LSTMVis: A Tool for Visual Analysis of Hidden State Dynamics in Recurrent Neural Networks

Hendrik Strobelt, Sebastian Gehrmann, Hanspeter Pfister, and Alexander M. Rush
– Harvard School of Engineering and Applied Sciences –

arXiv:1606.07461v2 [cs.CL] 30 Oct 2017

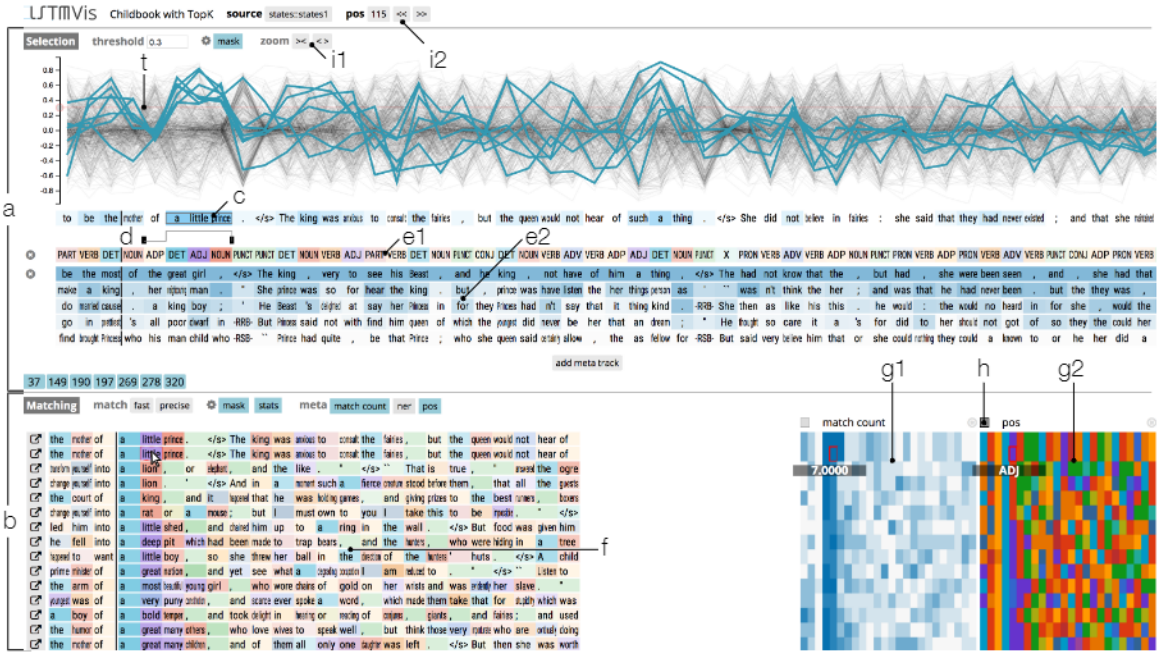

Fig. 1. The LSTMVIS user interface. The user interactively *selects* a range of text specifying a hypothesis about the model in the Select View (a). This range is then used to *match* similar hidden state patterns displayed in the Match View (b). The selection is made by specifying a start-stop range in the text (c) and an activation threshold (t) which leads to a selection of hidden states (blue lines). The start-stop range can be further constrained using the pattern plot (d). The meta-tracks below depict extra information per word position like POS (e1) or the top K predictions (e2). 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 encoded as heatmaps (g1,g2). The color of one heatmap (g2) can be mapped (h) to the word matrix (f) which allows the user to see patterns that lead to further refinement of the selection hypothesis. Navigation aids provide convenience (i1, i2).

Abstract— Recurrent neural networks, and in particular long short-term memory (LSTM) networks, 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 users 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 structural annotations from their domain. We show several use cases of the tool for analyzing specific hidden state properties on dataset containing nesting, phrase structure, and chord progressions, and demonstrate how the tool can be used to isolate patterns for further statistical analysis. We characterize the domain, the different stakeholders, and their goals and tasks. Long-term usage data after putting the tool online revealed great interest in the machine learning community.

1 Introduction

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. These models all share a common property in that they utilize a *hidden* feature representation of their input, not pre-specified by the user, which is learned for the task at hand. These hidden representations have proven to be very effective for classification. However, the black-box nature of these learned representations make the models themselves difficult to interpret. So while it is possible for users to produce high-performing systems, it is difficult for them to analyze what the system has learned.

While all deep neural networks utilize hidden features, different model structures have shown to be effective for different tasks. Standard deep neural networks (DNNs) learn fixed-size features, whereas convolutional neural networks (CNNs), dominant in image recognition, will learn a task-specific filter-bank to produce spatial feature maps. In this work, we focus on deep neural network architectures known as recurrent neural networks (RNNs) that produce a time-series of hidden feature-state representations.

