Context Encoder for Analogies on Strings

Tianjing Zhao
Graduate School of Information,
Production and Systems,
Waseda University
bilibilimisaka@akane.waseda.jp

Yves Lepage
Graduate School of Information,
Production and Systems,
Waseda University
yves.lepage@waseda.jp

Abstract

We propose a model based on context encoder to solve formal analogies on strings like *aaabbbccc* : *aaaabbbbcccc* :: *abc* : x $\Rightarrow$ x = *aabbcc* or *ubid* : *tubid* :: *ofjid* : x $\Rightarrow$ x = *tofjid*. As a context encoder model, it consists of a generator and a discriminator. The generator attempts at generating the result of an analogical equation, while the discriminator attempts at discriminating solutions coming out of the generator against the real solution of the analogical equation. We conduct experiments on publicly available data sets to compare the performance of our model with a previously published method designed for the same task. Our results show slight increases in accuracy, in comparison to a fully connected neural network architecture.

1 Introduction

To design machines or programs capable of solving analogical equations is still a challenge for artificial intelligence. There are several levels on which analogies can be solved:

- world knowledge level:
  England : *English* :: *Lichtenstein* : x $\Rightarrow$ x = *German;*
- semantic level:
  man : *woman* :: *king* : x $\Rightarrow$ x = *queen;*
- grammatical level:
  I play : *she plays* :: *I am* : x $\Rightarrow$ x = *she is;*
- morphological level:
  cat : *cats* :: *child* : x $\Rightarrow$ x = *children.*

As for *semantic* analogies, the work in (Turney and Littman, 2005; Turney and Pantel, 2010) has shown that a computer program can reach average human performance in the task of solving SAT questions. Nowadays, word embeddings are used to address this problem (e.g. (Drozd et al., 2016)).

The example analogies given above are all irregular in form. However, a number of analogies which hold at these same levels may exhibit regularities in the word forms themselves.

- *England* : *English* :: *Poland* : x $\Rightarrow$ x = *Polish;*
- *man* : *woman* :: *policeman* : x $\Rightarrow$ x = *policewoman;*
- *I play* : *she plays* :: *I accept* : x $\Rightarrow$ x = *she accepts;*
- *cat* : *cats* :: *dog* : x $\Rightarrow$ x = *dogs.*

Such regularities account for a good part of language productivity, i.e., the creation of new word forms from old ones. The regularities encountered in such new word forms can be described in terms of prefixing, suffixing, parallel infixing and circumfixing. This can be reduced to fundamental edit operations found in defining edit distances (Levenshtein, 1966): insertion, deletions and substitutions. With such a formalisation, the problem becomes purely formal, and can be illustrated with examples not linked to any language, like: *aaabbbccc* : *aaaabbbbcccc* :: *abc* : x $\Rightarrow$ x = *aabbcc* or *lbmc* : *lvmbjc* :: *nvtkje* : x $\Rightarrow$ x = *nvtbjkje.*

The previous English examples (and those in the figures in the rest of this paper) may leave the impression that the problem is in the reach of regular expression techniques. However, in all generality, the problem is more complex as already illustrated with the last two formal examples of the previous paragraph. It involves parallel infixing, a phenomenon necessary for the description of Semitic languages (e.g., Hebrew: *mélex* : *mlaxím* :: *dérex* : *draxím*; see more examples in Table 3). Parallel infixing is also ubiquitous when the strings are sentences. This actually places the general problem in the world of context-sensitive languages.

The purpose of this paper is to propose a method to solve analogical equations between strings on the level of form only but in all generality. We first introduce the representation adopted in our method. We then give the necessary background knowledge concerning generative adversarial nets and context encoder networks. Based on this, we propose and design a context encoder model to solve analogical equations. The last part describes experiments on well-established data and how the proposed model behaves when varying some of its hyper-parameters. We also compare the results of our model with previous models.

2 Approach and Processing Steps

We design a context encoder model to solve analogical equations between strings, $A : B :: C : D$, that we solve for D. As for the general flow of data, we follow (Kaveeta and Lepage, 2016). The main difference of our work with this previous work is described in Sect. 2.5. Our contribution lies in the neural network architecture adopted. There also exists a second difference in the post-processing step; it will be mentioned in Sect. 2.4.

