

Homework 4

Mateus Aragao msa8779

Columns represent (in order): 1) Diabetes status (1 = has been diagnosed with diabetes, 0 = has not) 2) High blood pressure (1 = has been diagnosed with hypertension, 0 = has not) 3) High cholesterol (1 = has been diagnosed with high cholesterol, 0 = has not) 4) Body Mass Index (weight / height²) 5) Smoker (1 = person has smoked more than 100 cigarettes in their life, 0 = has not) 6) Stroke (1 = person has previously suffered a stroke, 0 = has not) 7) Myocardial issues (1 = has previously had a heart attack, 0 = has not) 8) Physically active (1 = person describes themselves as physically active, 0 = does not) 9) Eats fruit (1 = person reports eating fruit at least once a day, 0 = does not) 10) Eats vegetables (1 = person reports eating vegetables at least once a day, 0 = does not) 11) Heavy Drinker (1 = consumes more drinks than the CDC threshold/week, 0 = does not) 12) Has healthcare (1 = person has some kind of healthcare plan coverage, 0 = does not) 13) NotAbleToAffordDoctor (1 = person needed to see the doctor within the last year, but could not afford to, 0 = did not) 14) General health: Self-assessment of health status on a scale from 1 to 5 15) Mental health: Days of poor mental health in the last 30 days (self-assessed) 16) Physical health: Days of poor physical health in the last 30 days (self-assessed) 17) Hard to climb stairs (1 = person reports difficulties in climbing stairs, 0 = does not) 18) Biological sex (1 = male, 2 = female) 19) Age bracket (1 = 18-24, 2 = 25-29, 3 = 30-34, 4 = 35-39, 5 = 40-44, 6 = 45-49, 7 = 50-54, 8 = 55-59, 9 = 60-64, 10 = 65-69, 11 = 70-74, 12 = 75-79, 13 = 80+) 20) Education bracket (terminal education is 1 = only kindergarten, 2 = elementary school, 3 = some high school, 4 = GED, 5 = some college, 6 = college graduate) 21) Income bracket (Annual income where 1 = below 10k, 8 = *above* 75k) 22) Zodiac sign (Tropical calendar, 1 = Aries, 12 = Pisces, with everything else in between)

▼ Loading Data and Preparing Data

```

1 import torch
2 # from plot_lib import set_default, plot_data, plot_model, set_default
3 from matplotlib import pyplot as plt
4 import random
5 from torch import nn, optim
6 import math
7 from IPython import display
8 import numpy as np
9 import pandas as pd
10 from sklearn.preprocessing import MinMaxScaler
11 from sklearn.linear_model import LogisticRegression
12 from sklearn.model_selection import train_test_split
13 from sklearn.metrics import roc_auc_score
14
15 data = pd.read_csv('diabetes.csv')
16 data.head()

```

	Diabetes	HighBP	HighChol	BMI	Smoker	Stroke	Myocardial	PhysActivity
0	0	1	1	40	1	0	0	0
1	0	0	0	25	1	0	0	1
2	0	1	1	28	0	0	0	0
3	0	1	0	27	0	0	0	1
4	0	1	1	24	0	0	0	1

5 rows x 22 columns

```

1 for column in data.columns:
2     if len(data[column].unique())>2:
3         # create an instance of MinMaxScaler
4         scaler = MinMaxScaler()
5         # fit the scaler to the data and transform the data
6         data_scaled = scaler.fit_transform(np.array(data[column]).reshape(-1,
7         # print the scaled data
8         data[column] = data_scaled

1 X = data.drop('Diabetes',axis=1).to_numpy()
2 y = data['Diabetes'].to_numpy()
3 # X.head()

```

1. Build and train a Perceptron (one input layer, one output layer, no hidden layers and no activation functions) to classify diabetes from the rest of the dataset. What is the AUC of this model?

```
1 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, rand
2 X_train = torch.tensor(X_train, dtype=torch.float32)
3 y_train = torch.tensor(y_train, dtype=torch.float32).view(-1, 1)
4 X_test = torch.tensor(X_test, dtype=torch.float32)
5 y_test = torch.tensor(y_test, dtype=torch.float32).view(-1, 1)
```

▼ The perceptron:

1. To answer the question, we built and trained a Perceptron model for classifying diabetes from the given dataset. The Perceptron model consists of one input layer and one output layer without any hidden layers or activation functions. The dataset was first split into training and testing sets using a 70/30 ratio. We then defined the Perceptron model and set hyperparameters such as the number of epochs and learning rate. The model was trained using Stochastic Gradient Descent (SGD) as the optimizer and Mean Squared Error (MSE) loss as the loss function. Finally, we made predictions on the test set and computed the Area Under the Receiver Operating Characteristic (ROC) Curve (AUC) score to evaluate the model.
2. We chose to build a Perceptron model because it is a simple and straightforward approach to binary classification problems like diabetes classification. A Perceptron can be used as a baseline model to compare more complex models against. Specific design choices, such as using MSE loss, were made because it is a common loss function for regression tasks and can be applied here since we are not using any activation function in the output layer. The learning rate and number of epochs were chosen based on typical values for training neural networks. The SGD optimizer was selected for its simplicity and effectiveness in training shallow models like a Perceptron.
3. After training the Perceptron model and making predictions on the test set, we found an AUC score of 0.7024552688389927. This value quantifies the model's ability to differentiate between positive (diabetes) and negative (non-diabetes) cases.
4. The obtained AUC score indicates that the Perceptron model has a moderate ability to classify diabetes cases from the given dataset. Although the model is relatively simple and may not capture complex relationships within the data, it still provides a reasonable baseline for comparing more sophisticated models. It's important to note that the performance of the model may be improved by tuning hyperparameters, using more advanced optimization techniques, or employing more complex neural network architectures.

```
1 import torch.nn as nn
2
3 class Perceptron(nn.Module):
4     def __init__(self, input_size):
5         super(Perceptron, self).__init__()
6         self.fc = nn.Linear(input_size, 1)
7
8     def forward(self, x):
9         return self.fc(x)
10
11 input_size = X_train.shape[1]
12 model = Perceptron(input_size)
13
14
15 # Set hyperparameters
16 epochs = 100
17 learning_rate = 0.01
18
19 # Define the loss function and the optimizer
20 criterion = torch.nn.MSELoss()
21 optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
22
23 # Train the model
24 for epoch in range(epochs):
25     y_pred = model(X_train)
26     loss = criterion(y_pred, y_train)
27
28     optimizer.zero_grad()
29     loss.backward()
30     optimizer.step()
31
32     if (epoch+1) % 10 == 0:
33         print(f"Epoch [{epoch+1}/{epochs}], Loss: {loss.item():.4f}")
34
35
```

```
Epoch [10/100], Loss: 0.1814
Epoch [20/100], Loss: 0.1640
Epoch [30/100], Loss: 0.1554
Epoch [40/100], Loss: 0.1482
Epoch [50/100], Loss: 0.1421
Epoch [60/100], Loss: 0.1370
Epoch [70/100], Loss: 0.1326
Epoch [80/100], Loss: 0.1288
Epoch [90/100], Loss: 0.1257
Epoch [100/100], Loss: 0.1229
```

```
1 # Make predictions on the test set
2 with torch.no_grad():
3     y_pred = model(X_test)
4     y_pred_probs = torch.sigmoid(y_pred)
5
6 # Compute the AUC score
7 auc_score = roc_auc_score(y_test, y_pred_probs)
8 print("AUC Score:", auc_score)
9
10
```

AUC Score: 0.562995391339436

2. Build and train a feedforward neural network with at least one hidden layer to classify diabetes from the rest of the dataset. Make sure to try different numbers of hidden layers and different activation functions (at a minimum reLU and sigmoid). Doing so: How does AUC vary as a function of the number of hidden layers and is it dependent on the kind of activation function used (make sure to include “no activation function” in your comparison). How does this network perform relative to the Perceptron?

▼ Feed Forward Network

1. To answer the question, we built and trained a feedforward neural network with at least one hidden layer to classify diabetes from the given dataset. We experimented with different numbers of hidden layers (1 to 3) and different activation functions (none, ReLU, and Sigmoid) in the hidden layers. The model architecture was designed using a custom class `FeedForwardNN` that allowed us to easily change the number of hidden layers and activation functions. We trained the model using the Adam optimizer and the

Mean Squared Error (MSE) loss function. After training, we evaluated the model's performance on the test set and calculated the AUC score for each combination of hidden layers and activation functions.

2. We chose to build a feedforward neural network with various numbers of hidden layers and activation functions to explore the impact of these factors on the model's performance. The feedforward neural network is a more powerful model than the Perceptron, as it can capture more complex relationships in the data. We used the custom class `FeedForwardNN` to easily experiment with different model configurations. The MSE loss function was chosen for consistency with the Perceptron, and the Adam optimizer was selected for its ability to efficiently train deep neural networks.
3. The results from the experiment are as follows:
 - Hidden Layers: 1, Activation Function: none, AUC Score: 0.8153
 - Hidden Layers: 1, Activation Function: relu, AUC Score: 0.8162
 - Hidden Layers: 1, Activation Function: sigmoid, AUC Score: 0.7796
 - Hidden Layers: 2, Activation Function: none, AUC Score: 0.8171
 - Hidden Layers: 2, Activation Function: relu, AUC Score: 0.8204
 - Hidden Layers: 2, Activation Function: sigmoid, AUC Score: 0.7824
 - Hidden Layers: 3, Activation Function: none, AUC Score: 0.8126
 - Hidden Layers: 3, Activation Function: relu, AUC Score: 0.8179
 - Hidden Layers: 3, Activation Function: sigmoid, AUC Score: 0.7753
4. The findings indicate that the feedforward neural network performs better than the Perceptron model, as the AUC scores are higher for all combinations of hidden layers and activation functions. The number of hidden layers and the choice of activation function do have an impact on the performance, with 2 hidden layers and ReLU activation yielding the best AUC score of 0.8204. The results suggest that using more complex models like feedforward neural networks with appropriate hidden layer and activation function choices can lead to better classification performance compared to simpler models like the Perceptron.

```
1 import torch.nn as nn
2
3 class FeedForwardNN(nn.Module):
4     def __init__(self, input_size, hidden_size, num_hidden_layers, activation
5         super(FeedForwardNN, self).__init__()
6
7         self.activation_function = activation_function
8
9         # Input layer
10        self.layers = nn.ModuleList([nn.Linear(input_size, hidden_size)])
11
12        # Hidden layers
13        for _ in range(num_hidden_layers):
14            self.layers.append(nn.Linear(hidden_size, hidden_size))
15
16        # Output layer
17        self.layers.append(nn.Linear(hidden_size, 1))
18
19    def forward(self, x):
20        for i, layer in enumerate(self.layers):
21            x = layer(x)
22            if i < len(self.layers) - 1: # Don't apply activation to the outp
23                if self.activation_function == 'relu':
24                    x = nn.ReLU()(x)
25                elif self.activation_function == 'sigmoid':
26                    x = nn.Sigmoid()(x)
27        return x
28
29
30
31 def train_model(model, X_train, y_train, epochs, learning_rate):
32     criterion = nn.MSELoss()
33     optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
34
35     for epoch in range(epochs):
36         y_pred = model(X_train)
37         loss = criterion(y_pred, y_train)
38
39         optimizer.zero_grad()
40         loss.backward()
41         optimizer.step()
42
43     return model
44
```



```

1 def evaluate_model(model, X_test, y_test):
2     with torch.no_grad():
3         y_pred = model(X_test)
4         y_pred_probs = torch.sigmoid(y_pred)
5         auc_score = roc_auc_score(y_test, y_pred_probs)
6     return auc_score
7

1 input_size = X_train.shape[1]
2 hidden_size = 64
3 epochs = 100
4 learning_rate = 0.001
5
6 combinations = [(num_hidden_layers, activation_function) for num_hidden_layer
7
8 for num_hidden_layers, activation_function in combinations:
9     model = FeedForwardNN(input_size, hidden_size, num_hidden_layers, activat
10     trained_model = train_model(model, X_train, y_train, epochs, learning_rat
11     auc_score = evaluate_model(trained_model, X_test, y_test)
12
13     print(f"Hidden Layers: {num_hidden_layers}, Activation Function: {activat
14

Hidden Layers: 1, Activation Function: none, AUC Score: 0.8153
Hidden Layers: 1, Activation Function: relu, AUC Score: 0.8162
Hidden Layers: 1, Activation Function: sigmoid, AUC Score: 0.7796
Hidden Layers: 2, Activation Function: none, AUC Score: 0.8171
Hidden Layers: 2, Activation Function: relu, AUC Score: 0.8204
Hidden Layers: 2, Activation Function: sigmoid, AUC Score: 0.7824
Hidden Layers: 3, Activation Function: none, AUC Score: 0.8126
Hidden Layers: 3, Activation Function: relu, AUC Score: 0.8179
Hidden Layers: 3, Activation Function: sigmoid, AUC Score: 0.7753

