Solving Analogical Equations Between Strings of Symbols Using Neural Networks

Vivatchai Kaveeta and Yves Lepage

Abstract. A neural network model to solve analogical equations between strings of symbols is proposed. The method transforms the input strings into two fixed size alignment matrices. The matrices act as the input of the neural network which predicts two output matrices. Finally, a string decoder transforms the predicted matrices into the final string output. By design, the neural network is constrained by several characteristics of analogy. The experimental results show a very fast learning rate with a highest prediction accuracy of 95.68% while a baseline analogy solving algorithm gives 94.47% accuracy rate.

1 Introduction

We design a neural network model to solve analogical equations between strings of symbols. A matrix representation is proposed which transforms the input strings into a matrices. Re-sampling and filtering on the matrices are applied. Next, the neural network design is introduced. We exploit characteristics of proportional analogy between strings of symbols to introduce constraints on the neural network. Finally, the output prediction matrices are decoded into the final string output. Experiments are performed to evaluate the effects of configurable parameters. The method is compared with an algorithm based on a formal characterization of proportional analogy between strings of symbols [7].

This paper is organized as follows. Section 2 describes the background on proportional analogy between strings of symbols and previous related works. In Sect. 3, the proposed methods and neural network models are explained. Experiments are detailed in Sect. 4. Section 5 shows the experiment results.

2 Background

Proportional analogy between sequences of symbols, being they phonemes or characters is stated as the relationship between four strings in the form of 'A is to B as C is to D' denoted by $A : B :: C : D$. Analogical equations are the following problems: if three strings A, B and C are given, how to coin the fourth string? Proportional analogies are seen at work to coin new words or new sentences. In this work, we focus on a type of analogies called analogies of commutation[1]. We

[1] One can distinguish between four types of analogies between string of symbols: repetition (e.g., $A : B :: C : D$), reduplication (e.g., $cat : caat :: dog : doog$), mirror (e.g., $abc : wxyz :: cba : zyxw$) and commutation (examples in the text).

do not deal with semantic analogies like: *queen* : *king* :: *woman* : *man*. Rather, the computational analogy directly works on the symbolic level. [7] gives an algorithm to solve analogies of commutation on strings. [10] proposes a similar algorithm. Both algorithms base on the notion of edit distance. In [9], the formalization was successfully applied in the development of an analogy-based machine translation system. In this work we refer to these previous formalizations in designing an appropriate structure for a neural network.

Neural networks have been successfully applied to many tasks. Their main advantage is their ability to learn from examples without predefined knowledge of the problem. Assuming that an appropriate model structure is used, the network can estimate the underlying structure of the problem. Although many neural networks are proposed for different tasks, no specific neural network seems to have ever been proposed to solve analogical equations on strings of symbols. [1, 3] proposed networks to generate new images based on the previous image samples for classification model training. However, these problems are not expressed in the form of analogy equations. In [13], neural networks generate new images by solving analogy equations between images. This is similar to our problem of analogical equations on strings. The successful implementations point at the possibility of developing neural network to solve analogical equations on strings.

3 Proposed Method

In Sect. 3.1, a method to transform input strings into matrices is introduced. The matrices are re-sampled into fixed-size matrices in Sect. 3.2. Two filtering methods are introduced in Sect. 3.3. The neural network is explained in Sect. 3.4. The output matrices are decoded into a final string by the decoder in Sect. 3.5.

3.1 Alignment Matrices

The usual approach for processing strings of characters with neural networks is vector encoding. Dictionary based one-hot-vectors are used in [4, 6, 15]. For the analogy resolution task, vectors could be built at the character level. Unfortunately, this vector representation presents some problems. First, strings are variable in length. Second, dictionary-based vectors are language-specific. These limitations make the usage of one-hot-vector limited to fixed length and specific language strings.

Proportional analogy can be processed by calculating similarities through edit distances. Consequently, a representation for the similarity of strings seems appropriate. Alignment matrices are widely adopted in the genetic sequence alignment task [2]. They encode a pair of sequences into a matrix where each cell represents a local matching point. Figure 1a (left) shows the alignment between the strings 'harder' and 'hard'. Local matching positions with the value of 1.0 are shown as black cells. Unmatched positions with a value of 0.0 are shown as white cells.

3.2 Matrix Re-sampling

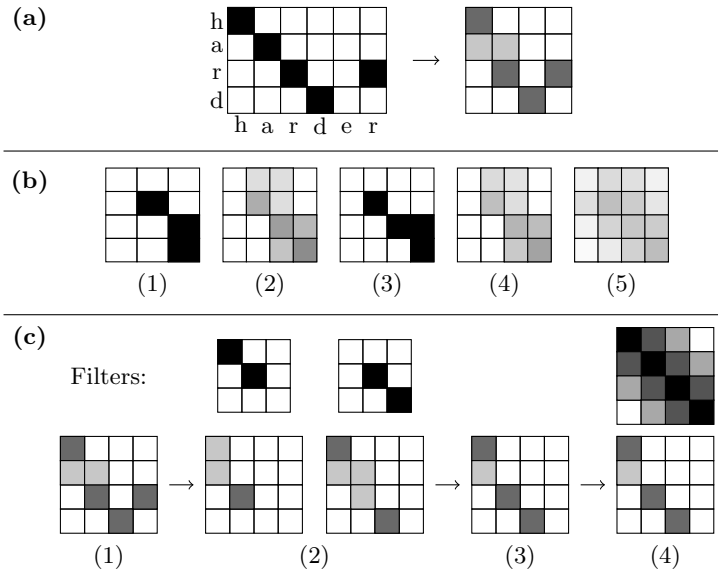

Fig. 1. a An example of alignment matrix. Original (Left) Re-sampled (Right). **b** Comparison between re-sampling methods. Original matrix (1), Proposed (2), Nearest Neighbor (3), Bilinear (4) and Bicubic (5). **c** An example of filtering methods

The alignment matrices can vary in dimension depending on the lengths of the input strings. To feed the alignment data to a neural network, the matrices need to be re-sampled into fixed dimension matrices.

$$\begin{bmatrix} a_1b_1 & \dots & a_mb_1 \\ \vdots & \ddots & \\ a_1b_n & & a_mb_n \end{bmatrix} \Rightarrow \begin{bmatrix} AB_{11} & \dots & AB_{z1} \\ \vdots & \ddots & \\ AB_{1z} & & AB_{zz} \end{bmatrix} \tag{1}$$

$$AB_{xy} = \sum_{i=\lfloor I(y,m) \rfloor}^{\lceil I(y+1,m) \rceil} \sum_{j=\lfloor I(x,n) \rfloor}^{\lceil I(x+1,n) \rceil} (a_ib_j \times f(i,x,n) \times f(j,y,m)) \tag{2}$$

$$f(s,t,u) = \min\left(I(s+1,u), t+1\right) - \max\left(I(s,u), t+1\right), \quad I(w,n) = w \times \frac{z}{n}$$

Input strings A and B with lengths m and n are denoted as $a_1a_2 \dots a_m$ and $b_1b_2 \dots b_n$ respectively. The original alignment matrix (Fig. 1, left) with dimension $m \times n$ has been re-sampled into AB with dimension $z \times z$ (Fig. 1, right). The formula is shown in (2). The matrix AB is using a non-uniform linear transformation on both axes. Figure 1b illustrates this. The difference with other image

re-sampling methods is shown on Fig. 1c. Our method can generate sharp edges while maintaining the diagonal line visible.

3.3 Filtering

Anomaly black cells appear because of duplicate characters. As they do not belong to any valid alignment, these cells degrade the prediction quality. We thus introduce two filtering methods to remediate this.

Mathematical Morphology Originally proposed in [14], mathematical morphology enhances an input image by specified filters and operations. Alignment noise usually appears in positions out of the main diagonal. An example is the cell on the right of Fig. 1a which matches the character 'r' in 'hard' with the last character of 'harder'. Two 3×3 filters are shown in Fig. 1b (2,top). The original matrix is filtered by both filters for grey scale erosion. The two filtered matrices are combined by their maximum values. Using this procedure, black cells appearing out of a diagonal line are filtered out, while diagonal lines keep sharp ending. Figure 1b (2,bottom) shows this.

Diagonal Weight Usually, the valid alignments are located on the main primary diagonal line. So, we apply a linear weighting scheme along this diagonal line. Cells which are further away from the line are gradually less weighted. Figure 1b (4,top) shows the weighting filter. Equation (3) denotes the value at (x, y) of the weight-filtered matrix AB. z is the size of the matrix. Figure 1b (4,bottom) shows the result of this filtering method.

$$AB_{xy} = a_x b_y \times \left(1 - \frac{|x - y|}{z}\right) \tag{3}$$

3.4 Neural Network Model

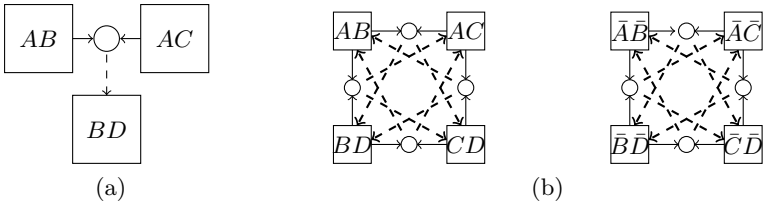

Fig. 2. Equivalent prediction flows

To design an appropriate neural network structure, we rely on the characteristics of analogies. An important characteristic is the equivalent forms of analogy.

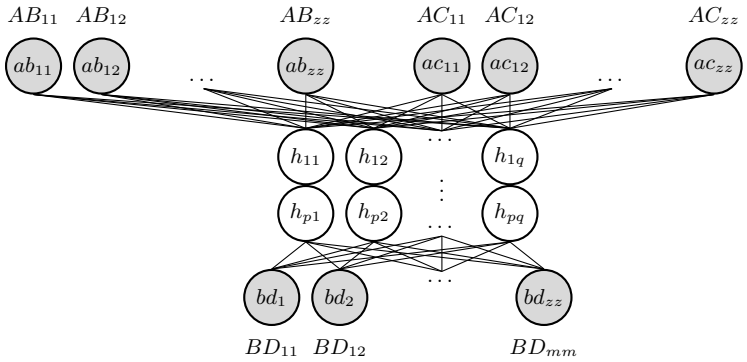

Fig. 3. Neural network structure

As described in [8], a single form $A : B :: C : D$ has 7 other equivalent forms: $A : C :: B : D$, $B : A :: D : C$, $B : D :: A : C$, $C : A :: D : B$, $C : D :: A : B$, $D : B :: C : A$, $D : C :: B : A$.

Another characteristic is the mirroring of strings. If $A : B :: C : D$ is an analogy, then $\bar{A} : \bar{B} :: \bar{C} : \bar{D}$ holds too, where $\bar{A}$ represents the mirror of A. The mirror of string $a_1 a_2 \dots a_m$ is $a_m a_{m-1} \dots a_1$. As a result, we get eight additional equivalent equations. Equivalent prediction flows are shown in Fig. 2. Four boxes on corners represent the alignment matrix generated from the input strings (mirror versions on the right). Matrix AB is the fixed-dimension alignment matrix build against string A and B using the method explained in Sect. 3.1 to 3.3. The two input matrices are flattened and concatenated to form the input vector. The concatenations are represented as a circle connection. The merged data are fed into the neural network. The dashed line is representing the neural network structure with shared parameters for all equivalent prediction flows. The network is trained on all equivalent data flows. Note that the direction of the input alignment matrices needs to be in the correct orientation.

The neural network $AB \circ AC \dashrightarrow BD$ is detailed in Fig. 3. The network predicts the matrix BD. The input data is the flattened and concatenated representation of matrices AB and AC. The total number of input nodes is thus $2 \times z^2$. The flow goes into p fully connected hidden layers, where each layer has q nodes. The output layer has z^2 nodes. Pairs of predicted output matrices from the network are decoded into a final string by a decoder algorithm explained below.

3.5 Decoder

$$|D| = |B| + |C| - |A| \tag{4}$$

$$|D|_a = |B|_a + |C|_a - |A|_a \,, \ \forall a \tag{5}$$

From the alignment matrices AB and AC, the neural network produces a pair of matrices that stand for the alignment BD and CD. The decoder decodes the pair of matrices into a final string which is a hypothesis for the solution D of

Algorithm 1: Decoder

Input: BD, CD: two re-sampled matrices
Input: A, B, C: input strings
Input: z: length of output string
Data: N: set of number of occurrences for each character
Data: $V[c, i]$: set of likelihood values of character c at each i position
Output: D: output string

$N, V, D \leftarrow []$ `// Initialize.`
foreach $c \in \{C \cup B \cup A\}$ **do**
  $N[c] \leftarrow |B|_c + |C|_c - |A|_c$ `// Enforce property (5).`
foreach $c \in \{A \cup B \cup C\}$ **do**
  for $j \leftarrow 1$ **to** z **do**
    $V[c, j] \leftarrow \sum(BD[i, j], \forall B[i] = c) +$ `// Compute likelihoods.`
      $\sum(CD[i, j], \forall C[i] = c)$
foreach $c \in \{A \cup B \cup C\}$ **do**
  for $i \leftarrow 1$ **to** $N[c]$ **do**
    $D[i] \leftarrow \arg_c \max(V[c, i])$
return D

the analogical equation between strings $A : B :: C : D$, In the decoder, we rely on a set of properties of analogies. The first property is the relationship between the lengths of the output string and the input strings. Equation (4) shows this relationship. The length of D is entirely determined by the lengths of A, B and C. Another piece of information is the number of occurrences of symbols in the output string. In (5), $|A|_a$ stands for the number of occurrences of symbol a in string A. (5) applied to all symbols implies (4). Our decoder is based on the following three pieces of information: the two predicted alignment matrices, the length of the output string, and the number of occurrences of each symbol.

4 Experiments

4.1 Dataset

We construct a data set of analogical equations on strings. The data set is a combination of the subset of the test given in [11] that contains only formal analogies (3370 analogies, e.g., *bright* : *brightest* :: *sweet* : *sweetest*) and formal analogies in multiple languages (some extracted from the appendix of [7]). We randomly selected 10% of the data as our test set, the rest is used for training. The statistics of the data set are: # of training samples = 5214, # of test samples = 579, average edit distance = 1.78, average length of string = 7.04, SD length of string = 2.54. Some examples of analogical equations in the data set are: (Japanese) 自由 : 不自由な :: 用意 : 不用意な, (Chinese) 读 : 读者 :: 学 : 学者.

4.2 Experimental Procedure

We performed a series of experiments to evaluate the performance of each parameter (see subsection below). Another experiment determines the highest accuracy rate. For each experiment, all parameters are constant except those parameters which are tested. The basic parameter settings are: size of alignment matrices = 16×16, re-sampling method = proposed, filtering method = both, number of hidden nodes = 128, number of hidden layers = 1, loss function = MSE, activation = ReLU[12], optimizer = Adam[5], and number of epochs = 200. Any alteration to these basic parameters is clearly stated in the description of each experiment.

		# of hyper parameters	Train time (m:s)	Train loss (MSE)	Test loss (MSE)	Accuracy (%)
Alignment matrices size	2 × 2	1,668	4:07	0.009	0.005	1.73
	4 × 4	6,288	4:53	0.013	0.008	16.75
	8 × 8	24,768	5:34	0.017	0.010	67.18
	16 × 16	98,688	7:12	0.024	0.017	79.10
	32 × 32	394,368	14:03	0.035	0.026	**84.11**
Re-sampling methods	NN	98,688	6:56	0.039	0.031	67.88
	Bilinear	98,688	7:10	0.015	0.010	72.71
	Bicubic	98,688	6:40	0.019	0.012	78.24
	Proposed	98,688	7:12	0.024	0.017	**79.10**
Filtering methods	None	98,688	6:28	0.056	0.044	77.72
	Morph	98,688	6:39	0.040	0.031	76.68
	Weight	98,688	7:32	0.034	0.025	79.45
	Both	98,688	7:12	0.024	0.017	**80.48**
Number of hidden nodes	128	98,688	7:12	0.024	0.017	80.83
	256	197,120	8:21	0.022	0.015	82.38
	512	393,984	9:51	0.021	0.017	83.07
	1024	787,712	13:28	0.019	0.012	**85.84**
Number of hidden layers	1	98,688	7:12	0.024	0.017	80.66
	2	115,200	9:55	0.023	0.015	84.44
	3	131,712	11:22	0.023	0.014	86.36
	4	148,224	11:54	0.023	0.014	**87.56**

Table 1. Experiment results

4.3 Evaluation

1. **Training Time**. A significant advantage of our design is its ability to learn at a high speed. We ran experiments to test the influence of various hyperparameters. We measured the training times after 200 epochs.

2. **Loss**. We measured the values of a loss function (or objective function). These values reflect the ability of the neural network to predict the correct alignment matrices by comparing the output with the reference ground truth. The Mean Square Error (MSE) function is given in (6). P is the predicted matrix from the neural network, and T is the ground truth matrix. Lower values reflect a more precise prediction, hence better model configurations.
3. **Accuracy**. We measured the accuracy of our network by the percentage of correct answers over the total number of test samples (see (7)). In any case if one or more characters in the output string are different from the reference string, the prediction is counted as a failure.

$$\text{MSE} = \frac{1}{z^2} \sum_{i=1}^{z} \sum_{j=1}^{z} (P_{ij} - R_{ij})^2 \tag{6}$$

$$\text{Accuracy} = \frac{\text{\# of correct answers}}{\text{total \# of test samples}} \times 100 \tag{7}$$

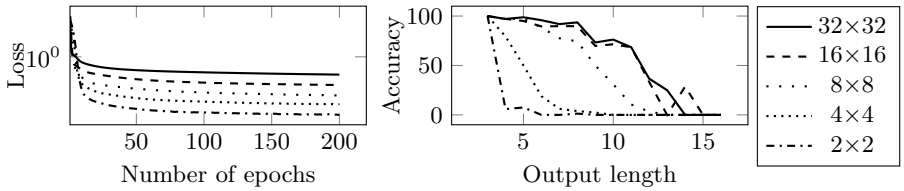

Fig. 4. Loss (left), accuracy against output length (right)

5 Experiment Results

Size of Alignment Matrices In this experiment, the size of alignment matrices are varied from 4 to 32 by subsequent powers of 2. Experiment results in Fig. 4 show expected behaviors. The bigger the alignment matrices, the higher the accuracy. The highest rate of 84.11% is obtained with 32×32 matrices. Figure 4 (right) shows that the size of alignment matrices directly contributes to their ability to output longer solutions. The downside of bigger alignment matrices is the number of network connections which causes an exponential explosion of hyper-parameters. Table 1 shows that training times increase with the number of hyper-parameters.

Matrix Re-sampling Methods We compared our re-sampling method with nearest neighbor, bilinear and bicubic methods. Results show that our re-sampling method achieves the highest accuracy.

Filtering methods As we proposed two filtering methods, we test all combinations: no filtering, only one, or both. This gives four combinations. The results in Table 1 show that the use of both filtering methods yields the highest accuracy. We observe that the use of mathematical morphology yields a lower accuracy rate than without any filtering. This may indicate some limitation of the selected morph filters.

Number of hidden nodes We set the number of hidden layers to one, but the number of nodes varies. A higher number of hidden nodes reflects the ability to recognize more complex patterns. Results show that networks with more hidden nodes can yield higher accuracy rates at the expense of the training time.

Number of hidden layers Usually, deeper network structures are favored for more complicated patterns. But a disadvantage is longer learning times. In this experiment, each network has the same number of hidden nodes (128) per layer, but with a different number of layers. As expected, a deeper structure give a better recognition rate. Interestingly, with four hidden layers, the number of hyper-parameters is a lot lower than one large single layer network as in the last experiment. Yet, a deeper network can achieve a better accuracy rate in less training time.

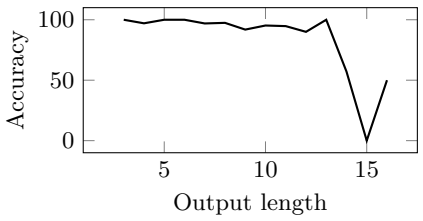

Fig. 5. Accuracy against output length

Table 2. Comparison to baseline

	Training time (m:s)	Accuracy (%)
Baseline	—	94.47
Proposed	35:54	**95.68**

Benchmark We select the most extreme configuration (alignment matrices = 32×32, number of hidden nodes = 1024, number of hidden layers = 2) to compare the results with a baseline algorithm given in [7]. Table 2 shows that our proposed neural network achieves a higher accuracy rate. This may come from the fact that our dataset contains some samples which do not comply the formalization of the baseline algorithm. Figure 5 shows some limitation in our network to solve equations with longer strings. Nevertheless, this result proves the effectiveness of our neural network model for the task.

6 Conclusion

In this work, a neural network design to solve analogical equations of commutation on strings of symbols has been proposed. We presented several methods to

transform the input strings to matrix representation and back to output string. Two filtering methods to reduce the alignment noise were introduced. The model parameters were tested on a number of experiments. They show promising results as an accuracy of more than 95% was achieved. The comparison to a baseline system showed higher accuracy rate on the test set.

We intend to further improve our neural network in the future. From the reported experiments, the accuracy degrades with the length of strings. Improvements in the network structure may help to improve the prediction accuracy. Also, the decoding algorithm impacts the accuracy of the final string. The decoding scheme can be further improved.

References

1. Dosovitskiy, A., Tobias Springenberg, J., Brox, T.: Learning to generate chairs with convolutional neural networks. IEEE Conference on Computer Vision and Pattern Recognition pp. 1538–1546 (2015)
2. Gibbs, A.J., McIntyre, G.A.: The diagram, a method for comparing sequences. European Journal of Biochemistry 16(1), 1–11 (1970)
3. Gregor, K., Danihelka, I., Graves, A., Rezende, D.J., Wierstra, D.: Draw: A recurrent neural network for image generation. In: Proceedings of the 32nd International Conference on Machine Learning (ICML-15) (2015)
4. Johnson, R., Zhang, T.: Semi-supervised convolutional neural networks for text categorization via region embedding. In: Proceedings of the Advances in neural information processing systems. pp. 919–927 (2015)
5. Kingma, D., Ba, J.: Adam: A method for stochastic optimization. In: Proceedings of the 3rd International Conference for Learning Representations (2014)
6. Lai, S., Xu, L., Liu, K., Zhao, J.: Recurrent convolutional neural networks for text classification. In: Proceedings of the 29th AAAI Conference on Artificial Intelligence. pp. 2267–2273 (2015)
7. Lepage, Y.: Solving analogies on words: an algorithm. In: Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics. Association for Computational Linguistics (1998)
8. Lepage, Y.: Languages of analogical strings. In: Proceedings of the 18th conference on Computational linguistics-Volume 1. pp. 488–494. Association for Computational Linguistics (2000)
9. Lepage, Y., Denoual, E.: Purest ever example-based machine translation: Detailed presentation and assessment. Machine Translation 19(3-4), 251–282 (December 2005)
10. Miclet, L., Delhay, A.: Analogy on sequences: a definition and an algorithm. Ph.D. thesis, INRIA (2003)
11. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. International Conference on Learning Representations (2013)
12. Nair, V., Hinton, G.E.: Rectified linear units improve restricted boltzmann machines. In: Proceedings of the 27th International Conference on Machine Learning (ICML-10). pp. 807–814 (2010)
13. Reed, S.E., Zhang, Y., Zhang, Y., Lee, H.: Deep visual analogy-making. In: Proceedings of the Advances in Neural Information Processing Systems. pp. 1252–1260 (2015)

14. Serra, J.: Image analysis and mathematical morphology. Academic press (1982)
15. Zhang, X., Zhao, J., LeCun, Y.: Character-level convolutional networks for text classification. In: Proceedings of the Advances in Neural Information Processing Systems. pp. 649–657 (2015)