# Reverse engineering RNN for sentiment classification reveals line attractor dynamics ... and a slingshot?

**Alexis Tremblay**
MILA
Université de Montréal
atremblay@umontreal.ca

## Abstract

Recurrent Neural Networks (RNN) are widely used to model sequential data such as time-series or texts. The can be used for regression (e.g. predicting stock market price) or classification (e.g. stock market will increase or decrease).In the original work, the authors explore the dynamic of a trained single-layer RNN for binary sentiment classification (positive versus negative). The authors used tools from dynamical system to find fixed points and linearize the nonlinear system around them to reveal line attractors with the RNN. In this work, I try to reproduce some of the results and improve on studing a 2-layer Gated Recurrent Unit trained on the movie review dataset IMDB. We will see that the lower layer is doing grunt work, building the internal representation, and letting the top layer quickly make the decision if a text is positive or negative.

## 1 Introduction

The original paper explored 4 types of RNN architectures: LSTM, GRU, Update Gate RNN and the original flavour of vanilla RNN. Each were trained on three binary sentiment classification dataset: IMDB, polarized movie reviews, Yelp, user reviews, and Stanfort Sentiment Treebank, also movie reviews. In each case they trained a single layer. I reached out to the main authors to get their code base, but never got any answer back. So the exact details of the neural architectures are unknown. Did they use dropout? Did they limit the vocabulary in any way? Did they do preprocessing on the texts? There are many details I would have liked to have in order to reproduce as closely as possible their results, but alas I relied on my own experience and what I assume they did.

To answer the call of the reproducibility challenge, lauched by Joelle Pineau for NeurIPS 2019, I coded everything from scratch with what little implementation details are available in order to recreate some of the results. If we give ourselves some leaway I believe we can come to the conclusion that the original results are reproducible.

I focused my efforts on using a GRU trained on the IMDB dataset. My main exploration for this project was to see what kind of dynamic can be observed if we train a two layer GRU instead of just one.

## 2 Methods

### 2.1 Preliminaries

The n-th RNN cell, GRU in all cases in this work, is denoted as $\mathbf{h}^n_{t+1} = F^n(\mathbf{h}^n_t, \mathbf{x}^n_t)$ where $\mathbf{x}_t$ is the input and $\mathbf{h}_t$ is the hidden state at time $t$. Note that during training $\mathbf{x}^n_t = \mathbf{h}^{n-1}_t$ for $n > 1$.

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

### 2.2 Training

A very simple, but efficient, model was built. Embedding layer (100 dimensions) -> Dropout(0.5) -> n layers of GRU (256 dimensions)-> Dropout(0.5) -> Fully Connected. The Adam optimizer [1] with default parameters was used and the loss function is binary cross entropy $Loss = -\frac{1}{N}\sum_{n=1}^{N} y \log(\hat{y}) + (1-y)\log(1-\hat{y})$.

Typical training procedure for machine learning is applied. The dataset is split in train-validation-test sets. Each models are trained, on the train set, for 10 epochs and the parameters giving the lowest loss on the validation are kept for the optimal model.All the analysis in this work are done with the test set only, so the model has never seen those datapoints and no human leakage is seeping through the choice of models. This typically the best, and standard, procedure to get a machine learning model that generalizes as best as possible. I will briefly talk, in the futur direction section (5), how we should perhaps get out of this accepted pattern.

There were no indications in the original paper if all the weights in the networks were learned from scratch or not. Since it is standard practice to use pretrained word embedding I chose to kickstart the learning process with GloVe [4] words vector of 100 dimensions as initial embeddings. Those vectors were trained on a corpus of 6 billion words. "The training objective of GloVe is to learn word vectors such that their dot product equals the logarithm of the words' probability of co-occurrence."[2]. Word2Vec [2] was also considered, but given that they are often on par and that GloVe was readily available in the library I used I decided to take the path of least resistance. FastText [3] is also another common choice and quite often seen as a better quality word embedding. Studying the impact of word embeddings is left for futur direction. My educated opinion is that it wouldn't change the observations made here.

Since the author found that their finding is valid for all their dataset I chose to focus on only IMDB movie review dataset. This standard dataset is already split in train-test sets. The train set was subsequently split in train-valid set for the training procedure.

