A Deep Learning Approach to Machine Transliteration

Thomas Deselaers and **Saša Hasan** and **Oliver Bender** and **Hermann Ney**
Human Language Technology and Pattern Recognition Group – RWTH Aachen University
<surname>@cs.rwth-aachen.de

Abstract

In this paper we present a novel transliteration technique which is based on deep belief networks. Common approaches use finite state machines or other methods similar to conventional machine translation. Instead of using conventional NLP techniques, the approach presented here builds on deep belief networks, a technique which was shown to work well for other machine learning problems. We show that deep belief networks have certain properties which are very interesting for transliteration and possibly also for translation and that a combination with conventional techniques leads to an improvement over both components on an Arabic-English transliteration task.

1 Introduction

Transliteration, i.e. the transcription of words such as proper nouns from one language into another or, more commonly from one alphabet into another, is an important subtask of machine translation (MT) in order to obtain high quality output.

We present a new technique for transliteration which is based on deep belief networks (DBNs), a well studied approach in machine learning. Transliteration can in principle be considered to be a small-scale translation problem and, thus, some ideas presented here can be transferred to the machine translation domain as well.

Transliteration has been in use in machine translation systems, e.g. Russian-English, since the existence of the field of machine translation. However, to our knowledge it was first studied as a machine learning problem by Knight and Graehl (1998) using probabilistic finite-state transducers. Subsequently, the performance of this system was greatly improved by combining different spelling and phonetic models (Al-Onaizan and Knight, 2002). Huang et al. (2004) construct a probabilistic Chinese-English edit model as part of a larger alignment solution using a heuristic bootstrapped procedure. Freitag and Khadivi (2007) propose a technique which combines conventional MT methods with a single layer perceptron.

In contrast to these methods which strongly build on top of well-established natural language processing (NLP) techniques, we propose an alternative model. Our new model is based on deep belief networks which have been shown to work well in other machine learning and pattern recognition areas (cf. Section 2). Since translation and transliteration are closely related and transliteration can be considered a translation problem on the character level, we discuss various methods from both domains which are related to the proposed approach in the following.

Neural networks have been used in NLP in the past, e.g. for machine translation (Asunción Castaño et al., 1997) and constituent parsing (Titov and Henderson, 2007). However, it might not be straight-forward to obtain good results using neural networks in this domain. In general, when training a neural network, one has to choose the structure of the neural network which involves certain trade-offs. If a small network with no hidden layer is chosen, it can be efficiently trained but has very limited representational power, and may be unable to learn the relationships between the source and the target language. The DBN approach alleviates some of the problems that commonly occur when working with neural networks: 1. they allow for efficient training due to a good initialisation of the individual layers. 2. Overfitting problems are addressed by creating generative models which are later refined discriminatively. 3. The network structure is clearly defined and only a few structure parameters have to be set. 4. DBNs can be interpreted as Bayesian probabilistic generative models.

Recently, Collobert and Weston (2008) proposed a technique which applies a convolutional DBN to a multi-task learning NLP problem. Their approach is able to address POS tagging, chunking, named entity tagging, semantic role and similar word identification in one model. Our model is similar to this approach in that it uses the same machine learning techniques but the encoding and the processing is done differently. First, we learn two independent generative models, one for the source input and one for the target output. Then, these two models are combined into a source-to-target encoding/decoding system (cf. Section 2).

Regarding that the target is generated and not searched in a space of hypotheses (e.g. in a word graph), our approach is similar to the approach presented by Bangalore et al. (2007) who present an MT system where the set of words of the target sentence is generated based on the full source sentence and then a finite-state approach is used to reorder the words. Opposed to this approach we do not only generate the letters/words in the target sentence but we generate the full sentence with ordering.

We evaluate the proposed methods on an Arabic-English transliteration task where Arabic city names have to be transcribed into the equivalent English spelling.

2 Deep Belief Networks for Transliteration

Although DBNs are thoroughly described in the literature, e.g. (Hinton et al., 2006), we give a short overview on the ideas and techniques and introduce our notation.

Deep architectures in machine learning and artificial intelligence are becoming more and more popular after an efficient training algorithm has been proposed (Hinton et al., 2006), although the idea is known for some years (Ackley et al., 1985). Deep belief networks consist of multiple layers of restricted Boltzmann machines (RBMs). It was shown that DBNs can be used for dimensionality reduction of images and text documents (Hinton and Salakhutdinow, 2006) and for language modelling (Mnih and Hinton, 2007). Recently, DBNs were also used successfully in image retrieval to create very compact but meaningful representations of a huge set of images (nearly 13 million) for retrieval (Torralba et al., 2008).

DBNs are built from RBMs by first training an RBM on the input data. A second RBM is built on the output of the first one and so on until a sufficiently deep architecture is created. RBMs are stochastic generative artificial neural networks with restricted connectivity. From a theoretical viewpoint, RBMs are interesting because they are able to discover complex regularities and find notable features in data (Ackley et al., 1985).

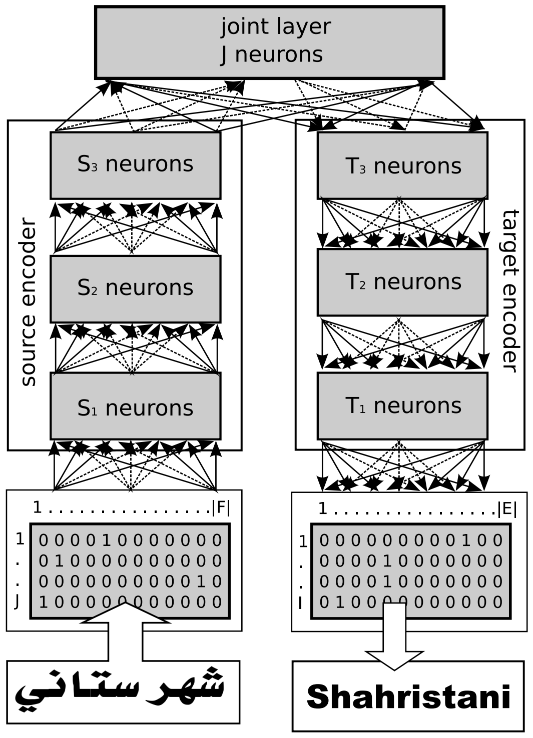

Figure 1: A schematic representation of our DBN for transliteration.

Hinton and Salakhutdinow (2006) present a deep belief network to learn a tiny representation of its inputs and to reconstruct the input with high accuracy which is demonstrated for images and textual documents. Here, we use DBNs similarly: first, we learn encoders for the source and target words respectively and then connect these two through a joint layer to map between the two languages. This joint layer is trained in the same way as the top-level neurons in the deep belief classifier from (Hinton et al., 2006).

In Figure 1, a schematic view of our DBN for transliteration is shown. On the left and on the right are encoders for the source and target words respectively. To transliterate a source word, it is passed through the layers of the network. First, it traverses through the source encoder on the left, then it passes into the joint layer, finally traversing down through the target encoder. Each layer consists of a set of neurons receiving the output of the preceding layer as input. The first layers in the source and target encoders consist of S_1 and T_1 neurons, respectively; the second layers have S_2 and T_2 nodes, and the third layers have S_3 and T_3 nodes, respectively. A joint layer with J nodes connects the source and the target encoders.

Here, the number of nodes in the individual layers are the most important parameters. The more nodes a layer has, the more information can be conveyed through it, but the harder the training: the amount of data needed for training and thus the computation time required is exponential in the size of the network (Ackley et al., 1985).

To transliterate a source word, it is first encoded as a D_F-dimensional binary vector S_F (cf. Section 2.1) and then fed into the first layer of the source encoder. The S_1-dimensional output vector O_{S1} of the first layer is computed as

$$O_{S1} \leftarrow 1/\exp(1 + w_{S1}S_F + b_{S1}), \quad (1)$$

where w_{S1} is a $S_1 \times D_F$-dimensional weight matrix and b_{S1} is an S_1-dimensional bias vector.

