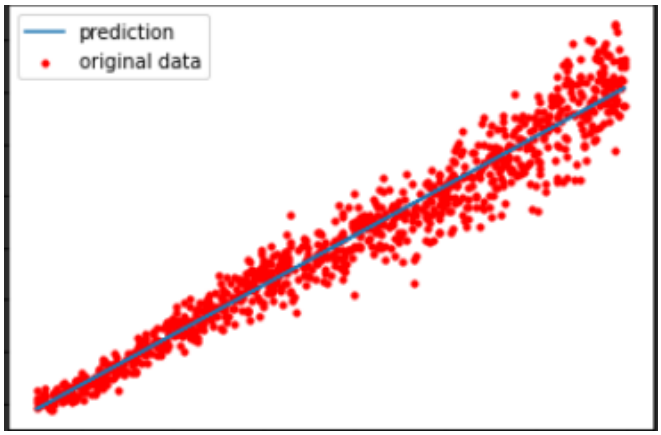
maskinlæring 2021

sigmund Granaas

September 2021

1 Oppgave 1



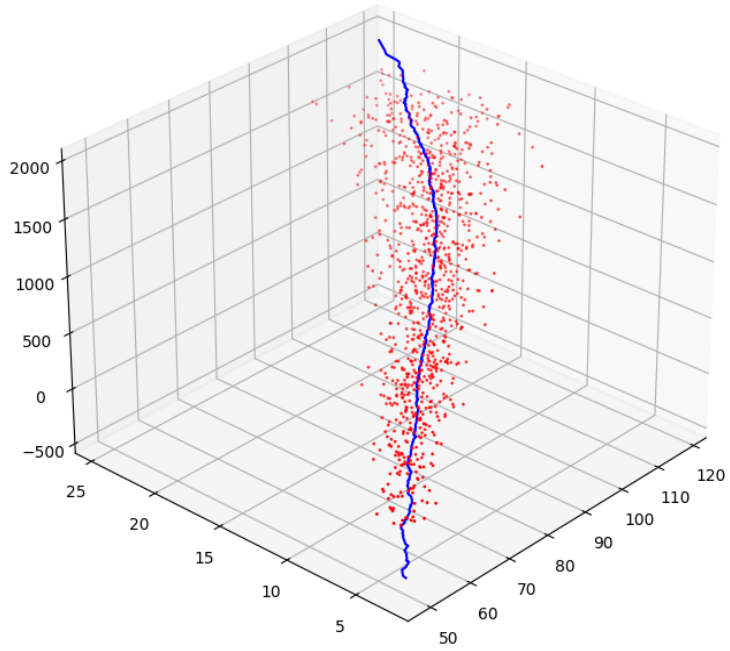
Code:

```
import torch
torch.set_default_dtype(torch.float64)
# Training the model
x_train = torch.reshape(torch.tensor(training_data_x), (-1, 1))
y_train = torch.reshape(torch.tensor(training_data_y), (-1, 1))

w1 = torch.rand(1, 1, requires_grad=True)
b1 = torch.rand(1, requires_grad=True)
lr = 0.0001
epochs = 600000
stepping = 10000

#%%

# Training the model with with manual optimizer

for i in range(1, epochs):
```

```
    y_pred = ((x_train @ w1) + b1)

    loss = (((y_pred - y_train).pow(2).sum())/training_data_y.size)


    if i % stepping == 0:
        print ("iteration: ", i, "loss: ", loss.item())

    loss.backward()

    with torch.no_grad():
        w1 -= lr * w1.grad
        b1 -= lr * b1.grad
        w1.grad.zero_()
        b1.grad.zero_()

print("trained w1: ",w1)
print("trained b1: ",b1)
```

trained W: 0.2384 trained b: -8.5137

2 Oppgave 2



Jeg klarte ikke lage en helt linær

Code:

```
import pandas as pd
%matplotlib qt
import matplotlib.pyplot as plt
# Importing and converting the data
training_data = pd.read_csv('./data2.csv')
training_data_xy = training_data[['length', 'weight']].to_numpy()
training_data_x = training_data['length'].to_numpy()
training_data_y = training_data['weight'].to_numpy()
training_data_z = training_data['day'].to_numpy()

#%%

import torch
```

