BÀI TẬP THỰC HÀNH MÔN NHẬP MÔN PHÂN TÍCH DỮ LIỆU VÀ HỌC SÂU

- Bài tập được thiết kế theo từng lab, mỗi lab là 3 tiết có sự hướng dẫn của GV.
- Cuối mỗi buổi thực hành, sinh viên nộp lại phần bài tập mình đã thực hiện cho GV hướng dẫn.
- Những câu hỏi mở rộng/khó giúp sinh viên trau dồi thêm kiến thức của môn học. Sinh viên phải có trách nhiệm nghiên cứu, tìm câu trả lời nếu chưa thực hiện xong trong giờ thực hành.

Nội dung

LAB 1:

BÀI THỰC HÀNH THAO TÁC DỮ LIỆU

Nội dung: Thao tác dữ liệu điểm thi đại học của học sinh được cho bởi bảng bên dưới
Mục tiêu: Sinh viên đạt được kiến thức sau

1. Tìm hiểu nghiệp vụ dữ liệu
2. Nhập liệu bằng công cụ từ file excel
3. Xác định dữ liệu định tính và định lượng
4. Hiệu chỉnh các thang đo phù hợp và kiểu giá trị dữ liệu cho từng biến số
5. Hiệu chỉnh dữ liệu và xử lý dữ liệu thiếu
6. Chuyển đổi (transformation) dữ liệu theo khoảng cho trước
7. Tạo biến số phụ thuộc theo biến độc lập
8. Tạo biến định tính phân loại

Dữ liệu **dulieuxettuyendaihoc.csv** được mô tả như sau
Dữ liệu lưu trữ điểm trung bình môn, khu vực, khối thi và điểm thi đại học của 100 học sinh.

- T1, L1, H1, S1, V1, X1, D1, N1 lần lượt là điểm trung bình các môn Toán, Lý ,Hóa, Sinh, Văn, Sử, Địa, Ngoại ngữ năm lớp 10
- T2, L2, H2, S2, V2, X2, D2, N2 lần lượt là điểm trung bình các môn Toán, Lý ,Hóa, Sinh, Văn, Sử, Địa, Ngoại ngữ năm lớp 11
- T6, L6, H6, S6, V6, X6, D6, N6 lần lượt là điểm trung bình các môn Toán, Lý ,Hóa, Sinh, Văn, Sử, Địa, Ngoại ngữ năm lớp 12
- GT: Giới tính
- DT: Dân tộc
- KV, KT lần lượt là khu vực thi và khối thi
- DH1, DH2, DH3 lần lượt là điểm thi đại học môn 1, môn 2, môn 3

Sử dụng Pandas để thực hiện các yêu cầu sau đây

1. Xác định và phân loại dữ liệu định tính và định lượng
2. Định nghĩa các thang đo phù hợp cho từng biến số
3. Sử dụng Python để tải dữ liệu lên chương trình và in ra màn hình 10 dòng đầu tiên và 10 dòng cuối cùng
4. Thống kê dữ liệu thiếu cho cột dân tộc và hiệu chỉnh dữ liệu thiếu như sau: Mặc định thiếu thì điền giá trị 0.

Hướng dẫn

1. Lập bảng tần số, tần suất để khảo sát dữ liệu thiếu, bao nhiêu dữ liệu riêng biệt (pandas unique)
2. Thực hiện thay thế dữ liệu thiếu bằng phương pháp điền dữ liệu 0

5. Thống kê dữ liệu thiếu cho biến T1 và hiệu chỉnh dữ liệu, lưu ý việc thay thế dữ liệu thiếu sử dụng phương pháp Mean.

Hướng dẫn

1. Lập bảng tần số, tần suất để khảo sát dữ liệu thiếu
2. Thực hiện thay thế dữ liệu thiếu bằng phương pháp Mean

6. Hãy thực hiện xử lý lần lượt tất cả dữ liệu thiếu cho các biến về điểm số còn lại.
7. Tạo các biến TBM1, TBM2, TBM3 tương ứng với trung bình môn của các năm lớp 10, 11 và 12.
    - Công thức tính: TBM = (T*2 + L + H + S + V*2 + X + D + N) / 10
8. Tạo các biến xếp loại XL1, XL2 và XL3 dựa trên TBM1,TBM2 và TBM3 cho từng năm lớp 10, 11, 12 như sau:
    - Nhỏ hơn 5.0 xếp loại: yếu (kí hiệu là Y)
    - Từ 5.0 đến dưới 6.5: trung bình (kí hiệu là TB)
    - Từ 6.5 đến dưới 8.0: khá (kí hiệu là K)
    - Từ 8.0 đến dưới 9.0: giỏi (kí hiệu là G)
    - Từ 9.0 trở lên: xuất sắc (kí hiệu là XS)
9. Tạo các biến US_TBM1, US_TBM2 và US_TBM3 để chuyển điểm trung bình các năm lớp 10, 11 và 12 từ thang điểm 10 của Việt Nam sang thang điểm 4 của Mỹ. Sử dụng phương pháp Min-Max Normalization
10. Tạo biến kết quả xét tuyển (kí hiệu là KQXT) nhằm xác định sinh viên đậu (giá trị "1") và rớt (giá trị "0") vào các khối dựa trên điểm DH1, DH2 và DH3 như sau
    - Với khối A, A1 nếu [(DH1*2 + DH2 + DH3)/4] lớn hơn hoặc bằng 5.0 thì đậu, ngược lại là rớt
    - Với khối B nếu [(DH1 + DH2*2 + DH3)/4] lớn hơn hoặc bằng 5.0 thì đậu, ngược lại là rớt
    - Với khối khác nếu [(DH1+ DH2 + DH3)/3] lớn hơn hoặc bằng 5.0 thì đậu, ngược lại là rớt
11. Lưu trữ dữ liệu xuống ổ đĩa thành file **processed_dulieuxettuyendaihoc.csv**

LAB 2:

BÀI THỰC HÀNH TRÌNH BÀY DỮ LIỆU

Nội dung: Trực quan hóa dữ liệu điểm thi đã được xử lý **processed_dulieuxettuyendaihoc.csv**
Mục tiêu: Sinh viên đạt được kiến thức sau.

- Trình bày dữ liệu cơ bản
- Trực quan hóa dữ liệu cơ bản

Phần 1: Thống kê dữ liệu

1. Hãy sắp xếp dữ liệu điểm DH1 theo thứ tự tăng dần
2. Hãy sắp xếp dữ liệu điểm DH2 tăng dần theo nhóm giới tính
3. Hãy tạo pivot-table để thống kê các giá trị count, sum, mean, median, min, max, std, Q1, Q2 và Q3 của DH1 theo KT
4. Hãy tạo pivot-table để thống kê các giá trị count, sum, mean, median, min, max, std, Q1, Q2 và Q3 của DH1 theo KT và KV
5. Hãy tạo pivot-table để thống kê các giá trị count, sum, mean, median, min, max, std, Q1, Q2 và Q3 của DH1 theo KT, KV và DT

Phần 2: Trình bày dữ liệu

1. Hãy trình bày dữ liệu biến: GT

Gợi ý

- Lập bảng tần số và tần suất
- Vẽ biểu đồ tần số (cột), biểu đồ tần suất (tròn).

2. Hãy trình bày dữ liệu lần lượt các biến: **US_TBM1, US_TBM2 và US_TBM3**
3. Hãy trình bày dữ liệu biến DT với các học sinh là nam
4. Hãy trình bày dữ liệu biến KV với các học sinh là nam thuộc dân tộc Kinh, có điểm thỏa mãn điều kiện (DH1 >= 5.0 và DH2 >= 4.0 và DH3 >= 4.0)
5. Hãy trình bày dữ liệu lần lượt các biến DH1, DH2, DH3 lớn hơn bằng 5.0 và thuộc khu vực 2NT

Phần 3: Trực quan hóa dữ liệu theo nhóm phân loại

1. Trực quan dữ liệu học sinh nữ trên các nhóm XL1, XL2, XL3 dạng unstacked

Gợi ý

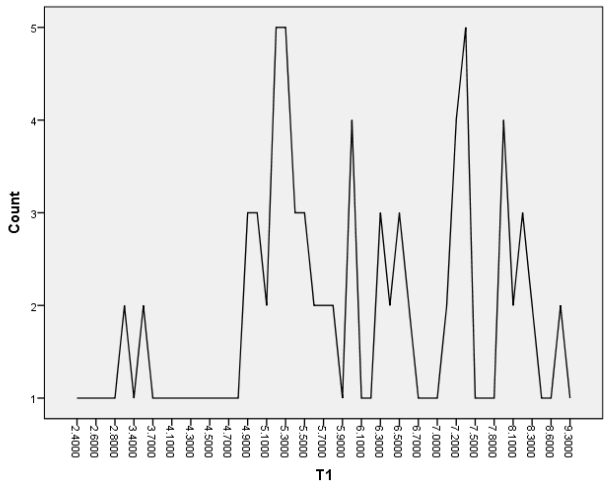
- Lọc dữ liệu giới tính là nữ
- Oy: Chiều cao biểu đồ cột thể hiển số lượng học sinh theo xếp loại
- Màu sắc thể hiện giá trị xếp loại: [Y, TB, K, G, XS]
- Ox: thể hiện nhóm XL1, XL2 và XL3

2. Trực quan dữ liệu KQXT trên nhóm học sinh có khối thi A, A1, B thuộc khu vực 1, 2
3. Trực quan dữ liệu số lượng thí sinh từng khu vực dựa trên từng nhóm khối thi
4. Trực quan dữ liệu số lượng thí sinh đậu, rớt trên từng nhóm khối thi
5. Trực quan dữ liệu số lượng thí sinh đậu rớt trên từng nhóm khu vực.
6. Trực quan dữ liệu số lượng thí sinh đậu rớt dựa trên từng nhóm dân tộc
7. Trực quan dữ liệu số lượng thí sinh đậu rớt dựa trên từng nhóm giới tính.

Phần 4: Trực quan hóa dữ liệu nâng cao

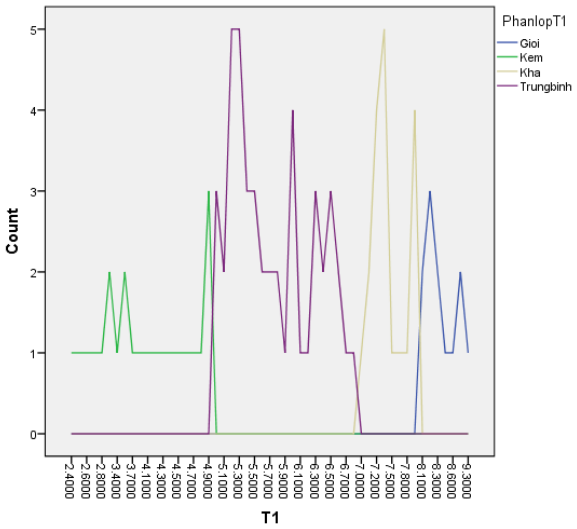
1. Vẽ biểu đồ đường Simple cho biến T1

Kết quả



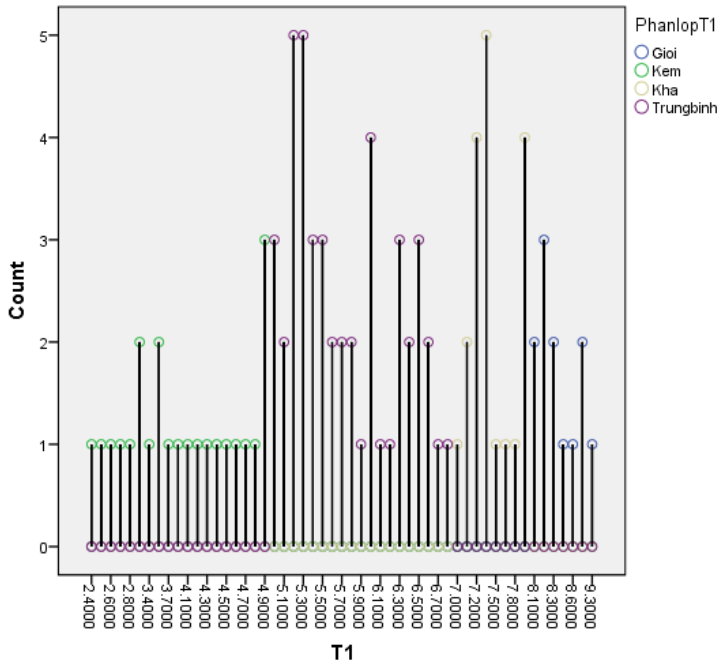
2. Hãy tạo biến phân loại (phanlopt1) cho môn toán (T1) như sau:
    a. Từ 0 đến dưới 5 = kém (ký hiệu "k")
    b. Từ 5 đến dưới 7 = trung bình (ký hiệu "tb")
    c. Từ 7 đến dưới 8 = khá (ký hiệu "k")
    d. Từ 8 trở lên = giỏi (ký hiệu "g")
3. Lập bảng tần số cho biến phanloait1
4. Vẽ biểu đồ đường Multiple Line cho biến T1 được phân loại bởi biến phanlopt1

Kết quả



5. Vẽ biểu đồ Drop-line cho biến T1 được phân loại bởi biến phanlopt1

Kết quả



Phần 5: Mô tả dữ liệu và khảo sát dạng phân phối

1. Hãy mô tả và khảo sát phân phối cho biến T1

Gợi ý

- Mô tả độ tập trung và phân tán của dữ liệu T1
- Vẽ biểu đồ Box-Plot và xác định các 10 đại lượng trong biểu đồ đó
- Mô tả hình dáng lệch của phân phối T1 dựa vào các đại lượng hướng tâm
- Vẽ biểu đồ Histogram biểu thị hình dáng phân phối
- Mô tả các đặc trưng của phân phối, mức độ lệch và mức độ nhọn
- Kiểm chứng phân phối chuẩn QQ-Plot
- Nhận xét và đánh giá về phân phối của T1

2. Hãy mô tả và khảo sát phân phối cho biến T1 trên từng nhóm phân lớp (phanlopT1)

Gợi ý

- Trực quan hóa biểu đồ Box-plot, histogram và QQ-plot theo phân nhóm là giá trị của 'phanlopT1'.

3. Hãy khảo sát tương quan giữa biến DH1 theo biến T1

Gợi ý

- Nhận xét giá trị Covariance hoặc Correlation
- Vẽ biểu đồ Scatter thể hiện liên hệ của biến phụ thuộc DH1 theo biến độc lập T1

4. Hãy khảo sát tương quan giữa biến DH1 theo biến T1 trên từng nhóm khu vực
5. Hãy khảo sát tương quan giữa các biến DH1, DH2, DH3

Gợi ý

- Nhận xét ma trận hiệp phương sai hoặc ma trận tương quan
- Vẽ biểu đồ Scatter giữa các biến

LAB 3:

LÀM SẠCH DỮ LIỆU CƠ BẢN

Nội dung: Xử lý dữ liệu y khoa về huyết áp của bệnh nhân
Mục tiêu: Sinh viên biết cách sử dụng gói Pandas để xử lý dữ liệu

1. Tiến hành hiểu dữ liệu từ chuyên gia

   "*The data set has been kept small enough for you to be able to grok it all at once. The data is in csv format. Each row in the dataset has data about different individuals and their heart rate details for different time intervals. The columns contain information such as individual's Age, Weight, Sex and Heart Rates taken at different time intervals.*"
2. Thông thường ta thường xử lý các vấn đề sau về dữ liệu
   1. Thiếu dòng tiêu đề ở file csv
   2. Nhiều biến lưu ở một cột
   3. Dữ liệu cột chứa các giá trị đơn vị không nhất quán
   4. Dữ liệu có một dòng trống
   5. Dữ liệu có các dòng trùng lặp
   6. Các ký tự không phải ASCII
   7. Giá trị bị mất
   8. Tiêu đề cột là giá trị chứ không phải tên biến
3. ***Vấn đề 1:*** Tiến hành tải dữ liệu vào chương trình ứng dụng Python và giải quyết vấn đề "Missing header in the csv file"

```
#Problem 1
# Thêm header vào dataframe để diễn giải dữ liệu
column_names= ["Id", "Name", "Age", "Weight",'m0006','m0612','m1218','f0006','f0612','f1218']
# Đọc file dữ liệu
df = pd.read_csv("patient_heart_rate.csv", names = column_names)
#Hiển thị một vài dòng dữ liệu đầu tiên ra màn hình
print(df.head())
```

4. ***Vấn đề 2:*** Xử lý vấn đề một cột lưu hỗn hợp nhiều dữ liệu, ở đây là cột "Name" chứa bao gồm "Firstname" và "Lastname", giải pháp là ta sẽ tách ra làm 2 cột

```
#Problem 2
df[['Firstname','Lastname']] = df['Name'].str.split(expand=True)
df = df.drop('Name', axis=1)
print (df)
```

5. ***Vấn đề 3:*** Cột Weight có vấn đề về không thống nhất các đơn vị đo lường trong dữ liệu. Ta sẽ chuyển các đơn vị về thành đơn vị chuẩn "kg"

```
# Problem 3
#Get the Weight column
weight = df['Weight']

for i in range (0 ,len(weight)):
    x= str(weight[i])
    #Incase lbs is part of observation remove it
    if "lbs" in x[-3:]:
        #Remove the lbs from the value
        x = x[:-3:]
        #Convert string to float
        float_x = float(x)
        #Covert to kgs and store as int
        y =int(float_x/2.2)
        #Convert back to string
        y = str(y)+"kgs"
        weight[i]= y
print (df)
```

6. ***Vấn đề 4:*** Vấn đề về xuất hiện dòng dữ liệu rỗng (không có giá trị: NaN). Giải pháp có thể đưa ra là xóa bỏ

```
# Problem 4:
df.dropna(how="all", inplace=True)
print(df)
```

7. ***Vấn đề 5:*** Có nhiều dòng dữ liệu bị trùng lắp thông tin hoàn toàn[fullname, lastname, age, weight,....], giải pháp đưa ra là chỉ giữ lại một dòng dữ liệu, tuy nhiên giải pháp phải dựa trên nghiệp vụ của tập dữ liệu và quan sát của người xử lý.

```
df = df.drop_duplicates(subset=['Firstname','Lastname', 'Age', 'Weight'])
print (df)
```

8. ***Vấn đề 6:*** Xuất hiện dữ liệu bị ảnh hưởng bởi lỗi non-ASCII, không định dạng ASCII. Giải pháp: Tùy vào nghiệp vụ ta có thể: xóa dữ liệu tại đó, thay thế bằng dữ liệu khác hoặc thay bằng việc đánh dấu bằng một kí tự khác (ví dụ: ‘warning’)

```
#Problem 6:
df.Firstname.replace({r'[^\x00-\x7F]+':''}, regex=True, inplace=True)
df.Lastname.replace({r'[^\x00-\x7F]+':''}, regex=True, inplace=True)
print (df)
```

9. ***Vấn đề 7:*** “Missing values”, vấn đề này xảy ra tại các cột “Age”, “Weight” và “Heart Rate”. Thiếu dữ liệu (dữ liệu không đầy đủ) là vấn đề xảy ra nhiều trong các nguồn dữ liệu do nhiều nguyên nhân chủ quan lẫn khách quan.Có một vài giải pháp để xử lý vấn đề này, chủ yếu dựa trên kinh nghiệm và nghiệp vụ về tập dữ liệu đó. Một số giải pháp đưa đề xuất từ chuyên gia như sau:
    a. **Deletion**: Remove records with missing values
    b. **Dummy substitution**: Replace missing values with a dummy but valid value: e.g.: 0 for numerical values.
    c. **Mean substitution**: Replace the missing values with the mean.
    d. **Frequent substitution**: Replace the missing values with the most frequent item.

e. **Improve the data collector**: Your business folk will talk to the clients and inform them about why it is worth fixing the problem with the data collector.

……

Yêu cầu:

- Thống kê thông tin dữ liệu thiếu trên từng biến Age và Weight
- Yêu cầu xử lý dữ liệu thiếu như sau: Nếu dòng nào có Age hoặc Weight có dữ liệu thì phần Age hoặc Weight được tính như bên dưới, nếu thiếu cả 2 thông tin thì xóa dòng
    - *Age*: Giá trị thay thế là mean của các giá trị trong cột Age

10. ***Vấn đề 8:*** "một cột chứa quá nhiều thông tin cần được phân rã", như trong bài toán này ta thấy header "m0006" chứa các nội dung bao gồm: m → male, 1218 ~ 12-18 (mm-dd). Còn giá trị thì là kết quả huyết áp.

3	4.0	NaN	78kgs	78	79	72	-	-	-	Scrooge	McDuck
4	5.0	54.0	90kgs	-	-	-	69	NaN	75	Pink	Panther
5	6.0	52.0	85kgs	-	-	-	68	75	72	Huey	McDuck
6	7.0	19.0	56kgs	-	-	-	71	78	75	Dewey	McDuck
7	8.0	32.0	78kgs	78	76	75	-	-	-	Scööpy	Doo

Chúng ta sẽ tách nội dung của cột này ra làm 3 cột sau: PulseRate : giá trị huyết áp, Sex: giới tính (m: male, f: female) và time: thời gian (tháng-ngày) như sau:

	Id	Age	Weight	Firstname	Lastname	PulseRate	Sex	Time
0	1.0	56.0	70kgs	Micky	Mous	72	m	00-06
9	1.0	56.0	70kgs	Micky	Mous	69	m	06-12
18	1.0	56.0	70kgs	Micky	Mous	71	m	12-18
27	1.0	56.0	70kgs	Micky	Mous	-	f	00-06
36	1.0	56.0	70kgs	Micky	Mous	-	f	06-12
45	1.0	56.0	70kgs	Micky	Mous	-	f	12-18

Gợi ý:

```
#Melt the Sex + time range columns in single column
df = pd.melt(df,id_vars=['Id','Age','Weight','Firstname','Lastname'], value_name="PulseRate",var_name="sex_and_time").sort_values(['Id','Age','Weight','Firstname','Lastname'])

# Extract Sex, Hour Lower bound and Hour upper bound group
tmp_df = df["sex_and_time"].str.extract("(\D)(\d+)(\d{2})",expand=True)

# Name columns
tmp_df.columns = ["Sex", "hours_lower", "hours_upper"]

# Create Time column based on "hours_lower" and "hours_upper" columns
tmp_df["Time"] = tmp_df["hours_lower"] + "-" + tmp_df["hours_upper"]

# Merge
df = pd.concat([df, tmp_df], axis=1)

# Drop unnecessary columns and rows
df = df.drop(['sex_and_time','hours_lower','hours_upper'], axis=1)
df = df.dropna()
df.to_csv('outputcleanup.csv',index=False)
print (df)
```

11. Hãy khảo sát tỉ lệ dữ liệu thiếu trên biến huyết áp. Dữ liệu bị thiếu thì hãy xử lý bằng phương pháp sau
    - Thay thế bằng giá trị trung bình liền trước và liền sau của người đó. Nếu không được thì dùng 2)

- Thay thế bằng giá trị trung bình 2 giá liền trước của người đó. Nếu không được thì dùng 3)
- Thay thế bằng giá trị trung bình 2 giá liền sau của người đó. Nếu không được thì dùng 4)
- Trung bình của các giá trị huyết áp của người đó. Nếu không được thì dùng 5).
- Trung bình của các giá trị huyết áp của nhóm giới tính. Nếu không được thì dùng 6)
- Trung bình của các giá trị dữ liệu. Nếu không được thì thay bằng mức ổn định trong y học.

12. Hãy rút gọn dữ liệu phù hợp và reindex lại dữ liệu. Sau đó, lưu trữ dữ liệu đã xử lý thành công với tên file ***patient_heart_rate_clean.csv***

Lưu ý: Ngoài ra còn rất nhiều vấn đề về mặt xử lý dữ liệu dựa trên nhiều khía cạnh khác nhau tùy vào sự am hiểu về dữ liệu của các chuyên gia như:

- Handling dates
- Correcting character encodings (a problem you hit when you scrape data off the web)

LAB 4:

BÀI THỰC HÀNH CHUẨN BỊ DỮ LIỆU

Nội dung: Chuẩn bị dữ liệu – Data Preparation
Tham khảo: seaborn: statistical data visualization — seaborn 0.11.1 documentation (pydata.org)
Mục tiêu: Sinh viên nắm được các kiến thức sau

1. Data Cleansing
2. Exploration Data Analysis (EDA)
3. Kỹ thuật function chain trong Pandas – pipe()
4. Feature Engineering
5. Data Wrangling

Mô tả dữ liệu: The sinking of the Titanic is one of the most infamous shipwrecks in history. On April 15, 1912, during her maiden voyage, the widely considered “unsinkable” RMS Titanic sank after colliding with an iceberg. Unfortunately, there weren’t enough lifeboats for everyone onboard, resulting in the death of 1502 out of 2224 passengers and crew. While there was some element of luck involved in surviving, it seems some groups of people were more likely to survive than others.

Variable	Definition	Key
PassengerId	Identifier	
Survived	Survival	0 = No, 1 = Yes
Pclass	Ticket class	1 = 1st, 2 = 2nd, 3 = 3rd
Name	Passenger name	
Sex	Sex	
Age	Age in years	
SibSp	# of siblings / spouses aboard the Titanic	
Parch	# of parents / children aboard the Titanic	
Ticket	Ticket number	
Fare	Passenger fare	
Cabin	Cabin number	
Embarked	Port of Embarkation	C = Cherbourg, Q = Queenstown, S = Southampton

Yêu cầu: *Hãy chuẩn bị dữ liệu phục vụ cho bài toán: "Xây dựng mô hình dự báo nhóm hành khách có khả năng sống sót với các thông số đầu vào là các đặc trưng của hành khách (name, age, gender, socio-economic class, ...), trong sự kiện Titanic lịch sử"*

PHẦN 1: DATA CLEANSING & FEATURE ENGINEERING

Hướng dẫn

1. Viết hàm load_data() để tải dữ liệu lên ứng dụng. Sau đó, hiển thị ra màn hình 10 dòng đầu tiên.
2. Thống kê dữ liệu thiếu trên các biến số và trực quan hóa dữ liệu thiếu bằng biểu đồ (Heat map). Hãy cho nhận xét về tình trạng thiếu dữ liệu Age, Cabin và Embarked
3. Xử lý tên cột tên Name, tách ra làm 2 cột: firstName và secondName. Lưu ý: Sau khi tách cột xong thì xóa luôn cột Name
4. Xử lý rút gọn kích thước dữ liệu trên cột Sex như sau: thay thế male → M và female → F
5. Xử lý dữ liệu thiếu trên biến Age bằng cách thay thế bằng giá trị trung bình tuổi: Hãy đưa ra quyết định dùng giá trị trung bình tuổi toàn bộ hành khách hay theo từng nhóm hạng vé (hạng hành khách: Pclass). Ta tiến hành làm các bước sau
    a. Sử dụng Seaborn để vẽ biểu đồ (Box plot) trực quan dữ liệu để xác định phân phối tuổi trên từng hạng hành khách. Nhận xét về tuổi trung bình giữa các nhóm hành khách. Từ đó đưa ra quyết định cách thay thế giá trị tuổi bị thiếu.
    b. Tiến hành thay thế giá trị Age bị thiếu. Sau đó, hiển thị kết quả dạng bảng và trực quan dữ liệu đã xử lý thiếu cho cột'Age' bằng biểu đồ Heat map.
6. Xây dựng biến số Agegroup có thang đo thứ tự được ánh xạ theo thang đo khoảng dựa trên độ tuổi của hành khách như sau: (age =< 12] → Kid; (12, 18]: Teen, (18, 60]: Adult và (age > 60): Older
7. Tiến hành thêm đặc trưng về danh xưng (namePrefix) trong xã hội bằng cách tách Mr, Mrs, Miss, Master ra khỏi "secondName"
8. Khai thác thêm thông tin số lượng thành viên đi theo nhóm thân quen (familySize) đối với mỗi hành khách trên chuyến hải trình; family size = 1+ SibSp + Parch

9. Tạo thêm đặc trưng 'Alone' để xác định hành khách đi theo nhóm hay cá nhân bằng cách dựa trên familySize như sau: Nếu familySize = 0 thì giá trị Alone = 1 và ngược lại là 0.
10. Tiến hành tách loại cabin (typeCabin) mà hành khách ở để lọc và phân tích đặc tính cabin. Loại cabin được kí hiệu bởi chữ cái đầu tiên. Lưu ý: Đối với dữ liệu cabin bị thiếu thì thay thế bằng "Unknown"
11. Loại bỏ dữ liệu thừa đối với các hành khách xuất hiện trong cả 2 tập dữ liệu huấn luyện (train.csv) và đánh giá (test.csv). Ưu tiên giữ lại dữ liệu trong tập huấn luyện.

PHẦN 2: KHAI THÁC THÔNG TIN HỮU ÍCH – EDA

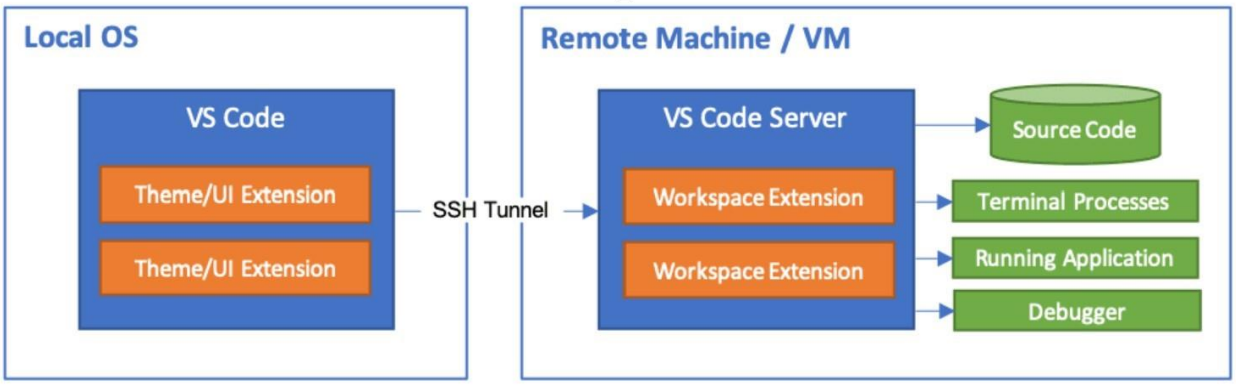
Hướng dẫn: ***Sinh viên cần đưa ra nhận xét sau mỗi biểu đồ trực quan nhằm rút trích được thông tin có giá trị về hành khách sống sót dựa trên các đặc trưng bên trên***

12. Trực quan thông tin tương quan tỉ lệ sống sót và thiệt mạng trên từng nhóm giới tính.
13. Trực quan thông tin hành khách sống sót trên từng nhóm phân loại hành khách (Pclass).
14. Trực quan thông tin hành khách sống sót trên từng nhóm giới tính và thang đo tuổi tác
15. Trực quan xác suất hành khách sống sót dựa trên thông tin nhóm đi cùng
16. Trực quan xác suất hành khách sống sót dựa trên thông tin giá vé
17. Trực quan số lượng người thiệt mạng và sống sót theo phân lớp (Pclass) hành khách và cảng sẽ cập bến.

LAB 5:

NHẬN DẠNG QUẦN ÁO GIÀY DÉP THỜI TRANG VỚI BỘ DỮ LIỆU FASHION-MNIST, DÙNG THƯ VIỆN PYTORCH HUẤN LUYỆN MÔ HÌNH TRÊN GPU CỦA FIT-LAB

I. Kết nối FIT-LAB bằng MS Visual Studio Code và nghi thức SSH

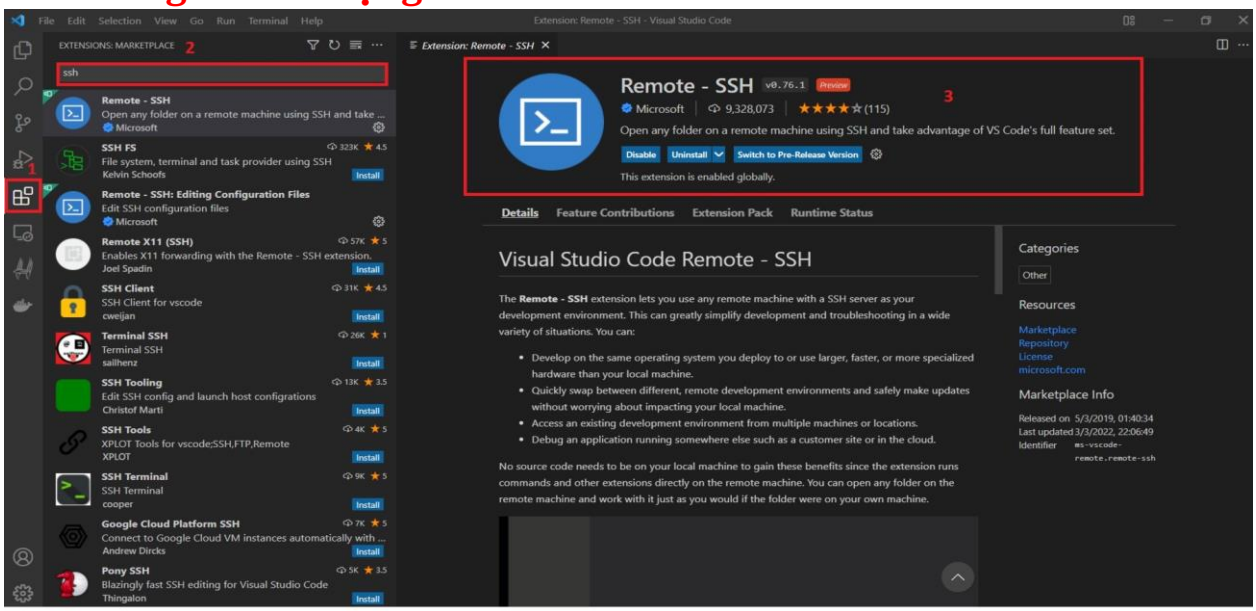


MS Visual Studio Code (**MS VSC**) là môi trường lập trình phần mềm rất phổ biến dùng để truy cập các mã nguồn (codes), chạy các đoạn mã và bắt lỗi phần mềm do Công ty Microsoft cung cấp. MS VSC có thể cài đặt trên nhiều nền tảng hệ điều hành khác nhau như Windows, Linux, MacOS. Tiện ích mở rộng **Remote - SSH** trong **MS VSC** cho phép người dùng truy cập, quản trị các hệ thống tập tin, thư mục cá nhân trong môi trường điện toán đám mây từ xa trên Microsoft Azure, Amazone Web services, Google Cloud Platform, Digital Oceans, Github, Gitlan, Docker Hub, tận dụng toàn bộ các tính năng của MS VSC như khi làm việc trên PC, Sau khi kết nối

thành công với máy chủ, người dùng có thể tương tác với các máy chủ lưu trữ dữ liệu theo mô hình điện toán đám mây (**On-Clouds**).

Từ năm học 2023-2024, sinh viên và giảng viên Khoa CNTT đã có thể sử dụng MS VSC để kết nối với môi trường **FIT-LAB** theo nghi thức Secure Shell (**SSH**), sử dụng các công nghệ ảo hóa **Docker Containers, Kubernetes,** lập trình với thư viện *NVIDIA GPU* **CUDA**, lập trình ứng dụng **AI/ML/DL/Blockchain/AToT.** Đây là các nền tảng phát triển phần mềm tiên tiến đã và đang được triển khai trong môi trường phát triển vận hành phần mềm **(DevOps)** trên **FIT-LAB**. Toàn bộ quy trình triển khai và lưu trữ dữ liệu trên FIT-LAB DevOpt đều thực hiện bên trong mạng nội bộ, tuân thủ các quy định về bảo vệ dữ liệu riêng tư của cá nhân và tổ chức **(On-Premis)**.

II. Hướng dẫn sử dụng MS VSC với Remote – SSH



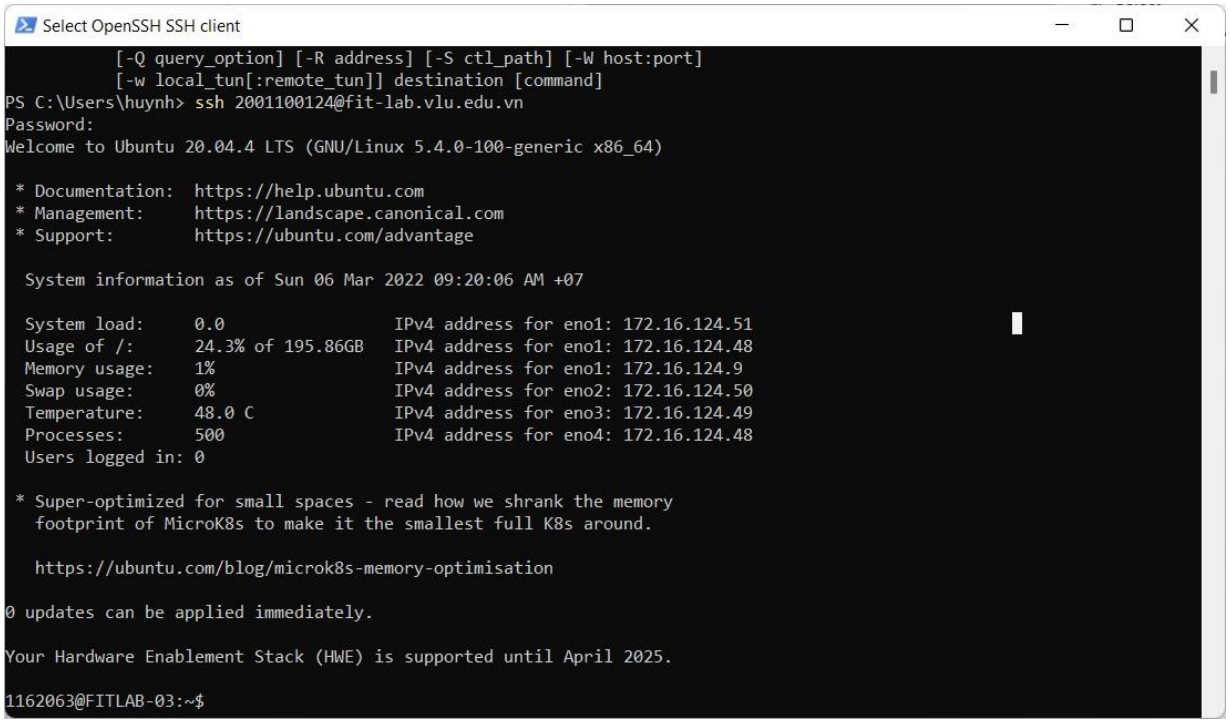
Trong giao diện MS VSC, kích chọn **Extension**, gõ SSH và chọn **Remote – SSH** chọn Install. Sau khi cài đặt thành công Remote – SSH, kết quả sẽ hiện thị như trên
Remote SSH host: Chọn máy chủ hệ điều hành **Linux. Connect to a Remote**

Host Kiểm tra kết nối bằng cách chọn **Terminal** trong MS VSC và gõ lệnh

ssh <mã người dùng>@fit-lab.vlu.edu.vn Trả lời các câu hỏi bằng tuy chọn

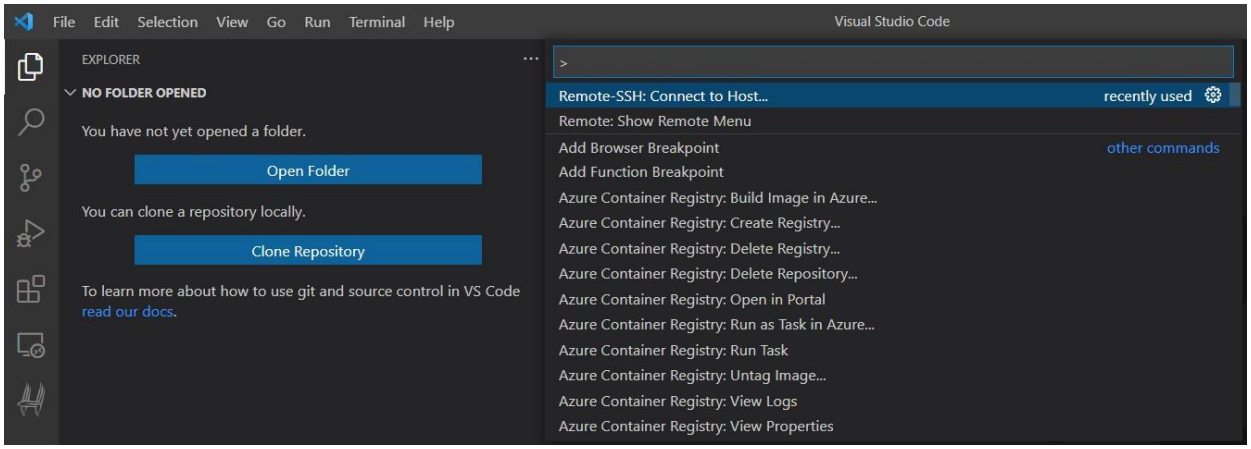
"yes" nếu có.

Lưu ý: Người dùng phải đăng nhập FIT-LAB, vào Menu Admin hay Dashboard để đổi mật khẩu (**Password**) truy cập dịch vụ Cloud Storage (trên máy chủ FIT-LAB).

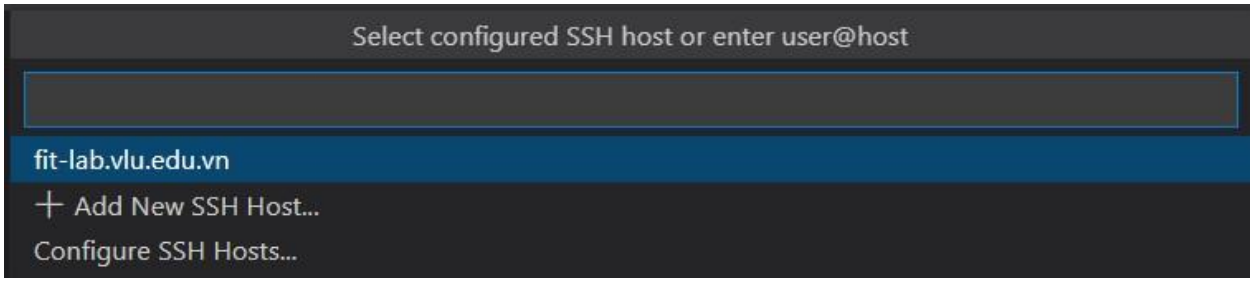


Kết nối thư mục người dùng trên FIT-LAB bằng MS VSC

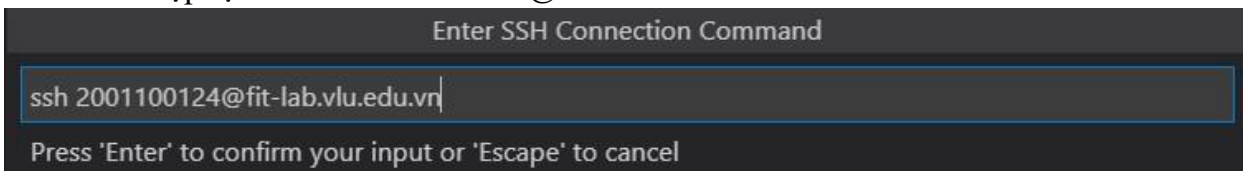
Bước 1: Trên Visual Studio Code, Chọn Remote-SSH: Connect to Host từ tổ hợp phím Ctrl+Shift+P hoặc F1, sử dụng user như ở ví dụ minh họa trên để đăng nhập.



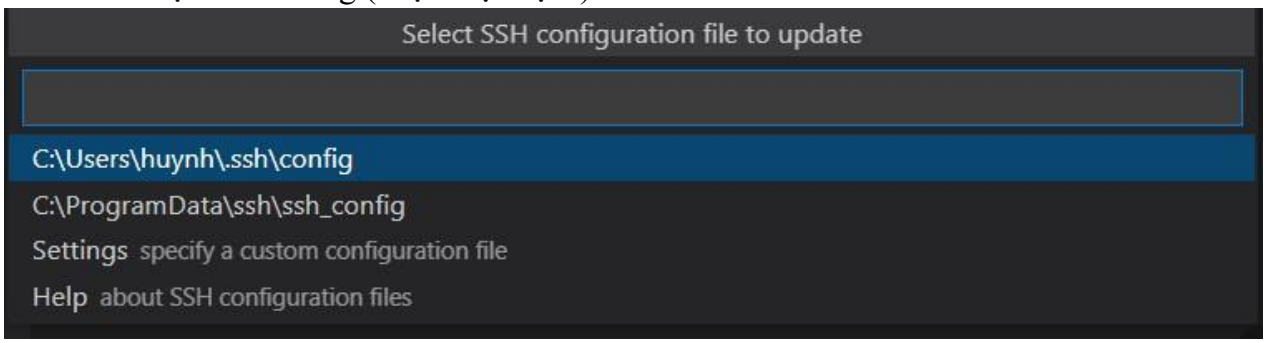
Bước 2: Chọn Add New SSH Host …



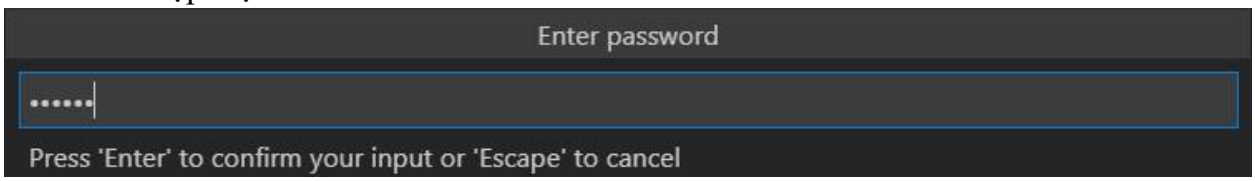
Bước 3: Nhập lệnh ssh: ssh username@host



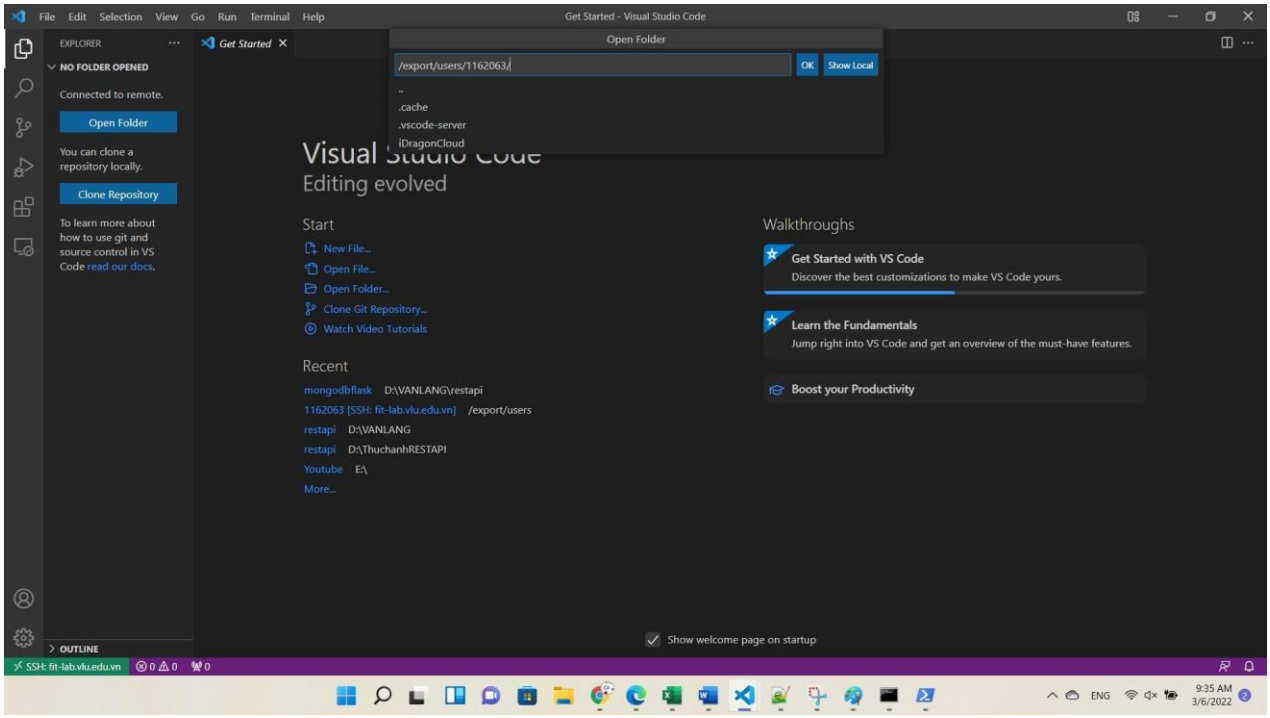
Bước 4: Chọn file config (chọn mặc định)



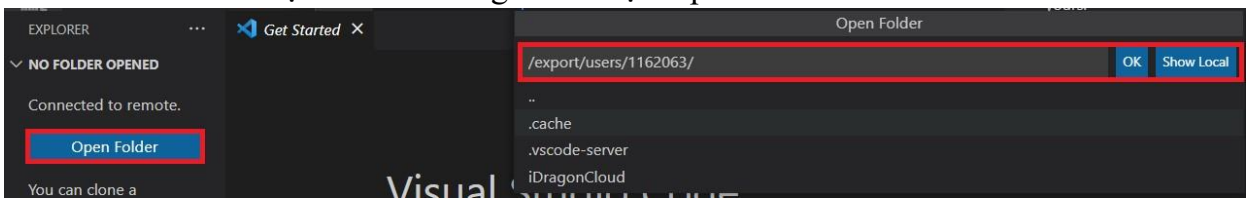
Bước 5: Nhập mật khẩu

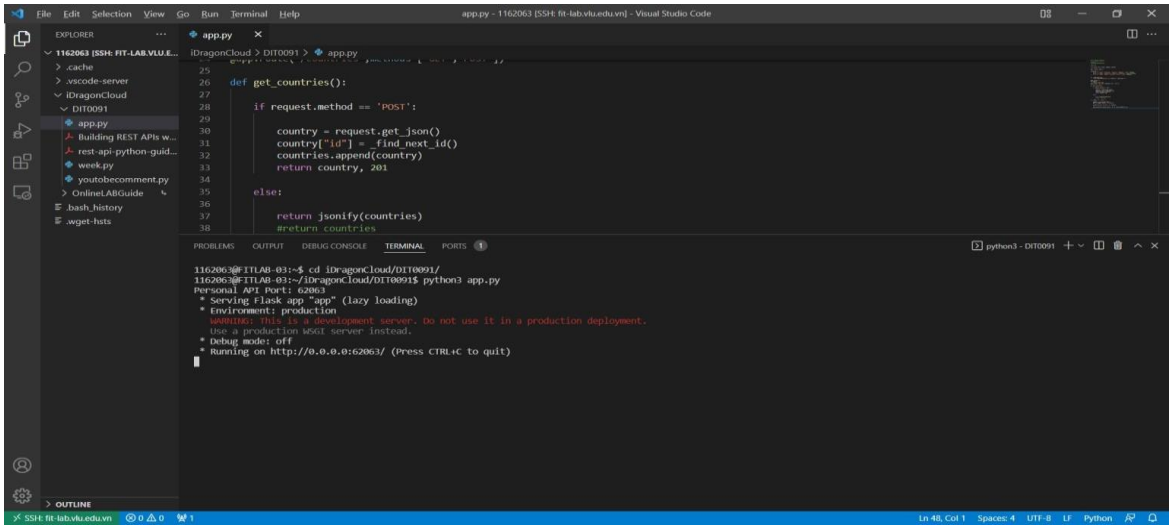


Bước 6: Việc kết nối thành công khi username và mật khẩu mà bạn sử dụng đăng nhập hợp lệ. Khi đó bạn quan sát gốc dưới bên tay trái màn hình có dòng chữ màu xanh (SSH: fitlab.vlu.edu.vn).



Bước 7: Xem thư mục cá nhân bằng cách chọn Open folder > nút OK





Bước 8 Chỉnh quyền truy cập cấu hình của người dùng

Trong một số trường hợp trong quá trình cài đặt phần mềm MS Visual Studio

Code chưa cấp đủ quyền quản lý các tập tin cấu hình nên khi người dùng đăng nhập bằng nghi thức SSH từ MS VSC gặp lỗi, cần phải điều chỉnh quyền truy cập thư mục lưu trữ cấu hình của MS VSC như sau:

1. Mở File Explorer của Hệ điều hành Windows, tìm thư mục làm việc của người dùng trên Windows (**C:/Users/ <Username>**)
2. Mở menu Properties của thư mục **C:/Users/<Username>/.ssh**
3. Chọn tab **Security**. Chọn tab **Advanced**. Chọn tab **Disable Inheritance.** Chọn **Remove All** và click **OK**
4. Chọn tab **Add**. Chọn **Add Principal**. Gõ lại Tên người dùng **<Username>**
5. Chọn tab Full Control và click OK

Làm quen với một số lệnh cơ bản của Hệ điều hành Linux

Trong môi trường MS VSC người dùng có thể mở các cửa sổ dòng lệnh **(Terminal)** để thực thi các lệnh Linux trong môi trường ngầm định **Bash Shell**. Thí dụ lệnh Linux Lệnh duyệt tập tin và thư mục, thay đổi thư mục

```
$ ls -l
$ cd
```

Lệnh kiểm tra tài khoản người dùng

```
$ id
```

Lệnh xem các tiến trình đang hoạt động

```
$  ps ux
```

Lệnh xem thông tin về hệ điều hành

```
$ uname -a
$ top
```

III. NHẬN DẠNG QUẦN ÁO GIÀY DÉP THỜI TRANG VỚI BỘ DỮ LIỆU FASHION-MNIST:

1. Chuẩn bị dữ liệu cho bài toán phân loại:

Bắt đầu bằng cách tải xuống tập dữ liệu và import các gói có liên quan.

Gói torchvision chứa nhiều bộ dữ liệu khác nhau – một trong số đó là bộ dữ liệu FashionMNIST

```
from torchvision import datasets
import torch
data_folder = '~/data/FMNIST' # This can be any directory you want
# to download FMNIST to
fmnist = datasets.FashionMNIST(data_folder, download=True, train=True)
```

Tiếp theo, chúng ta phải lưu trữ những hình ảnh có sẵn trong fmnist.data với biến tr_images và các nhãn (targets) có sẵn trong fmnist.targets với tên biến tr_targets:

```
tr_images = fmnist.data
tr_targets = fmnist.targets
```

Kiểm tra các tensor mà chúng ta đang xử lý:

```
unique_values = tr_targets.unique()
```

```
print(f'tr_images & tr_targets:\n\tX - {tr_images.shape}\n\tY -
{tr_targets.shape}\n\tY - Unique Values : {unique_values}')
print(f'TASK:\n\t{len(unique_values)} class Classification')
print(f'UNIQUE CLASSES:\n\t{fmnist.classes}')
```

Đầu ra của đoạn mã trước như sau:

tr_images & tr_targets:

X - torch.Size([60000, 28, 28])

Y - torch.Size([60000])

Y - Unique Values : tensor([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

TASK:

10 class Classification

UNIQUE CLASSES:

['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

Ở đây, chúng ta có thể thấy rằng có 60.000 hình ảnh có kích thước 28 x 28 và có 10 lớp có thể có trên tất cả các hình ảnh. Lưu ý rằng **tr_targets** chứa các giá trị số cho mỗi lớp, trong khi fmnist.classes cung cấp cho chúng ta các tên tương ứng với từng giá trị số trong **tr_targets**.

Vẽ một mẫu ngẫu nhiên gồm 10 hình ảnh cho tất cả 10 lớp có thể:

Import các gói có liên quan để vẽ đồ thị các hình ảnh và để bạn cũng có thể làm việc trên các mảng:

```
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
```

Tạo một biểu đồ trong đó chúng ta có thể hiển thị một lưới 10 x 10, trong đó mỗi hàng của lưới tương ứng với một lớp và mỗi cột trình bày một hình ảnh mẫu thuộc về lớp của hàng. Lặp lại các số lớp duy nhất (label_class) và tìm nạp chỉ mục của các hàng (label_x_rows) tương ứng với số lớp đã cho:

```
R, C = len(tr_targets.unique()), 10
fig, ax = plt.subplots(R, C, figsize=(10,10))
for label_class, plot_row in enumerate(ax):
    label_x_rows = np.where(tr_targets == label_class)[0]
    for plot_cell in plot_row:
        plot_cell.grid(False); plot_cell.axis('off')
        ix = np.random.choice(label_x_rows)
        x, y = tr_images[ix], tr_targets[ix]
        plot_cell.imshow(x, cmap='gray')
plt.tight_layout()
```

Kết quả như sau:

Lưu ý rằng trong các hình ảnh trên, mỗi hàng đại diện cho một mẫu gồm 10 hình ảnh khác nhau, tất cả đều thuộc cùng một lớp.

Bây giờ chúng ta đã học cách nhập tập dữ liệu, trong phần tiếp theo, chúng ta sẽ tìm hiểu cách huấn luyện mạng thần kinh bằng PyTorch để nó nhận một hình ảnh và dự đoán loại của hình ảnh đó. Hơn nữa, chúng ta cũng sẽ tìm hiểu về tác động của các siêu tham số khác nhau đối với độ chính xác của dự đoán.

2. Huấn luyện mạng nơ-ron

Để huấn luyện mạng nơ-ron, chúng ta phải thực hiện các bước sau:

1. Import các gói có liên quan.
2. Xây dựng tập dữ liệu có thể tìm nạp dữ liệu từng ảnh một lần.
3. Gói DataLoader từ tập dữ liệu.
4. Xây dựng mô hình, sau đó xác định hàm sai số và trình tối ưu hóa.
5. Xác định hai hàm để huấn luyện và xác thực một bó dữ liệu tương ứng.
6. Xác định hàm sẽ tính toán độ chính xác của dữ liệu.
7. Thực hiện cập nhật trọng số theo từng bó dữ liệu tăng dần qua từng epoch.

Các bước thực hiện cụ thể như sau:

1. Import các gói có liên quan và bộ dữ liệu FMNIST:

```
from torch.utils.data import Dataset, DataLoader
import torch
import torch.nn as nn
```

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
device = "cuda" if torch.cuda.is_available() else "cpu"
from torchvision import datasets
data_folder = '~/data/FMNIST' # This can be any directory you want to
# download FMNIST to
fmnist = datasets.FashionMNIST(data_folder, download=True, train=True)
tr_images = fmnist.data
tr_targets = fmnist.targets
```

2. Xây dựng một lớp tìm nạp tập dữ liệu. Chúng ta cần ba hàm __init__, __getitem__ và __len__:

```
class FMNISTDataset(Dataset):
    def __init__(self, x, y):
        x = x.float()
        x = x.view(-1,28*28)
        self.x, self.y = x, y
    def __getitem__(self, ix):
        x, y = self.x[ix], self.y[ix]
        return x.to(device), y.to(device)
    def __len__(self):
        return len(self.x)
```

3. Tạo một hàm tạo DataLoader – trn_dl từ tập dữ liệu – được gọi là FMNISTDataset. Điều này sẽ lấy mẫu 32 ảnh ngẫu nhiên cho kích thước bó:

```
def get_data():
    train = FMNISTDataset(tr_images, tr_targets)
    trn_dl = DataLoader(train, batch_size=32, shuffle=True)
    return trn_dl
```

4. Xác định một mô hình, cũng như hàm sai số và trình tối ưu hóa:

```
from torch.optim import SGD
def get_model():
    model = nn.Sequential(
        nn.Linear(28 * 28, 1000),
        nn.ReLU(),
        nn.Linear(1000, 10)
    ).to(device)
    loss_fn = nn.CrossEntropyLoss()
    optimizer = SGD(model.parameters(), lr=1e-2)
    return model, loss_fn, optimizer
```

```
tr_images = fmnist.data
tr_targets = fmnist.targets
```

5. Xác định hàm sẽ huấn luyện tập dữ liệu trên một loạt hình ảnh:

```
def train_batch(x, y, model, opt, loss_fn):
    model.train() # <- let's hold on to this until we reach dropout section
    # call your model like any python function on your batch of inputs
    prediction = model(x)
    # compute loss
    batch_loss = loss_fn(prediction, y)
    # based on the forward pass in `model(x)` compute all the gradients of
    # 'model.parameters()'
    batch_loss.backward()
    # apply new-weights = f(old-weights, old-weight-gradients) where
    # "f" is the optimizer
    optimizer.step()
    # Flush gradients memory for next batch of calculations
    optimizer.zero_grad()
    return batch_loss.item()
```

6. Xây dựng hàm tính toán độ chính xác của tập dữ liệu nhất định:

```
# since there's no need for updating weights,
# we might as well not compute the gradients.
# Using this '@' decorator on top of functions
# will disable gradient computation in the entire function
@torch.no_grad()
def accuracy(x, y, model):
    model.eval() # <- let's wait till we get to dropout section
    # get the prediction matrix for a tensor of `x` images
    prediction = model(x)
    # compute if the location of maximum in each row coincides
    # with ground truth
    max_values, argmaxes = prediction.max(-1)
    is_correct = argmaxes == y
    return is_correct.cpu().numpy().tolist()
```

7. Huấn luyện mạng nơ-ron bằng cách sử dụng các đoạn code sau:

```
trn_dl = get_data()
model, loss_fn, optimizer = get_model()
losses, accuracies = [], []
for epoch in range(5):
    print(epoch)
```

```
    epoch_losses, epoch_accuracies = [], []
    for ix, batch in enumerate(iter(trn_dl)):
        x, y = batch
        batch_loss = train_batch(x, y, model, optimizer, loss_fn)
        epoch_losses.append(batch_loss)
    epoch_loss = np.array(epoch_losses).mean()
    for ix, batch in enumerate(iter(trn_dl)):
        x, y = batch
        is_correct = accuracy(x, y, model)
        epoch_accuracies.extend(is_correct)
    epoch_accuracy = np.mean(epoch_accuracies)
    losses.append(epoch_loss)
    accuracies.append(epoch_accuracy)
epochs = np.arange(5)+1
plt.figure(figsize=(20,5))
plt.subplot(121)
plt.title('Loss value over increasing epochs')
plt.plot(epochs, losses, label='Training Loss')
plt.legend()
plt.subplot(122)
plt.title('Accuracy value over increasing epochs')
plt.plot(epochs, accuracies, label='Training Accuracy')
plt.gca().set_yticklabels(['{:.0f}%'.format(x*100) for x in
plt.gca().get_yticks()])
plt.legend()
```

Kết quả như sau:

Độ chính xác của mô hình huấn luyện là 12% vào cuối epoch. Lưu ý rằng giá trị sai số không giảm đáng kể khi số lượng epoch ngày càng tăng. Nói cách khác, cho dù chúng ta có chờ đợi bao lâu thì mô hình cũng khó có thể mang lại độ chính xác cao (chẳng hạn như trên 80%). Điều này đòi hỏi chúng ta phải hiểu các siêu tham số khác nhau được sử dụng tác động như thế nào đến độ chính xác của mạng rơ-ron của chúng ta.

Lưu ý rằng vì chúng ta không lưu lại torch.random_seed(0), kết quả có thể thay đổi khi ta thực thi đoạn code. Tuy nhiên, kết quả ta nhận được cũng sẽ đưa đến kết luận tương tự.

Bây giờ ta đã có bức tranh hoàn chỉnh về cách huấn luyện mạng rơ-ron, hãy nghiên cứu một số phương pháp hay mà chúng ta nên tuân theo để đạt được hiệu suất mô hình tốt và lý do đằng sau việc sử dụng chúng. Điều này có thể đạt được bằng cách tinh chỉnh các siêu tham số khác nhau, một số trong đó chúng ta sẽ xem xét trong các phần tiếp theo.

3. Chia tỉ lệ tập dữ liệu để cải thiện độ chính xác của mô hình

Chia tỷ lệ tập dữ liệu là quá trình đảm bảo rằng các biến được giới hạn trong một phạm vi hữu hạn. Trong phần này, chúng ta sẽ giới hạn giá trị của các biến độc lập ở các giá trị từ 0 đến 1 bằng cách chia từng giá trị đầu vào cho giá trị tối đa có thể có trong tập dữ liệu. Đây là giá trị 255, tương ứng với các pixel màu trắng.

1. Tìm nạp tập dữ liệu cũng như các hình ảnh và gán nhãn vào biến **tr_targets** như chúng ta đã làm trong phần trước:

```
from torchvision import datasets
from torch.utils.data import Dataset, DataLoader
import torch
import torch.nn as nn
device = "cuda" if torch.cuda.is_available() else "cpu"
import numpy as np
data_folder = '~/data/FMNIST' # This can be any directory you want
# to download FMNIST to
fmnist = datasets.FashionMNIST(data_folder, download=True, train=True)
tr_images = fmnist.data
tr_targets = fmnist.targets
```

2. Sửa đổi FMNISTDataset để tìm nạp dữ liệu sao cho hình ảnh đầu vào được chia cho 255 (cường độ/giá trị tối đa của pixel):

```
class FMNISTDataset(Dataset):
    def __init__(self, x, y):
        x = x.float()/255
        x = x.view(-1,28*28)
        self.x, self.y = x, y
    def __getitem__(self, ix):
        x, y = self.x[ix], self.y[ix]
        return x.to(device), y.to(device)
    def __len__(self):
        return len(self.x)
```

3. Huấn luyện một mô hình, giống như chúng ta đã làm ở các bước 4, 5, 6 và 7 của phần trước:

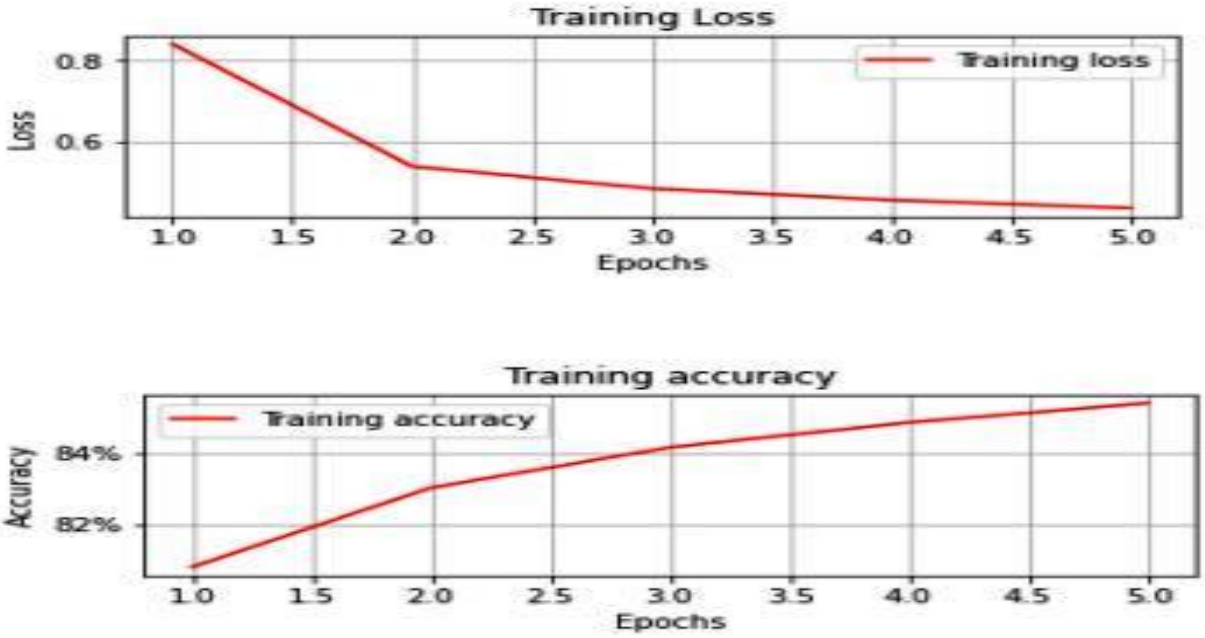
```
def get_data():
    train = FMNISTDataset(tr_images, tr_targets)
    trn_dl = DataLoader(train, batch_size=32, shuffle=True)
    return trn_dl
from torch.optim import SGD
def get_model():
    model = nn.Sequential(
        nn.Linear(28 * 28, 1000),
        nn.ReLU(),
        nn.Linear(1000, 10)
    ).to(device)
    loss_fn = nn.CrossEntropyLoss()
    optimizer = SGD(model.parameters(), lr=1e-2)
    return model, loss_fn, optimizer
def train_batch(x, y, model, opt, loss_fn):
    model.train()
    # call your model like any python function on your batch of inputs
    prediction = model(x)
    # compute loss
    batch_loss = loss_fn(prediction, y)
    # based on the forward pass in `model(x)` compute all the gradients of
    # 'model.parameters()'
    batch_loss.backward()
    # apply new-weights = f(old-weights, old-weight-gradients)
    # where "f" is the optimizer
    optimizer.step()
    # Flush memory for next batch of calculations
    optimizer.zero_grad()
    return batch_loss.item()
def accuracy(x, y, model):
    model.eval()
    # since there's no need for updating weights, we might
    # as well not compute the gradients
    with torch.no_grad():
        # get the prediction matrix for a tensor of `x` images
        prediction = model(x)
    # compute if the location of maximum in each row coincides
    # with ground truth
    max_values, argmaxes = prediction.max(-1)
    is_correct = argmaxes == y
    return is_correct.cpu().numpy().tolist()
trn_dl = get_data()
model, loss_fn, optimizer = get_model()
```

