

تمرین سری دو

درس شبکه‌های عصبی

فرهاد دلیرانی

۹۶۱۳۱۱۲۵

dalirani@aut.ac.ir

dalirani.1373@gmail.com

فهرست

۱	ابزارهای استفاده شده و دیتاست
۳	کدهای Multi-layer perceptron
۳	کد خواندن دیتاست (read_dataset.py)
۴	کلاس MLP
۱۲	کد رسم نمودارهای مختلف در زمان آموزش شبکه
۱۳	تابع k-fold
۱۴	کد پیدا کردن پارامترهای مناسب برای MLP (best_MLP.py)
۱۶	کد و رابطه‌ی کاربری برای استفاده از کلاس MLP و رسم نمودارها (run_mlp.py)
۱۷	بررسی تاثیر تعداد نرون‌ها در یک شبکه‌ی سه لایه بر دقت دسته‌بند
۱۷	عنوان:
۱۷	شرایط آزمایش:
۲۳	نتیجه انجام آزمایش:
۵۷	نتیجه گیری:
۶۰	بررسی تاثیر تعداد لایه‌ی های نهان بر روی عملکرد شبکه
۶۰	عنوان:
۶۰	شرایط آزمایش:
۶۴	نتیجه‌ی انجام آزمایش:
۹۳	نتیجه گیری
۹۵	برای مسیله‌ی رسم الخط هندی، شبکه‌ی عمیق بهتر است یا عریض
۹۵	عنوان
۹۵	پاسخ
۹۸	بررسی دقت دسته‌بندی و سرعت همگرایی در سه روش batch, steepest descent, The Levenberg-Marguardt
۹۸	عنوان
۹۹	شرایط آزمایش
۱۰۳	نتایج آزمایش
۱۲۴	نتیجه گیری

۱۲۹.....	آموزش شبکه با تابع هزینه‌ی Cross Entropy و MSE
۱۲۹.....	عنوان
۱۲۹.....	شرایط آزمایش
۱۳۱.....	نتیجه انجام آزمایش
۱۴۲.....	نتیجه‌گیری
۱۴۸.....	آموزش شبکه‌ی پرسپترونی چند لایه بر روی دیتاست رقم‌ها و حرف‌ها
۱۴۸.....	عنوان
۱۴۸.....	شرایط آزمایش
۱۵۰.....	نتیجه‌های آزمایش
۱۶۱.....	نتیجه‌گیری

ابزارهای استفاده شده و دیتاست

زبان برنامه نویسی: پایتون ۳.۶.۴

محیط توسعه: pycharm 2017

سیستم عامل: ویندوز ۱۰

از آنجایی که حجم فایل برای ارسال بزرگ است به ناچار دیتاستها را پاک کردم. برای اجرای کدها، دیتاستها باید در محل مناسب، بر اساس شکل‌های زیر قرار بگیرند:

Name	Date modified	Type	Size
.idea	3/29/2018 11:58 PM	File folder	
__pycache__	3/29/2018 10:43 PM	File folder	
dataset	3/26/2018 8:46 PM	File folder	
activation_functions.py	3/21/2018 10:44 AM	Python File	2 KB
input-e02 - Copy.json	3/28/2018 11:03 PM	JSON File	1 KB
input-e02 - cut.json	3/29/2018 11:02 PM	JSON File	1 KB
input-e02.json	3/29/2018 10:41 PM	JSON File	1 KB
input-e07.json	3/28/2018 5:16 AM	JSON File	1 KB
k_fold.py	3/28/2018 4:45 AM	Python File	3 KB
mlp.py	3/29/2018 11:58 PM	Python File	30 KB
MLP-output.json	3/29/2018 11:24 PM	JSON File	1,806 KB
plotting.py	3/28/2018 5:19 AM	Python File	4 KB
read_dataset.py	3/26/2018 10:00 PM	Python File	13 KB
run_mlp.py	3/29/2018 10:43 PM	Python File	8 KB
test.py	3/27/2018 1:04 PM	Python File	1 KB

Name	Date modified	Type	Size
character	3/26/2018 8:46 PM	File folder	
digit	3/11/2018 4:37 PM	File folder	

part1 > dataset > digit >

Name	Date modified	Type
digit_0	3/11/2018 4:36 PM	File folder
digit_1	3/11/2018 4:36 PM	File folder
digit_2	3/11/2018 4:36 PM	File folder
digit_3	3/11/2018 4:36 PM	File folder
digit_4	3/11/2018 4:36 PM	File folder
digit_5	3/11/2018 4:36 PM	File folder
digit_6	3/11/2018 4:36 PM	File folder
digit_7	3/11/2018 4:36 PM	File folder
digit_8	3/11/2018 4:36 PM	File folder
digit_9	3/11/2018 4:36 PM	File folder

part1 > dataset > character >

Name	Date modified	Type
character_1_ka	3/11/2018 4:34 PM	File folder
character_2_kha	3/11/2018 4:34 PM	File folder
character_3_ga	3/11/2018 4:35 PM	File folder
character_4_gha	3/11/2018 4:35 PM	File folder
character_5_kna	3/11/2018 4:35 PM	File folder
character_6_cha	3/11/2018 4:35 PM	File folder
character_7_chha	3/11/2018 4:35 PM	File folder
character_8_ja	3/11/2018 4:35 PM	File folder
character_9_jha	3/11/2018 4:36 PM	File folder
character_10_yna	3/11/2018 4:33 PM	File folder
character_11_taamatar	3/11/2018 4:33 PM	File folder
character_12_thaa	3/11/2018 4:33 PM	File folder
character_13_daa	3/11/2018 4:33 PM	File folder
character_14_dhaa	3/11/2018 4:33 PM	File folder
character_15_adna	3/11/2018 4:33 PM	File folder
character_16_tabala	3/11/2018 4:33 PM	File folder
character_17_tha	3/11/2018 4:33 PM	File folder
character_18_da	3/11/2018 4:34 PM	File folder

کدهای Multi-layer perceptron

ک خواندن دیتاست (read_dataset.py)

در پوشه‌ی part1 فایل read_dataset.py قرار دارد که در آن دوتابع زیر وجود دارد

```
def read_digit(train_ratio=0.80, valid_ratio=0.10, test_ratio=0.10, random_state=None):
    """
    Read Digit dataset.
    :param train_ratio: portion of dataset that should be assigned to training
    :param valid_ratio: portion of dataset that should be assigned to validation
    :param test_ratio: portion of dataset that should be assigned to testing
    :param random_state: for controlling generating random number and making function reproducible
    :return: trainX, validX, testX, trainY, validY, testY

        X format : ndarray [[observation1],[observation2],...,[observationN]]
        Y format : ndarray [[label observation1],[label observation2],...,[label observationN]]
    """

```

```
def read_digit_letter(train_ratio=0.80, valid_ratio=0.10, test_ratio=0.10, random_state=None):
    """
    Read digits and letter dataset.
    :param train_ratio: portion of dataset that should be assigned to training
    :param valid_ratio: portion of dataset that should be assigned to validation
    :param test_ratio: portion of dataset that should be assigned to testing
    :param random_state: for controlling generating random number and making function reproducible
    :return: trainX, validX, testX, trainY, validY, testY

        X format : ndarray [[observation1],[observation2],...,[observationN]]
        Y format : ndarray [[label observation1],[label observation2],...,[label observationN]]
    """

```

که با گرفتن درصد مجموعه‌ی آموزش، ارزیابی و تست و یک عدد صحیح مثبت دیتاست تصویرهای عددها و یا عددها به علاوه‌ی حروف را می‌خوانند. آرگمان random_state برای تولید عددهای تصادفی و شافل کردن دیتاست است و برای تکرار پذیر کردن آزمایش است.

هر نمونه در دیتاست یک تصویر 32×32 است که به صورت یک وکتور 1024×1 در می‌آید و از آنجایی که اعداد هر پیکسل آن بین ۰ تا ۲۵۵ است هر تصویر را برابر ۲۵۵ تقسیم کرده‌ام و با این کار نمونه‌ها را نرمال سازی کرده‌ام.

برچسب هر نمونه را با استفاده از روش One Hot مشخص کرده‌ام. برچسب هر نمونه برای زمانی که دیناست عددها را می‌خوانیم یک وکتور 46×1 است و در زمانی که دیتاست عددها و حروف را می‌خوانیم یک وکتور 10×1 است.

کلاس MLP

در این بخش کدی که برای شبکه‌ی چند لایه‌ی پرسپترون را نوشتہام را توضیح می‌دهم. کد شبکه‌ی چند لایه‌ای که نوشتہام در پوشه‌ی 1 part و در فایل mlp.py قرار دارد.

این کد به **صورت جنرال** نوشته شده است:

- تعداد feature های ورودی ها قابل تنظیم است
- تعداد لایه‌های شبکه قابل تنظیم است
- تعداد نرون‌ها در هر لایه قابل تنظیم است
- هر لایه قابل تنظیم است. Activation function
- می‌توان تابع هزینه‌ی MSE و یا Cross-Entropy را انتخاب کرد.
- می‌توان برای آموزش حالت‌های steepest descent و stochastic, momentum, batch را انتخاب کرد.
- برای خاتمه‌ی آموزش می‌توان حداکثر تعداد epoch و یا کمتر شدنند مقدار تابع هزینه از حد مشخصی را انتخاب کرد.
- شبکه قابلیت نوشته شدن در فایل و بارگذاری شدن از فایل را دارد.

در ادامه قسمت‌های مختلف کلاس MLP موجود در فایل mlp.py را توضیح می‌دهم.

در تصویر زیر کلاس MLP و متدهایش را مشاهده می‌کنید که در ادامه آن‌ها را توضیح می‌دهم:

```
y x mlp.py x
class MLP:
    def __init__(self, n, activations, type_of_cost='MSE', learning_rate=0.01):
        self.n = n
        self.activations = activations
        self.type_of_cost = type_of_cost
        self.learning_rate = learning_rate
        self.w = []
        self.b = []
        self.z = []
        self.a = []
        self.I = []
        self.Y = []
        self.grad_w = []
        self.grad_b = []
        self.cost = 0.0
    def cost_function(self, target, y):
        if self.type_of_cost == 'MSE':
            return 0.5 * ((y - target) ** 2).sum()
        else:
            raise ValueError("Unknown cost function")
    def forward_propagation(self, X):
        self.I[0] = X
        self.a[0] = X
        for i in range(1, self.n):
            self.w.append(np.random.rand(self.I[i-1].shape[1], self.I[i].shape[1]))
            self.b.append(np.zeros((1, self.I[i].shape[1])))
            self.I[i] = np.dot(self.I[i-1], self.w[i]) + self.b[i]
            self.a[i] = self.activations[i](self.I[i])
        self.Y = self.a[-1]
    def backward_propagation(self, target, catch_I, catch_Y):
        self.forward_propagation(target)
        self.error = self.Y - target
        self.dJ_dY = self.error
        for i in range(self.n-1, 0, -1):
            self.dJ_dY = np.dot(self.dJ_dY, self.w[i].T)
            self.dJ_dY = self.dJ_dY + self.error * self.activations[i].d_prime(self.I[i])
            self.error = self.dJ_dY
        self.dJ_dw = []
        self.dJ_db = []
        for i in range(1, self.n):
            self.dJ_dw.append(np.dot(self.a[i-1].T, self.dJ_dY))
            self.dJ_db.append(self.dJ_dY.sum(axis=0))
        self.dJ_dY = None
    def backward_propagation_stochastic(self, target, catch_I, catch_Y):
        self.forward_propagation(target)
        self.error = self.Y - target
        self.dJ_dY = self.error
        for i in range(self.n-1, 0, -1):
            self.dJ_dY = np.dot(self.dJ_dY, self.w[i].T)
            self.dJ_dY = self.dJ_dY + self.error * self.activations[i].d_prime(self.I[i])
            self.error = self.dJ_dY
        self.dJ_dw = []
        self.dJ_db = []
        for i in range(1, self.n):
            self.dJ_dw.append(np.dot(self.a[i-1].T, self.dJ_dY))
            self.dJ_db.append(self.dJ_dY.sum(axis=0))
        self.dJ_dY = None
    def fit_batch(self, X, y):
        self.backward_propagation(X, self.I, self.Y)
        self.w = [w - lr * dw for w, dw in zip(self.w, self.dJ_dw)]
        self.b = [b - lr * db for b, db in zip(self.b, self.dJ_db)]
    def fit_momentum_stochastic(self, X, y):
        self.backward_propagation_stochastic(X, self.I, self.Y)
        self.w = [w - lr * dw for w, dw in zip(self.w, self.dJ_dw)]
        self.b = [b - lr * db for b, db in zip(self.b, self.dJ_db)]
    def fit_steepest_descent(self, X, y):
        self.backward_propagation(X, self.I, self.Y)
        self.w = [w - lr * dw for w, dw in zip(self.w, self.dJ_dw)]
        self.b = [b - lr * db for b, db in zip(self.b, self.dJ_db)]
    def fit(self, X, y):
        self.fit_steepest_descent(X, y)
    def update_weight_bias_normal(self, grad_w, grad_b):
        self.w = [w - lr * dw for w, dw in zip(self.w, grad_w)]
        self.b = [b - lr * db for b, db in zip(self.b, grad_b)]
    def update_weight_bias_momentum(self, grad_w, grad_b):
        self.w = [w - lr * dw for w, dw in zip(self.w, grad_w)]
        self.b = [b - lr * db for b, db in zip(self.b, grad_b)]
    def predict(self, X):
        self.I[0] = X
        self.a[0] = X
        for i in range(1, self.n):
            self.I[i] = np.dot(self.I[i-1], self.w[i]) + self.b[i]
            self.a[i] = self.activations[i](self.I[i])
        return self.a[-1]
    def save(self):
        np.savez('mlp.npz', w=self.w, b=self.b)
    def load(self, file_name):
        npz = np.load(file_name)
        self.w = npz['w']
        self.b = npz['b']
    def get_number_of_layer(self):
        return len(self.w) + 1
    def get_weights(self):
        return self.w
    def get_bias(self):
        return self.b
    def get_number_of_neurons(self):
        return self.n
    def get_activations_of_layers(self):
        return self.activations
```

```

def __init__(self, n, activations, type_of_cost='MSE', learning_rate=0.01, max_epoch=300, mode='batch',
            random_state=None, plot_info=False, cut_cost=None):
    """
    Constructor of class MLP
    :param n: is a list which n[i] demonstrates number of neurons
              in layer i, n[0] is number of features
    :param activations: is a list which activations[i] is a string
                         that is activation of layer i
    :param type_of_cost: a string 'MSE' or 'cross_entropy'
    :param learning_rate: Learning rate for updating Learnable parameters
    :param max_epoch: maximum epoch that the network is allowed to do
    :param mode: is a string that is equal 'batch' or 'stochastic' or 'momentum' or 'steepest descent'
    :param random_state: random state for generating weights and biases and making algorithms
                         reproducible for different runs.
    :param plot_info: if it be true, MLP class saves all weights and biases for each epoch
    :param cut_cost: if it be none, it will be ignored, otherwise during training, if cost fall under
                     cost_cut, training is finished.
    """

```

این تابع constructor کلاس MLP است که آرگمان‌های مختلف آن در تصویر شرح داده شده‌است. این متاد از کلاس پارامترهای مختلف شبکه را از طریق آرگمان‌های ورودی می‌گیرد و آن‌ها را در ویژگی‌های کلاس ذخیره می‌کند.

```

def cost_function(self, target, y):
    """
    Cost function of MLP can be MSE and cross_entropy
    :param target: actual labels (correct labels)
    :param y: output labels of MLP
              [instance1 LABEL, instance2 LABEL, ..., instanceN LABEL]
    """

```

این تابع cost function شبکه است که با توجه به مقدار ویژگی نوع تابع هزینه کلاس، هزینه از نوع MSE و Cross Entropy را می‌تواند حساب کند.

محاسبه‌ی :MSE

$$J(\theta) = \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2$$

محاسبه‌ی :Cross Entropy

$$C = -\frac{1}{n} \sum_x [y \ln a + (1 - y) \ln(1 - a)]$$

```
def forward_propagation(self, X):
    """
    Do forward propagation
    :param X: is an numpy ndarray of instance(s) in form:
              [instance1, instance2, ..., instanceN]
    :return:   a list that contains net input(I) of each layer
              a list that contains output(Y) of each layer
    """

```

این تابع نمونه‌های X را می‌گیرد و عمل forward propagation را انجام می‌دهد و خروجی‌های شبکه عصبی را برای آن نمونه‌ها محاسبه می‌کند. این تابع می‌تواند یک نمونه و یا بیش از یک نمونه دریافت کند و خروجی شبکه را برای آن‌ها محاسبه کند. این تابع از Vectorization استفاده می‌کند تا سرعت انجام محاسبات را تا حد قابل قبولی افزایش دهد.

از آنجایی که در زمانه محاسبه‌ی گرادیان‌ها و انجام back propagation به مقدار خروجی و net input هر لایه نیاز داریم، این تابع خروجی و net input تمام لایه‌ها را باز می‌گرداند.

```
def backward_propagation(self, target, catch_I, catch_Y):
    """
    This function does backward propagation and calculated gradients
    :param catch_I: catch for net input(I) of each layer during forward propagation
    :param catch_Y: catch for output(Y) of each layer during forward propagation
    :param target: true labels of instance(s)
    :return
    """

```

این تابع خروجی و net input هر لایه و مقدار واقعی برچسب نمونه‌ها را دریافت می‌کند و با توجه به اینکه تابع هزینه‌ی کلاس از نوع Cross Entropy و یا MSE است گرادیان‌های وزن‌ها (weights) و بایاس‌ها (biases) را محاسبه می‌کند. این تابع می‌تواند هم به صورت batch و هم به صورت stochastic اقدام به محاسبه گرادیان‌ها بکند.

```
def backward_propagation_stochastic(self, target, catch_I, catch_Y):
```

این تابع دقیقاً مانند تابع قبل است با این تفاوت که تابع قبل هم به صورت stochastic و هم به صورت batch عمل می‌کرد ولی این تابع فقط به صورت stochastic عمل می‌کند.

```
def update_weight_bias_normal(self, grad_w, grad_b):
```

```
    """
```

```
    This function gets gradient of w and b and updates them
```

```
    :param grad_w:
```

```
    :param grad_b:
```

```
    :return:
```

```
    """
```

این تابع گرادیان‌های محاسبه شده توسط تابع back propagation را می‌گیرد و وزن‌ها و بایاس‌ها را آپدیت می‌کند. وزن‌ها و بایاس‌های لایه‌ی شبکه بدین صورت آپدیت می‌شوند:

```
weights[l] = weights[l] - learning rate * grad w[l]
```

```
bias[l] = bias[l] - learning rate * grad b[l]
```

```
def update_weight_bias_momentum(self, grad_w, grad_b):
```

```
    """
```

```
    This function gets gradient of w and b and updates them.
```

```
    It considers momentum for updating.
```

```
    :param grad_w:
```

```
    :param grad_b:
```

```
    :return:
```

```
    """
```

این تابع گرادیان‌های محاسبه شده توسط تابع back propagation را می‌گیرد و وزن‌ها و بایاس‌ها را با توجه به روش momentum محاسبه می‌کند.

```

def fit_batch(self, X, y):
    """
    Learn Learnable parameters by using back propagation in batch mode
    :param X: training dataset, each column is an observation
    :param y: label of training dataset
    :return:
    """

```

این تابع با گرفتن مجموعه‌ی آموزش و برچسب‌هایش به صورت **batch** شبکه‌ی عصبی پرسپترون چند لایه را با استفاده از forward and backward propagation آموزش می‌دهد و وزن‌ها و بایاس‌های مناسب شبکه را پیدا می‌کند.

```

def fit_momentum_stochastic(self, X, y):
    """
    Learn Learnable parameters by using back propagation with momentum
    :param X: training dataset, each column is an observation
    :param y: label of training dataset
    :return:
    """

```

این تابع با گرفتن مجموعه‌ی آموزش و برچسب‌هایش و با توجه به اینکه چگونه باید شبکه را آموزش دهیم، شبکه‌ی پرسپترون را با استفاده از stochastic forward and backward propagation آموزش می‌دهد. وزن‌ها در حالت momentum در هنگام آپدیت شدن علاوه بر گرادیان‌ها از مقدار وزن‌ها در مرحله‌های قبل نیز تاثیر می‌پذیرند.

```

def fit_steepest_descent(self, X, y):
    """
    Learn Learnable parameters by using back propagation with steepest descent
    :param X: training dataset, each column is an observation
    :param y: label of training dataset
    :return:
    """

```

این تابع شبکه را با استفاده از انتشار به جلو و عقب آپدیت می‌کند و برای این کار از روش **steepest descent** استفاده می‌کند به طوری که در هر بار در صورتی که **cost** کاهش یابد ضریب یادگیری دو برابر می‌شود و در صورتی که مقدار **cost function** کاهش نیابد ضریب یادگیری نصف می‌شود و وزن‌ها تغییر نمی‌کنند و این کار دوباره تکرار می‌شود.

```
def fit(self, X, y):
    """
    Learn learnable parameters by using back propagation
    :param X: training dataset, each column is an observation
    :param y: label of training dataset
    :return:
```

این تابع با توجه به اینکه شبکه را باید چگونه آموزش دهیم یکی از تابع‌های بالا را فرا می‌خواند.

```
def predict(self, X):
    """
    Predict label of input instance(s)
    :param X:
    :return: predicted labels
    """
```

این تابع از تابع انتشار به جلو برای محاسبه‌ی خروجی شبکه برای یک یا بیش از یک ورودی استفاده می‌کند و در آخر برای هر نمونه خروجی‌ای که بیشترین مقدار را دارد را برابر با یک قرار می‌دهد و سایر خروجی‌ها را صفر می‌کند. (One Hot)

```
def save(self):
    """
    save MLP into a file for future use, MLP is saved
    in MLP-output.json
    :return:
    """
```

این تابع تعداد لایه‌ها، تعداد نرون‌ها در هر لایه، نوع تابع فعال سازی هر لایه، وزن‌ها و بایاس‌های شبکه را در فایلی به نام `MLP-output.json` ذخیره می‌کند تا شبکه در آینده قابل بازیابی باشد.

```
def load(self, file_name):
    """
    Load trained MLP from file
    :param file_name: file should be in directory of code like: input.json
    :return:
    """
```

این تابع با گرفتن نام یک فایل `Json` که در دایرکتوری‌ای که کد قرار دارد موجود است، شبکه‌ی عصبی درون فایل را `load` می‌کند.

```
def plotting(mlp_object, trainX, trainY, validX, validY):
    """
    This function should be run inside a thread during training MLP
    Plot:
        sum of weights in each epoch
        plot cost function in each epoch
        plot train error in each epoch
        plot validation error in each epoch
    :param mlp_object: an object of class MLP
    :param trainX: training data each column is as observation
    :param trainY: training data label each column is an one hot label
    :param validX: validation data each column is as observation
    :param validY: validation data label each column is an one hot label
    :return:
    """

```

در پوشه‌ی part 1 و در فایل plotting.py تابعی برای رسم نمودار خطای آموزش، ارزیابی، مجموع وزن‌ها و بایاس‌ها و مقدار تابع هزینه نوشته‌ام. این تابع همزمان با تابع آموزش شبکه در یک **thread** موازی با برنامه فراخوانی می‌شود و وزن‌ها و بایاس‌های ایپاک‌ها مختلف را از شی کلاس MLP می‌گیرد و با محاسبه‌ی خطای مجموعه آموزش، ارزیابی، مجموع وزن‌ها و بایاس‌ها و مقدار تابع هزینه برای هر ایپاک آن‌ها را رسم می‌کند.

تابع k-fold

```
def k_fold_cross_validation(learner, argument_of_learner, X_, y_, k):
    """
    This function uses k-fold-cv to evaluate learner, learner can be
    multi-layer perceptron, perceptron, adaline and ....
    :param learner: Is an instance of multi-layer perceptron, perceptron or adaline or ... that learns
        from data to predict label of new inputs.
    :param argument_of_learner: is a list
        that contains necessary argument of
        LearningFunction.
    :param X: training data
    :param y: labels
    :param K: number of folds
    :return: return average k-fold cv error
    """
```

تابع `k_fold_cross_validation` موجود در فایل `k_fold.py` یک شی از کلاس `MLP` می‌گیرد و داده‌ها را به k قسمت تقسیم می‌کند و هر بار یک قسمت را به عنوان مجموعه‌ی ارزیابی قرار می‌دهد و قسمت‌های دیگر را برای آموزش استفاده می‌کند و شبکه را آموزش می‌دهد و خطای آن را حساب می‌کند و در آخر میانگین خطای k بار را برمی‌گرداند.

کد پیدا کردن پارامترهای مناسب برای MLP (best_MLP.py)

در پوششی part1 و در فایل best_MLP.py کدی نوشته‌ام که مشخصات یک شبکه‌ی MLP مانند تعداد لایه‌ها، تعداد نرون‌ها در هر لایه، تابع Activation هر لایه، نوع Cost Function، نوع آموزش و آپدیت وزن‌ها (...، batch, momentum) و دیتاست (اعداد و یا اعداد به علاوه‌ی حروف) را می‌گیرد و به ازای ضریب‌های یادگیری و حداکثر تعداد ایپاک‌های مختلف شبکه را آموزش می‌دهد و نتایج را در فایل ذخیره می‌کند. با استفاده از این کد برای ساختارهای مختلف شبکه‌ی عصبی ضریب یادگیری و تعداد حداچیر ایپاک‌های مناسب را پیدا می‌کنم. در تصویر زیر نمونه‌ای از خروجی این کد را برای یک شبکه‌ی سه لایه که لایه نهان آن ۲۰ نرون دارد و تابع فعال سازی آن \tanh است و لایه آخر آن ۱۰ نرون دارد و تابع فعال سازی آن sigmoid است و تابع هزینه‌ی آن MSE است و به روش momentum کار می‌کند را مشاهده می‌کنید:

```
289 =====V=====
290
291 Max Epoch:
292 40
293 Learning rate:
294 0.1
295 Train Error:
296 0.011838235294117648
297 Valid Error:
298 0.05176470588235294
299 Test Error:
300 0.04647058823529412
301 =====V=====
302
303 Max Epoch:
304 40
305 Learning rate:
306 1
307 Train Error:
308 0.20647058823529413
309 Valid Error:
310 0.22588235294117648
311 Test Error:
312 0.19588235294117648
313 =====|=|
314 Best Test Error:
315 0.041176470588235294
316 Train Error of Best Test Error:
317 0.011102941176470588
318 Validation Error of Best Test Error:
319 0.0488235294117647
320 Learning Rate of Best Test Error:
321 0.01
322 Max Epoch of Best Test Error:
323 30
```

همان طور که در تصویر بالا مشاهده می شود برای ساختار مثال بالا بهترین خطای مجموعه تست برابر با ۴ درصد است که زمانی حاصل می شود که ضریب یادگیری برابر با 1×10^{-3} باشد و حداکثر تعداد ایپاک ها برابر با ۳۰ باشد. خروجی مثال بالا در فایل result_momentum_MSE_layers [1024, 20, 10]_digits.dat موجود است.

با تغییر پارامترهای شبکه، می توان پارامترهای مناسب را برای ساختار جدید پیدا کرد.

کد و رابطه‌ی کاربری برای استفاده از کلاس MLP و رسم نمودارها (run_mlp.py)

در پوشه‌ی `part1` و در فایل `run_mlp.py` کدی نوشته‌ام که در ابتدا از کاربر می‌خواهد که مشخص کند که آیا می‌خواهد یک شبکه‌ی MLP آموزش دهد و یا یک شبکه‌ی آموزش دیده شده را از فایل بخواند. در صورتی که کاربر بخواهد یک شبکه‌ی آموزش بدهد یک فایل `json` از کاربر می‌خواهد که پارامترهای لازم برای آموزش یک شبکه در آن قرار دارد. در صورتی که کاربر بخواهد یک شبکه‌ی آموزش دیده را از فایل بخواند برنامه از کاربر اسم یک فایل `json` را می‌خواهد که یک شبکه‌ی عصبی در آن موجود است. سپس با توجه به ورودی‌ها، دیتاست عددها و یا دیتاست عددها به علاوه‌ی حرف‌ها از فایل خوانده می‌شود. در صورتی که کاربر خواسته باشد که `k-fold_cross_validation` را محاسبه کند یک `thread` برای محاسبه‌ی آن ایجاد می‌شود که تابع `fold` را فرا می‌خواند. در ادامه یک شبکه MLP از کلاس `MLP` با مشخصاتی که کاربر داده است ایجاد می‌شود و یا یک شبکه از فایل خوانده می‌شود. در صورتی که کاربر بخواهد یک شبکه آموزش دهد و در حین آموزش نمودارهای خطای آموزش، ارزیابی، مجموع وزن‌ها-بایاس‌ها و `Cost Function` را رسم کند یک `thread` ایجاد می‌شود که در آن تابع `plotting` فراخوانده می‌شود و هم زمان با آموزش شبکه، خطاهای میزان `cost function` و مجموعه وزن‌ها و بایاس‌ها را محاسبه و رسم می‌کند. در آخر خطای شبکه بر روی مجموعه‌ی آموزش، ارزیابی و تست نمایش داده می‌شود.

بررسی تاثیر تعداد نرون‌ها در یک شبکه‌ی سه لایه بر دقت دسته‌بند

عنوان: با ایجاد یک شبکه عصبی پرپرتونی سه لایه با تعداد نuron‌های لایه‌های مخفی مختلف (سه مقدار مختلف برای تعداد این نuron‌ها)، تاثیر عرض شبکه را در دقت دسته بندی این ارقام را بررسی کنید.

شرایط آزمایش: برای بررسی تاثیر تغییر تعداد نuron‌ها بر دقت شبکه سه لایه، ۷ شبکه عصبی ایجاد می‌کنم که شبکه‌ی اول در لایه‌ی نهان خود ۵ نرون دارد، شبکه‌ی دوم در لایه‌ی نهان ۱۰ نرون و شبکه‌ی سوم در لایه‌ی نهان خود ۳۰ نرون دارد و شبکه‌ی چهارم در لایه‌ی نهان خود ۶۴ نرون دارد و شبکه‌ی پنجم در لایه‌ی نهان خود ۲۰۰ نرون دارد، شبکه‌ی ششم در لایه‌ی نهان خود ۲۵۶ نرون و شبکه‌ی آخر در لایه‌ی نهان خود ۵۱۲ نرون دارد

در زیر محتوای فایل‌های config (input-0.json, input-1.json, input-2.json, input-3.json, input-4.json, input-5.json, input6.json) هفت شبکه را مشاهده می‌کنید که پارامترهای لازم برای ساخت و یادگیری در آن قرار دارند.

```
input-0.json
1 {{"digi...
2   "digi...
3   "ratio_...
4   "ratio_...
5   "ratio_...
6   "learni...
7   "max_...
8   "cut_...
9   "random...
10  "K_fold": 5,
11  "neurons_layes": [1024, 5, 10],
12  "type_layers": ["tanh", "sigmoid"],
13  "type_of_cost": "MSE",
14  "mode": "momentum"
15 }
```

شبکه‌ی شماره صفر: از دیتاست عده‌ها استفاده می‌شود، ۸۰ درصد دیتاست برای آموزش است، ۱۰ درصد دیتاست برای ارزیابی است، ۱۰ درصد دیتاست برای تست است، ضریب یادگیری برابر با ۰,۰۱ است، حداکثر تعداد ایپاک‌ها ۱۵ است، شرط کمتر شدن تابع هزینه از یک مقداری برای توقف آموزش در نظر گرفته نمی‌شود، تولید عده‌های تصادفی با عدد ۲ بذرپاشی می‌شود، تعداد فیچرها ۱۰۲۴ تا است، لایه نهان اول ۵ نرون دارد. لایه خروجی ۱۰

نرون دارد. تابع فعال سازی لایه‌ی نهان اول **tanh** است و تابع فعال سازی لایه‌ی خروجی **sigmoid** است، تابع هزینه **MSE** است، از روش **momentum** استفاده شده است.

```
input-1.json
1 {
2   "digits_letters_digits": "digits",
3   "ratio_of_train_set": 0.80,
4   "ratio_of_valid_set": 0.10,
5   "ratio_of_test_set": 0.10,
6   "learning_rate": 0.01,
7   "max_epochs": 15,
8   "cut_cost": -1,
9   "random_state": 2,
10  "K_fold": 5,
11  "neurons_layes": [1024, 10, 10],
12  "type_layers": ["tanh", "sigmoid"],
13  "type_of_cost": "MSE",
14  "mode": "momentum"
15 }
```

شبکه‌ی شماره یک: از دیتاست عده‌ها استفاده می‌شود، ۸۰ درصد دیتاست برای آموزش است، ۱۰ درصد دیتاست برای ارزیابی است، ۱۰ درصد دیتاست برای تست است، ضریب یادگیری برابر با ۰.۰۱ است، حداکثر تعداد ایپاک‌ها ۱۵ است، شرط کمتر شدن تابع هزینه از یک مقداری برای توقف آموزش در نظر گرفته نمی‌شود، تولید عده‌های تصادفی با عدد ۲ بذرپاشی می‌شود، تعداد فیچرها ۱۰۲۴ تا است، لایه نهان اول ۱۰ نرون دارد. لایه خروجی ۱۰ نرون دارد. تابع فعال سازی لایه‌ی نهان اول **tanh** است و تابع فعال سازی لایه‌ی خروجی **sigmoid** است، تابع هزینه **MSE** است، از روش **momentum** استفاده شده است.

```

input-2.json
1  {
2      "digits_letters_digits": "digits",
3      "ratio_of_train_set": 0.80,
4      "ratio_of_valid_set": 0.10,
5      "ratio_of_test_set": 0.10,
6      "learning_rate": 0.01,
7      "max_epochs": 15,
8      "cut_cost": -1,
9      "random_state": 2,
10     "K_fold": 5,
11     "neurons_layes": [1024, 30, 10],
12     "type_layers": ["tanh", "sigmoid"],
13     "type_of_cost": "MSE",
14     "mode": "momentum"
15 }

```

شبکه‌ی شماره دو: از دیتاست عددها استفاده می‌شود، ۸۰ درصد دیتاست برای آموزش است، ۱۰ درصد دیتاست برای ارزیابی است، ۱۰ درصد دیتاست برای تست است، ضریب یادگیری برابر با ۰.۰۱ است، حداکثر تعداد ایپاک‌ها ۱۵ است، شرط کمتر شدنتابع هزینه از یک مقداری برای توقف آموزش در نظر گرفته نمی‌شود، تولید عددهای تصادفی با عدد ۲ بذرپاشی می‌شود، تعداد فیچرها ۱۰۲۴ تا است، لایه نهان اول ۳۰ نرون دارد. لایه خروجی ۱۰ نرون دارد. تابع فعال سازی لایه نهان اول **tanh** است و تابع فعال سازی لایه خروجی **sigmoid** است، تابع هزینه **MSE** است، از روش **momentum** استفاده شده است.

input-3.json

```

1  {
2      "digits_letters_digits": "digits",
3      "ratio_of_train_set":0.80,
4      "ratio_of_valid_set":0.10,
5      "ratio_of_test_set":0.10,
6      "learning_rate":0.01,
7      "max_epochs":15,
8      "cut_cost": -1,
9      "random_state":2,
10     "K_fold": 5,
11     "neurons_layes": [1024, 64, 10],
12     "type_layers": ["tanh", "sigmoid"],
13     "type_of_cost": "MSE",
14     "mode": "momentum"
15 }

```

شبکه‌ی شماره سوم: از دیتاست عده‌ها استفاده می‌شود، ۸۰ درصد دیتاست برای آموزش است، ۱۰ درصد دیتاست برای ارزیابی است، ۱۰ درصد دیتاست برای تست است، ضریب یادگیری برابر با ۰.۰۱ است، حداکثر تعداد ایپاک‌ها ۱۵ است، شرط کمتر شدن تابع هزینه از یک مقداری برای توقف آموزش در نظر گرفته نمی‌شود، تولید عده‌های تصادفی با عدد ۲ بذرپاشی می‌شود، تعداد فیچرها ۱۰۲۴ تا است، لایه نهان اول ۶۴ نرون دارد. لایه خروجی ۱۰ نرون دارد. تابع فعال سازی لایه نهان اول tanh است و تابع فعال سازی لایه خروجی sigmoid است، تابع هزینه MSE است، از روش momentum استفاده شده است.

```

input-4.json
1  {
2      "digits_letters_digits": "digits",
3      "ratio_of_train_set": 0.80,
4      "ratio_of_valid_set": 0.10,
5      "ratio_of_test_set": 0.10,
6      "learning_rate": 0.01,
7      "max_epochs": 15,
8      "cut_cost": -1,
9      "random_state": 2,
10     "K_fold": -1,
11     "neurons_layes": [1024, 140, 10],
12     "type_layers": ["tanh", "sigmoid"],
13     "type_of_cost": "MSE",
14     "mode": "momentum"
15 }

```

شبکه‌ی شماره چهارم: از دیتاست عده‌ها استفاده می‌شود، ۸۰ درصد دیتاست برای آموزش است، ۱۰ درصد دیتاست برای ارزیابی است، ۱۰ درصد دیتاست برای تست است، ضریب یادگیری برابر با ۰.۰۱ است، حداکثر تعداد ایپاک‌ها ۱۵ است، شرط کمتر شدنتابع هزینه از یک مقداری برای توقف آموزش در نظر گرفته نمی‌شود، تولید عده‌های تصادفی با عدد ۲ بذرپاشی می‌شود، تعداد فیچرها ۱۰۲۴ تا است، لایه نهان اول ۱۴۰ نرون دارد. لایه خروجی ۱۰ نرون دارد. تابع فعال سازی لایه نهان اول tanh است و تابع فعال سازی لایه خروجی sigmoid است، تابع هزینه MSE است، از روش momentum استفاده شده است.

input-5.json

```
1  {
2      "digits_letters_digits": "digits",
3      "ratio_of_train_set": 0.80,
4      "ratio_of_valid_set": 0.10,
5      "ratio_of_test_set": 0.10,
6      "learning_rate": 0.01,
7      "max_epochs": 15,
8      "cut_cost": -1,
9      "random_state": 2,
10     "K_fold": -1,
11     "neurons_layes": [1024, 256, 10],
12     "type_layers": ["tanh", "sigmoid"],
13     "type_of_cost": "MSE",
14     "mode": "momentum"
15 }
```

شبکه‌ی شماره پنجم: از دیتاست عده‌ها استفاده می‌شود، ۸۰ درصد دیتاست برای آموزش است، ۱۰ درصد دیتاست برای ارزیابی است، ۱۰ درصد دیتاست برای تست است، ضریب یادگیری برابر با ۰.۰۱ است، حداکثر تعداد ایپاک‌ها ۱۵ است، شرط کمتر شدن تابع هزینه از یک مقداری برای توقف آموزش در نظر گرفته نمی‌شود، تولید عده‌های تصادفی با عدد ۲ بذرپاشی می‌شود، تعداد فیچرها ۱۰۲۴ تا است، لایه نهان اول ۲۵۶ نرون دارد. لایه خروجی ۱۰ نرون دارد. تابع فعال سازی لایه نهان اول **tanh** است و تابع فعال سازی لایه خروجی **sigmoid** است، تابع هزینه **MSE** است، از روش **momentum** استفاده شده است.

input-6.json

```

1  {
2      "digits_letters_digits": "digits",
3      "ratio_of_train_set":0.80,
4      "ratio_of_valid_set":0.10,
5      "ratio_of_test_set":0.10,
6      "learning_rate":0.01,
7      "max_epochs":15,
8      "cut_cost": -1,
9      "random_state":2,
10     "K_fold": -1,
11     "neurons_layes": [1024, 512, 10],
12     "type_layers": ["tanh", "sigmoid"],
13     "type_of_cost": "MSE",
14     "mode": "momentum"
15 }

```

شبکه‌ی شماره ششم: از دیتاست عده‌ها استفاده می‌شود، ۸۰ درصد دیتاست برای آموزش است، ۱۰ درصد دیتاست برای ارزیابی است، ۱۰ درصد دیتاست برای تست است، ضریب یادگیری برابر با ۰.۰۱ است، حداکثر تعداد ایپاک‌ها ۱۵ است، شرط کمتر شدن تابع هزینه از یک مقداری برای توقف آموزش در نظر گرفته نمی‌شود، تولید عده‌های تصادفی با عدد ۲ بذرپاشی می‌شود، تعداد فیچرها ۱۰۲۴ تا است، لایه نهان اول ۵۱۲ نرون دارد. لایه خروجی ۱۰ نرون دارد. تابع فعال سازی لایه نهان اول **tanh** است و تابع فعال سازی لایه خروجی **sigmoid** است، تابع هزینه **MSE** است، از روش **momentum** استفاده شده است.

میزان ضریب یادگیری و حداکثر تعداد ایپاک‌ها بر اساس کد **best_MLP.py** به دست آمده است.

نتیجه انجام آزمایش:

نتیجه‌های شبکه عصبی صفرم:

مقدار تابع هزینه در آخرین ایپاک، خطای آموزش، خطای ارزیابی، خطای تست، خطای 5-fold :

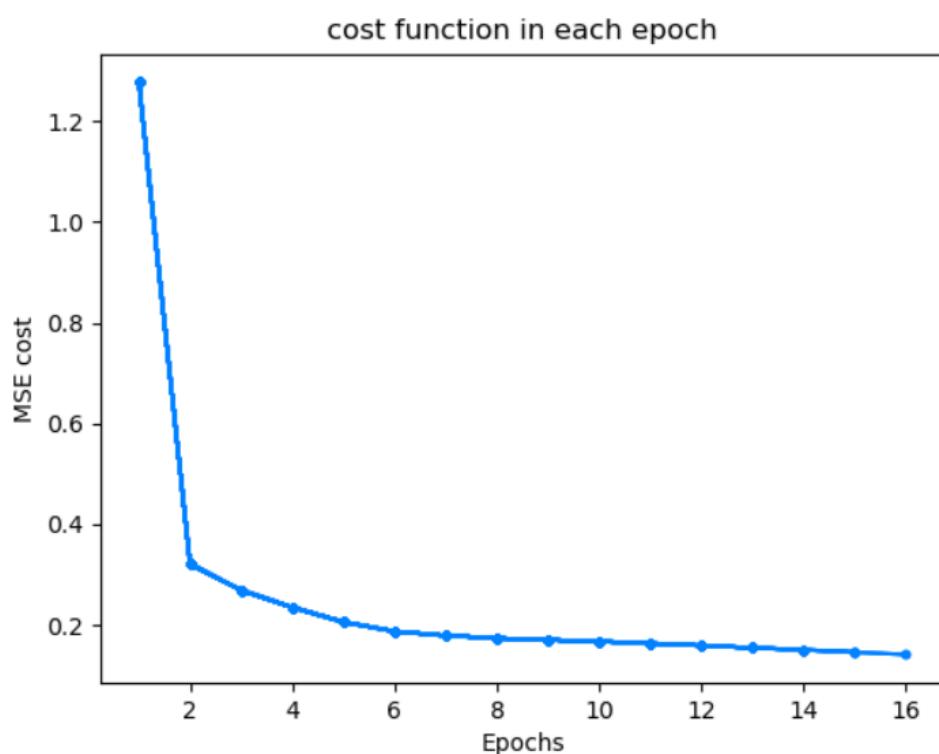
```
Iter 14/15, cost 0.146413183259181/6 ...
Iter 15/15, Cost 0.14179115731741884 ...
=====
Train Error: 0.1763970588235294
Validation Error: 0.20352941176470588
Test Error: 0.18823529411764706
```

5-fold error is: 0.20022058823529415

```
=====
^=====
```

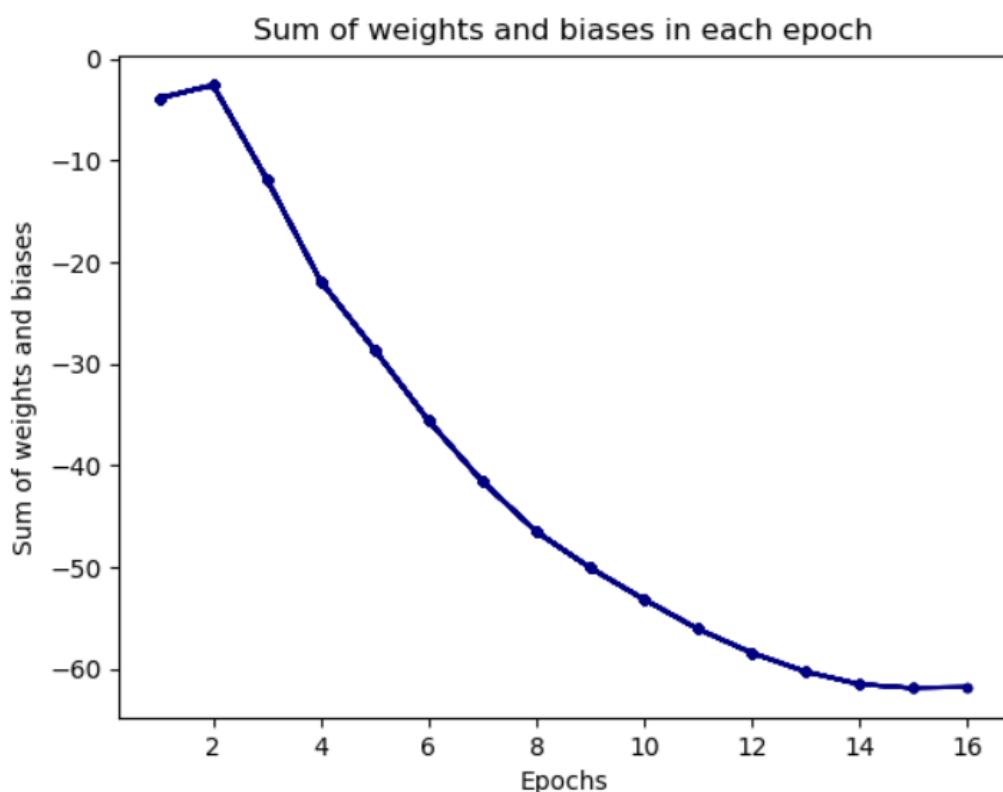
مقدار تابع هزینه بعد از آخرین ایپاک برابر با ۰,۱۴ است. خطای مجموعه آموزش برابر با ۰,۱۷ است، خطای 5-fold برابر با ۰,۲۰ درصد است، خطای مجموعه ارزیابی برابر با ۰,۳۵ درصد است و خطای مجموعه تست برابر با ۰,۲۰ درصد است. تعداد کم نرون‌ها در لایه نهان باعث شده است شبکه قادر به یادگیری مناسب نباشد.

مقدار تابع هزینه در هر ایپاک:



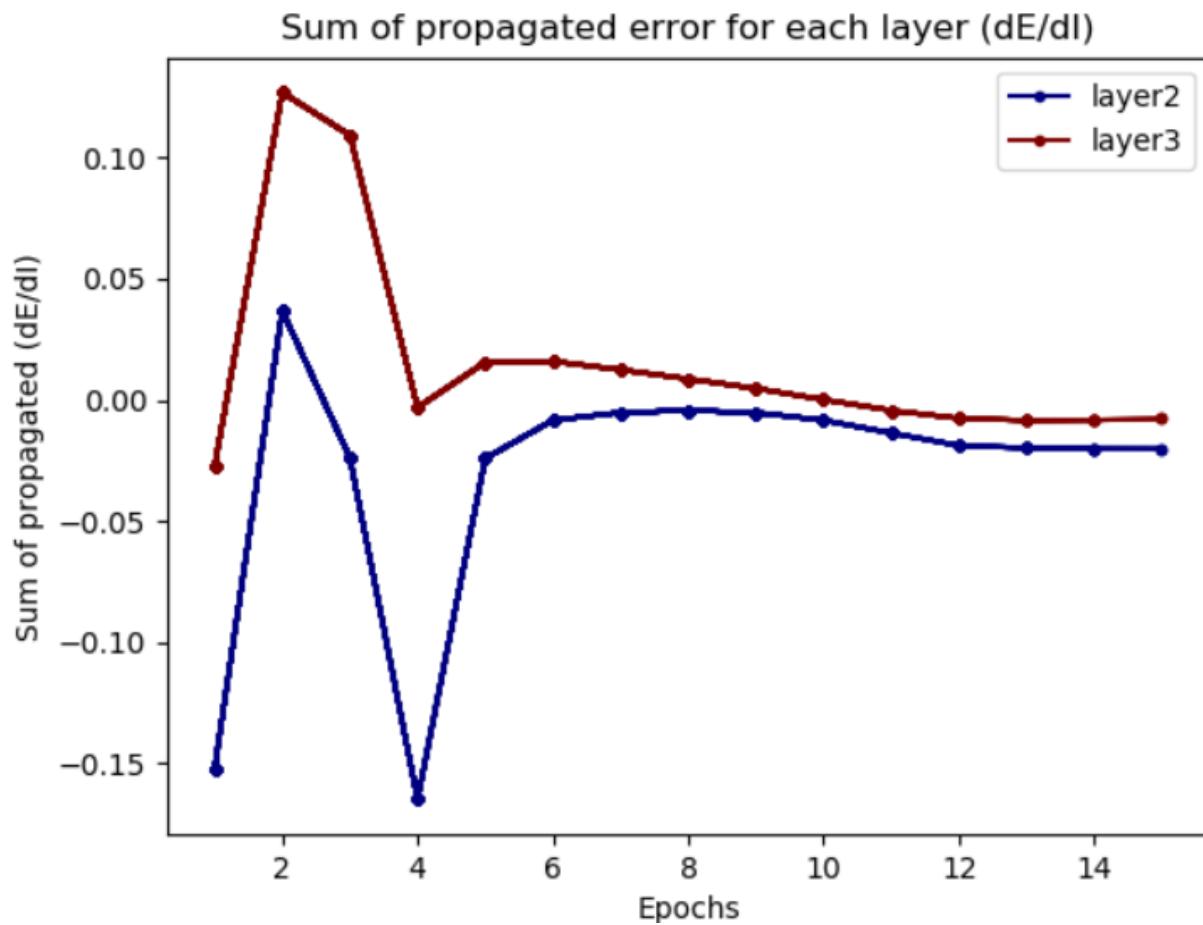
در طی ۱۵ ایپاک تابع هزینه MSE از ۱,۲ به ۰,۱۴ رسیده است که میزان کاهش مناسب نیست. زیرا در جلوتر می‌بینیم با افزایش نرون‌ها این مقدار کمتر می‌شود. مقدار تابع هزینه سیر نزولی داشته است و بیشترین تغییرات در زمان انجام ایپاک اول بوده است و هر چه جلوتر رفته‌ایم در ایپاک‌های بعد از میزان تغییرات کاسته شده است و با روند کند تری تابع هزینه کم شده است.

مجموع وزن‌ها و بایاس‌ها در هر ایپاک:



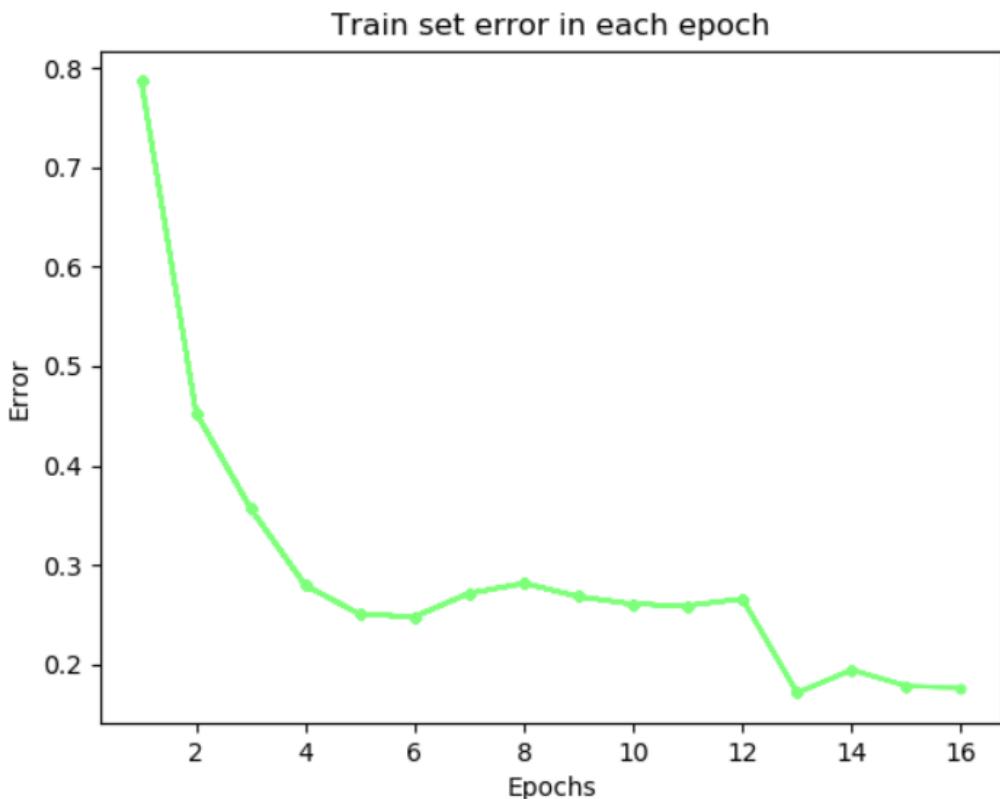
در ایپاک‌های آخر میزان تغییران مجموعه وزن‌ها و بایاس‌ها کم شده‌است و تقریباً در آن محدوده باقی می‌ماند.

مجموع خطای منتشر شده در هر لایه:



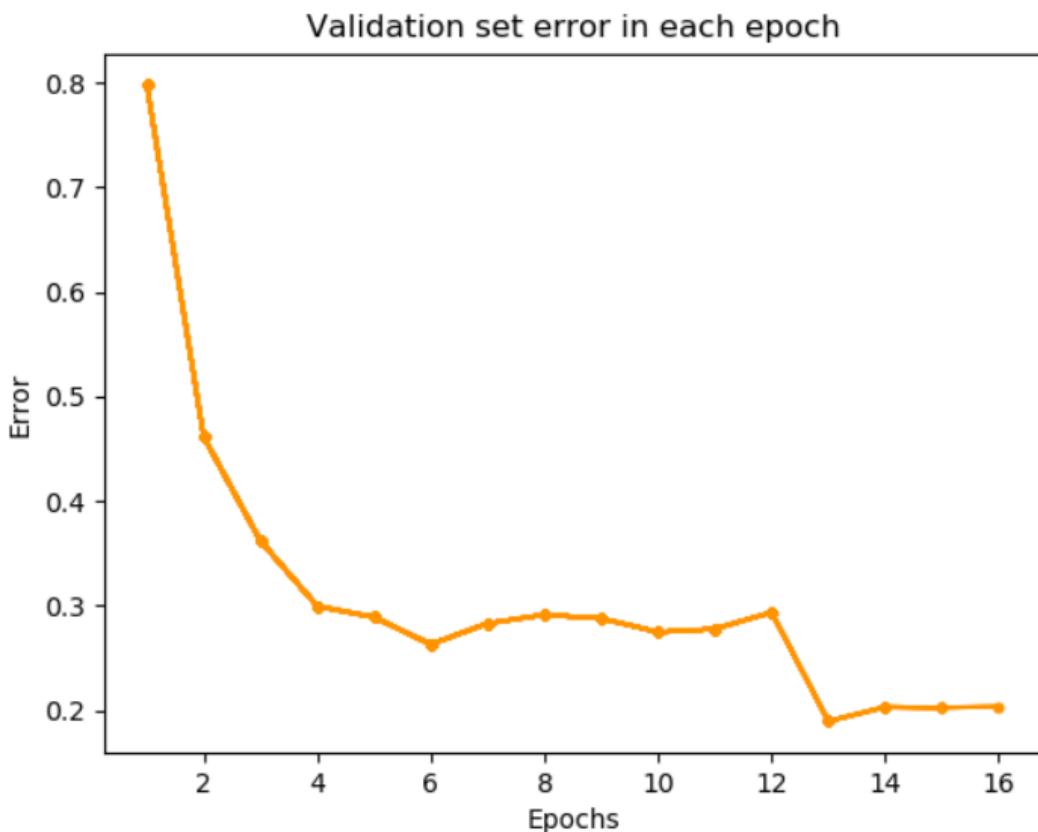
با گذر زمان میزان تغییرات گرادیان‌ها کم می‌شود و به صفر میل می‌کند.

خطای مجموعه‌ی آموزش در هر ایپاک:



خطای مجموعه آموزش به طور کلی و با آموزش بیشتر کاهش داشته است. بیشتر کاهش در ایپاک اول رخ داده است و در ایپاک های آخر میزان تغییرات خطأ کم شده است و به مقدار ثابتی رسیده است.

خطای مجموعه ارزیابی در هر ایپاک:



خطای ارزیابی در هر ایپاک با آموزش بیشتر کاهش داشته است فقط در ایپاک ۱۱ و ۱۳ مقداری افزایش داشته است. در ایپاک‌های اول بیشترین تغییرات را داشته‌ایم و در ایپاک‌های پایانی از میزان تغییرات کاسته شده است.

این شبکه در فایل MLP-output-0.json ذخیره شده است.

نتیجه‌های شبکه عصبی یکم:

مقدار تابع هزینه در آخرین ایپاک، خطای آموزش، خطای ارزیابی، خطای تست :

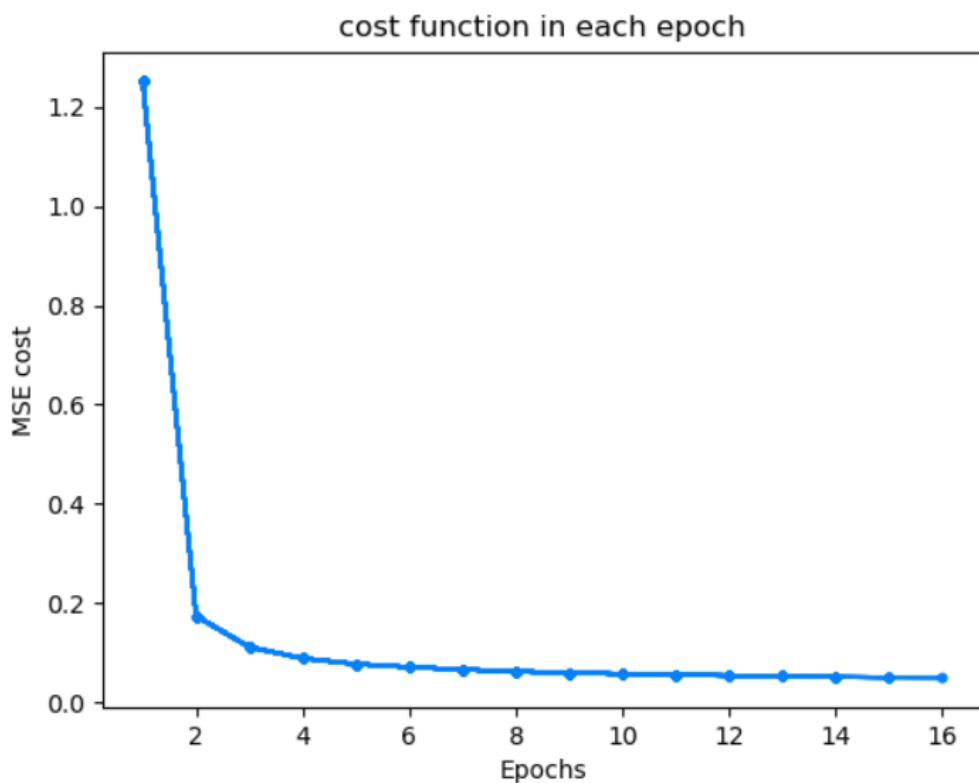
```

Iter 14/15, Cost 0.049700954502338815 ...
Iter 15/15, Cost 0.0487928132747739 ...
=====
Train Error: 0.049779411764705885
Validation Error: 0.08294117647058824
Test Error: 0.07294117647058823

5-fold error is: 0.05970588235294118
=====^=====

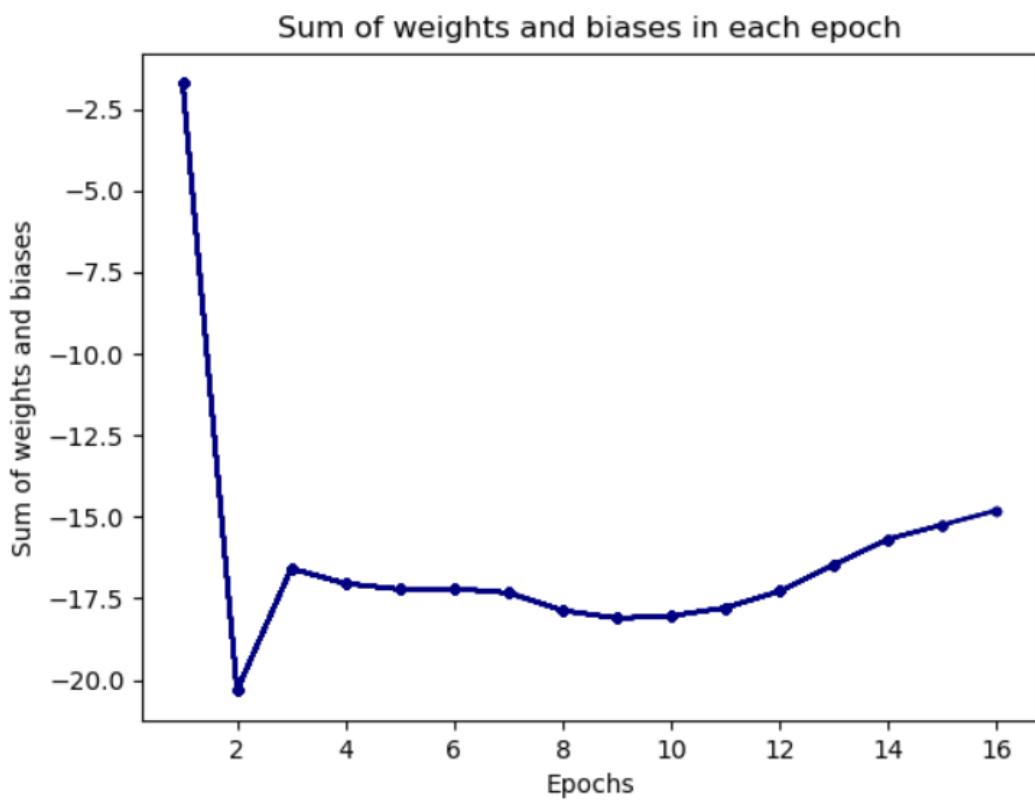
```

مقدار تابع هزینه بعد از آخرین ایپاک برابر با 0.0487928132747739 درصد است، خطای مجموعه آموزش برابر با 0.049779411764705885 درصد است، خطای مجموعه ارزیابی برابر با 0.08294117647058824 درصد است. خطای 5-fold برابر با 0.07294117647058823 درصد است. مقدار تابع هزینه در هر ایپاک:



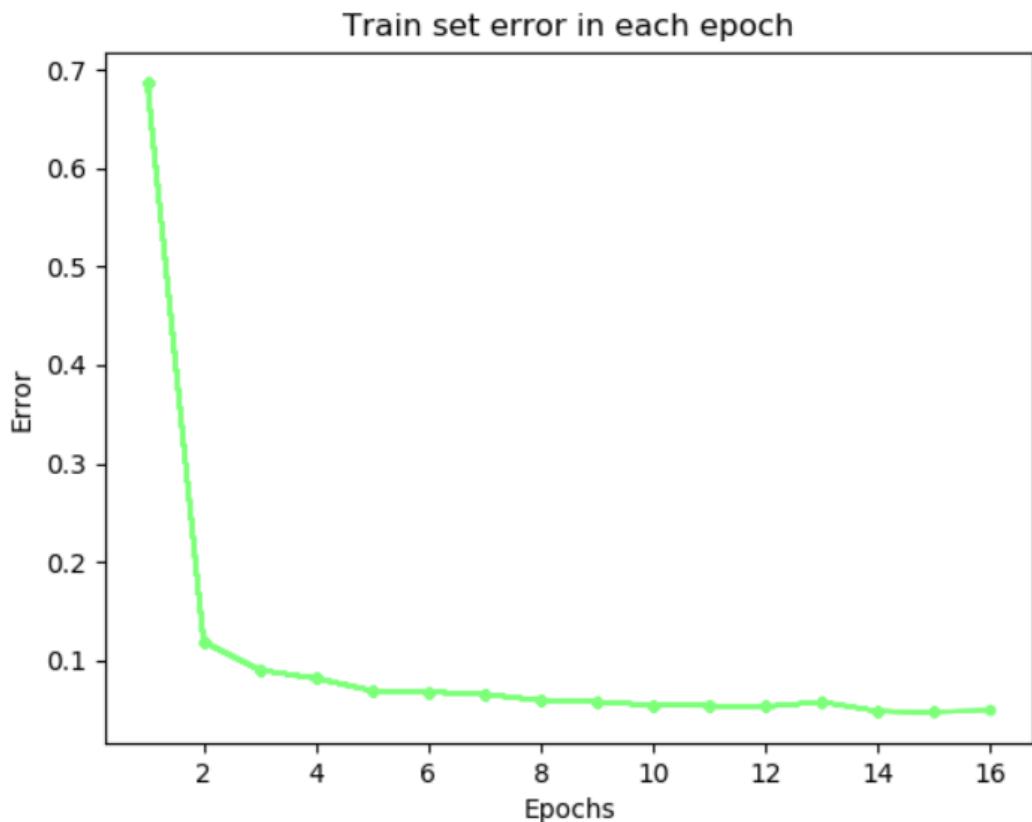
در طی ۱۵ ایپاک تابع هزینه MSE از ۱,۲ به ۰,۰۴۸ رسیده است که میزان کاهش چشمگیر است. مقدار تابع هزینه سیر نزولی داشته است و بیشترین تغییرات در زمان انجام ایپاک اول بوده است و هر چه جلوتر رفتهایم در ایپاک‌های بعد از میزان تغییرات کاسته شده است و با روند کندتری تابع هزینه کم شده است.

مجموع وزن‌ها و بایاس‌ها در هر ایپاک:



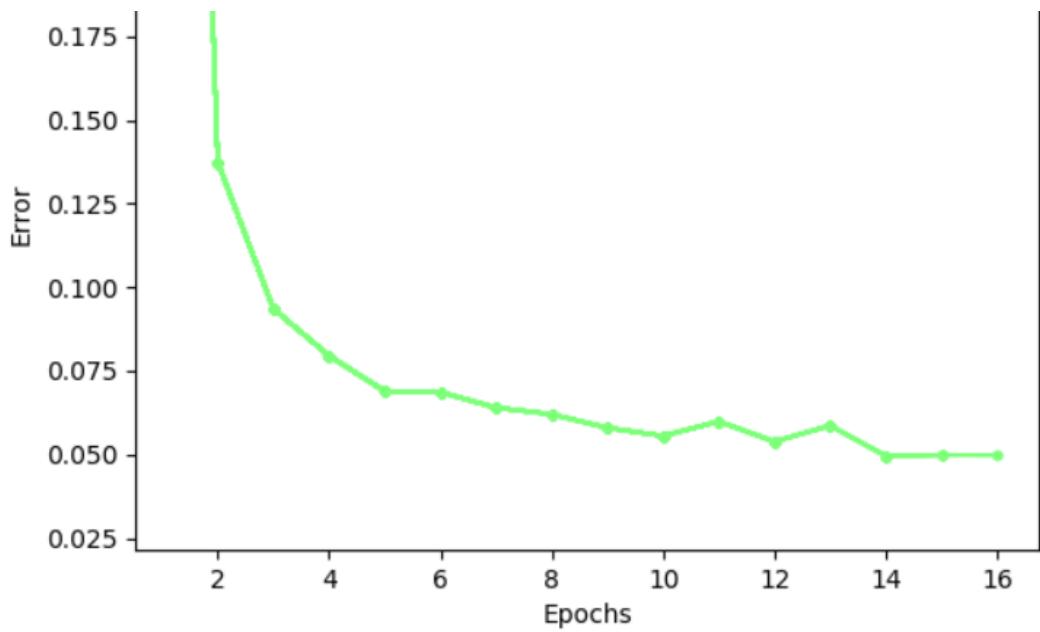
در ایپاک‌های آخر میزان تغییران مجموعه وزن‌ها و بایاس‌ها کم شده است و تقریباً در آن محدوده باقی می‌ماند. زیرا هر چه جلوتر می‌رویم معمولاً گرادیان‌ها کوچک‌تر می‌شوند و تغییرات وزن‌ها نسبت به مراحل قبل کمتر می‌شود.

خطای مجموعه‌ی آموزش در هر ایپاک:

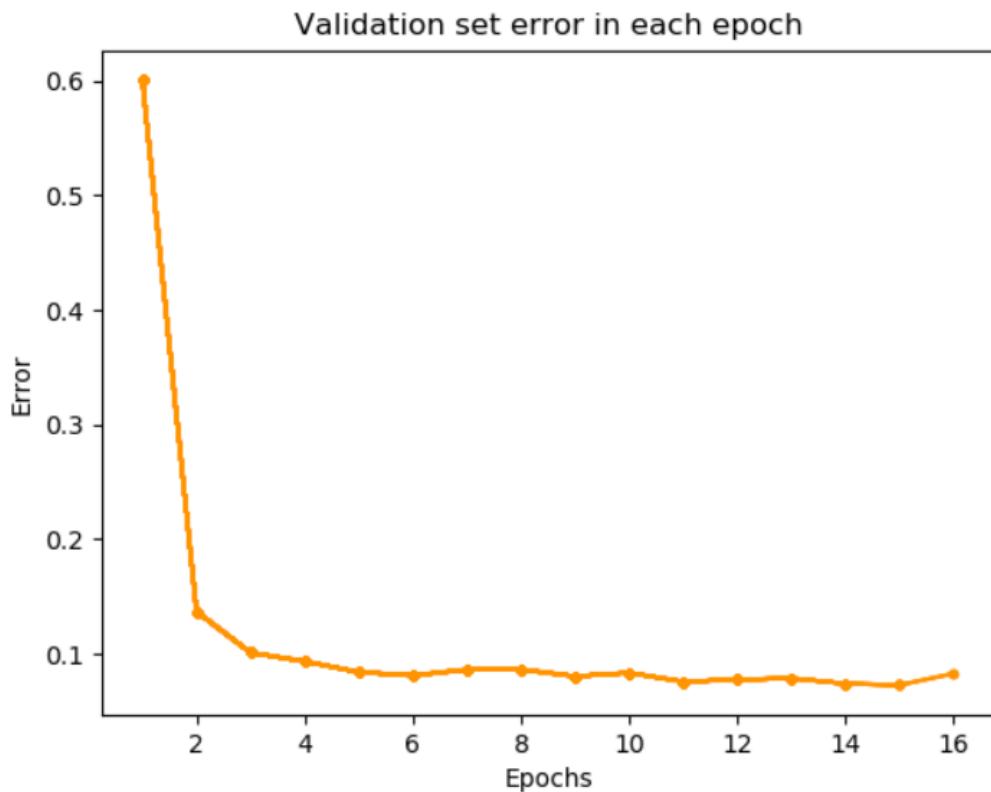


خطای مجموعه آموزش با آموزش بیشتر شبکه کاهش داشته است. بیشتر کاهش در ایپاک اول رخ داده است و در ایپاک های آخر میزان تغییرات خطا کم شده است و به مقدار ثابتی رسیده است.

بخشی از نمودار بالا با بزرگنمایی:

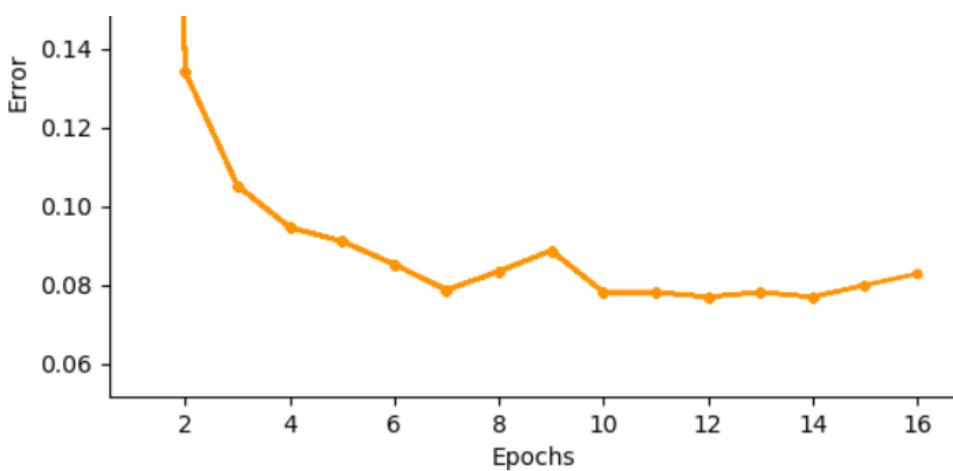


خطای مجموعه ارزیابی در هر ایپاک:



خطای ارزیابی در هر ایپاک با آموزش بیشتر، کاهش داشته است. در ایپاک‌های اول بیشترین تغییرات را داشته‌ایم و در ایپاک‌های پایانی از میزان تغییرات کاسته شده است.

بخشی از نمودار بالا با بزرگنمایی:



این شبکه در فایل MLP-output-1.json ذخیره شده است.

نتیجه‌های شبکه عصبی دوم:

مقدار تابع هزینه در آخرین ایپاک، خطای آموزش، خطای ارزیابی، خطای تست :

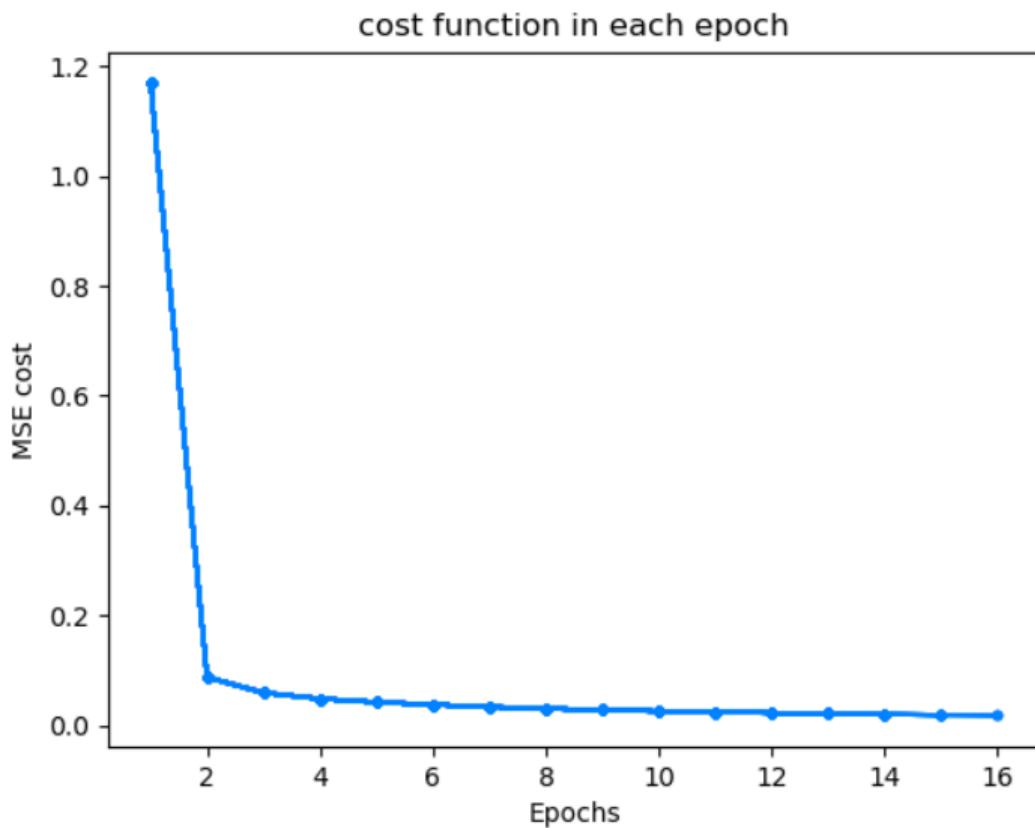
```
Iter 13/15, Cost 0.019832292400507632 ...
Iter 14/15, Cost 0.018756050092906682 ...
Iter 15/15, Cost 0.017810198088021756 ...
=====
Train Error: 0.01375
Validation Error: 0.03882352941176471
Test Error: 0.029411764705882353

5-fold error is: 0.01926470588235294
```

```
=====
|
```

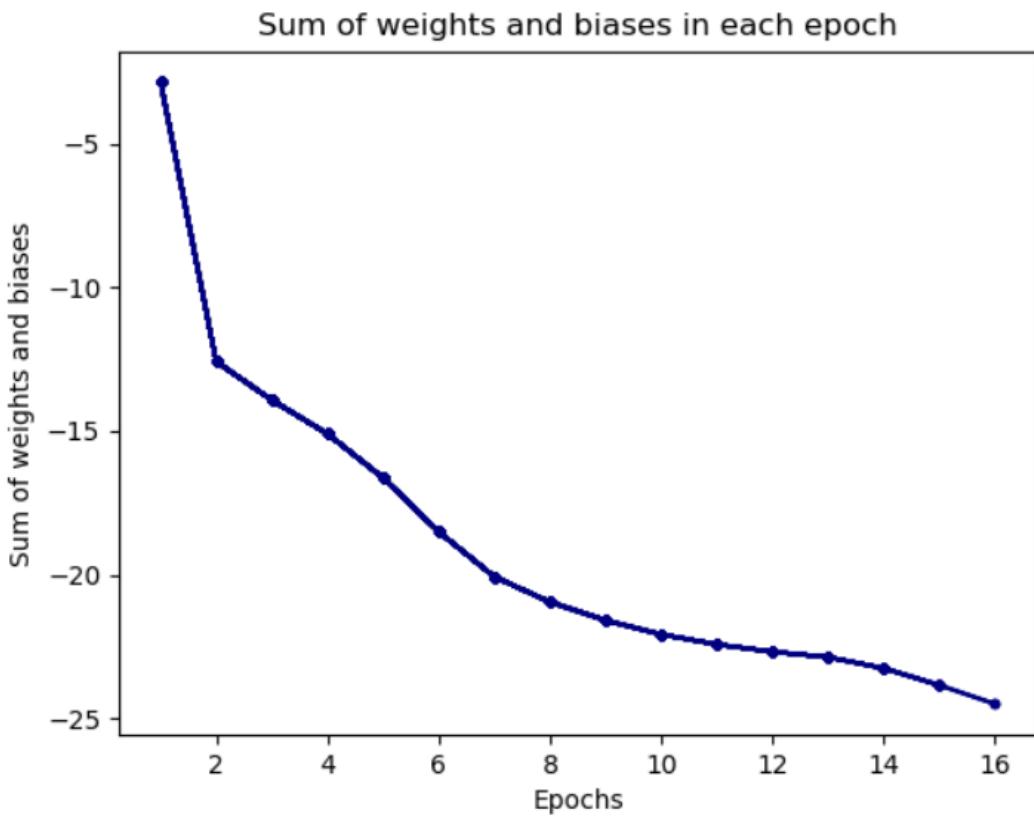
مقدار تابع هزینه بعد از آخرین ایپاک برابر با 0.017 است. خطای مجموعه آموزش برابر با ۱,۳ درصد است، خطای مجموعه ارزیابی برابر با ۳,۸ درصد است و خطای مجموعه تست برابر با ۲,۹ درصد است. خطای k-fold برابر با ۱,۹ درصد است.

مقدار تابع هزینه در هر ایپاک:



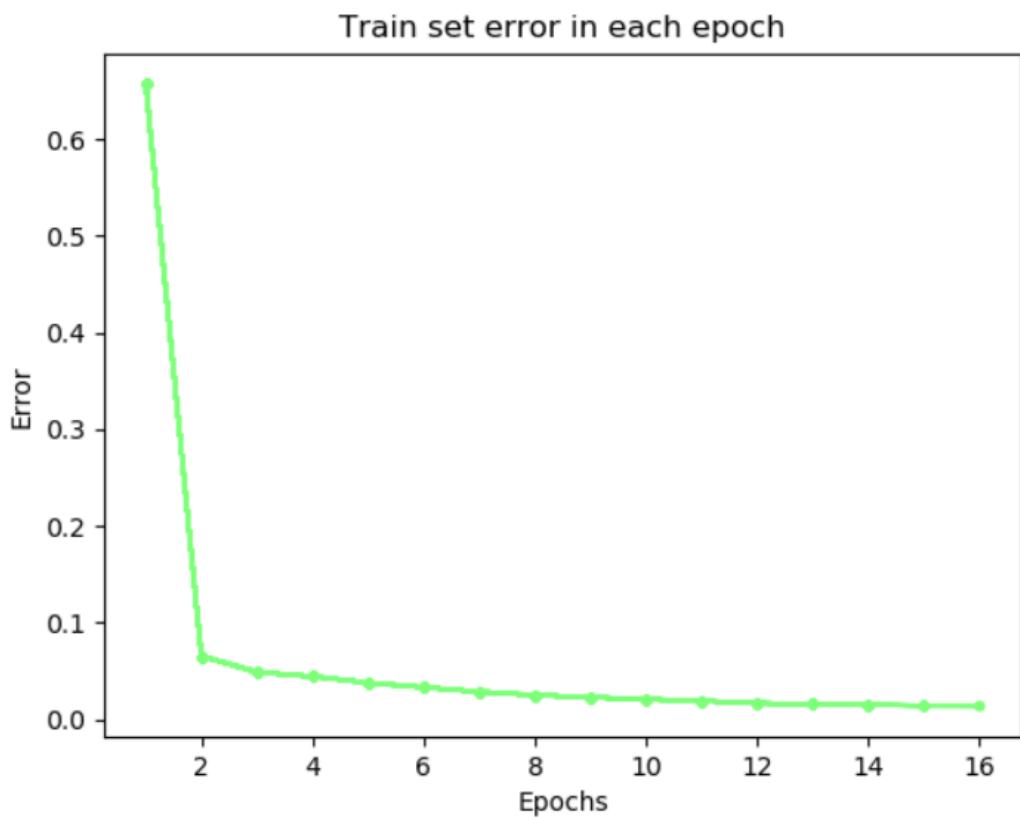
در طی ۱۵ ایپاک تابع هزینه‌ی MSE از ۱,۲ به ۰,۰۱ رسیده است که میزان کاهش چشم‌گیر است. مقدار تابع هزینه سیر نزولی داشته است و بیشترین تغییرات در زمان انجام ایپاک اول بوده است و هر چه جلوتر رفته‌ایم در ایپاک‌های بعد از میزان تغییرات کاسته شده است و با روند کند تری تابع هزینه کم شده است.

مجموع وزن‌ها و بایاس‌ها در هر ایپاک:



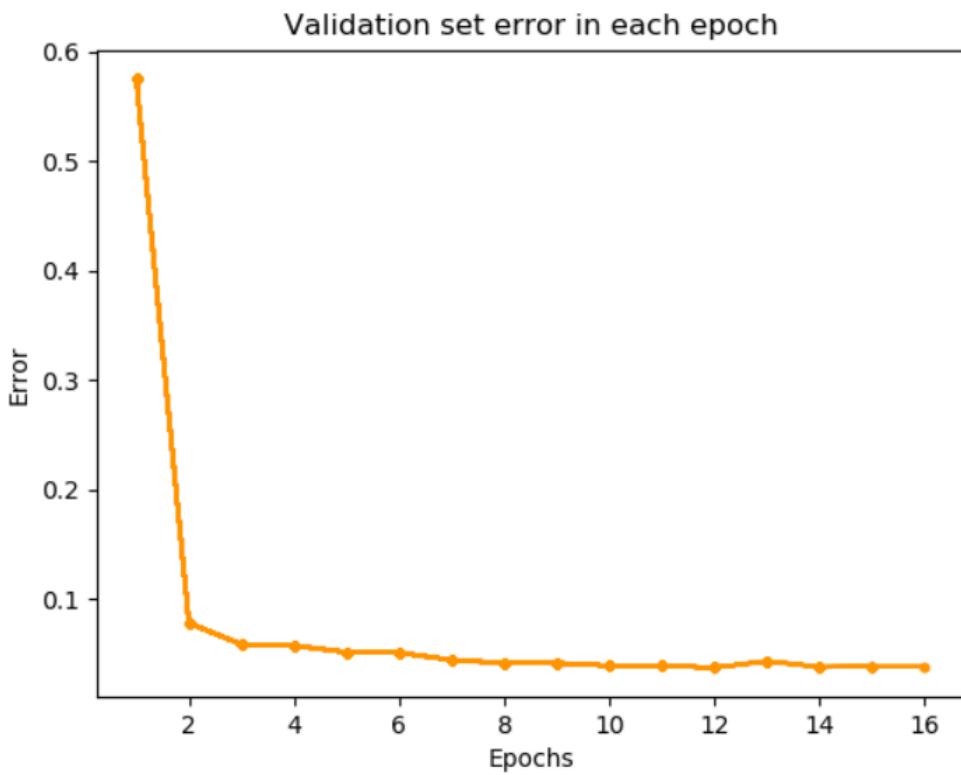
در ایپاک‌های آخر میزان تغییران مجموعه وزن‌ها و بایاس‌ها کم شده است و تقریباً در آن محدوده باقی می‌ماند. زیرا هر چه جلوتر می‌رویم معمولاً گرادیان‌ها کوچک‌تر می‌شوند و تغییرات وزن‌ها نسبت به مراحل قبل کمتر می‌شود.

خطای مجموعه‌ی آموزش در هر ایپاک:



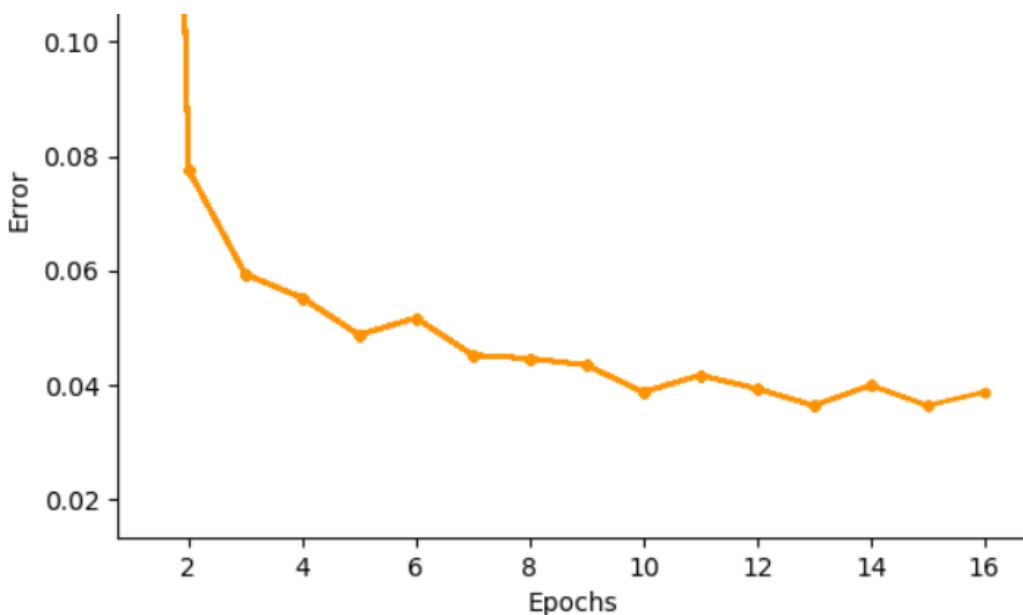
خطای مجموعه آموزش با آموزش بیشتر شبکه کاهش داشته است. بیشتر کاهش در ایپاک اول رخ داده است و در ایپاک های آخر میزان تغییرات خطا کم شده است و به مقدار ثابتی رسیده است.

خطای مجموعه ارزیابی در هر ایپاک:



خطای ارزیابی در هر ایپاک با آموزش بیشتر، کاهش داشته است. در ایپاک‌های اول بیشترین تغییرات را داشته‌ایم و در ایپاک‌های پایانی از میزان تغییرات کاسته شده است.

بخشی از نمودار بالا با بزرگنمایی بیشتر:



این شبکه در فایل MLP-output-2.json ذخیره شده است.

نتیجه‌های شبکه عصبی سوم:

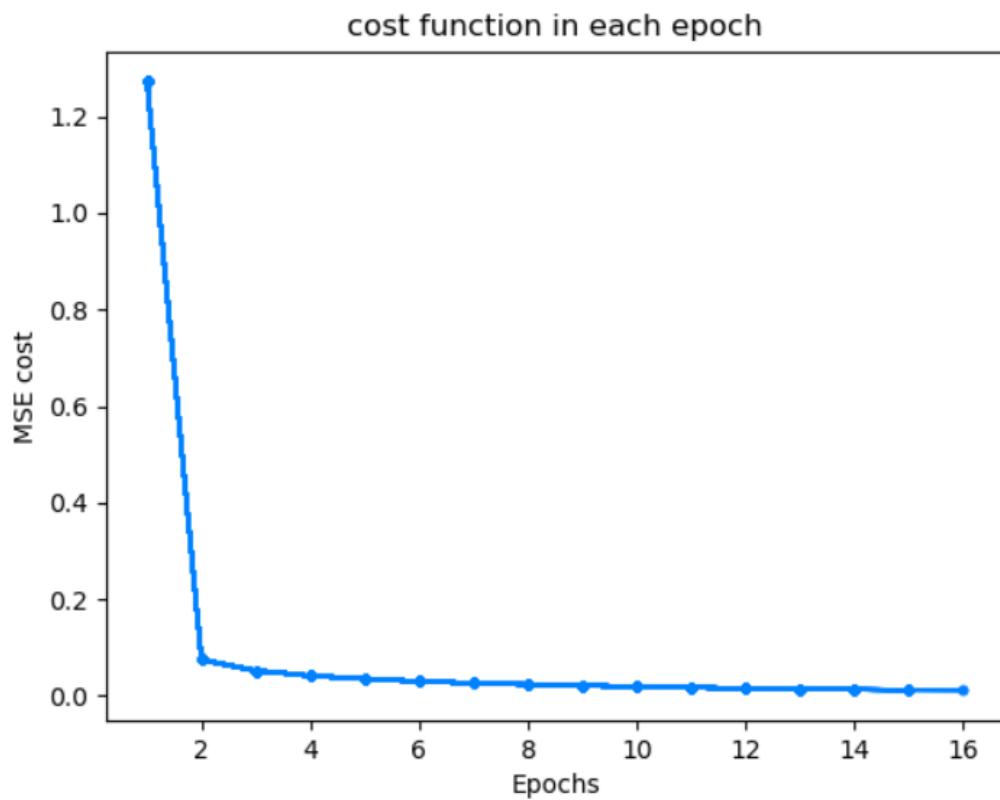
مقدار تابع هزینه در آخرین ایپاک، خطای آموزش، خطای ارزیابی، خطای تست :

```
Iter 14/15, Cost 0.012324640606699943 ...
Iter 15/15, Cost 0.011447621362099555 ...
=====
Train Error: 0.008014705882352941
Validation Error: 0.025294117647058825
Test Error: 0.025294117647058825

5-fold error is: 0.009852941176470589
=====^=====
```

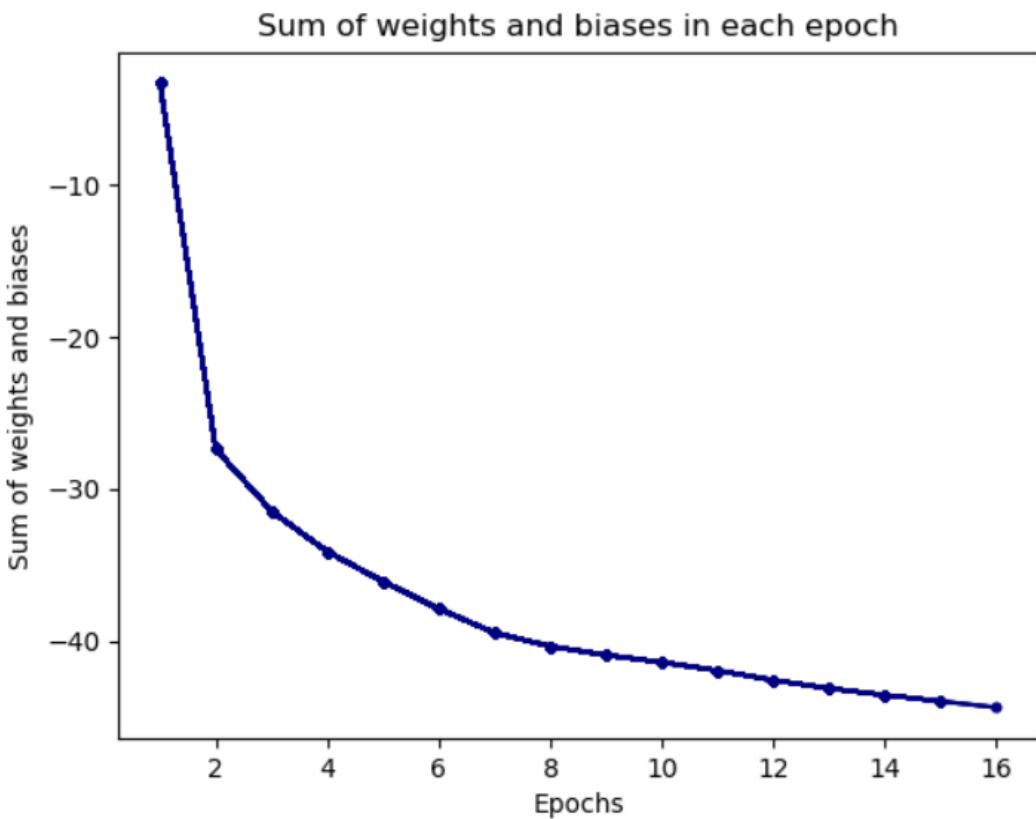
مقدار تابع هزینه بعد از آخرین ایپاک برابر با 0.011 است. خطای مجموعه آموزش برابر با 0.8 درصد است، خطای مجموعه ارزیابی برابر با 2.5 درصد است و خطای مجموعه تست برابر با 2.52 درصد است. خطای k-fold برابر با 0.9 درصد است.

مقدار تابع هزینه در هر ایپاک:



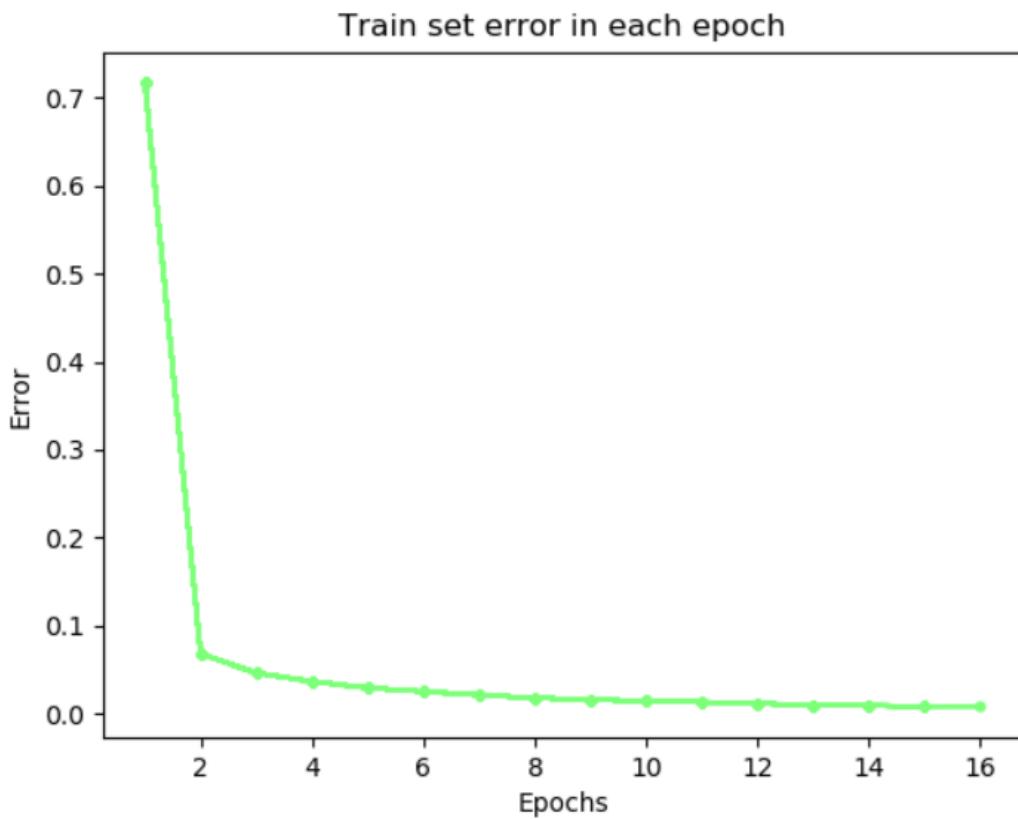
در طی ۱۵ ایپاک تابع هزینه‌ی MSE از ۱,۲ به ۰,۱ رسیده است که میزان کاهش چشم‌گیر است. مقدار تابع هزینه سیر نزولی داشته است و بیشترین تغییرات در زمان انجام ایپاک اول بوده است و هر چه جلوتر رفتهایم در ایپاک‌های بعد از میزان تغییرات کاسته شده است و با روند کند تری تابع هزینه کم شده است.

مجموع وزن‌ها و بایاس‌ها در هر ایپاک:



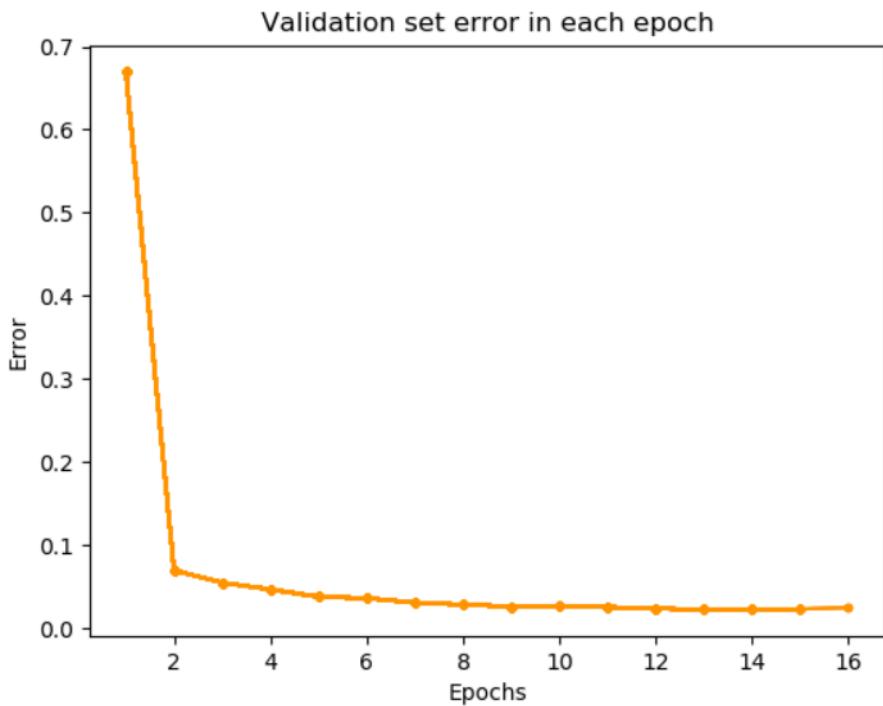
در ایپاک‌های آخر میزان تغییران مجموعه وزن‌ها و بایاس‌ها کم شده است و تقریباً در آن محدوده باقی می‌ماند. زیرا هر چه جلوتر می‌رویم معمولاً گرادیان‌ها کوچک‌تر می‌شوند و تغییرات وزن‌ها نسبت به مراحل قبل کمتر می‌شود.

خطای مجموعه‌ی آموزش در هر ایپاک:



خطای مجموعه آموزش با آموزش بیشتر شبکه کاهش داشته است. بیشتر کاهش در ایپاک اول رخ داده است و در ایپاک های آخر میزان تغییرات خطا کم شده است و به مقدار ثابتی رسیده است.

خطای مجموعه ارزیابی در هر ایپاک:



خطای ارزیابی در هر ایپاک با آموزش بیشتر، کاهش داشته است. در ایپاک‌های اول بیشترین تغییرات را داشته‌ایم و در ایپاک‌های پایانی از میزان تغییرات کاسته شده است.

این شبکه در فایل `MLP-output-3.json` ذخیره شده است.

نتیجه‌های شبکه عصبی چهارم:

مقدار تابع هزینه در آخرین ایپاک، خطای آموزش، خطای ارزیابی، خطای تست :

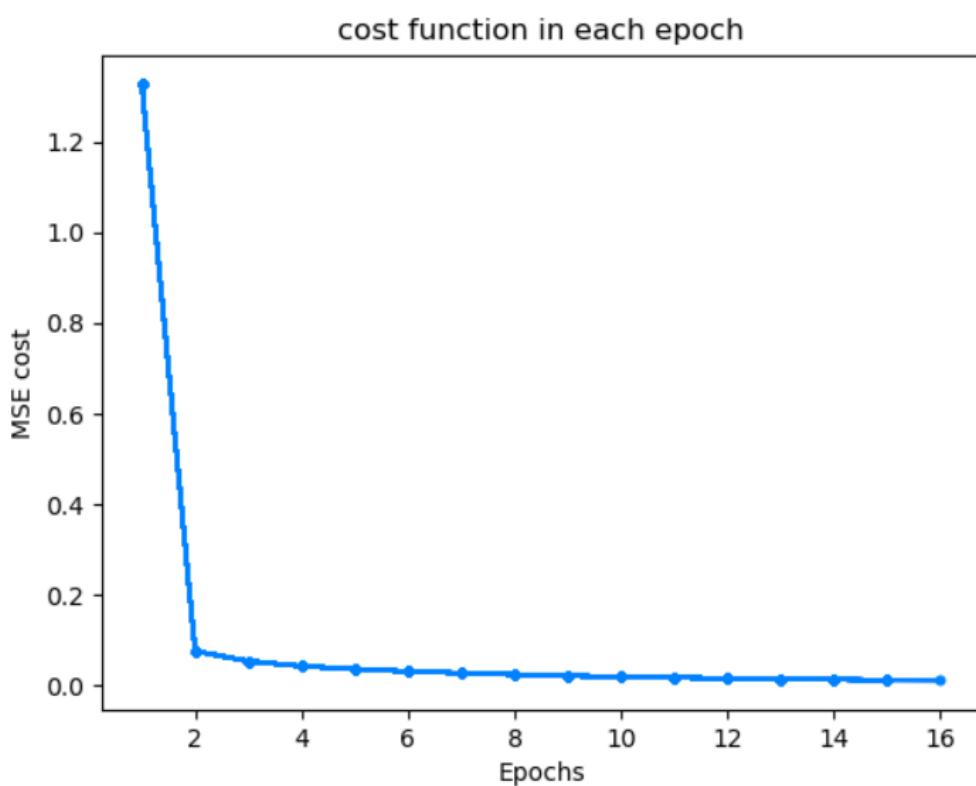
```

Iter 14/15, Cost 0.011291596438623734 ...
Iter 15/15, Cost 0.010370676924920759 ...
=====
Train Error: 0.007794117647058824
Validation Error: 0.01764705882352941
Test Error: 0.02411764705882353
=====

```

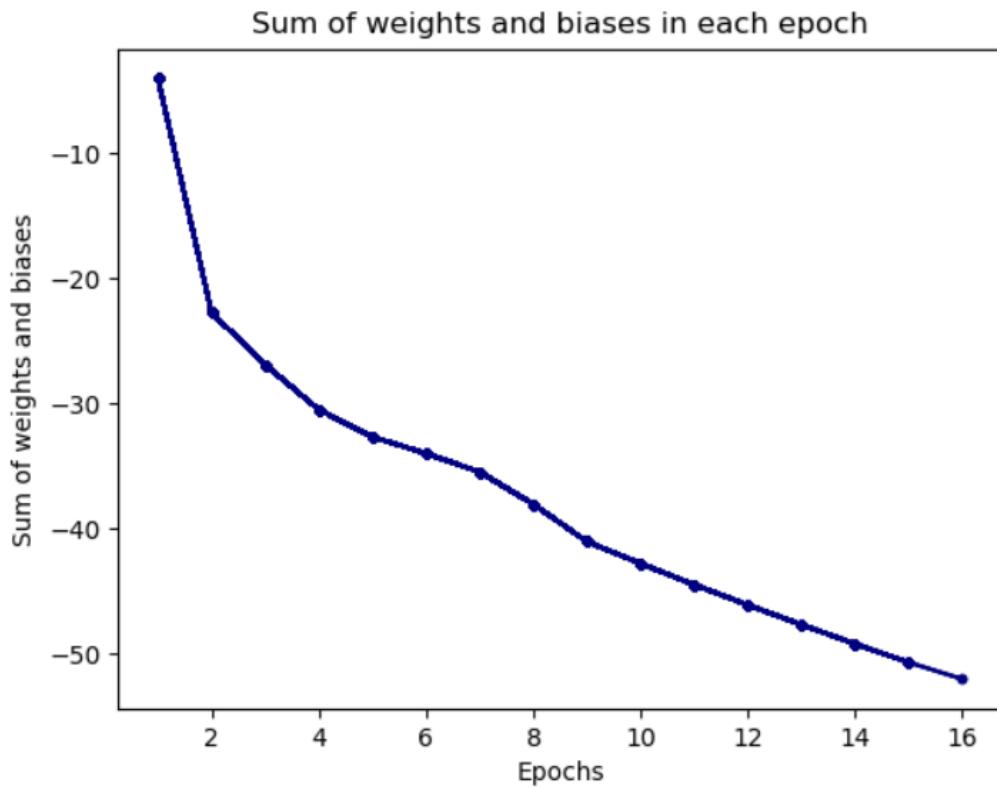
مقدار تابع هزینه بعد از آخرین ایپاک برابر با 0.0103 است. خطای مجموعه آموزش برابر ۰,۷ درصد است، خطای مجموعه ارزیابی برابر با ۱,۷ درصد است و خطای مجموعه تست برابر با ۲,۴ درصد است.

مقدار تابع هزینه در هر ایپاک:



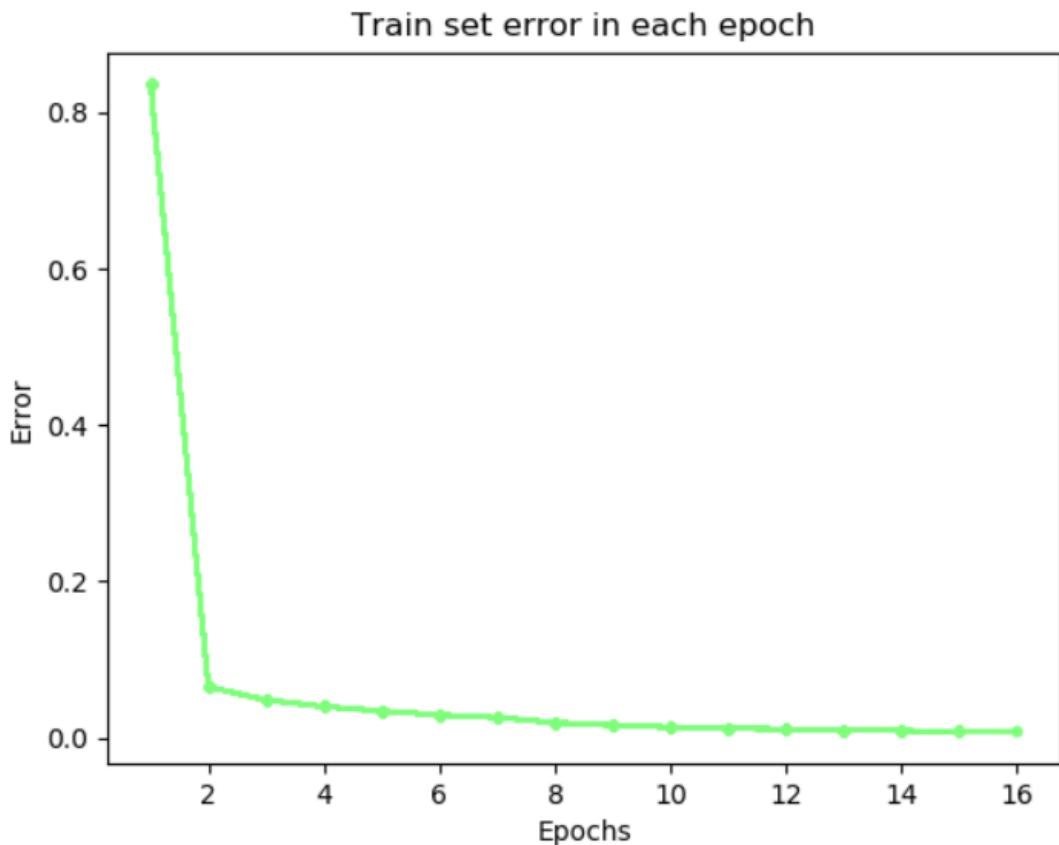
در طی ۱۵ ایپاک تابع هزینه MSE از ۱,۲ به ۰,۰ رسیده است که میزان کاهش چشمگیر است. مقدار تابع هزینه سیر نزولی داشته است و بیشترین تغییرات در زمان انجام ایپاک اول بوده است و هر چه جلوتر رفته‌ایم در ایپاک‌های بعد از میزان تغییرات کاسته شده است و با روند کندتری تابع هزینه کم شده است.

