5-31-2018

A Neural Network Model for Classifying Bubble-Based Instructor Evaluations, and an Accompanying Web Portal

Jason Held
University of Massachusetts Boston

A NEURAL NETWORK MODEL FOR CLASSIFYING BUBBLE-BASED INSTRUCTOR EVALUATIONS, AND AN ACCOMPANYING WEB PORTAL

A Thesis Presented

by

Jason Held

Submitted to the Office of Graduate Studies, University of Massachusetts Boston, in partial fulfillment of the requirements for the degree of

Master of Science

May 2018

Computer Science Program

A NEURAL NETWORK MODEL FOR CLASSIFYING BUBBLE-BASED INSTRUCTOR EVALUATIONS, AND AN ACCOMPANYING WEB PORTAL

A Thesis Presented

by

Jason Held

Approved as to style and content by:

Swami Iyer, Lecturer
Chairperson of Committee

Dan Simovici, Professor
Member

Ming Ouyang, Assistant Professor
Member

Dan Simovici, Program Director
Computer Science Program

Peter Fejer, Chairperson
Computer Science Department

ABSTRACT

A NEURAL NETWORK MODEL FOR CLASSIFYING BUBBLE-BASED INSTRUCTOR EVALUATIONS, AND AN ACCOMPANYING WEB PORTAL

May 2018

Jason Held,
B.S., University of Kansas
M.S., University of Massachusetts Boston

Directed by Lecturer Swami Iyer

We propose a neural network model for classifying bubbles (circles) used in instructor course evaluations. The model is trained on prior (labeled) objects consisting of bubbles and general text. The trained model is then used to determine the positions of bubble answer options on a given evaluation form. A Web portal accompanies the classification system and facilitates management of the network and analysis of the results. The departmental staff will upload an unevaluated form per course and the system will execute the neural network model on it; application logic will be responsible for ensuring data persistence of the bubble positions in addition to student long-form question answers. Once the departmental staff uploads an electronic copy of the filled out evaluations for a course into the portal, the application server will aggregate the results based on the output from the neural network. The instructor for the course is able to view the evaluation results once granted access by the departmental staff.

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

CHAPTER 1

INTRODUCTION

The world of academia continues to have key areas that have not been revolutionized or enhanced through the process of solutions involving computing. One such area of concern is the process of conducting and processing instructor course evaluations. Universities (such as the University of Massachusetts Boston) have a loosely coupled system for providing students with the ability produce a completed evaluation survey for the lecture they've taken that semester. The first step for creating an evaluation form, is to generate a document per-department for which questions are relevant for a course-lecturer evaluation, and which sorts of answers will be meaningful. The next step is actually very generic in its current form: a Scantron sheet which the student will use to mark down their answers to the questions on the per-department form.

The students return both sheets and the Scantron surveys will then be processed by a Scantron evaluation machine. The results of this will then be recorded by the University and may be used in decisions regarding the course and instructor at a future date. This thesis document is not concerned with any decisions that the university faculty may enact upon; having a high-level picture of the steps involved, though, is crucial to the problem space as well as the solution that this document proposes.

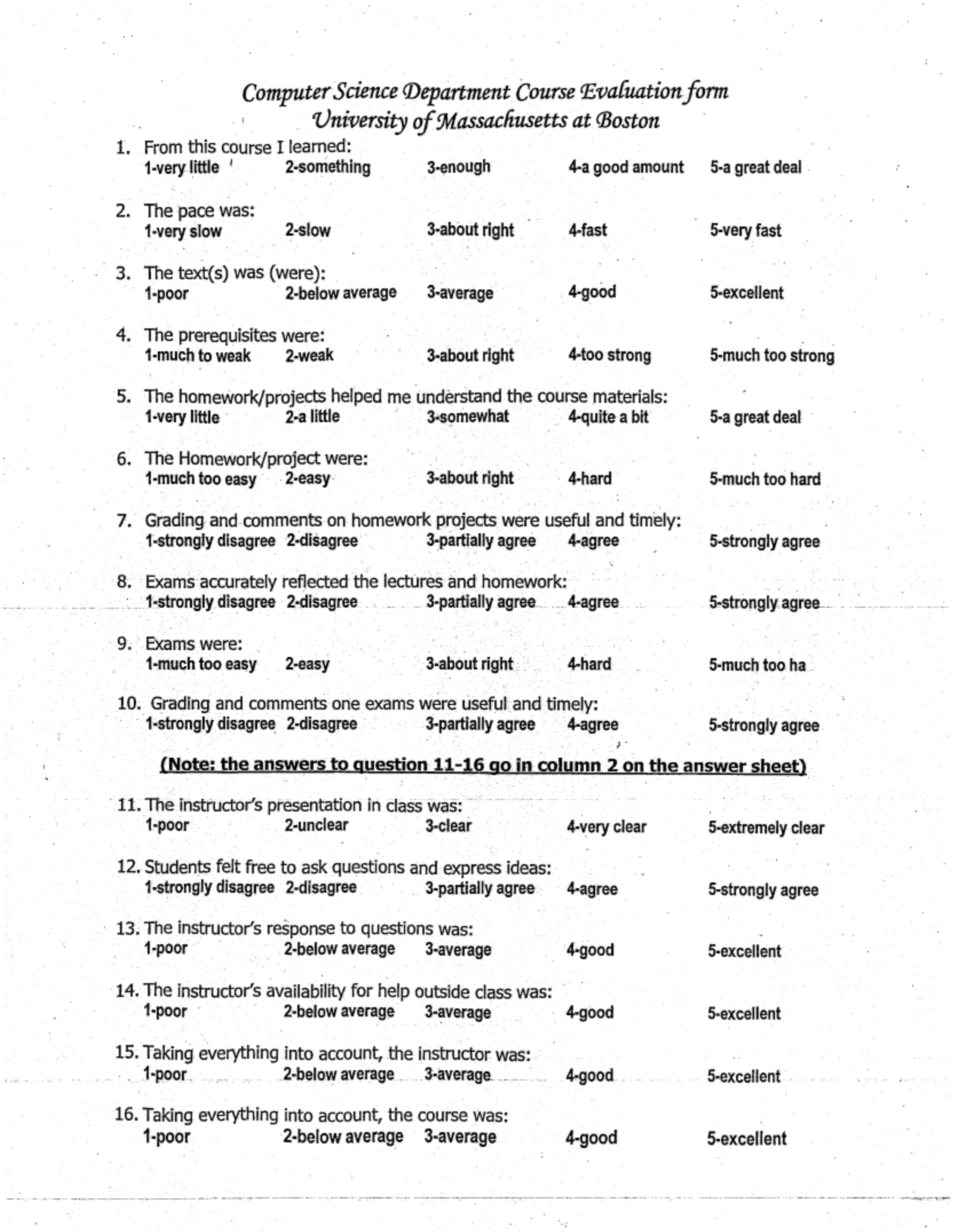
Current Workflow

The first step, as noted above, is for the faculty to determine which questions and answer choices may be relevant to a given department. The questions probably would not change often, nor the applicable choices per question. However, that is not a strict rule.

A common practice for encoding a set of choices for answer to a question is to assign a value (such as letters or numbers). For instance, 1. Always, 2. Sometimes.

The student then pulls up their Scantron document, finds the appropriate row number corresponding to the given numbered question (Question 10, for instance), and then fills in the associated bubble or circle on the row. Each student must use a graphite or lead-based pencil, given the restrictions of the Scantron machine.

Depending on the administration, there may be a further step that the department must perform before they use the Scantron machine. This would include a re-transcription process, whereby humans would take the filled out forms, and on a new sheet, re-mark the answers. This sort of addition to the workflow would be used for an accuracy enhancement. Students may, in the process of answering a question, fill out more than one option, and try to erase the one they didn't want, leaving another as a darkened option. However, perhaps the Scantron machine would be unable to determine which answer was intended (depending on the shades of the filled in ones). If indeed there is a human re-transcription aspect in the flow, there is the possibility that the wrong choice is re-transcribed, whether by the original author's marks, or by a mistake on behalf of the second marker. This document is not particularly concerned with the error rates and specific implementation details affiliated with the Scantron products. We detail this process because we are proposing a solution which is more efficient and scalable on the whole over the current approach.



Computer Science Department Course Evaluation form
University of Massachusetts at Boston

1. From this course I learned:
1-very little | 2-something | 3-enough | 4-a good amount | 5-a great deal

2. The pace was:
1-very slow | 2-slow | 3-about right | 4-fast | 5-very fast

3. The text(s) was (were):
1-poor | 2-below average | 3-average | 4-good | 5-excellent

4. The prerequisites were:
1-much to weak | 2-weak | 3-about right | 4-too strong | 5-much too strong

5. The homework/projects helped me understand the course materials:
1-very little | 2-a little | 3-somewhat | 4-quite a bit | 5-a great deal

6. The Homework/project were:
1-much too easy | 2-easy | 3-about right | 4-hard | 5-much too hard

7. Grading and comments on homework projects were useful and timely:
1-strongly disagree | 2-disagree | 3-partially agree | 4-agree | 5-strongly agree

8. Exams accurately reflected the lectures and homework:
1-strongly disagree | 2-disagree | 3-partially agree | 4-agree | 5-strongly agree

9. Exams were:
1-much too easy | 2-easy | 3-about right | 4-hard | 5-much too ha

10. Grading and comments one exams were useful and timely:
1-strongly disagree | 2-disagree | 3-partially agree | 4-agree | 5-strongly agree

(Note: the answers to question 11-16 go in column 2 on the answer sheet)

11. The instructor's presentation in class was:
1-poor | 2-unclear | 3-clear | 4-very clear | 5-extremely clear

12. Students felt free to ask questions and express ideas:
1-strongly disagree | 2-disagree | 3-partially agree | 4-agree | 5-strongly agree

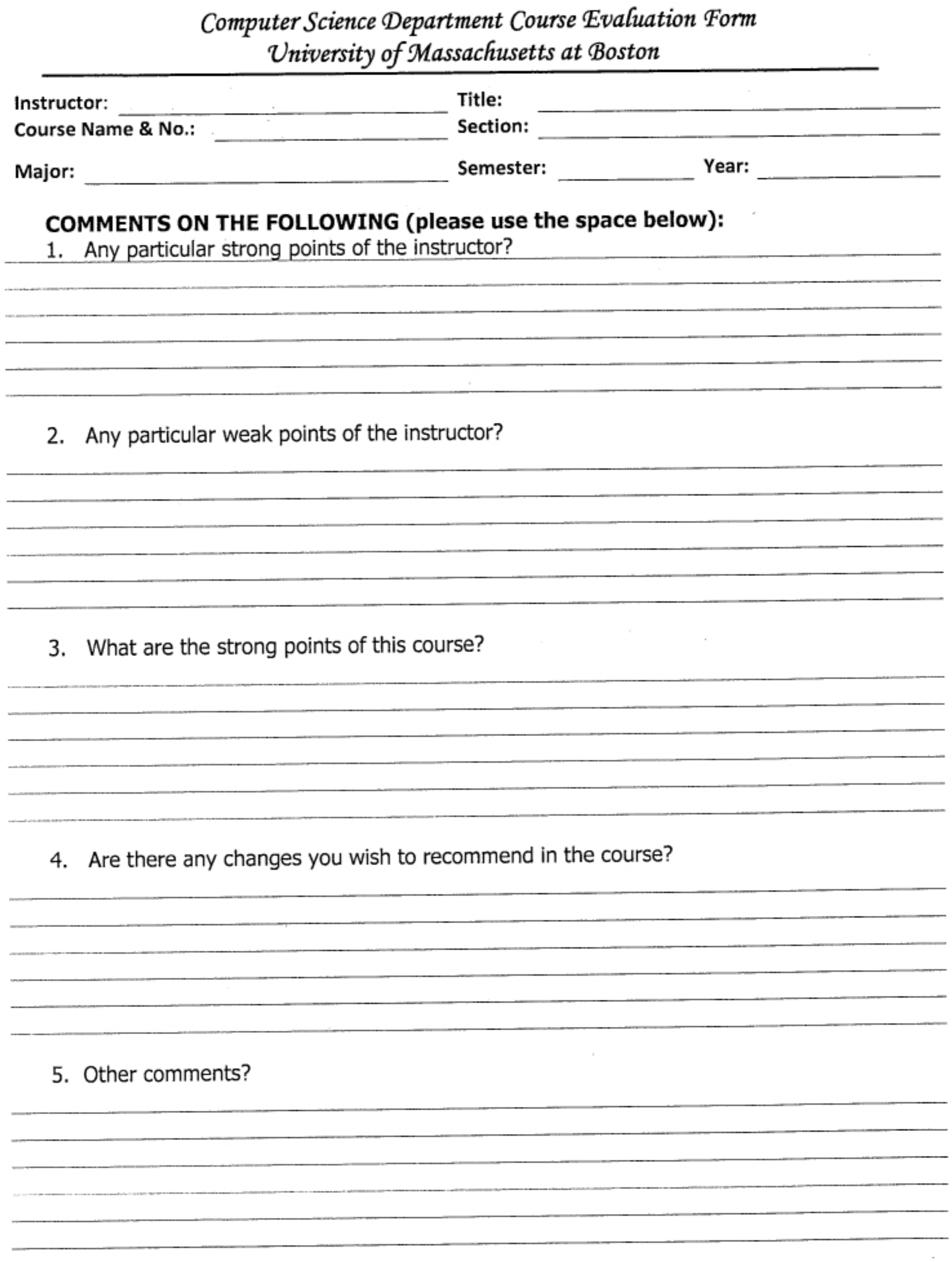
13. The instructor's response to questions was:
1-poor | 2-below average | 3-average | 4-good | 5-excellent

14. The instructor's availability for help outside class was:
1-poor | 2-below average | 3-average | 4-good | 5-excellent

15. Taking everything into account, the instructor was:
1-poor | 2-below average | 3-average | 4-good | 5-excellent

16. Taking everything into account, the course was:
1-poor | 2-below average | 3-average | 4-good | 5-excellent

Figure 1: Current Answer Choices



Computer Science Department Course Evaluation Form
University of Massachusetts at Boston

Instructor: ________ **Title:** ________
Course Name & No.: ________ **Section:** ________
Major: ________ **Semester:** ________ **Year:** ________

COMMENTS ON THE FOLLOWING (please use the space below):

1. Any particular strong points of the instructor?

2. Any particular weak points of the instructor?

3. What are the strong points of this course?

4. Are there any changes you wish to recommend in the course?

5. Other comments?

Figure 2: Current Long Form

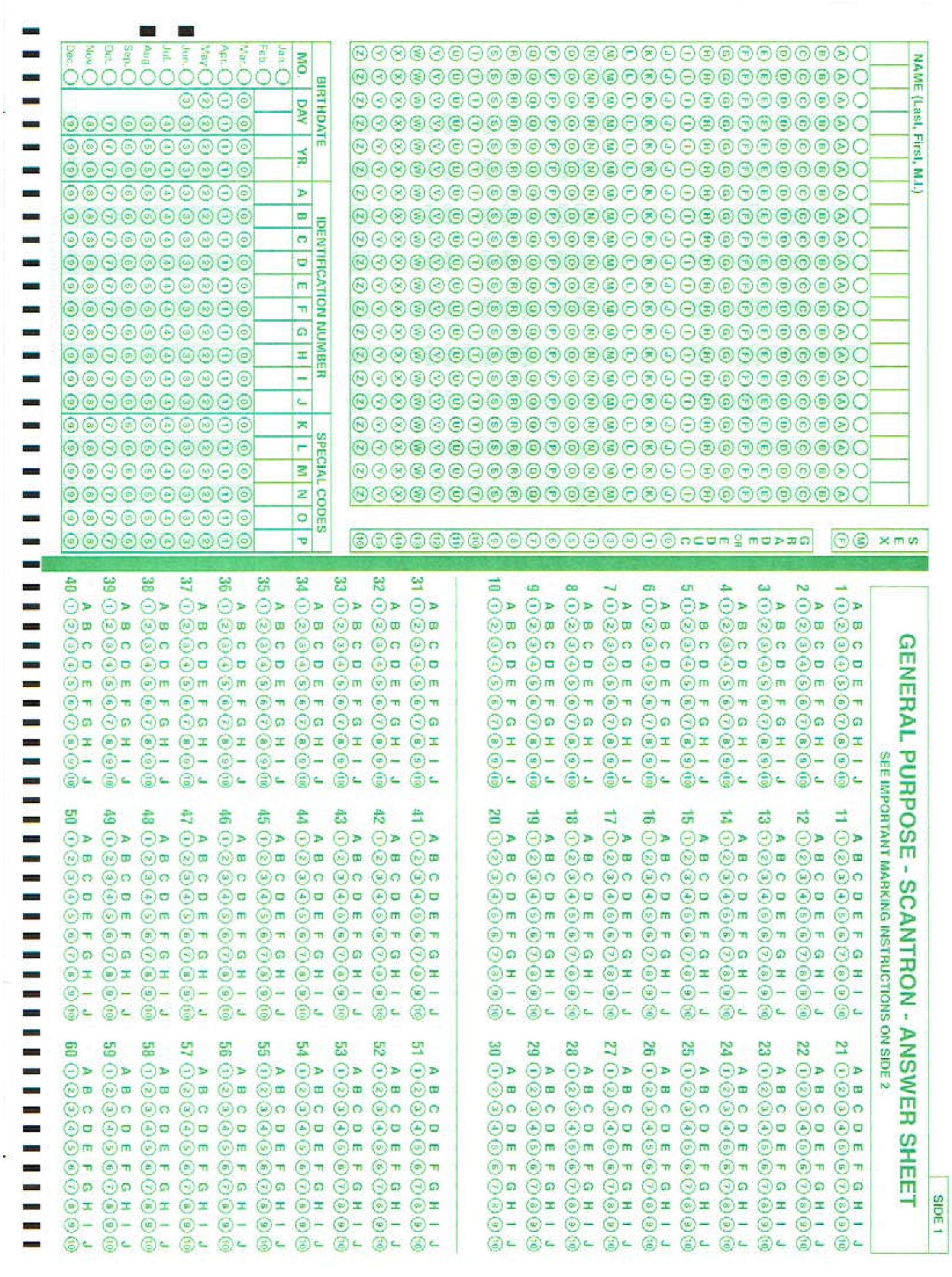
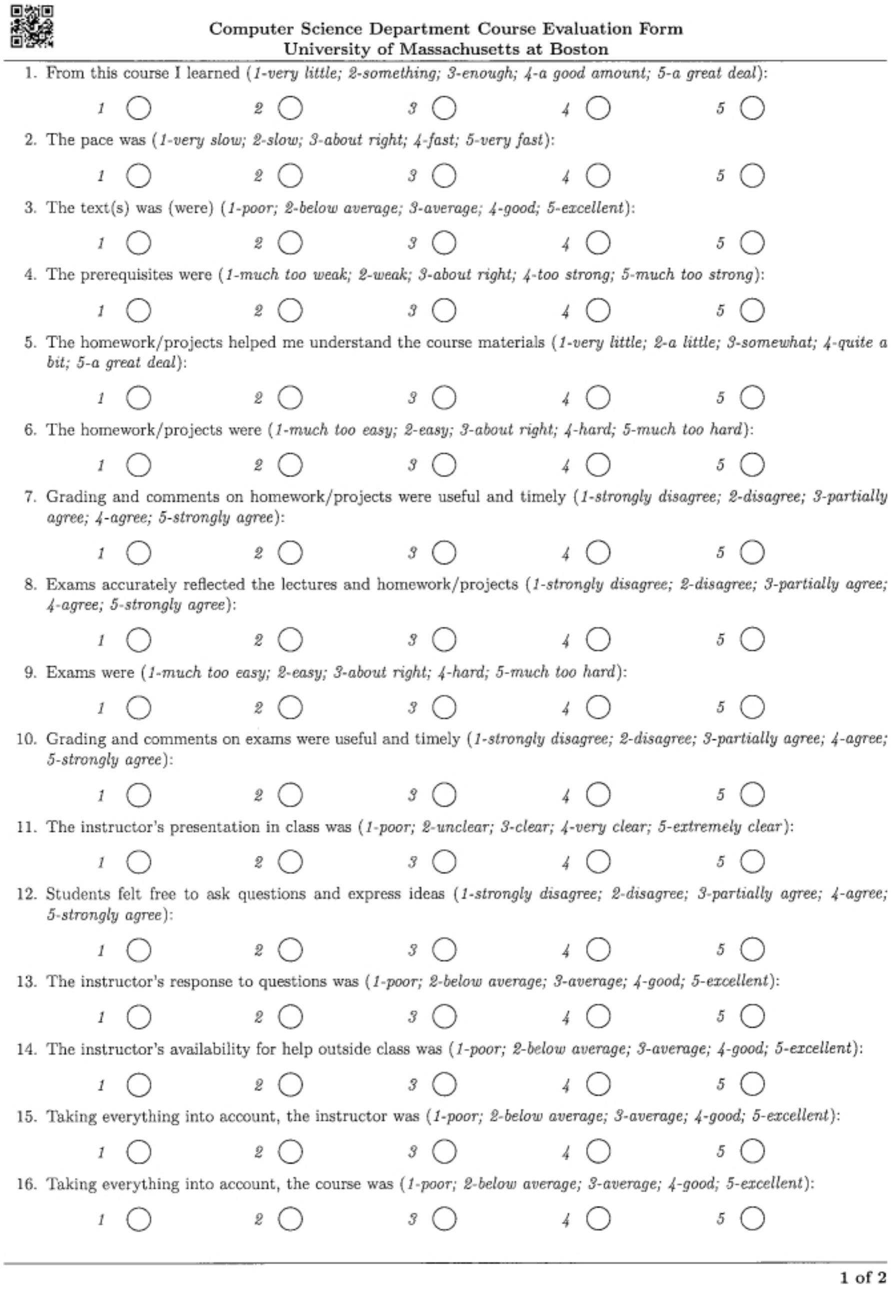

GENERAL PURPOSE - SCANTRON - ANSWER SHEET
SEE IMPORTANT MARKING INSTRUCTIONS ON SIDE 2
SIDE 1
NAME (Last, First, M.I.)
SEX
M
F
GRADE OR EDUC
BIRTHDATE
MO.
DAY
YR.
Jan.
Feb.
Mar.
Apr.
May
Jun.
Jul.
Aug.
Sep.
Oct.
Nov.
Dec.
IDENTIFICATION NUMBER
SPECIAL CODES
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P

Figure 3: Scantron Answer Sheet



Computer Science Department Course Evaluation Form
University of Massachusetts at Boston

1. From this course I learned (*1-very little; 2-something; 3-enough; 4-a good amount; 5-a great deal*):

   1 ○ *2* ○ *3* ○ *4* ○ *5* ○

2. The pace was (*1-very slow; 2-slow; 3-about right; 4-fast; 5-very fast*):

   1 ○ *2* ○ *3* ○ *4* ○ *5* ○

3. The text(s) was (were) (*1-poor; 2-below average; 3-average; 4-good; 5-excellent*):

   1 ○ *2* ○ *3* ○ *4* ○ *5* ○

4. The prerequisites were (*1-much too weak; 2-weak; 3-about right; 4-too strong; 5-much too strong*):

   1 ○ *2* ○ *3* ○ *4* ○ *5* ○

5. The homework/projects helped me understand the course materials (*1-very little; 2-a little; 3-somewhat; 4-quite a bit; 5-a great deal*):

   1 ○ *2* ○ *3* ○ *4* ○ *5* ○

6. The homework/projects were (*1-much too easy; 2-easy; 3-about right; 4-hard; 5-much too hard*):

   1 ○ *2* ○ *3* ○ *4* ○ *5* ○

7. Grading and comments on homework/projects were useful and timely (*1-strongly disagree; 2-disagree; 3-partially agree; 4-agree; 5-strongly agree*):

   1 ○ *2* ○ *3* ○ *4* ○ *5* ○

8. Exams accurately reflected the lectures and homework/projects (*1-strongly disagree; 2-disagree; 3-partially agree; 4-agree; 5-strongly agree*):

   1 ○ *2* ○ *3* ○ *4* ○ *5* ○

9. Exams were (*1-much too easy; 2-easy; 3-about right; 4-hard; 5-much too hard*):

   1 ○ *2* ○ *3* ○ *4* ○ *5* ○

10. Grading and comments on exams were useful and timely (*1-strongly disagree; 2-disagree; 3-partially agree; 4-agree; 5-strongly agree*):

    1 ○ *2* ○ *3* ○ *4* ○ *5* ○

11. The instructor's presentation in class was (*1-poor; 2-unclear; 3-clear; 4-very clear; 5-extremely clear*):

    1 ○ *2* ○ *3* ○ *4* ○ *5* ○

12. Students felt free to ask questions and express ideas (*1-strongly disagree; 2-disagree; 3-partially agree; 4-agree; 5-strongly agree*):

    1 ○ *2* ○ *3* ○ *4* ○ *5* ○

13. The instructor's response to questions was (*1-poor; 2-below average; 3-average; 4-good; 5-excellent*):

    1 ○ *2* ○ *3* ○ *4* ○ *5* ○

14. The instructor's availability for help outside class was (*1-poor; 2-below average; 3-average; 4-good; 5-excellent*):

    1 ○ *2* ○ *3* ○ *4* ○ *5* ○

15. Taking everything into account, the instructor was (*1-poor; 2-below average; 3-average; 4-good; 5-excellent*):

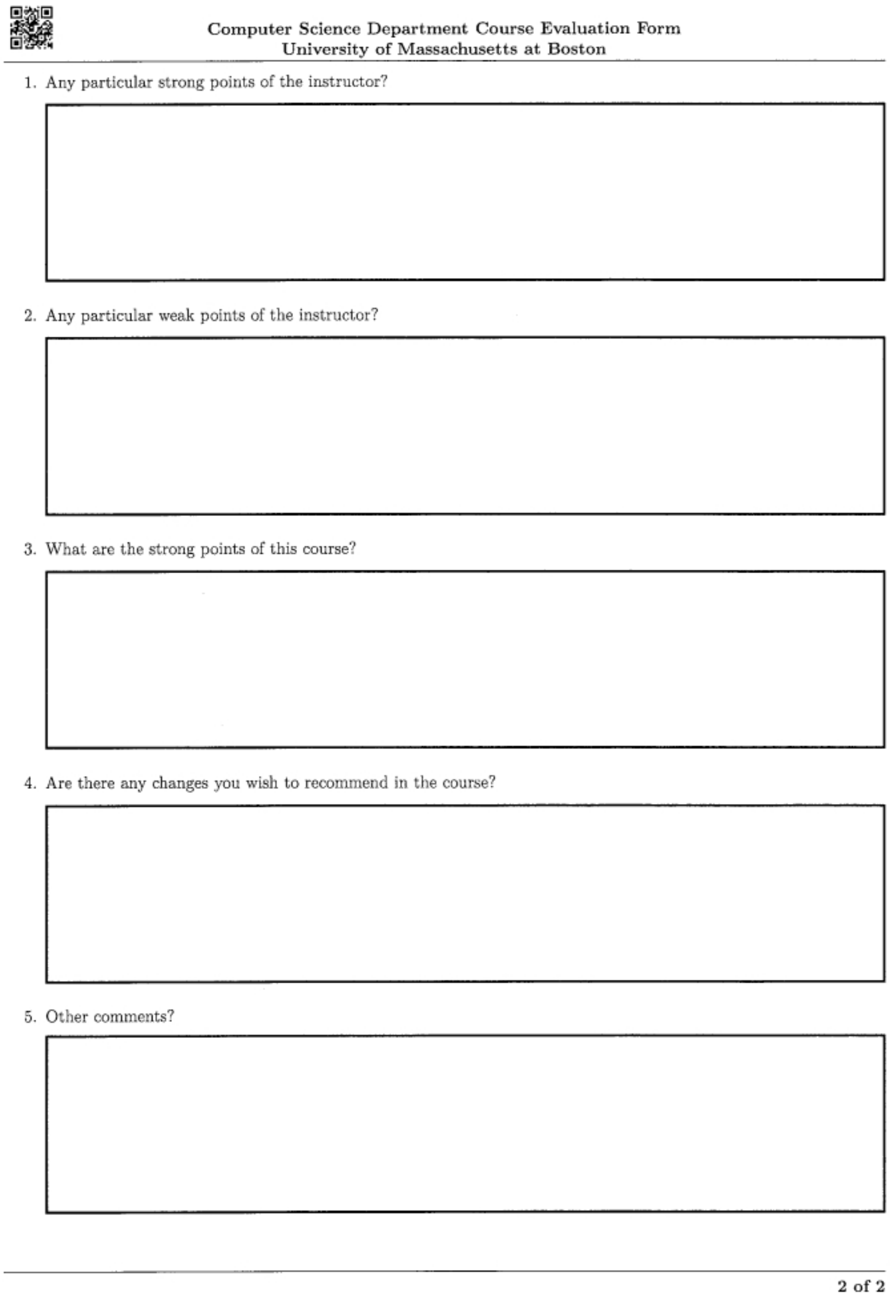
    1 ○ *2* ○ *3* ○ *4* ○ *5* ○

16. Taking everything into account, the course was (*1-poor; 2-below average; 3-average; 4-good; 5-excellent*):

    1 ○ *2* ○ *3* ○ *4* ○ *5* ○

1 of 2

Figure 4: New Style Evaluation Answer Choices



Computer Science Department Course Evaluation Form
University of Massachusetts at Boston

