# pylint: disable=protected-access, arguments-differ

"""

Command-line argument parsing for the Trainer.

"""

import argparse

from argparse import ArgumentError

from operator import attrgetter

import tempfile

import twml

import tensorflow.compat.v1 as tf

SERIAL = "serial"

TREE = "tree"

LOG\_LEVELS = {

"debug": tf.logging.DEBUG,

"info": tf.logging.INFO,

"warn": tf.logging.WARN,

"error": tf.logging.ERROR}

class SortingHelpFormatter(argparse.HelpFormatter):

"""

Used to sort args alphabetically in the help message.

"""

def add\_arguments(self, actions):

actions = sorted(actions, key=attrgetter('option\_strings'))

super(SortingHelpFormatter, self).add\_arguments(actions)

def \_set\_log\_level(level=None):

"""Sets the tensorflow log level to the input level."""

if level is None:

return None

level = level.lower()

if level not in LOG\_LEVELS.keys():

raise ValueError(f"Unexpected log level {level} was given but expected one of {LOG\_LEVELS.keys()}.")

tf.logging.set\_verbosity(LOG\_LEVELS[level])

tf.logging.info(f"Setting tensorflow logging level to {level} or {LOG\_LEVELS[level]}")

return level

def get\_trainer\_parser():

"""

Add common commandline args to parse for the Trainer class.

Typically, the user calls this function and then parses cmd-line arguments

into an argparse.Namespace object which is then passed to the Trainer constructor

via the params argument.

See the `code <\_modules/twml/argument\_parser.html#get\_trainer\_parser>`\_

for a list and description of all cmd-line arguments.

Args:

learning\_rate\_decay:

Defaults to False. When True, parses learning rate decay arguments.

Returns:

argparse.ArgumentParser instance with some useful args already added.

"""

parser = twml.DefaultSubcommandArgParse(formatter\_class=SortingHelpFormatter)

parser.add\_argument(

"--save\_dir", type=str, default=tempfile.mkdtemp(),

help="Path to the training result directory."

"supports local filesystem path and hdfs://default/<path> which requires "

"setting HDFS configuration via env variable HADOOP\_CONF\_DIR ")

parser.add\_argument(

"--export\_dir", type=str, default=None,

help="Path to the directory to export a SavedModel for prediction servers.")

parser.add\_argument(

"--log\_aggregation\_app\_id", type=str, default=None,

help="specify app\_id for log aggregation. disabled by default.")

parser.add\_argument(

"--train.batch\_size", "--train\_batch\_size", type=int, default=32,

dest='train\_batch\_size',

help="number of samples per training batch")

parser.add\_argument(

"--eval.batch\_size", "--eval\_batch\_size", type=int, default=32,

dest='eval\_batch\_size',

help="number of samples per cross-validation batch. Defaults to train\_batch\_size")

parser.add\_argument(

"--train.learning\_rate", "--learning\_rate", type=float, default=0.002,

dest='learning\_rate',

help="learning rate. Scales the gradient update.")

parser.add\_argument(

"--train.steps", "--train\_steps", type=int, default=-1,

dest='train\_steps',

help="number of training batches before running evaluation."

"Defaults to -1 (runs through entire dataset). "

"Only used for Trainer.[train,learn]. "

"For Trainer.train\_and\_evaluate, use train.max\_steps instead. ")

parser.add\_argument(

"--eval.steps", "--eval\_steps", type=int, default=-1,

dest="eval\_steps",

help="number of steps per evaluation. Each batch is a step."

"Defaults to -1 (runs through entire dataset). ")

parser.add\_argument(

"--eval.period", "--eval\_period", type=int, default=600,

dest="eval\_period",

help="Trainer.train\_and\_evaluate waits for this long after each evaluation. "

"Defaults to 600 seconds (evaluate every ten minutes). "

"Note that anything lower than 10\*60seconds is probably a bad idea because TF saves "

"checkpoints every 10mins by default. eval.delay is time to wait before doing first eval. "

"eval.period is time between successive evals.")

parser.add\_argument(

"--eval.delay", "--eval\_delay", type=int, default=120,

dest="eval\_delay",

help="Trainer.train\_and\_evaluate waits for this long before performing the first evaluation"

"Defaults to 120 seconds (evaluate after first 2 minutes of training). "

"eval.delay is time to wait before doing first eval. "

"eval.period is time between successive evals.")

parser.add\_argument(

"--train.max\_steps", "--train\_max\_steps", type=int, default=None,

dest="train\_max\_steps",

help="Stop training after this many global steps. Each training batch is its own step."

"If set to None, step after one train()/evaluate() call. Useful when train.steps=-1."

"If set to a non-positive value, loop forever. Usually useful with early stopping.")

parser.add\_argument(

"--train.log\_metrics", dest="train\_log\_metrics", action="store\_true", default=False,

help="Set this to true to see metrics during training. "

"WARNING: metrics during training does not represent model performance. "

"WARNING: use for debugging only as this slows down training.")

parser.add\_argument(

"--train.early\_stop\_patience", "--early\_stop\_patience", type=int, default=-1,

dest="early\_stop\_patience",

help="max number of evaluations (epochs) to wait for an improvement in the early\_stop\_metric."

"Defaults to -1 (no early-stopping)."

"NOTE: This can not be enabled when --distributed is also set.")

parser.add\_argument(

"--train.early\_stop\_tolerance", "--early\_stop\_tolerance", type=float, default=0,

dest="early\_stop\_tolerance",

help="a non-negative tolerance for comparing early\_stop\_metric."

"e.g. when maximizing the condition is current\_metric > best\_metric + tolerance."

"Defaults to 0.")

parser.add\_argument(

"--train.dataset\_shards", "--train\_dataset\_shards",

dest="train\_dataset\_shards",

type=int, default=None,

help="An int value that indicates the number of partitions (shards) for the dataset. This is"

" useful for codistillation and other techniques that require each worker to train on disjoint"

" partitions of the dataset.")

parser.add\_argument(

"--train.dataset\_shard\_index", "--train\_dataset\_shard\_index",

dest="train\_dataset\_shard\_index",

type=int, default=None,

help="An int value (starting at zero) that indicates which partition (shard) of the dataset"

" to use if --train.dataset\_shards is set.")

parser.add\_argument(

"--continue\_from\_checkpoint", dest="continue\_from\_checkpoint", action="store\_true",

help="DEPRECATED. This option is currently a no-op."

" Continuing from the provided checkpoint is now the default."

" Use --overwrite\_save\_dir if you would like to override it instead"

" and restart training from scratch.")

parser.add\_argument(

"--overwrite\_save\_dir", dest="overwrite\_save\_dir", action="store\_true",

help="Delete the contents of the current save\_dir if it exists")

parser.add\_argument(

"--data\_threads", "--num\_threads", type=int, default=2,

dest="num\_threads",

help="Number of threads to use for loading the dataset. "

"num\_threads is deprecated and to be removed in future versions. Use data\_threads.")

parser.add\_argument(

"--max\_duration", "--max\_duration", type=float, default=None,

dest="max\_duration",

help="Maximum duration (in secs) that training/validation will be allowed to run for before being automatically terminated.")

parser.add\_argument(

"--num\_workers", type=int, default=None,

help="Number of workers to use when training in hogwild manner on a single node.")

parser.add\_argument(

"--distributed", dest="distributed", action="store\_true",

help="Pass this flag to use train\_and\_evaluate to train in a distributed fashion"

"NOTE: You can not use early stopping when --distributed is enabled"

)

parser.add\_argument(

"--distributed\_training\_cleanup",

dest="distributed\_training\_cleanup",

action="store\_true",

help="Set if using distributed training on GKE to stop TwitterSetDeployment"

"from continuing training upon restarts (will be deprecated once we migrate off"

"TwitterSetDeployment for distributed training on GKE)."

)

parser.add\_argument(

"--disable\_auto\_ps\_shutdown", default=False, action="store\_true",

help="Disable the functionality of automatically shutting down parameter server after "

"distributed training complete (either succeed or failed)."

)

parser.add\_argument(

"--disable\_tensorboard", default=False, action="store\_true",

help="Do not start the TensorBoard server."

)

parser.add\_argument(

"--tensorboard\_port", type=int, default=None,

help="Port for tensorboard to run on. Ignored if --disable\_tensorboard is set.")

parser.add\_argument(

"--health\_port", type=int, default=None,

help="Port to listen on for health-related endpoints (e.g. graceful shutdown)."

