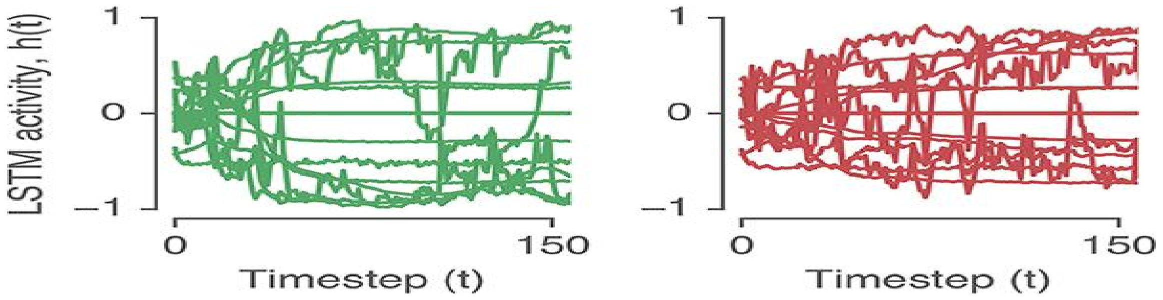
# Reverse engineering recurrent networks for sentiment classification reveals line attractor dynamics - Benjamin Cohen

## Personal interest :

- RNNs have been utilized in various applications such as speech recognition, sentiment analysis, music, and video. In neuroscience, RNNs have been used for modeling large-scale neural recordings and as a generator of scientific hypotheses by studying the network's learned representations. However, RNNs are generally viewed as black boxes. While there has been progress in understanding their operation on simple tasks, rigorously understanding how they solve complex tasks remains a significant challenge.In the industry the results of our algorithms lead to decision-making and having the power to **linearize** our problem help us explain with words and arguments why a decision is made and help us understand further how the network learns its decisions and decides.

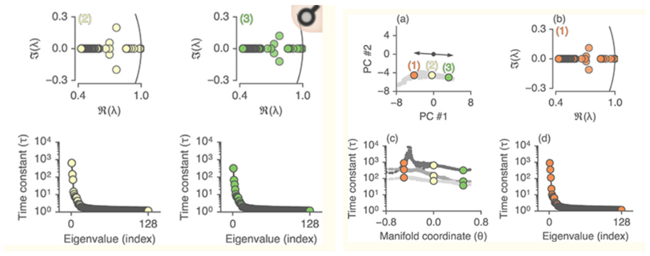


- As shown in this LSTM hidden state activity evolution (negative and positive documents/red and green) there are **no clear patterns** and it seems **unusable**. It seems to me that learning how to reverse engineer our RNN could be very helpful in gaining back **explainability** that we traded off against accuracy working with a black box. The combination of the **accuracy** of a RNN and the **explainability** of a linear system could lead to both excellent results and simple ways to explain how we got those results. To even get a better approximation, notice that we used a **Taylor** expansion to the first order, to get a more precise **approximation** of the variation of the vector state $\mathbf{h}_t$ we could take into account those other terms. But then we would lose the linearity so it would be a very different task.
- A very similar problem would be the very famous article **Opening the black box: low-dimensional dynamics in high-dimensional recurrent neural networks** which also tackled this need for explainability. Thus in this article we want to apply dynamical systems analysis to understand how RNNs solve sentiment analysis.

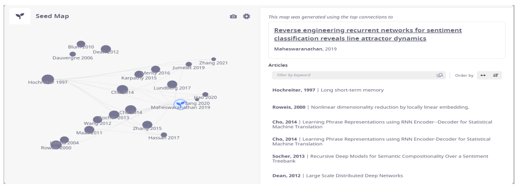
## Your understanding of the related works, and possibly, of what followed: did the paper have an impact; was it improved on later by the authors, or others ?:

- The purpose of the paper is that given a trained RNN, we would like to **reverse engineer** it to obtain a quantitative, interpretable description of how it solves a particular task. In order to do so we linearize around the fixed points of our trained network by taking the eigenvalues of the **Jacobian**.These fixed points are basically values of x where, if we let the system evolve from initial conditions, we will fall into these points. Generally we would expect the number of **eigenvalue eigenvector** pairs to be equal to **the number of dimensions of the system**.
- We describe the behavior of the perturbations around the fixed points in terms of partial derivatives. The **Jacobian** matrix describes our linear system. We can always describe the system's behavior as the decomposition of **exponential growth and decay** along eigenvectors of the **concatenated hidden states**. Instead of having growth or decay, complex eigenvalues are going to describe the frequency of oscillations around a fixed point.
- The top eigenmode across fixed points has a time constant on the order of hundreds to thousands of tokens. Each mode of the system either reduces to zero or diverges exponentially fast. Visualising the values and the spectra of the eigenvalues (most of them

are quickly null because they have fast time constants) we understand that **our RNN states only explore a low-dimensional subspace when performing sentiment analysis**.



- So basically, the important idea here is that RNNs are high-dimensional dynamical systems. They are systems that have a **state vector.**
- For nonlinear dynamics, we can find fixed points and **linearize the dynamics around them** to understand the computational role of those fixed points in solving a task for a RNN.
- The paper was published in June 2019 and did not get improved later by the authors but the article was cited in 24 articles according to its seed map. Some work on recognizing the patterns behind the brain function, others on the RNN black box aspect or others on the more specific domain of sentiment analysis.



*This map was generated using the top connections to*

Reverse engineering recurrent networks for sentiment classification reveals line attractor dynamics
**Maheswaranathan**, 2019

Articles

Filter by keyword | Order by

**Hochreiter, 1997** | Long short-term memory

**Roweis, 2000** | Nonlinear dimensionality reduction by locally linear embedding.

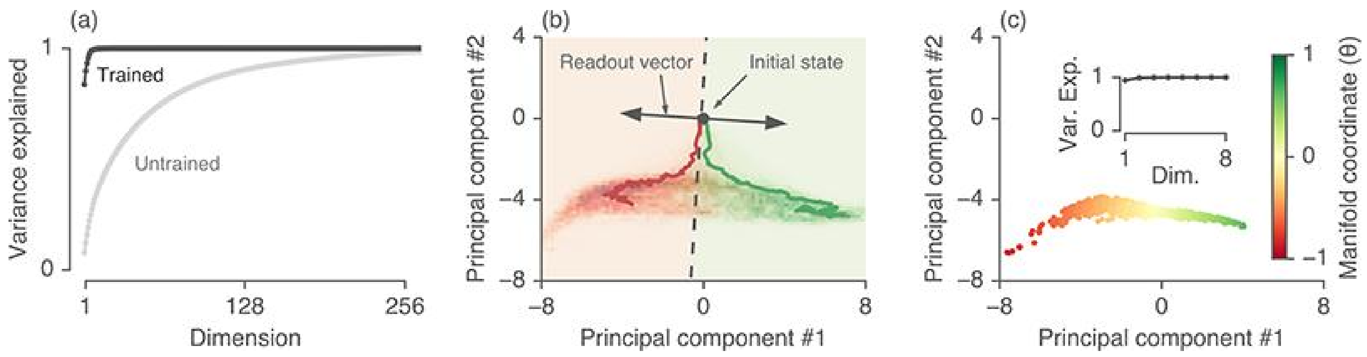
**Cho, 2014** | Learning Phrase Representations using RNN Encoder--Decoder for Statistical Machine Translation

**Cho, 2014** | Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation

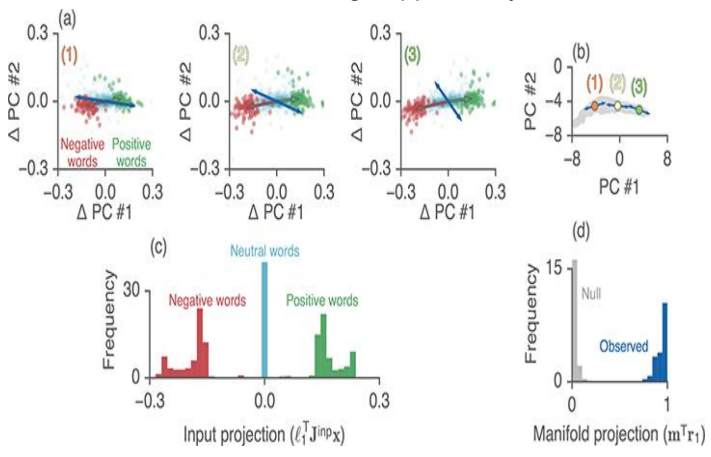
**Socher, 2013** | Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank

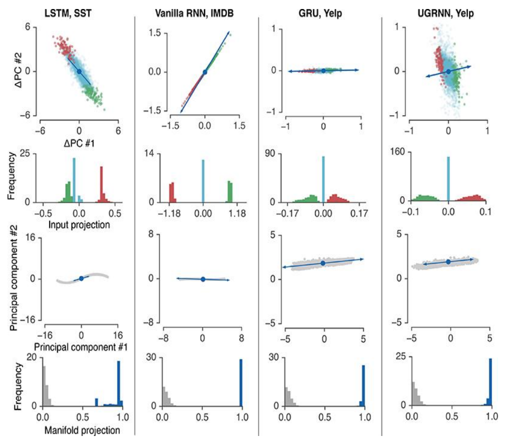
**Dean, 2012** | Large Scale Distributed Deep Networks

- **Reverse engineering recurrent neural networks with Jacobian switching linear dynamical systems** is another example of an article published very recently on 01/11/2021 that helped explain what we observe in our work which is a bigger error in the accumulated linear approximation of our RNN than even a simple linear model. They explain that one of the main difficulties is choosing which fixed point to expand around when studying RNN dynamics and computing error accumulation when reconstructing the nonlinear dynamics with the linearized dynamics. They combine ideas from reverse engineering RNNs and switching linear dynamical systems (**SLDS**) to address these challenges. **JSLDS RNN** can automatically learn to switch about a discrete number of fixed points mapping each point in the state space to a fixed point. **JSLDS** does not require post-training fixed point optimizations and that significantly reduces the linearized approximation error. Their observations provide evidence that **JSLDS** can regularize a nonlinear RNN towards solutions better described by switching between linearized dynamics around fixed points. This regularization towards a switching linear structure could potentially have beneficial performance effects for robustness and generalization on the test data.
- **An highlight and explanation of the main contribution(s) of the paper:**
- Many studies tried to understand RNNs by visualizing the activity of individual RNN units and memory gates during NLP tasks. Unlike them, this work aims to interpret the whole hidden state to **infer computational mechanisms underlying trained RNNs**.
- The nice thing about linear systems is that they can be described using matrix multiplication and linear algebra more generally. Despite their theoretical capacity to implement complex, high-dimensional computations, using analysis techniques from **dynamical systems theory**, we find that trained networks converge to highly interpretable, low-dimensional representations. Overall, these results demonstrate that surprisingly universal and human interpretable computations can arise across a range of recurrent networks (LSTMs, GRUs, and vanilla RNNs).These results demonstrate how a very simple operation such as **linear integration** arises as a universal mechanism in various nonlinear recurrent architectures that solve a real world task.



- By using methods such as **PCA** on the concatenated hidden states of the LSTM we observe that the variance (Fig.a) is explained by a low dimensional setting indicating to us that the dynamics of RNNs in resolving the task of sentiment classification of Yelp reviews are low dimensional too. It is shown Fig.c, we have only a **1-D manifold** to describe most of the variance of the data. Indeed by projecting hidden states (Fig.b) onto the two first principal components we get to see how our decision is made (positive or negative).These projections onto this 2D space are clearly separated in two halves as we would like it to work. With the paths of the example sentences we clearly see the link between this way of choosing and how the brain works by creating its own decision with each word enumerated. To make a choice we are integrating evidence along this line. We would like to find something alike in our **RNN dynamics**.
- The RNN state vectors are pushed towards the aspect of each word of the sentences (positive,negative,neutral). They evolve along the **1D-manifold** of the stable fixed points. In the absence of input the RNN state should rapidly converge to the closest fixed point and then should not change appreciably : it is the **line attractor hypothesis**.





- Here we may observe that experimentally by visualising the effect of different word inputs on the LSTM state vector. Fig 5.a) shows us the **1-D manifold** as we have the same arrow pointing towards the negative on one side and the positive words on the other. Choosing for a whole sentence then could be summed up as walking over this 1D-line and picking the closest center of mass of each region. Fig 5.b) tells us that we indeed have the right words in the right regions (bad,neutral,positive) only by using the coordinate of our point on the **1-D manifold**.
- Fig.6) reveals to us the **universal mechanisms** among all the RNN architectures. The different networks converge toward the same form of decision making (choosing on the **1-D manifold)**. Notice that the paper also shows that networks trained on other datasets used for sentiment classification also have those **universal mechanisms**.

