Machine Learning Using Python Classes

Class Implementation of Single Neuron Model - Notebook Summary

Basic use of Python classes to implement machine learning models.

Scope: This document is a summary of the "Module 2: Class Implementation of Single Neuron Model" notebook from the course "Deep Learning: Mastering Neural Networks" by MIT xPRO. This summary offers a quick overview of the main aspects included in the notebook.

Requirements & Background: This notebook revisits the "Machine Learning Using Python" notebook, and reimplements the single neuron models but now using Python classes. This notebook therefore uses code from the earlier notebook .

Table of Contents

1. Class Implementation for Single Neuron Models

The functions developed for regression and classification models of a single neuron can be turned into methods using Python classes. In addition, the variables (the weights) for a specific model can also be stored as member variables of a specific class instance. In this way, the model becomes a single object integrating both its specific weights and the methods that perform calculations based on those weights.

Remember:

- An object is anything that you wish to manipulate while working through code. Depending on the type (class) of the object, it can have associated data attributes.
- A class is a code template for creating and operating on an object.
- An instance is an individual object of a class.
- An instance attribute is a variable that holds data or properties for that particular instance.
- A method is a function that is part of a class, and which typically operates on instances of the class.

In addition to turning functions into methods within a class, programmers can use the NumPy library to represent vectors instead of using standard Python lists [ ]. NumPy provides benefits over standard lists including the ability to write more concise code, and higher performance operations. For additional resources, please check out NumPy.org for tutorials.

This document focuses on the use of Python libraries and classes for a single neuron regression model. The same Python techniques and tools can be applied to other machine learning methods.

1.a. Python Library: NumPy

The following code script exemplifies how to `import` NumPy and demonstrates some of the features mentioned previously.

```
# Import the NumPy library
import numpy as np

# Create a NumPy array from a list
x = np.array([1,2,3,4])
y = [1,2,3,4]
z = np.array(y)
print("x:", x)
print("y:", y)
print("z:", z)
print("Notice the difference between NumPy arrays x and z and the python list
y.")

# See the dimensions of an array by using shape.
print("Shape of x:", x.shape)

# Create a multi-dimensional array of 2x3 filled with 1's
a = np.ones((2,3))
print("A multidimensional array")
print(a)
print("Shape of a:", a.shape)

# Perform operations on these vectors and arrays without the use of for loops.
x = np.array([1,2])
w = np.array([2,2])
z = np.dot(x,w.T)
print("The dot product of x and transpose of w is:", z)

# Define a scalar variable for subtraction and multiplication of a scalar value
learning_rate = .01
# Define NumPy array to perform subtraction with floats, make sure you initialize
# your array value with float 0. and not the integer 0
# Keep in mind that w_new = np.array([0,0]) would not run
w_new = np.array([0.,0.])
w_new -= learning_rate * x
print("The value of w_new:", w_new)
```

In the following , the `print` results of the script above helps provide an understanding of NumPy's benefits over coding without Python libraries. Examine the `print` results of the script to identify NumPy features.

2. Single Neuron Model Class

In this section, the single neuron models developed in the documentation "Machine Learning using Python " are rewritten to use Python classes and NumPy.

The script on the next page exemplifies the creation of a `SingleNeuronModel` superclass for different types of single neuron models, including regression and classification.

For each of these classes, the member variables `w` and `w_0` remain the same, along with the `forward` function. Also, the initial values of these weights are automatically determined by the input data with the implementation of NumPy and built-in Python functions.

The activation and gradient functions are left unimplemented in the `SingleNeuronModel` superclass as each subclass implements their own version of the activation and gradient functions to differentiate their models.