مجموع وزن‌ها و بایاس‌ها در هر ایپاک:



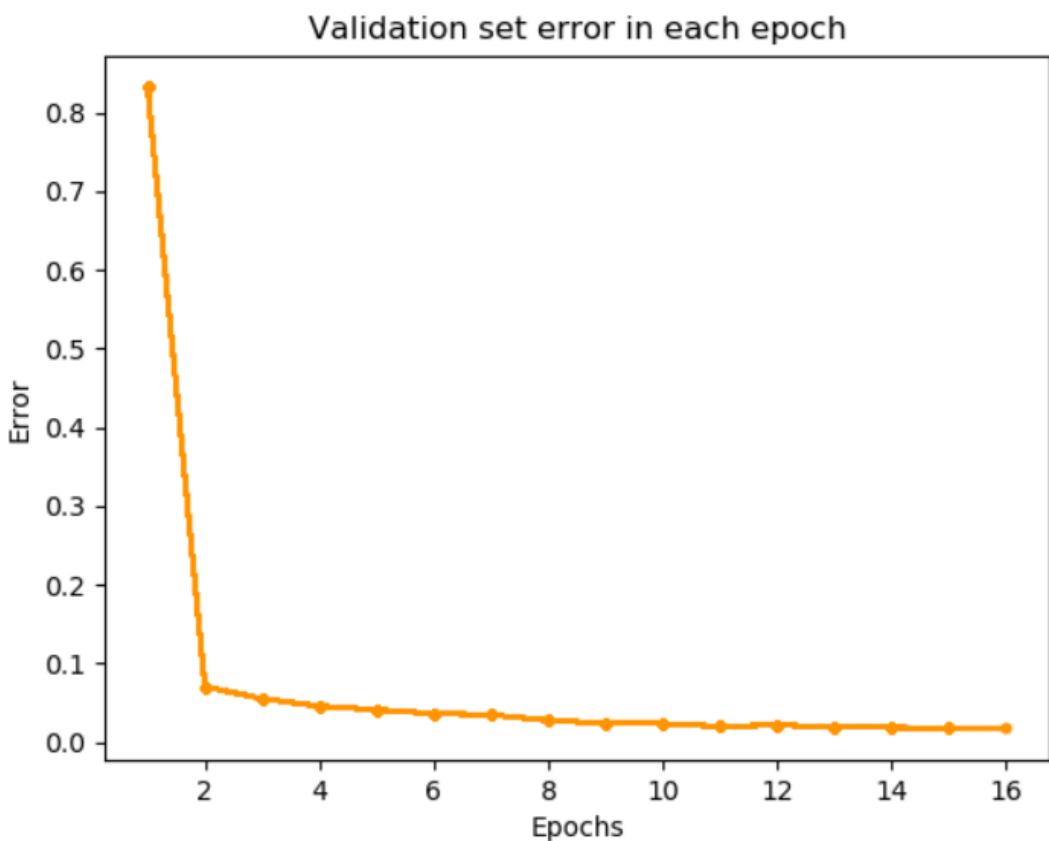
در ایپاک‌های آخر میزان تغییران مجموعه وزن‌ها و بایاس‌ها کم شده است و تقریباً در آن محدوده باقی می‌ماند و تقریباً در چند ایپاک آخر مقدار مجموع ثابت مانده است. زیرا هر چه جلوتر می‌رویم معمولاً گرادیان‌ها کوچک‌تر می‌شوند و تغییرات وزن‌ها نسبت به مراحل قبل کمتر می‌شود.

خطای مجموعه‌ی آموزش در هر ایپاک:



خطای مجموعه آموزش با آموزش بیشتر شبکه کاهش داشته است. بیشتر کاهش در ایپاک اول رخ داده است و در ایپاک های آخر میزان تغییرات خطای کم شده است و به مقدار ثابتی رسیده است.

خطای مجموعه ارزیابی در هر ایپاک:



خطای ارزیابی در هر ایپاک با آموزش بیشتر کاهش داشته است. در ایپاک‌های اول بیشترین تغییرات را داشته‌ایم و در ایپاک‌های پایانی از میزان تغییرات کاسته شده است.

این شبکه در فایل `MLP-output-4.json` ذخیره شده است.

نتیجه‌های شبکه عصبی پنجم:

مقدار تابع هزینه در آخرین ایپاک، خطای آموزش، خطای ارزیابی، خطای تست :

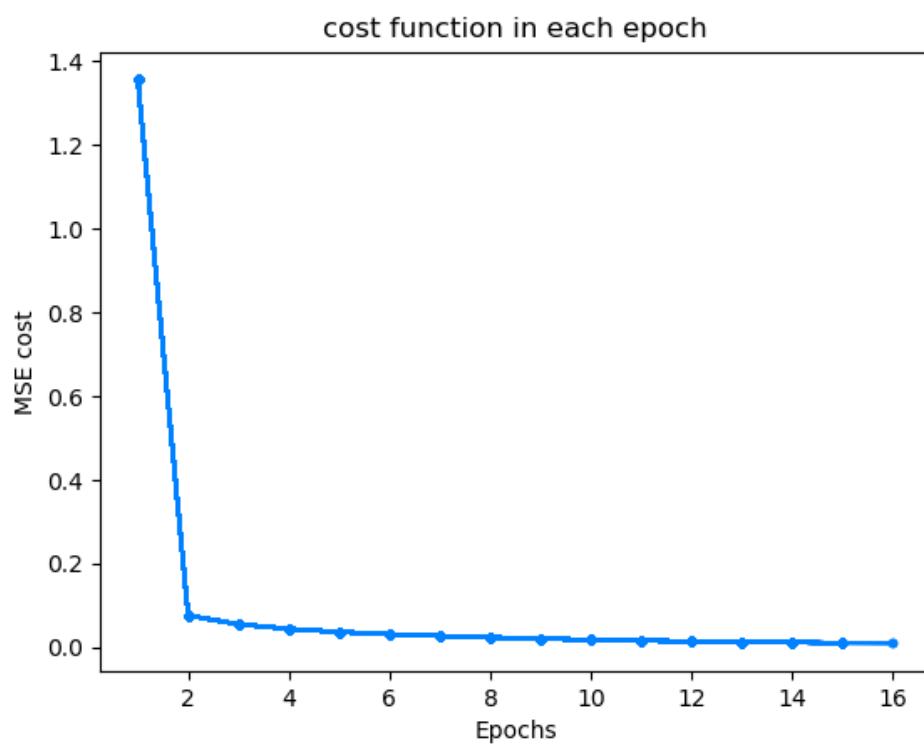
```

run_mlp | best_MLP |
iter 11/15, cost 0.014951117341509272 ...
Iter 12/15, Cost 0.013568935861897367 ...
Iter 13/15, Cost 0.012355079364729292 ...
Iter 14/15, Cost 0.011255917684113785 ...
Iter 15/15, Cost 0.010266776424818018 ...
=====
Train Error: 0.007941176470588234
Validation Error: 0.018235294117647058
Test Error: 0.02
=====
|

```

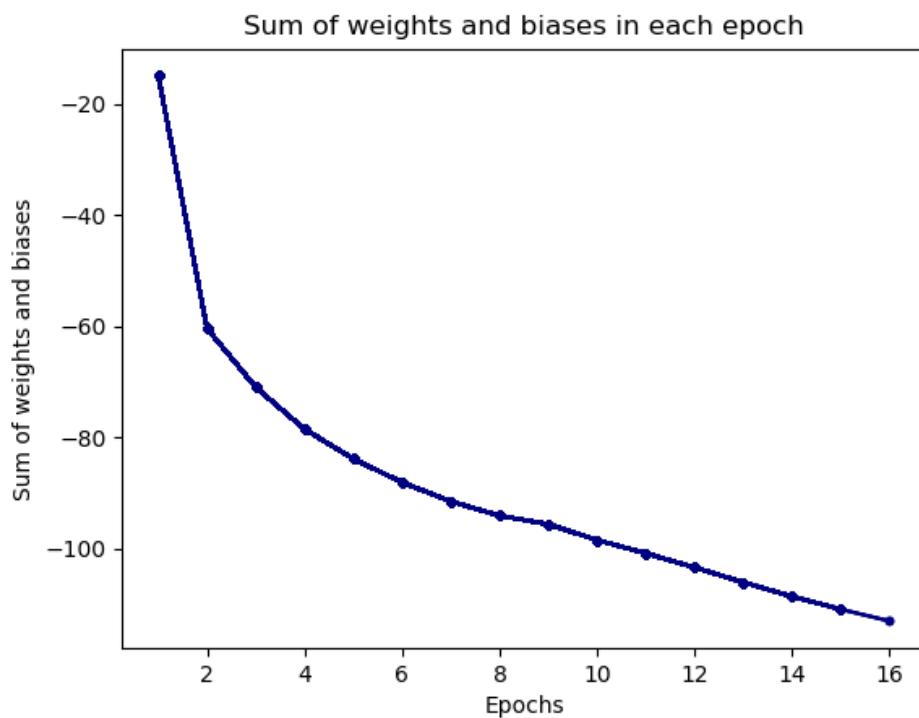
مقدار تابع هزینه بعد از آخرین ایپاک برابر با 0.0102 است. خطای مجموعه آموزش برابر ۰,۷ درصد است، خطای مجموعه ارزیابی برابر با ۱,۸ درصد است و خطای مجموعه تست برابر با ۲ درصد است.

مقدار تابع هزینه در هر ایپاک:



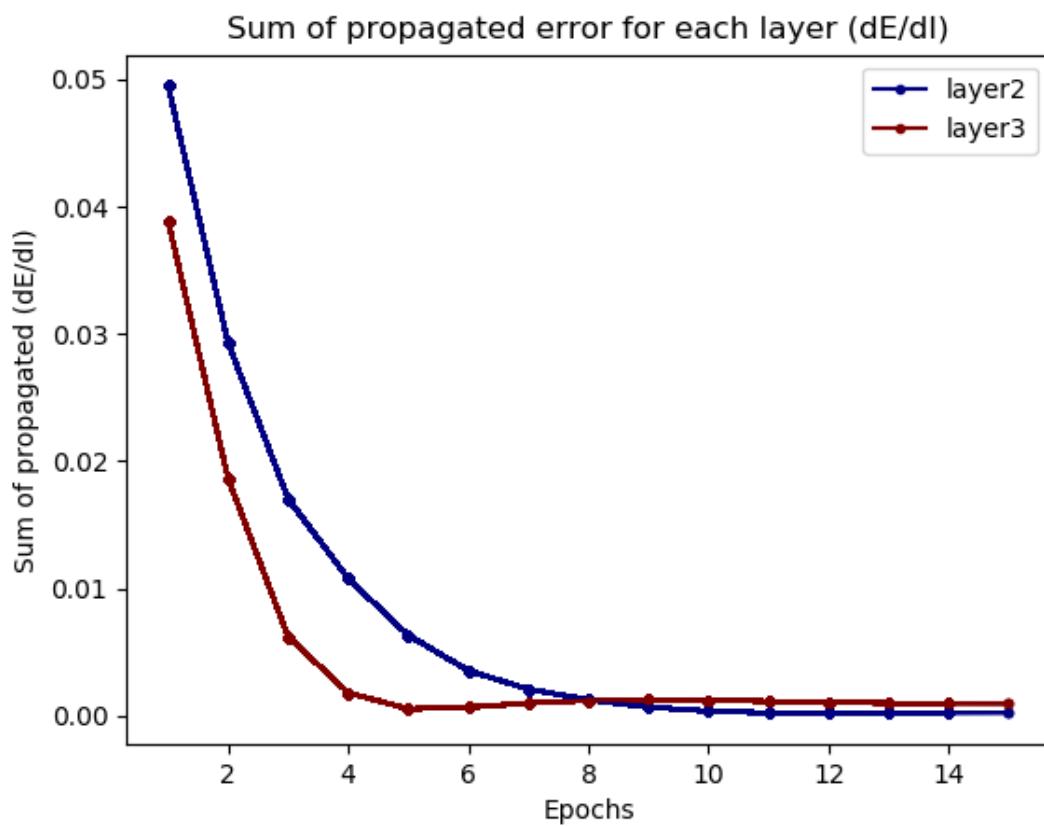
در طی ۱۵ ایپاک تابع هزینه MSE از ۱,۲ به ۰,۰ رسیده است که میزان کاهش چشمگیر است. مقدار تابع هزینه سیر نزولی داشته است و بیشترین تغییرات در زمان انجام ایپاک اول بوده است و هر چه جلوتر رفتهایم در ایپاک‌های بعد از میزان تغییرات کاسته شده است و با روند کندتری تابع هزینه کم شده است.

مجموع وزن‌ها و بایاس‌ها در هر ایپاک:



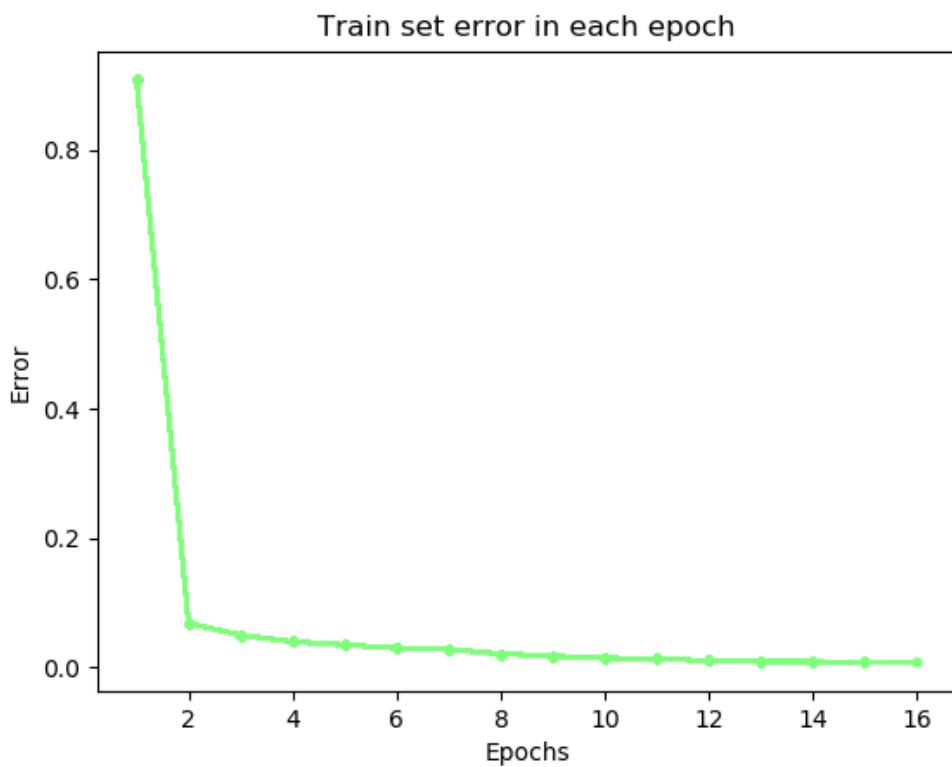
در ایپاک‌های آخر میزان تغییران مجموعه وزن‌ها و بایاس‌ها کم شده است و تقریباً در آن محدوده باقی می‌ماند و تقریباً در چند ایپاک آخر مقدار مجموع ثابت مانده است. زیرا هر چه جلوتر می‌رویم معمولاً گرادیان‌ها کوچک‌تر می‌شوند و تغییرات وزن‌ها نسبت به مراحل قبل کمتر می‌شود.

مجموع خطاهای منتشر شده در لایه‌ی سوم و دوم:



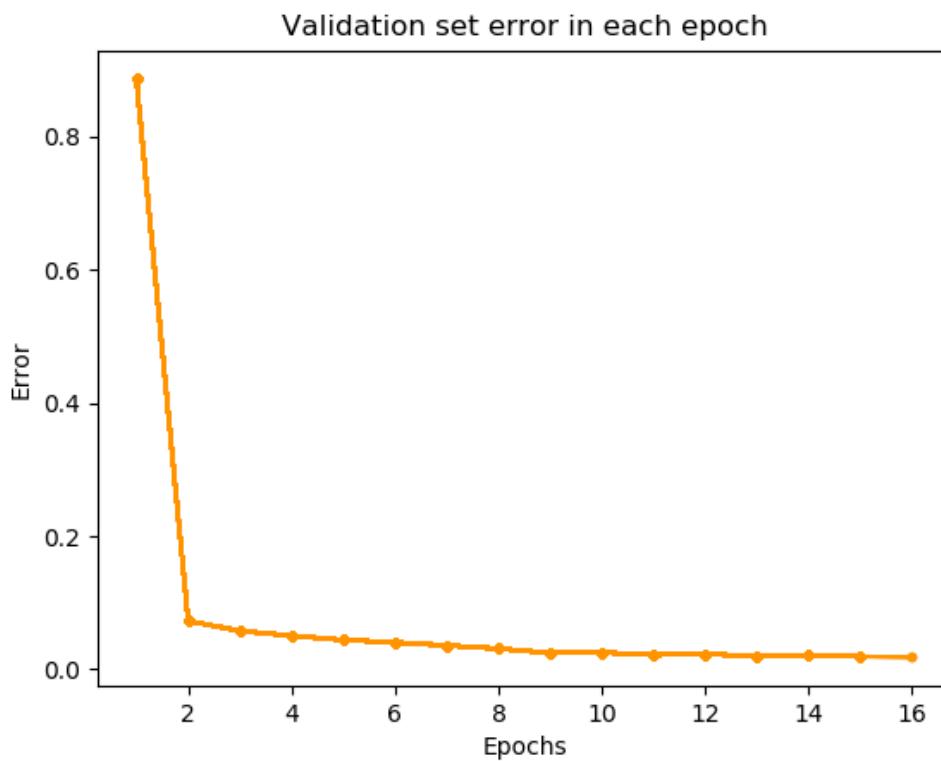
همین طور که مشاهده می شود مجموع خطاهای هر لایه بعد از مدتی در ثابت می شود زیرا هر چه به ایپاک های جلوتری می رویم گرادیان ها کمتر و کمتر می شود.

خطای مجموعه‌ی آموزش در هر ایپاک:



خطای مجموعه آموزش با آموزش بیشتر شبکه کاهش داشته است. بیشتر کاهش در ایپاک اول رخ داده است و در ایپاک های آخر میزان تغییرات خطا کم شده است و به مقدار ثابتی رسیده است.

خطای مجموعه ارزیابی در هر ایپاک:



خطای ارزیابی در هر ایپاک با آموزش بیشتر کاهش داشته است. در ایپاک‌های اول بیشترین تغییرات را داشته‌ایم و در ایپاک‌های پایانی از میزان تغییرات کاسته شده است.

این شبکه در فایل `MLP-output-5.json` ذخیره شده است.

نتیجه‌های شبکه عصبی ششم:

مقدار تابع هزینه در آخرین ایپاک، خطای آموزش، خطای ارزیابی، خطای تست :

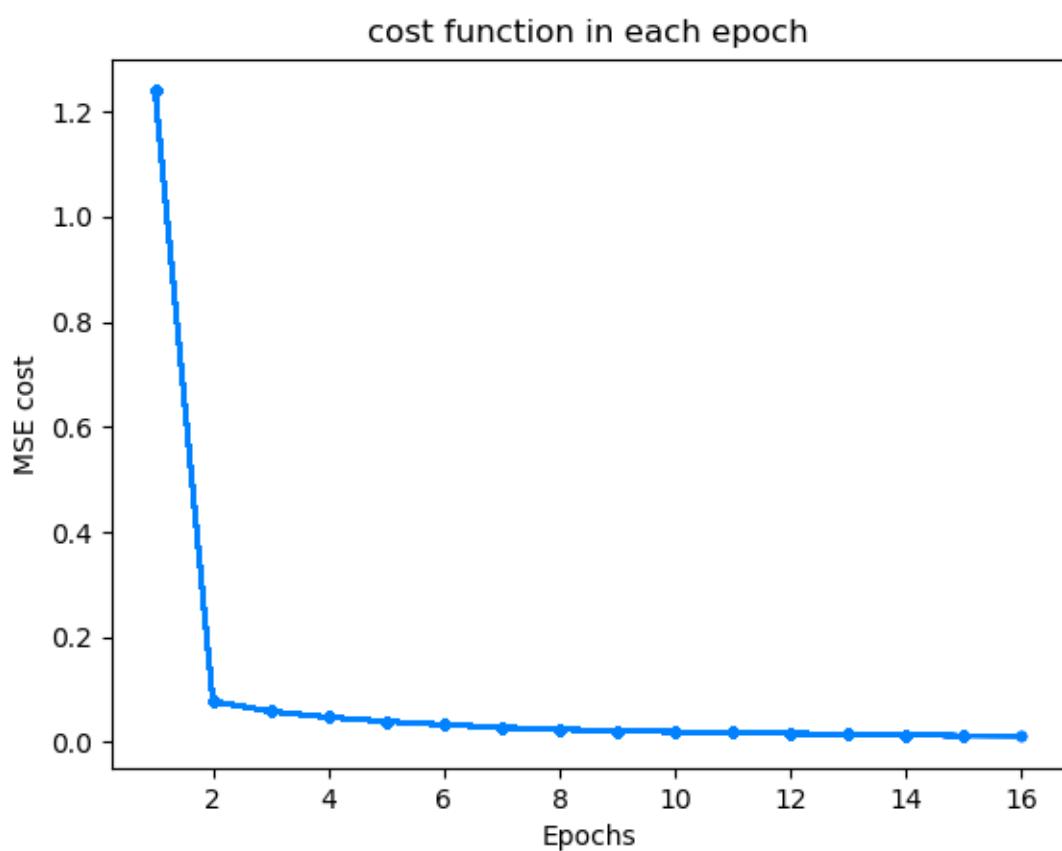
```

    ۱۲۸۱   ۱۷/۱۵   ۰.۳۱۰۵۴۸۱۴۵۲۴۵۲۵/۰۹   ...
Iter 11/15, Cost 0.01679448875680463 ...
Iter 12/15, Cost 0.015178043357875714 ...
Iter 13/15, Cost 0.013728808127540676 ...
Iter 14/15, Cost 0.012441106958105295 ...
Iter 15/15, Cost 0.01131018507241259 ...
=====
Train Error: 0.008602941176470588
Validation Error: 0.01764705882352941
Test Error: 0.020588235294117647
=====

```

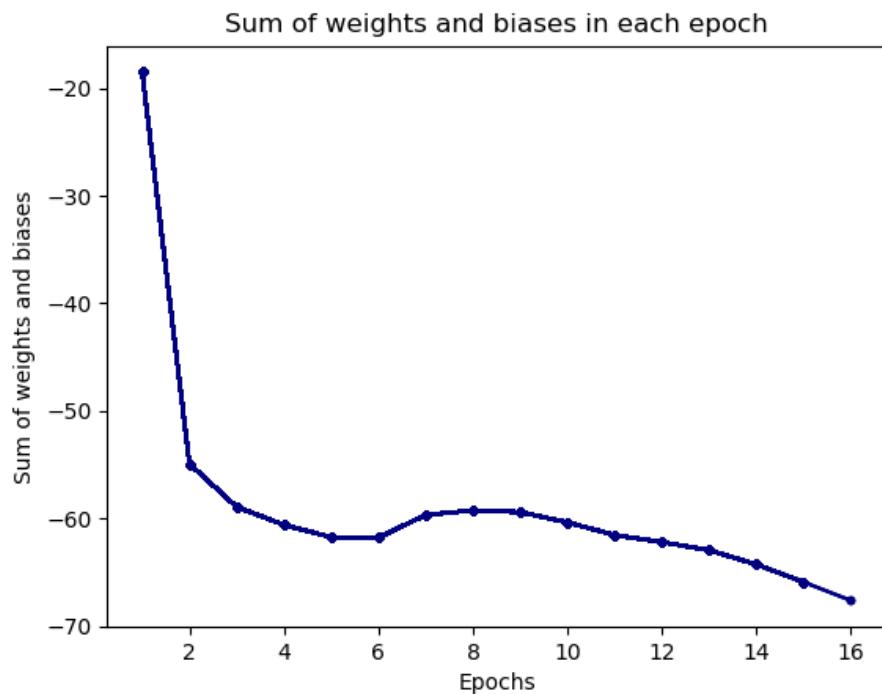
مقدار تابع هزینه بعد از آخرین ایپاک برابر با 0.011 است. خطای مجموعه آموزش برابر ۰.۸ درصد است، خطای مجموعه ارزیابی برابر با ۱.۷ درصد است و خطای مجموعه تست برابر با ۲.۰۵ درصد است.

مقدار تابع هزینه در هر ایپاک:



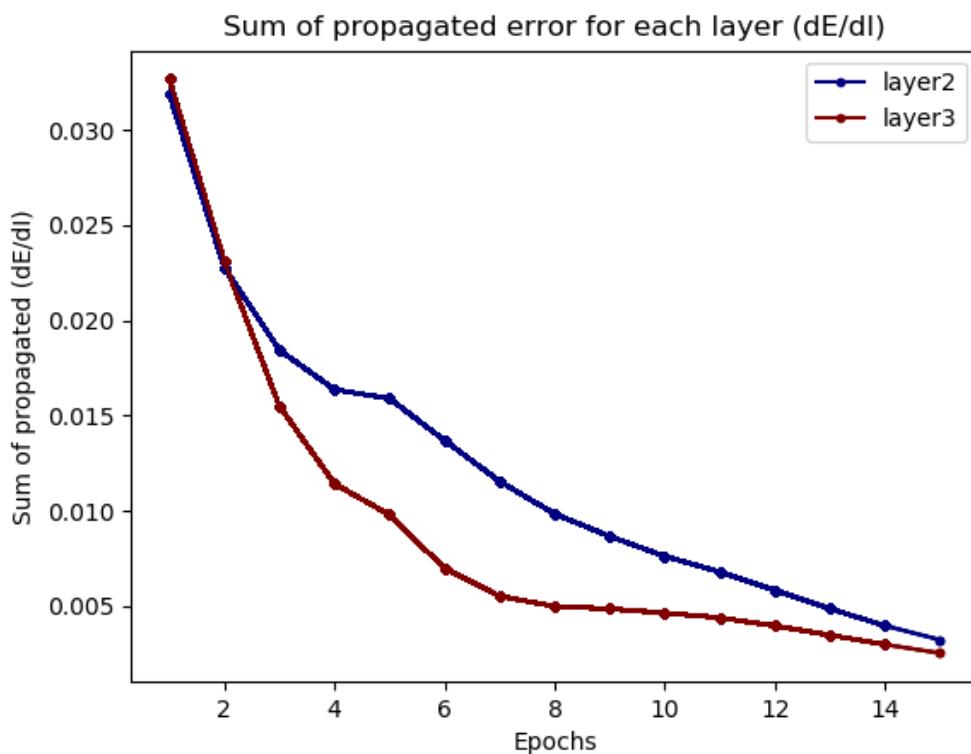
در طی ۱۵ ایپاک تابع هزینه MSE از ۱۱,۰ به ۱,۲ رسیده است که میزان کاهش چشمگیر است. مقدار تابع هزینه سیر نزولی داشته است و بیشترین تغییرات در زمان انجام ایپاک اول بوده است و هر چه جلوتر رفتهایم در ایپاک‌های بعد از میزان تغییرات کاسته شده است و با روند کندتری تابع هزینه کم شده است.

مجموع وزن‌ها و بایاس‌ها در هر ایپاک:



در ایپاک‌های آخر میزان تغییران مجموعه وزن‌ها و بایاس‌ها کم شده‌است و تقریباً در آن محدوده باقی می‌ماند و تقریباً در چند ایپاک آخر مقدار مجموع ثابت مانده است. زیرا هر چه جلوتر می‌رویم معمولاً گرادیان‌ها کوچک‌تر می‌شوند و تغییرات وزن‌ها نسبت به مراحل قبل کمتر می‌شود.

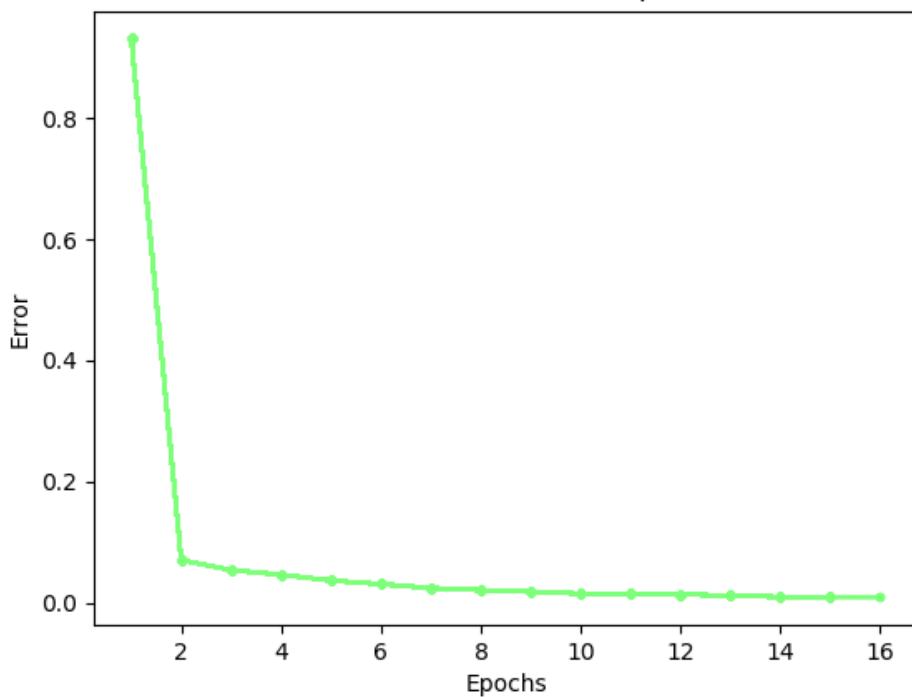
مجموع خطاهای منتشر شده در لایه‌ی سوم و دوم:



همین طور که مشاهده می شود مجموع خطاهای هر لایه بعد از مدتی در ثابت می شود زیرا هر چه به ایپاک های جلوتری می رویم گرادیان ها کمتر و کمتر می شود و به صفر نزدیک تر می شود.

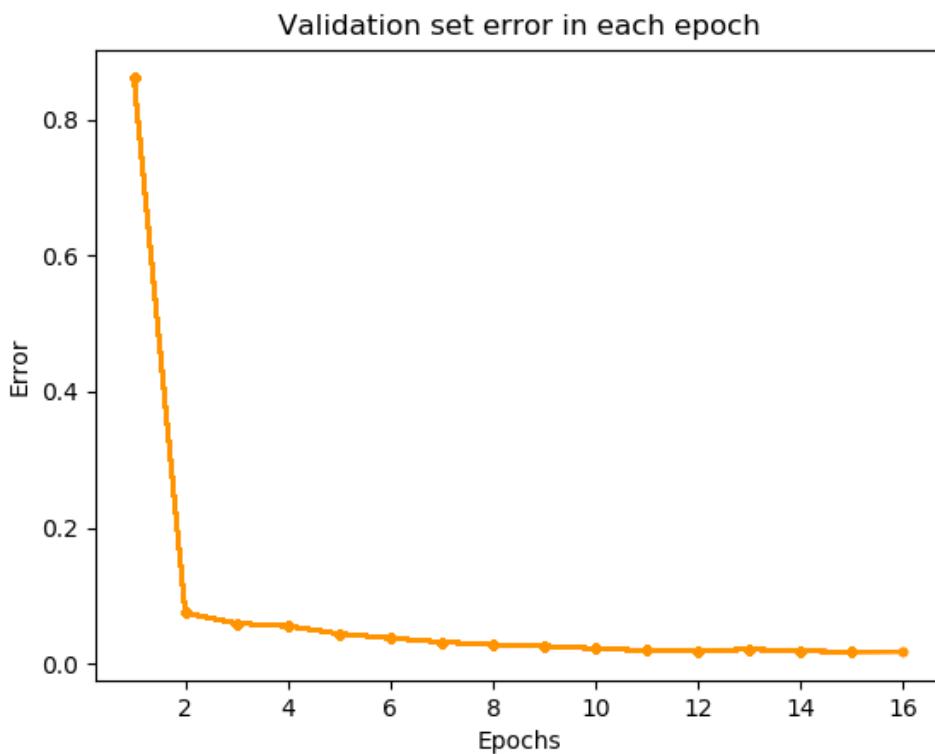
خطای مجموعه‌ی آموزش در هر ایپاک:

Train set error in each epoch



خطای مجموعه آموزش با آموزش بیشتر شبکه کاهش داشته است. بیشتر کاهش در ایپاک اول رخ داده است و در ایپاک های آخر میزان تغییرات خطأ کم شده است و به مقدار ثابتی رسیده است.

خطای مجموعه ارزیابی در هر ایپاک:



خطای ارزیابی در هر ایپاک با آموزش بیشتر کاهش داشته است. در ایپاک‌های اول بیشترین تغییرات را داشته‌ایم و در ایپاک‌های پایانی از میزان تغییرات کاسته شده است.

این شبکه در فایل `MLP-output-6.json` ذخیره شده است.

نتیجه گیری:

برای دیدن تغییری که تعداد نرون‌ها در یک شبکه سه لایه ایجاد می‌کنند ۷ شبکه عصبی یکسان با پارامترهای یادگیری و ساختار یکسان ایجاد کردم که فقط در تعداد نرون‌های لایه‌ی نهان با یک دیگر تفاوت دارند. در زیر خطا مجموعه‌ی تست را مشاهده می‌کنید:

تعداد نرون در لایه نهان	خطای مجموعه تست (درصد)
۵	% ۱۸,۸۲
۱۰	% ۷,۲۹
۳۰	% ۲,۹۴
۶۴	% ۲,۵۲
۱۴۰	% ۲,۴۱
۲۵۶	% ۲
۵۱۲	% ۲,۰۵

در شبکه‌ی صفرم که تعداد نرون‌ها در لایه‌ی نهان ۵ عدد است و میزان خطا بر روی مجموعه‌ی تست ۱۸,۸۲ درصد است. شبکه قدرت کافی برای یادگیری را ندارد و شبکه bias است با افزایش تعداد نرون‌های لایه‌ی نهان در شبکه‌ی یکم به ۱۰ عدد، خطای مجموعه‌ی تست کمتر شده است و به ۷,۲۹ درصد رسیده است. با زیاد کردن تعداد نرون‌ها در شبکه‌های دوم، سوم، چهارم، پنجم و ششم به خطای ۲,۹۴ درصد، ۲,۲۵ درصد، ۲,۴۱ درصد، ۲ درصد و ۲,۰۵ درصد رسیده ایم.

می‌توان نتیجه گرفت در صورتی که تعداد نرون‌های لایه‌ی نهان کم باشد شبکه قادر به یادگیری مناسب نیست و با زیاد کردن تدریجی نرون‌ها می‌توان به عملکرد خوبی رسید و در هر مرحله خطرا کاهش داد. **ولی** بعد از زیاد کردن نرون‌ها تا حد مشخصی دیگر زیاد کردن نرون‌ها، منجر به تغییر محسوس و قابل لمسی در شبکه نمی‌شود و فقط باعث زیاد شدن زمان آموزش و تست می‌شود و حتی ممکن است باعث overfit و کاهش دقت هم شود.

به همین دلیل بهتر است از تعداد کمی نرون شروع به ساختن لایه‌ی نهان کنیم و با افزایش آن خطرا کاهش دهیم و زمانی که افزایش نرون‌ها باعث بهبود قابل لمسی نشد از زیاد کردن نرون‌ها خوداری کنیم.

بررسی تاثیر تعداد لایه‌های نهان بر روی عملکرد شبکه

عنوان:

سپس با ایجاد شبکه عصبی چند لایه با تعداد لایه‌های مختلف مخفی مختلف برای تعداد لایه‌های مخفی)، تأثیر عمق شبکه (تعداد لایه‌های شبکه) را در دقت بررسی کنید. این دسته بندی را ارزیابی کنید. دقت کنید که در این حالت تعداد نورون هر لایه مخفی به مراتب کمتر از حالت قبل و محدود است. ۲ در این بخش مجموع خطای پس انتشار یافته از هر لایه در طول فرایند آموزش در یک نمودار نمایش داده شده و نتیجه بر اساس عمق شبکه خود تجزیه و تحلیل کنید.

شرایط آزمایش:

برای بررسی تاثیر تعداد لایه‌ها بر دقت شبکه‌ی عصبی، چهار شبکه با تعداد لایه‌های نهان مختلف می‌سازم، که هر لایه نهان ۲۰ نرون را شامل می‌شود. پارامترهای مناسب هر کدام از شبکه‌های عصبی را با استفاده از کد **MLP_best.py** که شبکه را با پارامترهای مختلف تست می‌کند، پیدا می‌کنم.

برای پیدا کردن شبکه‌ی های مناسب برای پاسخ به این قسمت از سوال، شبکه‌های مختلف را آموزش داده‌ام و برای هر کدام پارامترهای مناسب آموزش را پیدا کرده‌ام:

result_momentum_MSE_layers [1024, 5, 5, 5, 10].digits.dat	4/3/2018 3:28 AM	KMP - MPEG Movi...	7 KB
result_momentum_MSE_layers [1024, 5, 5, 10].digits.dat	4/3/2018 2:48 AM	KMP - MPEG Movi...	7 KB
result_momentum_MSE_layers [1024, 5, 10].digits.dat	4/3/2018 2:09 AM	KMP - MPEG Movi...	7 KB
result_momentum_MSE_layers [1024, 10, 10, 10, 10, 10, 10].digits.dat	4/2/2018 7:51 AM	KMP - MPEG Movi...	6 KB
result_momentum_MSE_layers [1024, 10, 10, 10, 10].digits.dat	4/2/2018 5:38 AM	KMP - MPEG Movi...	8 KB
result_momentum_MSE_layers [1024, 12, 10].digits.dat	4/2/2018 1:23 PM	KMP - MPEG Movi...	7 KB
result_momentum_MSE_layers [1024, 12, 12, 12, 12, 12, 12, 10].digits.dat	4/2/2018 9:48 AM	KMP - MPEG Movi...	6 KB
result_momentum_MSE_layers [1024, 12, 12, 12, 12, 12, 12, 12, 12, 10].digits.dat	4/2/2018 11:42 AM	KMP - MPEG Movi...	6 KB
result_momentum_MSE_layers [1024, 20, 10].digits.dat	4/3/2018 2:11 AM	KMP - MPEG Movi...	6 KB
result_momentum_MSE_layers [1024, 20, 15, 10].digits.dat	4/3/2018 2:31 PM	KMP - MPEG Movi...	8 KB
result_momentum_MSE_layers [1024, 20, 20, 10].digits.dat	4/3/2018 5:41 AM	KMP - MPEG Movi...	7 KB
result_momentum_MSE_layers [1024, 20, 20, 20, 10].digits.dat	4/3/2018 4:53 AM	KMP - MPEG Movi...	7 KB
result_momentum_MSE_layers [1024, 20, 20, 20, 20, 20, 10].digits.dat	4/3/2018 7:18 AM	KMP - MPEG Movi...	7 KB
result_momentum_MSE_layers [1024, 20, 20, 20, 20, 20, 20, 10].digits.dat	4/3/2018 9:30 AM	KMP - MPEG Movi...	7 KB
result_momentum_MSE_layers [1024, 20, 20, 20, 20, 20, 20, 20, 20, 20, 10].digits.dat	4/3/2018 12:37 PM	KMP - MPEG Movi...	7 KB
result_momentum_MSE_layers [1024, 30, 20, 15, 10].digits.dat	4/4/2018 2:58 AM	KMP - MPEG Movi...	7 KB
result_momentum_MSE_layers [1024, 40, 10].digits.dat	4/6/2018 8:06 AM	KMP - MPEG Movi...	8 KB
result_momentum_MSE_layers [1024, 40, 30, 20, 15, 10].digits.dat	4/6/2018 3:12 AM	KMP - MPEG Movi...	7 KB
result_momentum_MSE_layers [1024, 40, 40, 10].digits.dat	4/6/2018 6:25 AM	KMP - MPEG Movi...	7 KB
result_momentum_MSE_layers [1024, 40, 40, 40, 40, 10].digits.dat	4/6/2018 4:55 AM	KMP - MPEG Movi...	7 KB
result_momentum_MSE_layers [1024, 80, 10].digits.dat	4/1/2018 2:45 AM	KMP - MPEG Movi...	6 KB

از بین شبکه‌های مختلف، شبکه‌های چند لایه با ۲۰ نرون در هر لایه نهان را انتخاب می‌کنم.

در زیر فایل‌های ورودی به برنامه برای ایجاد چهار شبکه مختلف را مشاهده می‌کنید:

فایل ورودی شبکه‌ی اول - :input-10.json

```
input-10.json
1  {
2      "digits_letters_digits": "digits",
3      "ratio_of_train_set":0.80,
4      "ratio_of_valid_set":0.10,
5      "ratio_of_test_set":0.10,
6      "learning_rate":0.01,
7      "max_epochs":30,
8      "cut_cost": -1,
9      "random_state":2,
10     "K_fold": 5,
11     "neurons_layes": [1024, 20, 10],
12     "type_layers": ["tanh", "sigmoid"],
13     "type_of_cost": "MSE",
14     "mode": "momentum"
15 }
```

شبکه‌ی شماره اول: از دیتاست عده‌ها استفاده می‌شود، ۸۰ درصد دیتاست برای آموزش است، ۱۰ درصد دیتاست برای ارزیابی است، ۱۰ درصد دیتاست برای تست است، ضریب یادگیری برابر با ۰.۰۱ است، حداکثر تعداد ایپاک‌ها ۳۰ است، شرط کمتر شدنتابع هزینه از یک مقداری برای توقف آموزش در نظر گرفته نمی‌شود، تولید عده‌های تصادفی با عدد ۲ بذرپاشی می‌شود، تعداد فیچرها ۱۰۲۴ تا است، لایه نهان اول ۲۰ نرون دارد. لایه خروجی ۱۰ نرون دارد. تابع فعال سازی لایه نهان اول **tanh** است و تابع فعال سازی لایه خروجی **sigmoid** است، تابع هزینه **MSE** است، از روش **momentum** استفاده شده است. از **5-fold cross validation** استفاده می‌شود.

فایل ورودی شبکه‌ی دوم - :input-11.json

```

input-11.json
1  {
2      "digits_letters_digits": "digits",
3      "ratio_of_train_set":0.80,
4      "ratio_of_valid_set":0.10,
5      "ratio_of_test_set":0.10,
6      "learning_rate":0.01,
7      "max_epochs":80,
8      "cut_cost": -1,
9      "random_state":2,
10     "K_fold": 5,
11     "neurons_layes": [1024, 20, 20, 20, 10],
12     "type_layers": ["tanh", "tanh", "tanh", "sigmoid"],
13     "type_of_cost": "MSE",
14     "mode": "momentum"
15 }

```

شبکه‌ی شماره دوم: از دیتاست عده‌ها استفاده می‌شود، ۸۰ درصد دیتاست برای آموزش است، ۱۰ درصد دیتاست برای ارزیابی است، ۱۰ درصد دیتاست برای تست است، ضریب یادگیری برابر با ۰.۰۱ است، حداکثر تعداد ایپاک‌ها ۸۰ است، شرط کمتر شدن تابع هزینه از یک مقداری برای توقف آموزش در نظر گرفته نمی‌شود، تولید عده‌های تصادفی با عدد ۲ بذرپاشی می‌شود، تعداد فیچرها ۱۰۲۴ تا است، سه لایه‌ی نهان داریم، لایه نهان اول، دوم و سوم هر کدام ۲۰ نرون دارد. لایه خروجی ۱۰ نرون دارد. تابع فعال سازی لایه‌ی نهان‌های نهان tanh است و تابع فعال سازی لایه‌ی خروجی sigmoid است، تابع هزینه MSE است، از روش momentum استفاده شده است. از ۵-fold cross validation استفاده می‌شود.

فایل ورودی شبکه‌ی سوم – input-12.json

input-12.json

```
1  {
2      "digits_letters_digits": "digits",
3      "ratio_of_train_set": 0.80,
4      "ratio_of_valid_set": 0.10,
5      "ratio_of_test_set": 0.10,
6      "learning_rate": 0.006,
7      "max_epochs": 80,
8      "cut_cost": -1,
9      "random_state": 2,
10     "K_fold": 5,
11     "neurons_layers": [1024, 20, 20, 20, 20, 20, 20, 10],
12     "type_layers": ["tanh", "tanh", "tanh", "tanh", "tanh", "tanh", "sigmoid"],
13     "type_of_cost": "MSE",
14     "mode": "momentum"
15 }
```

شبکه‌ی شماره سوم: از دیتاست عده‌ها استفاده می‌شود، ۸۰ درصد دیتاست برای آموزش است، ۱۰ درصد دیتاست برای ارزیابی است، ۱۰ درصد دیتاست برای تست است، ضریب یادگیری برابر با 0.006 است، حداکثر تعداد ایپاک‌ها ۸۰ است، شرط کمتر شدن تابع هزینه از یک مقداری برای توقف آموزش در نظر گرفته نمی‌شود، تولید عده‌های تصادفی با عدد ۲ بذرپاشی می‌شود، تعداد فیچرها ۱۰۲۴ تا است، شش لایه‌ی نهان داریم، لایه‌های نهان ۲۰ نرون دارند. لایه خروجی ۱۰ نرون دارد. تابع فعال سازی لایه‌ی نهان tanh است و تابع فعال سازی sigmoid است، تابع هزینه MSE است، از روش momentum استفاده شده است. از ۵-fold cross validation استفاده می‌شود.

فایل ورودی شبکه‌ی چهارم - input-13.json

input-13.json

```
1  {
2      "digits_letters_digits": "digits",
3      "ratio_of_train_set":0.80,
4      "ratio_of_valid_set":0.10,
5      "ratio_of_test_set":0.10,
6      "learning_rate":0.006,
7      "max_epochs":80,
8      "cut_cost": -1,
9      "random_state":2,
10     "K_fold": 5,
11     "neurons_layes": [1024, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 10],
12     "type_layers": ["tanh", "tanh", "tanh", "tanh", "tanh",
13                     "tanh", "tanh", "tanh", "tanh", "tanh", "tanh", "tanh", "sigmoid"],
14     "type_of_cost": "MSE",
15     "mode": "momentum"
16 }
```

شبکه‌ی شماره چهارم: از دیتاست عده‌ها استفاده می‌شود، ۸۰ درصد دیتاست برای آموزش است، ۱۰ درصد دیتاست برای ارزیابی است، ۱۰ درصد دیتاست برای تست است، ضریب یادگیری برابر با 0.006 است، حداکثر تعداد ایپاک‌ها ۸۰ است، شرط کمتر شدن تابع هزینه از یک مقداری برای توقف آموزش در نظر گرفته نمی‌شود، تولید عده‌های تصادفی با عدد ۲ بذرپاشی می‌شود، تعداد فیچرها ۱۰۲۴ تا است، دوازده لایه‌ی نهان داریم، لایه‌های نهان هر کدام ۲۰ نرون دارد. لایه خروجی ۱۰ نرون دارد. تابع فعال سازی لایه‌های نهان tanh است و تابع فعال سازی لایه‌ی خروجی sigmoid است، تابع هزینه MSE است، از روش momentum استفاده شده است. از ۵-fold cross validation استفاده می‌شود.

نتیجه‌ی انجام آزمایش:

نتیجه‌های شبکه عصبی اول:

مقدار تابع هزینه در آخرین ایپاک، خطای آموزش، خطای ارزیابی، خطای تست :

```

Iter 29/30, Cost 0.01/0.000000000000000 ...
Iter 30/30, Cost 0.016700069683452518 ...
=====
Train Error: 0.012867647058823529
Validation Error: 0.04
Test Error: 0.04647058823529412

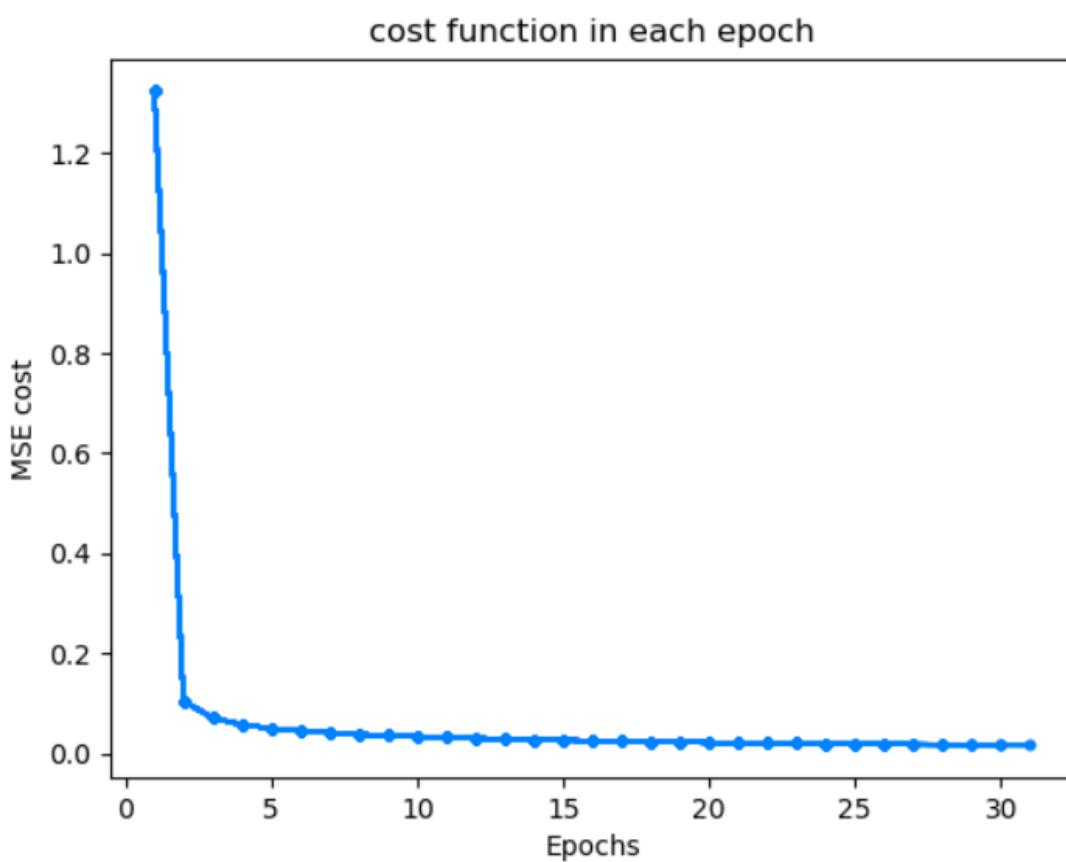
5-fold error is: 0.021985294117647058

=====

```

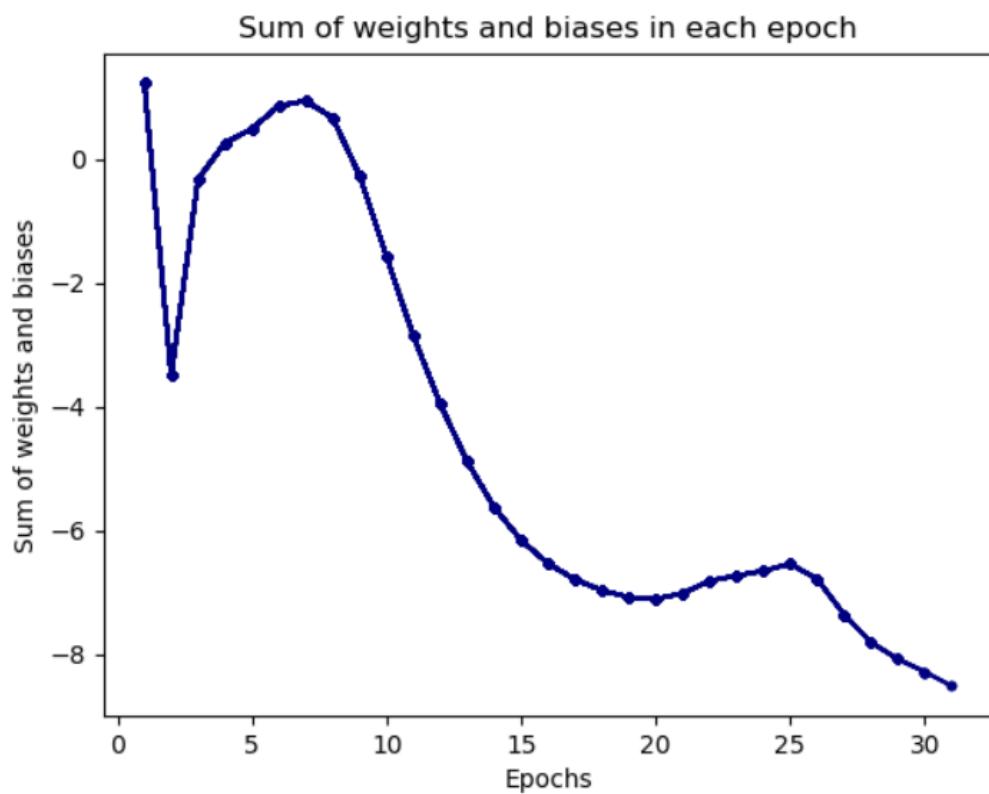
مقدار تابع هزینه بعد از آخرین ایپاک برابر با 0.016 است. خطای مجموعه آموزش برابر ۱,۲ درصد است، خطای مجموعه ارزیابی برابر با ۴ درصد است و خطای مجموعه تست برابر با ۴,۶ درصد است. خطای 5-fold CV برابر با ۲,۱ درصد است.

مقدار تابع هزینه در هر ایپاک:



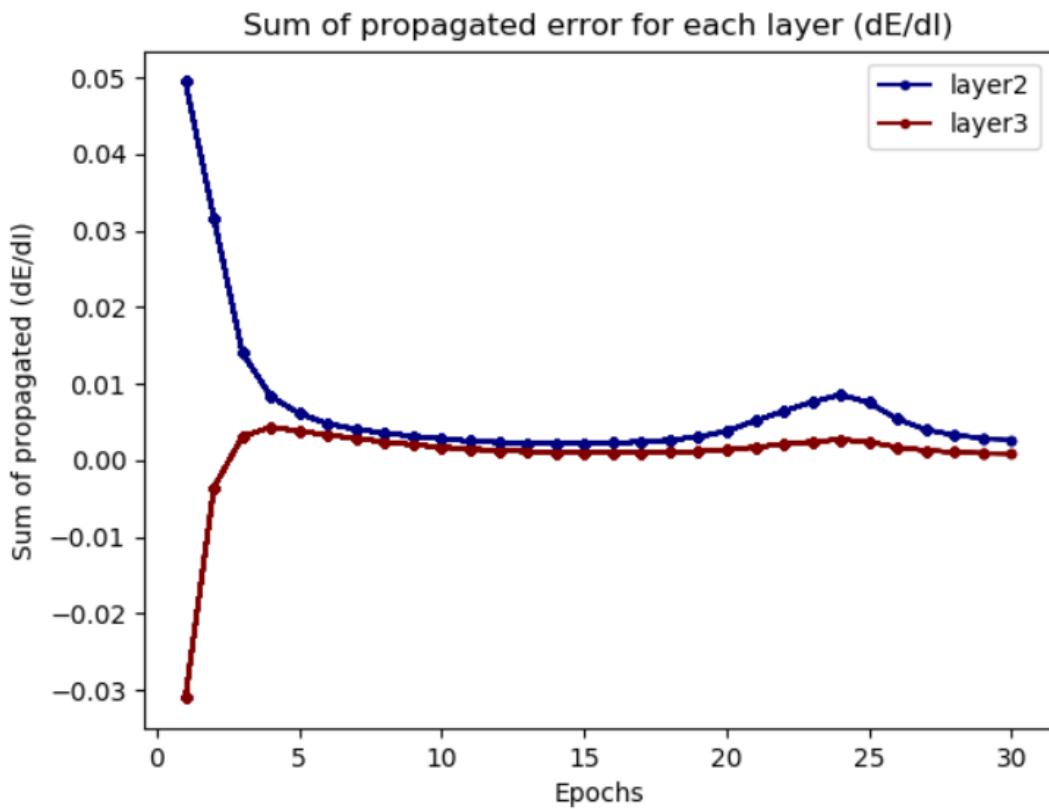
در طی ۳۰ ایپاک تابع هزینه MSE از ۱,۳۲ به ۰,۰ رسیده است که میزان کاهش چشمگیر است. مقدار تابع هزینه سیر نزولی داشته است و بیشترین تغییرات در زمان انجام ایپاک اول بوده است و هر چه جلوتر رفتهایم در ایپاک‌های بعد از میزان تغییرات کاسته شده است و با روند کندتری تابع هزینه کم شده است.

