# Introduction to Machine Learning

Daniel Sandu

2023

# One of the most used buzz words of this decade

Depending on who you ask artificial intelligence is…

# One of the most used buzz words of this decade

Depending on who you ask artificial intelligence is…

- The horseman of the apocalypse

# One of the most used buzz words of this decade

Depending on who you ask artificial intelligence is…

- The horseman of the apocalypse

- Going to take your job

# One of the most used buzz words of this decade

Depending on who you ask artificial intelligence is…

- The horseman of the apocalypse
- Going to take your job
- A new era of global prosperity

# Artificial Intelligence

But what is (artificial) intelligence?

# Artificial Intelligence

But what is (artificial) intelligence?

The property of a biological or artificial entity to become better at solving problems without being told explicitly what to do.

# Artificial Intelligence

Why artificial intelligence?

Why not procedural programming?

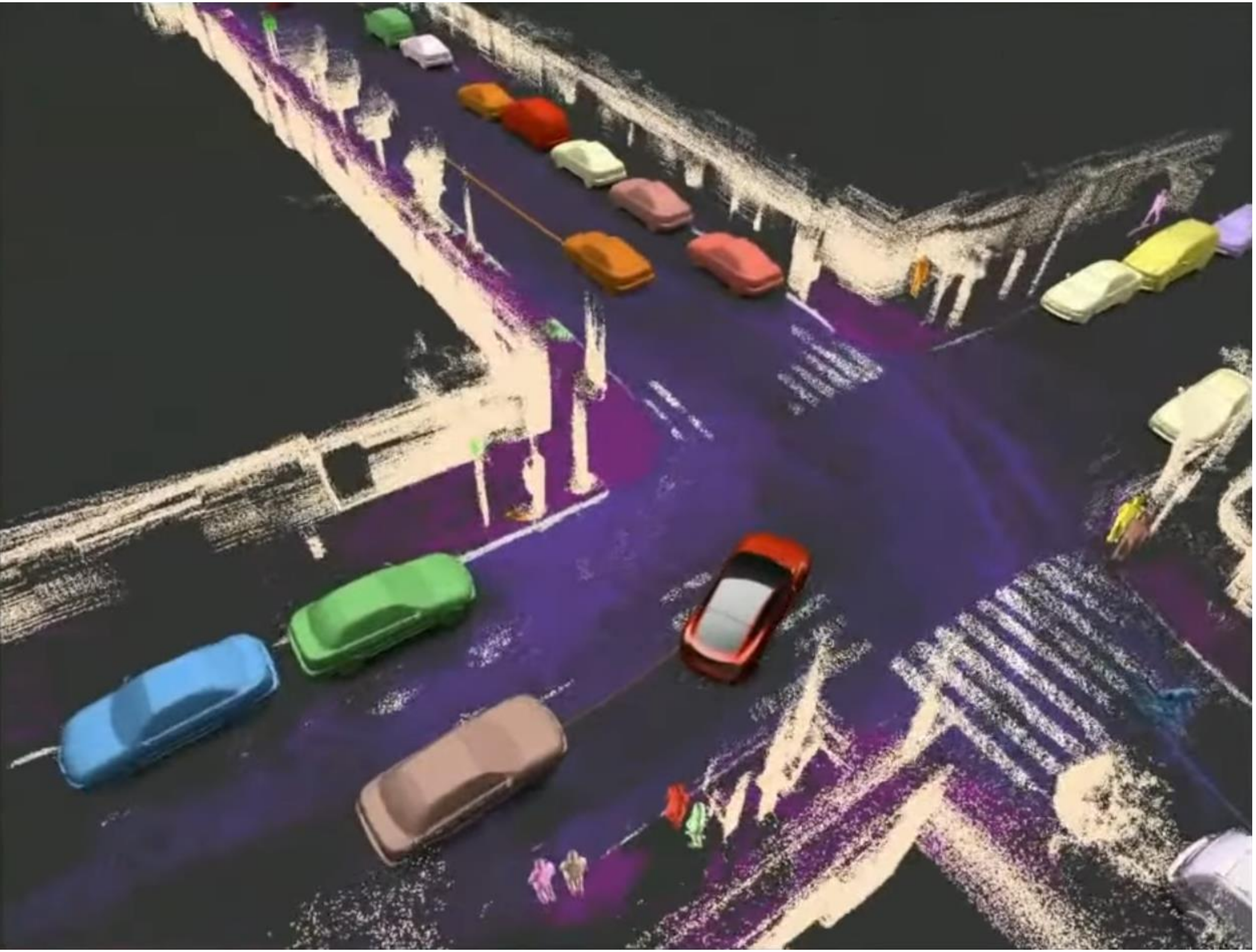
A couple of examples…

# Stable Diffusion

# ChatGPT

What is artificial intelligence?

Artificial intelligence, or AI, refers to the development of computer systems that can perform tasks that normally require human intelligence, such as visual perception, voice recognition, decision-making, and language translation. These systems use algorithms, mathematical models, and data to learn and improve their performance over time. AI has the potential to revolutionize many aspects of our lives, from the way we work and communicate to the way we diagnose and treat diseases.



# Tesla Autopilot

# Objective of this course

Design machine learning algorithms starting from almost nothing

# Objective of this course

Design machine learning algorithms starting from almost nothing

What you'll need to complete this course…

# Objective of this course

Design machine learning algorithms starting from almost nothing

What you'll need to complete this course…

- High school mathematics – linear algebra and derivatives

# Objective of this course

Design machine learning algorithms starting from almost nothing

What you'll need to complete this course…

- High school mathematics – linear algebra and derivatives
- Basic programming – keep it simple and always write tests

# Objective of this course

Design machine learning algorithms starting from almost nothing

What you'll need to complete this course…

- High school mathematics – linear algebra and derivatives
- Basic programming – keep it simple and always write tests
- And most importantly… **ask questions if you get stuck**

# Tools of the trade

You’ll need python version 3.11 (earlier may work but not tested)

# Tools of the trade

You’ll need python version 3.11 (earlier may work but not tested)

Run the following command in the console to install necessary packages

```
python -m pip install numpy matplotlib
```

# Types of machine learning

# Types of machine learning

**Supervised learning**

The algorithm learns by being supplied labeled data.

*Example: learning to provide diagnostics using metrics such as MRI scans and blood tests from previously given diagnostics*

# Types of machine learning

**Supervised learning**

The algorithm learns by being supplied labeled data.

*Example: learning to provide diagnostics using metrics such as MRI scans and blood tests from previously given diagnostics*

**Unsupervised learning**

The algorithm learns by itself and finds structure in data.

*Example: clustering people together using time spent or frequency of various activities to identify athletes, gamers, tv show watchers or other insightful categories*

# Types of machine learning

**Supervised learning**

The algorithm learns by being supplied labeled data.

*Example: learning to provide diagnostics using metrics such as MRI scans and blood tests from previously given diagnostics*

**Unsupervised learning**

The algorithm learns by itself and finds structure in data.

*Example: clustering people together using time spent or frequency of various activities to identify athletes, gamers, tv show watchers or other insightful categories*

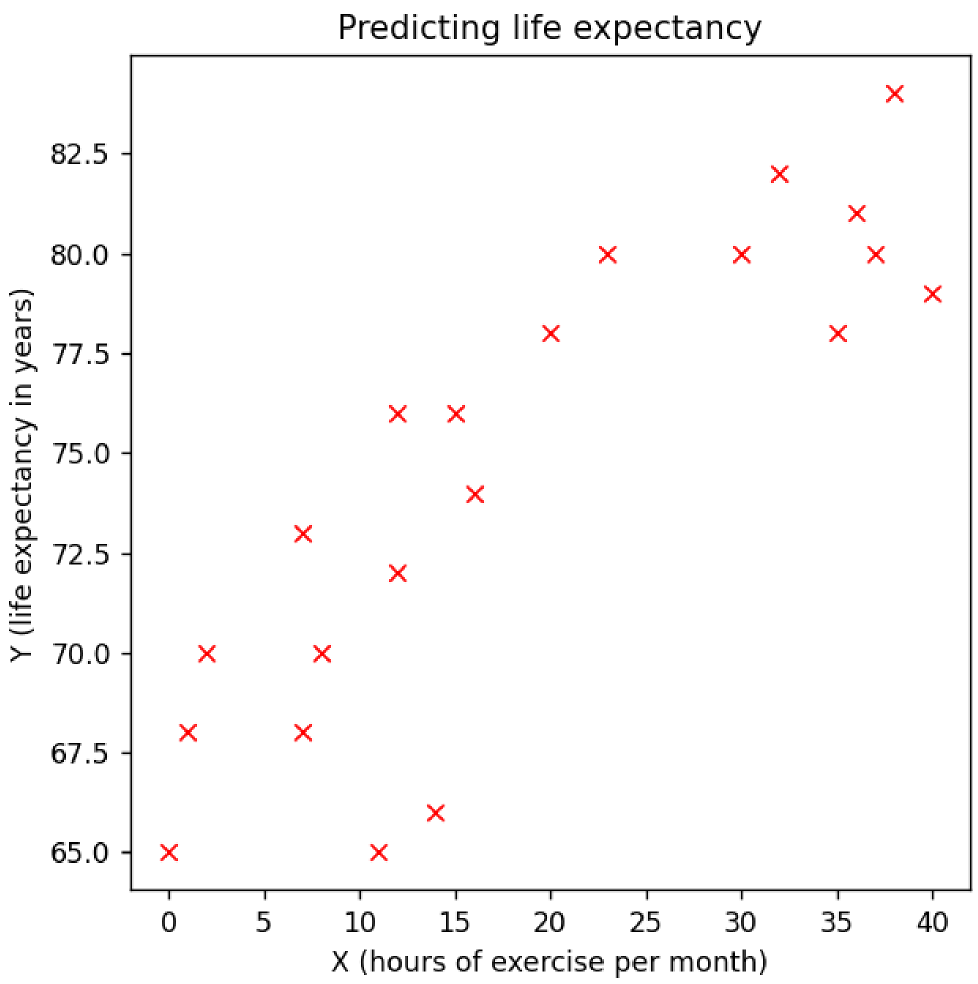
We'll be focusing on **supervised learning** in this course

# Supervised learning

**Linear Regression**

Given a data set of monthly hours of exercise and life expectancy train a model to make predictions.

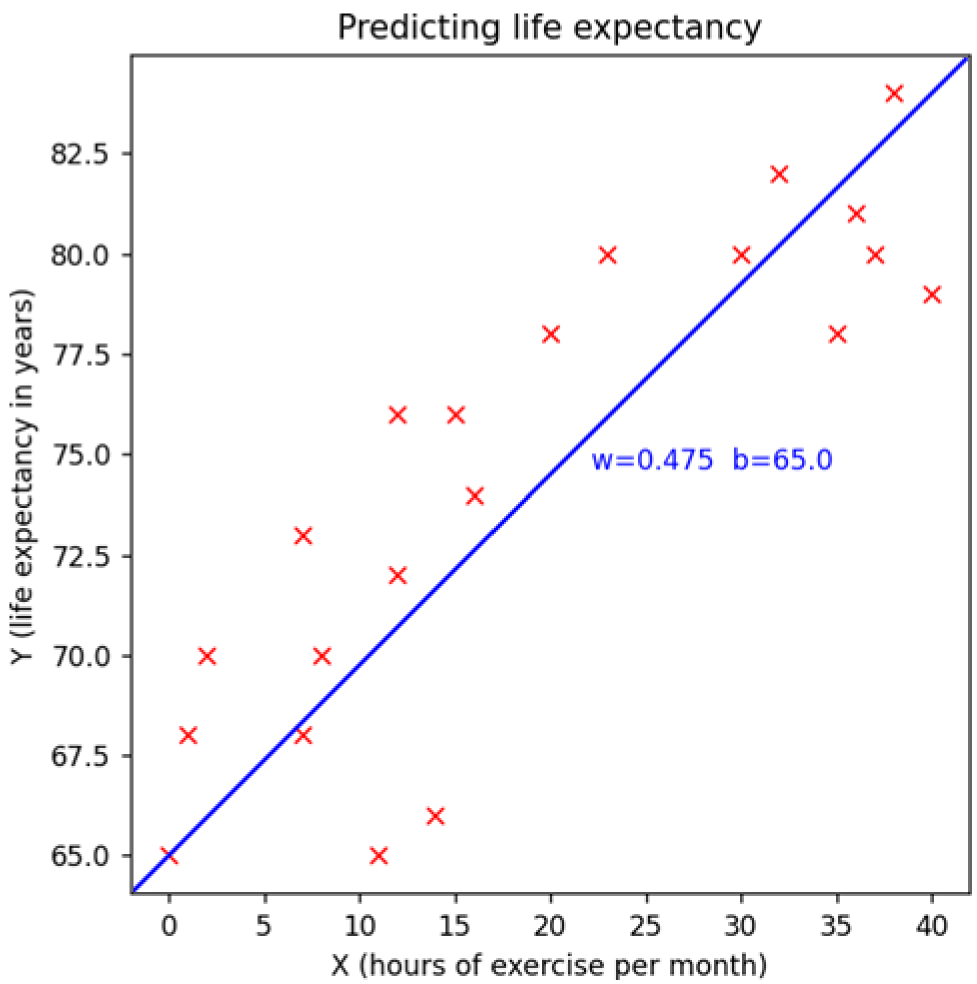
| **Hours of exercise** | **Life expectancy** |
|---|---|
| 20 | 78 |
| 0 | 65 |
| 36 | 81 |
| 12 | 72 |



# Linear Regression

$\boxed{x}$ → $\boxed{f(x) = wx + b}$ → $\boxed{\hat{y}}$

feature (hours of exercise) | model | prediction (life expectancy)




$w$ is the weight, also called parameter, of the model.

$b$ is the bias of the model. It's a flat increase to the prediction.

$\hat{y}$ is our prediction and $y$ is the ground truth.

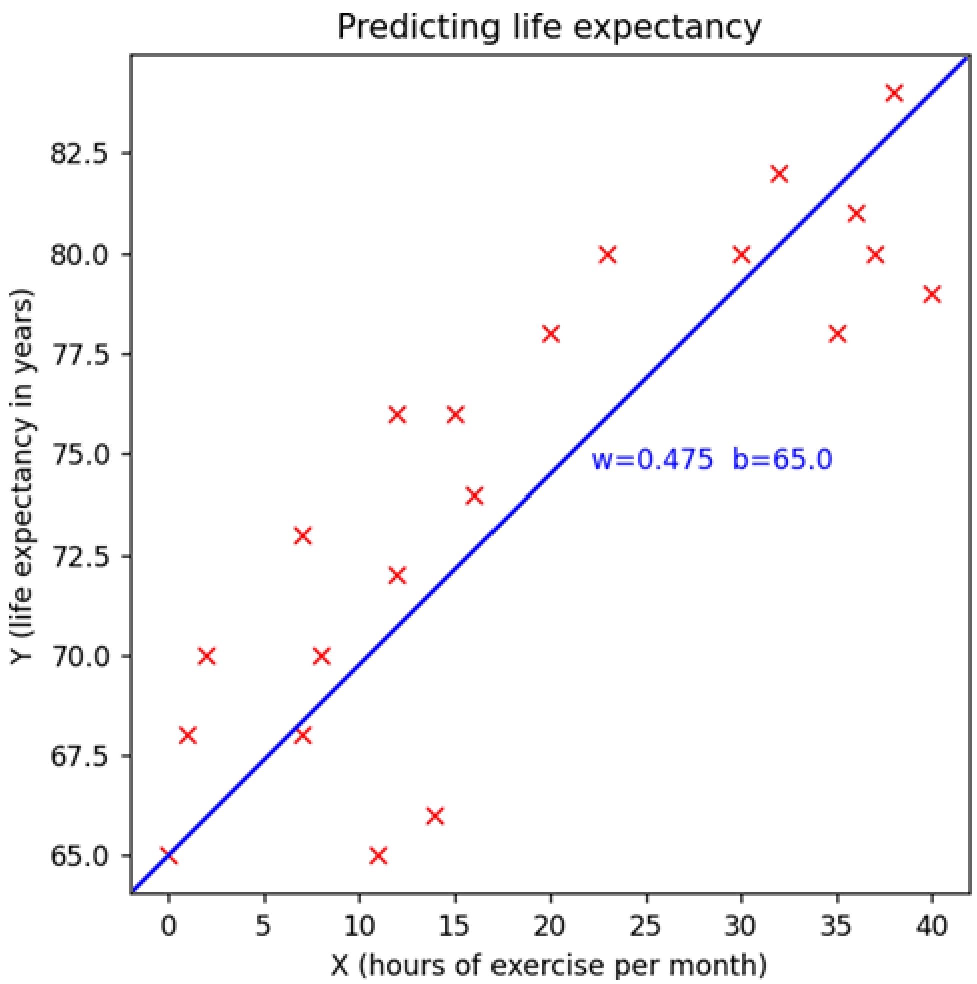
# Linear Regression

**Questions**

What happens if we change $w$?

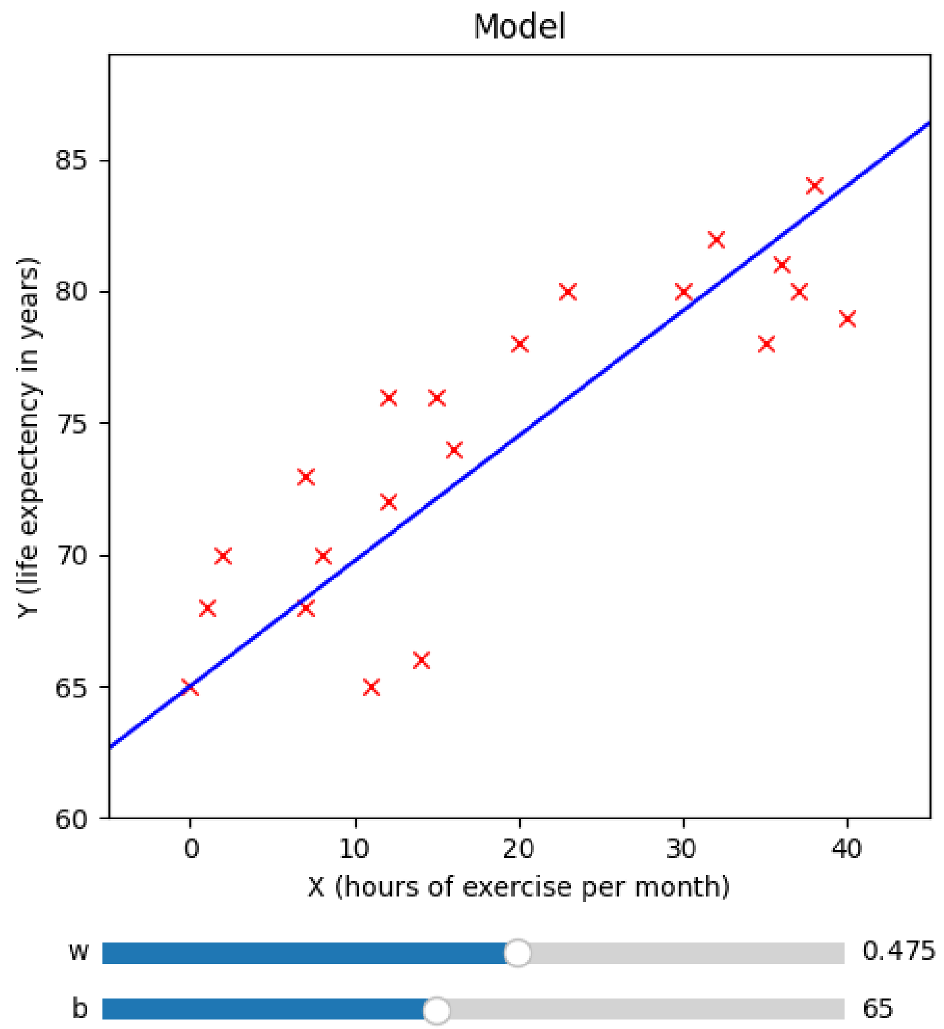
What about $b$?

Can we change $w$ and $b$ automatically to obtain a good model?



# Exercise – Linear Regression

Run the **predict_life.py** script and play with the $w$ and $b$ values to see how the model changes



# Linear Regression

**Can we find the parameters automatically?**

We need a “score” system for each model to judge its performance

The score should be good if our predictions are close to the ground truth

We choose the model with the best score to make future predictions

# Linear Regression – Loss function

Previously we mentioned $\hat{y}$ to be our prediction and $y$ to be the ground truth

Consider for our score, called a loss function, the simple function $\hat{y} - y$

If we are on target the score is $0$

The further away we are from the target the further away from $0$ we are

What are the problems with this loss function?

## Linear Regression – Loss function

We can change the loss function to $|\hat{y} - y|$ to obtain positive numbers

Can we do better?

# Linear Regression – Mean Squared Error loss function

Now consider the loss function $(\hat{y} - y)^2$

What are the benefits of this loss function?

# Linear Regression – Mean Squared Error loss function

Now consider the loss function $(\hat{y} - y)^2$

What are the benefits of this loss function?

- Values are always positive
- When we are on target the result is zero
- The further away from the target the larger the loss
- Big mispredictions are taxed more than small mispredictions
- Under-predictions are symmetrical to over-predictions
- It’s differentiable on its entire domain

# Linear Regression – Mean Squared Error loss function

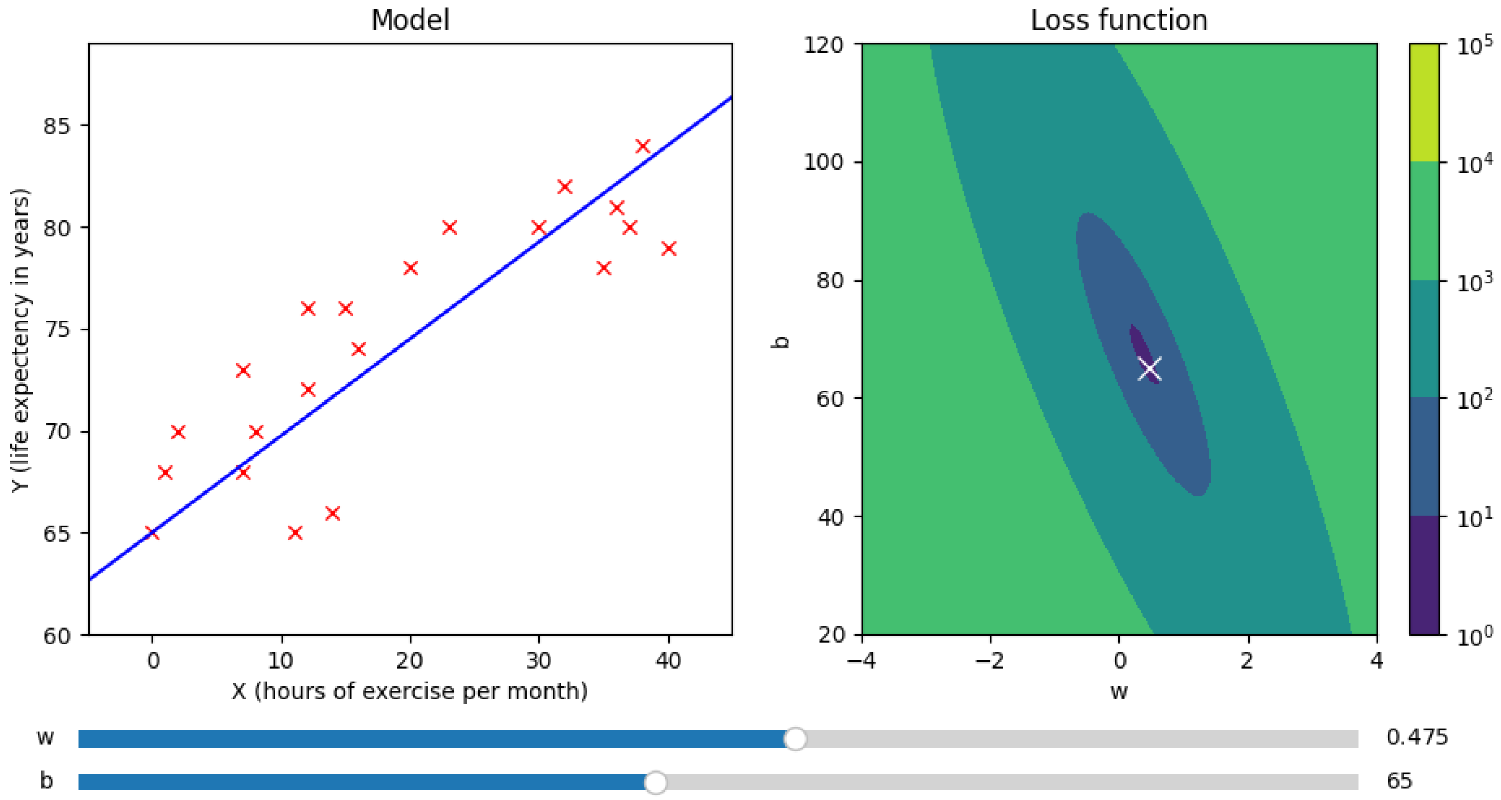
To get the overall performance of a model with parameters $w$ and $b$ we generalize the loss function to all samples in the data set

$$loss(w, b) = \frac{1}{2m}\sum_{i=1}^{m}\left(\hat{y}^{(i)} - y^{(i)}\right)^2$$

$m$ is the number of samples in the data set
$\hat{y}^{(i)}$ is the prediction for the $i^{th}$ sample in the data set
$y^{(i)}$ is the ground truth for the $i^{th}$ sample in the data set

# Exercise – Mean Squared Error loss function

Run the **loss_function.py** script and play with the $w$ and $b$ values to see how the loss function changes.



# Linear Regression – Finding a good model

Now we have a way to measure the performance of our models

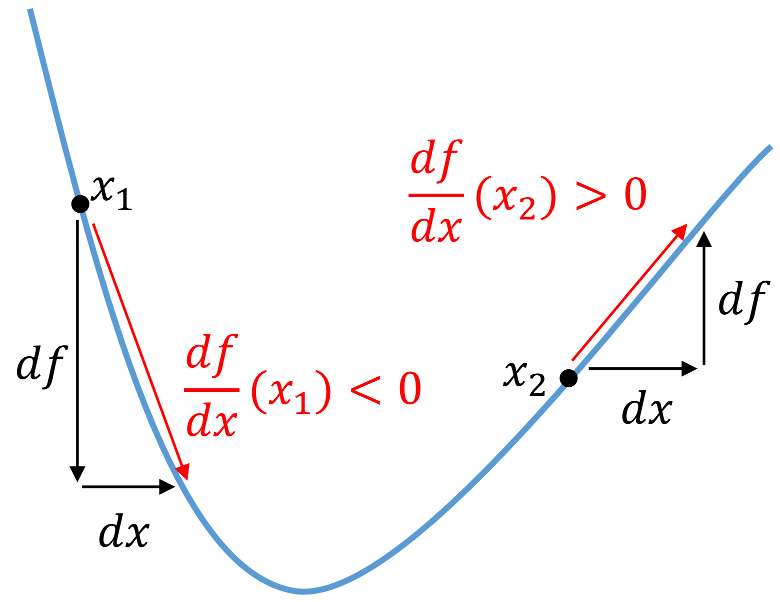
How can we find the parameters $w$ and $b$ automatically?

We can test values in a grid and pick up the best model, but this is slow and doesn't scale well

We can use some mathematical magic to find good parameters

# Linear Regression – Derivatives

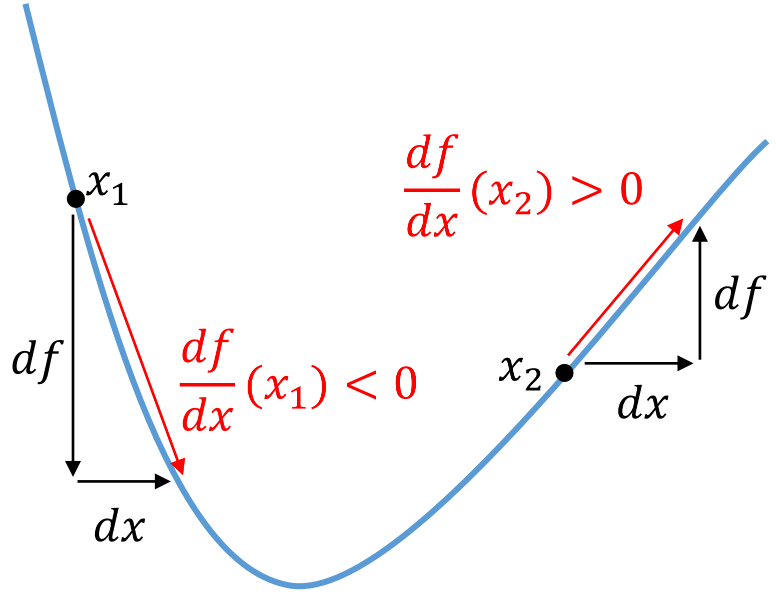
The derivative of a function $f$ tells use how much $f(x)$ changes if we change $x$ by a tiny amount



# Linear Regression – Derivatives

The derivative of a function $f$ tells use how much $f(x)$ changes if we change $x$ by a tiny amount

We can use the derivative of $f$ to change $x$ such that $f(x)$ is decreasing



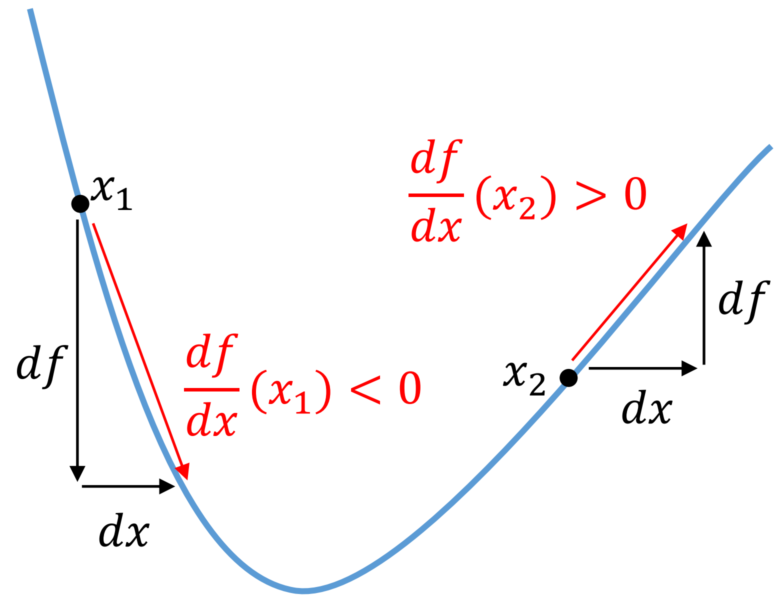
# Linear Regression – Derivatives

The derivative of a function $f$ tells use how much $f(x)$ changes if we change $x$ by a tiny amount

We can use the derivative of $f$ to change $x$ such that $f(x)$ is decreasing

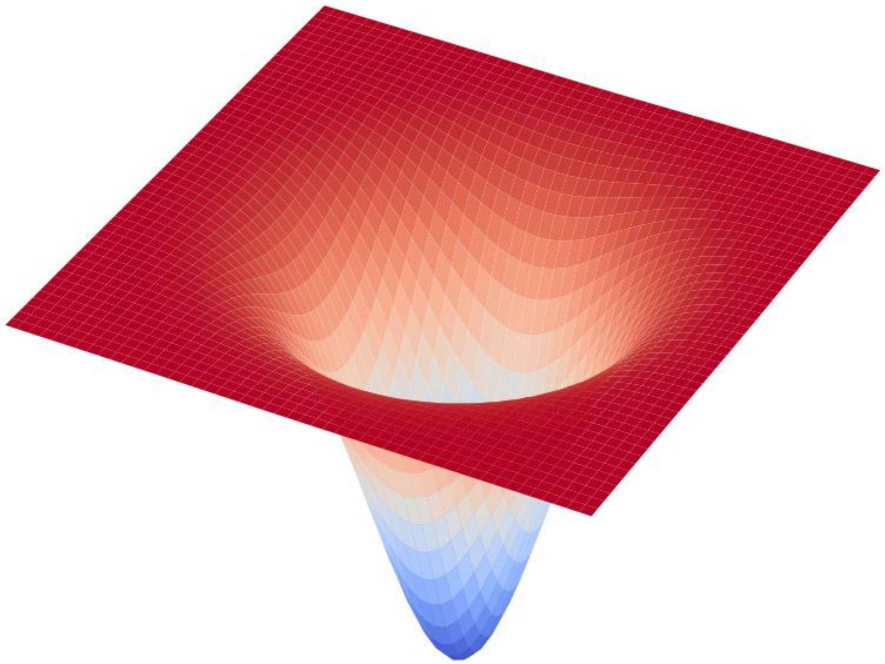
Changing $x$ to $x + \frac{df}{dx}(x)$ will increase $f(x)$

Changing $x$ to $x - \frac{df}{dx}(x)$ will decrease $f(x)$



# Linear Regression – Finding a good model

We can generalize the derivative from single to multiple function variables with partial derivatives



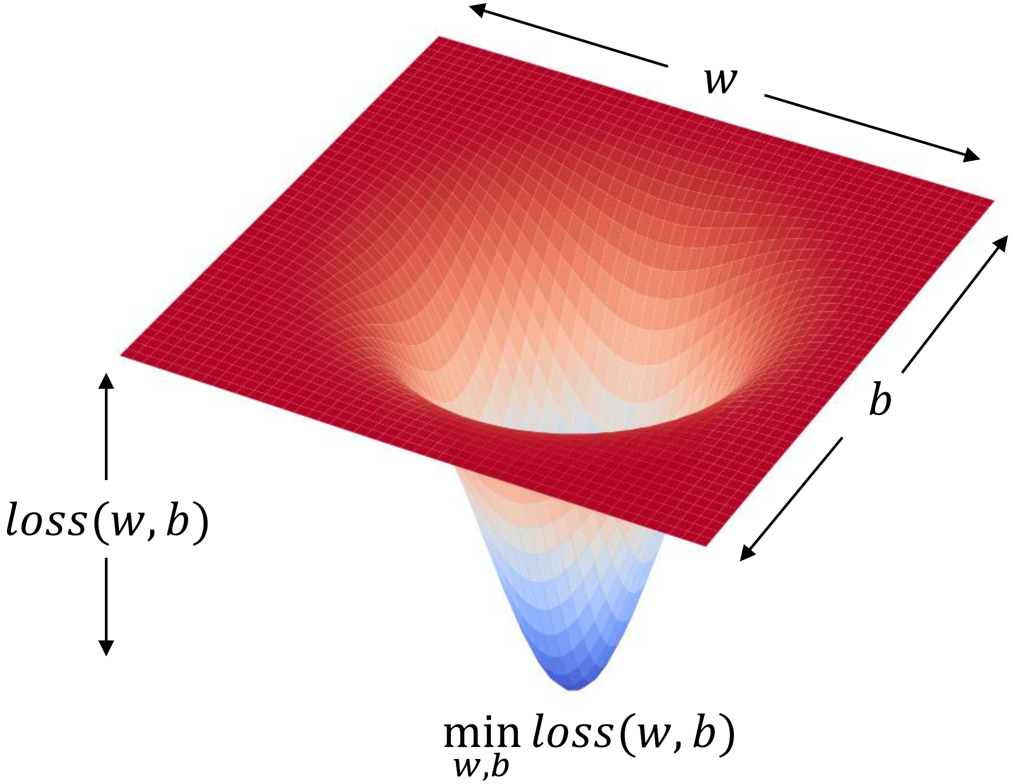
# Linear Regression – Finding a good model

We can generalize the derivative from single to multiple function variables with partial derivatives

Calculating the derivative of the loss function and subtracting it from each variable will decrease the loss function

$$w := w - \frac{\partial}{\partial w} loss(w, b)$$

$$b := b - \frac{\partial}{\partial b} loss(w, b)$$



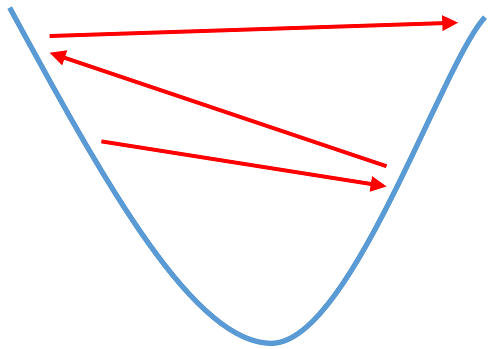
# Linear Regression – Gradient Descent

The iterative process of updating the variables of the loss function is called **Gradient Descent**

# Linear Regression – Gradient Descent

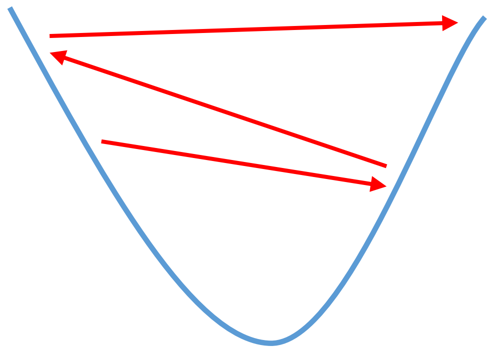
The iterative process of updating the variables of the loss function is called **Gradient Descent**

The gradient can be very large causing the algorithm to overshoot the minimum and to not converge



# Linear Regression – Gradient Descent

The iterative process of updating the variables of the loss function is called **Gradient Descent**



The gradient can be very large causing the algorithm to overshoot the minimum and to not converge

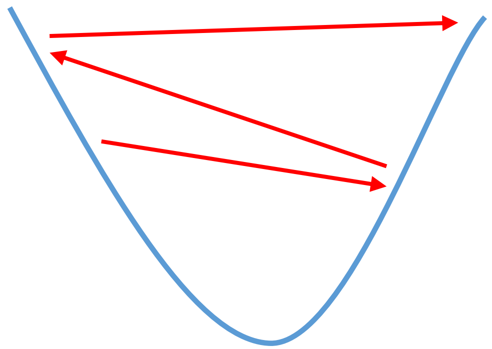
We can add the learning rate $\alpha$ to reduce the gradient

$$w \coloneqq w - \alpha \frac{\partial}{\partial w} loss(w, b)$$

$$b \coloneqq b - \alpha \frac{\partial}{\partial b} loss(w, b)$$

# Linear Regression – Gradient Descent

We can also normalize the data set by mapping all features to the interval $[-1,1]$ centering them at $0$



$$X_n^{(i)} = \frac{X^{(i)} - \frac{\max X + \min X}{2}}{\frac{\max X - \min X}{2}}$$

The loss function and gradient will be drastically reduced for data sets with large features

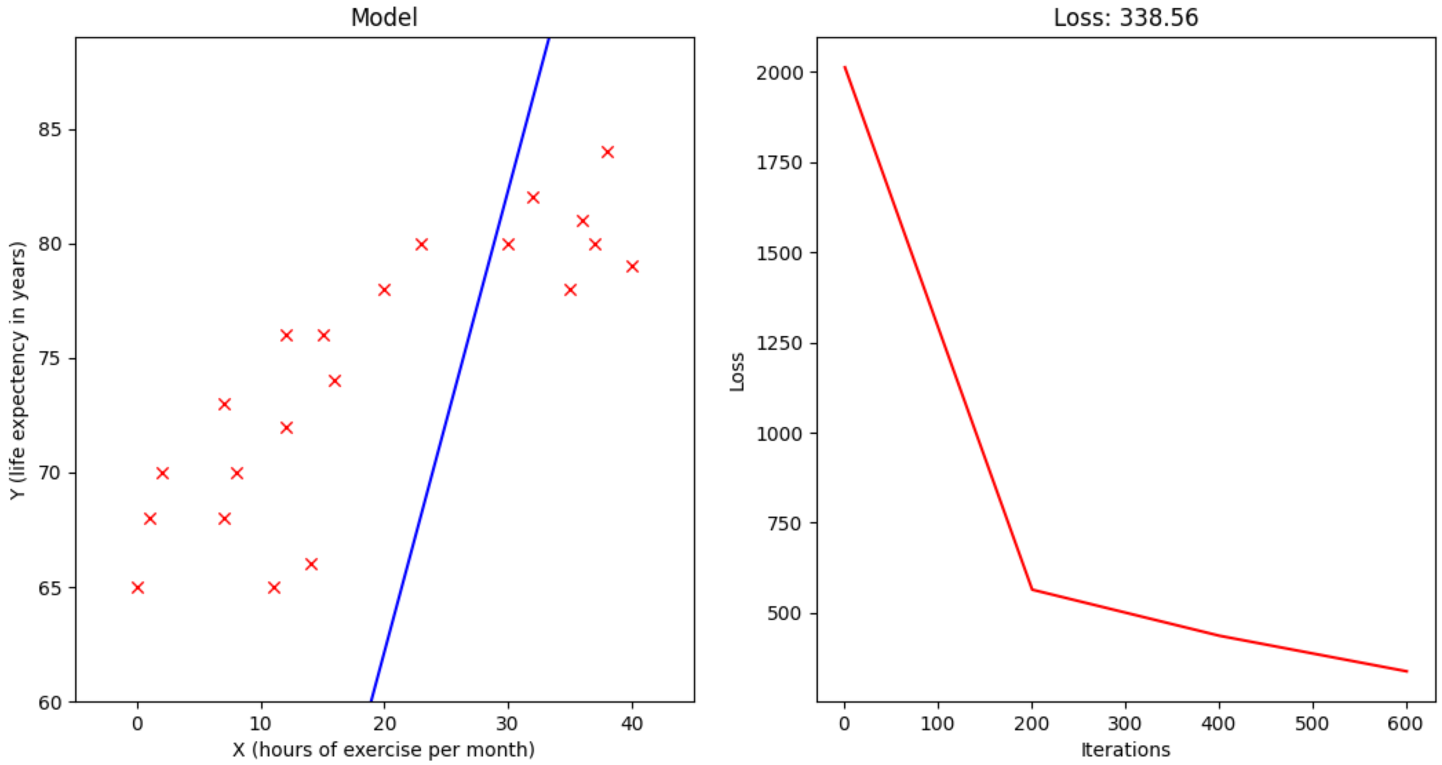
$$w \coloneqq w - \alpha \frac{\partial}{\partial w} loss(w,b)$$

The algorithm's convergence speed will also be increased

$$b \coloneqq b - \alpha \frac{\partial}{\partial b} loss(w,b)$$

# Exercise – Linear Regression Gradient Descent

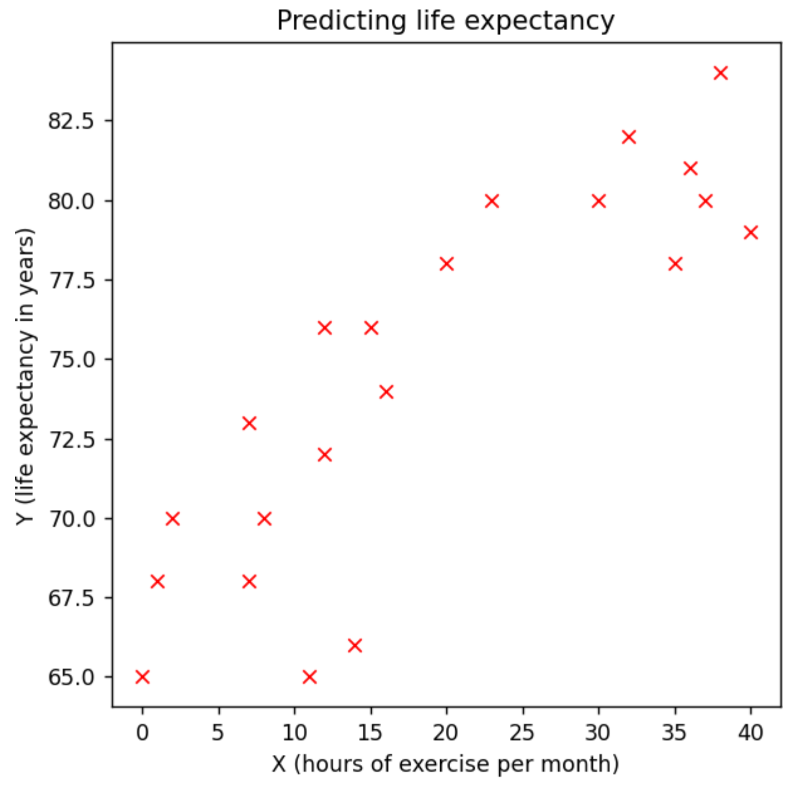
Implement gradient descent to automatically find good $w$ and $b$ parameters starting with the **gradient_descent.py** script.



# Lunch break

# Discrete predictions

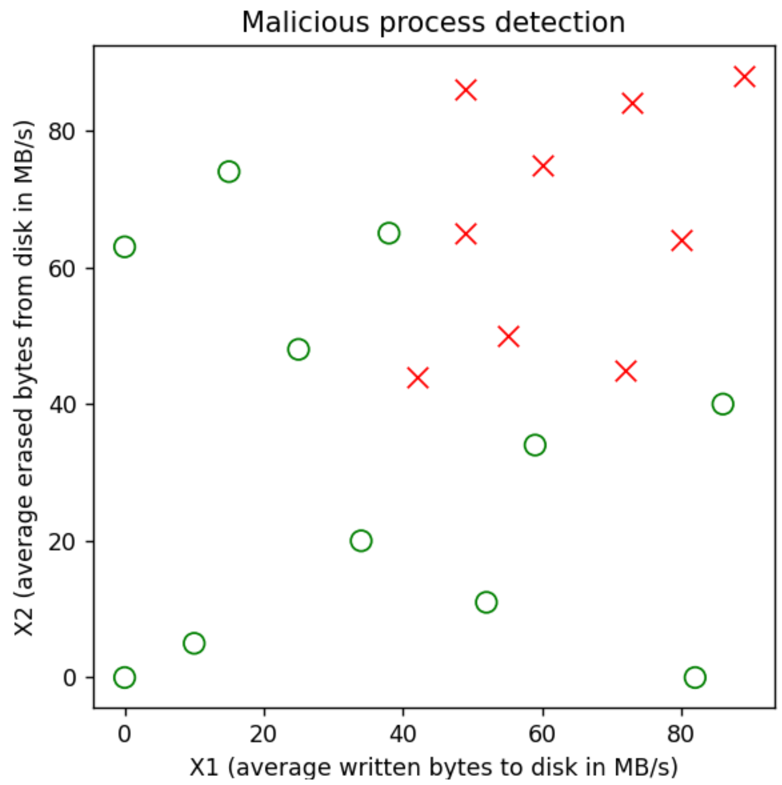
**Linear Regression** allows us to map features to continuous outputs



# Discrete predictions

**Linear Regression** allows us to map features to continuous outputs

What if we want to map to discrete outputs such as 0 and 1?

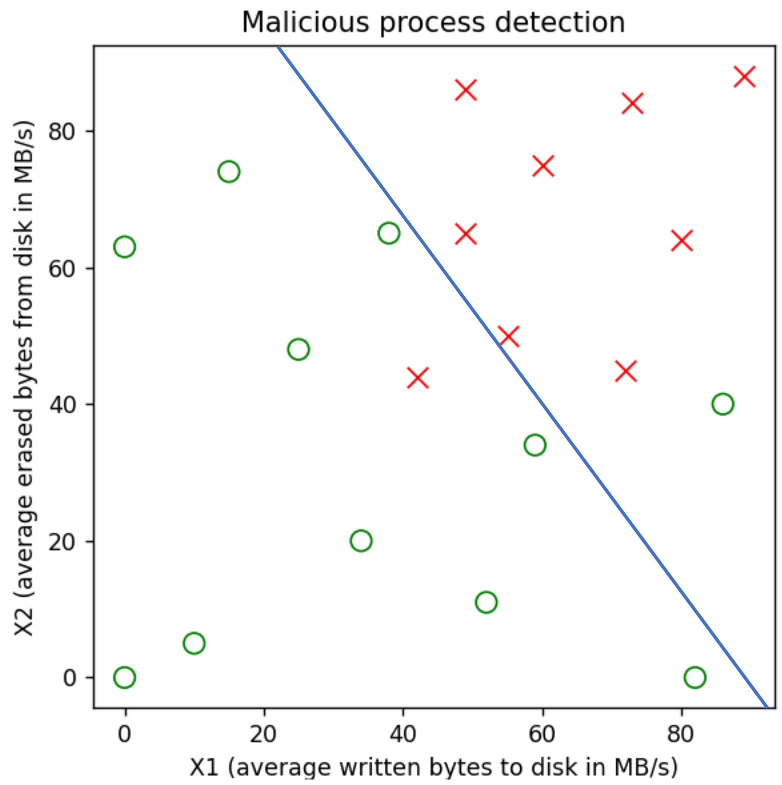


# Discrete predictions

**Linear Regression** allows us to map features to continuous outputs

What if we want to map to discrete outputs such as 0 and 1?

**Logistic Regression** creates a boundary line to separate positives from negatives



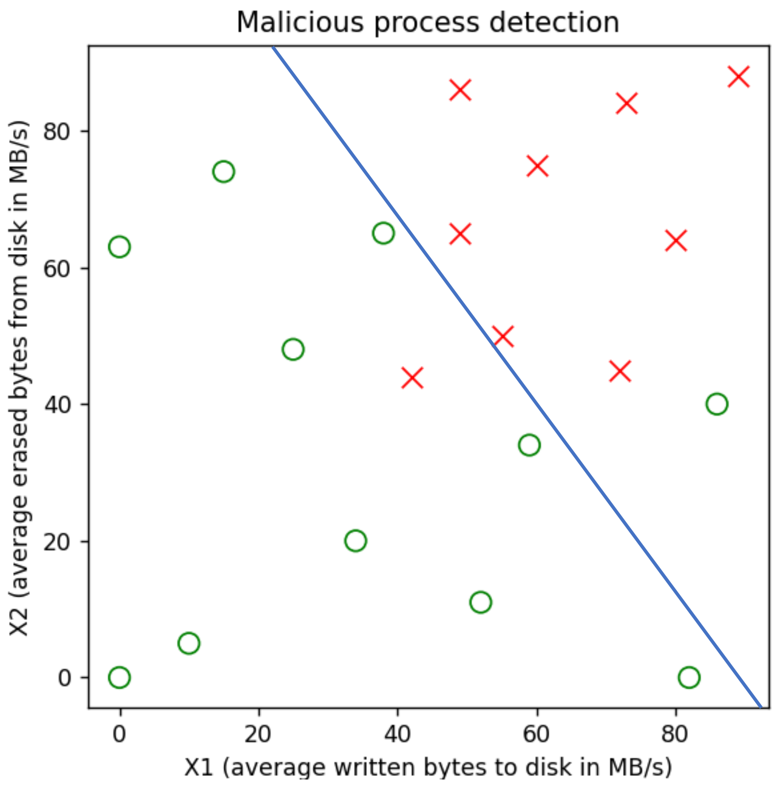
# Logistic Regression

$$X \rightarrow f(X) = w_1x_1 + w_2x_2 + b \rightarrow a(f(X)) \rightarrow \hat{y}$$

features
(disk activity)

model

prediction
(malicious or not)




Calculate the "distance" between $X$ and the decision boundary $f$

Map the "distance" to the interval $(0, 1)$ using the activation function $a$

If $a\big(f(X)\big) < 0.5$ then the prediction is negative $\hat{y} = 0$
otherwise it's positive $\hat{y} = 1$

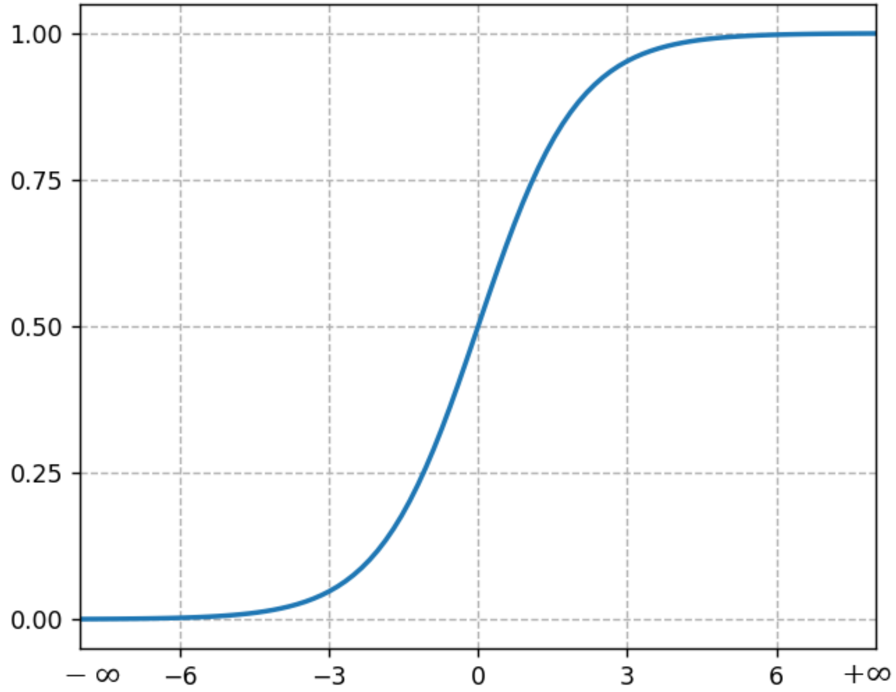
# Logistic Regression

$X \rightarrow f(X) = w_1x_1 + w_2x_2 + b \rightarrow a(f(X)) \rightarrow \hat{y}$

features (disk activity) | model | prediction (malicious or not)

A common activation function is the **sigmoid** activation function

$$a(z) = \frac{1}{e^{-z} + 1}$$



# Logistic Regression – Loss function

**Linear Regression Mean Squared Error** loss function

$$loss(w, b) = \frac{1}{2m}\sum_{i=1}^{m}\left(\hat{y}^{(i)} - y^{(i)}\right)^2$$

# Logistic Regression – Loss function

**Linear Regression Mean Squared Error** loss function

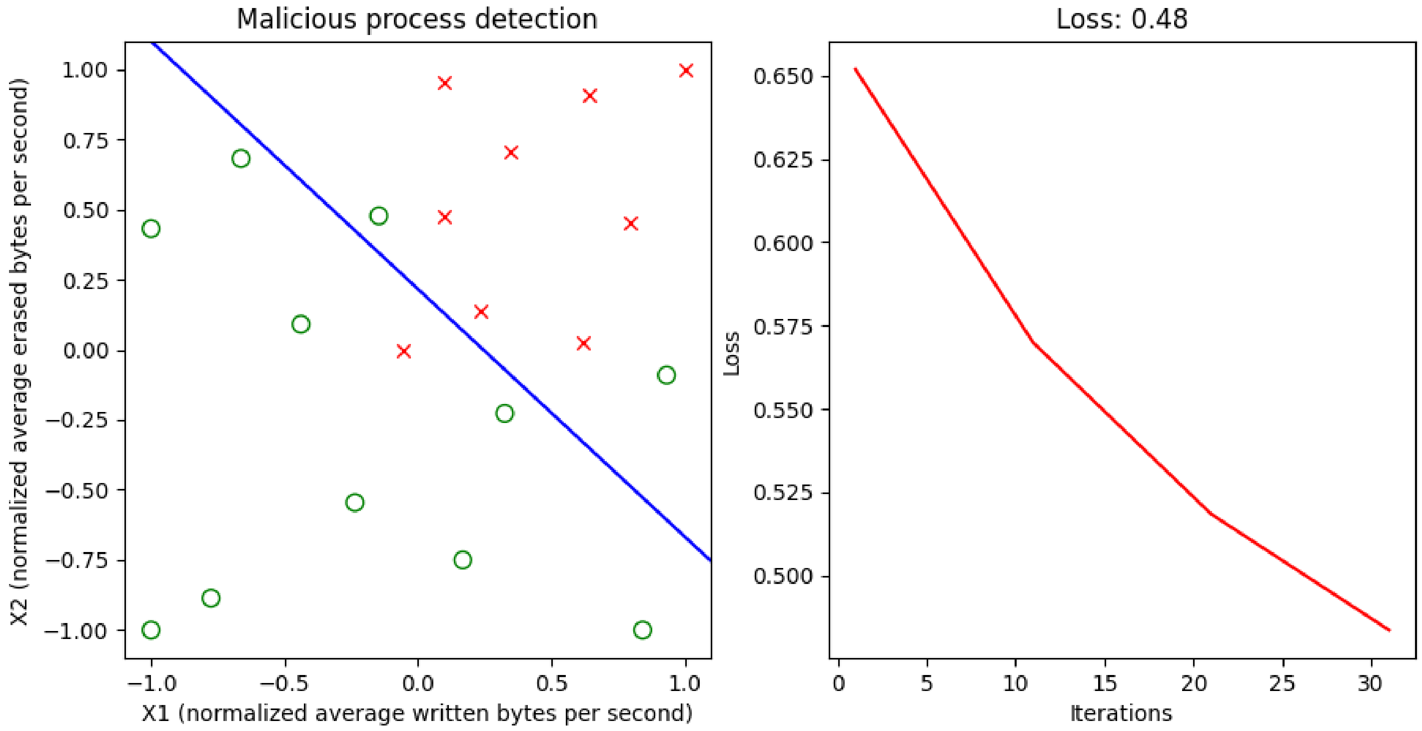
$$loss(w,b) = \frac{1}{2m}\sum_{i=1}^{m}\left(\hat{y}^{(i)} - y^{(i)}\right)^2$$

