February status update: Echo State Networks

Hector Morlet

22247737

March 7, 2025

1 Terms

Terms we've used in the past but to be clear:

- "ON" is ordinal network
- "ON ESN" refers to the layered ESN where each ordinal partition has its own layer.
- "Stochastic ON ESN" refers to the stochastic version of the "ON ESN".
- "Vanilla ESN" refers to an ESN without any of the layering nonsense

2 Error fixing

First I fixed a couple errors which we found in our last meeting, including: For the Stochastic ON ESN, I was masking all the connections between layers other than the single randomly chosen connection only for the active layer (i.e. the layer whose partition has just been seen). I fixed it so that it chooses a connected layer for each layer according to the transition probabilities of the corresponding partitions. For the ON ESN, there was a silly error where each node in each layer was connected to itself by a fixed amount. I removed these self-connections. But I might reintroduce them later to see what effect they have.

3 Stochastic ON ESN: Re-scaling the reservoir weights each time the different connections are masked.

Background: For each partition, the stochastic ON ESN chooses one other partition based on the transition probabilities. It then masks (sets to zero) all connections between layers, other than the chosen connections. The issue is that this means the reservoir weight matrix is constantly changing. Thus, the overall 'signal amplitude' in the network may be too low or too high. Of course, this is usually controlled by scaling the weights initially using the spectral radius.

Hypothesis: will rescaling the reservoir weight matrix using the spectral radius after the non-chosen weights are masked improve the prediction?

Result: calculating the maximum eigen value of the weight matrix each time step makes the ESN run *extremely* slow. So I don't have any results. I think I will try to overcome this by caching all the possible combinations of chosen partitions and calculating the associated maximum eigenvalue for each combination. Then it won't have to calculate it each time, and probably won't need too much more memory.

4 Testing the ON ESN using multi-step predictions.

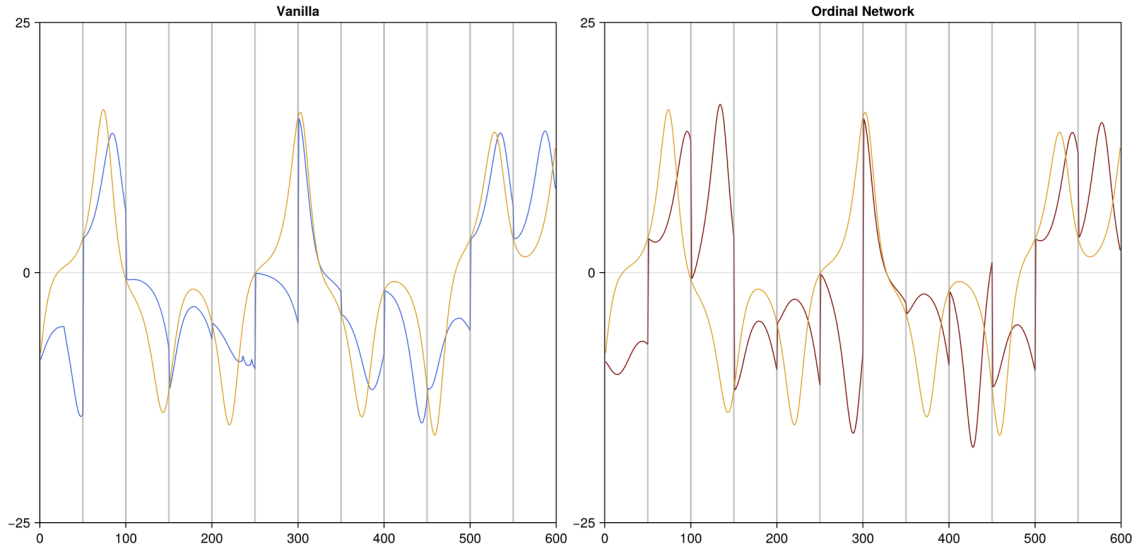
I can't remember why, but I stopped testing for multi-step predictions and I think my original method was producing wild results because it would feed in the partition of the true data alongside previous predictions (which would not match the partition). Now I have fixed it.

The way it now works is to run the ESN over the whole (true) time series and save the state at each time step. Then it splits the ESN into chunks and looks up the state of the ESN at the start of each chunk. Then it creates a prediction from that state and determins what the next partition is using that prediction and feeds them both back into the ESN to get the next state. It repeats this for the length of the chunk. Then the error is calculated from the predictions within each chunk over the whole series.

This is what it looks like:

