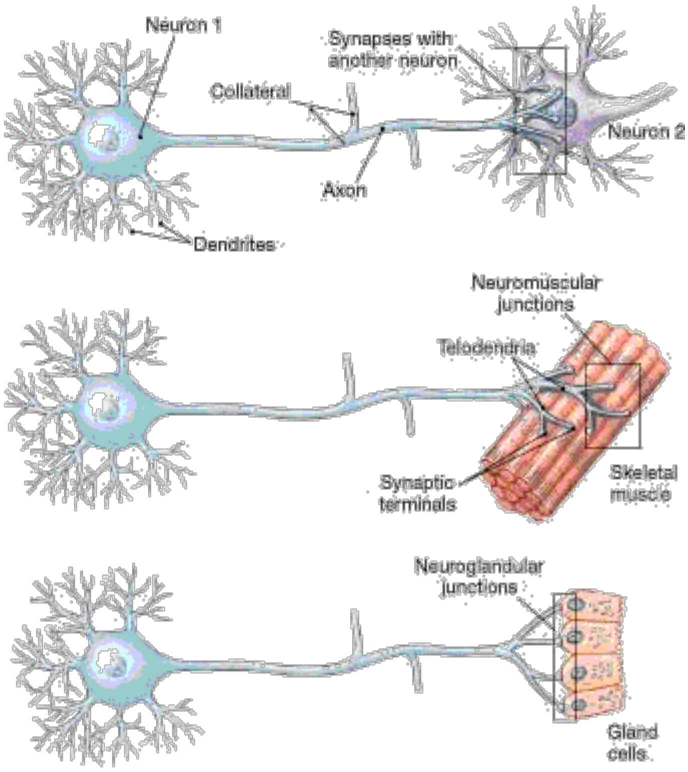
Neural Networks

Neural network learning was originally inspired by neurons in human (mammal) brains



Linear threshold unit

This first model was introduced by McCulloch and Pitts in 1943. The idea was to establish logical processing on ones (True) and zeroes (False) similar to logical computation machines back in that time.

$$y = \begin{cases} 1, \text{ if } w_0 + w_1 \cdot x_1 + w_2 \cdot x_2 \geq \theta \\ 0, \text{ if } w_0 + w_1 \cdot x_1 + w_2 \cdot x_2 < \theta \end{cases} . \tag{1}$$

Perceptron (Päättelin)

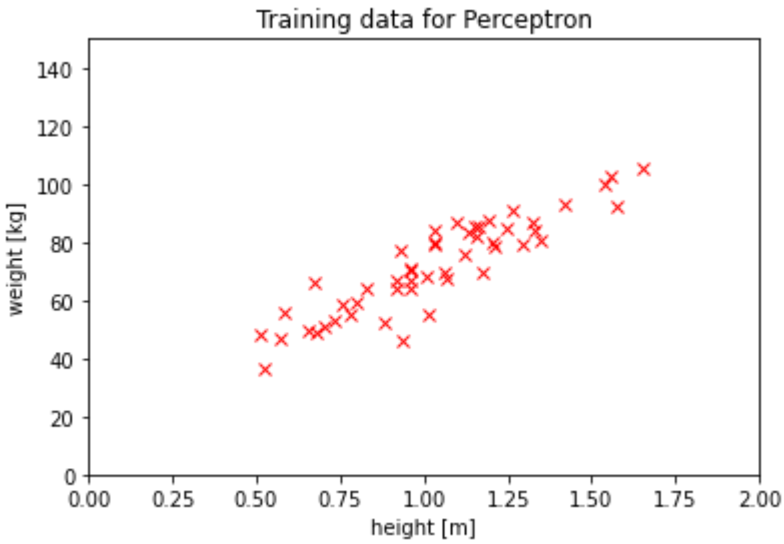
Perceptron added a training algorithm based on the so called *Hebbian rule*:

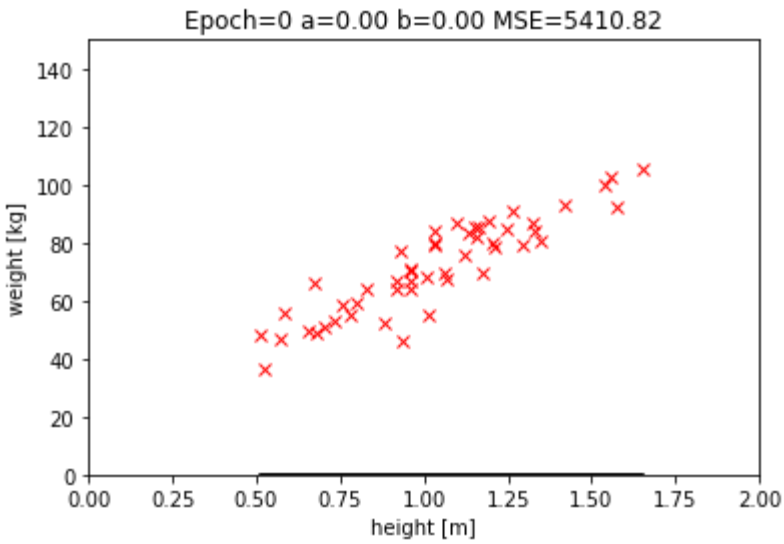
$$w_i^{t+1} = w_i^t + \eta(y - \hat{y})x \tag{2}$$