The output of each layer is used as input to the next layer as follows:

$$\begin{aligned} O_{S2} &\leftarrow 1/\exp(1 + w_{S2}O_{S1} + b_{S2}), && (2)\\ O_{S3} &\leftarrow 1/\exp(1 + w_{S3}O_{S2} + b_{S3}). && (3) \end{aligned}$$

After the source encoder has been traversed, the joint layer is reached which processes the data twice: once using the input from the source encoder to get a state of the hidden neurons O_{SJ} and then to infer an output state O_{JT} as input to the topmost level of the output encoder

$$\begin{aligned} O_{SJ} &\leftarrow 1/\exp(1 + w_{SJ}O_{S3} + b_{SJ}), && (4)\\ O_{JT} &\leftarrow 1/\exp(1 + w_{JT}O_{SJ} + b_{JT}). && (5) \end{aligned}$$

This output vector is decoded by traversing downwards through the output encoder:

$$\begin{aligned} O_{T3} &\leftarrow 1/\exp(1 + w_{T3}O_{JT} + b_{T3}), && (6)\\ O_{T2} &\leftarrow 1/\exp(1 + w_{T2}O_{T3} + b_{T2}), && (7)\\ O_{T1} &\leftarrow w_{T1}O_{T2} + b_{T1}, && (8) \end{aligned}$$

where O_{T1} is a vector encoding a word in the target language.

Note that this model is intrinsically bidirectional since the individual RBMs are bidirectional models and thus it is possible to transliterate from source to target and vice versa.

2.1 Source and Target Encoding

A problem with DBNs and transliteration is the data representation. The input and output data are commonly sequences of varying length but a DBN expects input data of constant length. To represent a source or target language word, it is converted into a sparse binary vector of dimensionality $D_F = |F| \cdot J$ or $D_E = |E| \cdot I$, respectively, where $|F|$ and $|E|$ are the sizes of the alphabets and I and J are the lengths of the longest words. If a word is shorter than this, a *padding letter* w_0 is used to fill the spaces. This encoding is depicted in the bottom part of Figure 1.

Since the output vector of the DBN is not binary, we infer the maximum a posterior hypothesis by selecting the letter with the highest output value for each position.

2.2 Training Method

For the training, we follow the method proposed in (Hinton et al., 2006). To find a good starting point for backpropagation on the whole network, each of the RBMs is trained individually. First, we learn the generative encoders for the source and target words, i.e. the weights w_{S1} and w_{T1}, respectively. Therefore, each of the layers is trained as a restricted Boltzmann machine, such that it learns to generate the input vectors with high probability, i.e. the weights are learned such that the data values have low values of the trained cost function.

After learning a layer, the activity vectors of the hidden units, as obtained from the real training data, are used as input data for the next layer. This can be repeated to learn as many hidden layers as desired. After learning multiple hidden layers in this way, the whole network can be viewed as a single, multi-layer generative model and each additional hidden layer improves a lower bound on the probability that the multi-layer model would generate the training data (Hinton et al., 2006).

For each language, the output of the first layer is used as input to learn the weights of the next layers w_{S2} and w_{T2}. The same procedure is repeated to learn w_{S3} and w_{T3}. Note that so far no connection between the individual letters in the two alphabets is created but each encoder only learns feature functions to represent the space of possible source and target words. Then, the weights for the joined layer are learned using concatenated outputs of the top layers of the source and target encoders to find an initial set of weights w_{SJ} and w_{JT}.

After each of the layers has been trained individually, backpropagation is performed on the whole network to tune the weights and to learn the connections between both languages. We use the average squared error over the output vectors between reference and inferred words as the training criterion. For the training, we split the training data into batches of 100 randomly selected words and allow for 10 training iterations of the individual layers and up to 200 training iterations for the backpropagation. Currently, we only optimise the parameters for the source to target direction and thus do not retain the bidirectionality[1].

