Effect of Non-linear Deep Architecture in Sequence Labeling

Mengqiu Wang
Computer Science Department
Stanford University
Stanford, CA 94305, USA
mengqiu@cs.stanford.edu

Christopher D. Manning
Computer Science Department
Stanford University
Stanford, CA 94305, USA
manning@cs.stanford.edu

Abstract

If we compare the widely used Conditional Random Fields (CRF) with newly proposed "deep architecture" sequence models (Collobert et al., 2011), there are two things changing: from linear architecture to non-linear, and from discrete feature representation to distributional. It is unclear, however, what utility non-linearity offers in conventional feature-based models. In this study, we show the close connection between CRF and "sequence model" neural nets, and present an empirical investigation to compare their performance on two sequence labeling tasks – Named Entity Recognition and Syntactic Chunking. Our results suggest that non-linear models are highly effective in low-dimensional distributional spaces. Somewhat surprisingly, we find that a non-linear architecture offers no benefits in a high-dimensional discrete feature space.

1 Introduction

Sequence labeling encompasses an important class of NLP problems that aim at annotating natural language texts with various syntactic and semantic information, such as part-of-speech tags and named-entity labels. Output from such systems can facilitate downstream applications such as Question Answering and Relation Extraction. Most methods developed so far for sequence labeling employ generalized linear statistical models, meaning methods that describe the data as a combination of linear basis functions, either directly in the input variables space (e.g., SVM) or through some transformation of the probability distributions (e.g., "log-linear" models).

Recently, Collobert et al. (2011) proposed "deep architecture" models for sequence labeling (named Sentence-level Likelihood Neural Nets, abbreviated as SLNN henceforth), and showed promising results on a range of tasks (POS tagging, NER, Chunking, and SRL). Two new changes were suggested: extending the model from a linear to non-linear architecture; and replacing discrete feature representations with distributional feature representations in a continuous space. It has generally been argued that non-linearity between layers is vital to the power of neural models (Bengio, 2009). The relative contribution of these changes, however, is unclear, as is the question of whether gains can be made by introducing non-linearity to conventional feature-based models.

In this paper, we illustrate the close relationship between CRF and SLNN models, and conduct an empirical investigation of the effect of nonlinearity with different feature representations. Experiments on Named Entity Recognition (NER) and Syntactic Chunking tasks suggest that non-linear models are highly effective in low-dimensional distributed feature space, but offer no benefits in high-dimensional discrete space. Furthermore, both linear and non-linear models improve when we combine the discrete and continuous feature spaces, but a linear model still outperforms the non-linear one.

2 From CRFs To SLNNs

A CRF models the conditional probability of structured output variables $\mathbf{y}$ given observations $\mathbf{x}$. In sequence modeling, the observations are typically words in a sentence, and the output variables are some syntactic or semantic tags we are trying to predict for each word (e.g., POS, named-entity tags, etc.). The most commonly used CRF model has a linear chain structure, where prediction y_i

at position i is independent of other predictions, given its neighbors y_{i-1} and y_{i+1}. It is customary to describe the model as an undirected graphical model, with the following probability definition:

$$P(\mathbf{y}|\mathbf{x}) = \frac{\prod_{i=1}^{|\mathbf{x}|} \Psi(\mathbf{x}, y_i; \Theta) \prod_{j=1}^{|\mathbf{x}|} \Phi(\mathbf{x}, y_j, y_{j-1}; \Lambda)}{Z(\mathbf{x})}$$

$$\Psi(\mathbf{x}, y_i; \Theta) = \exp\left\{\sum_{k=1}^{m} \theta_{(k,y_i)} f_k(\mathbf{x})\right\}$$

$$\Phi(\mathbf{x}, y_i, y_{i-1}; \Lambda) = \exp\left\{\sum_{k=1}^{m'} \lambda_{(k,y_i,y_{i-1})} g_k(\mathbf{x})\right\}$$

$$Z(\mathbf{x}) = \sum_{\mathbf{y}'} \left(\prod_{i=1}^{|\mathbf{x}|} \Psi(\mathbf{x}, y'_i) \prod_{j=1}^{|\mathbf{x}|} \Phi(\mathbf{x}, y'_j, y'_{j-1}) \right)$$

$\Psi(\mathbf{x}, y_i)$ denotes node clique potentials in this graph, and $\Phi(\mathbf{x}, y_i, y_{i-1})$ denotes edge clique potentials. $f_k(\mathbf{x})$ is the set of node-level feature functions, m is the number of node features, and $\theta_{(k,y_i)}$ is a weight parameter of feature k associated with a particular output y_i; similarly for edges we have $g_k(\mathbf{x})$, m', and $\lambda_{(k,y_i,y_{i-1})}$. $Z(\mathbf{x})$ is the partition function that sums over all possible assignments of output variables in the entire sequence.

