# Laboratorijska vježba 5

Cilj vježbe je upoznavanje sa drugim tipom nadziranog učenja - regresijom. Studenti se upoznaju sa razlikama pri dizajniranju arhitekture neuronske mreže za regresiju u odnosu na klasifikaciju te rješavaju jednostavan zadatak regresije koristeći Keras razvojni okvir.

# Zadaci za rad u laboratoriji

## Zadatak 1 - Predviđanje cijene kuća u Boston

U ovom zadatku cilj je izvršiti predikciju cijene prilikom kupovine kuće u Bostonu na bazi podataka iz 1970. godine. Za potrebe vježbe koristiti će se historijski podaci, a oni uključuju informacije kao što je nivo kriminala u regiji, lokalni porez, itd. Ciljne (eng. target) vrijednosti su izražene u hiljadama dolara.

a) Učitati The Boston Housing Price skup podataka. Skup podataka je dio Keras biblioteke te se može učitati sljedećim kodom:

```
from keras.datasets import boston_housing
(train_data, train_targets), (test_data, test_targets) = boston_housing.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/bostor
57344/57026 [==============================] - 0s 0us/step
65536/57026 [==============================] - 0s 0us/step
```

b) Koji je oblik podataka za treniranje, a koji za testiranje? Koliko ima ukupno atributa (značajki)?

```
print("Oblik podataka za treniranje je ",train_data.shape)
print("Oblik podataka za testiranje je ",test_data.shape)
print("Ukupno atributa (znacajki) je ", train_data.shape[1])

```

```
Oblik podataka za treniranje je  (404, 13)
Oblik podataka za testiranje je  (102, 13)
Ukupno atributa (znacajki) je  13
```

c) Koji je opseg vrijednosti (min i max ) za svaki atribut u skupu podataka? A koji je opseg vrijednosti za ciljne (eng. target) vrijednosti?

```
import pandas as pd
data_frame_train=pd.DataFrame(train_data)
data_frame_test=pd.DataFrame(test_data)
data_frame_targets_train=pd.DataFrame(train_targets)
data_frame_targets_test=pd.DataFrame(test_targets)
data_frame=pd.concat([data_frame_train, data_frame_test,], axis=0)
data_frame_targets=pd.concat([data_frame_targets_train, data_frame_targets_test,], axis=1)

print("Opseg vrijednosti za svaki atribut u skupu podataka: ")
data_frame.describe()
```

```
Opseg vrijednosti za svaki atribut u skupu podataka:
```

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| **count** | 506.000000 | 506.000000 | 506.000000 | 506.000000 | 506.000000 | 506.000000 | 506.000000 |
| **mean** | 3.613524 | 11.363636 | 11.136779 | 0.069170 | 0.554695 | 6.284634 | 68.574901 |
| **std** | 8.601545 | 23.322453 | 6.860353 | 0.253994 | 0.115878 | 0.702617 | 28.148861 |
| **min** | 0.006320 | 0.000000 | 0.460000 | 0.000000 | 0.385000 | 3.561000 | 2.900000 |
| **25%** | 0.082045 | 0.000000 | 5.190000 | 0.000000 | 0.449000 | 5.885500 | 45.025000 |
| **50%** | 0.256510 | 0.000000 | 9.690000 | 0.000000 | 0.538000 | 6.208500 | 77.500000 |
| **75%** | 3.677083 | 12.500000 | 18.100000 | 0.000000 | 0.624000 | 6.623500 | 94.075000 |
| **max** | 88.976200 | 100.000000 | 27.740000 | 1.000000 | 0.871000 | 8.780000 | 100.000000 |

```
print("Opseg vrijednosti za ciljne (eng.target) vrijednosti: ")
data_frame_targets.describe()
```

```
Opseg vrijednosti za ciljne (eng.target) vrijednosti:
```

d) S obzirom da podaci zauzimaju široke spektre vrijednosti, izvršiti skaliranje korištenjem MinMaxScaler-a. Obavezno povesti računa da se parametri scaler-a za skaliranje izračunaju nad trening skupom podataka, a da se skaliranje izvrši i nad trening i nad test skupom.

```
      std      9.210442     9.168863
```

```
1 from sklearn.preprocessing import MinMaxScaler
2 scaler = MinMaxScaler().fit(train_data)
3 train_data = scaler.fit_transform(train_data)
4 test_data = scaler.fit_transform(test_data)
5
```

```
      75%     24.800000    27.075000
```

e) Napisati pomoćnu funkciju def model_mreze() koja će vraćati model mreže (return model) opisan u nastavku. Model treba da ima 2 skrivena Dense sloja sa po 64 neurona sa relu aktivacijskim funkcijama. Dimenzije ulaznog sloja odrediti na osnovu broja atributa skupa podataka. Treći, koji je i posljednji sloj, treba imati samo jedan neuron i ne treba imati aktivacijsku funkciju.

```
1 from keras import models
2 from keras import layers
3 def model_mreze():
4   n=train_data.shape[1]
5   model = models.Sequential()
6   model.add(layers.Dense(64, activation='relu', input_shape=(n,)))
7   model.add(layers.Dense(64, activation='relu'))
8   model.add(layers.Dense(1))
9   return model
10
```

f) Kompajlirati model da koristi mse funkciju gubitka, adam optimizator i mae metriku. S obzirom da su ciljne vijednosti izražene u hiljadama dolara, metrika MAE govori koliko model griješi u hiljadama dolara. Na primjer, ukoliko je vrijednost MAE = 4.2, to znači da predviđanja modela odstupaju u prosjeku za $4200;

```
1 model=model_mreze()
2 model.compile(optimizer='adam',loss='mse',metrics=['mae'])
```

g) Izvršiti treniranje modela na 100 epoha, sa veličinom batch-a jednakoj 1. Koristiti 10% trening skupa za validaciju. Kolika je postignuta vrijednost funkcije gubitka na kraju treniranja, a kolika je vrijednost metrike MAE? Grafički prikazati.

```
1 history = model.fit(train_data, train_targets, epochs=100, batch_size=1, validation_split
```

```
history = model.fit(train_data, train_targets, epochs=100, batch_size=1, validation_split
363/363 [==============================] - 1s 2ms/step - loss: 6.8203 - mae: 1.8786 -
Epoch 65/100
363/363 [==============================] - 1s 2ms/step - loss: 7.1683 - mae: 1.9135 -
Epoch 66/100
363/363 [==============================] - 1s 2ms/step - loss: 7.8459 - mae: 1.9563 -
Epoch 67/100
363/363 [==============================] - 1s 2ms/step - loss: 7.1506 - mae: 1.9242 -
Epoch 68/100
363/363 [==============================] - 1s 2ms/step - loss: 7.4588 - mae: 1.8960 -
Epoch 69/100
363/363 [==============================] - 1s 2ms/step - loss: 6.9041 - mae: 1.8747 -
Epoch 70/100
363/363 [==============================] - 1s 2ms/step - loss: 6.5759 - mae: 1.8735 -
Epoch 71/100
363/363 [==============================] - 1s 2ms/step - loss: 7.2828 - mae: 1.8884 -
Epoch 72/100
363/363 [==============================] - 1s 2ms/step - loss: 6.6212 - mae: 1.8617 -
Epoch 73/100
363/363 [==============================] - 1s 2ms/step - loss: 6.7225 - mae: 1.8767 -
Epoch 74/100
363/363 [==============================] - 1s 2ms/step - loss: 6.1475 - mae: 1.8265 -
Epoch 75/100
363/363 [==============================] - 1s 2ms/step - loss: 6.3386 - mae: 1.8427 -
Epoch 76/100
363/363 [==============================] - 1s 2ms/step - loss: 6.5288 - mae: 1.8153 -
Epoch 77/100
363/363 [==============================] - 1s 2ms/step - loss: 6.5832 - mae: 1.7865 -
Epoch 78/100
363/363 [==============================] - 1s 2ms/step - loss: 6.3756 - mae: 1.8237 -
Epoch 79/100
363/363 [==============================] - 1s 2ms/step - loss: 6.5167 - mae: 1.8439 -
Epoch 80/100
363/363 [==============================] - 1s 2ms/step - loss: 6.6368 - mae: 1.8926 -
Epoch 81/100
363/363 [==============================] - 1s 2ms/step - loss: 6.3227 - mae: 1.8250 -
Epoch 82/100
363/363 [==============================] - 1s 2ms/step - loss: 6.3219 - mae: 1.8299 -
Epoch 83/100
363/363 [==============================] - 1s 2ms/step - loss: 6.2133 - mae: 1.8161 -
Epoch 84/100
363/363 [==============================] - 1s 2ms/step - loss: 6.1364 - mae: 1.8202 -
Epoch 85/100
363/363 [==============================] - 1s 2ms/step - loss: 5.9935 - mae: 1.7840 -
Epoch 86/100
363/363 [==============================] - 1s 2ms/step - loss: 6.7375 - mae: 1.8629 -
Epoch 87/100
363/363 [==============================] - 1s 2ms/step - loss: 5.8862 - mae: 1.7537 -
Epoch 88/100
363/363 [==============================] - 1s 2ms/step - loss: 5.7004 - mae: 1.7974 -
Epoch 89/100
363/363 [==============================] - 1s 2ms/step - loss: 5.7809 - mae: 1.7383 -
Epoch 90/100
363/363 [==============================] - 1s 2ms/step - loss: 5.6621 - mae: 1.7387 -
Epoch 91/100
363/363 [==============================] - 1s 2ms/step - loss: 5.7739 - mae: 1.7595 -
```

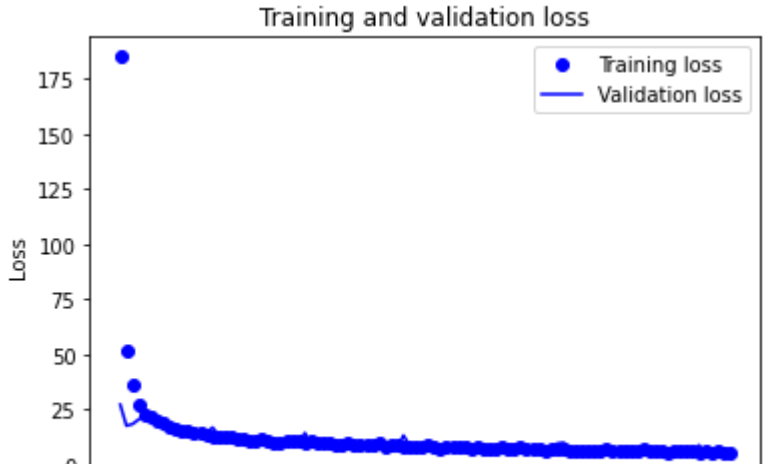
```
Epoch 92/100
363/363 [==============================] - 1s 2ms/step - loss: 5.9312 - mae: 1.7863 -
```

```
import math
import matplotlib.pyplot as plt

#citanje potrebnih vrijednosti iz history objekta
acc = history.history['mae']
loss_values = history.history['loss']
val_loss_values = history.history['val_loss']
epochs = range(1, len(acc) + 1)

#pripremanje za crtanje grafa
plt.plot(epochs, loss_values, 'bo', label='Training loss')
plt.plot(epochs, val_loss_values, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
plt.clf()

#citanje potrebnih vrijednosti iz history objekta
acc_values = history.history['mae']
val_acc_values = history.history['val_mae']
plt.plot(epochs, acc_values, 'bo', label='Training mae')
plt.plot(epochs, val_acc_values, 'b', label='Validation mae')
plt.title('Training and validation mae')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
```




