C3W3_Siamese_Network

July 28, 2025

1 Creating a Siamese model: Ungraded Lecture Notebook

In this notebook you will learn how to create a siamese model in TensorFlow.

[1]:
```
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras.models import Model, Sequential
from tensorflow import math
import numpy

# Setting random seeds
numpy.random.seed(10)
```

```
2025-07-21 14:26:45.356219: I tensorflow/tsl/cuda/cudart_stub.cc:28] Could not
find cuda drivers on your machine, GPU will not be used.
2025-07-21 14:26:45.508624: I tensorflow/tsl/cuda/cudart_stub.cc:28] Could not
find cuda drivers on your machine, GPU will not be used.
2025-07-21 14:26:45.510224: I tensorflow/core/platform/cpu_feature_guard.cc:182]
This TensorFlow binary is optimized to use available CPU instructions in
performance-critical operations.
To enable the following instructions: AVX2 AVX512F FMA, in other operations,
rebuild TensorFlow with the appropriate compiler flags.
2025-07-21 14:26:46.802500: W
tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not
find TensorRT
```

1.1 Siamese Model

To create a `Siamese` model you will first need to create a LSTM model. For this you can stack layers using the`Sequential` model. To retrieve the output of both branches of the Siamese model, you can concatenate results using the `Concatenate` layer. You should be familiar with the following layers (notice each layer can be clicked to go to the docs): - `Sequential` groups a linear stack of layers into a `tf.keras.Model` - `Embedding` Maps positive integers into vectors of fixed size. It will have shape (vocabulary length X dimension of output vectors). The dimension of output vectors (called `model_dimension`in the code) is the number of elements in the word embedding. - `LSTM` The Long Short-Term Memory (LSTM) layer. The number of units should be specified and should match the number of elements in the word embedding. - `GlobalAveragePooling1D` Computes global average pooling, which essentially takes the mean across a desired axis. GlobalAveragePooling1D uses one tensor axis to form groups of values and replaces each group with the mean value of that group. -

`Lambda` Layer with no weights that applies the function f, which should be specified using a lambda syntax. You will use this layer to apply normalization with the function - `tfmath.l2_normalize(x)`

- `Concatenate` Layer that concatenates a list of inputs. This layer will concatenate the normalized outputs of each LSTM into a single output for the model.
- `Input`: it is used to instantiate a Keras tensor.. Remember to set correctly the dimension and type of the input, which are batches of questions.

Putting everything together the Siamese model will look like this:

[2]:
```
vocab_size = 500
model_dimension = 128

# Define the LSTM model
LSTM = Sequential()
LSTM.add(layers.Embedding(input_dim=vocab_size, output_dim=model_dimension))
LSTM.add(layers.LSTM(units=model_dimension, return_sequences = True))
LSTM.add(layers.AveragePooling1D())
LSTM.add(layers.Lambda(lambda x: math.l2_normalize(x)))

input1 = layers.Input((None,))
input2 = layers.Input((None,))

# Concatenate two LSTMs together
conc = layers.Concatenate(axis=1)((LSTM(input1), LSTM(input2)))


# Use the Parallel combinator to create a Siamese model out of the LSTM
Siamese = Model(inputs=(input1, input2), outputs=conc)

# Print the summary of the model
Siamese.summary()
```

```
Model: "model"
__________________________________________________________________________________
_________________
 Layer (type)                Output Shape              Param #   Connected to
==================================================================================
=================
 input_1 (InputLayer)        [(None, None)]            0         []

 input_2 (InputLayer)        [(None, None)]            0         []

 sequential (Sequential)     (None, None, 128)         195584
['input_1[0][0]',
'input_2[0][0]']

 concatenate (Concatenate)   (None, None, 128)         0
['sequential[0][0]',
```

```
'sequential[1][0]']

==================================================================================
==================
Total params: 195584 (764.00 KB)
Trainable params: 195584 (764.00 KB)
Non-trainable params: 0 (0.00 Byte)
----------------------------------------------------------------------------------
------------------
```

Next is a helper function that prints information for every layer:

[3]:
```
def show_layers(model, layer_prefix):
    print(f"Total layers: {len(model.layers)}\n")
    for i in range(len(model.layers)):
        print('========')
        print(f'{layer_prefix}_{i}: {model.layers[i]}\n')

print('Siamese model:\n')
show_layers(Siamese, 'Parallel.sublayers')

print('Detail of LSTM models:\n')
show_layers(LSTM, 'Serial.sublayers')
```

```
Siamese model:

Total layers: 4

========
Parallel.sublayers_0: <keras.src.engine.input_layer.InputLayer object at
0x74dedc0db220>

========
Parallel.sublayers_1: <keras.src.engine.input_layer.InputLayer object at
0x74dedc0dadd0>

========
Parallel.sublayers_2: <keras.src.engine.sequential.Sequential object at
0x74dedfbcf730>

========
Parallel.sublayers_3: <keras.src.layers.merging.concatenate.Concatenate object
at 0x74dedc0d8a30>

Detail of LSTM models:

Total layers: 4
```

```
========
Serial.sublayers_0: <keras.src.layers.core.embedding.Embedding object at
0x74dedfbcdb70>

========
Serial.sublayers_1: <keras.src.layers.rnn.lstm.LSTM object at 0x74dedfbce5c0>

========
Serial.sublayers_2: <keras.src.layers.pooling.average_pooling1d.AveragePooling1D
object at 0x74dedc0db940>

========
Serial.sublayers_3: <keras.src.layers.core.lambda_layer.Lambda object at
0x74dedc0db190>
```

Try changing the parameters defined before the Siamese model and see how it changes!

You will actually train this model in this week's assignment. For now you should be more familiarized with creating Siamese models using TensorFlow. **Keep it up!**

[ ]: