C3_W4_lecture_nb_1_siamese

December 17, 2023

1 Creating a Siamese model using Trax: Ungraded Lecture Notebook

[ ]:
```
import trax
from trax import layers as tl
import trax.fastmath.numpy as np
import numpy

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

1.1 L2 Normalization

Before building the model you will need to define a function that applies L2 normalization to a tensor. This is very important because in this week's assignment you will create a custom loss function which expects the tensors it receives to be normalized. Luckily this is pretty straightforward:

[ ]:
```
def normalize(x):
    return x / np.sqrt(np.sum(x * x, axis=-1, keepdims=True))
```

Notice that the denominator can be replaced by `np.linalg.norm(x, axis=-1, keepdims=True)` to achieve the same results and that Trax's numpy is being used within the function.

[ ]:
```
tensor = numpy.random.random((2,5))
print(f'The tensor is of type: {type(tensor)}\n\nAnd looks like this:\n\n␣
 ↪{tensor}')
```

[ ]:
```
norm_tensor = normalize(tensor)
print(f'The normalized tensor is of type: {type(norm_tensor)}\n\nAnd looks like␣
 ↪this:\n\n {norm_tensor}')
```

Notice that the initial tensor was converted from a numpy array to a jax array in the process.

1.2 Siamese Model

To create a `Siamese` model you will first need to create a LSTM model using the `Serial` combinator layer and then use another combinator layer called `Parallel` to create the Siamese model. You should be familiar with the following layers (notice each layer can be clicked to go to the docs): - `Serial` A combinator layer that allows to stack layers serially using function composition. - `Embedding` Maps discrete tokens to vectors. It will have shape `(vocabulary length X dimension of output vectors)`. The dimension of output vectors (also called `d_feature`) is the number of elements in the word embedding. - `LSTM` The LSTM layer. It leverages another Trax layer called `LSTMCell`. The number of units should be specified and should match the number of elements in the word embedding. - `Mean` Computes the mean across a desired axis. Mean uses one tensor axis to form groups of values and replaces each group with the mean value of that group. - `Fn` Layer with no weights that applies the function f, which should be specified using a lambda syntax. - `Parallel` It is a combinator layer (like `Serial`) that applies a list of layers in parallel to its inputs.

Putting everything together the Siamese model will look like this:

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

# Define the LSTM model
LSTM = tl.Serial(
        tl.Embedding(vocab_size=vocab_size, d_feature=model_dimension),
        tl.LSTM(model_dimension),
        tl.Mean(axis=1),
        tl.Fn('Normalize', lambda x: normalize(x))
    )

# Use the Parallel combinator to create a Siamese model out of the LSTM
Siamese = tl.Parallel(LSTM, LSTM)
```

Next is a helper function that prints information for every layer (sublayer within `Serial`):

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

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

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

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 Trax. **Keep it up!**