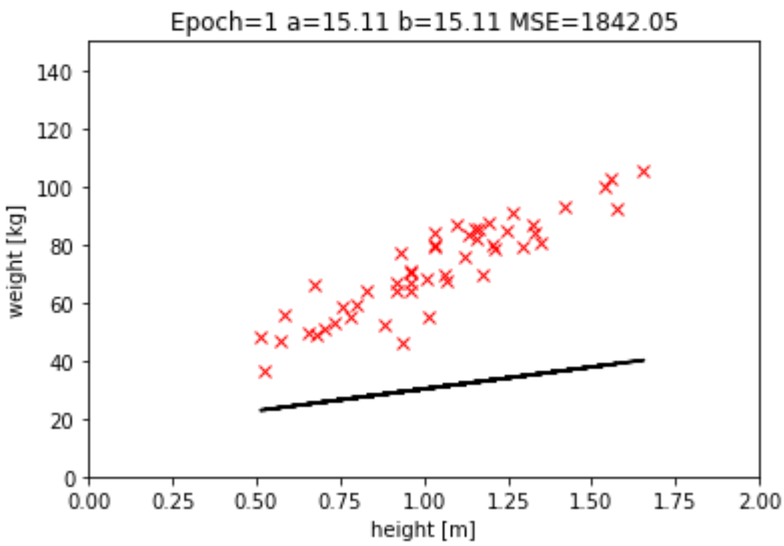
Example: Perceptron regression with Hebbian learning

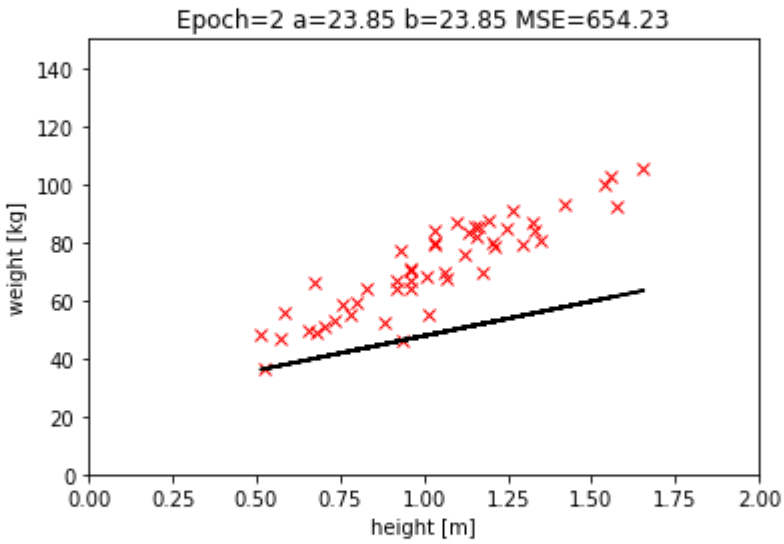
In [2]:

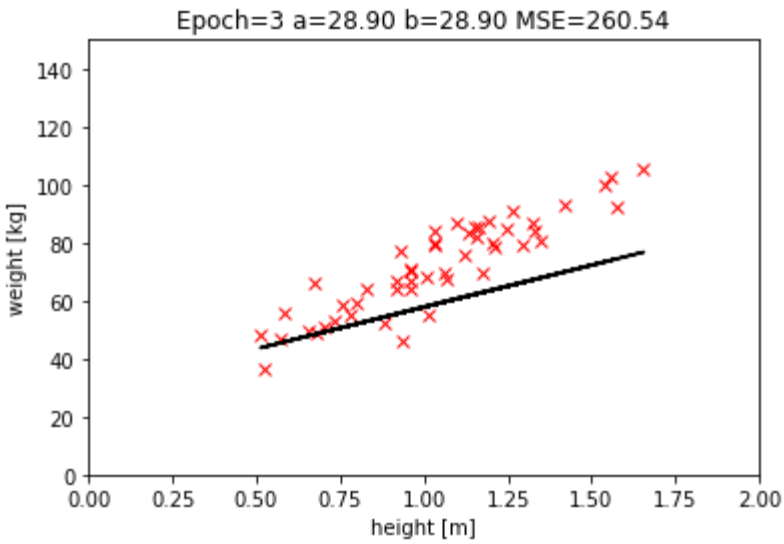
```
import matplotlib.pyplot as plt
import numpy as np

# Coodinate system
plt.xlabel('height [m]')
plt.ylabel('weight [kg]')
plt.axis([0,2,0,150])

# Generate random points
np.random.seed(42) # to always get the same points
N = 50
x = np.random.normal(1.1,0.3,N)
a_gt = 50.0
b_gt = 20.0
y_noise =  np.random.normal(0,8,N) # Measurements from the class 1\n",
y = a_gt*x+b_gt+y_noise
plt.plot(x,y,'rx')
plt.title('Training data for Perceptron')
plt.show()

# Compute MSE heat map for different a and b
a_t = 0
b_t = 0
num_of_epochs = 10
learning_rate = 0.005

y_h = a_t*x+b_t
MSE = np.sum((y-y_h)**2)/N
plt.title(f'Epoch={0} a={a_t:.2f} b={b_t:.2f} MSE={MSE:.2f}')
plt.xlabel('height [m]')
plt.ylabel('weight [kg]')
plt.axis([0,2,0,150])
plt.plot(x,y,'rx')
plt.plot(x,a_t*x+b_t,'k-')
plt.show()
for e in range(num_of_epochs):
    for x_e_ind,x_e in enumerate(x):
        # Hebbian learning implemented
        y_e = a_t*x_e+b_t
        a_t = a_t+learning_rate*(y[x_e_ind]-y_e)*x_e
        b_t = b_t+learning_rate*(y[x_e_ind]-y_e)*x_e
    # Compute train error
    y_h = a_t*x+b_t
    MSE = np.sum((y-y_h)**2)/N
    plt.title(f'Epoch={e+1} a={a_t:.2f} b={b_t:.2f} MSE={MSE:.2f}')
    plt.xlabel('height [m]')
    plt.ylabel('weight [kg]')
    plt.axis([0,2,0,150])
    plt.plot(x,y,'rx')
    plt.plot(x,a_t*x+b_t,'k-')
    plt.show()
```

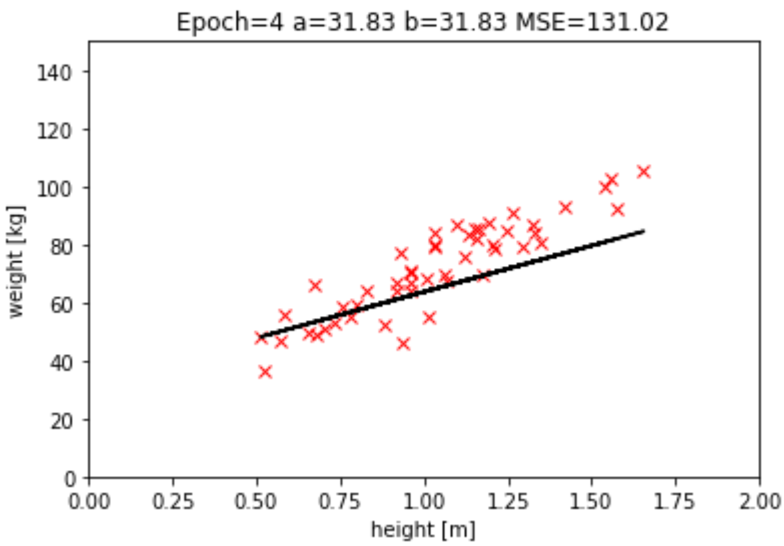

Training data for Perceptron
weight [kg]
height [m]
0
20
40
60
80
100
120
140
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

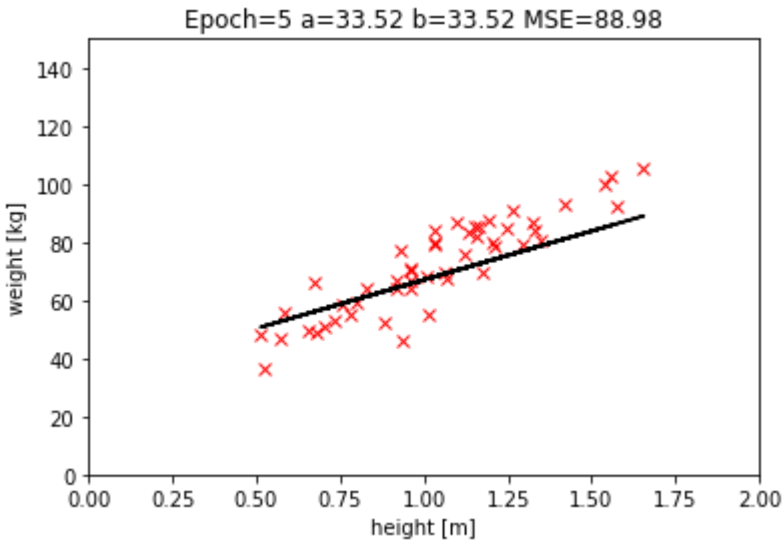

Epoch=0 a=0.00 b=0.00 MSE=5410.82
weight [kg]
height [m]
0
20
40
60
80
100
120
140
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

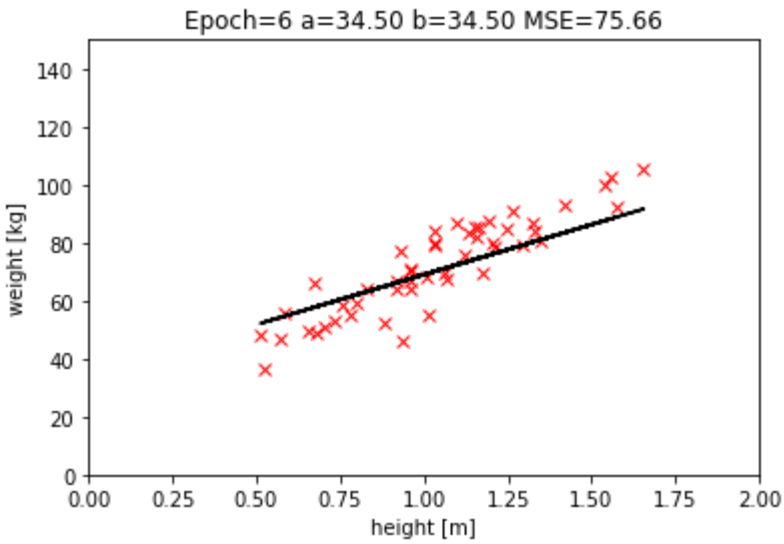

Epoch=1 a=15.11 b=15.11 MSE=1842.05
weight [kg]
height [m]
0
20
40
60
80
100
120
140
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

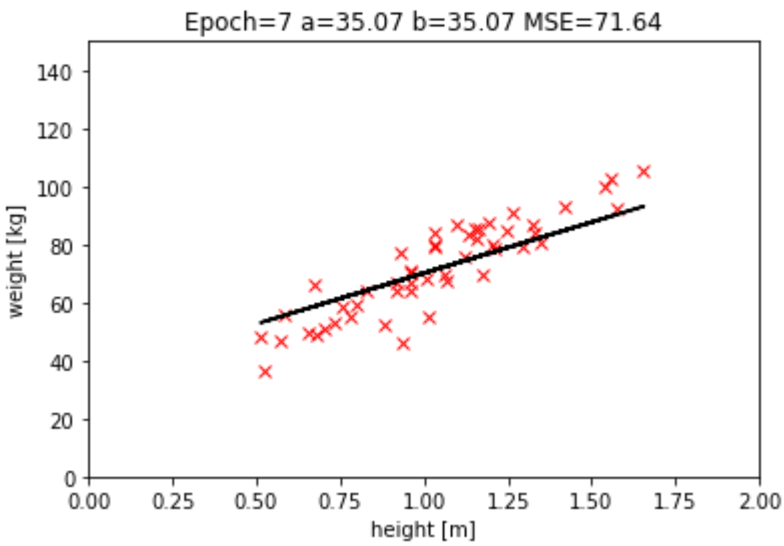

Epoch=2 a=23.85 b=23.85 MSE=654.23
weight [kg]
0
20
40
60
80
100
120
140
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
height [m]

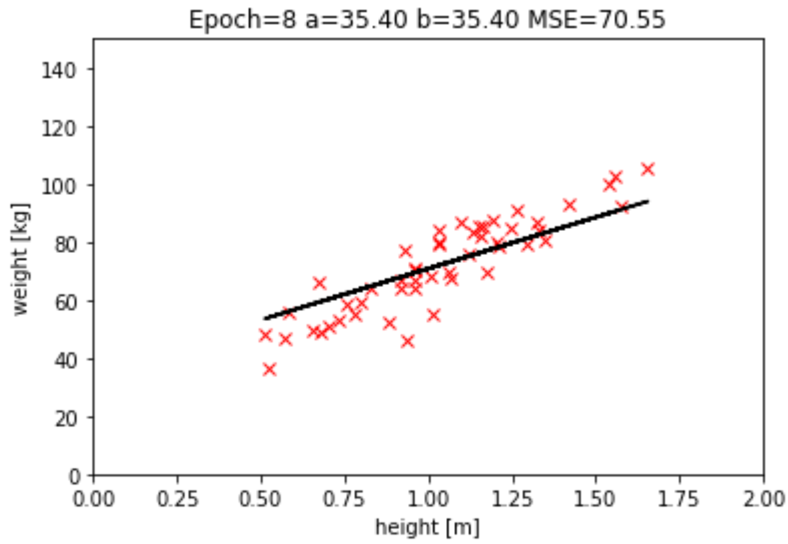

Epoch=3 a=28.90 b=28.90 MSE=260.54
weight [kg]
0
20
40
60
80
100
120
140
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
height [m]

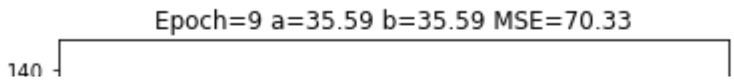

Epoch=4 a=31.83 b=31.83 MSE=131.02
weight [kg]
0
20
40
60
80
100
120
140
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
height [m]


Epoch=5 a=33.52 b=33.52 MSE=88.98
weight [kg]
height [m]
0
20
40
60
80
100
120
140
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00


Epoch=6 a=34.50 b=34.50 MSE=75.66
weight [kg]
height [m]
0
20
40
60
80
100
120
140
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00


Epoch=7 a=35.07 b=35.07 MSE=71.64
weight [kg]
height [m]
0
20
40
60
80
100
120
140
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00





Gradient descent

If a non-linear transfer function, such as logsig(), is added to the perceptron, then we need to optimize it using an iterative algorithm. Then Gradient Descent (GD) was introduced.

Example: Perceptron regression using GD (TensorFlow implementation)

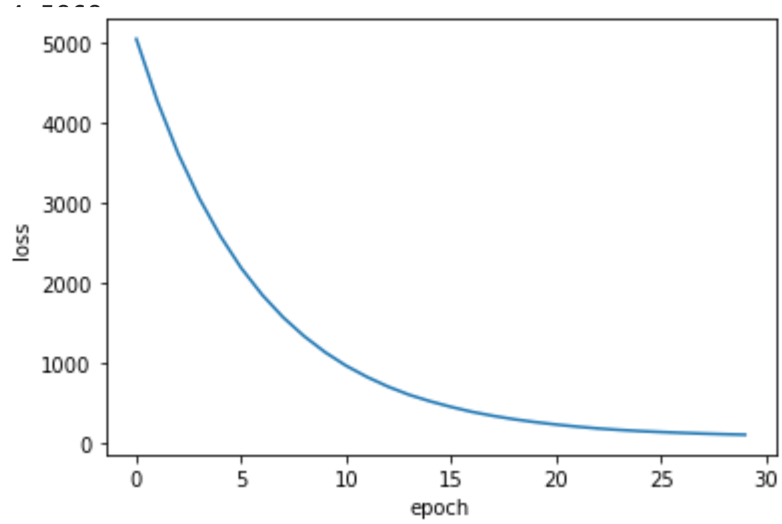
In [3]:

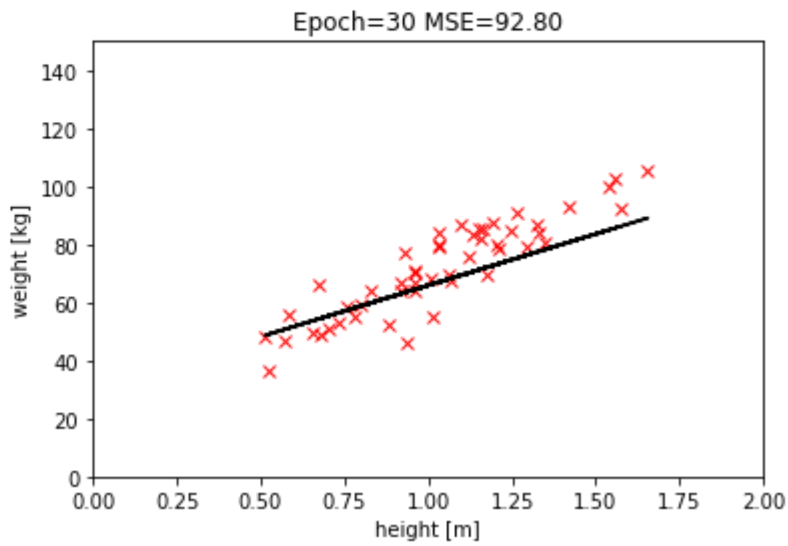
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import keras
# Model sequential
model = Sequential()
# 1st hidden layer (we also need to tell the input dimension)
#   10 neurons, but you can change to play a bit
model.add(Dense(1, input_dim=1, activation='linear'))
## 2nd hidden layer - YOU MAY TEST THIS
#model.add(Dense(10, activation='sigmoid'))
# Output layer
#model.add(Dense(1, activation='sigmoid'))
#model.add(Dense(1, activation='tanh'))
# Learning rate has huge effect
keras.optimizers.SGD(lr=0.5)
model.compile(optimizer='sgd', loss='mse', metrics=['mse'])

num_of_epochs = 30
tr_hist = model.fit(x, y, epochs=num_of_epochs, verbose=1)

plt.plot(tr_hist.history['loss'])
plt.ylabel('loss')
plt.xlabel('epoch')
#plt.legend(['opetus'], loc='upper right')
plt.show()


y_h = np.squeeze(model.predict(x))
MSE = np.sum((y-y_h)**2)/N
plt.xlabel('height [m]')
plt.ylabel('weight [kg]')
plt.title(f'Epoch={num_of_epochs} MSE={MSE:.2f}')
plt.axis([0,2,0,150])
plt.plot(x,y,'rx')
plt.plot(x,y_h,'k-')
plt.show()
```

```
2022-09-22 14:25:51.789676: I tensorflow/compiler/mlir/mlir_graph_optimizat
ion_pass.cc:116] None of the MLIR optimization passes are enabled (register
ed 2)
2022-09-22 14:25:51.807352: I tensorflow/core/platform/profile_utils/cpu_ut
ils.cc:112] CPU Frequency: 2899885000 Hz
Epoch 1/30
2/2 [==============================] - 0s 2ms/step - loss: 5201.5078 - mse:
5201.5078
Epoch 2/30
2/2 [==============================] - 0s 2ms/step - loss: 4330.4606 - mse:
4330.4606
Epoch 3/30
2/2 [==============================] - 0s 2ms/step - loss: 3666.5986 - mse:
3666.5986
Epoch 4/30
2/2 [==============================] - 0s 1ms/step - loss: 3128.4430 - mse:
3128.4430
Epoch 5/30
2/2 [==============================] - 0s 1ms/step - loss: 2550.3516 - mse:
2550.3516
Epoch 6/30
2/2 [==============================] - 0s 1ms/step - loss: 2203.9054 - mse:
2203.9054
```

```
Epoch 7/30
2/2 [==============================] - 0s 1ms/step - loss: 1851.8087 - mse:
1851.8087
Epoch 8/30
2/2 [==============================] - 0s 2ms/step - loss: 1595.0334 - mse:
1595.0334
Epoch 9/30
2/2 [==============================] - 0s 2ms/step - loss: 1348.7713 - mse:
1348.7713
Epoch 10/30
2/2 [==============================] - 0s 3ms/step - loss: 1114.8876 - mse:
1114.8876
Epoch 11/30
2/2 [==============================] - 0s 1ms/step - loss: 985.8984 - mse:
985.8984
Epoch 12/30
2/2 [==============================] - 0s 2ms/step - loss: 825.2534 - mse:
825.2534
Epoch 13/30
2/2 [==============================] - 0s 2ms/step - loss: 681.2597 - mse:
681.2597
Epoch 14/30
2/2 [==============================] - 0s 2ms/step - loss: 627.0206 - mse:
627.0206
Epoch 15/30
2/2 [==============================] - 0s 4ms/step - loss: 537.1965 - mse:
537.1965
Epoch 16/30
2/2 [==============================] - 0s 3ms/step - loss: 416.7895 - mse:
416.7895
Epoch 17/30
2/2 [==============================] - 0s 2ms/step - loss: 398.0670 - mse:
398.0670
Epoch 18/30
2/2 [==============================] - 0s 2ms/step - loss: 330.2783 - mse:
330.2783
Epoch 19/30
2/2 [==============================] - 0s 1ms/step - loss: 296.8544 - mse:
296.8544
Epoch 20/30
2/2 [==============================] - 0s 3ms/step - loss: 259.9581 - mse:
259.9581
Epoch 21/30
2/2 [==============================] - 0s 2ms/step - loss: 225.9912 - mse:
225.9912
Epoch 22/30
2/2 [==============================] - 0s 2ms/step - loss: 187.4820 - mse:
187.4819
Epoch 23/30
2/2 [==============================] - 0s 2ms/step - loss: 177.2939 - mse:
177.2939
Epoch 24/30
2/2 [==============================] - 0s 2ms/step - loss: 151.1858 - mse:
151.1858
Epoch 25/30
2/2 [==============================] - 0s 2ms/step - loss: 154.3157 - mse:
154.3157
Epoch 26/30
2/2 [==============================] - 0s 2ms/step - loss: 133.8714 - mse:
133.8714
Epoch 27/30
2/2 [==============================] - 0s 2ms/step - loss: 115.3947 - mse:
115.3947
```

```
Epoch 28/30
2/2 [==============================] - 0s 2ms/step - loss: 106.4019 - mse:
106.4019
Epoch 29/30
2/2 [==============================] - 0s 1ms/step - loss: 102.3186 - mse:
102.3186
Epoch 30/30
2/2 [==============================] - 0s 1ms/step - loss: 94.5868 - mse: 9
```





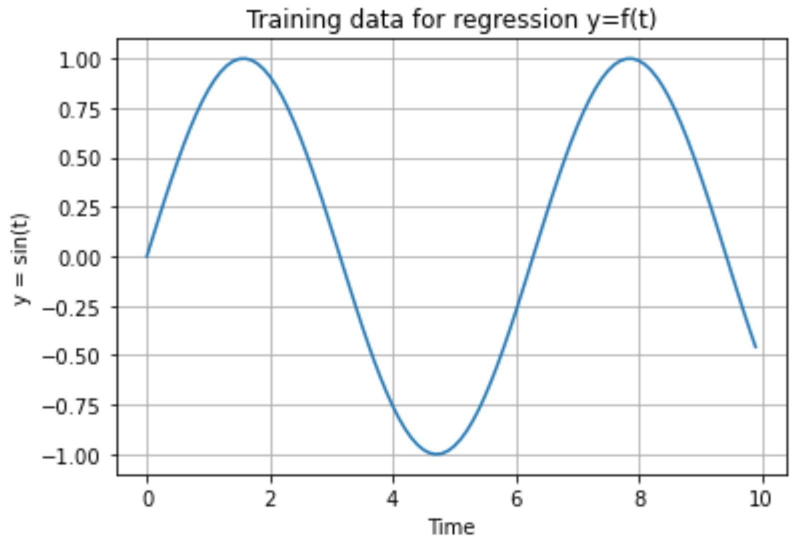
Multi-layer Perceptron (MLP)

Multi-layer Perceptron is the father of modern convolutional neural networks (CNNs) - the principal idea of training MLPs and CNNs is same.

Example: MLP regression of sinusoidal

In [4]:

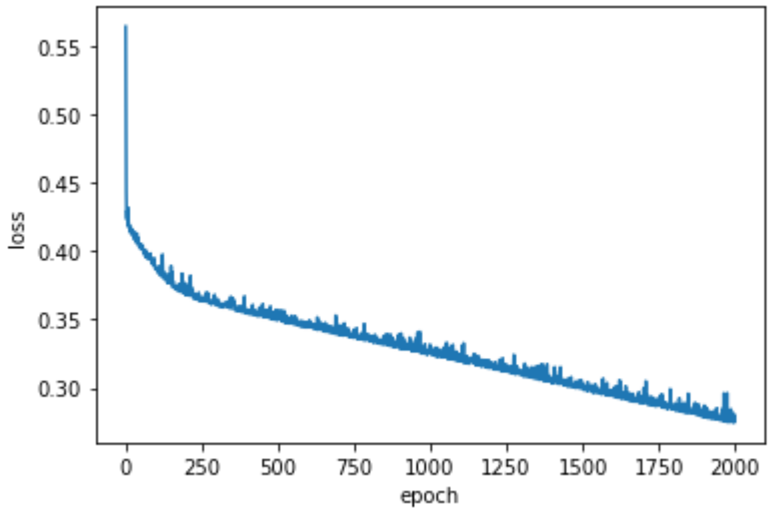
```
# Generate a sine wave
t = np.arange(0, 10, 0.1);
y = np.sin(t)
plt.plot(t, y)
plt.title('Training data for regression y=f(t)')
plt.xlabel('Time')
plt.ylabel('y = sin(t)')
plt.grid(True, which='both')
plt.show()
```



In [5]:
```
# Construct a MPL

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import keras
# Model sequential
model = Sequential()
# 1st hidden layer (we also need to tell the input dimension)
#   10 neurons, but you can change to play a bit
model.add(Dense(10, input_dim=1, activation='sigmoid'))
## 2nd hidden layer - YOU MAY TEST THIS
#model.add(Dense(10, activation='sigmoid'))
# Output layer
#model.add(Dense(1, activation='sigmoid'))
model.add(Dense(1, activation='tanh'))
# Learning rate has huge effect
keras.optimizers.SGD(lr=0.2)
model.compile(optimizer='sgd', loss='mse', metrics=['mse'])
```

We train the network for number of epochs (10-10000, but you may test different values). Set vebose=1 to see progress during training.

In [6]:
```
tr_hist = model.fit(t, y, epochs=2000, verbose=0)
plt.plot(tr_hist.history['loss'])
plt.ylabel('loss')
plt.xlabel('epoch')
#plt.legend(['opetus'], loc='upper right')
plt.show()
```

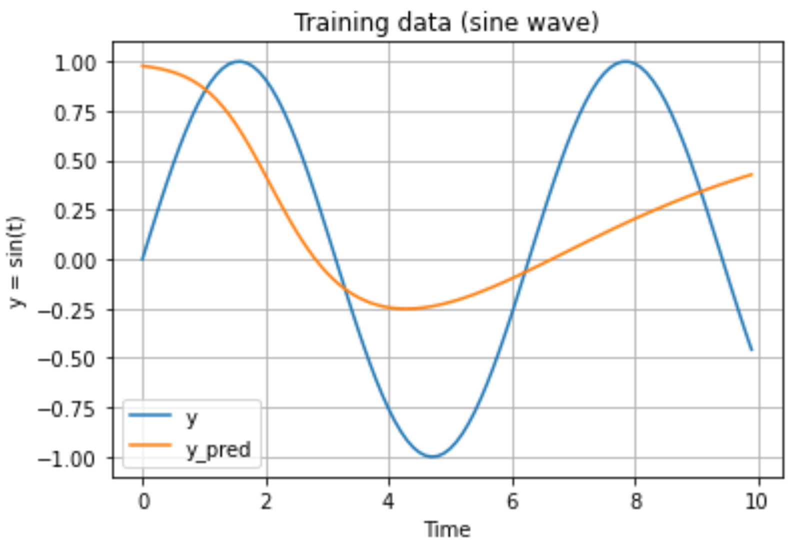


Let's test how well the network models the data

In [7]:

```
from sklearn.metrics import mean_squared_error
y_pred = model.predict(t)
print(y[1])
print(y_pred[1])
print(np.sum(np.absolute(np.subtract(y,y_pred)))/len(t))
print(np.square(np.subtract(y,y_pred)).mean())
print(len(y))
print(np.divide(np.sum(np.square(y-y_pred)),len(y)))
print('MSE=',mean_squared_error(y,y_pred))
plt.plot(t, y, label='y')
plt.plot(t, y_pred, label='y_pred')
plt.title('Training data (sine wave)')
plt.xlabel('Time')
plt.ylabel('y = sin(t)')
plt.grid(True, which='both')
plt.legend()
plt.show()
```

```
0.09983341664682815
[0.9730344]
63.96807082195586
0.5824907576780833
100
58.249075767808336
MSE= 0.27439735453959974
```

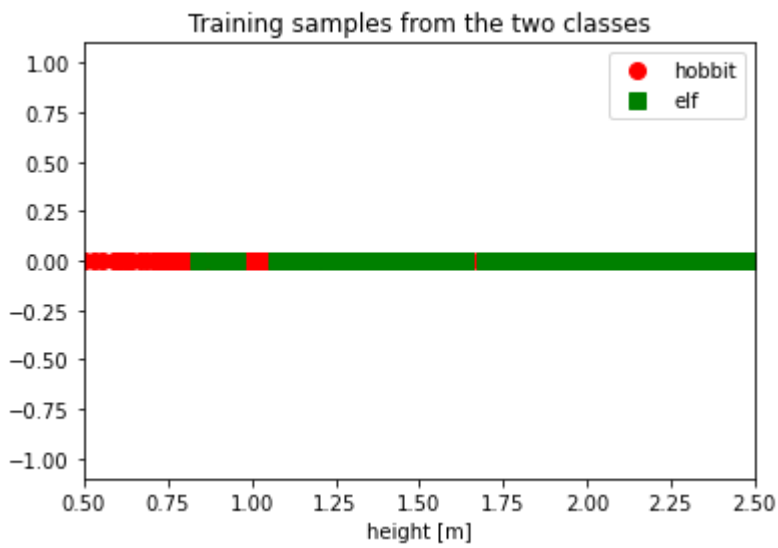


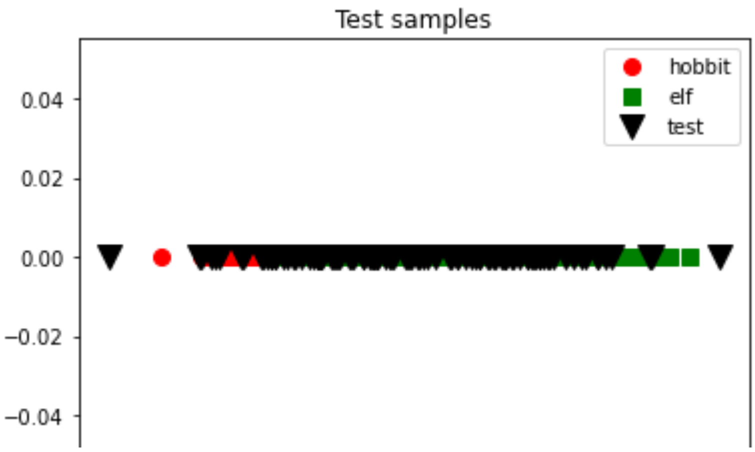
Example: MLP classification

Samples (height of hobits and elves)

In [8]:

```
# Coodinate system
plt.xlabel('height [m]')
#plt.ylabel('paino [kg]')
plt.axis([0.5,2.5,-1.1,+1.1])

# Generate random points for training
np.random.seed(11) # to always get the same points
N = 200
x_h = np.random.normal(1.1,0.3,N)
x_e = np.random.normal(1.9,0.4,N)
plt.plot(x_h,np.zeros([N,1]),'ro', markersize=8, label="hobbit")
plt.plot(x_e,np.zeros([N,1]),'gs', markersize=8, label="elf")
plt.title('Training samples from the two classes')
plt.legend()
plt.show()

# Generate random points for testing
N_t =  50
x_h_test = np.random.normal(1.1,0.3,N_t) # h as hobit
x_e_test = np.random.normal(1.9,0.4,N_t) # e as elf
plt.plot(x_h,np.zeros([N,1]),'ro', markersize=8, label="hobbit")
plt.plot(x_e,np.zeros([N,1]),'gs', markersize=8, label="elf")
plt.plot(x_e_test,np.zeros([N_t,1]),'kv',linewidth=1, markersize=12, label=
plt.plot(x_h_test,np.zeros([N_t,1]),'kv', markersize=12)
plt.title('Test samples')
plt.legend()
plt.show()

# 1-NN classifier

# Form the train input and output vectors  (1: hobit, 2: elf)
x_tr = np.concatenate((x_h,x_e))
y_tr = np.concatenate((1*np.ones([x_h.shape[0],1]),2*np.ones([x_e.shape[0]

# Form the test input and output vectors
x_te = np.concatenate((x_h_test,x_e_test))
y_te = np.concatenate((1*np.ones([N_t,1]),2*np.ones([N_t,1])))
```





In [9]:

```
# With this example you learn the meaning of network size (# of neurons in
# and the effect of learning rate 0.1 vs. 0.001

# Tee neuroverkko
model = Sequential()
# 1 tai 100
model.add(Dense(100, input_dim=1, activation='sigmoid'))
# Ulostuloja aina kaksi, yksi kummallekin luokalle
model.add(Dense(2, activation='sigmoid'))
# 0.1 tai 0.001
opt = keras.optimizers.SGD(lr=0.1)
model.compile(optimizer=opt, loss='mse', metrics=['mse'])

# Yksi-kuuma (one hot) -koodataan luokka 1 -> [1 0] 2 -> [0 1]
y_tr_2 = np.empty([y_tr.shape[0],2])
y_tr_2[np.where(y_tr==1),0] = 1
y_tr_2[np.where(y_tr==1),1] = 0
y_tr_2[np.where(y_tr==2),0] = 0
y_tr_2[np.where(y_tr==2),1] = 1

# Opetus - epokkeja 1 tai 100
model.fit(x_tr, y_tr_2, epochs=100, verbose=0)

# Tulokset opetuspisteille
y_tr_pred = np.empty(y_tr.shape)
y_tr_pred_2 = np.squeeze(model.predict(x_tr))
for pred_ind in range(y_tr_pred_2.shape[0]):
    if y_tr_pred_2[pred_ind][0] > y_tr_pred_2[pred_ind][1]:
        y_tr_pred[pred_ind] = 1
    else:
        y_tr_pred[pred_ind] = 2

tot_correct = len(np.where(y_tr-y_tr_pred == 0)[0])
print(f'Classication accuracy (training data): {tot_correct/len(y_tr)*100}

# Tulokset testauspisteille
y_te_pred = np.empty(y_te.shape)
y_te_pred_2 = np.squeeze(model.predict(x_te))
for pred_ind in range(y_te_pred_2.shape[0]):
    if y_te_pred_2[pred_ind][0] > y_te_pred_2[pred_ind][1]:
        y_te_pred[pred_ind] = 1
    else:
        y_te_pred[pred_ind] = 2

tot_correct = len(np.where(y_te-y_te_pred == 0)[0])
print(f'Classication accuracy (test data): {tot_correct/len(y_te)*100}%')
```

```
Classication accuracy (training data): 86.5%
Classication accuracy (test data): 89.0%
```

References

C.M. Bishop (2006): Pattern Recognition and Machine Learning, Chapter 5