Journal of Machine Learning Research 1 (2021) 1-48 Submitted 4/00; Published 10/00

Interpretable Text Classification Via Prototype Trajectory

Dat Hong DAT-HONG@UIOWA.EDU
Department of Computer Science
University of Iowa
Iowa City, IA, USA

Stephen Baek BAEK@VIRGINIA.EDU
School of Data Science
University of Virginia
Charlottesville, VA, USA

Tong Wang* TONG-WANG@UIOWA.EDU
Tippie College of Business
University of Iowa
Iowa City, IA, USA

Editor:

Abstract

We propose a novel interpretable deep neural network for text classification, called ProtoryNet, based on a new concept of prototype trajectories. Motivated by the prototype theory in modern linguistics, ProtoryNet makes a prediction by finding the most similar prototype for each sentence in a text sequence and feeding an RNN backbone with the proximity of each sentence to the corresponding active prototype. The RNN backbone then captures the temporal pattern of the prototypes, which we refer to as *prototype trajectories*. Prototype trajectories enable intuitive and fine-grained interpretation of the reasoning process of the RNN model, in resemblance to how humans analyze texts. We also design a prototype pruning procedure to reduce the total number of prototypes used by the model for better interpretability. Experiments on multiple public data sets show that ProtoryNet is more accurate than the baseline prototype-based deep neural net and reduces the performance gap compared to state-of-the-art black-box models. In addition, after prototype pruning, the resulting ProtoryNet models only need less than or around 20 prototypes for all datasets, which significantly benefits interpretability. Furthermore, we report a survey result indicating that human users find ProtoryNet more intuitive and easier to understand than other prototype-based methods.

1. Introduction

Deep neural networks have become widely adopted for numerous tasks involving unstructured data, such as texts. State-of-the-art deep neural networks for text data include recurrent neural network based models with attention mechanism (Wang et al., 2016; Galassi et al., 2020), convolutional neural networks (Yin et al., 2017; Young et al., 2018), or Transformers (Devlin et al., 2018; Liu et al., 2019). Despite the good predictive performance, more and more real-world applications also desire AI models to be interpretable, such that end-users can understand the rationale in the decision-making process in order to trust, adopt, and work with the model. However, in their conventional form, deep neural networks are black-boxes, where features undergo multiple layers of non-linear transformation, which quickly become intractable and incomprehensible to users.

The black-box nature of existing DNNs for text data has motivated a body of research aiming to achieve model *interpretability.* This line of research can be categorized into two broad directions. One popular group of approaches is to generate *post-hoc* explanations (Jacovi et al., 2018). Yet, they suffer from fundamental limitations for post-hoc explanations in general. As pointed out by recent research (Rudin, 2019; Alvarez-Melis and Jaakkola, 2018), there may exist inconsistency and unfaithfulness in the explanations, since explainer methods only try to approximate the decision-making process, but they are not the real decision-maker. Another type of approach to understanding the inner workings in deep neural networks is to leverage certain architecture designs such as *attention-based* methods. The attention-based approaches (Karpathy et al., 2015; Strobelt et al., 2017; Choi et al., 2016; Guo et al., 2018) weigh the importance of each hidden state in a sequence. However, while a few of them could be expository, the attention weights are, in general, not always intelligible, as pointed out by Jain and Wallace (2019). Furthermore, analyzing attention weights requires a certain level of understanding of how RNNs work in theory. Hence, novice/non-technical users may find it difficult to understand, and, thus, the broader use in real-world applications might not be so feasible.

Recent efforts have been invested in redesigning neural networks towards making them *inherently interpretable*, based on the classic framework of prototypical learning (Datta and Kibler, 1995). These models use prototypes to explain a decision in a more intuitive fashion, where a prototype is often a representative case from previous observations. The process is analogous to how, for example, human experts (*e.g.* doctors or judges) make decisions on a new case by referring to similar previous cases and deducing a decision based on them. From the interpretability standpoint, such prototypes provide an intuitive explanation of how the model has reached a conclusion in a form that even a layperson can understand, as long as they understand the similarity by reading the prototypes. Existing prototype-based models all adopt this reasoning logic (Chen et al., 2019; Ming et al., 2019; Arık and Pfister, 2020). For instance, ProSeNet (Ming et al., 2019) predicts a review to be positive because it is similar to some other reviews from the training data, which are also positive, and the total score is a linear summation of contributions from these prototypes.

In this paper, we identify two designs in existing prototype-based models that are not so suitable for text data. First, existing prototype-based DNN models define prototypes at the document level (Ming et al., 2019; Arık and Pfister, 2020) and decompose a prediction into contributions from each prototype. However, when the input text is long, it becomes difficult to relate the input document to prototypes given the possible complexity of the document, which may include twists, changes of tones, etc. For example, if the input is as simple as "The food is very delicious!", it is easy for a user to understand why it is similar to the prototype "Great food!". But if the input consists of 10 sentences which first talks about the long wait at the restaurant, and proceeds to compliment on the food, but then complains about the rude waiter, and finally concludes that the overall experience was not worth the money spent, it is then difficult for a user to understand why this input is similar to a particular set of prototypes that also talk about several things at the same time. The complexity of understanding the rationale increases quickly as the text becomes longer. In

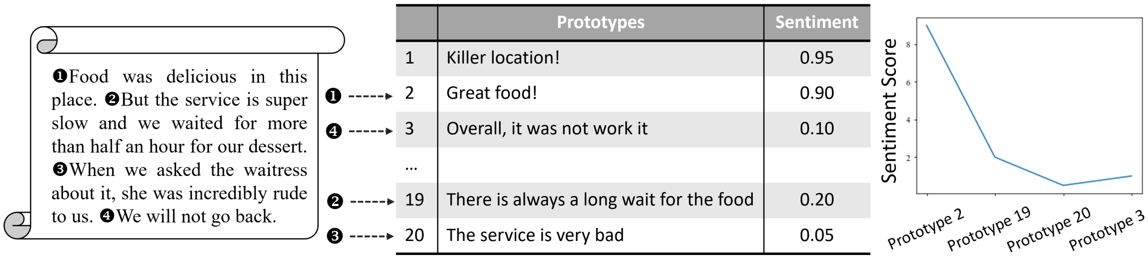


Figure 1: Prototype trajectory-based explanation.

addition, text data are sequences, which naturally allow dynamics of sentiments throughout the documents. But when prototypes are defined at the document-level, such finer-grained understanding is not possible and it is difficult for users to relate sentiments to individual sentences. Second, existing methods generate a large number of prototypes, which is difficult for end-users to comprehend. For example, a ProSeNet model (Ming et al., 2019) needs to use hundreds of prototypes to achieve reliable performance. The ProtoAttend Arık and Pfister (2020) adds sparsity regularization in the model design. But while the number of prototypes involved in each prediction is small, the total number of prototypes the model generates, used by different inputs, is still large, since prototypes are defined on the entire training set. This means users still need to examine $\Omega(|\mathcal{D}|)$ prototypes ($\mathcal{D}$ is the training set) when making predictions.

To improve from the two aspects above, we design a new type of prototype-based DNN model, which makes the reasoning process more suitable for text data and uses much fewer prototypes in total. See Figure 1 for an example. Each sentence is mapped to only one prototype. Thus we can relate the the sentences to the corresponding sentiments obtained by the model, generating a trajectory of prototypes as well as a trajectory of sentiments. We explain the motivations of the designs below.

Motivated by the nature of text data being sequences, we propose a new concept, *prototype trajectory*, that defines prototypes at the sentence level. The prototype proximity values are then fed into an RNN backbone, which then captures latent patterns across sentences via the trajectory of prototypes. Prototype trajectories, therefore, illuminate the semantic structure of a text sequence and the logical flow therein and, hence, provides a highly intuitive and useful interpretation of how the model has predicted an output. In fact, the prototype theory in modern linguistics supplies a strong justification for the proposed idea. In the prototype theory, linguists view "a sentence as the smallest linguistic unit that can be used to perform a complete action" (Alston, 1964) and analyze texts with individual sentences as building blocks. Linguists assume that the sentences of a category are distributed along a continuum: at one extreme of this continuum are sentences having a maximal number of common properties; while on the other extreme are sentences that have only one or very few of such properties (Panther and Köpcke, 2008). Here, the "ideal" sentence that possesses the maximally shared common properties can be considered as a *prototypical sentence* or a *prototype* of the category. Thus, in some sense, this paper takes a meaningful first step towards mathematically formalizing the prototype theory in modern linguistics and its analysis methods by incorporating the above view in a computational framework and emulating how linguists analyze a text.

In addition, to reduce the number of prototypes used for each prediction and the total number of prototypes generated by the model, we add two designs to the model. First, each sentence in an input document is mapped to only one prototype, which we call *active prototype* for that sentence. This design greatly reduces the complexity in the explanations since only T prototypes are used in explaining a prediction, where T is the number of sentences in the document. Thus an input document can be represented by a sequence of prototypes. The idea bears similarity to the "winner-take-all" mechanism in competitive learning (Rumelhart and Zipser, 1985; Chung and Lee, 1994), where a fundamental module in these neural net models involves taking an input computing its similarities to a collection of prototypes, and then selecting only the most similar prototype to "activate". Our experiments show using one active prototype for each sentence performs similarly to using all prototypes, with approximately a 1% drop in accuracy, while greatly benefiting the ease of understandability. Second, to reduce the total number of prototypes used by the model, we introduce *prototype pruning* in our proposed model, which prunes away prototypes that are never or rarely mapped to, and then retrain the model with the remaining prototypes. Our experiments find that even when the model initializes with 200 prototypes, we end up pruning the majority of them without hurting the predictive performance at all. For all datasets we use, our model use about 20 prototypes while achieving the same predictive performance as using 200 prototypes.

With our design, ProtoryNet permits a fine-grained understanding of sequence data alongside an intuitive explanation of the dynamics of the subsequences while being simpler to understand than baselines. Since the technical details are hidden in the prototypes, a non-technical user can easily comprehend the interpretation.

The rest of the paper is organized as follows. Section 2 discusses related work, and in particular, compares ProtoryNet with the closest prototype-based DNN for text classification. Section 3 presents the architecture of ProtoryNet model and Section 4 describes the training procedure. We show detailed experimental results in Section 5 and human subjects evaluation of the interpretability of ProtoryNet in Section 6 while comparing with another interpretable baseline. Finally, Section 7 concludes the paper and discusses possible future directions.

2. Related Work

We first review post-hoc explanation methods and attention mechanisms for explaining DNN models, and then we discuss prototype-based DNN in depth and compare it with our proposed model.