Let us focus our discussion on the node clique potentials Ψ for now. We call the operand of the exponentiation operator in Ψ a *potential function* ψ. In a CRF, this can be expressed in matrix notation as:

$$\psi(\mathbf{x}, y_i; \Theta) = |\Theta^\intercal \mathbf{f}(\mathbf{x})|_{\widehat{y_i}1}$$

We use the notation $\widehat{y_i}$ to denote the ordinal index of the value assigned to y_i. This linear potential function ψ can be visualized using a neural network diagram, shown in the left plot in Figure 1. Each edge in the graph represents a parameter weight $\theta_{(k,\widehat{y_i})}$, for feature $f_k(\mathbf{x})$ and a variable assignment of y_i. In neural network terminology, this architecture is called a single-layer Input-Output Neural Network (IONN). [1] Normalizing locally in a logistic regression is equivalent to adding a *softmax* layer to the output layer of the IONN, which was commonly done in neural networks, such as in Collobert et al. (2011).

We can add a hidden linear layer to this architecture to formulate a two-layer Linear Neural Network (LNN), as shown in the middle diagram of Figure 1. The value of the node z_j in the hidden layer is computed as $z_j = \sum_k \omega_{(k,j)} f_k(\mathbf{x})$. The value y_i for nodes in the output layer is computed as: $y_i = \sum_j \delta_{(j,i)} z_j = \sum_j \delta_{(j,i)} \sum_k \omega_{(k,j)} f_k(\mathbf{x})$. where $\omega_{(k,j)}$ and $\delta_{(j,i)}$ are new parameters introduced in the model. In matrix form, it can be written as $\mathbf{y} = \Delta^\intercal \mathbf{z} = \Delta^\intercal \Omega^\intercal \mathbf{f}(\mathbf{x})$. The node potential function now becomes:

$$\psi'(\mathbf{x}, y_i; \Omega, \Delta) = |\Delta^\intercal \Omega^\intercal \mathbf{f}(\mathbf{x})|_{\widehat{y_i}1}$$

This two-layer network is actually no more powerful than the previous model, since we can always compile it down to a single-layer IONN by making $\Theta = \Omega\Delta$. In the next step, we take the output of the hidden layer in the LNN, and send it through a non-linear activation function, such as a *sigmoid* or *tanh*, then we arrive at a two-layer Deep Neural Network (DNN) model. Unlike the previous two models, the DNN is non-linear, and thus capable of representing a more complex decision surface.

So far we have extended the potential function used in node cliques of a CRF to a non-linear DNN. And if we keep the potential function for edge cliques the same as before, then in fact we have arrived at an identical model to the SLNN in Collobert et al. (Collobert et al., 2011). The difference between a SLNN and an ordinary DNN model is that we need to take into consideration the influence of edge cliques, and therefore we can no longer normalize the clique factors at each position to calculate the local marginals, as we would do in a logistic regression. The cardinality of the output variable vector $\mathbf{y}$ grows exponentially with respect to input sequence length. Fortunately, we can use *forward-backward* style dynamic programming to compute the marginal probabilities efficiently.

It is also worth pointing out that this model has in fact been introduced a few times in prior literature. It was termed *Conditional Neural Fields* by Peng et al. (2009), and later *Neural Conditional Random Fields* by Do and Artieres (2010). Unfortunately, the connection to Collobert and Weston (2008) was not recognized in either of these two studies; vice versa, neither of the above were referenced in Collobert et al. (2011). This model also appeared previously in the speech recognition literature in Prabhavalkar and Fosler-Lussier (2010).

