EE8101

Progress Journal

Name: Wang Yujing

Contents

[EE8101 1](#_Toc66462215)

[Progress Journal 1](#_Toc66462216)

[1. Where do I get the learning resources? 3](#_Toc66462217)

[2. Convert video to numpy arrays 4](#_Toc66462218)

[i. What we have? 4](#_Toc66462219)

[ii. What we want? 4](#_Toc66462220)

[iii. Trial Realization 4](#_Toc66462221)

[iv. Difficulty that I faced 6](#_Toc66462222)

[v. Teammate’s solution 6](#_Toc66462223)

[vi. Code Realization 6](#_Toc66462224)

[3. Trial Model Building 8](#_Toc66462225)

[3.1 Loading of data 8](#_Toc66462226)

[3.2 Multi-Layer-Perception Model 9](#_Toc66462227)

[3.3 GRU model 10](#_Toc66462228)

[3.4 Things need to be added or modified 14](#_Toc66462229)

# Where do I get the learning resources?

1. Online training camp
   1. Machine Learning Training Camp by Prof Zhang Min from Tsinghua University.

I signed up this camp after I decided to take EE8101 in October 2020 to lay a solid foundation for this course, and graduated in Jan. I have learnt the basic methodologies and algorithms in Machine Learning. I have learnt Decision Trees, Linear Regression, Bayesian Learning, Instance-Based Learning, Support Vector Machine, Un-supervised Learning, Ensemble Learning, and Deep Learning including CNN, RNN, GRU, LSTM.

* 1. Natural Language Training Camp by Prof Liu Zhiyuan from Tsinghua University.

I signed up this camp after our team decided to do sign language translation to learn some basics in Natural Language processing.

1. Other online resources including videos, tutorials, APIs to guide me through coding.

* My notes are on OneNote and codes are either on my computer or on my google drive. If needed, I can provide the files or links.

# Convert video to numpy arrays

## What we have?

17875 Videos named XXXXX\_wwww.mp4 (XXXXX presents number and wwww represents the word)

Each word has about 10 videos of length around 2 sec, however the number of frames are different, and the size of videos varies.

The large size of video with a lot of unnecessary details (e.g. back ground or the other parts of the body other than hands) would make the reading of data very slow thus inefficient.

## What we want?

a Numpy array of size (no. of video frames, no. of hands (2), no. of features) for each video.

We want to extract features from the video frames.

We use mediapipe to extract hand landmarks from the videos, which consists of 21 landmarks \* 3 (x, y, z) for each hand in each frame.

## Trial Realization

I tried to write some code to realize this. Use words ‘accident’ and ‘abdomen’ as example.

!pip install mediapipe

import mediapipe as mp

lst = os.listdir('/content/drive/MyDrive/Colab Notebooks/EE8101/videos/accident')

path = '/content/drive/MyDrive/Colab Notebooks/EE8101/videos/accident/'

array\_path = '/content/drive/MyDrive/Colab Notebooks/EE8101/arrays/accident/'

hands = mp\_hands.Hands(

    static\_image\_mode=True,

    max\_num\_hands=2,

    min\_detection\_confidence=0.5)

for a in lst:

  ##count the number of frames

  # vid = cv2.VideoCapture(path + a)

  # query = True

  # num = 0

  # while query:

  #   ret, frame = vid.read()

  #   if ret:

  #     num += 1

  #   else:

  #     query = False

  # print(num)

  v\_list = []

  vid = cv2.VideoCapture(path + a)

  query = True

  while query:

    ret, frame = vid.read()

    if ret:

      results = hands.process(cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB))

      # print('Handedness:', results.multi\_handedness)

      res = results.multi\_hand\_landmarks

      if res:

        fram\_m = []

        for l in res[0].landmark:

          fram\_m.append([l.x, l.y, l.z])

          # print(hand\_lm)

      v\_list.append(fram\_m)

    else:

      query = False

  v\_array = np.array(v\_list)

  save\_path = array\_path + a.split('.')[0] + '.npy'

  print(v\_array.shape)

  np.save(save\_path, v\_array)

## Difficulty that I faced

Some video frames have one hand detected while some have two or none.

In my codes, I ignore the video frames with no hands detected, and only extract the feature at axis 0 (only one of the hands.)

Problem: If only one hand is detected, how to know whether it is left hand or right hand? And how to differentiate when two hands are detected?

(difficulty with the code, namely results.multi\_handedness,

While is in the structure of

Handedness: [classification {

index: 0

score: 0.927302896976471

label: "Left"

}

, classification {

index: 1

score: 0.9994159936904907

label: "Right"

}

]

how to get the index and label from this?

## Teammate’s solution

Since each sign gesture has the same meaning when it is mirrored, thus right or left hand does not matter. Just extract landmarks for two hands.

Case1: no hand is detected: put np.nan

Case2: one hand is detected: put extracted landmarks into the array at axis 0; put np.nan at axis 1

Case3: two hands are detected: put extracted landmarks into the array.

## Code Realization

def hand\_tracking\_to\_np\_array(vid\_name, frame\_num):

    with mp\_hands.Hands(

        static\_image\_mode=True,

        max\_num\_hands=2,

        min\_detection\_confidence=0.5) as hands:

            total = np.zeros((frame\_num, 2, 21, 3), dtype = np.float\_)

            vid = cv2.VideoCapture("/content/drive/MyDrive/Videos/" + vid\_name)

            query = True

            frame\_idx = 0

            while query:

                ret, frame = vid.read()

                if ret:

                    results = hands.process(cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB))

                    res = results.multi\_hand\_landmarks

                    so = 0

                    if res == None:

                        total[frame\_idx, :, :, :] = np.nan

                    elif len(res) == 1:

                        lm = 0

                        hand\_lm = res[0].landmark

                        for lmark in hand\_lm:

                            total[frame\_idx, 0, lm, :] = np.array([lmark.x, lmark.y, lmark.z])

                            lm += 1

                        total[frame\_idx, 1, :, :] = np.nan

                    else:

                        lm = 0

                        hand\_lm = res[0].landmark

                        for lmark in hand\_lm:

                            total[frame\_idx, 0, lm, :] = np.array([lmark.x, lmark.y, lmark.z])

                            lm += 1

                        lm = 0

                        hand\_lm = res[1].landmark

                        for lmark in hand\_lm:

                            total[frame\_idx, 1, lm, :] = np.array([lmark.x, lmark.y, lmark.z])

                            lm += 1

                    frame\_idx += 1

                else:

query = False

            np.save("/content/drive/MyDrive/EE8101 Deep Learning/Vid\_to\_np\_array/" + vid\_name, total)

# Trial Model Building

## Loading of data

In this trial, I only use the first 28 frames to train and test the models. 7 videos each for the words ‘abdomen’ and ‘accident’

lst = os.listdir('/content/drive/MyDrive/Colab Notebooks/EE8101/arrays/abdomen')

abdomen\_list = []

for a in lst:

  m = np.load("/content/drive/MyDrive/Colab Notebooks/EE8101/arrays/abdomen/" + a)

  abdomen\_list.append(m)

lst = os.listdir('/content/drive/MyDrive/Colab Notebooks/EE8101/arrays/accident')

accident\_list = []

for a in lst:

  m = np.load("/content/drive/MyDrive/Colab Notebooks/EE8101/arrays/accident/" + a)

  accident\_list.append(m)

X = []

y = []

for i in range(7):

  X.append(abdomen\_list[i][:28])

  y.append(0)

  X.append(accident\_list[i][:28])

  y.append(1)

X = np.array(X).reshape(14, 28, -1)

y = np.array(y)

X.shape

#Output: (14, 28, 63)

X has the shape of (sample\_size, seq\_length, no\_of\_features)

y has the shape of (sample\_size), y is the target which are the label of each sample. In this example, 0 stands for the word ‘abdomen’ and 1 stands for the word ‘accident’.

## Multi-Layer-Perception Model

X\_mlp = X.reshape(14, -1)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_mlp, y, test\_size = 4, random\_state=223)

from sklearn.neural\_network import MLPClassifier

clf = MLPClassifier(random\_state=1, max\_iter=300)

clf.fit(X\_train, y\_train)

clf.predict\_proba(X\_train), clf.predict\_proba(X\_test)

from sklearn.metrics import roc\_auc\_score

roc\_auc\_score(y\_train, clf.predict\_proba(X\_train)[:,1])

#output: 1.0

roc\_auc\_score(y\_test, clf.predict\_proba(X\_test)[:,1])

#output: 0.0

y\_test

#output: array([1, 0, 0, 0])

np.argmax(clf.predict\_proba(X\_test), axis = 1)

#output: array([1, 0, 1, 0])

Accuracy: 75%

The performance is very unstable. Sometimes, the accuracy and auc score is quite high, while sometimes it is very low. This is because MLP is not good at solve problems involving sequence. Thus the information inside the sequence is not effectively used here. Another cause may be the small sample size.

## GRU model

Similar to LSTM model, GRU model is able solve the problem of vanishing gradient in long sequences in RNN model. Compared to LSTM model, GRU model has similar performance while the training of GRU is more efficient than LSTM. Thus GRU model is chosen here.

(codes are modified from one of my previous project when learning RNN)

import math

def create\_tensor(tensor):

  if USE\_GPU:

    device = torch.device("cuda:0")

    tensor = tensor.to(device)

  return tensor

def make\_tensors(features, labels):

  seq\_lengths = torch.LongTensor(features.size(0))

  seq\_lengths = seq\_lengths.long()

  labels = labels.long()

  return create\_tensor(features), create\_tensor(seq\_lengths), create\_tensor(labels)

def time\_since(since):

  s = time.time() - since

  m = math.floor(s / 60)

  s -= m \* 60

  return '%dm %ds' % (m, s)

import torch

from torch.utils.data import DataLoader,Dataset

HIDDEN\_SIZE = 64

BATCH\_SIZE = 10

N\_LAYER = 3

N\_EPOCHS = 100

USE\_GPU = False

N\_CLASS = 2

class Trial\_Dataset(Dataset): #train and test sets

  def \_\_init\_\_(self, is\_train\_set = True):

    self.X = X\_train if is\_train\_set else X\_test

    self.X = torch.tensor(self.X).type(torch.float32)

    self.y = y\_train if is\_train\_set else y\_test

    self.y = torch.LongTensor(self.y)

    # self.one\_hot = torch.nn.functional.one\_hot(self.y, 2)

    self.len = self.X.shape[1]

  def \_\_getitem\_\_(self, index):

    return self.X[:, index, :], self.y[index]

  def \_\_len\_\_(self):

    return self.len

trainset = Trial\_Dataset(is\_train\_set = True)

trainloader = DataLoader(trainset, batch\_size = BATCH\_SIZE, shuffle = True)

testset = Trial\_Dataset(is\_train\_set = False)

testloader = DataLoader(testset, batch\_size = BATCH\_SIZE, shuffle = False)

from sklearn.metrics import roc\_auc\_score

def proba1(output):

  output0 = output[:, 0].cpu()

  output1 = output[:, 1].cpu()

  proba1 = np.exp(output1) / (np.exp(output0) + np.exp(output1))

  return proba1

class RNNClassifier(torch.nn.Module):

  def \_\_init\_\_(self, input\_size, hidden\_size, output\_size, n\_layers=1, bidirectional= True):

    super(RNNClassifier, self).\_\_init\_\_()

    self.hidden\_size = hidden\_size

    self.n\_layers = n\_layers

    self.n\_directions = 2 if bidirectional else 1

    self.gru = torch.nn.GRU(input\_size, hidden\_size, n\_layers,

                            bidirectional=bidirectional)

    self.fc = torch.nn.Linear(hidden\_size \* self.n\_directions, output\_size)

  def \_init\_hidden(self, batch\_size):

    hidden = torch.zeros(self.n\_layers \* self.n\_directions,

                         batch\_size, self.hidden\_size)

    return create\_tensor(hidden)

  def forward(self, input, seq\_lengths):

    # input shape : B x S -> S x B

    input = input.transpose(0, 1)

    batch\_size = input.shape[1]