Post-hoc Explanatoins and Attention Based Methods Various post hoc explanation methods have been proposed for explaining DNN models, such as Integrated Gradients (Sundararajan et al., 2017), DeepLift (Shrikumar et al., 2017), NeuroX (Dalvi et al., 2019). Specifically, to understand RNN models, Tsang et al. (2018) proposes a hierarchical explanation for neural networks to capture interactions, and Jin et al. (2019) adapts the idea to text classification to quantify the importance of each word and phrase. For sentiment analysis, Murdoch et al. (2018) proposes a contextual decomposition method for analyzing individual predictions made by LSTMs, which identify words and phrases of contrasting sentiment and how they are combined to yield the LSTM's final prediction. In addition to the external explanation methods, many have considered attention-based approaches interpretable. For example, Bahdanau et al. (2014) implemented an attention mechanism in a decoder, which weighs which part of the source sentence the model needs to pay attention to. Similarly, Rocktäschel et al. (2015) analyzed word-to-word attention weights for achieving insights into how a long short term memory (LSTM) classifier reasons about entailment. Similar strategies can be found in a number of other works (*e.g.* Ismail et al. (2019); Choi et al. (2016)). However, recent research has found attention-based methods controversial and many works believe they are not explanations (Jain and Wallace, 2019). In addition, the attention-based approaches are mostly intended for expert users. Many non-technical users in the real world, who lack basic knowledge of how RNNs work (or even neural networks in general), may find them difficult to understand.

Prototype-based DNN Prototype-based approaches argue that the intuitiveness of interpretation can be significantly enhanced by visualizing the reasoning process in terms of prototypes. In fact, prototype-based reasoning has a long history as a fundamental interpretability mechanism in traditional models (Cupello and Mishelevich, 1988; Fikes and Kehler, 1985; Kim et al., 2014). One of the first work that introduces prototypical learning into a deep neural network is Chen et al. (2019), which designed a new neural network architecture for image classification. A prototype layer was added after convolutional layers to compare the convolution responses at different locations with prototypes. From this, users can understand, for example, a bird is classified as a 'red-bellied woodpecker' because it has the typical prominent red tint at the belly and the top of its head, as well as the black and white stripes on its wings.

We discuss two prototype-based DNN for text classification. The first model is ProSeNet (Ming et al., 2019), which first uses a sequence encoder to obtain a representation of an input sequence, then uses a prototype layer similar to the one in Chen et al. (2019) to compare it with a set of prototypes. ProSeNet computes the similarities between an input sequence (usually a short prose) and prototypes and produces the final prediction as a linear combination of the similarities. Another more recent work is ProtoAttend (Arık and Pfister, 2020) that can work with image, text, and tabular data. ProtoAttend utilizes an attention mechanism to select prototypes, and it allows interpreting the contribution of each prototype via the attention outputs. Similar to ProSeNet, ProtoAttend also relates an input to a linear combination of multiple prototypes.

Issues We Aim to Solve Two potential issues might arise in practice for the two models above. First, the prototypes are defined at the document level, therefore when the text is too long, it will be difficult to represent the input with a prototype, and it will be difficult to convince users of their similarity. The original paper of ProSeNet (Ming et al., 2019) validates ProSeNet only on paragraphs shorter than 25 words. However, it is easily fathomable that ProSeNet may fail to assimilate long paragraph data due to large degrees of freedom that complicate matching of a prototypical example, as validated in our experiments. This may render some practical concerns. For instance, in sentiment classification, even if a paragraph is classified as "negative," it could consist of several twists of sentiments along with sentences (*e.g.*, sarcastic use of positive proses). With an increased length, such kinds of twists can get harder to be represented with a prototype, thus making the interpretation difficult and the explanation less credible. This claim is further supported by findings in modern linguistics, which suggests that sentences, instead of paragraphs, should be regarded as the basic elements for text analysis (Panther and Köpcke, 2008). A second potential issue is the number of prototypes produced, which determines the complexity of the explanations. ProSeNet needs to use K prototypes to explain a prediction, and according to the original paper (Ming et al., 2019), K is at the scale of hundreds. ProtoAttend attends to this issue by including a sparsity regularization in the form of entropy in the training objective. This will make sure there are only a few active prototypes for each prediction. However, the total number of prototypes the model needs to store is at the scale of the size of the training data, which means human users may still need to manually validate and understand all these prototypes when using the model.

ProtoryNet solves the first issue by defining the prototype at the sentence level and solves the second issue by designing specific training objectives and prototype pruning procedures, which will be presented in detail in the next section.

3. ProtoryNet

We present the architecture of ProtoryNet, describe components in the model and then formulate the learning objective.

3.1 The ProtoryNet Architecture

Suppose we work with a data set $\mathcal{D} = \left\{(\mathbf{X}^{(i)}, \mathbf{y}^{(i)}) : i = 1, \ldots, N\right\}$ of size N, comprised of text sequences $\mathbf{X}^{(i)}$ and the corresponding labels $\mathbf{y}^{(i)}$. Here, note that the superscript (i) may be dropped for notational convenience hereinafter, unless necessary. Each instance $\mathbf{X}$ can be understood as a sequence of sentences $\mathbf{x}_t \in \mathbb{R}^V$ at t-th position, yielding the representation $\mathbf{X} = (\mathbf{x}_t)_{t=1}^T$, where V is the size of vocabulary and $T := |\mathbf{X}|$ is the number of sentences in the sequence $\mathbf{X}$. $\mathbf{y} \in \mathbb{R}^C$ is a multi-hot encoded vector representing the class labels associated with the sequence $\mathbf{X}$, *i.e.*, the c-th element y_c of $\mathbf{y}$ equals 1 if the label c is associated with $\mathbf{X}$ or 0 otherwise. C is the total number of classes.

$X^{(i)}$

$y^{(i)}$

$x_t \in R^V$ at t-th

ProtoryNet interfaces with text data via a sentence encoder (Figure 2a) modeled as a mapping $r : \mathbb{R}^V \to \mathbb{R}^J$, where J is the dimension of sentence encoding specified by the user. That is, the encoder takes each sentence $\mathbf{x}_t \in \mathbf{X}$ and produces a sentence embedding:

$$\textbf{Sequence Encoder Layer} \quad : \quad \mathbf{e}_t = r(\mathbf{x}_t). \tag{1}$$

The development of the encoder $r(\cdot)$ is beyond the scope of this paper and, hence, we employ a state-of-the-art Transformer encoder, Google Universal Encoder (Cer et al., 2018), where $J = 512$ by default. The encoder layer may or may not be fine-tuned, which will have an impact on the predictive performance. For now, we defer the discussion to Section 5.1.

The sentence embeddings $\mathbf{e}_t$ are then fed into the *prototype layer* (Figure 2b), in which a set of trainable prototypes $\mathcal{P} = \left\{\mathbf{p}_k \in \mathbb{R}^J : k = 1, \ldots, K\right\}$ are compared with $\mathbf{e}_t$, where $K := |\mathcal{P}|$ is the number of prototypes specified by the user, and each prototype vector $\mathbf{p}_k$ has dimension of J. Note that prototypes $\mathcal{P}$ are trainable parts of the model. Then, given a distance metric $d : \mathbb{R}^J \to \mathbb{R}^+$, the proximity $s_{t,k}$ of the sentence embedding $\mathbf{e}_t$ to a given

Google Universal Encoder

e_t p_k

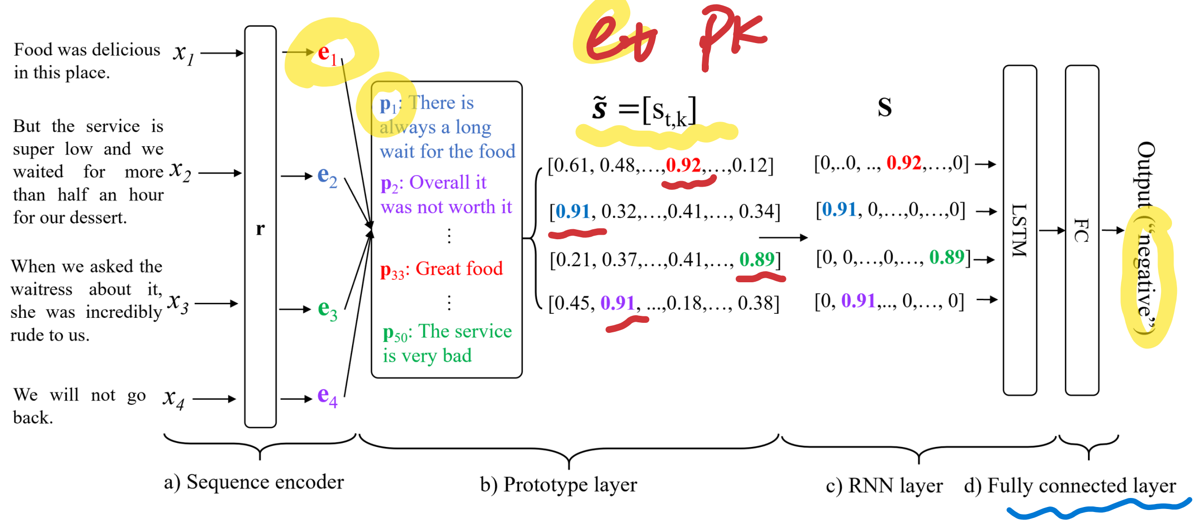


Figure 2: The architecture of ProtoryNet.

prototype $\mathbf{p}_k$ is measured as

$$\textbf{Prototype Layer}: s_{t,k} := \exp\left(-\frac{d(\mathbf{e}_t, \mathbf{p}_k)}{\psi^2}\right), \tag{2}$$

where $\psi \in \mathbb{R}$ is a user-specified constant which we set it to be $\psi^2 = 10$. Between two popular choices for the distance metric $d(\cdot)$, namely the cosine distance and the Euclidean distance, we find that there is no significant difference between the two. Hence, we use the Euclidean distance in our experiments for convenience.

Note that the intermediate throughput of the prototype layer is the similarity matrix $\tilde{\mathbf{S}} = [s_{t,k}]$ of the size $T \times K$, associating the t-th sentence with the k-th prototype. The rows of the similarity matrix $\tilde{\mathbf{S}}$ then constitute the input to the LSTM backbone at time step t (Figure 2c), which then finally produces an output prediction. However, doing so means each sentence is associated with all K prototypes. With the total number of sentences being T, the explanation will then involve $T \cdot K$ prototypes. To ensure better interpretability, we would like to generate easy explanations where each sentence is mapped to only one prototype instead of K prototypes. And we want it to be the most similar prototype to make the explanation more convincing. This means we need set each row in $\tilde{\mathbf{S}}$ to zero except for the position where $s_{t,k}$ is the maximum. That is, each row of the transformed similarity matrix $\mathbf{S}$ would be of the same topology as the one-hot encoded vector, whose elements equal s_{t,k^*} at $k^* := \text{argmax}_k\, s_{t,k}$ and 0 otherwise. For future reference, we denote the most similar prototype for a given sentence the **active prototype**. In this case prototype k^* is the active prototype for the $t-$th sentence.

