Visual Analysis of Hidden State Dynamics in Recurrent Neural Networks

Hendrik Strobelt, Sebastian Gehrmann, Bernd Huber, Hanspeter Pfister, and Alexander M. Rush
Harvard School of Engineering and Applied Sciences[1]
{hstrobelt, gehrmann, huber, pfister, rush}@seas.harvard.edu

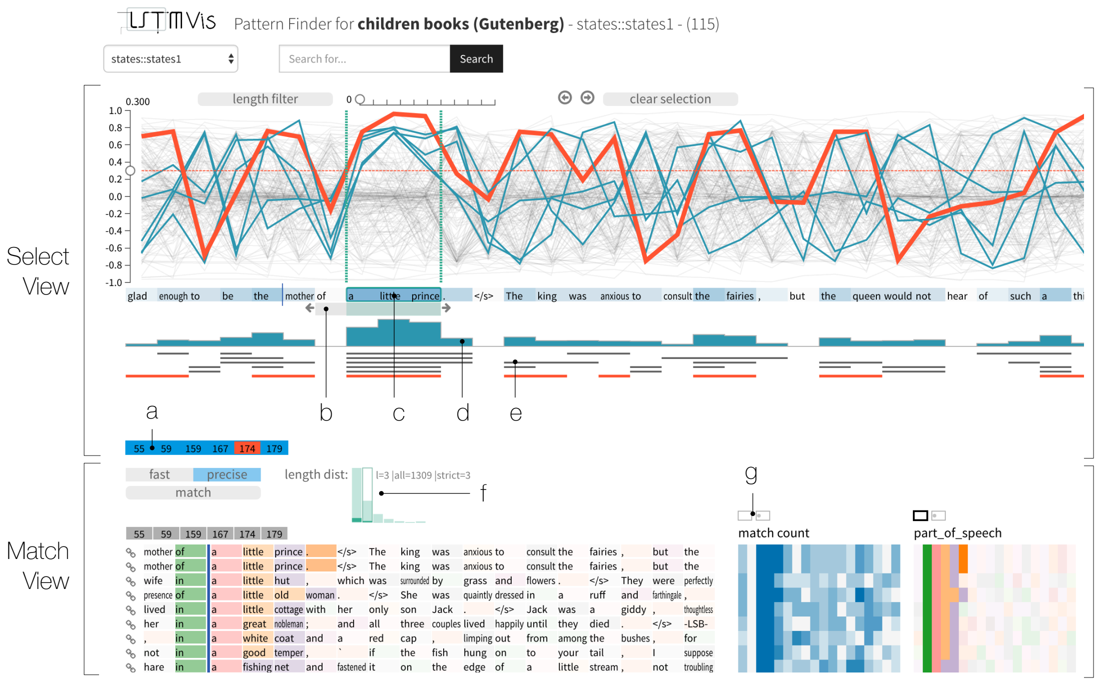


Fig. 1. The LSTMVis user interface. The user interactively *selects* a range of text specifying a hypothesis about the model. This range is then used to *match* similar hidden state patterns in the dataset which are displayed below. The selection is made by specifying a start-stop range in the text (gray border (b) and blue highlight (c)) and an activation threshold (red dashed line). The selection is visualized by (a) the hidden states selected, (d) the number of active states and (e) the activation ranges for each hidden state. The tool can then *match* this selection with similar hidden state patterns in the data set of varying lengths (f), providing insight into the representations learned by the model. The match view additionally includes user-defined meta-data (such as part-of-speech tags) (g) which allows the user to further refine or confirm the selection hypothesis.

Abstract— Recurrent neural networks, and in particular long short-term memory networks (LSTMs), are a remarkably effective tool for sequence modeling that learn a dense black-box hidden representation of their sequential input. Researchers interested in better understanding these models have studied the changes in hidden state representations over time and noticed some interpretable patterns but also significant noise. In this work, we present LSTMVis a visual analysis tool for recurrent neural networks with a focus on understanding these hidden state dynamics. The tool allows a user to select a hypothesis input range to focus on local state changes, to match these states changes to similar patterns in a large data set, and to align these results with domain specific structural annotations. We further show several use cases of the tool for analyzing specific hidden state properties on data sets containing nesting, phrase structure, and chord progressions, and demonstrate how the tool can be used to isolate patterns for further statistical analysis.

1 Introduction

Recurrent neural networks (RNNs) [6] have proven to be a very effective general-purpose model for capturing long-term dependencies in textual applications. Recent strong empirical results indicate that internal representations learned by RNNs capture complex relationships between the words within a sentence or document. These improved representation have led directly to end applications in machine translation [12,22], speech recognition [2], music generation [4], and text classification [5], among a variety of other applications.

While RNNs have shown clear improvements for sequence modeling, the models themselves are black boxes, and it remains unclear exactly *how* a particular model is representing long-distance relationships within a sequence. Typically, RNNs contain millions of parameters and utilize repeated non-linear transformations of large hidden representations under time-varying conditions. These factors make the model inter-dependencies challenging to interpret without sophisticated mathematical tools. How do we enable users to explore complex network interactions in an RNN and directly connect these abstract representations to human understandable inputs?

In this paper, we focus on visual analysis to allow experimenters to explore and form hypotheses about RNN hidden state dynamics in their models.

- We develop a visual encoding for exploring hidden state dynamics around a selected input phrase and finding similar hidden state patterns in a large dataset.
- We present use cases applying this technique to identify and explore patterns in RNNs trained on large datasets for text and other domains.
- We introduce the LSTMVIS tool to allows users to analyze a set of pre-trained models. A live system can be accessed via `lstm.seas.harvard.edu` and the source code is provided.

We start in Section 2 by formally introducing the recurrent neural network model and in Section 3 by describing related techniques for visualizing RNNs in practice. In Section 4 we describe domain goals and their mapping to visualization tasks, and then in Section 5 present the visual design choices made to satisfy these goals. In Section 6 we turn toward practical use cases and demonstrate the application of the tool to three different problems. We conclude by discussing implementation details and future challenges.

2 Background: Recurrent Neural Networks

In recent years, deep neural networks have become a central modeling tool for many artificial cognition tasks, such as image recognition, speech recognition, and text classification. While the architectures for these tasks differ, the models each learn a series of non-linear transformations to map an input into a hidden black-box feature representation. This hidden representation is learned to perform an end task.

For text processing and other sequence modeling tasks, recurrent neural networks (RNNs) are a central architecture. A major challenge of working with variable-length text sequences is producing compact representations which capture or summarize long-distance relations in the text. These relationships are particularly important for tasks that require processing and generating sequences such as machine translation. RNN-based models seem to effectively learn representations for this information.

Throughout this work, we will assume that we are given a sequence of words $w_1, \dots, w_T$ for time 1 to T. These might consist of English words that we want to translate or a sentence whose sentiment we would like to detect, or even some other symbolic input such as musical notes or code. Additionally we will assume that we have a mapping from each word into vector representation $\mathbf{x}_1, \dots, \mathbf{x}_T$. This representation can either be a standard fixed mapping, such as word2vec [20], or can be learned with the rest of the model.

Formally, RNNs are a class of neural networks that sequentially map input word vectors $\mathbf{x}_1 \dots \mathbf{x}_T$ to a sequence of fixed-length representations $\mathbf{h}_1, \dots, \mathbf{h}_T$. This is achieved by learning a function **RNN**, which is applied recursively at each time-step $t \in 1 \dots T$:

$$\mathbf{h_t} \leftarrow \mathbf{RNN}(\mathbf{x_t}, \mathbf{h_{t-1}})$$

which takes input vector $\mathbf{x_t}$ and a hidden state vector $\mathbf{h_{t-1}}$ and gives a new hidden state vector $\mathbf{h}_t$. Each hidden state vector $\mathbf{h}_t$ is in $\mathbb{R}^D$. These vectors, and particularly how they change over time, will be the main focus of this work. We are interested in each $c \in \{1 \dots D\}$ and the change of a single *hidden state* $h_{t,c}$ as t varies.

The model learns these hidden states to represent the features of the input words. As such they can be learned for any modeling tasks utilizing discrete sequential input. In this paper we will focus primarily on the task of RNN language modeling [19, 27], a core task in natural language processing. In language modeling, at time t the prefix of words $w_1, \dots, w_t$ is taken as input and the goal is to model the distribution over the next word $p(w_{t+1}|w_1, \dots, w_t)$. An RNN is used to produce this distribution by applying a linear model over the hidden state vector $\mathbf{h}_t$. Formally we define this as $p(w_{t+1}|w_1, \dots, w_t) = \text{softmax}(\mathbf{W}\mathbf{h_t} + \mathbf{b})$ where $\mathbf{W}, \mathbf{b}$ are parameters. The full computation of an RNN language model is shown in Figure 2.

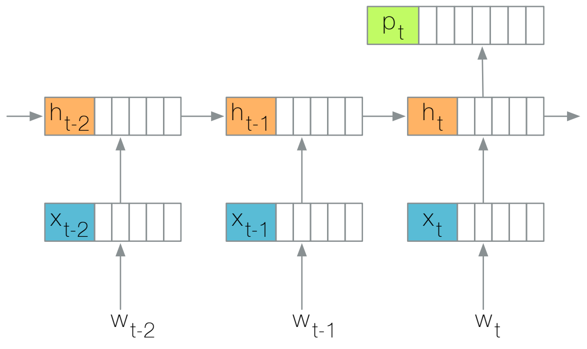


Fig. 2. A recurrent neural network language model being used to compute $p(w_{t+1}|w_1, \dots, w_t)$. At each time step, a word w_t is converted to a word vector $\mathbf{x}_t$, which is then used to update the hidden state $\mathbf{h_t} \leftarrow \mathbf{RNN}(\mathbf{x}_t, \mathbf{h}_{t-1})$. This hidden state vector can be used for prediction. In language modeling (shown) it is used to define the probability of the next word, $p(w_{t+1}|w_1, \dots, w_t) = \text{softmax}(\mathbf{W}\mathbf{h_t} + \mathbf{b})$.

It has been widely observed that the hidden states are able to capture important information about the structure of the input sentence necessary to perform this prediction. However, it has been difficult to trace how this is captured and what exactly is learned. For instance, it has been shown in some cases that RNNs can count parentheses or match quotes, but is unclear whether RNNs naturally discover aspects of language such as phrases, grammar, or topics. In this work, we focus particularly on exploring this question by examining the dynamics of the hidden states through time.

Finally, we note that our experiments will mainly focus on long short-term memory networks (LSTM) (hence the name LSTMVIS) [9]. LSTMs define a variant of the function **RNN** that has a modified hidden state update which can more effectively learn long-term interactions[1]. As such these models are widely used in practice. In addition, LSTMs and RNNs can be *stacked* in layers to produce multiple hidden state vectors at each time step, which further improves performance. While our results mainly use stacked LSTMs, our visualization only requires access to some time evolving abstract vector representation, and therefore can be used for any layer of the model.

