English to Persian Transliteration using Attention-based Approach in Deep Learning

Mohammad Mahdi Mahsuli
PhD Candidate
Department of Computer Engineering and IT
Amirkabir University of Technology
Tehran, Iran
Email: mahsuli@aut.ac.ir

Reza Safabakhsh
Professor
Department of Computer Engineering and IT
Amirkabir University of Technology
Tehran, Iran
Email: safa@aut.ac.ir

Abstract— **In this paper, transliteration is carried out by using the attention-based approach in deep learning. Unlike the previous works which randomly initialize the weights in the encoder, word vector representation of the source vocabulary has been used as an initial value for the weights. The representation is computed by counting the co-occurrences between different characters. Experimental results on an English to Persian transliteration corpus with more than 14000 word pairs show the superior performance of the proposed method (up to 4.21 BLEU points improvement) over the basic attention-based approach.**

Keywords- Deep Learning; Encoder-decoder Approach; Attention-based Approach; Recurrent Neural Network; Vector Representation

I. INTRODUCTION

Deep learning is a sub-branch of machine learning which is based on modeling the representation of knowledge and features in multiple layers [1]. In the past decade, with increasing computing power and providing appropriate solutions to solve optimization problems in deep structures, there was a significant consideration towards deep learning.

A recent application of deep learning is neural machine translation (a task in natural language processing), in which a neural model acquires the ability to translate unseen sentences after seeing training samples of source sentences and their translations. Another task in natural language processing (NLP) is transliteration, which is the conversion of a text from one script to another. Transliteration can be applied as a useful sub-module in many NLP tasks. For example, in a machine translation system, one can use transliteration to convert untranslated names in the source sentence.

In this paper, we apply a novel approach of neural machine translation (known as attention-based approach) to the task of transliteration and propose an extension to the system by intelligently initializing the weights which makes the network converge more rapidly. Inputs and outputs of this task are words in Persian and English languages which are splitted into characters.

The following sections are organized as follows: section II reviews the main previous works done for neural machine translation, (i.e. encoder-decorer approach and the recent attention-based approach) and defines their advantages and disadvantages. In section III, we describe the proposed method in detail and show the procedure of applying attention-based approach to transliteration. Section IV contains the experimental setup and an analysis on the results. Finally in section V, we have a brief conclusion and show the outlook for future works.

II. LITERATURE REVIEW

A. Encoder-decoder Approach

One of the most important works in neural machine translation is the encoder-decoder approach proposed by Cho et al. [2]. In this approach, the feature vector corresponding to each word is first fed to the input of encoder. This feature vector has a fixed length for different words.

In Figure 1. , the repeating module of encoder-decoder approach is depicted in an unrolled manner. At each time step, the feature vector corresponding to a source word enters the repeating encoder module. Then, the hidden state in the encoder captures the information relevant to the seen words up to that time step. This vector is computed based on the hidden state of the previous time step, and the input of the current one:

$$h_t = \phi(h_{t-1}, x_t) = f(W^{(hh)}h_{t-1} + W^{(hx)}x_t) \qquad (1)$$

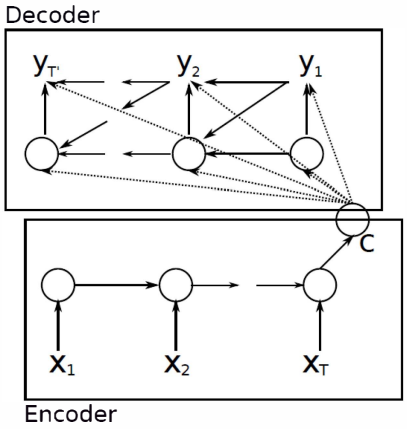

Figure 1. Encoder-decoder approach for machine translation [2]

978-1-5090-5963-8/17/$31.00 ©2017 IEEE

In (1), ϕ is a nonlinear activation function which is usually considered to be sigmoid function. Weight vectors $W^{(hh)}$ and $W^{(hx)}$ are learned during the training process. These vectors define the weights between hidden states at different moments, as well as the weights between hidden states and inputs. At each time step, a word from the source sentence is given to the encoder. Once the last word in the source sentence reaches the encoder, the input state of the encoder at that time step will represent the whole input sentence. This vector is then fed to the decoder in order to emit the first word in the destination sentence. Afterward, the decoder produces the next destination words using the previous hidden state and produced destination words. The equation of computing hidden state h_t and output y_t at time step t is as follows:

$$h_t = \phi(h_{t-1}) = f\left(W^{(hh)} h_{t-1}\right) \tag{2}$$

$$y_t = softmax(W^{(S)} h_t)$$

Through this approach, the destination sentence is produced in a way that the cross entropy error is minimized over all destination words which are conditioned on source words.

