CS 464

Introduction to Machine Learning

Fall 2021

Homework 3

Due: Jan 02, 2022 17:00 (GMT+3)

Instructions

- This homework contains both written and programming questions about neural networks. You should implement programming questions on this notebook. Your plots should also be produced in this notebook. Each programming question has its own cell for your answer. You can implement your code directly in these cells, or you can call required functions which are defined in a different location for the given question.
- For questions that you need to plot, your plot results have to be included in the cell output. For written questions, you may provide them either as comments in code cells or as seperate text cells.
- You are **NOT ALLOWED** to use different libraries than given libraries in the code segments of this homework except for libraries inclueded in Python Standard Library (https://docs.python.org/3/library/).
- You are **NOT ALLOWED** to use a different deep learning framework than PyTorch.
- While submitting the homework file, please package notebook(".ipynb") and model (".pth") files as a gzipped TAR file or a ZIP file with the name cs464_hw3_section#_Firstname_Lastname. Please do not use any Turkish letters for any of your files including code files and model files. Upload your homework to Moodle.
- This is an individual assignment for each student. That is, you are NOT allowed to share your work with your classmates.
- If you do not follow the submission routes, deadlines and specifications, it will lead to a significant grade deduction.
- If you have any questions, please contact "hakansivuk@gmail.com".

Environment Setup

This homewrok is prepeared by using Google CoLab which already has required libraries. However, if you are using your own local Jupyter or any other Python notebook editor, you may use both anaconda or pip to install PyTorch to your own computer.

Anaconda Installation

- Download anaconda from https://www.anaconda.com/download
- Follow the instructions provided in https://conda.io/docs/user-guide/install/index.html#regular-installation

Creation of Virtual Environment

- Create python3.7 virtual environment for your hw3 using follow command from the command line
  > conda create -n HW3 python=3.7 anaconda
- Activate your virtual environment
  > source activate HW3
- To install auxiliary libraries, replace the "package_name" in the following command and run it in activated "hw3" environment
  > pip install "package_name"
- *When you create your virtual environment with "anaconda" metapackage, jupyter notebook should be installed. Try:*
  > jupyter notebook

Pytorch Installation with Anaconda

You should install PyTorch to your virtual environment which is created for the hw3. Therefore, you should activate your homework virtual environment before to start PyTorch installation.

- *> source activate HW3*

After you have activated the virtual environment, then use one of the following commands to install pytorch for CPU for your system. See https://pytorch.org/ for help.

- *For MacOS:*
  > conda install pytorch torchvision -c pytorch
- *For Linux:*
  > conda install pytorch-cpu torchvision-cpu -c pytorch
- *For Windows:*
  > conda install pytorch-cpu torchvision-cpu -c pytorch

Pip3 Installation

- *Download pip3 from https://pip.pypa.io/en/stable/installing/*
- *If you are using Windows, you may need to add Python to your enviroment variables. You may use the following tutorial to install Python and pip. https://phoenixnap.com/kb/how-to-install-python-3-windows*

PyTorch Installation with Pip

- *For MacOS:*
  > pip3 install torch torchvision
- *For Linux:*
  > pip3 install torch==1.3.1+cpu torchvision==0.4.2+cpu -f https://download.pytorch.org/whl/torch_stable.html
- *For Windows:*
  > pip3 install torch==1.3.1+cpu torchvision==0.4.2+cpu -f https://download.pytorch.org/whl/torch_stable.html

Question 1 [12 pts.]

*Answer the given questions with **at most a sentence**.*

a) Why do people use validation data?

b) What is the difference between mean squared error and mean absolute error?

c) What is the main problem of using sigmoid as activation function in an artificial neural network (ANN)?

d) What does it mean to overfit your data model?

e) Your input image size is 3x64x64. If you apply 3x3 convolution with input_channel=3, output_channel=6, padding=0, stride=2, what would be the size of the output?

f) In the previous question, how many trainable parameters are there? (you should also consider bias terms in addition to weights)

Question 2 [88 pts.]

Computer vision (CV) is the field of study that deals with how computers can gain high-level understanding from digital images or videos. Your task for this question is to classify scenes according to their contexts by using simple machine learning algorithms developed for CV problems on scene images.

