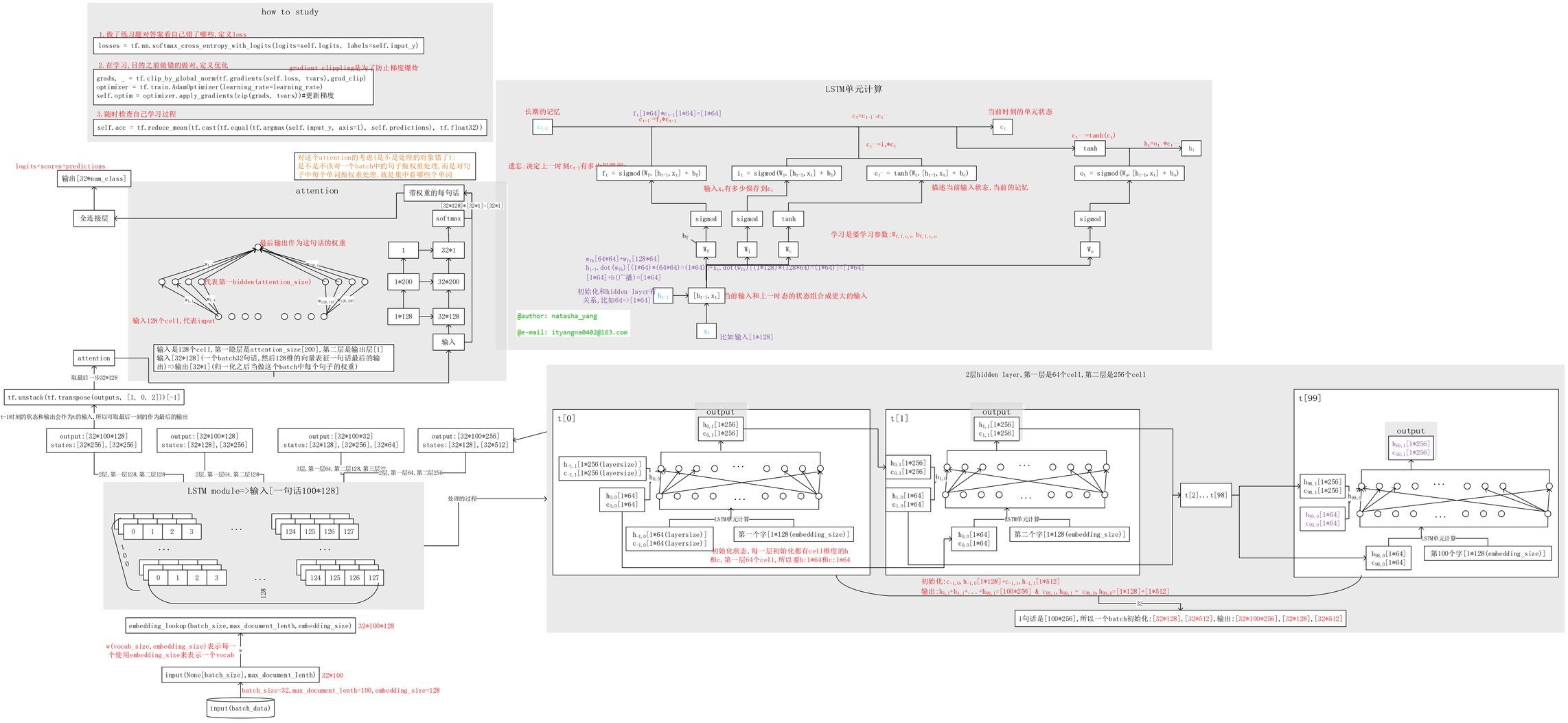
how to study
1.做了练习题对答案看自己错了哪些,定义loss
losses = tf.nn.softmax_cross_entropy_with_logits(logits=self.logits, labels=self.input_y)
2.在学习,目的之前做错的做对,定义优化
gradient clippling是为了防止梯度爆炸
grads, _ = tf.clip_by_global_norm(tf.gradients(self.loss, tvars),grad_clip)
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
self.optim = optimizer.apply_gradients(zip(grads, tvars))#更新梯度
3.随时检查自己学习过程
self.acc = tf.reduce_mean(tf.cast(tf.equal(tf.argmax(self.input_y, axis=1), self.predictions), tf.float32))
LSTM单元计算
长期的记忆
c_{t-1}
$f_t[1*64]*c_{t-1}[1*64]=[1*64]$
$c_{t-1}=f_t*c_{t-1}$
$c_t=c_{t-1}+c_t^{\sim}$
当前时刻的单元状态
c_t
$c_t^{\sim}=i_t*c_t^{\sim}$
$c_t^{\sim\cdots}=\tanh(c_t)$
$h_t=o_t*c_t^{\sim\cdots}$
h_t
遗忘:决定上一时刻c_{t-1}有多少保留到c_t
$f_t = \mathrm{sigmod}(W_f, [h_{t-1}, x_t] + b_f)$
$i_t = \mathrm{sigmod}(W_i, [h_{t-1}, x_t] + b_i)$
$c_t^{\sim} = \tanh(W_c, [h_{t-1}, x_t] + b_c)$
$o_t = \mathrm{sigmod}(W_o, [h_{t-1}, x_t] + b_o)$
输入x_t有多少保存到c_t
描述当前输入状态,当前的记忆
sigmod
sigmod
tanh
sigmod
b_f
W_f
W_i
W_c
W_o
学习是要学习参数:$W_{f,i,c,o}$ $b_{f,i,c,o}$
$w_h[64*64]+w_x[128*64]$
$h_{t-1}.\mathrm{dot}(w_h)[(1*64)*(64*64)=(1*64)]+x_t.\mathrm{dot}(w_x)[(1*128)*(128*64)=(1*64)]=[1*64]$
$[1*64]+b$(广播)$=[1*64]$
初始化和hidden layer有关系,比如64=>[1*64]
h_{t-1}
$[h_{t-1}, x_t]$
当前输入和上一时态的状态组合成更大的输入
x_t
比如输入[1*128]
@author: natasha_yang
@e-mail: ityangna0402@163.com
logits+scores+predictions
输出[32*num_class]
对这个attention的考虑(是不是处理的对象错了):是不是不该对一个batch中的句子做权重处理,而是对句子中每个单词做权重处理,就是集中看哪些个单词
attention
带权重的每句话
[32*128]*[32*1]=[32*1]
全连接层
softmax
最后输出作为这句话的权重
1
32*1
1*200
32*200
1*128
32*128
输入
代表第一hidden(attention_size)
输入128个cell,代表input
输入是128个cell,第一隐层是attention_size[200],第二层是输出层[1]
