Additive Neural Networks for Statistical Machine Translation

Lemao Liu[1], **Taro Watanabe**[2], **Eiichiro Sumita**[2], **Tiejun Zhao**[1]
[1]School of Computer Science and Technology
Harbin Institute of Technology (HIT), Harbin, China
[2]National Institute of Information and Communication Technology (NICT)
3-5 Hikari-dai, Seika-cho, Soraku-gun, Kyoto, Japan
{lmliu | tjzhao}@mtlab.hit.edu.cn
{taro.watanabe | eiichiro.sumita}@nict.go.jp

Abstract

Most statistical machine translation (SMT) systems are modeled using a log-linear framework. Although the log-linear model achieves success in SMT, it still suffers from some limitations: (1) the features are required to be linear with respect to the model itself; (2) features cannot be further interpreted to reach their potential. A neural network is a reasonable method to address these pitfalls. However, modeling SMT with a neural network is not trivial, especially when taking the decoding efficiency into consideration. In this paper, we propose a variant of a neural network, i.e. additive neural networks, for SMT to go beyond the log-linear translation model. In addition, word embedding is employed as the input to the neural network, which encodes each word as a feature vector. Our model outperforms the log-linear translation models with/without embedding features on Chinese-to-English and Japanese-to-English translation tasks.

1 Introduction

Recently, great progress has been achieved in SMT, especially since Och and Ney (2002) proposed the log-linear model: almost all the state-of-the-art SMT systems are based on the log-linear model. Its most important advantage is that arbitrary features can be added to the model. Thus, it casts complex translation between a pair of languages as feature engineering, which facilitates research and development for SMT.

Regardless of how successful the log-linear model is in SMT, it still has some shortcomings. On the one hand, features are required to be linear with respect to the objective of the translation model (Nguyen et al., 2007), but it is not guaranteed that the potential features be linear with the model. This induces modeling inadequacy (Duh and Kirchhoff, 2008), in which the translation performance may not improve, or may even decrease, after one integrates additional features into the model. On the other hand, it cannot deeply interpret its surface features, and thus can not efficiently develop the potential of these features. What may happen is that a feature p does initially not improve the translation performance, but after a nonlinear operation, e.g. $log(p)$, it does. The reason is not because this feature is useless but the model does not efficiently interpret and represent it. Situations such as this confuse explanations for feature designing, since it is unclear whether such a feature contributes to a translation or not.

A neural network (Bishop, 1995) is a reasonable method to overcome the above shortcomings. However, it should take constraints, e.g. the decoding efficiency, into account in SMT. Decoding in SMT is considered as the expansion of translation states and it is handled by a heuristic search (Koehn, 2004a). In the search procedure, frequent computation of the model score is needed for the search heuristic function, which will be challenged by the decoding efficiency for the neural network based translation model. Further, decoding with non-local (or state-dependent) features, such as a language model, is also a problem. Actually, even for the (log-) linear model, efficient decoding with the language model is not trivial (Chiang, 2007).

In this paper, we propose a variant of neural networks, i.e. additive neural networks (see Section 3 for details), for SMT. It consists of two components: a linear component which captures non-local (or state dependent) features and a non-linear component (i.e., neural nework) which encodes lo-

This joint work was done while the first author visited NICT.

Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics, pages 791–801,
Sofia, Bulgaria, August 4-9 2013.

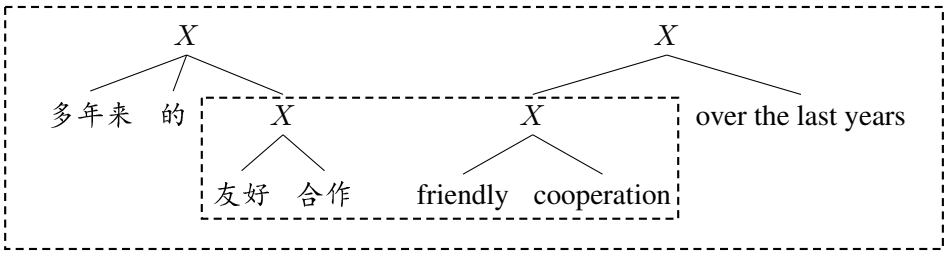

Figure 1: A bilingual tree with two synchronous rules, $r_1 : X \rightarrow \langle$友好 合作; friendly cooperation$\rangle$ and $r_2 : X \rightarrow \langle$多年来 的 $X; X$ over the last years$\rangle$. The inside rectangle denotes the partial derivation $d_1 = \{r_1\}$ with the partial translation e_1 ="friendly cooperation", and the outside rectangle denotes the derivation $d_2 = \{r_1, r_2\}$ with the translation e_2="friendly cooperation over the last years".

cal (or state independent) features. Compared with the log-linear model, it has more powerful expressive abilities and can deeply interpret and represent features with hidden units in neural networks. Moreover, our method is simple to implement and its decoding efficiency is comparable to that of the log-linear model. We also integrate word embedding into the model by representing each word as a feature vector (Collobert and Weston, 2008). Because of the thousands of parameters and the non-convex objective in our model, efficient training is not simple. We propose an efficient training methodology: we apply the mini-batch conjugate sub-gradient algorithm (Le et al., 2011) to accelerate the training; we also propose pre-training and post-training methods to avoid poor local minima. The biggest contribution of this paper is that it goes beyond the log-linear model and proposes a non-linear translation model instead of re-ranking model (Duh and Kirchhoff, 2008; Sokolov et al., 2012).

On both Chinese-to-English and Japanese-to-English translation tasks, experiment results show that our model can leverage the shortcomings suffered by the log-linear model, and thus achieves significant improvements over the log-linear based translation.

2 Log-linear Model, Revisited

2.1 Log-linear Translation Model

Och and Ney (2002) proposed the log-linear translation model, which can be formalized as follows:

$$\mathbf{P}(e, d|f; W) = \frac{\exp\left\{W^\top \cdot h(f, e, d)\right\}}{\sum_{e',d'} \exp\left\{W^\top \cdot h(f, e', d')\right\}}, \quad (1)$$

where f denotes the source sentence, and $e(e')$ denotes its translation candidate; $d(d')$ is a derivation over the pair $\langle f, e\rangle$, i.e., a collection of synchronous rules for Hiero grammar (Chiang, 2005), or phrase pairs in Moses (Koehn et al., 2007); $h(f, e, d) = (h_1(f, e, d), h_2(f, e, d), \cdots, h_K(f, e, d))^\top$ is a K-dimensional feature vector defined on the tuple $\langle f, e, d\rangle$; $W = (w_1, w_2, \cdots, w_K)^\top$ is a K-dimensional weight vector of h, i.e., the parameters of the model, and it can be tuned by the toolkit MERT (Och, 2003). Different from Brown's generative model (Brown et al., 1993), the log-linear model does not assume strong independency holds, and allows arbitrary features to be integrated into the model easily. In other words, it can transform complex language translation into feature engineering: it can achieve high translation performance if reasonable features are chosen and appropriate parameters are assigned for the weight vector.

2.2 Decoding By Search

Given a source sentence f and a weight W, decoding finds the best translation candidate $\hat{e}$ via the programming problem:

$$\begin{aligned}\langle \hat{e}, \hat{d}\rangle &= \arg\max_{e,d} \mathbf{P}(e, d|f; W) \\ &= \arg\max_{e,d} \left\{W^\top \cdot h(f, e, d)\right\}. \end{aligned} \quad (2)$$

Since the range of $\langle e, d\rangle$ is exponential with respect to the size of f, the exact decoding is intractable and an inexact strategy such as beam search is used instead in practice.

The idea of search for decoding can be shown in **Figure** 1: it encodes each search state as a partial translation together with its derivation, e.g. $\langle e_1, d_1\rangle$; it consequently expands the states from the initial ($empty$) state to the end state $\langle e_2, d_2\rangle$ according to the translation rules r_1 and r_2. During the state expansion process, the score $w_i \cdot$

$h_i(f, e, d)$ for a partial translation is calculated repeatedly. In the log-linear model, if $h_i(f, e, d)$ is a local feature, the calculation of its score $w_i \cdot h_i(f, e, d)$ has a substructure, and thus it can be calculated with dynamic programming which accelerates its decoding. For the non-local features such as the language model, Chiang (2007) proposed a cube-pruning method for efficient decoding. The main reason why cube-pruning works is that the translation model is linear and the model score for the language model is approximately monotonic (Chiang, 2007).

3 Additive Neural Networks

3.1 Motivation

Although the log-linear model has achieved great progress for SMT, it still suffers from some pitfalls: it requires features be linear with the model and it can not interpret and represent features deeply. The neural network model is a reasonable method to overcome these pitfalls. However, the neural network based machine translation is far from easy.

As mentioned in Section 2, the decoding procedure performs an expansion of translation states. Firstly, let us consider a simple case in neural network based translation where all the features in the translation model are independent of the translation state, i.e. all the components of the vector $h(f, e, d)$ are local features. In this way, we can easily define the following translation model with a single-layer neural network:

$$\mathbf{S}(f, e, d; W, M, B) = W^\top \cdot \sigma(M \cdot h(f, e, d) + B), \quad (3)$$

where $M \in \mathbb{R}^{u \times K}$ is a matrix, and $B \in \mathbb{R}^u$ is a vector, i.e. bias; σ is a single-layer neural network with u hidden units, i.e. an element wise sigmoid function $sigmoid(x) = 1/\big(1 + \exp(-x)\big)$. For consistent description in the rest, we also represent Eq. (3) as a function of a feature vector h, i.e. $\mathbf{S}(h; W, M, B) = W^\top \cdot \sigma(M \cdot h + B)$.

Now let us consider the search procedure with the model in Eq. (3) using **Figure** 1 as our example. Suppose the current translation state is encoded as $\langle e_1, d_1 \rangle$, which is expanded into $\langle e_2, d_2 \rangle$ using the rule r_2 ($d_2 = d_1 \cup \{r_2\}$). Since h is state-independent, $h(f, e_2, d_2) = h(f, e_1, d_1) + h(r_2)$. However, since $S(f, e, d; W, M, B)$ is non-decomposable as a linear model, there is no substructure for calculating $S(f, e_2, d_2; W, M, B)$, and one has to re-calculate it via Eq. (3) even if the score of $S(f, e_1, d_1; M, B)$ for its previous state $\langle e_1, d_1 \rangle$ is available. When the size of the parameter (W, M, B) is relatively large, it will be a challenge for the decoding efficiency.

In order to keep the substructure property, $S(f, e_2, d_2; W, M, B)$ should be represented as $F\big(S(f, e_1, d_1; W, M, B); S(h(r_2); M, B)\big)$ by a function F. For simplicity, we suppose that the additive property holds in F, and then we can obtain a new translation model via the following recursive equation:

$$S(f, e_2, d_2; W, M, B) = S(f, e_1, d_1; W, M, B) + S\big(h(r_2); W, M, B\big). \quad (4)$$

Since the above model is defined only on local features, it ignores the contributions from non-local features. Actually, existing works empirically show that some non-local features, especially language model, contribute greatly to machine translation.

Scoring for non-local features such as a n-gram language model is not easily done. In log-linear translation model, Chiang (2007) proposed a cube-pruning method for scoring the language model. The premise of cube-pruning is that the language model score is approximately monotonic (Chiang, 2007). However, if scoring the language model with a neural network, this premise is difficult to hold. Therefore, one of the solutions is to preserve a linear model for scoring the language model directly.

3.2 Definition

According to the above analysis, we propose a variant of a neural network model for machine translation, and we call it **Ad**ditive **N**eural **N**etworks or **AdNN** for short.

The AdNN model is a combination of a linear model and a neural network: non-local features, e.g. LM, are linearly modeled for the cube-pruning strategy, and local features are modeled by the neural network for deep interpretation and representation. Formally, the AdNN based translation model is discriminative but non-probabilistic, and it can be defined as follows:

$$S(f, e, d; \theta) = W^\top \cdot h(f, e, d) + \sum_{r \in d} W'^\top \cdot \sigma\big(M \cdot h'(r) + B\big), \quad (5)$$

where h and h' are feature vectors with dimension K and K' respectively, and each component of h' is a local feature which can be defined on a rule $r : X \rightarrow \langle \alpha, \gamma \rangle$; $\theta = (W, W', M, B)$ is the model parameters with $M \in \mathbb{R}^{u \times K'}$. In this paper, we focus on a single-layer neural network for its simplicity, and one can similarly define σ as a multi-layer neural network.

Again for the example shown in **Figure** 1, the model score defined in Eq. (5) for the pair $\langle e_2, d_2 \rangle$ can be represented as follows:

$$S(f, e_2, d_2; \theta) = W^\top \cdot h(f, e_2, d_2) + W'^\top \cdot \sigma\big(M \cdot h'(r_1) + B\big) + W'^\top \cdot \sigma\big(M \cdot h'(r_2) + B\big).$$

Eq. (5) is similar to both additive models (Buja et al., 1989) and generalized additive neural networks (Potts, 1999): it consists of many additive terms, and each term is either a linear or a non-linear (a neural network) model. That is the reason why our model is called "additive neural networks". Of course, our model still has some differences from both of them. Firstly, our model is decomposable with respect to rules instead of the component variables. Secondly, some of its additive terms share the same parameters (M, B).

There are also strong relationships between AdNN and the log-linear model. If we consider the parameters (M, B) as constant and $\sigma\big(M \cdot h'(r) + B\big)$ as a new feature vector, then AdNN is reduced to a log-linear model. Since both (M, B) and (W, W') are parameters in AdNN, our model can jointly learn the feature $\sigma\big(M \cdot h'(r) + B\big)$ and tune the weight (W, W') of the log-linear model together. That is different from most works under the log-linear translation framework, which firstly learn features or sub-models and then tune the log-linear model including the learned features in two separate steps. By joint training, AdNN can learn the features towards the translation evaluation metric, which is the main advantage of our model over the log-linear model.

In this paper, we apply our AdNN model to hierarchical phrase based translation, and it can be similarly applied to phrase-based or syntax-based translation. Similar to Hiero (Chiang, 2005), the feature vector h in Eq. (5) includes 8 default features, which consist of translation probabilities, lexical translation probabilities, word penalty, glue rule penalty, synchronous rule penalty and language model. These default features are included because they empirically perform well in the log-linear model. For the local feature vector h' in Eq (5), we employ word embedding features as described in the following subsection.

3.3 Word Embedding features for AdNN

Word embedding can relax the sparsity introduced by the lexicalization in NLP, and it improves the systems for many tasks such as language model, named entity recognition, and parsing (Collobert and Weston, 2008; Turian et al., 2010; Collobert, 2011). Here, we propose embedding features for rules in SMT by combining word embeddings.

Firstly, we will define the embedding for the source side α of a rule $r : X \rightarrow \langle \alpha, \gamma \rangle$. Let $\mathcal{V}_S$ be the vocabulary in the source language with size $|\mathcal{V}_S|$; $\mathbb{R}^{n \times |\mathcal{V}_S|}$ be the word embedding matrix, each column of which is the word embedding (n-dimensional vector) for the corresponding word in $\mathcal{V}_S$; and $maxSource$ be the maximal length of α for all rules. We further assume that the α for all rules share the same length as $maxSource$; otherwise, we add $maxSource - |\alpha|$ words "$NULL$" to the end of α to obtain a new α. We define the embedding of α as the concatenation of the word embedding of each word in α. In particular, for the non-terminal in α, we define its word embedding as the vector whose components are 0.1; and we define the word embedding of "$NULL$" as $\mathbf{0}$. Then, we similarly define the embedding for the target side of a rule, given an embedding matrix for the target vocabulary. Finally, we define the embedding of a rule as the concatenation of the embedding of its source and target sides.

In this paper, we apply the word embedding matrices from the RNNLM toolkit (Mikolov et al., 2010) with the default settings: we train two RNN language models on the source and target sides of training corpus, respectively, and then we obtain two matrices as their by-products[1]. It would be potentially better to train the word embedding matrix from a much larger corpus as (Collobert and Weston, 2008), and we will leave this as a future task.

3.4 Decoding

Substituting the $\mathbf{P}(e, d|f; W)$ in Eq. (2) with $S(f, e, d; \theta)$ in Eq. (5), we can obtain its corre-

[1] In the RNNLM toolkit, the default dimension for word embedding is $n = 30$. In our experiments, the maximal length of α and γ are 5 and 12 respectively. Thus the dimension for h' is $K' = 30 \times (5 + 12) = 510$.

sponding decoding formula:

$$\langle \hat{e}, \hat{d} \rangle = \arg\max_{e,d} S(f, e, d; \theta).$$

Given the model parameter $\theta = (W, W', M, B)$, if we consider (M, B) as constant and $\sigma\big(M \cdot h'(r) + B\big)$ as an additional feature vector besides h, then Eq. (5) goes back to being a log-linear model with parameter (W, W'). In this way, the decoding for AdNN can share the same search strategy and cube pruning method as the log-linear model.

4 Training Method

4.1 Training Objective

For the log-linear model, there are various tuning methods, e.g. MERT (Och, 2003), MIRA (Watanabe et al., 2007; Chiang et al., 2008), PRO (Hopkins and May, 2011) and so on, which iteratively optimize a weight such that, after re-ranking a k-best list of a given development set with this weight, the loss of the resulting 1-best list is minimal. In the extreme, if the k-best list consists only of a pair of translations $\langle \langle e^*, d^* \rangle, \langle e', d' \rangle \rangle$, the desirable weight should satisfy the assertion: if the BLEU score of e^* is greater than that of e', then the model score of $\langle e^*, d^* \rangle$ with this weight will be also greater than that of $\langle e', d' \rangle$. In this paper, a pair $\langle e^*, e' \rangle$ for a source sentence f is called as a preference pair for f. Following PRO, we define the following objective function under the max-margin framework to optimize the AdNN model:

$$\frac{1}{2}\|\theta\|^2 + \frac{\lambda}{N} \sum_f \sum_{e^*, d^*, e', d'} \delta(f, e^*, d^*, e', d'; \theta), \quad (6)$$

with

$$\delta(\cdot) = \max\big\{S(f, e', d'; \theta) - S(f, e^*, d^*; \theta) + 1, 0\big\}$$

where f is a source sentence in a given development set, and $\langle \langle e^*, d^* \rangle, \langle e', d' \rangle \rangle$ is a preference pair for f; N is the number of all preference pairs; $\lambda > 0$ is a regularizer.

4.2 Optimization Algorithm

Since there are thousands of parameters in Eq. (6) and the tuning in SMT will minimize Eq. (6) repeatedly, efficient and scalable optimization methods are required. Following Le et al. (2011), we apply the mini-batch **C**onjugate **S**ub-**G**radient (mini-batch CSG) method to minimize Eq. (6). Compared with the sub-gradient descent, mini-batch CSG has some advantages: (1) it can accelerate the calculation of the sub-gradient since it calculates the sub-gradient on a subset of preference pairs (i.e. mini-batch) instead of all of the preference pairs; (2) it reduces the number of iterations since it employs the conjugate information besides the sub-gradient. Algorithm 1 shows the procedure to minimize Eq. (6).

Algorithm 1 Mini-batch conjugate subgradient

Input: θ^1, T, $CGIter$, $batch\text{-}size$, $k\text{-}best\text{-}list$

1: **for all** t such that $1 \leq t \leq T$ **do**
2: Sample mini-batch preference pairs with size $batch\text{-}size$ from $k\text{-}best\text{-}list$
3: Calculate some quantities for CG, e.g. training objective Obj, subgradient Δ, according to Eq. (6) defined over the sampled preference pairs
4: $\theta^{t+1} = CG(\theta^t, Obj, \Delta, CGIter)$
5: **end for**

Output: θ^{T+1}

In detail, line 2 in Algorithm 1 firstly follows PRO to sample a set of preference pairs from $k\text{-}best\text{-}list$, and then uniformly samples $batch\text{-}size$ pairs from the preference pair set. Line 3 calculates some quantities for CG, and Line 4 calls a CG optimizer [2] and obtains θ^{t+1}. At the end of the algorithm, it returns the result θ^{T+1}. In this work, we set the maximum number of CG iterations, $CGIter$, to a small number, which means θ^{t+1} will be returned within $CGIter$ iterations before the CG converges, for faster learning.

4.3 Pre-Training and Post-Training

Since Eq. (6) is non-linear, there are many local minimal solutions. Actually, this problem is inherent and is one many works based on the neural network for other NLP tasks such as language model and parsing, also suffer from. And these works empirically show that some pre-training methods, which provide a reasonable initial solution, can improve the performance. Observing the structure of Eq. (5) and the relationships between our model and a log-linear model, we propose the following simple pre-training method.