1. Any particular strong points of the instructor?

2. Any particular weak points of the instructor?

3. What are the strong points of this course?

4. Are there any changes you wish to recommend in the course?

5. Other comments?

2 of 2

Figure 5: New Style Evaluation Long Form Answers

The current approach will (regardless of a expensive re-transcription), will then have basic statistics generated for the faculty to use for the evaluation itself. Given the loose-coupling between the machine and the statistic gathering, though, the University would have no native way to perform any further advanced analysis on the responses to the evaluation. That is, there would have to be an additional implementation for instructor or course evaluation trends and metrics. These would be very useful to the University, but would require storing the results from the external Scantron into a database; this cost may be prohibitive, both from a human perspective (manual data entry), as well as updating the technology itself over time (software packages, fixing any bugs, security, etc). Of course, having a new, modern interface to this would not overcome the presence of bugs, but depending on the implementation of it (which we will discuss later), the resolution for fixing issues would have an easier and cheaper workflow.

Our Proposal

The solution we're targeting is to use machine learning on the evaluation forms. This type of change to the process would make for a very different workflow for the university. One modest improvement, which admittedly would alter a key piece from before, is to only have the one form – no additional Scantron sheet. The questions and answers would be on the same sheet, and the student would fill in the bubble explicitly where the question is.

This change would make the form incompatible with the Scantron solution. The Scantron wouldn't know where to look for answers, nor is there any guarantee that the bubble size itself would be exactly the same, even if the position was similar.

We thus suggest a new requirement to this process. In place of the Scantron, we will use a normal computer itself no advanced lighting and image sensing software or hardware. The specific solution is to use machine learning. We will train a neural network and with application-logic, be able to discern the most likely answers to questions that the students marked. An added side-effect here is that once the network is deemed adequate, there will be little to no need for a secondary transcription, if that is the University's current practice.

The software that is built will take the evaluated forms, determine the answers, and then will store the results into a database. This database will sit behind a web framework which will allow management of the data, as well as serve to perform more advanced data analysis than the current solution allows.

A side effect of being able to combine everything into one form is that the number of pages of paper that must be used and printed is equal to the number of students.

This new system's general intention is to make the whole process both easier and more accurate on the general use. The administration staff will interface with the system through a web-browser and PDF scanner.

The process for creating a neural network system is on the whole fairly well-defined. In the world of machine learning, there are many ways to implement a solution that works. The notion of "works", though is translated to approximated. We will discuss this concept further later.

Machine learning and neural networks in general are becoming more standard, in the industry, as well as for expensive theoretical applications.

Producing a single form is beneficial in many regards, aside from a consumption standpoint. The less that a department is required to make for the evaluation process to work, the easier it will be. The format will thus change to incorporate this, but additionally, there will be other restrictions. One such is that the form must have a decent amount of spacing between the circles, and between the circle and the question text. The neural network will be trained against a particular set of data, so it is possible that less space would still be appropriate, but nonetheless it is would be recommended to follow a set of document format guidelines. These guidelines will not be detailed much in this document, however, we will review other portions of it later which will affect the system from the administration's perspective.

Keeping a general guideline for production would allow additional departments to use this as a template to more quickly allow them to integrate with this newer system. The questions could change, as well as the number of answers. It only depends on the context of the department. It is possible that the neural network would have to be re-trained per department, however that may only depend on how much they deviate from the standard document template. It should not drastically affect any part of the flow, but it would be a recommendation for use with the system that they not change anything substantial. That is, if they decide instead to use boxes (squares) for answers, then the network would most likely need to be re-trained. Or, if they significantly altered the proportions of the bubble.

Research and Implementation

We will discuss in detail the background and topics of neural networks. This includes the mathematics and challenges that come with constructing a network, and a sample of the problems that can and have been solved with them.

Additionally, we will cover how we created a neural network to introduce our proposed solution to the current approach documented in this chapter. We will enumerate the network setup, the results we have attained, and the interface for the department through the web portal.

CHAPTER 2

NEURAL NETWORKS

Background

In the 1980s and 1990s many research organizations and companies were looking for ways to teach machines how to learn. This was not a new area, so much as with more powerful hardware and software came ways to realize more meaningful ways to analyze data and process it to learn something. Researchers had already studied the brain and how it could learn as well as general biological processes. What they determined was that we could emulate a network of neurons ourselves by the use of a computer program.

The types of problems a researcher might look at could be the likelihood of a person contracting heart disease, or even something as entertaining which football team might win the super-bowl next.

The data scientist or software developer will execute the network by providing it an input that is representative of what they are looking to retrieve an answer for from the network. The network takes this set of inputs, and will output what might be called a "prediction." This answer is made from the prior training of the network. That is, the network was told what the type of data it is told to look at beforehand, and it will, with the use of linear algebra and other mathematics, learn.

Network Construction

Neural networks implement a series of layers of virtual neurons. They are trained such that certain combinations of inputs will be able to make a neuron fire. As the chains of firings work through the network, eventually they terminate at the output, which will generally be used as an object classifier. This is a similar way to how our eyes have several layers; these enable edge-detection of images, and then discrete pieces interior to the object. In that way, for instance, a human face could be recognized as such.

Neural networks can be adopted in many different strategies to solve a problem, often to classify objects.

There are two general ways to train a neural network. One method which may allow a network to learn interesting aspects of its input is called unsupervised learning. In this way, the executor of the network (a human or client program), will feed as much data into the network as possible, and allow the network to learn aspects of the input as it sees it. Eventually it should be able to determine interesting patterns of the input that can be used by scientists to see how else they might wish to tackle a problem.

The other type is called supervised learning. In this strategy, the network still pushes samples through the network, but at the output of it, the classification is compared against a given label for that input. If a network was supposed to determine if something was a banana or orange, then if a given input was a banana, the classification would be compared against that label. If it was wrong, then there will be a notable cost associated (and error rate). The job of the data scientist or software developer will then be to implement ways in which to lower that error rate, which in turn will help the network to become a stable solution to a given classification problem.

It can be challenging to train a neural network; one of the leading time-intensive areas of training is determining the specific configuration for how to run the network is in choosing the hyper-parameters. This includes any regularization of the output (penalizing outlying results), the learning rate (how fast or slow does the network learn per input), the number of hidden layers, and the cardinality of nodes per layer. We will discuss these later in this chapter.

Network Nodes

Perceptron

Perceptrons may take multiple [weighted] inputs to produce a given single result in the output. This can be any sort of problem, from simple decision making (should a given person drink a cup of coffee), to more advanced scenarios. However, the output is either a 1 or a 0, and thus there is no way to determine partial matches. Perceptrons are powerful, but if the consumer of the network must have multiple input parameters to the network, and certain examples cannot get above a threshold to hit 1, then the result would be 0.

Figure 6 shows a general neuron node, which the Perceptron also takes the form of. However, all nodes for neural network design follow the same structure. In that figure, the $+1$ is provided as the bias the neuron, while each edge has a (unseen) weight assigned to it.

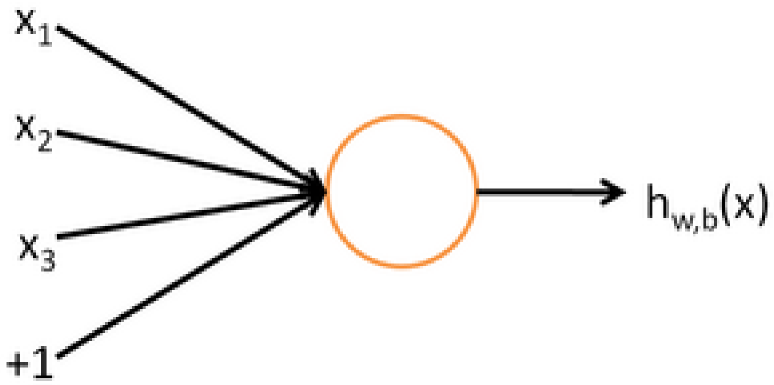


Figure 6: Neuron

[Wik]

Equation 2.1 notes the formula for a perceptron. *w* is the vector of weights against the input vector *x*, while *b* is the applied bias.

$$\sigma(x) = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases} \tag{2.1}$$

Neuron

Neurons are an enhanced form of a perceptron. These neurons have a larger scope of possible output values, and the algorithm to compute it is able to adjust to change in a much more subtle way. In this document we will focus mostly on the Sigmoid neuron.

Perceptrons may only take on values of 0 or 1, so to attain more granularity into the results, we would need more than simple 0 or 1s. Sigmoid neurons instead have a larger breath of valid values. A sigmoid neuron is similar to a percepton, but the value of a sigmoid is a continuous distribution from 0 to 1. Perceptrons, due to how the weight calculation occurs, are not particularly good at allowing a change in one direction (classifying one thing), while not ruining the classification of another. With a sigmoid, it can hold a wider variety of values in the middle of the network processing. Because many problems do not always allow for exact matches of classification problems, a sigmoid can be a great fit for these.

According to Nielson [Niea], although sigmoidal neurons are similar to perceptrons, they can be modified so that small changes in their weights and bias do not drastically affect the output, unlike perceptrons. Given this property, we can use this to build better learning models (those that can learn in smaller intervals) into a neural network.

Equation 2.2 shows the formula for a sigmoid neuron.

$$\sigma(w,b,x) = \frac{1}{1+\exp(-\sum_j w_j x_j - b)}. \tag{2.2}$$

x is the input data, w the weight, b the bias. In simpler terms these can be represented as the variable z, in which case it is $\frac{1}{1+e^{-z}}$

Learning Activation Functions

We have so far talked about the Sigmoid activation function. However, there are a couple others that may be used. [GB10] performed research on the challenges in training deep feed forward neural networks. They used the following major types of activation functions:

- Sigmoid (2.2)
- Softsign (2.3)
- tanh, or Hyperbolic Tangent (2.4)

In their testing, they found that Sigmoid, with standard initialization, does not perform as fast as the other functions, and may not find optimal local minima in the space.

They state that the Softsign function tends to have better performance potentially due to its gentler non-linearity.

Equation 2.3 shows the formula for the Softsign Activation function. In this equation, z is the computation of the weights against the input, along with the bias.

$$\sigma(z) = \frac{z}{1+|z|} \tag{2.3}$$

Equation 2.4 shows the formula for the tanh Activation function. In this equation, the terms are same, with an additional usage of hyperbolic sin and cos functions.

$$\sigma(z) = \frac{\sinh(z)}{\cosh(z)} \tag{2.4}$$

Their research also showed that with tanh networks, if they used their proposed normalized initialization then they could see how well the transformations of the network's learning was maintained over activations and the gradients.

This document is not concerned with specific applications of these individual funtions and the results of such, but mentions these to give background on the range of functions and their possible inefficiencies or drawbacks.

Cost Functions

We will discuss the following cost functions, also known as error functions. The need for these functions is to help the neural network determine, on a given feed forward, how close it was to attaining the correct result for a given input sample. The smaller the value becomes, the closer it believes it is. The value from the output of this function will then be used to adjust the weights and biases for the network accordingly.

It is possible for the network to find a local minima in the error space such that the cost does not change much, but it is very small. That is, the network may stall in the learning process if it is unable to find a better minima. There are numerous cost functions and using an alternative may help, but there are several other parameters that one may use to train the network. We will discuss these further later in the document.

Equation 2.5 shows the formula for the quadratic cost function. This is also known as the mean squared error.

$$C = \frac{1}{2n} \sum_x \|y(x) - a\|^2. \tag{2.5}$$

In this equation, w is the input weight, and b is the input bias, to the neuron. x itself is the actual data input to the network, while y is the network itself, taking x as input. a is the labeled vector of outputs for this run of x (the labeled values we give during training). That is, the cost function will compute the difference of the real output against what it is supposed to be. x in particular is an item in the set of all inputs of the data. So we will go through each x, take the difference of the actual versus expected, squaring each, sum those, and then divide by $2n$.

The cross-entropy cost function is an improvement over the quadratic cost function.

Equation 2.6 notes the formula for the cross-entropy function.

$$C = -\frac{1}{n} \sum_x \left[y(x) \ln a + (1 - y(x)) \ln(1 - a) \right], \tag{2.6}$$

In this equation, n is the full size of the dataset, while the remaining terms are the same as those in Equation 2.5.

