SeqCondenser: Inductive Representation Learning of Sequences by Sampling Characteristic Functions

Maixent Chenebaux[1] and Tristan Cazenave[2]

[1]Vectors Group, Paris, France
[2]LAMSADE, Université Paris Dauphine - PSL, CNRS, Paris, France

Abstract. In this work, we introduce SeqCondenser, a neural network layer that compresses a variable-length input sequence into a fixed-size vector representation. The SeqCondenser layer samples the empirical characteristic function and its derivatives for each input dimension, and uses an attention mechanism to determine the associated probability distribution. We argue that the features extracted through this process effectively represent the entire sequence and that the SeqCondenser layer is particularly well-suited for inductive sequence classification tasks, such as text and time series classification. Our experiments show that SCoMo, a SeqCondenser-based architecture, outperforms the state-of-the-art inductive methods on nearly all examined text classification datasets and also outperforms the current best transductive method on one dataset.

1 Introduction

Text classification is a crucial task in natural language processing (NLP). It can be used for various purposes, such as opinion mining, sentiment analysis, fact checking and detecting fake news. One challenge in classifying text is the variability in the length of the sequences. There are several approaches to representing text for classification, including representing it as a sequence of words or characters, as a fixed-size vector, or as a graph [18].

Transductive learning techniques are currently the top performers in text classification. These techniques generally use a network containing both the training set and the testing set, without labels for the latter. Patterns observed in unlabeled data also contribute to the classification process, leading to better performance. ROBERTaGNN [12], a transductive document classification algorithm, is currently the state-of-the-art on many reference datasets. It fine-tunes BERT [6] and uses the generated vectors as node features, then builds a graph where nodes are text units and edges are based on semantic similarity. While this method leverages both the transformer architecture and graph neural networks, it has the same limitations as other transductive techniques, including longer training time, limited generalization, and deployment challenges. Transductive methods also require retraining when dealing with new data points, making them difficult to use in a production environment where documents are classified individually or in batches asynchronously.

Inductive learning is a powerful and versatile approach to machine learning, where the model is trained on a labeled dataset and then makes predictions on unseen data. Unlike transductive learning, inductive learning does not rely on access to the test set

during training, making it more suitable for real-world applications where new data continuously arrives. Inductive methods are diverse, with different algorithms achieving better results on different datasets. In recent years, text classification has typically been done by fine-tuning a pre-trained model, such as a transformer-based model, and using aggregation methods, such as average pooling, max over time, or attentive pooling. Alternatively, the BERT model can perform prediction directly through the special token [CLS]. These methods have been successful, but their limitation lies in the ability of the pooling techniques to retain essential information for classification while keeping the dimensionality of the final representation as low as possible.

Present work: We propose SeqCondenser, a layer that transforms a varying-size sequence of vectors into a compact, fixed-size vector representation that can be used for sequence classification and regression. The benefits of SeqCondenser are as follows: (1) it is fast to train; (2) it is inductive by design; (3) it can easily be added to any TensorFlow model [1] with a single line of code, making all the results presented in this work easily reproducible; (4) it does not need to transform the input data into a graph or another datastructure, and can work on any sequence. The compact representation of the sequence is efficient at classification tasks, and achieves new state-of-the-art performances when compared to both inductive and transductive methods. On all the datasets studied, the direct replacement of any pooling layer with SeqCondenser leads to an immediate performance increase.

Main contributions: The main contributions of our work are as follows:

1. We introduce a new sequence-to-vector method that relies on a family of characteristic functions and their two subsequent derivatives, where the affiliation probabilities are predicted by a differentiable attention mechanism.
2. We show that dimensionality reduction through random projections can be applied to the SeqCondenser's aggregation vector to improve accuracy.
3. We experimentally demonstrate that this method can be successfully applied to text classification tasks and improve the current state-of-the-art for both inductive and transductive techniques.
4. Our model is implemented using TensorFlow, and an easy-to-use SeqCondenser layer is made publicly available for Keras [4].

