import tensorflow.compat.v1 as tf

from twml.contrib.utils import masks, math\_fns

def get\_pair\_loss(pairwise\_label\_scores, pairwise\_predicted\_scores,

params):

"""

Paiwise learning-to-rank ranknet loss

Check paper https://www.microsoft.com/en-us/research/publication/

learning-to-rank-using-gradient-descent/

for more information

Args:

pairwise\_label\_scores: a dense tensor of shape [n\_data, n\_data]

pairwise\_predicted\_scores: a dense tensor of shape [n\_data, n\_data]

n\_data is the number of tweet candidates in a BatchPredictionRequest

params: network parameters

mask options: full\_mask and diag\_mask

Returns:

average loss over pairs defined by the masks

"""

n\_data = tf.shape(pairwise\_label\_scores)[0]

if params.mask == "full\_mask":

# full\_mask that only covers pairs that have different labels

# (all pairwise\_label\_scores = 0.5: selfs and same labels are 0s)

mask, pair\_count = masks.full\_mask(n\_data, pairwise\_label\_scores)

else:

# diag\_mask that covers all pairs

# (only selfs/diags are 0s)

mask, pair\_count = masks.diag\_mask(n\_data, pairwise\_label\_scores)

# pairwise sigmoid\_cross\_entropy\_with\_logits loss

loss = tf.cond(tf.equal(pair\_count, 0), lambda: 0.,

lambda: \_get\_average\_cross\_entropy\_loss(pairwise\_label\_scores,

pairwise\_predicted\_scores, mask, pair\_count))

return loss

def get\_lambda\_pair\_loss(pairwise\_label\_scores, pairwise\_predicted\_scores,

params, swapped\_ndcg):

"""

Paiwise learning-to-rank lambdarank loss

faster than the previous gradient method

Note: this loss depends on ranknet cross-entropy

delta NDCG is applied to ranknet cross-entropy

Hence, it is still a gradient descent method

Check paper http://citeseerx.ist.psu.edu/viewdoc/

download?doi=10.1.1.180.634&rep=rep1&type=pdf for more information

for more information

Args:

pairwise\_label\_scores: a dense tensor of shape [n\_data, n\_data]

pairwise\_predicted\_scores: a dense tensor of shape [n\_data, n\_data]

n\_data is the number of tweet candidates in a BatchPredictionRequest

params: network parameters

swapped\_ndcg: swapped ndcg of shape [n\_data, n\_data]

ndcg values when swapping each pair in the prediction ranking order

mask options: full\_mask and diag\_mask

Returns:

average loss over pairs defined by the masks

"""

n\_data = tf.shape(pairwise\_label\_scores)[0]

if params.mask == "full\_mask":

# full\_mask that only covers pairs that have different labels

# (all pairwise\_label\_scores = 0.5: selfs and same labels are 0s)

mask, pair\_count = masks.full\_mask(n\_data, pairwise\_label\_scores)

else:

# diag\_mask that covers all pairs

# (only selfs/diags are 0s)

mask, pair\_count = masks.diag\_mask(n\_data, pairwise\_label\_scores)

# pairwise sigmoid\_cross\_entropy\_with\_logits loss

loss = tf.cond(tf.equal(pair\_count, 0), lambda: 0.,

lambda: \_get\_average\_cross\_entropy\_loss(pairwise\_label\_scores,

pairwise\_predicted\_scores, mask, pair\_count, swapped\_ndcg))

return loss

def \_get\_average\_cross\_entropy\_loss(pairwise\_label\_scores, pairwise\_predicted\_scores,

mask, pair\_count, swapped\_ndcg=None):

"""

Average the loss for a batchPredictionRequest based on a desired number of pairs

"""

loss = tf.nn.sigmoid\_cross\_entropy\_with\_logits(labels=pairwise\_label\_scores,

logits=pairwise\_predicted\_scores)

loss = mask \* loss

if swapped\_ndcg is not None:

loss = loss \* swapped\_ndcg

loss = tf.reduce\_sum(loss) / pair\_count

return loss

def get\_listmle\_loss(labels, predicted\_scores):

r"""

listwise learning-to-rank listMLE loss

Note: Simplified MLE formula is used in here (omit the proof in here)

