TensorNetworks in Neural Networks.

Here, we have a small toy example of how to use a TN inside of a fully connected neural network.

First off, let's install tensornetwork

```
# !pip install tensornetwork

import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
# Import tensornetwork
import tensornetwork as tn
import random
import time
# Set the backend to tesorflow
# (default is numpy)
tn.set_default_backend("tensorflow")
np.random.seed(42)
random.seed(42)
tf.random.set_seed(42)
```

TensorNetwork layer definition

Here, we define the TensorNetwork layer we wish to use to replace the fully connected layer. Here, we simply use a 2 node Matrix Product Operator network to replace the normal dense weight matrix.

We TensorNetwork's NCon API to keep the code short.

```
class TNLayer(tf.keras.layers.Layer):

  def __init__(self):
    super(TNLayer, self).__init__()
    # Create the variables for the layer.
    self.a_var = tf.Variable(tf.random.normal(shape=(32, 32, 2),
                                              stddev=1.0/32.0),
                             name="a", trainable=True)
    self.b_var = tf.Variable(tf.random.normal(shape=(32, 32, 2),
                                              stddev=1.0/32.0),
                             name="b", trainable=True)
    self.bias = tf.Variable(tf.zeros(shape=(32, 32)),
                            name="bias", trainable=True)

  def call(self, inputs):
    # Define the contraction.
    # We break it out so we can parallelize a batch using
    # tf.vectorized_map (see below).
    def f(input_vec, a_var, b_var, bias_var):
      # Reshape to a matrix instead of a vector.
      input_vec = tf.reshape(input_vec, (32, 32))

      # Now we create the network.
      a = tn.Node(a_var)
      b = tn.Node(b_var)
      x_node = tn.Node(input_vec)
      a[1] ^ x_node[0]
      b[1] ^ x_node[1]
      a[2] ^ b[2]

      # The TN should now look like this
      #   |     |
      #   a --- b
      #    \   /
      #      x

      # Now we begin the contraction.
      c = a @ x_node
      result = (c @ b).tensor

      # To make the code shorter, we also could've used Ncon.
      # The above few lines of code is the same as this:
      # result = tn.ncon([x, a_var, b_var], [[1, 2], [-1, 1, 3], [-2, 2, 3]])

      # Finally, add bias.
      return result + bias_var

    # To deal with a batch of items, we can use the tf.vectorized_map
    # function.
    # https://www.tensorflow.org/api_docs/python/tf/vectorized_map
```

```
    result = tf.vectorized_map(
        lambda vec: f(vec, self.a_var, self.b_var, self.bias), inputs)
    return tf.nn.relu(tf.reshape(result, (-1, 1024)))
```

Smaller model

These two models are effectively the same, but notice how the TN layer has nearly 10x fewer parameters.

```
Dense = tf.keras.layers.Dense
fc_model = tf.keras.Sequential(
    [
     tf.keras.Input(shape=(2,)),
     Dense(1024, activation=tf.nn.relu),
     Dense(1024, activation=tf.nn.relu),
     Dense(1, activation=None)])
fc_model.summary()
```

```
Model: "sequential_40"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_100 (Dense)           (None, 1024)              3072

 dense_101 (Dense)           (None, 1024)              1049600

 dense_102 (Dense)           (None, 1)                 1025

=================================================================
Total params: 1053697 (4.02 MB)
Trainable params: 1053697 (4.02 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
tn_model = tf.keras.Sequential(
    [
     tf.keras.Input(shape=(2,)),
     Dense(1024, activation=tf.nn.relu),
     # Here, we replace the dense layer with our MPS.
     TNLayer(),
     TNLayer(),
     Dense(1, activation=None)])
tn_model.summary()
```

```
Model: "sequential_41"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_103 (Dense)           (None, 1024)              3072

 tn_layer_22 (TNLayer)       (None, 1024)              5120

 tn_layer_23 (TNLayer)       (None, 1024)              5120

 dense_104 (Dense)           (None, 1)                 1025

=================================================================
Total params: 14337 (56.00 KB)
Trainable params: 14337 (56.00 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Training a model

You can train the TN model just as you would a normal neural network model! Here, we give an example of how to do it in Keras.

```
X = np.concatenate([np.random.randn(20, 2) + np.array([3, 3]),
                    np.random.randn(20, 2) + np.array([-3, -3]),
                    np.random.randn(20, 2) + np.array([-3, 3]),
                    np.random.randn(20, 2) + np.array([3, -3])])

Y = np.concatenate([np.ones((40)), -np.ones((40))])
```

```
seconds = time.time()
print("Time in seconds since beginning of run:", seconds)
local_time = time.ctime(seconds)
print(local_time)
```

```
Time in seconds since beginning of run: 1709522004.4777458
Mon Mar  4 03:13:24 2024
```

```
tn_model.compile(optimizer="adam", loss="mean_squared_error")
tn_model.fit(X, Y, epochs=300, verbose=2)
```

