COGS 125 / CSE 175
Introduction to Artificial Intelligence
Specification for Programming Assignment #4

David C. Noelle

Due: 11:59 P.M. on Friday, December 15, 2023

Overview

This programming assignment has three main learning goals. First, the assignment will provide you with an opportunity to practice developing programs in Python using the PyCharm integrated development environment (IDE). Second, this assignment will give you some experience building artificial neural networks using the *PyTorch* system. Third, by encouraging you to examine the learning performance of such networks, this assignment will help you develop insights concerning the strengths and weaknesses of various network architectures.

You will be provided with Python source code that (incompletely) implements three artificial neural network models and trains them to solve a classic classification task. You are charged with completing this implementation by writing Python code that describes three network architectures: a linear network that directly connects inputs to outputs, a network with a single hidden layer, and a network with two hidden layers.

Submission for Evaluation

To complete this assignment, you must generate one Python script file, called "`arch.py`". That file is to contain three Python classes, each with two methods. Each class embodies a particular artificial neural network architecture, with the three classes varying in their use of hidden units. The methods include a constructor for each class and a corresponding implementation of the forward activation propagation process, using *PyTorch* tools. This is the only file that you should submit for evaluation.

To submit your completed assignment for evaluation, log onto the class site on CatCourses and navigate to the "Assignments" section. Then, locate "Programming Assignment #4" and select the option to submit your solution for this assignment. Provide your single program file as an attachment. Do not upload any other files as part of this submission. Comments to the teaching

team should appear as header comments in your Python source code file. Header comments are also expected to provide written answers to two analysis questions, described later in this document.

Submissions must arrive by 11:59 P.M. on Friday, December 15th. You should plan to allow time for potential system slowness immediately prior to this deadline. You are allowed to submit assignment solutions multiple times, and only the most recently submitted file will be evaluated. As discussed in the course syllabus, late assignments will not be evaluated and will receive *no credit*.

If your last submission for this assignment arrives by 11:59 P.M. on Tuesday, December 12th, you will receive a 10% bonus to the score that you receive for this assignment. This bonus is intended to encourage you to try to complete this assignment early.

Activities

Overview

The *PyTorch* system is currently one of the leading software tools for making artificial intelligence systems using deep artificial neural networks. It provides fairly abstract methods for building such machine learning models, training them on data, and applying the trained models to novel inputs. This assignment provides you with some introductory experience using *PyTorch* software.

When complete, this assignment results in a Python program that constructs three different network architectures:

- `AnnLinear` — A network with no hidden layer, connecting a 4-element input vector to a 3-element linear output layer.
- `AnnOneHid` — A network with a single hidden layer of 20 processing units, mediating the production of a 3-element output vector from a 4-element input vector.
- `AnnTwoHid` — A network with two successive hidden layers between a 4-element input and a 3-element output, with the first hidden layer having 16 processing units and the second having 12 processing units.

The program trains each constructed network on a simple classification task, reporting progress during training and calculating the accuracy of each learned model on a set of testing examples.

The classification task is one that has been studied for many years. The goal is to learn to identify the species of an iris plant based on the size of various parts of the plant. Each plant is a member of one of three possible species: *setosa*, *versicolor*, or *virginica*. Four numerical features of the plant are input to allow the system to categorize it by species. The four input measurements are sepal length and width and petal length and width. A total of 150 examples are provided for use as training data and testing data. Details concerning the data set may be found at the *UC Irvine Machine Learning Repository*:

`https://archive.ics.uci.edu/ml/datasets/iris`

This is a fairly simple classification task to learn, making it easy to produce models with high categorization accuracy, even on previously unseen inputs.

Python Packages

This program makes use of three publicly available Python packages, all of which need to be installed in the PyCharm IDE before the program can be successfully run.

- *PyTorch* (`torch`) — tools for building artificial neural network models
- *Pandas* (`pandas`) — tools used for data loading and manipulation
- *Scikit-learn* (`scikit-learn`) — tools used to randomly sample training and testing sets from a collection of examples

These packages can be easily installed in PyCharm. On the `Preferences` window (or the `Settings` window), there is an option on the project pane which is labeled "Python Interpreter". Selecting this option shows a table of installed packages. The plus-sign button may be selected to install a new package. The installation window contains a search bar, through which the three needed packages may be found. Once found, a package may be installed from this window by selecting the "Install Package" button.

Program Structure

The program is implemented in three Python script files. One of these files is incomplete.

- "`main.py`" — This file provides code to train and test all three network architectures, printing performance results.
- "`ann.py`" — This file provides three utility functions.
  - `load_iris_data` — Load the iris classification data set from the *UC Irvine Machine Learning Repository*.
  - `train_model` — Train a given network model using a specified set of examples.
  - `test_model` — Calculate the accuracy of a given network model on a specified set of testing examples.
- "`arch.py`" — This file defines three Python classes describing the three artificial neural network architectures of interest.

Of these three files, only the third, "`arch.py`", is incomplete. A template is provided, but a solution to this assignment must provide the code necessary to implement the three architectures.

Architecture Specification

In *PyTorch*, an artificial neural network architecture is specified by defining a new Python class that inherits from `nn.Module`. The "`arch.py`" file defines three such classes: `AnnLinear`, `AnnOneHid`, and `AnnTwoHid`. Each class has an initialization method, called when constructing an instance, and a method that performs forward pass computations to produce network outputs.