Your dataset consist of scene images from 4 contexts. Images of each context is stored under separate folders in the compressed file given to you. The dataset has been processed in such a way that each class has approximately 2500 samples.

Download the dataset from the following link:
https://drive.google.com/file/d/1l51t3aTY7B131fwq92ACI_b_D5Idq5In/view?usp=sharing

Libraries that are required in this question is given in the following code cell.

In [ ]:

```
# Mount Google Drive
from google.colab import drive
drive.mount('/content/drive')
# PyTorch
import torch
import torch.nn as nn
import torch.nn.functional as F
import torchvision.models as models

# To Read Data
from torch.utils.data import Dataset, DataLoader
import numpy as np
from PIL import Image
# To Interpret results & obtain plots
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, f1_score, precision_score, recall_score, accuracy_score
import matplotlib.pyplot as plt

# You could add your own libraries form Python Standard Library in this cell. Any other external librari
```

Data Loader [6 pts.]

An important part of such a task is to implement your own data loader. In this homework, a partial loader is provided to you. This loader is going to be based on a base class named "Dataset", provided in PyTorch library. You need to complete the code below to create your custom "SceneDataset" class which will be able to load your dataset. Implement the functions whose proptotypes are given. Follow the TODO notes below. You have to divide the files into three sets as ***train (70%)****,* ***validation (10%)*** *and* ***test (20%)*** *sets. These non-overlapping splits, which are subsets of SceneDataset, should be retrieved using the "get_dataset" function.*

Hint: The dataset is not normalized and your results will heavily depend on your input.

In [ ]:

```
class SceneDataset(Dataset):
    # TODO:
    # Define constructor for SceneDataset class
    # HINT: You can pass processed data samples and their ground truth values as parameters
    #def __init__(self, **kwargs): # you are free to change parameters

    '''This function should return sample count in the dataset'''
    #def __len__(self):
    #    return self.data.shape[0]

    '''This function should return a single sample and its ground truth value from the dataset correspon
    #def __getitem__(self, index):
        #return _x, _y
```

In [ ]:

```
def get_dataset(root):
    # TODO:
    # Read dataset files
    # Construct training, validation and test sets
    # Normalize datasets

    #return train_dataset, val_dataset, test_dataset#
```

Model Implementation [7 pts]

Now implement your CNN. ConvNet class will represent your convolutional neural network. Implement 3 layers of convolution:

- *(1) 4 filters with size of 3 x 3 with stride 1 and padding 1, (2) ReLU*
- *(3) 8 filters with size of 3 x 3 with stride 1 and padding 1, (4) ReLU and (5) MaxPool 2 x 2*
- *(6) 16 filters with size of 3 x 3 with stride 1 and padding 1, (7) ReLU and (8) MaxPool 2 x 2*

As the classifier layer, you need to add only one linear layer at the end of the network. You need to choose the appropriate input and output neuron sizes and the activation function for the dense layer.

In [ ]:

```
class ConvNet(nn.Module):
    '''Define your neural network'''
    def __init__(self, **kwargs): # you can add any additional parameters you want
    # TODO:
    # You should create your neural network here

    def forward(self, X): # you can add any additional parameters you want
    # TODO:
    # Forward propagation implementation should be here
```

Stochastic Gradient Descent [25 pts.]

Training with SGD [15 pts.]

Train your model up to 300 epochs with properly processed inputs, i.e. call your "get_dataset" function. Use SGD as your optimizer. Tune your learning rate, weight decay. Do not add additional parameters to SGD. Save your best model as "best_cnn_sgd.pth". The best model should be selected based on validation dataset. You could use any measurement and/or metric to decide on the best model. However, you must explain your reasoning in your choice.

During training, you need to plot two figures:

1. *training loss and validation loss vs. epoch*
2. *training accuracy and validation accuracy vs. epoch*

Name your axes and plots properly.

In [ ]:

```
# HINT: note that your training time should not take more than 2 hours.
```

```
# TODO:
# Pick your hyper parameters
max_epoch = 300
# train_batch =
# test_batch =
# learning_rate = # try learning rate from the interval [1e-1, 1e-4]

#use_gpu = torch.cuda.is_available()


# Create train dataset loader
# Create validation dataset loader
# Create test dataset loader
# initialize your network
model = ConvNet()

# define your loss function

optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate, weight_decay=5e-04) # you can play with

# start training
# for each epoch calculate validation performance
# save best model according to validation performance

# for epoch in range(max_epoch):
#    model=model.train()
#    iterate over training batches
#    ...

#    Validation
#    model = model.eval()
#    with torch.no_grad():
#     iterate over validation batches
#    if ???????:
#       torch.save(model, best_path)

# plot losses vs epoch
# ...
# plt.show()

# plot accuracies vs epoch
# ...
# plt.show()
```

Test with SGD [10 pts.]

Report the following for your best model on your test set which has not been seen by the model yet.

1. *A heatmap for confusion matrix*
2. *Accuracy*
3. *Macro Precision*
4. *Macro Recall*
5. *F1 Score*

In [ ]:

```
# Test CNN
# load best model
# best_path = "/content/drive/My Drive/.../best_cnn_sgd.pth"
# model = torch.load(best_path)

# evaluate on test set
# model = model.eval()

# with torch.no_grad():
#   iterate over test batches
#   get confusion matrix
#   calculate accuracy
#   calculate precision
#   calculate recall
#   calculate F1 score

# print metrics
# print("Mean Loss:", losses, "\nMean Acc:", acc,"\nMean Macro Precision:", pre, "\nMean Macro Recall:",

# plot confusion matrix
# fig, ax = plt.subplots()
```

```
# im = ax.imshow(conf_matrix)
# We want to show all ticks...
# ax.set_xticks(np.arange(15))
# ax.set_yticks(np.arange(15))

# fig.tight_layout()
# plt.show()
```

Adam Optimizer [25 pts.]

Adam is an adaptive learning rate optimization algorithm that has been designed specifically for training deep neural networks. It was presented by Diederik Kingma from OpenAI and Jimmy Ba from the University of Toronto in their 2015 ICLR paper (poster) titled "Adam: A Method for Stochastic Optimization".

Nowadays, most of machine learning frameworks, including tensorflow, Pytorch, and Keras, choose Adam as the default optimizer. In this question, you will experiment with it and try to understand why it replaced SGD as the default optimizer.

Training with ADAM [15 pts.]

Train your model up to 300 epochs with properly processed inputs, i.e. call your "get_dataset". This time use Adam Optimizer as your optimizer. Tune your learning rate, weight decay. Save your best model as "best_cnn_adam.pth". The best model should be selected based on validation dataset. You could use any measurement and/or metric to decide on the best model for each network. However, you must explain your reasoning in your choice.

During training, you need to plot:

1. *training loss and validation loss vs. epoch*
2. *training accuracy and validation accuracy vs. epoch*

Name your axes and plots properly.

In [ ]:

```
# HINT: note that your training time should not take more than 2 hours.

# TODO:
# Pick your hyper parameters
max_epoch = 300
# train_batch =
# test_batch =
# learning_rate = # try learning rate from the interval [1e-1, 1e-4]

#use_gpu = torch.cuda.is_available()


# Create train dataset loader
# Create validation dataset loader
# Create test dataset loader
# initialize your network
model = ConvNet()

# define your loss function

optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate, weight_decay=5e-04) # you can play wit

# start training
# for each epoch calculate validation performance
# save best model according to validation performance

# for epoch in range(max_epoch):
#    model=model.train()
#    iterate over training batches
#    ...

#    Validation
#    model = model.eval()
#    with torch.no_grad():
#     iterate over validation batches
#    if ???????:
#       torch.save(model, best_path)

# plot losses vs epoch
# ...
# plt.show()

# plot accuracies vs epoch
# ...
```

```
# plt.show()
```

Test with ADAM [10 pts.]

Report the following for your best model on your test set which has not been seen by the model yet.

