Exploring the SoPa Model

263-5352-00L Final Project (Spring 2022)

Quynh Anh Nguyen, **William Andersson** and **Michele Meziu**
ETH Zürich University of Milan
{quynguyen, anwillia, mezium}@ethz.ch

Abstract

The project aims to explore the SoPa model (Schwartz et al., 2018), which is characterized by the patterns matching mechanism. As in the SoPa experiment, 100 movie reviews from SST dataset with binary labels are also used in our ablation study. We executed the study on three levels, including automaton level, automata level, as well as on the SoPa as a whole. We conclude valuable insights regarding how parameters affect the model performance through multiple experiments. First, increasing patterns' width improves model performance while the length has little effect. Second, in case excluding ϵ-transitions, the bigger max-steps-forward is, the better performance is yielded. Also, there is a downward trend in test accuracy as we increase the value of `shared_sl` because it decreases the complexity of the model. Third, the log-space max-times semiring achieves the best accuracy. Finally, training the patterns on multiple documents improves the gradient computation and hence the convergence rate.

1 Introduction

The goal of this project is to discover SoPa model from the paper *SoPa: Bridging CNNs, RNNs, and Weighted Finite-State Machines* (Schwartz et al., 2018). The paper proposes a model based on learning weights for underlying WFSAs. The primary focus of our investigation will be to find out how the hyper parameters and structure of each WFSA affects the model performance. This is an interesting experiment as it allows us to apply our knowledge gained from Advanced Formal Language Theory lectures. Considering the authors' lack of explanation about their chosen WFSA structure and the semiring used to encode its weights, the project provides useful insight into what motivated their decisions.

Moreover, in the original paper (Schwartz et al., 2018), the model hyperparameters are tuned using random search, without clarifying what role they play in the model. We will experiment with various model hyperparameters to examine their impact on model performance.

We were not able to find any related work that builds on SoPa or investigates it. Enhancing our understanding of SoPa will hopefully motivate researchers to use this model in their work and try to improve it.

2 Background and Preliminaries

SoPa is introduced as a WFSA-based model and is designed to represent text as a collection of surface pattern occurrences (Schwartz et al., 2018). Patterns (Giles et al., 1992) are particularly useful tool in NLP (Lin et al., 2003; Etzioni et al., 2005; Schwartz et al., 2015). In this section, we will briefly introduce the model regarding how each pattern is formed, how to measure the match between a pattern and a text spans, a documents as well as how multi-patterns matches multi text spans.

WFSA - a matching pattern

Classical pattern A pattern is a sequence of words and wildcards. By replacing wildcards with concrete words, these patterns can be used to extract specific text spans which have the same structure. For example, in the sequence `'X is a Y'`, **X, Y** are wildcards and **is, a** are words. Replacing **X, Y** by `{'Mickey', 'cat'}`, or `{'Tom', 'dog'}`, we can match the pattern with the text span `'Mickey is a cat'` and `'Tom is a dog'`. Those mentioned text spans have the same 'pattern' `'X is a Y'`. However, this classical definition of a pattern imposes a strict rule on how a they can be matched with a text span. Flexible hard pattern (Davidov et al., 2010), which supports partial matching of the pattern, was thus introduced.

Flexible hard pattern While the classical type of pattern is rigid in term of text span matching, flexible hard pattern (Davidov et al., 2010) is in a *soft* manner as it can be matched with a given text

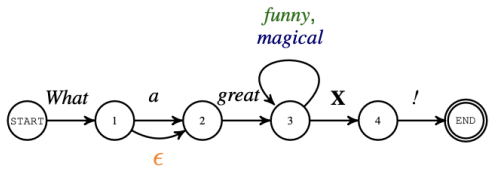

Figure 1: A representation of a surface pattern as a six-state automaton (Giles et al., 1992). Self-loops allow for repeatedly inserting words (e.g., "funny"). ε-transitions allow for dropping words (e.g., "a").

by skipping some of the words in the pattern, or introducing new words. Similar to flexible hard pattern, SoPa model (Schwartz et al., 2018) is designed as a collection of WFSAs that supports partial matching mechanism. The WFSAs in SoPa are also called soft patterns. Soft patterns are not only flexible in text span matching but also have the transition weights which are given by differentiable functions. Through differentiable functions, every single WFSA has the power to capture concrete words, wildcards, and everything in between.

