UNIVERSITY OF POTSDAM

MASTER'S THESIS

SoPa++: Leveraging explainability from hybridized RNN, CNN and weighted finite-state neural architectures

Author:
Atreya SHANKAR

1st Supervisor:
Dr. Sharid LOÁICIGA
University of Potsdam

2nd Supervisor:
Mathias MÜLLER
University of Zurich

A thesis submitted in fulfillment of the requirements for the degree of Cognitive Systems: Language, Learning, and Reasoning (M.Sc.)

in the

Foundations of Computational Linguistics Research Group
Department of Linguistics

April 1, 2021

Contents

Chapter 1

Introduction

1.1 Motivation

With the recent trend of increasingly large deep learning models achieving State-Of-The-Art (SOTA) performance on a myriad of Machine Learning (ML) tasks (Figure 1), several studies argue for focused research into Explainable Artificial Intelligence (XAI) to address emerging concerns such as security risks and inductive biases associated with black-box models (Doran, Schulz, and Besold, 2017; Townsend, Chaton, and Monteiro, 2019; Danilevsky et al., 2020; Arrieta et al., 2020). Of these studies, Arrieta et al. (2020, Page 4, Section 2.2) provide the following novel definition of XAI based on an extensive literature review of recent XAI research:

> *"Given an audience, an **explainable** Artificial Intelligence is one that produces details or reasons to make its functioning clear or easy to understand."*

In addition, Arrieta et al. (2020) explore and classify a variety of machine-learning models into transparent and black-box categories depending on their degrees of transparency. Furthermore, they explore taxonomies of post-hoc explainability methods aimed at effectively explaining black-box models. Of high relevance to this study are the local explanations, feature relevance and explanations by simplification post-hoc explainability techniques.

Through a survey of recent literature on explanations by simplification applied in the Natural Language Processing (NLP) field, we came across several prominent studies employing techniques to simplify black-box neural networks into constituent Finite-State Automata (FAs) and/or Weighted Finite-State Automata (WFAs) (Schwartz, Thomson, and Smith, 2018; Peng et al., 2018; Suresh et al., 2019; Wang and Niepert, 2019; Jiang et al., 2020).

In this thesis, we build upon the work of Schwartz, Thomson, and Smith (2018) by further developing their **So**ft **Pa**tterns (SoPa) model; which represents a hybridized RNN, CNN and Weighted Finite-State Automaton neural network architecture. We modify the SoPa model by changing key aspects of its architecture which ultimately allows us to conduct effective explanations by simplification; which was not possible with the previous SoPa architecture. We abbreviate this modified model as **SoPa++**, which signifies an improvement or major modification to the SoPa model. Finally, we evaluate both the performance and explainability of the SoPa++ model on the Facebook Multilingual Task Oriented Dialog data set (FMTOD; Schuster et al. 2019); focusing on the English-language intent classification task.

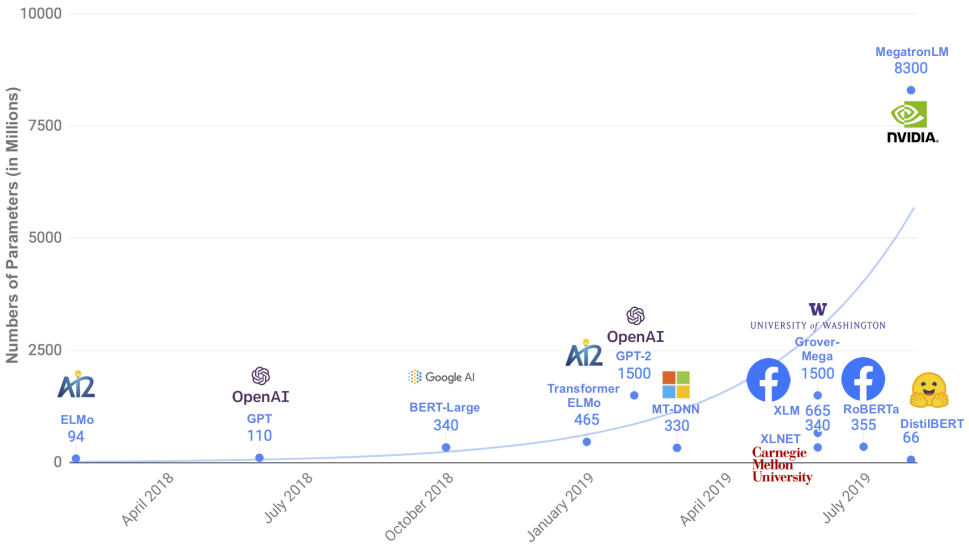

FIGURE 1: Parameter counts of recently released pre-trained language models which showed competitive or SOTA performance when fine-tuned over a range of NLP tasks; figure taken from Sanh et al. (2019)

1.2 Research questions

With the aforementioned modifications to the SoPa architecture and the introduction of the SoPa++ architecture, we aim to answer the following three research questions:

1. To what extent does SoPa++ contribute to competitive performance[1] on the FMTOD data set?
2. To what extent does SoPa++ contribute to effective explanations by simplification on the FMTOD data set?
3. What interesting and relevant explanations can SoPa++ provide on the FMTOD data set?

1.3 Thesis structure

With the aforementioned research questions, we summarize the structure and contents of this thesis.

Chapter 1: Introduce this thesis, its contents and our research questions.

Chapter 2: Describe the background concepts utilized in this thesis.

Chapter 3: Describe the methodologies pursued in this thesis.

Chapter 4: Describe the results obtained from our methodologies.

Chapter 5: Discuss the implications of the aforementioned results.

Chapter 6: Conclude this thesis by answering the research questions.

Chapter 7: Document future work to expand on our research questions.

[1] We define competitive performance as the scenario where a mean performance metric on a certain data set falls within the range obtained from other recent studies on the same data set

Chapter 2

Background concepts

2.1 Explainable artificial intelligence

In this section, we lay out background concepts for Explainable Artificial Intelligence (XAI) which have been largely adopted from Arrieta et al. (2020). The study is particularly helpful for us since it summarizes the findings of approximately 400 XAI contributions and presents these findings in the form of well-defined concepts and taxonomies. In addition, the study discusses the many possible future directions of XAI research.

2.1.1 Transparency

One of the key contributions of Arrieta et al. (2020) to the field of XAI is the introduction of the concept of transparency, as well as the classification of various Machine Learning (ML) models into transparent and black-box categories. In this section, we provide an adapted definition of transparency and list examples of transparent and black-box ML models.

Definition 1 (Transparency; Arrieta et al. 2020)**.** A model is considered to be transparent if it is understandable on its own without any external techniques to assist its understandability. Since a model can provide different degrees of understandability, transparent models are split into three possibly mutually-inclusive categories; specifically simulatable models, decomposable models and algorithmically transparent models.

Remark 1.1. *Simulatability* refers to the ability of a model's inner-mechanisms being simulated strictly by a human. Therefore, model simplicity plays a major role in this class.

Remark 1.2. *Decomposability* refers to the ability to clearly understand the individual parts of a model; such as its inputs, parameters and basic computational mechanisms.

Remark 1.3. *Algorithmic transparency* refers to the ability of a human to understand the algorithmic processes used in the model to produce any given output from any given input.

Remark 1.4. A model is considered transparent if it falls into one or more of the aforementioned transparency categories. If a model cannot satisfy any of the requirements of being transparent, then it is classified as a *black-box* model.

Remark 1.5. As recommended by Arrieta et al. (2020, Page 3, Section 2.1), we use the terms transparency and interpretability to refer to the same feature. Therefore, we use these terms equivalently and interchangeably.

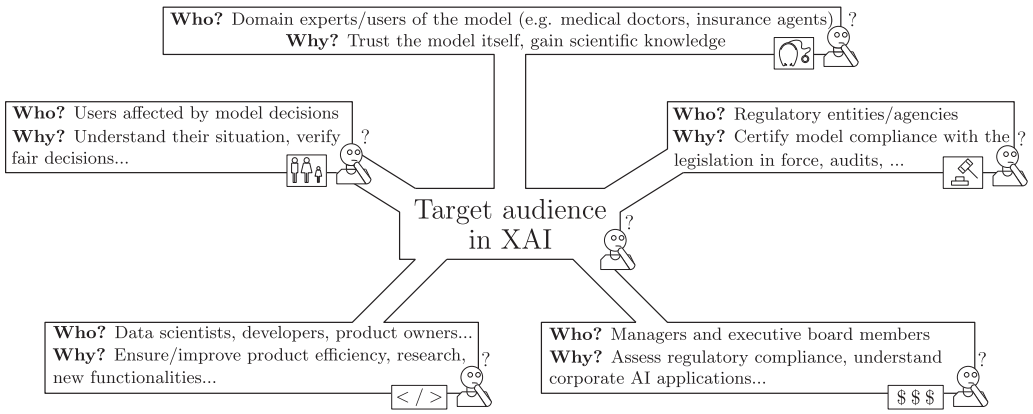

FIGURE 2: Examples of various target audiences in XAI; figure taken from Arrieta et al. (2020)

Examples of well-known transparent ML models are linear/logistic regressors, decision trees and rules-based learners. Similarly, common examples of black-box ML models are tree ensembles and deep neural networks. Arrieta et al. (2020) provide extensive justifications using the aforementioned three criteria in conducting model classifications into the transparent and black-box categories. We would direct the reader to their study for a full analysis and justification of these classifications.

2.1.2 Explainability and XAI

Another key contribution of Arrieta et al. (2020) is a unified conceptualization of explainability and XAI based on an extensive literature review. Drawing inspiration from their study, we provide adapted definitions of explainability and XAI, as well as comment on interesting aspects related to XAI.

Definition 2 (Explainability; Arrieta et al. 2020)**.** Explainability refers to the interface between humans and a model, such that this interface is both an accurate proxy of the model and comprehensible to humans.

Definition 3 (Explainable Artificial Intelligence; Arrieta et al. 2020)**.** An explainable artificial intelligence is one that produces details to make its functioning understandable to a given target audience.

Arrieta et al. (2020) observe that black-box ML models are increasingly being employed to make important predictions in critical contexts, citing high-risk areas such as precision medicine and autonomous vehicles. Of particular relevance to the field of Natural Language Processing (NLP), the study notes a myriad of issues related to inductive biases within training data sets and the ethical issues involved in using black-box models trained on such data sets. As a result, they describe an increased demand for transparency in black-box ML models from the various stakeholders in Artificial Intelligence (AI). In addition, Arrieta et al. (2020) emphasize the presence of a target audience for XAI; implying that different XAI techniques should be employed for different target audiences. In their study, they provide examples of target audiences such as domain experts, end-users and managers (Figure 2).

2.1.3 Terminology clarification

Based on a review of roughly 400 XAI studies, Arrieta et al. (2020) observe that many studies tend to misuse the terms interpretability and explainability by using them interchangeably. To address this, they provide clear conceptual differences between the terms. For example, Arrieta et al. (2020, Page 3, Section 2.1) state that *"interpretability refers to a passive characteristic of a model referring to the level at which a given model makes sense for a human observer."* In contrast, Arrieta et al. (2020, Page 3, Section 2.1) state that *"explainability can be viewed as an active characteristic of a model, denoting any action or procedure taken by a model with the intent of clarifying or detailing its internal functions."*

In summary, we gather that interpretability (or transparency as per Remark 1.5) refers to an inherent or passive feature of a model. On the other hand, explainability refers to an active characteristic undertaken by the model and its developers to explain the model's inner mechanisms. In addition, explainability entails the presence of a target audience; which may not necessarily be the case for interpretability or transparency.

2.1.4 Explainability techniques

Based on the aforementioned classification of ML models into transparent and black-box models, Arrieta et al. (2020) expound on explainability techniques for each of these model types. Due to their transparent nature, the study states that transparent ML models are usually explainable in themselves to most target audiences and therefore usually do not require any external technique to extract explanations. The study does however highlight some target audiences, such as non-expert users, who may require external explainability techniques such as model output visualizations in order to explain the inner workings of transparent ML models.

For the case of black-box models, Arrieta et al. (2020) argue that separate or external techniques must be utilized in order to reasonably explain these models. Such explainability techniques are referred to in the study as post-hoc explainability techniques; which is derived from the idea that explanations for such models are usually extracted post-modeling. Notable examples of post-hoc explainability techniques include local explanations, feature relevance and explanations by simplification; for which we provide adapted definitions below.

Definition 4 (Local explanations; Arrieta et al. 2020)**.** Local explanations operate by segmenting a model's solution space into subspaces and provide explanations for the less complex model subspaces.

Remark 4.1. Local explanations are commonly used in XAI research and function by using differentiating properties on model solution space subsets.

Remark 4.2. Two well-known examples of local explainability techniques are Local Interpretable Model-Agnostic Explanations (LIME; Ribeiro, Singh, and Guestrin 2016) and G-REX (Konig, Johansson, and Niklasson, 2008).

Definition 5 (Feature relevance; Arrieta et al. 2020)**.** Feature relevance explanation methods operate by computing an importance score for the model's input variables over the model's output variables. These scores typically quantify how sensitive the model's output is to perturbations in the model's inputs.

Remark 5.1. Feature relevance methods can be considered as indirect methods of explaining a model.

