LSTM: A Search Space Odyssey

By: Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R. Steunebrink, Jürgen Schmidhuber

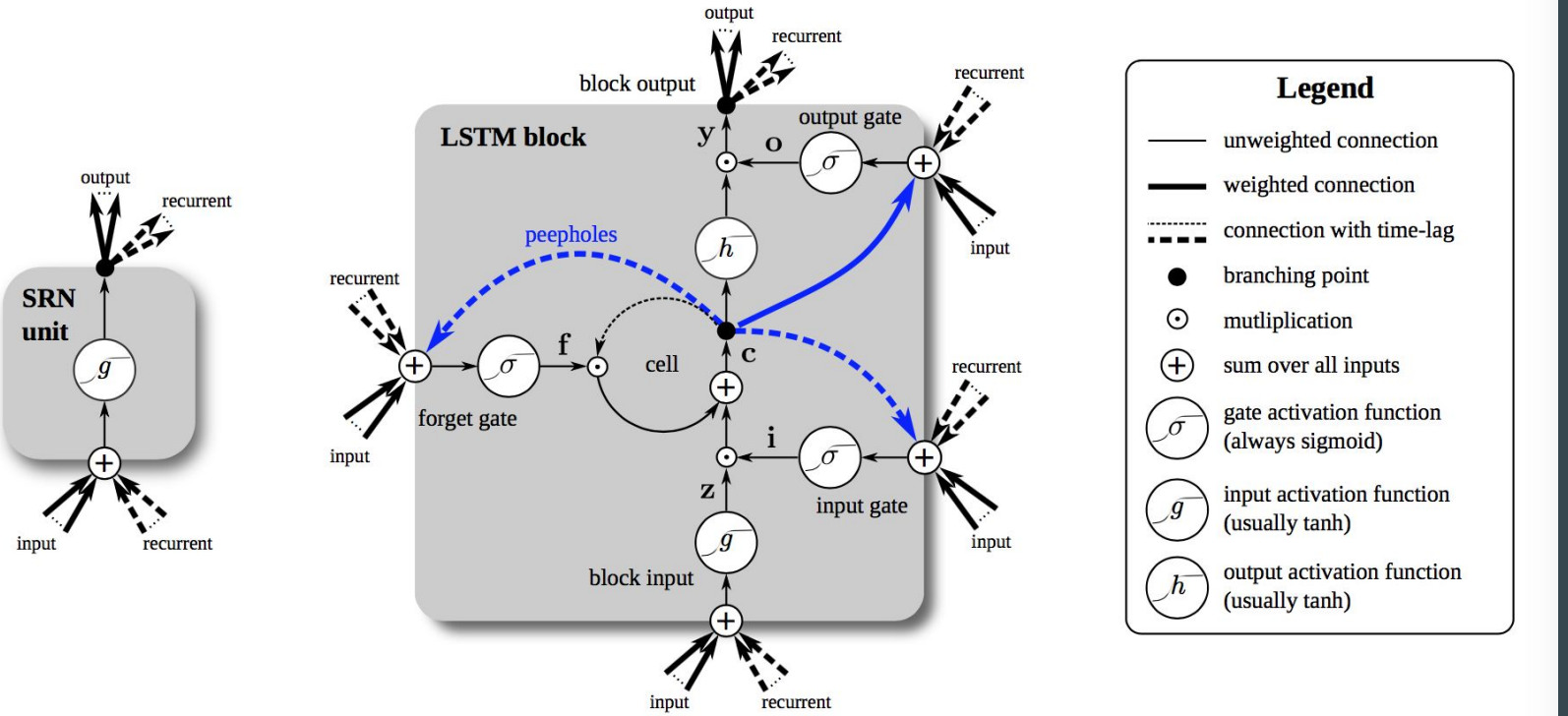
Presented By: Anshul Sacheti, YangLu Piao

Abstract

- In this paper, the authors analyze performance of eight LSTM variants on three representative tasks: speech recognition, handwriting recognition and polyphonic music modeling.

- Hyperparameters for each variant were optimized individually using random search and importance was gauged using fANOVA (an efficient approach for assessing hyperparameter importance)

Long Short-Term Memory (LSTM)



Hereby referred to as vanilla LSTM

Vanilla LSTM

- 3 Gates: Input, forget, output
- Block input
- Single Cell (Constant Error Carousel)
- Output activation function
- Peephole connections
- Output recurrently connected back to block input and all gates

Vanilla LSTM Math

$$\mathbf{z}^t = g(\mathbf{W}_z\mathbf{x}^t + \mathbf{R}_z\mathbf{y}^{t-1} + \mathbf{b}_z) \quad \textit{block input}$$

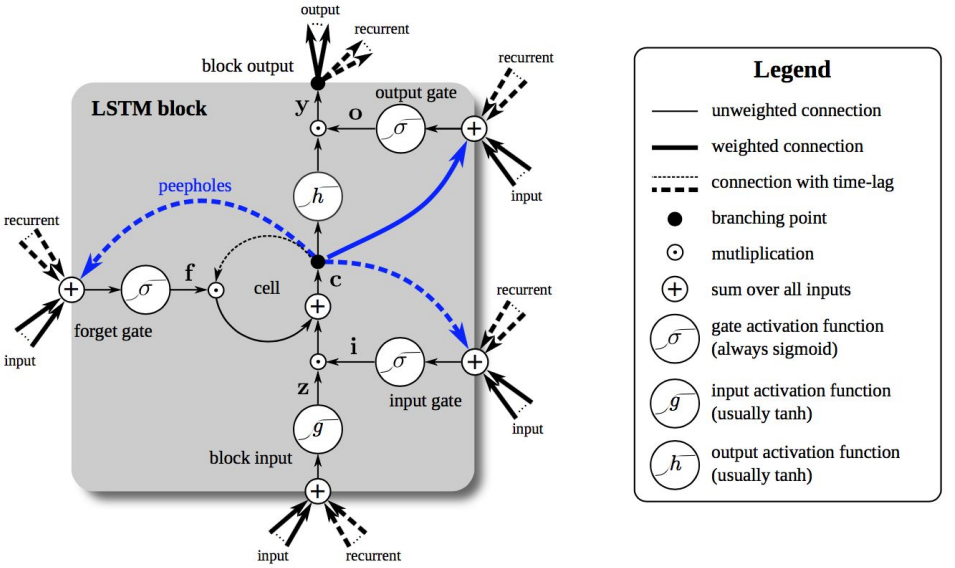
$$\mathbf{i}^t = \sigma(\mathbf{W}_i\mathbf{x}^t + \mathbf{R}_i\mathbf{y}^{t-1} + \mathbf{p}_i \odot \mathbf{c}^{t-1} + \mathbf{b}_i) \quad \textit{input gate}$$

$$\mathbf{f}^t = \sigma(\mathbf{W}_f\mathbf{x}^t + \mathbf{R}_f\mathbf{y}^{t-1} + \mathbf{p}_f \odot \mathbf{c}^{t-1} + \mathbf{b}_f) \quad \textit{forget gate}$$

$$\mathbf{c}^t = \mathbf{i}^t \odot \mathbf{z}^t + \mathbf{f}^t \odot \mathbf{c}^{t-1} \quad \textit{cell state}$$

$$\mathbf{o}^t = \sigma(\mathbf{W}_o\mathbf{x}^t + \mathbf{R}_o\mathbf{y}^{t-1} + \mathbf{p}_o \odot \mathbf{c}^t + \mathbf{b}_o) \quad \textit{output gate}$$

$$\mathbf{y}^t = \mathbf{o}^t \odot h(\mathbf{c}^t) \quad \textit{block output}$$



x^t is the input at time t, W = input weight matrices, R = square recurrent weight matrices, p = peephole weight vectors, b = bias vectors

LSTM History

- Initial Version
  - (possibly multiple) cells, input/output gates, no forget gates or peephole connections, training with Real Time Recurrent Learning (RTRL) / Backpropagation Through Time (BPTT),
  - Only gradient for cell was propagated back, other gradients truncated
  - Full gate recurrence - all gates received recurrent inputs from all gates at the previous time-step in addition to the recurrent inputs from the block outputs

LSTM Modifications

- Forget Gate (Gers et al., 1999) to reset a block's own state
- Peephole Connections (Gers & Schmidhuber, 2000) added to connect the cell to the gates
- Full Gradient (Graves & Schmidhuber, 2005) proposed via BPTT algorithm
- Gated Recurrent Unit (GRU) (Cho et al., 2014) simplification of LSTM with no peepholes, output activation functions, and coupled input and forget gate. Output gate only gates recurrent connections to block input

