A BEGINNER'S TORCH7 TUTORIAL

BAOGUANG SHI (石葆光)

HUAZHONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

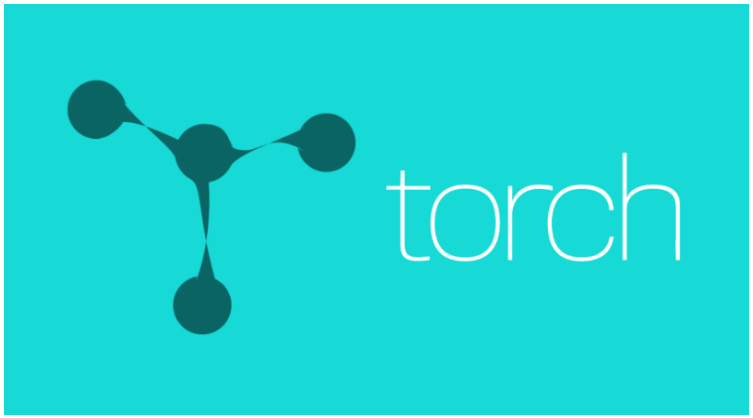
VALSE 2016, APRIL 22RD, 2016

WUHAN, CHINA

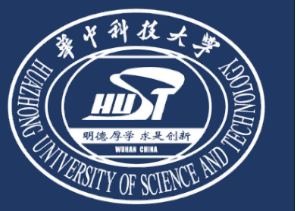
Part-1 Basics

What is Torch7?

- A machine learning programming framework built on Lua(JIT), C++, and CUDA
- Open-source, BSD license
- Nowadays widely used for deep learning
  - Google DeepMind ("Human-level control through deep reinforcement learning", Nature, 2015)
  - Facebook AI Research



