CS 539 Machine Learning

Project 3 - Neural Networks

Chris Winsor

The Face Recognition Dataset

The first dataset we investigate is the CMU Face Images Data Set from Tom Mitchell at the School of Computer Science, Carnegie Mellon University. The dataset is available at
http://archive.ics.uci.edu/ml/datasets/CMU+Face+Images

The database consists of 32 images for each of 20 people (640 images). Each image shows the individual in one of four profile positions, with one of four expressions, with or without sunglasses. Sixteen of the 32 images are marked as bad resulting in 624 images. We use the quarter-resolution images which are 32x30 pixels. The .pgm file contains the image.

The name of the file indicates the target class values.

UserId	an2i, at33, boland, bpm, ch4f, cheyer, choon, danieln, glickman, karyadi, kawamura, kk49, megak, mitchell, night, phoebe, saavik, steffi, sz24, tammo
HeadPosition	straight, left, right, up
FacialExpression	neutral, happy, sad, angry
EyeState	open, sunglasses
Scale	{1,2,4} indicating full-resolution, half-resolution, quarter resolution

Guiding Questions

- We first want to simply explore the effectiveness of a 2-layer feedforward Neural Network on this dataset. Specifically we will investigate how many hidden nodes are necessary to accurately predict the target attributes above, based on the image. To get some level of confidence in our results we use both Matlab (Experiment 1) and Weka (Experiment 2).

The Spambase Dataset

The second dataset we investigate is the Spambase Dataset originated from the University of California (UCI) Data Repository. It is available at http://archive.ics.uci.edu/ml/datasets/Spambase

The database consists of 4601samples. There are 57 attributes and one class variable.

48	numeric	real[0..100]	word_freq_WORD	number of times WORD shows up in the email
6	numeric	real[0..100]	char_freq_CHAR	percentage of characters in the email that are CHARl
1	numeric	real[0..100]	capital_run_length_average	average length of uninterrupted sequences of capital letters
1	numeric	integer[1..]	capital_run_length_longest	length of longest sequence of capitals
1	numeric	integer	capital_run_length_total	total number of capitals in the e-mail
1 (class)	nominal	{0,1}	spam	denotes whether the e-mail was (1) or was not (0) spam

Guiding Questions

Here we want to explore the modeling of this dataset using a 2-layer feedforward Neural Network. In this case the dataset has a more complicated set of attributes when compared to the image dataset. In this case the output is a single binary class output. We will want to see the impact of these characteristics on the predictive ability of the neural network. We will again investigate how the number of hidden nodes, and again reproduce our work using both Matlab and Weka.

Algorithms and Code (Weka)

In the experiments we will use Weka's neural networks which are are based on their Multilayer Perceptron class. Multilayer Perceptron uses a least-means-squared error to adjust weights during training. A reset feature is implemented if training is shown to be not improving during training. Parameters for learning rate and momentum are available.

Multilayer Perceptron
Nodes in Multilayer Perceptron are the NeuralConnection class which implements a sigmoid output stage. Primary input and output nodes use the NeuralEnd class which, in the case of an output (class), behaves as an unthresholded linear unit.

Input Format
Inputs to the network are assumed to be numeric and by default are normalized as they are applied to the network. To "normalize" in this case means the attribute is scaled based on its max and min values such that the result is in the range -1 and +1. There is a -I switch to indicate that normalizing the attributes should not be done.

Output Format
The class output of the MultilayerPerceptron can have either a numeric or normal format. The determination of format is set by the dataset when the network is constructed. If the output is numeric it will be "normalized" by default. In this context "normalize" means the output is scaled based on the range of values that the output can take. The -C switch can be used to indicate that normalizing the numeric class should not be done.

MultilayerPerceptron.buildClassifier()
buildClassifier() is the primary method to build, train, and test the MultilayerPerceptron network. Its operation is summarized as follows:

```
// buildClassifier() first builds the network.  Inputs and outputs are created.  These are
instances of NeuralEnd which in the case of an input node is the attribute number and in the
case of an output node is the output value.   The hidden nodes are instances of NeuralNode
which is the full node.
setupInputs()
setupOutputs()
setupHiddenLayer().

// The code then trains and tests the network:
for n epochs {
  for each instance (input sample) {
    reset the network
    calculate the network outputs
    calculate a weighted learning rate
    calculate error (mean squared error)
    update network weights based on weighted learning rate, considering momentum
  }

  // do validation testing
  for each instance in the validation set {
    reset the network
    calculate network outputs
    calculate mean square error using the validation set
  }
  if (this networks performance (error) is better than the last one) {
    if (this network's performance is better than the best one) {
      save this set of weights
    }
  } else {
    driftOff++ (increment the count of unimproved epochs)
  }
  if (driftOff has exceeded the threshold for lack of improvement) {
    set a flag to accept these weights - we are done
  }
}
```