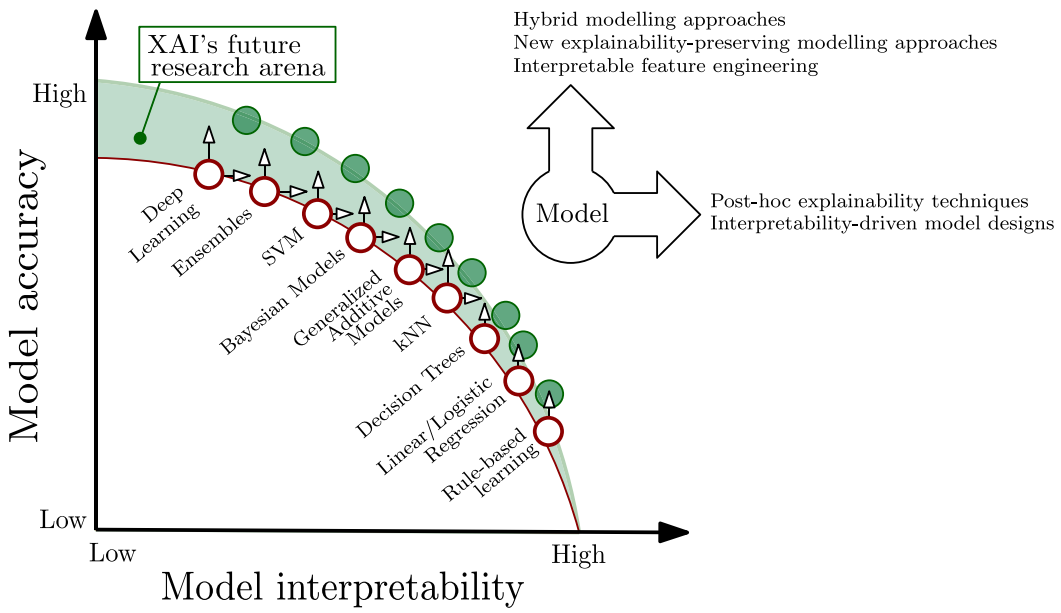

FIGURE 3: Schematic visualizing the performance-interpretability tradeoff; figure taken from Arrieta et al. (2020)

Remark 5.2. Well-known feature relevance explainability techniques include the Shapley Additive Explanations (SHAP; Lundberg and Lee 2017) and the occlusion sensitivity method (Zeiler and Fergus, 2014).

Definition 6 (Explanations by simplification; Arrieta et al. 2020)**.** Explanations by simplification refer to techniques where a simplified proxy model is built to approximate and explain a more complex model. The simplified proxy model usually has to fulfil the the joint criteria of reducing its complexity compared to its antecedent model while maximizing its resemblance to its antecedent and keeping a similar performance score.

Remark 6.1. In this thesis, we refer to the original black-box model as an *antecedent* model and the simplified model as the *proxy* model. Furthermore, we qualify that all proxy models must be designed to globally approximate their respective antecedent models.

Remark 6.2. Bastani, Kim, and Bastani (2017) and Tan et al. (2018) are examples of studies that extract and distill simpler proxy models from complex antecedent models.

Through a survey of recent literature on explanations by simplification applied in the Natural Language Processing (NLP) field, we came across several prominent studies employing explanations by simplification to simplify antecedent black-box neural networks into proxy Finite-State Automata (FAs) and/or Weighted Finite-State Automata (WFAs) (Schwartz, Thomson, and Smith, 2018; Peng et al., 2018; Suresh et al., 2019; Wang and Niepert, 2019; Jiang et al., 2020). We expound more on WFAs and Schwartz, Thomson, and Smith (2018) in Sections 2.3 and 2.4 respectively.

2.1.5 Performance-interpretability tradeoff

An interesting and insightful contribution of Arrieta et al. (2020, Page 18, Section 5.1) is the description of the performance-interpretability tradeoff; which they introduce

with the caveat that *"the matter of interpretability versus performance is one that repeats itself through time, but as any other big statement, has its surroundings filled with myths and misconceptions."* To address some of the aforementioned myths and misconceptions, Arrieta et al. (2020) first disprove the generic statement that more complex models are always more accurate by pointing to certain case studies to support this argument. In particular, they show that complex models are not necessarily more accurate in cases where the function to be modeled is not complex, data is well-structured and input features are available with high quality.

Next, Arrieta et al. (2020) provide the case where the aforementioned statement, that complex models perform better, tends to be true. According to the study, this is usually true when the function to be modeled is sufficiently complex and where the input data has high diversity or variance; and possibly contains significant noise. In such cases, Arrieta et al. (2020) argue that the performance-interpretability tradeoff can be observed; as exemplified in Figure 3.

2.1.6 Explainability metrics

Towards the end of their study, Arrieta et al. (2020) note two major limitations of the current state of XAI research. Firstly, they observe the lack of a unified conceptualization of explainability between various studies. They furthermore acknowledge that their study, while limited, could provide a good starting point for other XAI studies. Secondly, Arrieta et al. (2020) note the lack of a unified metric that denotes how explainable any given model is. They further explain why developing such a metric has been a difficult process for many XAI studies; particularly because such a metric would entail incorporating psychological, sociological and cognitive elements to accommodate the goodness of fit of an explainability method to a certain target audience. Furthermore, incorporating such elements might involve significant amounts of subjectivity in the desired metric.

To reduce some of the aforementioned subjectivity involved, Miller (2019) and Arrieta et al. (2020) provide basic guidelines of what could constitute a good explanation based on human psychology, sociology and cognitive sciences. Firstly, they observe that explanations are better when *constrictive*; meaning an explanation is good if it not only explains why a model made decision X, but also why it made decision X over decision Y. Next, they suggest that good explanations should be able to communicate causal links over probabilities; which could be a challenge for black-box models which generally compute aggregate probabilities without necessarily considering causal links. Finally, they recommend that explanations are better when *selective*; meaning that a good explanation should be able to selectively provide the most important causal links instead of all possible causal links as these might be irrelevant or confusing to the target audience.

2.2 Straight-through estimator

Activation quantized neural networks refer to neural networks that contain layers which transmit piece-wise discrete signals and have been an object of active research in regards to low-precision and low-resource computing. One of the main challenges in training such activation quantized neural networks is that their gradients tend to vanish almost everywhere because their derivatives typically default to zero (Bengio, Léonard, and Courville, 2013; Courbariaux and Bengio, 2016; Yin et al., 2019). One significant workaround for this issue was proposed by Bengio, Léonard, and

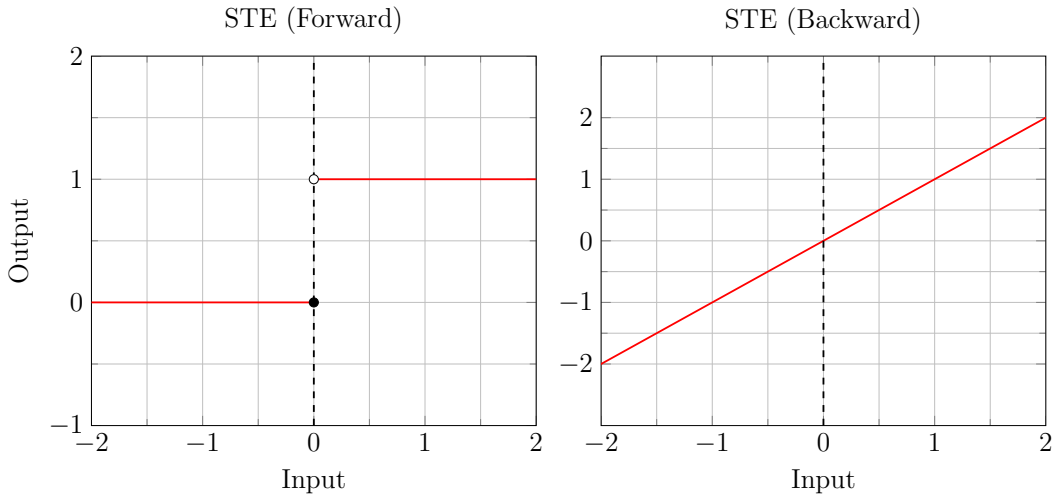

FIGURE 4: Schematic visualizing the vanilla STE's forward and backward passes

Courville (2013) through the introduction of the Straight-Through Estimator (STE). In the vanilla version of the STE, the STE neuron emits a signal of 1 when its input is strictly positive, and emits a zero signal in all other cases (Equation 1). The STE then uses a simple identity function to estimate the gradient during the backward pass (Equation 2). The vanilla STE is visualized through a schematic in Figure 4.

$$\text{STE}(x) = \begin{cases} 1 & x \in (0, +\infty) \\ 0 & x \in (-\infty, 0] \end{cases} \tag{1}$$

$$\text{STE}'(x) = x \tag{2}$$

Aside from the vanilla STE, several other flavors of STEs have been proposed by studies such as Courbariaux and Bengio (2016) and Yin et al. (2019). In general, these studies have shown that activation quantized neural networks perform competitively with their non-quantized counterparts; while providing performance-related benefits related to lower precision computing as well as enabling further research into threshold driven neural activation functions.

2.3 Finite-state automata

As mentioned in Section 2.1.4, we found multiple studies in NLP that conducted explanations by simplification on black-box neural networks by simplifying them into FAs and WFAs. As this will eventually be a focus of this thesis, we define key concepts related to the FAs and WFAs in this section.

2.3.1 Finite-state automaton

Finite-state automata is a collective term for two sub-categories of models; namely deterministic and nondeterministic finite-state automata. Here we provide definitions and descriptions for these mutually exclusive model categories.

Definition 7 (Deterministic finite-state automaton; Sipser 1996)**.** A deterministic finite-state automaton is a 5-tuple $\mathcal{M} = \langle \Sigma, Q, \delta, q_0, F \rangle$, with:

– a finite input alphabet Σ;

- a finite state set $\mathcal{Q}$;
- a transition function $\delta : \mathcal{Q} \times \Sigma \to \mathcal{Q}$;
- an initial state $q_0 \in \mathcal{Q}$;
- and a set of final or accepting states $F \subseteq Q$.

Definition 8 (Nondeterministic finite-state automaton; Sipser 1996)**.** A nondeterministic finite-state automaton is a 5-tuple $\mathcal{M} = \langle \Sigma, \mathcal{Q}, \delta, q_0, F \rangle$, with:

- a finite input alphabet Σ;
- a finite state set $\mathcal{Q}$;
- a transition function $\delta : \mathcal{Q} \times \{\Sigma \cup \{\epsilon\}\} \to \mathcal{P}(\mathcal{Q})$;
- an initial state $q_0 \in \mathcal{Q}$;
- and a set of final or accepting states $F \subseteq Q$.

Remark 8.1. $\epsilon \notin \Sigma$ refers to an empty-string transition and results in a change of state without consuming any input symbol. These transitions are unique to nondeterministic finite-state automata.

Remark 8.2. Self-loop transitions refer to special transitions which consume an input token while staying at the same state. These are allowed in both deterministic and nondeterministic finite-state automata.

Remark 8.3. $\mathcal{P}(\mathcal{Q})$ refers to the power set of $\mathcal{Q}$, or otherwise the collection of all subsets of $\mathcal{Q}$.

Definition 9 (Linear-chain finite-state automaton; Schwartz, Thomson, and Smith 2018)**.** Any finite-state automaton is considered linear-chain if there exists only one set of consecutive transition states leading from the start to the accepting state. Linear-chain finite-state automata furthermore possess only one start and end state and only allow transitions to the same or next state which is closer to the accepting state.

Remark 9.1. A linear-chain finite-state automaton can be considered as *strict* if it does not permit self-loop transitions and therefore only permits transitions to adjacent states which are strictly closer to the accepting state. An example of a strict linear-chain NFA can be seen in Figure 5.

Remark 9.2. The presence of the linear-chain terminology is effectively absent in theoretical computer science literature but is prevalent in practical research related to Hidden Markov Models (HMMs) and Conditional Random Fields (CRFs) such as in Tsuruoka, Tsujii, and Ananiadou (2009).

A string $\boldsymbol{x}$ is said to be *accepted* by any FA $\mathcal{M}$ if the current state after consuming $\boldsymbol{x}$ is the accepting state. Similarly, a string $\boldsymbol{x}$ is said to be *rejected* by any FA $\mathcal{M}$ if the string $\boldsymbol{x}$ cannot be consumed or the current state after consuming $\boldsymbol{x}$ is a non-accepting state. The key difference between DFAs and NFAs is present in their transition functions; specifically that the former guarantees that each state allows for a unique transition to another state given a non-empty input string. Contrastingly, NFAs allow for arbitrary transitions without necessarily consuming an input symbol.

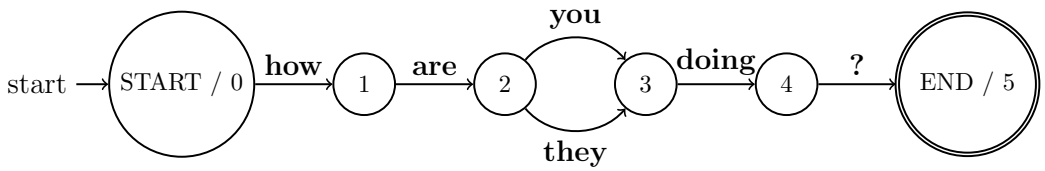

FIGURE 5: Regular expression ‘`how are (you|they) doing ?`’ converted into a linear-chain NFA; double circles on state 5 indicate that this is an accepting state

2.3.2 Regular expressions

Regular expressions (REs) and FAs have been shown to be equivalent in their expressive power in that they both recognize regular languages (Sipser, 1996). Several algorithms have been historically studied and optimized for converting FAs to REs and vice-versa (McNaughton and Yamada, 1960; Thompson, 1968). Figure 5 shows a simple example of the regular expression ‘`how are (you|they) doing ?`’ converted into a strict linear-chain NFA. The aforementioned regular expression is written using the Perl-compatible regular expression syntax with the ‘`|`’ character indicating alternative possibilities.

