problem-shared-attention

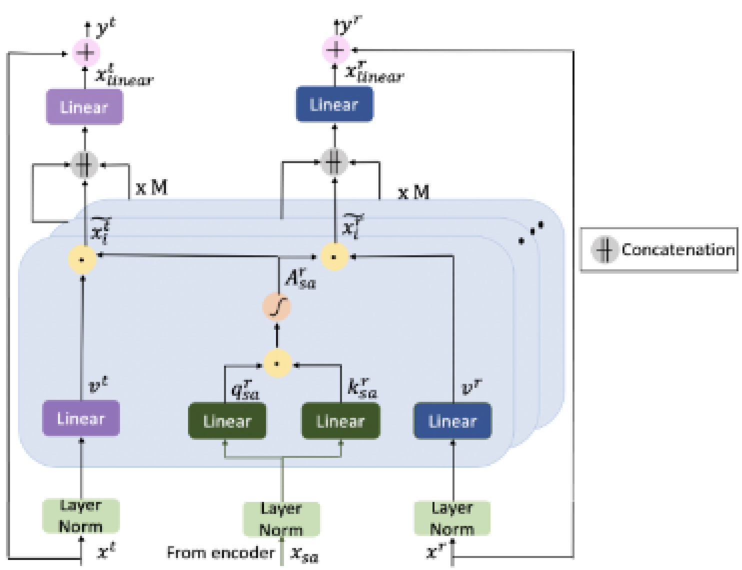
November 14, 2022

1 TDT17 2022-P3-T2-ST1 task

Shared attention is a concept used in the MulT architecture in order for a multi-task network to learn between-task dependencies. In MulT, all tasks share the same encoders, while deploying task-specific decoders. The shared attention blocks are introduced in each decoder block, and this task will have you calculate the outputs for a shared attention block on an input that has been simplified to suit the time constraint of the TDT17 tasks format. The goal of this task is not to interpret how the output is affected by a specific inputs (especially since the linear layers are stochastically generated here), but instead to gain an insight in what separates shared attention from self-attetion. A single task t is considered, with an additional reference task r. The details of what these tasks are is not important.

1.1 Given:

- The shared attention block from MulT



- A skip connection x_s from the shared encoder
- The upsampled output x^t from the previous decoder stage
- The upsampled output x^r from the previous decoder stage of some "reference" task
- M = 6 attention heads
- Helper functions to generate matrices and vectors

1.2 Find:

- The dimensions for the projections in each attention head
- The dimensions for the projections of the multi-head attention output
- The outputs y^t and y^r of the shared attention block
- Answer: Which modifications have been made vs. "standard" self-attention?
- Answer: Where does the shared attention block utilize information from other tasks?

Note: Wherever a learnable parameter is used in MulT, it is fine simply replacing it with something randomly generated :)

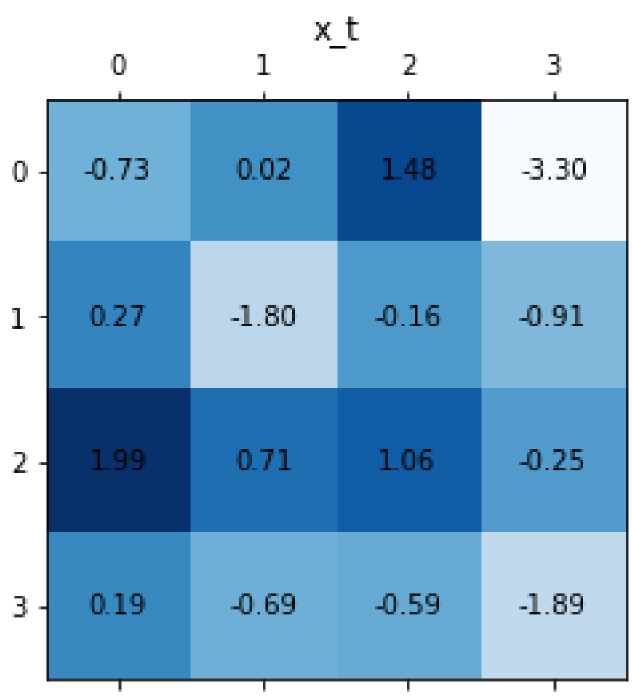
2 Helper functions

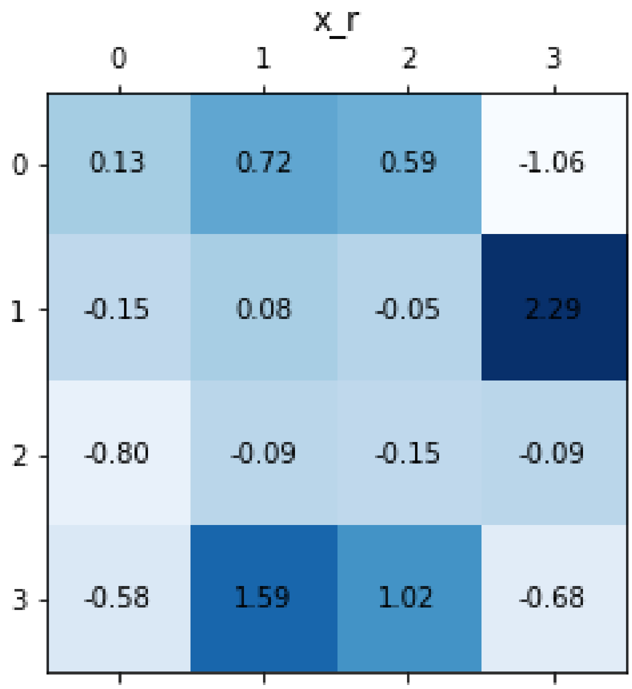
[10]:
```python
import numpy as np
import matplotlib.pyplot as plt

def softmax(x, axis=0):
    return np.exp(x) / np.sum(np.exp(x), axis=axis)

def normalize(x):
    return (x - x.mean()) / x.std()

def linear(n, m):
    """
    Generate an nxm array of normally distributed values
    """
    return np.random.normal(0, 1, n * m).reshape((n, m))

def x(n):
    """
    Generate a nx1 vector of normally distributed values.
    """
    return np.random.normal(0, 1, n).reshape((n, 1))


def draw(data, title: str = "") -> None:
        """Display a visualization of the matrix values

        See: https://stackoverflow.com/questions/40887753/
 ↪display-matrix-values-and-colormap

        Assumes data on the form nx1 where n is a perfect square number
        """
        fig, ax = plt.subplots()

        n = int(np.sqrt(data.shape[0]))
        data = data.reshape((n, n))
```

```
        ax.matshow(data, cmap=plt.cm.Blues)
        ax.set_title(title)

        for col, row in np.ndindex(data.shape):
            ax.text(col, row, f"{data[row, col]:.2f}", va='center', ha='center')
```

3 Given variables

[11]:
```
M = 6

H = 4
W = 4

ndim = H * W

x_t = x(ndim)
x_s = x(ndim)
x_r = x(ndim)

Cr = 1 # Number of channels

draw(x_t, "x_t")
draw(x_r, "x_r")
```





4 Your solution

[12]:
```
# Hint: The most straight-forward solution will likely be to use a loop over
# range(M) and calculate the single-head shared attention. These can be
# concatenated and projected to obtain the linear outputs. Lastly add the skip
# connection from the input to the output.
#
# Use normalize() to normalize the inputs, linear(m, n) to generate the linear
# layers, and draw() to visualize your output in the end.
```

5 Questions

1. Which modifications have been made vs. “standard” self-attention?

TODO

2. Where does the shared attention block utilize information from other tasks?

TODO