Reproducing and analyzing Adaptive Computation Time in PyTorch and TensorFlow

Víctor Campos

Source code: https://git.io/vND5S

References:

Abstract

The complexity of solving a problem can differ greatly to the complexity of posing that problem. Building a Neural Network capable of dynamically adapting to the complexity of the inputs is a great challenge for machine learning. One of the most promising approaches is Adaptive Computation Time for Recurrent Neural Network (ACT). In this thesis, we implement ACT in PyTorch and TensorFlow and make both implementations publicly available. We evaluate the capability of ACT to learn algorithms from examples and compare it with a new more appropriate baseline to discuss its benefits and drawbacks with respect to a classic Recurrent Neural Network.

Main Contributions

- Implementation of ACT in two of the most popular deep learning frameworks.
- Design and implementation based on a novel baseline capable of solving the same tasks as ACT, based on a fixed amount of repetitions set at training time as a new hyperparameter.
- Comparison of ACT with the proposed baseline which, surprisingly, indicates that ACT does not bring any gain and opening this way unexpected directions for research that we plan to address in the future.

Adaptive Computation Time

Recurrent Neural Networks (RNNs) are used in sequence modeling tasks. A simple RNN takes an input sequence \( x = (x_1, ..., x_T) \) and generates a state sequence \( s = (s_1, ..., s_T) \) by iteratively applying a parametric state transition model \( S \) from \( t=1 \) to \( T \):

\[
s_1 = S(s_{t-1}, x_t)
\]

Adaptive Computation Time (ACT) allows the network to ponder each input \( x_i \) more than once, and decide how many times to do it adaptively. Its new equations are:

\[
s^o_i = \begin{cases} 
S(s_{t-1}, x_t) & \text{if } n = 1 \\
S(s_{t-1}, x_t) & \text{if } n > 1 
\end{cases}
\]

To decide when to stop pondering an input, a sigmoidal unit is added. This sigmoidal unit gives the halting probabilities at each step, and when the cumulative probability reaches one, the network stops and gives the output.

\[
p^t_i = \sigma(W^p s^o_i + b^p) \\
N(t) = \min \left\{ k : \sum_{j=1}^{k} p_j^t > 1 - \varepsilon \right\} \\
s_t = \sum_{j=1}^{N(t)} p_j^t s_j^t
\]

Limiting Computation

We define the ponder cost of a sequence as:

\[
P(x) = \frac{1}{N(t)} \sum_{t=1}^{N(t)} (N(t) + R(t))
\]

And minimize it adding it to the loss.

\[
\tilde{L}(x, y) = L(x, y) + \tau P(x)
\]

We can also limit the maximum amount of iterations at each timestep to \( M \):

\[
N(t) = \min \left\{ M, \min \left\{ n' : \sum_{k=1}^{n'} p^t_k > 1 - \varepsilon \right\} \right\}
\]

New Baseline: Repeat-RNN

We decided on a new baseline with which to compare ACT, consisting on repeating the input at each time step a fixed number of times. We augmented each input with a binary flag for the network to be able to differentiate between repeated inputs and new inputs.

We can also limit the maximum amount of iterations at each timestep to \( M \):

\[
N(t) = \min \left\{ M, \min \left\{ n' : \sum_{k=1}^{n'} p^t_k > 1 - \varepsilon \right\} \right\}
\]

Surprisingly, this new and simple modification performs as good or even better than ACT, opening new directions for research.

Experiments

Parity

<table>
<thead>
<tr>
<th>Model</th>
<th>Task solved</th>
<th>Updates until solved</th>
<th>Mean repetitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>RNN</td>
<td>No</td>
<td>-</td>
<td>1.000</td>
</tr>
<tr>
<td>ACT-RNN, ( \tau = 10^{-5} )</td>
<td>Yes</td>
<td>3000</td>
<td>1.805</td>
</tr>
<tr>
<td>ACT-RNN, ( \tau = 10^{-3} )</td>
<td>Yes</td>
<td>35600</td>
<td>2.027</td>
</tr>
<tr>
<td>ACT-RNN, ( \tau = 5 \cdot 10^{-3} )</td>
<td>Yes</td>
<td>5000</td>
<td>2.044</td>
</tr>
<tr>
<td>Repeat-RNN, ( \rho = 2 )</td>
<td>Yes</td>
<td>2000</td>
<td>2</td>
</tr>
<tr>
<td>Repeat-RNN, ( \rho = 3 )</td>
<td>Yes</td>
<td>4000</td>
<td>3</td>
</tr>
<tr>
<td>Repeat-RNN, ( \rho = 5 )</td>
<td>Yes</td>
<td>7000</td>
<td>5</td>
</tr>
<tr>
<td>Repeat-RNN, ( \rho = 8 )</td>
<td>Yes</td>
<td>2000</td>
<td>8</td>
</tr>
</tbody>
</table>

Addition

<table>
<thead>
<tr>
<th>Model</th>
<th>Task solved</th>
<th>Updates until solved</th>
<th>Mean repetitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSTM</td>
<td>No</td>
<td>-</td>
<td>1.012</td>
</tr>
<tr>
<td>ACT-LSTM, ( \tau = 10^{-1} )</td>
<td>Yes</td>
<td>89000</td>
<td>5.679</td>
</tr>
<tr>
<td>ACT-LSTM, ( \tau = 10^{-3} )</td>
<td>Yes</td>
<td>98800</td>
<td>6.736</td>
</tr>
<tr>
<td>ACT-LSTM, ( \tau = 5 \cdot 10^{-3} )</td>
<td>Yes</td>
<td>11900</td>
<td>11.907</td>
</tr>
<tr>
<td>Repeat-LSTM, ( \rho = 2 )</td>
<td>Yes</td>
<td>99700</td>
<td>2</td>
</tr>
<tr>
<td>Repeat-LSTM, ( \rho = 3 )</td>
<td>Yes</td>
<td>51400</td>
<td>3</td>
</tr>
<tr>
<td>Repeat-LSTM, ( \rho = 5 )</td>
<td>Yes</td>
<td>57800</td>
<td>5</td>
</tr>
<tr>
<td>Repeat-LSTM, ( \rho = 8 )</td>
<td>Yes</td>
<td>57800</td>
<td>8</td>
</tr>
</tbody>
</table>

Reference: