# 源码分析

Person Search with Natural Language Description项目采用lua语言，基于torch框架实现。通过torch框架的cmdline模块，可以指定训练过程中的各项参数，包括最大迭代次数，输入数据路径等，具体如下。

|  |  |  |  |
| --- | --- | --- | --- |
| cnn\_optim\_beta | 0.999 | cnn\_model | model/VGG16\_iter\_50000.caffemodel |
| finetune\_cnn\_after | -1 | drop\_prob\_lm | 0.5 |
| batch\_size | 32 | grad\_clip | 5 |
| val\_images\_use | 500 | cnn\_weight\_decay | 0 |
| optim\_epsilon | 1.00E-08 | input\_json | data/reidtalk.json |
| input\_encoding\_size | 512 | num\_layers | 1 |
| losses\_log\_every | 25 | cnn\_proto | model/VGG\_ILSVRC\_16\_layers\_deploy.prototxt |
| id |  | learning\_rate\_decay\_start | -1 |
| optim\_beta | 0.999 | cnn\_optim | adam |
| input\_h5 | data/reidtalk.h5 | max\_iters | -1 |
| rnn\_size | 512 | checkpoint\_path | snapshot |
| cnn\_learning\_rate | 1.00E-05 | save\_checkpoint\_every | 500 |
| cnn\_optim\_alpha | 0.8 | learning\_rate | 0.0004 |
| neg\_time | 3 | start\_from |  |
| seed | 123 | backend | cudnn |
| optim | adam | learning\_rate\_decay\_every | 50000 |
| gpuid | -1 | optim\_alpha | 0.8 |

项目代码中有两个关键子网络分别用于处理language和image。其中Image network使用了caffee 格式的VGG16，使用loadcaffe包来进行解析加载，对应如下代码：

local cnn\_raw = loadcaffe.load(opt.cnn\_proto, opt.cnn\_model, cnn\_backend)

protos.cnn = net\_utils.build\_cnn(cnn\_raw, {encoding\_size = opt.input\_encoding\_size, backend = cnn\_backend})

。Lanuage model基于nn包中module模块自行实现。nn是torch中用于实现各种神经网络计算的包，nn中的module是一个抽象类，定义了各种训练一个神经网络的基本方法，并且支持序列化，以便于保存训练出的模型。下面摘录部分源码作具体说明：

-----------------------------------------------------

-- Language Model core

-----------------------

local layer, parent = torch.class('nn.LanguageModel', 'nn.Module')

function layer:\_\_init(opt)

function layer:createClones()

function layer:training()

function layer:evaluate()

function layer:updateOutput(input)

可以看到，代码定义了nn.LanguageModel，继承自nn.module。Layer：\_\_init函数用于初始化language model, opt是用于初始化的参数，具体如下：

|  |  |
| --- | --- |
| seq\_length | 50 |
| input\_encoding\_size | 512 |
| batch\_size | 32 |
| num\_layers | 1 |
| vocab\_size | 3800 |
| dropout | 0.5 |
| rnn\_size | 512 |

参数中指定了用于batch trainning的大小，neural network的layer的数目，language tensor的size等信息。实现了两个关键子网络之后，要选择合适的Criterions。Criterions用于帮助训练神经网络，给定输入值和目标值之后，Criterions用来根据loss函数计算梯度。代码中选用了BCECriterion，即二值交叉熵，公式如下：

crossentropy(t,o) = -(t \* log(o) + (1 - t) \* log(1 - o))，对应代码如下：protos.crit = nn.BCECriterion() 。确定了language modle和image network，以及Criterions之后，还需要定义loss function，snippet如下：

local function lossFun()

protos.cnn:training()

protos.lm:training()

grad\_params:zero()

if opt.finetune\_cnn\_after >= 0 and iter >= opt.finetune\_cnn\_after then

cnn\_grad\_params:zero()

end

local data = loader:getBatch{batch\_size = opt.batch\_size, split = 'train', seq\_per\_img = 1, neg\_time = opt.neg\_time}

local feats = protos.cnn:forward(data.images)

local logprobs = protos.lm:forward{feats, data.labels, data.seqlen}

local loss = protos.crit:forward(logprobs, data.cls:cuda())

local predictions = logprobs:float():ge(0.5)

local correct = predictions:long():eq(data.cls:long():view(logprobs:size(1), 1):expandAs(logprobs))

top1 = correct:sum() / logprobs:size(1)

local dlogprobs = protos.crit:backward(logprobs, data.cls:cuda())

local dfeats, ddummy = unpack(protos.lm:backward({feats, data.labels, data.seqlen}, dlogprobs))

if opt.finetune\_cnn\_after >= 0 and iter >= opt.finetune\_cnn\_after then

local dx = protos.cnn:backward(data.images, dfeats)

end

grad\_params:clamp(-opt.grad\_clip, opt.grad\_clip)

if opt.cnn\_weight\_decay > 0 then

cnn\_grad\_params:add(opt.cnn\_weight\_decay, cnn\_params)

cnn\_grad\_params:clamp(-opt.grad\_clip, opt.grad\_clip)

end

local losses = { total\_loss = loss }

return losses

end

其中比较关键的有trainning，forward，和backward几个函数，其中training函数是自定义函数用于具体的训练逻辑，以language model为例子，代码如下：

function layer:training()

if self.clones == nil then self:createClones() end -- create these lazily if needed

for k,v in pairs(self.lookup\_tables) do v:training() end

for k,v in pairs(self.emb\_imgs) do v:training() end

for k,v in pairs(self.clones) do v:training() end

for k,v in pairs(self.attentions) do v:training() end

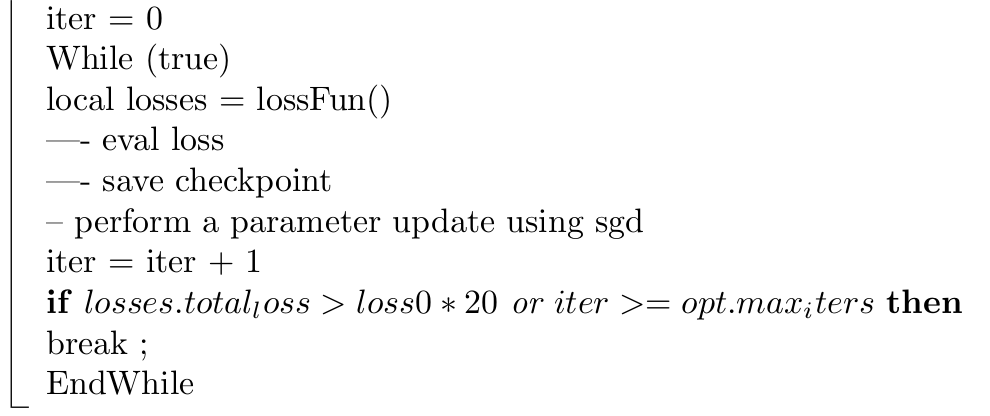
self.sigmoid:training()

end

。

Forward和backward是继承自module的函数，函数签名分别为：[output] forward(input)，[gradInput] backward(input, gradOutput)。Backward即用来执行backpropagation这一步。

定义完模型的各个组件之后，就可以开始进行训练，也就进入了代码main loop之中，main loop 部分语义如下：



可以看到当loss满足一定条件或到底指定迭代次数之后，训练停止。