3 Related Work

Understanding RNNs through Visualization

Our core contribution, visualizing the state dynamics of LSTM in a structured way, is inspired by previous work on convolutional networks for in vision applications [21, 28]. In linguistic tasks, visualizations have shown to be useful tool for understanding certain aspects of LSTMs. In [14], static visualization techniques are used to help understand LSTM hidden states in language models. This work demonstrates that selected cells can model clear events such as open parentheses and the start of URLs. In [17], additional techniques are presented, particularly the use of gradient-based saliency to find important words. This work also looks at several different models and datasets including text classification and auto-encoders. In [10, 11], the authors show that RNNs specifically learn lexical categories and grammatical functions that carry semantic information, partially by modifying the inputs fed to the model. While inspired by these techniques, our approach tries to extend beyond single examples and provide a general interactive visualization approach of the raw data for exploratory analysis.

Extending RNN Models for Interpretability

Recent work has also developed methods for extending RNNs for certain problems to make them easier to interpret (along with improving the models). One

[1]Note that LSTMs maintain both a cell state vector and a hidden state vector at each time step. Our system can be used to analyze either or both of these vectors (or even the LSTM gates), and in our experiments we found that the cell states are easier to work with. For simplicity, however, we refer to these vectors generically as “hidden states” throughout the paper.

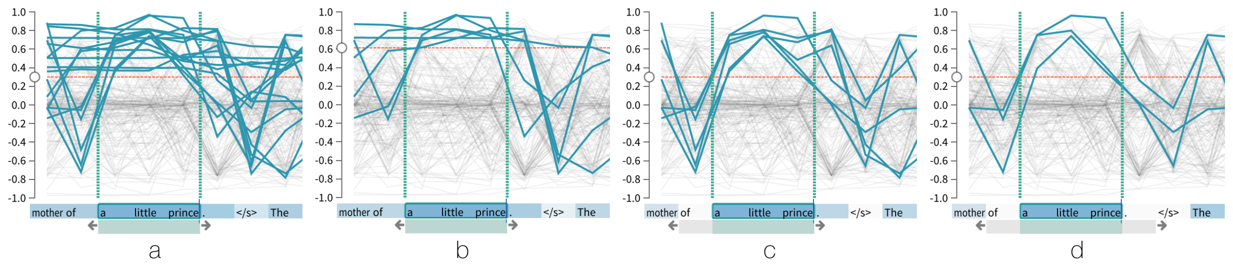


Fig. 3. The hypothesis selection process. In (a) the selection covers `a little prince` and has a threshold $\ell = 0.3$. Blue hidden highlighted states are selected. In (b) the threshold ℓ is raised to 0.6. In (c) the bottom gray slider is extended left, eliminating hidden states with values above ℓ after reading `of` (word to the left). In (d) the gray slider is additionally extended right, removing hidden states above the threshold after reading "`.`" (word to the right).

popular technique has been to use a neural attention-mechanism to allow the model to focus in on a particular aspect of the input. In [3] attention is used for soft alignment in machine translation, in [26] attention is used to identify important aspects of an image for captioning, and in [8] attention is used to find important aspects of a document for an extraction task. These approaches have the side benefit that they "show" what aspect of the model they are using. This approach differs from our work in that it requires changing the underlying model structure, whereas we attempt to interpret the hidden states of a fixed model directly.

Interactive Visualization of Neural Networks There has been some work on interactive visualization for interpreting machine learning models. In [24], the authors present a visualization system for feed-forward neural networks with the goal of interpretation, and in [13], the authors give a user-interface for tuning the learning itself. The recent Prospector system [15] provides a general-purpose tool for practitioners to better understand their ML model and its predictions. There has also been work on user interfaces for constructing models such as TensorBoard [1] and the related playground for convolutional neural models `playground.tensorflow.org/`. Our work is most similar in spirit to [24] in that we are mostly concerned with interpreting the hidden states of a particular model, however our specific goals and visual design are significantly different.

4 Goals and Tasks

Given that RNNs act as a black-box, their success leaves open the question of why they are so effective at representing the history of words. LSTMVis focuses particularly on the dynamics of RNN hidden states and targets the related question:"What information does an RNN capture in its hidden states?". Addressing this question is the main goal of our project and the focus of a series of discussions. During this iterative process, we identified the following domain goals for a user of LSTMVis:

- **G1 - Formulate a hypothesis** about (linguistic) properties that the hidden states might learn to capture for a specific model. This hypothesis requires an initial understanding of hidden state values over time and a close read of the original text.
- **G2 - Refine the hypothesis** based on insights about learned textual similarities based on patterns in the dynamics of the hidden states. Refining a hypothesis may also mean rejecting it.
- **G3 - Compare models and datasets** to allow early generalization about the insights the representations provide, and to observe how task and domain may alter the patterns in the hidden states.

