Neural Speed Reading

Minjoon Seo\textsuperscript{1,2*}, Sewon Min\textsuperscript{3*}, Ali Farhadi\textsuperscript{2,4,5}, Hannaneh Hajishirzi\textsuperscript{2}
NAVER Clova\textsuperscript{1}, University of Washington\textsuperscript{2}, Seoul National University\textsuperscript{3},
Allen Institute for AI\textsuperscript{4}, XNOR.AI\textsuperscript{5}
Feb 15, 2018
* denotes equal contribution.
Sentiment Classification with Recurrent Neural Network (RNN)
RNNs are slow...

- RNNs cannot be parallelized over time
  - Time complexity is linear in the length of sequence.
  - GPUs cannot take full advantage of parallelization.
  - Recent works to overcome slow RNNs: Google’s Transformer (2017), Facebook’s CNN-based MT (2017).
RNNs are slow...

• RNNs cannot be parallelized over time
  • Time complexity is linear in the length of sequence.
  • GPUs cannot take full advantage of parallelization.
  • Recent works to overcome slow RNNs: Google’s Transformer (2017), Facebook’s CNN-based MT (2017).

• CPUs are not great for RNNs either
  • Neural networks require so many computations.
(very rough) FLOP Complexity of RNN

• Hidden state size = \( d \) (= input size)

• \( h_{t+1} = \sigma(W^{(x)}x_t + W^{(h)}h_t + b) \)

• Total number of operations: \( 3d + 2d^2 \)
• If \( d \) is sufficiently large, matrix multiplication is the bottleneck.
Can we improve inference speed on CPUs?

- CPUs are often more desirable options for production
- Small devices (often CPU-only)
- Latency-critical applications
  - CPUs can have lower latency.
How do humans ‘speed-read’?

• Speed readers **skim** unimportant parts and **fully read** important text (Just and Carpenter, 1987).
How do humans ‘speed-read’?

• Speed readers **skim** unimportant parts and **fully read** important text (Just and Carpenter, 1987).

• ‘Reading’ is similar to *matrix multiplication* in RNN.
  • **Skim**: use small matrix multiplication.
  • **Fully read**: use big matrix multiplication.
Sentiment Classification with Skim-RNN

\[ h_0 \]

\[ x_1 \]

“Intelligent”
Sentiment Classification with Skim-RNN

“Intelligent”
Sentiment Classification with Skim-RNN

"Intelligent"
Sentiment Classification with Skim-RNN

```
Intelligent
```

```
# READ!
```

```
Q_1=1
```

```
x_1
```

```
"Intelligent"
```
Sentiment Classification with Skim-RNN

![Diagram of Skim-RNN model]

- Input: \( x_1 \) ("Intelligent")
- Input: \( x_2 \) ("and")
- Read transition: \( Q_1 = 1 \)
- Hidden state: \( h_0 \) (left) and \( h_1 \) (right)
- Big RNN block

**Key Points**:
- Skim-RNN approach for sentiment classification.
- Utilizes recurrent neural networks (RNNs) for processing sequences of words.
- The model reads different parts of the input sequence based on a conditional variable \( Q_1 \).
Sentiment Classification with Skim-RNN

Intelligent and

$Q_1 = 1$

$Q_2 = 2$

"Intelligent" "and"
Sentiment Classification with Skim-RNN
Sentiment Classification with Skim-RNN

\[ h_0 \rightarrow \text{Big RNN} \]

\[ Q_1 = 1 \]

\[ \text{READ} \]

\[ x_1 \]

"Intelligent"

\[ h_1 \rightarrow \text{Small RNN} \]

\[ Q_2 = 2 \]

\[ \text{SKIM} \]

\[ x_2 \]

"and"
Sentiment Classification with Skim-RNN

Intelligent and invigorating film
Skim-RNN

• Consists of two RNNs:
  • **Big RNN**: hidden state size = \(d\)
  • **Small RNN**: hidden state size = \(d'\)
  • \(d >> d'\) (e.g. \(d=100, d'=5\))
Skim-RNN