RNNs [7] have proven to be an effective general-purpose approach

- *HS – contact: hendrik@strobelt.com*
- *SG, HP, AR – contact: {gehrmann, pfister,rush}@seas.harvard.edu.*

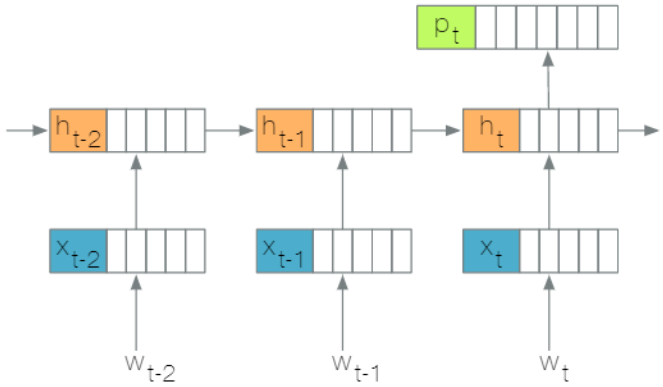

Fig. 2. A recurrent neural network language model being used to compute $p(w_{t+1}|w_1,\ldots,w_t)$. At each time step, a word w_t is converted to a word vector $\mathbf{x}_t$, which is then used to update the hidden state $\mathbf{h_t} \leftarrow \mathbf{RNN}(\mathbf{x}_t, \mathbf{h}_{t-1})$. This hidden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

for capturing representation in sequence-modeling applications, such as text processing. Recent strong empirical results indicate that internal representations learn to capture complex relationships between the words within a sentence or document. These improved representation have led directly to end applications in machine translation [14, 28], speech recognition [2], music generation [4], and text classification [6], among a variety of other applications.

While RNNs have shown clear improvements for sequence modeling, it has proven very difficult to interpret their feature representation. As such, 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 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 work, we focus on the visual analysis of hidden features in RNNs. We have developed LSTMVIS, a tool to allow advanced user groups to explore and form hypotheses about RNN hidden state dynamics. We analyzed neural network users and identified three major user roles, each with a different set of requirements: *architects* who develop novel deep learning structures, *trainers* who develop new data sets to train existing models, and *end users* who apply deep models to new data. LSTMVis is focused on architects and trainers, for which we performed a goal and task analysis to develop effective visual encodings and interactions. LSTMVis combines a time-series based *select* interface with an interactive *match* tool to search for similar hidden state patterns in a large dataset. A live system can be accessed via `lstm.seas.harvard.edu` and the source code is provided. We present use cases applying our technique to identify and explore patterns in RNNs trained on large real-world datasets for text, speech recognition, biological sequence analysis and other domains. We discuss our release-before-publish strategy used to develop and improve the tool based on user feedback.

2 Background: Recurrent Neural Networks

RNNs are a type of deep neural network architecture that has proven effective for sequence modeling tasks such as text processing. A major challenge of working with variable-length text sequences is producing features that 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 effectively learn hidden representations at each time-step which are then used for decision making. We refer to the change in these representations over time as the *hidden state dynamics* produced by the model.

Throughout this work, we will assume that we are given a sequence of words $w_1,\ldots,w_T$ 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 $\mathbf{x}_1,\ldots,\mathbf{x}_T$. This representation can either be a standard fixed mapping, such as word2vec [23], or can be learned with the rest of the model.

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

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

This function 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 $c \in \{1\ldots D\}$ and in particular 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. One notable application is RNN language modeling [22, 33], a core task in natural language processing. In language modeling, at time t the prefix of words $w_1,\ldots,w_t$ is taken as input and the goal is to model the distribution over the next word $p(w_{t+1}|w_1,\ldots,w_t)$. An RNN is used to produce this distribution by applying multi-class classification based on hidden feature-state vector $\mathbf{h}_t$. Formally we define this as $p(w_{t+1}|w_1,\ldots,w_t) = \text{softmax}(\mathbf{W}\mathbf{h_t} + \mathbf{b})$, where $\mathbf{W}, \mathbf{b}$ are parameters. The full computation of an RNN language model is shown in Figure 2.

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 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.

Our use cases will mainly focus on long short-term memory networks (LSTM) [11]. LSTMs define a variant of RNN that has a modified hidden state update which can more effectively learn long-term interactions. 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 a wide variety of models.

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.

3 Related Work

Understanding RNNs through Visualization Our core contribution, visualizing the state dynamics of RNNs in a structured way, is inspired by previous work on convolutional neural networks in vision applications [27, 34]. In linguistic tasks, visualizations have shown to be useful tool for understanding certain aspects of RNNs. Karpathy et al. [16] use static visualization techniques to help understand hidden states in language models. Their work demonstrates that selected cells can model clear events such as open parentheses and the start of URLs.

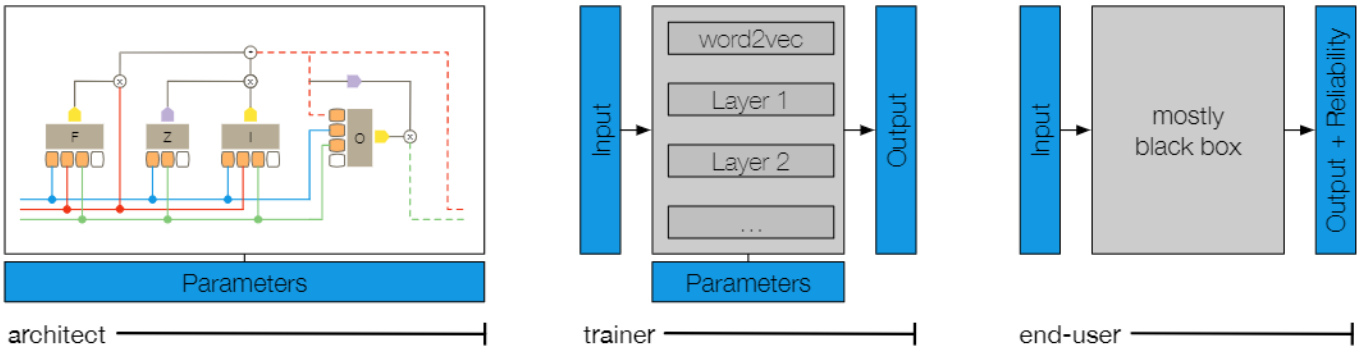

Fig. 3. Views on neural network models for different user roles. The **architect** analyzes and modifies all components of the system. The **trainer** abstracts the model to the main components and parameters and focuses on training on different data sets. The **end user** has the most abstract view on the model and considers whether the output is coherent for a given input.