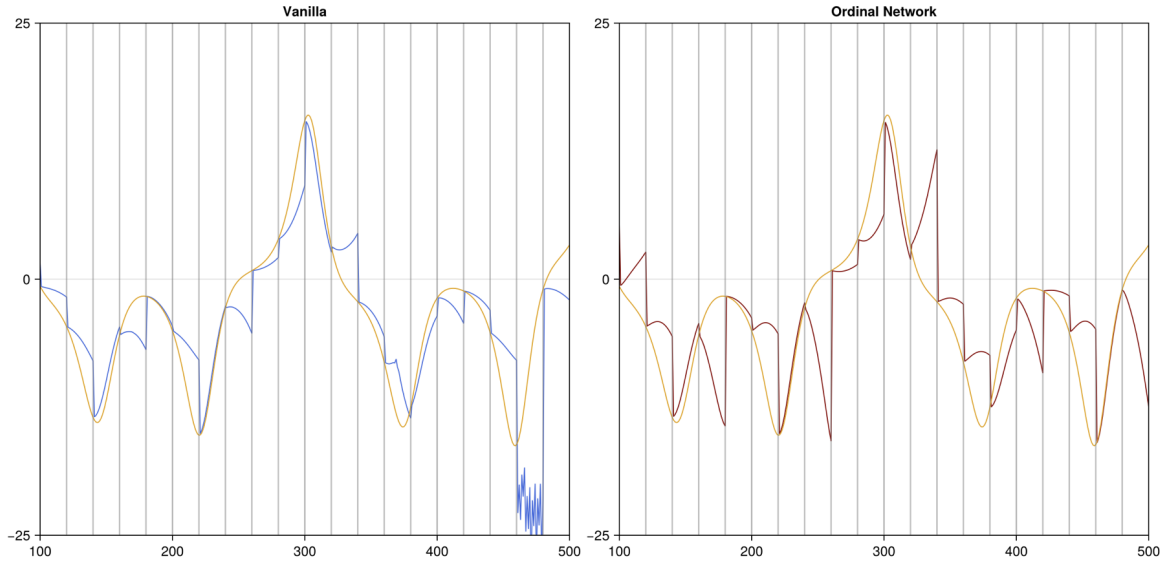
```
test_multi_step(lo_train, lo_test, 3, 100, n_steps=50, from=0, to=600; ignore_first=100)
```

```
Created reservoir of size: (600,)
Created reservoir of size: (600,)
Overall RMSE:
    Vanilla: 6.0286121479729395
    Ordinal network reservoir: 7.753602748653324
Turning partition RMSE:
    Vanilla: 4.8126326088754245
    Ordinal network reservoir: 8.895123913256526
```



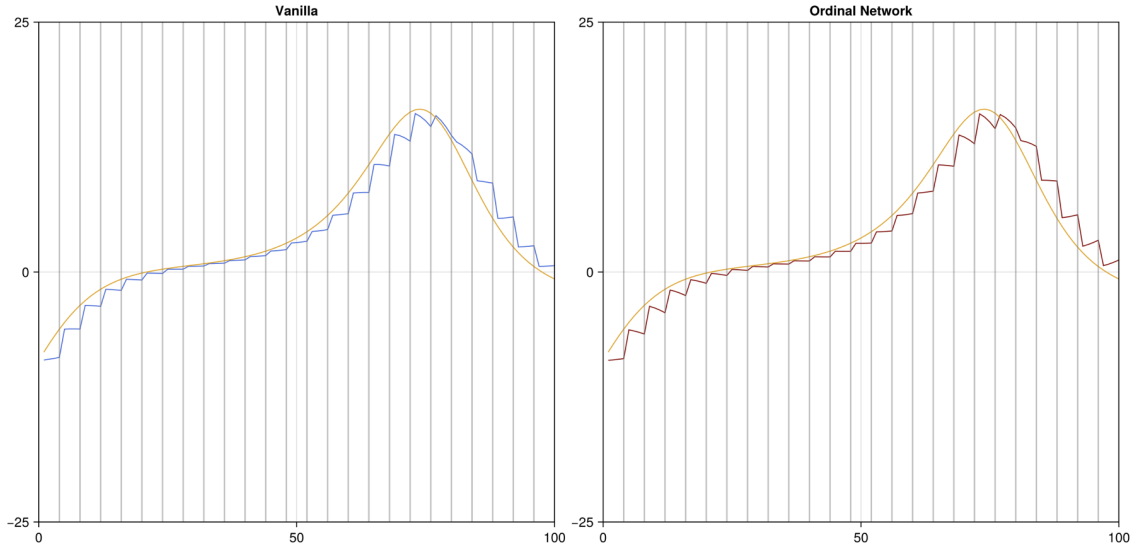

```
test_multi_step(lo_train, lo_test, 4, 100, n_steps=20, from=100, to=500)
```

```
Created reservoir of size: (600,)
Created reservoir of size: (600,)
Overall RMSE:
    Vanilla: 3.258012010720273
    Ordinal network reservoir: 4.82531572370914
Turning partition RMSE:
    Vanilla: 3.8929526111687207
    Ordinal network reservoir: 5.584167576502982
```



```
test_multi_step(lo_train, lo_test, 3, 400, n_steps=4, from=0, to=100)
```

```
Created reservoir of size: (2400,)
Created reservoir of size: (2400,)
Overall RMSE:
    Vanilla: 1.6844827824116342
    Ordinal network reservoir: 1.190564081384711
Turning partition RMSE:
    Vanilla: 1.6287624164192689
    Ordinal network reservoir: 0.6907764955186297
```

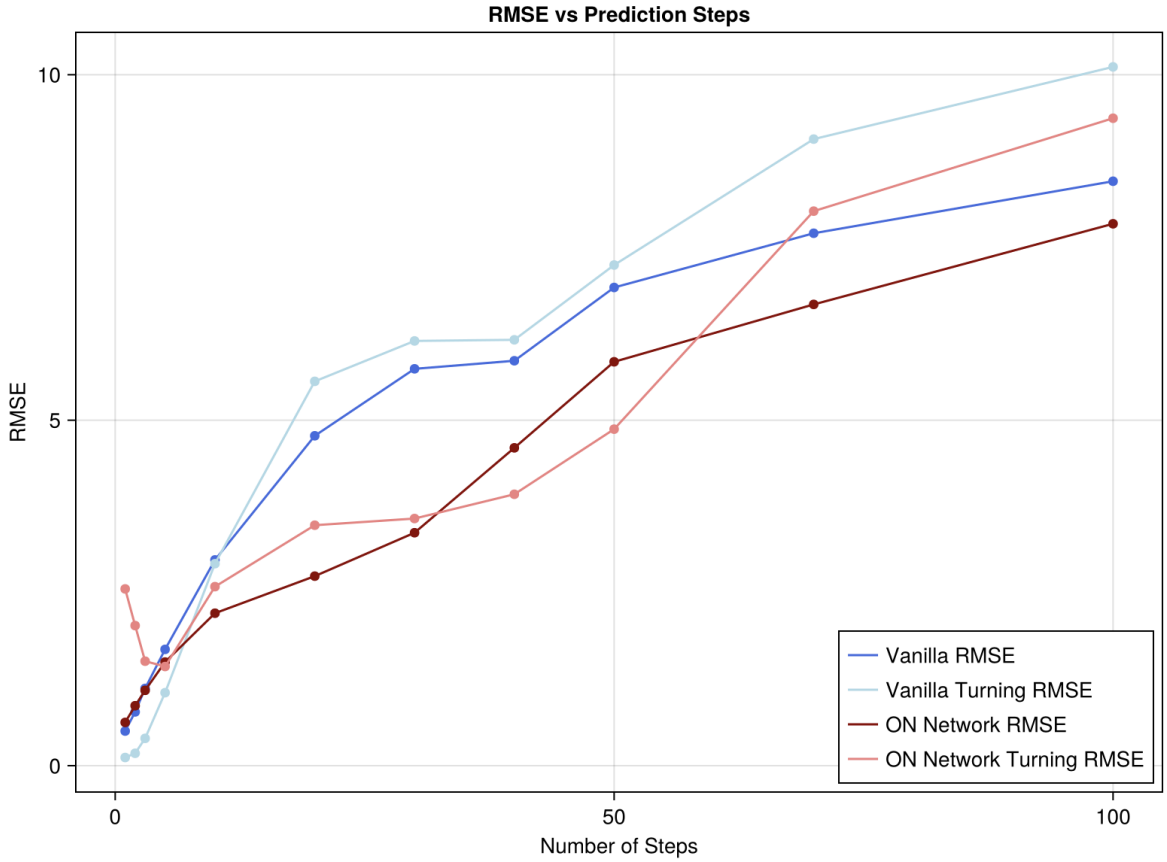


Since each run randomly initialises the ESNs, I decided to do a sample of 30 of these, calculate the RMSE for each sample and take the mean of these sample RMSEs. I feel like this is a bit suspect, and that maybe I should take the MSE over all predictions over all samples before taking the squareroot. What do you think?

Anyway then I did this sampling over different lengths of multi-step prediction to create these plots. For each plot, the total k of the ON ESN vs the vanilla ESN are equal.

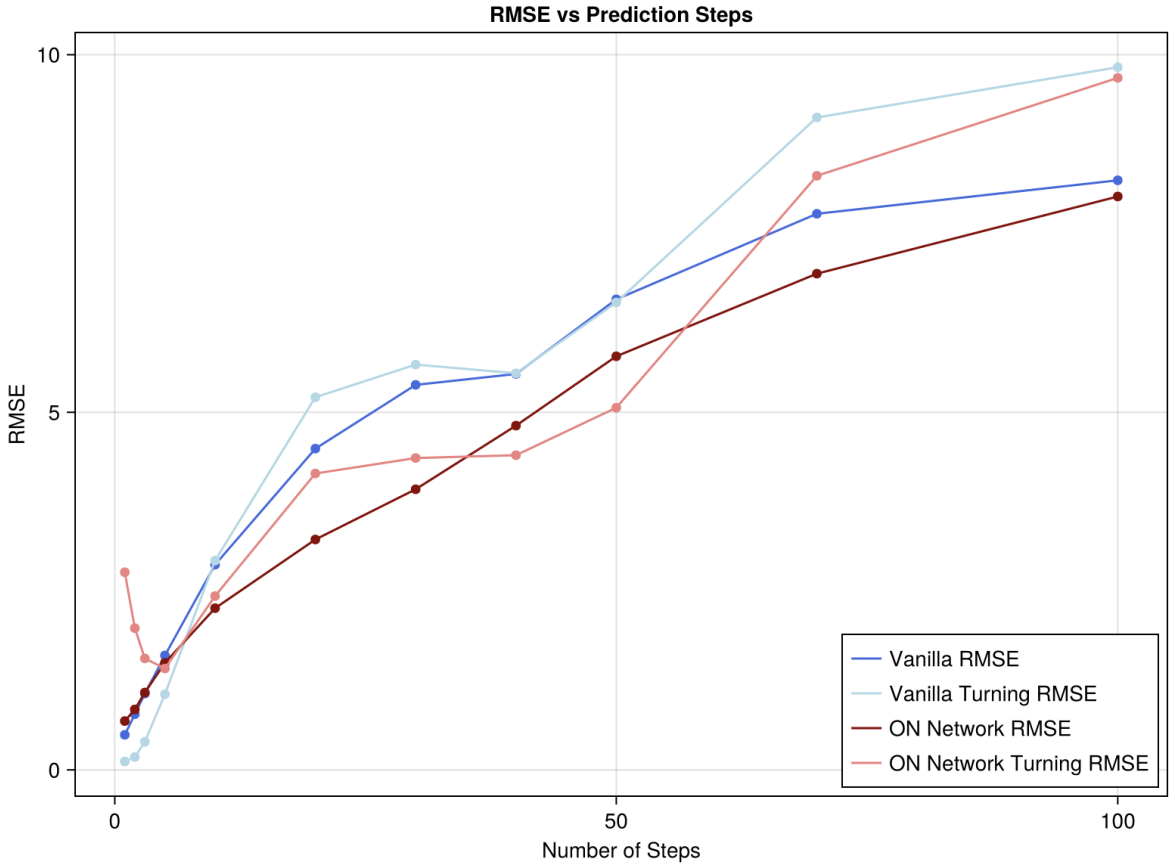
Vanilla ESN: total k = 300

ON ESN: layer k = 50, m = 3, number of partitions/layers = 6



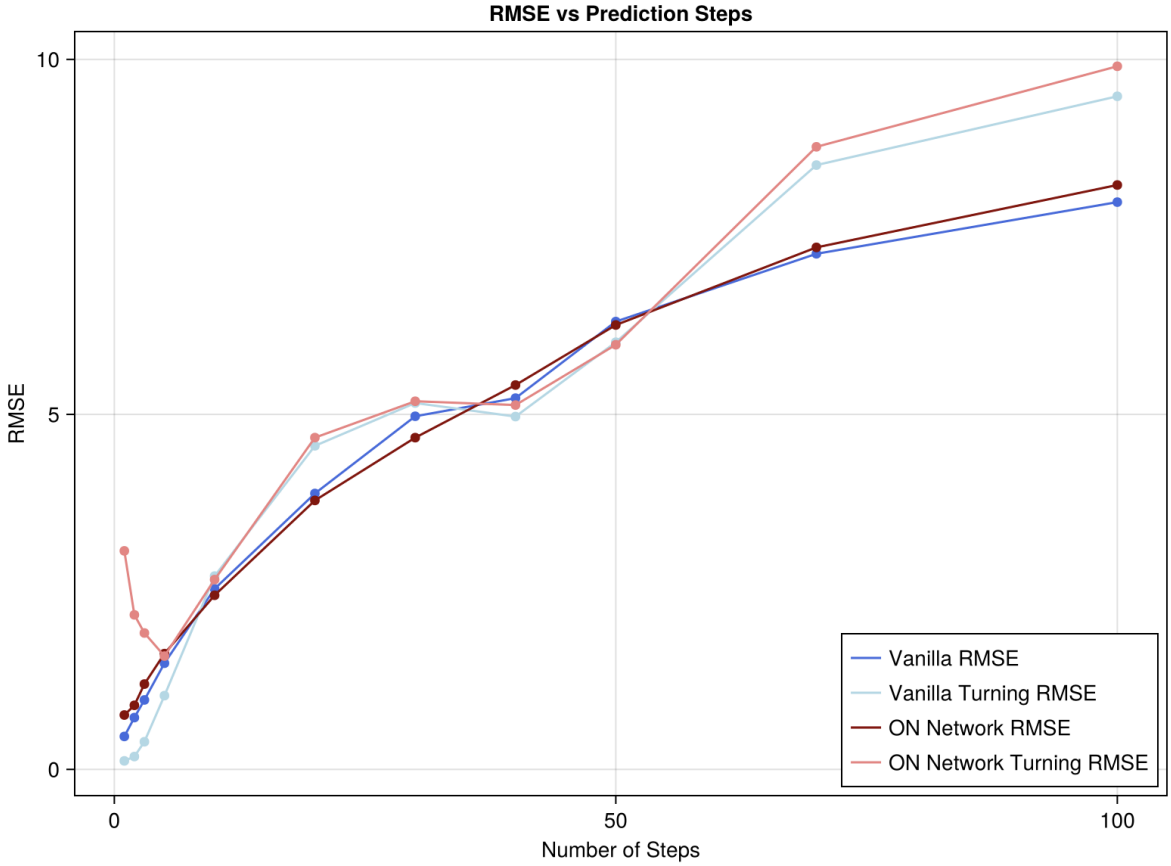
Vanilla ESN: total k = 600

ON ESN: layer k = 100, m = 3, number of partitions/layers = 6



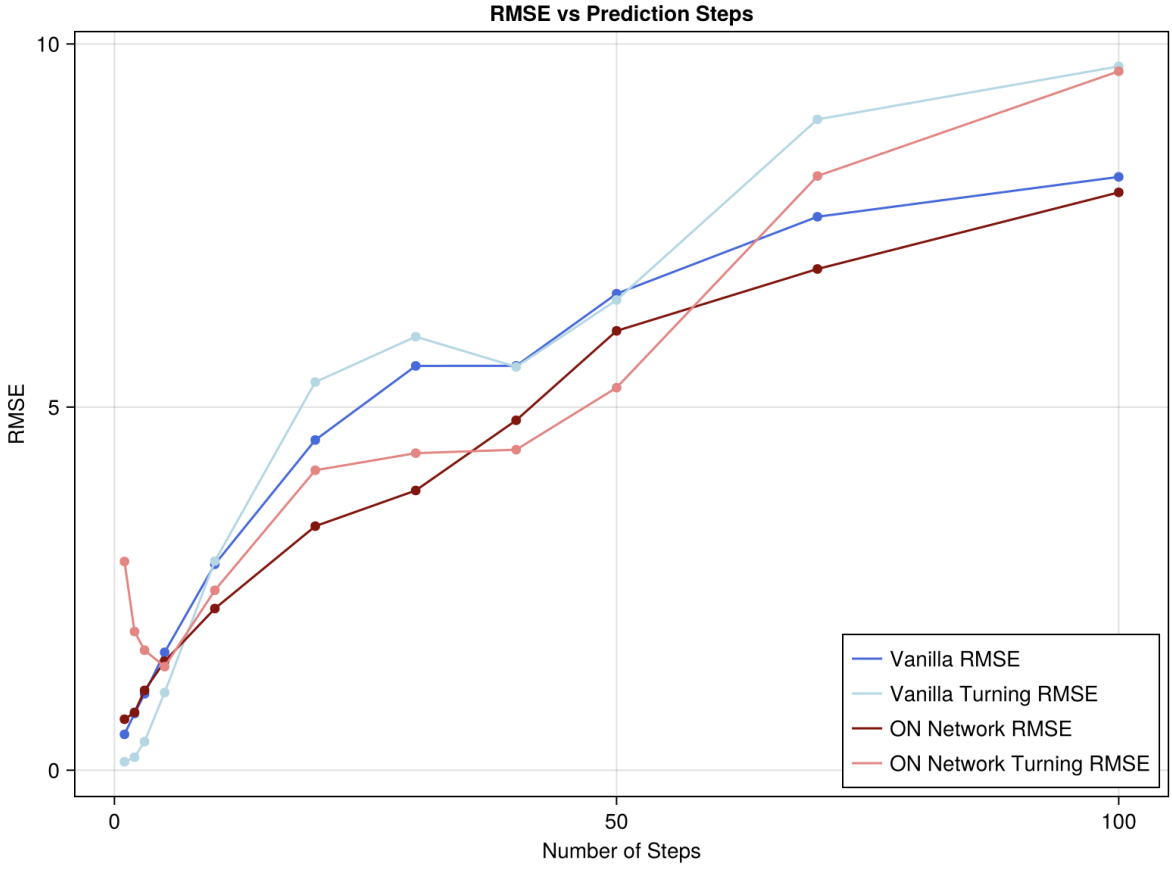
Vanilla ESN: total k = 1800

ON ESN: layer k = 300, m = 3, number of partitions/layers = 6



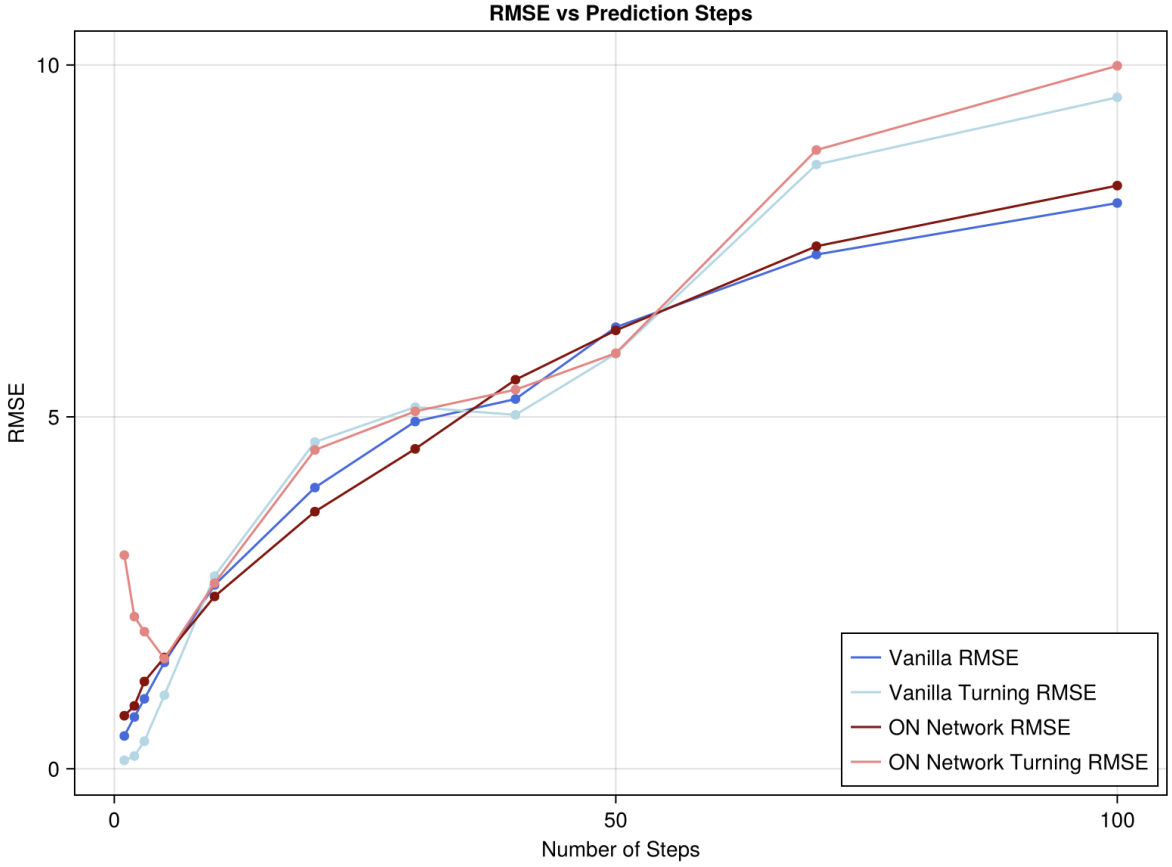
Vanilla ESN: total k = 1300

ON ESN: layer k = 100, m= 4, number of partitions/layers = 13



Vanilla ESN: total k = 3250

ON ESN: layer k = 250, m= 4, number of partitions/layers = 13



As you can see the ON ESN performs worse than the Vanilla ESN for no. steps predicted below 5 but appears to perform better above 5 step predicted. However, this effect seems to wash out for higher values of k (size of reservoir).

Note: I spent a while making a little testing framework to make it easier to parse parameters for comparison. Most of the code is now in scripts, with notebooks just to show the graphs.

