LONG SHORT-TERM MEMORY LANGUAGE MODELS WITH ADDITIVE MORPHOLOGICAL FEATURES FOR AUTOMATIC SPEECH RECOGNITION

Daniel Renshaw

University of Edinburgh, UK
daniel.renshaw@ed.ac.uk

Keith B. Hall

Google Inc., United States
kbhall@google.com

ABSTRACT

Abstract Models of morphologically rich languages suffer from data sparsity when words are treated as atomic units. Word-based language models cannot transfer knowledge from common word forms to rarer variant forms. Learning a continuous vector representation of each morpheme allows a compositional model to represent a word as the sum of its constituent morphemes' vectors. Rare and unknown words containing common morphemes can thus be represented with greater fidelity despite their sparsity. Our novel neural network language model integrates this additive morphological representation into a long short-term memory architecture, improving Russian speech recognition word error rates by 0.9 absolute, 4.4% relative, compared to a robust n–gram baseline model.

***Index Terms*—** language modeling, neural networks, long short-term memory, compositional morphology

1. INTRODUCTION

Continuous space language models (CSLMs) have been shown to reduce automatic speech recognition (ASR) word error rate (WER) when used to smooth an n–gram language models (NGLMs) [1]. Typically, CSLMs learn, or use, a continuous vector representation, or embedding, of each word with no explicit consideration for morphology. Any clustering, and thus implied similarity, of embeddings for related forms, such as *hope*, *hopeful*, *hopefully*, etc., is achieved indirectly via contextual similarity alone. Notably, such models are unable to transfer information learned about one form to other related forms. Knowing that *hope* often appears in similar contexts as *optimism* tells these models nothing about how likely *hopeful* is to appear in similar contexts to *optimistic*. This is of particular relevance to languages with rich morphology such as Czech, Russian, and German; languages which may also suffer from more limited training data compared to English.

Morphological language models aim to achieve improved performance by using knowledge of the common morphemes found in rare or unknown words to help inform the task decision. For example, such a model can apply what it knows about the common stem *hope* to the less common variant form *hopefully* since the latter contains the same stem.

We present an alternate formulation of compositional morphology for word representations used previously within a log-bi-linear (LBL) language model [2]. This previous work showed that learning embeddings for each morpheme type, composed additively to form word representations, can provide benefits both intrinsically (perplexity reductions) and extrinsically (better word similarity and machine translation BLEU scores). We show how the same additive morphological representations can be formulated as a hidden layer within a neural network language model (NNLM). Botha and Blunsom [2] experimented with limited quantities of training data; we show a morphology-based NNLM outperforms the NGLM even when large quantities of training data are available.

Word-based NNLMs have been shown to outperform NGLMs [3, 4]. We replicate this general result and go on to show that morphology-based NNLMs outperform word-based NNLMs. We demonstrate improvements in Russian ASR but our approach is general and may help in other tasks or with other morphologically rich languages. Mousa et al. [5] demonstrated the value of morphological inputs to a non-recurrent neural network language model when working with, the morphologically rich, Egyptian Arabic. Unlike [5], we use multi-hot word-morpheme input vectors instead of concatenating one-hot vectors for each component.

Deep NNLMs can achieve better results than single-layer NNLMs [6]. Such models present each word, along with their fixed length contexts, as independent training instances. In contrast, we use a recurrent architecture where each word is presented alone and the model is required to retain whatever information is pertinent from its context internally. Our recurrent model, a variant of the Long Short-Term Memory (LSTM) architecture [7, 8], is shown to outperform a conventional recurrent neural network (RNN) [9].

2. ADDITIVE MORPHOLOGICAL REPRESENTATION

Following [2], we represent a word as the sum of the fixed-length vectors representing its morphemes. For example, if $\overrightarrow{un} = (1, 0, 3)'$, $\overrightarrow{sight} = (2, 4, 1)'$, and $\overrightarrow{ly} = (0, 1, 1)'$ then $\overrightarrow{unsightly} = \overrightarrow{un} + \overrightarrow{sight} + \overrightarrow{ly} = (3, 5, 5)'$.