[1] The bias parameter "*b*" commonly seen in Neural Network convention can be encoded as an "always on" feature in the input layer.

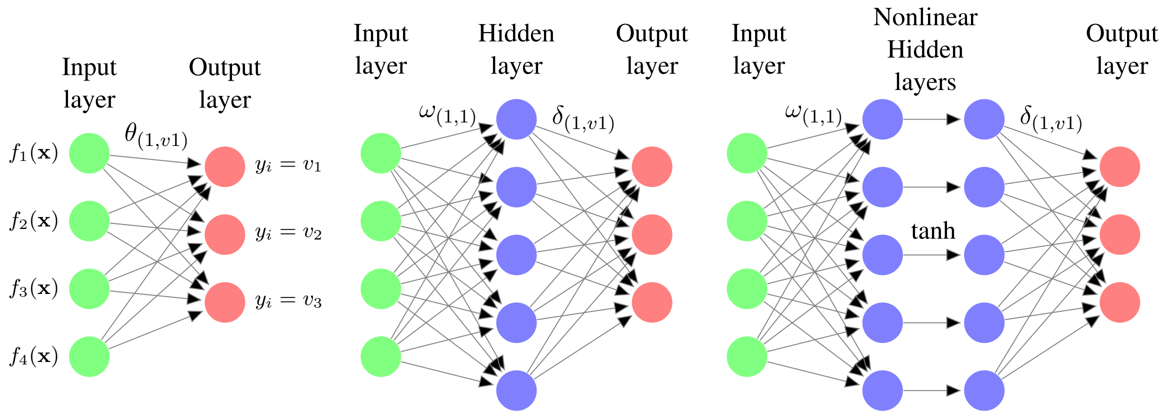

Figure 1: In this diagram, we assume the random variable y_i has three possible value assignments (v_1, v_2, v_3). On the left side is the linear potential function ψ in CRF, illustrated as a single-layer Input-Output Neural Network. In the middle is a potential function as a two-layer Linear Neural Network; on the right side is a two-layer Deep Neural Network.

3 Parameter Learning

Supervised conditional training of the SLNN model amounts to maximizing the objective function $\mathcal{L}$, which is given by the sum of log-probabilities over training examples:

$$\mathcal{L}(\mathbf{Y}^*|\mathbf{X}) = \sum_{l=1}^{|\mathbf{X}|} \Big(\sum_{i=1}^{|\mathbf{x}^l|} \psi'(\mathbf{x}^l, \mathbf{y}^{l*}_i) + \sum_{j=1}^{|\mathbf{x}^l|} \phi(\mathbf{x}^l, \mathbf{y}^{l*}_j, \mathbf{y}^{l*}_{j-1}) \Big) - \sum_{l=1}^{|\mathbf{X}|} \log Z(\mathbf{x})$$

The change in node potential function from ψ to ψ' does not affect the inference procedure, and thus we can employ the same dynamic programming algorithm as in a CRF to calculate the log sum over $Z(\mathbf{x})$ and the expectation of feature parameters.

We adopted the simple L-BFGS algorithm for training weights in this model (Liu and Nocedal, 1989). Although L-BFGS is in general slower than mini-batch SGD – another common optimization algorithm used to train neural networks (Bengio et al., 2006, *inter alia*), it has been found to be quite stable and suitable for learning neural networks (Socher et al., 2011). The gradient of a parameter $\omega_{(k,j)}$ is calculated as the following:

$$\frac{\partial \mathcal{L}}{\partial \omega_{(k,j)}} = \sum_{l=1}^{|\mathbf{X}|} \sum_{i=1}^{|\mathbf{x}_l|} \left(\frac{\partial \psi'(\mathbf{x}^l, \mathbf{y}^l{}_i)}{\partial \omega_{(k,j)}} - \mathbb{E}_{P(\mathbf{y}^l|\mathbf{x}^l)} \left[\frac{\partial \psi'(\mathbf{x}^l, \mathbf{y}^l{}_i)}{\partial \omega_{(k,j)}} \right] \right)$$

The partial derivative of the potential function $\frac{\partial \psi'(\mathbf{x}^l, \mathbf{y}^l{}_i)}{\partial \omega_{(k,j)}}$ can be calculated using the back-propagation procedure, identical to how gradients of a standard Multilayer Perceptron are calculated. The gradient calculation for output layer parameters Δ and edge parameters Λ follow the same form. We apply ℓ_2-regularization to prevent over-fitting.

