
 <p>دانشگاه صنعتی خواجه نصیرالدین طوسی دانشکده مهندسی برق - گروه مهندسی کنترل</p>	<p>به نام خدا</p> <p>دانشگاه تهران - دانشکاه صنعتی خواجه نصیرالدین طوسی تهران</p> <p>دانشکده مهندسی برق و کامپیوتر</p>	
<h2 style="text-align: center;">شبکه‌های پرسپترون چندلایه</h2>		

<p>محمد جواد احمدی</p>	<p>نام و نام خانوادگی</p>
<p>۴۰۱۰۰۰۸۶</p>	<p>شماره دانشجویی</p>

## فهرست مطالب

۴	پاسخ پرسش چهارم	۱
۴	۱.۱ آشنایی و کار با دیتاست (پیش‌پردازش)	
۴	۱.۱.۱ قسمت I	
۵	۲.۱.۱ قسمت II	
۶	۳.۱.۱ قسمت III	
۷	۴.۱.۱ قسمت IV	
۹	۵.۱.۱ قسمت V	
۱۱	۶.۱.۱ قسمت VI	
۱۴	۷.۱.۱ قسمت VII	
۱۵	۸.۱.۱ قسمت VIII	
۱۷	۲.۱ Multi-Layer Perceptron	
۱۷	۱.۲.۱ قسمت I	
۱۸	۲.۲.۱ قسمت II	
۲۱	۳.۲.۱ قسمت III	
۲۸	۴.۲.۱ قسمت IV	
۳۲	۵.۲.۱ قسمت V	

## فهرست تصاویر

۱۱	..... ماتریس هم‌بستگی برای حالت‌های مختلف	۱
۱۲	..... ماتریس هم‌بستگی نسبت به قیمت	۲
۱۴	..... نمودار توزیع آماری قیمت	۳
۱۵	..... نمودار قیمت برحسب فیچر با بیش‌ترین هم‌بستگی	۴
۱۹	..... نمودار مقایسه‌ای برخی بهینه‌سازها	۵
۲۲	..... تابع اتلاف Huber	۶
۲۹	..... نتایج مربوط به شبکه‌های با تعداد لایه مخفی مختلف	۷
۳۰	..... نتایج مربوط به بهینه‌سازهای مختلف	۸
۳۱	..... نتایج مربوط به توابع اتلاف مختلف	۹

## فهرست جداول

## پرسش ۴. Multi-Layer Perceptron

### ۱ پاسخ پرسش چهارم

#### توضیح پوشه کدهای Multi-Layer Perceptron

کدهای مربوط به این قسمت، علاوه بر پوشه محلی کدها در [این لینک](#) آورده شده است.

#### ۱.۱ آشنایی و کار با دیتاست (پیش‌پردازش)

##### ۱.۱.۱ قسمت I

در گام اول، مجموعه داده را روی [گوگل درایو](#) بارگذاری می‌کنیم. فراخوانی این فایل بدون نیاز به Mount کردن و استفاده از gdown در محیط گوگل کولب ممکن است با خطا مواجه شود که با استفاده از دستورات زیر مشکل حل می‌شود.

```
1 !pip install --upgrade --no-cache-dir gdown
2 !gdown 17KnUeG8_I9vgGU5mqym2hfFjnJPSEH-X
```

هم‌چنین می‌توانیم با فراخوانی دستورات زیر، امکان بارگذاری داده‌ها را در محیط کولب فراهم آوریم:

```
1 from google.colab import files
2 uploaded = files.upload()
```

برای خواندن فایل csv و فراخوانی تابع info از Pandas، از دستورات زیر استفاده می‌کنیم. دستور df.info() در زبان پایتون، یک دستور برای نمایش خلاصه‌ای از اطلاعات یک دیتافریم است. این دستور اطلاعاتی را مانند تعداد ردیف‌ها، تعداد ستون‌ها، نوع داده‌ای هر ستون، تعداد مقادیر نال و غیره را در مورد داده‌های موجود در دیتافریم به ما نشان می‌دهد. این دستور به طور خاص، اطلاعاتی را در مورد دیتافریم مانند نوع داده‌ها (float, int, object، و غیره)، تعداد مقادیر غیرنال در هر ستون و حجم کلی داده‌ها نشان می‌دهد. همچنین، اگر دیتافریم شامل ستون‌هایی با داده‌هایی از نوع datetime و یا تاریخ باشد، تعداد و مقدار این ستون‌ها نیز نمایش داده می‌شود. با استفاده از این دستور، می‌توانیم اطلاعات مهمی را در مورد دیتافریم مورد نظر به دست آوریم و با کمک این اطلاعات، به سادگی بتوانیم تحلیل‌های مورد نیاز خود را روی داده‌های خود انجام دهیم.

```
1 import pandas as pd
2 df = pd.read_csv("/content/CarPrice_Assignment.csv")
3 df.info()
```

نتیجه به صورت زیر خواهد بود:

```
1 <class 'pandas.core.frame.DataFrame'>
2 RangeIndex: 205 entries, 0 to 204
3 Data columns (total 26 columns):
4 #    Column                                Non-Null Count  Dtype
5  ...  ...
```

```

5  ---  -----  -----  -----
6  0    car_ID      205 non-null  int64
7  1    symboling    205 non-null  int64
8  2    CarName      205 non-null  object
9  3    fueltype     205 non-null  object
10 4    aspiration    205 non-null  object
11 5    doornumber    205 non-null  object
12 6    carbody       205 non-null  object
13 7    drivewheel    205 non-null  object
14 8    enginelocation 205 non-null  object
15 9    wheelbase     205 non-null  float64
16 10   carlength     205 non-null  float64
17 11   carwidth      205 non-null  float64
18 12   carheight     205 non-null  float64
19 13   curbweight    205 non-null  int64
20 14   enginetype    205 non-null  object
21 15   cylindernumber 205 non-null  object
22 16   enginesize    205 non-null  int64
23 17   fuelsystem    205 non-null  object
24 18   boreratio     205 non-null  float64
25 19   stroke        205 non-null  float64
26 20   compressionratio 205 non-null  float64
27 21   horsepower    205 non-null  int64
28 22   peakrpm       205 non-null  int64
29 23   citympg       205 non-null  int64
30 24   highwaympg    205 non-null  int64
31 25   price         205 non-null  float64
32 dtypes: float64(8), int64(8), object(10)
33 memory usage: 41.8+ KB

```

مشاهده کردیم که مجموعه داده مورد نظر ما شامل ۲۰۵ داده در ۲۶ ستون است. هم چنین برخی از ویژگی‌ها عددی و برخی غیر عددی هستند.

## ۲.۱.۱ قسمت II

در گام بعدی می‌خواهیم تعداد داده‌هایی که Nan هستند را برحسب هر ستون نمایش دهیم. دستور `df.isna()` در پایتون، برای شناسایی مقادیر خالی (Null) یا مقادیر NaN (Not a Number) در یک داده‌فریم (DataFrame) استفاده می‌شود. این دستور یک داده‌فریم را دریافت می‌کند و برای هر یک از سلول‌های آن مقدار True یا False برمی‌گرداند. مقدار True به معنی این است که مقدار آن سلول خالی (Null) یا NaN است و مقدار False به معنی این است که سلول دارای مقداری غیر خالی است. این دستور معمولاً برای شناسایی سلول‌هایی که دارای مقدار خالی هستند و انجام عملیات‌های مربوط به پر کردن سلول‌ها (imputation) یا حذف آن‌ها از داده‌فریم استفاده می‌شود. دستور `df.isnull()` در پایتون، همانند دستور `df.isna()` برای شناسایی مقادیر خالی در یک داده‌فریم استفاده می‌شود. در واقع این دو دستور معادل هستند و می‌توان از هر دو برای شناسایی مقادیر خالی در یک داده‌فریم استفاده کرد. با این توضیحات، برای هدف این سوال می‌توانیم از یکی از دو دستور زیر استفاده

کنیم:

```
1 df.isnull().sum()

1 for i in df.columns:
2     print('Number of NaN in',i,'=',df.isna().sum().sum())
```

نتیجه به صورت زیر خواهد بود:

```
1 car_ID          0
2 symboling       0
3 CarName         0
4 fueltype        0
5 aspiration       0
6 doornumber      0
7 carbody         0
8 drivewheel      0
9 enginelocation  0
10 wheelbase      0
11 carlength      0
12 carwidth       0
13 carheight      0
14 curbweight     0
15 enginetype      0
16 cylindernumber  0
17 enginesize     0
18 fuelsystem      0
19 boreratio      0
20 stroke         0
21 compressionratio 0
22 horsepower     0
23 peakrpm        0
24 citympg        0
25 highwaympg     0
26 price          0
27 dtype: int64
```

همان طور که از نتایج مشاهده می شود، هیچ ستونی دارای داده Nan نیست.

### ۳.۱.۱ قسمت III

برای هدف مدنظر این سوال دستوراتی تعریف می کنیم که کلمه اول در ستون CarName را در ستونی جدید با نام CompanyName ذخیره کند و ستون های CarName، ID\_car و symboling را حذف کند. برای اصلاح غلط های املایی یک مجموعه از نام های معتبر برای ستون CompanyName تعریف می کنیم و نام هایی که غیر از این لیست باشند را به نزدیک ترین نام موجود در لیست معتبر تبدیل کند.

```

1 import pandas as pd
2 import difflib
3
4 # list of valid car company names
5 valid_names = ['alfa-romero', 'audi', 'bmw', 'chevrolet', 'dodge', 'honda', 'isuzu', 'jaguar', '
    mazda', 'buick', 'mercury', 'mitsubishi', 'nissan', 'peugeot', 'plymouth', 'porsche', '
    renault', 'saab', 'subaru', 'toyota', 'volkswagen', 'vw', 'volvo']
6
7 # read the original csv file into a pandas dataframe
8 df = pd.read_csv('/content/CarPrice_Assignment.csv')
9
10 # extract the first word of each CarName into a new CompanyName column
11 df['CompanyName'] = df['CarName'].str.split().str[0]
12
13 # replace any invalid names with the nearest match from the list
14 df['CompanyName'] = df['CompanyName'].apply(lambda x: difflib.get_close_matches(x, valid_names, n
    =1)[0])
15
16 # reorder the columns so that CompanyName is the first column
17 df = df[['CompanyName', *df.columns[:-4]]]
18
19 # drop the CarName, car_ID, and symboling columns
20 df = df.drop(['CarName', 'car_ID', 'symboling'], axis=1)
21
22 # save the modified dataframe to a new csv file
23 df.to_csv('CarPrice_Assignment1.csv', index=False)

```

نتیجه را با نام جدید CarPrice\_Assignment1.csv ذخیره می‌کنیم که تمام خواسته‌های مدنظر سوال را برآورده کرده است.

#### ۴.۱.۱ قسمت IV

این کار را به حالت‌های مختلفی می‌توانیم انجام دهیم. در حالت اول کدگذاری را به صورت عددی انجام می‌دهیم. دستور `pd.factorize` در کتابخانه Pandas در زبان برنامه‌نویسی پایتون، برای تبدیل یک آرایه از مقادیر به فاکتورهای عددی استفاده می‌شود. فاکتورها به طور خاص در داده‌های دسته‌ای که به عنوان مقادیر داده‌ها از کلمات استفاده می‌شود، کاربرد دارند. این دستور مقادیر یک آرایه را به شکل یک tuple با دو عضو تبدیل می‌کند. عضو اول شامل فاکتورهای عددی است که به مقادیر ورودی نسبت داده شده‌اند و عضو دوم شامل لیست مقادیر یکتای موجود در آرایه ورودی است. بنابراین، در حالت اول از دستور زیر استفاده می‌کنیم که ابتدا فایل را می‌خواند، سپس ستون‌های غیر عددی را تشخیص می‌دهد و در نهایت به جای رشته‌های غیر عددی، عدد (۱، ۲ و...) جای‌گذاری می‌کند:

```

1 import pandas as pd
2
3 # read the csv file
4 df = pd.read_csv('/content/CarPrice_Assignment1.csv')
5

```



```

6 # iterate over each column in the dataframe
7 for col in df.columns:
8     # check if the column has a non-numeric data type
9     if df[col].dtype == 'object':
10         # use the pandas factorize() method to encode the values as integers
11         df[col] = pd.factorize(df[col])[0]
12
13 # save the updated dataframe to a new csv file
14 df.to_csv('CarPrice_Assignment2.csv', index=False)

```

اگر بخواهیم داده‌های هر ستون را هم نرمال کنیم می‌توانیم از این دستورات استفاده کنیم:

```

1 import pandas as pd
2
3 # load the csv file
4 df = pd.read_csv('/content/CarPrice_Assignment2.csv')
5
6 # normalize each column by its own maximum value
7 normalized_df = df.apply(lambda x: x / x.max(), axis=0)
8
9 # save the normalized dataframe to a new csv file
10 normalized_df.to_csv('CarPrice_Assignment3.csv', index=False)

```

آن‌چه که احتمالاً مقصود طراحان این سوال بوده است راهکار دوم و استفاده از دستور `pd.get_dummies` بوده است. دستور `pd.get_dummies` یکی از توابع کتابخانه Pandas در زبان برنامه‌نویسی پایتون است که برای تبدیل متغیرهای دسته‌ای یا Cat-egorical Variables به متغیرهای دودویی یا Binary Variables استفاده می‌شود. به عبارت دیگر، این دستور یک فرآیند برای تبدیل داده‌های دسته‌ای به داده‌های عددی را انجام می‌دهد. با فراخوانی تابع `pd.get_dummies` بر روی یک ستون از یک `DataFrame`، هر مقدار ممکن در آن ستون به یک ستون جدید تبدیل می‌شود و برای هر ردیف، مقدار ۱ در ستون متناظر با دسته‌بندی مربوطه و ۰ در ستون‌های دیگری که به دسته‌بندی‌های دیگر مربوط می‌شوند قرار می‌گیرد. این تبدیل می‌تواند بسیار مفید باشد، به عنوان مثال، در مدل‌سازی داده‌های دسته‌ای. به کمک تبدیل داده‌های دسته‌ای به داده‌های عددی، می‌توان به سادگی این داده‌ها را به عنوان ورودی به الگوریتم‌های یادگیری ماشین داد و با استفاده از آنها مدل‌هایی با دقت بالاتر و پیچیدگی کمتر ایجاد کرد. با این توضیحات، در حالت دوم از کدگذاری `onehot` استفاده می‌کنیم؛ اما نتایج را به شکل ستون‌های جداگانه ذخیره‌سازی می‌کنیم. این حالت مرسوم و استاندارد است؛ اما برای بررسی همبستگی ستون‌ها مناسب به نظر نمی‌رسد. مثلاً ما می‌خواهیم همبستگی ستون نام کارخانه سازنده ماشین با قیمت را بسنجیم و نه همبستگی هر کارخانه سازنده با قیمت را. دستورات مربوط به این حالت به شرح زیر است:

```

1 import pandas as pd
2
3 # Load CSV file
4 df = pd.read_csv('/content/CarPrice_Assignment1.csv')
5
6 # Loop through each column in the dataframe
7 for col in df.columns:
8     # Check if the column contains non-numeric values (i.e. objects)

```

```

9     if df[col].dtype == 'object':
10         # Convert non-numeric values to one-hot encoded columns
11         df = pd.concat([df, pd.get_dummies(df[col], prefix=col)], axis=1)
12         # Drop the original column
13         df.drop(columns=[col], inplace=True)
14
15 # Save result to a new CSV file
16 df.to_csv('CarPrice_Assignment4.csv', index=False)

```

در حالت سوم از کدگذاری onehot تک‌ستونه (به نوعی باینری) استفاده می‌کنیم. دستورات مربوط به آن به شرح زیر است:

```

1 import pandas as pd
2
3 # Load CSV file
4 df = pd.read_csv('/content/CarPrice_Assignment3.csv')
5
6 # Group columns by the characters before the first underscore in their headers
7 groups = {}
8 for col in df.columns:
9     prefix = col.split('_')[0]
10    if prefix not in groups:
11        groups[prefix] = []
12    groups[prefix].append(col)
13
14 # Merge columns within each group and remove original columns
15 for prefix, cols in groups.items():
16     if len(cols) > 1:
17         new_col = '_'.join([prefix] + [''.join(c.split(prefix+'_')[1:]) for c in cols])
18         df[new_col] = df[cols].apply(lambda x: ''.join(x.astype(str)), axis=1)
19         df.drop(columns=cols, inplace=True)
20
21 # Save result to a new CSV file
22 df.to_csv('CarPrice_Assignment5.csv', index=False)

```

## ۵.۱.۱ قسمت V

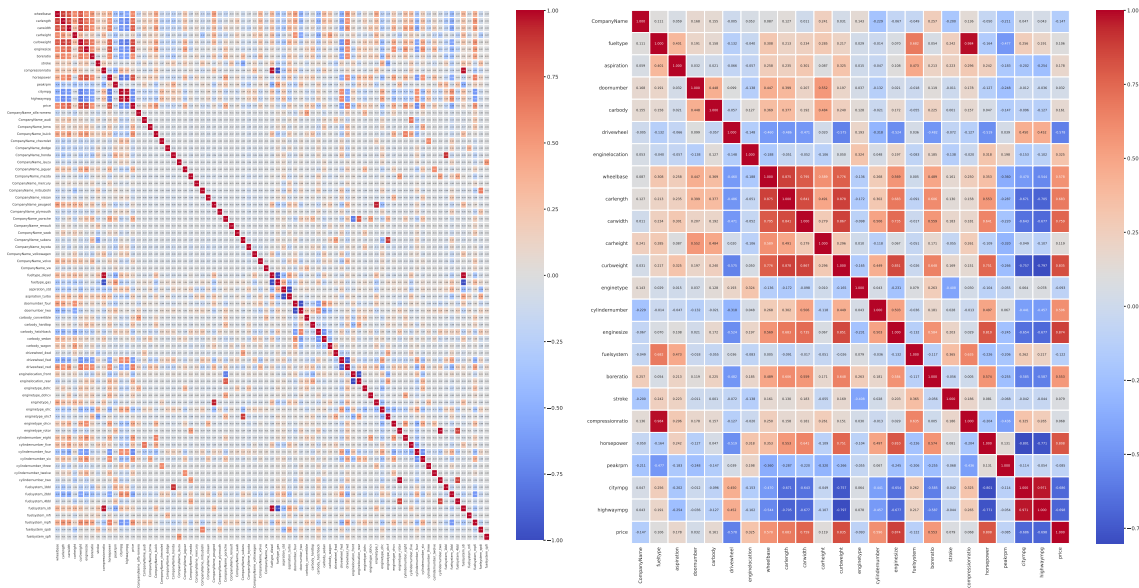
به این قسمت، با توجه به تمام حالاتی که در **قسمت IV** در نظر گرفتیم، پاسخ می‌دهیم. برای ساختن ماتریس هم‌بستگی بین متغیرهای داده‌ای موجود در یک دیتافریم، می‌توانیم از دستور `df.corr()` استفاده کنیم. ابتدا با تابع `corr()`، ماتریس هم‌بستگی بین تمام متغیرها محاسبه شده و در متغیر `CM` ذخیره می‌شود. سپس با استفاده از تابع `style.background_gradient()`، رنگ زمینه هر سلول جدول (ماتریس) هم‌بستگی بر اساس مقدار آن سلول تعیین می‌شود. ما از `'coolwarmcmap='` برای تعیین رنگ‌ها استفاده کرده‌ایم. دستور `3f. 'fmt='` برای نمایش اعداد تا سه رقم اعشار استفاده شده است. هم‌چنین قسمت `ticklabels` برای نمایش صحیح نام ستون‌ها در محور عمودی اضافه شده است. این کار برای نمایش در حالت دوم به‌صورت دیگری (با تعریف دوگان چرخشی) که در دفترچه‌کد مشخص است انجام شده است. از `'tightbbox_inches='` هم برای

نمایش صحیح ماتریس در فایل ذخیره‌شده به صورت پی‌دی‌اف استفاده شده است. دستوری که از آن استفاده کرده‌ایم به شرح زیر است.

```
1 import pandas as pd
2 import seaborn as sns
3 import matplotlib.pyplot as plt
4
5 # Read CSV file into DataFrame
6 df = pd.read_csv('/content/CarPrice_Assignment2.csv')
7
8 # Calculate correlation matrix
9 corr_matrix = df.corr()
10
11 # Create heatmap using seaborn
12 plt.figure(figsize=(25,25))
13 sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', linewidths=0.5, annot_kws={"size": 3}, fmt=
    '.3f', yticklabels=corr_matrix.columns)
14
15 # Adjust font size of annotations
16 plt.xticks(fontsize=8)
17 plt.yticks(fontsize=8)
18
19 # Adjust margins of PDF file
20 plt.savefig('PicS1.pdf', bbox_inches='tight')
```

نتایج در شکل ۱ نشان داده شده است. برای حالت اول (عددی) در قسمت IV نتایج به صورت تصویر ۱(آ) و برای حالت دوم در قسمت IV نتایج به صورت تصویر ۱(ب) خواهد بود. در ادامه دستورات زیر را می‌نویسیم تا مقادیر هم‌بستگی هرستون با متغیر قیمت را به صورت مرتب‌شده نمایش دهیم. نتایج برای حالات اول و دوم در شکل ۲ آورده شده است.

```
1 import pandas as pd
2 import seaborn as sns
3 import matplotlib.pyplot as plt
4
5 # Read CSV file into DataFrame
6 df = pd.read_csv('/content/CarPrice_Assignment4.csv')
7
8 # Select columns to include in correlation matrix
9 cols = df.columns.tolist()
10 cols.remove('price')
11
12 # Calculate correlation matrix
13 corr_matrix = df[cols].corrwith(df['price']).sort_values(ascending=False)
14
15 # Create heatmap using seaborn
16 plt.figure(figsize=(2,30))
```



(ب) حالت دوم

(ا) حالت اول

شکل ۱: ماتریس هم‌بستگی برای حالت‌های مختلف.

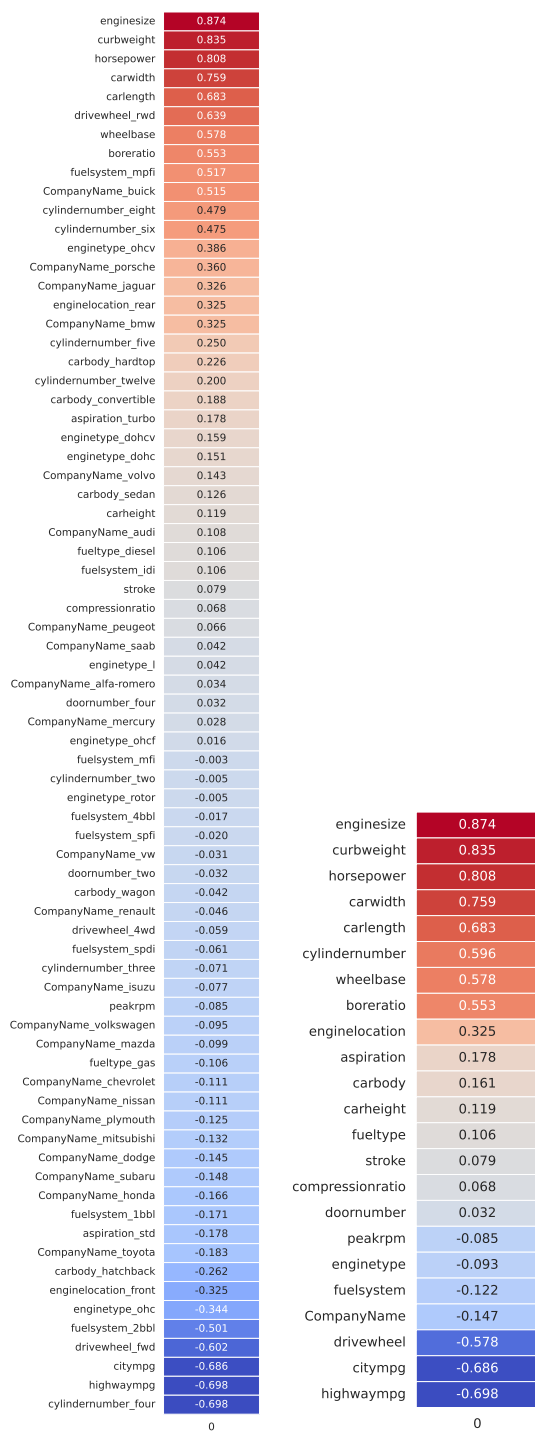
```

17 sns.heatmap(corr_matrix.to_frame(), annot=True, cmap='coolwarm', linewidths=0.5, annot_kws={"size
   ": 12}, fmt='.3f', cbar=False)
18
19 # Rotate x-axis tick labels to be horizontal
20 plt.xticks(rotation=0)
21
22 # Adjust font size of annotations
23 plt.xticks(fontsize=12)
24 plt.yticks(fontsize=12)
25
26 # Adjust margins of PDF file
27 plt.tight_layout()
28 plt.savefig('priceCM2.pdf', bbox_inches='tight')