A Quick Look

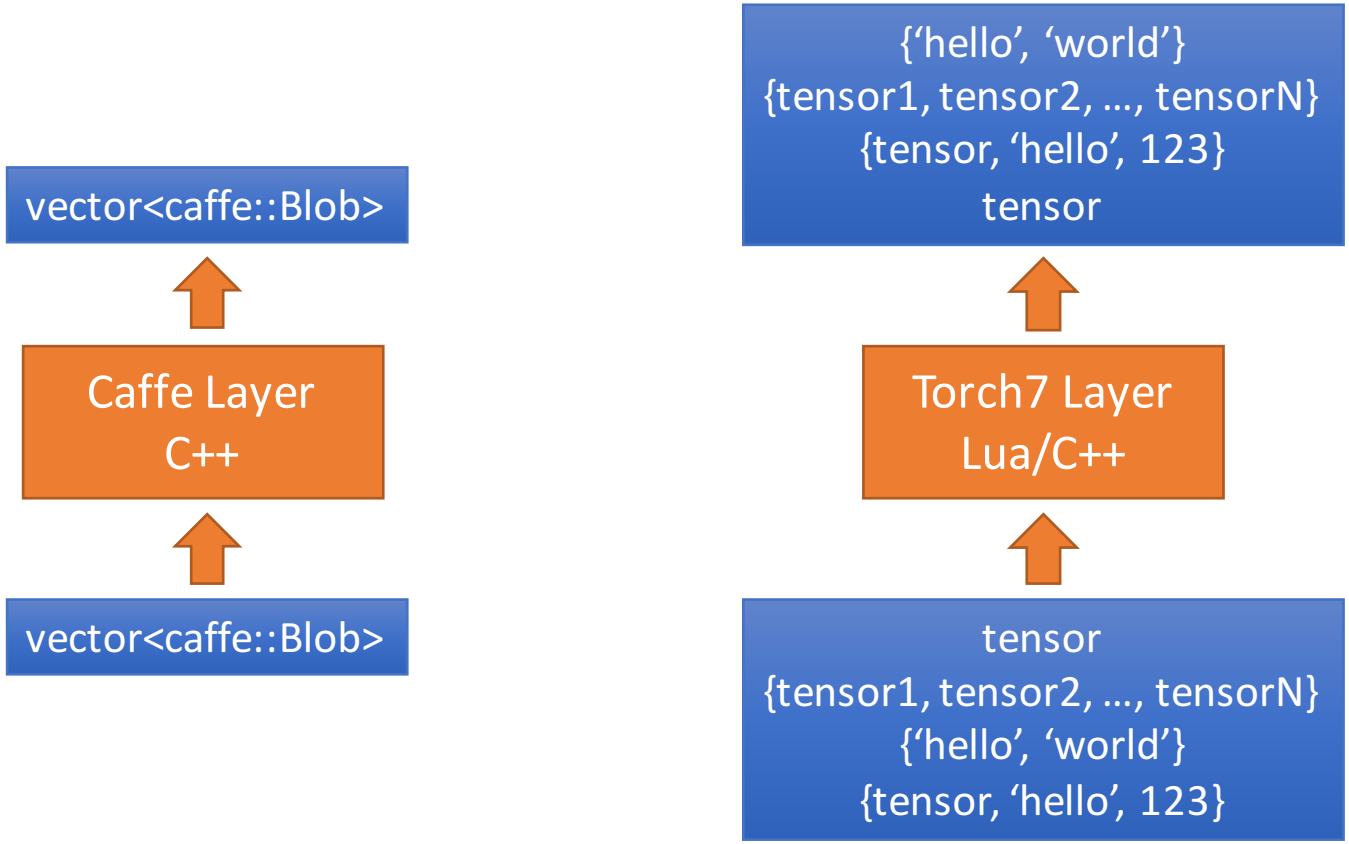


- Construct a multi-layered perceptron model, and perform forward propagation

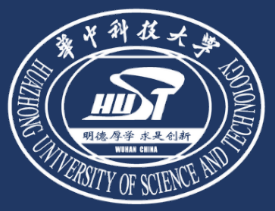
```
require('nn')

-- create an MLP model
model = nn.Sequential()
model:add(nn.Linear(512, 256))
model:add(nn.ReLU())
model:add(nn.Linear(256, 10))
model:add(nn.SoftMax())

-- generate a random input
input = torch.Tensor(10, 512):uniform(-1,1)

-- forward propagation
output = model:forward(input)
```

Why Use Torch7?

- Great flexibility



Why Use Torch7?



- Torch7 is fast
  - LuaJIT is blazingly fast
  - CUDA/C++ accelerated backends

	Fortran gcc 5.1.1	Julia 0.4.0	Python 3.4.3	R 3.2.2	Matlab R2015b	Octave 4.0.0	Mathematica 10.2.0	JavaScript V8 3.28.71.19	Go go1.5	LuaJIT gsl-shell 2.3.1	Java 1.8.0_45
fib	0.70	2.11	77.76	533.52	26.89	9324.35	118.53	3.36	1.86	1.71	1.21
parse_int	5.05	1.45	17.02	45.73	802.52	9581.44	15.02	6.06	1.20	5.77	3.35
quicksort	1.31	1.15	32.89	264.54	4.92	1866.01	43.23	2.70	1.29	2.03	2.60
mandel	0.81	0.79	15.32	53.16	7.58	451.81	5.13	0.66	1.11	0.67	1.35
pi_sum	1.00	1.00	21.99	9.56	1.00	299.31	1.69	1.01	1.00	1.00	1.00
rand_mat_stat	1.45	1.66	17.93	14.56	14.52	30.93	5.95	2.30	2.96	3.27	3.92
rand_mat_mul	3.48	1.02	1.14	1.57	1.12	1.12	1.30	15.07	1.42	1.16	2.36

Figure: benchmark times relative to C (smaller is better, C performance = 1.0).

**image from http://julialang.org/*

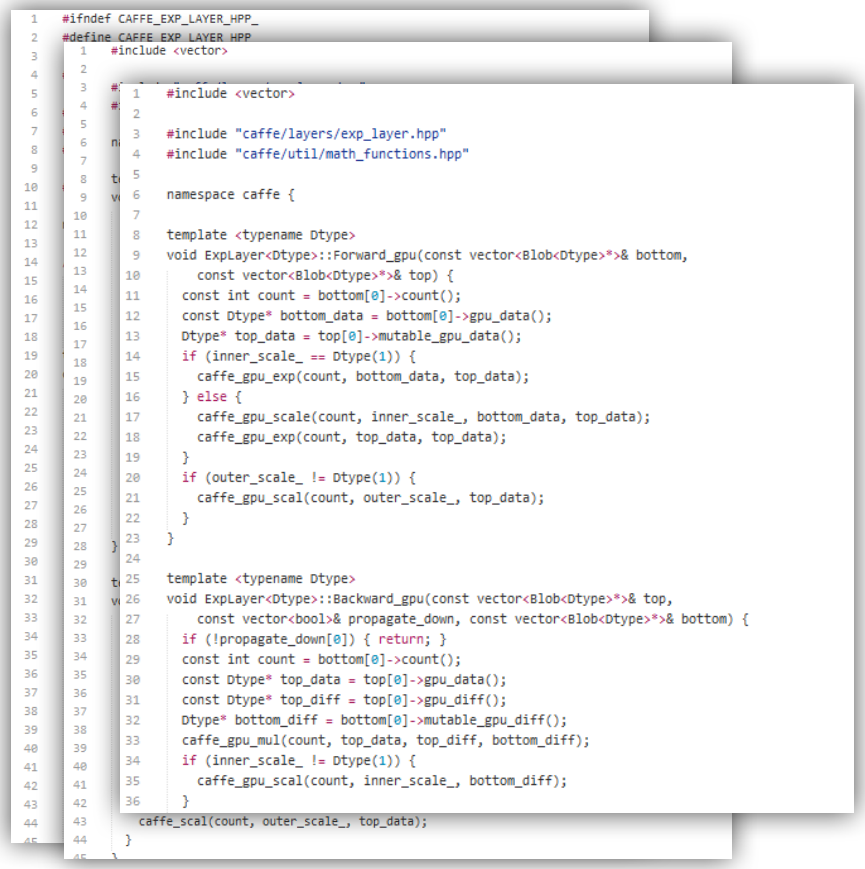
Why Use Torch7?

