Parallelizable StackLSTM

Shuoyang Ding   Philipp Koehn

NAACL 2019 Structured Prediction Workshop
Minneapolis, MN, United States
June 7th, 2019
Outline

• What is StackLSTM?
• Parallelization Problem
• Homogenizing Computation
• Experiments
What is StackLSTM?
A Partial Tree

join

Vinken

will

Pierre, old,

years

61
Good Edge?

join

Vinken

will

board

Pierre, old,

years

the

61
Good Edge?

- Vinken
- will
- board
- join
- Pierre
- old
- the
- years
- 61
LSTM?

Pierre Vinken, 61 years old, will join
Pierre Vinken, 61 years old, will join...
StackLSTM

- An LSTM whose states are stored in a stack
- Computation is conditioned on the stack operation

*Dyer et al. (2015)*
*Ballesteros et al. (2017)*
StackLSTM

Vinken
Push,

Vinken,
Pop

Vinken
Push 61

Vinken, 61
Push years

Vinken, 61 years
Push old

Vinken, 61 years old
Pop

Vinken, 61 years old
Pop

Vinken, 61 years old
Pop

Vinken, 61 years old
Push,

Vinken, 61 years old,
Pop

Vinken, 61 years old,
Push will

Vinken, 61 years old, will
Push join

Vinken, 61 years old, will join
Parallelizable StackLSTM

Vinken, 61 years old, will join
Parallelization Problem
LSTM

Parallelizable StackLSTM
Parallelizable StackLSTM
Batched LSTM

Parallelizable StackLSTM
Batched... StackLSTM?
Parallelizable StackLSTM
Wouldn’t it be nice if…

Parallelizable StackLSTM
Homogenizing Computation
Push

- read the stack top hidden state $h_{p(t)}$;
- perform LSTM forward computation with $x(t)$ and $h_{p(t)}$;
- write new hidden state to $h_{p(t) + 1}$;
- update stack top pointer $p(t+1) = p(t) + 1$;
Push

- read the stack top hidden state $h_{\{p(t)\}}$;

- perform LSTM forward computation with $x(t)$ and $h_{\{p(t)\}}$;

- write new hidden state to $h_{\{p(t) + 1\}}$;

- update stack top pointer $p(t+1) = p(t) + 1$;
Push

- read the stack top hidden state $h_{p(t)}$;
- perform LSTM forward computation with $x(t)$ and $h_{p(t)}$;
- write new hidden state to $h_{p(t) + 1}$;
- update stack top pointer $p(t+1) = p(t) + 1$;
Push

- read the stack top hidden state $h_{p(t)}$;
- perform LSTM forward computation with $x(t)$ and $h_{p(t)}$;
- write new hidden state to $h_{p(t) + 1}$;
- update stack top pointer $p(t+1) = p(t) + 1$;
Push

• read the stack top hidden state $h_{\{p(t)\}}$;

• perform LSTM forward computation with $x(t)$ and $h_{\{p(t)\}}$;

• write new hidden state to $h_{\{p(t) + 1\}}$;

• update stack top pointer
  $p(t+1) = p(t) + 1$;
- update stack top pointer
  \[ p(t+1) = p(t) - 1; \]
Pop

- update stack top pointer
  \[ p(t+1) = p(t) - 1; \]
Observation 1

- read the stack top hidden state $h_{p(t)}$;

- perform LSTM forward computation with $x(t)$ and $h_{p(t)}$;

- write new hidden state to $h_{p(t)+1}$;

- update stack top pointer $p(t+1) = p(t) + 1$;

- update stack top pointer $p(t+1) = p(t) - 1$;
Observation 1

- read the stack top hidden state $h_{p(t)}$;
- perform LSTM forward computation with $x(t)$ and $h_{p(t)}$;
- write new hidden state to $h_{p(t) + 1}$;
- update stack top pointer $p(t+1) = p(t) + op$;

Use $op = +1$ for push and $op = -1$ for pop
Observation 1

The computation performed for Pop operation is a subset of Push operation.
Observation 2

Is it safe to do the other computations for push for pop as well?
Observation 2

- read the stack top hidden state $h_{p(t)}$;

- perform LSTM forward computation with $x(t)$ and $h_{p(t)}$;

- write new hidden state to $h_{p(t) + 1}$;

- update stack top pointer $p(t+1) = p(t) + op$;
Observation 2

- read the stack top hidden state $h_{\{p(t)\}}$;

- perform LSTM forward computation with $x(t)$ and $h_{\{p(t)\}}$;

- update stack top pointer $p(t+1) = p(t) + op$;

- write new hidden state to $h_{\{p(t) + 1\}}$;

- update stack top pointer $p(t+1) = p(t) + op$;
Observation 2

A write will always happen before the stack top pointer advances.
Observation 2

If one wants to write anything in the higher position than the current stack top pointer...
Observation 2

If one wants to write anything in the higher position than the current stack top pointer...

Just do it!
Observation 2

- read the stack top hidden state $h_{p(t)}$;
- perform LSTM forward computation with $x(t)$ and $h_{p(t)}$;
- write new hidden state to $h_{p(t) + 1}$;
- update stack top pointer $p(t + 1) = p(t) + \text{op}$;
Observation 2

- read the stack top hidden state $h_{p(t)}$;
- perform LSTM forward computation with $x(t)$ and $h_{p(t)}$;
- write new hidden state to $h_{p(t) + 1}$;
- update stack top pointer $p(t+1) = p(t) + op$;
- read the stack top hidden state $h_{p(t)}$;
- perform LSTM forward computation with $x(t)$ and $h_{p(t)}$;
- write new hidden state to $h_{p(t) + 1}$;
- update stack top pointer $p(t+1) = p(t) + op$;
Done!

- read the stack top hidden state $h_{p(t)}$;
- perform LSTM forward computation with $x(t)$ and $h_{p(t)}$;
- write new hidden state to $h_{p(t) + 1}$;
- update stack top pointer $p(t+1) = p(t) + op$;
Experiments
Benchmark

Transition-based dependency parsing on Stanford Dependency Treebank

PyTorch, Single K80 GPU
Hyperparameters

- Largely following Dyer et al. (2015); Ballesteros et al. (2017), except:
  - Adam w/ ReduceLROnPlateau and warmup
  - Arc-Hybrid w/o composition function
  - Slightly larger models (200 hidden, 200 state, 48 action embedding) perform better
Speed

![Graph showing the relationship between training speed and batch size. The red line represents the training speed, and the blue dashed line represents Dyer et al. 2015. The x-axis represents the batch size, and the y-axis represents sentences per second (sent/s).]
Speed

![Graph showing the relationship between batch size and training speed.](graph.png)
Performance

Parallelizable StackLSTM
Conclusion
Conclusion

• We propose a parallelization scheme for StackLSTM architecture.

• Together with a different optimizer, we are able to train parsers of comparable performance within 1 hour.

https://github.com/shuoyangd/hoolock