B. Attention-based Approach

The solution proposed in [2] is a nice idea. But due to using a fixed-length representation vector in the encoder, the method is unable to model long-range dependencies when sentences become larger. To overcome this issue, Bahdanau et al. [3] have proposed the idea of using a variable-length vector. The proposed approach consists of a bidirectional neural network [4] as an encoder, and a decoder which does the search task in a sentence while decoding a translation.

In an ordinary recurrent neural network, the input sequence **x** is read from its initial symbol x_1 to its final symbol x_{T_x}. But the goal in [3] is that each annotation not only has an abstract of the past words, but it has the abstract of the next one. To this end, a bidirectional recurrent neural network is applied which previously has been used successfully for speech recognition in [5].

A bidirectional recurrent neural network consists of forward and backward recurrent nets. The forward recurrent network $\vec{f}$ reads the data in its normal order (from x_1 to x_{T_x}) and computes the sequence of forward hidden states $(\vec{h}_1, \dots, \vec{h}_{T_x})$. On the other hand, the backward recurrent network $\overleftarrow{f}$ reads the input sequence in the reverse order (from x_{T_x} to x_1) and computes the sequence of backward hidden states $(\overleftarrow{h}_1, \dots, \overleftarrow{h}_{T_x})$. The annotation corresponding to each word x_j is then calculated by concatenating the forward state $\vec{h}_j$ and the backward state $\overleftarrow{h}_j$. In other words, $h_j = \left[\vec{h}_j^T; \overleftarrow{h}_j^T\right]^T$. Then annotation h_j will contain the abstract of previous and next words in the source sequence. Because recurrent networks tend to display recent inputs, annotation h_j is considered in words around x_j. After that, this sequence of annotations is used by the decoder and the alignment model to compute the context vector corresponding to x_j. Figure 2. shows the graphical illustration of the attention-based approach.

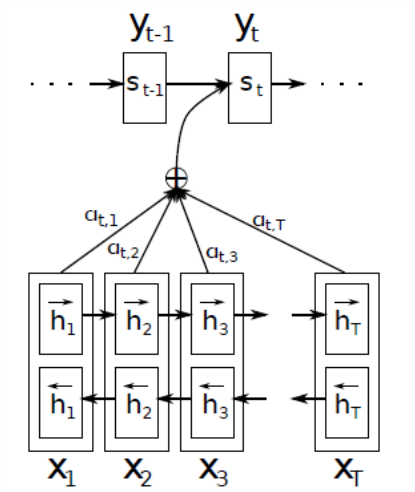

Figure 2. Attention-based model aiming to produce the *t*-th destination word y_t given source sentence $(x_1, x_2, \dots, x_T)$ [3]

In the decoder section, each conditional probability of a destination word given source words and emitting previous destination words is defined as:

$$p(y_i | y_1, \dots, y_{i-1}, \mathbf{x}) = g(y_{i-1}, s_i, c_i) \tag{3}$$

In (3), s_i is the hidden state of the recurrent neural network at time i and is computed as below:

$$s_i = f(s_{i-1}, y_{i-1}, c_i) \tag{4}$$

Function f should be a nonlinear one. Note that in contrast to the usual encoder-decoder approach [2], here the probability of each destination word y_i is conditioned on a specific context vector c_i. This context vector depends on a sequence of annotations $(h_1, \dots, h_{T_x})$ to which the encoder maps the input sentence. Each annotation h_i contains the information relevant to the whole input sequence with a strong focus on the words near the *i*-th word in the input sentence. Then, the context vector c_i is calculated as a weighted sum of these annotations:

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j. \tag{5}$$

The weight α_{ij} is computed by applying the softmax function on the corresponding energy:

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})}, \tag{6}$$

where $e_{ij} = a(s_{i-1}, h_j)$ is an alignment model which scores how much the inputs around the *j*-th word and the outputs around *i*-th position conform. This score is based on hidden state s_{i-1} (right before emitting y_i in (3)) and the *j*-th annotation of the input sentence (h_j).

