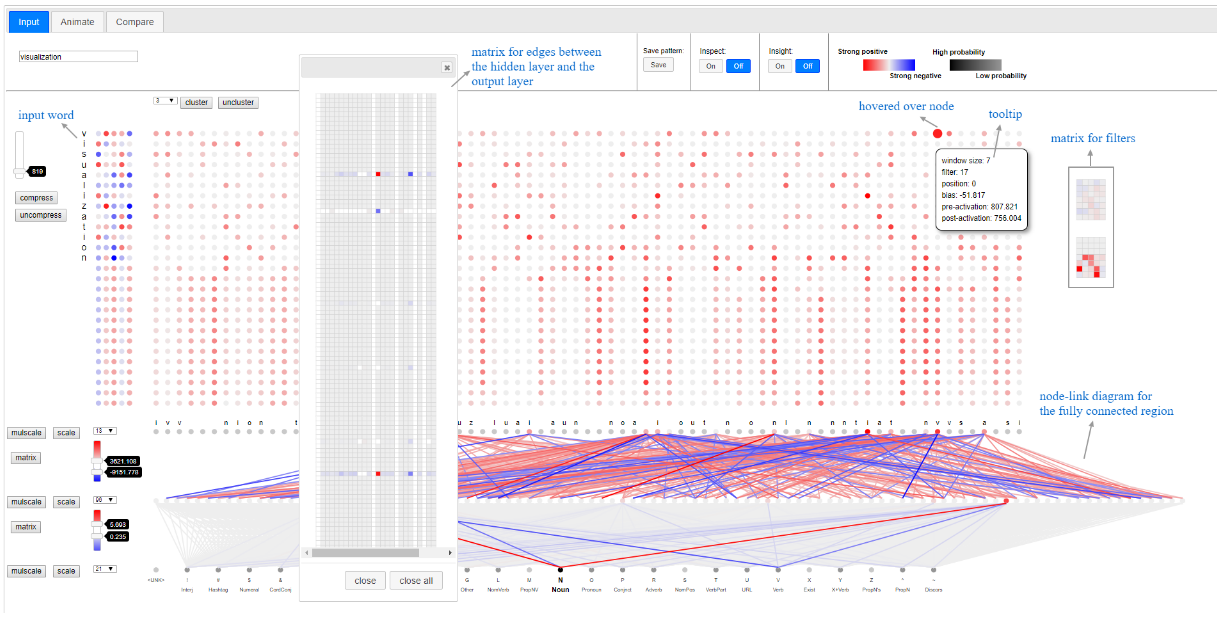
Visualizing Deep Neural Networks for Text Analytics

Shaoliang Nie   Christopher G. Healey*   Kalpesh Padia
North Carolina State University

Samuel Leeman-Munk   Jordan Riley Benson   Dave Ciara   Saratendu Sethi   Ravi Devarajan
SAS Institute



window size: 7
filter: 17
position: 0
bias: -51.817
pre-activation: 807.821
post-activation: 756.004

Figure 1: Overall structure of the visualization of a part of speech convolutional neural network model with *visualization* as input

Abstract

Deep neural networks (DNNs) have made tremendous progress in many different areas in recent years. How these networks function internally, however, is often not well understood. Advances in understanding DNNs will benefit and accelerate the development of the field. We present TNNVis, a visualization system that supports understanding of deep neural networks specifically designed to analyze text. TNNVis focuses on DNNs composed of fully connected and convolutional layers. It integrates visual encodings and interaction techniques chosen specifically for our tasks.. The tool allows users to: (1) visually explore DNN models with arbitrary input using a combination of node–link diagrams and matrix representation; (2) quickly identify activation values, weights, and feature map patterns within a network; (3) flexibly focus on visual information of interest with threshold, inspection, insight query, and tooltip operations; (4) discover network activation and training patterns through animation; and (5) compare differences between internal activation patterns for different inputs to the DNN. These functions allow neural network researchers to examine their DNN models from new perspectives, producing insights on how these models function. Clustering and summarization techniques are employed to support large convolutional and fully connected layers. Based on several part of speech models with different structure and size, we present multiple use cases where visualization facilitates an understanding of the models.

*e-mail: healey@ncsu.edu

1 Introduction

Recently, deep neural networks (DNNs) have achieved remarkable results in various areas. In image classification, multi-column DNNs [31] improved the state of art on several common image classification benchmarks, such as NORB and CIFAR-10. Krizhevsky et al. [17] trained a convolutional deep neural network (CNN), improving the record on the ImageNet 2012 classification benchmark. In speech recognition, feed forward DNNs outperformed traditional Gaussian mixture models on various speech recognition benchmarks [11]. In natural language processing, DNNs are being applied to a series of tasks: sentiment analysis for short texts [4], sentence modeling [12] and sentence classification [15]; producing improved results compared to the baselines. LeCun et al. [19] give a review on the achievements of DNNs in various fields.

However, as noted in [42, 43], our understanding of the internal behaviors of a DNN are less well developed. Various techniques to examine the internal logic of traditional learning methods [16] such as decision trees, naive Bayes and support vector machines are available. Unfortunately, similar techniques have yet to be fully developed for DNNs, which often function in a black box manner. An intuition and recognition of the working mechanisms of the model are often missing. As the models grow larger, it becomes more difficult to understand how the network produces its final results. Many researchers believe that each hidden layer represents features of the inputs, and that deeper layers identify more detailed features, but what the features are is unknown in most cases. Without a clear understanding of the working mechanisms, it becomes difficult to create and train new models, or to assign confidence to the results of these models for specific tasks. Better knowledge of how the DNN operates can identify new ways of training and improvement, making the models more robust and capable [28].

In the computer vision community, efforts are devoted to improve the understanding of CNNs for image-based tasks [8]. They provide insights on the working mechanisms of CNNs [43], and support the design of new models [26, 42]. In the text domain, several works [13,21,36] have been proposed for recurrent neural networks (RNNs). Much less attention is paid to CNNs in the text domain, however. Though RNNs are an effective choice for many text–based tasks, a large number of CNNs have been proposed to achieve comparable performance. For examples, Kim [15] developed several CNN models to improve the state of the art for sentence classification, Santos et al. [4] constructed a new CNN model to achieve high performance on sentiment analysis of short texts. Nguyen et al. [25] introduced CNNs that outperform existing models for relation extraction. LeCun et al. [44] proposed several CNN models for text classification and compared their performance with traditional methods and long-short term memory RNNs. The results demonstrated that the CNNs outperformed RNNs on all selected datasets.

Our work focuses on fully connected and convolutional neural networks in the text domain. We developed TNNVis to visualize *text-based DNN models*. Use of the term *DNN* in this paper refers to convolutional and fully connected DNNs. By carefully selecting and integrating multiple encodings and interaction techniques, TNNVis supports interactive exploration and discovery through a basic set of functions for visual representation, together with three operational modes to apply the functions under different conditions. The system is applicable to DNNs in different text domains, and is specifically designed to scale to larger networks. The main contributions of this paper include:

1. **Requirements Analysis.** We collaborated with several DNN researchers to design, refine and evaluate the visualization system. A list of requirements is presented in the paper.
2. **DNN Visualization.** Our system integrates node–link diagrams and matrix representations to provide a set of functions. These functions are used in multiple modes to explore DNNs and gain insights.
3. **Use Cases.** We apply our system to several DNNs with different structure and size to demonstrate how it can be used to facilitate understanding.
4. **Generality and Scalability.** We demonstrate how our system scales in both the convolutional and fully connected layers to visualize large DNNs.

2 Background

In this section, we briefly introduce the structure of fully connected and convolutional neural networks in the text domain, and the models we use to present our visualization.

2.1 Fully Connected Neural Network

Fully connected neural networks (FCNs) are also called *multilayer perceptrons*. These models consist of multiple layers. The first and last layers are the input and output layers, respectively. Every layer between the input and output layers is called a hidden layer. Layers are composed of a set of nodes or neurons. Every neuron in one layer is connected to every neuron in its adjacent layers. All neurons except those in the input layer contain bias, and every edge contains a weight. Neurons in the hidden layers and the output layer are assigned an activation function f. In the text domain, the inputs and outputs associated with neurons in the input layer are vector representations of characters, words or sentences. Given activations from a previous layer $l-1$, the weighted input to neuron j in layer l is defined as $z_j^{(l)} = b_j^{(l)} + \sum_k w_{j,k}^{(l)} a_k^{(l-1)}$, where $b_j^{(l)}$ is the bias of the target neuron, $w_{j,k}^{(l)}$ is the weight of the k_{th} edge connected to the neuron, and $a_k^{(l-1)}$ is the activation of neuron k in layer $l-1$. Once $z_j^{(l)}$ is calculated, activation is defined as $a_j^{(l)} = f(z_j^{(l)})$, where

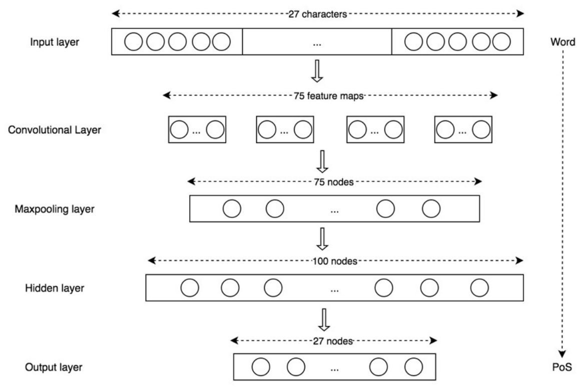


Figure 2: Part of speech CNN structure

f is an activation function. A common activation function is ReLU, which prunes negative parts of an input to zero and retains positive parts. In the output layer, a softmax function is applied to calculate the probability of each possible output, producing the normalized value of the exponential of the activations in the output layer:

$$\text{ReLU}(x) = max(0,x); \quad \text{softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}} \qquad (1)$$

2.2 Convolutional Neural Network

An FCN and a convolutional neural network (CNN) share many similarities. They are both layered structures with nodes and edges, and the activations of nodes are calculated in the same way, so the definition and description of how activations are calculated in an FCN directly apply to a CNN. The major difference is how adjacent layers are connected.

While adjacent layers are fully connected in an FCN, they are locally connected in a CNN. Every node in layer l is locally connected to a number of n nodes in layer $l-1$. Moreover, every such local connection uses the same set of n weights. This local connection is called a *filter* of size n, since it can be seen as a filter convolving through one layer to produce the next. The layer produced by this convolutional operation is called a *feature map*, and layers connected in this way are called *the convolutional layer*.

In a CNN, the input layer is typically connected to multiple filters with different sizes in parallel, producing multiple feature maps. Then, a maxpooling operation is applied to each feature map to select its maximum value. The maximum values from all the feature maps are concatenated to form a *maxpooling layer*. The maxpooling layer acts as an input layer to several fully connected hidden layers and an output layer. In this way a CNN normally includes an FCN component after it completes its feature map construction.

2.3 Part of Speech Neural Networks

The DNNs we use to develop and present our system are part of speech (PoS) FCNs and CNNs built by our deep learning collaborators, loosely based on the text normalization model in [20]. Although our system supports both FCNs and CNNs, we use CNNs to present our system. A CNN includes an FCN component, so visualization of an FCN can be illustrated in the visualization of a CNN. Although our PoS model has a specific number of filters in the convolutional layer, a specific number of fully connected layers and a specific number of neurons in the fully connected layers, any of these values can be changed, since our system generalizes to arbitrary DNNs.

The PoS CNN is a classification model, taking a word as input and estimating its part of speech (Fig. 2). The CNN's input layer accepts words of up to 27 characters with 5 *embedded weights* assigned to each character, forming a five-node vector representation of the character. This produces a a total of 135 nodes. If an input word's length is less than 27 character, it is padded with blanks. The input layer is followed by a convolutional layer with 75 filters producing 75 feature maps. Next, a maxpooling layer identifies the 75 largest feature maps values, concatenates them, and feeds them as input to a hidden layer with 100 nodes. The output layer following the hidden layer has 27 nodes, representing 27 candidate parts of speech. The maxpooling layer, the hidden layer and the output layer form the fully connected region of the CNN. An ReLU activation function is applied to the convolutional and hidden layers, while softmax is applied to the output layer.

3 Related Work

Various systems exist to visualize artificial neural networks (ANNs). Tzeng et al. [38] displays neurons as nodes and uses color and size to represent multiple statistics such as activation values. A 3D interactive prototype was developed to show a CNN's activations for a given input in Harley [9]. More recently, a prototype called TensorFlow Playground [34] was developed by Google's TensorFlow team to visualize training parameters and allow users to explore the internal behavior of a model processing a given instance. Currently, however, these systems do not appear to scale to large DNNs.

3.1 Heatmap Visualizations

A number of techniques are presented to interpret CNNs in the computer vision community. A taxonomy is proposed to organize these techniques into three categories [8]. *Input modification methods* modify the input and measure the resulting change in the hidden or output layers [43]. *Deconvolutional methods* employ different approaches to trace and determine the contribution of one pixel of the input image by starting with the target node of interest and iteratively calculating the contribution of each neuron in the next lower layer to the target node [33,43]. *Input construction methods* construct an artificial image by maximally activating a target node of interest [24]. The resulting visualization of these techniques is either a heatmap imposed on a given image or an artificial image. The focus of these methods is to compute useful numerical results, with a simple visual artifact to present the results. As these techniques are specifically designed for computer vision, their usefulness relies on recognizable features typically present in images.

In the text domain, Li et al. [21] adapted deconvolutional methods from the vision community to analyze the contribution of each word in an input sentence to an output classification node in a text classification model. The result is visualized as a saliency heatmap. Karpathy et al. [13] directly imposed a heatmap to an input text sequence based on the activation value of a target node. This highlights the presence of nodes that are potentially responsible for detecting newlines and parentheses. While this work gives intuition as to how each part of the input is associated with a target node, the structure of a DNN and its internal mechanisms remain unstudied.

3.2 Point-Based and Network-Based Visualizations

Point-based visualizations refer to techniques that reveal relationships between DNN components, such as neurons or learned representations, by using scatterplots [23]. They typically employ dimensional reduction techniques such as PCA [41] and t-SNE [40] to transform components into 3D or 2D vectors represented as points. Rauber et al. [27] visualized the learned representations of test samples of a DNN using t-SNE. This provides evidence for the hypothesis that the DNN has learned representations useful for classification. Other point-based systems include Google's *Embedding Projector* [35] and ReVACNN [3], which employ dimension reduction to visualize high dimensional representations. Point-based techniques are useful to present the relationships among a number of components in a DNN, but they cannot reflect the network's structural information, and so provide no comprehensive presentation of how nodes relate to and interact with one another.

