells = [layers.LSTMCell(hidden_size) for _ in range(num_layers)]

    inputs = keras.Input(shape=(seq_length, input_size))
    x = layers.RNN(lstm_cells)(inputs)  # this is one block
    outputs = layers.Dense(output_size)(x)
    model = keras.Model(inputs, outputs, name="simple LSTM")

    return model
```


Figure: Basic LSTM done in keras.

- use default LSTM in keras
  $\rightarrow$ literally just swap out `RNNCell` with `LSTMCell`
  $\rightarrow$ leads to about a factor of four in degrees of freedom

LSTMs: Lotka-Volterra

- LSTM as applied to the Lotka-Volterra again
  $\to$ sequential one step prediction, “stalls”

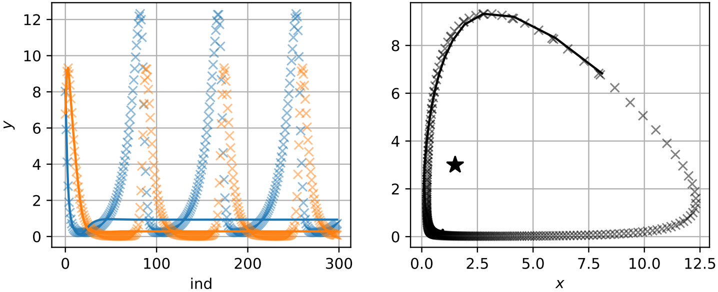

Figure: Harder sequential test with RNN trained on a sequence of one time-step (which slightly defeats the point of LSTMs...) Actual data are markers, and RNN predictions are given as lines.

LSTMs: Lotka-Volterra

- increasing sequence length gives periodicity back
  $\to$ phase lags are good in terms of ordering
  $\to$ amplitudes and actual phases are not great

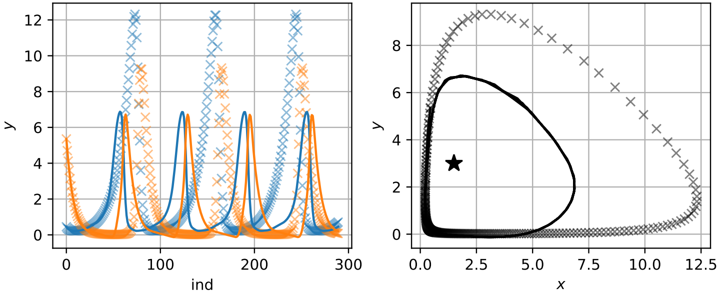

Figure: Harder sequential test with RNN trained on a sequence of ten time-steps. Actual data are markers, and RNN predictions are given as lines.

ConvLSTMs

- RNNs have dense layers within it, but no reason why we can't use **convolution** layers instead

- for predicting sequences of graphs/images, e.g.
  $\rightarrow$ video frames
  $\rightarrow$ satellite images
  $\rightarrow$ sequences of text in principle

- consider here learning form outputs of the KdV equation

$$\frac{\partial u}{\partial t} - 6u\frac{\partial u}{\partial x} + \frac{\partial^3 u}{\partial x^3} = 0,$$

where my data is going to be $u(x, t)$

ConvLSTMs: KdV

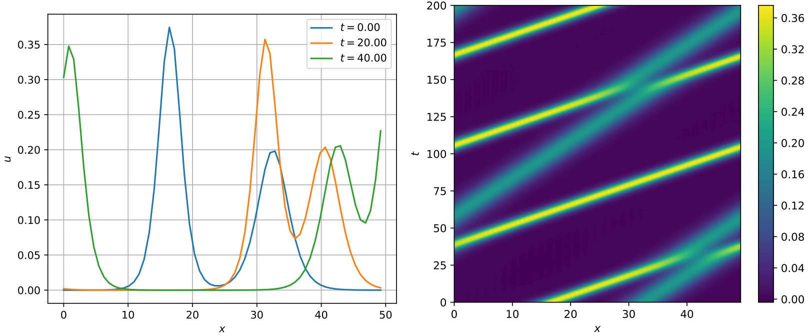

Figure: Solution from a numerical solve of the KdV equations with an approximate two-soliton solution initialisation.

- solutions of KdV have particular properties that you can look up yourselves

ConvLSTMs: KdV