The sparsity transformation, unfortunately, is not differentiable and may lead to an unexpected training behavior during auto-differentiation in deep learning packages. We get around this issue by the following approximation technique. Suppose the similarity matrix $\tilde{\mathbf{S}} = [\tilde{\mathbf{s}}_1, \ldots, \tilde{\mathbf{s}}_T]^\top$, where $\tilde{\mathbf{s}}_t \in \mathbb{R}^K$ is a row vector whose elements indicate how similar the t-th sentence is to each of the prototypes. If we let Softmax$(\cdot)$ to denote the softmax function, then for some large constant γ,

$$\mathbf{\Gamma} = [\text{Softmax}(\gamma \cdot \tilde{\mathbf{s}}_1), ..., \text{Softmax}(\gamma \cdot \tilde{\mathbf{s}}_T)] \tag{3}$$

approximates the selection matrix whose element equals to 1 at the position corresponding to where $\mathbf{s}_t$ is the maximum for each column t and 0 elsewhere. Here, we find $\gamma \geq 1e^6$ gives a reasonable approximation empirically. With the selection matrix, the sparsity transformation can be approximated as follows without explicitly computing the maximum:

$$\textbf{Sparsity Transformation}: \quad \mathbf{S} \approx \mathbf{\Gamma} \odot \tilde{\mathbf{S}} \tag{4}$$

where $\odot$ is the Hadamard product. Note that the softmax function is differentiable and, thus, is $\mathbf{S}$.

The sparsity transformation of $\tilde{\mathbf{S}}$ to $\mathbf{S}$ enhances the interpretability of the architecture, by enforcing each sentence be matched with the most similar prototype and, thus, disentangling the information. This is accomplished only at a small cost of accuracy, as observed from an ablation study in Section 5.4. Since an input text now can be regarded as a sequence of prototypes, one can think of the matrix $\tilde{\mathbf{S}}$ as a type **prototype encoding** and matrix $\mathbf{S}$ is a sparse prototype encoding. Unlike other sequence encoders (e.g., using embedding techniques) that yield feature vectors that are not sensible to humans, here the prototype encoding return features (i.e., prototype) that are meaningful and easily understandable.

Next, each row of $\mathbf{S}$ is fed into an LSTM model, followed by a few fully connected layers, denoted as,

$$\textbf{RNN Layer}: \quad z = \gamma(\mathbf{S}) \tag{5}$$

$$\textbf{Fully Connected Layer}: \quad \hat{y} = \phi(z), \tag{6}$$

where $\gamma(\cdot)$ represents the LSTM layer and $\phi(\cdot)$ represents a fully connected layer transformation.

Motivating Example We present an example to further demonstrate the model. The text data in Figure 2 exemplifies the use of ProtoryNet for sentiment analysis (text classification). In this example, the task is to predict whether the review of a restaurant is positive or not. The input text data $\mathbf{X}$ is comprised of $T = 4$ sentences, in this particular case, and the label $\mathbf{y}$ is the binary sentiment label of the review, either "positive" ($[1, 0]$) or "negative" ($[0, 1]$). ProtoryNet converts the text data into sentence embeddings, each of which is then matched with the closest prototype. Observe, in the figure, that the prototypes that ProtoryNet produced are, indeed, morphosyntactically equivalent to the corresponding input sentences, well-exemplifying them semantically. The one-hot-like similarity vectors between the sentences and the prototypes are then fed into the LSTM backbone, which captures the patterns and trends in the trajectory of prototypes and, finally, predicts the final sentiment label, which, in this case, is "negative."

3.2 Objective Functions

The training objectives of ProtoryNet entail four different terms aiming to achieve both the prediction accuracy and the interpretability. Below are the details of their definitions.

Accuracy The *accuracy loss* is defined as the mean squared loss between the predicted value and the ground truth label, promoting the model to make accurate predictions:

$$\mathcal{L}_{\text{acc}}(\mathcal{D}) := \frac{1}{N} \sum_{i=1}^{N} \left\| \mathbf{y}^{(i)} - \hat{\mathbf{y}}^{(i)} \right\|^2. \tag{7}$$

Diversity To ensure diverse and non-overlapping prototypes, we define the *diversity loss* term added to enforce the minimum mutual distance δ among the prototypes:

$$d_{\min} := \min_{k_1,k_2} d(\mathbf{p}_{k_1}, \mathbf{p}_{k_2}), \tag{8}$$

$$\mathcal{L}_{\text{div}}(\mathcal{D}) := \sigma\left(\eta(\delta - d_{\min})\right), \tag{9}$$

where $d(\cdot)$ is the Euclidean distance, $\sigma(\cdot)$ is the sigmoid function and η is a smoothing constant, which we set $\eta = 1$ empirically. The constant $\delta \in \mathbb{R}_*^+$ is a positive real number defined by the user, to enforce the minimum separation among prototypes. Hence, when the distances among the prototypes do not meet the minimum separation requirement *i.e.*, $d_{\min} < \delta$, the $\eta(\delta - d_{\min})$ term will have some positive value, making the diversity loss term $\mathcal{L}_{\text{div}}$ active; on the other hand, when the minimum separation requirement is met and thus, $d_{\min} > \delta$, then the sigmoid function will pull the loss term to zero. Note that a smaller η will make such a transition by the sigmoid function smoother.

Prototypicality With only the accuracy and the diversity terms alone, it is observable a prominent tendency of prototypes diverging away from the sentences during training. Such a behavior introduces overfitting, in which prototypes become less generalizable, as the prototypes lose their representativity of a category. In addition, it also hurts the prototypicality of the prototypes since the prototypes are too far away from the sentences to properly represent the stencnes. Hence, we introduce the *prototypicality* loss, which promotes each sentence in the database to have a representative prototype close to it, i.e., we encourage the distance between a sentence and its active prototype to be small:

$$\mathcal{L}_{\text{proto}} = \frac{1}{M} \sum_{\mathbf{X} \in \mathcal{D}} \sum_{\mathbf{x}_t \in \mathbf{X}} \min_k d(\mathbf{r}(\mathbf{x}_t), \mathbf{p}_k), \tag{10}$$

where M is the total number of sentences in the data set.

Final loss The final loss function combines the above loss terms:

$$\mathcal{L} = \mathcal{L}_{\text{acc}} + \alpha \mathcal{L}_{\text{div}} + \beta \mathcal{L}_{\text{proto}} \tag{11}$$

Empirically, coefficients values of $\alpha = 0.1$ and $\beta = 1e^{-4}$ are used by default in this paper except in the sensitivity analysis.

Remarks on Prototype Interpretability The diversity and prototypicality terms are designed for improving the interpretability. Here, to achieve good explanations, prototypes need to be different from each other to avoid redundancy, thus the diversity term. In addition, each input sentence needs to be mapped to a prototype that is similar enough to make the explanation convincing, thus the prototypicality term. These two loss terms can be considered regularization terms to serve interpretability purposes. Similar loss terms have been introduced in other prototype-based DNN models (Ming et al., 2019; Chen et al., 2019). We will later show in experiments that these two terms do not hurt the predictive performance. This can be explained by the recent research on "Rashomon Set" (Semenova et al., 2019; Rudin, 2019), that there exist many models with very similar performance, so one can add customized constraints to the model to achieve additional benefits, such as interpretability.

4. Training a ProtoryNet

For the training of ProtoryNet, the adaptive moment estimation (ADAM) optimizer (Kingma and Ba, 2014) was employed. The learning rate was set to be $1e^{-4}$ and the exponential decay rates for the first and the second moment estimates were 0.9 and 0.999, respectively. Below are further details used for generating the results in this paper.

4.1 Prototype Initialization

The training of ProtoryNet can benefit from the initialization method described below. We first embed all sentences separately in the training data set. Then, in the embedding space, all sentences in the data set are clustered using the k-medoids clustering algorithm to categorize sentences by their semantic meaning. The medoids obtained from the k-medoids algorithm can be considered as the representative examples of each cluster and, hence, plausible candidates for prototypes. Thus, for the training of ProtoryNet, we use these medoids to initialize prototypes, which in turn accelerates the convergence.

4.2 Prototype Projection

It should be noted that the numerical solutions for the prototypes are found in the embedding space. These numerical solutions are not automatically intelligible to human users and need to be deciphered. To this end, we project the prototypes to the closest sentence from the training data in the embedding space every 10 epochs during the training process, similar to the technique proposed in Ming et al. (2019); Chen et al. (2019):

$$\mathbf{s}_k = \underset{\mathbf{x}_t \in X^{(i)}, \forall X^{(i)} \in \mathcal{D}}{\operatorname{argmin}} d(\mathbf{r}(\mathbf{x}_t), \mathbf{p}_k), \qquad k \in [1, K] \tag{12}$$

4.3 Prototype Pruning

In our analysis and experiments, we find that prototypes have significantly different probabilities to be selected (mapped to as the *active prototype*). While the prototypicality term makes sure each sentence is close enough to at least one prototype, we observe that sentences are usually close to a small subset of prototypes, leaving the rest rarely or even never "activated" in inference.

For demonstration, we show an example from the experiment section later in the paper. This model is trained on the Amazon dataset, and the original K is set to 200. We compute the frequencies of prototypes being active for the model trained on the Amazon dataset. Out of 200 prototypes, 92 prototypes have never been mapped to by any sentences in the validation set, which means that these prototypes can already be pruned away without affecting the performance. Then, we plot the frequencies of the remaining 108 prototypes in Figure 3. The prototypes are ranked in descending order of frequencies of being active, i.e., the left-most prototype has the highest frequency: more than 40% sentences are mapped to this prototype, while the right-most prototype has the lowest frequency of less than 0.01%. We observe that the frequencies decay rapidly, indicating that only the top-ranked prototypes are heavily used by the model.