Thus, the whole training procedure consists of 4 phases. First, an autoencoder for the source words is learnt. Second, an autoencoder for the target words is learnt. Third, these autoencoders are connected by a top connecting layer, and finally backpropagation is performed over the whole network for fine-tuning of the weights.

2.3 Creation of *n*-Best Lists

N-best lists are a common means for combination of several systems in natural language processing and for rescoring. In this section, we describe how a set of hypotheses can be created for a given input. Although these hypotheses are not n-best lists because they have not been obtained from a search process, they can be used similarly and can better be compared to randomly sampled '*good*' hypotheses from a full word-graph.

Since the values of the nodes in the individual layers are probabilities for this particular node to be activated, it is possible to sample a set of states from the distribution for the individual layers, which is called Gibbs sampling (Geman and Geman, 1984). This sampling can be used to create several hypotheses for a given input sentence, and this set of hypotheses can be used similar to an n-best list.

The layer in which the Gibbs sampling is done can in principle be chosen arbitrarily. However, we believe it is natural to sample in either the first layer, the joint layer, or the last layer. Sampling in the first layer leads to different features traversing the full network. Sampling in the joint layer only affects the generation of the target sentence, and sampling in the last layer is equal to directly sampling from the distribution of target hypotheses.

Conventional Gibbs sampling has a very strong impact on the outcome of the network because the smoothness of the distributions and the encoding of similar matches is entirely lost. Therefore, we use a weak variant of Gibbs sampling. Instead of replacing the states' probabilities with fully discretely sampled states, we keep the probabilities and add a fraction of a sampled state, effectively modifying the probabilities to give a slightly better score to the last sampled state. Let p be the D-dimensional vector of probabilities for D nodes in an RBM to be on. Normal Gibbs sampling would sample a D-dimensional vector S containing a state for each node from this distribution. Instead of replacing the vector p with S, we use $p' \leftarrow p + \varepsilon S$, leading to smoother changes than conventional Gibbs sampling. This process can easily be repeated to obtain multiple hypotheses.

3 Experimental Evaluation

In this section we present experimental results for an Arabic-English transliteration task. For evaluation we use the character error rate (CER) which is the commonly used word error rate (WER) on character level.

We use a corpus of 10,084 personal names in Arabic and their transliterated English ASCII representation (LDC corpus LDC2005G02). The Arabic names are written in the usual way, i.e. lacking vowels and diacritics. 1,000 names were randomly sampled for development and evaluation, respectively (Freitag and Khadivi, 2007). The vocabulary of the source language is 33 and the target language has 30 different characters (including the padding character). The longest word on both sides consists of 14 characters, thus the feature vector on the source side is 462-dimensional and the feature vector on the target side is 420-dimensional.

3.1 Network Structure

First, we evaluate how the structure of the network should be chosen. For these experiments, we fixed the numbers of layers and the size of the bottom layers in the target and source encoder and evaluate different network structures and the size of the joint layer.

The experiments we performed are described in Table 1. The top part of the table gives the results for different network structures. We compare networks with increasing layer sizes, identical layer sizes, and decreasing layer sizes. It can be seen that decreasing layer sizes leads to the best results. In these experiments, we choose the number of nodes in the joint layer to be three times as large as the topmost encoder layers.

In the bottom part, we kept most of the network structure fixed and only vary the number of nodes

[1]Note that it is easily possible to extend the backpropagation to include both directions, but to keep the computational demands lower we decided to start with only one direction.

Table 1: Transliteration experiments using different network structures.

number of nodes				CER [%]		
S_1,T_1	S_2,T_2	S_3,T_3	J	train	dev	eval
400	500	600	1800	0.3	27.2	28.1
400	400	400	1200	0.7	26.1	25.2
400	350	300	900	1.8	25.1	24.3
400	350	300	1000	1.7	24.8	24.0
400	**350**	**300**	**1500**	**1.3**	**24.1**	**22.7**
400	350	300	2000	0.2	24.2	23.5

in the joint layer. Here, a small number of nodes leads to suboptimal performance and a very high number of nodes leads to overfitting which can be seen in nearly perfect performance on the training data and an increasing CER on the development and eval data.

