ZAD 5

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
df = pd.read_csv('Boston.csv')
print(df)
```

```
     Unnamed: 0     crim    zn  indus  chas    nox     rm   age     dis  rad  \
0             1  0.00632  18.0   2.31     0  0.538  6.575  65.2  4.0900    1
1             2  0.02731   0.0   7.07     0  0.469  6.421  78.9  4.9671    2
2             3  0.02729   0.0   7.07     0  0.469  7.185  61.1  4.9671    2
3             4  0.03237   0.0   2.18     0  0.458  6.998  45.8  6.0622    3
4             5  0.06905   0.0   2.18     0  0.458  7.147  54.2  6.0622    3
..          ...      ...   ...    ...   ...    ...    ...   ...     ...  ...
501         502  0.06263   0.0  11.93     0  0.573  6.593  69.1  2.4786    1
502         503  0.04527   0.0  11.93     0  0.573  6.120  76.7  2.2875    1
503         504  0.06076   0.0  11.93     0  0.573  6.976  91.0  2.1675    1
504         505  0.10959   0.0  11.93     0  0.573  6.794  89.3  2.3889    1
505         506  0.04741   0.0  11.93     0  0.573  6.030  80.8  2.5050    1

     tax  ptratio   black  lstat  medv
0    296     15.3  396.90   4.98  24.0
1    242     17.8  396.90   9.14  21.6
2    242     17.8  392.83   4.03  34.7
3    222     18.7  394.63   2.94  33.4
4    222     18.7  396.90   5.33  36.2
..   ...      ...     ...    ...   ...
501  273     21.0  391.99   9.67  22.4
502  273     21.0  396.90   9.08  20.6
503  273     21.0  396.90   5.64  23.9
504  273     21.0  393.45   6.48  22.0
505  273     21.0  396.90   7.88  11.9

[506 rows x 15 columns]
```

```
df.head()
```

	Unnamed: 0	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lstat	medv
0	1	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
1	2	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
2	3	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
3	4	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
4	5	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2

```
rm=df.iloc[:,6]
```

```
rm
```

```
0      6.575
1      6.421
2      7.185
3      6.998
4      7.147
       ...
501    6.593
502    6.120
503    6.976
504    6.794
505    6.030
Name: rm, Length: 506, dtype: float64
```

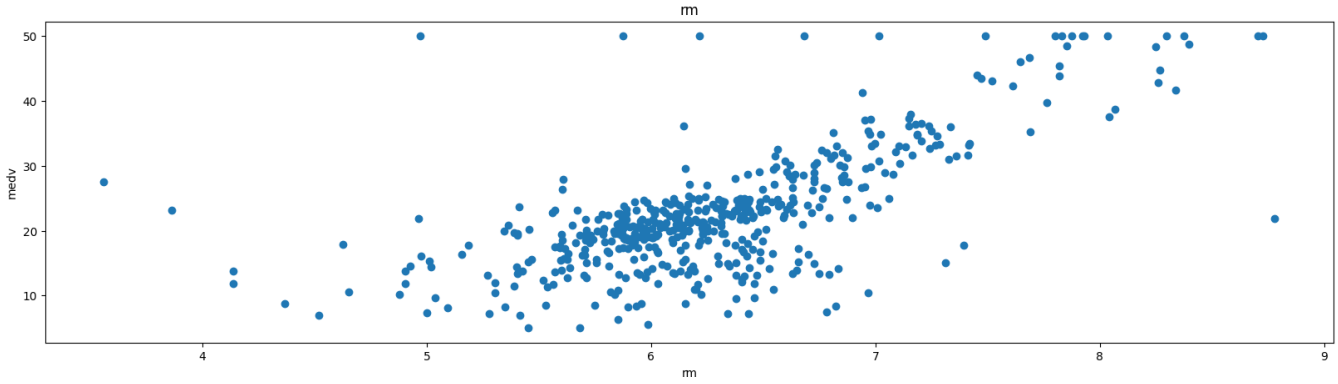
```
medv=df.iloc[:,14]
```