2.3.3 Weighted finite-state automaton

Weighted finite-state automata (WFAs) are *soft* extensions of finite-state automata which allow for the assignment of numerical weights to transitions; given an algebraic semiring to govern this numerical projection. Compared to the aforementioned FAs that only accept or reject strings, WFAs numerically score strings which has made them useful for several applications in NLP such as tokenization and part-of-speech tagging (Maletti, 2017).

Definition 10 (Semiring; Kuich and Salomaa 1986)**.** A semiring is a set $\mathbb{K}$ along with two binary associative operations $\oplus$ (addition) and $\otimes$ (multiplication) and two identity elements: $\bar{0}$ for addition and $\bar{1}$ for multiplication. Semirings require that addition is commutative, multiplication distributes over addition, and that multiplication by $\bar{0}$ annihilates, i.e., $\bar{0} \otimes a = a \otimes \bar{0} = \bar{0}$.

Remark 10.1. Semirings follow the following generic notation: $\langle \mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1} \rangle$.

Remark 10.2. A simple and common semiring is the real or sum-product semiring: $\langle \mathbb{R}, +, \times, 0, 1 \rangle$. Two important semirings for this thesis are shown below.

Remark 10.3. **Max-sum** semiring: $\langle \mathbb{R} \cup \{-\infty\}, \max, +, -\infty, 0 \rangle$

Remark 10.4. **Max-product** semiring: $\langle \mathbb{R}_{>0} \cup \{-\infty\}, \max, \times, -\infty, 1 \rangle$

Definition 11 (Weighted finite-state automaton; Peng et al. 2018)**.** A weighted finite-state automaton over a semiring $\mathbb{K}$ is a 5-tuple $\mathcal{A} = \langle \Sigma, \mathcal{Q}, \boldsymbol{\Gamma}, \boldsymbol{\lambda}, \boldsymbol{\rho} \rangle$, with:

- a finite input alphabet Σ;
- a finite state set $\mathcal{Q}$;
- transition matrix $\boldsymbol{\Gamma} : \mathcal{Q} \times \mathcal{Q} \times (\Sigma \cup \{\epsilon\}) \to \mathbb{K}$;
- initial vector $\boldsymbol{\lambda} : \mathcal{Q} \to \mathbb{K}$;
- and final vector $\boldsymbol{\rho} : \mathcal{Q} \to \mathbb{K}$.

Remark 11.1. Σ^* refers to the (possibly infinite) set of all strings over the alphabet Σ, where * represents the Kleene star operator.

Remark 11.2. As with finite-state automata, it is also possible to construct linear-chain weighted finite-state automata.

Definition 12 (Path score; Peng et al. 2018)**.** Let $\boldsymbol{\pi} = \langle \pi_1, \pi_2, \ldots, \pi_n \rangle$ be a sequence of adjacent transitions in $\mathcal{A}$, with each $\pi_i = \langle q_i, q_{i+1}, z_i \rangle \in \mathcal{Q} \times \mathcal{Q} \times (\Sigma \cup \{\epsilon\})$. The path $\boldsymbol{\pi}$ derives the ϵ-free string $\boldsymbol{x} = \langle x_1, x_2, \ldots, x_m \rangle \in \Sigma^*$; which is a substring of the ϵ-containing string $\boldsymbol{z} = \langle z_1, z_2, \ldots, z_n \rangle \in (\Sigma \cup \{\epsilon\})^*$. $\boldsymbol{\pi}$'s score in $\mathcal{A}$ is given by:

$$\mathcal{A}[\boldsymbol{\pi}] = \boldsymbol{\lambda}(q_1) \otimes \left(\bigotimes_{i=1}^{n} \boldsymbol{\Gamma}(\pi_i) \right) \otimes \boldsymbol{\rho}(q_{n+1}) \tag{3}$$

Definition 13 (String score; Peng et al. 2018)**.** Let $\Pi(\boldsymbol{x})$ denote the set of all paths in $\mathcal{A}$ that derive $\boldsymbol{x}$. Then the string score assigned by $\mathcal{A}$ to string $\boldsymbol{x}$ is given by:

$$\mathcal{A}[\![\boldsymbol{x}]\!] = \bigoplus_{\boldsymbol{\pi} \in \Pi(\boldsymbol{x})} \mathcal{A}[\boldsymbol{\pi}] \tag{4}$$

Remark 13.1. Since $\mathbb{K}$ is a semiring, $\mathcal{A}[\![\boldsymbol{x}]\!]$ can be efficiently computed using the Forward algorithm (Baum and Petrie, 1966). Its dynamic program is summarized below without ϵ-transitions for simplicity. $\Omega_i(q)$ gives the aggregate score of all paths that derive the substring $\langle x_1, x_2, \ldots, x_i \rangle$ and end in state q:

$$\Omega_0(q) = \boldsymbol{\lambda}(q) \tag{5a}$$

$$\Omega_{i+1}(q) = \bigoplus_{q' \in \mathcal{Q}} \Omega_i(q') \otimes \boldsymbol{\Gamma}(q', q, x_i) \tag{5b}$$

$$\mathcal{A}[\![\boldsymbol{x}]\!] = \bigoplus_{q \in \mathcal{Q}} \Omega_n(q) \otimes \boldsymbol{\rho}(q) \tag{5c}$$

Remark 13.2. The Forward algorithm can be generalized to any semiring (Eisner, 2002) and has a runtime of $O(|Q|^3 + |Q|^2|\boldsymbol{x}|)$ (Schwartz, Thomson, and Smith, 2018); notably with a linear runtime with respect to the length of the input string $\boldsymbol{x}$.

Remark 13.3. A special case of Forward is the Viterbi algorithm, where the addition $\oplus$ operator is constrained to the maximum function (Viterbi, 1967). Viterbi therefore returns the highest scoring path $\boldsymbol{\pi}$ that derives the input string $\boldsymbol{x}$.

2.4 SoPa

Schwartz, Thomson, and Smith (2018) present a novel hybridized RNN, CNN and WFA-based neural architecture called **So**ft **Pa**tterns (SoPa). This architecture resembles a RNN because it processes text sequentially and can encode strings of arbitrary lengths. Similarly, the architecture contains a variable number of constrained linear-chain WFAs with variable pattern lengths or window sizes; which resembles both one-layer CNNs and an ensemble of linear-chain WFAs. The combination of the aforementioned neural features allows SoPa to learn soft versions of traditional textual surface patterns.

In their study, Schwartz, Thomson, and Smith (2018) test the SoPa architecture on three separate sentiment classification tasks and compare the results with four baselines which included a bidirectional LSTM and a CNN. With their results, they show that SoPa performed on par or better than all baselines on all tasks. Additionally,

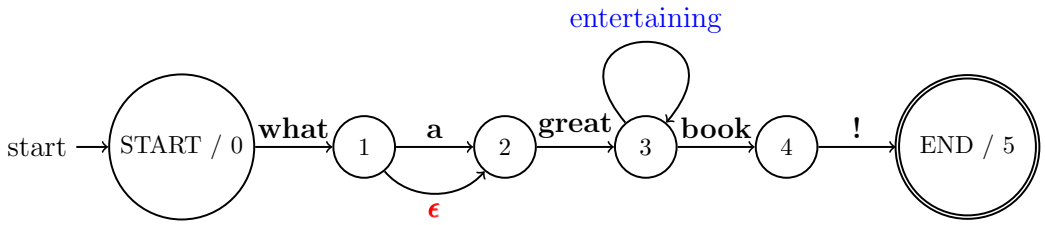

FIGURE 6: Schematic visualizing a linear-chain NFA with self-loop (blue), ϵ (red) and main-path (black) transitions; figure adapted from Schwartz, Thomson, and Smith (2018)

they show that SoPa outperformed all baselines significantly in low-data settings. Finally, the authors present a simple method to explain the SoPa model using both local explanations and feature relevance.

2.4.1 Linear-chain WFA

In regards to the WFAs used in SoPa, Schwartz, Thomson, and Smith (2018) utilize linear-chain WFAs where each linear-chain WFA is alloted a sequence of $|\mathcal{Q}|$ states. Each state i in the linear-chain WFA has three possible outgoing transitions; namely a **self-loop transition** which consumes a token but stays in the same state i, an **ϵ-transition** which does not consume a token but transitions to state $i+1$ and a **main-path transition** which consumes a specific token and transitions to state $i+1$. Furthermore, Schwartz, Thomson, and Smith (2018) utilize only the max-sum and max-product semirings in their linear-chain WFAs. Figure 6 shows a sample NFA extracted from the linear-chain WFA with all three aforementioned transitions.

To more concretely express these transitions, Schwartz, Thomson, and Smith (2018) provide the following formulation of the transition matrix $\mathbf{\Gamma}$ under the linear-chain structure. Here, $\mathbf{\Gamma}(x)$ represents a $|Q| \times |Q|$ matrix containing transition scores when consuming an input token x. $[\mathbf{\Gamma}(x)]_{i,j}$ corresponds to the cell value in $\mathbf{\Gamma}(x)$ for row i and column j and represents the transition score when consuming token x and transitioning from state i to j.

$$[\mathbf{\Gamma}(x)]_{i,j} = \begin{cases} \boldsymbol{u}_i \cdot \boldsymbol{v}_x + a_i & \text{if } j = i \text{ (self-loop transition)}, \\ \boldsymbol{w}_i \cdot \boldsymbol{v}_x + b_i & \text{if } j = i+1 \text{ (main-path transition)}, \\ \bar{0} & \text{otherwise.} \end{cases} \tag{6}$$

Here, $\boldsymbol{u}_i$ and $\boldsymbol{w}_i$ are learnable vectors and a_i and b_i are learnable scalar biases parameterizing transitions out of state i. $\boldsymbol{v}_x$ represents the word embedding for token x and $\bar{0}$ represents the zero value in the semiring used (Definition 10). Similarly, ϵ-transitions are parameterized with the following representation in $\mathbf{\Gamma}$:

$$[\mathbf{\Gamma}(\epsilon)]_{i,j} = \begin{cases} c_i & \text{if } j = i+1 \text{ (ϵ-transition)} \\ \bar{0} & \text{otherwise.} \end{cases} \tag{7}$$

Here, c_i represents a learnable scalar bias for ϵ-transitions out of state i. Next, Schwartz, Thomson, and Smith (2018) fix the initial vector $\boldsymbol{\lambda} = [\bar{1}, \bar{0}, \ldots, \bar{0}]$ and the final vector $\boldsymbol{\rho} = [\bar{0}, \bar{0}, \ldots, \bar{1}]$, where $\bar{1}$ and $\bar{0}$ represent the one and zero values specified in the semiring (Definition 10). This is a formalism to imply that there only exists one start and one final state and these are present at both extremes of the linear-chain WFA; which is ultimately consistent with the linear-chain definition (Definition 9).

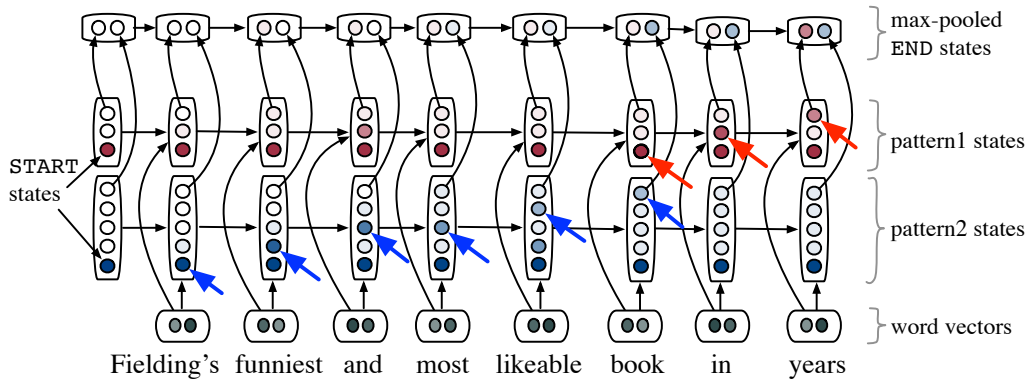

FIGURE 7: Schematic visualizing the SoPa model framework with two constituent WFAs highlighted in blue and red; figure taken from Schwartz, Thomson, and Smith (2018)

The aforementioned constraints in the linear-chain WFA structure result in the transition matrix $\mathbf{\Gamma}$ being reduced to a sparse diagonal matrix with the runtime of the linear-chain WFAs being reduced to $O(|Q||\boldsymbol{x}|)$, compared to the original runtime in Remark 13.2. Finally, it is worth noting that Schwartz, Thomson, and Smith (2018, Page 3, Section 3.1) refer to the linear-chain WFAs simply as *"patterns"*. For brevity and consistency, we refer to linear-chain WFAs as patterns interchangeably.

2.4.2 Document score

Definition 13 describes how a WFA can be used to compute a string score $\mathcal{A}[\![\boldsymbol{x}]\!]$ for an arbitrary string $\boldsymbol{x}$. Since SoPa was intended to compute scores for entire documents and not just short strings, Schwartz, Thomson, and Smith (2018, Page 3, Section 3.2) propose computing the string score over all consecutive substrings in the document which ultimately returns a document score $s_{\text{doc}}(\boldsymbol{y})$ for an arbitrary document $\boldsymbol{y}$. The document score $s_{\text{doc}}(\boldsymbol{y})$ for a WFA would represent an aggregated score over all consecutive substrings and would therefore also depend on the semiring used in the WFAs. In the case of max-based semirings using the Viterbi algorithm, the document score $s_{\text{doc}}(\boldsymbol{y})$ would reflect the highest scoring path score which corresponds to a substring in document $\boldsymbol{y}$.

2.4.3 Computational graph

Figure 7 shows a sample computational graph for the SoPa model with two WFAs highlighted in blue and red. The WFAs compute consecutive string scores as they traverse the document. Given max-based semirings, the max-pooled scores accumulate in an output layer as shown on the top of Figure 7. After traversing the full document, the max-pooled scores are passed through a Multi-Layer Perceptron (MLP) which conducts the final classification to an output label. It is worth noting that without ϵ-transitions and self-loops, a linear-chain WFA with $|\mathcal{Q}|$ states should always consume $|\mathcal{Q}| - 1$ tokens. However, by allowing the aforementioned special transitions; it is possible for strings of variable lengths to be consumed since an ϵ-transition can transition to the next state without consuming tokens while a self-loop can consume tokens without transitioning to the next state. This is indeed the case for Pattern 2 in Figure 7, when a self-loop is encountered in the transition from the token *"and"* to the token *"most"*.

2.4.4 Pattern hyperparameter

Training the SoPa model requires both commonly used and special hyperparameters. Commonly used hyperparameters include the learning rate, neuron dropout and word dropout. A special hyperparameter in the SoPa model is the pattern hyperparameter which contains information on the number of WFAs and their respective number of states allowed. This hyperparameter is encoded as a string with the following syntax: $\texttt{Length}_1\texttt{-Count}_1\texttt{\_}\ldots\texttt{\_Length}_n\texttt{-Count}_n$. An example of this hyperparameter could be `5-15_4-10_3-5`, which would signify 15 patterns of length 5, 10 patterns of length 4 and 5 patterns of length 3.

2.4.5 Transparency

Since we expounded on XAI in Section 2.1 and made a case for viewing ML models from the lens of XAI, it would only make sense to extend the same standards to the SoPa model. Based on the arguments made by Arrieta et al. (2020), we can classify the SoPa model as a black-box model since it closely resembles RNNs and CNNs; and a strong case has already been made in their study regarding the black-box natures of both RNNs and CNNs. Naturally, this would imply that post-hoc explainability methods are required to explain the SoPa model.

2.4.6 Post-hoc explainability methods

In their study, Schwartz, Thomson, and Smith (2018, Page 7, Section 7) describe simple methods of *"interpreting"* the SoPa model. One method involves the usage of back-pointers during the Viterbi computation to determine the patterns and substrings in a document which contributed the highest pattern scores. Another method involves zeroing out the corresponding patterns or WFAs via the occlusion sensitivity method to determine which pattern had the greatest impact on each classification decision. It is worth noting that the usage of *interpretation* in Schwartz, Thomson, and Smith (2018) is likely inconsistent with our definitions presented in Section 2.1. Given the common terminology misuse described in Section 2.1.3, it is more accurate to present the aforementioned *"interpretability"* method as an explainability method.

Using the post-hoc explainability taxonomies described in Section 2.1.4, we can correspondingly classify the explainability methods presented in Schwartz, Thomson, and Smith (2018) using terminology consistent with XAI research. The first explainability method uses individual text samples to determine the highest scoring substrings in documents; as well as the patterns or WFAs corresponding to them. Since this analysis is conducted at an individual document level and is never synthesized to a more global context, we would classify this under the local explanations explainability method. The next technique involves an occlusion or sensitivity analysis over all patterns and documents to determine which pattern had the greatest impact for each class. Since this involves systematic perturbation to determine the importance of pattern features, we would classify this method as a feature relevance explainability method.

Section 2.1.6 describes basic guidelines that could help elucidate what constitutes a good explanation; namely that a good explanation should be constrictive, provide causal links and be selective. We attempt to apply these guidelines with the aforementioned explainability techniques presented in SoPa. For the constrictive quality, it is likely that the explainability methods do not meet this criterion since they only highlight individual features that were important for SoPa, and do not necessarily go into detail regarding why these features superseded adjacent features. Next, the

explainability methods likely do not fulfill the criterion of providing causal links; since they generally provide explanations by aggregating (ultimately) probabilistic quantities inside SoPa. Finally in contrast, the explainability methods likely pass the criterion of being selective since they only provide the most important features for an explanation.

Chapter 3

Methodologies

In this chapter, we describe the methodologies used in this thesis. Comprehensive source code reflecting these methodologies can be found in our public GitHub repository[1].

3.1 Facebook Multilingual Task Oriented Dialog

Schuster et al. (2019) originally released the Facebook Multilingual Task Oriented Dialog (FMTOD) data set to encourage research in cross-lingual transfer learning for Natural Language Understanding (NLU) tasks; specifically from from high-resource to low-resource languages. The authors released the FMTOD data set with English as the high-resource language providing ∼43k annotated utterances, and Spanish and Thai as low-resource languages providing a total of ∼14k utterances. Furthermore, they streamlined the data set on two key tasks; namely intent detection and textual slot filling. In this thesis, we focus solely on the English language intent detection task in the FMTOD data set. This intent detection task entails a multi-label sequence classification task with a total of 12 classes from alarm, reminder and weather-related domains.

3.1.1 Motivation

We chose to work with the FMTOD data set since it is both a recently released and well-studied data set (Schuster et al., 2019; Zhang et al., 2019; Zhang, Lyu, and Callison-Burch, 2020). We focus on the English language intent classification task since it is a relatively straightforward task which allows us to place a greater focus on performance and explainability. Furthermore, the English language subset entails the highest resources in the FMTOD data set. Finally, we find the FMTOD data set's intent detection classification especially attractive because it allows us to test the SoPa++ model on a multi-class NLU problem; which is significantly different from the focus on binary classification sentiment detection tasks in SoPa (Schwartz, Thomson, and Smith, 2018).

3.1.2 Preprocessing

We enumerate our preprocessing steps below:

1. Similar to Schwartz, Thomson, and Smith (2018), we convert all FMTOD text samples to a lowercased format. This assists in simplifying the data set further.

[1] https://github.com/atreyasha/spp-explainability

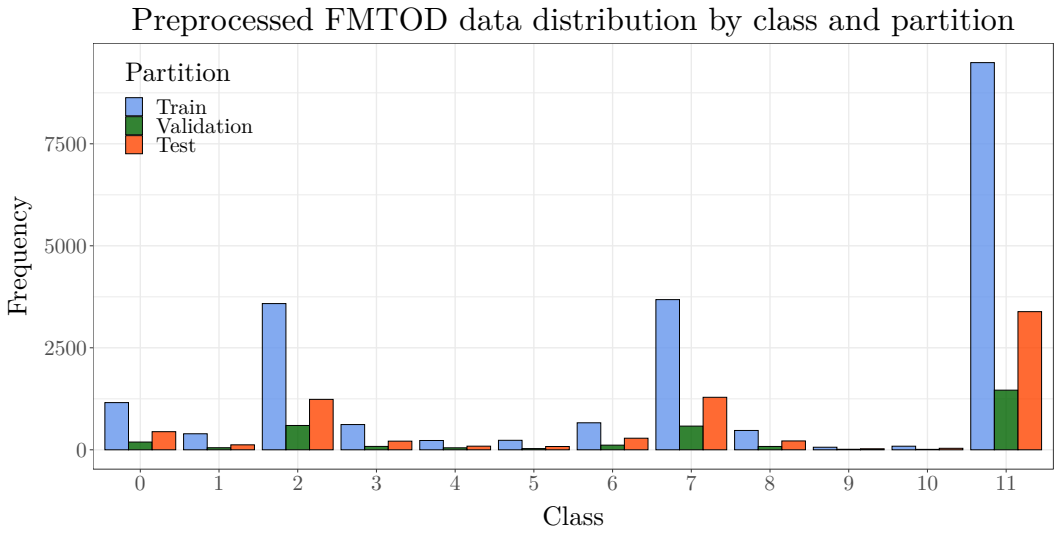

FIGURE 8: Data distribution of the preprocessed FMTOD data set grouped by classes and partitions

Class and description	Train	Validation	Test	Σ
0: `alarm/cancel_alarm`	1157	190	444	1791
1: `alarm/modify_alarm`	393	51	122	566
2: `alarm/set_alarm`	3584	596	1236	5416
3: `alarm/show_alarms`	619	83	212	914
4: `alarm/snooze_alarm`	228	49	89	366
5: `alarm/time_left_on_alarm`	233	30	81	344
6: `reminder/cancel_reminder`	662	114	284	1060
7: `reminder/set_reminder`	3681	581	1287	5549
8: `reminder/show_reminders`	474	82	217	773
9: `weather/check_sunrise`	63	13	25	101
10: `weather/check_sunset`	88	11	37	136
11: `weather/find`	9490	1462	3386	14338
Σ	20672	3262	7420	31354

TABLE 1: Frequency of the preprocessed FMTOD data set classes grouped by partitions; Σ signifies the cumulative frequency statistic

2. Next, we search through the pre-provided training, validation and test data partitions to remove duplicates within each partition.

3. Finally, we remove data duplicates which overlap between partitions. During this step, we do not remove any cross-partition duplicates from the test partition in order to keep it as similar as possible to the original test partition. This comes into importance later when we compare performance evaluations on the test set with other studies.

Many of the duplicates observed were already present in the original FMTOD data set, with additional duplicates being created from the initial lowercasing step. After preprocessing, we obtain a lowercased variant of the FMTOD data set with strictly unique data partitions. In the next section, we describe the summary statistics of the preprocessed FMTOD data set.

Class and description	Utterance length†	Example‡
0: `alarm/cancel_alarm`	5.6 ± 1.9	cancel weekly alarm
1: `alarm/modify_alarm`	7.1 ± 2.5	change alarm time
2: `alarm/set_alarm`	7.5 ± 2.5	please set the new alarm
3: `alarm/show_alarms`	6.9 ± 2.2	check my alarms.
4: `alarm/snooze_alarm`	6.1 ± 2.1	pause alarm please
5: `alarm/time_left_on_alarm`	8.6 ± 2.1	minutes left on my alarm
6: `reminder/cancel_reminder`	6.6 ± 2.2	clear all reminders.
7: `reminder/set_reminder`	8.9 ± 2.5	birthday reminders
8: `reminder/show_reminders`	6.8 ± 2.2	list all reminders
9: `weather/check_sunrise`	6.7 ± 1.7	when is sunrise
10: `weather/check_sunset`	6.7 ± 1.7	when is dusk
11: `weather/find`	7.8 ± 2.3	jacket needed?
μ	7.7 ± 2.5	—

†Summary statistics follow the mean $\pm$ standard-deviation format
‡Short and simple examples were chosen for brevity and formatting purposes

TABLE 2: Tabular summary of utterance length statistics and examples for FMTOD data classes; μ signifies the cumulative summary statistics

Study	Summary	Accuracy
Schuster et al. (2019)	BiLSTM jointly trained on both the slot filling and intent detection English language tasks	99.1%
Zhang et al. (2019)	BERT along with various decoders jointly fine-tuned on both the slot filling and intent detection English language tasks	96.6–98.9%
Zhang, Lyu, and Callison-Burch (2020)	RoBERTa and XLM-RoBERTa fine-tuned on the English language and multilingual intent detection tasks respectively along with WikiHow pre-training	99.3–99.5%

TABLE 3: Tabular summary of studies that addressed the FMTOD intent detection English language task, along with their relevant summaries and accuracy range(s)

3.1.3 Summary statistics

Figure 8 shows the summary statistics of the preprocessed FMTOD data set grouped by classes and data set partitions. Similarly, Table 1 shows the same summary statistics in a tabular form with explicit frequencies. Based on the summary statistics, we can observe that the preprocessed FMTOD data set is significantly imbalanced with $\sim$45% of samples falling into Class 11 alone. We take this observation into consideration in later sections and apply fixes to mitigate this data imbalance. In addition, we observe from Table 2 that input utterances in the preprocessed FMTOD data set are generally short; with a mean input utterance length of 7.7 and a standard deviation of 2.5 tokens. Utterance length summary statistics were computed with the

assistance of NLTK's default `Treebank` word tokenizer (Bird and Loper, 2004).

3.1.4 Performance range

Several studies have optimized deep learning models on the FMTOD English language intent classification task using a variety of models from BiLSTMs to XLM-RoBERTa (Schuster et al., 2019; Zhang et al., 2019; Zhang, Lyu, and Callison-Burch, 2020). Table 3 summarizes these studies along with their reported accuracy scores on the FMTOD English language intent classification task. Based on the presented results from these recent studies, we can infer that the general competitive accuracy range for the FMTOD English language intent classification task is from 96.6% to 99.5%.

3.2 SoPa++

In this section we describe our SoPa++ model's architecture and present both similarities and differences compared to the SoPa model in Schwartz, Thomson, and Smith (2018).

