Project Report: Artificial Neural Networks
CSCE 633 - Machine Learning

Vandana Bachani (UIN:220009534)

March 21, 2012

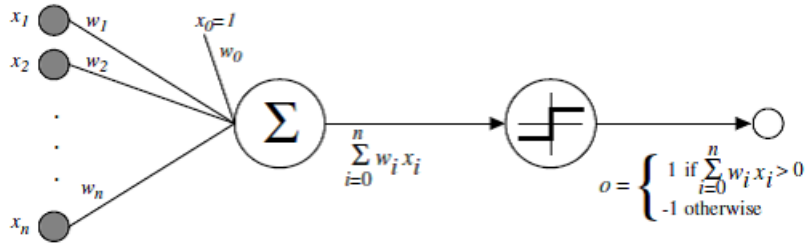
Abstract

Artificial Neural Networks are a learning technique used to learn complex data models. The project is aimed at analyzing how neural networks can be trained to model and solve classification problems on real life datasets. The main goal is to understand how the various parameters such as network structure, learning rate, gradient convergence can be statistically tuned as part of the training process and the effect of these on the performance and accuracy of the network.

1 Introduction

Artificial Neural Networks (ANNs) are an efficient learning technique inspired by the working of human brain. An artificial neural network consists of interconnected layers, of processing elements called "Perceptrons". Each processing unit in a layer receives inputs either from the environment (the dataset) or from outputs of other perceptrons. Each input connection has a connection weight associated with it. The processing unit produces an output that is a weighted linear combination of its inputs applied to an activation function. ANNs are a non-parametric approach to learning as they do not assume any distribution about the input.

Owing to the layered design of neural networks, they are capable of learning complex non-linear data models and are used for parallel processing. The most basic neural network comprises of a layer of perceptrons producing a linear combination of features in the input data, applying a threshold (activation) function to get binary output i.e. applying a step function to do binary classification (+1/-1). Generally neural networks involve more than one layer of perceptrons, i.e. intermediate layers comprising of "hidden units" connecting the input and output layers.



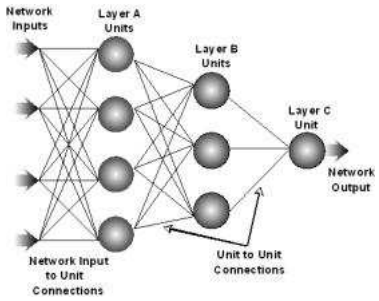
The important aspects of a neural network are:

1. The network structure, i.e. the number of hidden units, the interconnection pattern between different layers of perceptrons, etc.
2. The activation function that converts the unit's weighted input combination to its output activation.
3. The learning process for updating the weights of the interconnections.
4. The rate of learning and the adaptability of the network.

In the next section, follows a discussion of how to approach the above mentioned aspects of learning an artificial neural network.

2 Learning an Artificial Neural Network

1. **The Network Structure**
   The basic neural network, i.e. single-layer-perceptrons, is capable of learning only non-disjunctive concepts which are linearly separable. In order to learn some non-linear or more complex data models, we need to have hidden layers with hidden units in the network structure. Please refer the diagram below for an example network structure.



The number of hidden units required is evaluated by running experiments on a validation set with different network configurations and choosing the best one as is described in the Experiments and Results section.

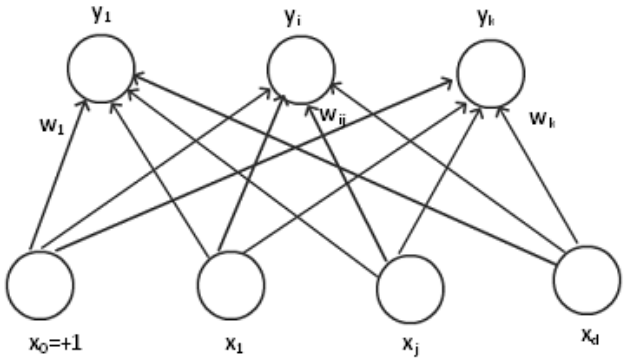
The other aspect which influences the network structure is the input and output encoding. Neural networks are very well suited for continuous input, regression problems where a continuous output value needs to be predicted or binary classification problems, wherein the inputs and outputs cause no overhead to the network structure. But in case of discrete inputs or multi-class classification problems, the input and output need to be encoded. Using one-hot encoding for discrete input attributes requires one to create additional nodes per discrete attribute to capture the various values that attribute can take. Similarly for multi-class classification, in case of 'k' class classification problem 'k' output nodes are created and the output is encoded such that there is a unique binary representation (0/1 output for each unit) based on the encoding for each class output. The increase in the number of processing units increases the number of connection weights.

