A Survey on the Application of Recurrent Neural Networks to Statistical Language Modeling

Article *in* Computer Speech & Language · January 2014

DOI: 10.1016/j.csl.2014.09.005

3 authors, including:

Wim de mulder
Ghent University
17 PUBLICATIONS **74** CITATIONS
SEE PROFILE

Marie-Francine Moens
KU Leuven
298 PUBLICATIONS **3,573** CITATIONS
SEE PROFILE

Available online at www.sciencedirect.com

ScienceDirect

Computer Speech and Language 30 (2015) 61–98

COMPUTER SPEECH AND LANGUAGE

www.elsevier.com/locate/csl

Review

A survey on the application of recurrent neural networks to statistical language modeling☆

Wim De Mulder[a,*], Steven Bethard[b], Marie-Francine Moens[a,**]

[a] *Katholieke Universiteit Leuven, Celestijnenlaan 200A, B-3001 Heverlee, Belgium*
[b] *University of Alabama at Birmingham, Birmingham, AL 35294, USA*

Received 7 September 2013; received in revised form 10 July 2014; accepted 17 September 2014
Available online 28 September 2014

Abstract

In this paper, we present a survey on the application of recurrent neural networks to the task of statistical language modeling. Although it has been shown that these models obtain good performance on this task, often superior to other state-of-the-art techniques, they suffer from some important drawbacks, including a very long training time and limitations on the number of context words that can be taken into account in practice. Recent extensions to recurrent neural network models have been developed in an attempt to address these drawbacks. This paper gives an overview of the most important extensions. Each technique is described and its performance on statistical language modeling, as described in the existing literature, is discussed. Our structured overview makes it possible to detect the most promising techniques in the field of recurrent neural networks, applied to language modeling, but it also highlights the techniques for which further research is required.

MSC: 68T05; 68T10; 68T50

Keywords: Recurrent neural networks; Natural language processing; Language modeling; Speech recognition; Machine translation

Contents

☆ This paper has been recommended for acceptance by E. Briscoe.

* Corresponding author.
** Corresponding author. Tel.: + 32 16 32 53 83; fax: +32 16 32 79 96.
E-mail addresses: wim.demulder@cs.kuleuven.be (W. De Mulder), bethard@cis.uab.edu (S. Bethard), sien.moens@cs.kuleuven.be, Marie-Francine.moens@cs.kuleuven.be (M.-F. Moens).

http://dx.doi.org/10.1016/j.csl.2014.09.005

1. Introduction

1.1. Statistical language modeling

Statistical language modeling (SLM) amounts to estimating the probability distribution of various linguistic units, such as words, sentences, and whole documents (Rosenfeld, 2000). Applications of statistical language modeling include, but are not limited to, speech recognition (Jelinek, 1998; Schwenk, 2010), spelling correction (Ahmed et al., 2009), text generation (de Novais et al., 2010), machine translation (Brown et al., 1993; Och and Ney, 2002; Kirchhoff and Yang, 2005), syntactic (Huang et al., 2014) and semantic processing (Deschacht et al., 2012), optical character recognition and handwriting recognition (Vinciarelli et al., 2004).

A traditional task in SLM is to model the probability that a given word appears next after a given sequence of words. From a purely linguistic point of view this task is ill-defined, since the term 'word' has no unique meaning (Manning and Schuetze, 1999). In fact there are at least four nonequivalent, definitions of this concept.[1] Nevertheless, the theoretical debate about the term 'word' has not worried machine learning researchers, who have developed practical models to deal with language. *N*-gram models were among the earliest techniques to model the probability of observing a given word after some previous words (Bahl et al., 1983; Jelinek, 1998; Church, 1988). The N in N-gram refers to the number of considered previous words plus the next word in the sequence. $N-1$ is the context length, that is the number of words that the model takes into account to estimate the probability of the next word. The estimations by N-gram models result from word co-occurrence frequencies. Basically, the probability that a certain word appears next after a given sequence of words is estimated as the number of times that the given sequence augmented with the given word appears in the training data divided by the number of times that the given sequence is present in the training data (the training data is typically a large amount of plain text in this case). In practice, the probability distributions are smoothed by assigning non-zero probabilities to words that are not present in the training data. One reason for smoothing is to compensate the very small fraction of all proper names that are mentioned in any given training data set (Kneser and Ney, 1995; Chelba et al., 2010; Moore, 2009).

Soon more advanced language models were developed, including models based on decision trees (Potamianosa and Jelinek, 1998; Heeman, 1999) and maximum entropy based techniques (Rosenfeld, 1994; Peters and Klakow, 1999; Wang et al., 2005). These models allowed the incorporation of various features (e.g., part of speech tags, syntactic structure) into the language models, rather than having to rely on the words alone. Then with the widely cited work of Bengio et al. (2003), artificial neural networks found their way into the domain of SLM. Bengio et al. (2003) applied a feedforward neural network (FNN) to a training set consisting of a sequence of words, showing how the neural model

[1] http://www.sussex.ac.uk/english/documents/essay—what-is-a-word.pdf.

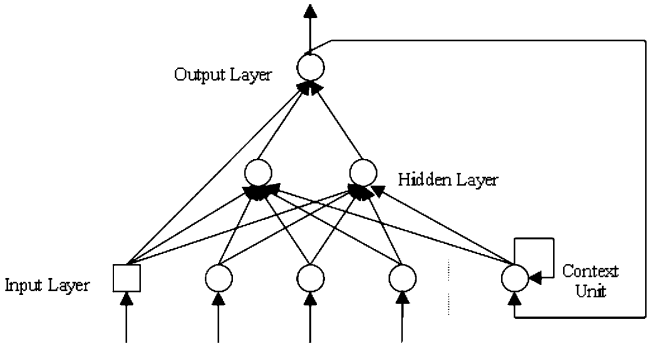

Fig. 1. Jordan neural network.
Taken from Balkin (1997).

could simultaneously learn the probability that a certain word appears next after a given sequence of words *and* at the same time learn a real valued vector representation for every word in a predefined vocabulary. Thus the FNN model learned an appropriate set of features while it was learning how to predict the next word in a sentence. In contrast, the decision tree and maximum entropy language models typically required the features to be manually engineered before any model could be trained (Heeman, 1999; Peters and Klakow, 1999).

It is this ability to infer a real valued vector for each word in a vocabulary that has driven much of the recent interest in using neural networks for language modeling. Recent research suggests that such vector representations carry important linguistic information. That is, the vector representation is not just a placeholder for the corresponding word, but the different components encode useful pieces of meaning (Turian et al., 2010). For example, Collobert et al. (2011) show that using a single set of vector representations, neural networks perform well on several natural language processing tasks in the absence of any other features. Mikolov et al. (2013, 2013) show that the vector representations are at least partially compositional, for example, given the vector for *King*, if you subtract the vector for *Man* and add the vector for *Woman*, you end up with a vector similar to that of *Queen*. Recent work by Chen et al. (2013) demonstrates that the vector representations also capture relations like synonymy, antonymy and regional spelling variations.

Stated in a more general way, neural networks have the capability to project the vocabulary into hidden layers, so that semantically similar words are clustered. This explains why neural networks can provide better estimates for *N*-grams which have never been seen during training, offering an answer to the problem of data sparseness (Bengio et al., 2003; Deoras et al., 2011; Kombrink et al., 2011).

1.2. Recurrent neural networks

The FNNs of Bengio et al. (2003) have an important drawback: only a fixed number of previous words can be taken into account to predict the next word. This limitation is inherent to the structure of FNNs, since they lack any form of 'memory': only the words that are presented via the fixed number of input neurons can be used to predict the next word and all words that were presented during earlier iterations are 'forgotten', although these words can be essential to determine the context and thus to determine a suitable next word.

Ingenious solutions were proposed to introduce some memory in the FNN architecture to overcome the limited context length. Noteworthy variants of the original FNN are the Jordan (1986) and Elman (1990) networks where extra neurons are incorporated that are connected to the hidden layer like the other input neurons. These extra neurons are called context neurons and hold the contents of one of the layers as it existed when the previous pattern was trained. This model allows a sort of short term memory. Training can still be done in the same way as for FNNs. Fig. 1 is a graphical example of a Jordan network, showing the network output fed back into the context unit, which in turn sends it to the hidden layer in the next iteration. In an Elman network a context neuron is fed by a hidden layer.

The context length was extended to indefinite, one could even say infinite, size by using a recurrent version of neural networks, conveniently called recurrent neural networks (RNN), which can handle arbitrary context lengths. Initial enthusiasm about RNN as statistical language modelers, mainly driven by their abilities to learn vector representations for words (as in FNN) and to handle arbitrarily long contexts, in addition to the fact that they are universal approximators (Schäfer and Zimmerman, 2007), was quickly tempered by their extremely slow learning and by the observation that

the theoretical incorporation of arbitrarily long context lengths does not manifest itself in practice. Consequently, many variations on the original RNN have been developed to cope with these limitations and, at the same time, to specialize their structure towards SLM (Kombrink et al., 2011).

This survey presents the main RNN architectures that resulted from the attempts to turn the original, theoretically well founded RNN with its limited practical usefulness into more practically oriented structures, where researchers were willing to exchange the theoretical soundness with heuristic reasoning (at least to some degree), but without giving up the aforementioned strengths. Each RNN architecture is compared to the original structure, and empirical evaluations, as done by researchers in the field, are presented.

2. Outline of the paper

Section 3 describes the most commonly used measures to evaluate statistical language models and outlines some important deficiencies of these measures. In Section 4, we give a general description of basic recurrent neural networks, i.e. the RNNs as they were originally developed. The description is general in the sense that it is independent of the intended task, although we will restrict attention to supervised RNNs, since supervised machine learning techniques are much more prevalent in SLM than their unsupervised counterparts. In Section 5, we provide an in-depth account on the application of basic RNN on the task of modeling the probability of observing a word after a given sequence of words. It is shown how basic RNNs can be applied to perform this task and how these models perform in comparison to other models. This section ends with three main limitations of a basic RNN: long training times, the need to choose a fixed number of hidden neurons in advance, and the observation that in practical applications the context length that is taken into account is small. Sections 6–8 discuss the most important methods that have been developed to overcome these limitations. Although the paper focuses on the training of recurrent neural networks in the context of language modeling, Section 9 describes some important approaches for decoding when using RNN based language models. In Section 10 we discuss two extensions of the basic RNN that came into existence to further increase the performance of RNN. Finally, Section 11 concludes the paper and provides some further research directions.

3. Evaluation of language models

The most commonly used measures to evaluate language models are perplexity (PPL) and, for speech recognition systems, word error rate (WER) (Jelinek, 1998). The PPL measure is the inverse of the geometric average probability assigned by the model to each word in a test data set, given some sequence of previous words. The popularity of this measure for evaluating language models is due to the fact that it allows an easy comparison between different models. WER directly measures the quality of the speech recognition system, by counting the number of mistakes between the output of the system and the reference transcription which is provided by a human annotator.

As these measures are widely used, they will pop up frequently in the evaluation of RNNs below. However, one should be aware that these measures have been widely criticized. For example, numerous examples are known where language models provide a large improvement in perplexity over some baseline model, but with no or little improvement in the word error rate (Clarkson and Robinson, 1998; Iyer et al., 1997; Martin et al., 1997). Furthermore, comparisons between different models in terms of perplexity requires great care, since the perplexity depends not only on the language model but also on the training data and on the underlying vocabulary. Consequently, comparisons are only meaningful when the same data and the same vocabulary are used, a fact which is often overlooked. The WER, on the other hand, puts an overemphasis on frequent, but uninformative words, and some techniques can yield large WER improvements when applied to simple systems, while showing no improvement at all for more complex systems. Comparison of relative WER reductions when applying different techniques to different systems is practically useless (Mikolov, 2012), again a crucial aspect that is all too often not taken into account by researchers in the field.

These critiques have motivated researchers to develop new evaluation measures. In Clarkson and Robinson (2001), the authors point out that the calculation of perplexity is based solely on the probabilities of words contained within the test text, thereby disregarding the probabilities of alternative words which will be competing with the correct word within the decoder. They show that by considering the probabilities of the alternative words it is possible to derive measures of language model quality which are better correlated with word error rate than perplexity is. It is argued that optimizing language model parameters with respect to these new measures leads to a significant reduction in the word error rate. Another attempt to extend perplexity to take more information into account is described in Chen et al. (1998).

Unfortunately, these more advanced evaluation measures haven't been picked up by language modelers yet, despite the fact that these measures compensate for some essential drawbacks of PPL and WER. While WER measures the dissimilarity between a system's output and the expected ground truth, the BLEU (Bilingual Evaluation Understudy) metric measures the similarity between a system's output and the ground truth (Papineni et al., 2002). The latter metric is widely used in machine translation. Another error metric is TER (Translation Error Rate) that measures the number of edits required to change a system's output into the ground truth output (Snover et al., 2006).

4. General description of basic recurrent neural networks

Recurrent neural networks are trained via a sequence of training examples $((\mathbf{x}(1), \mathbf{y}(1)), (\mathbf{x}(2), \mathbf{y}(2)), \ldots, (\mathbf{x}(m), \mathbf{y}(m)))$ with $\mathbf{x}(t) \in \mathbb{R}^{n_I}$, $\mathbf{y}(t) \in \mathbb{R}^{n_J}$ for $1 \leq t \leq m$. The vectors $\mathbf{x}(t)$ are given as inputs to the network, while the vectors $\mathbf{y}(t)$ denote the target output. A subscript will be used to refer to a specific component of a vector, e.g. $x_i(t)$ refers to the ith component of $\mathbf{x}(t)$. The network calculates output values $o_1(t), \ldots, o_{n_J}(t)$ in several steps:

$$a_j(t) = \sum_{i=1}^{n_I} \alpha_{ji}\, x_i(t) + \sum_{i=1}^{n_H} \rho_{ji}\, h_i(t-1), \quad j = 1, \ldots, n_H \tag{1}$$

$$h_j(t) = F(a_j(t)), \quad j = 1, \ldots, n_H \tag{2}$$

$$b_j(t) = \sum_{i=1}^{n_H} \beta_{ji}\, h_i(t), \quad j = 1, \ldots, n_J \tag{3}$$

$$o_j(t) = G(b_j(t)), \quad j = 1, \ldots, n_J \tag{4}$$

where α_{ji}, β_{ji}, ρ_{ji} are parameters, also called weights, to be learned by the system. It is customary to view such a network as consisting of neuron-like units, called neurons, that receive inputs, perform some transformation, and either send the result to other neurons or present the result as the output of the network. Especially noteworthy are the so-called hidden neurons that receive the input vector $\mathbf{x}(t)$, calculate a linear combination of the individual components, followed by a nonlinear transformation F and send the result to the so-called output neurons that will calculate the output values $o_j(t), j = 1, \ldots, n_J$. The functions F and G are (typically) nonlinear functions to be determined by the user. The function F is often chosen as the sigmoid:

$$F(x) = \frac{1}{1 + e^{-x}} \tag{5}$$

or as the hyperbolic tangent:

$$F(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

To speed up computation time, the tangent hyperbolic can be approximated by the *hard* tangent hyperbolic (Collobert et al., 2011):

$$\begin{aligned} F(x) &= -1 \quad \text{if} \quad x < -1 \\ &= x \quad \text{if} \quad -1 \leq x \leq 1 \\ &= 1 \quad \text{if} \quad x > 1 \end{aligned}$$

A popular choice for the output function G, especially in the context of statistical language modeling, is the softmax function:

$$G(b_j(t)) = \frac{e^{b_j(t)}}{\sum_{q=1}^{n_J} e^{b_q(t)}} \tag{6}$$

The number of hidden neurons n_H is determined in advance by the user and is fixed during training. Unfortunately, there does not exist any generally accepted rule of thumb, let alone a general theory. In Sheela and Deepa (2013), a review is presented on methods that heuristically determine the number of hidden neurons for neural networks. Some simple heuristics are described in Panchal et al. (2011):

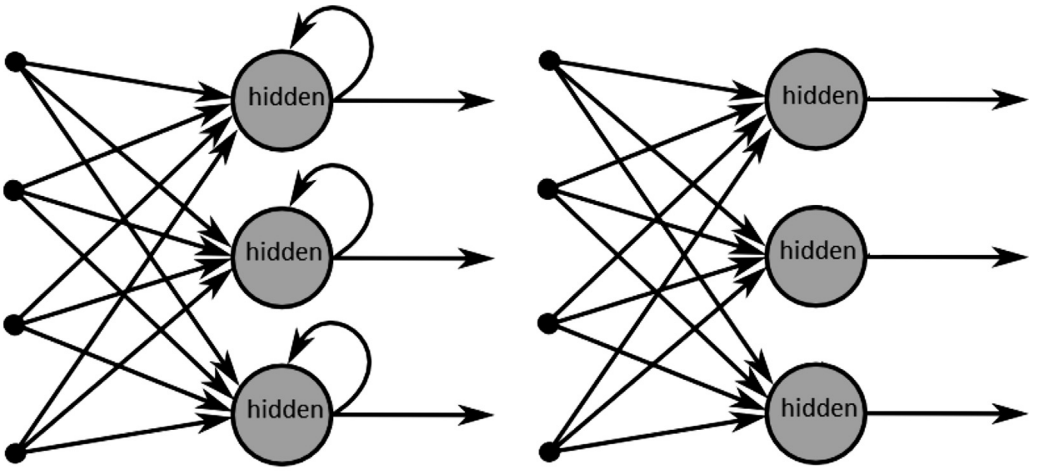

Fig. 2. Recurrent versus feedforward neural network.

