CSC242 Intro to AI
Project 4: Learning

In this project you will have the opportunity to implement and evaluate several important machine learning methods. This is a huge area of research and development, and is well-covered in upper-level Computer Science courses. It's also nearing the end of the term. So we can really only scratch the surface with this project. But remember, the more that you do **yourself**, the more you will learn.

For this project, you **must** implement linear classifiers as described in AIMA Sec. 19.6 (3rd ed. 18.6) and in class. Linear classifiers are the basis of neural networks (AIMA Chapter 21; 3rd ed. 18.7), so for extra credit (max 20%) you may also implement neural networks. Both aspects of the project are described in detail below.

For this project, you **must** also produce a short report describing what you did and presenting the results of your learning program(s). The requirements for this are also detailed below. It is more than a README.

Supervised Learning

Supervised learning is covered in AIMA Sec. 19.2 (3rd ed. 18.2). You should understand what a **hypothesis**, a **hypothesis space**, a **training set**, and a **testing set** are. You should understand both **accuracy** (on the training set) and **generalization** (on a testing set). You should understand what **overfitting** is, how it arises, and what you can do to avoid it.

If you don't know those things yet: **GO THINK OR READ ABOUT THEM NOW.**

You should also understand the relationship between **error** and **loss**, and how the latter can be defined and computed. You should understand that the goal of machine learning is... **WHAT? THINK ABOUT IT BEFORE CONTINUING.**

The goal of machine learning is, given a hypothesis space, to find **the** hypothesis that **minimizes** the loss over a testing set. Recall that a hypothesis space defines a set of **parameters**. Each choice of values for the parameters defines a hypothesis. So the goal of machine learning is to find **the** set of parameters that yield the hypothesis with the minimum loss over the testing set.

As a Computer Scientist, you should understand that for a hypothesis space with many parameters and/or many possible values for them: "it can be **computationally intractable** to systematically search [for the best hypothesis]."

Gradient Descent

Gradient descent is covered in AIMA Sec. 19.6.2 (18.6.1 3rd ed.). You should understand that it is... **WHAT? THINK ABOUT IT BEFORE CONTINUING.**

Gradient descent is a general, iterative method for finding the parameters of a hypothesis space that minimize the loss of the hypothesis with those parameters over the testing set. It is a form of local search (see Unit 1) that uses the gradient of the loss function to guide the search towards the optimal parameters.

The pseudocode is on p. 678 of AIMA 4th ed. (3rd ed. p. 719). You should know what "convergence" is, and the role of "update rules" in the method. If not: please read the textbook, review the lecture slides, and perhaps also go to study session.

Linear Classifiers (Required)

Linear classifiers are well covered in AIMA Sec. 19.6 (3rd ed. 18.6). You should understand what problem they solve, what hypothesis space they use, and **why** that hypothesis space is reasonable for a classifier. **THINK ABOUT IT BEFORE CONTINUING.**

What problem do linear classifiers solve? A: Classification. What hypothesis space do they use? A: Linear combinations of the features passed through a non-linear threshold. Why is that a reasonable thing to use? A: Discussed in class.

Think about what it takes to represent a linear classifier and the data used to train it in a computer program. **THINK ABOUT IT NOW**. Go read the textbook if necessary.

I hope that you read about and thought about things like input vectors, outputs, weight vectors, and update rules. None of these are hard to implement, but you should think through the design before jumping in with unstructured code.

You will learn the most if you develop your implementation yourself. If you need a little help getting started, look at the documentation for the code I have provided from package "`lc`". It will suggest some classes and give you their APIs. That may be enough for you to write the code. But if you need a bit more help, you may build off the code I have provided. You will still need to implement the crucial classes and/or methods for actually learning the linear classifiers.

Requirements

You **must** implement both a hard-threshold (perceptron) classifier (AIMA 19.6.4; 3rd ed. 18.6.3) **and** a logistic classifier (AIMA 19.6.5; 3rd ed. 18.6.4). Almost all of the code can be shared if you design it right.

You should be able to replicate something like the textbook results for the seismic problem (AIMA Figs. 19.16 and 19.18; 3rd ed. 18.15, 18.16, and 18.18). Note that these report accuracy on the training data, which is not ideal but ok for this project.

Demonstrate your program on the seismic data (both clean and noisy datasets) and the "house votes" dataset ("numerical" version), all of which are provided in our code bundle.