h) U prethodnom grafičkom prikazu problem predstavlja skala i visoka varijansa nad validacijskim skupom po- dataka. Iz tog razloga je korisno prikazati usrednjene vrijednosti i zanemariti prvih 10 uzoraka radi visoke skale u početnim epohama. Ovakav prikaz nad validacijskim skupom podataka se može postići sljedećim kodom:

```
def smooth_curve(points, factor=0.9):
  smoothed_points = []
  for point in points:
    if smoothed_points:
       previous = smoothed_points[-1]
       smoothed_points.append(previous * factor + point * (1 - factor))
    else:
        smoothed_points.append(point)
  return smoothed_points
mae_history = history.history['val_mae']

smooth_mae_history = smooth_curve(mae_history[10:])
plt.plot(range(1, len(smooth_mae_history) + 1), smooth_mae_history)
plt.xlabel('Epochs')
plt.ylabel('Validation MAE')
plt.show()

```

i) Ponovo definisati model i istrenirati ga na 500 epoha. Ostale parametre ostaviti nepromijenjenim. Izvršiti grafički prikaz usrednjenih vrijednosti kao u prethodnom zadatku. Uporediti ova dva grafička prikaza. Šta se može zaključiti iz ovih grafika?

```
model=model_mreze()
model.compile(optimizer='adam',loss='mse',metrics=['mae'])

#stavljamo treniranje na 500 epoha
history = model.fit(train_data, train_targets, epochs=500, batch_size=1, validation_split
```

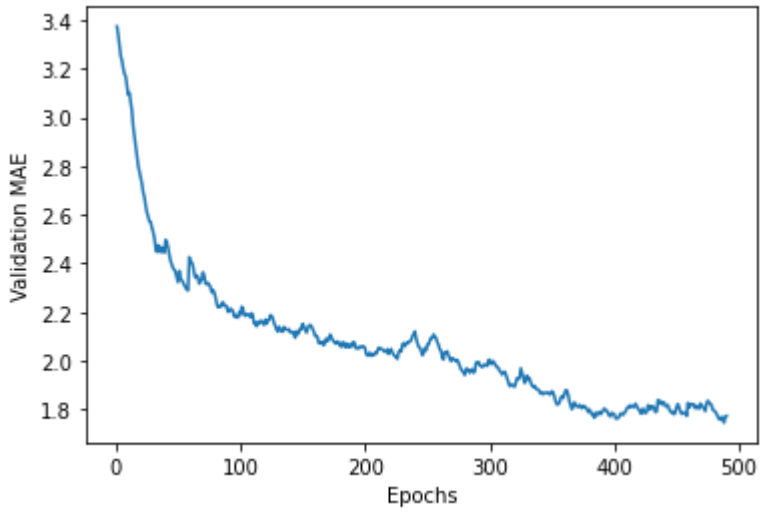
```
363/363 [==============================] - 1s 2ms/step - loss: 3.1302 - mae: 1.2529 -
Epoch 376/500
363/363 [==============================] - 1s 2ms/step - loss: 1.8477 - mae: 1.0045 -
Epoch 377/500
363/363 [==============================] - 1s 2ms/step - loss: 1.6620 - mae: 0.9580 -
Epoch 378/500
363/363 [==============================] - 1s 2ms/step - loss: 1.8920 - mae: 1.0282 -
Epoch 379/500
363/363 [==============================] - 1s 2ms/step - loss: 2.5112 - mae: 1.1283 -
Epoch 380/500
363/363 [==============================] - 1s 2ms/step - loss: 1.7905 - mae: 1.0142 -
Epoch 381/500
363/363 [==============================] - 1s 2ms/step - loss: 1.7633 - mae: 0.9809 -
Epoch 382/500
363/363 [==============================] - 1s 2ms/step - loss: 1.9399 - mae: 1.0383 -
Epoch 383/500
363/363 [==============================] - 1s 2ms/step - loss: 1.9300 - mae: 1.0388 -
Epoch 384/500
363/363 [==============================] - 1s 2ms/step - loss: 1.8418 - mae: 1.0350 -
Epoch 385/500
363/363 [==============================] - 1s 2ms/step - loss: 2.2120 - mae: 1.0826 -
Epoch 386/500
363/363 [==============================] - 1s 2ms/step - loss: 1.8456 - mae: 1.0112 -
Epoch 387/500
363/363 [==============================] - 1s 2ms/step - loss: 2.0762 - mae: 1.0495 -
Epoch 388/500
363/363 [==============================] - 1s 2ms/step - loss: 1.8560 - mae: 0.9784 -
Epoch 389/500
363/363 [==============================] - 1s 2ms/step - loss: 1.8240 - mae: 0.9901 -
Epoch 390/500
363/363 [==============================] - 1s 2ms/step - loss: 2.2807 - mae: 1.1413 -
Epoch 391/500
363/363 [==============================] - 1s 2ms/step - loss: 1.8281 - mae: 0.9985 -
Epoch 392/500
363/363 [==============================] - 1s 2ms/step - loss: 1.6987 - mae: 1.0003 -
Epoch 393/500
363/363 [==============================] - 1s 2ms/step - loss: 1.7686 - mae: 0.9500 -
Epoch 394/500
363/363 [==============================] - 1s 2ms/step - loss: 1.7932 - mae: 0.9892 -
Epoch 395/500
363/363 [==============================] - 1s 2ms/step - loss: 1.7343 - mae: 1.0055 -
```

```
Epoch 396/500
363/363 [==============================] - 1s 2ms/step - loss: 2.0742 - mae: 1.0602 -
Epoch 397/500
363/363 [==============================] - 1s 2ms/step - loss: 2.1038 - mae: 1.0684 -
Epoch 398/500
363/363 [==============================] - 1s 2ms/step - loss: 1.8119 - mae: 1.0056 -
Epoch 399/500
363/363 [==============================] - 1s 2ms/step - loss: 1.6924 - mae: 0.9951 -
Epoch 400/500
363/363 [==============================] - 1s 2ms/step - loss: 2.2882 - mae: 1.1399 -
Epoch 401/500
363/363 [==============================] - 1s 2ms/step - loss: 1.6688 - mae: 0.9540 -
Epoch 402/500
363/363 [==============================] - 1s 2ms/step - loss: 1.7684 - mae: 0.9866 -
Epoch 403/500
363/363 [==============================] - 1s 2ms/step - loss: 1.7311 - mae: 0.9925 -
```