1. The number of hidden neurons should be between the number of input components and the number of output components.
2. The number of hidden neurons should be 2/3 of the number of the input components, plus the number of output components.
3. The number of hidden neurons should be less than twice the number of the input components.

For RNN in particular, the work described in Gil et al. (2009) is interesting, since a method is described that estimates an appropriate number of hidden neurons for RNN that takes the recurrent structure of the network into account.

A graphical representation of a basic RNN is given in Fig. 2(a). This figure shows that the hidden neurons receive input values from both the input neurons and the hidden neurons. This is in contrast to feedforward neural networks that only receive input values from the input neurons, as shown in Fig. 2(b).

5. Application of basic recurrent neural networks to statistical language modeling

We focus our attention on the SLM task of modeling the probability of observing a word after a given sequence of words. The reasons for this restriction are (1) that this task is a basic one in the domain of SLM and (2) we think that a detailed account on the application of basic RNN to one specific task will provide more insight into this model than providing a shallow description on a range of tasks (space constraints prevent the detailed description of basic RNN on *all* SLM tasks where they can be useful). For convenience, we refer to this task as the basic SLM task.

5.1. Corpora

Training data for SLM are usually taken from corpora, i.e. large and structured sets of texts. Frequently used corpora are the Reuters corpus[2] (containing Reuters News stories), the Wikicorpus[3] (with large portions of Wikipedia), the Brown corpus[4] (consisting of 500 samples of English-language text, distributed across 15 genres, totaling roughly one million words), The Wall Street Journal (WSJ) corpus (distributed by LDC, that consists of read speech with texts drawn from a machine-readable corpus of Wall Street Journal news texts), and English Gigaword[5] (containing over ten million documents from seven news sources).

[2] http://about.reuters.com/researchandstandards/corpus/.
[3] http://www.lsi.upc.edu/~nlp/wikicorpus/.
[4] http://icame.uib.no/brown/bcm.html.
[5] http://www.ldc.upenn.edu/Catalog/catalogEntry.jsp?catalogId=LDC2011T07.

5.2. Vocabulary

The text documents contained in corpora need some preprocessing before they can be used by neural networks. It is customary to construct a fictitious vocabulary V containing all distinct words that are present in a considered corpus (or part of a corpus), in an arbitrary but fixed order. This requires the splitting of the corpus into individual words. One widely used tool to perform this task is the Stanford Parser.[6] Depending on the application, punctuation symbols may or may not be removed. To this vocabulary one adds a placeholder word, often denoted as the word UNK (unknown word), to represent all words that are seen in a test data set, after training, but that are not contained in the vocabulary. It is convenient to give the placeholder word UNK the label w_1 and all the other words labels $w_2, \ldots, w_N$. Thus we have that $V = (w_1, \ldots, w_N)$.

Each word in the vocabulary is then represented as a vector with N components via the following function τ:

$$\begin{aligned}\tau(w_1) &= (1, 0, 0, \ldots, 0)\\ \tau(w_2) &= (0, 1, 0, \ldots, 0)\\ &\vdots\\ \tau(w_N) &= (0, 0, \ldots, 0, 1)\end{aligned}$$

This function implements the so-called *1-of-N coding* which provides a suitable representation of words that can be processed by a neural network. It is clear that τ is invertible, which allows us to designate a 1-of-N coding vector with the corresponding word, for convenience. Thus we feel free to use such expressions as 'a word is given as input to the network' as shorthand for 'a 1-of-N coding vector of a word is given as input to the network'.

The corpus itself can, after tokenization and defining the vocabulary, be represented as a sequence of vectors, by replacing each word in the text by its corresponding 1-of-N coding vector. We denote this sequence as $D = (\boldsymbol{\omega}_1, \ldots, \boldsymbol{\omega}_n)$, where n is the number of words in the corpus.

5.3. Training data set

For the described task the training examples are all pairs $(\boldsymbol{\omega}_i, \boldsymbol{\omega}_{i+1})$, $i = 1, \ldots, n-1$. The vector $\boldsymbol{\omega}_i$, corresponding to the ith word in the considered corpus, is given as input, while $\boldsymbol{\omega}_{i+1}$ serves as target output. Providing the network with training examples of this form and choosing the output function as the softmax function (6) ensures that the network will learn the task of modeling the probability of a given word as the next word after a given sequence of previous words. Although the input of each training example contains only one word, the recurrent structure of the network implies that the network does not only use the previous word, but also incorporates some other contextual information, to construct the required probability distributions. This will be made clear below.

As the training examples are presented to the network in a definite order, we can denote the training data set as $\mathcal{T} = ((\mathbf{x}(1), \mathbf{y}(1)), \ldots, (\mathbf{x}(|\mathcal{T}|), \mathbf{y}(|\mathcal{T}|)))$, with $|\mathcal{T}|$ denoting the number of training instances in $\mathcal{T}$. Notice that we denote the ith training example by $(\mathbf{x}(i), \mathbf{y}(i))$ and not by $(\boldsymbol{\omega}_i, \boldsymbol{\omega}_{i+1})$, since it is not required that words are presented to the network in the same order as they appear in the corpus. We come back to this point in the next section.

If all instances belonging to $\mathcal{T}$ have been presented to the network and the weights have not yet converged to final values, we continue training by presenting $(\mathbf{x}(1), \mathbf{y}(1))$ again to the network, followed by $(\mathbf{x}(2), \mathbf{y}(2))$, and so on. The term *epoch* is used to denote the presentation of all examples in $\mathcal{T}$ once to the network.

Training is stopped when a certain stopping criterion is met. For example, training can be stopped when the decrease in training error is smaller than a certain threshold. A subset of the training data called validation set is usually reserved for validating the model by measuring the training error on this data.

[6] http://nlp.stanford.edu/software/lex-parser.shtml.

5.4. Batch

The parameters of a neural network are updated each time a certain, not necessarily constant, number of training instances have been presented to the network. We refer with the term *batch* to the presentation of such a number of training instances. Thus each time a batch has been processed, the network parameters are updated. This implies that the training data set $\mathcal{T}$ can be considered as consisting of batches $B(1), \ldots, B(L)$, i.e. $\mathcal{T} = (B(1), \ldots, B(L))$ where the ith batch $B(i)$ is of the form

$$((\mathbf{x}(j), \mathbf{y}(j)), (\mathbf{x}(j+1), \mathbf{y}(j+1)), \ldots, (\mathbf{x}(|B(i)| + j - 1), \mathbf{y}(|B(i)| + j - 1)))$$

and where $|B(i)|$ denotes the number of training instances in $B(i)$. A batch can be as small as one training instance or as large as the training data itself.

Although a batch can contain arbitrarily selected training examples from $\mathcal{T}$, an obvious choice is to let a batch consist of training instances corresponding to a fixed number of consecutive sentences in the given text. This is referred to as the *consecutive scheme*. If the batches $B(1), B(2), \ldots B(L)$ are also in the same order as the sentences in the text, then this is referred to as the *genuine consecutive scheme*. Alternatively, batches can consist of training examples coming from randomly selected sentences in the given text. This is referred to as the *random scheme*. Some researchers claim that randomizing the order of the sentences results in faster training (Bengio et al., 2003; Mikolov et al., 2011). Note that only in the genuine consecutive scheme it is true that $(\mathbf{x}(i), \mathbf{y}(i)) = (\boldsymbol{\omega}_i, \boldsymbol{\omega}_{i+1})$.

It is convenient to extend the notation for a training instance to incorporate the batch to which it belongs. Thus we will use the notation $(\mathbf{x}(k, t), \mathbf{y}(k, t))$ to refer to the tth training instance of the kth batch.

5.5. Description of basic recurrent neural networks applied to the basic SLM task

Given that a training example is of the form $(\boldsymbol{\omega}_i, \boldsymbol{\omega}_{i+1})$, see Section 5.3, with $\boldsymbol{\omega}_i$ and $\boldsymbol{\omega}_{i+1}$ 1-of-N coding vectors of words from the vocabulary, we have that $n_I = N = |V|$ and $n_J = N = |V|$, where n_I and n_J denote the number of input and output components of an RNN (see Section 4). Since we use the notation $(\mathbf{x}(k, t), \mathbf{y}(k, t))$ to denote an arbitrary training example, see Section 5.4, the description of an RNN (1)–(4) takes the following form when applied to the basic SLM task:

$$a_j(k,t) = \sum_{i=1}^{|V|} \alpha_{ji}(k)\, x_i(k,t) + \sum_{i=1}^{n_H} \rho_{ji}(k)\, h_i(k, t-1), \quad j = 1, \ldots, n_H \tag{7}$$

$$h_j(k,t) = F(a_j(k,t)), \quad j = 1, \ldots, n_H \tag{8}$$

$$b_j(k,t) = \sum_{i=1}^{n_H} \beta_{ji}(k)\, h_i(k,t), \quad j = 1, \ldots, |V| \tag{9}$$

$$o_j(k,t) = G(b_j(k,t)), \quad j = 1, \ldots, |V| \tag{10}$$

5.6. Parameters

The parameters to be learned are $\alpha_{ji}, j = 1, \ldots n_H, i = 1, \ldots, |V|$, $\rho_{ji}, j = 1, \ldots, n_H, i = 1, \ldots, n_H$ and $\beta_{ji}, j = 1, \ldots, |V|$, $i = 1, \ldots, n_H$. These parameter values are typically randomly initialized. A common choice is the Gaussian distribution. For example, in Mikolov (2012) a normal distribution with mean 0 and variance 0.1 is used. Another distribution that is sometimes chosen to initialize the parameter values is the uniform distribution over some interval $[-\delta, \delta]$, $\delta > 0$ (Rojas, 1996). More sophisticated initialization methods can be found in, for example, Fernández-Redondo and Hernández-Espinosa (2000) and Marichal et al. (2007). A lot of practical 'tricks' in applying back-propagation learning can be found in LeCun et al. (1998). As outlined above, the parameter values are updated every time a batch has been processed.

The number of hidden units n_H must be chosen by the user, as discussed in Section 4.

5.7. Initial value for $h_i(k, 0)$

It is customary to define $h_i(1, 0) = 0$ for all $i = 1, \ldots, n_H$. For any other batch $B(k)$, $k > 1$, the initial values depend on the scheme that is used.

First, if the genuine consecutive scheme is used it is natural to define $h_i(k, 0) = h_i(k-1, |B(k-1)|)$, since in this case the order in which words are presented to the network equals the order in which they appear in the given text. Loosely speaking, in this scheme all words from the given text are supplied to the system, one by one, in the order in which they appear in the given text, and by defining $h_i(k, 0) = h_i(k-1, |B(k-1)|)$, the network will calculate the output based on the given word and *all* previous words in the text.

Second, if the consecutive scheme is used, sentences contained in a batch correspond to consecutive sentences in the text, but the last sentence of a certain batch and the first sentence of the next batch are not consecutive. Thus when a new batch is started, its context cannot be considered as contained in the previous batches. In this case it is natural to define $h_i(k, 0) = 0$. This implies that when a word is given as input to the network, only the previous words contained in the same batch are taken as context into account.

Third, if the random scheme is used, we also define $h_i(k, 0) = 0$. In this case, it is even natural that when a training example $(\mathbf{x}(k, t), \mathbf{y}(k, t))$ is given to the network for which $\mathbf{x}(k, t)$ corresponds to the first word of a sentence in the text, to define $h_i(k, t-1) = 0$ for all i.

A natural question is why one would choose the consecutive scheme or random scheme, given the fact that in these schemes the context that is taken into account is limited, and does not go all the way back to the first word of the text as in the genuine consecutive scheme? An important reason is that by defining $h_i(k, 0) = h_i(k-1, |B(k-1)|)$ rounding errors are carried over batches and get accumulated. The same applies to other types of errors, for example, errors in the tool that is used to split the given text into individual words. Furthermore, the fact that in the genuine consecutive scheme all previous words are taken into account in calculating the output should not be interpreted as saying that all previous words are given as input. Rather, one word is given as input and there is only an *influence* of all previous words. Thus the context size that is effectively taken into account in the genuine consecutive scheme does not necessarily exceed the context size in the other two schemes in a significant way. We continue this discussion in the following section. For convenience, we assume the genuine consecutive scheme in the remainder of this section, since in this scheme the words are presented to the network in the order in which they appear in the corpus, which is intuitively appealing.

5.8. Interpretation of the produced output

Using the softmax (6) as output function ensures that the outputs $o_1(k, t), \ldots, o_{|V|}(k, t)$ can be interpreted as the probabilities that each of the words in the vocabulary V appears next after a certain sequence of previous words. However, since the input to an RNN is limited to one word and since the recurrent term $h_i(k, t-1)$ in Eq. (7) only ensures that nonlinear transformations of previous words are cycled through the network, we can informally state that the context that is taken into account is something *between* the previous word and all previous words. More formally, when a trained network has been presented a sequence of words $(\tau(w_{j_1}), \ldots, \tau(w_{j_{m-1}}))$, with $w_{j_i} \in V$, and it is now given the word $\tau(w_{j_m})$, the produced output is not to be interpreted as $P(W_{j_{m+1}} = w_1 \mid w_{j_1} \ldots, w_{j_m}), \ldots, P(W_{j+1} = w_{|V|} \mid w_{j_1}, \ldots, w_{j_m})$, where W_{j+1} represents the word to appear next after the given sequence as a random variable. Rather, there exist functions $f_1, \ldots, f_{|V|}$ such that the outputs can be interpreted as

$$P\left(W_{j+1} = w_i \mid f_i(w_{j_1}, \ldots, w_{j_{m-1}}), w_{j_m}\right), \quad i = 1, \ldots, |V|$$

5.9. Training error and validation error

The error function that is widely used on the basic SLM task is cross-entropy (Frinken et al., 2012). For a given training example $(\mathbf{x}(k, t), \mathbf{y}(k, t))$, this criterion defines the error between the target output $\mathbf{y}(k, t)$ and the produced output $(o_1(k, t), \ldots, o_{|V|}(k, t))$ as:

$$E(k, t) = -\sum_{p=1}^{|V|} \ln o_p(k, t)\, y_p(k, t) \tag{11}$$

Since $\mathbf{y}(k, t)$ is the 1-of-N coding vector of a certain word in V, only the component corresponding to the word $w \in V$ for which $\tau(w) = \mathbf{y}(k, t)$ is one, and all other components are zero. Thus we can write $E(k, t)$ also as

$$E(k,t) = -\sum_{p=1}^{|V|} \ln o_p(k,t)\, \Delta_p(k,t) \tag{12}$$

where

$$\Delta_p(k,t) = 1 \quad \text{if} \quad y_p(k,t) = 1 \tag{13}$$

$$= 0 \quad \text{otherwise} \tag{14}$$

The error associated with a batch $B(k)$ is then defined as $E(k) = \sum_t E(k, t)$ and the error associated with an epoch is analogously defined as $E = \sum_k E(k)$. When referring to 'error' or 'training error' further in this paper, we specifically mean the error associated with a batch, i.e. $E(k)$, since it is after having processed the training examples in a batch that the parameter values are updated. After having processed a batch we typically want to know the current error in training the system and naturally we then take $E(k)$ to mean this current error.

If we have a validation data set, we can evaluate the generalization capability of the network at any appropriate moment during training. The validation error is typically calculated after some batch $B(k)$ has been processed, by giving all examples from $\mathcal{V}$ to the network, and then by calculating the associated error $E^v(k)$ made by the network in the same way as $E(k)$ is calculated:

$$E^v(k) = -\sum_{p=1}^{M_v} \sum_{j=1}^{|V|} \ln o_j(k,p)\, \Delta_j(k,p) \tag{15}$$

where $E^v(k)$ thus denotes the validation error and where it is implicitly understood that the network is applied on the examples in $\mathcal{V}$ with parameter values $\alpha_{ji}(k)$, $\beta_{ji}(k)$ and $\rho_{ji}(k)$. The importance of the validation error is that it can be considered as an approximation for the generalization error.

5.10. Updating the parameters

Many methods have been developed to update the parameters in such a way that the training error is minimized. Update rules that are extensions of gradient descent can be considered as the most basic. In gradient descent a parameter p is updated as

$$p(k+1) = p(k) - \lambda \frac{\partial E(k)}{\partial p(k)} \tag{16}$$

where $p(k+1)$ is the value of parameter p that will be used by the network to process training examples from batch $B(k+1)$, and where p refers to either α_{ji} or β_{ji} or ρ_{ji}. The parameter $0 < \lambda < 1$ is called the learning rate and can be chosen as, for example, 0.1 (Mikolov, 2012). The gradients $\partial E(k)/\partial p(k)$ are derived in Appendix A.

Although the update rule above is often used in RNN and other types of neural networks, it is well known that a fixed learning rate tends to make the learning process inefficient (Polak, 1997). Too low of a learning rate makes the network learn very slowly, while too high of a learning rate makes the weights and objective function diverge, implying that there is no learning at all. This can be solved by adapting the learning rate during training, for example by reducing λ as $1/t$ (Seung, 2002).

Alternatively, the learning rate is kept fixed, but the update rule (16) is extended with the inclusion of a momentum term $\mu \in [0, 1]$, such that a parameter p is updated as

$$p(k+1) = p(k) + \mu\,(p(k) - p(k-1)) - \lambda \frac{\partial E(k)}{\partial p(k)}$$

It has been found that such a momentum term increases the rate of convergence dramatically (Rumelhart et al., 1986). Typical choices for λ and μ in the above update rule are $\lambda = 0.1$, $\mu = 0.9$ (Matignon, 2005).

Many other advanced update rules have been developed. As an illustration, ALAP$_1$ (Almeida et al., 1998) updates a parameter as

$$p(k+1) = p(k) - \lambda(k) \frac{\partial E(k)}{\partial p(k)}$$

with variable learning rate $\lambda(k)$. After all parameters have been updated, the learning rate is updated via the rule

$$\lambda(k+1) = \lambda(k) + \gamma < \nabla E(k), \nabla E(k-1) >$$

where <.,.> denotes the inner product and γ denotes a positive constant, e.g. $\gamma = 0.1$.

In all the update rules above, the derivative $\frac{\partial E(k)}{\partial p(k)}$ is needed. This derivative with respect to $\alpha_{ji}(k)$ requires one to calculate $\delta_j^h(k,t) = \frac{\partial E(k,t)}{\partial a_j(k,t)}$, which in turn requires one to calculate $\delta_p^h(k,t+1) = \frac{\partial E(k,t+1)}{\partial a_p(k,t+1)}$, $p = 1, \ldots, n_H$ (see Eqs. (A.9) and (A.3) in Appendix A). Consequently, parameters can only be updated after all subsequent training examples from the current batch are processed by the system.

More concretely, parameters are updated in two passes, a forward pass and a backward pass. In the forward pass, the values $a_j(k,t)$, $h_j(k,t)$, $b_j(k,t)$ and $o_j(k,t)$ are calculated according to (7)–(10) and this over the training examples $(\mathbf{x}(k,t), \mathbf{y}(k,t))$, $t = 1, \ldots, |B(k)|$. This provides all necessary values to calculate $\delta_j^h(k,t)$, as defined in (A.3), recursively, starting from $\delta_j^h(k,|B(k)|)$, then calculating $\delta_j^h(k,|B(k)|-1)$, and so on, up to $\delta_j^h(k,1)$. Each value $\delta_j^h(k,t)$ uses the values $\delta_j^h(k,t+1)$, $j = 1, \ldots, n_H$, which is why this pass is performed backwards. Notice that to calculate $\delta_j^h(k,|B(k)|)$ we need to define $\delta_j^h(k,|B(k)|+1)$. These values are typically defined as zero. The values $\delta_j^h(k,t)$ are then used to update $\rho_{ji}(k)$ and $\alpha_{ji}(k)$, which follow from Eqs. (A.8) and (A.9) in Appendix A. Eq. (A.7) shows that to update $\beta_{ji}(k)$ only a forward pass is needed.

5.11. Stopping criterion

Training is stopped when a certain stopping criterion is met. For example, training can be stopped when the decrease in training error is smaller than a certain threshold. This condition can be made more severe by requiring that the training error is smaller than a certain threshold for some predetermined number S of consecutive batches. However, to avoid overfitting it is necessary to incorporate the validation error into the stopping criterion. Training is then typically stopped whenever the validation error appears to start increasing, for example if the validation error is larger than the previous validation error. The rationale is that at the start of the training, the network is not yet adapted to the intended task, which will result in large errors on $\mathcal{T}$ and $\mathcal{V}$. These errors will decrease as the network is trained. While the training error keeps decreasing, the validation error will start to increase after some learning period, when overfitting occurs.

In Prechelt (1997), it is rightly pointed out that the validation error does not necessarily show a smooth decrease-increase behavior. The fact that the validation error may show a jagged pattern can be accounted for by continuing training until the *training error* meets some stopping criterion, but to select as optimal parameter values those values for which the corresponding *validation error* is the smallest among all calculated validation errors. One implementation of this strategy is to define a threshold ϵ and to stop training when $E(k)/E(k-1) < \epsilon$. The optimal parameter values are then taken as the parameter values corresponding to the batch κ for which it holds that $\kappa = \underset{k}{arg\min} E^v(k)$, where k runs over all batches that were processed during the training process. This strategy requires one to keep in memory all parameter values that were calculated during training.

5.12. Detailed training algorithm in pseudocode

We provide a training algorithm for the basic recurrent neural network described above applied to the basic SLM task. To be concrete, we choose as update rule gradient descent with momentum (17) and initialize the parameter values according to the normal distribution with mean 0 and variance 0.1. Furthermore, the batches are assumed to be arranged according to the consecutive scheme (see Section 5.4; if another scheme is used, Section 5.7 should be taken into account).

This is not to say that this is the best training algorithm. More advanced training algorithms could result in better optima (although often at the cost of increased training time), but the algorithm below is advanced enough to be useful

in practice, and at the same time it is basic enough to provide insight into the implementation of a host of more advanced extensions on it.

Algorithm 1.

1. Decide on the following:
   - A corpus (see Section 5.1).
   - The value for n_H, i.e. the number of hidden neurons.
   - The values for λ and μ in Eq. (17).
   - A value $\epsilon > 0$ that is used as threshold on the training error to stop training.
   - A value for $S \in \mathbb{N}$ that indicates the number of consecutive times that the training error should be below the threshold before training is stopped. We can choose $S = 1$, but because in practice the training error will show a jagged behavior, it is advised to consider a larger S.
   - The number of consecutive sentences contained in a batch.
2. Preprocess the data set. Split the text into words $(w_1, \ldots, w_N)$, construct the vocabulary V, and convert the words w_i to the corresponding 1-of-N coding $\boldsymbol{\omega}_i = \tau(w_i)$, as outlined in Section 5.2.
3. Split the data into training data and validation data, and divide the training data into batches, as outlined in Sections 5.3 and 5.4.
4. Initialize the parameter values. Define $\alpha_{ji}(1)$, $\beta_{ji}(1)$ and $\rho_{ji}(1)$ as random values from the normal distribution with mean 0 and variance 0.1. Using gradient descent with momentum, we also need to initialize the parameters for the second batch. It is common to initialize them to be equal to the parameter values for the first batch, i.e. $\alpha_{ji}(2) = \alpha_{ji}(1)$, $\beta_{ji}(2) = \beta_{ji}(1)$ and $\rho_{ji}(2) = \rho_{ji}(1)$.
5. Initialize $k = 1$, where k is the index that keeps track of the batches and initialize $s = 0$ which will be used to keep track of the current number of consecutive times that the training error is below the threshold.
6. If no batch has been selected before or if the previously selected batch was the last batch from the training data set, define $B(k)$ as the first batch from the training data set. Otherwise, define $B(k)$ as the next batch from the training data set. Initialize

$$\begin{aligned} h_i(k, 0) &= 0, \quad i = 1, \ldots, n_H \\ \delta_i^h(k, |B(k)| + 1) &= 0, \quad i = 1, \ldots, n_H \\ t &= 1 \end{aligned}$$

7. Consider training example $(\mathbf{x}(k, t), \mathbf{y}(k, t))$ from $B(k)$. Do the following calculations:

$$\begin{aligned} a_j(k, t) &= \sum_{i=1}^{|V|} \alpha_{ji}(k)\, \Delta_i(k, t) + \sum_{i=1}^{n_H} \rho_{ji}(k)\, h_i(k, t-1), \quad j = 1, \ldots, n_H \\ h_j(k, t) &= F(a_j(k, t)), \quad j = 1, \ldots, n_H \\ b_j(k, t) &= \sum_{i=1}^{n_H} \beta_{ji}(k)\, h_i(k, t), \quad j = 1, \ldots, |V| \\ o_j(k, t) &= G(b_j(k, t)), \quad j = 1, \ldots, |V| \\ \delta_j^o(k, t) &= o_j(k, t) - \Delta_j(k, t), \quad j = 1, \ldots, |V| \\ E(k, t) &= -\sum_{j=1}^{|V|} \ln o_j(k, t)\, \Delta_j(k, t) \end{aligned}$$

8. Keep in memory the following values:

$$h_j(k,t), \quad j=1,\ldots,n_H$$
$$o_j(k,t), \quad j=1,\ldots,|V|$$
$$\delta^o_j(k,t), \quad j=1,\ldots,|V|$$
$$E(k,t)$$

9. If $t=|B(k)|$, go to the next step. Otherwise, increase t by 1 and go to step 1.
10. Calculate the training error corresponding to the current batch $B(k)$:

$$E(k)=\sum_{t=1}^{|B(k)|}E(k,t) \tag{17}$$

11. Keep in memory the following values:

$$\alpha_{ji}(k), \quad j=1,\ldots,n_H, i=1,\ldots,|V|$$
$$\rho_{ji}(k), \quad j=1,\ldots,|V|, i=1,\ldots,n_H$$
$$\beta_{ji}(k), \quad j=1,\ldots,|V|, i=1,\ldots,n_H$$

12. Calculate the error on the validation set, $E^v(k)$, as given by Eq. (15). Keep this value in memory.
13. Check whether $E(k)/E(k-1)<\epsilon$. If so, increase s by 1. If not, set $s=0$.
14. If $s=S$, stop training. Find the minimum validation error $E^v(k)$. Return the corresponding parameter values $\alpha_{ji}(k)$, $\beta_{ji}(k)$ and $\rho_{ji}(k)$ as optimal parameter values.
    If $s<S$ go to the next step.
15. Perform the following calculations:

$$\delta^h_j(k,t)=h_j(k,t)(1-h_j(k,t))\left[\sum_{p=1}^{|V|}\delta^o_p(k,t)\,\beta_{pj}(k)+\sum_{p=1}^{n_H}\delta^h_p(k,t+1)\,\rho_{pj}(k)\right]$$

    for $j=1,\ldots,n_H$. Keep the values $\delta^h_j(k,t)$ in memory.
16. Reduce t by 1.
17. If $t=0$ go to the next step. Otherwise, go to step 15.
18. If $k>2$, calculate $\partial E(k)/\beta_{ji}(k)$, $\partial E(k)/\rho_{ji}(k)$ and $\partial E(k)/\alpha_{ji}(k)$, as given by Eqs. (A.7)–(A.9).
19. If $k>2$ update the parameter values as

$$\beta_{ji}(k+1)=\beta_{ji}(k)+\mu\left(\beta_{ji}(k)-\beta_{ji}(k-1)\right)-\lambda\frac{\partial E(k)}{\partial\beta_{ji}(k)}, \quad j=1,\ldots,|V|, i=1,\ldots,n_H$$
$$\rho_{ji}(k+1)=\rho_{ji}(k)+\mu\left(\rho_{ji}(k)-\rho_{ji}(k-1)\right)-\lambda\frac{\partial E(k)}{\partial\rho_{ji}(k)}, \quad j=1,\ldots,n_H, i=1,\ldots,n_H$$
$$\alpha_{ji}(k+1)=\alpha_{ji}(k)+\mu\left(\alpha_{ji}(k)-\alpha_{ji}(k-1)\right)-\lambda\frac{\partial E(k)}{\partial\alpha_{ji}(k)}, \quad j=1,\ldots,n_H, i=1,\ldots,|V|$$

Otherwise define

$$\beta_{ji}(k+1) = \beta_{ji}(k)$$
$$\rho_{ji}(k+1) = \rho_{ji}(k)$$
$$\alpha_{ji}(k+1) = \alpha_{ji}(k)$$

20. Go to step 6.

5.13. Empirical evaluation

In Mikolov (2010), several configurations of the basic recurrent neural network (e.g. different numbers of hidden neurons) are trained to perform several speech recognition tasks. Training data was derived from the NYT section of English Gigaword. Several baseline models were considered, but only the modified Kneser-Ney smoothed 5-gram model (Kneser and Ney, 1995), hereafter KN5 for short, is compared to the basic RNN. The other baseline models are compared to more advanced RNNs, which we consider below. PPL and WER were used to evaluate the performance. It was found that the basic RNN outperforms the KN5 model in terms of these measures. The values of the evaluation measures for the worst performing RNN (containing the smallest amount of hidden neurons, namely 60) were PPL = 229 and WER = 13.2, while the KN5 model had values PPL = 221, WER = 13.5. The best configuration (with the largest amount of hidden neurons, namely 400) significantly outperformed the KN5 model: PPL = 171, WER = 12.5.

In Mikolov et al. (2011) and Mikolov (2012), an extensive experiment was performed on the Penn Treebank portion of the WSJ corpus. RNN were compared to several N-gram models and to more advanced models, namely the maximum entropy model (Rosenfeld, 1994), the random clusterings language model (LM) (Emami and Jelinek, 2005), the random forest LM (Xu, 2005), the structured LM developed by Filimonov and Harper (2009), and the within and across sentence boundary LM (Momtazi et al., 2010). The authors observed that the basic RNN had a lower perplexity than all other models, except for the within and across sentence boundary LM which had a perplexity of 116.6, while the basic RNN had a perplexity of 124.7.

An experimental evaluation in the domain of machine translation was performed in Le et al. (2012). It was observed that a FNN with context length 10 performed slightly better in terms of PPL than a RNN on a large-scale English to French translation task. Performance in terms of BLEU was identical for both network architectures.

5.14. Limitations

The basic RNN has some important drawbacks:

- *Training time*. Training a RNN is known to be very slow. For example, the RNNs used in the experiments described in Mikolov (2010) took several weeks of training, although the authors considered only about 17% of the NYT section of English Gigaword for training. From (7) to (10) it follows that the total training time is proportional to $2 \times P \times n_H \times (n_H + |V|)$ given a complexity of order $P \times n_H \times (n_H + |V|)$, where P denotes the number of epochs needed to reach convergence. Usually it takes about 10–50 training epochs to achieve convergence (Mikolov et al., 2011), although cases have been reported where even thousands of epochs were needed (Xu and Rudnicky, 2000). In addition the size of the vocabulary $|V|$) which for many language and speech applications is usually very large, plays a crucial role in the real complexity of the training.
- *Fixed number of hidden neurons*. The number of hidden neurons n_H has to be fixed in advance. However, in practice the user has no clue as how to choose an appropriate number of hidden neurons, since there does not exist a generally accepted method that determines this number. A lot of rules of thumb are available (see Section 5.6), but these rules can give very different values for n_H.
- *Small context size in practice*. Although in theory the context size that can be taken into account is unlimited (if the genuine consecutive scheme is used, the history of words that is taken into account equals all previous words relative to the current word), the range of context that is actually accessed is quite limited. This observation is often referred to as the vanishing gradient problem (Hochreiter et al., 2001).

5.15. Recurrent versus feedforward neural networks

In this final section on the application of (basic) recurrent neural networks to statistical language modeling we provide a short overview of the main differences between FNN and RNN:

- When using a FNN, one is restricted to use a fixed context size that has to be determined in advance. RNNs use in principle the whole context, although practical applications indicate that the context size that is effectively used is rather limited (see Section 5.14). However, RNNs have at least the advantage that one has not to decide on the context size, a parameter for which a suitable value is very difficult to determine.
- Comparisons between RNNs and FNNs in applications on statistical language modeling typically favor RNNs, as will become clear in later sections.
- FNNs have the advantage that the output can be easily interpreted. The output associated with the ith output neuron can simply be interpreted as $P(W_{j+1} = w_i \mid w_{j_1}, \ldots, w_{j_m})$, where W_{j+1} represents the next word after a given sequence of words $w_{j_1}, \ldots, w_{j_m}$, represented as a random variable. Such an intuitive interpretation does not hold for RNNs, as we outlined in Section 5.8.
- Because RNNs are dynamical systems, some issues can be encountered that cannot arise in FNNs. For example, it may happen that the influence of a given input on the network output blows up exponentially as subsequent training examples are presented (Hochreiter et al., 2001), a highly undesired artefact.

6. Methods to reduce training time

6.1. Limiting the size of the vocabulary

6.1.1. Description

As the training time is proportional to $P \times n_H \times (n_H + |V|)$, see Section 5.14, one obvious way to reduce the training time is to restrict the size of the vocabulary. This can be done by leaving out the least frequent words, where the frequency of a word w_i simply refers to the number of times that w_i appears in the considered corpus divided by the total number of words in the corpus. Leaving a word out of consideration simply means replacing it by the placeholder word UNK (see Section 5.2). A compromise has to be made between developing an as accurate system as possible, thereby choosing $|V|$ very large, and reducing the training time as much as possible, i.e. choosing $|V|$ small. A typical choice for $|V|$ is between 50 000 and 300 000 (Mikolov, 2012), although in Collobert and Weston (2008) $V = 30\,000$ was chosen.

6.1.2. Empirical evaluation

Limiting the vocabulary size $|V|$ can provide very significant speedups, but at a noticeable cost of accuracy. Schwenk and Gauvain (2005) used a vocabulary size of 2000, but at the cost of a significant performance degradation as was later shown in Le et al. (2011).

