Interpreting Convolutional Sequence Model by Learning Local Prototypes with Adaptation Regularization

Jingchao Ni,[1] Zhengzhang Chen,[1] Wei Cheng,[1] Bo Zong,[2] Dongjin Song,[3]
Yanchi Liu,[1] Xuchao Zhang,[1] Haifeng Chen[1]
[1]NEC Laboratories America, [2]Salesforce, [3]University of Connecticut
[1]{jni, zchen, weicheng, yanchi, xuczhang, haifeng}@nec-labs.com, [2]lerry.z@gmail.com, [3]dongjin.song@uconn.edu

ABSTRACT

In many high-stakes applications of machine learning models, outputting only predictions or providing statistical confidence is usually insufficient to gain trust from end users, who often prefer a transparent reasoning paradigm. Despite the recent encouraging developments on deep networks for sequential data modeling, due to the highly recursive functions, the underlying rationales of their predictions are difficult to explain. Thus, in this paper, we aim to develop a sequence modeling approach that explains its own predictions by breaking input sequences down into evidencing segments (*i.e.*, sub-sequences) in its reasoning. To this end, we build our model upon convolutional neural networks, which, in their vanilla forms, associates local receptive fields with outputs in an obscure manner. To unveil it, we resort to case-based reasoning, and design *prototype modules* whose units (*i.e.*, prototypes) resemble exemplar segments in the problem domain. Each prediction is obtained by combining the comparisons between the prototypes and the segments of an input. To enhance interpretability, we propose a training objective that delicately adapts the distribution of prototypes to the data distribution in latent spaces, and design an algorithm to map prototypes to human-understandable segments. Through extensive experiments in a variety of domains, we demonstrate that our model can achieve high interpretability generally, together with a competitive accuracy to the state-of-the-art approaches.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**.

KEYWORDS

Deep learning; Sequence modeling; Interpretation

ACM Reference Format:
Jingchao Ni,[1] Zhengzhang Chen,[1] Wei Cheng,[1] Bo Zong,[2] Dongjin Song,[3], Yanchi Liu,[1] Xuchao Zhang,[1] Haifeng Chen[1]. 2021. Interpreting Convolutional Sequence Model by Learning Local Prototypes with Adaptation Regularization. In *Proceedings of the 30th ACM International Conference on Information and Knowledge Management (CIKM '21), November 1–5, 2021, Virtual Event, QLD, Australia.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3459637.3482355

CIKM '21, November 1–5, 2021, Virtual Event, QLD, Australia

ACM ISBN 978-1-4503-8446-9/21/11…$15.00
https://doi.org/10.1145/3459637.3482355

1 INTRODUCTION

Sequential data are prevalent in a variety of real-life applications, including protein sequences [12], electronic health records (EHR) [11], and event logs of marketing campaigns [50]. Recent rapid developments on deep learning have produced many complicated models that achieve encouragingly precise predictions on sequences, such as LSTM [25], WaveNet [41], BERT [13], XLNet [51], and GPT-3 [7]. As these models penetrate critical domains, *e.g.*, medicine, criminal justice systems, and financial markets, the inability of humans to comprehend these models is becoming a concern [8]. In many scenarios, outputting only predictions or statistical confidence is insufficient for one to assess trust, and decide whether to take any actions or not. As such, there have been immense interests in designing transparent and intelligible machine learning systems on sequence modeling [4, 11, 29].

Recently, the growing demands for interpretability have spurred on an explosion of work on explainable sequence modeling, where a second (post-hoc) model is often created to explain the first black-box model [9, 29, 38, 44, 47]. For example, Karpathy et al. [29] proposed an approach to visualize the cell activation for illuminating the long-range dependencies learned by LSTMs. Ribeiro et al. [44] enforced if-then rules that anchor several segments of the input to justify its predicted output. In contrast, self-explaining models with built-in interpretability have received less attention [35]. As discussed in [45], since the reasoning process of a model is independent to the post-hoc explanations, the inferred explanations may not faithfully reflect the reasoning processes. Thus, one desired way forward is to design models with intrinsic interpretability during learning for further understanding of the complicated models.

In this work, we are interested in deep models on sequences that can interpret their own reasoning processes. In particular, we seek to define a form of interpretability in sequence processing that relies on local segments (or sub-sequences[1]). This is because not all information in a sequence are useful, and short segments can facilitate presenting explanation in a succinct form. Such a reasoning mechanism also imitates the way that human process sequences: given a long sequence to be classified, it is intuitive to locate a few interesting areas and compare them with exemplar segments in different classes. For example, physicians may compare irregular shapes of suspected segments in electrocardiography (ECG) signals with their past observations for diagnosing Arrhythmia [27].

For a long time, sequence modeling is synonymous with recurrent networks [19]. Whereas recent results indicate that convolutional architectures can outperform recurrent networks across wide tasks, while demonstrating longer effective memory [3]. One

[1]In this work, we use segments and sub-sequences interchangeably.

uniqueness of convolution based sequence models lies in their capability in using filters to locate receptive fields of different sizes, and temporally associating important local patterns with outputs. Such a paradigm aligns closely with our desired interpretability via discovering explainable segments, but the underlying rationale of its reasoning procedure is obscure up till now. It is still challenging to uncover this black-box model and transform its inner-workings into a human-understandable manner.

To address this problem, we resort to case-based reasoning [31], and draw conclusions for a new sample by comparing it with a few *prototypes* (*i.e.*, typical cases) in the problem domain. Recently, prototype learning has been successfully leveraged for interpreting image classification [10, 22, 32], yet it is underdeveloped for interpreting the predictions on sequential data. Until very recently, there is one method incorporating prototype learning into LSTM for explaining sequence classification [36]. However, the recurrent structure restricts its interpretation to be based on prototypes that represent *entire sequences*, rather than *local segments*. Although a beam search based algorithm was proposed to remove irrelevant tokens, the method does not guarantee to generate short segments since it tends to preserve as much information as in the entire sequences. Hence, a more flexible method that ensures succinct, meaningful, and resolution-controllable segments, is in demand.

In this work, we combine sequence convolutional network with prototype learning, and propose a new model SCNPro, on sequential data. Building upon convolutional architectures, SCNPro exerts filters to learn from the set of training sequences a limited number of prototypical segments, whose resolution (*i.e.*, size) is fully controllable by tuning the filter sizes. The controllable resolution does not only ensure short segments, but also enable learning prototypes that reflect experts' intuition and knowledge in a domain. SCNPro learns prototypical segments purely by stochastic gradient descent, without trying out lots of combinations of pre-defined candidates. It learns an internal notion of similarity, which is aware of the lengths of segments for fair comparisons. Prediction on a new sequence is made based on a weighted combination of the similarities between its segments and the inferred prototypes. To ensure meaningful prototypes, we design a new learning objective from the perspective of minimizing distribution discrepancy. It incorporates a regularization derived from maximum mean discrepancy (MMD) statistic [21] for adapting the distribution of prototypes to best fit the distribution of the training data. The main contributions of this work are summarized as follows.

- We propose a novel convolutional model on sequences, SCNPro, with inherent interpretability, which learns local and resolution-controllable prototypes for explaining its own reasoning processes.
- We design an effective learning objective from the perspective of distribution adaptation and a corresponding algorithm for learning prototypes that are both meaningful and understandable. The entire model is end-to-end trainable.
- We conduct comprehensive experiments on datasets across various domains, including customer reviews in e-commerce, ECG signals in healthcare, and protein sequences in biology. The results demonstrate that the predictions of SCNPro are both accurate and intuitively interpretable.

The rest of the paper is organized as follows. Sec. 2 reviews related work, Sec. 3 introduces SCNPro architecture, the optimization problem, and our training strategy. Sec. 4 discusses the experimental results, and Sec. 5 concludes the paper.

2 RELATED WORK

To our best knowledge, this is the first work to investigate the reasoning processes of convolutional sequence models through an inherent segment-based interpretability. Existing interpretable sequence models are mostly designed on recurrent networks [1, 11, 29, 38, 39]. However, recent studies indicate that convolutional models can achieve state-of-the-art performance in numerous applications, including audio synthesis [41], machine translation [16, 17], and medical signal classification [27]. More recent empirical evaluations even found wider tasks where convolutional models outperform recurrent variants [3]. Despite the superior performance, the underlying mechanism of convolutional models remains more obscure than recurrent networks due to fewer interpretable variants in literature, and our work seeks to fill the gap.

