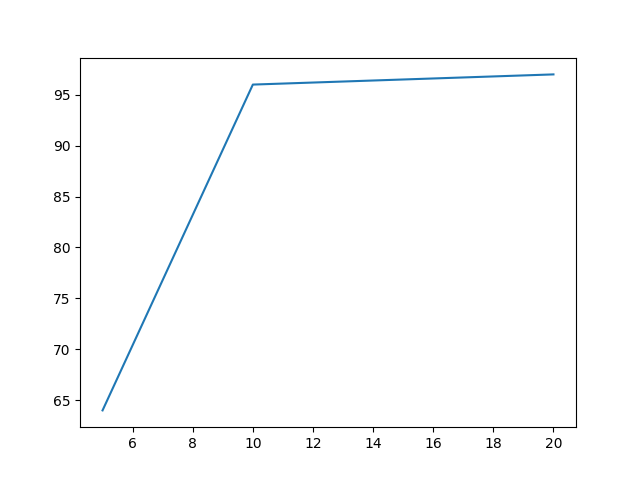
My MLP begins training by first ingesting the 7352 lines of data, and stripping the data of leading spaces and line breaks. The actual data points are then converted into float values.

At this point, my data list is stored as a list of tuples representing each line, with the first item in the tuple being the data from each line, and the second item the activity being performed. I then shuffle the data list ensuring a fair representation of all of the data, before taking my training cases, validation cases, and test cases off of the front of the list, deleting the processed data as I go along to ensure that the data points between training, validation and test cases are all different. I then turn the single integer representing which activity is being performed into a target array where the activity performed is represented with a 1, with the rest of the list empty.

Fortunately, I am running tests on a higher-end machine so I can run larger tests in a much more manageable timeframe. I found that training my MLP with a learning rate of 0.25 and over 100 iterations yielded the best results. Interestingly, I found that initializing the MLP with a logistic output type yielded much lower results than the linear and softmax types. 

These are the test [[342. 3. 1. 0. 0. 0.] results when the [ 3. 307. 0. 0. 0. 0.]

MLP was fed 2000 [ 0. 1. 251. 0. 0. 0.]

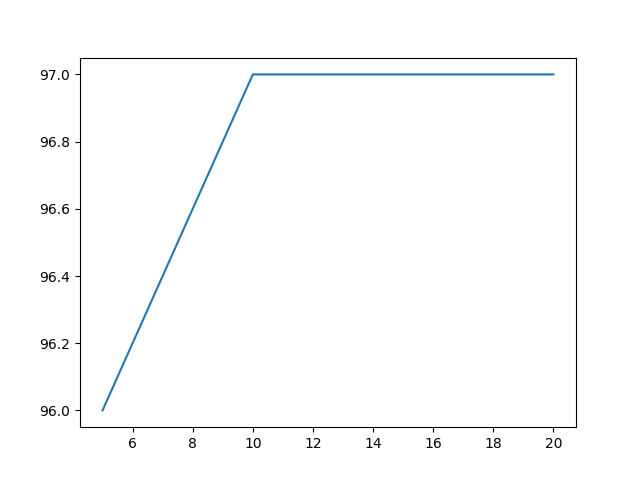
points for training [ 0. 0. 0. 316. 21. 0.]

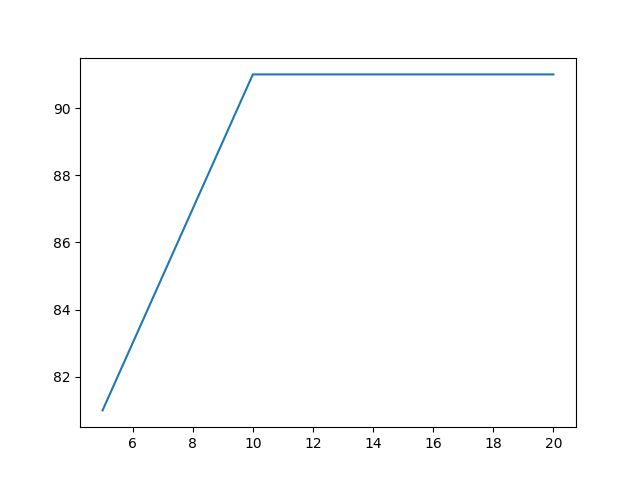
and testing, with [ 0. 0. 0. 26. 340. 0.]

1000 for validation [ 0. 0. 0. 2. 0. 387.]]

and a linear output

type.



These are the results of using the softmax output type, with the same sizes as the inputs. Note that the accuracy started much higher than with the linear output type.

This is the softmax output type once more, this time with the inputs divided by 10 (200 and 100)

[[327. 3. 2. 0. 0. 0.]

[ 0. 270. 4. 0. 0. 0.]

[ 3. 0. 260. 0. 0. 1.]

[ 0. 0. 0. 340. 16. 0.]

[ 0. 0. 0. 12. 385. 1.]

[ 0. 0. 0. 1. 0. 375.]]

[[31. 1. 1. 0. 0. 0.]