• Consists of two RNNs:
  • **Big RNN**: hidden state size = $d$
  • **Small RNN**: hidden state size = $d'$
  • $d >> d'$ (e.g. $d=100$, $d'=5$)

• Hidden state is shared between the RNNs.
• Big RNN updates the entire hidden state.
• Small RNN updates only a small portion of the hidden state.
Skim-RNN

- Consists of two RNNs:
  - **Big RNN**: hidden state size = $d$
  - **Small RNN**: hidden state size = $d'$
  - $d >> d'$ (e.g. $d=100$, $d'=5$)

- Hidden state is shared between the RNNs.
- Big RNN updates the entire hidden state.
- Small RNN updates only a small portion of the hidden state.
- When using small RNN, the inference requires smaller # of FLOP.
  - $O(d^2) >> O(d'd)$
- Dynamically makes decision on which size of RNN to use
How to train?

• Decisions (which RNN to use) are so non-differentiable

• Policy gradient (Williams, 1992)
  • REINFORCE
  • Unbiased estimation
  • High variance; hard to train

• Gumbel-softmax (Jang et al., 2017)
  • Biased estimation
  • Low variance; good empirical results
  • Fully differentiable during training via reparameterization
Gumbel-softmax Reparameterization

- Start with soft decision (attention) $p$

$$r_t^i = \frac{\exp((\log(p_t^i) + g_t^i)/\tau)}{\sum_j \exp((\log(p_t^j) + g_t^j)/\tau)}$$

- Slowly decrease temperature ($\tau$), making the distribution more discrete

- Sampling with the attention weights ($r$) approximate true distribution

- Reparameterization ($g$) allows differentiation with stochasticity
LSTM-Jump (Yu et al., 2017)

• Orthogonal
  • Skipping: you decide to skip next words / sentences before reading at all.
  • Skimming: you use very small RNN to read unimportant words fast.
    • Both can be combined.

• Concurrent work
LSTM-Jump (Yu et al., 2017)

• Orthogonal
  • Skipping: you decide to skip next words / sentences before reading at all.
  • Skimming: you use very small RNN to read unimportant words fast.
  • Both can be combined.

• Concurrent work

• Skim-RNN has output for every time step
  • Useful for applications that need output every time.
  • Easy to replace existing RNNs.

• LSTM-Jump has GPU advantage
### Classification Results

<table>
<thead>
<tr>
<th>LSTM Model</th>
<th>$d'/\gamma$</th>
<th>SST</th>
<th></th>
<th>Rotten Tomatoes</th>
<th></th>
<th>IMDb</th>
<th></th>
<th>AGNews</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Acc</td>
<td>Sk</td>
<td>Flop-r</td>
<td>Sp</td>
<td>Acc</td>
<td>Sk</td>
<td>Flop-r</td>
<td>Sp</td>
</tr>
<tr>
<td>Standard</td>
<td>86.4</td>
<td>-</td>
<td>1.0x</td>
<td>1.0x</td>
<td></td>
<td>82.5</td>
<td>-</td>
<td>1.0x</td>
<td>1.0x</td>
</tr>
<tr>
<td>Skim</td>
<td>5/0.01</td>
<td>86.4</td>
<td>58.2</td>
<td>2.4x</td>
<td>1.4x</td>
<td>84.2</td>
<td>52.0</td>
<td>2.1x</td>
<td>1.3x</td>
</tr>
<tr>
<td>Skim</td>
<td>10/0.01</td>
<td>85.8</td>
<td>61.1</td>
<td>2.5x</td>
<td>1.5x</td>
<td>82.5</td>
<td>58.5</td>
<td>2.4x</td>
<td>1.4x</td>
</tr>
<tr>
<td>Skim</td>
<td>5/0.02</td>
<td>85.6</td>
<td>62.3</td>
<td>2.6x</td>
<td>1.5x</td>
<td>81.8</td>
<td>63.7</td>
<td>2.7x</td>
<td>1.5x</td>
</tr>
<tr>
<td>Skim</td>
<td>10/0.02</td>
<td>86.4</td>
<td>68.0</td>
<td>3.0x</td>
<td>1.7x</td>
<td>82.5</td>
<td>63.0</td>
<td>2.6x</td>
<td>1.5x</td>
</tr>
<tr>
<td>LSTM-Jump</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>79.3</td>
<td>-</td>
<td>-</td>
<td>1.6x</td>
</tr>
<tr>
<td>SOTA</td>
<td>89.5</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>83.4</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
**Question Answering Results**

