

تمرین سری **یک**

درس شبکه‌های عصبی

فرهاد دلیرانی

۹۶۱۳۱۱۲۵

[dalirani@aut.ac.ir](mailto:dalirani@aut.ac.ir)

[dalirani.1373@gmail.com](mailto:dalirani.1373@gmail.com)

## فهرست

۱.....	ابزارهای استفاده شده ...
۲.....	بخش اول - دسته‌بندی ...
۲.....	خواندن دیتاست ...
۳.....	تابع K Fold
۴.....	پیاده‌سازی Perceptron عادی با یک خروجی باینری ...
۷.....	پیاده‌سازی Perceptrons One Vs All
۱۰.....	ایجاد شی و فراخوانی کلاس (run_perceptrons_one_vs_all.py) perceptrons one vs all
۱۶.....	پیدا کردن پارامترهای مناسب برای perceptron one vs all
۱۸.....	پیاده‌سازی یک آدالاین عادی با خروجی ۱- و ۱-
۲۱.....	پیاده‌سازی adaline one vs all
۲۳.....	ایجاد شی و فراخوانی کلاس ش (run_adaline_one_vs_all.py) adaline one vs all
۲۹.....	پیدا کردن پارامترهای مناسب برای adaline one vs all
۳۱.....	a بررسی عملکرد پرسپترون (one vs all) و آدالاین (one vs all) در حالت خطی ...
۴۳.....	B جدایزیری کلاس‌ها در پرسپترون خطی و بررسی جدایزیری آن‌ها در پرسپترون درجه دو ...
۵۰.....	C بررسی عملکرد الگوریتم‌ها به ازای نرخ‌های یادگیری مختلف ...
۷۳.....	D بررسی تغییر وزن‌ها در کار کرد الگرویتم‌ها ...
۹۵.....	E استفاده از functional link neural network و پرسپترون چندن کلاسه ...
۹۹.....	F استفاده از یکی از ابزارهای موجود sklearn perceptron
۱۰۲.....	بخش دوم - رگرسیون ...
۱۰۲.....	A تخمین MPG با استفاده از ویژگی‌های پیوسته: ...
۱۰۵.....	B تخمین MPG با استفاده از همه‌ی ویژگی‌ها: ...
۱۰۹.....	C تخمین MPG با استفاده از همه‌ی ویژگی‌ها و همین طور termهای درجه دو: ...
۱۱۴.....	D تخمین MPG با استفاده از ویژگی‌های پیوسته و همین طور termهای درجه دو: ...



ابزارهای استفاده شده

زبان برنامه نویسی: پایتون ۳.۶، ۴

محیط توسعه: PyCharm 2017.2.3

سیستم عامل: ویندوز ۱۰

## بخش اول - دسته‌بندی

### خواندن دیتاست

کدهای مربوط به خواندن دیتاست در فایل `read_iris.py` قرار دارد. در این فایل دو تابع برای خواندن دیتاست قرار دارد که در ادامه آن‌ها را شرح می‌دهم.

```
def read_iris_dataset(rand_state=0, train_ratio=0.70, valid_ratio=0.15, test_ratio=0.15, order='1'):
    """
    This function reads Iris dataset, replace string class name with
    0, 1, 2. split data to training, validation and test. it shuffles data.
    :param rand_state: seed for shuffling
    :param train_ratio: portion of data that should be separated for training.
    :param valid_ratio: portion of data that should be separated for validation.
    :param test_ratio: portion of data that should be separated for testing.
    :param order: if degree is '1' read data set and return it, if it's '2'
                  generate order 2 of features (XiXj) then return data set.
    :return: trainX, validX, testX, trainY, validY, testY
    """
    ...
```

این تابع دیتاست موجود در فایل `iris.data` را می‌خواند. آرگمان `rand_state` به عنوان `seed` برای `shuffle` کردن داده‌ها استفاده می‌شود تا بتوان آزمایش را تکرار کرد. آرگمان `train_ratio` نسبتی از دیتاست را نشان می‌دهد که باید به داده‌های آموزش اختصاص داده شود که عددی بین ۰،۰ تا ۱،۰ است. آرگمان `valid_ratio` نسبتی از دیتاست را نشان می‌دهد که باید به داده‌های `validation` و `test` اختصاص داده شوند. آرگمان `order` درجه‌ی داده‌ای را نشان می‌دهد که این تابع بازمی‌گرداند. در صورتی که `order` برابر با "۱" باشد دیتاست بعد از خواندن به سه قسمت `train`, `validation` و `test` تقسیم می‌شود و فیچرهای `train` در متغیر `trainX` قرار می‌گیرد و `label`‌های هر `observation` درون `trainX` در `trainY` قرار می‌گیرد. برای قسمت `test` و `validation` هم به همین ترتیب عمل می‌شود. در صورتی که `order` برابر با "۲" باشد قبل از انجام کارهای حالتی که `order` برابر با ۱ است `feature`‌های مرتبه‌ی دو دیتاست ساخته می‌شود و در کنار `feature`‌های دیتاست قرار می‌گیرد. این فیچرهای این صورت اند:

- $X_i^2$
- $x_i x_j \text{ for } i < j$

خروجی‌ها از نوع کلاس `np.ndarray` هستند.

```

def read_iris_dataset_FLNN(rand_state=0, train_ratio=0.70, valid_ratio=0.15, test_ratio=0.15):
    """
    This function reads Iris dataset, replace string class name with
    0, 1, 2. split data to training and test. also it
    shuffles data. and generate Functional link NN terms
    :param rand_state: seed for shuffling
    :param train_ratio: portion of data that should be separated for training.
    :param valid_ratio: portion of data that should be separated for validation.
    :param test_ratio: portion of data that should be separated for testing.
    :param order: if degree is '1' read data set and return it, if it's '2'
                  generate order 2 of features  $X_i X_j$  then return data set.
    :return: trainX, validX, testX, trainY, validY, testY
    """

```

این تابع هم بسیار شبیه تابع قبلی است با این تفاوت که term های درجه دویی که ایجاد می‌کند به صورت زیر است:

$$X_i X_j \quad i < j$$

### K Fold تابع

تابع k\_fold\_cross\_validation موجود در فایل k\_fold.py مستقل از الگوریتم یادگیری است که از آن استفاده می‌کنیم.

```

def k_fold_cross_validation(learner, argument_of_learner, X, y, k):
    """
    This function uses k-fold-cv to evaluate learner, learner can be
    perceptron, adaline and ....

    :param learner: Is an instance of 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
    """

```

این تابع یک شی از یک الگوریتم یادگیری مانند پرسپترون و یا آدالاین(learner) را می‌گیرد سپس داده‌ها را به k قسمت تقسیم می‌کند و هر بار 1 قسمت را برای ارزیابی استفاده می‌کند و از k-1 قسمت دیگر برای آموزش استفاده می‌کند و در آخر خطای میانگین را به عنوان خطا باز می‌گرداند.

## پیاده‌سازی عادی با یک خروجی باینری Perceptron

برای پیاده‌سازی یک پرسپترون عادی که یک خروجی باینری دارد کلاس `perceptron` را نوشتہام که در فایل `perceptron.py` قرار دارد. در زیر متدهای مختلف این کلاس را شرح می‌دهم. این کلاس به صورت جنرال نوشته شده است و هر دیتاستی با تعداد فیچرهای مختلف را به آن بدھیم برای آن مشکلی پیش نمی‌آید.

```
def __init__(self, learning_rate=0.01, max_epoch=100, cut_error=None):
    """
    This function is constructor of Perceptron which sets
    different parameter of Perceptron.
    :param learning_rate: learning rate of Perceptron for updating weights
    :param max_epoch: maximum number of iterations that perceptron is
                      allowed to use for training
    :param: cut_error: stop training if error fall under specific value
    """
```

این تابع `constructor` کلاس `perceptron` است. آرگمان `learning_rate` ضریب یادگیری پرسپترون است که از آن برای بهروز رسانی وزن‌ها استفاده می‌شود. آرگومان `max_epoch` حداکثر `epoch` هایی را مشخص می‌کند که یک شی از کلاس پرسپترون اجازه دارد در حین آموزش انجام دهد. اگر `cut_error` برابر `None` باشد نادیده گرفته می‌شود و در صورتی که مقادرش بین  $0, 1$  باشد در صورتی که حین آموزش پرسپترون خطای پرسپترون کمتر از آن شود، آموزش متوقف می‌شود.

```
def net_input(self, X):
    """
    Net input of perceptron for instance X, which equal to: transpose(W)*X+W[0]
    :param X: Input instance
    :return: net input of perceptron
    """
```

این متد از کلاس پرسپترون با گرفتن یک ورودی `Net Input` پرسپترون را به صورت زیر محاسبه می‌کند:

$$\text{net input} = w^T x + w_0$$

```

def activation(self, net_inp):
    """
    This function calculate activation of perceptron(hardLim),
    if net input is smaller than zero, returns 1 otherwise returns 0
    :param net_inp: net input of perceptron
                    np.array[net input1,...,net inputM]
    :return: activation of perceptron
    """

```

این تابع با گرفتن Net Input یک پرسپترون، خروجی Activation پرسپترون را به صورت زیر حساب می‌کند:

$$\text{Activation} = \begin{cases} 1 & \text{net} - \text{input} \geq 0 \\ 0 & \text{net} - \text{input} < 0 \end{cases}$$

```

def predict(self, X):
    """
    This function predict output of perceptron for instance X
    :param X: input instance.
    :return: corresponding Label
    """

```

این تابع یک نمونه ورودی را می‌گیرد و با استفاده از دو تابع predict و net\_input برچسب کلاس را محاسبه می‌کند.

```

def fit(self, X, y, validX=None, validY=None, class_name="", rand_state=0, plotting=False):
    """
    This method of perceptron class gets training dataset and
    its labels, then it trains perceptron. after doing "max_epoch" epochs,
    it will stop training.
    :param X: Training set features: numpy array of numpy arrays
              [[feature1 x1, feature2 x2,...,featuren xn],
               ...,
               [feature1 xm, feature2 xm,...,featuren xm]]
    :param y: Training set labels: numpy array
              [[label x1], [label x2], ..., [label xn]]
    :param validX: validation set (optional)
    :param validY: labels of validation set (optional)
    :param rand_state: seed for generating random number for weights
    :param plotting: if it's true, this function plots
                     cost and weights during learning
    :return: self
    """

```

این تابع با گرفتن مجموعه‌ی آموزش  $X$  و برچسب‌های آن ( $y$ ) پرسپترون را آموزش می‌دهد و وزن‌های پرسپترون را آپدیت می‌کند.  $validX$  و  $validY$  مجموعه‌ی ارزیابی هستند که در صورتی که به تابع داده شده باشند عمکرد پرسپترون در حین آموزش بر روی آن‌ها سنجیده می‌شود.  $Class\_name$  یک رشته است که در صورت که به تابع داده شود در حین نمایش نمودارها نام کلاس را هم در بالای نمودار می‌نویسد، مانند `Iris-setosa`. آرگمان `Rand_state` آرگمان ورودی‌ای است که برای `seed` دادن به تولید عددهای تصادفی استفاده می‌شود. در این تابع وزن‌های پرسپترون به صورت تصادفی ایجاد می‌شود برای اینکه بتوانیم آزمایش‌ها را تکرار کنم از `seed` استفاده کرده‌ام. در صورتی که آرگمان `Plotting` برابر با `True` باشد در حین آموزش پرسپترون نمودارهای وزن‌های پرسپترون، خطای پرسپترون در هر اپوک بر روی مجموعه‌ی آموزش و ارزیابی رسم می‌شود. برای آپدیت وزن‌ها به ازای یک ورودی بدین صورت عمل کرده‌ام:

```
y_hat = predict(x)
weights[1::] += learning_rate * (y - y_hat) * x
weights[0] += learning_rate * (y - y_hat)
```

این کار به ازای همه‌ی نمونه‌ها و اپوک‌های مختلف تکرار می‌شود.

```

def load_weights(self, weights):
    """
    Load weights of perceptron
    :param weights: a list in form [w0, w1, w2, ..., wn]
    :return:
    """

    self.__weights = np.array(weights)
    return self

def get_learning_rate(self):
    """
    Return Learning Rate
    """
    return self.__learning_rate

def get_weights(self):
    """
    Return Weights
    """
    return self.__weights

def get_valid_error_in_epochs(self):
    """
    Return error in each epochs
    """
    return self.__valid_errors

def get_error_in_epochs(self):
    """
    Return error in each epochs
    """
    return self.__errors

```

تابع‌های بالا هم ویژگی‌های مختلف پرسپترون مانند ضریب یادگیری، وزن‌ها خطای پرسپترون بر روی مجموعه‌ی آموزش و ارزیابی و ... را `set` می‌کنند یا بر می‌گردانند. در ادامه‌ی کد یک کد کوچک برای تست درستی کلاس نوشته‌ام.

### پیاده‌سازی One Vs All

برای پیاده‌سازی لایه‌ای از پرسپترون‌ها که به صورت One Vs All یک مسیله‌ی چند کلاسه را حل می‌کند، کلاس `perceptron_one_vs_all.py` را نوشته‌ام که در فایل `perceptron_one_vs_all` موجود است.

این کلاس از کلاس پرسپترون عادی با یک خروجی ۰ و ۱ که در بالا آن را معرفی کردم استفاده می‌کند و به کمک بیش از یک پرسپترون یک مسیله‌ی چند کلاسه را حل می‌کند.

این کلاس را به صورت جنرال نوشتہام و محدودیت نسبت به دیتاست و یا تعداد برچسب‌های مختلف ندارد.

در زیر متدهای مختلف این کلاس را توضیح می‌دهم.

```
def __init__(self, number_classes, learning_rate=0.01, max_epoch=100, cut_error=None):
    """
    :param number_classes: number of different classes(labels)
    :param learning_rate: learning rate for training Perceptrons of one vs all
    :param max_epoch: maximum number of epochs that perceptron is
                      allowed to use for training
    :param cut_error: stop training if error fall under specific
                      value, if it's none don't consider it
    """
```

این متد constructor کلاس پرسپترون به روش one vs all است که ویژگی‌های مختلف را set می‌کند. آرگمان number\_classes تعداد برچسب‌ها را در مسیله‌ی چند کلاس را نشان می‌دهد. آرگمان learning\_rate ضریب یادگیری هر پرسپترون را مشخص می‌کند. آرگمان max\_epoch مشخص می‌کند هر پرسپترون حداقل مجاز به انجام چند epoch است. آرگمان cut\_error در صورتی که none نباشد و یک عدد بین صفر و یک باشد در حین آموزش پرسپترون‌ها در صورتی که خطای یک پرسپترون از cut\_error کم شود آموزش آن پرسپترون متوقف می‌شود. در این تابع به تعداد number\_classes پرسپترون ساده با خروجی باینری ساخته می‌شود.

```
def fit(self, X, y, validX=None, validY=None, classes_name=None, rand_state=0, plotting=False, write_in_file=False):
    """
    Train perceptrons by using on vs all. y should contains labels 1,2,..., number_classes
    :param X: Training set features: numpy array of numpy arrays
              [[feature1 x1, feature2 x2,...,featuren xn],
               ...,
               [feature1 xm, feature2 xm,...,featuren xm]]
    :param y: Training set labels: numpy array
              [[label x1], [label x2], ..., [label xn]]
              labels shoud be 1,2, ..., number_of_classes
    :param validX: validation set (optional)
    :param validY: labels of validation set (optional)
    :param classes_name: name of classes(optional). ['class1 name',..., 'class2 name']
    :param rand_state: for seeding while generating weights
    :param plotting: if it's true, this function plots
                     errors of train and validation also, weights during Learning
    :param write_in_file: if true, it writes weights of perceptrons in perceptron_weights.json
    :return:
    """
```

این تابع مجموعه‌ی آموزش ( $X$ ) و برچسب‌هایی (y) را می‌گیرد به ازای هر کلاس برچسب‌های آن کلاس در دیتاست را برابر با ۱ قرار می‌دهد و برچسب‌های کلاس‌های دیگر را برابر ۰ قرار می‌دهد. سپس پرسپترون هر کلاس را که شی‌ای از کلاس **perceptron** است را با استفاده از تابع **fit** کلاس پرسپترون که در بالا شرح دادیم، آموزش می‌دهد. هر پرسپترون مستقل از سایر پرسپترون‌ها آموزش داده می‌شود. سایر آرگمان‌ها را در متدهای **fit** کلاس **perceptron** توضیح داده شده‌اند. در صورتی که مقدارش **true** باشد وزن‌های **perceptron** ذخیره می‌شوند.

```
def load_perceptorns_from_file(self, file_name):
    """
    This function read weights of trained perceptorns from file
    :param file_name: it's the name of a json file like: 'input.json'.
                      file should be in same directory with code
    :return:
    """
```

این تابع نام کامل یک فایل json که در محل کد است را می‌گیرد و وزن‌های پرسپترون‌های تعدادی پرسپترون که به صورت **one vs all** عمل می‌کنند را می‌خواند و وزن پرسپترون‌ها را بازگذاری می‌کند.

```
def predict(self, x):
    """
    This function predict output of perceptorns for instance x.
    it uses one vesus all. it determinse label of instance by using
    maximum net input of perceptorns
    :param x: input instance.
    :return: a number as label
    """
```

این تابع یک ورودی می‌گیرد و باید مشخص کند نمونه‌ی  $x$  به کدام کلاس  $1, 2, \dots, \text{number\_classes}$  تعلق دارد. برای این کار ورودی  $x$  را به پرسپترون‌ها می‌دهد و برچسب کلاس برابر می‌شود با کلاس پرسپترونی که بزرگ‌ترین **net input** را برای آن نمونه می‌دهد.

در ادامه‌ی کد یک کد کوچک برای تست درستی کلاس نوشته‌ام.

ایجاد شی و فراخوانی کلاس (run\_perceptrons\_one\_vs\_all.py) `perceptrons one vs all` در فایل `run_perceptrons_one_vs_all.py` یک رابط کاربری متنی ایجاد کرده‌ام که به سه روش پارامترها را از کاربر می‌گیرد:

- پارامترهای مورد نیاز برای خواندن دیتاست و ایجاد یک لایه پرسپترون One vs all را به صورت متنی در ترمینال از کاربر می‌گیرد.
- پارامترها را که در شکل یک فایل json می‌گیرد
- وزن‌های یک لایه پرسپترون One vs all را که در یک فایل json است را می‌گیرد.

```
C:\Users\Home\AppData\Local\Programs\Python\Python36\python.exe "C:/Users/Home/Dropbox/code
How do you want to give learning parameters to the program?
> 1: I give parameters via Standard Input / I have trained perceptrons in a file
> 2: I give parameters via a file
Input 1 or 2: |
```

این پارامترها شامل موردهای زیر است:

- نسبتی از دیتاست که باید صرف مجموعه‌ی آموزشی شود
- نسبتی از دیتاست که باید صرف مجموعه‌ی ارزیابی شود
- نسبتی از دیتاست که باید صرف مجموعه‌ی تست شود
- درجه‌ی یک بودن فیچرها یا فیچرها حاوی فیچرهای درجه دو تولید شده هم باشند
- حداکثر تعداد ایپوکها
- ضریب یادگیری
- نحوه‌ی پایان آموزش
- Seed برای تولید اعداد تصادفی و تکرار پذیری مسیله
- تعداد fold ها برای k-fold

در صورتی که از یکی از دو حالت اول استفاده کرده باشیم، دیتاست را می‌خواند و آن را با روش اسکیل می‌کند و سپس پرسپترون‌ها را آموزش می‌دهد و در حین آموزش خطای تست و ارزیابی و وزن‌های هر پرسپترون را نمایش می‌دهد و در آخر خطا/دقت مجموعه‌ی آموزش و ارزیابی و K-fold و تست را گزارش می‌کند. در صورتی که از روش سوم استفاده کنیم از آنجایی که فاز آموزش نداریم و وزن‌های یادگرفته شده را بارگذاری می‌کنیم فقط خطای مجموعه‌ی آموزش و تست گزارش می‌شود.

در ادامه یک نمونه از فراخوانی‌های کلاس یک در مقابل همه را با استفاده از کد و رابط کاربری موجود در run\_perceptrons\_one\_vs\_all.py را ارایه می‌کنم.

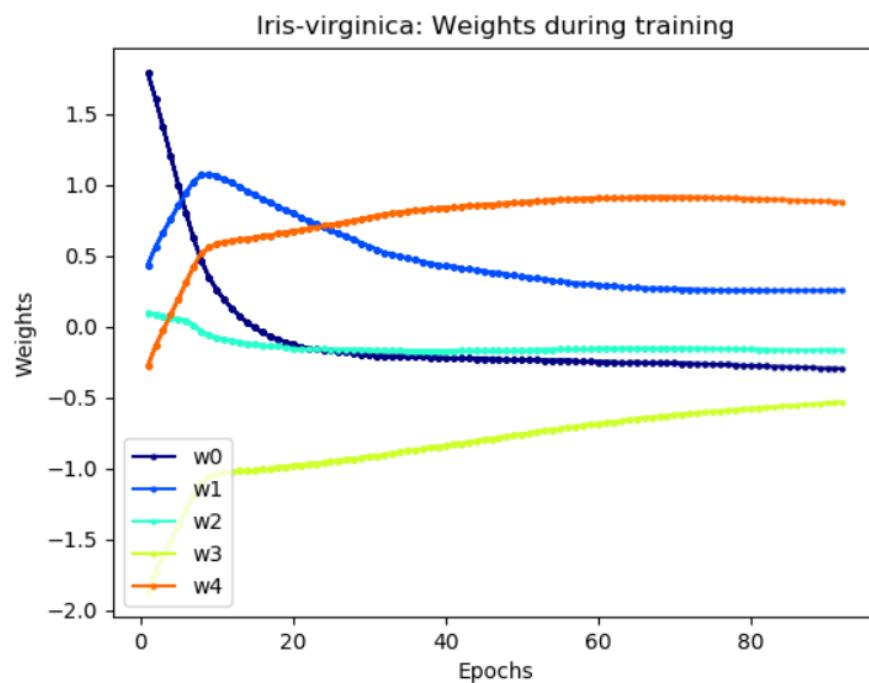
برای نمونه به فراخوانی زیر را انجام داده ام:

پارامترهای فراخوانی اول در فایل input-perceptrons-one-vs-all-2.json موجود است:

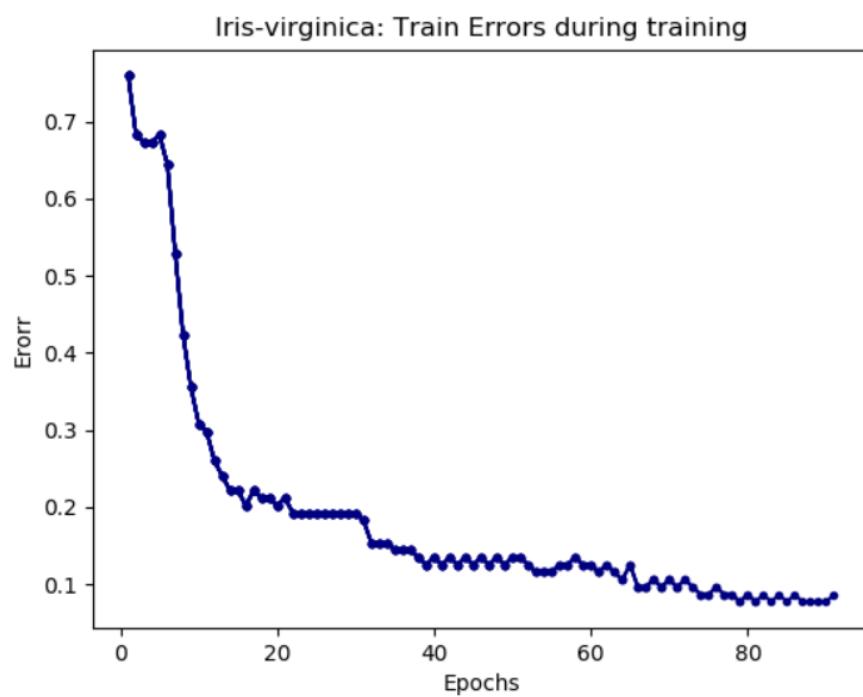
```
input-perceptrons-one-vs-all-2.json
1  {
2      "ratio_of_train_set":0.70,
3      "ratio_of_valid_set":0.15,
4      "ratio_of_test_set":0.15,
5      "learning_rate":0.003,
6      "max_epochs":100,
7      "cut_error": -1,
8      "random_state":3,
9      "K_fold": 10,
10     "order":"1"
11 }
```

ضریب یادگیری برابر با  $0.003$  است، حداکثر تعداد اپوکها برابر با  $100$  است، برای پایان آموزش فقط رسیدن به حداکثر تعداد اپوکها چک می‌شود، برای K-Fold تعداد  $10$  در نظر گرفته می‌شود، از فیچرها بدون اضافه کردن فیچرهای مرتبه‌ی  $2$  استفاده می‌شود،  $70$  درصد داده‌ها به آموزش  $15$  درصد به ارزیابی و  $15$  درصد به تست اختصاص پیدا می‌کند. از seed  $3$  برای کار با عده‌های تصادفی و تولید وزن‌ها استفاده شده است.

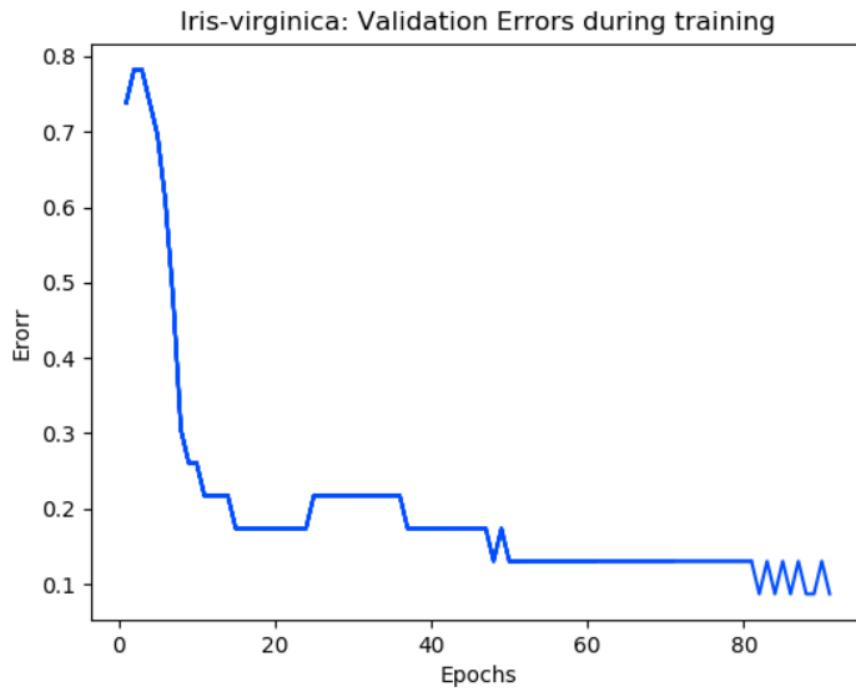
تغییر وزن‌ها در هر ایپاتک برای پرسپترون کلاس Iris-virginica



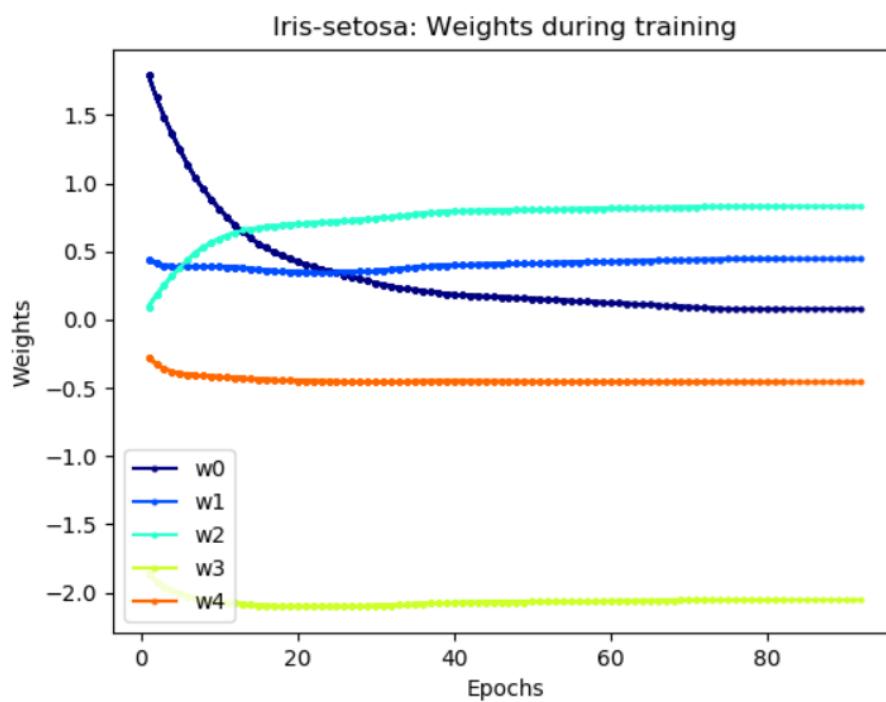
تغییر خطای مجموعه آموزش در هر ایپاک برای پرسپترون کلاس Iris-virginica



تغییر خطای مجموعه ارزیابی در هر ایپاک برای پرسپترون کلاس Iris-virginica



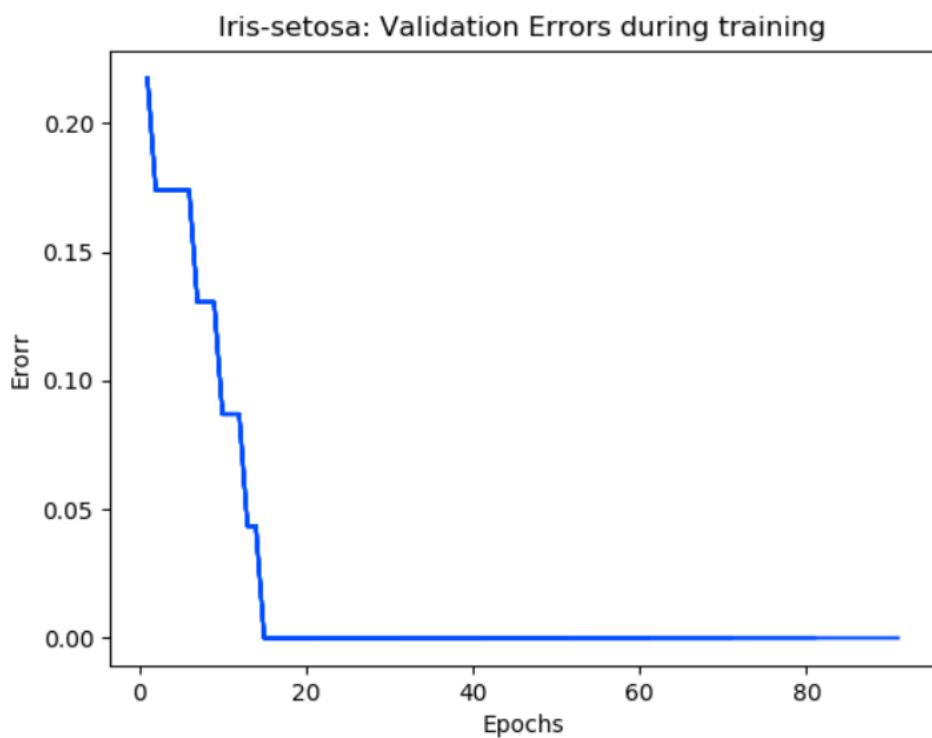
تغییر وزن‌ها در هر ایپاک برای پرسپترون کلاس Iris-setosa



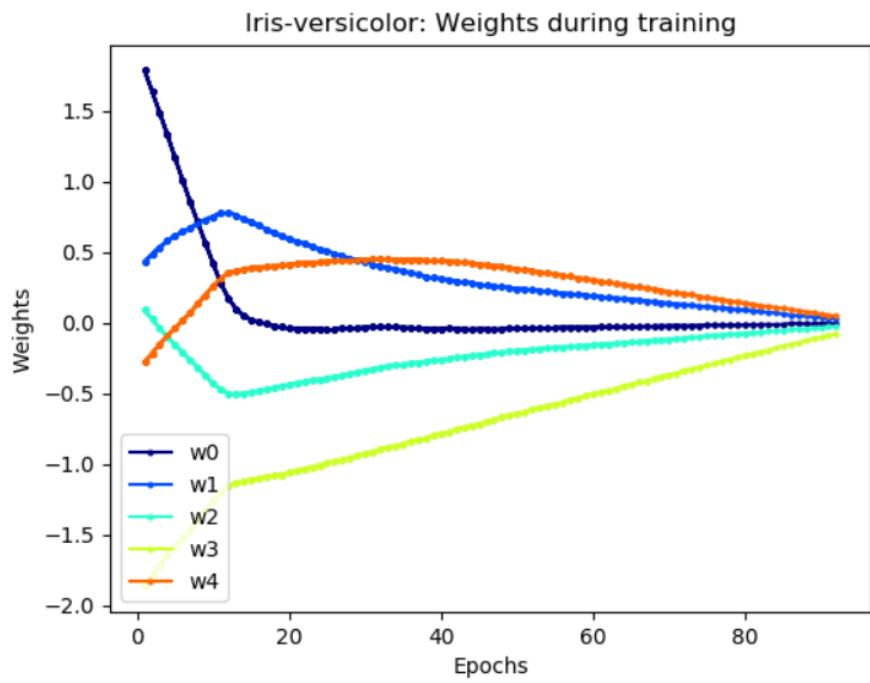
تغییر خطای مجموعه آموزش در هر ایپاک برای پرسپترون کلاس Iris-setosa



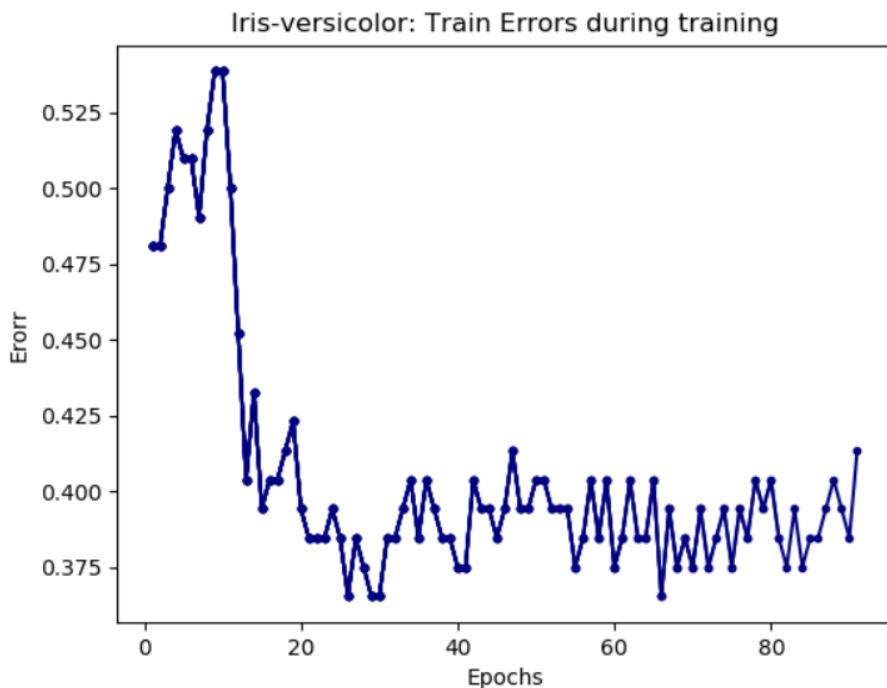
تغییر خطای مجموعه ارزیابی در هر ایپاک برای پرسپترون کلاس Iris-setosa



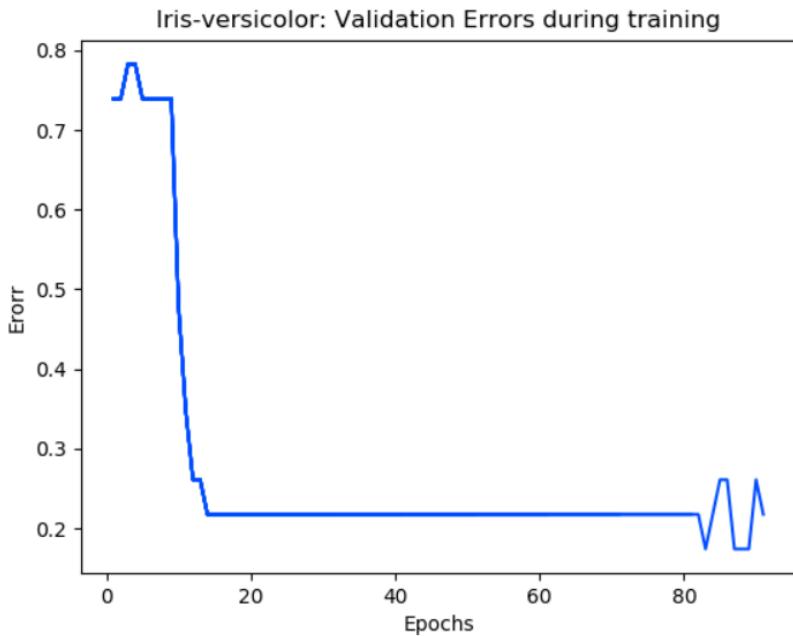
تغییر وزن‌ها در هر ایپاک برای پرسپترون کلاس Iris-versicolor



تغییر خطای مجموعه آموزش در هر ایپاک برای پرسپترون کلاس Iris-versicolor



تغییر خطای مجموعه ارزیابی در هر ایپاک برای پرسپترون کلاس Iris-versicolor



خطای k-fold و خطای عادی آموزش و خطای تست در پایان:

```
Calculating 10-Fold Cross Validation ...
10-fold-cross-validation error is: 0.12857142857142861
Train Error(Ordinary Error, not k-fold)= 0.07692307692307693
Test Error= 0.043478260869565216
```

[پیدا کردن پارامترهای مناسب برای perceptron one vs all](#)

در فایل `best_perceptron_one_vs_all.py` لایه‌ای از پرسپترون‌های یک برابر همهی کلاس `perceptron one vs all` را با پارامترهای مختلف فراخوانی می‌کنم تا پارامترایی کناسب برای دست یابی به عملکردی مناسب پیدا کنم. خروجی آزمایش‌های مختلف در فایل `find_good_perceptrons.txt` نوشته می‌شود. در تصویر زیر بخشی از این فایل را مشاهده می‌کنید:

find\_good\_perceptrons.txt - Notepad

File Edit Format View Help

VV=====VVV=====VV  
learning\_rate: 0.02200000000000002  
random state: 5  
ratio of train set: 0.7  
10-fold-cross-validation error is: 0.05  
Train Error(Ordinary Error, not k-fold)= 0.028846153846153848  
Test Error= 0.1739130434782608  
=====^

VV=====VVV=====VV  
learning\_rate: 0.025  
random state: 5  
ratio of train set: 0.7  
10-fold-cross-validation error is: 0.05  
Train Error(Ordinary Error, not k-fold)= 0.028846153846153848  
Test Error= 0.21739130434782608  
=====^

VV=====VVV=====VV  
learning\_rate: 0.028  
random state: 5  
ratio of train set: 0.7  
10-fold-cross-validation error is: 0.05  
Train Error(Ordinary Error, not k-fold)= 0.038461538461538464  
Test Error= 0.1739130434782608  
=====^

VV=====VVV=====VV  
learning\_rate: 0.001  
random state: 6  
ratio of train set: 0.7  
10-fold-cross-validation error is: 0.12428571428571429  
Train Error(Ordinary Error, not k-fold)= 0.07692307692307693  
Test Error= 0.08695652173913043  
=====^

VV=====VVV=====VV  
learning\_rate: 0.004  
random state: 6  
ratio of train set: 0.7  
10-fold-cross-validation error is: 0.06  
Train Error(Ordinary Error, not k-fold)= 0.038461538461538464  
Test Error= 0.0

## پیاده‌سازی یک آدالاین عادی با خروجی 1- و 1-

در فایل `adaline.py` کلاسی به نام `adaline` پیاده سازی کرده‌ام که یک کلاس برای `adaline` با خروجی 1- و 1 است در زیر متد‌های این کلاس را توضیح می‌دهم. این کلاس به صورت جنرال نوشته شده است و به دیتابست و یا تعداد مشخصی فیچر وابسته نیست.

```
def __init__(self, learning_rate=0.01, max_epoch=100, cut_error=None):
    """
    This function is constructor of adaline which sets
    different parameter of adaline.
    :param learning_rate: learning rate of adaline
    :param max_epoch: maximum number of iteration that adaline is
                      allowed to use for training
    :param: cut_error: stop training if mse error fall under specific value
    """
```

این متد سازنده‌ی کلاس `adaline` است که پارامترهای مختلف یک آدالاین را مقدار دهی می‌کند. `max_epoch` ضریب یادگیری آدالاین است که در آپدیت وزن‌ها از آن استفاده می‌کند. `Learning_rate` حداقل ایپاک‌ها را مشخص می‌کند. در صورتی که `cut_error` برابر با `None` نباشد علاوه بر شرط ایپاک‌ها برای توقف آموزش در صورتی که خطأ در حین آموزش از `cut_error` کمتر شود آموزش آدالاین متوقف می‌شود.

```
def net_input(self, X):
    """
    Net input of adaline for instance X, which equal: transpose(W)*X+W[0]
    :param X: Input instance
    :return: net input of adaline
    """
```

این متد یک ورودی می‌گیرد و بر اساس رابطه‌ی زیر آدالاین را محاسبه می‌کند.

$$\text{net input} = w^T x + w_0$$

```

def predict(self, X):
    """
    This function predict output of adaline for instance X
    :param X: input instance(s).
    :return: corresponding activation function
    """
    net_inputs = self.net_input(X)
    #activation_out = self.activation(net_inp=net_inputs)
    activations_out = np.where(net_inputs >= 0.0, 1, -1)

    return activations_out

```

این تابع یک نمونه می‌گیرد و با استفاده از متدهای `network input` و `net_input` آدالین را محاسبه می‌کند در صورتی که بزرگ‌تر مسای صفر باشد برچسب آن برابر با ۱ می‌شود در غیر این صورت برچسب آن منفی یک می‌شود.

```

def fit(self, X, y, validX=None, validY=None, class_name="", rand_state=0, plotting=False):
    """
    This method of adaline class gets training dataset and
    its label, then it trains adaline. after doing "_max_epoch" epochs,
    it stops training, or when it fall under cut off error.
    :param X: Training set features: numpy array of numpy arrays
              [[feature1 x1, feature2 x2,...,featuren xn],
               ...,
               [feature1 xm, feature2 xm,...,featuren xm]]
    :param y: Training set labels: numpy array
              [[label x1], [label x2], ..., [label xn]]
    :param validX: validation set (optional)
    :param validY: labels of validation set (optional)
    :param rand_state: seed for random number
    :param plotting: if it's true, this function plots
                     cost and weights during learning
    :return: self
    """

```

این متدهای کلاس کار آموزش آدالین را انجام می‌دهد. این متدهای آموزشی `X` و `y` را می‌گیرد سپس وزن‌ها را به صورت تصادفی و با استفاده از تابع توزیع نرمال استاندارد ایجاد می‌کند، سپس طبق رابطه زیر به ازای یک ورودی وزن‌ها را با روش گرادیان نزولی به روزرسانی می‌کند:

`weights[1::] += learning rate * (target - y hat) * x`

`weights[0] += learning rate * (target - y hat)`

و این کار را برای تمام داده‌ها و به ازای ایپاک‌های مجاز انجام می‌دهد. در صورتی که validX و validY داده شده باشند در حین آموزش نمودار خطای آdalain بر روی مجموعه ارزیابی در هر ایپاک رسم می‌شود. Class\_name یک رشته است که یک آرگمن اخیر است. در صورتی که داده شود در حین رسم نمودارها اسم کلاس در بالای نمودارها رسم می‌شود. برای کنترل تولید وزن‌ها به صورت تصادفی و تکرار پذیر کردن آزمایش استفاده می‌شود. در صورتی که plotting مقدارش True باشد در حین آموزش نمودار خطای آموزش، ارزیابی و تغییر وزن‌ها بر روی صفحه رسم می‌شود.

```
def load_weights(self, weights):
    """
    Load weights of adaline from file
    :param weights: a list in form [w0, w1, w2]
    :return:
    """
    self._weights = np.array(weights)
    return self

def get_learning_rate(self):
    """Return Learning Rate"""
    return self._learning_rate

def get_weights(self):
    """Return Weights"""
    return self._weights

def get_valid_error_in_epochs(self):
    """Return error in each epochs"""
    return self._valid_errors

def get_error_in_epochs(self):
    """Return error in each epochs"""
    return self._errors
```

این کلاس تابع‌های دیگری هم دارد که ویژگی‌های مختلف کلاس را set یا برمی‌گرداند. در ادامهی کد، کدی برای تست درستی این کلاس نوشته‌ام.