On the other hand, network-based visualizations expose the network structure [23]. CNNVis [22] models a DNN as a directed acyclic graph and combines several techniques to scale to large DNNs. Clustering is used to group layers and neurons, while edge bundling is used to group edges. This identifies important image features extracted by sets of neurons, and displays the connections between clustered layer groups. It uses precomputed clusters of nodes and layers. Once the summarized view is formed, it has limited flexibility and interactivity. Our system uses a different strategy to address scalability by adopting interactive summarization and expansion of each layer, making it flexible to scale an arbitrary part of a DNN on demand. Furthermore, CNNVis does not allow studying a DNN's behavior by feeding single instances, subsets, or comparison of different inputs, while our system supports all of these types of analysis. ActiVis [14] is another interactive system that integrates several coordinated views to allow instance-based and subset-based exploration of DNNs. It provides views for *model architecture*, *neuron activation* and *instance selection*. ActiVis uses computational graphs to present the structure of DNNs. Although computational graphs are effective to show overall structure, they hide information such as the number of nodes in a layer. A region of the graph must be selected in ActiVis to show activated nodes. Our system, on the other hand, presents more detailed information along with the overall structure. Both ActiVis and our system support subset-based analysis by showing the average activations for a subset. But our system also animates the aggregating process, making it possible to observe if a DNN has a stable response to a subset. Comparison of different inputs are also supported both in ActiVis and our system. Since ActiVis only allows the inspection of details on a selected part of the computational graph, comparison is locally restricted. Our system allows global comparison to identify differences at different locations simultaneously.

A final visualization is LSTMVis [36]. It focuses on recurrent neural networks (RNNs). The hidden states of an RNN are visualized as parallel coordinates. A user can interactively choose a subsequence of the input to formulate and validate hypotheses. Although we are extending our system to include RNNs, the work discussed in this paper is focused on FCNs and CNNs, and so is not comparable to LSTMVis.

4 Design Process

Our design process aligns with the methodology proposed by Sedlmair et al. [32]. We first identify and confirm collaboration with three DNN experts in a precondition phase. During the core phase, we hold biweekly meetings with the experts to learn about their domain, to characterize problems, and to refine requirements. Based on the requirements, we design and implement data abstraction, visual encoding and interaction. Requirements analysis, design and implementation are essentially iterative. One informs and provides feedback to another. Our biweekly meetings continued throughout the collaboration to support the iterative process. This paper is the result of the analysis phase, which reflects on the design process and produces conclusions. The following is a list of requirements produced from our study.

R1. Structural overview. An overview presents the structure of a DNN, including the number of layers and nodes in a layer. All DNN experts commented that an overview is a starting point for understanding. It provides a useful visual and mental representation to guide analysis.

R2. Behavior. According to the DNN experts, the behavior of a model is reflected by the activation values of nodes and edges. To observe how a DNN behaves, it is important to present activation information. As activation values are calculated, the ability to examine results at each computation stage is helpful to understand how the model produces a final activation value.

R3. Region identification. To develop insights into the internal

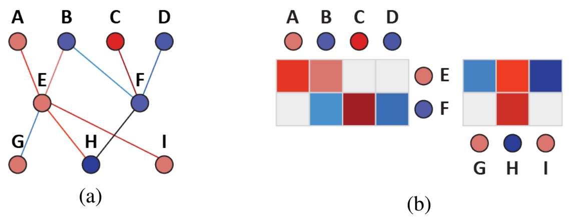


Figure 3: Graph representation: (a) node–link diagram; (b) quilt, each pair of levels in the graph produces a matrix whose cells encode edges between nodes

mechanism of a DNN, it is necessary to examine a network in detail. Based on the overview, DNN experts require the ability to focus on specific regions such as a subset of layers or certain nodes in a layer. This includes a requirement to identify. Before focusing on a specific region, effective mechanisms are needed to identify regions of interest.

R4. Instance-based analysis. Instance-based analysis tracks how an instance moves through a model. This approach is widely-applied in a number of neural network visualizations [9]. Users normally hold assumptions about the instance they choose, then validate the assumptions by observing how a model behaves. The DNN experts noted that during instance-based analysis they often switch back and forth between two instances to observe any differences. It is thus desirable to present comparisons in a single view.

R5. Subset-based and pattern analysis. Instance-based analysis is inefficient for finding patterns. A DNN expert's effort to try different inputs in the hope of discovering regularity in a model's behavior can be futile, since there is no guarantee such regularity exists. Subset-based analysis is similar to instance-based analysis, except that the focus is to study how a model behaves for a subset of instances, often grouped by common features [14]. By carefully designing the techniques to perform subset-based analysis, it is possible to "force" a model to respond in a way that exposes patterns.

5 Visualization Prototype

Our visualization system, TNNVis, is motivated by the list of requirements. TNNVis uses two visualization techniques: node–link diagrams and adjacency matrices, to visualize the structure of a DNN (**R1**). It offers a set of basic functions. *Coloring* encodes the activation values and visually guides users to specific regions (**R2, R3**), *thresholding* and *inspection* reduce visual clutter and allow focus on certain areas (**R3**), and *insight* and *tooltips* give more detailed information nodes' behaviors (**R2, R3**). These functions can be applied in three operational modes: *input*, *animate* and *compare*, to provide instance-based and subset-based analysis as well as techniques to expose patterns (**R4, R5**). The following sections discuss these aspects in more detail.

5.1 Prototype Layout

A neural network is naturally represented as a graph. Different visualizations are proposed for graphs [10, 18]. Node–link diagrams and adjacency matrices are considered the most effective approaches [10]. Node–link diagrams are widely used, representing the structure of a network in a natural and intuitive way. Nodes are represented as circles and edges as line segments connecting pairs of nodes.

Adjacency matrices [2] are an alternative visualization technique to visualize edges as matrices where each cell in a matrix represents an edge. Based on adjacency matrices, quilts [1] were proposed to visualize layered networks. A quilt is formed by arranging nodes into layers and concatenating the visualization of layers in a zigzag manner. Fig. 3 visualizes a simple network structure with a node–link diagram (Fig. 3a) and a quilt (Fig. 3b). While node–link diagrams are intuitive, they suffer from edge overlap and occlusion as networks grow larger. Adjacency matrices and quilts have no such issue

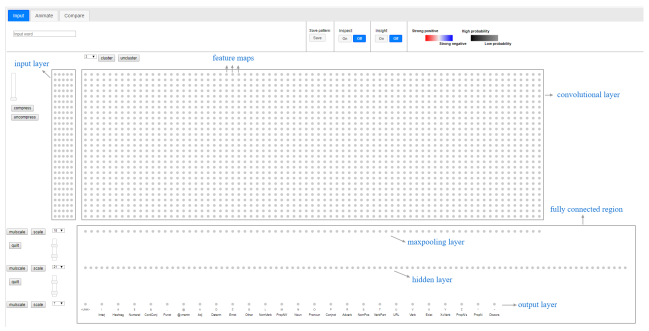


Figure 4: Structure of the CNN PoS visualization prior to input

since each edge is represented in a separate cell. To effectively visualize the structure of a DNN while supporting pattern discovery, we chose to combine both techniques: matrices and quilts to highlight patterns in the edge structure and node–link diagrams to provide a more holistic visual representation of the network [7]. For any DNN, a node–link diagram displays the overall structure. For small DNNs, a quilt is displayed as an alternative visualization by default. For large DNNs, users can request the matrix representation of edges between any pair of adjacent layers.

To layout a DNN, multiple techniques were considered. Widely used graph layout algorithms include force directed [5] and layered [37]. Force directed layouts position nodes by running a physical simulation with repulsive force assigned to all nodes and attractive force assigned to connected nodes. Layered layouts manually assign nodes to layers and position the nodes based on their layers. DNNs have an inherent layered structure. Force directed layouts can break this structure, while manual layout is not needed since layers are present by definition. A DNN's inherent layered structure leads to a straightforward layout by positioning layers along one axis and nodes in a layer along the other on a 2D plane (**R1**).

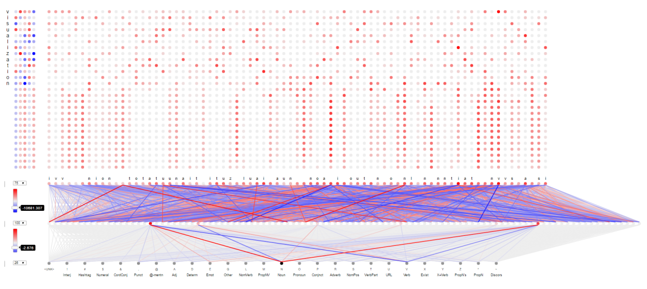
Fig. 4 shows annotations of the layout of the PoS CNN presented in Section 2.3 prior to input. The input layer is on the left, and the convolutional layers are on the right. Each column in the convolutional layer corresponds to a feature map. The maxpooling layer is below the convolutional layer, with nodes aligned with their corresponding feature maps. Under the maxpooling layer is a hidden layer, then the output layer.

Fig. 1 shows the system when an input is fed into the DNN. The input characters are displayed to the left of the input layer, aligned with their five embedded weights (embeddings). In the convolutional layer, feature maps are independent of one another. No links exist between them. Hovering over a node in the convolutional layer displays the filter applied to produce the node's activation value visualized as two matrices to the right: a filter weight matrix (upper matrix), and a convolution matrix produced by multiplying the filter weights with the embedded weights for the current node (bottom matrix). The three layers below the convolutional layer form a fully connected network, visualized as a node–link diagram. For the edges of any two adjacent layers, a user can choose to see the matrix representation by clicking on the *matrix* button.

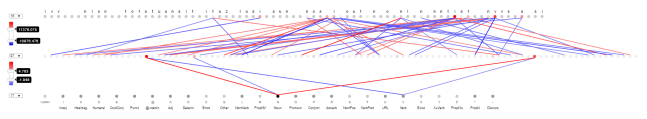
5.2 Basic Functions

The system provides five basic functions: coloring to represent node and edge activation values, thresholding to restrict information being visualized, inspection to investigate a single node's incoming and outgoing edges, insight query to determine which input components most strongly activate a node, and tooltips to expose detailed information about a node's or edge's weight, pre and post-activation, and bias values. Coloring and tooltips are available in the convolutional and fully connected regions of the system; thresholding, inspection and insight query are exclusive to the fully connected region.

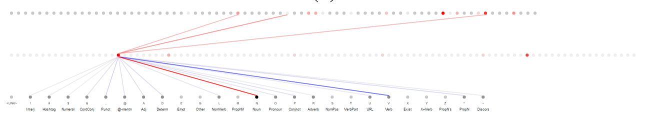
Coloring. The behavior of a DNN is reflected by activation values.



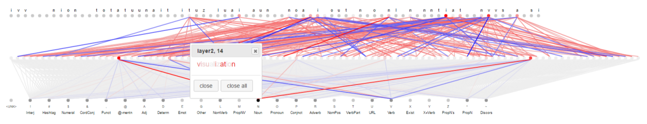
(a)



(b)



(c)



(d)

Figure 5: Visualizing *visualization*: (a) color scheme; (b) node–link thresholding to control which nodes and edges are visualized; (c) inspection of a node's incoming and outgoing edges; (d) query of input characters and embedding(s) that most strongly activate a node

We use a color scheme to allow observation of model behaviors and guide users' attention (**R1, R3**). To present activation values, different visual channels that encode quantitative information were considered. Position is unavailable since it is used in the layout. Size becomes less effective for large networks since size can only vary within a small extent while displaying the entire network. We thus chose to use a red–blue color scale to encode activation values. Fig. 5a shows the word *visualization* visualized in the PoS model. A node or edge is colored blue if its activation value is negative, or red if it is positive. More saturated colors represent larger activations. We chose this method to highlight nodes and edges with high impact in the model, while reducing the visual influence of those with activations near zero. In the output layer, greyscale is used to represent class probabilities, darker for higher probabilities.

Consider Fig. 5a as an example. All nodes and edges are colored based on their activation values. The value for nodes in the CNN's input layer are the embedding values obtained from the trained PoS model. The value of a node in the convolutional layer is the result of an ReLU operation applied to the sum of its convolution matrix entries plus a bias associated with the node. Nodes in the maxpooling layer, which act as input to the fully connected portion of the CNN, store the maximum activation values from their associated feature map columns. In the fully connected region, an edge's activation value is the product of its source node's activation and the edge's weight. A node's activation value in a hidden layer is the sum of all incoming edge activations, passed through an ReLU function. Finally, the activation value of nodes in the output layer are the sum of all incoming edge activations, normalized to represent the probability of the given output.

Thresholding. In the fully connected region of the DNN, visual clutter accumulates when many nodes and edges are visualized. This makes it difficult to locate potentially useful information, thus impeding effective exploration of a DNN's behavior or draw attention to specific regions. To reduce visual clutter and allow users to focus on relevant details, we provide thresholding on both nodes and edges (**R1, R2**). The node threshold controls how many nodes are colored. TNNVis ranks all nodes by absolute activation, then colors nodes up to the threshold number. The remaining nodes are drawn in light gray. Only edges between colored nodes are visualized. Two edge thresholds are also available for each set of edges: a maximum threshold and a minimum threshold. Edges with activation values above the maximum or below the minimum will be displayed. Each layer of nodes and edges has separate threshold controls. The thresholds act on the matrix representation and the node–link diagram simultaneously. Fig. 5b shows the same example as Fig. 5a with thresholding activated, reducing the number of colored nodes and edges, and the corresponding visual clutter.

Node Inspection. While thresholds are effective to filter information, our DNN collaborators expressed the need to focus on individual nodes. This usually happens when their attention is led to a few nodes by the coloring scheme and thresholding that they want to study in isolation. We thus provide node inspection to support this need (**R3**). During inspection a user selects a node of interest. The visualization retains only the incoming and outgoing edges for the selected node. Node inspection acts on the matrix representation and the node–link diagram simultaneously. Thresholds cannot be changed during inspection. Once inspection is completed, the visualization returns to its original state. Fig. 5c shows the same example in Fig. 5b with thresholding and inspection. At this level of detail, differences between a node's incoming and outgoing edges can more easily be compared. This assists users in determining the contribution of each incoming edge to the final output of a node, as well as the influence of a node on its outgoing edges.

Insight Query. In addition to activation values, the association of activation and input also reflects the behavior of DNNs. Similar ideas are explored in the vision community by tracing from a node to an input [8]. We provide insight query to relate node activations to the textual input information (**R2, R3**). An insight query is similar to the image patches presented after computing learned features of a neuron in CNNVis [22]. For our implementation, patches correspond to text. Insight query determines which input characters and corresponding embedded weights most strongly activate a target node. We first identify an input character for each node in the maxpooling layer by backtracking the mostly highly activated node a_{max} in its corresponding feature map, then selecting the input character centered over a_{max}'s filter. Once each node in the maxpooling layer is assigned a character, selecting a node in the hidden or output layer of the model presents a dialog identifying which input characters (i.e., which maxpooling characters) most strongly activate the node. This is done by following highly activated edges back to the maxpooling layer. The dialog displays characters that activate the node using saturation to represent activation strength. This allows users to gain insight into which part(s) of the input a node responds to, providing intuition into the purpose of the node. Fig. 5d shows the same example in Fig. 5b with a high activation node selected from the output layer. The resulting dialog shows the node responds most strongly to *v*, *a*, *t* and *n*.

Tooltips. Activation values are calculated in several steps. It is important to provide the intermediate calculation results when requested, especially when inspecting nodes with unexpected activations. We provide this detailed information using tooltips (**R3**). Nodes in the input layer show each node's position and embedded weights. Nodes in the convolutional layer show the size of the applied filter, a node's position in the feature map, its weight and activation. Nodes in the maxpooling layer show the node's position and its activation. Nodes in the fully connected layer show the sum of incoming edge activations, bias, and output as an ReLU operation applied to the edge activation sum plus bias. Edges in the node–link diagram or the matrix representation show the source node's activation, the edge weight, and the edge activation. Finally, nodes in the output layer show the sum of all incoming edges' activations, bias and normalized output as a classification probability value.

5.3 Operational Modes

The tabs on the left top of Fig. 4 show the system's three operational modes: input, animate and compare. The basic coloring, thresholding, inspection, insight query, and tooltip functions are available under each operational mode. One can see different operational modes as different ways to supply data to the visualization. Once the data is received, the basic functions are used to explore the network.

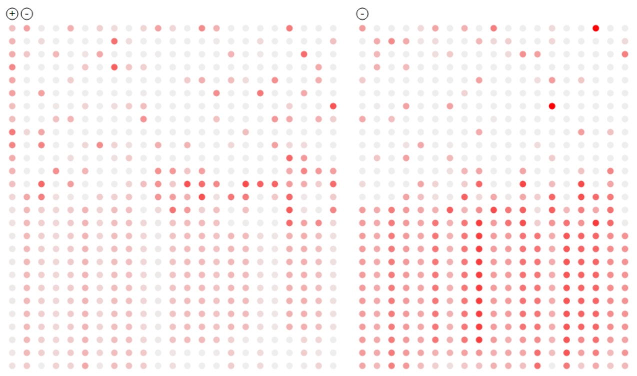
Input. Input mode supports instance-based analysis (**R4**). In this mode, a user provides an arbitrary input to a network model, and the activations of the model are calculated and visualized. A user can then explore and study the behavior and patterns of the network using the basic functions.

Animate. In animate mode, we provide three different types of animation. The first animates over a set of inputs, which can be specified by the user. For example, the set of inputs can be chosen to have the same classification. At each time step, an input from the set is fed to the system. The animation visualizes the average activations aggregated over all inputs to date. Our DNN experts expect certain nodes and edges to become highly activated during the animation. This allows users to determine whether the model converges to a stable pattern of activation for a set of inputs with common features or user assumptions.

The second animation takes an input and feeds it to a model at different training steps during the training process, animating how the model activates during training for the input. Our DNN experts expect the activations of the model to vary significantly in early steps, then settle into a more stable state in later steps. This allows users to observe how the model responds at different points in the training process. The third animation is similar to the second. The difference is that, instead of activations, it animates the gradients of the model's parameters, showing the evolution of the gradients during the training process.

The purpose of the first animation is two-fold. On the one hand, it allows subset-based analysis by exploring the visualization when the animation completes. On the other hand, users can determine whether a model behaves consistently to the input set by observing the animation. If the visualization stabilizes, it represents a stable behavior of the model to this type of input, exposing a useful pattern (**R5**). Cognitive load is a common concern for animation [6]. Because of this, we built our animations to ensure there are no structural changes to the network. Alternative approaches to visualize dynamic changes in a graph are available [18]. We considered time series in nodes [30], but it becomes infeasible due to space constraints. Small multiples [39] is another technique to visualize the model's progression with a set of inputs. Due to the large size of our models, it is only viable to display a few small multiples, which would exclude much of the information contained in the animation. To provide a level of analogous support for these alternatives, a user can save the visualization at any time step and later compare time steps in the *compare* mode.

Compare. Compare mode allows comparison between instances and subsets to further support instance- and subset-based analysis (**R4, R5**). A user can save any activation pattern to a data file that can be later retrieved and visualized. In compare mode, two files are selected for comparison. The user can visualize each file or their comparison, which is the activation difference between every node and edge in the DNN. This highlights nodes and edges that respond most differently. Comparison information is potentially useful in many cases, such as comparing two inputs from the same category, or comparing an average pattern to a single input pattern.


(a)

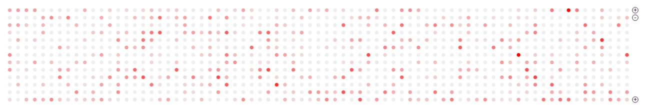

(b)

Figure 6: Clustering and compression: (a) the convolutional layer clustered into three clusters with two of them expanded; (b) the convolutional layer compressed into three groups with one of them expanded

6 Scalability

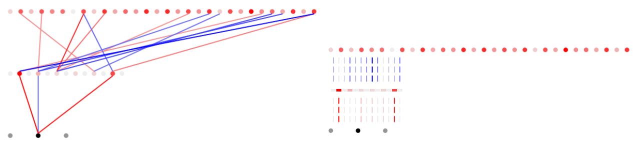
To address large DNNs, the system provides summarization techniques to scale and provide "details on demand" to allow viewers to focus their exploration. The main concern of designing and choosing scalability techniques is to preserve the network topology and value distribution as faithfully as possible, so the summarized view reflects the underlying detailed view effectively. We first use the current CNN model to illustrate the techniques, then demonstrate how it works for a much larger model. We describe how we scale the convolutional and fully connected layers separately since this is performed differently.

6.1 Convolutional Layer Summarization

For convolutional layers, the system provides clustering in the horizontal direction (*i.e.*, clustering feature map columns) and compression in the vertical direction (*i.e.*, merging of adjacent rows). Users can choose to cluster or compress independently, or perform both sequentially.

Clustering. The system uses k-means clustering combine the feature maps in convolutional layers. Users begin by choosing the number of clusters k. After clustering, the centroids of the clusters are visualized. Users can expand any cluster to see which feature maps belong to the cluster. This provides a summarized view of the feature maps, groups similar feature maps together and reduces the size of the visualization in the horizontal direction. Users can return to the full detail visualization using an *unclustering* button. We do not limit the number of clusters to provide the flexibility to experiment. Fig. 6a shows an example of clustering.

Compression. In the vertical direction, we refer to nodes spanning the feature maps and aligned to the same input character as a *component* in the convolutional layers. To compress, the system merges adjacent components as long as their Euclidean distance is below a chosen threshold. The average values of the merged components are visualized following compression. Users can choose the compression threshold to control the number of components that are merged. Similar to clustering, merged components can be expanded to see which components are in a given group. The full detail visualization can be retrieved by using an *uncompress* button. Fig. 6b shows an example of compression.


(a) (b)

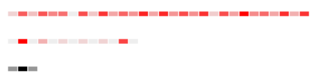

(c)

Figure 7: Node and layer summarization for the fully connected layers: (a) node summarization for the maxpooling, hidden and output layers; (b) layer summarization for the hidden layer; (c) layer summarization for all layers in the fully connected region

Clustering and compression can be used together to summarize in both the horizontal and the vertical direction. Since each feature map is produced independently, reordering in the clustering process is possible. Order matters for horizontal components, however, because they are aligned with input sequences. Because of this, we choose to compress adjacent components to preserve order.

6.2 Fully Connected Layer Summarization

For fully connected layers, our system provides node summarization to summarize nodes in one layer and layer summarization to summarize across layers.

Node Summarization. Nodes in a layer are summarized by merging groups of adjacent nodes that have activations that are either all below or all above the average activation of all nodes in the layer. A summarized node is colored based on the average of the original nodes' activations. The incoming and outgoing edges of a summarized node are colored based on the average of the original nodes' edge activations. As the node–link diagram is summarized, the matrices are summarized accordingly. This technique preserves the order of the nodes in a layer while still providing a summarized view of the nodes. Node and link thresholding are available after compression to further reduce the number of compressed nodes and links. One drawback of this summarization technique is that the amount of compression depends on the statistical variance of the activations across adjacent nodes. A suggestion from the DNN experts was to threshold the nodes prior to merging, thereby only retaining high or low activation nodes. Users can compress a single layer or multiple layers at the same time. Fig. 7a shows the fully connected region of the DNN when the maxpooling layer, the hidden layer and the output layer are all summarized.

Layer Summarization. For scaling across layers, users can interactively "activate" or "deactivate" individual layers or groups of layers. Active layers retain full detail: all nodes will be visualized, and edges between adjacent layers will be shown. Inactive layers will be reduced to a single line. During visual aggregation, it is important to provide appropriate summary information to allow viewers to choose effectively where to request additional detail. To this end, an inactive hidden layer line is subdivided into *n* subblocks, one per node. Each subblock is colored based on the activation of the node it represents. This allows viewers to examine an inactive layer's activation pattern, something our DNN experts emphasized was critical for their investigations.

Summarizing layers affects their edges. Given adjacent layers l_1 and l_2, the edges are displayed based on the layers' states.

- l_1 and l_2 are active: all edges between l_1 and l_2 are shown
- l_1 and l_2 are inactive: no edges are shown
- l_1 is active, l_2 is inactive: for each node $n_{i,2} \in l_2$, a single edge visualized as a dashed line is displayed representing the average weight of all edges from $n_{i,2}$ to all nodes in l_1

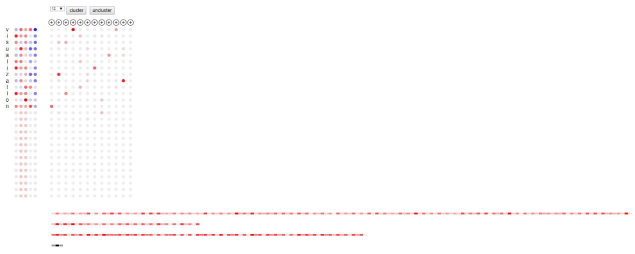


Figure 8: Summarized view of the large DNN

The vertical space between an inactive layer and an active layer can be reduced since the edges are summarized without overlap among them, thereby requiring less space to differentiate visually. The vertical space between two inactive layers can be further reduced as there are no edges between them. Node summarization can be performed before layer summarization to scale fully connected layers both horizontally and vertically. In Fig. 7b, the hidden layer is summarized, while in Fig. 7c, the maxpooling layer, the hidden layer and output layer are all summarized. While these techniques summarize information to allow visualization of large DNNs, the network topology and layer structure are preserved, to effectively guide users' exploration through the summarized views.

6.3 Visualizing Large DNNs

To study scalability, we applied the visualization system to a large PoS CNN. The large model still takes input of 27 characters, each with 5 embeddings, but it has a much larger convolutional layer with 1200 feature maps. The corresponding maxpooling layer has 1200 nodes, followed by two fully connected layers each with 1000 nodes, followed by an output layer with 27 nodes representing candidate classifications. This model is comparable to the models proposed for the text domain in the deep learning community. The CNN for sentence classification proposed by Kim [15] has one convolutional layer with 300 feature maps, and the maxpooling layer is directly connected to the output layer. The PoS tagging CNN proposed by Santos et al. [29] has one convolutional layer with 50 feature maps and one hidden layer with 300 nodes.

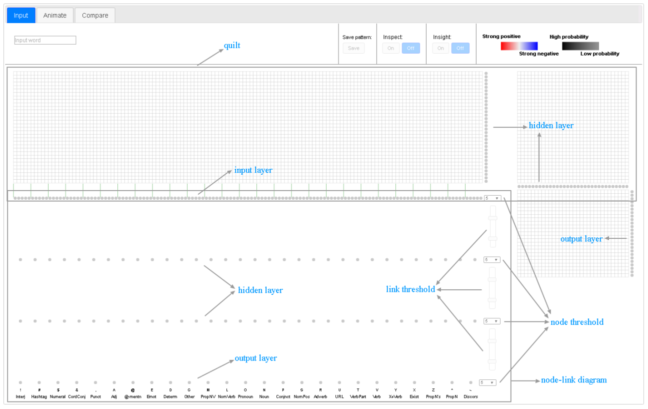
To reduce memory usage and accelerate the speed of rendering, we remove the nodes with zero activation in the fully connected region of the large model before rendering. Since nodes with zero activation carry no information, removing them does not exclude any important parts of the model. Before using scaling techniques, a large portion of the visualization is not visible since the size of the convolutional layer and the fully connected region exceeds a typical desktop monitor's resolution. After clustering the convolutional layer into 12 clusters and summarizing the fully connected region, the entire model is visible on screen (Fig. 8). The summarized view presents the overall structure of the model. Coloring preserves activation patterns. Based on the summarized view, users can interactively inspect the details of any part of the model.

7 Use Cases

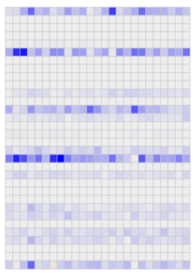
In this section, we demonstrate how our system helps experts understand DNN models by presenting several use cases. The visualization is applied to three models with different size and structure. The same experts are involved in every use case. Each scenario demonstrates how the experts employ the visualization system to make discoveries within a model. These findings pointed to a number of potentially useful future directions for experts to examine. They noted that these directions may have gone unnoticed without the visualization tool.

7.1 FCN PoS Model Characteristics

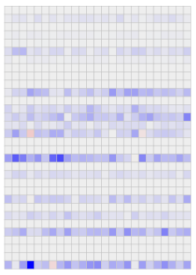
In this use case, the experts aim to study whether the coloring scheme combined with instance-based analysis reveals patterns of a model's



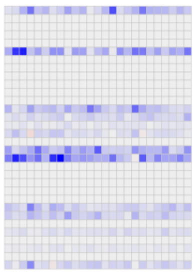
(a)



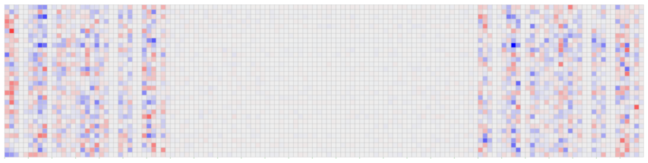
(b)



(c)



(d)



(e)

Figure 9: FCN PoS model: (a) overall structure; (b–d) link colors hidden → output layer for *visualizing*, *computation* and *ideally* respectively; (e) link colors input → hidden layer for *visualizing*

behavior. They use a fully connected feed forward PoS model. Fig. 9a shows the overall visualization of the model. The system allows the experts to display quilts by default when sufficient space. By trying different inputs in the input mode, certain characteristic of the FCN PoS model were uncovered. For every input, the edges between the hidden layer and the output layer are mostly blue, which indicates negative activation for those edges. Fig. 9b–d show the quilt visualizations between the second hidden layer and the output layer for *visualizing*, *computation* and *ideally* as example inputs, respectively. This clearly contrasts with the edges from the input layer to the first hidden layer, which are a mixture of positive and negative values (Fig. 9e with *ideally* as input).

Since an ReLU function is applied to the hidden layer, the node activations of the layer are non-negative. An edge is colored based on the multiplication of its weight and its source node's activation, so this means that most of the edges between the second hidden layer and the output layer have negative weights. Why is this the case? The DNN experts noted that mathematically, the model uses a softmax function in the output layer to calculate probabilities (Eq. 1). Based on this equation, there is no requirement that the edge weights are negative, but it does lead to an interesting insight by the DNN experts: negative weights represent a reversal of how they assumed the DNN functioned. Rather than collecting features representing a given class to encourage selection, the DNN seems to be collecting features that represent the *absence* of a class to *discourage* selection, similar to a "process of elimination."

This example demonstrates the usefulness of the red–blue coloring scheme, as well as the matrix representations to quickly detect color distribution patterns [7]. For a general neural network model, experts can identify the distributions of positive and negatives values of the nodes and edges. Areas with concentrated red or blue draw attention, identifying potentially important findings to pursue.

