Long Short-Term Memory Networks

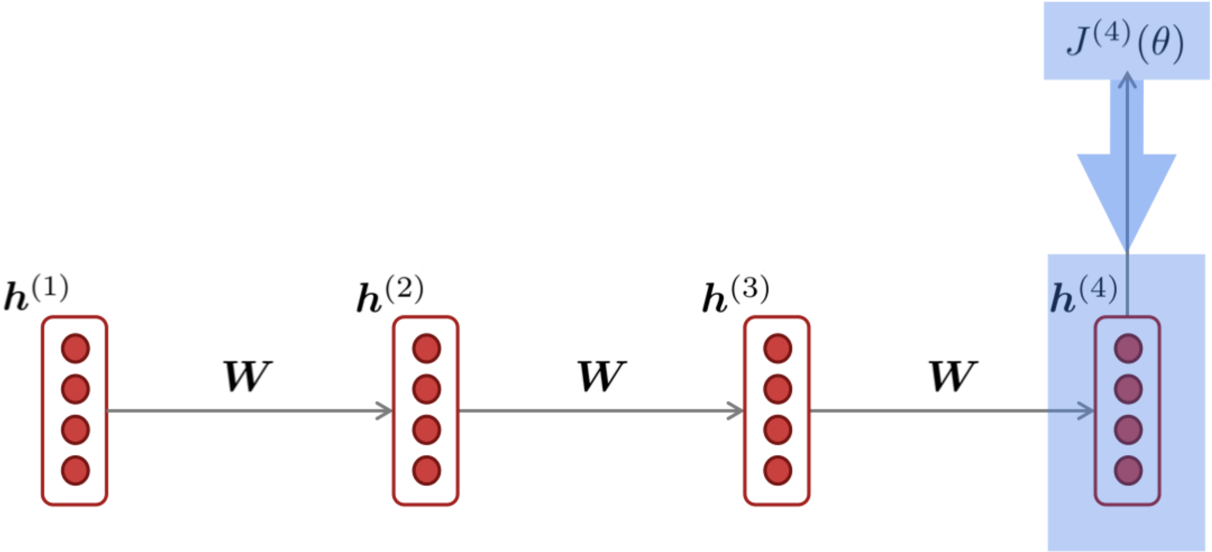
1. Why LSTMs?

What is wrong with me? :(asks RNN

Why LSTMs?

- Traditional RNNs had a lot of matrix multiplication involved.
- Naturally this caused the vanishing gradients problem during back-propagation.

Vanishing Gradients in RNNs



$$\frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(1)}} = \frac{\partial \boldsymbol{h}^{(2)}}{\partial \boldsymbol{h}^{(1)}} \times \frac{\partial \boldsymbol{h}^{(3)}}{\partial \boldsymbol{h}^{(2)}} \times \frac{\partial \boldsymbol{h}^{(4)}}{\partial \boldsymbol{h}^{(3)}} \times \frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(4)}}$$

Why is vanishing gradients a problem?

- Gradient signal from far away is lost because it’s much smaller than the gradient from close-by.
- As a result, model weights are updated only with respect to near effects, not long-term effects.

Why is vanishing gradients a problem?

- Suppose a language modelling task:

  The leaves of the Banyan tree ____ … (is or are?).

  Correct answer: The leaves of the Banyan tree <u>are</u> …

Why is vanishing gradients a problem?

- An RNN suffering from vanishing gradients may answer by giving weightage to sequential recency.
- **Syntactic Recency:** The **leaves** of the Banyan tree <u>are</u> …
- **Sequential Recency:** The leaves of the Banyan **tree** <u>is</u> …

How to fix the vanishing gradients problem?

- In an RNN, the hidden state $h^{(t)}$ is constantly being rewritten. Hence, it is very difficult for the model to preserve information over many time steps.
- LSTMs propose a separate memory for the RNNs.

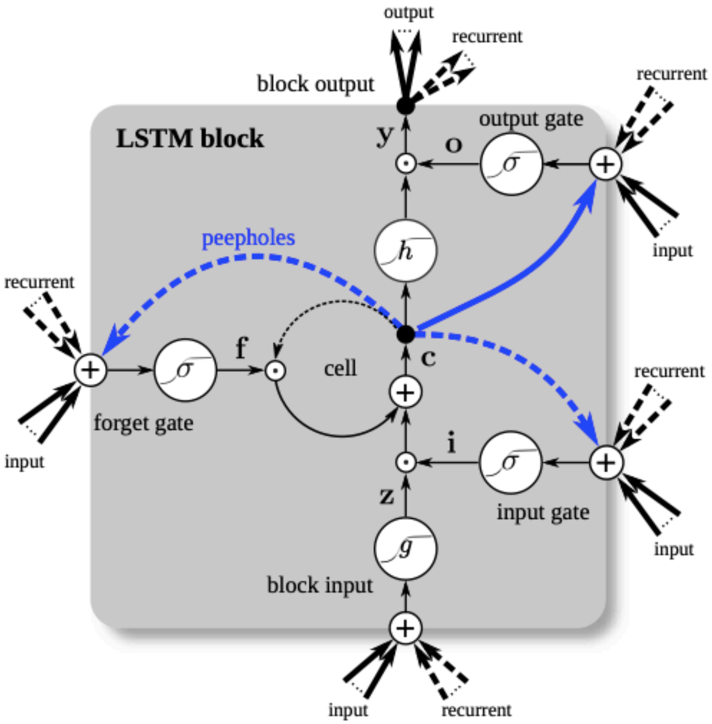
How does an LSTM solve vanishing gradients?

- LSTMs reduce the no. of matrix multiplications in favour of Hadamard product and addition of Matrices.
- LSTMs make it easier for the RNN to preserve information over many time steps.

2. What are Vanilla LSTMs?

Interesting stuff ahead…

Vanilla LSTMs



Vanilla LSTMs Forward Pass

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

Vanilla LSTMs Back-propagation through Time

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

3. History of LSTMs

Crash Course: Modern History…

Original LSTM

- The original LSTM did not have the forget gate and the peephole connections.
- The model was trained using a mixture of Real Time Recurrent Learning and Back-propagation through Time.
- It used *full gate recurrence.*

Forget Gate

- Forget Gate was introduced in 1999, 4 years after the introduction of LSTMs in 1995.
- This allowed learning of continual tasks such as embedded Reber Grammar.

Peephole Connections

- Peephole Connections were introduced in 2000, 5 years after the introduction of LSTMs in 1995.
- This allowed the cell to control the gates in order to learn precise timings.
- Output activation was omitted.

Full Gradient

- Full Back-propagation through Time was used to train the LSTMs.
- LSTM gradients could be checked using finite differences, making practical implementations more reliable.

4. Evaluation Setup

On your mark, get set!…

Setup

- We have to compare various variants of LSTM. So, we compare them across three different tasks on three different datasets.
- Hyper-parameters are separately tuned for each variant, so as to compare their best performances.

Dataset Setup

- Each Dataset is split into Training, Validation and Testing Dataset.
- **TIMIT Speech Corpus:** *Speech Recognition*
- **IAM Online:** *Handwriting Recognition*
- **JSB Chorales:** *Music Modelling*

Network Architecture

- Single LSTM hidden layer and sigmoid output layer for **JSB Chorales** task.
- Bidirectional LSTM for the other two tasks.
- Cross-Entropy Loss for **TIMIT** and **JSB Chorales** tasks.
- Connectionist Temporal Classification (CTC) Loss for **IAM Online**.

Training

- Initial weights for all the networks were drawn from a normal distribution of s.d. = 0.1
- **Optimiser:** SGD + Nesterov Momentum
- The Learning Rate was rescaled by a factor of (1-momentum)
- Gradients were computed using BPTT
- 15 epochs

LSTM Variants

- NIG: No Input Gate: $i^{(t)} = 1$
- NFG: No Forget Gate: $f^{(t)} = 1$
- NOG: No Output Gate: $o^{(t)} = 1$
- NIAF: No Input Activation Function: $g(x) = x$
- NOAF: No Output Activation Function: $h(x) = x$
- CIFG: Coupled Input and Forget Gate: $f^{(t)} = 1 - i^{(t)}$
- FGR: Full Gate Recurrence
- NP: No peepholes

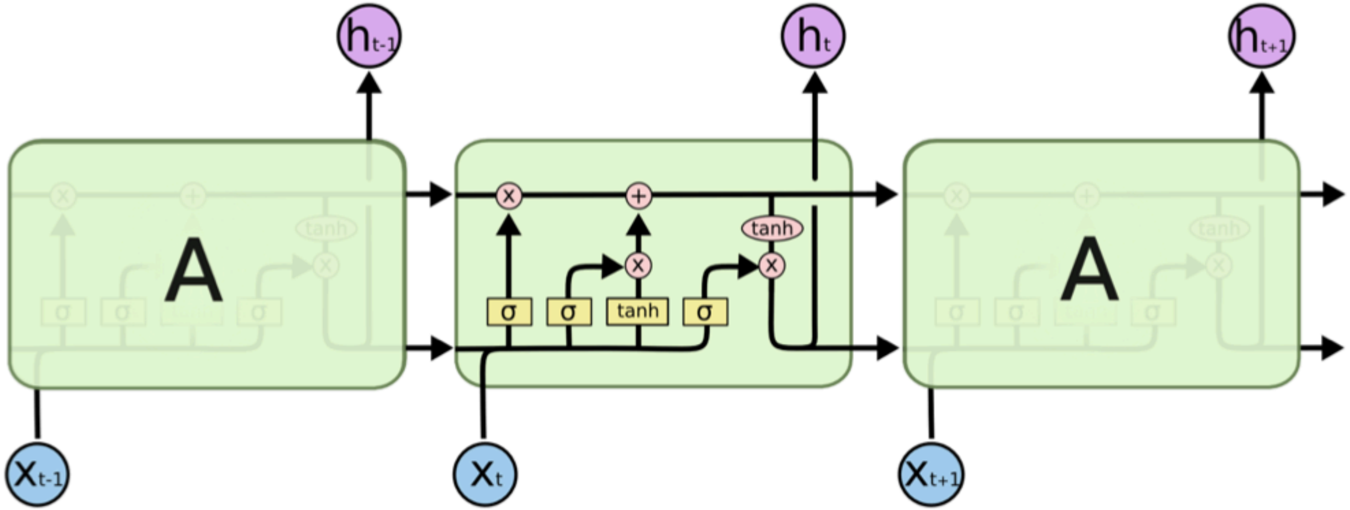
Full Gate Recurrence

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

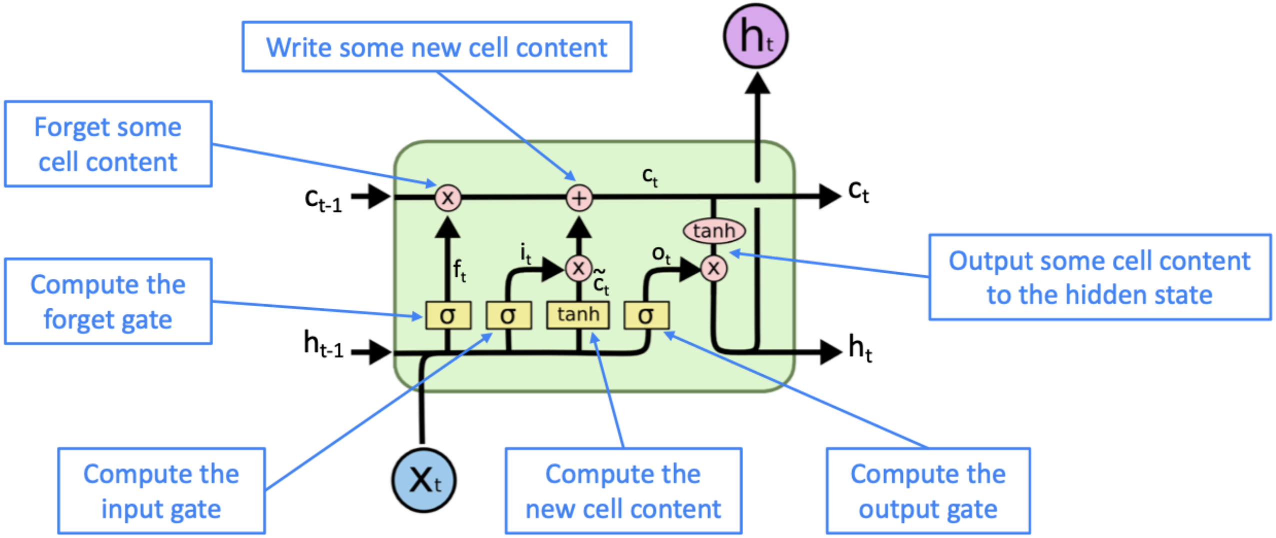
NP LSTMs

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

NP LSTMs



NP LSTMs

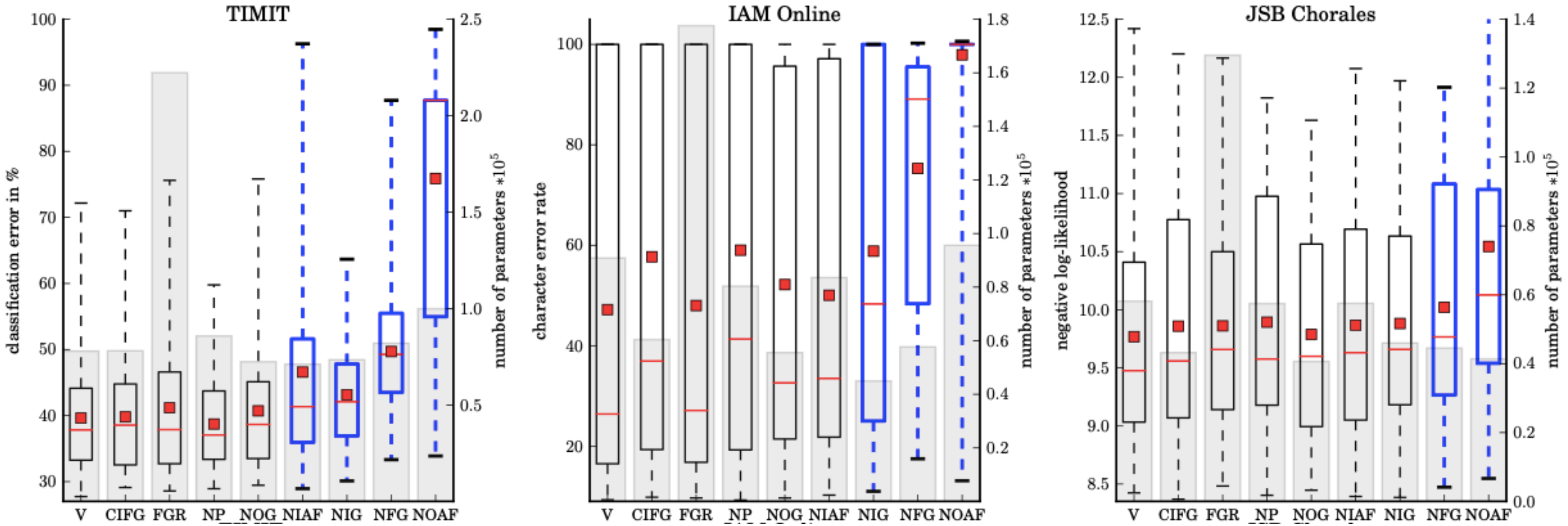


Write some new cell content
Forget some cell content
Compute the forget gate
Compute the input gate
Compute the new cell content
Compute the output gate
Output some cell content to the hidden state
c_{t-1}
h_{t-1}
c_t
h_t
f_t
i_t
$\tilde{c}_t$
o_t
σ
tanh
x_t

