SoPa: Bridging CNNs, RNNs, and Weighted Finite-State Machines

Roy Schwartz[*] [◇♡] **Sam Thomson**[*] [♣] **Noah A. Smith**[◇]
[◇]Paul G. Allen School of Computer Science & Engineering, University of Washington
[♣]Language Technologies Institute, Carnegie Mellon University
[♡]Allen Institute for Artificial Intelligence
{roysch,nasmith}@cs.washington.edu, sthomson@cs.cmu.edu

Abstract

Recurrent and convolutional neural networks comprise two distinct families of models that have proven to be useful for encoding natural language utterances. In this paper we present SoPa, a new model that aims to bridge these two approaches. SoPa combines neural representation learning with weighted finite-state automata (WFSAs) to learn a soft version of traditional surface patterns. We show that SoPa is an extension of a one-layer CNN, and that such CNNs are equivalent to a restricted version of SoPa, and accordingly, to a restricted form of WFSA. Empirically, on three text classification tasks, SoPa is comparable or better than both a BiLSTM (RNN) baseline and a CNN baseline, and is particularly useful in small data settings.

1 Introduction

Recurrent neural networks (RNNs; Elman, 1990) and convolutional neural networks (CNNs; LeCun, 1998) are two of the most useful text representation learners in NLP (Goldberg, 2016). These methods are generally considered to be quite different: the former encodes an arbitrarily long sequence of text, and is highly expressive (Siegelmann and Sontag, 1995). The latter is more local, encoding fixed length windows, and accordingly less expressive. In this paper, we seek to bridge the gap between RNNs and CNNs, presenting **SoPa** (for **So**ft **Pa**tterns), a model that lies in between them.

SoPa is a neural version of a weighted finite-state automaton (WFSA), with a restricted set of transitions. Linguistically, SoPa is appealing as it is able to capture a soft notion of surface patterns (e.g., *"what a great **X** !"*; Hearst, 1992), where some words may be dropped, inserted, or replaced with similar words (see Figure 1). From a modeling perspective, SoPa is interesting because WFSAs are well-studied and come with efficient and flexible inference algorithms (Mohri, 1997; Eisner, 2002) that SoPa can take advantage of.

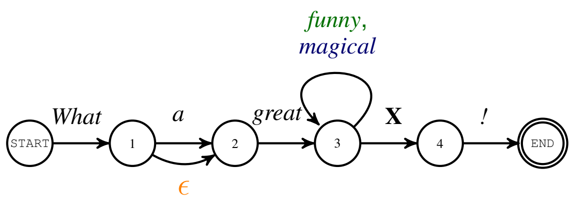

Figure 1: A representation of a surface pattern as a six-state automaton. **Self-loops** allow for repeatedly inserting words (e.g., "*funny*"). **ϵ-transitions** allow for dropping words (e.g., "*a*").

SoPa defines a set of soft patterns of different lengths, with each pattern represented as a WFSA (Section 3). While the number and lengths of the patterns are hyperparameters, the patterns themselves are learned end-to-end. SoPa then represents a document with a vector that is the aggregate of the scores computed by matching each of the patterns with each span in the document. Because SoPa defines a hidden state that depends on the input token and the previous state, it can be thought of as a simple type of RNN.

We show that SoPa is an extension of a one-layer CNN (Section 4). Accordingly, one-layer CNNs can be viewed as a collection of linear-chain WFSAs, each of which can only match fixed-length spans, while our extension allows matches of flexible-length. As a simple type of RNN that is more expressive than a CNN, SoPa helps to link CNNs and RNNs.

[*]The first two authors contributed equally.

To test the utility of SoPa, we experiment with three text classification tasks (Section 5). We compare against four baselines, including both a bidirectional LSTM and a CNN. Our model performs on par with or better than all baselines on all tasks (Section 6). Moreover, when training with smaller datasets, SoPa is particularly useful, outperforming all models by substantial margins. Finally, building on the connections discovered in this paper, we offer a new, simple method to interpret SoPa (Section 7). This method applies equally well to CNNs. We release our code at `https://github.com/Noahs-ARK/soft_patterns`.

2 Background

Surface patterns. Patterns (Hearst, 1992) are particularly useful tool in NLP (Lin et al., 2003; Etzioni et al., 2005; Schwartz et al., 2015). The most basic definition of a pattern is a sequence of words and wildcards (e.g., "**X** is a **Y**"), which can either be manually defined or extracted from a corpus using cooccurrence statistics. Patterns can then be matched against a specific text span by replacing wildcards with concrete words.

Davidov et al. (2010) introduced a *flexible* notion of patterns, which supports partial matching of the pattern with a given text by skipping some of the words in the pattern, or introducing new words. In their framework, when a sequence of text partially matches a pattern, hard-coded partial scores are assigned to the pattern match. Here, we represent patterns as WFSAs with neural weights, and support these partial matches in a soft manner.

WFSAs. We review weighted finite-state automata with ϵ-transitions before we move on to our special case in Section 3. A WFSA-ϵ with d states over a vocabulary V is formally defined as a tuple $F = \langle \pi, \mathbf{T}, \eta \rangle$, where $\pi \in \mathbb{R}^d$ is an initial weight vector, $\mathbf{T} : (V \cup \{\epsilon\}) \rightarrow \mathbb{R}^{d \times d}$ is a transition weight function, and $\eta \in \mathbb{R}^d$ is a final weight vector. Given a sequence of words in the vocabulary $\boldsymbol{x} = \langle x_1, \ldots, x_n \rangle$, the Forward algorithm (Baum and Petrie, 1966) scores $\boldsymbol{x}$ with respect to F. Without ϵ-transitions, Forward can be written as a series of matrix multiplications:

$$p'_{\text{span}}(\boldsymbol{x}) = \pi^\top \left(\prod_{i=1}^{n} \mathbf{T}(x_i) \right) \eta \tag{1}$$

ϵ-transitions are followed without consuming a word, so Equation 1 must be updated to reflect the possibility of following any number (zero or more) of ϵ-transitions in between consuming each word:

$$p_{\text{span}}(\boldsymbol{x}) = \pi^\top \mathbf{T}(\epsilon)^* \left(\prod_{i=1}^{n} \mathbf{T}(x_i)\mathbf{T}(\epsilon)^* \right) \eta \tag{2}$$

where * is matrix asteration: $\mathbf{A}^* := \sum_{j=0}^{\infty} \mathbf{A}^j$. In our experiments we use a first-order approximation, $\mathbf{A}^* \approx \mathbf{I} + \mathbf{A}$, which corresponds to allowing zero or one ϵ-transition at a time. When the FSA F is probabilistic, the result of the Forward algorithm can be interpreted as the marginal probability of all paths through F while consuming $\boldsymbol{x}$ (hence the symbol "p").

The Forward algorithm can be generalized to any semiring (Eisner, 2002), a fact that we make use of in our experiments and analysis.[1] The vanilla version of Forward uses the sum-product semiring: $\oplus$ is addition, $\otimes$ is multiplication. A special case of Forward is the Viterbi algorithm (Viterbi, 1967), which sets $\oplus$ to the $\max$ operator. Viterbi finds the *highest scoring* path through F while consuming $\boldsymbol{x}$. Both Forward and Viterbi have runtime $O(d^3 + d^2 n)$, requiring just a single linear pass through the phrase. Using first-order approximate asteration, this runtime drops to $O(d^2 n)$.[2]

Finally, we note that Forward scores are for *exact matches*—the entire phrase must be consumed. We show in Section 3.2 how phrase-level scores can be summarized into a document-level score.

3 SoPa: A Weighted Finite-State Automaton RNN

We introduce SoPa, a WFSA-based RNN, which is designed to represent text as collection of surface pattern occurrences. We start by showing how a single pattern can be represented as a WFSA-ϵ (Section 3.1). Then we describe how to score a complete document using a pattern (Section 3.2), and how multiple patterns can be used to encode a document (Section 3.3). Finally, we show that SoPa can be seen as a simple variant of an RNN (Section 3.4).

[1]The semiring parsing view (Goodman, 1999) has produced unexpected connections in the past (Eisner, 2016). We experiment with max-product and max-sum semirings, but note that our model could be easily updated to use any semiring.

[2]In our case, we also use a sparse transition matrix (Section 3.1), which further reduces our runtime to $O(dn)$.

3.1 Patterns as WFSAs

We describe how a pattern can be represented as a WFSA-ϵ. We first assume a single pattern. A pattern *is* a WFSA-ϵ, but we impose hard constraints on its shape, and its transition weights are given by differentiable functions that have the power to capture concrete words, wildcards, and everything in between. Our model is designed to behave similarly to flexible hard patterns (see Section 2), but to be learnable directly and "end-to-end" through backpropagation. Importantly, it will still be interpretable as simple, almost linear-chain, WFSA-ϵ.

Each pattern has a sequence of d states (in our experiments we use patterns of varying lengths between 2 and 7). Each state i has exactly three possible outgoing transitions: a **self-loop**, which allows the pattern to consume a word without moving states, a **main path** transition to state $i+1$ which allows the pattern to consume one token and move forward one state, and an **ϵ-transition** to state $i+1$, which allows the pattern to move forward one state without consuming a token. All other transitions are given score 0. When processing a sequence of text with a pattern p, we start with a special `START` state, and only move forward (or stay put), until we reach the special `END` state.[3] A pattern with d states will tend to match token spans of length $d-1$ (but possibly shorter spans due to ϵ-transitions, or longer spans due to self-loops). See Figure 1 for an illustration.

Our transition function, $\mathbf{T}$, is a parameterized function that returns a $d \times d$ matrix. For a word x:

$$[\mathbf{T}(x)]_{i,j} = \begin{cases} E(\mathbf{u}_i \cdot \mathbf{v}_x + a_i), & \text{if } j = i \text{ } \textit{(self-loop)} \\ E(\mathbf{w}_i \cdot \mathbf{v}_x + b_i), & \text{if } j = i+1 \\ 0, & \text{otherwise,} \end{cases} \tag{3}$$

where $\mathbf{u}_i$ and $\mathbf{w}_i$ are vectors of parameters, a_i and b_i are scalar parameters, $\mathbf{v}_x$ is a fixed pre-trained word vector for x,[4] and E is an encoding function, typically the identity function or sigmoid. ϵ-transitions are also parameterized, but don't consume a token and depend only on the current state:

$$[\mathbf{T}(\epsilon)]_{i,j} = \begin{cases} E(c_i), & \text{if } j = i+1 \\ 0, & \text{otherwise,} \end{cases} \tag{4}$$

where c_i is a scalar parameter.[5] As we have only three non-zero diagonals in total, the matrix multiplications in Equation 2 can be implemented using vector operations, and the overall runtimes of Forward and Viterbi are reduced to $O(dn)$.[6]

Words vs. wildcards. Traditional *hard* patterns distinguish between words and wildcards. Our model does not explicitly capture the notion of either, but the transition weight function can be interpreted in those terms. Each transition is a logistic regression over the next word vector $\mathbf{v}_x$. For example, for a main path out of state i, $\mathbf{T}$ has two parameters, $\mathbf{w}_i$ and b_i. If $\mathbf{w}_i$ has large magnitude and is close to the word vector for some word y (e.g., $\mathbf{w}_i \approx 100\mathbf{v}_y$), and b_i is a large negative bias (e.g., $b_i \approx -100$), then the transition is essentially matching the specific word y. Whereas if $\mathbf{w}_i$ has small magnitude ($\mathbf{w}_i \approx \mathbf{0}$) and b_i is a large positive bias (e.g., $b_i \approx 100$), then the transition is ignoring the current token and matching a wildcard.[7] The transition could also be something in between, for instance by focusing on specific dimensions of a word's meaning encoded in the vector, such as POS or semantic features like animacy or concreteness (Rubinstein et al., 2015; Tsvetkov et al., 2015).

3.2 Scoring Documents

So far we described how to calculate how well a pattern matches a token span exactly (consuming the whole span). To score a complete document, we prefer a score that aggregates over all matches on subspans of the document (similar to "search" instead of "match" in regular expression parlance). We still assume a single pattern.

Either the Forward algorithm can be used to calculate the expected count of the pattern in the document, $\sum_{1 \le i \le j \le n} p_{\text{span}}(\boldsymbol{x}_{i:j})$, or Viterbi to calculate $s_{\text{doc}}(\boldsymbol{x}) = \max_{1 \le i \le j \le n} s_{\text{span}}(\boldsymbol{x}_{i:j})$, the score of the highest-scoring match. In short documents, we expect patterns to typically occur at most once, so in our experiments we choose the Viterbi algorithm, i.e., the max-product semiring.

