MNIST and CIFAR10 Neural Networks

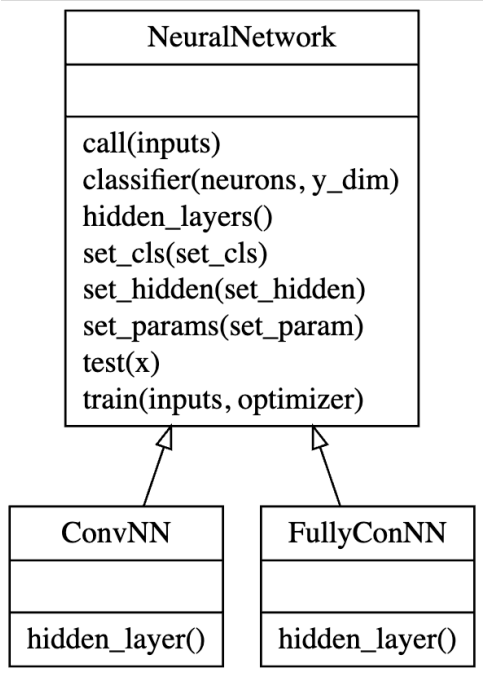
GRA-4152

BI Norwegian Business School
November 19, 2023

1 NeuralNetwork: classes and subclasses

1.1 Inheritance diagram

NeuralNetwork is the superclass of our project. This class is inherited by two subclasess: ConvNN and FullyConNN. The following image shows the inheritance diagram of our classes.



The ConvNN and FullyConNN subclasses differ in the hidden_layer method. This method creates the hidden layers for the different neural networks. The only inheritance that is not visible in the chart is that the NeuralNetwork superclass inherits from the tf.keras.Model class. The public interface is different for the two subclasses. While for the ConvNN class, the user can define the input shape, neurons, filters, kernel size, and strides parameters, for the FullyConNN class the possible inputs are only input shape and neurons. Moreover, with creating an instance for any of these classes the constructor invokes all necessary methods thus the neural network is ready to be trained. In addition, the __repr__ method can be used to print information about the class.

1.2 Methods used in these classes

- **__init__():** the constructor of the superclass inherits the constructor from the keras Model class In addition, the constructor creates hidden, cls, params and neurons instance variables. Neurons instance variable is created in the superclass because we need the parameter both for the classifier and hidden_layer method. The last method is only implemented in the subclass. This method receives as parameter the number of neurons for each layer in the hidden layers and the classifier.
- **__repr__():** this method prints information about the class. This is an abstract method, implemented later in the subclass.
- **hidden_layers():** this method returns the hidden layers for a neural network. This is an abstract method, implemented later in the subclass.
- **call():** this method returns the loss function based on the hidden and cls and the input values. Receives as parameters the inputs (x,y) a tuple where x is the input data and y is the result and returns the hidden layer object.

- **classifier():** this method defines the classifier layer in the neural network. This layer is the same for both subclasses. Receives as parameters the number of classes in the classification task and returns the classifier layer object.
- **train():** the train helps to train the model with the training data. Receives as parameters:
  - **inputs:** a tuple (x_tr, y_tr) wheres x is the training dataset and y is the training dataset in order to train the model.
  - **optimizer:** optimizer algorithm to improve the weights in the neural networks. The default is Adam.

  After the training, returns the loss function of the neural network.
- **test():** The test method returns the estimated class and the probability for each class based on the trained model and the test set input values.
- **set_hidden():** this is an auxiliar method that sets the hidden instance variable to a new object.
- **set_params():** this is an auxiliar method that sets the trainable parameters to a new object.
- **set_cls():** this is an auxiliar method that sets the parameters of the classifier layer.

1.3 Inheritance, overriding and polymorphism

Inheritance: we used inheritance first when the NeuralNetwork class inherited from the keras Model class. Then the subclasses, ConvNN and FullyConNN inherited the implemented methods from NeuralNetwork superclass. For instance, the classifier method should be implemented in the same way for both subclasses, therefore it can be implemented in the superclass.

