

Torch Tutorial

Selected Topics in
Advanced Machine Learning

Useful links

- Example codes for this tutorial: <https://github.com/mbojarski/Tutorials>
- Getting started: <http://torch.ch/docs/getting-started.html>
- Cheatsheet: <https://github.com/torch/torch7/wiki/Cheatsheet>
- Tutorials: <https://github.com/torch/tutorials>
- Math Docs: <https://github.com/torch/torch7/blob/master/doc/math.md>
- Tensor Docs: <https://github.com/torch/torch7/blob/master/doc/tensor.md>
- NN Docs: <https://github.com/torch/nn/tree/master/doc>
- Programming Lua: <https://www.lua.org/pil/contents.html#P1>

Basic Instructions

- Starting notepad: `th` or `qlua`
- Executing script: `qlua script.lua`
- Executing script in notepad: `dofile script.lua`
- Selecting GPU: `cutorch.setDevice(gpuid)`
- Including library: `require 'libname'`

Tensor Operations 1/2

- `X = torch.FloatTensor(dim1, dim2, dim3, dim4)`
- `Y = torch.zeros(dim1, dim2, dim3, dim4)`
- `X:add(1)`, `X:add(Y)` – addition, X and Y must have same size
Note: `X:sub()` - is not subtraction, but sub-tensor!!
- `X:mul(2)` – multiply by scalar
- `X:cmul(Y)` – point wise multiplication, X and Y must have same size
- `X:fill(3)` – fill tensor with scalar
- `X:normal(mean, stdv)` – fill tensor with random values from normal distribution
- `X:unifrom(min, max)` – fill tensor with random values from uniform distribution
- `Y = X:cuda()` - copy tensor to GPU
- `Y = X:float()` - convert tensor to float (if X was CudaTensor then it will be copy to CPU)

Tensor Operations 2/2

- `torch.save('filename', tensor)` – save tensor to file
- `Torch.load('filename')` – load tensor from file
- `X[a][b][c][d]` – access single entry in 4 dimensional
- `X:narrow(1, 2, 4)` – take only range from 2 to 4 of the first dimension of the tensor
- `X:copy(Y)` – copy Y into X, X and Y must have same size

Other Useful Stuff

- `X = {}` - create empty table
- `X[1]` – accessing field in table by index
- `X.name` – accessing field in table by name
- `string.format("%10.5f, %10.5f\n", x, y)` – formatting string
- `string = 'text' .. 'text'` – concatenate strings
- `print('text: ' .. value)` – printing text
- `disp1 = image.display{image=img, win=disp1, zoom=2}` – display image

Creating and Using a Model

- `Model = nn.Sequential()` – create the container
- `Model.add(nn.Linear(in, out))` – add linear layer
- `Model.add(nn.ReLU())` – add ReLU layer
- ...
- `Model.forward(input)` – prediction
- `Model.backward(input, gradients)` – back propagation
- `Model.training()` – set model to training mode
- `Model.evaluate()` – set model to evaluation mode
- `criterion = nn.MSECriterion()` – set criterion

Optimization 1/2

- `OptimState = {
 learningRate = 0.1,
 weightDecay = 0.0,
 momentum = 0.9
}` – setup the optimization parameters
- `optimMethod = optim.sgd` – select optimization method
- `parameters, gradParameters = model:getParameters()` – get model parameters
- `optimMethod(feval, parameters, optimState)` – run the optimization

Optimization 2/2

- `local feval = function(x)`
 `gradParameters:zero()`
 `local f = doBatch(model, criterion, data, data)`
 `return f, gradParameters`
`end` – setup evaluation function
- `local function doBatch(mod, crit, data, labels)`
 `local output = mod:forward(data)`
 `local f = crit:forward(output, labels)`
 `local df_do = crit:backward(output, labels)`
 `mod:backward(data, df_do)`
 `return f`
`end` – setup “do batch” function