Q1.1 Theory

For index i in vector $x = \begin{bmatrix} x_1 \\ \cdot\cdot \\ x_j \end{bmatrix}$:

$$softmax(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

If we have translation c:

$$x + c = \begin{bmatrix} x_1 + c \\ \cdot\cdot \\ x_j + c \end{bmatrix} \rightarrow softmax(x_i + c) = \frac{e^{x_i+c}}{\sum_j e^{x_j+c}} = \frac{e^c e^{x_i}}{e^c \sum_j e^{x_j}} = \frac{e^{x_i}}{\sum_j e^{x_j}} = softmax(x_i)$$

By subtracting each number from $\max x_i$, all of the numbers in vector x are going to be less than zero which causes the e^{x_i} to be between 0 and 1 which confines the range of the results and prevent from exploding numbers in numerator.

Q1.2 Theory

- The range of each number is between zero and one and sum of all elements is one.
- Softmax takes an arbitrary real valued vector x and turns it into a probability distribution which higher the number was in the original vector would have a higher value in the transformed probability distribution which increases the chance of being chosen among other numbers.
- $s_i = e^{x_i}$ transfers the number into a positive space.

  $S = \sum s_i$ We use the sum of the exponential values as a normalizer.

  $softmax(x_i) = \frac{1}{S} s_i$ Computing the probability of each of the values.

Q1.3 Theory

Consider having X as input vector of network, W as weights for the first hidden layer and also B as the bias for the first hidden layer. We propagate forward and calculate the input to the nodes of first hidden layer before activation. (X_b= XW+B) . Considering the activation function as linear which output ax+b having the input of x, the output of the activation function would be:

$$X_o = a(X_b) + b = a(XW + B) + b = aXW + aB + b = X(aW) + (aB + b) = XW' + B'$$

As can be seen above, by choosing a linear activation function, defining several hidden layer would not have any effect and the whole network can be simplified and be defined in a one-layer network which is equivalent to linear regression on the input data.

Q1.4 Theory

$$\boldsymbol{\sigma(x) = \frac{1}{1 + e^{-x}}}$$

$$\boldsymbol{\frac{\delta\sigma(x)}{\delta x} = \frac{e^{-x}}{(1 + e^{-x})^2} = \frac{e^{-x}}{(1 + e^{-x})} \frac{1}{(1 + e^{-x})} = \frac{1 + e^{-x} - 1}{(1 + e^{-x})} \frac{1}{(1 + e^{-x})}}$$

$$\boldsymbol{= \left(1 - \frac{1}{(1 + e^{-x})}\right)\left(\frac{1}{(1 + e^{-x})}\right) = (1 - \sigma(x))\sigma(x)}$$

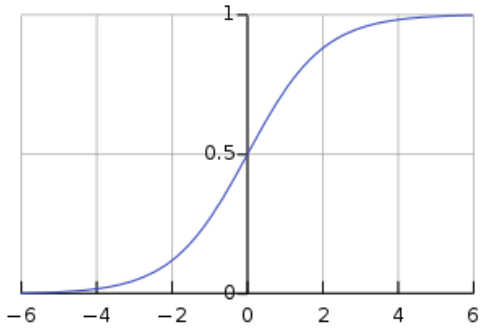
Q1.5 Theory

We have $\frac{\delta J}{\delta y} = \boldsymbol{\delta}$:

$$\frac{\delta J}{\delta W} = \frac{\delta J}{\delta y}\frac{\delta y}{\delta W} = \boldsymbol{x\delta^T},\ \frac{\delta J}{\delta x} = \frac{\delta J}{\delta y}\frac{\delta y}{\delta x} = \boldsymbol{W\delta},\ \frac{\delta J}{\delta b} = \frac{\delta J}{\delta y}\frac{\delta y}{\delta b} = \boldsymbol{\delta}$$