Unfortunately, the influence of the vocabulary size on the performance of RNNs seems to be an as yet untouched research topic. One noteworthy publication is Le et al. (2011), where the authors compared different configurations of a feedforward SOUL network (see Section 6.3) to several FNN configurations with $|V| = 12\,000$. The SOUL network used a much larger vocabulary of about 300 000 entries. The FNN and SOUL network configurations were obtained by varying context lengths (4 and 6) and by interpolating these models with a 4-gram LM. Experiments were performed on the Arabic GALE[7] transcription task, i.e. a speech-to-text task on Arabic data, on three evaluation sets. In terms of PPL the SOUL networks consistently outperformed the FNNs with the same context length on all the test sets. Relative improvements of about 10% for stand-alone SOUL networks and 7% for interpolated models are observed for context length 4 on different test sets. For context length 6, the gains were even larger, about 15% and 12% in stand-alone and interpolated scenarios respectively. These results carried over to speech recognition experiments. Of course, these results apply to *feedforward* neural networks and thus cannot necessarily be extrapolated to their recurrent counterpart.

[7] http://projects.ldc.upenn.edu/gale/.

It has also to be noticed that a vocabulary size of 12 000 is small in comparison to the sizes that are more often used in practice, as outlined in Section 6.1.

6.2. Hierarchical probabilistic networks

6.2.1. Description

Another way to reduce the computational complexity in terms of $|V|$ was proposed by Morin and Bengio by the introduction of hierarchical probabilistic networks (Morin and Bengio, 2005), where words from the vocabulary are grouped into classes. The basic idea is that if words are grouped into classes, the probability $P(W_{j+1} = w \mid w_1, \ldots, w_m)$ as learned by the network (see Section 5.8), can be replaced by a probability of the form $P(W_{j+1} = w \mid C = c(w), w_1, \ldots, w_m)P(C = c(w) \mid w_1, \ldots, w_m)$, where C denotes the class to which the next word belongs as a random variable, and where c is a mapping from the vocabulary to the set of classes, mapping each word to the class to which it belongs. The resulting decrease in computational complexity is easily seen: suppose that V contains 100 000 words, and that these words are divided over 100 classes each containing 100 words. Instead of calculating 100 000 conditional probabilities, it now suffices to calculate 200 conditional probabilities, i.e. 100 probabilities of the form $P(W_{j+1} = w \mid C = c(w), s_1, \ldots, s_m)$ and 100 probabilities of the form $P(C = c(w) \mid s_1, \ldots, s_m)$. In Morin and Bengio (2005), the classes are constructed by combining empirical statistics with prior knowledge from the WordNet resource (Fellbaum, 1998).

Although in Morin and Bengio (2005) this technique was applied to feedforward neural networks, the technique can be equally well applied to RNN. Indeed, Mikolov applied it to RNN (Mikolov et al., 2011), but his approach to construct the classes was much simpler. Assignment to classes is done before the training starts, and is based on the relative frequency of the words (known as frequency binning). Huang et al. (2013) also apply frequency binning but introduce a two-level hierarchy of classes, such that words are binned into classes by word frequency, and then classes are binned into superclasses by class frequency.

Shi et al. (2013) also used classes in an RNN architecture, but the construction of classes was more advanced. Instead of grouping words purely in terms of their frequency, they applied the Brown clustering algorithm, a data-driven hierarchical word clustering algorithm (Brown et al., 1992). This clustering technique receives as input a text, i.e. a sequence of words, and produces as output a binary tree, the leaves of which are words.

A simple illustration of a hierarchical RNN is given in Fig. 3. For illustration purposes only one output neuron is included, although in practice there will be many output neurons. What is important is that the output neuron does not represent one particular word, as in the basic RNN, but a class of (semantically related) words.

6.2.2. Empirical evaluation

In Morin and Bengio (2005), experiments were performed on the Brown corpus, with $|V| = 10\,000$. It was observed that using classes reduced training time with a factor of about 250 (as noted in the previous section, the method of classes was applied to *feedforward* neural networks in this study). However, the perplexity of the network increased from 195.3 for the original network to 220.7 when classes were used. The network with classes still outperformed the baseline trigram model which had a perplexity of 268.7.

RNNs with classes were evaluated by Mikolov (2012) on a speech recognition task with Wall Street Journal training data. Mikolov used 400 classes, a hidden layer size up to 800 units and a vocabulary size of 20 000. In terms of WER, the RNN outperformed the benchmark models that were used, namely KN5, the discriminative LM developed by Xu et al. (2009), and the joint LM by Filimonov and Harper (2009).

RNNs with a two-level class hierarchy were evaluated by Huang et al. (2013) on a speech recognition task on a Microsoft dataset of dictated short messages. Huang used 1200–1600 classes, 40 superclasses and 25–200 hidden units. The two-level class-based RNN achieved WER equal to or better than a regular RNN and trained in roughly half the time.

Shi et al. (2013) used Wall Street Journal training data to compare the RNN with frequency based classes (RNN-F hereafter), as applied by Mikolov, to the RNN with classes constructed using the Brown clustering algorithm (RNN-B hereafter). It was observed that RNN-B trained over 13 epochs obtained the same perplexity as RNN-F trained over 24 epochs. Another experiment involved a speech recognition task, again using Wall Street Journal training data, showing that perplexity of RNN-B reduced the perplexity by approximately 5%, compared to RNN-F. The last experiment, a

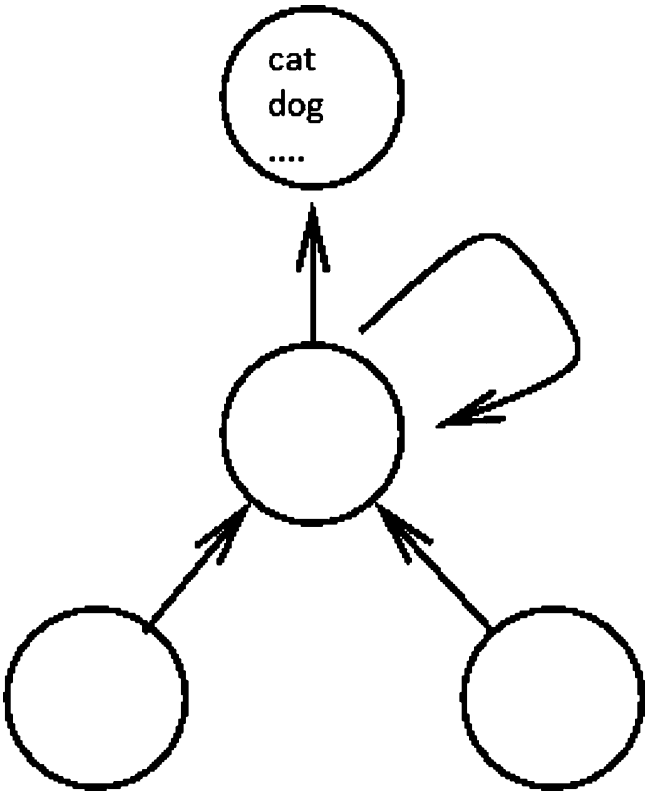

Fig. 3. Hierarchical recurrent neural network.

speech recognition task with the Switchboard-I training set,[8] demonstrated that the perplexity of RNN-B with 9 epochs competes with that of RNN-F trained for 16 epochs.

6.3. SOUL network

6.3.1. Description

Recently, Le et al. (2011) extended the hierarchical probabilistic network model by structuring the vocabulary according to a clustering tree, as illustrated in Fig. 4. Classes can now belong to different levels of a hierarchy, such that a certain class is a member of another class. This structure is represented as a tree of depth U, where each word of the vocabulary corresponds to a leaf node, and where there is a unique path from the root node to each leaf node. This means that for a word $w \in V$ there is a sequence of classes $x_0, \ldots, x_U$ that represents this word. The class x_0 denotes the root node, while the class x_U denotes the leaf node corresponding to w, i.e. x_U, as a leaf node, is a class containing only one word, namely the word w. A probability of the form $P(W_{j+1} = w \mid w_1, \ldots, w_m)$ is now replaced by

$$P(x_0, \ldots, x_U \mid w_1, \ldots, w_m) = P(x_0 \mid w_1, \ldots, w_m) \prod_{u=1}^{U} P(x_u \mid w_1, \ldots, w_m, x_0, \ldots, x_{u-1})$$

Each class conditional probability of the form $P(x_u \mid w_1, \ldots, w_m, x_0, \ldots, x_{u-1})$ is estimated using a softmax layer in the neural network.

Although Le originally developed this structure for feedforward neural networks, in Le (2013) he introduced an extension of the SOUL model that approximates the recurrent architecture with a limited history. As it turns out that the training scheme for the SOUL network cannot be directly applied to RNN, Le restricted the recurrence to a dozen or so previous words, which results in an N-gram version of RNN. The SOUL training scheme is then adopted for this

[8] http://www.ldc.upenn.edu/Catalog/catalogEntry.jsp?catalogId=LDC97S62.

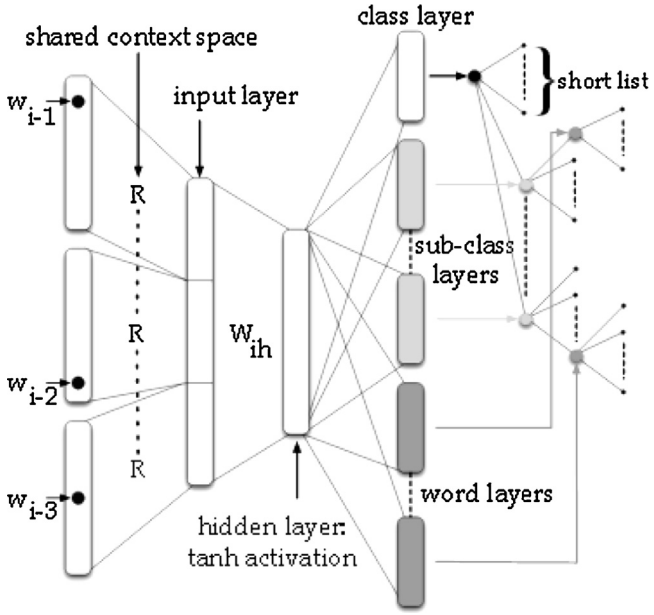

Fig. 4. SOUL network.
Taken from Le et al. (2011).

new kind of model. After training is finished, the parameter values of this *N*-gram version of RNN are plugged into a truly recurrent architecture. This true RNN is then trained for several epochs until convergence.

6.3.2. *Empirical evaluation*

The SOUL network has been evaluated in several studies on speech recognition and machine translation. The findings of these studies are summarized in Le (2013).

We first discuss the results for SOUL *feedforward* neural networks. The speech recognition task involved Mandarin and Arabic data. For the Mandarin data task, the vocabulary consisted of 56 052 words. Perplexity and character error rate (CER) were used as evaluation measures. Results were compared to a 4-gram language model and to FNNs interpolated with a *N*-gram model. The SOUL network, also interpolated with an *N*-gram model, gained a relative improvement of 23% in perplexity and 7–9% in CER compared to the 4-gram language model. It also outperformed the FNN; for example, the perplexity of their best configuration of the SOUL model was 162, while the lowest perplexity attained by a FNN was 172. On the Arabic data, for which a vocabulary of about 300 000 words was constructed, perplexity on all test sets was lower for the SOUL network than for the 4-gram and FNN models. The results were comparable to those for the Mandarin data and were confirmed in terms of WER.

For the machine translation task the total amount of training data was approximately 12 million sentence pairs for the bilingual part, and about 2.5 billion of words for the monolingual part. Translations were performed from English to French and from English to German. The vocabulary for each language contained about 500 000 words, while the hidden layer contained 300 units. They observed for the English-French task a BLEU improvement of 0.3, as well as a similar trend in TER over conventional *N*-gram baseline models. The results on the English-German task showed the same trend with a 0.5 BLEU point improvement.

The SOUL *recurrent* neural network was evaluated on the above machine translation task. In terms of perplexity, it was observed that the SOUL RNN, having PPL 81, performed better than a SOUL FNN with context lengths 4 and 6 (PPL of 92 and 82 resp.), but worse than the SOUL FNN with context lengths 8 and 10 (perplexity 78 and 77 resp.). In terms of BLEU very similar results were obtained: the SOUL RNN obtained a BLEU score of 30.4, outperforming SOUL FNN with context lengths 4 and 6 (BLEU = 29.8 resp. BLEU = 30.2), but performing worse than SOUL FNN with context lengths 8 and 10 (BLEU = 30.6 resp. BLEU = 30.5).

6.4. Hierarchical subsampling networks

6.4.1. Description

Another way to reduce the training time when dealing with very long sequences of data is by using a hierarchical subsampling network. Subsampling means that rather than operating on a single training example at a time, the network operates on blocks of k training examples. For a language model, this typically means taking k consecutive words in the text simultaneously as input rather than a single word. The subsampling is hierarchical when multiple hidden layers are arranged in a hierarchy such that in each layer, the input is subsampled, processed by the recurrent hidden layer, and then passed on as output to the next layer (Graves and Schmidhuber, 2009). For each layer in the hierarchy of a hierarchical subsampling RNN, the forward pass equations are identical to those of a standard RNN (see Eqs. (1) and (7)) except that the sum over input units is replaced by the sum over sums over the subsampling block.

Since each subsampling operation decreases the length of the output sequence by a factor of k, hierarchical subsampling networks reduce both the computation cost of the hidden layers higher in the hierarchy, and the effective separation between points in the input sequence. Thus inputs that are widely separated at the bottom of the hierarchy are transformed to features that are close together at the top. Besides reducing the training time, this kind of recurrent neural networks thus also increases the context size.

A detailed account on hierarchical subsampling recurrent neural networks (HSRNN) can be found in Graves (2012). The gain in efficiency often comes at the cost of an increased difficulty of training the network due to the addition of the extra hidden layers.

6.4.2. Empirical evaluation

In Graves (2012), several experiments on speech and handwriting recognition with HSRNN are described. HSRNN were used in the offline Arabic handwriting recognition competition at the 2009 International Conference on Document Analysis and Recognition (Märgner and Abed, 2009) and they outperformed all other systems, both in terms of accuracy and speed. In the online Arabic handwriting recognition competition at ICDAR 2009 (Abed et al., 2009), HSRNN was surpassed by a recognition system developed by VisionObjects.[9] Two other competitions were won with the use of HSRNN, namely the French handwriting recognition competition at ICDAR 2009 (Grosicki and Abed, 2009) and the Farsi/Arabic character classification competition at ICDAR 2009.

6.5. Higher order gradient descent techniques

6.5.1. Description

Pure gradient-descent techniques for optimization generally suffer from slow convergence when the curvature of the error surface is different in different directions. In that situation, on the one hand the learning rate must be chosen small to avoid instability in the directions of high curvature, but on the other hand, this small learning rate might lead to unacceptably slow convergence in the directions of low curvature. A general remedy is to incorporate curvature information into the gradient descent process. This requires the calculation of the second-order derivatives, for which several approximative techniques have been proposed in the context of recurrent neural networks. These calculations are expensive, but can accelerate convergence especially near an optimum where the error surface can be reasonably approximated by a quadratic function (Jaeger, 2002). For example, the extended Kalman filter can be used as a second-order gradient descent algorithm to estimate optimal weights for an RNN, as explained in Jaeger (2002).

6.5.2. Empirical evaluation

Using higher order gradient descent techniques only affects the training algorithm and does not create another kind of RNN, in contrast to most models discussed above and below. Thus we should not expect that using a higher order gradient descent technique results in a significantly better or worse solution. However, it can, of course, significantly

[9] http://www.visionobjects.com/.

influence the training time. Unfortunately, we are not aware of the training of RNN applied to statistical language modeling using higher order gradient descent techniques.

6.6. Reservoir computing

6.6.1. Description

In 2001 and 2002 a fundamentally new approach to RNN design and training was proposed independently by Maass under the name of Liquid State Machines (Maass et al., 2002) and by Jaeger under the name of Echo State Networks (Jaeger, 2001). This approach is now increasingly referred to as Reservoir Computing (RC). The RC paradigm avoids some shortcomings of gradient-descent RNN training, especially the long training times.

The RC model can be seen as an alternative version of a basic RNN in the following way (Lukoˇsevi ˇcius and Jaeger, 2009):

1. A recurrent neural network is randomly created and remains unchanged during training. This RNN is called the reservoir. It is passively excited by the input signal and maintains in its state a nonlinear transformation of the input history.
2. The desired output signal is generated as a linear combination of the neurons' signals from the input-excited reservoir. This linear combination is obtained by linear regression, using the teacher signal as a target.

Another view is to compare them to kernel methods. The key idea behind kernel methods is to pre-process the input by applying a form of transformation from the input space to the feature space, where the latter is usually far higher dimensional than the former. The classification or regression is then performed in this feature space. The power of kernel methods lies mainly in the fact that this transformation does not need to be computed explicitly, which would be either too costly or simply impossible. This transformation into a higher-dimensional space is similar to the functionality of the reservoir, but there exist two major differences: first, in the case of reservoirs the transformation is explicitly computed, and second kernels are not equipped to cope with temporal signals (Schrauwen et al., 2007).

The state equations describing a RC system are given by

$$\mathbf{x}(t+1) = f(W^{res}_{res}\mathbf{x}(t) + W^{res}_{inp}\mathbf{u}(t) + W^{res}_{out}\mathbf{y}(t) + W^{res}_{bias})$$
$$\hat{\mathbf{y}}(t+1) = W^{out}_{res}\mathbf{x}(t+1) + W^{out}_{inp}\mathbf{u}(t) + W^{out}_{out}\mathbf{y}(t) + W^{out}_{bias}$$

All weights matrices into the reservoir, W^{res}, are initialized at random, while all connections to the output units, W^{out}, are trained. Since only weights associated to the connections from the reservoir to the output units are updated, training can be done very fast. A graphical illustration of a RC system is given in Fig. 5.

6.6.2. Empirical evaluation

The reservoir computing technique was applied to RNN in an isolated spoken digits recognition task, where improvement in terms of WER on benchmark data was observed from 0.6% for the previous best system to 0.2% for the RNN with reservoir computing (Verstraeten et al., 2006). More generally, in Jaeger (2001) some encouraging observations are described, such as the fact that RC is computationally universal for continuous-time, continuous-value real-time systems modeled with bounded resources (including time and value resolution), they are biologically plausible (RC provides explanations of why biological brains can carry out accurate computations with an inaccurate and noisy physical substrate) and the advantages of RC in terms of extensibility and parsimony (new items are represented by new output units, which are appended to the previously established output units of a given reservoir. Since the output weights of different output units are independent of each other, catastrophic interference is a non issue). Although RC has shown to be a valid alternative to feedforward and traditional RNN, sometimes showing good accuracy and reliability compared to even the best recurrent methods for some applications (see, for example, (de Vos, 2012)), it is generally acknowledged that further research is crucial to validate the usefulness of this technique (as recognized in, e.g., (de Vos, 2012; Jaeger, 2001)).

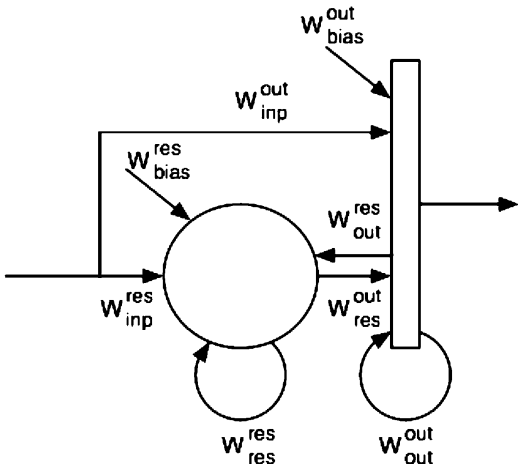

Fig. 5. Reservoir computing network.
Taken from Schrauwen et al. (2007).

7. Methods to dynamically adapt the number of hidden neurons

7.1. Growing and pruning neural networks

7.1.1. Description

Growing and pruning methods have been developed to dynamically adapt the number of hidden neurons during training.

In growing methods, new hidden units are added during the training process. One well known growing strategy is cascade correlation, where the network begins with only input and output neurons. During the training process, hidden neurons are selected from a pool of candidates and added to the hidden layer. The strategy refers to a cascade because the output from all neurons already in the network feed into new neurons. As new neurons are added to the hidden layer, the learning algorithm attempts to maximize the magnitude of the correlation between the new neurons' output and the residual error of the network which one is trying to minimize. The strategy was originally developed for feedforward neural networks (Fahlman and Lebiere, 1989), but it was straightforwardly extended to recurrent neural networks (Fahlman, 1991). A drawback of growing methods is that the network can get trapped in local minima and that they are sensitive to initial conditions (Maldonado and Manry, 2002). An overview of some growing methods can be found in Qiang et al. (2010).

In pruning methods, one starts with a large network and hidden units and/or weights that are less useful, according to some criterion, are removed. A very simple, brute-force pruning method is to set a certain weight to zero and to evaluate the resulting change in error. If the error increases too much then the weight is restored to its original value, otherwise it is removed. This procedure is repeated for all weights. It is clear that this requires a prohibitively amount of computation time. More advanced pruning algorithms can be found in Reed (1993).

7.1.2. Empirical evaluation

To our knowledge growing or pruning recurrent neural networks have not yet been applied to statistical language modeling.

7.2. Genetic algorithms

7.2.1. Description

Genetic algorithms (GA) are a global search technique based on the main principles of Darwinian evolution. A GA starts with a population of randomly generated solutions, called chromosomes, and advances towards better

solutions by applying genetic operators. The 'genetic information' of an individual is encoded in a bit string of fixed length, called the parameter string. The number of hidden neurons, for example, can be considered as a variable in the parameter string, thus avoiding the need to fix this number in advance. Unlike the growing or pruning strategy, there is not one single network at the start of the training, but a set of networks. Individual networks evolve by mutation, where some values(s) in the parameter string are changed, and by crossover, where two parameter strings are combined into a new parameter string. The process, i.e. training, is finished when one parameter string remains. Studies suggest that this process ensures a more global search than the forward-backward pass training algorithm used in traditional neural networks (White, 1993). An application of genetic algorithms to RNN can be found in, for example, Blanco et al. (2000). In this work a genetic algorithm is presented that is capable of obtaining not only the optimal topology of a recurrent neural network but also the least number of connections necessary.

In Schmidhuber et al. (2001) it was shown that gradient-based learning algorithms can yield suboptimal results when rough error surfaces lead to inescapable local minima. The authors observed that many RNN problems involving long-term dependencies that were considered challenging benchmarks in the 1990s, turned out to be trivial in that they could be solved by random weight guessing. In their words: "these problems were difficult only because learning relied solely on gradient information; there was actually a high density of solutions in the weight space, but the error surface was too rough to be exploited using the local gradient. By repeatedly selecting weights at random, the network does not get stuck in a local minimum, and eventually happens upon one of the plentiful solutions." To overcome this drawback, they developed EVOLINO (EVOlution of recurrent systems with LINear Outputs), where weights from the input units to the hidden units are evolved, while computing optimal linear mappings from the hidden layer to the output layer using methods such as pseudo-inverse-based linear regression or support vector machines. This can drastically reduce computation time, since only a subset of all weights undergo the evolution process, and since gradient information is used to find good directions in the search space. EVOLINO allows linear models to handle much of the problem, leaving the evolution process to handle non-linearity and recurrence. The authors applied this method to the LSTM architecture, an extension of the basic RNN (see further, Section 8.2).

7.2.2. Empirical evaluation

The EVOLINO model was applied to context-sensitive languages, i.e. languages that cannot be recognized by deterministic finite-state automata, and are therefore more complex in some respects than regular languages. In general, determining whether a string of symbols belongs to a context-sensitive language requires remembering all the symbols in the string seen so far, which rules out the use of non-recurrent architectures. The authors experimented with 8 different training sets and found that the EVOLINO model shows a dramatic improvement in terms of generalization over the standard, gradient-based LSTM described in Gers and Schmidhuber (2001). LSTM is described in Section 8.2 below.

8. Methods to increase the context size

8.1. Bidirectional neural networks

8.1.1. Description

The basic recurrent neural network described in (7)–(10) only takes the 'left context' into account, i.e. the context contained in previous training examples. The RNN can be easily extended to include both left and right context by introducing an extra hidden layer. This extra hidden layer, called the backward hidden layer, is only connected to the output layer and thus does not interact with the original hidden layer, which is called the forward hidden layer to distinguish it from the backward hidden layer. We extend our notation by using the superscripts F and B to refer to a variable or parameter that is used in the forward resp. backward hidden layer. The bidirectional recurrent neural network (BRNN) is then described as:

$$
\begin{aligned}
a_j^F(k,t) &= \sum_{i=1}^{|V|} \alpha_{ji}^F(k)\, x_i(k,t) + \sum_{i=1}^{n_H} \rho_{ji}^F(k)\, h_i^F(k,t-1), \quad j = 1,\ldots,n_H \\
h_j^F(k,t) &= F(a_j^F(k,t)), \quad j = 1,\ldots,n_H \\
a_j^B(k,t) &= \sum_{i=1}^{|V|} \alpha_{ji}^B(k)\, x_i(k,t) + \sum_{i=1}^{n_H} \rho_{ji}^B(k)\, h_i^B(k,t+1), \quad j = 1,\ldots,n_H \\
h_j^B(k,t) &= F(a_j^B(k,t)), \quad j = 1,\ldots,n_H \\
b_j(k,t) &= \sum_{i=1}^{n_H} \beta_{ji}^F(k)\, h_i^F(k,t) + \sum_{i=1}^{n_H} \beta_{ji}^B(k)\, h_i^B(k,t), \quad j = 1,\ldots,|V| \\
o_j(k,t) &= G(b_j(k,t)), \quad j = 1,\ldots,|V|
\end{aligned}
\tag{18}
$$

Due to the term $h_i^F(k,t-1)$ in Eq. (18) all previous training examples from the current batch ($\mathbf{x}(k,1)$, $\mathbf{y}(k,1)$), …, ($\mathbf{x}(k,t-1)$, $\mathbf{y}(k,t-1)$) are taken into account in calculating the network's output, while the term $h_i^B(k,t+1)$ in Eq. (18) ensures that all subsequent training examples ($\mathbf{x}(k,t+1)$, $\mathbf{y}(k,t+1)$), …, ($\mathbf{x}(k,|B(k)|)$, $\mathbf{y}(k,|B(k)|)$) are also taken into account. The recursiveness requires the initialization of the h-values. For example, we might initialize $h_i^F(k,0) = 0, i = 1,\ldots,n_H$ and $h_i^B(k,|B(k)|+1) = 0, i = 1,\ldots,n_H$. Other initialization methods can be derived from Section 5.7. Training the network will now require to calculate δ-values associated with the forward *and* backward hidden layer. While the δ-values related to the forward hidden layer still follow equation (A.3), the δ-values related to the backward hidden layer, denoted as $\delta_j^{h,B}$, are calculated as follows

$$
\delta_j^{h,B}(k,t) = h_j^B(k,t)(1-h_j^B(k,t)) \left[\sum_{p=1}^{|V|} \delta_p^o(k,t)\, \beta_{pj}^B(k) + \sum_{p=1}^{n_H} \delta_p^{h,B}(k,t-1)\, \rho_{pj}^B(k) \right]
$$

The bidirectional neural network is illustrated in Fig. 6.

8.1.2. Empirical evaluation

Before BRNN arose, researchers simulated the behavior of what became later known as bidirectional recurrent neural networks by training two recurrent neural networks, one taking the left context into account and the other one incorporating the right context, and merging the outputs of both networks. Schuster (1999) applied BRNNs to the TIMIT speech database, which consists of 6300 sentences spoken by 630 speakers. The BRNN resulted in the best classification rate on a phoneme classification task (68.53%), outperforming a (unidirectional) recurrent neural

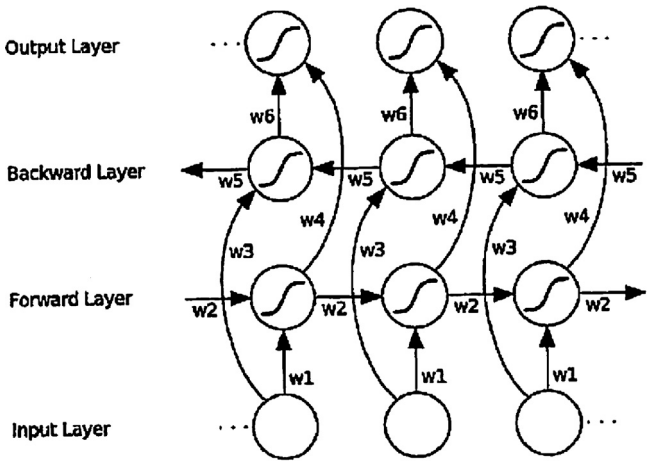

Fig. 6. Bidirectional neural network.
Taken from Graves (2012).

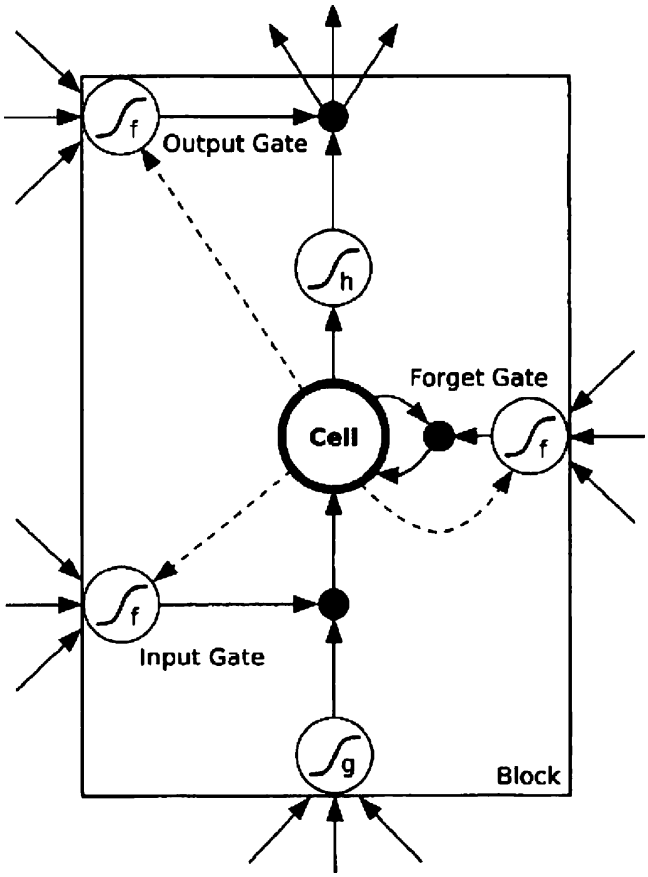

Fig. 7. Memory block in a LSTM network.
Taken from Graves (2012).

network, a mixture of two (unidirectional) recurrent neural networks and a feedforward neural network. However, in a phoneme sequence recognition task, none of the BRNNs achieved a better phoneme recognition rate than the regular RNN system described in Robinson (1994). Fukada et al. (1999) applied BRNNs to phoneme boundary estimation and found that BRNNs perform significantly better than a hidden Markov model and a multilayer perceptron-based method. Graves and Schmidhuber (2005) found bidirectional networks to be significantly more effective than unidirectional ones on the TIMIT speech database.

8.2. *Long short-term memory*

The long short-term memory (LSTM) contains recurrently connected subnets, also called memory blocks, and has been developed to increase the effective context size of RNN (Hochreiter and Schmidhuber, 1997). The architecture extends a basic RNN in that each hidden unit is replaced by a memory block. Each block contains one or more self-connected memory cells and three multiplicative units (called the input, output and forget gates) that provide continuous analogues of write, read and reset operations for the cells. The units allow the memory cells to store and access information over many training examples. For example, as long as the input gate has an activation value near zero, the activation of the cell will not be overwritten by the new inputs arriving in the network, and can therefore be made available to the net much later in the sequence, by opening the output gate. An illustration of a memory block is shown in Fig. 7.

The LSTM can be made bidirectional resulting in the bidirectional LSTM or BLSTM for short (Graves and Schmidhuber, 2005).

8.2.1. Empirical evaluation

In the aforementioned study by Graves and Schmidhuber (2005), see Section 8.1.2, several neural network architectures were compared to each other: feedforward neural networks, (unidirectional) RNN, BRNN, (unidirectional) LSTM and BLSTM. The researchers reported that bidirectional networks perform better than unidirectional ones (for RNN as well as LSTM), and that LSTM is slightly more accurate than RNN, BRNN and feedforward neural networks. The most important gain in using BLSTM appears to be in the reduction of training time. Whereas the three configurations of BLSTM that were used in the experiment required relatively few epochs before the network converged (15, 17 and 20), the two configurations of RNN required 120 and 139 epochs, the BRNN configuration 170 and the two feedforward neural networks 835 and 990. In particular, the BRNN took more than 8 times as long to converge as BLSTM, despite having more or less equal computational complexity per time-step. There was a similar time increase between the unidirectional LSTM and RNN, and the FNNs were slower still. In very recent work, Graves achieves a test set error of 17.7% on the TIMIT phoneme recognition benchmark with BLSTM, which is the best recorded score, to their knowledge (Graves et al., 2013).

BLSTM was also used in a recent experiment on language modeling for handwriting recognition (Frinken et al., 2012). Using a N-gram model with context length 2, the authors show that the various configurations of BLSTM in the experiment have word error rates between 22.2% and 22.4%, outperforming the baseline model which has word error rate 23.5%. This result appears to be statistically significant ($\alpha = 0.05$). However, as the bi-gram language model is very basic, this research does tell us much about the performance of BLSTM in handwriting recognition relative to state-of-the-art systems. Results of BLSTM applied on named entity recognition are also very moderate (Hammerton, 2003).

Another handwriting recognition experiment was done in Graves et al. (2009). A BLSTM model with connectionist temporal classification (CTC) output layer (Graves et al., 2006) was compared to a HMM (Hidden Markov Model), on a online and offline handwriting recognition task. The online and offline databases used were the IAM-OnDB (Liwicki and Bunke, 2005), a database acquired from a 'smart' whiteboard, and the IAM-DB (Marti and Bunke, 2002), a database acquired from scanned images of handwritten forms. To make the comparisons fair, the same online and offline preprocessing was used for both the HMM and recurrent systems. The recurrent neural network architecture substantially outperformed the HMM on both databases in terms of word accuracy with a relative error reduction of over 40% in some cases.

