COMBINATORIAL CNNS FOR WORDS

Karen Sargsyan
Institute of Chemistry, Academia Sinica, Taipei, Taiwan
{karen.sarkisyan}@gmail.com

ABSTRACT

This paper illustrates the difficulties that deep learning models encounter in detecting and exploiting patterns that remain consistent through one-to-one transformations, which we define as "combinatorial patterns." We contend that providing neural networks with detailed representations of these combinatorial patterns is crucial to effectively addressing prediction problems that depend only on such patterns. To substantiate our proposition, we unveil a new Combinatorial Convolutional Neural Network tailored for word classification. This demonstrates the potential of integrating combinatorial pattern recognition into deep learning architectures.

1 INTRODUCTION

Neural networks may not prioritize combinatorial patterns—those features that remain unchanged under bijective transformations of the input—when selecting features for prediction tasks where outcomes remain invariant to those transformations. This is partly because the training data for these networks is usually unbalanced and does not encompass all potential bijective transformations for each element. For example, in a task of distinguishing cats from dogs, datasets often focus on natural colors and settings found in typical photographs. This approach may not equip the model to identify these animals in images with unusual color schemes caused by bijective transformations. These issues are recognized in domain generalization and adaptation researchZhou et al. (2022).

Certain prediction tasks depend substantially on the recognition of combinatorial patterns. For instance, studies have indicated that similarities in nucleotide patterns within the genomes of viruses and their hosts can reveal insights beyond the mere encoding of amino acids, which may aid in forecasting the emergence of new pathogens (Babayan et al. (2018); Iuchi & et al. (2023)). Likewise, the frequency and placement of certain words within a text can suggest whether those words are key terms (Ortuño & et al. (2002); Najafi & Darooneh (2015)). In the context of RNA, the arrangement of nucleotide sequences can indicate whether they form critical motifs within the RNA structure (Sargsyan & Carmay (2010)). Moreover, to assess the significance of combinatorial patterns for a given predictive task, one might develop a model trained to recognize these patterns, which would complement more traditional predictive approaches.

Recognizing the value of prediction based on combinatorial patterns, one could suggest augmenting the dataset by applying all possible transformations to its elements. Yet, the sheer volume of potential transformations is typically overwhelming if one aims to achieve firm guarantees that only combinatorial patterns are learned. We propose presenting the neural network with data encompassing the full spectrum of combinatorial patterns in the input and allowing the network to determine the most critical combinatorial factors for making predictions. In this paper, we introduce a specific methodology for processing text data in this way.

2 WORDS AND COMBINATORICS

Using notation from Morita & Terui (2009); Morita (2010), α is a word over some alphabet, and $\Omega(\alpha)$ denotes the set of distinct letters appearing in α. We define $S(\alpha)$ as the set of all subwords(i.e. contiguous substrings) of α. By adding an additional empty subword / letter ε to $S(\alpha)$ we construct a new set $W(\alpha) = \{\varepsilon\} \cup S(\alpha)$.

For any two subwords of α, $\lambda = Y_1Y_2...Y_s$ and $\mu = Z_1Z_2...Z_t$, we make the matrix (m_{ij}), where $m_{ij} = Y_i$ if $Y_i = Z_j$ and $m_{ij} = \varepsilon$ otherwise. Using it, we construct the associated graph, whose vertices are (i, j) for all $1 \leqslant i \leqslant s$ and $1 \leqslant j \leqslant t$ and edges $(i, j) \to (k, l)$ if $k = i + 1, l = j + 1, m_{ij} \neq \varepsilon, m_{kl} \neq \varepsilon$. For every connected component of the graph, we construct the corresponding subword α using the values of (m_{ij}) along the graph path starting from the vertex with the lowest index in a given connected component. We further define combinatorics for α, denoted by $M(\alpha)_\nu(\lambda, \mu)$, as the multiplicity of the subword α being observed in connected components of the graph. With an extra set of rules:

$$M(\alpha)_\nu(\lambda, \varepsilon) = \delta_{\nu,\varepsilon} * s, M(\alpha)_\nu(\varepsilon, \mu) = \delta_{\nu,\varepsilon} * t, M(\alpha)_\nu(\varepsilon, \varepsilon) = \delta_{\nu,\varepsilon}, \tag{1}$$

we obtain a map:

$$M(\alpha) : W(\alpha) \times W(\alpha) \times W(\alpha) \to \mathrm{Z}_{\geqslant 0}, (\lambda, \mu, \nu) \longmapsto M(\alpha)_\nu(\lambda, \mu). \tag{2}$$

To demonstrate that two words a and b have the same patterns one has to provide bijection φ from $\Omega(a)$ to $\Omega(b)$ such that $a = X_1X_2...X_r$ and $b = \varphi(X_1)\varphi(X_2)...\varphi(X_r)$. Based on theorems proven in Morita & Terui (2009), we may compare combinatorics $M(\alpha)$ instead.

3 COMBINATORIAL CNNS

We propose to use $M(\alpha)$ as a 3D tensor input for convolutional neural networks (CNN) for an input word α. To demonstrate the feasibility of such an approach, we present two examples of a classification task where only combinatorial patterns matter. In addition, we show that modern neural networks may learn more than just patterns required.