From these three goals we propose tasks for visual data analysis. The mapping of these tasks to domain goals is indicated by square brackets:

- **T1 - Visualize hidden states** over time to allow exploration of the hidden state dynamics in their raw form. [G1]
- **T2 - Filter hidden states** by using discrete textual selection along with continuous thresholding. These selections methods allow the user to form hypotheses and to separate visual signal from noise. [G1,G2]
- **T3 - Match selections to similar examples** based on hidden state activation pattern. A matched phrase should have intuitively similar characteristics as the selection to support or reject a hypothesis. [G2]
- **T4 - Align textual annotations** visually to matched phrases. These annotations allow the user to compare the learned representation with alternative structural hypotheses such as part-of-speech tags or known grammars. The set of annotation data should be easily extensible. [G2,G3]
- **TX - Provide a general interface** that can be used with any RNN model and text-like dataset. It should make it easy to generate crowd knowledge and trigger discussions on similarities and differences between a wide variety of models. [G3]

5 Visual Design

LSTMVis supports the formulation of a hypothesis (T1, T2, G1) in the Select View (Sect. 5.1) and can trigger refinement of a hypothesis (T3, T4, G2) in the Match View (Section 5.2), while remaining agnostic to the underlying data or model (TX). We first describe the visual design and interaction paradigms used in the two views and how they facilitate the domain goals, and then discuss design iterations for LSTMVis in Section 5.4.

5.1 Select View

The Select View, shown in the top half of Figure 1, is centered around a single time-series plot. The x-axis is labeled with the word inputs $w_1, \dots, w_T$ for the corresponding time step. (If words do not fit into the fixed width for time steps they are distorted). In the plot itself, we show the hidden state vectors $\mathbf{h}_1, \dots, \mathbf{h}_T$ at each time step (one point for each of the D values). The hidden state dynamics are encoded through time to form a parallel coordinates plot (T1). That is there is one line for each of the D hidden states between each time-steps. Figure 1 shows the movement of each hidden state though the full sequence.

The full plot of hidden state dynamics can be difficult to comprehend directly. Therefore, LSTMVis allows the user to formulate a hypothesis (G1) about the semantics of a subset of hidden states localized to a range of text. The user selects a phrase that may express an interesting property. For instance, the user may select a range within a shared nesting levels in tree-structured text (see Section 6.1), a representative noun phrase in a text corpus (see Section 6.2), or a chord progression in a musical corpus(see Section 6.3).

To select, the user brushes over a range of words that form the pattern of interest. In this process, she implicitly focuses on the hidden states that are "on" in the selected range. The dashed red line on the parallel coordinates plot indicates a user-defined threshold value, ℓ, that partitions the hidden states into "on" (all timesteps $\geq \ell$) and "off" (any $< \ell$) within this range. In addition to selecting a range, the user can modify the brush slider below (gray) to define that hidden states must also be "off" immediately before or after the selected range. Figure 3 shows different combinations of slider configurations and the corresponding hidden state selections. We call this set of selected hidden states $\mathscr{S}_1 \subset \{1 \ldots D\}$.

The defined selection of hidden states is mirrored in a discrete plot below the word labels. Blue bar charts indicate the percentage of "on" cells at each visible time step that are in the set selected $\mathscr{S}_1$. The gray lines underneath reveal the ranges that are covered by each of the selected hidden states. These elements enable the user to preview the coverage of sequences in the local neighborhood. To eliminate high-frequency changes along the time axis, a length filter can be applied.

At the bottom of the Select View, the full set $\mathscr{S}_1$ is listed. Hovering over a hidden state representation in one of the described plots highlights this hidden state across all plots. Hidden states can also be deselected individually. Figure 1 shows a selected hidden state highlighted in red.

The described interactive methods allow the user to define a hypothesis range which results in the selection of a subset of hidden states based on the definition of a specific threshold (T2, G1) and only relies on the hidden state vectors themselves (TX). To refine or reject the hypothesis the user can then make use of the Match View.

5.2 Match View

The Match View, shown in the bottom half of Figure 1, provides evidence for or against the selected hypothesis. The view provides a set of relevant matched phrases that have similar hidden state patterns as the phrase selected by the user. This style of nearest neighbors search can provide an intuitive view of the hidden states that are on for the hypothesis.

With the goal of maintaining an intuitive match interface, we define the matches to be "ranges in the data set that would have lead to a similar set of on hidden states under the selection criteria". Formally, assume that the user has selected a threshold ℓ with hidden states $\mathscr{S}_1$ and has not limited the selection to the right or left further. We rank all possible candidate ranges in the dataset starting at time a and ending at time b with a two step process

1. collect the set of all hidden states that are "on" for the range,

$$\mathscr{S}_2 = \{c \in \{1 \ldots D\} : h_{t,c} \geq \ell \text{ for all } \ a \leq t \leq b\}$$

2. rank the candidates by the number of overlapping states $|\mathscr{S}_1 \cap \mathscr{S}_2|$ using the inverse of number of additional "on" cells $-|\mathscr{S}_1 \cup \mathscr{S}_2|$ and candidate length $b-a$ as tiebreaks.

If the original selection is limited on either side (as in Figure 3), we modify step (2) to take this into account for the candidates. For instance if there is a limit on the left, we only include state indices c in $\mathscr{S}_2$ in that also satisfy $h_{a-1,c} < \ell$.

For efficiency, in practice we do not score at all possible candidate ranges (datasets typically have $T > 1$ million). We limit the candidate set by filtering to ranges with a minimum number of hidden states from $\mathscr{S}_1$ over the threshold ℓ. These candidate sets can be computed efficiently using run-length encoding.

A length histogram is also generated that indicates the distribution of phrase lengths in the matches. Hovering over a histogram bin reveals details about this bin and clicking on one filters the matches to the desired length.