[Niea] characterizes the Cross-entropy function as a form of measuring the "surprise" that a neuron will receive when it finds out that its wrong. The greater the surprise, the faster the change in learning. It is this behavior which is what allows this function to perform better than the quadratic cost function.

Network Architectures

Here we will discuss common types of network configurations and their associated learning functions and workflows.

Figure 6 shows a percepton and 8 shows a neuron network with three layers, one of which is used specifically for learning. This is called a hidden layer, and we will discuss this further. Figure 8 is an example configuration of a deep neural network which shows an output layer that may be used for multiple object classification (e.g. more than one node on the output).

Layer Configurations

As shown in the above figures, a neural network will in practice consist of an input layer, some number of hidden layers, and an output layer.

The input layer to a neural network is a special layer. It is the actual data that the network is built for that is fed in here. It is up to the designer of the network to determine what information is passed into it. When building for image recognition, for instance, a common practice is to spread the pixels of the image into a 1 dimensional array, and pass each of them into a given node of the input layer. We will discuss this our approach in the next chapter.

The hidden layer(s) are put in place to induce the network into learning something interesting and valuable about the data it is given. The network will be trained to classify the objects it is given. Adding additional hidden layers into this section may give the

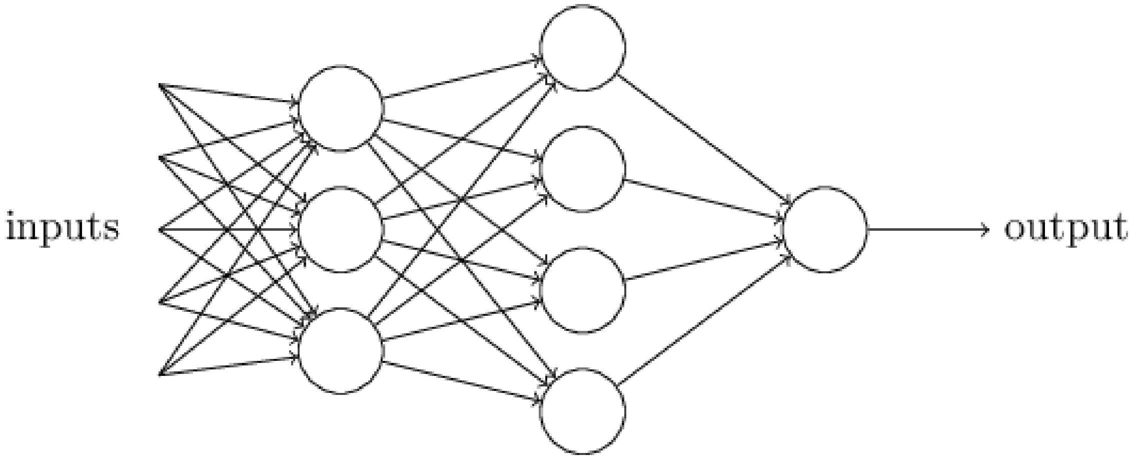


Figure 7: Neuron-Based Network

[Niea]

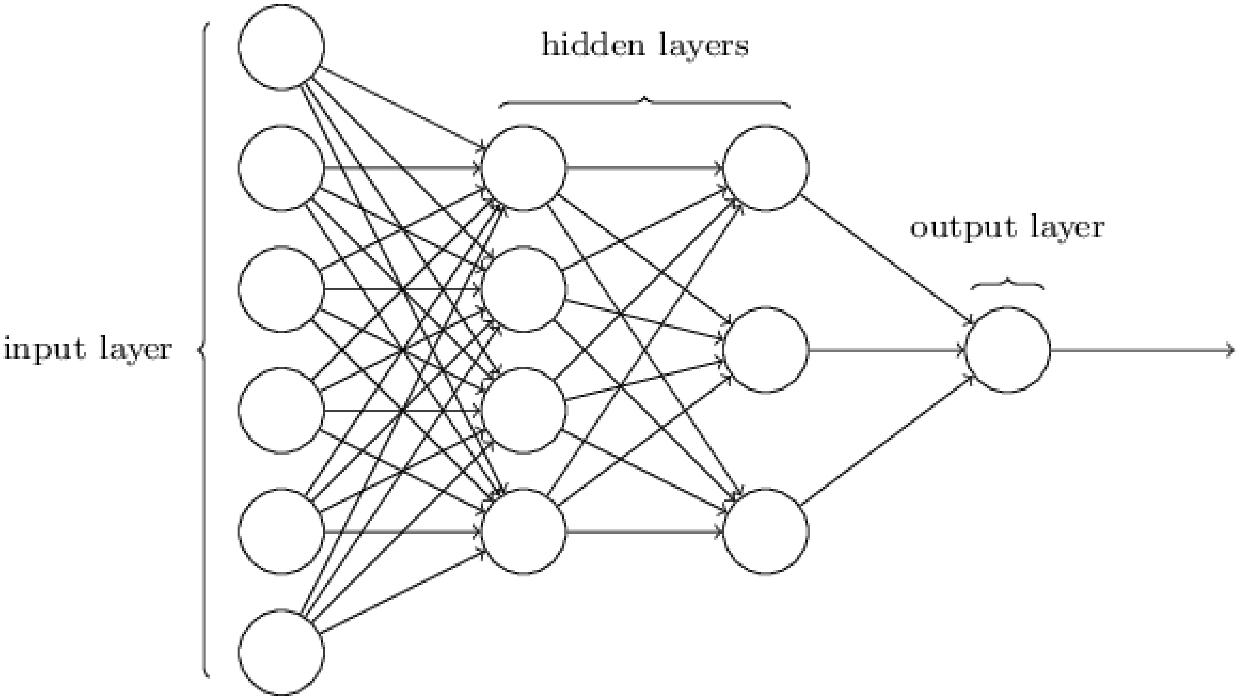


Figure 8: Deep Neural Network with Classifications

[Niea]

network further power for classification. The creator of the network will determine how many nodes are in each hidden layer – which can be equal, less, or greater than the number of nodes in the hidden layer.

Finally, the output layer is responsible for classifying objects which the hidden layer has begun to separate out. The cardinality of this layer (the number of nodes) for a supervised network will be equal to the number of discrete object types that the creator needs to recognized; this is known ahead of time, and as such, the creator will specify this statically.

Gradient Descent

Gradient descent is a learning algorithm. The algorithm will invoke the cost function and apply that result into a change that will alter the weights and biases for a given network layer. The goal of gradient descent is to descend into the global minimum of the error space. If it can find the global minimum, then it will have reduced the cost (error) to the smallest it will be. For a neural network that classifies objects, we could then say that for a given object the network should be able to accurately classify it (a banana should be a banana).

However, it requires looking at the entire batch of sample data per epoch (number of runs to train) all at once. For large enough samples (think 50,000 or more samples), this can severely slow the runtime of a computer algorithm implementing the training of a neural network.

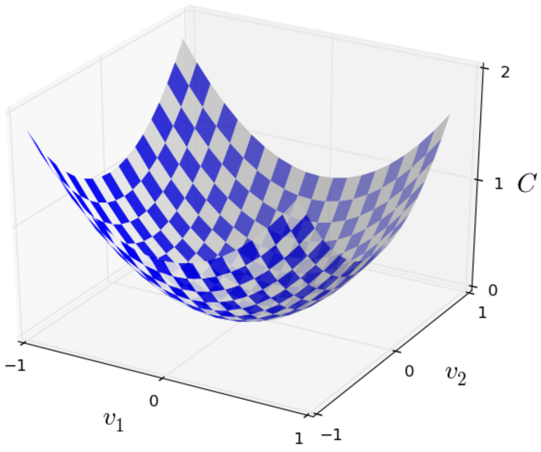


Figure 9: Gradient Descent Valley

[Niea]

Stochastic Gradient Descent

Instead, a common practice is to use stochastic gradient descent, also referred to as SGD. It is used by defining a mini-batch size while performing gradient descent. That is, instead of using the full sample size, the trainer of the network will choose some small size that can partition the sample size into batches to speed up the learning processing.

Whereas one would think needing to look at all samples altogether when learning would be the optimal approach, in many cases that may be quite inefficient. We could easy the training process (make it faster), but using a mini-batch approach instead of the full batch [Niea]. In Nielson's [Niea] use of the public MNIST data sample [LC], he goes to say that using a training batch size of $n = 60,000$, we could choose a mini-batch size of $m = 10$, which will yield a factor of 6,000 speedup in estimating the gradient. Please note that the full sample size will be processed by the neural network during training, but again, it is performed on an optimized stochastic design. The SGD therefore will be an approximation of the set per feed-forward into the network. But because the learning rate follows the same direction as gradient descent, then the system continues to work.

Furthermore, we have not discussed how one might choose their batch size. We will discuss later some techniques for general hyper-parameters selection, but we can still look into the batch size itself specifically.

According to [KMN16], perhaps you could still use a large batch size, but it has been seen in practice that this may limit the classification power of the network; it may lose its ability to generalize against new inputs and thus prove not to be a sufficiently trained network. They have found that a batch size of 32-512 tends to be small enough so as not to be as easy to overfit against the data. Of course, any given project may vary, but the consensus so far is to try to remain in the small batch size.

Back Propagation

So far we have discussed the flow of a feed-forward neural network. A network is given an input into its input layer, and then it feeds the input through each successive layer until it arrives at the output layer.

Certainly we can compute the cost function, but without realizing the changes downstream to the earlier layers, the network will have a very hard time learning. To accomplish this, we apply gradient descent using back propagation. As we compute the direction of the changes on the cost function, we apply these changes against each previous layer in the network.

Back propagation is a common, powerful technique to accomplish this learning [LBO98]. In practice they say it can be seen more of an art than science, but when in general it may levy good results.

[LBO98] goes to say that in order to bring out the benefits of the simple and efficient back propagation technique, the programmer may have to make several adjustments to hyper parameters such as: number and types of nodes, layers, learning rates, and others.

This technique works by recomputing the weights from gradient descent using the given cost function from the output layer all the way back to the input layer. That is, as the network is being trained, the gradient cost results are computed back down the layers, so that the entire network can benefit from the cost function result of that given flow.

Below are the four fundamental equations that define Back Propagation:

$$\delta^L = \nabla_a C \odot \sigma'(z^L). \tag{2.7}$$

In 2.7, $\nabla_a C$ is the partial derivative of the cost for the output activation ($\partial C/\partial a_j^L$); it is the gradient computation for that specific activation. $\odot$ is the elementwise product of this derivative of σ. σ is the activation function applied with the input z, at the given layer, L, and then we take its derivative.

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l), \tag{2.8}$$

In 2.8, we will compute the error in the layer l in terms of the error in $l+1$. The first term, though, a little lengthy, is a computation of the transpose of the weight matrix at layer $l+1$. It is then computed in an elementwise product with the derivative of the activation function at layer l, as in, the current layer. What we are accomplishing here is taking the weights at the layer above us (already computed), and making an estimate of the error at this particular layer (the ancestor layer of layer $l+1$). Using (2.7), we can then use (2.8) to use its output to compute this previous layer's error.

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l. \tag{2.9}$$

In 2.9 we are taking the partial derivative of the cost function of the network output against any particular bias in the network, noted by layer l, and the particular node on that layer is in position j.

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l. \tag{2.10}$$

We then must also compute the rate of change with respect to a particular weight in the network. As such, in 2.10 we compute the partial derivative of the cost function against the partial derivative of the weight at layer l, and that layer's node is at the kth position, against the weight of the jth node in the $l-1$ layer. This equation computes the rate of change for a given weight on a particular node in a particular layer, $l-1$.

Hyper-Parameter Selection

Hyper-parameters are comprised of the following:

- number of hidden nodes
- number of hidden layers
- learning rate

- regularization
- weight initialization
- activation function (e.g. Sigmoid)
- number of epochs
- batch size

In general, the higher the number of hidden nodes in a given hidden layer, the more features the network can compute against. Similarly, the more hidden layers specified, the more a network may be able to learn about the subtleties of what it is being trained against, from simple, to more complex. There is no hard and fast rule regarding the selection of these. Some algorithms for choosing these will be discussed later in this section.

The learning rate for a given neural network has been neatly sidled into gradient descent, but we have not yet discussed it in detail. As gradient descent is performed on the network, we want it to find the global minimum. However, in doing so, it may be possible for the network to "skip" over an area which would lead it toward the optimal solution. We say "skip" because it evokes the notion of speed. As a ball rolls down a hill, it gains speed unless it falls into a minima. It may never reach its destination, or it just may take a very long time. Being able to adjust its rate of descent could help. This is what the learning rate is for. It is denoted by η.

In terms of who cost function, we can see how the learning rate is applied via Equation 2.11.

$$\Delta v = -\eta \nabla C, \tag{2.11}$$

Regularization

Regularization is technique that may be used to reduce overfitting during the training of the network. If generating new samples is challenging, or the structure of the network cannot easily be adjusted, then using regularization may be an appropriate next step. The regularization technique we will focus on is the weight decay or $L2$ regularization.

The way this works is by adding an additional term to the cost function.

Equation 2.12 shows how we can add regularization to the quadratic cost function. It is the second term $\frac{\lambda}{2n}\sum_w w^2$ which implements the regularization.

When λ is 0 (the default), then regularization has no impact on the cost.

$$C = \frac{1}{2n}\sum_x \|y - a^L\|^2 + \frac{\lambda}{2n}\sum_w w^2. \tag{2.12}$$

Equation 2.13 shows how we can use L2 regularization against the cross entropy cost function.

Once again, when 0, regularization is an empty term in the equation.

$$C = -\frac{1}{n}\sum_{xj}\left[y_j \ln a_j^L + (1-y_j)\ln(1-a_j^L)\right] + \frac{\lambda}{2n}\sum_w w^2. \tag{2.13}$$

Weight Initialization

Another aspect in determining the learning process for the network is the initialization of weights for each layer of the network. If the network is supposed to learn, then presumably many values for weights will work. In practice this is true.

As we saw in equation 2.2 these weights will directly affect the output of a given node. To ensure that each node can learn differently from its neighbor, a useful initialization is produced by a random distribution.

There are several initialization techniques. A common one [FH01] is a uniform random initialization. [FH01] uses an interval from [-0.05, 0.05] but the range can be different depending on network implementer. [Nieb] uses a Guassian distribution with a mean of 0 and standard deviation of 1 as a default for their implementation on a general neural network.

[FH01] looked at twelve initialization strategies, using normal random initialization as a base line to compare against.

Given that there are a multitude of ways to provide these high level parameters to the network, one might ask, how can we know which is best? Does the programmer need to train some and check the samples to see how well they were classified? This can be a extremely time-consuming process, depending on the research, dataset, and network implementation. One way to determine which hyper-parameters are suitable is to test the output accuracy against the sample dataset.

A general solution to the choosing of parameters is to implement a grid-search. If you had 5 choices for learning parameters to attempt, and 4 for the number of epochs, and 3 for the number of regularization options, then the programmer could run the total combination of these. Whichever one (or ones) come up with the highest accuracy, are those to look into further after that portion of the training.

As stated, process can take a long time, depending on the type of network, how large the training set is, and whether or not there is a GPU available.

An alternative method to grid-search is random search. According to [BBB11], this form of search can identify the optimal hyper-parameter selection. Unlike grid-search, this algorithm looks through a search space and can be run over several iterations; these results will then be aggregated by the application, and the best of them will then be selected and analyzed further.

While training and selecting the number of epochs, a common problem in machine learning is the notion of overfitting. This may occur when the classification accuracy becomes nearly or is 100 %. This may give the creator of the network a false sense of security in their implementation. After training, they will send an unseen example into the network and be confused as to why the wrong classification for the object came through the network.

This issue is normally seen because the network will start to find peculiarities of the sample data, and not see the patterns of the data it is supposed to be solving for. Some ways around this are: increasing the sample data size, adjusting the hyper-parameters, or even invoking early stopping. Early stopping can allow the network to complete its run early (by some number R), such that when it determines that the cost function and accuracy are no longer moving in the correct direction in a good change set over the epochs (its rate of change slows), then it will cease the descent calculation. This may help with overfitting because as a network begins to learn only oddities from the set, its accuracy and cost should have very small changes. Changing the mini-batch size, too, could help (how many samples in the batch are seen per batch throughout an epoch), but early stopping may be a good solution.

Choosing the mini batch is important because it defines the particular size of the samples that flow through the network at a given time. A batch size of ten would push each example through the network (feed-forward), compute the cost/error, and then back-propagate that run. Then the next in the batch will be run, and so on. Larger batches will simply include more information on this cycle as the epoch runs. Lowering the size will force the network to compute the costs more frequently and with less information to base against per run. That is, the weights and biases may be adjusted at a much higher rate than with a larger size.

Industry Standard Dataset

One common example of a good solution to a problem using a neural network is MNIST handwritten digit classification.[LC]

We won't go into too much detail, but it is worth providing some background to see how flexible networks can be, and how to set one up.

The goal of the MNIST problem is to classify any given handwritten (English) digit. That is, if it passed an input which is supposed to be the digit "8", then the node on the output layer that corresponds to the "8" classifier should be fired high. In this case, there are ten nodes on the output layer, one for each digit classification.

These dataset are 784 pixels, and using one hidden layer, it is possible to achieve nearly 100 % correct classifications on the entire sample data set.

Figure **??** shows an example network configuration used by Nielson [Niea] to solve MNIST.

In his solution, he has formed a network using sigmoid neurons, the cross-entropy cost function, and L2 regularization.

In the next chapter, we will look into our proposed network to solve the evaluation form bubble problem.

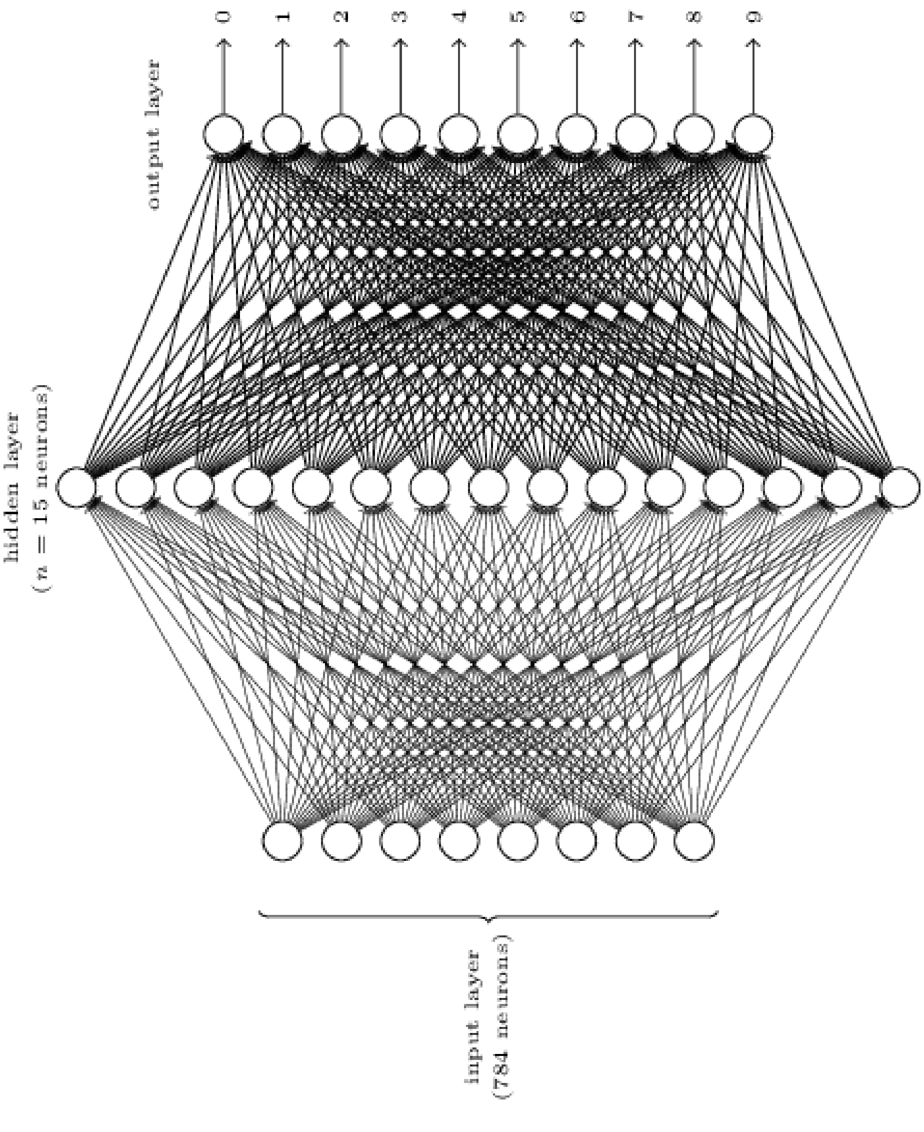


Figure 10: Nielson's Network for MNIST

CHAPTER 3

IMPLEMENTATION

The next question is how should we setup our neural network? What structures in an image (JPEG of scanned form) should it look for?

Object Classification

Initially we considered a network would decide that a positive classification would be if it found a filled in bubble. We would consider a majority dark bubble as filled in. And if the student filled in multiple answers on the same row, we would choose the one with the highest fill density – which would itself be the highest classification by the network.

However, later we wondered if perhaps designing a neural network that knew how to classify bubbles itself, as opposed to filled in ones, may be more useful.

From a data sample generation perspective, we would need to make an arbitrary number of filled in bubbles. Taking one bubble, we could, using the environment's random number generator, construct samples of varying darkness density within a circle's borders. Additionally, we could use a human approach – printing off multiple copies of forms so that we can fill in the bubbles with many fill strategies.

For this document we have tested the trained neural network on fifty hand-written evaluated forms. We will present a histogram showing the results the network and post-processing compiled, as well as a comparison to what the answers were in reality. Note that this dataset was not used to train the proposed network, but instead for showing real-world usage of the network and application logic.

If we could identify a bubble (not filled in), then we could use application code to determine density. That is, we could form a neural network to classify whether a given sample is a bubble. If it is, we could remember the positioning of that section within the general document/form. An additional benefit to looking for unfilled circles is that the department faculty will be able to have a very short feed-back loop to determining how well their given form will work with the neural network. Using our form template, we have a perfect match for each bubble. As such, the proposed solution will offer this template as the default format such that additional departments will have less work in terms of form creation. The network is designed to recognize bubbles, so even with moderate adjustments to the form, it will still find the bubbles.

Network Design

We chose to construct our network such that the output layer had two nodes. This is also known as a binary classifier. In our situation, the requirements are to identify whether a given input is a bubble, or not a bubble. That is, one node will be a bucket for bubbles, and the other, for anything but (whitespace, text, QR-code, etc). When we push an input through the network which is a bubble, we can inspect the node on the output layer which is used for bubble classifications and we should see that it will have at least a value greater than 0.5. Note, we used a sigmoid neuron for this network. As such, we want a bubble to be classified as close to 1.0 as possible. The other node would classify other objects as close to 1.0 respectively. Given that we are solving for a specific solution, we want bubbles to be 1.0, and as such, we will ignore less certain classifications (say, 0.6.).

We thus determined exactly what we wished to classify. In terms of neural network architecture, this levies the following:

1. Input is a 23x23 pixel window
2. output is the classification type, which yields a binary classifier of two nodes.

This sort of structure is common for machine learning. We chose this model to look like other example sets have, for instance with the MNIST dataset [LC]. That dataset has had much success with the input layer of the network being 786 wide, that is 28x28, which is the size of each sample of the dataset. Its output is a series of 10 classifications, one class per digit [0...9].

In our usage, we have a 529 wide input layer (23x23 window), and 2 classifications on the output, one for a bubble, and one for a non-bubble.

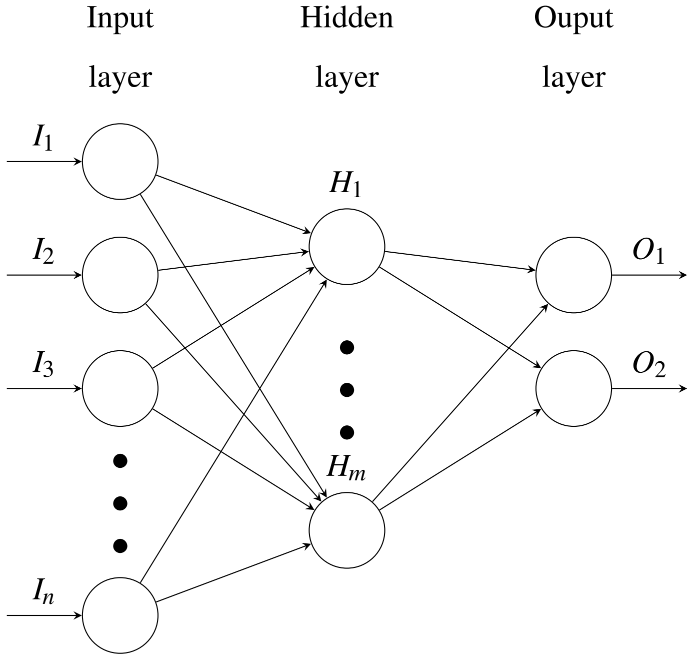
Another common structure for neural networks is to include just 1 hidden layer. This is not a requirement and in many scenarios may not be a good fit for a given project. As discussed previously, multiple hidden layers may help the network to learn more subtle and discrete features about its input. Depending on the network setup, it is no more powerful than a single hidden layer, but it may help simplify some of the hyper-parameters. We will instead stay with the practice of one hidden layer. We chose 30 nodes in our hidden layer. The general neural network's structure is: 529 input, 1 hidden with 30 nodes, and an output layer with 2 nodes.

We have used a modified form of Nielson's code [Nieb]. It is in pure Python code [Helb], and is a generic enough implementation for neural network development. It uses a sigmoid for the classification representation, and stochastic gradient descent to minimize the error function over the sample data.

The error function we use is the cross-entropy function.

Below is our proposed network architecture:

$n = 529$ and $m = 30$



As stated before in Chapter 2, a standard way to determine how well the sample data works with the neural network is to split the data between training, validation, and test. We took a random sample of 70 % of the whole, 15 % for validation, and the remaining 15 % for test.

The Nielson [Niea] code has built-in support to use training and validation datasets. The training is used for back-propagation; the training and validation data set are each run against the cost function so we will be able to see how the cost and accuracy changes over the run of the epochs.

Our script after SGD completion then shows a high level accuracy for the test set. The network would not have yet seen any data from the test, so it is at least a reasonable extra check. However, that does not mean that the network has seen enough data if the test accuracy is high.

We issued the following hyper-parameter selections as the final version of our training:

Learning rate η	0.5
Regularization λ	0.0
epochs	30
hidden layers	1
number of hidden nodes in the layer	30
early stopping	5
number of epochs on completion	9

Table 1: Network Training Hyper-Parameters

Using those, we attained the following results:

Cost on training data	0.00016
Accuracy on training data	70237 / 70237, 100%
Cost on evaluation data	0.00039
Accuracy on evaluation data	15049 / 15050, 99.99%
Test accuracy	15047 / 15051, 99.97%

Table 2: Network Training Statistics

These results show that the network was able to almost perfectly classify all of the examples in the sample data set. Only one example, via the test set, was misclassified.

Data Representation

What does the data look like? The target window size is 23x23. However, this is larger than the circle we're intending to classify, and that is a requirement that we have for a couple of reasons: allow the student to have room in which they can fill a circle such that they might not stay within the lines and not go from the center, but something more like a broad set of strokes (this way we still catch the marking), but also so that when the PDF is created, if the scanner comes in at a slight angle, then we're likely to still capture the relevant area.

Based on the representation and style of the circle (via LaTeX), we created an external document with a circle. It is from this circle that the remaining positive classification labels are derived from.