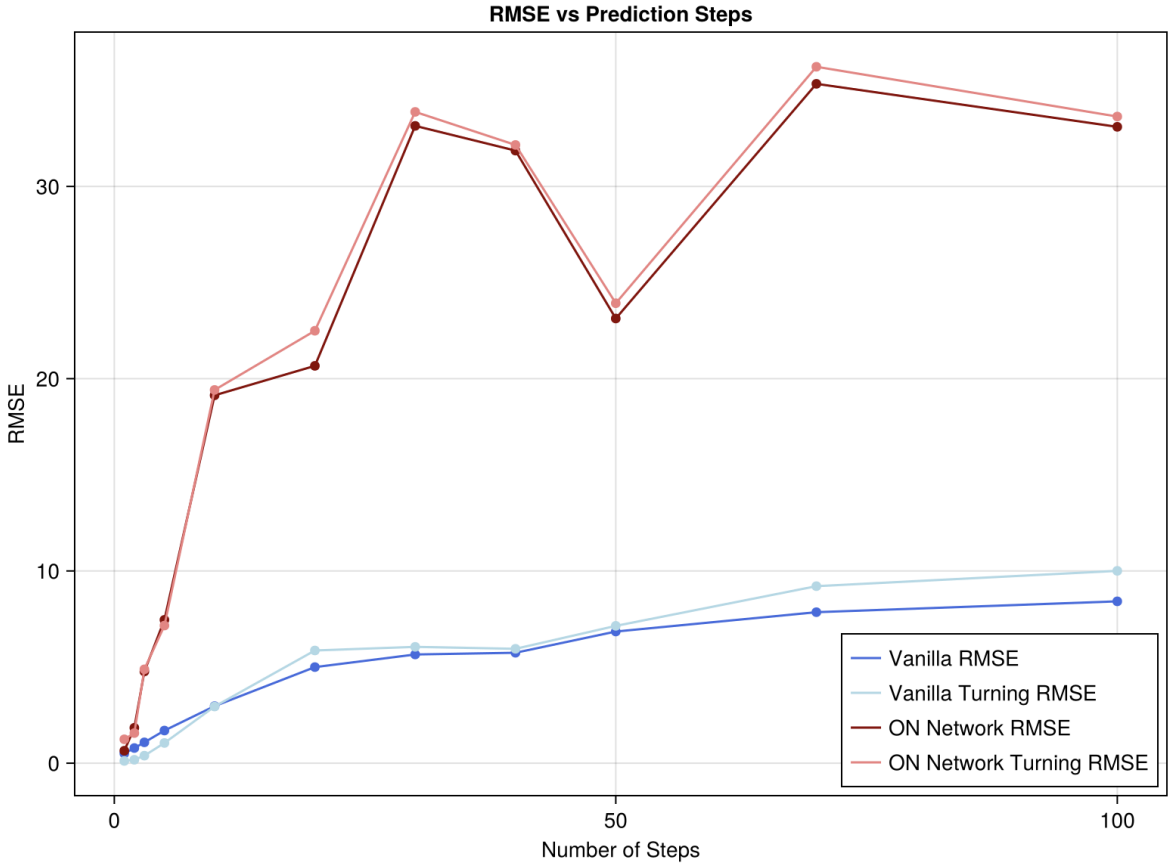
5 Readout Masking

Hypothesis: masking all layers other than the active layer (corresponding to the current ordinal partition) before performing the readout (i.e. just before doing predictions = states*R) will improve prediction.

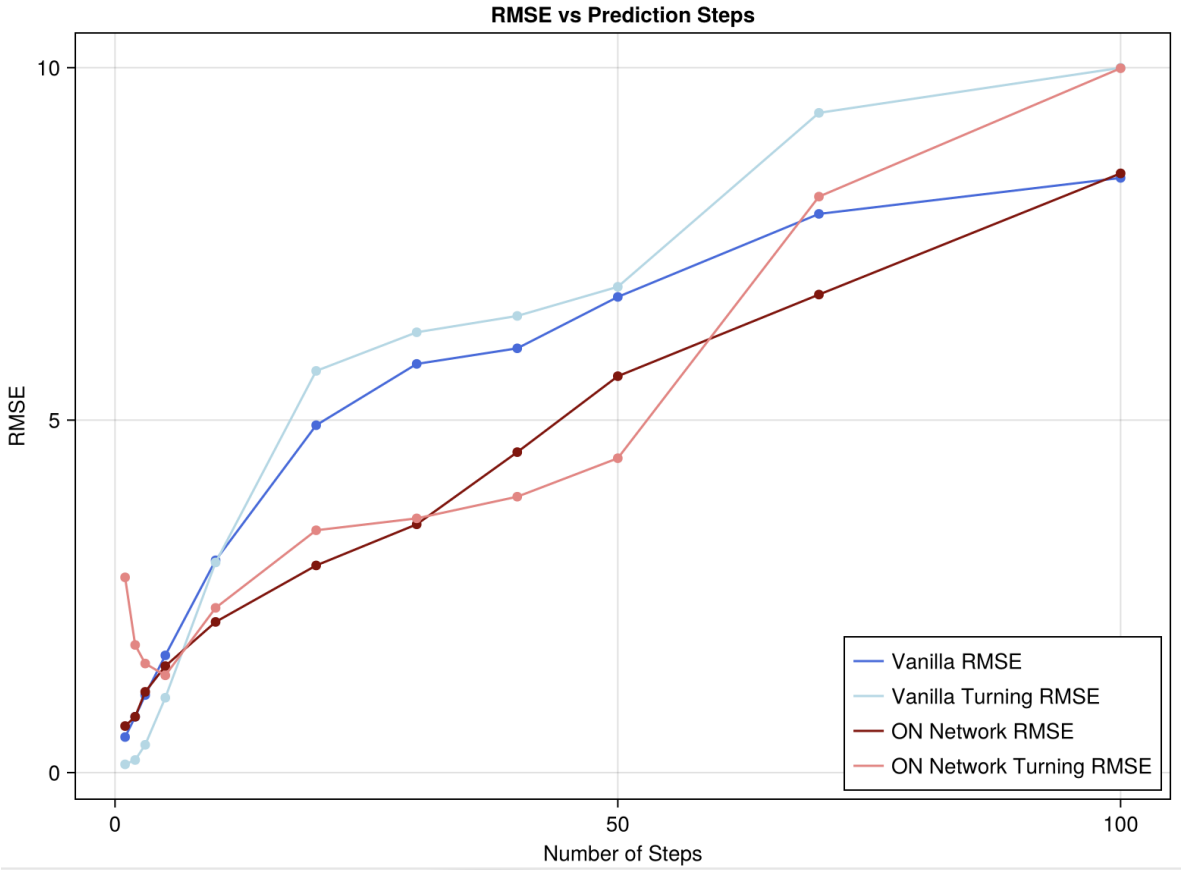
Findings:

Initially, it looked as though it didn't make a difference whether or not the readout was masked, based on single step predictions. I came to this opinion just from running the single step predictions a few times and seeing that the RMSE was approximately the same.