"Not user-facing as it is set automatically by the twml\_cli."

)

parser.add\_argument(

"--stats\_port", type=int, default=None,

help="Port to listen on for stats endpoints"

)

parser.add\_argument(

"--experiment\_tracking\_path",

dest="experiment\_tracking\_path",

type=str, default=None,

help="The tracking path of this experiment. Format: \

user\_name:project\_name:experiment\_name:run\_name. The path is used to track and display \

a record of this experiment on ML Dashboard. Note: this embedded experiment tracking is \

disabled when the deprecated Model Repo TrackRun is used in your model config. ")

parser.add\_argument(

"--disable\_experiment\_tracking",

dest="disable\_experiment\_tracking",

action="store\_true",

help="Whether experiment tracking should be disabled.")

parser.add\_argument(

"--config.save\_checkpoints\_secs", "--save\_checkpoints\_secs", type=int, default=600,

dest='save\_checkpoints\_secs',

help="Configures the tf.estimator.RunConfig.save\_checkpoints\_secs attribute. "

"Specifies how often checkpoints are saved in seconds. Defaults to 10\*60 seconds.")

parser.add\_argument(

"--config.keep\_checkpoint\_max", "--keep\_checkpoint\_max", type=int, default=20,

dest='keep\_checkpoint\_max',

help="Configures the tf.estimator.RunConfig.keep\_checkpoint\_max attribute. "

"Specifies how many checkpoints to keep. Defaults to 20.")

parser.add\_argument(

"--config.tf\_random\_seed", "--tf\_random\_seed", type=int, default=None,

dest='tf\_random\_seed',

help="Configures the tf.estimator.RunConfig.tf\_random\_seed attribute. "

"Specifies the seed to use. Defaults to None.")

parser.add\_argument(

"--optimizer", type=str, default='SGD',

help="Optimizer to use: SGD (Default), Adagrad, Adam, Ftrl, Momentum, RMSProp, LazyAdam, DGC.")

parser.add\_argument(

"--gradient\_noise\_scale", type=float, default=None,

help="adds 0-mean normal noise scaled by this value. Defaults to None.")

parser.add\_argument(

"--clip\_gradients", type=float, default=None,

help="If specified, a global clipping is applied to prevent "

"the norm of the gradient to exceed this value. Defaults to None.")

parser.add\_argument(

"--dgc.density", "--dgc\_density", type=float, default=0.1,

dest="dgc\_density",

help="Specifies gradient density level when using deep gradient compression optimizer."

"E.g., default value being 0.1 means that only top 10%% most significant rows "

"(based on absolute value sums) are kept."

)

parser.add\_argument(

"--dgc.density\_decay", "--dgc\_density\_decay", type=bool, default=True,

dest="dgc\_density\_decay",

help="Specifies whether to (exponentially) decay the gradient density level when"

" doing gradient compression. If set 'False', the 'density\_decay\_steps', "

"'density\_decay\_rate' and 'min\_density' arguments will be ignored."

)

parser.add\_argument(

"--dgc.density\_decay\_steps", "--dgc\_density\_decay\_steps", type=int, default=10000,

dest="dgc\_density\_decay\_steps",

help="Specifies the step interval to perform density decay."

)

parser.add\_argument(

"--dgc.density\_decay\_rate", "--dgc\_density\_decay\_rate", type=float, default=0.5,

dest="dgc\_density\_decay\_rate",

help="Specifies the decay rate when perfoming density decay."

)

parser.add\_argument(

"--dgc.min\_density", "--dgc\_min\_density", type=float, default=0.1,

dest="dgc\_min\_density",

help="Specifies the minimum density level when perfoming density decay."

)

parser.add\_argument(

"--dgc.accumulation", "--dgc\_accumulation", type=bool, default=False,

dest="dgc\_accumulation",

help="Specifies whether to accumulate small gradients when using deep gradient compression "

"optimizer."

)

parser.add\_argument(

"--show\_optimizer\_summaries", dest="show\_optimizer\_summaries", action="store\_true",

help="When specified, displays gradients and learning rate in tensorboard."

"Turning it on has 10-20%% performance hit. Enable for debugging only")

parser.add\_argument(

"--num\_mkl\_threads", dest="num\_mkl\_threads", default=1, type=int,

help="Specifies how many threads to use for MKL"

"inter\_op\_ parallelism\_threds is set to TWML\_NUM\_CPUS / num\_mkl\_threads."

"intra\_op\_parallelism\_threads is set to num\_mkl\_threads.")

parser.add\_argument("--verbosity", type=\_set\_log\_level, choices=LOG\_LEVELS.keys(), default=None,

help="Sets log level to a given verbosity.")

parser.add\_argument(

"--feature\_importance.algorithm", dest="feature\_importance\_algorithm",

type=str, default=TREE, choices=[SERIAL, TREE],

help="""

There are two algorithms that the module supports, `serial` and `tree`.

The `serial` algorithm computes feature importances for each feature, and

the `tree` algorithm groups features by feature name prefix, computes feature

importances for groups of features, and then only 'zooms-in' on a group when the

importance is greater than the `--feature\_importance.sensitivity` value. The `tree` algorithm

will usually run faster, but for relatively unimportant features it will only compute an

upper bound rather than an exact importance value. We suggest that users generally stick

to the `tree` algorithm, unless if they have a very small number of features or

near-random model performance.

""")

parser.add\_argument(

"--feature\_importance.sensitivity", dest="feature\_importance\_sensitivity", type=float, default=0.03,

help="""

The maximum amount that permuting a feature group can cause the model performance (determined

by `feature\_importance.metric`) to drop before the algorithm decides to not expand the feature

group. This is only used for the `tree` algorithm.

""")

parser.add\_argument(

"--feature\_importance.dont\_build\_tree", dest="dont\_build\_tree", action="store\_true", default=False,

help="""

If True, don't build the feature trie for the tree algorithm and only use the extra\_groups

""")

parser.add\_argument(

"--feature\_importance.split\_feature\_group\_on\_period", dest="split\_feature\_group\_on\_period", action="store\_true", default=False,

help="If true, split feature groups by the period rather than the optimal prefix. Only used for the TREE algorithm")

parser.add\_argument(

"--feature\_importance.example\_count", dest="feature\_importance\_example\_count", type=int, default=10000,

help="""

The number of examples used to compute feature importance.

Larger values yield more reliable results, but also take longer to compute.

These records are loaded into memory. This number is agnostic to batch size.

""")

parser.add\_argument(

"--feature\_importance.data\_dir", dest="feature\_importance\_data\_dir", type=str, default=None,

help="Path to the dataset used to compute feature importance."

"supports local filesystem path and hdfs://default/<path> which requires "

"setting HDFS configuration via env variable HADOOP\_CONF\_DIR "

"Defaults to eval\_data\_dir")

parser.add\_argument(

"--feature\_importance.metric", dest="feature\_importance\_metric", type=str, default="roc\_auc",

help="The metric used to determine when to stop expanding the feature importance tree. This is only used for the `tree` algorithm.")

parser.add\_argument(

"--feature\_importance.is\_metric\_larger\_the\_better", dest="feature\_importance\_is\_metric\_larger\_the\_better", action="store\_true", default=False,

help="If true, interpret `--feature\_importance.metric` to be a metric where larger values are better (e.g. ROC\_AUC)")

parser.add\_argument(

"--feature\_importance.is\_metric\_smaller\_the\_better", dest="feature\_importance\_is\_metric\_smaller\_the\_better", action="store\_true", default=False,

help="If true, interpret `--feature\_importance.metric` to be a metric where smaller values are better (e.g. LOSS)")

subparsers = parser.add\_subparsers(help='Learning Rate Decay Functions. Can only pass 1.'