With this circle, we have saved out a sample which is inside a [invisible] bounding box. The box is 23x23 pixels (529 total), which again, will allow some white space around the circle itself. If we only use one circle's representation for what a bubble is, then the network training won't have nearly enough samples to produce anything meaningful. Knowing that in real world examples the input might have a scanner that doesn't have every pixel at the same RGB or greyscale value (due to image scan conversion), we can perform some adjustments against the source circle.

Sample Generation

In training certain neural networks, we have so far seen that the MNIST dataset is common to verify the learning structure of a network.

There are many other datasets. [DK17] holds 425 datasets for various classifications. They span in popularity in recent years (since 2007), and include new datasets as well, ranging from GPU kernel performance, to Health News in Twitter.

However, we need something specific to text and shape classification. For our purposes, we only need to know what is, and what is not a bubble.

As stated before, we have created a bubble based on the same styling as what will normally be seen in an instructor course evaluation. However, in order to effectively train the network, we must attain a high number of samples.

Because the samples we are generating must fit into a 23x23 pixel window, they must all be scaled down if they were generated with a higher dots per inch (DPI). We have a strict target around 70 DPI. As such, we have taken care to ensure that the default inputs will be at the correct resolution, otherwise we would have inaccurate sample data.

Once we have our data, we want to approximate real world positive and negative (bubble) examples. As such, we must show samples that are not uniform – that is, there will be multiple of a given kind but with slight modifications to them.

We perform a set of mutations on the input samples to create a large breadth of possibilities for attaining the correct classifications on the network.

We have generated approximately 20,000 bubble-based samples via these mutations.

Position Manipulation

The first mutation we used was to shift the circle within the bounding box. Starting from the top left, shifting along the horizontal axis, then to the vertical axis, until we have reached the bottom right. In each of these shifts, the bubble is entirely visible.

Not only does this help us to create additional samples, but it helps to offset for custom form templates and for scanner shifting.

Unsharpen Filter

An additional, orthogonal mutation we used was to perform an Unsharpening filter effect on the input sample. For each position manipulation, we make a new sample with the features unsharpened. This effect will be able to in some cases approximate the effect of a poor PDF scan or poor PDF to JPEG conversion. This is a explicit defensive move to further ensure compatibility with multiple document sources and conversions.

Pixel Density Manipulation

The last required mutation we perform is to turn off (make white) a given dark pixel, but leaving the rest of the source in tact. If we apply this some arbitrary number of times, we end up with a decent amount of sample data, all from a good source of truth.

We were able to generate some distortion (pixel to white) using the environment's random number generator. We applied this for 1% off, through 19 % off, and made 35 samples per each. Some of the samples might overlap, but given the random module's generation, we do end up with upwards of 20,000 samples with a large variation of representation. By turning off only up to 10 percent of the circle, we guarantee that we still hold the general shape.

Sample Rotation

To further combat the noise in a given image source, we have enable optional sample rotation. That is, when in use, we will generate an additional number of samples with given labels. For bubbles, this means a further *n* number of samples, each rotated (presumably 45 degrees from the previous).

This rotation manipulation is in effect for the non-bubble samples without the creator specifying.

Non-Bubble Samples

Approximately 20,000 images for bubbles allows a large enough sample to be able to train on, but we still have no non-bubble data. In order to effectively train the network to classify multiple sets of objects, we need a substantial sample of non-bubbles. Fortunately, this is not overly challenging. Given that we're working with a text document, coming up with text is a well-defined problem. We can use real words, but the lorem ipsum strategy (using fake but real-looking words) provides a lot of opportunity for non-bubble classification.

We also would like to be able to say that white-space is not a bubble. One way to generate this was to take a bubble image, and turn off from 90% to 99% of its pixels, and again, making 1000 samples per. There is a lot of white space in these documents, so we needed to ensure that we considered it for the training. This yields 7,000 samples.

Additionally, each evaluation form will come with a QR-code to identify the course and the semester. As such, we wish to ignore the QR-code whilst feeding into the network. We have a stored image of a QR-code and we run the sliding window along it. This yields 38,220 samples – it is a notable number, but given how large and varied QR-codes are, this was an important and useful dataset to include.

Although this sample data generation has provided us with a lot for the network to work with, it is not indicative of the full types of images it will see. Indeed, as we slide our window over the document, we'll see many partial words and letters. We have generated documents with lorem ipsum, but we don't yet have anything partial. To this end, we alter our generation of samples of non-bubbles (via lorem ipsum) to follow the same sliding window that the network will use when being given sections of the source. This allows us to have a large breath of partial word and letter data and helps to classify general words in a sentence as non-bubbles.

This manipulation does follow the algorithm of the fourth type in which the bubble samples are generated. This yields approximately 80,000 non-bubble samples.

Network Verification

Once the network had been trained, we had to then execute it upon an un-evaluated copy of the evaluation form. The purpose of this is to see what the network determines is a bubble on real-world data. If it was very good at classifying real examples on the type of source image we needed, then we could consider this a proof of concept. The term proof of concept may seem a bit more like a draft than production software. However, given that the evaluation forms are not intended to change their format drastically (and keep circles), the runtime should behave as intended.

We then processed a couple additional example documents to see how well the network would perform. In general, knowing what sort of features the network was trained against, there were no surprises. The bubbles in general are intended to have some space around them, so if a document has circles very close to text, the odds of the system classifying those circles is low.

Data Post-Processing Logic

We have built application logic around the execution of the network while it works on a given 23x23 window of the source document. As we slide a cropped section of the image through the network, it will return the classification accuracy per class-type. If it determines that it believes a given section is a bubble, then we hold onto the information.

When a bubble itself has been consumed by the network, we get the proper classification. This is due to the training of the network. It understands the shape of the circle.

However, due to the level of noise in which text (words, numbers, other shapes) can outstrip the sample data (lorem ipsum), and we can have false-positives; how can we programmatically determine if the network's classification is incorrect?

Certain pieces of text can trip through and exploit the network's weights. Sometimes they can also be more accurate than even a bubble. This is partially due to the training itself, but also how much dynamic input the network sees. That is, if only a circle is a circle, then everything else isn't; but how can you sample everything that can be in a text document?

To accomplish the goal of removing most if not all false-positives, we have implemented stress-logic on the post-process of a given positive classification.

An interesting and useful property of a circle is that as you rotate it, it remains a circle. Due to conversion from PDF to JPEG and anti-aliasing, not all of the pixels will necessarily be uniform throughout the circle's body definition, so it is not an exact copy that you receive from a rotation, but it is a very close approximation to the real source circle. As such, as you rotate the image, if it is a circle, it should remain a circle; what was classified as a bubble will remain classified in such a constant manner.

Using this rotation scheme, we can drop out false-positives from the data output mix. Due to the large number of variations in the input data, we chose to use 7 rotations in addition to the first classification. The first is technically a rotation of 0 (or 360) degrees, and the 7 following it are the next 45 degree rotation (45, 90, etc, up to the rotation before 360). The circle will remain a 90 % classification accuracy while the false-positives drop out at various rotations a few even have 6 positive rotations, but we require 7 to ensure maximum accuracy.

At this point, we attain several high positive classifications per actual bubble sliding window. This is because we have trained against bubbles that are not necessary fully centered in the space. From a conceptual level, this would be enough to stop and we could simply return just the positive crops. However, the application of this research is for allowing the output of this to be measurable and useful for evaluation forms. We must perform a de-duplication step in order not to have several options for a specific answer choice on the form. For instance, if we have 20 positive matches in a given answer choice, how shall we decide which is the one we wish to use for the evaluation process itself? We have chosen to use the output with the highest bubble classification (the one closest to 1.0).

Figure 11 shows a real-world run of the network on a given data set.

We have compiled a set of 50 evaluated forms. We have determined, by human method, the number of responses per answer per question. Figure **??** shows each plot's totaling comparison results. The plots themselves indicate how many responses were per each question's possible answer options. We then show per answer the machine and human totaling.

In order to show how well the network and post-process logic compares to a human, we did the following:

1. Ran the unevaluated form against the neural network
2. Stored the positions of the objects classified as bubbles
3. Iterated over each evaluated form (student markings) to compile the number of responses to each answer for each question, based on the positions we attained in the previous step

Although not required for verifying the network output, for our use case, we must only show unique bubbles. As stated before, we allow the bubble to be in many different positions of its bounding box. As such, in order to accurately note and visualize the output of the network, we employ the following algorithm:

1. First sort the output by y-axis starts
2. Then per y-axis start, sort all of the output by the x-axis starts.

   To ensure that we're looking at one specific row at a time, we limit the upper selection of x-axis to a threshold within the y-start. That is, we begin to row-ify the output.
3. On this particular row we find all of the positives that match the y-axis threshold, and as we find one, we must only grab one slot at a time. The term slot here is referred to as all of the outputs that are within a certain x and y threshold of the given one found (a specific answer choice in the form). So the first output on the y-threshold becomes a slot. We then find the one with the best classification with a threshold of that x and that y.
4. We repeat that step, adding up the "best" per slot, until the row (within that y) has been exhausted

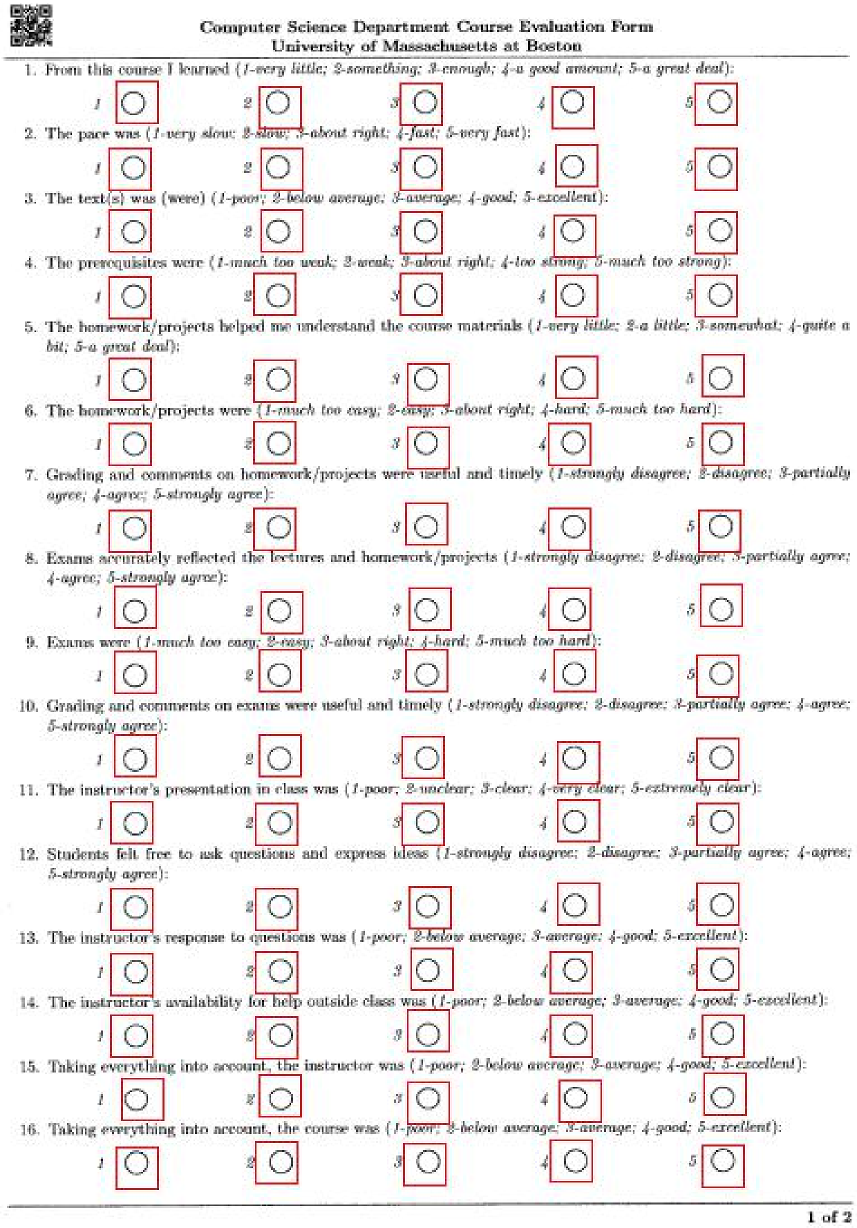
5. We repeat that step until all rows have been processed. This algorithm is required so that when the web system is given a set of evaluations, we can then link a given filled in bubble's position with which row, and thus which question, it relates to. It is also used for general network output processing to show the administrator what the system has recognized, uniquely.

Of note, the row/column structuring is not required for this step. We do this simply to optimize for the general case which is for evaluation forms which must link to a specific question's answer. That is, we must be able to find all of the answer choices for Question 1.

Additionally, we have generated a separate sheet to be run on the network. This result shows that the network is able to generalize beyond the form it was created for – it correctly recognizes bubbles and non-bubbles. This is shown in Figure 13.

The results from Figures 11 and 12 show that the network and post-process algorithms help us attain near or better than human totaling accuracy.

Actual usage of the network for the Department's staff, however, will not be in running the forms on the network directly. The next chapter will discuss a modern interface to this process using a web portal.