3.2.1 Strict linear-chain WFA-ω

As mentioned in Section 2.4, Schwartz, Thomson, and Smith (2018) constructed the SoPa model with an ensemble of linear-chain WFAs which permitted both ϵ and self-loop transitions. As noted in Section 2.4.3, ϵ and self-loop transitions are useful constructs in abstracting WFAs and allowing them to consume or match variable length strings. However based on our experimentation during our development phase, we observed a key concern that the highest scoring paths in the WFAs in SoPa tended to have a large variation of string lengths due to the effect of both ϵ-transitions and self-loops. We believe that this reduced the impact of SoPa's explainability methods and as a result, the first change we decided for was to remove both ϵ and self-loop transitions. With this change, we could at least ensure that each WFA would always consume strings of fixed lengths.

However, consuming strings of fixed lengths could also be seen as a form of overfitting in the model; since a model could simply memorize short strings or phrases and would not necessarily "learn" to generalize. To address this concern, we include a wildcard transition which we define here as a ω-transition. Allowing for such a transition was only natural since wildcards are already crucial parts of regular expressions; which as we mentioned are equivalent to FAs. To support this, we provide the following definition for a modified WFA-ω:

Definition 14 (Weighted finite-state automaton-ω)**.** A weighted finite-state automaton-ω over a semiring $\mathbb{K}$ is a 5-tuple $\mathcal{A} = \langle \Sigma, \mathcal{Q}, \boldsymbol{\Gamma}, \boldsymbol{\lambda}, \boldsymbol{\rho} \rangle$, with:

- a finite input alphabet Σ;
- a finite state set $\mathcal{Q}$;
- transition matrix $\boldsymbol{\Gamma} : \mathcal{Q} \times \mathcal{Q} \times (\Sigma \cup \{\omega\}) \to \mathbb{K}$;
- initial vector $\boldsymbol{\lambda} : \mathcal{Q} \to \mathbb{K}$;
- and final vector $\boldsymbol{\rho} : \mathcal{Q} \to \mathbb{K}$.

Remark 14.1. An ω transition is equivalent to a wildcard transition, which consumes an arbitrary token input and moves to the next state

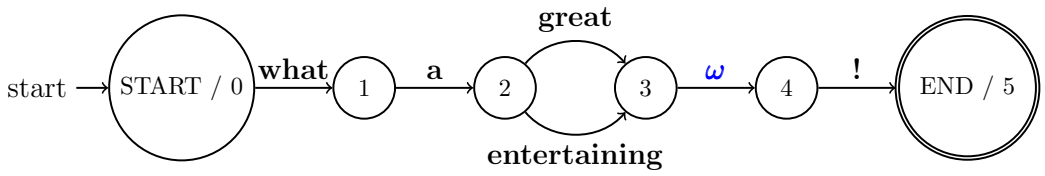

FIGURE 9: Schematic visualizing a strict linear-chain NFA with ω (blue) and main-path (black) transitions

Remark 14.2. Besides the inclusion of the ω-transition and removal of the ϵ-transition, an WFA-ω has all of the same characteristics as the WFA defined in Definition 11.

Comparing with the linear-chain WFAs used in Schwartz, Thomson, and Smith (2018) as mentioned in Section 2.4.1, our linear-chain WFA-ω is similarly alloted a sequence of $|Q|$ states. However, each state i in the linear-chain WFA-ω has only two possible outgoing transitions; namely a **ω-transition** which consumes an arbitrary input token and transitions to state $i+1$ and a **main-path transition** which consumes a specific token and transitions to state $i+1$. Similar to Schwartz, Thomson, and Smith (2018), we utilize only the max-sum and max-product semirings in our linear-chain WFA-ω's. Because of the elimination of self-loop transitions, we refer to our linear-chain WFA-ω as *strict* linear-chain WFA-ω as per Definition 9.

Next, we provide a mathematical formulation of the modified transition matrix $\mathbf{\Gamma}$ in our linear-chain WFA-ω. Here, $\mathbf{\Gamma}(x)$ represents a $|Q| \times |Q|$ matrix containing transition scores when consuming an input token x. $[\mathbf{\Gamma}(x)]_{i,j}$ corresponds to the cell value in $\mathbf{\Gamma}(x)$ for row i and column j and represents the transition score when consuming token x and transitioning from state i to j.

$$[\mathbf{\Gamma}(x)]_{i,j} = \begin{cases} \boldsymbol{w}_i \cdot \boldsymbol{v}_x + b_i & \text{if } j = i+1 \text{ (main-path transition),} \\ \bar{0} & \text{otherwise.} \end{cases} \tag{8}$$

Here, $\boldsymbol{w}_i$ and b_i are learnable vector and scalar parameters parameterizing transitions out of state i to state $i+1$. $\boldsymbol{v}_x$ represents the word embedding for token x and $\bar{0}$ represents the zero value in the semiring used (Definition 10). Similarly, ω-transitions are parameterized with the following representation in $\mathbf{\Gamma}$:

$$[\mathbf{\Gamma}(\omega)]_{i,j} = \begin{cases} c_i & \text{if } j = i+1 \text{ (ω-transition)} \\ \bar{0} & \text{otherwise.} \end{cases} \tag{9}$$

Here, c_i represents a learnable scalar bias for ω-transitions out of state i to state $i+1$. Finally as per Schwartz, Thomson, and Smith (2018), we fix the initial vector $\boldsymbol{\lambda} = [\bar{1}, \bar{0}, \ldots, \bar{0}]$ and the final vector $\boldsymbol{\rho} = [\bar{0}, \bar{0}, \ldots, \bar{1}]$, where $\bar{1}$ and $\bar{0}$ represent the one and zero values specified in the semiring (Definition 10). The time-complexity of the Viterbi algorithm to compute the string score for our linear-chain WFA-ω's is $O(|Q||\boldsymbol{x}|)$, where $|Q|$ refers to the number of states and $|\boldsymbol{x}|$ refers to the length of the input string.

Ultimately, the introduction of a strict linear-chain WFA-ω allows us to attain fixed string length consumption with an added layer of generalization because of the introduction of wildcards. An example of a strict linear-chain NFA extracted from a strict linear-chain WFA-ω is shown in Figure 9. Interestingly, we can observe that this FA corresponds to the Perl-compatible regular expression "`what a (great|entertaining) [^\\s]+ !`", where `[^\\s]+` refers to any set of consecutive characters which are not separated by a space character.

Algorithm 1 Strict linear-chain WFA-ω document score*

Input(s): Strict linear-chain WFA-ω (denoted as $\mathcal{A}$) and document $\boldsymbol{y}$
Output(s): Document score $s_{\text{doc}}(\boldsymbol{y})$

function DOCSCORE($\mathcal{A}, \boldsymbol{y}$)
  $\boldsymbol{h}_0 \leftarrow [\bar{1}, -\infty, \ldots, -\infty]$ ▷ Create initial hidden state vector $\boldsymbol{h}_0 : \mathcal{Q} \to \mathbb{K}$
  for $i \leftarrow 1, 2, \ldots, |\boldsymbol{y}|$ **do** ▷ Sequentially iterate over each token y_i in string $\boldsymbol{y}$
    $\boldsymbol{m} \leftarrow [[\boldsymbol{\Gamma}(y_i)]_{1,2}, [\boldsymbol{\Gamma}(y_i)]_{2,3}, \ldots, [\boldsymbol{\Gamma}(y_i)]_{|\mathcal{Q}|-1,|\mathcal{Q}|}]$ ▷ Main-path scores for token y_i
    $\boldsymbol{\omega} \leftarrow [[\boldsymbol{\Gamma}(\omega)]_{1,2}, [\boldsymbol{\Gamma}(\omega)]_{2,3}, \ldots, [\boldsymbol{\Gamma}(\omega)]_{|\mathcal{Q}|-1,|\mathcal{Q}|}]$ ▷ State-wise wildcard scores
    $\boldsymbol{m}' \leftarrow \boldsymbol{m} \otimes \boldsymbol{h}_{i-1}[:-1]$ ▷ Path score with main-path transitions†
    $\boldsymbol{\omega}' \leftarrow \boldsymbol{\omega} \otimes \boldsymbol{h}_{i-1}[:-1]$ ▷ Path score with ω transitions†
    $\boldsymbol{h}_i \leftarrow [\bar{1}] \,\|\, \max(\boldsymbol{m}', \boldsymbol{w}')$ ▷ Concatenate $\bar{1}$ to maximum of $\boldsymbol{m}$ and $\boldsymbol{\omega}$
  end for
  $s_{\text{doc}}(\boldsymbol{y}) \leftarrow \max_{i \in 1,2,\ldots,|\boldsymbol{y}|} \boldsymbol{h}_i[-1]$ ▷ Get maximum of hidden vector final states‡
  return $s_{\text{doc}}(\boldsymbol{y})$
end function

*$\otimes$ and $\bar{1}$ are derived from max-based semirings where all semiring operations are element-wise
†$\boldsymbol{h}_i[:-1]$ follows the Python indexing syntax and implies keeping all elements of $\boldsymbol{h}_i$ except the last
‡$\boldsymbol{h}_i[-1]$ follows the Python indexing syntax and implies retrieving the last element of the vector $\boldsymbol{h}_i$

3.2.2 Document score

Similar to Schwartz, Thomson, and Smith (2018), SoPa++ was intended to compute scores for entire documents and not just fixed-length strings using the strict linear-chain WFA-ω. To achieve this, we propose Algorithm 1 to compute the document score $s_{\text{doc}}(\boldsymbol{y})$ for an arbitrary document $\boldsymbol{y}$. Here, we score all consecutive substrings in a document $\boldsymbol{y}$ using either the max-sum or max-product semirings assisted with the Viterbi algorithm. Following from this, the document score $s_{\text{doc}}(\boldsymbol{y})$ for an arbitrary document $\boldsymbol{y}$ would reflect the highest scoring path which corresponds to the a substring in document $\boldsymbol{y}$.

3.2.3 TauSTE

In Section 2.2 we described the concept of a STE in activation quantized neural networks and explained how STEs function in both their forward and backward passes. Furthermore, we provided a motivation as to why STEs and other quantized activation functions are of interest; for example in relation to computational savings linked to low-precision computing. In our implementation of the SoPa++ model, we make use of a variant of the STE activation function; which we define here as the Tau straight-through estimator (TauSTE).

$$\text{TauSTE}(x) = \begin{cases} 1 & x \in (\tau, +\infty) \\ 0 & x \in (-\infty, \tau] \end{cases} \tag{10}$$

$$\text{TauSTE}'(x) = \begin{cases} 1 & x \in (1, +\infty) \\ x & x \in [-1, 1] \\ -1 & x \in (-\infty, -1) \end{cases} \tag{11}$$

Figure 10 shows a schematic of the TauSTE's forward and backward passes. As we can see, there are two key changes from the vanilla STE to the TauSTE. Firstly, the threshold for activation in the forward function is now governed by some variable $\tau \in \mathbb{R}$. This was done to allow for some degree of freedom in deciding the activation

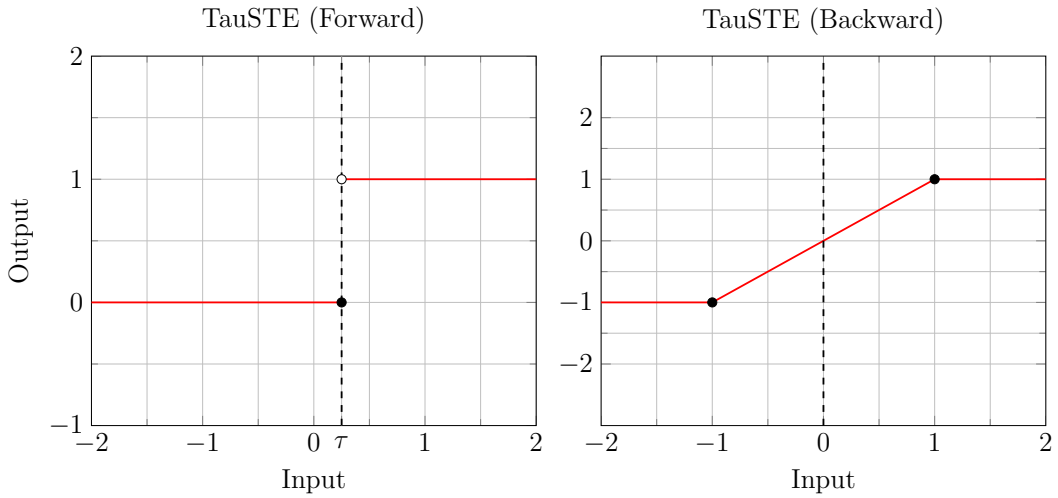

FIGURE 10: Schematic visualizing the TauSTE's forward and backward passes

threshold. Secondly, the backward pass returns the identity function on a limited range on inputs; specifically on $x \in [-1, 1]$. This restriction on the backward pass was placed to ensure that gradients do not blow up in size.

As we describe in detail in the next sections, we chose to use this special variant of the STE because we believe that the TauSTE could prove to be useful for the explainability purposes of our SoPa++ model. This is mainly because the TauSTE (or even the STE) activation function simplifies continuous inputs into discrete outputs; thereby reducing the information content of the signal it receives as inputs. In later sections, we show how we utilize and capitalize this reduction in signal information in our SoPa++ model.

3.2.4 Tokenization and embeddings

Similar to Schwartz, Thomson, and Smith (2018), we utilize NLTK's default `Treebank` word tokenizer (Bird and Loper, 2004) to conduct tokenization of input utterances into word-level tokens. Next, we pad input utterances with special `[START]` and `[END]` tokens at the start and end indices of the utterances to signify the location where the utterance begins and ends. Finally, while Schwartz, Thomson, and Smith (2018) utilize GloVe 840B 300-dimensional true-cased embeddings, we utilize GloVe 6B 300-dimensional uncased word-level embeddings (Pennington, Socher, and Manning, 2014) to project the input tokens in utterances to continuous numerical spaces. We utilized this smaller subset of word-level embeddings since it allowed us to work with lower-cased words, as well as experiment with a variety of embedding dimensions during our development phase.