\sum\_{s=1}^{n-1} (-predicted\_scores + ln(\sum\_{i=s}^n exp(predicted\_scores)))

n is tf.shape(predicted\_scores)[0]

Check paper http://icml2008.cs.helsinki.fi/papers/167.pdf for more information

Args:

labels: a dense tensor of shape [n\_data, 1]

n\_data is the number of tweet candidates in a BatchPredictionRequest

predicted\_scores: a dense tensor of same shape and type as labels

Returns:

average loss

"""

labels = tf.reshape(labels, [-1, 1])

n\_data = tf.shape(labels)[0]

predicted\_scores = tf.reshape(predicted\_scores, [-1, 1])

predicted\_scores\_ordered\_by\_labels = \_get\_ordered\_predicted\_scores(labels,

predicted\_scores, n\_data)

loss = (-1) \* tf.reduce\_sum(predicted\_scores)

# sum over 1 to n\_data - 1

temp = tf.gather(predicted\_scores\_ordered\_by\_labels, [n\_data - 1])

temp = tf.reshape(temp, [])

loss = tf.add(loss, temp)

exps = tf.exp(predicted\_scores\_ordered\_by\_labels)

exp\_sum = tf.reduce\_sum(exps)

# clip exp\_sum for safer log

loss = tf.add(loss, math\_fns.safe\_log(exp\_sum))

iteration = tf.constant(0)

def \_cond(iteration, loss, exp\_sum, exp):

return tf.less(iteration, n\_data - 2)

def \_gen\_loop\_body():

def loop\_body(iteration, loss, exp\_sum, exps):

temp = tf.gather(exps, [iteration])

temp = tf.reshape(temp, [])

exp\_sum = tf.subtract(exp\_sum, temp)

# clip exp\_sum for safer log

loss = tf.add(loss, math\_fns.safe\_log(exp\_sum))

return tf.add(iteration, 1), loss, exp\_sum, exps

return loop\_body

iteration, loss, exp\_sum, exps = tf.while\_loop(\_cond, \_gen\_loop\_body(),

(iteration, loss, exp\_sum, exps))

loss = loss / tf.cast(n\_data, dtype=tf.float32)

return loss

def \_get\_ordered\_predicted\_scores(labels, predicted\_scores, n\_data):

"""

Order predicted\_scores based on sorted labels

"""

sorted\_labels, ordered\_labels\_indices = tf.nn.top\_k(

tf.transpose(labels), k=n\_data)

ordered\_labels\_indices = tf.transpose(ordered\_labels\_indices)

predicted\_scores\_ordered\_by\_labels = tf.gather\_nd(predicted\_scores,

ordered\_labels\_indices)

return predicted\_scores\_ordered\_by\_labels

def get\_attrank\_loss(labels, predicted\_scores, weights=None):

"""

Modified listwise learning-to-rank AttRank loss

Check paper https://arxiv.org/abs/1804.05936 for more information

Note: there is an inconsistency between the paper statement and

their public code

Args:

labels: a dense tensor of shape [n\_data, 1]

n\_data is the number of tweet candidates in a BatchPredictionRequest

predicted\_scores: a dense tensor of same shape and type as labels

weights: a dense tensor of the same shape as labels

Returns:

average loss

"""

# The authors immeplemented the following, which is basically listnet

# attention\_labels = \_get\_attentions(labels)

# attention\_labels = tf.reshape(attention\_labels, [1, -1])

# predicted\_scores = tf.reshape(predicted\_scores, [1, -1])

# loss = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits(labels=attention\_labels,

# logits=predicted\_scores))

# The paper proposed the following

# attention\_labels = \_get\_attentions(labels)

# # However the following line is wrong based on their statement

# # as \_get\_attentions can give 0 results when input < 0

# # and the result cannot be used in \_get\_attrank\_cross\_entropy

# # log(a\_i^S)

# # attention\_predicted\_scores = \_get\_attentions(predicted\_scores)

# loss = \_get\_attrank\_cross\_entropy(attention\_labels, attention\_predicted\_scores)

# # the range of attention\_predicted\_scores is [0, 1)

# # this gives sigmoid [0.5, 0.732)

# # hence, it is not good to use in sigmoid\_cross\_entropy\_with\_logits either

# Implemented the following instead

# \_get\_attentions is applied to labels

# softmax is applied to predicted\_scores

reshaped\_labels = tf.reshape(labels, [1, -1])

attention\_labels = \_get\_attentions(reshaped\_labels)

reshaped\_predicted\_scores = tf.reshape(predicted\_scores, [1, -1])

attention\_predicted\_scores = tf.nn.softmax(reshaped\_predicted\_scores)

loss = \_get\_attrank\_cross\_entropy(attention\_labels, attention\_predicted\_scores)

return loss

def \_get\_attentions(raw\_scores):