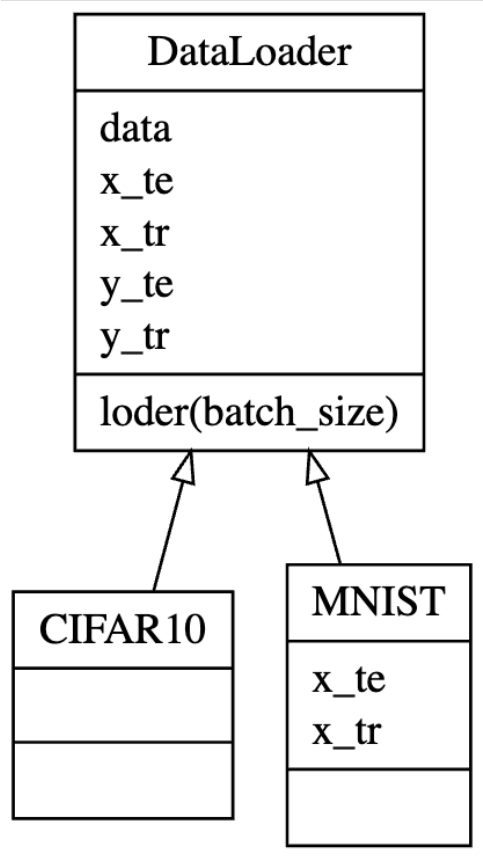
Overriding: is used for example in the case of the call method that is inherited from tf.keras.Model class. In the NeuralNetworks class we override the call method to give us the chosen loss function.

Polymorphism: is used for the hidden_layer method. Both subclasses should have this method, but the implementation should be different for these classes. Therefore, first, we created an abstract method in the NeuralNetworks superclass, and this method is inherited by the subclasses. In the subclasses this not implemented method is overridden in a different way for the different classes. Finally, this hidden_layer method is implemented for all subclasses but works in a different way.

2 Dataloader: classes and subclasses

2.1 Inheritance diagram

DataLoader is another superclass of our project. This class is not as important as important as NeuralNetwork but it plays a vital role by being responsible for loading the data correctly. DataLoader has two subclasses: MNIST and CIFAR10, these are used to load specific datasets. The following image shows the inheritance diagram of our classes.



Both CIFAR10 and MNIST extend the constructor method of DataLoader loading the keras datasets CIFAR10 and MNIST respectively. For the MNIST class specifically the methods x_te and x_tr are overriden in order to change the shape of the arrays that they return.

2.2 Methods used in the class

Both MNIST and CIFAR10 have two types of methods:

- **Accessor methods:** there are 5 accessor methods in each class *data, x_te, x_tr, y_te* and *y_tr*
  - **data:** returns a tuple with the training and test datasets
  - **x_te, x_tr, y_te** and **y_tr:** return a specific part of data. This can be either a training or test dataset and can be for either the group of independent variables or for the dependent variable.
- **loder:** this is the main method of the class, receives as an input the number of batches to be train and returns an array of this size of tuples containing the independent variables and the dependent variable.

2.3 Inheritance, overriding

Inheritance: is used in every method, but for all the methods of CIFAR10 and for the methods *data, y_te, y_tr* and *loder* not overriding is required and its final implementation is completely inherited from the DataLoader super class.

Overriding: is used in __init__() method to load either the MNIST or CIFAR10 dataset. Additionally in MNIST is used in *x_tr* and *x_te* to the extend the functionality of these methods.

Firstly the super method is inherited and then its output is reshaped to a more suitable form to be trained later by the neural networks.

3 Training and testing

To train and test the model a specific file was created *train.py*. This file takes advantage of the argparse class in python in order to receive parameters and to test the neural networks. The argparse has two components

- **doc:** this argument needs to be specified in order to get access to the documentation of the classes. To get the documentation, the class name is needed. Additionally, the methods argument can be specified to get the documentation of each method in the class:

```
python train.py --doc --class_name CIFAR10 --methods
```

- **test:** this argument is used to test the neural networks. The batch size, the number of epochs to be trained, the dataset, the type of neural network and the number of neurons of the model can all be modified to test the program.

```
python train.py --test --dset cifar10 --nn_type ConvNN --epochs 10
python train.py --test --dset mnist --nn_type FullyConNN --epochs 10
```