## پیاده‌سازی adaline one vs all

در فایل `adaline_one_vs_all.py` کلاس `adaline_one_vs_all` قرار دارد که از  $n$  آدالاین برای حل یک مسیله‌ی  $n$  برچسبه استفاده می‌کند. پیاده‌سازی این کلاس به صورت جنرال است و از دیتاست و تعداد فیچرها و کلاس‌های مختلف مستقل است. در زیر متدهای مختلف این کلاس را توضیح می‌دهم.

```
def __init__(self, number_classes, learning_rate=0.1, max_epoch=200, cut_error=None):
    """
    :param number_classes: number of different classes
    :param learning_rate: learning rate of adaline
    :param max_epoch: maximum number of epochs that adaline is
                      allowed to use for training
    :param cut_error: stop training if error fall under specific
                      value, if it's none don't consider it
    """
```

این تابع سازنده این کلاس است و تعداد کلاس‌ها (برچسب‌ها) در مسیله‌ی چند کلاسه را مشخص می‌کند. ضریب یادگیری هر آدالاین معمولی با خروجی  $-1$  و  $1$  در لایه‌ای از آدالاین‌ها است. `learning_rate` مشخص می‌کند هر آدالاین موجود در روش `one vs all` حداقل چند ایپاک می‌تواند آموزش `max_epoch` ببیند. در صورتی که `None` نباشد در صورتی که در طول آموزش خطا به کمتر از `cut_error` برسد آموزش متوقف می‌شود.

```
def fit(self, X, y, validX=None, validY=None, classes_name=None, rand_state=0, plotting=False, write_in_file=False):
    """
    Train adalines by using on vs all. y should contains labels 1,2,..., number_classes
    :param X: Training set features: numpy array of numpy arrays
              [[feature1 x1, feature2 x2,...,featuren xn],
               ...,
               [feature1 xm, feature2 xm,...,featuren xm]]
    :param y: Training set labels: numpy array
              [[label x1], [label x2], ..., [label xn]]
              Labels shoud be 1,2, ..., number_of_classes
    :param validX: validation set (optional)
    :param validY: labels of validation set (optional)
    :param classes_name: name of classes(optional). ['class1 name',..., 'class2 name']
    :param rand_state: for seeding while generating weights
    :param plotting: if it's true, this function plots
                     cost and weights during learning
    :param write_in_file: if true, write weights of adalines in adaline_weights.json
    :return:
    """
```

این تابع آموزش لایه‌ای از adaline های عادی با یک خروجی -1 و 1 را انجام می‌دهد. X و y داده‌های آموزش اند، در روش one vs all به تعداد برجسب‌های مختلف آدالاین‌های مستقل از هم داریم که برای آموزش هر کدام برچسب داده‌های مربوط به کلاس آدالاین را برابر با 1 قرار می‌دهیم و سایرین را برابر با منفی یک قرار می‌دهیم. و هر کدام را با استفاده از تابع fit کلاس adaline عادی آموزش می‌دهیم.

```
def load_adalines_from_file(self, file_name):
    """
    This function read weights of trained adalines from file
    :param file_name: it's name of a jason file with format like: 'input.json'
                      with code should be in same directory
    :return:
    """
    # read weights from file
    with open(file_name) as json_file:
        weights = json.load(json_file)

    # read adalines from file
    for i in range(0, self.__number_of_class):
        self.__adalines[i].load_weights(weights=weights[i])
    return self
```

وزن‌های آدالاین‌های یک لایه one vs all که آموزش دیده شده‌اند را می‌خواند و در آدالاین‌ها بارگذاری می‌کند.

```
def predict(self, x):
    """
    This function predict output of adalines(one vs all) for instance x.
    it uses one versus all.
    :param x: input instance.
    :return: a number as label
    """
```

این تابع یک نمونه می‌گیرد و برای آن نمونه net input همه‌ی آدالاین‌ها را محاسبه می‌کند و برچسب نمونه برابر با کلاس آدالاینی می‌شود که بزرگ‌ترین net input را داشته است.

در ادامه‌ی کد، کدی برای سنجش درستی کلاس نوشته‌ام.

ایجاد شی و فراخوانی کلاس ش all (run\_adaline\_one\_vs\_all.py) adaline one vs all در فایل run\_adaline\_one\_vs\_all.py یک رابط کاربری متنی ایجاد کرده‌ام که به سه روش پارامترها را از کاربر می‌گیرد:

- پارامترهای مورد نیاز برای خواندن دیتاست و ایجاد یک لایه آadalain One vs all را به صورت متنی در ترمینال از کاربر می‌گیرد.
- پارامترها را که در شکل یک فایل json می‌گیرد
- وزن‌های یک لایه آadalain one vs all را که در یک فایل json است را می‌گیرد.

```
C:\Users\Home\AppData\Local\Programs\Python\Python36\python.exe "C:/Users/Home/Dropbox/code
How do you want to give learning parameters to the program?
> 1: I give parameters via Standard Input / I have trained adalines in a file
> 2: I give parameters via a file
Input 1 or 2:
```

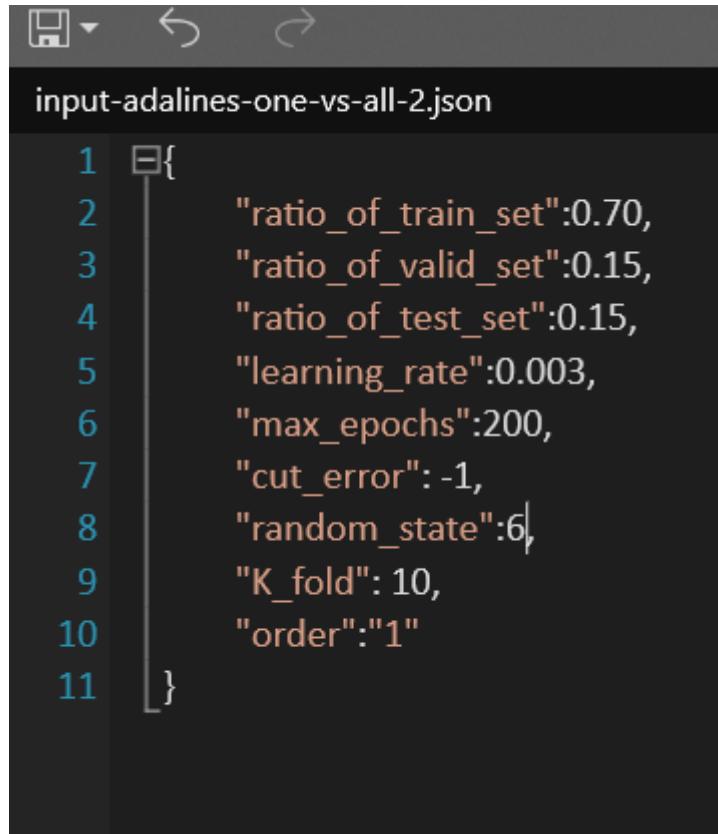
این پارامترها شامل موردهای زیر است:

- نسبتی از دیتاست که باید صرف مجموعه‌ی آموزشی شود
- نسبتی از دیتاست که باید صرف مجموعه‌ی ارزیابی شود
- نسبتی از دیتاست که باید صرف مجموعه‌ی تست شود
- درجه‌ی یک بودن فیچرها یا فیچرها حاوی فیچرهای درجه دو تولید شده هم باشند
- حداکثر تعداد ایپوکها
- ضریب یادگیری
- نحوه‌ی پایان آموزش
- Seed برای تولید اعداد تصادفی و تکرار پذیری مسیله
- تعداد fold ها برای k-fold

در صورتی که از یکی از دو حالت اول را استفاده کرده باشیم، دیتاست را می‌خواند و آن را با روش standardization اسکیل می‌کند و سپس آdalain‌ها را آموزش می‌دهد و در حین آموزش خطای تست و ارزیابی و وزن‌های هر آdalain را نمایش می‌دهد و در آخر خطای دقت مجموعه‌ی آموزش و ارزیابی و K-fold و تست را گزارش می‌کند. در صورتی که از روش سوم استفاده کنیم از آنجایی که فاز آموزش نداریم و وزن‌های یادگرفته شده را بارگذاری می‌کنیم فقط خطای مجموعه‌ی آموزش و تست گزارش می‌شود.

در ادامه چند نمونه از فراخوانی‌های کلاس یک در مقابل همه را با استفاده از کد و رابط کاربری موجود در `run_adalines_one_vs_all.py` را ارایه می‌کنم.

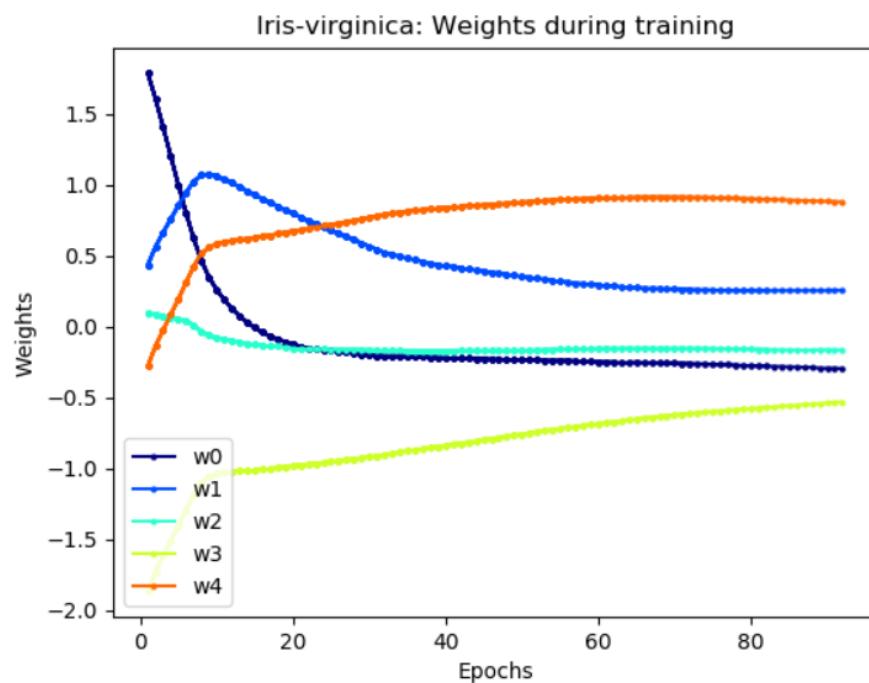
برای نمونه به فراخوانی زیر را انجام داده ام:  
پارامترهای فراخوانی اول در فایل `input-adalines-one-vs-all-1.json` موجود است:



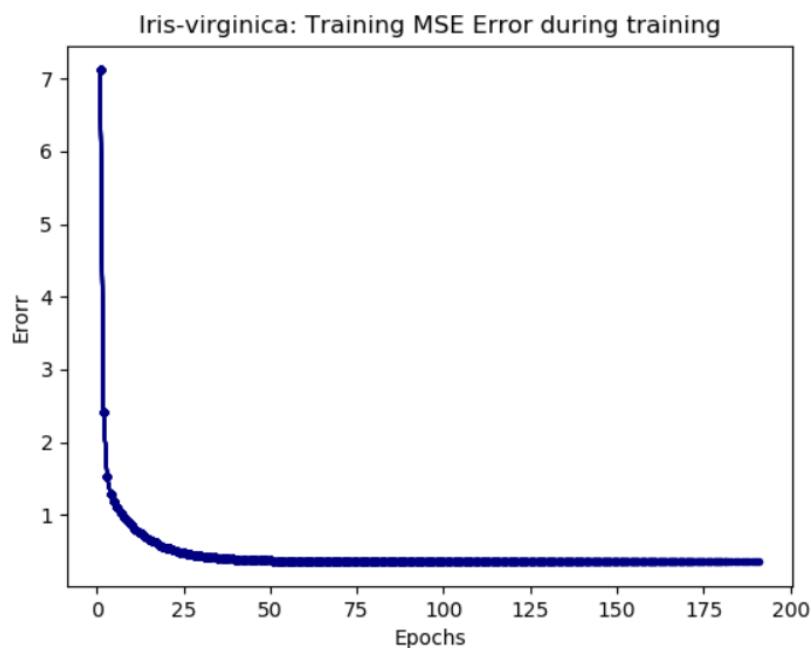
```
input-adalines-one-vs-all-2.json
1  {
2      "ratio_of_train_set":0.70,
3      "ratio_of_valid_set":0.15,
4      "ratio_of_test_set":0.15,
5      "learning_rate":0.003,
6      "max_epochs":200,
7      "cut_error": -1,
8      "random_state":6,
9      "K_fold": 10,
10     "order":"1"
11 }
```

ضریب یادگیری برابر با  $0.003$  است، حداکثر تعداد اپوکها برابر با  $200$  است، برای پایان آموزش فقط رسیدن به حداکثر تعداد اپوکها چک می‌شود، برای K-Fold تعداد  $10$  در نظر گرفته می‌شود، از فیچرها بدون اضافه کردن فیچرهای مرتبه‌ی  $2$  استفاده می‌شود،  $70$  درصد داده‌ها به آموزش  $15$  درصد به ارزیابی و  $15$  درصد به تست اختصاص پیدا می‌کند. از `seed 6` برای کار با عده‌های تصادفی و تولید وزن‌ها استفاده شده است.

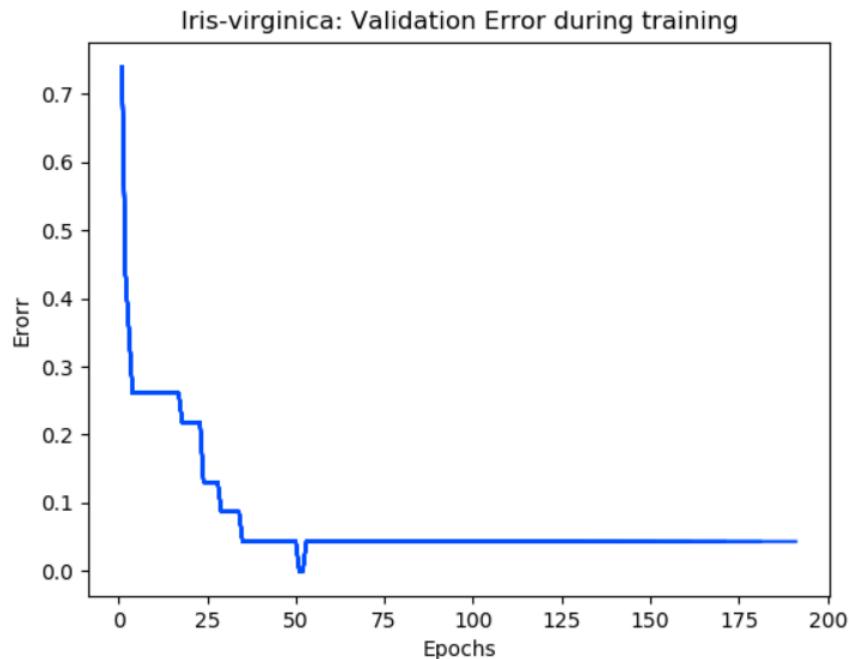
تغییر وزن‌ها در هر ایپاتک برای آadaline کلاس `Iris-virginica`



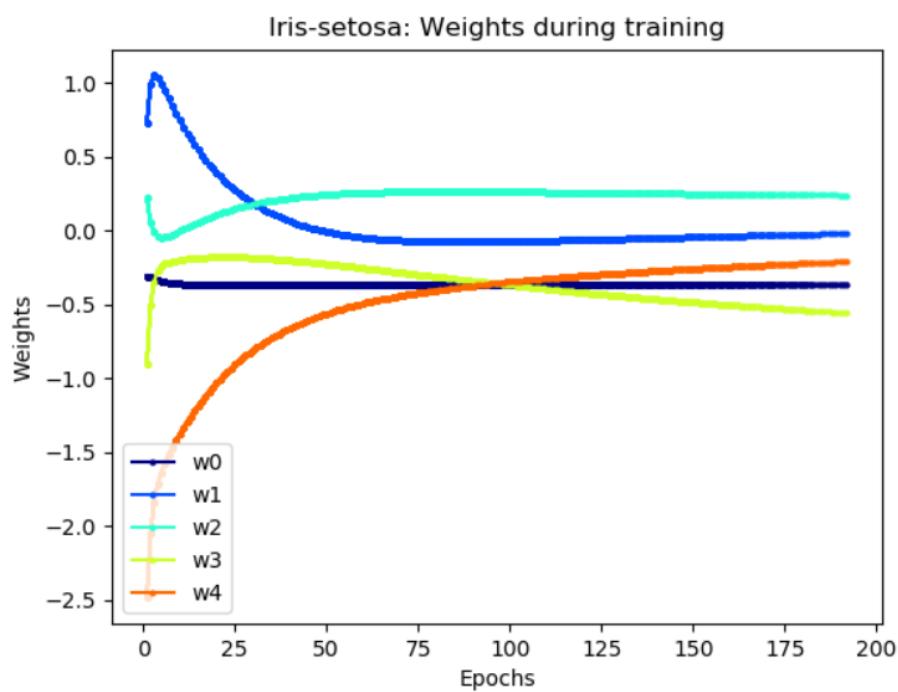
تغییر خطای مجموعه آموزش در هر ایپاک برای آدالین کلاس Iris-virginica



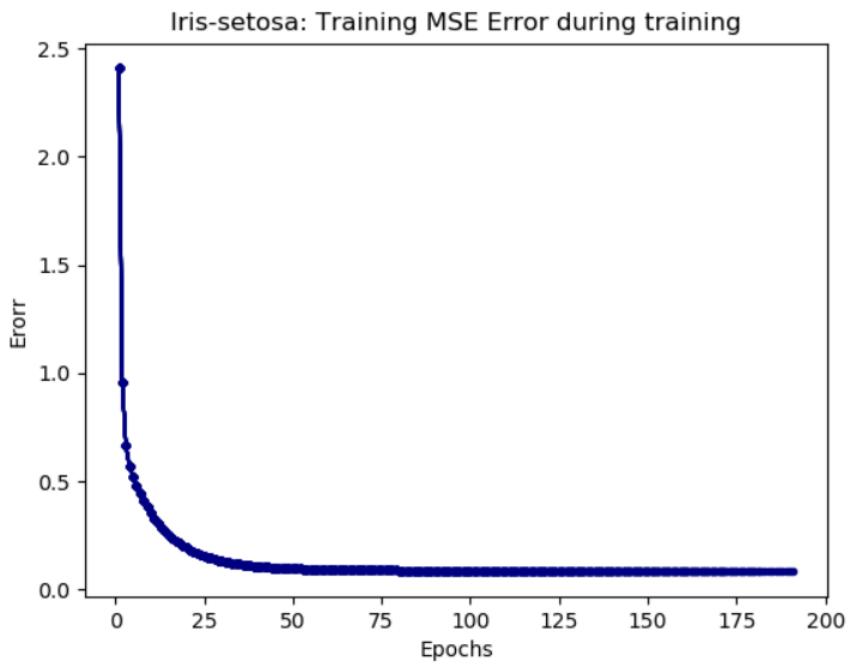
تغییر خطای مجموعه ارزیابی در هر ایپاک برای آدالین کلاس Iris-virginica



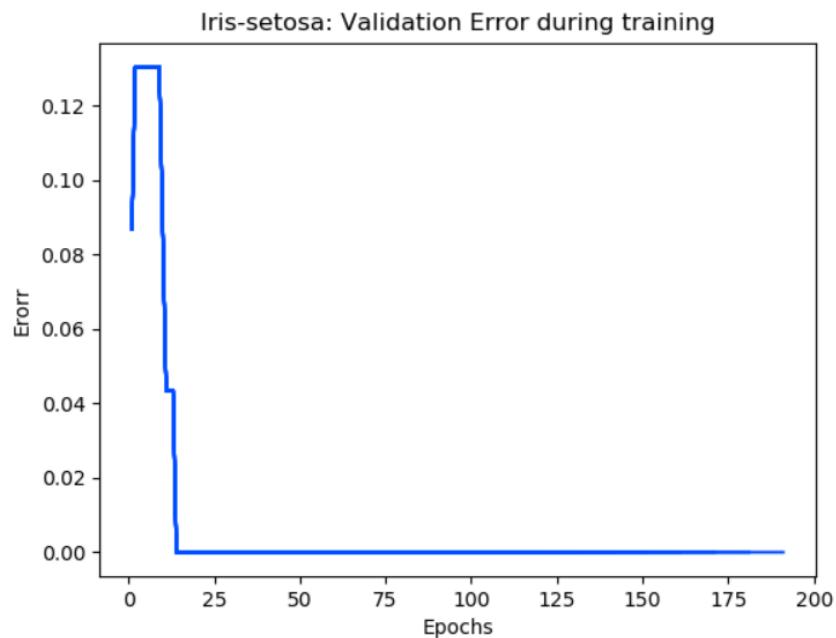
تغییر وزن‌ها در هر ایپاک برای آدالین کلاس Iris-setosa



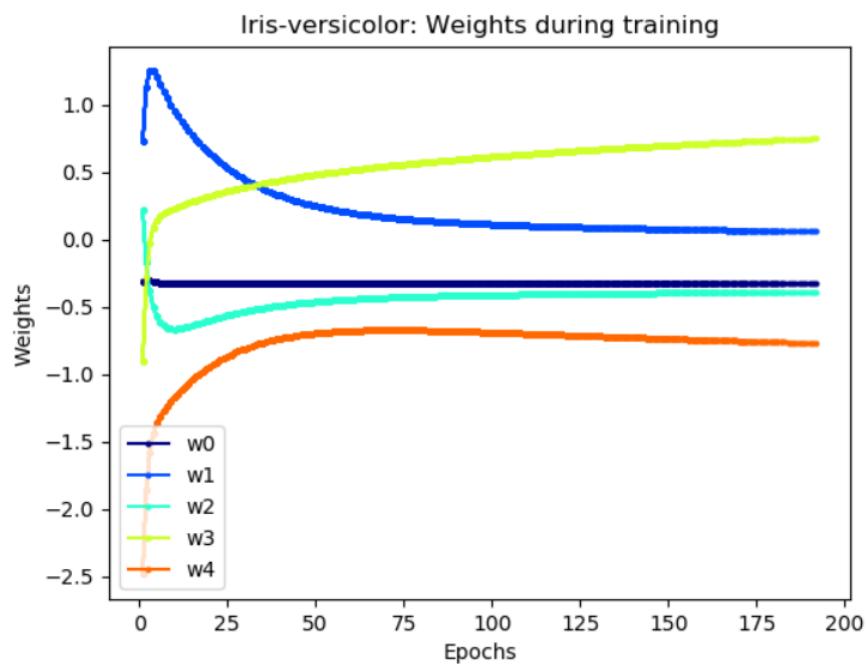
تغییر خطای مجموعه آموزش در هر ایپاک برای آدالین کلاس Iris-setosa



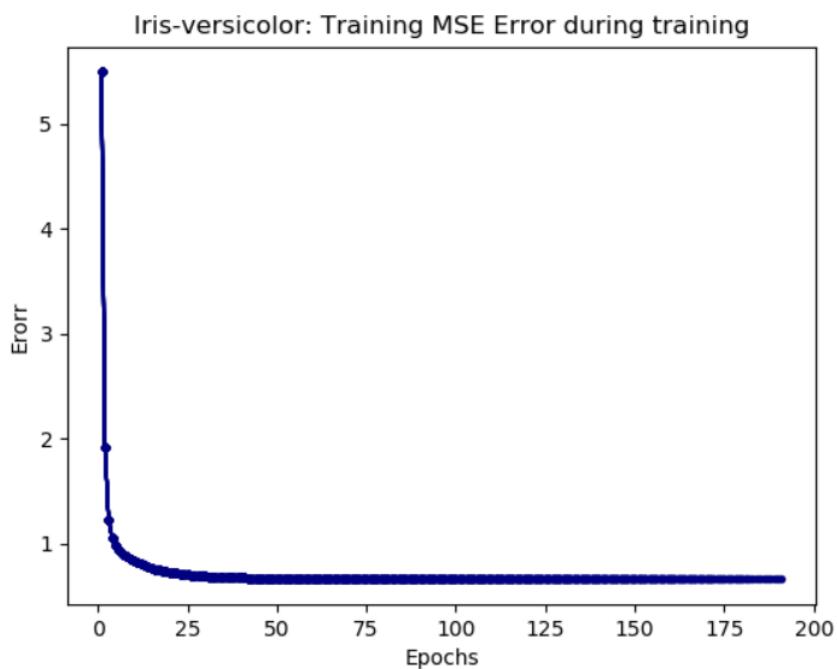
تغییر خطای مجموعه ارزیابی در هر ایپاک برای آدالین کلاس Iris-setosa



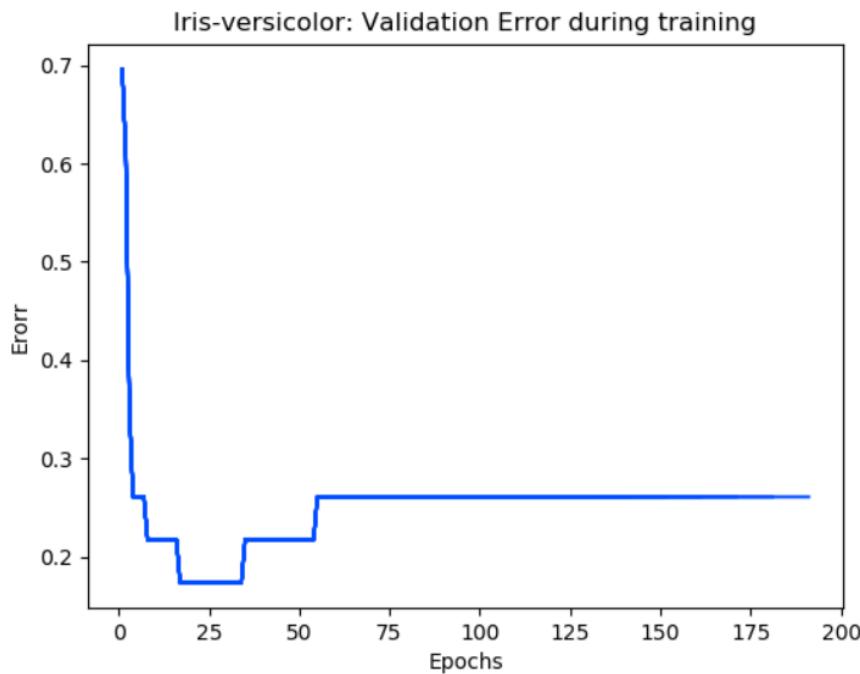
تغییر وزن‌ها در هر ایپاک برای آدالین کلاس Iris-versicolor



تغییر خطای مجموعه آموزش در هر ایپاک برای آدالاین کلاس Iris-versicolor



تغییر خطای مجموعه ارزیابی در هر ایپاک برای آدالاین کلاس Iris-versicolor



خطای k-fold و خطای عادی آموزش و خطای تست در پایان:

```
Calculating 10-Fold Cross Validation ...
10-fold-cross-validation error is: 0.18142857142857144
Train Error(Ordinary Error, not k-fold)= 0.1346153846153846
Test Error= 0.13043478260869565
```

پیدا کردن پارامترهای مناسب برای adaline one vs all در فایل `best_adaline_one_vs_all.py` لایهای از پرسپترون‌های یک برابر همه‌ی، کلاس adalines one vs all را با پارامترهای مختلف فراخوانی می‌کنم تا پارامترهایی مناسب برای دست یابی به عملکردی مناسب پیدا کنم. خروجی آزمایش‌های مختلف در فایل `find_good_adalines.txt` نوشته می‌شود. در تصویر زیر بخشی از این فایل را مشاهده می‌کنید:

find\_good\_adalines.txt - Notepad

File Edit Format View Help

```
random state: 2
ratio of train set: 0.7
10-fold-cross-validation error is: 0.13714285714285718
Train Error(Ordinary Error, not k-fold)= 0.11538461538461539
Test Error= 0.21739130434782608
=====
VV=====VV=====
learning_rate: 0.017
random state: 2
ratio of train set: 0.7
10-fold-cross-validation error is: 0.13714285714285718
Train Error(Ordinary Error, not k-fold)= 0.11538461538461539
Test Error= 0.21739130434782608
=====
VV=====VV=====
learning_rate: 0.019000000000000003
random state: 2
ratio of train set: 0.7
10-fold-cross-validation error is: 0.12714285714285717
Train Error(Ordinary Error, not k-fold)= 0.11538461538461539
Test Error= 0.17391304347826086
=====
VV=====VV=====
learning_rate: 0.001
random state: 3
ratio of train set: 0.7
10-fold-cross-validation error is: 0.21428571428571427
Train Error(Ordinary Error, not k-fold)= 0.16346153846153846
Test Error= 0.08695652173913043
=====
VV=====VV=====
learning_rate: 0.003
random state: 3
ratio of train set: 0.7
10-fold-cross-validation error is: 0.22142857142857145
Train Error(Ordinary Error, not k-fold)= 0.1346153846153846
Test Error= 0.043478260869565216
=====^
```

a بررسی عملکرد پرسپترون (one vs all) و آدالاین(adalines) در حالت خطی برای بررسی عملکرد پرسپترون و آدالاین در حالت خطی آنها را با پارامترهای یکسان مورد آزمایش قرار می‌دهیم. و نتایج آنها را مقایسه می‌کنیم.

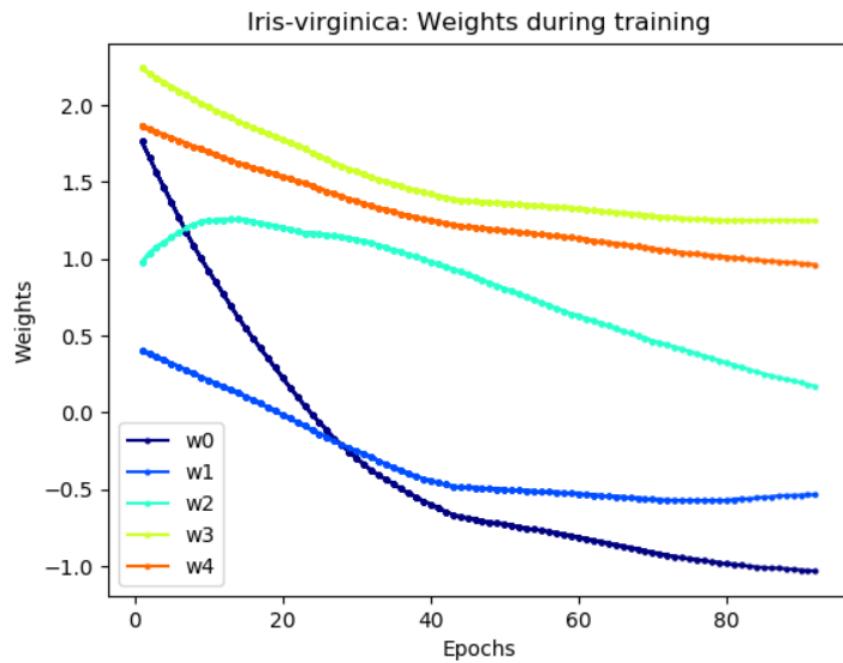
هر دو را با پارامترهای یکسان زیر که در فایل input-adalines-one-vs-all-3.json و input-perceptrons-one-vs-all-3.json مورد هستند مورد آزمایش قرار می‌دهیم و کدهای run\_adaline\_one\_vs\_all.py و run\_perceptrons\_one\_vs\_all.py را با آن دو فایل ورودی اجرا می‌کنیم:

```

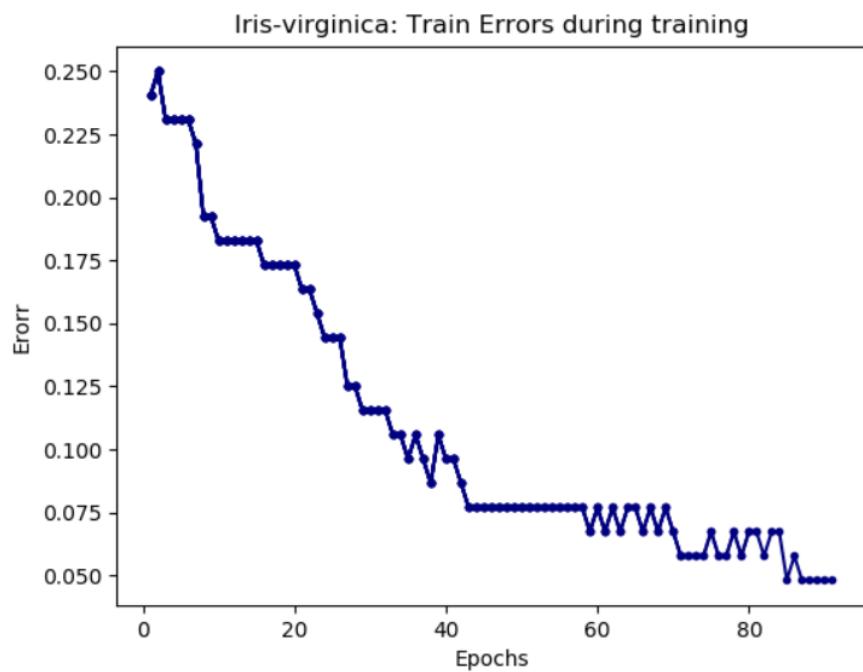
1  {
2      "ratio_of_train_set":0.70,
3      "ratio_of_valid_set":0.15,
4      "ratio_of_test_set":0.15,
5      "learning_rate":0.004,
6      "max_epochs":100,
7      "cut_error": -1,
8      "random_state":0,
9      "K_fold": 10,
10     "order":"1"
11 }
```

ضریب یادگیری برابر با  $4 \times 10^{-4}$  است، حداکثر تعداد اپوکها برابر با ۱۰۰ است، برای پایان آموزش فقط رسیدن به حداکثر تعداد ایپوکها چک می‌شود، برای K-Fold تعداد ۱۰ در نظر گرفته می‌شود، از فیچرها بدون اضافه کردن فیچرهای مرتبه‌ی ۲ استفاده می‌شود، ۷۰ درصد داده‌ها به آموزش ۱۵ درصد به ارزیابی و ۱۵ درصد به تست اختصاص پیدا می‌کند. از seed ۰ برای کار با عدهای تصادفی و تولید وزن‌ها استفاده شده است. در کدهای best adaline one vs all و best perceptron one vs all دو الگوریتم را با پارامترهای مختلفی فرا خوانده‌ام که آن کدها را در بالاتر توضیح داده‌ام. با نگاه کردن به نتایج پارامترهای مختلف و انتخاب یک حالت جنرال که استثنای نباشد این پارامترها را انتخاب کرده‌ام.

تغییر وزن‌ها در هر ایپاک برای پرسپترون کلاس Iris-virginica



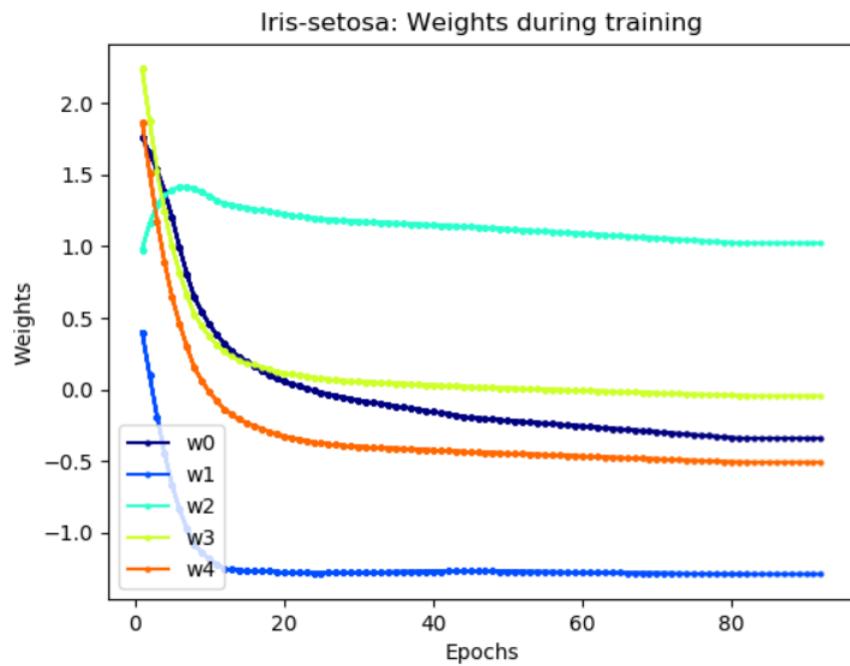
تغییر خطای مجموعه آموزش در هر ایپاک برای پرسپترون کلاس Iris-virginica



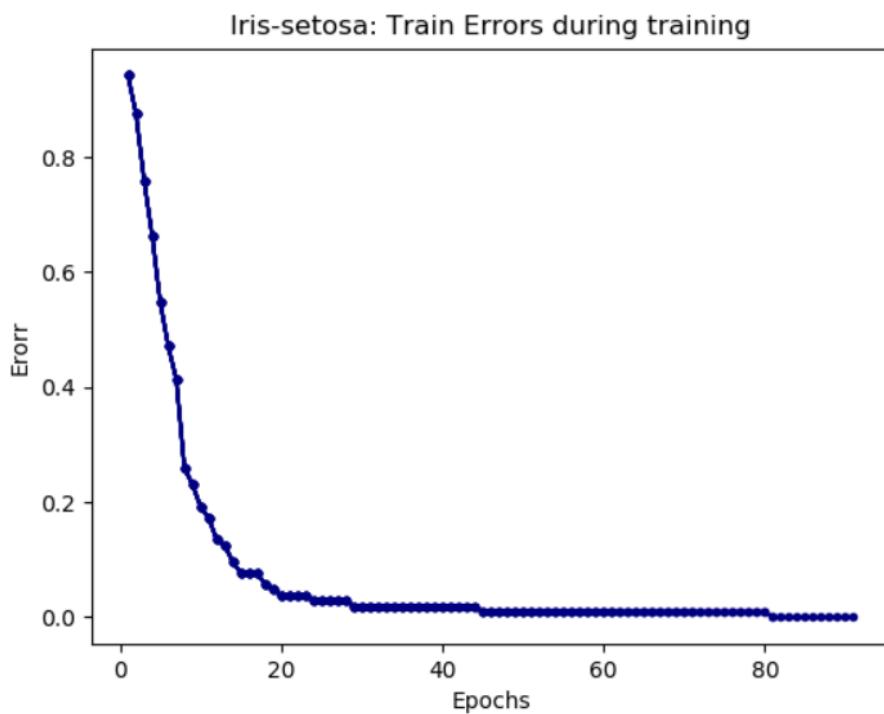
تغییر خطای مجموعه ارزیابی در هر ایپاک برای پرسپترون کلاس Iris-virginica



تغییر وزن‌ها در هر ایپاک برای پرسپترون کلاس Iris-setosa

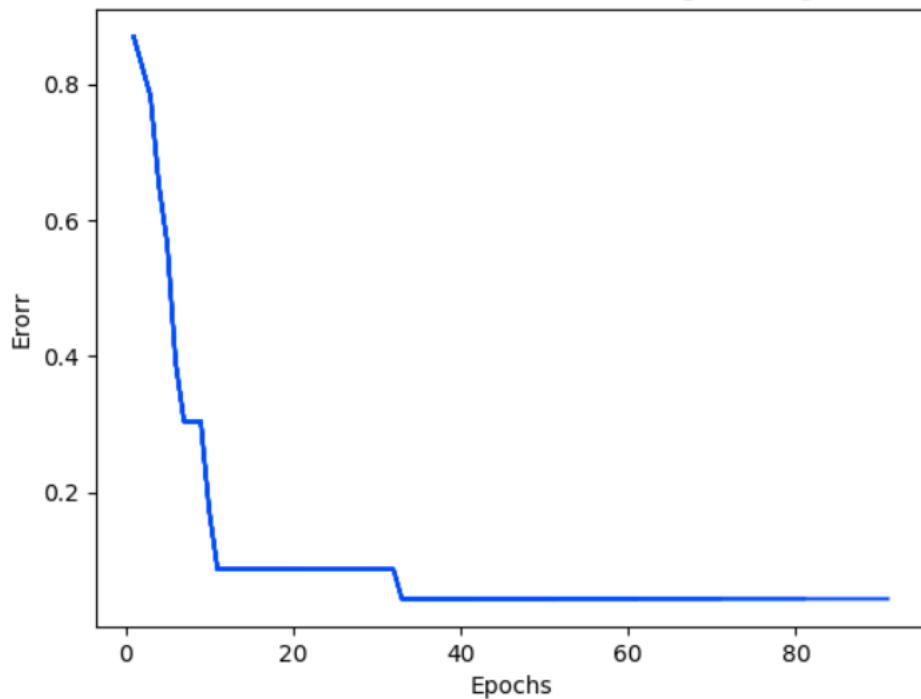


تغییر خطای مجموعه آموزش در هر ایپاک برای پرسپترون کلاس Iris-setosa



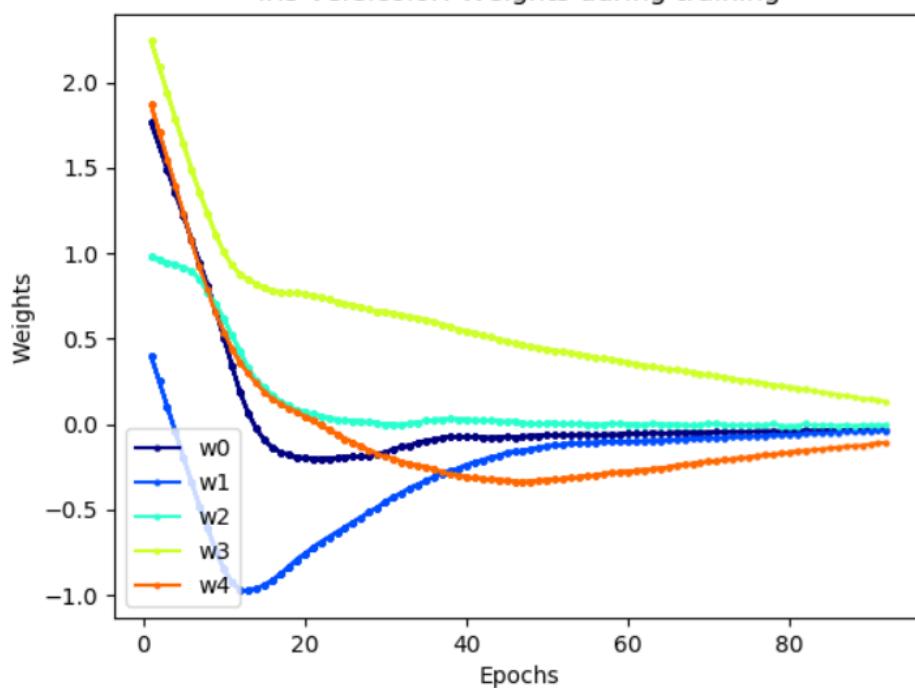
تغییر خطای مجموعه ارزیابی در هر ایپاک برای پرسپترون کلاس Iris-setosa