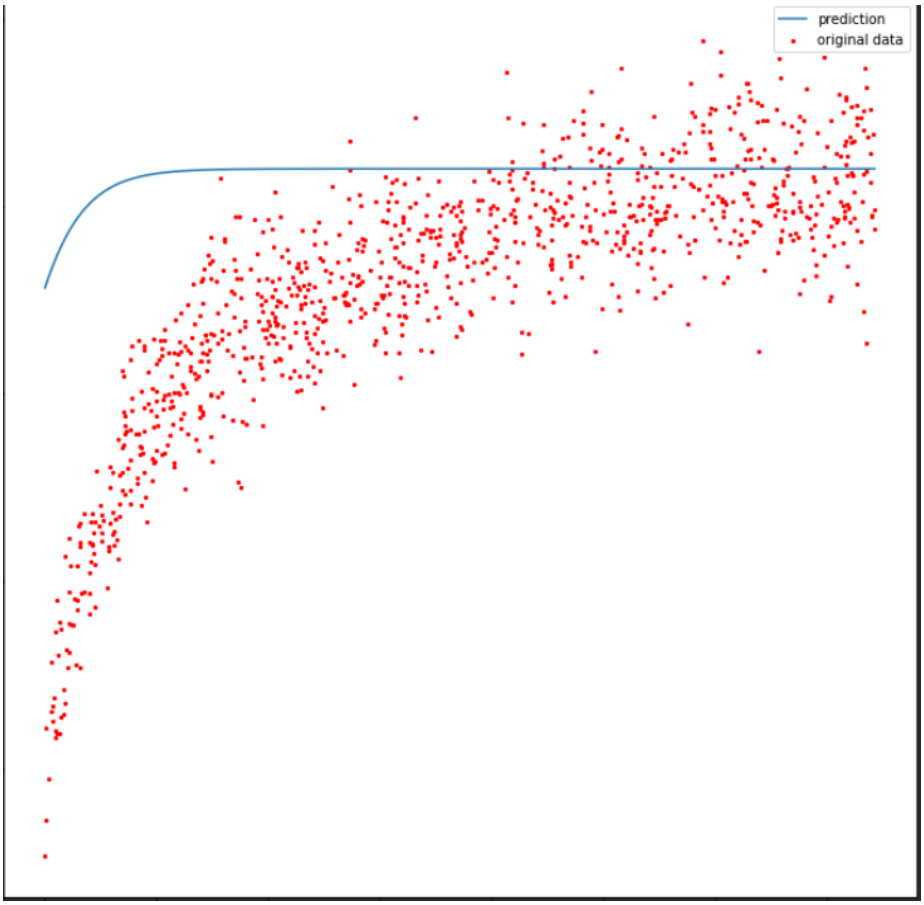
```
torch.set_default_dtype(torch.float64)
# Training the model
xy_train = torch.reshape(torch.tensor(training_data_xy), (-1, 2))
z_train = torch.reshape(torch.tensor(training_data_z), (-1, 1))

w1 = torch.rand((2, 1), requires_grad=True)
b1 = torch.rand(1, requires_grad=True)

lr = 0.0001
epochs = 1000000
stepping = 10000

# Training the model with with manual optimizer

for i in range(1, epochs):
    z_pred = ((xy_train @ w1) + b1)

    loss = (((z_pred - z_train).pow(2).sum())/training_data_z.size)


    if i % stepping == 0:
        print ("iteration: ", i, "loss: ", loss.item())

    loss.backward()

    with torch.no_grad():
        w1 -= lr * w1.grad
        b1 -= lr * b1.grad
        w1.grad.zero_()
        b1.grad.zero_()

print("trained w1: ",w1)
print("trained b1: ",b1)

#%%

import numpy as np
from scipy import interpolate

prediction = ((xy_train @ w1) + b1).detach().numpy().flatten()
prediction.sort()



sortedx = training_data_x.copy()
```

```
sortedy = training_data_y.copy()

sortedx.sort()
sortedy.sort()


plt.figure(figsize=(8, 8))
ax = plt.axes(projection='3d')
ax.plot3D(sortedx, sortedy, prediction, label='prediction', color='blue')
ax.scatter3D(training_data_x, training_data_y, training_data_z, c='red', s=1, la
plt.show()
```

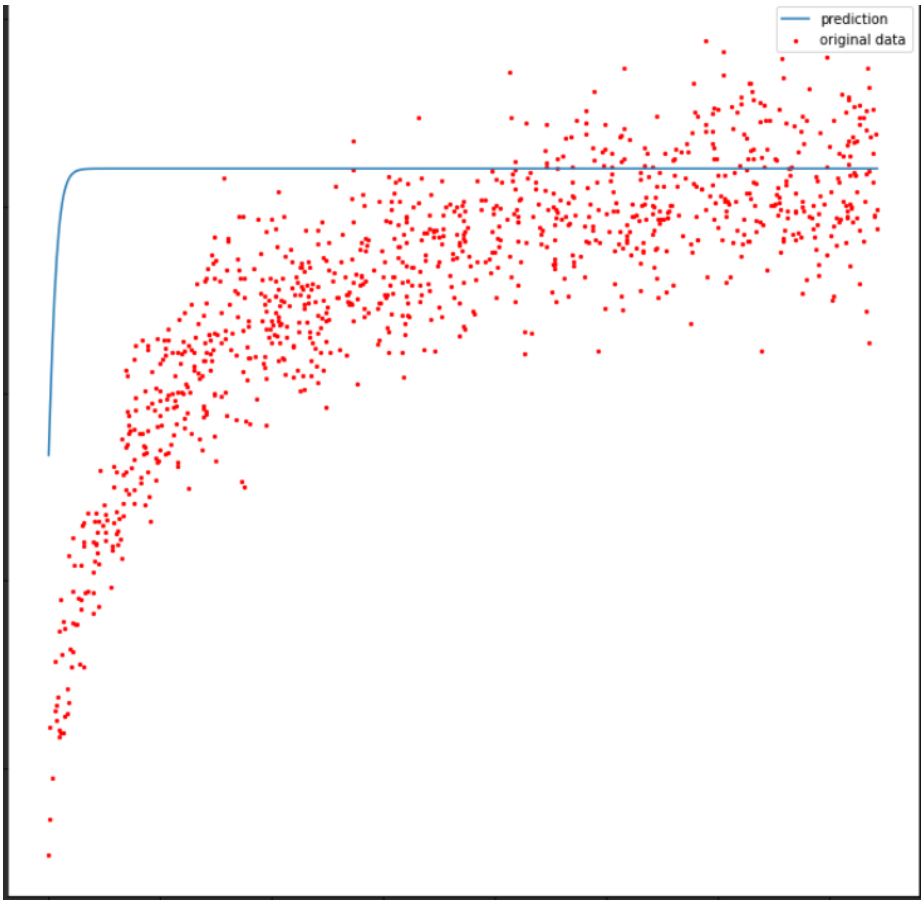
trained w: [[26.4541], [30.8922]] trained b: [-1845.7401]