3.2 Network Size

Next, we evaluate systems with different numbers of nodes. Therefore, we start from the best parameters (400-350-300-1500) from the previous section and scale the number of nodes in the individual layers by a certain factor, i.e. factor 1.5 leads to (600-525-450-2250).

In Figure 2 and Table 2, the results from the experimental evaluation on the transliteration task are given. The network size denotes the number of nodes in the bottom layers of the source and the target encoder (i.e. S_1 and T_1) and the other layers are chosen according to the results from the experiments presented in the previous section.

The results show that small networks perform badly, the optimal performance is reached with medium sized networks of 400-600 nodes in the bottom layers, and larger networks perform worse, which is probably due to overfitting.

For comparison, we give results for a state-of-the-art phrase-based MT system applied on the character level with default system parameters (labelled as 'PBT untuned'), and the same system, where all scaling factors were tuned on dev data (labelled as 'PBT tuned'). The tuned phrase-based MT system clearly outperforms our approach.

Additionally, we perform an experiment with a standard multi-layer perceptron. Therefore, we choose the network structure with 400-350-300-1500 nodes, initialised these randomly and trained the entire network with backpropagation training.

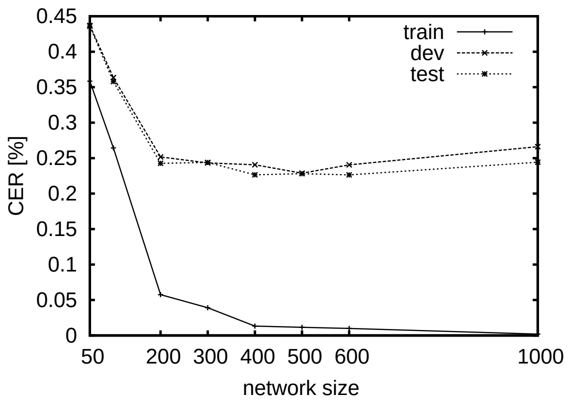

Figure 2: Results for the Arabic-English transliteration task depending on the network size.

The results (line 'MLP-400' in Table 2) of this experiment are far worse than any of the other results, which shows that, apart from the convenient theoretical interpretation, the creation of the DBN as described is a suitable method to train the system. The reason for the large difference is likely the bad initialisation of the network and the fact that the backpropagation algorithm gets stuck in a local optimum at this point.

3.3 Reordering capabilities

Although reordering is not an issue in transliteration, the proposed model has certain properties which we investigated and where interesting properties can be observed.

To investigate the performance under adverse reordering conditions, we also perform an experiment with reversed ordering of the target letters (i.e. a word $w = c_1, c_2, \ldots, c_J$ is now written $c_J, c_{J-1}, \ldots, c_1$). Since the DBN is fully symmetric, i.e. each input node is connected with each output node in the same way and vice versa, the DBN result is not changed except for some minor numerical differences due to random initialisation. Indeed, the DBN obtained is nearly identical except for a changed ordering of the weights in the joint layer, and if desired it is possible to construct a DBN for reverse-order target language from a fully trained DBN by permuting the weights.

On the same setup an experiment with our phrase-based decoder has been performed and here the performance is strongly decreased (bottom line of Table 2). The phrase-based MT system for this experiment used a reordering with IBM block-limit constraints with distortion limits and all default parameters were reasonably tuned. We observed that the position-independent

Table 2: Results for the Arabic-English transliteration task depending on the network size and a comparison with state of the art results using conventional phrase-based machine translation techniques

network size	CER [%] train	dev	eval
50	35.8	43.7	43.6
100	26.4	36.3	35.8
200	5.8	25.2	24.3
300	3.9	24.3	24.4
400	1.3	24.1	22.7
500	1.2	22.9	22.8
600	1.0	24.1	22.6
1000	0.2	26.6	24.4
MLP-400	22.0	64.1	63.2
untuned PBT	4.9	23.3	23.6
tuned PBT	2.2	12.9	13.3
(Freitag and Khadivi, 2007)	n/a	11.1	11.1
reversed task: PBT	13.0	35.2	35.7

error rate of the phrase-based MT system is hardly changed which also underlines that, in principle, the phrase-based MT system is currently better but that under adverse reordering conditions the DBN system has some advantages.

