Table of Contents

Exercise 4 - Machine Learning

Part A

1. Learn about Pytorch. Install Pytorch. Uses of tools like Jupyter Notebook, Google Colab.
2. Vector, Matrix, Tensor, Gradient descent, Numpy, Torch, TorchVision.
3. Process of vectorization
4. Linear regression model.
5. Dataset, Weight, Bias. Parameters Vs HyperParameters.
6. Generating prediction using linear regression model.
7. Loss function Vs Cost function.
8. Mean squared error.
9. Forward Vs Backward function.
10. Relationship between gradient descent and weights and biases.
11. Epoch, Benefits of increasing epoch.
12. Learning rate.
13. Optimizer in Pytorch

Practical: Train a model with unstructured data with a minimum of 100 epochs and prove that increasing epoch decreases the loss.

Part B

1. Linear regression vs Logistic regression. Model?
2. Importance of batch size while training.
3. What is cross entropy? Accuracy vs Loss function.
4. Importance of hidden layer. Deep Neural Network.**Practical:**Train the same model of Part 1 with one hidden layer.Document the performance improvement on using this layer.
5. Training a model?
6. Necessity of GPU in training. Device parameter? **Practical**:Compare the training times on a CPU vs. GPU

Part C

1. Convolution Neural Network. nn.Sequential , nn.Conv2d, nn.ReLU, nn.MaxPool2d layers.
2. Overfitting, How to avoid it? Learning rate scheduling? Weight decay. Gradient clipping?
3. Regularization of data? Data Normalization and Augmentation?
4. Importance of testset validation.
5. Audio - https://dylanmeeus.github.io/posts/audio-from-scratch-pt1/

6. https://pudding.cool/2018/02/waveforms/
7. Signal processing techniques.
8. Importance of ONNX

Practical Tasks for Machine Learning(Audio Processing) - Option 1:

1. Load an audio of 16000Hz frequency into tensor using pytorch.
2. Resample the audio using pytorch to 48000 Hz frequency. This audio will be considered as original audio. [https://pytorch.org/tutorials/beginner/audio_preprocessing_tutorial.html#resampling]
3. Apply STFT on this audio and store the magnitude and phase.
4. Now load the magnitude and phase from the stored data and convert this into complete audio using ISTFT.
5. Compare the torch points of original audio and result audio. This should be same.
6. Calculate torchmetrics PESQ and STOI for original audio and resultant audio and compare the result. This should be same.
7. Perform this same operation in cpp. i.e, Create an application in cpp which uses STFT and gets the magnitude and phase component from the audio.
8. Use Pybind and get this magnitude and phase returned from cpp into Python.
9. Compare the magnitude and phase return by both cpp(using Pybind) and python using Pytest.

Practical Tasks for Machine Learning(Audio Processing) - Option 2:

1. Load an audio of 16000Hz frequency into tensor using pytorch.
2. Apply STFT on this audio and store the magnitude and phase.
3. Apply Mel-filter on this magnitude and find MelSpectrogram.
4. Load the same audio in cpp.
5. Apply STFT using cpp and find magnitude and phase.
6. Apply Mel-filter on this STFT magnitude with the same attributes as was done in python.
7. Design a python program where you could get the output[STFT and Mel] of cpp program using pybind.
8. Now compare the values and ensure both are same.
9. Write a Pytest to prove the same.

Practical Tasks for Machine Learning(Audio Processing) - Option 3:

1. Create python library which takes audio as input and provides magnitude after applying RFFT on audio.

2. Export this method of python library into onnx model.
3. Run this onnx model and pass the audio and get magnitude as output.
4. Create a cpp program which takes audio as input and stores this audio as array.You can use any thirdparty library for this.
5. Access the onnx model using cpp program and get the magnitude as output. [**Hint**: https://onnxruntime.ai/docs/tutorials/mnist_cpp.html]
6. Increase the intra-processing threads to 10 and note the performance changes.
7. Perform RFFT in cpp and compare the values obtained from onnx to that calculated in cpp itself.
8. Ensure comparison says that the result are same from python and cpp.

Practical Tasks for Machine Learning(Audio Processing) - Option 4:

1. Create python library which will get audio as input using pytorch
2. Apply the following augmentation techniques: Reverberation, Reduce the speed by 0.5
3. Now write a cpp program to get the audio input and store it as array.
4. Use python bindings, to read the array of audio return from cpp program and apply augmentation technique listed above and create a new audio file with the augmentation changes.
5. Calculate the torchmetrics PESQ and STOI on the audio files with and without the augmentation changes.
6. Provide the output in the form of a CSV file.

Practical Tasks for Machine Learning(Audio Processing) - Option 5:

Learn about Streamlit.
Develop an streamlit application, which should

1. List the zips in the directory.This zip should contain audio files.
2. On clicking a zip, extract the zip. List the audio in zip. Provide option to play audio in UI.
3. Write the attributes of the audio like sampling rate, channels, encoding , sample size in a CSV file.
4. Upload this CSV file in UI.
5. While displaying the attributes in UI, show the audio as well.
6. If the component size gets larger, add necessary functionality like scrolling.

If it takes time to upload the audio, indicate it by a loading symbol.

Practical Tasks - Part II

1. Install VM in your machine
2. Add an OS(Ubuntu) as virtual machine.
3. Now run the same cpp application done in the above program in virtual machine.
4. Initiate remote debugger.
5. Check the stack trace while debugging and note down the CPU and memory usage during execution of every module.
6. Document the module occupying highest CPU and memory usage and find the reason for the usage.
7. Understand the usage of CPU register caches and note down the importance of each cache by disabling each cache separately.