Matching a text span to a pattern

Transition Matrix A WFSA is parametrized by a transition matrix T. In SoPa model, each transition matrix T has three diagonals, corresponding to three possible outgoing transitions: self-loop transitions, main path transitions and ε-transitions[1]. Figure 1 illustrates an example of a WFSA, e.g. a soft pattern. A WFSA with d states will tend to match text spans of $d-1$ length [2]. The transition function T is a parametrized function that returns a $d \times d$ matrix. For a word x:

$$[\mathbf{T}(x)]_{i,j} = \begin{cases} E(u_i.v_x + a_i), & \text{if } j = i \\ E(w_i.v_x + b_i), & \text{if } j = i+1 \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

$$[\mathbf{T}(\epsilon)]_{i,j} = \begin{cases} E(c_i), & \text{if } j = i+1 \\ 0, & \text{otherwise} \end{cases} \quad (2)$$

Equation 1 shows how main path transition are parametrized. In particular, where u_i and w_i are vectors of parameters, a_i and b_i are scalar parameters, v_x is a fixed pre-trained word vector represented for word x[3], and E is an encoding function, typically the identity function or sigmoid. ϵ-transitions are also parametrized in equation 2.

Scoring text spans A WFSA-ε with d states over a vocabulary V is formally defined as a tuple $F = (\pi, T, \eta)$ [4]. Given a sequence of words in the vocabulary $x = [x_1, ..., x_n]$, the Forward algorithm (Baum and Petrie, 1966) scores x with respect to F. Forward can be written as a series of matrix multiplications:

$$p_{span}(x) = \pi^{\mathbf{T}} \mathbf{T}(\varepsilon)^* \left(\prod_{i=1}^{n} \mathbf{T}(x_i) \mathbf{T}(\varepsilon)^* \right) \eta \quad (3)$$

where $\mathbf{T}(\varepsilon)^* := \sum_{j=0}^{\infty} \mathbf{T}(\varepsilon)^j$

Matching a document to a pattern

Assume we have a single pattern. We will score over all the matches between the unique pattern and possible text spans in the document. The score is aggregated using different formulas, depending on the semiring. When using the Viterbi algorithm, the document score is computed as $s_{doc}(\mathbf{x}) = max_{1<i<j<n} s_{span}(\mathbf{x}_{i:j})$. When using the Forward algorithm, the document score is $\sum_{1<i<j\leq n} p_{span}(\mathbf{x}_{i:j})$. Equation 3 shows how each p_{span} is scored with the Forward algorithm. In SoPa (Schwartz et al., 2018), authors experimented using Viterbi algorithm.

Matching a document to multiple patterns

Beyond the unique pattern assumption, we now consider SoPa model with multiple patterns. We describe how SoPa model scores over all matches between given patterns and multiple text spans within the document using Viterbi algorithm.

Given k patterns and the document, the model yields k different s_{doc} scores since patterns to typically occur at most once in Viterbi algorithm. These score are stacked into a vector $z \in \mathbb{R}^k$ and constitute the final document representation. This representation vector is then fed into a multilayer perceptron (MLP) (Rosenblatt, 1958) to get the probability distribution over document labels.

[1] A self-loop transition allows the pattern to consume a word without moving states, a main path transition to state $i+1$ allows the pattern to consume one token and move forward one state, and an ε-transition to state $i+1$ allows the pattern to move forward one state without consuming a token. All other transitions are given score 0

[2] The spans length are possibly shorter due to ε-transitions or longer due to self-loops.

[3] GloVe 300d 840B word embeddings (Pennington et al., 2014a)

[4] $\pi \in \mathbb{R}^d$ is an initial weight vector, $T : (V \cup \varepsilon) \rightarrow R^d$ is a transition weight function, and $\eta \in \mathbb{R}^d$ is a final weight vector. When processing a sequence of text with a pattern p, it is started with a special START state, and only move forward (or stay put), until the special END state is reached. Thus, SoPa was setup by fixing $\pi = [1, 0, ..., 0] and \eta = [0, ..., 0, 1]$.

3 Experimental Setup

We performed a variety of experiments to elucidate the effects of various hyperparameters on the SoPa model.

3.1 Dataset

In order to have a comparable analysis, we use the one of the datasets used for SoPa model's initial experiments. These are the Stanford Sentiment Treebank (Socher et al., 2013), Amazon Review Corpus[5], and the ROC story cloze task[6]. In our experiment, we use the provided SST dataset of movie reviews and ratings (Socher et al., 2013). The labels are binarised: a label of 0 represents negative sentiment, whereas a label of 1 represents a positive sentiment. From the provided data, we designed identically experiments as well as exploring SoPa model using 100 training samples. Hence, it could be a reason for the low performance relative to what the paper reports for the full dataset.

3.2 Baseline Model

Our code is forked from the paper's implementation, as we don't believe it would be productive to rewrite it. To perform our experiments we will have to make (somewhat extensive) modifications to it, in particular to generalize it for our experiments.

Before we dive into the experiments themselves, we should state the base hyperparameters. Unless specified otherwise, they are used for all experiments. All were found via grid search on the hyperparameter set the authors provide in their experimental setup (Schwartz et al., 2018). The specific ones we found to work best in general are as follows, while the ones not in the list are left as the default setting in their implementation.

Finally, models used to find test accuracy are chosen by hand, based on performance on the validation set (dev) at training time. Unless specified otherwise, we use the same hyperparameters for the rest of our experiments as well.

