An Empirical Exploration of Recurrent Network Architectures

Rafal Jozefowicz, Wojciech Zaremba, Ilya Sutskever
Recurrent Neural Networks

- Recurrent Neural Networks (RNNs) are very capable models for sequence modelling
Recurrent Neural Networks
Standard RNN

\[ H' = \text{Tanh}(W_X X + W_H H + b) \]
Exploding gradients problem

It is relatively easy to handle by simply shrinking gradients whose norms exceed a threshold (Gradient Clipping)

- works well as long as gradient has small norm for majority of time
Vanishing gradients problem

- RNNs can easily learn short-term but not long-term dependencies
- The LSTM addresses this problem by reparameterization
  - Instead of directly computing new state ($S_t$) from the previous one ($S_{t-1}$), the LSTM computes the difference between them and adds to the previous value
    $$S_t = S_{t-1} + \Delta S_t \quad \text{and} \quad S_T = \Sigma_i \Delta S_i$$
  - This way the gradients of the long-term dependencies cannot vanish
LSTM - Long Short-Term Memory

\[ F = \text{Sigmoid}(W_{F_1}X + W_{F_2}H + b_F) \]
\[ I = \text{Sigmoid}(W_{I_1}X + W_{I_2}H + b_I) \]
\[ J = \text{Tanh}(W_{J_1}X + W_{J_2}H + b_J) \]
\[ O = \text{Sigmoid}(W_{O_1}X + W_{O_2}H + b_O) \]
\[ C' = C \times F + I \times J \]
\[ H' = \text{Tanh}(C') \times O \]
LSTM\((X, C, H) = [C', H']\)

\[ F = \text{Lin}_{FX}(X) + \text{Lin}_{FH}(H) \]
\[ I = \text{Lin}_{IX}(X) + \text{Lin}_{IH}(H) \]
\[ J = \text{Lin}_{JX}(X) + \text{Lin}_{JH}(H) \]
\[ O = \text{Lin}_{OX}(X) + \text{Lin}_{OH}(H) \]

where:
\[ \text{Lin}_i(T) = \text{Dot}(W_i, T) + b_i \]

\[ C' = C \text{Sigmoid}(F) + \text{Sigmoid}(I) \text{Tanh}(J) \]
\[ H' = \text{Tanh}(C') \text{Sigmoid}(O) \]
The problem

- The LSTM is complicated and is ad-hoc
- Does there exist a much better RNN architecture?
  - **Goal:** use exhaustive search to determine if there exists an architecture that is much better than the LSTM
- Are the various LSTM gates important?
  - **Goal:** determine what happens when we remove the various LSTM gates
Gated Recurrent Unit - a recently proposed LSTM alternative

\[
R = \text{Sigmoid}(W_R X + W_R H + b_R)
\]
\[
Z = \text{Sigmoid}(W_Z X + W_Z H + b_Z)
\]
\[
\hat{H} = \tanh(W_{\hat{H}} X + W_{\hat{H}} (R \times H) + b_{\hat{H}})
\]
\[
H' = Z \times H + (1 - Z) \times \hat{H}
\]
Experiments

- Our main experiment is an extensive architecture search
  - We evaluated over 10,000 architectures
  - We optimized the hyperparameters of the promising architectures
- We also performed an ablative study of different LSTM variations
Repeat the following steps:

- Pool of promising candidates
- Choosing an architecture to test
- Evaluate on a trivial task
- Evaluate on 3 challenging problems
- Update promising candidate pool
We maintain a list of 100 best performing architectures so far. The pool is initialized with only the LSTM and the GRU. The architectures are ranked with a simple metric:

$$\min_{\text{task}} \frac{\text{architecture's best accuracy on task}}{\text{GRU's best accuracy on task}}$$

By selecting the minimum over all tasks, we make sure that the search procedure will look for an architecture that works well on every task.
Architecture Search - candidate selection

Either:

- Randomly pick an architecture from the pool, evaluate it on a new hyperparameter settings and update its performance estimate
- Propose a new architecture by choosing one from the pool and mutating it
Architecture Search - Mutation

Given a parent architecture, we are allowed to modify it in following ways:

- Insert/Replace/Remove Activation Node:
  Tanh(x), Relu(x), Sigmoid(x), Lin(x), Lin(x)+0.9, Lin(x)+1, Lin(x)+1.1
- Insert/Replace/Remove Elementwise Binary Operator (Add/Mul/Sub)
- Replace Node with one of its ancestors (dependencies) - graph reduction

We pick 1-3 such transformations and apply it to each node with a randomly chosen probability.
Architecture Search - Trivial Task

Simple memorization problem to filter unpromising architectures

- 5 symbols in sequence are to be read and then reproduced in the same order (alphabet size 26)
- Discard architecture if its accuracy is below 95% with teacher forcing
- Might re-examine the architecture in a future on a different set of hyperparameters if it’s randomly picked again at some point
If an architecture passes the first stage, we evaluate it on the first task on a set of 20 new random hyperparameters