Iris-setosa: Validation Errors during training

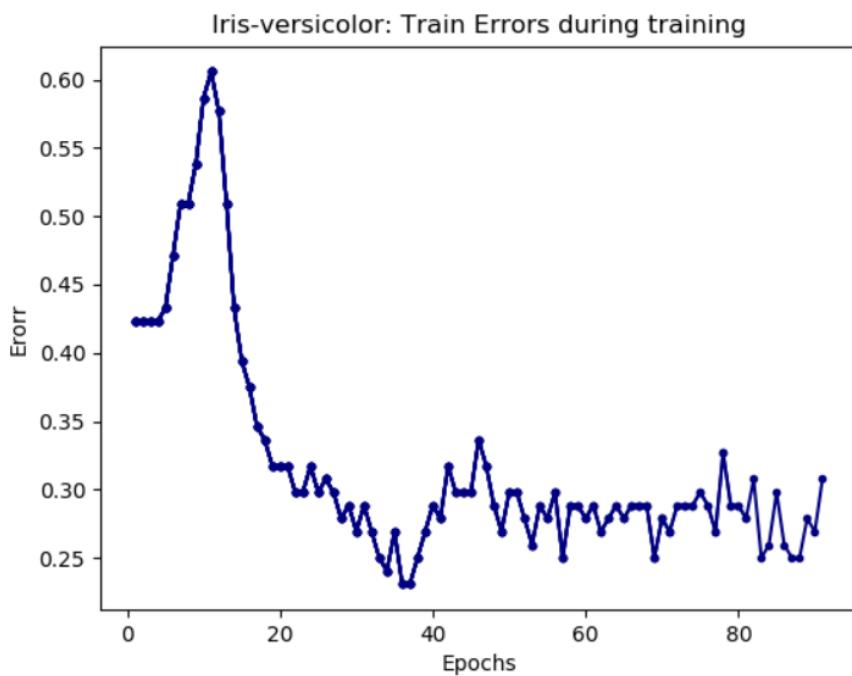


تغییر وزن‌ها در هر ایپاک برای پرسپترون کلاس Iris-versicolor

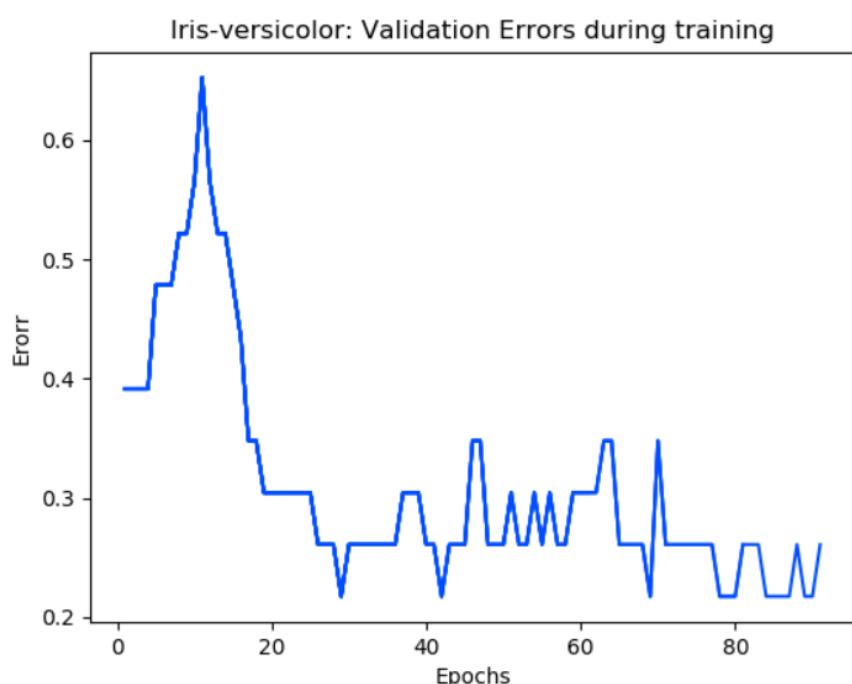
Iris-versicolor: Weights during training



تغییر خطای مجموعه آموزش در هر ایپاک برای پرسپترون کلاس Iris-versicolor



تغییر خطای مجموعه ارزیابی در هر ایپاک برای پرسپترون کلاس Iris-versicolor



خطای k-fold و خطای عادی آموزش و خطای تست در پایان:

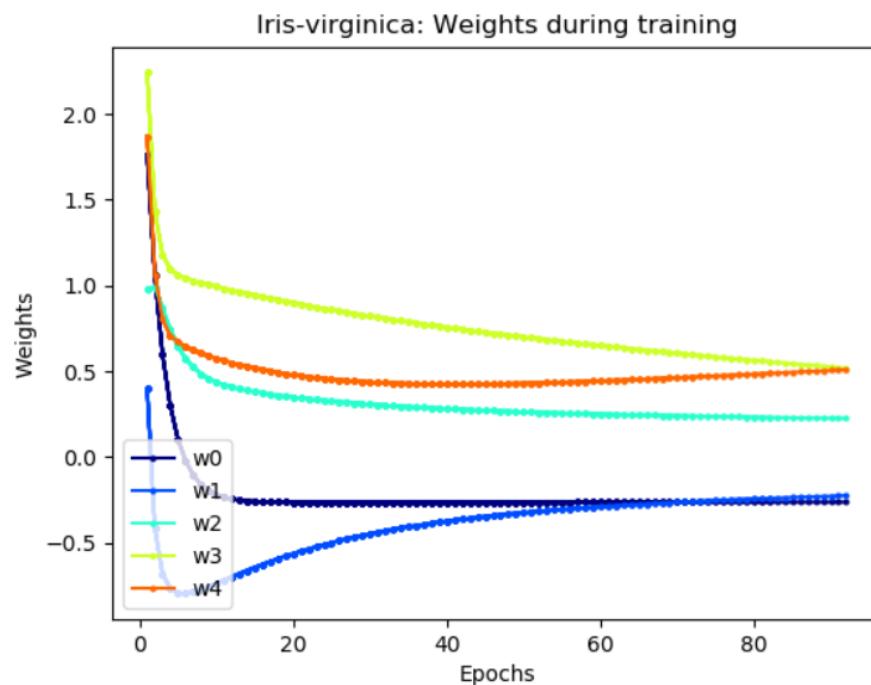
```

Calculating 10-Fold Cross Validation ...
10-fold-cross-validation error is: 0.06999999999999999
Train Error(Ordinary Error, not k-fold)= 0.04807692307692308
Test Error= 0.0

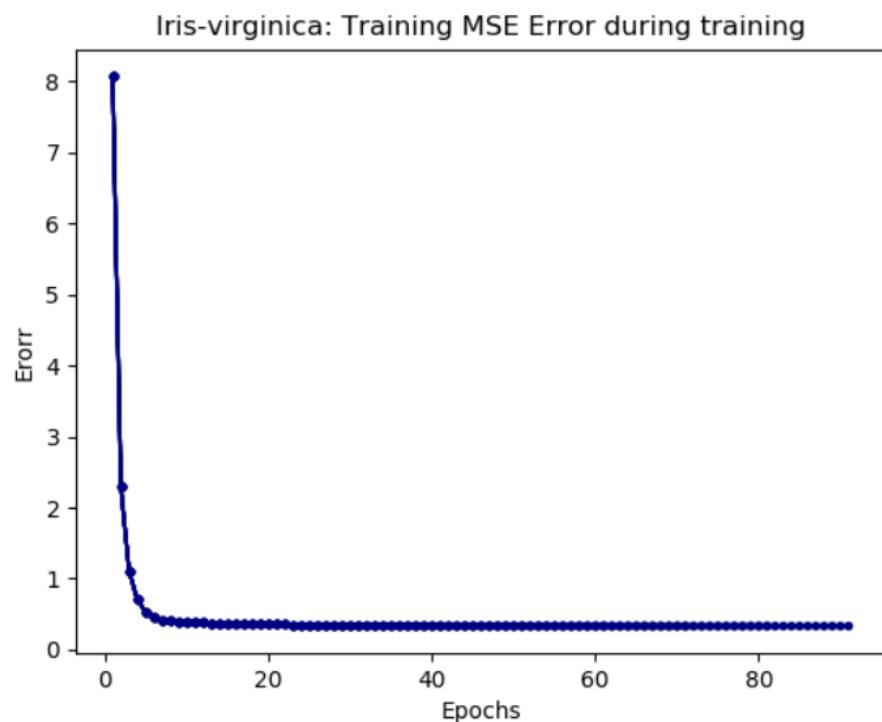
```

همین طور که از خطای آموزش و ارزیابی دیده می‌شود کلاس Iris setosa جدا پذیر خطی است زیرا خطای آموزش برای آن صفر می‌شود ولی دو کلاس دیگر جداپذیر خطی نیستند. در هر سه کلاس تغییرات وزن‌ها با زیاد شدن ایپاک‌ها کمتر می‌شود. در هر سه کلاس به طور کلی میانگین با گذر زمان خطای آموزش و ارزیابی کاهش یافته است ولی در بعضی ایپاک‌ها نسبت به ایپاک قبلی نوساناتی دیده شود دلیل آن این است که پرسپترون مانند k-fold coincidence optimization است و الزاماً بر کم شدن خطا در هر ایپاک وجود ندارد. خطای best\_perceptron\_one\_vs\_all.py که در بالا شرح دادم، این کار را با پارامترهای مختلف انجام دادم که نتایجش در find\_good\_perceptrons-order-1.txt موجود است. بیشتر نتایجی که به دست آوردم خطایی بین ۰ تا ۱۰ درصد داشتند که می‌توان نتیجه گرفت perceptron one vs all به دقتی بین ۹۰ تا ۱۰۰ درصد برای این دیتا است می‌رسد.

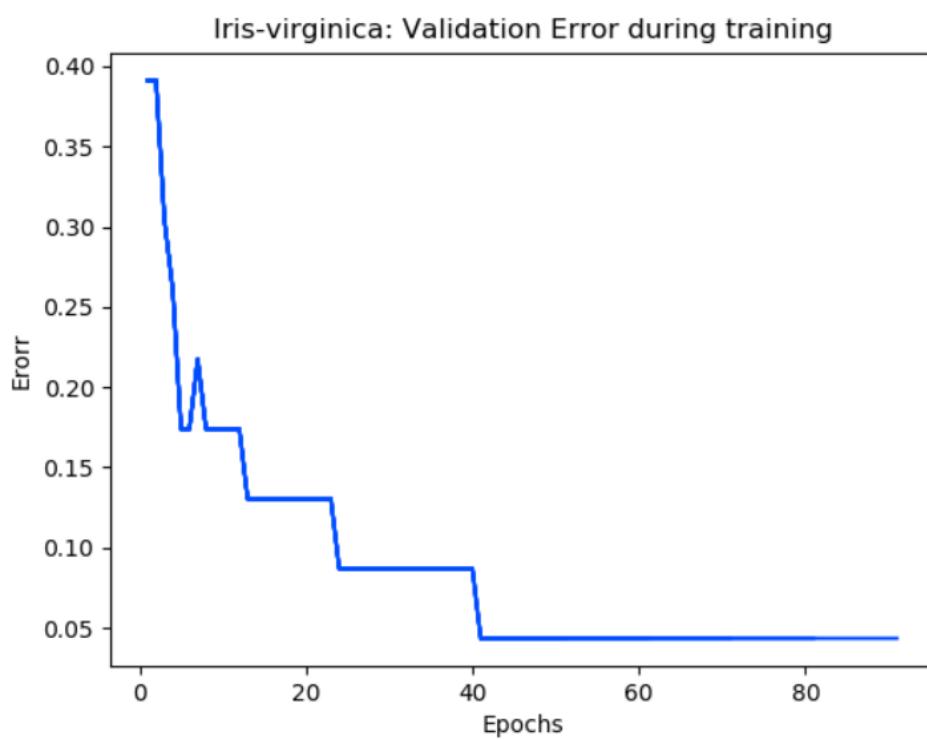
تغییر وزن‌ها در هر ایپاک برای آدالاین کلاس Iris-virginica



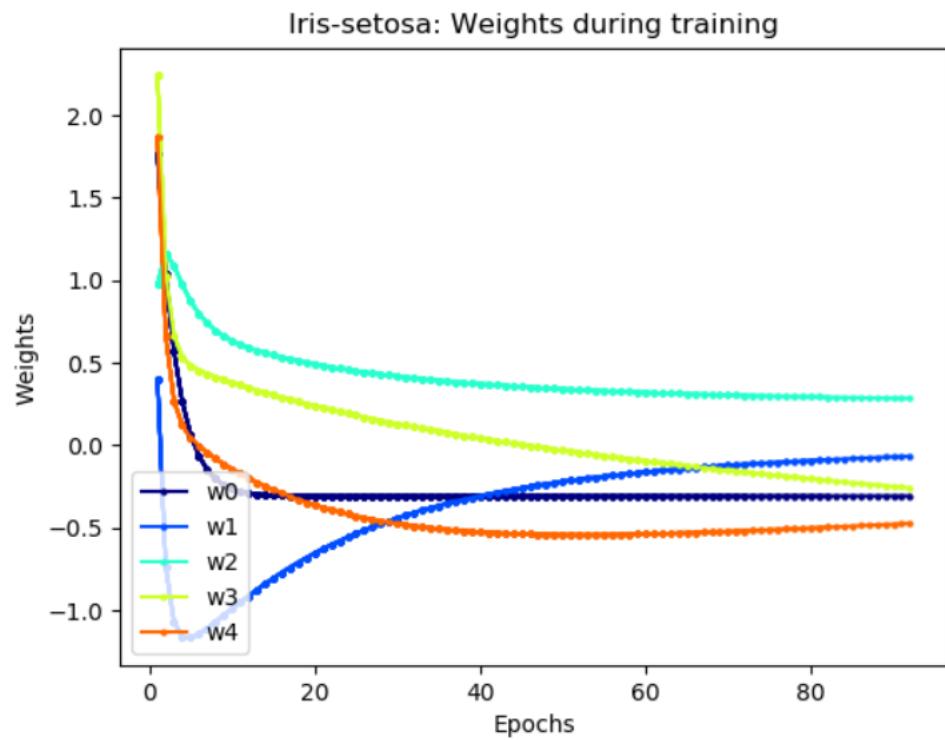
تغییر خطای مجموعه آموزش در هر اپاک برای آدالین کلاس Iris-virginica



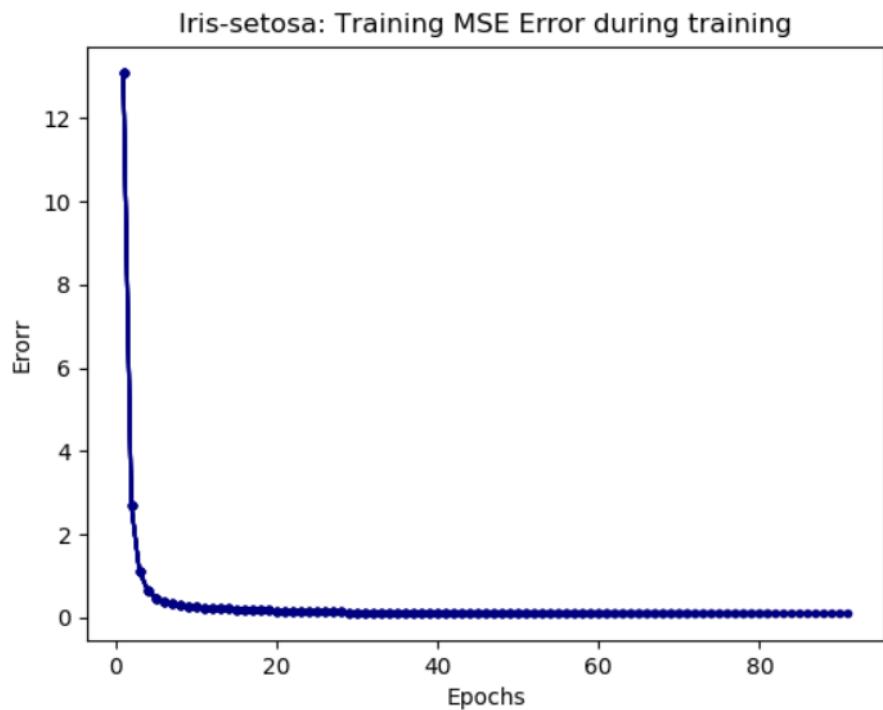
تغییر خطای مجموعه ارزیابی برای آدالین کلاس Iris-virginica



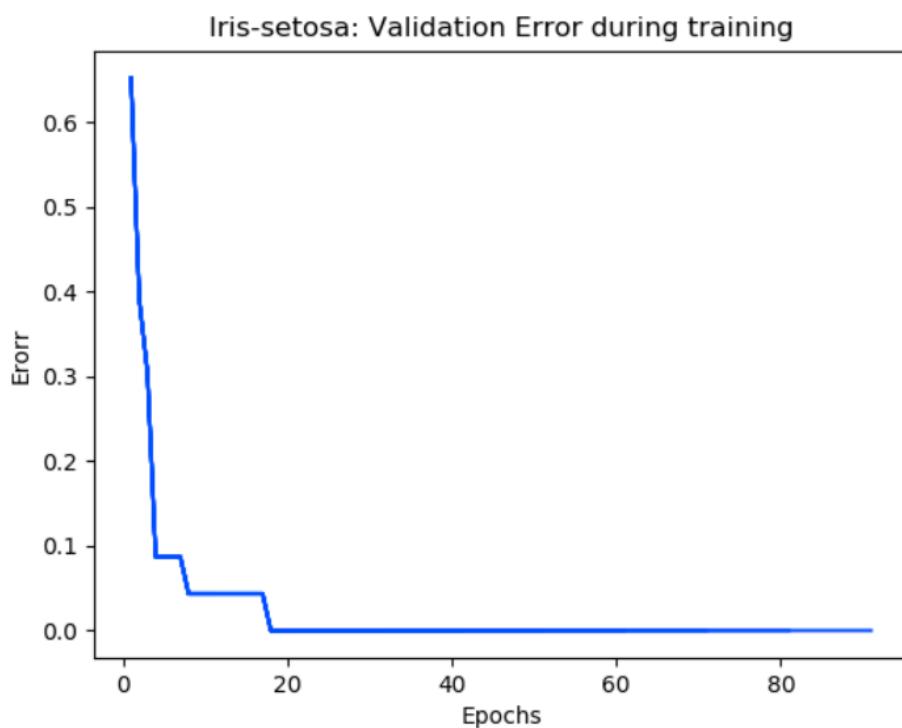
تغییر وزن‌ها در هر ایپاتک برای آدالین کلاس Iris-setosa



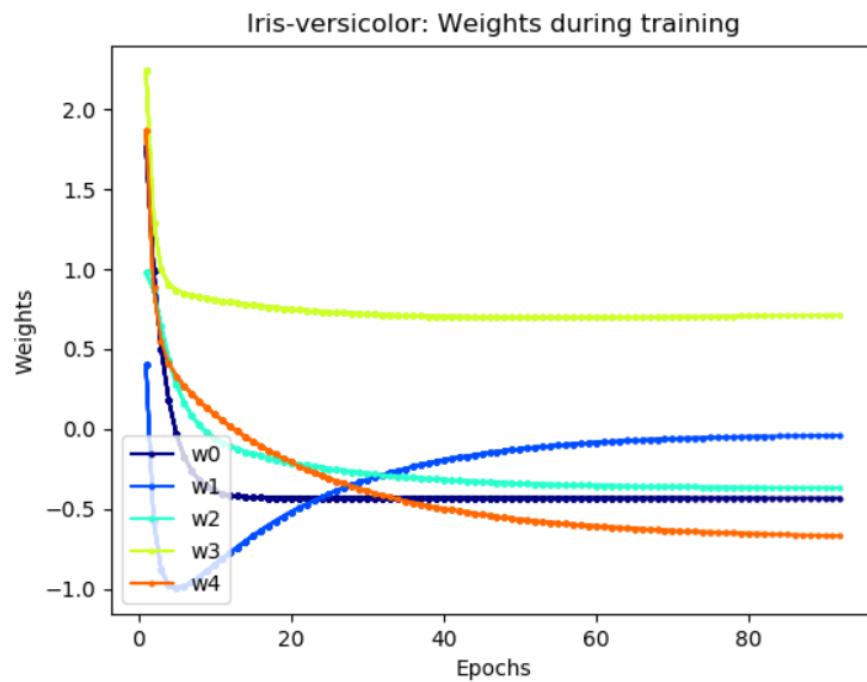
تغییر خطای مجموعه آموزش در هر ایپاک برای آدالین کلاس Iris-setosa



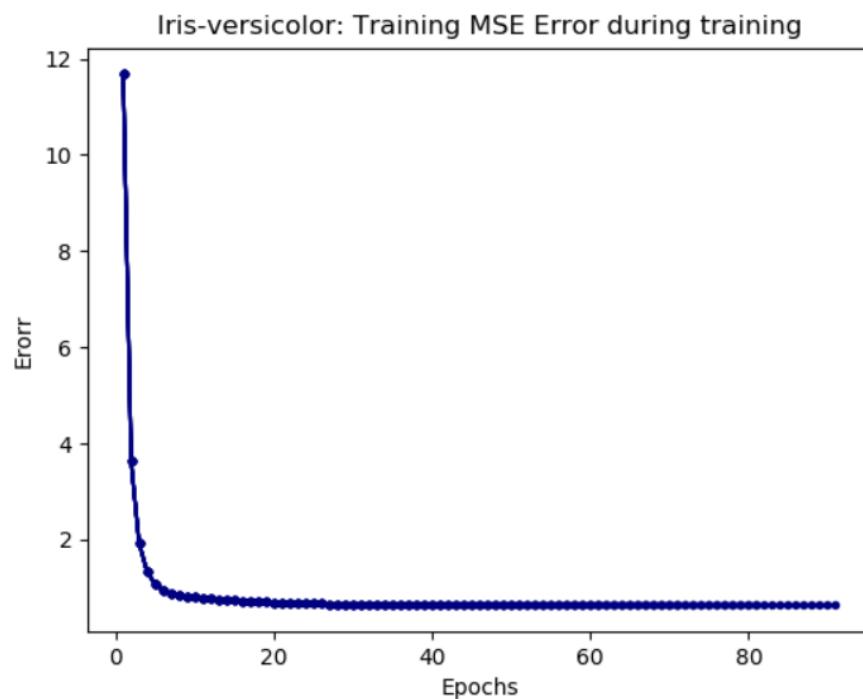
تغییر خطای مجموعه ارزیابی در هر ایپاک برای آدالین کلاس Iris-setosa



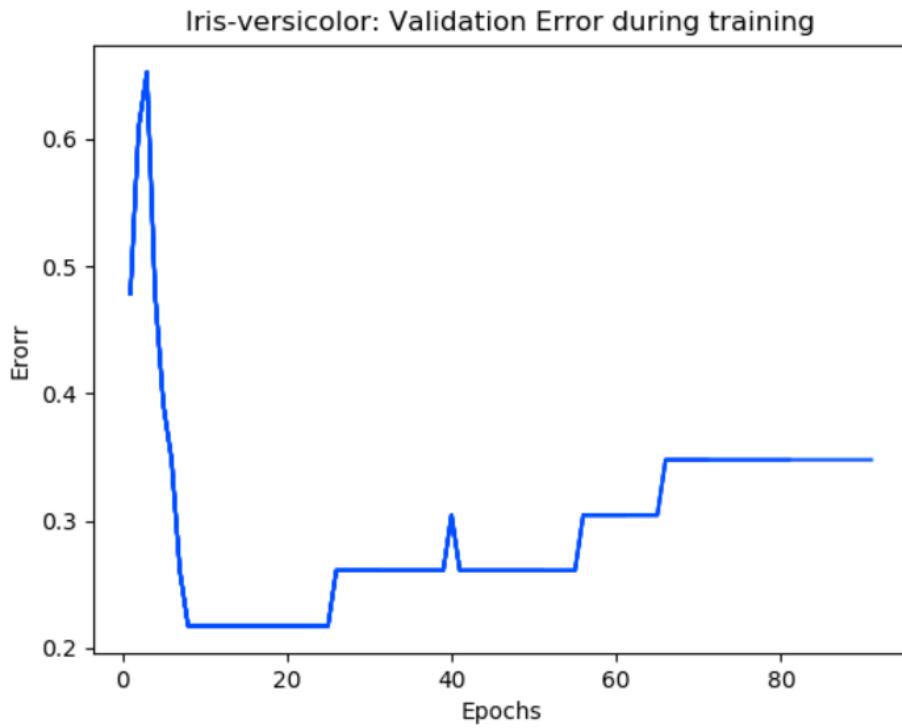
تغییر وزن‌ها در هر ایپاک برای آدالین کلاس Iris-versicolor



تغییر خطای مجموعه آموزش در هر ایپاک برای آدالین کلاس Iris-versicolor



تغییر خطای مجموعه ارزیابی در هر ایپاک برای آدالین کلاس Iris-versicolor



خطای k-fold و خطای عادی آموزش و خطای تست در پایان:

```
Calculating 10-Fold Cross Validation ...
10-fold-cross-validation error is: 0.19714285714285715
Train Error(Ordinary Error, not k-fold)= 0.16346153846153846
Test Error= 0.30434782608695654
```

در آدالاین نمودار خطای آموزش در حین آموزش نوسان ندارد و حالت نزولی دارد. وزن‌ها در ایپاک‌های بالاتر تغییرات کمتری دارند. در کد best\_adaline\_one\_vs\_all.py این کار را با پارامترهای مختلف انجام داده‌ام که عموماً به خطای بین ۱۳ تا ۳۰ درصد رسیده‌ام، یعنی آدالاین برای این دیتاست معمولاً دقیقی بین ۷۰ تا ۸۷ درصد دارد.

در پرسپترون خطای آموزش نوسان دارند و گاهی زیاد می‌شود در حالی که آدالاین اینگونه نیست و خطای مجموعه آموزش نزولی است. در قسمت‌های قبل (پیاده سازی پرسپترون و آدالاین) نحوه‌ی آموزش و آپدیت وزن‌های هر دو را شرح دادم. پرسپترون بر اساس coincidence optimization است ولی آدالاین اینگونه

نیست و از گرادیان نزولی و مشتق‌ها برای کمتر کردن خطای گام استفاده می‌کنند. در پرسپترون خطای ارزیابی معمولاً دچار نوسان است در حالی که این موضوع در آدالین بسیار کم دیده می‌شود. خطای ارزیابی مرتب کاهش می‌یابد تا زمانی که دچار over fit می‌شویم و با زید شدن ایپاک‌ها خطای ارزیابی زیاد می‌شود ولی نوسان به ندرت دیده می‌شود.

در اجراهای مختلف برای این دیتاست به پارامترهای یکسان عموماً پرسپترون بهتر از آدالین عمل می‌کند. **عموماً** دقیق پرسپترون چیزی بین ۹۰ تا ۱۰۰ درصد است ولی آدالین دقیق بین ۷۰ تا ۸۷ نشان می‌دهد.

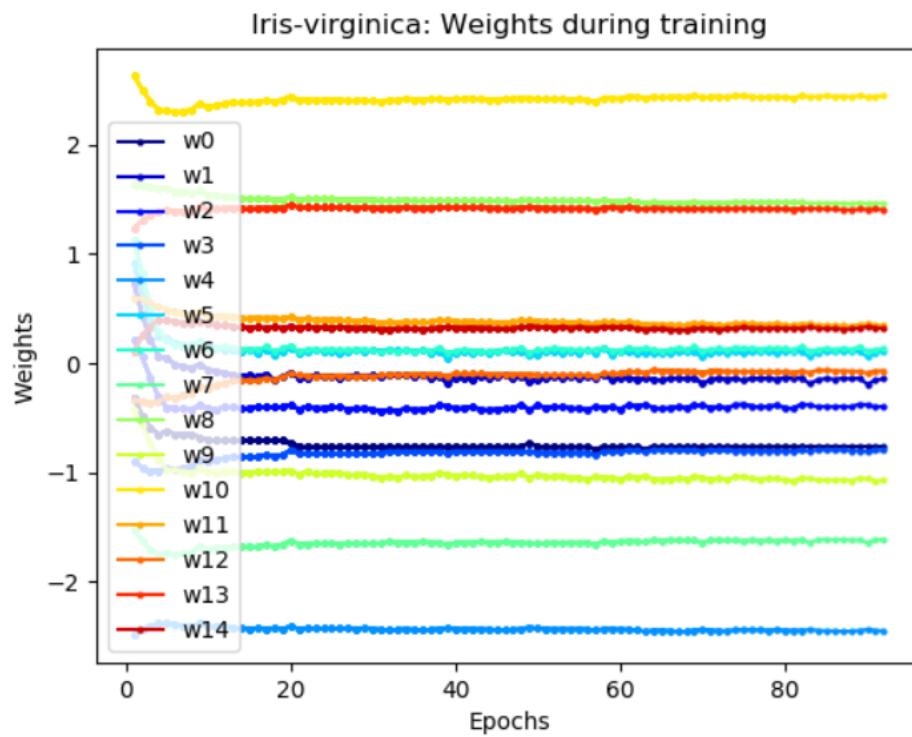
B جاپذیری کلاس‌ها در پرسپترون خطی و بررسی جاپذیری آن‌ها در پرسپترون درجه دو در اجرای پرسپترون در قسمت قبل (a) و قسمت "ایجاد شی و فراخوانی کلاس perceptrons one vs all" (run\_perceptrons\_one\_vs\_all.py) با دیدن نمودار خطای مجموعه‌ی آموزش و ارزیابی در زمان آموزش پرسپترون‌های کلاس‌های مختلف مشاهده کردیم فقط کلاس Iris\_setosa به خطای صفر می‌رسد و سایر کلاس‌ها به خطای صفر نرسیدند. به همین دلیل کلاس‌ها با پرسپترون خطی جاپذیر نیستند.

برای چک کردن این موضوع کد run\_perceptrons\_one\_vs\_all.py را با پارامترهای موجود در فایل input-perceptrons-one-vs-all-4.json را فرا می‌خوانیم تا جاپذیر بودن کلاس‌ها در پرسپترون درجه دو را چک کنیم:

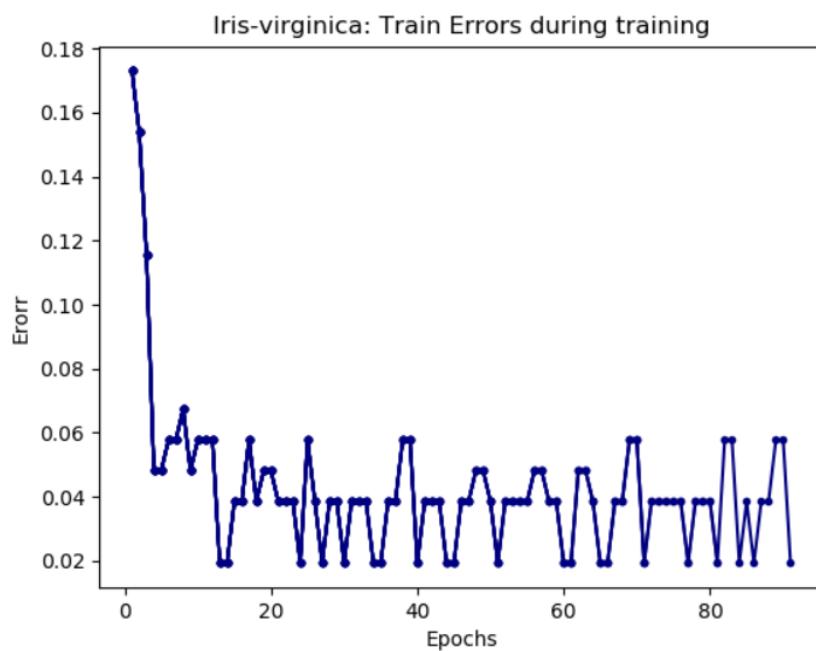
```
input-perceptrons-one-vs-all-4.json
1  {
2      "ratio_of_train_set":0.70,
3      "ratio_of_valid_set":0.15,
4      "ratio_of_test_set":0.15,
5      "learning_rate":0.028,
6      "max_epochs":100,
7      "cut_error": -1,
8      "random_state":6,
9      "K_fold": 10,
10     "order":"2"
11 }
```

ضریب یادگیری برابر با  $0.028$  است، حداکثر تعداد اپوکها برابر با  $100$  است، برای پایان آموزش فقط رسیدن به حداکثر تعداد ایپوکها چک می‌شود، برای K-Fold تعداد  $10$  در نظر گرفته می‌شود، از فیچرها با اضافه کردن فیچرهای مرتبه  $2$  استفاده می‌شود،  $70$  درصد داده‌ها به آموزش  $15$  درصد به ارزیابی و  $15$  درصد به تست اختصاص پیدا می‌کند. از  $6$  برای کار با عده‌های تصادفی و تولید وزن‌ها استفاده شده است. در کدهای `best adaline one vs all` و `best perceptron one vs all` دو الگوریتم را با پارامترهای مختلفی فرا خوانده‌ام که آن کدها را در بالاتر توضیح داده‌ام. با نگاه کردن به نتایج پارامترهای مختلف و انتخاب یک حالت جنرال که استثناء نباشد این پارامترها را انتخاب کرده‌ام.

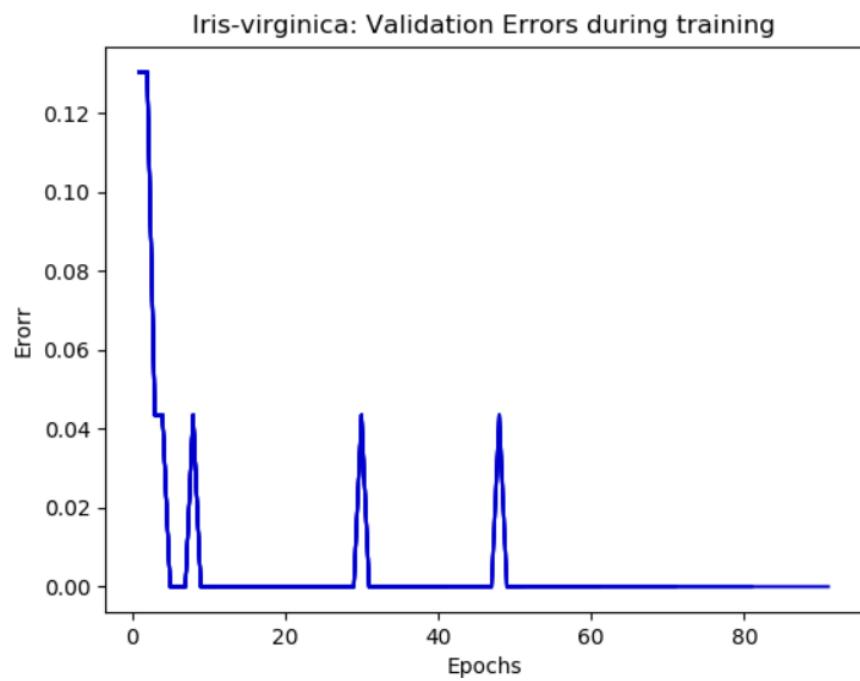
تغییر وزن‌ها در هر ایپاتک برای پرسپترون کلاس `Iris-virginica`



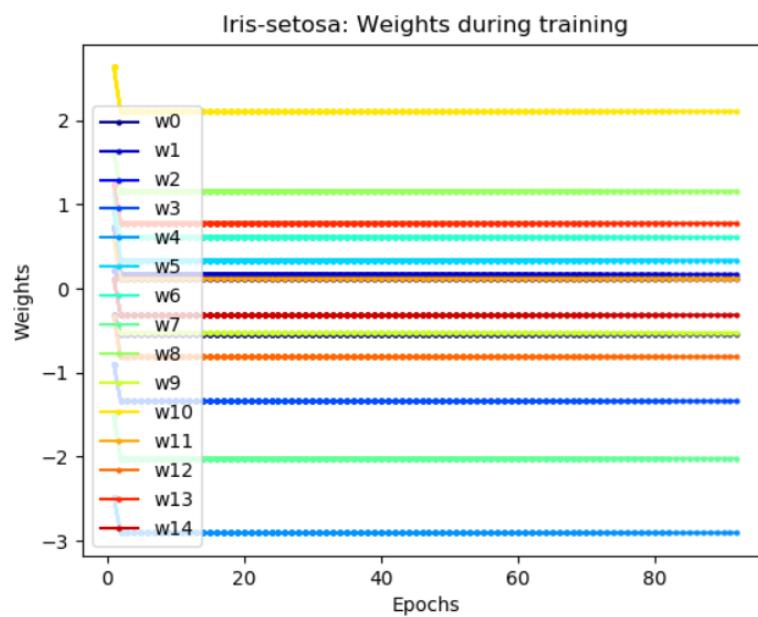
تغییر خطای مجموعه آموزش در هر ایپاک برای پرسپترون کلاس Iris-virginica



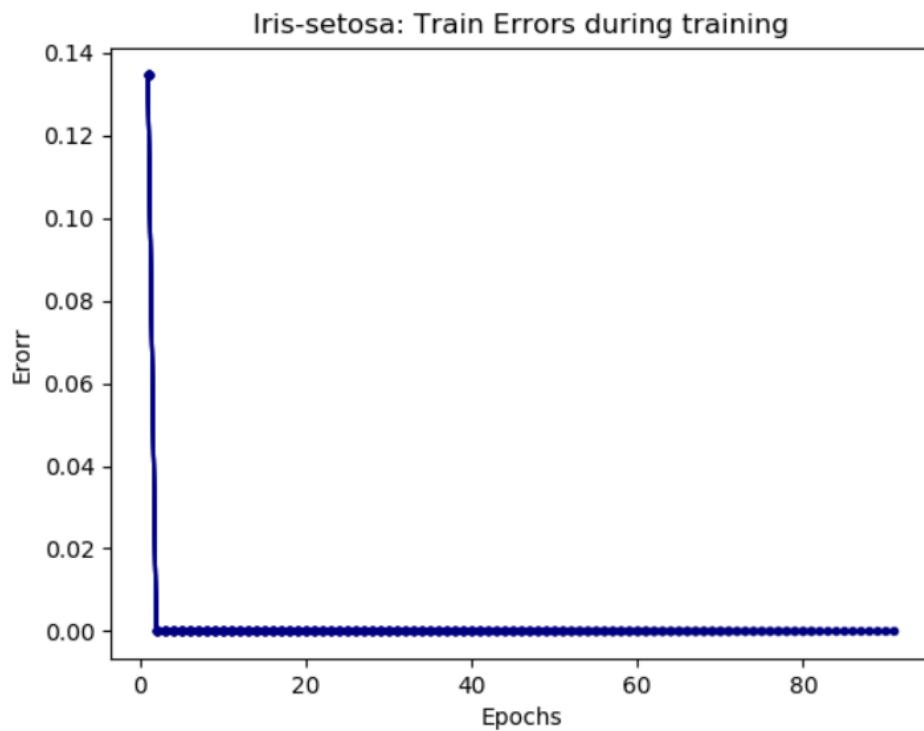
تغییر خطای مجموعه ارزیابی در هر ایپاک برای پرسپترون کلاس Iris-virginica



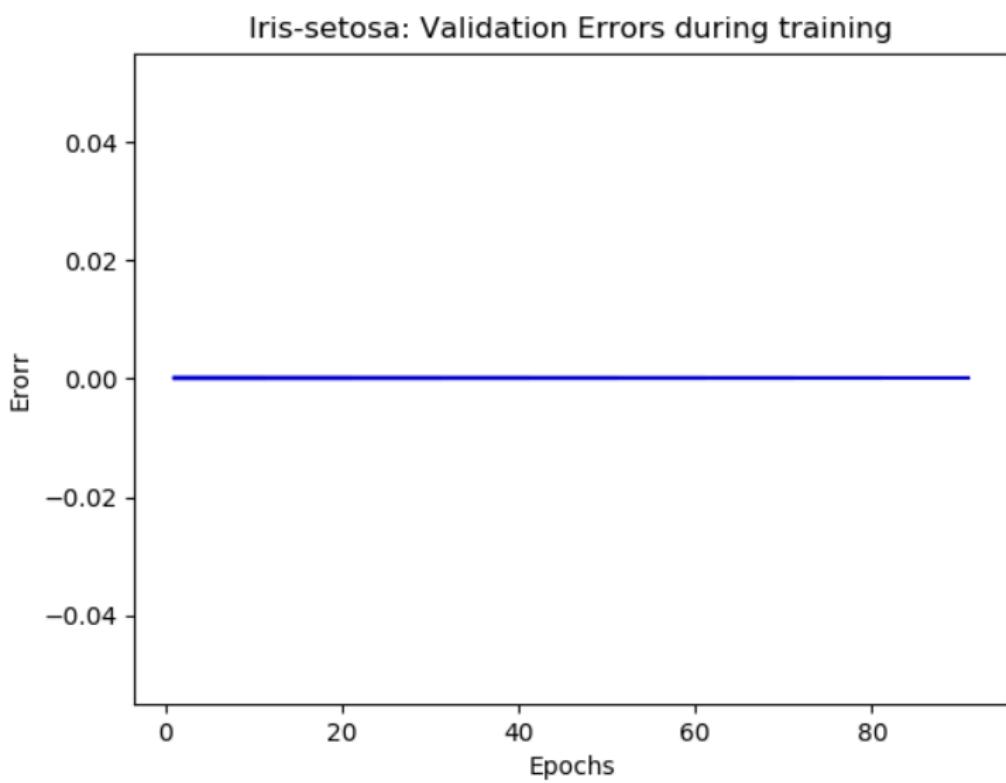
تغییر وزن‌ها در هر ایپاک برای پرسپترون کلاس Iris-setosa



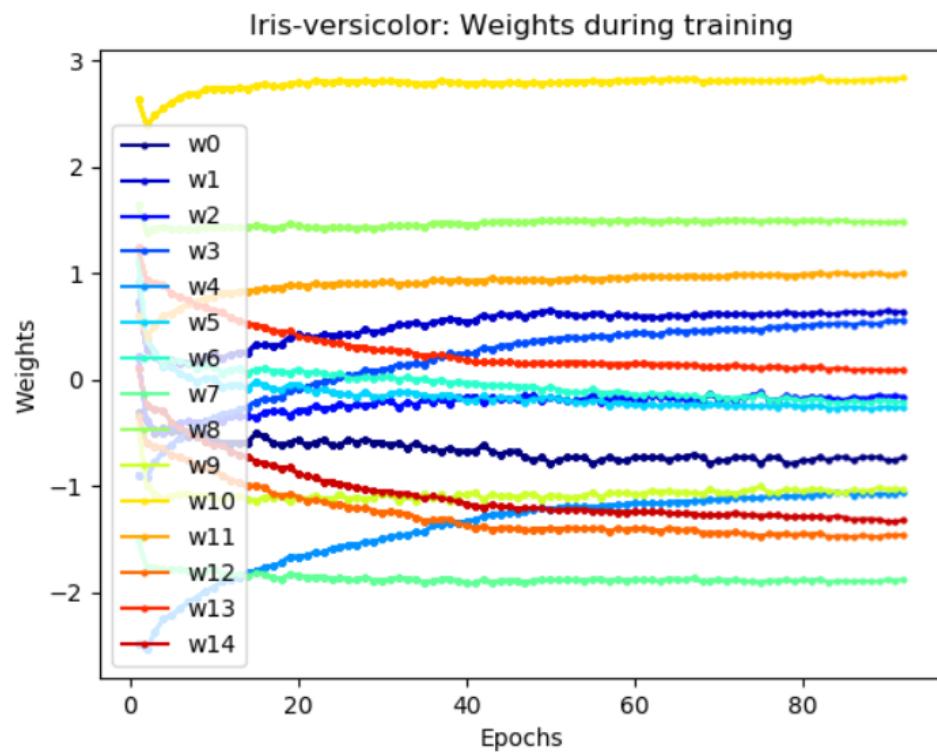
تغییر خطای مجموعه آموزش در هر ایپاک برای پرسپترون کلاس Iris-setosa



تغییر خطای مجموعه ارزیابی در هر ایپاک برای پرسپترون کلاس Iris-setosa

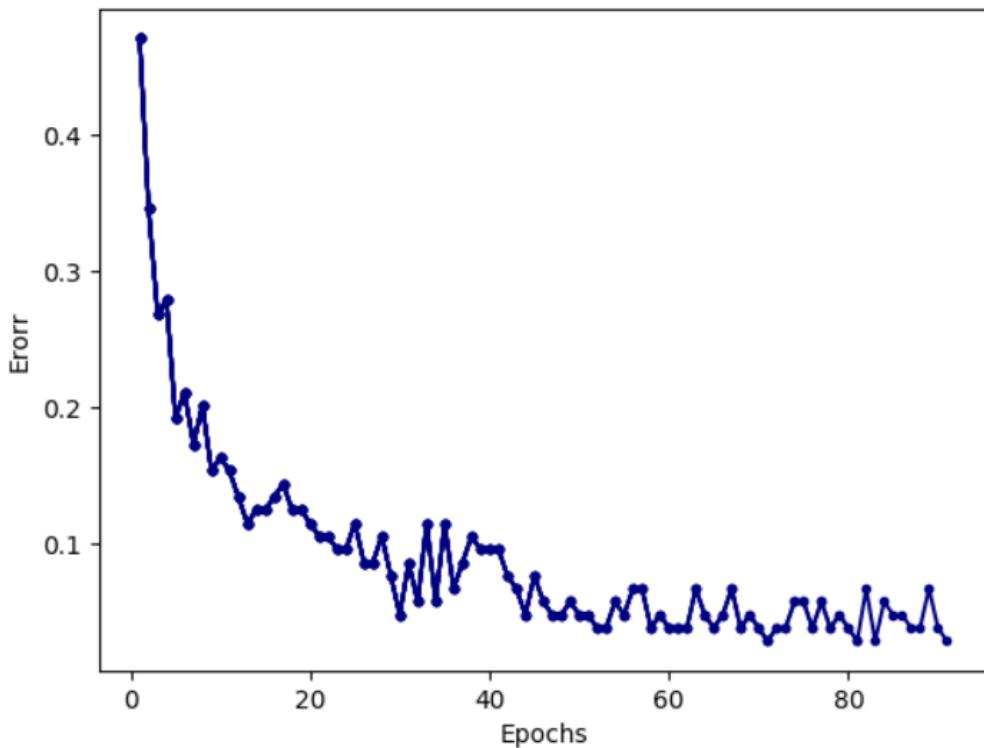


تغییر وزن ها در هر ایپاک برای پرسپترون کلاس Iris-versicolor



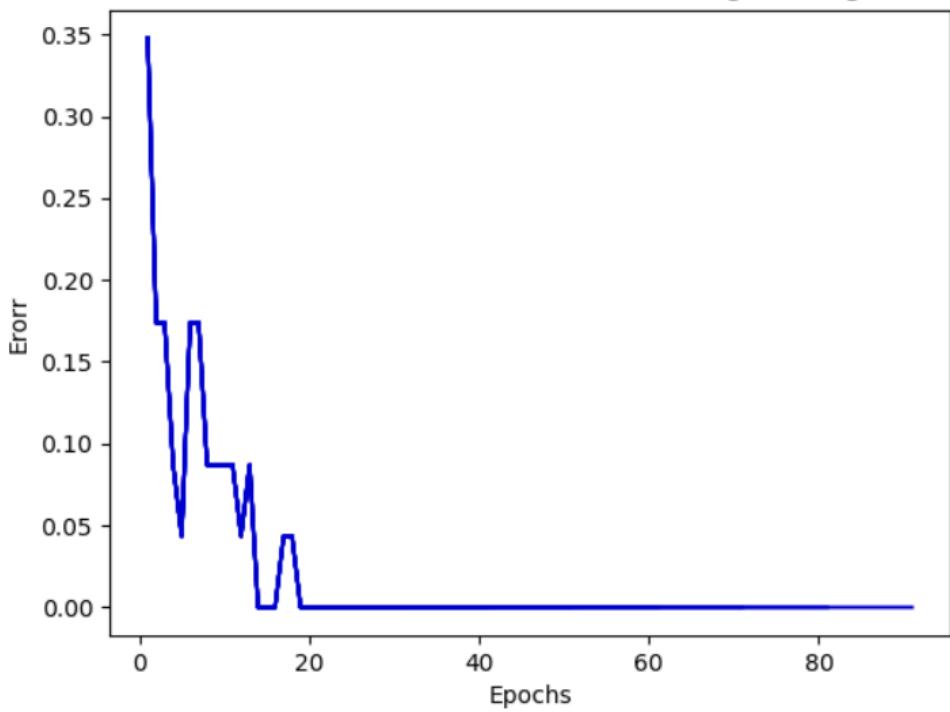
تغییر خطای مجموعه آموزش در هر ایپاک برای پرسپترون کلاس Iris-versicolor

Iris-versicolor: Train Errors during training



تغییر خطای مجموعه ارزیابی در هر ایپاک برای پرسپترون کلاس Iris-versicolor

Iris-versicolor: Validation Errors during training



خطای k-fold و خطای عادی آموزش و خطای تست در پایان:

```
Calculating 10-Fold Cross Validation ...
10-fold-cross-validation error is: 0.04
Train Error(Ordinary Error, not k-fold)= 0.028846153846153848
Test Error= 0.0
```

همین طور که در نمودارهای خطای مجموعه‌ی آموزش برای ۳ کلاس دیده می‌شود فقط کلاس Iris-setosa با پرسپترون درجه‌ی دو جدا پذیر خطی بوده است و دو کلاس دیگر به خطای صفر در مجموعه‌ی آموزش خود نرسیده‌اند در نتیجه با پرسپترون درجه‌ی دو جدا پذیر خطی نیستند.

با استفاده از کد best perceptron on vs all با پارامترهای مختلف پرسپترون درجه‌ی دو را آزمودم ولی نتایج مانند مثال بالا بود و آن دو کلاس جدا پذیر نشدند.

### C بررسی عملکرد الگوریتم‌ها به ازای نرخ‌های یادگیری مختلف

برای این کار هر دو الگرویتم را با پارامترهای کاملاً یکسان به طوری که فقط ضریب یادگیری متفاوتی داشته باشند فرا می‌خوانیم.

برای پرسپترون و آadalines one vs all و run perceptron one vs all run adalines one vs all کد input-perceptrons-one-vs-all-5-2.json و input-perceptrons-one-vs-all-5-1.json ورودی فرا می‌خوانیم.

### input-perceptrons-one-vs-all-5-1.json

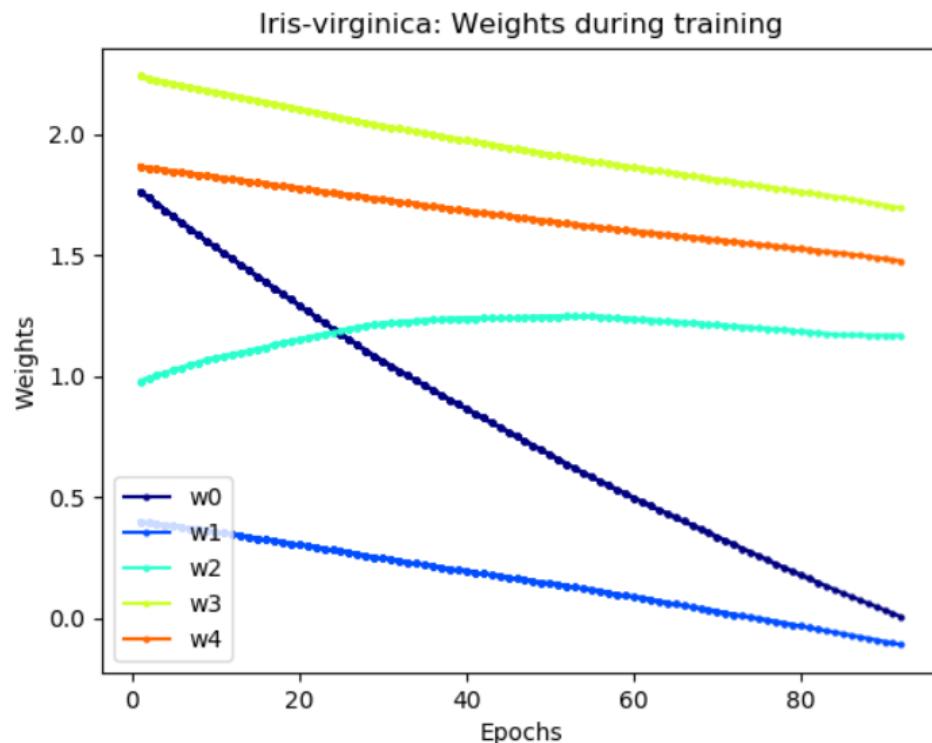
```
1  {  
2      "ratio_of_train_set":0.70,  
3      "ratio_of_valid_set":0.15,  
4      "ratio_of_test_set":0.15,  
5      "learning_rate":0.001,  
6      "max_epochs":100,  
7      "cut_error": -1,  
8      "random_state":0,  
9      "K_fold": 10,  
10     "order":"1"  
11 }
```

```
1  {  
2      "ratio_of_train_set":0.70,  
3      "ratio_of_valid_set":0.15,  
4      "ratio_of_test_set":0.15,  
5      "learning_rate":0.025,  
6      "max_epochs":100,  
7      "cut_error": -1,  
8      "random_state":0,  
9      "K_fold": 10,  
10     "order":"1"  
11 }
```

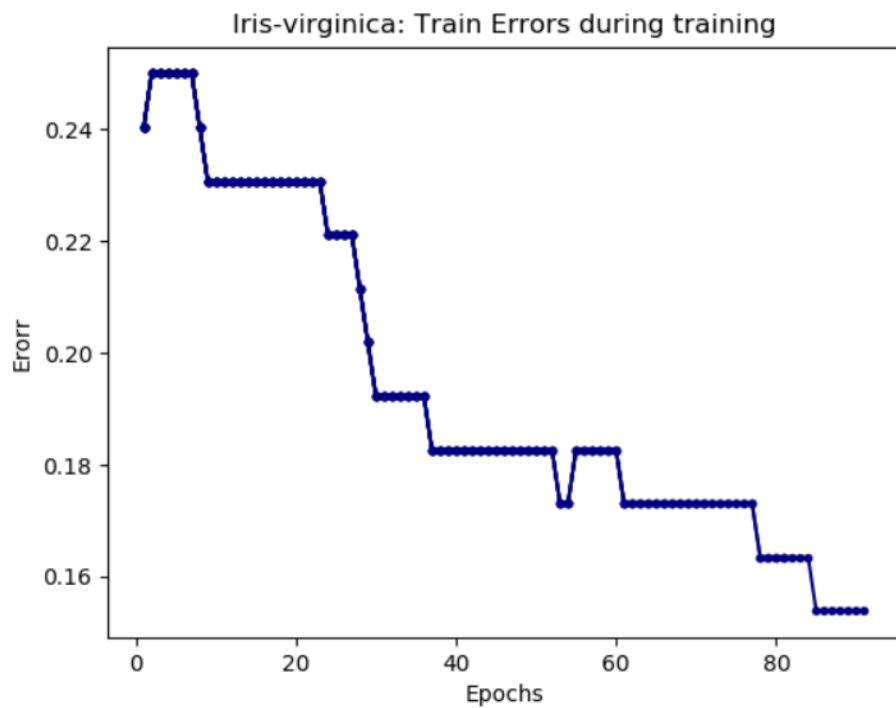
ضریب یادگیری برابر با ۰.۰۲۵ و برای دیگری برابر با ۰.۰۰۱ است، برای پایان آموزش فقط رسیدن به حداقل تعداد ایپوکها چک می‌شود، برای K-Fold تعداد ۱۰ fold در نظر گرفته می‌شود، از فیچرها بدون اضافه کردن فیچرهای مرتبه‌ی ۲ استفاده می‌شود، ۷۰ درصد داده‌ها به آموزش ۱۵ درصد به ارزیابی و ۱۵ درصد به تست اختصاص پیدا می‌کند. از seed ۰ برای کار با عددهای تصادفی و تولید وزن‌ها استفاده شده است. در کدهای best adaline one vs all و best perceptron one vs all الگوریتم را با پارامترهای مختلفی فرا خوانده‌ام که آن کدها را در بالاتر توضیح داده‌ام. با نگاه کردن به نتایج پارامترهای مختلف و انتخاب یک حالت جنرال که استثنای نباشد این پارامترها را انتخاب کرده‌ام.

خروجی‌های فایل ورودی اول پرسپترون

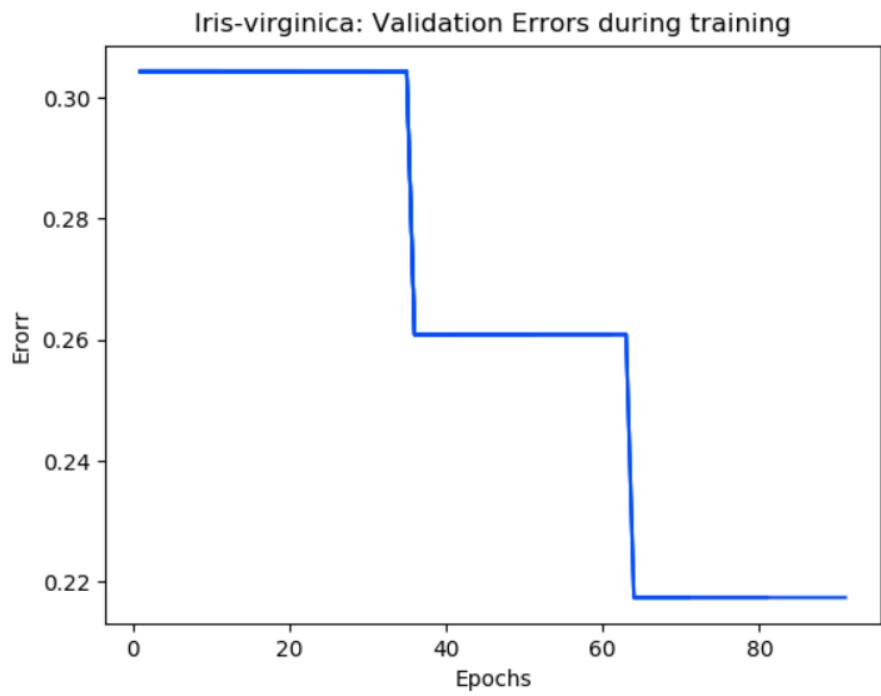
تغییر وزن‌ها در هر ایپاک برای پرسپترون کلاس Iris-virginica



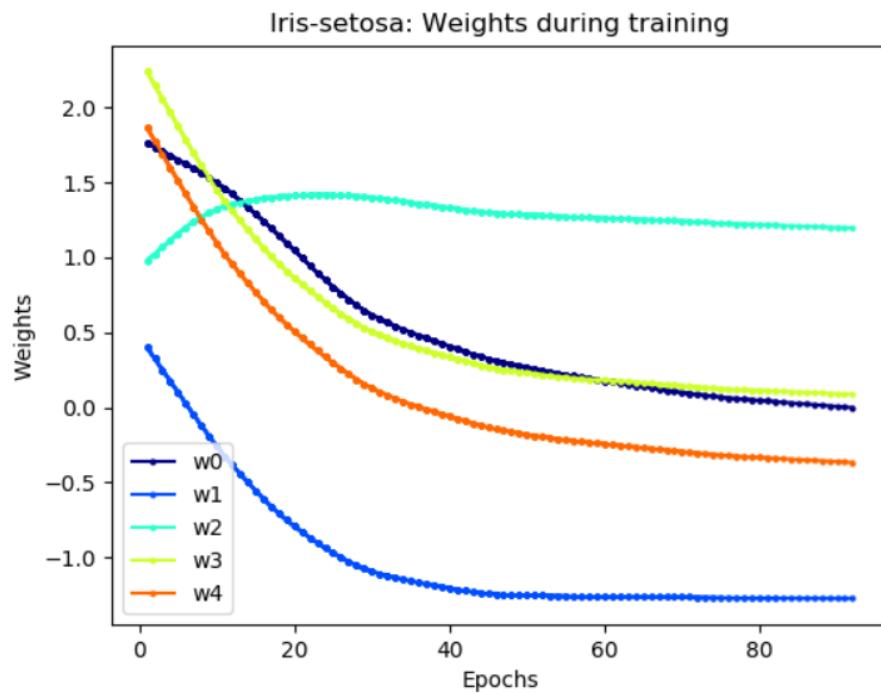
تغییر خطای مجموعه آموزش در هر ایپاک برای پرسپترون کلاس Iris-virginica



تغییر خطای مجموعه ارزیابی در هر ایپاک برای پرسپترون کلاس Iris-virginica

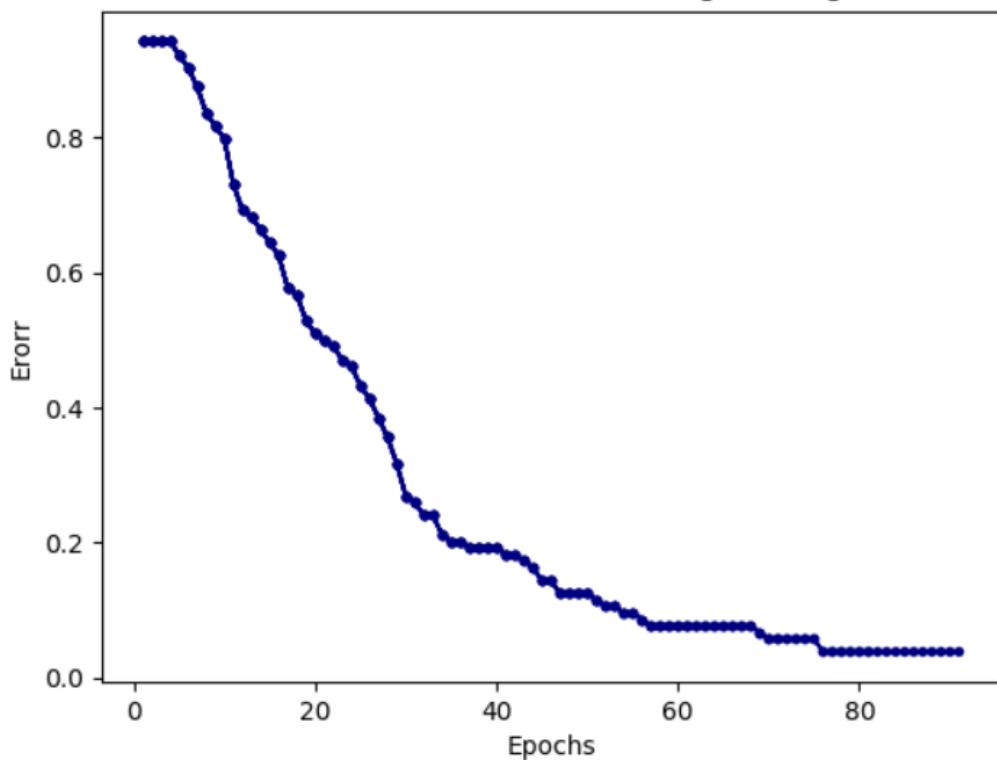


تغییر وزن ها در هر ایپاک برای پرسپترون کلاس Iris-setosa



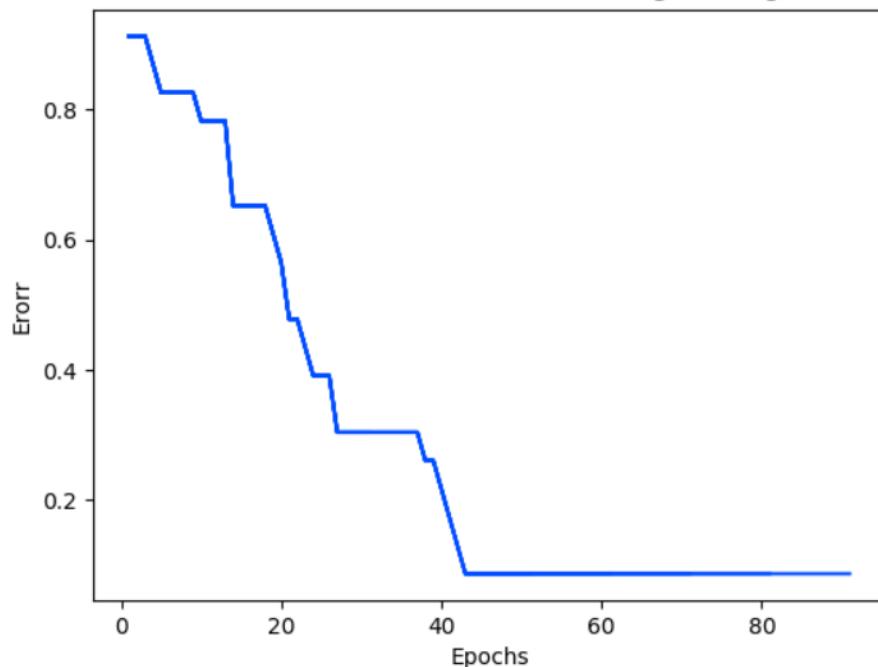
تغییر خطای مجموعه آموزش در هر ایپاک برای پرسپترون کلاس Iris-setosa

Iris-setosa: Train Errors during training

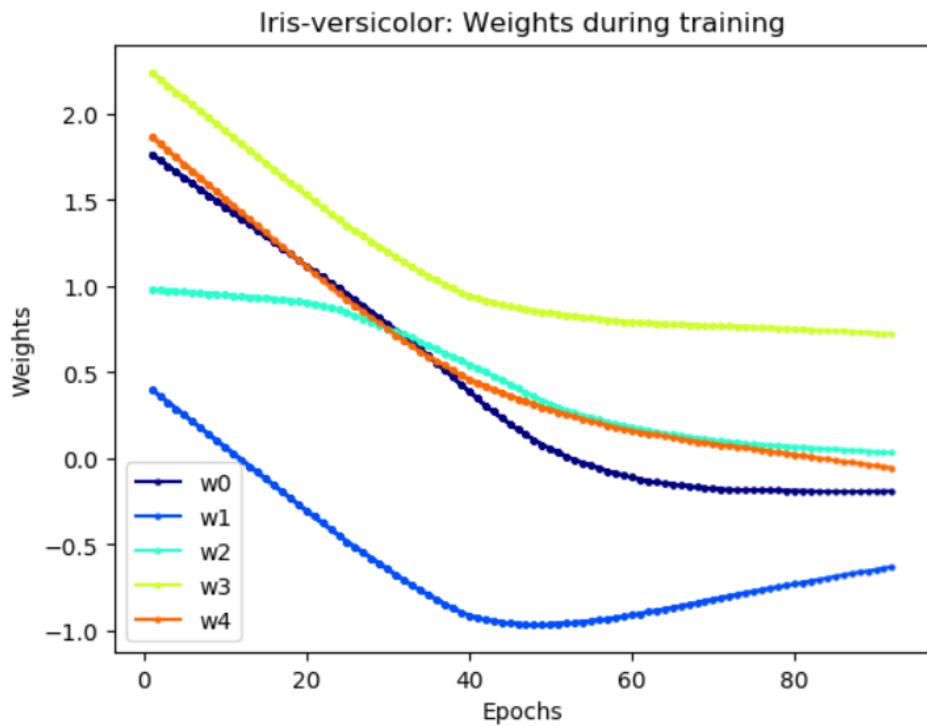


تغییر خطای مجموعه ارزیابی در هر ایپاک برای پرسپترون کلاس Iris-setosa

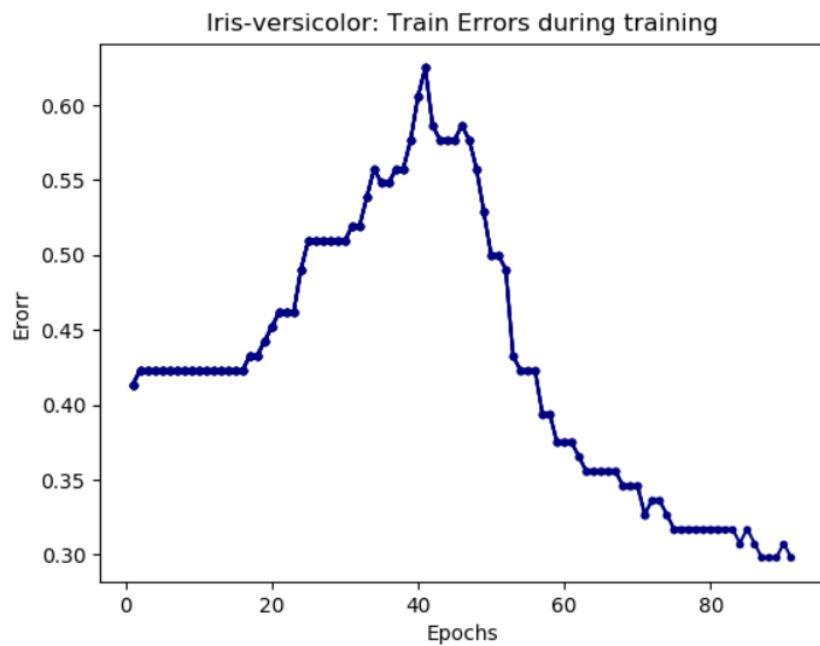
Iris-setosa: Validation Errors during training



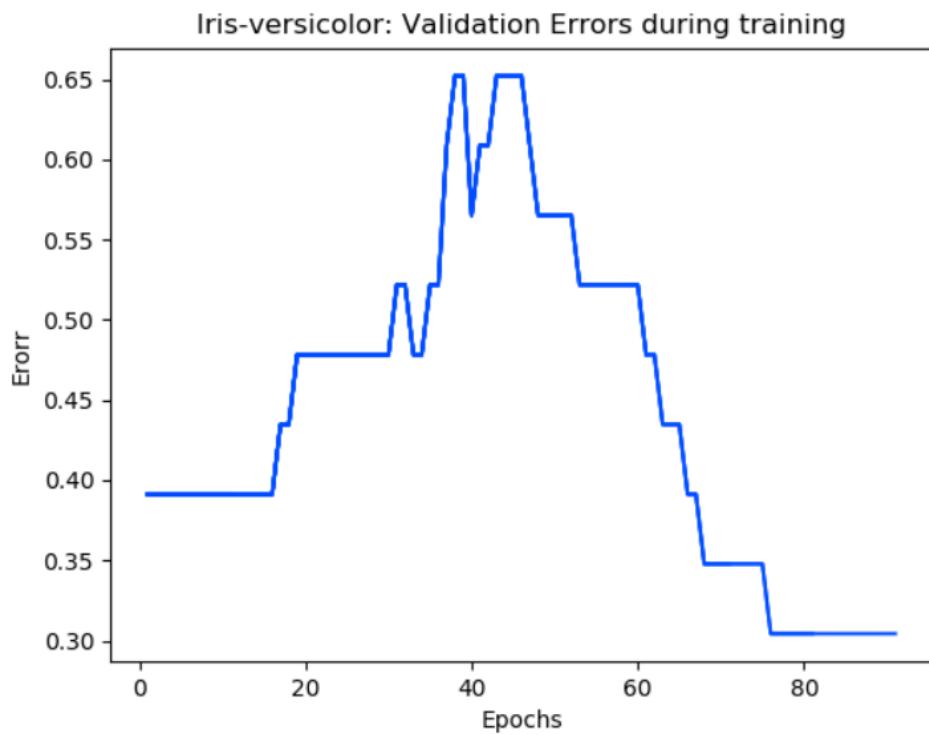
تغییر وزن‌ها در هر ایپاک برای پرسپترون کلاس Iris-versicolor



تغییر خطای مجموعه آموزش در هر ایپاک برای پرسپترون کلاس Iris-versicolor



تغییر خطای مجموعه ارزیابی در هر ایپاک برای پرسپترون کلاس Iris-versicolor

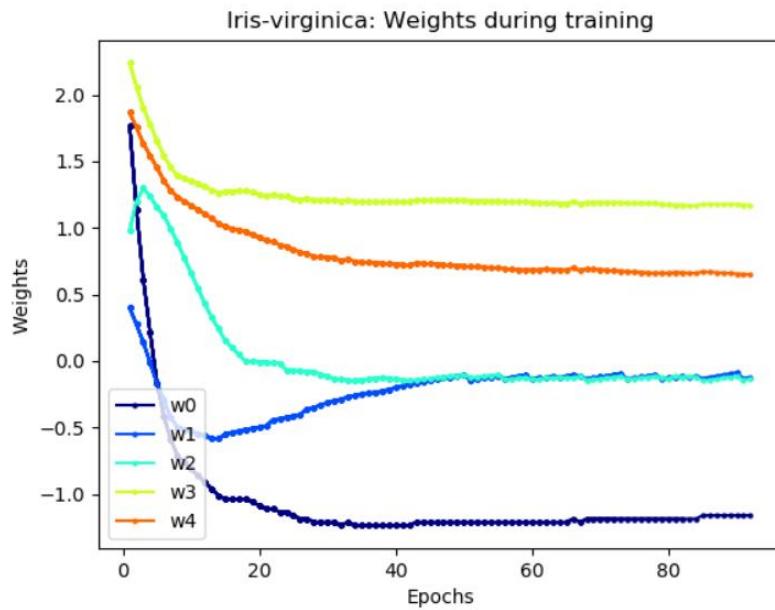


خطای k-fold و خطای عادی آموزش و خطای تست در پایان:

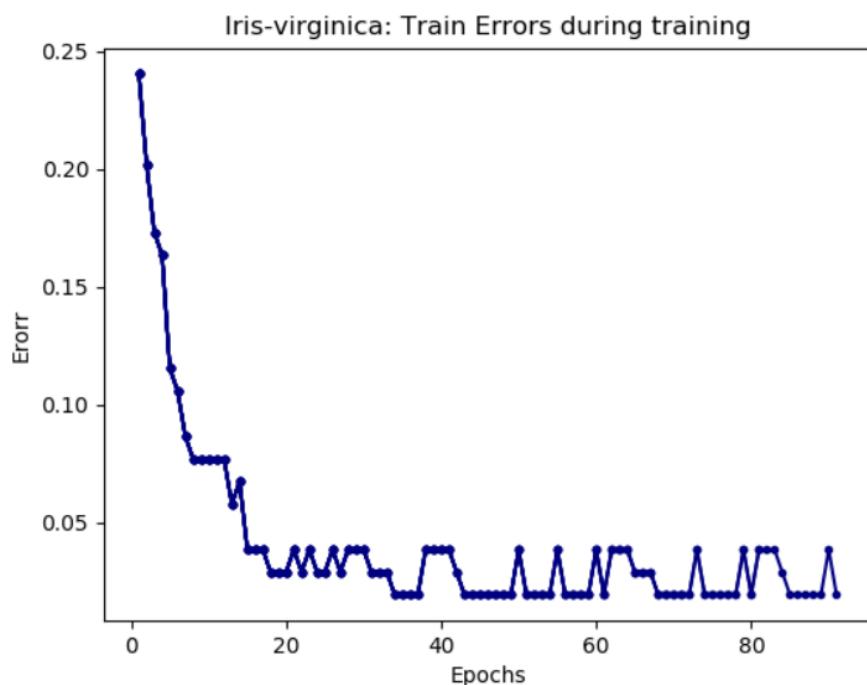
```
Calculating 10-Fold Cross Validation ...
10-fold-cross-validation error is: 0.17714285714285713
Train Error(Ordinary Error, not k-fold)= 0.14423076923076922
Test Error= 0.30434782608695654
```

خروجی‌های فایل ورودی دوم پرسپترون

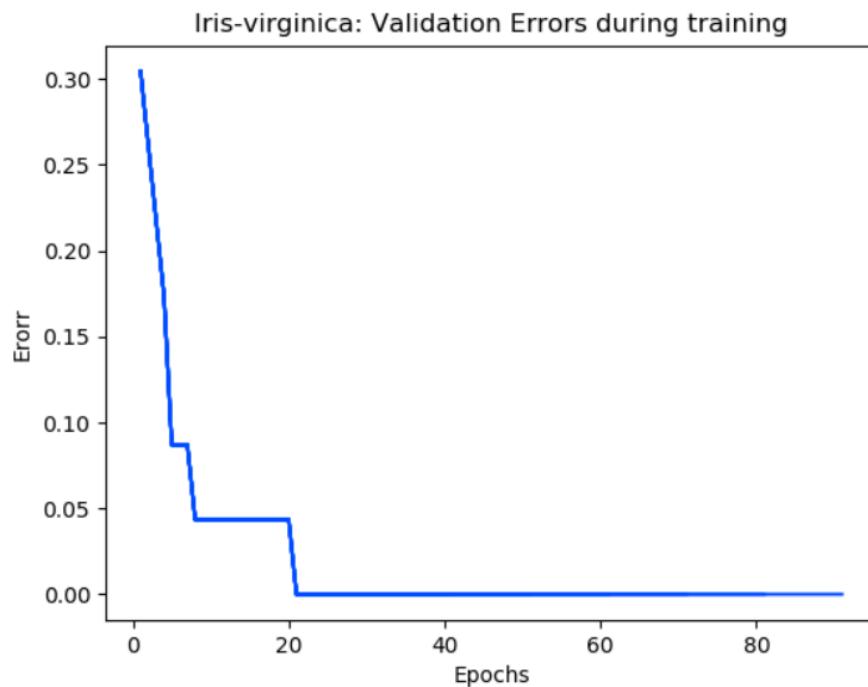
تغییر وزن‌ها در هر ایپاتک برای پرسپترون کلاس Iris-virginica



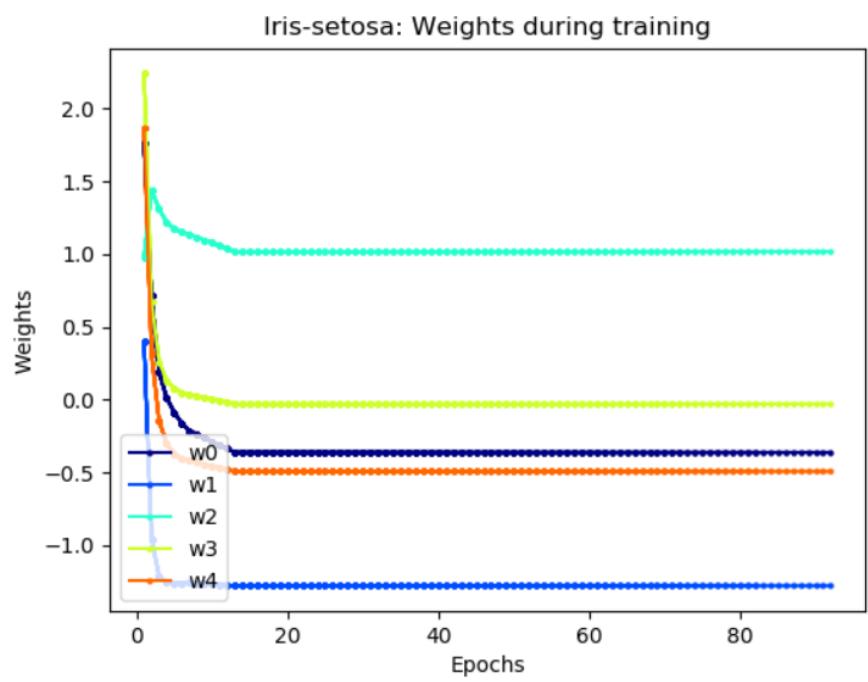
تغییر خطای مجموعه آموزش در هر ایپاک برای پرسپترون کلاس Iris-virginica



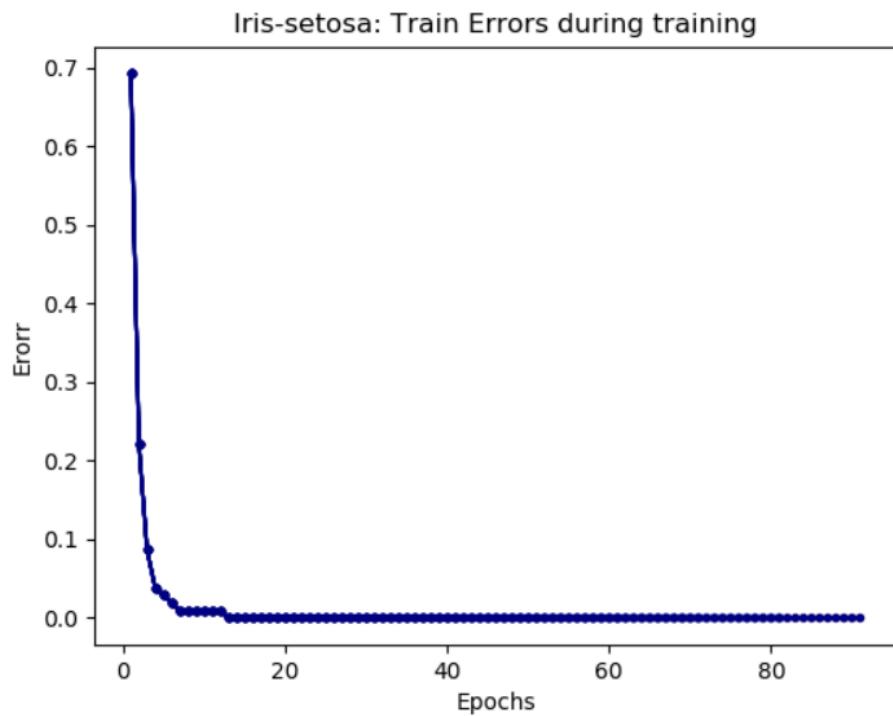
تغییر خطای مجموعه ارزیابی در هر ایپاک برای پرسپترون کلاس Iris-virginica



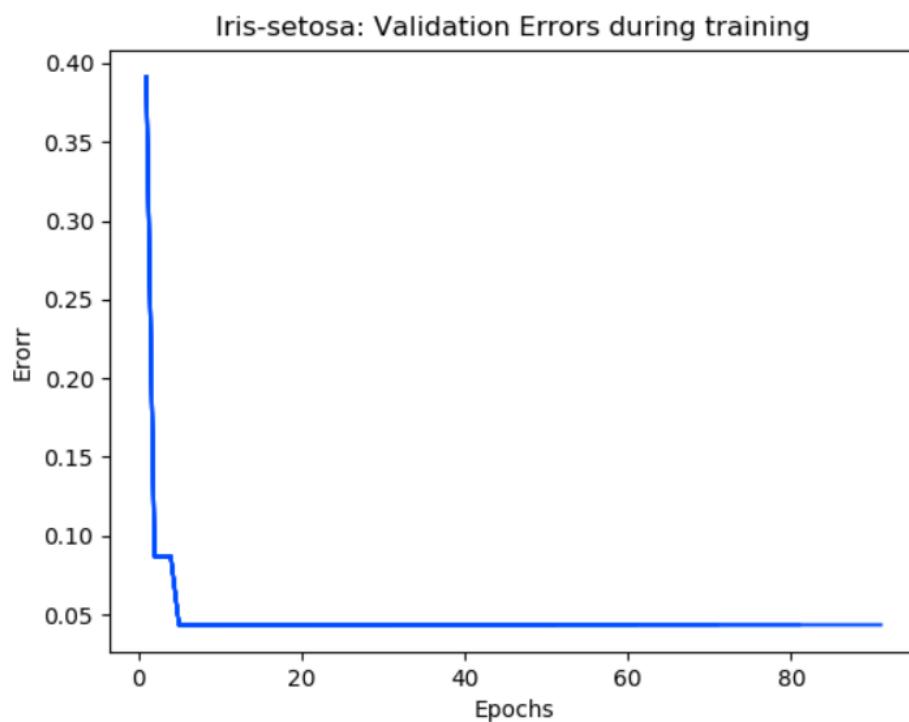
تغییر وزن‌ها در هر ایپاک برای پرسپترون کلاس Iris-setosa



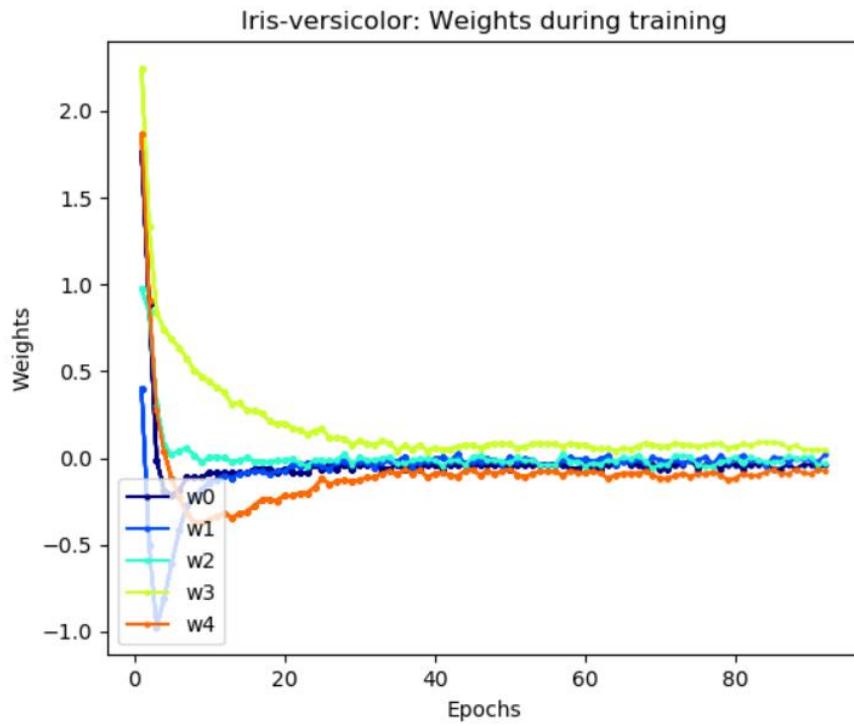
تغییر خطای مجموعه آموزش در هر ایپاک برای پرسپترون کلاس Iris-setosa



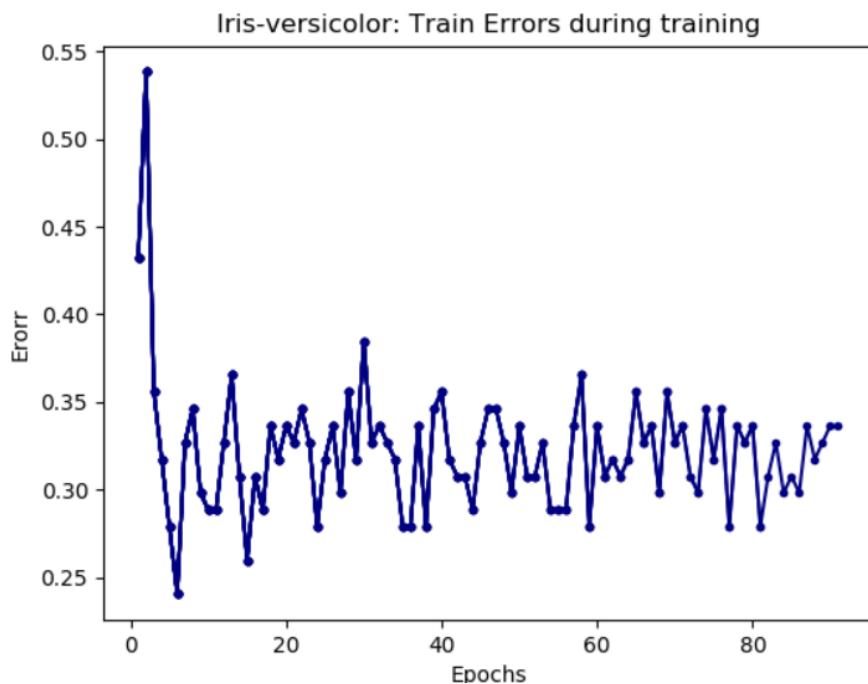
تغییر خطای مجموعه ارزیابی در هر ایپاک برای پرسپترون کلاس Iris-setosa



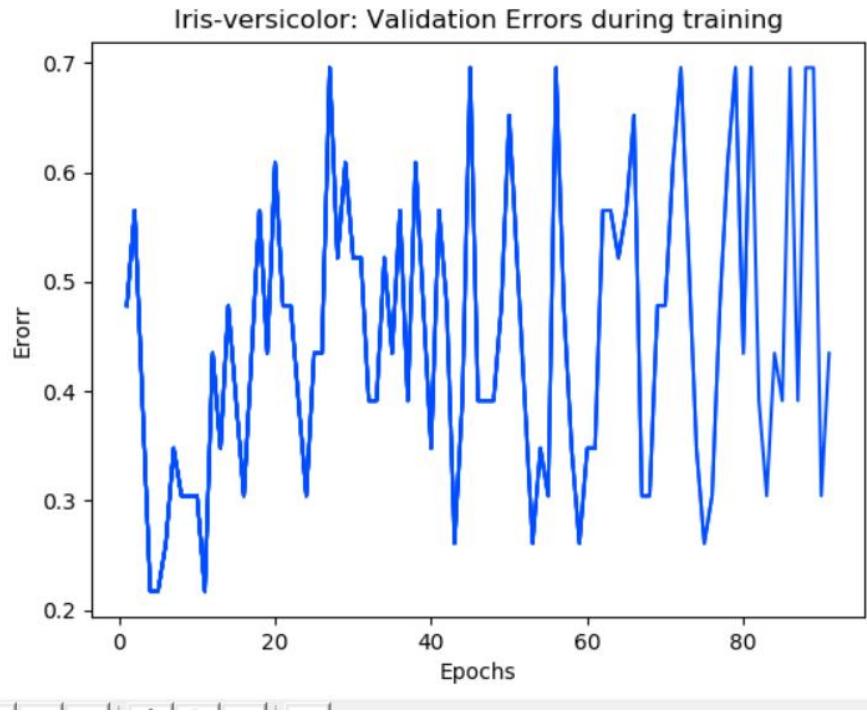
تغییر وزن‌ها در هر ایپاک برای پرسپترون کلاس Iris-versicolor



تغییر خطای مجموعه آموزش در هر ایپاک برای پرسپترون کلاس Iris-versicolor



تغییر خطای مجموعه ارزیابی در هر ایپاک برای پرسپترون کلاس Iris-versicolor



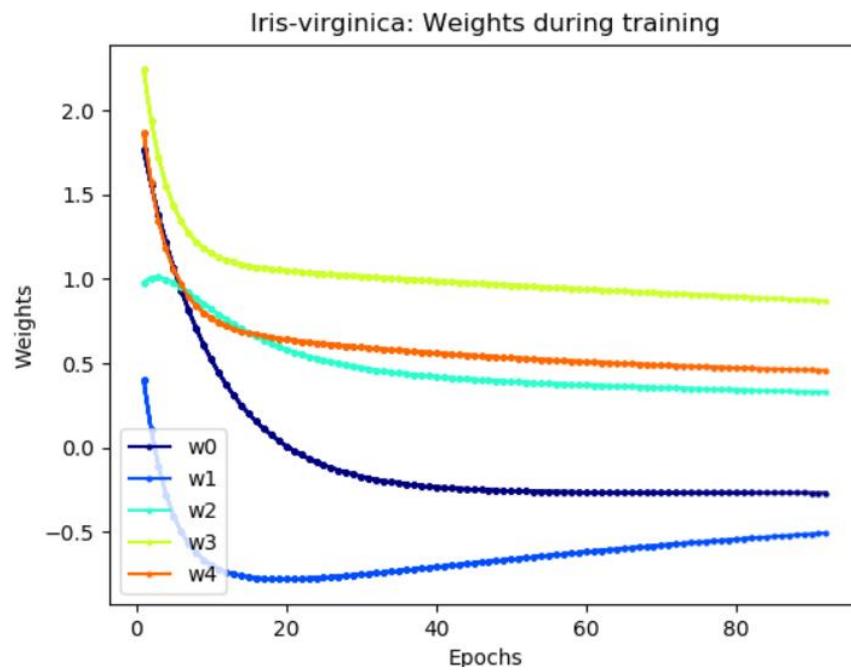
خطای k-fold و خطای عادی آموزش و خطای تست در پایان:

```
Calculating 10-Fold Cross Validation ...
10-fold-cross-validation error is: 0.05
Train Error(Ordinary Error, not k-fold)= 0.038461538461538464
Test Error= 0.0
```

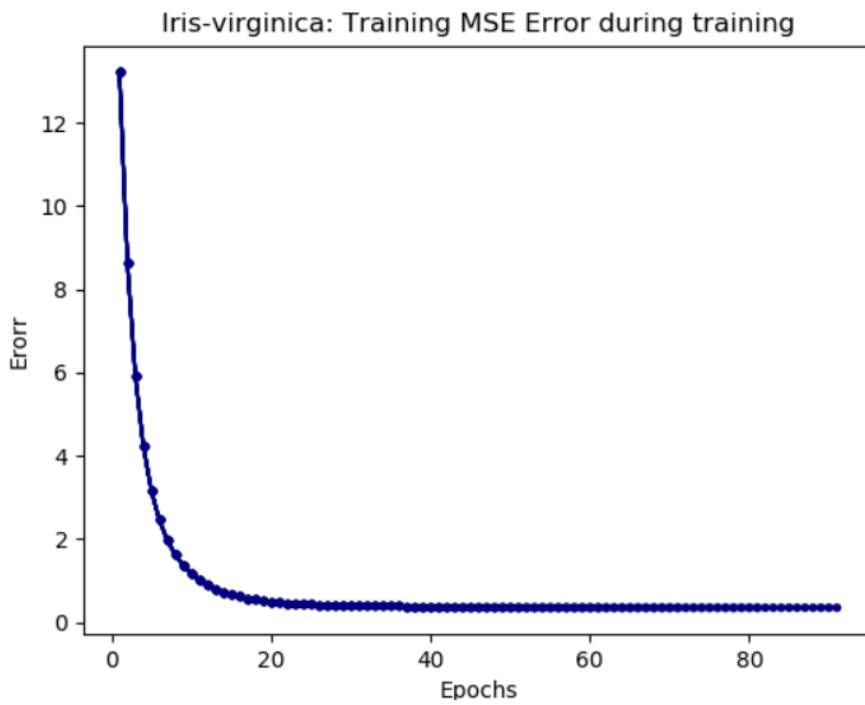
همین طور که مشاهده می شود در دو حالت با ضریب های یادگیری متفاوت دو پرسپترون خطای آموزش و تست و k - fold متفاوتی دارند و اگر به نمودارهای تغییرات وزن ها و خطای مجموعه آموزش و ارزیابی نگاه کنیم متوجه می شویم آن ها نیز متفاوت در نتیجه تغییر ضریب یادگیری بر نتایج تاثیر می گذارد.

خروجی فایل اول ورودی برای آدالاین یکی در برابر همه:

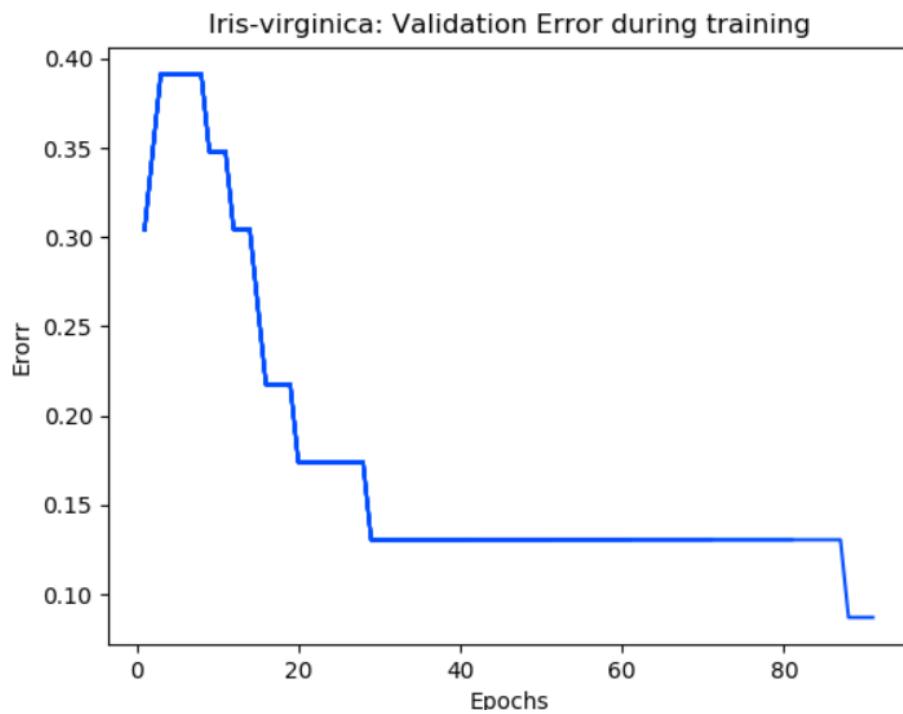
Iris-virginica تغییر وزن ها در هر ایپاتک برای آدالاین کلاس



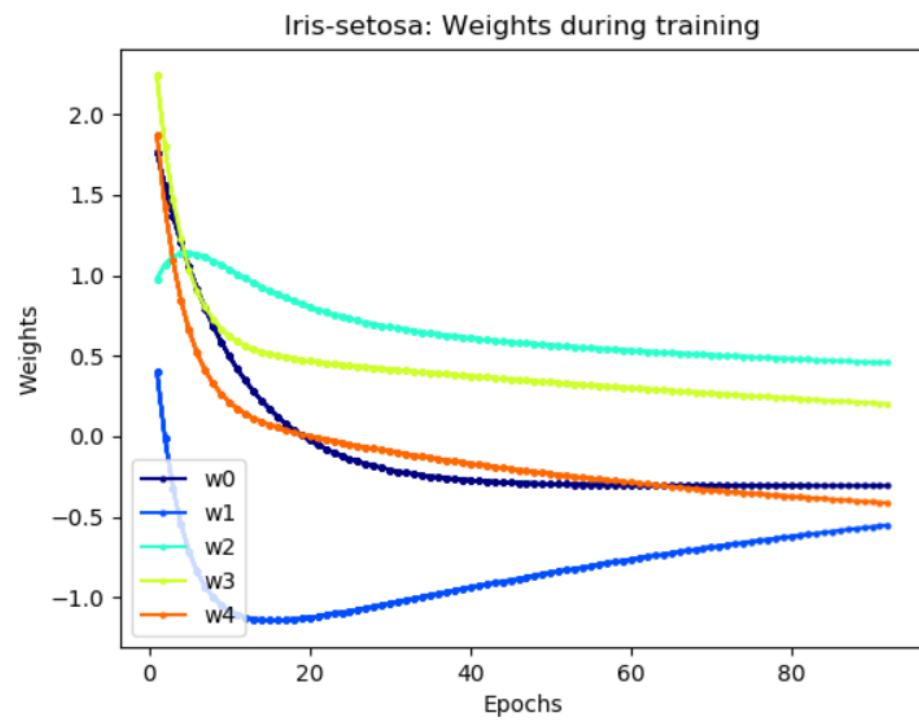
تغییر خطای مجموعه آموزش در هر ایپاک برای آدالاین کلاس Iris-virginica



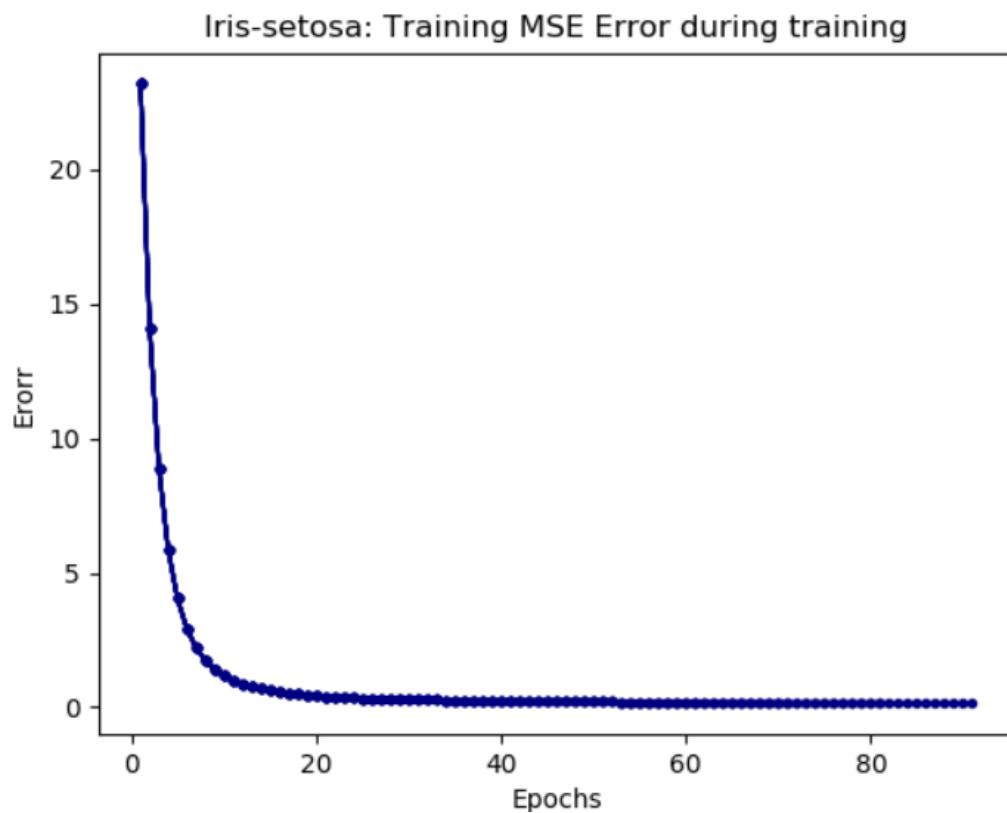
تغییر خطای مجموعه ارزیابی در هر ایپاک برای آدالاین کلاس Iris-virginica



تغییر وزن‌ها در هر ایپاتک برای آدالین کلاس Iris-setosa

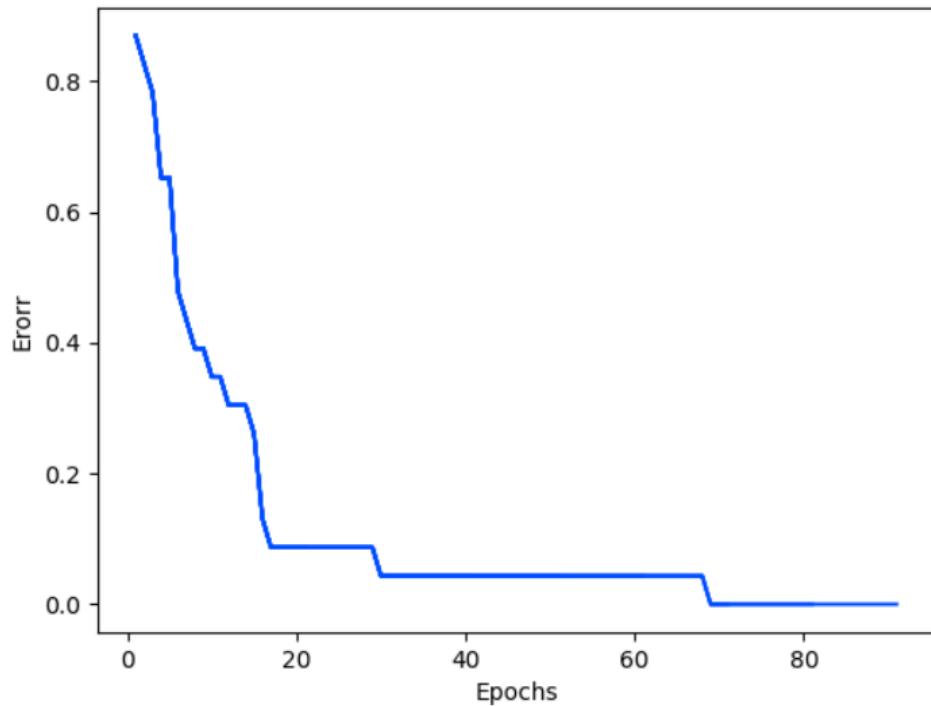


تغییر خطای مجموعه آموزش در هر اپاک برای آدالین کلاس Iris-setosa

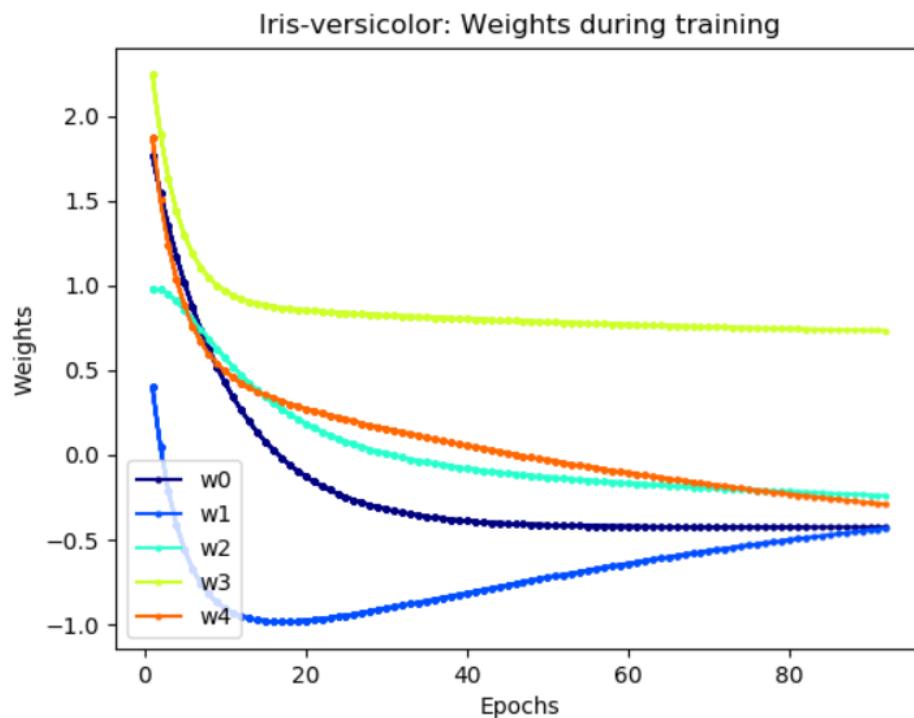


تغییر خطای مجموعه ارزیابی در هر اپاک برای آدالین کلاس Iris-setosa

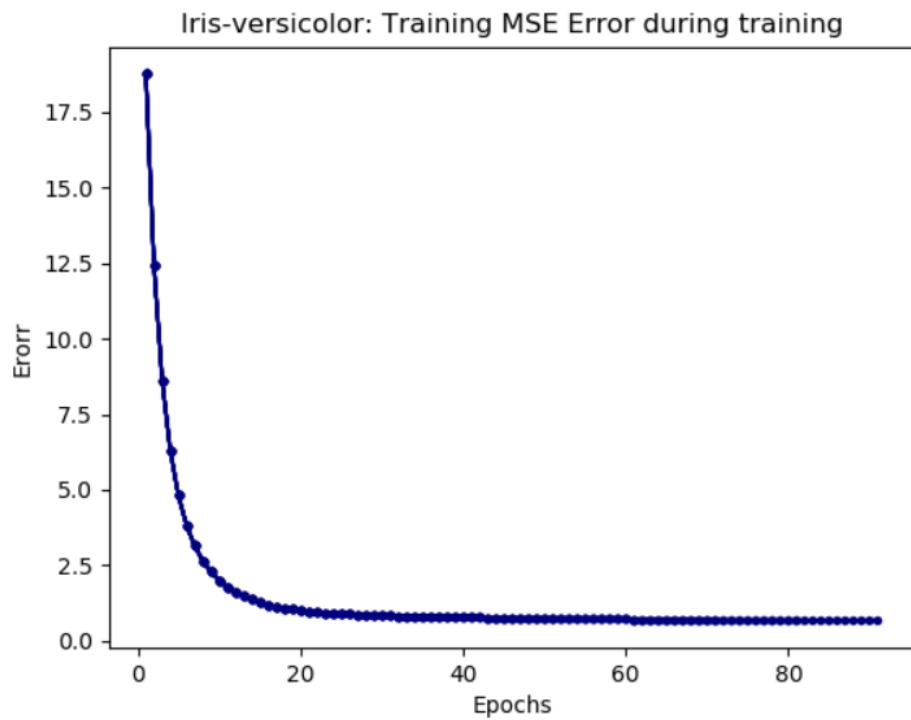
Iris-setosa: Validation Error during training



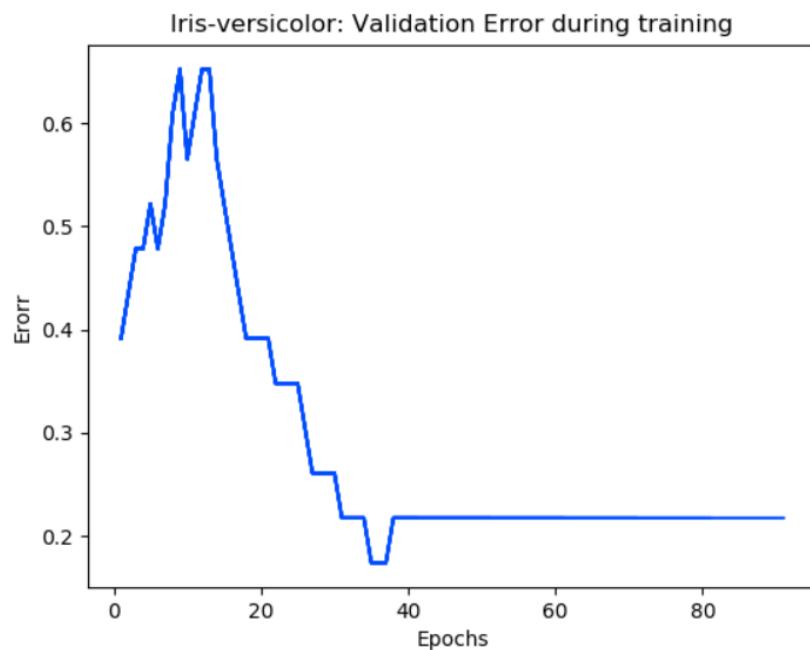
تغییر وزن‌ها در هر ایپاک برای آدالین کلاس Iris-versicolor



تغییر خطای مجموعه آموزش در هر ایپاک برای آدالین کلاس Iris-versicolor



تغییر خطای مجموعه ارزیابی در هر ایپاک برای آدالین کلاس Iris-versicolor



خطای k-fold و خطای عادی آموزش و خطای تست در پایان:

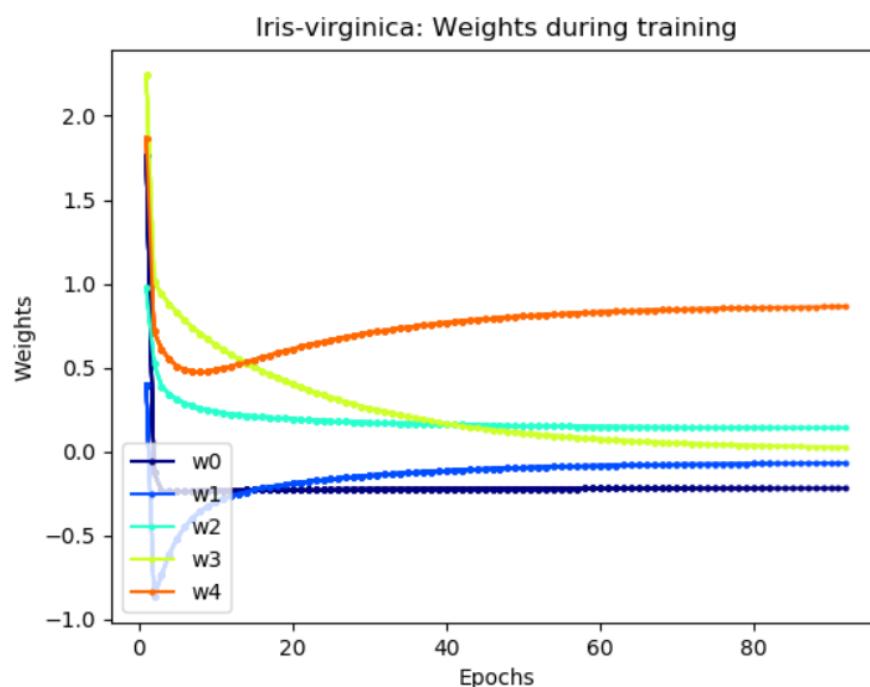
```

Calculating 10-Fold Cross Validation ...
10-fold-cross-validation error is: 0.18714285714285717
Train Error(Ordinary Error, not k-fold)= 0.16346153846153846
Test Error= 0.30434782608695654

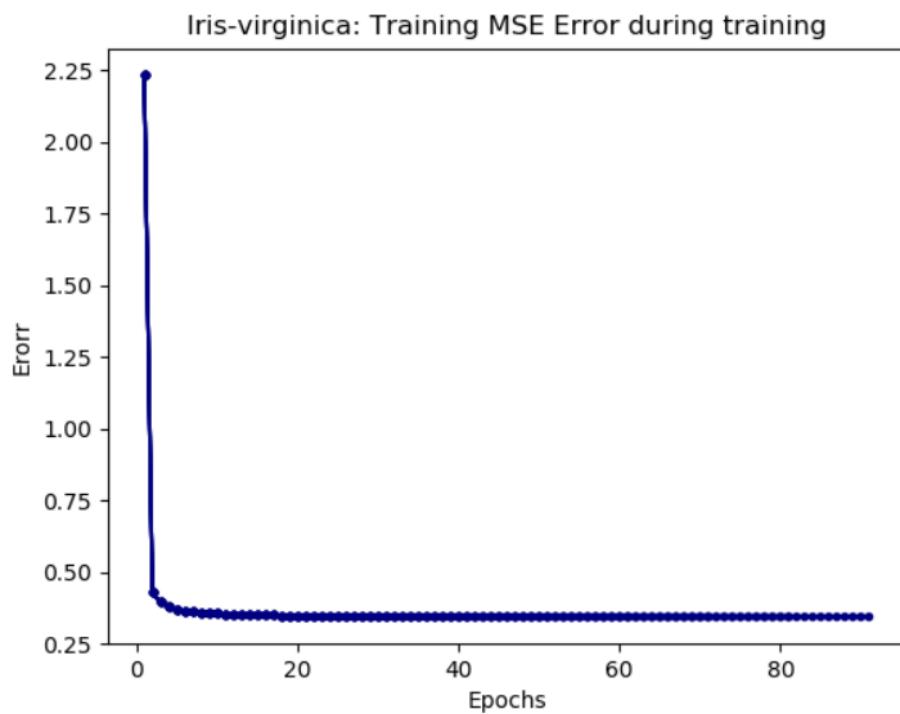
```

خروجی فایل دوم ورودی برای آدالین یکی در برابر همه:

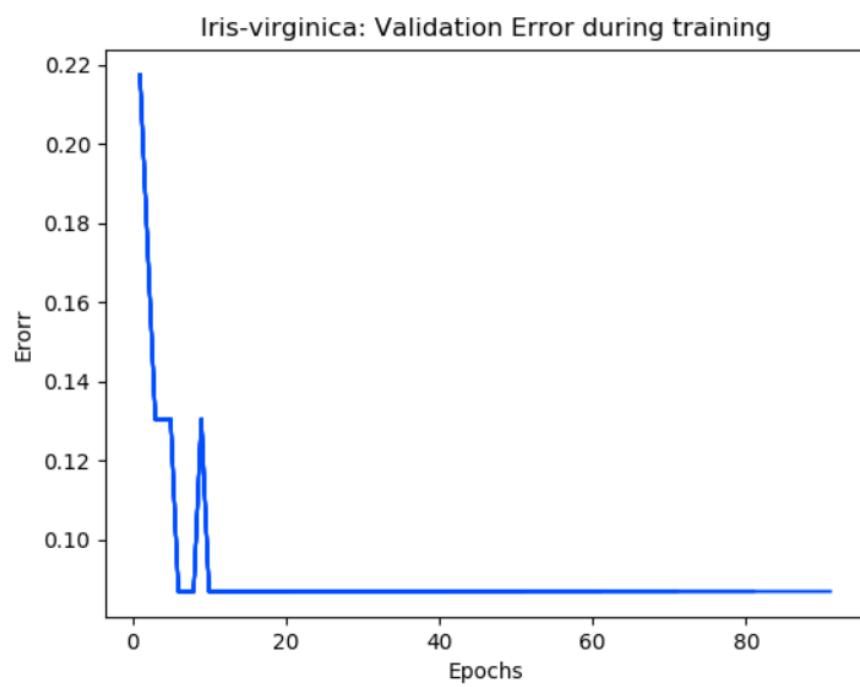
تغییر وزن‌ها در هر ایپاک برای آدالین کلاس Iris-virginica



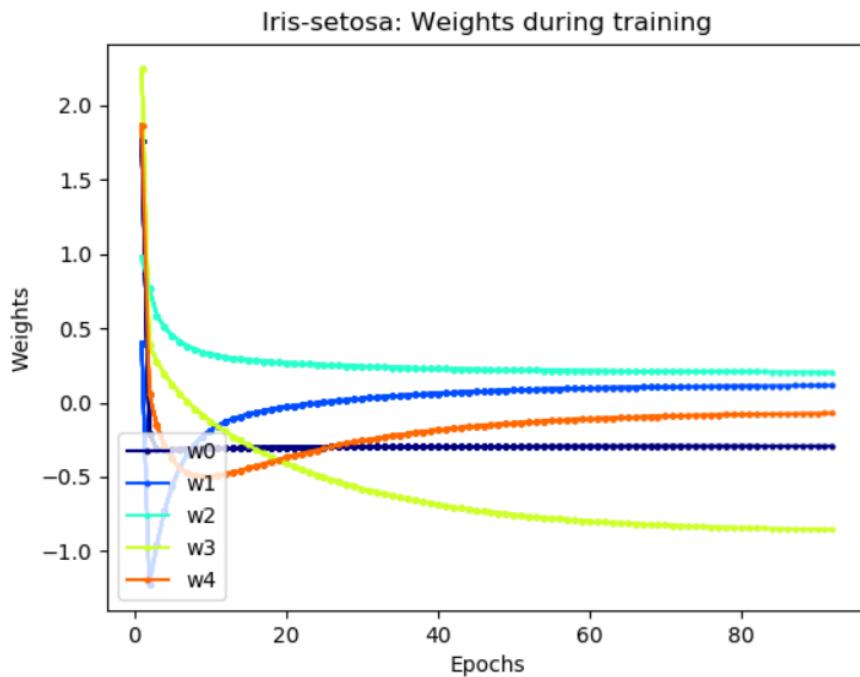
تغییر خطای مجموعه آموزش در هر ایپاک برای آدالین کلاس Iris-virginica



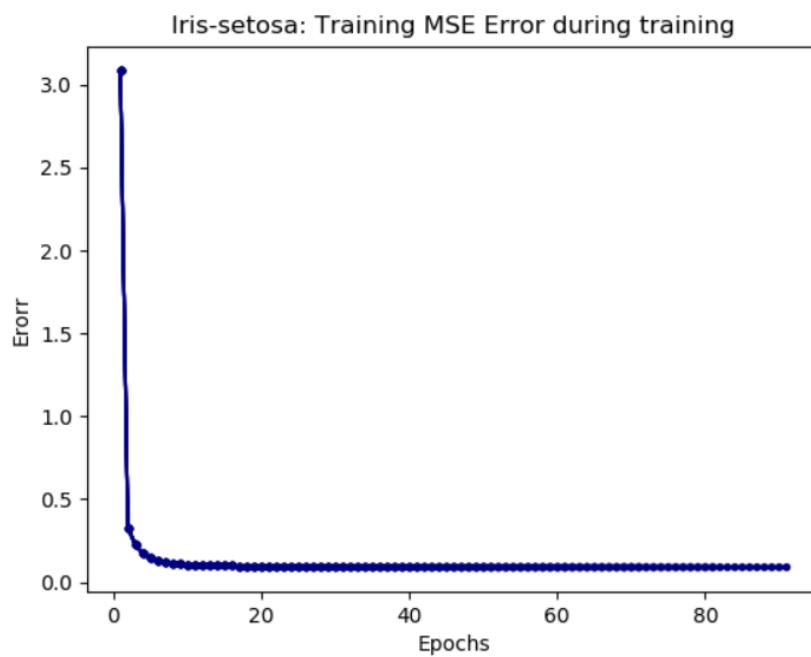
تغییر خطای مجموعه ارزیابی در هر ایپاک برای آدالین کلاس Iris-virginica



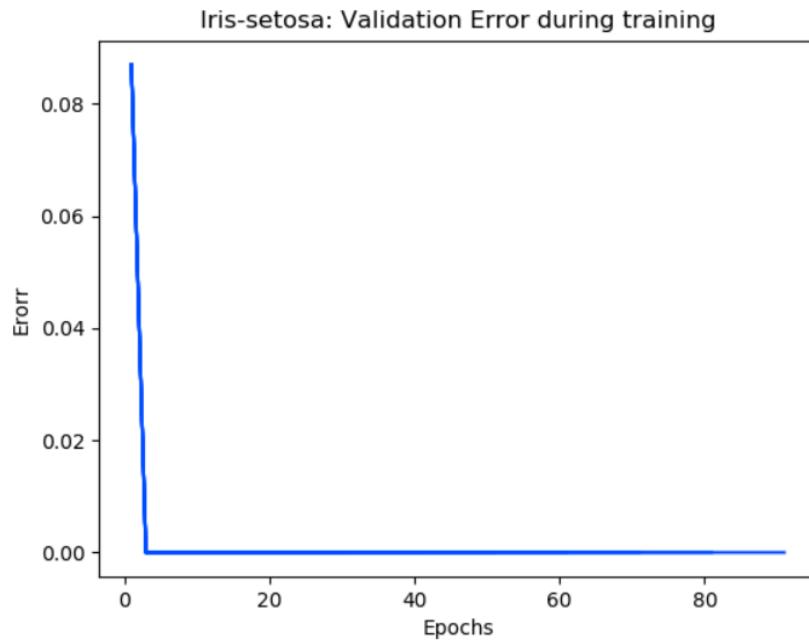
تغییر وزن ها در هر ایپاک برای آدالین کلاس Iris-setosa



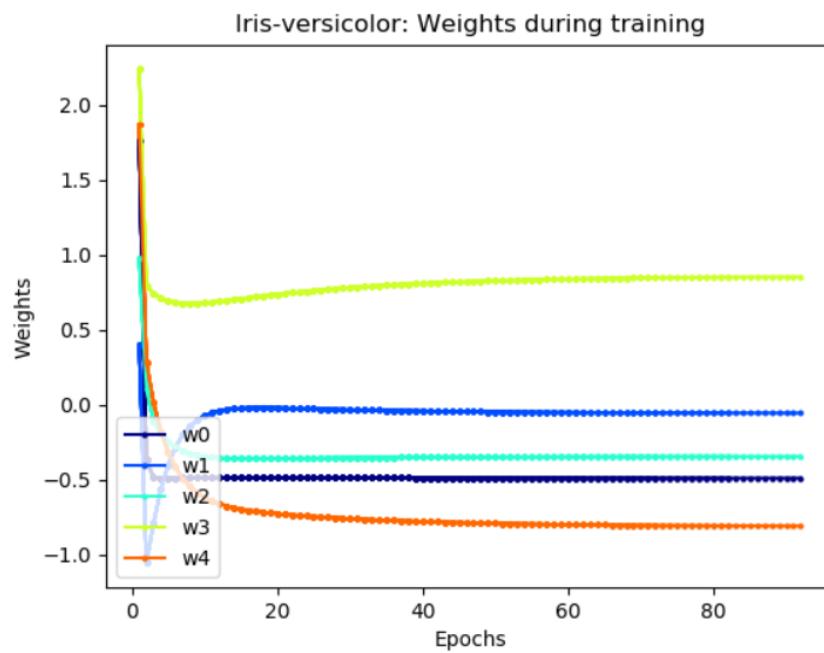
تغییر خطای مجموعه آموزش در هر ایپاک برای آدالین کلاس Iris-setosa



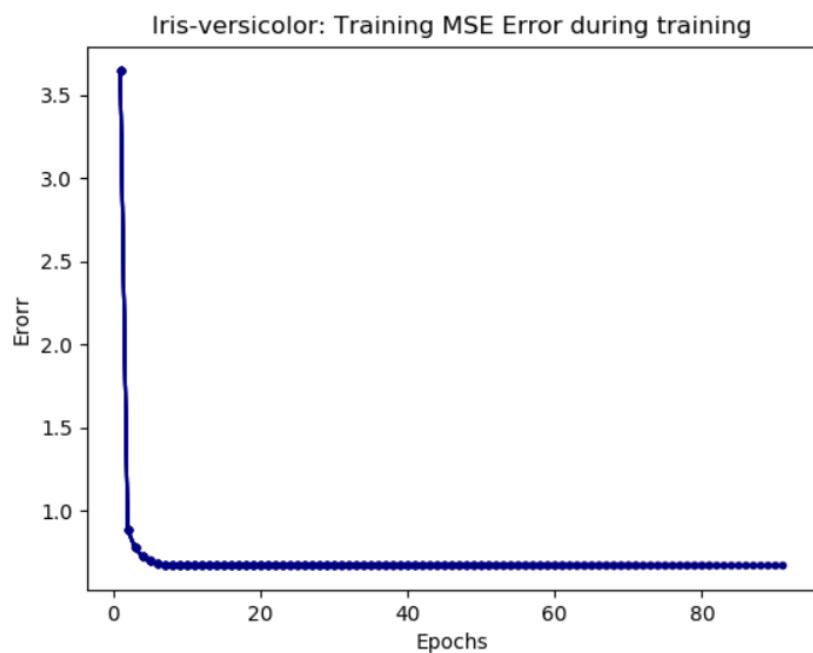
تغییر خطای مجموعه ارزیابی در هر ایپاک برای آدالین کلاس Iris-setosa



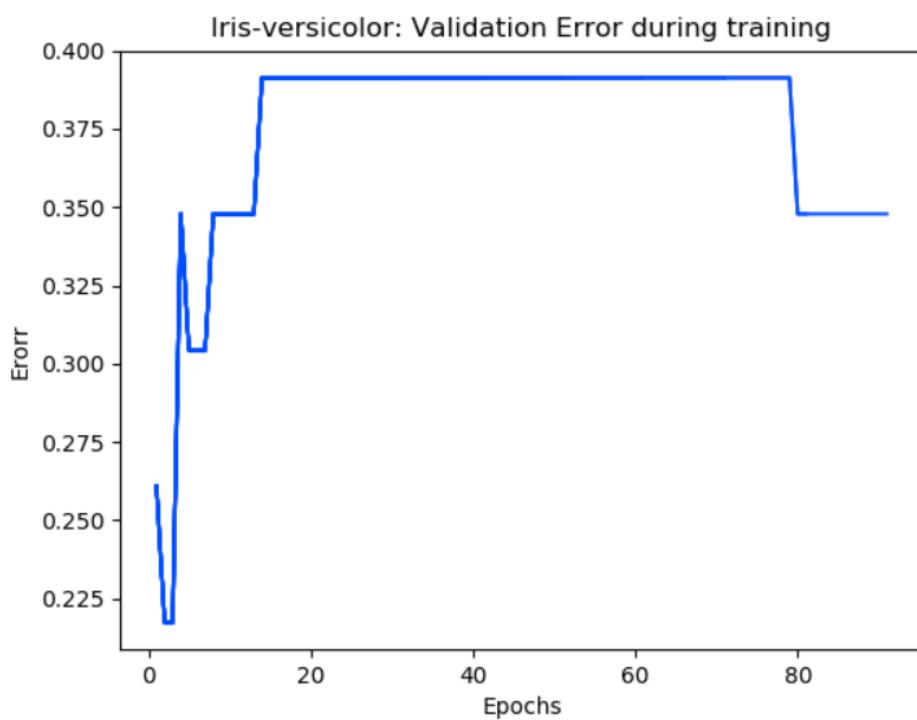
تغییر وزن‌ها در هر ایپاک برای آدالین کلاس Iris-versicolor



تغییر خطای مجموعه آموزش در هر ایپاک برای آدالین کلاس Iris-versicolor



تغییر خطای مجموعه ارزیابی در هر اپاک برای آدالین کلاس Iris-versicolor



خطای k-fold و خطای عادی آموزش و خطای تست در پایان:

```

Calculating 10-Fold Cross Validation ...
10-fold-cross-validation error is: 0.20714285714285716
Train Error(Ordinary Error, not k-fold)= 0.17307692307692307
Test Error= 0.34782608695652173

```

در دو فراخوانی adaline one vs all با پارامترهای یکسان بجز ضرایب یادگیری یکسان را اگر مقایسه کنیم می‌بینیم در خطای K-fold و مجموعه‌ی تست متفاوت اند همچنین نمودارهای خطای ارزیابی و آموزش و تغییر وزن‌ها هم متفاوت است به همین دلیل تغییر ضریب یادگیری تاثیر گذار است. یکی از دلایل آن این است که آدالین از گرادیان نزولی استفاده می‌کند در مینیمم محلی گیر می‌کند و با تغییر ضریب یادگیری ممکن است در مینیمم‌های مختلفی بی‌افتد.

در هر دو حالت پرسپترون و آدالین تغییر ضریب یادگیری موجب تغییر نتایج شد.

#### D بررسی تغییر وزن‌ها در کار کرد الگرویتم‌ها

برای این کار هر دو الگرویتم را با پارامترهای کاملاً یکسان به طوری که فقط وزن‌های اولیه متفاوتی داشته باشند فرا می‌خوانیم.

برای پرسپترون و آدالین کد run adalines one vs all و run perceptron one vs all ورودی input-perceptrons-one-vs-all-6-2.json و input-perceptrons-one-vs-all-6-1.json فرا می‌خوانیم.

```

input-perceptrons-one-vs-all-6-1.json
1  {
2      "ratio_of_train_set":0.70,
3      "ratio_of_valid_set":0.15,
4      "ratio_of_test_set":0.15,
5      "learning_rate":0.001,
6      "max_epochs":100,
7      "cut_error": -1,
8      "random_state":0,
9      "K_fold": 10,
10     "order":"1"
11 }

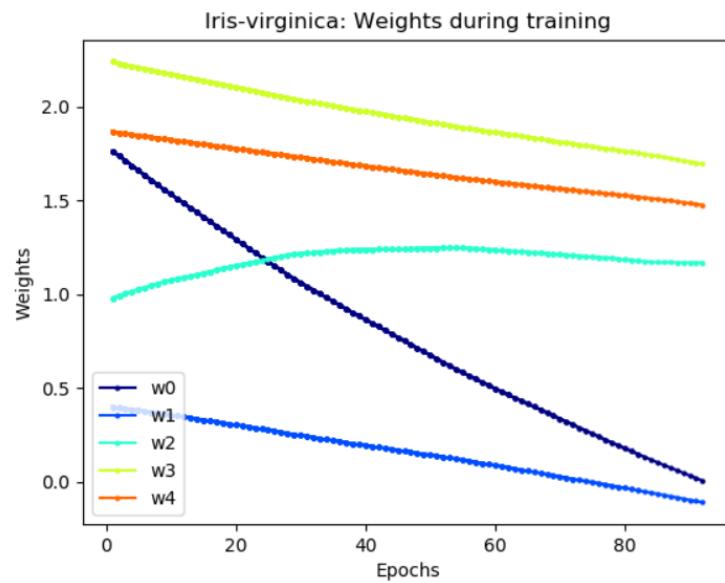
```

```
input-perceptrons-one-vs-all-6-2.json
1  {
2      "ratio_of_train_set":0.70,
3      "ratio_of_valid_set":0.15,
4      "ratio_of_test_set":0.15,
5      "learning_rate":0.001,
6      "max_epochs":100,
7      "cut_error": -1,
8      "random_state":2,
9      "K_fold": 10,
10     "order":"1"
11 }
```

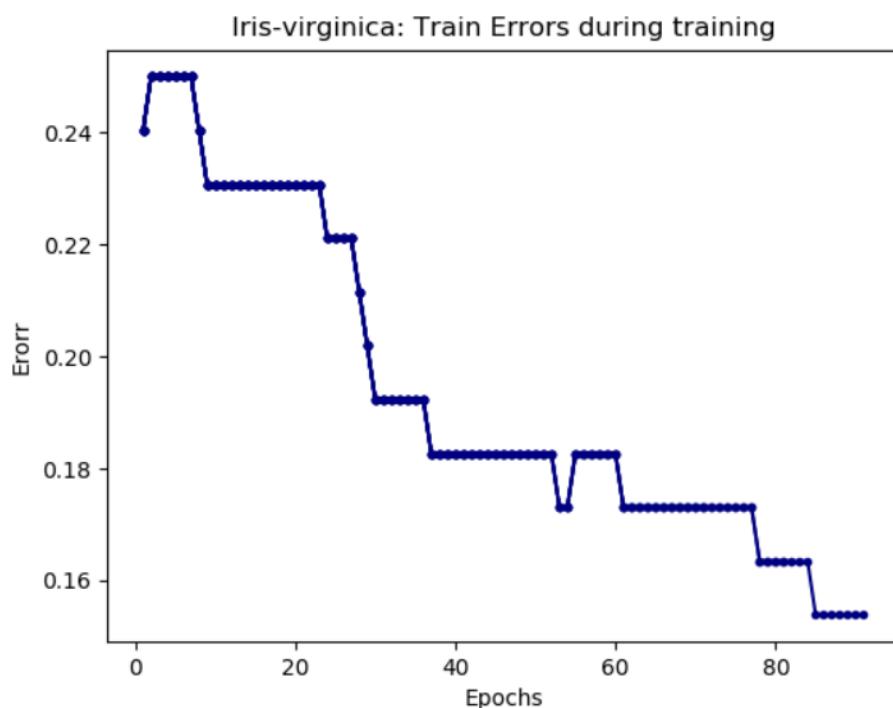
ضریب یادگیری برابر با ۱،۰۰۰ است، حداکثر تعداد اپوکها برابر با ۱۰۰ است، برای پایان آموزش فقط رسیدن به حداکثر تعداد اپوکها چک می‌شود، برای K-Fold تعداد ۱۰ fold در نظر گرفته می‌شود، از فیچرها بدون اضافه کردن فیچرهای مرتبه‌ی ۲ استفاده می‌شود، ۷۰ درصد داده‌ها به آموزش ۱۵ درصد به ارزیابی و ۱۵ درصد به تست اختصاص پیدا می‌کند. از seed ۰ برای تولید وزن‌های تصادفی در فایل پaramترهای یک استفاده می‌کنیم و از seed ۲ برای تولید پارامترهای تصادفی در فایل پارامترهای دوم استفاده کرده ایم تا منجر به تولید وزن های تصادفی و اولیه یکسان شوند. در کدهای all دو الگوریتم را با پارامترهای مختلفی فرا خوانده‌ام که آن کدها را در بالاتر توضیح داده‌ام. با نگاه کردن به نتایج پارامترهای مختلف و انتخاب یک حالت جنرال که استثنای نباشد این پارامترها را انتخاب کرده‌ام.

