# Mixture of Experts - Experiment 1

1. Architecture

In this experiment, I will implement a structure of neural networks for a classification task. I will describe here about the datasets first, then the structure implemented.

The data for classification is the combination of mnist and cifar-10. Both datasets have images with 1 of 10 labels, so the combined dataset will have 20 classes in total, where the first 10 classes are from mnist and the second 10 classes are from cifar-10. Images from both datasets are resized to 32x32 and gray-scaled. Let X be the tensor of an image, Y be the corresponding true label, Y' be the corresponding prediction resulted by the neural network.

In this first experiment, I try to compare a mixture of 2 experts against 2 distinct experts, where each is given the input only from mnist or only from cifar-10. The compared models share this equation:

$$Y'(X) = softmax\big(Gate(X)_1 * NN_1(X) + Gate(X)_2 * NN_2(X)\big),$$

Where $Gate(X)$ is a probability distribution vector of length 2, its subscript denotes the value of the corresponding term, and $NN_1$ and $NN_2$ denotes sub- neural networks, which output vectors of length 20 of given the image.

In this experiment, sub- neural networks will have the same structure for all subscripts in both baseline (the disctinct experts) and the MoE (mixture of experts) -structures. The structure is defined in python-tensorflow as following:

```
layer = X
layer = tf.layers.conv2d(layer, 128, (3, 3), padding='same', activation=tf.nn.relu)
layer = tf.layers.max_pooling2d(layer, (2, 2), (2, 2), padding='same')
layer = tf.layers.conv2d(layer, 128, (3, 3), padding='same', activation=tf.nn.relu)
layer = tf.layers.max_pooling2d(layer, (2, 2), (2, 2), padding='same')
layer = tf.layers.flatten(layer)
layer = tf.layers.dense(layer, 128, tf.nn.relu)
layer = tf.layers.dense(layer, 20)
Y_Logits = layer
```

Or briefly, it is a combination of convolution layers, max-pooling layers and fully connected layers. In principle, sub- neural networks could have different structures.

The $Gate$ functions varies in baseline and MoE-structures. For the baseline, I define the gate function as such: $Gate(X) = IF\ IsMnist(X)\ THEN\ [1,0]\ ELSE\ [0,1]$. The function IsMnist gives the ground truth result if the image is from the mnist dataset. This should guarantee, that the gate function is fixed in baseline, the sub- neural networks are distinct. For the MoE-structures, I define here 2 gate functions for 2 test cases: a convolutional gate and a dense gate. The implementation of the convolutional gate looks like this:

```
layer = X
layer = tf.layers.conv2d(layer,64,(3,3),padding='same')
layer = tf.layers.max_pooling2d(layer, (2, 2), (2, 2), padding='same')
layer = tf.layers.conv2d(layer,64,(3,3),padding='same')
layer = tf.layers.max_pooling2d(layer, (2, 2), (2, 2), padding='same')
layer = tf.layers.flatten(layer)
layer = tf.layers.dense(layer, 64, tf.nn.relu)
layer = tf.layers.dense(layer, 2, tf.nn.softmax)
gate = layer
```

And the other one looks like this:

```
layer = X
layer = tf.layers.flatten(layer)
layer = tf.layers.dense(layer, 64, tf.nn.relu)
layer = tf.layers.dense(layer, 2, tf.nn.softmax)
gate = layer
```

2. Research questions
   a.
      As baseline varies from MoE-structures in gate functions. The gate for baseline specializes each sub- neural network for the corresponding dataset. Classification should be simpler knowing which subset of classes the input belongs to. Therefore, the baseline should give an upperbound of accuracy of MoE-structures.
      Plot the accuracy of both training and testing for baseline, convolutional gate and dense gate (2*3=6 curves) for comparison.

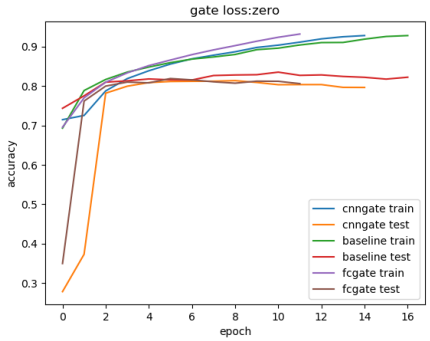
      The accuracies are measured as following:
      Training accuracy is cumulatively measured before the gradient is applied for each batch.
      Test accuracy is measured when the epoch is finished.
   b.
      Plot a table with axis standing for [Mnist, Cifar]x[NN1, NN2] and contents for activation rate.
      The activation rate is defined as following:

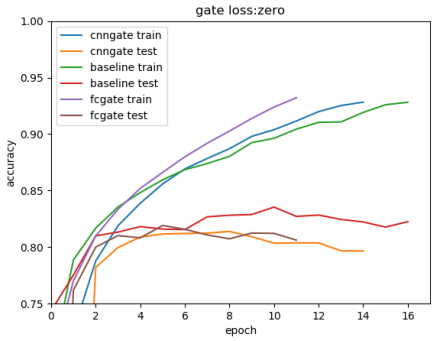
      $$ActivationRate(dataset, NN_i) = Mean[Gate(X)_i | X\ in\ dataset]$$

      For numbers not close to 0, corresponding NN is not ignored for corresponding dataset.
      If for some NN, the sum is close to 0, then the NN is ignored for all dataset.
      The ideal result looks like the identity matrix so, that each NN is specialized to a dataset.

3. Results

a. Here are a plot of the epoch-accuracy curve and a zoomed version below.




Both MoE-structures underperform on testing.

However, they seem to overperform on training.

b.

| | trainNN0 | trainNN1 | | testNN0 | testNN1 |
|---|---|---|---|---|---|
| cnngatemnist | 1.0 | 1.13724546e-29 | | 1.0 | 2.9693655e-30 |
| cnngatecifar | 1.0 | 1.9429693e-31 | | 1.0 | 7.551104e-37 |
| | | | | | |
| baselinemnist | 1.0 | 0.0 | | 1.0 | 0.0 |
| baselinecifar | 0.0 | 1.0 | | 0.0 | 1.0 |
| | | | | | |
| fcgatemnist | 0.5692206 | 0.43077937 | | 0.5669755 | 0.43302447 |
| fcgatecifar | 0.0038144253 | 0.9961853 | | 0.0036498767 | 0.99635 |

Table of the baseline is correct.

NN1 with convolutional gate is ignored.

NN0 with dense gate is specialized on mnist, but NN1 takes both mnist and cifar data.

4. Possible improvement
   a. Different learning rate or learning strategy for Gate and NN.
   b. Additional gate loss for penalizing ignored NN or balancing between different NNs.
   c. Repeat more experiments. Test on more variations for Gate and NN with configurations of different complexity.
   d. Measure activation rate for each class, test on a mixture of more experts. Plot larger table for it (hard to look up?).
5. Additional works done

Since the analysis based on 3b shows, that in many cases some NNs are ignored. The most reliable way to solve the problem and move on is implementing some ideas about 4b.

Here I describe the things I implemented and plot things to report.

5.1 Implemented

The idea of 4b is about an additional loss, which somehow regularize the gate. In the earlier experiment, the total loss to be minimized is the classification loss, which is the cross-entropy of $Y$ and $Y'$. To have an additional loss, is just to have such equation:

$$Loss_{total} = Loss_{classification} + Loss_{gate} = CrossEntropy(Y, Y') + Loss_{gate}$$

The solve the problem, the activation rates (defined in 2b) must be increased. Toward the ideal result, it is also needed, that for an image from either dataset, has a specialized network. Here, I experimented on two similar losses:

$$Loss_{gate} = Loss_{entropy} = \alpha Loss_{batch-entropy} + \beta Loss_{sample-entropy}$$

and

$$Loss_{gate} = Loss_{variance} = \alpha Loss_{batch-variance} + \beta Loss_{sample-variance}$$

Let's begin with $\text{Loss}_{entropy}$. First, we define the entropy of a probability distribution vector $\boldsymbol{p}$:

$$Entropy(\boldsymbol{p}) = -\sum_i \boldsymbol{p}_i \ln(\boldsymbol{p}_i)$$

As a training step is ran on a batch of images, the $\boldsymbol{X}$ is denoted as the batch:

$$Loss_{batch-entropy}(\boldsymbol{X}) = -Entropy\left(\frac{1}{n}\sum_i Gate(\boldsymbol{X}_i)\right),$$

Where $\frac{1}{n}\sum_i Gate(\boldsymbol{X}_i)$ is a probabilistic distribution vector and is mean of the gate among all images, $n$ is the batch-size, and the minus-sign in the beginning is for the case, that we want to maximize the batch-entropy. As the batch-entropy is away from the minima 0, the sub- neural networks are not ignored.

The sample-entropy is computed element wise:

$$Loss_{sample-entropy}(\boldsymbol{X}) = \frac{1}{n}\sum_i Entropy\big(Gate(\boldsymbol{X}_i)\big)$$

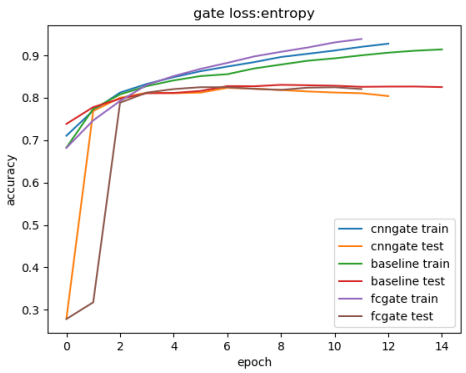
It is positive to encourage an image to use a single sub- neural network.

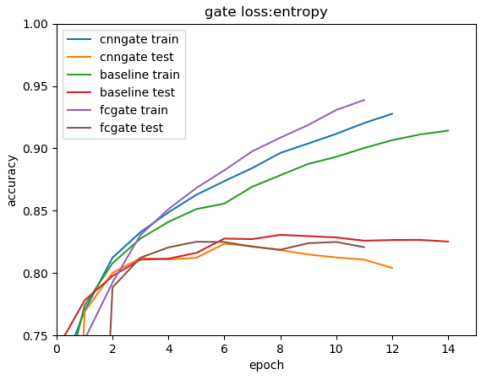
Variance loss is defined similarly, using variance instead of entropy.

5.2 report

Just to specify, $\alpha = 0.1$, $\beta = 0.01$ and $n = 128$.

The plots for entropy-loss:

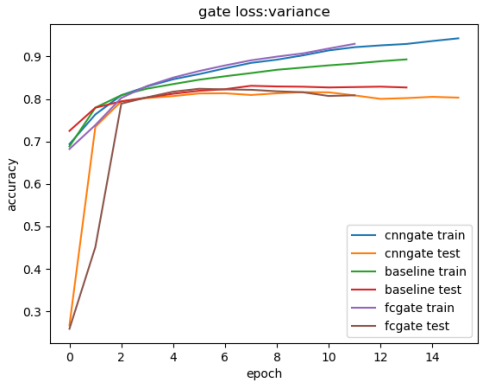


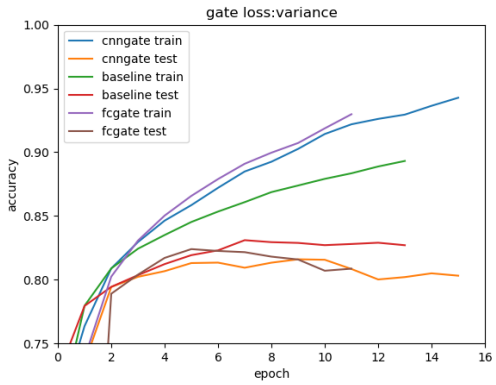




| | trainNN0 | trainNN1 | | testNN0 | testNN1 |
|---|---|---|---|---|---|
| cnngatemnist | 3.3892565e-12 | 1.0 | | 2.801041e-12 | 1.0 |
| cnngatecifar | 1.4379872e-13 | 1.0 | | 1.7602685e-14 | 1.0 |
| baselinemnist | 1.0 | 0.0 | | 1.0 | 0.0 |
| baselinecifar | 0.0 | 1.0 | | 0.0 | 1.0 |
| fcgatemnist | 0.5294198 | 0.47058028 | | 0.52429134 | 0.4757087 |
| fcgatecifar | 0.676661 | 0.32333913 | | 0.67688125 | 0.32311872 |

For both MoE, the batch-entropies are not close to its minima [0.5,0.5]. It has small to no effect on more complex convolutional gate, but it seems to have significant effect on simpler dense gate.

With same configuration, the variance loss did not achieve similar results. Only 1 sub- neural network is activated:




| | trainNN0 | trainNN1 | | testNN0 | testNN1 |
|---|---|---|---|---|---|
| cnngatemnist | 2.3545892e-12 | 1.0 | | 8.3334733e-13 | 1.0 |
| cnngatecifar | 1.5829963e-09 | 1.0 | | 1.9886076e-10 | 1.0 |
| baselinemnist | 1.0 | 0.0 | | 1.0 | 0.0 |
| baselinecifar | 0.0 | 1.0 | | 0.0 | 1.0 |
| fcgatemnist | 1.0 | 2.3551654e-07 | | 0.9999998 | 2.1355557e-07 |
| fcgatecifar | 0.99999994 | 6.1346135e-08 | | 1.0 | 1.0147267e-08 |

There are things to mention:

a. The gate-table does not tell if a single image is specialized on a sub- neural network.
b. MoE would never overperform the baseline, if the gate is to separate mnist-cifar.
c. Both batch-entropy and batch-variance encourages balanced separations.