The top 50 results are shown in the Match View. For each time step, the matches are encoded as a linked heatmap, which indicates the amount of overlap with $\mathscr{S}_1$ at each timestep. The color of the heatmap for each time step can be applied directly to the background of the results to better see the matches.

Furthermore the user can provide additional annotations which are displayed as categorical heatmaps (T4). We imagine these annotations can act as ground truth data, e.g. part-of-speech tags for a text corpora, or as further information to help calibrate the hypotheses. Mapping annotation data to the matches is a simple method to reveal pattern across results. These results can lead to further data analysis or a refinement of the current hypothesis.

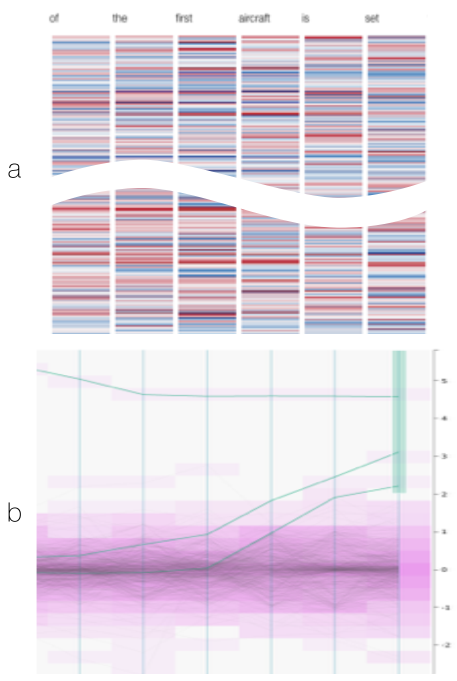


Fig. 4. Early-stage prototypes of the system. (a) Hidden state vectors are encoded as heatmaps over time. This style places emphasis on the relationships between neighboring (vertically adjacent) states, which has no particular meaning for this model. (b) A selection prototype utilizing parallel coordinates. This prototype emphasized selections based on small movements of state values directly on the plot, which made it difficult to specify connections between hidden state values and source text.

5.3 Navigation Along the Time Axis

LSTMVis provides several convenience methods to navigate to specific time steps. Buttons on the timeline can be used to move forward and backward. LSTMVis also offers search functionality to find specific phrases. Finally, the selection panel on the top left can be used to efficiently switch between the different layers of the same model and between datasets (TX). As all different layers and datasets can be displayed in the same way, the user can easily compare models.

5.4 Design Iterations

During the course of the project we developed seven interactive prototypes of varying complexity highlighting different aspects of the data. In this section we present two fundamental design decisions that lead to the final system.

5.4.1 Visual Encoding of State Dynamics

Inspired by a standard static visualization in the RNN literature, we first encoded hidden state vectors as a heatmap along the time-axis (Figure 4(a)). This style has been favored as a view of the complete set of hidden states $\mathbf{h}_1, \ldots, \mathbf{h}_T$. However, this approach has several drawbacks in an interactive visualization. Foremost, the heatmaps do not scale well with increasing dimensionality D of hidden state vectors. They use a non-effective encoding for the most important information, i.e. hidden state values by color hue. Additionally they emphasize the order of hidden states in each vector, but this relative order of abstract hidden states is not actually used by the model itself.

Instead we decided to consider each hidden state as a data item and time-steps as dimensions for each data item in a parallel coordinates plot. Doing so, we encode the hidden state value using the more

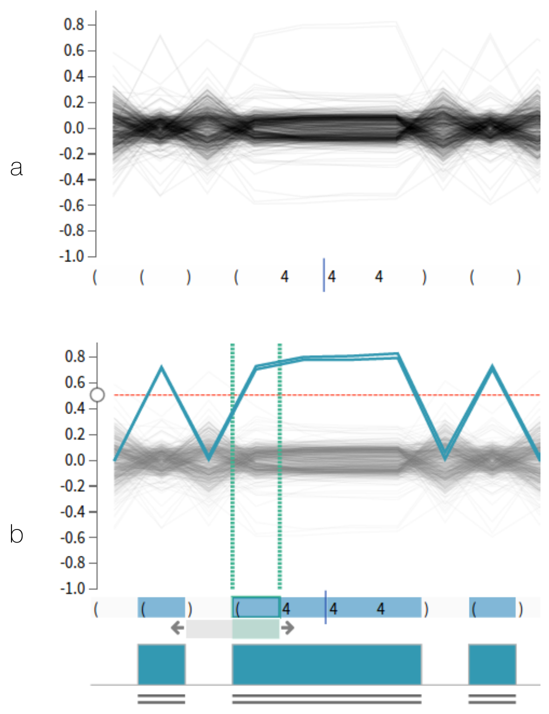
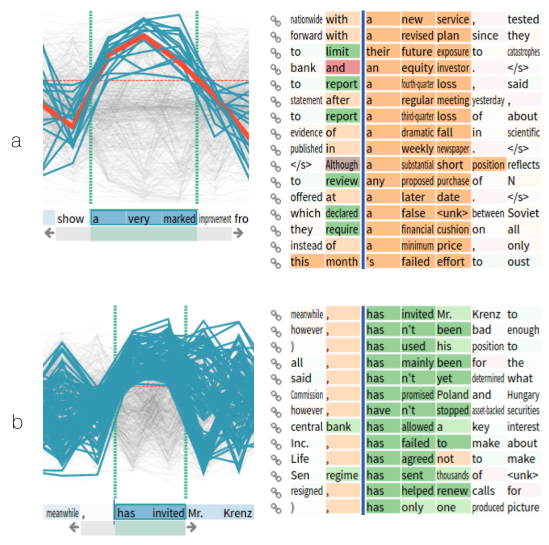


