LSTM: A Search Space Odyssey

Klaus Greff, Rupesh K. Srivastava, Jan Koutník, Bas R. Steunebrink, and Jürgen Schmidhuber

Abstract—Several variants of the long short-term memory (LSTM) architecture for recurrent neural networks have been proposed since its inception in 1995. In recent years, these networks have become the state-of-the-art models for a variety of machine learning problems. This has led to a renewed interest in understanding the role and utility of various computational components of typical LSTM variants. In this paper, we present the first large-scale analysis of eight LSTM variants on three representative tasks: speech recognition, handwriting recognition, and polyphonic music modeling. The hyperparameters of all LSTM variants for each task were optimized separately using random search, and their importance was assessed using the powerful functional ANalysis Of VAriance framework. In total, we summarize the results of 5400 experimental runs ($\approx$15 years of CPU time), which makes our study the largest of its kind on LSTM networks. Our results show that none of the variants can improve upon the standard LSTM architecture significantly, and demonstrate the forget gate and the output activation function to be its most critical components. We further observe that the studied hyperparameters are virtually independent and derive guidelines for their efficient adjustment.

Index Terms—Functional ANalysis Of VAriance (fANOVA), long short-term memory (LSTM), random search, recurrent neural networks, sequence learning.

I. INTRODUCTION

RECURRENT neural networks with long short-term memory (LSTM) have emerged as an effective and scalable model for several learning problems related to sequential data. Earlier methods for attacking these problems have either been tailored toward a specific problem or did not scale to long time dependences. LSTMs on the other hand are both general and effective at capturing long-term temporal dependences. They do not suffer from the optimization hurdles that plague simple recurrent networks (SRNs) [1], [2] and have been used to advance the state of the art for many difficult problems. This includes handwriting recognition [3]–[5] and generation [6], language modeling [7] and translation [8], acoustic modeling of speech [9], speech synthesis [10], protein secondary structure prediction [11], analysis of audio [12], and video data [13] among others.

The central idea behind the LSTM architecture is a memory cell, which can maintain its state over time, and nonlinear gating units, which regulate the information flow into and out of the cell. Most modern studies incorporate many improvements that have been made to the LSTM architecture since its original formulation [14], [15]. However, LSTMs are now applied to many learning problems, which differ significantly in scale and nature from the problems that these improvements were initially tested on. A systematic study of the utility of various computational components, which comprise LSTMs (see Fig. 1), was missing. This paper fills that gap and systematically addresses the open question of improving the LSTM architecture.

We evaluate the most popular LSTM architecture (*vanilla LSTM*; Section II) and eight different variants thereof on three benchmark problems: acoustic modeling, handwriting recognition, and polyphonic music modeling. Each variant differs from the vanilla LSTM by a single change. This allows us to isolate the effect of each of these changes on the performance of the architecture. Random search [16]–[18] is used to find the best performing hyperparameters for each variant on each problem, enabling a reliable comparison of the performance of different variants. We also provide insights gained about hyperparameters and their interaction using functional ANalysis Of VAriance (fANOVA) [19].

II. VANILLA LSTM

The LSTM setup most commonly used in the literature was originally described by Graves and Schmidhuber [20]. We refer to it as *vanilla LSTM* and use it as a reference for comparison of all the variants. The vanilla LSTM incorporates changes by Gers *et al.* [21] and Gers and Schmidhuber [22] into the original LSTM [15] and uses full gradient training. Section III provides descriptions of these major LSTM changes.

A schematic of the vanilla LSTM block can be seen in Fig. 1. It features three gates (input, forget, and output), block input, a single cell (the constant error carousel), an output activation function, and peephole connections.[1] The output of the block is recurrently connected back to the block input and all of the gates.

A. Forward Pass

Let $\mathbf{x}^t$ be the input vector at time t, N be the number of LSTM blocks, and M the number of inputs. Then, we get the following weights for an LSTM layer.

1) *Input Weights:* $\mathbf{W}_z, \mathbf{W}_s, \mathbf{W}_f, \mathbf{W}_o \in \mathbb{R}^{N \times M}$.

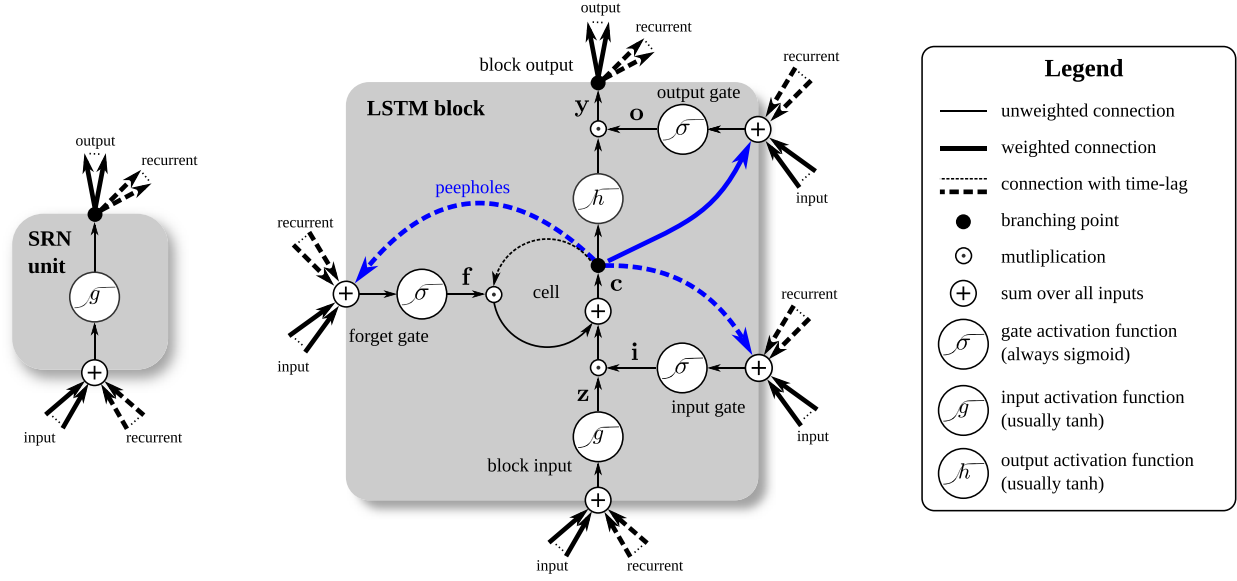[1]Some studies omit peephole connections, described in Section III-B.

Fig. 1.   Detailed schematic of the SRN unit (left) and an LSTM block (right) as used in the hidden layers of a recurrent neural network.

2) *Recurrent Weights:* $\mathbf{R}_z, \mathbf{R}_s, \mathbf{R}_f, \mathbf{R}_o \in \mathbb{R}^{N \times N}$.
3) *Peephole Weights:* $\mathbf{p}_s, \mathbf{p}_f, \mathbf{p}_o \in \mathbb{R}^N$.
4) *Bias Weights:* $\mathbf{b}_z, \mathbf{b}_s, \mathbf{b}_f, \mathbf{b}_o \in \mathbb{R}^N$.

Then the vector formulas for a vanilla LSTM layer forward pass can be written as

$$\bar{\mathbf{z}}^t = \mathbf{W}_z \mathbf{x}^t + \mathbf{R}_z \mathbf{y}^{t-1} + \mathbf{b}_z$$
$$\mathbf{z}^t = g(\bar{\mathbf{z}}^t) \qquad\qquad \text{block input}$$
$$\bar{\mathbf{i}}^t = \mathbf{W}_i \mathbf{x}^t + \mathbf{R}_i \mathbf{y}^{t-1} + \mathbf{p}_i \odot \mathbf{c}^{t-1} + \mathbf{b}_i$$
$$\mathbf{i}^t = \sigma(\bar{\mathbf{i}}^t) \qquad\qquad \text{input gate}$$
$$\bar{\mathbf{f}}^t = \mathbf{W}_f \mathbf{x}^t + \mathbf{R}_f \mathbf{y}^{t-1} + \mathbf{p}_f \odot \mathbf{c}^{t-1} + \mathbf{b}_f$$
$$\mathbf{f}^t = \sigma(\bar{\mathbf{f}}^t) \qquad\qquad \text{forget gate}$$
$$\mathbf{c}^t = \mathbf{z}^t \odot \mathbf{i}^t + \mathbf{c}^{t-1} \odot \mathbf{f}^t \qquad\qquad \text{cell}$$
$$\bar{\mathbf{o}}^t = \mathbf{W}_o \mathbf{x}^t + \mathbf{R}_o \mathbf{y}^{t-1} + \mathbf{p}_o \odot \mathbf{c}^t + \mathbf{b}_o$$
$$\mathbf{o}^t = \sigma(\bar{\mathbf{o}}^t) \qquad\qquad \text{output gate}$$
$$\mathbf{y}^t = h(\mathbf{c}^t) \odot \mathbf{o}^t \qquad\qquad \text{block output}$$

where σ, g, and h are pointwise nonlinear activation functions. The logistic sigmoid ($\sigma(x) = (1/1 + e^{-x})$) is used as the gate activation function and the hyperbolic tangent ($g(x) = h(x) = \tanh(x)$) is usually used as the block input and output activation function. Pointwise multiplication of two vectors is denoted by $\odot$.