3.2.5 Computational graph

In this subsection, we describe the computational graph of the SoPa++ model by specifically referring to its several neural components. This description is linked to the visualization of the computational graph of SoPa++ in Figure 11. Firstly as described in the previous section, we tokenize the input utterance, pad it with special tokens and project input tokens to numerical spaces using the aforementioned GloVe word embeddings. Following this, we use our ensemble of $m \in \mathbb{N}$ strict linear-chain WFA-ω's to traverse the input utterance and provide an ultimate document score for

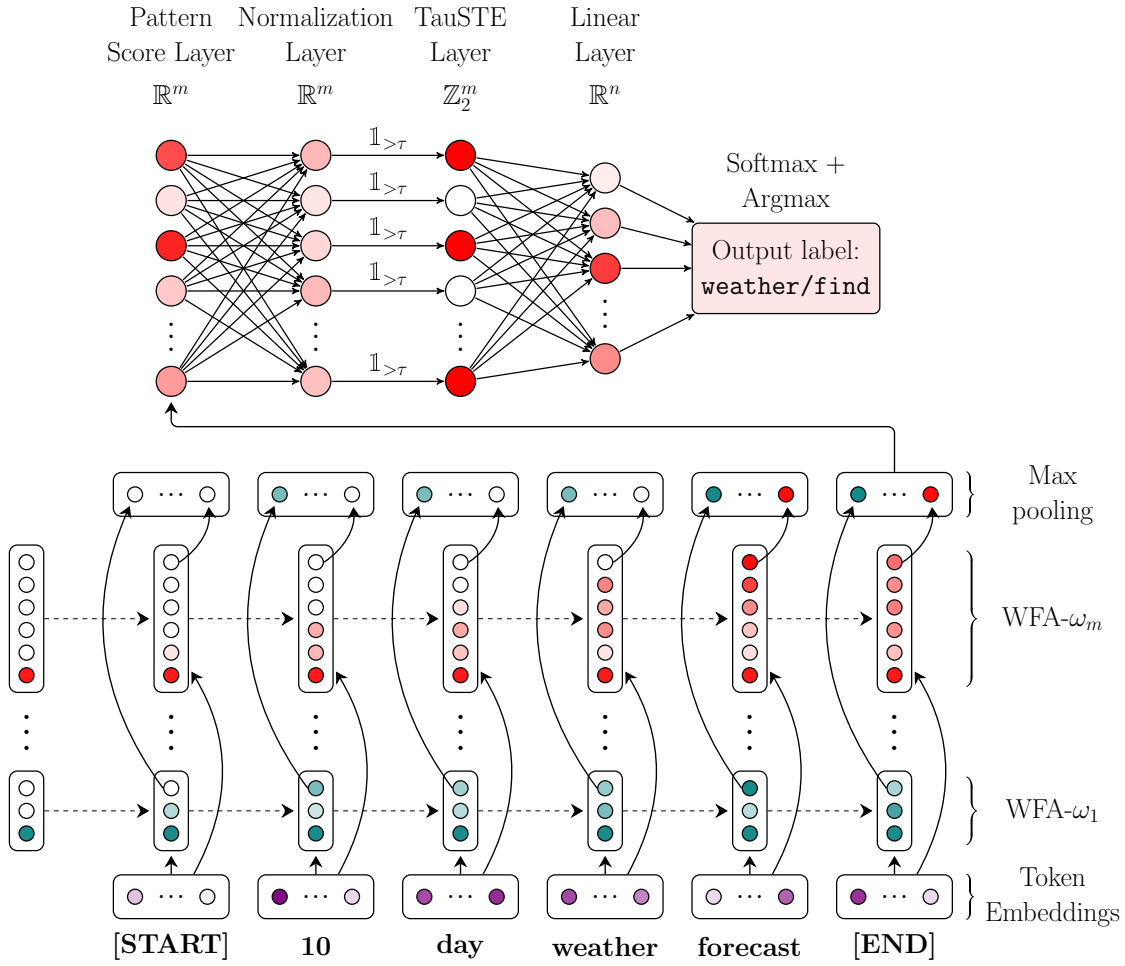

FIGURE 11: Schematic visualizing the computational graph of the SoPa++ model

this utterance; as prescribed by Algorithm 1. When processing each input token, we monitor the end states of each of the $m \in \mathbb{N}$ WFA-ω and max-pool the score present in this state. In the edge case that an input string was too short for the end state of WFA-ω to register a non-$\bar{0}$ score, we simply discard this score in further analysis. This description so far corresponds to the lower half of Figure 11.

After max-pooling scores from all the $m \in \mathbb{N}$ WFA-ω's, we then pass this collection of pattern scores for further processing using SoPa++'s other neural components. This can be seen in the upper portion of Figure 11. Firstly, we apply layer normalization (Ba, Kiros, and Hinton, 2016) to all pattern scores without applying any additional affine transformations. We omit the affine transformation to not alter any of the pattern score information content and use layer normalization as an expedient means of projecting the values of pattern scores to a standard normal distribution. This furthermore guarantees that pattern scores would be smaller in size and would have a roughly even distribution of small positive and negative values around 0.

This projection to a standard normal distribution becomes very useful as we next encounter the TauSTE layer. Here, this layer maps all inputs which are strictly larger than the $\tau \in \mathbb{R}$ threshsold to 1 and all others inputs to zero. The TauSTE layer naturally is only useful when it is able to discriminate the inputs by mapping some of them to 1 and some to 0, instead of always mapping all inputs to either 1 or 0. Without layer normalization, the TauSTE layer would not be able to perform its

function since pattern scores tend to be mostly positive with differing ranges. Layer normalization therefore helps to project these variations of scores to a uniform range; which ultimately allows the TauSTE layer to be useful.

A natural criticism of the TauSTE layer could be that it strongly limits the flow of information in the SoPa++ model. While the binarization present in this layer does indeed limit the rich flow of continuous numerical information, it is still worth noting that this layer can preserve sufficient information given a sufficiently large $m \in \mathbb{N}$ value and therefore number of WFA-ω and TauSTE neurons. For example, if we allow for $m = 40$ and therefore provide 40 WFA-ω and TauSTE neurons, we can have a total of $2^{40} \approx 1.1 \times 10^{12}$ binary state possibilities; which is slightly greater than one trillion state possibilities. To provide some context to this order of magnitude, the aforementioned number of possibilities is roughly equal to estimated number of stars present in the Andromeda Galaxy (Peñarrubia et al., 2014). This would imply that despite the reduction in information content on the TauSTE layer, there are still sufficient mappable states available to learn various representations relevant to output classes.

After binarizing the input values in the TauSTE layer, we apply a simple linear or affine transformation to the inputs to modify their dimensionality from $m \in \mathbb{N}$ to $n \in \mathbb{N}$, where the n represents the number of output classes. We specifically chose a linear regression layer over a MLP because linear regressors are known to be transparent models (Arrieta et al., 2020) and this is a feature which ultimately assists us in the explainability portion of the SoPa++ model, which we describe in greater detail in the next sections.

Finally, after applying the linear transformation; we apply a softmax function over the linear outputs to project them to probabalistic spaces and then extract the highest scoring index to represent the predicted class. In the case of Figure 11, the output class for the input pre-processed sentence ‘‘`[START] 10 day weather forecast [END]`’’ is the `weather/find` label which corresponds to class 11.

3.2.6 Transparency

Similar to the SoPa model as mentioned in Section 2.4.5, SoPa++ is also a hybrid model consisting of RNN, CNN and weighted finite-state neural components. Following the standards of transparency recommended by Arrieta et al. (2020), we can conclude that SoPa++ would correspondingly fall into the black-box ML model category. Because of this classification, the SoPa++ model would require post-hoc explainability methods in order to explain its inner mechanisms. In the next section, we expound more on the explanations by simplification post-hoc explainability method that can be used on the SoPa++ model.

3.3 RE proxy

In Section 2.1.4 we introduced three important post-hoc explainability techniques; specifically local explanations, feature relevance and explanations by simplification. In this section, we describe how we leverage the latter explanations by simplifications technique with our SoPa++ model. Specifically, we show how we can use special features of the antecedent SoPa++ model to effectively simplify it into a Regular Expression (RE) proxy model.

Algorithm 2 Strict linear-chain WFA-ω document score with corresponding path*

Input(s): Strict linear-chain WFA-ω (denoted as $\mathcal{A}$) and document $\boldsymbol{y}$
Output(s): Document score $s_{\text{doc}}(\boldsymbol{y})$ and its corresponding path $\pi_{\text{doc}}(\boldsymbol{y})$

function DOCSCORE_TRACE($\mathcal{A}, \boldsymbol{y}$)
  $\boldsymbol{h}_0 \leftarrow [\bar{1}, -\infty, \ldots, -\infty]$ ▷ Create initial hidden state vector $\boldsymbol{h}_0 : \mathcal{Q} \rightarrow \mathbb{K}$
  for $i \leftarrow 1, 2, \ldots, |\boldsymbol{y}|$ **do** ▷ Sequentially iterate over each token y_i in string $\boldsymbol{y}$
    $\boldsymbol{m} \leftarrow [[\boldsymbol{\Gamma}(y_i)]_{1,2}, [\boldsymbol{\Gamma}(y_i)]_{2,3}, \ldots, [\boldsymbol{\Gamma}(y_i)]_{|\mathcal{Q}|-1,|\mathcal{Q}|}]$ ▷ Main-path scores for token y_i
    $\boldsymbol{\omega} \leftarrow [[\boldsymbol{\Gamma}(\omega)]_{1,2}, [\boldsymbol{\Gamma}(\omega)]_{2,3}, \ldots, [\boldsymbol{\Gamma}(\omega)]_{|\mathcal{Q}|-1,|\mathcal{Q}|}]$ ▷ State-wise wildcard scores
    $\boldsymbol{m'} \leftarrow \boldsymbol{m} \otimes \boldsymbol{h}_{i-1}[:-1]$ ▷ Path score with main-path transitions†
    $\boldsymbol{\omega'} \leftarrow \boldsymbol{\omega} \otimes \boldsymbol{h}_{i-1}[:-1]$ ▷ Path score with ω transitions†
    $\boldsymbol{h}_i \leftarrow [\bar{1}] \,\|\, \max(\boldsymbol{m'}, \boldsymbol{w'})$ ▷ Concatenate $\bar{1}$ to maximum of $\boldsymbol{m}$ and $\boldsymbol{\omega}$
    $\pi_i \leftarrow \text{trace}(h_i[-1])$ ▷ Back-trace path π_i corresponding to $h_i[-1]$‡
  end for
  $j \leftarrow \arg\max_{i \in 1,2,\ldots,|\boldsymbol{y}|} \boldsymbol{h}_i[-1]$ ▷ Get index of hidden vector final states' maximum‡
  $s_{\text{doc}}(\boldsymbol{y}) \leftarrow \boldsymbol{h}_j[-1]$ ▷ Extract maximum hidden vector final state value‡
  $\pi_{\text{doc}}(\boldsymbol{y}) \leftarrow \pi_j$ ▷ Extract path corresponding to maximum value
  return $[s_{\text{doc}}(\boldsymbol{y}), \pi_{\text{doc}}(\boldsymbol{y})]$
end function

*$\otimes$ and $\bar{1}$ are derived from max-based semirings where all semiring operations are element-wise
†$\boldsymbol{h}_i[:-1]$ follows the Python indexing syntax and implies keeping all elements of $\boldsymbol{h}_i$ except the last
‡$\boldsymbol{h}_i[-1]$ follows the Python indexing syntax and implies retrieving the last element of the vector $\boldsymbol{h}_i$

3.3.1 Document score with corresponding path

As described in Definition 13, the Viterbi algorithm returns the highest path score which can naturally be attributed to a certain path or set of transitions. In order to simplify the SoPa++ model into a RE proxy model, we first need to modify our document scoring algorithm to not only return the document score $s_{\text{doc}}(\boldsymbol{y})$ but also its corresponding path $\pi_{\text{doc}}(\boldsymbol{y})$. This process is described in Algorithm 2 where we trace the exact path of each transition and return the exact path in addition to the highest path-score. This algorithm ultimately allows us to extract the highest scoring path for each strict linear-chain WFA-ω used in SoPa++ per input utterance. Similar to Algorithm 1, this algorithm also has a time-complexity of $O(|Q||\boldsymbol{x}|)$, where $|\mathcal{Q}|$ refers to the number of states and $|\boldsymbol{x}|$ refers to the length of the input string. It is also worth noting that the highest scoring path returned ultimately reflects a set of transitions in the form of a strict linear-chain NFA; which can correspondingly be transformed into a regular expression. Therefore, it can be inferred that Algorithm 2 returns the best scoring regular expression corresponding to a certain strict linear-chain WFA-ω.

3.3.2 Simplifying SoPa++ to RE proxy

We now describe the simplification process from a neural SoPa++ model into a RE proxy model as reflected in Algorithm 3. First, we assume that already have a fully trained SoPa++ model given a certain task. We leave the details of training to the next sections. With this trained SoPa++ model, we use Algorithm 2 to retrieve the best scoring paths for each training data instances and each of the $m \in \mathbb{N}$ strict linear-chain WFA-ω's in the SoPa++ model. This would imply that for each input utterance in the training set, we collect the m best path transitions for the m WFA-ω's, with each best path corresponding to a regular expression. Naturally, not all best paths and regular expressions were important for the classification decision of the

Algorithm 3 Extracting regular expression lookup layer from SoPa++

Input(s): Trained SoPa++ model $\mathcal{S}$ and training data $[\boldsymbol{y}_1, \ldots, \boldsymbol{y}_t]$
Output(s): Regular expression lookup layer $[\{\mathbf{RE}\}_1, \ldots, \{\mathbf{RE}\}_m]$

function EXTRACT_LOOKUP($\mathcal{S}, [\boldsymbol{y}_1, \ldots, \boldsymbol{y}_t]$)
  $[\{\mathbf{RE}\}_1, \ldots, \{\mathbf{RE}\}_m] \leftarrow [\emptyset, \ldots, \emptyset]$ ▷ Initialize regex lookup layer with empty sets
  for $i \leftarrow 1, 2, \ldots, \boldsymbol{t}$ **do** ▷ Loop over training data
    for $j \leftarrow 1, 2, \ldots, m$ **do** ▷ Loop over all WFA-ω's in $\mathcal{S}$
      $\mathcal{A}_j \leftarrow$ get_WFA($\mathcal{S}, j$) ▷ Get a WFA-ω by index
      $s^j_{\text{doc}}(\boldsymbol{y}_i), \pi^j_{\text{doc}}(\boldsymbol{y}_i) \leftarrow$ DOSCORE_TRACE($\mathcal{A}_j, \boldsymbol{y}_i$)
      if TauSTE($s^j_{\text{doc}}(\boldsymbol{y}_i)$) == 1 **then** ▷ Save current $\pi^j_{\text{doc}}(\boldsymbol{y}_i)$ if it activates TauSTE
        $\{\mathbf{RE}\}_j \leftarrow \{\mathbf{RE}\}_j \cup \pi^j_{\text{doc}}(\boldsymbol{y}_i)$
      end if
    end for
  end for
  return compress($[\{\mathbf{RE}\}_1, \ldots, \{\mathbf{RE}\}_m]$) ▷ Compress regular expression lookup layer
end function

antecedent SoPa++ model. Specifically, only the regular expressions corresponding to activated TauSTE neurons were of importance to the decision of the antecedent SoPa++ model; while all other regular expressions were effectively discarded at the TauSTE layer.

This leads us to several interesting conclusions. Firstly, we observe here that the TauSTE layer is not only useful for reducing information complexity; but also for attributing causal links to the model's decision-making. This is shown by the layer effectively reducing complexity and attributing the decision-making process to certain regular expressions. Next, we can also observe an effect of the $\tau \in \mathbb{R}$ threshold here. If the τ-threshold is low, we effectively allow more TauSTE neurons to be activated and therefore allow more regular expressions from the strict linear-chain WFA-ω to affect the SoPa++ model's final decision. Contrastingly, if the τ-threshold is high; we allow fewer TauSTE neurons to be activated and therefore allow fewer regular expressions to be used for the final decision making process. Therefore, we can infer that the TauSTE operates as a filtration layer which allows us to construct causal links to simplify the SoPa++ model into a RE proxy model.

With this, we collect regular expressions corresponding to strict linear-chain WFA-ω's whose document scores led to activations in the TauSTE layer. We compress and save these regular expressions in a database which we refer to as the regular expression lookup layer. We ultimately use these collection of regular expressions as a proxy to approximate the SoPa++ model's internal mechanisms until the TauSTE layer. This collection of important regular expressions comes into use later. We then discard all other parts of the SoPa++ model except the linear regression layer. Finally, we combine the regular expression lookup layer and linear regression layer to form our RE proxy model.

3.3.3 Computational graph

With a pre-trained SoPa++ model, we have shown that we can extract a unique RE proxy model that approximates the antecedent SoPa++ model. In this subsection we describe the computational graph of the RE proxy model with the aid of Figure 12.

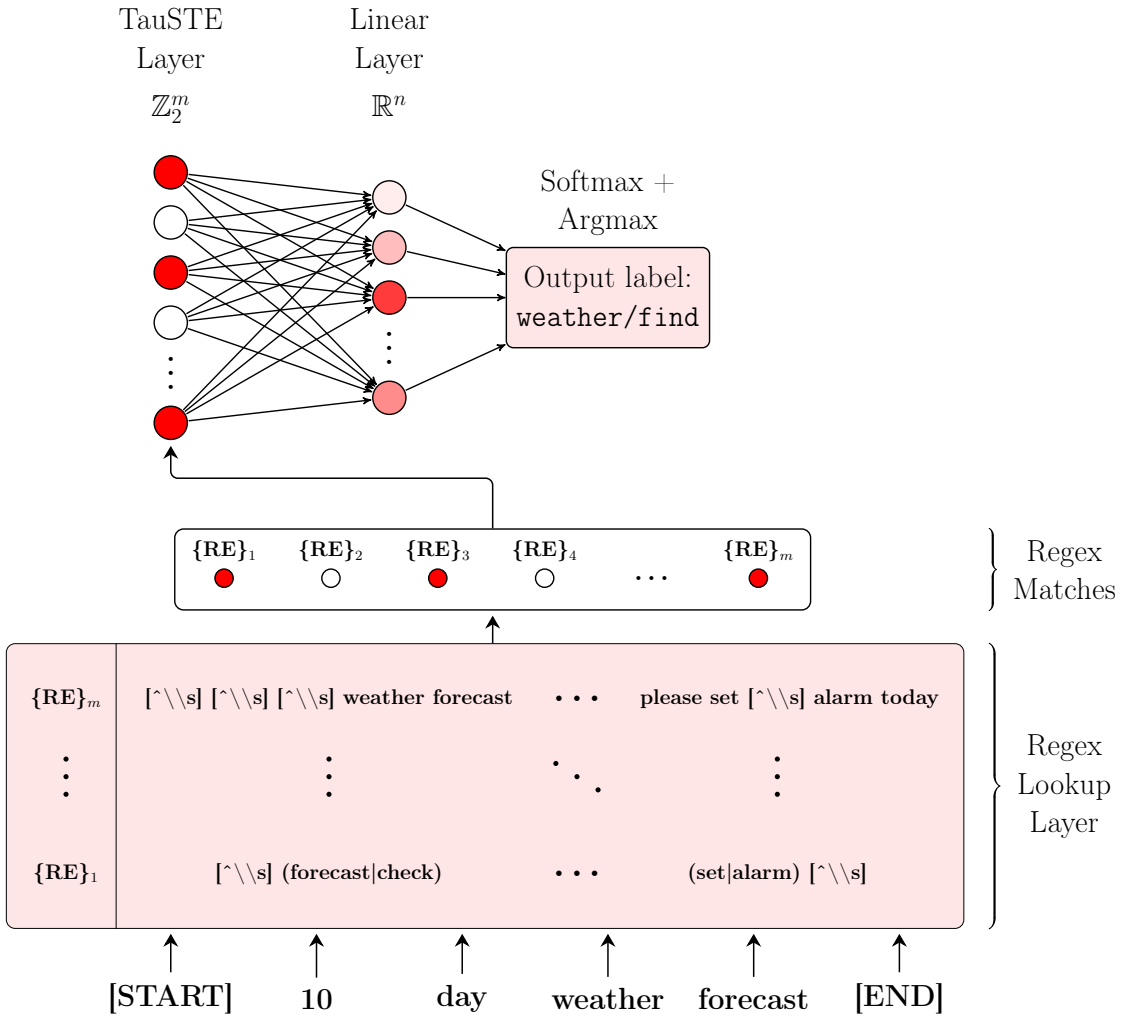

FIGURE 12: Schematic visualizing the computational graph of the RE proxy model

Similar to the computational graph of SoPa++, we process our input text sequentially. However, instead of using word embeddings and neural network components; we pass our input utterance through our RE lookup layer which conducts substring matches over all sets of regular expressions in $[\{\mathbf{RE}\}_1, \ldots, \{\mathbf{RE}\}_m]$. If any regular expression set $\{\mathbf{RE}\}_i$ matches the input utterance, the index of this set if given a value of 1. If no regular expression in the set matches, its index is given a value of 0. These values are assembled into a vector of 1's and 0's and resemble the values present in the TauSTE layer in the antecedent SoPa++ model. This binary vector is then passed to the linear regression layer; which transforms the binary vector back to continuous space. Following a softmax and argmax, we arrive at our output label.

3.3.4 Transparency

As mentioned, the RE proxy model is obtained using an explanations by simplification post-hoc explainability method on the antecedent SoPa++ model. As per Definition 6, this method should show that the proxy model is simpler or more transparent compared to the antecedent model. As discussed previously in Section 3.2.6, SoPa++ can be classified as a black-box model. The RE proxy model, on the other hand, can

Characteristic	SoPa	SoPa++
Input casing	True-cased	Lower-cased
Tokenization	NLTK `Treebank` word tokenizer	NLTK `Treebank` word tokenizer
Token embeddings	GloVe 840B 300-dimensions	GloVe 6B 300-dimensions
WFAs	Linear-chain WFAs with epsilon, self-loop and main-path transitions	Strict linear-chain WFA-ω's with ω and main-path transitions
Hidden layers	MLP after max-pooling	Layer normalization, TauSTE and linear regression layer
Transparency	Black-box	Black-box
Post-hoc explainability	Local explanations, feature relevannce	Explanations by simplification

TABLE 4: Tabular comparison of SoPa vs. SoPa++ models

be seen as a hybrid of two simpler transparent models. For one, the RE lookup layer can be seen as a rules-based sub-model which uses pre-existing rules in the form of regular expressions to generate hidden model features. Next, these hidden features are passed onto a linear regression layer for weighting. According to Arrieta et al. (2020, Page 7, Section 3), both rules-based and linear regression ML models are transparent models. Using this, we can generally conclude the the RE proxy should also be a transparent model.

It is worth noting that there could be exceptions to this rule as mentioned in Arrieta et al. (2020, Page 9, Table 2); especially when there exist too many rules or REs in the RE lookup layer or when there exist too many features for the linear regression layer. These will be further discussed in the later Discussions chapter.

3.4 SoPa vs. SoPa++

Table 4 summarizes key differences between the SoPa and SoPa++ models. The most significants modifications to SoPa++ include the modification of linear-chain WFAs to strict linear-chain WFA-ω's, the addition of the TauSTE layer and the significantly different explanations by simplification post-hoc explainability technique.

3.5 RQ1: Evaluating performance of SoPa++

In this section, we describe the methodologies used to answer our first research question regarding the competitive performance of the SoPa++ model on the FMTOD data set. Specifically, we describe how we trained the SoPa++ model and afterwards, how we proceeded to evaluate and compare its performance to other studies.

Hyperparameter	Values
Patterns	{`6-10_5-10_4-10_3-10`, `6-25_5-25_4-25_3-25`, `6-50_5-50_4-50_3-50`}
τ-threshold	$\{0.00, 0.25, 0.50, 0.75, 1.00\}$

TABLE 5: Grid-search hyperparameters and their respetive values

3.5.1 Training

First and foremost, we address the issue of data imbalance in the FMTOD data set as mentioned in Section 3.1.3. For this, we chose a simple but effective solution of up-sampling all minority data classes such that they would have the same frequency as the majority class. We chose this approach over other approaches such as gradient-weighting since this is generally a model agnostic and straightforward approach.

Returning to the computational graph of SoPa++ in Figure 11, we compute the cross-entropy loss using the softmax output and target labels. Since SoPa++ is in essence a deep neural network, we utilize the well-studied gradient descent technique to train and update our SoPa++ model. We utilize the `autograd` engine of PyTorch[2] to assist with gradient computations, backward passes and parallelized computations. We utilize the Adam optimizer (Kingma and Ba, 2015) to stabilize the gradient descent learning technique with $\beta_1 = 0.9$ and $\beta_2 = 0.999$.

In terms of fixed training variables, we utilize a learning rate of 0.001, a batch size of 256 input utterances and neuron/word dropout probabilities of 0.2. We only apply neuron dropouts on the transition matrices of all the WFA-ω's in SoPa++. We furthermore apply batch sorting based on input utterance lengths to ensure maximum efficiency when computing pattern/document scores. Based on our own experiments, we observed more stable training with the max-sum semiring compared to the max-product semiring. For simplicity, we therefore only use the max-sum semiring in all of our WFA-ω's. Finally, we train all models for a maximum of 50 epochs with 10 patience epochs for early stopping in case of a performance plateau or performance worsening. To trigger early stopping in the patience framework, we monitor the loss over the validation data set. We run all SoPa++ training experiments on a single NVIDIA GeForce GTX 1080 Ti GPU.

To obtain some variation in our SoPa++ models' performance, we decide to use a grid-search technique while varying the patterns and τ-threshold hyperparameters. We vary the patterns hyperparameters to test SoPa++ models with different numbers of constituent WFA-ω's; namely `6-10_5-10_4-10_3-10`, `6-25_5-25_4-25_3-25` and `6-50_5-50_4-50_3-50`. We correspondingly refer to SoPa++ models with the aforementioned pattern hyperparameters as small, medium and large models respectively. Similarly, we vary the τ-threshold to experiment how this threshold could impact both performance and explainability later on. The exact values which we varied can be found in Table 5. Finally, we repeat all training runs for a total of 10 iterations with different initial random seed values. We perform this step to be able to analyze our final results with standard deviations.