The Variants and Datasets

Eight LSTM variants (each a single change to vanilla LSTM):

- No Input Gate (NIG); No Forget Gate (NFG) ; No Output Gate (NOG); No Input Activation Function (NIAF); No Output Activation Function (NOAF); No Peepholes (NP); Coupled Input and Forget Gate (CIFG); Full Gate Recurrence (FGR)

Datasets:

- TIMIT (speech corpus), IAM Online (handwriting db) , JSB Chorales (polyphonic music modeling dataset)

TIMIT Speech Corpus (Garofolo et al., 1993)

- Frame-wise classification task such that each audio-frame is classified against 61 phones

- Performance = classification error percentage

- Training = 3696 sequences, Testing = 400 sequences, Validation = 192 sequences, each with 304 frames on average

IAM Online Handwriting Database (Liwicki & Bunke, 2005)

- English sentences as time series of pen movements that are mapped to characters. 4 input features: change in x/y pen positions, time since current stroke started, and binary indicating whether the pen is lifted

- Performance = Character Error Rate (CER)

- Training = 5355 sequences, Testing = 2956 sequences, Validation = 3859 sequences, each with 334 frames on average

JSB Chorales (Allan & Williams, 2005)

- Polyphonic music modeling dataset. Consists of sequences of binary vectors and the task is next-step prediction

- Performance = Negative Log Likelihood on validation/test set

- Training = 229 sequences, Testing = 76 sequences, Validation = 77 sequences, each with 61 frames on average

Evaluation Setup

- Goals: Compare different LSTM variants and not to achieve state-of-art results. Keep setup simple and comparisons fair. Performance+impact of hyperparameters
- Vanilla LSTM is used as a baseline.
- LSTM variants: NIG, NFG, NOG, NIAF, NOAF, NP, CIFG, FGR
- Datasets: TIMIT, IAM, JSB

Network Architectures

- Bidirectional LSTM which consists of two hidden layers, forward layer and backward layer, was used for TIMIT and IAM Online task; normal unidirectional LSTM with one hidden layer and a sigmoid output layer was used for JSB Chorales taks.
- As loss function, they employed Cross-Entropy Error for TIMIT and JSB Chorales, while for the IAM Online taks the Connectionist Temporal Classification(CTC) Error was used.
- The initial weights were drawn from a normal distribution with standard deviation of 0.1.

Training

- Training was done using SGD with Nesterov-style momentum.
- The learning rate was rescaled by a factor of [1-momentum].
- Gradients were computed using full BPTT(BackPropagation Through Time) for LSTMs.
- Training stopped after 150 epochs or once there was no improvement on the validation set for more than 15 epochs.

*Momentum vs. Nesterov momentum

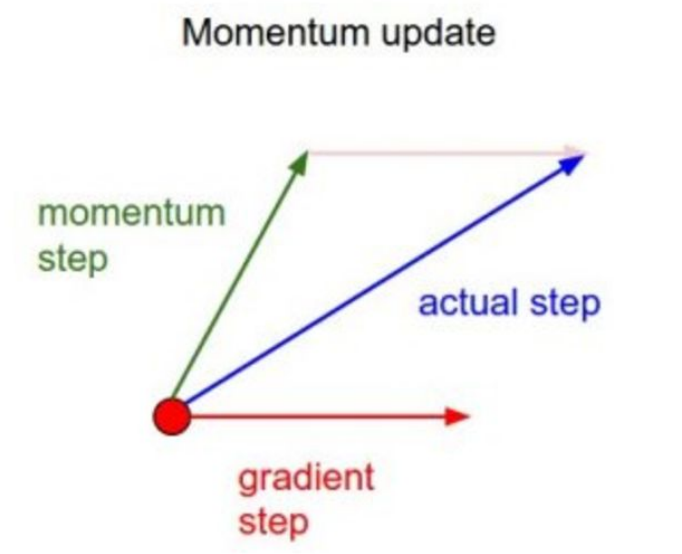
Momentum:

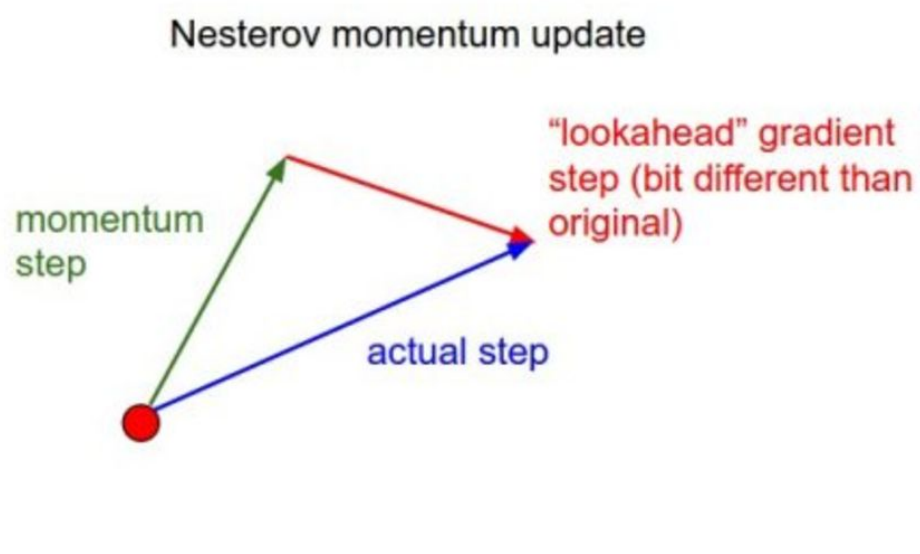
```
# Momentum update
v = mu * v - learning_rate * dx # integrate velocity
x += v # integrate position
```

Nesterov Momentum:

```
v_prev = v # back this up
v = mu * v - learning_rate * dx # velocity update stays the same
x += -mu * v_prev + (1 + mu) * v # position update changes form
```

*Momentum vs. Nesterov momentum





Nesterov momentum. Instead of evaluating gradient at the current position (red circle), we know that our momentum is about to carry us to the tip of the green arrow. With Nesterov momentum we therefore instead evaluate the gradient at this "looked-ahead" position.

Implementation of LSTM variants

Baseline(Vanilla):

```
z = g(tf.matmul(inputs, W_z) + tf.matmul(y_prev, R_z) + b_z)
i = tf.sigmoid(tf.matmul(inputs, W_i) + tf.matmul(y_prev, R_i) + tf.multiply(c_prev, p_i) + b_i)
f = tf.sigmoid(tf.matmul(inputs, W_f) + tf.matmul(y_prev, R_f) + tf.multiply(c_prev, p_f) + b_f)
c = tf.multiply(i, z) + tf.multiply(f, c_prev)
o = tf.sigmoid(tf.matmul(inputs, W_o) + tf.matmul(y_prev, R_o) + tf.multiply(c, p_o) + b_o)
y = tf.multiply(h(c), o)
```

NIG: i = 1.0 NFG: f = 1.0 NOG: o = 1.0 NIAF: no g for z

NOAF: h(c)->c NP: no tf.multiply(c_prev,*) and tf.multiply(c,*)

CIFG: f = 1-i FGR:

Implementation of LSTM variants

```
z = g(tf.matmul(inputs, W_z) + tf.matmul(y_prev, R_z) + b_z)

i_bar = tf.matmul(inputs, W_i) + tf.matmul(y_prev, R_i) + tf.multiply(c_prev, p_i) + b_i +
        tf.matmul(i_prev, R_ii) + tf.matmul(f_prev, R_fi) + tf.matmul(o_prev, R_oi)
i = tf.sigmoid(i_bar)

f_bar = tf.matmul(inputs, W_f) + tf.matmul(y_prev, R_f) + tf.multiply(c_prev, p_f) + b_f +
        tf.matmul(i_prev, R_if) + tf.matmul(f_prev, R_ff) + tf.matmul(o_prev, R_of)
f = tf.sigmoid(f_bar)

c = tf.multiply(i, z) + tf.multiply(f, c_prev)

o_bar = tf.matmul(inputs, W_o) + tf.matmul(y_prev, R_o) + tf.multiply(c, p_o) + b_o +
        tf.matmul(i_prev, R_io) + tf.matmul(f_prev, R_fo) + tf.matmul(o_prev, R_oo)
o = tf.sigmoid(o_bar)

y = tf.multiply(h(c), o)

return y, tf.concat([c, y, i, f, o],axis=1)
```

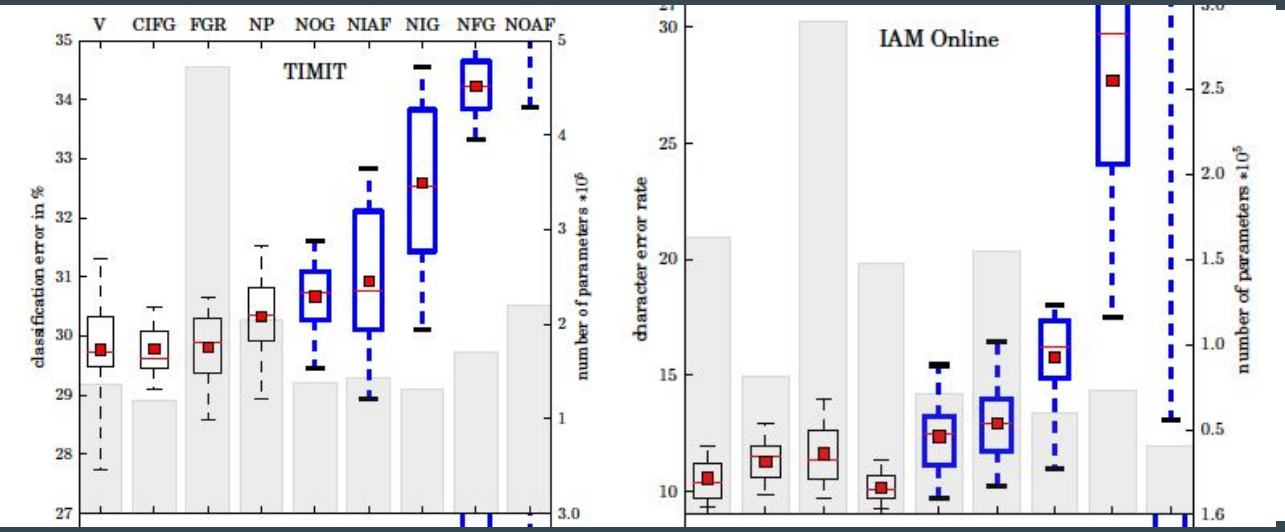
Hyperparameter Search

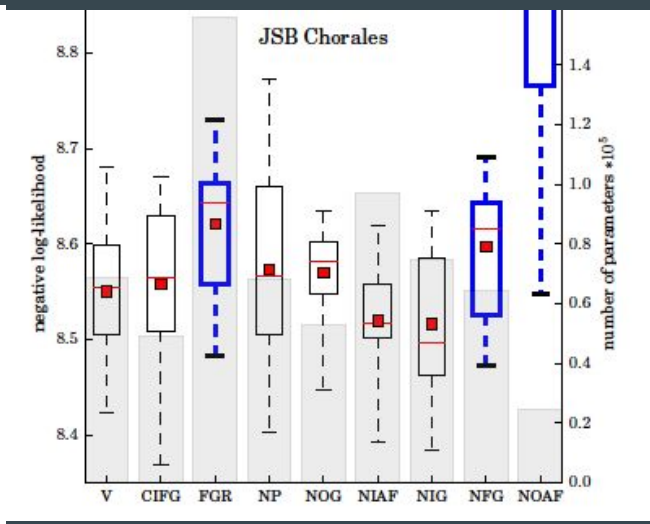
- Number of LSTM blocks per hidden layer: log-uniform samples from [20, 200]
- Learning rate: log-uniform samples from [10^-6, 10^-2]
- Momentum: 1 - log-uniform samples from [0.01, 1.0]
- Standard deviation of Gaussian input noise: uniform samples from [0, 1]
- In the case of the TIMIT dataset, two additional hyperparameter(booleans) were considered: choice between traditional momentum and Nesterov-style momentum; whether to clip the gradients to the range [-1, 1]. The second one turned out to hurt overall performance.

Observations

- 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.
- For TIMIT and JSB: 29.6% classification error(CIFG) and log-likelihood of -8.38(NIG) respectively.
- For IAM: Character Error Rate of 9.26%(NP).

Observations-performance





Observations-performance

1 . NOAF and NFG significantly hurt performance on all three datasets, which shows that the ability to forget old information and the squashing of the cell state appear to be critical for the LSTMs.

2. CIFG and NP did not lead to significant changes to the performance, but they simplify LSTMs and reduce the computational complexity, so it might be worthwhile to incorporate these changes into the architecture.

