Assignment No 11

Title: How to Train a Neural Network with TensorFlow/ Pytorch and evaluation of logistic regression using tensorflow

Problem Statement:
How to Train a Neural Network with TensorFlow/Pytorch and evaluation of logistic regression using tensorflow

Objective:

- Training a Neural Network
- Evaluation of Logistic Regression (using TensorFlow)

Theory:

What is Tensor Flow:

TensorFlow allows developers to create dataflow graphs—structures that describe how data moves through a graph, or a series of processing nodes. Each node in the graph represents a mathematical operation, and each connection or edge between nodes is a multidimensional data array, or tensor.

TensorFlow applications can be run on most any target that's convenient: a local machine, a cluster in the cloud, iOS and Android devices, CPUs or GPUs. If you use Google's own cloud, you can run TensorFlow on Google's custom TensorFlow Processing Unit (TPU) silicon for further acceleration. Models created by TensorFlow can be deployed on most any device to serve predictions.

what is Pytorch?

PyTorch is an optimized Deep Learning tensor library based on Python and Torch and is mainly used for applications using GPUs and CPUs. PyTorch is favored over other Deep Learning frameworks like TensorFlow and Keras since it uses dynamic computation graphs and is completely Pythonic. It allows scientists, developers, and neural network debuggers to run and test portions of the code in real-time. Thus, users don't have to wait for the entire code to be implemented to check if a part of the code works or not.

The two main features of PyTorch are:

- Tensor Computation (similar to NumPy) with strong GPU (Graphical Processing Unit) acceleration support
- Automatic Differentiation for creating and training deep neural networks

What is regression?

Regression is a statistical method used to model the relationship between a dependent variable (often denoted as Y) and one or more independent variables (often denoted as X). The

goal of regression analysis is to understand how the independent variables influence the dependent variable and to predict the value of the dependent variable based on the values of the independent variables.

What is logistic regression?

Logistic regression is the appropriate regression analysis to conduct when the dependent variable is dichotomous (binary). Like all regression analyses, logistic regression is a predictive analysis. It is used to describe data and to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables.

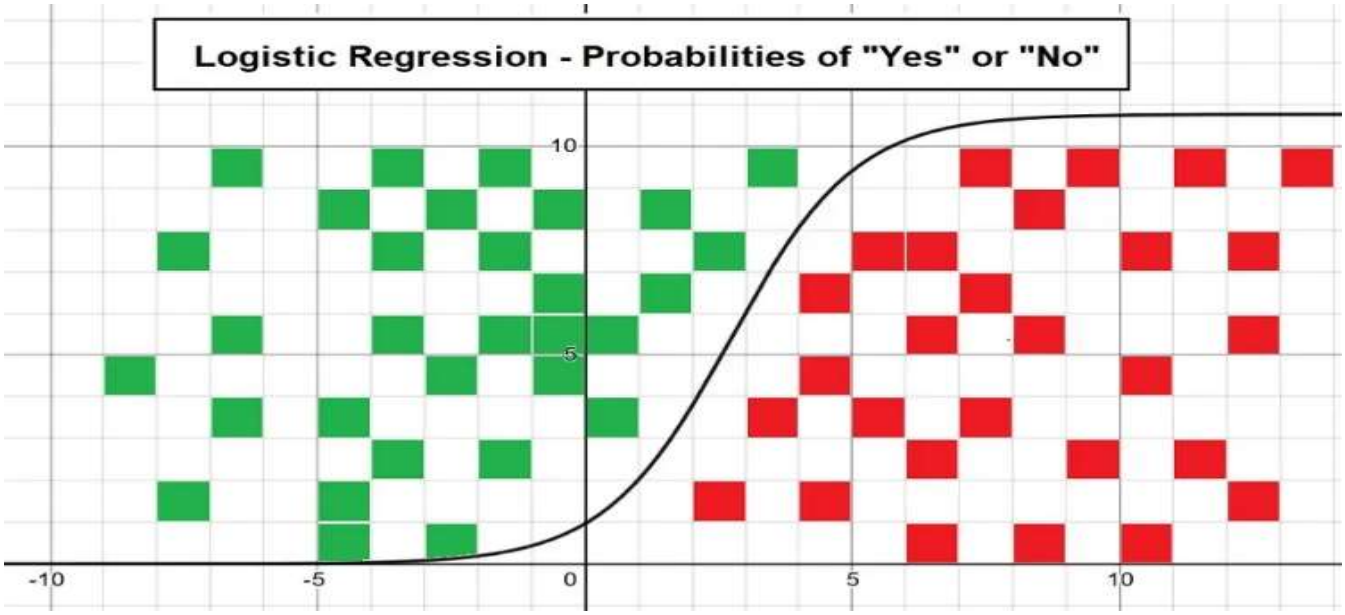
Logistic Regression is another statistical analysis method borrowed by Machine Learning. It is used when our dependent variable is dichotomous or binary. It just means a variable that has only 2 outputs, for example, A person will survive this accident or not, The student will pass this exam or not. The outcome can either be yes or no (2 outputs). This regression technique is similar to linear regression and can be used to predict the Probabilities for classification problems.

Type of Logistic Regression:

1. Binomial: There can be only two possible types of dependent variables, such as 0 or 1, Yes or No, etc.
2. Multinomial: There can be three or more possible unordered types of the dependent variable, such as “cat,” “dogs,” or “sheep.”
3. Ordinal: There can be three or more possible ordered types of dependent variables, such as “low,” “Medium,” or “High.”

Example of Logistic Regression Plot

The following plot shows logistic regression by considering the probabilities of “Yes” and “No”:



Assumptions of logistic regression

- The output data is yes and no
- Linearity
- Little outliers
- Independence in value

As we mentioned above, the answer to our logistic regression model is between 0 and 1. To ensure our output is at that value, we have to squash that data into that range. The best tool for doing this would be a sigmoid function.

Algorithm:
Step 1: Import all necessary modules
Step 2: Loading and preparing the mnist data set
Step 3: Setting up hyper parameters and data set parameters.
Step 4: Shuffling and batching the data
Step 5: Initialize weight and biases
Step 6: Define logistic regression and cost function
Step 7: Defining the optimizers and updating weight and biases
Step 8: Optimization process and updating weight and biases
Step 9: The training loop
Step 10: Testing model accuracy using the test data.

Conclusion:
We implement Neural network with TensorFlow/ Pytorch and evaluation of logistic regression using tensorflow

Assignment No 12:

Title: TensorFlow/Pytorch implementation of CNN

Problem Statement:

TensorFlow/Pytorch implementation of CNN

Objective:

Implement CNN using TensorFlow/Pytorch

Theory:

What is CNN:

Convolutional Neural Networks(CNN) is a type of Deep Learning algorithm which is highly instrumental in learning patterns and features in images. CNN has a unique trait which is its ability to process data with a grid-like topology whereas a typical Artificial Neural Network(Dense or Sparse) generally takes input by flattening the tensors into a one-dimensional vector. This facilitates it to learn and differentiate between features in images, which when represented digitally are essentially a grid of numbers.

Convolutional Neural Networks are typically comprised of multiple layers. Usually, the initial layers are used to detect simple features such as edges, and complex features are detected down the line, as we go deeper into the network.

CNN has countless qualities that make it so suitable for processing images. Let's take a look at some of them:-

- They require much less data pre-processing than other Deep Learning Algorithms.
- A well-trained CNN model has the ability to learn and classify features in an image, which gives much better accuracy in the classification and detection of features in images.
- It can save a lot of computational resources by methods like increasing the convolutional and pooling layers.

What are Convolutional and Pooling Layers in CNN?

Convolutional Layers:

These are the first layers in a CNN, and they can be thought of as "Filters" for an image. Just like Filters in Instagram detect our face, a convolutional layer detects features or filters such as edges in an image, wherever they might be present.

Pooling Layers:

The pooling layers mainly reduce the computational cost by reducing the spatial size of the image. The best way to describe it would be that it makes the grids of information smaller by taking a "lump-sum" of the images' spatial resolution.

Stepwise implementation

Step 1: Load and Preprocess the Dataset

- Load the dataset you want to work with (e.g., CIFAR-10, MNIST).
- Preprocess the data by normalizing pixel values to the range [0, 1] and reshaping if necessary.
- Split the dataset into training and testing sets.

Step 2: Define the CNN Model Architecture

- Initialize a Sequential model using tf.keras.models.Sequential().
- Add convolutional layers using tf.keras.layers.Conv2D() with appropriate parameters such as number of filters, kernel size, and activation function.
- Add pooling layers using tf.keras.layers.MaxPooling2D() to downsample the feature maps.
- Optionally, add additional convolutional and pooling layers for deeper network architecture.
- Flatten the 2D feature maps into a 1D vector using tf.keras.layers.Flatten().
- Add fully connected (Dense) layers using tf.keras.layers.Dense() for classification, with appropriate activation functions.

Step 3: Compile the Model

- Compile the model using model.compile() function.
- Choose an optimizer (e.g., Adam, SGD) and specify its parameters.
- Specify the loss function (e.g., categorical cross-entropy, sparse categorical cross-entropy) appropriate for your task.
- Optionally, specify additional metrics to monitor during training (e.g., accuracy).

Step 4: Train the Model

- Train the model on the training data using model.fit() function.
- Specify the training data, number of epochs, batch size, and validation data (if available).
- Monitor the training progress and adjust hyperparameters as needed.

Step 5: Evaluate the Model

- Evaluate the trained model on the testing data using model.evaluate() function.
- Calculate metrics such as accuracy, loss, and any additional metrics specified during compilation.

Step 6: Fine-tuning and Optimization

- Experiment with different model architectures, hyperparameters, and optimization algorithms to improve performance.
- Apply regularization techniques such as dropout or L2 regularization to prevent overfitting.
- Perform hyperparameter tuning using techniques such as grid search or random search.
- Explore techniques such as data augmentation to increase the diversity of training data and improve generalization.

Step 7: Deployment and Inference

- Once satisfied with the model performance, deploy the model for inference on new data.
- Serialize the trained model using TensorFlow's SavedModel format or HDF5 format.
- Integrate the model into your application or deploy it to a production environment for real-time predictions.

By following these step-by-step instructions, you can implement a CNN using TensorFlow for image classification tasks and achieve state-of-the-art performance on various datasets.

Conclusion:

We learn implementation of CNN using TensorFlow for image classification

Assignment No 13

Title: MNIST Handwritten Character Detection using PyTorch, Keras and Tensorflow

Problem Statement:
MNIST Handwritten Character Detection using PyTorch, Keras and Tensorflow

Objective:
Implement MNIST Handwritten Character Detection using PyTorch, Keras and Tensorflow

Theory:
Handwritten digit recognition using MNIST dataset is a major project made with the help of Neural Network. It basically detects the scanned images of handwritten digits.
The handwritten digit recognition is the ability of computers to recognize human handwritten digits. It is a hard task for the machine because handwritten digits are not perfect and can be made with many different flavors. The handwritten digit recognition is the solution to this problem which uses the image of a digit and recognizes the digit present in the image.

Approach:
We will approach this project by using a three-layered Neural Network.

- The input layer: It distributes the features of our examples to the next layer for calculation of activations of the next layer.
- The hidden layer: They are made of hidden units called activations providing nonlinear ties for the network. A number of hidden layers can vary according to our requirements.
- The output layer: The nodes here are called output units. It provides us with the final prediction of the Neural Network on the basis of which final predictions can be made.
- A neural network is a model inspired by how the brain works. It consists of multiple layers having many activations, this activation resembles neurons of our brain. A neural network tries to learn a set of parameters in a set of data which could help to recognize the underlying relationships. Neural networks can adapt to changing input; so the network generates the best possible result without needing to redesign the output criteria.

What is Keras?

Keras is an effective high-level neural network Application Programming Interface (API) written in Python. This open-source neural network library is designed to provide fast experimentation with deep neural networks, and it can run on top of CNTK, TensorFlow, and Theano.
Keras focuses on being modular, user-friendly, and extensible. It doesn't handle low-level computations; instead, it hands them off to another library called the Backend.

Keras was adopted and integrated into TensorFlow in mid-2017. Users can access it via the tf.keras module. However, the Keras library can still operate separately and independently.

What is PyTorch?

PyTorch is a relatively new deep learning framework based on Torch. Developed by Facebook's AI research group and open-sourced on GitHub in 2017, it's used for natural language processing applications. PyTorch has a reputation for simplicity, ease of use, flexibility, efficient memory usage, and dynamic computational graphs. It also feels native, making coding more manageable and increasing processing speed.