The attention-based approach is a novel method for handling long sequences and capturing the long-range dependencies in them. But in case of low resources, convergence of the system becomes a challenge.

III. PROPOSED METHOD

In this paper, we aim to solve English to Persian transliteration using deep neural networks. Some related works which use neural networks for transliteration are [6] and [7]. Transliteration is the conversion of a text from one script to another [8]. For instance, "haft" is the English transliteration of "هفت" in Persian. There can be some ambiguities in this task. For example, the word "adam" can be transliterated to "آدم" (/Adam/) or "آدام" (/AdAm/).

To overcome this issue, we consider the transliteration task as a machine translation task, with the difference that in translation, atomic units are words; but here, each character is an atomic unit and consequent characters are separated by a blank space.

In the attention-based approach, each atomic unit (symbol) in the training data is stored as a long vector (same length as the vocabulary size) with all-zero elements except the one which denotes the index of that symbol in the vocabulary. This representation is called 1-hot representation. In contrast, there is another representation called vector representation which maps each symbol into a vector of real numbers. This representation can carry meaningful information about the symbols which 1-hot representation is unable to carry. To this end, we have used word2vec toolkit proposed by the researchers at Google [9]. This tool contains a 2-layer neural network which maps each symbol to a vector using the information of co-occurrences between different symbols. Note that this tool is not a deep neural network, but it maps texts into numerical vectors which can be used by deep neural networks. Since in this method, all the symbols are converted to numbers, the resulting numbers are then language-independent and include hidden concepts of words. This is also true for characters: Since the vector representation of characters is obtained using the co-occurrences between them, this representation can show the tendency of characters to be placed next to each other. Word2vec is similar to an auto-encoder, encoding each word in a vector, but rather than training against the input words through reconstruction, as a restricted Boltzmann machine does, word2vec trains words against other words that neighbor them in the input corpus.

It does so in one of two ways, either using context to predict a target word (a method known as continuous bag of words, or CBOW), or using a word to predict a target context, which is called skip-gram. We use the latter method because it produces more accurate results on large datasets. The proposed architectures of word2vec are shown in Figure 3. .

In CBOW, all words get projected into the same position (their vectors are averaged). We can see that the order of words in the history does not influence the projection. Note that the weight matrix between the input and the projection layer is shared for all word positions. Continuous skip-gram is similar to CBOW, but instead of predicting the current word based on the context, it tries to maximize classification of a word based on another word in the same sentence [9].

In order to speed up the convergence of the attention-based approach and improve the performance, we initialize the input weights of the encoder with vector representation of characters. If E_j be the vector representation of the j-th character, then we can produce the representation matrix of a language (in transliteration task) by concatenating these vectors:

$$E_{v\times n} = \begin{bmatrix} E_1 \\ \vdots \\ E_j \\ \vdots \\ E_v \end{bmatrix} = \begin{bmatrix} E_{11} & \cdots & E_{1n} \\ \vdots & \vdots & \vdots \\ E_{j1} & \cdots & E_{jn} \\ \vdots & \vdots & \vdots \\ E_{v1} & \cdots & E_{vn} \end{bmatrix} \quad (7)$$

In the above equation, v is the vocabulary size (i.e. the number of distinct characters in the desired language), and n is the size of the representation vector for each character. Now if we define ind_j to be the 1-hot representation of each character, and multiply that vector by the representation matrix E, then we can retrieve the representation vector corresponding to that character:

$$E_j = ind_j \times \begin{bmatrix} E_1 \\ \vdots \\ E_j \\ \vdots \\ E_v \end{bmatrix}_{v\times n} \quad , \quad ind_j = \begin{bmatrix} 0_{j-1,1} \\ 1 \\ 0_{v-j,1} \end{bmatrix}_{1\times v} \quad (8)$$

where $0_{j-1,1}$ and $0_{v-j,1}$ are column vectors of sizes $j-1$ and $v-j$ with all-zero elements.

We apply the above process to the source side, but one can also apply the same process to the target side. By applying this technique, we initialize the input weights of the encoder with the vector representation of characters. This can help the attention-based approach to more rapidly converge to a better configuration.

