b01502069 顏毓均

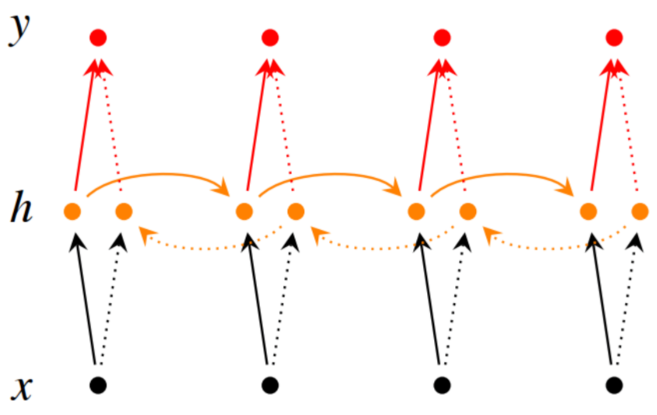
ADL2016 Hw3 Report

Model

under multi_task_model.py

Bidirectional RNN

Architecture



The principle of BRNN is to split the neurons of a regular RNN into two directions, one for positive time direction(forward states), and another for negative time direction(backward states). Those two states' output are not connected to inputs of the opposite direction states. The general structure of RNN and BRNN can be depicted in the right diagram. By using two time directions, input information from the past and future of the current time frame can be used unlike standard RNN which requires the delays for including future information.[1]

Training

BRNN can be trained using similar algorithms compared to RNN, because the two directional neurons do not have any interactions. However, when back-propagation is applied, additional processes are needed because updating input and output layers cannot be done at once. General procedures for training are as follows: For forward pass, forward states and backward states are passed first, then output neurons are passed. For backward pass, output neurons are passed first, then forward states and backward states are passed next. After forward and backward passes are done, the weights are updated.

With dropout rate = 0.5

Learn and Improve

Using GPU in Tensorflow

To use specific devices:

```
with tf.device('/cpu:0'):
```

To see which parameter stay on where:

```
log_device_placement=True
```

Enable dynamic GPU memory grow:

```
config.gpu_options.allow_growth = True
```

To improve

Enlarge **maximun step** and **Embedding size** can help.

Code

To create or reload model

```
ckpt = tf.train.get_checkpoint_state(FLAGS.train_dir)
  if ckpt and tf.gfile.Exists(ckpt.model_checkpoint_path):
    print("Reading model parameters from %s" %
ckpt.model_checkpoint_path)
    model_train.saver.restore(session, ckpt.model_checkpoint_path)
  else:
    print("Created model with fresh parameters.")
    session.run(tf.initialize_all_variables())
  return model_train, model_test
```

Bidirectional or not

```
params = tf.trainable_variables()
    if not forward_only:
      opt = tf.train.AdamOptimizer()
      if task['joint'] == 1:
        # backpropagate the intent and tagging loss, one may further
adjust
        # the weights for the two costs.
        gradients = tf.gradients([self.tagging_loss,
self.classification_loss], params)
      elif task['tagging'] == 1:
        gradients = tf.gradients(self.tagging_loss, params)
      elif task['intent'] == 1:
        gradients = tf.gradients(self.classification_loss, params)
      clipped_gradients, norm = tf.clip_by_global_norm(gradients,
                                                       max_gradient_norm)
      self.gradient_norm = norm
      self.update = opt.apply_gradients(
          zip(clipped_gradients, params), global_step=self.global_step)
    self.saver = tf.train.Saver(tf.all_variables())
```

Comment and reference

https://www.tensorflow.org/versions/r0.12/how_tos/using_gpu/index.html
https://en.wikipedia.org/wiki/Bidirectional_recurrent_neural_networks
https://github.com/MarkYan/ADL2016/tree/master/hw3