- Write less code

Torch7 (1 file)

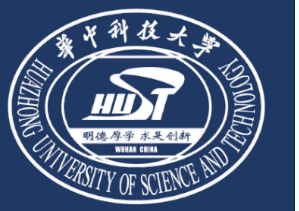
```lua
local Exp = torch.class('nn.Exp', 'nn.Module')

function Exp:updateOutput(input)
  return self.output:exp(input)
end

function Exp:updateGradInput(input, gradOutput)
  return self.gradInput:cmul(self.output, gradOutput)
end
```

Caffe (3 files)



```
#ifndef CAFFE_EXP_LAYER_HPP_
#define CAFFE_EXP_LAYER_HPP
#include <vector>
#include <vector>

#include "caffe/layers/exp_layer.hpp"
#include "caffe/util/math_functions.hpp"

namespace caffe {

template <typename Dtype>
void ExpLayer<Dtype>::Forward_gpu(const vector<Blob<Dtype>*>& bottom,
    const vector<Blob<Dtype>*>& top) {
  const int count = bottom[0]->count();
  const Dtype* bottom_data = bottom[0]->gpu_data();
  Dtype* top_data = top[0]->mutable_gpu_data();
  if (inner_scale_ == Dtype(1)) {
    caffe_gpu_exp(count, bottom_data, top_data);
  } else {
    caffe_gpu_scale(count, inner_scale_, bottom_data, top_data);
    caffe_gpu_exp(count, top_data, top_data);
  }
  if (outer_scale_ != Dtype(1)) {
    caffe_gpu_scal(count, outer_scale_, top_data);
  }
}

template <typename Dtype>
void ExpLayer<Dtype>::Backward_gpu(const vector<Blob<Dtype>*>& top,
    const vector<bool>& propagate_down, const vector<Blob<Dtype>*>& bottom) {
  if (!propagate_down[0]) { return; }
  const int count = bottom[0]->count();
  const Dtype* top_data = top[0]->gpu_data();
  const Dtype* top_diff = top[0]->gpu_diff();
  Dtype* bottom_diff = bottom[0]->mutable_gpu_diff();
  caffe_gpu_mul(count, top_data, top_diff, bottom_diff);
  if (inner_scale_ != Dtype(1)) {
    caffe_gpu_scal(count, inner_scale_, bottom_diff);
  }
  caffe_scal(count, outer_scale_, top_data);
}
```

Getting Started



- Prerequisites:
  - Linux (Ubuntu 14.04 x64 recommended)
  - (optional) NVIDIA GPU, CUDA

- Install Torch7

```
git clone https://github.com/torch/distro.git ~/torch --recursive
cd ~/torch; bash install-deps;
./install.sh
```

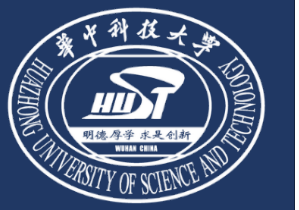
- Install packages

```
luarocks install image
luarocks install cunn
```

- Run Torch7

```
th main.lua
```

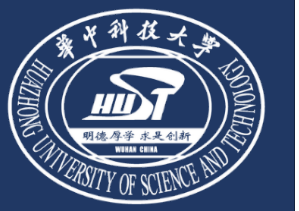
Lua Basics



- Lua is an interpretive language, like Python and MATLAB

```
-- this is a comment
num = 42
s = 'hello'

for i = 1, 10 do
  if i % 2 == 0 then
    num = num + 1
  else
    num = num - 2
  end
end

function printNumber(n)
  print(string.format('The number is %d', n))
end

printNumber(num)
```

Lua Basics



- Tables: the *only* compound data structure in Lua
- Used as dictionary

```
person = {id=1234, name='Jane Doe', registered=true}
print(person.id) -- output 1234
person.age = 32 -- add a new key 'age'
```

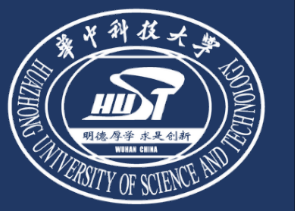
- Used as list

```
v = {'jane', 36, 'doe'}
print(v[1]) -- output 'jane'
print(#v) -- output 3
v[4] = 32 -- add a new element
```

- Used as class container (meta-table)

- Recommended tutorial: "Learn Lua in 15 mins"
  http://tylerneylon.com/a/learn-lua/

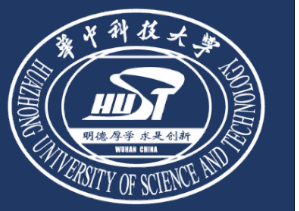
Torch7 Core Structure: Tensor



- Tensor: multi-dimensional array
- Similar to `np.array` in Numpy and matrix in MATLAB
- Use tensor

```
x = torch.Tensor(128,3,224,224)
```

Torch7 Core Structure: Tensor

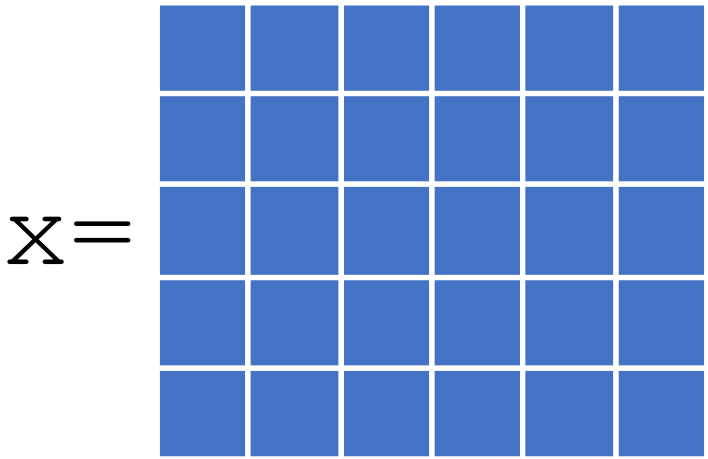


- Data types
  - Byte: `torch.ByteTensor`
  - Integer: `torch.IntTenosr`
  - Long: `torch.LongTensor`
  - Float: `torch.FloatTensor`
  - Double: `torch.DoubleTensor`
  - GPU float: `torch.CudaTensor`
- Default tensor: `torch.Tensor`, set by `torch.setdefaulttensortype('...')`
- Type casting: `x:type('torch.FloatTensor')`

Tensor Operations

- Create & initialize a tensor

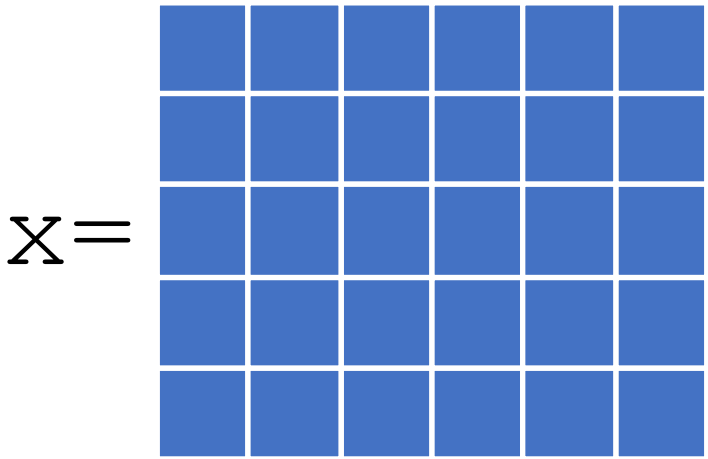
```
x = torch.Tensor(5,6)
x:fill(0) -- fill with zeros
x:uniform(-1,1) -- fill with random values between -1 and 1
```



Tensor Operations

- Number of dimensions, total number of elements, size of each dimension

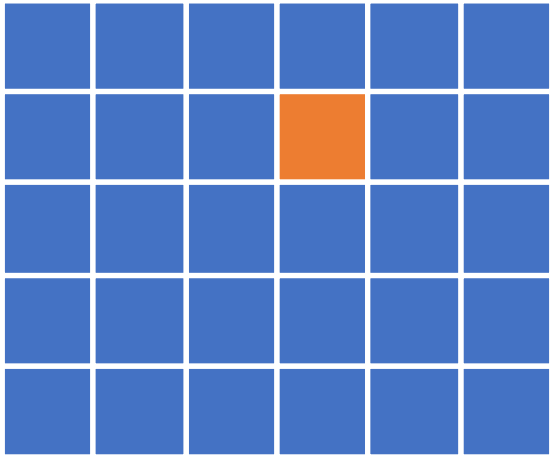
```
x:nDimension() -- return 2
x:nElements() -- returns 30
x:size() -- returns torch.LongTensor({5,6})
x:size(2) -- returns 6
```



Tensor Operations

- Access an element

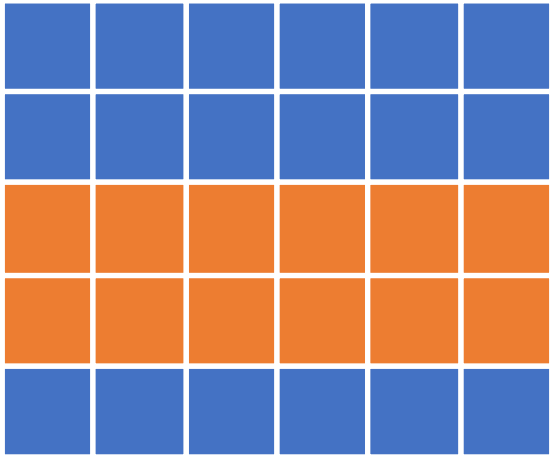
```
y = x[2][4]
```



Tensor Operations

- Slice sub-tensor (does not create a new tensor)

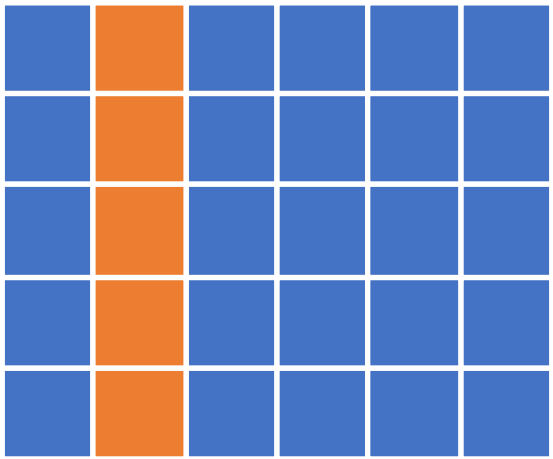
```
y = x:narrow(1, 3, 2)
```



Tensor Operations

- Slice sub-tensor (does not create a new tensor)