Your program **must** produce the data used for graphs like those seen in the textbook, and the graphs **must** must be included in your report. See below for more details about the report.

Neural Networks (Extra Credit max. 20%)

Neural networks are covered in AIMA Chapter 21 (3rd ed. Section 18.7). Both editions cover all the important definitions for both single-layer and multi-layer feed-forward networks.

The 3rd edition of AIMA also provides the algorithm for backpropagation in multi-layer networks (Fig. 18.24), but in the 4th edition it is just treated as an application of gradient descent (p. 755 and 765). To help you implement it, I have included the pseudocode from the 3rd edition in an appendix to this document.

The presentation is very concise in both editions. So if you choose to implement this type of learner, be prepared to do some thinking and/or additional research as you develop and evaluate your system.

It isn't hard to think about what you need to represent a neural network in a computer program. **THINK ABOUT IT NOW**.

I hope you thought about "units," layers, connections, weights, activation functions, inputs, and outputs. Remember that a basic feed-forward neural network is simply a graph of linear classifiers (typically using a logistic threshold).

It is not hard to design classes incorporating these elements. However I suggest that you understand how the backpropagation algorithm works before you lock in your design. In particular, note that it requires that you be able to go both forward and backward through the layers of your networks, even though the network is "feed-forward."

As always, you will learn the most if you develop your implementation yourself. If you need a little help, look at the documentation for the code I have provided from the package "`nn`". That will give you some suggestions for classes and APIs. And if you need more help, you may build off the code I have provided. You will still need to implement the crucial classes and/or methods for the computational units in `nn.core`.

If you choose to do this (for extra credit), you **must** implement a multi-layer network with hidden units. A single-layer network is essentially a set of one or more linear classifiers. A multi-layer network is a "true" neural network that must be trained using backpropagation.

You **must** demonstrate your program on **both** the Iris dataset and the MNIST dataset (handwritten digit recognition). Files, or pointers to data files in the case of MNIST, are provided in our code bundle.

For the Iris dataset, try a two-layer network with four inputs, seven hidden units, and three output units (one for each species of Iris). Of course you can try other network topologies also (more or less hidden units, more layers).

For MNIST, visit `http://yann.lecun.com/exdb/mnist/` and try some of the networks described there. For example, a two-layer network with 300 or 1000 hidden units, or a three-layer network with 500 and 150 units (first and second layer, respectively).

The 3rd edition of the textbook says: "Unfortunately, for any *particular* network structure, it is harder to characterize exactly which functions can be represented and which ones cannot." The 4th edition says (p. 759): "At the time of writing [2021], there is little understanding as to why some structures seem to work better than others for some particular problem."

For both datasets: Your program **must** produce the data for learning curves like Fig. 19.7 (3rd ed. Fig. 18.25), and your report must include graphs made from the output of your program. See below for details regarding your report.

Datasets for Machine Learning

There are many, many datasets available online for training different types of machine learning systems. One of the best sources is the UCI Machine Learning Archive: `http://archive.ics.uci.edu/ml`. Some of the datasets from this archive are downloadable with the project description.

There is always some work reading these files and getting them into the proper form for your learning system. It's easy for simple datasets, like the "Iris" dataset, and harder for more complicated datasets. For problems based on images (or other media), there is often a dataset with a set of attribute values extracted from the data already available, so that you can focus on machine learning rather than image processing. Or you may be interested in extracting the attributes from the media yourself. Note that continuous-valued datasets may need to be discretized for some types of learning (unless you want to implement more sophisticated algorithms).

Start simple. Use the seismic dataset for linear classifiers, or the Iris dataset for neural networks, to get started. They are manageable. Then try something bigger and more interesting.

Report Requirements

For this project, your report is not simply a README. Your report **must** be a PDF prepared using LaTeX, Word, Google Docs, or a similar document preparation application.

The first section of your report **must** include the exact commands needed to build and run your programs.

Linear Classifiers

For the required linear classifiers, your programs' output **must** include data sufficient to produce *training curves* as seen in AIMA Figs. 19.16 and 19.18 (3rd ed. 18.16, and 18.18).

And then you **must** use that output to produce the graphs and include them in your report (clearly labelled and explained). You may use Excel or Google Docs or `gnuplot` or some other graphing package or application. Your graphs **must** have labels on the axes. **Screenshots of unlabelled graphical visualizations (like mine seen in class) are NOT ACCEPTABLE.**

Your report **must** include the commands necessary to produce the data for all six graphs from AIMA for the seismic data, and include the graphs themselves. For the "house votes" dataset, do something similar and include the commands and the graphs in your report.