```
medv
```

```
0      24.0
1      21.6
2      34.7
3      33.4
4      36.2
       ...
501    22.4
502    20.6
503    23.9
504    22.0
```

```
505    11.9
Name: medv, Length: 506, dtype: float64
```

```
plt.figure(figsize=(20, 5))

features = ['rm']
target = df['medv']

for i, col in enumerate(features):
    plt.subplot(1, len(features) , i+1)
    x = df[col]
    y = target
    plt.scatter(x, y, marker='o')
    plt.title(col)
    plt.xlabel(col)
    plt.ylabel('medv')
```




```
real_rm = np.array(rm)
real_medv = np.array(medv)
```

```
real_rm
```

```
array([6.575, 6.421, 7.185, 6.998, 7.147, 6.43 , 6.012, 6.172, 5.631,
       6.004, 6.377, 6.009, 5.889, 5.949, 6.096, 5.834, 5.935, 5.99 ,
       5.456, 5.727, 5.57 , 5.965, 6.142, 5.813, 5.924, 5.599, 5.813,
       6.047, 6.495, 6.674, 5.713, 6.072, 5.95 , 5.701, 6.096, 5.933,
       5.841, 5.85 , 5.966, 6.595, 7.024, 6.77 , 6.169, 6.211, 6.069,
       5.682, 5.786, 6.03 , 5.399, 5.602, 5.963, 6.115, 6.511, 5.998,
       5.888, 7.249, 6.383, 6.816, 6.145, 5.927, 5.741, 5.966, 6.456,
       6.762, 7.104, 6.29 , 5.787, 5.878, 5.594, 5.885, 6.417, 5.961,
       6.065, 6.245, 6.273, 6.286, 6.279, 6.14 , 6.232, 5.874, 6.727,
       6.619, 6.302, 6.167, 6.389, 6.63 , 6.015, 6.121, 7.007, 7.079,
       6.417, 6.405, 6.442, 6.211, 6.249, 6.625, 6.163, 8.069, 7.82 ,
       7.416, 6.727, 6.781, 6.405, 6.137, 6.167, 5.851, 5.836, 6.127,
       6.474, 6.229, 6.195, 6.715, 5.913, 6.092, 6.254, 5.928, 6.176,
       6.021, 5.872, 5.731, 5.87 , 6.004, 5.961, 5.856, 5.879, 5.986,
       5.613, 5.693, 6.431, 5.637, 6.458, 6.326, 6.372, 5.822, 5.757,
       6.335, 5.942, 6.454, 5.857, 6.151, 6.174, 5.019, 5.403, 5.468,
       4.903, 6.13 , 5.628, 4.926, 5.186, 5.597, 6.122, 5.404, 5.012,
       5.709, 6.129, 6.152, 5.272, 6.943, 6.066, 6.51 , 6.25 , 7.489,
       7.802, 8.375, 5.854, 6.101, 7.929, 5.877, 6.319, 6.402, 5.875,
       5.88 , 5.572, 6.416, 5.859, 6.546, 6.02 , 6.315, 6.86 , 6.98 ,
       7.765, 6.144, 7.155, 6.563, 5.604, 6.153, 7.831, 6.782, 6.556,
       7.185, 6.951, 6.739, 7.178, 6.8  , 6.604, 7.875, 7.287, 7.107,
       7.274, 6.975, 7.135, 6.162, 7.61 , 7.853, 8.034, 5.891, 6.326,
       5.783, 6.064, 5.344, 5.96 , 5.404, 5.807, 6.375, 5.412, 6.182,
       5.888, 6.642, 5.951, 6.373, 6.951, 6.164, 6.879, 6.618, 8.266,
       8.725, 8.04 , 7.163, 7.686, 6.552, 5.981, 7.412, 8.337, 8.247,
       6.726, 6.086, 6.631, 7.358, 6.481, 6.606, 6.897, 6.095, 6.358,
       6.393, 5.593, 5.605, 6.108, 6.226, 6.433, 6.718, 6.487, 6.438,
       6.957, 8.259, 6.108, 5.876, 7.454, 8.704, 7.333, 6.842, 7.203,
       7.52 , 8.398, 7.327, 7.206, 5.56 , 7.014, 8.297, 7.47 , 5.92 ,
       5.856, 6.24 , 6.538, 7.691, 6.758, 6.854, 7.267, 6.826, 6.482,
       6.812, 7.82 , 6.968, 7.645, 7.923, 7.088, 6.453, 6.23 , 6.209,
       6.315, 6.565, 6.861, 7.148, 6.63 , 6.127, 6.009, 6.678, 6.549,
       5.79 , 6.345, 7.041, 6.871, 6.59 , 6.495, 6.982, 7.236, 6.616,
       7.42 , 6.849, 6.635, 5.972, 4.973, 6.122, 6.023, 6.266, 6.567,
       5.705, 5.914, 5.782, 6.382, 6.113, 6.426, 6.376, 6.041, 5.708,
       6.415, 6.431, 6.312, 6.083, 5.868, 6.333, 6.144, 5.706, 6.031,
       6.316, 6.31 , 6.037, 5.869, 5.895, 6.059, 5.985, 5.968, 7.241,
```

```
            6.54 , 6.696, 6.874, 6.014, 5.898, 6.516, 6.635, 6.939, 6.49 ,
            6.579, 5.884, 6.728, 5.663, 5.936, 6.212, 6.395, 6.127, 6.112,
            6.398, 6.251, 5.362, 5.803, 8.78 , 3.561, 4.963, 3.863, 4.97 ,
            6.683, 7.016, 6.216, 5.875, 4.906, 4.138, 7.313, 6.649, 6.794,
            6.38 , 6.223, 6.968, 6.545, 5.536, 5.52 , 4.368, 5.277, 4.652,
            5.   , 4.88 , 5.39 , 5.713, 6.051, 5.036, 6.193, 5.887, 6.471,
            6.405, 5.747, 5.453, 5.852, 5.987, 6.343, 6.404, 5.349, 5.531,
            5.683, 4.138, 5.608, 5.617, 6.852, 5.757, 6.657, 4.628, 5.155,
            4.519, 6.434, 6.782, 5.304, 5.957, 6.824, 6.411, 6.006, 5.648,
            6.103, 5.565, 5.896, 5.837, 6.202, 6.193, 6.38 , 6.348, 6.833,
            6.425, 6.436, 6.208, 6.629, 6.461, 6.152, 5.935, 5.627, 5.818,
            6.406, 6.219, 6.485, 5.854, 6.459, 6.341, 6.251, 6.185, 6.417,
            6.749, 6.655, 6.297, 7.393, 6.728, 6.525, 5.976, 5.936, 6.301,
            6.081, 6.701, 6.376, 6.317, 6.513, 6.209, 5.759, 5.952, 6.003,
            5.926, 5.713, 6.167, 6.229, 6.437, 6.98 , 5.427, 6.162, 6.484,
            5.304, 6.185, 6.229, 6.242, 6.75 , 7.061, 5.762, 5.871, 6.312,
            6.114, 5.905, 5.454, 5.414, 5.093, 5.983, 5.983, 5.707, 5.926,
            5.67 , 5.39 , 5.794, 6.019, 5.569, 6.027, 6.593, 6.12 , 6.976,
            6.794, 6.03 ])
```

```
import tensorflow as tf
import keras
from keras.layers import Dense
from keras.models import Sequential
```

```
model=Sequential()
```

```
model.add(Dense(units = 1, use_bias=True, input_dim=1, activation = "linear"))
```