- if the parent is heavily evaluated, 80% of hyperparameters comes from best 100 parameters of the parent and the rest is uniform
- otherwise 33% comes from LSTM, 33% from GRU and the rest is uniform

If the results were within 90% of best GRU’s results, we’d evaluate the architecture on the second and then on the third task.
Overall, we evaluated more than 10,000 architectures
1,000 of them succeeded on the trivial task
They were evaluated on average 220 times on 3 problems
Hyperparameter ranges

- The initialization scale is in \{0.3, 0.7, 1, 1.4, 2, 2.8\} and the weights are initialized uniformly in \([-x, x]\), where \(x = \text{scale} / \sqrt{\text{#units in layer}}\)
- The learning rate is in \{0.1, 0.2, 0.3, 0.5, 1, 2, 5\} divided by minibatch size
- The maximal gradient norm was set to \{1, 2.5, 5, 10, 20\}
- The number of layers was chosen from \{1, 2, 3, 4\}
- The number of parameters was also a hyperparameter (task-dependent)
  - But we observed only slight performance gains for the larger models
Evaluation Tasks: ARITHMETIC

- Arithmetic addition and subtraction of up to 8 digit numbers
- The input is given one character at the time
- To make the task more difficult we introduced distractor symbols between the successive input characters.

A typical instance:

\[3e36d9-h1h39f94eeh43keg3c=-13991064.\]

which represents

\[3369-13994433=-13991064.\]
Evaluation Tasks: XML

- The goal is to predict the next character in the synthetic XML dataset.
- To perform this task successfully, the network needs to learn how to stack memory.

A short sample:

```
<etdomp><pegshmnaj><zbhbmg></zbhbmg></pegshmnaj><autmh><autmh></etdomp>
```
Evaluation Tasks: PTB

- Word-level language modeling task on the Penn TreeBank dataset
- 1M words with a vocabulary of size 10,000
- During the architecture search we monitored accuracy for early stopping
Evaluation Tasks: MUSIC

- Polyphonic music datasets from Boulanger-Lewandowski et al. (2012)
- Each timestep is a binary vector representing the played notes
- It’s different than other tasks as we’re predicting binary vectors

We didn’t use these datasets during the architecture search, but we used them to evaluate the learned models.
Additional experiments

- The LSTM has three gates: the input gate $j$, the forget gate $f$, and the output gate $o$
- We removed each of the gates and evaluated the resulting architecture
  - We optimized the hyperparameters for each of the architectures
Most applications of LSTMs simply initialize weights with small random values

- This effectively sets the forget gate to 0.5 and the vanishing gradient with a factor of 0.5 per timestep

\[
C' = C \cdot \text{Sigmoid}(F) + \text{Sigmoid}(I) \cdot \text{Tanh}(J)
\]

\[
H' = \text{Tanh}(C') \cdot \text{Sigmoid}(O)
\]
Most applications of LSTMs simply initialize weights with small random values:
- This effectively sets the forget gate to 0.5 and the vanishing gradient with a factor of 0.5 per timestep.
- The problem can be addressed by initializing the forget gates biases to a large value like 1 or 2.

\[
C' = C \text{ Sigmoid}(F + 1) + \text{Sigmoid}(I)\text{Tanh}(J)
\]

\[
H' = \text{Tanh}(C') \text{ Sigmoid}(O)
\]
Best Architectures found during the search -- similar to GRU

**MUT1**

\[ z = \text{sigm}(W_{xz}x_t + W_{hz}h_t + b_z) \]
\[ r = \text{sigm}(x_t + W_{hr}h_t + b_r) \]
\[ h_{t+1} = \text{tanh}(W_{hh}(r \odot h_t) + W_{xh}x_t + b_h) \odot z + h_t \odot (1 - z) \]

**MUT2**

\[ z = \text{sigm}(W_{xz}x_t + b_z) \]
\[ r = \text{sigm}(W_{xr}x_t + W_{hr}h_t + b_r) \]
\[ h_{t+1} = \text{tanh}(W_{hh}(r \odot h_t) + \text{tanh}(x_t) + b_h) \odot z + h_t \odot (1 - z) \]