'Should be specified after all the optional arguments'

'and followed by its specific args'

'e.g. --learning\_rate 0.01 inverse\_learning\_rate\_decay\_fn'

' --decay\_rate 0.0004 --min\_learning\_rate 0.001',

dest='learning\_rate\_decay')

# Create the parser for the "exponential\_learning\_rate\_decay\_fn"

parser\_exponential = subparsers.add\_parser('exponential\_learning\_rate\_decay',

help='Exponential learning rate decay. '

'Exponential decay implements:'

'decayed\_learning\_rate = learning\_rate \* '

'exponential\_decay\_rate ^ '

'(global\_step / decay\_steps')

parser\_exponential.add\_argument(

"--decay\_steps", type=float, default=None,

help="Required for 'exponential' learning\_rate\_decay.")

parser\_exponential.add\_argument(

"--exponential\_decay\_rate", type=float, default=None,

help="Required for 'exponential' learning\_rate\_decay. Must be positive. ")

# Create the parser for the "polynomial\_learning\_rate\_decay\_fn"

parser\_polynomial = subparsers.add\_parser('polynomial\_learning\_rate\_decay',

help='Polynomial learning rate decay. '

'Polynomial decay implements: '

'global\_step = min(global\_step, decay\_steps)'

'decayed\_learning\_rate = '

'(learning\_rate - end\_learning\_rate) \* '

'(1 - global\_step / decay\_steps) ^ '

'(polynomial\_power) + end\_learning\_rate'

'So for linear decay you can use a '

'polynomial\_power=1 (the default)')

parser\_polynomial.add\_argument(

"--end\_learning\_rate", type=float, default=0.0001,

help="Required for 'polynomial' learning\_rate\_decay (ignored otherwise).")

parser\_polynomial.add\_argument(

"--polynomial\_power", type=float, default=0.0001,

help="Required for 'polynomial' learning\_rate\_decay."

"The power of the polynomial. Defaults to linear, 1.0.")

parser\_polynomial.add\_argument(

"--decay\_steps", type=float, default=None,

help="Required for 'polynomial' learning\_rate\_decay. ")

# Create the parser for the "piecewise\_constant\_learning\_rate\_decay\_fn"

parser\_piecewise\_constant = subparsers.add\_parser('piecewise\_constant\_learning\_rate\_decay',

help='Piecewise Constant '

'learning rate decay. '

'For piecewise\_constant, '

'consider this example: '

'We want to use a learning rate '

'that is 1.0 for'

'the first 100000 steps,'

'0.5 for steps 100001 to 110000, '

'and 0.1 for any additional steps. '

'To do so, specify '

'--piecewise\_constant\_boundaries=100000,110000'

'--piecewise\_constant\_values=1.0,0.5,0.1')

parser\_piecewise\_constant.add\_argument(

"--piecewise\_constant\_values",

action=parse\_comma\_separated\_list(element\_type=float),

default=None,

help="Required for 'piecewise\_constant\_values' learning\_rate\_decay. "

"A list of comma seperated floats or ints that specifies the values "

"for the intervals defined by boundaries. It should have one more "

"element than boundaries.")

parser\_piecewise\_constant.add\_argument(

"--piecewise\_constant\_boundaries",

action=parse\_comma\_separated\_list(element\_type=int),

default=None,

help="Required for 'piecewise\_constant\_values' learning\_rate\_decay. "

"A list of comma seperated integers, with strictly increasing entries.")

# Create the parser for the "inverse\_learning\_rate\_decay\_fn"

parser\_inverse = subparsers.add\_parser('inverse\_learning\_rate\_decay',

help='Inverse Leaning rate decay. '

'Inverse implements:'

'decayed\_lr = max(lr /(1 + decay\_rate \* '

'floor(global\_step /decay\_step)),'

' min\_learning\_rate)'

'When decay\_step=1 this mimics the behaviour'

'of the default learning rate decay'

'of DeepBird v1.')

parser\_inverse.add\_argument(

"--decay\_rate", type=float, default=None,

help="Required for 'inverse' learning\_rate\_decay. Rate in which we decay the learning rate.")

parser\_inverse.add\_argument(

"--min\_learning\_rate", type=float, default=None,

help="Required for 'inverse' learning\_rate\_decay.Minimum possible learning\_rate.")

parser\_inverse.add\_argument(

"--decay\_steps", type=float, default=1,

help="Required for 'inverse' learning\_rate\_decay.")