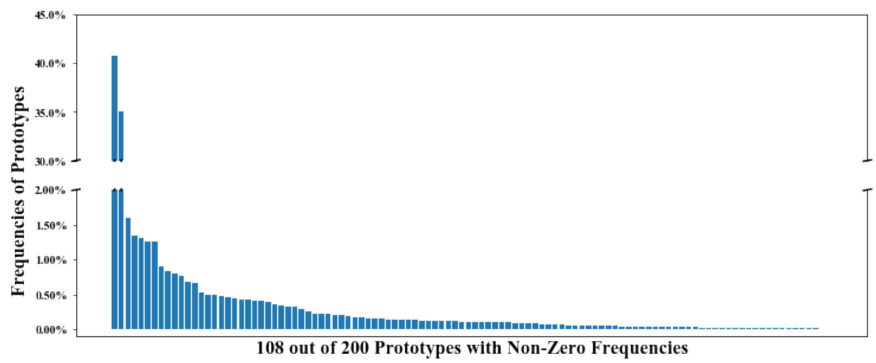


Figure 3: The frequencies of prototypes from a ProtoryNet model trained on Amazon datsaet with K = 200

We observe similar patterns in other datasets where only a subset of prototypes are used, and the remaining are never activated or rarely activated. This is an encouraging observation that justifies the idea of prototype pruning, that in addition to its obvious benefit of improving the model interpretability, prototype pruning seems to reduce the redundancy in the model without hurting the performance. We hypothesize that this is due to the diversity term in the objective, which keeps prototypes distant from each other. Then, when only a few prototypes are sufficient for covering the data space, redundant prototypes are pushed away from all sentences since they need to remain δ away from other prototypes. Because of this, we propose to do prototype pruning, which is to remove these redundant prototypes after the training is complete, based on their frequencies of being active, evaluated on a validation set. If the frequency is smaller than a threshold θ, then the prototypes are removed. The steps are described in line 11 and 12 in Algorithm 1. Let $\mathcal{K}$ represent the indices of remaining prototypes. $\mathcal{K} \subset \{1, 2, \cdots, K\}$ and $|\mathcal{K}| = \hat{K}$. The remaining prototype vectors are $\{\mathbf{p}_k\}_{k\in\mathcal{K}}$. In practice, the threshold θ can be tuned via a validation set.

When implementing the prototype pruning, we build a new ProtoryNet, denoted as $\tilde{f}$. $\tilde{f}$ consists of the same sentence encoder layer $r(\cdot)$ and the $\hat{K}$ prototypes $\{\mathbf{p}_k\}_{k\in\mathcal{K}}$ selected above. We freeze $r(\cdot)$ and $\{\mathbf{p}_k\}_{k\in\mathcal{K}}$ and allow the rest of the layers in $\tilde{f}$ to be trained, i.e., the LSTM layer, denoted as $\tilde{\gamma}(\cdot)$, and fully connected layer, denoted as $\tilde{\phi}$. The steps are described from line 14 to 19 in Algorithm 1.

Sentiment Scores for Prototypes Once the training is done, ProtoryNet returns a set of $\hat{K}$ prototypes and $\hat{K} < K$. We then feed the prototypes back into the trained ProtoryNet one at a time. The outputs from the model are the corresponding sentiment scores of each prototype. These sentiment scores will later be used to provide quantitative visualizations of how the tones and sentiments change within text data.

We summarize the training procedure in Algorithm 1 [1].

Algorithm 1 Training Procedure for ProtoryNet

Input: K, $\mathcal{D}_{\text{train}}, \mathcal{D}_{\text{val}}, \alpha, \beta, \delta, \theta$, FineTuning
Initialization: Build a ProtoryNet $f = \{r, \{\mathbf{p}_1, \cdots, \mathbf{p}_K\}, \text{LSTM-layer}, \phi\}$ and set $r, \mathbf{p}_1, \cdots, \mathbf{p}_K$, LSTM-layer, and ϕ to trainable
\# —— this block trains the model to obtain K prototypes ——
if FineTuning = *FALSE* **then**
  $r(\cdot) \leftarrow$ non-trainable
end if
for $j \leftarrow 0$ to n_{epoch} **do**
  train f with ADAM
end for
\# —— prototype pruning ——
Compute the frequencies of active prototypes using $\mathcal{D}_{\text{val}}$
Select $\{\mathbf{p}_k\}_{k\in\mathcal{K}}$ whose frequencies are larger than θ.
\# —— retrain the model with $\hat{K}$ prototypes fixed ——
Build a new ProtoryNet $\tilde{f} = \{r, \{\mathbf{p}_k\}_{k\in\mathcal{K}}, \tilde{\gamma}, \tilde{\phi}\}$, where the $r(\cdot)$ and $\{\mathbf{p}_k\}_{k\in\mathcal{K}}$ are identical to those in f
$r(\cdot)$, $\{\mathbf{p}_k\}_{k\in\mathcal{K}} \rightarrow$ non-trainable
$\tilde{\gamma}, \tilde{\phi} \rightarrow$ trainable
for $j \leftarrow 0$ to n_{epoch} **do**
  train $\tilde{f}(\cdot)$ using ADAM
end for
Evaluate the sentiment scores for each prototypes $\mathbf{p}_k = r(\mathbf{s}_k), |\mathcal{K}| = \hat{K}$.
$\{\mathbf{s}_k\}_{k=1}^K \leftarrow$ Prototype projection of $\{\mathbf{p}_k\}_{k=1}^K$ using Formula (12)
Return: $\hat{f}(\cdot), \{\mathbf{s}_k\}_{k=1}^K$

5. Experiments

In this section, we evaluate ProtoryNet on six data sets (a detailed description and data preparation for the datasets are included in Section A.1 in the Appendix). Our method is compared against a vanilla LSTM method, an accurate black-box model, DistilBERT Sanh et al. (2019), and a state-of-the-art prototype-based interpretable model, ProSeNet Ming et al. (2019). We also compare our method with a non-neural bag-of-words baseline, which can provide explanations at a word level. See a description of the model setup in Section A.2 in the Appendix.

Our goal is to investigate whether ProtoryNet is comparable to other interpretable baselines and how much accuracy it may lose compared to the black-box model. Then, we analyze the effect of prototype pruning on the prediction performance. In addition, we

1. Code can be found at `https://github.com/dathong/ProtoryNet`

Table 1: Performance of ProtoryNet in comparison with other benchmark models.

Data set	DistilBERT	**ProtoryNet** (Fine-tuned)	**ProtoryNet** (Not fine-tuned)	ProSeNet	Vanilla LSTM	Bag-of-words
IMDB	0.931	0.914	0.871	0.863	0.871	0.877
Amazon Reviews	0.940	0.918	0.890	0.875	0.884	0.830
Yelp Reviews	0.967	0.962	0.941	0.932	0.952	0.908
Rotten Tomatoes	0.903	0.881	0.771	0.869	0.877	0.785
Hotel Reviews	0.976	0.961	0.949	0.930	0.949	0.905
Steam Reviews	0.955	0.924	0.876	0.834	0.864	0.844

provide a detailed example of ProtoryNet using one of the datasets and demonstrate the complexity of sentiment trajectories. Finally, we study the effect of various hyper-parameters in the model.

5.1 Prediction Accuracy

We foremost demonstrate that our inherently-interpretable model design does not cause significant degradation in performance while beating other interpretable baselines

We implement two types of ProtoryNet in the experiments, a **fine-tuned** version where the sentence encoder continues to be trained on the target dataset with the rest of ProtoryNet and a **non-fine-tuned** version where the universal sentence encoder is used as a service but not updated during training. The non-fine-tuned ProtoryNet needs to train significantly fewer coefficients, about 0.03% of the fine-tuned models, thus consuming less energy and computing resources. The goal is to explore 1) the best performance ProtoryNet can achieve, with the help of the state-of-the-art sentence encoder and 2) the more economic solution for uses in resource constrained scenarios.

Here we keep the parameters same for all datasets: $K = 200, \alpha = 0.01, \beta = 1e^{-4}, \delta = 1, \eta = 1$. Our intention is to show that the method is robust enough to solve different text classification problems with varying complexity using one single model architecture and hyperparameters. This way, the ProtoryNet will be practically accountable and easier to use in practice since it does not necessarily need exhaustive tuning of hyper-parameters. We will later explain in Section 5.2 why ProtoryNet is insensitive to K and investigate its sensitivity to α and β in Section 5.4. In addition, we also do not do prototype pruning in this part of the experiment and we will investigate it in detail in Section 5.2.

Reported in Table 1 are performance on the six data sets. First, we acknowledge the performance gap compared to the black-box models. As expected, the black box model (DistilBERT) has the best performance in all data sets used. But still, both versions of ProtoryNet reduced the gap. They both outperform the two interpretable baselines and Vanila LSTM, and if we allow fine-tuning, ProtoryNet becomes even better, with a more significant increase compared to the baselines and only 1.9% away on average from the black-box DistilBERT.

Fine-Tuning vs Non-Fine-Tuning To choose between the fine-tuned and non-fine-tuned ProtoryNet in practice, users need to trade-off between the time and computing resource consumption and the predictive performance. There are more than 256 million parameters in the sentence encoder and only 68 thousand parameters in the rest of ProtoryNet (when $K = 200$), which means that the non-fine-tuned ProtoryNet can be trained only with less than 0.03% of the parameters compared to the fine-tuned version. In addition, training a fine-tuned ProtoryNet takes much longer in time (approximately three times longer on the same Google Colab notebook with a GPU accelerator) than a non-fine-tuned ProtoryNet. In summary, the non-fine-tuned ProtoryNet is much smaller and more energy efficient, while still beating the interpretable baselines. Answering the increasing call for Green-AI Schwartz et al. (2019), non-fine-tuned ProtoryNet will be better than the fine-tuned ProtoryNet when smaller, and lighter models are preferred.

Comparing Short and Long Reviews Between ProSeNet and ProtoryNet, ProtoryNet outperformed ProSeNet for all six cases overall. In particular, the performance difference was clearer when long text data were analyzed. Since the fine-tuned ProtoryNet significantly outperforms ProSeNet, here we only compare ProSeNet with the weaker version of ProtoryNet, the non fine-tuned models. In Table 2, we split each data set into *short* and *long* samples—texts that were less than 25 words were classified as short samples, following the criterium used in the ProSeNet paper (Ming et al., 2019). As shown in the table, ProSeNet was on par or better than ProtoryNet on short texts, while ProtoryNet was better than ProSeNet when long paragraphs were concerned. In fact, this is an advantage of ProtoryNet since long texts (more than 25 words) are more prevalent than short texts in most real-world datasets, as evidenced in Table 2. This also explains why the non-fine-tuned ProtoryNet performs worse than ProSeNet on the Rotten Tomatoes dataset since more than 65% of the reviews are short reviews with less than 25 words.

Table 2: Comparison between ProSeNet and ProtoryNet (non-fine-tuned) on text data of different lengths.

