Deep Learning Teaching Kit
Lab 4, GRU Sample Solution

```
local function gru(x, prev_c, prev_h)

   local i2h = nn.Linear(params.rnn_size, 3*params.rnn_size)(x)
   local h2h = nn.Linear(params.rnn_size, 3*params.rnn_size)(prev_h)
   local gates = nn.CAddTable()({
           nn.Narrow(2, 1, 2 * params.rnn_size)(i2h),
           nn.Narrow(2, 1, 2 * params.rnn_size)(h2h),
    })

   gates = nn.SplitTable(2)(nn.Reshape(2, params.rnn_size)(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 * params.rnn_size+1, params.rnn_size)(i2h),
           nn.CMulTable()({resetgate,
           nn.Narrow(2, 2 * params.rnn_size+1, params.rnn_size)(h2h),})
   }))

   local next_h = nn.CAddTable()({prev_h,
           nn.CMulTable()({updategate,
           nn.CSubTable()({output, prev_h,}),}),
   })

   local next_c = prev_c
   return next_c, next_h

end
```

To minimize the modification in other part of codes, "prev_c" and "next_c" are kept in the gru function although they are not used. The gru function is called in the create_network function.

```
function create_network()
   local x                = nn.Identity()()
   local y                = nn.Identity()()
   local prev_s           = nn.Identity()()
   local i                = {[0] = nn.LookupTable(params.vocab_size,
      params.rnn_size)(x)}
   local next_s           = {}
   local split            = {prev_s:split(2 * params.layers)} -- c1,h1,c2,h2
   for layer_idx = 1, params.layers do
      local prev_c       = split[2 * layer_idx - 1]
      local prev_h       = split[2 * layer_idx]
      local dropped      = nn.Dropout(params.dropout)(i[layer_idx - 1])
      local next_c, next_h = nil, nil
      if params.encoder == 'lstm' then
         next_c, next_h = lstm(dropped, prev_c, prev_h)
```

```
    elseif params.encoder == 'gru' then
      next_c, next_h = gru(dropped, prev_c, prev_h)
    end
    table.insert(next_s, next_c)
    table.insert(next_s, next_h)
    i[layer_idx] = next_h
  end
  local h2y             = nn.Linear(params.rnn_size, params.vocab_size)
  local dropped         = nn.Dropout(params.dropout)(i[params.layers])
  local pred            = nn.LogSoftMax()(h2y(dropped))
  local err             = nn.ClassNLLCriterion()({pred, y})
  local module          = nn.gModule({x, y, prev_s},
  {err, nn.Identity()(next_s), pred})
  -- initialize weights
  module:getParameters():uniform(-params.init_weight, params.init_weight)
  return transfer_data(module)
end
```