The authors provide some interesting possible explanations for the significant superiority of the recurrent model. Internal states of a standard HMM are discrete and univariate meaning that for an HMM with n states, only $O(\log n)$ bits of information about the past observation sequence are carried by the internal state, while RNNs, on the other hand, have a continuous, multivariate internal state (the hidden layer) whose information capacity grows linearly with its size. Another cited reason is that HMMs are generative models, while RNNs are discriminative models, which yield normalized label probabilities that can be used to assess prediction confidence, or to combine the outputs of several classifiers.

A speech recognition experiment was performed in Sundermeyer et al. (2013) to compare FNN and LSTM. The training data comprised 350 hours of manually transcribed broadcast news and broadcast conversational speech. The baseline back-off LM was trained on 1.6 billion running words for a vocabulary of size 200 K using Kneser-Ney smoothing. The experimental comparison of different neural network architectures was based on the five state-of-the-art French speech recognition systems from RWTH that were trained for the 2012 evaluation of the Quaero research project.[10] These obtained the best results among those systems evaluated on this task. The size of the hidden layer of the neural networks was varied between 300 and 500 units. For the LSTM architecture only a limited amount of training data could be used which comprised a subset of 27 M running words of in-domain data. Nevertheless, LSTM achieved perplexities similar to those of the back-off model. An ensemble (see further, Section 10.1) of two LSTMs outperformed the back-off model. The context-length of the FNN was increased until no further improvement was observed, resulting in context length 8, having perplexity 146.9 on the test set. In spite of the long context size of the FNN, the LSTM perplexities were lower by 15–17% relative compared to the feedforward architecture. In terms of WER none of the feedforward architectures performed as well as the LSTM architectures. On the test data, LSTM networks improved by 0.4% absolute over FNNs, even when an ensemble of two FNNs was considered.

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

8.3. Hierarchical subsampling networks

Hierarchical subsampling networks, discussed for their ability to reduce training time in Section 6.4, also provide a way of increasing the context size. See Section 6.4 for details.

9. Methods to reduce decoding time

Long-span language models that capture long-distance dependencies (e.g., language models built by considering the sentences in a discourse) are expected to be more powerful than low-order language models, but they bring the additional challenge of overcoming the computational complexity when decoding, that is, finding the most probable sequence of words given the trained language model for a specific discourse context in tasks such as automatic speech recognition (ASR), machine translation (MT), or optical character recognition (OCR).

9.1. Multi-pass strategies

9.1.1. Description

A standard practice when using language models is to carry out a first pass with a low-order *N*-gram model (e.g., trigram model) to generate a list of *K* best hypotheses and then to rescore the hypotheses using longer-span models. For instance, in a machine translation task, the focus then is less on producing the single best translation and more on being able to generate a rich space of possible translations that can be effectively exploited by subsequent post-processing and combination techniques. The large size of the search space means that it is sometimes impossible to apply the most sophisticated models to the full space of translation hypotheses. For this reason, it is common practice to generate a large subset of the most likely translations according to the so-called first-pass decoder, and then re-rank the hypotheses with more sophisticated models. A lattice is a space-efficient representation consisting of a large number of ranked translation alternatives and scores. The goal of lattice re-scoring is to re-rank translation hypotheses so that their ranking better reflects their quality.

But relying on a simple language model in the first pass might lead to suboptimal performance, so researchers actively study how to use a long-span language model such as a RNN model during this first pass and how to rescore the lattice of such a language model in a computationally tractable way. A multi-pass strategy for ASR was proposed in Deoras et al. (2011). This strategy cuts the lattice into many smaller self-contained lattices, which amounts to replacing the problem of global search over the lattice by series of local search problems over the small lattices in an iterative manner. A local neighborhood, i.e. a small lattice, can be constructed by considering sentences that are at, e.g., 1 word edit distance relative to a certain sentence where one of the words is 'confused', or as in Deoras et al. (2011), by considering the size of the neighborhood as a variable that depends on the confusability of the region. Once these small lattices have been produced, the many small re-scoring problems are performed by a recurrent neural network.

As an alternative to the above approaches it is possible to efficiently compute the probability of an *N*-gram as the free energy of a Restricted Boltzmann Machine (RBM), where the *N*-gram computation is integrated directly in a machine translation process, instead of using the language model in a re-ranking step (Niehues and Waibel, 2012).

9.1.2. Empirical evaluation

In Deoras et al. (2011) an experiment was performed using state-of-the-art acoustic models trained on the English Broadcast News corpus (430 hours of audio) provided by IBM (Chen et al., 2009). Two RNNs were trained, one on 58M tokens with 400 hidden units, and one on 400M tokens with 320 hidden units. A combination of both RNNs (i.e. an ensemble, see Section 10.1) was used for re-scoring (Deoras et al., 2010) and was compared against the K best list approach (Stolcke et al., 1997). A speed up technique based on the entropy of the lattice was used for this RNN combination. The K best list approach achieved its lowest possible WER after evaluating as many as 33.8K sentence hypotheses on average. The RNN combination obtained the same performance by evaluating just 1.6K sentence hypotheses, thus reducing the search space by a factor of 21.

Finch et al. (2012) updated their phrase-based machine transliteration system described in Finch et al. (2011). They added a re-scoring step with three re-scoring models: an RNN target language model, an RNN joint source-channel model, and a maximum entropy model. A 5-gram language model trained with Witten-Bell smoothing was used as baseline. In 9 out of the 15 experiments the updated model had lower PPL than the baseline model (experiments include

Arabic to English, Chinese to English, English to Hindi, etc.). They also tried an interpolated model, where the N-gram model was combined with the updated model, which outperformed each of the individual models.

9.2. Variational approximation

9.2.1. Description

Deoras (2011) and Deoras et al. (2011, 2013) proposed to approximate a long-span model using variational approximation techniques. Given a long-span model P, i.e. typically a sophisticated language model with complex statistical dependencies, the idea is to seek a simple and computationally tractable model $Q^{\star}$ that will be a good surrogate for P. Specifically, among all models Q of a certain class of tractable models, the idea is to seek the one that minimizes the Kullback-Leibler divergence (Cover and Thomas, 1991) from P. The class of tractable models in Deoras et al. (2013) is chosen as the family of distributions parameterized by N-grams, while the long-span complex model is chosen from the class of recurrent neural networks (although the presented method is general enough to be applied to other complex models).

In multi-pass strategies (see Section 9.1) complex language models are seldom used in the first pass of large vocabulary continuous speech recognition systems due to the prohibitive increase in the size of the sentence-hypotheses search space. Instead, N-gram language models are typically used for this purpose. Deoras et al. (2013) propose to use a variational approximation of a complex language model, such as RNN, for the first pass decoding and also for lattice re-scoring.

The first step in their procedure is to simulate text using P. To perform this step, the next word from the probability distribution as represented by P is sampled, conditioned on already generated words. This is repeated until the number of generated words reaches a predetermined value. In the next step, after having generated a sample corpus, the maximum likelihood estimate belonging to the class of N-gram models is selected based on this sample corpus. It is shown that maximizing the likelihood of the sampled text will have a Kullback-Leibler divergence with P of zero in the limit as the sampled text size goes to infinity, since the sampled text has as underlying distribution P.

9.2.2. Empirical evaluation

In Deoras et al. (2013) four extensive experiments are presented: (1) perplexity experiments on WSJ training data, (2) recognition on the MIT lectures corpus (Glass et al., 2007) using state-of-the-art acoustic models trained on the English Broadcast News corpus (430 hours of audio) provided by IBM (Chen et al., 2009); (3) two conversational speech recognition tasks: the transcription of multi party meetings, and of conversational telephone speech, (4) a variation on the Broadcast News setup where the original training data was huge, consisting of hundreds of millions of word tokens. The authors observe that with large amounts of simulated data, the variational model becomes better, resulting in improved performance over conventional N-gram models estimated from the original training data. The authors also find that in situations where it is expensive to change decoder models and parameters, the variational models can be used and achieve improved performance by just re-scoring the first pass lattices.

9.3. Reduction to simpler models

9.3.1. Description

Another alternative to decoding with RNNs is to first convert the RNN into an simpler model, such as an N-gram model, and then use that for decoding. This results in a faster model for decoding, at the cost of a model that only approximates the underlying probability distribution of the RNN.

Lecorvé and Motlicek (2012) propose to reduce RNNs to weighted finite state transducers (WFST). They fist create WFST states by applying K-means clustering to the continuous states of the RNN. This allows any word, represented as an RNN continuous state, to be mapped to one of the WFST states, where it is represented as the centroid of a K-means cluster. Weights on the edges between two WFST states are derived from the RNN by predicting the probability of observing the latter centroid given the earlier one. To produce a smaller WFST, edges are pruned based on their entropy and the relative change in probability before and after pruning.

Arisoy et al. (2014) propose to reduce FNNs to *N*-gram models by explicitly storing the probabilities of a selected set of *N*-grams. To decide which *N*-grams to store, they use entropy-based pruning, calculating the entropy of the *N*-gram model both with and without each *N*-gram, and pruning away *N*-grams whose removal results in an acceptably small difference in entropy. To calculate the *N*-gram model's backoff parameters, they either train FNNs with smaller histories (e.g., for a 4-gram model, a 3-gram FNN and a 2-gram FNN are trained), or they set the FNN activations only for the words in the smaller history. (Note that neither of these strategies are directly applicable to RNNs.)

A major issue with neural probabilistic language models including the ones based on RNN is that the probabilities are required to be normalized, and this normalization process is expensive because it considers all words in the vocabulary. There are a number of strategies to avoid this cost and that in some way simplify the model. One approach is to have the neural network output probabilities that are approximately normalized. Mnih and Teh (2012) propose a fast and simple algorithm based on noise contrastive estimation for training a neural based language model, where the normalization constant can be estimated as any other parameter of the model by using very simple variants of the log-bilinear model (Mnih and Kavukcuoglu, 2013). Another approach that reduces the computational complexity of the normalization of the neural network based language models uses short-lists to restrict the calculation of the language model probabilities to a subset of the vocabulary. This is, for example, done by Schwenk (2010), where language model probability estimation is performed in the framework of machine translation.

9.3.2. Empirical evaluation

Lecorvé and Motlicek (2012) evaluate a RNN reduced to a WFST in the speech recognition task of the NIST RT 2007 evaluation set. They reduce a RNN trained on 26.5M words with a 65K word vocabulary to a WFST with 512 clusters. They then use a multi-pass architecture to compare this WFST to a bigram model, decoding with the WFST or bigram model and then rescoring the lattice with the RNN. The WFST and bigram versions achieve comparable WER, demonstrating the plausibility of their WFST conversion. However, most current decoders use a larger n than 2 in their decoders, so further experiments would be necessary to validate these results.

Arisoy et al. (2014) evaluate an FNN converted to a backoff *N*-gram model on an English Broadcast News speech recognition task. They use a multi-pass architecture, decoding with either a regular *N*-gram model, or the FNN converted to an *N*-gram model, and then rescoring with the FNN. Using the FNN-derived model instead of the *N*-gram model for decoding results in a 1% absolute reduction in WER without rescoring, and a 0.4% absolute reduction in WER with rescoring. However, these results are for a FNN, and some additional work would be necessary to extend them to RNNs.

Vaswani et al. (2013) use the noise-contrastive estimation technique in combination with a neural probabilistic language model in a machine translation system both by reranking k-best lists and by direct integration into the decoder. The model in Schwenk (2010) yielded slight gains in BLUE scores for several machine translation tasks involving several languages and corpora.

10. Other extensions of the basic recurrent neural network

10.1. Ensemble methods

10.1.1. Description

An ensemble consists of a set of individually trained classifiers (such as neural networks or decision trees) whose predictions are combined when classifying novel instances. It is a widely assumed that an ensemble generally outperforms any of the individual classifiers, for which theoretical as well as experimental evidence can be found (Dietterich, 2000). It is crucial to appreciate that constructing a good ensemble entails more than arbitrarily defining various configurations of a certain class of models and combining their outputs to a single output. Theoretical (Hansen and Salamon, 1990) and empirical (Hashem, 1997) research show that a good ensemble requires the individual classifiers to be both accurate and making their errors on different parts of the input space. Consequently, methods have been developed to select appropriate individual classifiers. Two popular methods are bagging (Breiman, 1996) and boosting (Schapire et al., 1997).

Bagging is a bootstrap ensemble method that creates individuals for its ensemble by training each classifier on a random redistribution of the training set. Boosting attempts to produce new classifiers that are better able to predict examples for which the current ensemble's performance is poor. An extensive empirical study on bagging and boosting can be found in Opitz and Macline (1999).

10.1.2. Empirical evaluation

In Mikolov (2010) an ensemble consisting of 3 RNNs was compared to several state-of-the-art language models, namely the discriminative language model developed by Xu et al. (2009), the joint language model developed by Filimonov and Harper (2009), and the lattices generated by the AMI system (Hain et al., 2005). The evaluation was done on several standard speech recognition tasks with Wall Street Journal training data. It was found that the ensemble outperforms the other models in terms of WER.

In Section 5.13 we noticed that in the experiments described in Mikolov (2012), Mikolov et al. (2011) on the Penn Treebank data set, the basic RNN was only beaten by a state-of-the-art within and across sentence boundary LM. In the same study several basic RNNs, with different random initializations, were combined into an ensemble (where the output was calculated as the average of the outputs of the individual networks) and it was found that this ensemble performed significantly better than the within and across sentence boundary LM.

Buabin (2012) constructed an ensemble of RNNs with a boosting algorithm on a text document classification task. As training data the ModApte version of the Reuters news text corpus was taken, containing news text documents that appeared in the Reuters newswire in 1987. They found that the ensemble outperformed a single RNN.

10.2. Hybrid models

10.2.1. Description

Non-neural-network models have been combined with RNNs in the hope that such a hybrid model performs better than any of the individual models. In a hybrid model, models from different classes are combined, in contrast to ensemble methods, where models from the same class are combined.

It would lead us too far off topic to discuss all meaningful hybrid models where RNN is one of the involved models. We restrict our discussion to one particular hybrid model, which can serve as an example. The HMM-BLSTM is a hybrid model where a hidden Markov model (HMM) is combined with a BLSTM (see Section 8.2).

Hybrids of HMMs and neural networks were proposed by several researchers in the 1990s as a way of overcoming the drawbacks of HMMs. The introduction of neural networks was intended to provide more discriminating training, improved modeling of phoneme duration, richer, nonlinear function approximation and increased use of contextual information (Graves et al., 2005). In their simplest form, hybrid models used HMMs to align the segment classifications provided by the neural network into a temporal classification of the entire label sequence (Renals et al., 1994). Other architectures to hybridize HMMs and neural networks were developed. We focus here on the HMM-BLSTM model as described in Graves et al. (2005). Graves used an iterative approach, where the alignment provided by the HMM is used to successively retrain the neural network, as was originally done in Robinson (1994). This means that at each step the likelihood of the observations conditioned on the hidden states are found by dividing the posterior class probabilities defined by the network outputs by the prior class probabilities found in the data. These likelihoods are used to train the HMM. The alignment provided by the trained HMM is then used to define a new framewise training signal for the neural network, and the whole process is repeated until convergence.

10.2.2. Empirical evaluation

Graves performed experiments on a phoneme recognition task using the TIMIT speech database (Graves, 2012). The HMM-BLSTM model is compared to a HMM-LSTM hybrid model and to a context-dependent (triphone) and context-independent (mono-phone) HMM model. Sixty-one context-independent and 5491 tied context-dependent models were used. To ensure a fair comparison of the acoustic modeling capabilities of the systems, no prior linguistic information (such as a phonetic language model) was used.

It was found that HMM-BLSTM outperformed both context-dependent and context-independent HMMs, as well as the HMM-LSTM model. An interesting remark is that the hybrid systems had considerably fewer free parameters than the context-dependent HMMs, as a consequence of the high number of states required for HMMs to model contextual dependencies.

11. Conclusion and further research

In this paper, we presented a survey on the application of recurrent neural networks to statistical language modeling. The basic version of recurrent neural networks was discussed at great length. These networks have interesting characteristics, including their ability to learn vector representations for linguistic units and their capacity to take into account arbitrarily long sequences of preceding words. However, they suffer from drawbacks that limit their practical usefulness, including long training times, fixed numbers of hidden units, and limited use of distant context. The main extensions that have been developed to overcome these disadvantages have been outlined, together with an overview of their performance as reported in the literature.

We think that especially bidirectional neural networks, reservoir computing, class-based networks and hierarchical subsampling networks are promising techniques. Bidirectional neural networks have also the advantage that they automatically inherit the theoretical soundness of the basic (unidirectional) recurrent neural networks. However, these models cannot be used when only left context is available, such as in online speech recognition where words not yet spoken are unavailable. Reservoir computing ensures a dramatic decrease in training time and is theoretically well founded. However, further research is required to establish its performance. Class-based recurrent neural networks (which include hierarchical probabilistic networks and the SOUL network) reduce the vocabulary size $|V|$, which is a crucial parameter in the computational complexity. Hierarchical subsampling networks have achieved the top performance in several competitions. However, the competitions were oriented towards handwriting recognition, and further experiments are needed to evaluate their performance on other tasks.

We outline some further research directions:

- Evaluating the robustness of RNNs with respect to initial parameter values. This would make clear to which degree global search techniques, such as genetic algorithms, are required to perform the training.
- Reservoir computing has only very recently been applied to statistical language modeling. It is not clear yet whether its good performance in other fields can be extrapolated to language modeling.
- Although recurrent neural networks have been compared to a lot of models from other classes, an extensive comparison between different (extensions of) recurrent neural network models is still lacking.
- Although recurrent neural networks have shown promising performance on several tasks, most evaluations were done in terms of widely criticized evaluation measures: perplexity and word error rate. Evaluating the described recurrent neural network models according to more advanced measures seems a prerequisite to mark these models as truly remarkable statistical language models.

Acknowledgements

The research was financed by the EU FP7-296703 (FET-open call) project MUSE (Machine Understanding for interactive StorytElling). We thank the anonymous reviewers for their valuable remarks and suggestions.

Appendix A. Partial derivatives of error function with respect to the parameters

We derive the partial derivatives of the error function with respect to the parameters to be learned in several steps. We choose F as the sigmoid (5) and G as the softmax (6).

Step 1: Derivation of $\delta_j^o(k,t) = \frac{\partial E(k,t)}{\partial b_j(k,t)}$

We define $\delta_j^o(k,t) = \frac{\partial E(k,t)}{\partial b_j(k,t)}$. Let the target output of the network be given by $\mathbf{y}(k,t) = \boldsymbol{\omega}_\gamma$, which implies that $E(k,t) = -\ln o_\gamma(k,t)$. We then find that

$$\begin{aligned}
\delta_j^o(k,t) &\equiv \frac{\partial E(k,t)}{\partial b_j(k,t)} \\
&= -\frac{\partial \ln o_\gamma(k,t)}{\partial b_j(k,t)} \\
&= -\frac{1}{o_\gamma(k,t)} \frac{\partial o_\gamma(k,t)}{\partial b_j(k,t)} \\
&= -\frac{1}{o_\gamma(k,t)} \frac{\partial \frac{e^{b_\gamma(k,t)}}{\sum_{p=1}^{|V|} e^{b_p(k,t)}}}{\partial b_j(k,t)}
\end{aligned}$$

First suppose that $j = \gamma$. Then

$$\begin{aligned}
\frac{\partial \frac{e^{b_\gamma(k,t)}}{\sum_{p=1}^{|V|} e^{b_p(k,t)}}}{\partial b_j(k,t)} &= \frac{e^{b_\gamma(k,t)}}{\sum_{p=1}^{|V|} e^{b_p(k,t)}} - \left(\frac{e^{b_\gamma(k,t)}}{\sum_{p=1}^{|V|} e^{b_p(k,t)}} \right)^2 \\
&= o_\gamma(k,t) - o_\gamma(k,t)^2 \\
&= o_j(k,t) - o_j(k,t)^2
\end{aligned}$$

Now suppose that $j \neq \gamma$. Then

$$\begin{aligned}
\frac{\partial \frac{e^{b_\gamma(k,t)}}{\sum_{p=1}^{|V|} e^{b_p(k,t)}}}{\partial b_j(k,t)} &= -\frac{e^{b_\gamma(k,t)} e^{b_j(k,t)}}{\left(\sum_{p=1}^{|V|} e^{b_p(k,t)}\right)^2} \\
&= -o_\gamma(k,t)\, o_j(k,t)
\end{aligned}$$

Thus

$$\begin{aligned}
\delta_j^o(k,t) &= -\frac{1}{o_\gamma(k,t)} \left(o_\gamma(k,t) - o_\gamma(k,t)^2 \right) = -1 + o_\gamma(k,t) \quad \text{if} \quad j = \gamma \\
&= -\frac{1}{o_\gamma(k,t)} \left(-o_\gamma(k,t)\, o_j(k,t) \right) \qquad = o_j(k,t) \quad \text{if} \quad j \neq \gamma
\end{aligned}$$

This can be shortly written as

$$\delta_j^o(k,t) = o_j(k,t) - \Delta_j(k,t)$$

Step 2: Derivation of $\delta_j^h(k,t) = \frac{\partial E(k,t)}{\partial a_j(k,t)}$

We notice that

$$\begin{aligned}
a_j(k,t+1) &= \sum_{i=1}^{|V|} \alpha_{ji}(k)\, x_i(k,t+1) + \sum_{i=1}^{n_H} \rho_{ji}(k)\, h_i(k,t), \quad j = 1, \ldots, n_H \\
&= \sum_{i=1}^{|V|} \alpha_{ji}(k)\, x_i(k,t+1) + \sum_{i=1}^{n_H} \rho_{ji}(k)\, F(a_i(k,t)).
\end{aligned}$$

Thus $E(k,t)$ also depends on $a_j(k,t)$ through $a_p(k,t+1)$, $p=1,\ldots,n_H$. We thus have the following:

$$\begin{aligned}\delta_j^h(k,t) &= \sum_{p=1}^{|V|} \frac{\partial E(k,t)}{\partial b_p(k,t)} \frac{\partial b_p(k,t)}{\partial a_j(k,t)} + \sum_{p=1}^{n_H} \frac{\partial E(k,t)}{\partial a_p(k,t+1)} \frac{\partial a_p(k,t+1)}{\partial a_j(k,t)} \\ &= \sum_{p=1}^{|V|} \delta_p^o(k,t) \frac{\partial b_p(k,t)}{\partial a_j(k,t)} + \sum_{p=1}^{n_H} \delta_p^h(k,t+1) \frac{\partial a_p(k,t+1)}{\partial a_j(k,t)}\end{aligned}$$

From

$$\begin{aligned}\frac{\partial b_p(k,t)}{\partial a_j(k,t)} &= \sum_{i=1}^{n_H} \beta_{pi}(k) \frac{\partial h_i(k,t)}{\partial a_j(k,t)} \\ &= \beta_{pj}(k,t) \frac{\partial h_j(k,t)}{\partial a_j(k,t)} \\ &= \beta_{pj}(k,t)\, F'(a_j(k,t))\end{aligned}$$

and

$$\begin{aligned}\frac{\partial a_p(k,t+1)}{\partial a_j(k,t)} &= \sum_{p=1}^{n_H} \rho_{pi}(k) \frac{dF(a_i(k,t))}{da_j(k,t)} \\ &= \rho_{pj}(k)\, F'(a_j(k,t))\end{aligned}$$

we find that

$$\delta_j^h(k,t) = F'(a_j(k,t)) \left[\sum_{p=1}^{|V|} \delta_p^o(k,t)\, \beta_{pj}(k) + \sum_{p=1}^{n_H} \delta_p^h(k,t+1)\, \rho_{pj}(k) \right]$$

It is noticed that to calculate $\delta_j^h(k,t)$ we need $\delta_p^h(t+1)$, $p=1,\ldots,n_H$. Thus the values $\delta_j^h(k,t)$ are to be calculated recursively.

Choosing $F(x)$ as the sigmoid (5) we have that $F'(x)=F(x)(1-F(x))$ and thus

$$\delta_j^h(k,t) = F(a_j(k,t))(1-F(a_j(k,t))) \left[\sum_{p=1}^{|V|} \delta_p^o(k,t)\, \beta_{pj}(k) + \sum_{p=1}^{n_H} \delta_p^h(k,t+1)\, \rho_{pj}(k) \right] \tag{A.1}$$

$$= h_j(k,t)(1-h_j(k,t)) \left[\sum_{p=1}^{|V|} \delta_p^o(k,t)\, \beta_{pj}(k) + \right. \tag{A.2}$$

$$\left. \sum_{p=1}^{n_H} \delta_p^h(k,t+1)\, \rho_{pj}(k) \right] \tag{A.3}$$

Step 3: Some other derivatives

It is easy to derive that

$$\begin{aligned}\frac{\partial a_j(k,t)}{\partial \alpha_{ji}(k)} &= x_i(k,t) \\ \frac{\partial a_j(k,t)}{\partial \rho_{ji}(k)} &= h_i(k,t-1) \\ \frac{\partial b_j(k,t)}{\partial \beta_{ji}(k)} &= h_i(k,t)\end{aligned}$$

Step 4: Partial derivatives of error function with respect to the parameters

We then find:

$$\begin{aligned}\frac{\partial E(k,t)}{\beta_{ji}(k)} &= \frac{\partial E(k,t)}{\partial b_j(k,t)}\frac{\partial b_j(k,t)}{\partial \beta_{ji}(k)} \\ &= \delta_j^o(k,t)\, h_i(k,t) \\ &= (o_j(k,t) - \Delta_j(k,t))\, h_i(k,t)\end{aligned} \tag{A.4}$$

$$\begin{aligned}\frac{\partial E(k,t)}{\rho_{ji}(k)} &= \frac{\partial E(k,t)}{\partial a_j(k,t)}\frac{\partial a_j(k,t)}{\partial \rho_{ji}(k)} \\ &= \delta_j^h(k,t)\, h_i(k,t-1)\end{aligned} \tag{A.5}$$

$$\begin{aligned}\frac{\partial E(k,t)}{\alpha_{ji}(k)} &= \frac{\partial E(k,t)}{\partial a_j(k,t)}\frac{\partial a_j(k,t)}{\partial \alpha_{ji}(k)} \\ &= \delta_j^h(k,t)\, x_i(k,t)\end{aligned} \tag{A.6}$$

and thus over one batch:

$$\frac{\partial E(k)}{\beta_{ji}(k)} = \sum_{t=1}^{|B(k)|} \left(o_j(k,t) - \Delta_j(k,t)\right)\, h_i(k,t) \tag{A.7}$$

$$\frac{\partial E(k)}{\rho_{ji}(k)} = \sum_{t=1}^{|B(k)|} \delta_j^h(k,t)\, h_i(k,t-1) \tag{A.8}$$

$$\frac{\partial E(k)}{\alpha_{ji}(k)} = \sum_{t=1}^{|B(k)|} \delta_j^h(k,t)\, x_i(k,t) \tag{A.9}$$

References

Abed, H.E., Märgner, V., Kherallah, M., Alimi, A.M., 2009. Online Arabic handwriting recognition competition. In: Proceedings of the 10th International Conference on Document Analysis and Recognition, pp. 1388–1392.

Ahmed, F., De Luca, E.W., Nürnberger, A., 2009. Revised *N*-gram based automatic spelling correction tool to improve retrieval effectiveness. Polibits 40, 39–48.

Almeida, L.B., Langlois, T., Amaral, J.D., Plankhov, A., 1998. Parameter adaptation in stochastic optimization. In: Saad, D. (Ed.), On-Line Learning in Neural Networks. Cambridge University Press, pp. 111–134.

Arisoy, E., Chen, S.F., Ramabhadran, B., Sethy, A., 2014. Converting neural network language models into back-off language models for efficient decoding in automatic speech recognition. IEEE/ACM Trans. Audio Speech Lang. Process. 22 (1), 184–192.

Bahl, L.R., Jelinek, F., Mercer, R., 1983. A maximum likelihood approach to continuous speech recognition. IEEE Trans. Pattern Anal. Mach. Intell., 179–190.

Balkin, S.D., 1997. Using recurrent neural network for time series forecasting. In: Proceedings of the International Symposium on Forecasting, Barbados.

Bengio, Y., Ducharme, R., Vincent, P., 2003. A neural probabilistic language model. J. Mach. Learn. Res. 3, 1137–1155.

Blanco, A., Delgado, M., Pegalajar, M.C., 2000. A genetic algorithm to obtain the optimal recurrent neural network. Int. J. Approx. Reason. 23 (1), 67–83.

Breiman, L., 1996. Stacked regressions. Mach. Learn. 24 (1), 49–64.

Brown, P.F., deSouza, P.V., Mercer, R.L., Pietra, V.J.D., Lai, J.C., 1992. Class-based *N*-gram models for natural language. Comput. Linguist. 18, 467–479.

Brown, P.F., Della Pietra, V.J., Della Pietra, S.A., Mercer, R.L., 1993. The mathematics of statistical machine translation: parameter estimation. Comput. Linguist. 19 (2), 263–311.

Buabin, E., 2012. Boosted hybrid recurrent neural classifier for text document classification on the Reuters news text corpus. Int. J. Mach. Learn. Comput. 2 (5), 588–592.

Chelba, C., Brants, T., Neveitt, W., Xu, P., 2010. Study on interaction between entropy pruning and Kneser-Ney smoothing. In: Proceedings of Interspeech, pp. 2242–2245.

Chen, S., Beeferman, D., Rosenfeld, R., 1998. Evaluation metrics for language models. In: Proceedings of the DARPA Broadcast News Transcription and Understanding Workshop.
Chen, S.F., Mangu, L., Ramabhadran, B., Sarikaya, R., Sethy, A., 2009. Scaling shrinkage-based language models. In: Proceedings of IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU), pp. 299–304.
Chen, Y., Perozzi, B., Al-Rfou, R., Skiena, S., 2013. The expressive power of word embeddings. Technical report. Cornell University Library arXiv:1301.3226v4.
Church, K.W., 1988. A stochastic parts program and noun phrase parser for unrestricted text. In: Proceedings of the Second Conference on Applied Natural Language Processing, pp. 136–143.
Clarkson, P., Robinson, T., 1998. The applicability of adaptive language modelling for the broadcast news task. In: Proceedings 5th International Conference on Spoken Language Processing, pp. 233–236.
Clarkson, P., Robinson, T., 2001. Improved language modelling through better language model evaluation measures. Comput. Speech Lang. 15 (1), 39–53.
Collobert, R., Weston, J., 2008. A unified architecture for natural language processing: deep neural networks with multitask learning. In: Proceedings of the 25th International Conference on Machine learning, pp. 160–167.
Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., Kuksa, P., 2011. Natural language processing (almost) from scratch. J. Mach. Learn. Res. 12, 2493–2537.
Cover, T., Thomas, J., 1991. Elements of Information Theory. John Wiley and Sons Inc.
Deoras, A., Filimonov, D., Harper, M., Jelinek, F., 2010. Model combination for speech recognition using empirical Bayes risk minimization. In: Proceedings of the IEEE Workshop on Spoken Language Technology.
Deoras, A., 2011. Search and Decoding Strategies For Complex Lexical Modeling in LVCSR. Johns Hopkins University (PhD dissertation).
Deoras, A., Mikolov, T., Kombrink, S., Karafiat, M., Khudanpur, S., 2011. Variational approximation of long-span language models for LVCSR. In: Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 5532–5535.
Deoras, A., Mikolov, T., Church, K., 2011. A fast re-scoring strategy to capture long-distance dependencies. In: EMNLP'11 Proceedings of the Conference on Empirical Methods in Natural Language Processing, pp. 1116–1127.
Deoras, A., Mikolov, T., Karafiát, M., Khudanpur, S., 2011. Variational approximation of long-span language models in LVCSR. In: Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 5532–5535.
Deoras, A., Mikolov, T., Kombrink, S., Church, K., 2013. Approximate inference: a sampling based modeling technique to capture complex dependencies in a language model. Speech Commun. 55 (1), 162–177.
de Novais, E.M., Tadeu, T.D., Paraboni, I., 2010. Improved text generation using *N*-gram statistics. In: Proceedings of the 12th Ibero-American Conference on Advances in Artificial Intelligence, pp. 316–325.
Deschacht, K., De Belder, J., Moens, M.-F., 2012. The latent words language model. Comput. Speech Lang. 26 (5), 384–409.
de Vos, N.J., 2012. Reservoir computing as an alternative to traditional artificial neural networks in rainfall-runoff modelling. Hydrol. Earth Syst. Sci., Discussion 9.
Dietterich, T.G., 2000. Ensemble methods in machine learning. In: Proceedings of the First International Workshop on Multiple Classifier Systems, pp. 1–15.
Elman, J.L., 1990. Finding structure in time. Cogn. Sci. 14 (2), 179–211.
Emami, A., Jelinek, F., 2005. Random clusterings for language modeling. In: Proceedings of the International Conference on Acoustics, Speech and Signal Processing, pp. 581–584.
Fahlman, S.E., Lebiere, C., 1989. The cascade-correlation learning architecture. Technical Report. Carnegie Mellon University.
Fahlman, S.E., 1991. The recurrent cascade-correlation learning architecture. Technical Report. Carnegie Mellon University.
Fellbaum, C., 1998. WordNet: An Electronic Lexical Database. MIT Press.
Fernández-Redondo, M., Hernández-Espinosa, C., 2000. A comparison among weight initialization methods for multilayer feedforward networks. In: Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks, pp. 543–548.
Filimonov, D., Harper, M., 2009. A joint language model with fine-grain syntactic tags. In: Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing, pp. 1114–1123.
Finch, A., Dixon, P., Sumita, E., 2011. Integrating models derived from non-parametric Bayesian co-segmentation into a statistical machine transliteration system. In: Proceedings of the Named Entities Workshop, pp. 23–27.
Finch, A., Dixon, P., Sumita, E., 2012. Rescoring a phrase-based machine transliteration system with recurrent neural network language models. In: NEWS'12 Proceedings of the 4th Named Entity Workshop, pp. 47–51.
Frinken, V., Zamora-Martinez, F., Espana-Boquera, S., Castro-Bleda, M.J., Fischer, A., Bunke, H., 2012. Long-short term memory neural networks language modeling for handwriting recognition. In: Int. Conf. Pattern Recognit, pp. 701–704.
Fukada, T., Schuster, M., Sagisaka, Y., 1999. Phoneme boundary estimation using bidirectional recurrent neural networks and its applications. Syst. Comput. Jpn. 30 (4), 20–30.
Gers, F.A., Schmidhuber, J., 2001. LSTM recurrent networks learn simple context free and context sensitive languages. IEEE Trans. Neural Netw. 12 (6), 1333–1340.
Gil, P., Cardoso, A., Palma, L., 2009. Estimating the number of hidden neurons in recurrent neural networks for nonlinear system identification. In: Proceedings of the IEEE International Symposium on Industrial Electronics, pp. 2053–2058.
Glass, J., Hazen, T., Cyphers, S., Malioutov, I., Huynh, D., Barzilay, R., 2007. Recent progress in MIT spoken lecture processing project. In: Proceedings Interspeech.
Graves, A., Schmidhuber, J., 2005. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. Neural Netw. 18 (5–6), 602–610.