[2] In implementation, we call the CG toolkit (Hager and Zhang, 2006), which requires overloading objective and subgradient functions. For easier description, we substitute overloading functions and transform the value of functions in the pseudo-code.

If we set $W' = 0$, the model defined in Eq. (5) can be regarded as a log-linear model with features h. Therefore, we pre-train W using MERT or PRO by holding $W' = 0$, and use $(W, W' = 0, M, B)$ as an initializer[3] for Algorithm 1.

Although the above pre-training would provide a reasonable solution, Algorithm 1 may still fall into local minima. We also propose a post-training method: after obtaining a solution with Algorithm 1, we modify this solution slightly to get a new solution. The idea of the post-training method is similar to that of the pre-training method. Suppose $\theta = (W, W', M, B)$ be the solution obtained from Algorithm 1. If we consider both M and B to be constant, the Eq. (5) goes back to the log-linear model whose features are $(h, \sigma\big(M \cdot h' + B)\big)$ and parameters are (W, W'). Again, we train the parameters (W, W') with MERT or PRO and get the new parameters $(\bar{W}, \bar{W}')$. Therefore, we can set $\theta = (\bar{W}, \bar{W}', M, B)$ as the final solution for Eq. (6). The advantage of post-training is that it optimizes a convex programming derived from the original nonlinear (non-convex) programming in Eq. (6), and thus it may decrease the risk of poor local optima.

4.4 Training Algorithm

Algorithm 2 Training Algorithm

Input: $MaxIter$, a dev set, parameters (e.g. λ) for Algorithm 1

1: Pre-train to obtain $\theta_1 = (W, W' = 0, M, B)$ as the initial parameter
2: **for all** i such that $1 \leq i \leq MaxIter$ **do**
3: Decode with θ_i on the dev set and merge all k-best-lists
4: Run Algorithm 1 based on the merged k-best-list to obtain θ_{i+1}
5: **end for**
6: Post-train based on $\theta_{MaxIter+1}$ to obtain θ

Output: θ

The whole training for the AdNN model is summarized in Algorithm 2. Given a development set, we first run pre-training to obtain an initial parameter θ_1 for Algorithm 1 in line 1. Secondly, it iteratively performs decoding and optimization for $MaxIter$ times in the loop from line 2 to line 5: it decodes with the parameter θ_i and merges all the k-best-lists in line 3; and it then runs Algorithm 1 to optimize θ_{i+1}. Thirdly, it runs the post-training to get the result θ based on $\theta_{MaxIter+1}$.

Of course, we can run post-training after running Algorithm 1 at each iteration i. However, since each pass of post-training (e.g. PRO) takes several hours because of multiple decoding times, we run it only once, at the end of the iterations instead.

5 Experiments and Results

5.1 Experimental Setting

We conduct our experiments on the Chinese-to-English and Japanese-to-English translation tasks. For the Chinese-to-English task, the training data is the FBIS corpus (news domain) with about $240k$ sentence pairs; the development set is the NIST02 evaluation data; the development test set is NIST05; and the test datasets are NIST06, and NIST08. For the Japanese-to-English task, the training data with $300k$ sentence pairs is from the NTCIR-patent task (Fujii et al., 2010); the development set, development test set, and two test sets are averagely extracted from a given development set with 4000 sentences, and these four datasets are called test1, test2, test3 and test4, respectively.

We run GIZA++ (Och and Ney, 2000) on the training corpus in both directions (Koehn et al., 2003) to obtain the word alignment for each sentence pair. Using the SRILM Toolkits (Stolcke, 2002) with modified Kneser-Ney smoothing, we train a 4-gram language model for the Chinese-to-English task on the Xinhua portion of the English Gigaword corpus and a 4-gram language model for the Japanese-to-English task on the target side of its training data. In our experiments, the translation performances are measured by case-sensitive BLEU4 metric[4] (Papineni et al., 2002). The significance testing is performed by paired bootstrap re-sampling (Koehn, 2004b).

We use an in-house developed hierarchical phrase-based translation (Chiang, 2005) for our baseline system, which shares the similar setting as Hiero (Chiang, 2005), e.g. beam-size=100, k-best-size=100, and is denoted as **L-Hiero** to emphasize its log-linear model. We tune L-Hiero with two methods MERT and PRO implemented in the Moses toolkit. On the same experiment settings, the performance of L-Hiero is comparable

[3] To avoid the symmetry in the solution, we sample a very small (M, B) from the gaussian distribution in practice instead of setting $(M, B) = \mathbf{0}$.

[4] We use mteval-v13a.pl as the evaluation tool(Ref. http://www.itl.nist.gov/iad/mig/tests/mt/2008/scoring.html).

	Seconds/Sent
L-Hiero	1.77
AdNN-Hiero-E	1.88

Table 1: The decoding time comparison on NIST05 between L-Hiero and AdNN-Hiero-E.

to that of Moses: on the NIST05 test set, L-Hiero achieves 25.1 BLEU scores and Moses achieves 24.8. Further, we integrate the embedding features (See Section 3.3) into the log-linear model along with the default features as L-Hiero, which is called **L-Hiero-E**. Since L-Hiero-E has hundreds of features, we use PRO as its tuning toolkit.

AdNN-Hiero-E is our implementation of the AddNN model with embedding features, as discussed in Section 3, and it shares the same codebase and settings as L-Hiero. We adopt the following setting for training AdNN-Hiero-E: u=10; $batch\text{-}size$=1000 and $CGiter$=3, as referred in (Le et al., 2011), and T=200 in Algorithm 1; the pre-training and post-training methods as PRO; the regularizer λ in Eq. (6) as 10 and 30, and $MaxIter$ as 16 and 20 in Algorithm 2, for Chinese-to-English and Japanese-to-English tasks, respectively. Although there are several parameters in AdNN which may limit its practicability, according to many of our internal studies, most parameters are insensitive to AdNN except λ and $MaxIter$, which are common in other tuning toolkits such as MIRA and can be tuned[5] on a development test dataset.

Since both MERT and PRO tuning toolkits involve randomness in their implementations, all BLEU scores reported in the experiments are the average of five tuning runs, as suggested by Clark et al. (2011) for fairer comparisons. For AdNN, we report the averaged scores of five post-training runs, but both pre-training and training are performed only once.

5.2 Results and Analysis

As discussed in Section 3, our AdNN-Hiero-E shares the same decoding strategy and pruning method as L-Hiero. When compared with L-Hiero, decoding for AdNN-Hiero-E only needs additional computational times for the features in the hidden units, i.e. $\sigma\big(M \cdot h'(r) + B\big)$. Since these features are not dependent on the translation states, they are computed and saved to memory when loading the translation model. During decoding, we just look up these scores instead of re-calculating them on the fly. Therefore, the decoding efficiency of AdNN-Hiero-E is almost the same as that of L-Hiero. As shown in Table 1 the average decoding time for L-Hiero is 1.77 seconds/sentence while that for AdNN-Hiero-E is 1.88 seconds/sentence on the NIST05 test set.

[5]For easier tuning, we tuned these two parameters on a given development test set without post-training in Algorithm 2.

Chinese-to-English				
		NIST05	NIST06	NIST08
L-Hiero	MERT	25.10^{+}	24.46^{+}	17.42^{+}
	PRO	25.57^{+}	25.27^{+}	18.33^{+}
L-Hiero-E	PRO	24.80^{+}	24.46^{+}	18.20^{+}
AdNN-Hiero-E		26.37	25.93	19.42
Japanese-to-English				
		test2	test3	test4
L-Hiero	MERT	24.35^{+}	25.62^{+}	23.68^{+}
	PRO	24.38^{+}	25.55^{+}	23.66^{+}
L-Hiero-E	PRO	24.47^{+}	25.86^{+}	24.03^{+}
AdNN-Hiero-E		25.14	26.32	24.45

Table 2: The BLEU comparisons between AdNN-Hiero-E and Log-linear translation models on the Chinese-to-English and Japanese-to-English tasks. + means the comparison is significant over AdNN-Hiero-E with $p < 0.05$.

Word embedding features can improve the performance on other NLP tasks (Turian et al., 2010), but its effect on log-linear based SMT is not as expected. As shown in Table 2, L-Hiero-E gains little over L-Hiero for the Japanese-to-English task, and even decreases the performance over L-Hiero for the Chinese-to-English task. These results further prove our claim in Section 1, i.e. the log-linear model requires the features to be linear with the model and thus limits its expressive abilities. However, after the single-layer non-linear operator (sigmoid functions) on the embedding features for deep interpretation and representation, AdNN-Hiero-E gains improvements over both L-Hiero and L-Hiero-E, as depicted in Table 2. In detail, for the Chinese-to-English task, AdNN-Hiero-E improves more than 0.6 BLEU scores over L-Hiero on both test sets: the gains over L-Hiero tuned with PRO are 0.66 and 1.09 on NIST06 and NIST08, respectively, and the gains over L-Hiero tuned with MERT are even more. Similar results are achieved on the Japanese-to-English task. AdNN-Hiero-E gains about 0.7 BLEU scores on

Chinese-to-English			
	NIST05	NIST06	NIST08
L-Hiero	25.57^{+}	25.27^{+}	18.33^{+}
AdNN-Hiero-E	26.37	25.93	19.42
AdNN-Hiero-D	26.21	26.07	19.54
Japanese-to-English			
	test2	test3	test4
L-Hiero	24.38	25.55	23.66
AdNN-Hiero-E	25.14^{+}	26.32^{+}	24.45^{+}
AdNN-Hiero-D	24.42	25.46	23.73

Table 3: The effect of different feature setting on AdNN model. + means the comparison is significant over AdNN-Hiero-D with $p < 0.05$.

both test sets.

In addition, to investigate the effect of different feature settings on AdNN, we alternatively design another setting for h' in Eq. (5): we use the default features for both h' and h. In particular, the language model of a rule for h' is locally calculated without the contexts out of the rule as described in (Chiang, 2007). We call the AdNN model with this setting **AdNN-Hiero-D**[6]. Although there are serious overlaps between h and h' for AdNN-Hiero-D which may limit its generalization abilities, as shown in Table 3, it is still comparable to L-Hiero on the Japanese-to-English task, and significantly outperforms L-Hiero on the Chinese-to-English translation task. To investigate the reason why the gains for AdNN-Hiero-D on the two different translation tasks differ, we calculate the perplexities between the target side of training data and test datasets on both translation tasks. We find that the perplexity of the 4-gram language model for the Chinese-to-English task is 321.73, but that for the Japanese-to-English task is only 81.48. Based on these similarity statistics, we conjecture that the log-linear model does not fit well for difficult translation tasks (e.g. translation task on the news domain). The problem seems to be resolved by simply alternating feature representations through non-linear models, i.e. AddN-Hiero-D, even with single-layer networks.

