From Feedforward to Recurrent LSTM Neural Networks for Language Modeling

Martin Sundermeyer, Hermann Ney, *Fellow, IEEE*, and Ralf Schlüter, *Member, IEEE*

***Abstract*—Language models have traditionally been estimated based on relative frequencies, using count statistics that can be extracted from huge amounts of text data. More recently, it has been found that neural networks are particularly powerful at estimating probability distributions over word sequences, giving substantial improvements over state-of-the-art count models. However, the performance of neural network language models strongly depends on their architectural structure. This paper compares count models to feedforward, recurrent, and long short-term memory (LSTM) neural network variants on two large-vocabulary speech recognition tasks. We evaluate the models in terms of perplexity and word error rate, experimentally validating the strong correlation of the two quantities, which we find to hold regardless of the underlying type of the language model. Furthermore, neural networks incur an increased computational complexity compared to count models, and they differently model context dependences, often exceeding the number of words that are taken into account by count based approaches. These differences require efficient search methods for neural networks, and we analyze the potential improvements that can be obtained when applying advanced algorithms to the rescoring of word lattices on large-scale setups.**

***Index Terms*—Feedforward neural network, Kneser-Ney smoothing, language modeling, long short-term memory (LSTM), recurrent neural network (RNN).**

I. Introduction

In MANY natural language-related problems, a language model (LM) is the essential statistical model that captures how meaningful sentences can be constructed from individual words. Arising directly from a factorization of Bayes'decision rule, the LM estimates the probability of occurrence for a given word sequence. Incorporating such probability estimates is vital for state-of-the-art performance in many applications like automatic speech recognition, handwriting recognition, and statistical machine translation.

The probability distribution over word sequences can be directly learned from large amounts of text data, and numerous approaches for obtaining accurate probability estimates have been proposed in the literature (cf. [1] for an overview). However, count-based LMs as defined in [2] and subsequently improved in [3] and [4] have consistently outperformed any other method for decades. This family of models relies on the observation that relative frequencies are the optimum solution for estimating word probabilities, given a maximum likelihood training criterion under a Markov assumption (cf. [5]).

Only more recently, by the introduction of neural networks to the field of language modeling in [6], substantial improvements in perplexity (PPL) over count LMs could be achieved. Since the first successful application of neural networks to language modeling, different types of neural networks have been explored, where the performance of the original approach could be considerably improved. In particular, previous work has concentrated on feedforward neural networks ([7], [8]), recurrent neural networks ([9]), and recurrent long short-term memory (LSTM) neural networks ([10]). Besides the type of the neural network, the precise structure can have a strong impact on the overall performance. E.g., in acoustic modeling it was observed that deep neural networks greatly improve over shallow architectures [11], where in the former case multiple neural network layers are stacked on top of each other, as opposed to the latter case where only a single hidden layer is used.

Apart from the modeling itself, there arises the problem of how to apply the neural network in decoding. While decoding with count LMs seems well understood (see e.g. [12] for efficient decoding in speech recognition), it is more difficult to incorporate probabilities from neural network LMs in decoding due to their high computational complexity and a potential mismatch in context size: Count LMs usually do not exceed a context size of three to four words, but feedforward neural networks can be trained with considerably higher context sizes, and for a recurrent neural network LM the context size in theory is unbounded. This paper makes the following scientific contributions:

1) We study the search issue for long-range feedforward neural network LMs and especially for recurrent neural network LMs.
2) We present a systematic comparison of count LMs, feedforward, recurrent, and LSTM neural network LMs.

Manuscript received July 27, 2014; revised November 25, 2014; accepted January 19, 2015. Date of current version February 26, 2015. This work was supported in part by OSEO, French State agency for innovation, and partly realized as part of the Quaero programme and in part by the European Union Seventh Framework Programme (FP7/2007-2013) under Grants 287658 (EU-Bridge) and 287755 (transLectures). The work of H. Ney was supported in part by a senior chair award from DIGITEO, a French research cluster in Ile-de-France. Experiments were performed with computing resources granted by JARA-HPC from RWTH Aachen University under project "jara0085." The guest editor coordinating the review of this manuscript and approving it for publication was Prof. Marcello Federico.

M. Sundermeyer and R. Schlüter are with the Chair of Computer Science 6, Computer Science Department, RWTH Aachen University, 52062 Aachen, Germany (e-mail: sundermeyer@cs.rwth-aachen.de; schlueter@cs.rwth-aachen.de).

H. Ney is with the Chair of Computer Science 6, Computer Science Department, RWTH Aachen University, 52062 Aachen, Germany, and also with the Spoken Language Processing Group, LIMSI-CNRS, 91400 Orsay, France (e-mail: ney@cs.rwth-aachen.de).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TASLP.2015.2400218

3) Experiments are conducted that verify the strong correlation between perplexity and word error rate (WER), regardless of the LM that is used.
4) Inspired by the success of deep neural networks in acoustic modeling, we address the question whether deep neural networks are helpful for language modeling as well.

We present experimental results on two well-tuned English and French large-vocabulary speech recognition tasks. Nevertheless, the conclusions we draw are not limited to speech recognition, but can be generalized to other problems like statistical machine translation, as shown e.g. in [13]. The paper is organized as follows: In Section II we present related work on neural network language modeling. Then, we give a complete review of the neural network language model variants investigated in this work, where we also cover details of the training algorithms and speed-up techniques. We continue by describing advanced lattice rescoring techniques that allow the efficient application of neural network LMs to large-scale speech recognition. Finally, in Section V we show experimental results, and Section VI concludes our work.

II. Related Work

In [14] and [15], a feedforward and a recurrent neural network (RNN) were compared in terms of perplexity on one million and on 24 million running words of Wall Street Journal data, respectively. In both cases, the RNN performed consistently better by more than 10% relative. A more detailed study based on perplexities was presented in [16], where it was investigated in which cases count LMs and neural network LMs performed best. This analysis was extended to WER results in [17], where feedforward and recurrent LSTM networks were compared on 27 million running words of French broadcast conversational data. The LSTM obtained a perplexity which was lower by more than 15% relative compared to the feedforward model, and these improvements also carried over to the word error rate level. Only in [18] it was observed that a recurrent neural network was outperformed by a feedforward architecture. A feedforward 10-gram obtained a perplexity which was lower by 5% relative on a large scale English to French translation task, but the final performance in terms of BLEU was identical for both networks. For efficiency reasons, the RNN was not trained with standard backpropagation through time.

To the best of our knowledge, so far only [15] investigated the potential gains by deep networks for language modeling. Virtually no improvements in PPL or WER were obtained when interpolating a count LM and a feedforward model trained on the same amount of data.

A comparative evaluation of different kinds of LMs is closely related to the algorithms that are used for decoding. All of the aforementioned works have either not covered any decoding experiments, or restricted the analysis of recurrent neural networks to the rescoring of comparatively small n-best lists (where n is at most 300). In [19] and [20], an improved rescoring of n-best lists was investigated, reducing the computational effort by caching and compressing the lists into a prefix tree. As n-best lists can only encode a small-sized search space with little variation, the use of n-best lists may hide the full potential of neural network LMs. Early work on using long-span structured LMs for lattice rescoring includes [21]. For neural network LMs, this was first addressed in [22], where a hill climbing algorithm was presented that could be applied to word lattices instead of n-best lists. In [23], the push-forward algorithm for RNN LMs was introduced in a machine translation setting. This algorithm extracts the single best hypothesis from a word lattice in an efficient manner. The work did not take into account the rescoring of all hypotheses in a word lattice with an RNN LM, which was subsequently addressed in [24] and [25] for speech recognition. While [24] creates the rescored lattices directly based on an algorithm from [26], in [25] rescored lattices were obtained as a by-product of the push-forward algorithm, without a degradation in Viterbi word error rate. Additional improvements were obtained by applying confusion network rescoring (cf. [27]) on the lattices.

More recently, research also focussed on using RNN LMs directly in first pass decoding. In [28], a fine-grained cache architecture was presented to reduce the computational overhead incurred by the RNN, but compared to the push-forward algorithm, a degradation in WER was observed. In [29], an RNN LM was integrated into a weighted finite state transducer-based decoder, which helped reducing the latency of the speech recognition process, but only led to very small gains in WER. In addition, no higher order expansion of the RNN was considered. In summary, there is little evidence that it is necessary to integrate an RNN LM into first pass speech decoding to obtain optimum word error rate results. However, as noted in [17] and [25], it is important to obtain a fully rescored lattice incorporating neural network probabilities, because this allows advanced methods such as confusion network rescoring, which we will take into account in our comparative study.

III. Review of Neural Network LMs

In this section, we give a brief overview of the neural network LM types that we investigate in this paper. For a review of count LMs, we refer to [4].

A. Feedforward Neural Network LMs

Neural networks were first introduced to the field of language modeling based on a feedforward architecture (cf. [6]). Similar to count models, they are based on a Markov assumption, i.e., the probability of a word sequence w_1^N is decomposed as

$$p(w_1^I) = \prod_{i=1}^{I} p(w_i | w_{i-n+1}^{i-1}) \tag{1}$$

such that only the most recent $(n-1)$ preceding words are considered for predicting the current word w_i. An example of a trigram feedforward neural network LM is depicted in Fig. 1, which corresponds to the following set of equations:

$$y_i = A_1 \hat{w}_{i-2} \circ A_1 \hat{w}_{i-1} \tag{2}$$

$$z_i = \sigma(A_2 y_i) \tag{3}$$

$$p(c(w_i)|w_{i-2}, w_{i-1}) = \varphi(A_3 z_i)|_{c(w_i)} \tag{4}$$

$$p(w_i|c(w_i), w_{i-2}, w_{i-1}) = \varphi(A_{4,c(w_i)} z_i)|_{w_i} \tag{5}$$

Here, by A_1, A_2, A_3 and $A_{4,\cdot}$ we denote the weight matrices of the neural network. We do not explicitly include a bias term in

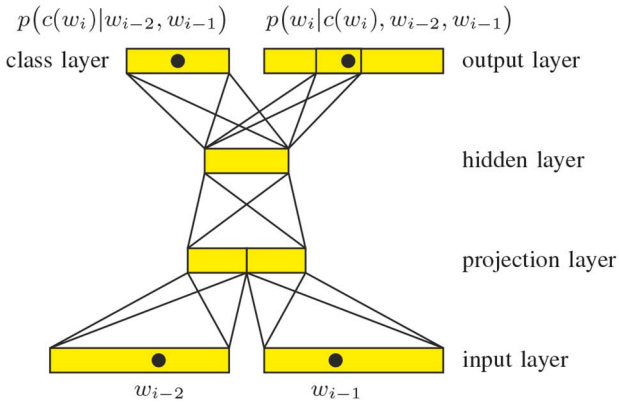

Fig. 1. Trigram feedforward neural network architecture with word classes at the output layer.

the above formulas, as these can be included in the weight matrix multiplication (see e.g. [30]). The input data $\hat{w}_{i-2}$ and $\hat{w}_{i-1}$ are the one-hot encoded predecessor words w_{i-2} and w_{i-1}, where the weight matrix A_1 is tied for all history words. The vectors $A_1\hat{w}_{i-2}$ and $A_1\hat{w}_{i-1}$ are then concatenated, indicated by the $\circ$ operator, to form the projection layer activation y_i. Multiplying y_i with A_2, and applying the sigmoid activation function

$$\sigma(x) = \frac{1}{1 + \exp(-x)}, \tag{6}$$

which is computed element-wise for the vector $A_2 y_i$, results in the hidden layer activation z_i.

The evaluation of the output layer incurs most of the effort for obtaining the word posterior probability $p(w_i|w_{i-2}, w_{i-1})$. To reduce the computational complexity, in [32] (based on an idea from [31]) a word class mapping c was introduced, which assigns a unique word class to each word from the vocabulary. Given the word class mapping, the following factorization is used:

$$\begin{aligned} p(w_i|w_{i-2}, w_{i-1}) = {} & p(c(w_i)|w_{i-2}, w_{i-1}) \\ & \cdot p(w_i|c(w_i), w_{i-2}, w_{i-1}). \end{aligned} \tag{7}$$

For a given trigram (w_{i-2}, w_{i-1}, w_i), the distribution $p(\cdot|w_{i-2}, w_{i-1})$ needs to be computed only for the distinct word classes, and the distribution $p(\cdot|c(w_i), w_{i-2}, w_{i-1})$ needs to be evaluated only for the words assigned to the class $c(w_i)$. In practice, the word classing can be chosen such that, on average, both the number of word classes and the number of words assigned to an individual class are significantly smaller than the vocabulary size, which leads to a huge speedup. For the class posterior probability (4), the weight matrix A_3 is used, whereas for the word posterior probability (5), the weight matrix $A_{4,c(w_i)}$ is dependent on the class $c(w_i)$ of the word w_i. By φ, we indicate the softmax activation function

$$\varphi(x)|_j = \frac{\exp(x_j)}{\sum_{k=1}^{|x|} \exp(x_k)}, \tag{8}$$

which enforces normalization of the probability estimates, where $|x|$ is the dimension of the vector x.

Word classes can be derived from the training data in an unsupervised fashion in various ways. E.g., in [14], word classes were obtained based on unigram frequencies. In [33] and [34], a perplexity-based approach was proposed for obtaining word classes. Regarding the use of word classes in neural network LMs, these clustering algorithms were compared in [35], and it was found that perplexity-based word classes resulted in considerable improvements over the unigram frequency variant when used for the output layer factorization of a neural network LM. In this work, we thus stick to perplexity-based word classes, where we rely on a refined optimization algorithm presented in [36] that strongly reduces the training costs compared to the original version from [33], while the speedup has no effect on accuracy.

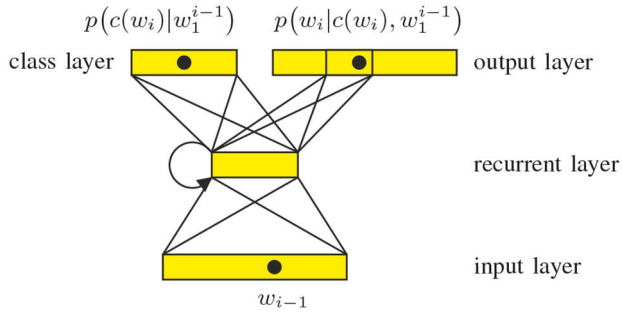

Fig. 2. Architecture of a standard recurrent neural network.

B. Recurrent Neural Network LMs

Recurrent neural network LMs as introduced in [9] follow a very similar topology compared to feedforward neural networks. An example of a recurrent neural network is shown in Fig. 2, where the RNN is defined by the following equations:

$$y_i = \sigma(A_1\hat{w}_{i-1} + Ry_{i-1}) \tag{9}$$

$$p(c(w_i)|w_1^{i-1}) = \varphi(A_2 y_i)|_{c(w_i)} \tag{10}$$

$$p(w_i|c(w_i), w_1^{i-1}) = \varphi(A_{3,c(w_i)} y_i)|_{w_i}. \tag{11}$$

As before, the weight matrices of the RNN are denoted by A_1, A_2, and $A_{3,\cdot}$. However, there is an additional weight parameter matrix R for the recurrent connections, which is multiplied by the previous hidden layer activation vector y_{i-1}. At the very beginning of a word sequence, when $i = 1$, the vector y_0 is commonly set to zero. The RNN LM is only explicitly conditioned on the direct predecessor word, and in that way, it much resembles a bigram feedforward neural network LM. On the other hand, by evaluating the RNN equations word by word for an entire sequence, the output layer result for a word w_i depends on the entire sequence of history words w_1^{i-1}. The activations of the recurrent hidden layer then act as a memory that automatically stores previous word information. Unlike feedforward neural networks, RNNs are thus operating on sequences instead of unrelated individual events.

C. LSTM Neural Network LMs

One of the main advantages of RNN LMs over feedforward neural network LMs is that no explicit dependence on a pre-defined context length has to be assumed, and, at least conceptually, it is possible to take advantage of long range word dependences. While the RNN architecture itself facilitates the use of long range history information, in [37] it was found that standard

gradient-based training algorithms fall short of learning RNN weight parameters in such a way that long-range dependences can be exploited. This is due to the fact that, as dependences get longer, the gradient calculation becomes increasingly unstable: Gradient values can either blow up or decay exponentially with increasing context lengths. As a solution to this problem, in [38] a novel RNN architecture was developed which avoids vanishing and exploding gradients, while it can be trained with conventional RNN learning algorithms. This idea was subsequently extended in [39] and [40]. The resulting improved RNN architecture is referred to as long short-term memory neural network (LSTM).

The neural network architecture for LSTMs can be chosen exactly as depicted in Fig. 2, except that the standard recurrent hidden layer is replaced with an LSTM layer instead. More precisely, this means that Eq. (9) is replaced with the sequence of equations

$$\iota_i = \sigma(A_{x\iota}x_i + A_{y\iota}y_{i-1} + A_{c\iota}c_{i-1}) \tag{12}$$
$$\phi_i = \sigma(A_{x\phi}x_i + A_{y\phi}y_{i-1} + A_{c\phi}c_{i-1}) \tag{13}$$
$$c_i = \phi_i c_{i-1} + \iota_i \tanh(A_{xc}x_i + A_{yc}y_{i-1}) \tag{14}$$
$$\omega_i = \sigma(A_{x\omega}x_i + A_{y\omega}y_{i-1} + A_{c\omega}c_i) \tag{15}$$
$$y_i = \omega_i \tanh(c_i). \tag{16}$$