IV. EXPERIMENTS

The statistics of the used corpus is shown in TABLE I. . This corpus is the English-Persian (EnPe) part of the transliteration shared task at Named Entities Workshop (NEWS 2011) [10]. In the corpus, each sample contains of one English word and its Persian transliteration.

TABLE I. STATISTICS OF TRAIN, DEVELOPMENT AND TEST CORPORA

		English	Persian
Train	Words	14,758	
	Characters	91,088	76,039
	Vocabulary	26	28
Dev	Words	1,000	
	Characters	6,180	5,098
	Vocabulary	26	27
Test	Words	1,000	
	Characters	6,190	5,183
	Vocabulary	26	26

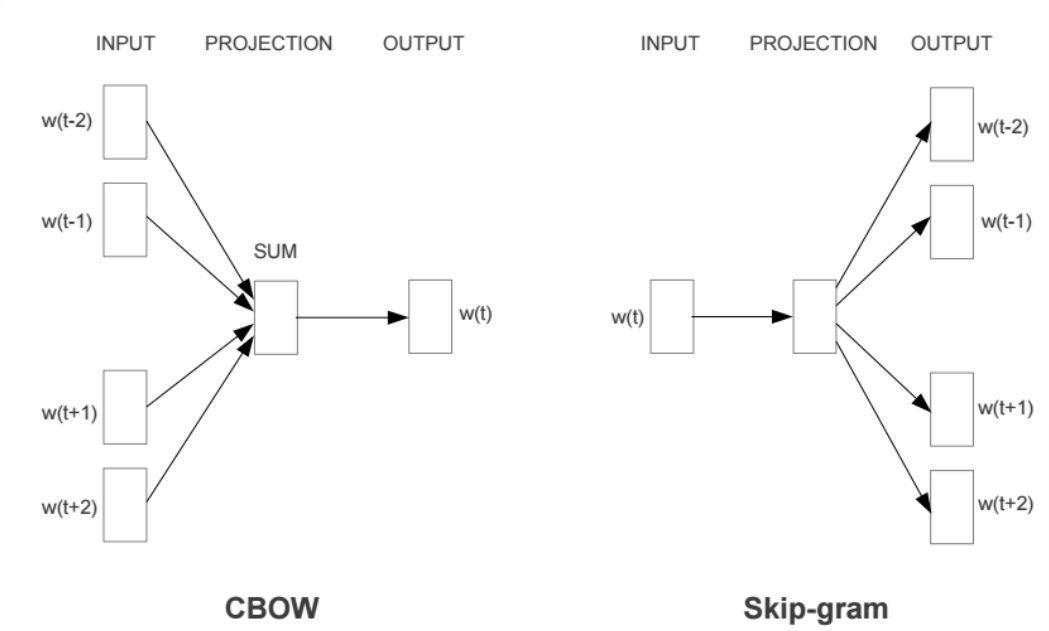

Figure 3. Proposed architectures of word2vec. The CBOW architecture predicts the current word based on the context, and the Skip-gram predicts surrounding words given the current word [9].

As the baseline system, we experimented with the usual attention-based approach [3]. The baseline system and the proposed extension (Word2Vec-Init) are implemented in Python and on Blocks framework [11]. This framework helps programmers build neural network models on top of Theano. Blocks currently supports and provides constructing parametrized Theano operations (called "bricks"), pattern matching to select variables and bricks in large models, algorithms to optimize the model, saving and resuming of training, monitoring and analyzing values during training progress (on the training set as well as on test sets), and application of graph transformations, such as dropout.

For both the baseline and the proposed systems, we chose three different structures: 50, 100, and 200 hidden neurons in the encoder with the same number of hidden neurons in the decoder. The data is fed to the systems in batches of size 80. Training samples (words) with the length greater than 20 characters are discarded. The source (English) vocabulary is set to 27 and the target (Persian) side has a 35-character vocabulary. Any extra character will be considered as <UNK> (unknown). To detect the beginning and the end of samples, <S> and </S> tags are added to them. This way, the models know where to start and end the process of producing symbols at the target side. To optimize the neural networks, AdaDelta algorithm is used and dropout is applied. In both models, the maximum allowed iteration is set to 5,000. After the 4,000-th iteration, validation based on the BLUE score [12] on the development set is applied in every 250 iterations. The vector representation used in the proposed system consists of 30 real-valued numbers for each character. In order to make the proposed model settle easier to its optimal configuration and prevent oscillation, we have made the initial weights (vector representations) small enough via multiplying them by the factor of 0.1.