```
losses, accuracies = [], []
for epoch in range(5):
    print(epoch)
    epoch_losses, epoch_accuracies = [], []
    for ix, batch in enumerate(iter(trn_dl)):
        x, y = batch
        batch_loss = train_batch(x, y, model, optimizer, loss_fn)
        epoch_losses.append(batch_loss)
    epoch_loss = np.array(epoch_losses).mean()
    for ix, batch in enumerate(iter(trn_dl)):
        x, y = batch
        is_correct = accuracy(x, y, model)
        epoch_accuracies.extend(is_correct)
    epoch_accuracy = np.mean(epoch_accuracies)
    losses.append(epoch_loss)
    accuracies.append(epoch_accuracy)
```

4. Vẽ biểu đồ độ chính xác và sai số:

```
epochs = np.arange(5)+1
import matplotlib.pyplot as plt
%matplotlib inline
plt.figure(figsize=(20,5))
plt.subplot(121)
plt.title('Loss value over increasing epochs')
plt.plot(epochs, losses, label='Training Loss')
plt.legend()
plt.subplot(122)
plt.title('Accuracy value over increasing epochs')
plt.plot(epochs, accuracies, label='Training Accuracy')
plt.gca().set_yticklabels(['{:.0f}%'.format(x*100) for x in
plt.gca().get_yticks()])
plt.legend()
```

Kết quả như sau:



Như chúng ta có thể thấy, sai số huấn luyện giảm liên tục và độ chính xác huấn luyện tăng liên tục, do đó tăng epoch lên độ chính xác ~ 85%.

So sánh kết quả đầu ra trước đó với kịch bản trong đó dữ liệu đầu vào không được chia tỷ lệ, trong đó sai số huấn luyện không giảm một cách nhất quán và độ chính xác của tập dữ liệu huấn luyện ở cuối 5 epoch chỉ là 12%.

Chúng ta hãy đi sâu vào lý do vì sao việc chia tỉ lệ đầu vào lại hữu ích ở đây.

Hãy lấy ví dụ về cách tính giá trị sigmoid:

$$\text{Sigmoid} = 1/(1+ e^{-1(\text{Input}*\text{Weight})})$$

Trong bảng sau, chúng ta đã tính cột Sigmoid dựa trên công thức trước đó:

Input	Weight	Sigmoid
255	0.00001	0.501
255	0.0001	0.506
255	0.001	0.563
255	0.01	0.928
255	0.1	1.000
255	0.2	1.000
255	0.3	1.000
255	0.4	1.000
255	0.5	1.000
255	0.6	1.000
255	0.7	1.000
255	0.8	1.000
255	0.9	1.000
255	1	1.000

Input	Weight	Sigmoid
1	0.00001	0.500
1	0.0001	0.500
1	0.001	0.500
1	0.01	0.502
1	0.1	0.525
1	0.2	0.550
1	0.3	0.574
1	0.4	0.599
1	0.5	0.622
1	0.6	0.646
1	0.7	0.668
1	0.8	0.690
1	0.9	0.711
1	1	0.731

Trong bảng bên trái, chúng ta có thể thấy rằng khi giá trị trọng số lớn hơn 0,1, giá trị Sigmoid không thay đổi khi giá trị trọng số tăng (thay đổi).
Hơn nữa, giá trị Sigmoid chỉ thay đổi một chút khi trọng số cực kỳ nhỏ; cách duy nhất để thay đổi giá trị sigmoid là thay đổi trọng số thành một lượng rất nhỏ.
Tuy nhiên, giá trị Sigmoid thay đổi đáng kể ở bảng bên phải khi giá trị đầu vào nhỏ.
Lý do cho điều này là hàm mũ của một giá trị âm lớn (kết quả từ việc nhân giá trị trọng số với một số lớn) rất gần bằng 0, trong khi giá trị hàm mũ thay đổi khi trọng số được nhân với một đầu vào được chia tỷ lệ, như đã thấy trong bàng bên phải.
Bây giờ chúng ta đã hiểu rằng giá trị Sigmoid không thay đổi đáng kể trừ khi các giá trị trọng số rất nhỏ, bây giờ chúng ta sẽ tìm hiểu về cách các giá trị trọng số có thể bị ảnh hưởng đến một giá trị tối ưu.

4. Hiểu được tác động của việc thay đổi kích thước bó

Trong phần trước, 32 hình ảnh đã được xem xét mỗi đợt trong tập dữ liệu huấn luyện. Điều này dẫn đến số lượng cập nhật trọng số trên mỗi epoch lớn hơn vì có 1.875 cập nhật trọng số trên mỗi epoch (60.000/32 gần bằng 1.875, trong đó 60.000 là số lượng hình ảnh huấn luyện).
Hơn nữa, chúng ta không xem xét hiệu suất của mô hình trên tập dữ liệu không nhìn thấy (tập dữ liệu xác thực). Chúng ta sẽ khám phá điều này trong phần này.
Trong phần này chúng ta sẽ so sánh như sau:

- Giá trị sai số và độ chính xác của dữ liệu huấn luyện và xác thực khi kích thước bó huấn luyện là 32.
- Giá trị sai số và độ chính xác của dữ liệu huấn luyện và xác thực khi kích thước bó huấn luyện là 10.000.

Kích thước bó 32

1. Tải xuống và import các hình ảnh huấn luyện và gán nhãn vào biến **tr_targets**:

```
from torchvision import datasets
import torch
data_folder = '~/data/FMNIST' # This can be any directory you want to
# download FMNIST to
```

```
fmnist = datasets.FashionMNIST(data_folder, download=True, train=True)
tr_images = fmnist.data
tr_targets = fmnist.targets
```

2. Theo cách tương tự với hình ảnh huấn luyện, chúng ta phải tải xuống và nhập tập dữ liệu xác thực bằng cách chỉ định train = False trong khi gọi phương thức FashionMNIST trong bộ dữ liệu của chúng ta:

```
val_fmnist = datasets.FashionMNIST(data_folder, download=True, train=False)
val_images = val_fmnist.data
val_targets = val_fmnist.targets
plt.title('Accuracy value over increasing epochs')
plt.plot(epochs, accuracies, label='Training Accuracy')
plt.gca().set_yticklabels(['{:.0f}%'.format(x*100) for x in
plt.gca().get_yticks()])
plt.legend()
```

3. Import các gói có liên quan và xác định thiết bị:

```
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
from torch.utils.data import Dataset, DataLoader
import torch
import torch.nn as nn
device = 'cuda' if torch.cuda.is_available() else 'cpu'
```

4. Xác định lớp tập dữ liệu (FashionMNIST), các hàm sẽ được sử dụng để huấn luyện trên một bó dữ liệu (train_batch), tính toán độ chính xác (accuracy), sau đó xác định kiến trúc mô hình, hàm sai số và trình tối ưu hóa (get_model). Lưu ý rằng hàm lấy dữ liệu sẽ là hàm duy nhất có sai lệch so với những gì chúng ta đã thấy trong các phần trước (vì chúng ta hiện đang nghiên cứu các tập dữ liệu huấn luyện và xác thực), vì vậy chúng ta sẽ xây dựng nó trong bước tiếp theo:

```
class FMNISTDataset(Dataset):
    def __init__(self, x, y):
        x = x.float()/255
        x = x.view(-1,28*28)
        self.x, self.y = x, y
    def __getitem__(self, ix):
        x, y = self.x[ix], self.y[ix]
        return x.to(device), y.to(device)
    def __len__(self):
        return len(self.x)

from torch.optim import SGD, Adam
def get_model():
```

```
    model = nn.Sequential(
        nn.Linear(28 * 28, 1000),
        nn.ReLU(),
        nn.Linear(1000, 10)
    ).to(device)

    loss_fn = nn.CrossEntropyLoss()
    optimizer = Adam(model.parameters(), lr=1e-2)
    return model, loss_fn, optimizer

def train_batch(x, y, model, opt, loss_fn):
    model.train()
    prediction = model(x)
    batch_loss = loss_fn(prediction, y)
    batch_loss.backward()
    optimizer.step()
    optimizer.zero_grad()
    return batch_loss.item()

def accuracy(x, y, model):
    model.eval()
    # this is the same as @torch.no_grad
    # at the top of function, only difference
    # being, grad is not computed in the with scope
    with torch.no_grad():
        prediction = model(x)
    max_values, argmaxes = prediction.max(-1)
    is_correct = argmaxes == y
    return is_correct.cpu().numpy().tolist()
```

5. Xác định hàm sẽ lấy dữ liệu; hàm get_data. Hàm này sẽ trả về dữ liệu huấn luyện với kích thước bó là 32 và tập dữ liệu xác thực có kích thước bó bằng độ dài của dữ liệu xác thực (chúng ta sẽ không sử dụng dữ liệu xác thực để huấn luyện mô hình; chúng ta sẽ chỉ sử dụng dữ liệu đó để hiểu độ chính xác của mô hình khi không nhìn thấy được dữ liệu):

```
def get_data():
    train = FMNISTDataset(tr_images, tr_targets)
    trn_dl = DataLoader(train, batch_size=32, shuffle=True)
    val = FMNISTDataset(val_images, val_targets)
    val_dl = DataLoader(val, batch_size=len(val_images), shuffle=False)
    return trn_dl, val_dl
```

6. Xác định hàm tính toán việc mất dữ liệu xác thực; cái đó là, val_loss. Lưu ý rằng chúng ta đang tính toán giá trị này một cách riêng biệt vì mất dữ liệu huấn luyện đang được tính toán trong khi huấn luyện mô hình:

```
@torch.no_grad()
def val_loss(x, y, model):
    prediction = model(x)
    val_loss = loss_fn(prediction, y)
    return val_loss.item()
```

7. Tìm nạp DataLoaders huấn luyện và xác thực. Ngoài ra, hãy khởi tạo mô hình, hàm sai số và trình tối ưu hóa:

```
trn_dl, val_dl = get_data()
model, loss_fn, optimizer = get_model()
```

8. Huấn luyện mô hình như sau:

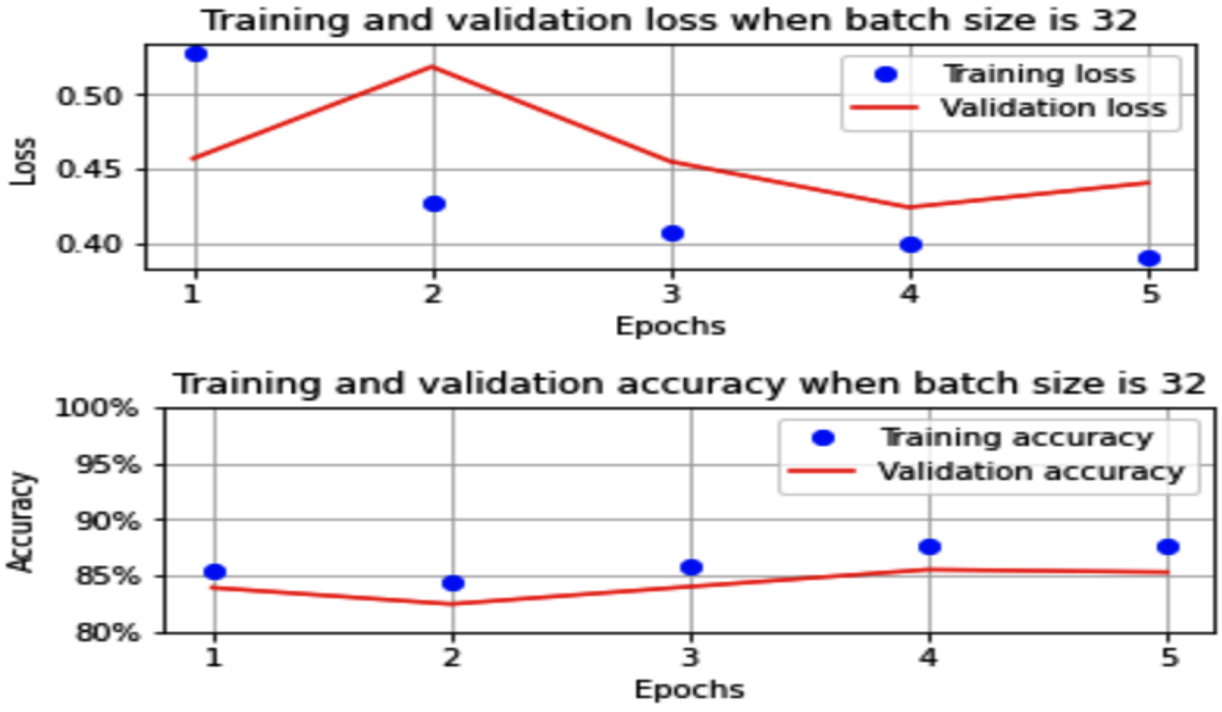
```
train_losses, train_accuracies = [], []
val_losses, val_accuracies = [], []
for epoch in range(5):
    print(epoch)
    train_epoch_losses, train_epoch_accuracies = [], []
    for ix, batch in enumerate(iter(trn_dl)):
        x, y = batch
        batch_loss = train_batch(x, y, model, optimizer, loss_fn)
        train_epoch_losses.append(batch_loss)
    train_epoch_loss = np.array(train_epoch_losses).mean()

    for ix, batch in enumerate(iter(trn_dl)):
        x, y = batch
        is_correct = accuracy(x, y, model)
        train_epoch_accuracies.extend(is_correct)
    train_epoch_accuracy = np.mean(train_epoch_accuracies)
    for ix, batch in enumerate(iter(val_dl)):
        x, y = batch
        val_is_correct = accuracy(x, y, model)
        validation_loss = val_loss(x, y, model)
    val_epoch_accuracy = np.mean(val_is_correct)
    train_losses.append(train_epoch_loss)
    train_accuracies.append(train_epoch_accuracy)
    val_losses.append(validation_loss)
    val_accuracies.append(val_epoch_accuracy)
```

9. Trực quan hóa sự cải thiện về độ chính xác và giá trị sai số trong bộ dữ liệu huấn luyện và xác thực qua các epoch ngày càng tăng:

```
epochs = np.arange(5)+1
import matplotlib.ticker as mtick
import matplotlib.pyplot as plt
import matplotlib.ticker as mticker
```

```
%matplotlib inline
plt.subplot(211)
plt.plot(epochs, train_losses, 'bo', label='Training loss')
plt.plot(epochs, val_losses, 'r', label='Validation loss')
plt.gca().xaxis.set_major_locator(mticker.MultipleLocator(1))
plt.title('Training and validation loss when batch size is 32')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.grid('off')
plt.show()
plt.subplot(212)
plt.plot(epochs, train_accuracies, 'bo', label='Training accuracy')
plt.plot(epochs, val_accuracies, 'r', label='Validation accuracy')
plt.gca().xaxis.set_major_locator(mticker.MultipleLocator(1))
plt.title('Training and validation accuracy when batch size is 32')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.gca().set_yticklabels(['{:.0f}%'.format(x*100) for x in
plt.gca().get_yticks()])
plt.legend()
plt.grid('off')
plt.show()
```

Kết quả như sau



Như bạn có thể thấy, độ chính xác trong quá trình huấn luyện và xác thực là ~85% vào cuối 5 epoch khi kích thước bó là 32. Tiếp theo, chúng ta sẽ thay đổi tham số batch_size khi huấn luyện DataLoader trong hàm get_data để xem tác động của nó đến độ chính xác kết thúc tại cuối 5 epoch.

Kích thước bó 10.000:

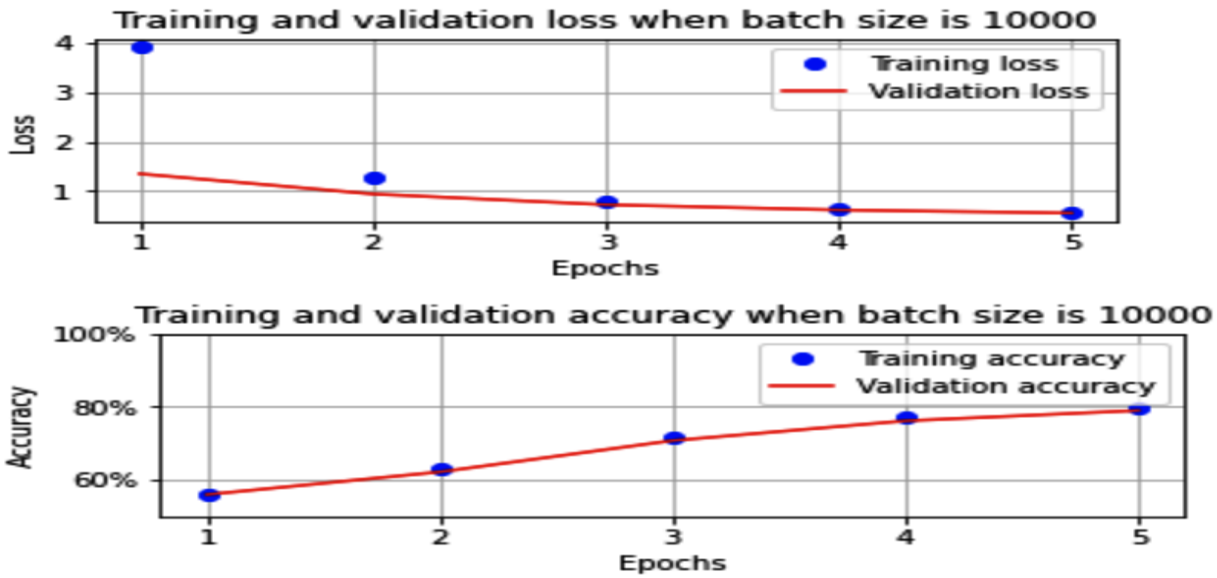
Trong phần này, chúng ta sẽ sử dụng 10.000 ảnh mỗi đợt để có thể hiểu tác động của việc thay đổi quy mô bó.

Chúng ta sẽ sửa đổi get_data để nó có kích thước bó là 10.000 trong khi tìm nạp DataLoader huấn luyện từ tập dữ liệu huấn luyện, như sau:

```
def get_data():
    train = FMNISTDataset(tr_images, tr_targets)
    trn_dl = DataLoader(train, batch_size=10000, shuffle=True)
    val = FMNISTDataset(val_images, val_targets)
    val_dl = DataLoader(val, batch_size=len(val_images), shuffle=False)
    return trn_dl, val_dl
trn_dl, val_dl = get_data()
model, loss_fn, optimizer = get_model()
train_losses, train_accuracies = [], []
val_losses, val_accuracies = [], []
for epoch in range(5):
    print(epoch)
    train_epoch_losses, train_epoch_accuracies = [], []
    for ix, batch in enumerate(iter(trn_dl)):
        x, y = batch
        batch_loss = train_batch(x, y, model, optimizer, loss_fn)
        train_epoch_losses.append(batch_loss)
    train_epoch_loss = np.array(train_epoch_losses).mean()

    for ix, batch in enumerate(iter(trn_dl)):
        x, y = batch
        is_correct = accuracy(x, y, model)
        train_epoch_accuracies.extend(is_correct)
    train_epoch_accuracy = np.mean(train_epoch_accuracies)
    for ix, batch in enumerate(iter(val_dl)):
        x, y = batch
        val_is_correct = accuracy(x, y, model)
        validation_loss = val_loss(x, y, model)
    val_epoch_accuracy = np.mean(val_is_correct)
    train_losses.append(train_epoch_loss)
    train_accuracies.append(train_epoch_accuracy)
    val_losses.append(validation_loss)
    val_accuracies.append(val_epoch_accuracy)
epochs = np.arange(5)+1
import matplotlib.ticker as mtick
import matplotlib.pyplot as plt
```

```
import matplotlib.ticker as mticker
%matplotlib inline
plt.subplot(211)
plt.plot(epochs, train_losses, 'bo', label='Training loss')
plt.plot(epochs, val_losses, 'r', label='Validation loss')
plt.gca().xaxis.set_major_locator(mticker.MultipleLocator(1))
plt.title('Training and validation loss when batch size is 10000')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.grid('off')
plt.show()
plt.subplot(212)
plt.plot(epochs, train_accuracies, 'bo', label='Training accuracy')
plt.plot(epochs, val_accuracies, 'r', label='Validation accuracy')
plt.gca().xaxis.set_major_locator(mticker.MultipleLocator(1))
plt.title('Training and validation accuracy when batch size is 10000')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.gca().set_yticklabels(['{:.0f}%'.format(x*100) for x in
plt.gca().get_yticks()])
plt.legend()
plt.grid('off')
plt.show()
```

Kết quả như sau:



Ở đây, chúng ta có thể thấy rằng các giá trị độ chính xác và sai số không đạt đến mức tương tự như kịch bản trước đó, trong đó kích thước bó là 32. Trong trường hợp kích thước bó là 10.000, có sáu lần cập nhật trọng số mỗi epoch vì có 10.000 ảnh mỗi bó, có nghĩa là tổng kích thước dữ liệu huấn luyện là 60.000.

Chúng ta đã học cách chia tỷ lệ tập dữ liệu cũng như tác động của việc thay đổi kích thước bó đến thời gian huấn luyện của mô hình để đạt được độ chính xác nhất định. Trong phần tiếp theo, chúng ta sẽ tìm hiểu về tác động của việc thay đổi trình tối ưu hóa sai số trên cùng một tập dữ liệu.

Hiểu được tác động của việc thay đổi trình tối ưu hóa sai số.

Chúng ta đã tối ưu hóa sai số dựa trên trình tối ưu hóa Adam. Trong phần này, chúng ta sẽ làm như sau:

- Sửa đổi trình tối ưu hóa để nó trở thành trình tối ưu hóa giảm dần độ dốc ngẫu nhiên (SGD)
- Kích thước bó là 32 khi tìm nạp dữ liệu trong DataLoader
- Tăng số epoch lên 10 (để chúng ta có thể so sánh hiệu suất của SGD và Adam trong một số epoch lớn hơn)

Thực hiện những thay đổi này có nghĩa là chỉ một bước trong kích thước bó là 32 sẽ thay đổi; nghĩa là chúng ta sẽ sửa đổi trình tối ưu hóa thành trình tối ưu hóa SGD.

Sửa đổi trình tối ưu hóa mà ta đang sử dụng thành trình tối ưu hóa SGD trong hàm get_model trong khi vẫn đảm bảo rằng mọi thứ khác vẫn giữ nguyên:

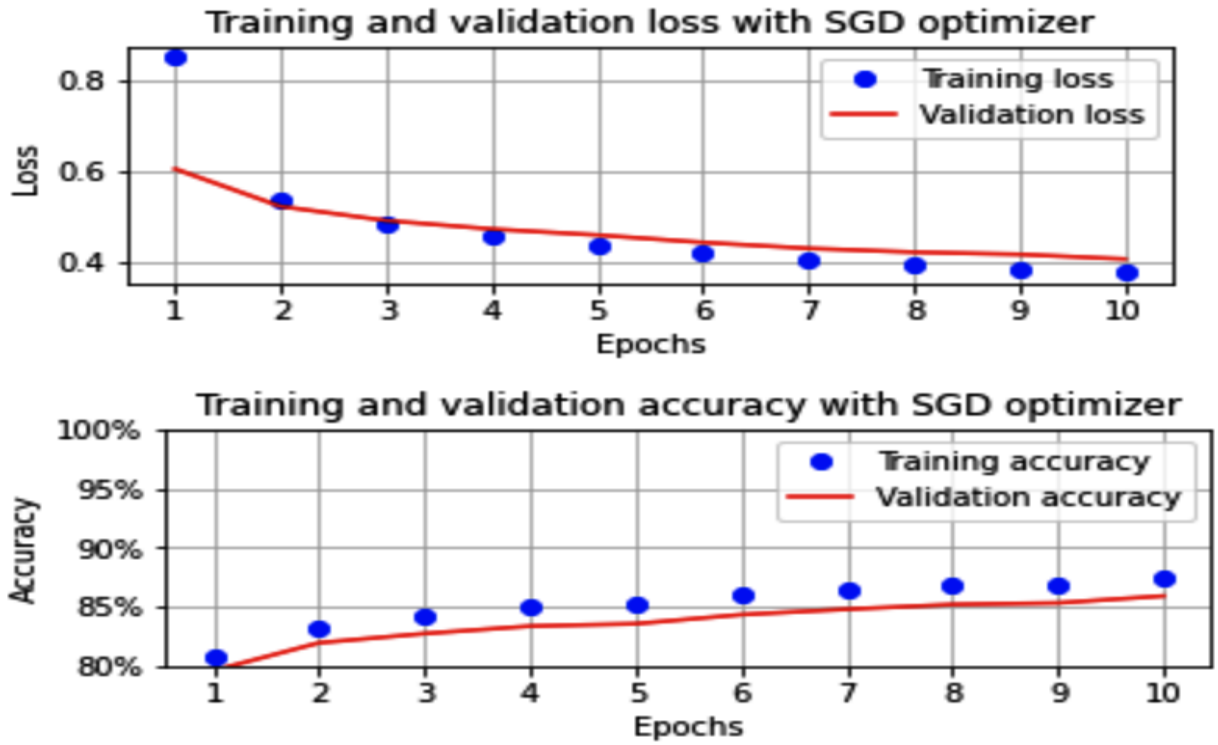
```
from torchvision import datasets
import torch
data_folder = '~/data/FMNIST' # This can be any directory you want to
# download FMNIST to
fmnist = datasets.FashionMNIST(data_folder, download=True, train=True)
tr_images = fmnist.data
tr_targets = fmnist.targets

val_fmnist = datasets.FashionMNIST(data_folder, download=True, train=False)
val_images = val_fmnist.data
val_targets = val_fmnist.targets

import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
from torch.utils.data import Dataset, DataLoader
import torch
import torch.nn as nn
device = 'cuda' if torch.cuda.is_available() else 'cpu'

# SGD optimizer
class FMNISTDataset(Dataset):
    def __init__(self, x, y):
        x = x.float()/255
        x = x.view(-1,28*28)
```

```
        self.x, self.y = x, y
    def __getitem__(self, ix):
        x, y = self.x[ix], self.y[ix]
        return x.to(device), y.to(device)
    def __len__(self):
        return len(self.x)

from torch.optim import SGD, Adam
def get_model():
    model = nn.Sequential(
        nn.Linear(28 * 28, 1000),
        nn.ReLU(),
        nn.Linear(1000, 10)
    ).to(device)

    loss_fn = nn.CrossEntropyLoss()
    optimizer = SGD(model.parameters(), lr=1e-2)
    return model, loss_fn, optimizer

def train_batch(x, y, model, opt, loss_fn):
    model.train()
    prediction = model(x)
    batch_loss = loss_fn(prediction, y)
    batch_loss.backward()
    optimizer.step()
    optimizer.zero_grad()
    return batch_loss.item()

def accuracy(x, y, model):
    model.eval()
    # this is the same as @torch.no_grad
    # at the top of function, only difference
    # being, grad is not computed in the with scope
    with torch.no_grad():
        prediction = model(x)
    max_values, argmaxes = prediction.max(-1)
    is_correct = argmaxes == y
    return is_correct.cpu().numpy().tolist()

def get_data():
    train = FMNISTDataset(tr_images, tr_targets)
    trn_dl = DataLoader(train, batch_size=32, shuffle=True)
    val = FMNISTDataset(val_images, val_targets)
    val_dl = DataLoader(val, batch_size=len(val_images), shuffle=False)
```

```
    return trn_dl, val_dl
@torch.no_grad()
def val_loss(x, y, model):
    prediction = model(x)
    val_loss = loss_fn(prediction, y)
    return val_loss.item()
trn_dl, val_dl = get_data()
model, loss_fn, optimizer = get_model()

train_losses, train_accuracies = [], []
val_losses, val_accuracies = [], []
for epoch in range(10):
    print(epoch)
    train_epoch_losses, train_epoch_accuracies = [], []
    for ix, batch in enumerate(iter(trn_dl)):
        x, y = batch
        batch_loss = train_batch(x, y, model, optimizer, loss_fn)
        train_epoch_losses.append(batch_loss)
    train_epoch_loss = np.array(train_epoch_losses).mean()

    for ix, batch in enumerate(iter(trn_dl)):
        x, y = batch
        is_correct = accuracy(x, y, model)
        train_epoch_accuracies.extend(is_correct)
    train_epoch_accuracy = np.mean(train_epoch_accuracies)
    for ix, batch in enumerate(iter(val_dl)):
        x, y = batch
        val_is_correct = accuracy(x, y, model)
        validation_loss = val_loss(x, y, model)
    val_epoch_accuracy = np.mean(val_is_correct)
    train_losses.append(train_epoch_loss)
    train_accuracies.append(train_epoch_accuracy)
    val_losses.append(validation_loss)
    val_accuracies.append(val_epoch_accuracy)

epochs = np.arange(10)+1
import matplotlib.ticker as mtick
import matplotlib.pyplot as plt
import matplotlib.ticker as mticker
%matplotlib inline
plt.subplot(211)
plt.plot(epochs, train_losses, 'bo', label='Training loss')
plt.plot(epochs, val_losses, 'r', label='Validation loss')
plt.gca().xaxis.set_major_locator(mticker.MultipleLocator(1))
```

```
plt.title('Training and validation loss with SGD optimizer')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.grid('off')
plt.show()
plt.subplot(212)
plt.plot(epochs, train_accuracies, 'bo', label='Training accuracy')
plt.plot(epochs, val_accuracies, 'r', label='Validation accuracy')
plt.gca().xaxis.set_major_locator(mticker.MultipleLocator(1))
plt.title('Training and validation accuracy with SGD optimizer')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.gca().set_yticklabels(['{:.0f}%'.format(x*100) for x in
plt.gca().get_yticks()])
plt.legend()
plt.grid('off')
plt.show()
```

Kết quả như sau:

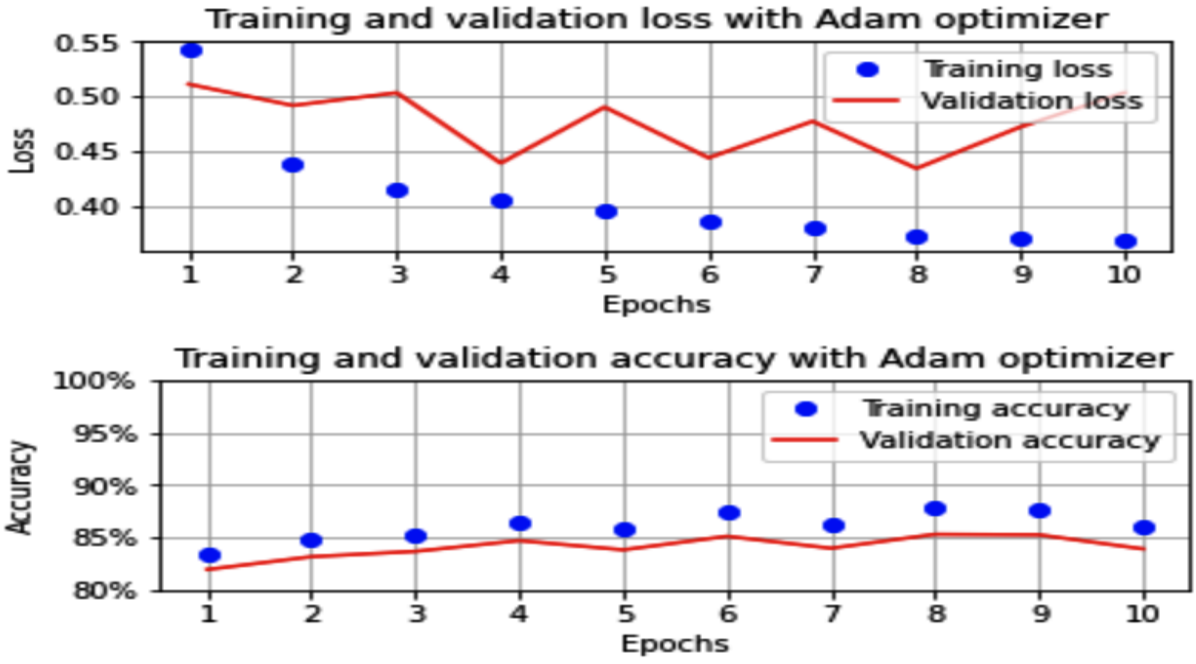


Adam optimizer

```
from torch.optim import SGD, Adam
def get_model():
    model = nn.Sequential(
        nn.Linear(28 * 28, 1000),
        nn.ReLU(),
        nn.Linear(1000, 10)
    ).to(device)

    loss_fn = nn.CrossEntropyLoss()
    optimizer = Adam(model.parameters(), lr=1e-2)
    return model, loss_fn, optimizer
trn_dl, val_dl = get_data()
model, loss_fn, optimizer = get_model()

train_losses, train_accuracies = [], []
val_losses, val_accuracies = [], []
for epoch in range(10):
    print(epoch)
    train_epoch_losses, train_epoch_accuracies = [], []
    for ix, batch in enumerate(iter(trn_dl)):
        x, y = batch
        batch_loss = train_batch(x, y, model, optimizer, loss_fn)
        train_epoch_losses.append(batch_loss)
    train_epoch_loss = np.array(train_epoch_losses).mean()

    for ix, batch in enumerate(iter(trn_dl)):
        x, y = batch
        is_correct = accuracy(x, y, model)
        train_epoch_accuracies.extend(is_correct)
    train_epoch_accuracy = np.mean(train_epoch_accuracies)
    for ix, batch in enumerate(iter(val_dl)):
        x, y = batch
        val_is_correct = accuracy(x, y, model)
        validation_loss = val_loss(x, y, model)
    val_epoch_accuracy = np.mean(val_is_correct)
    train_losses.append(train_epoch_loss)
    train_accuracies.append(train_epoch_accuracy)
    val_losses.append(validation_loss)
    val_accuracies.append(val_epoch_accuracy)
epochs = np.arange(10)+1
import matplotlib.ticker as mtick
import matplotlib.pyplot as plt
import matplotlib.ticker as mticker
```

```
%matplotlib inline
plt.subplot(211)
plt.plot(epochs, train_losses, 'bo', label='Training loss')
plt.plot(epochs, val_losses, 'r', label='Validation loss')
plt.gca().xaxis.set_major_locator(mticker.MultipleLocator(1))
plt.title('Training and validation loss with Adam optimizer')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.grid('off')
plt.show()
plt.subplot(212)
plt.plot(epochs, train_accuracies, 'bo', label='Training accuracy')
plt.plot(epochs, val_accuracies, 'r', label='Validation accuracy')
plt.gca().xaxis.set_major_locator(mticker.MultipleLocator(1))
plt.title('Training and validation accuracy with Adam optimizer')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.gca().set_yticklabels(['{:.0f}%'.format(x*100) for x in
plt.gca().get_yticks()])
plt.legend()
plt.grid('off')
plt.show()
```

Kết quả như sau:



Như chúng ta có thể thấy, khi chúng ta sử dụng trình tối ưu hóa Adam, độ chính xác vẫn rất gần với 85%. Tuy nhiên, lưu ý rằng, tốc độ học là 0,01. Trong phần tiếp theo, chúng ta sẽ tìm hiểu về tác động của tốc độ học đối với độ chính xác của tập dữ liệu xác thực.

Hiểu tác động của việc thay đổi tốc độ học.

Để hiểu tác động của tốc độ học khác nhau, chúng ta sẽ xem xét tình huống sau:

- Tốc độ học cao hơn (0,1) trên tập dữ liệu được chia tỷ lệ
- Tốc độ học thấp hơn (0,00001) trên tập dữ liệu được chia tỷ lệ
- Tốc độ học thấp hơn (0,001) trên tập dữ liệu không được chia tỷ lệ
- Tốc độ học cao hơn (0,1) trên tập dữ liệu không được chia tỷ lệ

Nhìn chung, trong phần này, chúng ta sẽ tìm hiểu về tác động của các giá trị tốc độ học khác nhau đối với các tập dữ liệu được chia tỷ lệ và không chia tỷ lệ.

Tác động của tốc độ học lên tập dữ liệu được chia tỷ lệ

Trong phần này, chúng ta sẽ so sánh độ chính xác của tập dữ liệu huấn luyện và xác thực với các tốc độ học sau:

- Tốc độ học cao
- Tốc độ học trung bình
- Tốc độ học thấp

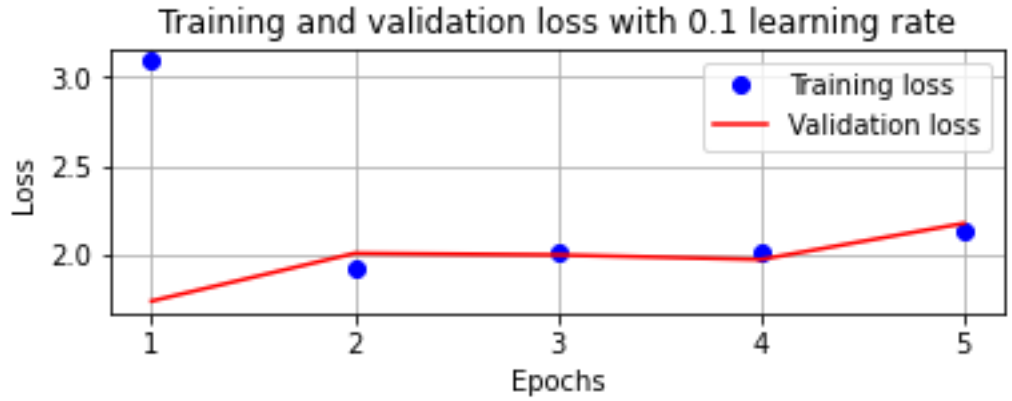
```
from torchvision import datasets
import torch
data_folder = '~/data/FMNIST' # This can be any directory you want to
# download FMNIST to
fmnist = datasets.FashionMNIST(data_folder, download=True, train=True)
tr_images = fmnist.data
tr_targets = fmnist.targets

val_fmnist = datasets.FashionMNIST(data_folder, download=True, train=False)
val_images = val_fmnist.data
val_targets = val_fmnist.targets

import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
from torch.utils.data import Dataset, DataLoader
import torch
import torch.nn as nn
device = 'cuda' if torch.cuda.is_available() else 'cpu'

# High Learning Rate
class FMNISTDataset(Dataset):
    def __init__(self, x, y):
        x = x.float()/255
        x = x.view(-1,28*28)
        self.x, self.y = x, y
```

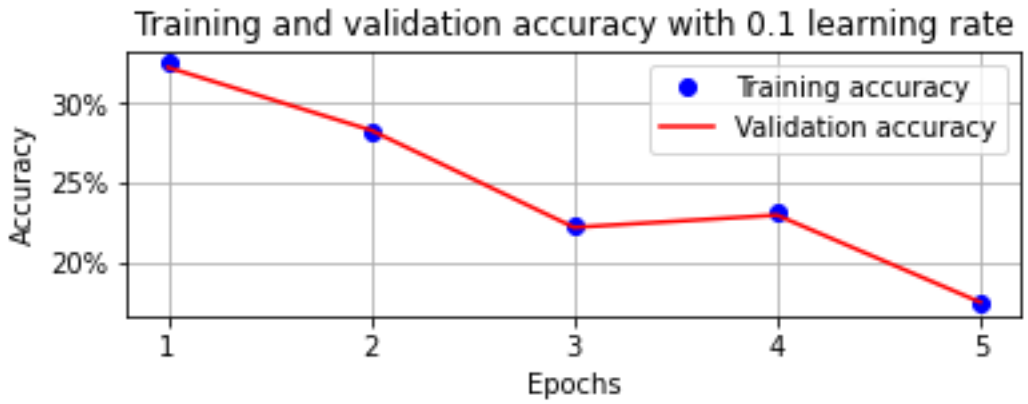
```
    def __getitem__(self, ix):
        x, y = self.x[ix], self.y[ix]
        return x.to(device), y.to(device)
    def __len__(self):
        return len(self.x)

from torch.optim import SGD, Adam
def get_model():
    model = nn.Sequential(
        nn.Linear(28 * 28, 1000),
        nn.ReLU(),
        nn.Linear(1000, 10)
    ).to(device)

    loss_fn = nn.CrossEntropyLoss()
    optimizer = Adam(model.parameters(), lr=1e-1)
    return model, loss_fn, optimizer

def train_batch(x, y, model, opt, loss_fn):
    model.train()
    prediction = model(x)
    batch_loss = loss_fn(prediction, y)
    batch_loss.backward()
    optimizer.step()
    optimizer.zero_grad()
    return batch_loss.item()

def accuracy(x, y, model):
    model.eval()
    # this is the same as @torch.no_grad
    # at the top of function, only difference
    # being, grad is not computed in the with scope
    with torch.no_grad():
        prediction = model(x)
    max_values, argmaxes = prediction.max(-1)
    is_correct = argmaxes == y
    return is_correct.cpu().numpy().tolist()

def get_data():
    train = FMNISTDataset(tr_images, tr_targets)
    trn_dl = DataLoader(train, batch_size=32, shuffle=True)
    val = FMNISTDataset(val_images, val_targets)
    val_dl = DataLoader(val, batch_size=len(val_images), shuffle=False)
    return trn_dl, val_dl
```

```
@torch.no_grad()
def val_loss(x, y, model):
    prediction = model(x)
    val_loss = loss_fn(prediction, y)
    return val_loss.item()

trn_dl, val_dl = get_data()
model, loss_fn, optimizer = get_model()

train_losses, train_accuracies = [], []
val_losses, val_accuracies = [], []
for epoch in range(5):
    print(epoch)
    train_epoch_losses, train_epoch_accuracies = [], []
    for ix, batch in enumerate(iter(trn_dl)):
        x, y = batch
        batch_loss = train_batch(x, y, model, optimizer, loss_fn)
        train_epoch_losses.append(batch_loss)
    train_epoch_loss = np.array(train_epoch_losses).mean()

    for ix, batch in enumerate(iter(trn_dl)):
        x, y = batch
        is_correct = accuracy(x, y, model)
        train_epoch_accuracies.extend(is_correct)
    train_epoch_accuracy = np.mean(train_epoch_accuracies)
    for ix, batch in enumerate(iter(val_dl)):
        x, y = batch
        val_is_correct = accuracy(x, y, model)
        validation_loss = val_loss(x, y, model)
    val_epoch_accuracy = np.mean(val_is_correct)
    train_losses.append(train_epoch_loss)
    train_accuracies.append(train_epoch_accuracy)
    val_losses.append(validation_loss)
    val_accuracies.append(val_epoch_accuracy)
epochs = np.arange(5)+1
import matplotlib.ticker as mtick
import matplotlib.pyplot as plt
import matplotlib.ticker as mticker
%matplotlib inline
plt.subplot(211)
plt.plot(epochs, train_losses, 'bo', label='Training loss')
plt.plot(epochs, val_losses, 'r', label='Validation loss')
plt.gca().xaxis.set_major_locator(mticker.MultipleLocator(1))
```

```
plt.title('Training and validation loss with 0.1 learning rate')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.grid('off')
plt.show()
plt.subplot(212)
plt.plot(epochs, train_accuracies, 'bo', label='Training accuracy')
plt.plot(epochs, val_accuracies, 'r', label='Validation accuracy')
plt.gca().xaxis.set_major_locator(mticker.MultipleLocator(1))
plt.title('Training and validation accuracy with 0.1 learning rate')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.gca().set_yticklabels(['{:.0f}%'.format(x*100) for x in
plt.gca().get_yticks()])
plt.legend()
plt.grid('off')
plt.show()
```

Kết quả như sau:



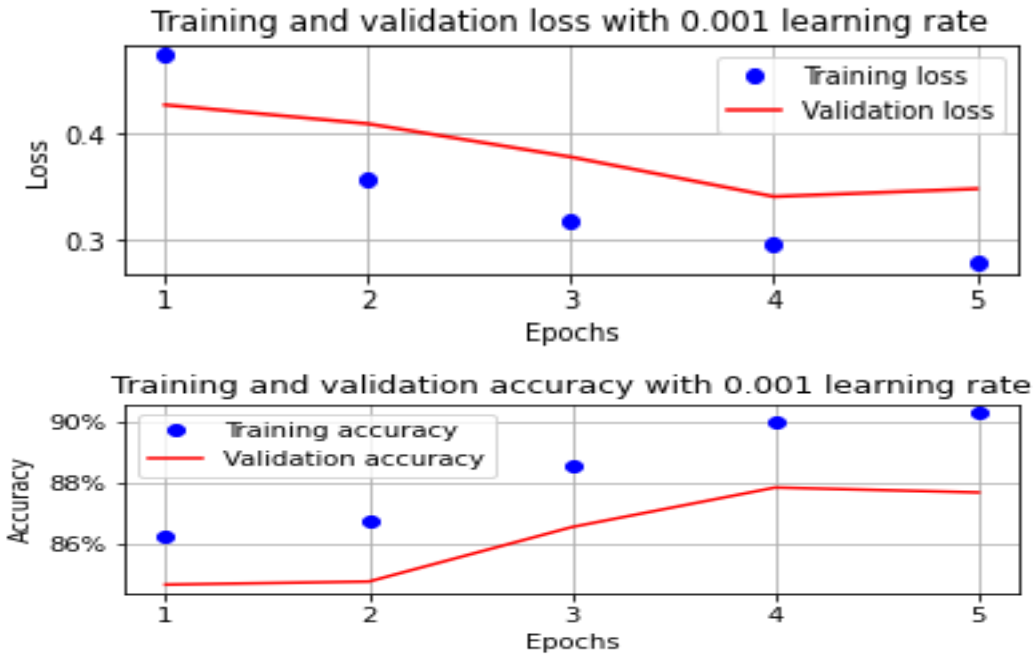


Tốc độ học trung bình

```
def get_model():
    model = nn.Sequential(
        nn.Linear(28 * 28, 1000),
        nn.ReLU(),
        nn.Linear(1000, 10)
    ).to(device)

    loss_fn = nn.CrossEntropyLoss()
    optimizer = Adam(model.parameters(), lr=1e-3)
    return model, loss_fn, optimizer
trn_dl, val_dl = get_data()
model, loss_fn, optimizer = get_model()

train_losses, train_accuracies = [], []
val_losses, val_accuracies = [], []
for epoch in range(5):
    print(epoch)
    train_epoch_losses, train_epoch_accuracies = [], []
    for ix, batch in enumerate(iter(trn_dl)):
        x, y = batch
        batch_loss = train_batch(x, y, model, optimizer, loss_fn)
        train_epoch_losses.append(batch_loss)
    train_epoch_loss = np.array(train_epoch_losses).mean()

    for ix, batch in enumerate(iter(trn_dl)):
        x, y = batch
        is_correct = accuracy(x, y, model)
        train_epoch_accuracies.extend(is_correct)
    train_epoch_accuracy = np.mean(train_epoch_accuracies)
    for ix, batch in enumerate(iter(val_dl)):
        x, y = batch
        val_is_correct = accuracy(x, y, model)
        validation_loss = val_loss(x, y, model)
    val_epoch_accuracy = np.mean(val_is_correct)
    train_losses.append(train_epoch_loss)
    train_accuracies.append(train_epoch_accuracy)
    val_losses.append(validation_loss)
    val_accuracies.append(val_epoch_accuracy)

epochs = np.arange(5)+1
import matplotlib.ticker as mtick
import matplotlib.pyplot as plt
import matplotlib.ticker as mticker
```

```
%matplotlib inline
plt.subplot(211)
plt.plot(epochs, train_losses, 'bo', label='Training loss')
plt.plot(epochs, val_losses, 'r', label='Validation loss')
plt.gca().xaxis.set_major_locator(mticker.MultipleLocator(1))
plt.title('Training and validation loss with 0.001 learning rate')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.grid('off')
plt.show()
plt.subplot(212)
plt.plot(epochs, train_accuracies, 'bo', label='Training accuracy')
plt.plot(epochs, val_accuracies, 'r', label='Validation accuracy')
plt.gca().xaxis.set_major_locator(mticker.MultipleLocator(1))
plt.title('Training and validation accuracy with 0.001 learning rate')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.gca().set_yticklabels(['{:.0f}%'.format(x*100) for x in
plt.gca().get_yticks()])
plt.legend()
plt.grid('off')
plt.show()
```

Kết quả như sau:

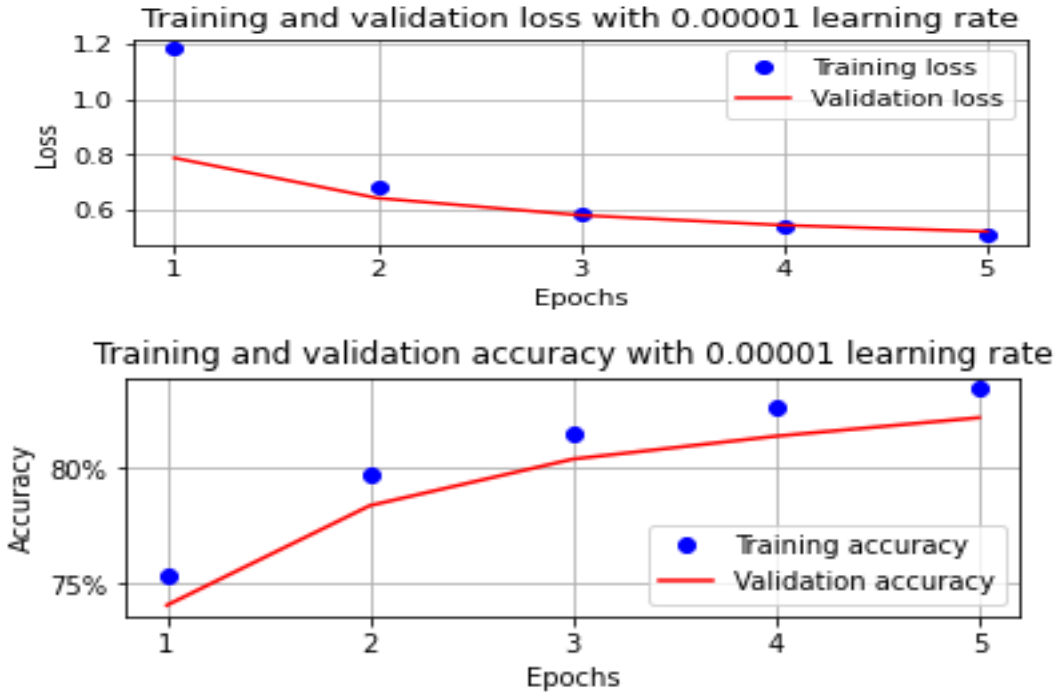


Tốc độ học thấp

```
def get_model():
    model = nn.Sequential(
        nn.Linear(28 * 28, 1000),
        nn.ReLU(),
        nn.Linear(1000, 10)
    ).to(device)

    loss_fn = nn.CrossEntropyLoss()
    optimizer = Adam(model.parameters(), lr=1e-5)
    return model, loss_fn, optimizer

trn_dl, val_dl = get_data()
model, loss_fn, optimizer = get_model()

train_losses, train_accuracies = [], []
val_losses, val_accuracies = [], []
for epoch in range(5):
    print(epoch)
    train_epoch_losses, train_epoch_accuracies = [], []
    for ix, batch in enumerate(iter(trn_dl)):
        x, y = batch
        batch_loss = train_batch(x, y, model, optimizer, loss_fn)
        train_epoch_losses.append(batch_loss)
    train_epoch_loss = np.array(train_epoch_losses).mean()

    for ix, batch in enumerate(iter(trn_dl)):
        x, y = batch
        is_correct = accuracy(x, y, model)
        train_epoch_accuracies.extend(is_correct)
    train_epoch_accuracy = np.mean(train_epoch_accuracies)
    for ix, batch in enumerate(iter(val_dl)):
        x, y = batch
        val_is_correct = accuracy(x, y, model)
        validation_loss = val_loss(x, y, model)
    val_epoch_accuracy = np.mean(val_is_correct)
    train_losses.append(train_epoch_loss)
    train_accuracies.append(train_epoch_accuracy)
    val_losses.append(validation_loss)
    val_accuracies.append(val_epoch_accuracy)

epochs = np.arange(5)+1
import matplotlib.ticker as mtick
import matplotlib.pyplot as plt
```

```
import matplotlib.ticker as mticker
%matplotlib inline
plt.subplot(211)
plt.plot(epochs, train_losses, 'bo', label='Training loss')
plt.plot(epochs, val_losses, 'r', label='Validation loss')
plt.gca().xaxis.set_major_locator(mticker.MultipleLocator(1))
plt.title('Training and validation loss with 0.00001 learning rate')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.grid('off')
plt.show()
plt.subplot(212)
plt.plot(epochs, train_accuracies, 'bo', label='Training accuracy')
plt.plot(epochs, val_accuracies, 'r', label='Validation accuracy')
plt.gca().xaxis.set_major_locator(mticker.MultipleLocator(1))
plt.title('Training and validation accuracy with 0.00001 learning rate')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.gca().set_yticklabels(['{:.0f}%'.format(x*100) for x in
plt.gca().get_yticks()])
plt.legend()
plt.grid('off')
plt.show()
```

Kết quả như sau:

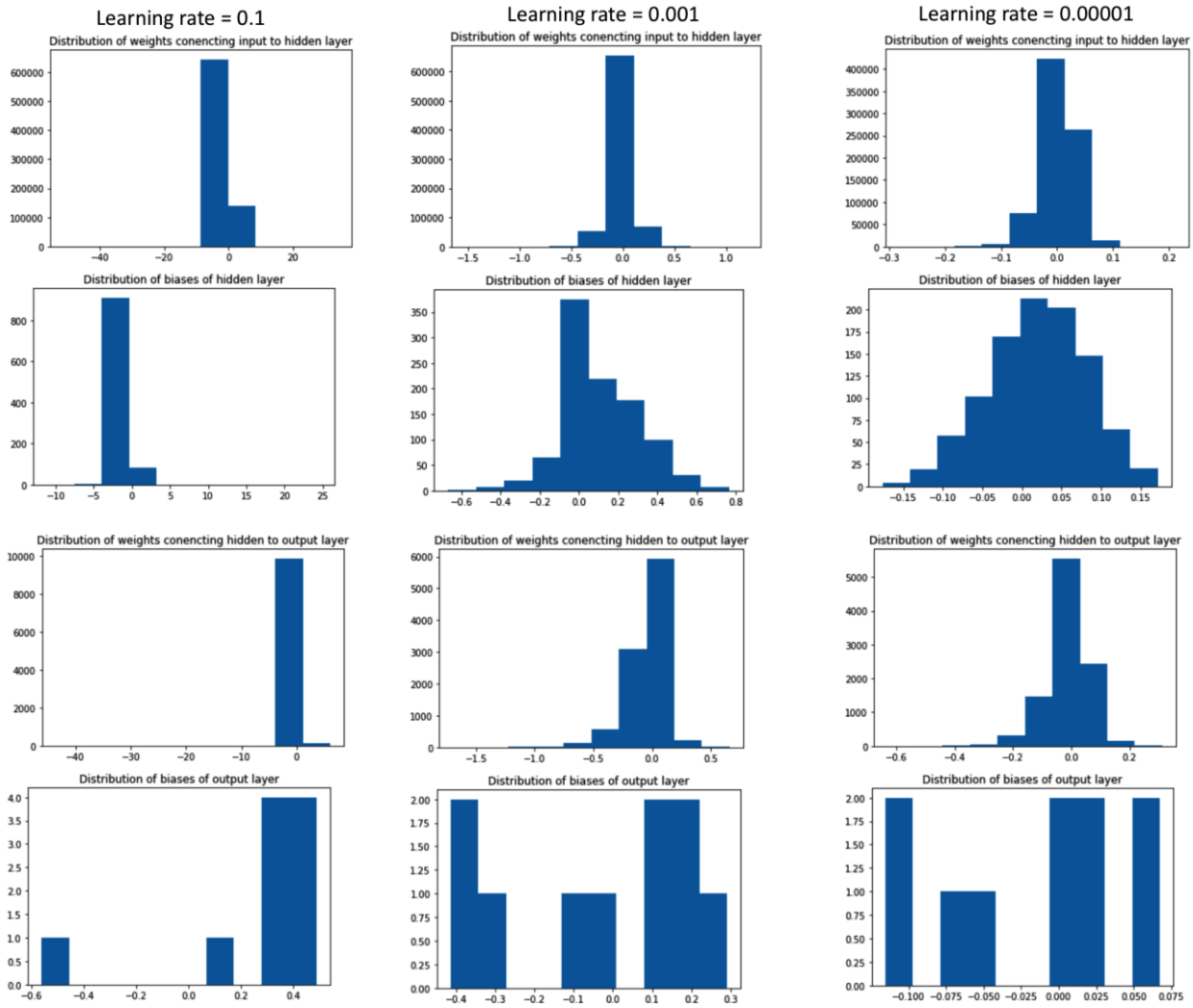


Từ các biểu đồ trước, chúng ta có thể thấy rằng mô hình học chậm hơn nhiều so với kịch bản trước đó (tốc độ học trung bình). Ở đây, phải mất ~ 100 epoch để đạt được độ chính xác ~ 89% so với 8 epoch khi tốc độ học là 0,001.

Ngoài ra, chúng ta cũng cần lưu ý rằng khoảng cách giữa sai số huấn luyện và xác thực thấp hơn nhiều khi tốc độ học thấp so với kịch bản trước đó (khi tồn tại khoảng cách tương tự vào cuối epoch 4). Lý do là do cập nhật trọng số thấp hơn nhiều khi tốc độ học thấp, điều đó có nghĩa là khoảng cách giữa quá trình huấn luyện và mất xác thực không mở rộng nhanh chóng.

Chúng ta đã tìm hiểu về tác động của tốc độ học đối với độ chính xác của tập dữ liệu huấn luyện và xác thực. Trong phần tiếp theo, chúng ta sẽ tìm hiểu cách phân bổ các giá trị trọng số khác nhau giữa các lớp đối với các giá trị tốc độ học khác nhau.

Phân phối tham số giữa các lớp cho các tốc độ học khác nhau



- Khi tốc độ học cao, các tham số có phân bố lớn hơn nhiều so với tốc độ học trung bình và thấp.
- Khi các tham số có phân bố lớn hơn, tình trạng quá khớp (overfitting) sẽ xảy ra.

Chúng ta đã nghiên cứu tác động của việc thay đổi tốc độ học trên một mô hình đã được huấn luyện trên tập dữ liệu được chia tỷ lệ.

Trong phần tiếp theo, chúng ta sẽ tìm hiểu về tác động của việc thay đổi tốc độ học lên một mô hình đã được huấn luyện trên dữ liệu không chia tỷ lệ.

Tác động của việc thay đổi tốc độ học trên tập dữ liệu không được chia tỷ lệ

Trong phần này, chúng ta sẽ quay lại làm việc trên một tập dữ liệu bằng cách không thực hiện chia cho 255 trong lớp mà chúng ta xác định tập dữ liệu. Điều này có thể được thực hiện như sau:

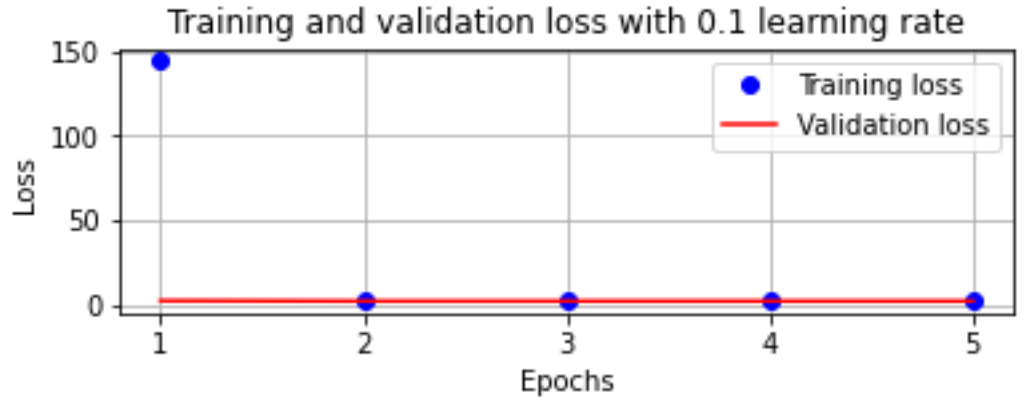
```
from torchvision import datasets
import torch
data_folder = '~/data/FMNIST' # This can be any directory you want to
# download FMNIST to
fmnist = datasets.FashionMNIST(data_folder, download=True, train=True)
tr_images = fmnist.data
tr_targets = fmnist.targets

val_fmnist = datasets.FashionMNIST(data_folder, download=True, train=False)
val_images = val_fmnist.data
val_targets = val_fmnist.targets

import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
from torch.utils.data import Dataset, DataLoader
import torch
import torch.nn as nn
device = 'cuda' if torch.cuda.is_available() else 'cpu'

#Hingh Learning Rate
class FMNISTDataset(Dataset):
    def __init__(self, x, y):
        x = x.float()
        x = x.view(-1,28*28)
        self.x, self.y = x, y
    def __getitem__(self, ix):
        x, y = self.x[ix], self.y[ix]
        return x.to(device), y.to(device)
    def __len__(self):
        return len(self.x)

from torch.optim import SGD, Adam
def get_model():
    model = nn.Sequential(
        nn.Linear(28 * 28, 1000),
        nn.ReLU(),
        nn.Linear(1000, 10)
    ).to(device)
```

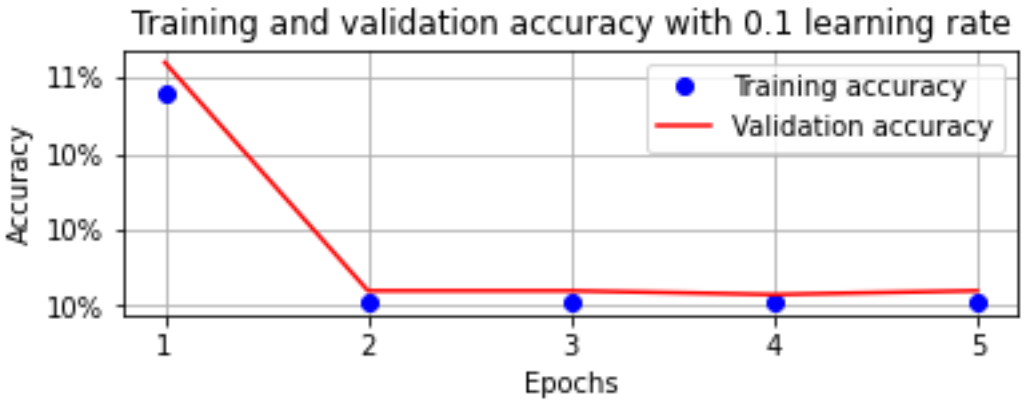
```
    loss_fn = nn.CrossEntropyLoss()
    optimizer = Adam(model.parameters(), lr=1e-1)
    return model, loss_fn, optimizer

def train_batch(x, y, model, opt, loss_fn):
    model.train()
    prediction = model(x)
    batch_loss = loss_fn(prediction, y)
    batch_loss.backward()
    optimizer.step()
    optimizer.zero_grad()
    return batch_loss.item()

def accuracy(x, y, model):
    model.eval()
    # this is the same as @torch.no_grad
    # at the top of function, only difference
    # being, grad is not computed in the with scope
    with torch.no_grad():
        prediction = model(x)
    max_values, argmaxes = prediction.max(-1)
    is_correct = argmaxes == y
    return is_correct.cpu().numpy().tolist()

def get_data():
    train = FMNISTDataset(tr_images, tr_targets)
    trn_dl = DataLoader(train, batch_size=32, shuffle=True)
    val = FMNISTDataset(val_images, val_targets)
    val_dl = DataLoader(val, batch_size=len(val_images), shuffle=False)
    return trn_dl, val_dl

@torch.no_grad()
def val_loss(x, y, model):
    prediction = model(x)
    val_loss = loss_fn(prediction, y)
    return val_loss.item()

trn_dl, val_dl = get_data()
model, loss_fn, optimizer = get_model()

train_losses, train_accuracies = [], []
val_losses, val_accuracies = [], []
for epoch in range(5):
```

```
    print(epoch)
    train_epoch_losses, train_epoch_accuracies = [], []
    for ix, batch in enumerate(iter(trn_dl)):
        x, y = batch
        batch_loss = train_batch(x, y, model, optimizer, loss_fn)
        train_epoch_losses.append(batch_loss)
    train_epoch_loss = np.array(train_epoch_losses).mean()

    for ix, batch in enumerate(iter(trn_dl)):
        x, y = batch
        is_correct = accuracy(x, y, model)
        train_epoch_accuracies.extend(is_correct)
    train_epoch_accuracy = np.mean(train_epoch_accuracies)
    for ix, batch in enumerate(iter(val_dl)):
        x, y = batch
        val_is_correct = accuracy(x, y, model)
        validation_loss = val_loss(x, y, model)
    val_epoch_accuracy = np.mean(val_is_correct)
    train_losses.append(train_epoch_loss)
    train_accuracies.append(train_epoch_accuracy)
    val_losses.append(validation_loss)
    val_accuracies.append(val_epoch_accuracy)

epochs = np.arange(5)+1
import matplotlib.ticker as mtick
import matplotlib.pyplot as plt
import matplotlib.ticker as mticker
%matplotlib inline
plt.subplot(211)
plt.plot(epochs, train_losses, 'bo', label='Training loss')
plt.plot(epochs, val_losses, 'r', label='Validation loss')
plt.gca().xaxis.set_major_locator(mticker.MultipleLocator(1))
plt.title('Training and validation loss with 0.1 learning rate')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.grid('off')
plt.show()
plt.subplot(212)
plt.plot(epochs, train_accuracies, 'bo', label='Training accuracy')
plt.plot(epochs, val_accuracies, 'r', label='Validation accuracy')
plt.gca().xaxis.set_major_locator(mticker.MultipleLocator(1))
plt.title('Training and validation accuracy with 0.1 learning rate')
plt.xlabel('Epochs')
```