Fig. 5. Plot of a phrase from the parenthesis synthetic language. In (a), the full set of hidden states is shown. Note the strong movement of states at parenthesis boundaries. In (b), a selection is made at the start of the fourth level of nesting. Even in the select view it is clear that several hidden states represent a four-level nesting count.



nationwide	with	a	new	service	,	tested
forward	with	a	revised	plan	since	they
to	limit	their	future	exposure	to	catastrophes
bank	and	an	equity	investor	.	</s>
to	report	a	fourth-quarter	loss	,	said
statement	after	a	regular	meeting	yesterday	,
to	report	a	third-quarter	loss	of	about
evidence	of	a	dramatic	fall	in	scientific
published	in	a	weekly	newspaper	.	</s>
</s>	Although	a	substantial	short	position	reflects
to	review	any	proposed	purchase	of	N
offered	at	a	later	date	.	</s>
which	declared	a	false	<unk>	between	Soviet
they	require	a	financial	cushion	on	all
instead	of	a	minimum	price	,	only
this	month	's	failed	effort	to	oust

meanwhile	,	has	invited	Mr.	Krenz	to
however	,	has	n't	been	bad	enough
)	,	has	used	his	position	to
all	,	has	mainly	been	for	the
said	,	has	n't	yet	determined	what
Commission	,	has	promised	Poland	and	Hungary
however	,	have	n't	stopped	asset-backed	securities
central	bank	has	allowed	a	key	interest
Inc.	,	has	failed	to	make	about
Life	,	has	agreed	not	to	make
Sen	regime	has	sent	thousands	of	<unk>
resigned	,	has	helped	renew	calls	for
)	,	has	only	one	produced	picture

Fig. 6. Phrase selections and match annotations in the Wall Street Journal. In (a), the user selects a closed selection of `a very marked improvement` (turning off when improvement is seen). The matches found are entirely other noun phrases, and start with different words. Note that here ground-truth noun phrases are indicated with an orange highlight. In (b) we select an open range starting with `has invited`. The results are various open verb phrases (green highlight). Note that for both examples the model can return matches of varying lengths.

effective visual variable *position*. Figure 4(b) shows the first iteration on using a parallel coordinates plot. The abundance of data points along the plot is additionally encoded with a heatmap in the background to emphasize dense regions (e.g. around the zero value) but also highlight sparse regions. In the final iteration, we omitted this redundant encoding for the sake of clarity and to highlight wider regions of text.

5.4.2 Formulating a Hypothesis

One challenge we faced in early design iterations was allowing the user to easily express hypotheses with selection. In Figure 4(b), we show a preliminary draft using a common filter method for parallel coordinates along each axis. When experimenting with this kind of selection, two major drawbacks of this approach became evident. First, it was very cumbersome to formulate a hypothesis for a longer range by adjusting many y-axis brush selectors at a fine granularity. Second, selecting directly on the hidden state values felt decoupled from the original source of information – the text. The key idea to facilitate this selection process was allow the user to easily discretize the data based on a threshold and select on and off ranges directly on top of the words (as described in Section 5). This idea generalizes and adds interactivity to the manual approaches developed in [14].

6 Use Cases

In experimenting with the system we trained and explored many different RNN models, datasets and tasks, including word and character language models, neural machine translation systems, auto-encoders, summarization systems, and classifiers. Additionally we also experimented with other types of real and synthetic input data.

In this section we highlight three findings that demonstrate the general applicability of LSTMVis paradigms for analysis of hidden states.

6.1 Proof-of-Concept: Parenthesis Language

As proof of concept we trained an LSTM as language model on synthetic data generated from a very simple counting language with a parenthesis and letter alphabet $\Sigma = \{$`( ) 0 1 2 3 4` $\}$. The language is constrained to match parentheses, and nesting is limited to at most 4 levels deep, where each opening parenthesis increases nesting level and each closing parenthesis decreases the nesting level. Numbers are generated randomly, but are constrained to indicate the nesting level at their position. For example a string in the language looks like:

```
( 1 ( 2 ) () ) 0 ( ( ( 3 ) ) 1 )
```
,

where blue lines indicates ranges of nesting level $\geq$1. Similarly, orange and green lines indicates nesting level $\geq$2 and $\geq$3.

To analyze this language, we view the states in LSTMVis (we show the the cell states of a multi-layer 2x300 LSTM model). An example is shown in Figure 5(a). Here even the initial parallel coordinates plot shows a strong regularity, as hidden state changes occur predominately at parentheses.

Our hypothesis is that the hidden states mainly reflex the nesting level. To test this, we select a range spanning nesting level four by selecting the phrase `( 4`. We immediately see that several hidden states seem to cover this pattern and that in the local neighborhood several other occurrences of our hypothesis are covered as well, e.g. the empty parenthesis and the full sequence `( 4 4 4`. This observation simply confirms earlier observations that has demonstrate simple context-free models in RNNs and LSTMs [7, 25].