```python
# block 1
x = layers.ConvLSTM1D(
    filters=16,
    kernel_size=5,
    padding="same",
    return_sequences=True,
    activation="relu",
)(inputs)
if batch_normalisation:
    x = layers.BatchNormalization()(x)


outputs = layers.Conv2D(filters=1,
                        kernel_size=(3, 3),
                        activation="sigmoid",
                        padding="same"
                       )(x)
model = keras.Model(inputs, outputs, name="simple ConvLSTM")

return model
```

Figure: A model implementation using ConvLSTM layers. Note the use of `ConvLSTM1D` layers (and various options in there) and `Conv2D` at the end.

ConvLSTMs: KdV

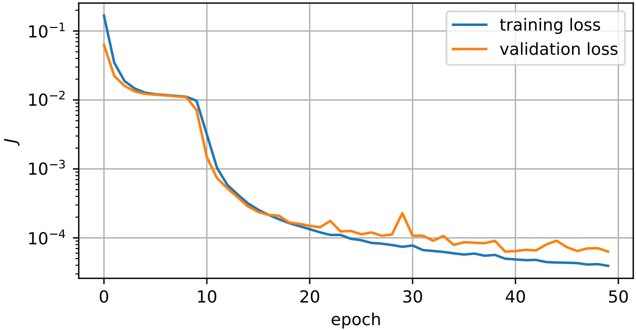

Figure: Training and validation loss from the ConvLSTM training. Note the axis is on a log scale here.

- be careful interpreting the magnitude of loss, because data has not been scaled to [0, 1] or similar

ConvLSTMs: KdV

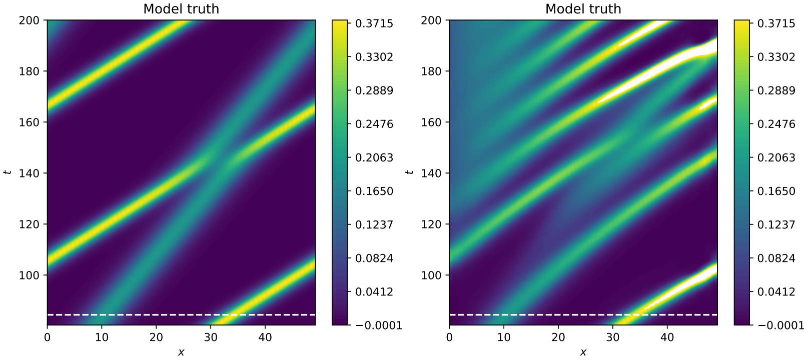

Figure: Hovmöller plot of (left) target data truth and (right) ConvLSTM sequential prediction for a model trained over ten time-steps. White dashed line denotes time of first prediction.

Demonstration

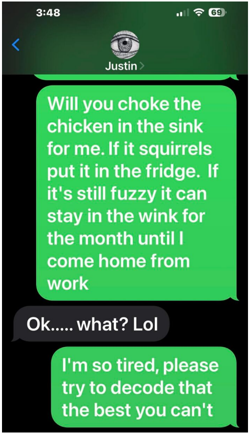

Figure: still le wat?

- RNNs, LSTMs and ConvLSTMs
  $\rightarrow$ longer sequences?
  $\rightarrow$ deeper + wider? data size?
  $\rightarrow$ experiment with **batch size**?
  $\rightarrow$ can **constrain** these? (see PINNs and maybe SINDy) later
- try it with some **text/satellite/simulation** data

Moving to a Jupyter notebook $\rightarrow$