2. **The activation function**
   In case of multi-layer perceptrons, with one of more hidden layers, a non-linear continuous differentiable activation function is applied to the weighted sum of inputs at the hidden nodes. The function needs to be differentiable because the learning equations are gradient-based. Several functions can be used such as,
   sigmoid function i.e. $f(x) = \frac{1}{(1+e^{-x})}$,
   hyperbolic tangent function given by $f(x) = tanh(x)$, etc.
   The sigmoid function is used for the implementation of this project.

3. **The Learning Process**
   The three important components of the learning process are:

   (a) The output of each unit can be represented as a linear combination of the inputs with each connection associated with a weight.

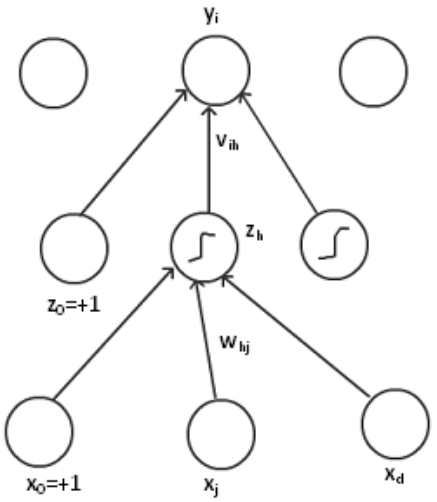
   For k perceptrons (for k-class classification problem),



   $y_i = \sum_{j=1}^{d} w_{ij}x_j + w_{i0} = \mathbf{w}_i^T\mathbf{x}$ .... (1)

   $\mathbf{y} = \mathbf{W}\mathbf{x}$ .... (2)

   For multilayer perceptrons,



$z_h = sigmoid(\mathbf{w}_h^T \mathbf{x}) = \frac{1}{(1+e^{[-\sum_{j=1}^{d} w_{hj} x_j + w_{h0}]})}, h = 1, .., H$ .... (3)

$y_i = \mathbf{v}_i^T \mathbf{z} = \sum_{h=1}^{H} v_{ih} z_h + v_{i0}$ .... (4)

The inputs are normalized to zero mean and variance one so as to nullify the effect of high-valued attributes. Random weights, between [-0.1, 0.1], are assigned to all the connections at the beginning of the training.

Missing Values

Missing discrete values in the input are treated as another value represented by a "?". Missing continuous values are substituted by the mean value for the particular attribute.

(b) The error estimate that is used to learn the weights is the squared error given by (generalized for multilayer perceptron),

$E(\mathbf{W}, \mathbf{V}|X) = \frac{1}{2} \sum_t \sum_{i=1}^{k} (r_i^t - y_i^t)^2$ .... (5)

(c) The Backpropagation algorithm.

Gradient Descent approach is used to incrementally learn the weights, wherein individual weights are incremented with each training example and the error in computation is propagated back into the network as contributions to the weight update equation.The weight update equations depend on the activation function at the units and on the deltas of error at the unit in the layers above it in the network structure.

The weight update equations can be derived using the simple derivative for any weight, w, by taking derivative of E w.r.t. w, i.e.

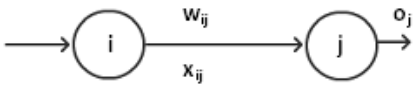
$\Delta w = -\eta \frac{\partial E}{\partial w}$, where η is the "learning rate".

When using the sigmoid function as the activation function and o_i is the output of i^{th} unit, a generic weight update equation which characterizes the backpropagation approach can be given as follows:

For each output unit k
$\delta_k = o_k(1 - o_k)(r_k - o_k)$, where r_k is the target value. .... (6)
If sigmoid function is not applied at the output,
$\delta_k = (r_k - o_k)$ .... (6)

For each hidden unit h
$\delta_h = o_h(1 - o_h) \sum_{k \in outputs} w_{h,k} \delta_k$ .... (7)

Each network weight $w_{i,j}$ can be updated as,
$w_{i,j} = w_{i,j} + \Delta w_{i,j}$ .... (8)
where
$\Delta w_{i,j} = \eta \delta_j x_{i,j}$ .... (9) for the connection,



The algorithm (in case of one hidden layer):

Initialize all v_{ih} and w_{hj} to rand$(-0.01, 0.01)$
Repeat
  For all $(\boldsymbol{x}^t, r^t) \in \mathcal{X}$ in random order
    For $h = 1, \ldots, H$
      $z_h \leftarrow \text{sigmoid}(\boldsymbol{w}_h^T \boldsymbol{x}^t)$
    For $i = 1, \ldots, K$
      $y_i = \boldsymbol{v}_i^T \boldsymbol{z}$
    For $i = 1, \ldots, K$
      $\Delta \boldsymbol{v}_i = \eta(r_i^t - y_i^t)\boldsymbol{z}$
    For $h = 1, \ldots, H$
      $\Delta \boldsymbol{w}_h = \eta(\sum_i (r_i^t - y_i^t) v_{ih}) z_h (1 - z_h) \boldsymbol{x}^t$
    For $i = 1, \ldots, K$
      $\boldsymbol{v}_i \leftarrow \boldsymbol{v}_i + \Delta \boldsymbol{v}_i$
    For $h = 1, \ldots, H$
      $\boldsymbol{w}_h \leftarrow \boldsymbol{w}_h + \Delta \boldsymbol{w}_h$
Until convergence

Δw as calculated in Equations (6),(7),(8),(9) can be used to generalize the algorithm for multiple layers.

The criteria of convergence of gradient descent and for stopping the algorithm is based on the mean square error on a validation set (different from the training set), details of which are mentioned in the Experiments and Results section.

4. **The rate of learning and adaptability of the network**
   The learning rate, η, is an important factor which affects the speed and accuracy of prediction of the neural network. In the current implementation the learning rate is tuned with experiments on a validation set. Please refer the Experiments and Results section for the results.

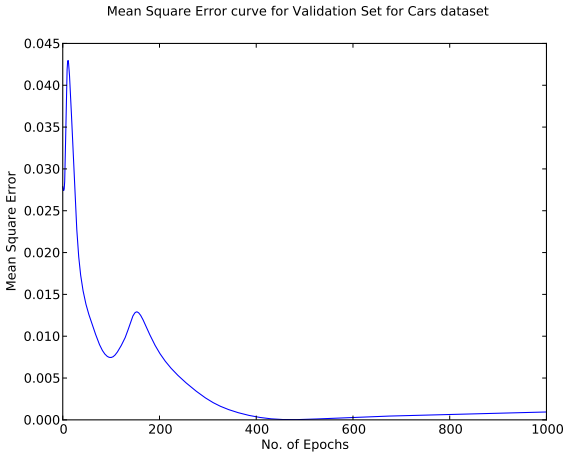
   A momentum factor can also be introduced in the weights learning process, given by,
   $\Delta w_i^t = -\eta \frac{\partial E^t}{\partial w_i} + \alpha \Delta w_i^{t-1}$
   The implementaion uses the momentum equation to speed up the learning process. Please refer comparison results in the Experiments and Results section.

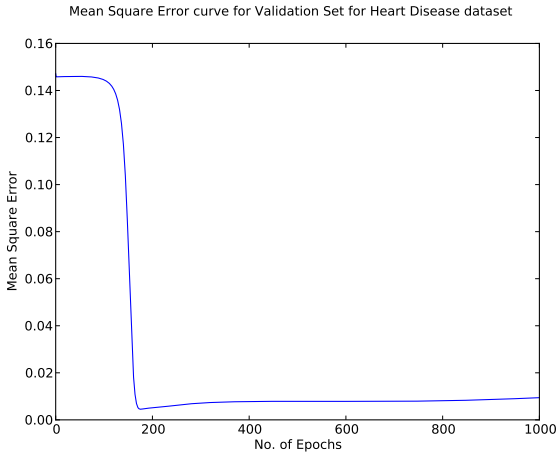
3 Experiments and Results

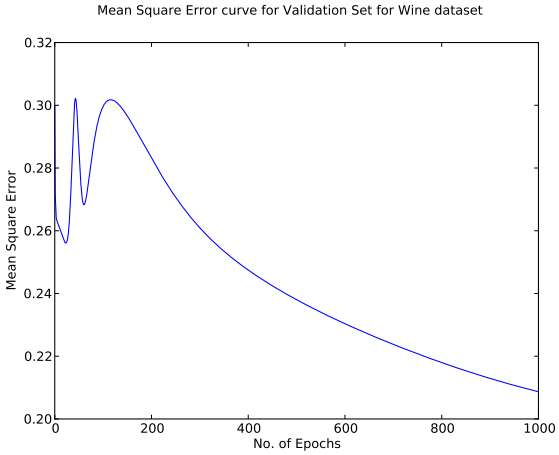
1. **Convergence Criteria**
   The implementaion uses the "half square error" on "validation set" as the criteria for convergence. The square error decreases on a validation set through the training process upto a certain number of training epochs. After sometime the square error is expected to increase on the validation set due to overfitting, at which stage the algorithm must stop. The least square error, network weights configuration is taken as the best model to predict accuracy on the test set.

   Square Error, $E = \frac{1}{2} \sum_{i=1}^{k} (r^i - y^i)^2$, where k is the number of examples in the validation set.

   But the criteria for convergence is tricky given the diversity of the datasets, i.e. after how many increasing values of square error on the validation set should the algorithm make a decision? Choosing this value is critical as otherwise the algorithm may stop at a local minima and one may not get the best accuracy for the network.
   The diversity in datasets and the variation in mean square error on validation set for 1000 epochs is depicted in the following graphs:







As observed in the above graphs, for cars and wine dataset there are local minima in which the algorithm may stop if it looks forward at 5 or 10 epochs of increasing square error.
Hence the value of stop criterion, i.e. no. of epochs to look forward for increasing validation set square error, is customized for datasets.
Following are the convergence results of the runs of the algorithm for the various datasets and the stop criterion used for each:

dataset	StopCrit	Iter1 Acc|Stop	Iter2 Acc|Stop	Iter3 Acc|Stop	Iter4 Acc|Stop	Iter5 Acc|Stop
iris	20	100.0 | 55	100.0 | 99	93.33 | 0	93.33 | 60	90.0 | 50
cars	100	93.91 | 492	93.04 | 341	83.19 | 116	79.71 | 84	73.04 | 26
mushroom	5	100.0 | 11	100.0 | 6	100.0 | 11	100.0 | 6	100.0 | 29
voting	100	97.7 | 318	96.55 | 34	96.55 | 39	96.55 | 29	93.1 | 39
heart	100	60.0 | 22	57.78 | 21	57.78 | 23	55.56 | 21	53.33 | 21
wine	20	100.0 | 172	100.0 | 92	100.0 | 165	97.14 | 86	97.14 | 91

2. **Experiments to determine the most effective network structure**
The configuration of the network specific to a dataset in terms of the number of hidden nodes to be used is found by running a set of experiments on the datasets. The experiment involves finding the accuracy of the network for a given configuration for a validation set.

The number of hidden layers is represented with an array, where each element corresponds to an additional hidden layer and the number in the position corresponds to the number of units in that hidden layer.
Example: [3,5] means the network has 2 hidden layers and the first layer contains 3 hidden units and the second layer contains 5 hidden units.
As the UCI datasets are not very complex, the experiments are conducted with configurations upto 2 hidden layers and not many hidden units. The implementation is generic and can support any number of hidden layers and hidden units.

Accuracies for different network configurations

--------------------Iris--------------------

Config	Accuracy	MSE	Stop
[3]	96.6667	6.14228351373e-05	466
[]	83.3333	0.0331059869308	319
[3, 5]	33.3333	0.395654272324	21
[5]	100.0	0.0588252621899	61
[10]	96.6667	0.00126544287954	490

--------------------Cars--------------------

Config	Accuracy	MSE	Stop
[3]	77.971	0.0431054233001	83
[]	94.2029	0.0046125419138	0
[3, 5]	94.2029	0.0107160857827	838
[5]	79.1304	0.0170570488364	108
[10]	94.7826	0.00855669842999	0

--------------------Mushroom--------------------

Config	Accuracy	MSE	Stop
[3]	99.8768	0.00472777053088	6
[]	99.9384	4.64399725711e-05	15
[3, 5]	99.8768	0.00434172382899	8
[5]	100.0	2.62293161311e-06	0
[10]	100.0	2.71563076846e-07	0

--------------------Heart Disease--------------------

Config	Accuracy	MSE	Stop
[3]	53.3333	0.0139488291885	248
[]	53.3333	0.0804968895975	0
[3, 5]	57.7778	0.136277866982	21
[5]	53.3333	0.0340742737333	272
[10]	53.3333	0.0326824805013	231
[20]	53.3333	0.00692472369956	121
[10, 5]	57.7778	0.136252351786	21

--------------------Voting--------------------

Config	Accuracy	MSE	Stop
[3]	96.5517	0.0039519428665	386
[]	97.7011	0.0060991674969	0
[3, 5]	65.5172	0.148847199706	22
[5]	96.5517	0.00363330061924	340
[10]	95.4023	0.000106870996771	41

--------------------Wine--------------------

Config	Accuracy	MSE	Stop
[3]	100.0	0.00513858073305	243
[]	97.1429	0.0185107421953	55
[3, 5]	0.0	0.421532018899	22
[5]	100.0	0.0068261622036	236
[10]	100.0	0.0124598219186	375