+ Code + Text

```
def smooth_curve(points, factor=0.9):
  smoothed_points = []
  for point in points:
    if smoothed_points:
       previous = smoothed_points[-1]
       smoothed_points.append(previous * factor + point * (1 - factor))
    else:
        smoothed_points.append(point)
  return smoothed_points
mae_history = history.history['val_mae']

smooth_mae_history = smooth_curve(mae_history[10:])
plt.plot(range(1, len(smooth_mae_history) + 1), smooth_mae_history)
plt.xlabel('Epochs')
plt.ylabel('Validation MAE')
plt.show()
```




j) Izvršiti evaluaciju modela nad testnim skupom podataka. Koja je postignuta vrijednost funkcije gubitka, a koja vrijednost metrike MAE? Koliko prosječno u dolarima griješi model u svojim

predikcijama?

```
1 results = model.evaluate(test_data,test_targets)
```

```
4/4 [==============================] - 0s 3ms/step - loss: 33.4500 - mae: 4.8708
```

```
1 import math
2 print("Funkcija gubitka: ", results[0])
3 print("Vrijednost metrike MAE: ", results[1])
4 print("Model prosječno griješi {} u dolarima".format(math.floor(1000000*results[1])/1000.0
```

```
Funkcija gubitka:  33.44998550415039
Vrijednost metrike MAE:  4.870757579803467
Model prosječno griješi 4870.757 u dolarima
```

✓ 0s completed at 2:35 AM