مجموع وزن‌ها و بایاس‌ها در هر ایپاک:



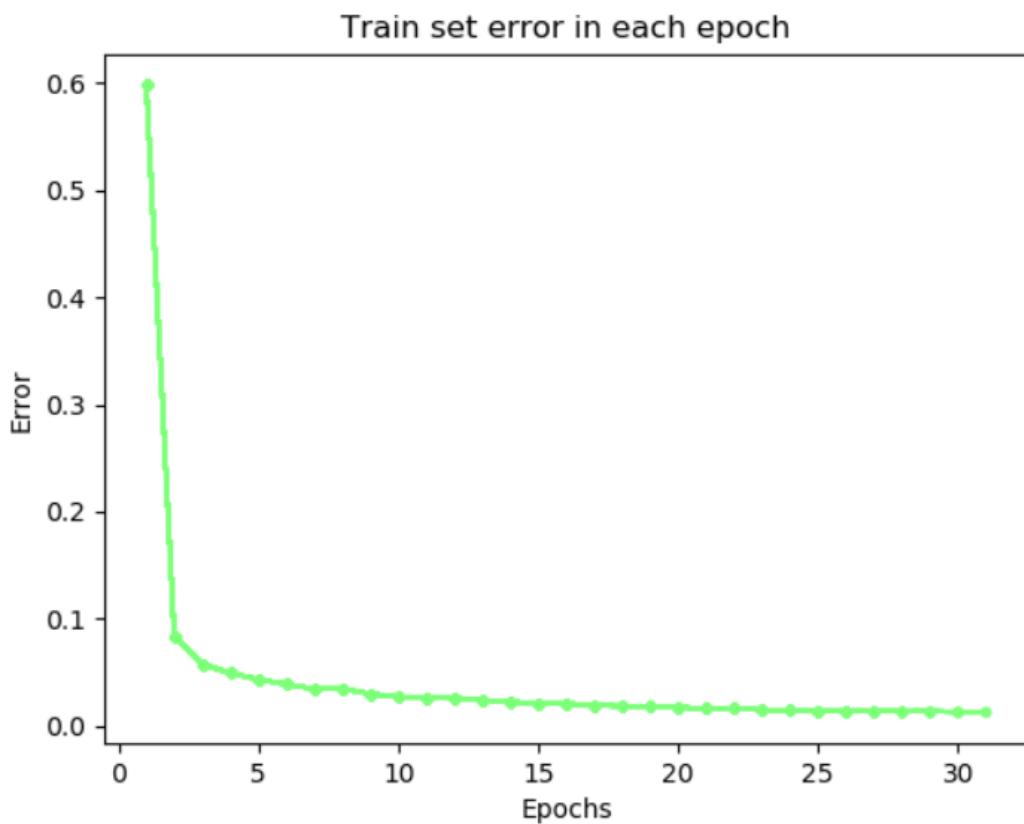
در ایپاک‌های آخر میزان تغییرات مجموعه وزن‌ها و بایاس‌ها کم شده است و تقریباً در آن محدوده باقی می‌ماند و تقریباً در چند ایپاک آخر مقدار مجموع ثابت مانده است. زیرا هر چه جلوتر می‌رویم معمولاً گرادیان‌ها کوچک‌تر می‌شوند و تغییرات وزن‌ها نسبت به مراحل قبل کمتر می‌شود.

مجموع خطاهای منتشر شده در هر کدام از لایه‌های نهان:



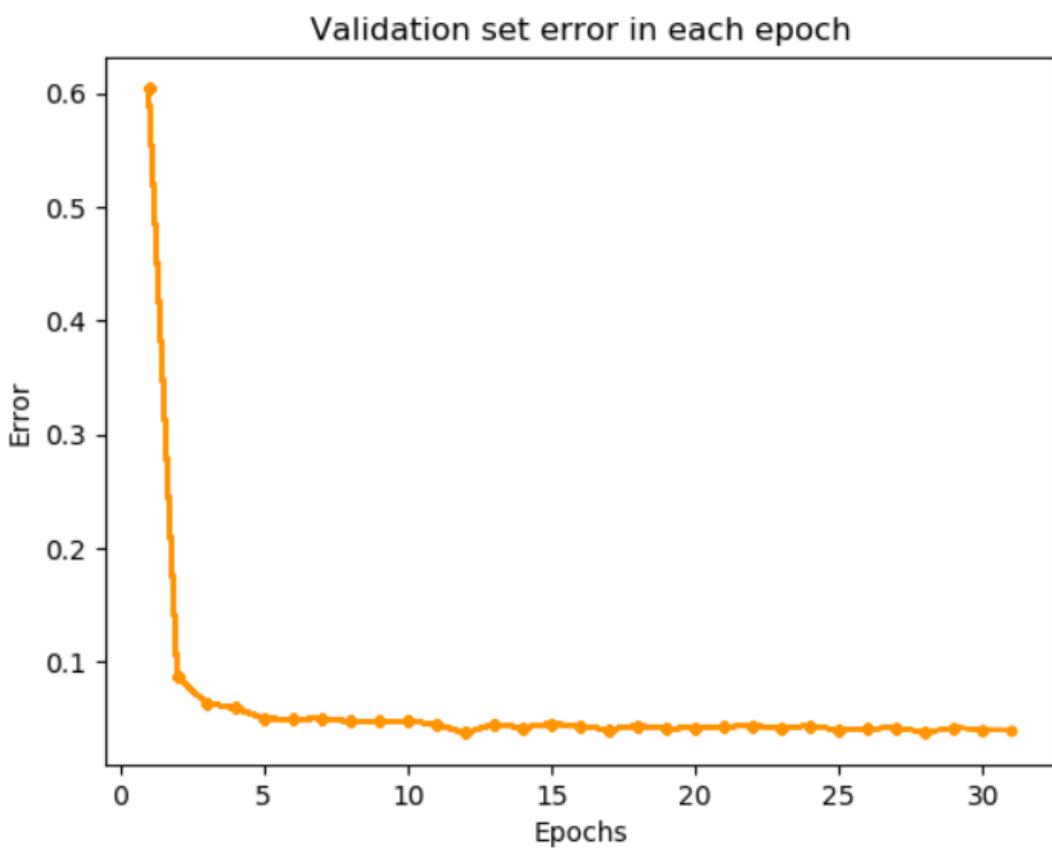
همین طور که مشاهده می شود مجموع خطاهای هر لایه بعد از مدتی، ثابت می شود زیرا هر چه به ایپاک های جلوتری می رویم گرادیان ها کمتر و کمتر می شود و به صفر میل می کنند.

خطای مجموعه‌ی آموزش در هر ایپاک:



خطای مجموعه آموزش با آموزش بیشتر شبکه کاهش داشته است. بیشتر کاهش در ایپاک اول رخ داده است و در ایپاک‌های آخر میزان تغییرات خطا کم شده است و به مقدار ثابتی میل کرده است.

خطای مجموعه ارزیابی در هر ایپاک:



خطای ارزیابی در هر ایپاک با آموزش بیشتر کاهش داشته است. در ایپاک‌های اول بیشترین تغییرات را داشته‌ایم و در ایپاک‌های پایانی از میزان تغییرات کاسته شده است.

این شبکه در فایل `MLP-output-10.json` ذخیره شده است.

نتیجه‌های شبکه عصبی دوم:

مقدار تابع هزینه در آخرین ایپاک، خطای آموزش، خطای ارزیابی، خطای تست :

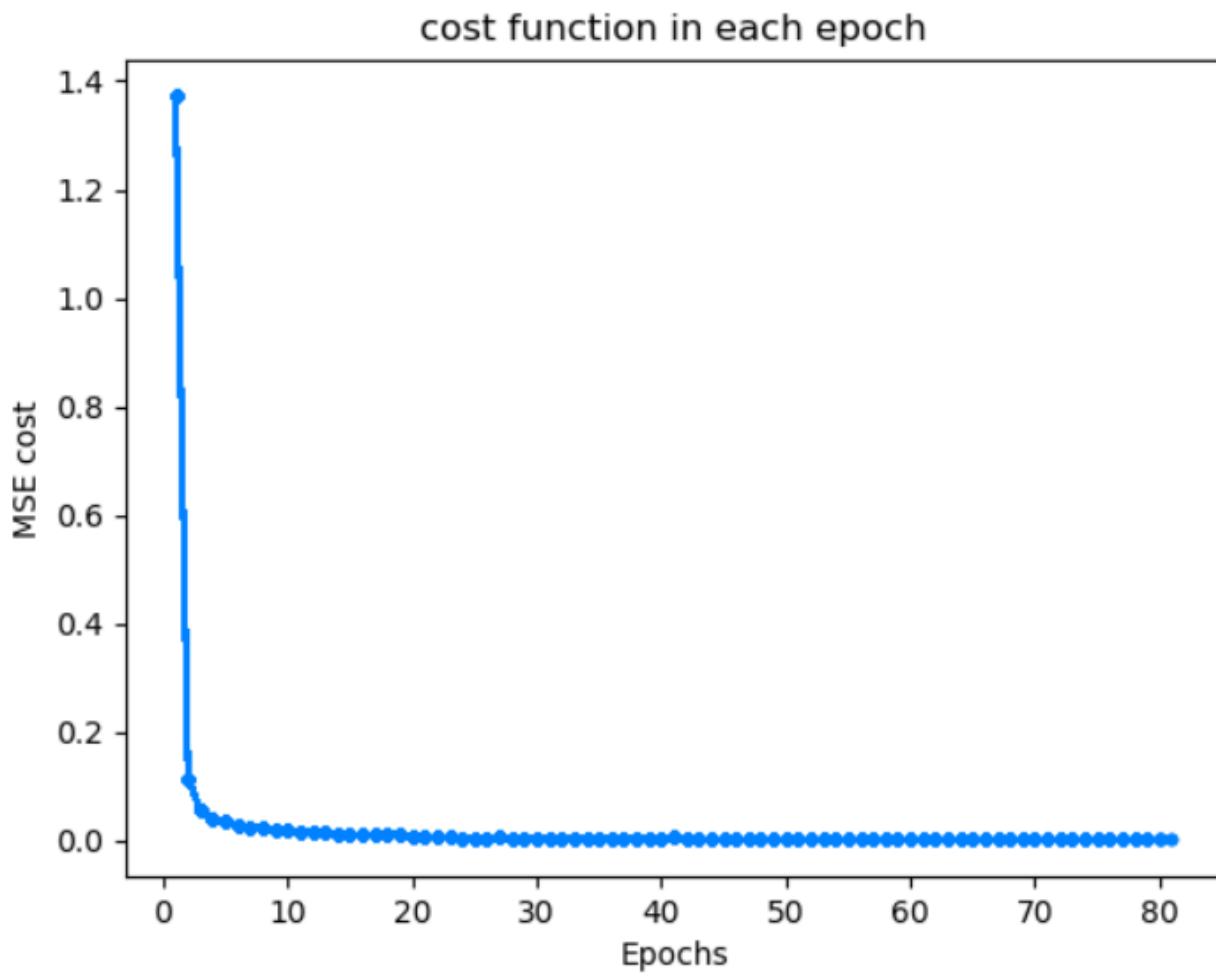
```

Iter 79/80, Cost 0.0018601627440086007 ...
Iter 80/80, Cost 0.0018476628768038614 ...
=====
Train Error: 0.002573529411764706
Validation Error: 0.02588235294117647
Test Error: 0.03529411764705882

```

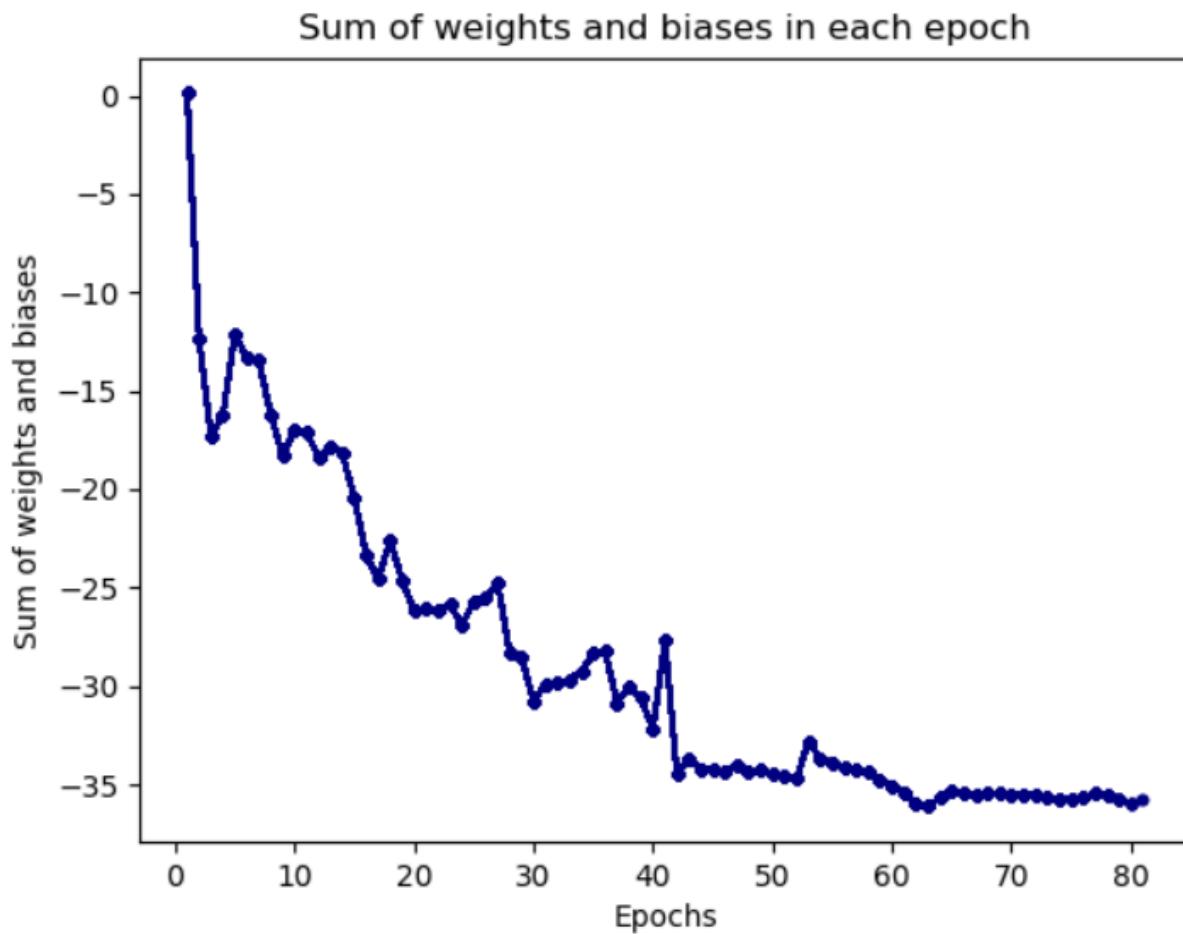
مقدار تابع هزینه بعد از آخرین ایپاک برابر با 0.018 است. خطای مجموعه آموزش برابر 0.25 درصد است، خطای مجموعه ارزیابی برابر با 2.58 درصد است و خطای مجموعه تست برابر با 3.52 درصد است. خطای 5-fold CV برابر با درصد است.

مقدار تابع هزینه در هر ایپاک:



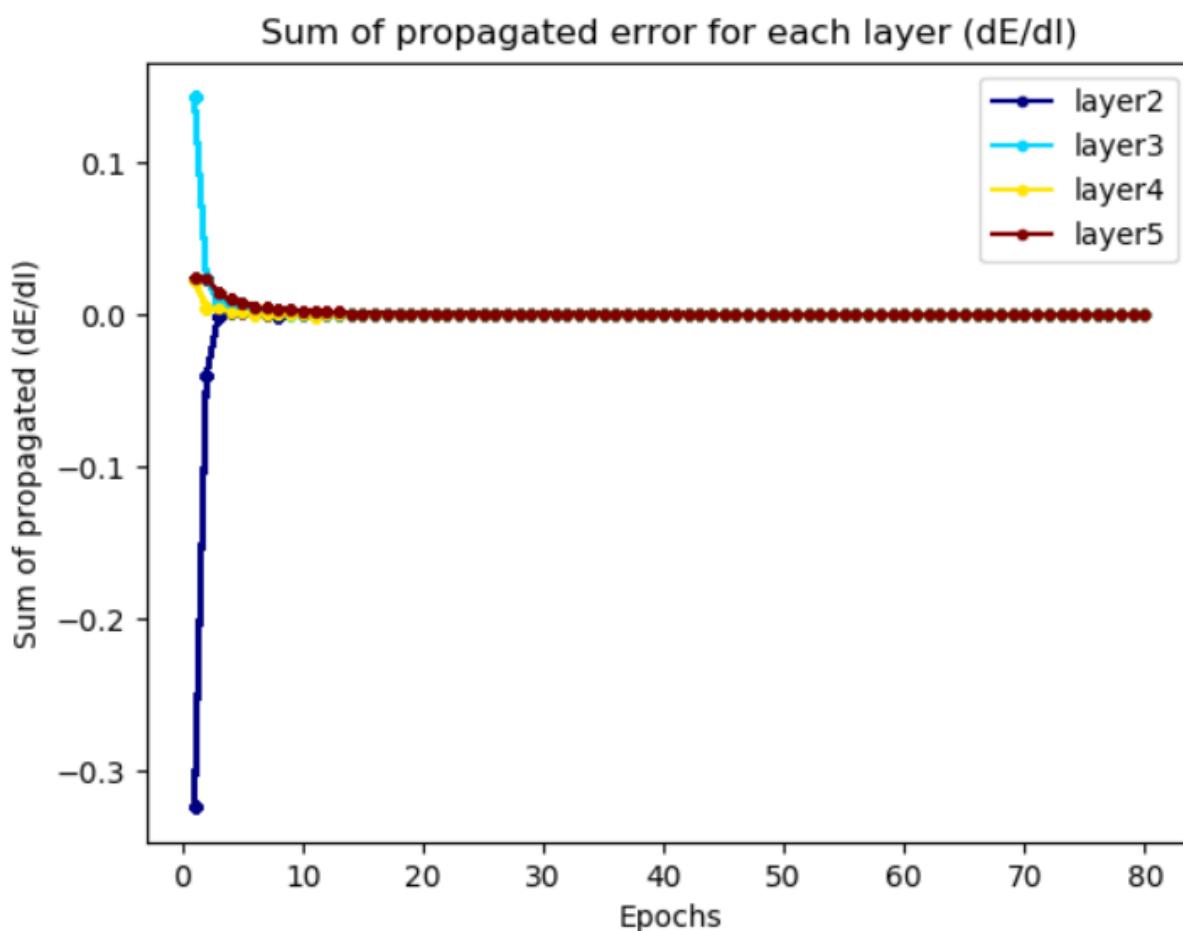
در طی ۳۰ ایپاک تابع هزینه MSE از ۱,۳۲ به ۰,۰ رسیده است که میزان کاهش چشمگیر است. مقدار تابع هزینه سیر نزولی داشته است و بیشترین تغییرات در زمان انجام ایپاک اول بوده است و هر چه جلوتر رفتهایم در ایپاک‌های بعد از میزان تغییرات کاسته شده است و با روند کندتری تابع هزینه کم شده است.

مجموع وزن‌ها و بایاس‌ها در هر ایپاک:



در ایپاک‌های آخر میزان تغییرات مجموعه وزن‌ها و بایاس‌ها کم شده است و تقریباً در آن محدوده باقی می‌ماند و تقریباً در چند ایپاک آخر مقدار مجموع ثابت مانده است. زیرا هر چه جلوتر می‌رویم معمولاً گرادیان‌ها کوچک‌تر می‌شوند و تغییرات وزن‌ها نسبت به مراحل قبل کمتر می‌شود.

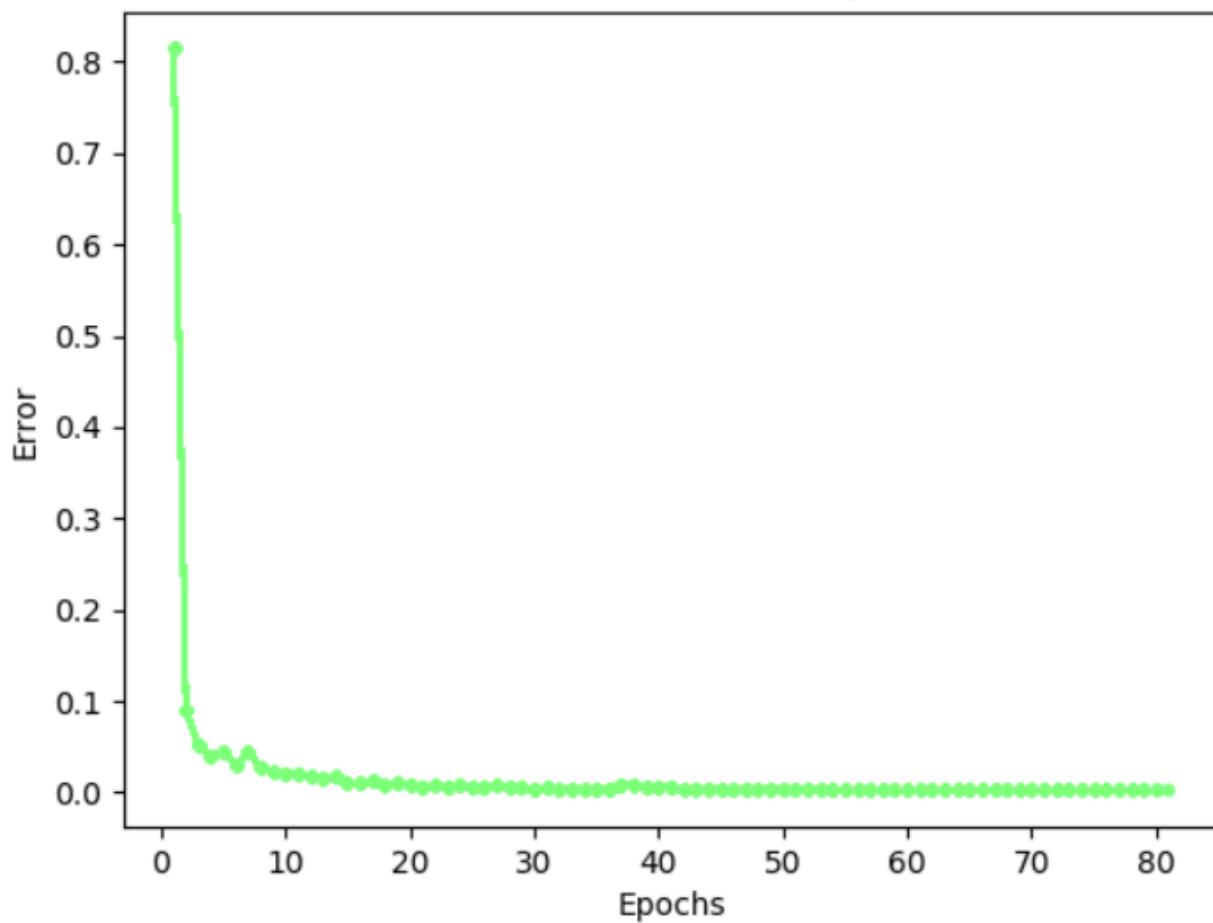
مجموع خطاهای منتشر شده در هر کدام از لایه‌های نهان:



همین طور که مشاهده می‌شود مجموع خطاهای هر لایه بعد از مدتی، ثابت می‌شود زیرا هر چه به ایپاک‌های جلوتری می‌رویم گرادیان‌ها کمتر و کمتر می‌شود و به صفر میل می‌کنند.

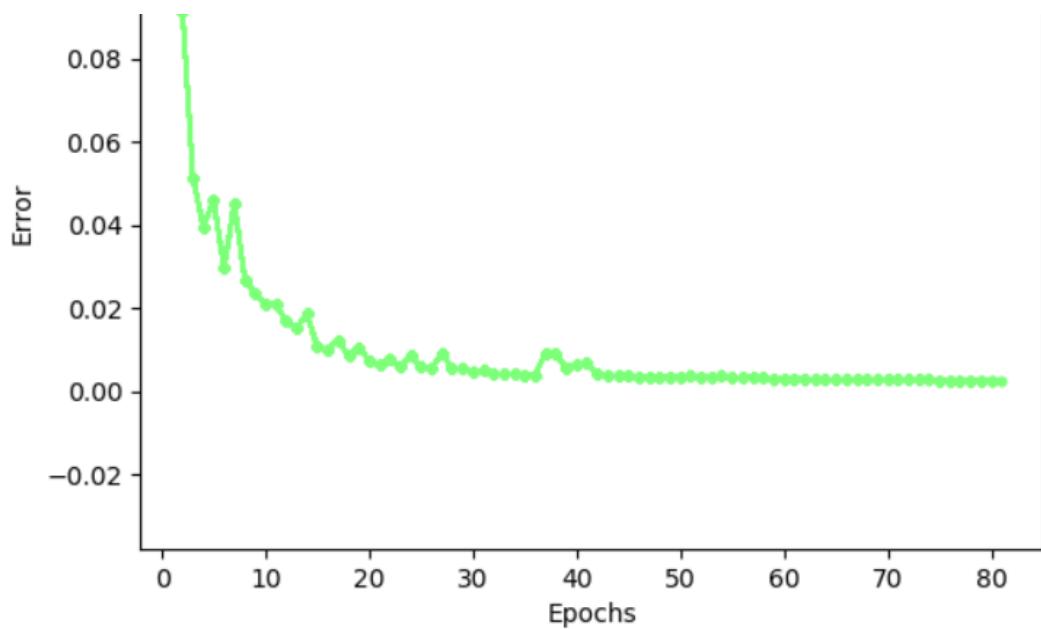
خطای مجموعه‌ی آموزش در هر ایپاک:

Train set error in each epoch

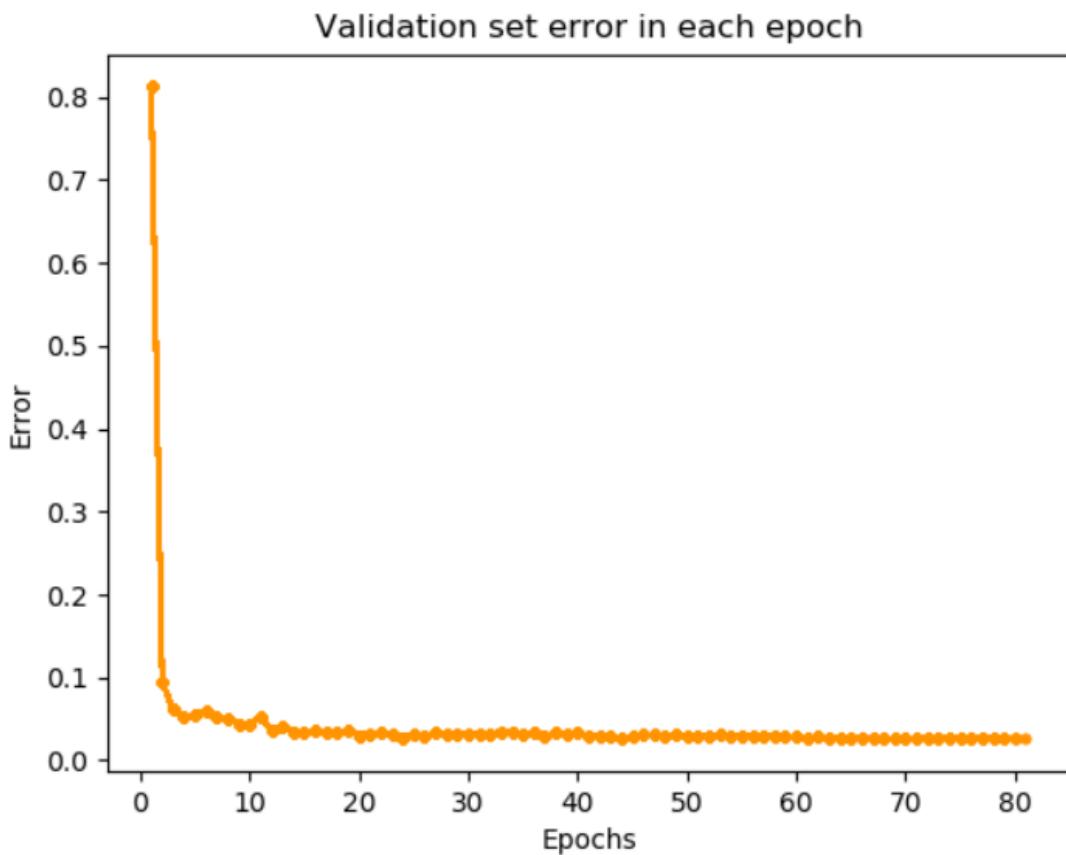


خطای مجموعه آموزش با آموزش بیشتر شبکه کاهش داشته است. بیشتر کاهش در ایپاک اول رخ داده است و در ایپاک های آخر میزان تغییرات خطا کم شده است و به مقدار ثابتی میل کرده است.

بخشی از نمودار بالا با بزرگنمایی:

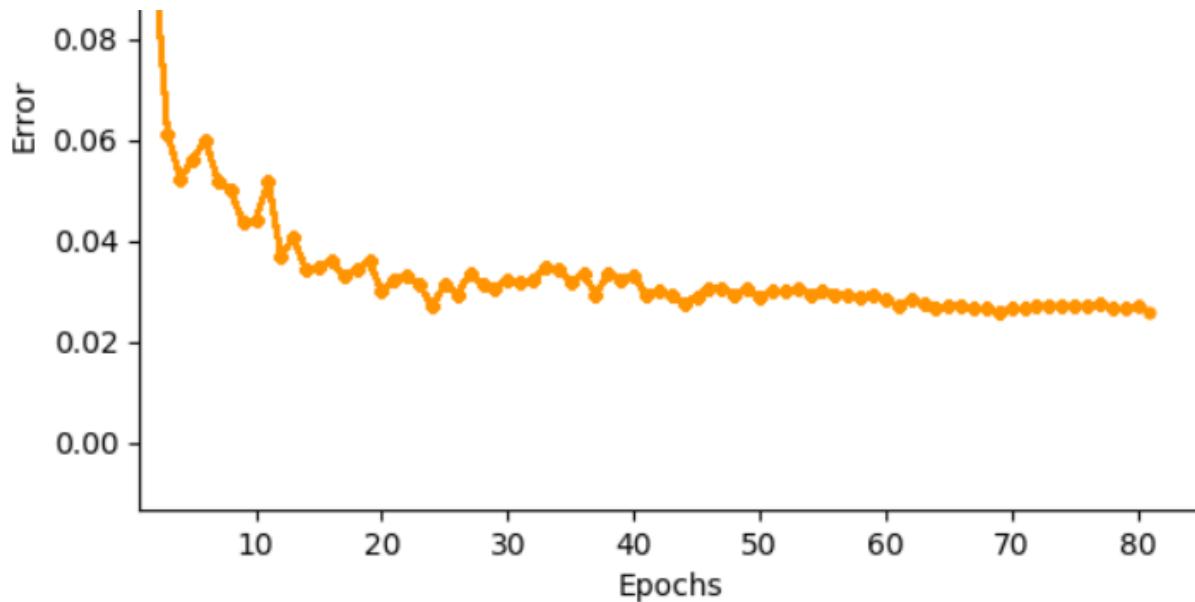


خطای مجموعه ارزیابی در هر ایپاک:



خطای ارزیابی در هر ایپاک با آموزش بیشتر کاهش داشته است. در ایپاک‌های اول بیشترین تغییرات را داشته‌ایم و در ایپاک‌های پایانی از میزان تغییرات کاسته شده است.

بخشی از نمودار بالا با بزرگنمایی:



این شبکه در فایل MLP-output-11.json ذخیره شده است.

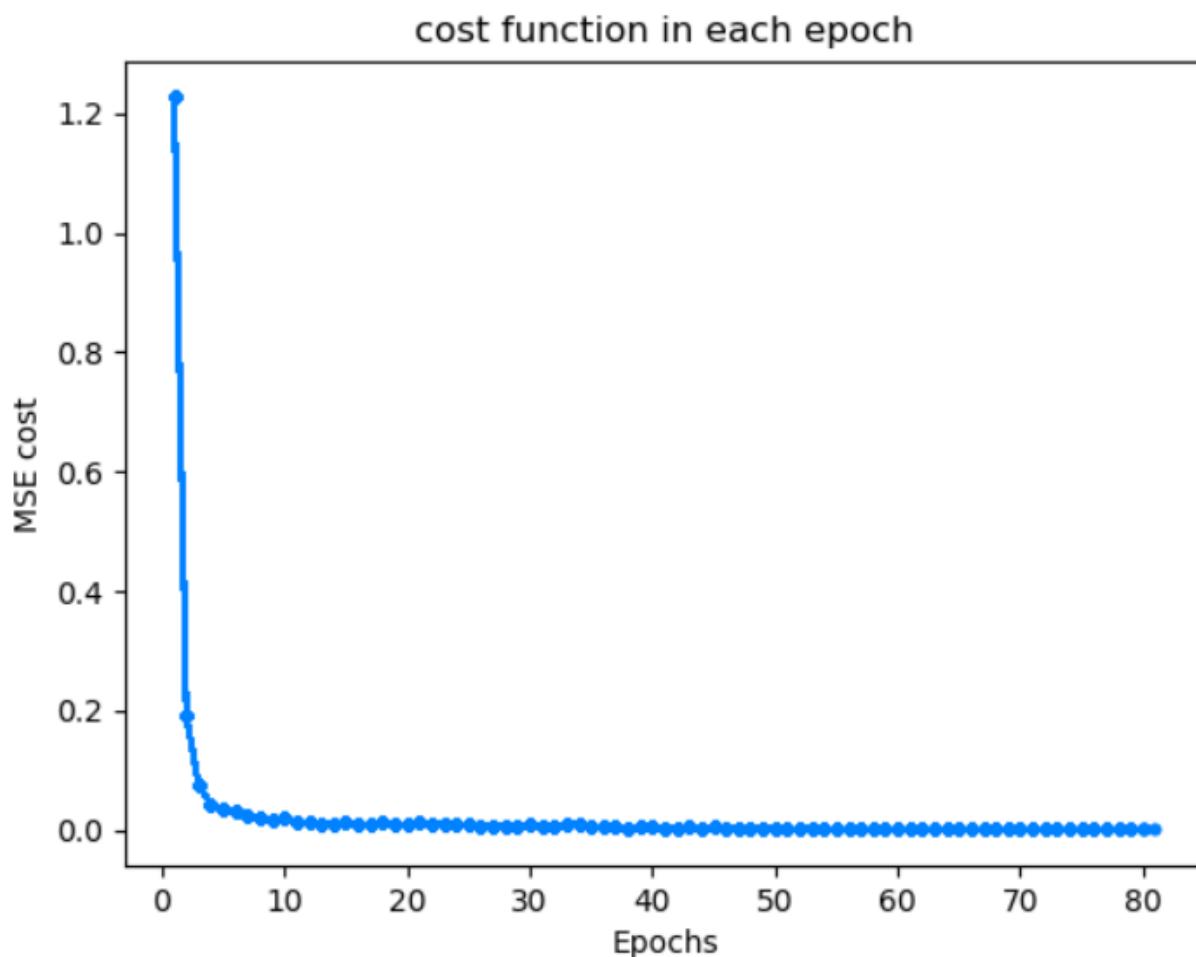
نتیجه‌های شبکه عصبی سوم:

مقدار تابع هزینه در آخرین ایپاک، خطای آموزش، خطای ارزیابی، خطای تست :

```
Iter 79/80, Cost 0.0016691693271815215 ...
Iter 80/80, Cost 0.0016648205667539695 ...
=====
Train Error: 0.0021323529411764706
Validation Error: 0.028823529411764706
Test Error: 0.028823529411764706
```

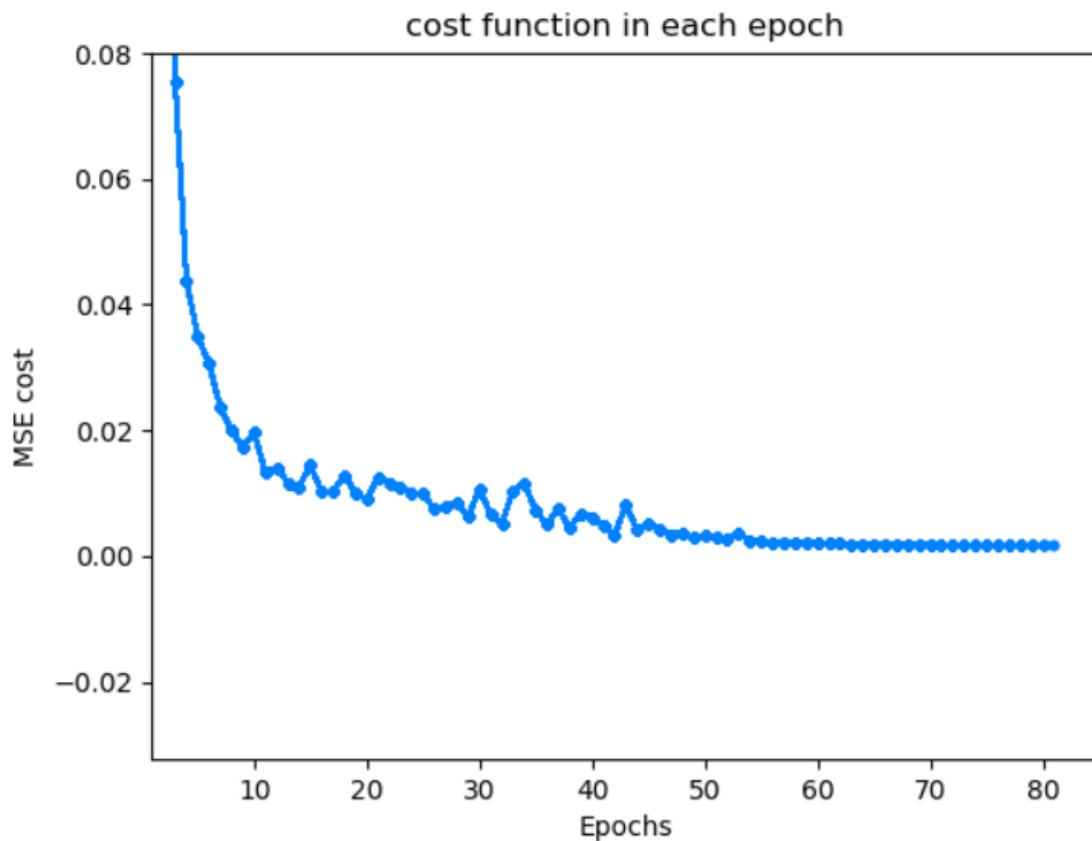
مقدار تابع هزینه بعد از آخرین ایپاک برابر با 0.0016 می‌باشد. خطای مجموعه آموزش برابر 0.21 درصد است، خطای مجموعه ارزیابی برابر با 2.88 درصد است و خطای مجموعه تست برابر با 2.88 درصد است.

مقدار تابع هزینه در هر ایپاک:

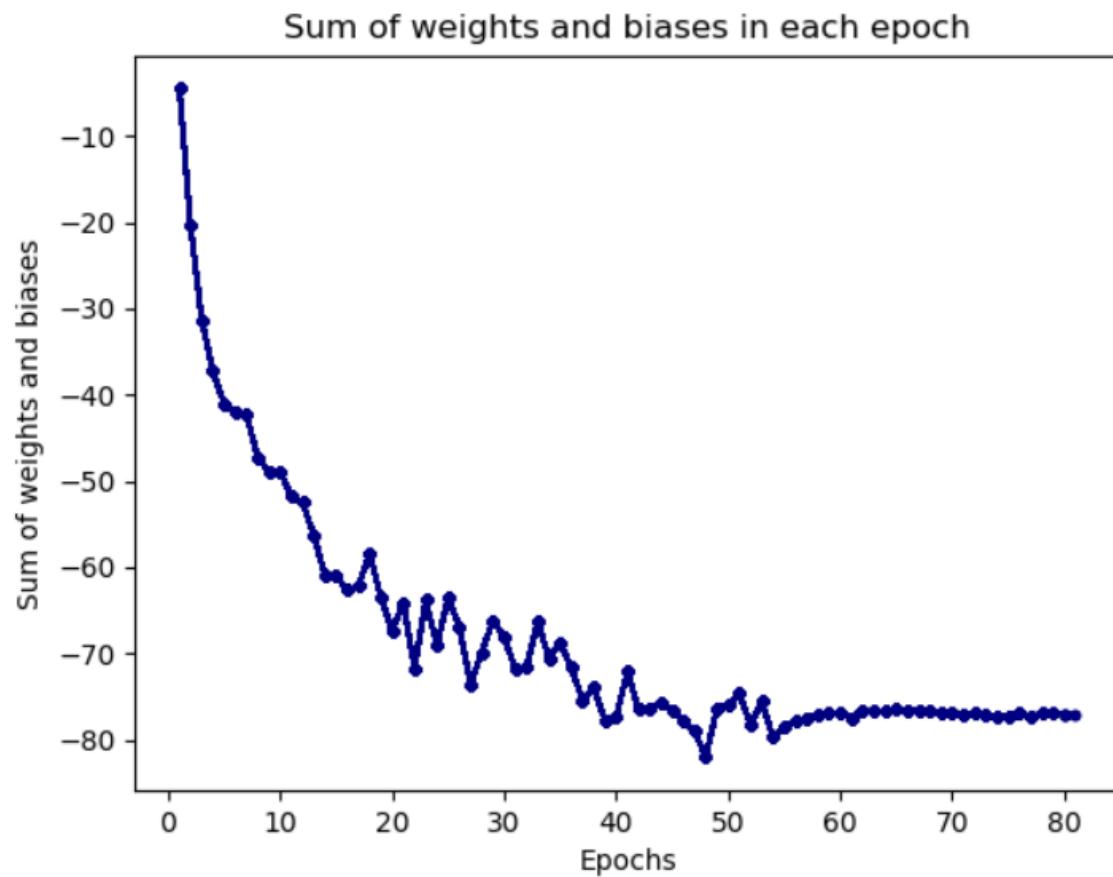


در طی ۳۰ ایپاک تابع هزینه‌ی MSE از ۱,۳۲ به ۰,۰۰۱۶ رسیده است که میزان کاهش چشمگیر است. مقدار تابع هزینه سیر نزولی داشته است و بیشترین تغییرات در زمان انجام ایپاک اول بوده است و هر چه جلوتر رفته‌ایم در ایپاک‌های بعد از میزان تغییرات کاسته شده است و با روند کند تری تابع هزینه کم شده است.

قسمتی از نمودار بالا با بزرگنمایی:

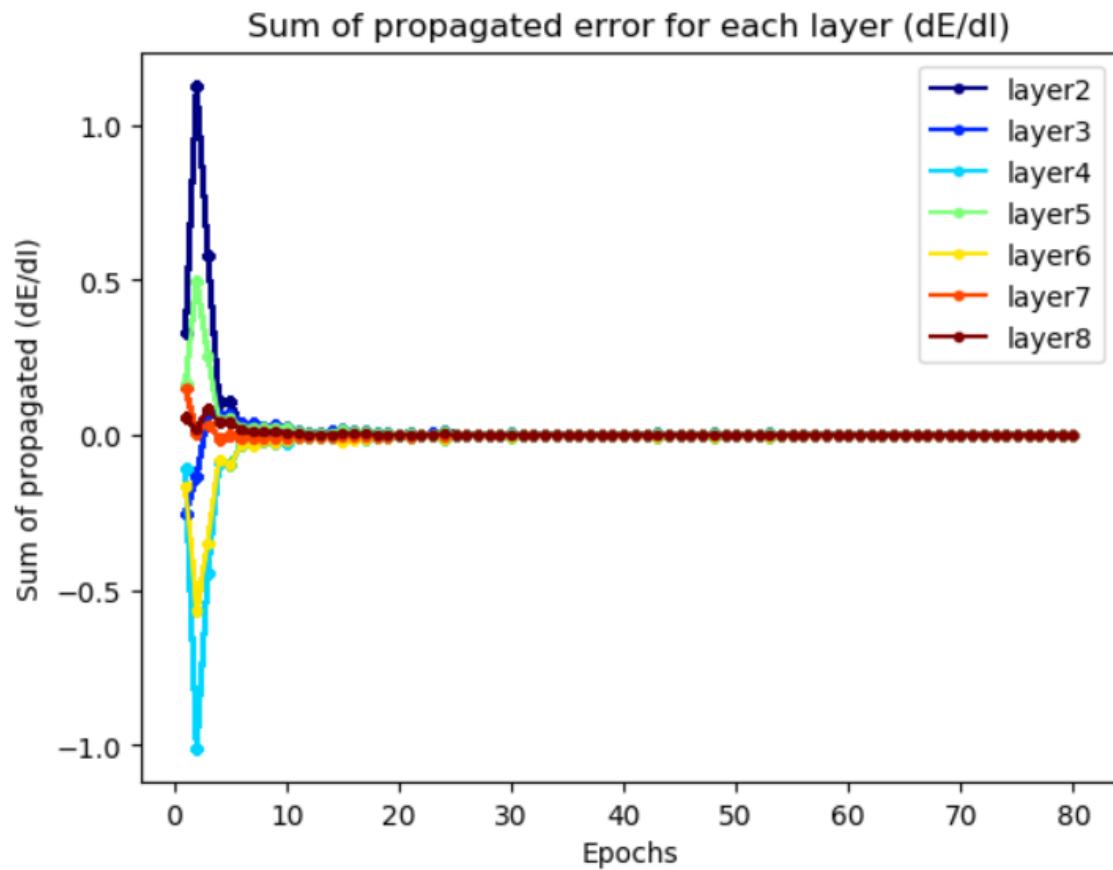


مجموع وزن‌ها و بایاس‌ها در هر ایپاک:



در ایپاک‌های آخر میزان تغییرات مجموعه وزن‌ها و بیاس‌ها کم شده است و تقریباً در آن محدوده باقی می‌ماند و تقریباً در چند ایپاک آخر مقدار مجموع ثابت مانده است. زیرا هر چه جلوتر می‌رویم معمولاً گرادیان‌ها کوچک‌تر می‌شوند و تغییرات وزن‌ها نسبت به مراحل قبل کمتر می‌شود.

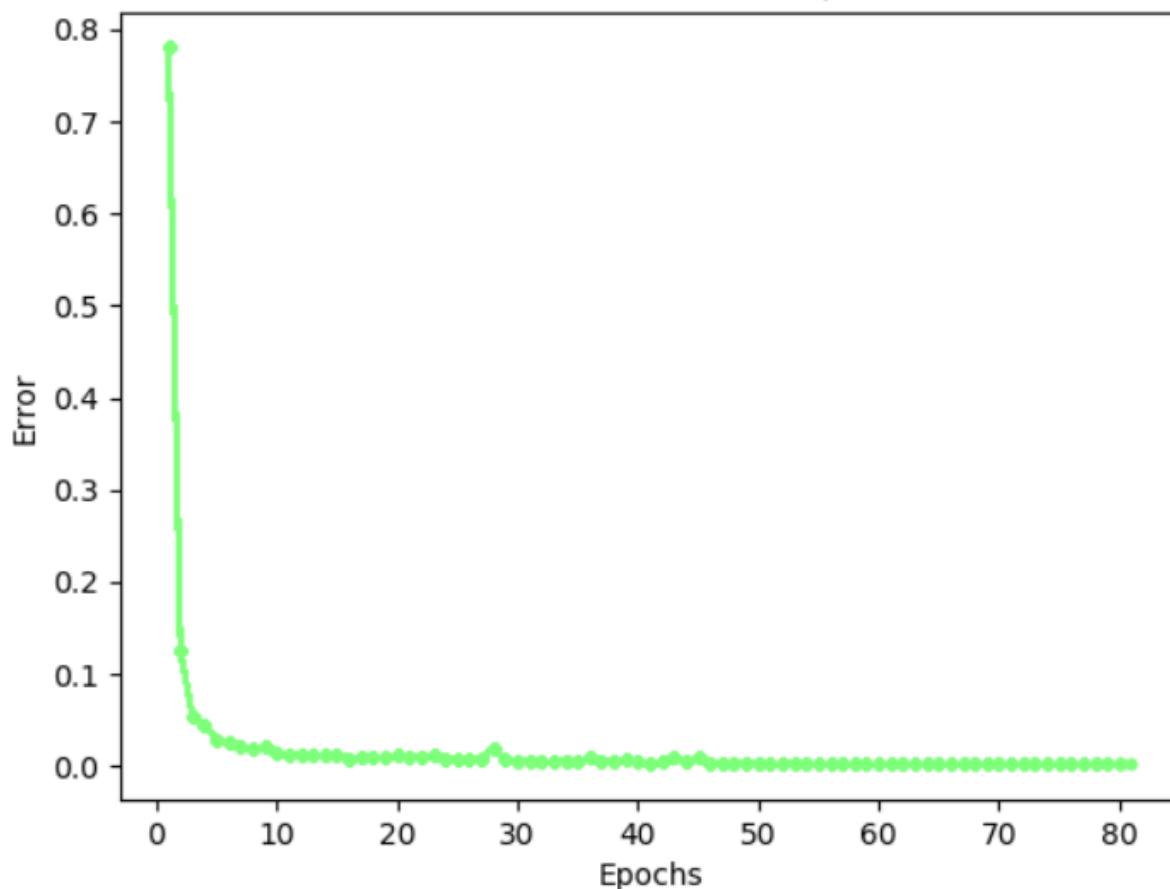
مجموع خطاهای منتشر شده در هر کدام از لایه‌های نهان:



همین طور که مشاهده می شود مجموع خطاهای هر لایه بعد از مدتی، ثابت می شود زیرا هر چه به ایپاک های جلوتری می رویم گرادیان ها کمتر و کمتر می شود و به صفر میل می کنند.

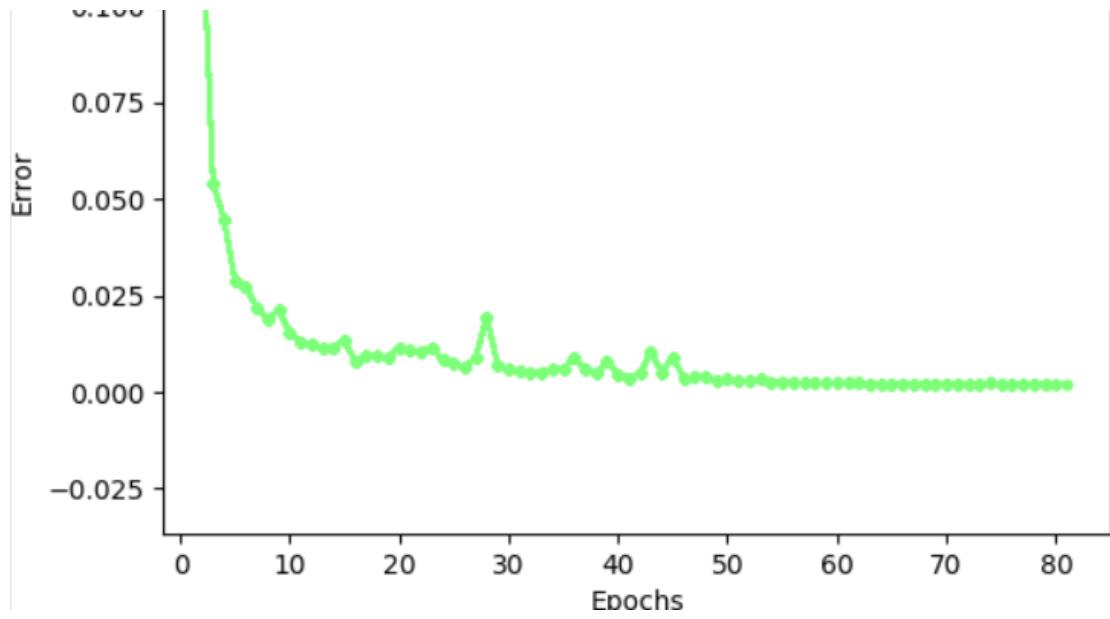
خطای مجموعه‌ی آموزش در هر ایپاک:

Train set error in each epoch

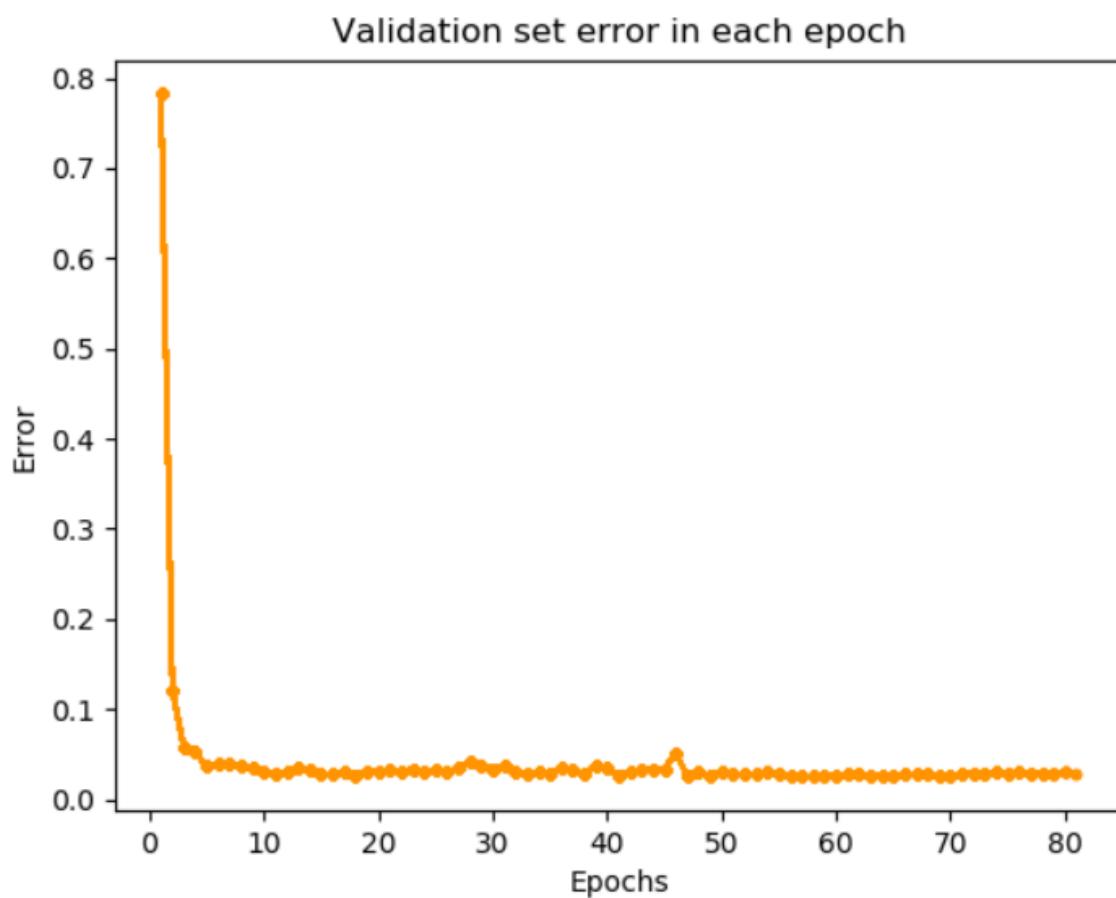


خطای مجموعه آموزش با آموزش بیشتر شبکه کاهش داشته است. بیشتر کاهش در ایپاک اول رخ داده است و در ایپاک های آخر میزان تغییرات خطا کم شده است و به مقدار ثابتی میل کرده است.

بخشی از نمودار بالا با بزرگنمایی:

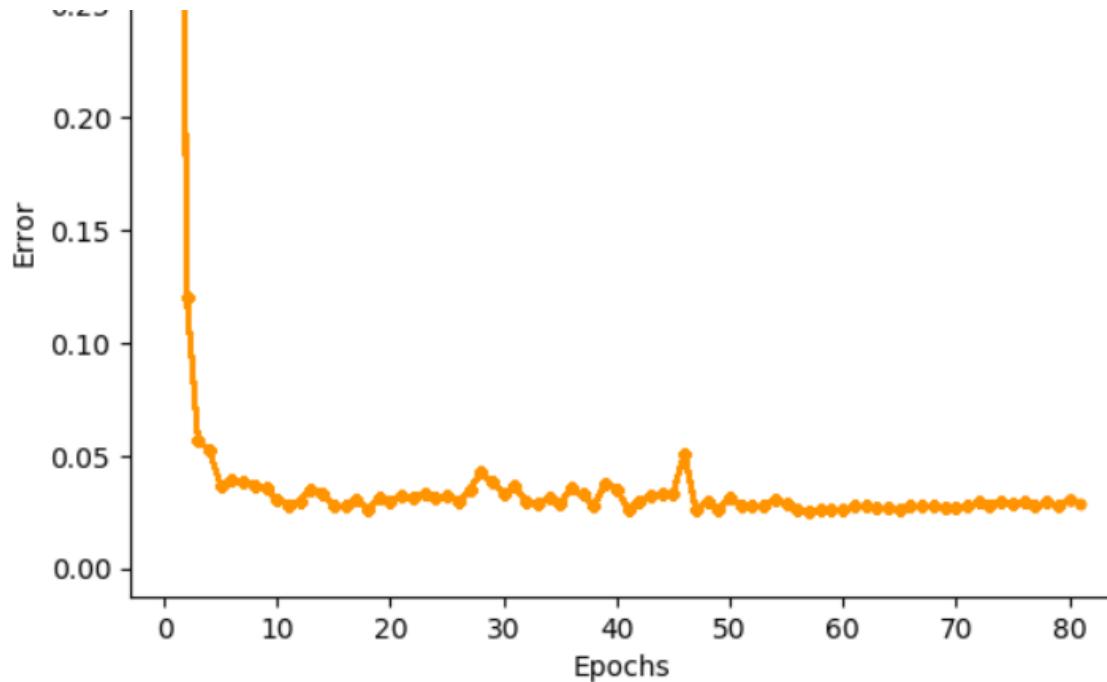


خطای مجموعه ارزیابی در هر ایپاک:



خطای ارزیابی در هر ایپاک با آموزش بیشتر کاهش داشته است. در ایپاک‌های اول بیشترین تغییرات را داشته‌ایم و در ایپاک‌های پایانی از میزان تغییرات کاسته شده است.

بخشی از نمودار بالا با بزرگنمایی:



این شبکه در فایل MLP-output-12.json ذخیره شده است.

نتیجه‌های شبکه عصبی چهارم:

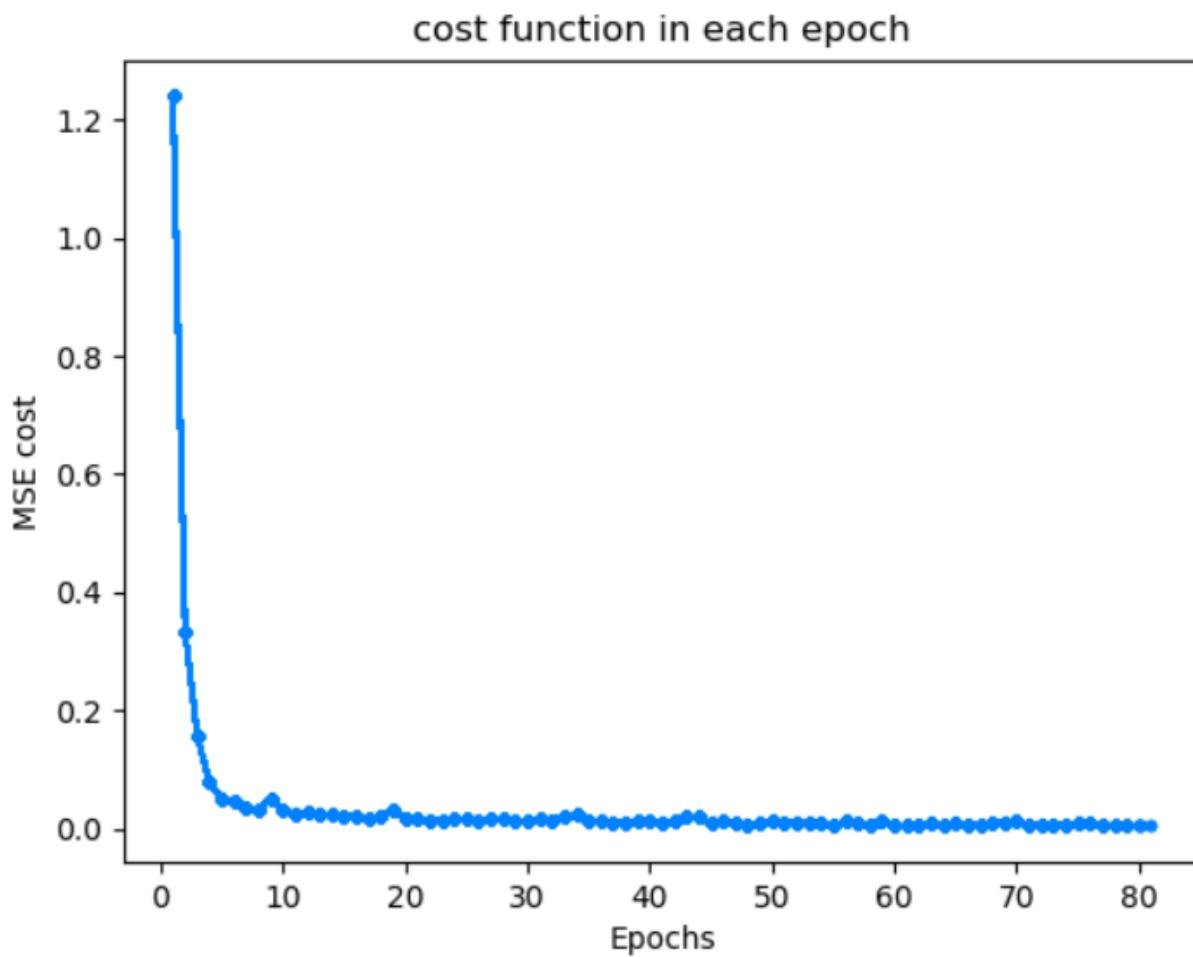
مقدار تابع هزینه در آخرین ایپاک، خطای آموزش، خطای ارزیابی، خطای تست :

```
Iter /9/80, Cost 0.006/64832586206983 ...
Iter 80/80, Cost 0.004845131033178415 ...
=====
Train Error: 0.005294117647058823
Validation Error: 0.03
Test Error: 0.03
```

5-fold error is: 0.01933823529411765

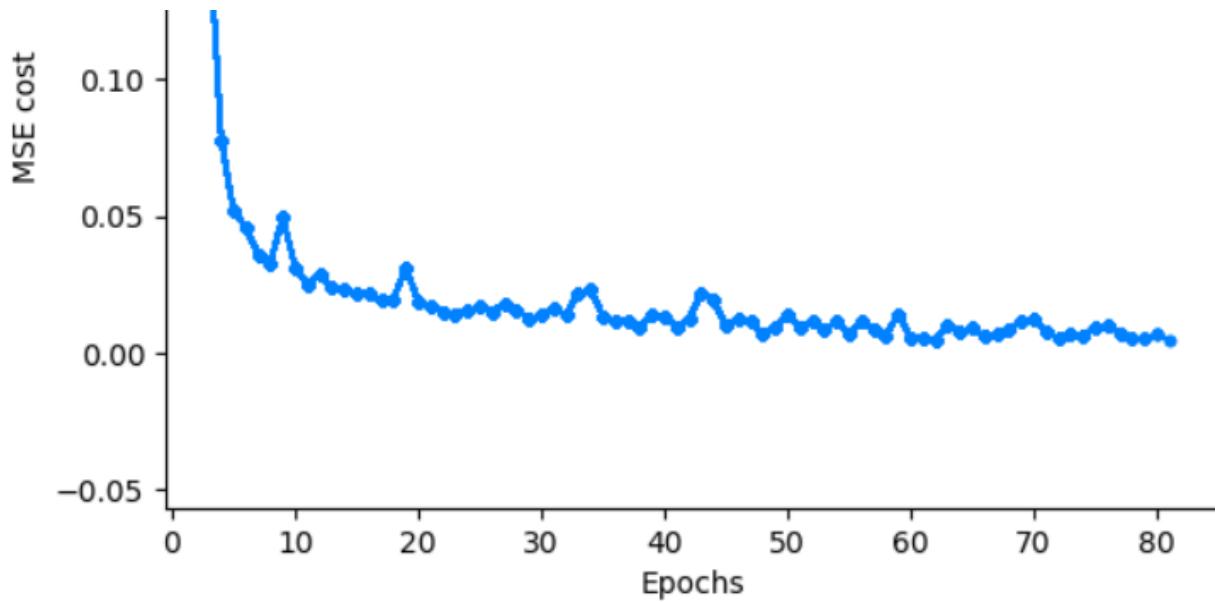
مقدار تابع هزینه بعد از آخرین ایپاک برابر با ۰.۰۰۴۸ است. خطای مجموعه آموزش برابر ۰.۵۲ درصد است، خطای مجموعه ارزیابی برابر با ۳ درصد است و خطای مجموعه تست برابر با ۳ درصد است. خطای K-Fold CV برابر با ۱.۹ درصد است.

مقدار تابع هزینه در هر ایپاک:



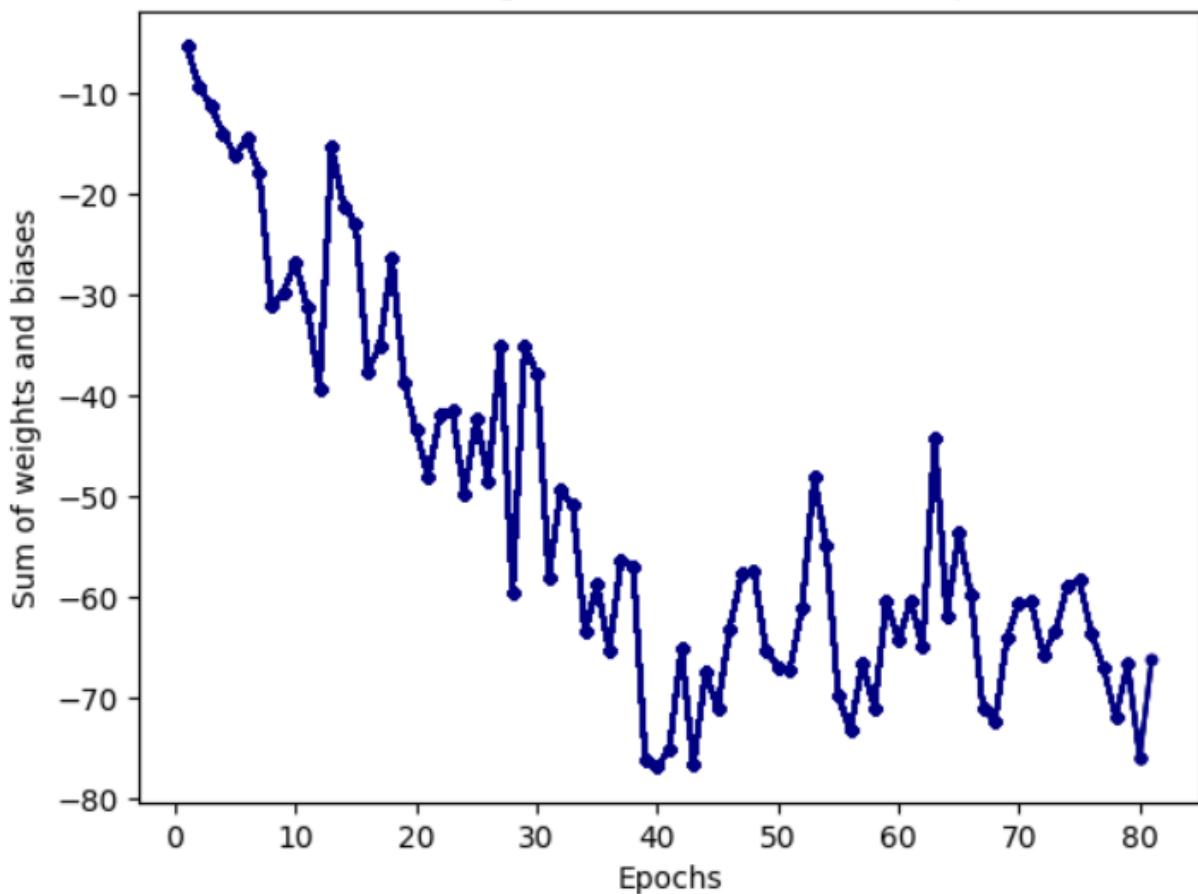
در طی ۸۰ ایپاک تابع هزینه‌ی MSE از ۱.۳۲ به ۰.۰۰۴۸ رسیده است که میزان کاهش چشم‌گیر است. مقدار تابع هزینه سیر نزولی داشته است و بیشترین تغییرات در زمان انجام ایپاک اول بوده است و هر چه جلوتر رفته‌ایم در ایپاک‌های بعد از میزان تغییرات کاسته شده است و با روند کند تری تابع هزینه کم شده است.

قسمتی از نمودار بالا با بزرگنمایی:



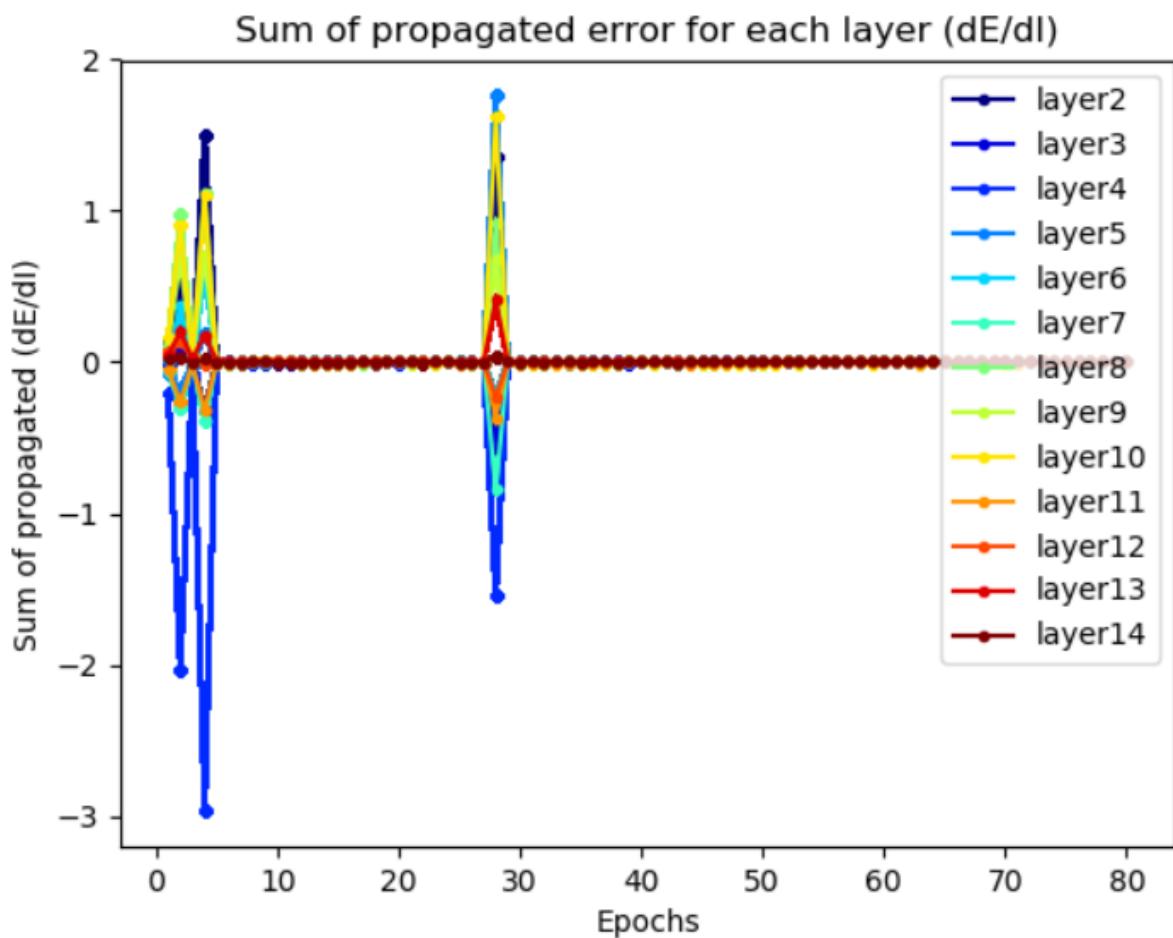
مجموع وزن‌ها و بایاس‌ها در هر ایپاک:

Sum of weights and biases in each epoch



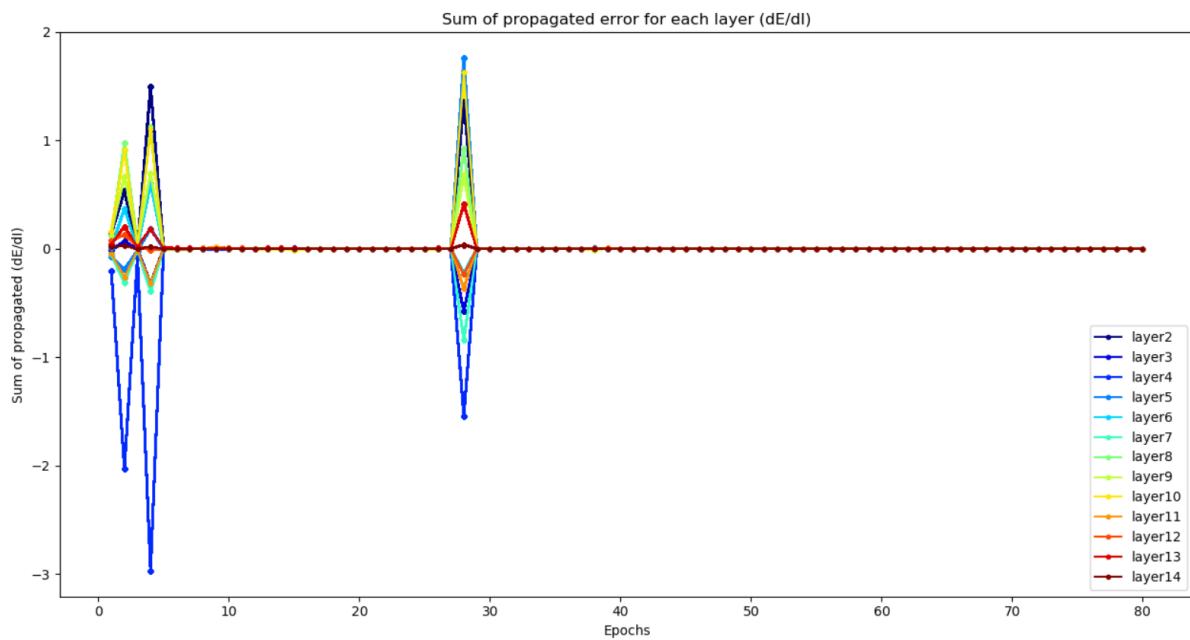
در ایپاک‌های آخر میزان تغییرات مجموعه وزن‌ها و بایاس‌ها کم شده است و تقریباً در آن محدوده باقی می‌ماند و تقریباً در چند ایپاک آخر مقدار مجموع ثابت مانده است. زیرا هر چه جلوتر می‌رویم معمولاً گرادیان‌ها کوچک‌تر می‌شوند و تغییرات وزن‌ها نسبت به مراحل قبل کمتر می‌شود.

مجموع خطاهای منتشر شده در هر کدام از لایه‌های نهان:

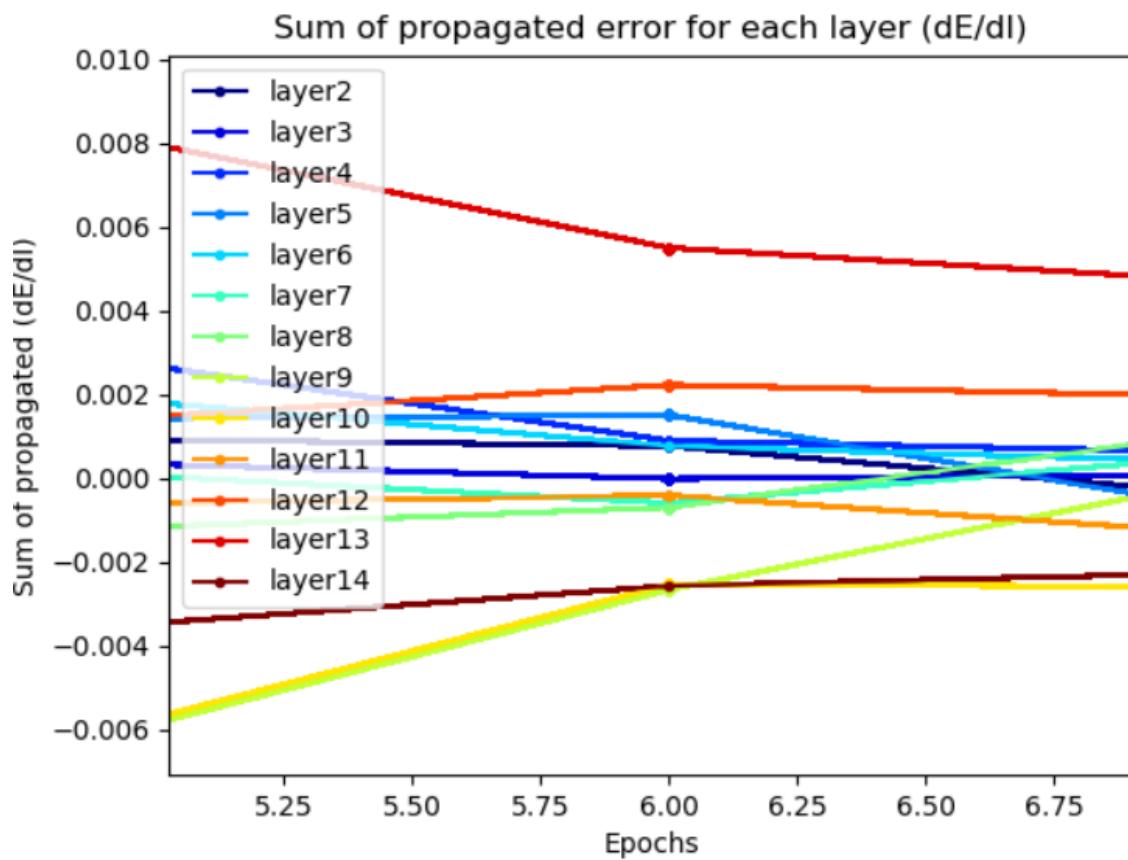


همین طور که مشاهده می شود مجموع خطاهای هر لایه بعد از مدتی، ثابت می شود زیرا هر چه به ایپاکهای جلوتری می رویم گرادیان ها کمتر و کمتر می شود و به صفر میل می کنند.

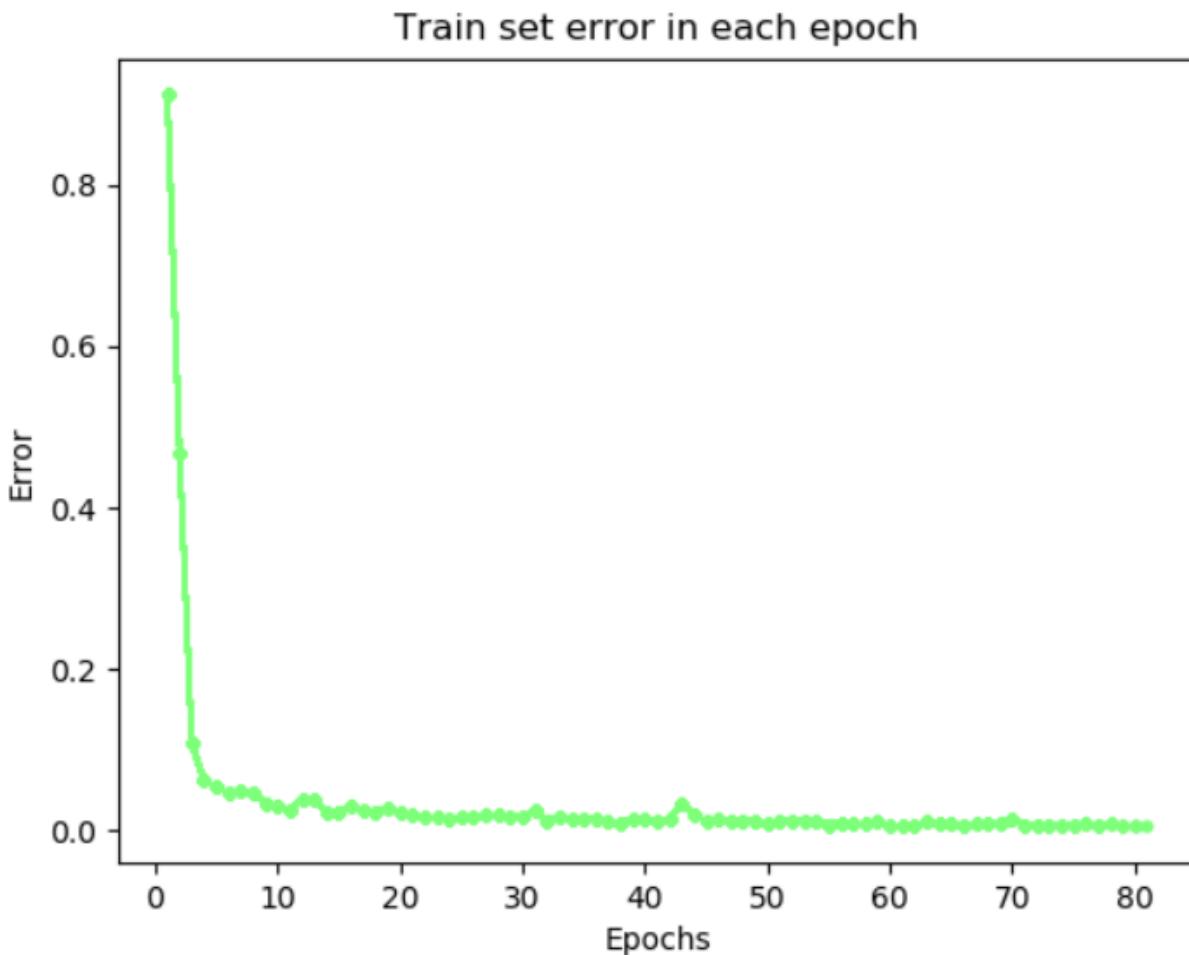
نمودار بالا با نمایشی دیگر:



بخشی از نمودار بالا با بزرگنمایی:

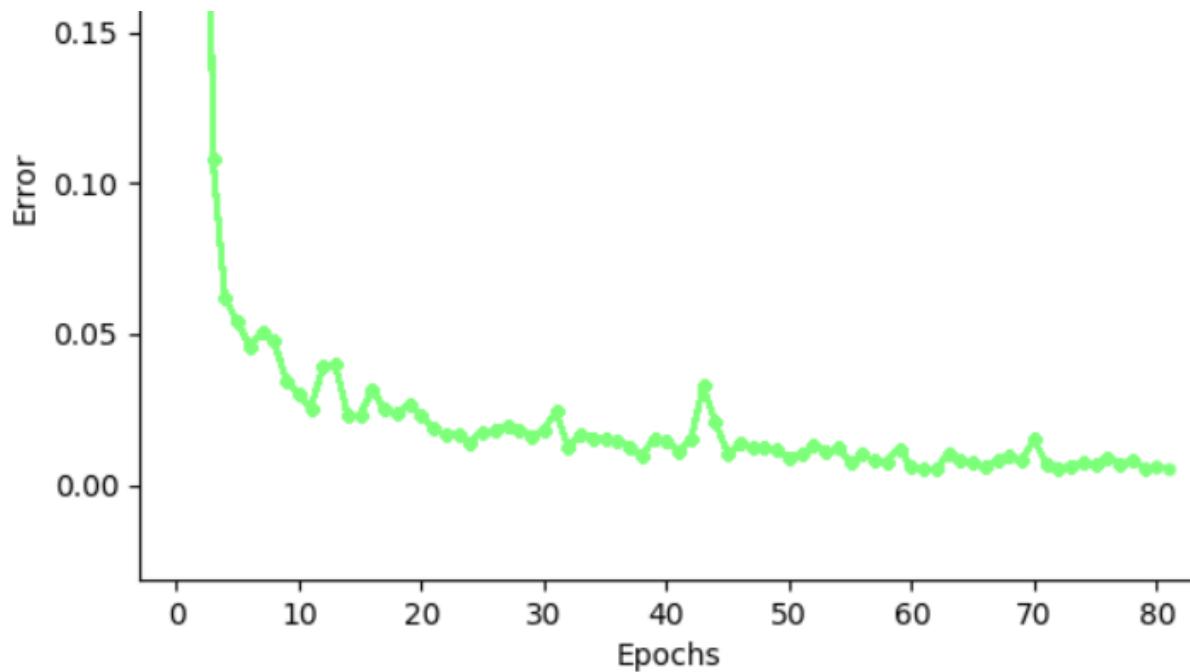


خطای مجموعه‌ی آموزش در هر ایپاک:

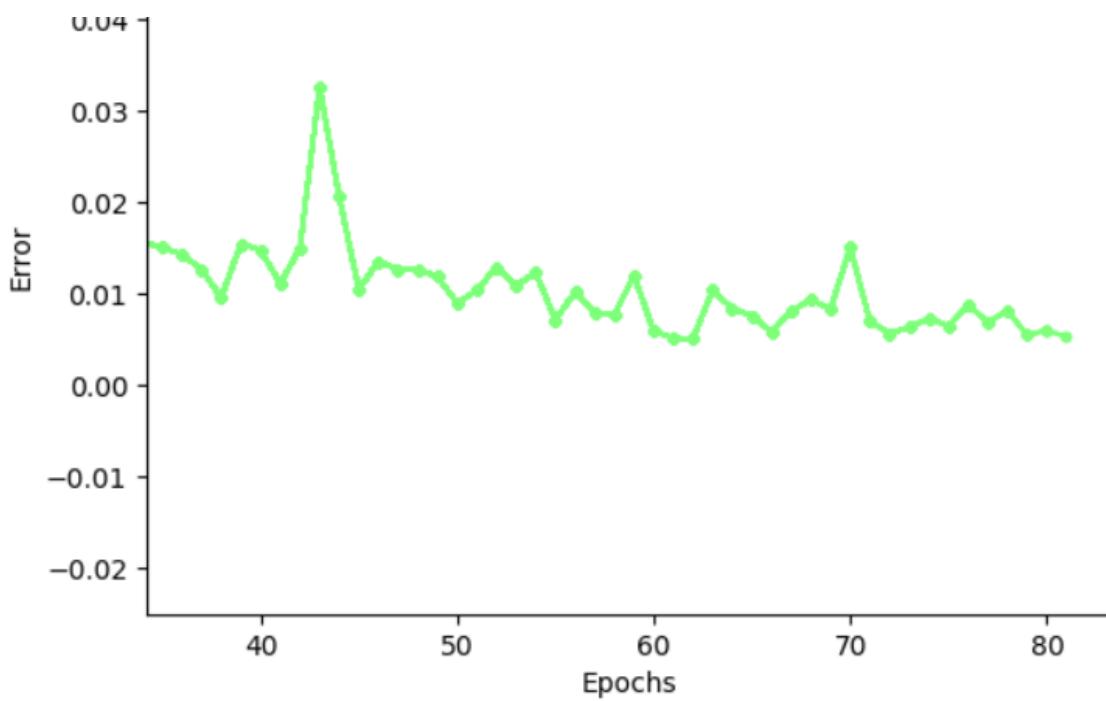


خطای مجموعه آموزش با آموزش بیشتر شبکه کاهش داشته است. بیشتر کاهش در ایپاک اول رخ داده است و در ایپاک‌های آخر میزان تغییرات خطا کم شده است و به مقدار ثابتی میل کرده است.

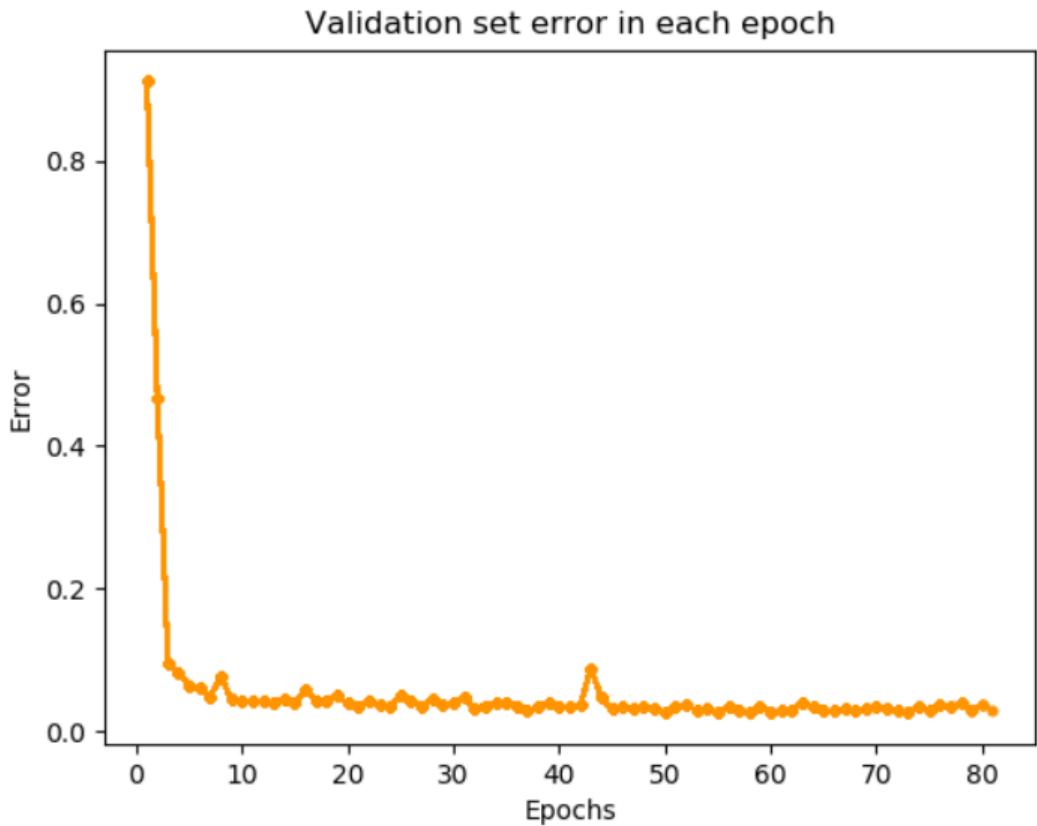
بخشی از نمودار بالا با بزرگنمایی:



بخشی دیگر از نمودار بالا با بزرگ‌نمایی بیشتر برای نشان دادن جزئیات بیشتر از تغییرات خطای:

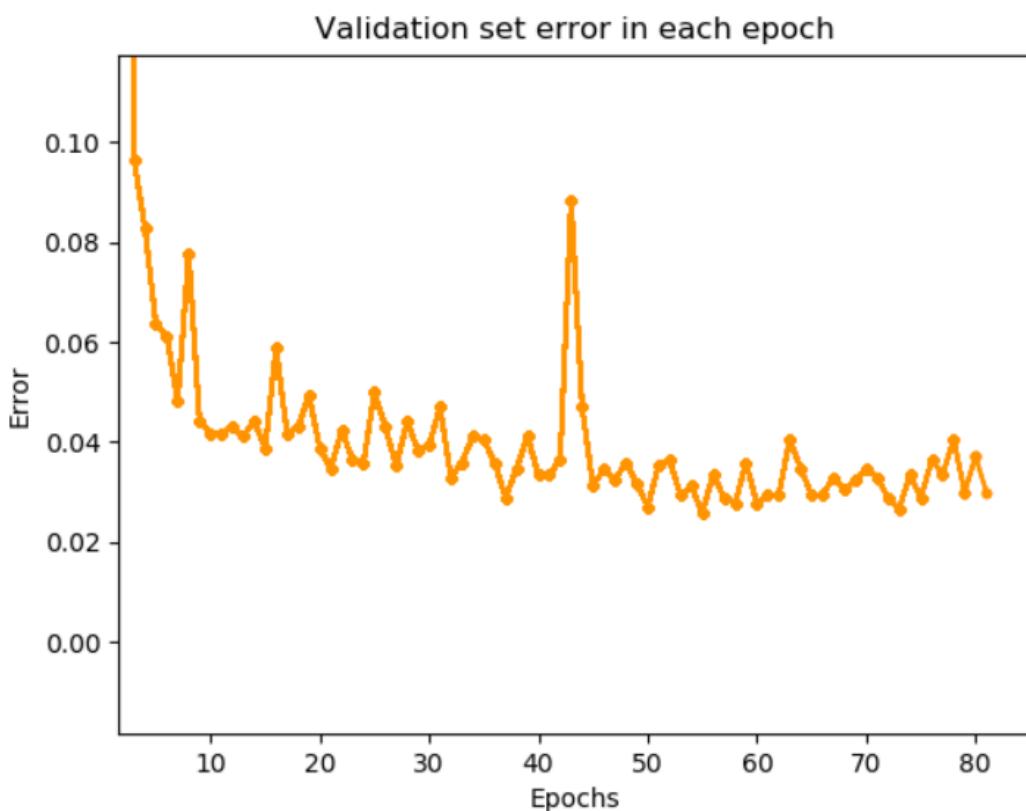


خطای مجموعه‌ی ارزیابی در هر ایپاک:



خطای ارزیابی در هر ایپاک با آموزش بیشتر کاهش داشته است. در ایپاک‌های اول بیشترین تغییرات را داشته‌ایم و در ایپاک‌های پایانی از میزان تغییرات کاسته شده است.

بخشی از نمودار بالا با بزرگنمایی:



این شبکه در فایل MLP-output-13.json ذخیره شده است.

نتیجه‌گیری

در بالا، ۴ شبکه‌ی عصبی مختلف را آزمودیم که پارامترهای مناسب هر کدام را با استفاده از کد MLP_best.py به دست آوردیم و سپس شبکه‌ها را با پارامترهای مناسب اجرا کردیم. هر چهار شبکه در هر کدام از لایه‌های نهان خود ۲۰ نرون دارند. خلاصه‌ی نتیجه‌های بالا را در جدول زیر مشاهده می‌کنید:

خطای مجموعه‌ی تست	تعداد لایه‌های نهان
۴,۶۴ درصد	۱
۳,۵۲ درصد	۳
۲,۸۸ درصد	۶
۳ درصد	۱۲

خطای مجموعه‌ی تست برای شبکه‌ی عصبی اول (یک لایه‌ی نهان با ۲۰ نرون) برابر با ۴,۶۴ درصد است، با توجه به نتایج شبکه‌های دیگر این شبکه قدرت کافی برای یادگیری را ندارد و شبکه bias است، با اضافه کردن دو لایه‌ی نهان دیگر شبکه عصبی دوم را ساختیم که سه لایه‌ی نهان دارد و هر کدام ۲۰ نرون دارند، خطای مجموعه‌ی تست آن برابر با ۳,۵۲ شد. اضافه کردن ۲ لایه به شبکه‌ی اول باعث شد پارامترهای بیشتری داشته باشیم و قدرت یادگیری مدل افزایش یابد، به همین دلیل خطای شبکه برای مجموعه‌ی تست نسبت به شبکه‌ی اول کاهش یافت. شبکه‌ی عصبی سوم دارای ۶ لایه‌ی نهان است که هر کدام ۲۰ نرون دارند، با افزودن تعداد لایه‌ها نسبت به شبکه‌ی قبلی پارامترهای بیشتری داریم و شبکه توانایی بیشتری برای یادگیری داده‌ها دارد و خطای مجموعه‌ی تست نسبت به شبکه‌ی قبل کاهش داشته است و به ۲,۸۸ درصد رسیده است. شبکه‌ی چهارم از ۱۲ لایه‌ی نهان تشکیل شده است که هر کدام از لایه‌ها ۲۰ نرون دارند، خطای این مدل نسبت به مدل قبل افزایش داشته است و بهبودی رخ نداده است، زیرا با زیاد شدن تعداد پارامترها قدرت و توانایی شبکه برای یادگیری داده‌ها از میزان لازم فراتر رفته است و شبکه شروع به حفظ کردن (Memorize) داده‌ها کرده است و Overfitting رخ داده است.

در صورتی که بخواهیم یک شبکه عمیق استفاده کنیم، می‌توان نتیجه گرفت که ابتدا کار را با یک لایه شروع می‌کنیم و به ترتیب لایه‌ها را افزایش می‌دهیم تا زمانی که افزایش تعداد لایه‌ها منجر به بهبود عملکرد شبکه می‌شوند این کار را ادامه می‌دهیم ولی بعد از مدتی این کار باعث Overfitting و افزایش خطا می‌شود، وقتی افزودن لایه‌ها باعث افزایش خطا شد متوقف می‌شویم و از افزودن لایه‌های جدید دوری می‌کنیم.

هر چه تعداد لایه‌ها بیشتر شده است شبکه بزرگ‌تر شده است و اجرای برنامه زمان بیشتر برای اجرا می‌گیرد.

برای مسیله‌ی رسم الخط هندی، شبکه‌ی عمیق بهتر است یا عریض

عنوان

با توجه به نتایج حاصل به نظر شما در مساله دسته‌بندی ارقام این رسم‌الخط، استفاده از شبکه عمیق مناسب‌تر است یا شبکه عریض؟ به طور کامل هم در مورد بهترین دقت حاصل از هر روش و هم در مورد تعداد وزنهای بهتریم مدل هر کدام از این روش‌ها بحث نمایید.

پاسخ

در دو قسمت قبلی سوال دقت تعدادی شبکه با یک لایه نهان و چند لایه‌ی نهان را بررسی کردیم، در زیر خلاصه‌ای از نتیجه‌ها را که در بالاتر کاملاً توضیح دادم را مشاهده می‌کنید:

شبکه‌های مختلف با یک لایه نهان:

تعداد نرون در لایه نهان	خطای مجموعه تست (درصد)
۵	% ۱۸,۸۲
۱۰	% ۷,۲۹
۳۰	% ۲,۹۴
۶۴	% ۲,۵۲
۱۴۰	% ۲,۴۱
۲۵۶	% ۲
۵۱۲	% ۲,۰۵

شبکه‌ها با تعداد لایه‌های نهان مختلف، هر لایه‌ی نهان ۲۰ نرون دارد:

تعداد لایه‌های نهان	خطای مجموعه‌ی تست
۱	۴,۶۴ درصد
۳	۳,۵۲ درصد
۶	۲,۸۸ درصد
۱۲	۳ درصد

در شبکه‌ها با یک لایه‌ی نهان، بهترین عملکرد با شبکه‌ی سه لایه‌ای که ۲۵۶ نرون در لایه‌ی نهان خود دارد به دست آمد که خطای مجموعه‌ی تست برای آن ۲ درصد شد.

در شبکه‌ها با چند لایه‌ی نهان، بهترین عملکرد توسط شبکه‌ای با ۶ لایه نهان (هر لایه نهان ۲۰ نرون دارد) به دست آمد.

بهترین عملکرد بین تمام شبکه‌ها توسط شبکه با یک لایه نهان که ۲۵۶ نرون دارد به دست می‌آید در نتیجه اگر عملکرد شبکه‌ی عریض بهتر بوده است.

همچنین اگر به نتیجه‌ها دقیق کنیم، عملکرد شبکه‌های عریض با بیش از ۳۰ نرون در لایه‌ی نهان خود از تمام شبکه‌های چند لایه بهتر بوده است، به طور مثال شبکه‌ی عصبی سه لایه با ۶۴ نرون در لایه‌ی میانی خود، خطای تست ۲,۵۲ درصد دارد و از $64 + 10 = 74$ نرون تشکیل شده است در حالی که بهترین نتیجه به دست آمده توسط شبکه‌های چند لایه توسط شبکه‌ی چند لایه با ۶ لایه‌ی نهان به دست آمده است که از $20 + 10 = 30$ نرون تشکیل شده است.

یک شبکه سه لایه با ۶۴ نرون در لایه‌ی نهان که در مجموع ۷۴ نرون (جز نرون‌های ورودی) دارد و حتی بهترین شبکه در بین شبکه‌ها با یک لایه‌ی نهان نیست، عملکرد بهتری نسبت به بهترین شبکه‌ی عصبی چند لایه با ۶ لایه‌ی نهان با مجموع ۱۳۰ نرون، دارد.

از آنجایی که شبکه‌های عصبی سه لایه با بیش از ۳۰ نرون در لایه‌ی نهان خود عملکرد بهتری نسبت به بهتری شبکه‌ی عصبی چند لایه دارند نتیجه می‌گیریم که شبکه‌های عریض عملکرد بهتری نسبت به شبکه‌های عمیق داشته‌اند.

بررسی تعداد وزن‌ها و بایاس‌ها:

تعداد وزن‌ها و بایاس‌ها در شبکه‌ی سه لایه با ۶۴ نرون در لایه نهان (۲,۵۲ درصد):

	تعداد نرون در لایه‌ی نهان ۱	تعداد نرون در لایه‌ی خروجی	مجموع بایاس‌ها و وزن‌ها
لایه‌ی ورودی	۶۴	۱۰	
تعداد وزن‌ها	$64 * 1024 = 65536$	$10 * 64 = 640$	66250
تعداد بایاس‌ها	64	10	

تعداد وزن‌ها و بایاس‌ها در شبکه‌ی سه لایه با ۲۵۶ نرون در لایه نهان (۲ درصد):

	تعداد نرون در لایه‌ی نهان ۱	تعداد نرون در لایه‌ی خروجی	مجموع بایاس‌ها و وزن‌ها
لایه‌ی ورودی	۲۵۶	۱۰	
تعداد وزن‌ها	$256 * 1024 = 262144$	$10 * 256 = 2560$	264970
تعداد بایاس‌ها	256	10	

تعداد وزن‌ها و بایاس‌ها در شبکه‌ی ۸ لایه با ۶ لایه‌ی نهان (۲,۸۸ درصد):

	تعداد نرون لایه نهان ۱	تعداد نرون لایه نهان ۲	تعداد نرون لایه نهان ۳	تعداد نرون لایه نهان ۴	تعداد نرون لایه نهان ۵	تعداد نرون لایه نهان ۶	تعداد نرون لایه	تعداد نرون لایه	خروجی
	۱۰۲۴	۲۰	۲۰	۲۰	۲۰	۲۰	۲۰	۱۰	
تعداد وزن‌ها		۲۰۴۸۰	۴۰۰	۴۰۰	۴۰۰	۴۰۰	۴۰۰	۲۰۰	۲۲۸۱۰
تعداد بایاس‌ها		۲۰	۲۰	۲۰	۲۰	۲۰	۲۰	۱۰	

تعداد وزن‌ها و بایاس‌ها در شبکه‌های یک لایه با ۶۴ نرون در لایه نهان و ۲۵۶ نرون در لایه نهان بسیار بیشتر از شبکه‌ی ۸ لایه با ۶ لایه نهان (هر لایه‌ی نهان ۲۰ نرون دارد) است. به همین دلیل آن دو شبکه‌ی در هنگام آموزش زمان بیشتر نسبت به شبکه‌ی چند لایه می‌برند.

شبکه‌ی سه لایه با ۶۴ نرون در لایه‌ی نهان با اینکه بهترین شبکه ۳ لایه از نظر دقت نیست، ولی دقتی بهتر از تمام شبکه‌های چند لایه دارد و زمان آموزش آن با زمان آموزش بهترین شبکه‌ی چند لایه تقریباً یکسان است، پس با در نظر گرفتن تعداد وزن‌ها و بایاس‌ها باز شبکه‌های عریض با تعداد نرون مناسب، برای این دیتاست مناسب‌تر اند.

ولی اگر تفاوت یک و دو درصدی در میزان خطا برایمان مهم نباشد و سرعت آموزش و تست، برایمان اهمیت بیشتری داشته باشد، شبکه‌های چند لایه و عمیق بهتر اند زیرا پارامترهای کمتری دارند و سرعت‌شان بسیار بیشتر است.

هر چه تعداد لایه‌ها بیشتر می‌شود شبکه‌ی عصبی سطح بیشتری از abstraction را پوشش می‌دهند، به عنوان مثال وقتی یک شبکه‌ی عمیق convolution را برای تشخیص Object ها را بررسی کنیم، مثلاً می‌بینیم که لایه‌ی اول یادگرفته است که لبه‌ها را تشخیص دهد، لایه‌ی دوم مجموعه‌ای از لبه‌ها (شکل)، لایه‌ی سوم مجموعه‌ای از شکل‌ها و ولی وقتی فضای دیتاست به گونه‌ای است نیاز به فیچرهای پیچیده‌تر نیست و می‌توان با یک لایه با دقت و زمان مناسب، مساله را حل کرد از شبکه‌های عریض استفاده می‌کنیم.

بررسی دقت دسته‌بندی و سرعت همگرایی در سه روش The Levenberg-
batch, steepest descent, The Levenberg-Marguardt

عنوان
بررسی دقت دسته‌بندی و سرعت همگرایی در سه روش The levenberge- و Steepest Descent batch
Marguardt

شرایط آزمایش

برای دیدن تفاوت سرعت همگرایی آموزش شبکه عصبی در حالت **steepest descent** و **batch** دو شبکه‌ی عصبی یکسان با وزن‌ها و بایاس‌های اولیه‌ی یکسان و پارامترها و ساختار یکسان را آموزش می‌دهیم. از آنجایی که در **best_MLP.py** ضریب یادگیری آپدیت می‌شود با استفاده از کد **steepest descent** بهترین پارامترها را برای حالت **batch** به دست آورده‌ام و هر دو شبکه را با پارامترهای یکسان اجرا می‌کنم و شرط اتمام آموزش هم کمتر شدن میزان تابع هزینه از حد مشخصی است، از آنجایی که هر دو شبکه در ابتدا وزن‌ها و بایاس‌های یکسانی دارند و تابع هزینه‌شان قبل از شروع آموزش یکسان است به خوبی می‌توانیم تفاوت در سرعت را ببینیم.

برای دیدن تفاوت در خطای دو روش دو شبکه‌ی سه و چهار را ایجاد می‌کنم که هر دو سه لایه با ۶۴ نرون در لایه‌ی نهان را دارند. شبکه‌ی سه با **steepest descent** کار می‌کند و شبکه‌ی چهار با **batch** و پارامترهای مناسب هر کدام را با استفاده از کد **best_MLP** پیدا می‌کنم و شرط خاتمه آموزش حداکثر تعداد ایپاک است:

فایل ورودی شبکه‌ی اول - **:input-20.json**

input-20.json

```
1  {
2      "digits_letters_digits": "digits",
3      "ratio_of_train_set": 0.80,
4      "ratio_of_valid_set": 0.10,
5      "ratio_of_test_set": 0.10,
6      "learning_rate": 0.5,
7      "max_epochs": 1500,
8      "cut_cost": 0.026,
9      "random_state": 2,
10     "K_fold": 5,
11     "neurons_layers": [1024, 64, 10],
12     "type_layers": ["tanh", "sigmoid"],
13     "type_of_cost": "MSE",
14     "mode": "batch"
15 }
```

شبکه‌ی شماره اول: از دیتاست عده‌ها استفاده می‌شود، ۸۰ درصد دیتاست برای آموزش است، ۱۰ درصد دیتاست برای ارزیابی است، ۱۰ درصد دیتاست برای تست است، ضریب یادگیری برابر با ۰,۵ است، حداقل تعداد ایپاک‌ها ۱۵۰۰ است، در صورتی که مقدارتابع هزینه کمتر از ۰.۰۲۶ شود آموزش متوقف می‌شود، تولید عده‌های تصادفی با عدد ۲ بذرپاشی می‌شود، تعداد فیچرها ۱۰۲۴ تا است، لایه نهان اول ۶۴ نرون دارد. لایه خروجی ۱۰ نرون دارد. تابع فعال سازی لایه نهان اول tanh است و تابع فعال سازی لایه خروجی sigmoid است، تابع هزینه MSE است، از روش batch 5-fold cross validation استفاده شده است. از

فایل ورودی شبکه‌ی دوم - input-21.json

input-21.json

```
1  {
2      "digits_letters_digits": "digits",
3      "ratio_of_train_set": 0.80,
4      "ratio_of_valid_set": 0.10,
5      "ratio_of_test_set": 0.10,
6      "learning_rate": 0.5,
7      "max_epochs": 1500,
8      "cut_cost": 0.026,
9      "random_state": 2,
10     "K_fold": 5,
11     "neurons_layes": [1024, 64, 10],
12     "type_layers": ["tanh", "sigmoid"],
13     "type_of_cost": "MSE",
14     "mode": "steepest"
15 }
```

شبکه‌ی شماره دوم: از دیتاست عده‌ها استفاده می‌شود، ۸۰ درصد دیتاست برای آموزش است، ۱۰ درصد دیتاست برای ارزیابی است، ۱۰ درصد دیتاست برای تست است، ضریب یادگیری برابر با ۰.۵ است، حداکثر تعداد ایپاک‌ها ۱۵۰۰ است، در صورتی که مقدارتابع هزینه کمتر از ۰.۰۲۶ شود آموزش متوقف می‌شود، تولید عده‌های تصادفی با عدد ۲ بذرپاشی می‌شود، تعداد فیچرها ۱۰۲۴ نا است، لایه نهان اول ۶۴ نرون دارد. لایه خروجی ۱۰ نرون دارد. تابع فعال سازی لایه‌ی نهان اول tanh است و تابع فعال سازی لایه‌ی خروجی sigmoid است، تابع هزینه MSE است، از روش steepest descent 5-fold cross validation استفاده شده است. از روشنودی شبكه‌ی سوم -

input-22.json

```

input-22.json
1  {
2      "digits_letters_digits": "digits",
3      "ratio_of_train_set":0.80,
4      "ratio_of_valid_set":0.10,
5      "ratio_of_test_set":0.10,
6      "learning_rate":0.01,
7      "max_epochs":20,
8      "cut_cost": -1,
9      "random_state":2,
10     "K_fold": 5,
11     "neurons_layers": [1024, 64, 10],
12     "type_layers": ["tanh", "sigmoid"],
13     "type_of_cost": "MSE",
14     "mode": "steepest"
15 }

```

شبکه‌ی شماره سوم: از دیتاست عده‌ها استفاده می‌شود، ۸۰ درصد دیتاست برای آموزش است، ۱۰ درصد دیتاست برای ارزیابی است، ۱۰ درصد دیتاست برای تست است، ضریب یادگیری برابر با ۰.۰۱ است، حداکثر تعداد ایپاک‌ها ۲۰ است، شرط قطع شدن آموزش در صورت کمتر شدن تابع هزینه از یک حد مشخصی چک نمی‌شود، تولید عده‌های تصادفی با عدد ۲ بذرپاشی می‌شود، تعداد فیچرها ۱۰۲۴ تا است، لایه نهان اول ۶۴ نرون دارد. لایه خروجی ۱۰ نرون دارد. تابع فعال سازی لایه نهان اول tanh است و تابع فعال سازی لایه خروجی sigmoid است، تابع هزینه MSE است، از روش steepest descent استفاده شده است. از ۵-fold cross validation استفاده می‌شود.