**Logistic Regression Cross-Entropy** (fancy name, right?) loss function

$$loss(w,b) = -\frac{1}{m}\sum_{i=1}^{m}\left(1 - y^{(i)}\right)\log\left(1 - \hat{y}^{(i)}\right) + y^{(i)}\log\left(\hat{y}^{(i)}\right)$$

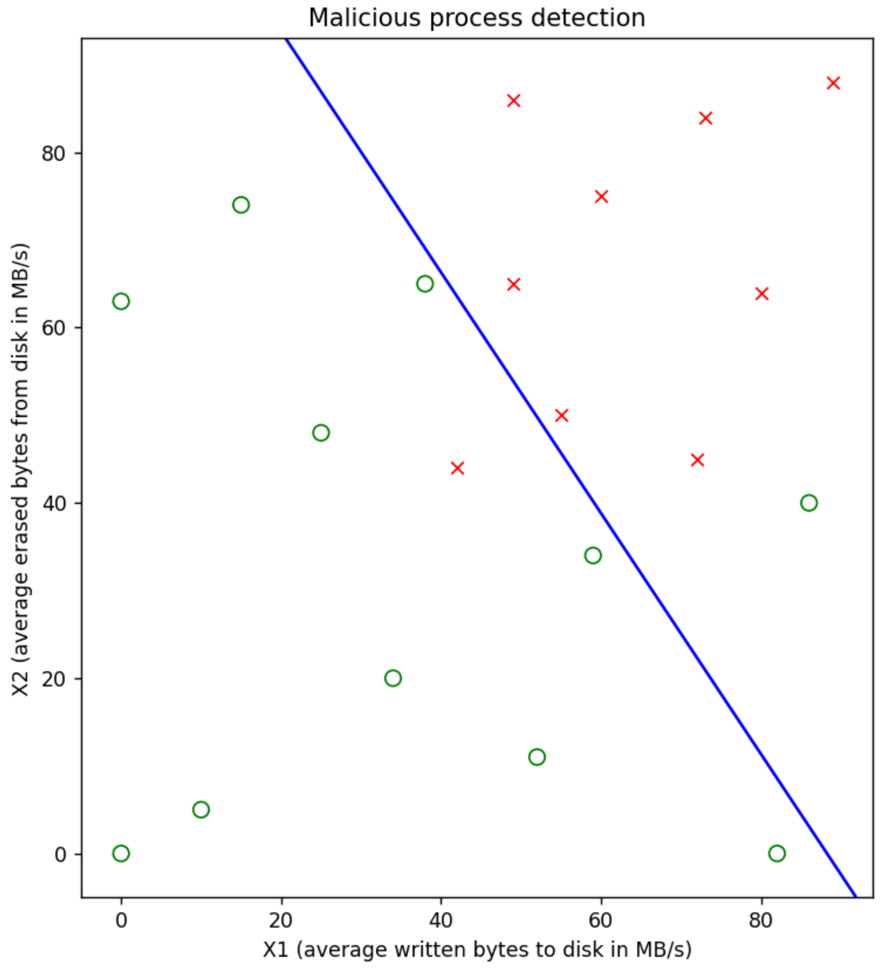
# Exercise – Logistic Regression Gradient Descent

Implement gradient descent for Logistic Regression starting with the **logistic_regression.py** script. Note that the data has been normalized for you.



# Logistic Regression – Can we do better?

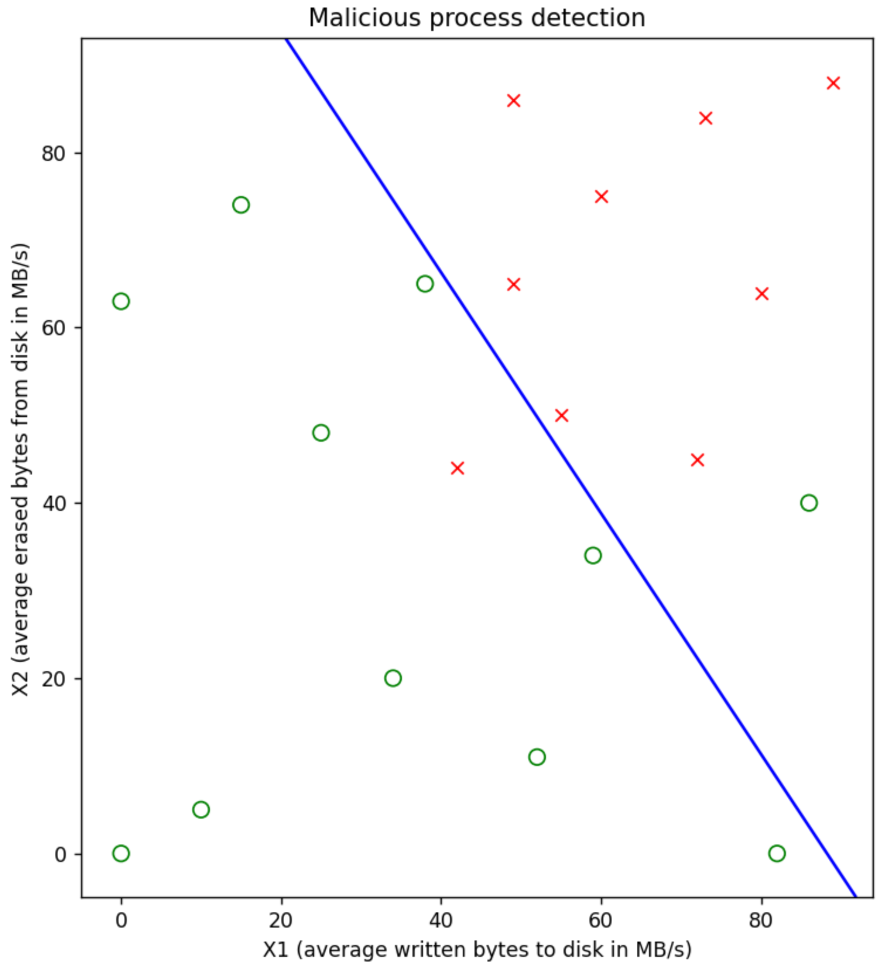
Our simple model doesn't fit the data well.



# Logistic Regression – Can we do better?

Our simple model doesn't fit the data well.

We can add the features $x_1^2$, $x_1 x_2$ and $x_2^2$ to get a better fit. Our boundary will be a polynomial and thus be more flexible than a simple line.

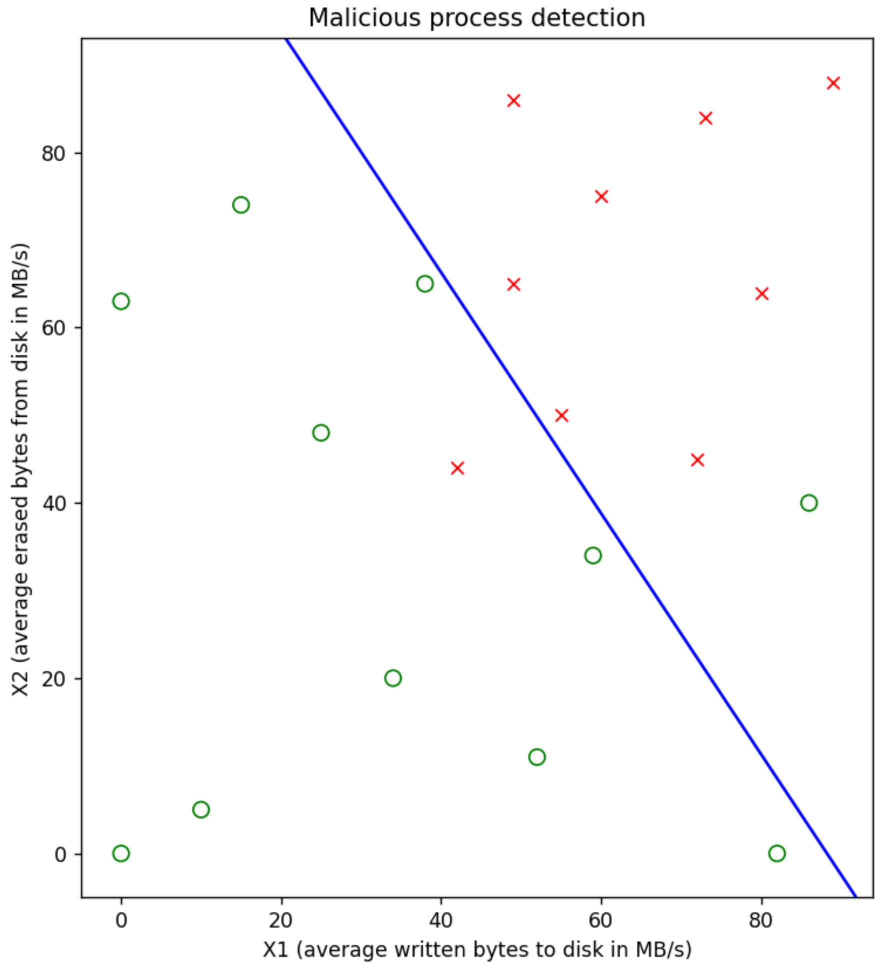


# Logistic Regression – Can we do better?

Our simple model doesn't fit the data well.

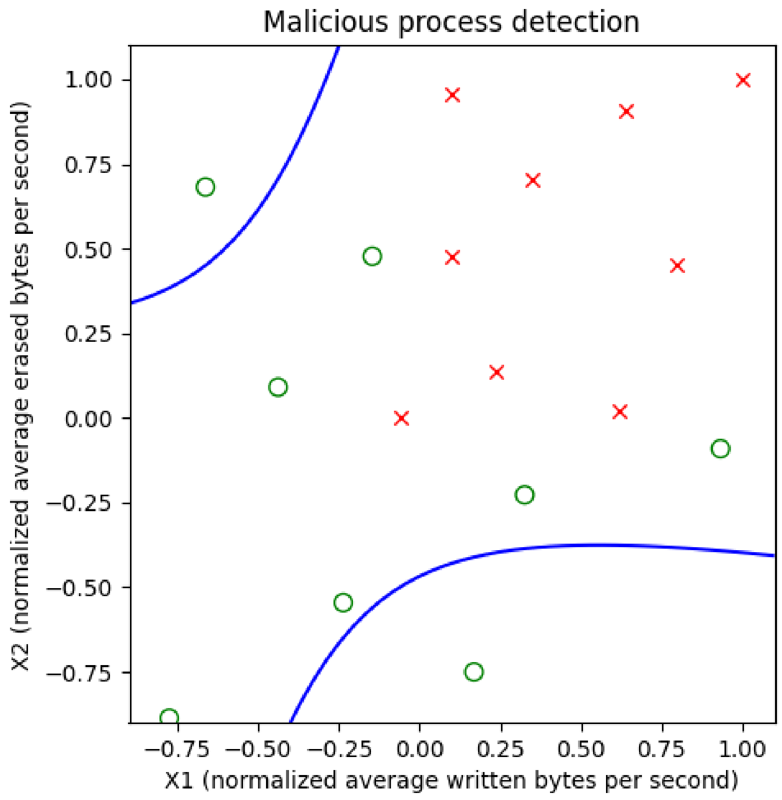
We can add the features $x_1^2$, $x_1 x_2$ and $x_2^2$ to get a better fit. Our boundary will be a polynomial and thus be more flexible than a simple line.

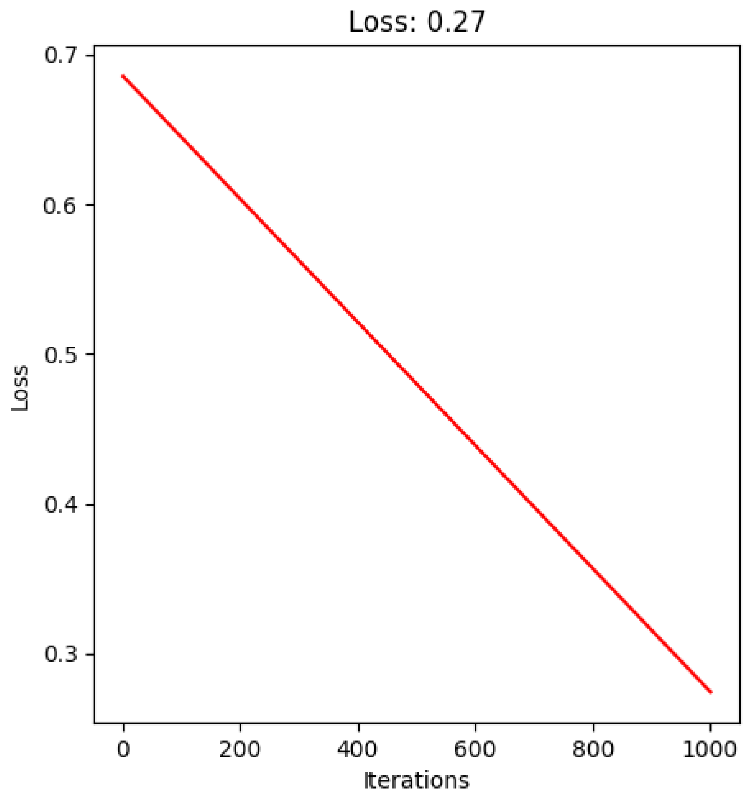
We are not adding new features such as the number of files modified but instead reuse the features $x_1$ and $x_2$ to keep the dimensionality to 2D. This way we can better visualize the data. In a real-life scenario adding new features is common.



# Exercise – Logistic Polynomial Regression

Implement logistic polynomial regression starting with the **polynomial.py** script to obtain a better fit of the data. Note that the extra features have been synthesized for you.



# Homework – Polynomial Regression

Implement gradient descent with polynomial regression starting with the **polynomial.py** script file to obtain a better fit of the life expectancy data set previously covered by Linear Regression. Try different degrees for the polynomial to gain intuition how the model becomes more flexible.

**Congratulations!**

**You just finished day #1 of this course!**

**See you tomorrow for day #2!**

# Disadvantages of Linear and Logistic Regression

# Disadvantages of Linear and Logistic Regression

We must choose polynomial features manually for complex data sets ($x_1^2$, $x_1 x_2$ …)

# Disadvantages of Linear and Logistic Regression

We must choose polynomial features manually for complex data sets ($x_1^2$, $x_1 x_2$ …)

How can we scale better for more complex data sets?