B. Backpropagation Through Time

The deltas inside the LSTM block are then calculated as

$$\delta \mathbf{y}^t = \Delta^t + \mathbf{R}_z^T \delta \mathbf{z}^{t+1} + \mathbf{R}_i^T \delta \mathbf{i}^{t+1} + \mathbf{R}_f^T \delta \mathbf{f}^{t+1} + \mathbf{R}_o^T \delta \mathbf{o}^{t+1}$$
$$\delta \mathbf{o}^t = \delta \mathbf{y}^t \odot h(\mathbf{c}^t) \odot \sigma'(\bar{\mathbf{o}}^t)$$
$$\delta \mathbf{c}^t = \delta \mathbf{y}^t \odot \mathbf{o}^t \odot h'(\mathbf{c}^t) + \mathbf{p}_o \odot \delta \mathbf{o}^t + \mathbf{p}_i \odot \delta \mathbf{i}^{t+1}$$
$$\qquad\qquad + \mathbf{p}_f \odot \delta \mathbf{f}^{t+1} + \delta \mathbf{c}^{t+1} \odot \mathbf{f}^{t+1}$$
$$\delta \mathbf{f}^t = \delta \mathbf{c}^t \odot \mathbf{c}^{t-1} \odot \sigma'(\bar{\mathbf{f}}^t)$$
$$\delta \mathbf{i}^t = \delta \mathbf{c}^t \odot \mathbf{z}^t \odot \sigma'(\bar{\mathbf{i}}^t)$$
$$\delta \mathbf{z}^t = \delta \mathbf{c}^t \odot \mathbf{i}^t \odot g'(\bar{\mathbf{z}}^t).$$

Here, Δ^t is the vector of the deltas passed down from the layer above. If E is the loss function, it formally corresponds to $(\partial E / \partial \mathbf{y}^t)$, but not including the recurrent dependences. The deltas for the inputs are only needed if there is a layer below that needs training, and can be computed as follows:

$$\delta \mathbf{x}^t = \mathbf{W}_z^T \delta \mathbf{z}^t + \mathbf{W}_i^T \delta \mathbf{i}^t + \mathbf{W}_f^T \delta \mathbf{f}^t + \mathbf{W}_o^T \delta \mathbf{o}^t.$$

Finally, the gradients for the weights are calculated as follows, where $\star$ can be any of $\{\mathbf{z}, \mathbf{i}, \mathbf{f}, \mathbf{o}\}$, and $\langle \star_1, \star_2 \rangle$ denotes the outer product of two vectors:

$$\delta \mathbf{W}_\star = \sum_{t=0}^{T} \langle \delta \star^t, \mathbf{x}^t \rangle \quad \delta \mathbf{p}_i = \sum_{t=0}^{T-1} \mathbf{c}^t \odot \delta \mathbf{i}^{t+1}$$
$$\delta \mathbf{R}_\star = \sum_{t=0}^{T-1} \langle \delta \star^{t+1}, \mathbf{y}^t \rangle \quad \delta \mathbf{p}_f = \sum_{t=0}^{T-1} \mathbf{c}^t \odot \delta \mathbf{f}^{t+1}$$
$$\delta \mathbf{b}_\star = \sum_{t=0}^{T} \delta \star^t \quad \delta \mathbf{p}_o = \sum_{t=0}^{T} \mathbf{c}^t \odot \delta \mathbf{o}^t.$$

III. History of LSTM

The initial version of the LSTM block [14], [15] included (possibly multiple) cells and input and output gates, but no forget gate (NFG) and no peephole (NP) connections. The output gate, unit biases, or input activation function were omitted for certain experiments. Training was done using a mixture of real-time recurrent learning [23], [24] and backpropagation through time (BPTT) [24], [25]. Only the gradient of the cell was propagated back through time, and the gradient for the other recurrent connections was truncated. Thus, that study did not use the exact gradient for training. Another feature of that version was the use of full gate recurrence (FGR), which means that all the gates received recurrent inputs from all the gates at the previous time step in addition to the recurrent inputs from the block outputs. This feature did not appear in any of the later papers.

A. Forget Gate

The first paper to suggest a modification of the LSTM architecture introduced the forget gate [21], enabling the LSTM to reset its own state. This allowed learning of continual tasks such as embedded Reber grammar.

B. Peephole Connections

Gers and Schmidhuber [22] argued that in order to learn precise timings, the cell needs to control the gates. So far, this was only possible through an open output gate. Peephole connections [connections from the cell to the gates Fig. 1 (blue curves)] were added to the architecture in order to make precise timings easier to learn. In addition, the output activation function was omitted, as there was no evidence that it was essential for solving the problems that LSTM had been tested on so far.

C. Full Gradient

The final modification toward the vanilla LSTM was done by Graves and Schmidhuber [20]. This paper presented the full BPTT training for LSTM networks with the architecture described in Section II, and presented results on the TIMIT [26] benchmark. Using full BPTT had the added advantage that LSTM gradients could be checked using finite differences, making practical implementations more reliable.

D. Other Variants

Since its introduction, the vanilla LSTM has been the most commonly used architecture, but other variants have been suggested too. Before the introduction of full BPTT training, Gers *et al.* [27] used a training method based on the extended Kalman filtering, which enabled the LSTM to be trained on some pathological cases at the cost of high computational complexity. Schmidhuber *et al.* [28] proposed using a hybrid evolution-based method instead of BPTT for training but retained the vanilla LSTM architecture.

Bayer *et al.* [29] evolved different LSTM block architectures that maximize fitness on context-sensitive grammars. A larger study of this kind was later done by Jozefowicz *et al.* [30]. Sak *et al.* [9] introduced a linear projection layer that projects the output of the LSTM layer down before recurrent and forward connections in order to reduce the amount of parameters for LSTM networks with many blocks. By introducing a trainable scaling parameter for the slope of the gate activation functions, Doetsch *et al.* [5] were able to improve the performance of LSTM on an offline handwriting recognition data set. In what they call *Dynamic Cortex Memory*, Otte *et al.* [31] improved the convergence speed of the LSTM by adding recurrent connections between the gates of a single block (but not between the blocks).

Cho *et al.* [32] proposed a simplified variant of the LSTM architecture called gated recurrent unit (GRU). They used neither peephole connections nor output activation functions, and coupled the input and the forget gate into an *update gate*. Finally, their output gate (called *reset gate*) only gates the recurrent connections to the block input ($\mathbf{W}_z$).

Chung *et al.* [33] performed an initial comparison between the GRU and the Vanilla LSTM and reported mixed results.

IV. Evaluation Setup

The focus of this paper is to empirically compare different LSTM variants, and not to achieve the state-of-the-art results. Therefore, our experiments are designed to keep the setup simple and the comparisons fair. The vanilla LSTM is used as a baseline and evaluated together with eight of its variants. Each variant adds, removes, or modifies the baseline in exactly one aspect, which allows isolating their effect. They are evaluated on three different data sets from different domains to account for cross-domain variations.

