# pylint: disable=no-member,arguments-differ, attribute-defined-outside-init

"""

Implementing Full Dense Layer

"""

from tensorflow.python.layers import core as core\_layers

from tensorflow.python.ops import init\_ops

from tensorflow.python.framework import tensor\_shape

from tensorflow.python.keras.engine.base\_layer import InputSpec

import tensorflow.compat.v1 as tf

class FullDense(core\_layers.Dense):

"""

Densely-connected layer class.

This is wrapping tensorflow.python.layers.core.Dense

This layer implements the operation:

.. code-block:: python

outputs = activation(inputs.weight + bias)

Where ``activation`` is the activation function passed as the ``activation``

argument (if not ``None``), ``weight`` is a weights matrix created by the layer,

and ``bias`` is a bias vector created by the layer.

Arguments:

output\_size:

Integer or Long, dimensionality of the output space.

activation:

Activation function (callable). Set it to None to maintain a linear activation.

weight\_initializer:

Initializer function for the weight matrix.

bias\_initializer:

Initializer function for the bias.

weight\_regularizer:

Regularizer function for the weight matrix.

Ensure to add tf.losses.get\_regularization\_loss() to your loss for this to take effect.

bias\_regularizer:

Regularizer function for the bias.

Ensure to add tf.losses.get\_regularization\_loss() to your loss for this to take effect.

activity\_regularizer:

Regularizer function for the output.

weight\_constraint:

An optional projection function to be applied to the

weight after being updated by an `Optimizer` (e.g. used to implement

norm constraints or value constraints for layer weights). The function

must take as input the unprojected variable and must return the

projected variable (which must have the same shape). Constraints are

not safe to use when doing asynchronous distributed training.

bias\_constraint:

An optional projection function to be applied to the

bias after being updated by an `Optimizer`.

trainable:

Boolean, if `True` also add variables to the graph collection

``GraphKeys.TRAINABLE\_VARIABLES`` (see `tf.Variable

<https://www.tensorflow.org/versions/master/api\_docs/python/tf/Variable>`\_).

name:

String, the name of the layer. Layers with the same name will

share weights, but to avoid mistakes we require ``reuse=True`` in such cases.

Properties:

output\_size:

Python integer, dimensionality of the output space.

activation:

Activation function (callable).

weight\_initializer:

Initializer instance (or name) for the weight matrix.

bias\_initializer:

Initializer instance (or name) for the bias.

weight:

Weight matrix (TensorFlow variable or tensor). (weight)

bias:

Bias vector, if applicable (TensorFlow variable or tensor).

weight\_regularizer:

Regularizer instance for the weight matrix (callable)

bias\_regularizer:

Regularizer instance for the bias (callable).

activity\_regularizer:

Regularizer instance for the output (callable)

weight\_constraint:

Constraint function for the weight matrix.

bias\_constraint:

Constraint function for the bias.

"""

def \_\_init\_\_(self, output\_size,

weight\_initializer=None,

weight\_regularizer=None,

weight\_constraint=None,

bias\_constraint=None,

num\_partitions=None,

\*\*kwargs):

super(FullDense, self).\_\_init\_\_(units=output\_size,

kernel\_initializer=weight\_initializer,

kernel\_regularizer=weight\_regularizer,

kernel\_constraint=weight\_constraint,

\*\*kwargs)

self.\_num\_partitions = num\_partitions

def build(self, input\_shape):

'''

code adapted from TF 1.12 Keras Dense layer:

https://github.com/tensorflow/tensorflow/blob/r1.12/tensorflow/python/keras/layers/core.py#L930-L956

'''

input\_shape = tensor\_shape.TensorShape(input\_shape)

if input\_shape[-1] is None:

raise ValueError('The last dimension of the inputs to `Dense` '

'should be defined. Found `None`.')

self.input\_spec = InputSpec(min\_ndim=2,

axes={-1: input\_shape[-1]})

partitioner = None

if self.\_num\_partitions:

partitioner = tf.fixed\_size\_partitioner(self.\_num\_partitions)

self.kernel = self.add\_weight(

'kernel',

shape=[input\_shape[-1], self.units],

initializer=self.kernel\_initializer,

regularizer=self.kernel\_regularizer,

constraint=self.kernel\_constraint,

dtype=self.dtype,

partitioner=partitioner,

trainable=True)

if self.use\_bias:

self.bias = self.add\_weight(

'bias',

shape=[self.units, ],

initializer=self.bias\_initializer,

regularizer=self.bias\_regularizer,

constraint=self.bias\_constraint,

dtype=self.dtype,

trainable=True)

else:

self.bias = None

self.built = True

@property

def output\_size(self):

"""

Returns output\_size

"""

return self.units

@property

def weight(self):

"""

Returns weight

"""

return self.kernel

@property

def weight\_regularizer(self):

"""

Returns weight\_regularizer

"""

return self.kernel\_regularizer

@property

def weight\_initializer(self):

"""

Returns weight\_initializer

"""

return self.kernel\_initializer

@property

def weight\_constraint(self):

"""

Returns weight\_constraint

"""

return self.kernel\_constraint

def full\_dense(inputs, output\_size,

activation=None,

use\_bias=True,

weight\_initializer=None,

bias\_initializer=init\_ops.zeros\_initializer(),

weight\_regularizer=None,

bias\_regularizer=None,

activity\_regularizer=None,

weight\_constraint=None,

bias\_constraint=None,

trainable=True,

name=None,

num\_partitions=None,

reuse=None):

"""Functional interface for the densely-connected layer.

This layer implements the operation:

`outputs = activation(inputs.weight + bias)`

Where `activation` is the activation function passed as the `activation`

argument (if not `None`), `weight` is a weights matrix created by the layer,

and `bias` is a bias vector created by the layer

(only if `use\_bias` is `True`).

Arguments:

inputs: Tensor input.

units: Integer or Long, dimensionality of the output space.

activation: Activation function (callable). Set it to None to maintain a

linear activation.

use\_bias: Boolean, whether the layer uses a bias.

weight\_initializer: Initializer function for the weight matrix.

If `None` (default), weights are initialized using the default

initializer used by `tf.get\_variable`.

bias\_initializer:

Initializer function for the bias.

weight\_regularizer:

Regularizer function for the weight matrix.

Ensure to add tf.losses.get\_regularization\_loss() to your loss for this to take effect.

bias\_regularizer:

Regularizer function for the bias.

Ensure to add tf.losses.get\_regularization\_loss() to your loss for this to take effect.

activity\_regularizer:

Regularizer function for the output.

weight\_constraint:

An optional projection function to be applied to the

weight after being updated by an `Optimizer` (e.g. used to implement

norm constraints or value constraints for layer weights). The function

must take as input the unprojected variable and must return the

projected variable (which must have the same shape). Constraints are

not safe to use when doing asynchronous distributed training.

bias\_constraint:

An optional projection function to be applied to the

bias after being updated by an `Optimizer`.

trainable:

Boolean, if `True` also add variables to the graph collection

`GraphKeys.TRAINABLE\_VARIABLES` (see `tf.Variable`).

name:

String, the name of the layer.

reuse:

Boolean, whether to reuse the weights of a previous layer

by the same name.

Returns:

Output tensor the same shape as `inputs` except the last dimension is of

size `units`.

Raises:

ValueError: if eager execution is enabled.

"""

layer = FullDense(output\_size,

activation=activation,

use\_bias=use\_bias,

weight\_initializer=weight\_initializer,

bias\_initializer=bias\_initializer,

weight\_regularizer=weight\_regularizer,

bias\_regularizer=bias\_regularizer,

activity\_regularizer=activity\_regularizer,

weight\_constraint=weight\_constraint,

bias\_constraint=bias\_constraint,

trainable=trainable,

name=name,

dtype=inputs.dtype.base\_dtype,

num\_partitions=num\_partitions,

\_scope=name,

\_reuse=reuse)

return layer.apply(inputs)