- Patterns: {7:10, 6:10, 5:10, 4:10, 3:10, 2:10}[7]
- Semiring: log-space
- Learning rate: 0.001
- Dropout: 0
- MLP hidden layer dimension: 100
- Epochs: 250
- Patience: 30 epochs[8]

As in the paper, we use the max-product semiring (actually implemented as `LogSpaceMaxTimesSemiring` in the codebase). For the same reason, we use the GloVe "Common Crawl" embedding with 840B tokens and 300-dimensional vectors[9] (Pennington et al., 2014b).

3.3 Semiring experiments

The first investigation we did was into the various semirings available. The authors mention two semirings: the probability semiring and the max-times (aka max-product, Viterbi) semiring (Schwartz et al., 2018). They decide to use the max-product semiring because "in short documents, [they] expect patterns to typically occur at most once" (Schwartz et al., 2018). At an intuitive level this makes sense: multiplying scores and taking the maximum will favor uncommon patterns more than using the pattern's expected count (in the probability semiring).

However, no further justification is offered. In fact, their implementation of the max-times semiring is actually done in log space, where multiplication is done by adding the logarithm of the two values. Their codebase also contains implementations of the probability and max-plus semirings, though the latter is not mentioned in their paper.

In order to find out why the authors made these decisions, we decided to run experiments on a variety of semirings, beginning with the three for already implemented. See table 1.

The next problem is to decide how to initialize the transition weights. In the authors' implementation (Schwartz et al., 2018), they sample from a standard normal distribution and use this to initialize all weights (after normalising), disregarding the particular semiring at hand. This leads to very small weights, on the order of 10^{-6}. Qualitative analysis found that starting from such an initialisation, the weights converge to around 10^{-4} as the model learns. Another issue is that such sampling

[5] http://riejohnson.com/cnn_data.html

[6] https://www.cs.rochester.edu/nlp/rocstories/

[7] m:n is to be interpreted as n WFSAs (patterns) with m states.

[8] Stop early if validation loss hasn't improved after this many epochs

[9] glove.840B.300d

Semiring	Set	$\oplus$	$\otimes$	$\overline{\mathbf{0}}$	$\overline{\mathbf{1}}$
Probability	$\mathbb{R}^+$	$+$	$\times$	0	1
Max-plus	$\mathbb{R} \cup \{-\infty\}$	$\max$	$+$	$-\infty$	0
Log-space max-times	$\mathbb{R}^+$	$\max$	$+$	$-\infty$	0
Max-times	$\mathbb{R}^+$	$\max$	$\times$	0	1
Boolean	$\{0, 1\}$	$\vee$	$\wedge$	0	1
Viterbi	$[0, 1]$	$\max$	$\times$	0	1
Min-plus	$\mathbb{R} \cup \{+\infty\}$	$\min$	$+$	$+\infty$	0

Table 1: Semirings used in our experiments. Attributed to (Huang, 2008) and the lecture notes of *Natural Language Processing*, by Prof. R. Cotterell (2021)

produces negative weights, which is not valid for the probability semiring.[10]

Despite these questionable properties, some cursory experiments with other initialisation techniques (without normalisation, shifting the distribution, sampling from a uniform distribution) showed that the authors' sampling technique *is* the most effective.

As a result, we based most of our initialisation around normal distributions. For the semirings that are defined on $\mathbb{R}$, we did not change the initialisation. For those defined on $\mathbb{R}^+$, we sample from the Gaussian distribution $\mathcal{N}(2, 1)$, where the idea is to avoid putting much mass in the negative domain, which will be rounded up to 0 anyway. For the Boolean semiring, we sample each weights randomly from $\{0, 1\}$. Finally, for the Viterbi semiring, we sample from $\mathcal{N}(0.5, 0.5)$ and clamp to $[0, 1]$. Again the goal is to center most of the mass inside the valid domain, here centered in the middle of it.

3.4 Pattern experiments

Another aspect of the SopA model we investigated was the pattern architecture itself. Each pattern can be described by its "width" (how many of each WFSAs there are) and its "length" (what is the longest WFSA). In particular, we adhere to the convention set out by the authors (Schwartz et al., 2018) that each WFSA with longest pattern m also contains WFSAs with length $m-1, m-2, ..., 2$. Our goal is to understand how these two dimensions affect the expressivity of the WFSAs and therefore the SoPa model.

In principle we would assume that the more WFSAs in the pattern set, the more expressive the model is. In particular, we should not lose accuracy by allowing larger or longer WFSAs, though with diminishing returns. This is because a WFSA encodes–in a certain sense–a sentence, and so with epsilon transitions a longer WFSA should be more general.

As stated in (Schwartz et al., 2018), the authors found that pattern lengths between 2 and 7, widths of 10 or 20, and total number of patterns between 30 and 70 produced the best results. Our initial hyperparameter setup, we verified this claim: the longest pattern amongst those they suggested proved to give the highest accuracy. Nevertheless, we believe it is still worth exploring, as the the authors provide little justification for why those patterns work well.