What is TensorFlow?

TensorFlow is an end-to-end open-source deep learning framework developed by Google and released in 2015. It is known for documentation and training support, scalable production and deployment options, multiple abstraction levels, and support for different platforms, such as Android.

TensorFlow is a symbolic math library used for neural networks and is best suited for dataflow programming across a range of tasks. It offers multiple abstraction levels for building and training models.

Implementation

Step 1: Import the libraries and load the dataset

we are going to import all the modules that we are going to need for training our model. The Keras library already contains some datasets and MNIST is one of them. So we can easily import the dataset and start working with it. The mnist.load_data() method returns us the training data, its labels and also the testing data and its labels.

Step 2: Preprocess the data

The image data cannot be fed directly into the model so we need to perform some operations and process the data to make it ready for our neural network. The dimension of the training data is (60000,28,28). The CNN model will require one more dimension so we reshape the matrix to shape (60000,28,28,1).

Step 3: Create the model

Now we will create our CNN model in Python data science project. A CNN model generally consists of convolutional and pooling layers. It works better for data that are represented as grid structures, this is the reason why CNN works well for image classification problems. The dropout layer is used to deactivate some of the neurons and while training, it reduces offer fitting of the model. We will then compile the model with the Adadelta optimizer.

Step 4: Train the model

The model.fit() function of Keras will start the training of the model. It takes the training data, validation data, epochs, and batch size.

It takes some time to train the model. After training, we save the weights and model definition in the 'mnist.h5' file.

Step 5: Evaluate the model

We have 10,000 images in our dataset which will be used to evaluate how good our model works. The testing data was not involved in the training of the data therefore, it is new data for our model. The MNIST dataset is well balanced so we can get around 99% accuracy.

Step 6: Create GUI to predict digits

Now for the GUI, we have created a new file in which we build an interactive window to draw digits on canvas and with a button, we can recognize the digit. The Tkinter library comes in the Python standard library. We have created a function predict_digit() that takes the image as input and then uses the trained model to predict the digit.

Conclusion: Hence we implemented MNIST Handwritten Character Detection using PyTorch, Keras and Tensorflow

OutPut:

```
from keras.models import load_model
from tkinter import *
import tkinter as tk
import win32gui
from PIL import ImageGrab, Image
import numpy as np

model = load_model('mnist.h5')

def predict_digit(img):
    #resize image to 28x28 pixels
    img = img.resize((28,28))
    #convert rgb to grayscale
    img = img.convert('L')
    img = np.array(img)
    #reshaping to support our model input and normalizing
    img = img.reshape(1,28,28,1)
    img = img/255.0
    #predicting the class
    res = model.predict([img])[0]
    return np.argmax(res), max(res)

class App(tk.Tk):
    def __init__(self):
        tk.Tk.__init__(self)

        self.x = self.y = 0

        # Creating elements
        self.canvas = tk.Canvas(self, width=300, height=300, bg = "white", cursor="cross")
        self.label = tk.Label(self, text="Thinking..", font=("Helvetica", 48))
        self.classify_btn = tk.Button(self, text = "Recognise", command =         self.classify_handwriting)
        self.button_clear = tk.Button(self, text = "Clear", command = self.clear_all)

        # Grid structure
        self.canvas.grid(row=0, column=0, pady=2, sticky=W, )
        self.label.grid(row=0, column=1,pady=2, padx=2)
        self.classify_btn.grid(row=1, column=1, pady=2, padx=2)
        self.button_clear.grid(row=1, column=0, pady=2)

        #self.canvas.bind("<Motion>", self.start_pos)
        self.canvas.bind("<B1-Motion>", self.draw_lines)

    def clear_all(self):
```

```
        self.canvas.delete("all")

    def classify_handwriting(self):
        HWND = self.canvas.winfo_id() # get the handle of the canvas
        rect = win32gui.GetWindowRect(HWND) # get the coordinate of the canvas
        im = ImageGrab.grab(rect)

        digit, acc = predict_digit(im)
        self.label.configure(text= str(digit)+', '+ str(int(acc*100))+'%')

    def draw_lines(self, event):
        self.x = event.x
        self.y = event.y
        r=8
        self.canvas.create_oval(self.x-r, self.y-r, self.x + r, self.y + r, fill='black')

app = App()
mainloop()
```