6.2 Phrase Separation in Language Modeling

Next we consider the case of a real-world natural language model. For this experiment we trained a 2-layer LSTM language model with 650 hidden states on the Penn Treebank [18] following the medium-sized model of [27]. While the model is trained for language modeling (predict the next word), we were interested in seeing if it additionally learned properties about the underlying language structure. To test this, we additionally include annotations in the model from the Penn Treebank. We experimented with including part-of-speech tags, named entities, and parse structure.

Here we focus on the case of *phrase chunking*. We annotated the dataset with the gold-standard phrase chunks provided by the CoNLL 2003 shared task [23] for a subset of the treebank (Sections 15-18).

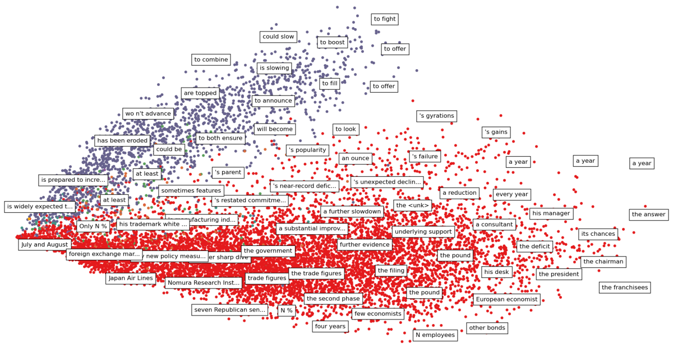


Fig. 7. PCA projection of the hidden state patterns ($\mathscr{S}_1$) of all multi-word phrasal chunks in the Penn Treebank, as numerical follow-up to the phrase chunking hypothesis. Red points indicate noun phrases, blue points indicate verb phrases, other colors indicate remaining phrase types. While trained for language modeling, the model separates out these two phrase classes in its hidden states.

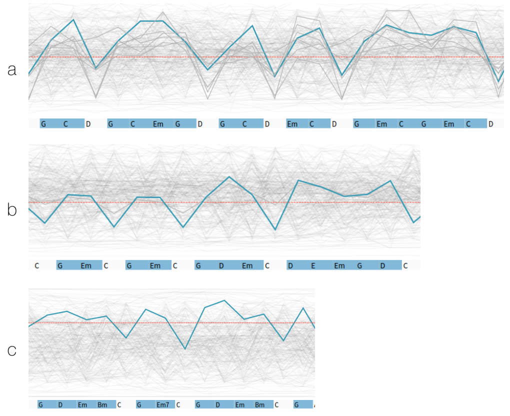


Fig. 8. Three examples of single state patterns in the guitar chord dataset. In (a), we see several permutation of the very common I - V - vi - IV progression (informally, the "Don't Stop Believing" progression). In (b) we see several patterns ending in a variant of the I- vi- IV- V (the 50's progression). In (c), we see two variants of I - V - vi -iii - IV - I (beginning of the Pachelbel's Canon progression). Chord progression patterns are based on `http://openmusictheory.com/`.

These include annotations for noun phrases and verb phrases, along with prepositions and several other less common phrase types.

While running experimental analysis, we found a strong pattern that selecting noun phrases as hypotheses leads to almost entirely noun phrase matches. Additionally we found that selecting verb phrase prefixes would lead to primarily verb phrase matches. In Figure 5.4.2(a,b) we show two examples of these selections and matches.

This hints that the model has implicitly learned a representation for language modeling that can differentiate between the two types of phrases. Of course the tool itself cannot confirm or deny this type of hypothesis, but the aim is to provide clues for further analysis. We can check, outside of the tool, if the model is clearly differentiating between the classes in the phrase dataset. To do this we compute the set $\mathscr{S}_1$ for every noun and verb phrase in the shared task. We then run PCA on the vector representation for each set. The results are shown in Figure 6.2, which shows that indeed these on-off patterns are enough to partition the noun phrases and verb phrases.

6.3 Musical Chord Progressions

Finally we looked at some non-text data sets to get a better understanding of long-range patterns. Past work on LSTM structure has emphasized cases where single hidden states are semantically interpretable. For text data sets, we found that with a few exceptions (quotes, brackets, and commas) this was rarely the case. However, for datasets with more regular long-term structure, single states could be quite meaningful.

As a simple example, we collected a large set of songs with annotated chords for rock and pop songs to use as a training data set, 219k chords in total. We then trained an LSTM language model to predict the next chord w_{t+1} in the sequence, conditioned on previous chord symbols (chords are left in their raw format).

When we viewed the results in LSTMVIS we found that the regular repeating structure of the chord progressions is strongly reflected in the hidden states. Certain states will turn on at the beginning of a standard progression, and remain on though variant-length patterns until a resolution is reached. In Figure 6.3, we examine three very common general chord progressions in rock and pop music. We select a prototypical instance of the progression and show a single state that captures the pattern, i.e. remains on when the progression begins and turns off upon resolution.

7 Implementation

LSTMVIS consists of two modules, the visualization system and the RNN modeling component.

The visualization is a client-server system that uses Javascript and D3 on client side and Python, Flask, h5py, and numpy on server side. Timeseries data (RNN hidden states and input) is loaded dynamically through HDF5 files. Optional annotation files can be specified to map categorical data to labels (T4). New data sets can be added easily by a declarative YAML configuration file.

The RNN modeling system is completely separated from the visualization to allow compatibility with any deep learning framework (TX). For our experiments we utilized the Torch framework and the Element RNN library [16]. We trained our models separately and exported results to the visualization.

The source code and models are available at `lstm.seas.harvard.edu`.