For fair comparison, the setup needs to be similar for each variant. Different variants might require different settings of hyperparameters to give good performance, and we are interested in the best performance that can be achieved with each variant. For this reason, we chose to tune the hyperparameters like learning rate or amount of input noise individually for each variant. Since hyperparameter space is large and impossible to traverse completely, random search was used in order to obtain good-performing hyperparameters [18] for every combination of variant and data set. Random search was also chosen for the added benefit of providing enough data for analyzing the general effect of various hyperparameters on the performance of each LSTM variant (see Section V-B).
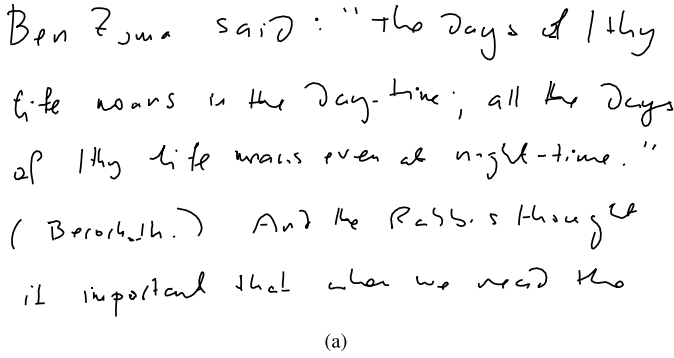
A. Data Sets

Each data set is split into three parts: a training set, a validation set used for early stopping and for optimizing the hyperparameters, and a test set for the final evaluation.

1) TIMIT: The TIMIT Speech corpus [26] is large enough to be a reasonable acoustic modeling benchmark for speech recognition, yet it is small enough to keep a large study such as ours manageable. Our experiments focus on the framewise classification task for this data set, where the objective is to classify each audio frame as one of 61 phones.[2] From the raw audio, we extract 12 mel frequency cepstrum coefficients [35] + energy over 25 ms hamming windows with a stride of 10 ms and a preemphasis coefficient of 0.97. This preprocessing is standard in speech recognition and was chosen in order to stay comparable with earlier LSTM-based results [20], [36]. The 13 coefficients along with their first and second derivatives comprise the 39 inputs to the network and were normalized to have zero mean and unit variance.

The performance is measured as the classification error percentage. The training, testing, and validation sets are split in line with Halberstadt [37] into 3696, 400, and 192 sequences, having 304 frames on average.

We restrict our study to the *core test set*, which is an established subset of the full TIMIT corpus, and use the splits into training, testing, and validation sets as detailed by Halberstadt [37]. In short, that means we only use the core

---

[2]Note that in linguistics a *phone* represents a distinct speech sound independent of the language. In contrast, a *phoneme* refers to a sound that distinguishes two words *in a given language* [34]. These terms are often confused in the machine learning literature.

(a)

```
Ben Zoma said: "The days of 1thy
life means in the day-time; all the days
of 1thy life means even at night-time ."
(Berochoth .) And the Rabbis thought
it important that when we read the
```

(b)

Fig. 2. (a) Example board (a08-551z, training set) from the IAM-OnDB dataset and (b) its transcription into character label sequences.

test set and drop the SA samples[3] from the training set. The validation set is built from some of the discarded samples from the full test set.

2) IAM Online: The IAM Online Handwriting Database [38][4] consists of English sentences as time series of pen movements that have to be mapped to characters. The IAM-OnDB data set splits into one training set, two validation sets, and one test set, having 775, 192, 216, and 544 boards each. Each board, see Fig. 2, contains multiple handwritten lines, which in turn consist of several strokes. We use one line per sequence, and joined the two validation sets together, so the final training, validation, and testing sets contain 5 355, 2 956, and 3 859 sequences, respectively.

Each handwriting line is accompanied with a target character sequence [see Fig. 2(b)] assembled from the following 81 ASCII characters:
```
abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
0123456789 ⊔ !''#&\'()*+,-./[]:;?
```
The board labeled as a08-551z (in the training set) contains a sequence of 11% characters that does not have an image in the strokes, and the percent character does not occur in any other board. That board was removed from the experiments.

We subsampled each sequence to half its length, which speeds up the training and does not harm performance. Each frame of the sequence is a 4-D vector containing Δx, Δy (the change in pen position), t (time since the beginning of the stroke), and a fourth dimension that contains value of one at the time of the pen lifting (a transition to the next stroke) and zeroes at all other time steps. Possible starts and ends of characters within each stroke are not explicitly marked.

---

[3]The dialect sentences (the SA samples) were meant to expose the dialectal variants of the speakers and were read by all 630 speakers. We follow [37] and remove them, because they bias the distribution of phones.

[4]The IAM-OnDB was obtained from http://www.iam.unibe.ch/fki/databases/iam-on-line-handwriting-database.

No additional preprocessing (such as baseline straightening, cursive correction, and so on) was used.

The networks were trained using the connectionist temporal classification (CTC) error function by Graves *et al.* [39] with 82 outputs (81 characters plus the special empty label). We measure performance in terms of the character error rate (CER) after decoding using best path decoding [39].

3) JSB Chorales: JSB Chorales [40] is a collection of 382 four-part harmonized chorales by Bach, consisting of 202 chorales in major keys and 180 chorals in minor keys. We used the preprocessed piano-rolls provided by Boulanger-Lewandowski *et al.* [41].[5] These piano-rolls were generated by transposing each MIDI sequence in C major or C minor and sampling frames every quarter note. The networks where trained to do next-step prediction by minimizing the negative log-likelihood. The complete data set consists of 229, 76, and 77 sequences (training, validation, and test sets, respectively) with an average length of 61.

B. Network Architectures & Training

A network with a single LSTM hidden layer and a sigmoid output layer was used for the JSB Chorales task. Bidirectional LSTM [20] was used for TIMIT and IAM Online tasks, consisting of two hidden layers, one processing the input forward, and the other one backward in time, both connected to a single softmax output layer. As loss function, we employed Cross-Entropy Error for TIMIT and JSB Chorales, while for the IAM Online task, the CTC loss by Graves *et al.* [39] was used. The initial weights for all networks were drawn from a normal distribution with a standard deviation of 0.1. Training was done using stochastic gradient descent with Nesterov-style momentum [42] with updates after each sequence. The learning rate was rescaled by a factor of (1 − momentum). Gradients were computed using full BPTT for LSTMs [20]. Training stopped after 150 epochs or once there was no improvement on the validation set for more than 15 epochs.

C. LSTM Variants

The vanilla LSTM from Section II is referred as Vanilla (V). For activation functions, we follow the standard and use the logistic sigmoid for σ and the hyperbolic tangent for both g and h. The derived eight variants of the V architecture are the following. We only report differences to the forward pass formulas presented in Section II-A.

NIG: No input gate: $\mathbf{i}^t = \mathbf{1}$
NFG: No forget gate: $\mathbf{f}^t = \mathbf{1}$
NOG: No output gate: $\mathbf{o}^t = \mathbf{1}$
NIAF: No input activation function: $g(\mathbf{x}) = \mathbf{x}$
NOAF: No output activation function: $h(\mathbf{x}) = \mathbf{x}$
CIFG: Coupled input and forget gate: $\mathbf{f}^t = \mathbf{1} - \mathbf{i}^t$
NP: No peepholes:

$$\bar{\mathbf{i}}^t = \mathbf{W}_i\mathbf{x}^t + \mathbf{R}_i\mathbf{y}^{t-1} + \mathbf{b}_i$$
$$\bar{\mathbf{f}}^t = \mathbf{W}_f\mathbf{x}^t + \mathbf{R}_f\mathbf{y}^{t-1} + \mathbf{b}_f$$
$$\bar{\mathbf{o}}^t = \mathbf{W}_o\mathbf{x}^t + \mathbf{R}_o\mathbf{y}^{t-1} + \mathbf{b}_o.$$

---

[5]Available at http://www-etud.iro.umontreal.ca/~boulanni/icml2012. at the time of writing.

