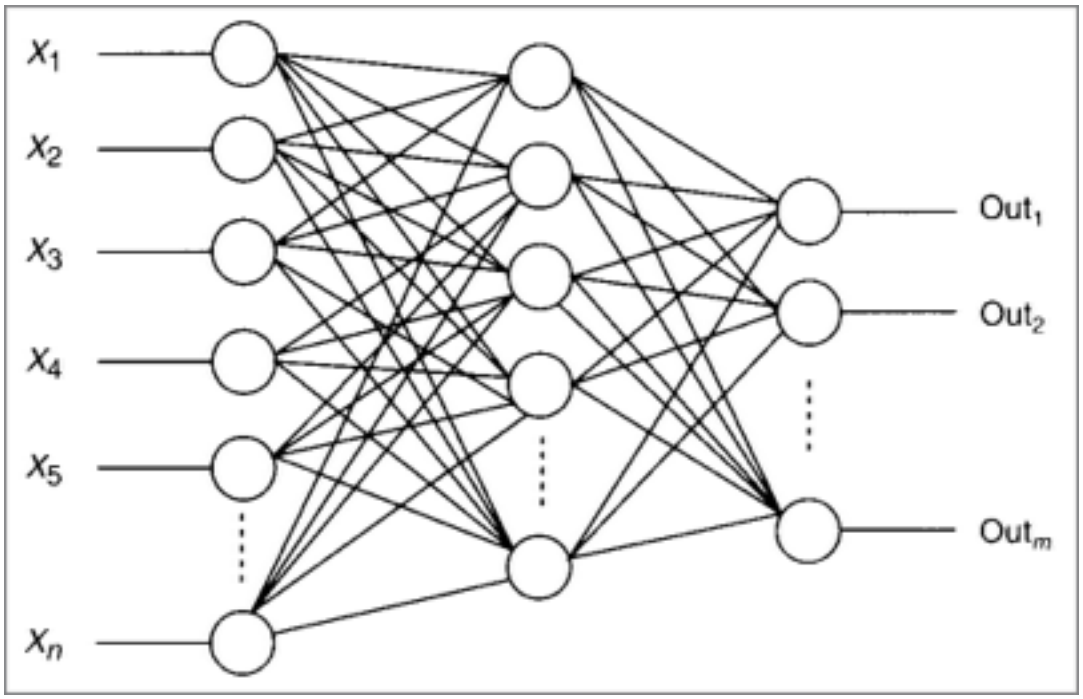
Artificial Neural Networks Report



BITS F464

20 November 2016

Varun Natu	2014A7PS841H
Aditi Ghosh	2014A5PS055H
Akanksha Pandey	2014A7PS151H
Ruthvi Reddy	2014A7PS040H

Artificial Neural Networks

Introduction 3
Data Set ,,..... 5
Preprocessing and Representation..... 6
Implementation Brief 7
Network Structure 8
Accuracy and Results 9
Conclusion 14

Introduction

The Algorithm

The backpropagation algorithm, that was originally introduced in the 1970s, is considered the workhorse of learning in neural networks. The goal and motivation behind this algorithm is to find a way to train a multi-layered neural network so that it can learn the appropriate internal representations to allow it to learn any arbitrary mapping of input to output.
The essence of backpropagation lies in understanding how the changes in weights and biases in a network affect the cost function.

The algorithm:

Stochastic Backpropagation(training examples, , ni, nh, no)
Each training example is of the form where is the input vector and is the target vector. is the learning rate (e.g., .05). ni, nh and no are the number of input, hidden and output nodes respectively. Input from unit i to unit j is denoted xji and its weight is denoted by wji.
Create a feed-forward network with ni inputs, nh hidden units, and no output units.
Initialize all the weights to small random values (e.g., between -.05 and .05)
Until termination condition is met, Do
For each training example , Do
1.Input the instance and compute the output ou of every unit.
2.For each output unit k, calculate
3.For each hidden unit h, calculate
4.Update each network weight wji as follows:

Applications:

The Backpropagation algorithm has been employed successfully in a wide variety of applications, a few of which have been listed below.