8 Conclusion

LSTMVIS provides an interactive visualization to facilitate data analysis of recurrent neural network hidden states. The tool is based on a two-step process where a user can *select* a range of text to represent a hypothesis about the RNN representation, the tool then can *match* this selection to other examples in the data set. The tool easily allows for external annotations to verify or reject hypothesizes. It minimally requires a time-series of hidden states, which makes it easy to adopt for a wide range of visual analyses of different data sets and models, and even different tasks (language modeling, translation etc.).

To demonstrate the use of the model we presented three case studies describing how the tool can be applied to different data sets. On synthetic data, the tool clearly separates out the core underlying structure. On natural language data, states are noisier, but we can find clear splits between known linguistic structures like noun and verb phrases. For these tasks the tool not only helps narrow down hypotheses but also provides specific information such as hidden states and textual annotations to spur on further statistical testing. For future work, we would like to explore different matching criteria, to allow other forms of annotation, and to analyze the usage of the tool in practice.

Acknowledgments

This work was supported in part by the Air Force Research Laboratory and DARPA grant FA8750-12-C-0300.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[2] D. Amodei, R. Anubhai, E. Battenberg, C. Case, J. Casper, B. C. Catanzaro, J. Chen, M. Chrzanowski, A. Coates, G. Diamos, E. Elsen, J. Engel, L. Fan, C. Fougner, T. Han, A. Y. Hannun, B. Jun, P. LeGresley, L. Lin, S. Narang, A. Y. Ng, S. Ozair, R. Prenger, J. Raiman, S. Satheesh, D. Seetapun, S. Sengupta, Y. Wang, Z. Wang, C. Wang, B. Xiao, D. Yogatama, J. Zhan, and Z. Zhu. Deep speech 2: End-to-end speech recognition in english and mandarin. *CoRR*, abs/1512.02595, 2015.

[3] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014.

[4] N. Boulanger-Lewandowski, Y. Bengio, and P. Vincent. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. In *Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 - July 1, 2012*. icml.cc / Omnipress, 2012.

[5] A. M. Dai and Q. V. Le. Semi-supervised sequence learning. In *Advances in Neural Information Processing Systems*, pp. 3079–3087, 2015.

[6] J. L. Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

[7] F. A. Gers and E. Schmidhuber. Lstm recurrent networks learn simple context-free and context-sensitive languages. *IEEE Transactions on Neural Networks*, 12(6):1333–1340, 2001.

[8] K. M. Hermann, T. Kociský, E. Grefenstette, L. Espeholt, W. Kay, M. Suleyman, and P. Blunsom. Teaching machines to read and comprehend. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, eds., *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pp. 1693–1701, 2015.

[9] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[10] A. Kádár, G. Chrupała, and A. Alishahi. Lingusitic analysis of multi-modal recurrent neural networks. 2015.

[11] Á. Kádár, G. Chrupała, and A. Alishahi. Representation of linguistic form and function in recurrent neural networks. *arXiv preprint arXiv:1602.08952*, 2016.

[12] N. Kalchbrenner and P. Blunsom. Recurrent continuous translation models. In *EMNLP*, vol. 3, p. 413, 2013.

[13] A. Kapoor, B. Lee, D. Tan, and E. Horvitz. Interactive optimization for steering machine classification. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 1343–1352. ACM, 2010.

[14] A. Karpathy, J. Johnson, and F.-F. Li. Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*, 2015.

[15] J. Krause, A. Perer, and K. Ng. Interacting with predictions: Visual inspection of black-box machine learning models. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pp. 5686–5697. ACM, 2016.

[16] N. Léonard, S. Waghmare, and Y. Wang. Rnn: Recurrent library for torch. *arXiv preprint arXiv:1511.07889*, 2015.

[17] J. Li, X. Chen, E. Hovy, and D. Jurafsky. Visualizing and understanding neural models in nlp. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 681–691. Association for Computational Linguistics, San Diego, California, June 2016.

[18] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330, 1993.

[19] T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur. Recurrent neural network based language model. In *Interspeech*, vol. 2, p. 3, 2010.

[20] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pp. 3111–3119, 2013.

[21] K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

[22] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, pp. 3104–3112, 2014.

[23] E. F. Tjong Kim Sang and F. De Meulder. Introduction to the conll-2003 shared task: Language-independent named entity recognition. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*, pp. 142–147. Association for Computational Linguistics, 2003.

[24] F.-Y. Tzeng and K.-L. Ma. Opening the black box-data driven visualization of neural networks. In *VIS 05. IEEE Visualization, 2005.*, pp. 383–390. IEEE, 2005.

[25] P. R. J. Wiles. Recurrent neural networks can learn to implement symbol-sensitive counting. In *Advances in Neural Information Processing Systems 10: Proceedings of the 1997 Conference*, vol. 10, p. 87. MIT Press, 1998.

[26] K. Xu, J. Ba, R. Kiros, K. Cho, A. C. Courville, R. Salakhutdinov, R. S. Zemel, and Y. Bengio. Show, attend and tell: Neural image caption generation with visual attention. In F. R. Bach and D. M. Blei, eds., *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, vol. 37 of *JMLR Proceedings*, pp. 2048–2057. JMLR.org, 2015.

[27] W. Zaremba, I. Sutskever, and O. Vinyals. Recurrent Neural Network Regularization. *arXiv:1409.2329*, 2014.

[28] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *European Conference on Computer Vision*, pp. 818–833. Springer, 2014.