```

## ۶.۱.۱ قسمت VI

برای هدف مدنظر این قسمت دستوراتی را نوشته‌ایم که با خواندن فایل مربوط به داده‌ها، در نمودار اول و با استفاده از تابع `hist`، توزیع قیمت خودروها را با بازه‌های ۵۰ تایی نمایش می‌دهد. محور افقی قیمت خودروها و محور عمودی تعداد خودروهایی است که در هر بازه قیمتی وجود دارند. هم‌چنین در نمودار دیگری و با استفاده از تابع `boxplot`، قیمت خودروها به صورت یک جعبه نمایش داده می‌شود. جهت این نمودار را پارامتر `vert` تغییر می‌کند و عرض آن با پارامتر `widths` تنظیم می‌شود. نمودار دوم شامل اطلاعاتی مانند میانه، چارک‌ها و اکستریم‌های قیمتی خودروهاست. در ادامه و با استفاده از دستور `pd.cut`، قیمت خودروها را به بازه‌های ۸۰۰۰ دلاری تقسیم کردیم و نتایج نشان می‌دهد که بیش‌تر خودروها در بازه قیمتی ۸۰۰۰ تا ۱۶۰۰۰



(ب) حالت دوم

(آ) حالت اول

شکل ۲: ماتریس هم‌بستگی نسبت به قیمت.

دلار هستند. نتایج مربوط به دو نمودار در شکل ۳ آورده شده و دستورات به شرح زیر است.

```
import pandas as pd
```

```

2 import matplotlib.pyplot as plt
3
4 # Load the CSV file into a pandas dataframe
5 df = pd.read_csv('/content/CarPrice_Assignment4.csv')
6
7 # Plot the distribution of the 'price' column as a histogram
8 plt.figure(figsize=(8, 6)) # Set the size of the figure
9 plt.subplot(2, 1, 1) # Create the first subplot
10 plt.hist(df['price'], bins=50, color='blue')
11 plt.xlabel('Price')
12 plt.ylabel('Count')
13 plt.title('Distribution of Prices')
14
15 # Plot a boxplot of the 'price' column
16 plt.subplot(2, 1, 2) # Create the second subplot
17 plt.boxplot(df['price'], vert=False, widths=0.7)
18 plt.xlabel('Price')
19 plt.title('Boxplot of Prices')
20
21 plt.tight_layout() # Automatically adjust subplot parameters
22
23 # Save the plot as a PDF file with a fit margin
24 plt.savefig('dist2.pdf', bbox_inches='tight')
25 plt.show()
26
27 # Define the bins for the price ranges
28 price_ranges = pd.cut(df['price'], bins=range(0, 160000, 8000))
29 # Calculate the count of prices in each range
30 price_counts = price_ranges.value_counts().sort_index()
31 # Display the price range counts
32 print(price_counts)

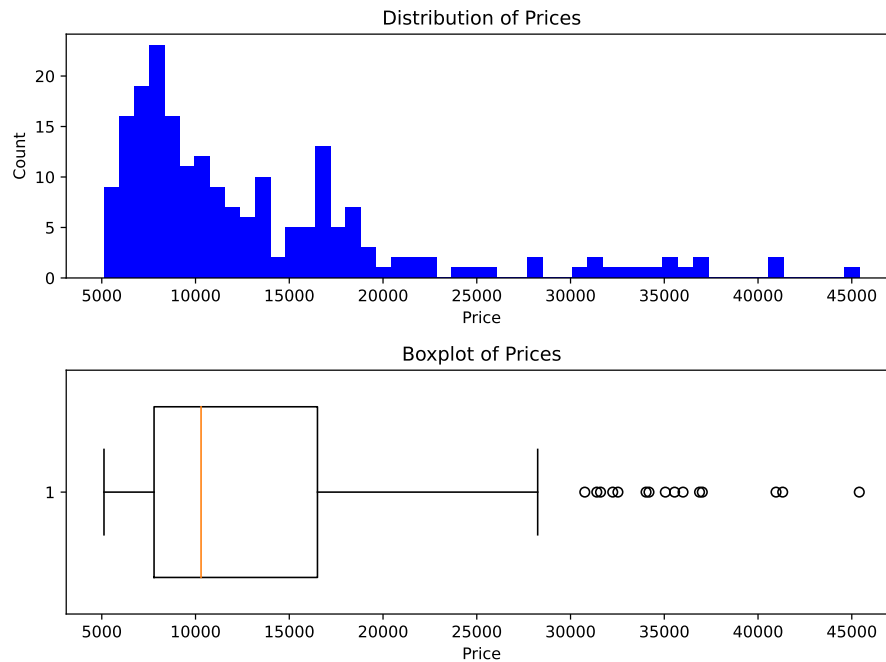
```

از قسمت V می‌دانیم که قیمت با فیچر enginesize بیش‌ترین هم‌بستگی را دارد. برای رسم نمودار از seaborn استفاده کرده و یک نمودار اسکاتر از رابطه بین قیمت خودرو و حجم موتور رسم می‌کنیم. با رسم این نمودار و با استفاده از تابع `sns.regplot`، میانگین خطی بین دو متغیر را به‌صورت خودکار به نمودار اضافه می‌کنیم. نمودار خروجی این کد نشان‌دهنده توزیع نقاط بر روی صفحه است که هر نقطه نمایانگر یک خودرو و موقعیت آن بر حسب حجم موتور و قیمت آن خودرو است. به‌طور خاص، مشاهده می‌شود که با افزایش حجم موتور، قیمت خودرو نیز افزایش می‌یابد و این رابطه با یک خط تقریباً خطی نشان داده شده است. همچنین با مشاهده توزیع نقاط در بخش پایین نمودار می‌توان تخمینی از دامنه قیمت خودروها برای هر حجم موتور داد. نتایج مربوط به این نمودار در شکل ۴ نمایش داده شده و دستورات آن به شرح زیر است:

```

1 import pandas as pd
2 import seaborn as sns
3 import matplotlib.pyplot as plt
4
5 # Load the CSV file into a pandas dataframe

```



شکل ۳: نمودار توزیع آماری قیمت.

```

6 df = pd.read_csv('/content/CarPrice_Assignment4.csv')
7
8 # Create a scatter plot of 'price' versus 'engine size'
9 sns.regplot(x=df['engine size'], y=df['price'])
10 plt.xlabel('Engine Size')
11 plt.ylabel('Price')
12 plt.title('Price vs. Engine Size Scatter Plot')
13
14 # Save the plot as a PDF file with a fit margin
15 plt.savefig('pricevsenginesize2.pdf', bbox_inches='tight')
16
17 plt.show()

```

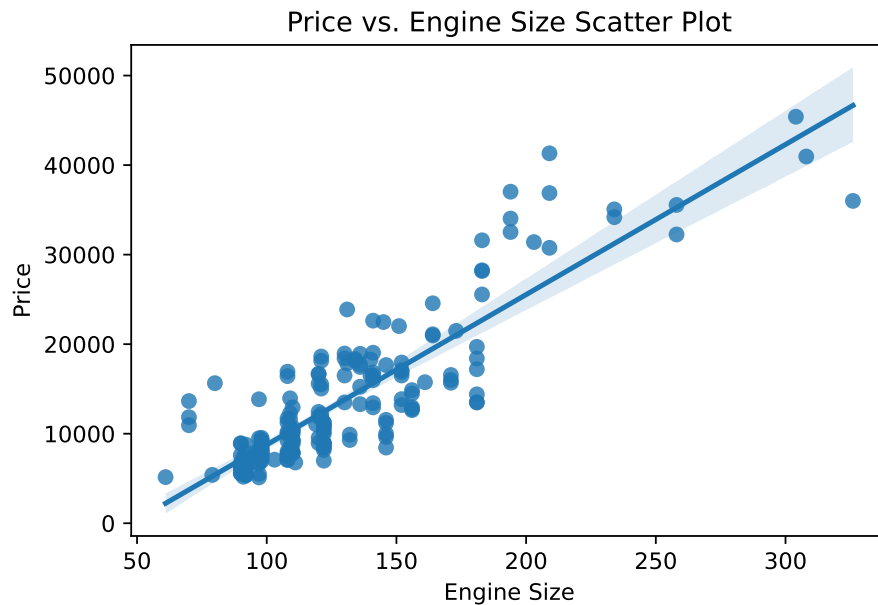
## ۷.۱.۱ قسمت VII

برای هدف مورد نظر این سوال از دستوراتی استفاده می‌کنیم که در آن با فراخوانی کتابخانه‌های مدنظر و خواندن مجموعه داده، ستون قیمت به عنوان برچسب یا هدف در نظر گرفته می‌شود و بقیه ستون‌ها به عنوان داده. سپس ضمن مخلوط کردن داده‌ها دو مجموعه داده‌ی آموزشی و آزمایشی با نسبت گفته شده در صورت سوال ایجاد می‌کنیم. در نهایت تمام مجموعه‌ها را ذخیره می‌کنیم. دستورات به شرح زیر است:

```

1 import pandas as pd
2 from sklearn.model_selection import train_test_split

```



شکل ۴: نمودار قیمت برحسب فیچر با بیش‌ترین هم‌بستگی.

```

3
4 # Read data from CSV file into a Pandas dataframe
5 data = pd.read_csv('/content/CarPrice_Assignment4.csv')
6
7 # Split the data into train and test sets
8 train_data, test_data, train_label, test_label = train_test_split(data.drop(['price'], axis=1),
9                             data['price'], test_size=0.15, random_state=42)
10
11 # Save train and test data to new CSV files
12 train_data.to_csv('train_data.csv', index=False)
13 test_data.to_csv('test_data.csv', index=False)
14
15 # Save train and test labels to new CSV files
16 train_label.to_csv('train_label.csv', index=False)
17 test_label.to_csv('test_label.csv', index=False)

```

### ۸.۱.۱ قسمت VIII

در این قسمت دستوراتی نوشته‌ایم که با استفاده از MinMaxScaler داده‌ها را بین صفر و یک مقیاس کنیم. در این عملیات توجه داریم که از اطلاعات داده‌های آموزشی استفاده نکنیم؛ چراکه، باعث نشت اطلاعات می‌گردد. نشت اطلاعات به معنای انتقال اطلاعات از داده‌های تستی به مدل یادگیری است که باعث می‌شود مدل بهترین عملکرد را بر روی داده‌های تستی نشان دهد؛ اما درواقع مدل تعمیم‌پذیری و عمل‌کرد خوبی ندارد. دستورات مربوط به این قسمت به شرح زیر است:



```

1 import pandas as pd
2 from sklearn.model_selection import train_test_split
3 from sklearn.preprocessing import MinMaxScaler
4
5 # read csv file into a pandas dataframe
6 data = pd.read_csv('/content/CarPrice_Assignment4.csv')
7
8 # extract label column as y
9 y = data['price']
10
11 # extract all other columns as X
12 X = data.drop(columns=['price'])
13
14 # split data into train and test sets with a 85/15 split
15 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.15, random_state=42)
16
17 # scale the training data
18 scaler = MinMaxScaler()
19 X_train_scaled = scaler.fit_transform(X_train)
20
21 # scale the test data using the same scaler used for training data
22 X_test_scaled = scaler.transform(X_test)
23
24 # save train and test data/label in new files
25 pd.DataFrame(X_train_scaled).to_csv('Xtrain.csv', index=False)
26 y_train.to_csv('ytrain.csv', index=False)
27 pd.DataFrame(X_test_scaled).to_csv('Xtest.csv', index=False)
28 y_test.to_csv('ytest.csv', index=False)

```

برای اضافه کردن مجموعه اعتبارسنجی به مجموعه‌های آموزش و آزمایش نیز دستورات زیر را نوشته‌ایم:

```

1 import pandas as pd
2 from sklearn.model_selection import train_test_split
3 from sklearn.preprocessing import MinMaxScaler
4
5 # Read the CSV file
6 data = pd.read_csv('/content/CarPrice_Assignment4.csv')
7
8 # Separate the label and data
9 label = df['price']
10 data = df.drop('price', axis=1)
11
12 # Remove the header from the variables
13 data = data.values
14 label = label.values
15

```

```

16 # Split the data into train/validation/test sets
17 train_data, test_data, train_label, test_label = train_test_split(data, label, test_size=0.15,
    random_state=42)
18 train_data, val_data, train_label, val_label = train_test_split(train_data, train_label,
    test_size=0.15, random_state=42)
19
20 # Scale the data using MinMaxScaler
21 scaler = MinMaxScaler()
22 train_data = scaler.fit_transform(train_data)
23 val_data = scaler.transform(val_data)
24 test_data = scaler.transform(test_data)
25
26 # Save the train/validation/test data/label to new files
27 pd.DataFrame(train_data).to_csv('/content/finaltrain_data.csv', index=False, header=False)
28 pd.DataFrame(val_data).to_csv('/content/finalval_data.csv', index=False, header=False)
29 pd.DataFrame(test_data).to_csv('/content/finaltest_data.csv', index=False, header=False)
30 pd.DataFrame(train_label).to_csv('/content/finaltrain_label.csv', index=False, header=False)
31 pd.DataFrame(val_label).to_csv('/content/finalval_label.csv', index=False, header=False)
32 pd.DataFrame(test_label).to_csv('/content/finaltest_label.csv', index=False, header=False)

```

## Multi-Layer Perceptron ۲.۱

### ۱.۲.۱ قسمت I

برای این قسمت، مجموعه‌دستوراتی را می‌نویسیم که شامل تعریف سه مدل شبکه عصبی چندلایه است. هر مدل شامل چند لایه خطی است که با استفاده از تابع فعال‌ساز غیرخطی ReLU به صورت پشت سر هم بهم متصل شده‌اند. تعریف هریک از این سه مدل به صورت یک کلاس از `nn.Module` پایتورچ انجام گرفته است. این سه مدل در تعداد لایه‌های پنهان با یکدیگر متفاوت هستند. تابع سازنده هر مدل، ورودی‌های لازم برای تعریف مدل را دریافت می‌کند و لایه‌های مورد نظر را با استفاده از `nn.Linear` تعریف می‌کند. هر مدل، یک تابع پیش‌بینی در مسیر رو به جلو دارد که با دریافت ورودی  $x$ ، ابتدا خروجی هر لایه خطی با اعمال تابع فعال‌ساز محاسبه می‌شود و سپس خروجی نهایی مدل برای ورودی  $x$  برگردانده می‌شود. دستورات به شرح زیر است:

```

1 # Define the MLP models
2 class MLP1(nn.Module):
3     def __init__(self, input_dim, hidden_dim, output_dim):
4         super(MLP1, self).__init__()
5         self.fc1 = nn.Linear(input_dim, hidden_dim)
6         self.fc2 = nn.Linear(hidden_dim, output_dim)
7
8     def forward(self, x):
9         x = torch.relu(self.fc1(x))
10        x = self.fc2(x)
11        return x

```

```

12
13 class MLP2(nn.Module):
14     def __init__(self, input_dim, hidden_dim1, hidden_dim2, output_dim):
15         super(MLP2, self).__init__()
16         self.fc1 = nn.Linear(input_dim, hidden_dim1)
17         self.fc2 = nn.Linear(hidden_dim1, hidden_dim2)
18         self.fc3 = nn.Linear(hidden_dim2, output_dim)
19
20     def forward(self, x):
21         x = torch.relu(self.fc1(x))
22         x = torch.relu(self.fc2(x))
23         x = self.fc3(x)
24         return x
25
26 class MLP3(nn.Module):
27     def __init__(self, input_dim, hidden_dim1, hidden_dim2, hidden_dim3, output_dim):
28         super(MLP3, self).__init__()
29         self.fc1 = nn.Linear(input_dim, hidden_dim1)
30         self.fc2 = nn.Linear(hidden_dim1, hidden_dim2)
31         self.fc3 = nn.Linear(hidden_dim2, hidden_dim3)
32         self.fc4 = nn.Linear(hidden_dim3, output_dim)
33
34     def forward(self, x):
35         x = torch.relu(self.fc1(x))
36         x = torch.relu(self.fc2(x))
37         x = torch.relu(self.fc3(x))
38         x = self.fc4(x)
39         return x

```

## ۲.۲.۱ قسمت II

در کارهای رگرسیون به دنبال بهترین مدلی هستیم که بیشترین دقت را در پیش‌بینی مقدار یک متغیر وابسته بر اساس متغیرهای مستقل دیگر داشته باشد. اما پیدا کردن بهترین مدل می‌تواند به دلیل پیچیدگی داده‌ها کاری زمان‌بر و سخت باشد. در این جا به برخی بهینه‌گرها و توابع اتلاف مناسب برای این کار اشاره می‌کنیم. برای انتخاب دو بهینه‌ساز اول از اطلاعات معروف موجود در شکل ۵ استفاده کردیم و الگوریتم‌های بهینه‌سازی Adam<sup>۱</sup> و Adagrad<sup>۲</sup> را برگزیدیم.

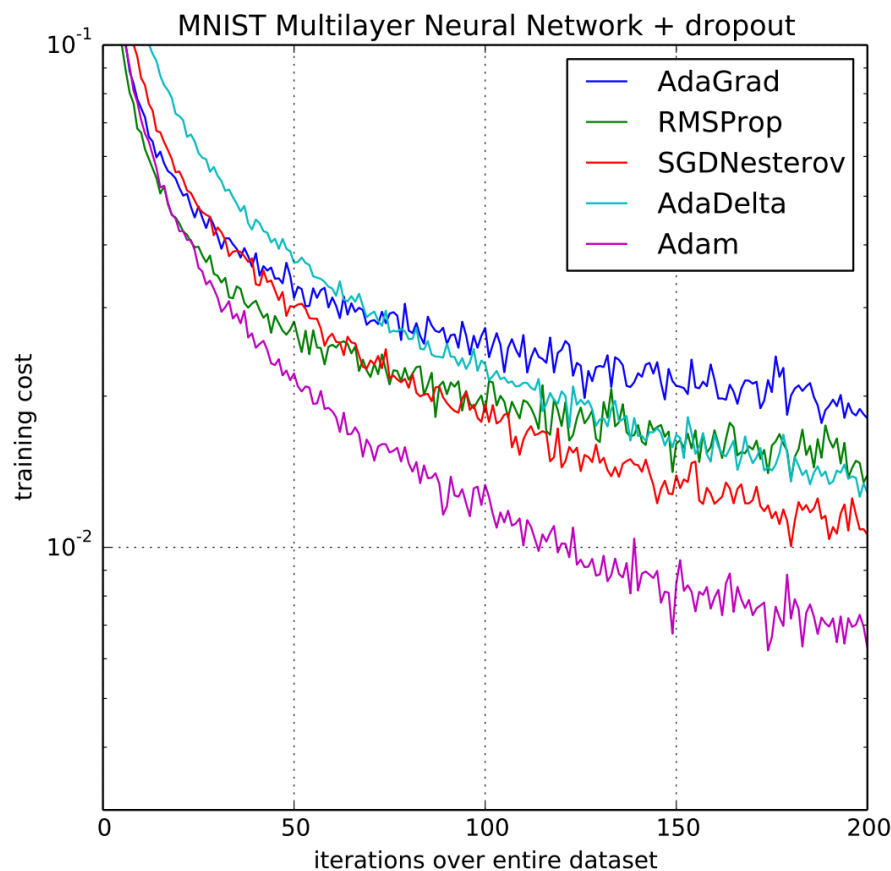
- بهینه‌ساز Adam: این الگوریتم، یک الگوریتم بهینه‌سازی انباشت گرادینان و نسخه تعمیم‌یافته الگوریتم SGD است که در شبکه‌های عصبی عمیق به خوبی کار می‌کند. این الگوریتم با ترکیب دو مفهوم بسیار قدرتمند AdaGrad<sup>۳</sup> و RMSProp<sup>۴</sup> طراحی شده است و در مقایسه با برخی بهینه‌سازهای دیگر سرعت هم‌گرایی بالا و دقت بیش‌تری در پیش‌بینی داده‌ها دارد.

<sup>۱</sup>Adaptive moment estimation

<sup>۲</sup>Adaptive Gradient Algorithm (Adagrad)

<sup>۳</sup>Adaptive Gradient Algorithm

<sup>۴</sup>Root Mean Square Propagation



شکل ۵: نمودار مقایسه‌ای برخی بهینه‌سازها (لینک).

از مریت‌های این بهینه‌ساز می‌توان به آسانی پیاده‌سازی، محاسبات بهینه، نیاز به حافظه کم، عدم تغییر در برابر مقیاس‌دهی مجدد قطری گرادیان‌ها، مناسب بودن برای داده‌ها و پارامترهای بزرگ، مناسب بودن برای مسائل دارای گرادیان‌های بسیار نویزدار یا نامتراکم، و نیاز به تنظیم پارامتر اندک اشاره کرد. در این الگوریتم بهینه‌یازی هر نرخ یادگیری برای پارامترهای مختلف از گشتاورهای اول و دوم گرادیان‌ها محاسبه می‌شود و در آن به جای انطباق نرخ‌های یادگیری پارامترها فقط براساس میانگین گشتاور اول (مانند الگوریتم RMSProp) از میانگین گشتاور دوم گرادیان‌ها هم استفاده می‌شود. این الگوریتم با چهار پارامتر پیکربندی می‌شود: آلفا یا  $\alpha$  که نرخ یادگیری یا طول گام است و نسبتی است که اوزان بر اساس آن به‌روز می‌شوند. هرچه آلفا بزرگ‌تر باشد یادگیری اولیه سریع‌تر خواهد بود.  $\beta_1$  نرخ فروپاشی نمایی برای تخمین‌های گشتاور اول را تعیین می‌کند.  $\beta_2$  هم نرخ فروپاشی نمایی برای تخمین‌های گشتاور دوم را تعیین می‌کند. این مقدار در مسائلی با گرادیان‌های نزدیک به یک در نظر گرفته شود. و در نهایت  $\epsilon$  عددی است بسیار کوچک که برای جلوگیری از تقسیم بر صفر در پیاده‌سازی‌ها در نظر گرفته شده است. آقای Andrej Karpathy در یکی از پست‌های وبلاگ خود اشاره کرده بود که نرخ یادگیری 0.0003 معمولاً مناسب است که ما با شروع از این مقدار و ایجاد تغییرات جزئی مقدار بهینه برای کاربرد خود را پیدا خواهیم کرد.

- بهینه‌ساز **Adagrad**: این الگوریتم، یک الگوریتم بهینه‌سازی است که به صورت خاص برای مسائل یادگیری ماشین طراحی شده است. این الگوریتم تلاش می‌کند برای بهبود کارایی فرآیند یادگیری با بهبود روش انتخاب شدن نرخ یادگیری،

به ویژه برای داده های بزرگ و پیچیده. اولین چیزی که باید در مورد این الگوریتم بدانیم، این است که این الگوریتم نرخ یادگیری را برای هر پارامتر به طور مستقل از سایر پارامترها محاسبه می کند. به عبارت دیگر، نرخ یادگیری برای هر پارامتر متفاوت است و با توجه به رفتار هر پارامتر در طول فرآیند یادگیری تغییر می کند. روش کار این الگوریتم به این صورت است که در هر مرحله، برای هر پارامتر از یک نرخ یادگیری جدید استفاده می شود که بر اساس سابقه گرادیان های مربوط به آن پارامتر محاسبه می شود. به عبارت دیگر، اگر گرادیان یک پارامتر در گام های قبلی بیشتر بوده باشد، نرخ یادگیری برای آن پارامتر در گام بعدی کاهش پیدا می کند و برعکس، اگر گرادیان کوچکتر باشد، نرخ یادگیری برای آن پارامتر در این گام بعدی افزایش پیدا می کند. به عنوان مثال، فرض کنید در گام اول، گرادیان یک پارامتر برابر با ۱۰ بوده است. در این صورت، نرخ یادگیری برای آن پارامتر در گام بعدی کاهش می یابد. اگر در گام دوم، گرادیان همان پارامتر برابر با ۵ باشد، نرخ یادگیری برای آن پارامتر افزایش پیدا می کند. این الگوریتم برای مقابله با چالش هایی مانند پارامترهای بسیار زیاد و تغییرات ناگهانی در گرادیان ها طراحی شده است. محاسبه ی گرادیان ها در هر مرحله از بهینه سازی، هزینه بر است و برای دیتاست های بزرگ و پارامترهای بیشمار، این هزینه می تواند بسیار زیاد شود. بنابراین، Adagrad با کاهش نرخ یادگیری برای پارامترهایی که بیشترین تغییر را در گرادیان دارند، به این مشکل راه حل ارائه می دهد. الگوریتم Adagrad برای هر پارامتر  $w$  یک مقدار  $r$  را نگهداری می کند که مجموع مربعات گرادیان های قبلی را نگهداری می کند. در ابتدا، این مقدار برای هر پارامتر را صفر می گذاریم. بعد از هر بار به روزرسانی، مقدار  $r$  برای هر پارامتر  $w$  به صورت زیر به روزرسانی می شود:  $r \leftarrow r + g^2$ . که  $g$  گرادیان جدید محاسبه شده است. در واقع،  $r$  میزان تجمع گرادیان مربعاتی است. سپس، نرخ یادگیری برای هر پارامتر با استفاده از فرمول زیر محاسبه می شود:  $\eta' = \frac{\eta}{\sqrt{r+\epsilon}}$ . که در اینجا،  $\eta$  نرخ یادگیری اولیه است و  $\epsilon$  یک عدد کوچک مثبت است که برای جلوگیری از تقسیم بر صفر در صورتی که  $r$  برای یک پارامتر خیلی کوچک باشد، استفاده می شود. بعد از محاسبه ی  $\eta'$ ، مقدار پارامتر جدید برای هر پارامتر با استفاده از فرمول زیر به روزرسانی می شود:  $w \leftarrow w - \eta' \cdot g$ . که  $g$  گرادیان جدید است. با این روش، نرخ یادگیری به شکل مستقل از تعداد اپوک ها و درون یک اپوک، به شکل خودکار تعیین می شود و بهترین نرخ یادگیری برای هر پارامتر با توجه به تاریخچه ی گرادیان های قبلی به دست می آید. همچنین، میزان تاثیر گرادیان هایی که بیشترین تغییر را دارند، افزایش می یابد و می تواند کمک کند تا به دنبال تغییرات ناگهانی در مسیر بهینه سازی، الگوریتم به دنبال آن ها بیشتر بگردد. Adagrad به خوبی در بسیاری از الگوریتم های یادگیری ماشین مانند شبکه های عصبی و مدل های خطی عملکرد بسیار خوبی داشته و باعث بهبود سرعت و دقت آن ها می شود. اما یکی از مشکلات این الگوریتم، افزایش مجموع مربعات گرادیان ها در طول زمان است که باعث کاهش نرخ یادگیری می شود. برای حل این مشکل، الگوریتم های بهینه سازی دیگری مانند RMSProp و Adam پیشنهاد شده اند.

توابع اتلاف یا همان توابع هزینه در مسائل رگرسیون به عنوان معیاری برای اندازه گیری خطا و اختلاف پیش بینی ها و مقادیر واقعی استفاده می شوند. در زیر دو تابع اتلاف برای کاربردهای رگرسیون را معرفی می کنیم:

#### • تابع اتلاف میانگین مربعات (Mean Squared Error Loss):

این تابع اتلاف یکی از معروف ترین توابع اتلاف در مسائل رگرسیون است. این تابع برای اندازه گیری خطا و اختلاف پیش بینی ها و مقادیر واقعی استفاده می شود. در این تابع، فاصله بین پیش بینی شده توسط مدل ( $\hat{y}_i$ ) و مقدار واقعی داده ( $y_i$ ) با استفاده از مربع این فاصله محاسبه می شود. سپس میانگین این مربعات برای تمامی داده ها محاسبه می شود. برای  $n$  داده، تابع اتلاف میانگین مربعات به صورت زیر تعریف می شود:

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

عامل  $\frac{1}{n} \sum$  که در تمامی توابع اتلاف وجود دارد برای میانگین‌گیری از تمام نمونه‌ها به کار می‌رود و نکته خاصی ندارد. قسمت بعدی که یک عامل مربعی است اهمیت دارد. در این تابع، هرچه که فاصله بین پیش‌بینی شده توسط مدل و مقدار واقعی برای یک داده بیشتر باشد، مربع این فاصله نیز بزرگ‌تر خواهد بود و به تبع آن، مقدار تابع اتلاف نیز بیشتر خواهد بود. به همین دلیل، تابع اتلاف میانگین مربعات معمولاً برای مسائل رگرسیون استفاده می‌شود. استفاده از این تابع اتلاف در الگوریتم‌های یادگیری ماشین، مدل را تشویق می‌کند که پارامترهای خود را طوری تغییر دهد که میانگین خطاهای مربوط به پیش‌بینی‌هایش را کمینه کند. در واقع، هدف این تابع این است که پارامترهای مدل را بهینه کند تا پیش‌بینی‌های آن نزدیک‌تر به مقادیر واقعی باشد.

از مزایای این تابع اتلاف می‌توان به رواج در کارهای متعدد، سادگی در پیاده‌سازی و بهینه‌سازی (مشتق‌گیری ساده) اشاره کرد. از معایب این تابع اتلاف می‌توان به حساسیت بالا به Outlierها اشاره کرد. مثلاً اگر داده‌ای پرتی داشته باشیم خطی که برازش را انجام می‌دهد به نوعی برای به دست آوردن دل آن داده‌ها خط را جابجا می‌کند؛ اما ما دوست نداریم این اتفاق بیافتد. دلیل این اتفاق وجود عامل مرتبه دوم است. بنابراین، اختلاف فاصله اهمیت بالایی پیدا می‌کند. این اتفاق می‌تواند منجر به به‌روزرسانی وحشتناک گرادیان و پارامترها شود.

- تابع اتلاف L1 (Mean Absolute Error Loss): این تابع اتلاف نیز یکی دیگر از توابع اتلاف معمول در مسائل رگرسیون است. در این تابع، فاصله بین پیش‌بینی شده توسط مدل ( $\hat{y}_i$ ) و مقدار واقعی داده ( $y_i$ ) با استفاده از مقدار مطلق این فاصله محاسبه می‌شود. سپس میانگین این فواصل مطلق برای تمامی داده‌ها محاسبه می‌شود. برای  $n$  داده، تابع اتلاف L1 به صورت زیر تعریف می‌شود:

$$MAE = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

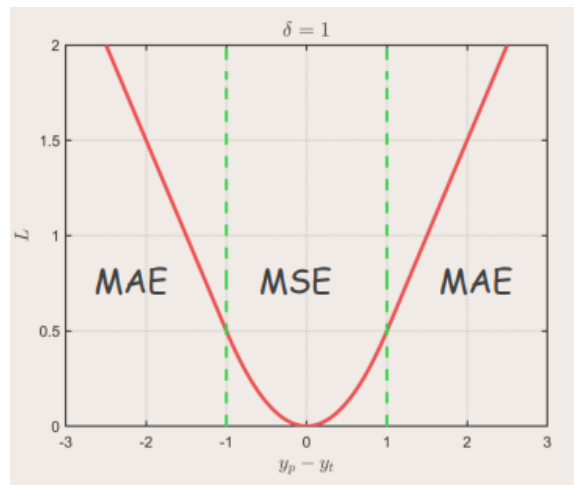
در این تابع، به عکس تابع MSE که در آن مربع فاصله بین پیش‌بینی شده و واقعی استفاده شده بود، در تابع MAE از فاصله مطلق استفاده شده است. به همین دلیل، تابع MAE معمولاً در حالت‌هایی مفید است که نیاز به حساسیت به انحراف‌های بزرگ از پیش‌بینی داریم. در الگوریتم‌های یادگیری ماشین، استفاده از تابع اتلاف L1 می‌تواند باعث شود که مدل، پارامترهای خود را طوری تغییر دهد که میانگین فاصله مطلق بین پیش‌بینی‌ها و مقادیر واقعی کمینه شود. به دلیل اینکه تابع MAE حساسیت کمتری به پرت‌ها دارد، ممکن است بهتر از تابع MSE برای مواجهه با داده‌های پرت و نویزی عمل کند.

از مزایای این تابع اتلاف می‌توان به سادگی و حساسیت کم به Outlierها اشاره کرد؛ چراکه، عامل درجه دوم تعدیل شده است. اما از معایب این تابع اتلاف می‌توان به بهینه‌سازی پیچیده‌تر (مثلاً مشتق‌ناپذیری در نقطه صفر) اشاره کرد. برای حل برخی مشکلات اشاره‌شده می‌توان از تابع اتلاف Huber استفاده کرد (رابطه ۱ و شکل ۶):

$$L = \begin{cases} 0.5 (y_p - y_t)^2 & |y_p - y_t| < \delta \\ \delta * (|y_p - y_t| - 0.5 * \delta) & \text{otherwise} \end{cases} \quad (1)$$

### ۳.۲.۱ قسمت III

معیار R2 Score (Coefficient of Determination) یکی از معیارهای مهم است که در تحلیل رگرسیون استفاده می‌شود. این معیار، نشان می‌دهد که چه میزان از تغییرات متغیر وابسته توسط متغیرهای مستقل قابل پیش‌بینی است. در واقع، Score R2 از



شکل ۶: تابع اتلاف Huber.

۰ تا ۱ بین می‌گیرد و نشان می‌دهد که چه میزان از تغییرات متغیر وابسته توسط متغیرهای مستقل مورد پوشش قرار گرفته است. عدد صفر به معنای عدم توانایی مدل در پیش‌بینی و عدد یک به معنای پیش‌بینی دقیق تمامی مقادیر است. به عنوان مثال، فرض کنید که مدل رگرسیونی شما با  $R^2$  Score برابر با ۰.۸۰ مدل شده است. این بدان معنی است که متغیرهای مستقل که در مدل شما استفاده شده‌اند، حدود ۸۰ درصد از تغییرات متغیر وابسته را توضیح می‌دهند و می‌توانند آن را پیش‌بینی کنند. تحلیل  $R^2$  Score به شکل زیر است:

- $R^2$  Score بالاتر بهتر است، به این معنی که مدل شما قادر به پوشش بیشتر تغییرات متغیر وابسته است.
- اگر  $R^2$  Score بسیار کم باشد، بهتر است بررسی کنید که آیا متغیرهای مستقل شما به درستی انتخاب شده‌اند و یا آیا مدل رگرسیونی شما به درستی پیاده‌سازی شده است.
- اگر  $R^2$  Score بسیار بالا باشد، بهتر است بررسی کنید که آیا مدل شما به درستی برای پیش‌بینی ارزش‌های جدید کاربردی است یا خیر.

همچنین، نکته مهم دیگری که باید در نظر داشته باشید این است که  $R^2$  Score فقط می‌تواند برای تحلیل رگرسیون استفاده شود و بنابراین  $R^2$  Score فقط در مورد تحلیل رگرسیون قابل استفاده است و برای سایر مدل‌های یادگیری ماشین، مانند الگوریتم‌های دسته‌بندی، معیارهای دیگری وجود دارد که می‌توانیم استفاده کنیم. همچنین، باید توجه داشت که استفاده از  $R^2$  Score به تنهایی برای انتخاب بهترین مدل رگرسیونی مناسب نیست. به عنوان مثال، در مقایسه دو مدل رگرسیون، هر کدام با  $R^2$  Score بالایی، می‌توانند دارای سایر خطاهای آماری باشند، از جمله ضرایب بیش از حد بزرگ یا کوچک، رفتارهای نامطلوب در برخی نقاط داده‌ها و یا نقص‌های دیگری. بنابراین، بهتر است همواره با توجه به معیارهای دیگری مانند MSE و MAE، مدل‌های مختلف را مقایسه کرد و بهترین مدل را برای هر مسئله خاص انتخاب کرد. در کل،  $R^2$  Score یک معیار مهم و کاربردی در تحلیل رگرسیون است که می‌تواند به تحلیل و پیش‌بینی تغییرات متغیر وابسته کمک کند. اما برای انتخاب بهترین مدل رگرسیونی، باید از معیارهای دیگری نیز استفاده کنیم و در کنار آن‌ها، معیار  $R^2$  Score را نیز مورد بررسی قرار دهیم. برای محاسبه مقدار  $R^2$  Score در تحلیل رگرسیون، ابتدا باید مقادیر پیش‌بینی شده مدل را با مقادیر واقعی داده‌ها مقایسه کرد. سپس، مجموع مربعات اختلاف بین پیش‌بینی‌های مدل و مقادیر واقعی را با مجموع مربعات اختلاف بین میانگین داده‌های واقعی (به عنوان یک baseline ساده) مقایسه می‌کنیم. در نهایت،  $R^2$  Score برابر است با یک منهای نسبت مجموع مربعات خطای پیش‌بینی مدل به مجموع

مربعات خطای پیش‌بینی یک مدل ساده با میانگین داده‌های واقعی:

$$R2Score = 1 - (MSE(model)/MSE(baseline))$$

که در اینجا MSE مخفف عبارت Mean Squared Error است و به عنوان یکی از معیارهای پایه در تحلیل رگرسیون بکار می‌رود. مقدار R2 Score بین ۰ و ۱ قرار می‌گیرد و به عنوان یک معیار کیفیت پیش‌بینی مدل رگرسیونی استفاده می‌شود. مقدار ۱ به معنی پیش‌بینی کامل و صحیح تمامی داده‌ها توسط مدل است، در حالی که مقدار ۰ به معنی پیش‌بینی افتضاح مدل (مدلی که توانایی پیش‌بینی بهتری نسبت به میانگین داده‌های واقعی ندارد) است. معنای منفی شدن مقدار R2 Score این است که مدل شما بهتر از مدل ساده یعنی پیش‌بینی مقدار میانگین نیست. به عبارت دیگر، مدل شما بهتر از یک حد پایینی نیست و پیش‌بینی‌های آن بدتر از پیش‌بینی با میانگین داده‌های واقعی هستند. در این حالت، نیاز است که مدل را با مدل‌های دیگر مقایسه کنید و از معیارهای دیگری مانند MSE استفاده کنید تا عملکرد مدل خود را بیشتر بررسی کنید. همچنین، در بعضی موارد، ممکن است که مدل شما به دلیل نوع داده‌ها، تعداد نمونه‌ها یا عدم تعادل در توزیع داده‌ها بهبود نیاز داشته باشد. در این صورت، می‌توانید با استفاده از تکنیک‌هایی مانند تغییر نوع مدل، افزایش تعداد داده‌ها یا اعمال تعدیل‌هایی در داده‌های ورودی، بهبود عملکرد مدل خود را تضمین کنید. در کل، مقدار R2 Score یکی از معیارهای پایه در تحلیل رگرسیون است که به شما امکان می‌دهد کیفیت پیش‌بینی مدل خود را با یک مدل ساده (پیش‌بینی با میانگین داده‌های واقعی) مقایسه کنید. همچنین، باید توجه داشت که این معیار نمی‌تواند به تنهایی برای ارزیابی عملکرد مدل‌های رگرسیونی به کار برده شود و باید با دیگر معیارهای کیفیت پیش‌بینی مانند MAE، MSE و Rmse همراه شود تا بتوان به صورت جامع عملکرد مدل را بررسی کرد.

در ادامه با انتخاب بهینه‌ساز Adam و تابع اتلاف MSE به نوشتن دستوراتی برای برآوردن هدف این قسمت از سال می‌پردازیم. این کد یک شبکه عصبی چند لایه با تعداد لایه‌های مخفی متفاوت را با استفاده از کتابخانه پایتورچ آموزش می‌دهد. در طول آموزش، کارایی شبکه با استفاده از R2 score برای مجموعه‌های آموزش و اعتبارسنجی محاسبه می‌شود و خطا با استفاده از MSE Loss محاسبه می‌شود. در پایان بهترین مدل بر اساس کارایی برای مجموعه اعتبارسنجی انتخاب شده و بازگشت داده می‌شود. در بخش تعریف مدل سه کلاس برای مدل‌های MLP تعریف شده است. هر کلاس، یک مدل با تعداد لایه‌های مخفی مختلف است. در این مدل‌ها از تابع فعال‌سازی ReLU استفاده شده است. در ادامه یک تابع برای آموزش مدل‌ها و ذخیره‌سازی و نمایش نتایج نوشته شده است. این تابع یک مدل شبکه عصبی را با استفاده از الگوریتم بهینه‌سازی و تابع هزینه داده‌شده به آن، برای داده‌های آموزشی و اعتبارسنجی آموزش می‌دهد و بهترین مدل را بر اساس R2 score برای داده‌های اعتبارسنجی برمی‌گرداند. ورودی‌های تابع شامل این موارد است:

- **model**: مدل شبکه عصبی که باید آموزش داده شود.
- **optimizer**: الگوریتم بهینه‌سازی که برای آموزش مدل استفاده می‌شود.
- **criterion**: تابع هزینه که برای آموزش مدل استفاده می‌شود.
- **train data**: داده‌های آموزشی.
- **train label**: برچسب‌های داده‌های آموزشی.
- **val data**: داده‌های اعتبارسنجی.
- **val label**: برچسب‌های داده‌های اعتبارسنجی.
- **num epochs**: تعداد اپوک‌ها برای آموزش مدل.



این تابع ابتدا مدل را به حالت آموزشی در می‌آورد و بهینه‌ساز را صفر می‌کند. سپس خروجی مدل را با داده‌های آموزش و تابع خطا مقایسه کرده و از این طریق مقدار خطای آموزش را محاسبه می‌کند. در ادامه، مقدار خطا را به عقب منتقل کرده و بهینه‌ساز را فعال می‌کند. سپس، مقدار خطای آموزش و R2 score مربوط به آموزش را در لیست‌های مربوطه ذخیره می‌کند. در مرحله بعد، مدل را به حالت ارزیابی در می‌آورد و با استفاده از داده‌های اعتبارسنجی، خروجی مدل و مقدار خطای آن را محاسبه می‌کند. سپس R2 score مربوط به اعتبارسنجی را محاسبه کرده و آن را در لیست‌های مربوطه ذخیره می‌کند. همچنین، اگر R2 score بهبود یافته باشد، بهترین مدل و مقدار آن را در لیست‌های مربوطه ذخیره می‌کند. در نهایت، این تابع با استفاده از لیست‌های خطا و R2 score، منحنی‌های مربوطه را رسم کرده و آن‌ها را به صورت فایل پی‌دی‌اف ذخیره می‌کند. در ادامه، داده‌های آموزش و آزمایش با استفاده از کلاس Tensor از پایتورچ تعریف شده‌اند. سپس هاپرپارامترهای مدل تعریف شده‌اند که شامل ابعاد ورودی، ابعاد خروجی، اندازه لایه‌های مخفی و نرخ یادگیری است. سپس از سه مدل مختلف از MLP با تعداد لایه‌های مخفی متفاوت استفاده شده است. برای هر یک از این مدل‌ها، یک بهینه‌ساز و یک تابع هزینه نیز تعریف شده‌اند و با استفاده از تابع آموزش تعریف شده، مدل‌ها آموزش داده شده‌اند. در نهایت، بهترین مدل برگشت داده می‌شود. در نهایت علاوه بر منطق خواسته شده در **قسمت V**، یک سیستم ارزیابی کلی روی داده‌های تست طراحی شده است که در آن ابتدا برای هر یک از مدل‌ها، پارامترهای بهترین مدل با استفاده از توابع ذخیره‌سازی درون پایتورچ بارگیری می‌شوند. سپس با استفاده از تابع eval() روی مدل‌ها، به آن‌ها می‌گوییم که حالت آزمایشی دارند و قادر به به‌روزرسانی پارامترهای خود نیستند. در نهایت، با استفاده از تابع no\_grad() ارزیابی مدل‌ها روی داده‌های آزمایشی انجام می‌شود و خروجی‌های تولید شده توسط مدل‌ها با داده‌های آزمایشی مقایسه می‌شود. برای اندازه‌گیری دقت مدل‌ها، از دو معیار، یعنی میزان خطا (اتلاف) و ضریب تعقیب (R2 score) استفاده می‌شود. در نهایت، نتایج ارزیابی برای هر مدل در خروجی چاپ می‌شود. با ارائه این توضیحات دستورات استفاده شده به شرح زیر است:

```
1 import torch
2 import torch.nn as nn
3 import torch.optim as optim
4 from sklearn.metrics import r2_score
5 import matplotlib.pyplot as plt
6
7 # Define the MLP models
8 class MLP1(nn.Module):
9     def __init__(self, input_dim, hidden_dim, output_dim):
10         super(MLP1, self).__init__()
11         self.fc1 = nn.Linear(input_dim, hidden_dim)
12         self.fc2 = nn.Linear(hidden_dim, output_dim)
13
14     def forward(self, x):
15         x = torch.relu(self.fc1(x))
16         x = self.fc2(x)
17         return x
18
19 class MLP2(nn.Module):
20     def __init__(self, input_dim, hidden_dim1, hidden_dim2, output_dim):
21         super(MLP2, self).__init__()
22         self.fc1 = nn.Linear(input_dim, hidden_dim1)
23         self.fc2 = nn.Linear(hidden_dim1, hidden_dim2)
```

```

24     self.fc3 = nn.Linear(hidden_dim2, output_dim)
25
26     def forward(self, x):
27         x = torch.relu(self.fc1(x))
28         x = torch.relu(self.fc2(x))
29         x = self.fc3(x)
30         return x
31
32 class MLP3(nn.Module):
33     def __init__(self, input_dim, hidden_dim1, hidden_dim2, hidden_dim3, output_dim):
34         super(MLP3, self).__init__()
35         self.fc1 = nn.Linear(input_dim, hidden_dim1)
36         self.fc2 = nn.Linear(hidden_dim1, hidden_dim2)
37         self.fc3 = nn.Linear(hidden_dim2, hidden_dim3)
38         self.fc4 = nn.Linear(hidden_dim3, output_dim)
39
40     def forward(self, x):
41         x = torch.relu(self.fc1(x))
42         x = torch.relu(self.fc2(x))
43         x = torch.relu(self.fc3(x))
44         x = self.fc4(x)
45         return x
46
47 # Define the training function
48 def train(model, optimizer, criterion, train_data, train_label, val_data, val_label, num_epochs
    =1000):
49     train_loss_list = []
50     val_loss_list = []
51     r2score_list = []
52     train_r2score_list = []
53     best_model = None
54     best_r2score = -1
55
56     for epoch in range(num_epochs):
57         # Training
58         model.train()
59         optimizer.zero_grad()
60         train_output = model(train_data)
61         train_loss = criterion(train_output.squeeze(), train_label)
62         train_loss.backward()
63         optimizer.step()
64         train_loss_list.append(train_loss.item())
65         train_r2score = r2_score(train_label, train_output.squeeze().detach().numpy())
66         train_r2score_list.append(train_r2score)
67

```

```

68     # Validation
69     model.eval()
70     with torch.no_grad():
71         val_output = model(val_data)
72         val_loss = criterion(val_output.squeeze(), val_label)
73         val_loss_list.append(val_loss.item())
74         r2score = r2_score(val_label, val_output.squeeze().detach().numpy())
75         r2score_list.append(r2score)
76         if r2score > best_r2score:
77             best_r2score = r2score
78             best_model = model.state_dict()
79
80     # Print loss and R2 score every 100 epochs
81     if (epoch+1) % 5 == 0:
82         print(f"Epoch [{epoch+1}/{num_epochs}], Train Loss: {train_loss.item():.4f}, Val Loss
      : {val_loss.item():.4f}, Train R2 score: {train_r2score:.4f}, Val R2 score: {r2score:.4f}")
83
84     # Plot loss curve and save as pdf
85     plt.plot(train_loss_list, label='Train Loss')
86     plt.plot(val_loss_list, label='Validation Loss')
87     plt.xlabel('Epoch')
88     plt.ylabel('Loss')
89     plt.legend()
90     plt.title(type(model).__name__)
91     plt.savefig(f"{type(model).__name__}_loss.pdf")
92     plt.show()
93
94     # Plot R2 score curve and save as pdf
95     plt.plot(train_r2score_list, label='Train R2 score')
96     plt.plot(r2score_list, label='Val R2 score')
97     plt.xlabel('Epoch')
98     plt.ylabel('R2 score')
99     plt.legend()
100    plt.title(type(model).__name__)
101    plt.savefig(f"{type(model).__name__}_r2score.pdf")
102    plt.show()
103
104    # Return the best model based on the validation R2 score
105    return best_model
106
107 # Define the data and labels
108 train_data = torch.Tensor(train_data)
109 train_label = torch.Tensor(train_label)
110 val_data = torch.Tensor(val_data)
111 val_label = torch.Tensor(val_label)

```

```

112 test_data = torch.Tensor(test_data)
113 test_label = torch.Tensor(test_label)
114
115 # Define the hyperparameters
116 input_dim = 23
117 output_dim = 1
118 hidden_dim1 = 16
119 hidden_dim2 = 64
120 hidden_dim3 = 128
121 learning_rate = 0.001
122 num_epochs = 120
123
124 # Train the models
125 model1 = MLP1(input_dim, hidden_dim1, output_dim)
126 optimizer1 = optim.Adam(model1.parameters(), lr=learning_rate)
127 criterion1 = nn.MSELoss()
128 best_model1 = train(model1, optimizer1, criterion1, train_data, train_label, val_data, val_label,
129                     num_epochs=num_epochs)
130
131 model2 = MLP2(input_dim, hidden_dim1, hidden_dim2, output_dim)
132 optimizer2 = optim.Adam(model2.parameters(), lr=learning_rate)
133 criterion2 = nn.MSELoss()
134 best_model2 = train(model2, optimizer2, criterion2, train_data, train_label, val_data, val_label,
135                     num_epochs=num_epochs)
136
137 model3 = MLP3(input_dim, hidden_dim1, hidden_dim2, hidden_dim3, output_dim)
138 optimizer3 = optim.Adam(model3.parameters(), lr=learning_rate)
139 criterion3 = nn.MSELoss()
140 best_model3 = train(model3, optimizer3, criterion3, train_data, train_label, val_data, val_label,
141                     num_epochs=num_epochs)
142
143 # Evaluate the models on the test set
144 model1.load_state_dict(best_model1)
145 model1.eval()
146 with torch.no_grad():
147     test_output1 = model1(test_data)
148     test_loss1 = criterion1(test_output1.squeeze(), test_label)
149     test_r2score1 = r2_score(test_label, test_output1.squeeze().detach().numpy())
150
151 model2.load_state_dict(best_model2)
152 model2.eval()
153 with torch.no_grad():
154     test_output2 = model2(test_data)
155     test_loss2 = criterion2(test_output2.squeeze(), test_label)
156     test_r2score2 = r2_score(test_label, test_output2.squeeze().detach().numpy())

```

```

154
155 model3.load_state_dict(best_model3)
156 model3.eval()
157 with torch.no_grad():
158     test_output3 = model3(test_data)
159     test_loss3 = criterion3(test_output3.squeeze(), test_label)
160     test_r2score3 = r2_score(test_label, test_output3.squeeze().detach().numpy())
161
162 # Print the test set results
163 print("MLP1 Test Set Results:")
164 print(f"Loss: {test_loss1:.4f}, R2 Score: {test_r2score1:.4f}")
165
166 print("MLP2 Test Set Results:")
167 print(f"Loss: {test_loss2:.4f}, R2 Score: {test_r2score2:.4f}")
168
169 print("MLP3 Test Set Results:")
170 print(f"Loss: {test_loss3:.4f}, R2 Score: {test_r2score3:.4f}")

```

نتایج مربوط به هر پیاده‌سازی در شکل ۷ آورده شده است. همان‌طور که مشاهده می‌شود نتایج مربوط به مدل سوم توانسته به کمینه مقادیر اتلاف و بیشینه مقدار R2 score دست پیدا کند که این موضوع در نتیجه مربوط به آزمایش که به شرح زیر است هم تأیید شده است:

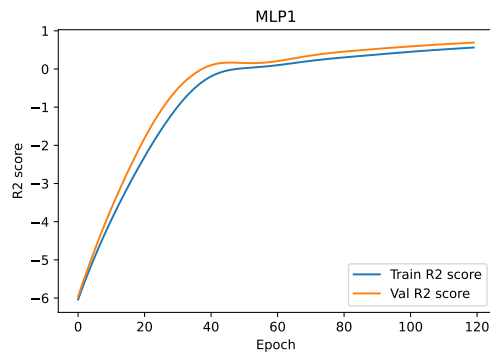
```

1 MLP1 Test Set Results:
2 Loss: 0.0157, R2 Score: 0.6370
3 MLP2 Test Set Results:
4 Loss: 0.0062, R2 Score: 0.8566
5 MLP3 Test Set Results:
6 Loss: 0.0045, R2 Score: 0.8952

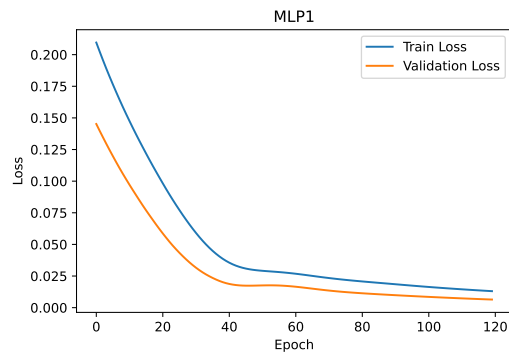
```

#### ۴.۲.۱ قسمت IV

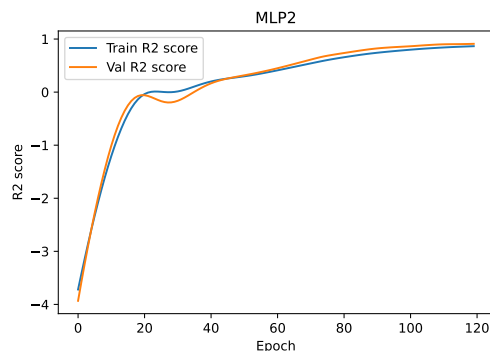
در دفترچه‌کد مربوط به این سوال، ما تمام توابع اتلاف و بهینه‌سازها مختلف را روی هر سه مدل آزمایش کردیم؛ اما در این‌جا نتایج را فقط برای مدلی که در قسمت III بهترین نتیجه را به دست داده بود گزارش می‌کنیم. شکل ۹ هر دو بهینه‌ساز Adam و Adagrad الگوریتم‌های مؤثری هستند که برای بهبود عملکرد الگوریتم‌های یادگیری ماشین استفاده می‌شوند. اما هر کدام از این الگوریتم‌ها در شرایط مختلف و برای مسائل خاص، بهترین عملکرد را دارد. در شرایطی که دیتای بسیار پراکنده داریم و گرادیان‌ها با تغییرات بزرگ همراه هستند، Adagrad بهترین عملکرد را از خود نشان می‌دهد. به عنوان مثال در بسیاری از مسائل مربوط به پردازش زبان طبیعی که داده‌های بسیار پراکنده دارند، Adagrad می‌تواند بهترین راه حل باشد. از سوی دیگر، در شرایطی که گرادیان‌ها با تغییرات بزرگی همراه نیستند و تابع هزینه یکنواخت و بهینه‌سازی بر روی یک مسئله بهینه است، بهینه‌ساز Adam می‌تواند عملکرد بهتری نسبت به Adagrad داشته باشد. در کل، همواره بهتر است برای هر مسئله و در شرایط خاص خود، الگوریتم بهینه‌سازی مناسب را انتخاب کرد و در صورت نیاز با آزمایش و انتخاب بهترین الگوریتم بهینه‌سازی، عملکرد مدل را بهبود بخشید. مقایسه نتایج مربوط به استفاده از دو بهینه‌ساز Adam و AdaGrad را نشان می‌دهد. همان‌طور که مشاهده می‌شود بهینه‌ساز Adam بهتر عمل کرده. به عنوان بررسی ثانویه و دست‌یابی احتمالی به نتایج



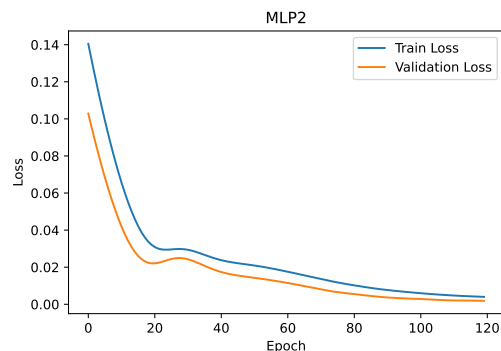
R2 score MLP1 (ب)



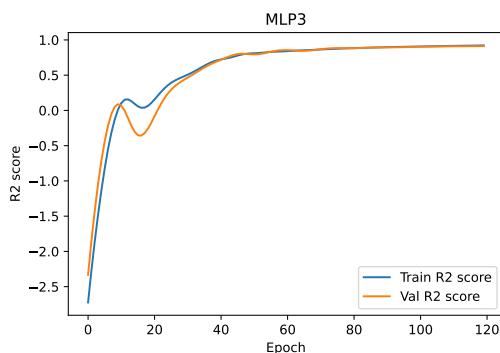
Loss MLP1 (ا)



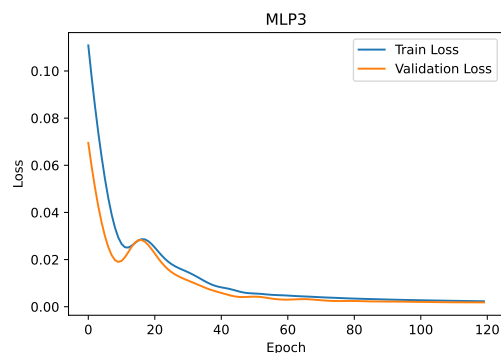
R2 score MLP2 (د)



Loss MLP2 (ج)



R2 score MLP3 (و)



Loss MLP3 (ه)

شکل ۷: نتایج مربوط به شبکه‌های با تعداد لایه مخفی مختلف.

بهتر با استفاده از بهینه‌ساز AdaGrad، نرخ یادگیری آن را بیش‌تر می‌کنیم و همان‌طور که مشاهده می‌شود در ازای دستیابی به نتایج بهتر، نوسانات ریزی ایجاد می‌شود که البته فارغ از واگرایی است (تصویر ۸(ه)). هم‌چنین نتایج روی داده‌های آزمایشی به شرح زیر است:

1 MLP3 Test Set Results (Adam):

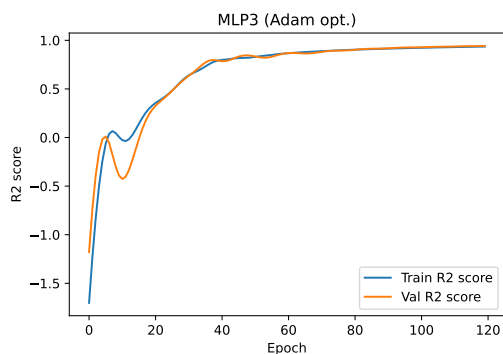
2 Loss: 0.0045, R2 Score: 0.8969

3 MLP3 Test Set Results (AdaGrad, lr=0.001):

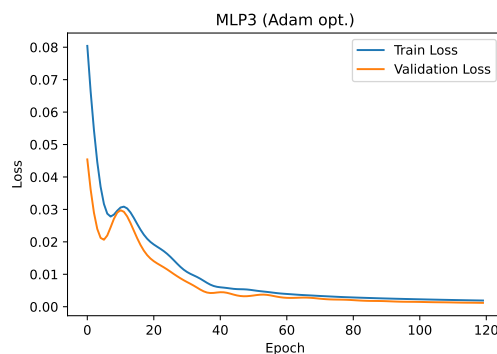
4 Loss: 0.0197, R2 Score: 0.5445

5 MLP3 Test Set Results (AdaGrad, lr=0.01):

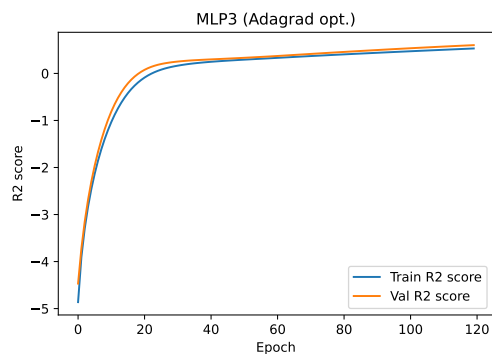
6 Loss: 0.0035, R2 Score: 0.9187



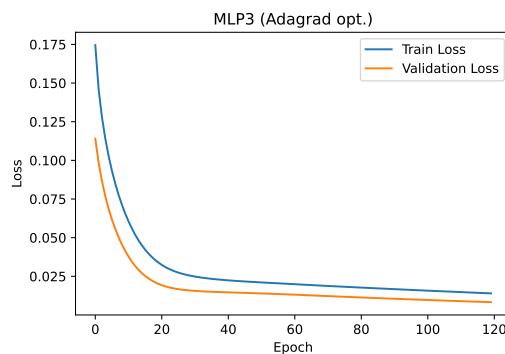
R2 score Adam (ب)



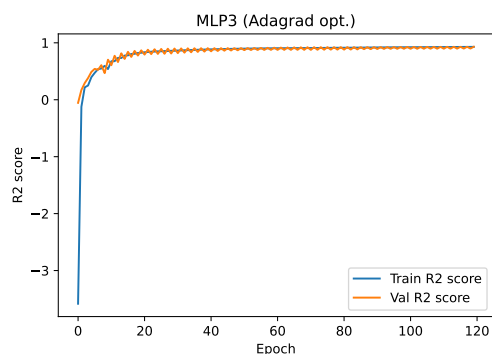
Loss Adam (ا)



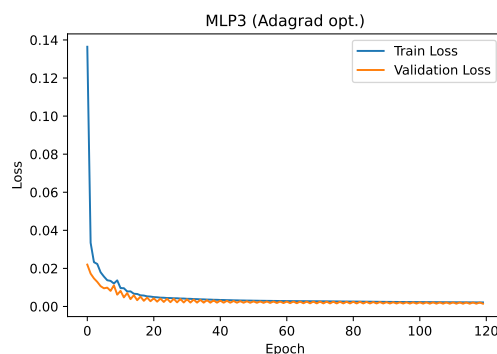
R2 score AdaGrad (lr=0.001) (د)



Loss AdaGrad (lr=0.001) (ج)



R2 score AdaGrad (lr=0.01) (و)



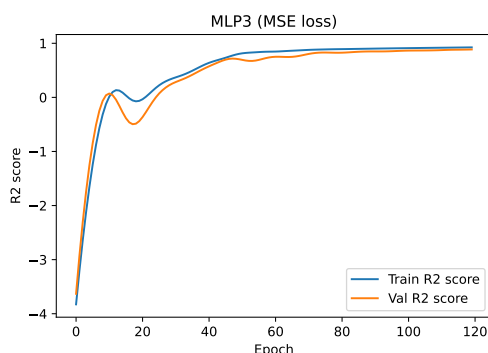
Loss AdaGrad (lr=0.01) (ه)

شکل ۸: نتایج مربوط به بهینه‌سازهای مختلف.

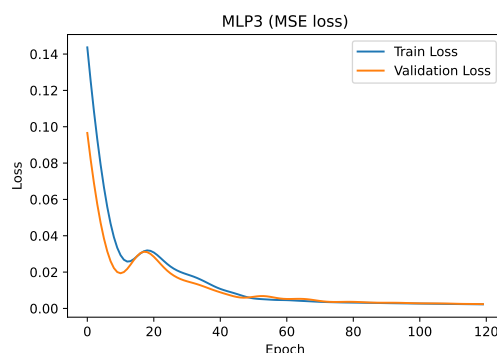
تابع اتلاف (Mean Squared Error) MSE و تابع اتلاف (Mean Absolute Error) MAE دو تابع اتلاف پرکاربرد در

یادگیری ماشین هستند. هر دو تابع اتلاف برای محاسبه خطا در پیش‌بینی مقدار یک متغیر پیوسته استفاده می‌شوند. تابع اتلاف MSE خطا را با میانگین مربعات فاصله بین پیش‌بینی شده و مقدار واقعی محاسبه می‌کند. این تابع برای محاسبه خطای شبکه‌های عصبی کارآمد است، زیرا گرادینت‌ها را به صورت پیوسته محاسبه می‌کند و باعث می‌شود که یادگیری مدل سریع‌تر صورت بگیرد. تابع اتلاف IrMAE خطا را با میانگین مطلق فاصله بین پیش‌بینی شده و مقدار واقعی محاسبه می‌کند. این تابع برای مسائلی که نویز بیشتری دارند، به عنوان مثال در پیش‌بینی ارزش خانه‌ها که فروشندگان ممکن است قیمت‌هایی را بیش از یا کمتر از میانگین قیمت‌ها اعلام کنند، عملکرد بهتری از MSE دارد. در کل، انتخاب تابع اتلاف به مسئله و داده‌های ورودی بستگی دارد. اگر داده‌های ورودی بیشتر نویز داشته باشند، MAE ممکن است بهترین راه‌حل باشد. اگر داده‌های ورودی دقیق‌تر باشند و نویز کمتری داشته باشند، MSE بهترین راه‌حل است. با این حال، هر دو تابع اتلاف در بسیاری از مسائل می‌توانند عملکرد خوبی داشته باشند و ممکن است در صورت نیاز، هر دو مورد استفاده شوند. در ادامه، همین فرآیند مقایسه‌ای را به‌ازای توابع اتلاف MSE و MAE (L1) تکرار می‌کنیم. نتایج به‌صورتی است که در ؟؟ نشان داده شده است. نتایج مربوط به داده‌های آزمایشی هم به شرح زیر است. مشاهده می‌شود که تغییر معناداری در نتایج به دست نیامده است.

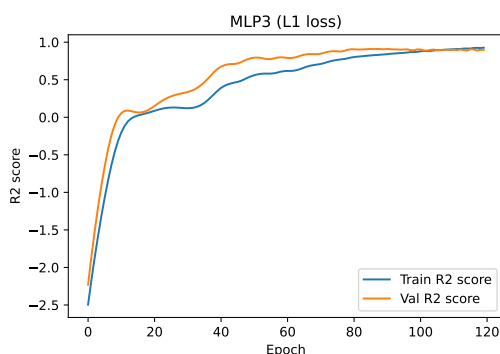
1 MLP3 Test Set Results (MSE):  
2 Loss: 0.0035, R2 Score: 0.9183  
3 MLP3 Test Set Results (MAE):  
4 Loss: 0.0388, R2 Score: 0.8944



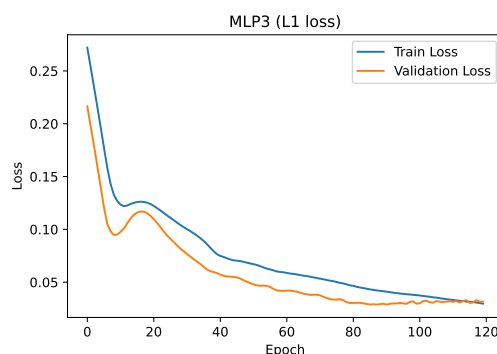
R2 score MSE (ب)



Loss MSE (ا)



R2 score MAE (د)



Loss MAE (ج)

شکل ۹: نتایج مربوط به توابع اتلاف مختلف.



## ۵.۲.۱ قسمت V

با ارائه این توضیحات دستورات زیر را به مجموعه دستورات قبلی اضافه کرده و ضمن برگرداندن مقیاس دهی انجام شده به مقایسه مقادیر واقعی و مقادیر پیش‌بینی شده قیمت می‌پردازیم:

```

1 import random
2
3 # Select 5 random indices from the test set
4 test_indices = random.sample(range(len(test_data)), 5)
5
6 # Evaluate the models on the selected test data
7 for i in test_indices:
8     input_data = test_data[i]
9     true_label = test_label[i]
10
11     model1.load_state_dict(best_model1)
12     model1.eval()
13     with torch.no_grad():
14         output1 = model1(input_data)
15         unscaled_output1 = scaler_y.inverse_transform(output1.reshape(-1, 1))
16         pred_label1 = unscaled_output1.squeeze().item()
17         unscaled_true_label = scaler_y.inverse_transform(true_label.reshape(-1, 1))
18         diff1 = abs(pred_label1 - unscaled_true_label.item())
19
20     model2.load_state_dict(best_model2)
21     model2.eval()
22     with torch.no_grad():
23         output2 = model2(input_data)
24         unscaled_output2 = scaler_y.inverse_transform(output2.reshape(-1, 1))
25         pred_label2 = unscaled_output2.squeeze().item()
26         unscaled_true_label = scaler_y.inverse_transform(true_label.reshape(-1, 1))
27         diff2 = abs(pred_label2 - unscaled_true_label.item())
28
29     model3.load_state_dict(best_model3)
30     model3.eval()
31     with torch.no_grad():
32         output3 = model3(input_data)
33         unscaled_output3 = scaler_y.inverse_transform(output3.reshape(-1, 1))
34         pred_label3 = unscaled_output3.squeeze().item()
35         diff3 = abs(pred_label3 - unscaled_true_label.item())
36
37 # Print the predicted and true values, and their differences
38 print(f"Data {i}:")
39 print("MLP1 Predicted Value: {:.4f}, True Value: {:.4f}, Difference: {:.4f}".format(
    pred_label1, unscaled_true_label.item(), diff1))

```

```

40 print("MLP2 Predicted Value: {:.4f}, True Value: {:.4f}, Difference: {:.4f}".format(
    pred_label2, unscaled_true_label.item(), diff2))
41 print("MLP3 Predicted Value: {:.4f}, True Value: {:.4f}, Difference: {:.4f}".format(
    pred_label3, unscaled_true_label.item(), diff3))

```

نتیجه به صورتی است که در زیر آمده است. همان‌طور که مشاهده می‌شود، مطابق انتظار، مقادیر پیش‌بینی شده شبکه MLP3 اختلاف کم‌تری با مقادیر واقعی دارند. عمل کرد MLP2 هم به آن نزدیک است. اما عمل کرد MLP1 به وضوح بدتر است. این نتایج منطبق بر نتایجی بود که در قسمت‌های قبلی و با ساختار تست جداگانه‌ای که طراحی کرده بودیم (نتایج تست تحمیلی و درصدی)، به دست آوردیم. هم‌چنین در این مجموعه ۵ تایی آزمایش مشاهده می‌شود که میزان اختلاف قیمت پیش‌بینی و واقعی، حداکثر ۶ درصد میزان قیمت واقعی است که این هم منطبق بر نتایج به دست آمده است.

```

1 # Data 22:
2 MLP1 Predicted Value: 14151.8072, True Value: 11721.8557, Difference: 2429.9515
3 MLP2 Predicted Value: 9716.8267, True Value: 11721.8557, Difference: 2005.0289
4 MLP3 Predicted Value: 11012.7253, True Value: 11721.8557, Difference: 709.1303
5 # Data 1:
6 MLP1 Predicted Value: 16950.5866, True Value: 19376.9675, Difference: 2426.3809
7 MLP2 Predicted Value: 17296.7602, True Value: 19376.9675, Difference: 2080.2072
8 MLP3 Predicted Value: 19032.5010, True Value: 19376.9675, Difference: 344.4664
9 # Data 5:
10 MLP1 Predicted Value: 11692.4676, True Value: 11363.4019, Difference: 329.0656
11 MLP2 Predicted Value: 10818.7804, True Value: 11363.4019, Difference: 544.6215
12 MLP3 Predicted Value: 11029.7326, True Value: 11363.4019, Difference: 333.6693
13 # Data 17:
14 MLP1 Predicted Value: 16690.7608, True Value: 15210.8052, Difference: 1479.9555
15 MLP2 Predicted Value: 16094.3459, True Value: 15210.8052, Difference: 883.5407
16 MLP3 Predicted Value: 14967.0500, True Value: 15210.8052, Difference: 243.7552
17 # Data 28:
18 MLP1 Predicted Value: 12509.8616, True Value: 11522.7143, Difference: 987.1473
19 MLP2 Predicted Value: 10460.0885, True Value: 11522.7143, Difference: 1062.6258
20 MLP3 Predicted Value: 11092.1927, True Value: 11522.7143, Difference: 430.5216

```