The paper is structured as follows: Section 2 reviews related literature. The SeqCondenser layer and its theoretical foundations are discussed in Section 3. The design and experimental results of SCoMo, a sequence classification architecture utilizing the SeqCondenser layer, are outlined in Sections 4 and 5 respectively. The paper concludes in Section 6.

2 Related Work

Pooling strategies: In text classification, the input is typically a variable-length sequence of words or characters. Summarizing its features through pooling strategies is an essential step for the task. Pooling, in this context, aims at extracting features from the input and creating a fixed-length representation of it. One approach is to use convolutional filters to extract features from the input text, and apply average or max over time pooling to summarize semantic features [3]. However, average and max pooling have the limitation

of not being able to efficiently select salient features that may be particularly useful for the classification task.

To address this limitation, several approaches have been proposed. One technique is to create document representations by summing word embeddings weighted by their corresponding TF-IDF scores, as proposed in [11]. Another approach is to use attentive pooling, which allows the neural network to learn to distinguish important features and create a summary that focuses on relevant information, while ignoring irrelevant information. One example of attentive pooling is the DeepMoji attention layer [8], which creates a dense representation of a sequence of vectors by using attention scores as weights for a weighted summation over all the time steps. These weights are calculated by taking the scalar product of each input vector with a single trainable vector and then applying the softmax function along the time dimension. Another example of an attentive pooling technique is APLN [17], which uses a similar approach to DeepMoji but interprets the weighted sum as an $\mathcal{L}^1$ norm. APLN allows the model to learn the most appropriate $\mathcal{L}^p$ norm for the classification task.

Characteristic functions: Characteristic functions are mathematical functions that describe the probability distribution of random variables. In the field of graph representation learning, characteristic functions have been used for summarizing information and creating node embeddings. Two notable approaches that utilize characteristic functions are GraphWave [7] and FEATHER [15].

GraphWave is a method for capturing structural roles in graphs using heat diffusion wavelets. Nodes with the same structural role have similar heat diffusion patterns, and the characteristic function is used to summarize the histogram of heat distribution across the network. The node embeddings created through this method are independent of the size of the network, and roles are predicted through the use of PCA and a clustering algorithm applied to the embeddings.

FEATHER is an approach for creating node embeddings in a network that encodes the distribution of features among the neighbors of a given node. It does this by evaluating the characteristic function at differentiable evaluation points, where the probability distribution of the features is deterministically defined in terms of random walks on the graph. This allows FEATHER to capture the salient features in the neighborhood of a given node and create a fixed-length representation of it.

Overall, characteristic functions have proven to be a powerful tool for summarizing information and creating fixed-length representations of data in the context of graph representation learning. In our work, we aim to leverage the power of characteristic functions by incorporating them into an end-to-end trainable sequence classification pipeline.

3 SeqCondenser

In this section, we present the mathematical foundations and rationale for the SeqCondenser layer. A visual representation of the method is provided in Figure 2.

3.1 Characteristic Functions on Sequences

The characteristic function is a widely-used concept in probability theory that provides a comprehensive description of the probability distribution of any discrete or continuous

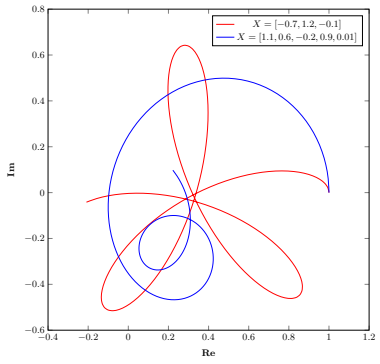

Fig. 1. Graph of two characteristic functions computed from a variable-length list of numbers, where each outcome is assigned an equal probability, showing a partial view of the complex plane representation

random variable X [9]. The kth derivative of the characteristic function, ϕ, at zero is proportional to the kth moment of X: $\phi^{(k)}(0) = i^k \mathbf{E}\left[X^k\right]$. The curve described by the characteristic function lives in the complex plane and theoretically contains all the information needed to describe X. In the discrete case, it can map a random variable with any finite number of outcomes to a curve in the complex plane, as long as a probability is assigned to each outcome. This concept is illustrated in Figure 1. One way to construct a fixed-size representation of the curve of ϕ is to sample it at different points, however this can result in a loss of important information about the curve's shape. For example, the GraphWave and FEATHER graph algorithms only consider $\phi^{(0)}$. We argue that the use of higher-order derivatives of the characteristic function provides crucial information for the learning process.

