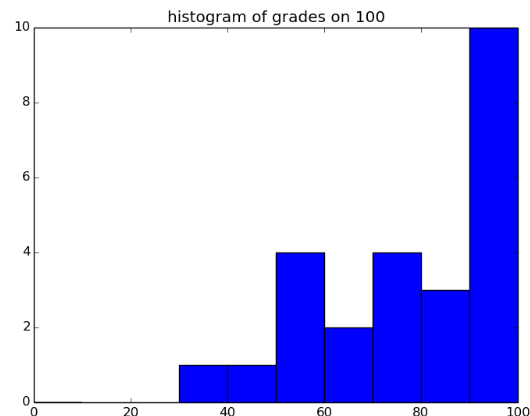
Solutions for assignment 4 Deep Learning 2015, Spring

Overview

Here are the class' test set perplexities, sorted:

181.6 / 193.0 / 195.9 / 206.3 / 207.4 / 216.9 / 228.9 / 263.0 / 263.6 / 294.6 / 294.6 / 297.5 / 298.6 / 299.2 / 304.1 / 329.0 / 331.0 / 348.6 / 437.1 / 478.9 / 540.6 / 563.4 / 609.4 / 1042.3 / 2922.7 / 4043.5 / 4951073.3 / 1170522979.7 / 3259496191.7 / 1509526042623.5 / 232326228518835879936.0



Here is the grade histogram (out of 100)

Answers to the questions

Q1

I want to see you check the result with a manual calculation with torch primitives or simple numbers!

```
require 'nngraph'

x1 = nn.Identity()()
x2 = nn.Identity()()
lx3 = nn.Linear(1,3)()
lx3.data.module.bias:fill(0) -- easier for manual evaluation
W = lx3.data.module.weight
x2x3 = nn.CMulTable()({x2, lx3})
z = nn.CAddTable()({x1, x2x3})
m = nn.gModule({x1, x2, lx3}, {z})

inputs={torch.rand(3), torch.rand(3), torch.rand(1)}
print(inputs[1]+torch.cmul(inputs[2],torch.mv(W,inputs[3])))
print(m:forward(inputs))
```

Q2

For lstm layer l at time t:

- `i` is the hidden state (output) of the previous layer, i.e. h_t^{l-1}, on which dropout is applied.
- `prev_c` is c_{t-1}^l, the LSTM cell state (or cell memory) at time $t-1$.
- `prev_h` is h_{t-1}^l, the LSTM cell output (or hidden state) at time $t-1$.

Q3

`create_network()` returns the stack of LSTM cells. It is not unrolled (unrolled = unfolded); the unrolling happens with `g_clone_many_times` to make copies of this network for every timestep. It takes as input the current word, the target word, the previous network state, and outputs the nll and current network state (containing both c and h).

Q4

- `model.s` is the unrolled model state: it contains both c and h, for each layer in the network, for each timestep in the unrolled sequence. Look at `create_network()` how `next_s` is constructed.
- `model.ds` is the (approximate bc truncated) gradient of the objective function wrt the hidden states.
- `model.start_s` is the state that gets carried over from the end of one sequence to the start of the next. It is only reset once we cycled through the whole training data.

Q5

Gradient clipping: if the L_2 norm of the gradient exceeds `params.max_grad_norm`, the gradient is rescaled to have L_2 norm of `params.max_grad_norm`.

Note that different versions of gradient clipping are described and used across different papers and published code. Christian asked Tomas Mikolov about this and he says: you can just try different things.

I asked Wojciech about this and he says he used the L2 rescaling after the full batch computation, because it's simpler to implement and he likes to keep his code as simple as possible. It is worth trying different methods of clipping and see if it improves performance.

Q6

Simple minibatch SGD, no momentum, learning rate decay when epoch more than `params.max_epoch`.

Note that the approximate gradients are calculated with BPTT, but BPTT is not an optimization method.

Q7

Add pred as an output node to the gmodule. In the backward pass, we do not want to change the gradient. Therefore we add a zero-matrix for the corresponding `gradOutput`.

Relevant parts of my code:

```
  local module           = nn.gModule({x, y, prev_s},
                                      {err, nn.Identity()(next_s), pred})
...
-- define bp_dy outside of bp(), only once.
```

```
bp_dy  = transfer_data(torch.zeros(params.batch_size,params.vocab_size))
...
    local tmp = model.rnns[i]:backward({x, y, s},
                                       {bp_derr, model.ds, bp_dy})[3]
```

Test set performance

Here are the class' test set perplexities, sorted:

181.6 / 193.0 / 195.9 / 206.3 / 207.4 / 216.9 / 228.9 / 263.0 / 263.6 / 294.6 / 294.6 / 297.5 / 298.6 / 299.2 / 304.1 / 329.0 / 331.0 / 348.6 / 437.1 / 478.9 / 540.6 / 563.4 / 609.4 / 1042.3 / 2922.7 / 4043.5 / 4951073.3 / 1170522979.7 / 3259496191.7 / 1509526042623.5 / 232326228518835879936.0

```
FYI the model from the starter code with sequence_length=50 (baseline) I got perplexity:
   - on the validation set: 314.2
   - on the test set:       264.6
I did notice serious variation from different random seeding, so
I take test set perplexity 280 as baseline.
FYI Graves reports 122 to 167 perplexity for his char-level ptb models. Some people got pretty close.
The test set perplexity performance translates to a score on 25 as follows:
+ below 280: 25/25
+ higher than 280: max(10, 25 * (1 - (test_perplexity-280) / (1000-280)))
+ If it didn't run: 0/25.
```

Code

Here are some code aspects I looked at, and general remarks about what I saw:

- Correct implementation of pred & gradOutput. - Don't reallocate & transfer the zero-matrix for pred gradOutput on every backward pass! I didn't deduct points for this though.
- Interactive `query_sentences` and evaluation loop
  - No need to keep working with tensors of size batch_size in test loop
  - Dummy y value in the loss for query_sentences and char_predictions, has no influence or meaning
  - Disable dropout

Write-up

```
+ In general I look for effort above all.
+ Argumentation for trying different things.
+ Having read and understood the relevant parts of the two cited papers or preferably more.
+ Many people just try to make the network deeper or wider, or just adding 0.50 dropout,
  without changing the other parameters like number of epochs, seq\_length, weight initialization.
  When increasing the model capacity, it would have been smart to look at the big model that is
  commented out and try either that, OR a middle ground between the 1h (baseline) model and 1day model
+ If you didn't properly desribe what is your final model, and what other models you tried, and
  the train \& val perplexity, I deducted points.
+ I posted properly named write-ups in http://cims.nyu.edu/~ts2387/writeups
```