```
plt.ylabel('Accuracy')
plt.gca().set_yticklabels(['{:.0f}%'.format(x*100) for x in
plt.gca().get_yticks()])
plt.legend()
plt.grid('off')
plt.show()
```

Kết quả như sau:





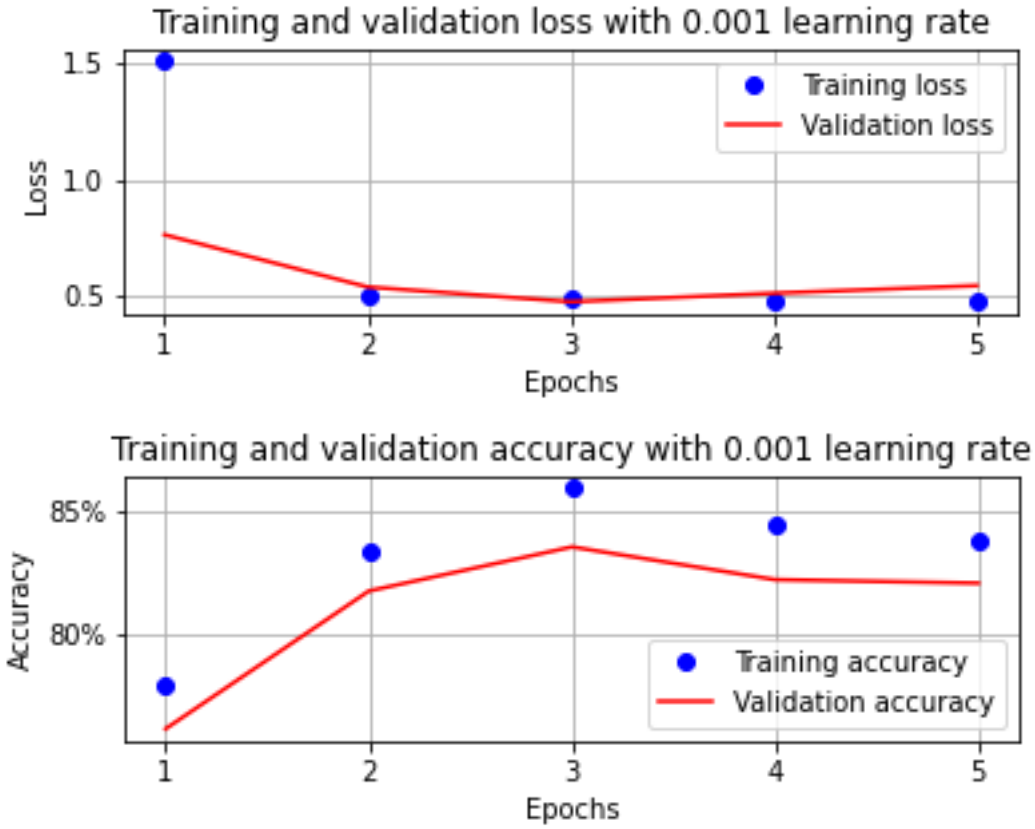
Tốc độ học trung bình:

```
def get_model():
    model = nn.Sequential(
        nn.Linear(28 * 28, 1000),
        nn.ReLU(),
        nn.Linear(1000, 10)
    ).to(device)

    loss_fn = nn.CrossEntropyLoss()
    optimizer = Adam(model.parameters(), lr=1e-3)
    return model, loss_fn, optimizer
trn_dl, val_dl = get_data()
model, loss_fn, optimizer = get_model()

train_losses, train_accuracies = [], []
```

```
val_losses, val_accuracies = [], []
for epoch in range(5):
    print(epoch)
    train_epoch_losses, train_epoch_accuracies = [], []
    for ix, batch in enumerate(iter(trn_dl)):
        x, y = batch
        batch_loss = train_batch(x, y, model, optimizer, loss_fn)
        train_epoch_losses.append(batch_loss)
    train_epoch_loss = np.array(train_epoch_losses).mean()

    for ix, batch in enumerate(iter(trn_dl)):
        x, y = batch
        is_correct = accuracy(x, y, model)
        train_epoch_accuracies.extend(is_correct)
    train_epoch_accuracy = np.mean(train_epoch_accuracies)
    for ix, batch in enumerate(iter(val_dl)):
        x, y = batch
        val_is_correct = accuracy(x, y, model)
        validation_loss = val_loss(x, y, model)
    val_epoch_accuracy = np.mean(val_is_correct)
    train_losses.append(train_epoch_loss)
    train_accuracies.append(train_epoch_accuracy)
    val_losses.append(validation_loss)
    val_accuracies.append(val_epoch_accuracy)
epochs = np.arange(5)+1
import matplotlib.ticker as mtick
import matplotlib.pyplot as plt
import matplotlib.ticker as mticker
%matplotlib inline
plt.subplot(211)
plt.plot(epochs, train_losses, 'bo', label='Training loss')
plt.plot(epochs, val_losses, 'r', label='Validation loss')
plt.gca().xaxis.set_major_locator(mticker.MultipleLocator(1))
plt.title('Training and validation loss with 0.001 learning rate')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.grid('off')
plt.show()
plt.subplot(212)
plt.plot(epochs, train_accuracies, 'bo', label='Training accuracy')
plt.plot(epochs, val_accuracies, 'r', label='Validation accuracy')
plt.gca().xaxis.set_major_locator(mticker.MultipleLocator(1))
plt.title('Training and validation accuracy with 0.001 learning rate')
```

```
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.gca().set_yticklabels(['{:.0f}%'.format(x*100) for x in
plt.gca().get_yticks()])
plt.legend()
plt.grid('off')
plt.show()
```

Kết quả như sau:



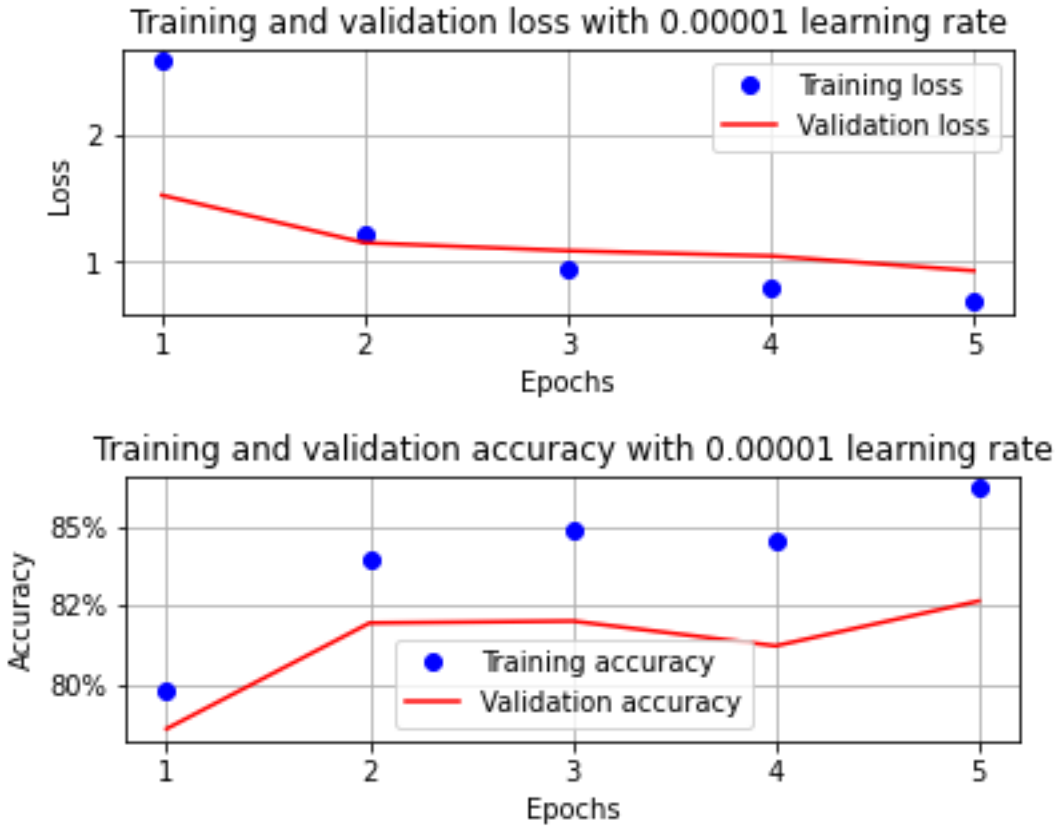
Tốc độ học thấp:

```
def get_model():
    model = nn.Sequential(
        nn.Linear(28 * 28, 1000),
        nn.ReLU(),
        nn.Linear(1000, 10)
    ).to(device)

    loss_fn = nn.CrossEntropyLoss()
    optimizer = Adam(model.parameters(), lr=1e-5)
    return model, loss_fn, optimizer
trn_dl, val_dl = get_data()
```

```
model, loss_fn, optimizer = get_model()

train_losses, train_accuracies = [], []
val_losses, val_accuracies = [], []
for epoch in range(5):
    print(epoch)
    train_epoch_losses, train_epoch_accuracies = [], []
    for ix, batch in enumerate(iter(trn_dl)):
        x, y = batch
        batch_loss = train_batch(x, y, model, optimizer, loss_fn)
        train_epoch_losses.append(batch_loss)
    train_epoch_loss = np.array(train_epoch_losses).mean()

    for ix, batch in enumerate(iter(trn_dl)):
        x, y = batch
        is_correct = accuracy(x, y, model)
        train_epoch_accuracies.extend(is_correct)
    train_epoch_accuracy = np.mean(train_epoch_accuracies)
    for ix, batch in enumerate(iter(val_dl)):
        x, y = batch
        val_is_correct = accuracy(x, y, model)
        validation_loss = val_loss(x, y, model)
    val_epoch_accuracy = np.mean(val_is_correct)
    train_losses.append(train_epoch_loss)
    train_accuracies.append(train_epoch_accuracy)
    val_losses.append(validation_loss)
    val_accuracies.append(val_epoch_accuracy)
epochs = np.arange(5)+1
import matplotlib.ticker as mtick
import matplotlib.pyplot as plt
import matplotlib.ticker as mticker
%matplotlib inline
plt.subplot(211)
plt.plot(epochs, train_losses, 'bo', label='Training loss')
plt.plot(epochs, val_losses, 'r', label='Validation loss')
plt.gca().xaxis.set_major_locator(mticker.MultipleLocator(1))
plt.title('Training and validation loss with 0.00001 learning rate')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.grid('off')
plt.show()
plt.subplot(212)
plt.plot(epochs, train_accuracies, 'bo', label='Training accuracy')
```

```
plt.plot(epochs, val_accuracies, 'r', label='Validation accuracy')
plt.gca().xaxis.set_major_locator(mticker.MultipleLocator(1))
plt.title('Training and validation accuracy with 0.00001 learning rate')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.gca().set_yticklabels(['{:.0f}%'.format(x*100) for x in
plt.gca().get_yticks()])
plt.legend()
plt.grid('off')
plt.show()
```

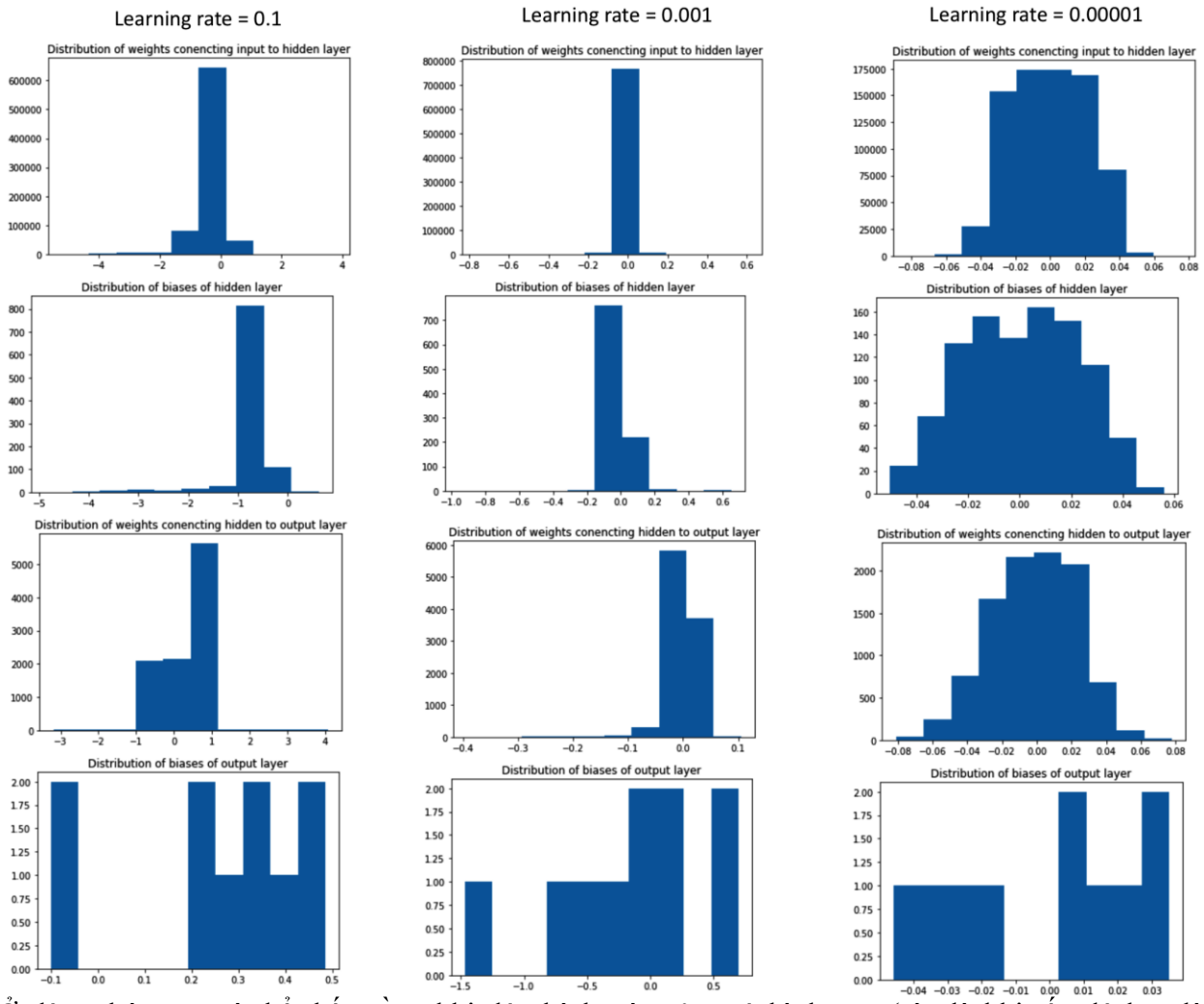
Kết quả như sau:



Như chúng ta có thể thấy, ngay cả khi tập dữ liệu không được chia tỷ lệ, chúng ta không thể huấn luyện một mô hình chính xác khi tốc độ học là 0,1. Hơn nữa, độ chính xác không cao như phần trước khi tỷ lệ học là 0,001.

Cuối cùng, khi tốc độ học rất nhỏ (0,00001), mô hình có thể học tốt như trong các phần trước, nhưng lần này bị tình trạng **overfitting** trên dữ liệu huấn luyện.

Hãy tìm hiểu lý do tại sao điều này xảy ra bằng cách xem xét phân phối tham số giữa các lớp như sau:



Ở đây, chúng ta có thể thấy rằng khi độ chính xác của mô hình cao (tức là khi tốc độ học là 0,00001), các trọng số có phạm vi nhỏ hơn nhiều (thường nằm trong khoảng từ -0,05 đến 0,05 trong trường hợp này) so với khi tốc độ học cao .

Các trọng số có thể được điều chỉnh về một giá trị nhỏ vì tốc độ học nhỏ. Lưu ý rằng kịch bản trong đó tốc độ học là 0,00001 trên tập dữ liệu không được chia tỷ lệ tương đương với kịch bản tốc độ học là 0,001 trên tập dữ liệu được chia tỷ lệ. Điều này là do các trọng số bây giờ có thể di chuyển về một giá trị rất nhỏ (vì gradient * learning rate là một giá trị rất nhỏ, do tốc độ học là nhỏ).

LAB 6:

XÂY DỰNG MẠNG NƠ-RON HỌC SÂU

Yêu cầu: sinh viên tự thực hiện xây dựng mạng nơ-ron học sâu cho bài toán nhận dạng quần áo giày dép thời trang với bộ dữ liệu FASHION-MNIST tương tự Lab 5.

LAB 7:

PHÂN TÍCH DỮ LIỆU DẠNG VĂN BẢN VỚI NLTK

Nội dung:

1. Giới thiệu về thư viện NLTK
2. Tìm 1 từ với NLTK
3. Phân tích tần số của các từ
4. Lựa chọn các từ trong văn bản
5. Bigrams và collocations
6. Sử dụng văn bản trên mạng
7. Rút trích văn bản từ trang html
8. Phân tích cảm xúc người dùng
9. Bài tập áp dụng

1. Giới thiệu về thư viện NLTK:

Trong các phần trước, ta đã học cách phân tích dữ liệu dạng số hoặc dạng bảng thông qua các biểu thức toán học hoặc kỹ thuật thống kê. Nhưng hầu hết dữ liệu bao gồm dạng văn bản, quy tắc ngữ pháp khác nhau tùy vào ngôn ngữ nào đó. Trong một văn bản, các từ và nghĩa của từ có thể là nguồn thông tin rất hữu ích. Trong lab này, ta sẽ tìm hiểu một số kỹ thuật phân tích văn bản bằng thư viện NLTK cho phép bạn thực hiện các phép toán phức tạp. Hơn nữa giúp ta hiểu được một phần quan trọng trong phân tích dữ liệu.

Cài đặt thư viện NLTK:

Vào của sổ command line, gõ lệnh pip install NLTK

Import thư viện NLTK và download công cụng NLTK

```
import nltk
nltk.download_shell()
```

Kết quả như sau

```
NLTK Downloader
---------------------------------------------------------------------------
    d) Download   l) List    u) Update   c) Config   h) Help   q) Quit
---------------------------------------------------------------------------
Downloader> l

Packages:
  [ ] abc..................... Australian Broadcasting Commission 2006
  [ ] alpino.................. Alpino Dutch Treebank
  [*] averaged_perceptron_tagger Averaged Perceptron Tagger
  [ ] averaged_perceptron_tagger_ru Averaged Perceptron Tagger (Russian)
```

```
  [ ] basque_grammars..... Grammars for Basque
  [ ] biocreative_ppi..... BioCreAtIvE (Critical Assessment of Information
                           Extraction Systems in Biology)
  [ ] bllip_wsj_no_aux.... BLLIP Parser: WSJ Model
  [ ] book_grammars....... Grammars from NLTK Book
  [ ] brown............... Brown Corpus
  [ ] brown_tei........... Brown Corpus (TEI XML Version)
  [ ] cess_cat............ CESS-CAT Treebank
  [ ] cess_esp............ CESS-ESP Treebank
  [ ] chat80.............. Chat-80 Data Files
  [ ] city_database....... City Database
  [ ] cmudict............. The Carnegie Mellon Pronouncing Dictionary
(0.6)
  [ ] comparative_sentences Comparative Sentence Dataset
  [ ] comtrans............ ComTrans Corpus Sample
  [ ] conll2000........... CONLL 2000 Chunking Corpus
  [ ] conll2002........... CONLL 2002 Named Entity Recognition Corpus
Hit Enter to continue:
  [ ] conll2007........... Dependency Treebanks from CoNLL 2007 (Catalan
                           and Basque Subset)
  [ ] crubadan............ Crubadan Corpus
  [ ] dependency_treebank. Dependency Parsed Treebank
  [ ] dolch............... Dolch Word List
  [ ] europarl_raw........ Sample European Parliament Proceedings Parallel
                           Corpus
  [ ] floresta............ Portuguese Treebank
  [ ] framenet_v15........ FrameNet 1.5
  [ ] framenet_v17........ FrameNet 1.7
  [ ] gazetteers.......... Gazeteer Lists
  [ ] genesis............. Genesis Corpus
  [*] gutenberg........... Project Gutenberg Selections
  [ ] ieer................ NIST IE-ER DATA SAMPLE
  [ ] inaugural........... C-Span Inaugural Address Corpus
  [ ] indian.............. Indian Language POS-Tagged Corpus
  [ ] jeita............... JEITA Public Morphologically Tagged Corpus (in
                           ChaSen format)
  [ ] kimmo............... PC-KIMMO Data Files
  [ ] knbc................ KNB Corpus (Annotated blog corpus)
  [ ] large_grammars...... Large context-free and feature-based grammars
                           for parser comparison
Hit Enter to continue:
  [ ] lin_thesaurus....... Lin's Dependency Thesaurus
  [ ] mac_morpho.......... MAC-MORPHO: Brazilian Portuguese news text with
                           part-of-speech tags
  [ ] machado............. Machado de Assis -- Obra Completa
  [ ] masc_tagged......... MASC Tagged Corpus
  [ ] maxent_ne_chunker... ACE Named Entity Chunker (Maximum entropy)
  [ ] maxent_treebank_pos_tagger Treebank Part of Speech Tagger (Maximum
entropy)
  [ ] moses_sample........ Moses Sample Models
  [*] movie_reviews....... Sentiment Polarity Dataset Version 2.0
  [ ] mte_teip5........... MULTEXT-East 1984 annotated corpus 4.0
  [ ] mwa_ppdb............ The monolingual word aligner (Sultan et al.
                           2015) subset of the Paraphrase Database.
```

```
  [*] names............... Names Corpus, Version 1.3 (1994-03-29)
  [ ] nombank.1.0......... NomBank Corpus 1.0
  [ ] nonbreaking_prefixes Non-Breaking Prefixes (Moses Decoder)
  [ ] nps_chat............ NPS Chat
  [ ] omw................. Open Multilingual Wordnet
  [ ] opinion_lexicon..... Opinion Lexicon
  [ ] panlex_swadesh...... PanLex Swadesh Corpora
  [ ] paradigms........... Paradigm Corpus
  [ ] pe08................ Cross-Framework and Cross-Domain Parser
                           Evaluation Shared Task
Hit Enter to continue:
  [ ] perluniprops........ perluniprops: Index of Unicode Version 7.0.0
                           character properties in Perl
  [ ] pil................. The Patient Information Leaflet (PIL) Corpus
  [ ] pl196x.............. Polish language of the XX century sixties
  [ ] porter_test......... Porter Stemmer Test Files
  [ ] ppattach............ Prepositional Phrase Attachment Corpus
  [ ] problem_reports..... Problem Report Corpus
  [ ] product_reviews_1... Product Reviews (5 Products)
  [ ] product_reviews_2... Product Reviews (9 Products)
  [ ] propbank............ Proposition Bank Corpus 1.0
  [ ] pros_cons........... Pros and Cons
  [ ] ptb................. Penn Treebank
  [*] punkt............... Punkt Tokenizer Models
  [ ] qc.................. Experimental Data for Question Classification
  [ ] reuters............. The Reuters-21578 benchmark corpus, ApteMod
                           version
  [ ] rslp................ RSLP Stemmer (Removedor de Sufixos da Lingua
                           Portuguesa)
  [ ] rte................. PASCAL RTE Challenges 1, 2, and 3
  [ ] sample_grammars..... Sample Grammars
  [ ] semcor.............. SemCor 3.0
Hit Enter to continue:
  [ ] senseval............ SENSEVAL 2 Corpus: Sense Tagged Text
  [ ] sentence_polarity... Sentence Polarity Dataset v1.0
  [ ] sentiwordnet........ SentiWordNet
  [ ] shakespeare......... Shakespeare XML Corpus Sample
  [ ] sinica_treebank..... Sinica Treebank Corpus Sample
  [ ] smultron............ SMULTRON Corpus Sample
  [ ] snowball_data....... Snowball Data
  [ ] spanish_grammars.... Grammars for Spanish
  [ ] state_union......... C-Span State of the Union Address Corpus
  [*] stopwords........... Stopwords Corpus
  [ ] subjectivity........ Subjectivity Dataset v1.0
  [ ] swadesh............. Swadesh Wordlists
  [ ] switchboard......... Switchboard Corpus Sample
  [ ] tagsets............. Help on Tagsets
  [ ] timit............... TIMIT Corpus Sample
  [ ] toolbox............. Toolbox Sample Files
  [ ] treebank............ Penn Treebank Sample
  [ ] twitter_samples..... Twitter Samples
  [ ] udhr2............... Universal Declaration of Human Rights Corpus
                           (Unicode Version)
  [ ] udhr................ Universal Declaration of Human Rights Corpus
```

```
Hit Enter to continue:
  [ ] unicode_samples..... Unicode Samples
  [ ] universal_tagset.... Mappings to the Universal Part-of-Speech Tagset
  [ ] universal_treebanks_v20 Universal Treebanks Version 2.0
  [ ] vader_lexicon....... VADER Sentiment Lexicon
  [ ] verbnet............. VerbNet Lexicon, Version 2.1
  [ ] webtext............. Web Text Corpus
  [ ] wmt15_eval.......... Evaluation data from WMT15
  [ ] word2vec_sample..... Word2Vec Sample
  [ ] wordnet............. WordNet
  [ ] wordnet_ic.......... WordNet-InfoContent
  [ ] words............... Word Lists
  [ ] ycoe................ York-Toronto-Helsinki Parsed Corpus of Old
                           English Prose

Collections:
  [P] all-corpora......... All the corpora
  [P] all-nltk............ All packages available on nltk_data gh-pages
                           branch
  [P] all................. All packages
  [P] book................ Everything used in the NLTK Book
  [P] popular............. Popular packages
  [P] tests............... Packages for running tests
Hit Enter to continue:
  [ ] third-party......... Third-party data packages

([*] marks installed packages; [P] marks partially installed collections)

---------------------------------------------------------------------------
-
    d) Download   l) List    u) Update   c) Config   h) Help   q) Quit
---------------------------------------------------------------------------
-
Downloader> d

Download which package (l=list; x=cancel)?
  Identifier> gutenberg
    Downloading package gutenberg to
        C:\Users\nelli\AppData\Roaming\nltk_data...
      Package gutenberg is already up-to-date!

---------------------------------------------------------------------------
-
    d) Download   l) List    u) Update   c) Config   h) Help   q) Quit
---------------------------------------------------------------------------
-
Downloader> q
```

Để có thể tạo một số các ví dụ để tìm hiểu về thư viện, bạn cần một số các văn bản để làm việc. Một nguồn văn bản tuyệt vời phù hợp cho mục đích này là kho ngữ liệu Gutenberg, có trong bộ sưu tập kho ngữ liệu. Kho ngữ liệu Gutenberg gồm các văn bản được trích xuất từ kho lưu trữ điện tử có tên là dự án Gutenberg (http: // www.gutenberg.org/). Có hơn 25.000 sách điện tử trong kho lưu trữ này. Để tải xuống gói này, trước tiên hãy nhập tùy chọn d để tải xuống. Công cụ sẽ hỏi bạn cho tên gói, vì vậy bạn nhập tên gutenberg.

```
---------------------------------------------------------------------------
    d) Download   l) List    u) Update   c) Config   h) Help   q) Quit
---------------------------------------------------------------------------
Downloader> d

Download which package (l=list; x=cancel)?
  Identifier> gutenberg
```

Tại thời điểm này, các gói sẽ bắt đầu download.

Đối với những lần sau, nếu bạn đã biết tên của gói bạn muốn tải xuống, chỉ cần nhập lệnh nltk.download() với tên gói là tham số đầu vào. Thao tác này sẽ không mở công cụ NLTK Downloader, nhưng sẽ trực tiếp tải xuống gói yêu cầu. Vì vậy, thao tác trước đó tương đương với việc viết lệnh sau:

```
nltk.download('gutenberg')
```

Sau khi hoàn tất, bạn có xem tên các tập tin có trong gói 'gutenberg' bằng lệnh fileids().

```
gb = nltk.corpus.gutenberg
print("Gutenberg files : ", gb.fileids())
```

Kết quả như sau

```
Gutenberg files :  ['austen-emma.txt', 'austen-persuasion.txt', 'austen-
sense.txt', 'bible-kjv.txt', 'blake-poems.txt', 'bryant-stories.txt',
'burgess-busterbrown.txt', 'carroll-alice.txt', 'chesterton-ball.txt',
'chesterton-brown.txt', 'chesterton-thursday.txt', 'edgeworth-
parents.txt', 'melville-moby_dick.txt', 'milton-paradise.txt',
'shakespeare-caesar.txt', 'shakespeare-hamlet.txt', 'shakespeare-
macbeth.txt', 'whitman-leaves.txt']
```

Để truy cập nội dung bên trong của một trong các tập tin này, trước tiên bạn chọn một tập tin, chẳng hạn Macbeth của Shakespeare (shakespeare-macbeth.txt), sử dụng hàm words (), rồi gán cho một biến nào đó.

```
macbeth = nltk.corpus.gutenberg.words('shakespeare-macbeth.txt')
```

Nếu bạn muốn biết chiều dài của văn bản trên (bao nhiêu từ), bạn dùng hàm len()

```
len(macbeth)
```

Kết quả như sau

23140

Nếu bạn muốn hiển thị 10 từ đầu tiên của tập tin

```
macbeth [:10]
```

Kết quả như sau

```
['[',
 'The',
 'Tragedie',
 'of',
 'Macbeth',
 'by',
 'William',
 'Shakespeare',
 '1603',
 ']']
```

Hàm trên trích ra 10 từ đầu tiên là tiêu đề của tập tin, dấu ngoặc vuông được tính là 1.

Ta muốn trích 5 câu đầu tiên của tập tin (một câu được kẹp trong cặp ngoặc vuông), ta dùng hàm sent()

```
macbeth_sents = nltk.corpus.gutenberg.sents('shakespeare-macbeth.txt')
macbeth_sents[:5]
```

Kết quả như sau

```
[['[',
  'The',
  'Tragedie',
  'of',
  'Macbeth',
  'by',
  'William',
  'Shakespeare',
  '1603',
  ']'],
 ['Actus', 'Primus', '.'],
 ['Scoena', 'Prima', '.'],
 ['Thunder', 'and', 'Lightning', '.'],
 ['Enter', 'three', 'Witches', '.']]
```

2. Tìm 1 từ với NLTK:

Tìm từ 'Stage' xuất hiện trong văn bản text

```
text = nltk.Text(macbeth)
text.concordance('Stage')
```

Kết quả như sau

```
Displaying 3 of 3 matches:
nts with Dishes and Seruice ouer the Stage . Then enter Macbeth Macb . If
it we with mans Act , Threatens his bloody Stage : byth ' Clock ' tis Day,
And yet d struts and frets his houre vpon the Stage , And then is heard no
more . It is
```

Tìm từ xuất hiện trước và sau từ 'Stage'

```
text.common_contexts(['Stage'])
```

Kết quả như sau

```
the_then bloody_byth the_and
```

Tìm từ tương tự từ 'Stage'

```
text.similar('Stage')
```

Kết quả như sau

```
fogge ayre bleeding reuolt good shew heeles skie other sea feare
consequence heart braine seruice herbenger lady round deed doore
```

3. Phân tích tần số của các từ

Phân tích số lần xuất hiện của từ:

Muốn xem 10 từ thông dụng nhất trong văn bản xuất hiện bao nhiêu lần, dùng lệnh most_common()

```
fd = nltk.FreqDist(macbeth)
fd.most_common(10)
```

Kết quả như sau

```
[(',', 1962),
 ('.', 1235),
 ("'", 637),
 ('the', 531),
 (':', 477),
```

```
 ('and', 376),
 ('I', 333),
 ('of', 315),
 ('to', 311),
 ('?', 241)]
```

Stopword là những từ thông dụng, thường ít có ý nghĩa trong quá trình phân tích văn bản và thường cần được loại bỏ.

Muốn download stopword

```
nltk.download('stopwords')
```

Kết quả như sau

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\nelli\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
True
```

Muốn xem các stopword trong tiếng Anh, dùng lệnh

```
sw = set(nltk.corpus.stopwords.words('english'))
print(len(sw))
list(sw)[:10]
```

Kết quả như sau

```
179
['once',
 'o',
 "hasn't",
 'i',
 "you'd",
 'because',
 'yourselves',
 'being',
 "wouldn't",
 'been']
```

Có 179 stopword trong từ vựng tiếng Anh. Ta sẽ loại bỏ các từ stopword trong biến macbeth

```
macbeth_filtered = [w for w in macbeth if w.lower() not in sw]
len(macbeth_filtered)
```

Kết quả như sau

```
14946
```

```
fd = nltk.FreqDist(macbeth_filtered)
fd.most_common(10)
```

Kết quả như sau

```
[(',', 1962),
 ('.', 1235),
 ("'", 637),
 (':', 477),
 ('?', 241),
 ('Macb', 137),
 ('haue', 117),
 ('-', 100),
 ('Enter', 80),
 ('thou', 63)]
```

Bây giờ, 10 từ phổ biến nhất đầu tiên được trả về, các stopword đã được loại bỏ nhưng vẫn còn các dấu câu, ta cần loại bỏ các dấu câu theo lệnh sau:

```
import string
punctuation = set(string.punctuation)
macbeth_filtered2 = [w.lower() for w in macbeth if w.lower() not in sw an
d w.lower() not in punctuation]
fd = nltk.FreqDist(macbeth_filtered2)
fd.most_common(10)
```

Kết quả như sau

```
[('macb', 137),
 ('haue', 122),
 ('thou', 90),
 ('enter', 81),
 ('shall', 68),
 ('macbeth', 62),
 ('vpon', 62),
 ('thee', 61),
 ('macd', 58),
 ('vs', 57)]
```

4. Lựa chọn các từ trong văn bản

Lựa chọn các từ trong văn bản:

Một dạng khác trong việc xử lý và phân tích dữ liệu là quá trình chọn một từ dựa vào 1 đặc điểm nào đó. Ví dụ như bạn cần quan tâm rút trích từ dựa vào độ dài của từ.

Rút trích các từ có độ dài lớn nhất, ví dụ các từ có độ dài lớn hơn 12 ký tự , dùng lệnh sau:

```
long_words = [w for w in macbeth if len(w)> 12]
sorted(long_words)
```

Kết quả như sau

['Assassination',
'Chamberlaines',
'Distinguishes',
'Gallowgrosses',
'Metaphysicall',
'Northumberland',
'Voluptuousnesse',
'commendations',
'multitudinous',
'supernaturall',
'vnaccompanied']

Rút trích các từ có chứa chuỗi ‘ious’

```
ious_words = [w for w in macbeth if 'ious' in w]
ious_words = set(ious_words)
sorted(ious_words)
```

Kết quả như sau

['Auaricious',
'Gracious',
'Industrious',
'Iudicious',

'Luxurious',
'Malicious',
'Obliuious',
'Pious',
'Rebellious',
'compunctious',
'furious',
'gracious',
'pernicious',
'pernitious',
'pious',
'precious',
'rebellious',
'sacrilegious',
'serious',
'spacious',
'tedious']

5. Bigrams và collocations

Một phần cơ bản trong phân tích văn bản là xem xét các cặp từ thay thế cho các từ đơn. Từ 'is' và 'yellow' là 1 ví dụ về bigram vì chúng có thể kết hợp lại và có ý nghĩa, vì thế "is yellow" có thể được tìm thấy trong văn bản. Ví dụ "fast food", "pay attention", "good morning",..,những bigram này gọi là collocation.

Lọc các bigram sau khi đã loại các stopword và các dấu câu, dùng lệnh sau:

```
bgrms = nltk.FreqDist(nltk.bigrams(macbeth_filtered2))
bgrms.most_common(15)
```

Kết quả như sau

[(('enter', 'macbeth'), 16),
(('exeunt', 'scena'), 15),
(('thane', 'cawdor'), 13),
(('knock', 'knock'), 10),
(('st', 'thou'), 9),
(('thou', 'art'), 9),
(('lord', 'macb'), 9),
(('haue', 'done'), 8),
(('macb', 'haue'), 8),
(('good', 'lord'), 8),
(('let', 'vs'), 7),
(('enter', 'lady'), 7),
(('wee', 'l'), 7),
(('would', 'st'), 6),
(('macbeth', 'macb'), 6)]

Ngoài bigram ra, còn có trigram, sự kết hợp của 3 từ, ta dùng lệnh trigrams()

```
tgrms = nltk.FreqDist(nltk.trigrams (macbeth_filtered2))
tgrms.most_common(10)
```

Kết quả như sau

[(('knock', 'knock', 'knock'), 6),
(('enter', 'macbeth', 'macb'), 5),
(('enter', 'three', 'witches'), 4),
(('exeunt', 'scena', 'secunda'), 4),
(('good', 'lord', 'macb'), 4),
(('three', 'witches', '1'), 3),
(('exeunt', 'scena', 'tertia'), 3),
(('thunder', 'enter', 'three'), 3),
(('exeunt', 'scena', 'quarta'), 3),
(('scena', 'prima', 'enter'), 3)]

6. Sử dụng văn bản trên mạng

Các ví dụ trước, chúng ta đã làm việc với văn bản sử dụng thư viện NLTK và gói 'gutenberg'. Nhưng trong thực tế, bạn sẽ cần phải truy cập Internet để trích xuất văn bản và thu thập nó như một kho tài liệu được sử dụng để phân tích với NLTK.

Import thư viện và mở url để đọc file

```
from urllib import request
url = "http://www.gutenberg.org/files/2554/2554-0.txt"
response = request.urlopen(url)
raw = response.read().decode('utf8')
raw[:75]
```

Kết quả như sau

'\ufeffThe Project Gutenberg EBook of Crime and Punishment, by Fyodor
Dostoevsky\r'

Thay bằng các lệnh sau:

```
from urllib import request
url = "http://www.gutenberg.org/files/2554/2554-0.txt"
response = request.urlopen(url)
raw = response.read().decode('utf8-sig')
raw[:75]
```

Kết quả như sau

'The Project Gutenberg EBook of Crime and Punishment, by Fyodor Dostoevsky\
r\n'

Thực hiện các lệnh sau:

```
tokens = nltk.word_tokenize (raw)
webtext = nltk.Text (tokens)
webtext[:12]
```

Kết quả như sau

['The',
'Project',
'Gutenberg',
'EBook',
'of',
'Crime',
'and',
'Punishment',

',',
'by',
'Fyodor',
'Dostoevsky']

7. Rút trích văn bản từ trang html

Trong ví dụ vừa rồi, ta tạo 1 copus NLTK từ 1 văn bản download từ Internet. Nhưng hầu như các văn bản trên Internet ở dạng trang html. Trong phần này, chúng ta sẽ tìm hiểu cách rút trích văn bản từ trang html.

```
url = "http://news.bbc.co.uk/2/hi/health/2284783.stm"
html = request.urlopen(url).read().decode('utf8')
html[:120]
```

Kết quả như sau

'<!doctype html public "-//W3C//DTD HTML 4.0 Transitional//EN"
"http://www.w3.org/TR/REC-html40/loose.dtd">\r\n<html>\r\n<hea'

Ta dùng thư viện bs4 (BeautifulSoup) cung cấp cho bạn các trình phân tích cú pháp phù hợp có thể nhận dạng HTML và trích xuất văn bản.

```
from bs4 import BeautifulSoup
raw = BeautifulSoup(html, "lxml").get_text()
tokens = nltk.word_tokenize(raw)
text = nltk.Text(tokens)
```

8. Phân tích cảm xúc người dùng

Phân tích cảm xúc là một lĩnh vực nghiên cứu gần đây để đánh giá quan điểm của người dùng về một chủ đề nào đó. Việc phân tích này dựa trên về các kỹ thuật khác nhau sử dụng phân tích văn bản, các lĩnh vực trên các mạng xã hội và các diễn đàn.

Nhờ nhận xét và đánh giá của người dùng, các thuật toán phân tích cảm xúc có thể đánh giá mức độ đánh giá hoặc đánh giá dựa trên các từ khóa nhất định. Mức độ đánh giá được gọi là ý kiến và có ba giá trị có thể có: **positive, neutral hoặc negative**. Việc đánh giá ý kiến này do đó trở thành một dạng phân loại.

Mục đích của ví dụ dưới đây là tìm các từ lặp lại nhiều nhất dạng positive và dạng negative hoặc các từ khóa liên quan đến một ý kiến nào đó, sử dụng thuật toán Naïve Bayes để phân loại cảm xúc người dùng dựa vào các đoạn văn bản movie_reviews của họ.

Trước tiên, download các movie review dùng lệnh sau:

```
nltk.download('movie_reviews')
```

Sau đó xây dựng tập train dựa vào copus trên. Tạo 1 mảng tên là documents. Mảng này chứa cột đầu tiên là nội dung review của người dùng, và cột thứ 2 là cột đánh giá **positive, neutral, or negative.** Sau đó trộn các dòng này ngẫn nhiên.

```
import random
reviews = nltk.corpus.movie_reviews
documents = [(list(reviews.words(fileid)), category)
for category in reviews.categories()
for fileid in reviews.fileids(category)]
random.shuffle(documents)
```

Xem nội dung review đầu tiên (dòng 0, cột 0)

```
first_review = ' '.join(documents[0][0])
```

```
print(first_review)
```

Kết quả như sau

topless women talk about their lives falls into that category that I mentioned in the devil ' s advocate : movies that have a brilliant beginning but don ' t know how to end . it begins by introducing us to a selection of characters who all know each other . there is liz , who oversleeps and so is running late for her appointment , prue who is getting married ,...

Xem kết quả review đầu tiên (dòng 0, cột 1)

```
documents[0][1]
```

Kết quả như sau

'neg'

Ta cần tạo bảng phân phối tần số các từ trong copus, bảng này cần chuyển sang dạng list, ta dùng hàm list()

```
all_words = nltk.FreqDist(w.lower() for w in reviews.words())
word_features = list(all_words)
```

Sau đó, bước tiếp theo là xác định một hàm để tính toán các đặc trưng, tức là những từ đủ quan trọng để thiết lập ý kiến của một review.

```
def document_features(document, word_features):
    document_words = set(document)
    features = {}
    for word in word_features:
        features['{}'.format(word)] = (word in document_words)
    return features
```

Khi bạn định nghĩa hàm document_features(), bạn tạo 1 tập các documents

```
featuresets = [(document_features(d,word_features), c) for (d,c) in docum
ents]
len(featuresets)
```

Kết quả như sau

```
2000
```

Tạo tập train và tập test: 1500 dòng đầu dùng cho tập train và 500 dòng còn lại dùng cho tập test để đánh giá độ chính xác của mô hình.

```
train_set, test_set = featuresets[1500:], featuresets[:500]
classifier = nltk.NaiveBayesClassifier.train(train_set)
```

Dùng thuật toán Naïve Bayes để phân loại, dùng thư viện NLTK. Sau đó tính toán độ chính xác của thuật toán

```
train_set, est_set = featuresets[1500:], featuresets[:500]
classifier = nltk.NaiveBayesClassifier.train(train_set)
print(nltk.classify.accuracy(classifier, test_set))
```

Kết quả

```
0.85
```

Chúng ta đã hoàn tất việc phân tích, dưới đây các từ với trọng số lớn nhất của các review được đánh giá là positive và negative

```
classifier.show_most_informative_features(10)
```

Kết quả như sau

```
Most Informative Features
```

```
        blame = True              neg : pos    =     12.7 : 1.0
  wonderfully = True              pos : neg    =     12.4 : 1.0
          era = True              pos : neg    =      9.5 : 1.0
       inept = True               neg : pos    =      8.1 : 1.0
       anger = True               pos : neg    =      7.8 : 1.0
        porn = True               neg : pos    =      7.3 : 1.0
      finest = True               pos : neg    =      7.2 : 1.0
  assistant = True                pos : neg    =      7.2 : 1.0
        mass = True               pos : neg    =      6.6 : 1.0
   realizes = True                pos : neg    =      6.6 : 1.0
```

9. Bài tập áp dụng

1. Viết chương trình Python với thư viện NLTK để liệt kê các tên của copus.
2. Viết chương trình Python với thư viện NLTK để liệt kê danh sách các stopword bằng các ngôn ngữ khác nhau.
3. Viết chương trình Python với thư viện NLTK để kiểm tra danh sách các stopword bằng các ngôn ngữ khác nhau.
4. Viết chương trình Python với thư viện NLTK để loại bỏ các stopword từ một văn bản đã cho.
5. Viết chương trình Python với thư viện NLTK bỏ qua các stopword từ danh sách các stopword.
6. Viết một chương trình Python với thư viện NLTK để tìm định nghĩa và ví dụ của một từ đã cho bằng WordNet từ Wikipedia,
7. Viết chương trình Python với thư viện NLTK để tìm tập hợp các từ đồng nghĩa và trái nghĩa của một từ nào đó.
8. Viết chương trình Python với thư viện NLTK để có cái nhìn tổng quan về bộ tag, chi tiết của một tag cụ thể trong bộ tag và chi tiết về một số bộ tag liên quan, sử dụng biểu thức chính quy.
9. Viết chương trình Python với thư viện NLTK để so sánh sự giống nhau của hai danh từ đã cho.
10. Viết chương trình Python với thư viện NLTK để so sánh sự giống nhau của hai động từ đã cho.
11. Viết chương trình Python với thư viện NLTK để tìm số lượng tên nam và nữ trong các tên kho ngữ liệu. In tên 10 nam và nữ đầu tiên. Lưu ý: Kho văn bản tên chứa tổng cộng khoảng 2943 nam (male.txt) và 5001 nữ (Female.txt) tên. Kho được biên soạn bởi Kantrowitz, Ross.
12. Viết chương trình Python với thư viện NLTK để in 15 kết hợp ngẫu nhiên đầu tiên được gắn nhãn nam và được gắn nhãn tên nữ từ kho tên.
13. Viết chương trình Python với thư viện NLTK để trích xuất ký tự cuối cùng của tất cả các tên được gắn nhãn và tạo mảng mới với chữ cái cuối cùng của mỗi tên và nhãn được liên kết.