Our work relates to many post-hoc explanation approaches on sequences, which aim to fit explanations to a model's predictions. For instance, Karpathy et al. [29] visualized the cell activation for illuminating the long-range dependencies learned by LSTMs. Strobelt et al. [47] developed a visual analysis tool for understanding the hidden state dynamics of LSTMs. Some approaches decompose the predictions of a sequence into measurable contributions based on tokens [1, 39], segments [38], hierarchical clusters [46], and account for the interactions between these components [26]. Another approach [44] enforces if-then rules that anchor several segments of the input to justify its predicted output. Knowledge distillation [24] has also been adapted for interpretation [9, 43]. The idea is to approximate the predictions of a more complicated model by a simpler yet interpretable model. As discussed before, these approaches only fit explanations to observed predictions, but don't unveil the actual reasoning processes underlying the computation, thus may not provide results that are as faithful as those from the built-in interpretation approaches [45].

Attention mechanism is one option for building models with intrinsic interpretatbility [2], such as RETAIN (for EHR data) [11] and some large pre-trained language models [7, 13, 51]. Although these methods can highlight some pertinent parts to the models' predictions, attention cannot represent class-wise patterns that explicitly link inputs to outputs, and its usefulness depends on the types of explanations to seek [49]. In contrast, SCNPro has a transparent decision-making process via learning prototypes to summarize different classes, and is a general model for any sequential data.

Prototype learning was firstly combined with image classifiers for explainability [10, 22, 32]. Subsequently, it was used to explain time series classification by autoencoders [15], and sequence classification by LSTMs [36], which may be the most relevant works. However, as discussed in Sec. 1, using either the bottleneck features of autoencoders or the last hidden states of LSTMs will confine the internal comparison to be among entire sequences. This inevitably results in prototypes that are learned to represent sequences, rather than local segments. Although [36] introduced a beam search based algorithm to shrink prototypical sequences, it tends to remove

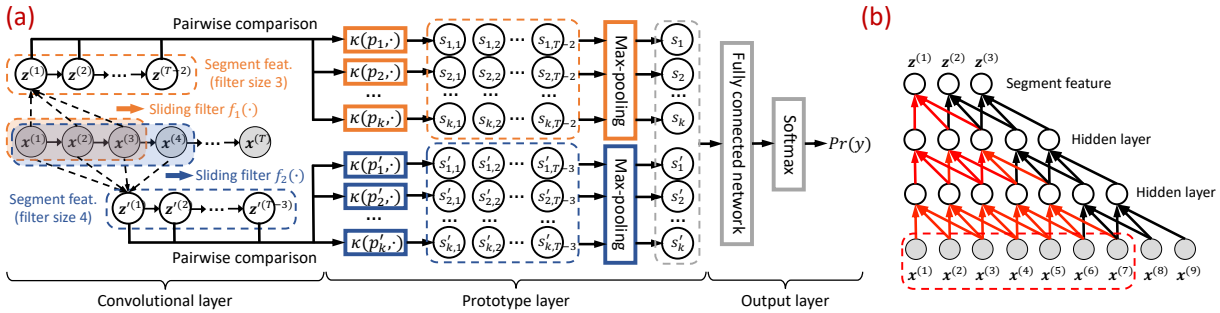

Figure 1: An illustration of (a) SCNPRO architecture, with two filter sizes $\{3, 4\}$ and two prototype modules, where $\kappa(\cdot,\cdot)$ is a kernel function; (b) deep convolution with filter size 3, where the highlighted receptive field is the segment that $\mathbf{z}^{(1)}$ represents.

marginal tokens as limited by its sequence-level comparisons, and has no resolution-control. Due to its large searching space of subsequences, this beam search algorithm also has a complexity infeasible for long sequences such as time series. In contrast, SCNPRO integrates fairly fine-grained comparison between segments, learns concise prototypes, and is efficient by using filters for searching segments.

3 OUR PROPOSED MODEL

In this section, we introduce the framework of SCNPRO, including its architecture, optimization problem, and training process.

Firstly, a word about some notations. Let $\mathcal{D} = \{((\mathbf{x}_i^{(t)})_{t=1}^{T}, y_i)\}_{i=1}^{n}$ be a labeled sequence dataset, where T is the length of a sequence (short sequences are zero-padded to length T), n is the total number of samples, $\mathbf{x}_i^{(t)} \in \mathbb{R}^{d\times 1}$ is the input vector at time step t of the i-th sequence, d is its dimensionality, and $y_i \in \{1, ..., c\}$ indicates the label of the i-th sequence.

Our goal is to learn a number of prototypes that represent some segments in the training sequences, such that when a new sequence is to be classified, its segments are compared with the learned prototypes, and the resulted similarity scores are combined and referred for determining its class. In the meantime, the prediction is explained by highlighting the most similar prototypes.

3.1 Model Architecture

In this section, we present a layer-by-layer description of SCNPRO. Fig. 1(a) illustrates the overall architecture, which consists of a convolutional layer, a prototype layer, and an output layer.

3.1.1 Convolutional Layer. Given an input sequence $(\mathbf{x}^{(t)})_{t=1}^{T}$ (the subscript i is omitted for simplicity), we use convolution as a way for sampling segments. The convolutional layer involves a *filter* $\mathbf{W} \in \mathbb{R}^{d\times w}$, where w is the filter size. The filter operates on a window of w time steps to sample a segment and produce a new feature. More formally, let $\mathbf{x}^{(t:t+w-1)}$ be a segment corresponding to a length-w window starting at t and ending at $t+w-1$, *i.e.*, $[\mathbf{x}^{(t)}, ..., \mathbf{x}^{(t+w-1)}]$, where $[\cdot,\cdot]$ represents the concatenation operator. A new feature $z^{(t)}$ is then generated from $\mathbf{x}^{(t:t+w-1)} \in \mathbb{R}^{d\times w}$ by

$$z^{(t)} = f(\mathbf{W} * \mathbf{x}^{(t:t+w-1)} + b) \tag{1}$$

where b is a bias, $f(\cdot)$ is a non-linear activation function, and the $*$ operator provides the sum of an element-wise multiplication.

Generalizing Eq. (1) from a single filter to h ($h \geq 1$) length-w filters, a vectorial feature $\mathbf{z}^{(t)} \in \mathbb{R}^{h\times 1}$ can be obtained for representing the segment $\mathbf{x}^{(t:t+w-1)}$ [23].

The sampling is performed by sliding the filter with stride 1 over each length-w window of the input sequence, $\mathbf{x}^{(1:w)}$, $\mathbf{x}^{(2:w+1)}$, ..., $\mathbf{x}^{(T-w+1:T)}$, to produce a *feature map* $[\mathbf{z}^{(1)}, ..., \mathbf{z}^{(T-w+1)}]$.

The ordinary convolutional neural networks typically exert a row-wise max-pooling, $\max([\mathbf{z}^{(1)}, ..., \mathbf{z}^{(T-w+1)}])$, to extract salient features for downstream tasks [30]. Despite its widespread use, this pooling has masked out the individual meaning of $\mathbf{z}^{(t)}$, thus hinders the use of segments for explanations. In light of this, we design SCNPRO to preserve the feature map without pooling, so that *segment feature* $\mathbf{z}^{(t)}$ can be compared with prototypes in latent spaces.

Furthermore, it is beneficial to use multiple filters with different sizes of w to take different resolutions of segments into account, which means a more comprehensive cover of local information in the input [30]. For instance, in Fig. 1(a), the leftmost part illustrates two filter sizes with $w = 3$ and $w = 4$, respectively.

3.1.2 Prototype Layer. Suppose there are g different filter sizes (*e.g.*, $g = 2$ in Fig. 1(a)), the prototype layer contains g different modules. Each module associates with one filter size. The reason for designing separate modules is to accommodate the non-comparable latent spaces that may be generated by different filter sizes. For example, different sizes may have different output channels (*i.e.*, different h), which generate latent features $\mathbf{z}^{(t)}$ of different dimensionalities.

In the following, for clarity, we introduce our model architecture using one filter size w. Extension to multiple filter sizes is similar since they are parallel.