FGR: Full gate recurrence

$$\bar{\mathbf{i}}^t = \mathbf{W}_i\mathbf{x}^t + \mathbf{R}_i\mathbf{y}^{t-1} + \mathbf{p}_i \odot \mathbf{c}^{t-1} + \mathbf{b}_i$$
$$+ \mathbf{R}_{ii}\mathbf{i}^{t-1} + \mathbf{R}_{fi}\mathbf{f}^{t-1} + \mathbf{R}_{oi}\mathbf{o}^{t-1}$$
$$\bar{\mathbf{f}}^t = \mathbf{W}_f\mathbf{x}^t + \mathbf{R}_f\mathbf{y}^{t-1} + \mathbf{p}_f \odot \mathbf{c}^{t-1} + \mathbf{b}_f$$
$$+ \mathbf{R}_{if}\mathbf{i}^{t-1} + \mathbf{R}_{ff}\mathbf{f}^{t-1} + \mathbf{R}_{of}\mathbf{o}^{t-1}$$
$$\bar{\mathbf{o}}^t = \mathbf{W}_o\mathbf{x}^t + \mathbf{R}_o\mathbf{y}^{t-1} + \mathbf{p}_o \odot \mathbf{c}^{t-1} + \mathbf{b}_o$$
$$+ \mathbf{R}_{io}\mathbf{i}^{t-1} + \mathbf{R}_{fo}\mathbf{f}^{t-1} + \mathbf{R}_{oo}\mathbf{o}^{t-1}.$$

The first six variants are self-explanatory. The CIFG variant uses only one gate for gating both the input and the cell recurrent self-connection—a modification of LSTM referred to as GRUs [32]. This is equivalent to setting $\mathbf{f}_t = \mathbf{1} - \mathbf{i}_t$ instead of learning the forget gate weights independently. The FGR variant adds recurrent connections between all the gates as in the original formulation of the LSTM [15]. It adds nine additional recurrent weight matrices, thus significantly increasing the number of parameters.

D. Hyperparameter Search

While there are other methods to efficiently search for good hyperparameters [43], [44], random search has several advantages for our setting: it is easy to implement, trivial to parallelize, and covers the search space more uniformly, thereby improving the follow-up analysis of hyperparameter importance.

We performed 27 random searches (one for each combination of the nine variants and three data sets). Each random search encompasses 200 trials for a total of 5400 trials of randomly sampling the following hyperparameters:
1) number of LSTM blocks per hidden layer: log-uniform samples from $[20, 200]$;
2) learning rate: log-uniform samples from $[10^{-6}, 10^{-2}]$;
3) momentum: $1 -$ log-uniform samples from $[0.01, 1.0]$;
4) standard deviation of Gaussian input noise: uniform samples from $[0, 1]$.

In the case of the TIMIT data set, two additional (boolean) hyperparameters were considered (not tuned for the other two data sets). The first one was the choice between the traditional momentum and the Nesterov-style momentum [42]. Our analysis showed that this had no measurable effect on performance, so the latter was arbitrarily chosen for all further experiments. The second one was whether to clip the gradients to the range $[-1, 1]$. This turned out to hurt the overall performance[6]; therefore, the gradients were never clipped in the case of the other two data sets.

Note that, unlike an earlier small-scale study [33], the number of parameters was not kept fixed for all variants. Since different variants can use their parameters differently, fixing this number can bias comparisons.

V. RESULTS & DISCUSSION

Each of the 5400 experiments was run on one of 128 AMD Opteron CPUs at 2.5 GHz and took 24.3 h on average

to complete. This sums up to a total single-CPU computation time of just below 15 years.

For TIMIT, the test set performance of the best trial were 29.6% classification error (CIFG), which is close to the best reported result of 26.9% [20]. Our best result of -8.38 log-likelihood (NIAF) on the JSB Chorales data set on the other hand is well below the -5.56 from Boulanger-Lewandowski *et al.* [41]. The best LSTM result is 26.9%. For the IAM Online data set, our best result was a CER of **9.26%** (NP) on the test set. The best previously published result is 11.5% CER by Graves *et al.* [45] using a different and much more extensive preprocessing.[7] Note that though the goal of this paper is not to provide the state-of-the-art results, but to do a fair comparison of different LSTM variants. So these numbers are only meant as a rough orientation for the reader.

A. Comparison of the Variants

A summary of the random search results is shown in Fig. 3. Welch's *t*-test at a significance level of $p = 0.05$ was used[8] to determine whether the mean test set performance of each variant was significantly different from that of the baseline. The box for a variant is highlighted in blue if its mean performance differs significantly from the mean performance of the vanilla LSTM.

The results in the top half of Fig. 3 represent the distribution of all 200 test set performances over the whole search space. Any conclusions drawn from them are therefore specific to our choice of search ranges. We have tried to choose reasonable ranges for the hyperparameters that include the best settings for each variant and are still small enough to allow for an effective search. The means and variances tend to be rather similar for the different variants and data sets, but even here some significant differences can be found.

In order to draw some more interesting conclusions, we restrict our further analysis to the top 10% performing trials for each combination of data set and variant [see Fig. 3 (bottom half)]. This way our findings will be less dependent on the chosen search space and will be representative for the case of reasonable hyperparameter tuning efforts.[9]

The first important observation based on Fig. 3 is that removing the output activation function (NOAF) or the forget gate (NFG) significantly hurt performance on all three data sets. Apart from the CEC, the ability to forget old information and the squashing of the cell state appear to be critical to the LSTM architecture. Indeed, without the output activation function, the block output can in principle grow unbounded. Coupling the input and the forget gate avoids this problem and might render the use of an output nonlinearity less important, which could explain why GRU performs well without it.

---

[7]Note that these numbers differ from the best test set performances that can be found in Fig. 3. This is the case because here we only report the single best performing trial as determined on the validation set. In Fig. 3, on the other hand, we show the test set performance of the 20 best trials for each variant.

[8]We applied the *Bonferroni adjustment* to correct for performing eight different tests (one for each variant).

[9]How much effort is reasonable will still depend on the search space. If the ranges are chosen much larger, the search will take much longer to find good hyperparameters.
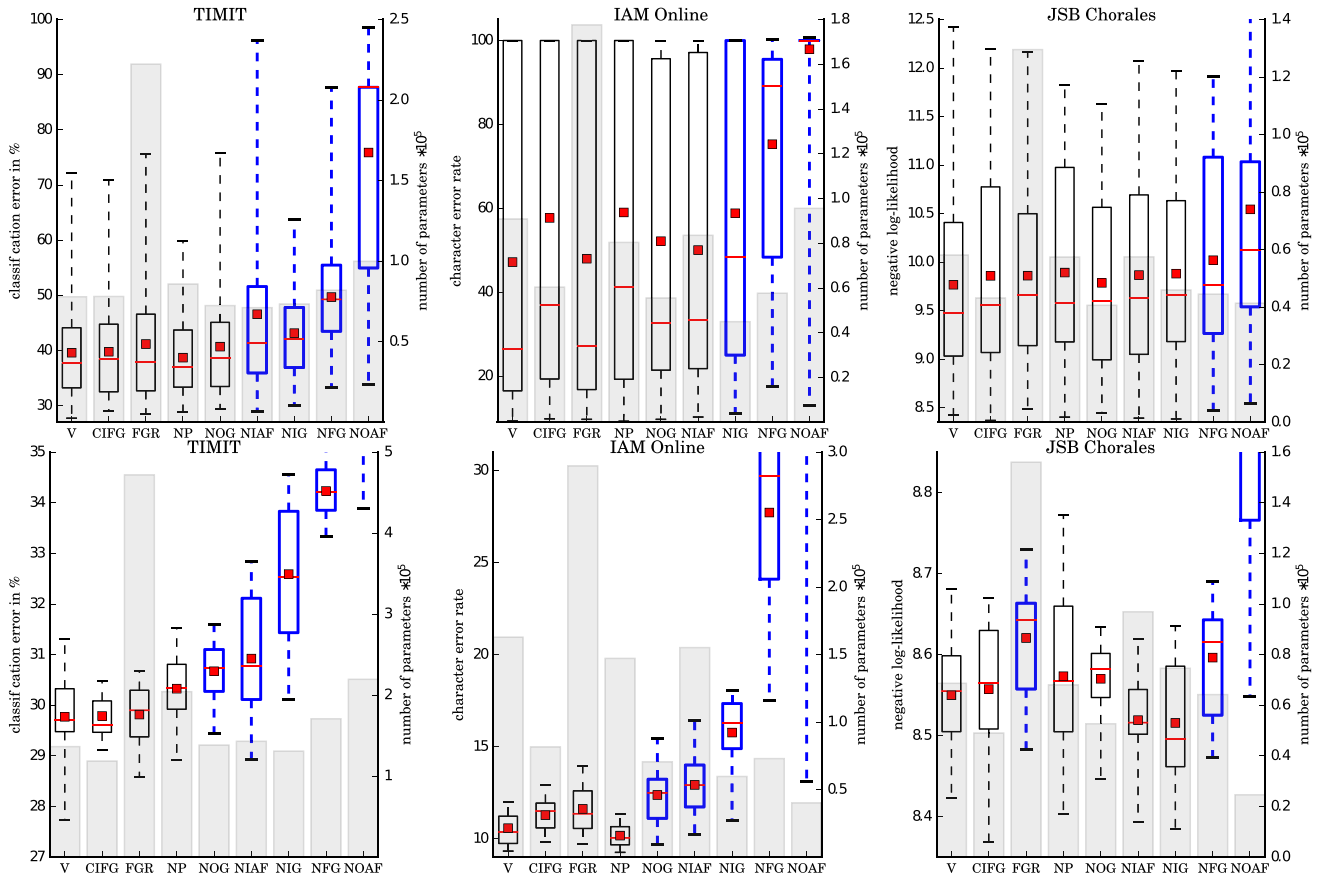
---

[6]Although this may very well be the result of the range having been chosen too tightly.