```
opt = tf.keras.optimizers.Adam(learning_rate=0.1)
```

```
model.compile(loss='MSE',optimizer=opt)
```

```
model.summary()
```

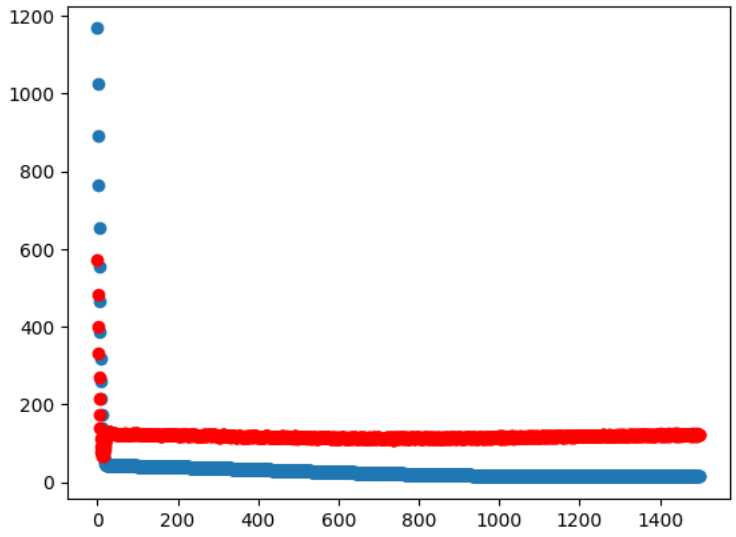
```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_7 (Dense)             (None, 1)                 2

=================================================================
Total params: 2 (8.00 Byte)
Trainable params: 2 (8.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
epochs = 1500
h = model.fit(real_rm,real_medv, verbose=1, epochs=epochs, batch_size=150, validation_split=0.3)
```

```
3/3 [==============================] - 0s 29ms/step - loss: 14.2362 - val_loss: 122.7493
Epoch 1488/1500
3/3 [==============================] - 0s 31ms/step - loss: 14.2250 - val_loss: 123.7145
Epoch 1489/1500
3/3 [==============================] - 0s 32ms/step - loss: 14.2441 - val_loss: 122.6565
Epoch 1490/1500
3/3 [==============================] - 0s 29ms/step - loss: 14.2114 - val_loss: 121.4249
Epoch 1491/1500
3/3 [==============================] - 0s 30ms/step - loss: 14.2198 - val_loss: 119.5829
Epoch 1492/1500
3/3 [==============================] - 0s 22ms/step - loss: 14.3272 - val_loss: 120.0848
Epoch 1493/1500
3/3 [==============================] - 0s 29ms/step - loss: 14.2398 - val_loss: 123.5596
Epoch 1494/1500
3/3 [==============================] - 0s 31ms/step - loss: 14.2376 - val_loss: 124.6923
Epoch 1495/1500
3/3 [==============================] - 0s 29ms/step - loss: 14.2899 - val_loss: 123.9037
Epoch 1496/1500
3/3 [==============================] - 0s 30ms/step - loss: 14.2402 - val_loss: 122.0051
Epoch 1497/1500
3/3 [==============================] - 0s 31ms/step - loss: 14.2193 - val_loss: 120.9633
Epoch 1498/1500
3/3 [==============================] - 0s 29ms/step - loss: 14.2304 - val_loss: 121.3467
Epoch 1499/1500
3/3 [==============================] - 0s 33ms/step - loss: 14.2181 - val_loss: 122.6344
Epoch 1500/1500
3/3 [==============================] - 0s 28ms/step - loss: 14.2076 - val_loss: 123.4345
```

```
#Loss = h.history['loss']
#Loss
```

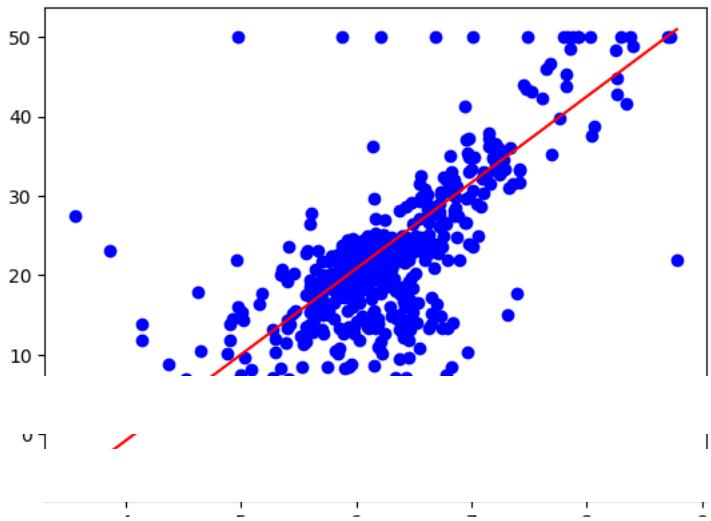
```
plt.scatter(np.arange(epochs),h.history['loss'])
plt.scatter(np.arange(epochs),h.history['val_loss'],c='r')
plt.show()
```