- Classification
- Function approximation
- Time series prediction
- Speech or voice recognition
- Image pattern recognition
- Medical diagnosis
- Automatic controls
- Geo-engineering(earth-slope movement and ground movement around tunnels, in particular)
- Debris flow prediction

Limitations:

The **step-size problem** occurs because the standard back-propagation method computed only $\partial E/\partial w$, the partial first derivative of the overall error function with respect to each weight in the network.
Give these derivatives, we can perform a gradient descent in weight space, reducing the error with each step. It is straightforward to show that if we take infinitesimal steps down the gradient vector, running a new training epoch to recompute the gradient after each step, we will eventually reach a local minimum of the error function. Experience has shown that in most situations this local minimum will be a global minimum as well, or at least "good enough" solution to the problem at hand.
In a practical learning system, however, we do not want to take infinitesimal steps; for fast learning, we want to take the largest steps that we can. Unfortunately, if we choose a step size that is too large, the network will not reliably converge to a good solution. In order to choose a reasonable step size, we need to know not just the slope of the error function, but something about its higher-order derivatives -- its curvature -- in the vicinity of the current point in weight space. This information is not available in the standard back-propagation algorithm.

A second source of inefficiency in back-propagation learning is what we call the **moving target problem**. Briefly stated, the problem is that each unit in the interior of the network is trying to evolve into a feature detector that will play some useful role in the network's overall computation, but its task is greatly complicated by the fact that all the other units are changing at the same time.
Instead of a situation in which each unit moves quickly and directly to assume some useful role, we see a complex dance among all the units that takes a long time to settle down.Many experimenters have reported that backpropagation learning slows

Data Set

We has used a set of images that were already divided into test and training sections, to accomplish out 3 tasks of face recognition, pose recognition and sunglass recognition.

The data set given consists of three sets, one training set and two test sets. The training set is made up of .pgm images of type P5. They are 8 bit grayscale images having a height of 30 pixels and a breadth of 32 pixels. The test sets have images of the same size and dimensions. The number of examples in the training and test sets vary depending upon the classification carried out and have been detailed below.

- **Pose Classifier**
  - **Train:** 277 images
  - **Test One:** 139 images
  - **Test Two:** 208 images
- **Sunglass Classifier**
  - **Train:** 70 images
  - **Test One:** 34 images
  - **Test Two:** 52 images
- **Face Classifier**
  - **Train:** 80 images
  - **Test One:** 36 images
  - **Test Two:** 40 images

Preprocessing and Representation

Image Representation

Each image is represented in an IMAGE structure. which stores the pixel intensity values parsed from the image file as a matrix of floating point numbers. The pixel intensities that are used are scaled down by 255.0 so that they remain as floating point numbers between 0.0 and 1.0 to aid learning. Further Implementation details can be found in the accompanying HTML documentation.

```
/*! \brief Representation of an Image
*
*   This structure is used to
*   represent the contents of an image
*   that is to be trained on or tested.
*/
typedef struct{
    float **data; //!< A Matrix of size nRows x nCols to store pixel intensities
    int nRows;    //!< The image height in pixels
    int nCols;    //!< The image width in pixels
    int maxGray;  //!< The maxmimum intensity value of the image, from 0 to maxGray
}IMAGE;
```

The images are then placed in an STL list to represent the set that is being used for training or testing.

Label Representation

Each classification tasks has a separate number of possible labels , namely POSE (4) , FACE (20), and SUNGLASS (2). In this effect, we have chosen to encode the outputs as an n-vector (STL) , where n corresponds to the number of outputs in the classification task. Thus each network has in it's output layer a number of nodes equal to the number of possible classifications of the task.

The n-vectors are made up of 1 value of 0.9 corresponding to the label and the remaining indices having a value of 0.1. The possible labels are enumerated to an integer. This integer specifies the field within the vector corresponding to the instance that takes 0.9 as its value.

This choice of label encoding is done because a sigmoid based neural network is unable to learn values of 0 or 1. Furthermore having a single output node with intermediate values to specify outputs would decreasing learning potential by a large margin.

These labels are created from the file name, by specifying to the below function a label index which specifies the location of the label in the filename where each field separated by an

```
int allocateExamples(list<IMAGE *> *trainingExamples,list<vector<float> > *labels, int labelIndex,char *imageList);
```

underscore _. These labels are once again stored as a list of vectors in the same order as the image structures. The image list and the Label List together make up the in-program representation of the data sets.

Network Representation

The Neural Network is represented as an STL list of layers. The order of these layers is innermost to outermost. i.e: The last entry in the list specified the output layer. Each of these layers is in itself an STL list made of up of nodes. These nodes are ordered in the list by their index. i.e: top to bottom. Each node in the neural network is represented as the structure shown below.

```
/*! \brief Representation of a Network node
*
*   Contains all the information needed to uniquely identify a node within a layer.
*   Each layer will be identified by it's own list and hence no layer identifier is needed
*   within the structure
*/
typedef struct{
    vector<float> *weights; //!< The weights associated with the node at the progressing iteration of the network \see update
    vector<float> *inputs; //!< The inputs that were used in the feed forwards stage and will be used in the update stage of the same iteration \see update
    float error; //!< The error as calculated during the backpropogation stage \see backProp
    float output; //!< The output of the node  \see generateOutput
    int layerIndex;//!< Index Within the Layer;
}node;
```

Implementation Brief

There are two main modules that have been written to accomplish the neural network task.