As is evident from the observations, the following configurations have been determined for the datasets: Iris: [5], Cars: [10], Mushroom: [5], Heart Disease: [10,5], Voting: [3] or [5], Wine: [5]

3. **10-fold cross validation for accuracy calculation & Dataset Specific Analysis**
   Results from 10-fold cross validation depicting the mean accuracy and variance for different datasets:

Dataset	Accuracy
Iris	96.000000 ± 3.787080
Cars	87.870801 ± 4.612165
Wine	97.647059 ± 1.786129
Voting	95.421512 ± 1.543244
Mushrm	99.963054 ± 0.048876
Heart	49.575758 ± 6.198178

   As can be observed from the accuracy results, Neural Networks do not perform well for the Heart Disease dataset. For all other datasets they perform reasonably well.

4. **Comparison with Perceptrons and Decision Trees**

The following table depicts the performance of neural networks compared to Perceptrons and Decision Trees:

```
Comparison between Perceptrons, Decision Trees and Neural Networks:

                Perceptron              NeuralNetwork           DecisionTree
Iris            77.333333 ± 9.637497    96.000000 ± 3.787080    94.000000 ± 2.892430
Mushroom        99.987685 ± 0.022899    99.963054 ± 0.048876    99.409048 ± 0.144128
Heart           46.515152 ± 3.535062    49.575758 ± 6.198178    52.484848 ± 5.975396
Cars            73.078811 ± 2.155257    87.870801 ± 4.612165    94.399225 ± 0.989713
Voting          95.397287 ± 1.581554    95.421512 ± 1.543244    94.723837 ± 2.313021
Wine            97.647059 ± 3.341541    97.647059 ± 1.786129    NA
```

The following observations were made:

(a) The accuracy on Iris dataset is best achieved through neural networks with one hidden layer. Perceptrons perform the worst on Iris.

(b) Neural Networks and Decision Trees are at par for most of the datasets. For the Cars dataset Decision Trees perform better than neural networks.

(c) Neural Networks give more precise predictions i.e. the variance in accuracy is lower than that of decision trees except for in case of heart disease dataset.

(d) Heart Disease dataset is still the worst performing dataset with no improvement in accuracy.

(e) Perceptron performance for Mushroom, Voting and Wine datasets is the same as that of neural networks.

5. **Momentum**

Momentum can be applied to the weight update process as given by the equation,
$\Delta w_i^t = -\eta \frac{\partial E^t}{\partial w_i} + \alpha \Delta w_i^{t-1}$

As can be seen in the results of above sections, the car dataset takes a lot of epochs to converge with a good accuracy. But using the momentum improves the convergence efficiency from approx. 250 epochs for a 90%+ accuracy to 78 epochs.

```
Accuracy:  89.2753623188 MSE:  0.000312435713869
Accuracy:  89.2753623188 MSE:  0.00031504873342
Accuracy:  89.2753623188 MSE:  0.000317447400069
Accuracy:  88.9855072464 MSE:  0.000319646860712
Accuracy:  88.9855072464 MSE:  0.000321654943983
Accuracy:  88.9855072464 MSE:  0.000323474154376
Accuracy:  88.9855072464 MSE:  0.000325104458774
Accuracy:  89.5652173913 MSE:  0.000326546675137
Accuracy:  89.5652173913 MSE:  0.000327806176859
Accuracy:  90.1449275362 MSE:  0.000328896568211
Accuracy:  90.1449275362 MSE:  0.000329842968986
Accuracy:  90.4347826087 MSE:  0.000330684567763
Accuracy:  90.7246376812 MSE:  0.000331476158061
Accuracy:  90.7246376812 MSE:  0.000332288453577
Accuracy:  90.7246376812 MSE:  0.00033320708024
Accuracy:  91.0144927536 MSE:  0.000334330255553
Accuracy:  91.0144927536 MSE:  0.000335765280726
Accuracy:  91.0144927536 MSE:  0.000337624078999
Accuracy:  91.3043478261 MSE:  0.000340018104479
Accuracy:  91.3043478261 MSE:  0.000343053010288
Accuracy:  91.3043478261 MSE:  0.000346823494041
Accuracy:  91.5942028986 MSE:  0.0192335365432
91.5942028986 0.0192335365432 78
```

References

[1] Machine Learning By Tom Mitchell.
http://www.cs.princeton.edu/courses/archive/spr07/cos424/papers/mitchell-dectrees.pdf

[2] Introduction to Machine Learning By Ethem Alpaydin.
http://www.cmpe.boun.edu.tr/ ethem/i2ml2e/