```
weights = model.get_weights()
print(weights[0][0][0])
print(weights[1][0])
```

```
10.832895
-44.149815
```

```
max = np.max(rm)
min = np.min(rm)
X = np.linspace(min, max, num=10)
plt.plot(X,weights[0][0][0]*X+weights[1][0],c='r')
plt.scatter(rm,medv,c="b")
plt.show()
```



Zad 6

```
model=Sequential()


model.add(Dense(units = 1, use_bias=True, input_dim=1, activation = "linear"))
model.add(Dense(units = 5, activation = "relu"))
model.add(Dense(units = 3, activation = "relu"))


opt = tf.keras.optimizers.Adam(learning_rate=0.1)


model.compile(loss='MSE',optimizer=opt)



model.summary()
```

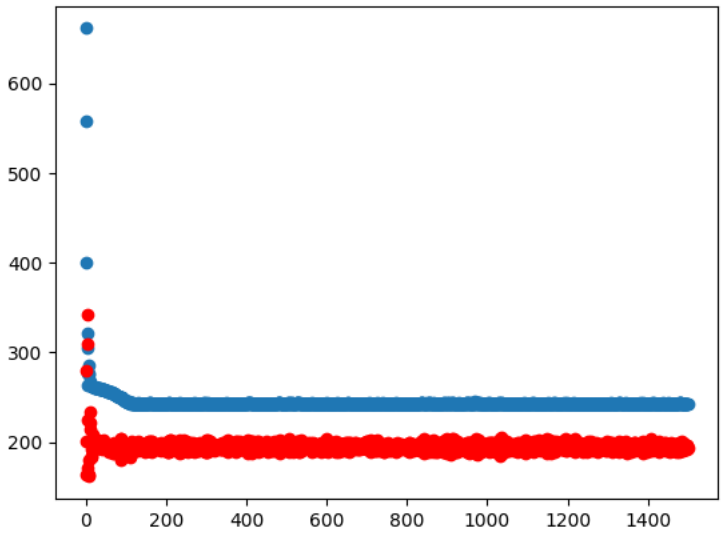
```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_8 (Dense)             (None, 1)                 2

 dense_9 (Dense)             (None, 5)                 10

 dense_10 (Dense)            (None, 3)                 18

=================================================================
Total params: 30 (120.00 Byte)
Trainable params: 30 (120.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
epochs = 1500
h = model.fit(real_rm,real_medv, verbose=1, epochs=epochs, batch_size=150, validation_split=0.3)
```

```
3/3 [==============================] - 0s 56ms/step - loss: 242.6173 - val_loss: 200.2683
Epoch 1487/1500
3/3 [==============================] - 0s 48ms/step - loss: 242.8028 - val_loss: 190.9590
Epoch 1488/1500
3/3 [==============================] - 0s 39ms/step - loss: 242.5109 - val_loss: 199.7923
Epoch 1489/1500
3/3 [==============================] - 0s 38ms/step - loss: 242.7168 - val_loss: 191.7453
Epoch 1490/1500
3/3 [==============================] - 0s 41ms/step - loss: 242.4712 - val_loss: 196.8964
Epoch 1491/1500
3/3 [==============================] - 0s 39ms/step - loss: 242.3704 - val_loss: 191.3038
Epoch 1492/1500
3/3 [==============================] - 0s 38ms/step - loss: 242.5006 - val_loss: 197.6787
Epoch 1493/1500
3/3 [==============================] - 0s 36ms/step - loss: 242.4430 - val_loss: 192.2410
Epoch 1494/1500
3/3 [==============================] - 0s 52ms/step - loss: 242.4415 - val_loss: 195.4892
Epoch 1495/1500
3/3 [==============================] - 0s 35ms/step - loss: 242.3931 - val_loss: 193.5282
Epoch 1496/1500
3/3 [==============================] - 0s 43ms/step - loss: 242.5252 - val_loss: 191.4294
Epoch 1497/1500
3/3 [==============================] - 0s 39ms/step - loss: 242.6077 - val_loss: 196.5936
Epoch 1498/1500
3/3 [==============================] - 0s 40ms/step - loss: 242.5193 - val_loss: 191.6232
Epoch 1499/1500
3/3 [==============================] - 0s 37ms/step - loss: 242.7334 - val_loss: 195.7521
Epoch 1500/1500
3/3 [==============================] - 0s 30ms/step - loss: 242.4809 - val_loss: 193.8521
```

```
#Loss = h.history['loss']
#Loss
```

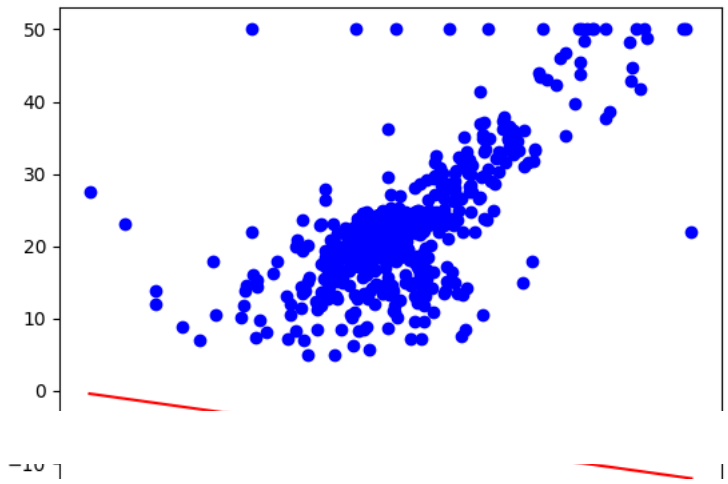
```
plt.scatter(np.arange(epochs),h.history['loss'])
plt.scatter(np.arange(epochs),h.history['val_loss'],c='r')
plt.show()
```




```
weights = model.get_weights()
print(weights[0][0][0])
print(weights[1][0])
```

```
-2.2499645
7.576745
```

```
max = np.max(rm)
min = np.min(rm)
X = np.linspace(min, max, num=10)
plt.plot(X,weights[0][0][0]*X+weights[1][0],c='r')
plt.scatter(rm,medv,c="b")
plt.show()
```




```
model=Sequential()


model.add(Dense(units = 1, use_bias=True, input_dim=1, activation = "linear"))
model.add(Dense(units = 5, activation = "relu"))
model.add(Dense(units = 3, activation = "relu"))


opt = tf.keras.optimizers.Adam(learning_rate=0.1)


model.compile(loss='BinaryCrossentropy',optimizer=opt)



model.summary()
```

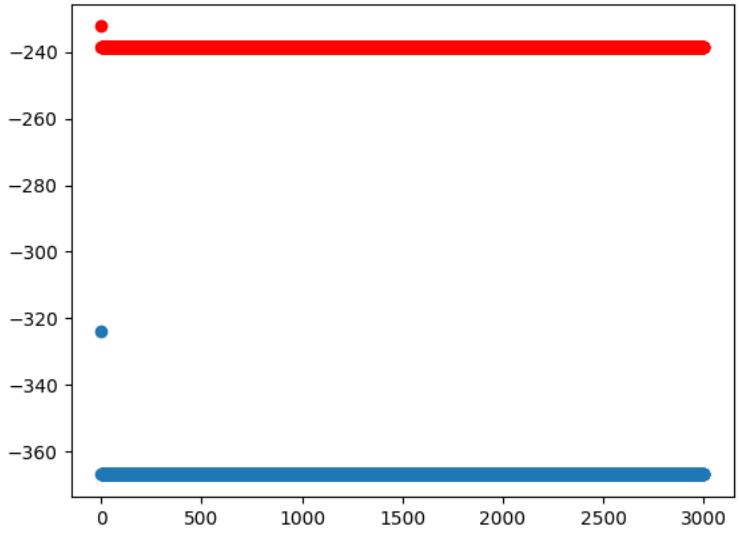
```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_11 (Dense)            (None, 1)                 2

 dense_12 (Dense)            (None, 5)                 10

 dense_13 (Dense)            (None, 3)                 18

=================================================================
Total params: 30 (120.00 Byte)
Trainable params: 30 (120.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
epochs = 3000
h = model.fit(real_rm,real_medv, verbose=1, epochs=epochs, batch_size=150, validation_split=0.3)
```