The indices of the weight matrices, e.g., x and ι in the case of $A_{x\iota}$, do not indicate a dependence on the values of a variable x or ι, but are only meant to distinguish the eleven LSTM weight matrices. In Fig. 3, the LSTM equations are depicted graphically. The quantities ι_i, ϕ_i and ω_i are sometimes referred to as input, forget, and output gates, respectively, as their values lie in the (0,1) interval. By x_i, we denote the input to the LSTM layer. For the RNN architecture from Fig. 2 without a projection layer, we have $x_i = \hat{w}_{i-1}$. Alternatively, we can set x to the activations of a projection layer, if such an additional layer for mapping words to continuous features is used. In [41], it was found that a projection layer would give only tiny improvements in perplexity for an RNN, and a similar observation was made in [10] for LSTMs. Therefore, in principle, it seems unnecessary to make use of a projection layer, and the model could be simplified by leaving out the additional layer. Nevertheless, it makes sense to incorporate a projection layer, at least for feedforward and LSTM neural network LMs: Even when word classes are used at the output layer, the main computational effort still accounts for the matrix multiplication between the last hidden layer and the output layer. For a feedforward network, the hidden layer size grows linearly with increasing n-gram order, and as a certain dimension for a continuous space word representation is required to obtain good performance, the projection layer can be quite large for higher order LMs. Therefore, it is preferable not to connect the projection layer directly to the output layer, but to have an intermediate hidden layer that compresses the weight matrix dimensions. This argument does not apply to recurrent neural networks in general, because they only model a bigram dependence explicitly. For LSTMs in particular, the effect of a projection layer is as follows. Let J be the size of an LSTM layer, and let I and K be the size of the previous and the next layer, respectively. From Eqs. (12) to (15) it immediately follows that there are $4IJ$ many weights connecting the LSTM layer with its predecessor layer, and JK many weights connecting the LSTM layer with its successor layer. When omitting the projection layer, we have $I = V$ for a vocabulary size of V. When adding a projection layer with $I \ll V$, we can reduce the number of neural network parameters at the input virtually to one fourth, without a degradation in performance. Therefore we always combine feedforward and LSTM neural network LMs with a projection layer.

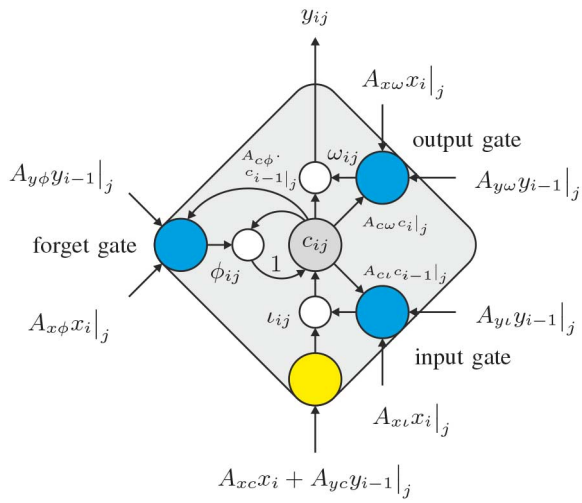

Fig. 3. Graphical depiction of the LSTM equations for the j-th LSTM unit of a hidden layer.

D. Neural Network Training

For training the neural network language models, we use the cross-entropy error criterion, which is equivalent to maximum likelihood, i.e., we minimize the objective function

$$F(A) = -\sum_{i=1}^{M} \log p_A(w_i|w_{i-n+1}^{i-1}), \tag{17}$$

where M is the number of running words in the text corpus.

We train the neural network LMs with the stochastic gradient descent algorithm (see e.g. [42]). In the case of feedforward neural networks, the gradient is computed with the backpropagation algorithm (cf. [43]). For obtaining the gradient of a recurrent neural network, several versions of the backpropagation through time (BPTT) algorithm exist (see [43] and [44]). E.g., in [45] a variant of the truncated BPTT algorithm was used for training standard RNNs: At each word position, an approximate gradient is computed by taking into account a fixed number of predecessor words, which is then used to update the weight parameters. To reduce the computational effort, the gradient computation and the weight update were carried out after a pre-defined number of words, instead of updating at each word position.

In [41], the epochwise BPTT algorithm was applied to the training of LSTM LMs. This algorithm performs two passes over a word sequence, computing the gradient and updating the weights once for the entire sequence. The time complexity of epochwise BPTT is actually lower than that of the truncated BPTT algorithm. However, epochwise BPTT may be more appropriate for training LSTMs than standard RNNs because of

the vanishing and exploding gradient problem, which gets more problematic when the gradient is computed for longer contexts, and in particular when the context is extended to a full word sequence. Epochwise BPTT requires separating the training data into sequences. In [41], several methods were proposed. For text corpora where the sentences are related and their order is maintained, it usually works best to concatenate multiple sentences up to a certain maximum length, and to use these as the underlying sequences for epochwise BPTT.

This also means that even for a recurrent neural network, the context length used in training is bounded either, similar to feedforward models, by a fixed number of words (truncated BPTT), or it cannot exceed the beginning of the sequence (epochwise BPTT). Nevertheless, the context length exploited when applying the neural network can be longer than the context length considered in training (cf. [38]).

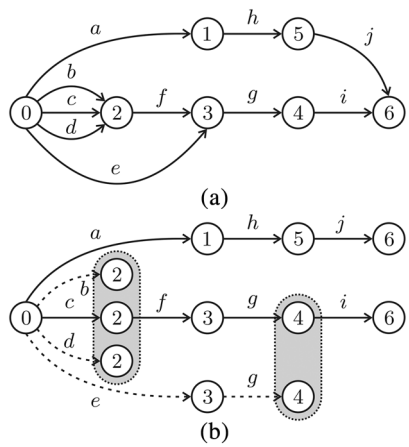

Fig. 4. (a) Example word lattice and (b) a corresponding traceback tree. Dashed arcs correspond to pruned paths, and background grey indicates recombined paths.

IV. Rescoring with Neural Network LMs

The simplest approach for turning the perplexity improvements of neural network LMs into word error rate reductions is by rescoring n-best lists. The LM probability according to the neural network is computed for each entry of the list, and the best scoring hypothesis is selected as the final result. An advantage of this method is that the neural network LM is applied without any approximations, regardless of the context size of the neural network. On the other hand, only relatively few hypotheses can be considered in this way, which impacts the performance of Viterbi rescoring (cf. [23]) and especially confusion network (CN) rescoring ([17]).

A better representation of the search space is obtained by rescoring lattices as a replacement of n-best lists. Lattices are usually created with a count LM using a context size of at most four words. If a feedforward neural network LM is used, it is possible to simply replace the count LM estimates with those of the neural network model, and to use standard rescoring algorithms directly on the lattice. The replacement step may require an expansion of the lattice ([47]). E.g., if the order of the count LM is n, and the neural network LM is of order $(n+1)$, a word arc in the lattice may have two different predecessor paths of length $(n+1)$ that match only in the last n word positions. As a result, assigning a probability to the word arc according to the neural network LM would be ambiguous, and the two paths have to be kept separate for rescoring. In practice, lattice expansion can dramatically increase the size of a lattice, and it becomes too expensive if the difference in LM order is large. In the case of an RNN, the expanded lattice degenerates into a prefix tree.

For these reasons, with higher order neural network LMs an approximate rescoring is necessary for lattices. When Viterbi rescoring is used, only the single best path, and, to some extent, its competing hypotheses need to be evaluated with the neural network LM. In the case of CN rescoring, neural network LM probability estimates are needed for all hypotheses encoded in the lattice.

A. Viterbi Rescoring

Given a lattice based search space, it is possible to extract the single best hypothesis with the push-forward algorithm from [23], which effectively expands the word lattice into a prefix tree on the fly.

The lattice is given as an undirected graph consisting of nodes and arcs. An example is depicted in Fig. 4(a), where the vocabulary consists of the words a to j. Each arc is labeled with a word recognized in first pass decoding, and each node is associated with a corresponding word end time. For a node, a list of hypotheses is maintained, where a hypothesis stores the RNN hidden layer that was obtained by evaluating the RNN on the word labels observed on the individual path from the start node of the lattice up to the current node. The nodes are visited in topological order, starting from the initial node of the lattice. At a given node, for each hypothesis and each outgoing arc, the RNN state from the hypothesis is expanded by the word label of the arc, and the resulting RNN state is added to the hypotheses list of the successor node. To reduce the computational effort, the number of hypotheses per node is limited to a maximum number, retaining only the best scoring hypotheses (cardinality pruning). In addition, only the single best hypothesis per n-gram context is kept at a node (recombination pruning).

As noted in [25], this algorithm can be improved by more advanced pruning and look-ahead techniques, similar to first pass decoding. Due to the availability of word end times, the nodes can be sorted by time in ascending order. Traversing the lattice in a time-synchronous fashion has the advantage of allowing to compare different hypotheses that correspond to the same portion of the acoustic signal. This facilitates efficient beam pruning on a lattice node level (see e.g. [48]): The most likely hypothesis for a word end time is obtained, and all hypotheses having a probability lower than the best one, multiplied by an empirically determined factor, are pruned.

In addition, other than in first pass decoding, it can be exploited that the complete acoustic and LM probabilities from the count LM are already present on the lattice arcs. Therefore, it is possible to incorporate the probabilities of future word arcs at the current word position, either by taking into account the sum over all future word arcs, or by considering the single best path only.

Another issue arising in rescoring relates to the interdependence of consecutive word sequences. If the speech data is separated into multiple utterances, it is most convenient to rescore

each utterance independently of the others, which also simplifies parallelized rescoring. However, conceptually it would be necessary to consider any hypothesis for the previous utterance as a candidate RNN state initialization for rescoring the current utterance. As a compromise, in this work the rescoring is performed either independently, or the single best previous hypothesis is used as initialization for the rescoring of the current utterance.