We include the word surface form as a pseudo-morpheme to account for non-compositional words and avoid order ambiguity, e.g., we wish to avoid $\overrightarrow{understand} = \overrightarrow{under} + \overrightarrow{stand}$, and $\overrightarrow{outlook} = \overrightarrow{out} + \overrightarrow{look} = \overrightarrow{lookout}$. These considerations are of more importance in languages other than English.

In principle, any feature of a word may be used in its additive representation. For example, we may include its phonemes, its length, or its part of speech. The set of such feature types is denoted $\mathcal{F}$. However, the compositional hypothesis breaks down when non-morpheme features are included so we limit ourselves to morpheme features and henceforth refer to $\mathcal{F}$ as the set of morpheme types.

We pack the morpheme vectors into matrix $Q \in \mathbb{R}^{|\mathcal{F}| \times d}$ with a row per morpheme type. This matrix defines a morpheme embedding space of dimensionality d.

The ordered set of known word types $\mathcal{V}$ (i.e. the set of all distinct words found in the training data) are statically morphologically analyzed to form the ordered set of known morpheme types $\mathcal{F}$ (i.e. the set of all distinct morphemes found in the decomposition of all words types in $\mathcal{V}$). The surface form is included as a pseudo-morpheme so

$\mathcal{V} \subseteq \mathcal{F}$. The mapping from words to morphemes is encoded in the sparse binary matrix $M \in \{0,1\}^{|\mathcal{V}|\times|\mathcal{F}|}$ such that $M_{i,j} = 1$ if and only if the i'th word type contains the j'th morpheme type.

Word tokens may be represented using 1-hot vectors. If $w_t \in [1, |\mathcal{V}|]$ is the index of the type of the t'th word of an utterance then $\mathbf{x}_t \in \{0,1\}^{|\mathcal{V}|}$ is a 1-hot vector representation such that $x_{t,i} = 1$ if and only if $w_t = i$.

Words may alternatively be represented via their morphemes using multi-hot vectors. $\mathbf{z}_t = M'\mathbf{x}_t \in \{0,1\}^{|\mathcal{F}|}$ is a multi-hot vector representation such that $z_{t,i} = 1$ if and only if $\mathcal{V}_{w_t} = \mathcal{F}_i$. Given this representation we may define a neural network hidden layer

$$\mathbf{h}_t = \sigma\left(W^{hz}\mathbf{z}_t + \mathbf{b}^h\right) \in \mathbb{R}^{d^h} \quad (1)$$

where $W^{hz} \equiv Q'$ and $d^h \equiv d$ yielding, when using the identity function for $\sigma()$ and $\mathbf{b} = \mathbf{0}$, $\mathbf{h}_t = Q'\mathbf{z}_t = Q'M'\mathbf{x}_t = \left(\sum_{i\in\{j|M_{w_t,j}=1\}} Q_i\right)'$, the additive morphological representation we sought. In practice we generalize with a non-linear activation function, $\sigma()$, and allow a non-zero bias, $\mathbf{b}^h$. Viewing the additive morphological representations in this way highlights the ease with which they can be incorporated into an arbitrary NNLM.

LBL++ [2] encodes the common and valid sequence patterns to be found in a language in the separate morpheme representations for context, Q, and target, R, and the position-dependent transformation matrices, C_j. Our model uses a single representation for morphemic features (including words) which we encode in a hidden layer formulation within a recurrent neural network architecture capable of learning the sequence patterns. Unlike [2], the embedding vectors we learn will thus represent the morphemes more generally with no explicit bias towards context or target positions.

3. NETWORK ARCHITECTURES

Given our morphological input projection layer $\mathbf{h}_t$ (equation 1), an RNN language model is formed by the network equations

$$\mathbf{r}_t = \sigma\left(W^{rh}\mathbf{h}_t + W^{rr}\mathbf{r}_{t-1} + \mathbf{b}^r\right) \in \mathbb{R}^{d^r} \quad (2)$$

$$\mathbf{y}_t = \phi\left(W^{yr}\mathbf{r}_t + \mathbf{b}^y\right) \in \mathbb{R}^{|\mathcal{V}|} \quad (3)$$

where d^r is the dimensionality of the recurrent layer. $\mathbf{r}_t$ can be thought of as the network's memory and may also be viewed as a fixed-size representation of the sequence prefix seen so far.