3 Oppgave 3

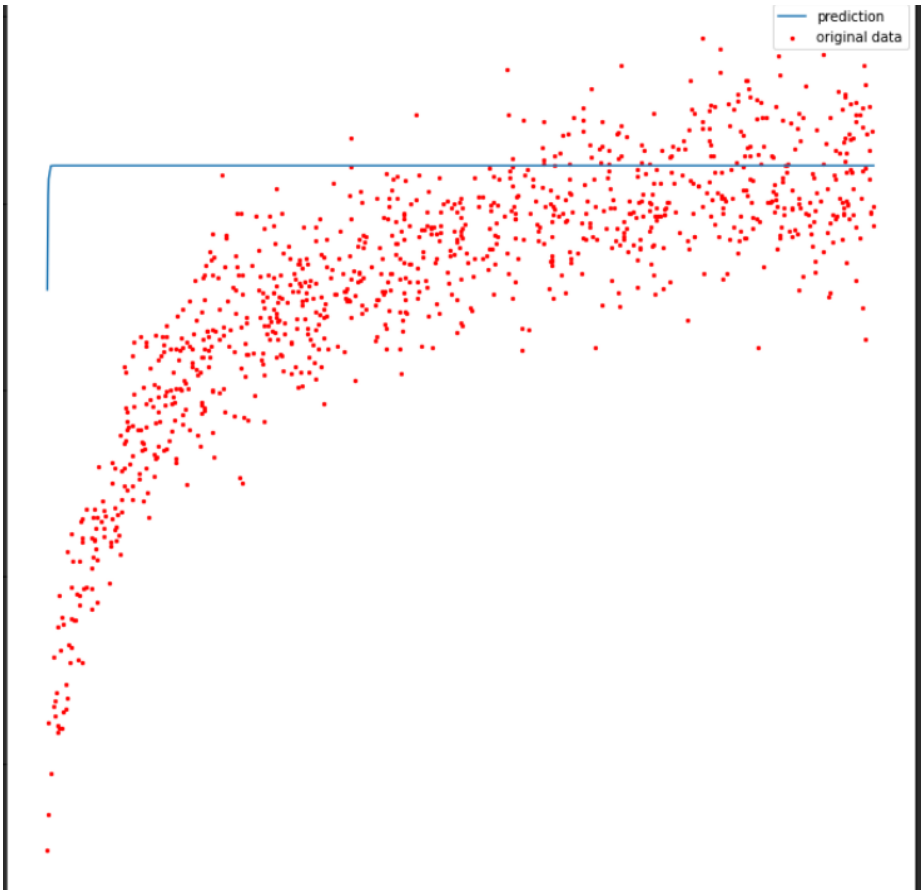
Jeg var usikker på hvordan vi skulle analysere dataen, jeg endte opp med å dele x-dataen på 100 for å få det til å passe med sigmoidfunksjonen, hvis ikke vil den bare returnere 1 på alle inputs. x-verdiene, vil da gå fra 0-17



Delte x-inputs på 100 trained w1: [[1.4101]] trained b1: [1.5314]



Delte x-inputs på 10 trained w1: [[0.7067]] trained b1: [0.9117]



ikke delt inputs trained w1: [[1.0070]] trained b1: [0.2965]

#%%

```
import pandas as pd
from torch import exp
import numpy as np
import itertools
import matplotlib.pyplot as plt
# Importing and converting the data
training_data = pd.read_csv('./data3.csv')
training_data_x = training_data['day'].to_numpy()/100
training_data_y = training_data['head'].to_numpy()
training_data_x_unsorted = training_data_x.copy()
training_data_y_unsorted = training_data_y.copy()

training_data_x.sort()
training_data_y.sort()
```

```
#%%

import matplotlib.pyplot as plt

# Plotting the data
plt.figure(figsize=(12, 12))
plt.scatter(training_data_x_unsorted,training_data_y_unsorted , c='red', s=10, l
plt.show()

#%%


import torch
torch.set_default_dtype(torch.float64)
# Training the model
x_train = torch.reshape(torch.tensor(training_data_x), (-1, 1))
y_train = torch.reshape(torch.tensor(training_data_y), (-1, 1))

w1 = torch.rand(1, 1, requires_grad=True)
b1 = torch.rand(1, requires_grad=True)
lr = 0.001
epochs = 6000
stepping = 1000



#%%

def sigmoid(z):
    sig = 1.0/(1.0 + exp(-z))
    return sig

def prediction(z):
    return 20 * z + 31
# Training the model with with manual optimizer

sig = torch.nn.Sigmoid()
optimizer = torch.optim.SGD([w1, b1], lr=0.01)
criterion = torch.nn.BCELoss()

for i in range(1, epochs):
    linear_model = x_train.mm(w1).add(b1)
    y_pred = sigmoid(linear_model)

    loss = criterion(y_pred, sigmoid(y_train))
```

```
    #loss = (((prediction(y_pred) - prediction(y_train)).pow(2).sum())/training_
    if i % stepping == 0:
        print ("iteration: ", i, "loss: ", loss.item())

    loss.backward()

    optimizer.step()
    optimizer.zero_grad()

print("trained w1: ",w1)
print("trained b1: ",b1)

#%%

predicted_tensor = (20 * sigmoid(x_train.mm(w1).add(b1)) + 31).flatten().detach(
plt.figure(figsize=(12, 12))

plt.scatter(training_data_x_unsorted, training_data_y_unsorted, c='red', s=5, la
plt.plot(x_train, predicted_tensor, label='prediction')
plt.xlabel('days')
plt.ylabel('head circumference')
plt.legend()

plt.show()
```