3.4 *N*-Best Lists

As described above, different possibilities to create n-best lists exists. Starting from the system with 400-350-300-1500 nodes, we evaluate the creation of n-best lists in the first source layer, the joint layer, and the last target layer. Therefore, we create n best lists with up to 10 hypotheses (sometimes, we have less due to duplicates after sampling, on the average we have 8.3 hypotheses per sequence), and evaluate the oracle error rate. In Table 3 it can be observed that sampling in the first layer leads to the best oracle error rates. The baseline performance (first best) for this system is 24.1% CER on the development data, and 22.7% CER on the eval data, which can be improved by nearly 10% absolute using the oracle from a 10-best list.

3.5 Rescoring

Using the n-best list sampled in the first source layer, we also perform rescoring experiments. Therefore, we rescore the transliteration hypotheses (after truncating the padding letters w_0) with additional models, which are commonly used in MT, and which we have trained on the training data:

Table 3: Oracle character error rates on 10-best lists.

sampling layer	oracle CER [%] dev	eval
S_1	15.8	14.8
joint layer	17.5	16.4
T_1	18.7	18.2

System		CER [%] dev	eval
DBN	w/o rescoring	24.1	22.7
	w/ rescoring	21.3	20.1

Table 4: Results from the rescoring experiments and fusion with the phrase-based MT system.

- IBM model 1 lexical probabilities modelling the probability for a target sequence given a source sequence

$$h_{\text{IBM1}}(f_1^J, e_1^I) = -\log\left(\frac{1}{(I+1)^J}\prod_{j=1}^{J}\sum_{i=0}^{I}p(f_j|e_i)\right)$$

- m-gram language model over the letter sequences

$$h_{\text{LM}}(e_1^I) = -\log\prod_{i=1}^{I}p(e_i|e_{i-m+1}^{i-1}),$$

  with m being the size of the m-gram, we choose $m = 9$.

- sequence length model (commonly referred to as word penalty).

Then, these models are fused in a log-linear model (Och and Ney, 2002), and we tune the model scaling factors discriminatively on the development n-best list using the downhill simplex algorithm. Results from the rescoring experiments are given in Table 4.

The performance of the DBN system is improved on the dev data from 24.1% to 21.3% CER and on the eval data from 22.7% to 20.1% CER.

3.6 Application Within a System Combination Framework

Although being clearly outperformed by the phrase-based MT system, we applied the transliteration candidates generated by the DBN approach within a system combination framework. Motivated by the fact that the DBN approach differs decisively from the other statistical approaches we applied to the machine transliteration task, we wanted to investigate the potential benefit of the diverse nature of the DBN transliterations. Taking the transliteration candidates obtained from another study which was intended to perform a comparison of various statistical approaches to the transliteration task, we performed the system combination as is customary in speech recognition, i.e. following the Recognizer Output Voting Error Reduction (ROVER) approach (Fiscus, 1997).

The following methods were investigated:

(Monotone) Phrase-based MT on character level: A state-of-the-art phrase-based SMT system (Zens and Ney, 2004) was used for name transliteration, i.e. translation of characters instead of words. No reordering model was employed due to the monotonicity of the transliteration task, and the model scaling factors were tuned on maximum transliteration accuracy.