For $\mathbf{y}_t$ to be the model's probabilistic prediction for w_{t+1}, $\phi()$ must be defined such that $0 \le y_{t,i} \le 1$ and $\sum_{i=1}^{|\mathcal{V}|} y_{t,i} = 1$; we use a hierarchical variant of the softmax function $\phi(\mathbf{s})_i = \frac{exp(s_i)}{\sum_j^{|\mathbf{s}|} exp(-s_j)}$ [10]. For the other two activation functions, which have no restrictions, we use the element-wise logistic function $\sigma(s_i) = \frac{1}{1+exp(-s_i)}$.

In principle, the RNN architecture can learn long distance dependencies by unrolling the recursion to cover the full sequence but in practice this capacity is limited by the vanishing gradient problem [11]. A solution to the vanishing gradient problem is offered by the Long Short-Term Memory (LSTM) model [7] that avoids the problematic non-linearity in the recursion ($\mathbf{r}_{t-1}$ is used inside σ in equation 2 but outside any non-linearity in equation 4). The LSTM also uses input, forget, and output *gates* ($\mathbf{i}_t$, $\mathbf{f}_t$, and $\mathbf{o}_t$ respectively in equations 4 and 5) to attenuate the input, recurrent, and output signals respectively; these allow the model to ignore unimportant inputs, memories, and outputs. We use an LSTM layer with one cell per memory block and an extension [8] that uses a recurrent output projection layer, $\mathbf{p}_t \in \mathbb{R}^{d^p}$, to reduce the number of parameters in the model (since typically $d^p < d^r$). In place of equations 2 and 3 the LSTM model uses equations

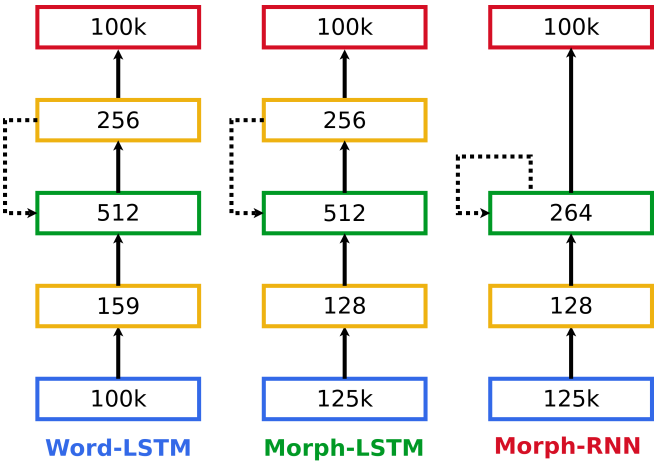

Fig. 1. Depiction of the neural network language models that are the focus of our experiments. Layers are fully connected where edges exist. Dotted edges denote connections to layers in the previous time-step.

$$\mathbf{r}_t = \mathbf{f}_t \odot \mathbf{r}_{t-1} + \mathbf{i}_t \odot \psi\left(W^{ch}\mathbf{h}_t + W^{cr}\mathbf{p}_{t-1} + \mathbf{b}^c\right) \quad (4)$$

$$\mathbf{p}_t = W^{rm}\left(\mathbf{o}_t \odot \psi\left(\mathbf{r}_t\right)\right) \in \mathbb{R}^{d^p} \quad (5)$$

$$\mathbf{y}_t = \phi\left(W^{yr}\mathbf{p}_t + \mathbf{b}^y\right) \in \mathbb{R}^{|\mathcal{V}|} \quad (6)$$

where $\odot$ is the element-wise product operator and $\psi()$ is another type of activation function (we use the element-wise hyperbolic tangent, $\psi(s_i) = tanh(s_i) = \frac{exp(2s_i)-1}{exp(2s_i)+1}$). The input, forget, and output gate activation values are given by equations

$$\mathbf{i}_t = \sigma\left(W^{ih}\mathbf{h}_t + W^{ir}\mathbf{p}_{t-1} + W^{ic}\mathbf{c}_{t-1} + \mathbf{b}^i\right) \quad (7)$$

$$\mathbf{f}_t = \sigma\left(W^{fh}\mathbf{h}_t + W^{fr}\mathbf{p}_{t-1} + W^{fc}\mathbf{c}_{t-1} + \mathbf{b}^f\right) \quad (8)$$

$$\mathbf{o}_t = \sigma\left(W^{oh}\mathbf{h}_t + W^{or}\mathbf{p}_{t-1} + W^{oc}\mathbf{c}_t + \mathbf{b}^o\right) \quad (9)$$

All matrices are dense with the exception of the diagonal matrices W^{ic}, W^{fc}, and W^{oc}– the "peephole connections" [12]. The sigmoid function, σ, in equations 7 through 9 must have the range $[0, 1]$ (we use the element-wise logistic function). This range ensures that the gates determine what proportion of the signals they are multiplied with are propagated.

Either model type, RNN or LSTM, may be trained on 1-hot word vectors or on multi-hot word-morpheme vectors by substituting $\mathbf{x}_t$ for $\mathbf{z}_t$ in the network equations as required. We use the name prefixes *Word–* and *Morph–* to refer to these two model variants.

We focus our experiments on the three neural network language models depicted in figure 1. The layer dimensionalities shown there are for a word vocabulary size of 100k, yielding a morpheme vocabulary size of 125k. In the case of *Word–LSTM*, the input is a 1-hot word vector. In the cases of *Morph–LSTM* and *Morph–RNN* the input is a multi-hot word-morphemes vector. The *Word–LSTM*'s input projection layer and the *Morph–RNN*'s recurrent layer are sized to give those models roughly the same total number of parameters as the *Morph–LSTM*: approximately 42 million.

4. IMPLEMENTATION

4.1. Training data

Our NNLM training data is a simple concatenation of five different written, or transcribed spoken domain, corpora comprising of 4.7 billion sentences or 15.4 billion word tokens. There are 67 million unique word types. 90% of the normalized data set is used for training; the remaining 10% is used to compute perplexity scores. Normalization includes, for example, lower casing but does not include verbalization (i.e., "*21*" $\not\rightarrow$ "*twenty one*"). 2% of our training data tokens were 'unknown' when using a vocabulary truncated at the 100k most frequent word types.

4.2. Morphological analysis

We apply a rule-based morphological analyzer to the task of decomposing words into morphemes. Our analyzer uses a rule-set based on Russian morphology described by Zaliznyak [13]. Analysis labels are attached to the morphemes to disambiguate morpheme types; for example *in* as a prefix and *in* as a stem ("$insight$" $\rightarrow$ "$PX = in$","$S = sight$" and "$in\ sight$" $\rightarrow$ "$S = in$","$S = sight$").

For NNLM training, the vocabulary is truncated, e.g., the 100k most frequently used word types. All word types in the truncated vocabulary are morphologically analyzed, producing a word-to-morphemes mapping. Morphemes that occur only once in the training data (i.e., those that appear in only one word type and where that word type is itself a singleton) are replaced by the pseudo-morpheme <*unk_morph*> because we cannot hope to learn an embedding for infrequently occurring morphemes but do need to learn a representation for unknown morphemes in general. Additional identity mappings are added for the surface forms and for the pseudo-tokens <*s*> and <*/s*>, denoting the start and end of an utterance respectively.

The word-to-morpheme mapping is implemented as a multimap hash table allowing the $\mathbf{z}_t = M'\mathbf{x}_t$ transformation to be computed in $O(1)$ time for known words. Unknown words are dynamically analyzed producing a mixture of known and unknown morphemes. All unknown morphemes in the resulting dynamic analysis are represented by <*unk_morph*>.

4.3. Neural network language model training

We train the NNLMs using back propagation through time (BPTT) with stochastic gradient descent distributed over multiple servers each running multiple threads via the *DistBelief* framework [14]. We use mini-batches of size 20 and a single exponentially decaying learning rate. To reduce the training time, we truncate the BPTT at 5 steps, limiting the model's ability to learn from longer-distance dependencies and giving these models some comparability with the baseline NGLM which is of order 5. Roughly 2.5 epochs of training were completed for each model. Matrix operations are implemented with the Eigen library [15] with OpenMP parallelization.

A typical issue for NNLMs is the high cost of computing the normalized probability scores at the output layer. Naively computing a softmax over a 100k vocabulary is too slow. Botha and Blunsom [2] used a class-based partitioning of the vocabulary to solve the problem. We use the hierarchical softmax [10] approach with cross-entropy objective and a minimum height tree over the vocabulary where internal nodes have up to 1024 children.

