ABSTRACT
Current search engines are largely text based, and the text vocabulary that they use is a poor match for the style based concerns. Search engines return pages whose content relates semantically to the keyword query; they cannot be used for searches along stylistic attributes such as “pages with funny backgrounds” or “pages that look professional” etc.

We propose a system\textsuperscript{1} that will allow users to query on stylistic attributes. The search query could be either language-keyword based or it could be based on examples i.e. a user can select a design and ask the system to show other examples in the corpus that are similar (or dissimilar) to the designs in the query. For example, a user can select a a web page, and query the system for other pages with similar (or dissimilar) layout and design.

In this paper we describe how a Recursive Neural Network (RNN) can be used to build such systems. Given a set of training examples consisting of web pages and the corresponding style label distributions, we learn the parameters of the system via a back-propagation neural network which is applied recursively to each node in the binary tree representation of the web page. Once these parameters are learned, the system can predict the style distribution for any given web page. Example-based querying can be implemented on top of this representation.

A comparative study shows that RNN works better than other ML techniques in context of this problem.

Author Keywords
Recursive Neural Networks, Search, Web Design

INTRODUCTION
A web page can be represented as a tree – a Document Object Model (DOM) tree, which describes the page’s content, structure, and style. The DOM representation of a web page consists of a tree with element nodes such as text nodes, images and links as leaf nodes and different container nodes such as div and span as non-leaf nodes. The top level nodes are head and body. Each node recursively inherits style attributes from its parents in addition to its own style attributes. For example,

\textsuperscript{1}We have made our system publicly available at the URL: http://mobisocial.stanford.edu/musemonkey/RNN

Consider the following HTML snippet and its corresponding DOM tree in Figure 1.

\begin{verbatim}
<html>
<head>
<title>Test</title>
</head>
<body style="background-color:green; color:yellow">
<div>
<p>
Sample Text with green background
</p>
</div>
<div style="background-color:red">
<p>
Sample Text with red background and yellow font
</p>
</div>
</body>
</html>
\end{verbatim}

\begin{figure}
\centering
\includegraphics[width=0.4\textwidth]{dom_tree.png}
\caption{DOM Tree Representation}
\end{figure}

Although for simple pages, the DOM closely mirrors the page’s visual structure, it only loosely approximates the page’s visual layout for complex pages, as modern CSS allows content to be arbitrarily repositioned. Thus, we use “Bento” \textsuperscript{1} – a page segmentation algorithm that adjusts the DOM accordingly and takes care to preserve layout details when nodes are reshuffled. We further convert this Bento tree into a full binary tree, because a RNN requires the same data structure for all of the internal nodes. We also have a human-generated mapping of style labels for each web page in our corpus, which is collected using a web-based crowdsourcing interface.

Once we have the binarized Bento tree representation...
of web pages and their style labels, we can define our training set such that each training instance \( i \) contains a mapping \((X^i, Y^i)\) where \( X^i \) represents the binarized Bento tree representation of the web page \( i \) and \( Y^i \) represents the style label distribution for the same page. We show that this tree representation helps in understanding the structure of the web page.

We then introduce the RNN algorithm for predicting recursive structure in multiple modalities. In the following section, we show the relative statistics for various other ML techniques vs. RNN. With a discussion, we conclude that RNN is the most suitable choice for this problem.

RELATED WORK
Prior work has focused mainly on classifying web pages based on content and not on the design attributes. Yu et al. [5] used SVM to classify web pages while Prabowo et al.[2] used ontologies to find similar pages. Ritchie et al.’s [3] “d.tour” facilitates people to search a gallery of web pages based on style attributes but it takes a weighted average of the feature vectors to find similarity between web pages. Although they have tried to solve a similar problem, they do not use any learning techniques and, thus, have limited scope.

Bricolage’s tree-matching algorithm [1] can theoretically be used to induce a distance metric on the space of page layouts. Since Bricolage searches for the optimal mapping between pages, the returned cost can be interpreted as an approximate distance metric on the space of page designs. However, since for every example-based query, the system would have to compute pairwise mappings between the query and all pages in the corpus, the tree-matching algorithm becomes an intractably expensive operation to scale. In the proposed RNN architecture, all those pairwise mapping computations become vector comparisons, making it a scalable solution.