B. Lattice Generation

When using advanced rescoring methods such as CN rescoring, or estimating word confidences, multiple recognition hypotheses are required, which are ideally encoded as a word lattice. Thus, in this section two extensions of the push-forward algorithm are described that not only extract the single best hypothesis from a lattice with respect to a neural network LM, but also output a new lattice including neural network LM probability estimates for all paths from the original lattice. To this end, it is helpful to make use of a so-called traceback data structure (from [48]). The traceback is a prefix tree that stores all the paths that were considered during a run of the push-forward algorithm, including the corresponding acoustic and language model probabilities. The tree structure makes it possible to extend a partial path by another word arc from the lattice in constant time, regardless of the length of the partial path. In Fig. 4(b), an example traceback is shown after a hypothetical rescoring run on the lattice from Fig. 4(a), where we assume that the paths $b-f-g-i$ and $d-f-g-i$ have been pruned at node 2, and the path $e-g-i$ has been pruned at node 4. As a result, the example traceback tree contains three paths not ending at the final node 6, which is indicated by dashed arrows. In the following, two extensions of push-forward algorithm are discussed that make use of the traceback tree to obtain an approximate word lattice of neural network LM probabilities.

1) Replacement Approximation: We first restrict to the case where the new lattice has the same topology as the original one, i.e., only the LM probabilities are replaced. We can sort the paths from the traceback tree by the word end time of the last node and the probability of the path in descending order. Then, for each path from the traceback tree, the word labels are followed in reverse order, and the LM probabilities associated with the traceback arcs are written on the corresponding arcs of the word lattice. In case a lattice arc is visited a second time, the LM probability of the arc does not get updated again. In this way, an arc gets assigned the word probability it obtained on the overall best path in the traceback tree. In addition, if the lattice encodes an n-best list, this approximate lattice rescoring strategy leads to an exact rescoring.

2) Traceback Approximation: As an alternative, we can directly make use of the traceback tree and convert it into a lattice, incorporating the neural network LM probabilities. Obviously, this will lead to an increased size of the rescored lattice because of the on-the-fly expansion. In a first step, we create a new final node, and connect all nodes from the traceback tree, that correspond to the final node of the original lattice, to the new final node via ε transitions.

Paths from the traceback tree that do not end at the final node have been pruned during Viterbi rescoring. A lattice may not contain pruned paths, thus pruned paths need to be extended to paths ending at the final node, in order to include them in the new lattice. We recombine a path that has been pruned at a particular lattice node with another path that survived pruning at this node. In the example of Fig. 4(b), there are two paths that have been expanded up to lattice node 4: The pruned path $e-g$, and the path $c-f-g$ that has survived pruning at node 4. Recombining the paths means that we merge the two traceback nodes corresponding to lattice node 4 in the new lattice. As a result, we extend the pruned path $e-g$ to a complete path $e-g-i$ that ends at the final lattice node 6.

Unlike in the example, there may be multiple surviving paths that can be used for recombination with a pruned path. For recombination we choose the path whose probability is closest to the pruned one, but which is still higher than that of the pruned path. Here, we denote the probability of a path as the product of the word probabilities from the initial traceback node up to the traceback node considered for merging. (Because of recombination pruning, a surviving path is not necessarily better than a pruned one: Recombination pruning will keep only the best path for a given n-gram context, and if paths with two different n-gram contexts h and h' survive beam pruning, then the second best path with context h may be better than the best path with context h'.) We ensure that the path for recombination is better than the pruned one to enforce that the Viterbi rescoring result of the rescored lattice will be the same as the result obtained by the push-forward algorithm. The original lattice contains the same number of paths as the rescored lattice. The increase in lattice size can be dynamically controlled by adjusting the pruning parameters of the push-forward algorithm.

V. Experimental Result

For the experiments of this work, we concentrate on two state-of-the-art speech recognition systems for English and French that have been developed for the project *Quaero*[1]. Both systems obtained the best word error rates in the final 2013 evaluation.

For acoustic modeling, a Gaussian mixture model was trained on manually transcribed broadcast news and broadcast conversations. In the case of English, a total of 250 hours was used, for French, the acoustic training data comprised 350 hours. Both systems included multilingual bottleneck multi-layer perceptron (MLP) features, as proposed in [50], following the tandem approach ([51]). The features were trained on 840 hours of English, French, German, and Polish data. More details on the acoustic setup of the systems can be found in [49] and [25]. Table I summarizes the LM corpora that were used in this work. For both languages, from all available training data, Kneser-Ney-smoothed LMs ([3]) were estimated on the individual text sources. Then, the LMs were linearly interpolated, where the interpolation weights were optimized to minimize the perplexity on the development data. Finally, the corpora having a non-negligible interpolation weight were selected for the full training data set. On these data, the final

[1]http://www.quaero.org

TABLE I
OVERVIEW OF THE LM CORPORA USED FOR THE EXPERIMENTS IN THIS WORK. NUMBERS EXCLUDE SENTENCE BOUNDARY TOKENS

Corpus			Running Words	Vocabulary
English	Train	Full	3.1 B	150 K
		Reduced	50 M	
	Dev		39 K	
	Test		35 K	
French	Train	Full	1.6 B	200 K
		Reduced	100 M	
	Dev		35 K	
	Test		41 K	

TABLE II
PERPLEXITY RESULTS ON THE FRENCH DEVELOPMENT AND TEST DATA

LM	Perplexity Dev	Perplexity Test
Count-based 4-gram (Reduced)	123.9	144.6
Count-based 4-gram (Full)	102.9	122.0
LSTM	98.6	114.9
+ Count-based 4-gram (Full)	79.9	94.4

Kneser-Ney model was estimated that was used for first pass decoding. As it is too time-consuming to train a neural network LM on all of the data, to this end a subset of 50 M and 100 M running words was selected for English and French, respectively. As we train neural network LMs for 30 epochs on all the data, billions of words are processed in training. The English data set was chosen to be smaller to compensate for the large hidden layer sizes used in the experimental comparison. Both the English and the French reduced data set were selected from all of the available training data by including the in-domain data first, and then adding more data from the second most relevant data source. To sort the data sources by relevance, we considered the interpolation weights of the large Kneser-Ney models, which we normalized with respect to the number of running words.

A. Rescoring with Neural Network LMs

In the first place, we analyze the effect of different rescoring techniques with a neural network LM. For the algorithms described in Section IV, we trained an LSTM neural network LM with a projection layer and a hidden layer each of size 300 on the French 100 M word training corpus. For speed-up, 1000 perplexity-based word classes were used to factorize the output layer. The perplexities of this LM are shown in Table II. For comparison, we include perplexity values of a Kneser-Ney (KN) 4-gram LM, once trained on the same data as the LSTM and once trained on the full data set. From the results it can be observed that the KN model is considerably improved by adding the additional data, reducing the perplexity by 16.9% relative on the development data and by 15.6% relative on the test data, i.e., the additional training data are relevant for the domain of the test data. Even though the LSTM is trained on the smaller data set, it still obtains consistently better perplexities than the KN model trained on the full data set. The lexicon of the speech recognizer contains 200 K words, but not all of them are covered by the reduced training data set of 100 M running words. For this reason, we include a special out-of-vocabulary (OOV) token in the LSTM LM vocabulary, and we map all words from

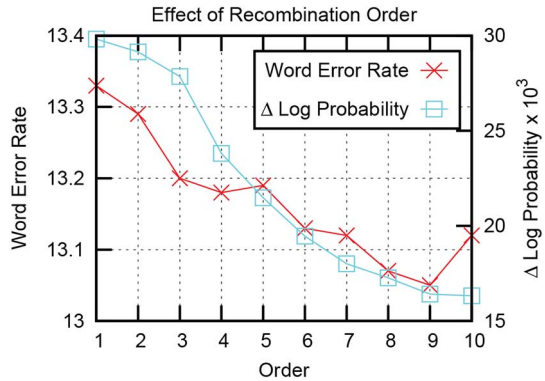

Fig. 5. Word error rates for different recombination orders, and corresponding negative log probability of the best path.

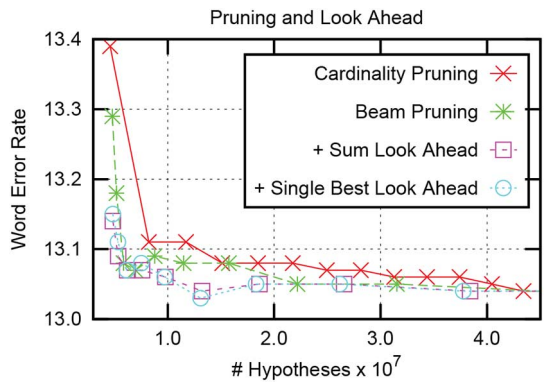

Fig. 6. Tradeoff between the number of hypotheses that is evaluated during search and the resulting word error rate, for various pruning and look ahead techniques.

the recognizer lexicon not observed in the training data to this token. By penalizing the OOV token as described in [17], we enforce that probability distributions are normalized, and perplexities can be compared among different kinds of LMs. By interpolating the large KN model and the LSTM, we obtain further improvements, resulting in a perplexity of 79.9 on the development data, and 94.4 on the test data. This interpolated model will be used throughout the French rescoring experiments.