The learning curves (see figure 2) flatten out but are still improving when training is terminated. The models are either too simple, or not trained for long enough, for us to experience over-fitting.

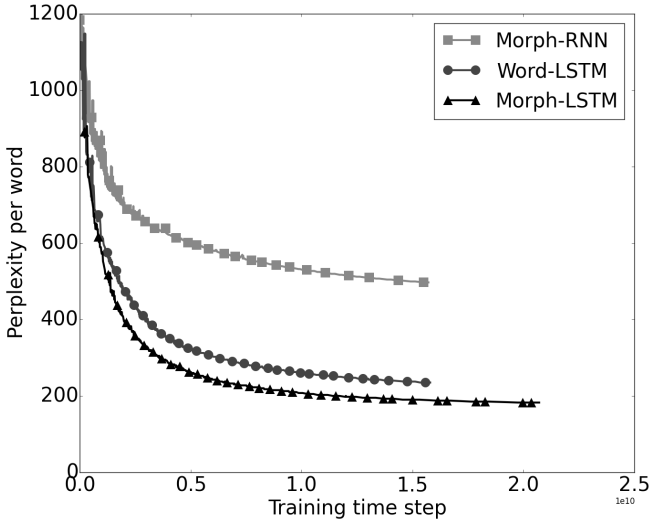

Fig. 2. Neural network language model learning curves. All unknown words are treated as instances of the pseudo-word type <*unk*> when computing perplexity.

Name	Utterances	Tokens
TS1	11,263	34,707
TS2	12,047	40,246
TS3	20,171	63,490
TS4	7,341	19,356

Table 1. Russian testing data; each set is either transcribed dictation or transcribed search queries.

4.4. Speech recognition pipeline and baseline

A production quality ASR pipeline produces a lattice of recognition hypotheses scored by a 1st–pass NGLM. The single best hypothesis from the NGLM scored lattice forms our baseline.

The 1st–pass NGLM is of order 5, has a vocabulary size of 100k, and is smoothed with Stupid Backoff [16]. It is trained on the same corpora as the NNLMs but with additional preprocessing performed (e.g., verbalization [17] in addition to normalization) and the corpora are interpolated instead of concatenated; 'all' is not used.

The NNLMs re-score the 500 best hypotheses from the NGLM scored lattice. The final hypothesis scores are an equal mixture of the original NGLM score, and the NNLM re-score. The NNLMs were not used alone in decoding due to the impractical computational cost of evaluating all possible hypotheses.

Table 1 details the four different test sets used in evaluation. All test sets are transcribed and only include those utterances whose transcriptions are agreed upon by at least two of three transcribers.

5. EXPERIMENTAL RESULTS

Our NNLMs are probabilistic language models and can be compared using the intrinsic evaluation measure perplexity per word (PPW), as long as the output vocabulary remains constant.

$$PPW = exp\left(-\frac{1}{N}\sum_{i=1}^{N} log\left(P\left(w_i \mid w_{1:i-1}\right)\right)\right) \qquad (10)$$

		WER				
Model	PPW	TS1	TS2	TS3	TS4	WM
1st–pass n–gram		18.7	19.8	19.4	26.4	20.4
Word–LSTM	223.5	18.0	19.5	18.8	**24.1**	19.6
Morph–LSTM	**172.1**	**17.8**	**19.4**	**18.6**	24.4	**19.5**
Morph–RNN	485.7	18.2	20.0	19.1	24.9	20.0

Table 2. Main results: word error rates computed over the four test sets described in table 1. All neural network language models trained on 'all' data set with 100k truncated vocabulary. PPW is perplexity per word. WM is test set size weighted mean. Bold values are column minima. The WER change from 1st–pass n–gram to *Morph–LSTM* is significant at $p = 0.02$ (TS2) and $p \leq 7e^{-6}$ (TS1, TS3, and TS4).