[2] https://pytorch.org/

3.5.2 Evaluation

From the previous section on training SoPa++, we assume that we will have fully trained SoPa++ model variants. In order to evaluate the performance of SoPa++ on the FMTOD data set, we preprocess the FMTOD test data set in the same way as we preprocessed the FMTOD training data sets; naturally without upsampling. We then run the FMTOD test data set through the computational graph of the SoPa++ model and compute the accuracy of the SoPa++ model performances. Finally, we compare the accuracies with the accuracy range of other recent papers on FMTOD as described in Section 3.1.4. If the accuracy range of our SoPa++ model(s) falls into the aforementioned competitive range, we can then conclude that our SoPa++ model performs competitively with other recent studies.

3.6 RQ2: Evaluating explanations by simplification

As mentioned in Definition 6, the purpose of explanations by simplification is to create a less complex proxy model which can both keep a similar performance score and maximize its resemblance to the antecedent model. We already discussed how the RE proxy derived from SoPa++ is likely to be a transparent model compared to the black-box SoPa++ model; which already satisfies the important criterion that the proxy model should be less complex. We discuss more regarding the actual vs. theoretical transparency of the RE proxy models in the Discussions chapter.

Another key factor related to explanations by simplification is that the proxy model should be more explainable compared to its antecedent model. We explore this improvement of explainability using the three criteria for "good" explainability techniques as mentioned in Section 2.1.6. An evaluation based on target audiences is also important for explainability; with the target audience of SoPa++ and its RE proxy model largely being experienced ML practitioners. Due to limitations of this study, we are unable to address a detailed survey of explainability on our target audience and instead leave this aspect to future work.

We now describe techniques of evaluating explanations by simplification which address the performance scores and resemblance portions of Definition 6. We use the FMTOD test set to do this since the test set represents unseen data for both SoPa++ and the RE proxy, and therefore could provide a more unbiased representation of these models' performance. Furthermore, we conduct this analysis on our small, medium and large models with all available τ-thresholds to obtain a variety of results for comparison.

3.6.1 Performance score

As previously mentioned, an important aspect of evaluating explanations by simplification is to ensure that the proxy and antecedent models both have similar performance scores on a given task. Given this, we evaluate the accuracy of both SoPa++ models and their RE proxy counterparts on the FMTOD test set and compare their accuracies.

3.6.2 Distance metrics

As previously mentioned, another criteria for effective explanations by simplification is to maximum resemblance between the proxy and antecedent models. While comparing the performance scores is one way of doing this, this can also be seen

as an aggregrate output-oriented comparison. We make an argument to test other distance metrics between SoPa++ and corresponding RE proxy model pairs. We describe these distance metrics and their mathematical formulations in this section.

Softmax distance norm: One method of quantifying a distance between the SoPa++ and RE proxy model pairs is to run both of them on the FMTOD test data set and obtain the softmax scores on each sample. Next, we can find the Euclidean norm of the difference of these two softmax vectors. A low softmax Euclidean distance norm would imply that the softmax distributions are similar and would imply, to a certain extent, that the models conducted similar weightings of input features. This method is not a perfect indicator of interpreting distances between models; but it is more refined in that it allows us to analyze continuous vector differences. We provide a mathematical formulation of the softmax distance norm $\delta(\sigma)$ below for a single sample where $\sigma_{\mathcal{S}}$ and $\sigma_{\mathcal{R}}$ represent the softmax vectors for the SoPa++ and RE proxy model on the given sample respectively and n represents the number of classes in the FMTOD data set:

$$\delta(\sigma) = \|\sigma_{\mathcal{S}} - \sigma_{\mathcal{R}}\|_2 = \sqrt{\sum_{i=1}^{n}(\sigma_{\mathcal{S}_i} - \sigma_{\mathcal{R}_i})^2} \tag{12}$$

Binary misalignment rate: Another technique of measuring the distance between the SoPa++ and RE proxy model would be to compare their internal representations activated in the TauSTE binary layer. This can be computed by calculating the 1-norm between the binary vectors and then normalizing this norm by the dimensionality of this vector. This would provide a mean binary misalignment rate $\delta(b)$, where $b_{\mathcal{S}}$ and $b_{\mathcal{R}}$ represent the TauSTE binary vectors for the SoPa++ and RE proxy model respectively on a given sample:

$$\delta(b) = \frac{\|b_{\mathcal{S}} - b_{\mathcal{R}}\|_1}{dim(b_{\mathcal{S}} - b_{\mathcal{R}})} = \frac{\sum_{i=1}^{n}|b_{\mathcal{S}_i} - b_{\mathcal{R}_i}|}{dim(b_{\mathcal{S}} - b_{\mathcal{R}})} \tag{13}$$

Bibliography

Arrieta, Alejandro Barredo, Natalia Díaz-Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador García, Sergio Gil-López, Daniel Molina, Richard Benjamins, et al. (2020). "Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI". In: *Information Fusion* 58, pp. 82–115.

Ba, Jimmy Lei, Jamie Ryan Kiros, and Geoffrey E. Hinton (2016). *Layer Normalization*. arXiv: 1607.06450 [stat.ML].

Bastani, Osbert, Carolyn Kim, and Hamsa Bastani (2017). "Interpretability via Model Extraction". In: *CoRR* abs/1706.09773. arXiv: 1706.09773. URL: http://arxiv.org/abs/1706.09773.

Baum, Leonard E and Ted Petrie (1966). "Statistical inference for probabilistic functions of finite state Markov chains". In: *The annals of mathematical statistics* 37.6, pp. 1554–1563.

Bengio, Yoshua, Nicholas Léonard, and Aaron C. Courville (2013). "Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation". In: *CoRR* abs/1308.3432. arXiv: 1308.3432. URL: http://arxiv.org/abs/1308.3432.

Bird, Steven and Edward Loper (July 2004). "NLTK: The Natural Language Toolkit". In: *Proceedings of the ACL Interactive Poster and Demonstration Sessions*. Barcelona, Spain: Association for Computational Linguistics, pp. 214–217. URL: https://www.aclweb.org/anthology/P04-3031.

Courbariaux, Matthieu and Yoshua Bengio (2016). "BinaryNet: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1". In: *CoRR* abs/1602.02830. arXiv: 1602.02830. URL: http://arxiv.org/abs/1602.02830.

Danilevsky, Marina, Kun Qian, Ranit Aharonov, Yannis Katsis, Ban Kawas, and Prithviraj Sen (Dec. 2020). "A Survey of the State of Explainable AI for Natural Language Processing". In: *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing*. Suzhou, China: Association for Computational Linguistics, pp. 447–459. URL: https://www.aclweb.org/anthology/2020.aacl-main.46.

Doran, Derek, Sarah Schulz, and Tarek R. Besold (2017). "What Does Explainable AI Really Mean? A New Conceptualization of Perspectives". In: *CoRR* abs/1710.00794. arXiv: 1710.00794. URL: http://arxiv.org/abs/1710.00794.

Eisner, Jason (2002). "Parameter estimation for probabilistic finite-state transducers". In: *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pp. 1–8.

Jiang, Chengyue, Yinggong Zhao, Shanbo Chu, Libin Shen, and Kewei Tu (2020). "Cold-start and Interpretability: Turning Regular Expressions into Trainable Recurrent Neural Networks". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 3193–3207.

Kingma, Diederik P. and Jimmy Ba (2015). "Adam: A Method for Stochastic Optimization". In: *3rd International Conference on Learning Representations, ICLR 2015,*

San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings. Ed. by Yoshua Bengio and Yann LeCun. URL: http://arxiv.org/abs/1412.6980.

Konig, Rikard, Ulf Johansson, and Lars Niklasson (2008). "G-REX: A versatile framework for evolutionary data mining". In: *2008 IEEE International Conference on Data Mining Workshops*. IEEE, pp. 971–974.

Kuich, Werner and Arto Salomaa (1986). "Linear Algebra". In: *Semirings, automata, languages*. Springer, pp. 5–103.

Lundberg, Scott M. and Su-In Lee (2017). "A Unified Approach to Interpreting Model Predictions". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Long Beach, California, USA: Curran Associates Inc., 4768–4777. ISBN: 9781510860964.

Maletti, Andreas (2017). "Survey: Finite-state technology in natural language processing". In: *Theoretical Computer Science* 679, pp. 2–17.

McNaughton, Robert and Hisao Yamada (1960). "Regular expressions and state graphs for automata". In: *IRE transactions on Electronic Computers* 1, pp. 39–47.

Miller, Tim (2019). "Explanation in artificial intelligence: Insights from the social sciences". In: *Artificial Intelligence* 267, pp. 1–38. ISSN: 0004-3702. DOI: https://doi.org/10.1016/j.artint.2018.07.007. URL: https://www.sciencedirect.com/science/article/pii/S0004370218305988.

Peng, Hao, Roy Schwartz, Sam Thomson, and Noah A. Smith (2018). "Rational Recurrences". In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Brussels, Belgium: Association for Computational Linguistics, pp. 1203–1214. DOI: 10.18653/v1/D18-1152. URL: https://www.aclweb.org/anthology/D18-1152.

Pennington, Jeffrey, Richard Socher, and Christopher D Manning (2014). "Glove: Global vectors for word representation". In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1532–1543.

Peñarrubia, Jorge, Yin-Zhe Ma, Matthew G. Walker, and Alan McConnachie (July 2014). "A dynamical model of the local cosmic expansion". In: *Monthly Notices of the Royal Astronomical Society* 443.3, pp. 2204–2222. ISSN: 0035-8711. DOI: 10.1093/mnras/stu879. eprint: https://academic.oup.com/mnras/article-pdf/443/3/2204/4932191/stu879.pdf. URL: https://doi.org/10.1093/mnras/stu879.

Ribeiro, Marco Tulio, Sameer Singh, and Carlos Guestrin (2016). ""Why Should I Trust You?": Explaining the Predictions of Any Classifier". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pp. 1135–1144.

Sanh, Victor, Lysandre Debut, Julien Chaumond, and Thomas Wolf (2019). "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter". In: *NeurIPS EMC^2 Workshop*.

Schuster, Sebastian, Sonal Gupta, Rushin Shah, and Mike Lewis (June 2019). "Cross-lingual Transfer Learning for Multilingual Task Oriented Dialog". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, pp. 3795–3805. DOI: 10.18653/v1/N19-1380. URL: https://www.aclweb.org/anthology/N19-1380.

Schwartz, Roy, Sam Thomson, and Noah A. Smith (July 2018). "Bridging CNNs, RNNs, and Weighted Finite-State Machines". In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Melbourne, Australia: Association for Computational Linguistics, pp. 295–305.

DOI: 10.18653/v1/P18-1028. URL: https://www.aclweb.org/anthology/P18-1028.

Sipser, Michael (1996). "Introduction to the Theory of Computation". In: *ACM Sigact News* 27.1, pp. 27–29.

Suresh, Ananda Theertha, Brian Roark, Michael Riley, and Vlad Schogol (Sept. 2019). "Distilling weighted finite automata from arbitrary probabilistic models". In: *Proceedings of the 14th International Conference on Finite-State Methods and Natural Language Processing*. Dresden, Germany: Association for Computational Linguistics, pp. 87–97. DOI: 10.18653/v1/W19-3112. URL: https://www.aclweb.org/anthology/W19-3112.

Tan, Sarah, Rich Caruana, Giles Hooker, and Yin Lou (2018). "Distill-and-compare: Auditing black-box models using transparent model distillation". In: *Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society*, pp. 303–310.

Thompson, Ken (1968). "Programming techniques: Regular expression search algorithm". In: *Communications of the ACM* 11.6, pp. 419–422.

Townsend, Joseph, Thomas Chaton, and João M Monteiro (2019). "Extracting relational explanations from deep neural networks: A survey from a neural-symbolic perspective". In: *IEEE transactions on neural networks and learning systems* 31.9, pp. 3456–3470.

Tsuruoka, Yoshimasa, Jun'ichi Tsujii, and Sophia Ananiadou (2009). "Fast full parsing by linear-chain conditional random fields". In: *Proceedings of the 12th Conference of the European Chapter of the ACL (EACL 2009)*, pp. 790–798.

Viterbi, Andrew (1967). "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm". In: *IEEE transactions on Information Theory* 13.2, pp. 260–269.

Wang, Cheng and Mathias Niepert (2019). "State-Regularized Recurrent Neural Networks". In: ed. by Kamalika Chaudhuri and Ruslan Salakhutdinov. Vol. 97. Proceedings of Machine Learning Research. Long Beach, California, USA: PMLR, pp. 6596–6606. URL: http://proceedings.mlr.press/v97/wang19j.html.

Yin, Penghang, Jiancheng Lyu, Shuai Zhang, Stanley J. Osher, Yingyong Qi, and Jack Xin (2019). "Understanding Straight-Through Estimator in Training Activation Quantized Neural Nets". In: *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=Skh4jRcKQ.

Zeiler, Matthew D and Rob Fergus (2014). "Visualizing and understanding convolutional networks". In: *European conference on computer vision*. Springer, pp. 818–833.

Zhang, Li, Qing Lyu, and Chris Callison-Burch (Dec. 2020). "Intent Detection with WikiHow". In: *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing*. Suzhou, China: Association for Computational Linguistics, pp. 328–333. URL: https://www.aclweb.org/anthology/2020.aacl-main.35.

Zhang, Zhichang, Zhenwen Zhang, Haoyuan Chen, and Zhiman Zhang (2019). "A joint learning framework with bert for spoken language understanding". In: *IEEE Access* 7, pp. 168849–168858.