We investigate the optimum order that is used for recombination pruning in combination with push-forward rescoring, where the beam pruning parameter is kept fixed at a large value. The corresponding results can be found in Fig. 5. It can be seen that the negative logarithmic probability of the best lattice rescoring hypothesis continuously decreases when increasing the recombination order, and the curve saturates at an order of about 9, which corresponds to a 10-gram recombination. Basically, the same tendency is reflected in the word error rate curve. For achieving the best performance, a 10-gram recombination is necessary, which is used in all subsequent experiments.

Fig. 6 depicts the effect of probability-based pruning techniques. With all of the pruning methods that are investigated, the optimum WER can be achieved. The main difference between the techniques lies in the size of the search space that needs to be considered to obtain a certain WER result. We can gradually decrease the search space size by using beam pruning instead of cardinality pruning, and further reductions are obtained by combining beam pruning with look ahead. (Look ahead does not have any effect on cardinality pruning, because it computes the

TABLE III
WORD ERROR RATE RESULTS ON THE FRENCH DEVELOPMENT DATA WITH A SINGLE-LAYER LSTM NEURAL NETWORK LM OF SIZE 300

	Push-Forward	Replacement Approx. V	Replacement Approx. CN	Replacement Approx. Density	Traceback Approx. V	Traceback Approx. CN	Traceback Approx. Density
Baseline	14.4		14.1	124			
100-best	13.4		13.4	104			
1000-best	13.2		13.0	1337			
Lattice 1	13.1	13.2	12.9	124	13.1	12.6	592
+ dependent	12.8	13.0	12.7		12.8	**12.5**	
Lattice 4	13.0	13.2	12.8	392	13.0	12.6	880
+ dependent	12.8	12.9	12.6		12.8	**12.5**	

TABLE IV
WORD ERROR RATE RESULTS ON THE FRENCH TEST DATA WITH A SINGLE-LAYER LSTM NEURAL NETWORK LM OF SIZE 300

	Push-Forward	Replacement Approx. V	Replacement Approx. CN	Replacement Approx. Density	Traceback Approx. V	Traceback Approx. CN	Traceback Approx. Density
Baseline	16.4		15.9	166			
100-best	14.8		14.7	106			
1000-best	14.7		14.5	1365			
Lattice 1	14.6	14.8	14.4	166	14.6	14.2	828
+ dependent	14.4	14.8	14.4		14.4	14.2	
Lattice 4	14.6	14.8	14.4	523	14.5	14.2	1209
+ dependent	14.4	14.7	14.3		14.4	**14.1**	

best successor for a given lattice node, and cardinality pruning only considers a single lattice node at a time. By contrast, look ahead can only pay off when hypotheses assigned to different lattice nodes are compared during the pruning process.) There is virtually no difference whether the sum over all future paths is computed, or whether only the single best path is taken into account for the look ahead. The search space size is directly proportional to the search effort, as the effort is dominated by the evaluation of the neural network LM equations. In summary, it is important to make use of advanced pruning techniques either if the search effort should be reduced to a minimum, or if the best possible WER needs to be obtained. In the former case, we can reduce the WER from 13.4% to 13.1%. In the latter case, we can reduce the search space size from 43 million to 13 million hypotheses. Then the real time factor for push-forward rescoring is about 0.5 on a single core of an Intel X5675 CPU.

In Tables III and IV, WER results are given on the French development and test data for different search space sizes and rescoring techniques. Here, the baseline refers to two pass speaker-adapted speech decoding with the large Kneser-Ney LM, which gives a WER of 14.4% on the development data. By creating a word lattice, and using confusion network (CN) rescoring on the lattice, the WER can be reduced to 14.1%. This lattice has a density of 124, where density is measured in lattice arcs per reference token. From the lattice, 100-best or 1000-best lists can be extracted. (We actually extract more than 100 or 1000 hypotheses to ensure that hypotheses are unique on the word sequence level.) The interpolation of the count LM and the LSTM LM is then used to rescore the n-best lists without approximations, leading to a reduction in WER of 1.0% absolute on the 100-best list, and 1.2% absolute on the 1000-best list. We can then apply CN rescoring on the n-best lists, but only small improvements of at most 0.2% absolute are observed. This is due to the fact that n-best lists encode relatively few hypotheses, and many of them even differ in a few word positions only. In comparison, a word lattice from the English setup can easily contain 10^{30} many hypotheses, while being considerably smaller than a 1000-best list. From Table III it can be seen that on average, the lattices from the French setup are smaller by more than a factor of 10 compared to the 1000-best lists.

We can apply the push-forward algorithm for rescoring word lattices, obtaining a WER of 13.1%. This 0.1% absolute gain is a very small improvement over the result on 1000-best lists, so the huge increase in search space size does not pay off in WER when the single best path is considered. For CN rescoring, multiple hypotheses are required, and we generate rescored lattices with the approximation techniques described in Section IV. For all lattice-based setups in this work, an oracle experiment was performed where the correct sentence was added to the lattice. This only led to minor improvements in word error rate. In this way, it was verified that the lattice density was sufficient for the experimental conditions. When using the replacement approximation, the density of the rescored lattices remains 124 as for the baseline. Then CN rescoring leads to a reduced WER of 12.9%. However, it can be observed that the Viterbi result of 13.2% on the rescored lattice is slightly worse than the 13.1% that we obtained by push-forward rescoring. This indicates that here the constraint of keeping the size of the original lattice negatively affects performance. As an alternative, we investigate the traceback approximation, where the LM contexts are dynamically expanded as needed. By construction, the Viterbi WER on the rescored lattice and the push-forward rescoring result are exactly the same. Furthermore, with CN rescoring the WER now is reduced to 12.6%, a notable gain of 0.5% absolute. At the same time, 1000-best lists are still larger by a factor of about two compared to the rescored lattices.

In all previously described rescoring experiments the utterances were considered independently of each other. Instead, we can initialize the LSTM state for the current utterance with the best LSTM state from the end of the previous utterance, as described in Section IV. Then rescoring needs to be performed sequentially, only the audio recordings can be processed in parallel. We mainly see improvements by such a dependent rescoring in the Viterbi WER, but after CN rescoring these improvements mostly vanish, therefore the increased latency resulting from sequential processing seems hardly justified.

The baseline lattices generated by the RWTH speech decoder ([52]) are not guaranteed to be expanded to a unique context of any order. Therefore, a unigram expansion needs to be assumed. We also analyzed the effect of a static expansion of the baseline lattices to a unique 4-gram context. As shown in Table III, this only gives tiny improvements over the unexpanded case, or even no change in WER at all. This behavior can be expected, because due to beam pruning, it does not make a difference whether the original lattice is expanded or not. An exception is that at each lattice node it is enforced that at least one hypothesis survives pruning, which accounts for the tiny differences observed by expansion.

Finally, Table IV summarizes the analogous results on the French test data. Basically, it can be seen that the WER reductions are very similar to those obtained on the development data, which suggests the proposed methods generalize well to

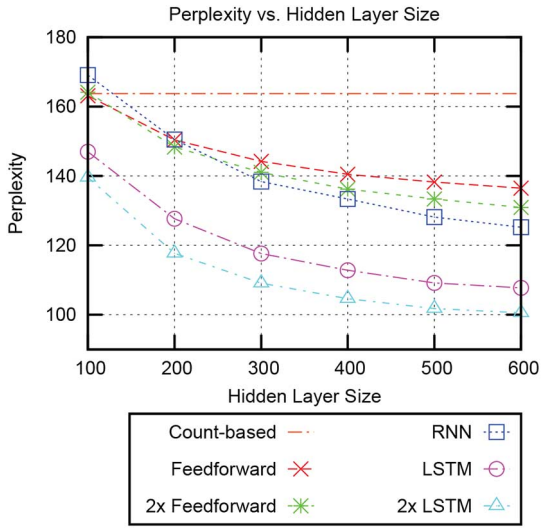

Fig. 7. Stand-alone perplexity on the English development data for different hidden layer configurations and neural network architectures. All models are estimated on the same amount of data. The vocabulary is restricted to words observed in the training data.

unseen data. We notice that for the independent rescoring of the expanded lattices there actually is a difference in the WER between the push-forward algorithm and the Viterbi rescoring of the traceback lattices (14.6% vs. 14.5%). This is only due to an additional LM scale, when using the same scales as for the push-forward rescoring case, both error rates are exactly the same.

In principle, as indicated in Figs. 5 and 6, most of the gain in WER can be obtained with small recombination orders and tight pruning parameters. However, as observed in Table III and IV, it seems necessary to take these settings to the limit to improve over simple n-best list rescoring, and for achieving substantial gains over n-best lists, lattices with CN rescoring are required.

B. Experimental Comparison of Different Language Models

On the English task, we investigate the performance of count-based LMs in relation to feedforward, recurrent, and LSTM neural network LMs. For all neural network variants, the output layer is factorized using 1000 word classes. The word classes were trained with the exchange algorithm from [33] based on a perplexity criterion with bigram dependences. In Fig. 7, the different types of LMs are compared in terms of perplexity on the English development data. For the sake of this comparison, we do not consider the full 150 K recognition vocabulary, but only the subset of words that occur in the 50 M word training data set, which amounts to 128 K words. We make use of a 4-gram count LM and a 10-gram feedforward neural network LM. For the feedforward network, the dimension of the word features is 300, which we found to give best performance in preliminary experiments. Thus, in total, the projection layer comprises 2700 units.