# Disadvantages of Linear and Logistic Regression

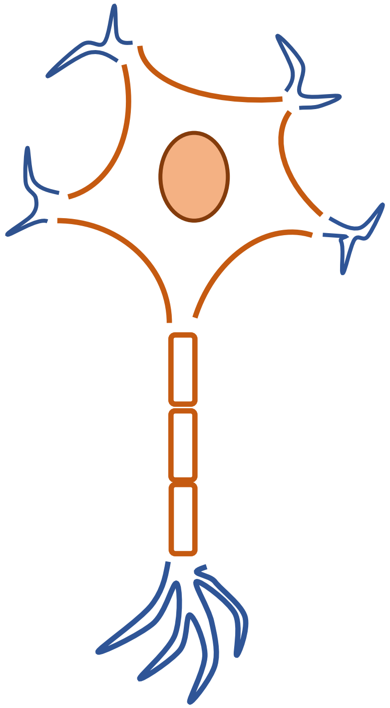
We must choose polynomial features manually for complex data sets ($x_1^2$, $x_1 x_2$ …)

How can we scale better for more complex data sets?

This shortcoming is addressed by **Artificial Neural Networks**

# Artificial Neural Networks

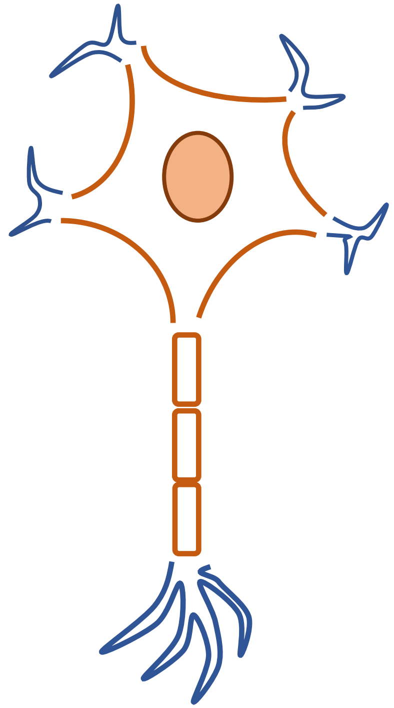
**Artificial Neural Networks** are inspired from biological neurons



# Artificial Neural Networks

**Artificial Neural Networks** are inspired from biological neurons

A nerve cell can receive impulses from other nerve cells and send its own impulses

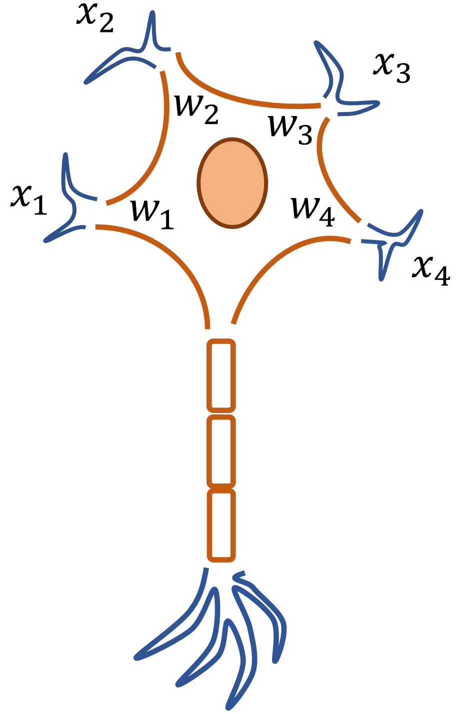


# Artificial Neural Networks

**Artificial Neural Networks** are inspired from biological neurons

A nerve cell can receive impulses from other nerve cells and send its own impulses

Can be modeled mathematically as the dot product between the output of other neurons and the weights of the current neuron and its activation




$$a(w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b)$$

# Artificial Neural Networks – Layering

We can use $\mathbb{R}^{m \times n}$ matrices to model multiple neurons

Below is the activation of an entire layer of $m$ neurons taking as input the output of the previous layer of $n$ neurons

$$A = a\left(\begin{pmatrix} w_{1,1} & \cdots & w_{1,n} \\ \vdots & \ddots & \vdots \\ w_{m,1} & \cdots & w_{m,n} \end{pmatrix}\begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix} + \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix}\right) = \begin{pmatrix} a(w_{1,1}a_1 + \cdots + w_{1,n}a_n + b_1) \\ \vdots \\ a(w_{m,1}a_1 + \cdots + w_{m,n}a_n + b_m) \end{pmatrix}$$

Note that if we have only one neuron then we end up with the **Logistic Regression** model

# Artificial Neural Networks – Layering

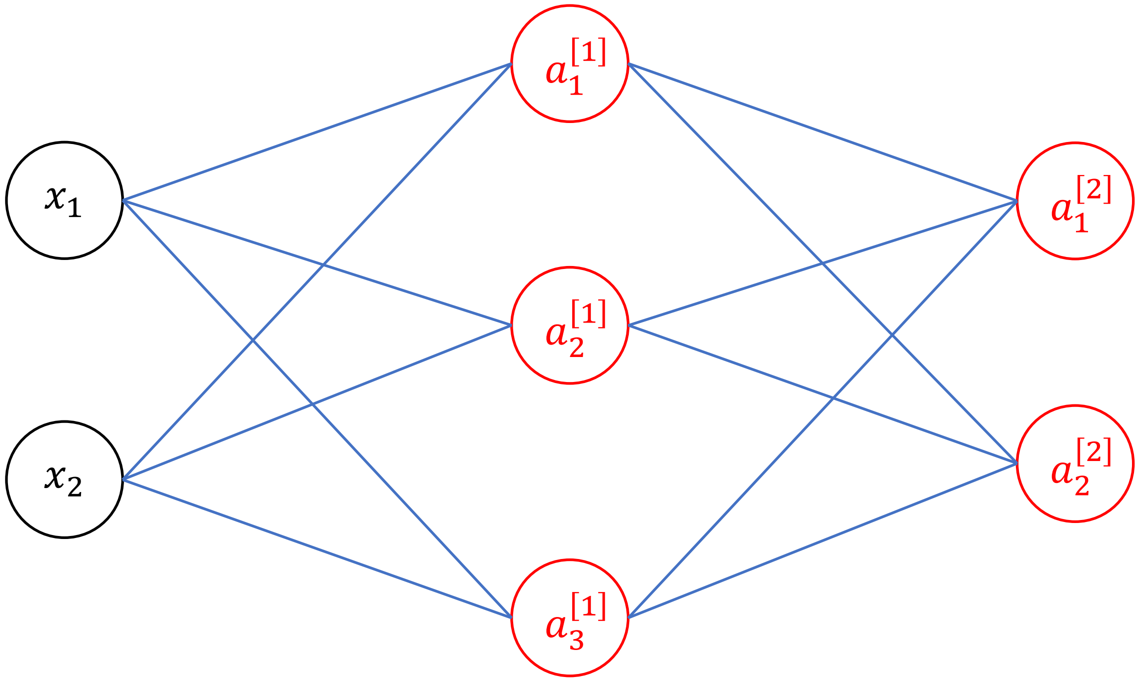
We can generalize the activation of an arbitrary layer $l$

$$A^{[l]} = a^{[l]}(W^{[l]}A^{[l-1]} + B^{[l]})$$

$A^{[l]}$ is the $\mathbb{R}^m$ activation vector of the current layer $l$ with $m$ neurons
$W^{[l]}$ is the $\mathbb{R}^{m \times n}$ weights matrix for the current layer $l$
$A^{[l-1]}$ is the $\mathbb{R}^n$ activation vector of the previous layer $l-1$ with $n$ neurons
$B^{[l]}$ is the $\mathbb{R}^m$ bias vector for the current layer $l$
$a^{[l]}$ is the activation function for the current layer $l$

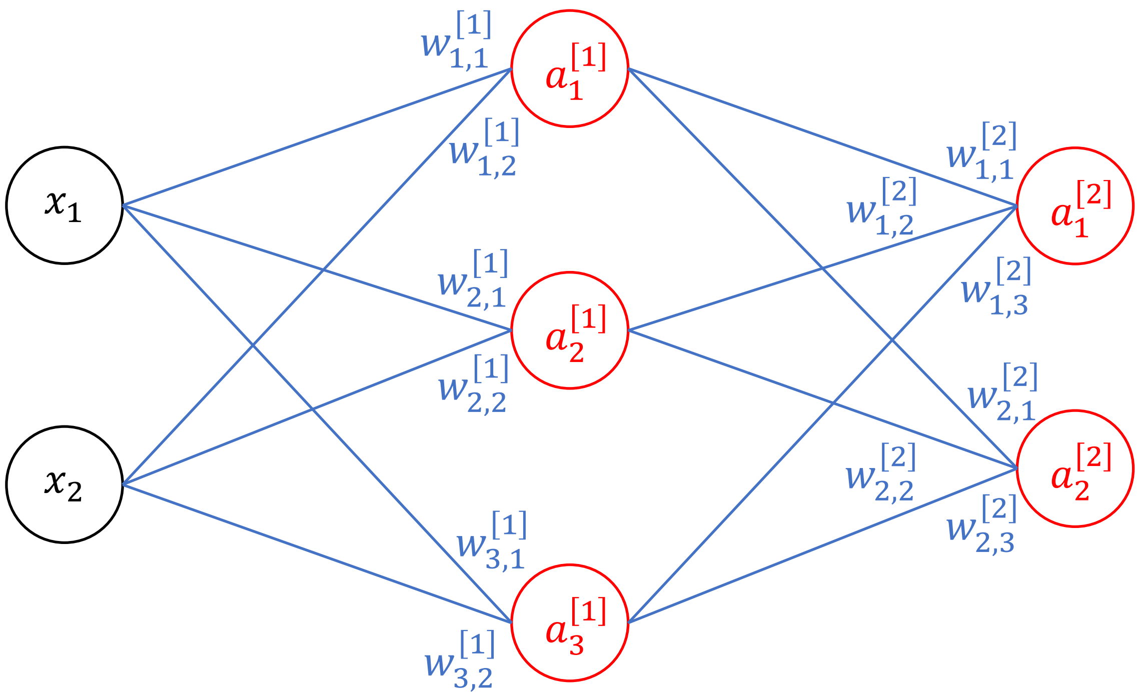
Note that $A^{[0]}$ is the feature vector $X$ from the data set

# Artificial Neural Networks – Layering




$a_n^{[l]}$ is the activation of the $n^{th}$ neuron from layer $l$

# Artificial Neural Networks – Layering




$a_n^{[l]}$ is the activation of the $n^{th}$ neuron from layer $l$

$w_{m,n}^{[l]}$ is the weight connecting the $n^{th}$ neuron from layer $l-1$ to the $m^{th}$ neuron from layer $l$

# Artificial Neural Networks – Forward Propagation

Evaluating the activation of each layer starting from the data set features and ending with the last layer is called **Forward Propagation**

Below is the **Forward Propagation** for a 3-layer neural network

$$
\begin{aligned}
A^{[1]} &= a^{[1]}(W^{[1]}X + B^{[1]}) \\
A^{[2]} &= a^{[2]}(W^{[2]}A^{[1]} + B^{[2]}) \\
A^{[3]} &= a^{[3]}(W^{[3]}A^{[2]} + B^{[3]})
\end{aligned}
$$

# Artificial Neural Networks – Forward Propagation

The activation function of the last layer decides the output of the neural network.

# Artificial Neural Networks – Forward Propagation

The activation function of the last layer decides the output of the neural network.

If we want the neural network to output $\mathbb{R}^n$ vectors we can set the activation function to be the identity function. This is the generalization of **Linear Regression** for multiple outputs.

$$a^{[L]}(Z)_i = z_i$$

# Artificial Neural Networks – Forward Propagation

The activation function of the last layer decides the output of the neural network.

If we want the neural network to output $\mathbb{R}^n$ vectors we can set the activation function to be the identity function. This is the generalization of **Linear Regression** for multiple outputs.

$$a^{[L]}(Z)_i = z_i$$

If we want the output $\mathbb{R}^n$ to be a probability distribution such that $0 \leq a^{[L]}(Z)_i \leq 1$ and $\sum_{i=1}^{n} a^{[L]}(Z)_i = 1$, then we can use the **softmax** activation function. This is the generalization of **Logistic Regression** for multiple outputs.

$$a^{[L]}(Z)_i = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}$$

# Artificial Neural Networks – Activation function

Why do we need activation functions for each intermediate layer?

# Artificial Neural Networks – Activation function

Why do we need activation functions for each intermediate layer?

We need to break the linearity of the model otherwise the input can be mapped to the output using only one transformation matrix regardless of how many layers we have.

# Artificial Neural Networks – Activation function

Why do we need activation functions for each intermediate layer?

We need to break the linearity of the model otherwise the input can be mapped to the output using only one transformation matrix regardless of how many layers we have.

The **sigmoid** activation for intermediate layers can lead to issues such as gradient saturation and slow convergence for neural networks with many layers.

# Artificial Neural Networks – Activation function

Why do we need activation functions for each intermediate layer?

We need to break the linearity of the model otherwise the input can be mapped to the output using only one transformation matrix regardless of how many layers we have.

The **sigmoid** activation for intermediate layers can lead to issues such as gradient saturation and slow convergence for neural networks with many layers.

