# Redistribution of data

In order to remedy the effect of imbalanced training set, we designed an algorithm that equally balance each data according to their labels. For each label, we pick EpL(Elements per Label) amount of data corresponding to that label stochastically. This also alleviates the effect of under-fitting by providing the neural network more amount of training data when chosen large enough EpL.

algorithm balanceData (T, L, EpL )

-Input-

T: Training data

L: Label for training data

EpL: Elements per Label

-Output-

B: Balanced set of data with size EpL \* number of labels. There are EpL number of data that has same label.

1. Initialize B with size EpL \* number of labels.

2. for each label in L as iL

3. repeat EpL times

4. select a random element R from T that has label iL

5. append the index of R from T into the array B.

# Dropout

Dropout is a method to relieve overfitting. In the feed-forwarding part of the neural network, the layer unit is stochastically dropped out to 0.

algorithm dropout(M, R)

-Input-

M: Neural Network model

R: Dropout rate

-Output-

M: Neural Network model with each layer stochastically dropped out

1. for each layer in M.layer except the final layer

2. for each unit in layer

3. Make the unit value 0 with rate R

# Neural Network

For algorithm to train SVHN dataset, we used neural network. The neural network consists of feedforwarding, back propagation, and weight update. For the hyper-parameters we used for this neural network,

Learning rate: 0.005

Number of hidden layers: 3

Number of units in hidden layers: [512, 256 512]

Batch size: 1

Elements per label: 700

algorithm trainNeuralNetwork(T, L)

-Input-

T: Training data

L: Label for training data

-Output-

M: Trained Neural Network model

// Initialize the model

1. Initialize M.layer, M.weight M.bias, M.epoch

2. repeat i = 1:size of T

3. M = Feedforward(M, T(i))

4. updateVal = Backpropagate(M, L(i))

5. M = WeightUpdate(M, updateVal)

algorithm Feedforward(M, inputData)

-Input-

M: Neural Network model

inputData: sample of data

-Output-

M: Neural Network model with updated layer values

1. for each layer from the second layer in M.layer

2. update the layer unit value by M.layer(layer) <- M.layer(layer -1) \* M.weight(layer) + M.bias(layer)

// For this project, we used tanh function for activation function.

3. activate M.layer with activation function

algorithm Backpropagate(M, labelData)

-Input-

M: Neural Network model feedforwarded

labelData: label for feedforwarded

-Output-

updateVal: layer gradient

1. Initialize updateVal with size of M.weight

2. dLdOut = M.layer(Last layer) - labelData

3. for index\_layer 1 through (number of layers -1)

4. updateVal = dLdOut \* derivative of activation function

5. for weight\_iteration (number of layers - index\_layer + 2) through (number of layers)

6. updateVal = M.weight(weight\_iteration)' \* updateVal

7. updateVal = updateVal .\* (1 - M.weight(weight\_iteration-1).^2)

8. updateVal = updateVal \* M.layer(number of layers - index\_layer)'

algorithm WeightUpdate(M, updateVal, R)

-Input-

M: Neural Network model

updateVal: layer gradient

R: Learning rate

-Output-

M: Updated Neural Network

1. for each layer from second layer in M.layer

2. M.weight(layer) = M.weight(layer) - updateVal(layer) \* R