"""

Used in attention weights in AttRank loss

for a query/batch/batchPreidictionRequest

(a rectified softmax function)

"""

not\_consider = tf.less\_equal(raw\_scores, 0)

mask = tf.ones(tf.shape(raw\_scores)) - tf.cast(not\_consider, dtype=tf.float32)

mask = tf.cast(mask, dtype=tf.float32)

expon\_labels = mask \* tf.exp(raw\_scores)

expon\_label\_sum = tf.reduce\_sum(expon\_labels)

# expon\_label\_sum is safe as a denominator

attentions = math\_fns.safe\_div(expon\_labels, expon\_label\_sum)

return attentions

def \_get\_attrank\_cross\_entropy(labels, logits):

# logits is not safe based on their satement

# do not use this function directly elsewhere

results = labels \* math\_fns.safe\_log(logits) + (1 - labels) \* math\_fns.safe\_log(1 - logits)

results = (-1) \* results

results = tf.reduce\_mean(results)

return results

def get\_listnet\_loss(labels, predicted\_scores, weights=None):

"""

Listwise learning-to-rank listet loss

Check paper https://www.microsoft.com/en-us/research/

wp-content/uploads/2016/02/tr-2007-40.pdf

for more information

Args:

labels: a dense tensor of shape [n\_data, 1]

n\_data is the number of tweet candidates in a BatchPredictionRequest

predicted\_scores: a dense tensor of same shape and type as labels

weights: a dense tensor of the same shape as labels

Returns:

average loss

"""

# top one probability is the same as softmax

labels\_top\_one\_probs = \_get\_top\_one\_probs(labels)

predicted\_scores\_top\_one\_probs = \_get\_top\_one\_probs(predicted\_scores)

if weights is None:

loss = tf.reduce\_mean(

\_get\_listnet\_cross\_entropy(labels=labels\_top\_one\_probs,

logits=predicted\_scores\_top\_one\_probs))

return loss

loss = tf.reduce\_mean(

\_get\_listnet\_cross\_entropy(labels=labels\_top\_one\_probs,

logits=predicted\_scores\_top\_one\_probs) \* weights) / tf.reduce\_mean(weights)

return loss

def \_get\_top\_one\_probs(labels):

"""

Used in listnet top-one probabilities

for a query/batch/batchPreidictionRequest

(essentially a softmax function)

"""

expon\_labels = tf.exp(labels)

expon\_label\_sum = tf.reduce\_sum(expon\_labels)

# expon\_label\_sum is safe as a denominator

attentions = expon\_labels / expon\_label\_sum

return attentions

def \_get\_listnet\_cross\_entropy(labels, logits):

"""

Used in listnet

cross entropy on top-one probabilities

between ideal/label top-one probabilities

and predicted/logits top-one probabilities

for a query/batch/batchPreidictionRequest

"""

# it is safe to use log on logits

# that come from \_get\_top\_one\_probs

# do not use this function directly elsewhere

results = (-1) \* labels \* math\_fns.safe\_log(logits)

return results

def get\_pointwise\_loss(labels, predicted\_scores, weights=None):

"""

Pointwise learning-to-rank pointwise loss

Args:

labels: a dense tensor of shape [n\_data, 1]

n\_data is the number of tweet candidates in a BatchPredictionRequest

predicted\_scores: a dense tensor of same shape and type as labels

weights: a dense tensor of the same shape as labels

Returns:

average loss

"""

if weights is None:

loss = tf.reduce\_mean(

tf.nn.sigmoid\_cross\_entropy\_with\_logits(labels=labels,

logits=predicted\_scores))

return loss

loss = tf.reduce\_mean(tf.nn.sigmoid\_cross\_entropy\_with\_logits(labels=labels,

logits=predicted\_scores) \* weights) / tf.reduce\_mean(weights)

return loss