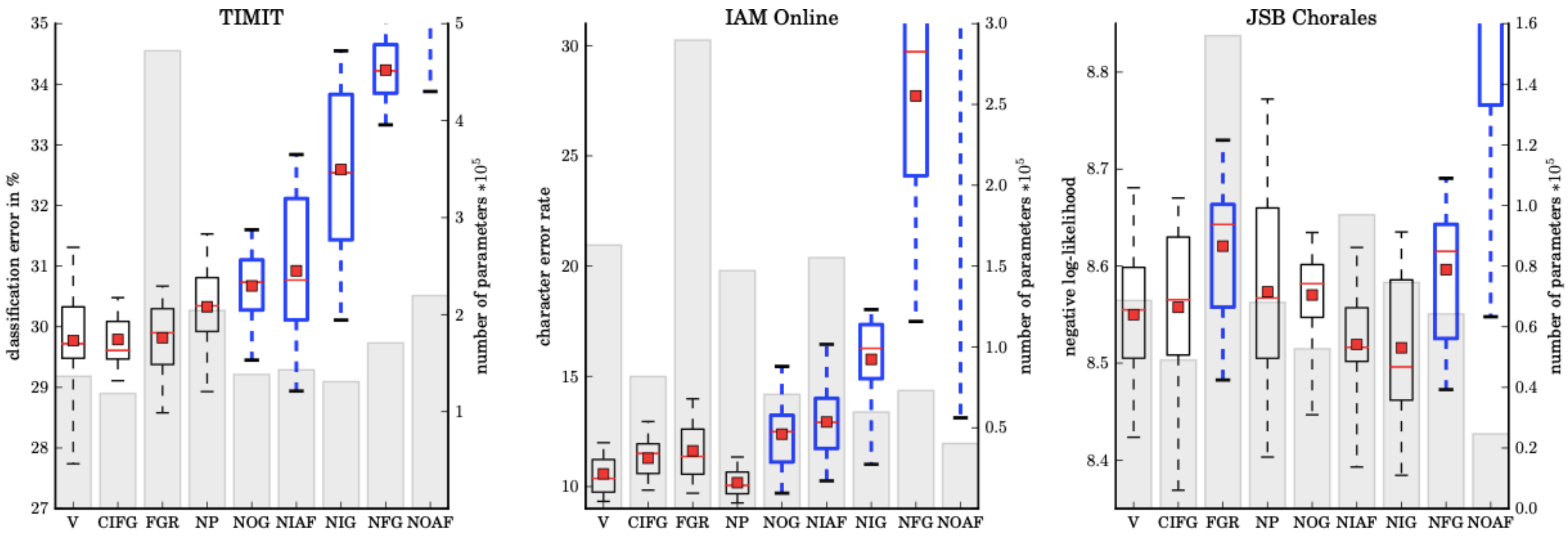
5. Results and Discussion

It’s result time…

Comparison of LSTM Variants (whole)



Comparison of LSTM Variants (top 10%)



Comparison of LSTM Variants

- Removing output activation function and forget gate, drastically reduce LSTM performance.
- CIFG slightly improved performance on Music Modelling, and NP did so in Handwriting Recognition.

Comparison of LSTM Variants

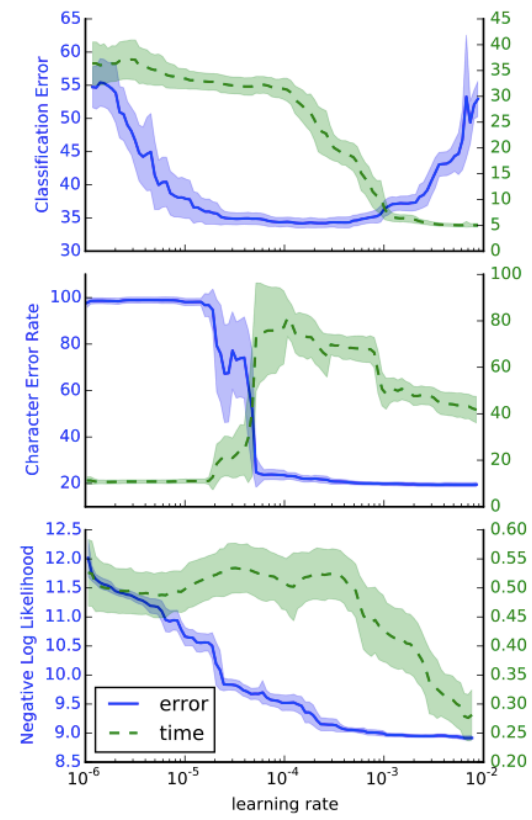
- FGR increases model complexity due to large no. of parameters, and reduces performance on Music Modelling.
- NIG, NOG, NIAF reduce model performance on speech and handwriting recognition significantly, but lead to a slight increase in Music Modelling.

Impact of Hyper-Parameters

- Hyper-Parameters affect Model performance.
- We discuss how each hyper-parameter affects model performance using the fANOVA framework.
- In fANOVA, we vary one hyper-parameter, while averaging over all others using regression trees.

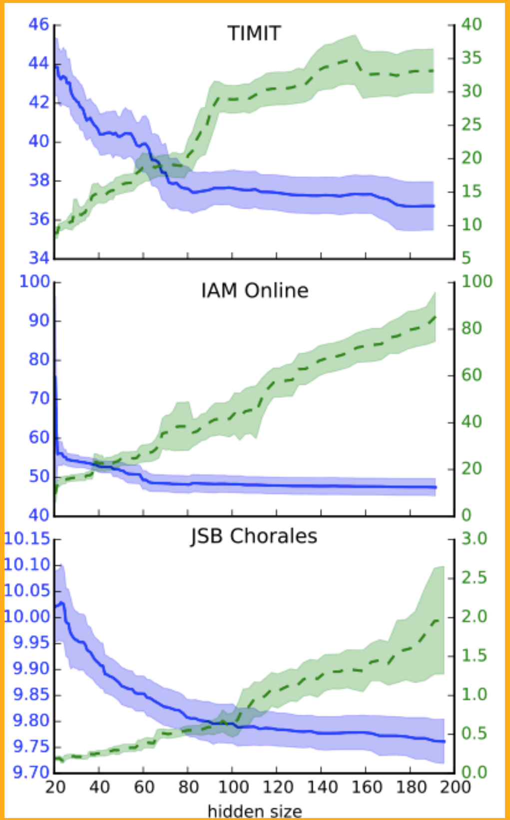
Learning Rate

- For each dataset, there is large basin of good learning rates inside of which performance doesn't vary much.
- Start with a high value, divide by 10 recursively, until performance stops increasing.



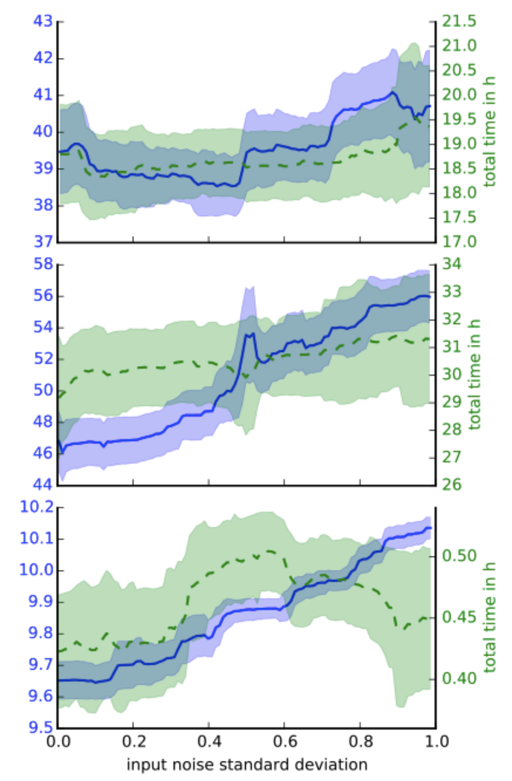
Hidden Layer Size

- As hidden layer size increases, model performance increases but with diminishing returns.
- After a point of time, the marginal time increases a lot, with very low increases in model performance.



Input Size

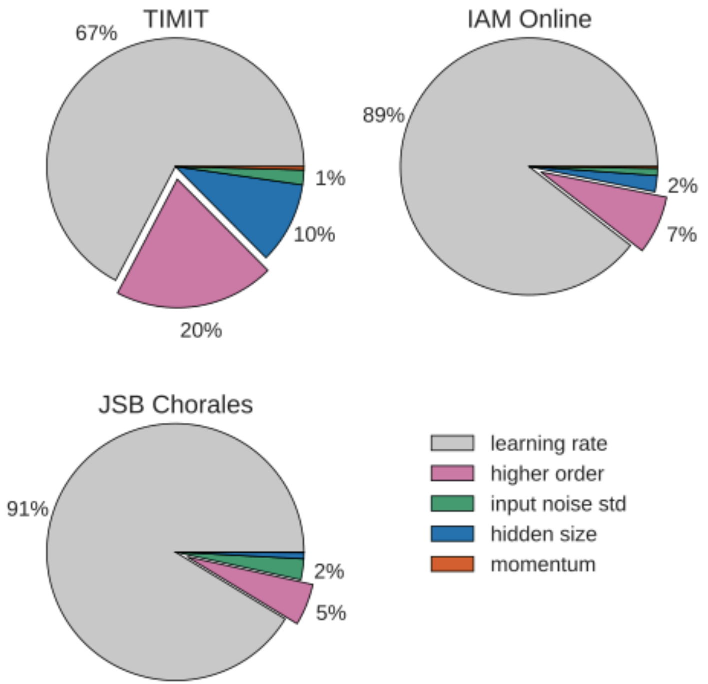
- Additive Gaussian Noise acts as a traditional regularizer for LSTMs just like other Neural Networks.
- But, in the case of LSTMs, it almost always hurts model performance, while increasing training time at the same time.



Momentum

- Momentum doesn’t affect Model performance by more than 1% of the variance of the test set performance.
- This is probably because we scaled the learning rate by a factor of (1 - momentum).

Analysis of Variance



6. Conclusion

But what did we learn?

- Vanilla LSTMs perform reasonably well on various datasets.
- None of the 8 variants improved model performance significantly. However, NP and CIFG simplified the model and reduced computational complexity significantly.

- The Forget Gate, and the Output Activation Function are the most critical components of the LSTM.
- Output activation function is used to take care of the unbounded cell state.

“

- Learning Rate is the most important Hyper-Parameter.
- Hyper-Parameters are almost independent of each other in LSTMs.