6 Related Work

Neural networks have achieved widespread attentions in many NLP tasks, e.g. the language model (Bengio et al., 2003); POS, Chunking, NER, and SRL (Collobert and Weston, 2008); Parsing (Collobert and Weston, 2008; Socher et al., 2011); and Machine transliteration (Deselaers et al., 2009). Our work is, of course, highly motivated by these works. Unlike these works, we propose a variant neural network, i.e. additive neural networks, starting from SMT itself and taking both of the model definition and its inference (decoding) together into account.

Our variant of neural network, AdNN, is highly related to both additive models (Buja et al., 1989) and generalized additive neural networks (Potts, 1999; Waal and Toit, 2007), in which an additive term is either a linear model or a neural network. Unlike additive models and generalized additive neural networks, our model is decomposable with respect to translation rules rather than its component variables considering the decoding efficiency of machine translation; and it allows its additive terms of neural networks to share the same parameters for a compact structure to avoid sparsity.

The idea of the neural network in machine translation has already been pioneered in previous works. Castaño et al. (1997) introduced a neural network for example-based machine translation. In particular, Son et al. (2012) and Schwenk (2012) employed a neural network to model the phrase translation probability on the rule level $\langle \alpha, \gamma \rangle$ instead of the bilingual sentence level $\langle f, e \rangle$ as in Eq. (5), and thus they did not go beyond the log-linear model for SMT.

There are also works which exploit non-linear models in SMT. Duh and Kirchhoff (2008) proposed a boosting re-ranking algorithm using MERT as a week learner to improve the model's expressive abilities; Sokolov et al. (2012) similarly proposed a boosting re-ranking method from the ranking perspective rather than the classification perspective. Instead of considering the re-ranking task in SMT, Xiao et al. (2010) employed a boosting method for the system combination in SMT. Unlike their post-processing models (either a re-ranking or a system combination model) in SMT, we propose a non-linear translation model which can be easily incorporated into the existing SMT framework.

7 Conclusion and Future Work

In this paper, we go beyond the log-linear model for SMT and propose a novel AdNN based trans-

[6] All its parameters are shared with AdNN-Hiero-E except λ and $MaxIter$, which are tuned on the development test datasets.

lation model. Our model overcomes some of the shortcomings suffered by the log-linear model: linearity and the lack of deep interpretation and representation in features. One advantage of our model is that it jointly learns features and tunes the translation model and thus learns features towards the translation evaluation metric. Additionally, the decoding of our model is as efficient as that of the log-linear model. For Chinese-to-English and Japanese-to-English translation tasks, our model significantly outperforms the log-linear model, with the help of word embedding.

We plan to explore more work on the additive neural networks in the future. For example, we will train word embedding matrices for source and target languages from a larger corpus, and take into consideration the bilingual information, for instance, word alignment; the multi-layer neural network within the additive neural networks will be also investigated in addition to the single-layer neural network; and we will test our method on other translation tasks with larger training data as well.

Acknowledgments

We would like to thank our colleagues in both HIT and NICT for insightful discussions, Tomas Mikolov for the helpful discussion about the word embedding in RNNLM, and three anonymous reviewers for many invaluable comments and suggestions to improve our paper. This work is supported by National Natural Science Foundation of China (61173073, 61100093, 61073130, 61272384), the Key Project of the National High Technology Research and Development Program of China (2011AA01A207), and the Fundamental Research Funds for Central Universities (HIT.NSRIF.2013065).

References

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. 2003. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3:1137–1155, March.

Christopher M. Bishop. 1995. *Neural Networks for Pattern Recognition*. Oxford University Press, Inc., New York, NY, USA.

Peter F. Brown, Vincent J. Della Pietra, Stephen A. Della Pietra, and Robert L. Mercer. 1993. The mathematics of statistical machine translation: parameter estimation. *Comput. Linguist.*, 19:263–311, June.

Andreas Buja, Trevor Hastie, and Robert Tibshirani. 1989. Linear smoothers and additive models. *The Annals of Statistics*, 17:453–510.

M. Asuncin Castaño, Francisco Casacuberta, and Enrique Vidal. 1997. Machine translation using neural networks and finite-state models. In *TMI*, pages 160–167.

David Chiang, Yuval Marton, and Philip Resnik. 2008. Online large-margin training of syntactic and structural translation features. In *Proc. of EMNLP*. ACL.

David Chiang. 2005. A hierarchical phrase-based model for statistical machine translation. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, pages 263–270, Stroudsburg, PA, USA. Association for Computational Linguistics.

David Chiang. 2007. Hierarchical phrase-based translation. *Comput. Linguist.*, 33(2):201–228, June.

Jonathan H. Clark, Chris Dyer, Alon Lavie, and Noah A. Smith. 2011. Better hypothesis testing for statistical machine translation: controlling for optimizer instability. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers - Volume 2*, HLT '11, pages 176–181, Stroudsburg, PA, USA. Association for Computational Linguistics.

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, ICML*.