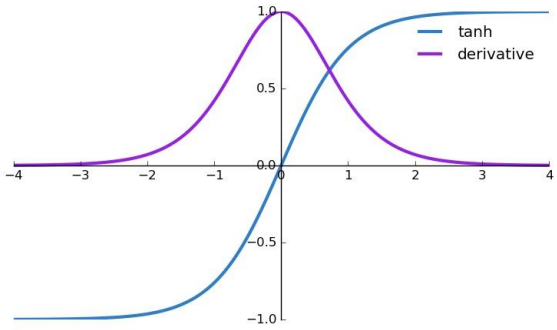
Q1.6 Theory

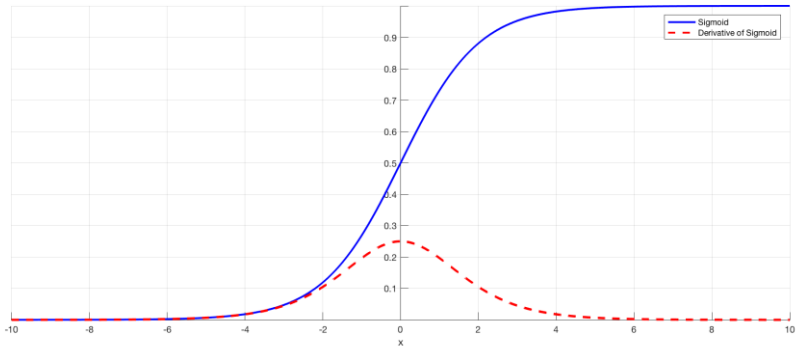
1. Based on the graph below when the sigmoid function value is either too high or too low, the derivative and also change in derivatives is very small which cause low change in the output even for large inputs. By propagating through network, the gradient becomes smaller and smaller which is called vanishing gradient.



2. The output range of sigmoid is between zero and one.
   The output range of tanh(x) is between -1 and 1.
   tanh works better than sigmoid which it maps the inputs to the range of [-1,1] which also includes negative values based on the input given and it has a lager range of values and the gradients are stronger which cause more rapid convergence.

3. Based on the graphs below and comparing the derivative of sigmoid and tanh function, it can be seen that the derivative of tanh has a larger range of values compared to sigmoid in the same range of x. ([-1,1]) which results in less vanishing gradient problem.





4.

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

$$tanh(x) = \frac{1-e^{-2x}}{1+e^{-2x}} \rightarrow tanh\left(\frac{x}{2}\right) = \frac{1-e^{-x}}{1+e^{-x}}$$

$$tanh\left(\frac{x}{2}\right) + 1 = \frac{1-e^{-x}}{1+e^{-x}} + 1 \rightarrow tanh\left(\frac{x}{2}\right) + 1 = \frac{1-e^{-x}+1+e^{-x}}{1+e^{-x}} = \frac{2}{1+e^{-x}} = 2\sigma(x)$$

$$tanh\left(\frac{x}{2}\right) + 1 = 2\sigma(x) \rightarrow tanh\left(\frac{x}{2}\right) = 2\sigma(x) - 1 \rightarrow tanh(x) = 2\sigma(2x) - 1$$

Q2.1.1

If we initialize all of the weights equal to zero, we are making the network independent of the inputs and the output would be not dependent on the input nodes which leads to preventing the network from learning.

Q2.1.3

In order to break the symmetry in the network and avoid all of the nodes to do the same thing and increase the chance f finding global optima, the weights are initialized randomly. If the weights are different and are assigned randomly, the output is going to be depended on the input and in the backpropagation, will learn.

In order to prevent vanishing and exploding gradient, the variance of the input and output of layers are set equal, so we depend the initialization based on the layer size.

Q3.1.2

By setting the batch_size=10 and learning_rate=0.008, the following graphs are calculated.

Fig 1: Learning_rate = 0.008

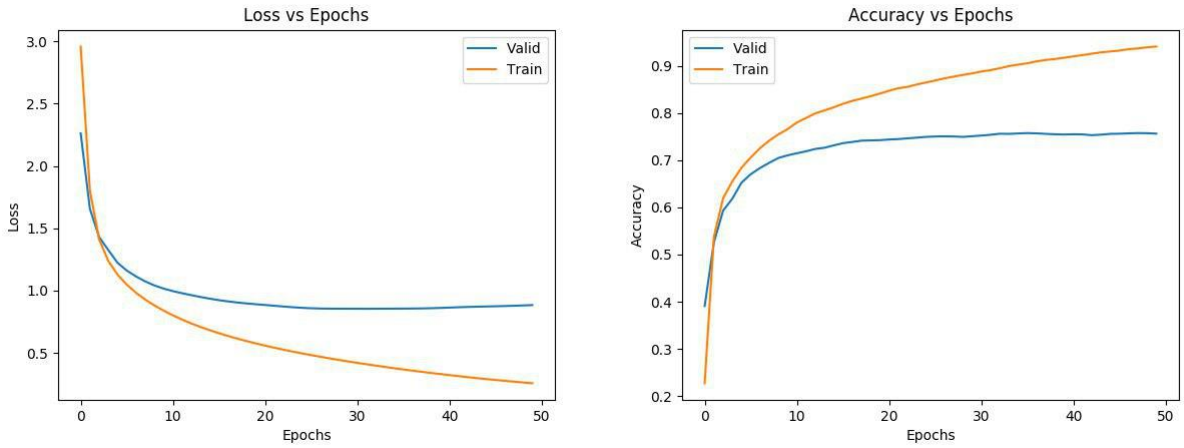


Fig 2:Learning_rate = 0.08

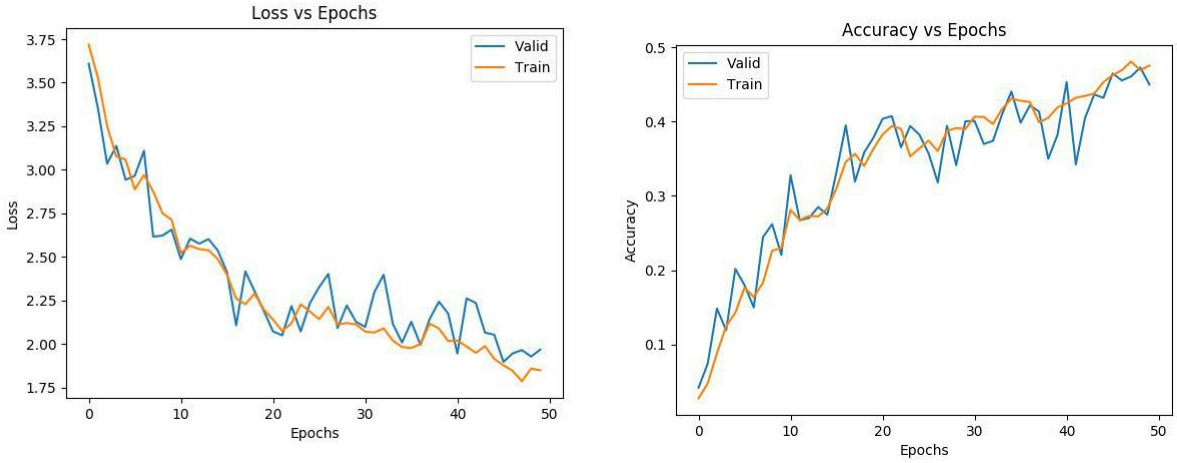
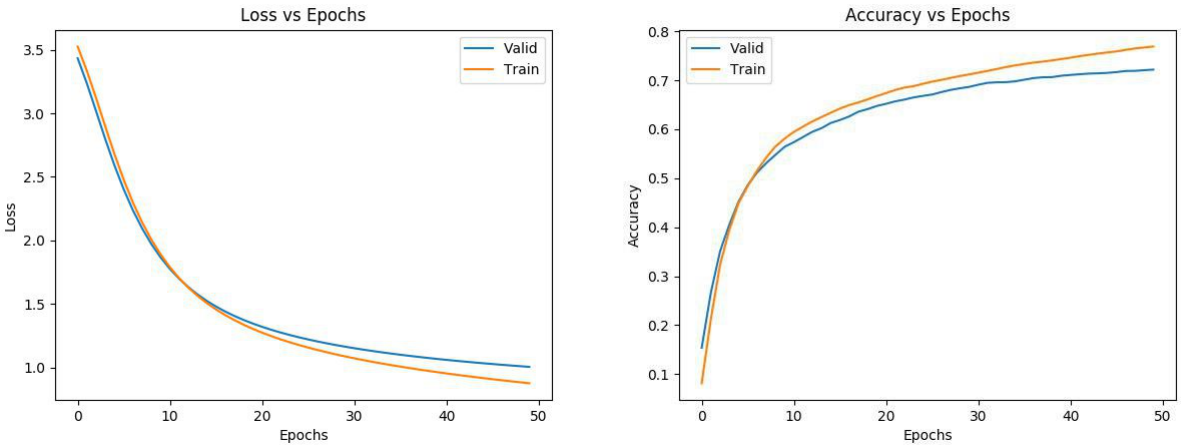


Fig 3: Learning_rate=0.0008



As can be seen based on Fig 3, as we decrease the learning rate, the learning becomes slower and convergence to the optimal solution happens in more epochs. Also, the under fitting happens and there is not much difference between validation and training set.

As can be seen based on Fig2, by increasing the learning rate, the oscillation happens a lot and global optima may be missed because of high learning rate and in some circumstances the algorithm may end up diverging.

As can be seen, based on Fig 1, learning_rate=0.008 is a good choice which causes the network to converge in low number of epochs.

Q3.1.3

Based on the images below, we can see that as the network is trained, the weights start to learn some simple patterns and shapes for presenting the digits and alphabets. The learned weights are different from random initialized weight since they include some shapes and forms in them.

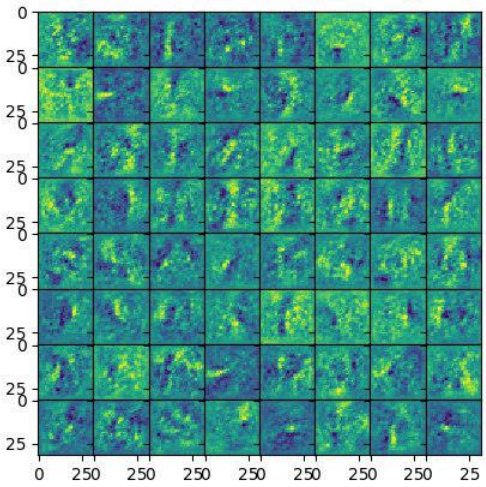


Figure 4: Final Learned Weights

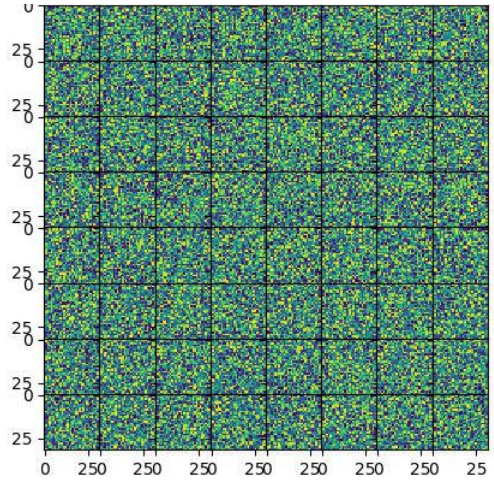


Figure 5: Initial Weights

Q3.1.4

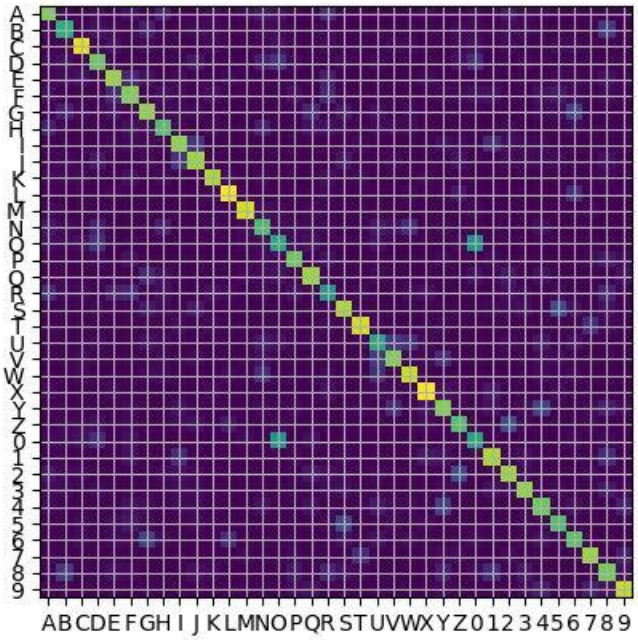


Figure 6: Confusion Matrix test data

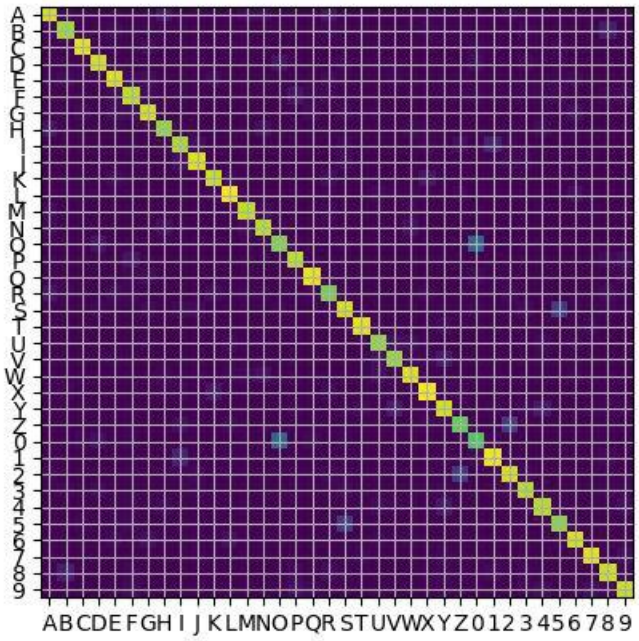
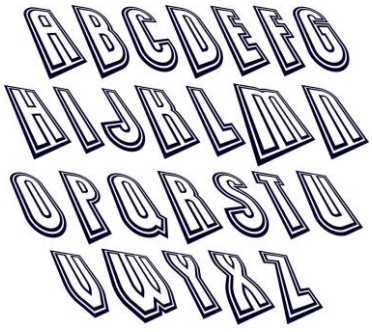


Figure 7: Confusion Matrix Train Data

Based on the images above depicting the confusion matrix for both test and training data, as can be noticed, there are some classes that are most probable to be confused with each other such as classes for “0” and “O” which happens both on training and test set which is reasonable based on their resemblance. Also classes, “2” and “Z”, “5” and “S”, “1” and “I”, “8” and “B”, “6” and “G” happen to be confused with each other because of the similarity between their shapes.

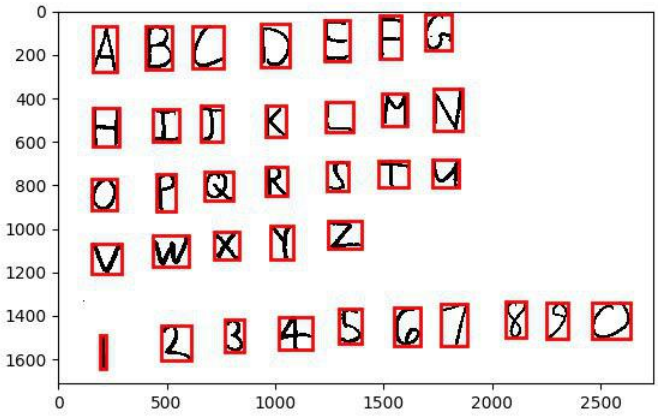
Q4.1

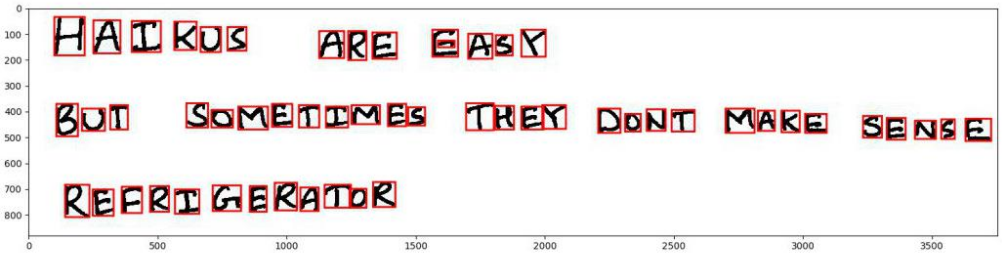
The network is trained on data that mostly comprises words having straight letters with no rotation which makes the network sensitive to the letters having inclined angles in words.

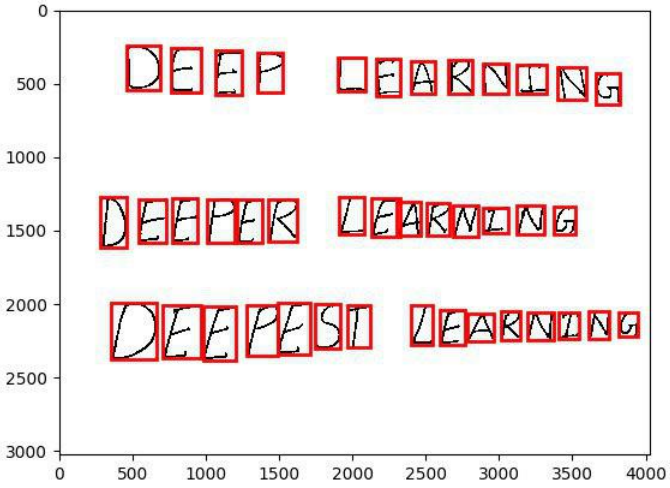


Also, the letters in words are written clearly and with reasonable space between the letters. If this space decreases and letters connect with each other, it makes it hard for the algorithm to identify the letters correctly.

Q4.3

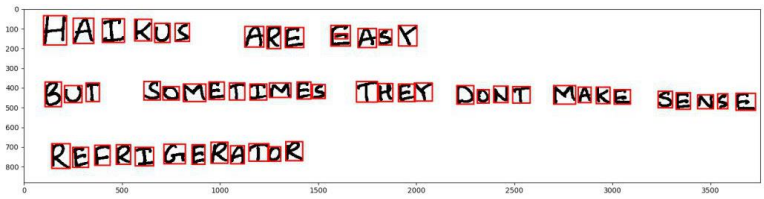




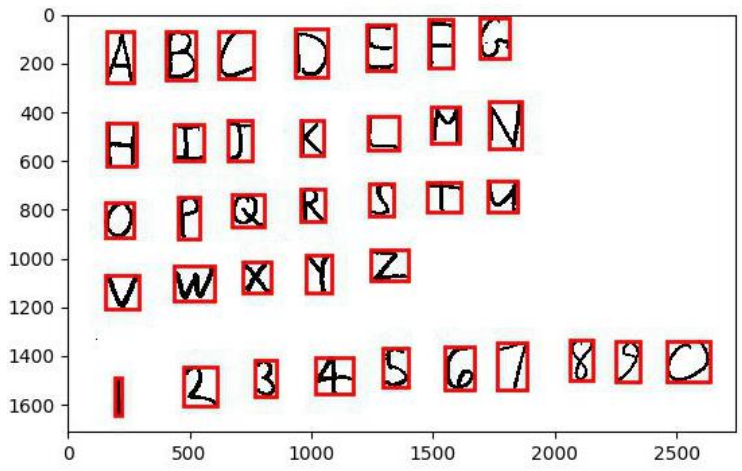
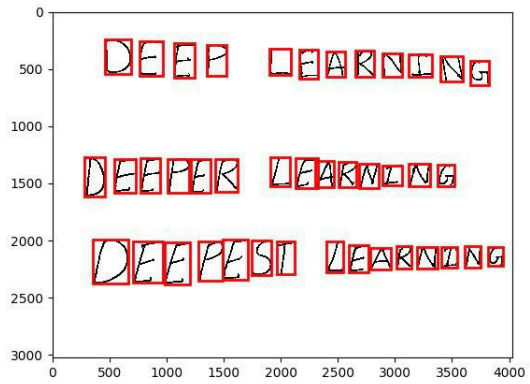


Q4.4

```
S B G D B F G
H I J K L M N
Q P Q R S T W
V W X Y Z
1 Z 3 G B G 7 8 9 J

F Q D O L I G T
L M A K 6 A T D D D L I G T
2 G H F G K D F F T H E F I R S T
T H I N G Q N T Q D Q L I S T
3 R 5 A L I Z E Y 0 U N A V E A L R 6 A D T
G Q M P L G T 5 D Z T H I N G G
4 R F W A R D Y Q W R G B L F W I T R
A N A P

H R I K G G A R G G M A B X
B U T S O M E T I M E G T R E X D O W T M A K G S B M Q R
R B F R I G B R R T Q R

J Y F P L H A R M I N G
D F F 9 F 8 L B A R Y I M G
D F 1 M F S F L F A R M I K G
```

	HRIKGGARGGMABX BUTSOMETIMEGTREXDOWTMAKGSBMQR RBFRIGBRRTQR
	Accuracy = 61.81 %

	FQDOLIGT LMAK6ATDDDLIGT 2GHFGKDFFTHEFIRST THINGQNTQDQLIST 3R5ALIZEY0UNAVEALR6ADT GQMPLGT5DZTHINGG 4RFWARDYQWRGBLFWITR ANAP
	Accuracy = 72.41 %

	SBGDBFG HIJKLMN QPQRSTW VWXYZ 1Z3G5G789J
	Accuracy = 72.22 %
	JYFPLHARMING DFF9F8LBARYIMG DF1MFSFLFARMIKG
	Accuracy = 51.21%