فایل ورودی شبکه‌ی چهارم :input-23.json

input-23.json

```
1  {
2      "digits_letters_digits": "digits",
3      "ratio_of_train_set":0.80,
4      "ratio_of_valid_set":0.10,
5      "ratio_of_test_set":0.10,
6      "learning_rate":0.5,
7      "max_epochs":10000,
8      "cut_cost": -1,
9      "random_state":2,
10     "K_fold": 5,
11     "neurons_layes": [1024, 64, 10],
12     "type_layers": ["tanh", "sigmoid"],
13     "type_of_cost": "MSE",
14     "mode": "batch"
15 }
```

شبکه‌ی شماره چهارم: از دیتاست عده‌ها استفاده می‌شود، ۸۰ درصد دیتاست برای آموزش است، ۱۰ درصد دیتاست برای ارزیابی است، ۱۰ درصد دیتاست برای تست است، ضریب یادگیری برابر با ۰.۵ است، حداکثر تعداد ایپاک‌ها ۱۰۰۰۰ است، شرط قطع شدن آموزش در صورت کمتر شدن تابع هزینه از یک حد مشخصی چک نمی‌شود، تولید عده‌های تصادفی با عدد ۲ بذرپاشی می‌شود، تعداد فیچرها ۱۰۲۴ تا است، لایه نهان اول ۶۴ نرون دارد. لایه خروجی ۱۰ نرون دارد. تابع فعال سازی لایه نهان اول tanh است و تابع فعال سازی لایه خروجی sigmoid است، تابع هزینه MSE است، از روش batch استفاده شده است. از 5-fold cross validation استفاده می‌شود.

نتایج آزمایش

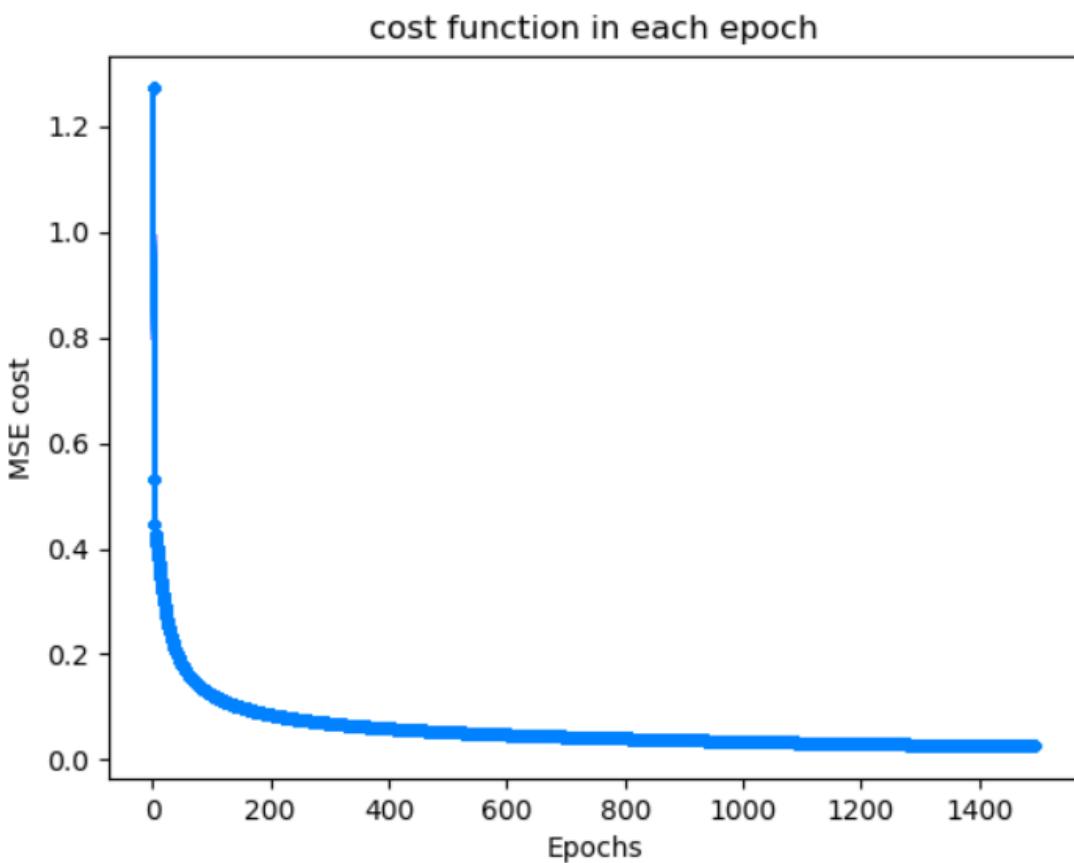
نتیجه‌های شبکه عصبی اول:

مقدار تابع هزینه در آخرین ایپاک، خطای آموزش، خطای ارزیابی، خطای تست :

```
Iter 1492/1500, Cost 0.02601635992343362 ...
Iter 1493/1500, Cost 0.026002594810525408 ...
Iter 1494/1500, Cost 0.025988843329713258 ...
=====
Train Error: 0.020073529411764705
Validation Error: 0.031176470588235295
Test Error: 0.03294117647058824
=====
|
```

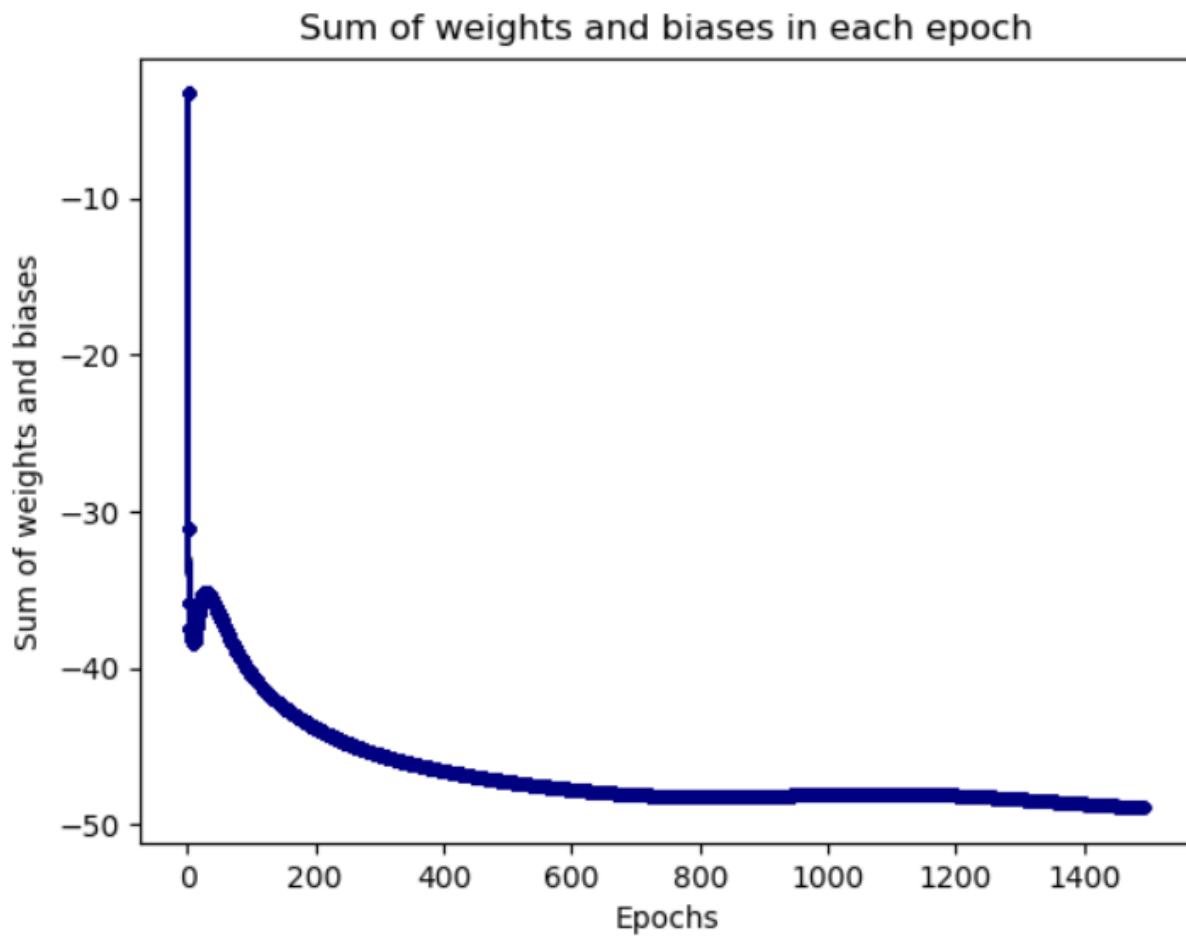
مقدار تابع هزینه بعد از 1494 ایپاک از 1.27 به ۰.۰۲۵ رسیده است. خطای مجموعه آموزش برابر ۰.۲ درصد است، خطای مجموعه ارزیابی برابر با ۳.۱ درصد است و خطای مجموعه تست برابر با ۳.۲۹ درصد است.

مقدار تابع هزینه در هر ایپاک:



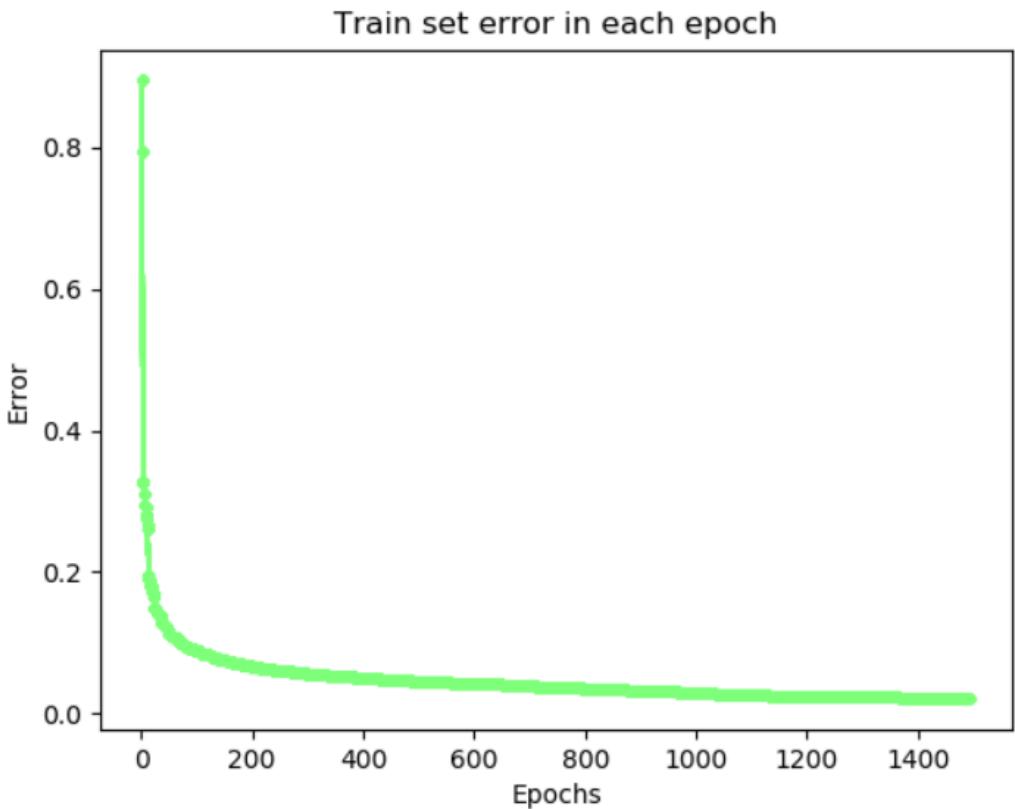
در طی ۱۴۹۴ ایپاک تابع هزینه‌ی MSE از ۰,۰۲۵ به ۱,۲۷ رسیده است که میزان کاهش چشم‌گیر است. مقدار تابع هزینه سیر نزولی داشته است و بیشترین تغییرات در زمان انجام ایپاک اول بوده است و هر چه جلوتر رفته‌ایم در ایپاک‌های بعد از میزان تغییرات کاسته شده است و با روند کند تری تابع هزینه کم شده است.

مجموع وزن‌ها و بایاس‌ها در هر ایپاک:



در ایپاک‌های آخر میزان تغییرات مجموعه وزن‌ها و بایاس‌ها کم شده است و تقریباً در آن محدوده باقی می‌ماند و تقریباً در چند ایپاک آخر مقدار مجموع ثابت مانده است. زیرا هر چه جلوتر می‌رویم معمولاً گرادیان‌ها کوچک‌تر می‌شوند و تغییرات وزن‌ها نسبت به مراحل قبل کمتر می‌شود.

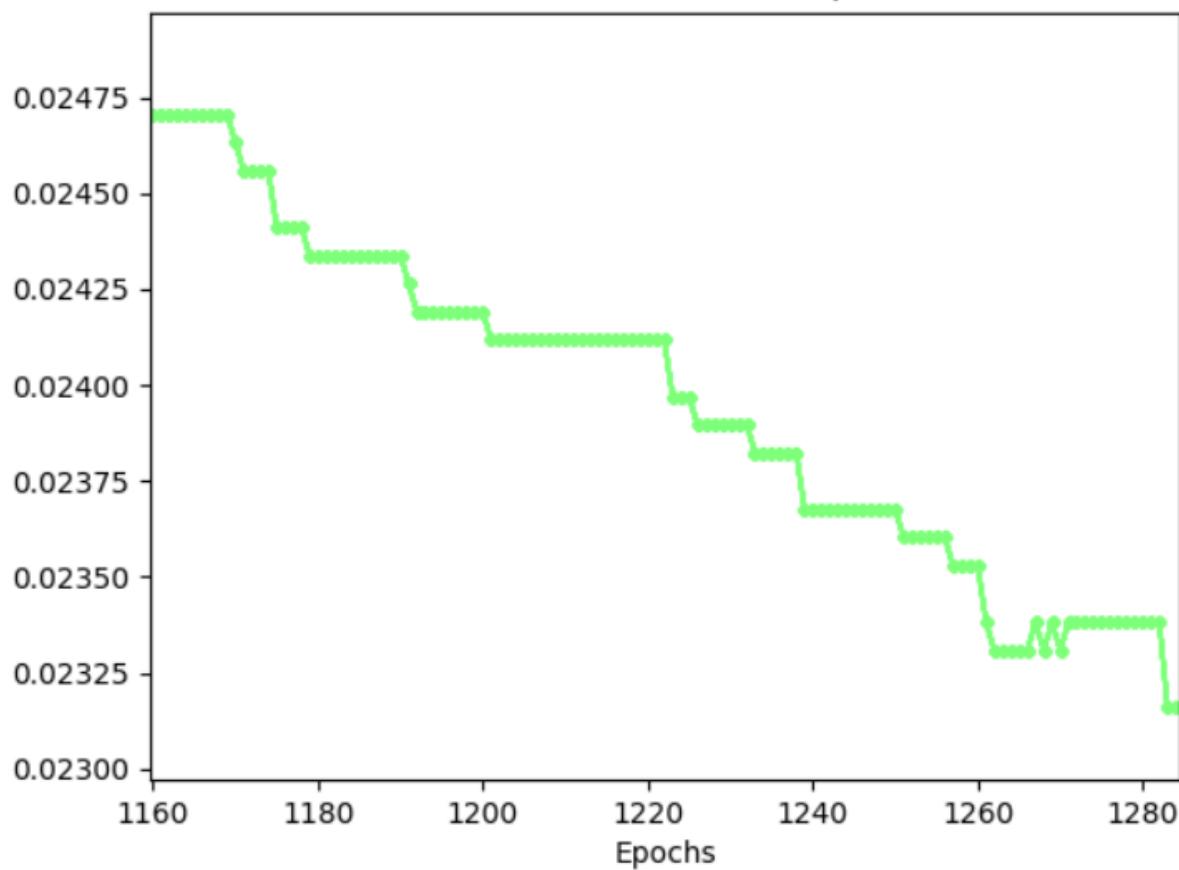
خطای مجموعه‌ی آموزش در هر ایپاک:



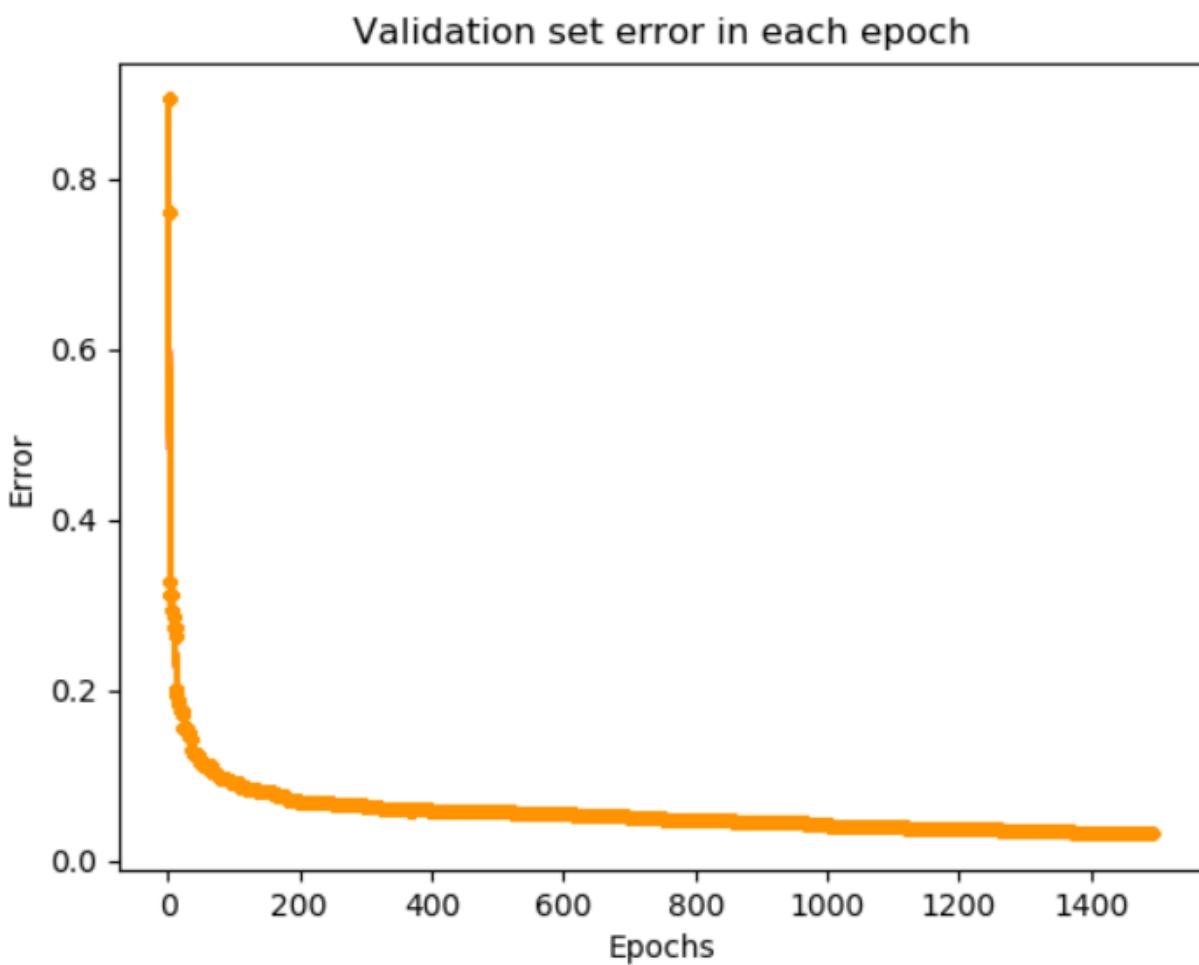
خطای مجموعه آموزش با آموزش بیشتر شبکه کاهش داشته است. بیشتر کاهش در ایپاک اول رخ داده است و در ایپاک های آخر میزان تغییرات خطای کم شده است و به مقدار ثابتی میل کرده است.

بخشی از نمودار بالا با بزرگنمایی بیشتر:

Train set error in each epoch



خطای مجموعه‌ی ارزیابی در هر ایپاک:



خطای ارزیابی در هر ایپاک با آموزش بیشتر کاهش داشته است. در ایپاک‌های اول بیشترین تغییرات را داشته‌ایم و در ایپاک‌های پایانی از میزان تغییرات کاسته شده است.

بخشی از نمودار بالا با بزرگنمایی بیشتر.

این شبکه در فایل MLP-output-20.json ذخیره شده است.

نتیجه‌های شبکه عصبی دوم:

مقدار تابع هزینه در آخرین ایپاک، خطای آموزش، خطای ارزیابی، خطای تست :

```
Calculating 5-fold in a separate thread has been began.  
Iter 0/1500, Cost 1.272158268971262 ...  
Iter 1/1500, Cost 0.16293858498276295 ...  
Iter 2/1500, Cost 0.07921938791370853 ...  
Iter 3/1500, Cost 0.06643056450075412 ...  
Iter 4/1500, Cost 0.06079560723780589 ...  
Iter 5/1500, Cost 0.04770889694030762 ...  
Iter 6/1500, Cost 0.03597537335987489 ...  
Iter 7/1500, Cost 0.034819399282898 ...  
Iter 8/1500, Cost 0.03025891640459626 ...  
Iter 9/1500, Cost 0.027392823485139838 ...  
Iter 10/1500, Cost 0.024265855210267608 ...  
=====V=====  
Train Error: 0.02985294117647059  
Validation Error: 0.04  
Test Error: 0.03705882352941176
```

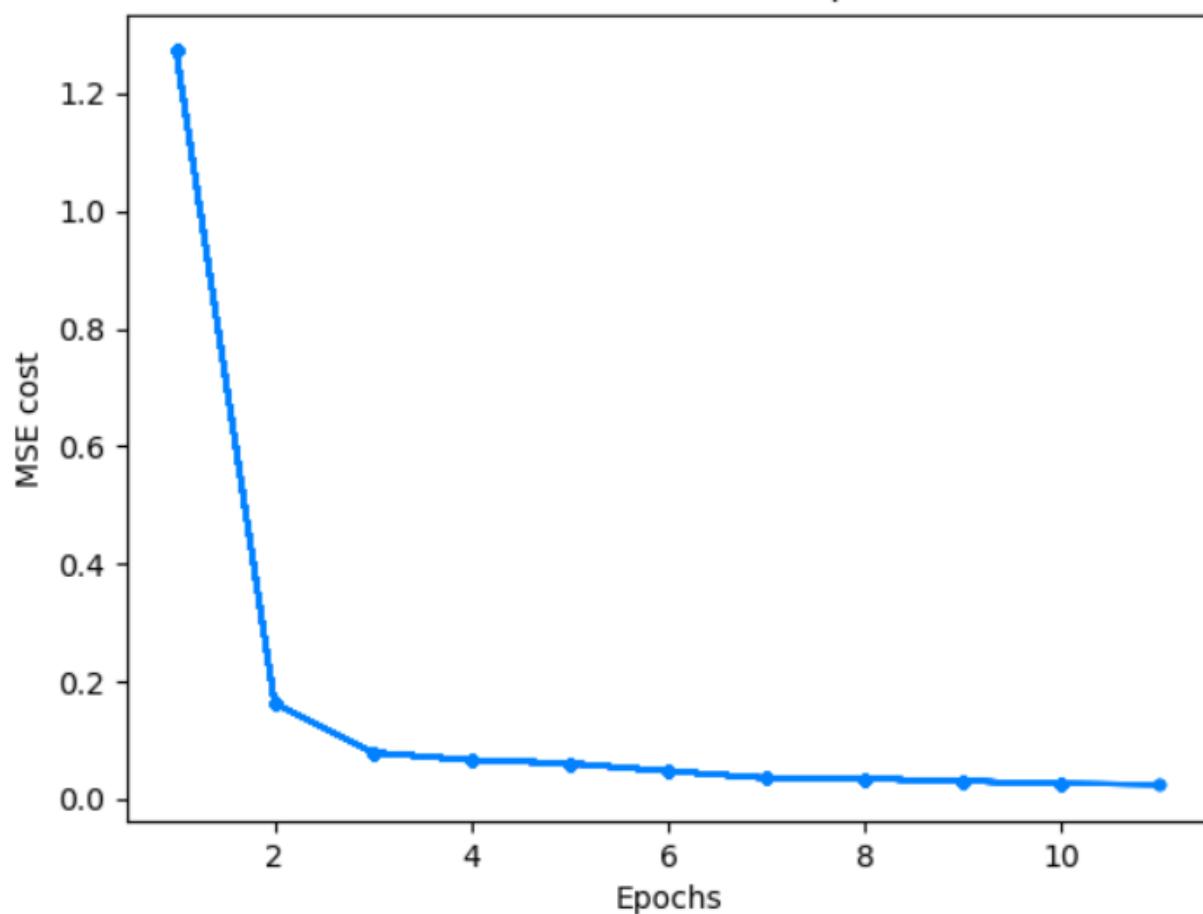
5-fold error is: 0.03455882352941177

=====^=====

مقدار تابع هزینه بعد از 11 ایپاک از 1.27 به ۰,۰۲۴ رسیده است در حالی که در حالت batch این کار ۱۴۹۴ ایپاک لازم داشت. خطای مجموعه آموزش برابر ۲,۹ درصد است، خطای مجموعه ارزیابی برابر با ۴ درصد است و خطای مجموعه تست برابر با ۳,۷ درصد است. خطای 5-fold CV برابر با ۳,۴ درصد است.

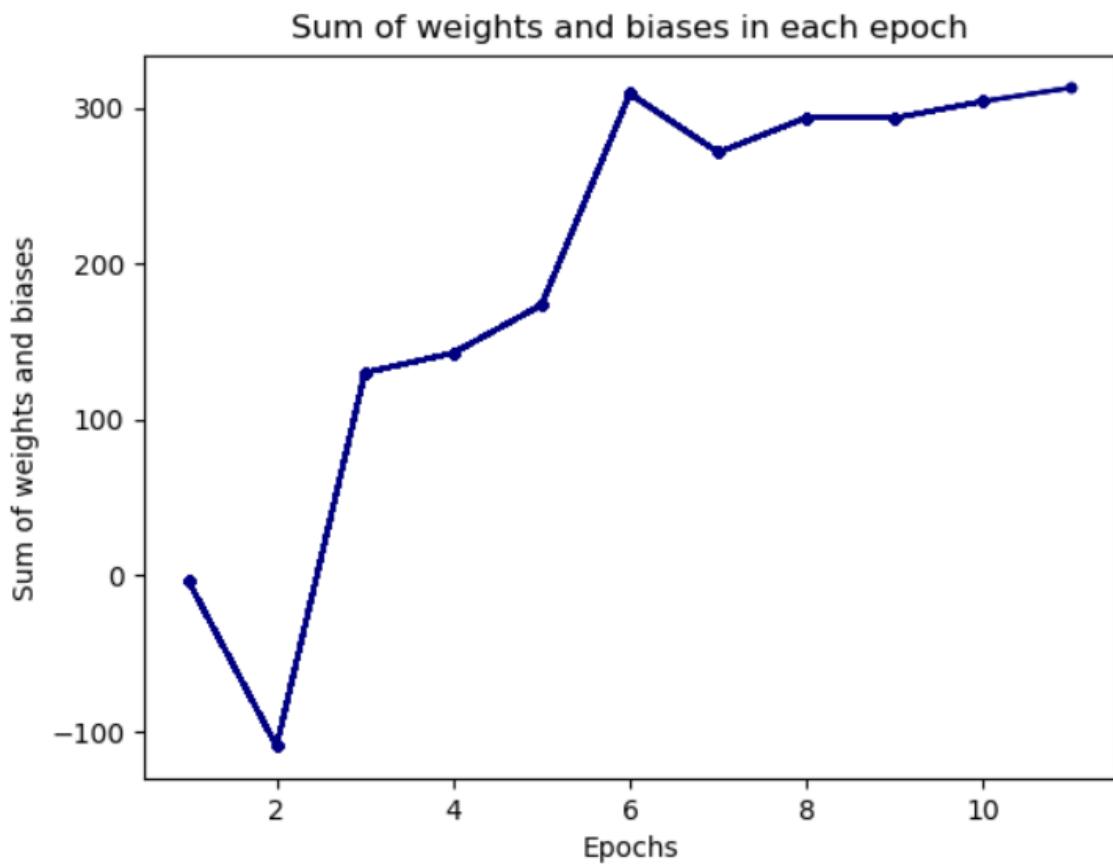
مقدار تابع هزینه در هر ایپاک:

cost function in each epoch



در طی ۱۱ ایپاک تابع هزینه‌ی MSE از ۰.۲۴ به ۰.۰۲۷ رسیده است که میزان کاهش چشم‌گیر است. مقدار تابع هزینه سیر نزولی داشته است و بیشترین تغییرات در زمان انجام ایپاک اول بوده است و هر چه جلوتر رفته‌ایم در ایپاک‌های بعد از میزان تغییرات کاسته شده است و با روند کند تری تابع هزینه کم شده است.

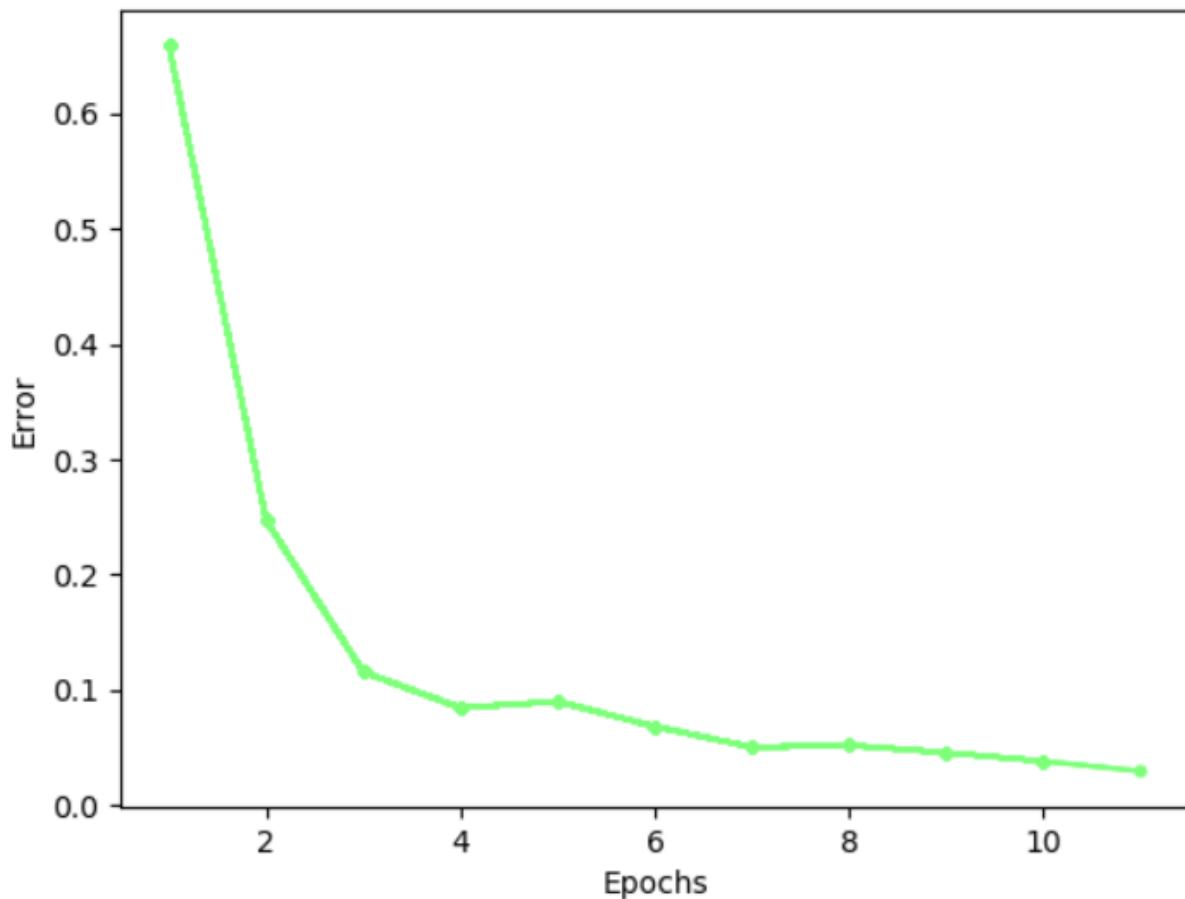
مجموع وزن‌ها و بایاس‌ها در هر ایپاک:



در ایپاک‌های آخر میزان تغییرات مجموعه وزن‌ها و بایاس‌ها کم شده است و تقریباً در آن محدوده باقی می‌ماند و تقریباً در چند ایپاک آخر مقدار مجموع ثابت مانده است. زیرا هر چه جلوتر می‌رویم معمولاً گرادیان‌ها کوچک‌تر می‌شوند و تغییرات وزن‌ها نسبت به مراحل قبل کمتر می‌شود.

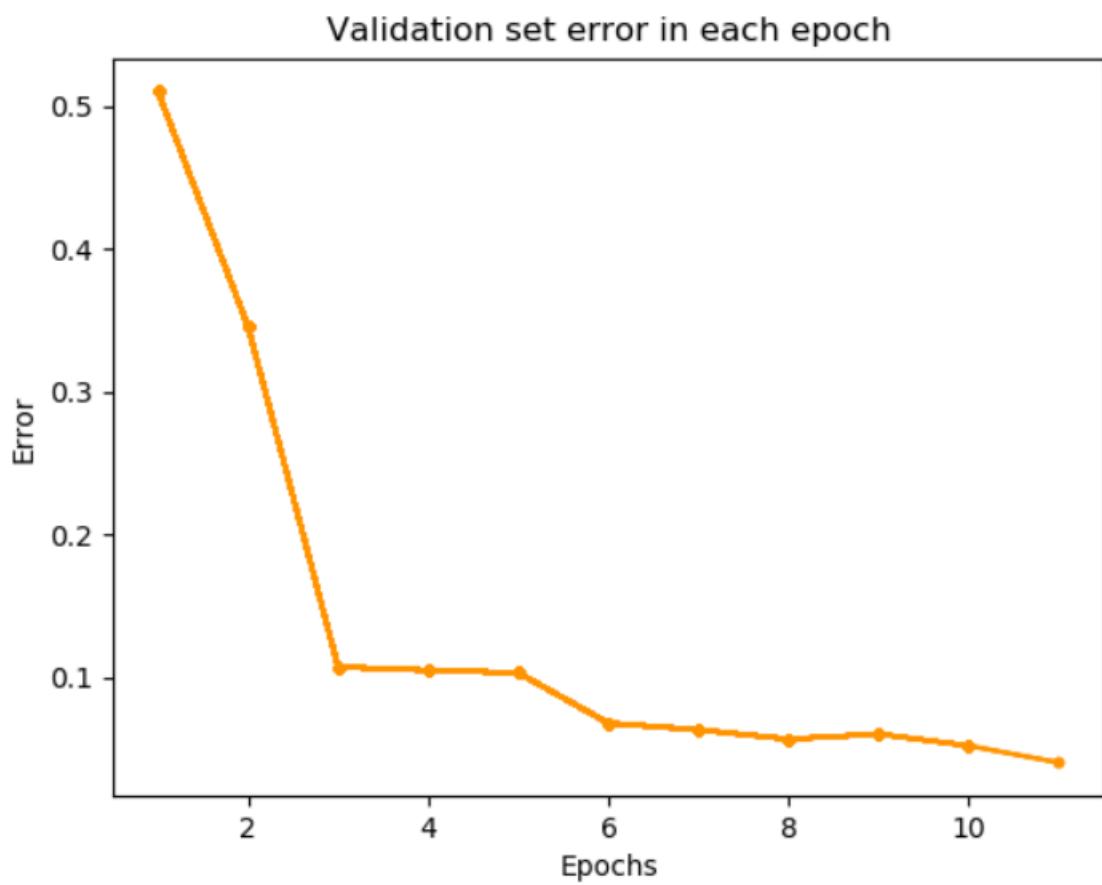
خطای مجموعه‌ی آموزش در هر ایپاک:

Train set error in each epoch



خطای مجموعه آموزش با آموزش بیشتر شبکه کاهش داشته است. بیشتر کاهش در ایپاک اول رخ داده است و در ایپاک های آخر میزان تغییرات خطا کم شده است و به مقدار ثابتی میل کرده است.

خطای مجموعه ارزیابی در هر ایپاک:



خطای ارزیابی در هر ایپاک با آموزش بیشتر کاهش داشته است. در ایپاک‌های اول بیشترین تغییرات را داشته‌ایم و در ایپاک‌های پایانی از میزان تغییرات کاسته شده است.
بخشی از نمودار بالا با بزرگنمایی بیشتر.

این شبکه در فایل MLP-output-21.json ذخیره شده است.

نتیجه‌های شبکه عصبی سوم:

مقدار تابع هزینه در آخرین ایپاک، خطای آموزش، خطای ارزیابی، خطای تست :

```

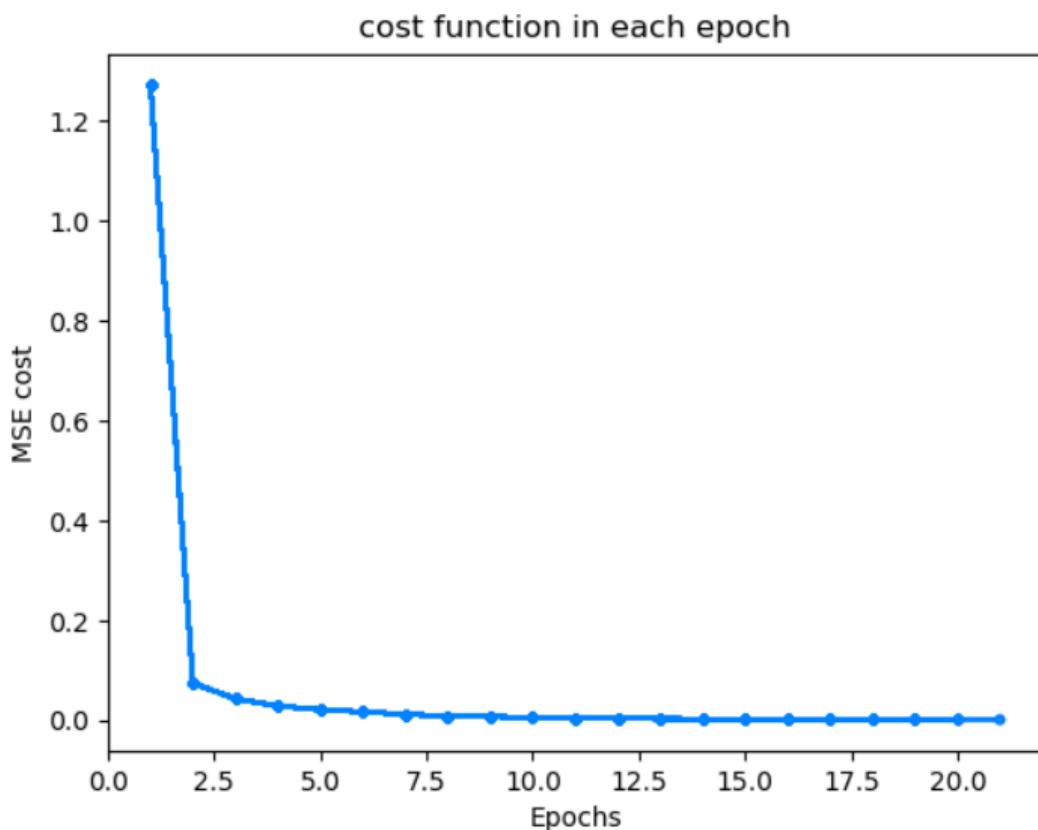
Iter 10/20, Cost 0.0049645236698/4122 ...
Iter 11/20, Cost 0.004474220217821746 ...
Iter 12/20, Cost 0.0036628605332573437 ...
Iter 13/20, Cost 0.003323970932066108 ...
Iter 14/20, Cost 0.0031019541620117344 ...
Iter 15/20, Cost 0.0028144414960108023 ...
Iter 16/20, Cost 0.002809814960722804 ...
Iter 17/20, Cost 0.0025149497828892515 ...
Iter 18/20, Cost 0.0023029479371471434 ...
Iter 19/20, Cost 0.0021693850919035367 ...
Iter 20/20, Cost 0.0021315804940415556 ...
=====
V=====

Train Error: 0.0026470588235294116
Validation Error: 0.020588235294117647
Test Error: 0.02

5-fold error is: 0.005514705882352941
=====
^=====
```

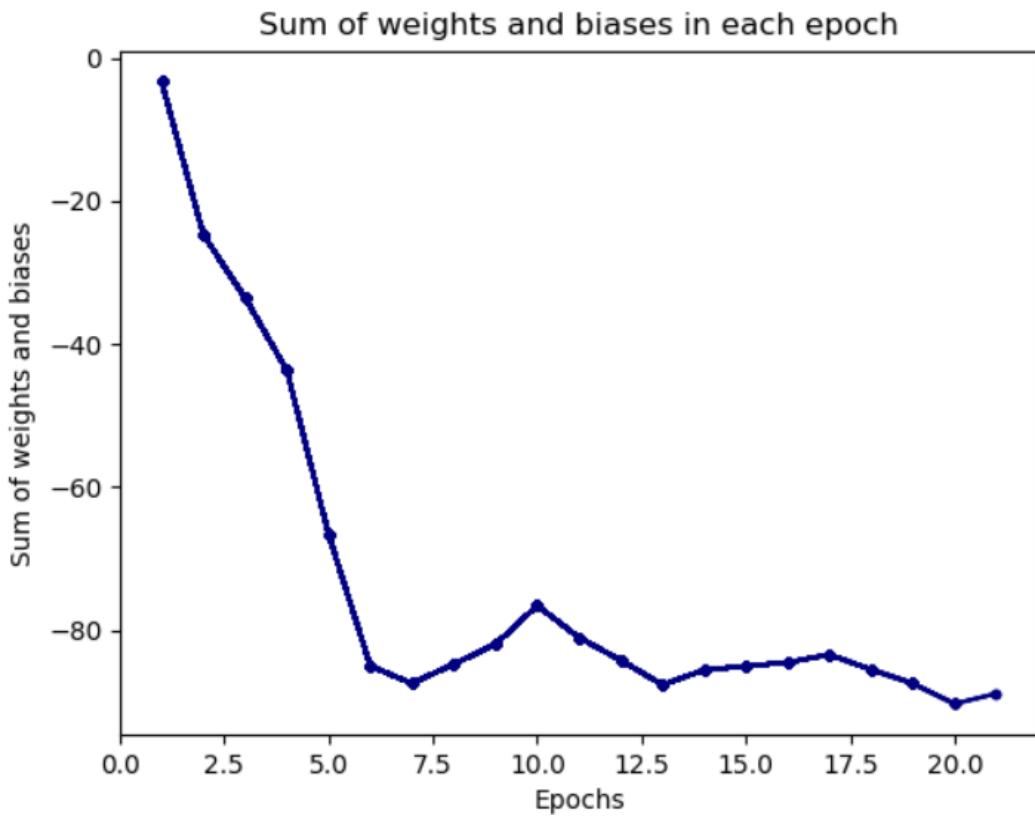
مقدار تابع هزینه بعد از 20 ایپاک از 1.27 به ۰،۰۲۱ رسیده است. خطای مجموعه آموزش برابر ۰.۰۲ درصد است، خطای مجموعه ارزیابی برابر با ۰،۰۵ درصد است و خطای مجموعه تست برابر با ۰،۰۲ درصد است. خطای 5-fold برابر با ۰.۵ درصد است.

مقدار تابع هزینه در هر ایپاک:



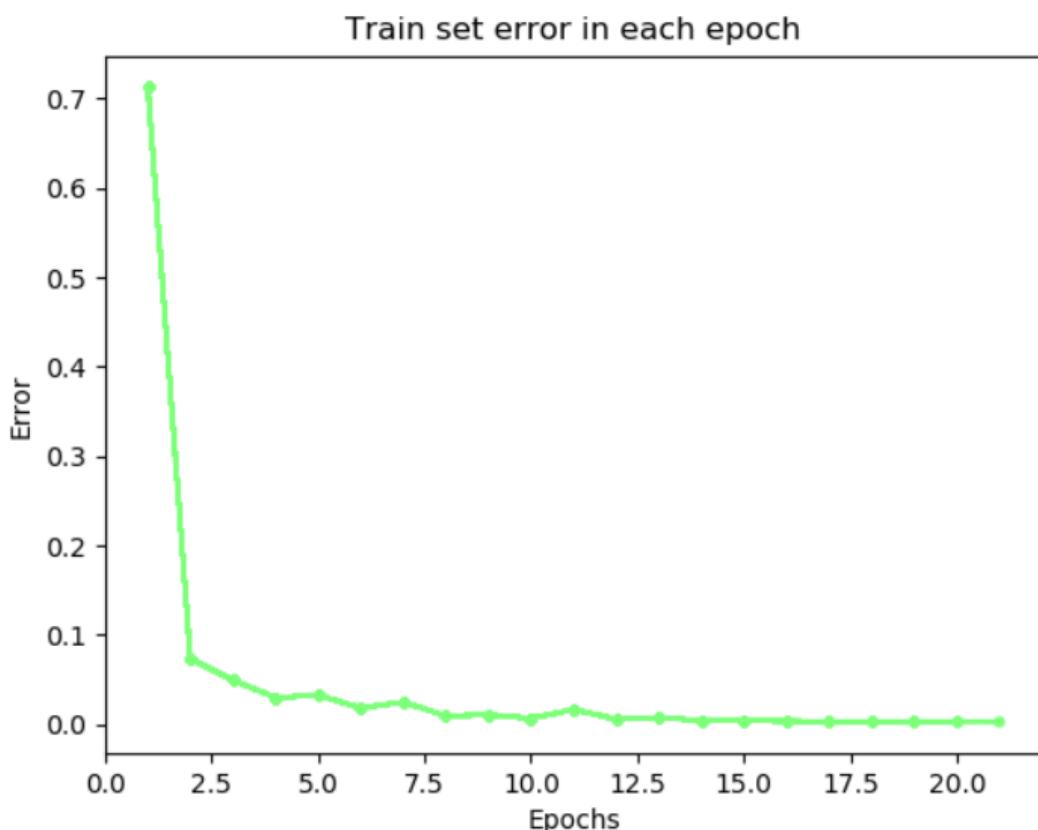
در طی ۲۰ ایپاک تابع هزینه‌ی MSE از ۱,۲۷ به ۰,۰۲۱ رسیده است. که میزان کاهش چشم‌گیر است. مقدار تابع هزینه سیر نزولی داشته است و بیشترین تغییرات در زمان انجام ایپاک اول بوده است و هر چه جلوتر رفته‌ایم در ایپاک‌های بعد از میزان تغییرات کاسته شده است و با روند کندتری تابع هزینه کم شده است.

مجموع وزن‌ها و بایاس‌ها در هر ایپاک:



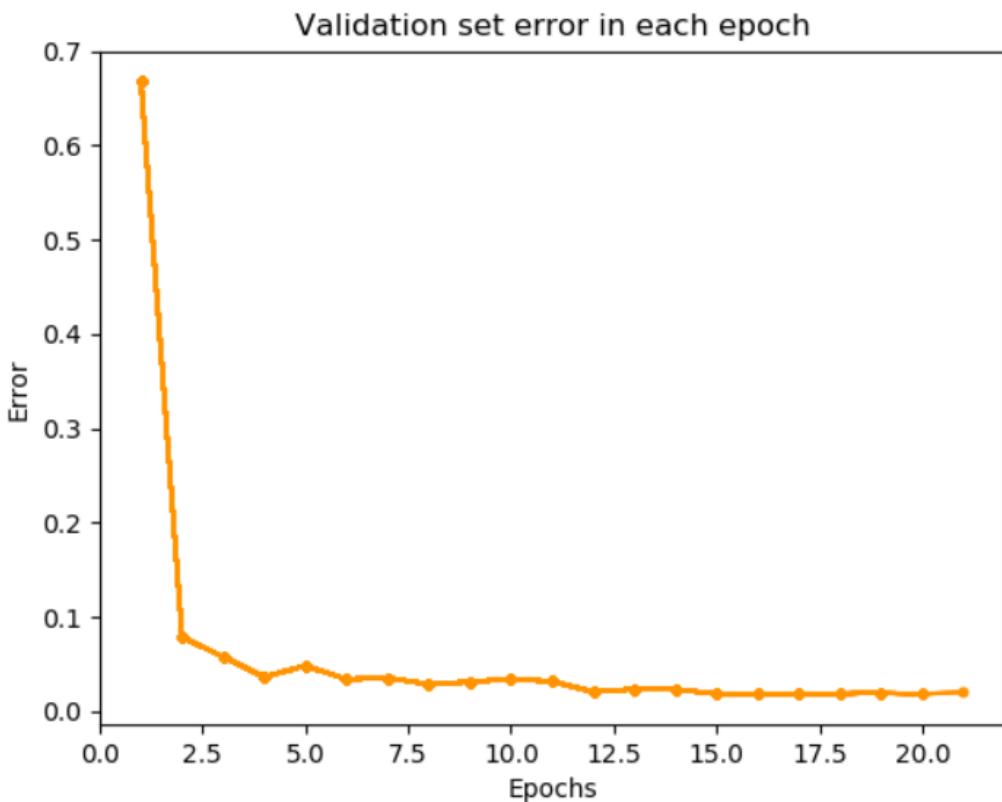
در ایپاک‌های آخر میزان تغییرات مجموعه وزن‌ها و بایاس‌ها کم شده است و تقریباً در آن محدوده باقی می‌ماند و تقریباً در چند ایپاک آخر مقدار مجموع ثابت مانده است. زیرا هر چه جلوتر می‌رویم معمولاً گرادیان‌ها کوچک‌تر می‌شوند و تغییرات وزن‌ها نسبت به مراحل قبل کمتر می‌شود.

خطای مجموعه‌ی آموزش در هر ایپاک:



خطای مجموعه آموزش با آموزش بیشتر شبکه کاهش داشته است. بیشتر کاهش در ایپاک اول رخ داده است و در ایپاک های آخر میزان تغییرات خطای کم شده است و به مقدار ثابتی میل کرده است.

خطای مجموعه ارزیابی در هر ایپاک:



خطای ارزیابی در هر ایپاک با آموزش بیشتر کاهش داشته است. در ایپاکهای اول بیشترین تغییرات را داشته‌ایم و در ایپاکهای پایانی از میزان تغییرات کاسته شده است.

بخشی از نمودار بالا با بزرگنمایی بیشتر.

این شبکه در فایل MLP-output-22.json ذخیره شده است.

نتیجه‌های شبکه عصبی چهارم:

مقدار تابع هزینه در آخرین ایپاک، خطای آموزش، خطای ارزیابی، خطای تست :

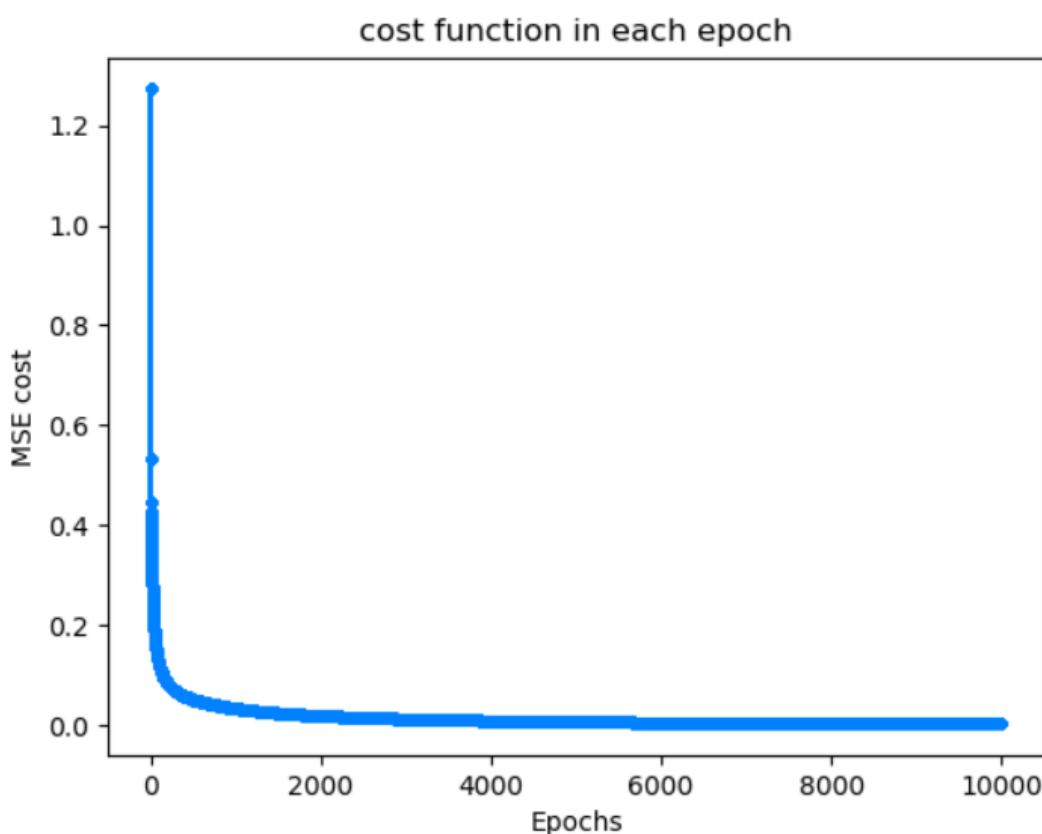
```

Iter 9999/10000, Cost 0.003936511022443483 ...
Iter 10000/10000, Cost 0.003936097591796118 ...
=====
Train Error: 0.0033823529411764705
Validation Error: 0.02235294117647059
Test Error: 0.023529411764705882

```

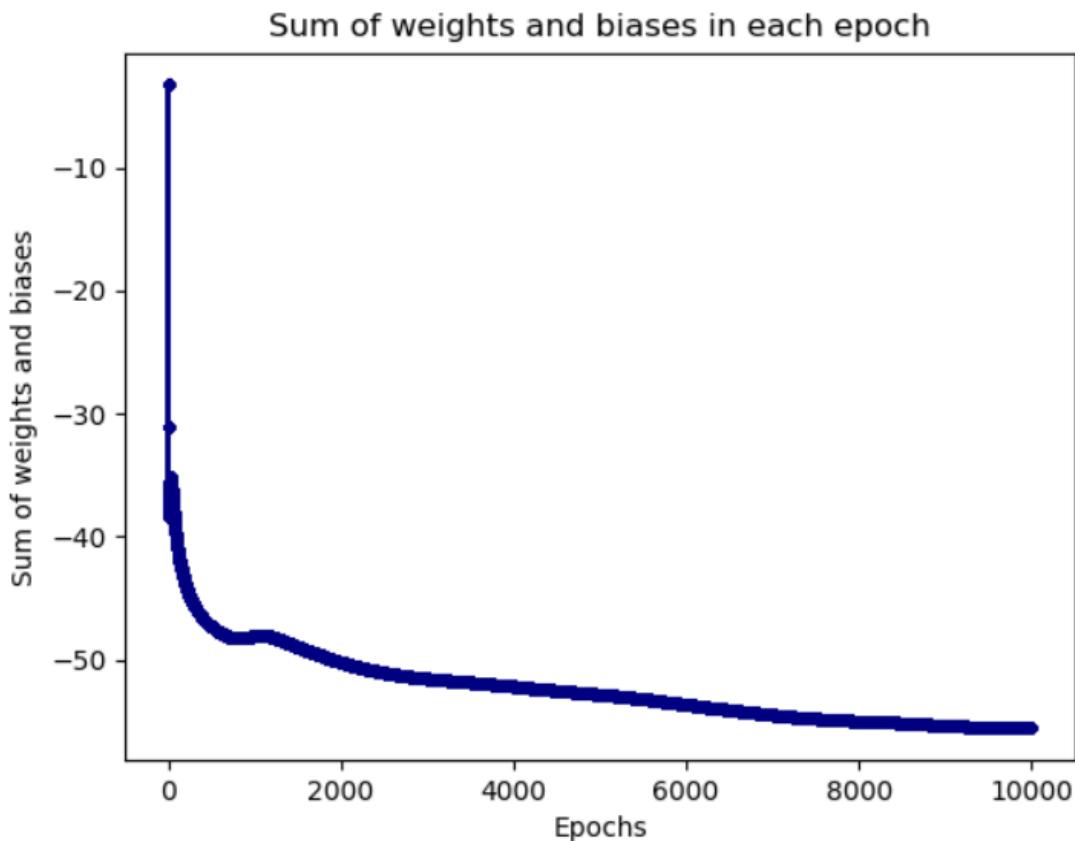
مقدار تابع هزینه بعد از 10000 ایپاک از ۱.۲۷ به ۰،۰۰۳۹ رسیده است. خطای مجموعه آموزش برابر ۰،۳۳ درصد است، خطای مجموعه ارزیابی برابر با ۲،۲۳ درصد است و خطای مجموعه تست برابر با ۲،۳۵ درصد است.