<table>
<thead>
<tr>
<th>Model</th>
<th>( \gamma )</th>
<th>F1</th>
<th>EM</th>
<th>Sk</th>
<th>Flop-r</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSTM+Att (1 layer)</td>
<td>-</td>
<td>73.3</td>
<td>63.9</td>
<td>-</td>
<td>1.3x</td>
</tr>
<tr>
<td>LSTM+Att ((d = 50))</td>
<td>-</td>
<td>74.0</td>
<td>64.4</td>
<td>-</td>
<td>3.6x</td>
</tr>
<tr>
<td>LSTM+Att</td>
<td>-</td>
<td>75.5</td>
<td>67.0</td>
<td>-</td>
<td>1.0x</td>
</tr>
<tr>
<td>Sk-LSTM+Att ((d' = 0))</td>
<td>0.1</td>
<td>75.7</td>
<td>66.7</td>
<td>37.7</td>
<td>1.4x</td>
</tr>
<tr>
<td>Sk-LSTM+Att ((d' = 0))</td>
<td>0.2</td>
<td>75.6</td>
<td>66.4</td>
<td>49.7</td>
<td>1.6x</td>
</tr>
<tr>
<td>Sk-LSTM+Att</td>
<td>0.05</td>
<td>75.5</td>
<td>66.0</td>
<td>39.7</td>
<td>1.4x</td>
</tr>
<tr>
<td>Sk-LSTM+Att</td>
<td>0.1</td>
<td>75.3</td>
<td>66.0</td>
<td>56.2</td>
<td>1.7x</td>
</tr>
<tr>
<td>Sk-LSTM+Att</td>
<td>0.2</td>
<td>75.0</td>
<td>66.0</td>
<td>76.4</td>
<td>2.3x</td>
</tr>
<tr>
<td>BiDAF ((d = 30))</td>
<td>-</td>
<td>74.6</td>
<td>64.0</td>
<td>-</td>
<td>9.1x</td>
</tr>
<tr>
<td>BiDAF ((d = 50))</td>
<td>-</td>
<td>75.7</td>
<td>65.5</td>
<td>-</td>
<td>3.7x</td>
</tr>
<tr>
<td>BiDAF</td>
<td>-</td>
<td>77.3</td>
<td>67.7</td>
<td>-</td>
<td>1.0x</td>
</tr>
<tr>
<td>Sk-BiDAF (0.01)</td>
<td>0.01</td>
<td>76.9</td>
<td>67.0</td>
<td>74.5</td>
<td>2.8x</td>
</tr>
<tr>
<td>Sk-BiDAF (0.001)</td>
<td>0.001</td>
<td>77.1</td>
<td>67.4</td>
<td>47.1</td>
<td>1.7x</td>
</tr>
<tr>
<td>SOTA (Wang et al., 2017)</td>
<td></td>
<td>79.5</td>
<td>71.1</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Comparing F1 & FLOP across diff configs.
### Visualization on IMDb Sentiment Classification