For a particular filter size, its associated module stores a set of k prototype vectors $\mathcal{P} = \{\mathbf{p}_j\}_{j=1}^{k}$, which are model parameters to be learned. Each prototype $\mathbf{p}_j$ has the same dimensionality as the segment feature $\mathbf{z}^{(t)}$. We allocate a pre-determined number of prototypes for each class. This ensures every class will be represented by some prototypes, and no class will be left out. Let $\mathcal{P}_r \subseteq \mathcal{P}$ be the subset of prototypes that are allocated to class r ($1 \leq r \leq c$), its prototypes are supposed to represent the most pertinent segments for distinguishing the sequences of class r from others.

This layer computes a Gaussian kernel between each prototype $\mathbf{p}_j$ $(1 \le j \le k)$ and each segment feature $\mathbf{z}^{(t)}$ $(1 \le t \le T - w + 1)$ to evaluate their proximity

$$s_{j,t} = \kappa(\mathbf{p}_j, \mathbf{z}^{(t)}) = e^{-\|\mathbf{p}_j - \mathbf{z}^{(t)}\|_2^2} \tag{2}$$

where $s_{j,t}$ represents a similarity score that measures the proximity, which ranges from 0 to 1. In Eq. (2), we empirically set bandwidth as 1 for its effectiveness and simplicity.

For each prototype $\mathbf{p}_j$, after calculating $s_{j,1}, ..., s_{j,(T-w+1)}$ for all segment feature $\mathbf{z}^{(t)}$'s, we perform a max-pooling

$$s_j = \max\left(\{s_{j,1}, ..., s_{j,(T-w+1)}\}\right) \tag{3}$$

to obtain a score s_j. The purpose of using max-pooling is to capture the occurrence of $\mathbf{p}_j$ in any segment of the input sequence. If s_j is large, then there exists a segment feature $\mathbf{z}^{(t)}$ very close to prototype $\mathbf{p}_j$ in the latent space. This in turn means a segment in the input sequence resembles the structure that $\mathbf{p}_j$ represents.

With the computed similarity scores s_j $(1 \le j \le k)$ for all k prototypes, we concatenate them to form a similarity vector $\mathbf{s} \in \mathbb{R}^{k \times 1}$, which is then fed to the output layer for target prediction. It is noteworthy that each entry of $\mathbf{s}$ represents how likely a particular prototype appears in the input sequence. Hence, using $\mathbf{s}$ facilitates revealing relevant prototypes that interprets predictions.

Remark. Our architecture is different from other prototype learning based methods on sequences [15, 36] in essence by characterizing different prototype modules for accommodating different filter sizes, and integrating the max pooling operation for capturing the occurrence of prototypical segments in sequences.

3.1.3 Output Layer. The output layer contains a fully connected network that computes the logits $\mathbf{a} = \mathbf{W}\mathbf{s}$, where $\mathbf{W} \in \mathbb{R}^{c \times k}$ is the weight matrix, and c is the number of unique labels.

To improve interpretability, we omit the bias in this layer, and add a constraint that $\mathbf{W}$ is non-negative. As such, W_{rj} explains the contribution of similarity score s_j to class r. To account for the allocated prototypes in different classes, we initialize $\mathbf{W}$ such that $W_{rj} = 1$ if $\mathbf{p}_j$ is assigned to class r, and $W_{rj} = 0$ otherwise. This initialization substantially speeds up convergence in practice.

Additionally, for classification task, a softmax layer is added to compute the predicted probability

$$Pr(y|\mathbf{x}^{(1)}, ..., \mathbf{x}^{(T)}) = \text{softmax}(\mathbf{a}) \tag{4}$$

Then, for every input sequence, we can draw its label from the categorical distribution $\text{Categorical}(Pr(y|\mathbf{x}^{(1)}, ..., \mathbf{x}^{(T)}))$.

Similarly, for multi-class classification, we can model the probability in Eq. (4) by $\text{sigmoid}(\mathbf{a})$, and draw labels from a Bernoulli distribution for every class independently.

3.1.4 Discussion on Receptive Fields. It is straightforward to stack multiple convolutional layers per filter size in Fig. 1(a) to build a deeper architecture. By doing so, each segment feature $\mathbf{z}^{(t)}$ will correspond to a lager receptive field, hence represent a longer segment.

For example, Fig. 1(b) illustrates a three-layer architecture with filter size $w = 3$. The size of the receptive field for $\mathbf{z}^{(t)}$ at the third layer is 7, which can be calculated by $l(w-1)+1$, where l is the number of layers. Therefore, each segment feature $\mathbf{z}^{(t)}$ maps to a segment $\mathbf{x}^{(t:t+l(w-1))}$, and correspondingly, SCNpro learns prototypical segments of length $l(w-1)+1$.

In applications where long segments are preferred (*e.g.*, audio analysis), dilated convolution can be employed [41]. Dilated convolution enables exponential receptive field growth with network depth, thus generates very large receptive fields with a few layers and a small filter size [3]. In this work, we focus on short segments, and leave the evaluation of dilated convolution in the future work.

3.2 The Optimization Problem

The optimization problem of SCNpro should reflect the needs for both accuracy and interpretability. For accuracy, we minimize the negative log-likelihood estimation over the training dataset

$$\ell_c = -\sum_{i=1}^{n} \log\left(Pr_{\boldsymbol{\theta}}(y_i|\mathbf{x}_i^{(1)}, ..., \mathbf{x}_i^{(T)})\right) \tag{5}$$

where $\boldsymbol{\theta}$ represents the set of all trainable parameters, including those of the neural networks, and the set of all prototypes $\mathcal{P}$.

In particular, for the categorical distribution based on Eq. (4), the above likelihood in Eq. (5) is equivalent to the cross-entropy loss for penalizing misclassifications. For Bernoulli distribution, Eq. (5) is equivalent to the binary cross-entropy loss over all classes.

To enhance interpretability, we propose to manipulate the distribution of prototypes in the latent spaces such that they are trained to adapt to the distribution of the segment samples in the data. Suppose the distributions of the segment features and prototypes are Q_z and Q_p, respectively. Similar to domain adaptation [33], we are interested in evaluating the difference between Q_z and Q_p. In this work, we investigate the feasibility of employing the empirical estimation of Maximum Mean Discrepancy (MMD) [21] as the measurement of difference. Formally, MMD defines a discrepancy

$$D_{\mathcal{H}}(Q_z, Q_p) \triangleq \sup_{f \in \mathcal{H}} \left(\mathbb{E}_{Z \sim Q_z}[f(Z)] - \mathbb{E}_{P \sim Q_p}[f(P)]\right) \tag{6}$$

where $\mathcal{H}$ is a class of functions. When $\mathcal{H}$ is a reproducing kernel Hilbert space (RKHS), Eq. (6) can be expressed as the distance between the mean embeddings of the Q_z and Q_p: $D_{\mathcal{H}}(Q_z, Q_p) = ||\mu_{Q_z} - \mu_{Q_p}||_{\mathcal{H}}^2$. The main theoretical result is that Q_z is indistinguishable from Q_p if and only if $D_{\mathcal{H}}(Q_z, Q_p) = 0$ [21].

In practice, given all segment samples $\{\mathbf{z}_i^{(1)}, ...\mathbf{z}_i^{(T-w+1)}\}_{i=1}^n$ and prototypes $\{\mathbf{p}_1, ..., \mathbf{p}_k\}$, we can estimate the difference between Q_z and Q_p using MMD by the square distance between the empirical kernel mean embeddings as

$$\begin{aligned}\hat{D}_{\mathcal{H}}(Q_z, Q_p) = &\frac{1}{n^2(T-w+1)^2} \sum_{i,i'=1}^{n} \sum_{t,t'=1}^{T-w+1} \kappa(\mathbf{z}_i^{(t)}, \mathbf{z}_{i'}^{(t')}) \\ &- \frac{2}{nk(T-w+1)} \sum_{i=1}^{n} \sum_{t=1}^{T-w+1} \sum_{j=1}^{k} \kappa(\mathbf{z}_i^{(t)}, \mathbf{p}_j) + \frac{1}{k^2} \sum_{j,j'=1}^{k} \kappa(\mathbf{p}_j, \mathbf{p}_{j'})\end{aligned} \tag{7}$$

where $\kappa(\cdot, \cdot)$ is the kernel function associated with the RKHS. It is noteworthy that our goal is to use MMD to regularize the learning of prototypes, such that the learned prototypes can capture the distribution of segments in the data. From the regularization perspective, the first term in Eq. (7) is constant w.r.t. $\{\mathbf{p}_i\}_{i=1}^k$. Thus in the next, we focus on the last two terms in Eq. (7), and derive regularizers for improving the interpretability.