Li et al. [18] present additional techniques, particularly the use of gradient-based saliency to find important words. Their work also looks at several different models and datasets including text classification and auto-encoders. Kadar et al. [12, 13] 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 popular technique has been to use a neural *attention* mechanism to allow the model to focus in on a particular aspect of the input. Bahdanau et al. [3] use attention for soft alignment in machine translation. Xu et al. [32] use attention to identify important aspects of an image for captioning, whereas Hermann et al. [10] use attention to find important aspects of a document for an extraction task. These approaches have the side benefit that they visualize the 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. Tzeng et al. [30] present a visualization system for feed-forward neural networks with the goal of interpretation, and Kapoor et al. [15] give a user-interface for tuning the learning itself. The Prospector system [17] provides a general-purpose tool for practitioners to better understand their machine learning model and its predictions.

Recent work also describes systems that focus on analysis of hidden states for convolutional neural networks. Liu et al. [20] utilize a DAG metaphor to show neurons, their connections, and learned features. Rauber et al. [25] use projections to explore relationships between neurons and learned observations. Other work has focused on user interfaces for constructing models, such as TensorBoard [1] and the related playground for convolutional neural models at `playground.tensorflow.org/`. Our work is most similar in spirit to the work by Tzeng et al. [30], Liu et al. [20], and Rauber et al. [25], in that we are concerned with interpreting the hidden states of neural network models. However, our specific goals focus on RNNs and the needs of specific users, and the resulting visual design is significantly different.

4 User Analysis and Goals

Deep neural networks are now widely employed both in the research and industrial setting by a diverse set of users with different needs. Before developing our visual analysis goals we first laid out a set of prototypical stakeholders who might benefit from improved analysis. In meetings with members of a natural language processing group and a computational biology group, we identified three user roles, their incentives, and their view on neural network models. Figure 3 summarizes the following roles and which aspect of a model they might hope to master:

- **Architects** are looking to develop new deep learning methodologies or to modify existing deep architectures for new domains. An architect is interested in training many variant network structures and comparing how the models capture the features of their domain. We assume that the architects are deeply knowledgeable about machine learning, neural networks, and the internal structure of the system. Their direct goal is comparing the performance of variant models and understanding the learned properties of the system.

- **Trainers** are those users interested in applying known architectures to tasks for which they are a domain experts. Trainers utilize RNNs as a tool and understand the key concepts of network optimization. However, their main focus is on the application domain and utilizing effective methods to solve known problems. Their goal is to use a known network architecture and to observe how it learns a novel model. Examples of trainers include a bioinformatician or an applied machine learning engineer.

- **End Users** make up the most prevalent role of network users. End users utilize general-purpose pretrained networks for various tasks. These users may not need to understand the training process at all, only how to apply the networks as an algorithm to new data. Their main desire is to explain the results and locate what is happening when something goes wrong. Examples of end-users include data scientists or product engineers using ML.

These user roles are general to the neural network domain and we believe they can help to describe and understand stakeholders in this space. For our analysis, we decided to focus on the more advanced end of this spectrum, particularly on the user role of *architects*. We aimed to provide these users with greater visibility into the internals of the system. User feedback from our first prototype further motivated us to include the *trainer* role as users with a focus on the predictions of the model. In future work, we want to develop systems to engage the end users of RNN and other deep neural networks more.

With these roles in mind, we aimed to help trainers and architects better understand the high-level question: "What information does an RNN capture in its hidden feature-states?". Addressing this question is the main goal of our project. Based on a series of dicussions with deep learning experts we identified the following domain goals:

- **G1 - Formulate a hypothesis** about 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 input.

- **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.

During the design phase, we developed the following list of tasks for visual data analysis from the three domain goals (G1–G3). The mapping of these tasks to 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]
- **T5 - 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]

This list of tasks provided a guideline for the design of LSTMVIS. In addition, tasks (T1-T4) define the core interaction mechanisms for discovery with LSTMVIS: Visualize (Section 5.1) – Filter & Select (Section 5.2) – Match & Align (Section 5.3). We will first describe the implementation of these interactions and later demonstrate their application to multiple use cases in Section 6.

5 Design of LSTMVis

LSTMVIS is composed of two major visual analysis components. The *Select View* (Section 5.2) supports the formulation of a hypothesis (T2, G1) by using a novel visual encoding for hidden state changes (T1, Section 5.1). The *Match View* (Section 5.3) allows refinement of a hypothesis (T3, T4, G2) while remaining agnostic to the underlying data or model (T5).

Our design decisions are the outcome of a long iterative process. First, we developed several interactive, low-fidelity prototypes of varying complexity with our expert users to highlight different aspects of the data (Figure 4). After developing a complete system, we released it to a broader audience online to collect long-term feedback (Section 7). Based on the feedback from this release we developed the current version several months later.

5.1 Visualization of Hidden State Changes

Visualizing the progression of hidden state vectors $\mathbf{h}_1, \ldots, \mathbf{h}_T$ along a sequence of words (time-steps) is at the core of LSTMVIS. In the following, we refer to hidden state as one dimension of the D-dimensional hidden state vector.