Fig. 3. *Test set* performance for all 200 trials (top) and for the best 10% (bottom) trials (according to the validation set) for each data set and variant. Boxes: range between the 25th and 75th percentiles of the data. Whiskers: whole range. Red dot: mean. Red line: median of the data. Thick blue lines: boxes of variants that differ significantly from the vanilla LSTM. Gray histogram in the background: average number of parameters for the top 10% performers of every variant.

Input and forget gate coupling (CIFG) did not significantly change the mean performance on any of the data sets, although the best performance improved slightly on music modeling. Similarly, removing peephole connections (NP) also did not lead to significant changes, but the best performance improved slightly for handwriting recognition. Both of these variants simplify LSTMs and reduce the computational complexity, so it might be worthwhile to incorporate these changes into the architecture.

Adding FGR does not significantly change performance on TIMIT or IAM Online, but led to worse results on the JSB Chorales data set. Given that this variant greatly increases the number of parameters, we generally advise against using it. Note that this feature was present in the original proposal of LSTM [14], [15], but has been absent in all following studies.

Removing the input gate (NIG), the output gate (NOG), and the input activation function (NIAF) led to a significant reduction in performance on speech and handwriting recognition. However, there was no significant effect on music modeling performance. A small (but statistically insignificant) average performance improvement was observed for the NIG and NIAF architectures on music modeling. We hypothesize that these behaviors will generalize to similar problems such as language modeling. For supervised learning on continuous real-valued data (such as speech and handwriting recognition),

the input gate, output gate, and input activation function are all crucial for obtaining good performance.

B. Impact of Hyperparameters

The fANOVA framework for assessing hyperparameter importance by Hutter *et al.* [19] is based on the observation that marginalizing over dimensions can be done efficiently in regression trees. This allows predicting the marginal error for one hyperparameter while averaging over all the others. Traditionally, this would require a full hyperparameter grid search, whereas here, the hyperparameter space can be sampled at random.

Average performance for any slice of the hyperparameter space is obtained by first training a regression tree and then summing over its predictions along the corresponding subset of dimensions. To be precise, a random regression *forest* of 100 trees is trained and their prediction performance is averaged. This improves the generalization and allows for an estimation of uncertainty of those predictions. The obtained marginals can then be used to decompose the variance into additive components using the fANOVA method [46], which provides an insight into the overall importance of hyperparameters and their interactions.

1) Learning Rate: Learning rate is the most important hyperparameter; therefore, it is very important to understand
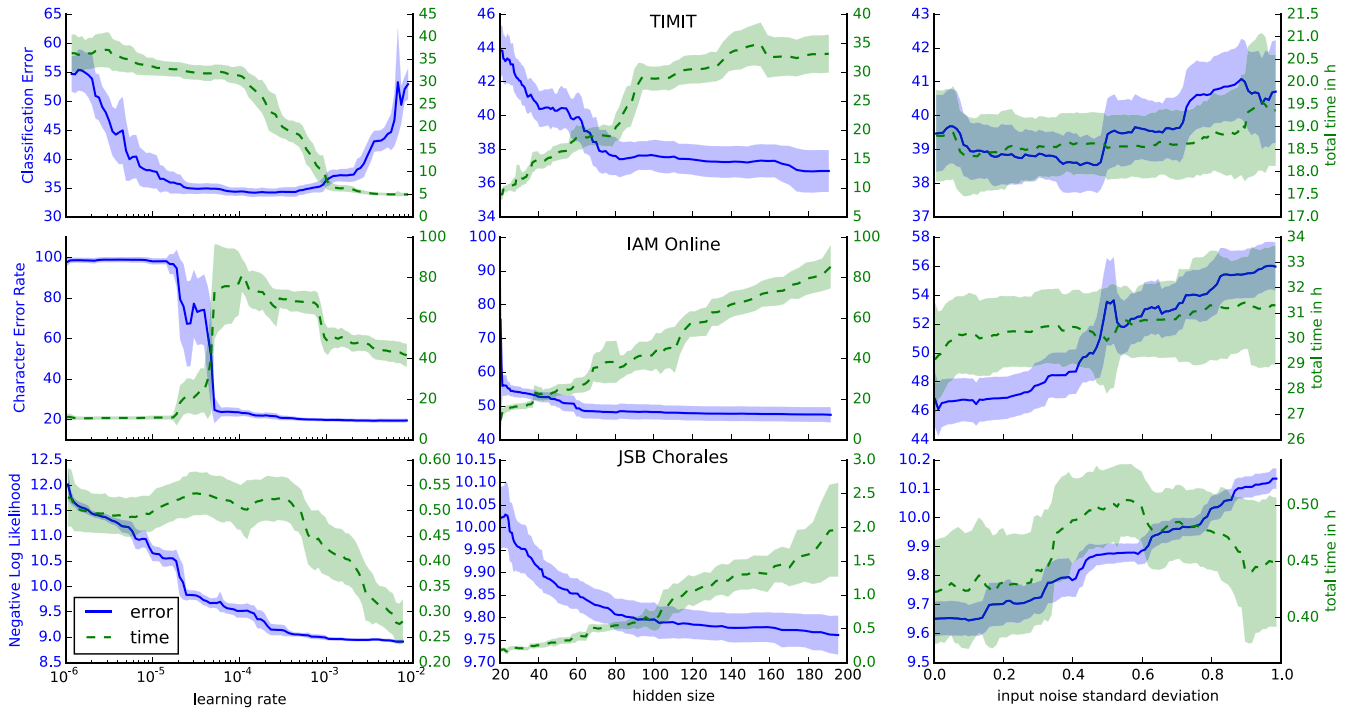
Fig. 4. Predicted marginal error (blue region) and marginal time for different values of the learning rate, hidden size, and the input noise (columns) for the test set of all three data sets (rows). Shaded area: standard deviation between the tree-predicted marginals and thus the reliability of the predicted mean performance. Note that each plot is for the vanilla LSTM but curves for all variants that are not significantly worse look very similar.

how to set it correctly in order to achieve good performance. Fig. 4 shows (blue region) how setting the learning rate value affects the predicted average performance on the test set. It is important to note that this is an average over all other hyperparameters and over all the trees in the regression forest. The shaded area around the curve indicates the standard deviation over tree predictions (not over other hyperparameters), thus quantifying the reliability of the average. The same is shown in green region with the predicted average training time.

The plots in Fig. 4 show that the optimal value for the learning rate is dependent on the data set. For each data set, there is a large basin (up to two orders of magnitude) of good learning rates inside of which the performance does not vary much. A related but unsurprising observation is that there is a sweet-spot for the learning rate at the high end of the basin.[10] In this region, the performance is good and the training time is small. Therefore, while searching for a good learning rate for the LSTM, it is sufficient to do a coarse search by starting with a high value (e.g., 1.0) and dividing it by ten until performance stops increasing.

Fig. 5 also shows that the fraction of variance caused by the learning rate is much bigger than the fraction due to interaction between learning rate and hidden layer size (some part of the higher order piece; for details, see Section V-B6). This suggests that the learning rate can be quickly tuned on a small network and then used to train a large one.