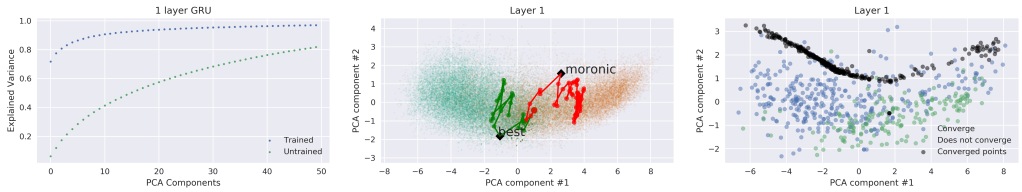
### 2.3 Fixed Point

Fixed points are found empirically by minimizing $q^n = \frac{1}{N}\|\mathbf{h}^n - F^n(\mathbf{h}^n, \mathbf{0})\|_2^2$ with respect to the hidden state so we end up at $\mathbf{h}_*^n = F^n(\mathbf{h}_*^n, \mathbf{0})$ where $\mathbf{h}_*$ is a hidden state that does not change much under the dynamic of a zero input ($\mathbf{x} = \mathbf{0}$). This is done by using standard differentiation techniques. If $q$ is less than $1^{-6}$ then we can consider the hidden state as a fixed point or quasi fixed point. The starting points are 500 hidden states, picked at random, explored by the GRU on the test set (after training). This is done on every layers.
Some technical difficulties arised. It appears that this cannot be done in bulk but rather one starting point at a time. Unclear if running the optimization in batch create degenerate cases. If the convergeance of two hidden states bring them to less than $1^{-3}$ in every dimensions then they are considered to have converged to the same point. This is based on the default value of the original fixed point finder code base [3].

## 3 Results

Discussion about reproducability of the original paper's result will be threaded in the rest of this report. It's a mix of hit-and-miss, but generally everything holds.

[1] `https://pytorch.org/docs/stable/optim.html#torch.optim.Adam`
[2] `https://nlp.stanford.edu/projects/glove/`
[3] `https://github.com/mattgolub/fixed-point-finder`




(a) (b) (c)

Figure 1: (1a) Explained Variance for a trained 1 layer GRU on IMDB versus untrained. The first few dimensions rapidly explain 90% of the variance. Only the first 50 dimensions are shown. (1b) First 2 PCA components on the 1 layer model colored by labels. The evolution of the hidden state of a positive and negative examples are overlayed. (1c) 500 hidden states sampled and updated until convergeance as described in (2.3).

### 3.1 Dynamics are low dimensional

Even though the hidden states of the GRU cells are 256 dimensions, most of the action, once the model is trained, is happening in much less dimensions.

#### 3.1.1 PCA

In order to observe the low dimensionality of the dynamics, a PCA is trained on all the hidden states visited by the model on the test set. A 1-layer GRU trained on the polarity task quickly learns to embed all the relevant information in very few dimensions, as can be seen in (1a). It takes only the first few PCA components to explain 80-90% of the variance. An untrained GRU cell contains, as expected, very little information and requires many more dimensions to reach the same level of explained variance. We will see later how having only 1 layer forces the network's hand. This result is somewhat similar to the original paper ([1] - figure 12).

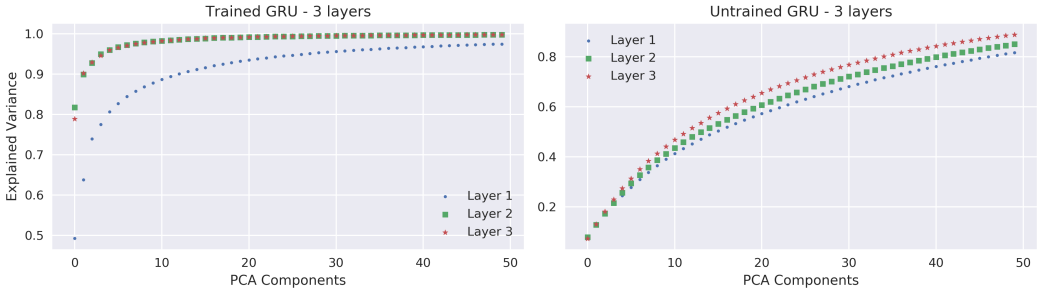



Figure 2: Explained Variance for a trained 3 layer GRU on IMDB versus untrained. Contrary to the 1 layer model, the first layer requires many more dimensions to explain the same variance. The second and third layer requires much less dimension to explain the same level of variance compared to the 1 layer model. Only the first 50 dimensions are shown.

Similar to the 1-layer case, a PCA was trained on the hidden states visited by each GRU layers on the test set. In other words, we have as many PCA as there are layers, they are trained independantly as they do not encode the same information and do not live in the same space (figure 3).

Unlike the explained variance of the 1-layer GRU, the first layer in this model requires much more dimensions to explain the same variance when considering the PCA components. After the first 50 dimensions we reach about the same level of explained variance. The biggest difference can be seen in the first 2 components where the 1-layer model explains close to 80% while the first layer of the 3-GRU barely explains 60%.

The second and third layer on the other hand encode all the relevant information in even less dimensions than the 1-layer model. The first 2 components explain over 90%. A possible explanation would be that the 1-layer is doing all the grunt work, weeding out the grass so that the subsequent

layer can focus on the relevant information, similar to a CEO getting all the distilled informations from the lower management.

### 3.1.2 Projections

Figures 1b and 3 show the projection of the visited hidden states on the test set by the 1-layer and 3-layer models respectively, on the first 2 PCA components, colored by their labels. Additionnally we look at the evolution of the hidden states of two movie reviews:

One positive

> It is by far one of the best comic book adaptations ever. I liked this one even more than X-men.

And one negative

> First of all I hate those moronic rappers, who couldn't act if they had a gun pressed against their foreheads. All they do is curse and shoot each other and acting like cliché'e version of gangsters.

Figure (1b) is a reproduction of the original results (see [1] - figure 12). The itinial starting hidden state (where the two path starts) is slightly different than the original paper, but this difference may very well be explained by the differences in training or even the random initialization of the network. Considering how the two examples evolve in this 2-dimension projection, we can see that they stay close to the middle at the beginning whene the words are non-informative. Once the emotionally charged words *best* and *moronic* are "read", then the hidden states go their separate ways and then keep more or less stumbling around. They are likely moving in higher dimensions and what we see here is a project so us mere mortal can visualize it.

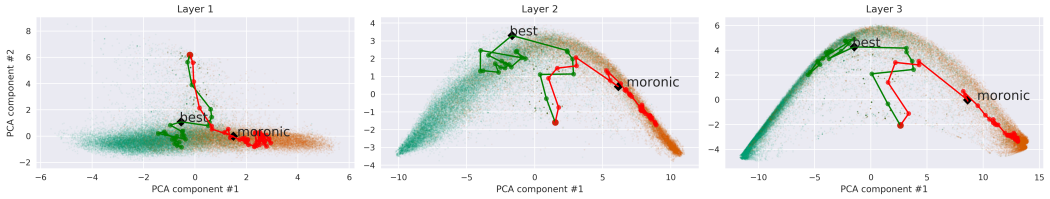



Figure 3: First 2 PCA components on each of the 3 layer model, colored by labels. The evolution of the hidden state of a positive and negative example are overlayed.

Now comes the interesting part. Visualizing the first two PCA components of each layers (figure 3) starts revealing interesting behaviours. Let's focus our attention to the evolution of the hidden states of the positive and negative examples. For the first layer, considering that the first two PCA components does not explain a whole lot of the variance, the path taken by the examples seems to be stumbling around. The activity is not as low dimensinal as in the 1-layer case.

The second and third layers have activity in much lower dimensions, even lower than the 1-layer case. The evolution of the examples are less stumbling around, more to the point for the task it was trained on.

An analogy that comes to mind is that the 1-layer model is like the undergrad student focusing on a higher understanding to get good enough grades and a good job. The 3-layer model is like a PhD that spent a lot of time to master the foundations of his field (first layer) and focused on his area of research during his graduate years (second and third layers). That student has a larger understanding of the foundation and when it comes to his specialization it's like a sharp shooter.

It may be strange that the starting points are far from the rest of the hidden points, but it makes total sense. Just like it makes no sense to go to the driving range and practice only your wood 1, there is far less starting shot in golf than approach shots, it makes no sense to expect starting points to be close to the rest because there are far less initial words than in-sentence words. Another interesting observation is the impact of emotionally charged words. In the positive example, the result

of "reading" the word *best* results in a sudden change in the hidden state in every layer. It's even more apparent in the top layers. Similarly, the word *moronic* makes the negative example shoot to the right, towards the negative area, very rapidely. It may be oversimplistic, but if we consider the first component to encode the task at hand, namely sentiment analysis, then the change of direction in that dimension could be used as low-tech explanation for the impact of the word. This may be an unwarranted simplification as this explanation could very well be overfitting these particular observations.

### 3.1.3 Fixed Points

In (1c) we can see where the fixed point procedure (2.3) ends up. 500 hidden states were sampled and ran until convergeance. Recall that the fixed point procedure uses a constant input vector $\mathbf{0}$ whereas all the hidden states present are actual hidden states from real text.

And now for the crux of the conversation we look at the fixed points. As described in the preliminaries 2.3, fixed point are found by minimizing the quantity $q^n = \frac{1}{N}\|\mathbf{h}^n - F^n(\mathbf{h}^n, \mathbf{0})\|_2^2$ at every layer $n$ where $\mathbf{h}^n$ is the hidden state going in at the n-th layer with a constant input of zero. Iteratively the starting hidden state is modified through gradient descent until it stabilizes (until $q < 1e-6$). If it never reaches that threshold then the starting hidden state is said to not converge. Figure 4 shows the start of 500 hidden states selected at random; the blue ones are points that ultimately converge and the green points don't. The black dots are where the convergeance ends. Two hidden states are converging to the same point if they end up in a hypercube of side $1^{-3}$ [4].