Algorithms and Code (Matlab)

We use Matlab's Neural Network Toolbox in Experiments 2 and 4. We will use patternnet() for network creation, the Levenberg-Marquardt backpropagation learning method, and net() to generate network outputs. Performance measurement is done using our own code which computes the absolute difference between the predicted and test class values (this is discussed further in the "Post Processing" section of the experiment).

Network Creation

- patternnet() - This function is the basis of the "Pattern Recognition and Classification" group of nets. It creates a model where targets are nominal (categories). The output stage is a sigmoid. Validation can use mean-squared-error and confusion matrix analysis.

Learning Function

- trainlm - Levenberg-Marquardt backpropagation
- [net,tr] = train(net,inputs,targets) trains the network

Network Use

- outputs = net(inputs) is the means to use the model for prediction

Preprocessing of the Data (Face Recognition Dataset)

Starting with the 624 .pgm files, we use the filename to establish the value of the class variable (HeadPosition).

Attributes
The 32x30 2-d pixelmap is the attribute. It is a numeric value. As part of preprocessing we remove the dimensionality (the images are reshaped from 2d pixelmap to a 1-d vector).

Class Variable
The target (class variable) for experiments 1, 2 and 3 is HeadPosition.

For testing with Matlab - a set of four binary class variables are established (HeadPositionStraight, HeadPositionLeft, HeadPositionRight, HeadPositionUp). The value for these target variables is 0.1 and 0.9 indicating FALSE and TRUE respectively. The reason values of 0 and 1 were chosen is because the sigmoid output of the perceptron cannot produce the value of 0 or 1 which makes training more difficult.

For purposes of testing using Weka - Weka requires a single normal class value. Here the class variable is HeadPosition, with four discrete values straight, left, right up.

Preprocessing of the Data (Spambase Dataset)

Attributes
The original dataset has 57 attributes.
They are numeric values, unbinned.
There is a lot of cross-correlation between attributes.

Preprocessing Steps::

- Remove highly correlated attributes
- Bin the remaining attributes

Class Variable
There is one class variable which identifies if the mail is spam or not. No preprocessing is required here.

Performance Metrics

For Experiments 1 through 4 we use Accuracy (percent correct) and Training Time as measures of performance.

Baseline Performance

Face Recognition Dataset

Method	Accuracy
ZeroR	25%
OneR	58%
J4.8	83% (2.34 seconds to build - 36 leaves, 71 nodes)

Spambase Dataset

Method	Accuracy
ZeroR	60.6%
OneR	78.1%
J4.8	92.7% (0.2 seconds to build - 86 leaves, 125 nodes)

Experiment 1 - Faces w/ Varying HiddenNode Count (Matlab)

In this experiment we use Matlab "patternnet()" to build 2-layer models of the Faces dataset while varying the number of hidden nodes. We observe the effect of number of hidden nodes on training time and model accuracy..

Additional Pre or Post Processing

Output of the network (a numeric value) is compared to that of the target. An absolute difference below 0.4 is considered a successful prediction, and above 0.4 an unsuccessful prediction.

Parameters and Settings

We use a 2-layer feedforward architecture (one input layer, one hidden layer, one output layer).

- `hiddenLayerSize =` **`<vary with values of 1,2,3,4,5,10,20,30 >`**`;`
- `net.trainFcn = 'trainlm';  % Levenberg-Marquardt`
- `net.performFcn = 'mse';  % Mean squared error`

Performance of the Resulting Model

Networks with number of hidden nodes greater than 10 result in an out-of-memory error during training.

Number of Hidden Nodes	Percent Correct (min)	Percent Correct (max)	Training Time in seconds (min)	Training Time in seconds (max)
1	80.9%	85.9%	7.1	12.8
2	90.3%	99.0%	7.1	41.8
3	78.7%	99.1%	7.1	54.2
4	98.9%	99.0%	7.1	159.5
5	85.1%	98.7%	7.1	227.7
10	95.5%	99.2%	7.1	1064.3

Analysis Of Results

- Performance of a single-hidden-node neural network is equivalent to that of the J4.8 classifier baseline. The 12 second training time is noteworthy.
- Increasing the number of nodes increases its ability to predict but this comes with an accompanying increase in maximum training time.
- At 10 hidden nodes the network is able to predict above 95% accurately but at this point the training time becomes excessive.