Computer Science Department Course Evaluation Form
University of Massachusetts at Boston

1. From this course I learned (*1-very little; 2-something; 3-enough; 4-a good amount; 5-a great deal*):

   1 ○ *2* ○ *3* ○ *4* ○ *5* ○

2. The pace was (*1-very slow; 2-slow; 3-about right; 4-fast; 5-very fast*):

   1 ○ *2* ○ *3* ○ *4* ○ *5* ○

3. The text(s) was (were) (*1-poor; 2-below average; 3-average; 4-good; 5-excellent*):

   1 ○ *2* ○ *3* ○ *4* ○ *5* ○

4. The prerequisites were (*1-much too weak; 2-weak; 3-about right; 4-too strong; 5-much too strong*):

   1 ○ *2* ○ *3* ○ *4* ○ *5* ○

5. The homework/projects helped me understand the course materials (*1-very little; 2-a little; 3-somewhat; 4-quite a bit; 5-a great deal*):

   1 ○ *2* ○ *3* ○ *4* ○ *5* ○

6. The homework/projects were (*1-much too easy; 2-easy; 3-about right; 4-hard; 5-much too hard*):

   1 ○ *2* ○ *3* ○ *4* ○ *5* ○

7. Grading and comments on homework/projects were useful and timely (*1-strongly disagree; 2-disagree; 3-partially agree; 4-agree; 5-strongly agree*):

   1 ○ *2* ○ *3* ○ *4* ○ *5* ○

8. Exams accurately reflected the lectures and homework/projects (*1-strongly disagree; 2-disagree; 3-partially agree; 4-agree; 5-strongly agree*):

   1 ○ *2* ○ *3* ○ *4* ○ *5* ○

9. Exams were (*1-much too easy; 2-easy; 3-about right; 4-hard; 5-much too hard*):

   1 ○ *2* ○ *3* ○ *4* ○ *5* ○

10. Grading and comments on exams were useful and timely (*1-strongly disagree; 2-disagree; 3-partially agree; 4-agree; 5-strongly agree*):

    1 ○ *2* ○ *3* ○ *4* ○ *5* ○

11. The instructor's presentation in class was (*1-poor; 2-unclear; 3-clear; 4-very clear; 5-extremely clear*):

    1 ○ *2* ○ *3* ○ *4* ○ *5* ○

12. Students felt free to ask questions and express ideas (*1-strongly disagree; 2-disagree; 3-partially agree; 4-agree; 5-strongly agree*):

    1 ○ *2* ○ *3* ○ *4* ○ *5* ○

13. The instructor's response to questions was (*1-poor; 2-below average; 3-average; 4-good; 5-excellent*):

    1 ○ *2* ○ *3* ○ *4* ○ *5* ○

14. The instructor's availability for help outside class was (*1-poor; 2-below average; 3-average; 4-good; 5-excellent*):

    1 ○ *2* ○ *3* ○ *4* ○ *5* ○

15. Taking everything into account, the instructor was (*1-poor; 2-below average; 3-average; 4-good; 5-excellent*):

    1 ○ *2* ○ *3* ○ *4* ○ *5* ○

16. Taking everything into account, the course was (*1-poor; 2-below average; 3-average; 4-good; 5-excellent*):

    1 ○ *2* ○ *3* ○ *4* ○ *5* ○

1 of 2

Figure 11: Bounded areas showing bubble recognition

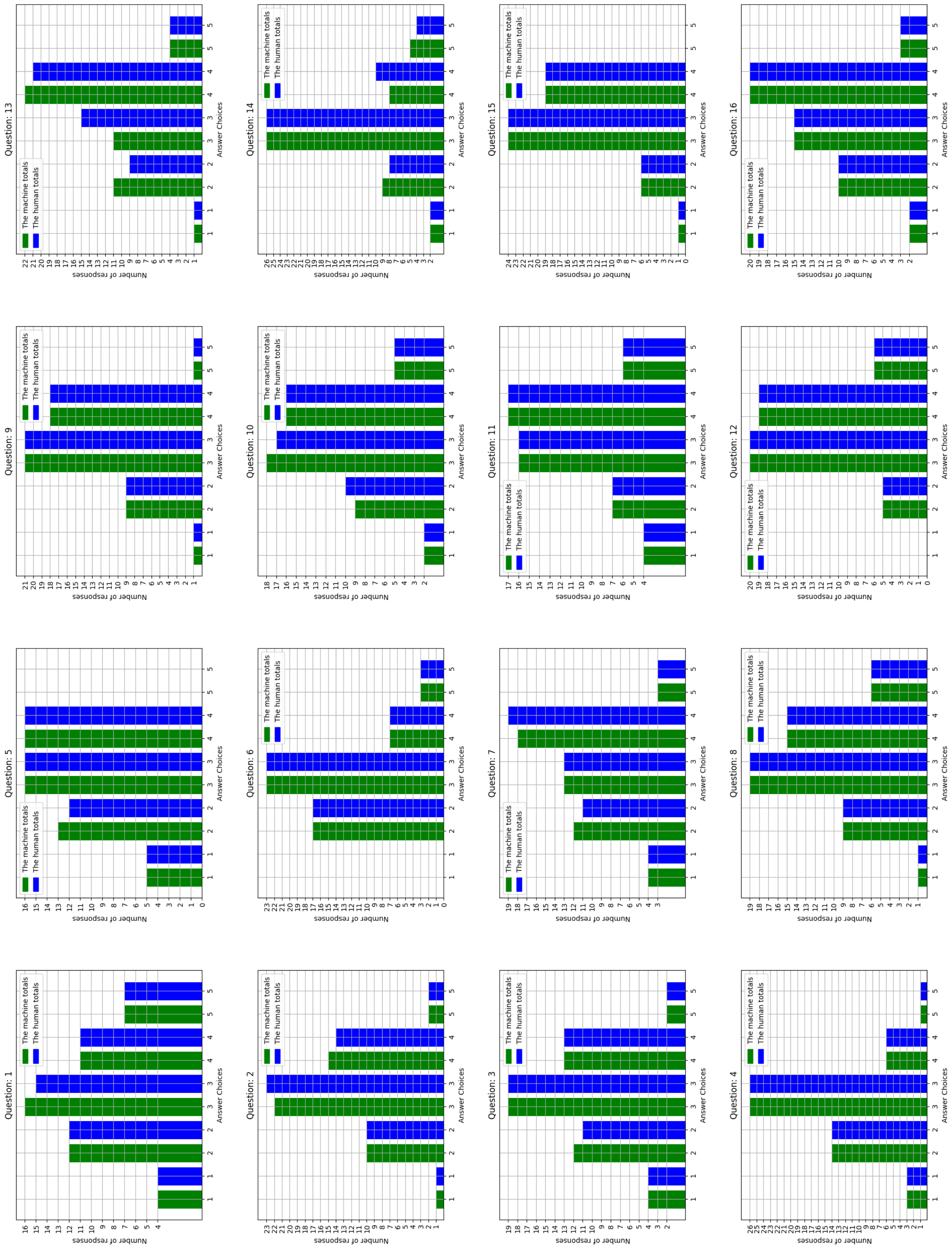
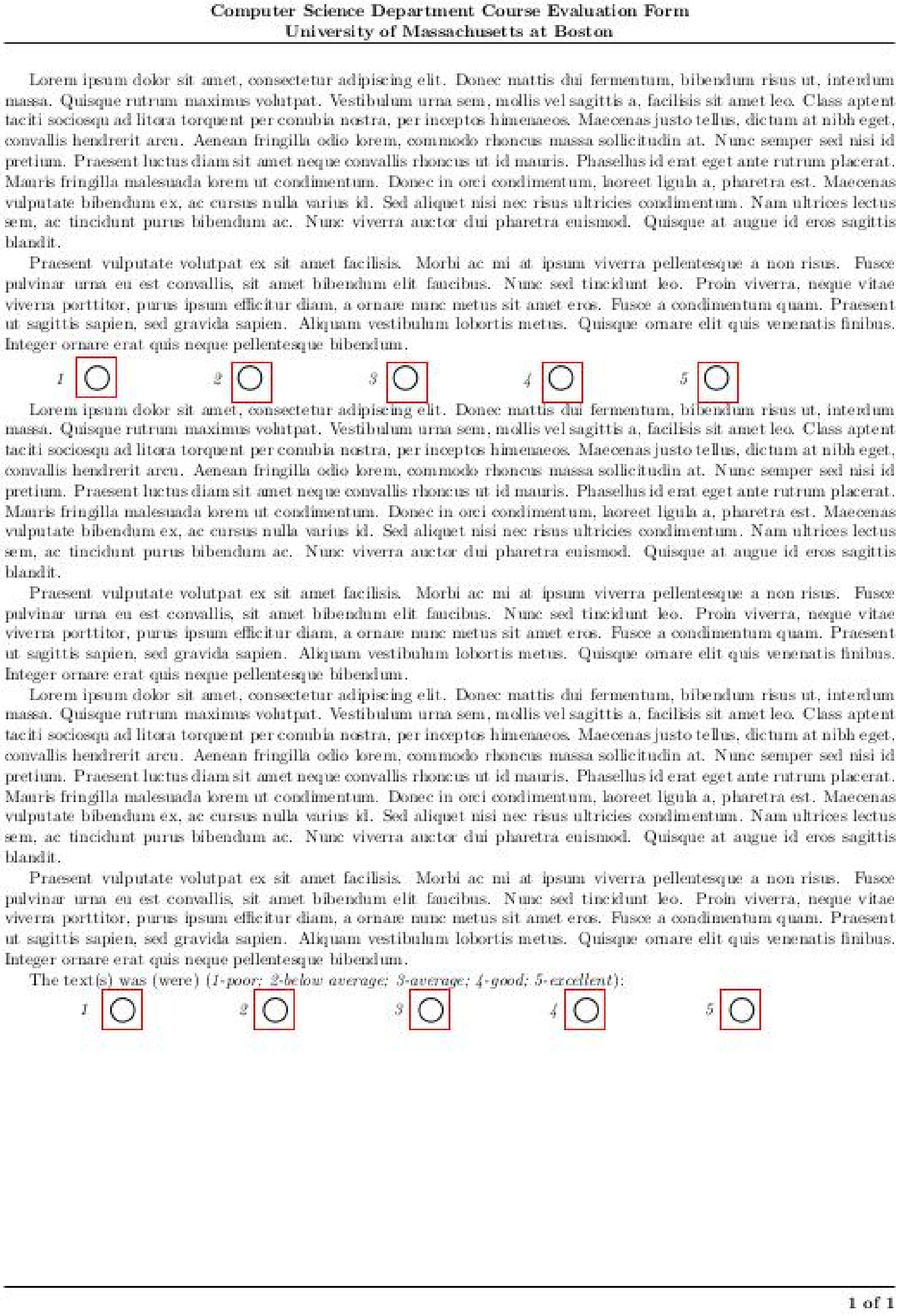


Figure 12: Machine Post Processing (green) Results Compared To Human Totaling (blue)



Computer Science Department Course Evaluation Form
University of Massachusetts at Boston

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec mattis dui fermentum, bibendum risus ut, interdum massa. Quisque rutrum maximus volutpat. Vestibulum urna sem, mollis vel sagittis a, facilisis sit amet leo. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Maecenas justo tellus, dictum at nibh eget, convallis hendrerit arcu. Aenean fringilla odio lorem, commodo rhoncus massa sollicitudin at. Nunc semper sed nisi id pretium. Praesent luctus diam sit amet neque convallis rhoncus ut id mauris. Phasellus id erat eget ante rutrum placerat. Mauris fringilla malesuada lorem ut condimentum. Donec in orci condimentum, laoreet ligula a, pharetra est. Maecenas vulputate bibendum ex, ac cursus nulla varius id. Sed aliquet nisi nec risus ultricies condimentum. Nam ultrices lectus sem, ac tincidunt purus bibendum ac. Nunc viverra auctor dui pharetra euismod. Quisque at augue id eros sagittis blandit.

Praesent vulputate volutpat ex sit amet facilisis. Morbi ac mi at ipsum viverra pellentesque a non risus. Fusce pulvinar urna eu est convallis, sit amet bibendum elit faucibus. Nunc sed tincidunt leo. Proin viverra, neque vitae viverra porttitor, purus ipsum efficitur diam, a ornare nunc metus sit amet eros. Fusce a condimentum quam. Praesent ut sagittis sapien, sed gravida sapien. Aliquam vestibulum lobortis metus. Quisque ornare elit quis venenatis finibus. Integer ornare erat quis neque pellentesque bibendum.

1 ○ *2* ○ *3* ○ *4* ○ *5* ○

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec mattis dui fermentum, bibendum risus ut, interdum massa. Quisque rutrum maximus volutpat. Vestibulum urna sem, mollis vel sagittis a, facilisis sit amet leo. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Maecenas justo tellus, dictum at nibh eget, convallis hendrerit arcu. Aenean fringilla odio lorem, commodo rhoncus massa sollicitudin at. Nunc semper sed nisi id pretium. Praesent luctus diam sit amet neque convallis rhoncus ut id mauris. Phasellus id erat eget ante rutrum placerat. Mauris fringilla malesuada lorem ut condimentum. Donec in orci condimentum, laoreet ligula a, pharetra est. Maecenas vulputate bibendum ex, ac cursus nulla varius id. Sed aliquet nisi nec risus ultricies condimentum. Nam ultrices lectus sem, ac tincidunt purus bibendum ac. Nunc viverra auctor dui pharetra euismod. Quisque at augue id eros sagittis blandit.

Praesent vulputate volutpat ex sit amet facilisis. Morbi ac mi at ipsum viverra pellentesque a non risus. Fusce pulvinar urna eu est convallis, sit amet bibendum elit faucibus. Nunc sed tincidunt leo. Proin viverra, neque vitae viverra porttitor, purus ipsum efficitur diam, a ornare nunc metus sit amet eros. Fusce a condimentum quam. Praesent ut sagittis sapien, sed gravida sapien. Aliquam vestibulum lobortis metus. Quisque ornare elit quis venenatis finibus. Integer ornare erat quis neque pellentesque bibendum.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec mattis dui fermentum, bibendum risus ut, interdum massa. Quisque rutrum maximus volutpat. Vestibulum urna sem, mollis vel sagittis a, facilisis sit amet leo. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Maecenas justo tellus, dictum at nibh eget, convallis hendrerit arcu. Aenean fringilla odio lorem, commodo rhoncus massa sollicitudin at. Nunc semper sed nisi id pretium. Praesent luctus diam sit amet neque convallis rhoncus ut id mauris. Phasellus id erat eget ante rutrum placerat. Mauris fringilla malesuada lorem ut condimentum. Donec in orci condimentum, laoreet ligula a, pharetra est. Maecenas vulputate bibendum ex, ac cursus nulla varius id. Sed aliquet nisi nec risus ultricies condimentum. Nam ultrices lectus sem, ac tincidunt purus bibendum ac. Nunc viverra auctor dui pharetra euismod. Quisque at augue id eros sagittis blandit.

