CMPE 297 – Machine Learning

Comparison of Logistic Regression and Artificial Neural Networks for Facial expression recognition

Report By:

Archana Ramalingam (SJSU ID: 010761114)
Nidhi Agrawal (SJSU ID: 010728094)
Sai Usha Sri Veguru (SJSU ID: 011456211)

May 2017

Table of Contents

Abstract

Facial expressions play a vital role in human communication, conveying around 55% of the information. This non-verbal method of interaction becomes even more important when the communication is between a human and a computer (desktop, laptop and smartphone). Currently most systems use interfaces like manual input (text), voice to help users to interact with computers, instead of more efficient and secure methods like facial expression recognition. Recognizing facial expressions is characterized in human cognition as a pattern recognition problem. When the computers are trained long enough on this pattern recognition task, it can potentially perform better than humans.

Recently, due to the ready availability of large amount of data and computational power, techniques like Machine Learning and Neural Networks have become practically possible. Though machine learning methods like Logistic Regression can solve the above problem, they only have limited generalizability for new data. Artificial Neural Networks (ANN) are more efficient in solving the problem, given a large dataset, as they extract undefined features from the data. Hence, ANNs are more generalizable, producing better results. ANN is a group of interconnected artificial neurons, modeled based on human brain. It is normally used to model complex relationship between the input and output and usually has multiple layers.

Introduction:

Logistic regression and Artificial Neural Networks are used in different fields of science and image recognition etc. Here they are compared based on their performance on same dataset of Kaggle for Facial Expression Recognition.

Problem Statement

In various social media applications, machine learning is used to detect and label the faces in an image. This is a popular way of tagging friends in Facebook.

This application can be extended to detecting facial expressions too.

In our project, we are using two algorithms to detect facial expressions - Softmax Logistic Regression and Artificial Neural Networks and then we are comparing them.

We are detecting 7 common expressions from the human face.

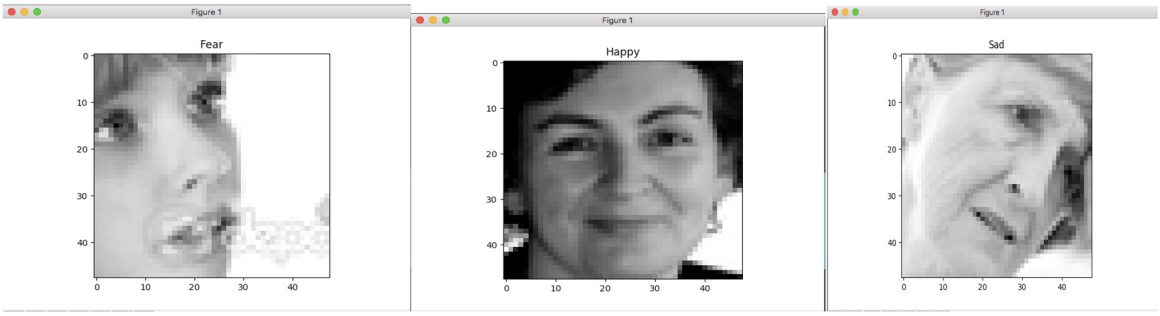
Data Source

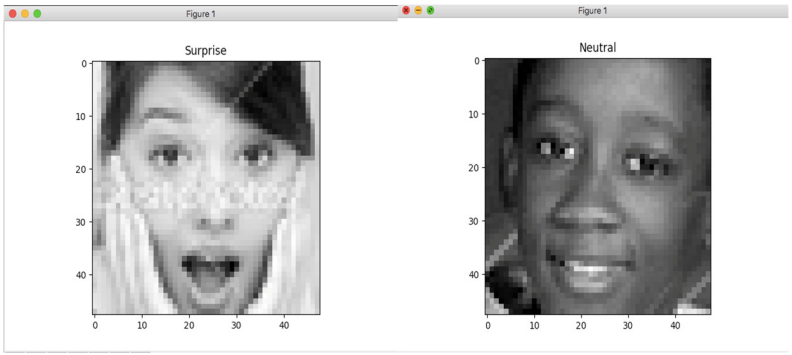
We got our dataset from Kaggle.com for "**Challenges in Representation Learning: Facial Expression Recognition Challenge**". It consist of a database of 48x48 grayscale images with human faces in the center.