To begin with, we look at the length of patterns. To ensure we are only adjusting one dimension, we keep the total number of patterns constant (at 90) for each of the pattern specifications. We then experiment with pattern specifications that have patterns ranging from length 2 to length 10.

For the next part of the pattern experiments, we change the number of WFSAs available while keeping the lengths constant. We range the widths from 2 to 100 and perform the experiment on lengths 3, 4, and 5.

3.5 Diagonals experiments

We now turn to the transition matrix $\mathbf{T}$, and investigate its diagonal. In particular, with these experiments, we would like to examine how the presence of ε-transitions and self-loops affect the model's performance.

As mentioned in section 2, patterns are parameterised by a transition matrix $\mathbf{T}$. The matrix $\mathbf{T(x)}$ includes two diagonals, whereby one represents the self-loop transition and the other represents one-steps-forward transitions of the automata. When the pattern has ε-transitions, we also need to weight

[10] Their implementation of the probability semiring did not ensure weights are non-negative; we fixed this oversight.

up $\mathbf{T}(\varepsilon)$ matrix as shown in equation 3.

Within SoPa (Schwartz et al., 2018), $\mathbf{T}(\varepsilon)^* := \sum_{j=0}^{\infty} \mathbf{T}(\varepsilon)^j$ in equation 3 is estimated as

$$\mathbf{T}(\varepsilon)^* \approx I + \mathbf{T}(\varepsilon) \tag{4}$$

Without ε-transitions To avoid the approximation in equation 4, we propose a novel way to represent these soft patterns. In particular, we consider not only 1 step forward transitions but also `{2}` steps forward transitions and (separately) `{ 3, 4, 5, 6}` steps forward transitions. In case of Viterbi algorithm, the equation 3 scoring the match of a span and a pattern becomes $p_{span}(x) = \pi^{\mathbf{T}} \left(\prod_{i=1}^{n} \mathbf{T}(x_i)\right) \eta$. For example, each pair of transitions in which a ε-transition follows a normal 1-step transition is then replaced by a 2-steps-transition.

The model is now parametrized by a transition matrix which has more than two diagonals, i.e., we consider 2-steps forward and larger steps-forward transitions in the automata. For the implementation, we set the parameter `no_eps = True`, `no_sl = True/False` and assign different values to `num_diags` depending on the number of steps forward.

With ε-transitions In the second part of the diagonal experiments, we do not only increase the number of diagonals but also enable the possibility of ε-transitions in every single automata. The parameter `no_eps = False` is set, which allows the existence of ε-transitions. The setup allows us to observe how SoPa model performs with adding extra steps forward transitions; whereas, in the previous setup, we aimed to observe the model performance when ε-transitions are replaced by other $2+$ step forward transitions.

3.6 Self-loop weight sharing experiments

In the implementation provided by the original SoPa paper (Schwartz et al., 2018) we find the parameter `shared_sl`. This determines whether weights are shared between edges along the main path and self-loops, giving three possible different options.

- `shared_sl = 0`: in the WFSA there is a separate weight for each self-loop and for each edge along the main path. This is the default option. In the probabilistic interpretation of the WFSA weights, this corresponds to making no assumption on the relation between the probability of taking the self loop or consuming a token.

- `shared_sl = 1`: the self-loop and the forward edge starting at node q share both the same weight. However, for each self loop in the WFSA there is a learnable scalar parameter that is multiplied with the weight to determine the true final weight of the self loop. This setting corresponds to assuming that the probability of taking a self-loop is a proportion of the probability of consuming a token, where the proportionality constant depends on the WFSA state.

- `shared_sl = 2`: the self-loop and the forward edge starting at a state q share the same weight. However, there is a unique scalar parameter that is multiplied with the weight to compute the final weight of the self-loop. With this model, we assume that that the probability of taking a self-loop is a fixed proportion of the probability of consuming a token. Since there is only one weight for each WFSA, this model has far fewer parameters than one with `shared_sl = 1`.

We recall that weights are real-valued vectors that are used to compute the dot product with the word embedding, as explained in equation 1.

We will train our model with all the possible values of the `shared_sl` parameter, combined with a transition matrix with two or three diagonals. We will verify with experiments how the `shared_sl` parameter influences training and test accuracy.

3.7 Batch size experiments

The provided implementation of SoPa of (Schwartz et al., 2018) allows training with batches of documents. Thus, instead of computing the WFSA scores on a single document, we do it on multiple documents. Those scores are then used as input of a multilayer perceptron and the error is used to update the automata weights. This improves training by obtaining more accurate estimates of the gradient that are based on the score on many documents.

We will train the model using various batch sizes and with two or three diagonals the transition matrix. We will report how the batch size influences the accuracy of the model.

4 Results and Discussion

4.1 Baseline model

We found the model to be very sensitive to learning rate. The ones larger than 0.001 would lead to degenerate behavior which leads to an accuracy of 49% on the SST dataset, where random predictions would give 50% accuracy.