For all neural network variants we tie the layer sizes to the same value (except for the projection layer of the feedforward neural networks, which is kept constant). We find that increasing the hidden layer size of the feedforward neural network considerably reduces the perplexity from 163.1 to 136.5. Furthermore, we can make the feedforward neural network deeper by adding another hidden layer. In comparison with a single-layer feedforward network, the perplexity is always improved by the second hidden layer. However, the perplexity reductions achieved are not very large. When switching to a recurrent neural network, we observe further gains: The RNN with a single hidden layer of size 600 obtains a perplexity of 125.2, whereas the perplexity of a feedforward network with even two hidden layers of size 600 is still 130.9. Here, the recurrent model is obtained with a modified version of `rnnlm` from [45], which allows RNN training with perplexity-based word classes. Following recommendations given in [45], we make use of a '`-bptt`' value of 6, and we train the RNN in block mode with '`-bptt -block`' set to 10 and '`-min-improvement`' set to 1. By replacing the RNN with an LSTM, we can still obtain perplexities that are much lower. E.g., the perplexity of the LSTM with one hidden layer of size 600 is 107.8, which corresponds to a relative improvement of 13.9% over the RNN. By adding a second LSTM layer, we find that perplexities still drop further to a value of 100.5. We did not investigate two-layer RNNs as `rnnlm` does not support RNN architectures with multiple hidden layers. In principle, such networks could be trained with `rwthlm`[2] from [41]. However, this toolkit implements epochwise BPTT, which computes gradients over full sequences and thus requires neural network architectures that are robust with respect to the vanishing and exploding gradient problem, like LSTM networks. As an outlook, we also trained a feedforward neural network on 500 million running words with 2 hidden layers comprising 800 hidden units each. In training, tens of billions of words were processed. In spite of the increase in training data and the larger hidden layer configuration, the perplexity of this feedforward neural network was 109.8 on the development data. According to Fig. 7, this is still significantly higher than the best LSTM perplexity of 100.5 that was obtained on 50 million running words only.

In our experiments, we also confirm the finding from [46] that during neural network training, it is important to process in-domain data towards the end of an epoch. As a result, any of the neural network architectures significantly outperforms a Kneser-Ney smoothed count LM. For this comparison, individual count LMs are trained on each data source, and the resulting models are then linearly interpolated such that the development perplexity is minimized.

We can analyze the perplexity results in more detail by computing order-wise perplexities as introduced in [16]. The perplexity for the n-th order PPL_n is defined as

$$\mathrm{PPL}_n = \exp\left(-\frac{1}{|\mathcal{I}_n|}\sum_{i\in\mathcal{I}_n}\log p(w_i|w_1^{i-1})\right),$$

where we let

$$\mathcal{I}_n = \{i|N(w_{i-n+1}^i) > 0 \wedge N(w_{i-n}^i) = 0\}$$

for training data counts $N(\cdot)$. The words from the test data are partitioned according to the order of the n-gram hit from the count LM, and perplexities are computed on the corresponding subsets of n-gram events. We transfer the partitioning of word

[2]http://www-i6.informatik.rwth-aachen.de/web/Software/rwthlm.php

TABLE V
ORDER-WISE PERPLEXITIES ON THE ENGLISH TEST DATA FOR THE COUNT-BASED LM AND THREE NEURAL NETWORK ARCHITECTURES WITH A SINGLE HIDDEN LAYER OF SIZE 600

Order	#Events	Perplexity			
		Count-based	Feedforward	RNN	LSTM
4	10 K	16.2	21.9	20.7	17.0
3	13 K	88.0	78.5	73.5	63.5
2	11 K	609.3	382.8	349.2	302.3
1	3 K	49234.2	18929.6	17980.5	14405.1
Total	36 K	161.4	135.7	126.5	107.2

positions on the test data from the count LM to the neural network variants (even though these may not rely on any kind of order information). In Table V these results are summarized. It can be seen that the count LM obtains very low perplexities in case of a 4-gram hit, but for lower orders, these perplexities increase strongly. The same holds true for all neural network LMs as well. On the other hand, we observe that neural networks obtain better perplexities than the count LM for all orders except the highest. (Only an LSTM with two hidden layers of 600 units slightly outperforms the count LM on the highest order, from 16.2 to 15.8.) This is due to the fact that for the highest order, the count LM probability estimates are very close to the relative frequencies. For lower orders, the count LM estimate does not take into account one or more of the history words, and the ability of a neural network to exploit this history word information results in better perplexities. For example, when a 4-gram count LM backs off to a unigram, the three most recent history words are ignored for probability estimation. By contrast, a feedforward 10-gram LM can still map all of the nine history words to a continuous space and take them into consideration for estimating probabilities, regardless of whether the 10-gram was observed in the training data or not.

At this point, the number of neural network parameters is of importance, which differs for the individual neural network architectures. The relation between the number of parameters and the hidden layer size is depicted in Fig. 8. Conceptually, the number of parameters grows quadratically with the hidden layer size, for all types of neural networks, with the exception of single-layer feedforward networks, where the dependence is only linear. However, we see that the growth is quasi-linear for the range of hidden layer sizes investigated. The reason is that the number of parameters is strongly influenced by the number of vocabulary words. Let H be the hidden layer size, and V be the vocabulary size. Then for recurrent neural networks we roughly have $2HV$ many parameters, and for feedforward variants there are about HV many parameters, due to the tying of the projection layer parameters over all the history words. In particular, LSTM networks use more parameters for the hidden connections than standard RNNs, but this is masked by the input and output dimensions of the network. On the other hand, only a small fraction of the parameters at the input and output layer are actually taken into consideration when processing a single word. For example, in the English corpus, the number of words per class on average is 140, so the effort for computing the class posterior probability is much higher than that for the word posterior probability of the output layer. Therefore, the bottom of Fig. 8 also shows the number of parameters that are accessed

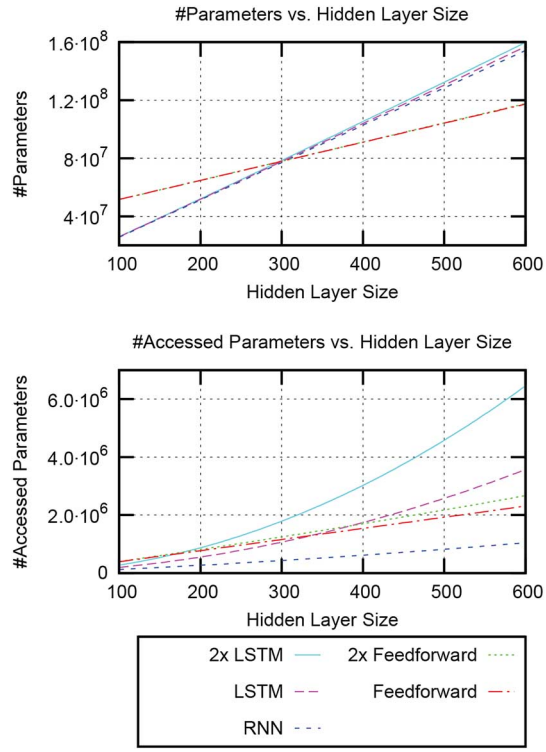

Fig. 8. Comparison of the total number of parameters and average number of accessed parameters for different neural network configurations. A corresponding 4-gram count model comprises 63 M n-grams.

on average when computing word probabilities with any of the neural network architectures. The resulting numbers give an indication about the processing speed of the neural networks. However, there is no direct dependence between the number of accessed parameters and the corresponding performance in terms of perplexity.

In Table VI, results are given on the English test data, in terms of perplexity and word error rate. For comparison, we also include character error rates (CER). In all cases, we interpolate a neural network LM with the large count LM trained on 3.1 B words. Here, we did not investigate the performance of neural networks without interpolating the count LM: In rescoring, the search space is initially generated with the count LM, therefore it is hard to clearly separate the influence of the count model and the neural network on the WER level. We generate rescored lattices with the traceback approximation for CN rescoring. Utterances are considered independently of each other, with the exception of standard RNNs: For the RNN results, we train the networks with `rnnlm` and then convert them to `rwthlm` format. In this way, we can evaluate all neural networks using the same rescoring technique. However, as `rnnlm` training is dependent, rescoring with standard RNNs has to be performed in a dependent way, too. Overall, we see that any neural network LM gives substantial improvements in PPL and WER. Feedforward models fall behind the performance of recurrent variants. Furthermore, adding another hidden layer clearly reduces the WER. As single-layer networks may still improve by increasing its hidden layer further, it is not obvious that a deeper model is required to obtain optimum performance. Nevertheless, for the ranges of hidden layer sizes investigated here, we can conclude

TABLE VI
PERFORMANCE OF DIFFERENT TYPES OF LANGUAGE MODELS ON THE ENGLISH TEST DATA. NEURAL NETWORK LMS ARE ALWAYS INTERPOLATED WITH THE LARGE COUNT LM

