import tensorflow as tf

import cPickle

import numpy as np

import time

MAX\_ITER = 100

class PriorCourseGrade(object):

def \_\_init\_\_(self, C, K=20, out="result/"):

self.C = C

self.K = K

self.out = out

# self.graph = None

# self.loss = None

self.setup\_graph()

def setup\_graph(self):

self.graph = tf.Graph()

with self.graph.as\_default():

with tf.name\_scope("input"):

self.input\_grades = tf.placeholder(tf.float32, shape=[None, self.C + 1], name="input\_grades")

self.output\_course = tf.placeholder(tf.int64, shape=[None], name="output\_course")

self.target\_grade = tf.placeholder(tf.float32, shape=[None], name="target\_grade")

self.setup\_network()

self.setup\_loss()

self.setup\_optimizer()

self.init = tf.global\_variables\_initializer()

self.graph.finalize()

def setup\_network(self):

with tf.name\_scope("embedding"):

self.embds = tf.Variable(tf.random\_uniform([self.C + 1, self.K], minval=0.0, maxval=1.0), name="embds") # self.C is bias embedding

input\_grades\_enlarge = tf.tile(tf.expand\_dims(self.input\_grades, axis=-1),[1,1,self.K])

input\_scaled\_embds = tf.multiply(input\_grades\_enlarge, self.embds)

# maximum pooling #

self.input\_embd = tf.reduce\_max(input\_scaled\_embds, axis=1)

# output course embd #

self.output\_embd = tf.nn.embedding\_lookup(self.embds, self.output\_course)

with tf.name\_scope("compare"):

self.diff = tf.nn.relu(self.output\_embd - self.input\_embd)

with tf.name\_scope("predict"):

self.predict\_grade = tf.divide(1.0, tf.add(1.0, tf.reduce\_sum(self.diff, axis=-1)))

def setup\_loss(self):

with tf.name\_scope("loss"):

self.loss = tf.losses.mean\_squared\_error(self.target\_grade, self.predict\_grade)

def setup\_optimizer(self):

self.optimizer = tf.train.AdamOptimizer(learning\_rate=0.01).minimize(self.loss)

def train(self, data, data\_valid=None, batch\_size=100, max\_iter=MAX\_ITER):

self.batch\_size = 100

print("start training")

start = time.time()

sess = tf.Session(graph=self.graph)

for epoch in xrange(max\_iter):

loss\_epoch = 0.0

N\_samples = 0

for data\_batched in data.batch\_generate(batch\_size=self.batch\_size):

[input\_grades\_batched, output\_course\_batched], target\_grade\_batched = data\_batched

feed\_dict = {

self.input\_grades: input\_grades\_batched,

self.output\_course: output\_course\_batched,

self.target\_grade: target\_grade\_batched

}

\_, loss, embds = sess.run([self.optimizer, self.loss, self.embds], feed\_dict=feed\_dict)

loss\_epoch += loss

N\_samples += self.batch\_size

print("epoch %d: loss %f" % (epoch, loss\_epoch / N\_samples))

with open(self.out + "embds\_epoch%03d" % epoch, "wb") as of:

cPickle.dump(embds, of)

end = time.time()

print("use %f s" % (end - start))

print("done training, using in total %f s" % (time.time() - start))