The first is the Image Parsing Module. This takes in a list of filenames of images that comprise of the set and create list representation of the images and labels as shown below.

The second module is the Network Module. This module performs two main tasks, that of learning an **arbitrary network** with specified number of output nodes. It also predicts the outputs and provides a classification accuracy on a test set and a pre learned neural network. The main driver functions for the two operations specified are shown below.

```
list<list<node *> > BackpropogationDriver(list<IMAGE *> *trainingExamples,list<vector<float> > *labels,int inputs,int iterations,list<int> layerNodes,float learningRate,int outputs);
float predict(list<list<node *> > LearnedNetwork, list<IMAGE *> instances, list<vector<float> > labels,int outputs);
```

Details of each parameter and it's use can be found in the accompanying HTML Documentation or by consulting the source code comments.

As such the functions are capable of learning any network over the given data and vector labels. The design of the network to be learned is specified through the list ***layerNodes*** and ***outputs.*** LayerNodes specifies the nodes in each layer in inner to outer order. Outputs specifies the number of classifications or output nodes which correspond to the number of values in each output label.

Network Structure

The number of layers that have been used in the tests for each classification task vary from 3 (with 1 hidden layer) , 2 (with 2 hidden layers) and 5 (with 3 hidden layers).

Each of the three networks have been tested for accuracy on a separate number of networks. Pose being the hardest set to classify from observation over initial data has been subject to to the largest number of structure in 36. The sunglass recogniser has been run on 36 Structures as shown below. Finally due to training time and processing constraints the Face Recogniser which has 20 output nodes was only tested for accuracy on 24 separate networks. The difference values for each classifier have been shown below.

Pose Recogniser (32)

- Number of Hidden Layers {1,2,3}
- Number of Nodes in a Layer {14,21}
- Number of Iterations {100, 300} for 1 hidden layer ; {200,400} for 2 and {350,500} for 3 hidden layers
- Learning Rate {0.4,0.55,0.7} for 1 and 2 hidden layers and {0.55,0.7} for 3 hidden layers

Sunglass Recogniser (24)

- Number of Hidden Layers {1,2}
- Number of Nodes in a Layer {7,14,21}
- Number of Iterations {200,300}
- Learning Rate {0.4,0.7}for 1 hidden layer and {0.5,0.8} for 2 hidden layers

Face Recogniser (24)

- Number of Layers {1,2,3}
- Number of Nodes in a Layer {20,40}
- Number of Iterations {200,350}
- Learning Rate {0.4, 0.6}

The results of each individual network can be found in the .txt files that accompany the source codes with the networks separated by layers in each classification task. The important results along with the maximum accuracies achieved have been listed below.

Accuracy and Results

While multiple networks may have given the same highest accuracy, in specifying the table we have used the simplest network (lesser layers followed by lesser nodes, followed by lesser iterations) that gave the best accuracy. For each classification task we have also examined the effect of each parameter that has been varied to obtain a general understanding of the effects of under training complex networks and overfitting simpler one, both of which ultimately have a negative effect on the test accuracy.

The results and accuracy obtained by each individual network tested can be found in the text files accompanying the code.

Additionally upon encountering a 100% classification accuracy on both the test data sets, we did not go ahead with training on networks with additional hidden layers

Sunglass Classifier

Maximum Accuracy on Test Data

Example Set	Best Accuracy	Hidden Layers	Nodes in a Layer	Iterations	Learning Rate
Test Set 1	100%	1	7	200	0.7
Test Set 2	100%	2	21	200	0.8
Test Set Average	100%	2	21	200	0.8
Training Set	100%	1	7	200	0.4

We did not proceed with an attempt using three hidden layers as we achieved a perfect classification using a two layer network with 21 nodes in each hidden layer and a learning rate 0.8

We now take a look at each one of the parameters and see their effect on the accuracy in a single layer network for the task of sunglass classification

- **Number of Nodes in a layer:**

We tested networks having 7, 14, and 21 nodes. We achieved a higher average accuracy for higher number of nodes. The reason behind this is that a larger number of nodes are able to learn a higher amount of information to represent the output function. In this way we were able to encode a larger amount of information in a 21 node layer than a 7 node layer.