Implementation details. We give the specific recurrences we use to score documents in a single

[3]To ensure that we start in the `START` state and end in the `END` state, we fix $\pi = [1, 0, \ldots, 0]$ and $\eta = [0, \ldots, 0, 1]$.

[4]We use GloVe 300d 840B (Pennington et al., 2014).

[5]Adding ϵ-transitions to WFSAs does not increase their expressive power, and in fact slightly complicates the Forward equations. We use them as they require fewer parameters, and make the modeling connection between (hard) flexible patterns and our (soft) patterns more direct and intuitive.

[6]Our implementation is optimized to run on GPUs, so the observed runtime is even *sublinear* in d.

[7]A large bias increases the eagerness to match *any* word.

pass with this model. We define:

$$[\operatorname{maxmul}(\mathbf{A}, \mathbf{B})]_{i,j} = \max_k A_{i,k} B_{k,j}. \quad (5)$$

We also define the following for taking zero or one ϵ-transitions:

$$\operatorname{eps}(\mathbf{h}) = \operatorname{maxmul}(\mathbf{h}, \max(\mathbf{I}, \mathbf{T}(\epsilon))) \quad (6)$$

where $\max$ is element-wise max. We maintain a row vector $\mathbf{h}_t$ at each token:[8]

$$\mathbf{h}_0 = \operatorname{eps}(\pi^\top), \quad (7a)$$

$$\mathbf{h}_{t+1} = \max(\operatorname{eps}(\operatorname{maxmul}(\mathbf{h}_t, \mathbf{T}(x_{t+1}))), \mathbf{h}_0), \quad (7b)$$

and then extract and aggregate END state values:

$$s_t = \operatorname{maxmul}(\mathbf{h}_t, \eta), \quad (8a)$$

$$s_{\text{doc}} = \max_{1 \le t \le n} s_t. \quad (8b)$$

$[\mathbf{h}_t]_i$ represents the score of the best path through the pattern that ends in state i after consuming t tokens. By including $\mathbf{h}_0$ in Equation 7b, we are accounting for spans that start at time $t+1$. s_t is the maximum of the exact match scores for all spans ending at token t. And s_{doc} is the maximum score of any subspan in the document.

3.3 Aggregating Multiple Patterns

We describe how k patterns are aggregated to score a document. These k patterns give k different s_{doc} scores for the document, which are stacked into a vector $\mathbf{z} \in \mathbb{R}^k$ and constitute the final document representation of SoPa. This vector representation can be viewed as a feature vector. In this paper, we feed it into a multilayer perceptron (MLP), culminating in a softmax to give a probability distribution over document labels. We minimize cross-entropy, allowing the SoPa and MLP parameters to be learned end-to-end.

SoPa uses a total of $(2e+3)dk$ parameters, where e is the word embedding dimension, d is the number of states and k is the number of patterns. For comparison, an LSTM with a hidden dimension of h has $4((e+1)h + h^2)$. In Section 6 we show that SoPa consistently uses fewer parameters than a BiLSTM baseline to achieve its best result.

[8]Here a row vector $\mathbf{h}$ of size n can also be viewed as a $1 \times n$ matrix.

3.4 SoPa as an RNN

SoPa can be considered an RNN. As shown in Section 3.2, a single pattern with d states has a hidden state vector of size d. Stacking the k hidden state vectors of k patterns into one vector of size $k \times d$ can be thought of as the hidden state of our model. This hidden state is, like in any other RNN, dependent of the input and the previous state. Using self-loops, the hidden state at time point i can in theory depend on the entire history of tokens up to x_i (see Figure 2 for illustration). We do want to discourage the model from following too many self-loops, only doing so if it results in a better fit with the remainder of the pattern. To do this we use the sigmoid function as our encoding function E (see Equation 3), which means that all transitions have scores strictly less than 1. This works to keep pattern matches close to their intended length. Using other encoders, such as the identity function, can result in different dynamics, potentially encouraging rather than discouraging self-loops.

Although even single-layer RNNs are Turing complete (Siegelmann and Sontag, 1995), SoPa's expressive power depends on the semiring. When a WFSA is thought of as a function from finite sequences of tokens to semiring values, it is restricted to the class of functions known as **rational series** (Schützenberger, 1961; Droste and Gastin, 1999; Sakarovitch, 2009).[9] It is unclear how limiting this theoretical restriction is in practice, especially when SoPa is used as a component in a larger network. We defer the investigation of the exact computational properties of SoPa to future work. In the next section, we show that SoPa is an extension of a one-layer CNN, and hence more expressive.

4 SoPa as a CNN Extension

A convolutional neural network (**CNN**; LeCun, 1998) moves a fixed-size sliding window over the document, producing a vector representation for each window. These representations are then often summed, averaged, or max-pooled to produce a document-level representation (Kim, 2014; Yin and Schütze, 2015). In this section, we show that SoPa is an extension of one-layer, max-pooled CNNs.

To recover a CNN from a soft pattern with $d+1$ states, we first remove self-loops and ϵ-transitions,

[9]Rational series generalize recognizers of *regular languages*, which are the special case of the Boolean semiring.

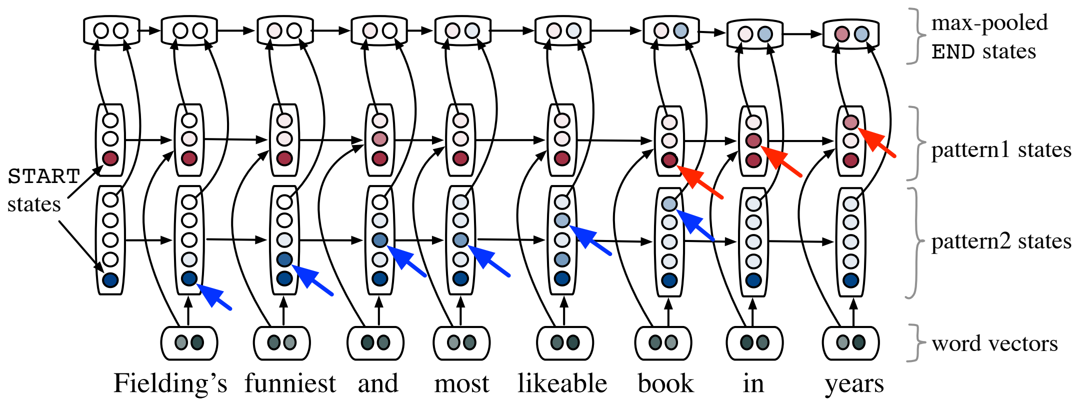

Figure 2: State activations of two patterns as they score a document. pattern1 (length three) matches on *"in years"*. pattern2 (length five) matches on *"funniest and most likeable book"*, using a self-loop to consume the token *"most"*. Active states in the best match are marked with arrow cursors.

retaining only the main path transitions. We also use the identity function as our encoder E (Equation 3), and use the max-sum semiring. With only main path transitions, the network will not match any span that is not exactly d tokens long. Using max-sum, spans of length d will be assigned the score:

$$s_{\text{span}}(\boldsymbol{x}_{i:i+d}) = \sum_{j=0}^{d-1} \mathbf{w}_j \cdot \mathbf{v}_{x_{i+j}} + b_j, \tag{9a}$$

$$= \mathbf{w}_{0:d} \cdot \mathbf{v}_{x_{i:i+d}} + \sum_{j=0}^{d-1} b_j, \tag{9b}$$

where $\mathbf{w}_{0:d} = [\mathbf{w}_0^\top; \ldots; \mathbf{w}_{d-1}^\top]^\top$, $\mathbf{v}_{x_{i:i+d}} = [\mathbf{v}_{x_i}^\top; \ldots; \mathbf{v}_{x_{i+d-1}}^\top]^\top$. Rearranged this way, we recognize the span score as an affine transformation of the concatenated word vectors $\mathbf{v}_{x_{i:i+d}}$. If we use k patterns, then together their span scores correspond to a linear filter with window size d and output dimension k.[10] A single pattern's score for a document is:

$$s_{\text{doc}}(\boldsymbol{x}) = \max_{1 \le i \le n-d+1} s_{\text{span}}(\boldsymbol{x}_{i:i+d}). \tag{10}$$

The $\max$ in Equation 10 is calculated for each pattern independently, corresponding exactly to element-wise max-pooling of the CNN's output layer.

Based on the equivalence between this impoverished version of SoPa and CNNs, we conclude that one-layer CNNs are learning an even more restricted class of WFSAs (linear-chain WFSAs) that capture only fixed-length patterns.

One notable difference between SoPa and arbitrary CNNs is that in general CNNs can use any filter (like an MLP over $\mathbf{v}_{x_{i:i+d}}$, for example). In contrast, in order to efficiently pool over flexible-length spans, SoPa is restricted to operations that follow the semiring laws.[11]

As a model that is more flexible than a one-layer CNN, but (arguably) less expressive than many RNNs, SoPa lies somewhere on the continuum between these two approaches. Continuing to study the bridge between CNNs and RNNs is an exciting direction for future research.

5 Experiments

To evaluate SoPa, we apply it to text classification tasks. Below we describe our datasets and baselines. More details can be found in Appendix A.

Datasets. We experiment with three binary classification datasets.

- **SST**. The Stanford Sentiment Treebank (Socher et al., 2013)[12] contains roughly 10K movie reviews from Rotten Tomatoes,[13] labeled on a scale of 1–5. We consider the binary task, which considers 1 and 2 as negative, and 4 and 5 as positive (ignoring 3s). It is worth noting that this dataset also contains syntactic phrase level annotations, providing a sentiment label to parts of

[10]This variant of SoPa has d bias parameters, which correspond to only a single bias parameter in a CNN. The redundant biases may affect optimization but are an otherwise unimportant difference.

[11]The max-sum semiring corresponds to a linear filter with max-pooling. Other semirings could potentially model more interesting interactions, but we leave this to future work.

[12]`https://nlp.stanford.edu/sentiment/index.html`

[13]`http://www.rottentomatoes.com`

sentences. In order to experiment in a realistic setup, we only consider the complete sentences, and ignore syntactic annotations at train or test time. The number of training/development/test sentences in the dataset is 6,920/872/1,821.

- **Amazon**. The Amazon Review Corpus (McAuley and Leskovec, 2013)[14] contains electronics product reviews, a subset of a larger review dataset. Each document in the dataset contains a review and a summary. Following Yogatama et al. (2015), we only use the reviews part, focusing on positive and negative reviews. The number of training/development/test samples is 20K/5K/25K.
- **ROC**. The ROC story cloze task (Mostafazadeh et al., 2016) is a story understanding task.[15] The task is composed of four-sentence story prefixes, followed by two competing endings: one that makes the joint five-sentence story coherent, and another that makes it incoherent. Following Schwartz et al. (2017), we treat it as a style detection task: we treat all "right" endings as positive samples and all "wrong" ones as negative, and we ignore the story prefix. We split the development set into train and development (of sizes 3,366 and 374 sentences, respectively), and take the test set as-is (3,742 sentences).

Reduced training data. In order to test our model's ability to learn from small datasets, we also randomly sample 100, 500, 1,000 and 2,500 SST training instances and 100, 500, 1,000, 2,500, 5,000, and 10,000 Amazon training instances. Development and test sets remain the same.

Baselines. We compare to four baselines: a BiLSTM, a one-layer CNN, DAN (a simple alternative to RNNs) and a feature-based classifier trained with hard-pattern features.

- **BiLSTM.** Bidirectional LSTMs have been successfully used in the past for text classification tasks (Zhou et al., 2016). We learn a one-layer BiLSTM representation of the document, and feed the average of all hidden states to an MLP.
- **CNN**. CNNs are particularly useful for text classification (Kim, 2014). We train a one-layer CNN with max-pooling, and feed the resulting representation to an MLP.
- **DAN**. We learn a deep averaging network with word dropout (Iyyer et al., 2015), a simple but strong text-classification baseline.
- **Hard**. We train a logistic regression classifier with hard-pattern features. Following Tsur et al. (2010), we replace low frequency words with a special wildcard symbol. We learn sequences of 1–6 concrete words, where any number of wildcards can come between two adjacent words. We consider words occurring with frequency of at least 0.01% of our training set as concrete words, and words occurring in frequency 1% or less as wildcards.[16]

Number of patterns. SoPa requires specifying the number of patterns to be learned, and their lengths. Preliminary experiments showed that the model doesn't benefit from more than a few dozen patterns. We experiment with several configurations of patterns of different lengths, generally considering 0, 10 or 20 patterns of each pattern length between 2–7. The total number of patterns learned ranges between 30–70.[17]

6 Results

Table 1 shows our main experimental results. In two of the cases (SST and ROC), SoPa outperforms all models. On Amazon, SoPa performs within 0.3 points of CNN and BiLSTM, and outperforms the other two baselines. The table also shows the number of parameters used by each model for each task. Given enough data, models with more parameters should be expected to perform better. However, SoPa performs better or roughly the same as a BiLSTM, which has 3–6 times as many parameters.

Figure 3 shows a comparison of all models on the SST and Amazon datasets with varying training set sizes. SoPa is substantially outperforming all baselines, in particular BiLSTM, on small datasets (100 samples). This suggests that SoPa is better fit to learn from small datasets.

Ablation analysis. Table 1 also shows an ablation of the differences between SoPa and CNN: max-product semiring with sigmoid vs. max-sum semiring with identity, self-loops, and ϵ-transitions. The last line is equivalent to a CNN with

[14]`http://riejohnson.com/cnn_data.html`

[15]`http://cs.rochester.edu/nlp/rocstories/`

[16]Some words may serve as both words and wildcards. See Davidov and Rappoport (2008) for discussion.

[17]The number of patterns and their length are hyperparameters tuned on the development data (see Appendix A).

Model	ROC	SST	Amazon
Hard	62.2 (4K)	75.5 (6K)	88.5 (67K)
DAN	64.3 (91K)	83.1 (91K)	85.4 (91K)
BiLSTM	65.2 (844K)	84.8 (1.5M)	**90.8** (844K)
CNN	64.3 (155K)	82.2 (62K)	90.2 (305K)
SoPa	**66.5** (255K)	**85.6** (255K)	90.5 (256K)
SoPa$_{ms_{\mathbb{1}}}$	64.4	84.8	90.0
SoPa$_{ms_{\mathbb{1}}} \setminus \{sl\}$	63.2	84.6	89.8
SoPa$_{ms_{\mathbb{1}}} \setminus \{\epsilon\}$	64.3	83.6	89.7
SoPa$_{ms_{\mathbb{1}}} \setminus \{sl, \epsilon\}$	64.0	85.0	89.5

Table 1: Test classification accuracy (and the number of parameters used). The bottom part shows our ablation results: SoPa: our full model. SoPa$_{ms_{\mathbb{1}}}$: running with max-sum semiring (rather than max-product), with the identity function as our encoder E (see Equation 3). sl: self-loops, ϵ: ϵ transitions. The final row is equivalent to a one-layer CNN.

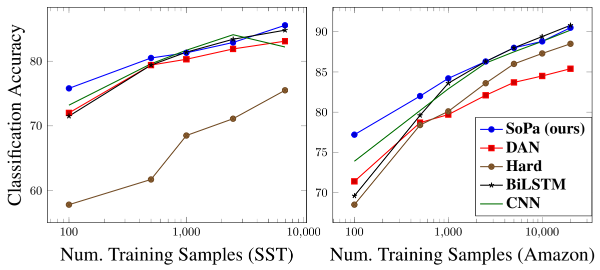

Figure 3: Test accuracy on SST and Amazon with varying number of training instances.

multiple window sizes. Interestingly, the most notable difference between SoPa and CNN is the semiring and encoder function, while ϵ transitions and self-loops have little effect on performance.[18]

7 Interpretability

We turn to another key aspect of SoPa—its interpretability. We start by demonstrating how we interpret a single pattern, and then describe how to interpret the decisions made by downstream classifiers that rely on SoPa—in this case, a sentence classifier. Importantly, these visualization techniques are equally applicable to CNNs.

Interpreting a single pattern. In order to visualize a pattern, we compute the pattern matching scores with each phrase in our training dataset, and select the k phrases with the highest scores. Table 2 shows examples of six patterns learned using the best SoPa model on the SST dataset, as

[18]Although SoPa does make use of them—see Section 7.

	Highest Scoring Phrases				
Patt. 1	thoughtful	,	reverent	portrait	of
	and	astonishingly	articulate	cast	of
	entertaining	,	thought-provoking	film	with
	gentle	,	mesmerizing	portrait	of
	poignant	and	uplifting	story	in
Patt. 2	's	ϵ	uninspired	story	.
	this	ϵ	bad	on	purpose
	this	ϵ	leaden	comedy	.
	a	ϵ	half-assed	film	.
	is	ϵ	clumsy ,$_{SL}$	the	writing
Patt. 3	mesmerizing	portrait	of	a	
	engrossing	portrait	of	a	
	clear-eyed	portrait	of	an	
	fascinating	portrait	of	a	
	self-assured	portrait	of	small	
Patt. 4	honest	,	and	enjoyable	
	soulful	, scathing$_{SL}$	and	joyous	
	unpretentious	, charming$_{SL}$	,	quirky	
	forceful	,	and	beautifully	
	energetic	,	and	surprisingly	
Patt. 5	is	deadly	dull		
	a	numbingly	dull		
	is	remarkably	dull		
	is	a	phlegmatic		
	an	utterly	incompetent		
Patt. 6	five	minutes			
	four	minutes			
	final	minutes			
	first	half-hour			
	fifteen	minutes			

Table 2: Six patterns of different lengths learned by SoPa on SST. Each group represents a single pattern p, and shows the five phrases in the training data that have the highest score for p. Columns represent pattern states. Words marked with $_{SL}$ are self-loops. ϵ symbols indicate ϵ-transitions. All other words are from main path transitions.

represented by their five highest scoring phrases in the training set. A few interesting trends can be observed from these examples. First, it seems our patterns encode semantically coherent expressions. A large portion of them correspond to sentiment (the five top examples in the table), but others capture different semantics, e.g., time expressions.

Second, it seems our patterns are relatively soft, and allow lexical flexibility. While some patterns do seem to fix specific words, e.g., "of" in the first example or "minutes" in the last one, even in those cases some of the top matching spans replace these words with other, similar words ("with" and "half-hour", respectively). Encouraging SoPa to have more concrete words, e.g., by jointly learning the word vectors, might make SoPa useful in other contexts, particularly as a decoder. We defer this direction to future work.

Finally, SoPa makes limited but non-negligible use of self-loops and epsilon steps. Interestingly, the second example shows that one of the pat-

Analyzed Documents
it 's dumb , **but more importantly** , *it 's just not scary*
though moonlight mile is replete with **acclaimed actors and actresses** and tackles a subject that 's **potentially moving** , the movie is *too predictable* and *too self-conscious to reach a* level of **high drama**
While **its careful pace and** seemingly *opaque story* may not satisfy every moviegoer 's appetite, the film 's final scene is **soaringly , transparently moving**
unlike the speedy wham-bam effect *of most hollywood offerings* , character development – and more **importantly, character empathy – is at the heart of** italian for beginners .
the band 's courage in the face of official repression **is inspiring , especially for** aging *hippies* (this one included) .

Table 3: Documents from the SST training data. Phrases with the largest contribution toward a positive sentiment classification are in **bold green**, and the most negative phrases are in *italic orange*.

terns had an ϵ-transition at the same place in every phrase. This demonstrates a different function of ϵ-transitions than originally designed—they allow a pattern to effectively shorten itself, by learning a high ϵ-transition parameter for a certain state.

Interpreting a document. SoPa provides an interpretable representation of a document—a vector of the maximal matching score of each pattern with any span in the document. To visualize the decisions of our model for a given document, we can observe the patterns and corresponding phrases that score highly within it.

To understand which of the k patterns contributes most to the classification decision, we apply a leave-one-out method. We run the forward method of the MLP layer in SoPa k times, each time zeroing-out the score of a different pattern p. The difference between the resulting score and the original model score is considered p's contribution. We then consider the highest contributing patterns, and attach each one with its highest scoring phrase in that document. Table 3 shows example texts along with their most positive and negative contributing phrases.

8 Related Work

Weighted finite-state automata. WFSAs and hidden Markov models[19] were once popular in automatic speech recognition (Hetherington, 2004; Moore et al., 2006; Hoffmeister et al., 2012) and remain popular in morphology (Dreyer, 2011; Cotterell et al., 2015). Most closely related to this work, neural networks have been combined with weighted finite-state transducers to do morphological reinflection (Rastogi et al., 2016). These prior works learn a single FSA or FST, whereas our model learns a collection of simple but complementary FSAs, together encoding a sequence. We are the first to incorporate neural networks both *before* WFSAs (in their transition scoring functions), and *after* (in the function that turns their vector of scores into a final prediction), to produce an expressive model that remains interpretable.

Recurrent neural networks. The ability of RNNs to represent arbitrarily long sequences of embedded tokens has made them attractive to NLP researchers. The most notable variants, the long short-term memory (LSTM; Hochreiter and Schmidhuber, 1997) and gated recurrent units (GRU; Cho et al., 2014), have become ubiquitous in NLP algorithms (Goldberg, 2016). Recently, several works introduced simpler versions of RNNs, such as recurrent additive networks (Lee et al., 2017) and Quasi-RNNs (Bradbury et al., 2017). Like SoPa, these models can be seen as points along the bridge between RNNs and CNNs.

Other works have studied the expressive power of RNNs, in particular in the context of WFSAs or HMMs (Cleeremans et al., 1989; Giles et al., 1992; Visser et al., 2001; Chen et al., 2018). In this work we relate CNNs to WFSAs, showing that a one-layer CNN with max-pooling can be simulated by a collection of linear-chain WFSAs.

Convolutional neural networks. CNNs are prominent feature extractors in NLP, both for generating character-based embeddings (Kim et al., 2016), and as sentence encoders for tasks like text classification (Yin and Schütze, 2015) and machine translation (Gehring et al., 2017). Similarly to SoPa, several recently introduced variants of CNNs support varying window sizes by either allowing several fixed window sizes (Yin and Schütze, 2015) or by supporting non-consecutive n-gram matching (Lei et al., 2015; Nguyen and Grishman, 2016).

Neural networks and patterns. Some works used patterns as part of a neural network. Schwartz et al. (2016) used pattern contexts for estimating word embeddings, showing improved word similarity results compared to bag-of-word

[19]HMMs are a special case of WFSAs (Mohri et al., 2002).

contexts. Shwartz et al. (2016) designed an LSTM representation for dependency patterns, using them to detect hypernymy relations. Here, we learn patterns as a neural version of WFSAs.

Interpretability. There have been several efforts to interpret neural models. The weights of the attention mechanism (Bahdanau et al., 2015) are often used to display the words that are most significant for making a prediction. LIME (Ribeiro et al., 2016) is another approach for visualizing neural models (not necessarily textual). Yogatama and Smith (2014) introduced structured sparsity, which encodes linguistic information into the regularization of a model, thus allowing to visualize the contribution of different bag-of-word features.

Other works jointly learned to encode text and extract the span which best explains the model's prediction (Yessenalina et al., 2010; Lei et al., 2016). Li et al. (2016) and Kádár et al. (2017) suggested a method that erases pieces of the text in order to analyze their effect on a neural model's decisions. Finally, several works presented methods to visualize deep CNNs (Zeiler and Fergus, 2014; Simonyan et al., 2014; Yosinski et al., 2015), focusing on visualizing the different layers of the network, mainly in the context of image and video understanding. We believe these two types of research approaches are complementary: inventing general purpose visualization tools for existing black-box models on the one hand, and on the other, designing models like SoPa that are interpretable by construction.

9 Conclusion

We introduced SoPa, a novel model that combines neural representation learning with WFSAs. We showed that SoPa is an extension of a one-layer CNN. It naturally models flexible-length spans with insertion and deletion, and it can be easily customized by swapping in different semirings. SoPa performs on par with or strictly better than four baselines on three text classification tasks, while requiring fewer parameters than the stronger baselines. On smaller training sets, SoPa outperforms all four baselines. As a simple version of an RNN, which is more expressive than one-layer CNNs, we hope that SoPa will encourage future research on the bridge between these two mechanisms. To facilitate such research, we release our implementation at `https://github.com/Noahs-ARK/soft_patterns`.

Acknowledgments

We thank Dallas Card, Elizabeth Clark, Peter Clark, Bhavana Dalvi, Jesse Dodge, Nicholas FitzGerald, Matt Gardner, Yoav Goldberg, Mark Hopkins, Vidur Joshi, Tushar Khot, Kelvin Luu, Mark Neumann, Hao Peng, Matthew E. Peters, Sasha Rush, Ashish Sabharwal, Minjoon Seo, Sofia Serrano, Swabha Swayamdipta, Chenhao Tan, Niket Tandon, Trang Tran, Mark Yatskar, Scott Yih, Vicki Zayats, Rowan Zellers, Luke Zettlemoyer, and several anonymous reviewers for their helpful advice and feedback. This work was supported in part by NSF grant IIS-1562364, by the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by NSF grant ACI-1548562, and by the NVIDIA Corporation through the donation of a Tesla GPU.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *Proc. of ICLR*.