Identified six facial expressions –we are dealing with a 7-class classification problem

1. 0=Angry,
2. 1=Disgust,
3. 2=Fear,
4. 3=Happy,
5. 4=Sad,
6. 5=Surprise,
7. 6=Neutral

Some sample images for each expression are shown below:





File descriptions

fer2013.tar.gz - photos of the training set

The training set consists of 28,709 examples, train.csv contains two columns, "emotion" and "pixels".

The "emotion" column contains a numeric code ranging from 0 to 6, inclusive, for the emotion that is present in the image.
The "pixels" column contains a string surrounded in quotes for each image. The contents of this string a space-separated pixel values in row major order. test.csv contains only the "pixels" column and your task is to predict the emotion column.

Models/Algorithm

In this project, we have implemented and compared the efficiency of Logistic Regression and Artificial Neural Network. Let's see the two models briefly. We have used Tensor flow library for implementing these models.

Tensor Flow

Tensor Flow is an open source library developed by the Google Brain Team. Due to a C, C++ backend, Tensor Flow is able to run faster than pure Python code.

A data flow graph has two basic units. A node represents a mathematical operation, and an edge represents a multi-dimensional array, known as a **tensor**.

- In this project, Images have a width and a height, so we'd need at least a two-dimensional structure. But remember, the pixels of an image are broken down into three color channels Red, Green, and Blue, or RGB So an image essentially needs three different matrices, to store the intensity of each color channel. So we can combine these three matrices together to form a tensor.
- To define variables, we simply use the command 'tf.variable'.
- Variables need to be initialized before a graph can be run in a session, which can be done using 'tf.initialize _all_ variables'.
- To update the value of a variable, we define an update function using 'tf. assign'. We can then run the operation.

When the graph is launched, we'll need to add an initialization operation for the variables. We can then start the **session** and run the graph. After the variables are

initialized, we print the initial value of the state variable, we run the update operation, and then we print the result after each update.

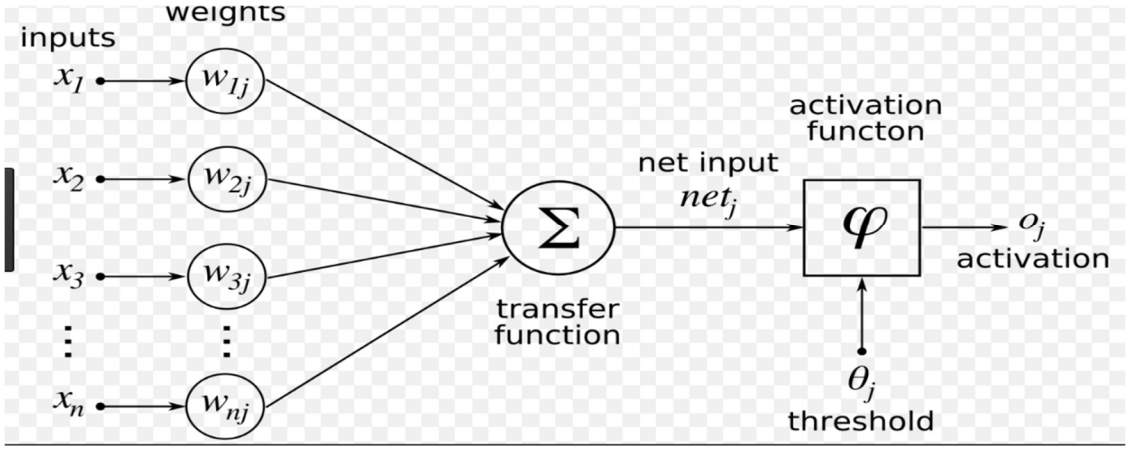
- If you want to feed data to Tensor Flow from outside a model, you will need to use **placeholders**.
- To create one, you can use the 'placeholder' method. You'll need to specify the data type, as well as the data type's precision in terms of the number of bits.

If you need to build a model to perform classification, with Tensor Flow, we can easily implement logistic regression.

Logistic Regression

Logistic Regression is an important tool for Machine Learning, and a great stepping stone for learning about more advanced models and algorithms. If we need to perform classification and determine which class a data point is likely to belong to, then Logistic Regression comes in role.

Logistic Regression provides a formula that predicts the likelihood that a given input belongs to a certain class. The model makes these predictions by analyzing the data's features, which are a set of independent variables that describe the data. Logistic Regression takes in a series of inputs and passes them through a set of three functions.



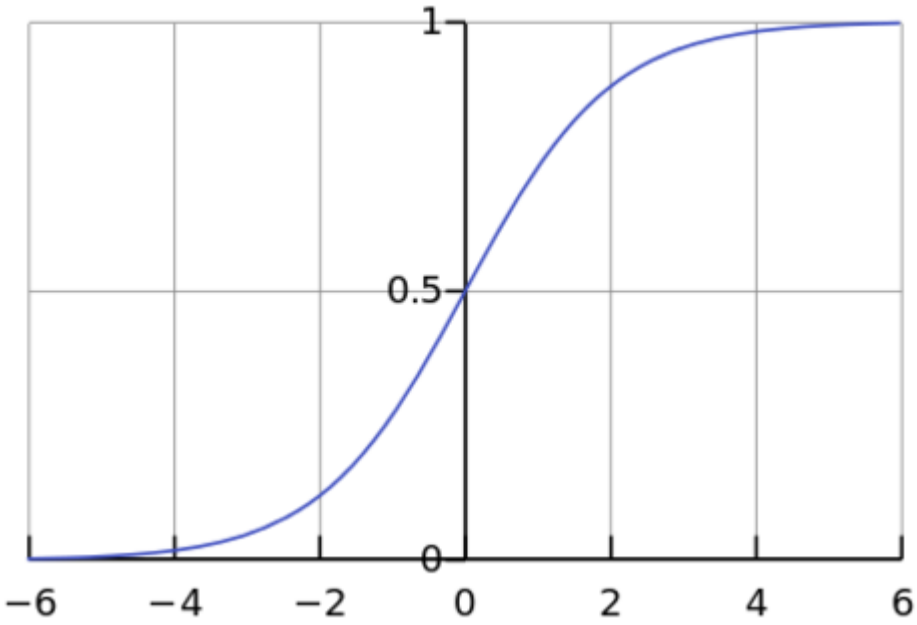
Logistic Regression Function model

Logistic Function:

The logistic function can be defined as a function that takes any real value as its input and provides an output whose value lies between zero and one and hence its called as probability.

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

A graph of logistic function is a sigmoid function:



Implementation of Algorithm:

First, the **inputs** are multiplied by a set of values known as the **weights**.

The weight values represent the relative amount of emphasis to put on each feature of the input. The model improves its classification ability by gradually updating these weights throughout the training process.'**matmul**' simply stands for matrix multiplication, and this function receives our input and the weights as parameters

The weight-multiplied value is then fed into another function, which adds on an additional value, called a **bias**. A bias is an important value that increases the flexibility of our model beyond what a set of weights could do alone. And like the weights, the bias is also updated throughout the training process. Result of the matrix multiplication step, and use the '**add**' function to add on the bias.

Our last step is to feed the resulting sum into something called a **sigmoid function**. The sigmoid function will map the sum onto a curve that represents the probability of the input belonging to a certain class**,** the sum through the **'nn.sigmoid'** function.

Softmax Regression or Multinomial Logistic Regression

It is a generalization of logistic regression to handle multiple classes. Logistic regression is used for binary classification only. To extend it to multiclass, we use Softmax function as the hypothesis, hence the name. Softmax function is a normalized exponential equation, which converts a K – dimensional vector of abstract values to a K dimensional vector containing only values from 0 to 1. Also, these values sum to 1, per the rule of probability. Softmax is also an activation function like sigmoid and is used in neural networks too. Here the output is a K-dimensional vector with probabilities, ranging from 0 to 1, that the given input belongs to a class.

- Label: Y can take K different values, instead of just two
- Hypothesis: estimates P(y=k|x) for each k = 0,1,...K
- Output: K-dimensional vector containing K estimated probabilities, with its elements summing to 1
- Hypothesis function:

$$h_\theta(x) = \begin{bmatrix} P(y=1|x;\theta) \\ P(y=2|x;\theta) \\ \vdots \\ P(y=K|x;\theta) \end{bmatrix} = \frac{1}{\sum_{j=1}^{K} \exp(\theta^{(j)\top} x)} \begin{bmatrix} \exp(\theta^{(1)\top} x) \\ \exp(\theta^{(2)\top} x) \\ \vdots \\ \exp(\theta^{(K)\top} x) \end{bmatrix}$$

To calculate the cost function, we need to calculate the indicator matrix from the output vector, with the position of the label value. Cost function is shown below.

- **Cost function:**

$$J(\theta) = -\left[\sum_{i=1}^{m}\sum_{k=1}^{K} \{\text{Indicator matrix}\} * \log \frac{\exp(\theta^{(k)\top} x^{(i)})}{\sum_{j=1}^{K} \exp(\theta^{(j)\top} x^{(i)})}\right]$$

Just like in logistic regression, we minimize the above cost function, *J(θ)* through gradient descent with partial derivative to optimize the weights. Then we update the weights and biases at every iteration through the equation shown below.

- Weight update equation:

W = W + learning rate * ∂J/∂W

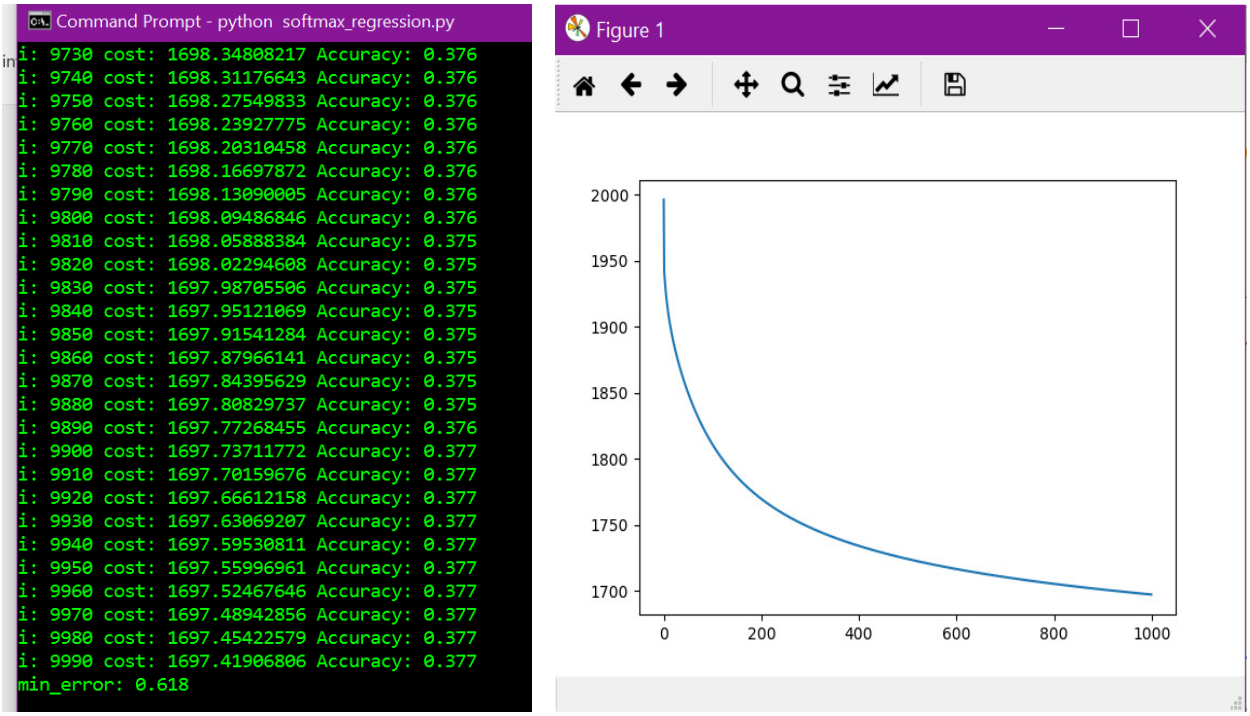
We have tried and optimized the values of regularization factor, number of epochs and learning rate. To maintain unbiased output, we balance the classes with minimum number of samples.

Pseudocode

N - # of samples, D - # of features, K - # of classes

```
softmax_regression{
        for i in range(N):
        indicator[i, y[i]] = 1
        W = np.random.randn(D, K) / np.sqrt(D)
            for i in range(10000)
            prediction_value = softmax(X.W+b) # forward propagation
            W -= learning_rate*(X.T.dot(prediction - indicator) +
        reg*self.W)
            cost(indicator_valid, prediction_valid)
            e = np.mean(Yvalid != np.argmax(prediction_valid, axis=1))
    }
```

Output:



We can see that with increase in iterations, the cost function is decreasing, error is decreasing and the accuracy is increasing. However, the maximum achievable accuracy is only around 37.7%. Also, even after 10000 epochs, the cost function does not seem to stabilize. The accuracy is very low and we need a more efficient method to solve the problem. Hence, we move onto Artificial neural networks. Approximate time taken to run this for 10000 epochs is around 2.5 hours.

Limitations of Logistic Regression:

- Identifying Independent Variables
- Limited Outcome Variables
- Independent observations required
- Overfitting the model

Applications of Logistic Regression:

Other applications of Logistic Regression include:

- Image segmentation
- Handwriting recognition
- Healthcare

- Geographical image recognition

Advantages of Logistic Regression:

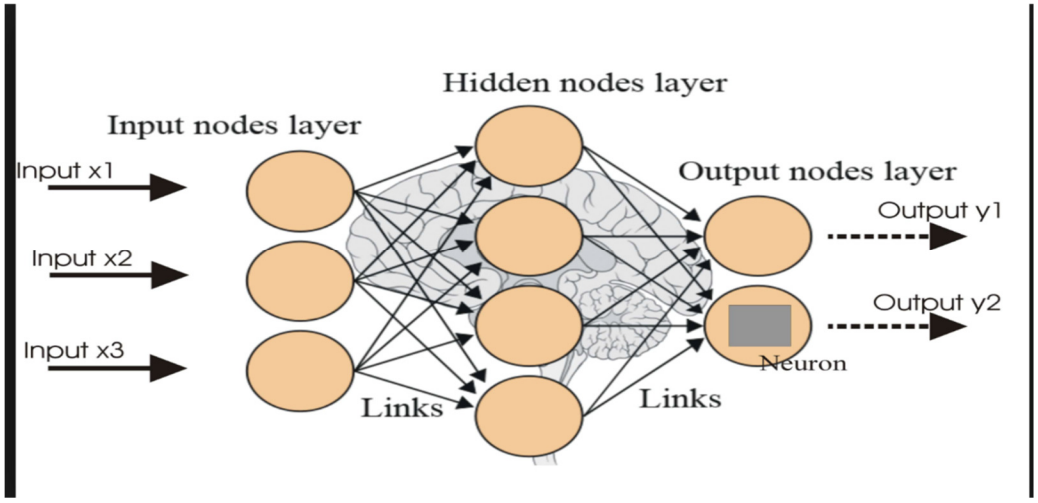
- It is very robust and values are between 0 and 1.
- The impact of independent variable on dependant variable is analyzed.

Artificial Neural Networks

It is a connection of simple units like neurons which are analogous to biological neurons. These neurons carry activation signal. If these neurons signals when connected give a strong activation function, the neuron is said to be fired and the signal is transmitted to other neurons.

A neural network's main function is to receive a set of inputs, perform progressively complex calculations, and then use the output to solve a problem.

First layer nodes get its inputs from the input layer, and activates. Its score is then passed on as input to the next **hidden layer** for further activation, until you reach the output layer, where the results of the classification are determined by the scores at each node. This happens for each set of inputs.



This series of events starting from the input where each activation is sent to the next layer, and then the next, all the way to the output, is known as forward

propagation, or **forward propagation**. Forward propagation is a neural net's way of classifying a set of inputs.

The reason is that each set of inputs is modified by **unique weights and biases**. For example, for that node, the first input is modified by a weight of 10, the second by 5, the third by 6 and then a bias of 9 is added on top. Each edge has a unique weight, and each node has a unique bias. This means that the combination used for each activation is also unique, which explains why the nodes fire differently.

The prediction accuracy of a neural net depends on its weights and biases. The **process of improving a neural net's accuracy** is called **training**, to train the net, the output from forward prop is compared to the output that is known to be correct, and the **cost** is the difference of the two.

The point of training is to make that cost as small as possible, across millions of training examples. To do this, the net tweaks the weights and biases step by step until the prediction closely matches the correct output. Once trained well, a neural net has the potential to make accurate predictions each time. This is a neural net in a nutshell.

Network Function:

The first layer has input neurons which send data to the second layer and the output of second layer to the third layer. These synapses store the parameters called weights that result in the actual output.

The three major terms that are important in neural networks are:

- The weights of the neurons for those layers
- The activation function that translate the neurons weighted outputs to output activation
- The type of interconnection between the different layers of neuron.

Mathematically, it can be represented as,

$$f(x) = K\left(\sum_i w_i g_i(x)\right)$$

K is the activation function, w_i, weights in those layers, and f(x) as input and g(x) as output.

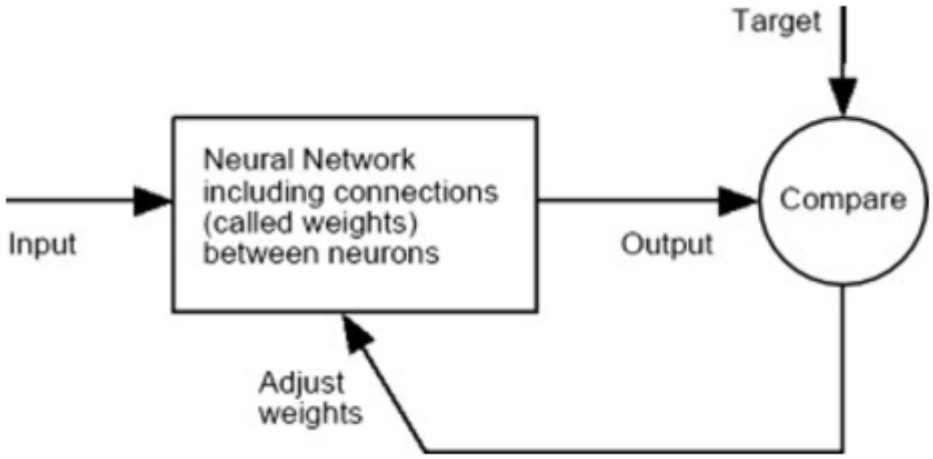
Technical Approach:

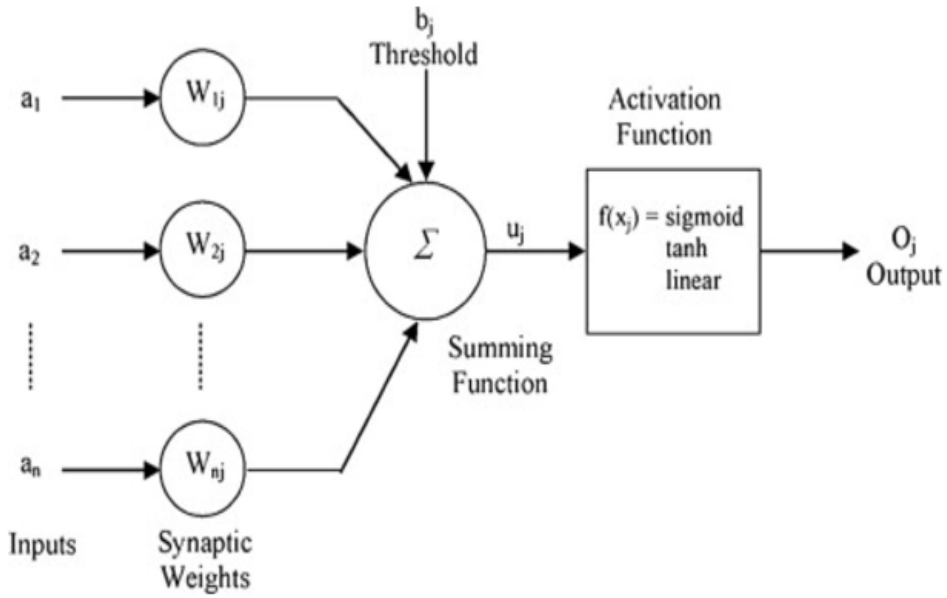
1. The kaggle dataset is included along with all the packages and util.py which consists of our activation function and the common functions that are reused in the program.
2. All the hidden layers are specified. The input and hidden layers are used in the following way:
   input > Weight > hidden layer 1 > Weights > Hidden Layer 2 > Weights > output layer
   (input -data * weights)+ biases
3. This is the feedforward mechanism that is used and each hidden layer consists of relu activation function. For each layer the output is compared with the intended value to reduce the cost.
4. Once the neural network is trained, the cost function is calculated as the difference between the predicted and the actual output value. An RMS optimizer is used for this purpose.

   Compare output to intended output > cost function

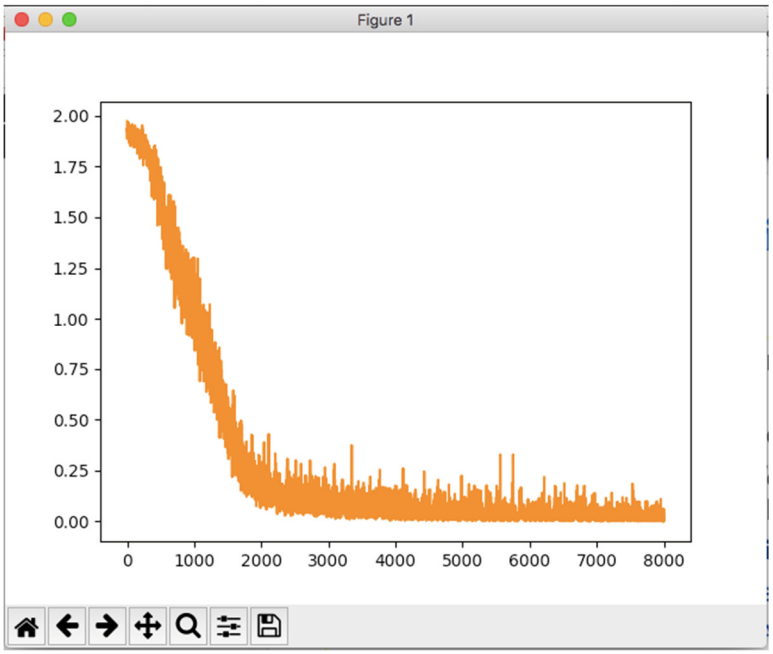
   Optimization Function > minimize cost (RMSOptimizer)

5. Feedforward + Backpropagation is epoch. This process is continued for the complete dataset fixing the number of epochs and segregating the data into batch size.
6. With the help of argmax the error rate is calculated and the plot between cost and input is obtained.


Target
Neural Network
including connections
(called weights)
between neurons
Input
Output
Compare
Adjust
weights


b_j
Threshold
a_1
W_{1j}
Activation
Function
a_2
W_{2j}
Σ
u_j
$f(x_j)$ = sigmoid
tanh
linear
O_j
Output
Summing
Function
a_n
W_{nj}
Inputs
Synaptic
Weights

Output:




```
i: 399 j: 300 nb: 392 cost: [None, 0.012881387] error rate: 0.471
i: 399 j: 320 nb: 392 cost: [None, 0.0084860818] error rate: 0.475
i: 399 j: 340 nb: 392 cost: [None, 0.0014409549] error rate: 0.476
i: 399 j: 360 nb: 392 cost: [None, 0.062462006] error rate: 0.469
i: 399 j: 380 nb: 392 cost: [None, 0.0012078271] error rate: 0.468
```

Limitations of ANN:

- Mostly for complex non-linear target functions.
- Training and computation is slow on normal CPU.
- Large memory requirements.

Other Applications of ANN:

- Pattern Recognition problems such as speech recognition, classification of sonar and radar signals.
- Financial and scientific forecasting.
- Multiprocessor scheduling and task assignment
- Credit scoring

Advantages of ANN:

- It is a non-linear model easy for understanding.
- It minimizes error using feedforward + backpropagation techniques.
- ANN has a good modelling technique.

Discussion

Deep learning doesn't really gives us plug and play black boxes

Correct learning rate?
Too high – the rate at which the weights change increases because of which we might not attain target function
Too low - slow convergence

Correct regularization
Too high- just make W very small, ignores actual pattern
Too low - Cost may explode to NaN

How many epochs?
Stop too early- steep slope not a minimum error
Stop too late- No effect on error, takes too long

Comparison between Logistic regression and Artificial Neural Network

- Softmax regression is a robust model, easier to implement for simple classification problems.
- ANN is more efficient and can be improved with additional layers and tuning backpropagation to learn complex functions.
- However, ANN is computationally costly and requires large amount of data for better performance.

Challenges Faced During Project:

1. Training neural network and understanding about the architecture was a very difficult task.

2. Obtaining the dataset and finding options to include in module
3. Running 400 epochs utilized lot of time and computation power.
4. Understanding tensor flow functions or both logistic and ANN.

Conclusion

Softmax regression is a robust model, easier to implement for simple classification problems. ANN is more efficient and can be improved with additional layers and tuning backpropagation to learn complex functions. However, ANN is computationally costly and requires large amount of data for better performance. ANN is more efficient than logistic regression from the two output graphs obtained the error rate is low for ANN than Logistic Regression.

Future Scope

- To increase the accuracy of the project, we could adopt more advanced models like Convolutional Neural Network
- The dataset considered has faces in the center and facing straight. We could extend our project for non-centered images with faces at an angle
- We are currently detecting only the predominant expression from the face in the image. This could be split into the percentage of all emotions. eg: 60% happy, 30% surprise, etc.

Acknowledgement

In this project, we have tried to understand the main concepts behind Logistic Regression and Artificial Neural Networks in simple terms. We have implemented this model using Tensor flow library and understood importance of Hyper parameters in this models and learnt how they affect the accuracy.

We would like to express my special gratitude to Prof. Rex Tsao for providing valuable information on Machine Learning basics concepts. We have taken effort in this report. However, it would not have been possible without references as listed in References section below. All images in this report belong to their respective authors.

References:

1) Christine L. Lisetti and David E. Rumelhart, Facial Expression Recognition Using a NeuralNetwork, Proceedings of the Eleventh International FLAIRS Conference, 1998
2) Deepthi.S, Archana.G.S, Dr.Jagathy Raj.V.P, Facial Expression Recognition UsingArtificial Neural Networks, IOSR Journal of Computer Engineering (IOSRJCE), 2013
3) Ali Mollahosseini, David Chan and Mohammad H. Mahoor, Going Deeper in Facial Expression Recognition using Deep Neural Networks, arXiv:1511.04110v1, 2015
4) https://en.wikipedia.org/wiki/Logistic_regression
5) https://en.wikipedia.org/wiki/Artificial_neural_network
6) http://www.sciencedirect.com/science/article/pii/S1110016812000518
7) http://homepages.cae.wisc.edu/~ece539/videocourse/notes/pdf/lec%2002%20applications.pdf
8) http://ufldl.stanford.edu/tutorial/supervised/SoftmaxRegression/
9) https://www.youtube.com/watch?v=PwAGxqrXSCs