But once I implemented the multi-step predictions and the associated error charts, it looks as though readout masking makes the prediction worse for longer predictions. Here is the RMSE for the ON ESN **with** readout masking vs the vanilla ESN:



Meanwhile, here is the RMSE for the ON ESN **without** readout masking vs the vanilla ESN:



Interestingly, at a lower number of step (i.e. less than 5) the ON ESN **with** readout masking looks like it doesn't suffer from the high turning partition RMSE that the ON ESN **without** readout masking does. Not sure why that would be.

6 Stochastic ON ESN

Hypothesis: the predictions will improve if we stochastically choose one other layer for each layer to be connected to based on the transition probabilities of the ON, while masking all other connections.

Results:

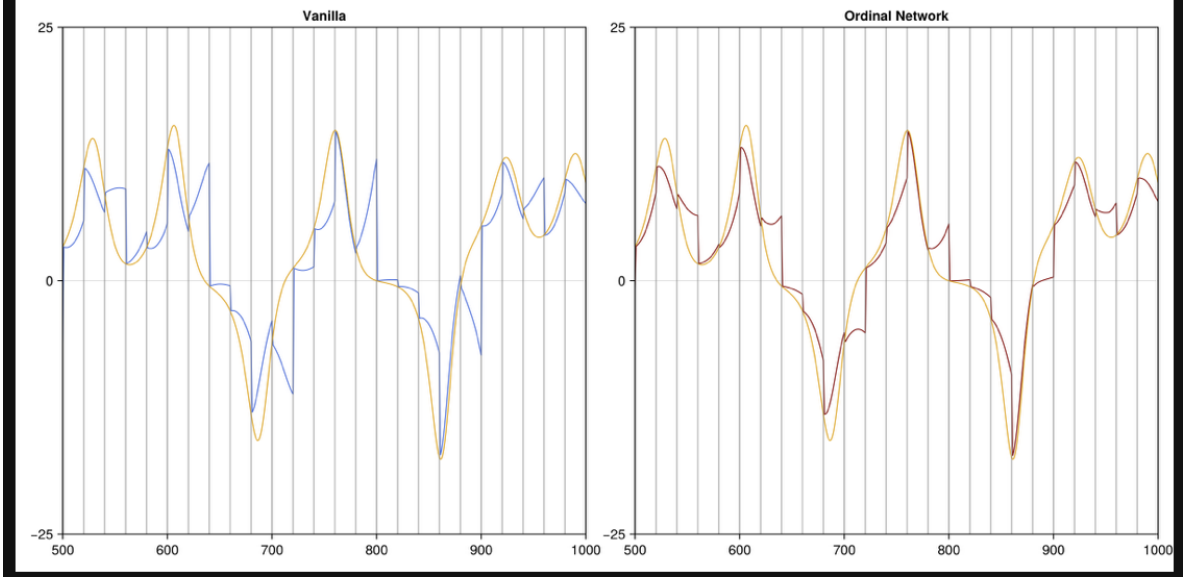
The stochastic ON ESN appears to perform well. In this case (see below) it had a lower RMSE.

Left: vanilla ESN with reservoir size 600

Right: Stochastic ON ESN with layer size 100 and 6 layers (given an m=3 to create

partitions) - total size 600

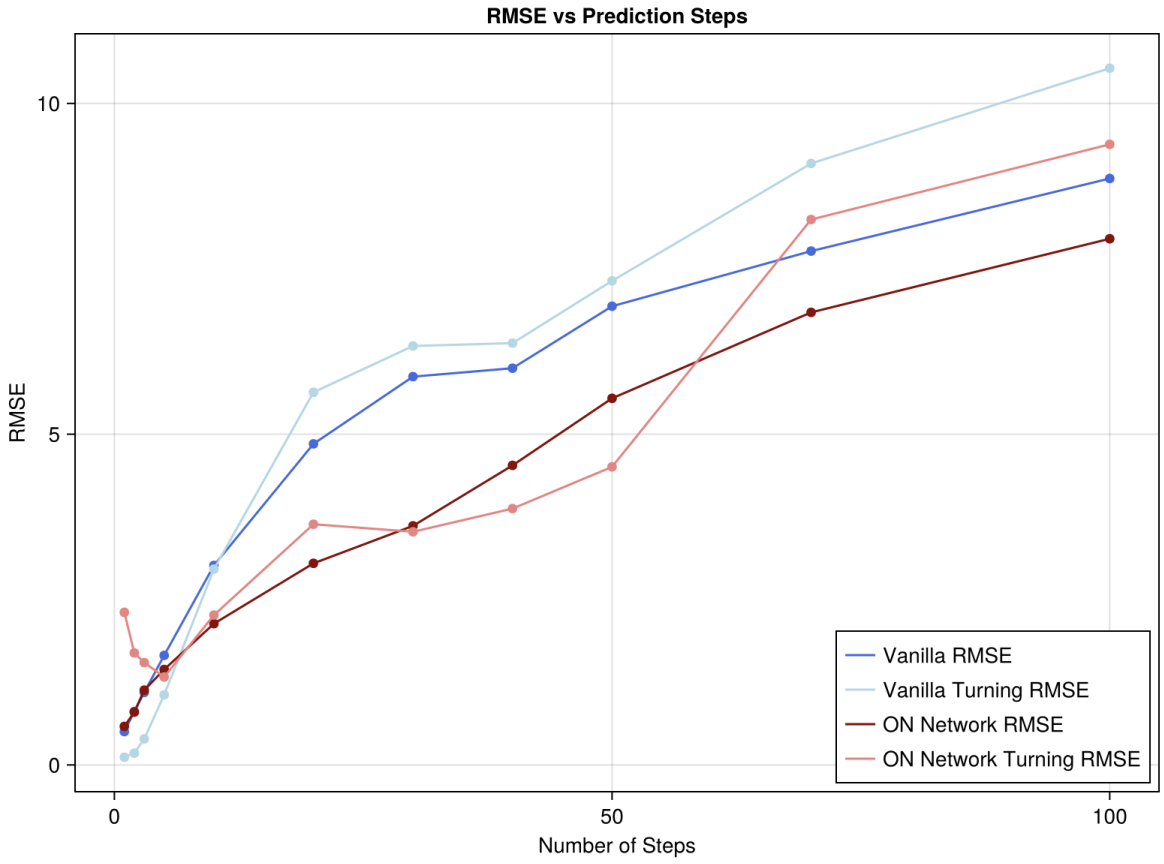
```
Overall RMSE:
    Vanilla: 4.134493761836858
    Ordinal network reservoir: 3.146646365221619
Turning partition RMSE:
    Vanilla: 4.774673211184325
    Ordinal network reservoir: 3.9945399173290523
```