3.2.1 Adaptation Regularized Prototype Learning. For the i-th sequence, the second term in Eq. (7) encourages a high proximity for every segment-prototype pair via kernel embeddings. This regularization is too strict to interpret, because practically not all segments of a sequence are important for its prediction. Instead, it is sufficient for a sequence to have at least one distinguishable segment for determining its label. To reflect this fact, we relax this term by

$$\sum_{i=1}^{n}\sum_{t=1}^{T-w+1}\sum_{j=1}^{k}\kappa(\mathbf{z}_i^{(t)},\mathbf{p}_j) \Rightarrow \sum_{i=1}^{n}\max_{1\le t\le(T-w+1)}\max_{\mathbf{p}_j\in\mathcal{P}_{y_i}}\kappa(\mathbf{z}_i^{(t)},\mathbf{p}_j) \quad (8)$$

where $\mathcal{P}_{y_i}$ is the subset of prototypes that are allocated to class y_i, *i.e.*, the corresponding class of the i-th sample. Recall in Sec. 3.1.2, every class has been allocated some prototypes.

Eq. (8) only evaluates the segment-prototype pair with the maximal kernel proximity, implying at least one explainable segment for each sequence. In light of prototype learning, Eq. (8) encourages every prototype to be trained to locate in a cluster of similar segments, such that it represents a segment pattern for distinguishing its class from others. Note $\mathcal{P}_{y_i}$ ensures a segment is only paired with a prototype of its own class. This constraint circumvents the dilemma of clustering prototypes and segments with mixed classes.

Moreover, to have a physical meaning, each prototype should resemble a real segment. To this end, we decompose the second term in Eq. (7) into two terms by dividing its coefficient 2. One of the terms has been relaxed by Eq. (8). For another, let $\mathcal{Z} = \{\mathbf{z}_i^{(1)},...\mathbf{z}_i^{(T-w+1)}\}_{i=1}^{n}$ be the set of all segment features, we perform the following relaxed transformation

$$\sum_{j=1}^{k}\sum_{i=1}^{n}\sum_{t=1}^{T-w+1}\kappa(\mathbf{p}_j,\mathbf{z}_i^{(t)}) = \sum_{j=1}^{k}\sum_{\mathbf{z}_i\in\mathcal{Z}}\kappa(\mathbf{p}_j,\mathbf{z}_i^{(t)}) \Rightarrow \sum_{j=1}^{k}\max_{\mathbf{z}_i^{(t)}\in\mathcal{Z}_j}\kappa(\mathbf{p}_j,\mathbf{z}_i^{(t)}) \quad (9)$$

where $\mathcal{Z}_j \subseteq \mathcal{Z}$ is a subset that encompasses segment features sampled from sequences belonging to the same class as $\mathbf{p}_j$.

For each prototype, Eq. (9) evaluates the maximal proximity to nearby segments. Thus, optimizing Eq. (9) will push each prototype toward its closest segment, whereby associating it with the meaning of that real segment. Similar to Eq. (8), here only segments of the prototype's own class, *i.e.*, $\mathcal{Z}_j$, are included. This is for avoiding misclassified association.

Putting Eq. (8) and Eq. (9) together with the third term in Eq. (7), we obtain a loss function for regularizing prototype learning

$$\begin{aligned}\ell_d = &-\frac{1}{n}\sum_{i=1}^{n}\max_{1\le t\le(T-w+1)}\max_{\mathbf{p}_j\in\mathcal{P}_{y_i}}\kappa(\mathbf{z}_i^{(t)},\mathbf{p}_j)\\ &-\frac{1}{k}\sum_{j=1}^{k}\max_{\mathbf{z}_i^{(t)}\in\mathcal{Z}_j}\kappa(\mathbf{p}_j,\mathbf{z}_i^{(t)}) + \frac{1}{k^2}\sum_{j,j'=1}^{k}\kappa(\mathbf{p}_j,\mathbf{p}_{j'})\end{aligned} \quad (10)$$

where the third term penalizes the proximity between every pair of prototypes, thus encourages diversity. This is useful for reducing duplicate prototypes, facilitating efficient use of parameters, and improving the generalization power of the model.

3.2.2 The Objective Function. Now, we can integrate the log likelihood in Eq (5) and the relaxed MMD regularization in Eq. (10) into a unified loss function

$$\ell(\mathcal{D};\boldsymbol{\theta}) = \ell_c + \sum_{v=1}^{g}\ell_d^{(v)} \quad (11)$$

where the adaptation regularization ℓ_d is applied for all prototype modules, corresponding to g different filter sizes.

In terms of the kernel, there could be multiple choices, such as Gaussian kernel, inverse multi-quadratics kernel, etc. In this work, we use Gaussian kernel $\kappa(\mathbf{x},\mathbf{x}') = e^{-\lambda\|\mathbf{x}-\mathbf{x}'\|_2^2}$ for its effectiveness, and leave the comparison of different kernels in our future work.

In terms of optimization, using $\kappa(\cdot,\cdot)$ is equivalent to $\log\kappa(\cdot,\cdot)$. Therefore, inserting logarithmic Gaussian kernel, Eq. (10) becomes

$$\begin{aligned}\ell_d = &\lambda_1\frac{1}{n}\sum_{i=1}^{n}\min_{1\le t\le(T-w+1)}\min_{\mathbf{p}_j\in\mathcal{P}_{y_i}}\|\mathbf{z}_i^{(t)}-\mathbf{p}_j\|_2^2\\ &+\lambda_2\frac{1}{k}\sum_{j=1}^{k}\min_{\mathbf{z}_i^{(t)}\in\mathcal{Z}_j}\|\mathbf{p}_j-\mathbf{z}_i^{(t)}\|_2^2 + \lambda_3\frac{1}{k^2}\sum_{j,j'=1}^{k}-\|\mathbf{p}_j-\mathbf{p}_{j'}\|_2^2\end{aligned} \quad (12)$$

where λ_1, λ_2, and λ_3 are derived from the bandwidth of Gaussian kernels. In Eq. (12), they are trade-off hyperparameters of different terms. The $\max(\cdot)\rightarrow\min(\cdot)$ transformation is due to extracting negative signs. Empirically, it is also beneficial to add a hinge loss on the last term, *i.e.*, $\max(0, d_{\min} - \|\mathbf{p}_j-\mathbf{p}_{j'}\|_2^2)$, with a threshold $d_{\min}$ to control the relative distances between prototypes.

Remark. Our regularization from the perspective of distribution adaptation is more general than the empirical regularizers used by existing methods [10, 15, 32, 36]. Also, our derivation indicates a connection between the MMD based adaptation and the empirical regularization used in those methods by highlighting the effects on *segment clustering* (Eq. (8)), *segment-prototype association* (Eq. (9)) and *prototype diversity* (the third term in Eq. (10)).

3.3 Model Training

The entire SCNpro model is end-to-end differentiable. The model parameters $\boldsymbol{\theta}$ include those in the neural networks and the set of all prototypes $\mathcal{P}$. In our experiments, all of the prototypes were initialized randomly. To train SCNpro, stochastic gradient descent (SGD) can be used to minimize the joint loss function ℓ in Eq. (11) on mini-batches.

After SGD converges, every prototype $\mathbf{p}_j$ has been pushed closely to a segment $\mathbf{z}_i^{(t)}$ in the training domain, because of the segment-prototype association property of Eq. (9). However, $\mathbf{p}_j$ is a vectorial representation in the latent space, thus is not readily interpretable. To induce a physical meaning, similar to [10, 36], we employ the projection approach to project each prototype onto the latent representation of its closest segment. This operation fills the last gap between a prototype and its neighboring segment in the latent space remained by gradient descent:

$$\mathbf{p}_j \leftarrow \arg\min_{\mathbf{z}_i^{(t)}\in\mathcal{Z}_j}\|\mathbf{p}_j-\mathbf{z}_i^{(t)}\|_2^2,\ \forall\,\mathbf{p}_j\in\mathcal{P} \quad (13)$$

As can be seen, Eq. (13) only projects a prototype to the closest segment of its own class, which is consistent with the second term in Eq. (12) (or Eq. (9)) for segment-prototype association. After this step, each prototype represents, and is interpreted by, a real segment on which it has been projected.

In [32], a decoder was jointly trained with its image classifier such that every prototype can be visualized by decoding its latent representation. However, when training on discrete sequential data,

the decoder may not necessarily translate a prototype to a meaningful segment, and the decoder will add overheads on model training. Additionally, the reconstruction loss on the outputs of the decoder may distort the classification results. In contrast, the projection step in Eq. (13) facilitates better explanation, efficiency and accuracy.

4 EXPERIMENTS

In this section, we perform extensive experiments to evaluate SCNPRO on a variety of real-life datasets. For classification, we quantitatively assess the performance. For interpretatbility, we follow existing works [10, 32] to highlight qualitative case studies in different domains.

4.1 Experimental Setup

We compare SCNPRO with sequence models that have state-of-the-art performance, including LSTM [25], BiLSTM [42], CNN [28], ResNet [23], LSTM-AT [2], RETAIN [11], and ProSeNet [36]. Among them, LSTM-AT and RETAIN provide interpretation by using attention for aggregating time steps. The difference is RETAIN reverses the order of input sequence of its LSTMs, and incorporates feature-level attentions. ProSeNet is an interpretable model as introduced in Sec. 2. It combines LSTMs and prototype learning, but learns sequence-level prototypes, rather than local segments.

Unless otherwise noted, we configure the compared models as following (which may be adjusted by validation sets in different applications later). LSTM, BiLSTM, LSTM-AT, RETAIN and ProSeNet have 2 layers, with 50 units per layer. For CNN and SCNPRO, we make use of the well-known CNN-non-static architecture proposed by [30], and set 50 output channels per filter size. The filter size depends on applications, and will be described later. We built ResNet with 7 residual blocks as similar to [23], and set 50 output channels. For ProSeNet, its hyperparameters were set by $\lambda_c = 0.1$, $\lambda_e = 0.1$, $\lambda_d = 0.01$, $\lambda_{l_1} = 0.001$, and its beam search mode is set on. For SCNPRO, we set $\lambda_1 = 0.1$, $\lambda_2 = 0.1$, $\lambda_3 = 0.01$, and initialized prototypes randomly. All neural network architectures were implemented using PyTorch framework, and trained by Adam optimizer with mini-batch sizes within $\{1000, 2000\}$.

4.2 Sentiment Analysis in E-Commerce

First, we evaluate SCNPRO on a sentiment classification task. We use the customer reviews from Yelp [48] and Amazon [40]. For both datasets, each review is tokenized into a sequence using NLTK [5]. We select reviews with lengths in $[30, 50]$ to avoid too long/short sequences. Two cases are considered in this experiment: we use the ratings (one to five) that are associated with each review as labels and perform evaluations on (1) binary (positive: >3, negative: <3) and (2) multiclass (five classes) classifications. In the binary case, neutral reviews (rating=3) were filtered out. As such, Yelp dataset has $1,030,678$ and $1,119,002$ reviews for binary and multiclass cases, respectively. For Amazon dataset, the numbers are $1,567,575$ and $1,746,694$. In the experiment, each dataset was randomly split into 60% training, 20% validation, and 20% testing sets.

For the compared methods, CNN and SCNPRO used filter sizes $\{3, 4, 5\}$. ResNet used filter size 5. ProSeNet used 150 and 200 prototypes in the binary and multiclass cases, respectively. For SCNPRO, 30 prototypes were allocated per class. The dimensionality of the word embeddings was 300. The word embeddings were randomly initialized and fine-tuned during model training.

Table 1: Average accuracy of the compared methods on the customer reviews in Yelp and Amazon datasets.

Method	Yelp		Amazon	
	Binary	Multiclass	Binary	Multiclass
LSTM	0.9636	0.7040	0.9418	0.6995
BiLSTM	0.9625	0.7084	0.9415	0.7019
ProSeNet	0.9636	0.6894	0.9401	0.6741
LSTM-AT	0.9555	0.7018	0.9327	0.6853
RETAIN	0.9564	0.6948	0.9330	0.6777
CNN	0.9569	0.7037	0.9275	0.6811
ResNet	0.9529	0.6956	0.9230	0.6899
SCNPRO	0.9517	0.7004	0.9205	0.6734

Table 1 summarizes the averaged classification accuracy over 10 runs. In most cases, SCNPRO achieves similar performance to other models, especially to the convolutional models and the interpretable models LSTM-AT, RETAIN, ProSeNet. In the binary cases, we observe RNN based methods slightly outperform CNN based methods, which may be a bias of the particular application.

Moreover, Table 2 showcases the reasoning process of SCNPRO on the positive and negative examples in the testing set. Within each review, the highlighted segments (with lengths in $\{3, 4, 5\}$) were automatically located by SCNPRO, which are the most pertinent evidences that SCNPRO used to predict labels. For example, in the review of Table 2(b), "is terrible ." and "very disappointed ." are intuitively negative phrases that summarize the sentiment of this review. SCNPRO relates them to the prototypes "absolutely terrible" and "is the worst" respectively with high similarities (0.99 and 0.98), which have high weights to the negative class (1.53 and 1.32). Additionally, the third segment "was nice but the" represents a transition of sentiment, which finally leads to a negative sentiment. This is similar to the prototype "good. but the", which also has a high weight (0.90) to the negative class.

Furthermore, SCNPRO is able to capture interesting patterns in this task. It does not only learn sentimentally significant prototypes (*e.g.*, "was awesome !") and transition of sentiments (*e.g.*, "good. but the"), but also infers aspect descriptions (*e.g.*, "easy to put on") and phrases with concession (*e.g.*, "only complaint is that"). All of them have intuitive sentiments, which are faithfully reflected by their class weights. As such, the prototypes provide a concise overview of the patterns of different classes.

A summary of the learned prototypes of SCNPRO are in Table 4 and 5. The prototypes are sorted by their weights to different classes. As can be seen from the tables, the learned weights are succinct and distinguishable. Most of the learned prototypes (which may not necessarily be phrases) are meaningful and have clear sentiments toward the class it is associated with, which is demonstrated by the learned weights. We observe some duplicated prototypes, but this has been alleviated a lot by the third term in Eq. (10). Therefore, the prototypes learned by SCNPRO provide a summary of distinguishable and succinct patterns of different classes, which cannot be achieved by the other compared methods. These prototypes also

Table 2: Explanation of SCNPRO on testing samples in the binary sentiment classification task. Three segments located by SCNPRO were highlighted for each review. (a)(b) are samples from Yelp dataset. (c)(d) are samples from Amazon dataset.

Input Review	Segment → Prototype (Similarity)	Pos.	Neg.	Prediction
(a) absolutely fantastic ! great customer service and delicious food . they were willing to modify my order exactly as i wanted and gave me just what i asked for . fabulous job !	absolutely fantastic ! → was awesome ! (0.98)	**1.54**	0.00	Positive
	delicious food . → are delicious . (0.97)	**1.26**	0.00	
	great customer service → service was great (0.93)	**0.76**	0.27	
(b) this place is terrible . they use no fruit and all ice and water . very disappointed . the girl at the register was nice but the drinks is terrible .	is terrible . → absolutely terrible . (0.99)	0.00	**1.53**	Negative
	very disappointed . → is the worst (0.98)	0.00	**1.32**	
	was nice but the → good. but the (0.95)	0.10	**0.90**	
(c) the mirror works great i received it a few days after purchase . easy to put on . the only thing is the mirror does get scratched from objects in your pocket or purse .	easy to put on → easy to put on (1.00)	**0.97**	0.17	Positive
	only thing is the → only complaint is that (0.80)	**1.36**	0.00	
	mirror works great i → it works great . (0.66)	**1.26**	0.00	
(d) this item is terrible . it is too small for any earbuds iv ever seen . the zipper broke in a day . save yourself do n't buy this !	do n't buy → do n't buy (1.00)	0.03	**1.06**	Negative
	. it is too small → but it is too small (0.97)	0.00	**1.32**	
	this item is terrible → stopped working at all (0.84)	0.02	**1.03**	

Table 3: Explanation of the compared methods. For LSTM-AT and RETAIN, darker colors indicate higher weights.

ProSeNet
(a) absolutely fantastic ! great customer service and delicious food . they were willing to modify my order exactly as i wanted and gave me just what i asked for . fabulous job ! (Prediction: Positive)
(0.99) the tacos were amazing ! with handmade tortillas ! so good ! if you 're looking for authentic tacos . this spot is the place to go . (Pos.: 0.41)
(0.96) this place is the greatest ! best family day ever ! our younger kids enjoyed earning tickets in the arcade while the older kids raced the (Pos.: 0.36)
(0.95) good variety . large bowl . protein regular bowl . they have unagi pineapple mango which not all have . fie happy hours which are from am till (Pos.: 0.40)
LSTM-AT
(a) absolutely fantastic ! great customer service and delicious food . they were willing to modify my order exactly as i wanted and gave me just what i asked for . fabulous job ! (Prediction: Positive)
RETAIN
(a) absolutely fantastic ! great customer service and delicious food . they were willing to modify my order exactly as i wanted and gave me just what i asked for . fabulous job ! (Prediction: Positive)

provide hints for users to evaluate and debug the decision-making process of the proposed method.

For comparison, Table 3 presents the explanation of ProSeNet, LSTM-AT and RETAIN. Although ProSeNet has a beam search based simplification algorithm for removing irrelevant tokens in its prototypical sequences, its internal similarity is defined between entire sequences in its loss function. Thus its training always tend to discover long sequences as prototypes, and users have no control over the resolution. This is validated by Table 3, where the long prototypes make the results too dense and less interpretable. Also, since ProSeNet has no segment-level comparison, it cannot locate the pertinent areas in the input sequence, which hinders interpretation as well. In contrast, LSTM-AT and RETAIN can highlight tokens relevant to their predictions, but they don't provide class-level evidences to interpret their reasoning processes. Thus there are much uncertainty and obscurity in their explanations.

4.3 ECG Time Series Classification

Next, we evaluate SCNPRO on real-valued time series using the MIT-BIH Arrhythmia and the PTB Diagnostic ECG datasets [6, 18, 37] Basically, time series are temporally ordered long sequences. ECG is widely used to monitor cardiac health. The MIT-BIH dataset has 109, 446 annotated ECG signals of heartbeats. The annotations

Table 4: The learned top prototypes on Yelp dataset.

Prototype (Positive)	Score	Prototype (Negative)	Score
is by far the best	1.673	food poisoning from their	1.887
was awesome !	1.537	was very rude and	1.813
delicious ! very friendly	1.446	worst customer service	1.687
amazing ! love this	1.442	absolutely terrible .	1.526
service was excellent they had	1.355	the service was terrible .	1.525
one of my favorite	1.326	was terrible . had	1.515
place is amazing ! my	1.325	is the worst i have	1.513
love this place !	1.324	took over an hour	1.430
. a little pricey but	1.299	not good . food	1.405
were able to	1.293	the worst i 've	1.357
...	...	...	...

Table 5: The learned top prototypes on Amazon dataset.

Prototype (Positive)	Score	Prototype (Negative)	Score
very pleased with	1.866	it stopped working at all	2.032
! she loves it and	1.757	your money .	1.787
i am very pleased with	1.750	a waste of money .	1.449
love it ! it	1.575	not recommend this case	1.371
love it ! it 's	1.370	cheaply made and	1.370
only complaint is that	1.355	but it is too small	1.317
awesome product . very easy	1.293	this case is terrible	1.304
. easy to	1.290	is a piece of	1.300
the best case i have	1.281	not worth the	1.285
case . but i knew	1.280	it fell apart in my	1.257
...	...	...	...

are mapped into 5 groups as per the AAMI standard [14]: Normal (N), Supraventricular Ectopic Beat (SVEB), Ventricular Ectopic Beat (VEB), Fusion Beat (F), and Unknown Beat (Q). The PTB dataset has 14, 552 ECG signals, which belong to two categories: Normal (N) and Myocardial Infarction (MI). The datasets were processed according to the protocol proposed by [27]. Each signal was downsampled to 125Hz, short signals were filtered out (with a threshold 50), and in each experiment, the datasets were randomly split into 60% training, 20% validation, and 20% testing sets.

In this experiment, CNN and SCNPRO used filter sizes $\{30, 50\}$. ResNet used filter size 35. SCNPRO was allocated 20 prototypes per class. For ProSeNet, the number of prototypes was 100 and 50 on the MIT-BIH and PTB datasets. LSTM, BiLSTM and ProSeNet were adjusted to use 1 layer, which is better than >2 layers on these datasets. Other configurations were kept the same as before.

Table 6 summarizes the classification performance of the compared methods. We observe SCNPRO performs comparably to other

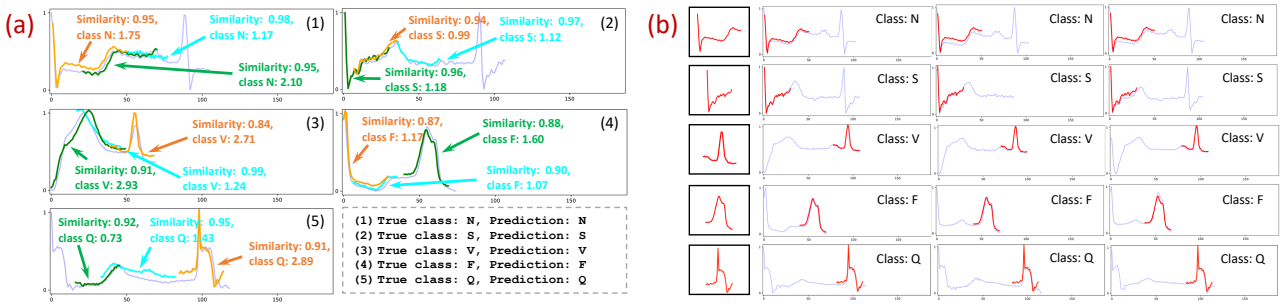

Figure 2: The explanation of SCNPRO on MIT-BIH dataset. (a) The reasoning process. For each testing sample, its top 3 similar prototypes are highlighted. (b) The closest neighbors of the learned prototypes (the leftmost column) in the testing set.

Table 6: Average accuracy of the compared methods on the ECG time series in MIT-BIH and PTB datasets.

Method	MIT-BIH	PTB
LSTM	0.9755	0.9526
BiLSTM	0.9718	0.9523
ProSeNet	0.9594	0.9451
LSTM-AT	0.9450	0.9519
RETAIN	0.9732	0.9427
CNN	0.9750	0.9694
ResNet	0.9783	0.9519
SCNPRO	0.9741	0.9643

methods on MIT-BIH dataset, and slightly outperforms them (except for CNN) on PTB dataset. On PTB dataset, we also observe CNN based methods generally outperform RNN based methods, which is different from the binary cases in Table 1.

Fig. 2(a) presents the reasoning process of SCNPRO on the testing samples of different classes in MIT-BIH dataset (PTB's results are similar thus are omitted for brevity). In each sample, its top three similar prototypes were highlighted at their corresponding locations of the signal. Each prototype is shown with its similarity score to the segment, and its largest class weight. From the figure, we can understand how does SCNPRO make predictions. For example, in Fig. 2(a)(1), the prototypes have similarity scores 0.98, 0.95, 0.95 to their corresponding areas. Thus, their high weights to class N (1.17, 1.75, 2.10) link this sample to class N.

From each sample in Fig. 2(a), we also selected one prototype, and investigated their neighbors in the testing set. Fig. 2(b) presents the prototypes, which correspond to the orange, green, orange, green, orange prototypes in Fig. 2(a)(1-5). Their closest neighbors are shown afterwards (there are more neighbors than presented, which are omitted for brevity). As can be seen, each prototype is representative for its nearby time series. They are also discriminative, and can capture the subtle difference between classes. For example, the samples of classes N and S have similar global waveforms, but are distinguishable at the head, which are effectively detected by prototypes. In addition, the last three prototypes all represent heartbeats, but their different shapes decide their different classes.

We also evaluated ProSeNet on these datasets, whose beam search algorithm is runnable because the time series in these datasets

Table 7: A prototypical protein segment and its neighboring sequences. The highlighted areas are located by SCNPRO. Blue indicates match, and red indicates mismatch.

Prototype: FDVIIIGGGHAGTEAAMAAARMGQQTLLLT, **Weight:** MnmG family (4.26)

Neighbors	Similarity
MFYPDPFDVIIIGGGHAGTEAAMAAARMGQQTLLLTHNIDTL . . . NQHQIFLEPE	1.000
MLYPVEFDVIVVGGGHAGTEAALAAARMGCQTLLLTHNIETL . . . KDSHQIFLEP	0.996
MQQFDIIVVGGGHAGVEAAAVAARMGARTALVSFDPQTIGAM . . . VFLEPEGLDD	0.994
MNFQENYDVVVIGGGHAGVEASLAAARMGSKTLLMTINLNMV . . . ADKPRHQLFL	0.991
MFDVIVIGGGHAGVEASAAAARMGKKTLLLTTLIEQIGAASC . . . IEPQTIDATE	0.990
MLKYDVIVIGGGHAGVEAAAASARLGVPTLLITLKPENLGEM . . . IFLEPEGLDD	0.989
MGRAMHDFDILVVGGGHAGVEAACAAARMGVRTALVSFDPAR . . . DGHQVFLEPE	0.985

are not very long. However, in many cases, ProSeNet learned prototypes that had almost the same lengths as their original signals. On average, each prototype accounted for 78.40% and 87.42% portion of its original signal in MIT-BIH and PTB datasets, respectively. Similar to the results in Sec. 4.2, the method tend to remove marginal signals, and the prototypes thus learned cannot provide a succinct summary of the datasets. As for LSTM-AT and RETAIN, since their attention weights are discrete, they cannot provide a continuous meaning on these time series.

It is noteworthy that the prototypes learned by SCNPRO resemble *shaplets* in conventional times series analysis [20, 52]. However, these methods either rely on a large pool of predefined shapelets [52], or is ad-hoc to shapelet discovery [20]. In contrast, our method is a general sequence model, and provides a new deep learning perspective for learning shapelets in this particular domain.

4.4 Protein Sequence Classification

In this section, we evaluate SCNPRO in biological domain using protein sequences in the UniProtKB database [12]. The database updates regularly, and there were 561, 568 manually annotated protein sequences when we collected the data. The protein sequences are composed of 20 standard amino acids, and can be grouped into families. We selected sequences with minimal length 50 from the 100 largest families, and clipped them with a maximal length of 300. The resulted dataset has 112, 446 sequences, and we split them into 60% training, 20% validation and 20% testing sets.

In this domain, two filters $\{10, 30\}$ were set for CNN and SCNPRO. ResNet used filter size 15. SCNPRO was allocated 5 prototypes per class. ProSeNet used 100 prototypes, and its λ_d was adjusted to 0. Other configurations were kept the same as before.

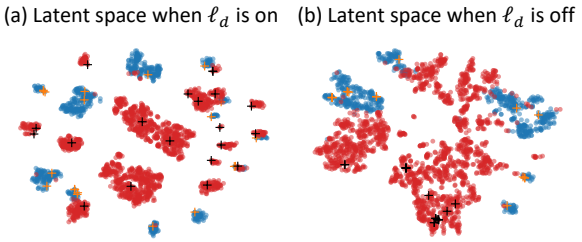

Figure 3: The tSNE visualization on PTB dataset. Circles are segment features. "+" markers are prototypes. Red and black indicate MI class. Blue and orange indicate N class.

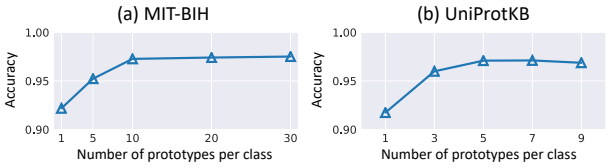

Figure 4: The impacts of the number of prototypes per class.

In 10 different runs, SCNPRO achieved an average accuracy of 0.9709, which is slightly lower than the best performed baselines, CNN (0.9791) and BiLSTM (0.9726), similar to ResNet (0.9719), and is slightly better than LSTM (0.9637), LSTM-AT (0.9685), RETAIN (0.9559), and ProSeNet (0.9596).

Table 7 presents a prototype of SCNPRO, and its closest neighbors in the testing set. The highlighted areas are the most similar segments to the prototype. Although there are a few mismatches, the overall similarities of these segments to the prototype are reasonably high. All of these neighbors belong to the MnmG family, on which the prototype has the largest weight (4.26). Thus, the prototype may represent a relevant structure of the MnmG family.

4.5 More Details on Effectiveness

4.5.1 Prototype distribution. As discussed in Sec. 3.2.1, SCNPRO manipulates prototypes to fit the distribution of segment features. To understand how does it work, we uniformly sampled a batch of training data for visualization (the full set is too large to be visualized). Then, in each sequence, we extracted the segment that is closest to some prototype. The representations of all extracted segments and all prototypes were visualized using tSNE [34] in a 2D space.

Fig. 3 shows the latent space for one prototype module (with filter size 50) on PTB dataset. We investigated the difference between when the adaptation regularizer ℓ_d is on and off in the objective in Eq. (11). In the figure, circles represent segments, "+" markers represent prototypes. The colors indicate classes. From Fig. 3(a), we observe our method fits prototypes to data well and encourages a clustering structure of different dense areas. From Fig. 3(b), it is interesting to see fitted areas even when ℓ_d is off. This phenomenon attributes to the similarity based classification, which inherently associates segments with prototypes. However, the data distribution is not well encoded by prototypes, especially in the central area, where many segments are far from any prototypes thus are hard to be interpreted. In contrast, every segment in Fig. 3(a) is tightly associated with a few prototypes, which enhances the interpretation of their corresponding input sequences.

Table 8: Ablation analysis.

Model	MIT-BIH	PTB	UniProtKB
SCNPRO	0.9741	0.9643	0.9709
(a) Set $\lambda_1 = 0$	0.9758	0.9646	0.9590
(b) Set $\lambda_2 = 0$	0.7328	0.8462	0.7599
(c) Set $\lambda_3 = 0$	0.9749	0.9626	0.9406

4.5.2 Impacts of the number of prototypes. Next, we evaluate how would the number of prototypes per class, $\hat{k}$, influence the performance of SCNPRO using MIT-BIH and UniProt datasets. Keeping other configurations the same as in Sec. 4.3 and 4.4, we varied $\hat{k}$ and evaluated SCNPRO on the testing set. As shown in Fig. 4, the accuracy improves quickly as $\hat{k}$ increases and becomes stable after $\hat{k}$ exceeds 10 and 5 for MIT-BIH and UniProtKB. When $\hat{k}$ is small (*e.g.*, $\hat{k} = 1$), the prototypes cannot adapt to the distribution well, which results in vague boundaries between classes. As $\hat{k}$ increases, prototypes fit clusters tightly, which helps separate classes. The results also suggest that usually $\hat{k}$ can be selected with a few grid-searches using validation sets, since often a small $\hat{k}$ is sufficient.

4.5.3 Ablation analysis. Table 8 summarizes the testing results of our ablation analysis using three datasets. In (a)-(c), we alternately removed the regularizers in Eq. (12) by setting $\lambda_1, \lambda_2, \lambda_3$ to 0, respectively, to investigate their impacts. First, in (a) and (c), we observe removing clustering ($\lambda_1 = 0$) or diversity ($\lambda_3 = 0$) regularizers has small impact on the accuracy on MIT-BIH and PTB datasets, while degrades the performance on UniProtKB. Thus both of them can be safely integrated into the objective ℓ in Eq. (11) as they are essential to enhance the interpretability (as demonstrated by Fig. 3). Removing the second loss, however, causes sharp degradations on all datasets. This is because, as discussed in Sec. 3.3, the last projection step in Eq. (13) may be confused when the prototypes are not optimized to be associated with any segment features by SGD. Incorrect projections, in turn, results in disordered classification and significant performance drop. Thus, the results justify the design of our learning objective.

5 CONCLUSION

Interpretability is important in modern machine learning systems. In this paper, We propose a neural sequence modeling approach SCNPRO, which has a transparent reasoning process on what it predicts. SCNPRO builds upon convolutional architectures, incorporates prototype modules, and exerts filters to learn prototypes that represent local patterns for reasoning its predictions. The learning objective reflects the goal on both accuracy and interpretability, which delicately adapts prototypes to the distribution of data in latent spaces. The extensive experiments on datasets in various domains demonstrate SCNPRO's general ability on modeling sequences, its state-of-the-art performance, and its succinct interpretations.