Include a short paragraph (not bullet points) describing the graphs and discussing the results. It doesn't have to be much, but it has to be readable. Bullet points or a handful of disconnected sentences or no text at all will get a lower grade.

Note: The graphs for the perceptron classifier, shown in AIMA Fig. 19.16 (3rd ed. 18.16), are labelled "Proportion Correct," meaning accuracy, which should go towards 100% as the classifier is trained. This is accuracy on the training set, which is not ideal, but ok for this project.

In the 3rd edition, the graphs for the logistic classifier, shown in Fig. 18.18, are labelled "squared error per sample." But the error won't (or shouldn't) go up as the classifier is trained, so I assume that the label is incorrect. In the 4th edition Fig. 19.18 uses "Proportion correct."

Neural Networks

If you choose to implement neural networks, your program's output **must** include the accuracy (on the training data) after each epoch.

For each required problem, you **must** use your program to produce data sufficient to produce curves similar to those shown in AIMA Fig. 19.7 (3rd ed. 18.25). For the Iris dataset, measure accuracy on the training data (you should really do k-fold cross-validation, but it is not required) and for MNIST report error on the testing set. The former should go up, the latter should go down.

Again: You may use Excel or Google Docs or `gnuplot` or some other graphing package or application to make your graphs. Your graphs **must** have labels on the axes. **Screenshots of unlabelled graphical visualizations (like mine seen in class) are NOT ACCEPTABLE.**

Include a short paragraph (not bullet points) discussing these results along with the graphs. Again, bullet points or disconnected sentences or no text at all will get a lower grade.

Additional Requirements and Policies

The short version:

- You may **only** use Java or Python. I **STRONGLY** recommend Java.
- You **must** use good object-oriented design (yes, even in Python).
- There are other language-specific requirements detailed below.
- You must submit a ZIP including your source code, a README, and a completed submission form by the deadline.
- You **must** tell us how to build your project in your README.
- You **must** tell us how to run your project in your README.
- Projects that do not compile will receive a grade of **0**.
- Projects that do not run or that crash will receive a grade of **0** for whatever parts did not work.
- Late projects will receive a grade of **0** (see below regarding extenuating circumstances).
- **You will learn the most if you do the project yourself**, but collaboration is permitted in groups of up to 3 students.

Detailed information follows...

Programming Requirements

- You may use Java or Python for this project.
  - I **STRONGLY** recommend that you use Java.
  - Any sample code we distribute will be in Java.
- You **must** use good object-oriented design.
  - You **must** have well-designed classes (and perhaps interfaces).
  - Yes, even in Python.
- No giant `main` methods or other unstructured chunks of code.
  - Yes, even in Python.
- Your code should use meaningful variable and function/method names and have plenty of meaningful comments.
  - I can't believe that I even have to say that.
  - And yes, even in Python.

Submission Requirements

You **must** submit your project as a ZIP archive containing the following items:

1. The source code for your project.
2. A file named `README.txt` or `README.pdf` (see below).
3. A completed copy of the submission form posted with the project description (details below).

Your README **must** include the following information:

1. The course: "CSC242"
2. The assignment or project (*e.g.*, "Project 1")
3. Your name and email address

4. The names and email addresses of any collaborators (per the course policy on collaboration)
5. Instructions for building and running your project (see below).

The purpose of the submission form is so that we know which parts of the project you attempted and where we can find the code for some of the key required features.

- **Projects without a submission form or whose submission form does not accurately describe the project will receive a grade of 0**.
- If you cannot complete and save a PDF form, submit a text file containing the questions and your (brief) answers.

Project Evaluation

You **must** tell us in your README file how to build your project (if necessary) and how to run it.

Note that we will **not** load projects into Eclipse or any other IDE. We **must** be able to build and run your programs from the command-line. If you have questions about that, go to a study session.

We **must** be able to cut-and-paste from your documentation in order to build and run your code. **The easier you make this for us, the better your grade will be.** It is **your** job to make the building of your project easy and the running of its program(s) easy and informative.