## Let’s see how to linearize :

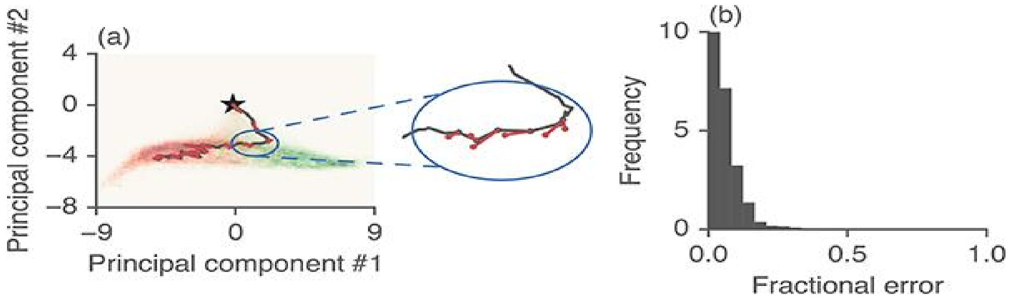
Given the last state ht−1 and the current input xt, the approach is to **locally approximate the update rule with a first-order Taylor expansion**. After simplification we get the simple formula for the variation of ht : $\Delta \mathbf{h}_t = \mathbf{J}^{\text{rec}} \Delta \mathbf{h}_{t-1} + \mathbf{J}^{\text{inp}} \mathbf{x}_t .$
with both jacobians depending on the chosen fixed point to linearize around.
(Jinp xt) is the instantaneous linear effect of xt on the RNN state vector.
That’s how we modify our RNN to create a linear system approximating its dynamics. With this linear aspect we may compute the variations in the hidden states with T inputs (words here) with the sum of each variation.

$$\mathbf{R}\Lambda^k \mathbf{L} \mathbf{J}^{\text{inp}} \mathbf{x}_t = \sum_{a=1}^{N} \lambda_a^k \mathbf{r}_a \boldsymbol{\ell}_a^\top \mathbf{J}^{\text{inp}} \mathbf{x}_t ,$$

Total effect of xt on the final RNN state

- **Visualising the differences between our linearized model and the actual RNN :**

**Full nonlinear system in black and the linear system in red**



We see that regardless of the architecture or dataset, **each network** approximately solves the task using the **same mechanism**. Now comparing the overall classification accuracy of the Jacobian linearized LSTM with the full nonlinear LSTM, we observe that the linearized version is much worse, presumably due to small errors in the linear approximation that accrue as the network processes a document. Whereas simply training a linear model gives us almost the same performance as the LSTM. This shows us that it is accumulating the **errors in the approximation** that make those problems of accuracy rise. Accumulating errors in the linearized LSTM cause trajectories to diverge so we get a glimpse of why **we cannot actually replace the full nonlinear LSTM with a unique linear system**.

We may have worked on NLP tasks, sentiment analysis more specifically which are not the most general exercises. But still, this way of choosing between good,bad,neutral for a sequence could be generalized to discover the underlying mechanism for choosing between categories by walking on a manifold with each input in the corresponding direction (+1 ; -1 , 0) and understanding the line attractor dynamics.

**To conclude** the hypothesis that RNNs approximate line attractor dynamics makes four specific predictions:

1. The fixed points form an approximately **1D manifold** that is aligned with the readout weights.
2. All fixed points are attractive and stable. This means that in the absence of input or null input, incoherent input, **the RNN state should rapidly converge to the closest fixed point** and then should not change appreciably.
3. Locally around each fixed point, inputs representing positive vs. negative evidence should produce **linearly separable effects** on the RNN state vector along some dimension.
4. These **instantaneous effects** should be integrated by the recurrent dynamics along the direction of the 1D fixed point manifold.

**In the end, these experiments helped us demonstrate that amazingly we can find a human and interpretable way of computing in some tasks solved across a wide range of RNNs. We have also shown that RNN architectures converge to a low-dimensional representation for the task of sentiment classification, even though they are able to implement high dimensional and nonlinear computations. So we may now quantitatively understand how a RNN solves the sentiment analysis task by interpreting its linear approximation. By understanding the linear dynamics in a general RNN solving certain tasks, we can gain back the explainability lost by those black boxes while keeping its accuracy!**