خروجی‌های فایل ورودی اول پرسپترون

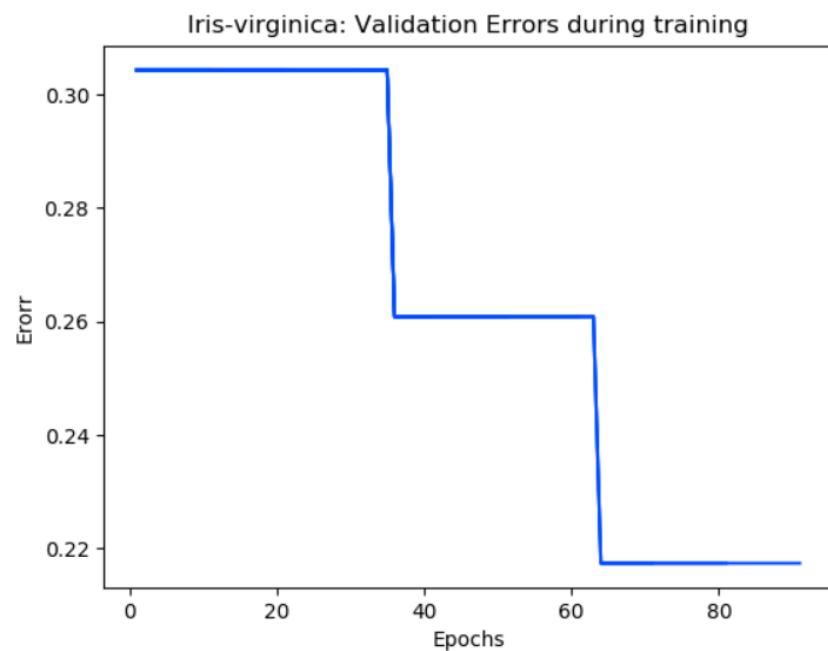
تغییر وزن‌ها در هر ایپاتک برای پرسپترون کلاس Iris-virginica



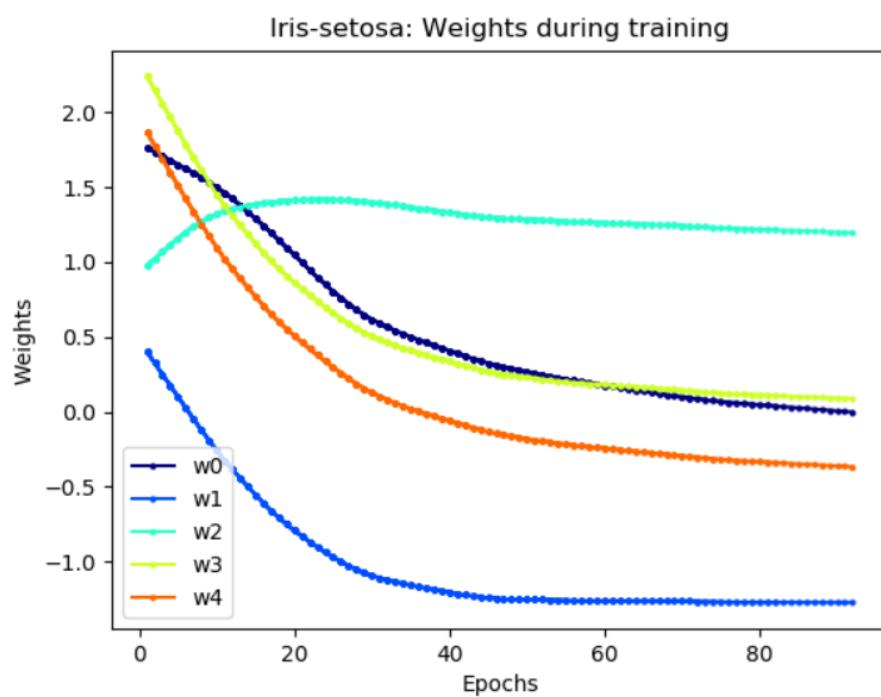
تغییر خطای مجموعه آموزش در هر ایپاک برای پرسپترون کلاس Iris-virginica



تغییر خطای مجموعه ارزیابی در هر ایپاک برای پرسپترون کلاس Iris-virginica

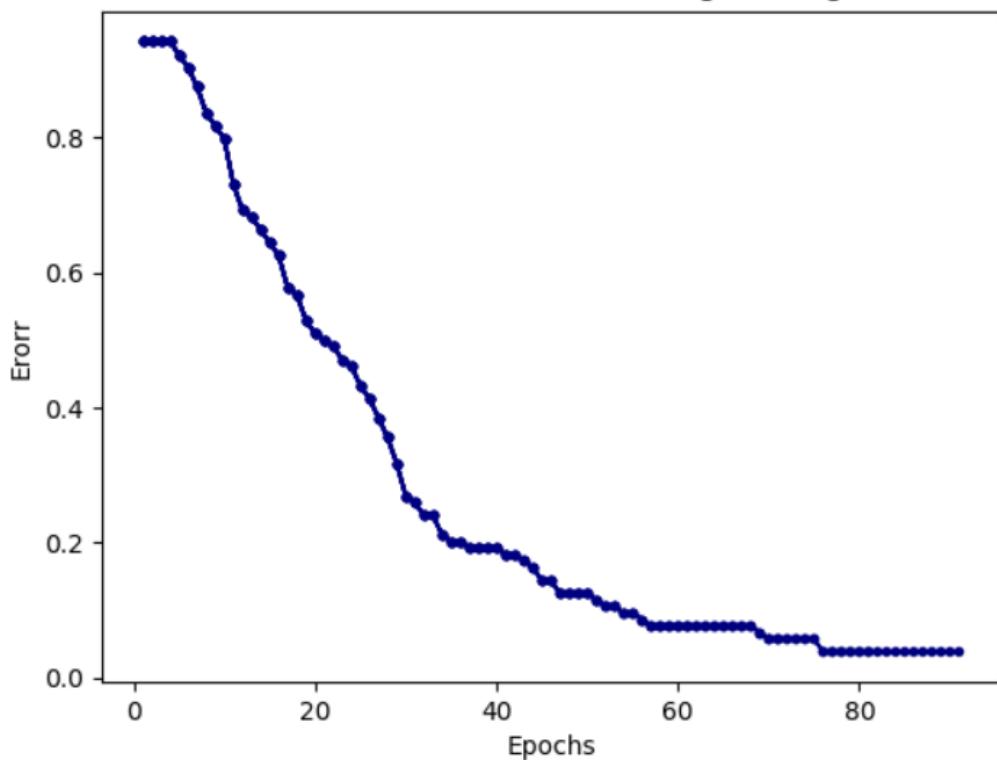


تغییر وزن‌ها در هر ایپاک برای پرسپترون کلاس Iris-setosa



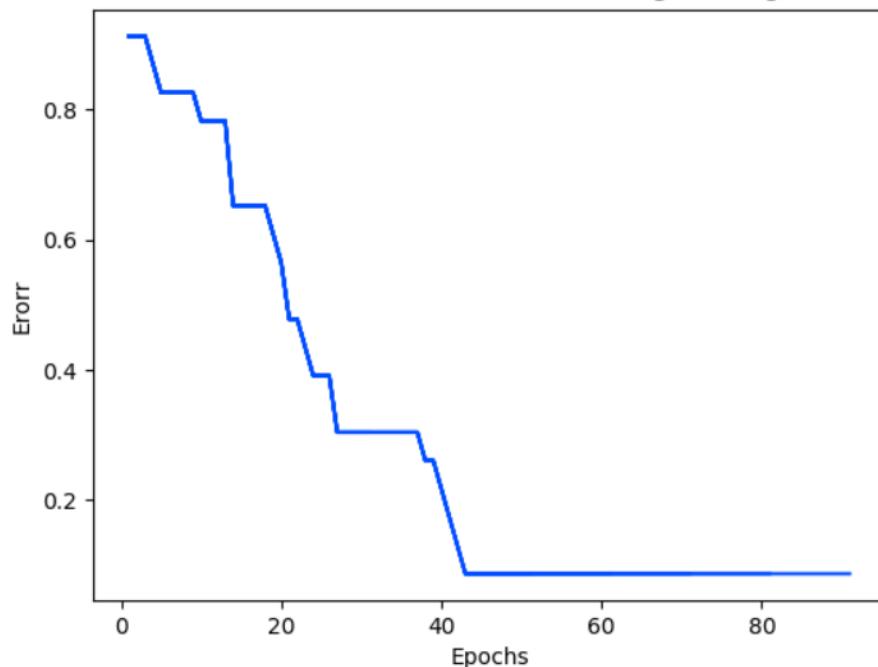
تغییر خطای مجموعه آموزش در هر ایپاک برای پرسپترون کلاس Iris-setosa

Iris-setosa: Train Errors during training

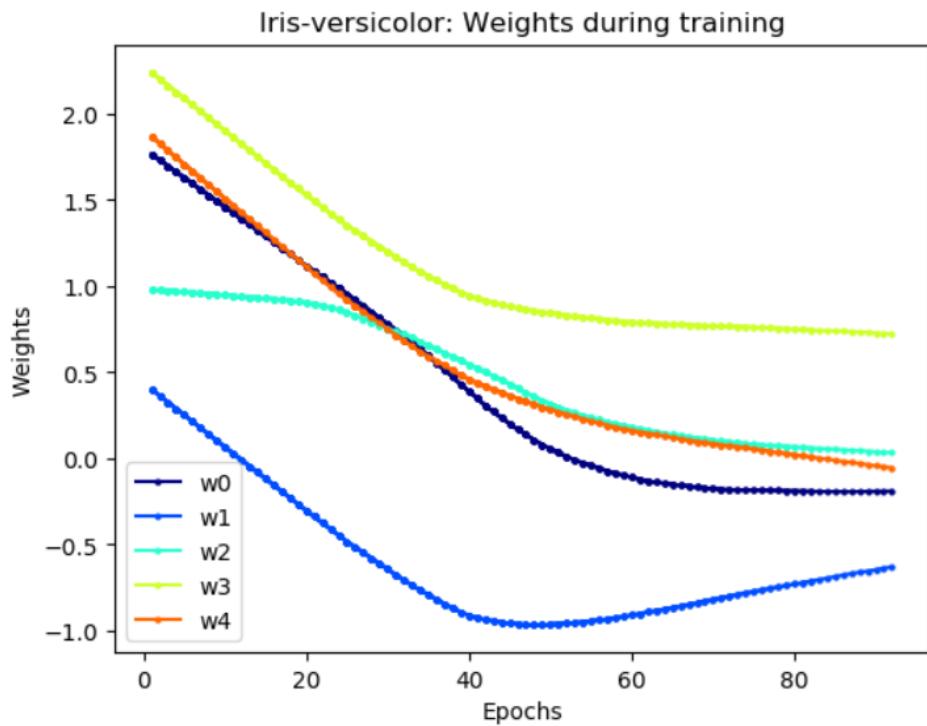


تغییر خطای مجموعه ارزیابی در هر ایپاک برای پرسپترون کلاس Iris-setosa

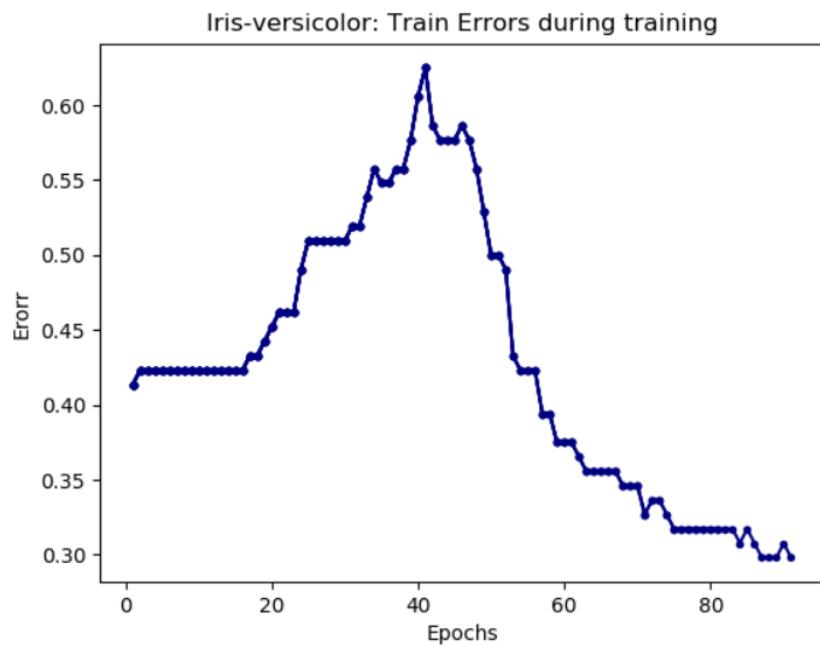
Iris-setosa: Validation Errors during training



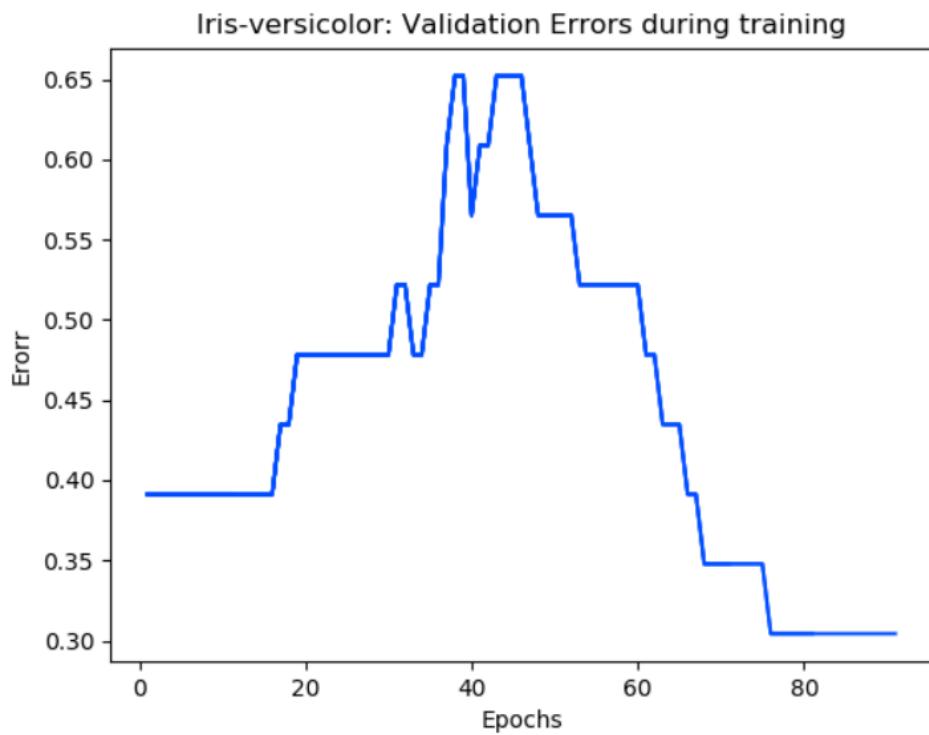
تغییر وزن‌ها در هر ایپاک برای پرسپترون کلاس Iris-versicolor



تغییر خطای مجموعه آموزش در هر ایپاک برای پرسپترون کلاس Iris-versicolor



تغییر خطای مجموعه ارزیابی در هر ایپاک برای پرسپترون کلاس Iris-versicolor

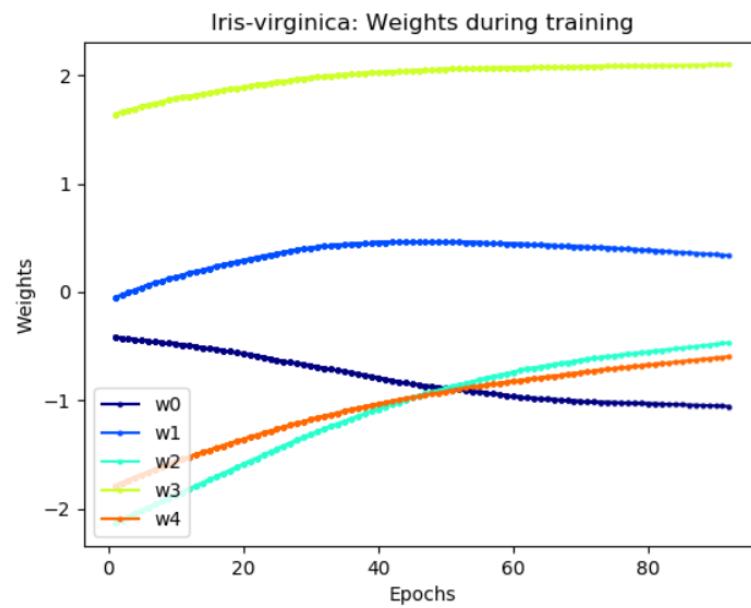


خطای k-fold و خطای عادی آموزش و خطای تست در پایان:

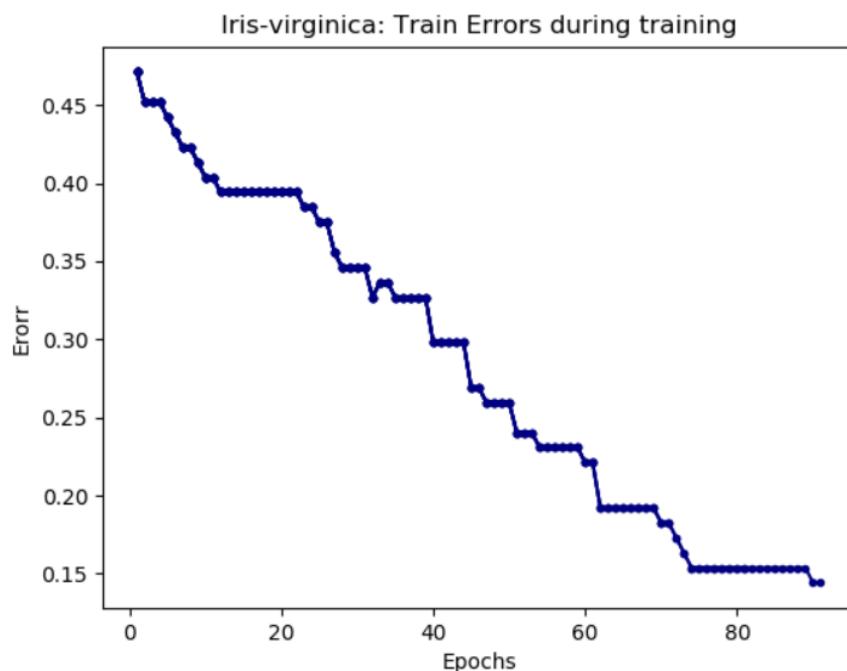
```
Calculating 10-Fold Cross Validation ...
10-fold-cross-validation error is: 0.17714285714285713
Train Error(Ordinary Error, not k-fold)= 0.14423076923076922
Test Error= 0.30434782608695654
```

خروجی‌های فایل ورودی دوم پرسپترون

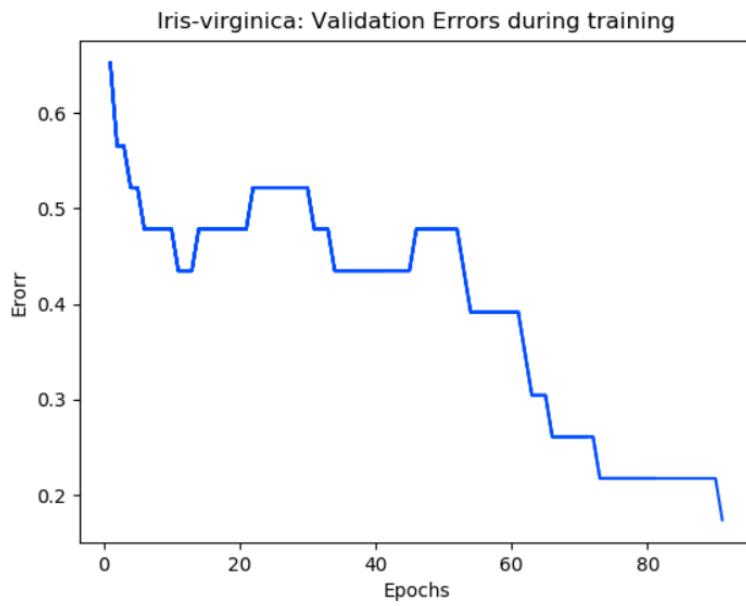
تغییر وزن‌ها در هر ایپاتک برای پرسپترون کلاس Iris-virginica



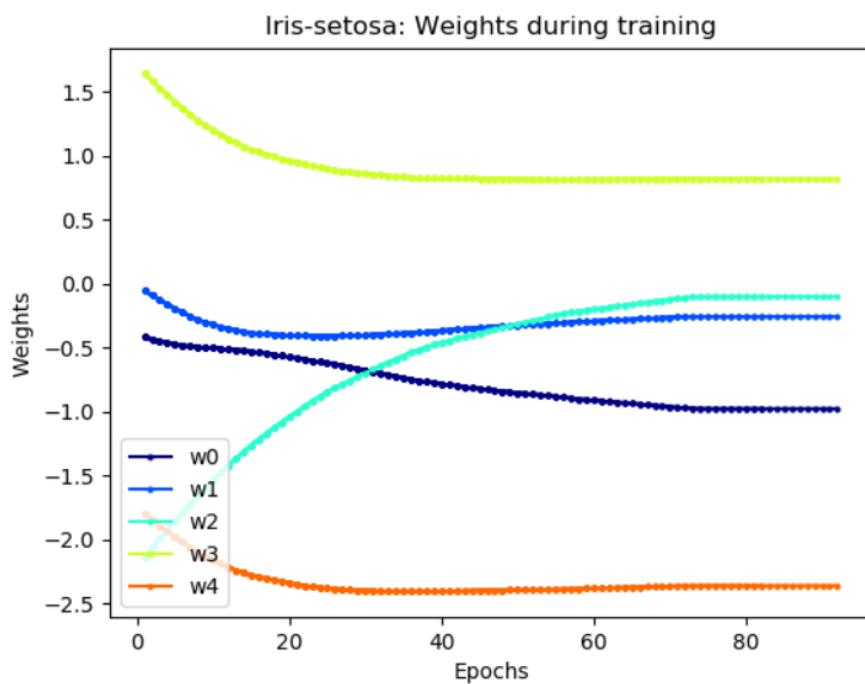
تغییر خطای مجموعه آموزش در هر ایپاک برای پرسپترون کلاس Iris-virginica



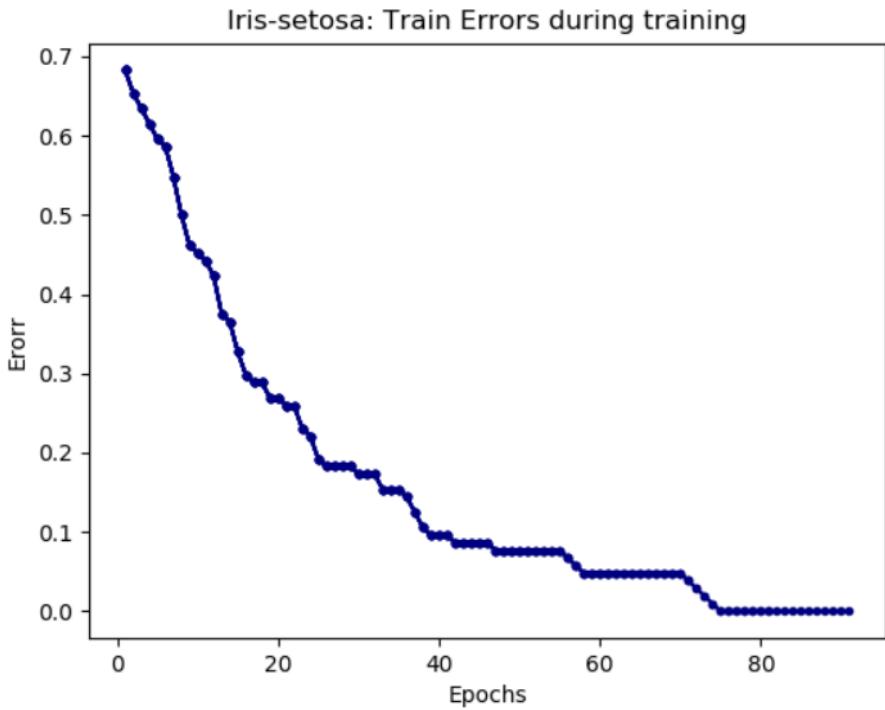
تغییر خطای مجموعه ارزیابی در هر ایپاک برای پرسپترون کلاس Iris-virginica



تغییر وزن‌ها در هر ایپاک برای پرسپترون کلاس Iris-setosa



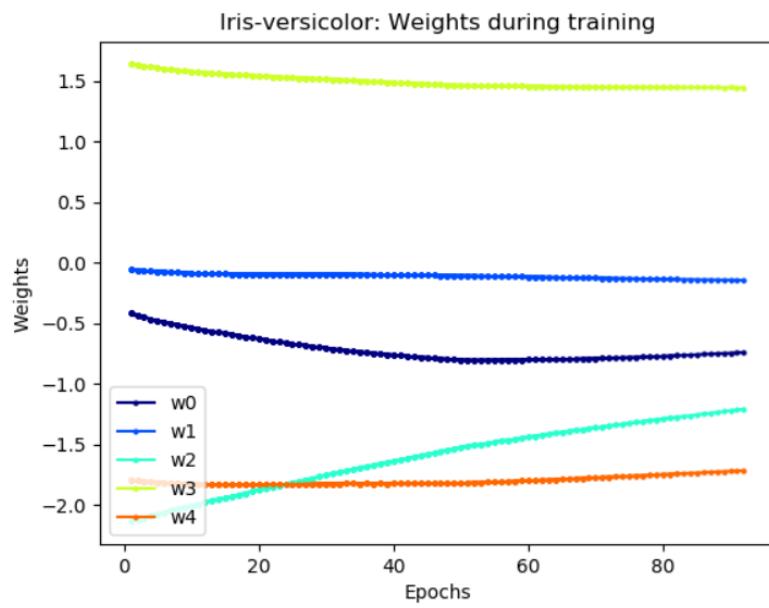
تغییر خطای مجموعه آموزش در هر ایپاک برای پرسپترون کلاس Iris-setosa



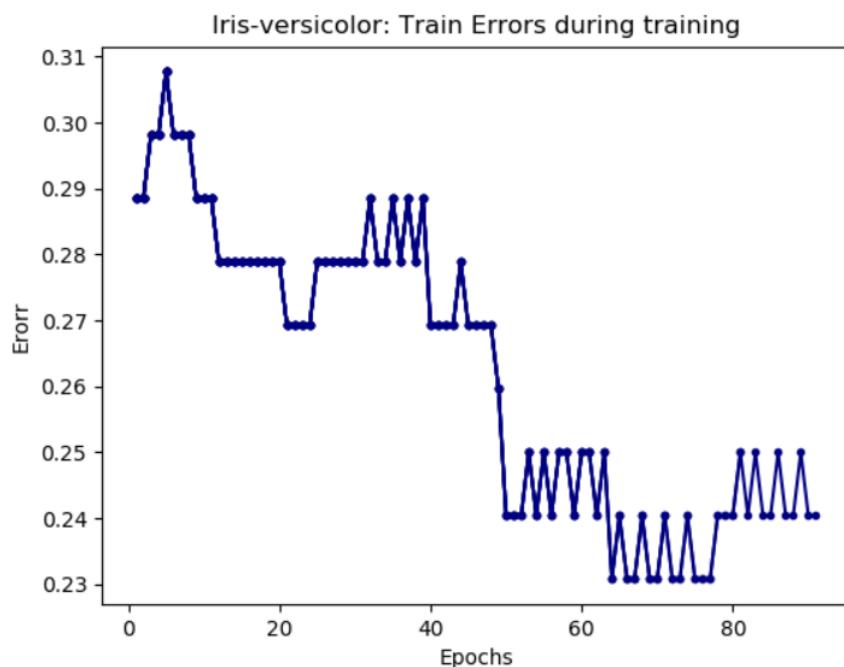
تغییر خطای مجموعه ارزیابی در هر ایپاک برای پرسپترون کلاس Iris-setosa



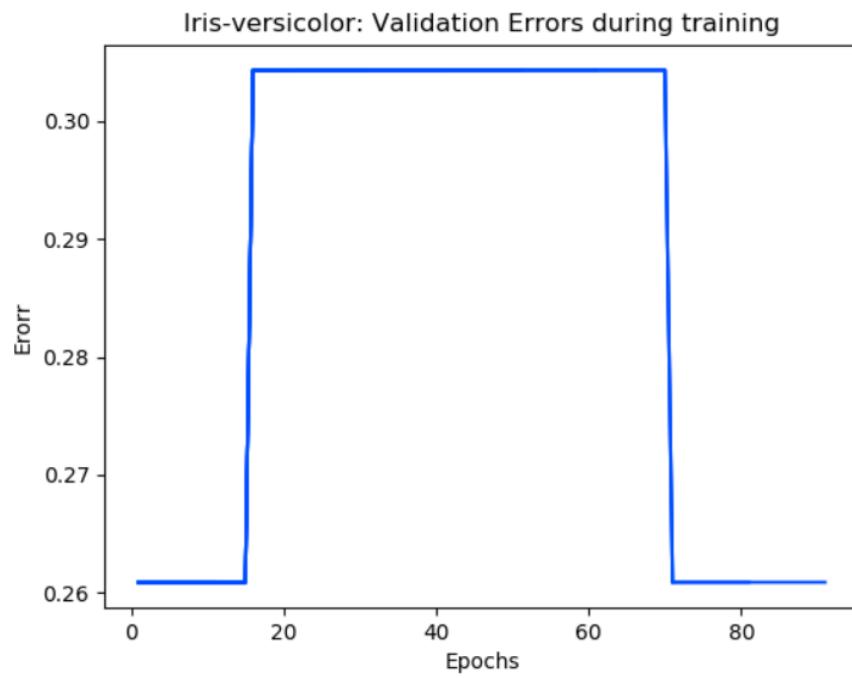
تغییر وزن‌ها در هر ایپاک برای پرسپترون کلاس Iris-versicolor



تغییر خطای مجموعه آموزش در هر ایپاک برای پرسپترون کلاس Iris-versicolor



تغییر خطای مجموعه ارزیابی در هر ایپاک برای پرسپترون کلاس Iris-versicolor



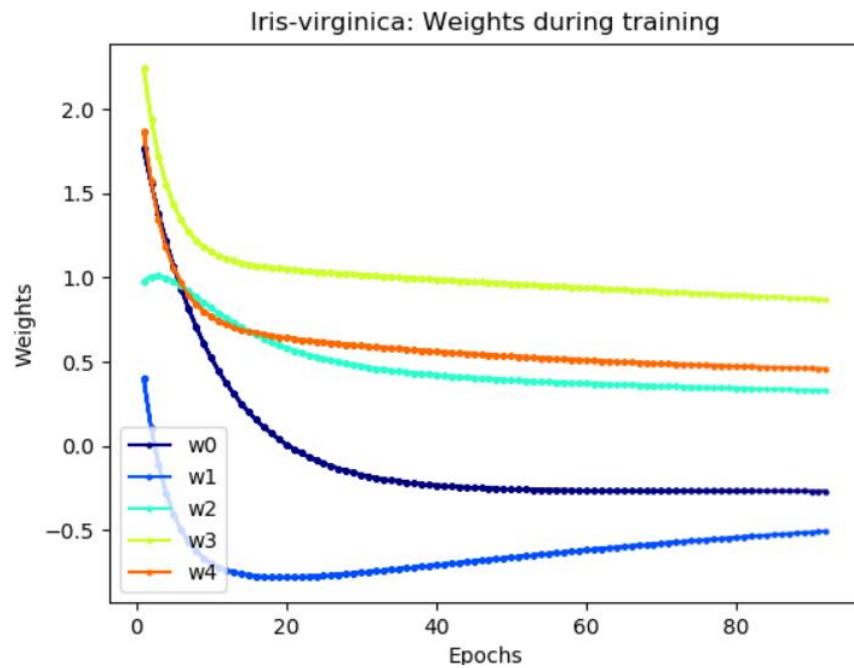
خطای k-fold و خطای عادی آموزش و خطای تست در پایان:

```
Calculating 10-Fold Cross Validation ...
10-fold-cross-validation error is: 0.13142857142857142
Train Error(Ordinary Error, not k-fold)= 0.125
Test Error= 0.17391304347826086
```

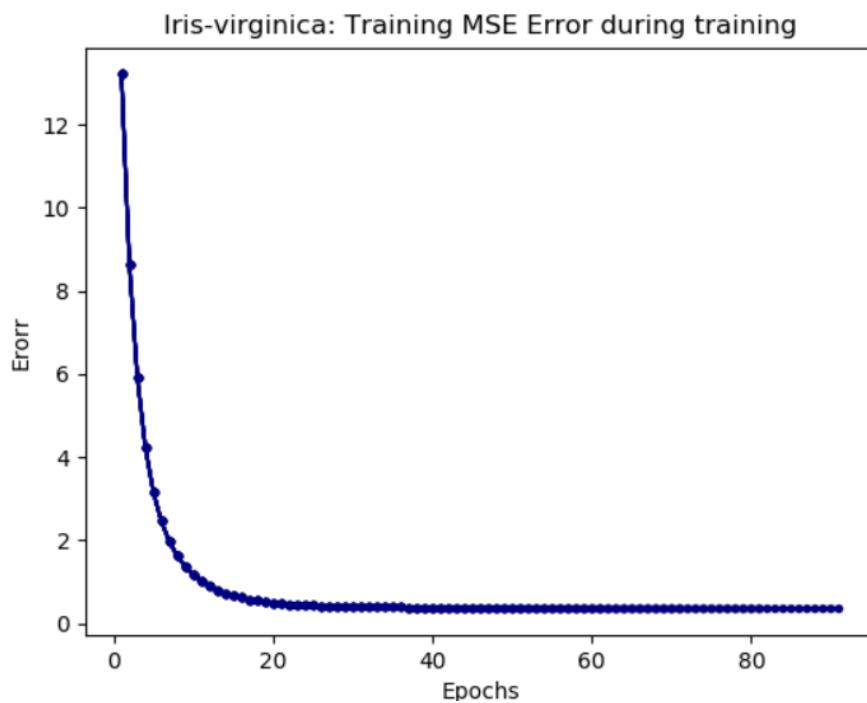
همین طور که مشاهده می شود در دو حالت وزن های اولیه متفاوت دو پرسپترون خطای آموزش و تست و k-fold متفاوتی دارند و اگر به نمودارهای تغییرات وزن ها و خطای مجموعه آموزش و ارزیابی نگاه کنیم متوجه می شویم آن ها نیز متفاوت در نتیجه تغییر وزن های اولیه بر نتایج تاثیر می گذارد. یکی از دلایل آن این است که در حالت کلی این الگوریتم یک مینیمم ندارد و معمولا بیش از یک مینیمم دارد.

خروجی فایل اول ورودی برای آدالاین یکی در برابر همه:

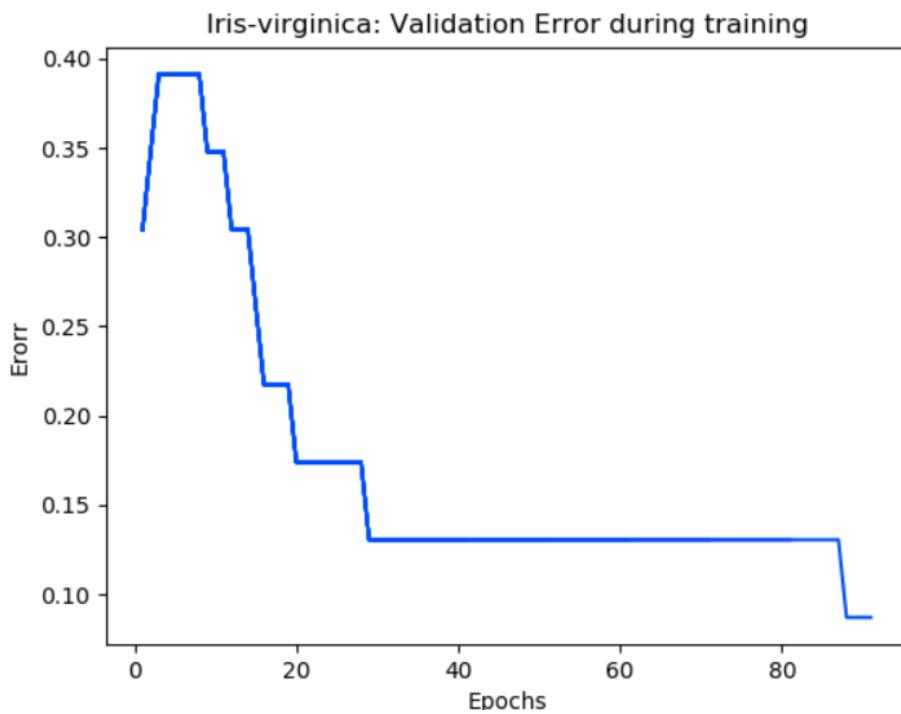
تغییر وزن ها در هر ایپاتک برای آدالاین کلاس Iris-virginica



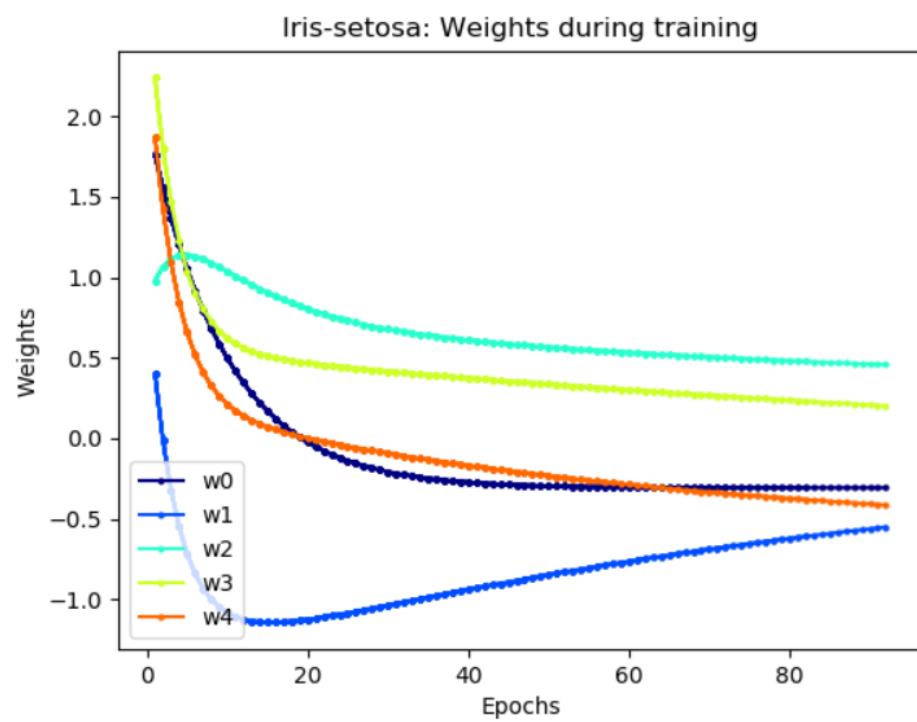
تغییر خطای مجموعه آموزش در هر ایپاک برای آدالین کلاس Iris-virginica



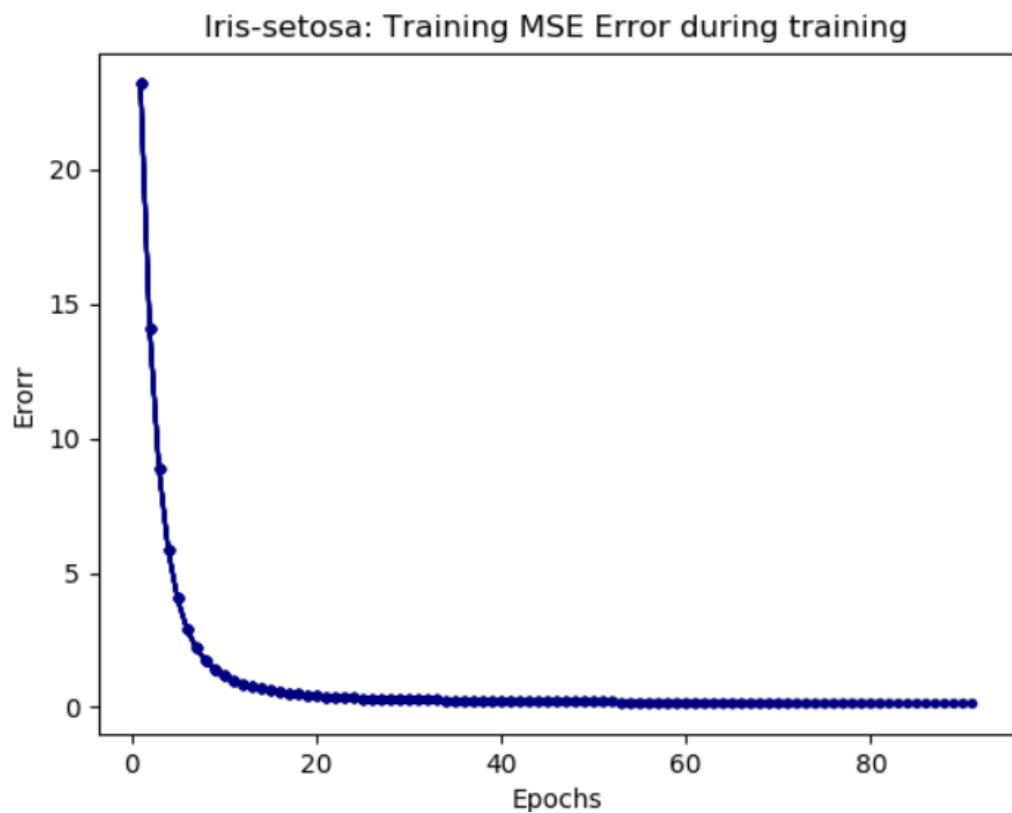
تغییر خطای مجموعه ارزیابی در هر ایپاک برای آدالین کلاس Iris-virginica