Experiments are performed on Nvidia® GeForce® GTX™ Titan X graphics card. This GPU includes 12GB of GDDR5 memory and utilizes 3072 CUDA cores. The experimental results are shown in TABLE II. .

In the baseline system, after a short training time, a reasonable accuracy on the test set is achieved. This shows that the attention-based approach is suitable for transliteration. However, when the network becomes larger by having more hidden units in its structure, we see that the accuracy falls. This is because the network is no longer able to tune its parameters (weights) in the dedicated training time.

On the other hand, we can see that using the vector representation of the characters to initialize the weights in the encoder helps the system to tune its parameters better and converge to its optimal configuration faster. This is especially important when we have a more complex task like machine translation in which the vocabulary size and the needed neural structure are much larger.

As we can see from the results, in all the cases, the proposed system has achieved a better performance than the baseline. The superior result of the proposed system is specifically clear in the 200/200 network structure, where the improvement is 4.21 BLEU points (5.86% relative). This fact confirms that the vector representation of characters used in the proposed system can provide helpful information for the encoder. It is important to note that we can obtain the vector representations only once and use it in the training process. So there is almost no time/computational overhead in the proposed system compared to the baseline. This can be seen in the running times of the systems.

TABLE II. EXPERIMENTAL RESULTS OF THE BASELINE AND THE PROPOSED MODELS

Model	Encoder/Decoder Hidden Units	Training Details			Test BLEU Score
		Iterations	Epochs	Running Time (seconds)	
Baseline	50/50	5,000	27	175	75.42
Word2Vec-Init				169	**76.60**
Baseline	100/100	5,000	27	196	71.51
Word2Vec-Init				197	**75.35**
Baseline	200/200	5,000	27	252	71.83
Word2Vec-Init				250	**76.04**

V. CONCLUSION AND OUTLOOK

In this paper, we aimed at English to Persian transliteration. This task was done by applying the attention-based mechanism previously used in neural machine translation, but at character level. To improve the results, we used the vector representation of characters as initial weights of the encoder. We saw that applying the proposed extension to the baseline system leads to a better performance in all cases.

A first step for the future works can be applying the same idea to the decoder and using the vector representation of the target characters in it. Thanks to the vast power of Titan X graphical processing unit, we will also be able to perform the same experiments on machine translation corpora in the future. However, machine translation has its own complexities like out-of-vocabulary (OOV) words that should be handled. The next step can be proposing a mechanism which overcomes current shortcomings like handling OOVs.

REFERENCES

[1] Bengio, Y. (2009). Learning deep architectures for AI. *Foundations and trends® in Machine Learning*, *2*(1), 1-127.

[2] Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078.*

[3] Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473.*

[4] Schuster, M., & Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *Signal Processing, IEEE Transactions on*, *45*(11), 2673-2681.

[5] Graves, A., Jaitly, N., & Mohamed, A. R. (2013, December). Hybrid speech recognition with deep bidirectional LSTM. In *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on* (pp. 273-278). IEEE.

[6] Shao, Y., & Nivre, J. (2016). Applying Neural Networks to English-Chinese Named EntityTransliteration. In *Sixth Named Entity Workshop, joint with 54th ACL*.

[7] Rosca, M., & Breuel, T. (2016). Sequence-to-sequence neural network models for transliteration. *arXiv preprint arXiv:1610.09565.*

[8] Muscat, O. (2011). The English transliteration of place names in Oman. *Journal of Academic and Applied Studies*, *1*(3), 1-27.

[9] Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781.*

[10] Zhang, M., Li, H., Kumaran, A., Liu, M. (2011). Report of NEWS 2011 Machine Transliteration Shared Task. In *Proceedings of the 5th International Joint Conference on Natural Language Processing (IJCNLP2011)*.

[11] Van Merriënboer, B., Bahdanau, D., Dumoulin, V., Serdyuk, D., Warde-Farley, D., Chorowski, J., & Bengio, Y. (2015). Blocks and fuel: Frameworks for deep learning. *arXiv preprint arXiv:1506.00619.*

[12] Papineni, K., Roukos, S., Ward, T., & Zhu, W. J. (2002, July). BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics* (pp. 311-318). Association for Computational Linguistics.