Data set	**% of short reviews**	**ProSeNet**		**ProtoryNet**	
		Short	Long	Short	Long
IMDB	0.17	0.868	0.863	0.868	0.871
Amazon Reviews	6.02	0.908	0.873	0.843	0.893
Yelp Reviews	8.85	0.943	0.931	0.863	0.949
Rotten Tomatoes	65.52	0.875	0.859	0.751	0.809
Hotel Reviews	2.11	1.000	0.928	1.000	0.949
Steam Reviews	23.75	0.791	0.848	0.860	0.881

5.2 Prototype Pruning

In previous experiments, we set K to a fixed number ($K = 200$) for all datasets to show that our method is robust enough to solve different text classification problems using the same parameters. In this section, we apply prototype pruning after a model is trained for the purpose of improving interpretability, since fewer prototypes there are, the easier it is for human users to understand the model and interpret the predictions.

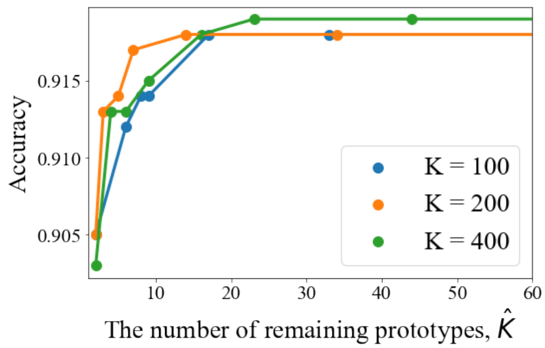


Figure 4: The effect of prototype pruning

Data set	$\hat{K}$
IMDB	8
Amazon Reviews	14
Yelp Reviews	15
Rotten Tomatoes	23
Hotel Reviews	7
Steam Reviews	17

Table 3: Numbers of remaining prototypes without hurting the accuracy.

We perform prototype pruning with varying pruning thresholds to obtain various sizes of remaining prototypes, to analyze how much the pruning impacts the predictive performance. The idea is, after we obtain a set of prototypes, we compute the frequencies of each prototype being mapped to, and then remove prototypes with frequencies lower than a threshold we choose. Then we train a new model with the remaining prototype. We use $\hat{K}$ to represent the number of remaining prototypes. For demonstration, we choose the Amazon dataset, and we set the pruning threshold to $\{0.2\%, 0.5\%, 1\%, 2\%, 5\%, 10\%\}$, which return various models with much fewer prototypes. Their predictive performance and the number of remaining prototypes are reported in Figure 4. Note that, with only 0.5% threshold, 186 prototypes, which is 93% of the total, were pruned. Meanwhile, removing these prototypes did not hurt the predictive performance at all: the accuracy after pruning is still 0.918, the same as when the model has 200 prototypes. This implies that a large number of prototypes are redundant and can be safely removed without hurting the model performance. This brings considerable benefits to interpretability since, after pruning, the model is only left with 14 prototypes with exactly the same accuracy as $K = 200$. This means that in practical uses, a human user, either model designer or end user, can easily check all prototypes to determine whether they make sense, like an example we provide in Section 5.3.

To test whether the observation applies to different initial K, we also set $K = 100$ and $K = 400$ and repeat the experiment. The curves are similar to $K = 200$, that the accuracy does not change even a large number of prototypes are pruned. Only when reaching a certain "tipping point", the performance starts to drop, then pruning hurt the performance. The results imply that the number of necessary prototypes for a given dataset is more or less the same, even provided with a different number of initial prototypes K. The findings above offer meaningful implications for model tuning, that one does not need to tune K heavily: as long as we supply the initial model with a large number of prototypes and let the model achieve the best predictive performance it can obtain, then we can prune the prototypes afterward for better interpretability.

Therefore, we conduct prototype pruning for all fine-tuned ProtoryNet models from Table 1 with $K = 200$ and report the minimum $\hat{K}$ that achieves the same accuracy as $K = 200$. $\hat{K}$ for all models are reported in Table 3. Results show that all datasets only require around 20 prototypes, which indicates a significant improvement in interpretability compared to other prototype-based DNNs, given the complexity of the dataset and task.

Table 4: Prototype information for Yelp review ($\hat{K} = 15$ after pruning).

ID	Prototypes	Sentiment
1	I love this place	0.972
2	The biggest breakfast in Pittsburgh, as far as I can tell - and delicious and cheap too	0.972
3	Went here for lunch yesterday with a friend and it was so yummy	0.968
4	The steak was cooked perfectly and the crabcake was a good size	0.962
5	Papa J's is by far my favorite restaurant in Pittsburgh, my hometown	0.949
6	My aunt insisted that we have lunch at Uno's Pizzeria & Grill as the food was delicious	0.603
7	From the minute we were seated, we were greeted by a server that was clearly inexperienced and didn't know the menu	0.171
8	We ended up spending a fortune on beer and mediocre appetizers	0.074
9	If I want to spend that kind of money, I'll go somewhere that I can get good service	0.028
10	They finally brought my food out and left it without asking for me to pay	0.027
11	The burgers were over cooked and the fries were soggie and the milkshake was runny at best	0.019
12	The waitress told me that the kitchen hadn't even started on my order yet, so I told her to cancel it and walked out	0.017
13	It took forever to order and then forever and the place was empty	0.013
14	I won't be going back	0.011
15	Food was terrible	0.011

Now model designers or users only need to examine the list of prototypes that could easily fit into a piece of paper, like Table 4, to understand or contest the model.

5.3 Prototypes and Prototype Trajectories

In this section, we illustrate an example ProtoryNet trained from the Yelp dataset with prototype pruning. As shown in Table 3, this model ends up with only 15 prototypes while achieving the same accuracy as $K = 200$, i.e., accuracy is 0.962 as reported in Table 1. The prototypes are shown in Table 4, together with their sentiment scores. The prototypes are ranked in the descending order of sentiment scores. The prototypes span all range of sentiments, from the most positive prototype, "I love this place" with a sentiment score of 0.972, to the least positive prototype, "I won't be going back" with a sentiment scores of 0.011. It is worth mentioning that his highly accurate model only needs prototypes that take half of a page and one can easily and quickly go through all of them.

Then, each input text can be represented by a sequence of prototypes selected from Table 4. One can regard the sequence of prototypes as a human-understandable representation of the input text. Unlike other sequence encoders based on embedding techniques where the features are not sensible to humans, here we can consider this prototype trajectory as "prototype encoding" and the features, i.e., prototypes, are easily understandable.

We show two positive examples in Table 5 and two negative examples in Table 6. Each sentence in a text instance is mapped to one of the prototypes from Table 4 as well as the

Table 5: Two Positive Examples

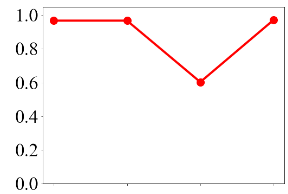
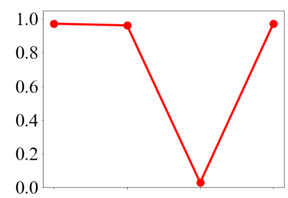
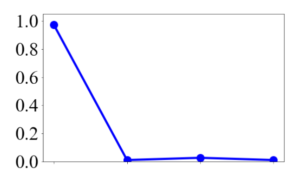
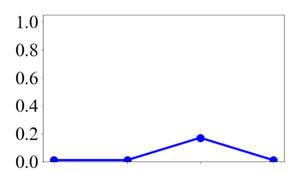
	Input Text	Prototype	Trajectory
Example 1	①This was our first visit to Paradise Bakery and all I can say is YUM	3	
	② It was so good, I went back for lunch the next day	3	
	③ The dining room is very pleasant and clean, the service is great and the sandwiches are super yummy	6	
	④ I love having this fairly close to our house	1	
Example 2	①This place is fantastic	1	
	② Impeccable service, great atmosphere and outstanding food	4	
	③Yes, it's pricey but well worth it.	9	
	④I've been here a couple of times and it never disappoints	1	

Table 6: Two Negative Examples

	Input Text	Prototype	Trajectory
Example 3	①I used to LOVE this place	1	
	② But the service was TERRIBLE	15	
	③The woman was so slow and put her FINGER in my food	10	
	④I won't be coming back	14	
Example 4	①Not good at all	15	
	② Average at best	15	
	③ Table we were sat at was sticky and needed wiping down, had to ask the server twice	7	
	④ Food was ok but not good	15	

corresponding sentiment score, generating a trajectory of prototypes and sentiments. For example, the first sentence in Example 1, "This is our first visit to Paradise Bakery and all I can say is YUM!" is mapped to prototype 3, "Went here for lunch yesterday with a friend and it was so yummy." The corresponding sentiment is 0.968, as shown in the sentiment trajectory in the figure. Note that different sentences in the text input can be mapped to the same prototype, such as the second sentence in Example 1: "It was so good, I went back for lunch the next day", which is also mapped to prototype 3 in Table 4.

We observe that the trajectories can be very different even for the same sentiment class. Example 1 stays positive for the entire review, while Example 2 starts and ends with positive

Table 7: Substituting LSTM with Other Interpretable Models

Data set	Average	Logistic Regression	Decision Tree	ProtoryNet
IMDB	0.602	0.859	0.836	0.914
Amazon Reviews	0.559	0.808	0.878	0.918
Yelp Reviews	0.802	0.825	0.923	0.962
Rotten Tomatoes	0.501	0.671	0.796	0.881
Hotel Reviews	0.896	0.905	0.918	0.961
Steam Reviews	0.771	0.815	0.790	0.924

sentiments but mentions a negative aspect in the middle, that it's pricey. Similarly, the two negative examples also yield different trajectories of sentiments. Example 3 starts with a positive sentiment since the customer "used to LOVE this place", which is mapped to prototype "I love this place" with a sentiment score of 0.972. Then the customer changes his tone and talks about negative aspects of the restaurant, i.e., bad service and unsanitary behavior of the waitress, and ends with a negative sentiment. On the other hand, Example 4 maintains a negative tone for the entire review, which is also reflected by the trajectory of sentiments. As such, interpretation of ProtoryNet can be more fine-grained, generating deeper insights to users.

Users can identify a more subtle sentiment development or change of tones in the text that document-level prototypes cannot achieve. From this, users can extract useful information. For example, by identifying the change of tones in positive reviews, the model indirectly teaches restaurants which aspects they should pay attention to and probably improve in the future. For instance, the sentiment trajectory for Example 3 points out that the price might be a little high, and it is something the restaurant needs to take a look at if they would like to increase customer satisfaction. Such information can potentially be more valuable to restaurants than simply predicting whether a review is positive or negative.