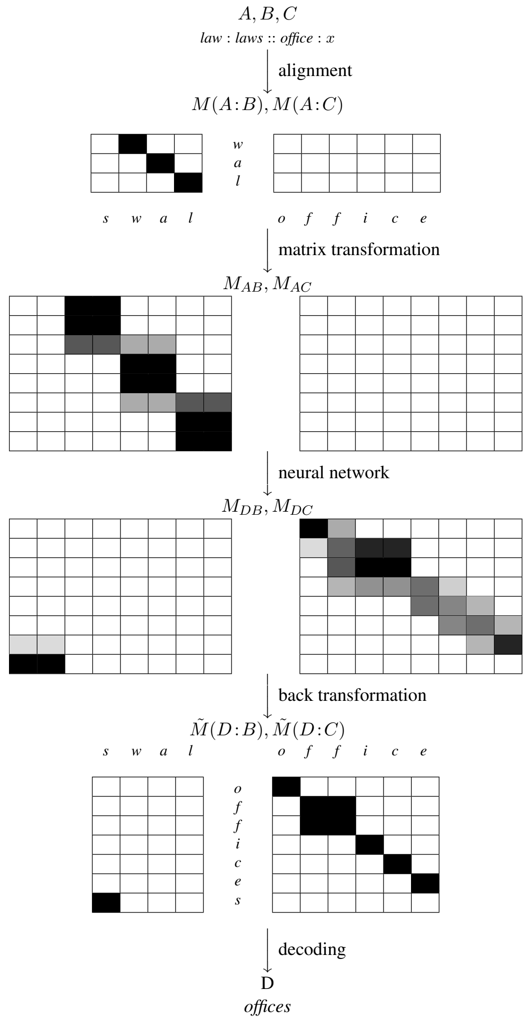


Figure 1: Sketch of the data flow adopted to solve $A : B :: C : D$ for D in the previous approach reported in (Kaveeta and Lepage, 2016) and in our approach.

Our proposed network does not directly solve the equation. The input to the network is not the string triple (A, B, C) itself, but two alignment matrices $M(A\!:\!B)$ and $M(A\!:\!C)$. Such alignment matrices are easy to build. It is of course possible to suggest a recurrent neural network model that would take the concatenation of the three strings as input and would output the fourth string directly. However, we are concerned with future extensions and want to leave the door open to the direct use of alignment matrices as found, for example in machine translation with sub-sentential alignments.

In order to meet the constraints of neural networks, especially fully connected networks, transformations are needed in a pre-processing step to obtain fixed-size square matrices, M_{AB} and M_{AC}.

The task of the network is to output two matrices M_{DB} and M_{DC} from the two input transformed alignment matrices M_{AB} and M_{AC}. These output matrices are of the same type as transformed matrices.

They are transformed back into two matrices $\tilde{M}(D : B)$ and $\tilde{M}(D : C)$ in a first post-processing step. We interpret these two matrices as alignment

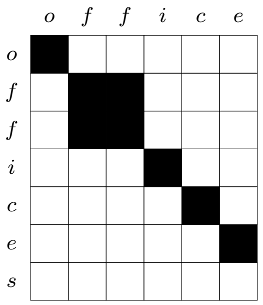


Figure 2: Alignment matrix $M(\textit{office}:\textit{offices})$ for the two strings *office* and *offices*. The black cells are the match points, i.e., the places where the same character is to be found in the two strings.

matrices between D and B, and D and C respectively, but they are not perfect in terms of alignment: the values are not discrete (0 or 1) as they range from 0 to 1, and some values may point simultaneously to different characters for the same position. This explains the different notation used: $\tilde{M}(D:B)$ instead of $M(D:B)$. We call such matrices quasi-alignment matrices.

These two quasi-alignment matrices are finally decoded into a string candidate D in a second post-processing step. Note however, that, because the two quasi-alignment matrices $\tilde{M}(D:B)$ and $\tilde{M}(D:C)$ are not perfect alignment matrices, there is a need for a dedicated algorithm.