	WER				
Model	TS1	TS2	TS3	TS4	WM
Morph–LSTM	17.8	19.4	18.6	24.4	19.5
Morph–LSTM–1m	+0.1	+0.3	+0.3	**-0.2**	+0.2
Morph–LSTM–50k	+0.2	+0.1	+0.3	=	+0.2
Morph–2LSTM	=	**-0.2**	=	**-0.4**	**-0.1**
Morph–LSTM–5pc	**-0.1**	=	+0.1	**-0.2**	=
Morph–LSTM–1pc	**-0.1**	+0.1	=	**-0.1**	=

Table 3. Supplemental results: word error rates change from best main result (in first row) for various model variants (*–1m*: 1 million word type vocabulary, *–50k*: 50 thousand word type vocabulary, *–2LSTM*: 2 LSTM layers in stack, *–5pc*: 5% 'all' training data, *–1pc*: 1% 'all' training data). WM is test set size weighted mean. Bold values are improvements.

PPW cannot be computed for the NGLM because it uses Stupid Backoff, with unnormalized scores as output, and is thus not a probabilistic model.

Of potentially more interest is to compare the models using the extrinsic WER evaluation measure, applied to ASR. WER is a more direct proxy for inverse recognition quality.

$$WER = \frac{substitutions + deletions + insertions}{length\,of\,transcription\,truth} \qquad (11)$$

The results given in table 2 show all our NNLMs improve, on average, over the baseline 1st–pass NGLM. The *Morph–RNN* test set size weighted mean WER improves on that of the NGLM by 0.4 absolute while the *Word–LSTM* improves by 0.8 absolute. The *Morph–LSTM* achieves a greater improvement compared to *Morph–RNN* than when compared to *Word–LSTM* suggesting the change from RNN to LSTM is more significant than the change from word representations to morphological representations. The best improvement over baseline incorporates both of these changes: *Morph–LSTM* achieves a weighted mean WER improvement of 0.9 absolute (4.4% relative).

Our perplexity differences are correlated with the WER differences and show a similar improvement pattern: switching from RNN to LSTM reduces perplexity more substantially than changing the word representation structure. The large difference between RNN and LSTM demonstrates the value of 'learning to forget' in Russian language modeling.

Additional variations of the *Morph–LSTM* model were evaluated to determine the impact of factors such as vocabulary size, network structure, and sensitivity to training data quantity. Table 3 compares these model variants. PPW values are not given since our simplistic handling of unknown words in the perplexity calculations makes the resulting values incomparable when the vocabulary size is changing.

Among the model variants, the largest WER losses compared to the *Morph–LSTM* model occur when the vocabulary size is increased to 1 million word types. It is unclear why increasing the vocabulary size impaired performance. One possibility is that a larger vocabulary includes rarer morphemes and, given the same quantity of training data and training time, the model's representations of these morphemes will be noisier. Furthermore, the rarer morphemes will be those that offer less advantage for transferring information to rare and unknown words. The noisier morpheme representations may thus overpower the expected improvements of transferring knowledge of known morphemes to rare and unknown words. WER also increases when the vocabulary is made smaller so the optimal vocabulary size is between 50k and 1m.

Beyond the results shown in table 3, we found that the WER change was barely noticeable when the NNLMs re-scored the top 1000 hypotheses instead of only the top 500. Heavily skewing the mixing of the 1st–pass NGLM's score and the NNLM's re-score in either direction (i.e., mixing weights of 0.1 or 0.9) substantially increased the resulting WER.

6. CONCLUSIONS AND FUTURE WORK

We have presented a model, *Morph–LSTM*, that reduces the WER of a production-quality Russian ASR system by 0.9 absolute when evenly mixing the model's scores with those of the 1st–pass NGLM on the 500 best hypotheses. The LSTM architecture shows a substantial improvement over the simpler RNN architecture and both offer substantial improvements over the 1st–pass baseline NGLM alone. Switching to morphological word representations provides less of a benefit than switching from the RNN to LSTM.

Other experiments, whose results are not presented here, suggest training on just one of the data sets alone can provide even better WER reductions. A NNLM that interpolates between the available data sets may provide the better average results.

Previous work [18, 19] has found that the dimensionality of word embeddings plays an important role in determining the quality of results. Our models used embeddings of size 128 which is larger than previous optimal sizes which tend to be in the region of 50. Further work is needed to optimize this and similar hyper-parameters.