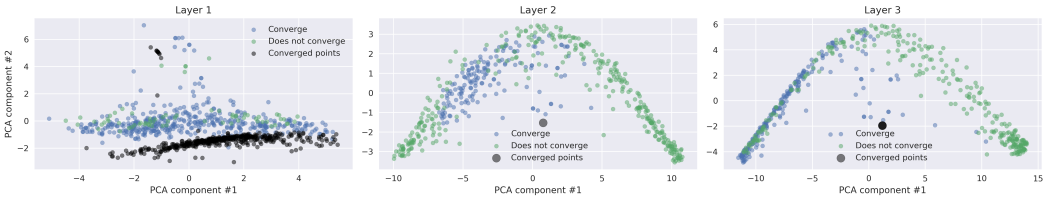



Figure 4: Fixed points for 2 layer GRU

The first layer has many of these fixed points and interestingly they are close to the projection of the initial hidden states. Whereas the second has only one and the third has three (two are so close they could be considered the same but are not due to the condition described above). Here we can observe a different behaviour, the rare fixed points are far from the rest of the points, standing suspiciously in the middle of the horse shoe. We will discuss this in the Fixed Point section below (3.1.3). One could think that there is a bug in the code, but considering that similar results as the orignal paper are reproduced (see figure 1c) then we can move forward with the analysis. Looking at the evolution of the hidden states of the positive and negative examples in the PCA space (figure 3) was already hinting at these fixed points and their linearization.

### 3.1.4 Linearization

The linearization around the fixed point of a GRU is not surprisingly done with a Taylor expansion

$$\mathbf{h}_t^n = F\left(\mathbf{h}_*^n + \Delta\mathbf{h}_{t-1}^n, \mathbf{x}_*^n + \Delta\mathbf{x}_{t-1}^n\right) \tag{1}$$

$$\approx F\left(\mathbf{h}_*^n, \mathbf{x}_*^n\right) + \mathbf{J}_n^{rec}\Delta\mathbf{h}_{t-1}^n + \mathbf{J}_n^{inp}\Delta\mathbf{x}_{t-1}^n \tag{2}$$

Where $\Delta\mathbf{h}_{t-1} = \mathbf{h}_{t-1} - \mathbf{h}_*$, $\Delta\mathbf{x}_{t-1} = \mathbf{x}_{t-1} - \mathbf{x}_*$ and $\left(\mathbf{J}^{rec}, \mathbf{J}^{inp}\right)$ are the Jacobian matrices of the system: $J_{ij}^{rec} = \frac{\partial F(\mathbf{h}^*, \mathbf{x}^*)_i}{\partial h_j^*}$ and $J_{ij}^{inp} = \frac{\partial F(\mathbf{h}^*, \mathbf{x}^*)_i}{\partial x_j^*}$. To not overload the notation the indices denoting the $n^{th}$ layer is not always shown. I trust my dear reader will be able to follow. A star always denote the fixed point, whether it's in subscript or superscript.
$\mathbf{h}_*$ is found numerically and $\mathbf{x}_* = \mathbf{0}$. Using those fixed points the linearization reduces to

$$\Delta\mathbf{h}_t = \mathbf{J}^{rec}\Delta\mathbf{h}_{t-1} + \mathbf{J}^{inp}\Delta\mathbf{x}_t.$$

[4] Taken directly from the default value of the original paper

Obviously the linearization depends on which hidden states we start from and we can see, in both figures 5 and 6, up to three different sets of eigenvalues of the Jacobian $\mathbf{J}^{rec}$ of the linearization around fixed points, for every layers.

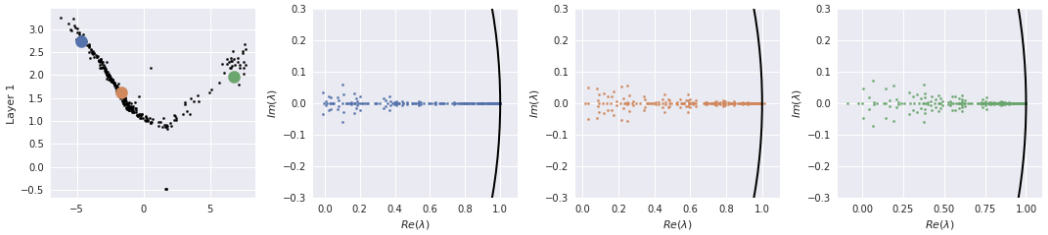



Figure 5: Eigenvalues of the linearization around different fixed points for the 1-layer model

This is where the results diverge the most from the original paper. No direct comparison can be done since they only show this part for an LSTM model trained on the Yelp dataset. Based on their description of their results, I was expecting to find quite similare results for GRU trained on IMDB. Alas this is not exactly the case. In their figure 3b ([1]) there are almost no visible imaginary part for their eigenvalues. Or perhaps it's a plotting defect and they are simply not visible enough. Regardless, figure 5 shows the eigenvalues of the linearization around three different fixed points. The conclusion of the autors was that across the range of fixed points, the eigenvalues are mostly real (no imaginary part) and less than 1. We can still observe the same behaviour, but not to the same extent. There are many more eigenvalues with imaginary part with positive real parts then what they are getting.