In the initialization method, network layers are defined. Each layer should be stored in a class variable. For this assignment, all layers calculate their "net input" values as a linear weighted sum of inputs, with the weights being learned parameters. In *PyTorch*, such a layer may be declared using `nn.Linear` in the following way:

```
self.my_layer = nn.Linear(in_features=6, out_features=9)
```

In this example line of Python code, a layer with 9 processing units is defined, with the layer receiving 6-dimensional inputs (e.g., the number of processing units in the preceding layer). The layer is stored in the "`my_layer`" variable.

In the method that implements the forward pass, aptly named "`forward`", processing element activation levels are successively calculated across layers, from inputs to outputs. The method is given network inputs as an argument, conventionally labeled "`x`". Weighted sums, calculating "net input" values, can be computed by treating layer variables as functions. For example, using "`my_layer`" as previously defined, the "net input" of the layer can be computed using this line of Python code:

```
my_layer_net = self.my_layer(x)
```

An activation function can be applied to the "net input" of a layer. For example, the *rectified linear* (ReLU) activation function may be applied as follows:

```
my_layer_act = F.relu(self.my_layer(x))
```

In this way, the activation of each layer can be calculated, from inputs to outputs.

Note that a few variable naming conventions are repeatedly used. Typically, "`x`" references an input to the network. The target outputs of the network are referenced as "`y`". The actual outputs of the network, to be compared with the targets, are referenced as "`y_hat`" (spelling out $\hat{y}$). Thus, the `forward` method takes `x` as an argument and should return the corresponding value of `y_hat`.

In this assignment, all three network architectures must take a 4-dimensional input, corresponding to the four measurements of the iris plants. Each architecture must provide a 3-dimensional output, with one element for each of the three species categories. The output element with the highest activation is taken as indicating the predicted species corresponding to the current network input. The network architecture with a single hidden layer should use 20 hidden units. The network architecture with two hidden layers should have 16 units in the first hidden layer and 12 in the second. All hidden layers should use the ReLU activation function, but network output layers should not. Output layers should compute the linear weighted sum of their inputs, without any nonlinear activation function.

Analysis

The "`arch.py`" file is to be completed, filling in code for the two methods in each of the three architecture classes. The header comment of this file should also contain answers to two analysis questions.

Once the artificial neural network architectures are implemented, running the program will display measures of training progress and testing set accuracy for each of the three architectures. The program should be run a dozen times or more, recording the displayed results. Because testing examples are sampled at random on each run, and each network begins with random initial weights, each run can produce different results. Still, patterns in performance should be evident. Based on your observations of many executions of the program, you should answer the following two questions:

1. Which network architecture achieves the lowest training set error?

2. Which network architecture tends to exhibit the best testing set accuracy?

With a little thought, you should be able to understand why the observed patterns of performance arose.

Evaluation

To receive full credit for this assignment, working network architecture implementations must be provided, and both questions must be answered, all within the "`arch.py`" script file. The three Python classes must work with the other provided Python script files *without modifying the other files in any way*. All of these files, including a template for "`arch.py`", are available in a ZIP archive file called "`PA4.zip`" in the "Assignments" section of the class CatCourses site, under "Programming Assignment #4". The contents of these Python code files will be briefly discussed during a laboratory session, and comments in these files should assist in your understanding of the provided code. Questions are welcome, and they should be directed to the teaching team.

Your submission will be evaluated primarily for accuracy, with efficiency being a secondary consideration. Your source code *will* be examined, however, and the readability and style of your implementation will have a substantial influence on how your assignment is evaluated. As a rough rule of thumb, consider the use of good software writing practices as accounting for approximately 10% to 20% of the value of this exercise. Please use the coding practices exemplified in the provided utility files as a guide to appropriate readability and style. Note also that, as discussed in the course syllabus, submissions that fail to run without crashing on the laboratory PyCharm IDE will not be evaluated and will receive *no credit*.

The code that you provide should write *no output*, as this will clutter the output produced by the "`main.py`" script. If you include any statements that write output in your code (perhaps as tools for debugging) these should be removed prior to submitting your code files for evaluation. You may receive *no credit* for your submitted solution if it produces extraneous output.

As for all assignments in this class, submitted solutions should reflect the understanding and effort of the individual student making the submission. **Not a single line of computer code should be shared between course participants. Not a single line of computer code should be received from any other person, such as students not enrolled in the class and people providing tutorial assistance.** If there is ever any doubt concerning the propriety of a given interaction, it is the student's responsibility to approach the instructor and clarify the situation *prior* to the submission of work results. Also, helpful conversations with fellow students, or any other person (including

members of the teaching team), should be explicitly mentioned in submitted assignments (e.g., in comments in the submitted source code files). These comments should also explicitly mention any written resources, including online resources, that were used to complete the exercise. Citations should clearly identify the source of any help received (e.g., "Dr. David Noelle" instead of "the professor", "The Python Tutorial at `docs.python.org/3/tutorial/`" instead of "Python documentation"). Failure to appropriately cite sources is called *plagiarism*, and it will not be tolerated! Policy specifies that detected acts of academic dishonesty must result minimally with a zero score on the assignment, and it may result in a failing grade in the class. Please see the course syllabus for details.

To be clear, please note that all of the following conditions must hold for a submitted solution to receive *any* credit:

- solution submitted by the due date
- solution runs without crashing on the laboratory PyCharm IDE
- solution produces no extraneous output
- solution works with unmodified utility code, as provided
- solution does *not* include code written by another person (with or without modifications)
- solution explicitly cites all help received, whether from a person or some other source

While partial credit will be given for submissions that meet these conditions, failure to meet one or more of these conditions will result in *no credit* and, in the case of academic dishonesty, may result in a failing grade in the course. Once again, please see the course syllabus for details.

The members of the teaching team stand ready to help you with the learning process embodied by this assignment. Please do not hesitate to request their assistance.