Graves, A., Fernández, S., Schmidhuber, J., 2005. Bidirectional LSTM networks for improved phoneme classification and recognition. In: ICANN05 Proceedings of the 15th International Conference on Artificial Neural Networks: Formal Models and their Applications, pp. 799–804.
Graves, A., Fernández, S., Gomez, F., Schmidhuber, J., 2006. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In: Proceedings of the International Conference on Machine Learning, pp. 369–376.
Graves, A., Schmidhuber, J., 2009. Offline handwriting recognition with multidimensional recurrent neural networks. In: Koller, D., Schuurmans, D., Bengio, Y., Bottou, L. (Eds.), Advances in Neural Information Processing Systems. MIT Press.
Graves, A., Liwicki, M., Fernandez, S., Bertolami, R., Bunke, H., Schmidhuber, J., 2009. A novel connectionist system for unconstrained handwriting recognition. IEEE Trans. Pattern Anal. Mach. Intell. 31 (5), 855–868.
Graves, A., 2012. Supervised Sequence Labelling with Recurrent Neural Networks. Springer.
Graves, A., Mohamed, A., Hinton, G., 2013. Speech recognition with deep recurrent neural networks. In: Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2013).
Grosicki, E., Abed, H.E., 2009. ICDAR 2009 handwriting recognition competition. In: 10th International Conference on Document Analysis and Recognition, pp. 1398–1402.
Hain, T., et al., 2005. The 2005 AMI system for the transcription of speech in meetings. In: Proceedings Rich Transcription 2005 Spring Meeting Recognition Evaluation Workshop.
Hammerton, J., 2003. Named entity recognition with long short-term memory. In: Proceedings of CoNLL-2003, pp. 172–175.
Hansen, L., Salamon, P., 1990. Neural network ensembles. IEEE Trans. Pattern Anal. Mach. Intell. 12 (10), 993–1001.
Hashem, S., 1997. Optimal linear combinations of neural networks. Neural Netw. 10 (4), 599–614.
Heeman, P.A., 1999. POS tags and decision trees for language modeling. In: Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora, pp. 129–137.
Hochreiter, S., Schmidhuber, J., 1997. Long short-term memory. Neural Comput. 9 (8), 1735–1780.
Hochreiter, S., Bengio, Y., Frasconi, P., Schmiduber, J., 2001. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In: Kremer, S.C., Kolen, J.F. (Eds.), A Field Guide to Dynamical Recurrent Neural Networks. IEEE Press.
Huang, F., Ahuja, A., Downey, D., Yang, Y., Guo, Y., Yates, A., 2014. Learning representations for weakly supervised natural language processing tasks. Comput. Linguist. 40 (1), 85–120.
Huang, Z., Zweig, G., Levit, M., Dumoulin, B., Oguz, B., Chang, S., 2013. Accelerating recurrent neural network training via two stage classes and parallelization. In: IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU), pp. 326–331.
Iyer, R., Ostendorf, M., Meteer, M., 1997. Analyzing and predicting language model improvements. In: Proceedings of the IEEE Workshop on Automatic Speech Recognition and Understanding.
Jaeger, H., 2001. The "echo state" approach to analysing and training recurrent neural networks. GMD Report 148. German National Research Center for Information Technology.
Jaeger, H., 2002. Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the "echo state network" approach. GMD Report 159. German National Research Center for Information Technology.
Jelinek, F., 1998. Statistical Methods for Speech Recognition. A Bradford Book.
Jordan, M., 1986. Serial order: a parallel distributed processing approach. Technical report. University of California, San Diego.
Kirchhoff, K., Yang, M., 2005. Improved language modeling for statistical machine translation. In: Proceedings of the ACL Workshop on Building and Using Parallel Texts, pp. 125–128.
Kombrink, S., Mikolov, T., Karafiát, M., Burget, L., 2011. Recurrent neural network based language modeling in meeting recognition. In: In Proceedings of Interspeech 2011, pp. 2877–2880.
Kneser, R., Ney, H., 1995. Improved backing-off for *N*-gram language modeling. In: In Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, pp. 141–184.
Le, H.-S., Oparin, I., Messaoudi, A., Allauzen, A., Gauvain, J.-L., Yvon, F., 2011. Large vocabulary SOUL neural network language odels. In: Proceedings Interspeech, pp. 1469–1472.
Le, H.-S., Oparin, I., Allauzen, A., Gauvain, J.-L., Yvon, F., 2011. Structured output layer neural network language model. In: Proceedings of the International Conference on Acoustics, Speech and Signal Processing, pp. 5524–5527.
Le, H.-S., Allauzen, A., Yvon, F., 2012. Measuring the influence of long range dependencies with neural network language models. In: Proceedings of the NAACL-HLT 2012 Workshop, pp. 1–10.
Le, H.-S., 2013. Continuous Space Models with Neural Networks in Natural Language Processing. Université Paris Sud (PhD dissertation).
Lecorvé, G., Motlicek, P., 2012. Conversion of recurrent neural network language models to weighted finite state transducers for automatic speech recognition. In: Proceedings of Interspeech, pp. 1666–1669.
LeCun, Y., Bottou, L., Orr, G., Muller, K., 1998. Efficient BackProp. In: Orr, G., Muller, K. (Eds.), Neural Networks, Tricks of the Trade. Springer.
Liwicki, M., Bunke, H., 2005. IAM-OnDB – an on-line English sentence database acquired from handwritten text on a whiteboard. In: Proceedings of the 8th International Conference on Document Analysis and Recognition, pp. 956–961.
Lukoˇsevi ˇcius, M., Jaeger, H., 2009. Reservoir computing approaches to recurrent neural network training. Comput. Sci. Rev. 3 (3), 127–149.
Maass, W., Natschläger, T., Markram, H., 2002. Real-time computing without stable states: a new framework for neural computation based on perturbations. Neural Comput. 14 (11), 2531–2560.
Maldonado, F.J., Manry, M.T., 2002. Optimal pruning of feedforward neural networks based upon the Schmidt procedure. In: Conference Record of the Thirty-Sixth Asilomar Conference on Signals, Systems and Computers, pp. 1024–1028.
Manning, C.D., Schuetze, H., 1999. Foundations of Statistical Natural Language Processing. MIT Press, Cambridge, MA.
Märgner, V., Abed, H.E., 2009. ICDAR 2009 Arabic handwriting recognition competition. In: Proceedings of the10th International Conference on Document Analysis and Recognition, pp. 1383–1387.

Marichal, R., Pi neiro, J.D., González, E., Torres, J., 2007. Novel recurrent neural network weight initialization strategy. In: Proceedings of the World Congress on Engineering and Computer Science, pp. 537–548.
Marti, U.-V., Bunke, H., 2002. The IAM-database: an English sentence database for offline handwriting recognition. Int. J. Doc. Anal. Recognit. 5 (1), 39–46.
Martin, S.C., Liermann, J., Ney, H., 1997. Adaptive topic-dependent language modelling using word-based varigrams. In: Proceedings of Eurospeech, pp. 1447–1450.
Matignon, R., 2005. Neural Network Modeling Using SAS Enterprise Miner. AuthorHouse.
Mikolov, T., 2010. Recurrent neural network based language model. In: Proceedings of the 11th Annual Conference of the International Speech Communication Association, pp. 1045–1048.
Mikolov, T., Deoras, A., Povey, D., Burget, L., Cernocky, J.H., 2011. Strategies for training large scale neural network language models. In: IEEE Automatic Speech Recognition and Understanding Workshop.
Mikolov, T., Kombrink, S., Burget, L., Cernocky, J.H., Khudanpur, S., 2011. Extensions of recurrent neural network language model. In: ICASSP 2011.
Mikolov, T., Deoras, A., Kombrink, S., Burget, L., Cernocky, J.H., 2011. Empirical evaluation and combination of advanced language modeling techniques. In: Proceedings of Interspeech, pp. 605–608.
Mikolov, T., 2012. Statistical Language Models Based on Neural Networks. Brno University of Technology (PhD dissertation).
Mikolov, T., Yih, W.T., Zweig, G., 2013. Linguistic regularities in continuous space word representations. In: Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pp. 746–751.
Mikolov, T., Chen, K., Corrado, G., Dean, J., 2013. Efficient estimation of word representations in vector space. Cornell University Library arXiv:1301.3781.
Mnih, A., Kavukcuoglu, K., 2013. Learning word embeddings efficiently with noise-contrastive estimation. Adv. Neural Inf. Process. Syst. 26, 2265–2273.
Mnih, A., Teh, Y.W., 2012. A fast and simple algorithm for training neural probabilistic language models. In: Proceedings of the 29th International Conference on Machine Learning, pp. 1751–1758.
Momtazi, S., Faubel, F., Klakow, D., 2010. Within and across sentence boundary language model. In: Proceedings of Interspeech, pp. 1800–1803.
Moore, R.C., 2009. Improved smoothing for *N*-gram language models based on ordinary counts. In: Proceedings of the ACL-IJCNLP 2009 Conference Short Papers, pp. 349–352.
Morin, F., Bengio, Y., 2005. Hierarchical probabilistic neural network language model. In: Proceedings eAISTATS'05, pp. 246–252.
Niehues, J., Waibel, A., 2012. Continuous space language models using restricted Boltzmann machines. In: Proceedings of the International Workshop on Spoken Language Translation, pp. 164–170.
Och, F.J., Ney, H., 2002. Discriminative training and maximum entropy models for statistical machine translation. In: Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL), pp. 295–302.
Opitz, D., Macline, R., 1999. Popular ensemble methods: an empirical study. J. Artif. Intell. Res. 11, 169–198.
Panchal, G., Ganatra, A., Kosta, Y.P., Panchal, D., 2011. Behaviour analysis of multilayer perceptrons with multiple hidden neurons and hidden layers. Int. J. Comput. Theory Eng. 3, 332–337.
Papineni, K., Roukos, S., Ward, T., Zhu, W.-J., 2002. BLEU: a method for automatic evaluation of machine translation. In: Proceedings of the 40th Annual Meeting on Association for Computational Linguistics, pp. 311–318.
Peters, J., Klakow, D., 1999. Compact maximum entropy language models. In: Proceedings of the IEEE Workshop on Automatic Speech Recognition and Understanding.
Polak, E., 1997. Optimization: Algorithms and Consistent Approximations. Springer-Verlag.
Potamianosa, G., Jelinek, F., 1998. A study of *N*-gram and decision tree letter language modeling methods. Speech Commun. 24 (3), 171–192.
Prechelt, L., 1997. Early stopping – but when? In: Neural Networks: Tricks of the Trade. Springer-Verlag.
Qiang, X., Cheng, G., Wang, Z., 2010. An overview of some classical growing neural networks and new developments. In: Proceedings of the 2nd International Conference on Education Technology and Computer, pp. 351–355.
Reed, R., 1993. Pruning algorithms – a survey. IEEE Trans. Neural Netw. 4 (5), 740–747.
Renals, S., Morgan, N., Bourlard, H., Cohen, M., Franco, H., 1994. Connectionist probability estimators in HMM speech recognition. IEEE Trans. Speech Audio Process. 2 (1), 161–174.
Robinson, A.J., 1994. An application of recurrent neural nets to phone probability estimation. IEEE Trans. Neural Netw. 5 (2), 298–305.
Rojas, R., 1996. Neural Networks: A Systematic Introduction. Springer-Verlag.
Rosenfeld, R., 1994. Adaptive Statistical Language Modeling: A Maximum Entropy Approach. Carnegie Mellon University (PhD dissertation).
Rosenfeld, R., 2000. Two decades of statistical language modeling: where do we go from here. In: Proceedings of the IEEE, pp. 1270–1278.
Rumelhart, D.E., Hinton, G., Williams, R.J., 1986. Learning internal representations by error propagation. In: Rumelhart, D.E., McClelland, J.L. (Eds.), Parallel Distributed Processing. MIT Press, pp. 318–362.
Schäfer, A.M., Zimmerman, H.G., 2007. Recurrent neural networks are universal approximators. Int. J. Neural Syst. 17 (4), 253–263.
Schapire, R., Freund, Y., Bartlett, P., Lee, W., 1997. Boosting the margin: a new explanation for the effectiveness of voting methods. In: Proceedings of the Fourteenth International Conference on Machine Learning, pp. 322–330.
Schmidhuber, J., Hochreiter, S., Bengio, Y., 2001. Evaluating benchmark problems by random guessing. In: Kremer, S.C., Kolen, J.F. (Eds.), A Field Guide to Dynamical Recurrent Neural Networks. IEEE Press.
Schrauwen, B., Verstraeten, D., Van Campenhout, J., 2007. An overview of reservoir computing: theory, applications and implementations. In: Proceedings of the 15th European Symposium on Artificial Neural Networks, pp. 471–482.
Schuster, M., 1999. On supervised learning from sequential data with applications for speech recognition. Nara Institute of Science and Technology (PhD dissertation).

Schwenk, H., 2010. Continuous space language models for statistical machine translation. Prague Bull. Math. Linguist. 93, 137–146.
Schwenk, H., Gauvain, J.-L., 2005. Training neural network language models on very large corpora. In: Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing, pp. 201–208.
Seung, S., 2002. Perceptron learning. Lecture. http://hebb.mit.edu/courses/9.641/2002/lectures/lecture03.pdf
Sheela, K.G., Deepa, S.N., 2013. Review on methods to fix number of hidden neurons in neural networks. Math. Prob. Eng., Article ID 425740.
Shi, Y., Zhang, W.-Q., Liu, J., Johnson, M.T., 2013. RNN language model with word clustering and class-based output layer. EURASIP J. Audio Speech Music Process. 22.
Snover, M., Dorr, B., Schwartz, R., Micciulla, L., Makhoul, J., 2006. A study of translation edit rate with targeted human annotation. In: Proceedings of Association for Machine Translation in the Americas, pp. 223–231.
Stolcke, A., König, Y., Weintraub, M., 1997. Explicit word error minimization in N-best list rescoring. In: Proceedings of the 5th European Conference on Speech Communication and Technology.
Sundermeyer, M., Oparin, I., Gauvain, J.-L., Freiberg, B., Schluter, R., Ney, H., 2013. Comparison of feedforward and recurrent neural network language models. In: Proceedings of the International Conference on Acoustics, Speech and Signal Processing, pp. 8430–8434.
Turian, J., Ratinov, L., Bengio, Y., 2010. Word representations: a simple and general method for semi-supervised learning. In: ACL.
Vaswani, A., Zhao, Y., Fossum, V., Chiang, D., 2013. Decoding with large-scale neural language models improves translation. In: Proceedings of Empirical Models in Natural Language Processing, pp. 1387–1392.
Verstraeten, D., Schrauwen, B., Stroobandt, D., 2006. Reservoir-based techniques for speech recognition. In: Proceedings of the IEEE International Joint Conference on Neural Networks, pp. 1050–1053.
Vinciarelli, A., Bengio, S., Bunke, H., 2004. Offline recognition of unconstrained handwritten texts using HMMs and statistical language models. IEEE Trans. Pattern Anal. Mach. Intell. 26 (6), 709–720.
Wang, S., Schuurmans, D., Peng, F., Zhao, Y., 2005. Combining statistical language models via the latent maximum entropy principle. Mach. Learn. 60 (1–3), 229–250.
White, D.W., 1993. GANNet: A Genetic Algorithm for Searching Topology and Weight Spaces in Neural Network Design. University of Maryland (PhD dissertation).
Xu, P., 2005. Random Forests and the Data Sparseness Problem in Language Modeling. Johns Hopkins University (PhD dissertation).
Xu, P., Karakos, D., Khudanpur, S., 2009. Self-supervised discriminative training of statistical language models. In: Proceedings of the IEEE Workshop on Automatic Speech Recognition & Understanding, pp. 317–322.
Xu, W., Rudnicky, A., 2000. Can artificial neural networks learn language models? In: Proceedings of 6th International Conference on Spoken Language Processing, pp. 202–205.