1/ RNN Layers

RNN Cell Forward: Previous hidden state h_{t-1}, current input x_t, then

$a_t = W_h * h_t + W_x * x_t + b$

$h_t = \tanh(a_t)$

Where: W_h: hidden-hidden weight matrix

W_x: input-hidden weight matrix

RNN Cell Backward: Let the gradient of the loss to variable x as $\mathbf{d}_x$. Then:

$d_{a(t)} = d_{h(t)}\ (1 - h_t ** 2)$

$d_{h(t-1)} = d_{a(t)} W_h^T$

$d_{x(t)} = d_{a(t)} W_x^T$

$d_{W(h)} = h_t^T d_{a(t)}$

$d_{W(x)} = x_t^T d_{a(t)}$

d_b = sum($d_{a(t)}$, axis along batch)

Note: To deal with NaN data, for any element inside the gradients, transform them to 0 if they are NaN.

RNN Forward:

First, append all training sequences with NaN such that they all have equal length.

Then, iterate through the whole sequence with RNN Cell. Finally output the last hidden state result.

RNN Backward:

Run the Backward of RNN Cell over the entire sequence through all timesteps. The gradients with respect to inputs and hidden states at each time step is calculated. **Note:** The gradients with respect to W_x, W_h, b is the sum of the gradients over all time steps.

Bidirectional RNN Forward:

Just run RNN forward in 2 opposite directions. Use reverse_temporal_data to get the reverse inputs for RNN to run in opposite direction.

Bidirectional RNN Backward:

Run backward on forward-RNN => out_grads_1
Run backward on backward-RNN => out_grads_2

out_grads = out_grads_1 + reverse_temporal_data(out_grads_2)

2/ ARCHITECTURE:

After manually fine-tuning other parameters, I fine tune the learning rate(from 0.0001 to 0.01), decay rate(from 0.0001 to 0.001) and training batch size(from 16,32,64,128) with Random Search on the validation dataset. Here is the architecture(Read more on *run.py* and *applications.py*)

FCLayer(vocab_size, 300)
RELU

Bidirectional-RNN(in_features=300, units=100)
Dropout(ratio=0.5)

FCLayer(200, 32)
RELU

Temporal Pooling
Dropout(0.5)

FCLayer(32, 2)

Optimizer: Adam(lr = 0.00485, decay = 0.000121, scheduler_func = cyclical_reboost).

Some other statistics:

+ training_batch: 128
+ epochs: 60

3/ INSIGHT:

- If the batch size is too small, it is possible for a large fluctuations in the training loss. A batch size of 32 or smaller will result in tremendously large fluctuations. Finally use a batch size of 128, and have to increase number of epoches to 60.

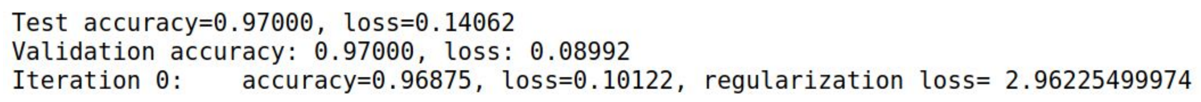
- Use Bi-directional RNN instead of RNN because I believe that the sentiment of a word depends on not just the words before it, but all words of the paragraphs.

- I notice that there is a big gap between the testing accuracy and training accuracy, so it was a good idea to add in 2 DropOut Layer with drop_ratio of 0.5.

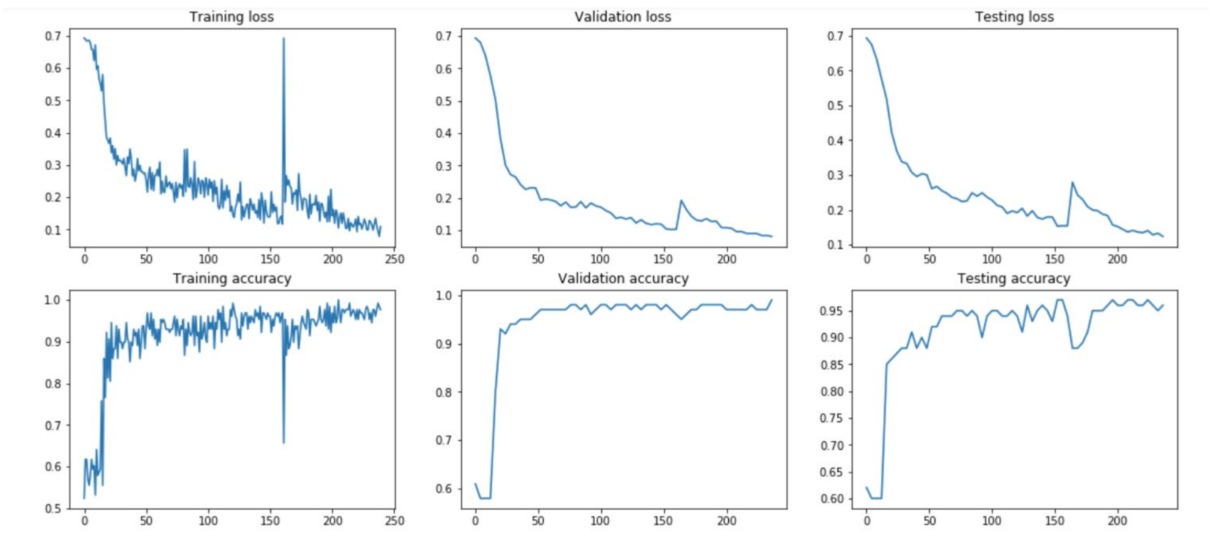
- In later iterations when the training loss has decreased a lot, the loss now becomes very unbalanced if we still maintain a large learning rate => I need to use a decay constant here. However, after the learning rate is so small for very long time, the loss becomes stuck in local minimum => Design a scheduler function that will reboost the learning rate to the original value 10 iterations(call it a cycle), and the cycle time is doubled after every re-boost.

RESULT:

Final test accuracy is 0.97000



```
Test accuracy=0.97000, loss=0.14062
Validation accuracy: 0.97000, loss: 0.08992
Iteration 0:    accuracy=0.96875, loss=0.10122, regularization loss= 2.96225499974
```


Training loss
Validation loss
Testing loss
Training accuracy
Validation accuracy
Testing accuracy