2) Hidden Layer Size: Not surprisingly the hidden layer size is an important hyperparameter affecting the LSTM

[10]Note that it is unfortunately outside the investigated range for IAM Online and JSB Chorales. This means that ideally we should have chosen the range of learning rates to include higher values as well.
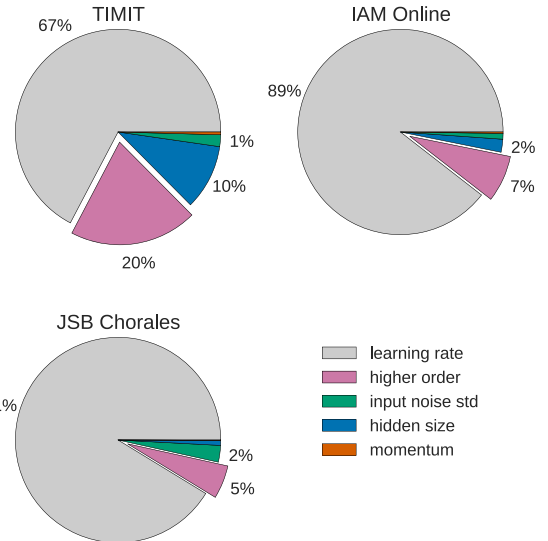

Fig. 5. Pie charts showing which fraction of variance of the test set performance can be attributed to each of the hyperparameters. The percentage of variance that is due to interactions between multiple parameters is indicated as higher order.

network performance. As expected, larger networks perform better, but with diminishing returns. It can also be seen in Fig. 4 (middle green region) that the required training time increases with the network size. Note that the scale here is wall-time, and thus factors in both the increased computation time for each epoch as well as the convergence speed.

3) Input Noise: Additive Gaussian noise on the inputs, a traditional regularizer for neural networks, has been used for LSTM as well. However, we find that not only does it almost
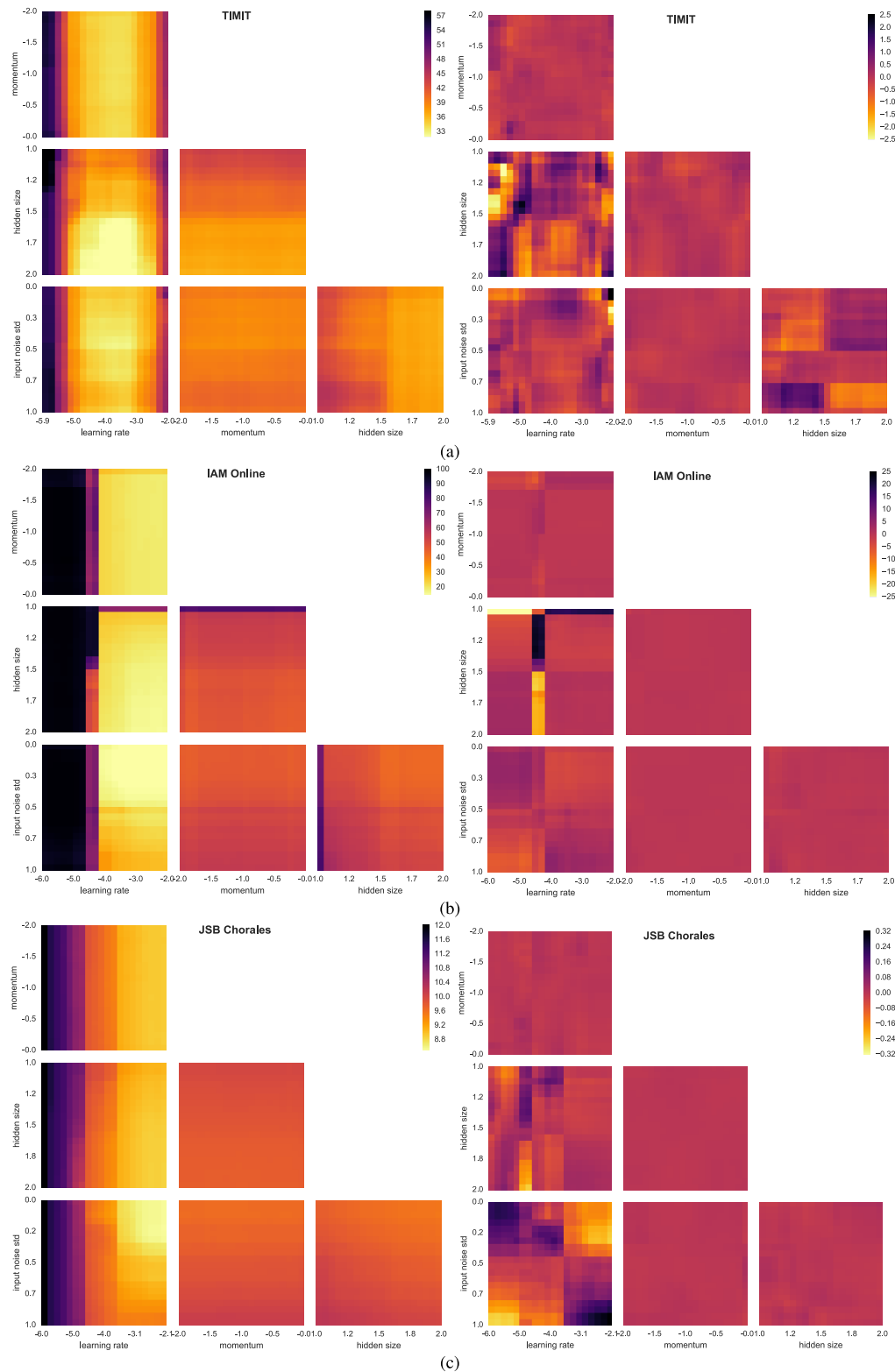
Fig. 6. Total marginal predicted performance for all pairs of hyperparameters (left) and the variation only due to their interaction (right). The plot is divided vertically into three subplots, one for every data set (TIMIT, IAM Online, and JSB Chorales). The subplots itself are divided horizontally into two parts, each containing a lower triangular matrix of heat maps. The rows and columns of these matrices represent the different hyperparameters (learning rate, momentum, hidden size, and input noise) and there is one heat map for every combination. The color encodes the performance as measured by the *Classification Error* for TIMIT, *CER* for IAM Online, and *Negative Log-Likelihood* for the JSB Chorales data set. For all datasets, low (blue) is better than high (red).

always hurt performance but also slightly increases training times. The only exception is TIMIT, where a small dip in error for the range of [0.2, 0.5] is observed.

4) Momentum: One unexpected result of this paper is that momentum affects neither performance nor training time in any significant way. This follows from the observation that

for none of the data sets, momentum accounted for more than 1% of the variance of test set performance. It should be noted that for TIMIT, the interaction between learning rate and momentum accounts for 2.5% of the total variance, but as with learning rate × hidden size (see Section V-B6), it does not reveal any interpretable structure. This may be the result of our choice to scale the learning rates dependent on momentum (Section IV-B). These observations suggest that momentum does not offer substantial benefits when training LSTMs with online stochastic gradient descent.

5) Analysis of Variance: Fig. 5 shows what fraction of the test set performance variance can be attributed to different hyperparameters. It is obvious that the learning rate is by far the most important hyperparameter, always accounting for more than two thirds of the variance. The next most important hyperparameter is the hidden layer size, followed by the input noise, leaving the momentum with less than 1% of the variance. Higher order interactions play an important role in the case of TIMIT, but are much less important for the other two data sets.

6) Interaction of Hyperparameters: Some hyperparameters interact with each other resulting in different performance from what could be expected by looking at them individually. As shown in Fig. 5, all these interactions together explain between 5% and 20% of the variance in test set performance. Understanding these interactions might allow us to speed up the search for good combinations of hyperparameters. To that end, we visualize the interaction between all pairs of hyperparameters in Fig. 6. Each heat map in the left part shows marginal performance for different values of the respective two hyperparameters. This is the average performance predicted by the decision forest when marginalizing over all other hyperparameters. Therefore, each one is the 2-D version of the performance plots from Fig. 4 in this paper.

The right side employs the idea of ANOVA to better illustrate the *interaction* between the hyperparameters. This means that variance of performance that can be explained by varying a single hyperparameter has been removed. In case two hyperparameters do not interact at all (are perfectly independent), that residual would thus be all zero (gray).

For example, looking at the pair *hidden size* and *learning rate* on the left side for the TIMIT data set, we can see that performance varies strongly along the *x*-axis (learning rate), first decreasing and then increasing again. This is what we would expect knowing the valley-shape of the learning rate from Fig. 4. Along the *y*-axis (hidden size), performance seems to decrease slightly from top to bottom. Again this is roughly what we would expect from the hidden size plot in Fig. 4.