```
3/3 - 0s - loss: 8.3169e-07 - 21ms/epoch - 7ms/step
Epoch 247/300
3/3 - 0s - loss: 8.7212e-07 - 17ms/epoch - 6ms/step
Epoch 248/300
3/3 - 0s - loss: 8.8973e-07 - 25ms/epoch - 8ms/step
Epoch 249/300
3/3 - 0s - loss: 9.4720e-07 - 18ms/epoch - 6ms/step
Epoch 250/300
3/3 - 0s - loss: 7.6689e-07 - 20ms/epoch - 7ms/step
Epoch 251/300
3/3 - 0s - loss: 7.4134e-07 - 18ms/epoch - 6ms/step
Epoch 252/300
3/3 - 0s - loss: 8.4110e-07 - 20ms/epoch - 7ms/step
Epoch 253/300
3/3 - 0s - loss: 7.7027e-07 - 22ms/epoch - 7ms/step
Epoch 254/300
3/3 - 0s - loss: 7.3369e-07 - 20ms/epoch - 7ms/step
Epoch 255/300
3/3 - 0s - loss: 7.1947e-07 - 21ms/epoch - 7ms/step
Epoch 256/300
3/3 - 0s - loss: 6.8687e-07 - 20ms/epoch - 7ms/step
Epoch 257/300
3/3 - 0s - loss: 6.8526e-07 - 18ms/epoch - 6ms/step
Epoch 258/300
3/3 - 0s - loss: 7.2039e-07 - 21ms/epoch - 7ms/step
Epoch 259/300
3/3 - 0s - loss: 8.6763e-07 - 17ms/epoch - 6ms/step
Epoch 260/300
3/3 - 0s - loss: 5.9998e-07 - 18ms/epoch - 6ms/step
Epoch 261/300
3/3 - 0s - loss: 8.0895e-07 - 19ms/epoch - 6ms/step
Epoch 262/300
3/3 - 0s - loss: 6.3816e-07 - 20ms/epoch - 7ms/step
Epoch 263/300
3/3 - 0s - loss: 6.7886e-07 - 18ms/epoch - 6ms/step
Epoch 264/300
3/3 - 0s - loss: 6.4195e-07 - 20ms/epoch - 7ms/step
Epoch 265/300
3/3 - 0s - loss: 6.7303e-07 - 18ms/epoch - 6ms/step
Epoch 266/300
3/3 - 0s - loss: 6.7099e-07 - 15ms/epoch - 5ms/step
Epoch 267/300
3/3 - 0s - loss: 5.9744e-07 - 18ms/epoch - 6ms/step
Epoch 268/300
3/3 - 0s - loss: 6.4065e-07 - 22ms/epoch - 7ms/step
Epoch 269/300
3/3 - 0s - loss: 7.0460e-07 - 19ms/epoch - 6ms/step
Epoch 270/300
3/3 - 0s - loss: 5.6204e-07 - 20ms/epoch - 7ms/step
Epoch 271/300
3/3 - 0s - loss: 5.7983e-07 - 22ms/epoch - 7ms/step
Epoch 272/300
3/3 - 0s - loss: 5.8025e-07 - 23ms/epoch - 8ms/step
Epoch 273/300
3/3 - 0s - loss: 6.5557e-07 - 23ms/epoch - 8ms/step
Epoch 274/300
3/3 - 0s - loss: 6.3053e-07 - 18ms/epoch - 6ms/step
Epoch 275/300
3/3 - 0s - loss: 5.9187e-07 - 18ms/epoch - 6ms/step
```

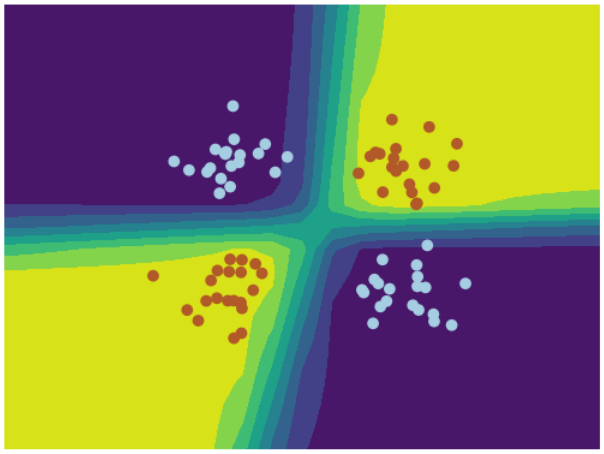
```
# Plotting code, feel free to ignore.
h = 1.0
x_min, x_max = X[:, 0].min() - 5, X[:, 0].max() + 5
y_min, y_max = X[:, 1].min() - 5, X[:, 1].max() + 5
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                     np.arange(y_min, y_max, h))

# here "model" is your model's prediction (classification) function
Z = tn_model.predict(np.c_[xx.ravel(), yy.ravel()])

# Put the result into a color plot
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z)
plt.axis('off')

# Plot also the training points
plt.scatter(X[:, 0], X[:, 1], c=Y, cmap=plt.cm.Paired)
```

```
14/14 [==============================] - 0s 4ms/step
<matplotlib.collections.PathCollection at 0x7cd65250f100>
```



```
seconds = time.time()
print("Time in seconds since end of run:", seconds)
local_time = time.ctime(seconds)
print(local_time)
```

```
Time in seconds since end of run: 1709522013.90653
Mon Mar  4 03:13:33 2024
```

```
seconds = time.time()
print("Time in seconds since beginning of run:", seconds)
local_time = time.ctime(seconds)
print(local_time)
```

```
Time in seconds since beginning of run: 1709522013.918321
Mon Mar  4 03:13:33 2024
```

VS Fully Connected

```
fc_model.compile(optimizer="adam", loss="mean_squared_error")
fc_model.fit(X, Y, epochs=300, verbose=2)
# Plotting code, feel free to ignore.
h = 1.0
x_min, x_max = X[:, 0].min() - 5, X[:, 0].max() + 5
y_min, y_max = X[:, 1].min() - 5, X[:, 1].max() + 5
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                     np.arange(y_min, y_max, h))

# here "model" is your model's prediction (classification) function
Z = fc_model.predict(np.c_[xx.ravel(), yy.ravel()])

# Put the result into a color plot
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z)
plt.axis('off')

# Plot also the training points
plt.scatter(X[:, 0], X[:, 1], c=Y, cmap=plt.cm.Paired)
```