R. Collobert. 2011. Deep learning for efficient discriminative parsing. In *AISTATS*.

Thomas Deselaers, Saša Hasan, Oliver Bender, and Hermann Ney. 2009. A deep learning approach to machine transliteration. In *Proceedings of the Fourth Workshop on Statistical Machine Translation*, StatMT '09, pages 233–241, Stroudsburg, PA, USA. Association for Computational Linguistics.

Kevin Duh and Katrin Kirchhoff. 2008. Beyond log-linear models: Boosted minimum error rate training for n-best re-ranking. In *Proceedings of ACL-08: HLT, Short Papers*, pages 37–40, Columbus, Ohio, June. Association for Computational Linguistics.

Atsushi Fujii, Masao Utiyama, Mikio Yamamoto, and Takehito Utsuro. 2010. Overview of the patent translation task at the ntcir-8 workshop. In *In Proceedings of the 8th NTCIR Workshop Meeting on Evaluation of Information Access Technologies: Information Retrieval, Question Answering and Cross-lingual Information Access*, pages 293–302.

William W. Hager and Hongchao Zhang. 2006. Algorithm 851: Cg_descent, a conjugate gradient method with guaranteed descent. *ACM Trans. Math. Softw.*, 32(1):113–137, March.

Mark Hopkins and Jonathan May. 2011. Tuning as ranking. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1352–1362, Edinburgh, Scotland, UK., July. Association for Computational Linguistics.

Philipp Koehn, Franz Josef Och, and Daniel Marcu. 2003. Statistical phrase-based translation. In *Proc. of HLT-NAACL*. ACL.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondřej Bojar, Alexandra Constantin, and Evan Herbst. 2007. Moses: open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions*, ACL '07, pages 177–180, Stroudsburg, PA, USA. Association for Computational Linguistics.

Philipp Koehn. 2004a. Pharaoh: A beam search decoder for phrase-based statistical machine translation models. In *AMTA*.

Philipp Koehn. 2004b. Statistical significance tests for machine translation evaluation. In *Proc. of EMNLP*. ACL.

Quoc V. Le, Jiquan Ngiam, Adam Coates, Ahbik Lahiri, Bobby Prochnow, and Andrew Y. Ng. 2011. On optimization methods for deep learning. In *ICML*, pages 265–272.

Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernocký, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *INTERSPEECH*, pages 1045–1048.

Patrick Nguyen, Milind Mahajan, and Xiaodong He. 2007. Training non-parametric features for statistical machine translation. In *Proceedings of the Second Workshop on Statistical Machine Translation*, pages 72–79, Prague, Czech Republic, June. Association for Computational Linguistics.

Franz Josef Och and Hermann Ney. 2000. Improved statistical alignment models. In *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*, ACL '00, pages 440–447, Stroudsburg, PA, USA. Association for Computational Linguistics.

Franz Josef Och and Hermann Ney. 2002. Discriminative training and maximum entropy models for statistical machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, pages 295–302, Stroudsburg, PA, USA. Association for Computational Linguistics.

Franz Josef Och. 2003. Minimum error rate training in statistical machine translation. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 160–167, Sapporo, Japan, July. Association for Computational Linguistics.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA, July. Association for Computational Linguistics.

William J. E. Potts. 1999. Generalized additive neural networks. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '99, pages 194–200, New York, NY, USA. ACM.

Holger Schwenk. 2012. Continuous space translation models for phrase-based statistical machine translation. In *Proceedings of the 24th International Conference on Computational Linguistics*, COLING '12, Mumbai, India. Association for Computational Linguistics.

Richard Socher, Cliff Chiung-Yu Lin, Andrew Y. Ng, and Christopher D. Manning. 2011. Parsing Natural Scenes and Natural Language with Recursive Neural Networks. In *Proceedings of the 26th International Conference on Machine Learning (ICML)*.

A. Sokolov, G. Wisniewski, and F. Yvon. 2012. Non-linear n-best list reranking with few features. In *AMTA*, San Diego, USA.

Le Hai Son, Alexandre Allauzen, and François Yvon. 2012. Continuous space translation models with neural networks. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, NAACL HLT '12, pages 39–48, Stroudsburg, PA, USA. Association for Computational Linguistics.

Andreas Stolcke. 2002. Srilm - an extensible language modeling toolkit. In *Proc. of ICSLP*.

Joseph Turian, Lev Ratinov, and Yoshua Bengio. 2010. Word representations: a simple and general method for semi-supervised learning. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, ACL '10, pages 384–394, Stroudsburg, PA, USA. Association for Computational Linguistics.

D. A. de Waal and J. V. du Toit. 2007. Generalized additive models from a neural network perspective. In *Proceedings of the Seventh IEEE International Conference on Data Mining Workshops*, ICDMW '07, pages 265–270, Washington, DC, USA. IEEE Computer Society.

Taro Watanabe, Jun Suzuki, Hajime Tsukada, and Hideki Isozaki. 2007. Online large-margin training for statistical machine translation. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 764–773, Prague, Czech Republic, June. Association for Computational Linguistics.

Tong Xiao, Jingbo Zhu, Muhua Zhu, and Huizhen Wang. 2010. Boosting-based system combination for machine translation. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, ACL '10, pages 739–748, Stroudsburg, PA, USA. Association for Computational Linguistics.