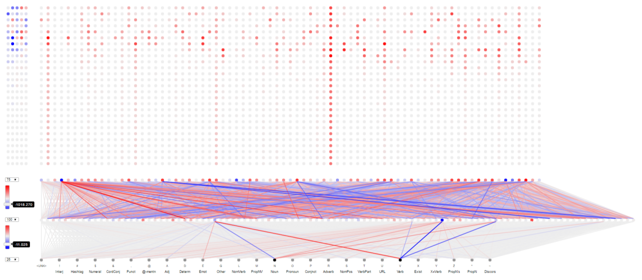


Figure 10: Noun versus verb differences in the CNN PoS model

7.2 Stable Patterns

In this use case, rather than trying individual instances, the experts intend to use the *animate* mode to perform subset-based analysis to expose patterns. A set of 100 correctly classified nouns were chosen as an input set to animate the FCN PoS model in the previous use case, while a set of 100 correctly classified verbs were chosen as an input set to animate the CNN PoS model presented at the beginning of the paper. For both models, the visualization has a noticeable number of fluctuations during the first 30 to 50 time steps, but stabilizes afterwards.

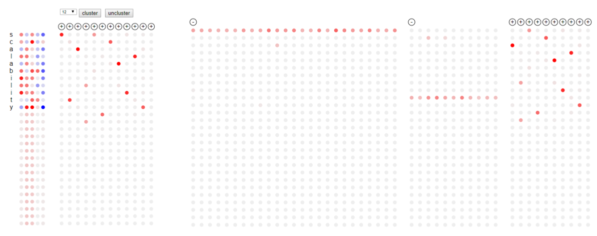
In the stabilized visualizations, the highly activated nodes and edges remain highly activated while low activation nodes continue to exhibit low activation. Experts commented that the convergence to a stable visualization exposes the fact that a common set of highly activated nodes and edges is responsible for assigning the set of inputs to a particular class. It represents a signature of a model's response to that class, exposed by the model's color encoding. Experts can use this as a starting point to further study the distribution and structure of the signature.

Animation is a general function as long as a set of inputs is specified. While inputs from the same class are a representative choice, there are many other possibilities. For example, words ending in *ly* can be used for word-based models, or sentences sharing words with strong sentiment can be used for sentiment classification models. The convergence to a stable visualization can reveal the most responsive nodes and edges for a given set of inputs. Failing to converge has several possible explanations. It may be that a model has not learned the feature chosen by an expert to group the input set, or that a model is not trained well enough to respond consistently to obvious features, or that a model simply reacts with a different subset of nodes and edges to each input from the input set (i.e., the input includes subclasses that the model and the visualization are exposing). In all cases, animation can confirm the existence or absence of a stable pattern, pointing to further directions to investigate.

Similar to aggregated activation, gradient animation and training animation can be applied. Stable patterns also emerge as the training proceeds. Gradient animation shows overall decreasing of gradient values and reveals the nodes with large gradients. We include several videos in the supplementary material to demonstrate these animations.

7.3 Noun–Verb Pattern Differences

In this use case, the experts seek to answer the question: How does a model respond differently to different PoS classes? The stabilized CNN PoS visualizations formed by 100 nouns and 100 verbs are saved and compared (Fig. 10). Highly saturated colors represent large activation differences, while gray represents small differences.



(a) (b)

Figure 11: A large CNN: (a) feature map patterns with 12 clusters using *scalability* as input; (b) first two clusters expanded

More highly saturated nodes are present in the maxpooling layer than in the hidden layer. Moreover, numerous highly saturated edges exist between the maxpooling and the hidden layer, but only a few occur between the hidden and the output layer. These noun–verb activation differences between involve more nodes in the maxpooling layer. Experts inferred the model has learned a more concise, higher level representation of the inputs after the hidden layer, requiring only a small number of nodes to differentiate one class from another. This corresponds to the general belief in the deep learning community that a DNN learns more abstract and higher level representations as the depth of layers increases. Experts anticipated that, for DNNs with more hidden layers, the number of highly activated nodes and edges should decrease as layer depth increases. If a model has a large number of highly activated nodes and edges in the final layers, increasing its depth might improve its performance. This approach can be used to diagnose, refine, and test a model.

7.4 Large CNN PoS Characteristics

To validate the system on large models, the experts applied the visualization to a large CNN PoS model introduced in the section on scalability (Section 6), which reveals several findings. By testing multiple inputs, then clustering the feature maps into a number of clusters approximately equal to the length of the inputs, experts found that the centroid of most clusters has one and only one highly activated node. Moreover, the node in each centroid is centered at a different character in the input word. This is in contrast to the CNN PoS model with a smaller number of features maps, which has a more distributed pattern (Fig. 6a). Expanding the clusters shows that most feature maps in the same cluster have the same pattern as the centroid map. Experts concluded that in the large CNN, every feature map is dedicated to feature extraction at a single position in the input sequence. This is probably due to the large number of feature maps, allowing each feature map to focus on one character. Furthermore, there is always one centroid with no highly activated node. Expanding the cluster shows that the majority of the feature maps fall in this cluster, suggesting the majority of the feature maps have zero activation and do not directly contribute to the result of the model. Thus, 1200 feature maps are likely more than needed for part of speech tagging. Using less feature maps might achieve the same performance, while reducing model size and training time. Fig. 11 show this observation with *scalability* as an example input.

8 Discussion and Limitations

While our visualization system is demonstrated with PoS models, it is applicable to other tasks in the text domain. Based on feedback from our DNN collaborators, the system generalizes to other types of models by replacing the final classification layer. For sentiment analysis, the final layer would predict -1, 0 or 1 for negative, neutral, or positive sentiment. Since the general structure of the model remains consistent, the visualization system can be directly applied. As research on deep learning accelerates, however, novel model architectures may emerge that do not fit into the structures we currently support. Though ideas such as coloring, thresholding, insight, animation and comparison can be applied with little modification, new layouts need to be designed for new architectures.

To visualize large DNNs, rather than initially presenting a detailed view, we can provide a summarized visualization by default. Users can choose to expand and explore the details of any region they find interesting. For animation, we would animate the summarized visualization. If DNNs stabilize in the full detail visualization, they will also stabilize in the summarized visualization. For comparison, we can again present the summarized visualization by default. However, it can become less effective for users to identify interesting regions of a model when layers exceed a certain threshold. Selectively hiding parts of a model or automatically suggesting regions of interest can be considered.

To render large DNNs in a reliable and responsive manner, preprocessing becomes necessary. For the convolutional layer, this is not an issue since a single weight matrix is shared for each feature map and the connections are sparse. But for fully connected layers, the number of edges is the square of the number of nodes between adjacent layers. Given several such layers, rendering the nodes and edges quickly exceeds the memory limits of a typical machine. To address this, we prune all nodes and associated edges with zero activation before rendering. The amount of pruning depends on a specific model. For our large model, it removes approximately sixty percent of the nodes and edges in the fully connected region. This heuristic usually preserves important information and proves to be effective at controlling the number of edges. But its effectiveness for different models requires further study.

9 Implementation

TNNVis is a server–client system that uses JavaScript, SVG and D3 on the client side, and Python, Flask and TensorFlow on the server side. Server and client communicate through multiple endpoints, using a predefined data format. The server can be built using any technology as long as it provides data in the specified format. The running time complexity of most operations, including animation and comparison, is linear with the number of nodes and edges in a DNN. Since scalability operations are performed on a selected layer, it runs in linear time with the number of nodes and associated edges in a layer. Clustering has the highest complexity $O(k \times d \times N)$ where k is the number of clusters, d the size of a feature map, and N the number of filters.

10 Conclusion and Future work

We present TNNVis, a visualization system to facilitate the understanding of DNNs for text analytics. TNNVis displays the structure of a DNN using a combination of node–link diagrams and matrix representations. Node and edge activations are encoded in red–blue, allowing rapid perception of value difference and distribution. Threshold, node inspection, insight query, and tooltips provide basic functions to control and explore information at various levels of detail. Visualizing an arbitrary input is performed in input mode; animating the aggregation of a set of inputs or the training process is performed in animate mode; and comparison of two activation patterns is performed in compare mode. This provides methods to visually explore and discover within a DNN, hinting at new and promising insights. Multiple summarization techniques are employed to generalize and scale to large DNNs. Several sample use cases are presented. The system helped identify basic model characteristics, stable patterns, and differences between patterns.

In the future, we plan to extend our system to support recurrent neural networks (RNN) [13, 36]. The global structure of an RNN is sequential and repetitive, which is largely different from FCNs or CNNs. Specific designs are needed to layout recurrent structures effectively. In the interim, our current approach can be employed to visualize the details of an RNN at any single timestep.

Acknowledgments

The authors would like to thank James Cox, Larry Lewis, and Mustafa Kabul of SAS Institute for their assistance participating as DNN experts during our system design and evaluation.

References

[1] J. Bae and B. Watson. Developing and evaluating quilts for the depiction of large layered graphs. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2268–2275, 2011.

[2] J. Bertin. *Sémiologie Graphique: Les diagrammes, lex réseaux, les cartes*. Gauthier-Villars, Paris, France, 1967.

[3] S. Chung, C. Park, S. Suh, K. Kang, J. Choo, and B. C. Kwon. ReVACNN: Steering convolutional neural network via real-time visual analytics. In *Future of Interactive Learning Machines Workshop*, 2016.

[4] C. N. Dos Santos and M. Gatti. Deep convolutional neural networks for sentiment analysis of short texts. In *COLING*, pages 69–78, 2014.

[5] P. Eades. A heuristics for graph drawing. *Congressus numerantium*, 42:146–160, 1984.

[6] P. Eades, W. Lai, K. Misue, and K. Sugiyama. Preserving the mental map of a diagram. Technical Report IIAS-RR-91-16E, Fujitsu, 1991.

[7] M. Ghoniem, J.-D. Fekete, and P. Castagliola. A comparison of the readability of graphs using node-link and matrix-based representations. In *Proceedings 2004 IEEE Symposium on Information Visualization*, pages 17–24, 2004.

[8] F. Grün, C. Rupprecht, N. Navab, and F. Tombari. A taxonomy and library for visualizing learned features in convolutional neural networks. *arXiv:1606.07757*, 2016.

[9] A. W. Harley. An interactive node-link visualization of convolutional neural networks. In *International Symposium on Visual Computing*, pages 867–877, 2015.

[10] I. Herman, G. Melançon, and M. S. Marshall. Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):24–43, 2000.

[11] G. Hinton, L. Deng, D. Yu, G. Dahl, A.-R. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. Sainath, and B. Kingsbury. Deep neural networks for acoustic modeling in speech recognition. *Signal Processing Magazine*, 29(6):82–97, 2012.

[12] N. Kalchbrenner, E. Grefenstette, and P. Blunsom. A convolutional neural network for modelling sentences. In *Proceedings 52nd Annual Meeting of the Association for Computational Linguistics*, pages 655–665, 2014.

[13] A. Karpathy, J. Johnson, and L. Fei-Fei. Visualizing and understanding recurrent networks. *arXiv:1506.02078*, 2015.

[14] M. Khang, P. Andrews, A. Kalro, and D. H. Chau. ActiVis: Visual exploration of industry-scale deep neural network models. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):88–97, 2017.

[15] Y. Kim. Convolutional neural networks for sentence classification. In *Proceedings 2014 Empirical Methods in Natural Language Processing*, Doha, Qatar, 2014.

[16] S. B. Kotsiantis, I. Zaharakis, and P. Pintelas. Supervised machine learning: A review of classification techniques. In I. Maglogiannis, editor, *Emerging Artificial Intelligence Applications in Computer Engineering*, pages 3–24. IOS Press, 2007.

[17] A. Krizhevsky, I. Sutskever, and G. Hinton. ImageNet classification with deep convolutional neural networks. In *Proceedings 25th International Conference on Neural Information Processing Systems (NIPS '12)*, pages 1097–1105, 2012.

[18] V. T. Landesberger and A. Kuijper. Visual analysis of large graphs: state-of-the-art and future research challenges. 30(6):1719–1749, 2011.

[19] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[20] S. Leeman-Munk, J. Lester, and J. A. Cox. NCSU_SAS_SAM: Deep encoding and reconstruction for normalization of noisy text. In *W-NUT 2015*, pages 154–161, 2015.

[21] J. Li, X. Chen, E. Hovy, and D. Jurafsky. Visualizing and understanding neural models in NLP. In *Proceedings NAACL-HLT*, pages 681–691, 2016.

[22] 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.

[23] S. Liu, X. Wang, M. Liu, and J. Zhu. Towards better analysis of machine learning models: A visual analytics perspective. *Visual Informatics*, 1(1):48–56, 2017.

[24] A. Mahendran and A. Vedaldi. Understanding deep image representations by inverting them. In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition*, pages 5188–5196, 2015.

[25] T. H. Nguyen and R. Grishman. Relation extraction: Perspective from convolutional neural networks. In *HLT-NAACL*, pages 39–48, 2015.

[26] N. Pezzotti, T. Höllt, J. van Gemert, B. P. Lelieveldt, E. Eisemann, and A. Vilanova. DeepEyes: Progressive visual analytics for designing deep neural networks. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):98–108, 2017.

[27] P. E. Rauber, S. G. Fadel, A. X. Falcao, and A. C. Telea. Visualizing the hidden activity of artificial neural networks. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):101–110, 2017.

[28] S. Russell, D. Dewey, and M. Tegmark. Research priorities for robust and beneficial artificial intelligence. *AI Magazine*, 36(4):105–114, 2015.

[29] C. D. Santos and B. Zadrozny. Learning character-level representations for part-of-speech tagging. In *Proceedings 31st International Conference on Machine Learning (ICML-14)*, pages 1818–1826, 2014.

[30] P. Saraiya, P. Lee, and C. North. Visualization of graphs with associated timeseries data. In *IEEE Symposium on Information Visualization*, pages 225–232, 2005.

[31] J. Schmidhuber. Multi-column deep neural networks for image classification. In *Proceedings 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2012)*, pages 3642–3649, 2012.

[32] M. Sedlmair, M. Meyer, and T. Munzner. Design study methodology: Reflections from the trenches and the stacks. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2431–2440, 2012.

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

[34] D. Smilkov, S. Carter, D. Sculley, F. B. Viegas, and M. Wattenberg. Direct manipulation visualization of deep networks. In *ICML Workshop on Visualization for Deep Learning*, 2016.

[35] D. Smilkov, N. Thorat, C. Nicholson, E. Reif, F. B. Viégas, and M. Wattenberg. Embedding projector: Interactive visualization and interpretation of embeddings. *arXiv:1611.05469*, 2016.

[36] H. Strobelt, S. Gehrmann, B. Huber, H. Pfister, and A. M. Rush. LSTMVis: A tool for visual analysis of hidden state dynamics in recurrent neural networks. *IEEE Transactions on Visualization and Computer Graphics*, 24:667–676, 2018.

[37] K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(2):109–125, 1981.

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

[39] S. van den Elzen and J. J. van Wijk. Small multiples, large singles: A new approach for visual data exploration. In *Computer Graphics Forum*, volume 32, pages 191–200, 2013.

[40] L. van der Maaten and G. Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

[41] S. Wold, K. Esbensen, and P. Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52, 1987.

[42] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson. Understanding neural networks through deep visualization. In *Proceedings 2015 International Conference on Machine Learning Deep Learning Workshop (ICML '15)*, Lille, France, 2015.

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

[44] X. Zhang, J. Zhao, and Y. LeCun. Character-level convolutional networks for text classification. In *Proceedings 28th International Conference on Neural Information Processing Systems (NIPS '15)*, pages 649–657, 2015.