مقدار تابع هزینه در هر ایپاک:



در طی ۱۰۰۰۰ ایپاک تابع هزینه‌ی MSE از ۱،۲۷ به ۰،۰۰۳۹ رسیده است که میزان کاهش چشم‌گیر است. مقدار تابع هزینه سیر نزولی داشته است و بیشترین تغییرات در زمان انجام ایپاک اول بوده است و هر چه جلوتر رفته‌ایم در ایپاک‌های بعد از میزان تغییرات کاسته شده است و با روند کند تری تابع هزینه کم شده است.

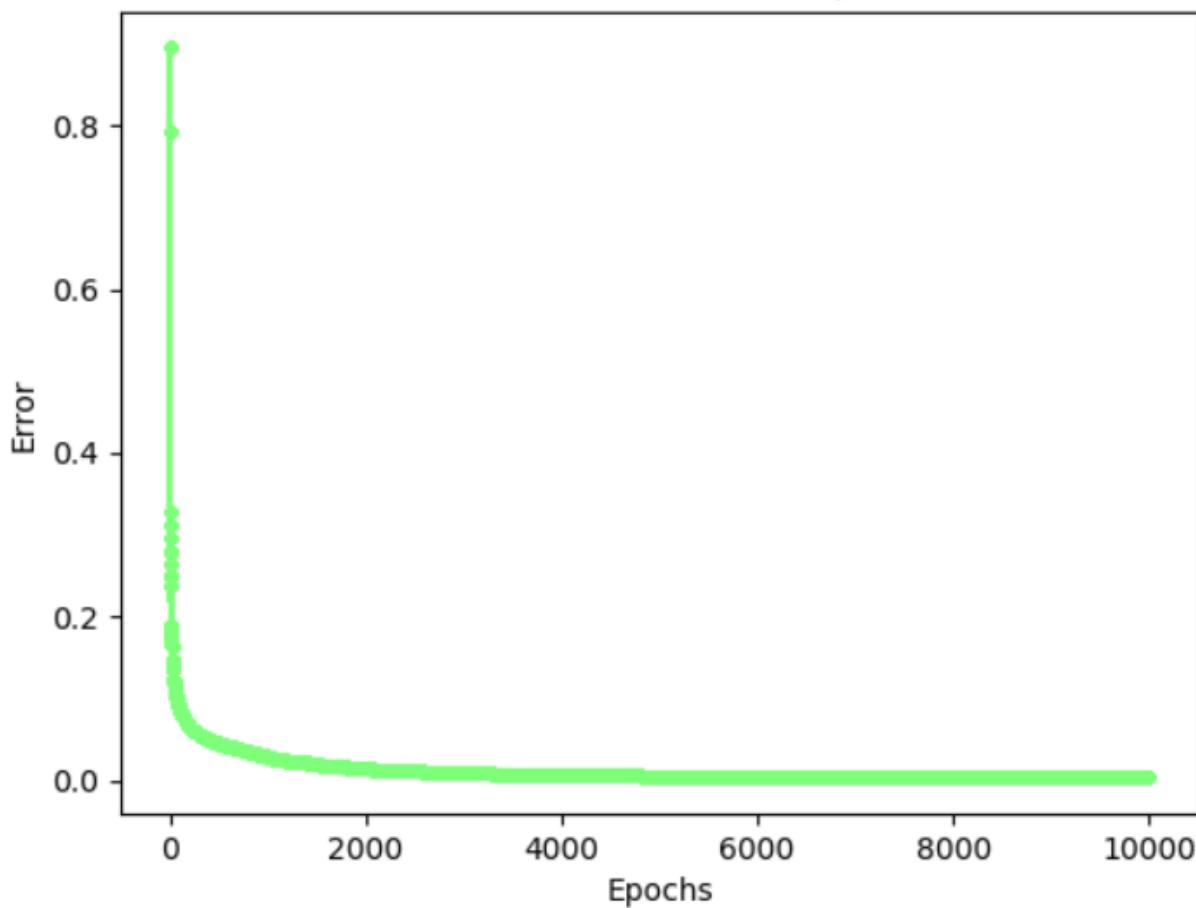
مجموع وزن‌ها و بایاس‌ها در هر ایپاک:



در ایپاک‌های آخر میزان تغییرات مجموعه وزن‌ها و بایاس‌ها کم شده است و تقریباً در آن محدوده باقی می‌ماند و تقریباً در چند ایپاک آخر مقدار مجموع ثابت مانده است. زیرا هر چه جلوتر می‌رویم معمولاً گرادیان‌ها کوچک‌تر می‌شوند و تغییرات وزن‌ها نسبت به مراحل قبل کمتر می‌شود.

خطای مجموعه‌ی آموزش در هر ایپاک:

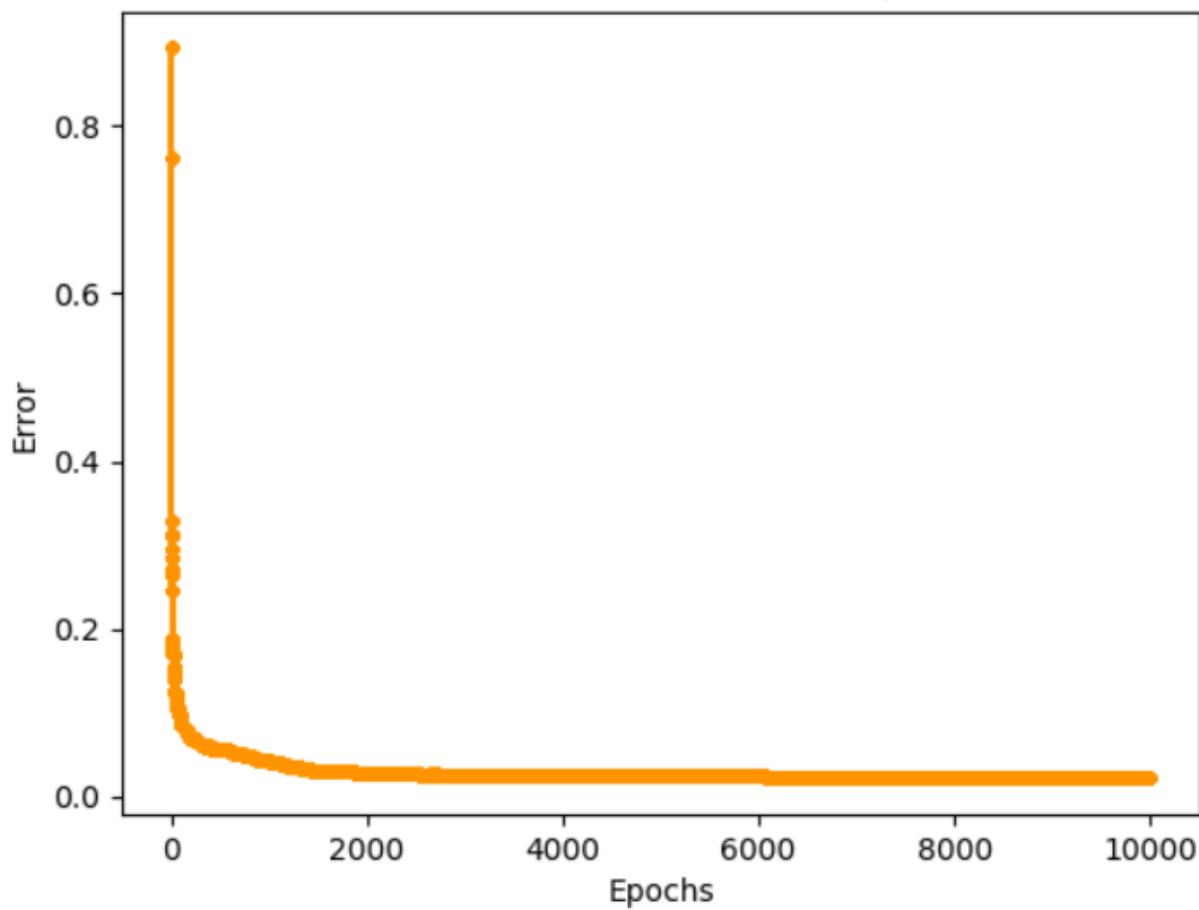
Train set error in each epoch



خطای مجموعه آموزش با آموزش بیشتر شبکه کاهش داشته است. بیشتر کاهش در ایپاک اول رخ داده است و در ایپاک‌های آخر میزان تغییرات خطا کم شده است و به مقدار ثابتی میل کرده است.

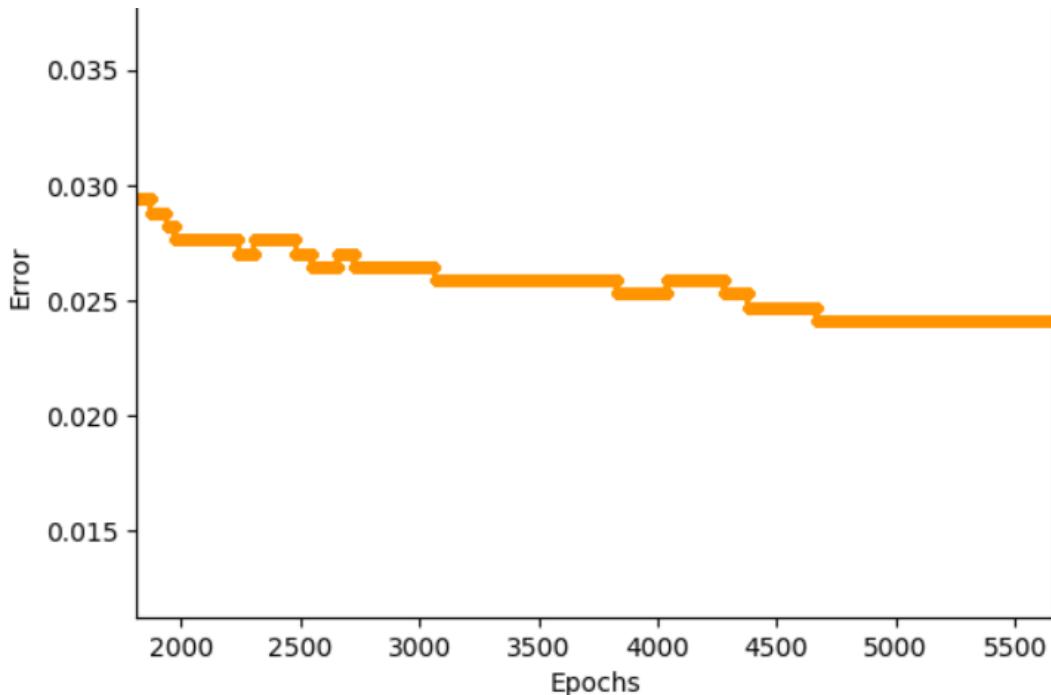
خطای مجموعه‌ی ارزیابی در هر ایپاک:

Validation set error in each epoch



خطای ارزیابی در هر ایپاک با آموزش بیشتر کاهش داشته است. در ایپاک‌های اول بیشترین تغییرات را داشته‌ایم و در ایپاک‌های پایانی از میزان تغییرات کاسته شده است.

بخشی از نمودار بالا با بزرگنمایی بیشتر.



این شبکه در فایل `MLP-output-23.json` ذخیره شده است.

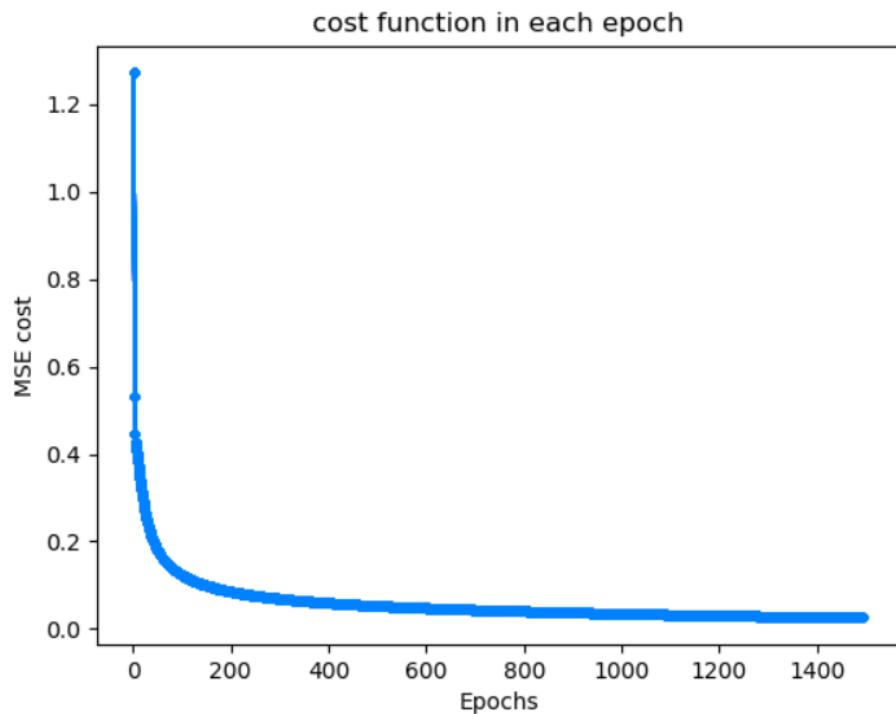
نتیجه‌گیری

سه شبکه‌ی اول را برای مقایسه `steepest descent` و `batch` آموزش دادم.

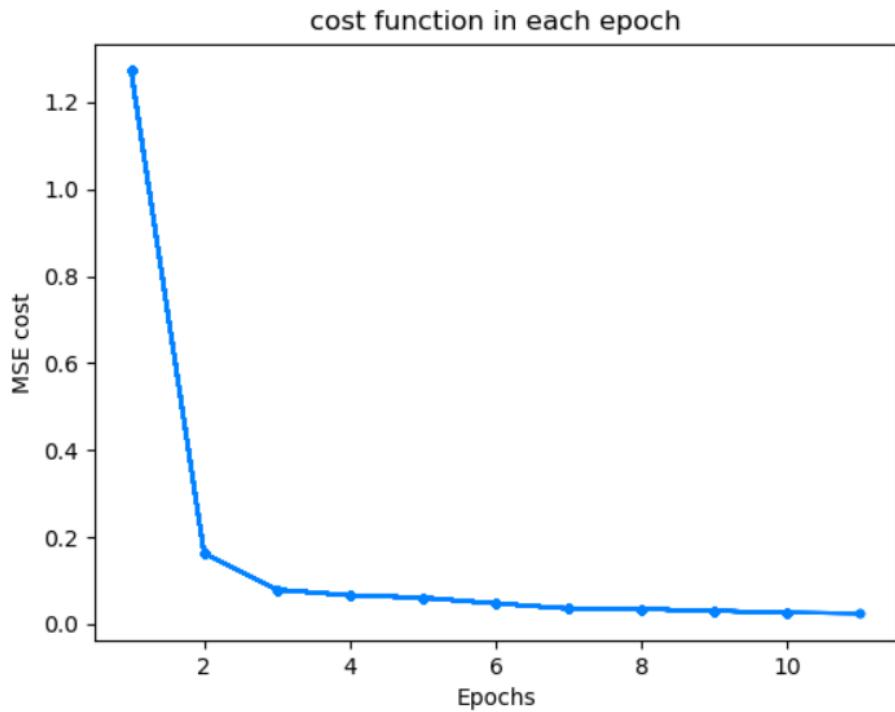
بررسی سرعت همگرایی: برای بررسی سرعت همگرایی دو شبکه‌ی یک و دو را مقایسه می‌کنیم. شبکه‌ی یک، یک شبکه‌ی ۳ لایه است که یک لایه‌ی نهان با ۶۴ نرون دارد و از روش `batch` استفاده می‌کند، با استفاده از کد `best_MLP.py` پارامترهای مناسب را برای آموزش آن پیدا کردم. سپس شبکه‌ی دو را ایجاد کردم که دقیقاً مانند شبکه‌ی یک است و از پارامترهای یکسانی استفاده می‌کند و وزن‌ها و بایاس‌های اولیه‌ی آن‌ها یکسان است، ولی شبکه‌ی دو از روش `steepest descent` استفاده می‌کند. شرط خاتمه‌ی آموزش هر دو شبکه را به طوری تنظیم کردم که در صورت کمتر شدن مقدار تابع هزینه از 0.026 آموزش شبکه‌ها متوقف شوند. در بالا و قسمت نتیجه‌های آزمایش، نمودارها و دقت‌های هر دو شبکه را نشان دادم،

در زیر نمودار Cost Function را در حین آموزش شبکه‌ی یک و دو مشاهده می‌کنید:

شبکه‌ی اول - batch



شبکه‌ی دوم - steepest descent



شبکه‌ی ۱۴۹۴ ایپاک لازم داشته‌است که مقدار تابع هزینه‌ی MSE برای آن از ۰,۰۲۶ برسد، ولی شبکه‌ی steepest descent به یازده ایپاک نیاز داشته است که مقدار تابع هزینه‌ی MSE برای آن از ۱,۲۷ به کمتر از ۰,۰۲۶ برسد، همین طور که مشاهده می‌شود سرعت همگرایی روش steepest descent از روش batch بسیار بیشتر است.

البته این نتیجه قابل پیش‌بینی بود، زیرا در steepest descent اگر بعد از هر ایپاک تابع هزینه کاهش یابد ضریب یادگیری دو برابر می‌شود ولی اگر تابع هزینه کاهش نداشته باشد وزن‌ها و بایاس‌ها آپدیت نمی‌شوند و ضریب یادگیری تقسیم بر دو می‌شود و ایپاک دوباره انجام می‌شود. به همین دلیل ضریب یادگیری همیشه در وضعیت مطلوبی قرار دارد و زمان‌هایی که به یک گام بلند در هنگام یادگیری نیاز داریم ضریب یادگیری زیاد است و زمان‌هایی که نیاز داریم که گام‌های کوچک برداریم، ضریب یادگیری کوچک است. این کار باعث می‌شود که با سرعت بسیار زیادی نسبت به حالتی که ضریب یادگیری ثابت است یا به آرامی کاهش پیدا می‌کند به سمت مینیمم محلی پیش رو حرکت کنیم.

بررسی دقیق: برای بررسی دقیق دو روش **steepest** و **batch**، شبکه‌ی سوم و چهارم را بررسی می‌کنیم. شبکه‌ی سه و چهار هر دو از نظر ساختار یکسان هستند و هر دو شبکه‌ی سه لایه با ۶۴ نرون در لایه میانی هستند. تنها تفاوت‌شان این است که شبکه‌ی چهارم از روش **batch** استفاده می‌کند و شبکه‌ی سوم از **Steepest Descent**. پارامترهای مناسب هر کدام برای آموزش را با استفاده کرد کد **best_MLP.py** به دست آورده‌ام که در آن با پارامترهای مختلف شبکه عصبی را آموزش داده‌ام و پارامترهای مناسب را یافته‌ام. شرط خاتمه در هر دو روش برخلاف دو شبکه‌ی یک و دو حداقل ایپاک است. در بخش نتیجه‌ی آزمایش در بالاتر، نمودارها و دقیق آن‌ها را گزارش کردم. در زیر دقیق دو شبکه را مشاهده می‌کنید:

خطاهای شبکه‌ی سوم:

```
Iter 19/20, Cost 0.0021693850919035367 ...
Iter 20/20, Cost 0.0021315804940415556 ...
=====
Train Error: 0.0026470588235294116
Validation Error: 0.020588235294117647
Test Error: 0.02

5-fold error is: 0.005514705882352941

=====^=====
```

شبکه‌ی چهارم:

```
Iter 9999/10000, Cost 0.003936511022443483 ...
Iter 10000/10000, Cost 0.003936097591796118 ...
=====
Train Error: 0.0033823529411764705
Validation Error: 0.02235294117647059
Test Error: 0.023529411764705882
```

شبکه‌ی دوم – Steepest Descent	شبکه‌ی چهارم – Batch	
۲,۰۵ درصد	۲,۲۳ درصد	خطای ارزیابی
۲ درصد	۲,۳۵ درصد	خطای تست
۰,۲۶ درصد	۰,۳۳ درصد	خطای آموزش

همین طور که در جدول بالا مشاهده می‌شود خطای ارزیابی و تست بهترین شبکه‌ای که برای یک شبکه‌ی سه لایه با ۶۴ نرون توانسته‌ام با روش Steepest Descent آموزش دهم، بیش از ۰,۲ درصد از خطای بهترین شبکه‌ای که برای یک شبکه‌ی سه لایه با ۶۴ نرون با روش Batch توانسته‌ام آموزش دهم بهتر است. این میزان تفاوت که در حدود ۰.۲ درصد است بسیار کم است و هر دو عملکردی نزدیک داشته‌اند. Steepest Descent با تفاوتی بسیار کم بهتر عمل کرده است. یکی از دلایل این است که batch در محاسبه‌ی گرادیان از تمام دیتاهای آموزش استفاده می‌کند. یکی دیگر از دلایل می‌تواند این باشد که در batch ضریب یادگیری ثابت است و متناسب با نقطه‌ی آغازین ممکن است در فاصله‌ی بسیار نزدیک به منیمم قرار بگیریم و به دلیل ثابت بودن ضریب یادگیری از حدی نتوانیم به آن نزدیک تر شویم ولی در روش steepest descent ضریب یادگیری تغییر می‌کند و مستقل از نقطه‌ی آغازین بهتر یا مساوی با روش batch می‌توانیم به نقطه‌ی منیمم نزدیک شویم.

آموزش شبکه با تابع هزینهی Cross Entropy و MSE

عنوان

یک بار شبکه را با تابع هزینهی MSE آموزش دهید و بار دیگر شبکه را با تابع هزینهی Cross Entropy آموزش دهید. رفتار شبکه را در هر دو حالت مقایسه کنید.

شرایط آزمایش

برای مقایسهی شبکه در زمان استفاده از MSE و Cross Entropy دو شبکهی سه لایه با تعداد ۶۴ نرون در لایهی نهان را که ساختار و پارامترها و وزن‌ها و بایاس‌های اولیهی یکسانی دارند را و فقط در نوع تابع هزینه با یک دیگر متفاوت‌اند را ایجاد می‌کنم و سپس سرعت همگرایی آن‌ها و دقتشان را بررسی می‌کنم. پارامترها را با استفاده از تابع best_MLP.py و با تست مقدارهای مختلف برای ضریب یادگیری و تعداد ایپاک به دست آورده‌ام. دز زیر فایل‌های ورودی برای ساخت دو شبکه را مشاهده می‌کنید:

فایل پیکربندی شبکهی اول (input-30.json):

```
input-30.json
1  {
2      "digits_letters_digits": "digits",
3      "ratio_of_train_set": 0.80,
4      "ratio_of_valid_set": 0.10,
5      "ratio_of_test_set": 0.10,
6      "learning_rate": 0.1,
7      "max_epochs": 2000,
8      "cut_cost": -1,
9      "random_state": 2,
10     "K_fold": 5,
11     "neurons_layes": [1024, 64, 10],
12     "type_layers": ["tanh", "sigmoid"],
13     "type_of_cost": "cross_entropy",
14     "mode": "batch"
15 }
```

شبکه‌ی شماره اول: از دیتاست عده‌ها استفاده می‌شود، ۸۰ درصد دیتاست برای آموزش است، ۱۰ درصد دیتاست برای ارزیابی است، ۱۰ درصد دیتاست برای تست است، ضریب یادگیری برابر با ۰.۱ است، حداکثر تعداد ایپاک‌ها ۲۰۰۰ است، شرط کمتر شدن تابع هزینه از حد مشخصی برای خاتمه‌ی آموزش چک نمی‌شود، تولید عده‌های تصادفی با عدد ۲ بذرپاشی می‌شود، تعداد فیچرها ۱۰۲۴ تا است، لایه نهان اول ۶۴ نرون دارد. لایه خروجی ۱۰ نرون دارد. تابع فعال سازی لایه نهان اول tanh است و تابع فعال سازی لایه خروجی sigmoid است، تابع هزینه Cross Entropy است، از روش batch 5-fold cross validation استفاده شده است. از ۵-fold cross validation استفاده می‌شود.

نکته: در این شبکه چون از Cross Entropy استفاده می‌شود خروجی لایه نرمال سازی می‌شود.

فایل پیکربندی شبکه‌ی دوم (input-31.json):

```
input-31.json
1 {
2   "digits_letters_digits": "digits",
3   "ratio_of_train_set": 0.80,
4   "ratio_of_valid_set": 0.10,
5   "ratio_of_test_set": 0.10,
6   "learning_rate": 0.1,
7   "max_epochs": 2000,
8   "cut_cost": -1,
9   "random_state": 2,
10  "K_fold": 5,
11  "neurons_layes": [1024, 64, 10],
12  "type_layers": ["tanh", "sigmoid"],
13  "type_of_cost": "MSE",
14  "mode": "batch"
15 }
```

شبکه‌ی شماره دوم: از دیتاست عده‌ها استفاده می‌شود، ۸۰ درصد دیتاست برای آموزش است، ۱۰ درصد دیتاست برای ارزیابی است، ۱۰ درصد دیتاست برای تست است، ضریب یادگیری برابر با ۰.۱ است، حداکثر تعداد ایپاک‌ها ۲۰۰۰ است، شرط کمتر شدن تابع هزینه از حد مشخصی برای خاتمه‌ی آموزش چک نمی‌شود، تولید عده‌های

تصادفی با عدد ۲ بذرپاشی می‌شود، تعداد فیچرها ۱۰۲۴ تا است، لایه نهان اول ۶۴ نرون دارد. لایه خروجی ۱۰ نرون دارد. تابع فعال سازی لایه نهان اول \tanh است و تابع فعال سازی لایه خروجی sigmoid است، تابع هزینه MSE است، از روش $5\text{-fold cross validation}$ استفاده شده است. از batch استفاده می‌شود.

نتیجه انجام آزمایش

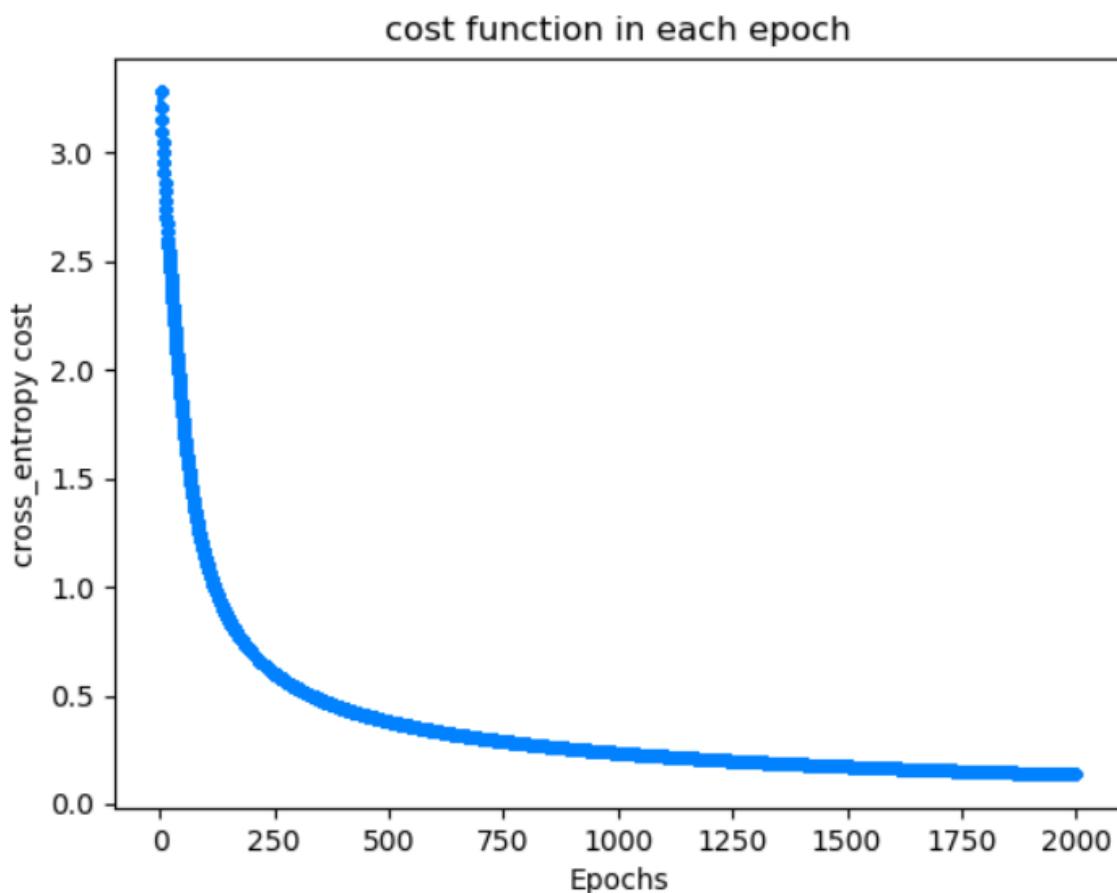
نتیجه‌های شبکه عصبی اول:

مقدار تابع هزینه در آخرین ایپاک، خطای آموزش، خطای ارزیابی، خطای تست :

```
Calculating 5-fold in a separate thread has been
Iter 0/2000, Cost 3.281165641269705 ...
Iter 1/2000, Cost 3.281165641269705 ...
Iter 2/2000, Cost 3.211750743990097 ...
Iter 3/2000, Cost 3.153416807955613 ...
Iter 4/2000, Cost 3.100632086848702 ...
Iter 5/2000, Cost 3.0505951686555632 ...
Iter 6/2000, Cost 3.002152890409355 ...
Iter 7/2000, Cost 2.955020648803279 ...
Iter 8/2000, Cost 2.90070560107910
...
Iter 1994/2000, Cost 0.1363962515337146 ...
Iter 1995/2000, Cost 0.1363370313138372 ...
Iter 1996/2000, Cost 0.1362778580183386 ...
Iter 1997/2000, Cost 0.1362187315797566 ...
Iter 1998/2000, Cost 0.13615965193086726 ...
Iter 1999/2000, Cost 0.1361006190046849 ...
Iter 2000/2000, Cost 0.1360416327344609 ...
=====V=====
Train Error: 0.00926470588235294
Validation Error: 0.02823529411764706
Test Error: 0.029411764705882353
```

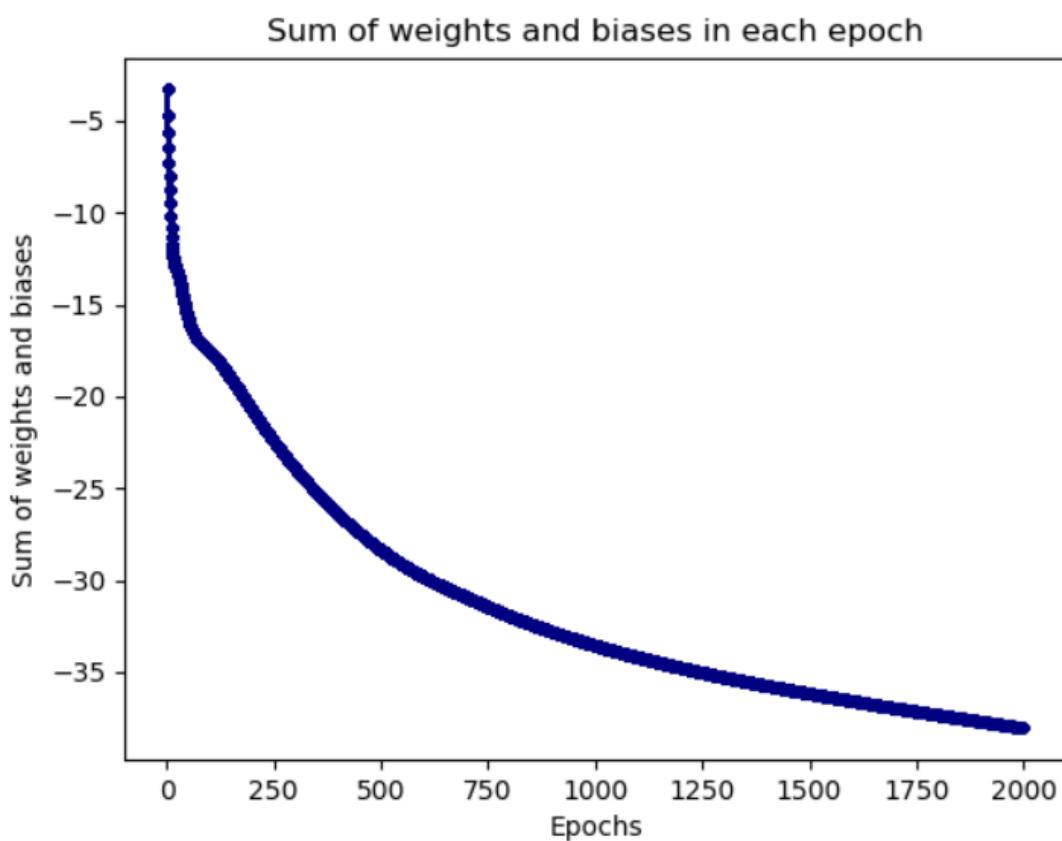
مقدار تابع هزینه بعد از ۲۰۰۰ ایپاک از ۳,۲۸ به ۰,۱۳ رسیده است. خطای مجموعه آموزش برابر ۹۲.۰ درصد است، خطای مجموعه ارزیابی برابر با ۲,۸۲ درصد است و خطای مجموعه تست برابر با ۲,۹۴ درصد است.

مقدار تابع هزینه در هر ایپاک:



در طی ۱۰۰۰۰ ایپاک تابع هزینه‌ی MSE از ۱,۲۷ به ۰,۰۰۳۹ رسیده است که میزان کاهش چشم‌گیر است. مقدار تابع هزینه سیر نزولی داشته است و بیشترین تغییرات در زمان انجام ایپاک اول بوده است و هر چه جلوتر رفته‌ایم در ایپاک‌های بعد از میزان تغییرات کاسته شده است و با روند کند تری تابع هزینه کم شده است.

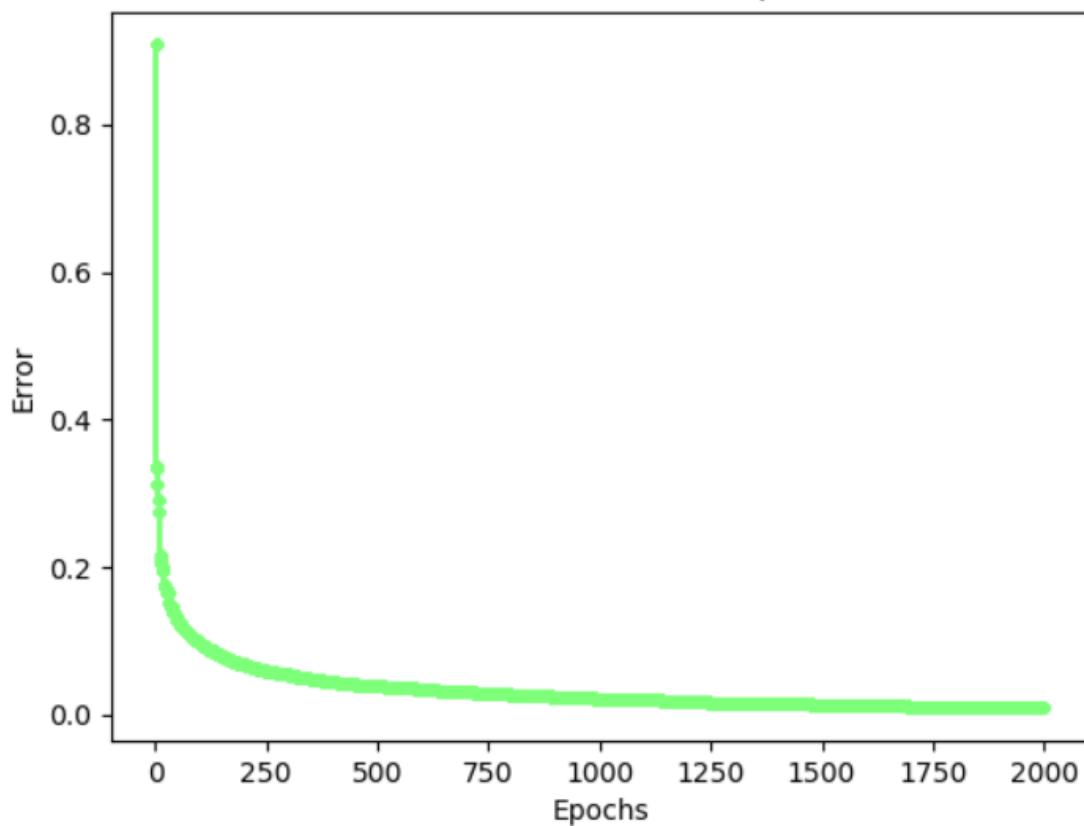
مجموع وزن‌ها و بایاس‌ها در هر ایپاک:



در ایپاک‌های آخر میزان تغییرات مجموعه وزن‌ها و بایاس‌ها کم شده است و تقریبا در آن محدوده باقی می‌ماند و تقریبا در چند ایپاک آخر مقدار مجموع ثابت مانده است. زیرا هر چه جلوتر می‌رویم معمولاً گرادیان‌ها کوچک‌تر می‌شوند و تغییرات وزن‌ها نسبت به مراحل قبل کمتر می‌شود.

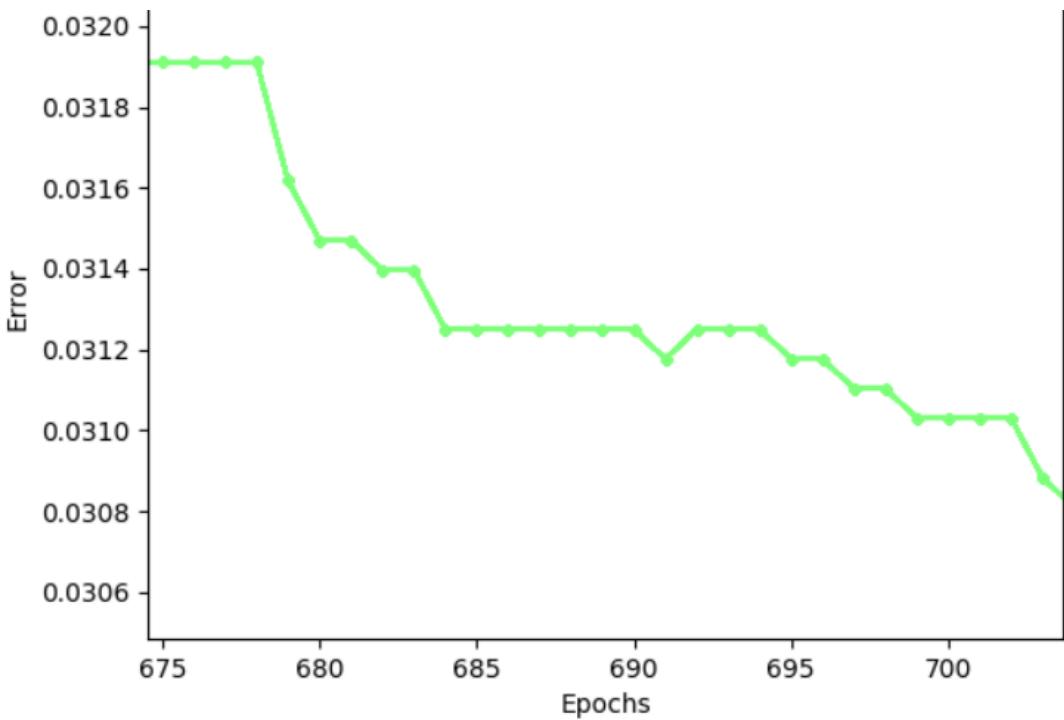
خطای مجموعه‌ی آموزش در هر ایپاک:

Train set error in each epoch

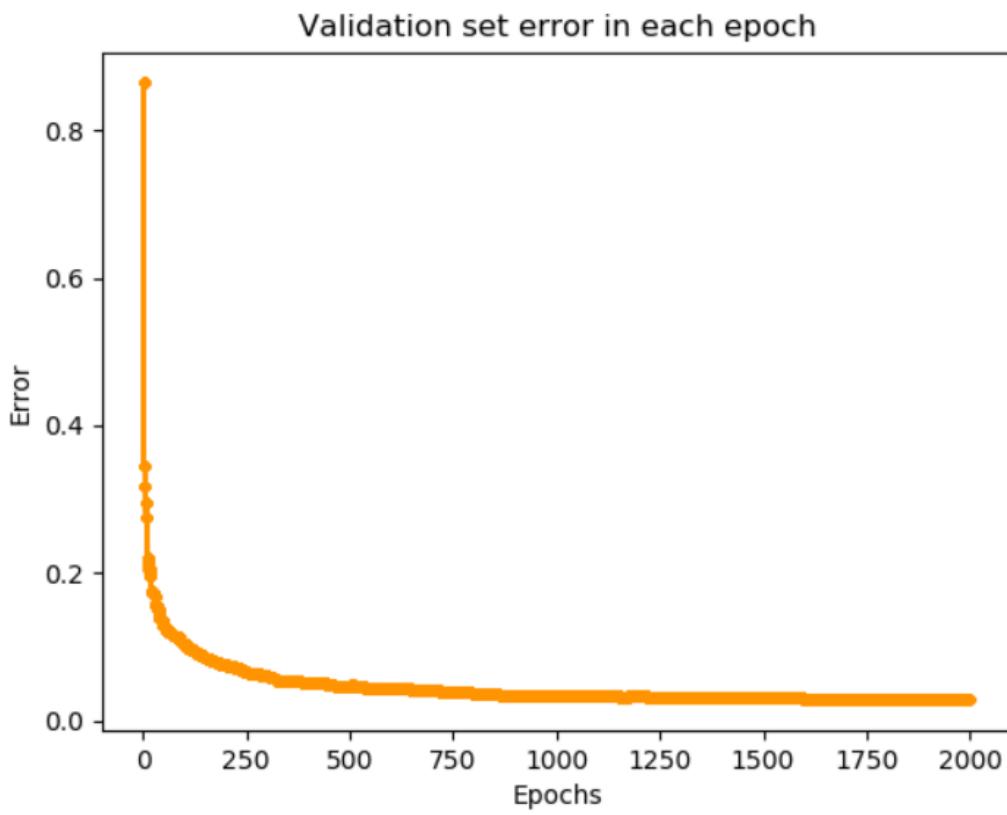


خطای مجموعه آموزش با آموزش بیشتر شبکه کاهش داشته است. بیشتر کاهش در ایپاک اول رخ داده است و در ایپاک های آخر میزان تغییرات خطا کم شده است و به مقدار ثابتی میل کرده است.

بخشی کوچکی از نمودار بالا با بزرگنمایی بیشتر:



خطای مجموعه ارزیابی در هر ایپاک:



خطای ارزیابی در هر ایپاک با آموزش بیشتر کاهش داشته است. در ایپاک‌های اول بیشترین تغییرات را داشته‌ایم و در ایپاک‌های پایانی از میزان تغییرات کاسته شده است.

این شبکه در فایل MLP-output-30.json ذخیره شده است.

نتیجه‌های شبکه عصبی دوم:

مقدار تابع هزینه در آخرین ایپاک، خطای آموزش، خطای ارزیابی، خطای تست :

```

Calculating 5-fold in a separate thread has been begun.
Iter 0/2000, Cost 1.272158268971262 ...
Iter 1/2000, Cost 1.272158268971262 ...
Iter 2/2000, Cost 0.8383200742140927 ...
Iter 3/2000, Cost 0.6936671006638369 ...
Iter 4/2000, Cost 0.6214793726158623 ...
Iter 5/2000, Cost 0.5755489908081496 ...
Iter 6/2000, Cost 0.5435266216851786 ...
Iter 7/2000, Cost 0.5200051194164075 ...

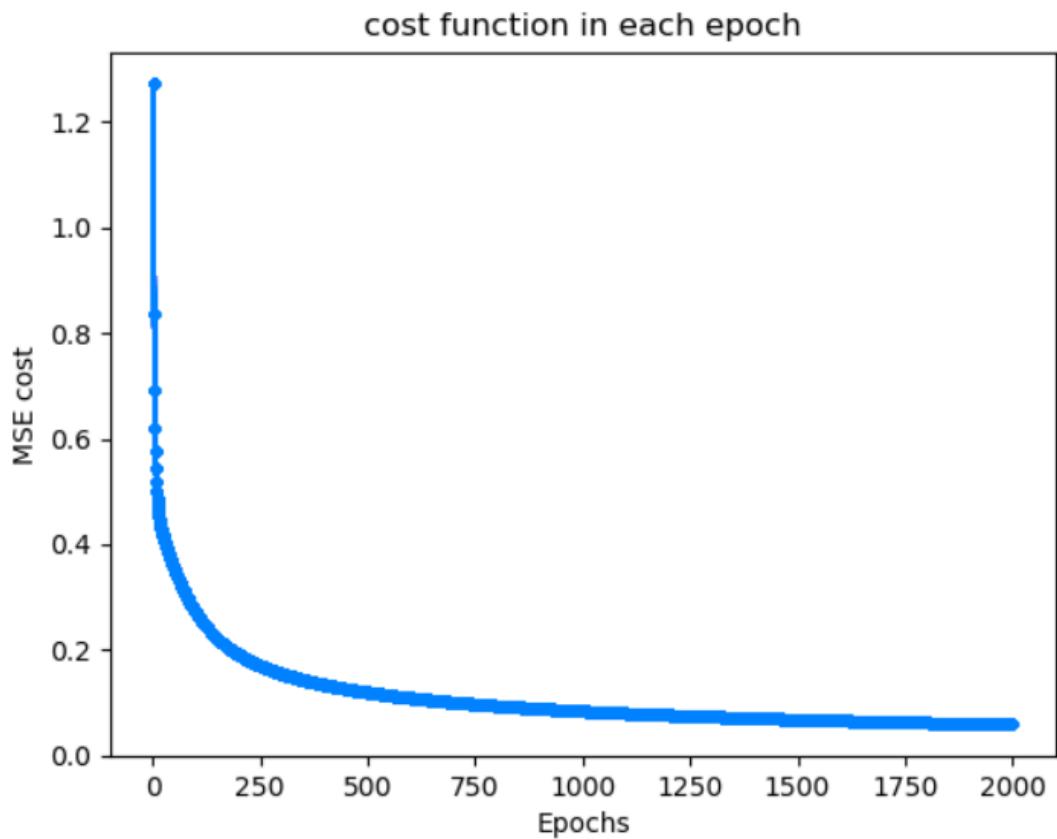
...
Iter 1993/2000, Cost 0.059258828906246265 ...
Iter 1994/2000, Cost 0.05924323343470585 ...
Iter 1995/2000, Cost 0.059227649095251626 ...
Iter 1996/2000, Cost 0.059212075873745426 ...
Iter 1997/2000, Cost 0.05919651375607392 ...
Iter 1998/2000, Cost 0.05918096272814868 ...
Iter 1999/2000, Cost 0.05916542277590606 ...
Iter 2000/2000, Cost 0.05914989388530712 ...
=====V=====
Train Error: 0.05014705882352941
Validation Error: 0.057058823529411766
Test Error: 0.05764705882352941

```

5-fold error is: 0.0325735294117647

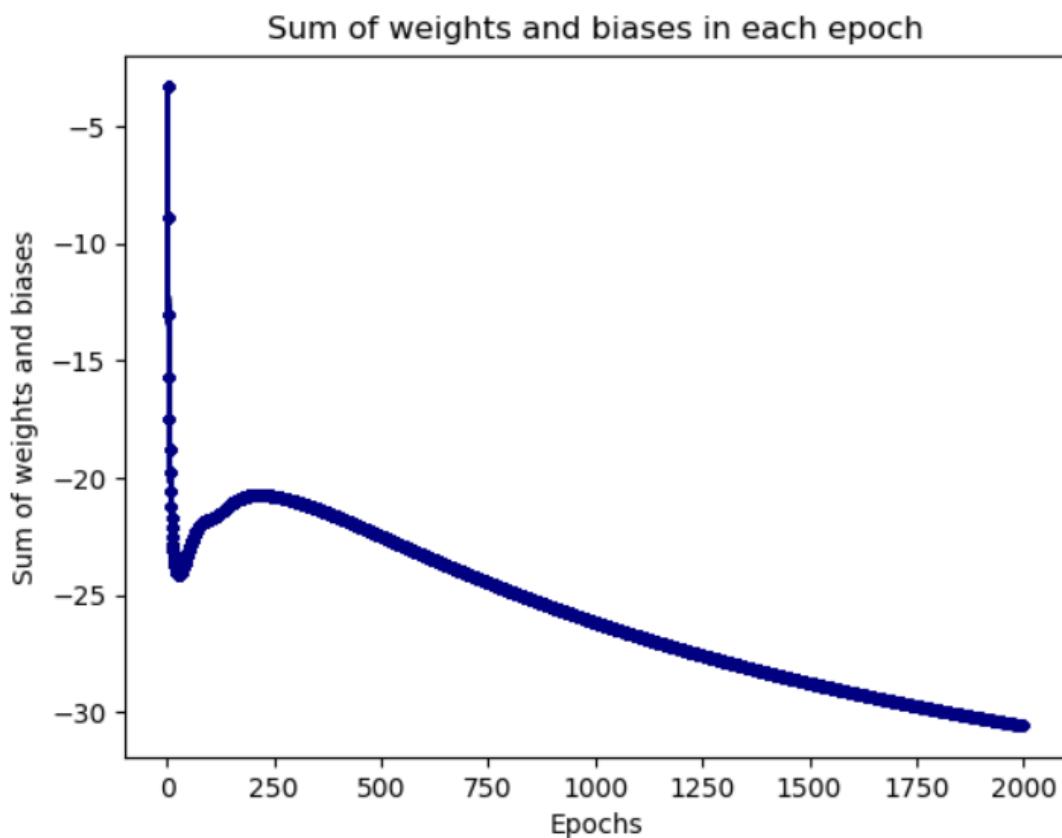
مقدار تابع هزینه بعد از ۲۰۰۰ ایپاک از ۱,۲۷ به ۰,۰۵۹ رسیده است. خطای مجموعه آموزش برابر ۵,۰۱ درصد است، خطای مجموعه ارزیابی برابر با ۵,۷۰ درصد است و خطای مجموعه تست برابر با ۵,۷۶ درصد است.

مقدار تابع هزینه در هر ایپاک:



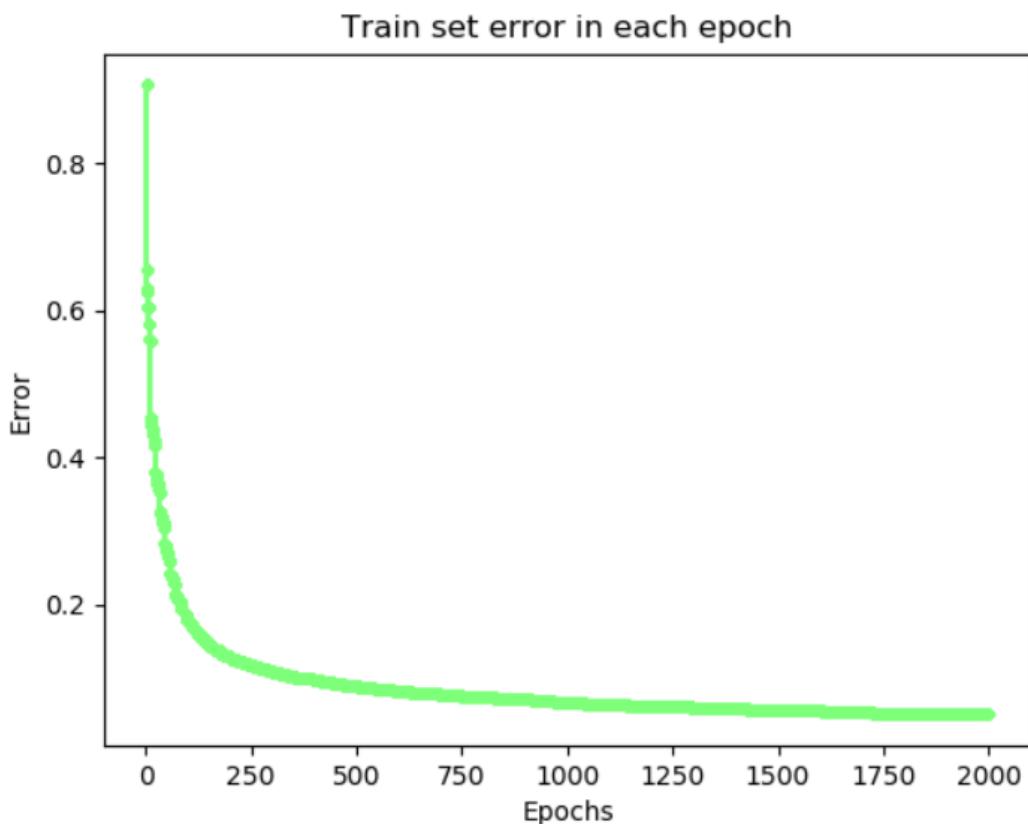
در طی ۱۰۰۰۰ ایپاک تابع هزینه‌ی MSE از ۱,۲۷ به ۰,۰۰۳۹ رسیده است که میزان کاهش چشم‌گیر است. مقدار تابع هزینه سیر نزولی داشته است و بیشترین تغییرات در زمان انجام ایپاک اول بوده است و هر چه جلوتر رفته‌ایم در ایپاک‌های بعد از میزان تغییرات کاسته شده است و با روند کند تری تابع هزینه کم شده است.

مجموع وزن‌ها و بایاس‌ها در هر ایپاک:



در ایپاک‌های آخر میزان تغییرات مجموعه وزن‌ها و بایاس‌ها کم شده است و تقریبا در آن محدوده باقی می‌ماند و تقریبا در چند ایپاک آخر مقدار مجموع ثابت مانده است. زیرا هر چه جلوتر می‌رویم معمولاً گرادیان‌ها کوچک‌تر می‌شوند و تغییرات وزن‌ها نسبت به مراحل قبل کمتر می‌شود.

خطای مجموعه‌ی آموزش در هر ایپاک:

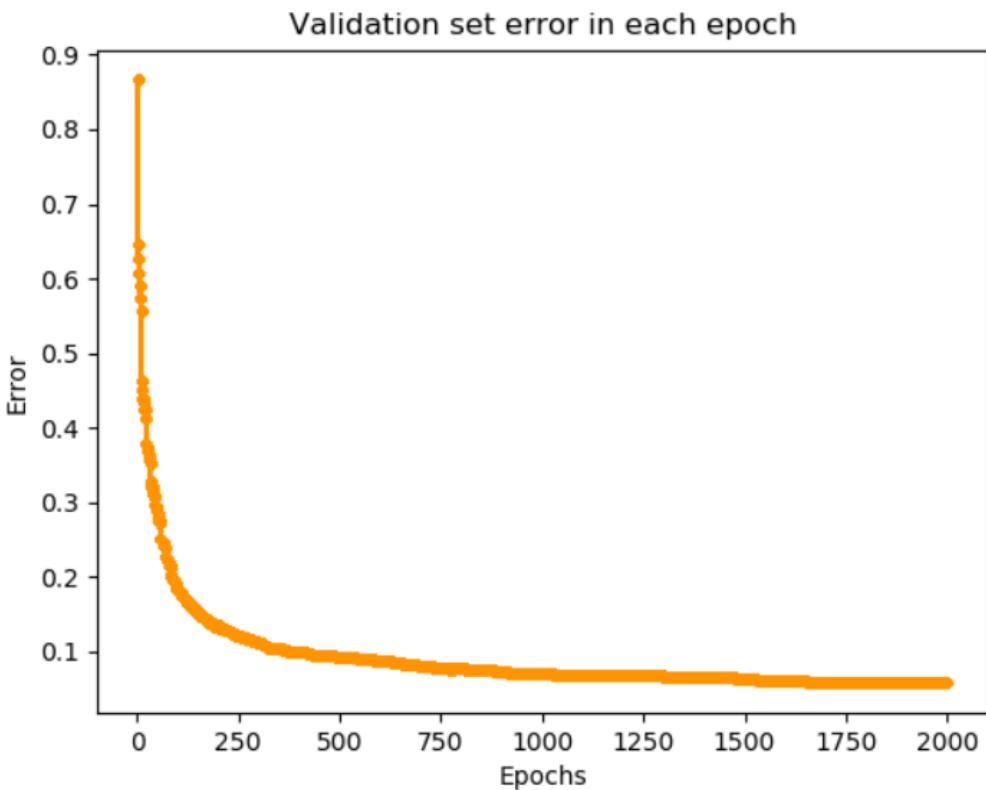


خطای مجموعه آموزش با آموزش بیشتر شبکه کاهش داشته است. بیشتر کاهش در ایپاک اول رخ داده است و در ایپاک‌های آخر میزان تغییرات خطا کم شده است و به مقدار ثابتی میل کرده است.

بخشی کوچکی از نمودار بالا با بزرگنمایی بیشتر:



خطای مجموعه‌ی ارزیابی در هر ایپاک:



خطای ارزیابی در هر ایپاک با آموزش بیشتر کاهش داشته است. در ایپاک‌های اول بیشترین تغییرات را داشته‌ایم و در ایپاک‌های پایانی از میزان تغییرات کاسته شده است.

این شبکه در فایل `MLP-output-31.json` ذخیره شده است.

نتیجه‌گیری

در بالا نتیجه‌های حاصل از آموزش دو شبکه‌ی سه لایه با تعداد ۶۴ نرون در لایه‌ی نهان را که وزن‌ها و بایاس‌های اولیه یکسان و ساختار یکسان را دارند را نمایش دادیم، این دو شبکه فقط در نوع تابع هزینه متفاوت‌اند. در زیر خلاصه‌ای از عملکرد دو شبکه را مشاهده می‌کنیم:

MSE-۲ شبکه‌ی	Cross Entropy - ۱ شبکه‌ی	
1.272	3.281	مقدار تابع هزینه قبل از شروع آموزش
0.059	0.136	مقدار تابع هزینه در انتها (ایپاک ۲۰۰۰)
1.213	3.145	اختلاف تابع هزینه در ابتدا و انتها
0.0006065	0.0015725	مقدار کاهش متوسط تابع هزینه در هر ایپاک
5.01 درصد	0.92 درصد	خطای آموزش
5.70 درصد	2.82 درصد	خطای ارزیابی
5.76 درصد	2.94 درصد	خطای تست

از آنجایی که دو شبکه ساختار یکسانی دارند و وزن‌ها و بایاس‌های اولیه‌ی آن‌ها برابر است جدول بالا اطلاعات بسیار ارزشمندی در اختیارمان قرار می‌دهد که امکان مقایسه‌ی MSE و Cross Entropy را می‌دهد.

شبکه با تابع هزینه‌ی Cross Entropy در ابتدا و قبل از شروع آموزش مقداری برابر با ۳,۲۸۱ دارد که طی ۲۰۰۰ ایپاک این مقدار به ۰,۱۳۶ رسیده است یعنی ۳,۱۴۵ واحد کاهش داشته است و به طور متوسط در هر ایپاک ۰,۰۰۰۱۵۷۲۵ واحد کاهش داشته است در حالی که شبکه دوم با تابع هزینه‌ی MSE در ابتدا و قبل از شروع آموزش مقدار تابع هزینه برای آن ۱,۲۷۲ بوده است و بعد از ۲۰۰۰ ایپاک مقدار تابع هزینه به ۰,۰۵۹ رسیده است و ۱,۲۱۳ واحد کاهش رخ داده است یعنی به طور متوسط در هر ایپاک ۰,۰۰۰۶۰۶۵ واحد تابع هزینه کاهش داشته است. همین طور که مشاهده می‌شود میزان کاهش در شبکه اول با تابع هزینه‌ی Cross Entropy بسیار بیشتر بوده است و در هر مرحله به طور میانگین تابع هزینه نسبت به شبکه MSE کاهش بسیار بیشتری داشته است. نسبت کاهش میانگین تابع هزینه در هر ایپاک در شبکه‌ی Cross Entropy تقریباً ۲,۵۹ برابر شبکه‌ی MSE است.

وزن‌ها و بایاس‌های هر دو شبکه در ابتدا یکسان است در نتیجه قبل از شروع آموزش هر دو شبکه بر روی مجموعه‌ی آموزش، ارزیابی و تست عملکرد یکسانی دارند ولی شبکه با تابع هزینه‌ی Cross Entropy در طی ۲۰۰۰ ایپاک خطای مجموعه‌ی تستش به ۰,۹۲ درصد، خطای مجموعه ارزیابی‌اش به ۲,۸۲ درصد و خطای

مجموعه‌ی تستش به ۲,۹۴ درصد رسیده است. در حالی که شبکه با تابع هزینه MSE طی همان تعداد ایپاک و با خطای‌های یکسان اولیه و قبل از آموزش با شبکه‌ی اول خطای آموزشش به ۱۵,۰ درصد و خطای ارزیابی اش به ۵,۷۰ درصد و خطای مجموعه‌ی تستش به ۵,۷۶ درصد رسیده است. همین طوری که مشاهده می‌شود شبکه با تابع هزینه Cross Entropy بسیار سریع تر همگرا می‌شود و مقدار تابع هزینه‌اش کاهش پیدا می‌کند و در مدت یکسانی با شبکه MSE به خطای‌های بهتری می‌رسد در نتیجه تابع Cross Entropy بسیار بهتر از MSE عمل می‌کند.

تابع هزینه‌ی MSE برای Regression مناسب است زیر مقدارهای پیوسته داریم ولی در Classification چند مقدار به عنوان کلاس داریم.

البته بهتر عمل کردن تابع Cross Entropy نسبت به MSE قابل پیش‌بینی بود، به دلایل زیر:

(توضیح زیر را از <http://rohanvarma.me/Loss-Functions/> کمک گرفته‌ام)

ما می‌توانیم نشان دهیم که برای یک مسیله‌ی دسته‌بندی استفاده از تابع هزینه‌ی cross entropy در گرادیان نرولی نسبت به MSE در گرادیان نرولی موجب افزایش سرعت یادگیری می‌شود. ابتدا نگاهی به قانون کلی آپدیت در گرادیان نرولی داشته باشیم:

```
For i = 1 ... N:
    Compute dJ/dw_i for i = 1 ... M parameters
    Let w_i = w_i - learning_rate * dJ/dw_i
```

در گرادیان نرولی مشتق جزیی تمام پارامترهایی که قابلیت یادگرفتن دارند محاسبه می‌شود و مشتق جزیی ضرب در یک ضریب به اسم ضریب یادگیری می‌شود و در نهایت نتیجه را از پارامتر برای آپدیت کردن کم می‌کنیم. اینگونه یک گام به سمت مینیمم محلی بر می‌داریم.

این نشان می‌دهد سرعت یادگیری و حرکت به سمت مینیمم محلی به دو چیز وابسته است: ضریب یادگیری و اندازه‌ی مشتقات جزیی. ضریب یادگیری یک hyper parameter است که مقدار آن را تنظیم می‌کنیم. در ادامه تمرکزمان را بر روی اندازه‌ی مشتقات جزیی می‌گذاریم.

فرض کنید دسته‌بندی باینری ای داریم به طوری که x_i یک ویکتور از feature‌ها است، y_i برچسب آن وکتور است و \hat{y}_i زیر را داریم:

$$\hat{y}_i = h_{\theta}(x_i)$$

ما نشان اگر مدلمان به صورت زیر باشد

$$h_{\theta}(x) = \sigma(Wx_i + b),$$

یادگیری با استفاده از cross entropy نسبت به MSE در گام‌های اول می‌تواند سریع تر خ دهد و این موضوع بسیار مهم است بخصوص زمانی که در ابتدا وزن‌ها به صورت تصادفی وزن دهی شده‌اند و مدل ضعیف عمل می‌کند.

در صورتی که تابع هزینه‌مان به صورت زیر باشد

$$J = \frac{1}{2} (y_i - \hat{y}_i)^2$$

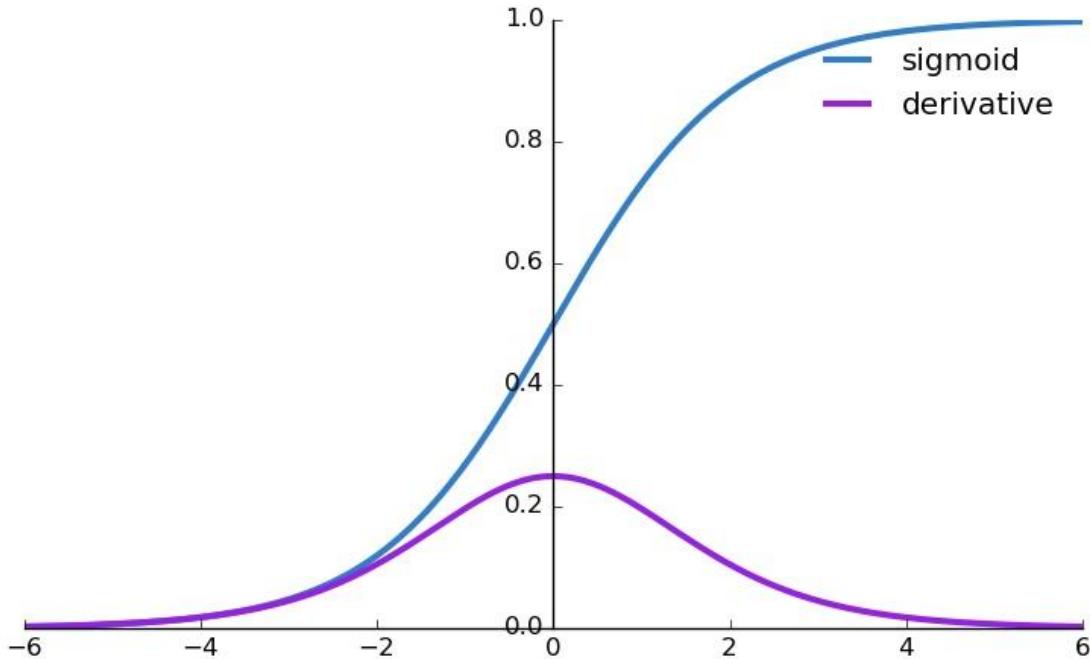
مشتق تابع هزینه نسبت به وزن‌ها با استفاده از قانون Chain Rule به صورت زیر به دست می‌آید:

$$\frac{dJ}{dW} = (y_i - \hat{y}_i) \sigma'(Wx_i + b) x_i$$

که مشتق سیگموید را می‌توان به صورت زیر هم نوشت:

$$\sigma(Wx_i + b)(1 - \sigma(Wx_i + b))$$

از آنجایی که در ابتدا وزن‌ها را به صورت تصادفی و نزدیک به صفر انتخاب می‌کنیم باعث می‌شود این مشتق‌های جزیی در ابتدا بسیار کوچک شوند، نمودار زیر که تابع سیگموید و مشتقش را نمایش می‌دهد، نشان می‌دهد زمانی که خروجی نزدیک به صفر است مشتق‌ها بسیار کوچک هستند.



این می‌تواند باعث کند شدن آموزش در ابتدا شود زیرا مشتق‌های جزیی کوچک سبب تغییر کوچک در وزن‌ها می‌شود و زمان بیشتری برای رسیدن به مینیمم مورد نیاز است.

اگر از تابع هزینه Cross Entropy استفاده کنیم

$$J = - \sum_{i=1}^N y_i \log(\sigma(Wx_i + b)) + (1 - y_i) \log(1 - \sigma(Wx_i + b))$$

مشتق‌های جزیی به صورت زیر به دست می‌آیند:

$$\frac{dJ}{dW} = - \sum_{i=1}^N \frac{y_i x_i \sigma'(z)}{\sigma(z)} - \frac{(1 - y_i) x_i \sigma'(z)}{1 - \sigma(z)}$$

که اگر مشتق سیگموید را بر اساس خودش بنویسیم و ساده سازی کنیم عبارت زیر به دست می‌آید:

$$\sum_{i=1}^N x_i(\sigma(z) - y_i)$$

همین طور که دیده می‌شود در این رابطه مشتق سیگموید حضور ندارد و اندازه‌ی مشتق جزیی کاملاً به سیگموید و مقدار اختلاف سیگموید و برچسب (چه میزان پیش‌بینی از واقعیت فاصله داشته است) بستگی دارد. همین باعث می‌شود در ابتدا مشتق‌ها بزرگ باشند و در ادامه مشتق‌ها کوچک و کوچک‌تر می‌شوند.

در ابتدا که خطا زیاد است گرادیان‌ها بزرگ‌تر هستند و در گام‌های جلوتر که خطا کم شده است گام‌ها کوچک‌می‌شود و این به معنای افزایش سرعت یادگیری است.

این توضیح‌ها برای یک نرون با sigmoid activation بود به همین طریق می‌توان این مفهوم را بر روی شبکه‌های چند لایه هم بست داد. در شبکه‌ی چند لایه که لایه‌ی آخر تابع فعال‌سازی سیگموید دارد، لایه‌ی خروجی دقیقاً مانند آنچه در بالا گفته شد عمل می‌کنند و خطای پس انتشار در لایه‌های نهان اینگونه محاسبه می‌شود:

$$\delta_{pj}^{(l)} = f'_j(I_{pj}^{(l)}) \cdot \sum_k \delta_{pk}^{(l+1)} \cdot w_{jk}^{(l+1)}$$

هرچه میزان خطای پس انتشار لایه‌های جلوتر بزرگ‌تر باشد خطای پس انتشار بزرگ‌تر می‌شود در نتیجه استفاده از Cross Entropy در MLP سبب افزایش سرعت یادگیری می‌شود.

آموزش شبکه‌ی پرسپترونی چند لایه بر روی دیتاست رقم‌ها و حرف‌ها

عنوان

با توجه به نتایج حاصل از آزمایش‌های قبلی، بهترین شبکه عصبی پرسپترونی چند لایه را برای این مجموعه داده طراحی کرده و نتایج خود را به همراه توضیح کامل معماری برای شبکه توضیح دهید.

شرایط آزمایش

برای این کار دو شبکه زیر را طراحی کرده ام که در ادامه دلیل انتخاب آن‌ها را بیان می‌کنم.

فایل کانفیگ شبکه‌ی اول(:input-40.json)

```
input-40.json
1  {
2      "digits_letters_digits": "letters_digits",
3      "ratio_of_train_set": 0.80,
4      "ratio_of_valid_set": 0.10,
5      "ratio_of_test_set": 0.10,
6      "learning_rate": 0.1,
7      "max_epochs": 10000,
8      "cut_cost": -1,
9      "random_state": 2,
10     "K_fold": -1,
11     "neurons_layes": [1024, 128, 46],
12     "type_layers": ["tanh", "sigmoid"],
13     "type_of_cost": "cross_entropy",
14     "mode": "batch"
15 }
```

شبکه‌ی شماره اول: از دیتاست عددها استفاده می‌شود، ۸۰ درصد دیتاست برای آموزش است، ۱۰ درصد دیتاست برای ارزیابی است، ۱۰ درصد دیتاست برای تست است، ضریب یادگیری برابر با ۰.۱ است، حداقل تعداد ایپاک‌ها ۱۰۰۰۰ است، شرط کمتر شدن تابع هزینه از حد مشخصی برای خاتمه‌ی آموزش چک نمی‌شود، تولید عددهای تصادفی با عدد ۲ بذرپاشی می‌شود، تعداد فیچرها ۱۰۲۴ تا است، لایه نهان اول ۱۲۸ نرون دارد. لایه خروجی ۴۶

نرون دارد. تابع فعال سازی لایه‌ی نهان اول **tanh** است و تابع فعال سازی لایه‌ی خروجی **sigmoid** است، تابع هزینه **Cross Entropy** است، از روش **batch** استفاده شده است.

نکته: در این شبکه چون از **Cross Entropy** استفاده می‌شود خروجی لایه‌ی آخر نرمال سازی می‌شود.

در قسمت‌های قبل نشان دادیم که تابع هزینه‌ی **Cross Entropy** بهتر است به همین دلیل در این شبکه از تابع هزینه‌ی **Cross Entropy** استفاده کردہ‌ام، همین طور بر روی دیتابست رقم‌ها بهترین عملکرد مربوط به شبکه‌های سه لایه شد، به همین دلیل از شبکه‌های سه لایه استفاده کردہ‌ام، در دسته‌بندی رقم‌ها با تعداد حداقل ۶۴ نرون به خطای تست کمتر از سه رسیدیم به همین دلیل تعداد نرون‌ها را زیاد کردہ‌ام و از ۱۲۸ نرون استفاده می‌کنم، روش **steepest descent** با تعداد ایپاک‌های کمی به شدت خطأ را کاهش می‌دهد ولی به علت طولانی بودن از نظر زمان و استفاده از **vectorization** جهت حداکثر کردن سرعت محاسبه در حالت **batch** استفاده می‌کنم. در این روش از **vectorization** استفاده می‌کنم تا سرعت را تا جای ممکن بالا ببرم. ضریب یادگیری و تعداد ایپاک هم با استفاده از آزمون و خطأ به دست آوردم.

فایل کانفیگ شبکه دوم (input-41.json):

```
input-41.json
1  {
2      "digits_letters_digits": "letters_digits",
3      "ratio_of_train_set": 0.80,
4      "ratio_of_valid_set": 0.10,
5      "ratio_of_test_set": 0.10,
6      "learning_rate": 0.1,
7      "max_epochs": 4768,
8      "cut_cost": -1,
9      "random_state": 2,
10     "K_fold": -1,
11     "neurons_layes": [1024, 64, 64, 64, 64, 64, 64, 46],
12     "type_layers": ["tanh", "tanh", "tanh", "tanh", "tanh", "tanh", "sigmoid"],
13     "type_of_cost": "cross_entropy",
14     "mode": "batch"
15 }
```

شبکه‌ی شماره دوم: از دیتاست عده‌ها استفاده می‌شود، ۱۰ درصد دیتاست برای آموزش است، ۱۰ درصد دیتاست برای ارزیابی است، ۱۰ درصد دیتاست برای تست است، ضریب یادگیری برابر با ۰,۱ است، حداکثر تعداد ایپاک‌ها ۴۷۶۸ است، شرط کمتر شدن تابع هزینه از حد مشخصی برای خاتمه‌ی آموزش چک نمی‌شود، تولید عده‌های تصادفی با عدد ۲ بذرپاشی می‌شود، تعداد فیچرها ۱۰۲۴ تا است، شبکه شش لایه‌ی نهان دارد که هر کدام ۶۴ نرون دارند. لایه خروجی ۴۶ نرون دارد. تابع فعال سازی لایه‌های نهان \tanh است و تابع فعال سازی لایه خروجی sigmoid است، تابع هزینه Cross Entropy است، از روش batch استفاده شده است.

نکته: در این شبکه چون از Cross Entropy استفاده می‌شود خروجی لایه‌ی آخر نرمال سازی می‌شود.

در قسمت‌های قبل نشان دادیم که تابع هزینه‌ی Cross Entropy بهتر است به همین دلیل در این شبکه از تابع هزینه‌ی Cross Entropy استفاده کردہ‌ام، همین طور بر روی دیتاست رقم‌ها بهترین عملکرد مربوط به شبکه‌های سه لایه شد، ولی این در شرایطی اتفاق افتاد که شبکه را برای دیتاست رقم‌ها آموزش دادیم و فضای دیتاست به گونه‌ای بود که شبکه‌ی یک لایه ببه خوبی از پس عمل کلاسیفیکیشن بر می‌آمد ولی در این قسمت که دیتاست شامل حروف و رقم‌ها است شبکه‌ی یک لایه مناسب نباشد به همین دلیل از شبکه‌های ۸ لایه با ۶ لایه‌ی نهان استفاده کردہ‌ام، روش steepest descent با تعداد ایپاک‌های کمی به شدت خطرا کاهش می‌دهد ولی به علت طولانی بودن از نظر زمان و استفاده از **vectorization** جهت حداکثر کردن سرعت محاسبه در حالت batch از batch استفاده می‌کنم. در این روش از **vectorization** استفاده می‌کنم تا سرعت را تا جای ممکن بالا ببرم. ضریب یادگیری و تعداد ایپاک هم با استفاده از آزمون و خطای به دست آوردم.

نتیجه‌های آزمایش

نتیجه‌های شبکه عصبی یک:

مقدار تابع هزینه در آخرین ایپاک، خطای آموزش، خطای ارزیابی، خطای تست :

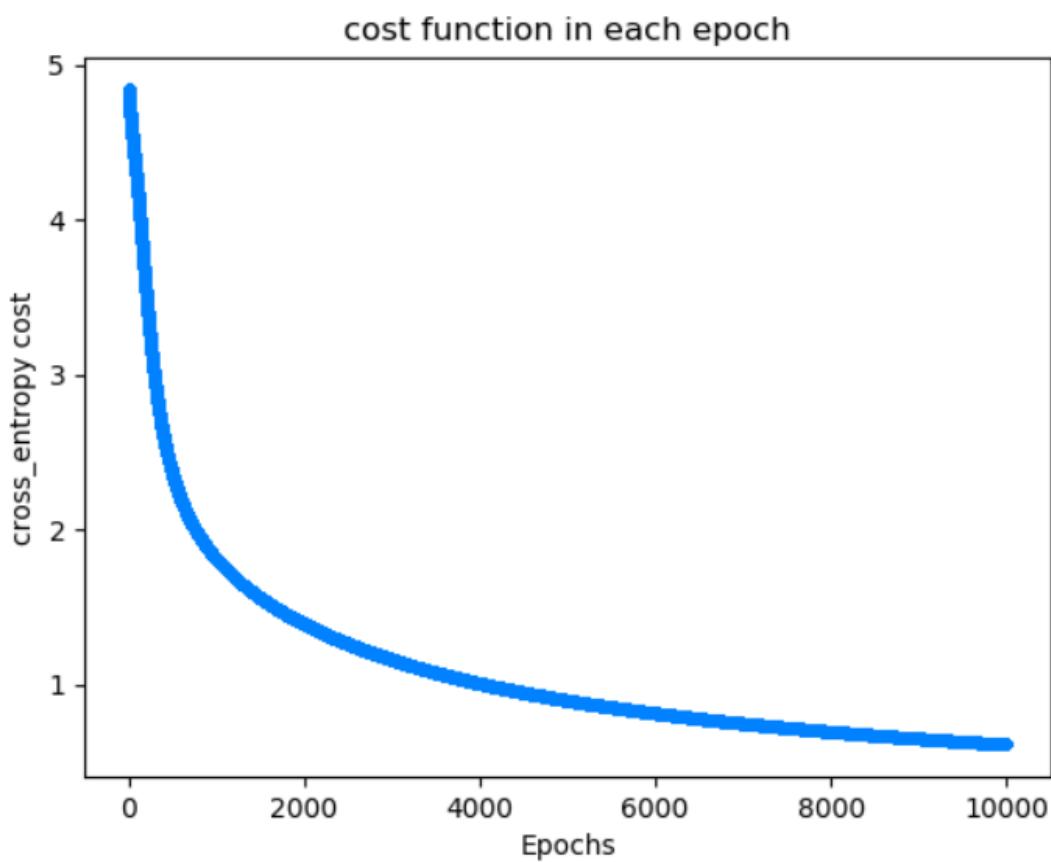
```

Iter 9997/10000, Cost 0.6161154654601126 ...
Iter 9998/10000, Cost 0.6160803824345792 ...
Iter 9999/10000, Cost 0.6160453043565707 ...
Iter 10000/10000, Cost 0.6160102312262551 ...
=====
Train Error: 0.06972506393861892
Validation Error: 0.12365728900255754
Test Error: 0.12148337595907928
=====

```

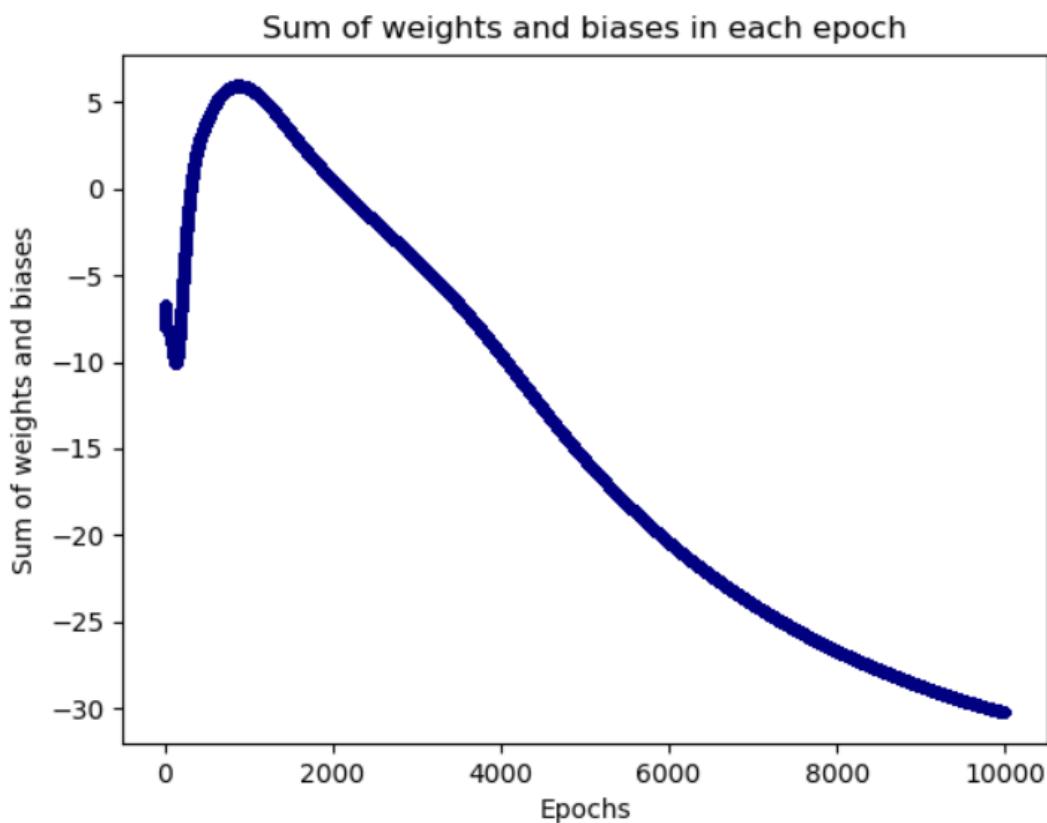
مقدار تابع هزینه بعد از ۲۰۰۰ ایپاک از **4.83** به ۰.۶۱ رسیده است. خطای مجموعه آموزش برابر ۶.۹۷ درصد است، خطای مجموعه ارزیابی برابر با ۱۲.۳۶ درصد است و خطای مجموعه تست برابر با ۱۲.۴۱ درصد است.

مقدار تابع هزینه در هر ایپاک:



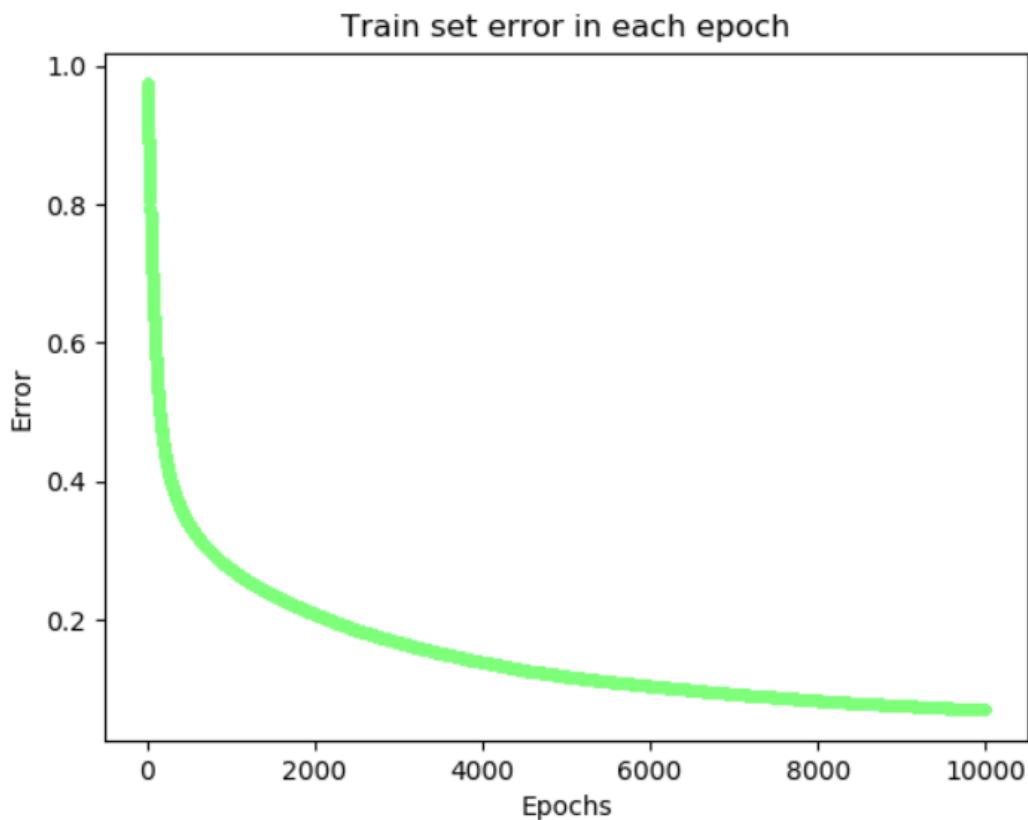
در طی ۱۰۰۰۰ ایپاک تابع هزینه‌ی MSE از **4.83** به ۰.۶۱ رسیده است که میزان کاهش چشم‌گیر است. مقدار تابع هزینه سیر نزولی داشته است و بیشترین تغییرات در زمان انجام ایپاک اول بوده است و هر چه جلوتر رفته‌ایم در ایپاک‌های بعد از میزان تغییرات کاسته شده است و با روند کند تری تابع هزینه کم شده است.

مجموع وزن‌ها و بایاس‌ها در هر ایپاک:



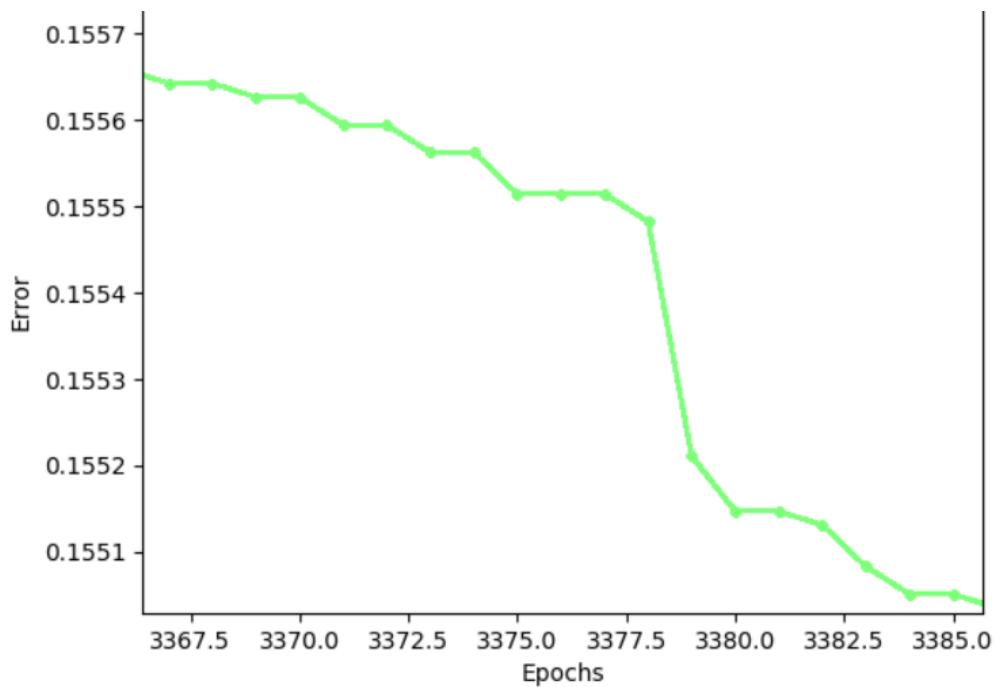
در ایپاک‌های آخر میزان تغییرات مجموعه وزن‌ها و بایاس‌ها کم شده است و تقریباً در آن محدوده باقی می‌ماند و تقریباً در چند ایپاک آخر مقدار مجموع ثابت مانده است. زیرا هر چه جلوتر می‌رویم معمولاً گرادیان‌ها کوچک‌تر می‌شوند و تغییرات وزن‌ها نسبت به مراحل قبل کمتر می‌شود.

خطای مجموعه‌ی آموزش در هر ایپاک:

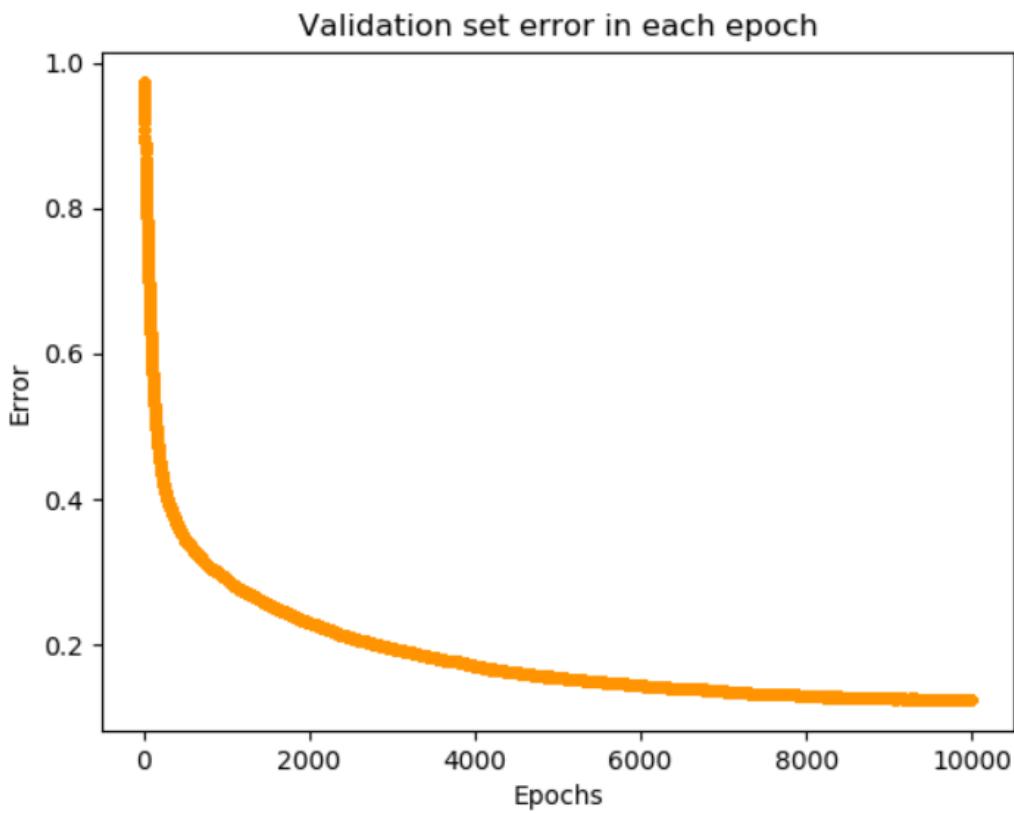


خطای مجموعه آموزش با آموزش بیشتر شبکه کاهش داشته است. بیشتر کاهش در ایپاک اول رخ داده است و در ایپاک‌های آخر میزان تغییرات خطا کم شده است و به مقدار ثابتی میل کرده است.

بخشی کوچکی از نمودار بالا با بزرگنمایی بیشتر:



خطای مجموعه‌ی ارزیابی در هر ایپاک:



خطای ارزیابی در هر ایپاک با آموزش بیشتر کاهش داشته است. در ایپاک‌های اول بیشترین تغییرات را داشته‌ایم و در ایپاک‌های پایانی از میزان تغییرات کاسته شده است.

این شبکه در فایل MLP-output-40.json ذخیره شده است.

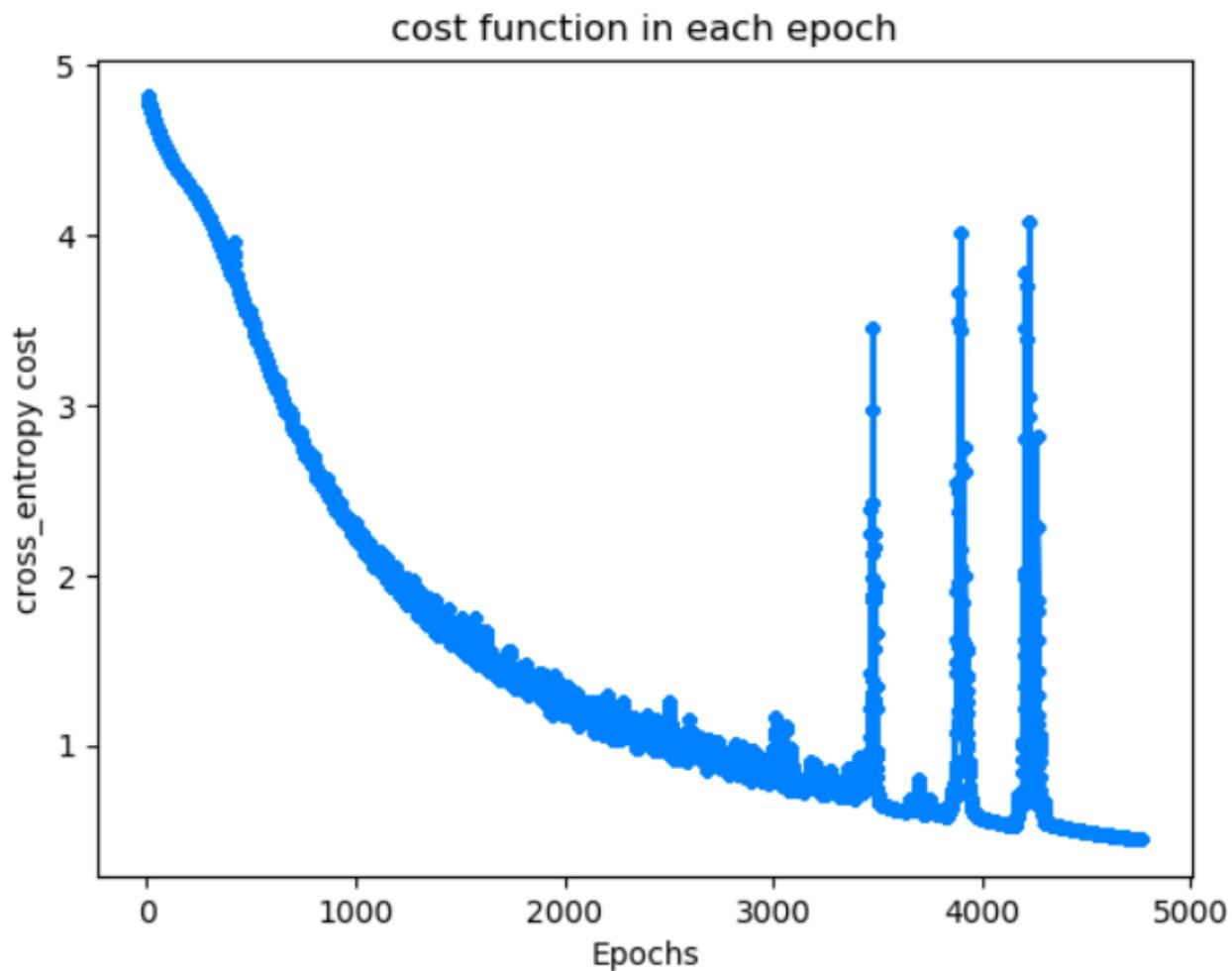
نتیجه‌های شبکه عصبی دوم:

مقدار تابع هزینه در آخرین ایپاک، خطای آموزش، خطای ارزیابی، خطای تست :

```
Iter 4764/4768, Cost 0.4482014083246422 ...
Iter 4765/4768, Cost 0.44805140835173907 ...
Iter 4766/4768, Cost 0.44790744817554046 ...
Iter 4767/4768, Cost 0.4477562344194967 ...
Iter 4768/4768, Cost 0.4476143615972352 ...
=====
Train Error: 0.054923273657289
Validation Error: 0.12915601023017903
Test Error: 0.13439897698209718
=====^=====
```

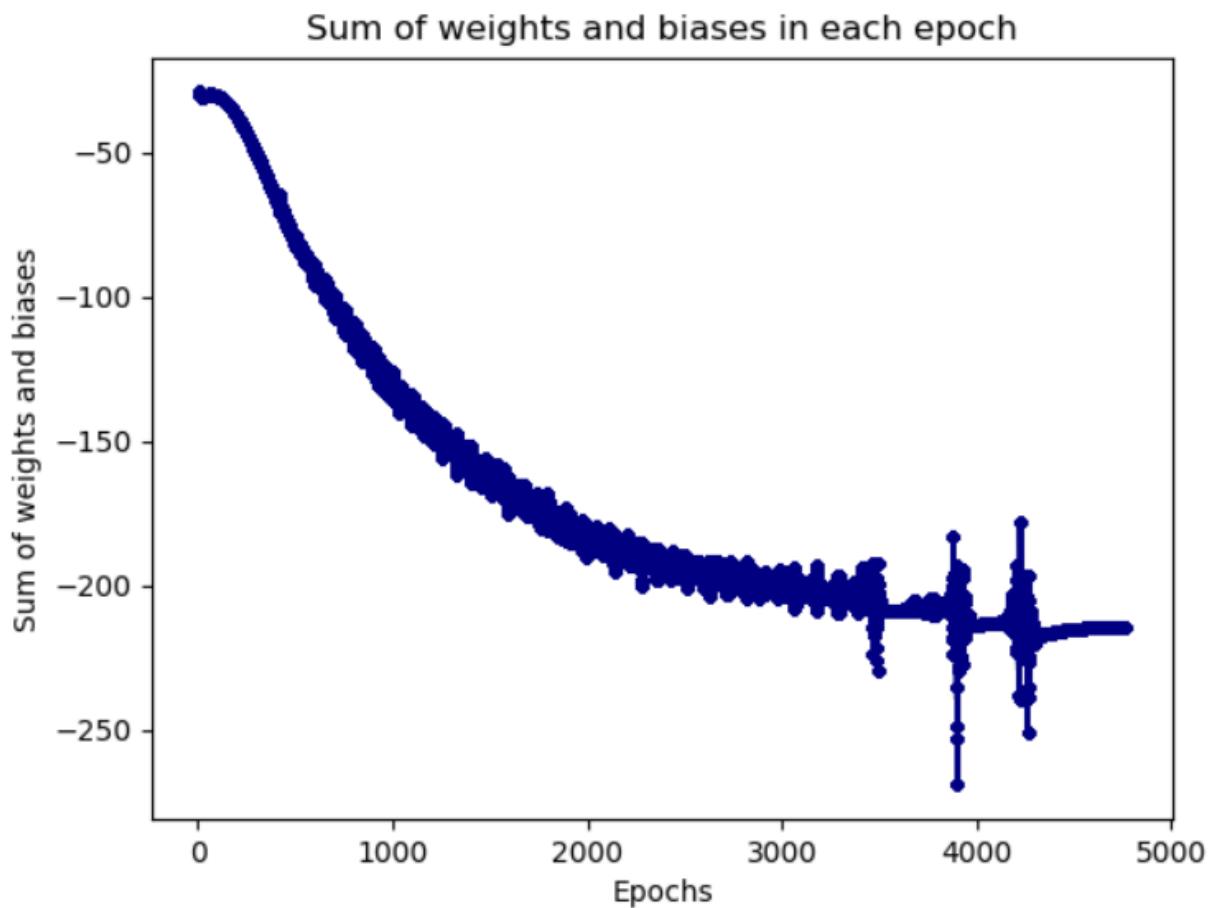
مقدار تابع هزینه بعد از ۵۰۰۰ ایپاک از **4.81** به ۰.۴۴ رسیده است. خطای مجموعه آموزش برابر ۵.۴۹ درصد است، خطای مجموعه ارزیابی برابر با ۱۲.۹۱ درصد است و خطای مجموعه تست برابر با ۱۳.۴۳ درصد است.

مقدار تابع هزینه در هر ایپاک:



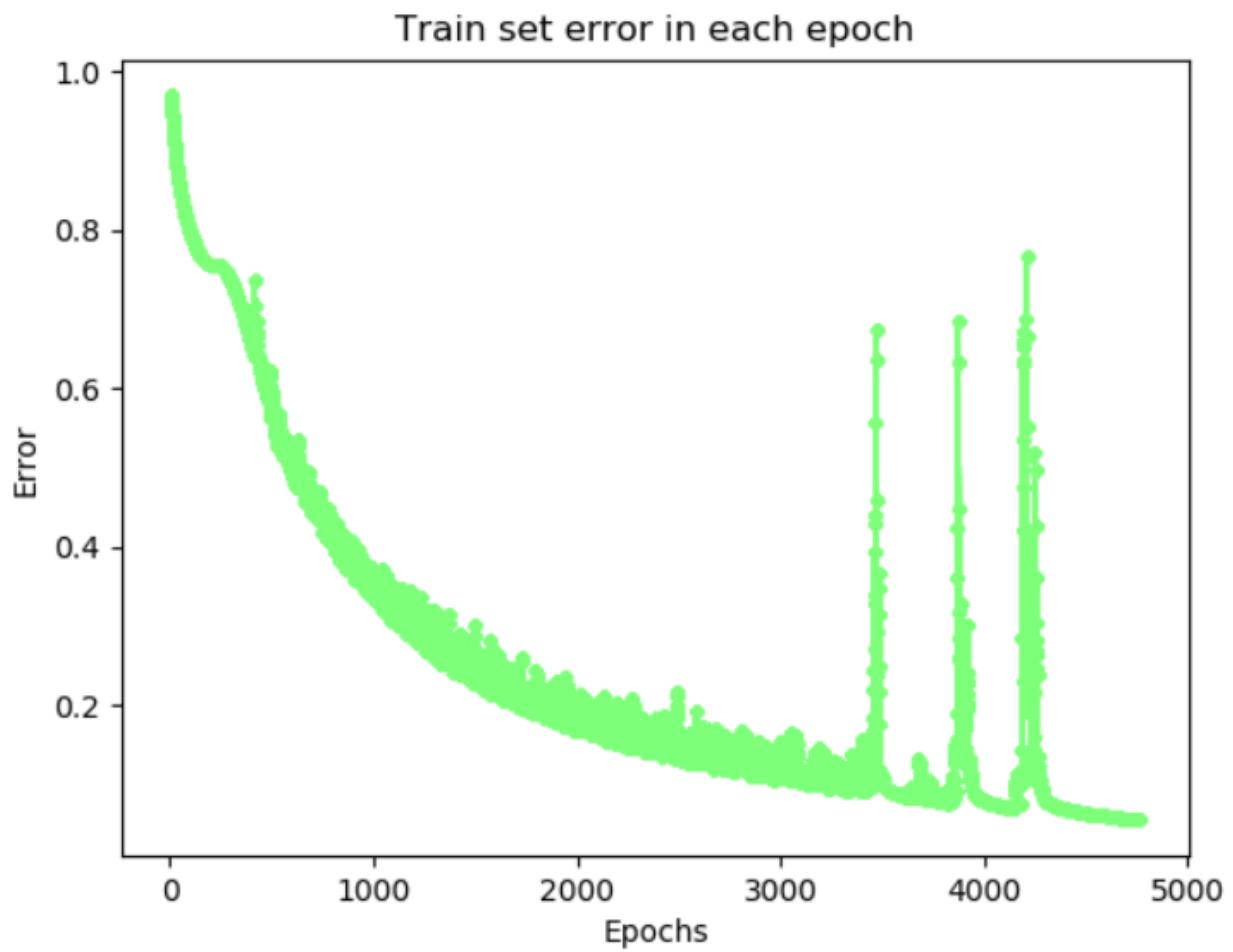
در طی ۱۰۰۰۰ ایپاک تابع هزینه‌ی MSE از **4.81** به ۰.۴۴ رسیده است که میزان کاهش چشم‌گیر است. مقدار تابع هزینه به طور کلی سیر نزولی داشته است و بیشترین تغییرات در زمان انجام ایپاک اول بوده است و هر چه جلوتر رفته‌ایم در ایپاک‌های بعد از میزان تغییرات کاسته شده است و با روند کند تری تابع هزینه کم شده است.

مجموع وزن‌ها و بایاس‌ها در هر ایپاک:



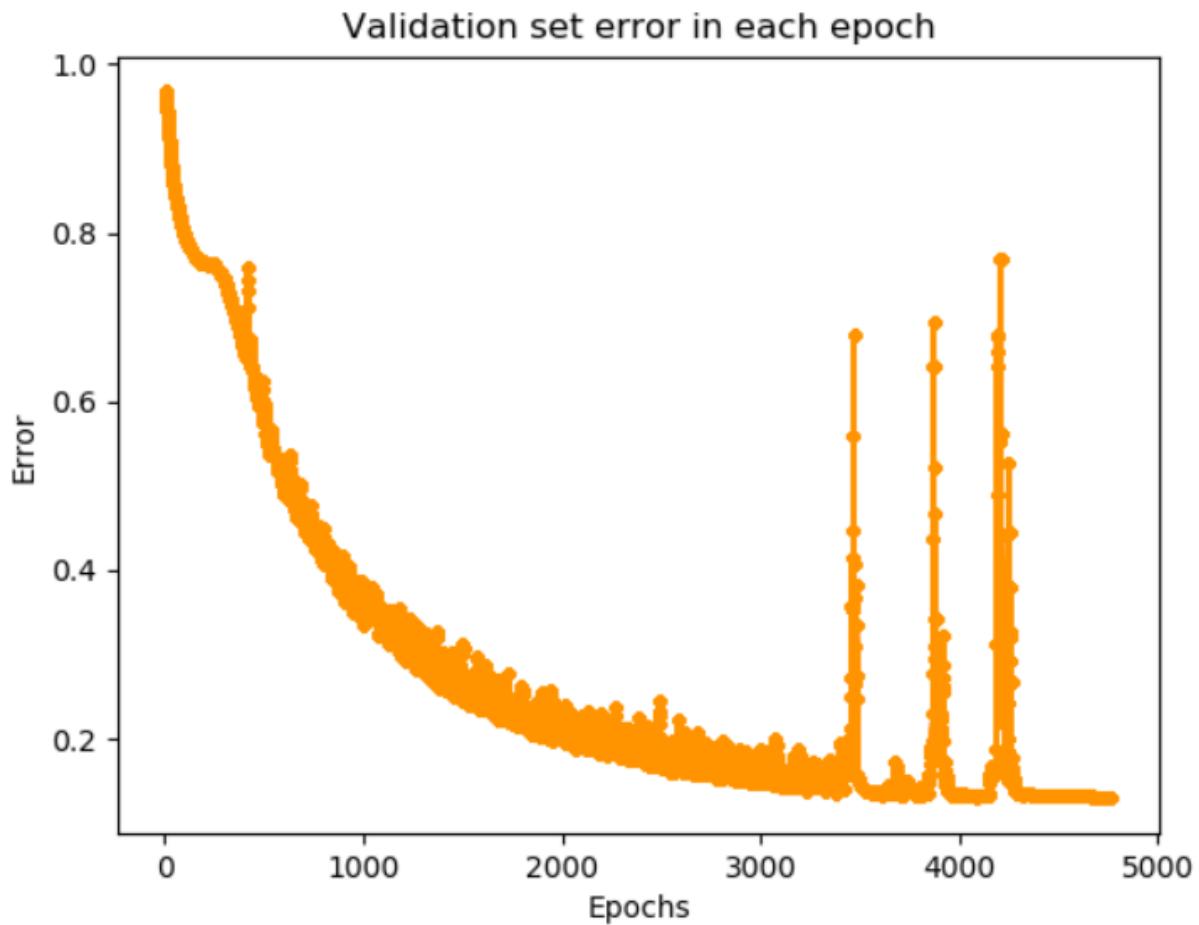
در ایپاک‌های آخر میزان تغییرات مجموعه وزن‌ها و بایاس‌ها کم شده است و تقریباً در آن محدوده باقی می‌ماند و تقریباً در چند ایپاک آخر مقدار مجموع ثابت مانده است. زیرا هر چه جلوتر می‌رویم معمولاً گرادیان‌ها کوچک‌تر می‌شوند و تغییرات وزن‌ها نسبت به مراحل قبل کمتر می‌شود.

خطای مجموعه‌ی آموزش در هر ایپاک:



خطای مجموعه آموزش با آموزش بیشتر شبکه کاهش داشته است. بیشتر کاهش در ایپاک اول رخ داده است و در ایپاک های آخر میزان تغییرات خطا کم شده است و به مقدار ثابتی میل کرده است.

خطای مجموعه ارزیابی در هر ایپاک:



خطای ارزیابی در هر ایپاک با آموزش بیشتر کاهش داشته است. در ایپاک‌های اول بیشترین تغییرات را داشته‌ایم و در ایپاک‌های پایانی از میزان تغییرات کاسته شده است.

این شبکه در فایل MLP-output-41.json ذخیره شده است.

نتیجهگیری

برای این قسمت از سوال دو شبکه آموزش دادم که دلیل ساختار آن‌ها را در بخش شرایط آزمایش در بالاتر توضیح داد. در قسمت نتایج آزمایش نتایج حاصل از هر کدام را توضیح دادم که در تصویر زیر خلاصه‌ای از نتایج‌ها را مشاهده می‌کنید:

خطای آموزش	شبکه‌ی سه لایه با ۱۲۸ نرون در لایه نهان- شبکه‌ی یک دو	شبکه‌ی ۸ لایه با ۶۴ نرون در
خطای ارزیابی	۱۲,۹۱ درصد	۵,۴۹ درصد
خطای تست	۱۲,۱۴ درصد	۱۳,۴۳ درصد

همین طور که مشاهده می‌شود شبکه‌ی سه لایه با اختلافی در حدود یک تا دو درصد بهتر عمل کرده است.