Experiment 2 - Faces w/ Varying HiddenNode Count (Weka)

In this experiment we use Weka "MultilayerPerceptron" to build 2-layer models of the Faces dataset while varying the number of hidden nodes. We observe the effect of number of hidden nodes on training time and model performance.

Additional Pre or Post Processing

There is no further pre- or post-processing.

Parameters and Settings

We use a 2-layer feedforward architecture (one input layer, one hidden layer, one output layer).

- hiddenLayers <vary with values of 1,2,3,4,5,10,20,30>

Performance of the Resulting Model

Number of Hidden Nodes	Percent Correct (min)	Percent Correct (max)	Training Time in seconds (min)	Training Time in seconds (max)
1	47.6	50	3	12
2	71.2	73.6	3	10
3	87	93.8	4	10
4	89.9	96.6	4	8
5	90.4	97.1	7	23
10	91.3	97.6	9	25
15	89.9	97.6	10	24
20	92.3	93.3	14	39
30	91.3	98.1	29	231

Analysis Of Results

- The single-hidden-node network starts with only 50% accuracy.
- At three nodes the neural network exceeded the performance of the J4.8 decision tree.
- As the number of nodes is increased the performance of the network increases consistently, reaching mid 90% accuracy at 30 nodes.
- We were able to build and run a 30 node Weka network without trouble.

Experiment 3 - Spambase w/ Varying HiddenNode Count (Matlab)

Objectives

In this experiment we use Matlab "patternnet()" to build 2-layer models of the Spambase dataset while varying the number of hidden nodes. We observe the effect of number of hidden nodes on training time and model performance on this model.

Additional Pre or Post Processing

Output of the network (a numeric value) is compared to that of the target. An absolute difference below 0.5 is considered a successful prediction, and above 0.5 an unsuccessful prediction.

Parameters and Settings

We use a 2-layer feedforward architecture (one input layer, one hidden layer, one output layer).

- `hiddenLayerSize =` **`<vary with values of 1,2,3,4,5,10,20 >`**`;`
- `net.trainFcn = 'trainlm';  % Levenberg-Marquardt`
- `net.performFcn = 'mse';  % Mean squared error`

Performance of the Resulting Model

Number of Hidden Nodes	Percent Correct (min)	Percent Correct (max)	Training Time in seconds (min)	Training Time in seconds (max)
1	96.2	96.6	1.59	2.87
2	96.1	96.3	1.88	3.71
3	96.4	96.6	1.61	1.97
4	96.1	96.6	1.93	2.14
5	96.4	96.6	1.82	2.40
10	96.1	96.6	2.25	3.56
20	96.6	96.9	3.51	4.98

Analysis Of Results

- In this case, the single-hidden-node network has an accuracy of 96% - well beyond the J4.8 baseline.
- Additional hidden nodes provides no significant benefit.
- The training time for this network is fast and does not increase significantly with additional nodes.

Experiment 4 - Spambase w/ Varying HiddenNode Count (Weka)

Objectives

In this experiment we use Weka "MultilayerPerceptron" to build 2-layer models of the Spambase dataset while varying the number of hidden nodes. We observe the effect of number of hidden nodes on training time and model performance.

Data, Additional Pre or Post Processing

The data is the Spambase Dataset described in the "Guiding Questions" section. There is no further pre- or post-processing.

Parameters and Settings

We use a 2-layer feedforward architecture (one input layer, one hidden layer, one output layer). Parameters used by the weka.classifiers.functions.MultilayerPerceptron are the same as those from Experiment 2.

The testing uses 3-fold cross-validation.

Performance of the Resulting Model

The table below shows the results of this experiment.

Number of Hidden Nodes	Percent Correct (min)	Percent Correct (max)	Training Time in seconds (min)	Training Time in seconds (max)
1	93.2	93.5	0.36	0.56
2	92.6	94.3	0.37	0.50
3	92.8	94.1	0.50	0.90
4	93.2	94.0	0.61	0.97
5	92.9	93.6	0.89	1.54
10	92.5	94.0	1.24	3.91
20	92.0	94.2	1.98	6.69

Analysis Of Results

- Here again, using Weka to analyze the Spambase database shows that a single-hidden-node network has an accuracy equal to that of the baseline J4.8 decision tree.
- In this case additional nodes did increase the training time however it never approached that of the Imaging dataset.

Summary of Results (Experiments 1 to 4)

The results of the four experiments were surprising:

- A neural network with even a few hidden nodes had a significant predictive power. In each case a neural network with three or fewer hidden nodes would equal or exceed our baseline references including the J4.8 decision tree.

- The spambase dataset was particularly surprising. In this case there was little benefit of adding more than a single hidden node. The nature of the attributes (the correlation between them) allowed a single-node network to predict accurately.

- Where both the number of hidden nodes and the number input attributes was large, the training time of the neural network became prohibitive.

Experiment 5 - Predicting Image from Characteristics

Objectives

In this experiment we explore the possibility that we can predict the image from characteristics.

Assumptions:
We assume the 'real world entity' we are modeling has behavior that can be described in terms of stimulus and response (stimulus goes in and response comes out). We further assume we are interested in that subset of real-world-entities which are stateless. With these assumptions it should follow that the stimulus-to-response behavior of those real-world entities is deterministic. That is, given a particular stimulus the real-world entity would produce a deterministic response. The same cannot be said for the opposite. That is, given a response there may be many stimuli that would result in such a response. In other words, given a response we cannot guarantee we can determine the stimulus.

In the faces dataset there is such a stimulus-response relationship - the stimulus is the real-world person, position, facial expression and eyewear. The response is the two dimensional image that falls on our retina or sensor array. There may be error in sampling, and the attributes we have access to may be a small fraction of factors in the real world, but those are characteristics of the sampling and data, not the real-world entity.

In the baseline experiments we are given image and attempt to predict the person, position, facial expression and eyewear. The model we createed in that was was a **response-to-stimulus model** (predicts stimulus given response).

In this experiment we wish to attempt to predict response (image) given stimulus (characteristics). To do this we create a **stimulus-to-response model.**

The motivation is twofold:
a) Intuitively we believe we should be able to take advantage of the deterministic nature of the real-world-entity when we create a stimulus-to-response model. Given our assumptions we know there is a deterministic relationship to be found. Can the Neural Network take advantage of this?
b) Is there a relationship that can be made between the stimulus-to-response model and the response-to-stimulus model. Can each leverage knowledge of the other? Perhaps they can share trained parameters? How are they similar to one-another in architecture ? How are they different?

In this experiment we investigate we use Weka "MultilayerPerceptron" to build 2-layer models of the Spambase dataset while varying the number of hidden nodes. We observe the effect of number of hidden nodes on training time and model performance.

Data, Additional Pre or Post Processing

The data is that of the Faces dataset. The Characteristics of the image (20 users, 4 head positions, 4 expressions, 2 eyewear) are converted from unordered nominal to binary nominal and become the Attributes. The image which is a set of 960 integer pixel values in the rante [0..255] are left as-is and become Class variables.

Parameters and Settings

We use Weka to construct a 2-layer feedforward architecture. The network has 2 hidden nodes and one class (output) node. [Due to limited skill and time we predict just a single pixel - number 460].

Performance of the Resulting Model

The table below shows the results of this experiment.

```
Scheme:      weka.classifiers.functions.MultilayerPerceptron -L 0.3 -M 0.2 -N
500 -V 15 -S 0 -E 20 -H 2 -R

Time taken to build model (1 hidden node): 0.04 seconds
Correlation coefficient                  0.8779
Root mean squared error             41.1191

Time taken to build model (2 hidden nodes): 0.08 seconds
Correlation coefficient                  0.8872
Root mean squared error             41.6447

Time taken to build model (10 hidden nodes): 0.64 seconds
Correlation coefficient                  0.9249
Root mean squared error             32.0435

Time taken to build model (30 hidden nodes): 1.13 seconds
Correlation coefficient                  0.9136
Root mean squared error             34.2106
```

Analysis Of Results (Experiment 5)

Based just on the results above, although we were able to construct a model we cannot make conclusions as to its quality. We predicted only a single attribute which is certainly not a representative sample. With just a single pixel we weren't able to do our subjective measure which is unfortunate!

But we can make the following observations:

There did not appear to be any way to share information (parameters, weights) between the response-to-stimulus models (Experiments 1 to 4) and this model (Experiment 5). The models are completely unrelated.

Although we knew (or assumed) there was a deterministic stimulus-to-response relationship there didn't seem to be any way to take advantage of this using the Neural Network.