LM	Hidden Layers	PPL	CER	WER
Count-based	–	131.2	7.6	12.4
+ Feedforward	100	121.1	7.5	11.8
	200	116.6	7.3	11.6
	300	114.7	7.3	11.5
	400	114.1	7.2	11.5
	500	113.4	7.2	11.5
	600	112.5	7.2	11.5
	2x 100	121.2	7.5	11.9
	2x 200	115.7	7.3	11.5
	2x 300	115.4	7.2	11.5
	2x 400	112.2	7.1	11.3
	2x 500	111.0	7.1	11.3
	2x 600	110.2	7.2	11.3
+ RNN	100	121.0	7.5	11.8
	200	117.6	7.3	11.7
	300	112.6	7.3	11.4
	400	111.5	7.2	11.3
	500	108.9	7.1	11.2
	600	108.1	7.0	11.1
+ LSTM	100	115.3	7.3	11.7
	200	106.8	7.1	11.2
	300	102.4	6.9	11.0
	400	99.9	6.9	10.9
	500	97.9	6.9	10.9
	600	96.7	6.8	10.8
	2x 100	111.0	7.2	11.4
	2x 200	101.6	7.0	11.0
	2x 300	97.5	6.8	10.8
	2x 400	95.2	6.9	10.8
	2x 500	93.1	**6.6**	10.5
	2x 600	**92.0**	6.7	**10.4**

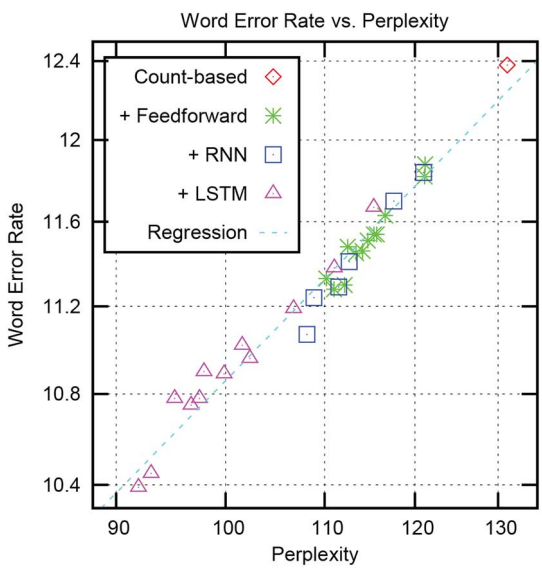

Fig. 9. Experimental analysis of the relationship between perplexity and word error rate on the English test data.

that deep architectures are beneficial. It is especially remarkable that LSTM networks benefit the most from an additional layer in our experiments, because a single-layer LSTM network can already be considered deep, as it corresponds to a deep feedforward network when unfolding it over time for training.

In Fig. 9, the word error rate on the test data is depicted as a function of the perplexity. Both axes are scaled logarithmically as suggested in [53]. By fitting a polynomial to the data, we find that the experimentally observed data points can be represented quite accurately by the function

$$\mathrm{WER}(\mathrm{PPL}) = 1.42 \cdot (\mathrm{PPL})^{0.44}.$$

We thereby confirm the strong correlation between the two quantities. In particular, we observe that this functional dependence between PPL and WER seems to hold independently of the type of LM that is used, and while [53] analyzed count-based LM variants only, the findings can also be transferred to the case of neural network LMs. In language modeling, the correlation between PPL and WER can thus be considered an advantage in comparison to acoustic modeling, where it is more difficult to find a relation between the word error rate and conventional cross validation criteria, like the frame error rate. Finally, we note that our experimental results indicate that, unlike standard RNNs, LSTM networks can be trained by computing exact gradients, backpropagating error information over long sequences without any kind of truncation or clipping of gradient values.

VI. CONCLUSION

First, in this paper advanced lattice-based rescoring algorithms for neural network LMs were investigated. We only obtained minor improvements for Viterbi rescoring by increasing the search space of the neural network LM from n-best lists to lattices. However, lattice-based rescoring techniques led to notable gains when combined with confusion network rescoring. Our rescoring approaches were also applied in a comparative study of language modeling techniques, for which we analyzed the performance of count LMs in relation to neural network LM variants. In summary, there seems to be a hierarchy of neural network architectures: Feedforward neural networks give considerable improvements when interpolated with count LMs. However, they are outperformed by recurrent neural networks, that show additional reductions in perplexity and word error rate over feedforward networks. RNNs in turn are outperformed by LSTMs, e.g., on the English development data, we see an additional reduction in perplexity by 14% relative. Furthermore, one of the key contributions of this paper is to show that deep architectures not only help for acoustic modeling but are also beneficial for the performance of neural network language models. Even though shallow LSTM networks already perform better than shallow feedforward networks in language modeling, LSTMs still show more consistent improvements when adding additional hidden layers than their feedforward counterparts. With a single two-layer LSTM, we were able to improve the count LM baseline word error rate from 12.4% to 10.4%, while the count LM was trained on 60 times more data, where the additional data proved relevant for the target domain. Finally, we observe that in our experimental analysis, perplexity improvements are an accurate predictor for word error rate reductions, and our results for neural networks are in line with previous work on count-based LMs.

For future work, it seems promising to investigate even larger hidden layer configurations and deeper neural network architectures. This may still give additional improvements, and it would be interesting to see the implications for the experimental comparison presented here, when no limitations on computational resources and training times are given.

ACKNOWLEDGMENT

The authors would like to thank the reviewers for their valuable comments.

REFERENCES

[1] R. Rosenfeld, "Two decades of statistical language modeling: Where do we go from here?," *Proc. IEEE*, vol. 88, no. 8, pp. 359–394, Aug. 2000.
[2] S. M. Katz, "Estimation of probabilities from sparse data for the language model component of a speech recognizer," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 35, no. 3, pp. 400–401, Mar. 1987.
[3] R. Kneser and H. Ney, "Improved backing-off for M-Gram language modeling," in *Proc. ICASSP*, 1995, pp. 181–184.
[4] S. F. Chen and J. Goodman, "An empirical study of smoothing techniques for language modeling," *Comput. Speech Lang.*, vol. 13, no. 4, pp. 359–393, 1999.
[5] H. Ney, S. Martin, F. Wessel, S. Young, and G. Bloothooft, "Statistical language modeling using leaving-one-out," in *Corpus-Based Methods in Language And Speech Processing*. Norwell, MA, USA: Kluwer, 1997, ch. 6, pp. 174–207.
[6] Y. Bengio and R. Ducharme, "A neural probabilistic language model," in *Proc. NIPS*, 2000, vol. 13, pp. 933–938.
[7] H. Schwenk, "Continuous space language models," *Comput. Speech Lang.*, vol. 21, pp. 492–518, 2007.
[8] H.-S. Le, I. Oparin, A. Allauzen, J.-L. Gauvain, and F. Yvon, "Structured output layer neural network language models for speech recognition," *IEEE Trans. Audio, Speech, Lang. Process.*, vol. 21, no. 1, pp. 197–206, Jan. 2013.
[9] T. Mikolov, M. Karafiát, L. Burget, J. Černocký, and S. Khudanpur, "Recurrent neural network based language model," in *Proc. Interspeech*, 2010, pp. 1045–1048.
[10] M. Sundermeyer, R. Schlüter, and H. Ney, "LSTM neural networks for language modeling," in *Proc. Interspeech*, 2012.
[11] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury, "Deep neural networks for acoustic modeling in speech recognition—the shared views of four research groups," *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 82–97, Nov. 2012.
[12] H. Ney and S. Ortmanns, "Dynamic programming search for continuous speech recognition," *IEEE Signal Process. Mag.*, vol. 16, no. 5, pp. 64–83, Sep. 1999.
[13] M. Sundermeyer, T. Alkhouli, J. Wuebker, and H. Ney, "Translation modeling with bidirectional recurrent neural networks," in *Proc. EMNLP*, 2014, pp. 14–25.
[14] T. Mikolov, S. Kombrink, L. Burget, J. Černocký, and S. Khudanpur, "Extensions of recurrent neural network language model," in *Proc. ICASSP*, 2011, pp. 5528–5531.
[15] E. Ar soy, T. N. Sainath, B. Kingsbury, and B. Ramabhadran, "Deep neural network language models," in *Proc. NAACL-HLT Workshop*, 2012, pp. 20–28.
[16] I. Oparin, M. Sundermeyer, H. Ney, and J.-L. Gauvain, "Performance analysis of neural networks in combination with n-gram language models," in *Proc. ICASSP*, 2012, pp. 5005–5008.
[17] M. Sundermeyer, I. Oparin, J.-L. Gauvain, B. Freiberg, R. Schlüter, and H. Ney, "Comparison of feedforward and recurrent neural network language models," in *Proc. ICASSP*, 2013, pp. 8430–8434.
[18] H. S. Le, A. Allauzen, and F. Yvon, "Measuring the influence of long range dependencies with neural network language models," in *Proc. NAACL-HLT Workshop*, 2012, pp. 1–10.
[19] S. Kombrink, T. Mikolov, M. Karafiát, and L. Burget, "Recurrent neural network based language modeling in meeting recognition," in *Proc. Interspeech*, 2011, pp. 2877–2880.
[20] Y. Si, Q. Zhang, T. Li, J. Pan, and A. Yan, "Prefix tree based N-best list re-scoring for recurrent neural network language model used in speech recognition system," in *Proc. Interspeech*, 2013, pp. 3419–3423.
[21] C. Chelba and F. Jelinek, "Recognition performance of a structured language model," in *Proc. Eurospeech*, 1999, vol. 4, pp. 1567–1570.
[22] A. Deoras, T. Mikolov, and K. Church, "A fast re-scoring strategy to capture long-distance dependencies," in *Proc. EMLNP*, 2011, pp. 1116–1127.
[23] M. Auli, M. Galley, C. Quirk, and G. Zweig, "Joint language and translation modeling with recurrent neural networks," in *Proc. EMNLP*, 2013, pp. 1044–1054.
[24] X. Liu, Y. Wang, X. Chen, M. J. F. Gales, and P. C. Woodland, "Efficient lattice rescoring using recurrent neural network language models," in *Proc. ICASSP*, 2014, pp. 4941–4945.
[25] M. Sundermeyer, Z. Tüske, R. Schlüter, and H. Ney, "Lattice decoding and rescoring with long-span neural network language models," in *Proc. Interspeech*, 2014, pp. 661–665.
[26] X. Liu, M. J. F. Gales, and P. C. Woodland, "Use of contexts in language model interpolation and adaptation," *Comput. Speech Lang.*, vol. 27, no. 1, pp. 301–321, 2013.
[27] L. Mangu, E. Brill, and A. Stolcke, "Finding consensus in speech recognition: Word error minimization and other applications of confusion networks," *Comput. Speech Lang.*, vol. 14, no. 4, pp. 373–400, 2000.
[28] Z. Huang, G. Zweig, and B. Dumoulin, "Cache based recurrent neural network language model inference for first pass speech recognition," in *Proc. ICASSP*, 2015, pp. 6404–6408.
[29] T. Hori, Y. Kubo, and A. Nakamura, "Real-time one-pass decoding with recurrent neural network language model for speech recognition," in *Proc. ICASSP*, 2015, pp. 6414–6418.
[30] C. M. Bishop, "Single-layer networks," in *Neural Networks for Pattern Recognition*. Oxford, U.K.: Oxford Univ. Press, 1995, ch. 3, pp. 77–115.
[31] J. Goodman, "Classes for fast maximum entropy training," in *Proc. ICASSP*, 2001, pp. 561–564.
[32] F. Morin and Y. Bengio, "Hierarchical probabilistic neural network language model," in *Proc. 10th Int. Workshop Artif. Intell. Statist.*, 2005, pp. 246–252.
[33] R. Kneser and H. Ney, "Forming word classes by statistical clustering for statistical language modelling," in *Proc. QUALICO*, 1991, pp. 221–226.
[34] P. F. Brown, P. V. deSouza, R. L. Mercer, V. J. Della Pietra, and J. C. Lai, "Class-based n-gram models of natural language," *Comput. Linguist.*, vol. 18, no. 4, pp. 467–479, 1992.
[35] G. Zweig and K. Makarychev, "Speed regularization and optimality in word classing," in *Proc. ICASSP*, 2013, pp. 8237–8241.
[36] S. Martin, J. Liermann, and H. Ney, "Algorithms for bigram and trigram word clustering," *Speech Commun.*, vol. 24, no. 1, pp. 19–37, 1998.
[37] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Trans. Neural Netw.*, vol. 5, no. 2, pp. 157–166, Mar. 1994.
[38] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.
[39] F. Gers, "Learning to forget: Continual prediction with LSTM," in *Proc. Int. Conf. Artif. Neural Netw.*, 1999, pp. 850–855.
[40] F. A. Gers, N. N. Schraudolph, and J. Schmidhuber, "Learning precise timing with LSTM recurrent networks," *J. Mach. Learn. Res.*, vol. 3, pp. 115–143, 2002.
[41] M. Sundermeyer, R. Schlüter, and H. Ney, "RWTHLM–The RWTH Aachen University neural network language modeling toolkit," in *Proc. Interspeech*, 2014, pp. 2093–2097.
[42] L. Bottou, G. Montavon, G. B. Orr, and K.-R. Müller, "Stochastic gradient descent tricks," in *Neural Networks: Tricks of the Trade*, 2nd Ed. ed. New York, NY, USA: Springer, 2012, ch. 18, pp. 421–436.
[43] D. E. Rumelhart, G. E. Hinton, R. J. Williams, J. L. McClelland, and D. E. Rumelhart, "Learning internal representations by error propagation," in *the PDP Research Group, Parallel Distributed Processing*. Cambridge, MA, USA: MIT Press, 1986, pp. 318–362.
[44] R. J. Williams, D. Zipser, Y. Chauvain, and D. E. Rumelhart, "Gradient-based learning algorithms for recurrent networks and their computational complexity," in *Backpropagation: Theory, Architectures, and Applications*. Hove, U.K.: Psychology Press, 1995, pp. 433–486.
[45] T. Mikolov, S. Kombrink, A. Deoras, L. Burget, and J. Černocký, "RNNLM–recurrent neural network language modeling toolkit," in *Proc. ASRU*, 2011, pp. 196–201.
[46] T. Mikolov, A. Deoras, D. Povey, L. Burget, and J. Cernocky, "Strategies for training large scale neural network language models," in *Proc. ASRU*, 2011, pp. 196–201.
[47] F. Weng, A. Stolcke, and A. Sankar, "Efficient lattice representation and generation," in *Proc. ICSP*, 1998.
[48] H. Ney and S. Ortmanns, "Progress in dynamic programming search for LVCSR," *Proc. IEEE*, vol. 88, no. 8, pp. 1224–1240, Aug. 2000.
[49] M. Sundermeyer, M. Nuß baum-Thom, S. Wiesler, C. Plahl, A. El-Desoky Mousa, S. Hahn, D. Nolden, R. Schlüter, and H. Ney, "The RWTH 2010 QUAERO ASR evaluation system for English, French, and German," in *Proc. ICASSP*, 2011, pp. 2212–2215.
[50] Z. Tüske, R. Schlüter, and H. Ney, "Multilingual hierarchical MRASTA features for ASR," in *Proc. Interspeech*, 2013, pp. 2222–2226.

[51] H. Hermansky, D. P. Ellis, and S. Sharma, "Tandem connectionist feature extraction for conventional HMM systems," in *Proc. ICASSP*, 2000, pp. 1635–1638.
[52] D. Rybach, S. Hahn, P. Lehnen, D. Nolden, M. Sundermeyer, Z. Tüske, S. Wiesler, R. Schlüter, and H. Ney, "RASR–the RWTH Aachen University open source speech recognition toolkit," in *Proc. ASRU*, 2011.
[53] D. Klakow and J. Peters, "Testing the correlation of word error rate and perplexity," in *Speech Commun.*, 2002, vol. 38, no. 1, pp. 19–28.

Martin Sundermeyer studied computer science at RWTH Aachen University, Germany, and ENST Paris, France. He received the Diplom degree from RWTH Aachen University in 2008. Since then he has been with the Human Language Technology and Pattern Recognition group at RWTH Aachen University, where he is working as a Research Assistant and Ph.D. student. His research interests include automatic speech recognition, language modeling, and statistical machine translation.

Hermann Ney received a master degree in physics in 1977 from the University of Goettingen, Germany, and a Dr.-Ing. degree in electrical engineering in 1982 from the Braunschweig University of Technology, Braunschweig, Germany. From 1977–1993, he was with Philips Research Laboratories, Hamburg and Aachen, Germany. From 1988–1989, he was a visiting scientist at ATT Bell Labs, Murray Hill, NJ. Since 1993, he has been a Professor of Computer Science at RWTH Aachen University in Aachen, Germany.

His research interests lie in the area of machine learning and human language technology including automatic speech recognition and machine translation of text and speech. In automatic speech recognition, he and his team worked on dynamic programming for large-vocabulary search, discriminative training and on language modelling. In machine translation, he and his team introduced the alignment tool GIZA++, the method of phrase-based translation, the use of dynamic programming based beam search for decoding, the log-linear model combination and system combination.

His work has resulted in more than 700 conference and journal papers with an H-index of 79 and 31000 citations (based on Google scholar). He is a fellow of both IEEE and ISCA (Int. Speech Communication Association). In 2005, he was the recipient of the Technical Achievement Award of the IEEE Signal Processing Society. In 2010, he was awarded a senior DIGITEO chair at LIMIS/CNRS in Paris, France. In 2012–2013, he was a Distinguished Lecturer of ISCA. In 2013, he received the IAMT award of honour (IAMT: Int. Association of Machine Translation).

Ralf Schlüter studied physics at RWTH Aachen University, Germany, and Edinburgh University, UK. He received the Dipl. degree with honors in physics in 1995 and the Dr.rer.nat. degree with honors in computer science in 2000, from RWTH Aachen University. From November 1995 to April 1996, he was with the Institute for Theoretical Physics B at RWTH Aachen, where he worked on statistical physics and stochastic simulation techniques. Since May 1996 he has been with the Computer Science Department at RWTH Aachen University, where he currently is Academic Director and leads the automatic speech recognition group at the Human Language Technology and Pattern Recognition chair. His research interests cover speech recognition, discriminative training, decision theory, stochastic modeling, and signal analysis.