In the related literature, it is common to encode hidden state vectors as a heatmap along the time-axis (Figure 4(a)). This style has been favored as a view of the complete hidden state vectors $\mathbf{h}_1, \ldots, \mathbf{h}_T$. 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,

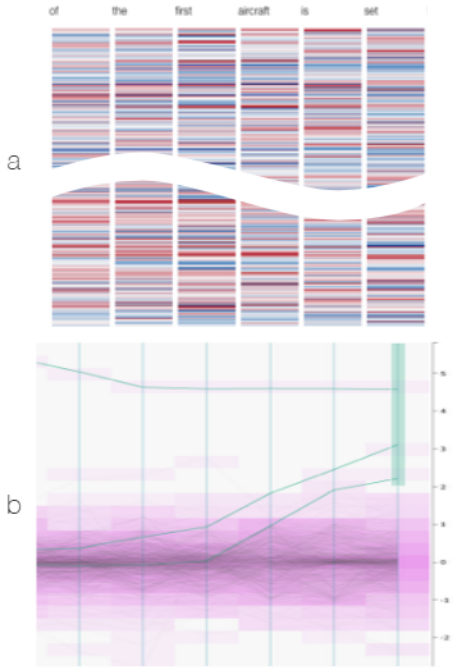

Fig. 4. Two early-stage prototypes of the system. (a) Hidden state vectors are encoded as heatmaps over time. This style places emphasis on the relationships between neighboring (vertically adjacent) states, which has no particular meaning for this model. (b) A selection prototype utilizing parallel 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.

but this relative order of abstract hidden states is not actually used by the model itself.

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 effective visual variable *position*. Figure 4(b) shows the first iteration on using a parallel coordinates plot. The number of data points along the plot is additionally encoded by a heatmap in the background to emphasize dense (e.g., around the zero value) and sparse regions.

However, it was very cumbersome to formulate a hypothesis for a longer range by adjusting many y-axis brush selectors at a fine granularity. Allowing the user to perform selection by directly manipulating the hidden state values felt decoupled from the original source of information—the text. The key idea to facilitate this selection process was to allow the user to easily discretize the data based on a threshold and select on and off ranges directly on top of the words (Section 5.3).

Figure 6 shows the first complete prototype that we put online to collect long-term user feedback (Section 7). Several user comments led us to believe that the redundant encoding of hidden states that are "on" in the Select View (Figure 6d,e) was not well understood. In the final design shown on the top in Figure 1 we omitted this redundant encoding for the sake of clarity and to highlight wider regions of text. The x-axis is labeled with the word inputs $w_1, \ldots, w_T$ for the corresponding time-step. If words do not fit into the fixed width for time steps they are distorted. Figure 1 shows the movement of a hidden state vector through an example sequence.

5.2 Select View

The Select View, shown in the top half of Figure 1, is centered around the parallel coordinates plot for hidden state dynamics. The full plot 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 by selecting a range of words that may express an

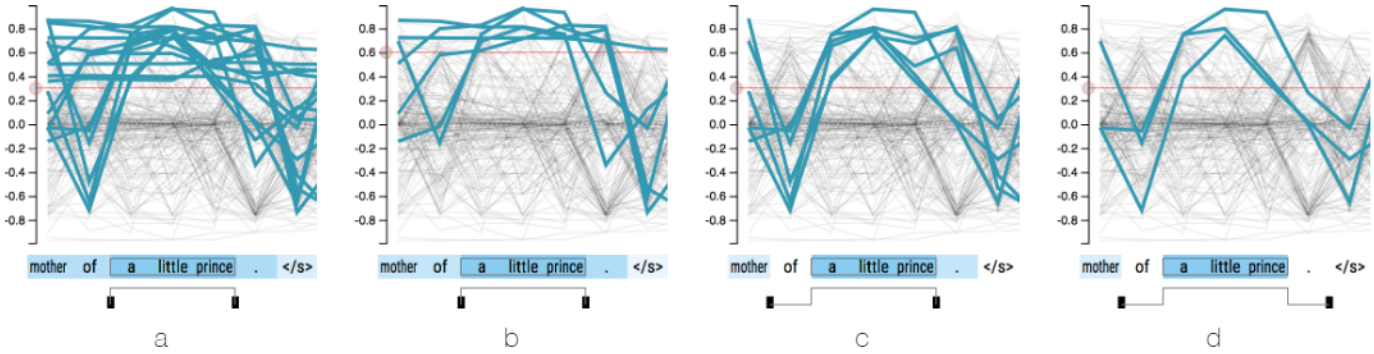

Fig. 5. The hypothesis selection process. (a) The selection covers `a little prince` and has a threshold $\ell = 0.3$. Blue highlighted hidden states are selected. (b) The threshold ℓ is raised to 0.6. (c) The pattern plot in the bottom is extended left, eliminating hidden states with values above ℓ after reading `of` (one word to the left). (d) The pattern plot is additionally extended right, removing hidden states above the threshold after reading "`.`" (one word to the right). I.e., only hidden states are selected with the following pattern: below threshold ℓ for one word before – above ℓ during `a little prince` – below ℓ for one word after.

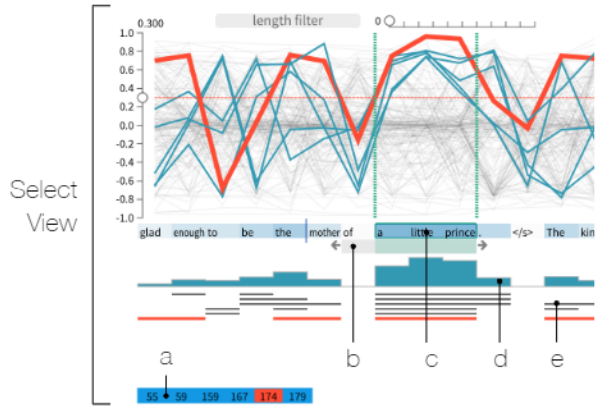

Fig. 6. Snippet of the first complete prototype. While the list of selected cells (a) and the brushing method (c) remained in the final version, we modified the pattern plot (b) and omitted the redundant encodings (d) and (e).

interesting property. For instance, the user may select a range within a shared nesting levels in tree-structured text (Section 6.1), a representative noun phrase in a text corpus (Section 6.2), or a chord progression in a musical corpus(Section 6.4).