**MUT3**

\[ z = \text{sigm}(W_{xz}x_t + W_{hz} \text{tanh}(h_t) + b_z) \]
\[ r = \text{sigm}(W_{xr}x_t + W_{hr}h_t + b_r) \]
\[ h_{t+1} = \text{tanh}(W_{hh}(r \odot h_t) + W_{xh}x_t + b_h) \odot z + h_t \odot (1 - z) \]
<table>
<thead>
<tr>
<th>Arch.</th>
<th>Arith.</th>
<th>XML</th>
<th>PTB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tanh</td>
<td>0.29493</td>
<td>0.32050</td>
<td>0.08782</td>
</tr>
<tr>
<td>LSTM</td>
<td>0.89228</td>
<td>0.42470</td>
<td>0.08912</td>
</tr>
<tr>
<td>LSTM-f</td>
<td>0.29292</td>
<td>0.23356</td>
<td>0.08808</td>
</tr>
<tr>
<td>LSTM-i</td>
<td>0.75109</td>
<td>0.41371</td>
<td>0.08662</td>
</tr>
<tr>
<td>LSTM-o</td>
<td>0.86747</td>
<td>0.42117</td>
<td>0.08933</td>
</tr>
<tr>
<td>LSTM-b</td>
<td>0.90163</td>
<td>0.44434</td>
<td>0.08952</td>
</tr>
<tr>
<td>GRU</td>
<td>0.89565</td>
<td>0.45963</td>
<td>0.09069</td>
</tr>
<tr>
<td>MUT1</td>
<td>0.92135</td>
<td>0.47483</td>
<td>0.08968</td>
</tr>
<tr>
<td>MUT2</td>
<td>0.89735</td>
<td>0.47324</td>
<td>0.09036</td>
</tr>
<tr>
<td>MUT3</td>
<td>0.90728</td>
<td>0.46478</td>
<td>0.09161</td>
</tr>
</tbody>
</table>
## Final evaluation on the MUSIC datasets (NLL)

<table>
<thead>
<tr>
<th>Arch.</th>
<th>N</th>
<th>N-dropout</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tanh</td>
<td>3.612</td>
<td>3.267</td>
<td>6.809</td>
</tr>
<tr>
<td>LSTM</td>
<td>3.492</td>
<td>3.403</td>
<td>6.866</td>
</tr>
<tr>
<td>LSTM-f</td>
<td>3.732</td>
<td>3.420</td>
<td>6.813</td>
</tr>
<tr>
<td>LSTM-i</td>
<td>3.426</td>
<td><strong>3.252</strong></td>
<td>6.856</td>
</tr>
<tr>
<td>LSTM-o</td>
<td>3.406</td>
<td><strong>3.253</strong></td>
<td>6.870</td>
</tr>
<tr>
<td>LSTM-b</td>
<td>3.419</td>
<td>3.345</td>
<td>6.820</td>
</tr>
<tr>
<td>GRU</td>
<td>3.410</td>
<td>3.427</td>
<td>6.876</td>
</tr>
<tr>
<td>MUT1</td>
<td><strong>3.254</strong></td>
<td>3.376</td>
<td><strong>6.792</strong></td>
</tr>
<tr>
<td>MUT2</td>
<td>3.372</td>
<td>3.429</td>
<td>6.852</td>
</tr>
<tr>
<td>MUT3</td>
<td>3.337</td>
<td>3.505</td>
<td>6.840</td>
</tr>
</tbody>
</table>
Final evaluation on the PTB dataset (NLL)

<table>
<thead>
<tr>
<th>Arch.</th>
<th>5M-tst</th>
<th>10M-v</th>
<th>20M-v</th>
<th>20M-tst</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tanh</td>
<td>4.811</td>
<td>4.729</td>
<td>4.635</td>
<td>4.582 (97.7)</td>
</tr>
<tr>
<td>LSTM</td>
<td>4.699</td>
<td>4.511</td>
<td>4.437</td>
<td>4.399 (81.4)</td>
</tr>
<tr>
<td>LSTM-f</td>
<td>4.785</td>
<td>4.752</td>
<td>4.658</td>
<td>4.606 (100.8)</td>
</tr>
<tr>
<td>LSTM-i</td>
<td>4.755</td>
<td>4.558</td>
<td>4.480</td>
<td>4.444 (85.1)</td>
</tr>
<tr>
<td>LSTM-o</td>
<td>4.708</td>
<td>4.496</td>
<td>4.447</td>
<td>4.411 (82.3)</td>
</tr>
<tr>
<td>GRU</td>
<td>4.684</td>
<td>4.554</td>
<td>4.559</td>
<td>4.519 (91.7)</td>
</tr>
<tr>
<td>MUT1</td>
<td>4.699</td>
<td>4.605</td>
<td>4.594</td>
<td>4.550 (94.6)</td>
</tr>
<tr>
<td>MUT2</td>
<td>4.707</td>
<td>4.539</td>
<td>4.538</td>
<td>4.503 (90.2)</td>
</tr>
<tr>
<td>MUT3</td>
<td>4.692</td>
<td>4.523</td>
<td>4.530</td>
<td>4.494 (89.47)</td>
</tr>
</tbody>
</table>
Summary

- The GRU outperformed the LSTM on all tasks with the exception of language modeling.
- MUT1 matched the GRU’s performance on language modeling and outperformed it on all other tasks.
- The LSTM significantly outperformed other architectures on PTB when dropout was allowed.
- Adding large forget gate bias greatly improves the LSTM performance.
- The LSTM’s forget gate is the most important one while the output gate is relatively unimportant.
Thanks!