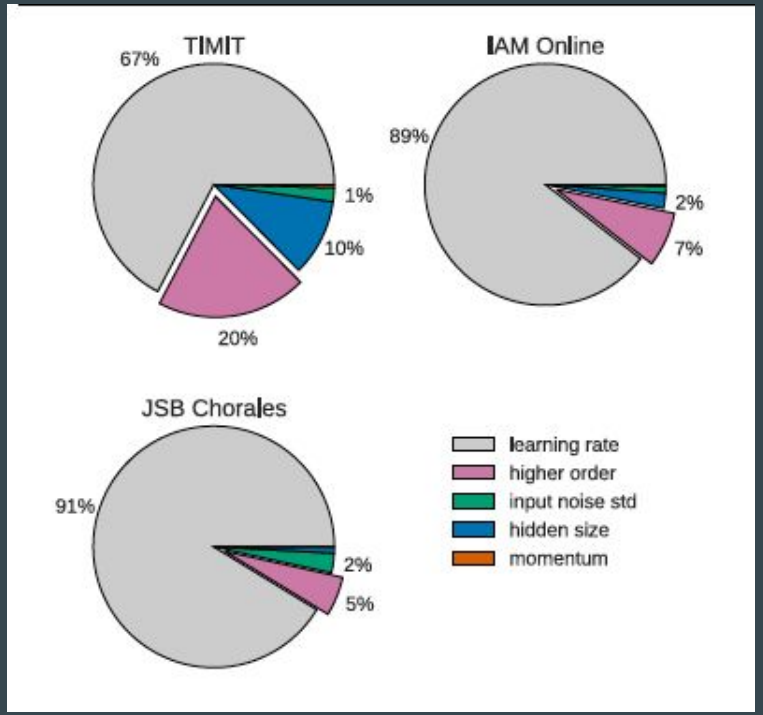
3. FGR greatly increased the number of parameter but did not significantly improve performance and even make it worse, so don’t use it.

Observations-performance

4. NIG, NOG and NIAF had no significant effect on music modeling performance. We can hypothesize that these behaviors will generalize to similar problems such as language modeling.

Observations-impact of hyperparameters

Higher order: The percentage of variance that is due to interactions between multiple parameters. (TIMIT)



Observations-impact of hyperparameters

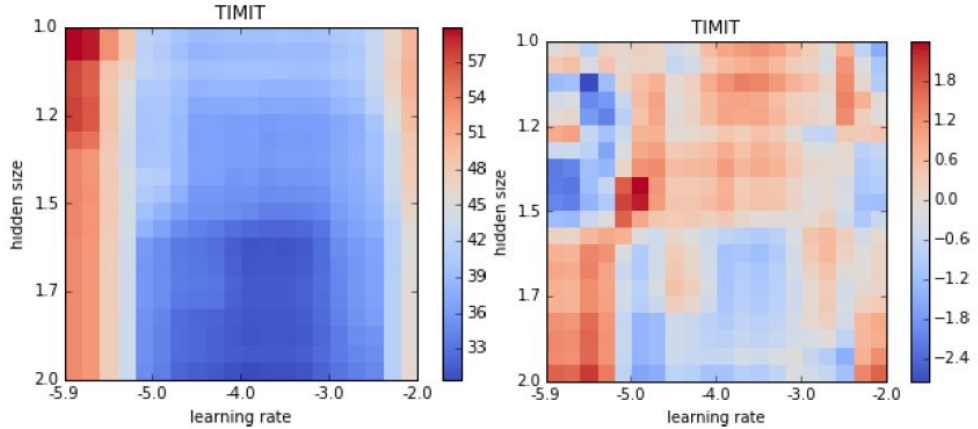


Figure 5. Left: The predicted marginal error for combinations of learning rate and hidden size. Right: The component that is solely due to the interaction of the two and cannot be attributet to changes in one of them alone. In other words the difference to the case of them being perfectly independent. (Blue is better than red.)

Observations-impact of hyperparameters

1. Learning rate is by far the most important hyperparameter. The next is the hidden layer size, followed by the input noise.

2. Higher order interactions play an important role in the case of TIMIT.

3. 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.

4. TIMIT:

$$
\begin{aligned}
\text{learning rate} \times \text{hidden size} &= 6.7\% \\
\text{learning rate} \times \text{input noise} &= 4.4\% \\
\text{hidden size} \times \text{input noise} &= 2.0\% \\
\text{learning rate} \times \text{momentum} &= 1.5\% \\
\text{momentum} \times \text{hidden size} &= 0.6\% \\
\text{momentum} \times \text{input noise} &= 0.4\%
\end{aligned}
$$

Observations-Impact of hyperparameters

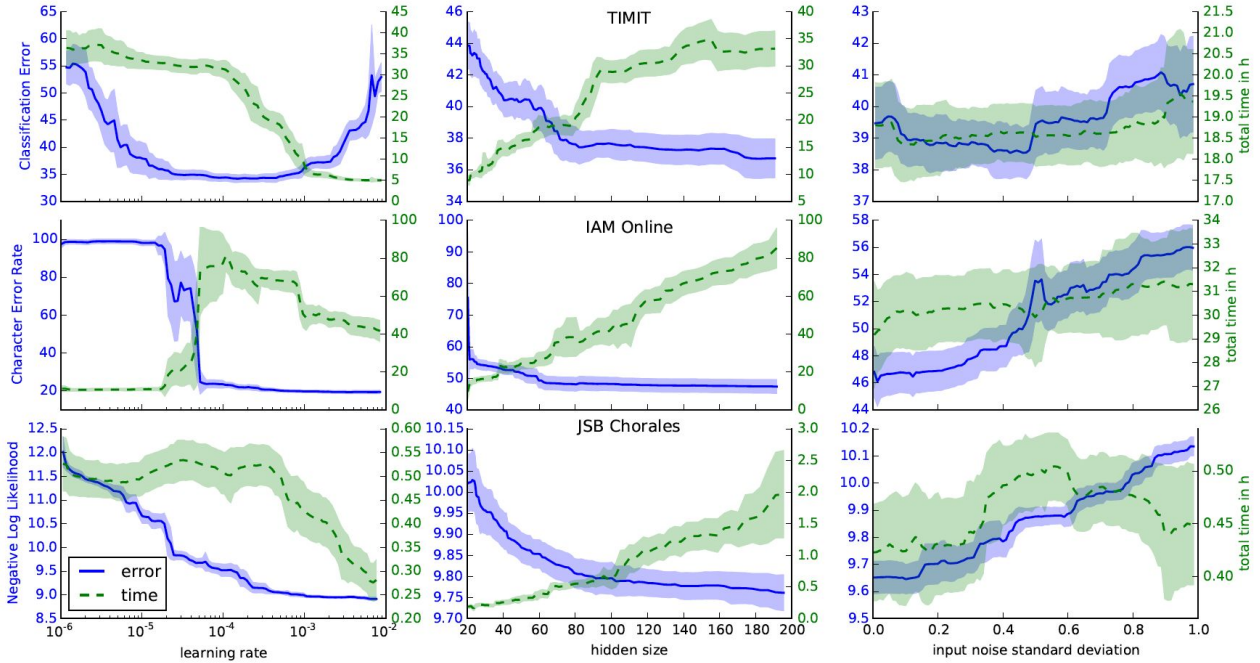


Figure 4. Predicted marginal error (blue) and marginal time for different values of the *learning rate*, *hidden size*, and the *input noise* (columns) for all three datasets (rows). The shaded area indicates the standard deviation between the tree-predicted marginals and thus indicating the reliability 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.

Observations-impact of hyperparameters

5. Larger network (bigger hidden layer size) perform better, and the required training time increases with the network size.

6. Adding Gaussian noise on the inputs almost always hurts performance and slightly increase training times. The only exception is TIMIT.

7. Momentum affects neither performances nor training time.

Conclusion

1. The most commonly used LSTM architecture(vanilla LSTM) performs reasonably well on various datasets and using any 8 possible modifications does not significantly improve the LSTM performance.

2. Certain modifications such as coupling the input and forget gates or removing peephole connections simplify LSTM without significantly hurting performance.

3. The forget gate and the output activation function are the critical components of the LSTM block.

Conclusion

4. Learning rate and network size are the most crucial tunable LSTM hyperparameters. The use of momentum was found to be unimportant. Gaussian noise on the inputs was found to be moderately helpful for TIMIT, but harmful for other datasets.

5. The analysis of hyperparameter interactions revealed that even the highest measured interaction (between learning rate and network size) is quite small. This implies that the hyperparameters can be tuned independently. In particular, the learning rate can be calibrated first using a fairly small network, thus saving a lot of experimentation time.

Thank you!