Object-Oriented Software Development Using C++ (OOP345SAB.05929.2184): 345 project - milestone 1

John Blair - john.blair@senecacollege.ca <do-not-reply@blackboard.com>

Thu 2018-07-12 8:26 AM

OOP346 NEURAL NETS PROJECT MILESTONE ONE

Milestone one is to translate a simple neural network written in only 9 lines of Python to C++.

Search for "How to build a simple neural network in 9 lines of Python code"

You will find it.

(https://medium.com/technology-invention-and-more/how-to-build-a-simple-neural-network-in-9-lines-of-python-code-cc8f23647ca1)

Here are the nine lines of Python code:

```
++++++++++++++++++++++++++++++++++++++++++ nine.py
++++++++++++++++++++++++++++++++++++++++++
from numpy import exp, array, random, dot
training_set_inputs = array([[0, 0, 1], [1, 1, 1], [1, 0, 1], [0, 1, 1]])
training_set_outputs = array([[0, 1, 1, 0]]).T
random.seed(1)
synaptic_weights = 2 * random.random((3, 1)) - 1
for iteration in xrange(10000):
  output = 1 / (1 + exp(-(dot(training_set_inputs, synaptic_weights))))
  synaptic_weights += dot(training_set_inputs.T, (training_set_outputs - output) * output * (1 - output))
print 1 / (1 + exp(-(dot(array([1, 0, 0]), synaptic_weights))))
++++++++++++++++++++++++++++++++++++++++++ nine.py
++++++++++++++++++++++++++++++++++++++++++
```

OOP345 is a C++ course, not a Python course.
If you can read C++, you can read simple Python

Many times in a job, you need to learn something to complete the assignemt. OOP345 is no different. In this milestone you will need to figure out how to express the Python code in C++.

So here is some survival Python to C++:

In C/C++, we sprinkle ';' after each statement. Python doesn't bother.

In C/C++, we surround a block of code by a matching '{' and '}'. Python uses indents, spaces, not tabs. A Python block has the same level of indentation.

In C/C++, we write a if statement as

```
  if (arithmetic-expression)
    statement;
  else
    statement;
```

In Python,
```
  if arithmetic-statement:
    statement
  else:
    statement
```

In C/C++, we write a switch statement as
```
  switch(arithmetic-expression) {
  case const-value-1:
    statement;
    break;
  case const-value-2:
    statement;
    break;
  ...
  default:
    statement;
    break;
  }
```

Python doesn't have a switch statement so they use an if
```
  if arithmetic-expression = const-value-1:
    statement
  elif arithmetic-expression = const-value-2:
    statement
  ...
```

In C/C++, we write a for statement as
```
  for(int i = 0; i < n; i++)
    statement
```
In Python,
```
  for i in xrange(n):
   statement
```
or
```
  for i in range(10,42):
   statement
```

In C/C++, we write a range-based-for statement as
```
  vector<T> v;
  for(auto& e: v)
    statement
```
In Python,
```
  for e in v:
   statement
```

In C/C++, we write a while statement as
```
  while(arithmetic-statement)
    statement
```

In Python,
```
  while arithmetic-expression:
    statement
```

Python does not have a do-while.

C++ try-throw-catch is Python try-raise-except

C/C++ '#include' is a Python 'import'

If you only want one function from an include, for example sin from math
 'from math import sin' does it.

Python dot(X,Y) is the same as C++ inner_product(X.begin(), X.end(), Y.begin(), 0.)

A C++ vector<T> v ={1,2,3} is a Python a = array([[1,2,3]]).T

A 2D C++ vector<vector<T>> v2d ={{1,2,3}, {4,5,6}, {7,8,9}};

In Python
 a2d = array ([[1,2,3], [4,5,6], [7,8,9]])

You don't need to learn to program in Python. You just need to learn to read enough Python to convert it to C++.

Here is link to Python syntax. It is what I use whenever I need to convert Python to C++.

https://www.programiz.com/python-programming/keyword-list

The link should suffice for you to understand enough Python.

The best source for Python documentation is https://docs.python.org/2/tutorial/index.html

For example Python calls [...] lists.
Everything you wanted to know about about Python lists but were afraid to ask can be found at
https://docs.python.org/2/tutorial/datastructures.html

Run the 9 lines of python and compare the output from your C++ code.

Time the Python and C++ code. Which one is faster?

When the output matches, submit your C++ code to black board.

OK, OK. The website has some code, 'main.py', that calculates the same thing.
It uses Python classes and is longer than nine lines. It is 74 lines.
Once you feel comfortable with the nine lines, take a look at 'main.py' and convert it to c++.

Milestone two and three will be similar but will use non-trivial neural net models.
Perhap we will use a third party neural net package such as tensor flow to do face recognition.

```
++++++++++++++++++++++++++++++++++++++++++++ main.py
++++++++++++++++++++++++++++++++++++++++++++

from numpy import exp, array, random, dot


class NeuralNetwork():
    def __init__(self):
        # Seed the random number generator, so it generates the same numbers
        # every time the program runs.
        random.seed(1)

        # We model a single neuron, with 3 input connections and 1 output connection.
        # We assign random weights to a 3 x 1 matrix, with values in the range -1 to 1
        # and mean 0.
        self.synaptic_weights = 2 * random.random((3, 1)) - 1

    # The Sigmoid function, which describes an S shaped curve.
    # We pass the weighted sum of the inputs through this function to
    # normalise them between 0 and 1.
    def __sigmoid(self, x):
        return 1 / (1 + exp(-x))

    # The derivative of the Sigmoid function.
    # This is the gradient of the Sigmoid curve.
    # It indicates how confident we are about the existing weight.
    def __sigmoid_derivative(self, x):
        return x * (1 - x)

    # We train the neural network through a process of trial and error.
    # Adjusting the synaptic weights each time.
    def train(self, training_set_inputs, training_set_outputs, number_of_training_iterations):
        for iteration in xrange(number_of_training_iterations):
            # Pass the training set through our neural network (a single neuron).
            output = self.think(training_set_inputs)

            # Calculate the error (The difference between the desired output
            # and the predicted output).
            error = training_set_outputs - output

            # Multiply the error by the input and again by the gradient of the Sigmoid curve.
            # This means less confident weights are adjusted more.
            # This means inputs, which are zero, do not cause changes to the weights.
            adjustment = dot(training_set_inputs.T, error * self.__sigmoid_derivative(output))

            # Adjust the weights.
            self.synaptic_weights += adjustment

    # The neural network thinks.
    def think(self, inputs):
        # Pass inputs through our neural network (our single neuron).
        return self.__sigmoid(dot(inputs, self.synaptic_weights))


if __name__ == "__main__":

    #Initialise a single neuron neural network.
    neural_network = NeuralNetwork()

    print "Random starting synaptic weights: "
    print neural_network.synaptic_weights

    # The training set. We have 4 examples, each consisting of 3 input values
    # and 1 output value.
    training_set_inputs = array([[0, 0, 1], [1, 1, 1], [1, 0, 1], [0, 1, 1]])
    training_set_outputs = array([[0, 1, 1, 0]]).T

    # Train the neural network using a training set.
    # Do it 10,000 times and make small adjustments each time.
    neural_network.train(training_set_inputs, training_set_outputs, 10000)
    print "New synaptic weights after training: "
    print neural_network.synaptic_weights

    # Test the neural network with a new situation.
    print "Considering new situation [1, 0, 0] -> ?: "
    print neural_network.think(array([1, 0, 0]))
```

++ main.py
++