Our first task is to classify 20-letter words into palindromes and non-palindromes. [1]. The validation accuracy for the classification task, obtained using our approach, is 98.8%. As an alternative, we randomly split palindrome dataset for training/testing parts and trained LSTM, GRU, CNN and Transformer to predict palindromes, achieving average validation accuracies of 90%, 89%, 93% and 74% respectively. To check how well conventional neural networks learn combinatorial patterns, we changed the letter A into $\mathring{A}$ (symbol present in encoding and absent in the training data) in each palindrome in the test set. This change resulted, on average, in a modification of 6% letters in the test dataset. We observed statistically significant drops in accuracy of 0.34%/0.6%/1.5% for LSTM/CNN/Transformer on the modified test set. Additionally, introducing an extra letter change to the test set led to drops in accuracy of 0.62%/1.6%/3% compared to the original test dataset. There were no statistically significant changes for GRU.

In the second task, we attempt to distinguish between weak and strong passwords based on combinatorial patterns. We generated 15-character passwords and assessed their strength using the *'password_strength'* Python package, considering passwords with scores above 0.7 on a [0,1] complexity scale as strong. The password strength is defined using entropy, making it a variety and pattern problem. Applying the same combinatorial CNN as in the previous task, we achieved 99% accuracy on the validation set. However, the same LSTM/GRU/CNN/Transformer as above exhibited fast overfitting, with both achieving validation accuracies of approximately 50%. After modifying the test set by replacing a letter in passwords with a previously unseen character, we did not observe a significant change, possibly due to the low validation accuracy achieved.

4 CONCLUSION

We have demonstrated that neural networks, in general, fail to learn from combinatorial patterns alone, especially when trained on a small dataset. To tackle this issue in the context of predicting combinatorial properties of words, we propose the use of combinatorics as an input to CNNs.

ACKNOWLEDGEMENTS

The author would like to thank Jun Morita for pointing out works on the combinatorial description of words and for providing valuable clarifications.

[1] The complete code and datasets are available at https://github.com/karsar/wordsCCNN.

URM STATEMENT

The authors acknowledge that at least one key author of this work meets the URM criteria of ICLR 2024 Tiny Papers Track.

REFERENCES

Simon A. Babayan, Richard J. Orton, and Daniel G. Streicker. Predicting reservoir hosts and arthropod vectors from evolutionary signatures in RNA virus genomes. *Science*, 362(6414):577–580, 2018.

Hitoshi Iuchi and et al. Bioinformatics approaches for unveiling virus-host interactions. *Computational and Structural Biotechnology Journal*, 21:1774–1784, 2023.

Jun Morita. Tilings, Lie theory and combinatorics. *Contemporary Mathematics*, 506:173–185, 2010.

Jun Morita and Akira Terui. Words, tilings and combinatorial spectra. *Hiroshima Math. J.*, 39: 37–60, 2009.

Elham Najafi and Amir H. Darooneh. The fractal patterns of words in a text: A method for automatic keyword extraction. *PLoS One*, 10(6):e0130617, 2015.

Miguel Ortuño and et al. Keyword detection in natural languages and DNA. *Europhys. Lett.*, 57: 759–764, 2002.

Karen Sargsyan and Lim Carmay. Arrangement of 3D structural motifs in ribosomal RNA. *Nucleic Acids Research*, 38:3512–3522, 2010.

Kaiyang Zhou, Ziwei Liu, Yu Qiao, Tao Xiang, and Chen Change Loy. Domain generalization: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2022.

A APPENDIX

DEFINITION OF BIJECTIVE MAPPING APPLIED TO A DATASET

To define a bijective mapping, or one-to-one, applied to a dataset, consider a set of data entries in the dataset, denoted $X_1, X_2, \ldots, X_n$, and a bijective (one-to-one) function, f, with its inputs and outputs being permissible entries in the dataset (potentially unobserved). A bijective mapping applied to a dataset will produce a new one: $f(X_1), f(X_2), \ldots, f(X_n)$. With a bijective function, there exists another function that can restore the original dataset when applied after f.

In this paper, we explore bijective mappings in the context of words, where a one-to-one mapping function replaces each letter in a word with another permissible letter, whether or not it has been observed before.

DATA GENERATION AND TRAINING DETAILS

In our study of a straightforward classification task, we employ basic configurations of the LSTM, GRU, CNN and lightweight Transformer models, as shown in the code, without optimizing for maximum performance. Each model is trained for 100 epochs, and this process is repeated 200 times with various data splits to ensure statistically significant results. Training and validation accuracies plateau by the end of each training cycle. The complete code is provided for reference.

We generate palindromes by first creating random strings and then converting them into palindromes, resulting in a set that might not represent all possible palindromes equally. For non-palindromes, we selected random strings after verifying that they are not palindromes. Both the dataset generation code and the dataset itself are available.

We created the password dataset by generating random words and assessing their strength as passwords using the *'password_strength'* Python tool. We maintained an equal number of weak and strong passwords for the experiment. The dataset generation code and the dataset itself are available.