| Positive | I liked this movie, not because Tom Selleck was in it, but because it was a good story about baseball and it also had a semi-over dramatized view of some of the issues that a BASEBALL player coming to the end of their time in Major League sports must face. I also greatly enjoyed the cultural differences in American and Japanese baseball and the small facts on how the games are played differently. Overall, it is a good movie to watch on Cable TV or rent on a cold winter’s night and watch about the "Dog Day’s" of summer and know that spring training is only a few months away. A good movie for a baseball fan as well as a good "DATE" movie. Trust me on that one! *Wink*
| Negative | No! no - No - NO! My entire being is revolting against this dreadful remake of a classic movie. I knew we were heading for trouble from the moment Meg Ryan appeared on screen with her ridiculous hair and clothing - literally looking like a scarecrow in that garden she was digging. Meg Ryan playing Meg Ryan - how tiresome is that?! And it got worse ... so much worse. The horribly cliché lines, the stock characters, the increasing sense I was watching a spin-off of "The First Wives Club" and the ultimate hackneyed schtick in the delivery room. How many times have I seen this movie? Only once, but it feel like a dozen times - nothing original or fresh about it. For shame!

*Black words are skimmed (small RNN), blue words are fully read.*
Visualization on Stanford Question Answering Dataset

<table>
<thead>
<tr>
<th>Q</th>
<th>What is one straightforward case of a probabilistic test?</th>
</tr>
</thead>
<tbody>
<tr>
<td>A particularly simple example of a <strong>probabilistic test</strong> is the <strong>Fermat primality test</strong>, which relies on the fact (<strong>Fermat</strong>'s little theorem) that $np \equiv n \pmod{p}$ for any $n$ if $p$ is a prime number. If you have a number $b$ that we want to <strong>test</strong> for <strong>primality</strong>, then we work out $nb \pmod{b}$ for a random value of $n$ as our <strong>test</strong>. A <strong>flaw</strong> with this <strong>test</strong> is that there are some composite numbers (<strong>the <strong>Carmichael</strong> numbers</strong>) that satisfy the <strong>Fermat</strong> identity even though they are not prime, so the <strong>test</strong> has no way of distinguishing between prime numbers and <strong>Carmichael</strong> numbers. <strong>Carmichael</strong> numbers are substantially rarer than prime numbers, though, so this <strong>test</strong> can be useful for practical purposes. More powerful extensions of the <strong>Fermat primality test</strong>, such as Baillie-PSW, Miller-Rabin, and Solovay-Strassen <strong>tests</strong>, are guaranteed to fail at least some of the time when applied to a composite number.</td>
<td></td>
</tr>
</tbody>
</table>

*Black words are skimmed (small RNN), blue words are fully read.*
Layer-wise Skim Visualization (SQuAD)

The successful scheduling, budgeting, construction-site safety, availability and transportation of building materials, logistics, inconvenience to the public caused by construction delays and bidding, etc. The largest construction projects are referred to as megaprojects.

Most RNN steps at higher layer is redundant!
Dynamically controlling # of FLOP

Flop-R (Float operation Reduction)

F1

\[ d' = 10 \]

\[ d' = 0 \]
Stability of Pretraining

![Graph showing the stability of pretraining across different layers and skim rates. The x-axis represents the layers (1st (fw), 1st (bw), 2nd (fw), 2nd (bw)), and the y-axis represents the skim rate. The graph includes various markers and lines indicating different skim rates and layer configurations.](image)
FLOP = Speed?

• On GPU: empirically NO
• On CPU: conditionally YES, with low-level programming
TensorFlow or PyTorch?

- **NumPy** is faster for small matrices
  - On CPUs (NumPy has no GPU compatibility)
  - Batch size = 1 (latency, not throughput)
  - $d < 220$ for TensorFlow
  - $d < 700$ for PyTorch

- TensorFlow and PyTorch seem to have more overheads
- All benchmarks are based on NumPy
Theoretical and Actual Speed Gains on NumPy
Conclusion

• **Skim-RNN**: switching between two different-size RNNs with shared hidden state.
  • Can be generalized to multiple RNNs.
• Speed gain can be substantial.
• More beneficial with larger hidden state size.
• Especially useful for latency.
  • To get throughput advantage, will need to go low-level.
Future Work

• Using multiple granularities of RNNs (not just two)
• Extension to latency-critical applications
  • Speech
  • Video
• Low-level implementation
Thanks!

• http://seominjoon.github.io

• minjoon.seo@