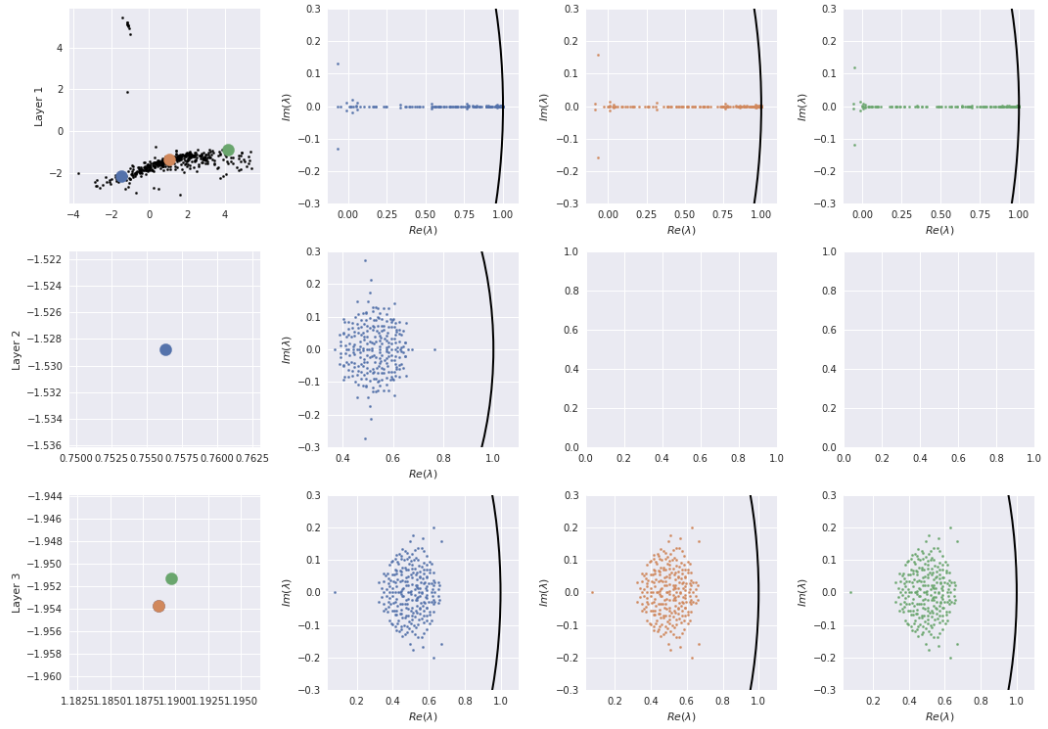



Figure 6: Eigenvalues of the linearization around different fixed points for the 3-layer model

And the fascinating cherry on top of the proverbial sunday is finally here. In the layer 1 of figure 3 we see the starting point of the positive and negative examples far from the point cloud only to converge to that cloud as words are input. Looking at the eigen values of the Jacobian $\mathbf{J}^{rec}$ for three different hidden states, a similar line attractor as described in the original paper appears. Most of the eigen

values have no imaginary part, thus no rotation, and real parts less than one. This means that there is convergeance and because every hidden states are showing similar behaviour we can empirically conclude that there is a line attractor. As mentioned above, the fixed points are basically in the cloud of hidden states.
The fixed points of the second and third layers are very different. Again, the starting hidden states are far from the cloud and get there as words are read, as if they were repelled from the fixed point. Looking at the eigen values of the Jacobian of the recurrent input reveals almost exclusively rotations going away from the fixed point (i.e. eigen values with imaginary parts and positve real parts). It's as if the first layer did all the grunt work and the top layers are all amped up and ready to run. They're running circles around their starting block and waiting for their signal. Building on the first layer's work, they are ready to jump to conclusion.

## 4 Discussion

As mentioned above, using the first PCA component to try and explain the impact of each word may be too simplistic. We could convince ourselves, perhaps on a case-by-case basis, that this is a reasonable thing to do. At the very least, more empirical observation could be made on binary task such as predicting bear/bull position for stock market or even multiclass problems where we cannot simply split the plane in half.

## 5 Futur direction

It would be interesting to observe the impact of train set size and play with underfitting and overfitting voluntarily. For example, a 4-layer model (D) will have the capacity to overfit more the training set than smaller models. What observations could we derive from this? Or even the 2-layer model (C) which seems to have a line attractor on the second layer as well.
All the analysis done above were on the test set. Would we find a completely different dynamic if it was done on the train/valid set instead?
Analyzing the behaviour of an RNN this way may push the boundaries in interpretability of deep learning. I am often asked why an NLP makes its prediction, why an obviously easy example (for a human) gets misclassified. It would be enlightening, from a business perspective, to be able to explain some of the decisions of the model to machine learning moguls. In my experience, I typically managed to calibrate everyone's expectation with the reality of machine learning, but you always find a surprising failed classification that is hard to rationalize. Local Interpretable Model-agnostic Explanations (LIME) [4] has been very helpful in that regards, but always felt more like a hack than actual explanation.

## Appendices

### A Model architecture

- Pretrained word embedding: GloVe trained on 6 billion words with vector of dimension 100.
- Vocabulary limited to 25k most common words
- Embedding layer (dim 100) -> Dropout (0.5) -> RNN (hidden state 256) -> Dropout (0.5) -> Fully Connected

## B Another appendix Subsection

## C 2 layers

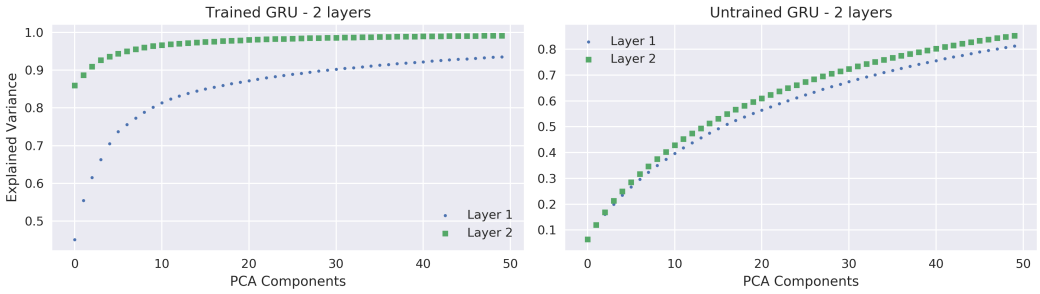



Figure 7: 2 layers GRU explained variance

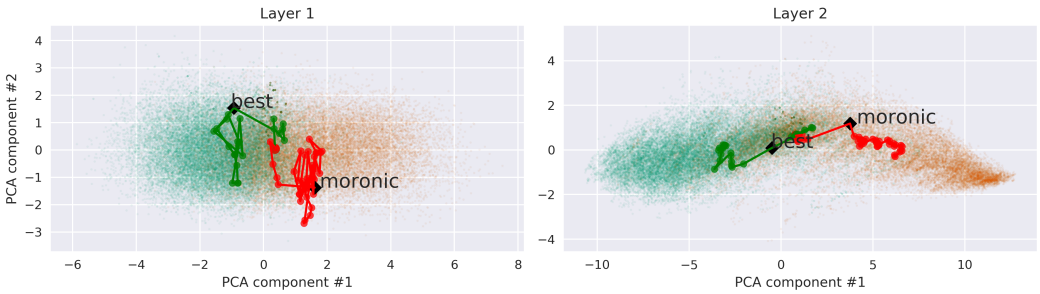



Figure 8: 2 layers GRU PCA

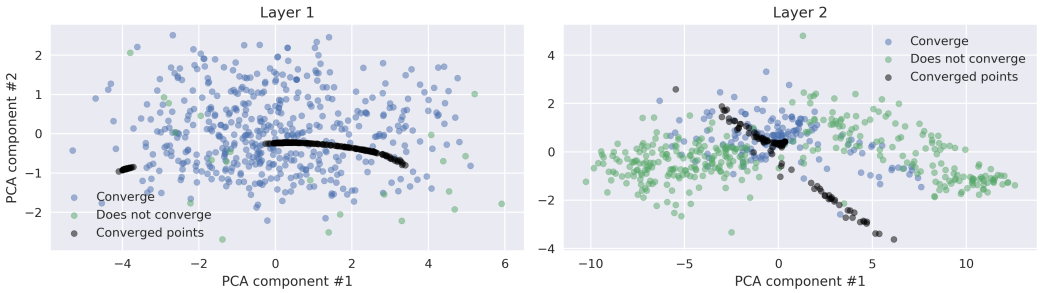



Figure 9: 2 layers GRU fixed points

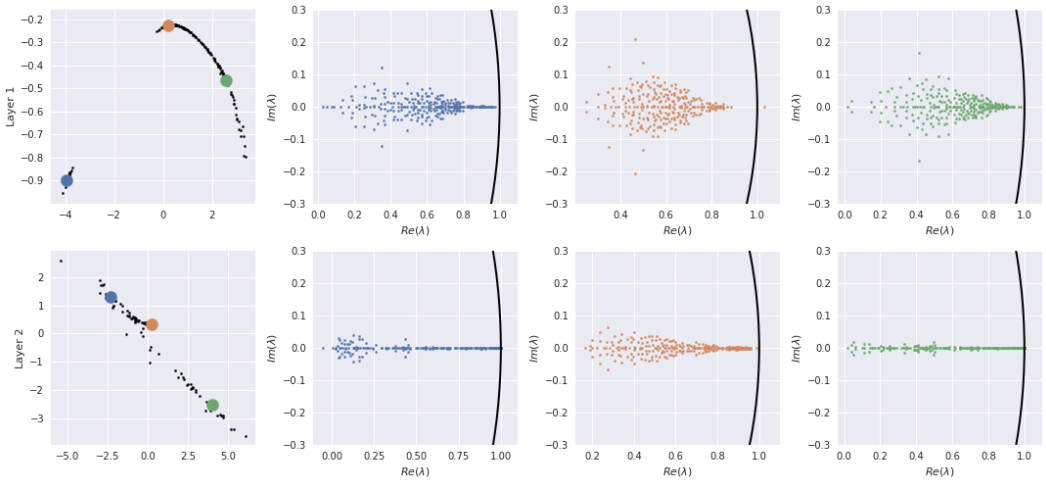



Figure 10: 2 layers GRU eigenvalues

## D 4 layers

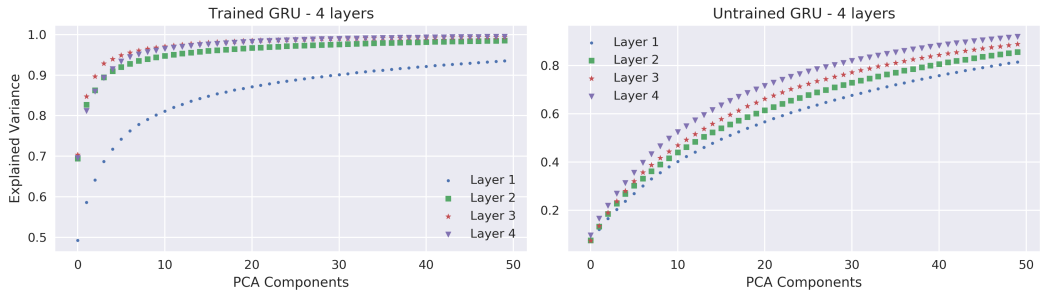