Leonard E. Baum and Ted Petrie. 1966. Statistical Inference for Probabilistic Functions of Finite State Markov Chains. *The Annals of Mathematical Statistics*, 37(6):1554–1563.

James Bradbury, Stephen Merity, Caiming Xiong, and Richard Socher. 2017. Quasi-Recurrent Neural Network. In *Proc. of ICLR*.

Yining Chen, Sorcha Gilroy, Kevin Knight, and Jonathan May. 2018. Recurrent neural networks as weighted language recognizers. In *Proc. of NAACL*.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proc. of EMNLP*.

Axel Cleeremans, David Servan-Schreiber, and James L McClelland. 1989. Finite state automata and simple recurrent networks. *Neural computation*, 1(3):372–381.

Ryan Cotterell, Nanyun Peng, and Jason Eisner. 2015. Modeling word forms using latent underlying morphs and phonology. *TACL*, 3:433–447.

Dmitry Davidov and Ari Rappoport. 2008. Unsupervised discovery of generic relationships using pattern clusters and its evaluation by automatically generated SAT analogy questions. In *Proc. of ACL*.

Dmitry Davidov, Oren Tsur, and Ari Rappoport. 2010. Enhanced sentiment learning using twitter hashtags and smileys. In *Proc. of COLING*.

Markus Dreyer. 2011. *A Non-parametric Model for the Discovery of Inflectional Paradigms from Plain Text Using Graphical Models over Strings*. Ph.D. thesis, Johns Hopkins University, Baltimore, MD, USA.

Manfred Droste and Paul Gastin. 1999. The Kleene–Schützenberger theorem for formal power series in partially commuting variables. *Information and Computation*, 153(1):47–80.

Jason Eisner. 2002. Parameter estimation for probabilistic finite-state transducers. In *Proc. of ACL*.

Jason Eisner. 2016. Inside-outside and forward-backward algorithms are just backprop (tutorial paper).

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

Oren Etzioni, Michael Cafarella, Doug Downey, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S. Weld, and Alexander Yates. 2005. Unsupervised named-entity extraction from the web: An experimental study. *Artificial intelligence*, 165(1):91–134.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann Dauphin. 2017. Convolutional sequence to sequence learning. In *Proc. of ICML*.

C. Lee Giles, Clifford B Miller, Dong Chen, Hsing-Hen Chen, Guo-Zheng Sun, and Yee-Chun Lee. 1992. Learning and extracting finite state automata with second-order recurrent neural networks. *Neural Computation*, 4(3):393–405.

Yoav Goldberg. 2016. A primer on neural network models for natural language processing. *JAIR*, 57:345–420.

Joshua Goodman. 1999. Semiring parsing. *Computational Linguistics*, 25(4):573–605.

Marti A. Hearst. 1992. Automatic acquisition of hyponyms from large text corpora. In *Proc. of COLING*.

Lee Hetherington. 2004. The MIT finite-state transducer toolkit for speech and language processing. In *Proc. of INTERSPEECH*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Björn Hoffmeister, Georg Heigold, David Rybach, Ralf Schlüter, and Hermann Ney. 2012. WFST enabled solutions to ASR problems: Beyond HMM decoding. *IEEE Transactions on Audio, Speech, and Language Processing*, 20:551–564.

Mohit Iyyer, Varun Manjunatha, Jordan Boyd-Graber, and Hal Daumé III. 2015. Deep unordered composition rivals syntactic methods for text classification. In *Proc. of ACL*.

Ákos Kádár, Grzegorz Chrupała, and Afra Alishahi. 2017. Representation of linguistic form and function in recurrent neural networks. *Computational Linguistics*, 43:761–780.

Yoon Kim. 2014. Convolutional Neural Networks for Sentence Classification. In *Proc. of EMNLP*.

Yoon Kim, Yacine Jernite, David Sontag, and Alexander M. Rush. 2016. Character-aware neural language models. In *Proc. of AAAI*.

Diederik Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *Proc. of ICLR*.

Yann LeCun. 1998. Gradient-based Learning Applied to Document Recognition. In *Proc. of the IEEE*.

Kenton Lee, Omer Levy, and Luke Zettlemoyer. 2017. Recurrent additive networks. arXiv:1705.07393.

Tao Lei, Regina Barzilay, and Tommi Jaakkola. 2015. Molding CNNs for text: non-linear, non-consecutive convolutions. In *Proc. of EMNLP*.

Tao Lei, Regina Barzilay, and Tommi Jaakkola. 2016. Rationalizing Neural Predictions. In *Proc. of EMNLP*.

Jiwei Li, Will Monroe, and Dan Jurafsky. 2016. Understanding Neural Networks through Representation Erasure. arXiv:1612.08220.

Dekang Lin, Shaojun Zhao, Lijuan Qin, and Ming Zhou. 2003. Identifying synonyms among distributionally similar words. In *Proc. of IJCAI*.

Julian McAuley and Jure Leskovec. 2013. Hidden factors and hidden topics: understanding rating dimensions with review text. In *Proc. of RecSys*.

Mehryar Mohri. 1997. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23:269–311.

Mehryar Mohri, Fernando Pereira, and Michael Riley. 2002. Weighted finite-state transducers in speech recognition. *Computer Speech & Language*, 16(1):69–88.

Darren Moore, John Dines, Mathew Magimai-Doss, Jithendra Vepa, Octavian Cheng, and Thomas Hain. 2006. Juicer: A weighted finite-state transducer speech decoder. In *MLMI*.

Nasrin Mostafazadeh, Nathanael Chambers, Xiaodong He, Devi Parikh, Dhruv Batra, Lucy Vanderwende, Pushmeet Kohli, and James Allen. 2016. A corpus and cloze evaluation for deeper understanding of commonsense stories. In *Proc. of NAACL*.

Thien Huu Nguyen and Ralph Grishman. 2016. Modeling Skip-Grams for Event Detection with Convolutional Neural Networks. In *Proc. of EMNLP*.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. 2011. Scikit-learn: Machine learning in Python. *JMLR*, 12:2825–2830.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global vectors for word representation. In *Proc. of EMNLP*.

Pushpendre Rastogi, Ryan Cotterell, and Jason Eisner. 2016. Weighting finite-state transductions with neural context. In *Proc. of NAACL*.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. "Why should I trust you?": Explaining the predictions of any classifier. In *Proc. of KDD*.

Dana Rubinstein, Effi Levi, Roy Schwartz, and Ari Rappoport. 2015. How well do distributional models capture different types of semantic knowledge? In *Proc. of ACL*.

Jacques Sakarovitch. 2009. Rational and recognisable power series. In Manfred Droste, Werner Kuich, and Heiko Vogler, editors, *Handbook of Weighted Automata*, pages 105–174. Springer.

M. P. Schützenberger. 1961. On the definition of a family of automata. *Information and Control*, 4(2):245–270.

Roy Schwartz, Roi Reichart, and Ari Rappoport. 2015. Symmetric pattern based word embeddings for improved word similarity prediction. In *Proc. of CoNLL*.

Roy Schwartz, Roi Reichart, and Ari Rappoport. 2016. Symmetric patterns and coordinations: Fast and enhanced representations of verbs and adjectives. In *Proc. of NAACL*.

Roy Schwartz, Maarten Sap, Ioannis Konstas, Li Zilles, Yejin Choi, and Noah A. Smith. 2017. The effect of different writing tasks on linguistic style: A case study of the roc story cloze task. In *Proc.of CoNLL*.

Vered Shwartz, Yoav Goldberg, and Ido Dagan. 2016. Improving hypernymy detection with an integrated path-based and distributional method. In *Proc. of ACL*.

Hava T. Siegelmann and Eduardo D. Sontag. 1995. On the computational power of neural nets. *Journal of Computer and System Sciences*, 50(1):132–150.

Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. 2014. Deep inside convolutional networks: Visualising image classification models and saliency maps. In *Proc. of ICLR Workshop*.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proc. of EMNLP*.

Oren Tsur, Dmitry Davidov, and Ari Rappoport. 2010. ICWSM–a great catchy name: Semi-supervised recognition of sarcastic sentences in online product reviews. In *Proc. of ICWSM*.

Yulia Tsvetkov, Manaal Faruqui, Wang Ling, Guillaume Lample, and Chris Dyer. 2015. Evaluation of word vector representations by subspace alignment. In *Proc. of EMNLP*.

Ingmar Visser, Maartje EJ Raijmakers, and Peter CM Molenaar. 2001. Hidden markov model interpretations of neural networks. In *Connectionist Models of Learning, Development and Evolution*, pages 197–206. Springer.

Andrew Viterbi. 1967. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269.

Ainur Yessenalina, Yisong Yue, and Claire Cardie. 2010. Multi-level structured models for document-level sentiment classification. In *Proc. of EMNLP*.

Wenpeng Yin and Hinrich Schütze. 2015. Multichannel Variable-Size Convolution for Sentence Classification. In *Proc. of CoNLL*.

Dani Yogatama, Lingpeng Kong, and Noah A. Smith. 2015. Bayesian optimization of text representations. In *Proc. of EMNLP*.

Dani Yogatama and Noah A. Smith. 2014. Linguistic structured sparsity in text categorization. In *Proc. of ACL*.

Jason Yosinski, Jeff Clune, Anh Mai Nguyen, Thomas J. Fuchs, and Hod Lipson. 2015. Understanding neural networks through deep visualization. In *Proc. of the ICML Deep Learning Workshop*.

Matthew D. Zeiler and Rob Fergus. 2014. Visualizing and understanding convolutional networks. In *Proc. of ECCV*.

Peng Zhou, Zhenyu Qi, Suncong Zheng, Jiaming Xu, Hongyun Bao, and Bo Xu. 2016. Text classification improved by integrating bidirectional LSTM with two-dimensional max pooling. In *Proc. of COLING*.

Appendices

A Experimental Setup

We implemented all neural models in PyTorch,[20] and the **Hard** baseline in scikit-learn (Pedregosa et al., 2011).[21] We train using Adam (Kingma and Ba, 2015) with a batch size of 150. We use 300-dimensional GloVe 840B embeddings (Pennington et al., 2014) normalized to unit length. We randomly initialize all other parameters. Our MLP has two layers. For regularization, we use dropout.[22]

In all cases, we tune the hyperparameters of our model on the development set by running 30 iterations of random search. The full list of hyperparameters explored for each model can be found in Table 4. Finally, we train all models for 250 epochs, stopping early if development loss does not improve for 30 epochs.

Type	Values	Models
Patterns	{5:10,4:10,3:10,2:10}, {6:10,5:10,4:10}, {6:10,5:10,4:10,3:10,2:10}, {6:20,5:20,4:10,3:10,2:10}, {7:10,6:10,5:10,4:10,3:10,2:10}	**SoPa**
Learning rate	0.01, 0.05, 0.001, 0.005	**SoPa, DAN, BiLSTM, CNN**
Dropout	0, 0.05, 0.1, 0.2	**SoPa, BiLSTM, CNN**
MLP hid. dim.	10, 25, 50, 100, 300	**SoPa, DAN, BiLSTM, CNN**
Hid. layer dim.	100, 200, 300	**BiLSTM**
Out. layer dim.	50, 100, 200	**CNN**
Window size	4, 5, 6	**CNN**
Word dropout	0.1, 0.2, 0.3, 0.4	**DAN**
Log. reg. param	1, 0.5, 0.1, 0.05, 0.01, 0.005, 0.001	**Hard**
Min. pattern freq.	2–10, 0.1%	**Hard**

Table 4: The hyperparameters explored in our experiments. **Patterns**: the number of patterns of each length. For example, {5:20,4:10} means 20 patterns of length 5 and 10 patterns of length 4. **MLP hid. dim.**: the dimension of the hidden layer of the MLP. **Hid. layer dim.**: the BiLSTM hidden layer dimension. **Out. layer dim.**: the CNN output layer dimension. **Window size**: the CNN window size. **Log. reg. param**: the logistic regression regularization parameter. **Min. pattern freq.**: minimum frequency for a pattern to be included as a logistic regression feature, expressed either as absolute count or as relative frequency in the train set. **Models**: the models to which each hyperparameter applies (see Section 5).

[20]`https://pytorch.org/`

[21]`http://scikit-learn.org/`

[22]**DAN** uses word dropout instead of regular dropout as its only learnable parameters are the MLP layer weights.