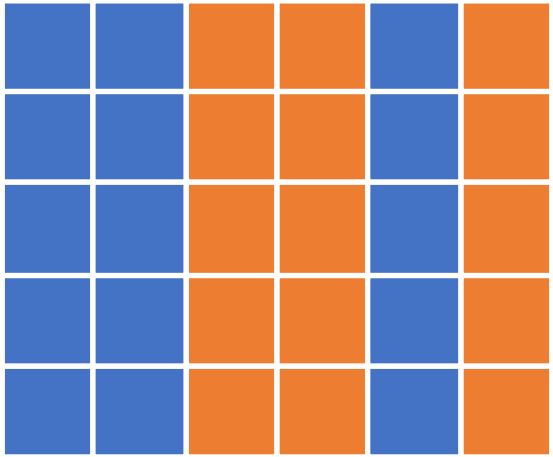
```
y = x:narrow(2, 2, 1)
```



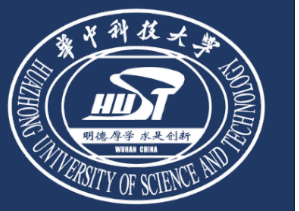
Tensor Operations

- Pick sub-tensor by indices

```
y = x:index(2, torch.LongTensor{3,4,6})
```



Object-Oriented Programming

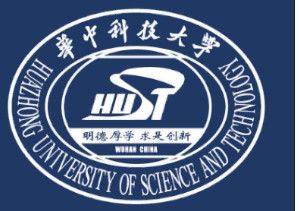


- Torch7 provides **`torch.class`** for OOP

```
local Rectangle, parent = torch.class('Rectnagle', 'Polygon')

function Rectangle:__init(w, h)
  parent.__init(self)
  self.w = w
  self.h = h
end

function Rectangle:area() -- override
  return self.w * self.h
end

function Rectangle:numVertices()
  return 4
end
```

Part-2 Building Neural Networks

The nn Package



- Construct neural networks on CPU/GPU

```
require('nn')
require('cunn')
```

- Convert a CPU model to GPU

```
model:cuda()
```

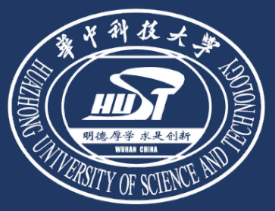
- Most neural nets layers inherits the base class: `nn.Module`, whose base methods are:

```
Module:updateOutput(input)
Module:updateGradInput(input, gradOutput)
Module:accGradParameters(input, gradOutput, scale)
```

- Also implemented criterions (loss functions), which inherit `nn.Criterion`

```
Criterion:updateOutput(input, target)
Criterion:updateGradInput(input, target)
```

The nn Package



- Implemented about 140 kinds of network layers/criterion (April 2016)

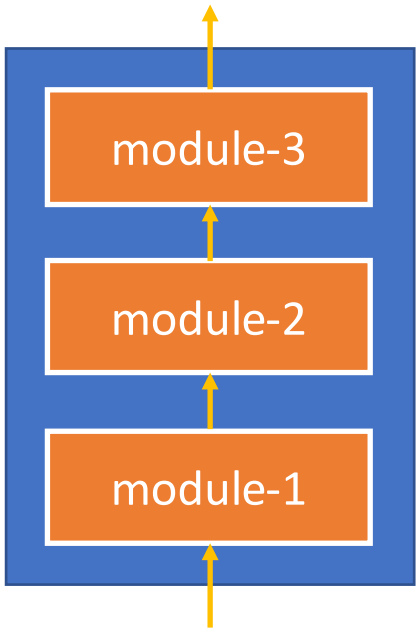
- Layers
  - Convolution: `nn.SpatialConvolution`, `nn.SpatialFullConvolution`
  - Activation functions: `nn.ReLU`, `nn.Tanh`, `nn.Sigmoid`, `nn.PReLU`
  - Normalization: `nn.BatchNormalization`, `nn.SpatialBatchNormalization`

- Criterions
  - Classification: `nn.ClassNLLCriterion`, `nn.BCECriterion`
  - Regression: `nn.MSECriterion`, `nn.WeightedMSECriterion`
  - Multi-task: `nn.MultiCriterion`