These issues are addressed by the $\boldsymbol{ReLU}$ activation function.

$$ReLU(Z)_i = \begin{cases} z_i & z_i > 0 \\ 0 & z_i \leq 0 \end{cases}$$

# Artificial Neural Networks – Loss function

We can generalize the **Mean Squared Error** loss function for $n$ continuous outputs…

$$loss\left(W^{[1]}, \dots, W^{[L]}, B^{[1]}, \dots, B^{[L]}\right) = \frac{1}{2m} \sum_{i=1}^{m} \sum_{j=1}^{n} \left(\hat{Y}_j^{(i)} - Y_j^{(i)}\right)^2$$

# Artificial Neural Networks – Loss function

We can generalize the **Mean Squared Error** loss function for $n$ continuous outputs…

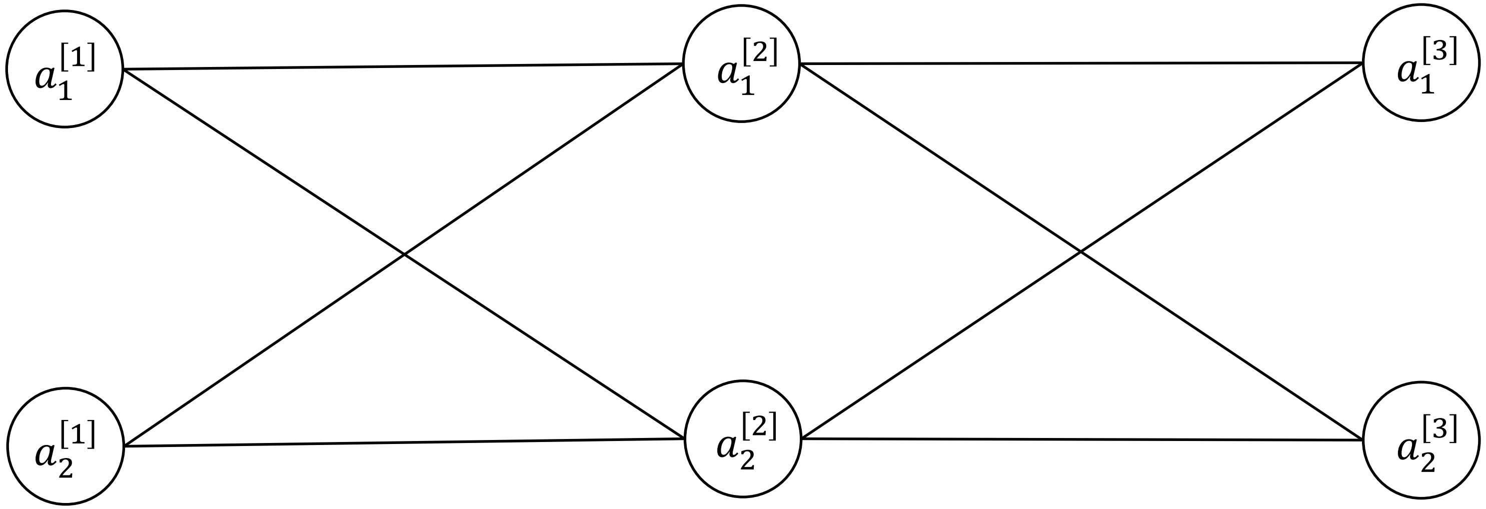
$$loss\left(W^{[1]}, \dots, W^{[L]}, B^{[1]}, \dots, B^{[L]}\right) = \frac{1}{2m} \sum_{i=1}^{m} \sum_{j=1}^{n} \left(\hat{Y}_j^{(i)} - Y_j^{(i)}\right)^2$$

And the **Cross-Entropy** loss function for $n$ discrete outputs

$$loss\left(W^{[1]}, \dots, W^{[L]}, B^{[1]}, \dots, B^{[L]}\right) = -\frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{n} Y_j^{(i)} \log\left(\hat{Y}_j^{(i)}\right)$$

## Artificial Neural Networks – Backpropagation

**Backpropagation** computes the partial derivative of each weight and bias parameter, layer by layer, from output to input, regarding the loss function.



# Artificial Neural Networks – Backpropagation

**Backpropagation** computes the partial derivative of each weight and bias parameter, layer by layer, from output to input, regarding the loss function.

$$\frac{\partial loss}{\partial W_{1,*}^{[3]}} = \frac{\partial a_1^{[3]}}{\partial W_{1,*}^{[3]}} \frac{\partial loss}{\partial a_1^{[3]}}$$

$a_1^{[1]}$ $a_1^{[2]}$ $a_1^{[3]}$

$$\frac{\partial loss}{\partial W_{2,*}^{[3]}} = \frac{\partial a_2^{[3]}}{\partial W_{2,*}^{[3]}} \frac{\partial loss}{\partial a_2^{[3]}}$$

$a_2^{[1]}$ $a_2^{[2]}$ $a_2^{[3]}$

# Artificial Neural Networks – Backpropagation

**Backpropagation** computes the partial derivative of each weight and bias parameter, layer by layer, from output to input, regarding the loss function.

$$\frac{\partial loss}{\partial W_{1,*}^{[2]}} = \frac{\partial a_1^{[2]}}{\partial W_{1,*}^{[2]}} \overbrace{\left( \frac{\partial a_1^{[3]}}{\partial a_1^{[2]}} \frac{\partial loss}{\partial a_1^{[3]}} + \frac{\partial a_2^{[3]}}{\partial a_1^{[2]}} \frac{\partial loss}{\partial a_2^{[3]}} \right)}^{\frac{\partial loss}{\partial a_1^{[2]}}}$$

$$\frac{\partial loss}{\partial W_{1,*}^{[3]}} = \frac{\partial a_1^{[3]}}{\partial W_{1,*}^{[3]}} \frac{\partial loss}{\partial a_1^{[3]}}$$

$a_1^{[1]}$ $a_1^{[2]}$ $a_1^{[3]}$

$$\frac{\partial loss}{\partial W_{2,*}^{[2]}} = \frac{\partial a_2^{[2]}}{\partial W_{2,*}^{[2]}} \overbrace{\left( \frac{\partial a_1^{[3]}}{\partial a_2^{[2]}} \frac{\partial loss}{\partial a_1^{[3]}} + \frac{\partial a_2^{[3]}}{\partial a_2^{[2]}} \frac{\partial loss}{\partial a_2^{[3]}} \right)}^{\frac{\partial loss}{\partial a_2^{[2]}}}$$

$$\frac{\partial loss}{\partial W_{2,*}^{[3]}} = \frac{\partial a_2^{[3]}}{\partial W_{2,*}^{[3]}} \frac{\partial loss}{\partial a_2^{[3]}}$$

$a_2^{[1]}$ $a_2^{[2]}$ $a_2^{[3]}$

# Artificial Neural Networks – Backpropagation

**Backpropagation** computes the partial derivative of each weight and bias parameter, layer by layer, from output to input, regarding the loss function.

$$\frac{\partial loss}{\partial W^{[1]}_{1,*}} = \frac{\partial a_1^{[1]}}{\partial W^{[1]}_{1,*}} \left( \overbrace{\frac{\partial a_1^{[2]}}{\partial a_1^{[1]}} \frac{\partial loss}{\partial a_1^{[2]}} + \frac{\partial a_2^{[2]}}{\partial a_1^{[1]}} \frac{\partial loss}{\partial a_2^{[2]}}}^{\frac{\partial loss}{\partial a_1^{[1]}}} \right)$$

$a_1^{[1]}$

$$\frac{\partial loss}{\partial W^{[2]}_{1,*}} = \frac{\partial a_1^{[2]}}{\partial W_{1,*}^{[2]}} \left( \overbrace{\frac{\partial a_1^{[3]}}{\partial a_1^{[2]}} \frac{\partial loss}{\partial a_1^{[3]}} + \frac{\partial a_2^{[3]}}{\partial a_1^{[2]}} \frac{\partial loss}{\partial a_2^{[3]}}}^{\frac{\partial loss}{\partial a_1^{[2]}}} \right)$$

$a_1^{[2]}$

$$\frac{\partial loss}{\partial W^{[3]}_{1,*}} = \frac{\partial a_1^{[3]}}{\partial W^{[3]}_{1,*}} \frac{\partial loss}{\partial a_1^{[3]}}$$

$a_1^{[3]}$

$$\frac{\partial loss}{\partial W^{[1]}_{2,*}} = \frac{\partial a_2^{[1]}}{\partial W^{[1]}_{2,*}} \left( \overbrace{\frac{\partial a_1^{[2]}}{\partial a_2^{[1]}} \frac{\partial loss}{\partial a_1^{[2]}} + \frac{\partial a_2^{[2]}}{\partial a_2^{[1]}} \frac{\partial loss}{\partial a_2^{[2]}}}^{\frac{\partial loss}{\partial a_2^{[1]}}} \right)$$

$a_2^{[1]}$

$$\frac{\partial loss}{\partial W^{[2]}_{2,*}} = \frac{\partial a_2^{[2]}}{\partial W^{[2]}_{2,*}} \left( \overbrace{\frac{\partial a_1^{[3]}}{\partial a_2^{[2]}} \frac{\partial loss}{\partial a_1^{[3]}} + \frac{\partial a_2^{[3]}}{\partial a_2^{[2]}} \frac{\partial loss}{\partial a_2^{[3]}}}^{\frac{\partial loss}{\partial a_2^{[2]}}} \right)$$

$a_2^{[2]}$

$$\frac{\partial loss}{\partial W^{[3]}_{2,*}} = \frac{\partial a_2^{[3]}}{\partial W^{[3]}_{2,*}} \frac{\partial loss}{\partial a_2^{[3]}}$$

$a_2^{[3]}$

# Artificial Neural Networks – Backpropagation

The same logic applies for biases by replacing $W_{k,*}^{[l]}$ with $B_k^{[l]}$

$$\frac{\partial loss}{\partial B_1^{[1]}} = \frac{\partial a_1^{[1]}}{\partial B_1^{[1]}} \left( \overbrace{\frac{\partial a_1^{[2]}}{\partial a_1^{[1]}} \frac{\partial loss}{\partial a_1^{[2]}} + \frac{\partial a_2^{[2]}}{\partial a_1^{[1]}} \frac{\partial loss}{\partial a_2^{[2]}}}^{\frac{\partial loss}{\partial a_1^{[1]}}} \right)$$

$a_1^{[1]}$

$$\frac{\partial loss}{\partial B_1^{[2]}} = \frac{\partial a_1^{[2]}}{\partial B_1^{[2]}} \left( \overbrace{\frac{\partial a_1^{[3]}}{\partial a_1^{[2]}} \frac{\partial loss}{\partial a_1^{[3]}} + \frac{\partial a_2^{[3]}}{\partial a_1^{[2]}} \frac{\partial loss}{\partial a_2^{[3]}}}^{\frac{\partial loss}{\partial a_1^{[2]}}} \right)$$

$a_1^{[2]}$

$$\frac{\partial loss}{\partial B_1^{[3]}} = \frac{\partial a_1^{[3]}}{\partial B_1^{[3]}} \frac{\partial loss}{\partial a_1^{[3]}}$$

$a_1^{[3]}$

$$\frac{\partial loss}{\partial B_2^{[1]}} = \frac{\partial a_2^{[1]}}{\partial B_2^{[1]}} \left( \overbrace{\frac{\partial a_1^{[2]}}{\partial a_2^{[1]}} \frac{\partial loss}{\partial a_1^{[2]}} + \frac{\partial a_2^{[2]}}{\partial a_2^{[1]}} \frac{\partial loss}{\partial a_2^{[2]}}}^{\frac{\partial loss}{\partial a_2^{[1]}}} \right)$$

$a_2^{[1]}$

$$\frac{\partial loss}{\partial B_2^{[2]}} = \frac{\partial a_2^{[2]}}{\partial B_2^{[2]}} \left( \overbrace{\frac{\partial a_1^{[3]}}{\partial a_2^{[2]}} \frac{\partial loss}{\partial a_1^{[3]}} + \frac{\partial a_2^{[3]}}{\partial a_2^{[2]}} \frac{\partial loss}{\partial a_2^{[3]}}}^{\frac{\partial loss}{\partial a_2^{[2]}}} \right)$$

$a_2^{[2]}$

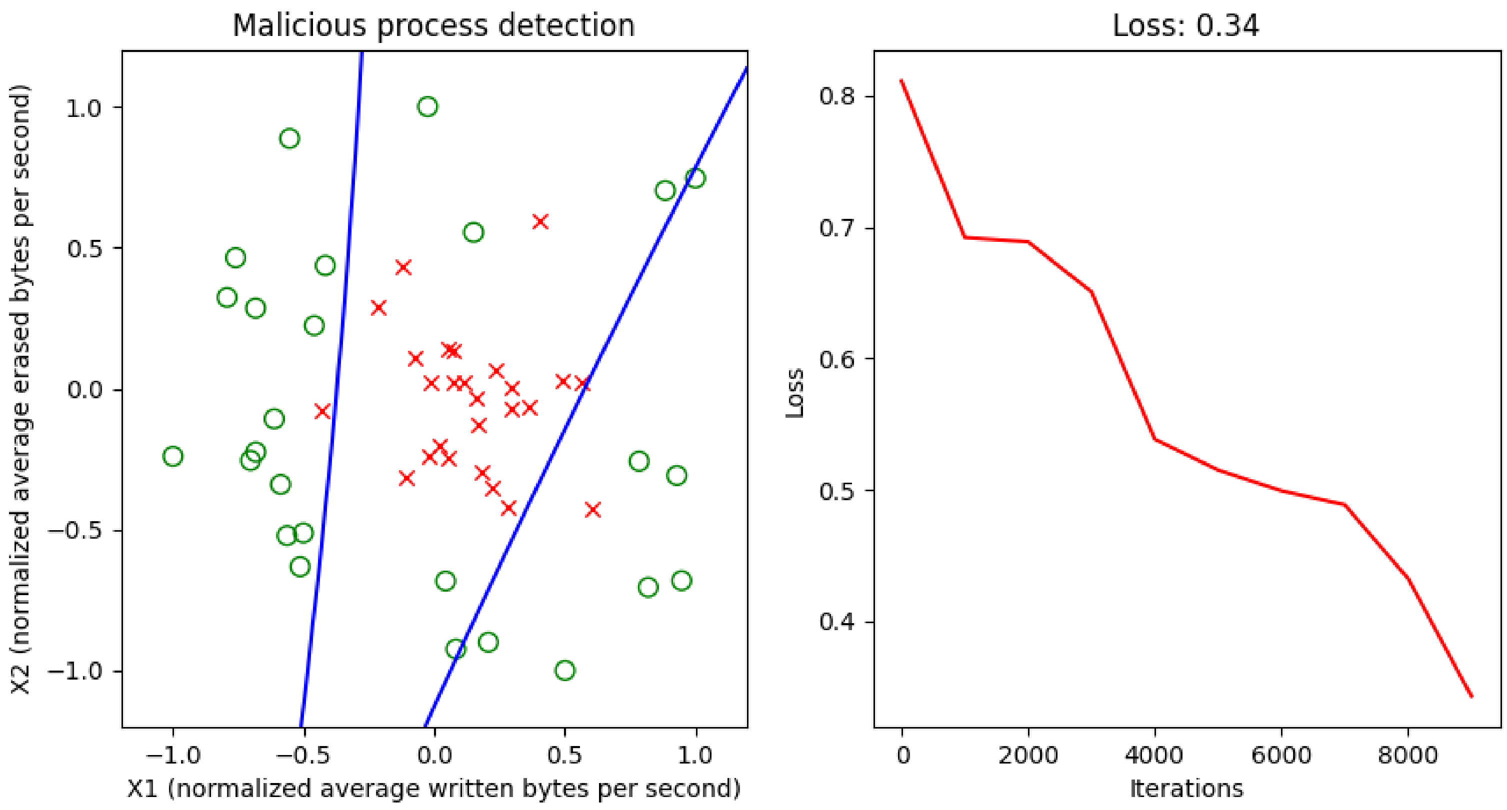
$$\frac{\partial loss}{\partial B_2^{[3]}} = \frac{\partial a_2^{[3]}}{\partial B_2^{[3]}} \frac{\partial loss}{\partial a_2^{[3]}}$$

$a_2^{[3]}$

# Lunch break

# Exercise – Artificial Neural Networks

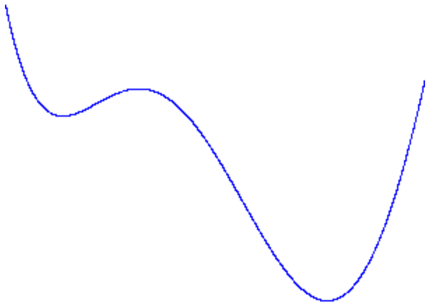
Implement and train an artificial neural network starting with the **neural.py** script file.



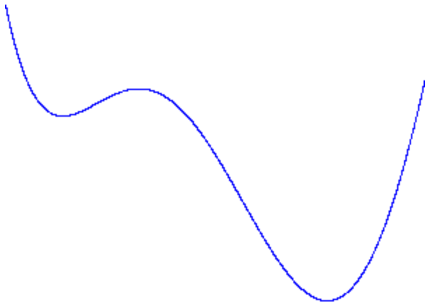
# Improvements to Gradient Descent - Momentum

Keep a part of the previous gradient as momentum

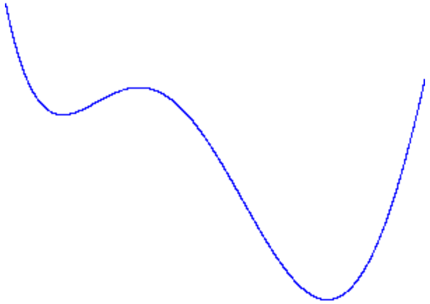
0% momentum



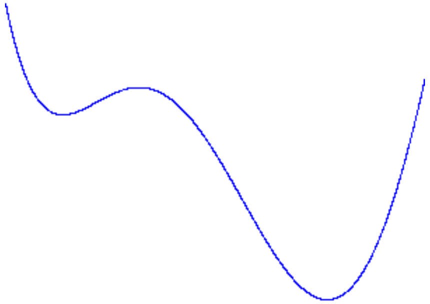
90% momentum



85% momentum



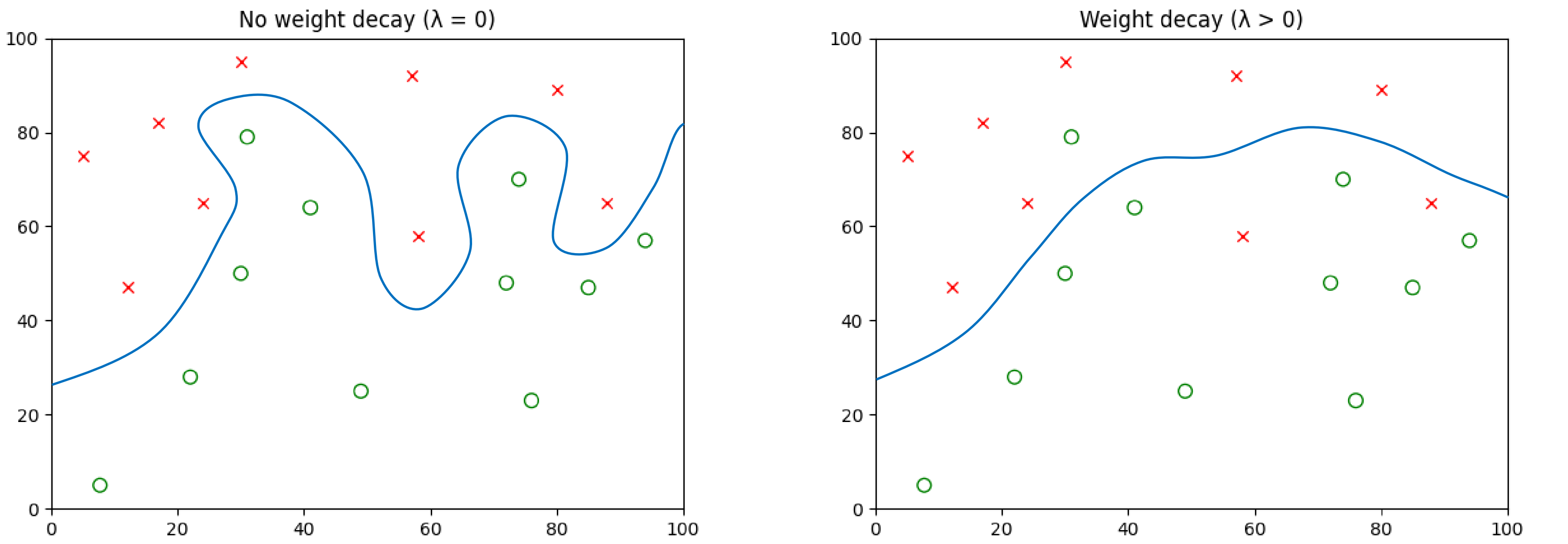
99% momentum



# Improvements to Gradient Descent – Weight Decay

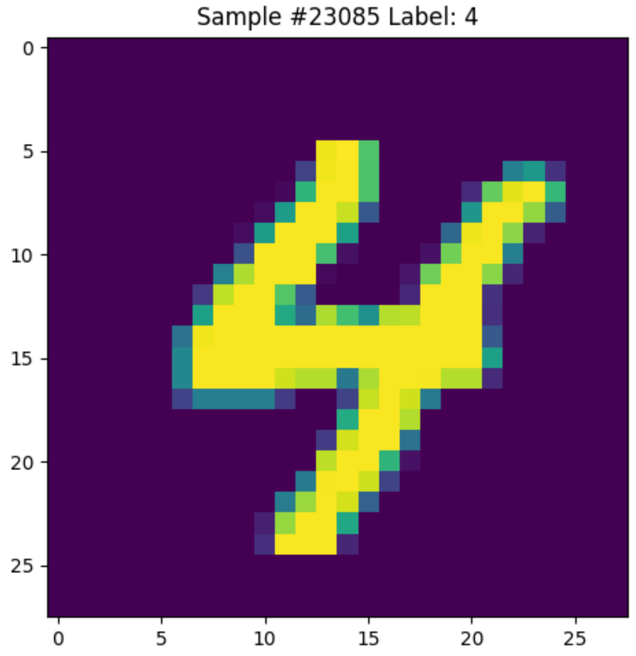
The model can generalize better if the magnitude of the weight parameters is reduced

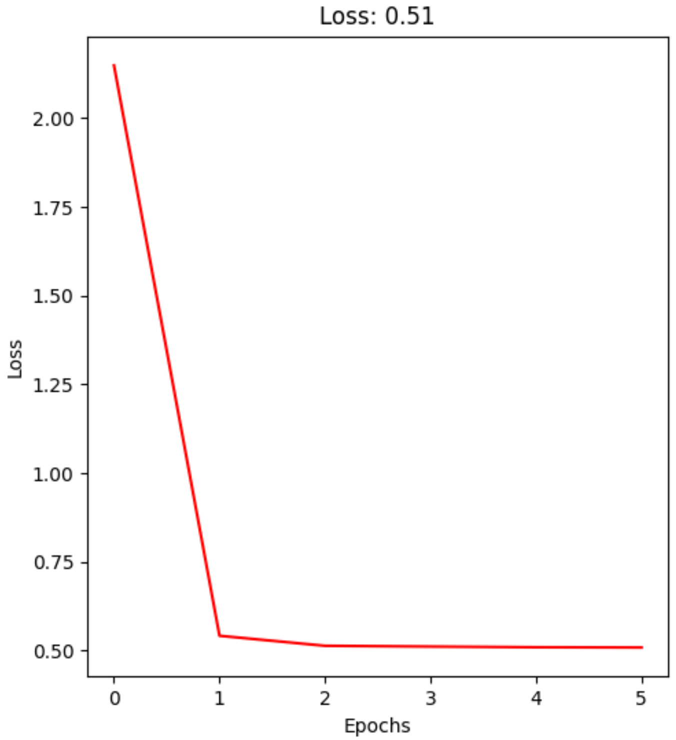
$$loss = -\frac{1}{m}\sum_{i=1}^{m}\left(1 - y^{(i)}\right)\log\left(1 - \hat{y}^{(i)}\right) + y^{(i)}\log\left(\hat{y}^{(i)}\right) + \frac{\lambda}{2}\sum W^2$$



## Exercise – Digit Recognition

Implement and train an artificial neural network to recognize digits starting with the **digit.py** script file. Test your artificial neural network model by classifying digits drawn by you or by your colleagues.



**Congratulations!**

**You have finished the course!**

# Derivations – Single class classification

$$\frac{\partial \hat{y}}{\partial z} = \frac{\partial}{\partial z}\left(\frac{1}{e^{-z}+1}\right) = -(e^{-z}+1)^{-2}e^{-z}(-1) = \frac{e^{-z}}{(e^{-z}+1)^2} = \frac{e^{-z}+1-1}{(e^{-z}+1)^2} = \frac{1}{e^{-z}+1} - \left(\frac{1}{e^{-z}+1}\right)^2 = \hat{y} - \hat{y}^2 = \hat{y}(1-\hat{y})$$

$$\frac{\partial loss}{\partial z} = \frac{\partial \hat{y}}{\partial z}\frac{\partial loss}{\partial \hat{y}} = \hat{y}^{(i)}(1-\hat{y}^{(i)})\frac{\partial}{\partial \hat{y}}\left(-\frac{1}{m}\sum_{i=1}^{m}\left(\left(1-y^{(i)}\right)\log\left(1-\hat{y}^{(i)}\right) + y^{(i)}\log\left(\hat{y}^{(i)}\right)\right)\right) =$$

$$= -\frac{1}{m}\sum_{i=1}^{m}\left(\left(1-y^{(i)}\right)\frac{1}{1-\hat{y}^{(i)}}(-1)\hat{y}^{(i)}\left(1-\hat{y}^{(i)}\right) + y^{(i)}\frac{1}{\hat{y}^{(i)}}\hat{y}^{(i)}\left(1-\hat{y}^{(i)}\right)\right) =$$

$$= -\frac{1}{m}\sum_{i=1}^{m}\left(\left(y^{(i)}-1\right)\hat{y}^{(i)} + y^{(i)}\left(1-\hat{y}^{(i)}\right)\right) = -\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}\hat{y}^{(i)} - \hat{y}^{(i)} + y^{(i)} - y^{(i)}\hat{y}^{(i)}\right) =$$

$$= \frac{1}{m}\sum_{i=1}^{m}\left(\hat{y}^{(i)} - y^{(i)}\right)$$

# Derivations – Multi class classification

$$i = j, \qquad \frac{\partial \hat{Y}_i}{\partial Z_j} = \frac{\partial}{\partial Z_j}\left(\frac{e^{Z_j}}{\sum_{k=1}^{n} e^{Z_k}}\right) = \frac{e^{Z_j}\sum_{k=1}^{n} e^{Z_k} - \left(e^{Z_j}\right)^2}{\left(\sum_{k=1}^{n} e^{Z_k}\right)^2} = \frac{e^{Z_j}}{\sum_{k=1}^{n} e^{Z_k}} - \left(\frac{e^{Z_j}}{\sum_{k=1}^{n} e^{Z_k}}\right)^2 = \hat{Y}_j - \hat{Y}_j^2 = \hat{Y}_j(1-\hat{Y}_j)$$

$$i \neq j, \qquad \frac{\partial \hat{Y}_i}{\partial Z_j} = \frac{\partial}{\partial Z_j}\left(\frac{e^{Z_i}}{\sum_{k=1}^{n} e^{Z_k}}\right) = -\frac{e^{Z_i}e^{Z_j}}{\left(\sum_{k=1}^{n} e^{Z_k}\right)^2} = -\frac{e^{Z_i}}{\sum_{k=1}^{n} e^{Z_k}}\frac{e^{Z_j}}{\sum_{k=1}^{n} e^{Z_k}} = -\hat{Y}_i\hat{Y}_j$$

$$\frac{\partial loss}{\partial Z_j} = \frac{\partial}{\partial Z_j}\left(-\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{n} Y_k^{(i)}\log\left(\hat{Y}_k^{(i)}\right)\right) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k\neq j}^{n} Y_k^{(i)}\frac{\partial}{\partial Z_j}\log\left(\hat{Y}_k^{(i)}\right) - \frac{1}{m}\sum_{i=1}^{m} Y_j^{(i)}\frac{\partial}{\partial Z_j}\log\left(\hat{Y}_j^{(i)}\right) =$$

$$= -\frac{1}{m}\sum_{i=1}^{m}\sum_{k\neq j}^{n} Y_k^{(i)}\frac{1}{\hat{Y}_k^{(i)}}\left(-\hat{Y}_k^{(i)}\hat{Y}_j^{(i)}\right) - \frac{1}{m}\sum_{i=1}^{m} Y_j^{(i)}\frac{1}{\hat{Y}_j^{(i)}}\hat{Y}_j^{(i)}\left(1-\hat{Y}_j^{(i)}\right) =$$

$$= \frac{1}{m}\sum_{i=1}^{m}\sum_{k\neq j}^{n} Y_k^{(i)}\hat{Y}_j^{(i)} - \frac{1}{m}\sum_{i=1}^{m}\left(Y_j^{(i)} - Y_j^{(i)}\hat{Y}_j^{(i)}\right) = \frac{1}{m}\sum_{i=1}^{m}\sum_{k\neq j}^{n} Y_k^{(i)}\hat{Y}_j^{(i)} + \frac{1}{m}\sum_{i=1}^{m} Y_j^{(i)}\hat{Y}_j^{(i)} - \frac{1}{m}\sum_{i=1}^{m} Y_j^{(i)} =$$

$$= \frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{n} Y_k^{(i)}\hat{Y}_j^{(i)} - \frac{1}{m}\sum_{i=1}^{m} Y_j^{(i)} = \frac{1}{m}\sum_{i=1}^{m}\left(\hat{Y}_j^{(i)} - Y_j^{(i)}\right)$$