[ 0. 31. 4. 0. 0. 0.]

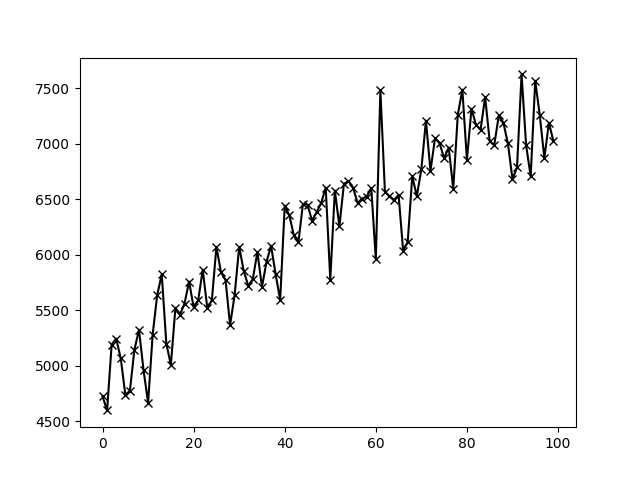
[ 0. 0. 31. 0. 0. 0.]

[ 0. 0. 0. 28. 3. 0.]

[ 0. 0. 0. 8. 26. 1.]

[ 0. 0. 0. 0. 0. 35.]]

My fitness evaluation function runs with the same softmax output type, with a training and testing size of 500 data points, and 100 validity training points. The scoring system I landed on was to multiply the number of removed genes from each given chromosome in the population with the percentage accuracy (over 80) that the chromosome achieved after being trained and tested with early stopping. For example, an accuracy of 90% with 200 removed genes would receive a score of 2000 (fitness = (score - 80) \* removed\_genes). Note that chromosomes with an accuracy of under 80 receive a fitness score of 0.



I then ran tests on a population size of 100 over 100 epochs, utilizing tournaments to help to retain the best chromosomes with 10 elites passing through to the next population every time.

The learning algorithm performed very well, with the best performing chromosome receiving a fitness score of 7625.8, having removed 419 genes and having an accuracy of 98.2%.

I find it very interesting to see the gradual increase of fitness over the 100 epochs, showing me that my programs really were working at accurately guessing the given activities using limited amounts of data inputs. The evolution of these chromosomes took place over a span of ~48 hours, needless to say, using substantial computational power.

I found the chromosome that returned these results and tested it with an MLP that was trained with 2000 data points, with 2000 data points for testing and a validity set of 1000 points.

The chromosome performed fairly well, with a score of 97.5%, this was the resulting confusion matrix.

[[348. 5. 1. 0. 0. 0.]

[ 2. 284. 1. 0. 0. 1.]

[ 2. 0. 254. 0. 0. 0.]

[ 0. 0. 0. 332. 17. 2.]

[ 0. 0. 0. 19. 370. 0.]

[ 0. 0. 0. 0. 0. 362.]]

On the next pages are the features that the best-evolved chromosome featured.

6 tBodyAcc-std()-Z

8 tBodyAcc-mad()-Y

9 tBodyAcc-mad()-Z

10 tBodyAcc-max()-X

15 tBodyAcc-min()-Z

20 tBodyAcc-iqr()-X

21 tBodyAcc-iqr()-Y

23 tBodyAcc-entropy()-X

26 tBodyAcc-arCoeff()-X,1

28 tBodyAcc-arCoeff()-X,3

33 tBodyAcc-arCoeff()-Y,4

37 tBodyAcc-arCoeff()-Z,4

38 tBodyAcc-correlation()-X,Y

40 tBodyAcc-correlation()-Y,Z

41 tGravityAcc-mean()-X

47 tGravityAcc-mad()-X

49 tGravityAcc-mad()-Z

50 tGravityAcc-max()-X

53 tGravityAcc-min()-X

55 tGravityAcc-min()-Z

58 tGravityAcc-energy()-Y

59 tGravityAcc-energy()-Z

62 tGravityAcc-iqr()-Z

67 tGravityAcc-arCoeff()-X,2

72 tGravityAcc-arCoeff()-Y,3

73 tGravityAcc-arCoeff()-Y,4

76 tGravityAcc-arCoeff()-Z,3

86 tBodyAccJerk-std()-Z

88 tBodyAccJerk-mad()-Y

92 tBodyAccJerk-max()-Z

93 tBodyAccJerk-min()-X

97 tBodyAccJerk-energy()-X

102 tBodyAccJerk-iqr()-Z

104 tBodyAccJerk-entropy()-Y

105 tBodyAccJerk-entropy()-Z

106 tBodyAccJerk-arCoeff()-X,1

112 tBodyAccJerk-arCoeff()-Y,3

115 tBodyAccJerk-arCoeff()-Z,2

123 tBodyGyro-mean()-Z

128 tBodyGyro-mad()-Y

130 tBodyGyro-max()-X

132 tBodyGyro-max()-Z

134 tBodyGyro-min()-Y

139 tBodyGyro-energy()-Z

142 tBodyGyro-iqr()-Z

145 tBodyGyro-entropy()-Z

149 tBodyGyro-arCoeff()-X,4

152 tBodyGyro-arCoeff()-Y,3

154 tBodyGyro-arCoeff()-Z,1

156 tBodyGyro-arCoeff()-Z,3

160 tBodyGyro-correlation()-Y,Z

171 tBodyGyroJerk-max()-Y

172 tBodyGyroJerk-max()-Z

174 tBodyGyroJerk-min()-Y

179 tBodyGyroJerk-energy()-Z

182 tBodyGyroJerk-iqr()-Z

183 tBodyGyroJerk-entropy()-X

184 tBodyGyroJerk-entropy()-Y

185 tBodyGyroJerk-entropy()-Z

186 tBodyGyroJerk-arCoeff()-X,1

191 tBodyGyroJerk-arCoeff()-Y,2

192 tBodyGyroJerk-arCoeff()-Y,3

194 tBodyGyroJerk-arCoeff()-Z,1

198 tBodyGyroJerk-correlation()-X,Y

201 tBodyAccMag-mean()

203 tBodyAccMag-mad()

207 tBodyAccMag-energy()

212 tBodyAccMag-arCoeff()3

213 tBodyAccMag-arCoeff()4

220 tGravityAccMag-energy()

223 tGravityAccMag-arCoeff()1

225 tGravityAccMag-arCoeff()3

231 tBodyAccJerkMag-min()

232 tBodyAccJerkMag-sma()

241 tBodyGyroMag-std()

242 tBodyGyroMag-mad()

246 tBodyGyroMag-energy()

247 tBodyGyroMag-iqr()

257 tBodyGyroJerkMag-min()

265 tBodyGyroJerkMag-arCoeff()4

271 fBodyAcc-std()-Z

278 fBodyAcc-min()-X

285 fBodyAcc-iqr()-X

286 fBodyAcc-iqr()-Y

298 fBodyAcc-kurtosis()-X

301 fBodyAcc-skewness()-Z

302 fBodyAcc-kurtosis()-Z

304 fBodyAcc-bandsEnergy()-9,16

326 fBodyAcc-bandsEnergy()-17,32

327 fBodyAcc-bandsEnergy()-33,48

332 fBodyAcc-bandsEnergy()-9,16

335 fBodyAcc-bandsEnergy()-33,40

341 fBodyAcc-bandsEnergy()-33,48

342 fBodyAcc-bandsEnergy()-49,64

344 fBodyAcc-bandsEnergy()-25,48

351 fBodyAccJerk-mad()-X

354 fBodyAccJerk-max()-X

373 fBodyAccJerk-meanFreq()-X

382 fBodyAccJerk-bandsEnergy()-1,8

383 fBodyAccJerk-bandsEnergy()-9,16

385 fBodyAccJerk-bandsEnergy()-25,32

386 fBodyAccJerk-bandsEnergy()-33,40

388 fBodyAccJerk-bandsEnergy()-49,56

402 fBodyAccJerk-bandsEnergy()-49,56

403 fBodyAccJerk-bandsEnergy()-57,64

407 fBodyAccJerk-bandsEnergy()-49,64

408 fBodyAccJerk-bandsEnergy()-1,24

411 fBodyAccJerk-bandsEnergy()-9,16

412 fBodyAccJerk-bandsEnergy()-17,24

425 fBodyGyro-mean()-Y

433 fBodyGyro-max()-X

435 fBodyGyro-max()-Z

441 fBodyGyro-energy()-Y

442 fBodyGyro-energy()-Z

445 fBodyGyro-iqr()-Z

446 fBodyGyro-entropy()-X

447 fBodyGyro-entropy()-Y

450 fBodyGyro-maxInds-Y

451 fBodyGyro-maxInds-Z

458 fBodyGyro-kurtosis()-Y

459 fBodyGyro-skewness()-Z

464 fBodyGyro-bandsEnergy()-25,32

467 fBodyGyro-bandsEnergy()-49,56

480 fBodyGyro-bandsEnergy()-41,48

482 fBodyGyro-bandsEnergy()-57,64

483 fBodyGyro-bandsEnergy()-1,16

484 fBodyGyro-bandsEnergy()-17,32

486 fBodyGyro-bandsEnergy()-49,64

494 fBodyGyro-bandsEnergy()-41,48

496 fBodyGyro-bandsEnergy()-57,64

501 fBodyGyro-bandsEnergy()-1,24

507 fBodyAccMag-min()

512 fBodyAccMag-maxInds

515 fBodyAccMag-kurtosis()

519 fBodyBodyAccJerkMag-max()

523 fBodyBodyAccJerkMag-iqr()

526 fBodyBodyAccJerkMag-meanFreq()

548 fBodyBodyGyroJerkMag-energy()

549 fBodyBodyGyroJerkMag-iqr()

550 fBodyBodyGyroJerkMag-entropy()

553 fBodyBodyGyroJerkMag-skewness()

558 angle(tBodyGyroJerkMean,gravityMean)