



به نام خدا



دانشگاه تهران
دانشکده مهندسی برق و کامپیوتر
درس شبکه‌های عصبی و یادگیری عمیق

تمرین سری دوم

نام و نام خانوادگی	محسن فیاض
شماره دانشجویی	810100524
تاریخ ارسال گزارش	17 فروردین 1400

فهرست گزارش سوالات

1

سوال 1 – MLP Classification

الف) داده های آموزش، تست ارزیابی چگونه تقسیم می نماید؟ روش‌های متفاوت را شرح دهید با ذکر دلیل مشخص نماید کدام روش مناسب‌تر است.

ب) ماتریس آشفتگی، پارامتر score-f1، پارامتر recall و پارامتر precision را با استفاده از رابطه‌ی ریاضی توضیح دهید.

ت) با استفاده ازروش Stochastic mini batch based مقدار batch را به صورت دلخواه انتخاب نماید سپس تعداد نورونهای موجود در هر لایه تغییر دهید تاثیر تفاوت تعداد نورونها را در خطای دقت و زمان آموزش شبکه بررسی نماید. مجموعه 3 مرتبه تعداد نورونها در لایه های مخفی تغییر دهید نتایج آرا در گزارش بیاورید.

د) با استفاده از بهترین مدل قسمت قبل، با استفاده ازروش Stochastic mini batch based چهار دسته اندازه های 128, 64, 32, 16 استفاده نماید تاثیر تفاوت اندازه دسته را در خطای دقت و زمان آموزش شبکه بررسی نماید.

ه) چهار تابع فعال ساز { ReLU , TanH , Sigmoid, Softmax } در نظر بگیرید، سپس مزایا و معایب این تابع فعال ساز را نسبت به دیگری بیان نماید. حال با استفاده از بهترین مدل قسمتهای قبل، توابع فعالساز هر لایه را تغییر دهید تاثیر توابع فعالسازی را در خطای دقت و زمان آموزش شبکه بررسی نماید.

و) سه تابع خطای را درنظر بگیرید، سپس با استفاده از بهترین مدل قسمتهای قبل، تابع خطای شبکه را تغییر دهید و تاثیر تابع خطای را بررسی نماید و نتایج آن را در گزارش بیاورید.

ج) سه تابع بهینه ساز را درنظر بگیرید، سپس با استفاده از بهترین مدل قسمتهای قبل، بهینه ساز شبکه تغییر دهید تاثیر بهینه سازهای متفاوت را بررسی نماید و نتایج آن را در گزارش بیاورید.

ح) با توجه به بهترین مدل قسمتهای قبل، افزودن لایه به شبکه تاثیری در خروجی دارد؟ فرضیه خود را با تغییر تعداد لایه های مختلف (سه مرتبه) بررسی کنید.

ط) با توجه به ارزیابی های انجام شده، انتخاب کدام پارامترها بهترین نتیجه را میدهد؟ دلیل خود را با توجه به قسمتهای قبل بطور کامل شرح دهید.

ن) بهترین مدل به دست آمده را در یک جدول بیان کرده و خطای دقت و زمان لازم برای آموزش شبکه و همچنین دقت و خطای داده های تست را در آن گزارش نماید.

27

سوال 2 – MLP Regression

الف) ابتدا پیش پردازش های لازم برای آماده سازی دیتا را انجام دهید و آنها را توضیح دهید. برای هر مرحله، توضیحات مختصری ارائه کنید. توجه داشته باشید در این قسمت مجاز به حذف هیچ یک از ابعاد داده نیستید.

ب) به صورت تصادفی 80 درصد داده ها به عنوان داده آموزشی و 20 درصد را به عنوان داده تست در نظر بگیرید. تعدادی از داده ها را به عنوان داده ارزیابی در نظر بگیرید (مثلا 10 تا 15 درصد داده های آموزشی).

شبکه عصبی چند لایه ای را طراحی کنید که قیمت خانه را پیش‌بینی کند. تعداد لایه تعداد نورون و تابع فعال ساز مناسب برای لایه را مشخص کنید آن را تحلیل کنید (دو حالت برای تعداد لایه و دو حالتی برای تابع فعال ساز درنظر بگیرید.)

ج) با ثابت در نظر گرفتن پارامترهای بدست آمده در قسمت(ب) با در نظر گرفتن MSE به عنوان تابع loss، مقادیر MSE و MAE را در هر اپیک برای داده آموزشی و داده تست (ارزیابی) در یک نمودار رسم کنید. (یک

نمودار برای معیار MSE یک نمودار برای MAE). برای داده تست نمودار مقادیر پیش بینی شده بر حسب مقادیر واقعی رسم نمایید. تعداد ایپاک بهینه در این قسمت را مشخص کنید.

34
د) با ثابت در نظر گرفتن پارامترهای بدست آمده در قسمت (ب) با در نظر گرفتن MAE به عنوانتابع loss، مقادیر metric های MSE و MAE را در هر ایپاک برای داده آموزشی و داده تست (ازیزایی) در یک نمودار رسم کنید (یک نمودار برای معیار MSE یک نمودار برای MAE). برای داده تست، نمودار مقادیر پیش بینی شده بر حسب مقادیر واقعی رسم نمایید. تعداد ایپاک بهینه در این قسمت را مشخص کنید.

35
36
ه) ابتدا روابط ریاضی MSE و MAE را بنویسید و سپس نتایج قسمت ج و د را باهم مقایسه کرده و توضیح دهید.

37 * امتیازی:

سوال 3 Dimensionality Reduction

الف) ایده روش PCA را بیان کنید.
با استفاده از روابط ریاضی توضیح دهید چگونه میتوان با استفاده از این روش ابعاد داده را کاهش داد. سپس منحنی تعداد components را بر حسب واریانس تجمعی برای دیتاست مجموعه داده CIFAR-10 رسم کنید و بیان کنید که کاهش بعد تا چه میزان قابل قبول است؟ در ادامه روش PCA در برای مجموعه داده CIFAR-10 پیاده سازی کنید شبکه را آموزش دهید.

39
نکته 2: در این قسمت نمیتوانید کتابخانه `matplotlib` برای پیاده سازی استفاده کنید باید خودتان پیاده سازی نمایید
42
ب) روش Autoencoder را برای مجموعه داده CIFAR-10 پیاده سازی کنید شبکه را آموزش دهید.

44
ج) نتایج مربوط به بهترین مدل به دست آمده از سوال یک را با نتایج به دست آمده در قسمت الف و ب همین سوال، در یک جدول مقایسه کنید و شهود خود را توضیح دهید.

46
د) ماتریس همبستگی مربوط به دیتاست سوال 2 را رسم کنید (مشابه شکل زیر) و توضیح مختصری در مورد آن بدھید.

47
و) با استفاده از مدل‌های Linear Regression Decision Tree، اهمیت هر ویژگی را در سوال 2 بدست آورید و در یک بار پلاس نمایش دهید. برای این کار می‌توانید از کتابخانه `scikit-learn` و متدهای `DecisionTreeRegressor` و `LinearRegression` استفاده کنید.

49 بازخورد

سوال 1 MLP Classification

کلاس‌های موجود در این دیتاست شامل موارد زیر است.

جدول 1 - کلاس‌های موجود در دیتاست

dog	5	airplane	0
frog	6	automobile	1
horse	7	bird	2
ship	8	cat	3
truck	9	deer	4

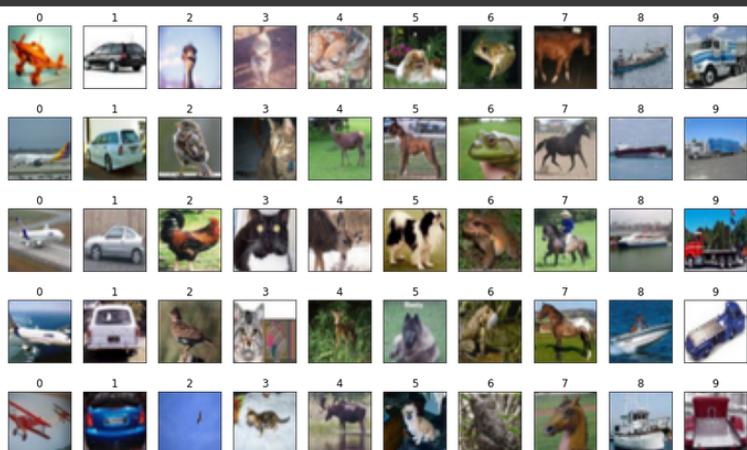
ابتدا همانطور که خواسته شده دیتاست را بارگذاری کرده و 10 تصویر آن را به صورت تصادفی نمایش می‌دهیم. (از هر کلاس یکی و 5 بار این کار را تکرار کردیم تا بهتر کلاس‌ها مشخص شوند.).

```
1 from keras.datasets import cifar10
2 import tensorflow as tf
3 import matplotlib.pyplot as plt
4 import numpy as np

1 [[x_train, y_train], (x_test, y_test)] = cifar10.load_data()
2 print("x_train:", x_train.shape)
3 print("y_train:", y_train.shape)

x_train: (50000, 32, 32, 3)
y_train: (50000, 1) <class 'numpy.ndarray'>

1 def plot_pics(x: np.ndarray, y: np.ndarray):
2     fig = plt.figure(figsize=(15, 4))
3     for class_num in range(10):
4         class_indices = np.where(y == class_num)[0]
5         class_x = x[class_indices]
6         rnd_idx = np.random.randint(len(class_x))
7         image = class_x[rnd_idx]
8         ax = fig.add_subplot(1, 10, class_num + 1, xticks=[], yticks=[])
9         ax.set_title(class_num)
10        plt.imshow(image)
11    plt.show()
12
13 for i in range(5):
14     plot_pics(x_train, y_train)
```



شکل 1 - بارگذاری و نمایش دیتاست

الف) داده های آموزش، تست ارزیابی چگونه تقسیم می نمایید؟ و شما متفاوت را شرح دهید با ذکر دلیل مشخص نمایید کدام روش مناسب است.

در این مسئله خاص 50000 داده آموزش و 10000 داده تست از قبیل مشخص شده‌اند. علت این کار این است که این دیتاست به عنوان بنچمارک استفاده می‌شود و برای مقایسه کیفیت مدل‌های مختلف باید همه روی یک بخش خاص تست شوند تا بتوان نتایج را مقایسه کرد. به صورت کلی داده تست داده‌ای است که کاملاً کنار گذاشته می‌شود و پس از آموزش مدل دقت روی آن سنجیده می‌شود. داده ارزیابی اگرچه مستقیم مدل روی آن آموزش نمی‌بیند، اما برای تعیین هایپر پارامترهای مانند اندازه بچ، یا نرخ آموزش، می‌توان از آن استفاده کرد به این صورت که پس از هر ایپاک آموزش، یک بار دقت و لاس روی این داده ارزیابی سنجیده می‌شود و یکی از هایپر پارامترهای دیگر که با آن مشخص می‌شود تعداد ایپاک‌های آموزش است که با استفاده از early stopping انجام می‌شود. برای تقسیم نسبت‌های متفاوتی می‌توان استفاده کرد که مثلاً به ترتیب 70 و 15 و 15 می‌تواند مقدار حدودی خوبی باشد. اما با توجه به دیتاست و اندازه آن معمولاً هر مقدار تعداد داده بیشتری موجود باشد می‌توانیم نسبت بیشتری به ارزیابی و تست اختصاص دهیم در حالیکه اگر تعداد داده کم باشد مجبور می‌شویم از آن دو بخش کم کنیم تا داده کافی برای آموزش و همگرایی و generalization مناسب مدل داشته باشیم.

در این سوال فقط باید بخش ارزیابی جداسازی شود. برای این کار می‌توان از توابعی که کتابخانه های مختلف دارند مانند `sklearn.model_selection.train_test_split`¹ استفاده کرد. از نکات مهم این است که حتماً shuffle باید انجام شود. مخصوصاً در بعضی دیتاست‌ها که مثلاً نصف اول دیتاست کلاس 0 و نصف دوم کلاس 1 است، در صورت shuffle نکردن، ممکن است فقط داده تست در کلاس 0 بیافتد و داده آموزش در یک کلاس دیگر. جدای از این مورد، چون معمولاً از mini batch gradient descent استفاده می‌کنیم، اگر چندین بچ پشت هم یک کلاس خاص باشد و کلاس دیگر دیده نشود، همگرایی سخت و در مواردی ناممکن می‌شود. بنابراین shuffle کردن برای نتیجه نهایی بسیار مهم است.

در این مسئله چون اجازه استفاده از keras را داریم، تابع `fit`² این کتابخانه خود دارای validation_split است که در بخش‌های بعدی از آن استفاده می‌کنیم. (همچنین shuffle نیز در این تابع به صورت پیش فرض قرار دارد)

¹ https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

² https://keras.io/api/models/model_training_apis/#fit-method

ب) ماتریس آشتفتگی، پارامتر precision و recall و پارامتر score-f1 استفاده از رابطه‌ی ریاضی توضیح دهد.

شکلی از ماتریس آشتفتگی در ادامه آمده است.

		gold standard labels		
		gold positive	gold negative	
system output labels	system positive	true positive	false positive	precision = $\frac{tp}{tp+fp}$
	system negative	false negative	true negative	
		recall = $\frac{tp}{tp+fn}$		
				accuracy = $\frac{tp+tn}{tp+fp+tn+fn}$

شکل 2 - ماتریس آشتفتگی

در ماتریس آشتفتگی، یک طرف مقدار واقعی برچسب دیتاست را نشان می‌دهد، و یک طرف برچسبی که مدل ما تشخیص داده است. تعداد 4 حالت ساخته شده از این دو مورد (در مسئله دو کلاسه) ماتریس آشتفتگی را می‌سازد. در این سوال که 10 کلاس داریم ماتریسی 10 در 10 خواهیم داشت.

پارامتر recall معادل نسبت تعداد داده‌هایی که مدل به درستی مثبت تشخیص داده است به تمام داده‌هایی که در دیتاست مثبت بوده‌اند است. این مقدار نشان می‌دهد که مدل چه مقدار از دیتاست را پوشش داده است.

در مقابل آن، مقدار precision معادل نسبت تعداد داده‌هایی که مدل به درستی مثبت تشخیص داده است به تمام داده‌هایی که مثبت تشخیص داده است می‌باشد. این مقدار نشان می‌دهد که مدل چه مقدار دقت نظر داشته و اگر چیزی را مثبت تشخیص بدهد چقدر می‌توان مطمئن بود که واقعاً مثبت است.

هرچند این دو پارامتر نشان دهنده عملکرد مدل هستند، اما در هر دو طرف می‌توان مقدار خوبی گرفت در حالیکه در دیگری بد بود. مثلاً مدلی که همیشه خروجی مثبت بدهد طبق توضیح گفته شده recall برابر 1 خواهد داشت در حالی که مدل خوبی نبوده و احتمالاً precision کمی دارد. بنابراین ترکیب این دو گزینه مناسب است.

پارامتر f1 score همان ترکیب precision و recall است، اما به جای میانگین از میانگین هارمونیک استفاده می‌شود تا روی این که هر دو مقدار بالا باشند و نه فقط یکی از آنها تاکید

بیشتری باشد. مثلا اگر یکی 100 و یکی 0 باشد، میانگین عادی برابر 50 می‌شد، در حالیکه میانگین هارمونیک مقدار 0 می‌دهد.

$$F_1 = \frac{2P \times R}{P + R}, P = \frac{tp}{tp + fp}, R = \frac{tp}{tp + fn}$$

پیش‌پردازش‌های لازم در زیر آمده است.

```
def onehot(a):
    a = a.flatten()
    o = np.zeros((a.size, a.max() + 1))
    o[np.arange(a.size), a] = 1
    return o

def preprocess(x, y):
    assert x.shape[1:] == (32, 32, 3)
    x = np.mean(x, axis=-1) # grayscale
    x = np.reshape(x, (-1, 32 * 32)) # flatten
    x /= 255 # normalize
    y = onehot(y) # onehot labels
    return x, y

x_train_processed, y_train_processed = preprocess(x_train, y_train)
x_test_processed, y_test_processed = preprocess(x_test, y_test)
x_train_processed.shape, y_train_processed.shape
→ ((50000, 1024), (50000, 10))
```

کد ۱ - پیش‌پردازش

ابتدا با استفاده از میانگین گیری عکس به خاکستری تبدیل شد. سپس با تغییر ابعاد به اصطلاح flatten انجام شد و عکس دو بعدی به یک بعد 1024 پیکسلی تبدیل شد. و در آخر با تقسیمی که انجام شد اعداد نرمال شدند و به بین 0 و 1 آمدند.

برای برچسب‌ها نیز، در طبقه‌بندی از ساخت یک ماتریس sparse یا همان onehot encoding استفاده می‌شود که هر نمونه یک وکتور خواهد داشت که در عدد برچسبش 1 و در بقیه خانه‌ها 0 است. این کار برای آموزش راحت‌تر مدل خیلی کمک می‌کند و آموزش و همگرایی را آسان‌تر می‌کند. در آخر ابعاد داده چاپ شده است.

ت) با استفاده از روش **Stochastic mini batch based** مقدار λ به صورت دلخواه انتخاب نمایید سپس تعداد نورونهای موجود در هر لایه تغییر دهید تا ثیر تفاوت تعداد نورونها را در خطا، دقت و زمان آموزش شبکه بررسی نمایید. مجموعه 3 مرتبه تعداد نورونها در لایه های مخفی تغییر دهید نتایج آرایه کارش بیاورد.

ابتدا به توضیح کد می پردازیم.

```
class Trainer:
    def __init__(self, units=1024, activation_function="relu", optimizer="adam", n_layers=4) -> None:
        self.optimizer = optimizer
        self.units = units
        self.activation_function = activation_function
        self.n_layers = n_layers
        self.model = self.build_mlp_model()
        self.history = None
        self.training_time = None

    def print_summary(self):
        print("#####")
        print(f"Training Time: {self.training_time:.2f}s")
        print(f"batch size: {self.batch_size}")
        print(f"n_layers: {self.n_layers}")
        print(f"optimizer: {self.optimizer}")
        print(self.model.summary())

    def build_mlp_model(self):
        model = tf.keras.Sequential([tf.keras.layers.Input(shape=(1024, ))])
        for n in range(self.n_layers):
            model.add(tf.keras.layers.Dense(self.units / 2**n, activation=self.activation_function))
        # model.add(tf.keras.layers.Dropout(0.1))
        model.add(tf.keras.layers.Dense(10, activation='softmax'))

        model.compile(
            optimizer=self.optimizer,
            loss='categorical_crossentropy',
            metrics=['accuracy']
        )
        return model

    def train(self, x_train, y_train, batch_size=32, epochs=50):
        self.batch_size = batch_size
        assert x_train.shape[1:] == (1024, )
        assert y_train.shape[1:] == (10, )
        early_stopping = tf.keras.callbacks.EarlyStopping(
            monitor='val_loss',
            verbose=1,
            patience=10,
            mode='min',
            restore_best_weights=True
        )
        start = time()
        self.history = self.model.fit(
            x_train, y_train,
            batch_size=batch_size,
            epochs=epochs,
            verbose=1,
            validation_split=0.2,
            callbacks=[early_stopping],
        )
        self.training_time = time() - start

    def plot_history(self):
        fig = plt.figure(figsize=(12, 4))
        metrics = ['loss', 'accuracy']
        for n, metric in enumerate(metrics):
            plt.subplot(1, 2, n+1)
            plt.plot(self.history.epoch, self.history.history[metric], label='Train')
            plt.plot(self.history.epoch, self.history.history[f"val_{metric}"], linestyle="--", label='Validation')
            plt.xlabel('Epoch')
            plt.ylabel(metric)
            plt.title(metric)
            plt.legend()
        plt.show()
```

```

def evaluate(self, x_test, y_test):
    assert x_test.shape[1:] == (1024, )
    assert y_test.shape[1:] == (10, )
    [test_loss,test_acc] = self.model.evaluate(x_test,y_test)
    print("Test Loss:", test_loss, "Test Accuracy:", test_acc)
    test_preds = np.argmax(self.model.predict(x_test), axis=-1)
    y_test = np.argmax(y_test, axis=-1)
    print(classification_report(y_test, test_preds))
    self.plot_cm(y_test, test_preds)

    def plot_cm(self, y_true, preds):
        cm = confusion_matrix(y_true, preds)
        plt.figure(figsize=(7, 5))
        ax = sns.heatmap(cm, annot=True, fmt="d")
        bottom, top = ax.get_ylim()
        ax.set_ylim(bottom + 0.5, top - 0.5)
        plt.title('Confusion Matrix')
        plt.ylabel('True Label')
        plt.xlabel('Predicted Label')
        plt.show()

for n_units in [128, 1024, 2048]:
    trainer = Trainer(
        units=n_units,
        activation_function="relu",
        optimizer=tf.keras.optimizers.Adam(learning_rate=1e-4),
        n_layers=4
    )
    trainer.train(x_train_processed, y_train_processed,
                  batch_size=64,
                  epochs=100)
    trainer.print_summary()
    trainer.plot_history()
    trainer.evaluate(x_test_processed, y_test_processed)

```

کد 2 - آموزش مدل

در این کد با استفاده از تعداد دلخواه لایه tf.keras.layers.Dense که به عنوان ورودی گرفته می‌شود مدل ساخته می‌شود. برای تعداد نورون هر لایه نیز از مقداری که مشخص شده شروع و هر لایه نصف می‌شود تا به لایه آخر که باید 10 کلاسه باشد برسد. در ابتدا برای هایپرپارامترهای مسئله که همگی به صورت ارگومان پیاده سازی شده اند مقادیر دلخواهی گذاشته‌ایم که در این بخش و بخش‌های بعدی مقادیر متفاوت را امتحان و بررسی می‌کنیم. همچنین پس از آموزش مدل، نتیجه مدل را روی بخش تست هم می‌سنجم و مواردی که در صورت پرتوه خواسته شده را نمایش می‌دهیم. ضمناً باید اشاره شود که از earlystopping هم با بررسی loss بخش ارزیابی استفاده شده است به این صورت که در 10 ایپاک اگر پیشرفتی حاصل نشود، آموزش متوقف می‌شود.

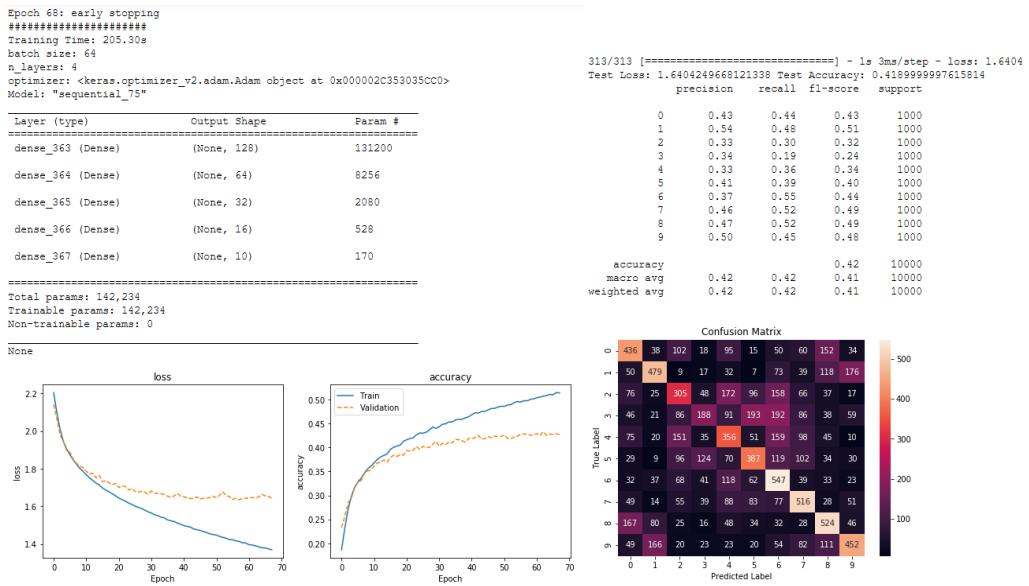
در این بخش بررسی تاثیر تعداد نورون بوده که مقادیر 128 و 1024 و 2048 را امتحان کردیم.
(بیشتر هم می‌شد ولی برای اجرای سریعتر از 2048 بیشتر نرفتیم.)

نتیجه 128 نورون:

● خطأ: 1.64

● دقت: 0.41

● زمان کل آموزش: 205.30s در 68 اپیاک (3.01 در اپیاک)



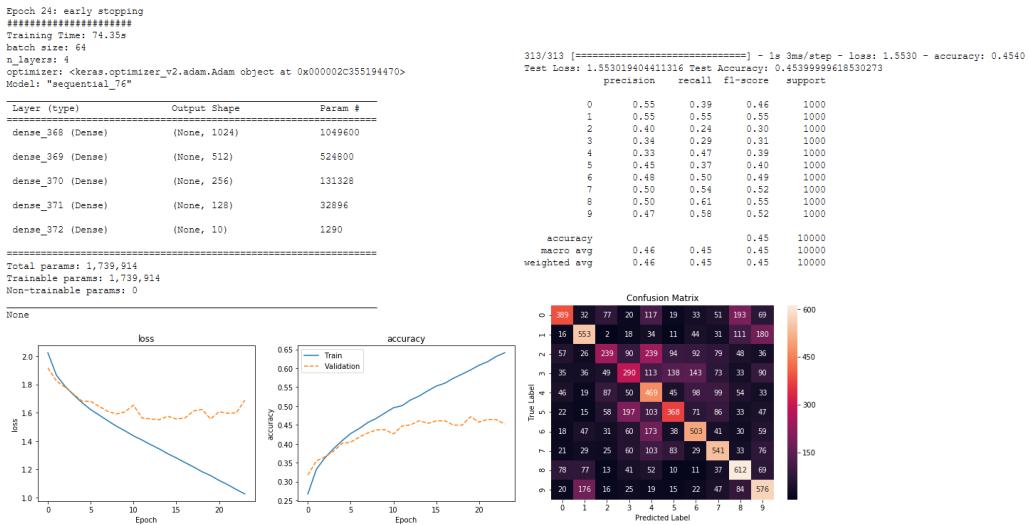
شكل 3 - نتیجه 128 نورون

نتیجه 1024 نورون:

● خطأ: 1.55

● دقت: 0.45

● زمان کل آموزش: 74.35s در 24 اپیاک (3.08 در اپیاک)



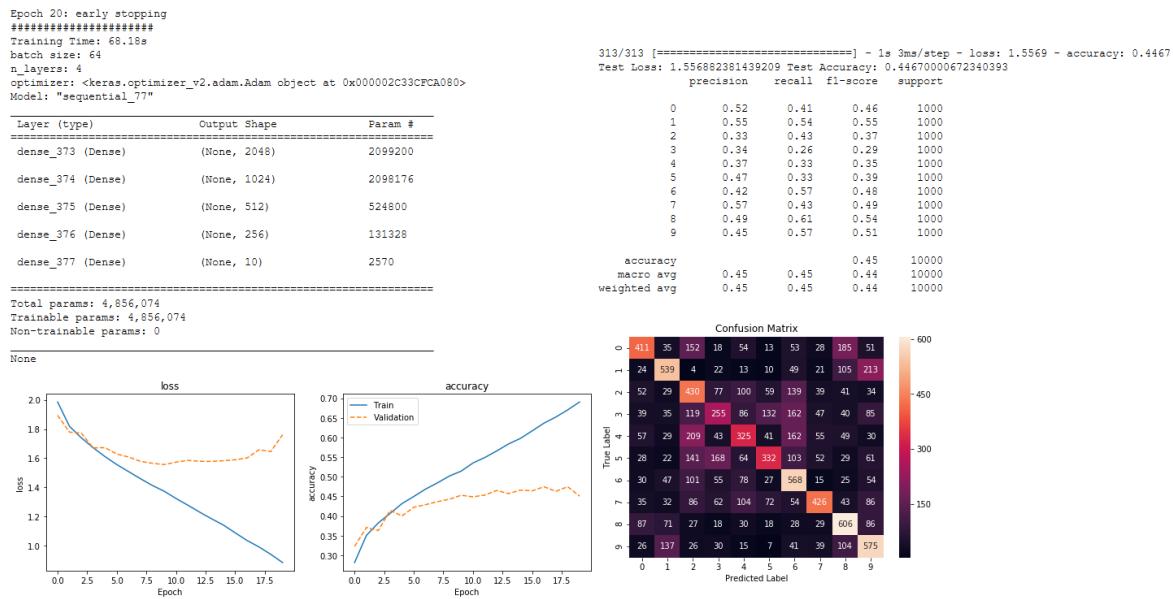
شكل 4 - نتیجه 1024 نورون

نتیجه 2048 نورون:

● خطأ: 1.55

● دقت: 0.45

● زمان کل آموزش: 68.18s در 20 اپیک (3.4 در اپیک)



شکل 5 - نتیجه 2048 نورون

از نظر خطأ و دقت با افرايش نورون و تقويت قدرت مدل، طبيعتا همانطور که دیده می شود خطأ کمتر شده و دقت بهتر می شود. (البته باید توجه داشت که بیش از حد پیچیده کردن مدل هم می تواند از generalization آن بکاهد و بدتر باعث overfit شدن و کاهش دقت تست شود که روش های regularization یا استفاده از لایه drop out یا ساده تر کردن مدل می تواند این مشکل را برطرف کند).

از نظر زمان نیز هر چه تعداد نورون بیشتر شود محاسبات بیشتری باید انجام شود و به همین دیده می شود که زمان متوسط اجرای یک اپیک با افرايش نورون ها چطور افرايش یافته است. البته باید توجه داشت که با اجرای پارالل نزوما این افرايش تاثیر مستقيم و رابطه خطی با زمان نخواهد داشت همچنان که این اجراءها روی GPU انجام شده اند و دیده می شود که با دو برابر کردن تعداد نورون زمان اجرا نزوما دو برابر نشده است ولی قطعا افزایش داشته است. به دلیل دقت مناسب 1024 نورون از این به بعد با این مقدار پیش می رویم.

ضمیما باید ذکر کرد که در اجراهای انجام شده لایه drop out نبود. اگر این لایه هم گذاشته شود تا از overfitting با خاموش کردن رندوم درصدی از نورون‌ها جلوگیری شود مقادیر دقت برای سه آزمایش انجام شده به ترتیب 44 و 47 و 46 می‌شود.

د) با استفاده از بهترین مدل قسمت قبل، با استفاده از روش Stochastic mini batch چهار دسته اندازه های 32, 64, 128 و 512 استفاده نمایید تاثیر تفاوت اندازه دسته را در خطای دقت و زمان آموزش شبکه بررسی نمایید.

کد را برای بچ سایزهای [32, 64, 128, 512] اجرا می‌کنیم.

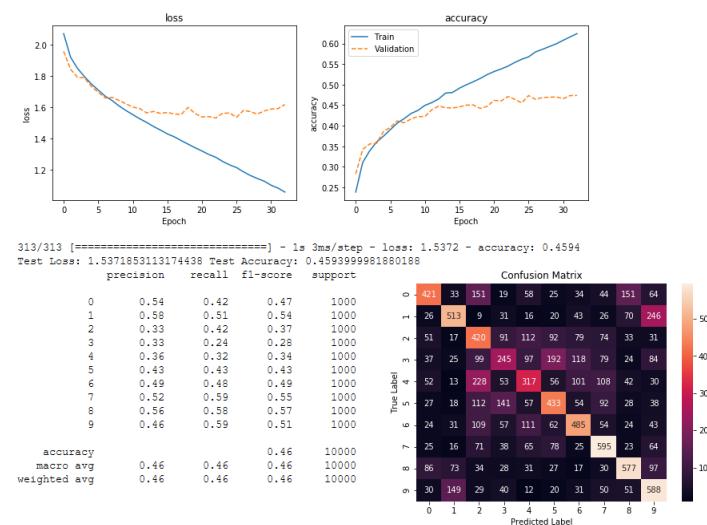
نتیجه اندازه 32:

خطای 1.537 ●

دقت: 0.459 ●

زمان کل آموزش: 215.15s در 33 اپیک (6.51 در اپیک)

```
Epoch 33: early stopping
#####
Training Time: 215.15s
batch size: 32
n_layers: 4
optimizer: <keras.optimizer_v2.adam.Adam object at 0x000002C354720F98>
Model: "sequential_83"
Layer (type)          Output Shape         Param #
=====
dense_403 (Dense)     (None, 1024)        1049600
dropout_198 (Dropout) (None, 1024)        0
dense_404 (Dense)     (None, 512)         524800
dropout_199 (Dropout) (None, 512)         0
dense_405 (Dense)     (None, 256)         131328
dropout_200 (Dropout) (None, 256)         0
dense_406 (Dense)     (None, 128)         32896
dropout_201 (Dropout) (None, 128)         0
dense_407 (Dense)     (None, 10)          1290
=====
Total params: 1,739,914
Trainable params: 1,739,914
Non-trainable params: 0
None
```



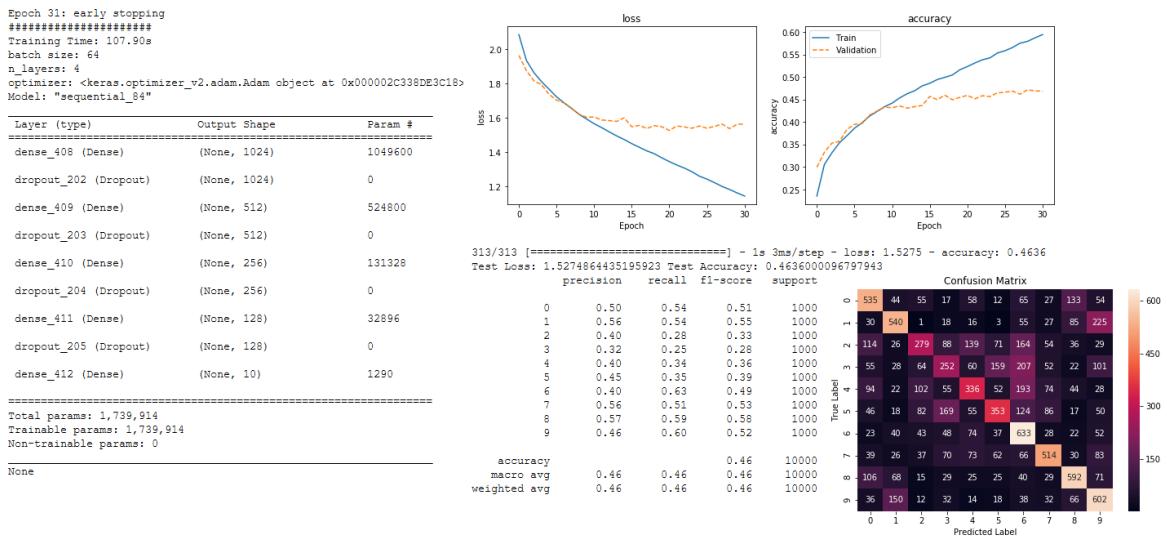
شکل 6 - نتیجه اندازه دسته 32

نتیجه اندازه 64 :

● خطأ: 1.527

● دقت: 0.463

● زمان کل آموزش: 3.45 در 31 اپیک (107.90s در اپیک)



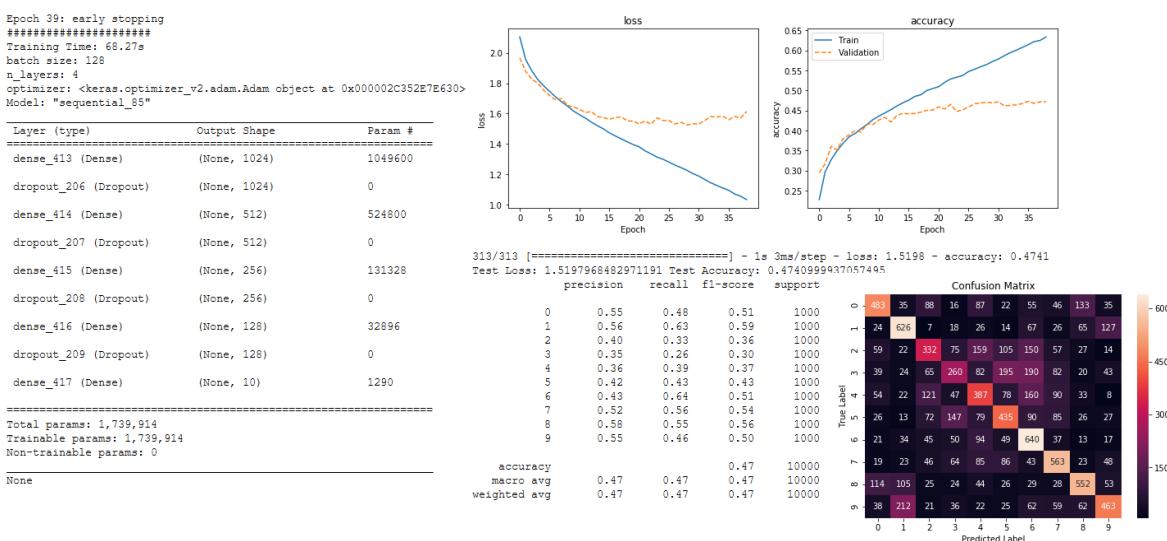
شكل 7 - نتیجه اندازه دسته 64

نتیجه اندازه 128 :

● خطأ: 1.519

● دقت: 0.474

● زمان کل آموزش: 1.74 در 39 اپیک (68.27s در اپیک)



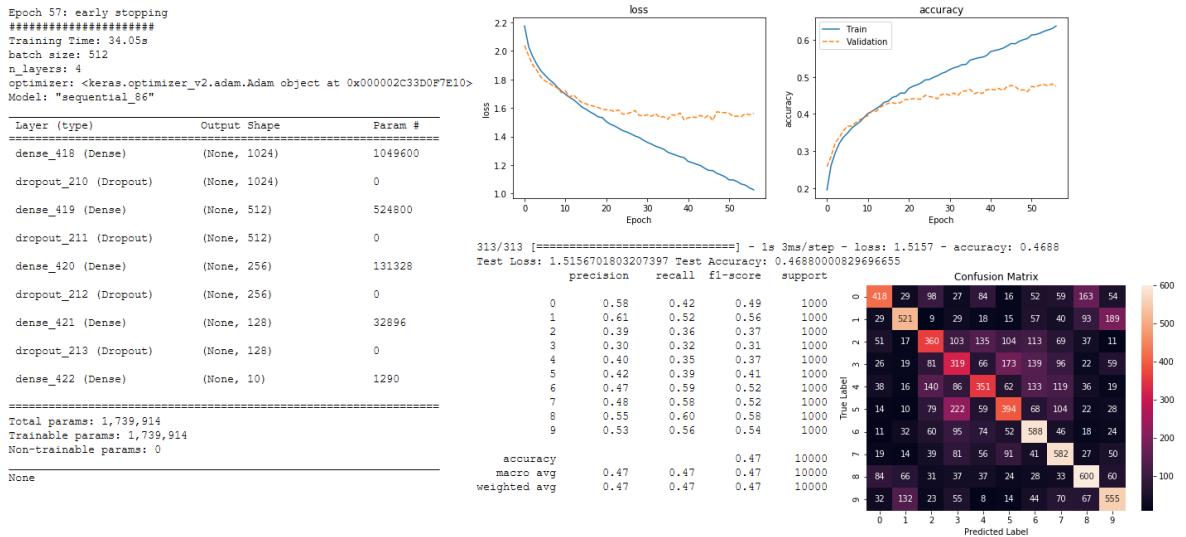
شكل 8 - نتیجه اندازه دسته 128

نتیجه اندازه 512:

● خطأ: 1.515

● دقت: 0.468

● زمان کل آموزش: 34.05s در 57 اپیک (0.59 در اپیک)

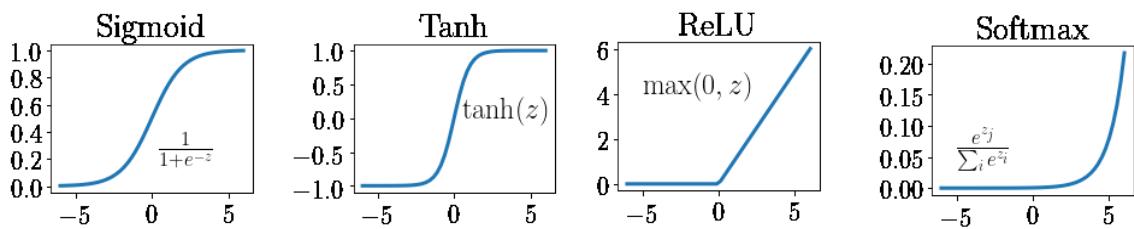


شکل 9 - نتیجه اندازه دسته 512

به صورت کلی با افزایش اندازه دسته، شاهد کاهش خطأ و افزایش دقت هستیم. علت این مورد را می‌توان این دانست که با این کار و هر چه به batch gradient descent واقعی نزدیک شویم، حرکت نرم تری برای حرکت به سمت مینیمم لاس خواهیم داشت و از حرکت‌های اشتباه یا رندوم دورتر می‌شویم. از طرفی زمان اجرای یک اپیک با افزایش اندازه دسته ها کاهش می‌یابد. زیرا هر دسته به صورت پارالل اجرا می‌شود و پس از آن یک step انجام می‌شود. بنابراین هر چه اندازه دسته بزرگتر باشد (تا جایی که GPU تواناییش را داشته باشد) باز هم به علت اجرای موازی تغییر خاصی در سرعت اجرای بچ نداریم، اما چون سایر بچ بزرگتر شده است، کل اپیک را سریعتر طی می‌کنیم. البته باید این را نیز در نظر داشت که هر چه اندازه بچ بزرگتر باشد یعنی تعداد step های کمتر در یک اپیک انجام می‌شود. در مجموع اندازه دسته 128 بهترین دقت را داشت و با این مقدار ادامه می‌دهیم.

ه) چهار تابع فعال ساز { **ReLU** , **Tanh** , **Sigmoid** , **Softmax** } در نظر بگیرید، سپس مزایا و معایب این تابع فعال ساز را نسبت به دیگری بیان نمایید. حال با استفاده از بهترین مدل قسمتهای قبل، توابع فعالساز هر کدام را تغییر دهید تاثیر توابع فعالسازی را در خطای دقت و زمان آموزش شبکه بررسی نمایید.

توابع گفته شده در زیر آمده‌اند.



شکل 10 - توابع فعال ساز

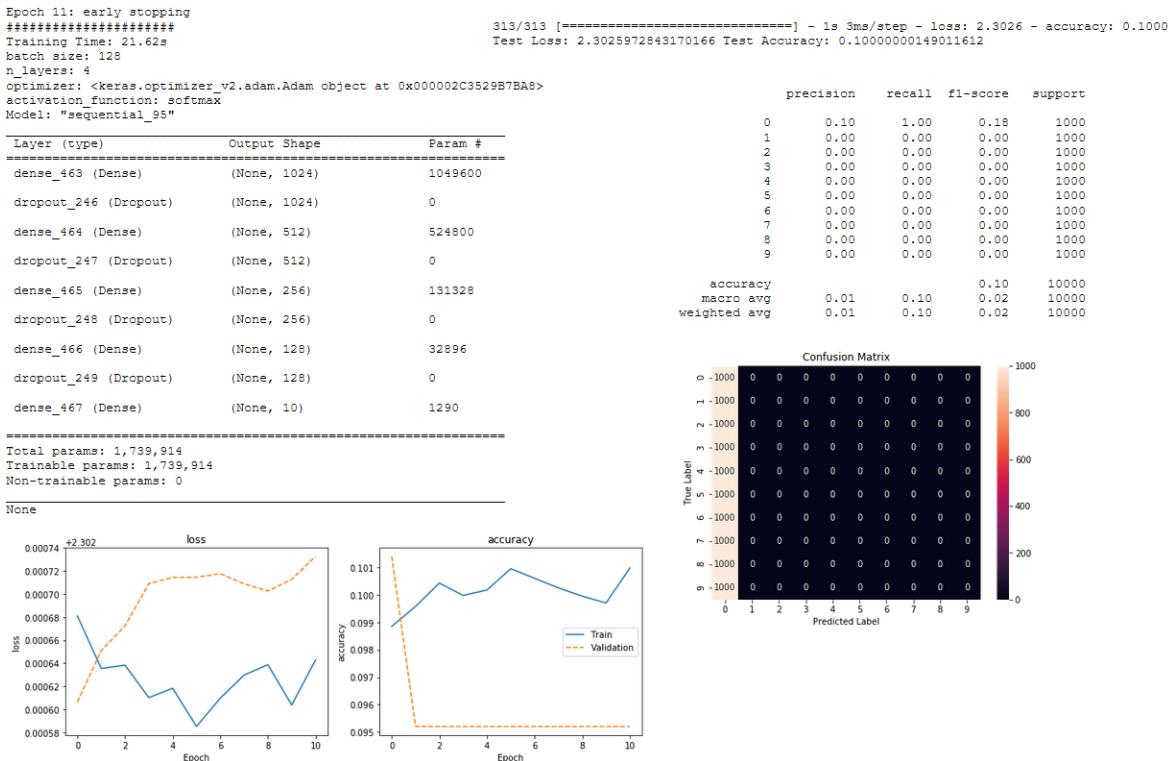
همانطور که دیده می‌شود از مزایای Sigmoid و Tanh این است که خروجی را محدود به 0 تا 1 یا -1 تا 1 می‌کنند. در حالیکه ReLU اجازه می‌دهد از طرف مثبت مقدار تا بینهایت برود. البته همین مورد را می‌توان قوت ReLU هم دانست چون در Sigmoid و Tanh وقته مقدار از 0 دور شود چه به سمت مثبت چه منفی دیگر یک خط صاف داریم که مشتق آن اطلاعات مفیدی در مورد اینکه مدل باید در چه جهتی حرکت کند نمی‌دهد و مشتقش نزدیک 0 می‌شود. در حالیکه ReLU این مشکل را ندارد و بهتر می‌تواند مدل را حتی در مقادیر دور از 0 راهنمایی کند. از طرفی Softmax معمولاً در لایه آخر مدل استفاده می‌شود و از مزایای آن این است که کاری می‌کند که مجموع خروجی‌ها یک شود. این مورد برای خروجی نهایی مدل خیلی مفید است زیرا در مسائل طبقه‌بندی که در انها می‌خواهیم چیزی مشابه احتمال کلاس‌ها را داشته باشیم این تابع می‌تواند این مقادیر که مجموعشان 0 شود و همگی بزرگتر یا مساوی 0 باشند را بدهد. اما همانطور که در شکل هم دیده می‌شود از نظر مشتق و آموزش مدل مناسب نیست و همگرایی را سخت می‌کند.

نتیجه softmax :

● خطای 2.302

● دقته 0.10

● زمان کل آموزش: 21.62s در 11 اپیک (1.90 در اپیک)



شکل 11 - نتیجه softmax

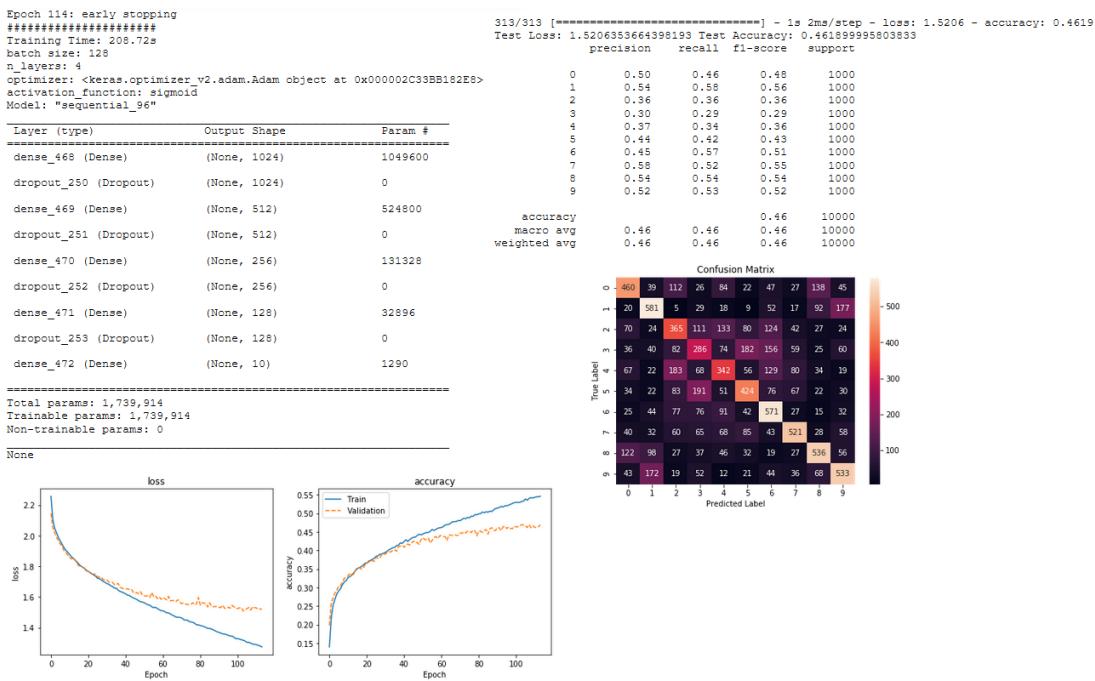
همانطور که توضیح داده شد استفاده از softmax فقط در لایه آخر استفاده می شود. اگر در لایه های میانی استفاده شود بخاطر شکلی که دارد، آموزش اصلا همگرا نمی شود و همانطور که دیده می شود نتیجه مطلوبی نمی گیریم.

نتیجه Sigmoid:

● خطای 1.520

● دقته 0.461

● زمان کل آموزش: 208.72s در 114 اپیک (1.82 در اپیک)



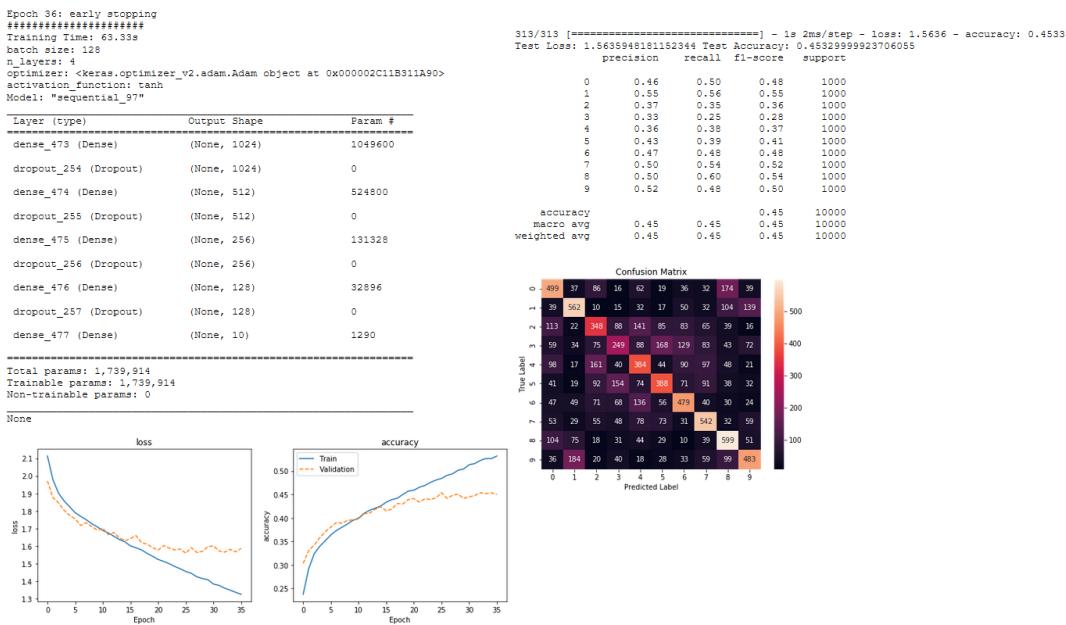
شکل 11 - نتیجه sigmoid

:Tanh نتیجه

خطا: 1.563 ●

دقت: 0.453 ●

زمان کل آموزش: 63.33s در 36 اپیک (1.75 در اپیک) ●



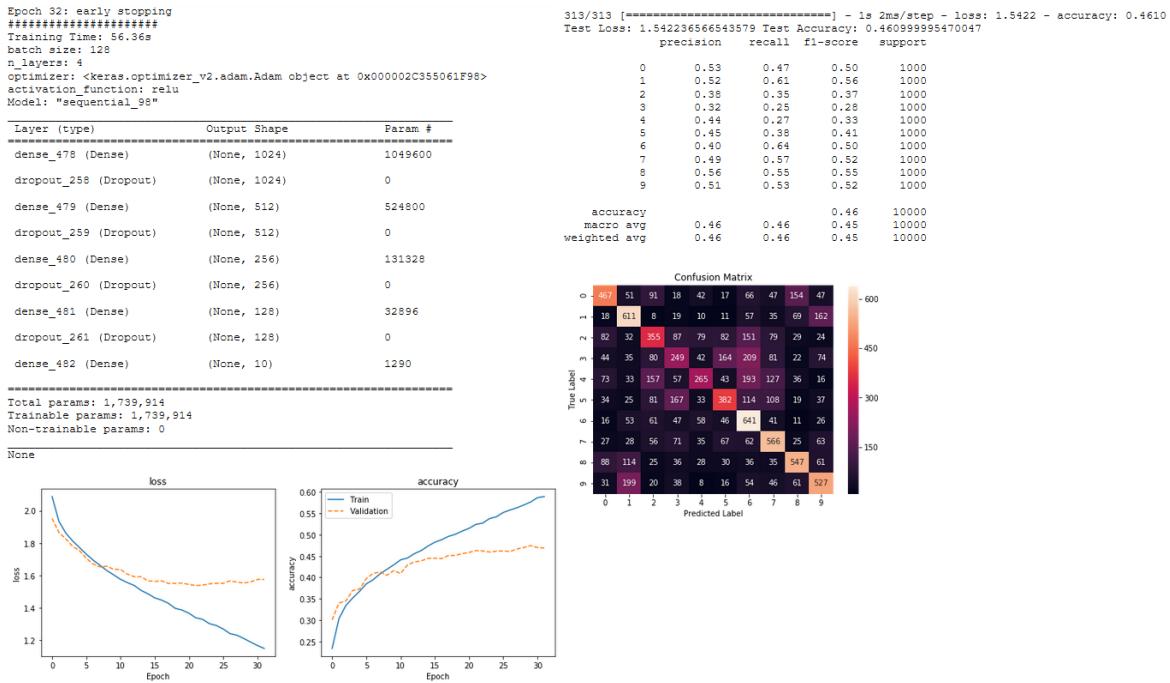
شکل 12 - نتیجه tanh

نتیجه ReLU

● خطای 1.542

● دقت: 0.460

● زمان کل آموزش: 56.36s در 32 اپیک (1.75 در اپیک)



شکل 13 - نتیجه ReLU

همانطور که توضیح داده شد، با این آزمایش‌ها مشخص‌تر شد که softmax برای لایه‌های میانی مناسب نیست. از طرفی همانطور که توضیح داده شد که sigmoid وقتی از 0 دور شود مشتق نزدیک به 0 می‌شود، دیده شد که آموزش بسیار کندر شد و 114 اپیک نیاز پیدا کرد تا اجرا شود که زمان خیلی بیشتری نسبت به بقیه است. در مقایسه، ReLU توانست از بقیه دقت نسبتاً بهتری به دست آورد و همچنین این کار را در زمان کمتری انجام داد که به همین دلایل در ادامه از آن استفاده می‌کنیم. ضمناً این تابع و مشابه آن یعنی Leaky ReLU از توابع پر استفاده هستند که در تحقیقات و مدل‌های امروزی بسیار استفاده می‌شوند و اینجا هم دیدیم طبق نتایج علت آن چیست.

و) سه تابع خطا را در نظر بگیرید، سپس با استفاده از بهترین مدل قسمتی‌ای قبل، تابع خطای شبکه را تغییر دهید و تاثیر تابع خطا را بررسی نمایید و نتایج آن را در گزارش بیاورد.

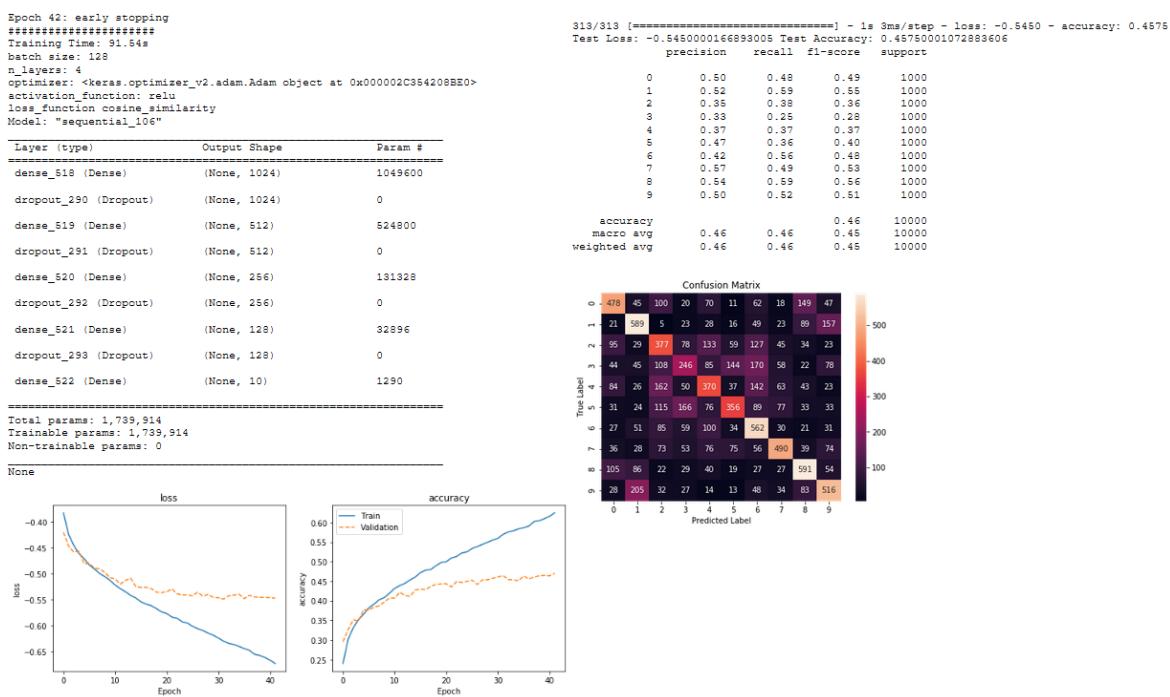
در این بخش از توابع خطای cosine_similarity, kl_divergence, mean_squared_error, categorical_crossentropy فاصله دو توزیع را محاسبه کنند که با حرکت در جهت عکس آن بتوانیم مدل خوبی به دست آوریم. اما با توجه به کاربرد و مسئله، ممکن است این تابع را بتوان خاص منظوره طراحی کرد. مثلا cosine similarity بیشتر برای نزدیک کردن دو وکتور در فضای چند بعدی استفاده می‌شود و crossentropy برای مسائلی که کلاس بندی دارند و برچسب نهایی 0 و 1 است. همچنین KL Divergence در بعضی از موارد knowledge distillation کاربرد دارد که فاصله دو مدل بزرگ و کوچک را کم کند. از طرفی MSE بیشتر در مسائل Regression کاربرد دارد که برچسب‌ها پیوسته می‌توانند باشند.

نتیجه : cosine_similarity

خطا: 0.545 ●

دقت: 0.457 ●

زمان کل آموزش: 91.54s در 42 اپیاک ●

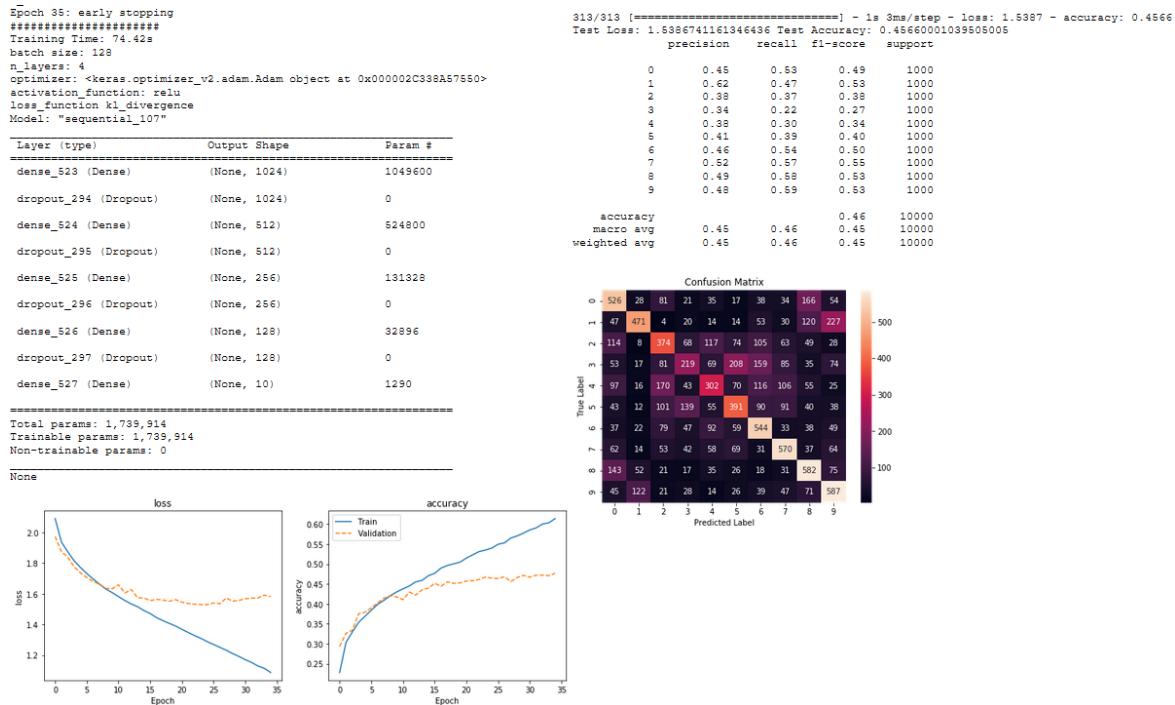


نتیجه : kl_divergence

خطا : 1.538 ●

دقت : 0.456 ●

زمان کل آموزش : 74.42s در 35 اپاک ●



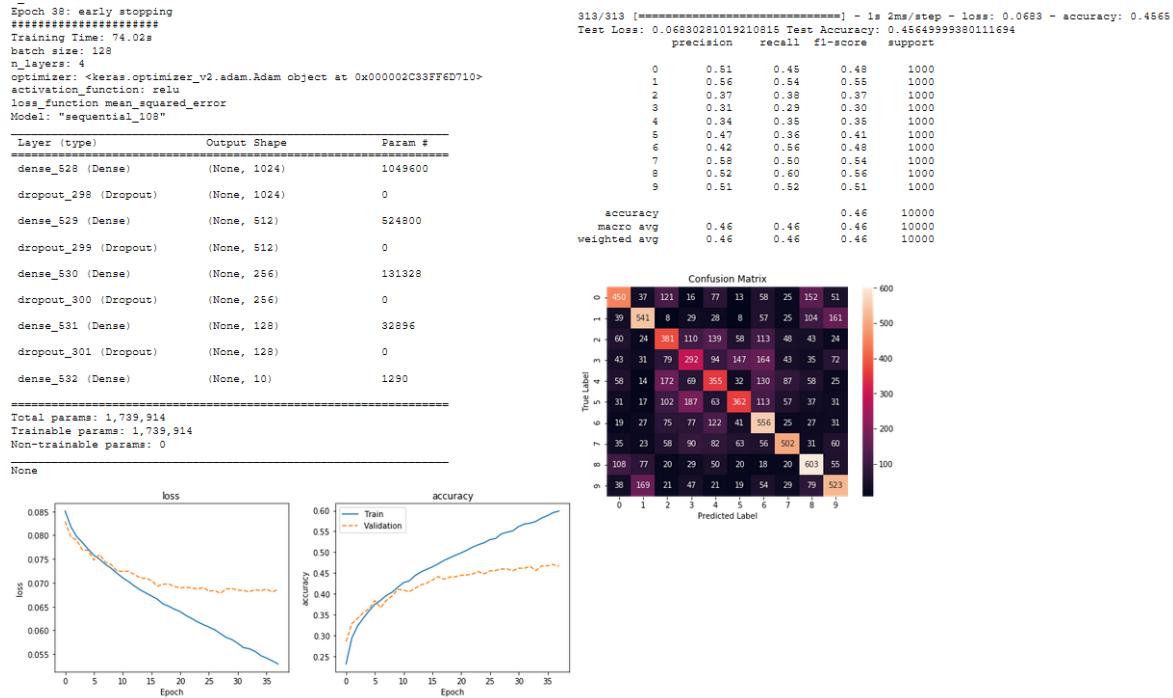
نمایه 15 - نتیجه kl_divergence

نتیجه : mean_squared_error

خطا : 0.068 ●

دقت : 0.456 ●

زمان کل آموزش : 74.02s در 38 اپاک ●



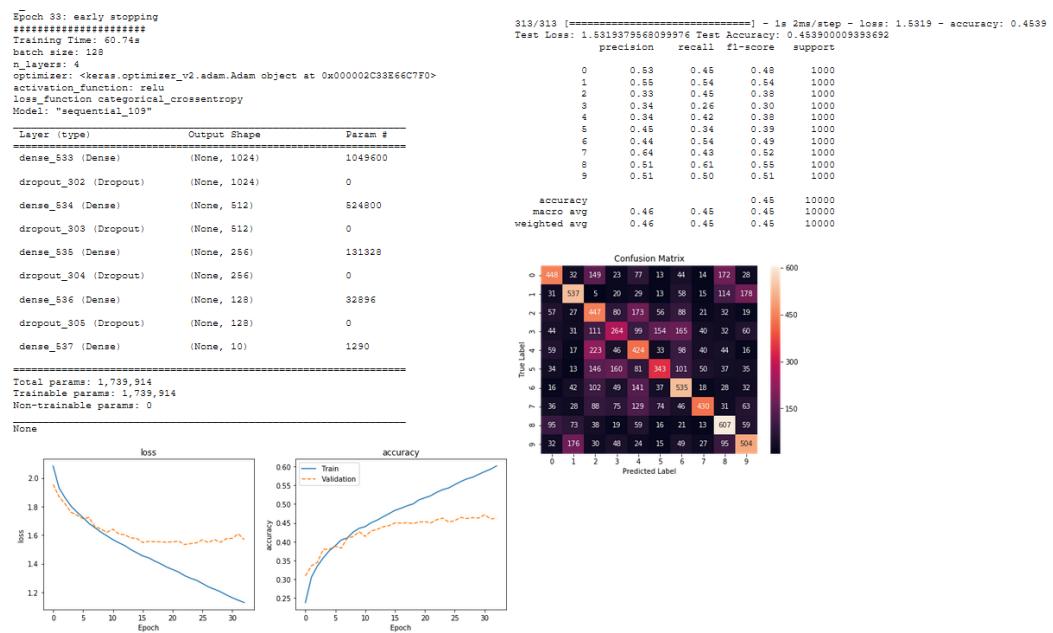
شكل 16 - نتیجه MSE

: categorical_crossentropy نتیجه

خطا: 1.531 ●

دقت: 0.453 ●

زمان کل آموزش: 60.74s در 33 اپاک ●



شكل 17 - نتیجه categorical_crossentropy

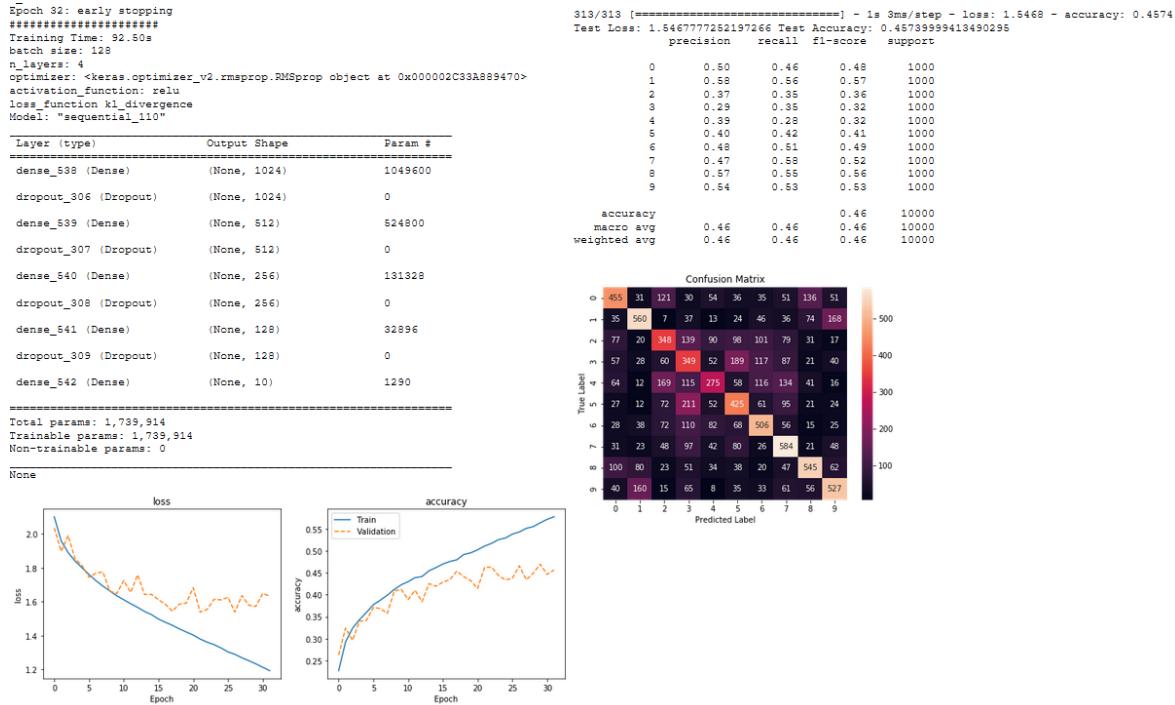
اولاً دیده می‌شود که مقدار خطاهای بسیار متفاوت هستند که علت آن تابع‌های متفاوت و بازه‌های متفاوت آن‌ها است. تنها مهم است که با کاهش لاس به سمت درستی حرکت کنیم و کاهش آن مهم است نه مقدار خالص آن. از طرفی cosine similarity که کاربرد آن بالاتر توضیح داده شد بدترین نتیجه را داشته و می‌توان آن را به همین دلیل دانست. اما باقی توابع همگی تابع مناسبی بودند و توانستند نتایج خوبی هم بگیرند. البته باید توجه داشت که در تحقیقات ماشین لرینگ اکثراً یک آزمایش چندین بار اجرا می‌شود و میانگین اجراهای به همراه sd آن گزارش می‌شود تا از نتایج مطمئن باشیم. بنابراین ممکن است این نتایج در چندین بار اجرا متفاوت هم شود اندکی و این تفاوت کم بین سه تابع دیگر به نظر چشمگیر و از نظر آماری بزرگ به نظر نمی‌رسند. اما به هر حال بهترین نتیجه از $kl_divergence$ به دست آمده که در ادامه از آن استفاده می‌کنیم.

ج) سه تابع بهینه ساز را درنظر بگیرید، سپس با استفاده از بهترین مدل قسمت‌های قبل، بهینه ساز شبکه تغییردهید تاثیر بهینه سازهای متفاوت را بررسی نمایید و نتایج آن را در گزارش بیاورید.

در این بخش از ۳ تابع بهینه‌ساز RMSprop و SGD و همچنین ADAM که به شکلی ترکیب این دو است استفاده می‌کنیم.

نتیجه : RMSprop

- خطأ: 1.5467
- دقت: 0.4573
- زمان کل آموزش: 92.50s در 32 اپاک



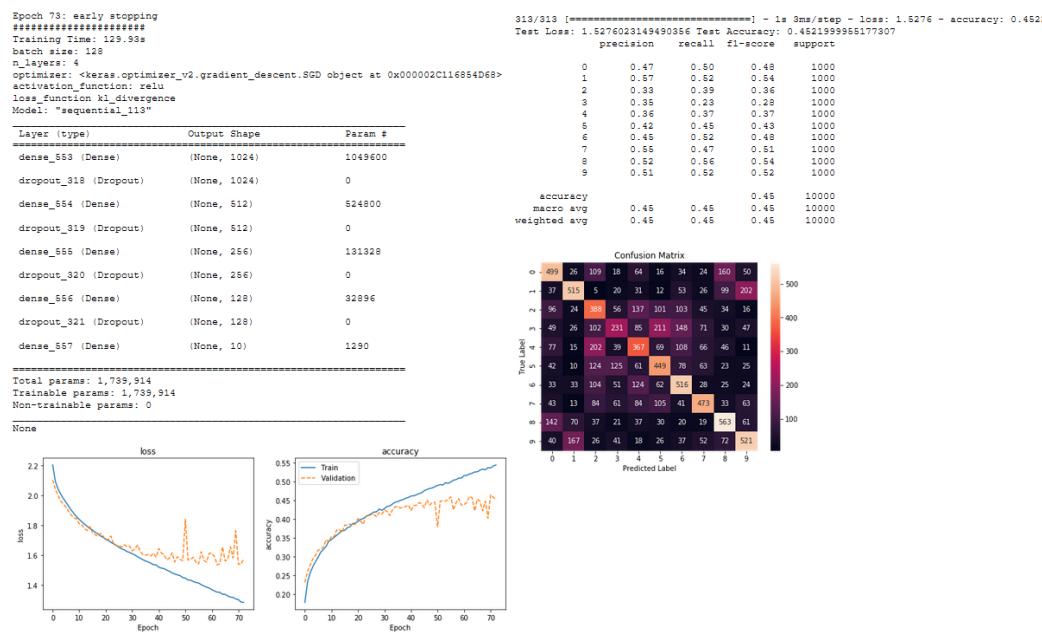
شکل 18 - نتیجه RMSprop

: SGD نتیجه

● خطای 1.5276

● دقت: 0.4521

● زمان کل آموزش: 129.93s در 73 اپیاک



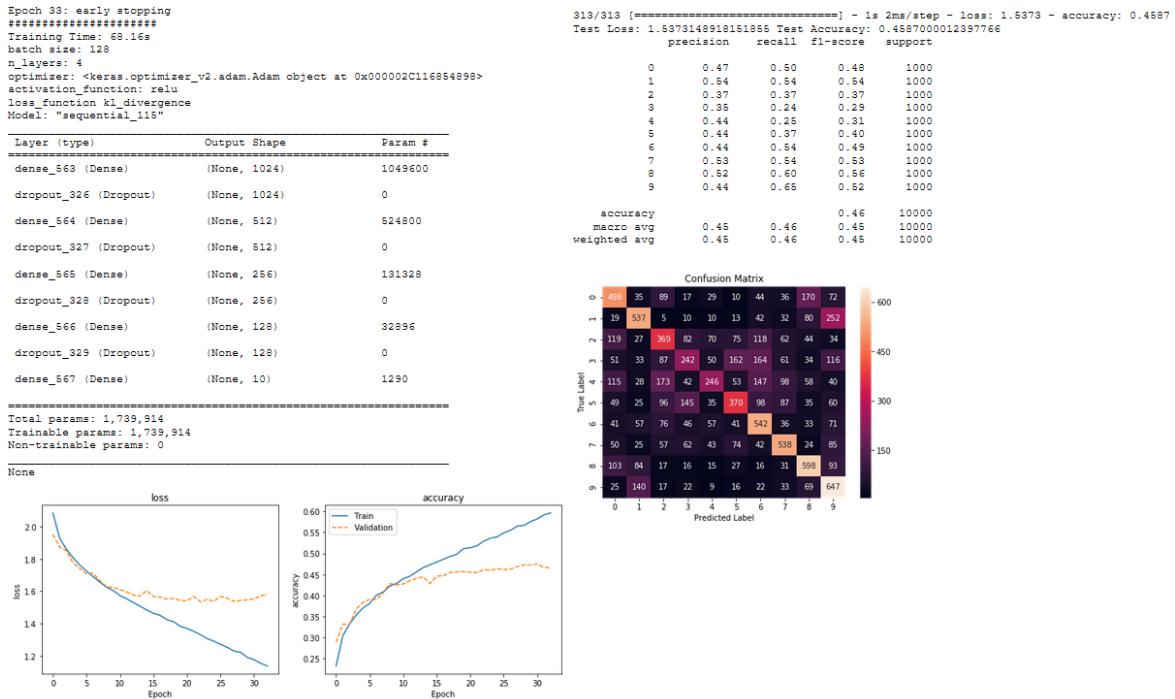
شکل 19 - نتیجه SGD

نتیجه :ADAM

● خطأ: 1.5373

● دقت: 0.4587

● زمان کل آموزش: 68.16s در 33 اپیک



شکل 20 - نتیجه ADAM

در مجموع می‌بینیم که به ترتیب SGD و RMSProp و ADAM بدترین تا بهترین نتایج را به دست آورده‌اند. هم از نظر دقت و خطأ و هم از نظر سرعت همگرايی. طبیعتاً در مقالات این توابع توضیحات تکمیلی در مورد آنها و ریزه کاری‌های آنها آمده است ولی کافیست بدانیم که ADAM به نوعی ترکیبی از دو بهینه‌ساز دیگر است و طبیعتاً بهتر از آنها عمل می‌کند و همچنین در اکثر تحقیقات امروزی نیز این بهینه ساز معروف ترین بهینه سازی است که استفاده می‌شود و نتایج خوبی را با سرعت بالاتر به ما می‌دهد. بنابراین در ادامه با این بهینه ساز ادامه می‌دهیم.

ح) با توجه به بهترین مدل قسمت‌های قبل، افزودن لایه به شبکه تاثیری در خروجی دارد؟ فرضیه خود را با تغییر تعداد لایه‌های مختلف (سه مرتبه) بررسی کنید.

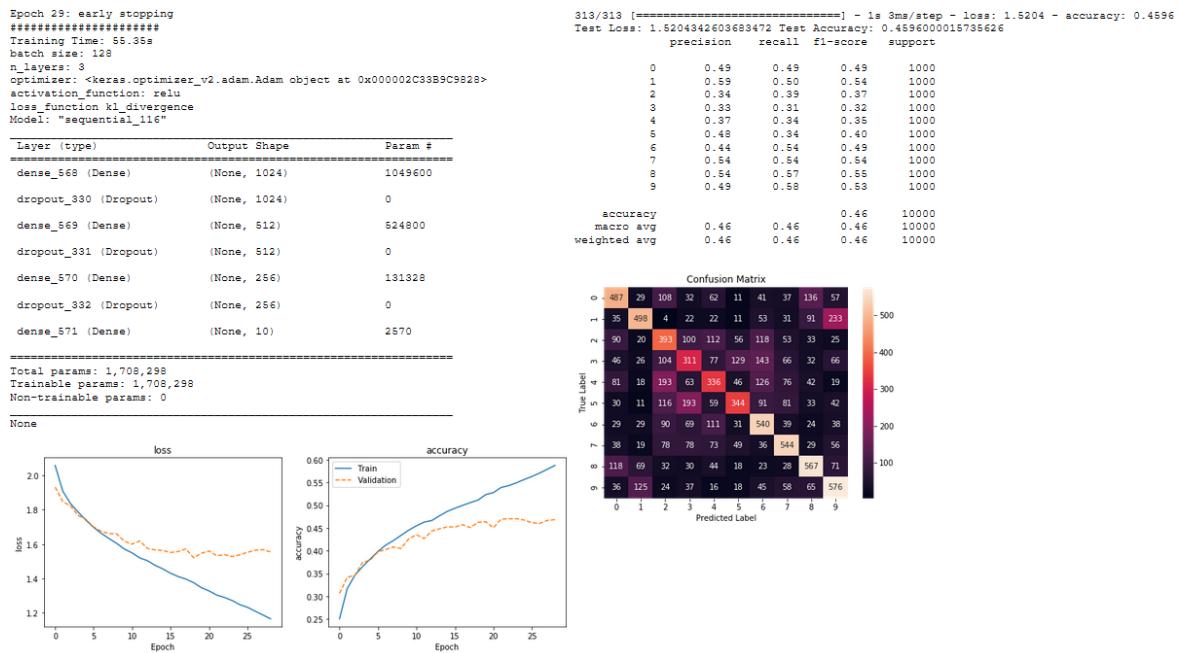
در این بخش تعداد لایه‌های 3 و 4 و 5 را آزمایش می‌کنیم.

نتیجه 3 لایه:

خطا: 1.5204 ●

دقت: 0.4596 ●

زمان کل آموزش: 55.35s در 29 اپیک



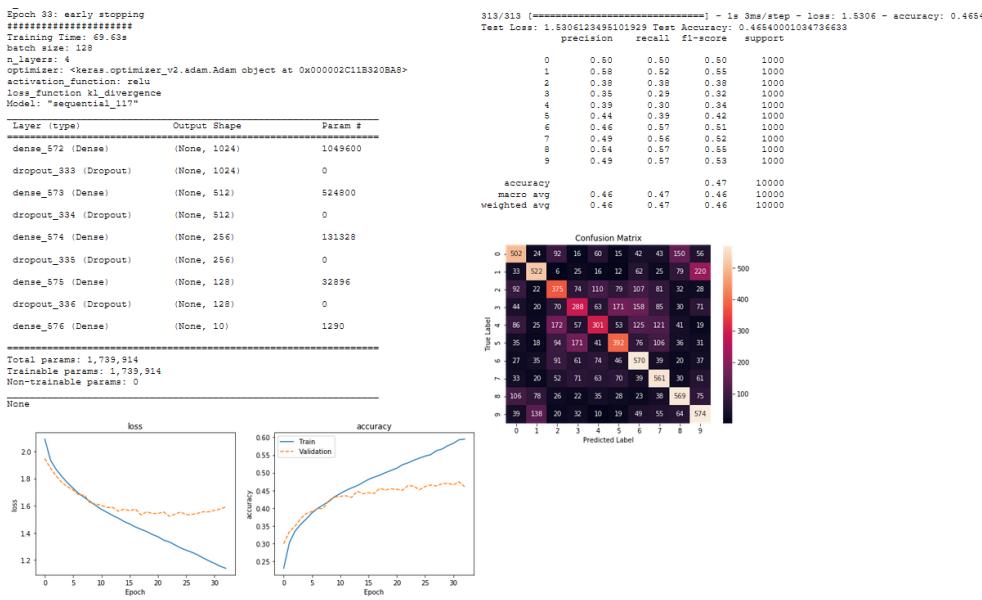
شکل 21 - نتیجه 3 لایه

نتیجه 4 لایه:

خطا: 1.5306 ●

دقت: 0.4654 ●

زمان کل آموزش: 69.63s در 33 اپیک



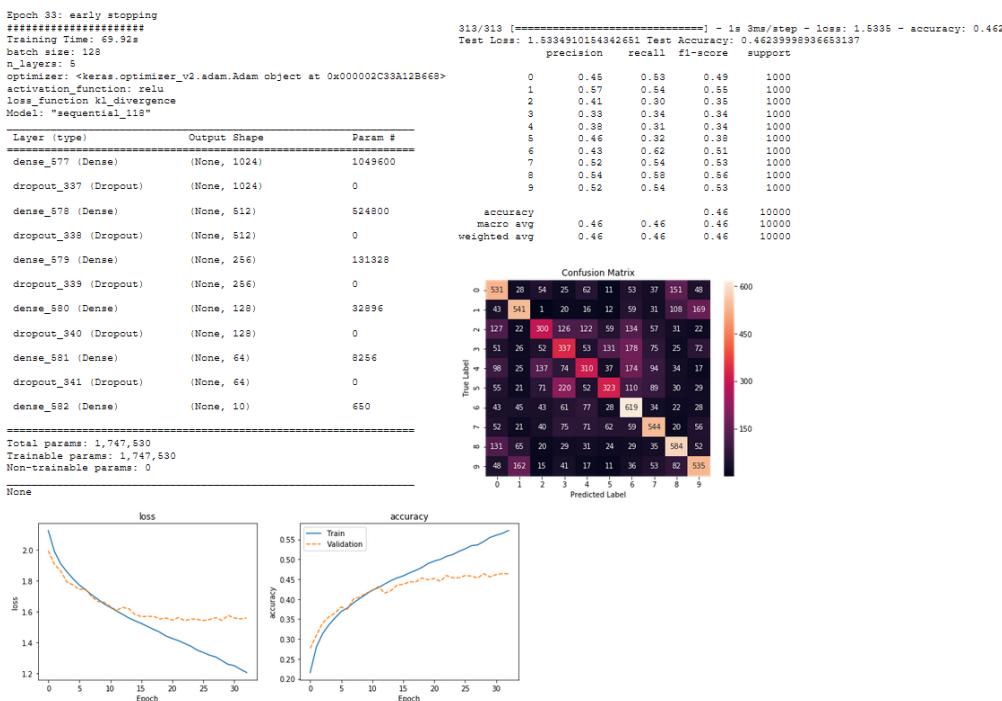
شکل 22 - نتیجه 4 لایه

نتیجه 5 لایه:

خطا: 1.5334 ●

دقت: 0.4623 ●

زمان کل آموزش: 69.92s در 33 اپیاک ●



شکل 23 - نتیجه 5 لایه

از آنجا که در ابتداء هم 4 لایه انتخاب شده بود و نسبتاً عمیق است، اضافه کردن به 4 لایه برای این مسئله نسبتاً ساده تغییر خاصی نخواهد داشت. همانطور که دیده می‌شود بهترین نتیجه برای 4 لایه و سپس 5 لایه و در آخر 3 لایه است. این مورد نشان می‌دهد که 3 لایه برای این مسئله کم است و هنوز جا داریم که مدل را پیچیده تر کنیم اما در عین حال 5 لایه نیز بیش از حد است و باعث بدتر شدن generalization و overfitting نسبی می‌شود و به همین دلیل 4 لایه که در این بین است مقدار معقولی به نظر می‌رسد چنانکه بیشتر از آن مدل را پیچیده می‌کند.

(ط) با توجه به ارزیابی‌های انجام شده، انتخاب کدام پارامترها بهترین نتیجه را میدهد؟ دلیل خود را با توجه به قسمت‌های قبل بطور کامل شرح دهید.

ویژگی‌های نهایی در زیر آمده است.

- neurons: 1024 (Trainable params: 1,739,914)
- batch size: 128
- activation_function: relu
- loss_function: kl_divergence
- optimizer: Adam
- n_layers: 4

در مورد تعداد نورون دیدیم که تعداد کم باعث underfitting و کمبود توان مدل برای مدلسازی می‌شود و بر عکس آن یعنی تعداد نورون بسیار زیاد باعث overfitting و کاهش generalization مدل می‌شود و در هر دو حالت، دقت تست کاهش می‌یافتد. به همین دلیل عددی میانه یعنی 1024 انتخاب شد. در مورد اندازه دسته‌ها نیز دیدیم که اندازه کم باعث کندتر اجرا شدن اپیاک‌ها اما افزایش تعداد step‌ها می‌شد و بر عکس آن اندازه بزرگ باعث می‌شد step‌ها کمتری داشته باشیم. باز هم عددی میانه بهترین نتیجه را دارد و دلیل واقعی آن تحقیقات به روزی هستند که نشان می‌دهند عدد خیلی بزرگ بچ هم مناسب نیست، اما حرکت کردن به آن سمت که باعث می‌شود نرم تر به سمت مینیمم لاس حرکت کنیم مناسب است. در انتها باز هم عددی میانه یعنی 128 انتخاب شد که بهترین نتیجه را داشت. در مورد activation function‌ها دیدیم که توابع دیگر بعضًا مشکل مشتق نزدیک 0 در مقادیر بزرگ را داشتند که باعث کندی آموزش می‌شد و همچنین از softmax نمی‌توانستیم در لایه‌های میانی استفاده کنیم چون همگرایی رخ نمیداد و بهترین تابع که هیچکدام از این مشکلات را نداشت ReLU بود که در

تحقیقات اخیر هم اکثرا همین تابع و بعضًا تابع Leaky ReLU استفاده می‌شود. برای انتخاب loss function باید دقت داشت که مسئله از چه جنسی است و همانطور که امتحان کردیم مثلاً نزدیک کردن cos similarity برای این مسئله خوب نبود و در انتهای kl divergence که از پشت این لاس و مسئله ما که از جنس طبقه بندی بود با هم به خوبی کار می‌کنند. از نظر بهینه‌ساز ابتدا دو اجزای سازنده ADAM را امتحان کردیم که کندر بودند و همچنین نمی‌توانستند بهترین نتیجه را بگیرند. اما ADAM که ترکیب این دو را دارد و در اکثر تحقیقات به روز هم استفاده می‌شود هم سریعتر بود و هم دقت بهتری را به ما می‌داد. از نظر تعداد لایه نیز می‌دانیم که مدل با یک لایه پنهان می‌تواند هر تابعی را مدل سازی کند اما تعداد نورون زیاد دریک لایه همچنین می‌تواند باعث overfitting شود. بنابراین لایه‌های بیشتر می‌تواند کمک کند توابع پیچیده‌تری با تعداد نورون کمتری مدل سازی شوند. البته دیدیم که افزایش بیش از حد تعداد لایه‌ها نیز مفید نخواهد بود و باید با توجه به مسئله مدلی با پیچیدگی متناسب انتخاب کرد که اینجا 4 لایه بهترین بود.

ن) بهترین مدل به دست آمده، ادریک جدول بیان کرده و خطای دقت و زمان لازم برای آموزش شبکه و همچنین دقت و خطای داده‌های تست را در آن گزارش نمایید.

جدول 2 - بهترین مدل به دست آمده

neurons	1024 (Trainable params: 1,739,914)
batch size	128
activation_function	ReLU
loss_function	kl_divergence
optimizer	ADAM
n_layers	4

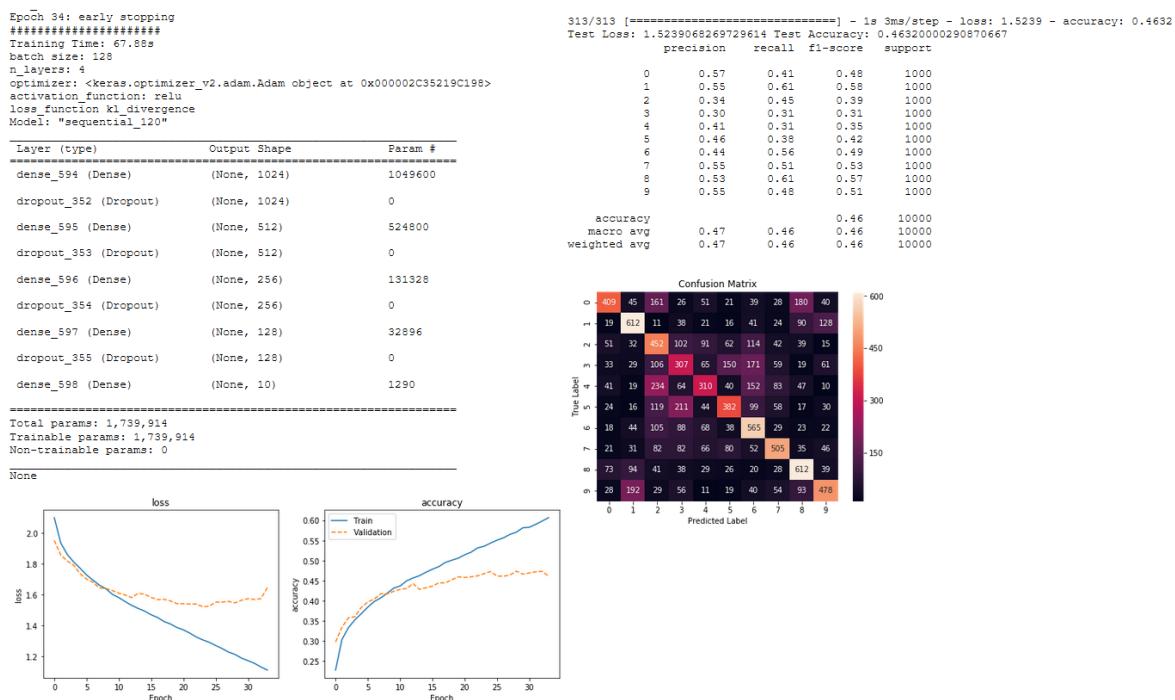
نتیجه بهترین اجرا:

- خطای تست: 1.5239

- دقت تست: 0.4632

- زمان آموزش: 67.88s در 34 اپیاک

همچنین ساختار شبکه، نمودار خطای و دقت در طول آموزش روی داده آموزش و ارزیابی در زیر آمده است.



شکل 24 - نتیجه بهترین مدل

لازم به ذکر است که همانطور که قبل تر هم توضیح داده شد از خطای early stopping بخش ارزیابی با میزان صبر 10 اپیاک استفاده شده است و در آخر نیز بهترین مدل از نظر خطای ارزیابی استفاده می‌شود. در نمودارها هم دیده می‌شود که جایی که دیگر لاس ارزیابی در حال بالا رفتن است (overfitting) دیگر متوقف می‌شویم تا generalization مدل را از دست ندهیم. همچنین باید دقت داشت که چون 10 کلاس داریم، دقت مدل رندوم 10 درصد خواهد بود که دقت 46 این مدل خوب است. البته با استفاده از CNN ها می‌توان دقت‌های خیلی بیشتر گرفت اما در این تمرین به FFN محدود بودیم.

سوال 2 MLP Regression – 2

الف) ابتدا پیش پردازش های لازم برای آماده سازی دیتا را انجام دهید و آنها را توضیح دهید. برای هر مرحله، توضیحات مختصری ارائه کنید. توجه داشته باشید در این قسمت مجاز به حذف هیچ یک از ابعاد داده نیستید.

ابتدا داده را با df = pd.read_csv ('data.csv') خواندیم و سپس با استفاده از df[col].unique تعداد مقادیر یکتای هر ستون موجود را بررسی کردیم.

(date, 70), (price, 1741), (bedrooms, 10), (bathrooms, 26), (sqft_living, 566),
(sqft_lot, 3113), (floors, 6), (waterfront, 2), (view, 5), (condition, 5),
(sqft_above, 511), (sqft_basement, 207), (yr_built, 115), (yr_renovated, 60),
(street, 4525), (city, 44), (statezip, 77), (country, 1)

از این مقادیر، ستونهای street و city و statezip متغیر هستند و date نیز است. بنابراین این ستونها باید اصلاح شوند. برای موارد متغیر که در اصل categorical هستند از get_dummies³ استفاده می کنیم و آنها را به ستونهای مجزا به شکل onehot در می آوریم. اما در مورد date، ابتدا به datetime تبدیل می کنیم و سپس با استفاده از value مقدار عددی آن را به دست می آوریم.

In [9]:	# Objects to Numbers df = raw_df.copy() CATEGORICAL_COLS = ["street", "city", "statezip", "country", "condition"] for col in CATEGORICAL_COLS: one_hot_df = pd.get_dummies(df[col], prefix=col) df = pd.concat([df.drop(columns=col), one_hot_df], axis=1) df["date"] = pd.to_datetime(df["date"]).apply(lambda x: x.value) df																																																																																																																																																												
Out[9]:	<table border="1"><thead><tr><th>view</th><th>sqft_above</th><th>...</th><th>statezip_WA_98198</th><th>statezip_WA_98199</th><th>statezip_WA_98288</th><th>statezip_WA_98354</th><th>country_USA</th><th>condition_1</th><th>condition_2</th><th>condition_3</th><th>condition_4</th><th>condition_5</th></tr></thead><tbody><tr><td>0</td><td>1340</td><td>...</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td></tr><tr><td>4</td><td>3370</td><td>...</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1930</td><td>...</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>0</td><td>1000</td><td>...</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>0</td><td>1140</td><td>...</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td></tr><tr><td>0</td><td>1510</td><td>...</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr><tr><td>0</td><td>1460</td><td>...</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td></tr><tr><td>0</td><td>3010</td><td>...</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1070</td><td>...</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1490</td><td>...</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr></tbody></table>	view	sqft_above	...	statezip_WA_98198	statezip_WA_98199	statezip_WA_98288	statezip_WA_98354	country_USA	condition_1	condition_2	condition_3	condition_4	condition_5	0	1340	...	0	0	0	0	1	0	0	1	0	0	4	3370	...	0	0	0	0	1	0	0	0	0	1	0	1930	...	0	0	0	0	1	0	0	0	1	0	0	1000	...	0	0	0	0	1	0	0	0	1	0	0	1140	...	0	0	0	0	1	0	0	0	1	0	0	1510	...	0	0	0	0	1	0	0	0	1	0	0	1460	...	0	0	0	0	1	0	0	1	0	0	0	3010	...	0	0	0	0	1	0	0	1	0	0	0	1070	...	0	0	0	0	1	0	0	1	0	0	0	1490	...	0	0	0	0	1	0	0	0	1	0
view	sqft_above	...	statezip_WA_98198	statezip_WA_98199	statezip_WA_98288	statezip_WA_98354	country_USA	condition_1	condition_2	condition_3	condition_4	condition_5																																																																																																																																																	
0	1340	...	0	0	0	0	1	0	0	1	0	0																																																																																																																																																	
4	3370	...	0	0	0	0	1	0	0	0	0	1																																																																																																																																																	
0	1930	...	0	0	0	0	1	0	0	0	1	0																																																																																																																																																	
0	1000	...	0	0	0	0	1	0	0	0	1	0																																																																																																																																																	
0	1140	...	0	0	0	0	1	0	0	0	1	0																																																																																																																																																	
...																																																																																																																																																	
0	1510	...	0	0	0	0	1	0	0	0	1	0																																																																																																																																																	
0	1460	...	0	0	0	0	1	0	0	1	0	0																																																																																																																																																	
0	3010	...	0	0	0	0	1	0	0	1	0	0																																																																																																																																																	
0	1070	...	0	0	0	0	1	0	0	1	0	0																																																																																																																																																	
0	1490	...	0	0	0	0	1	0	0	0	1	0																																																																																																																																																	

شکل 25 - تبدیل ابجکت ها به مقادیر عددی

³ https://pandas.pydata.org/docs/reference/api/pandas.get_dummies.html

سپس به سراغ مقادیر missing می‌رویم.

```
In [14]: # Missing Values
print(raw_df.isnull().sum())
print(raw_df[raw_df==0].count())

date          0           date          0
price         0           price        49
bedrooms      0           bedrooms     2
bathrooms     0           bathrooms    2
sqft_living   0           sqft_living  0
sqft_lot      0           sqft_lot     0
floors        0           floors       0
waterfront    0           waterfront  4567
view          0           view         4140
condition     0           condition    0
sqft_above    0           sqft_above   0
sqft_basement 0           sqft_basement 2745
yr_built      0           yr_built     0
yr_renovated  0           yr_renovated 2735
street        0           street       0
city          0           city         0
statezip      0           statezip    0
country       0           country     0
dtype: int64
```

شکل 26 - بررسی مقادیر Missing

ابتدا می‌بینیم که هیچ مقدار nan یا null در داده‌ها نیست. اما از نظر تعداد 0 در price و missing yr_renovated می‌بینیم که 0 موجود است که امکان پذیر نیست. بنابراین اینها مقادیر missing هستند. برای price چون متغیر اصلی که می‌خواهیم حدس بزنیم و فقط 49 تا 0 دارد، آن سطرها را کاملاً حذف می‌کنیم. اما در مورد yr_renovated چون تعداد 0 ها زیاد است، حذف کردن گزینه خوبی نیست و به جای این کار، میانگین بقیه را در خانه‌هایی که 0 است می‌گذاریم. برای این کار از کد زیر استفاده می‌کنیم.

```
In [25]: df = df.loc[df['price'] != 0]
df['yr_renovated'] = df['yr_renovated'].replace(0, np.NaN)
df['yr_renovated'].fillna((df['yr_renovated'].mean()), inplace=True)
df
```

```
Out[25]:
   date      price  bedrooms  bathrooms  sqft_living  sqft_lot  floors  waterfront  view  sqft_above ... statezip_WA
0  13989888000000000000  3.130000e+05    3.0     1.50     1340    7912     1.5       0       0     1340 ...
1  13989888000000000000  2.384000e+06    5.0     2.50     3650    9050     2.0       0       4     3370 ...
2  13989888000000000000  3.420000e+05    3.0     2.00     1930   11947     1.0       0       0     1930 ...
3  13989888000000000000  4.200000e+05    3.0     2.25     2000    8030     1.0       0       0     1000 ...
4  13989888000000000000  5.500000e+05    4.0     2.50     1940   10500     1.0       0       0     1140 ...
...
4595 14048640000000000000  3.081667e+05    3.0     1.75     1510    6360     1.0       0       0     1510 ...
4596 14048640000000000000  5.343333e+05    3.0     2.50     1460    7573     2.0       0       0     1460 ...
4597 14048640000000000000  4.169042e+05    3.0     2.50     3010    7014     2.0       0       0     3010 ...
4598 14049504000000000000  2.034000e+05    4.0     2.00     2090    6630     1.0       0       0     1070 ...
4599 14049504000000000000  2.206000e+05    3.0     2.50     1490    8102     2.0       0       0     1490 ...

4551 rows × 4665 columns
```

```
In [27]: print(df[df==0].count()[:15])
date          0
price         0
bedrooms      2
bathrooms     2
sqft_living    0
sqft_lot       0
floors        0
waterfront    4521
view          4103
sqft_above     0
sqft_basement 2718
yr_built       0
yr_renovated   0
```

شکل 27 - جایگزینی داده‌ها

و می‌بینیم که دیگر در مواردی که می‌خواستیم اصلاح کنیم صفری وجود ندارد و داده آماده استفاده است.

در آخر با استفاده از `price` ستون `to_numpy` را جدا برای `y` و باقی ستونها را برای `x` در نظر می‌گیریم که در بخش‌های بعدی استفاده خواهیم کرد.

برای همگرایی بهتر مدل و یادگیری آسان‌تر، قیمت‌ها که متغیر هدف هستند را نیز به فضای لگاریتمی می‌بریم که کاری مرسوم در مسائل به این شکل است. این کار باعث می‌شود مقادیر پرت تر مثل `outlier` ها اثرشان کمتر شود در عین حال که عمل برگشت پذیر است.

همچنین بخشی از پیش پردازش این است که داده‌ها نرمالایز شوند و ابعاد آن‌ها هم اندازه شود تا برحی اعداد خیلی بزرگ و برحی خیلی کوچک نباشند. اما برای این کار از یک لایه درون خود مدل استفاده خواهیم کرد که در بخش بعدی توضیح داده می‌شود. البته ستون تاریخ که به عدد تبدیل کردیم را به طور خاص با استفاده از کم کردن از میانگین و تقسیم بر `standard deviation` نرمال کردیم.

ب) به صورت تصادفی 80 درصد داده ها به عنوان داده آموزشی و 20 درصد را به عنوان داده تست در نظر بگیرید. تعدادی از داده ها را به عنوان داده ارزیابی در نظر بگیرید (مثلاً 10 تا 15 درصد داده های آموزشی).

شبکه عصبی چند لایه ای را طراحی کنید که قیمت خانه را پیش‌بینی کند. تعداد لایه تعداد نو، نون و تابع فعال ساز مناسب برای لایه، امشخص کنید آن را تحلیل کند (دو حالت برای تعداد لایه و دو حالت برای تابع فعال ساز را در نظر بگیرید).

برای بخش اول سوال، با استفاده از تابع `train_test_split` در کتابخانه `sklearn` این کار را انجام می‌دهیم. پس از تقسیم 80 و 20 آموزش و تست، سپس مقدار 15 درصد ارزیابی را در `fit` می‌دهیم تا ارزیابی روی آن مقدار از داده آموزشی انجام شود.

```
x_train, x_test, y_train, y_test = train_test_split(x, np.log(y), test_size=0.2, random_state=42)
class TrainerRegression:
    def __init__(self, units=1024, activation_function="relu",
                 optimizer="adam", n_layers=4, loss_function='mean_squared_error') -> None:
        self.optimizer = optimizer
        self.units = units
        self.activation_function = activation_function
        self.n_layers = n_layers
        self.loss_function = loss_function
        self.model = self.build_mlp_model()
        self.history = None
        self.training_time = None

    def print_summary(self):
        print("#####")
        print(f"Training Time: {self.training_time:.2f}s")
        print("batch size:", self.batch_size)
        print("n_layers:", self.n_layers)
        print("optimizer:", self.optimizer)
        print("activation_function:", self.activation_function)
        print("loss_function", self.loss_function)
        print(self.model.summary())

    def build_mlp_model(self):
        model = tf.keras.Sequential([
            tf.keras.layers.Input(shape=(4664, )),
            tf.keras.layers.Normalization(),
        ])
        model.add(tf.keras.layers.LayerNormalization())
        for n in range(self.n_layers):
            model.add(tf.keras.layers.Dense(self.units / 2**n))
            model.add(tf.keras.layers.LayerNormalization())
            model.add(tf.keras.layers.Activation(self.activation_function))
        model.add(tf.keras.layers.Dropout(0.1))
        model.add(tf.keras.layers.Dense(1))

        model.compile(
            optimizer=self.optimizer,
            loss=self.loss_function,
            metrics=['mean_squared_error', 'mean_absolute_error']
        )
        return model

    def train(self, x_train, y_train, batch_size=32, epochs=50):
        self.batch_size = batch_size
        early_stopping = tf.keras.callbacks.EarlyStopping(
            monitor='val_loss',
            verbose=1,
```

```

        patience=5,
        mode='min',
        restore_best_weights=True
    )
    start = time()
    self.history = self.model.fit(
        x_train, y_train,
        batch_size=batch_size,
        epochs=epochs,
        verbose=1,
        validation_split=0.15,
        callbacks=[early_stopping],
    )
    self.training_time = time() - start

    def plot_history(self):
        fig = plt.figure(figsize=(12, 4))
        metrics = ['loss', 'mean_squared_error', 'mean_absolute_error']
        for n, metric in enumerate(metrics):
            plt.subplot(1, 3, n+1)
            plt.plot(self.history.epoch, self.history.history[metric], label='Train')
            plt.plot(self.history.epoch, self.history.history[f"val_{metric}"], linestyle="--",
                     label='Validation')
            plt.xlabel('Epoch')
            plt.ylabel(metric)
            plt.title(metric)
            plt.legend()
        plt.show()

    def evaluate(self, x_test, y_test):
        [test_loss, test_mse, test_mae] = self.model.evaluate(x_test, y_test)
        print("Test Loss:", test_loss,
              "Test MSE:", test_mse,
              "Test MAE:", test_mae
        )
        preds = self.model.predict(x_test)
        plt.title('Targets vs. Predictions')
        plt.scatter(y_test, preds)
        plt.xlabel('Target')
        plt.ylabel('Prediction')
        plt.show()

```

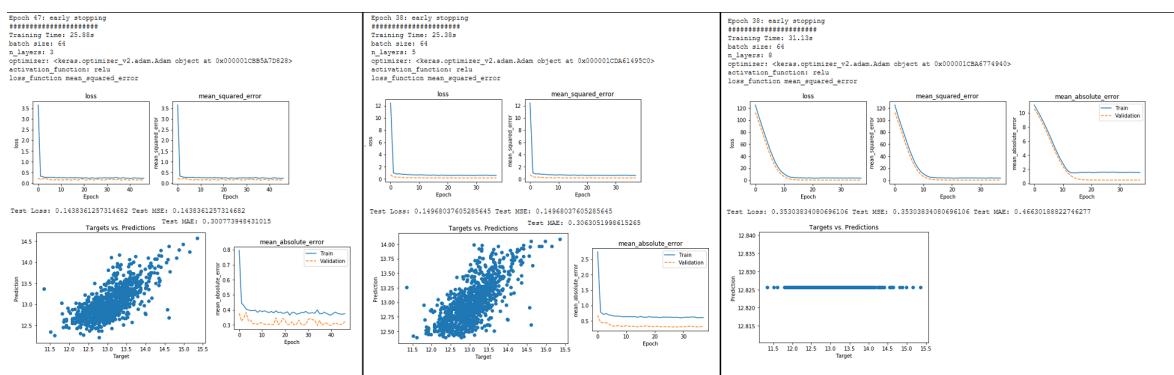
کد ۳ - آموزش و مدل Regression

مانند بخش قبلی، به شکلی کد نوشته شده است که می‌توان تعداد لایه و تابع فعالساز را مشخص کرد. اما در ساختار اصلی مدل لازم به ذکر است که ابتدا یک لایه `normalize` گذاشته ایم تا داده ها نرمال شوند. سپس لایه های `Dense` را داریم که مانند سوال قبلی تعداد `unit` های آن با هر لایه نصف می‌شود تا به ۱ که آخرین لایه است و باید عدد خروجی یا همان قیمت را مشخص کند می‌رسیم. همچنین برای آموزش بهتر، از `LayerNormalization` استفاده می‌کنیم که در مقاله <https://arxiv.org/abs/1607.06450> توضیح داده شده است. لازم به ذکر است که این لایه فقط نرمال سازی به تنها یعنی `mean` و `std` نهایی دارد که آموزش می‌بینند و لزوماً خروجی میانگین ۰ و `std` برای ۱ نخواهد داشت. نشان داده شده است که استفاده از این لایه آموزش مدل را بهتر می‌کند و در مدل‌های تحقیقات به روز نیز از آن استفاده می‌شود. چون ورودی حدود 4000 بعد دارد، نورون های لایه اول را نیز متناسب با آن

برابر 2048 قرار می‌دهیم. (این مورد بین سه حالت 1024، 2048 و 4096 تست شده و بهترین نتیجه را داشت. به ترتیب MSE این 3 حالت، 0.1514 و 0.1445 و 0.1502 با این حالت 12 میلیون پارامتر خواهیم داشت که برای این ابعاد و دیتاست مناسب است. همچنین چون دیتاست خیلی بزرگ نیست، بهتر است از اندازه دسته‌های کوچکتر استفاده کنیم (و همچنین چون مسئله Regression است و بعضی ممکن است خروجی میانگین شود به جای یادگیری واقعی). به همین دلیل از اندازه 64 استفاده می‌کنیم.

تعداد لایه و دو نوع تابع فعالساز که خواسته شده را در ادامه تست می‌کنیم و نتایج را می‌آوریم.

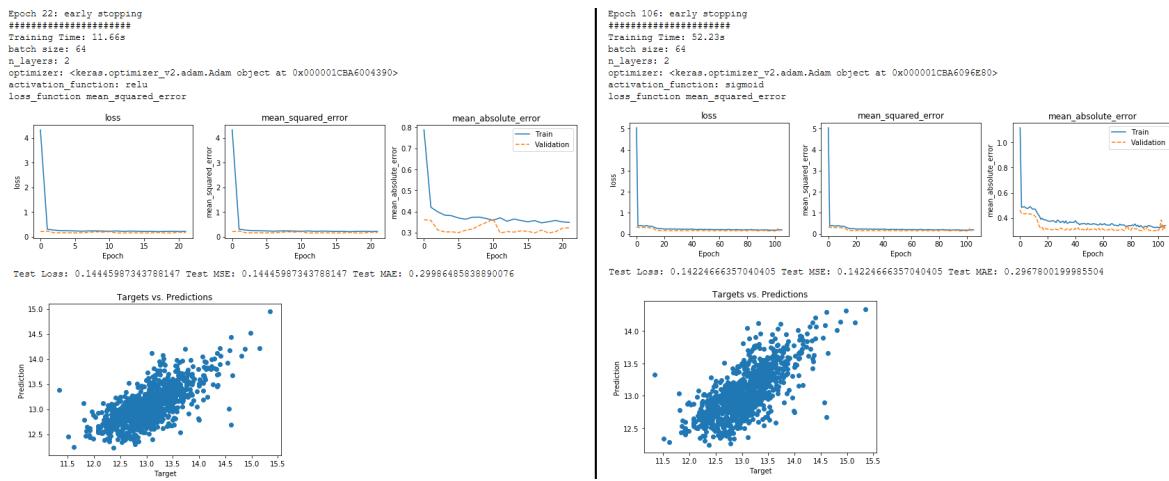
ابتدا برای تعداد لایه، بین 3 و 5 و 8 امتحان می‌کنیم.



شکل 28 - نتیجه تعداد لایه‌های مختلف

همانطور که دیده می‌شد، تعداد لایه 3 بهترین نتیجه را داشته است. این مورد نشان می‌دهد که تعداد لایه بیشتر در این مورد خاص باعث overfit شدن داده می‌شود و فقط داده آموزشی نتیجه بهتری می‌گیرد در حالیکه مثلا در تعداد لایه 8 دیگر اصلا برای داده تست مقدارهای مجازی نمی‌دهد و مشخصا overfit شده است تا داده آموزشی را به بهترین شکل تشخیص دهد. بنابراین در این مورد که دیتاست نسبتا کوچک است مدل کوچک نیز می‌تواند بهتری داشته باشد و همچنین تعداد لایه 2 نیز توانست نتیجه بهتر دهد و تعداد لایه 2 را انتخاب می‌کنیم.

سپس برای تابع فعالساز بین ReLU و sigmoid امتحان می‌کنیم که توضیحات تفاوت‌های این دو در سوال قبل آمده است.



شکل 29 - نتیجه تابع فعالساز مختلف

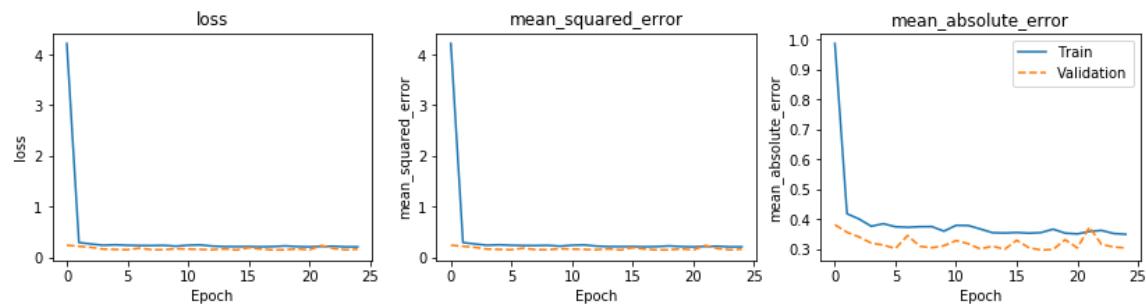
همانطور که قبلاً دیده شده بود، تابع sigmoid به علت vanishing gradient باعث می‌شود آموخت کندتر شود، در حالیکه ReLU این مشکل را ندارد و در اینجا هم می‌بینیم که 40 ثانیه سیگموید کندتر است تا همگرا شود. به همین دلیل در ادامه از ReLU استفاده می‌کنیم.

ج) با ثابت د، نظر گرفتن پارامترهای بدست آمده در قسمت (ب) با در نظر گرفتن MSE به عنوان تابع loss، مقادیر metric های MAE و MSE را در هر ایپاک برای داده آموزشی و داده تست (ارزیابی) در، یک نمودار، سم کنید. (یک نمودار، برای معیار MSE یک نمودار، برای MAE). برای داده تست نمودار، مقادیر پیش بینی شده بر حسب مقادیر واقعی رسم نمایید.

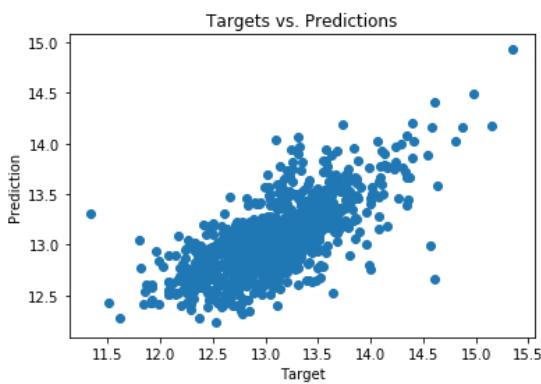
تعداد ایپاک بهینه در این قسمت را مشخص کنید.

نتیجه اجرا با تابع $loss$ در زیر آمده است.

```
Epoch 25: early stopping
#####
Training Time: 13.24s
batch size: 64
n_layers: 2
optimizer: <keras.optimizer_v2.adam.Adam object at 0x000001CDBF6D8E80>
activation_function: relu
loss_function mean_squared_error
```



Test Loss: 0.14392122626304626 Test MSE: 0.14392122626304626 Test MAE: 0.30151689052581787

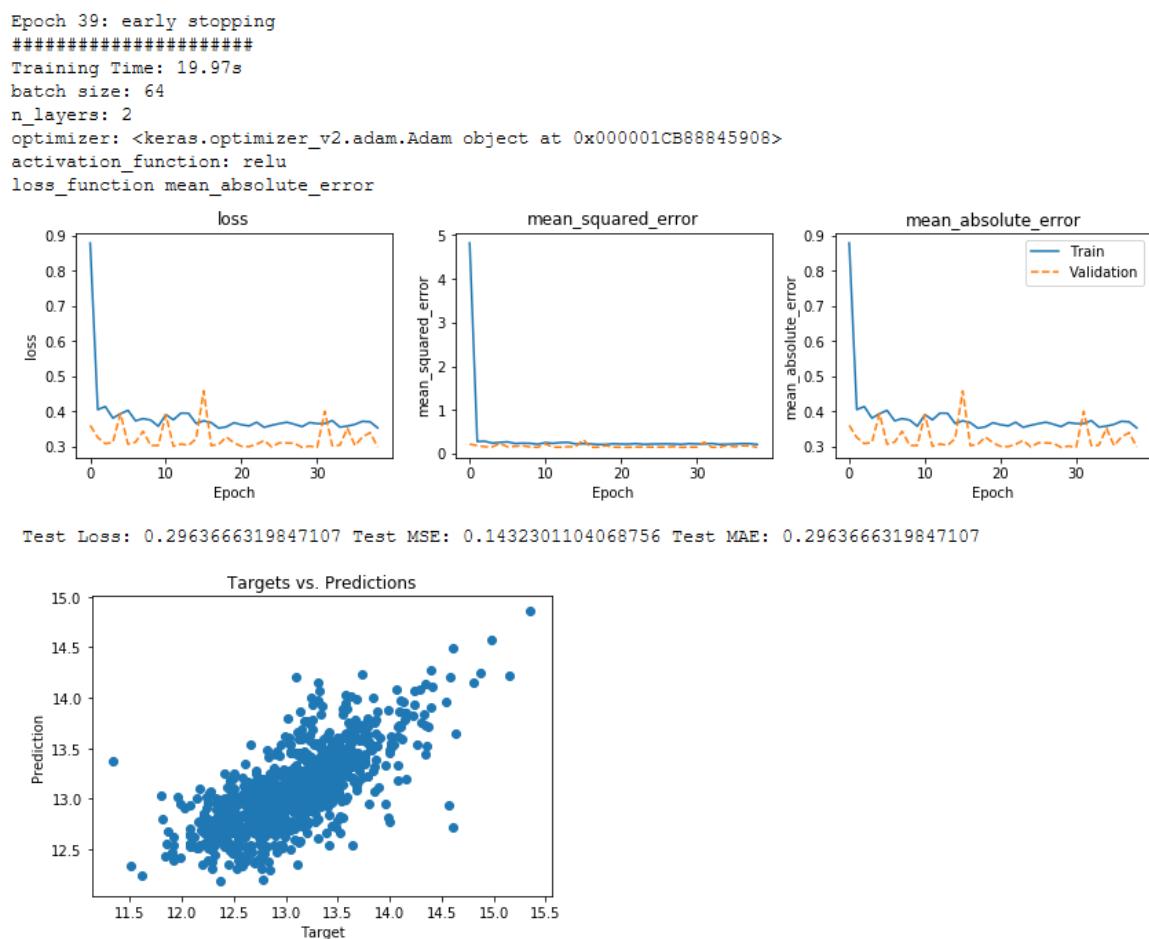


شکل 30 - نتیجه تابع loss

د) با ثابت در نظر گرفتن پارامترهای بدست آمده در قسمت (ب) با در نظر گرفتن **MAE** به عنوان تابع loss، مقادیر metric های **MAE** و **MSE** را در هر ایپاک برای داده آموزشی و داده تست (از زیابی) در یک نمودار، سم کنید (یک نمودار برای معیار **MSE** یک نمودار برای معیار **MAE**). برای داده تست، نمودار مقادیر پیش بینی شده بر حسب مقادیر واقعی رسم نمایید.

تعداد ایپاک بهینه در این قسمت را مشخص کنید.

نتیجه اجرا با تابع لاس MAE در زیر آمده است.



شکل 31 - نتیجه تابع لاس MAE

۵) ابتدا، روابط ریاضی **MSE** و **MAE** را بنویسید و سپس نتایج قسمت ج و د را باهم مقایسه کرده و توضیح دهید.
روابط ریاضی **MSE** و **MAE**:

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i^{real} - y_i^{pred})^2$$

$$MAE = \frac{1}{n} \sum_{i=1}^n |y_i^{real} - y_i^{pred}|$$

مهمترین تفاوت در ج و د این است که مقادیر **MSE** حدود 0.1432 و مقادیر **MAE** حدود 0.2963 است که نشان می‌دهد اعداد **MAE** بزرگتر است. علت آن این است که در **MSE** که به توان دو می‌رسانیم، وقتی اعداد کمتر از 1 هستند، کوچکتر هم می‌شوند، و اختلاف کمتر از فاصله **absolute** که در **MAE** داریم می‌شود. در این اجرای خاص، لاس **MAE** توانست نتایج بهتری بگیرد، اما اختلاف اندک است و نمی‌توان از نظر آماری این تفاوت را تفسیر پذیر دانست. اما می‌توان گفت که هر دو لاس توابع مناسبی برای یادگیری **Regression** هستند. از طرفی باید به تفاوت مشتق دو تابع هم اهمیت داد که در **MSE** مشتق پذیری داریم اما **MAE** در 0 مشتق پذیر نیست، و همچنین هر چه به 0 نزدیک شویم، **MSE** نیز مشتقش کوچک‌تر می‌شود، اما در **MAE** اینطور نیست که نشان می‌دهد اگر پیش‌بینی هر چه بهتر شود **MSE** تاثیرش را کمتر می‌کند، اما **MAE** نه. (این کار **MSE** را می‌توان مشابه **Focal Loss** هم دانست که نشان داده شده می‌تواند در آموزش مفید باشد. هر دو به شکلی تمرکز آموزش را بیشتر به سمت داده‌های سخت تر می‌برند و داده‌های راحت تر را کم ارزش‌تر می‌کنند.)

ضمنا باید اشاره کرد که **scatter plot** بین موارد واقعی و پیش‌بینی آن‌ها شکلی را نشان می‌دهد که **correlation** مثبت و نسبتاً نزدیک 1 را نشان می‌دهد که نمایش دهنده یادگیری مناسب مدل است.

* امتیازی:

مدل رگرسیون خطی از نظر تئوری بررسی کنید، این مدل چه مولفه هایی را چگونه تخمین می زند؟ در کتابخانه فوق الذکر مدل موسوم به ridge وجود دارد، این مدل را از نظر تابع هزینه همانند مدل رگرسیون خطی بررسی کنید، چه مزیتی نسبت به مدل رگرسیون خطی دارد؟ این مدل را برای این سوال پیاده سازی کنید، آیا بهبودی حاصل میشود؟

همانطور که در درس استنباط آماری داشتیم، رگرسیون خطی تلاش میکند عرض از مبدا و سپس ضریب هر کدام از ویژگی ها را به دست آورد که در انتها یک مدل خطی به ما می دهد.

- Recall that the least squares fitting procedure estimates $\beta_0, \beta_1, \dots, \beta_p$ using the values that minimize:

$$RSS = \sum_{i=1}^n \left(y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2$$

- In contrast the **ridge regression** coefficient estimates β_i that minimize:

$$\sum_{i=1}^n \left(y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^p \beta_j^2$$

where λ is a tuning parameter.

شکل 32 - تعریف رگرسیون خطی عادی و با رگولاژیشن Ridge

برای این کار همانطور که در شکل آمده سعی می شود خطای مشابه MSE کمینه شود. برای این کار مشتق آن نوشته می شود و سپس جایی که 0 می شود پارامترهای مورد نظر را به ما می دهد. اما تفاوتی که Ridge دارد این است که یک ترم Regularization از نوع L2 نیز اضافه می کند تا این پارامترها بیش از حد بزرگ نشوند و دوست داریم کوچک بمانند. این کار باعث جلوگیری از overfitting (در این مورد با جلوگیری از افزایش بیش از حد وزن یک ویژگی) و بهبود generalization می شود. همچنین Lasso نیز وجود دارد که به جای L2 از L1 استفاده می کند که قدر مطلق پارامترهاست و به همین دلیل مشتق پذیر نیست که باید با روش های دیگر در نقطه ای که مشتق ندارد عمل کرد.

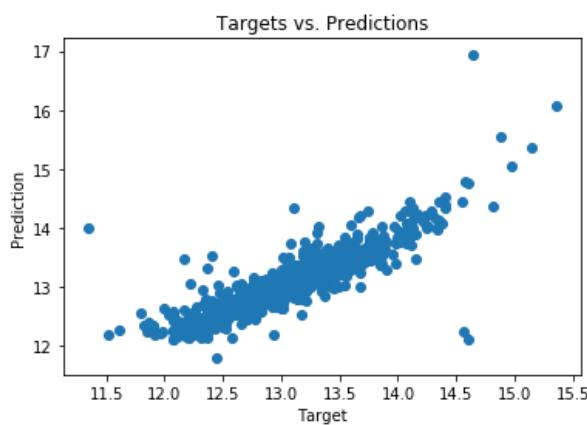
پیاده‌سازی خواسته شده و نتیجه آن در زیر آمده است.

```
: from sklearn.linear_model import Ridge

def evaluate(x_test, y_test, y_pred):
    print("Test MSE:", np.sum((y_pred - y_test) ** 2) / len(y_test),
          "Test MAE:", np.sum(np.abs(y_pred - y_test)) / len(y_test))
    )
    plt.title('Targets vs. Predictions')
    plt.scatter(y_test, y_pred)
    plt.xlabel('Target')
    plt.ylabel('Prediction')
    plt.show()

clf = Ridge(alpha=1.0)
clf.fit(x_train, y_train)
evaluate(x_test, y_test, clf.predict(x_test))
```

Test MSE: 0.07270271565619392 Test MAE: 0.16270201395509556



شکل ۳۳ - نتیجه Ridge

همانطور که دیده می‌شود این حل دقیق توانست نتیجه را بسیار بهبود دهد و مدل خیلی بهتری به ما ارائه کند.

سوال 3 Dimensionality Reduction – 3

الف) ایده روش PCA، بیان کنید.

با استفاده از روابط ریاضی توضیح دهید چگونه میتوان با استفاده از این روش ابعاد داده را کاهش داد. سپس منحنی تعداد components را بر حسب واریانس تجمعی برای دیتاست مجموعه داده CIFAR-10 رسم کنید و بیان کنید که کاهش بعد تا چه میزان قابل قبول است؟ در ادامه روش PCA در برای مجموعه داده CIFAR-10 پیاده سازی کنید شبکه را آموزش دهید.

نکته 2: در این قسمت نمیتوانید کتابخانه `scipy` برای پیاده سازی استفاده کنید باید خودتان پیاده سازی نمایید

در روش PCA به دنبال جهت هایی در فضای فیچرها می‌گردیم که داده‌ها بیشترین واریانس را در آن جهت داشته باشند، یا به عبارتی کمترین هزینه تصویر شدن روی آن بعد و بازگشت از آن را بدھیم. با حل مسئله بهینه سازی این موضوع، به این می‌رسیم که این جهت‌ها همان eigenvector های ماتریس کوواریانس فیچرها هستند.

Assume x denotes an n-dimensional point of the input space: $x^T = [1, x_1, x_2, \dots, x_n]$ (bias has been augmented)

There will be a linear correlation among different input dimensions, if for all samples of x there is a non-zero real vector (the normal vector), $a^T = [a_0, a_1, \dots, a_n]$, in order that:

$$a^T x = 0, \quad x \in \{x^i\}_{i=1}^m$$

One can show for r independent linear correlations, “ r ” Eigen values of the “Auto Correlation Matrix” (R) are zero and their corresponding Eigen vectors reveal “ r ” independent linear correlations.

$$\begin{aligned} R &= \sum_{i=1}^m x^i x^{i^T} = V \Lambda V^T \\ \Lambda &= \begin{bmatrix} \Lambda^1 & 0 \\ 0 & \Lambda^2 \end{bmatrix} \quad \Lambda^1 = \text{diag}([\lambda_1, \dots, \lambda_r]) = \mathbf{0} \quad \Lambda^2 = \text{diag}([\lambda_{r+1}, \dots, \lambda_n]) \\ V &= \begin{bmatrix} v_1 \dots v_r & v_{r+1} \dots v_n \\ \hline v_1 & v_2 \end{bmatrix} = [V_1 \ V_2] \end{aligned}$$

r independent Correlations: $v_j^T x = 0 (j \in \{1, \dots, r\})$

شکل 34 - روش PCA

طبق شکل، فرض می‌کنیم بردار a مخالف 0 وجود داشته باشد که بین آنها وابستگی باشد. و می‌گوییم با عمل SVD تبدیل می‌شود به ماتریس بردارهای ویژه در ماتریس قطعی مقادیر ویژه در transpose ماتریس بردارهای ویژه. در این صورت می‌توانیم نشان دهیم که داده‌هایی که ماتریس کوواریانس آن را در نظر گرفتیم، به اندازه تعداد مقادیر ویژه که 0 است کوئیلیشن‌های مستقل در ویژگی‌هایش وجود دارد. به این ترتیب ساخت ماتریس اتوکوئیلیشن ویژگی‌ها و تجزیه آن به مقادیر و بردارهای ویژه و مرتب سازی آن براساس مقادیر ویژه می‌توان وابستگی‌ها را تعریف کرد.

Dimensionality Reduction and Data Retrieving for Linear Correlations

$$V^T x = \begin{bmatrix} V_1^T \\ V_2^T \end{bmatrix} x = \begin{bmatrix} V_1^T x \\ V_2^T x \end{bmatrix} = \begin{bmatrix} 0_{r \times 1} \\ z_{(n-r) \times 1} \end{bmatrix}$$

Nullity part which will be removed.

Informative part which forms the new space

$$x \in R^n \xrightarrow{\text{Dimensionality Reduction}} z(x) = Tr(x) = V_2^T x \quad z \in R^{n-r}$$

$$z \in R^{n-r} \xrightarrow{\text{Data retrieving}} x = Tr^{-1}(z) = V \begin{bmatrix} 0_{r \times 1} \\ z \end{bmatrix} = VV^T x = x \in R^{n-r}$$

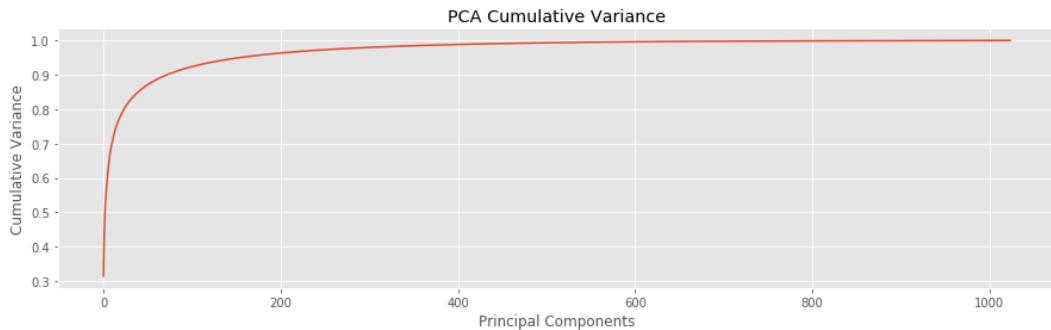
شکل 35 - ادامه روش PCA

برای رفتن از فضای n بعدی به $n-r$ بعدی کافیست که ترانسفورمیشن را روی x اعمال کیم. همچنین حتی می‌توانیم معکوس این کار را انجام دهیم و معکوس ترانسفورمیشن را روی z اعمال کنیم و به x می‌رسیم. اگر رابطه خطی بین ویژگی‌ها بوده باشد می‌توانیم دقیقاً به همان مقدار اولیه بازگردیم، اما اگر اینطور نباشد تقریبی از بازگشت را خواهیم داشت.

```
In [110]: x_train_processed = (x_train_processed - x_train_processed.mean()) / x_train_processed.std()
x_test_processed = (x_test_processed - x_test_processed.mean()) / x_test_processed.std()
cov_mat = np.dot(x_train_processed.T, x_train_processed)
eigenvalues, eigenvectors = np.linalg.eig(cov_mat)

# Each feature explained variance
explained_variance = [ev / np.sum(eigenvalues) for ev in eigenvalues]
# Cumulative Variance
cumulative_variance = [np.sum(explained_variance[:i+1]) for i in range(len(explained_variance))]

plt.style.use('ggplot')
plt.figure(figsize=(15, 4))
plt.plot(cumulative_variance)
plt.title('PCA Cumulative Variance')
plt.xlabel('Principal Components')
plt.ylabel('Cumulative Variance')
plt.show()
```



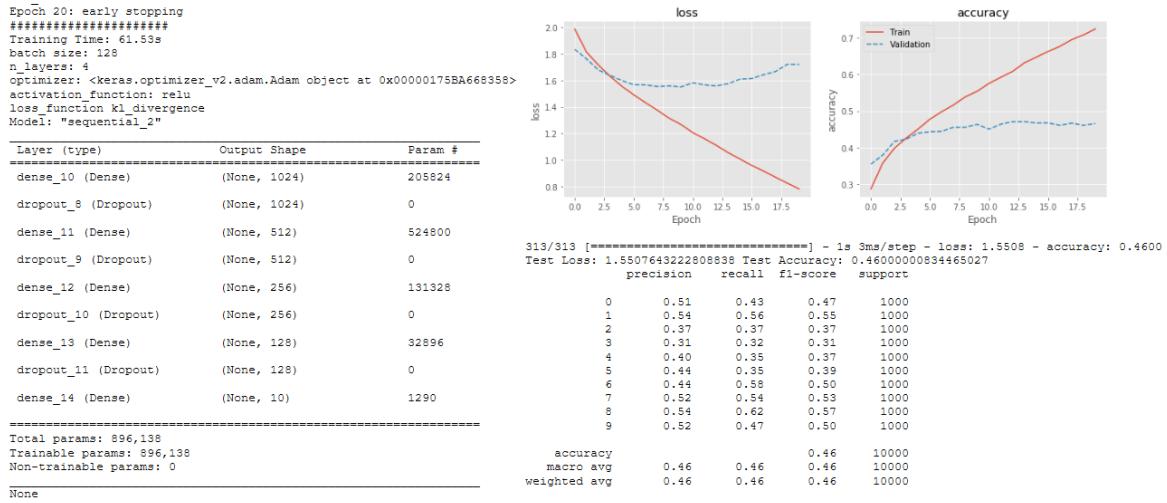
```
In [114]: x_train_processed_pca = np.dot(x_train_processed, eigenvectors[:,200].T)
x_test_processed_pca = np.dot(x_test_processed, eigenvectors[:,200].T)
x_train_processed_pca.shape
```

Out[114]: (50000, 200)

شکل 36 - پیاده سازی روش PCA

برای پیاده سازی ابتدا داده را نرمال می کنیم تا میانگین 0 شود. سپس می توانیم ماتریس کوواریانس را صرفا با یک ضرب ساده به دست بیاوریم. (نیازی به میانگین در این محاسبات نیست چون میانگین را با نرمالیزیشن 0 کردیم). سپس eigenvector های این ماتریس را به دست می آوریم که همان بردارهای مناسب PCA هستند. برای این که نمایش دهیم چه مقدار از واریانس توضیح داده می شود، ابتدا مقداری که هر بردار از واریانس را توضیح می دهد با تقسیم واریانس واریانس تجمعی را به دست می آوریم. سپس با تجمعی آنها، نمودار eigenvalue آن بر مجموع کل eigenvector ها به دست می آوریم. همانطور که دیده می شود در 100 کامپوننت بیش از 90 درصد واریانس توضیح داده می شود که مقدار مناسبی است. برای اطمینان بیشتر ما از 200 کامپوننت استفاده می کنیم که بیش از 95 درصد واریانس را توضیح می دهد و برای تصویر داده در این جهات کافیس یک ضرب انجام دهیم که دیده می شود در انتهای داده از شکل 1024 بعدی به 200 بعد تبدیل شده است با توجه به اینکه بیش از 95 درصد واریانس نیز حفظ شده است.

حال از این مقدار استفاده می کنیم تا مدل را آموزش دهیم.



شکل 37 - نتیجه اجرا با ابعاد کمتر به دست آمده از PCA

نتیجه در شکل بالا آمده است. توضیحات بیشتر و تحلیل را در بخش ج خواهیم داشت.

ب) روش Autoencoder پیاده سازی کنید شبکه آموزش دهید.

در این بخش برای اینکه بتوان مقایسه عادلانه‌ای با PCA داشته باشیم، اندازه بردارهای encode شده را برابر 200 در نظر می‌گیریم. به این ترتیب از 1024 به 200 و دوباره به 1024 برمی‌گردیم. کل autoencoder می‌شود کل بخش گفته شده که ورودی و خروجی آن برابر هم است و در آخر که آموزش تمام شد، کافیست تا مرحله تبدیل به 200 بعد را اجرا کنیم تا کاهش ابعاد را داشته باشیم.

```

class TrainAutoencoder:
    def __init__(self) -> None:
        self.model = self.build_autoencoder_model()
        self.history = None
        self.training_time = None

    def build_autoencoder_model(self):
        self.input_ = tf.keras.Input(shape=(1024,))
        self.encoded = tf.keras.layers.Dense(200, activation='relu')(self.input_)
        self.decoded = tf.keras.layers.Dense(1024)(self.encoded)
        autoencoder = tf.keras.Model(self.input_, self.decoded)
        autoencoder.compile(
            optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3),
            loss='mean_squared_error'
        )
        return autoencoder

    def encode(self, x_test):
        encoder = tf.keras.Model(self.input_, self.encoded)
        encoded_inputs = encoder.predict(x_test)
        return encoded_inputs

    def train(self, x_train, batch_size=128, epochs=50):
        print(self.model.summary())

```

```

    self.batch_size = batch_size
    assert x_train.shape[1:] == (1024, )
    early_stopping = tf.keras.callbacks.EarlyStopping(
        monitor='val_loss',
        verbose=1,
        patience=5,
        mode='min',
        restore_best_weights=True
    )
    start = time()
    self.history = self.model.fit(
        x_train, x_train,
        batch_size=batch_size,
        epochs=epochs,
        verbose=1,
        validation_split=0.2,
        callbacks=[early_stopping],
    )
    self.training_time = time() - start

    def plot_history(self):
        fig = plt.figure(figsize=(12, 4))
        metrics = ['loss']
        for n, metric in enumerate(metrics):
            plt.subplot(1, 1, n+1)
            plt.plot(self.history.epoch, self.history.history[metric], label='Train')
            plt.plot(self.history.epoch, self.history.history[f"val_{metric}"], linestyle="--",
                     label='Validation')
            plt.xlabel('Epoch')
            plt.ylabel(metric)
            plt.title(metric)
            plt.legend()
        plt.show()

    ta = TrainAutoencoder()
    ta.train(x_train_processed, batch_size=128, epochs=30)
    ta.plot_history()

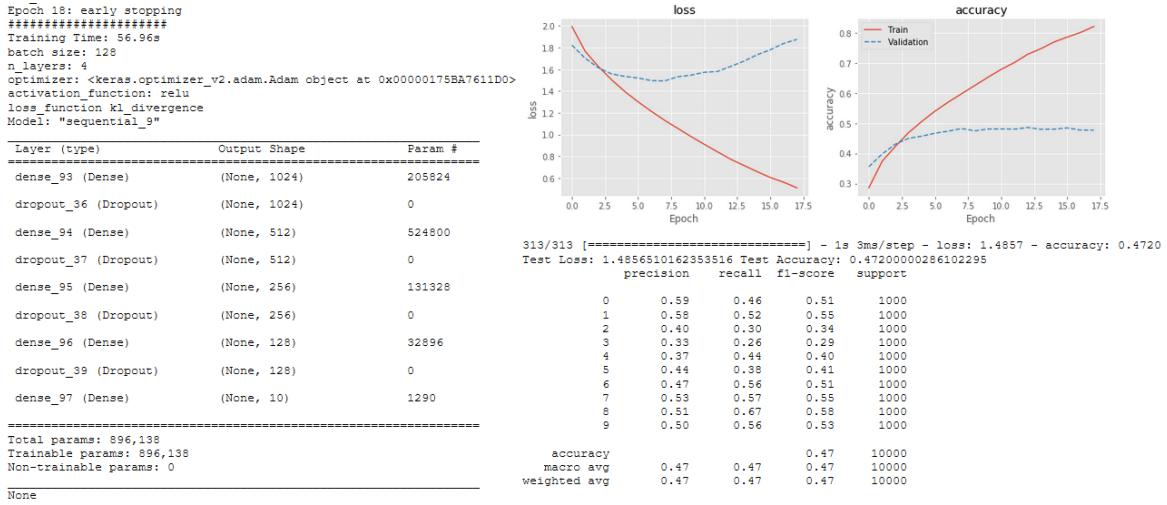
    x_train_processed_autoencoder = ta.encode(x_train_processed)
    x_test_processed_autoencoder = ta.encode(x_test_processed)
    x_train_processed_autoencoder.shape
    → (50000, 200)

```

کد ۴ - آموزش و مدل AutoEncoder

همانطور که دیده می شود، در انتها ابعاد از 1024 به 200 رسیده است.

در زیر نتایج آموزش با این ابعاد دیده می شود.



شکل 38 - نتیجه اجرا با ابعاد کمتر به دست آمده از AutoEncoder

تحلیل این نتایج در بخش ج خواهد آمد.

ج) نتایج مربوط به بهترین مدل به دست آمده از سوال یک، را با نتایج به دست آمده در قسمت الف و ب همین سوال، در یک جدول مقایسه کنید و شمود خود را توضیح دهید.

جدول 3 - مقایسه نتایج

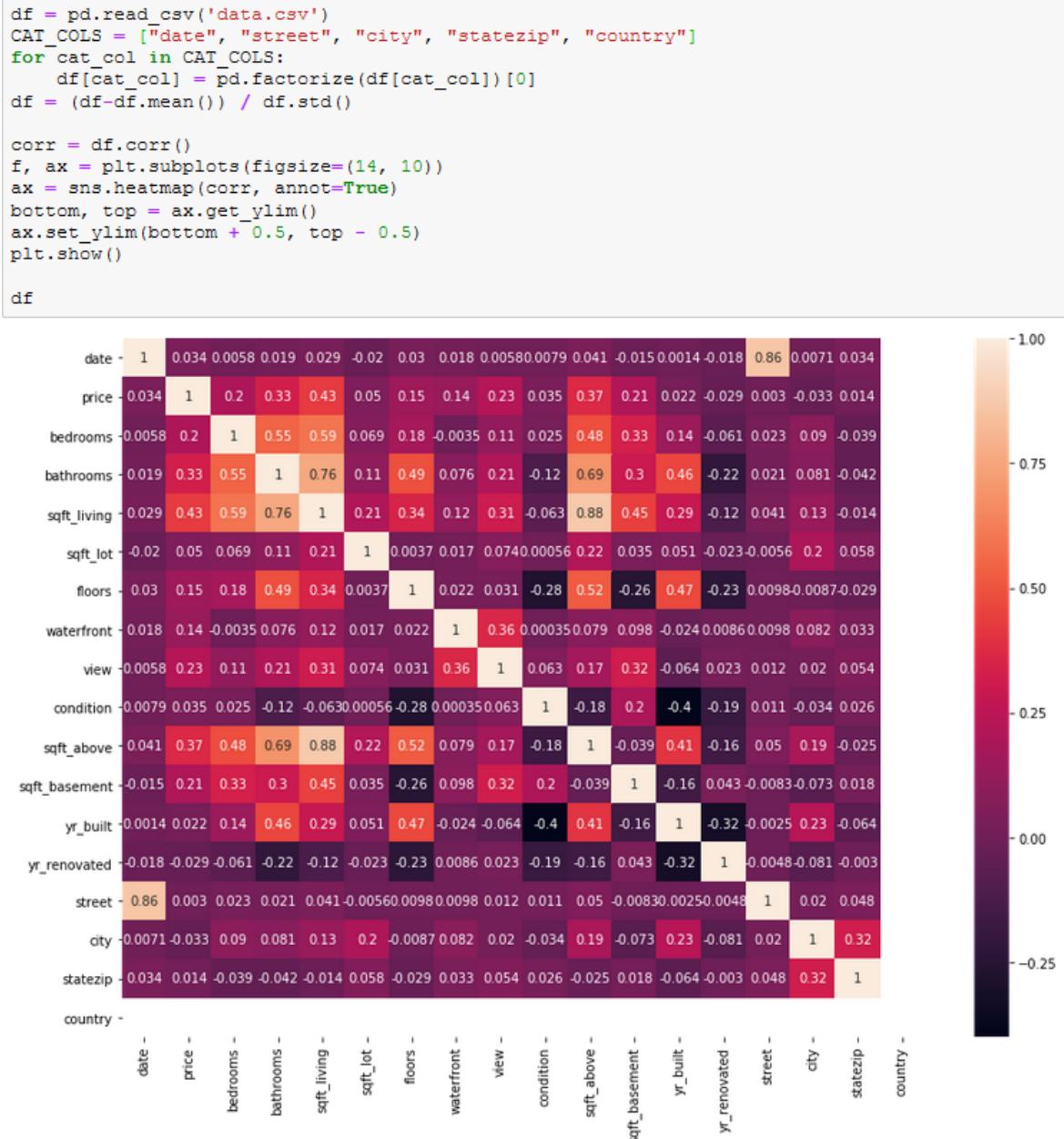
خطای تست	دقت تست	زمان کل آموزش	تعداد پارامتر	ابعاد ورودی	
1.5306	46.54%	69.63s	1,739,914	1024	بهترین مدل سوال 1
1.5507	46.00%	61.53s	896,138	200	PCA
1.4856	47.20%	56.96s	896,138	200	AutoEncoder

ابتدا باید اشاره کرد که با کاهش ابعاد، دیگر ورودی ها 1024 نیستند و در دو مدل این سوال، ورودی به اندازه 200 بوده است. همین باعث شده است که تعداد پارامترها نیز بسیار کاهش یابد و همچنین زمان آموزش نیز نسبت به آموزش روی کل ابعاد کمتر شود که نکته مثبت کاهش ابعاد است. (البته زمان آموزش مدل AutoEncoder اینجا در نظر گرفته نشده است و فقط زمان آموزش نهایی را مقایسه می‌کنیم. همچنین می‌بینیم که با کاهش ابعاد در AutoEncoder توانستیم بهترین نتیجه را بگیریم. علت این مورد را می‌توان این گفت که با کاهش ابعاد قدرت تعمیم و generalization مدل افزایش می‌یابد چون با نگه داشتن ابعاد مهمتر و کنار گذاشتن ابعاد بی اهمیت از overfitting مدل روی فیچرهای کم اهمیت جلوگیری

می کنیم و این مورد تعمیم پذیری مدل را بهبود می دهد. البته این موضوع در PCA پیش نیامده که می توان آن را به حالت خاص این مسئله ربط داد و همچنین باید توجه کرد که روش PCA با آموزشی که می بیند می تواند به صورت پیچیده تر و بهتری نسبت به AutoEncoder ابعاد را کاهش دهد که این مورد در نتایج دیده می شود. در کل یک کاهش ابعاد مناسب هم می تواند زمان و پارامترهای مسئله را کاهش دهد که عملکرد بهتری را حاصل می شود و همچنین با حذف ابعاد کم اهمیت تر باعث بهبود تعمیم پذیری مدل می شود.

د) ماتریس همبستگی مربوط به دیتاست سوال 2، ا، سم کنید (مشابه شکل زیر) و توضیح مختصری در مورد آن بدھید.

ماتریس همبستگی در زیر آمده است.



شکل 38 - ماتریس همبستگی

ماتریس همبستگی، کوریلیشن هر دو فیچر موجود را در یک ماتریس نمایش می‌دهد که چون کوریلیشن a و b معادل کوریلیشن b و a است، این ماتریس نیز متقارن خواهد بود. همچنین بیاد توجه داشت که بازه کوریلیشن بین -1- یعنی اثرگذاری معکوس، تا 1 یعنی اثرگذاری مثبت است و هر چه نزدیک 0 باشد یعنی رابطه خاصی بین دو فیچر وجود ندارد. مثلاً در این مثال خاص،

bathrooms sqft living با همبستگی 0.76 دارد که عدد نسبتا بزرگی محسوب می‌شود و نشان می‌دهد که با افزایش مساحت، تعداد bathrooms نیز افزایش می‌یابد و همبستگی مثبت دارند. همچنین باید اشاره کرد که روی قطر ماتریس کوریلیشن هر فیچر با خودش است که طبیعتا بیشترین مقدار یعنی 1 می‌شود.

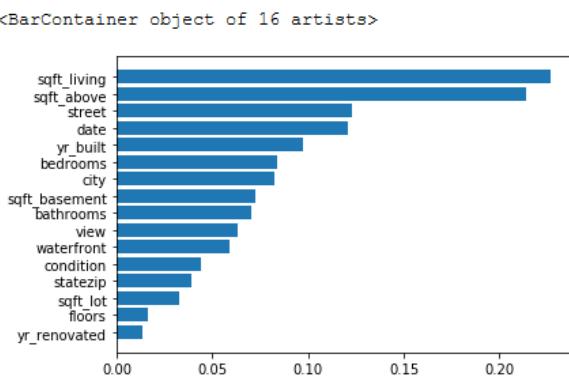
(و) با استفاده از مدل‌های Linear Regression Decision Tree، اهمیت هر ویژگی را در سوال 2 بدست آورید و در یک بار پلاس نمایش دهید. برای این کار می‌توانید از کتابخانه scikit-learn استفاده کنید.

در LinearRegression یک خط به داده ما فیت می‌شود که به ازای هر فیچر یک وزن یا coefficient نسبت داده می‌شود. هر چه این مقدار برای یک فیچر بزرگتر باشد، یعنی حضور آن فیچر تاثیر بزرگتری بر هدف نهایی ما دارد. البته باید توجه داشت که این ضریب متناسب با اندازه و scale مقادیر فیچرها هم هست و مثلا انتظار می‌رود یک فیچر بزرگ، ضریب کوچکتری داشته باشد تا خنثی شود. برای اینکه این موضوع را حل کنیم، قبل از فیت کردن مدل، فیچرها را نرمال می‌کنیم تا همه در یک اندازه باشند و همچنین در این مورد اندازه بیشتر از جهت برای مهم است به همین دلیل abs نیز می‌گیریم.

```
from sklearn.datasets import load_diabetes
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import LinearRegression

features = df.drop(["price", "country"], axis=1)
regressor = LinearRegression()
regressor.fit(features, df["price"])
intercept = regressor.intercept_

reg_importance = pd.DataFrame(zip(features.columns, np.abs(regressor.coef_)), columns=["feature", "coef"])
reg_importance = reg_importance.sort_values(by='coef', ascending=True)
plt.barh(reg_importance["feature"], reg_importance["coef"])
```



شکل 39 - اهمیت ویژگی با رگرسیون خطی

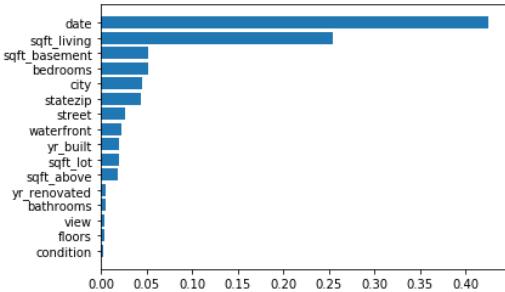
حالا به سراغ Decision Tree Regression می‌رویم که به صورت درختی تصمیم گیری می‌کند و طبیعتاً هر فیچری که بتواند جداسازی بهتری ارائه دهد و مهم‌تر باشد در این درخت به ریشه نزدیک‌تر و مهم‌تر است و از همین جهت می‌توان اهمیت فیچرهای را متوجه شد.

در زیر نتیجه این کار آمده است.

```
features = df.drop(["price", "country"], axis=1)
regressor = DecisionTreeRegressor()
regressor.fit(features, df["price"])

reg_importance = pd.DataFrame(zip(features.columns, regressor.feature_importances_), columns=["feature", "importance"])
reg_importance = reg_importance.sort_values(by='importance', ascending=True)
plt.barh(reg_importance["feature"], reg_importance["importance"])
```

<BarContainer object of 16 artists>



شکل 40 - اهمیت ویژگی با درخت تصمیم

بازخورد

با سلام و وقت بخیر

این تمرین زمان خیلی زیادی از من گرفت و حدود 50 صفحه گزارش برای آن نیز نشان دهنده این موضوع است. اگر پروژه‌ها کمی سبک‌تر طراحی شوند تا بتوانیم با بررسی دقیق‌تر سوالات و بدون استرس تمام کردن تمرین در زمان مشخص شده، یادگیری بهتری داشته باشیم قطعاً مفید‌تر خواهد بود.

با تشکر از وقتی که گذاشتید