تغییر وزن‌ها در هر ایپاتک برای آدالین کلاس Iris-setosa

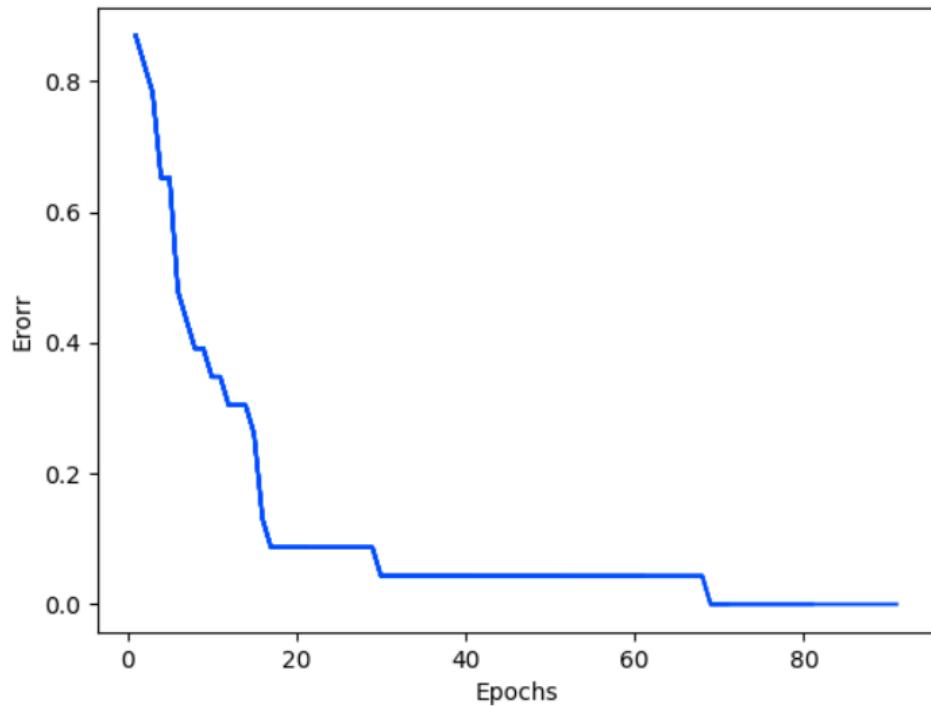


تغییر خطای مجموعه آموزش در هر اپاک برای آدالین کلاس Iris-setosa

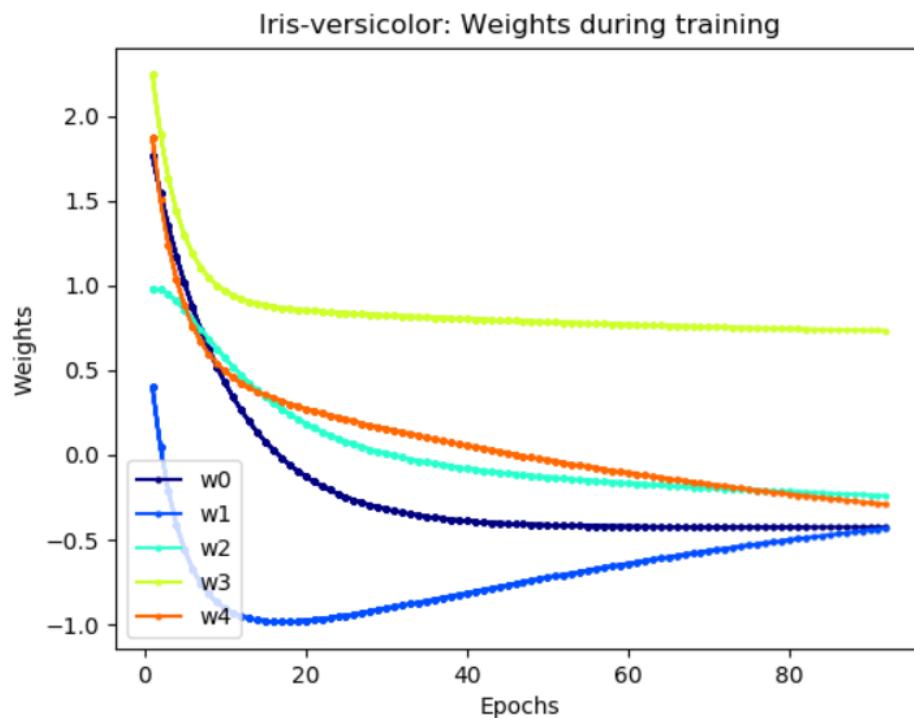


تغییر خطای مجموعه ارزیابی در هر اپاک برای آدالین کلاس Iris-setosa

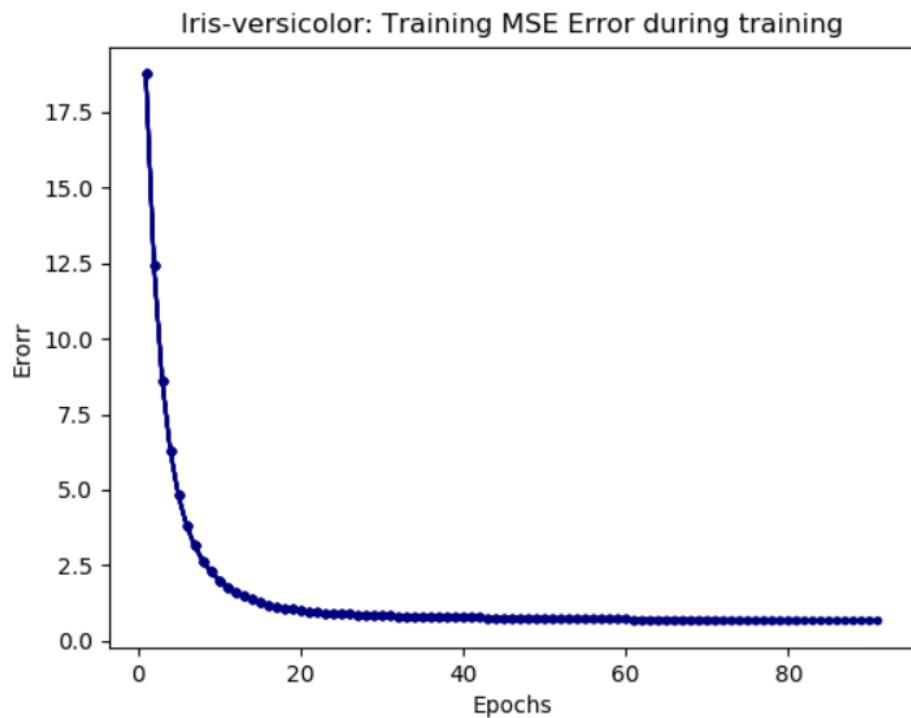
Iris-setosa: Validation Error during training



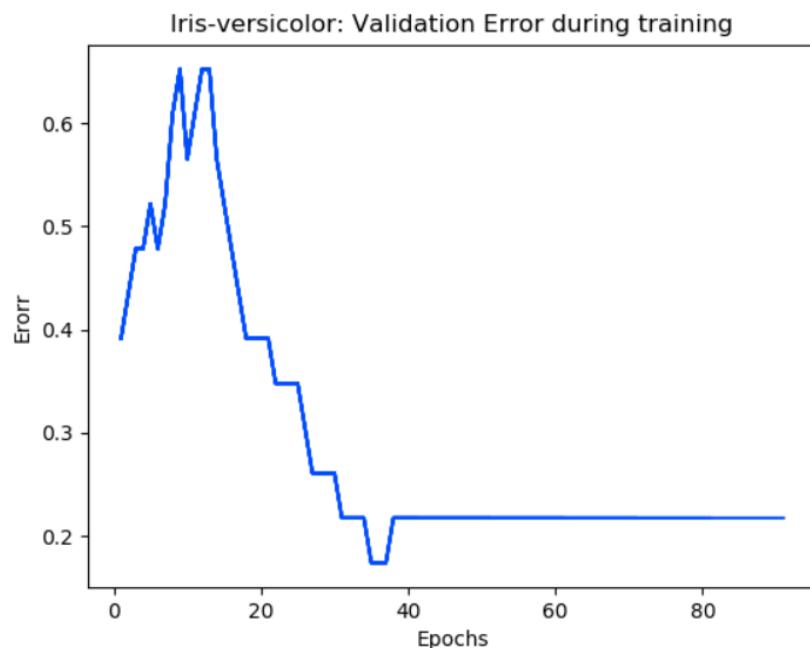
تغییر وزن‌ها در هر ایپاک برای آدالین کلاس Iris-versicolor



تغییر خطای مجموعه آموزش در هر ایپاک برای آدالین کلاس Iris-versicolor



تغییر خطای مجموعه ارزیابی در هر ایپاک برای آدالین کلاس Iris-versicolor

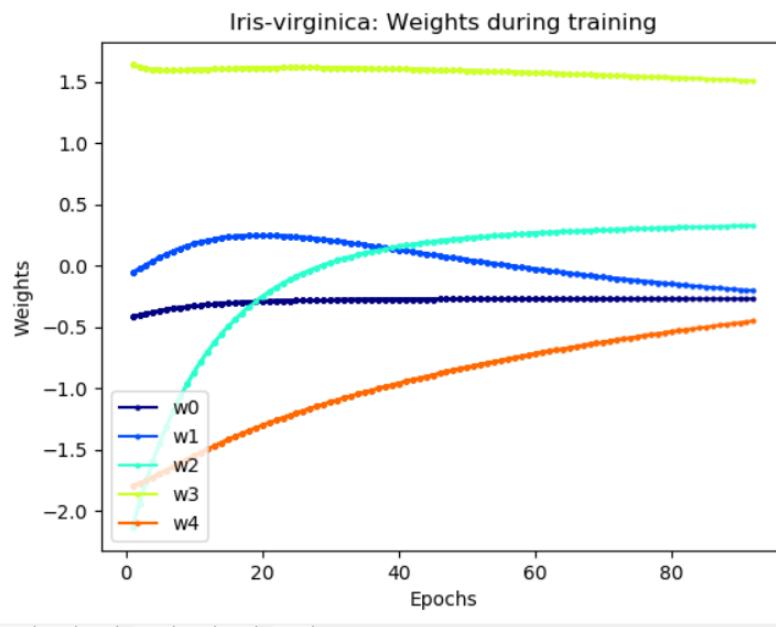


خطای k-fold و خطای عادی آموزش و خطای تست در پایان:

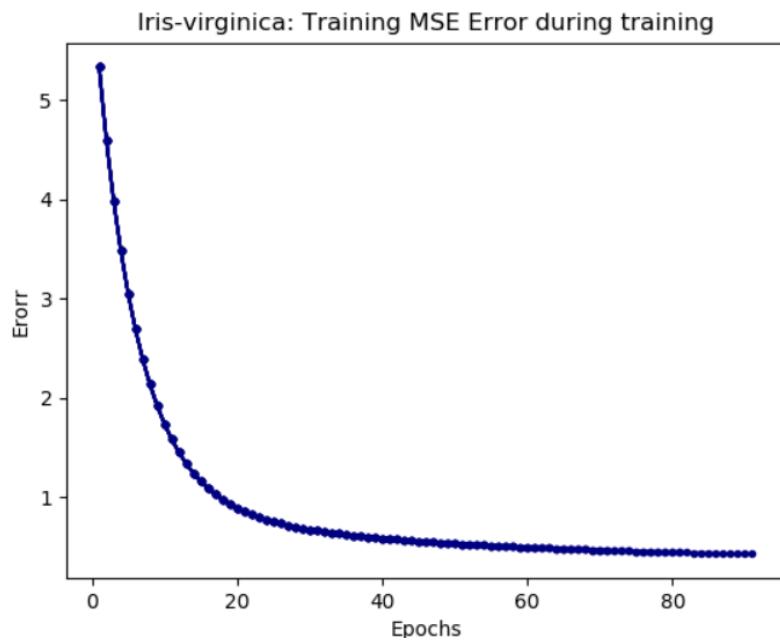
```
Calculating 10-Fold Cross Validation ...
10-fold-cross-validation error is: 0.18714285714285717
Train Error(Ordinary Error, not k-fold)= 0.16346153846153846
Test Error= 0.30434782608695654
```

خروجی فایل دوم ورودی برای آدالین یکی در برابر همه:

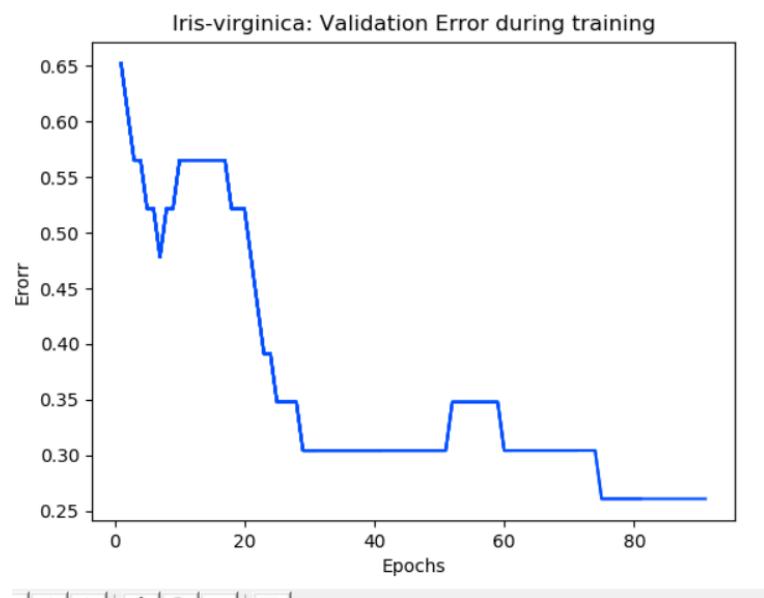
تغییر وزن‌ها در هر ایپاک برای آدالین کلاس Iris-virginica



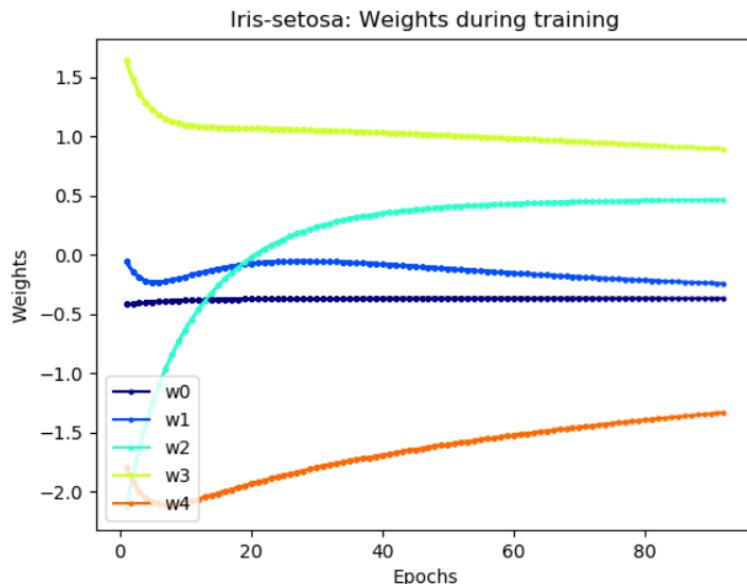
تغییر خطای مجموعه آموزش در هر ایپاک برای آدالین کلاس Iris-virginica



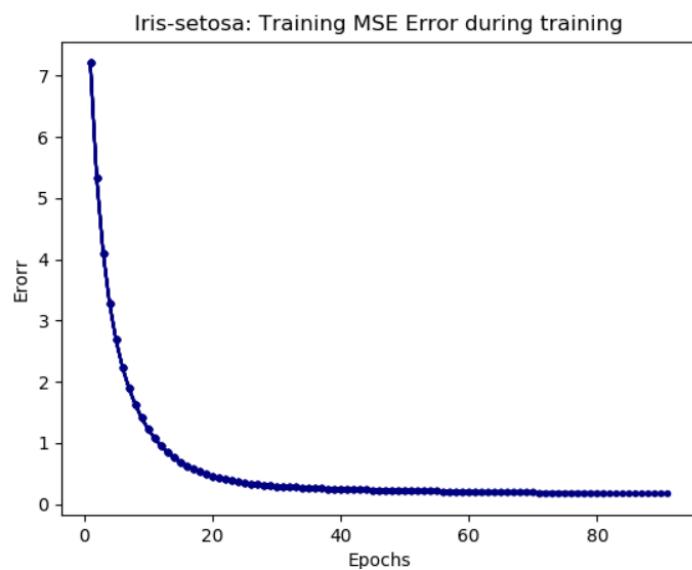
تغییر خطای مجموعه ارزیابی در هر ایپاک برای آدالین کلاس Iris-virginica



تغییر وزن‌ها در هر ایپاک برای آدالین کلاس Iris-setosa



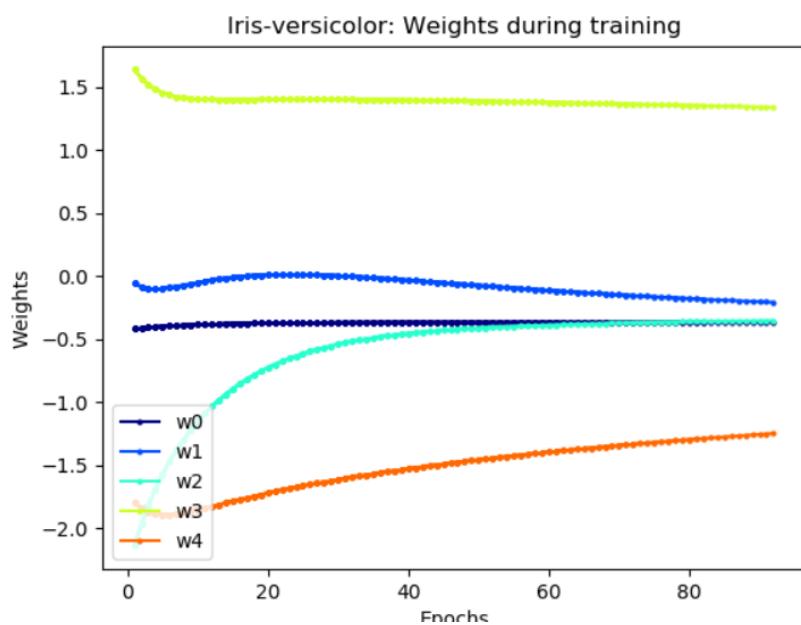
تغییر خطای مجموعه آموزش در هر ایپاک برای آدالین کلاس Iris-setosa



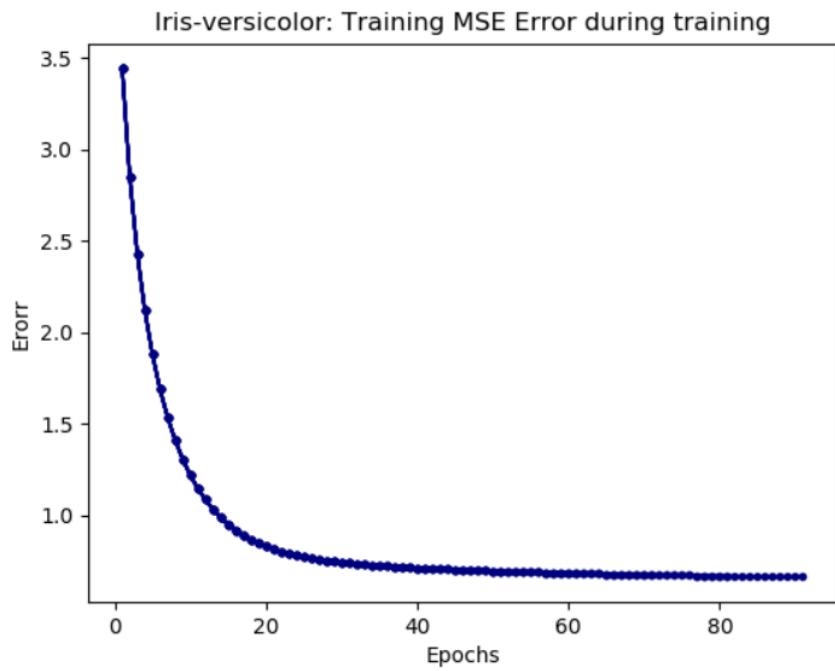
تغییر خطای مجموعه ارزیابی در هر ایپاک برای آدالین کلاس Iris-setosa



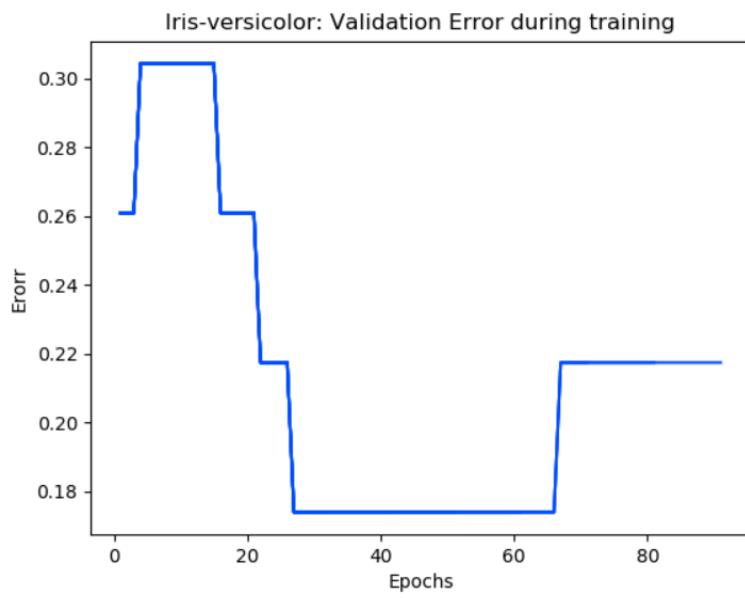
تغییر وزن‌ها در هر ایپاک برای آدالین کلاس Iris-versicolor



تغییر خطای مجموعه آموزش در هر ایپاک برای آدالین کلاس Iris-versicolor



تغییر خطای مجموعه ارزیابی در هر ایپاک برای آدالین کلاس Iris-versicolor



خطای k-fold و خطای عادی آموزش و خطای تست در پایان:

```

Calculating 10-Fold Cross Validation ...
10-fold-cross-validation error is: 0.15142857142857144
Train Error(Ordinary Error, not k-fold)= 0.1346153846153846
Test Error= 0.21739130434782608

```

در دو فراخوانی adaline one vs all با پارامترهای یکسان بجز وزن‌های اولیه‌ی یکسان را اگر مقایسه کیم می‌بینیم در خطای K-fold و مجموعه‌ی تست متفاوت اند همچنین نمودارهای خطای ارزیابی و آموزش و تغییر وزن‌ها هم متفاوت است به همین دلیل تغییر وزن‌های اولیه تاثیر گذار است. یکی از دلایل آن این است که آدالین از گرادیان نزولی استفاده می‌کند در مینیمم محلی گیر می‌کند و با تغییر وزن‌ها ممکن است در مینیمم‌های مختلفی افتاد.

در هر دو حالت پرسپترون و آدالین تغییر وزن‌های اولیه موجب تغییر نتایج شد.

استفاده از functional link neural network و پرسپترون چند کلاسیتابع `read_iris_dataset_FLNN` را قبل تر توضیح دادم که دیتاست را می‌خواند و ضرایب FLNN مناسب فیچرها را هم به آن اضافه می‌کرد. برای ایجاد `multi-class perceptron` کلاسی به اسم `multi_class_perceptron.py` در فایل `multi_class_perceptron` نوشته ام که بسیار شبیه به کلاس پرسپترون عادی است که نوشته‌ام و در ابتدا توضیح دادم ولی تفاوت‌های زیر را با پرسپترون عادی دارد:

این پرسپترون یک خروجی دارد که اگر  $n$  تا خروجی داشته باشیم وابسته به تعداد کلاس‌ها برای یک نمونه خروجی ای از ۱ تا  $n$  می‌دهد.

برای آپدیت این گره مانند پرسپترون نرمال عمل می‌کنیم

```
weights[1::] += learning_rate * (target - y_hat) * x
```

```
weights[0] += learning_rate * (target - y_hat)
```

با این تفاوت که `target` و ۰ نیست بلکه ۱ تا  $n$  است.

همین طور در روش پرسپترون عادی وقتی `net input` منفی می‌شد برچسبی که پرسپترون به نمونه اختصاص می‌داد برابر صفر بود و در غیر این صورت برابر با ۱ می‌بود ولی در اینجا با گرفتن یک نمونه `net input` را مانند گذشته محاسبه می‌کنیم و بعد مشخص می‌کنیم `net input` به کدام برچسب نزدیک تر است. مثلاً اگر ۳ کلاس

داشته باشیم و `net input` برابر با ۲,۶ شود، ۳ به ۲,۶ از ۲ و یک نزدیک تر است در نتیجه برچسب نمونه برابر با سه می‌شود.

در کد فایل `run_multi_class_perceptron.py` یک رابط کاربری متنی ایجاد کردہام که به سه روش پارامترها را از کاربر می‌گیرد:

- پارامترهای مورد نیاز برای خواندن دیتاست و ایجاد یک `multi-class perceptron` را به صورت متنی در ترمینال از کاربر می‌گیرد.
- پارامترها را که در شکل یک فایل `json` می‌گیرد
- وزن‌های یک `multi-class perceptron` را که در یک فایل `json` است را می‌گیرد.

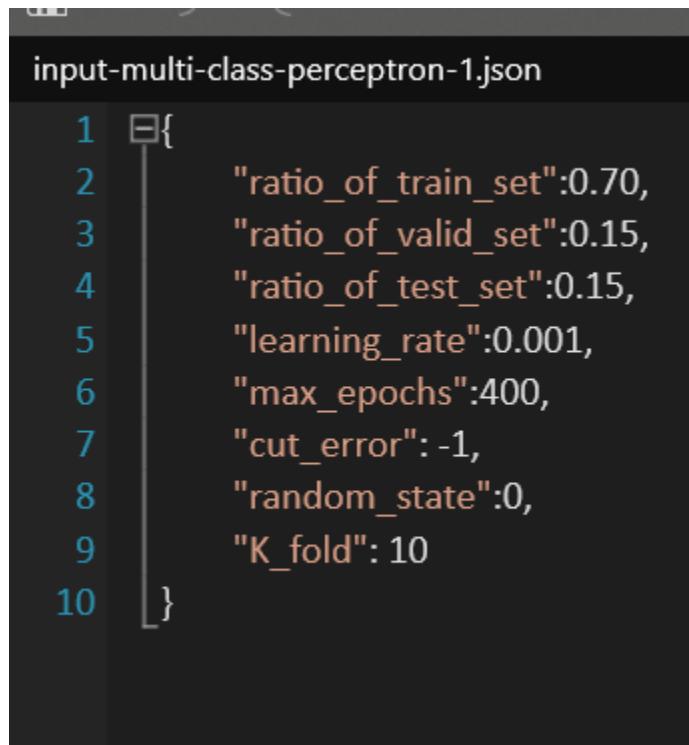
```
C:\Users\Home\AppData\Local\Programs\Python\Python36\python.exe "C:/Users/Home/Dropbox/codes/ANN/A  
How do you want to give learning parameters to the program?  
> 1: I give parameters via Standard Input / I have trained multi-class perceptrons in a file  
> 2: I give parameters via a file  
Input 1 or 2:
```

این پارامترها شامل موردهای زیر است:

- نسبتی از دیتاست که باید صرف مجموعه‌ی آموزشی شود
- نسبتی از دیتاست که باید صرف مجموعه‌ی ارزیابی شود
- نسبتی از دیتاست که باید صرف مجموعه‌ی تست شود
- درجه‌ی یک بودن فیچرها یا فیچرها حاوی فیچرهای درجه دو تولید شده هم باشند
- حداکثر تعداد ایپوکها
- ضریب یادگیری
- نحوه‌ی پایان آموزش
- `Seed` برای تولید اعداد تصادفی و تکرار پذیری مسیله
- تعداد `fold` ها برای `k-fold`

در صورتی که از یکی از دو حالت اول استفاده کرده باشیم، دیتاست را می‌خواند و آن را با روش `standardization` اسکیل می‌کند و سپس `multi-class perceptron` را آموزش می‌دهد و در حین آموزش خطای تست و ارزیابی و وزن‌های هر پرسپترون را نمایش می‌دهد و در آخر خطای دقت مجموعه‌ی آموزش و ارزیابی و `K-fold` و تست را گزارش می‌کند. در صورتی که از روش سوم استفاده کنیم از آنجایی که فاز آموزش نداریم و وزن‌های یادگرفته شده را بارگذاری می‌کنیم فقط خطای مجموعه‌ی آموزش و تست گزارش می‌شود.

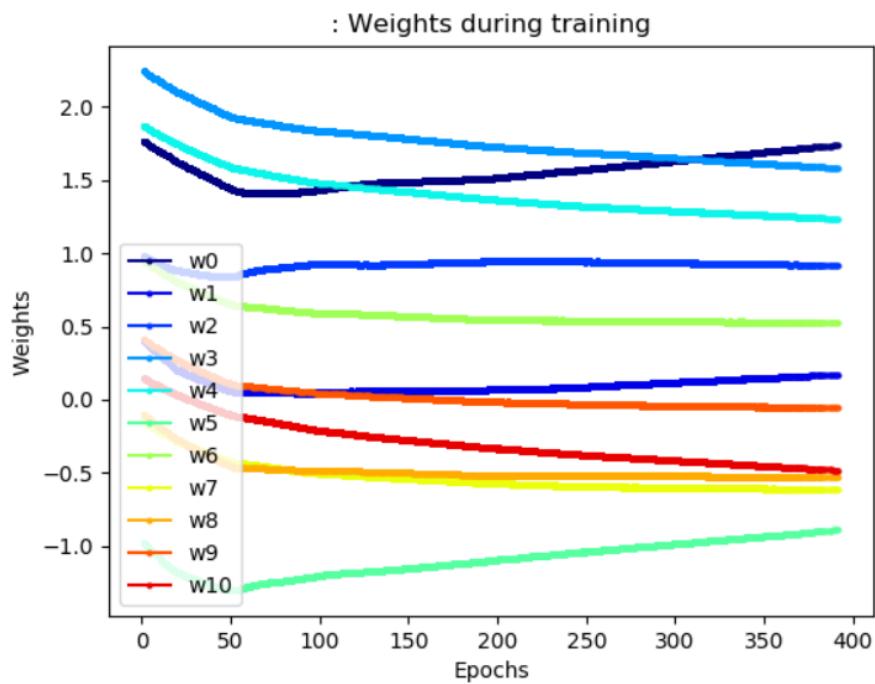
در ادامه یک نمونه از فراخوانی‌های کلاس یک در مقابل همه را با استفاده از کد و رابط کاربری موجود در run\_perceptrons\_one\_vs\_all.py را ارایه می‌کنم.



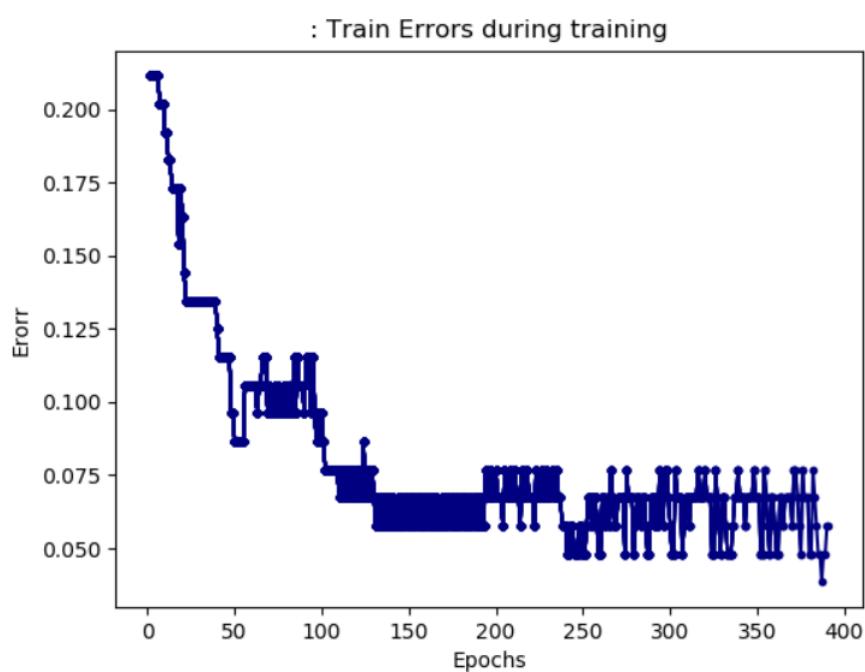
```
input-multi-class-perceptron-1.json
1  {
2      "ratio_of_train_set":0.70,
3      "ratio_of_valid_set":0.15,
4      "ratio_of_test_set":0.15,
5      "learning_rate":0.001,
6      "max_epochs":400,
7      "cut_error": -1,
8      "random_state":0,
9      "K_fold": 10
10 }
```

ضریب یادگیری برابر با  $1 \times 10^{-4}$  است، حداکثر تعداد اپوکها برابر با ۴۰۰ است، برای پایان آموزش فقط رسیدن به حداکثر تعداد ایپوکها چک می‌شود، برای K-Fold تعداد ۱۰ fold در نظر گرفته می‌شود، از فیچرها با اضافه کردن فیچرهای مرتبه ۲ استفاده می‌شود، ۷۰ درصد داده‌ها به آموزش ۱۵ درصد به ارزیابی و ۱۵ درصد به تست اختصاص پیدا می‌کند. از seed ۰ برای تولید وزن‌های تصادفی در فایل پaramترهای استفاده می‌کنیم.

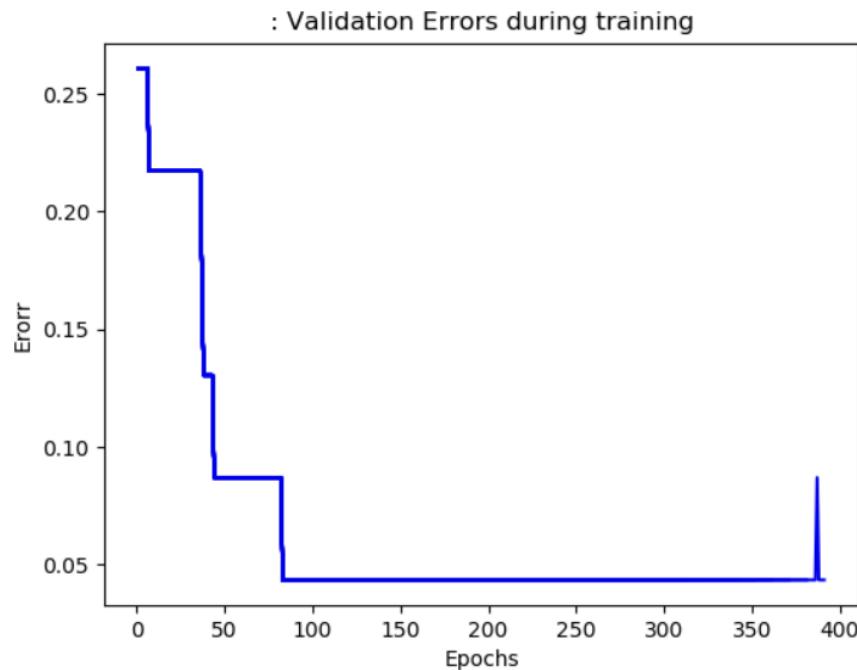
وزن‌های فیچرها(فیچرهای عادی و FLNN) در هر ایپاک



خطای مجموعه آموزش در هر ایپاک



خطای مجموعه ارزیابی در هر ایپاک



خطای k-fold و مجموعه تست

```
... running with labels found to put in reverse.
Calculating 10-Fold Cross Validation ...
Train Error(Ordinary Error, not k-fold)= 0.038461538461538464
Test Error= 0.043478260869565216
```

F استفاده از یکی از ابزارهای موجود [sklearn perceptron](#) در فایل `best_sklearn_perceptron.py` به ازای ضرایب یادگیری(`eta0`) مختلف و `random_state`های مختلف که منجر به تولید وزن‌های اولیه متفاوت می‌شوند با حداقل تعداد ۲۰۰ ایپاک پرسپترونی چند کلاسه آموزش داده‌ام. دیتاست را به ۷۰ درصد آموزش و ۱۵ درصد ارزیابی و ۱۵ درصد تست تقسیم کرده‌ام. خروجی‌های آن در فایل `find_good_sklearn_perceptrons.txt` قرار دارد که تصویر بخشی از آن را در زیر می‌بینید:

```
VV=====VVV=====VV  
learning_rate: 0.01  
random state: 0  
ratio of train set: 0.7  
10-fold-error error: 0.11545454545454545  
Train Error(Ordinary Error, not k-fold)= 0.11538461538461539  
Test Error= 0.08695652173913043  
=====^  
  
VV=====VVV=====VV  
learning_rate: 0.11  
random state: 0  
ratio of train set: 0.7  
10-fold-error error: 0.14545454545454545  
Train Error(Ordinary Error, not k-fold)= 0.14423076923076922  
Test Error= 0.13043478260869565  
=====^  
  
VV=====VVV=====VV  
learning_rate: 0.21000000000000002  
random state: 0  
ratio of train set: 0.7  
10-fold-error error: 0.14545454545454545  
Train Error(Ordinary Error, not k-fold)= 0.14423076923076922  
Test Error= 0.13043478260869565  
=====^  
  
VV=====VVV=====VV  
learning_rate: 0.31000000000000005  
random state: 0  
ratio of train set: 0.7  
10-fold-error error: 0.14545454545454545  
Train Error(Ordinary Error, not k-fold)= 0.14423076923076922  
Test Error= 0.13043478260869565  
=====^  
  
VV=====VVV=====VV  
learning_rate: 0.41000000000000003  
random state: 0  
ratio of train set: 0.7  
10-fold-error error: 0.14545454545454545  
Train Error(Ordinary Error, not k-fold)= 0.14423076923076922
```

---

```

VV=====VVV=====VV
learning_rate: 0.7100000000000001
random state: 9
ratio of train set: 0.7
10-fold-error error: 0.2718181818181818
Train Error(Ordinary Error, not k-fold)= 0.2692307692307692
Test Error= 0.17391304347826086
=====^

VV=====VVV=====VV
learning_rate: 0.81
random state: 9
ratio of train set: 0.7
10-fold-error error: 0.2718181818181818
Train Error(Ordinary Error, not k-fold)= 0.2692307692307692
Test Error= 0.17391304347826086
=====^

VV=====VVV=====VV
learning_rate: 0.91
random state: 9
ratio of train set: 0.7
10-fold-error error: 0.2718181818181818
Train Error(Ordinary Error, not k-fold)= 0.2692307692307692
Test Error= 0.17391304347826086
=====^

*****
min test error: 0.08695652173913043
10-fold-error error of best test error: 0.2718181818181818
train error of best test error: 0.11538461538461539
best learning rate: 0.01
best random state: 0
best weights: [[ 0.018  0.054 -0.078 -0.035]
 [ 0.045 -0.53   0.323 -0.688]
 [-0.766 -0.58   1.026  1.274]]

```

Sklearn یک کتابخانه رایگان یادگیری ماشین برای زبان پایتون است. که الگوریتم‌های زیادی از دسته‌بندی، خوشه‌بندی و رگرسیون در آن موجود است. همین طور که در نتایج ارایه شده در قسمت‌های قبل مشاهده می‌شود پرسپترون‌هایی که نوشته ام معمولاً دقیق بین ۹۰ تا ۱۰۰ دارند که با نتایج این ابزار مشابهت زیادی دارد.

## بخش دوم - رگرسیون

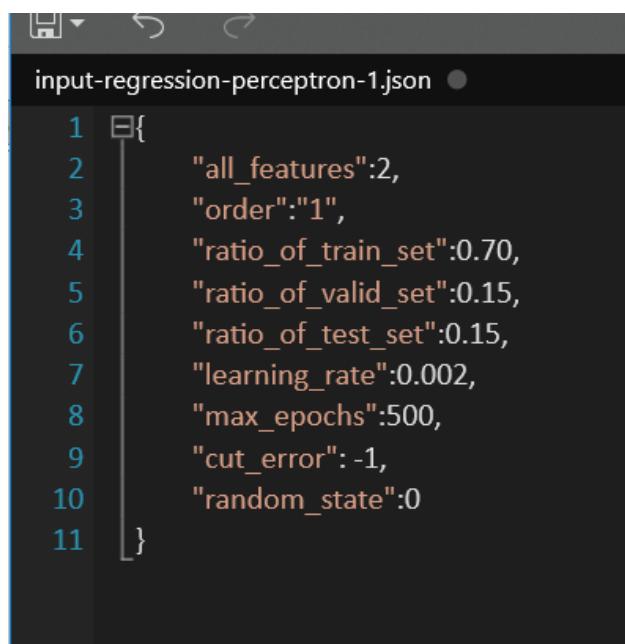
کد های درون `read_auto_mpg_dataset.py` امکان خواندن دیتاست را می دهد و با استفاده از پارامترهای مناسب می توان فیچرهای `categorical` را خواند یا نخواند و یا ترم های درجه دو را تولید کرد و یا آن ها را تولید نکرد. از آنجایی که فیچرهای `categorical` دیتاست از نوع `nomial` هستند یعنی ترتیب دارند مانند ۳ سیلندر ۶ سیلندر و ... آن ها را تغییر ندادم.

در `multi_perceptron_regression.py` یک کلاس برای انجام رگرسیون با پرسپترون است و کاملا شبیه به `class perceptron` در بخش یک است که آن را توضیح دادم با این تفاوت که خروجی نهایی پرسپترون با `net input` آن برابر است.

کدهای درون `run_perceptron_regression.py` هم واسطه کاربری مناسب را مانند سوال های قبل فراهم می کند.

A تخمین MPG با استفاده از ویژگی های پیوسته:

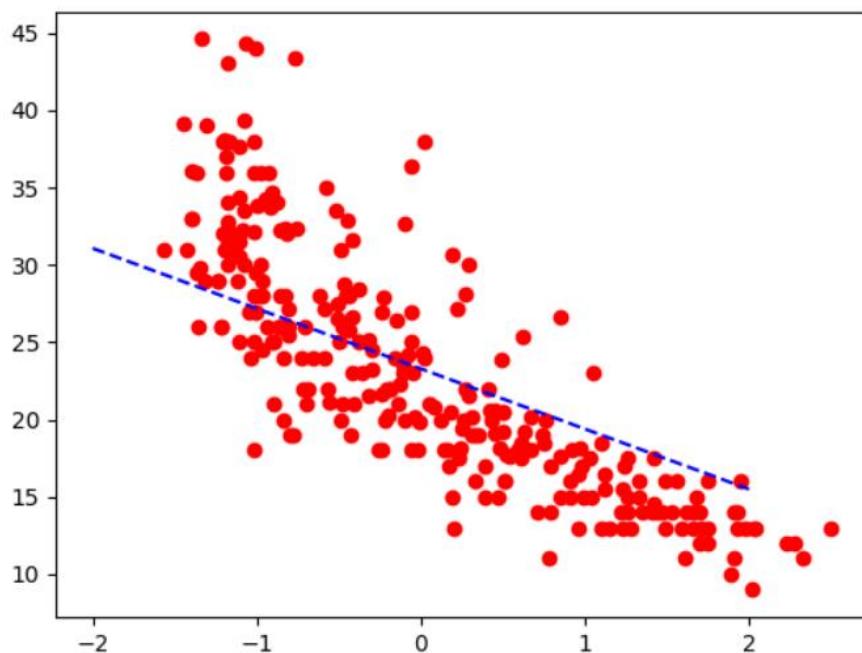
فایل ورودی برای انجام رگرسیون



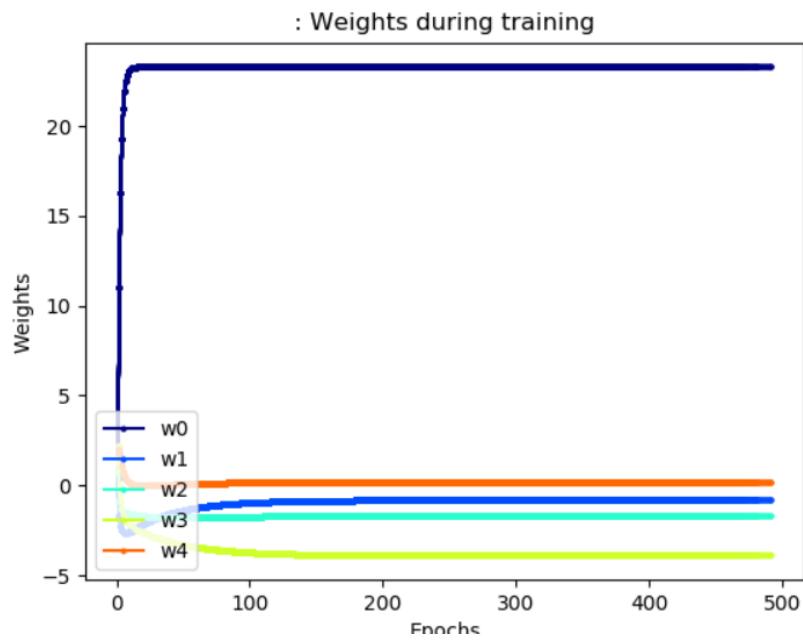
```
1 {  
2     "all_features":2,  
3     "order":"1",  
4     "ratio_of_train_set":0.70,  
5     "ratio_of_valid_set":0.15,  
6     "ratio_of_test_set":0.15,  
7     "learning_rate":0.002,  
8     "max_epochs":500,  
9     "cut_error": -1,  
10    "random_state":0  
11 }
```

از ویژگی‌های پیوسته استفاده کن، از فیچرهای مرتبه اول استفاده کن، ۷۰ درصد دیتاست برای آموزش است ۱۵ درصد برای تست و ۱۵ درصد دیگر برای ارزیابی، ضریب یادگیری برابر است با  $0.002$ ، حداکثر  $500$  ایپاک انجام بده، از `seed = 0` استفاده کن.

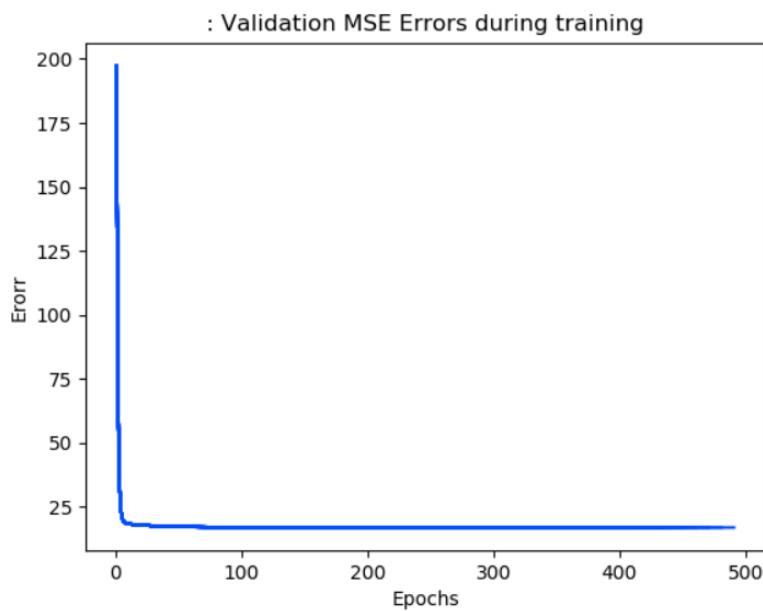
محود عمودی MPG، محور افقی وزن ماشین(اسکیل شده) خط چین آبی خط تخمین mpg :



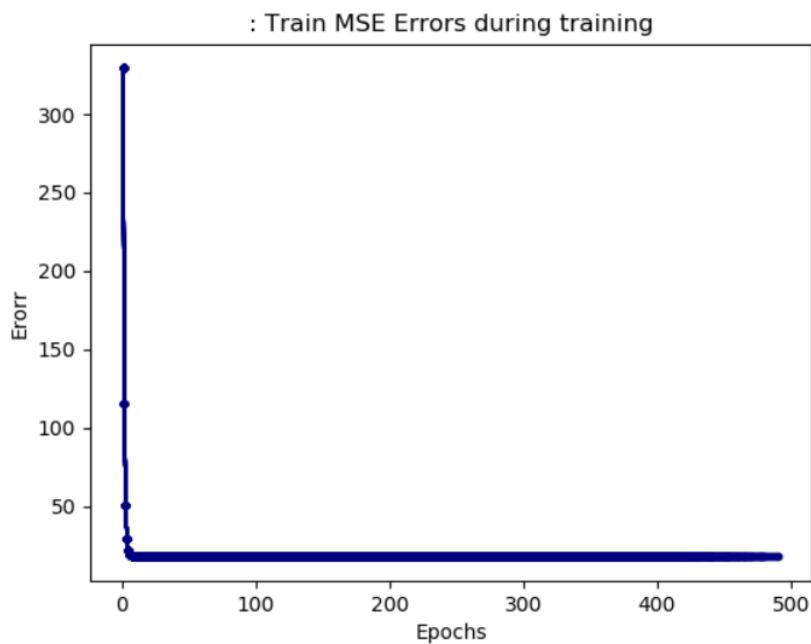
وزن ها در هر ایپاک



خطای MSE در هر ایپاک برای مجموعه‌ی ارزیابی.



خطای MSE در هر ایپاک برای مجموعه‌ی آموزش.

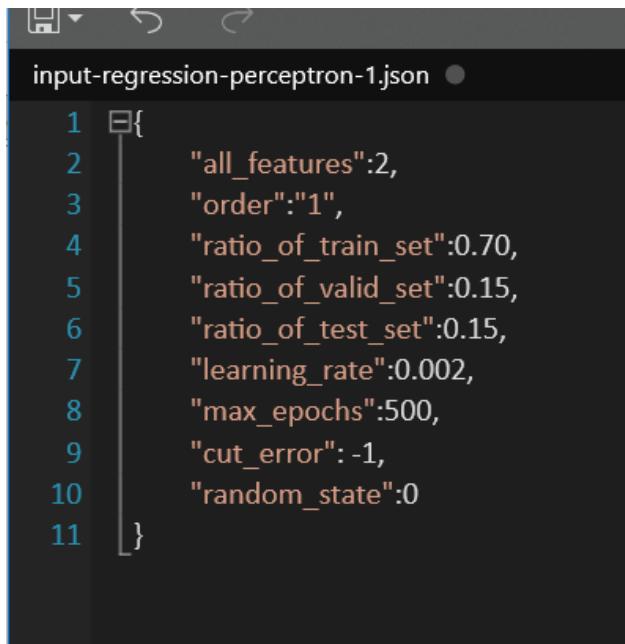


خطای MSE مجموعه آموزش، ارزیابی و تست:

```
Train MSE Error= [17.85422113]
Validation MSE-Error= [16.90649667]
Test MSE-Error= [21.90047505]
```

B تخمین MPG با استفاده از همهی ویژگی ها:

فایل ورودی برای انجام رگرسیون



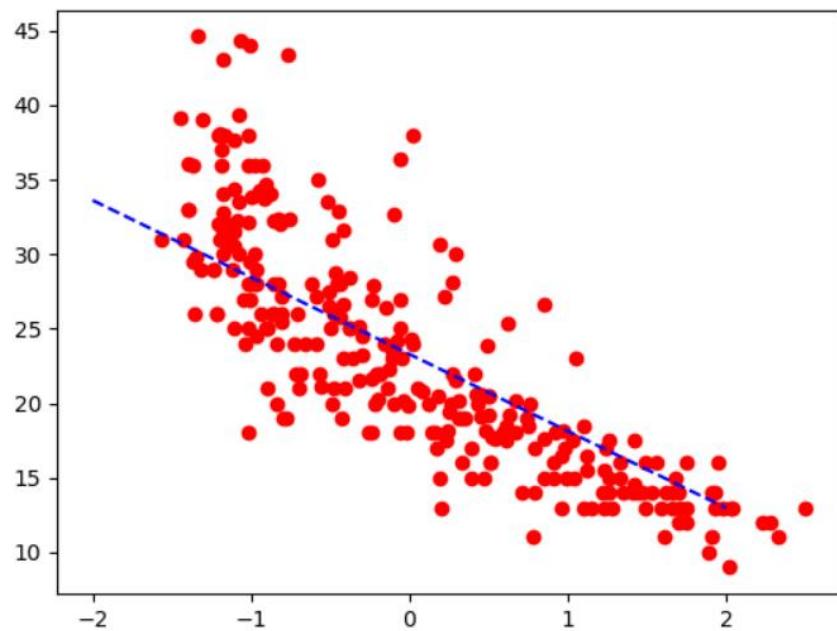
```
input-regression-perceptron-1.json
1 {
2     "all_features":2,
3     "order":"1",
4     "ratio_of_train_set":0.70,
5     "ratio_of_valid_set":0.15,
6     "ratio_of_test_set":0.15,
7     "learning_rate":0.002,
8     "max_epochs":500,
9     "cut_error": -1,
10    "random_state":0
11 }
```

از کلیه‌ی ویژگی‌ها استفاده کن، از فیچرهای مرتبه اول استفاده کن، ۷۰ درصد دیتاست برای آموزش است ۱۵ درصد برای تست و ۱۵ درصد دیگر برای ارزیابی، ضریب یادگیری برابر است با ۰،۰۰۲، حداقل ۵۰۰ ایپاک انجام بدء، از seed استفاده کن.

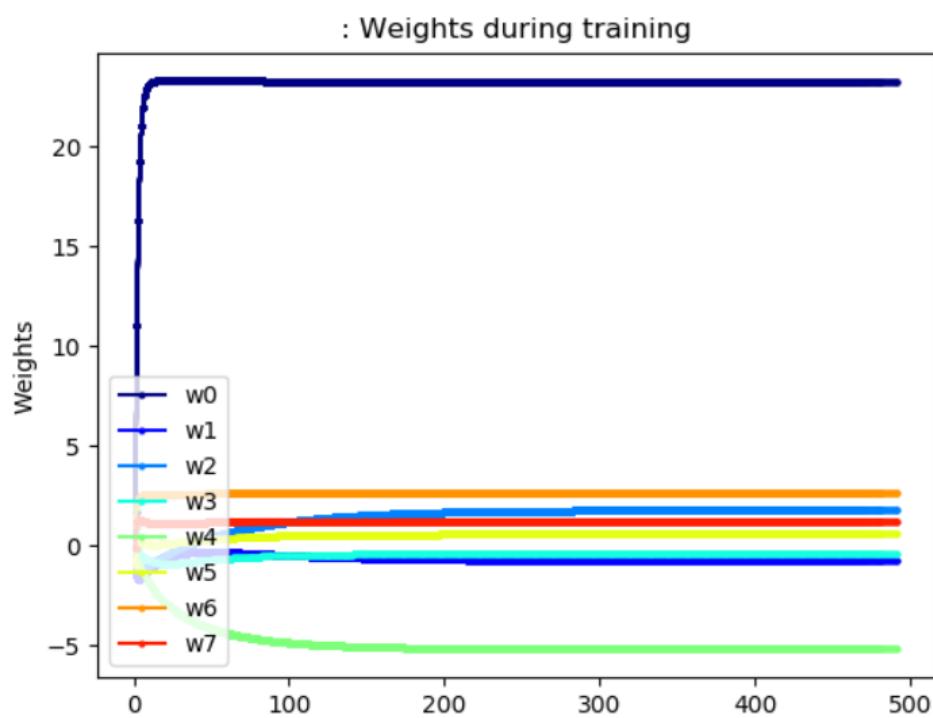
از آن جایی که این داده‌ها از نوع nominal categorical features هستند می‌توان آن‌ها را تغییر نداد و همان گونه که هستند استفاده کرد.

Machine Learning mit Python: das Praxis-Handbuch für Data Science, Predictive ...  
Book by Sebastian Raschka

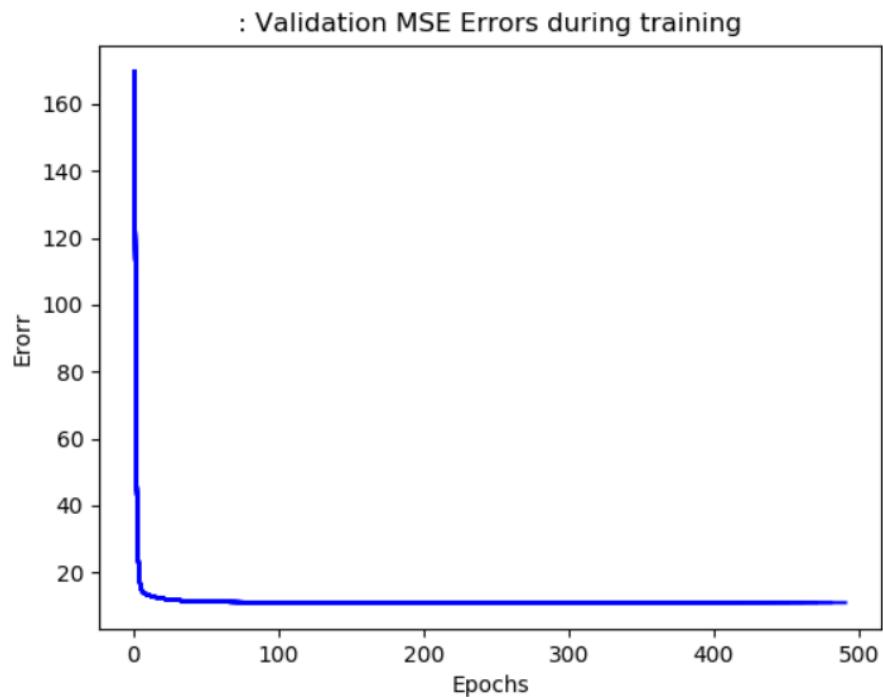
محود عمودی MPG، محور افقی وزن ماشین(اسکیل شده) خط چین آبی خط تخمین mpg :



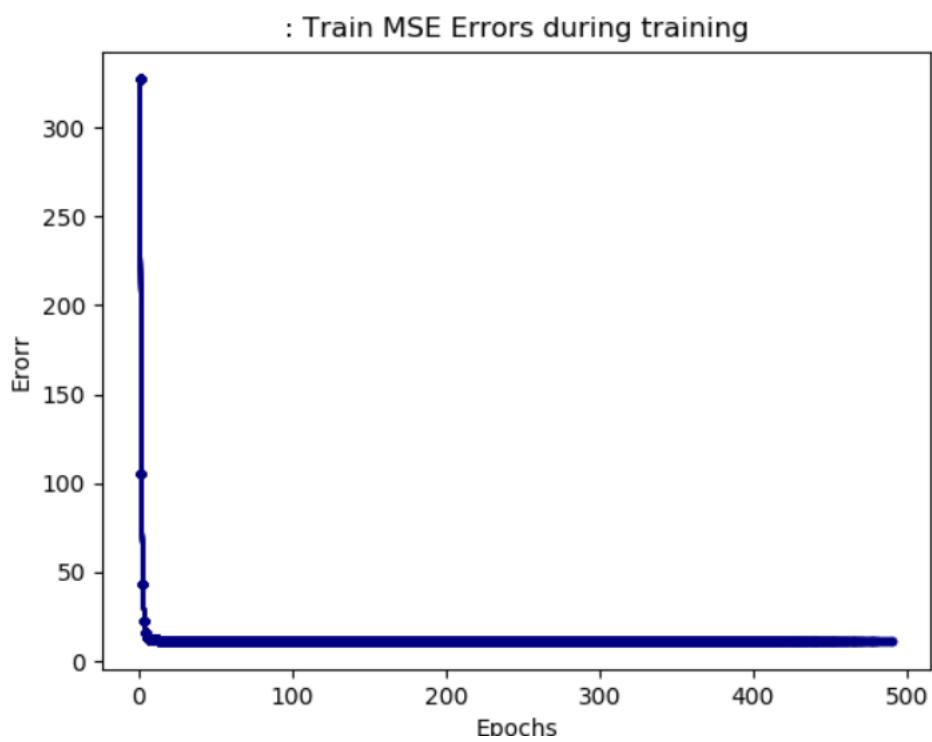
وزن ها در هر ایپاک



خطای MSE در هر ایپاک برای مجموعه ای ارزیابی.



خطای MSE در هر ایپاک برای مجموعه‌ی آموزش.



خطای MSE مجموعه آموزش، ارزیابی و تست:

```
Train MSE Error= [11.00677864]
Validation MSE-Error= [10.94279715]
Test MSE-Error= [12.59012426]
```

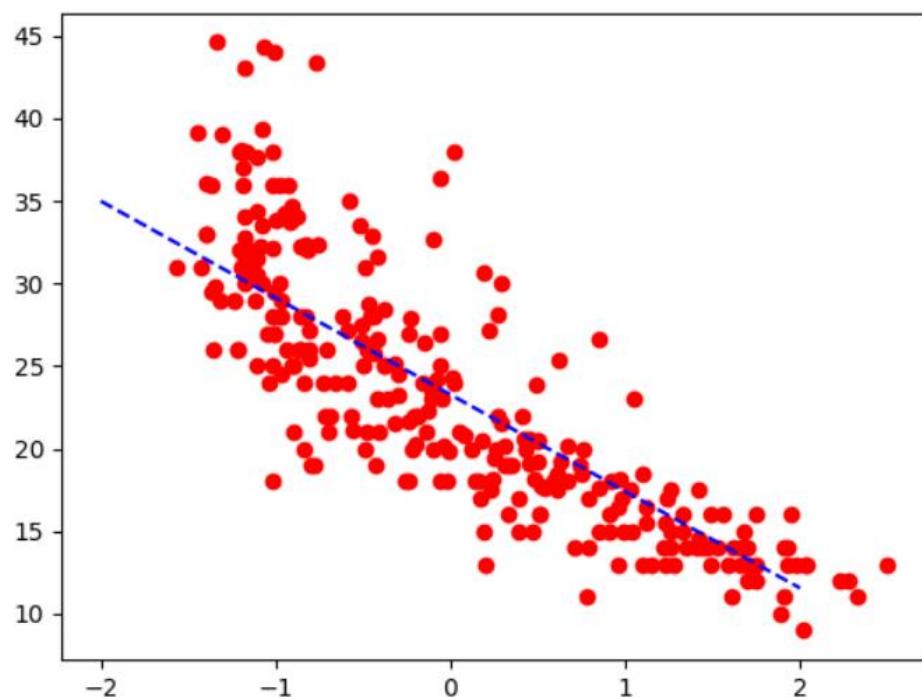
C تخمین MPG با استفاده از همهی ویژگی‌ها و همین طور term‌های درجه دو:

فایل ورودی برای انجام رگرسیون به صورت درجه دو

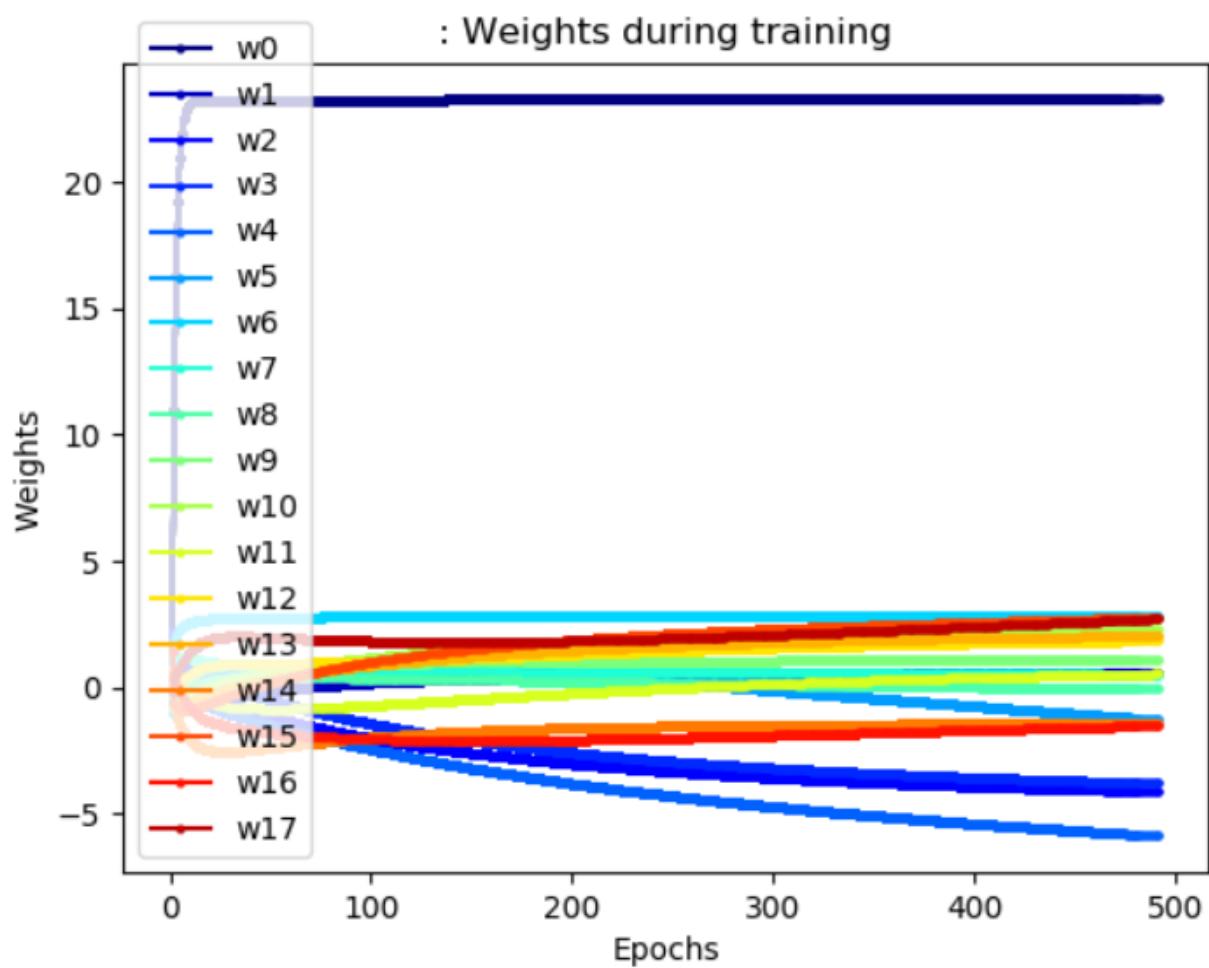
```
input-regression-perceptron-3.json
1  {
2    "all_features":1,
3    "order":"2",
4    "ratio_of_train_set":0.70,
5    "ratio_of_valid_set":0.15,
6    "ratio_of_test_set":0.15,
7    "learning_rate":0.002,
8    "max_epochs":500,
9    "cut_error": -1,
10   "random_state":0
11 }
```

از کلیهی ویژگی‌ها استفاده کن، term‌های درجه دو آن‌ها به فیچرهای اضافه کن، از فیچرهای مرتبه دوم استفاده کن، ۷۰ درصد دیتاست برای آموزش است ۱۵ درصد برای تست و ۱۵ درصد دیگر برای ارزیابی، ضریب یادگیری برابر است با ۰.۰۲، حداکثر ۵۰۰ ایپاک انجام بده، از seed استفاده کن.

محود عمودی MPG، محور افقی وزن ماشین(اسکیل شده) خط چین آبی خط تخمین mpg :

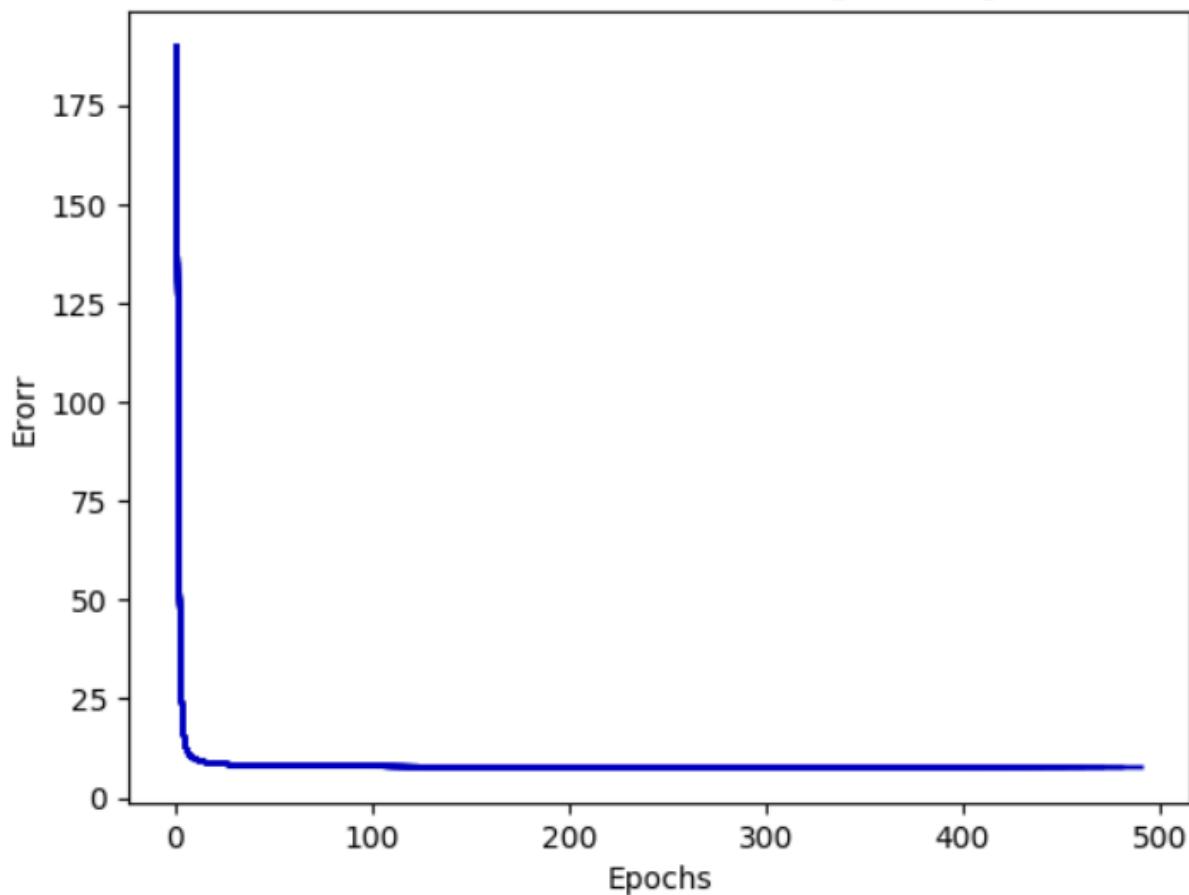


وزن ها در هر ایپاک

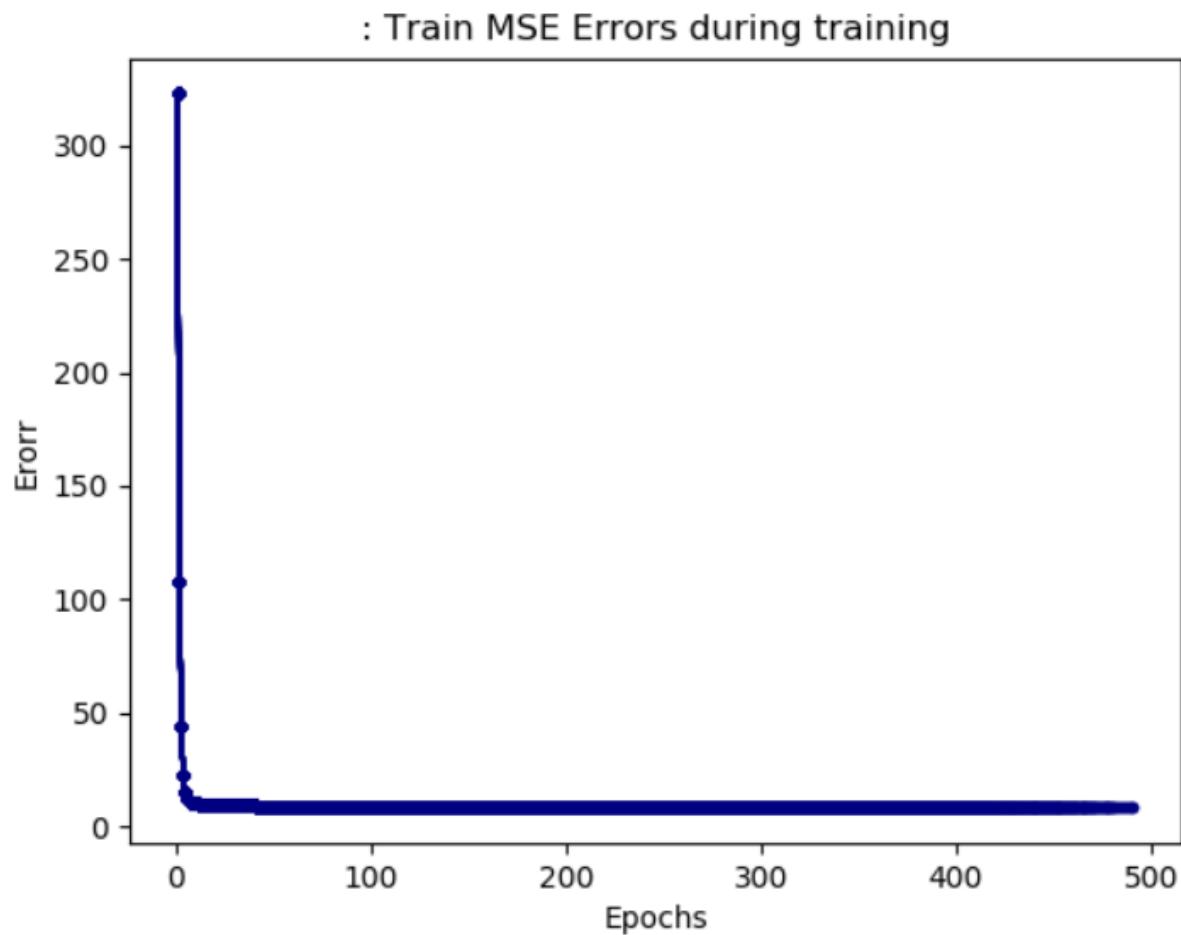


خطای MSE در هر ایپاک برای مجموعه‌ی ارزیابی.

: Validation MSE Errors during training



خطای MSE در هر ایپاک برای مجموعه‌ی آموزش.



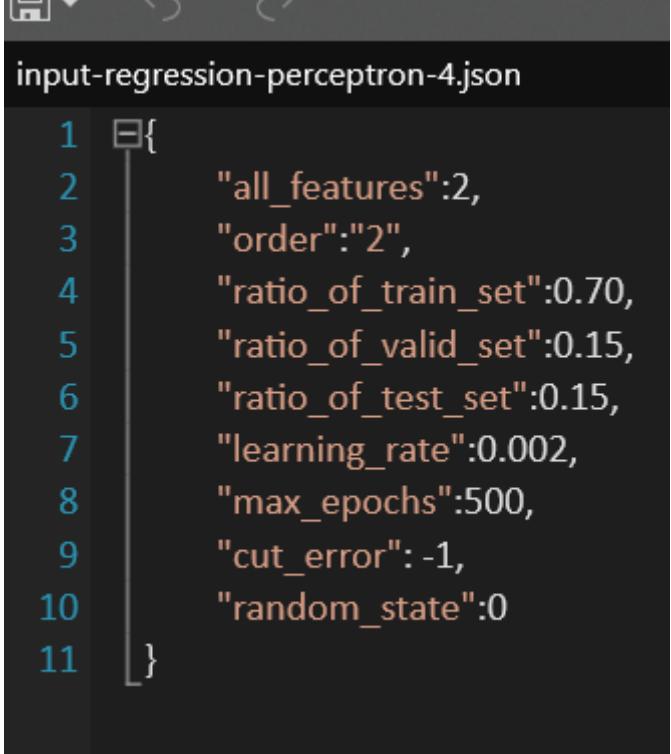
خطای MSE مجموعه آموزش، ارزیابی و تست:

```

Train MSE Error= [8.1233234]
Validation MSE-Error= [7.68683804]
Test MSE-Error= [10.72830739]
```

همین طور که مشاهده می شود در این جالت خطای MSE آموزش، ارزیابی و تست از دو حالت قبلی کمتر است.

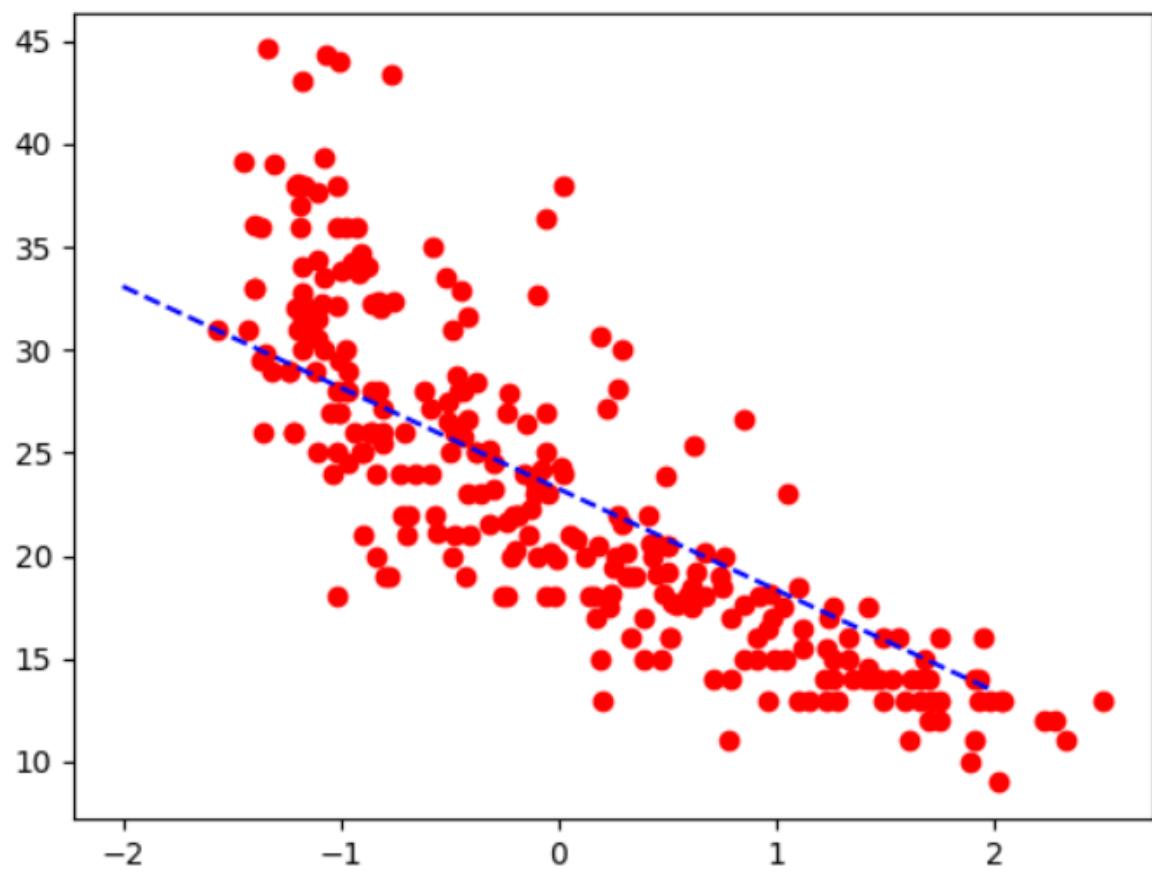
D تخمین MPG با استفاده از ویژگی‌های پیوسته و همین طور term‌های درجه دو: فایل ورودی برای انجام رگرسیون به صورت درجه دو



```
input-regression-perceptron-4.json
1  {
2    "all_features":2,
3    "order":"2",
4    "ratio_of_train_set":0.70,
5    "ratio_of_valid_set":0.15,
6    "ratio_of_test_set":0.15,
7    "learning_rate":0.002,
8    "max_epochs":500,
9    "cut_error": -1,
10   "random_state":0
11 }
```

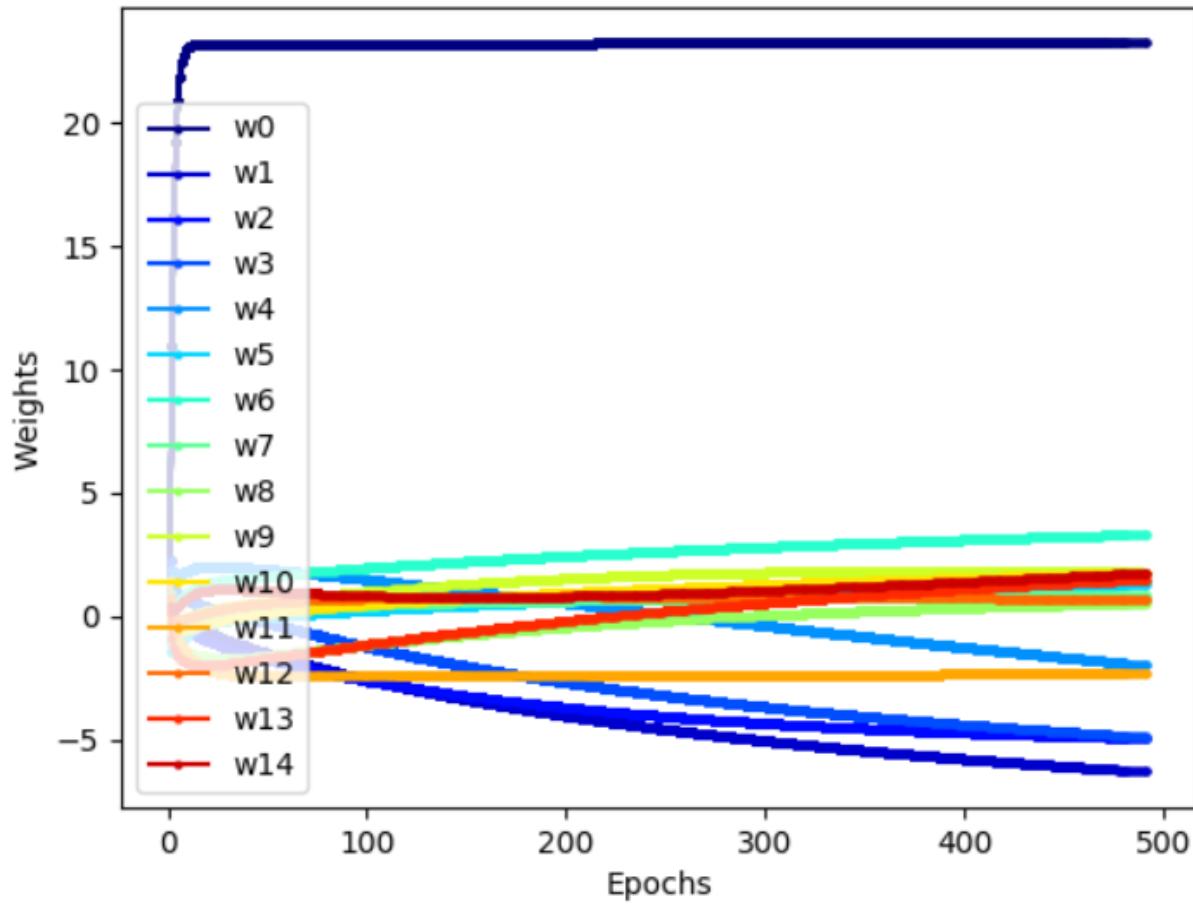
از کلیه‌ی ویژگی‌ها پیوسته استفاده کن، ترم‌های درجه دو آن‌ها به فیچرهای اضافه کن، از فیچرهای مرتبه دوم استفاده کن، ۷۰ درصد دیتاست برای آموزش است ۱۵ درصد برای تست و ۱۵ درصد دیگر برای ارزیابی، ضریب یادگیری برابر است با ۰.۰۲، حداقل ۵۰۰ ایپاک انجام بده، از seed استفاده کن.

محود عمودی MPG، محور افقی وزن ماشین(اسکیل شده) خط چین آبی خط تخمین mpg :

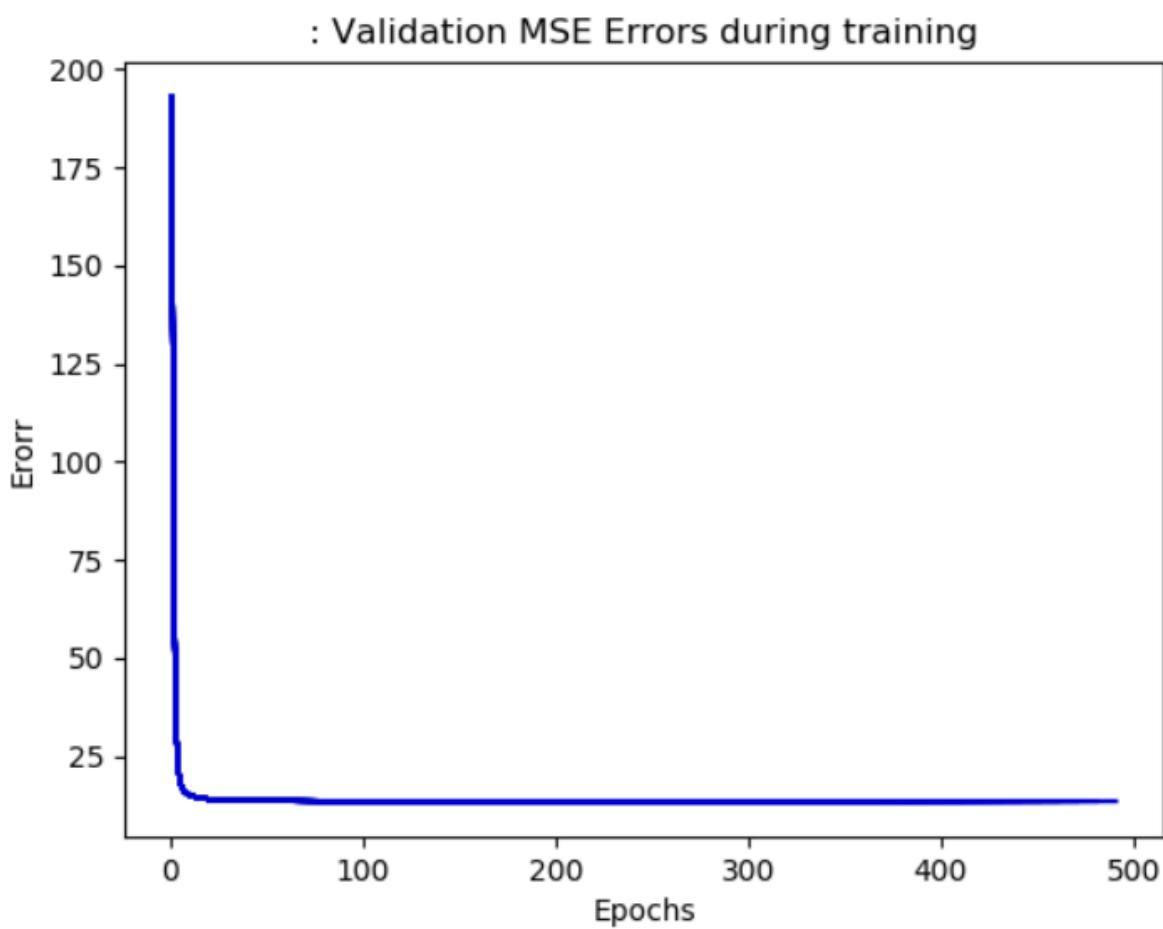


وزن ها در هر ایپاک

: Weights during training

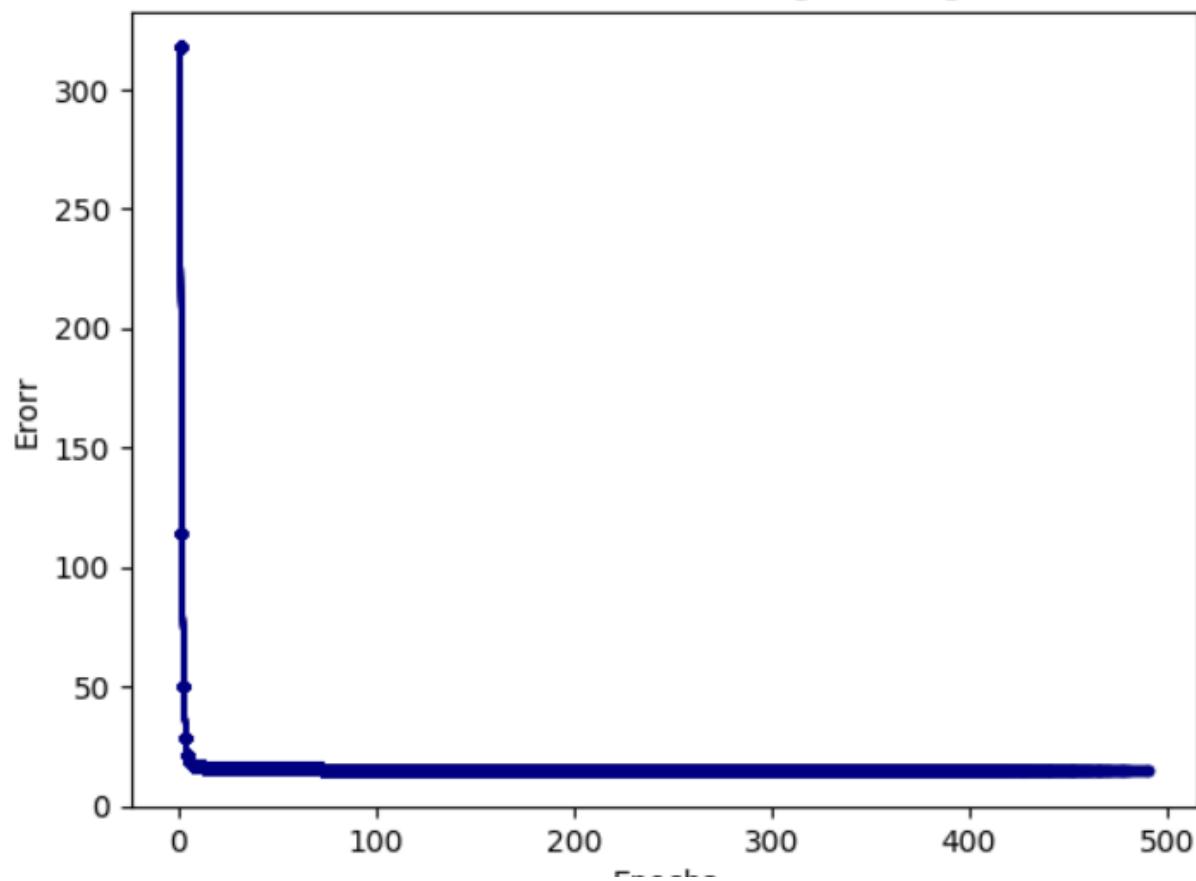


خطای MSE در هر ایپاک برای مجموعه‌ی ارزیابی.



خطای MSE در هر ایپاک برای مجموعه‌ی آموزش.

: Train MSE Errors during training



خطای MSE مجموعه آموزش، ارزیابی و تست:

Train MSE Error= [14.56587393]  
Validation MSE-Error= [13.46227744]  
Test MSE-Error= [20.97552949]

خطاهای این مدل از دو مدل پیوسته و پیوسته با ترم‌های درجه دو بدتر شد.