

CSI 5325 Assignment 3

Greg Hamerly

assigned February 12, 2009; due March 5, 2009

1 Expectations

For all assignments in this class you should work individually, but feel free to discuss your work with me or your classmates.

The writeups you do for this course should be of high quality; complete but concise. Please structure your writeups well, introducing ideas logically (rather than simply chronologically). Your writeup should be formatted as if you would submit it to a conference or journal. It should include the following sections in the given order:

- Introduction – Describe the problem(s) you are working on at a high level, and give the reader a summary of the results you found.
- Methodology – Explain your data sets, data preprocessing, the algorithms and techniques you apply, the hypotheses you are testing, and how you will evaluate the success of your experiments. Use tables and figures as are helpful.
- Results – Explain the results of the experiments, using graphs and tables and text. Then, analyze your results.
- Conclusion – Summarize the work again, and if applicable, describe further work that you think would build on what you’ve done.

For this particular assignment, each section (introduction, methodology, etc.) should address each of the parts of this assignment below. You could, for example, have a subsection which discusses each part within each section.

1.1 Submitting your work

You should turn in all your work in printed format in class, as well as by email to `hamerly@cs.baylor.edu`. Your email should have attached a zip file which has a single folder that contains all your materials. The zip file name should follow this pattern: ‘`lastname_nn.zip`’, where lastname is your last name, and nn is the assignment number (like ‘01’). Please DO include all source code with your emails, but DO NOT print out your source code to hand in (just print and turn in writeup).

1.2 Tools for the course

As suggested in assignment 1, your work should be composed in \LaTeX , and I suggest looking at MATLAB for programming.

2 Implement neural network and backpropagation

Implement a general software package which can train and evaluate neural networks. For this assignment, let's just stick with a simple 3-layer neural net, where each input node is connected to each hidden layer node, and each hidden layer node is connected to each output layer node. The input nodes just represent the input values (they do not modify them), but the hidden and output nodes should all be sigmoid nodes.

Your software should be able to operate in 2 modes:

1. Learning from training examples. Input is a neural network topology and a set of examples, as well as a learning rate (which may be set in software), output is a set of weights learned with backpropagation.
2. Making predictions on unseen examples. Input is a neural network topology, the network's weights, and a set of examples, output is a set of network outputs.

2.1 Input/output format

Let's use common formats for input and output of examples, network descriptions, and weights.

2.1.1 Format for reading examples

A set of examples will use the following conventions:

- An set of examples should be plain text. The examples will all use real-valued attributes. The first line should contain two integers, o and i , which represent the number of outputs and inputs, respectively. Obviously, o should match the number of output nodes in the network you train with these examples.
- This will be followed by a number of examples, one per line. Each example will start with a target vector, which is the correct set of o output values as real numbers. This is followed by d input values. All these numbers are space-separated. Of course, the input and output values should always come in the same order.

2.1.2 Format for describing a network topology

Since our networks will have simple 3-layer topologies, where each layer is fully connected to the layer above it, we can describe the topology with three numbers: i , h , and o . Here i represents the number of input nodes, h the number of hidden nodes, and o the number of output nodes.

2.1.3 Format for writing/reading network weights

We will number the weights so that they can be easily read and written. If there are i input nodes, h hidden nodes, and o output nodes, then there are a total of $i \times h + h \times o$ weights in the network. Let's number the input nodes as $0, 1, \dots, i - 1$, and the hidden nodes as $0, 1, \dots, h - 1$, and the output nodes as $0, 1, \dots, o - 1$.

For input node a and hidden node b , the weight connecting them is numbered $a \times h + b$. For hidden node b and output node c , the weight connecting them is numbered $b \times o + c + i \times h$.

2.1.4 Any bias must be explicit

We will NOT automatically provide a bias input layer node. However, it's generally a good idea to use a bias, so you should include that explicitly in the examples (by including an input that always has value 1) and the topology.

3 Learning problems

Use your software to do experiments on the following learning problems. For each problem, you should:

- formulate some datasets,
- choose a network topology (but it should be a 3-layer sigmoid network as described above),
- train the network using a chosen learning rate,
- and evaluate the results.

Repeat this process as necessary – not just to get better results, but also to find out where the network will break down (e.g. how small can the network be?). Evaluating your results can include:

- Error rates on test sets – are they good enough? How does the error rate compare with random guessing?
- Observing the learned weights – are they reasonable? Are they large? Small? What do they represent?
- What sort of representation is being learned by the hidden layer?
- How are the weights changing over training?
- How does the error rate change on the train/validation/test set as training continues?
- Do you observe overfitting?
- etc.

Use graphics whenever possible. Graphviz (<http://graphviz.org/>) can be useful for making pictures of network topologies.