Recent developments in deep learning introduce a structure prediction architecture based on recursive neural networks (RNNs) for understanding the structure of complex scene images and language. Socher et al.[4] have successfully used Recursive Neural Networks for parsing images as well as natural language phrases. In this paper we extend the existing structure prediction techniques to represent structure in Web design.

SYSTEM DESIGN
Extracting Features
Once we have the bento tree representation (xml) of all the web pages we extract the following DOM properties as features for the learning:

The important visual properties include: width, height, area, aspect ratio, font size, font weight, mean color, num of links, num of colors, num of children, num of images, num of siblings, sibling order, text area and word count.

The important semantic properties (boolean) include: search, footer, header, image, logo, login, navigation, bottom (if the node is in the bottom 10% of the page), top (if the node is in the top 10% of the page), fills_height (if the node extends more than 90% down the page) and fills_width (if the node extends more than 90% across the page).

Style Label Clustering
The style labels for each web page were collected through a web-based crowdsourcing interface, which internally uses Amazon Mechanical Turk. Although we had given some pre-defined style labels for Turkers to use, we didn’t restrict them to use only those labels and they were free to use a new style label if needed.

![Figure 2. 13 representative labels from the original 99 labels to reduce dimensionality](image)
RECURSIVE NEURAL NETWORK

The principle behind a recursive neural network (RNN) is to take in a recursive structure modeled as a binary tree (which must be full) and apply the same neural network to each node of the tree. Socher et. al in [4] solve the much harder problem of both trying to generate the tree as well as classifying the nodes correctly. Since web pages have a natural tree-like structure, we already have the generated (binarized bento) tree and instead we only want to determine the parameters for the neural network. Thus, we only have to calculate the tree and instead we only want to determine the parameters for merging multiple nodes to determine the tree.

Let the number of features be \( k \), the number of hidden layers in the network be \( n \) and the number of labels be \( l \).

Then, as leaf nodes always contain features, we can compute its activation vector directly. If node \( i \) contains features \( F_i \), its activation vector \( a_i = f(W^\text{bot} F_i + b^\text{bot}) \) where \( W^\text{bot} \in \mathbb{R}^{n \times k} \) is a matrix that is applied to only terminal nodes, \( b^\text{bot} \) is the bias vector and \( f \) is the sigmoid function.

Non-terminal nodes \( i \) will have their activations computed from the activation vectors of its children \( a_{i-1} \) and \( a_{i-2} \) along (but not required) with its features \( F_i \). Our project does not consider features for internal nodes so \( a_i = f(W[a_{i-1}; a_{i-2}] + b) \). Here, \( W \in \mathbb{R}^{n \times 2n} \) is a matrix that is applied to pair-wise activation vectors, \( b \) is the bias vector and \( f \) is the same sigmoid function described above.

When the activation vector is computed for node \( i \), we can also predict the label distribution for that node (denoted as \( p(i) \)) and this can be accomplished by adding a softmax layer after multiplying activation vector \( a_i \) with a matrix \( W^\text{label} \in \mathbb{R}^{l \times n+1} \). Thus, \( p(i) = \text{softmax}(W^\text{label} a_i) \).

This process is continuously executed until we have calculated the activation vector for the root node. The entire procedure describes the feedforward process and is shown in Figure 3.

The backpropagation algorithm is responsible for calculating the gradients for \( W^\text{bot} \), \( W \) and \( W^\text{label} \). It starts from the root node and calculates the difference between the predicted distribution and the actual distribution for the labels and updates the gradient for both \( W^\text{label} \) and each of the hidden layers as well (represented by \( d_n \)). Then if it is not a leaf, it gets the activations for its two children \( [a_{i-1}; a_{i-2}] \), calculates the gradient for \( W \) as \( d_n [a_{i-1}; a_{i-2}; 1] \) and passes this change onto its children. This process continues recursively until we reach a leaf in which case we update the change for \( W^\text{bot} \) according to the change from its parent multiplied by the features and also update the change for \( W^\text{label} \) based on the difference between the predicted and actual distribution of labels.

For classification of a web page, the RNN takes in the binary tree for that webpage and executes the feedforward process for each of the nodes in the tree using \( W^\text{bot} \), \( W \) and \( W^\text{cat} \) that were determined during the training step. The predicted label distribution for the root node is treated as the overall distribution for the web page. Since each of the pages can have a variable number of labels, we use a threshold on this distribution to determine which all labels to assign for that webpage. Thus, the goal of our classifier is to achieve a high F1-score on unseen examples as we need to balance both precision and recall simultaneously.

EXPERIMENTAL RESULTS AND DISCUSSIONS

Data

The webpages needed for our experiments were provided by the Stanford HCI department and consist of 292 examples and each page can have multiple labels. Since the number of examples available was small, we could only do a 90-10 split with 262 examples available for training and 30 examples for testing. The below figure shows a 2-d visualization of the data projected on the first two principal components.

![Figure 4. Two-dimensional visualization of the data showing the first two principal components.](Image)

Feature Selection

One of the benefits of using an RNN is that it is able to determine which features are relevant for the classification task. Thus, we used all of the features (52) that were described in System Design. We used feature selection algorithms such as forward search, backward search, mutual independence and domain knowledge to reduce the number of features to 11, but the performance of the RNN was actually worse and we, decided to use all 52 features instead. All of the features were normalized to have a mean of zero and standard deviation of one.

Parameter Tweaking

There are two main parameters to tweak here: (i) \( n \) – the number of hidden layers in the RNN and (ii) \( \tau \) – the threshold that a label probability must exceed in order to be assigned as a label for the corresponding web page. We derived these two values by performing 5-fold cross validation on the training data and choosing the values that give the best results on the average F1-scores of the five folds.
We increase the variance of our model if we increase \( n \) as training error decreases but test error increases. Similarly, a lower \( \tau \) will result in a lower precision, but a higher recall as even lower-probability labels now will be considered as possible labels. Since our evaluation metric is based on the F1-score (\( F1 \)) by giving equal weights to both precision (\( p \)) and recall (\( r \)), figuring out the right threshold is crucial for the accuracy of our application. Two other error metrics that we considered were one-error (\( oe \)) which evaluates how many times the top-ranked label is not actually a label for the webpage and hamming loss (\( hl \)) which evaluates how many times a webpage-label pair is misclassified averaged over the number of labels for each page. Both of these metrics are discussed in [6].

Results
We have experimented with varying the values of \( n \) and then finding the \( \tau \) that maximizes the \( F1 \) score for the 5-fold cross validated training data. We show the \( F1 \) scores on both the training and test sets made for various values of \( n \) in the below table given that the \( F1_{\text{train}} \) is the highest value possible for \( n \) and \( \tau \):

<table>
<thead>
<tr>
<th>( n )</th>
<th>( \tau )</th>
<th>( p_{\text{test}} )</th>
<th>( r_{\text{test}} )</th>
<th>( F1_{\text{test}} )</th>
<th>( F1_{\text{train}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0.08</td>
<td>0.384</td>
<td>0.754</td>
<td>0.500</td>
<td>0.500</td>
</tr>
<tr>
<td>10</td>
<td>0.13</td>
<td>0.463</td>
<td>0.944</td>
<td>0.500</td>
<td>0.505</td>
</tr>
<tr>
<td>15</td>
<td>0.12</td>
<td>0.508</td>
<td>0.961</td>
<td>0.533</td>
<td>0.540</td>
</tr>
<tr>
<td>20</td>
<td>0.14</td>
<td>0.468</td>
<td>0.386</td>
<td>0.423</td>
<td>0.486</td>
</tr>
</tbody>
</table>

Since the maximum \( F1_{\text{train}} \) occurs when \( n = 15 \) and \( \tau = 0.12 \), this is the best possible result that the RNN can give us without looking at the test set. Looking at the test set then shows that for the set of parameters chosen above, \( n = 15 \) and \( \tau = 0.12 \) also gives the highest \( F1_{\text{test}} \) score.

Upon Socher’s suggestion, we have also modified our algorithm to have non-terminal nodes contain features as well. Then, \( W \in \mathbb{R}^{n \times (2n+k)} \) and during back propagation, we ignore the \( \delta \) that is involved for the columns of \( W \) that interact with the features themselves. Accuracy decreases slightly as if \( n = 15 \), \( F1_{\text{train}} \) score is highest when \( \tau = 0.15 \) which will have a \( F1_{\text{test}} \) score of 0.52 < 0.53. However, due to time constraints, we could not analyze this algorithm in detail and the above conclusions may not actually be true.

Comparison with other algorithms
Since our problem involves dealing with outputs giving multiple labels, we have compared the results of the RNN with two specialized algorithms for predicting multi-label instances, BPMLL\(^2\) which uses neural network classification and MIMLNN\(^3\) which uses nearest-neighbor classification. In both of these algorithms, we use the features of the root node for the webpage only (which will be composed of the features from all of the other nodes in the tree along with its own features). MIMLNN suffers from abysmal recall if any regularization (\( \lambda \)) is done while precision drops if no regularization is included. Similarly, BPMLL (parameterized by \( h \) – the number of hidden layers in the neural network) also gives below average results as well due to having low recall. The results of the three algorithms (BPMLL, MIMLNN and RNN) for various parameters and evaluation metrics are shown in the following table:

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>( p )</th>
<th>( r )</th>
<th>( F1 )</th>
<th>( oe )</th>
<th>( hl )</th>
</tr>
</thead>
<tbody>
<tr>
<td>BPMLL (( h = 5 ))</td>
<td>0.54</td>
<td>0.25</td>
<td>0.34</td>
<td>0.60</td>
<td>0.19</td>
</tr>
<tr>
<td>BPMLL (( h = 10 ))</td>
<td>0.53</td>
<td>0.27</td>
<td>0.36</td>
<td>0.60</td>
<td>0.17</td>
</tr>
<tr>
<td>MIMLNN (( \lambda = 0 ))</td>
<td>0.49</td>
<td>0.21</td>
<td>0.29</td>
<td>0.60</td>
<td>0.19</td>
</tr>
<tr>
<td>MIMLNN (( \lambda = 1 ))</td>
<td>0.55</td>
<td>0.06</td>
<td>0.10</td>
<td>0.57</td>
<td>0.15</td>
</tr>
<tr>
<td>Best RNN</td>
<td>0.51</td>
<td>0.56</td>
<td>0.53</td>
<td>0.35</td>
<td>0.16</td>
</tr>
</tbody>
</table>

\(^2\)http://lamda.nju.edu.cn/datacode/BPMLL.htm
\(^3\)http://lamda.nju.edu.cn/code_MIML.ashx
As can be seen from the above table, performance of RNN is superior to the other methods as it is able to balance both precision and recall while classifiers that only classify the root node choose to optimize precision instead. The $F_1$-score difference is huge (by at least 0.15). Even $oe$ is much lower and $hl$ is around the same values as the other classifiers, making the RNN the classifier of choice to use for this problem.

One question that should be asked is why the $F_1$-score is still only $\sim 0.5$. One explanation is that the dataset given was not of the highest quality as webpages have multiple labels and our clustering algorithm is coarse and can make mistakes. This will result in incorrect labels assigned to web pages and can mislead the classifier. The other problem is that RNNs are very effective when the nodes of a tree have different label distribution, yet, in our case, we give the same distribution to all of the nodes. We can greatly improve performance if different nodes can have different distributions as the RNN can more fully exploit the structure of the trees given.

### CONCLUSION AND FUTURE WORK

This paper introduced the RNN algorithm for learning the recursive structure in web pages. We show that we can capture the visual structure and style of a web page in a format suitable for applying a RNN and build a model for a scalable style-based search system for web designs.

The dataset that was available to us was very small. Out of a total 292 web pages, we trained on 262 pages and tested on the remaining 30 pages. We will have a larger dataset available in early January and we aim to re-train our model on the new dataset. The current prototype employs 52 visual and semantic features. Adding more sophisticated properties such as those based on computer vision techniques will likely improve the quality of the classifier.

Future work could use the same RNN model to allow searching on similar designs for parts of a web page. For example, if a user selects a header element in a web page, the system will return other pages with similar (or dissimilar) headers. The interface can be extended to allow users to create abstractions that represent the content layout of pages and use them as queries. The user can draw nested rectangular boxes and affix known semantic labels to them (header, logo, image, etc.) The system could search for pages that mimicked the layout exactly, with similar content blocks positioned in similar relative locations. The RNN is ideal for this task as it is able to make predictions for every node of the tree, rather than at just the root.

### ACKNOWLEDGMENTS

We thank Richard Socher for advising us on the algorithm and helping us on the implementation of the RNN; Ranjitha Kumar for giving us the codebase (bento tree representation) and the dataset and Prof. Scott Klemmer for his helpful comments.

### REFERENCES


### APPENDIX

We have made our system publicly available at the URL: http://mobisocial.stanford.edu/musemonkey/RNN

---

**Figure 5. A screenshot below that shows the search results for keyword: ‘green’**