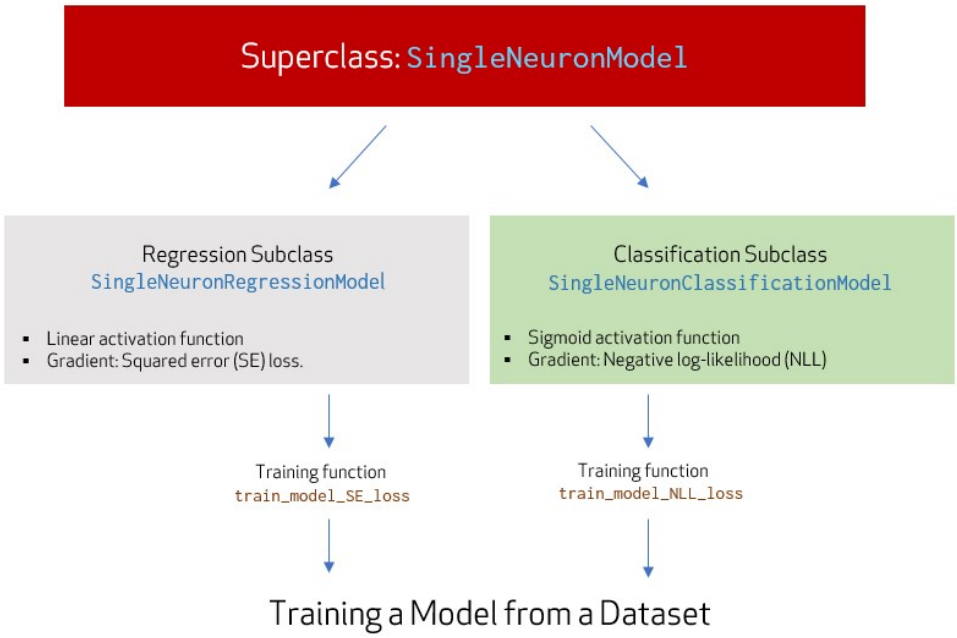
! Important: The code is based on "Machine Learning using Python " and Python libraries. Keep in mind that parts of the code are omitted to avoid repetition and to focus on new concepts. Please refer to the notebook "Module 2: Class Implementation of Single Neuron Model" for fully developed code samples.

The following code block creates the `SingleNeuronRegressionModel` and `SingleNeuronClassificationModel` subclasses and adapts the activation and gradient functions according to the respective machine learning model.

```
# Reimplement our single neuron regression model using the base SingleNeuronModel
class SingleNeuronRegressionModel(SingleNeuronModel):
    # Linear activation function for regression model
    def activation(self, z):
        return z

    # Gradient of output w.r.t. weights, for linear activation
    def gradient(self, x):
        self.grad_w = x
        self.grad_w_0 = 1.


# New implementation! Single neuron classification model
class SingleNeuronClassificationModel(SingleNeuronModel):
    # Sigmoid activation function for classification
    def activation(self, z):
        return 1 / (1 + np.exp(-z) + self.non_zero_tolerance)

    # Gradient of output w.r.t. weights, for sigmoid activation
    def gradient(self, x):
        self.grad_w = self.a * (1-self.a) * x
        self.grad_w_0 = self.a * (1-self.a)
```

2.a. Training a Model from a Dataset

This diagram offers a visual representation of using Python classes and NumPy tools for regression and classification models.



Training a Model from a Dataset

1. Programmer creates a model instance of one of the subclasses:

```
model = SingleNeuronClassificationModel(parameters are inherited from superclass)
```

2. Programmer calls training function with the model instance as one of its argument:

```
train_model_NLL_loss(model, input_data, labels, learning_rate, epochs)
```

We see that common methods expected of a model (including the ability to "forward" calculate an output, given an input) and to "update" model weights based on gradient calculations, are captured in the superclass. Then, different kinds of models, like regression models or classification models, can be specialized as subclasses. These encapsulate the specific activation function to be used for that kind of model, as well as provide methods for calculating gradients of the model. Functions implementing our gradient descent training procedures (with the appropriate loss functions) are called on specific instances of these classes (that is, on a specific "model") in order to learn or train the weights for the model, to best predict the training data.

The code below uses the class implementation to instantiate and train a single neuron classification model

```
# Input data: NumPy array of 2D linearly separable datapoints
input_data = np.array([[1, 1],[1, 5],[-2, 3],[3, -4])
# Corresponding labels
labels = [1, 1, 0, 1]

# Learning rate and epochs
learning_rate = 0.01
epochs = 100

# Create instance of SingleNeuronClassifictionModel subclass
# The in_features parameter is inherited from superclass SingleNeuronModel
# The in_features parameter equals to the length of the first 2D datapoint
model = SingleNeuronClassificationModel(in_features=len(input_data[0]))

# Call training function with instance as argument
train_model_NLL_loss(model, input_data, labels, learning_rate, epochs)
print("\nFinal weights:")
print(model.w, model.w_0)
```

Takeaway: Python classes and libraries allow us to write more flexible and modular code, which makes more efficient and readable machine learning models developed in Python.

With an understanding of classes for machine learning models, programmers can use the machine learning library PyTorch. The next module of this course introduces PyTorch and its features, which provides many pre-defined classes and methods for directly implementing machine learning models.