REFERENCES

[1] Dimitrios Alikaniotis, Helen Yannakoudakis, and Marek Rei. 2016. Automatic Text Scoring Using Neural Networks. In *ACL*. 715–725.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. (2015).

[3] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. 2018. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271* (2018).

[4] Joost Bastings, Wilker Aziz, and Ivan Titov. 2019. Interpretable neural predictions with differentiable binary variables. *arXiv preprint arXiv:1905.08160* (2019).

[5] Steven Bird, Edward Loper, and Ewan Klein. 2009. *Natural Language Processing with Python*. O'Reilly Media Inc.

[6] R Bousseljot, D Kreiseler, and A Schnabel. 1995. Nutzung der EKG-Signaldatenbank CARDIODAT der PTB über das Internet. *Biomedizinische Technik/Biomedical Engineering* 40, s1 (1995), 317–318.

[7] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165* (2020).

[8] Rich Caruana, Yin Lou, Johannes Gehrke, Paul Koch, Marc Sturm, and Noemie Elhadad. 2015. Intelligible models for healthcare: Predicting pneumonia risk and hospital 30-day readmission. In *KDD*. 1721–1730.

[9] Zhengping Che, Sanjay Purushotham, Robinder Khemani, and Yan Liu. 2016. Interpretable deep models for ICU outcome prediction. In *AMIA Annual Symposium*, Vol. 2016. American Medical Informatics Association, 371.

[10] Chaofan Chen, Oscar Li, Daniel Tao, Alina Barnett, Cynthia Rudin, and Jonathan K Su. 2019. This looks like that: deep learning for interpretable image recognition. In *NeurIPS*. 8928–8939.

[11] Edward Choi, Mohammad Taha Bahadori, Jimeng Sun, Joshua Kulas, Andy Schuetz, and Walter Stewart. 2016. Retain: An interpretable predictive model for healthcare using reverse time attention mechanism. In *NeurIPS*. 3504–3512.

[12] UniProt Consortium. 2019. UniProt: a worldwide hub of protein knowledge. *Nucleic Acids Res.* 47, D1 (2019), D506–D515.

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

[14] ANSI-AAMI EC57. 1998. Testing and reporting performance results of cardiac rhythm and ST segment measurement algorithms. *Association for the Advancement of Medical Instrumentation* (1998).

[15] Alan H Gee, Diego Garcia-Olano, Joydeep Ghosh, and David Paydarfar. 2019. Explaining deep classification of time-series data with learned prototypes. *arXiv preprint arXiv:1904.08935* (2019).

[16] Jonas Gehring, Michael Auli, David Grangier, and Yann Dauphin. 2017. A Convolutional Encoder Model for Neural Machine Translation. In *ACL*. 123–135.

[17] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. 2017. Convolutional sequence to sequence learning. In *ICML*. 1243–1252.

[18] Ary L Goldberger, Luis AN Amaral, Leon Glass, Jeffrey M Hausdorff, Plamen Ch Ivanov, Roger G Mark, Joseph E Mietus, George B Moody, Chung-Kang Peng, and H Eugene Stanley. 2000. PhysioBank, PhysioToolkit, and PhysioNet: components of a new research resource for complex physiologic signals. *Circulation* 101, 23 (2000), e215–e220.

[19] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep learning*. MIT press.

[20] Josif Grabocka, Nicolas Schilling, Martin Wistuba, and Lars Schmidt-Thieme. 2014. Learning time-series shapelets. In *KDD*. 392–401.

[21] Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, and Alexander Smola. 2012. A kernel two-sample test. *J. Mach. Learn. Res.* 13, 1 (2012), 723–773.

[22] Peter Hase, Chaofan Chen, Oscar Li, and Cynthia Rudin. 2019. Interpretable Image Recognition with Hierarchical Prototypes. In *HCOMP*, Vol. 7. 32–40.

[23] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *CVPR*. 770–778.

[24] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* (2015).

[25] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Comput.* 9, 8 (1997), 1735–1780.

[26] Xisen Jin, Junyi Du, Zhongyu Wei, Xiangyang Xue, and Xiang Ren. 2019. Towards Hierarchical Importance Attribution: Explaining Compositional Semantics for Neural Sequence Models. *arXiv preprint arXiv:1911.06194* (2019).

[27] Mohammad Kachuee, Shayan Fazeli, and Majid Sarrafzadeh. 2018. Ecg heartbeat classification: A deep transferable representation. In *ICHI*. IEEE, 443–444.

[28] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. 2014. A Convolutional Neural Network for Modelling Sentences. In *ACL*. 655–665.

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

[30] Yoon Kim. 2014. Convolutional Neural Networks for Sentence Classification. In *EMNLP*. 1746–1751.

[31] Janet L Kolodner. 1992. An introduction to case-based reasoning. *Artif. Intell. Rev.* 6, 1 (1992), 3–34.

[32] Oscar Li, Hao Liu, Chaofan Chen, and Cynthia Rudin. 2018. Deep learning for case-based reasoning through prototypes: A neural network that explains its predictions. In *AAAI*.

[33] Mingsheng Long, Yue Cao, Jianmin Wang, and Michael Jordan. 2015. Learning transferable features with deep adaptation networks. In *ICML*. 97–105.

[34] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *J. Mach. Learn. Res.* 9, Nov (2008), 2579–2605.

[35] David Alvarez Melis and Tommi Jaakkola. 2018. Towards robust interpretability with self-explaining neural networks. In *NeurIPS*. 7775–7784.

[36] Yao Ming, Panpan Xu, Huamin Qu, and Liu Ren. 2019. Interpretable and steerable sequence learning via prototypes. In *KDD*. 903–913.

[37] George B Moody and Roger G Mark. 2001. The impact of the MIT-BIH arrhythmia database. *IEEE Eng. Med. Biol.* 20, 3 (2001), 45–50.

[38] W James Murdoch, Peter J Liu, and Bin Yu. 2018. Beyond word importance: Contextual decomposition to extract interactions from LSTMs. *arXiv preprint arXiv:1801.05453* (2018).

[39] W James Murdoch and Arthur Szlam. 2017. Automatic rule extraction from long short term memory networks. *arXiv preprint arXiv:1702.02540* (2017).

[40] Jianmo Ni, Jiacheng Li, and Julian McAuley. 2019. Justifying recommendations using distantly-labeled reviews and fine-grained aspects. In *EMNLP-IJCNLP*. 188–197.

[41] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. 2016. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499* (2016).

[42] Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep Contextualized Word Representations. In *NAACL-HLT*. 2227–2237.

[43] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. " Why should i trust you?" Explaining the predictions of any classifier. In *KDD*. 1135–1144.

[44] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2018. Anchors: High-precision model-agnostic explanations. In *AAAI*.

[45] Cynthia Rudin. 2019. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nat. Mach. Intell.* 1, 5 (2019), 206–215.

[46] Chandan Singh, W James Murdoch, and Bin Yu. 2018. Hierarchical interpretations for neural network predictions. *arXiv preprint arXiv:1806.05337* (2018).

[47] Hendrik Strobelt, Sebastian Gehrmann, Hanspeter Pfister, and Alexander M Rush. 2017. Lstmvis: A tool for visual analysis of hidden state dynamics in recurrent neural networks. *IEEE Trans. Vis. Comput. Graph.* 24, 1 (2017), 667–676.

[48] Yi Tay, Anh Tuan Luu, and Siu Cheung Hui. 2018. Multi-pointer co-attention networks for recommendation. In *KDD*. 2309–2318.

[49] Sarah Wiegreffe and Yuval Pinter. 2019. Attention is not not Explanation. In *EMNLP-IJCNLP*. 11–20.

[50] Jingyuan Yang, Chuanren Liu, Mingfei Teng, Ji Chen, and Hui Xiong. 2017. A unified view of social and temporal modeling for B2B marketing campaign recommendation. *IEEE Trans. Knowl. Data. Eng.* 30, 5 (2017), 810–823.

[51] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. In *NeurIPS*. 5753–5763.

[52] Lexiang Ye and Eamonn Keogh. 2009. Time series shapelets: a new primitive for data mining. In *KDD*. 947–956.