We have not compared our new model to the LBL++ [2] or to a system that uses a larger, richer, 2nd–pass NGLM. It is unclear how the more complex non-linear neural network model compares with the simple bi-linear model.

There are many options for varying the network input and output forms. For example, instead of including the word surface forms within the vocabulary we could supply the word and its morphemes in two separate inputs allowing the network to learn when one or the other is more important. Similarly we could train the network to predict both the next word and its morphemes via two separate outputs; this may force the network to generalize better.

Using an alternate input representation with an input projection layer is a powerful and principled mechanism for incorporating information about the input beyond the words alone. There is a wide scope for variations and improvements on this theme.

Acknowledgments We would like to thank Brian Roark, Herman Kamper, and Eva Schlinger for their contributions during useful discussions, Haşim Sak and Kaisuke Nakajima for their help with implementation, and to the anonymous reviewers for their helpful feedback.

7. REFERENCES

[1] Holger Schwenk, "Efficient training of large neural networks for language modeling," in *Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on*. IEEE, 2004, vol. 4, pp. 3059–3064.

[2] Jan A. Botha and Phil Blunsom, "Compositional Morphology for Word Representations and Language Modelling," in *Proceedings of the 31st International Conference on Machine Learning (ICML)*, Beijing, China, jun 2014, *Award for best application paper*.

[3] Holger Schwenk, "Continuous space language models," *Computer Speech & Language*, vol. 21, no. 3, pp. 492–518, 2007.

[4] Tomas Mikolov, Jirí Kopecký, Lukas Burget, Ondrej Glembek, and Jan Cernocký, "Neural network based language models for highly inflective languages," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP 2009, 19-24 April 2009, Taipei, Taiwan*, 2009, pp. 4725–4728.

[5] A El-Desoky Mousa, H-KJ Kuo, Lidia Mangu, and Hagen Soltau, "Morpheme-based feature-rich language models using deep neural networks for lvcsr of egyptian arabic," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE, 2013, pp. 8435–8439.

[6] Ebru Arisoy, Tara N Sainath, Brian Kingsbury, and Bhuvana Ramabhadran, "Deep neural network language models," 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*. Association for Computational Linguistics, 2012, pp. 20–28.

[7] Sepp Hochreiter and Jürgen Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[8] Haşim Sak, Andrew Senior, and Françoise Beaufays, "Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition," *arXiv preprint arXiv:1402.1128*, 2014.

[9] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur, "Recurrent neural network based language model.," in *INTERSPEECH*, 2010, pp. 1045–1048.

[10] Frederic Morin and Yoshua Bengio, "Hierarchical probabilistic neural network language model," in *AISTATS*. Citeseer, 2005, vol. 5, pp. 246–252.

[11] Yoshua Bengio, Patrice Simard, and Paolo Frasconi, "Learning long-term dependencies with gradient descent is difficult," *Neural Networks, IEEE Transactions on*, vol. 5, no. 2, pp. 157–166, 1994.

[12] Felix A Gers, Nicol N Schraudolph, and Jürgen Schmidhuber, "Learning precise timing with lstm recurrent networks," *The Journal of Machine Learning Research*, vol. 3, pp. 115–143, 2003.

[13] Andrey Zaliznyak, *Grammaticheskij slovar' russkogo jazyka*, Russkiy Yazik, Moscow, 1977.

[14] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al., "Large scale distributed deep networks," in *Advances in Neural Information Processing Systems*, 2012, pp. 1223–1231.

[15] Gaël Guennebaud, Benoît Jacob, et al., "Eigen v3," http://eigen.tuxfamily.org, 2010.

[16] Thorsten Brants, Ashok C. Popat, Peng Xu, Franz J. Och, and Jeffrey Dean, "Large language models in machine translation," in *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, 2007, pp. 858–867.

[17] Hasim Sak, Françoise Beaufays, Kaisuke Nakajima, and Cyril Allauzen, "Language model verbalization for automatic speech recognition," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2013, pp. 8262–8266.

[18] Ronan Collobert and Jason Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in *Proceedings of the 25th international conference on Machine learning*. ACM, 2008, pp. 160–167.

[19] William Blacoe and Mirella Lapata, "A comparison of vector-based representations for semantic composition," in *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. Association for Computational Linguistics, 2012, pp. 546–556.