Fig. 1 pictures the overall flow of data. The following sections explain each step in the process.

2.1 Alignment Matrices

An alignment matrix between two strings of characters shows the match points, i.e., the positions at which equal characters are to be found in the two strings. Figure 2 shows the alignment matrix between the strings *office* and *offices* (example in inflectional morphology, noun - regular plurals, from BATS 3.0 data set (Gladkova et al., 2016)).

2.2 Pre-Processing Step: Interpolation

Alignment matrices between strings of different lengths obviously exhibit different lengths. Fully connected layers of neural networks require fixed-dimension input. For our problem, the use of fully connected layers requires that alignment matrices of different lengths be cast into fixed-size matrices before being fed to such network layers.

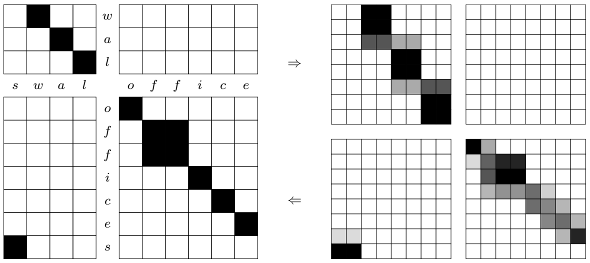


Figure 3: Linear matrix interpolation for the analogy *law* : *laws* :: *office* : *offices*. Original alignment matrices on the *left*; interpolated square matrices of size 5×5 on the *right*. On the *left*, the cells take only two discrete values, 0 or 1, represented by the white or black colours. The cells on the *right* take real values in the range $[0, 1]$, hence the various shades of gray

To allow for comparison, we use the same four interpolation methods as (Kaveeta and Lepage, 2016) to transform matrices of different lengths into fixed-size square matrices. Figure 3 illustrates the result of applying one of these methods, linear interpolation, on the four alignment matrices of the example analogy *law* : *laws* :: *office* : *offices*. On the *right* of the figure, the two matrices at the *top* are the two fixed-size matrices input to a fully connected layer of a neural network; the two matrices at the *bottom* are the expected output of the neural network.

With the adopted approach, the flow of the data during the resolution of the analogical equation *law* : *laws* :: *office* : $x \quad \Rightarrow \quad x =$ *offices*. is as follows. The *top left* matrices are transformed into the *top right* matrices. The neural network outputs the two *bottom right* matrices which are transformed back into the two *left bottom* matrices, from which the (vertical) string *offices* is decoded. Figure 3 is an ideal picture, because, e.g., the *bottom left* matrices obtained from the *bottom right* matrices may contain misplaced or inconsistent match points.

2.3 First Post-Processing Step: Back-Transformation

The two matrices M_{DB} and M_{DC} output by the network are back transformed into quasi alignment matrices so as to obtain matrices of the sizes fitted to the lengths of strings B, C and D. To do so, we just apply the same interpolation technique to each of the matrices M_{DB} and M_{DC} to obtain the matrices $\tilde{M}(D:B)$ and $\tilde{M}(D:C)$ with the desired lengths as constraints.

2.4 Second Post-Processing Step: String Decoder

For an analogical equation $A : B :: C : D$ where D is the unknown, (Lepage, 1998; Lepage, 2017) show that the properties of analogies of commutation between strings of symbols imply that the following features of the solution D can be computed in advance.

- The length of the solution D:

  $$\mid D \mid = \mid B \mid + \mid C \mid - \mid A \mid \qquad (1)$$

  Here, $\mid S \mid$ stands for the length of a string S.

- The number of occurrences of each of the characters in D:

  $$\mid D \mid_c = \mid B \mid_c + \mid C \mid_c - \mid A \mid_c, \ \forall c \qquad (2)$$

  Here $\mid S \mid_c$ denotes the number of occurrences of character c in string S. Equation (2) is also trivially true for characters which do not appear in any of the strings as $0 = 0 + 0 - 0$.

- The set of characters in D, deduced from the previous feature, as it is the set of characters c for which $\mid D \mid_c \neq 0$.

We make the choice of this interpretation of analogy. With this choice, the task of the string decoder[1] reduces to choose the right position of each character in D based on the knowledge of the quasi alignment matrices $\tilde{M}(D : B)$ and $\tilde{M}(D : C)$ plus, of course, the knowledge of the position of each character in strings B and C. An algorithm for this task has been proposed in (Kaveeta and Lepage, 2016, Algorithm 1). To select a character for a given position i in D, the algorithm relies on a specific criterion.

We use the same algorithm, but with a different criterion. This constitutes the second difference with this previous work. In the previous work, the character at position i in D is selected on the basis of the sum of the contributions of all possible positions where the candidate character is found in B and C. This is defined by Eq. (3).

$$\begin{aligned} V[c,i] = & \sum_{j/B[j]=c} \tilde{M}(D\!:\!B)[i,j] \\ & + \sum_{j/C[j]=c} \tilde{M}(D\!:\!C)[i,j] \end{aligned} \qquad (3)$$

We think that the selection should preferably identify only one point, either from B or from C, the one which has the largest contribution to character c for this given position in D. For this reason, our formula picks up the maximal value. This is defined in Eq. (4).

$$\begin{aligned} V[c,i] = \max(& \max_{j/B[j]=c} \tilde{M}(D\!:\!B)[i,j], \\ & \max_{j/C[j]=c} \tilde{M}(D\!:\!C)[i,j] \ ) \end{aligned} \qquad (4)$$

We just replace the criterion given in Eq. (3) by the one in Eq. (4) in the string decoder of (Kaveeta and Lepage, 2016). The algorithm iteratively scans each position i in D from the beginning to the end. For a position i, it selects the character c with the highest score $V[c, i]$. The number of occurrences of the chosen character c is decreased by 1 to prevent any further use as soon as it reaches 0. The next position $i + 1$ is considered in turn.

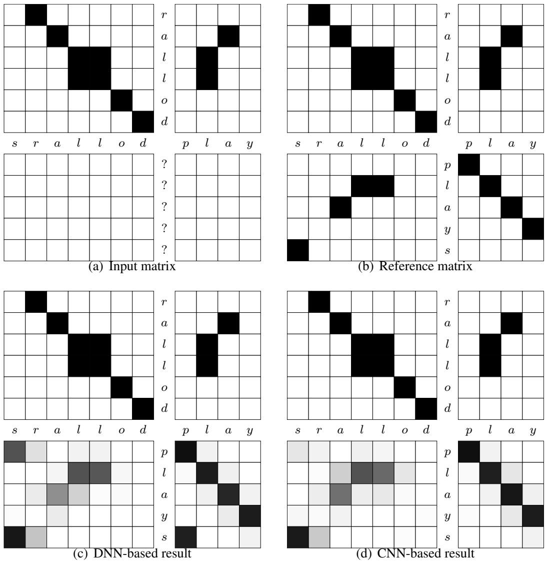
2.5 Related Work: Context Encoder

In (Pathak et al., 2016) a generative adversarial network (GAN) is customised into a network dedicated to feature learning for inpainting. This neural network, called a *context encoder,* is used to reconstruct the missing part of an image by learning the features which match a corresponding image with the removed part of the image. The model used consists of an encoder-decoder pipeline and several loss functions. The encoder-decoder pipeline is divided into three components.

The first component is the encoder, which encodes the input image into a more status space to obtain a latent feature representation. For the encoder model, they use a layer derived from AlexNet that contains only the convolutional and pooling parts. This is because AlexNet is suitable for classification tasks, but their task is not a classification task.

The second component connects the encoder and the decoder. They call it the channel-wise fully connected layer. Its main purpose is to propagate the

[1] Not to be confused with the decoder in the encoder-decoder architecture.



(a) Input matrix (b) Reference matrix

(c) DNN-based result (d) CNN-based result

Figure 4: Given the four matrices of an analogical equation where the two bottom ones are missing as in (a), a human can fill out the two bottom missing matrices correctly as in (b). The result of automatically filling the two bottom matrices using our DNN-based context encoder and using our CNN-based context-encoder are shown in (c) and (d).

feature map information, so its role is to connect the parameters of the encoder and the decoder. Originally their idea was to use a fully connected layer, but if a fully connected layer was used, this would lead to an explosion in the number of parameters leading to intractable training time. Instead, they use a convolutional layer. In our experiments, we also tried using a fully connected layer, but we abandoned for the same reason.

The third component is the decoder. It performs the reconstruction by upsampling the feature representation in the low dimensional space, so as to restore to the image size. This is done by deconvolution and upsampling.

Figure 4 shows that our task may be considered similar to the task in (Pathak et al., 2016). The four alignment matrices between the four terms of an analogy can be considered an image of black and white pixels. For an analogical equation where the last term D is unknown, the two bottom matrices are unknown. This is similar to the missing part in an image.

The loss function in (Pathak et al., 2016) is a joint loss, made up of two parts, an adversarial loss (*adv*) and a reconstruction loss (*rec*), each weighted by its own weight (λ_{adv} and λ_{rec}, both between 0 and 1), as defined by Eq. (5).

$$\ell = \lambda_{adv} \times \ell_{adv} + \lambda_{rec} \times \ell_{rec} \tag{5}$$

We adopt this joint loss scheme for our problem.

2.5.1 Adversarial Loss

For the problem of learning features for image inpainting, in (Pathak et al., 2016), a simplification of the loss function found in standard GANs (Goodfellow et al., 2014) is used, because GANs easily make the difference between generated and ground truth images based on boundary discontinuities. In our setting, this problem does not exist because the boundary extends over the entire horizontal size of the matrices and because the boundary does not exhibit specific discontinuities very different from the ones found elsewhere in the alignment matrices. For this reason, on the contrary to (Pathak et al., 2016), we use the standard loss function found in GANs[2]. It is defined by Eq. (6) as a logistic likelihood where a ground truth sample x is compared to a generated sample z created from a noise distribution by the generator $\mathbf{G}$. The two-player game between the discriminator $\mathbf{D}$ and the generator $\mathbf{G}$ is expressed in the formula by the $\min \max$ dilemma.

$$\begin{aligned} \ell_{adv} = \min_{\mathbf{G}} \max_{\mathbf{D}} (\ & \mathbb{E}_{x \in X} [\log \mathbf{D}(x)] \\ & + \mathbb{E}_{z \in Z} [\log(1 - \mathbf{D}(\mathbf{G}(z)))] \,) \end{aligned} \tag{6}$$

In standard GANs, Z represents a continuous noise distribution. In our setting, Z is not a noise distribution but a random sampling over the possible solution alignment matrices found in the training set. Rather than discriminating over a continuous distribution space as in standard GANs, in our setting, the discriminator performs a selection over a discrete space.

2.5.2 Reconstruction Loss

The role of the reconstruction loss function is to compare the reconstructed part of the image with the ground truth. In (Pathak et al., 2016), they tried the L^1 and the L^2 norms of the differences between the

[2] We also experimented with the adversarial loss function in (Pathak et al., 2016), for no observed improvement.

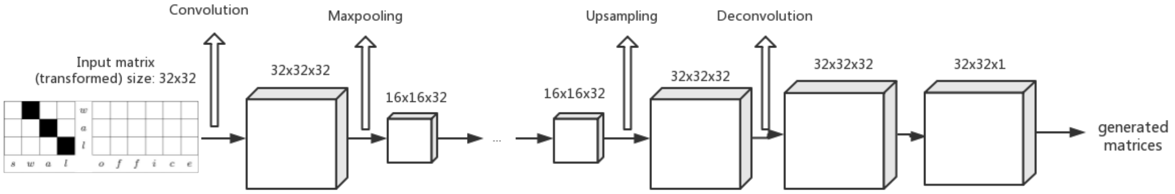


Figure 5: Architecture of our CNN-based generator

reconstructed part and the ground truth part of the original image. The L^2 norm delivered better results for their problem. We experimented with the L^2 norm and also with a classical mean squared error (MSE) scheme. We found that the MSE loss function leads to slightly better results and adopted it for this reason. In Equations (7) and (8), $\mathbf{F}$ stands for the function realised by the generator-discriminator pipeline, M is the ground truth matrix and N is its size.

$$\text{MSE}(\mathbf{F}(M), M) = \frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N}(\mathbf{F}(M)[i,j] - M[i,j])^2 \tag{7}$$

$$\ell_{rec} = \frac{1}{2}(\text{MSE}(\tilde{M}(D\!:\!B), M(D\!:\!B)) + \text{MSE}(\tilde{M}(D\!:\!C), M(D\!:\!C))\ ) \tag{8}$$

3 Proposed Method

Any kind of networks can be used for the generator and the discriminator, but not any combination of two networks is suitable for any task. Hereafter, we consider which network for the generator and the discriminator is suited to our task.

3.1 Generator of Quasi-Alignment Matrices

We propose two different generators to generate two output alignment matrices from the input alignment matrices.

The first one is a deep neural network (DNN) auto-encoder, with three fully connected layers, that directly tries to generate the output alignment matrices. It is illustrated in Fig. 6.

The second one is a deep convolution neural network (CNN) auto-encoder. It is illustrated in Fig. 5. We would conjecture that the analytic features of a CNN should deliver better performance thanks to the extraction of local features, while the softmax layers should be able to capture global alignment correspondences. This conjecture will be inspected when analyzing the experiment results in Sect. 4.3.

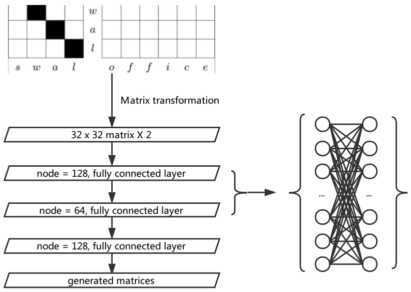


Figure 6: Architecture of our DNN-based generator

3.2 Adversarial Discriminator

As input, the discriminator accepts output alignment matrices, i.e., images of gray pixels generated by the generator, and real matrices of white and black pixels (match points) obtained from real data. Our discriminator is a traditional CNN, trying to make true-false judgments on the input matrix, i.e., it yields a probability between 0 and 1. Fig.7 shows the architecture of our discriminator. Thanks to backpropagation, the difference in nature of the generated output alignment matrices and ground truth alignment matrices will drive the generator to generate sharper and sharper matrices.

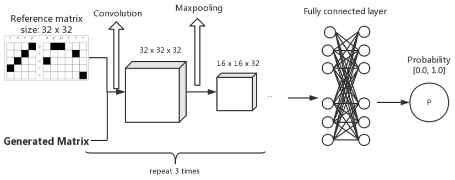


Figure 7: Architecture of our discriminator

4 Experiments

4.1 Data Set

For full comparison with the previous work reported in (Kaveeta and Lepage, 2016), we use the same data and the same measurement protocol. The data set was selected from different sources. Formal analogies were extracted automatically by checking the constraint on character occurrence number combined with a criterion on edit distance equality.

In total, there are 5,793 analogies. A majority are in English and German, but a certain number of them come from 11 other languages. They include examples of parallel infixing (Arabic, Hebrew, German) and circumfixing (Malay). Example analogies from the data set are shown in Appendix in Table 3. The average length of a string in the data set is 7.0 ± 2.6 characters.

In all experiments, to compare with previous work, we use 10-fold cross validation, i.e., the data set is randomly divided into ten slices of $10\,\%$ of the total size each, and one slice is used to test a system trained on the remaining nine slices. The average over these ten experiments is reported.

4.2 Evaluation Metrics

4.2.1 Training Time

As training times are a concern in training neural networks, we report training times for all the variants of our proposed model. They were obtained on a 4-core i7-3770 processor at 3.4 GHz and 4 Gb memory.

4.2.2 Loss

We report the values taken by the loss function to ensure that our results meet our objectives. These values reflect the ability of the neural network to predict the correct alignment matrices by comparing the output with the reference ground truth.

As mentioned in Sect. 2.5, the loss function for the generator is a Mean Squared Error loss function. For the adversarial discriminator, we use a Binary Cross-Entropy loss function.

4.2.3 Accuracy

The accuracy in solving analogical equations is computed as the ratio of the number of correct answers produced by the model to the total number of analogical equations. A correct answer is an exact match against the ground truth answer.

$$\text{Accuracy} = \frac{\text{\# of correct answers}}{\text{total \# of test samples}} \times 100\,\% \quad (9)$$

4.3 Experimental Results

Our baseline for comparison is the model reported in (Kaveeta and Lepage, 2016). It shares the same general flow of data but it uses a simpler model for the core network: a fully connected neural network. The second difference is a different selection scheme for characters in the string decoder (see Sect. 2.4).

4.3.1 Influence of Matrix Size on Accuracy

Table 1 gives the result of experiments in varying the size of the fixed-size matrices, input to the network.

The results show that our proposed model surpasses the results of the baseline system in accuracy. As for training times, they are comparable with the baseline, except for the largest matrix size, for which our model is twice as fast as the baseline. The loss values for our model seem slightly higher but it is not sure whether the values are directly comparable.

The results do not clearly confirm our conjecture mentioned in Sect. 3.1: the CNN-based context encoder used in the generator part does not always deliver better accuracy than the DNN-based one.

Because the average string length in the data is 7.0 ± 2.6 characters, it is natural that the larger the matrix size used, the higher the accuracy. It is also natural that a matrix size of only 4×4 delivers low accuracy (around or less than $20\,\%$). A matrix size of 8×8 allows the models to cross $50\,\%$ accuracy.

4.3.2 Influence of Interpolation Method on Accuracy

The results in Table 1 were obtained using the bicubic interpolation method (see Sect. 2.2). The

Table 1: Comparison of our method for training times and accuracy with a baseline method when the size of the matrices varies. All reported results use 10-fold cross-validation

	matrix size	Training time (m:s)	Loss	Accuracy (%)
CNN-based Context encoder (our method)	4 x 4	5:05	0.097	21.99
	8 x 8	5:14	0.088	69.72
	16 x 16	5:47	0.093	84.91
	32 x 32	8:13	0.080	**89.62**
DNN-based Context encoder (our method)	4 x 4	5:11	0.108	22.07
	8 x 8	5:23	0.092	66.18
	16 x 16	6:03	0.113	81.49
	32 x 32	8:17	0.094	**86.17**
Baseline: Fully connected NN (copied from (Kaveeta and Lepage, 2016))	4 x 4	4:53	0.013	16.75
	8 x 8	5:34	0.017	67.18
	16 x 16	7:12	0.024	79.10
	32 x 32	14:03	0.035	**84.11**

Table 2: Same as Table 1, but when using various re-sampling methods. The matrix size is 16×16. Again, all reported results use 10-fold cross-validation

	Interpolation method	Training time (m:s)	Loss	Accuracy (%)
CNN-based Context encoder (our method)	bicubic	5:25	0.060	**88.58**
	linear	5:47	0.093	84.91
	bilinear	5:18	0.050	84.11
	nearest	5:11	0.120	76.17
DNN-based Context encoder (our method)	bicubic	5:18	0.067	**85.87**
	linear	6:03	0.113	81.49
	bilinear	5:14	0.078	81.11
	nearest	5:17	0.144	69.66
Baseline: Fully connected NN (copied from (Kaveeta and Lepage, 2016))	bicubic	6:40	0.019	78.24
	linear	7:12	0.024	**79.10**
	bilinear	7:10	0.015	72.71
	nearest	6:56	0.039	67.88

purpose of Table 2 is to measure the influence of the interpolation method on accuracy. The results in this table were obtained using a matrix size of 16×16.

The results confirm that the bicubic method performs the best among the four interpolation methods we tried, as already reported in (Kaveeta and Lepage, 2016). Bicubic interpolation is often chosen for interpolation in image re-sampling, for the reason that compared with bilinear interpolation, it considers more pixels. For our problem of solving analogical equations, this shows that our model prefers configurations where more pixels are taken into account around a given pixel.

It may be observed that the change in interpolation method does not lead to dramatic changes in training times. By contrast with Table 1, the loss values in this experiment are more affected by the variation in re-sampling methods. They seem better correlated with accuracy.

