AN AUTOENCODER WITH BILINGUAL SPARSE FEATURES FOR IMPROVED STATISTICAL MACHINE TRANSLATION

Bing Zhao , Yik-Cheung Tam, and Jing Zheng

firstname.lastname@sri.com; SRI International, Menlo Park, CA 94025, USA

ABSTRACT

Though sparse features have produced significant gains over traditional dense features in statistical machine translation, careful feature selection and feature engineering are necessary to avoid overfitting in optimizations. However, many sparse features are highly overlapping with each other; that is, they cover the same or similar information of translational equivalence from slightly different points of view, and eventually overfit easily with only very feature training samples in given bilingual stochastic context-free grammar (SCFG) rules. We propose a natural autoencoder that maps all the discrete and overlapping sparse features for each SCFG rule into a continuous vector, so that the information encoded in sparse feature vectors becomes a dense vector that may enjoy more samples during training and avoid overfitting. Our experiments showed that for a 33-million bilingual SCFG rules statistical machine translation system, the autoencoder generalizes much better than sparse features alone using the same optimization framework.

Index Terms— machine translation, sparse features, SCFG grammar induction, optimization, autoencoder, PRO

1. INTRODUCTION

Sparse features were designed to fix very specific machine translation errors and dramatically improve translation quality for these cases. For instance, a lexical error can be fixed by checking a particular word-pair and assigning to the error a weighted penalty when it appears again in the search path. The weight is generally tuned via MIRA [1] or PRO [2] with sentence-level BLEU score approximations. However, most sparse features are too specific, and the weights tuned on them tend to remember the whole tuning set: e.g., the bigram of the target side of the relative frequencies for bilingual stochastic context-free grammar (SCFG) rule, the rule ID, which can be remembered almost exactly for a given tuning set; or the word-pair, which is heavily biased for the given tuning set. Also, sparse features tend to be highly overlapping with each other; for instance, a fertility feature tends to be partially redundant with the lexical word pairs; a bigram feature has redundancy with unigrams, and so on. Majority of these features have only very limited samples observed in optimization, and thus can have very large and unreliable weight for unseen data. Within a log-linear model, tuning on these overlapping features is difficult, and easily gets easily caught in local optima. Overfitting is a curse of sparse features because the tuning set tends to provide too few examples to learn a meaningful weight to generalize to unseen test sets.

With their overlapping nature, sparse features actually evaluate the translational equivalence for each SCFG rule. We propose to shrink the dimension of the millions of sparse features into a reasonably small dimension of dense features so that each dimension can use more training samples, instead of being too specific to recall only examples from a given tuning set. An autoencoder with different configurations can automatically shrink the dimensionality of the overlapping sparse features and hence improve the trainability of these features. An autoencoder has a strong relationship to principal component analysis (PCA) if the inner layer of a three-layer net is linear. With its flexibility for more layers and non-linear hidden layers, an autoencoder provides much more information and flexibility than PCA.

The proposed autoencoder simplifies feature engineering for machine translation by feeding forward the existing highly overlapping, discrete features to a neural network, and generating continuous real-valued dense features. These resulting dense features can simplify optimization and feature selection, leading to better operating points as each feature dimension reflects more training samples. They also help reduce overfitting, and generalize better to unseen test sets.

In this paper, we treat each sparse feature as an entity and represent it as a node in a neural network. In Section 2, sparse features for machine translations are outlined and analyzed. Section 3 describes an autoencoder for encoding the sparse features and different training strategies. Section 4 details our experiments using the autoencoder. Section 5 presents discussion and conclusions.

2. SPARSE FEATURES FOR MACHINE TRANSLATION

Statistical machine translation decoding employs a log-linear framework for modeling the translational equivalence for translating foreign language F into English E as in Eqn. 1

$$E^* = \arg\max_E P(E|F) \sim \arg\max \vec{\lambda} \cdot \vec{ff}(F, E), \qquad (1)$$

where $\vec{ff}(F, E)$ is a vector of rich features defined for a given source sentence and target sentence pair: F, E. The weights denoted as λ are learnt from a tuning set, usually consisting of a couple of thousand sentence-pairs. Simple dense features for machine translation include the IBM Model-1 [3] lexical scores, relative frequencies for bilingual stochastic context-free grammar (SCFG) rules learnt from the parallel data, and monolingual language model scores. Optimization methods such as PRO or MIRA can optimize million of features.

Sparse features are designed to help check the specific evidences in the SCFG rules in statistical machine translation. These features

This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. HR0011-12-C-0016. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the DARPA.

978-1-4799-2893-4/14/$31.00 ©2014 IEEE

Table 1. Sparse Feature Types and Examples

Feat Categories	Information	Examples
Lexical	if the word-pair is seen in a lexicon	$f-e$
Fertility	Source word fertility	$f-v0$, $f-v1$, $f-v2$, $f-v3+$
Rule type	Detailed Hiero rule types	F-X1-F-X2 $\leftrightarrow$ X1-E-X2
Reorder type	If the target side contains monotone or reordering of non-terminals	WX0WX1W
Target spontaneous words	Predefined English spontaneous words	the, this, such, was...
Bigrams	Bigrams seen in the target side of the phrases	BI_explosions_murders
Frequency of rules	Bind frequency if the observed rules	freq1, ..., freqK

are usually discrete and most of the time have binary values. They overlap with other features such as dense features like IBM Model-1 or language model N-grams scores. In Table 1, the lexicon features are derived from the IBM model-1, in which we check if a word-pair $f \leftrightarrow e$ occurs in the derivation. The fertility features check the number of times a word is aligned to 1 word, 2 words, or 3+ words. The reordering features will only check the reordering between nonterminals to provide a simple count of the reorderings in the derivation tree. The rule type is a more detailed description of the SCFG rules used. 38 types of rules are defined in our system. Several examples together with the weights are given in Table 2. The larger the weights for the features, the more preferable by the machine translation systems. All the weights were learnt from a PRO optimization framework with linear SVM classifier using a L2 regularizer.

2.1. Discrete valued features

As can be seen, most of the sparse features used in machine translation are simple binary-valued ones. They do not make use of the Gaussian distributions found in speech recognition Markov state emissions, and hence lack the state labels (from a decision tree) required for merging the class labels back to the encoding scheme. Furthermore, the properties of the normalizations for the sparse features cannot be preserved by a smooth neural network.

2.2. Overlapping nature of the features

The majority of the features overlap with each other. For instance, the lexical word-pair features were derived directly from IBM Model-1, an already dense feature in many translation engines. The rule type and reordering type also overlap because both consider the source side of a rule. The sparse bigrams features also overlap with the standard language model, though sparse feature values are binary. Similar translation equivalences are represented in many features, suggesting that we can do feature dimensionality reduction to weave all the evidence together, such that the features of different dimensions are decoupled and transformed into real values to use more samples during optimization

3. AUTOENCODER

The goal of an autoencoder is to project an input vector x into a hidden representation h , which produces the input vector $\hat{x} = f(x;\Lambda)$ where $f(\cdot)$ is a predictor function parametrized by Λ [4]. For optimization, we use a minimum squared error between the reproduced vector and the original input vector, subject to a regularization term:

$$L(X;\Lambda) = \frac{1}{2}\sum_{i=1}^{N}|x_i - f(x_i;\Lambda)|^2 + \frac{C}{2}|\Lambda|^2 \quad (2)$$

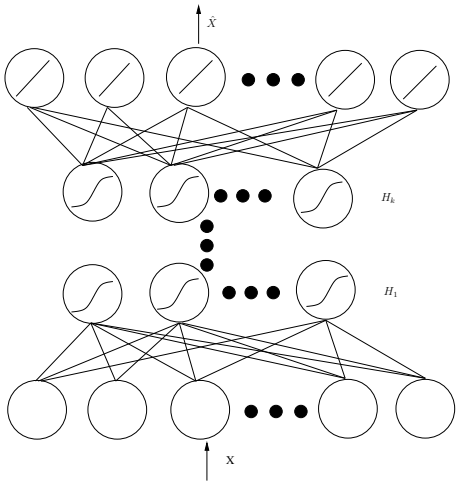

Fig. 1. Deep autoencoder architecture.

where C denotes the L2 regularization constant, which is usually tuned on a development set. The choice for the function predictor can have a multi-layer neural network architecture as shown in Figure 1. With this architecture, we can implement different types of autoencoder, including PCA when linear units are employed in the hidden layer of a 3-layer neural network. With the recent advances in deep learning, we use multiple hidden layers with sigmoidal units to implement the autoencoder. Since it is an autoencoder, linear output units are employed. Given training examples $\{x_i\}$, we feed x_i into the input and output layer of the neural network. Standard stochastic backpropagation is performed via gradient descent on a mini-batch of training examples.

$$\frac{\partial L(X;\Lambda)}{\partial \Lambda} = \sum_{i=1}^{M}(x_i - f(x_i;\Lambda)) \cdot \frac{\partial f(x_i;\Lambda)}{\partial \Lambda} + C\Lambda \quad (3)$$

where M denotes the size of the mini-batch. To prevent overfitting, cross-validation is carried out to ensure that each weight update will improve the performance on the cross validation set.

A simple autoencoder is a three-layer linear neural network, with the hidden layer as a linear one, and the same input and output dimensions. A cross validation set is used to check the learning progress on a sum-squared-error criterion; this check can be considered simple PCA.

3.1. Encoding sparse features

Previous work has focused on converting *words* via autoencoder into real-valued vector representation for ITG (inverse transduction

Table 2. Features and their weights. The positive weights are what MT system likes; negative weights are what system tries to penalize during decoding. X is a non-terminal, and W is referring to a word. F referring to a word in foreign language, and E a word in English. The order of $X1$ and $X0$ in a rule represents a reordering.

Features	Weights	Features	Weights	Features	Weights
WX0WX1W	0.0431	F-X-F-X-F→X-E-X-E	-0.0301	some	-0.0038
WX1WX0W	-0.0175	F-X-F-X → E-X-E-X-E	-0.0146	an	0.0087
X0WX1	-0.0346	X-F-X → X-E-X-E	0.0042	such	-0.0514
X1X0W	-0.0397	F-X-F-X-F → X-E-X	-0.0086	was	0.0128

grammar) for MT. Here we focused on a new type of autoencoder, where each note is considered a binary sparse feature (0 or 1), instead of a word as in previous work. The input dimension is in the range of several millions; as the value is binary, we do not need to have normalizations as used in typical speech setup.

4. EXPERIMENTS

Our translation engine is built on data from the DARPA Transtac program, a speech-to-speech translation program targeting tactical military communication. The source language is Iraqi Arabic, and target language is conversational English. We have $763K$ parallel sentence-pairs as training data and 6984 sentence-pairs for learning the weights for sparse features. We have one development set of 2862 sentences, and three unseen test sets. The development set has one reference, and all test sets have 4 references. Details are listed in Table 3.

Table 3. Features and their weights

Data	Sentences	Words
Train	763915	4884766
Tune	6985	64188
Dev	2862	29324
Tst1	560	7994
Tst2	579	9895
Tst3	689	8985

In our baseline machine translation engine, we incorporated 12 dense features for each SCFG rule after the Hiero grammar in [5], including: IBM Model-1 scores in both source-to-target and target-to-source directions, relative frequencies in both directions, count of phrases, count of Hiero rules, number of source content aligned to target spontaneous words, number of target spontaneous words aligned to source content words, three bined frequencies, and the number of unaligned source words. For sparse features, we computed the 7 categories of the sparse features as listed in Table 1. In total, we have $826,532$ sparse features (denoted as 800K hereafter).

We chose PRO training as our optimization framework, and sampled the pair-wise ranking data points from an n-best list with 5 BLEU points gap. Up to 5000 samples per sentence were drawn in order to learn a linear SVM classifier, with L2 norm regularizations. Weights were then interpolated back with the seed weight to form a new weight vector for the next optimization iteration. For the sparse feature baseline, as in the second row in Table 4, we also carried out additional feature selection via cross-validation on tuning data. Around 2300 features were left with a weight, whose l1 norm is larger than 1e-3 in our final system as shown in the 2nd row of Table 4. For the rest of the experimental setup that using dense features, we simply carried out the PRO optimization directly, with random restarts in the initializations.

4.1. Speeding up autoencoder

Because the input dimension is very large – in the range of a million nodes each in the input layer and output layer – the parameter space is quite large, and learning can be very slow. We decided to divide the tuning data into mini-batches and randomly sample from these mini-batch for each epoch. We then monitored the learnt parameters on a cross-validation set. It seems the neural network training is also sensitive to the size of the mini-batches, and it is almost task-specifics in our experiments. We chose a size of 1000 data points for each mini-batch empirically. This approach speeds up learning and gives reasonable model output, even though not all the training data maybe used. The biggest model required about 24 GB in memory, and about 1 week to process 3 epochs and complete the final model on a single CPU with a sampling through a total of 33 million data points where each sample was one bilingual SCFG rule.

4.2. Applying the autoencoder

In our experiments, we configured the autoencoder in several ways for sparse features. First, we simulated the simple PCA, a three-layer network, with a linear hidden layer. Second, we simulated the non-linear component analysis (NLCA) in a 5-layer network with a hidden middle layer sandwiched by two non-linear hidden layers. Third, we varied the configurations for the non-linear hidden layer together with a selection of hidden nodes for the hidden layer. This configuration seemed to produce the best configurations for improved translation quality.

After feeding forward the sparse features from each SCFG rule to get the hidden layers values, we derived the feature representations through the hidden layer. These derived features from auto encoder are like the dense features, and are added to the features for each rule. In our experimental setups, the only different factors are from the additional derived features appended to the sparse feature setup (baseline2). Empirically, we found the initialization for optimizations are very hard if we start from all zero weights; rather than that, we started from an optimized dense feature seed weightings, and then add sparse features, and then add the derived nnet features incrementally in PRO optimization framework . A slow learning rate of 0.0125 was chosen for updates to the inferred dense feature vector. For instance, we add 50 more dimensions to **sparse** in **n1**, and 100 to **sparse** setup as in **n6**. This strategy of optimizing features incrementally seems to work empirically well. Results are shown in Table 4, in which all the optimizations for autoencoder are based on the seed from **sparse** baseline2, which is in turn based on **dense** baseline1.

In baseline3, we added the derived dense features from an autoencoder directly to the baseline1; without pairing with the sparse features, this gives worse performances than baseline2. The derived feature set alone is not representative enough of the full information in sparse features yet, and need to be paired with original features.

Comparing **n1**, **n2**, the PCA setups, to the baseline **sparse** setups, the differences are not statistically significant. In a way, the

Table 4. Translation experiments using different autoencoder structures. 800K denotes the input/output sparse features; for the hidden layers, **L** means linear, and **NL** non-linear; NLCA stands for non-linear component analysis. PCA for principal component analysis.

network	structure	dev	tst1	tst2	tst3
dense (baseline1)	N/A	33.4	49.1	51.1	50.9
sparse (baseline2)	N/A	33.9	49.9	52.0	52.4
dense+ (baseline 3)	800Kx50Lx800K	33.9	49.1	51.4	51.6
n1 (PCA)	800Kx50Lx800K	33.9	49.8	52.1	52.4
n2 (PCA)	800Kx100Lx800K	34.0	49.8	52.1	52.4
n3 (NLCA)	800Kx2000Lx50NLx2000Lx800K	34.1	49.9	52.7	52.6
n4 (NLCA)	800Kx2000Lx100NLx2000Lx800K	34.1	50.0	52.9	52.6
n5 (autoencoder)	800Kx2kNLx50NLx2kNLx800K	34.2	50.2	53.1	52.8
n6 (autoencoder)	800Kx2kNLx100NLx2kNLx800K	34.3	50.2	53.2	52.8

additional features inferred from PCA setup do not seem to provide better clues for translations, mainly due to its linear mixing nature of the hidden layer. Our sparse setup (baseline2) already had integrated feature selection, and pruned the noisy ones via cross validations. On top of this, PCA with mainly linear layer does not seem to be very useful yet.

Comparing **n3**, **n4**, to the PCA setup, it clearly showed the usefulness of the non-linearity for mixing the features. However, the training time for learning the parameters are significantly slower than the PCA ones due to the parameter space. We empirically choose 2000 nodes in the 2nd layer and the layer to the last; as also,mentioned in the sparse feature setup, after careful feature selection and optimizations, the informative feature list after pruning is around 2300. due to long training epoch for learning such networks, we did not try other parameters. With 100 dimensions in the middle layer, it seems to be helpful for improving the translation a little bit more. Yet, the difference is only marginal for test set 2 (tst2) considering n4 and n3. However, the improved from baseline2 (sparse) for n3 and n4 are statistically significant.

Similar observations for improvements were observed for the set up of **n5** and **n6**, in which all the layers are sigmoid (non-linear). The difference between n5 and n6 are not significant, but the improvement over baseline2 is significant esp. for test set 2 (tst2), in which an improvement of almost 1.2 on BLEU. The companions over NLCA setups showed, again, the non-linarlity seems to be one of the key factor for encoding additional information beyond the sparse features.

Overall, comparing **n6** with the sparse feature baseline **sparse** in Table 4, we observed better generalizations to both development and unseen test sets. In all the autoencoder setup, such as the 100 dense features in n6, used significantly more samples per dimension during optimization, and thus can significantly mitigated the effect of overfitting as presented by the sparse feature setup.

As also in related work as in the easy-adaptation in [6], such autoencoder features can be considered as shrinking the weights for the overfitting sparse features as well. As the autoencoder features come from the mixing of the spars features, they statistically co-occur with these original features, and thus compete weights for the same samples for training these sparse features, with an effect of shrinking the weights esp. for these over fitting ones as they now introduce adverse samples which were otherwise not present for these over fitting sparse features.

4.3. Related work

There have been some related work in using autoencoder or DNN (deep neural network) for natural language processing; however, most of them focus on word embeddings, mapping a word into a fixed length real-valued vector. For instance, in continuous language modeling such as work in [7], [8], [9] proposed multilayer neural network for language modeling; more recently, works in [10] applies DNN on NLP tasks such as POS tagging and chunking. Works in [11] used autoencoder to build a classifier for identifying the reordering options in a ITG grammar for machine translation, in which again the words are the input nodes for the neural network. In [12], DNN was applied on word embedding to replace the emission probabilities for a HMM word aligner for machine translation. Other recent works also start with word embeddings like in [13] for Chinese word segmentations and POS tagging; [14] use a neural network for sentiment analysis; [15] use neural network to embed similar words in the context to encode better probabilities in the translation model. Extending the work in [16], and with auxiliary input layers in a RNN, work in [17] models the bilingual sentences in the form of translation model and language model jointly for improved translation results.

Different than these previous work, our work is embedding the sparse features directly, which is of much more challenging than the words as we have almost millions of input nodes easily, and more difficult as the sparse features are naturally mostly overlapping, and discrete-valued. We encode the sparse features further into derived dense-alike features for more training samples to learn meaningful weights; this improved on top of a strong baseline of using sparse features.

5. DISCUSSIONS AND CONCLUSIONS

In this work, we proposed an autoencoder to map high-dimensional sparse features onto lower dimensions with improved generalizations and translation quality relative to a strong baseline. The proposed method helps to automate feature engineering, optimization and feature selection in MT development and alleviates overfitting. Future work includes feeding the errors from the translation engine directly back into the learning of the neural network to ensure that parameters learnt from the sparse features are better targeted.