On the right side of Fig. 6, we can see for the same pair of hyperparameters how their interaction differs from the case of them being completely independent. This heat map exhibits less structure, and it may in fact be the case that we would need more samples to properly analyze the interplay between them. However, given our observations so far this might not be worth the effort. In any case, it is clear from the plot on the left that varying the hidden size does not change the region of optimal learning rate.

One clear interaction pattern can be observed in the IAM Online and JSB data sets between learning rate and input noise. Here, it can be seen that for high learning rates ($\gtrsim 10^{-4}$), lower input noise ($\lesssim 0.5$) is better like also observed in the marginals from Fig. 4. However, this trend reverses for lower learning rates, where higher values of input noise are beneficial. Though interesting this is not of any practical relevance, because performance is generally bad in that region of low learning rates. Apart from this, however, it is difficult to discern any regularities in the analyzed hyperparameter interactions. We conclude that there is a little practical value in attending to the interplay between hyperparameters. Therefore, for practical purposes, hyperparameters can be treated as approximately independent and thus optimized separately.

VI. CONCLUSION

This paper reports the results of a large-scale study on the variants of the LSTM architecture. We conclude that the most commonly used LSTM architecture (vanilla LSTM) performs reasonably well on various data sets. None of the eight investigated modifications significantly improves performance. However, certain modifications, such as coupling the input and forget gates (CIFG), or removing peephole connections (NP) simplified LSTMs in our experiments without significantly decreasing performance. These two variants are also attractive, because they reduce the number of parameters and the computational cost of the LSTM.

The forget gate and the output activation function are the most critical components of the LSTM block. Removing any of them significantly impairs performance. We hypothesize that the output activation function is needed to prevent the unbounded cell state to propagate through the network and destabilize learning. This would explain why the LSTM variant GRU can perform reasonably well without it: its cell state is bounded because of the coupling of input and forget gate.

As expected, the learning rate is the most crucial hyperparameter, followed by the network size. Surprisingly though, the use of momentum was found to be unimportant in our setting of online gradient descent. Gaussian noise on the inputs was found to be moderately helpful for TIMIT, but harmful for the other data sets.

The analysis of hyperparameter interactions revealed no apparent structure. Furthermore, even the highest measured interaction (between learning rate and network size) is quite small. This implies that for practical purposes, the hyperparameters can be treated as approximately independent. In particular, the learning rate can be tuned first using a fairly small network, thus saving a lot of experimentation time.

Neural networks can be tricky to use for many practitioners compared with other methods whose properties are already well understood. This has remained a hurdle for newcomers to the field, since a lot of practical choices are based on the intuitions of experts, as well as experiences gained over time. With this paper, we have attempted to back some of these intuitions with experimental results. We have also presented new insights, both on architecture selection and hyperparameter tuning for LSTM networks, which have emerged as

the method of choice for solving complex sequence learning problems. In the future work, we plan to explore more complex modifications of the LSTM architecture.

REFERENCES

[1] S. Hochreiter, "Untersuchungen zu dynamischen neuronalen Netzen," Dept. Fakultät für Informatik, M.S. thesis, Tech. Univ. Munich, Munich, Germany, 1991.

[2] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber, "Gradient flow in recurrent nets: The difficulty of learning long-term dependencies," in *A Field Guide to Dynamical Recurrent Networks*, S. C. Kremer and J. F. Kolen, Eds. Piscataway, NJ, USA: IEEE Press, 2001.

[3] A. Graves, M. Liwicki, S. Fernández, R. Bertolami, H. Bunke, and J. Schmidhuber, "A novel connectionist system for unconstrained handwriting recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 31, no. 5, pp. 855–868, May 2009.

[4] V. Pham, T. Bluche, C. Kermorvant, and J. Louradour. (Nov. 2013). "Dropout improves recurrent neural networks for handwriting recognition." [Online]. Available: http://arxiv.org/abs/1312.4569

[5] P. Doetsch, M. Kozielski, and H. Ney, "Fast and robust training of recurrent neural networks for offline handwriting recognition," in *Proc. 14th Int. Conf. Frontiers Handwriting Recognit.*, 2014, pp. 279–284. [Online]. Available: http://people.sabanciuniv.edu/berrin/cs581/Papers/icfhr2014/data/4334a279.pdf

[6] A. Graves. (Aug. 2013). "Generating sequences with recurrent neural networks." [Online]. Available: http://arxiv.org/abs/1308.0850

[7] W. Zaremba, I. Sutskever, and O. Vinyals. (Sep. 2014). "Recurrent neural network regularization." [Online]. Available: http://arxiv.org/abs/1409.2329

[8] M.-T. Luong, I. Sutskever, Q. V. Le, O. Vinyals, and W. Zaremba. (2014). "Addressing the rare word problem in neural machine translation." [Online]. Available: http://arxiv.org/abs/1410.8206

[9] H. Sak, A. Senior, and F. Beaufays, "Long short-term memory recurrent neural network architectures for large scale acoustic modeling," in *Proc. Annu. Conf. Int. Speech Commun. Assoc. (INTERSPEECH)*, 2014, pp. 338–342. [Online]. Available: http://193.6.4.39/~czap/letoltes/IS14/IS2014/PDF/AUTHOR/IS141304.PDF

[10] Y. Fan, Y. Qian, F. Xie, and F. K. Soong, "TTS synthesis with bidirectional LSTM based recurrent neural networks," in *Proc. Interspeech*, 2014, pp. 1964–1968.

[11] S. K. Sønderby and O. Winther. (Dec. 2014). "Protein secondary structure prediction with long short term memory networks." [Online]. Available: http://arxiv.org/abs/1412.7828

[12] E. Marchi, G. Ferroni, F. Eyben, L. Gabrielli, S. Squartini, and B. Schuller, "Multi-resolution linear prediction based features for audio onset detection with bidirectional LSTM neural networks," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, May 2014, pp. 2164–2168.

[13] J. Donahue *et al.* (Nov. 2014). "Long-term recurrent convolutional networks for visual recognition and description." [Online]. Available: http://arxiv.org/abs/1411.4389

[14] S. Hochreiter and J. Schmidhuber, "Long short-term memory," Dept. Fakultät für Informatik, Tech. Univ. Munich, Munich, Germany, Tech. Rep. FKI-207-95, Aug. 1995. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.51.3117

[15] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997. [Online]. Available: http://www.bioinf.jku.at/publications/older/2604.pdf

[16] R. L. Anderson, "Recent advances in finding best operating conditions," *J. Amer. Statist. Assoc.*, vol. 48, no. 264, pp. 789–798, Dec. 1953. [Online]. Available: http://www.jstor.org/stable/2281072

[17] F. J. Solis and R. J.-B. Wets, "Minimization by random search techniques," *Math. Oper. Res.*, vol. 6, no. 1, pp. 19–30, Feb. 1981. [Online]. Available: http://pubsonline.informs.org/doi/abs/10.1287/moor.6.1.19

[18] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, no. 1, pp. 281–305, 2012. [Online]. Available: http://dl.acm.org/citation.cfm?id=2188395

[19] F. Hutter, H. Hoos, and K. Leyton-Brown, "An efficient approach for assessing hyperparameter importance," in *Proc. 31st Int. Conf. Mach. Learn.*, 2014, pp. 754–762. [Online]. Available: http://jmlr.org/proceedings/papers/v32/hutter14.html

[20] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional LSTM and other neural network architectures," *Neural Netw.*, vol. 18, nos. 5–6, pp. 602–610, Jul./Aug. 2005. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893608005001206

[21] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with LSTM," in *Proc. 9th Int. Conf. Artif. Neural Netw. (ICANN)*, vol. 2. Sep. 1999, pp. 850–855.

[22] F. A. Gers and J. Schmidhuber, "Recurrent nets that time and count," in *Proc. IEEE-INNS-ENNS Int. Joint Conf. Neural Netw. (IJCNN)*, vol. 3. Jul. 2000, pp. 189–194.

[23] A. J. Robinson and F. Fallside, *The Utility Driven Dynamic Error Propagation Network*. Cambridge, U.K.: Univ. Cambridge Department of Engineering, 1987.

[24] R. J. Williams, "Complexity of exact gradient computation algorithms for recurrent neural networks," College Comput. Sci., Northeastern Univ., Boston, MA, USA, Tech. Rep. NU-CCS-89-27, 1989.