    # print(batch\_size)

    hidden = self.\_init\_hidden(batch\_size)

    output, hidden = self.gru(input, hidden)

    if self.n\_directions == 2:

      hidden\_cat = torch.cat([hidden[-1], hidden[-2]], dim=1)

    else:

      hidden\_cat = hidden[-1]

    fc\_output = self.fc(hidden\_cat)

    return fc\_output

def trainModel():

  total\_loss = 0

  for i, (features, labels) in enumerate(trainloader, 1):

    inputs, seq\_lengths, target = make\_tensors(features, labels)

    output = classifier(inputs, seq\_lengths)

    loss = criterion(output, target)

    optimizer.zero\_grad()

    loss.backward()

    optimizer.step()

    total\_loss += loss.item()

    if i % 10 == 0:

      print(f'[{time\_since(start)}] Epoch {epoch} ', end='')

      print(f'[{i \* len(inputs)}/{len(trainset)}] ', end='')

      print(f'loss={total\_loss / (i \* len(inputs))}')

  return total\_loss

def testModel():

  correct = 0

  total = len(testset)

  print("evaluating trained model ...")

  with torch.no\_grad():

    for i, (features, labels) in enumerate(testloader, 1):

      inputs, seq\_lengths, target = make\_tensors(features, labels)

      output = classifier(inputs, seq\_lengths)

      pred = output.max(dim=1, keepdim=True)[1]

      correct += pred.eq(target.view\_as(pred)).sum().item()

  percent = '%.2f' % (100 \* correct / total)

  print(f'Dev set: Accuracy {correct}/{total} {percent}%')

  return correct / total

def testModel\_auc():

  correct = 0

  proba = []

  total = len(testset)

  print("evaluating trained model ...")

  with torch.no\_grad():

    for i, (features, labels) in enumerate(testloader, 1):

      inputs, seq\_lengths, target = make\_tensors(features, labels)

      output = classifier(inputs, seq\_lengths)

      pred = output.max(dim=1, keepdim=True)[1]

      proba\_i = proba1(output)

      proba.append(proba\_i)

      correct += pred.eq(target.view\_as(pred)).sum().item()

  percent = '%.2f' % (100 \* correct / total)

  print(f'Dev set: Accuracy {correct}/{total} {percent}%')

  probas = proba[0]

  for i in range(1, len(proba)):

    probas = torch.cat((probas, proba[i]))

  probas = np.array(probas).reshape(-1, 1)

  AUC = roc\_auc\_score(y\_test, probas)

  print("Dev set: AUC ", AUC )

  return correct / total, AUC

import time

if \_\_name\_\_ == '\_\_main\_\_':

  classifier = RNNClassifier(63, HIDDEN\_SIZE, N\_CLASS, N\_LAYER, bidirectional = True)

  if USE\_GPU:

    device = torch.device("cuda:0")

    classifier.to(device)

  criterion = torch.nn.CrossEntropyLoss()

  optimizer = torch.optim.Adam(classifier.parameters(), lr=0.0005)

  start = time.time()

  print("Training for %d epochs..." % N\_EPOCHS)

  acc\_list = []

  auc\_list = []

  for epoch in range(1, N\_EPOCHS + 1):

    # Train cycle

    print('epoch {}/{}'.format(epoch, N\_EPOCHS))

    trainModel()

    acc, auc = testModel\_auc()

    acc\_list.append(acc)

    auc\_list.append(auc)

After training about 20 epochs, the model produces an accuracy about 75% - 100% and auc = 1.0 on the dev set.

Though the training and dev set is very small, thus this may not be statistically significant much better as compared MLP. But it is a signal that GRU is capable of solving this problem.

## 3.4 Things need to be added or modified

This example is based on only two words, while we have thousands of word, thus need to modify accordingly.

1. Loading of data
   1. Load the numpy arrays: file name: XXXXX\_wwww.npy
   2. Get the label for each sample from the file name
2. Class Lang:
   1. Word2Index dictionary
   2. Index2Word dictionary
   3. Word2Count dictionary
   4. N\_words
   5. Def index\_word
3. Filtering of data
   1. May need to filtering out the nan
   2. May need to pad the data?
4. Create dataset and data loader
5. Modify the model:
   1. N\_CLASS: no. of words
   2. Loss function: cross entropy can still be used
   3. It might be too difficult to predict one word from thousands of word, may choose to predict top 3 or top 10 instead of 1. (and calculate loss according to the ranking? )
6. GridSearch on dev set to find the best parameters

Improvements and Questions:

What to do with the nan in the data?

Is it necessary to make the sequence of the same length? ( for the purpose of use GPU to speed up training?)

if yes, how to do this? Padding or sampling?

If padding, what technic of padding is recommended? (head/tail/average/or simply 0?)