With this experimental setup we achieved a test accuracy of 75.29% (validation accuracy 74.80%) when taking the model at epoch 36.

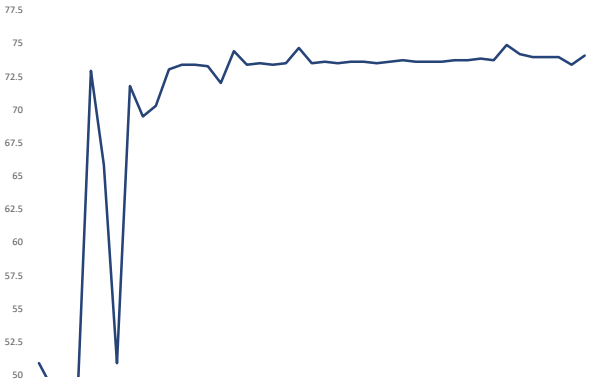

Figure 2: Baseline validation accuracy per epoch

4.2 Semiring experiments

Final results are reported in table 2, while convergence results are presented in figure 3. We report validation accuracy since we are interested more in the general convergence behavior than performance[11]. We now proceed with an analysis of 3.

We observe three strata, here listed in order of decreasing accuracy: log-space max-times, min-plus, and max-plus; probability and Viterbi; max-times. The Boolean semiring failed to achieve an accuracy over 50%, and in fact converged to around 47%. This is unsurprising, as Boolean values are not enough to capture the language semantics.

[11]Though validation and test accuracy are strongly correlated, see table 2.

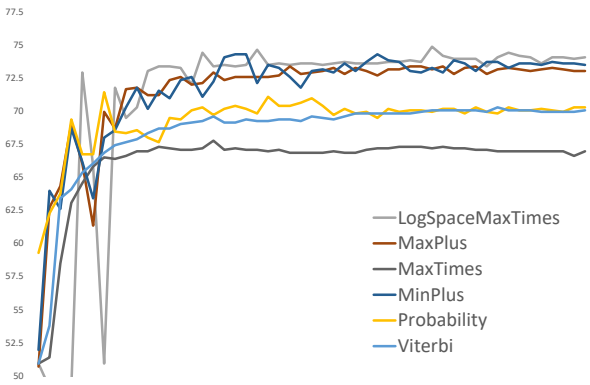

Figure 3: Validation accuracy per epoch, various semirings

Another interesting result is that the log-space max-times semiring performs significantly better than the max-times semiring, despite them supposedly being equivalent. The difference likely lies in the authors' non-standard implementation of max-times in log space.

Similarly, the Viterbi semiring achieves significantly higher accuracy than the max-times semiring despite the only difference being that the former is limited to $[0, 1]$ instead of $\mathbb{R}^+$. This implies that the weights converged to are relatively close to the initial ones, and that the initialisation of $\mathcal{N}(0.5, 0.5)$ is centered around a better minimum than $\mathcal{N}(2, 1)$.

This is also a plausible explanation for the probability semiring's similar performance, while its overall mid-level accuracy can be explained as earlier by (Schwartz et al., 2018).

Moving on to the top strata, we observe very close performance amongst the log-space max-times, the min-plus, and the max-plus semirings. One notable feature is that the convergence of log-space max-times is much more erratic when compared to the other two. This is likely a numerical stability issue, as the logarithm of small values (as we get with such initialisation) is relatively large in magnitude. One would expect that the min-plus and max-plus semirings would have identical performance, being so similar and with the same initialisation, but this is fully supported by these results; apparently learning from the least common sentences is more effective than learning from the most common ones.

As the others proposed, the log-space max-times semiring achieves the best accuracy. The reason why remains unclear, especially when one considers the worse performance of the max-times semiring. The discussed relations and conclusions hold when the models are tested on the set, as table 2 demonstrates.

4.3 Pattern experiments

4.3.1 Pattern length

We present the pattern length results in figure 4, which we will now analyse.

The trends shown here contradict our hypothesis (that larger patterns are not detrimental) to a large extent. We begin our analysis with the longest patterns (WFSAs) of those we considered. Lengths 8

Semiring	Val acc. (%)	Test acc. (%)
LogSpace	74.89	76.22
MinPlus	74.31	73.59
MaxPlus	73.49	73.20
Probability	71.44	70.84
Viterbi	70.30	69.85
MaxTimes	67.78	68.37
Boolean	52.87	49.49

Table 2: Test set results of various semirings

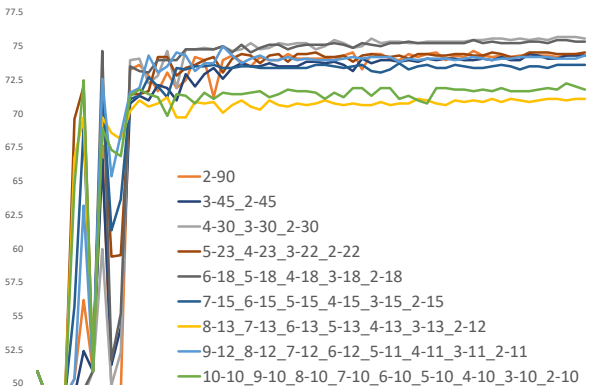

Figure 4: Validation accuracy per epoch, various pattern lengths

and 10 both performed remarkably poorly, which one could perhaps explain by the choice of hyperparameters or the structure of the dataset's sentences. Contradicting this however, length 9 performs well; its accuracy puts it solidly in the midfield.

To compound this confusion, we note that the *shortest* pattern performs better than all three of those lengths. In fact, it is arguably the third best performer across the entire set. Additionally, length 3 has a clear drop in accuracy compared to length 2.

In a similar fashion, lengths 4 and 6 perform markedly better than the others, but 5 and 7 are only of average performance.

There are no conclusions to be drawn from this experiment, except for perhaps that the top performing lengths are also the lengths of common sentences in the dataset. However, this ignores the possibility of epsilon transitions and so is an unsatisfactory answer. Perhaps the changing widths between pattern lengths explains this, though this logic dictates that the shortest lengths should perform the best.

4.3.2 Pattern width

We therefore move on to the next part of the experiment, where we alter the pattern widths. Indeed, these experiments do give some evidence that supports our hypothesis. Let us begin by analysing the performance of length 3 patterns in figure 5.

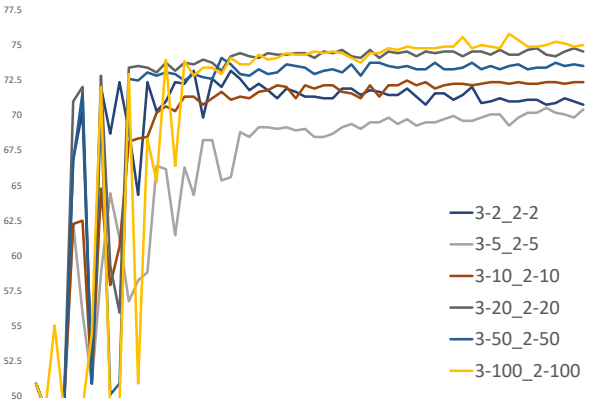

Figure 5: Validation accuracy per epoch, length 3, various pattern widths

We notice that increasing the width does in fact improve accuracy, albeit not entirely monotonically. The most accuracy pattern set is the widest, and the least accuracy is the second-thinnest. A factor of 20 separates the two widths. The accuracy gap is also significant: one is pushing 75%, while the other is struggling to reach 70%. However, as mentioned, the accuracy is not monotonic in width. Width 2 performs better than width 5, and width 20 outperforms width 50.

In any case, the positive trend of accuracy to width is still present. We move to an analysis of figure 6 to see of it holds for patterns of length 4.

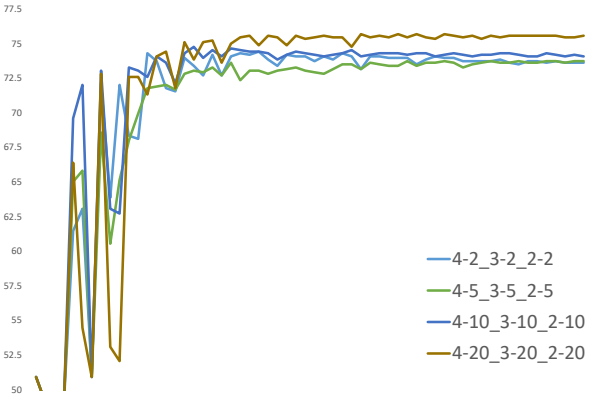

Figure 6: Validation accuracy per epoch, length 4, various pattern widths

Indeed it does. The trend is still clear, perhaps more so. The only disturbance in monotonicity is that width 2 sometimes has higher accuracy than width 5, though they converge after some time. One might have noticed that this graph is sparser than the previous one; this is because widths 50 and 100

were stuck with an accuracy of 49%. We believe this is an issue with hyperparameter tuning, as it is similar behavior to our models before we found the correct (low) learning rate.

Next we analyse length 5 in figure 7. The differences here are much clearer than in previous plots. Indeed, it shows clearly that increasing width improves accuracy: accuracy increases significantly each time width is increased. The exception to this is that width 20 outperforms width 50, though they seem to converge to similar accuracies near the final epochs. Again we note that width 100 failed to achieve an accuracy above 50%, though interestingly width 50 was fine.

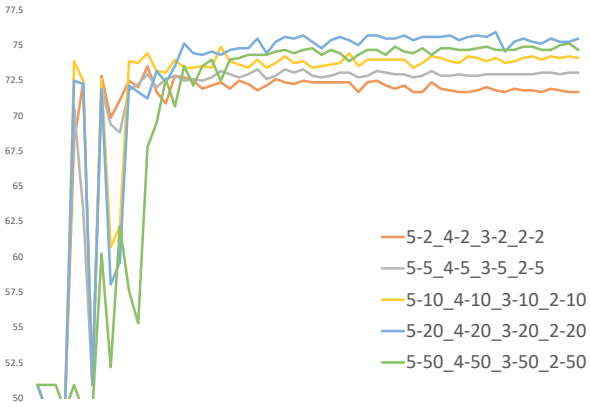

Figure 7: Validation accuracy per epoch, length 5, various pattern widths

These experiments on the width of patterns show that width has a noticeable impact effect on the accuracy of SoPa. The relationship is not monotone, but the rough trend is clearly present. Width could be likened to the capacity of a neural network, and maybe with this interpretation it makes sense that increasing width improves performance. We do not believe there is necessarily a language-theoretic explanation for this.

4.4 Diagonals experiments

Table 3 shows the results of diagonals experiments with and without ε-transitions. Through experiments, we observed (1) the minimum numbers of epochs that SoPa required to initialize learning process, (2) models performances when ε transitions are replaced by `{2, 3, 4, 5, 6}` steps forward transitions, (3) models performances considering both ε transitions and transitions with a large number of forward steps.

Optimal number of epochs In both experiments, the SoPa model starts to learn weights and improve the classification accuracy on the validation dataset after being trained for at least 10 epochs. Table 3 reveals that implemented models coverged after 20 to 30 epochs. We also observed that the model starts to learn after the 10^{th} epoch. In other words, the model yields 50% of validation accuracy to more than 60% after average 10 epochs.

Without ε-transitions The more connected the automata are, the better performance in test accuracy is. Indeed, table 3 shows an upward trend in test accuracy when replacing ε transitions with (> 1)-step forward transitions. In particular, models with `{1, 2}` steps forward transition produce test accuracies of 70.79% and 71.55%, while models with `{1, 2, 3, 4, 5, 6}` steps forward transitions yield better performances with 73.20% and 74.96%. Besides, given the same number of step forward transitions, we notice that permitting self-loops in each automaton notably reduces the models' performances. We set `max_steps_forward = 6` because all automata represented for soft patterns have 7 states as in our aforementioned setup. All in all, the experiment's results show that increasing the pattern length and implementing fully-connected automata without ε transitions, i.e. ε transitions are not used and the approximation in equation 4 is thus left out, potentially improves the model performance.

With ε-transitions With the SoPa model permitting both ε-transitions and more connected automata, adding a self-loop to the transition matrix also reduces the test accuracy of the model, similar to the previous experiment. In fact, adding the self-loop diagonal to the model makes the test accuracy drop from 75.84% to 67.88% when automata have `{1, 2}` step forward transitions and decrease from 71.99% to 69.41% when automata have `{1, 2, 3, 4, 5, 6}` step forward transitions. Moreover, it is noticeable that the model which added `{1, 2}` step forward transitions diagonals, removed self-loop diagonal, and retained the ε-transitions, did slightly outperform the baseline model which considers only one step forward transitions. Overall, the model, which has two diagonals of `{1, 2}` steps forward transitions and removes the self-loop diagonal, is the only model that outperformed the baseline. Also, in case of remaining the parameter represented for ε-transitions, removing self-loop transitions and increasing the number of steps forward transition in automata sig-

Max-forward steps	Diagonals	Self-loop	ε-transitions	Best epoch	Val. acc. (%)	Test acc. (%)
1	2	yes	yes	36	74.80	75.29
2	3	yes	**no**	55	70.80	70.79
2	2	no	**no**	20	71.60	71.55
6	7	yes	**no**	38	72.82	73.20
6	6	no	**no**	21	72.59	74.96
2	4	yes	yes	34	72.40	67.88
2	3	no	yes	34	74.40	75.84
6	8	yes	yes	20	72.82	69.41
6	7	no	yes	59	74.77	71.99

Table 3: Evaluation of diagonal experiments. Diagonals column refers to the number of diagonals in **T(x)**, Best epoch refers to the epoch that model results the best performance in all 250 epochs.

nificantly enhance the classification accuracy.

4.5 Shared_SL experiments

When the number of diagonals is two, there is a downward trend in test accuracy as we increase the value of `shared_sl`, as shown in table 4. We ascribe this to the decreasing complexity of the model. As explained in section 3.6, when `shared_sl` is 0, the model keeps a real-valued vector as a parameter which is as large as the word embedding. If `shared_sl` is 1, we drop this vector and a scalar parameter for each self-loop is used. If `shared_sl` is 2, we drop those constants and keep only one scalar for all the loops in the WFSA. Models that use more parameters to model self-loop transitions seem to compute more accurate scores for documents and hence yield better classification accuracies.