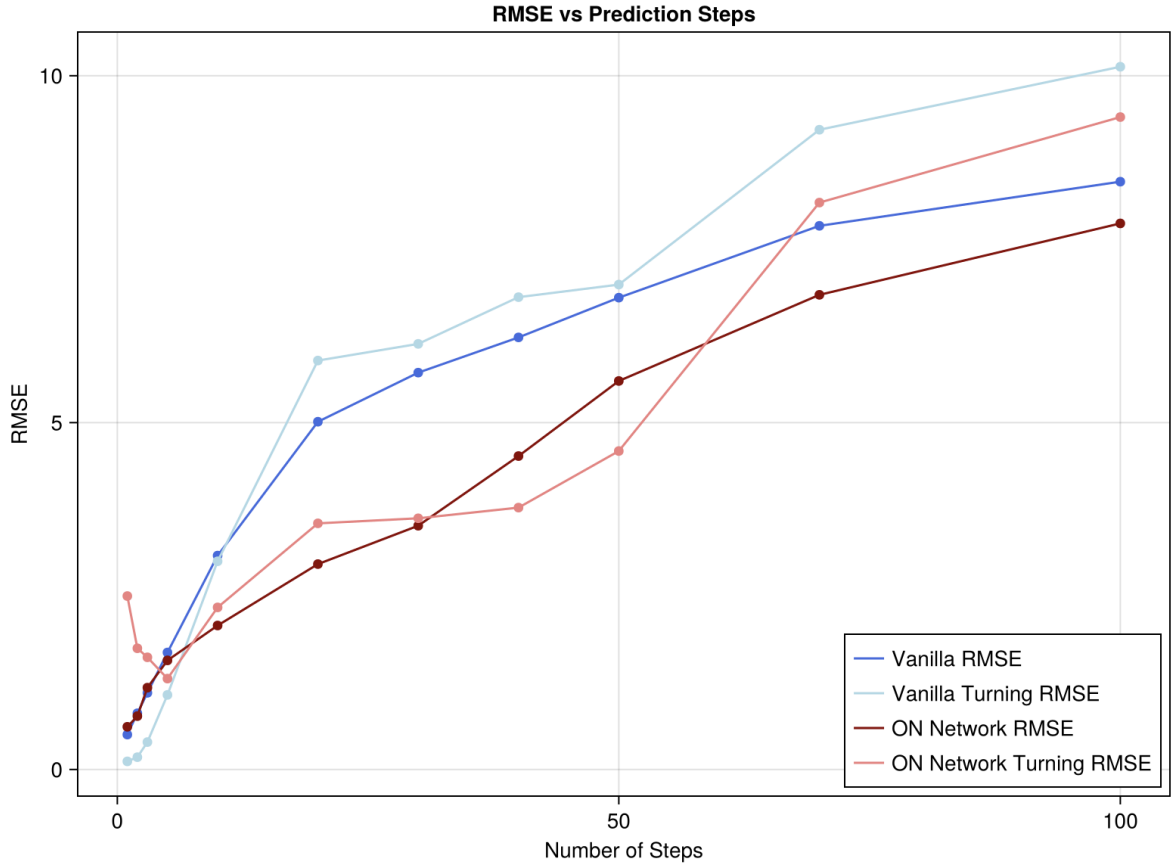
So far I have done one thorough test like those above. Here are the results so far for the stochastic ON ESN:

ON ESN: m = 3, layer k = 50, number of partitions/layers = 6

Vanilla ESN: k = 300



Compared to the non-stochastic ON ESN with the same settings:



I'm pretty shocked that the results are very similar to the non-stochastic ON ESN. I've checked thoroughly for errors but haven't found anything yet - maybe I will tomorrow lol. I'm almost expecting to. Otherwise it's pretty amazing that they'd be so similar. It could be explained by the fact that the time step for the Lorenz simulation is small at 0.01, so the stochastic switching of layer connections average out to be pretty similar to the deterministic layer connections. Perhaps I will compare the pair at a longer time step.

7 Next steps

I'm planning to:

- Make thorough multi-trial test for single step predictions, like I have for multi-step predictions.
- Keep testing the stochastic ON ESN and comparing it to the deterministic (regular) ON ESN.
- Thoroughly test the difference between:
  - scaling the layer connections according to the transition probabilities, vs
  - Having a constant value connection, vs
  - Randomising the connection, vs
  - Making it sparsely connected.
- Test adding self loops (low priority)

Looking forward to hearing your thoughts! As always you can check up on me here: https://github.com/HectorMorlet/echo-state-networks