Data-driven grapheme-to-phoneme conversion: In Grapheme-to-Phoneme conversion (G2P), or phonetic transcription, we seek the most likely pronunciation (phoneme sequence) for a given orthographic form (sequence of letters). Then, a grapheme-phoneme joint multi-gram, or *graphone* for short, is a pair of a letter sequence and a phoneme sequence of possibly different length (Bisani and Ney, 2008). The model training is done in two steps: First, maximum likelihood is used to infer the graphones. Second, the input is segmented into a stream of graphones and absolute discounting with leaving-one-out is applied to estimate the actual M-gram model. Interpreting the characters of the English target names as phonemes, we used the G2P toolkit of (Bisani and Ney, 2008) to transliterate the Arabic names.

Position-wise maximum entropy models / CRFs: The segmentation as provided by the G2P model is used and "null words" are inserted such that the transliteration task can be interpreted as a classical tagging task (e.g. POS, conceptual tagging, etc.). This means that we seek for a one-to-one mapping and define feature functions to model the posterior probability. Maximum entropy (ME) models are defined position-wise, whereas conditional random fields (CRFs) consider full sequences. Both models were trained according to the maximum class posterior criterion. We used an ME tagger (Bender et al., 2003) and the freely available CRF++ toolkit.[2]

Results for each of the individual systems and different combinations are given in Table 5. As expected, the DBN transliterations cannot keep up with the other approaches. The additional models (G2P, CRF and ME) perform slightly better than the PBT method. If we look at combinations of systems without the DBN approach, we observe only marginal improvements of around 0.1-0.2% CER. Interestingly, a combination of all 4 models (PBT, G2P, ME, CRF) works as good as individual 3-way combinations (the same 11.9% on dev are obtained). This can be interpreted as a potential "similarity" of the approaches. Adding e.g. ME to a combination of PBT, G2P and CRF does not improve results because the transliteration hypotheses are too similar. If we simply put together all 5 systems including DBN with equal weights, we have a similar trend. Since all systems are equally weighted and at least 3 of the systems are similar in individual performance (G2P, ME, CRF have all around 12% CER on the tested data sets), the DBN approach does not get a large impact on overall performance.

If we drop similar systems and tune for 3-way combinations, we observe a large reduction in CER if DBN comes into play. Compared to the best individual system of 12% CER, we now arrive at a CER of 10.9% for a combination of PBT, CRF and DBN which is significantly better than each of the individual methods. Our interpretation of this is that the DBN system has different hypotheses compared to all other systems and that the hypotheses from the other systems are too similar to be apt for combination. So, although DBN is much worse than the other approaches, it obviously helps in the system combination. Using the rescored variant of the DBN transliterations from

[2]`http://crfpp.sourceforge.net/`

System	CER [%] dev	CER [%] eval
DBN	24.1	22.7
PBT	12.9	13.3
G2P	12.2	12.1
ME	12.3	12.4
CRF	12.0	12.0
ROVER		
best setting w/o DBN	11.9	11.8
5-way equal weights	11.7	11.9
best setting w/ DBN	**10.9**	**10.9**

Table 5: Results from the individual methods investigated versus ROVER combination.

Section 3.5, performance is similar to the one obtained for the DBN baseline.

4 Discussion and Conclusion

We have presented a novel method for machine transliteration based on DBNs, which despite not having competitive results can be an important additional cue for system combination setups. The DBN model has some immediate advantages: the model is in principle fully bidirectional and is based on sound and valid theories from machine learning. Instead of common techniques which are based on finite-state machines or phrase-based machine translation, the proposed system does not rely on word alignments and beam-search decoding and has interesting properties regarding the reordering of sequences. We have experimentally evaluated the network structure and size, reordering capabilities, the creation of multiple hypotheses, and rescoring and combination with other transliteration approaches. It was shown that, albeit the approach cannot compete with the current state of the art, deep belief networks might be a learning framework with some potential for transliteration. It was also shown that the proposed method is suited for combination with different state-of-the-art systems and that improvements over the single models can be obtained in a ROVER-like setting. Furthermore, adding DBN-based transliterations, although individually far behind the other approaches, significantly improves the overall results by 1% absolute.