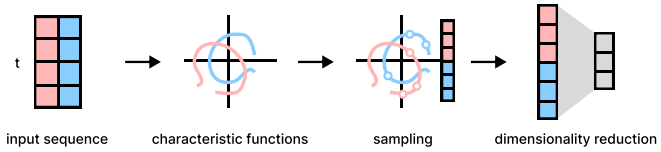

Fig. 2. Main steps of SeqCondenser

3.2 Sequence Feature Distribution Characterization

Let $\mathbf{x}$ be a sequence vector of length l where x_t is the feature value at time step t. We interpret $\mathbf{x}$ as the outcomes of a discrete random variable X. The characteristic function of X is defined as:

$$\phi(\theta) = \mathbf{E}[e^{iX\theta}] = \sum_t p_t e^{i\theta x_t} \tag{1}$$

In Equation 1, the affiliation probability p_t describes the probability of observing the outcome x_t. The Euler identity applied to Equation 1 yields: $\mathbf{Re}\left(\phi\left(\theta\right)\right) = \sum_t p_t \cos\left(\theta x_t\right)$ and $\mathbf{Im}\left(\phi\left(\theta\right)\right) = \sum_t p_t \sin\left(\theta x_t\right)$

We are also interested in the first and second derivatives of the characteristic function, as they provide the model with more information to fully characterize the sequence feature distribution. In this way, the model not only uses the position of the characteristic function in the complex plane at specific points, but also the direction, speed, and curvature of its trajectory.

It is worth noting that the characteristic function and its derivatives are independent of the length of the observed sequence, but can still uniquely describe it. This is the main idea behind SeqCondenser's ability to compactly represent variable-length sequences. It is also important to note that the characteristic function does not consider the order of the sequence. It is advisable to include a position-aware layer before SeqCondenser for effective classification.

3.3 Characterization of Multiple Features

In the case of a sequence with multiple features, we require a set of independent characteristic functions and affiliation probabilities for each feature k. This leads to the following definitions:

- $\mathbf{X} \in \mathbb{R}^{(l,d)}$ is the observed sequence of vectors, where l is the length of the sequence and d is the number of features.
- $\mathbf{x}^{(k)} \in \mathbb{R}^{l}$ is the observed sequence of values for the kth feature, i.e. the kth column of $\mathbf{X}$.
- $p_t^{(k)}$ is the affiliation probability associated with the random variable $X^{(k)}$.
- $x_t^{(k)}$ is the value of the kth feature at time step t.
- $\phi^{(k)}$ is the complex-valued characteristic function associated with the random variable $X^{(k)}$.
- $\phi'^{(k)}$ and $\phi''^{(k)}$ its two derivatives.

3.4 Sampling of the Characteristic Functions and their Derivatives

The characteristic function is a complex-valued function that takes a real input, and therefore cannot be directly used in a neural model. In order to incorporate the characteristic function into our model, we need to discretize it. Similar to the FEATHER approach, we allow the model to choose the evaluation points of the characteristic functions that are the most discriminative for the downstream classification task.

The evaluation points for the d characteristic functions are represented by a matrix $\Theta \in \mathbb{R}^{(d,n_\theta)}$, where n_θ is the number of sampling points.

3.5 Attention Mechanism to compute Probabilities

In order to compute the affiliation probabilities $\mathbf{P}$ for the input sequence, we employ an attention mechanism using a two-layered perceptron. The perceptron takes the entire sequence of vectors $\mathbf{X}$ as input and returns a matrix with the same shape, with the softmax function applied column-wise. This results in a matrix with positive columns that sum to 1, representing the probabilities of all observed features. $\mathbf{P}$ is calculated using the following formula: $\mathbf{P} = \text{softmax}(\mathbf{t} \odot \text{ReLU}(\text{ReLU}(\mathbf{X}W_1 + \mathbf{b})W_2 + \mathbf{c}))$ where the softmax is applied column-wise, $W_1, W_2 \in \mathbb{R}^{(d,d)}$ are two parameter matrices,

$\mathbf{b}, \mathbf{c} \in \mathbb{R}^d$ are two bias vectors, and $\mathbf{t} \in \mathbb{R}^d$ is a temperature vector that allows the model to independently and quickly adjust the softmax discriminatory power for each dimension. $\odot$ denotes element-wise multiplication. In our notation, $\mathbf{P}_{tk} = p_t^{(k)}$.

3.6 Aggregation of the Evaluations

We define $\psi^{(k)}$ as a weighted average of the kth characteristic function and its two derivatives applied at its corresponding evaluation points:

$$\psi^{(k)} = \alpha_1 \phi^{(k)}(\Theta^{(k)}) + \alpha_2 \phi'^{(k)}(\Theta^{(k)}) + \alpha_3 \phi''^{(k)}(\Theta^{(k)})$$

where α is a softmaxed trainable 3-dimensional vector. Combining the results of the three functions has two advantages: (1) it reduces the memory footprint of the method; (2) it allows the model to balance between the various descriptions of the complex trajectory depending on their discriminative power for the task.

This gives the following formulas for the real and imaginary parts of $\psi^{(k)}$:

$$\mathbf{Re}(\psi^{(k)}(\theta)) = \sum_{x_t} p_t^{(k)} \left[(\alpha_1 - \alpha_3 x_t^2)\cos(x_t\theta) - \alpha_2 x_t \sin(x_t\theta)\right]$$

$$\mathbf{Im}(\psi^{(k)}(\theta)) = \sum_{x_t} p_t^{(k)} \left[(\alpha_1 - \alpha_3 x_t^2)\sin(x_t\theta) + \alpha_2 x_t \cos(x_t\theta)\right]$$

The real and imaginary parts of the $\psi^{(k)}$ are then concatenated into a compact vector $\mathbf{h}$ of size $2 \cdot d \cdot n_\theta$: $\mathbf{h} = \bigoplus_{k=1}^{d} \left[\mathbf{Re}(\psi^{(k)}) \oplus \mathbf{Im}(\psi^{(k)})\right]$ with $\oplus$ the concatenation operation.

3.7 Dimensionality Reduction using Random Projections

Using $\mathbf{h}$ as the final vector of the layer can be challenging due to its large size. For example, with 10 evaluation points and a sequence of 500-dimensional embeddings, $\mathbf{h}$ has a size of 10,000. One approach to addressing this dimensionality issue is to use techniques such as Principal Component Analysis (PCA), as seen in GraphWave. However, this approach is not suitable for use in an end-to-end training task. Our solution is to multiply $\mathbf{h}$ with a fixed, non-trainable, randomly initialized, and column-orthogonal matrix $O \in \mathbb{R}^{(2 \cdot d \cdot n_\theta, r)}$, where r is the desired reduced dimensionality. This approach significantly improves the accuracy of the model (Table 3).

Choosing r to be equal to the input dimension d generally works well, although we find that lower values do not significantly penalize the model (Figure 4). As a final step, we apply the ReLU activation function after adding a scaler and a bias vector to the aggregation, giving: $\mathbf{h} \leftarrow \text{ReLU}\left(\beta(\mathbf{h}^T O) + \gamma\right)$.

3.8 Mathematical properties

Considering the first derivative of the characteristic function has an interesting mathematical property, summarized by the following proposition:

Proposition 1. *Let* $\mathbf{x}$ *and* $\mathbf{x}'$ *be two vectors representing two sequences. Assume that the second sequence differs from the first at a single element at time step* u*, where* $x'_u = -x_u$*, and that the associated probability is kept unchanged, i.e.* $p'_u = p_u$*. The modulus of the difference between their two characteristic functions is bounded by* $2p_u$*.*

Proof. We start by expressing the difference between the two characteristic functions as follows:

$$\begin{aligned}\Delta m_\phi &= \left|\sum_t p_t e^{ix_t\theta} - \sum_t p'_t e^{ix'_t\theta}\right| = \left|p_u e^{ix_u\theta} - p'_u e^{i(x'_u)\theta}\right| \\ &= \left|p_u e^{ix_u\theta} - p_u e^{-ix_u\theta}\right| = |p_u(e^{ix_u\theta} - e^{-ix_u\theta}) = |p_u(2i \cdot sin(x_u\theta))|\end{aligned}$$

Using the fact that the absolute value of $sin(x)$ is always less than or equal to 1, we get:

$$\Delta m_\phi \leq 2p_u$$

This implies that the maximum absolute difference between the two characteristic functions does not depend on the magnitude of x_u, leading to the inability of the model to distinguish opposite features when their probability weights have similar values. To address this issue, the first derivative of ϕ can be used. Carrying out a similar calculation on the derivative of the characteristic function gives us:

$$max(\Delta m_{\phi'}) = max\,|2p_u x_u cos(x_u\theta)| = 2p_u|x_u|$$

This result shows that the maximum absolute difference between the derivative of the characteristic functions is dependent on the magnitude of x_u. This property improves the model's expressivity by enabling it to better differentiate between positive and negative values.

4 Text Classification Model Architecture

In this section, we describe the neural network models used for text classification. We introduce two models: the SeqCondenser Model for Classification (referred to as SCoMo) and a simplified model called SCoMo-Bare, which is mainly used for comparison. Before discussing their architectures, we first introduce the units that make up these models.

4.1 Embeddings

As with most models that process text sequences, the input tokens are converted into d-dimensional embeddings. To perform this conversion and learn the representations automatically, the Keras Embedding layer is used as the first unit in the SeqCondenser models.

4.2 Self Attention

SCoMo uses a simplified version of the popular Self Attention layer [16]. Instead of using three different sets of projection matrices (Key, Query, and Value), we only use one. For a sequence X with shape (l, d), we compute the attention score matrix of shape (l, l) using the following formula: $Scores(X) = \text{softmax}(t \cdot \text{ReLU}(XWX^T + b))$ where the softmax function is applied along the rows, $W \in \mathbb{R}^{(d,d)}$ is a trainable matrix, b is a bias scalar, and t is a temperature scalar that controls the strength of the softmax.

The output of the simplified Self Attention layer is then given by: $SA(X) = Scores(X)X$

4.3 Positional Encoding

The characteristic function is insensitive to the order of the elements in the sequence. To make the model position-aware, we let the model learn its own positional embeddings. $PE(X) = X + \text{SWISH}(W)$ where X is the input sequence of shape (l, d), $W \in \mathbb{R}^{(l,d)}$ is a trainable parameter matrix, and $SWISH$ is an activation function used to avoid vanishing gradients [14]. The $SWISH$ function was chosen early on in our experiments because it provided better classification performance.

4.4 Model Architecture

The SCoMo model consists of a Positional Encoding layer, Self Attention layers, a SeqCondenser unit and a final dense layer for classificaton.

To buildS SCoMo-Bare, we follow the same structure as SCoMo, but remove the Self Attention layers and the Positional Encoding unit. The output of the Embedding layer is directly passed through the SeqCondenser layer, and the resulting fixed-size vector is used for classification. Despite its simplicity and disregard for the order of the sequence, SCoMo-Bare performs very well in comparison to other models, demonstrating the effectiveness of the SeqCondenser layer in capturing relevant information from the input sequence.

5 Experiments

In this section, we present experimental results to demonstrate the efficacy of our model.

5.1 Experiments Setups

For the text classification task, we run our experiments on five well-established text classification datasets, namely 20 Newsgroups (20NG), R8, R52, Ohsumed and Movie Review (MR). The 20 Newsgroups dataset consists of approximately 20,000 newsgroup posts from 20 different categories, and is widely used for multi-label classification tasks. The R8 and R52 datasets are collections of Reuters news articles, with 8 and 52 categories, respectively. The Ohsumed dataset is a collection of medical abstracts with 23 categories. The Movie Review dataset consists of movie reviews with binary labels indicating positive or negative sentiment. We use the standard splits for all datasets. Table 1 provides a numerical summary of the benchmarks.

Preprocessing was kept to a minimum. Documents were tokenized and truncated at 500 tokens if they exceeded that length. All tokens and punctuation were retained and no stopwords were removed. No pretrained word embeddings or language models were used.

We compare the performance of the proposed SeqCondenser architecture with two state-of-the-art inductive models: GraphStar [13] and SparseTensorClassifier (referred to as STC hereafter) [10]. As baselines, we used four popular algorithms implemented in Scikit-Learn [2]: Support Vector Machine (SVM), Multinomial Naive Bayes (MNB), Random Forest (RF), and K-Nearest Neighbors (KNN). Text vectorization for these models was performed using TF-IDF, a widely used approach for representing text data.

To compare the effectiveness of the SeqCondenser unit to other pooling methods, three models were built: SAt-avg, SAt-max, and SAt-weighted. The prefix 'SAt' stands for 'Self-Attention'. These models have identical architecture as SCoMo, with the only difference being the method of aggregation they use:

- **SAt-avg** aggregates the sequence by taking its average over the temporal dimension.
- **SAt-max** uses max over time pooling.
- **SAt-weighted** computes a weighted summation of the sequence, where the weights are predicted using an attention mechanism similar to DeepMoji.

Since no pretraining was utilized, both SCoMo and SAt models were kept small, incorporating only two Self-Attention layers for the experiments.

Table 1. Summary of Text Classification Datasets.

DATASET	N_{train}	N_{test}	CLASSES
20NEWSGROUPS	11,314	7,532	20
R8	4,937	2,189	8
R52	5,879	2,568	52
OHSUMED	3,021	4,043	23
MR	7,108	3,554	2

5.2 Settings

We kept the SeqCondenser model's architecture and parameter sizes consistent across all datasets to showcase its 'drop-in replacement' capabilities. Both the token embedding dimension d and the reduction dimension r were set to 500. To maintain a low model size, we used only 2 self-attention layers and 10 evaluation points. The matrix Θ was initialized by linearly interpolating values between -10 and 100. While fine-tuning the trainable weights or increasing the model size might have improved accuracies, this was not explored in this paper.

All models were trained using the RMSProp optimizer [5] and minimizing cross-entropy loss, with a learning rate of 0.001 and a batch size of 30 for 10 epochs.

5.3 Main Results

Table 2 shows that SCoMo outperforms all other inductive algorithms on all datasets except one. SeqCondenser also performs significantly better than all other pooling strategies, despite all models having otherwise the same architecture. This demonstrates that replacing a traditional pooling layer with SeqCondenser can significantly improve model performance. Additionally, using a SeqCondenser layer immediately after an embedding layer without self-attention or positional embedding (SCoMo-Bare) still outperforms more sophisticated models. At the time of writing, SCoMo achieves the highest reported accuracy on the R8 dataset according to the Papers with Code leaderboard, outperforming the transductive text classification algorithm RoBERTaGCN.

Table 2. Mean accuracies over 10 runs for different models. The best accuracies for inductive methods are in bold, and the best overall accuracies are in italic.

MODEL	20NG	R8	R52	OH	MR
SVM	77.7	94.6	88.9	48.6	75.5
MNB	73.6	83.6	71.2	29.8	**77.8**
RF	74.2	92.8	84.8	57.3	68.7
KNN	52.4	87.0	83.5	54.5	70.2
SAT-MAX	77.6	96.6	92.1	65.1	73.2
SAT-AVG	82.7	97.1	92.7	65.0	75.7
SAT-WEIGHTED	81.5	97.0	92.5	63.5	75.5
STC	86.3	95.1	90.9	67.4	75.7
GRAPHSTAR	86.9	97.4	95.0	64.2	76.6
SCOMO-BARE	85.6	97.6	94.8	68.7	76.9
SCOMO	**87.4**	***98.5***	**95.5**	**70.1**	76.6
ROBERTAGCN	*89.5*	98.2	*96.1*	*72.8*	*89.7*

Table 3. Mean accuracies over 10 runs for different dimensionality reduction schemes.

DIMENSIONALITY REDUCTION	R8	R52	OH
NON-TRAINABLE REDUCTION	**98.5**	**95.5**	**70.1**
TRAINABLE REDUCTION	97.8	95.2	66.6
NO REDUCTION	97.9	95.0	67.3

5.4 Importance of Dimensionality Reduction

As explained earlier, we introduce an additional step after the computation of the embeddings of the characteristic functions in order to reduce the size of the output of the aggregation step. This is achieved by multiplying the aggregation vector by a randomly initialized column-orthogonal matrix O with r columns, where r is less than $2dn_\theta$.

We used $r = 500$ in our experiments and evaluated testing accuracies according to the following three modalities:

- **non-trainable reduction:** the agregation vector is multiplied by a randomly initialized but fixed matrix O.
- **trainable reduction:** the agregation vector is multiplied by O, and O is trained alongside the rest of the neural network.
- **no reduction:** the aggregation vector is used as the output of the layer without being multiplied by O.

According to the results presented in Table 3, using a non-trainable O yields the best performance on the three datasets. These results suggest that using a non-trainable matrix is an effective approach for reducing the size of the output of the aggregation step and improving the test accuracy of the neural network.

5.5 Parameters Sensitivity

In this section, we investigate the influence of the two main model parameters on the quality of learning: the number of evaluation points and the size of the dimensionality reduction. To do this, we conducted an analysis of the model's accuracy by varying the number of evaluation points from 1 to 14 on three datasets: Ohsumed, R8, and R52. The

numbers presented in Figure 3 are the average accuracy over 10 runs. It is worth noting that error bars were not included in the graph as they are too small to be legible: the standard deviation of the accuracies is never more than 0.1 and typically falls within the range of 0.03 to 0.06.

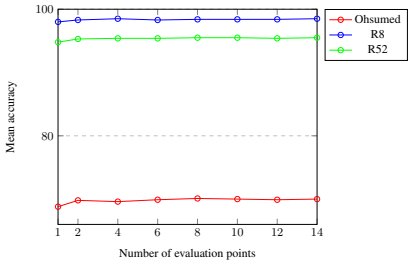

Fig. 3. Impact of the number of evaluation points on accuracy

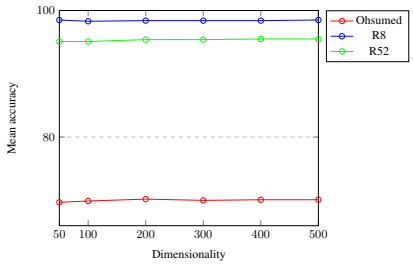

Fig. 4. Impact of the dimensionality reduction on accuracy

Based on the calculations, increasing the number of sampling points significantly improves accuracy from 1 to around 8 points. However, beyond 8 points, the improvement becomes negligible, indicating that the model's accuracy is stable across a range of sampling point values. Thus, using more than 8 evaluation points is unnecessary.

It is important to note that using a sampling point is not the same as using a single parameter. Instead, it represents a sampling point for each input feature, resulting in a $2d$-dimensional vector before reduction, contributing to the robustness of the results in this situation. Moreover, if the accuracies with a sampling point appear similar to larger values on Figure 3, it is primarily a visual artifact of the compression of the y-axis.

In addition, an analysis of the impact of the size of the low-dimensional space that the aggregation vector is projected onto reveals that the model is able to effectively compress information across a range of dimensionality values from 50 to 500. The observed high stability in accuracy across this range demonstrates the effectiveness of the SeqCondenser layer in producing a rich representation for the classification task.

6 Conclusion and Future Work

In this work, we introduced SeqCondenser, a novel TensorFlow layer that transforms any sequence into a fixed-size vector, serving as an alternative to traditional pooling layers. We presented SCoMo, a neural model for classification that incorporates SeqCondenser and achieved state-of-the-art results compared to both inductive and transductive models. We also introduced SCoMo-Bare, a simple model that is unaware of sequence order, yet still outperforms more sophisticated, position-aware models using standard architectures.

The SeqCondenser unit can be integrated into any Keras model for classification or regression tasks. Our experiments demonstrate the effectiveness of the SeqCondenser layer in providing a concise and accurate summary of a sequence. In addition, we found that using a dimensionality reduction step, which involves multiplying the aggregation

vector with a randomly initialized but fixed column-orthogonal matrix, significantly improves the training accuracy. Overall, the use of this dimensionality reduction step was a key factor in achieving the high accuracy and stability observed in our experiments.

Future work could explore the use of SeqCondenser to fine-tune language models such as BERT for classification purposes. We also plan to test SCoMo on classification tasks involving time series data, and preliminary results have been encouraging. We believe that the layer may also be successful in regression tasks, although we have not yet conducted any tests in this area.

References

1. Abadi, M., et al.: TensorFlow: Large-scale machine learning on heterogeneous systems (2015)
2. Buitinck, L., et al.: API design for machine learning software: experiences from the scikit-learn project. In: ECML PKDD Workshop: Languages for Data Mining and Machine Learning. pp. 108–122 (2013)
3. Chen, Y.: Convolutional neural network for sentence classification. Master's thesis, University of Waterloo (2015)
4. Chollet, F., et al.: Keras. https://keras.io (2015)
5. Dauphin, Y.N., de Vries, H., Chung, J., Bengio, Y.: Rmsprop and equilibrated adaptive learning rates for non-convex optimization. (2015)
6. Devlin, J., Chang, M., Lee, K., Toutanova, K.: BERT: pre-training of deep bidirectional transformers for language understanding. CoRR **abs/1810.04805** (2018)
7. Donnat, C., Zitnik, M., Hallac, D., Leskovec, J.: Spectral graph wavelets for structural role similarity in networks. CoRR **abs/1710.10321** (2017)
8. Felbo, B., Mislove, A., Søgaard, A., Rahwan, I., Lehmann, S.: Using millions of emoji occurrences to learn any-domain representations for detecting sentiment, emotion and sarcasm. In: Empirical Methods in Natural Language Processing. pp. 1615–1625 (2017)
9. Feller, W.: An Introduction to Probability Theory and Its Applications, vol. 1. Wiley, New York, third edn. (1968)
10. Guidotti, E., Ferrara, A.: An explainable probabilistic classifier for categorical data inspired to quantum physics (2021), https://arxiv.org/abs/2105.13988
11. Huang, E., Socher, R., Manning, C., Ng, A.: Improving word representations via global context and multiple word prototypes. In: Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics. pp. 873–882 (2012)
12. Lin, Y., Meng, Y., Sun, X., Han, Q., Kuang, K., Li, J., Wu, F.: Bertgcn: Transductive text classification by combining gcn and bert. arXiv preprint arXiv:2105.05727 (2021)
13. Lu, H., Huang, S.H., Ye, T., Guo, X.: Graph star net for generalized multi-task learning. CoRR **abs/1906.12330** (2019)
14. Ramachandran, P., Zoph, B., Le, Q.V.: Swish: a self-gated activation function. arXiv: Neural and Evolutionary Computing (2017)
15. Rozemberczki, B., Sarkar, R.: Characteristic Functions on Graphs: Birds of a Feather, from Statistical Descriptors to Parametric Models. In: CIKM. p. 1325–1334. ACM (2020)
16. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L.u., Polosukhin, I.: Attention is all you need. In: Advances in Neural Information Processing Systems. vol. 30. Curran Associates, Inc. (2017)
17. Wu, C., Wu, F., Qi, T., Cui, X., Huang, Y.: Attentive pooling with learnable norms for text representation. pp. 2961–2970 (01 2020). https://doi.org/10.18653/v1/2020.acl-main.267
18. Yao, L., Mao, C., Luo, Y.: Graph convolutional networks for text classification. CoRR **abs/1809.05679** (2018), http://arxiv.org/abs/1809.05679