# pylint: disable=too-many-branches

""" This module includes functions for managing learning rate decay """

import tensorflow.compat.v1 as tf

def get\_learning\_rate\_decay\_fn(params):

"""

Returns a learning rate decay function that takes the initial

learning\_rate and global\_step

as arguments and returns the current learning rate.

Currently supports params.learning\_rate\_decay values of:

exponential | polynomial | piecewise\_constant | cosine | cosine restarts.

See `Decaying the Leanring Rate

<https://www.tensorflow.org/api\_guides/python/train#Decaying\_the\_learning\_rate>`\_ for details.

Arguments:

params:

a tensorflow.contrib.train.HParams object containing the relevant hyperparameters.

"""

paramsv = params.values()

if 'learning\_rate\_decay' not in paramsv or params.learning\_rate\_decay == 'no\_learning\_rate\_decay':

return None

elif params.learning\_rate\_decay == 'exponential\_learning\_rate\_decay':

if 'decay\_steps' not in paramsv:

raise ValueError("Expecting params.decay\_steps for "

"params.learning\_rate\_decay == 'exponential'")

if 'exponential\_decay\_rate' not in paramsv:

raise ValueError("Expecting params.exponential\_decay\_rate for "

"params.learning\_rate\_decay == 'exponential'")

def exponential\_decay\_fn(learning\_rate, global\_step):

""" exponential decay function to be passed to optimize\_loss """

return tf.train.exponential\_decay(

learning\_rate=learning\_rate,

global\_step=global\_step,

decay\_steps=params.decay\_steps,

decay\_rate=params.exponential\_decay\_rate

)

return exponential\_decay\_fn

elif params.learning\_rate\_decay == 'piecewise\_constant\_learning\_rate\_decay':

if 'piecewise\_constant\_boundaries' not in paramsv:

raise ValueError("Expecting params.piecewise\_constant\_boundaries for "

"params.learning\_rate\_decay == 'piecewise\_constant'")

if 'piecewise\_constant\_values' not in paramsv:

raise ValueError("Expecting params.piecewise\_constant\_values for "

"params.learning\_rate\_decay == 'piecewise\_constant'")

# pylint: disable=unused-argument

def piecewise\_constant\_fn(learning\_rate, global\_step):

""" piecewise\_constant decay function to be passed to optimize\_loss """

return tf.train.piecewise\_constant(

x=global\_step,

boundaries=params.piecewise\_constant\_boundaries,

values=params.piecewise\_constant\_values

)

return piecewise\_constant\_fn

elif params.learning\_rate\_decay == 'polynomial\_learning\_rate\_decay':

if 'decay\_steps' not in paramsv:

raise ValueError("Expecting params.decay\_steps for "

"params.learning\_rate\_decay == 'polynomial'")

if 'end\_learning\_rate' not in paramsv:

raise ValueError("Expecting params.end\_learning\_rate for "

"params.learning\_rate\_decay == 'polynomial'")

def polynomial\_decay\_fn(learning\_rate, global\_step):

""" polynomial decay function to be passed to optimize\_loss """

return tf.train.polynomial\_decay(

learning\_rate=learning\_rate,

global\_step=global\_step,

decay\_steps=params.decay\_steps,

end\_learning\_rate=params.end\_learning\_rate,

power=params.polynomial\_power if 'polynomial\_power' in paramsv else 1.0,

)

return polynomial\_decay\_fn

elif params.learning\_rate\_decay == 'inverse\_learning\_rate\_decay':

if 'min\_learning\_rate' not in paramsv:

raise ValueError("Expecting params.min\_learning\_rate for "

"params.learning\_rate\_decay == 'inverse'")

if 'decay\_rate' not in paramsv:

raise ValueError("Expecting params.decay\_rate for "

"params.learning\_rate\_decay == 'inverse'")

if 'decay\_steps' not in paramsv:

raise ValueError("Expecting params.decay\_steps for "

"params.learning\_rate\_decay == 'inverse'")

def bounded\_inverse\_time\_decay\_fn(learning\_rate, global\_step):

'''

Returns the decayed learning\_rate by applying the function:

decayed\_lr = max(lr /(1 + decay\_rate \* floor(global\_step /decay\_step)),

min\_learning\_rate)

Arguments:

learning\_rate:

A scalar `float32` or `float64` `Tensor` or a Python number.

The initial learning rate.

global\_step:

A scalar `int32` or `int64` `Tensor` or a Python number.

Global step to use for the decay computation. Must not be negative.

min\_learning\_rate:

A scalar `int32` or `int64` `Tensor` or a Python number.

Minimum possible learning\_rate. The decayed learning\_rate will not be

smaller than the min\_learning\_rate

decay\_steps:

How often to apply decay. In dbv1, this should be 1.

decay\_rate:

A scalar `int32` or `int64` `Tensor` or a Python number.

Rate in which we decay the learning rate.

Returns:

A scalar `Tensor` of the same type as `learning\_rate`. The decayed

learning rate.

'''

decayed\_rate = tf.train.inverse\_time\_decay(

learning\_rate=learning\_rate,

global\_step=global\_step,

decay\_steps=params.decay\_steps,

decay\_rate=params.decay\_rate)

# Getting dtype of returned Tensor

dtype = decayed\_rate.dtype

# Casting the min\_learning rate the same dtype as decayes rate

min\_learning\_rate = tf.cast(params.min\_learning\_rate, dtype)

# Returning the maximum between the two

return tf.maximum(decayed\_rate, min\_learning\_rate)

return bounded\_inverse\_time\_decay\_fn

elif params.learning\_rate\_decay == 'cosine\_learning\_rate\_decay':

if 'decay\_steps' not in paramsv:

raise ValueError("Expecting params.decay\_steps for "

"params.learning\_rate\_decay == 'cosine\_decay'")

if "alpha" not in paramsv:

raise ValueError("Expecting params.alpha for "

"params.learning\_rate\_decay == 'cosine\_decay'")

def cosine\_decay\_fn(learning\_rate, global\_step):

""" cosine decay function to be passed to optimize\_loss """

return tf.train.cosine\_decay(

learning\_rate=learning\_rate,

global\_step=global\_step,

decay\_steps=params.decay\_steps,

alpha=params.alpha

)

return cosine\_decay\_fn

elif params.learning\_rate\_decay == 'cosine\_restarts\_learning\_rate\_decay':

if 'first\_decay\_steps' not in paramsv:

raise ValueError("Expecting params.first\_decay\_steps for "

"params.learning\_rate\_decay == 'cosine\_restarts\_decay'")

if 't\_mul' not in paramsv:

raise ValueError("Expecting params.t\_mul for "

"params.learning\_rate\_decay == 'cosine\_restarts\_decay'")

if 'm\_mul' not in paramsv:

raise ValueError("Expecting params.m\_mul for "

"params.learning\_rate\_decay == 'cosine\_restarts\_decay'")

if "alpha" not in paramsv:

raise ValueError("Expecting params.alpha for "

"params.learning\_rate\_decay == 'cosine\_restarts\_decay'")

def cosine\_restart\_decay\_fn(learning\_rate, global\_step):

""" cosine decay function to be passed to optimize\_loss """

return tf.train.cosine\_decay\_restarts(

learning\_rate=learning\_rate,

global\_step=global\_step,

first\_decay\_steps=params.first\_decay\_steps,

t\_mul=params.t\_mul,

m\_mul=params.m\_mul,

alpha=params.alpha

)

return cosine\_restart\_decay\_fn

raise ValueError("Unsupported params.learning\_rate\_decay: %s" % params.learning\_rate\_decay)