Deep Learning Teaching Kit

Lab 4 (designed for individual work)

---

1 nngraph

1. Warmup:

   (a) Write code for the following equation in nngraph. Include the code in your submission by the name `nngraph_warmup.lua`.

   $$a = tanh(W_x x + b_1)^2 \odot \sigma(W_y y + b_2)^2 + z \tag{1}$$

   a is a vector of size (2,). x is a vector of size (4,). y is a vector of size (5,). You can infer rest of the sizes yourselves. $\odot$ means element-wise multiplication.

   (b) Choose some values for x, y and z and include a function that prints the values after forward propagating and backward propagating through this network. Use a vector of ones as the `gradOutput`. Include this in `nngraph_warmup.lua`.

2. Draw a diagram of the following `grucell` function showing all the nngraph modules.

```
function grucell(input, prevh)
local i2h    = nn.Linear(nhid, 3 * nhid)(input)
local h2h   = nn.Linear(nhid, 3 * nhid)(prevh)
local gates = nn.CAddTable()({
     nn.Narrow(2, 1, 2 * nhid)(i2h),
     nn.Narrow(2, 1, 2 * nhid)(h2h),
})
gates = nn.SplitTable(2)(nn.Reshape(2, nhid)(gates))
local resetgate     = nn.Sigmoid()(nn.SelectTable(1)(gates))
local updategate = nn.Sigmoid()(nn.SelectTable(2)(gates))
local output = nn.Tanh()(nn.CAddTable()({
     nn.Narrow(2, 2 * nhid+1, nhid)(i2h),
     nn.CMulTable()({resetgate,
          nn.Narrow(2, 2 * nhid+1, nhid)(h2h),})
}))
local nexth = nn.CAddTable()({ prevh,
     nn.CMulTable()({ updategate,
          nn.CSubTable()({output, prevh,}),}),
})
return nexth
end
```

2 Language modeling

Start with the LSTM `sample_code` from `Lab4` in the Bitbucket repo: `https://bitbucket.org/junbo_jake_zhao/deeplearningkit/src/master/Lab4/sample_code/`. It has the preprocessed data included and fully working so you can just run `main.lua`. As you can see, the preprocessing replaced infrequent words by `<unkn>`, and inserted an `<eos>tag` for the end of a sentence. The error metric you optimize is perplexity, a standard language modeling metric.

2.1 Generating sequences

Implement a lua file `query_sentences.lua` that reads from stdin a number of words, and the network continues the sentence. Example of the file output on the default network run for 1 epoch:

```
Query: len word1 word2 etc
5 the committee decided to
the committee decided to have proven for the supreme
Query: len word1 word2 etc
10 the president
the president needs to provide precious <unk> plants from real kong oil
```

Some hints:

1. Construct an inverse mapping in data.lua and add it to the return value of the script
2. You will need to add an output node to the gModule to retrieve the predictions. How do you deal with the extra output in the backward pass?
3. Look at `run_test()` for inspiration
4. Look at `communication_loop.lua` for an example of how to do the stdin/stdout.
5. At each timestep you can take the most probable word from your predictions, but because of the bias in word count in the corpus that will just become something like <unkn>of the <unkn><unkn>and <unkn>.... You can make things more interesting by using `torch.multinomial()` to sample from the predicted distribution instead.

2.2 Suggested improvements to your model

Your next task is to improve upon the base model. Here are some ideas on how you can improve over this baseline.

1. Tune hyperparameters
2. Change gradient clipping
3. Change optimization method
4. GRUs [1] (This is mandatory)
5. SCRNNs [2]

When you implement GRUs or SCRNNs, use a gradient checker to check that your module works. Also, monitor your gradients. They may explode/vanish.

2.3 Write-up

Your submission should contain the following details about your approach to the RNN language model:

1. description of the architecture (number and type of layers, size of input, etc.)
2. description of the learning techniques applied (used dropout?, etc.)
3. description of the training procedure (learning rate, momentum, error metrics used, train/validation split, training/validation/test error)

2.4 Evaluation

1. 20% - Test set performance - at least beat the baseline (perplexity 150). To evaluate this, we ask you to create a script `result.lua` that runs your model over the test set and reports the final perplexity. This perplexity value should match the value reported in your write-up.
2. 20% - Answers to the questions of section 1.
3. 30% - Write-up on your model, specifically what ideas and model architectures you tried beyond the one provided. Max three page paper plus one extra page for references. Please use a NIPS/CVPR template (LaTeX). You can choose to include the answers to the questions as a normal chapter or as an appendix.
4. 30% - Simple, readable, commented, working code (result.lua, query_sentences.lua, nngraph_handin.lua).

Submission

Send your submission (write-up, `result.lua`, `nngraph_warmup.lua` and `query_sentences.lua`) to your corresponding TA by the deadline. Include a link to the trained model file in the email. Please use the following title for your email:

[*CourseName* YOUR_NAME] Submission Lab4

References

[1] Cho, Kyunghyun, et al. "Learning phrase representations using RNN encoder-decoder for statistical machine translation." arXiv preprint arXiv:1406.1078 (2014).

[2] Mikolov, Tomas, et al. "Learning longer memory in recurrent neural networks." arXiv preprint arXiv:1412.7753 (2014).