We will use 0/1 for input and output values for these examples, but this is just a convention for this assignment. Of course, the output values for test examples will probably not be exactly 0/1, so for prediction, use a threshold at 0.5 (for the AND/XOR and shapes problems), or look for the largest output (for the autoencoder problem). If you find that the network is training too hard to get to outputs of exactly 0 or 1, you might try modifying the outputs to be 0.1/0.9, as Mitchell suggests.

3.1 Learn AND, XOR

We'll start with some simple learning tasks to make sure that things are working properly. AND is represented by the following dataset (which includes a bias as the first input):

```
1 3
0 1 0 0
0 1 0 1
0 1 1 0
1 1 1 1
```

Experiment with learning this concept. Of course, AND is pretty easy. It can be learned by a linear function. XOR cannot be learned by a linear function, so train that next:

```
1 3
0 1 0 0
1 1 0 1
1 1 1 0
0 1 1 1
```

Note that this dataset also has a bias input. Experiment with learning this concept.

For both of these (AND, XOR), try to choose a small but sufficient number of hidden nodes. These datasets are small enough that using a validation or test set is not feasible.

3.2 Learn an autoencoder

Next, let's move on to a more complicated example. Use the following dataset to train a so-called 'autoencoder' neural network. Note that this does not include a bias input.

```
8 8
1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0
0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1
```

Again, a validation set or test set is not really feasible here due to the small number of examples.

Try this with 3 hidden nodes. Try it with more hidden nodes, and with fewer hidden nodes. If you add more hidden nodes, could you increase the ability of your autoencoder to learn (e.g. add more patterns it must autoencode)? After training, what happens if you give it patterns it's never seen before, such as 1 1 1 1 0 0 0 0?

3.3 Learn to recognize shapes

Now, on to the good stuff. Train a network which will recognize the presence or absence of an axis-aligned square and a circle in an image.

The input will be a small bitmap image, encoded as a vector (row-major). For example, the image might have 30×30 pixels (size is up to you), and the top row of the image is encoded first, followed by the next row, etc. To keep things simple, just make each pixel have values 0 (black) or 1 (white).

The output will be a vector with two elements. The first output will be 1 if there is a square in the image, and 0 if there is no square in the image. The second output will be 1 if there is a circle in the image, and 0 if there is no circle in the image.

Each image should contain zero or one square, and zero or one circle. Any shapes should be colored 1 (white) and filled in on a 0 (black) background. The shapes may appear anywhere in the image, but they should be fully contained in the image, and they should not overlap if they both appear in an image.

Here is a visual example of an image that you might feed through the network which contains both a circle and a square. Note that this example is a matrix for illustration purposes, but the input to your network will be a row-major vector representation, preceded by the target vector.

```
1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 0 0 0 0 1 0 0 0 0 0 0
1 1 1 1 1 0 0 1 1 1 1 1 1 0 0 0
0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0
0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0
0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0
0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0
0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

You can report the error rate for each output unit individually, but it is interesting also to consider the error rate of each type of image (none/circle/square/both).

3.3.1 Network topology

Obviously, this network will have 2 outputs and as many inputs as there are pixels in the image. However, you should try different numbers of hidden nodes to find what works best.

3.3.2 Example image generator

You can find a simple generator for sample images here: http://cs.baylor.edu/~hamerly/courses/5325-09s/assignments/a3_make_image.cpp You can use this to generate your own training, validation, and test images.

3.3.3 Generalization power

For further study, after training a good network, you might try some test images which violate the original assumptions to see how the network generalizes. For example, the shapes might overlap, or there might be more than two shapes in an image, or the shapes might be outlined (but not filled), etc. Or you might try it on some image which is completely different than the sort of image it has been trained on.

3.4 Going further

For some extra credit, train a network to identify a concept in real black-and-white photos, such as ‘there is a person in this image,’ or to identify people’s faces (where the test set has unseen photos of the same people as in the training set), etc.

You could also try adding additional hidden layers (if you think it is necessary), or adding momentum, or adding weight decay to the training.

4 Evaluation criteria

The following sections will be used for grading your assignment (these adapted from Charles Elkan’s grading criteria):

Category	Points
Introduction insightful on background and motivation	3
Precise and reproducible description of technical work	3
Sensible implementation decisions (code reuse, choice of PL, etc.)	2
Sufficiently conclusive experiments (large datasets, lesion studies, etc.)	3
Well-designed, well-described, reproducible experiments	3
Understandable presentation of results, preferably graphical	3
Clear, correct analysis of results (including statistical significance)	3
Insightful discussion of all major experimental results	3
Exciting and useful results with general applicability	3
Appropriate organization (logical, not chronological) and easy-to-read plain writing style	3
Correct spelling, grammar, and choice of words	3
Total	32

A lesion study means taking parts out of the system you are using to identify what will cause a system’s performance to degrade. For this assignment, you need not perform statistical significance tests, but careful analysis is still important.