输入[32*128](一个batch32句话,然后128维的向量表征一句话最后的输出)=>输出[32*1](归一化之后当做这个batch中每个句子的权重)
attention
取最后一步32*128
tf.unstack(tf.transpose(outputs, [1, 0, 2]))[-1]
t-1时刻的状态和输出会作为t的输入,所以可取最后一刻的作为最后的输出
output:[32*100*128]
states:[32*256],[32*256]
output:[32*100*128]
states:[32*128],[32*256]
output:[32*100*32]
states:[32*128],[32*256],[32*64]
output:[32*100*256]
states:[32*128],[32*512]
2层,第一层128,第二层128
2层,第一层64,第二层128
3层,第一层64,第二层128,第三层32
2层,第一层64,第二层256
LSTM module=>输入[一句话100*128]
处理的过程
0 1 2 3
124 125 126 127
100
128
embedding_lookup(batch_size,max_document_lenth,embedding_size)
32*100*128
w(vocab_size,embedding_size)表示每一个使用embedding_size来表示一个vocab
input(None[batch_size],max_document_lenth)
32*100
batch_size=32,max_document_lenth=100,embedding_size=128
input(batch_data)
2层hidden layer,第一层是64个cell,第二层是256个cell
t[0]
output
$h_{0,1}[1*256]$
$c_{0,1}[1*256]$
$h_{-1,1}[1*256(layersize)]$
$c_{-1,1}[1*256(layersize)]$
$h_{0,0}$
$h_{0,0}[1*64]$
$c_{0,0}[1*64]$
LSTM单元计算
$h_{-1,0}[1*64(layersize)]$
$c_{-1,0}[1*64(layersize)]$
第一个字[1*128(embedding_size)]
初始化状态,每一层初始化都有cell维度的h和c,第一层64个cell,所以要h:1*64和c:1*64
t[1]
output
$h_{1,1}[1*256]$
$c_{1,1}[1*256]$
$h_{0,1}[1*256]$
$c_{0,1}[1*256]$
$h_{1,0}$
$h_{1,0}[1*64]$
$c_{1,0}[1*64]$
LSTM单元计算
$h_{0,0}[1*64]$
$c_{0,0}[1*64]$
第二个字[1*128(embedding_size)]
t[2]...t[98]
t[99]
output
$h_{99,1}[1*256]$
$c_{99,1}[1*256]$
$h_{98,1}[1*256]$
$c_{98,1}[1*256]$
$h_{99,0}$
$h_{99,0}[1*64]$
$c_{99,0}[1*64]$
LSTM单元计算
$h_{98,0}[1*64]$
$c_{98,0}[1*64]$
第100个字[1*128(embedding_size)]
初始化:$c_{-1,0}, h_{-1,0}[1*128]+c_{-1,1}, h_{-1,1}[1*512]$
输出:$h_{0,1}+h_{1,1}+\ldots+h_{99,1}=[100*256]$ & $c_{99,1}, h_{99,1}+c_{99,0}, h_{99,0}=[1*128]+[1*512]$
32
1句话是[100*256],所以一个batch初始化:[32*128],[32*512],输出:[32*100*256],[32*128],[32*512]