# Create the parser for the "cosine\_learning\_rate\_decay\_fn"

parser\_cosine = subparsers.add\_parser('cosine\_learning\_rate\_decay',

help='Cosine Leaning rate decay. '

'Cosine implements:'

'decayed\_lr = 0.5 \* (1 + cos(pi \*\

global\_step / decay\_steps)) \* lr'

)

parser\_cosine.add\_argument(

"--alpha", type=float, default=0,

help="A scalar float32 or float64 Tensor or a Python number.\

Minimum learning rate value as a fraction of learning\_rate.")

parser\_cosine.add\_argument(

"--decay\_steps", type=float,

help="Required for 'inverse' learning\_rate\_decay.")

# Create the parser for the "cosine\_restart\_learning\_rate\_decay\_fn"

parser\_cosine\_restart = subparsers.add\_parser('cosine\_restarts\_learning\_rate\_decay',

help='Applies cosine decay with restarts \

to the learning rate'

'See [Loshchilov & Hutter, ICLR2016],\

SGDR: Stochastic'

'Gradient Descent with Warm Restarts.'

'https://arxiv.org/abs/1608.03983'

)

parser\_cosine\_restart.add\_argument(

"--first\_decay\_steps", type=float,

help="Required for 'cosine\_restart' learning\_rate\_decay.")

parser\_cosine\_restart.add\_argument(

"--alpha", type=float, default=0,

help="A scalar float32 or float64 Tensor or a Python number. \

Minimum learning rate value as a fraction of learning\_rate.")

parser\_cosine\_restart.add\_argument(

"--t\_mul", type=float, default=2,

help="A scalar float32 or float64 Tensor or a Python number. \

Used to derive the number of iterations in the i-th period")

parser\_cosine\_restart.add\_argument(

"--m\_mul", type=float, default=1,

help="A scalar float32 or float64 Tensor or a Python number. \

Used to derive the initial learning rate of the i-th period.")

# Create dummy parser for None, which is the default.

parser\_default = subparsers.add\_parser(

'no\_learning\_rate\_decay',

help='No learning rate decay') # noqa: F841

parser.set\_default\_subparser('no\_learning\_rate\_decay')

return parser

class DefaultSubcommandArgParse(argparse.ArgumentParser):

"""

Subclass of argparse.ArgumentParser that sets default parser

"""

\_DEFAULT\_SUBPARSER = None

def set\_default\_subparser(self, name):

"""

sets the default subparser

"""

self.\_DEFAULT\_SUBPARSER = name

def \_parse\_known\_args(self, arg\_strings, \*args, \*\*kwargs):

"""

Overwrites \_parse\_known\_args

"""

in\_args = set(arg\_strings)

d\_sp = self.\_DEFAULT\_SUBPARSER

if d\_sp is not None and not {'-h', '--help'}.intersection(in\_args):

for x\_val in self.\_subparsers.\_actions:

subparser\_found = (

isinstance(x\_val, argparse.\_SubParsersAction) and

in\_args.intersection(x\_val.\_name\_parser\_map.keys())

)

if subparser\_found:

break

else:

# insert default in first position, this implies no

# global options without a sub\_parsers specified

arg\_strings = arg\_strings + [d\_sp]

return super(DefaultSubcommandArgParse, self).\_parse\_known\_args(

arg\_strings, \*args, \*\*kwargs

)

def \_check\_value(self, action, value):

try:

super(DefaultSubcommandArgParse, self).\_check\_value(

action, value

)

except ArgumentError as error:

error.message += ("\nERROR: Deepbird is trying to interpret \"{}\" as a value of {}. If this is not what you expected, "

"then most likely one of the following two things are happening: Either one of your cli arguments are not recognized, "

"probably {} or whichever argument you are passing {} as a value to OR you are passing in an argument after "

"the `learning\_rate\_decay` argument.\n").format(value, action.dest, value, value)

raise error

def parse\_comma\_separated\_list(element\_type=str):

"""

Generates an argparse.Action that converts a string representing a comma separated list to a

list and converts each element to a specified type.

"""

# pylint: disable-msg=too-few-public-methods

class \_ParseCommaSeparatedList(argparse.Action):

"""

Converts a string representing a comma separated list to a list and converts each element to a

specified type.

"""

def \_\_call\_\_(self, parser, namespace, values, option\_string=None):

if values is not None:

values = [element\_type(v) for v in values.split(',')]

setattr(namespace, self.dest, values)

return \_ParseCommaSeparatedList