Outlook

In the future we plan to investigate several details of the proposed model: we will exploit the inherent bidirectionality, further investigate the structure of the model, such as the number of layers and the numbers of nodes in the individual layers. Also, it is important to improve the efficiency of our implementation to allow for working on larger datasets and obtain more competitive results. Furthermore, we are planning to investigate convolutional input layers for transliteration and use a translation approach analogous to the one proposed by Collobert and Weston (2008) in order to allow for the incorporation of reorderings, language models, and to be able to work on larger tasks.

Acknowledgement. We would like to thank Geoffrey Hinton for providing the Matlab Code accompanying (Hinton and Salakhutdinow, 2006).

References

D. Ackley, G. Hinton, and T. Sejnowski. 1985. A learning algorithm for Boltzmann machines. *Cognitive Science*, 9(1):147–169.

Y. Al-Onaizan and K. Knight. 2002. Machine transliteration of names in Arabic text. In *ACL 2002 Workshop on Computationaal Approaches to Semitic Languages*.

M. Asunción Castaño, F. Casacuberta, and E. Vidal. 1997. Machine translation using neural networks and finite-state models. In *Theoretical and Methodological Issues in Machine Translation (TMI)*, pages 160–167, Santa Fe, NM, USA, July.

S. Bangalore, P. Haffner, and S. Kanthak. 2007. Statistical machine translation through global lexical selection and sentence reconstruction. In *Annual Meeting of the Association for Computational Lingustic (ACL)*, pages 152–159, Prague, Czech Republic.

O. Bender, F. J. Och, and H. Ney. 2003. Maximum entropy models for named entity recognition. In *Proc. 7th Conf. on Computational Natural Language Learning (CoNLL)*, pages 148–151, Edmonton, Canada, May.

M. Bisani and H. Ney. 2008. Joint-sequence models for grapheme-to-phoneme conversion. *Speech Communication*, 50(5):434–451, May.

R. Collobert and J. Weston. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *International Conference on Machine Learning*, Helsinki, Finnland, July.

J. Fiscus. 1997. A post-processing system to yield reduced word error rates: Recognizer output voting error reduction (ROVER). In *IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, pages 347–354, Santa Barbara, CA, USA, December.

D. Freitag and S. Khadivi. 2007. A sequence alignment model based on the averaged perceptron. In *Conference on Empirical methods in Natural Language Processing*, pages 238–247, Prague, Czech Republic, June.

S. Geman and D. Geman. 1984. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 6(6):721–741, November.

G. Hinton and R. R. Salakhutdinow. 2006. Reducing the dimensionality of data with neural networks. *Science*, 313:504–507, July.

G. Hinton, S. Osindero, and Y.-W. Teh. 2006. A fast learning algorithm for deep belief nets. *Neural Computation*, 18:1527–1554.

F. Huang, S. Vogel, and A. Waibel. 2004. Improving named entity translation combining phonetic and semantic similarities. In *HLT-NAACL*.

K. Knight and J. Graehl. 1998. Machine transliteration. *Computational Linguistics*, 24(2).

A. Mnih and G. Hinton. 2007. Three new graphical models for statistical language modelling. In *ICML '07: International Conference on Machine Learning*, pages 641–648, New York, NY, USA. ACM.

F. Och and H. Ney. 2002. Discriminative training and maximum entropy models for statistical machine translation. In *Annual Meeting of the Assoc. for Computational Linguistics*, pages 295–302, Philadelphia, PA, USA, July.

I. Titov and J. Henderson. 2007. Constituent parsing with incremental sigmoid belief networks. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 632–639, Prague, Czech Republic, June.

A. Torralba, R. Fergus, and Y. Weiss. 2008. Small codes and large image databases for recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, Anchorage, AK, USA, June.

R. Zens and H. Ney. 2004. Improvements in phrase-based statistical machine translation. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics: HLT-NAACL 2004*, pages 257–264, Boston, MA, May.