Neural Machine Translation Hackathon

SMT Group
Informatics Institue
University of Amsterdam

Motivation

- Share a common vocabulary
- Stop worry and love neural networks
- Hand on experience with neural nets
- Build our NMT system

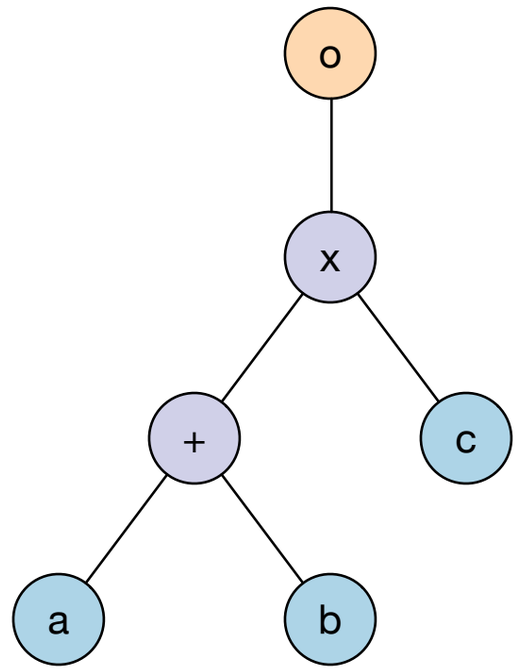
Content

- Introduction to Computational Graph & Torch-Autograd
- Long Short-Term Memory (LSTM) & Gated Recurrent Unit (GRU)
- Bidirectional LSTM, GRU
- Encoder-Decoder with LSTM/GRU (seq2seq)
- Attentional LSTM/GRU (Neural Machine Translation)

Focus

- The NLP way
    - Data processing
- Tensor manipulation
    - So you can build your model in other deep learning frameworks such as Tensorflow, Theano, Chainer…
- Translating from equations to code
    - Deep learning is moving really fast. You should be able to implement a new model within few hours
- Collaboration

Computational Graph




```
local o = (a+b)*c
```

- The graph above has tree structure (think of CFG in compiler's parser)

- Computational graph is a graph that describes data flow and transformation

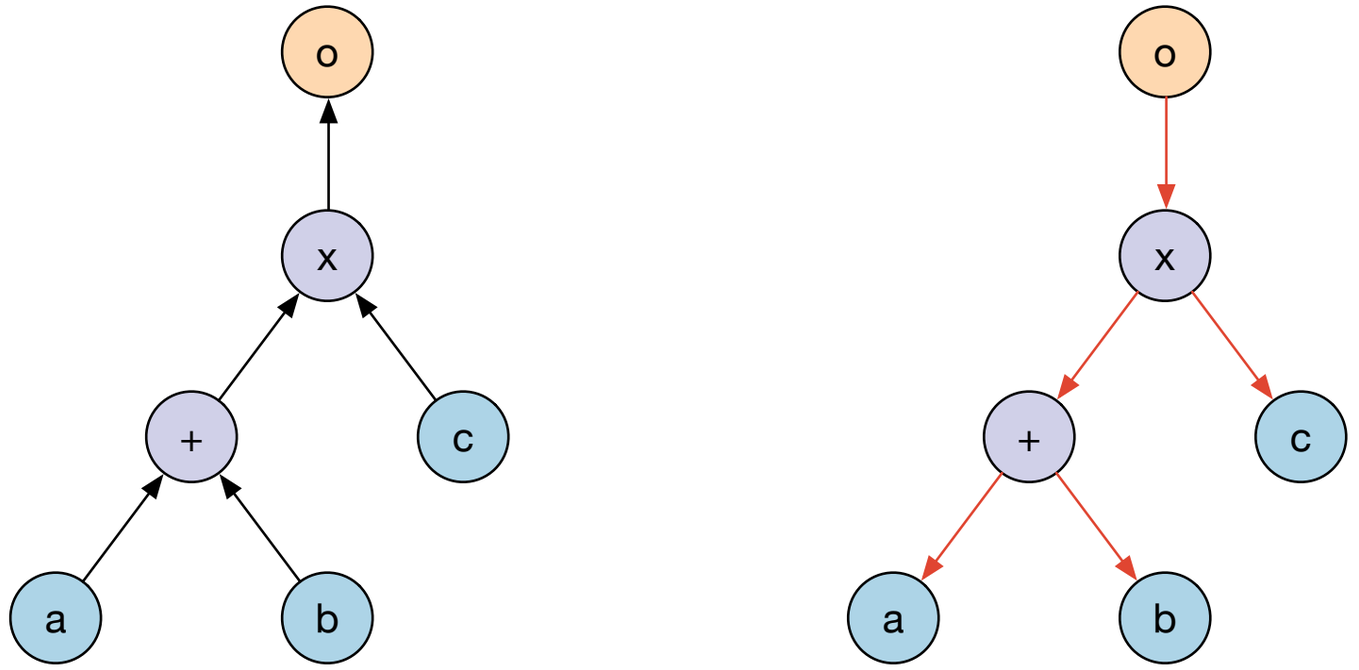
Why do we need CG?

- Setup
  - We have data **X**, model parameter **W** and an objective function *f*(**X**,**W**). We want to find **W** that minimizes *f*(**X**,**W**)
- How we do it when *f* is differentiable?
  - find derivative of *f*(**X**,**W**) w.r.t **W**
  - update **W** using SGD

Why do we need CG?

- Setup
  - We have data **X**, model parameter **W** and an objective function *f*(**X**,**W**). We want to find **W** that minimizes *f*(**X**,**W**)
- How we do it when *f* is differentiable?
  - find derivative of *f*(**X**,**W**) w.r.t **W**
  - update **W** using SGD

Why do we need CG?

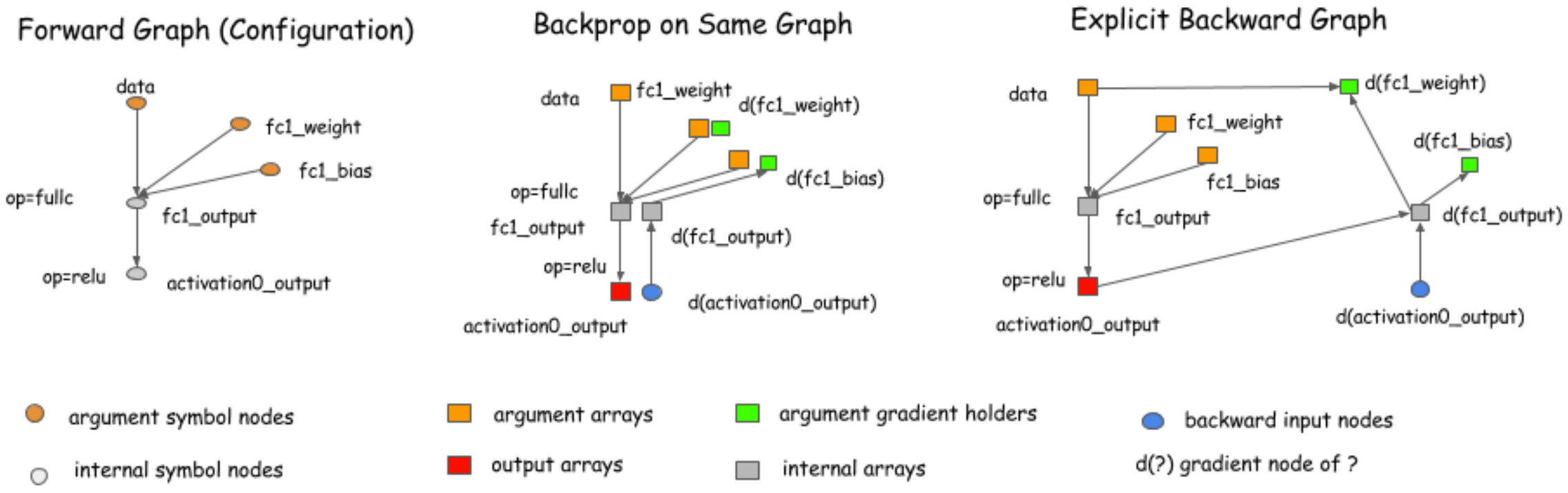


- Chain rule
- Traverse back the graph to compute gradient
- Each node in the graph needs to know gradient come into it from parents node and the data come from child nodes.

http://colah.github.io/posts/2015-08-Backprop/

What is CG good for?

- *Computational Graph* is collocated with *Automatic differentiation*
- CG is useful for computing gradient of very complicated functions
- Fitting billions of parameters into 12GB onboard memory



https://mxnet.readthedocs.org/en/latest/developer-guide/note_memory.html

History

- Before November 19, 2014
    - Torch was cool for Feedforward Neural Nets and ConvNets,
    - Nobody knows how to build (even a simple) RNN with Torch,
    - Many were increasingly of the opinion that they'd all made a big mistake in moving to Torch from Theano,
    - And some said that even Theano had been a bad move, and that no one should ever have came down from the trees.

History

- November 19, 2014 and after
  - Wojciech Zaremba released LSTM code in Torch. It's implemented with **nngraph**
  - Ever since, everybody uses the clone many times code from Zaremba's code

163 lines (135 sloc) | 5.03 KB

```
1
2  -- adapted from https://github.com/wojciechz/learning_to_execute
3  -- utilities for combining/flattening parameters in a model
4  -- the code in this script is more general than it needs to be, which is
5  -- why it is kind of a large
6
7  require 'torch'
8  local model_utils = {}
9  function model_utils.combine_all_parameters(...)
```

- Now
  - No need to clone in memory. Thanks to Torch-Autograd, Tensorflow

Torch-Autograd

- From Twitter: https://github.com/twitter/torch-autograd
- Inspired by Python Autograd
- Original Python Autograd does not support GPUs (Dec 8, 2015)
- Torch-Autograd supports GPUs and based on Torch
- Help us do complicated stuff at ease

Torch-Autograd

Setup

We have data **X**, model parameter **W** and an objective function *f*(**X**,**W**). We want to find **W** that minimizes *f*(**X**,**W**)

```
– define parameters that will be optimized
local params = {}
params.emb = torch.Tensor(vocab_size,emb_dim)
params.W1 = torch.Tensor(2*emb_dim,hid_dim)
params.b1 = torch.Tensor(hid_dim)
params.W2 = torch.Tensor(hid_dim, vocab_size)
params.b2 = torch.Tensor(vocab_size)
```

1

Torch-Autograd

```
– define objective function
function f(params, x, y, ..)
   – compute the loss

   return loss
end
```

2

```
– use torch autograd
grad = require ‘autograd’
– build computational graph and
– get derivative
df = grad(f)
```

3

```
– training procedure
for x,y in pairs(data) do
– get gradient w.r.t params
 gradient, loss = df(params, x, y)

– do SGD once you have gradient
  for k,w in pairs(params) do
    w:add(-lr, gradient[k])
  end
end
```

4

N-gram Neural Language Model

Warming up with Torch

https://github.com/ketranm/SMT-Hackathon

1. Read data

Input: PTB

Output:

- X: Tensor of history
- Y: Next word

2. Build model

Define n-gram language model:

1. What are parameters?
2. How the loss is computed?

Hint:

```
– quickly access lookup table
local util = require ‘autograd.util’

– define lookup table
params = {W = torch.Tensor(vocab_size, emb_dim)}

– x is a matrix of word ids
– reform looking up
local emb_x = util.lookup(params.W, x)
```

3. Train the model with SGD