To select, the user brushes over a range of words that form the pattern of interest (Figure 1c). In this process, she implicitly selects hidden states that are "on" in the selected range. The dashed red line on the parallel coordinates plot (Figure 1t) indicates a user-defined threshold value, ℓ, that partitions the hidden states into "on" (timesteps $\geq \ell$) and "off" (timesteps $< \ell$) within this range.

In addition to selecting a range, the user can modify the pattern plot below (Figure 1d) to define that hidden states must also be "off" immediately before or after the selected range. Figure 5 shows different combinations of pattern plot configurations and the corresponding hidden state selections. Using these selection criteria the user creates a set of selected hidden states $\mathscr{S}_1 \subset \{1 \ldots D\}$ that follow the specified on/off pattern w.r.t. the defined threshold.

To assist with the selection of ranges, the user can make use of a heatmap underlying the word sequence which depicts how many of the selected hidden states are "on" for each word (Figure 1c). This indicates repetitions of hidden state patterns in the close local neighborhood.

Additionally, a user can add aligned tracks of textual annotations (meta tracks) to the selection view. E.g., she can visualize part-of-speech (POS) annotations or named entity recognition (NER) results. The feature of meta tracks is the result of feedback from multiple online users asking us to include the tracks. Some users that used the tool for training also wanted to see the top K predictions (outcomes) for each word. Figure 1 shows examples for POS (e1) and topK (e2).

LSTMVIS also provides several convenience methods to navigate to specific time steps. Buttons on the timeline can be used to move forward and backward (Figure 1-i2). LSTMVIS also offers search functionality to find specific phrases. The selection panel on the top can be used to efficiently switch between the different layers (T5). The word-width can be decreased and increased (Figure 1-i1) to provide support for different average word lengths (e.g., character-based models vs. word-based models). The option to turn heatmaps on and off in Match View (Figure 1g1 and g2) provides convenience when working with many mate tracks.

These interaction methods allow the user to define a hypothesis as word range which results in the selection of a subset of hidden states following a specified pattern w.r.t. a defined threshold (T2, G1) that only relies on the hidden state vectors themselves (T5). To refine or reject the hypothesis the user can then make use of the Match View.

5.3 Match View

The Match View (Figure 1b) 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.

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 (threshold, on/off pattern). 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, as shown in Figures 5a and 5b. 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 the 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, e.g., in Figures 5c or 5d), 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, 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

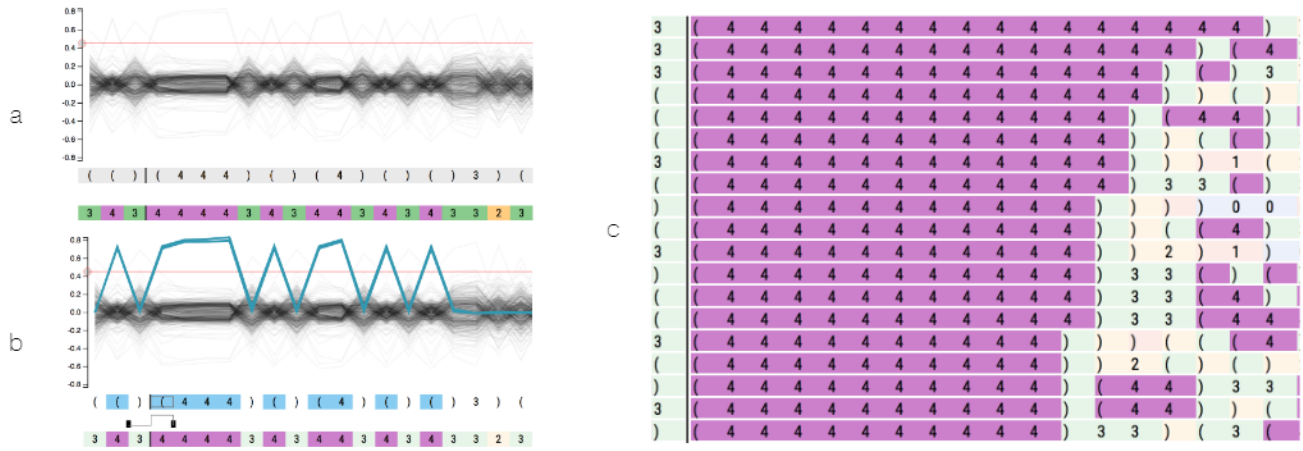

Fig. 7. Plot of a phrase from the parenthesis synthetic language. (a) The full set of hidden states. Note the strong movement of states at parenthesis boundaries. (b) A selection is made at the start of the fourth level of nesting. Even in the select view it is clear that several hidden states represent a four-level nesting count. In both plots the meta-track indicates the nesting level as ground truth. (c) The result of matching the selected states indicates that they seem to capture nesting level 4 phrases of variable length.

of hidden states from $\mathscr{S}_1$ over the threshold ℓ. These candidate sets can be computed efficiently using run-length encoding.

From the matching algorithm, we retrieve the top 50 results which are shown, one per row each, in a word matrix (e.g., Figure 1f) located in the Match View. Each cell in the word matrix is linked to a cell in corresponding heatmaps. These heatmaps encode additional information about each word in the matching results. The always available *match count* heatmap (Figure 1-g1) encodes the number of overlapping states $|\mathscr{S}_1 \cap \mathscr{S}_2|$ for each timestep.

The user can use additional annotations, similar to meta tracks in the Selection View, as heatmaps (T4). We imagine these annotations act as ground truth data, e.g., part-of-speech tags for a text corpora (Figure 1-g2), or as further information to help calibrate the hypotheses, e.g., nesting depth of tree-structured text. Figure 1 shows how hovering over "little" (mouse pointer) leads to highlights in the *match count* heatmap (g1) indicating seven overlapping states between match and selection for this position. The POS heatmap (g2) indicates that the word at this position acts as adjective (ADJ).

The heatmap colors can be mapped directly to the background of the matches (Figure 1h) as a simple method to reveal pattern across results (Figure 1f). Based on human identifiable patterns or alignments, the matching and mapping methods can lead to further data analysis or a refinement of the current hypothesis.

5.4 Design Considerations and Limitations

LSTMVis operates around as a bottom-up approach starting from a seed hypothesis and searching for similarities across the whole dataset. During the project, we experimented with top-down approaches following the Shneiderman mantra (overview first – zoom and filter later). We found that global overviews, such as projections of hidden state values or summary statistics, did little to reveal interpretable patterns on large datasets. LSTMVis differs from related work, in that instead of working with the prediction output, it operates directly on the hidden states to identify relationships. This design decision addresses our target group of architects and trainers who are familiar with model internals.

Another goal of our tool is to use a representation that is invariant to the ordering of hidden states in the hidden state vector while remaining scalable w.r.t. size of the vector. As addressed in Section 5.1, those two conditions are fulfilled by plotting hidden states individually onto parallel coordinates. However, even with this compact representation, applications with > 500 hidden states become challenging to analyze as a whole. Approaches to address the visual noise are to either filter the hidden states using our interactive methods or to use algorithmic preprocessing, like dimensionality reduction techniques or pruning methods. After testing the latter automated filtering approaches, we decided against using them within the application. We found that if the tool loses its inherent connection to the data, results were less interpretable to the user.

5.5 Technical Design and Implementation

LSTMVis consists of two modules, the visualization system and the RNN modeling component. The source code with documentation and some example models are available at `lstm.seas.harvard.edu`.

The visualization is a client-server system that uses Javascript and D3 on client side and Python, Flask, Connexion, 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 (T5). For our experiments we use the Torch framework. We trained our models separately and exported results to the visualization.

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 use cases that demonstrate the general applicability of LSTMVis for the 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. Each opening parenthesis increases and each closing parenthesis decreases the nesting level, respectively. Numbers are generated randomly, but are constrained to indicate the nesting level at their position. For example, a string in the language might look like:

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

Blue lines indicates ranges of nesting level $\geq$1, orange lines indicate nesting level $\geq$2, and green lines indicate nesting level $\geq$3.

To analyze this language, we view the states in LSTMVis. An example of the cell states of a multi-layer 2x300 LSTM modelis is shown in Figure 7(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 nicely confirms earlier work that demonstrates simple context-free models in RNNs and LSTMs [9, 31].

6.2 Phrase Separation in Language Modeling

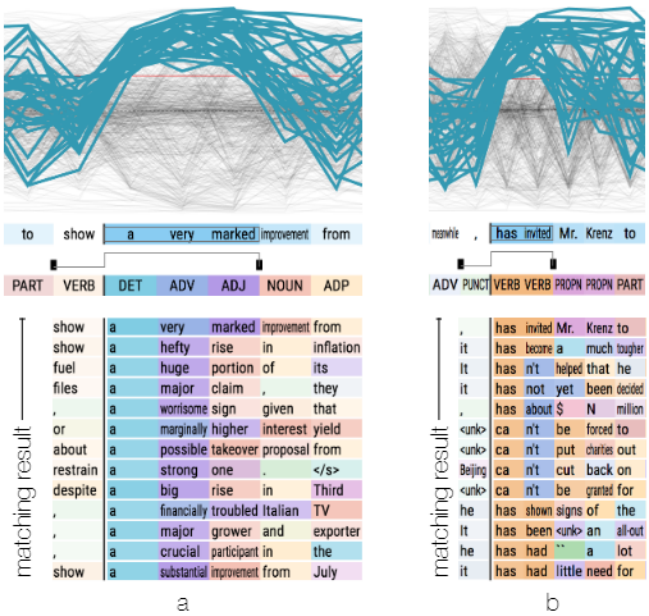

Fig. 8. Phrase selections and match annotations in the Wall Street Journal. (a) The user selects the phrase `a very marked improvement` (turning off when improvement is seen). The matches found are entirely other noun phrases, and start with different words. Note that here ground-truth noun phrases are indicated with a sequence of colors: cyan (DET), blue (ADV), violet (ADJ), red (NOUN). (b) We select a range starting with `has invited`. The results are various open verb phrases as sequence of colors orange (VERB) and blue (ADV). Note that for both examples the model can return matches of varying lengths.

Next we consider the case of a real-world natural language model from the perspective of an architect interested in the structure of the internal states and how they relate to underlying properties. For this experiment we trained a 2-layer LSTM language model with 650 hidden states on the Penn Treebank [21] following the medium-sized model of [33]. 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 linguistic annotations in the visual analysis 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 [29] for a subset of the treebank (Sections 15-18). 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 6.2 we show two examples of these selections and matches.