Praesent vulputate volutpat ex sit amet facilisis. Morbi ac mi at ipsum viverra pellentesque a non risus. Fusce pulvinar urna eu est convallis, sit amet bibendum elit faucibus. Nunc sed tincidunt leo. Proin viverra, neque vitae viverra porttitor, purus ipsum efficitur diam, a ornare nunc metus sit amet eros. Fusce a condimentum quam. Praesent ut sagittis sapien, sed gravida sapien. Aliquam vestibulum lobortis metus. Quisque ornare elit quis venenatis finibus. Integer ornare erat quis neque pellentesque bibendum.

The text(s) was (were) (*1-poor; 2-below average; 3-average; 4-good; 5-excellent*):

1 ○ *2* ○ *3* ○ *4* ○ *5* ○

1 of 1

Figure 13: Another Real World Example Output

CHAPTER 4

WEB PORTAL

The second piece of this project is a system to allow the university administrative faculty and instructors a way to manage and view the course evaluations.

Web portals via modern web browsers have become a common platform.

By presenting the interface to upload and analyze instructor evaluations via the web, we have defined a standard flow that the site's users should be familiar with.

We have implemented this via the Django web-framework [Fou]. It is written in Python, which is the same language we use for training and using our neural network.

Neural Network Interface

The user of the site will not work directly with the neural network. Instead, they will allow the portal to consume course evaluations, and behind the scenes, the network and application logic will perform the necessary operations for the staff and faculty.

The database will hold a record of each department per semester, with the associated neural network. A given course itself may override this and provide their own trained network. This allows for maximum flexibility, but the default case will use the department's version.

As such, a developer or system administrator will be involved not only with possibly training the network for a given department, but also in preparing the system to use it, via creating database records for the staff.

Uploading Evaluations

The department staff will upload a given course's unevaluated PDF form. It is this version of the form that is most useful to the network. Our network is trained to classify circles, so we do not allow the staff to upload evaluated copies of this form for this purpose. Those will be handled at a later time.

After this has been done, the system will then allow staff to upload the evaluated copies.

At this point, the staff member will then upload a single PDF file for the aggregate set of evaluations for a particular class. This file will be analyzed by the system when instructed to perform the evaluation statistics generation. This may be an asynchronous step; this will be detailed further later.

Permissions System

So far we have seen that staff will have the ability to run the network as well as generate statistics (via evaluated copies), but the department may wish to have permission controls over viewing and edits.

The form statistics may be run at any point in time once uploaded, however, the administrative staff will not mark them for release (viewable by instructors) until after the semester's grades have been assigned.

In order to implement such an access control layer, we intend to use the database as the source of truth to determine when and who can view a given evaluation result.

As we stated above, the department will mark to be released. This will be accomplished via a change in the *status* field for that evaluation. It will go into the *Active* status, to show that it is ready. If the field has any other value (e.g. , *Inactive*), then the instructor will be unable to view it.

Additionally, instructors should not be able to view any other results than those for the courses they have taught. As such, this will also be implemented as a database level check, verifying that the logged in user (instructor) is the same as the instructor stored on the course they are trying to look at.

Analytics

Given that the results of the evaluations will be stored into a database, this allows prime access for an analytics engine to compose meaningful data.

Visualization

Because we are using a web browser, we can easily show representations of the results as images (e.g. JPEG). The backend server will be responsible for rendering these statistics into that format.

An instructor or staff member can look at a given course for that semester, once the results are computed, and they will be able to access the data in the format. This will show a set of histograms (one per question, with the bar measuring replies per answer).

For the long-form hand-written answers, we will instead show the data as a set of images for the instructor to view.

Time-Series Data

Although allowing easy access to the results of a given course's data for that semester is crucial to the university. However,now that the system holds all of the results for each semester, there are many enhanced data views that we can provide. Time-series data will be moderately easy to show; it also provides an easier method to make decisions regarding the outcomes of staff and course offerings.

For instance, an administrator could select a time-series data analysis of results against a given course over a ten year period. They can see which answers, if any, had changed over time. Additionally, they can correlate this against a change in the instructor. Or, the administration could even track the number of negative feedback response with class size over time; such a query may be nothing more than a correlation, but the ability to present the information in such a way is powerful.

Database Model Schema

We are using a relational database to structure and store the evaluation and user data.

The schema must store Instructors, Courses, Semesters, Evaluation forms, Evaluation Results, and several other entities. The User model itself, though important for the runtime and authentication and permissions system, is closer to a standard User than anything custom for the application.

Entity Relation

An instructor may be part of several courses in a given semester, and more than one instructor may be affiliated with a given course. This relation is important because it allows for teaching and lab assistants to be involved in the process. There is no requirement that a Instructor record be bound to a given User, only that an Instructor is linked to some set of Courses.

We have generated a representation of this database schema in the Universal Markup Language (UML). Given that this is a database schema, it may be used as an Entity Relation Diagram (ER Diagram).

Figure **??** annotates the UML.

*Training the Network Using [Helb], we will perform the neural network training against the sample set for the unevaluated forms. Following the project's documentation (via the README), we will execute these steps:

1.
   ```
   python generate_samples.py --default_good=1
   --default_bad=1 --default_ipsum=1 --default_good_permutations=1
   --shuffle_samples=1 --x_size=23 --y_size=23
   ```

2.
   ```
   python format_samples.py --default_good_permutations=1
   ```

3.
   ```
   python setup_network.py 30 30 --default_input=1 --monitor_training=1
   --eta=0.84 --lmbda=0.0 --binary_classifier=1 --shuffle_input=1
   --default_good_permutations=1 --early_stopping_n=5
   --x_size=23 --y_size=23
   ```

The Python scripts have some parameterization, however, they are intended for usage with certain sample size resolutions. As such, if the developer needs to test a specific custom configuration, they may need to instead generate the samples on their own. There are functions to accompany this workflow, but the Python scripts themselves are not designed to handle high customizations.

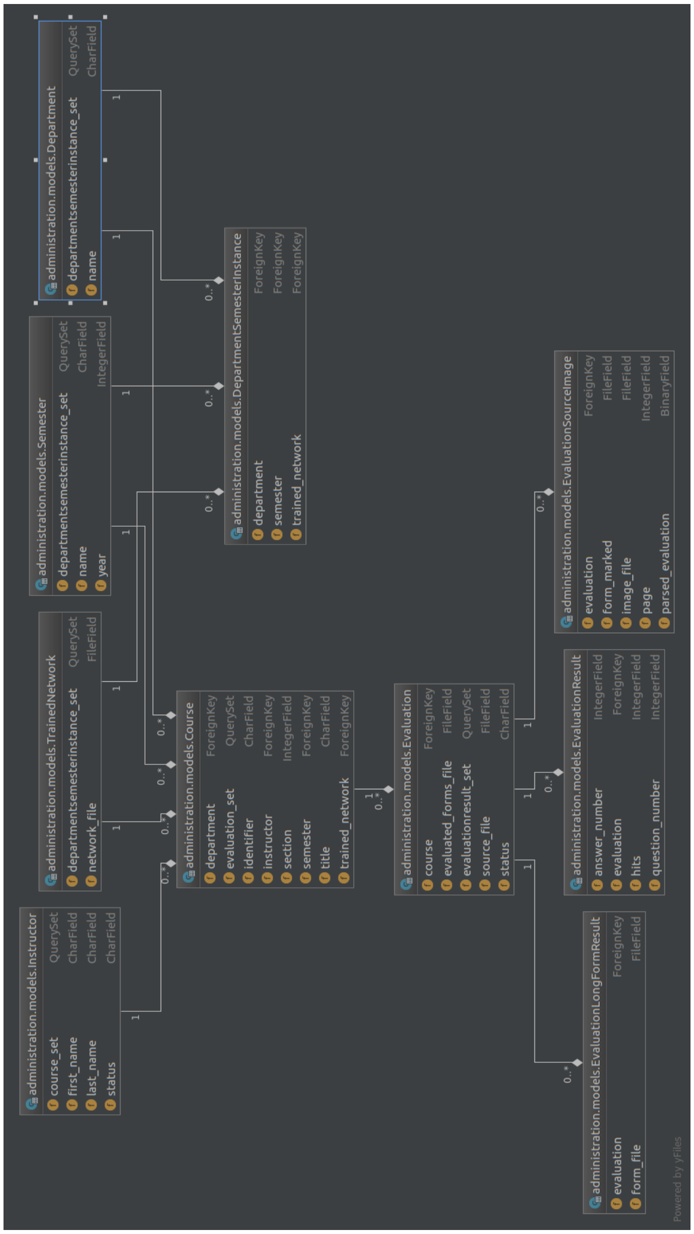

administration.models.Department
departmentsemesterinstance_set QuerySet
name CharField
administration.models.Semester
departmentsemesterinstance_set QuerySet
name CharField
year IntegerField
administration.models.TrainedNetwork
departmentsemesterinstance_set QuerySet
network_file FileField
administration.models.Instructor
course_set QuerySet
first_name CharField
last_name CharField
status CharField
administration.models.DepartmentSemesterInstance
department ForeignKey
semester ForeignKey
trained_network ForeignKey
administration.models.Course
department ForeignKey
evaluation_set QuerySet
identifier CharField
instructor ForeignKey
section IntegerField
semester ForeignKey
title CharField
trained_network ForeignKey
administration.models.Evaluation
course ForeignKey
evaluated_forms_file FileField
evaluationresult_set QuerySet
source_file FileField
status CharField
administration.models.EvaluationResult
answer_number IntegerField
evaluation ForeignKey
hits IntegerField
question_number IntegerField
administration.models.EvaluationSourceImage
evaluation ForeignKey
form_marked FileField
image_file FileField
page IntegerField
parsed_evaluation BinaryField
administration.models.EvaluationLongFormResult
evaluation ForeignKey
form_file FileField
1
0..*
Powered by yFiles

Figure 14: Database Schema as Django UML

Training the network, though, is highly parameterized. The developer can specify the size of the input layer and how many nodes per hidden layer. Other hyper-parameters (η, λ, etc) are also available via this script.

*Executing the Unevaluated Form on the Network Once the network has been trained, we will wish to run an unevaluated form against it. In order to accomplish this, we have designed an additional Python script that is runnable via the terminal, like the invocations in the previous chapter. We may issue a command like the following:

```
python run_evaluation.py network_name image_file_name
```

Intermediate files will be written to disk. These store representations of important steps of the form processing. That is, if the developer must run the form through either the network or the post-process step additional times, they can fine-tune the parameters to the script to use already executed build steps.

At the end of each run, the program will display the results of the form to the user with marked bounding boxes around the recognized bubbles.

One such parameters is "–parse_threshold", which defaults to 0.90 (Sigmoid values range from 0 through 1) for positive bubble classifications. There are certain forms which may have false positive recognitions. As such, it is appropriate to raise the threshold parameter. It is a floating point precision number so we can even use 0.999 if needed.

*Using the Web Portal [Hela] implements the web portal. It knows how to interface with the neural network that we have designed in [Helb]. It then provides a database and web server to run the system for the administration to use.

The system administrator must use Python version 3.5 or higher. A "requirements.txt" file exists for use with the PIP installer to ease installation.

In order to setup the system for use, we must run "python manage.py migrate".

The web portal requires a user account. To create this, we run “python manage.py createsuperuser”.

To run the server, “python manage.py runserver”. It can then be visited at “http://localhost:8000”.

General database management can be issued through “http://localhost:8000/admin”, including uploading forms and creating results.

REFERENCE LIST

[BBB11] James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. "Algorithms for hyper-parameter optimization." In *Advances in neural information processing systems*, pp. 2546–2554, 2011.

[BL17] Nikhil Buduma and Nicholas Locascio. *Fundamentals of deep learning: designing next-generation machine intelligence algorithms.* " O'Reilly Media, Inc.", 2017.

[DK17] Dua Dheeru and Efi Karra Taniskidou. "UCI Machine Learning Repository.", 2017. Available at "http://archive.ics.uci.edu/ml.

[FH01] Mercedes Fernández-Redondo and Carlos Hernandez-Espinosa. "Weight initialization methods for multilayer feedforward." In *ESANN*, pp. 119–124, 2001.

[Fou] Django Software Foundation. "Django Web Framework." Available at `https://www.djangoproject.com/`.

[GB10] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks." In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256, 2010.

[Hela] Jason Held. "Evaluations Web Portal to Interface with Neural Network." Available at `https://gitlab.com/jheld/evaluations`.

[Helb] Jason Held. "neuralnetworksanddeeplearning Python35." Available at `https://github.com/jheld/DeepLearningPython35`.

[Iva] Slav Ivanov. "37 Reasons why your Neural Network is not working." Available at `https://blog.slavv.com/37-reasons-why-your-neural-network-is-not-working-4020854bd607`.

[KMN16] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. "On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima." *CoRR*, **abs/1609.04836**, 2016. Available at `http://arxiv.org/abs/1609.04836`.

[LBB98] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-Based Learning Applied to Document Recognition." *Proceedings of the IEEE*, **86**(11):2278–2324, November 1998.

[LBO98] Yann LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. "Efficient backprop." In *Neural networks: Tricks of the trade*, pp. 9–50. Springer, 1998.

[LC] Yann LeCun and Corinna Cortes. "MNIST handwritten digit database." Available at "`http://yann.lecun.com/exdb/mnist/`".

[Niea] Michael Nielson. "Neural networks and deep learning." Available at `neuralnetworksanddeeplearning.com`.

[Nieb] Michael Nielson. "neuralnetworksanddeeplearning github." Available at `https://github.com/mnielsen/neural-networks-and-deep-learning`.

[RHW86] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors." *nature*, **323**(6088):533, 1986.

[Wik] Standford Wiki. "Neural Networks.".