Figure 11: 4 layers GRU explained variance

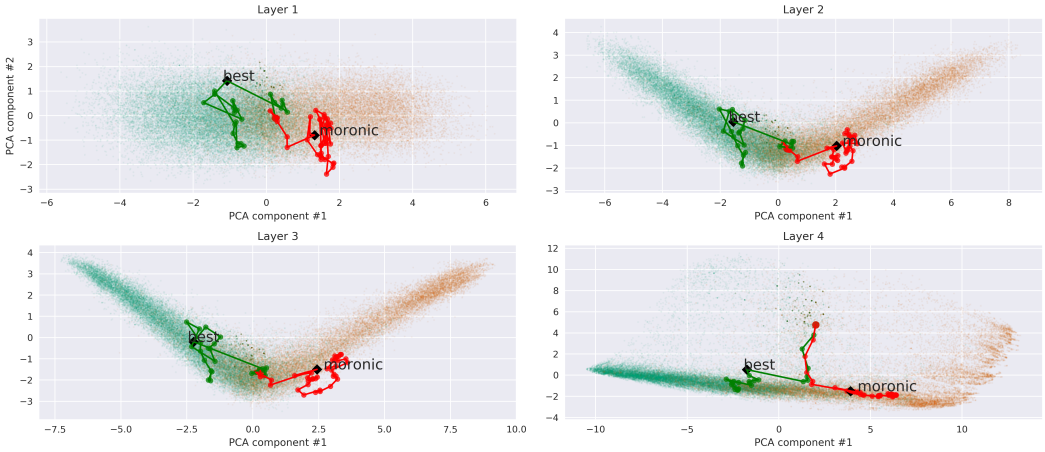



Figure 12: 4 layers GRU PCA

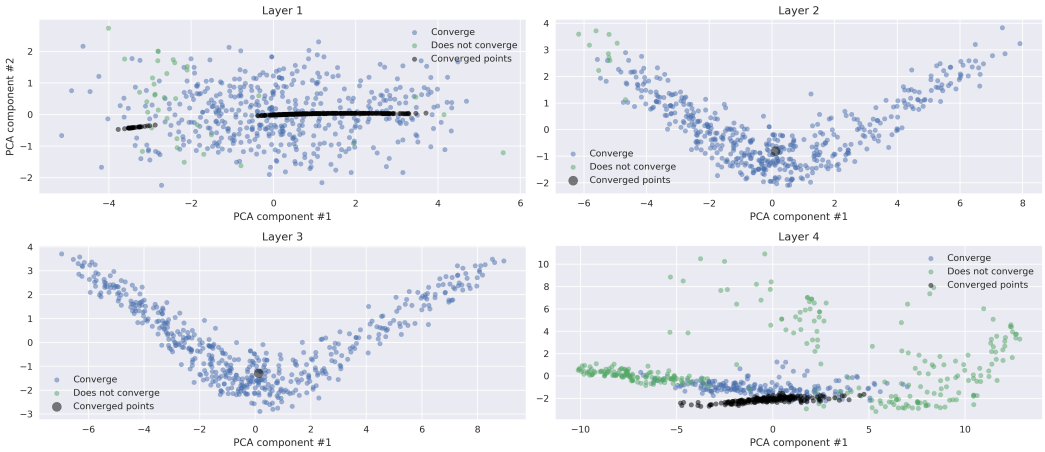



Figure 13: 4 layers GRU fixed points

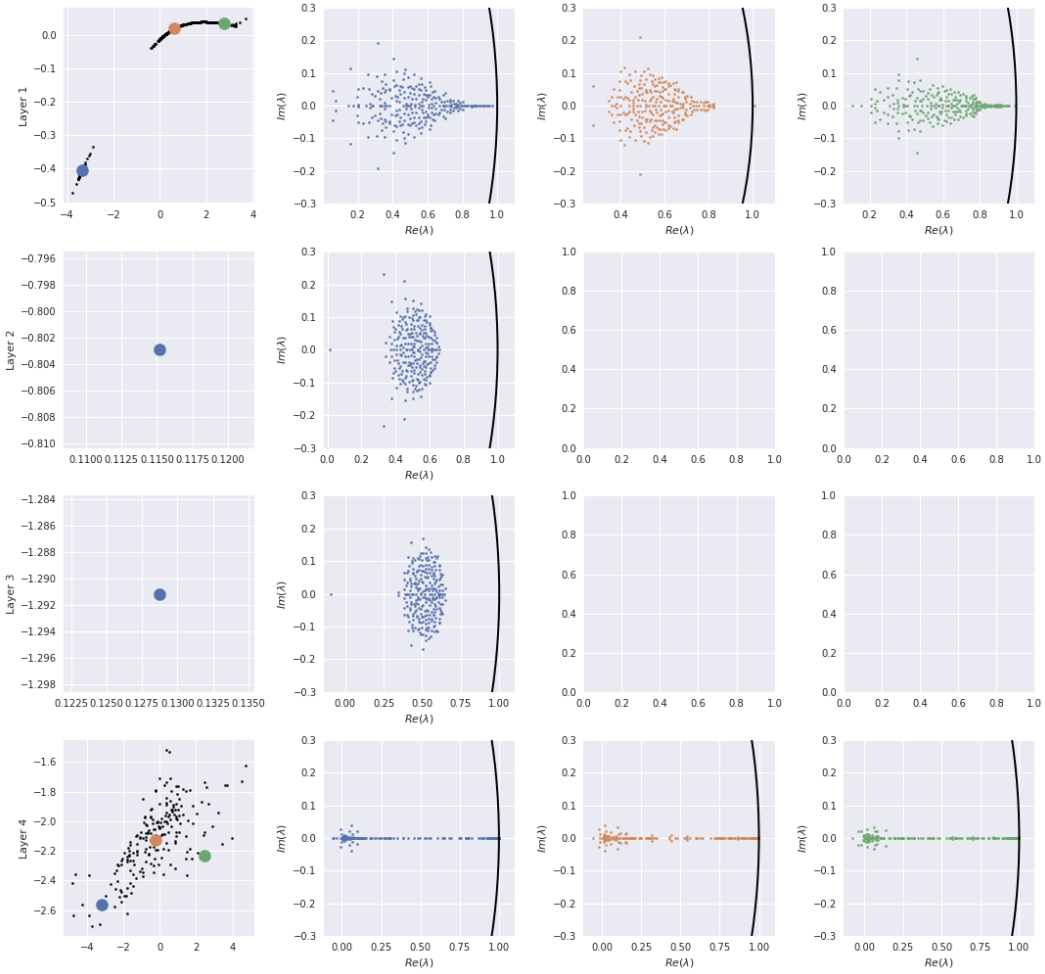



Figure 14: 4 layers GRU eigenvalues

[1] Matthew Golub and David Sussillo. FixedPointFinder: A tensorflow toolbox for identifying and characterizing fixed points in recurrent neural networks. Journal of Open Source Software, 3(31):1003, 2018.

[2] Bower, J.M. & Beeman, D. (1995) *The Book of GENESIS: Exploring Realistic Neural Models with the GEneral NEural SImulation System.* New York: TELOS/Springer–Verlag.

[3] Hasselmo, M.E., Schnell, E. & Barkai, E. (1995) Dynamics of learning and recall at excitatory recurrent synapses and cholinergic modulation in rat hippocampal region CA3. *Journal of Neuroscience* **15**(7):5249-5262.

[4] https://arxiv.org/pdf/1602.04938.pdf

## References

[1] Niru Maheswaranathan, Alex Williams, Matthew D. Golub, Surya Ganguli, and David Sussillo. Reverse engineering recurrent networks for sentiment classification reveals line attractor dynamics, 2019.

[2] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.

[3] Tomas Mikolov, Edouard Grave, Piotr Bojanowski, Christian Puhrsch, and Armand Joulin. Advances in pre-training distributed word representations. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*, 2018.

[4] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.