The visualization 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.

Fig. 9. 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.

6.3 Biological Sequence Analysis

RNN models are now commonly used for time-series analysis outside of the space of text processing. One area of interest is in biological sequence analysis for genomics where deep neural networks are directly applied to sequences of DNA for tasks such as classification and sequence labeling. We consider models trained for regulatory marker prediction, where neural networks are used to predict binding sites aligned to sequences of the genome, a task that has been recently explored with both CNN [35] and RNN models [24]. For this case study, we collaborated with a domain expert *trainer* who approached us after becoming aware of the release of our tool. He had employed a mixed RNN/CNN model that was trained over a genomic dataset made up of a 2.3 billion base pair long nucleotide sequence for this problem of regulatory marker prediction.

Notably this problem differs in several crucial ways from our previous use case: the granularity of the input is much smaller (base pairs as opposed to words), the training objective is different (0/1 binding site labels as opposed to the next word), and the problem is known to exhibit much longer term dependencies due to its latent 3D structure. Conversations around these issues led us to introduce several iterative features to our tool. These include the (a) ability to zoom in and out of the timeseries representation to adjust for granularity, (b) use of auto-scaling axes ranges to allow for switching between data of different activation ranges, such as CNN outputs, and most notably (c) ability to add/remove arbitrarily many annotation label along the word sequence timeline, to allow domain experts to view both the predictions and ground truth as an annotation track associated with the features at each time step. Figure 6.3 shows an example of the last feature which shows the predict binding locations for several different proteins along a region of DNA.

For this application the tool has so far been used primarily as a debugging assistant, providing a way for an expert in an ongoing research project to analyze the output of a trained model and make adjustments

based on mistakes of the model. For instance, in early use of the tool, the researcher noticed that one layer of the network was using very few of the available states, and another had significant artifacts from a poorly aligned convolutional layer. These observations have helped provide feedback for subsequent experimental design. Furthermore, recent work indicates an increasing interest in LSTMVis in the biomedical domain and for genomics (see Section 7).

Fig. 10. Biological Sequence Analysis. A selected region of the genome with seven different aligned annotation tracks. Each track indicates the 0/1 prediction of protein binding sites at each time step. Tracks can be activated/deactivated as part of the debugging process to compare predictions, ground truth, input properties, and other user-specified annotations.

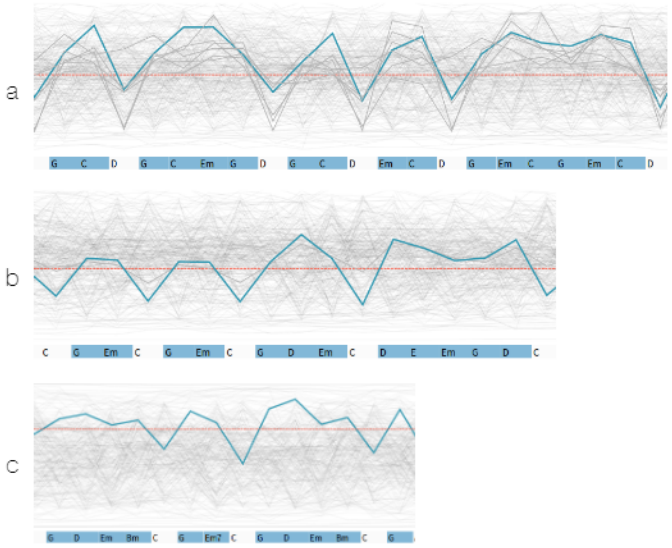

Fig. 11. Three examples of single state patterns in the guitar chord dataset. (a) We see several permutation of the very common I - V - vi - IV progression (informally, the "Don't Stop Believing" progression). (b) We see several patterns ending in a variant of the I- vi- IV- V (the 50's progression). (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/`.

6.4 Musical Chord Progressions

Past work on LSTM structure has emphasized cases where single hidden states are semantically interpretable. For text and biological 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 6.4, 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 Long-Term Case Study

Shneiderman and Plaisant [26] propose strategies for multi-dimensional in-depth long-term case (MILC) studies to evaluate information visualization tools. They observe that "scaling up by an order of magnitude in terms of number of users and duration of the observations is clearly beneficial." We decided to adopt their core ideas and report on qualitative feedback and quantitative success indicators after the open-source release of LSTMVis in June 2016.

We created a webpage and a video that introduces the core ideas of LSTMVis at `lstm.seas.harvard.edu`. The webpage provides an opportunity for users to leave comments. To advertise the tool online we followed a social media strategy that included collecting followers on Twitter, using broadcast media such as Reddit and Hackernews, and inviting editors of popular machine learning blogs to write guest articles.

The webpage contains a demo system with example datasets that allows us to acquire a large set of logging data. We noticed that users often try to reproduce the scenarios that are explained in the online video. To allow users to share insights from their exploratory analysis, we ensured that our URL parameter string contains all necessary information to replay and share a scenario.

We collected logging information about our webpage using Google Analytics. Within the first 7 days, the web page received ∼5,600 unique users, with 49% of traffic coming through social media and 39% arriving directly. The social media traffic was dominated by channels that we used to advertise the tool (Twitter 40%, Reddit 26%). To our surprise we also observed substantial traffic from channels that we did not contact directly (e.g., Sina Weibo 11%, Google+ 9%). After 300 days we recorded ∼19,500 page views with shrinking user traffic from social media (31%). At that point most users used search (22%) or accessed the webpage directly (39%). Only a small percentage (10%) of users tried the online demo. Most of these users used the datasets shown in the explanation video and did not explore further.