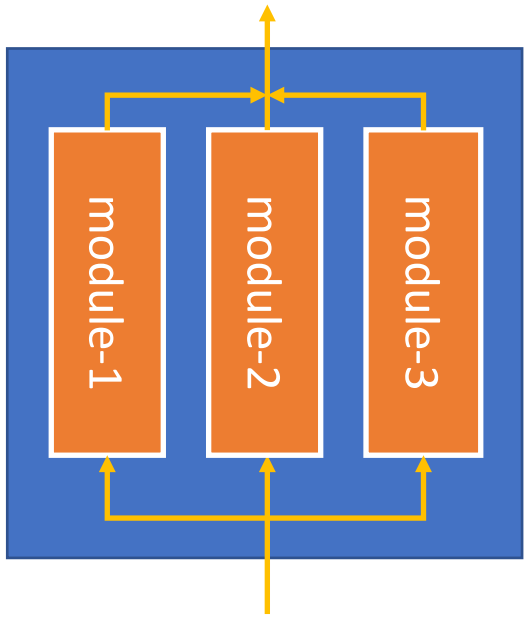
The nn Package

- Network containers

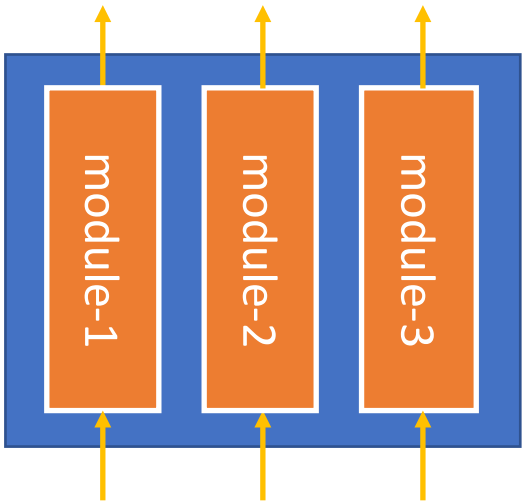
`nn.Sequential`



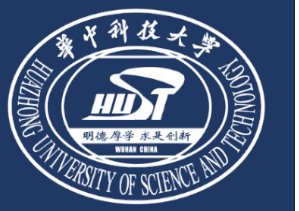
`nn.Concat` / `nn.ConcatTable`



`nn.Parallel` / `nn.ParallelTable`



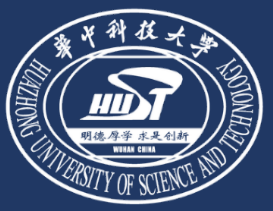
The nn Package



- Network containers

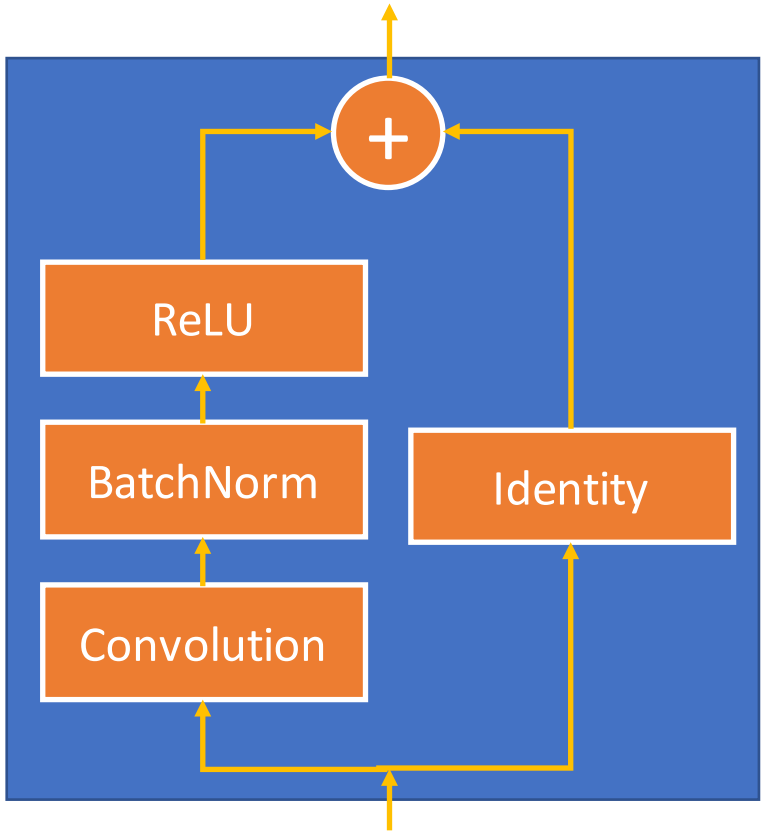
```
require('nn')

-- create an MLP model
model = nn.Sequential()
model:add(nn.Linear(512, 256))
model:add(nn.ReLU())
model:add(nn.Linear(256, 10))
model:add(nn.SoftMax())

-- generate a random input
input = torch.Tensor(10, 512):uniform(-1,1)

-- forward propagation
output = model:forward(input)
```

Example: Building a ResNet Block



- ResNet block: Sequential + ConcatTable

```
local resBlock = nn.Sequential()

local branch = nn.ConcatTable()

local conv = nn.Sequential()
conv:add(nn.SpatialConvolution(...))
conv:add(nn.SpatialBatchNormalization(...))
conv:add(nn.ReLU())
branch:add(conv)

branch:add(nn.Identity())

resBlock:add(concat)
resBlock:add(nn.CAddTable())
```

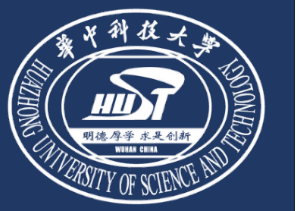


The nn Package

- Training networks (1/2)

```
-- create criterion (loss function)
local criterion = nn.ClassNLLCriterion()

-- create optimizer (SGD)
require('optim')
local optimizer = optim.sgd({learningRate=1e-3, momentum=0.9})
local optimState = {}

-- retrieve all parameters and gradients on paramters
local params, gradParams = model:getParameters()
```

The nn Package



- Training networks (2/2)

```
local function trainOneBatch(inputBatch, targetBatch)
  model:training()
  local feval = function(params)
    gradParams:zero()
    local outputBatch = model:forward(inputBatch)
    local f = criterion:forward(outputBatch, targetBatch)
    model:backward(inputBatch,
                   criterion:backward(outputBatch, targetBatch))
    return f, gradParams
  end
  optimizer(feval, params, optimState)
end

-- train loops
for i = 1, maxIterations do
  local inputBatch, targetBatch = trainDataset:getData()
  trainOneBatch(inputBatch, targetBatch)
end
```

The nngraph Package

- Construct complex graph-structured networks
  - LSTM
  - GRU
  - Attention-based models
  - ...

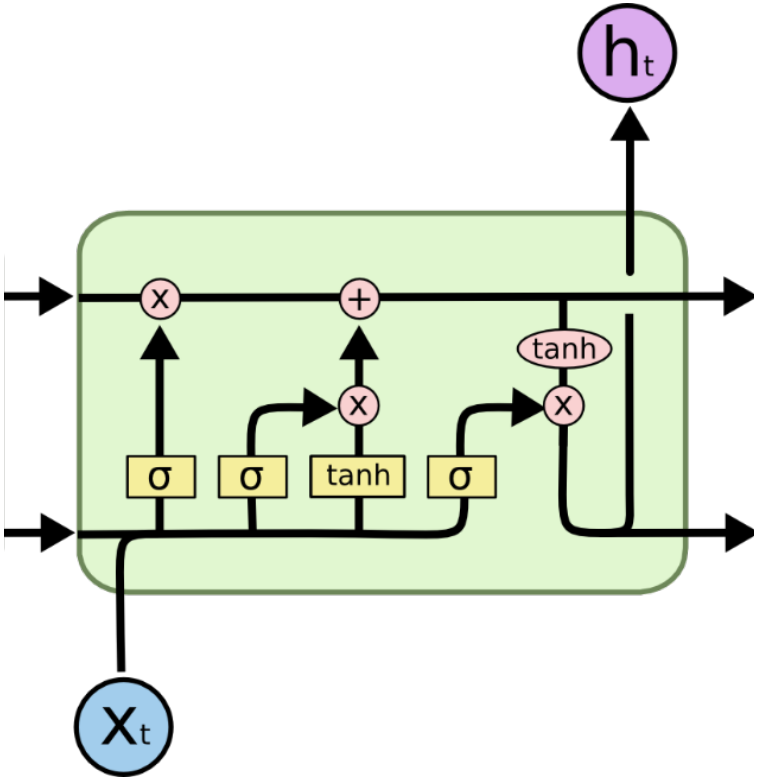
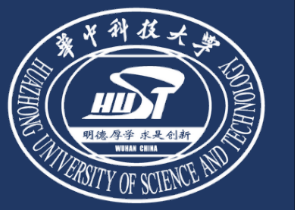


**image from http://colah.github.io/posts/2015-08-Understanding-LSTMs/*

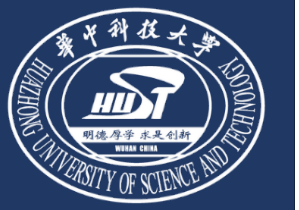
The nngraph Package



- The package nngraph provides a simple way to connect modules

```
require('nngraph')

-- x -> ReLU -> y1
y1 = nn.ReLU()(x)

-- x -> Linear -> y2
y2 = nn.Linear(4,4)(x)

-- z = y1 + y2
z = nn.CAddTable()({y1, y2})
```

The nngraph Package

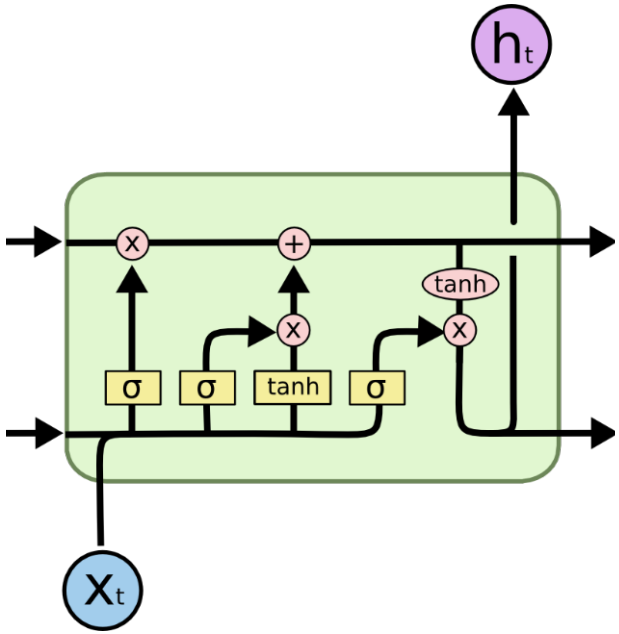


- Create a graph-structured unit

```
require('nngraph')

function makeRnnUnit(nIn, nHidden)
  -- define input interface
  local x = nn.Identity()()
  local prevH = nn.Identity()()
  local inputs = {prevH, x}

  -- define structure
  local x2h = nn.Linear(nIn, nHidden)(x)
  local h2h = nn.Linear(nHidden, nHidden)(prevH)
  local sum = nn.CAddTable()({x2h, h2h})
  local h = nn.Tanh()(sum)

  -- define output interface
  local outputs = {h}
  return nn.gModule(inputs, outputs)
end
```