```
3/3 [==============================] - 0s 38ms/step - loss: -366.8347 - val_loss: -238.7509
Epoch 2988/3000
3/3 [==============================] - 0s 34ms/step - loss: -366.8346 - val_loss: -238.7509
Epoch 2989/3000
3/3 [==============================] - 0s 29ms/step - loss: -366.8347 - val_loss: -238.7509
Epoch 2990/3000
3/3 [==============================] - 0s 39ms/step - loss: -366.8346 - val_loss: -238.7509
Epoch 2991/3000
3/3 [==============================] - 0s 34ms/step - loss: -366.8346 - val_loss: -238.7509
Epoch 2992/3000
3/3 [==============================] - 0s 31ms/step - loss: -366.8346 - val_loss: -238.7509
Epoch 2993/3000
3/3 [==============================] - 0s 43ms/step - loss: -366.8346 - val_loss: -238.7509
Epoch 2994/3000
3/3 [==============================] - 0s 33ms/step - loss: -366.8346 - val_loss: -238.7509
Epoch 2995/3000
3/3 [==============================] - 0s 34ms/step - loss: -366.8346 - val_loss: -238.7509
Epoch 2996/3000
3/3 [==============================] - 0s 38ms/step - loss: -366.8346 - val_loss: -238.7509
Epoch 2997/3000
3/3 [==============================] - 0s 37ms/step - loss: -366.8346 - val_loss: -238.7509
Epoch 2998/3000
3/3 [==============================] - 0s 29ms/step - loss: -366.8346 - val_loss: -238.7509
Epoch 2999/3000
3/3 [==============================] - 0s 34ms/step - loss: -366.8346 - val_loss: -238.7509
Epoch 3000/3000
3/3 [==============================] - 0s 31ms/step - loss: -366.8346 - val_loss: -238.7509
```

```
#Loss = h.history['loss']
#Loss
```

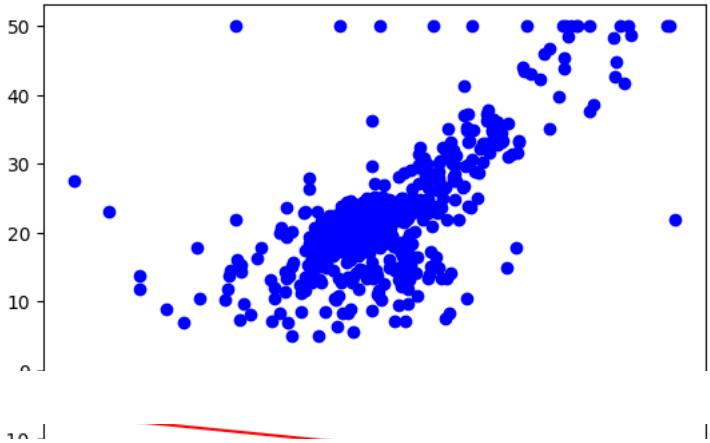
```
plt.scatter(np.arange(epochs),h.history['loss'])
plt.scatter(np.arange(epochs),h.history['val_loss'],c='r')
plt.show()
```




```
weights = model.get_weights()
print(weights[0][0][0])
print(weights[1][0])
```

```
-1.672512
-0.6176705
```

```
max = np.max(rm)
min = np.min(rm)
X = np.linspace(min, max, num=10)
plt.plot(X,weights[0][0][0]*X+weights[1][0],c='r')
plt.scatter(rm,medv,c="b")
plt.show()
```



Sieć z zadania 6 osiąga, DUŻO gorsze wyniki niż sieć zadania z 5, nie uczy się albo uczy się tylko na początku potem już nie (model jest zbyt rozbudowany)