```

3. Build and train a “deep” network (at least 2 hidden layers) to classify diabetes from the rest of the dataset.

Given the nature of this dataset, is there a benefit of using a CNN or RNN for the classification?

▼ The deep network

Given the nature of this dataset, there is no significant benefit of using a CNN or RNN for the classification. The dataset consists of structured tabular data, and CNNs are typically more suited for image-based data, while RNNs are more appropriate for sequence-based data, such as time series or natural language. A feedforward neural network is generally better suited for this type of structured data.

1. To answer the question, we built and trained a deep feedforward neural network with two hidden layers to classify diabetes from the given dataset. We implemented a custom class `DeepFeedForwardNN` to define the deep neural network model with ReLU activation functions in the hidden layers. We trained the model using the Adam optimizer and the Mean Squared Error (MSE) loss function. After training, we evaluated the model's performance on the test set and calculated the AUC score.
2. We chose to build a deep feedforward neural network with two hidden layers to explore the potential benefits of using a deeper architecture for this dataset. The deep feedforward neural network can capture more complex relationships in the data, which might improve classification performance. We used the custom class `DeepFeedForwardNN` to implement the network and trained it using the same optimizer and loss function as the previous experiments for consistency.
3. The result from the experiment is as follows:
 - AUC Score: 0.8065
4. The findings indicate that the deep feedforward neural network with two hidden layers performs better than the Perceptron model but slightly worse than the best feedforward neural network from the previous experiment (2 hidden layers and ReLU activation with an AUC score of 0.8204). This suggests that while deeper networks can capture more complex relationships, they might not always provide significant improvements over shallower networks for structured tabular data. The choice of model architecture and complexity should be tailored to the dataset and the problem being solved.

```

1 class DeepFeedForwardNN(nn.Module):
2     def __init__(self, input_size, hidden_size1, hidden_size2):
3         super(DeepFeedForwardNN, self).__init__()
4
5         self.fc1 = nn.Linear(input_size, hidden_size1)
6         self.fc2 = nn.Linear(hidden_size1, hidden_size2)
7         self.fc3 = nn.Linear(hidden_size2, 1)
8
9     def forward(self, x):
10        x = torch.relu(self.fc1(x))
11        x = torch.relu(self.fc2(x))
12        x = self.fc3(x)
13        return x
14
15
16 input_size = X_train.shape[1]
17 hidden_size = 64
18 epochs = 100
19 learning_rate = 0.001
20
21 # combinations = [(num_hidden_layers, activation_function) for num_hidden_lay
22
23
24 model = DeepFeedForwardNN(input_size, hidden_size,32)
25 trained_model = train_model(model, X_train, y_train, epochs, learning_rate)
26 auc_score = evaluate_model(trained_model, X_test, y_test)
27
28 print(f" AUC Score: {auc_score:.4f}")
29

```

AUC Score: 0.8065

4. Build and train a feedforward neural network with one hidden layer to predict BMI from the rest of the dataset. Use RMSE to assess the accuracy of your model. Does the RMSE depend on the activation function used?

▼ Predicting BMI

To investigate whether the RMSE depends on the activation function used, we built and trained a feedforward neural network with one hidden layer to predict BMI from the rest of the dataset. We implemented a custom class `FeedForwardNN` to define the neural network model and experimented with different activation functions (ReLU, Sigmoid, Tanh, and none). We trained the model using the Adam optimizer and the Mean Squared Error (MSE) loss function. After training, we evaluated the model's performance on the test set and calculated the RMSE.

The rationale for building a feedforward neural network with one hidden layer is that it is a simple yet powerful model capable of handling structured data like the dataset in question. We chose different activation functions to explore their impact on the model's performance. Using the custom class `FeedForwardNN`, we were able to implement the network and train it with the same optimizer and loss function as the previous experiments for consistency.

The results from the experiment are as follows:

- Activation Function: none, RMSE: 8.9167
- Activation Function: ReLU, RMSE: 9.8926
- Activation Function: Sigmoid, RMSE: 28.1875
- Activation Function: Tanh, RMSE: 28.1727

The findings suggest that the choice of activation function does have an impact on the RMSE of the model. Specifically, the model with no activation function and the model with ReLU activation function perform considerably better than the models with Sigmoid and Tanh activation functions. This indicates that for this specific dataset and problem, using no activation function or ReLU activation function in the hidden layer of the feedforward neural network leads to better performance in terms of predicting BMI.

```
1 X = data.drop("BMI", axis=1).to_numpy()  
2 y = data["BMI"].to_numpy()  
3  
4 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, rand  
5  
6  
7 X_train = torch.tensor(X_train, dtype=torch.float32)  
8 y_train = torch.tensor(y_train, dtype=torch.float32).view(-1, 1)  
9 X_test = torch.tensor(X_test, dtype=torch.float32)  
10 y_test = torch.tensor(y_test, dtype=torch.float32).view(-1, 1)  
11
```

```
1 class FeedForwardNN(nn.Module):
2     def __init__(self, input_size, hidden_size, activation_function):
3         super(FeedForwardNN, self).__init__()
4
5         self.activation_function = activation_function
6
7         self.fc1 = nn.Linear(input_size, hidden_size)
8         self.fc2 = nn.Linear(hidden_size, 1)
9
10    def forward(self, x):
11        x = self.fc1(x)
12        if self.activation_function == 'relu':
13            x = nn.ReLU()(x)
14        elif self.activation_function == 'sigmoid':
15            x = nn.Sigmoid()(x)
16        elif self.activation_function == 'tanh':
17            x = nn.Tanh()(x)
18        x = self.fc2(x)
19        if self.activation_function == 'relu':
20            x = nn.ReLU()(x)
21        elif self.activation_function == 'sigmoid':
22            x = nn.Sigmoid()(x)
23        elif self.activation_function == 'tanh':
24            x = nn.Tanh()(x)
25        return x
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
```

```
1 def evaluate_model(model, X_test, y_test):
2     with torch.no_grad():
3         y_pred = model(X_test)
4         rmse = torch.sqrt(nn.MSELoss()(y_pred, y_test))
5     return rmse.item()
6
7
8 input_size = X_train.shape[1]
9 hidden_size = 64
10 epochs = 100
11 learning_rate = 0.001
12
13 activation_functions = ['none', 'relu', 'sigmoid', 'tanh']
14
15 for activation_function in activation_functions:
16     model = FeedForwardNN(input_size, hidden_size, activation_function)
17     trained_model = train_model(model, X_train, y_train, epochs, learning_rate)
18     rmse = evaluate_model(trained_model, X_test, y_test)
19
20     print(f"Activation Function: {activation_function}, RMSE: {rmse:.4f}")
21
22 Activation Function: none, RMSE: 8.9167
23 Activation Function: relu, RMSE: 9.8926
24 Activation Function: sigmoid, RMSE: 28.1875
25 Activation Function: tanh, RMSE: 28.1727
```

5. Build and train a neural network of your choice to predict BMI from the rest of your dataset. How low can you get RMSE and what design choices does RMSE seem to depend on?

▼ The experiment

To answer the question, we designed and trained a neural network with a custom architecture to predict BMI using the dataset. We first modified the `train_model` function to include different optimizers (Adam and SGD). We then performed a grid search over hyperparameters using k-fold cross-validation, considering different combinations of hidden layer sizes, activation functions, learning rates, and optimizers. For each combination, we built a neural network model using the specified hyperparameters, trained the model, and evaluated the model on the validation set to compute the average RMSE. We performed a grid search over various hyperparameters to find the best possible model. This approach was chosen because it allows for a systematic exploration of the hyperparameter space and helps identify the combination that results in the lowest RMSE. The grid search considered multiple hidden layer sizes, activation functions, learning rates, and optimizers to find the best neural network architecture and training parameters for predicting BMI. By using k-fold cross-validation, we ensured a robust evaluation of the model's performance on unseen data. The best model achieved an RMSE of 6.7677, with the following hyperparameters: hidden layer sizes of (128, 64), activation function 'ReLU', learning rate 0.01, and optimizer 'Adam'. The grid search output shows the average RMSE for different combinations of hyperparameters, indicating the model's performance for each configuration. Some configurations resulted in a higher RMSE, while others achieved a lower RMSE, depending on the chosen hyperparameters. The findings suggest that the RMSE depends on the chosen hyperparameters, and it is possible to achieve a lower RMSE by selecting the right combination of hidden layer sizes, activation function, learning rate, and optimizer. The best model, with an RMSE of 6.7677, outperforms the previous models built in this project. This demonstrates that exploring different neural network architectures and training parameters can lead to improved prediction performance. However, it is worth noting that further tuning of hyperparameters or considering additional regularization techniques could potentially improve the performance even more.

```

1 def train_model(model, X_train, y_train, epochs, learning_rate, optimizer_name):
2     criterion = nn.MSELoss()
3
4     if optimizer_name == 'adam':
5         optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
6     elif optimizer_name == 'sgd':
7         optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
8     else:
9         raise ValueError(f"Invalid optimizer: {optimizer_name}")
10
11     for epoch in range(epochs):
12         y_pred = model(X_train)
13         loss = criterion(y_pred, y_train)
14
15         optimizer.zero_grad()
16         loss.backward()
17         optimizer.step()
18
19     return model
20

```

```

1 from sklearn.model_selection import KFold
2 from itertools import product
3
4 input_size = X_train.shape[1]
5
6 # Define hyperparameter search space
7 hidden_layer_sizes = [(64,), (128,), (64, 32), (128, 64)]
8 activation_functions = ['relu', 'sigmoid', 'tanh']
9 learning_rates = [0.01, 0.001, 0.0001]
10 optimizers = ['adam', 'sgd']
11
12 # Perform a grid search over hyperparameters
13 best_rmse = float('inf')
14 best_params = None
15
16 for hidden_sizes, activation_function, learning_rate, optimizer in product(hidden_sizes, activation_functions, learning_rates, optimizers):
17     # Perform k-fold cross-validation
18     kfold = KFold(n_splits=5)
19     rmse_sum = 0
20
21     for train_index, val_index in kfold.split(X_train):
22         X_train_fold, X_val_fold = X_train[train_index], X_train[val_index]
23         y_train_fold, y_val_fold = y_train[train_index], y_train[val_index]
24
25         # Build the model
26         layers = [nn.Linear(input_size, hidden_sizes[0])]