For **Java** projects:

- The current version of Java as of this writing is: 20.0.2 (OpenJDK)
- If you provide a `Makefile`, just tell us in your README which target to make to build your project and which target to make to run it.
- Otherwise, a typical instruction for building a project might be:

  ```
  javac *.java
  ```

  Or for an Eclipse project with packages in a `src` folder and classes in a `bin` folder, the following command can be used from the `src` folder:

```
javac -d ../bin `find . -name '*.java'`
```

- And for running, where `MainClass` is the name of the main class for your program:

  ```
  java MainClass [arguments if needed per README]
  ```

  or

  ```
  java -d ../bin pkg.subpkg.MainClass [arguments if needed per README]
  ```

- You **must** provide these instructions in your README.

For **Python** projects:

I strongly recommend that you **not** use Python for projects in CSC242. All CSC242 students have at least two terms of programming in Java. The projects in CSC242 involve the representations of many different types of objects with complicated relationships between them and algorithms for computing with them. That's what Java was designed for.

But if you insist...

- The latest version of Python as of this writing is: 3.11.4 (python.org).
- You must use Python 3 and we will use a recent version of Python to run your project.
- You may **NOT** use any non-standard libraries. This includes things like NumPy or pandas. Write your own code—you'll learn more that way.
- We will follow the instructions in your README to run your program(s).
- You **must** provide these instructions in your README.

For **ALL** projects:

We will **NOT** under any circumstances edit your source files. That is your job.

Projects that do not compile will receive a grade of 0. There is no way to know if your program is correct solely by looking at its source code (although we can sometimes tell that is incorrect).

Projects that do not run or that crash will receive a grade of 0 for whatever parts did not work. You earn credit for your project by meeting the project requirements. Projects that don't run don't meet the requirements.

Any questions about these requirements: go to study session **BEFORE** the project is due.

Late Policy

Late projects will receive a grade of 0. You **must** submit what you have by the deadline. If there are extenuating circumstances, submit what you have before the deadline and then explain yourself via email.

If you have a medical excuse (see the course syllabus), submit what you have and explain yourself as soon as you are able.

Collaboration Policy

I assume that you are in this course to learn. You will learn the most if you do the projects **yourself**.

That said, collaboration on projects is permitted, subject to the following requirements:

- Teams of no more than 3 students, all currently taking CSC242.
- Any team member **must** be able to explain anything they or their team submits, IN PERSON AT ANY TIME, at the instructor's or TA's discretion.
- One member of the team should submit code on the team's behalf in addition to their writeup. Other team members **must** submit a README (only) indicating who their collaborators are.
- All members of a collaborative team will get **the same grade** on the project.

Academic Honesty

I assume that you are in this course to learn. You will learn nothing if you don't do the projects **yourself**.

Do not copy code from other students or from the Internet.

Avoid Github and StackOverflow completely for the duration of this course.

There is code out there for all these projects. You know it. We know it.

Posting homework and project solutions to public repositories on sites like GitHub is a violation of the University's Academic Honesty Policy, Section V.B.2 "Giving Unauthorized Aid." Honestly, no prospective employer wants to see your coursework. Make a great project outside of class and share that instead to show off your chops.

Appendix A

```
function BACK-PROP-LEARNING(examples, network) returns a neural network
    inputs: examples, a set of examples, each with input vector x and output vector y
            network, a multilayer network with $L$ layers, weights $w_{i,j}$, activation function $g$
    local variables: $\Delta$, a vector of errors, indexed by network node

    repeat
        for each weight $w_{i,j}$ in network do
            $w_{i,j} \leftarrow$ a small random number
        for each example $(\mathbf{x}, \mathbf{y})$ in examples do
            /* Propagate the inputs forward to compute the outputs */
            for each node $i$ in the input layer do
                $a_i \leftarrow x_i$
            for $l = 2$ to $L$ do
                for each node $j$ in layer $l$ do
                    $in_j \leftarrow \sum_i w_{i,j}\, a_i$
                    $a_j \leftarrow g(in_j)$
            /* Propagate deltas backward from output layer to input layer */
            for each node $j$ in the output layer do
                $\Delta[j] \leftarrow g'(in_j) \times (y_j - a_j)$
            for $l$ from $L - 1$ to $1$ do
                for each node $i$ in layer $l$ do
                    $\Delta[i] \leftarrow g'(in_i)\ \sum_i w_{i,j}\, \Delta[j]$
            /* Update every weight in network using deltas */
            for each weight $w_{i,j}$ in the output layer do
                $w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta[j]$
    until some stopping criterion is satisfied
    return network
```

AIMA 3rd ed. Figure 18.24 (p. 734): The back-propagation algorithm for learning in multilayer networks