TEK5040 Assignment, Meta Learning

Narada Warakagoda

September 28, 2020

NOTE: This assignment is optional. Therefore, no submission is required. But you are strongly encouraged to perform the tasks in this assignment

1 Introduction

This assignment deals with meta learning with neural networks. More specifically it is centered around the implementation of matching networks published in *Matching Networks for One Shot Learning, Oriol Vinyals et.al. 2016.*
The code is based on *https://github.com/cnichkawde/MatchingNetwork*

1.1 Package Contents

`data.npy`	Data samples from Omniglot database
`datanway.py`	Python script providing training and testing data
`matchingnetwork.py`	Main Python/Tensorflow script implementing the matching network
`matchnn.py`	Python/Tensorflow script implementing the cosine distance based matching
`exercise.pdf`	This document describing the task

1.2 Operation

Thee Python/Tensorflow scripts have been provided: `datanway.py`, `matchingnetwork.py` and `matchnn.py`. You are not required to study `datanway.py` as it will only provide training and testing data from a data set called *Omniglot*. You can run the code by issuing the command:

```
python3 matchingnetwork.py
```

If you experience memory problems try to reduce the database size (`trainsize` and `valsize`) in line 14 of `matchingnetwork.py`.

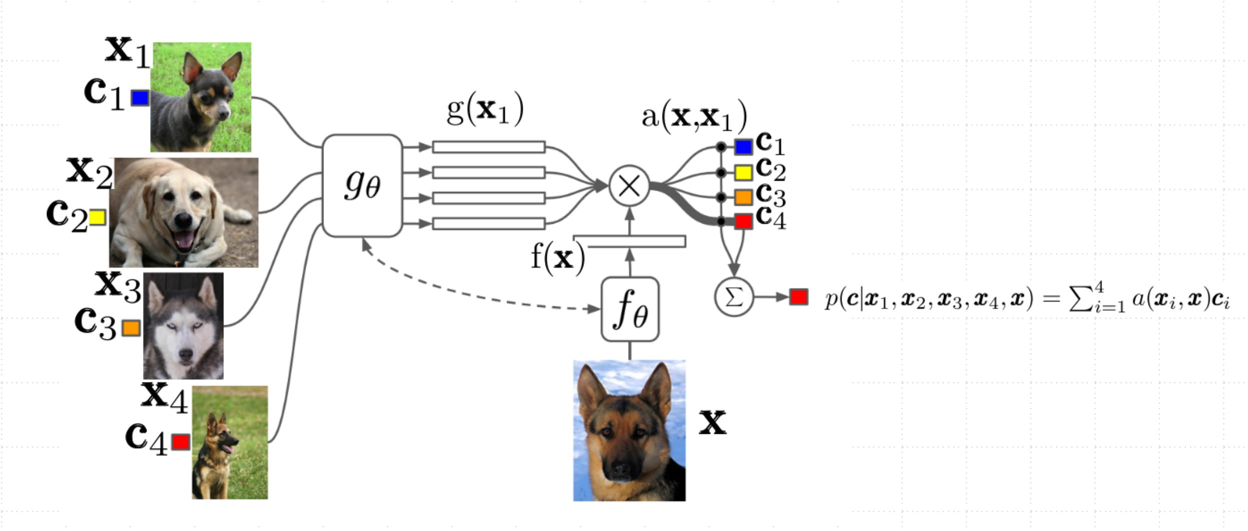

Figure 1: Matching network

If the script works correctly, you should see the typical Keras output of training and validation losses and it should run 10 epochs.

In the file `matchingnetwork.py`, lines 17-63, the meta-learning model (matching network) is defined. Line 64 compiles the model, whereas line 79 fits the model to data, according to the standard practise in Keras.

Lines 68-75 prints out the shapes of the data used for training and validation. You can take a look at the shapes and try to understand the data structures being fed to the model in training and validation (i.e. arguments to the function call `model.fit`).

In file `matchnn.py`, a custom layer (`MatchCosine`) is implemented by extending the base class `tensorflow.keras.layers.Layer`. This layer performs cosine distance based matching between support classes and target classes.

You are required to study the code in `matchingnetwork.py` and `matchnn.py` with reference to the matching network given in Figure 1 and the procedure outlined below (for simplicity we have removed the batch dimension in the procedure).

- Training: Use a *meta-training set* $\{D_{train}, D_{test}\}$
    1. $D_{train} = \{(\boldsymbol{x}_1, \boldsymbol{c}_1), (\boldsymbol{x}_2, \boldsymbol{c}_2), (\boldsymbol{x}_3, \boldsymbol{c}_3), (\boldsymbol{x}_4, \boldsymbol{c}_4), (\boldsymbol{x}_5, \boldsymbol{c}_5)\}$
    2. $D_{test} = \{(\boldsymbol{x}, c)\}$
    3. Compute $g_\theta(\boldsymbol{x}_i), \quad i = 1, 2, 3, 4, 5$
    4. Compute $f_\theta(\boldsymbol{x})$

5. Compute $d_i = d_{\cos}(g_\theta(\boldsymbol{x}_i), f_\theta(\boldsymbol{x})) = \frac{g_\theta^T(\boldsymbol{x}_i) f_\theta(\boldsymbol{x})}{|g_\theta(\boldsymbol{x}_i)||f_\theta(\boldsymbol{x})|}, \quad i = 1, 2, 3, 4, 5$
6. Compute $a(\boldsymbol{x}_i, \boldsymbol{x}) = d_i / \sum_j d_j, \quad i = 1, 2, 3, 4, 5$
7. Compute $\boldsymbol{p} = [p(C = 1), p(C = 2), p(C = 3), p(C = 4), p(C = 5)]^T = \sum_{i=1}^{5} a(\boldsymbol{x}_i, \boldsymbol{x}) \boldsymbol{c}_i$

2 Task

1. Which data structures in the code contain the meta training set described by points 1 and 2 in the training procedure above? Be specific as much as possible.

2. Write the filename/line numbers of the code corresponding to each of the steps 3-7 above. Name which Tensorflow APIs are used in each case and explain briefly how they are used.

3. Write an expression for the loss function used in the code with a description of the symbols used.

4. Why is the *sparse categorical loss* function used in the code instead of the *categorical loss* function?

5. In lines 54 and 55, two *Lambda* layers are used. What is the difference between these two? What is the main use of *Lambda* layers?

6. Functions $g_\theta(\cdot)$ and $f_\theta(\cdot)$ share the same weights in this implementation. Explain how this weight sharing has been achieved.