5 Conclusion

We proposed a context encoder model to solve formal analogical equations between strings of symbols. We chose to use alignment matrices to represent the input strings instead of feeding the input strings themselves. This leaves the door open to the resolution of more complex problems using, e.g., sub-sentential alignment matrices in machine translation. The output string is obtained through a string decoding post-processing of the two output alignment matrices predicted by our model.

Our model was inspired by work in image inpainting. In such a context encoder model, the loss function is a linear combination of an adversarial loss and a reconstruction loss. Our adversarial loss function is the standard loss function of GANs while the reconstruction loss function is simply a mean squared error scheme.

We tested two possible neural network architectures for the generator part: fully connected and convolutional neural networks. As fully connected neural networks require fixed-size input, we tested different interpolation techniques to transform the alignment matrices into fixed-sized square matrices to be fed into the network.

Experimental results show that our model outperformed a baseline model based on a fully-connected network architecture. We found that the use of a larger interpolated matrix size led to better accuracy. The best accuracy was achieved when using the bicubic interpolation method.

Acknowledgments

This work was supported in part by the Japanese Society for the Promotion of Science (JSPS) under grand-in-aid n^{o} 18K11447 entitled “Self-explainable and fast-to-train example-based machine translation using neural networks”.

A Example Data

Table 3: Example analogies from the data set used in our experiments

Language	Example analogy
Arabic	*kalb* : *kulaib* :: *masjid* : *musaijid*
	kataba : *kātib* :: *sakana* : *sākin*
Chinese	工程 : 工程师 :: 理发 : 理发师
	今年 : 今天 :: 明年 : 明天
English	*law* : *laws* :: *office* : *offices*
	fixed : *fixedness* :: *serious* : *seriousness*
French	*dues* : *indu* :: *nées* : *inné*
	logique : *logiciel* :: *ludique* : *ludiciel*
German	*fliehen* : *er floh* :: *schließen* : *er schloß*
	sprechen : *ihr sprächet* :: *nehmen* : *ihr nähmet*
Hebrew	*iahmod* : *mahmād* :: *ia‘abor* : *ma‘abār*
	mélex : *mlaxím* :: *rések* : *rsakím*
Japanese	痛い : 痛む :: 親しい : 親しむ
	飛びます : 飛ぶ :: 選びます : 選ぶ
Latin	*amo* : *amas* :: *oro* : *oras*
	facio : *conficio* :: *capio* : *concipio*
Malay	*beristeri* : *isteri* :: *berladang* : *ladang*
	keras : *mengeraskan* :: *kena* : *mengenakan*

References

Aleksandr Drozd, Anna Gladkova, and Satoshi Matsuoka. 2016. Word embeddings, analogies, and machine learning: Beyond king - man + woman = queen. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 3519–3530. The COLING 2016 Organizing Committee.

Anna Gladkova, Aleksandr Drozd, and Satoshi Matsuoka. 2016. Analogy-based detection of morphological and semantic relations with word embeddings: what works and what doesn't. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT 2016)*, pages 8–15.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.

Vivatchai Kaveeta and Yves Lepage. 2016. Solving analogical equations between strings of symbols using neural networks. In *ICCBR Workshops*, pages 67–76.

Yves Lepage. 1998. Solving analogies on words: an algorithm. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics (COLING-ACL 1998)*, volume 1, pages 728–734. Association for Computational Linguistics.

Yves Lepage. 2017. Character–position arithmetic for analogy questions between word forms. In *Proceedings of the Computational Analogy Workshop at the 24th International Conference on Case-Based Reasoning (ICCBR-17)*, pages 17–26, Trondheim, Norway, August.

Vladimir Iossifovitch Levenshtein. 1966. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics-doklady*, 10(8):707–710, February.

Deepak Pathak, Philipp Krahenbuhl, Jeff Donahue, Trevor Darrell, and Alexei A Efros. 2016. Context encoders: Feature learning by inpainting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2536–2544.

Peter D. Turney and Michael L. Littman. 2005. Corpus-based learning of analogies and semantic relations. *Machine Learning*, 60(1–3):251–278.

Peter Turney and Patrick Pantel. 2010. From frequency to meaning: Vector space models of semantics. *Journal of Artificial Intelligence Research*, 37:141–188.