Another area for research is to investigate the speeding up strategy of the training. With millions of input nodes (in our case, 800K), the first layer part alone is beyond a typical GPU memory (6GB in most cases), and empirical engineering efforts have to be carried out to enable the first layer parameter computing to happen in a CPU. Even with our sampling strategy, we found the training is sensitive to the random sampling/shuffling of the data and the size of the mini batch, in each epoch. A better and faster training strategy for autoencoder for machine translation is also in our future work efforts.

6. REFERENCES

[1] David Chiang, Kevin Knight, and Wei Wang, "11,001 new features for statistical machine translation," in *Proc. NAACL–HLT 2009*, 2009, pp. 218–226.

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

[3] Peter F. Brown, Stephen A. Della Pietra, Vincent. J. Della Pietra, and Robert L. Mercer, "The mathematics of statistical machine translation: Parameter estimation," in *Computational Linguistics*, 1993, vol. 19(2), pp. 263–331.

[4] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*, John Wiley and Sons, Inc., second edition, 2001.

[5] David Chiang, "Hierarchical phrase-based translation," in *Computational Linguistics*, 2007, vol. 33(2).

[6] Hal Daume III, "Frustratingly easy domain adaptation," in *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, Prague, Czech Republic, June 2007, pp. 256–263, Association for Computational Linguistics.

[7] Ahmad Emami and Frederick Jelinek, "A neural syntactic language model," in *Machine Learning Research*, 2005, pp. 60(1–3):195–227.

[8] Holger Schwenk, Daniel Dechelotte, and Jean-Luc Gauvain, "Continuous space language models for statistical machine translation," in *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*, Sydney, Australia, July 2006, pp. 723–730, Association for Computational Linguistics.

[9] Holger Schwenk, Anthony Rousseau, and Mohammed Attik, "Large, pruned or continuous space language models on a gpu for statistical machine translation," in *Proceedings of the NAACL-HLT 2012 Workshop: Will We Ever Really Replace the N-gram Model? On the Future of Language Modeling for HLT*, Montréal, Canada, June 2012, pp. 11–19, Association for Computational Linguistics.

[10] Ronan Collobert, Jason Weston, Leon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa, "Natural language processing from alms scratch," in *Journal of Machine Learning Research*, 2011, pp. 12:2493–2537.

[11] Peng Li, Yang Liu, and Maosong Sun, "Recursive autoencoders for ITG-based translation," in *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, Seattle, Washington, USA, October 2013, pp. 567–577, Association for Computational Linguistics.

[12] Nan Yang, Shujie Liu, Mu Li, Ming Zhou, and Nenghai Yu, "Word alignment modeling with context dependent deep neural network," in *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Sofia, Bulgaria, August 2013, pp. 166–175, Association for Computational Linguistics.

[13] Xiaoqing Zheng, Hanyang Chen, and Tianyu Xu, "Deep learning for Chinese word segmentation and POS tagging," in *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, Seattle, Washington, USA, October 2013, pp. 647–657, Association for Computational Linguistics.

[14] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts, "Recursive deep models for semantic compositionality over a sentiment treebank," in *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, Seattle, Washington, USA, October 2013, pp. 1631–1642, Association for Computational Linguistics.

[15] Hai-Son Le, Alexandre Allauzen, and François Yvon, "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*, Montréal, Canada, June 2012, pp. 39–48, Association for Computational Linguistics.

[16] Thomas Milolov, Stefan Kombrink, Lukas Burget, Jan Cernocky, and Sanjeev Khudanpur, "Extensions of recurrent neural network language model," in *ICASSP*, 2011, p. 55285531.

[17] Michael Auli, Michel Galley, Chris Quirk, and Geoffrey Zweig, "Joint language and translation modeling with recurrent neural networks," in *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, Seattle, Washington, USA, October 2013, pp. 1044–1054, Association for Computational Linguistics.