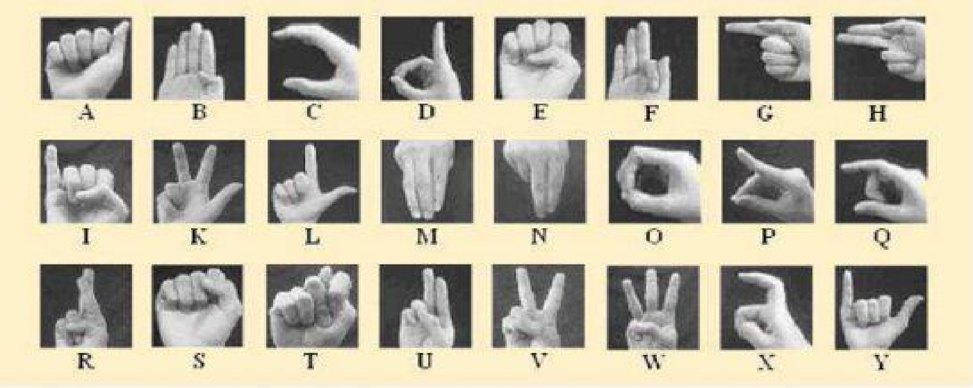
Hand Gesture Recognition



(Figure 1)

PROJECT REPORT

Under the Guidance of

Dr. Aruna Tiwari

Submitted by :-

Shrestha Kumar (150002034)
Shivam Tayal (150001034)

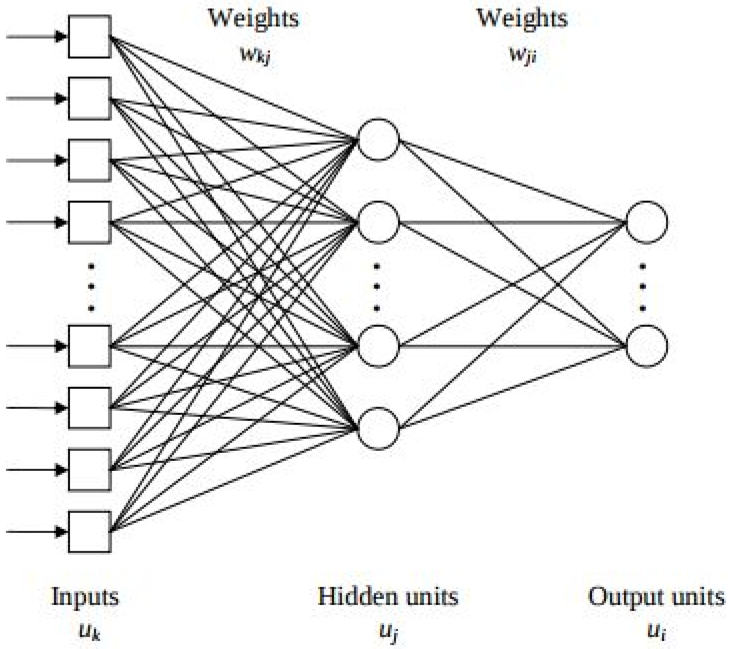
Introduction

- Nowadays many games are moving toward the gesture control system rather than the old joystick controls. This has brought a lot attention in the field of gesture recognition. There are lot of research going on for the real time gesture recognition.
- As an example, Figure 1 presents an image collection of hand gestures for the some English alphabet
- In this project we try to recognise only the English alphabets using the sign language
- We used the ‘MNIST Sign Language ‘ data, which is preprocessed and contains the pixel values (in GrayScale mode) of 28 * 28 pixels
- The values of pixels are stored in 8-bits i.e the pixel values range from 0 to 255

Approach

We make the following design decisions, some of which will be relaxed in the questions:

- **Topology**:
  We use a one-hidden-layer feedforward neural network, that consists of one layer of hidden units and one layer of output units.



Our Neural Network consists of **784 input neurons** , **10 hidden neurons** and **24 output neurons**.

- **Activation Function**:
  We use the sigmoid function as activation function for all units.