On the other hand, we do not see a clear pattern when the diagonals are three. The test accuracy when `shared_sl` is 1 is particularly interesting, because it is even lower than the test accuracy of the model with `shared_sl` $= 2$, which is a special case of the one with `shared_sl` $= 1$. We cannot provide a reasonable explanation for this and we leave further exploration of the model in this regime to future work.

We also notice that, for both numbers of diagonals, the duration of the training is significantly lower when `shared` is 2. This might be attributed to the fact that the model is learning fewer weights in this case.

4.6 Batch size experiments

In the experiments with models that use only 2 diagonals in the transition matrix, shown in table 5, it does not seem to be a clear relation between batch size and accuracy both in the validation and in the test set. However, using a large batch size of 100 has shown give good accuracy and is indeed the one used for training in the paper.

In the case with 3 diagonals, it seems that increasing the batch size improves accuracy in the test set. This trend is there also for the validation accuracy, except from the value when the batch size is 4.

As mentioned in section 3.7, the better overall performance can be motivated with the fact that training the patterns on multiple documents improves gradient computation and hence the convergence of the WFSA weights to the optimum.

5 Conclusion

This paper presented our findings in the ablation study on SoPa model. We explore the model at various depths, including automaton level (pattern experiments, diagonals experiments, self-loop weight sharing experiments), automata level (semiring experiments), and SoPa model level (batch size experiments). Through our experiments, we found several valuable insights. First, SoPa model is found to be especially sensitive to the learning rate. Second, increasing the width of patterns improves model performance while the length imposes little effect. Regarding the diagonals experiments, the bigger the max-steps-forward, the better performance the model yields in cases excluding ϵ-transitions. By permitting both ϵ-transitions and increasing max-steps-forward, the model results a worst test accuracy when a self-loop is added to the transition matrix. Besides, there is a downward trend in test accuracy as we increase the value of `shared_sl` because of the complexity decreasing of the model. The duration of the training is significantly lower

shared_sl	Diagonals	Best epoch	Val. acc. (%)	Test acc. (%)
0	2	36	74.8	75.29
1	2	43	73.28	64.58
2	2	20	73.17	64.47
0	3	34	72.4	67.88
1	3	34	73.9	59.36
2	3	11	71.9	73.20

Table 4: Various shared loop configurations

Batch size	Diagonals	Best epoch	Val. acc. (%)	Test acc. (%)
1	2	36	74.89	75.29
2	2	23	74.54	71.6
4	2	48	75.57	75.29
8	2	44	74.77	74.41
100	2	148	75.2	76.33
1	3	36	72.4	67.88
2	3	43	72.7	71.55
4	3	20	74.6	72.16
8	3	62	72.9	72.27

Table 5: Batch size experiment

when `shared_sl = 2` because of the decrease in the number of learning weights. Third, the log-space max-times semiring achieves the best accuracy compared to a variety of other semirings, including max-times. Last but not least, training the patterns on multiple documents improves the gradient computation and hence the convergence of the WFSA weights to the optimum.

In the future, it is possible to expand the experiment by implementing models with a full dataset and adding a more extensive pattern set. On the other hand, increasing the width and length of patterns and investigating experiments with other semiring are also potential approaches to improve the model's performance.

Reproducibility

The setup of our experiments has been extensively documented in their appropriate sections. Please find our implementation at this Github Repos.

Acknowledgement

The authors thank prof. Ryan Cotterell for the project proposal feedback.

References

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.

Dmitry Davidov, Oren Tsur, and Ari Rappoport. 2010. Enhanced sentiment learning using Twitter hashtags and smileys. In *Coling 2010: Posters*, pages 241–249, Beijing, China. Coling 2010 Organizing Committee.

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.

C. L. Giles, C. B. Miller, D. Chen, H. H. Chen, G. Z. Sun, and Y. C. Lee. 1992. Learning and Extracting Finite State Automata with Second-Order Recurrent Neural Networks. *Neural Computation*, 4(3):393–405.

Liang Huang. 2008. Advanced dynamic programming in semiring and hypergraph frameworks. In *Coling 2008: Advanced Dynamic Programming in Computational Linguistics: Theory, Algorithms and Applications - Tutorial notes*, pages 1–18, Manchester, UK. Coling 2008 Organizing Committee.

Dekang Lin, Shaojun Zhao, Lijuan Qin, and Ming Zhou. 2003. Identifying synonyms among distributionally

similar words. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, IJCAI'03, page 1492–1493, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014a. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014b. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

F. Rosenblatt. 1958. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408.

Roy Schwartz, Roi Reichart, and Ari Rappoport. 2015. Symmetric pattern based word embeddings for improved word similarity prediction. In *Proceedings of the Nineteenth Conference on Computational Natural Language Learning*, pages 258–267, Beijing, China. Association for Computational Linguistics.

Roy Schwartz, Sam Thomson, and Noah A. Smith. 2018. Sopa: Bridging cnns, rnns, and weighted finite-state machines. *CoRR*, abs/1805.06061.

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 *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA. Association for Computational Linguistics.