Substituting LSTM with Interpretable Models The previous analysis demonstrates the diversity in the sentiment trajectory, that even the predicted sentiments for the whole reviews are positive (or negative), the trajectories of sentiments could differ greatly from each other. Thus, the trajectory reflects the complexity as well as heterogeneity in the sentiment development along with the text reviews. In the ProtoryNet model, the sentiments are not directly used, but implicitly represented by the prototypes. When generating a prediction, a sequence of similarities to the active prototypes is fed to an LSTM model, which is processed by an LSTM model. The LSTM is used to learn the temporal pattern from the sequence to produce the final output. Note that the LSTM is an essential component since other *interpretable* models cannot remember as LSTM does. To demonstrate the value of LSTM, we conduct a set of experiments where we use an interpretable model to replace LSTM in the final step. The features are sentiments of the active prototypes for each sentence in a review. Since the interpretable models work with panel data, we truncate all reviews to 10 sentences and pad those with fewer sentences with the average sentiments from existing sentences. We experimented with two types of interpretable models, Logistic Regression and Decision Tree. In addition, we calculate the average sentiment of sentences in each review (without padding) and compare it with a threshold to obtain a prediction. Results

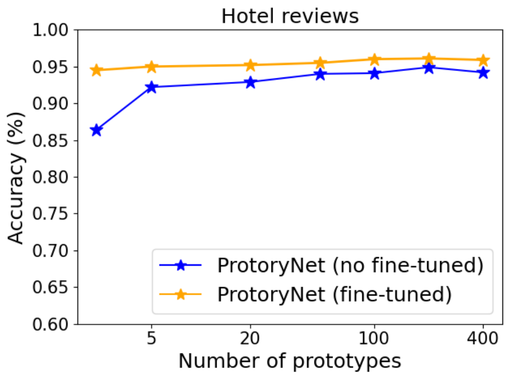


Figure 5: Effect of K on accuracy.

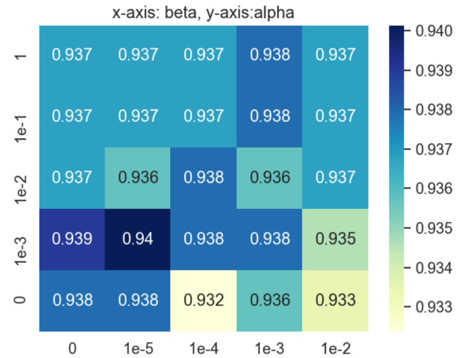


Figure 6: Sensitivity analysis of α and β.

are shown in Table 7 in comparison with ProtoryNet that uses an LSTM to process the sentiment change.

Table 7 shows that the original ProtoryNet using LSTM achieves much better performance than the interpretable baselines. The results prove the necessity of using an LSTM that processes the text as a sequence instead of treating them as a collection of sentiments. This means, not only do the sentiment scores matter, where they appear in the text also matter.

5.4 Ablation and Sensitivity Analysis

In previous experiments, we used the same set of hyper-perameters, to show that our ProtoryNet is easy to use in practice since it does not need heavy tuning and can still achieve reliable performance. In this section, we evaluate the effect of different hyper-parameters on the model performance. We also examine how much the sparsity transformation hurt the predictive performance, which is designed for better interpretability.

Effect of K We investigated how the initial number of prototypes, K, influences the performance of ProtoryNet[2]. In Figure 5, the performance of ProtoryNet on the Hotel Review data set is plotted for different values of K. Other hyperparameters were controlled to be the same. Curves in Figure 5 show that ProtoryNet is not so sensitive to K once K is sufficiently large. This observation can be explained by Table 7, that only a minimal number (about 20) of effective prototypes are actually needed to "cover" the feature space. More prototypes are only redundant for the classification task and can be safely removed. This finding reinforces the insights for parameter tuning: users just need to set K to a large number and then prune it back. For the fine-tuned ProtoryNet, the performance is already very well with a small K. This is because when fine-tuning is allowed, sentences can be moved towards the prototype they are mapped to, thus it does not need many prototypes to cover the whole space. On the other hand, for non-fine-tuned ProtoryNet, each sentence is represented by a fixed vector in the feature space. If there are very few prototypes, it becomes difficult for some sentences to be mapped to the correct prototype since they are far away from all prototypes.

2. Note that K was selected from [5,20,50,100,200,400] via a validation set when producing Table 1.

Effect of Diversity and Prototypicality Terms We performed a sensitivity analysis to understand the effect of the two terms on predictive performance using the Hotel dataset. Since our goal was to study the effect of α and β, we fixed the K to be 100 and tried different combinations of α, β, where $\alpha = 0, 1e^{-3}, 1e^{-2}, 1e^{-1}, 1$, and $\beta = 0, 1e^{-5}, 1e^{-4}, 1e^{-3}, 1e^{-2}$. As seen in Figure 6, our experiment revealed that the ProtoryNet achieves consistently high peformance with different α and β. This benefits the training and tuning process because users do not need to invest tremendous amount on parameter tuning. In this paper, we set $\alpha = 0.1$ and $\beta = 1e^{-4}$ to all experiments. In addition, we notice that the best performance was achieved when α and β are set to small values instead of 0, suggesting the positive impact of the diversity term and protypicality term we designed on the predictive performance. A possible explanation would be that, having some constraints on the prototypes' diversity ($\mathcal{L}_{\text{div}}$) and their representativeness ($\mathcal{L}_{\text{proto}}$) prevents overfitting as these terms "regulate" prototypes.

Effect of Sparsity Transformation Furthermore, we conducted an ablation study on the sparsity transformation. The sparsity transformation from $\tilde{\mathbf{S}}$ to $\mathbf{S}$ was used to enhance the interpretability of the model, which forces each sentence to be mapped to one closest prototype, i.e., the *active prototype*. Without the sparsity transformation, each sentence will be mapped to K prototypes, which will involve $T \cdot K$ prototypes in the explanation for the prediction. Despite the big advantage of sparsity transformation in interpretability, we investigate the impact of this sparsity transformation on predictive performance. We measured the change in prediction accuracy when the sparsity transformation step had been removed, and the dense similarity matrix $\tilde{\mathbf{S}}$ had been used directly. Specifically, we compare fine-tuned ProtoryNet's performance with and without the sparsity transformation and show the comparison in Table 8. As reported in Table 8, there was only a small drop of accuracy (approx. 1%) caused by the sparsity transformation.

Table 8: Performance comparison between non-sparse $\tilde{\mathbf{S}}$ and sparse $\mathbf{S}$ as the input to the LSTM layer. The validation accuracy for each case.

Data set	Dense (K active prototypes)	Sparse (1 active prototype)
IMDB	0.920	0.914
Amazon Reviews	0.921	0.918
Yelp Reviews	0.956	0.954
Rotten Tomatoes	0.896	0.881
Hotel Reviews	0.968	0.961
Steam Reviews	0.936	0.924

6. User Evaluation

The interpretability of ProtoryNet was further validated via a survey conducted on 111 individuals, among which 42 identified themselves as non-technical users. Subjects were recruited through two different channels. Individuals from the authors' home institution holding a master's degree or above having advanced knowledge of RNNs have been recruited as technical users. Non-technical users were recruited from Amazon Mechanical Turk. The survey designs are disclosed in Appendix.

We first evaluated the interpretability of the explanation by testing whether the model-selected prototypes were indeed representative of the input text to the human users. We asked the users to choose the most appropriate prototype for a given sentence out of four options presented to them, one of which was the actual prototype matched by the model, other two were randomly selected from the rest of the prototypes, and the other was "None of the above." We created 10 such questions by sampling reviews from the Yelp Review data set, each for ProtoryNet and ProSeNet. As reported in Figure 7a, ProtoryNet showed a more significant agreement between the model-selected prototype and the prototype that the human users found the most appropriate. For both technical users and non-technical users, ProtoryNet was significantly better than ProSeNet, as was validated by the t-test. The difference between technical users and non-technical users was insignificant, suggesting that non-technical users can comprehend prototypes equally well as technical users.

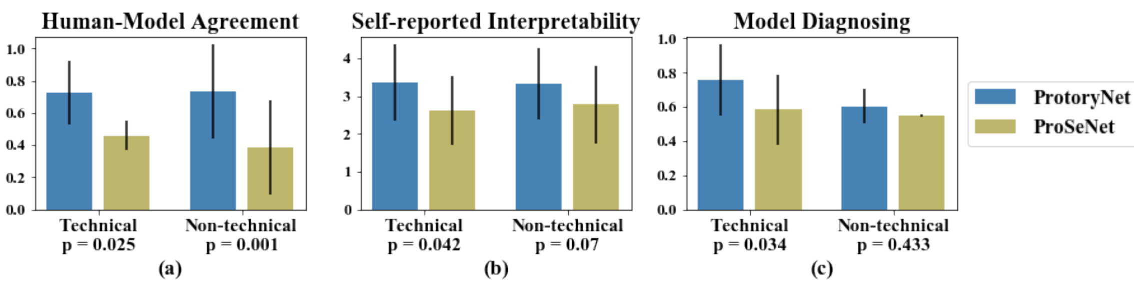


Figure 7: Interpretability of ProtoryNet assessed by human users. The p-values for t-test are evaluated for comparing the responses for ProtoryNet and ProSeNet on technical users and non-technical users, respectively.

The survey also included self-report questions to assess how easy it was for them to select a prototype in a score ranging between 1 (very difficult) and 5 (very easy). As reported in Figure 7b, subjects found that ProtoryNet was easier to interpret in general, and the improvement in interpretaiblity was more significant for technical users.

Finally, we measured how easily the users can learn to interpret the results of ProtoryNet. For this, each subject was randomly assigned to either ProtoryNet or ProSeNet and trained on how the model that they are assigned to makes predictions. Then, their proficiency was measured by showing them three examples on which the model had made an incorrect prediction and asking them to diagnose the problem by pointing out an inappropriately matched prototype. The problematic prototype (*i.e.*, the "correct answer" for the survey question) was determined via a discussion among the authors, which later turned out to be aligned with the consensus in the survey responses as well. As in Figure 7c, the both subject groups were more accurate at diagnosing ProtoryNet in general. An explanation to this should be that ProtoryNet uses shorter prototypes than ProSeNet and, thus, is easier to comprehend. We notice that the while technical users find ProtoryNet easier to debug, such difference was not significant for non-technical users. In fact, there was no significant difference between technical users and non-technical users when they use ProSeNet since it was almost equally difficult to these two groups of users.