4 Empirical Evaluation

We evaluate the CRF and SLNN models on two standard sequence labeling tasks: Syntactic Chunking and Named Entity Recognition (NER). In both experiments, we use the publicly available Stanford CRF Toolkit (Finkel et al., 2005).

4.1 Named Entity Recognition

We train all models on the standard CoNLL-2003 shared task benchmark dataset (Sang and Meulder, 2003), which is a collection of documents from Reuters newswire articles, annotated with four entity types: *Person*, *Location*, *Organization*, and *Miscellaneous*. We adopt the `BIO2` annotation standard. Beginning and intermediate positions of an entity are marked with *B-* and *I-* tags, and non-entities with *O* tag. The training set contains 204K words (14K sentences), the development set contains 51K words (3.3K sentences), and the test set contains 46K words (3.5K sentences).

To evaluate out-of-domain performance, we run the models trained on CoNLL-03 training data on two additional test datasets. The first dataset

(ACE) is taken from the ACE Phase 2 (2001-02) and ACE-2003 data. Although the ACE dataset also consists of newswire text and thus is not strictly out-of-domain, there is a genre or dialect difference in that it is drawn from mostly American news sources, whereas CoNLL is mostly English. The test portion of this dataset contains 63K words, and is annotated with 5 original entity types: *Person*, *Location*, *Organization*, *Fact*, and *GPE*. We remove all entities of type *Fact* and *GPE* by relabeling them as *O* during preprocessing, and discard entities tags of type *Miscellaneous* in the output of the models. The second dataset is the MUC7 Formal Run test set, which contains 59K words. It is also missing the *Miscellaneous* entity type, but includes 4 additional entity types that do not occur in CoNLL-2003: *Date*, *Time*, *Money*, and *Percent*. We converted the data to CoNLL-2003 type format using the same method applied to the ACE data.

We used a comprehensive set of features that comes with the standard distribution of Stanford NER model (Finkel et al., 2005). A total number of 437,905 features were generated for the CoNLL-2003 training dataset.

4.2 Syntactic Chunking

In Syntactic Chunking, we tag each word with its phrase type. For example, tag *B-NP* indicates a word starts a noun phrase, and *I-PP* marks an intermediate word of a prepositional phrase. We test the models on the standard CoNLL-2000 shared task evaluation set (Sang and Buchholz, 2000). This dataset comes from the Penn Treebank. The training set contains 211K words (8.9K sentences), and the test set contains 47K words (2K sentences). The set of features used for this task is:

- Current word and tag
- Word pairs: $w_i \wedge w_{i+1}$ for $i \in \{-1, 0\}$
- Tags: $(t_i \wedge t_{i+1})$ for $i \in -1, 0$; (t_{-1}, t_0, t_{i+1});
- The Disjunctive word set of the previous and next 4 positions

A total number of 317794 features were generated on this dataset.

4.3 Experimental Setup

In all experiments, we used the development portion of the CoNLL-2003 data to tune the ℓ_2-regularization parameter σ (variance in Gaussian prior), and found 20 to be a stable value. Overall tuning σ does not affect the qualitative results in our experiments. We terminate L-BFGS training when the average improvement is less than 1e-3. All model parameters are initialized to a random value in $[-0.1, 0.1]$ in order to break symmetry. We did not explicitly tune the features used in CRF to optimize for performance, since feature engineering is not the focus of this study. However, overall we found that the feature set we used is competitive with CRF results from earlier literature (Turian et al., 2010; Collobert et al., 2011). For models that embed hidden layers, we set the number of hidden nodes to 300. [2] Results are reported on the standard evaluation metrics of entity/chunk precision, recall and F1 measure.

