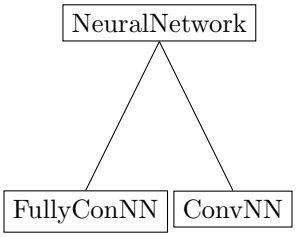
Group Project
GRA-4152 OOP with Python

1044345
1046716

Autumn 2023

1 Developing an Inheritance Hierarchy of Neural Networks

1.1 Inheritance Diagram and Public Interface



The superclass here is the NeuralNetwork and the subclasses that will inherit the superclass are FullyConNN (fully connected neural network) and ConvNN (convolutional neural network).

Public Interface of NeuralNetwork

1._ _init_ _(self, neurons = 50, y_dim = 10): this is the constructor that initializes the NeuralNetwork with specified numbers of neurons and y_dim. The number of neurons and y_dim for both the subclasses are the same. This superclass is inheriting from `tf.keras.Model`.

2.initialize_layers(self, input_shape): this method initializes the layers of the neural network. It takes input_shape as an argument and also initializes the hidden_layers and classifier methods. We chose to have this be a separate method because we had some issues with the timing of when the constructor was created (previously before the `self.hidden` had been constructed.

3. _ _repr_ _(self): this method returns a string description of the NeuralNetwork. It shows a formatted description of the hidden layer and classifier.

4. hidden_layers(self,input_shape, neurons): this method is just a placeholder that is meant to be implemented in the subclasses.

5. classifier(self,neurons,y_dim): this method defines the classifier that is a sequential model. It takes the number of neurons and y_dim as arguments.

6. call(self, inputs): this method takes inputs (where x,y = inputs) as an input parameter which will then calculate a cross entropy loss function.

7. test(self,x): this method takes x as an input parameter which will then calculate y_hat.

8. train(self,inputs,optimizer): this method takes inputs and optimizer as input parameters. This is responsible for training the data and will return loss.

Public Interface of FullyConNN

The public interface for this subclass is inherited from the NeuralNetwork superclass and it includes the methods defined in the superclass. It has its own constructor and hidden_layers method.

1. __init__(self, input_shape =(28*28), neurons = 50, y_dim =10): this is the constructor that initializes the fully connected neural network with specified numbers of input_shape, neurons and y_dim. It calls the superclass constructor and then initializes it.

2. __repr__(self): this method returns a string description of the fully connected neural network. It shows a formatted description of the hidden layer and classifier.

3. hidden_layers(self,input_shape, neurons): this method overrides the hidden_layers method in the superclass. It defines the hidden layers that are specific to the fully connected neural network.

The inherited methods from the NeuralNetwork superclass, such as initialize_layers, call, test, and train, are part of the public interface as well. The attributes neurons and y_dim are inherited from the superclass.

Public Interface of ConvNN

The public interface for this subclass is inherited from the NeuralNetwork superclass and it includes the methods defined in the superclass. It has its own constructor and hidden_layers method.

1. __init__(self, input_shape =(32,32,3), neurons = 50, y_dim =10): this is the constructor that initializes the convolutional neural network with specified numbers of input_shape, neurons and y_dim. It calls the superclass constructor and then initializes it.

2. __repr__(self): this method returns a string description of the convolutional neural network. It shows a formatted description of the hidden layer and classifier.

3. hidden_layers(self, input_shape, neurons, filters=32, kernel_size=3, strides=(2, 2)): This method overrides the hidden_layers method in the superclass. It defines the hidden layers that are specific to the convolutional neural network.

The inherited methods from the NeuralNetwork superclass, such as initialize_layers, call, test, and train, are part of the public interface as well. The attributes neurons and y_dim are inherited from the superclass.

1.2 Inheritance, Overriding, and Polymorphism

1. Inheritance

Inheritance is used to create a relationship between the superclass NeuralNetwork and its subclasses FullyConNN and ConvNN. Both subclasses inherit from NeuralNetwork using "class FullyConNN (NeuralNetwork)" and "class ConvNN (NeuralNetwork) syntax.

Inheritance is also used by the superclass NeuralNetwork as it is inheriting from the tensorflow model tensorflow.keras.Model.

2. Overriding

Overriding is used in both classes under the hidden_layers method. It is overridden to provide a specific information for the hidden layers of each subclass. Under FullyConNN the hidden layer is defined differently to define a fully connected neural network while under ConvNN, the hidden layer is defined differently for a convolutional neural network.

It is also used under __repr__ method for both subclasses which will return a custom string output depending on what neural network is initialized. The overridden __repr__ method includes different information for each subclass regarding hidden layers and classifier.

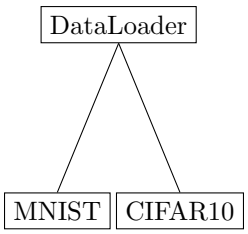
3. Polymorphism

Under the initialize_layers method of the superclass, polymorphism is shown as it is being called in the constructor under the subclasses.

Polymorphism is also evident under the methods call,test, and train in the way that they are called on instances of the subclasses.

2 DataLoader class

2.1 Inheritance Diagram and Public Interface



The superclass here is the DataLoader and the subclasses that will inherit the superclass are MNIST and CIFAR10.

Public Interface of DataLoader

1. _ _init_ _(self): this is the constructor that initializes the DataLoader. The constructor directly loads the data of either MNIST or CIFAR10 and assigns the training and testing variables.This design assumes that the load_data method is implemented in subclasses to handle the specifics of loading the dataset. By calling load_data in the constructor, the DataLoader class ensures that the training and testing data are available as soon as an instance of the class is created.

2. x_tr,y_tr,x_te,y_te: there are four decorators that have been created to be able to access the training and testing datasets.

3. load_data(self): this is an abstract method in the superclass which will be implemented and overridden in the subclasses to load the datasets. It raises a "NotImplementedError" if not implemented in a subclass.

4. preprocess_data(self): this method performs preprocessing steps like converting the datasets into float, scaling it to range [0,1] and converting y_tr and y_te to categorical.

5. loader(self,batch_size): this method loads a tuple of the loaded dataset into the in-build tensorflow data loader.

Public Interface of CIFAR10

The public interface for this subclass is inherited from the DataLoader superclass and it includes the methods and decorators defined in the superclass.
1. load_data(self): this method is an abstract method in the superclass and is inherited by the subclass. This method loads the dataset of CIFAR10 and returns a tuple for training and testing data.

2. preprocess_data(self): this method is also inherited from the superclass and performs the preprocessing needed for the data.

3. loader(self, batch_size): this method is also inherited from the superclass which loads the tuple into the in-build TensorFlow data loader.

Public Interface of MNIST

The public interface for this subclass is inherited from the DataLoader superclass and it includes the methods and decorators defined in the superclass.
1. load_data(self): this method is an abstract method in the superclass and is inherited by the subclass. This method loads the dataset of MNIST and returns a tuple for training and testing data. A part of this method which is different from CIFAR10 is that it reshapes the data from (N,28,28) to (N,784).

2. preprocess_data(self): this method is also inherited from the superclass and performs the preprocessing needed for the data.

3. loader(self, batch_size): this method is also inherited from the superclass which loads the tuple into the in-build tensorflow data loader.

2.2 Inheritance, Overriding, and Polymorphism

1. Inheritance

Inheritance is used to create a relationship between the superclass DataLoader and its subclasses MNIST and CIFAR10. Both subclasses inherit from DataLoader using "class MNIST (DataLoader)" and "class CIFAR10 (DataLoader) syntax.

2. Overriding

Both subclasses (MNIST and CIFAR10) override the load_data method inherited from the DataLoader superclass. This allows them to provide dataset-specific loading logic while reusing the common structure defined in the superclass.

The MNIST subclass also overrides the preprocess_data method from the DataLoader superclass. This allows it to include additional steps specific to the MNIST dataset, such as reshaping the input data.

3. Polymorphism

In these subclasses, polymorphism is used mainly by overriding the method load_data from both MNIST and CIFAR10. Polymorphic behavior ensures that the appropriate load_data and preprocess_data methods are called based on the actual type of the object. This flexibility and interchangeability are key aspects of polymorphism.

Running the code via terminal
Using the terminal and running it there using the following syntax:

"python train.py –dset cifar10 –nn_type conv –epochs 10"

returns an AUC for CIFAR10 dataset with a convolutional neural network.

While using this syntax:

"python train.py –dset cifar10 –nn_type conv –epochs 10"

returns an AUC for MNIST dataset with a fully connected neural network.