Our open source code release was stable yet simple enough to "...ensure that the tool has a reasonable level of reliability and support for basic features" [26]. For easy adoption, we provide user documentation that explains what data can be used and how to use the tool. We also provide convenience tools to prepare the data for import (Section 5.5). We asked students to act as testers for our source code. Based on their feedback we made several improvements to the installation process. For example, our prototype required NodeJS to resolve client-side dependencies. By providing the required libraries within our repository we removed the cumbersome step of installing NodeJS for our target audience. We observe that around 400 programmers liked the project (stars on GitHub) and over 95 practitioners copied (forked) the project to make custom modifications. We think that this demonstrate a reasonable interest in the system after only 300 days considering its highly specialized target audience.

Furthermore, we observe adoption of the tool for several documented use cases. Evermann et.al. [8] describe the application of LSTMVis to understand a trained model for a business process intelligence dataset. Liu et.al. [19] use LSTMVis in experiments to investigate language variants and vagueness in website privacy policies. We see an increasing interest to apply our tool for biomedical and genomic data. This is indicated by the use case we described in Section 6.3 or, e.g., in Ching

et.al. [5]

Besides the quantitative observations we also collected qualitative feedback from comments on the webpage, GitHub tickets (feature requests), and in-person presentations of the prototype. This qualitative feedback led us to make several changes to our system that were discussed in Section 5.

In retrospective, conducting a long-term user study benefited the project at multiple stages. Preparing the release of the prototype required us to focus strongly on simplicity, usability, and robustness of our tool. This lead to many small improvements to an internal prototype. The design iterations we inferred from user feedback strengthen the tool further. We think, that planning a successful long-term study requires four core ingredients: (1) reach out to your target audience by describing your approach (webpage, video) and inform them using social media, email, etc. (2) allow users to play with your tool by setting up a demo server, (3) allow engagement and experimentation with your tool by providing sufficiently documented, easily adoptable source code, and (4) make it as simple as possible for users to give you feedback via discussion forums, reported issues, or in person. During the study, we found it to be crucial to continuously engage with users and quickly take action based on their feedback.

8 Conclusion

LSTMVis provides an interactive visualization to facilitate data analysis of RNN hidden states. The tool is based on direct inference where a user *selects* a range of text to represent a hypothesis and the tool then *matches* this selection to other examples in the data set. The tool easily allows for external annotations to verify or reject hypothesizes. It only 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 several case studies of applying the tool to different data sets. Releasing the tool and source code online allows us to collect long-term user feedback that has already led us to make several improvements. In future work, we will explore how the wide variety of application cases can be adopted – beyond our imagined use cases and user groups. As example for such a use case, we got contacted by a highschool student using LSTMVis to learn about RNN methods. While we did not optimize for a learning scenario, we are now thinking about datasets and a blog publication that focus on learning. Another interesting future work is to support the user role of *end users* with simplified views on internals of RNN to help explaining specific model behavior.

Acknowledgments

This work was supported in part by the Air Force Research Laboratory, DARPA grant FA8750-12-C-0300, Oracle Research grant, and NIH grant U01CA198935.

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] T. Ching, D. S. Himmelstein, B. K. Beaulieu-Jones, A. A. Kalinin, B. T. Do, G. P. Way, E. Ferrero, P.-M. Agapow, W. Xie, G. L. Rosen, B. J. Lengerich, J. Israeli, J. Lanchantin, S. Woloszynek, A. E. Carpenter, A. Shrikumar, J. Xu, E. M. Cofer, D. J. Harris, D. DeCaprio, Y. Qi, A. Kundaje, Y. Peng, L. K. Wiley, M. H. S. Segler, A. Gitter, and C. S. Greene. Opportunities and obstacles for deep learning in biology and medicine. *bioRxiv*, 2017. doi: 10.1101/142760

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

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

[8] J. Evermann, J.-R. Rehse, and P. Fettke. Predicting process behaviour using deep learning. *Decision Support Systems*, pp. –, 2017. doi: 10.1016/j.dss.2017.04.003

[9] 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.

[10] 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.

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

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

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

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

[15] 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.

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

[17] 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.

[18] 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.

[19] F. Liu, N. L. Fella, and K. Liao. Modeling language vagueness in privacy policies using deep neural networks. In *2016 AAAI Fall Symposium Series*, 2016.

[20] M. Liu, J. Shi, Z. Li, C. Li, J. Zhu, and S. Liu. Towards better analysis of deep convolutional neural networks. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):91–100, 2017.

[21] 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.

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

[23] 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.

[24] D. Quang and X. Xie. Danq: a hybrid convolutional and recurrent deep neural network for quantifying the function of dna sequences. *Nucleic acids research*, p. gkw226, 2016.

[25] P. E. Rauber, S. G. Fadel, A. X. Falco, and A. C. Telea. Visualizing the hidden activity of artificial neural networks. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):101–110, Jan 2017. doi: 10.1109/TVCG.2016.2598838

[26] B. Shneiderman and C. Plaisant. Strategies for evaluating information visualization tools: multi-dimensional in-depth long-term case studies. In *Proceedings of the 2006 AVI Workshop on BEyond time and errors: novel evaluation methods for information visualization, BELIV 2006, Venice, Italy, May 23, 2006*, pp. 1–7, 2006. doi: 10.1145/1168149.1168158

[27] K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional

networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

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

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

[35] J. Zhou and O. G. Troyanskaya. Predicting effects of noncoding variants with deep learning-based sequence model. *Nature methods*, 12(10):931–934, 2015.