For experiments with continuous space feature representations (a.k.a., word embeddings), we took the word embeddings (130K words, 50 dimensions) used in Collobert et al. (2011), which were trained for 2 months over Wikipedia text. [3] All sequences of numbers are replaced with `num` (e.g., "PS1" would become "PS`num`"), sentence boundaries are padded with token `PAD`, and unknown words are grouped into `UNKNOWN`. We attempt to replicate the model described in Collobert et al. (2011) without task-specific fine-tuning, with a few exceptions: 1) we used the *soft tanh* activation function instead of *hard tanh*; 2) we use the `BIO2` tagging scheme instead of `BIOES`; 3) we use L-BFGS optimization algorithm instead of stochastic gradient descent; 4) we did not use Gazetteer features; 5) Collobert et al. (2011) mentioned 5 binary features that look at the capitalization pattern of words to append to the embedding as additional dimensions, but only 4 were described in the paper, which we implemented accordingly.

5 Results and Discussion

For both the CRF and SLNN models, we experiment with both the discrete binary valued feature representation used in a regular CRF, and the word embeddings described. Unless otherwise stated, the set of edge features is limited to pairs of predicted labels at the current and previous positions, i.e., (y_i, y_{i-1}). The same edge features were used in Collobert et al. (2011) and were called "transition scores" ($[A]_{i,j}$).

[2]We tried varying the number of hidden units in the range from 50 to 500, and the main qualitative results remain the same.

[3]Available at `http://ml.nec-labs.com/senna/`.

	CRF			SLNN		
	P	R	F1	P	R	F1
CoNLL$_d$	90.9	90.4	**90.7**	89.3	89.7	89.5
CoNLL$_t$	85.4	84.7	**85.0**	83.3	83.9	83.6
ACE	81.0	74.2	**77.4**	80.9	74.0	77.3
MUC	72.5	74.5	**73.5**	71.1	74.1	72.6
Chunk	93.7	93.5	**93.6**	93.3	93.3	93.3

Table 1: Results of CRF versus SLNN, over discrete feature space. CoNLL$_d$ stands for the CoNLL development set, and CoNLL$_t$ is the test set. Best F1 score on each dataset is highlighted in bold.

	CRF			LLN		
	P	R	F1	P	R	F1
CoNLL$_d$	90.9	90.4	**90.7**	89.5	90.6	90.0
CoNLL$_t$	85.4	84.7	**85.0**	83.1	84.7	83.9
ACE	81.0	74.2	**77.4**	80.7	74.3	77.3
MUC	72.5	74.5	73.5	72.3	75.2	**73.7**
Chunk	93.7	93.5	**93.6**	93.1	93.2	93.2

Table 2: Results of CRF versus LNN, over discrete feature space.

5.1 Results of Discrete Representation

The first question we address is the following: in the high-dimensional discrete feature space, would the non-linear architecture in SLNN model help it to outperform CRF?

Results from Table 1 suggest that SLNN does not seem to benefit from the non-linear architecture on either the NER or Syntactic Chunking tasks. In particular, on the CoNLL and MUC dataset, SLNN resulted in a 1% performance drop, which is significant for NER. The specific statistical properties of this dataset that lead to the performance drop are hard to determine, but we believe it is partially because the SLNN has a much harder non-convex optimization problem to solve – on this small dataset, the SLNN with 300 hidden units generates a shocking number of 100 million parameters (437905 features times 300 hidden dimensions), due to the high dimensionality of the input feature space.

To further illustrate this point, we also compared the CRF model with its Linear Neural Network (LNN) extension, which has exactly the same number of parameters as the SLNN but does not include the non-linear activation layer. Although this model is identical in representational power to the CRF as we discussed in Section 2, the optimization problem here is no longer convex (Ando and Zhang, 2005). To see why, consider applying a linear scaling transformation to the input layer parameter matrix Ω, and apply the inverse scaling to output layer Δ matrix. The resulting model has exactly the same function values. We can see from Table 2 that there is indeed a performance drop with the LNN model as well, likely due to difficulty with optimization. By comparing the results of LNN and SLNN, we see that the addition of a non-linear activation layer in SLNN does not seem to help, but in fact further decreases performance in all cases except Syntactic Chunking.

A distinct characteristic of NLP data is its high dimensionality. The vocabulary size of a decent sized text corpus is already in the tens of thousands, and bigram statistics are usually an order of magnitude larger. These basic information units are typically very informative, and there is not much structure in them to be explored. Although some studies argue that non-linear neural nets suffer less from the curse of dimensionality (Attali and Pagés, 1997; Bengio and Bengio, 2000; Pitkow, 2012), counter arguments have been offered (Camastra, 2003; Verleysen et al., 2003). The empirical results from our experiment seems to support the latter. Similar results have also been found in other NLP applications such as Text Classification. Joachims concluded in his seminal work: "non-linear SVMs do not provide any advantage for text classification using the standard kernels" (Joachims, 2004, p. 115). If we compare the learning curve of CRF and SLNN (Figure 2), where we vary the amount of binary features available in the model by random sub-sampling, we can further observe that SLNNs enjoy a small performance advantage in lower dimensional space (when less than 30% of features are used), but are quickly outpaced by CRFs in higher dimensional space as more features become available.

Another point of consideration is whether there is actually much non-linearity to be captured in sequence labeling. While in some NLP applications like grammar induction and semantic parsing, the data is complex and rich in statistical structures, the structure of data in sequence labeling is considerably simpler. This contrast is more salient if we compare with data in Computer Vision tasks such as object recognition and image segmentation. The interactions among local variables there are much stronger and more likely to be non-linear. Lastly, models like CRF actually already capture some of the non-linearity in the

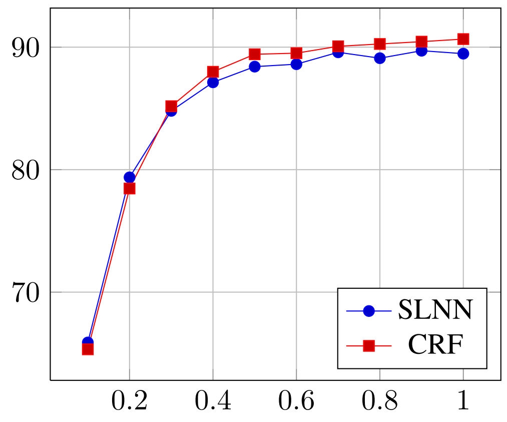

Figure 2: The learning curve of SLNN vs. CRF on CoNLL-03 dev set, with respect to the percentage of discrete features used (i.e., size of input dimension). Y-axis is the F1 score (out of 100), and X-axis is the percentage of features used.

	CRF			SLNN		
	P	R	F1	P	R	F1
CoNLL$_d$	80.7	78.7	79.7	86.1	87.1	**86.6**
CoNLL$_t$	76.4	75.5	76.0	79.8	81.7	**80.7**
ACE	71.5	71.1	71.3	75.8	74.1	**75.0**
MUC	65.3	74.0	69.4	65.7	76.8	**70.8**

Table 3: Results of CRF versus SLNN, over continuous space feature representations.

input space through the interactions of latent variables (Liang et al., 2008), and it is unclear how much additional gain we would get by explicitly modeling the non-linearity in local inputs.

5.2 Results of Distributional Representation

For the next experiment, we replace the discrete input features with a continuous space representation by looking up the embedding of each word, and concatenate the embeddings of a five word window centered around the current position. Four binary features are also appended to each word embedding to capture capitalization patterns, as described in Collobert et al. (2011). Results of the CRF and SLNN under this setting for the NER task is show in Table 3.

With a continuous space representation, the SLNN model works significantly better than a CRF, by as much as 7% on the CoNLL development set, and 3.7% on ACE dataset. This suggests that there exist statistical dependencies within this low-dimensional (300) data that cannot be effectively captured by linear transformations, but can be modeled in the non-linear neural nets. This perhaps coincides with the large performance improvements observed from neural nets in handwritten digit recognition datasets as well (Peng et al., 2009; Do and Artieres, 2010), where dimensionality is also relatively low.

	CoNLL$_d$	CoNLL$_t$	ACE	MUC
CRF$_{\text{discrete}}$	90.7	85.0	77.4	73.5
CRF$_{\text{join}}$	92.4	87.7	82.2	81.1
SLNN$_{\text{continuous}}$	86.6	80.7	75.0	70.8
SLNN$_{\text{join}}$	91.9	87.1	81.2	79.7

Table 4: Results of CRF and SLNN when word embeddings are appended to the discrete features. Numbers shown are F1 scores.

5.3 Combine Discrete and Distributional Features

When we join word embeddings with discrete features, we see further performance improvements, especially in the out-of-domain datasets. The results are shown in Table 4.

A similar effect was also observed in Turian et al. (2010). The performance of both the CRF and SLNN increases by similar relative amounts, but the CRF model maintains a lead in overall absolute performance.

6 Conclusion

We carefully compared and analyzed the non-linear neural networks used in Collobert et al. (2011) and the widely adopted CRF, and revealed their close relationship. Through extensive experiments on NER and Syntactic Chunking, we have shown that non-linear architectures are effective in low dimensional continuous input spaces, but that they are not better suited for conventional high-dimensional discrete input spaces. Furthermore, both linear and non-linear models benefit greatly from the combination of continuous and discrete features, especially for out-of-domain datasets. This finding confirms earlier results that distributional representations can be used to achieve better generalization.

Acknowledgments

The authors would like to thank Rob Voigt, Sida Wang, and the three anonymous reviewers, and acknowledge the support of the DARPA Broad Operational Language Translation (BOLT) program through IBM. Any opinions, findings, and conclusion or recommendations expressed in this material are those of the authors and do not necessarily reflect the view of DARPA or the US government.

References

Rie K. Ando and Tong Zhang. 2005. A framework for learning predictive structures from multiple tasks and unlabeled data. *JMLR*, 6:1817–1853.

Jean-Gabriel Attali and Gilles Pagés. 1997. Approximations of functions by a multilayer perceptron: a new approach. *Neural Networks*, 10:1069–1081.

Yoshua Bengio and Samy Bengio. 2000. Modeling high-dimensional discrete data with multi-layer neural networks. In *Proceedings of NIPS 12*.

Yoshua Bengio. 2009. Learning deep architectures for AI. *Found. Trends Mach. Learn.*, 2(1):1–127, January.

Francesco Camastra. 2003. Data dimensionality estimation methods: A survey. *Pattern Recognition*, 36:2945–2954.

Ronan Collobert and Jason Weston. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of ICML*.

Ronan Collobert, Jason Weston, Leon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *JMLR*, 12:2461–2505.

Trinh-Minh-Tri Do and Thierry Artieres. 2010. Neural conditional random fields. In *Proceedings of AISTATS*.

Jenny R. Finkel, Trond Grenager, and Christopher D. Manning. 2005. Incorporating non-local information into information extraction systems by Gibbs sampling. In *Proceedings of ACL*.

Thorsten Joachims. 2004. *Learning to Classify Text Using Support Vector Machines: Methods, Theory, and Algorithms*. Kluwer Academic Publishers.

Percy Liang, Hal Daume, and Dan Klein. 2008. Structure compilation: Trading structure for features. In *Proceedings of ICML*.

Dong C. Liu and Jorge Nocedal. 1989. On the limited memory BFGS method for large scale optimization. *Math. Programming*, 45:503–528.

Jian Peng, Liefeng Bo, and Jinbo Xu. 2009. Conditional neural fields. In *Proceedings of NIPS 22*.

Xaq Pitkow. 2012. Compressive neural representation of sparse, high-dimensional probabilities. In *Proceedings of NIPS 25*.

Rohit Prabhavalkar and Eric Fosler-Lussier. 2010. Backpropagation training for multilayer conditional random field based phone recognition. In *Proceedings of ICASSP*.

Erik F. Tjong Kim Sang and Sabine Buchholz. 2000. Introduction to the CoNLL-2000 shared task: Chunking. In *Proceedings of CoNLL*.

Erik F. Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the CoNLL-2003 shared task: language-independent named entity recognition. In *Proceedings of CoNLL*.

Richard Socher, Jeffrey Pennington, Eric H. Huang, Andrew Y. Ng, and Christopher D. Manning. 2011. Semi-supervised recursive autoencoders for predicting sentiment distributions. In *Proceedings of EMNLP*.

Joseph Turian, Lev Ratinov, and Yoshua Bengio. 2010. Word representations: A simple and general method for semi-supervised learning. In *Proceedings of ACL*.

Michel Verleysen, Damien Francois, Geoffroy Simon, and Vincent Wertz. 2003. On the effects of dimensionality on data analysis with neural networks. In *Proceedings of the 7th International Work-Conference on Artificial and Natural Neural Networks: Part II*.