7. Discussion and Conclusion

We introduced a novel idea of prototype trajectory in DNNs. Our model, ProtoryNet, maps a text input into a sequence of prototypical sentences, illuminating the underlying dynamics of semantics within the text data. Therefore, Users can identify a more subtle sentiment development or change of tones in text that document-level prototypes cannot achieve. ProtoryNet achieved a predictive performance higher than the state-of-the-art interpretable baselines and reduced the performance gap compared to black-box DNNs. Moreover, the human evaluation result suggested that ProtoryNet provided more intuitive prototypes than the baseline and that the novice users were able to interpret ProtoryNet equally well as the expert users. The prototype pruning we design has proved to be quite effective on all datasets we experimented with and the resulting models only need around 20 prototypes in total, which is a significant improvement compared to other baselines.

Our model has also shown be very easy to use. First, it does not rely on heavy parameter tuning (we used the same set of parameters in all of our experiments), which makes it convenient in practice. In addition, we experimented with two versions of ProtoryNet, a fine-tuned ProtoryNet to fully utilize the power of a state-of-the-art Transformer encoder and a non-fine-tuned ProtoryNet, which is much smaller, lighter, and energy-efficient. Results show that even the non-fine-tuned ProtoryNet already beats the interpretable baselines and can potentially be more promising with the increasing need for Green AI.

The benefit of prototype-based reasoning resides in the fact that it hides technical details by encapsulating them with prototypical examples while being still tractable numerically when desired. Hence, novice users can understand how the reasoning was achieved in RNNs so long as they can comprehend the prototypes, lowering the barrier for those numerous non-technical users who may use RNN-based applications in the real world. On the other hand, numerical weights assigned to prototypes alongside their association with the "nuts and bolts" of RNNs still allow experts to perform in-depth analyses of how a model has drawn a prediction. One can think of the prototypes as a special type of *feature representation* of the original text input. Compared to other types of latent features produced by complicated transformations through encoding layers, where the features are not sensible, the "prototype encoding" in ProtoryNet obtains a human-understandable feature representation. Prototypes make sense to humans while encapsulating all necessary information in them, thus they are able to obtain good predictive performance even using only the LSTM layers to process them.

ProtoryNet can potentially be applied to other sequence data other than text. However, one should be able to define meaningful and consistent sub-sequences, like sentences in a document. This definition is task-specific and may need to conform with application-specific constraints. In addition, one may remove the sentence encoder or replace it with some other feature extractor.

For future work, it would be interesting to mathematically formalize some of the well-established requirements to be a prototype in the linguistics literature. For example, Panther and Köpcke (Panther and Köpcke, 2008) assert several conditions that a prototype must possess—a prototypical sentence is an affirmative declarative sentence; the subject is in the nominative case; the verb in a prototype is in the active voice and in the indicative mood; to list a few. Albeit non-trivial, the mathematical translation of such conditions should bring more interpretability and, perhaps, better performance of ProtoryNet.

References

William P Alston. *Philosophy of language*. Prentice Hall, 1964.

David Alvarez-Melis and Tommi S. Jaakkola. On the robustness of interpretability methods. *ICML Workshop on Human Interpretability in Machine Learning*, 2018.

Sercan O Arık and Tomas Pfister. Protoattend: Attention-based prototypical learning. *Journal of Machine Learning Research*, 21:1–35, 2020.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St John, Noah Constant, Mario Guajardo-Céspedes, Steve Yuan, Chris Tar, et al. Universal sentence encoder. *arXiv preprint arXiv:1803.11175*, 2018.

Chaofan Chen, Oscar Li, Daniel Tao, Alina Barnett, Cynthia Rudin, and Jonathan K Su. This looks like that: Deep learning for interpretable image recognition. In *Advances in Neural Information Processing Systems*, pages 8928–8939, 2019.

Edward Choi, Mohammad Taha Bahadori, Jimeng Sun, Joshua Kulas, Andy Schuetz, and Walter Stewart. Retain: An interpretable predictive model for healthcare using reverse time attention mechanism. In *Advances in Neural Information Processing Systems*, pages 3504–3512, 2016.

Fu Lai Chung and Tong Lee. Fuzzy competitive learning. *Neural Networks*, 7(3):539–551, 1994.

James M Cupello and David J Mishelevich. Managing prototype knowledge/expert system projects. *Communications of the ACM*, 31(5):534–550, 1988.

Fahim Dalvi, Nadir Durrani, Hassan Sajjad, Yonatan Belinkov, Anthony Bau, and James Glass. What is one grain of sand in the desert? analyzing individual neurons in deep nlp models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 6309–6317, 2019.

Piew Datta and Dennis Kibler. Learning prototypical concept descriptions. In *Machine Learning Proceedings 1995*, pages 158–166. Elsevier, 1995.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Richard Fikes and Tom Kehler. The role of frame-based representation in reasoning. *Communications of the ACM*, 28(9):904–920, 1985.

Andrea Galassi, Marco Lippi, and Paolo Torroni. Attention in natural language processing. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

Tian Guo, Tao Lin, and Yao Lu. An interpretable LSTM neural network for autoregressive exogenous model. *arXiv preprint arXiv:1804.05251*, 2018.

Aya Abdelsalam Ismail, Mohamed Gunady, Luiz Pessoa, Hector Corrada Bravo, and Soheil Feizi. Input-cell attention reduces vanishing saliency of recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 10813–10823, 2019.

Alon Jacovi, Oren Sar Shalom, and Yoav Goldberg. Understanding convolutional neural networks for text classification. *arXiv preprint arXiv:1809.08037*, 2018.

Sarthak Jain and Byron C Wallace. Attention is not explanation. *arXiv preprint arXiv:1902.10186*, 2019.

Xisen Jin, Zhongyu Wei, Junyi Du, Xiangyang Xue, and Xiang Ren. Towards hierarchical importance attribution: Explaining compositional semantics for neural sequence models. *arXiv preprint arXiv:1911.06194*, 2019.

Andrej Karpathy, Justin Johnson, and Li Fei-Fei. Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*, 2015.

Been Kim, Cynthia Rudin, and Julie A Shah. The Bayesian case model: A generative approach for case-based reasoning and prototype classification. In *Advances in Neural Information Processing Systems*, pages 1952–1960, 2014.

Diederik P Kingma and Jimmy Ba. ADAM: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

Yao Ming, Panpan Xu, Huamin Qu, and Liu Ren. Interpretable and steerable sequence learning via prototypes. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 903–913, 2019.

W James Murdoch, Peter J Liu, and Bin Yu. Beyond word importance: Contextual decomposition to extract interactions from lstms. *arXiv preprint arXiv:1801.05453*, 2018.

Klaus-Uwe Panther and Klaus-Michael Köpcke. A prototype approach to sentences and sentence types. *Annual Review of Cognitive Linguistics*, 6(1):83–112, 2008.

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

Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using Siamese BERT-networks. *arXiv preprint arXiv:1908.10084*, 2019.

Tim Rocktäschel, Edward Grefenstette, Karl Moritz Hermann, Tomáš Kočiskỳ, and Phil Blunsom. Reasoning about entailment with neural attention. *arXiv preprint arXiv:1509.06664*, 2015.

Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5):206–215, 2019.

David E Rumelhart and David Zipser. Feature discovery by competitive learning. *Cognitive science*, 9(1):75–112, 1985.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. DistilBERT, a distilled version of BERT: Smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

Roy Schwartz, Jesse Dodge, Noah A Smith, and Oren Etzioni. Green ai. *arXiv preprint arXiv:1907.10597*, 2019.

Lesia Semenova, Cynthia Rudin, and Ronald Parr. A study in rashomon curves and volumes: A new perspective on generalization and model simplicity in machine learning. *arXiv preprint arXiv:1908.01755*, 2019.

Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. *arXiv preprint arXiv:1704.02685*, 2017.

Hendrik Strobelt, Sebastian Gehrmann, Hanspeter Pfister, and Alexander M Rush. LSTMVis: A tool for visual analysis of hidden state dynamics in recurrent neural networks. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):667–676, 2017.

Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. *arXiv preprint arXiv:1703.01365*, 2017.

Michael Tsang, Youbang Sun, Dongxu Ren, and Yan Liu. Can i trust you more? model-agnostic hierarchical explanations. *arXiv preprint arXiv:1812.04801*, 2018.

Yequan Wang, Minlie Huang, Xiaoyan Zhu, and Li Zhao. Attention-based lstm for aspect-level sentiment classification. In *Proceedings of the 2016 conference on empirical methods in natural language processing*, pages 606–615, 2016.

Wenpeng Yin, Katharina Kann, Mo Yu, and Hinrich Schütze. Comparative study of cnn and rnn for natural language processing. *arXiv preprint arXiv:1702.01923*, 2017.

Tom Young, Devamanyu Hazarika, Soujanya Poria, and Erik Cambria. Recent trends in deep learning based natural language processing. *ieee Computational intelligenCe magazine*, 13 (3):55–75, 2018.

Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *Advances in Neural Information Processing Systems*, pages 649–657, 2015.

Appendix A. Reproducibility

A.1 Data Sets Description

IMDB Movie Reviews The IMDB Movie Reviews data set is a standard benchmark data set for binary sentiment classification and is available at `https://ai.stanford.edu/~amaas/data/sentiment/`. The data set is perfectly balanced and comprised of 25,000 movie reviews for training and 25,000 for testings and we followed this original partition of the training and testing set and use 10% of the training data as validation.

Yelp Reviews The Yelp Reviews data set was obtained from `http://goo.gl/JyCnZq`. The data set is comprised of 580,000 Yelp review samples and their corresponding labels. The authors of the data set has binarized the sentiment scores by assuming 1 and 2 stars as a negative sentiment and 3 and 4 stars as a positive sentiment. They also already splitted the dataset into a training set with 550,000 reviews and a test set with 30,000 reviews. In this paper, we followed this data partition and partitioned the training set into 90% training and 10% validation.

Amazon Product Reviews Similar to Yelp Reviews dataset, we also obtained Amazon Reviews from `http://goo.gl/JyCnZq`. For this dataset, we took random samples of 30,000 reviews, in which 24,000 reviews are randomly selected as the training set and validation, and the remaining 6,000 reviews are used as the test set.

Rotten Tomatoes The Rotten Tomatoes Movie Review data set is a corpus of movie reviews used for sentiment analysis and is available at `https://github.com/nicolas-gervais/rotten-tomatoes-dataset,` which contains 480,000 reviews. We randomly split the dataset into training set and test set by ratio 80-20. Then 10% of the training data were used as validation.

Hotel Reviews The Hotel Reviews data set is comprised of 20,000 review samples evaluating 1,000 hotels and is available on Kaggle: `https://www.kaggle.com/datafiniti/hotel-reviews`. In this paper, we assumed a positive sentiment for reviews of 4 and 5 star ratings and a negative sentiment for reviews of 1 and 2 stars. Reviews with 3 stars were ignored. This assignment yields 17,746 positive reviews and 2,254 negative ones. To balance out the data set, we randomly picked 2,254 positive reviews to make them equal, making the total of 4,508 reviews used in our experiments.

Steam Reviews The dataset contains reviews from Steam's best selling games as February 2019 and is available on Kaggle `https://www.kaggle.com/luthfim/steam-reviews-dataset`. We preprocessed the data by by removing potentially incomplete reviews (with less than 10 characters or 2 sentences) and sampling 65,000 positive reviews and 65,000 negative ones.

For pre-processing, the period ('.'), the question mark ('?'), and the exclamation mark ('!') were used as delimiters to define the boundary between sentences. All words were then converted to the lowercase and punctuations were removed using the definition in `string.punctuation` constant in Python 3.5. In all experiments, we used pre-trained BERT-based language model with mean-tokens pooling Reimers and Gurevych (2019) to convert the raw sentence data to sentence embeddings.

A.2 Models

Vanilla LSTM We used 300-dimensional GloVe word embeddings Pennington et al. (2014) to encode words in sentences. An LSTM model with 2 hidden layers of size 128 each was used. The final prediction was made by a fully connected layer of size 256. A dropout layer of the rate 0.5 was used immediately before the fully connected layer. The implementation is done in Tensorflow 1.15.

DistilBERT DistilBERT Sanh et al. (2019) is considered as a light-weight version of the state-of-the-art BERT model with smaller, faster, and less expensive deployment time and resources. In our experiments, a pre-trained DistilBERT model was transferred and fine-tuned to each target data set. We used an implementation that was available in the Hugging Face Transformers Library (`https://github.com/huggingface/transformers`), which was implemented in PyTorch and TensorFlow 2.0.

ProSeNet ProSeNet Ming et al. (2019) is a state-of-the-art prototype-based interpretable RNN. For the implementation of ProSeNet, we used an LSTM layer with 2 hidden layers of size 128 and the dropout rate 0.5 for the sequence encoder. This is the same configuration as the ProtoryNet's RNN layer. We tuned K from $[5, 20, 50, 100, 200, 400]$ using a validation set.

ProtoryNet We used TensorFlow v2.3[3] to implement ProtoryNet (and v1.15 for other benchmark models). In addition, the LSTM layer in ProtoryNet was implemented to have the same architecture as the baseline methods to eliminate the bias. Just like ProSeNet, we tuned K from $[5, 20, 50, 100, 200, 400]$ using a validation set and fixed $\alpha = 0.1$ and $\beta = 1e^{-4}$.

Bag-of-words We followed the "Bag-of-words and its TFIDF" in Section 3.1 in paper Zhang et al. (2015). While being considered traditional, the method still achieved very good performance in many cases. We use TFIDF (term-frequency inverse-document-frequency) as the word counts, and Logistic Regression as the classifier for the purpose of interpretability. The method is implemented in Python and Scikit-learn libraries with default configuration.

Appendix B. Survey Questions

Figures below show a few examples of the survey questions we used for the user evaluation study.

For the prototype selection, we created 10 questions each, for ProSeNet and ProtoryNet. Here we only show one example in Figure 8.

Figure 9 and Figure 10 show how we educated the subjects about how ProtoryNet or ProSeNet work.

For diagnosing the ProSeNet and ProtoryNet, we create 3 questions for each model. We show one example for each model in Figure 11 and Figure 12.

3. https://www.tensorflow.org/

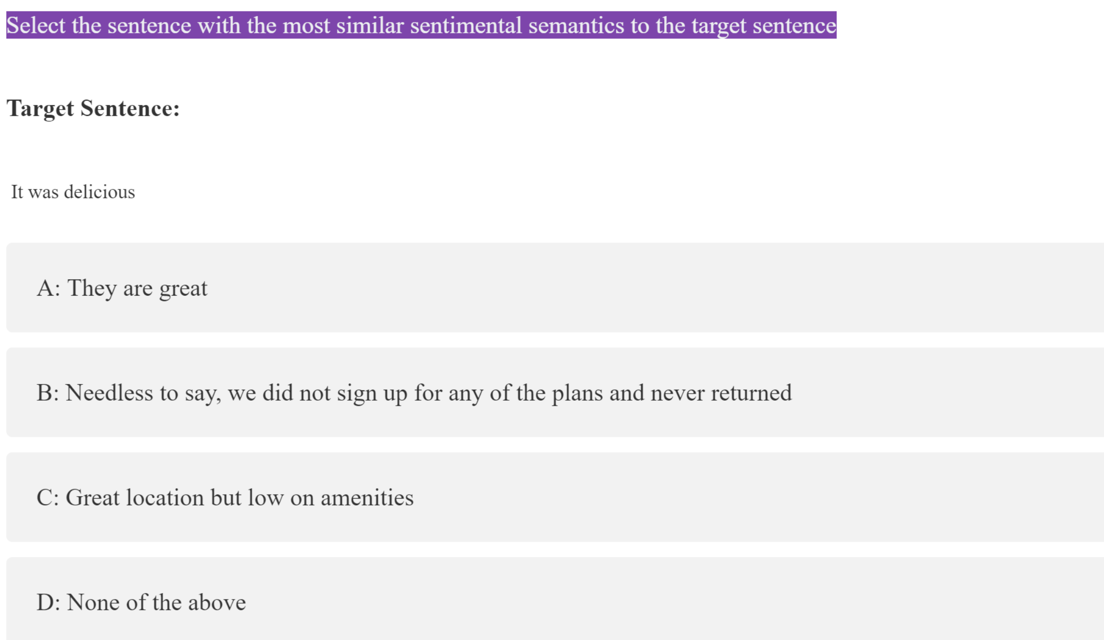


Figure 8: Prototype selection question.

Now we present to you a model ProtoryNet that is based on the similarity between sentences and "prototypical" sentences to make a prediction. A prototype sentence is a sentence that is selected by the model to represent a group of sentences with similar meanings.

Let us a score between 0 and 1 to represent whether a sentiment is positive or negative. Larger values (closer to 1) indicate more positive sentiment and smaller values (close to 0) indicate more negative sentiment.

ProtoryNet first maps each sentence in a paragraph to the most similar prototypical sentence and then based on the trajectory of the sentiment in a paragraph, the model determines the overall sentiment of a paragraph

[Example] Here's a review of four sentences

[s1] food was delicious at this pace. [s2] But the service is super low and we waited for more than half an hour for our desert. [s3] When we asked the waitress about it, she was incredibly rude to us. [s4] We will not go back again.

The model finds the following prototypes for each sentence.

Prototype for [s1]: great food (*sentiment*: 0.9)

Prototype for [s2]: there is always a long wait for the food （*sentiment*: 0.2)

Prototype for [s3]: the service is very bad (*sentiment*: 0.05)

Prototype for [s4]: Overall, it was not worth it (*sentiment*:0.1)

We can visually observe how the sentiment changes. Sentiment drops from 0.9 to very low values.

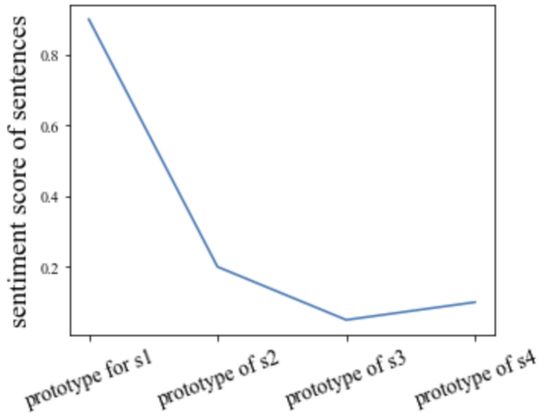


Figure 9: Education material for ProtoryNet

Input: excellent food . extremely clean . the staff is friendly and efficient . really good atmosphere .

Prediction: Positive(1.00)

Explanation:

(0.86) **EXCELLENT FOOD . NICE** decor . a little pricey for the proportions . the salmon **WAS AMAZING** . -> Positive(1.58)

(0.68) **GREAT FOOD SERVICE** and views . hard to beat . this is our **FAVORITE** <other> restaurant in the valley . -> Positive(1.23)

(0.50) **GREAT FOOD GREAT SERVICE** . vietnamese <other> rolls unbelievable . **GREAT** food just what the neighborhood needed . -> Positive(2.82)

Similarity between the input and each prototype
For example, 0.86 is the similarity between the input and 1st prototype

Confidence of the prototype (how much a model believes a prototype is positive or negative)

The total score is 0.86 * 1.58 + 0.68 * 1.23 + 0.5 *2.82 = 3.6052 >0
So the sentiment of the review is positive

Figure 10: Education material for ProSeNet

Example 1

True Sentiment: Negative. **Model prediction**: Positive

Review: [s1] Don't go. [s2] I got more problems and sounds on my car after I spent $800 there. [s3] Unbelievable!

Explanations
[s1] -> prototype: Definitely won't be going back to this location (sentiment: 0.005)
[s2] -> prototype: I'd be willing to bet they get SO MANY complaint letters they just can't keep up(sentiment: 0.198)
[s3] -> Prototype: but eh (sentiment: 0.683)

Can you identify which prototype caused the incorrect prediction of the input?

A: prototype 1

B: prototype 2

C: prototype 3

D: I can't decide

Figure 11: Diagnosis question for ProtoryNet model

Example 1

True Sentiment: Negative. **Model prediction**: Positive

Input: Don't go. I got more problems and sounds on my car after I spent $800 there. Unbelievable!
Explanations
(0.732) Prototype 1: Kuhn's automatically receive 1 for being open 24 hours. Beyond that, customer service has been great, the shelves are stocked well, prepared food and deli items are better for you than fast food with better ingredients at a better value! When I'm in there, this is my go-to shopping spot. Positive (0.784)

(0.718) Prototype 2: This Starbucks is teeny-tiny! Seating inside is VERY limited. This is a Starbucks to grab and go and continue your shopping at the Waterfront. Baristas are friendly and fast. Negative(0.445)

(0.715) Prototype 3: Exceeded my expectations! I had the fried chicken. It was tender and not greasy. The yams were tasty. Sweet but not overbearing. I will definitely visit again when I'm in the area! Negative (0.751)

Can you identify which prototype caused the incorrect prediction of the input?

prototype 1

prototype 2

prototype 3

I can't decide

Figure 12: Diagnosis question for ProSeNet model