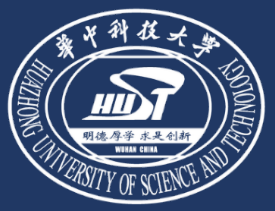
The nngraph Package

- LSTM



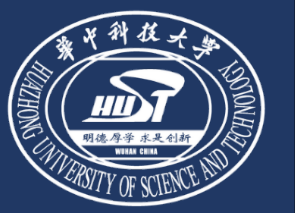

```lua
function makeLstmUnit(nIn, nHidden)
    local x, prev_c, prev_h = nn.Identity()(), nn.Identity()(), nn.Identity()()
    local inputs = {x, prev_c, prev_h}
    -- evaluate the input sums at once for efficiency
    local i2h            = nn.Linear(nIn,     4*nHidden)(x)
    local h2h            = nn.Linear(nHidden, 4*nHidden)(prev_h)
    local all_input_sums = nn.CAddTable()({i2h, h2h})
    -- decode the gates
    local sigmoid_chunk  = nn.Narrow(2, 1, 3*nHidden)(all_input_sums)
    sigmoid_chunk        = nn.Sigmoid()(sigmoid_chunk)
    local in_gate        = nn.Narrow(2,           1, nHidden)(sigmoid_chunk)
    local forget_gate    = nn.Narrow(2,   nHidden+1, nHidden)(sigmoid_chunk)
    local out_gate       = nn.Narrow(2, 2*nHidden+1, nHidden)(sigmoid_chunk)
    -- decode the write inputs
    local in_transform   = nn.Narrow(2, 3*nHidden+1, nHidden)(all_input_sums)
    in_transform         = nn.Tanh()(in_transform)
    -- perform the LSTM update
    local next_c         = nn.CAddTable()({
                               nn.CMulTable()({forget_gate, prev_c}),
                               nn.CMulTable()({in_gate    , in_transform})
                               })
    -- gated cells from the output
    local next_h         = nn.CMulTable()({out_gate, nn.Tanh()(next_c)})
    -- y (output)
    local y              = nn.Identity()(next_h)
    -- there will be 3 outputs
    local outputs = {next_c, next_h, y}
    local lstmUnit = nn.gModule(inputs, outputs)
    return lstmUnit
end
```

Create a New Network Layer



- Create a new class that inherits nn.Module, and override 3 methods:
  - `updateOutput(input)`, compute forward propagation
  - `updateGradInput(input,gradOutput)`, compute gradients on input
  - `accGradParameters(input,gradOutput,scale)`, compute gradients on parameters

Create a New Network Layer

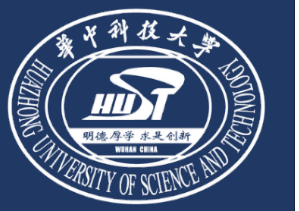


- Example: a layer that scales every input element by a trainable parameter ($y_i \rightarrow w_i x_i$)

```
1  local ElementwiseScale, parent = torch.class('nn.ElementwiseScale', 'nn.Module')
2
3  function ElementwiseScale:__init(inputSize)
4    self.weight = torch.Tensor(inputSize):fill(1.0)
5  end
6
7  function ElementwiseScale:updateOutput(input)
8    self.output = torch.cmul(input, self.weight)
9    return self.output
10 end
11
12 function ElementwiseScale:updateGradInput(input, gradOutput)
13   self.gradInput = torch.cmul(gradOutput, self.weight)
14   return self.gradInput
15 end
16
17 function ElementwiseScale:accGradParameters(input, gradOutput, scale)
18   self.gradWeight:add(torch.cmul(input, gradOutput), scale)
19 end
```

Part-3 Tips & Tricks

cuDNN acceleration

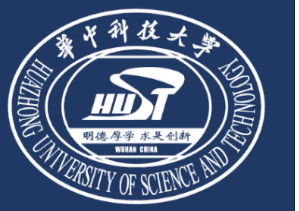


- Use cudnn to accelerate some network layers

```
>> luarocks install cudnn
require('cudnn')
```

- Replace nn layers with cudnn layers
  - `cudnn.SpatialConvolution`
  - `cudnn.ReLU`
  - `cudnn.SpatialBatchNormalization`
  - `(in beta) cudnn.RNN, cudnn.LSTM, cudnn.GRU`

Debugging in code



- Use fb.debugger (install fblualib first) for debugging Lua code

```
debugger = require('fb.debugger')

...
debugger.enter()
...
```

- After entering the debugging mode, print the value of a variable

```
>> p variable-name
```

- Continue and step

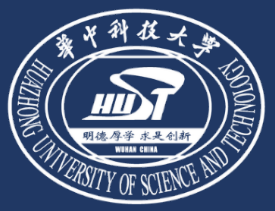
```
>> c            (continue)
>> n            (next line, skip function calls)
>> s            (step into functions)
```

Debugging in network

- Create a Probe layer to check the input/output of a network module

```
1  local Probe, parent = torch.class('nn.Probe', 'nn.Module')
2
3  function Probe:updateOutput(input)
4    print(input)
5    self.output = input
6    return self.output
7  end
8
9  function Probe:updateGradInput(input, gradOutput)
10   print(gradOutput)
11   self.gradInput = gradOutput
12   return self.gradInput
13 end
```

What's More?



- Mixed Lua - C/C++ programming
  - TH: mix Lua and C
  - THPP: mix Lua and C++
  - lua - FFI: directly calls C library functions
- Call Python Functions
  - fb.python
- Pretrained Models
  - Torch7 ModelZoo: https://github.com/torch/torch7/wiki/ModelZoo
  - Residual network: https://github.com/facebook/fb.resnet.torch

Thank You!