We also found that larger the number of nodes, a larger learning rate and iterations were required to achieve higher classification accuracy. We reason that this occurs because having a higher degree of freedom due to having a larger number of nodes in a layer, the network needs more time to progress to a minima. This minima can be achieved by either increasing

the number of iterations (taking more steps) or by increasing the step size which is done through increasing the Learning Rate.

On the other other for 7 nodes. we saw a dip in accuracy from 200 to 300 iteration for a learning rate of 0.4 . This was the result of **overfitting** the training examples.

- **Iterations:**

The effect of increasing the number of iterations was dependent upon the other factors of the networks. For networks such a single layer network with a high learning rate, the effect of increasing the number of iterations induce decreased the accuracy due to the occurrence of overfitting. However as classification accuracies were very high even for relatively simpler models. The through effect of iterations on classification can not be concluded from sunglass classification.

- **Learning Rate:**

While a higher number of iterations are akin to taking more steps along the negative gradient. A higher learning rate corresponds to taking larger steps.

In this regard, the effect of the learning rate can be seen starkly in it's effect on the two layer networks. While a learning rate of 0.5 was able to classify the second test set with an accuracy of 98% with 21 nodes and 300 iterations. The bump up to a learning rate of 0.8 was able to achieve perfect classification.

- **Effect of Layers**

We saw with respect to sunglass classification that the second test set could not be classified with perfect accuracy using a single layer network but the first one could. The addition of a second layer enabled a perfect classification on all of the test data. This was the result of representing a more highly non-linear function which was possible due to adding a second hidden layer of nodes.

Face Classifier

Maximum Accuracy on Test Data

Example Set	Best Accuracy	Hidden Layers	Nodes in a Layer	Iterations	Learning Rate
Test Set 1	100%	2	20	350	0.6
Test Set 2	95%	1	20	350	0.6
Test Set Average	97.36%	2	20	350	0.6
Training Set	100%	1	7	200	0.4

We now take a look at each one of the parameters and see their effect on the accuracy in a single layer network for the task of face classification

- **Number of Nodes in a layer:**

Similar to the discussion in the previous classification problem, adding nodes to a layer was very beneficial given that the learning rat and number of iterations were high enough to bring the function to it's optima. This can be clearly seen though the result where all of the highest accuracies whether in the case of single layer or double layer were acquired for the number of nodes in 20.

40 nodes on the other hand were not able to obtain the same classification accuracies as their counterpart networks with lesser nodes per layer. We put this down to not finding the optimum learning rate and iteration combination to extract the added abilities of a 40 node layer. However,

Furthermore the effect and ability of number of nodes to encode more information per layer can be seen in that, the only network which was able to obtain perfect classification of the first test set was the 20 node network with a learning rate of 0.6. This also tells us that just having the optimum number of nodes is not enough, but it has to be supplemented by a learning rate which goes with it.

Given more time and computing power we believe a wider range of possible learning rate - iteration - node combination should be tried to obtain a higher average classification accuracy.

- **Iterations:**

With regards to combinations that we tried, we found that the number of iterations were very sensitive to causing overfitting. Where an increase of just a 100 iterations as we had done would cause a sharp fall in the tested accuracy.

- **Learning Rate:**

We found the a higher learning rate was required for the two layer and 3 layer networks to reach comparable accuracies while a much larger number of iterations would be required for a smaller learning rate. While for the single layered network a larger learning rate often caused the minima to be over shot and not yielding the accuracy that a lower learning rate tended to yield.

- **Effect of Layers**

The addition of a layer to the network with a single hidden layer allowed for the perfect classification of the first data set. Also the layer network consistently performed better on the first test set as compared to the single layer network. With regards to the second data set however, there were no accuracy gains to be had with the two layer network. The two layer network performed as well as the first one but not better. The jump from a two layer to a three layer network did not increase the accuracy and in fact decreased the maximum classification accuracy on the first set from 100% to 97.2%. This fall may be due to either the occurrence of overfitting or not training for long enough.

Pose Classifier

Maximum Accuracy on Test Data

Example Set	Best Accuracy	Hidden Layers	Nodes in a Layer	Iterations	Learning Rate
Test Set 1	93.52%	1	14	100	0.7
Test Set 2	96.63%	1	21	300	0.55
Test Set Average	95.38%	1	21	300	0.55
Training Set	100%	1	14	100	0.4

We now take a look at each one of the parameters and see their effect on the accuracy in a single layer network for the task of pose classification

- **Number of Nodes in a layer:** Our results over 32 separate networks as pertaining to the pose classifications task showed that keeping all other parameters the same, an increase in the number of nodes per layer increased the prediction accuracy in a strong majority of the cases. Furthermore we saw that in no case did increasing the number of nodes result in a sharp decrease in predictions accuracy. The small decreases seen in going from 14 nodes to 21 in a 1 layer, 0.7 learning rate network with 100 iterations can be attributed to a random set of starting weights and cannot be seen as evidence of an inferior accuracy. Further more as mentioned in earlier sections, this can be seen clearly as the ability of a higher number of nodes to encode more information as a result of the same number of gradient descent steps was opposed to lesser nodes. Thus, the only drawback of increasing the number of nodes is a sharp increase in training time. This increase in training time gets more and more signification as the number of hidden layers are increased and must be kept in ind while choosing a desired network structure.
- **Iterations:** As expected from theory, the role of the number of gradient descent iterations varied from network structure to network structure. For more complex networks that didn't already reflect overfitting (a worse accuracy than a less complex network) increasing the number of iterations often improved accuracy, however for simpler networks and complex networks with an already high number of iterations, A further increase in the number of iterations almost always produced a sharp noticeable reduction in prediction accuracy which was due to overfitting on the training data. With respect to the pose classification, this was especially important, the pose classification being a very subtle task overfitting was achieved a much lower network complexities than the other two tasks. This can clearly be seen in our network with two Layers, ,each layer with 21 nodes and a Learning Rate 0.7, where an increase in the iterations from 200 to 400 resulted in a **5%** decrease in test accuracy on the first test set.

- **Learning Rate:** For our networks trained on the pose classifier we found that a lower learning rate always favoured a higher accuracy than a higher one. Such a uniform result was only seen in this task, the other two tasks sometimes showed a higher accuracy for a higher learning rate when combined with fewer iterations. However, int he case of the pose classifier a higher learning rate almost always resulted in an **overshooting** of the minima, and showed an inability to converge to an acceptable one.
- **Effect of Layers:** While our experience with the other two tasks showed us that adding further hidden layers, and thus encoding a more highly non linear function would lead to a higher training accuracy, this was not what we encountered with regards to our last task. In fact both the two hidden layer and 3 hidden layer networks performed worse than the networks with the single hidden layers, with he 3 hidden layer network doing worse than the 2 layer network. Our data shows this clearly as the best accuracy achieved by a single hidden layer network was 94.7 % over both sets combined, while the best accuracies achieved by the 2 and 3 hidden layer networks were only 93% and 92% respectively. This apparent overfitting occurring at such a low level of complexity we believe, is due to subtle nature of the task as discussed with regards to increasing iterations.

Conclusion

Overall we used our neural network implementation based on the backpropogation algorithm for three separate tasks, each of which brought forth separate challenges. We are able to perform very well on the face and sunglass recognition tasks, obtaining either perfect or near perfect prediction accuracy on both. The Pose classification however, being a much more subtle difference between images proved harder. Despite the larger number of training examples available for the pose classification task, the best average on the combination of both test sets that could be obtained was 95.38%.

We varied out neural networks on 4 separate parameters, namely : **nodes in a layer, iterations, learning rate , number of hidden layers**. With regards to these parameters, our results reflected the theory and did in fact show that the number of iterations and learning rate are closely related to the speed of convergence to a minima, while the number of nodes in a layer and the number of hidden layers themselves determine the representational power of the network. The number of nodes in a layer are able to determine how powerful an individual layer , while having more layers allows the network to be increasingly non linear. We have described the detailed effects of varying each of the parameters for each of the 3 tasks in their own sections previously in the report.

Through the generation and testing of over 80 neural networks , we also found that a majority of the networks with higher complexity often ran into the problem of over fitting, where adding an additional layer, or increasing the number of nodes would often decrease the testing accuracy. This effect can be clearly seen in the case of the Pose classifier where despite going all the way upto a network with 3 hidden layers and 21 nodes per layer, the best classification was obtained by a network with a single and 14/21 nodes.

Finally our inability of getting an accuracy above 96%+ for the pose classification tells us how much training time and computing power is actually required for a neural network task as well as the importance of an acceptable training set representative of the universe. Perhaps using the full resolution images would yield higher classification for the task but doing so would exponentially increase both training time and computing requirements.

In conclusion, the neural network task and reuse of the same network code for several different tasks and network structures reveals the generalised learning power of a neural network and the reason for it's extensive use in today's environment.