1. *A heatmap for confusion matrix*
2. *Accuracy*
3. *Macro Precision*
4. *Macro Recall*
5. *F1 Score*

*Then, discuss figures that you have plotted in the previous section, your test results and algorithm complexity with maximum 200 words. Compare two **optimizers**. Which one is more preferable? Why?*

In [ ]:

```
# Test CNN
# load best model
# best_path = "/content/drive/My Drive/.../best_cnn_adam.pth"
# model = torch.load(best_path)

# evaluate on test set
# model = model.eval()

# with torch.no_grad():
#   iterate over test batches
#   get confusion matrix
#   calculate accuracy
#   calculate precision
#   calculate recall
#   calculate F1 score

# print metrics
# print("Mean Loss:", losses, "\nMean Acc:", acc,"\nMean Macro Precision:", pre, "\nMean Macro Recall:",

# plot confusion matrix
# fig, ax = plt.subplots()
# im = ax.imshow(conf_matrix)
# We want to show all ticks...
# ax.set_xticks(np.arange(15))
# ax.set_yticks(np.arange(15))

# fig.tight_layout()
# plt.show()
```

Transfer Learning [25 pts.]

Instead of training CNNs from scratch, you can use pretrained models and apply them to your task. Transfer learning is a machine learning technique where you can reuse a pretrained machine learning model as a starting point for your own task. In this question, you will experiment with it and try to understand why it is used.

Training with Transfer Learning [15 pts.]

Get pretrained ResNet18 model from torchvision.models and finetune your model up to 20 epochs with properly processed inputs, i.e. call your "get_dataset". This time use transfer learning. Tune your learning rate, weight decay. Save your best model as "best_cnn_transfer.pth". The best model should be selected based on validation dataset. You could use any measurement and/or metric to decide on the best model for each network. However, you must explain your reasoning in your choice.

During training, you need to plot two figures:

1. *training loss and validation loss vs. epoch*
2. *training accuracy and validation accuracy vs. epoch*

Name your axes and plots properly.

In [ ]:

```
# HINT: note that your training time should not take more than 2 hours.

# TODO:
# Pick your hyper parameters
max_epoch = 20
# train_batch =
# test_batch =
# learning_rate = # try learning rate from the interval [1e-1, 1e-4]
```

```
#use_gpu = torch.cuda.is_available()


# Create train dataset loader
# Create validation dataset loader
# Create test dataset loader

# initialize your network
model = models.resnet18(pretrained=True)
num_features = model.fc.in_features
# model.fc = replace its output layer with a linear layer (in_features, proper number according to your c

# define your loss function

optimizer = torch.optim.SGD(model.parameters(), lr = learning_rate, weight_decay = weight_decay) # you ca

# start training
# for each epoch calculate validation performance
# save best model according to validation performance

# for epoch in range(max_epoch):
#    model=model.train()
#    iterate over training batches
#    ...

#    Validation
#    model = model.eval()
#    with torch.no_grad():
#     iterate over validation batches
#    if ???????:
#       torch.save(model, best_path)

# plot losses vs epoch
# ...
# plt.show()

# plot accuracies vs epoch
# ...
# plt.show()
```

Test for Transfer Learning [10 pts.]

Report the following for your best model on your test set which has not been seen by the model yet.

1. *A heatmap for confusion matrix*
2. *Accuracy*
3. *Macro Precision*
4. *Macro Recall*
5. *F1 Score*

Then, discuss figures that you have plotted in the previous section, your test results and algorithm complexity with maximum 200 words. Explain the advantages of using transfer learning. Is it better to reuse a pretrained model instead of training a model from scratch? Why?

In [ ]:

```
# Test CNN
# load best model
# best_path = "/content/drive/My Drive/.../best_cnn_transfer.pth"
# model = torch.load(best_path)

# evaluate on test set
# model = model.eval()

# with torch.no_grad():
#   iterate over test batches
#   get confusion matrix
#   calculate accuracy
#   calculate precision
#   calculate recall
#   calculate F1 score

# print metrics
# print("Mean Loss:", losses, "\nMean Acc:", acc,"\nMean Macro Precision:", pre, "\nMean Macro Recall:",

# plot confusion matrix
# fig, ax = plt.subplots()
# im = ax.imshow(conf_matrix)
```

```
# We want to show all ticks...
# ax.set_xticks(np.arange(15))
# ax.set_yticks(np.arange(15))

# fig.tight_layout()
# plt.show()
```

In [ ]: