Externally add a Variable to a model in TensorFlow

Asked 7 months ago Modified 7 months ago Viewed 258 times

0

I have an existing TensorFlow model, and I want to add a new "parameter" (a `tf.Variable`) to the model's list of parameters (such that it's trainable) and add it externally to the model's list of parameters / computational graph.

One approach that I tried, is to append the new parameters to the model's list of trainable weights, something like this (here `new_parameter` is a `tf.Variable`) -

```
model.layers[-1].trainable_weights.extend([new_parameter])
model.compile(....)
```

But I'm not sure if that's the best way to go about it. In PyTorch, we have `nn.Parameter` instead of `tf.Variable` , and we have [`register_parameter`]() using which we can register tensors as new parameter's to the model's list of parameters. Is there any equivalent of `register_parameter` in TensorFlow? Or some other way to achieve the same goal?

1

It is possible by custom dense layer too ‼

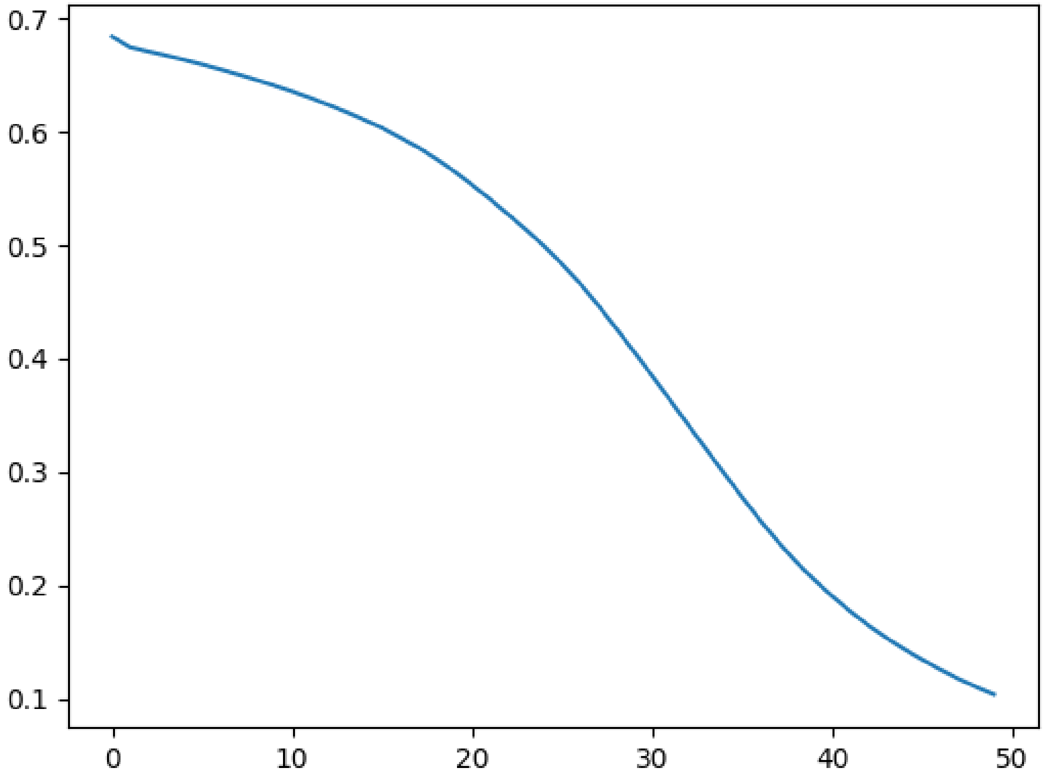
[Sample]:

```
import os
from os.path import exists

import tensorflow as tf
import tensorflow_io as tfio

import matplotlib.pyplot as plt
import numpy as np

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""
[PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
None
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""
physical_devices = tf.config.experimental.list_physical_devices('GPU')
assert len(physical_devices) > 0, "Not enough GPU hardware devices available"
config = tf.config.experimental.set_memory_growth(physical_devices[0], True)
print(physical_devices)
print(config)

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""
: Variables
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""
PATH = os.path.join('F:\\datasets\\downloads\\Actors\\train\\Pikaploy',
'*.tif')
PATH_2 = os.path.join('F:\\datasets\\downloads\\Actors\\train\\Candidt Kibt',
'*.tif')
files = tf.data.Dataset.list_files(PATH)
files_2 = tf.data.Dataset.list_files(PATH_2)

list_file = []
list_file_actual = []
```

```
list_label = []
list_label_actual = [ 'Pikaploy', 'Pikaploy', 'Pikaploy', 'Pikaploy',
'Pikaploy', 'Candidt Kibt', 'Candidt Kibt', 'Candidt Kibt', 'Candidt Kibt',
'Candidt Kibt' ]
for file in files.take(15):
    image = tf.io.read_file( file )
    image = tfio.experimental.image.decode_tiff(image, index=0)
    list_file_actual.append(image)
    image = tf.image.resize(image, [32,32], method='nearest')
    list_file.append(image)
    list_label.append(1)

for file in files_2.take(18):
    image = tf.io.read_file( file )
    image = tfio.experimental.image.decode_tiff(image, index=0)
    list_file_actual.append(image)
    image = tf.image.resize(image, [32,32], method='nearest')
    list_file.append(image)
    list_label.append(9)

checkpoint_path = "F:\\models\\checkpoint\\" +
os.path.basename(__file__).split('.')[0] + "\\TF_DataSets_01.h5"
checkpoint_dir = os.path.dirname(checkpoint_path)
loggings = "F:\\models\\checkpoint\\" + os.path.basename(__file__).split('.')
[0] + "\\loggings.log"

if not exists(checkpoint_dir) :
    os.mkdir(checkpoint_dir)
    print("Create directory: " + checkpoint_dir)

log_dir = checkpoint_dir

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""
: Class / Function
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""
class MyDenseLayer(tf.keras.layers.Layer):
    def __init__(self, num_outputs, num_add):
        super(MyDenseLayer, self).__init__()
        self.num_outputs = num_outputs
        self.num_add = num_add

    def build(self, input_shape):
        self.kernel = self.add_weight("kernel",
        shape=[int(input_shape[-1]),
        self.num_outputs])

    def call(self, inputs):
        temp = tf.add( inputs, self.num_add )
        temp = tf.matmul(temp, self.kernel)
        return temp

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""
DataSet
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""
list_file = tf.cast( list_file, dtype=tf.int64 )
list_file = tf.constant( list_file, shape=(33, 1, 32, 32, 4), dtype=tf.int64)
list_label = tf.cast( list_label, dtype=tf.int64 )
list_label = tf.constant( list_label, shape=(33, 1, 1, 1), dtype=tf.int64)

dataset = tf.data.Dataset.from_tensor_slices(( list_file, list_label ))

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""
: Model Initialize
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""
model = tf.keras.models.Sequential([
    tf.keras.layers.InputLayer(input_shape=( 32, 32, 4 )),
    tf.keras.layers.Normalization(mean=3., variance=2.),
    tf.keras.layers.Normalization(mean=4., variance=6.),
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Reshape((128, 225)),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(96,
return_sequences=True, return_state=False)),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(96)),
])

layer = MyDenseLayer(10, 5)

model.add(layer)
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(192, activation='relu'))
model.add(tf.keras.layers.Dense(10, activation='softmax'))
model.summary()

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""
: Optimizer
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""
optimizer = tf.keras.optimizers.Nadam(
    learning_rate=0.00001, beta_1=0.9, beta_2=0.999, epsilon=1e-07,
    name='Nadam'
)

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""
: Loss Fn
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""
lossfn = tf.keras.losses.SparseCategoricalCrossentropy(
    from_logits=False,
    reduction=tf.keras.losses.Reduction.AUTO,
    name='sparse_categorical_crossentropy'
)

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""
: Model Summary
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""
model.compile(optimizer=optimizer, loss=lossfn, metrics=['accuracy'])

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""
: FileWriter
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""
if exists(checkpoint_path) :
    model.load_weights(checkpoint_path)
    print("model load: " + checkpoint_path)
    input("Press Any Key!")

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""
: Training
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""
history = model.fit( dataset, batch_size=100, epochs=50 )
model.save_weights(checkpoint_path)


plt.plot(history.history['loss'])
plt.show()
plt.close()

input("...")
```

[Output]:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 normalization (Normalizatio  (None, 32, 32, 4)        0
 n)

 normalization_1 (Normalizat  (None, 32, 32, 4)        0
 ion)

 conv2d (Conv2D)             (None, 30, 30, 32)        1184

 max_pooling2d (MaxPooling2D  (None, 15, 15, 32)       0
 )

 dense (Dense)               (None, 15, 15, 128)       4224

 reshape (Reshape)           (None, 128, 225)          0

 bidirectional (Bidirectiona  (None, 128, 192)         247296
 l)

 bidirectional_1 (Bidirectio  (None, 192)              221952
 nal)

 my_dense_layer (MyDenseLaye  (None, 10)               1920
 r) ***  💬 custom layer added

 flatten (Flatten)           (None, 10)                0

 dense_1 (Dense)             (None, 192)               2112

 dense_2 (Dense)             (None, 10)                1930

=================================================================
Total params: 480,618
Trainable params: 480,618
Non-trainable params: 0
_________________________________________________________________
```


0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
10
20
30
40
50