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

The derivative of the sigmoid function is

$$\sigma'(x) = \sigma(x)(1 - \sigma(x)).$$

- **Input Encoding**:
  We subsample the images and then represent them as matrixes of intensity values, one per pixel, ranging from 0 (= black) to 255 (= white).the dataset which contains all the images preprocessed and

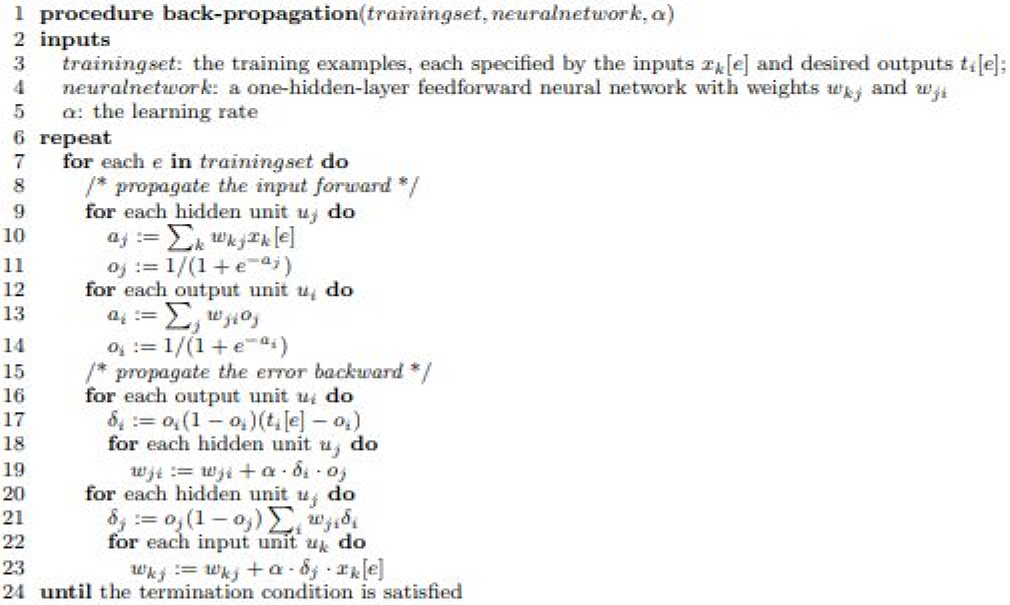
- **Output Encoding**:
  Each output of the neural network corresponds to a combination of the values of its output units. Imagine that a neural network has to decide whether an image contains the hand gesture “thumbs up.” In this case, we can use one output unit and map all output values greater than 0.5 to “yes” and all output values less than or equal to 0.5 to “no.”

- **Error Function**:
  We use the sum of squared errors as error functions. Consider a single training example *e*, where $t_i[e]$ represents the desired output and o_i the actual output of output unit u_i . Then, the difference of the desired and actual output is calculated as

$$E[e] = \frac{1}{2}\sum_i (t_i[e] - o_i)^2.$$

Algorithm

➢ We implemented the Convolutional Neural Network with Backpropagation algorithm in **C++**.



```
procedure back-propagation($trainingset, neuralnetwork, \alpha$)
inputs
  $trainingset$: the training examples, each specified by the inputs $x_k[e]$ and desired outputs $t_i[e]$;
  $neuralnetwork$: a one-hidden-layer feedforward neural network with weights $w_{kj}$ and $w_{ji}$
  $\alpha$: the learning rate
repeat
  for each $e$ in $trainingset$ do
    /* propagate the input forward */
    for each hidden unit $u_j$ do
      $a_j := \sum_k w_{kj} x_k[e]$
      $o_j := 1/(1+e^{-a_j})$
    for each output unit $u_i$ do
      $a_i := \sum_j w_{ji} o_j$
      $o_i := 1/(1+e^{-a_i})$
    /* propagate the error backward */
    for each output unit $u_i$ do
      $\delta_i := o_i(1-o_i)(t_i[e]-o_i)$
      for each hidden unit $u_j$ do
        $w_{ji} := w_{ji} + \alpha \cdot \delta_i \cdot o_j$
    for each hidden unit $u_j$ do
      $\delta_j := o_j(1-o_j)\sum_i w_{ji}\delta_i$
      for each input unit $u_k$ do
        $w_{kj} := w_{kj} + \alpha \cdot \delta_j \cdot x_k[e]$
until the termination condition is satisfied
```

Result

- We run the program on the training data with batch_size = 1
- The accuracy was coming out to be ~ 5%
- We compared our model with the python predefined model
- The accuracy that python predefined model achieved was also nearly 5 %

Future Work

- The neural network which we implemented takes input a pre-processed data
- We can make the neural network to responds to the real-time images using the Fast-RCNN networks and algorithms

Reference

Paper Link