[25] P. J. Werbos, "Generalization of backpropagation with application to a recurrent gas market model," *Neural Netw.*, vol. 1, no. 4, pp. 339–356, 1988.

[26] J. S. Garofolo, L. F. Lamel, W. M. Fisher, J. G. Fiscus, D. S. Pallett, and N. L. Dahlgren, "DARPA TIMIT acoustic-phonetic continous speech corpus CD-ROM. NIST speech disc 1-1.1," Nat. Inst. Standards Technol., Gaithersburg, MD, USA, Tech. Rep. LDC93S1, 1993.

[27] F. A. Gers, J. A. Pérez-Ortiz, D. Eck, and J. Schmidhuber, "DEKF-LSTM," in *Proc. 10th Eur. Symp. Artif. Neural Netw. (ESANN)*, 2002, pp. 369–376.

[28] J. Schmidhuber, D. Wierstra, M. Gagliolo, and F. Gomez, "Training recurrent networks by Evolino," *Neural Comput.*, vol. 19, no. 3, pp. 757–779, Mar. 2007.

[29] J. Bayer, D. Wierstra, J. Togelius, and J. Schmidhuber, "Evolving memory cell structures for sequence learning," in *Artificial Neural Networks—ICANN*. Berlin, Germany: Springer, 2009, pp. 755–764. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-642-04277-5_76

[30] R. Jozefowicz, W. Zaremba, and I. Sutskever, "An empirical exploration of recurrent network architectures," in *Proc. 32nd Int. Conf. Mach. Learn. (ICML)*, 2015, pp. 2342–2350.

[31] S. Otte, M. Liwicki, and A. Zell, "Dynamic cortex memory: Enhancing recurrent neural networks for gradient-based sequence learning," in *Artificial Neural Networks and Machine Learning* (Lecture Notes in Computer Science), vol. 8681. Cham, Germany: Springer, Sep. 2014, pp. 1–8. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-319-11179-7_1

[32] K. Cho *et al.* (2014). "Learning phrase representations using RNN encoder-decoder for statistical machine translation." [Online]. Available: http://arxiv.org/abs/1406.1078

[33] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. (Dec. 2014). "Empirical evaluation of gated recurrent neural networks on sequence modeling." [Online]. Available: http://arxiv.org/abs/1412.3555

[34] D. Crystal, *Dictionary of Linguistics and Phonetics*, vol. 30. New York, NY, USA: Wiley, 2011.

[35] P. Mermelstein, "Distance measures for speech recognition: Psychological and instrumental," in *Pattern Recognition and Artificial Intelligence*, C. H. Chen, Ed. New York, NY, USA: Academic, 1976, pp. 374–388.

[36] A. Graves, "Supervised sequence labelling with recurrent neural networks," Ph.D. dissertation, Dept. Fakultät für Informatik, Tech. Univ. Munich, Munich, Germany, 2008.

[37] A. K. Halberstadt, "Heterogeneous acoustic measurements and multiple classifiers for speech recognition," Ph.D. dissertation, Dept. Elect. Eng. Comput. Sci., Massachusetts Inst. Technol., Cambridge, MA, USA, 1998.

[38] M. Liwicki and H. Bunke, "IAM-OnDB—An on-line English sentence database acquired from handwritten text on a whiteboard," in *Proc. 8th Int. Conf. Document Anal. Recognit.*, 2005, pp. 956–961.

[39] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks," in *Proc. 23rd Int. Conf. Mach. Learn.*, 2006, pp. 369–376. [Online]. Available: http://dl.acm.org/citation.cfm?id=1143891

[40] M. Allan and C. K. I. Williams, "Harmonising chorales by probabilistic inference," in *Advances in Neural Information Processing Systems*, vol. 17. Cambridge, MA, USA: MIT Press, 2005, pp. 25–32.

[41] N. Boulanger-Lewandowski, Y. Bengio, and P. Vincent, "Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription," in *Proc. 29th Int. Conf. Mach. Learn.*, 2012, pp. 1159–1166. [Online]. Available: http://icml.cc/discuss/2012/590.html

[42] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," *J. Mach. Learn. Res.*, vol. 23, no. 3, pp. 1139–1147, 2013. [Online]. Available: http://jmlr.org/proceedings/papers/v28/sutskever13.html

[43] J. Snoek, H. Larochelle, and R. P. Adams, "Practical Bayesian optimization of machine learning algorithms," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Red Hook, NY, USA: Curran Associates, Inc., 2012, pp. 2951–2959.

[44] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *Proc. LION 5*, 2011, pp. 507–523.

[45] A. Graves, M. Liwicki, H. Bunke, J. Schmidhuber, and S. Fernández, "Unconstrained on-line handwriting recognition with recurrent neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2008, pp. 577–584.

[46] G. Hooker, "Generalized functional ANOVA diagnostics for high-dimensional functions of dependent variables," *J. Comput. Graph. Statist.*, vol. 16, no. 3, pp. 709–732, Sep. 2007. [Online]. Available: http://www.tandfonline.com/doi/abs/10.1198/106186007X237892

Jan Koutník received the Ph.D. degree in computer science from the Czech Technical University, Prague, Czech Republic, in 2008.

He is currently a Machine Learning Researcher with The Swiss AI Lab Istituto Dalle Molle di studi sull'Intelligenza Artificiale, Lugano, Switzerland. His current research interests include artificial neural networks, recurrent neural networks, evolutionary algorithms and deep-learning applied to reinforcement learning, control problems, image classification, handwriting, and speech recognition.

Klaus Greff received the Diploma degree in computer science from the University of Kaiserslautern, Kaiserslautern, Germany, in 2011. He is currently pursuing the Ph.D. degree with the Dalle Molle Institute for Artificial Intelligence Research, Lugano, Switzerland, under the supervision of Prof. J. Schmidhuber in the field of machine learning.

His current research interests include sequence learning and recurrent neural networks.

Bas R. Steunebrink is currently a Post-Doctoral Researcher with the Swiss AI lab Istituto Dalle Molle di studi sull'Intelligenza Artificiale, Lugano, Switzerland. He received the Ph.D. degree from Utrecht University, Utrecht, Netherlands, in 2010. His current research interests include artificial general intelligence, cognitive robotics, machine learning, resource-constrained control, and affective computing.

Jürgen Schmidhuber has pioneered self-improving general problem solvers since 1987, and deep learning neural networks (NNs) since 1991. He is currently a Professor of Artificial Intelligence with USI, Lugano, Switzerland. The recurrent NNs (RNNs) developed by his research groups with the Swiss AI Lab Istituto Dalle Molle di studi sull'Intelligenza Artificiale & Università della Svizzera italiana and SUPSI, Manno, Switzerland, and TU Munich, Munich, Germany, were the first RNNs to win official international contests. They have helped to revolutionize connected handwriting recognition, speech recognition, machine translation, optical character recognition, and image caption generation, and are now in use with Google, Apple, Microsoft, IBM, Baidu, and many other companies. His team's Deep Learners were also the first to win object detection and image segmentation contests, and achieved the world's first superhuman visual classification results, winning nine international competitions in machine learning & pattern recognition (more than any other team). They also were the first to learn control policies directly from high-dimensional sensory input using reinforcement learning. His research group also established the field of mathematically rigorous universal AI and optimal universal problem solvers. His formal theory of creativity, curiosity, and fun explains art, science, music, and humor. He also generalized algorithmic information theory and the manyworlds theory of physics, and introduced the concept of low-complexity art, the information age's extreme form of minimal art. He has authored 333 peer-reviewed papers.

Dr. Schmidhuber has been a member of the European Academy of Sciences and Arts since 2009. He received seven best paper/best video awards, and was a recipient of the 2013 Helmholtz Award of the International Neural Networks Society and the 2016 IEEE Neural Networks Pioneer Award.

Rupesh K. Srivastava is currently pursuing the Ph.D. degree with the Istituto Dalle Molle di studi sull'Intelligenza Artificiale & Università della Svizzera italiana, Lugano, Switzerland, under the supervision of Prof. J. Schmidhuber.

He currently involved in understanding and improving neural network architectures. He has involved in understanding the beneficial properties of local competition in neural networks, and new architectures which allow gradientbased training of extremely deep networks. He was involved in worked on reliability based design optimization using evolutionary algorithms with the Kanpur Genetic Algorithms Laboratory, Kanpur, India, supervised by Prof. K. Deb for his master's thesis.