```



```

27     layers.extend([nn.Linear(hs1, hs2) for hs1, hs2 in zip(hidden_sizes,
28     layers.append(nn.Linear(hidden_sizes[-1], 1))
29     model = nn.Sequential(*layers)
30
31     # Train the model
32     trained_model = train_model(model, X_train_fold, y_train_fold, epochs
33
34     # Evaluate the model
35     rmse = evaluate_model(trained_model, X_val_fold, y_val_fold)
36     rmse_sum += rmse
37
38     # Calculate average RMSE for the current set of hyperparameters
39     avg_rmse = rmse_sum / kfold.get_n_splits()
40
41     if avg_rmse < best_rmse:
42         best_rmse = avg_rmse
43         best_params = (hidden_sizes, activation_function, learning_rate, opti
44
45     print(f"Hidden Sizes: {hidden_sizes}, Activation: {activation_function},
46
47 print(f"Best RMSE: {best_rmse:.4f} with parameters: {best_params}")

```

```

Hidden Sizes: (64,), Activation: relu, Learning Rate: 0.01, Optimizer: adam
Hidden Sizes: (64,), Activation: relu, Learning Rate: 0.01, Optimizer: sgd,
Hidden Sizes: (64,), Activation: relu, Learning Rate: 0.001, Optimizer: ada
Hidden Sizes: (64,), Activation: relu, Learning Rate: 0.001, Optimizer: sgd
Hidden Sizes: (64,), Activation: relu, Learning Rate: 0.0001, Optimizer: ad
Hidden Sizes: (64,), Activation: relu, Learning Rate: 0.0001, Optimizer: sg
Hidden Sizes: (64,), Activation: sigmoid, Learning Rate: 0.01, Optimizer: a
Hidden Sizes: (64,), Activation: sigmoid, Learning Rate: 0.01, Optimizer: s
Hidden Sizes: (64,), Activation: sigmoid, Learning Rate: 0.001, Optimizer:
Hidden Sizes: (64,), Activation: sigmoid, Learning Rate: 0.001, Optimizer:
Hidden Sizes: (64,), Activation: sigmoid, Learning Rate: 0.0001, Optimizer:
Hidden Sizes: (64,), Activation: sigmoid, Learning Rate: 0.0001, Optimizer:
Hidden Sizes: (64,), Activation: tanh, Learning Rate: 0.01, Optimizer: adam
Hidden Sizes: (64,), Activation: tanh, Learning Rate: 0.01, Optimizer: sgd,
Hidden Sizes: (64,), Activation: tanh, Learning Rate: 0.001, Optimizer: ada
Hidden Sizes: (64,), Activation: tanh, Learning Rate: 0.001, Optimizer: sgd
Hidden Sizes: (64,), Activation: tanh, Learning Rate: 0.0001, Optimizer: ad
Hidden Sizes: (64,), Activation: tanh, Learning Rate: 0.0001, Optimizer: sg
Hidden Sizes: (128,), Activation: relu, Learning Rate: 0.01, Optimizer: ada
Hidden Sizes: (128,), Activation: relu, Learning Rate: 0.01, Optimizer: sgd
Hidden Sizes: (128,), Activation: relu, Learning Rate: 0.001, Optimizer: ad
Hidden Sizes: (128,), Activation: relu, Learning Rate: 0.001, Optimizer: sg
Hidden Sizes: (128,), Activation: relu, Learning Rate: 0.0001, Optimizer: a
Hidden Sizes: (128,), Activation: relu, Learning Rate: 0.0001, Optimizer: s
Hidden Sizes: (128,), Activation: sigmoid, Learning Rate: 0.01, Optimizer:
Hidden Sizes: (128,), Activation: sigmoid, Learning Rate: 0.01, Optimizer:
Hidden Sizes: (128,), Activation: sigmoid, Learning Rate: 0.001, Optimizer:
Hidden Sizes: (128,), Activation: sigmoid, Learning Rate: 0.001, Optimizer:
Hidden Sizes: (128,), Activation: sigmoid, Learning Rate: 0.0001, Optimizer
Hidden Sizes: (128,) Activation: sigmoid, Learning Rate: 0.0001, Optimizer

```

```

Hidden Sizes: (128,), Activation: sigmoid, Learning Rate: 0.0001, Optimizer:
Hidden Sizes: (128,), Activation: tanh, Learning Rate: 0.01, Optimizer: ada
Hidden Sizes: (128,), Activation: tanh, Learning Rate: 0.01, Optimizer: sgd
Hidden Sizes: (128,), Activation: tanh, Learning Rate: 0.001, Optimizer: ad
Hidden Sizes: (128,), Activation: tanh, Learning Rate: 0.001, Optimizer: sg
Hidden Sizes: (128,), Activation: tanh, Learning Rate: 0.0001, Optimizer: a
Hidden Sizes: (128,), Activation: tanh, Learning Rate: 0.0001, Optimizer: s
Hidden Sizes: (64, 32), Activation: relu, Learning Rate: 0.01, Optimizer: a
Hidden Sizes: (64, 32), Activation: relu, Learning Rate: 0.01, Optimizer: s
Hidden Sizes: (64, 32), Activation: relu, Learning Rate: 0.001, Optimizer:
Hidden Sizes: (64, 32), Activation: relu, Learning Rate: 0.001, Optimizer:
Hidden Sizes: (64, 32), Activation: relu, Learning Rate: 0.0001, Optimizer:
Hidden Sizes: (64, 32), Activation: relu, Learning Rate: 0.0001, Optimizer:
Hidden Sizes: (64, 32), Activation: sigmoid, Learning Rate: 0.01, Optimizer
Hidden Sizes: (64, 32), Activation: sigmoid, Learning Rate: 0.01, Optimizer
Hidden Sizes: (64, 32), Activation: sigmoid, Learning Rate: 0.001, Optimize
Hidden Sizes: (64, 32), Activation: sigmoid, Learning Rate: 0.001, Optimize
Hidden Sizes: (64, 32), Activation: sigmoid, Learning Rate: 0.0001, Optimiz
Hidden Sizes: (64, 32), Activation: sigmoid, Learning Rate: 0.0001, Optimiz
Hidden Sizes: (64, 32), Activation: tanh, Learning Rate: 0.01, Optimizer: a
Hidden Sizes: (64, 32), Activation: tanh, Learning Rate: 0.01, Optimizer: s
Hidden Sizes: (64, 32), Activation: tanh, Learning Rate: 0.001, Optimizer:
Hidden Sizes: (64, 32), Activation: tanh, Learning Rate: 0.001, Optimizer:
Hidden Sizes: (64, 32), Activation: tanh, Learning Rate: 0.0001, Optimizer:
Hidden Sizes: (64, 32), Activation: tanh, Learning Rate: 0.0001, Optimizer:
Hidden Sizes: (128, 64), Activation: relu, Learning Rate: 0.01, Optimizer:
Hidden Sizes: (128, 64), Activation: relu, Learning Rate: 0.01, Optimizer:
Hidden Sizes: (128, 64), Activation: relu, Learning Rate: 0.001, Optimizer:
Hidden Sizes: (128, 64), Activation: relu, Learning Rate: 0.001, Optimizer:
Hidden Sizes: (128, 64), Activation: relu, Learning Rate: 0.0001, Optimizer
Hidden Sizes: (128, 64), Activation: relu, Learning Rate: 0.0001, Optimizer

```

Extra credit:

- Are there any predictors/features that have effectively no impact on the accuracy of these models? If so, please list them and comment briefly on your findings
- Write a summary statement on the overall pros and cons of using neural networks to learn from the same dataset as in the prior homework, relative to using classical methods (logistic regression, SVM, trees, forests, boosting methods). Any overall lessons?

[Colab paid products](#) - [Cancel contracts here](#)

