Introduction to Machine Learning

Daniel Sandu

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?

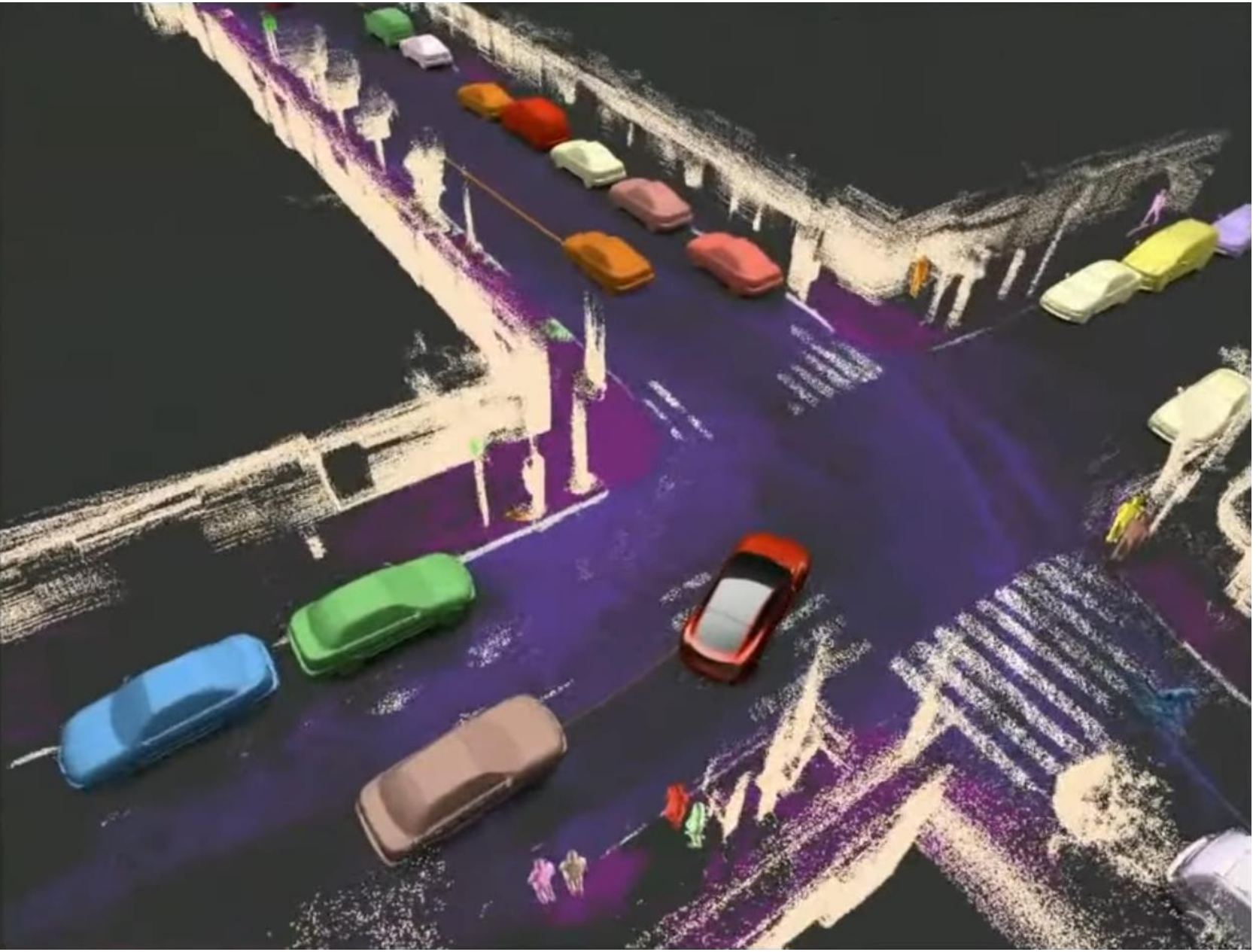
A couple of examples…

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 succeed…

Objective of this course

Design machine learning algorithms starting from almost nothing

What you'll need to succeed…

- High school mathematics – linear algebra and derivatives

Objective of this course

Design machine learning algorithms starting from almost nothing

What you'll need to succeed…

- 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 succeed…

- 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

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

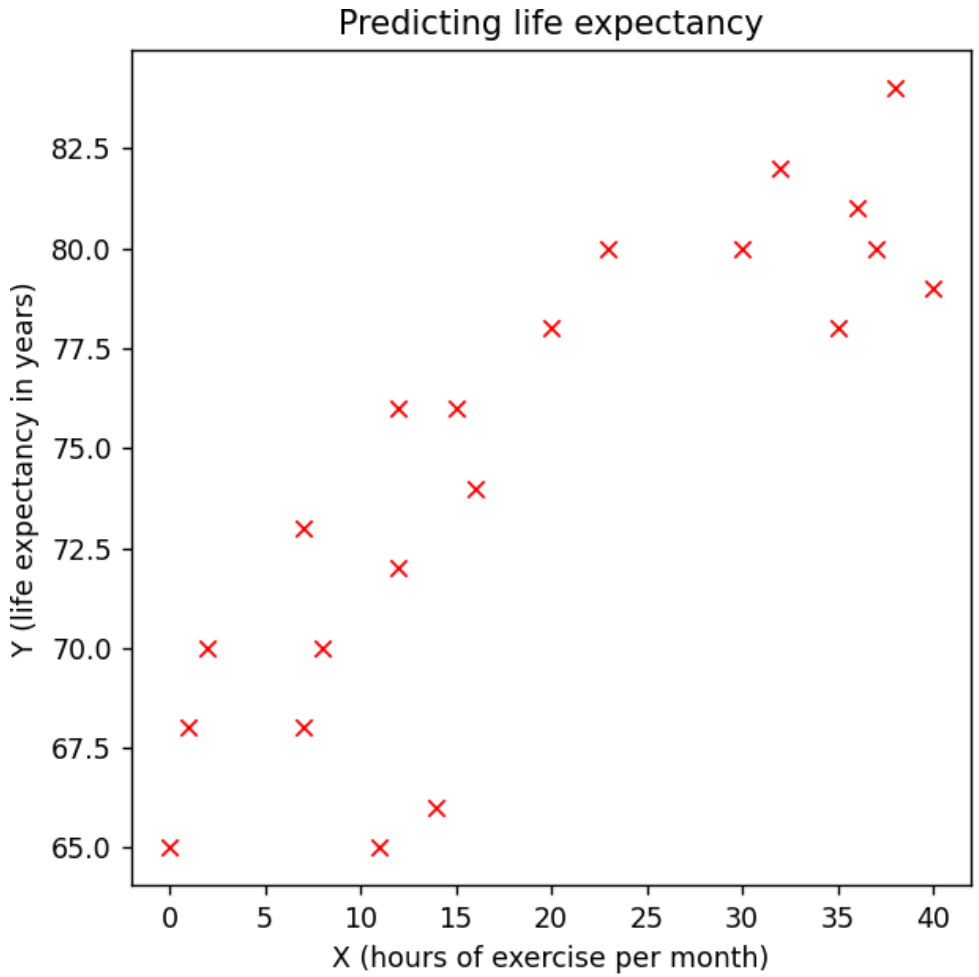
We'll be focusing on **supervised learning** in this course

Supervised learning

Linear Regression

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

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



Linear Regression

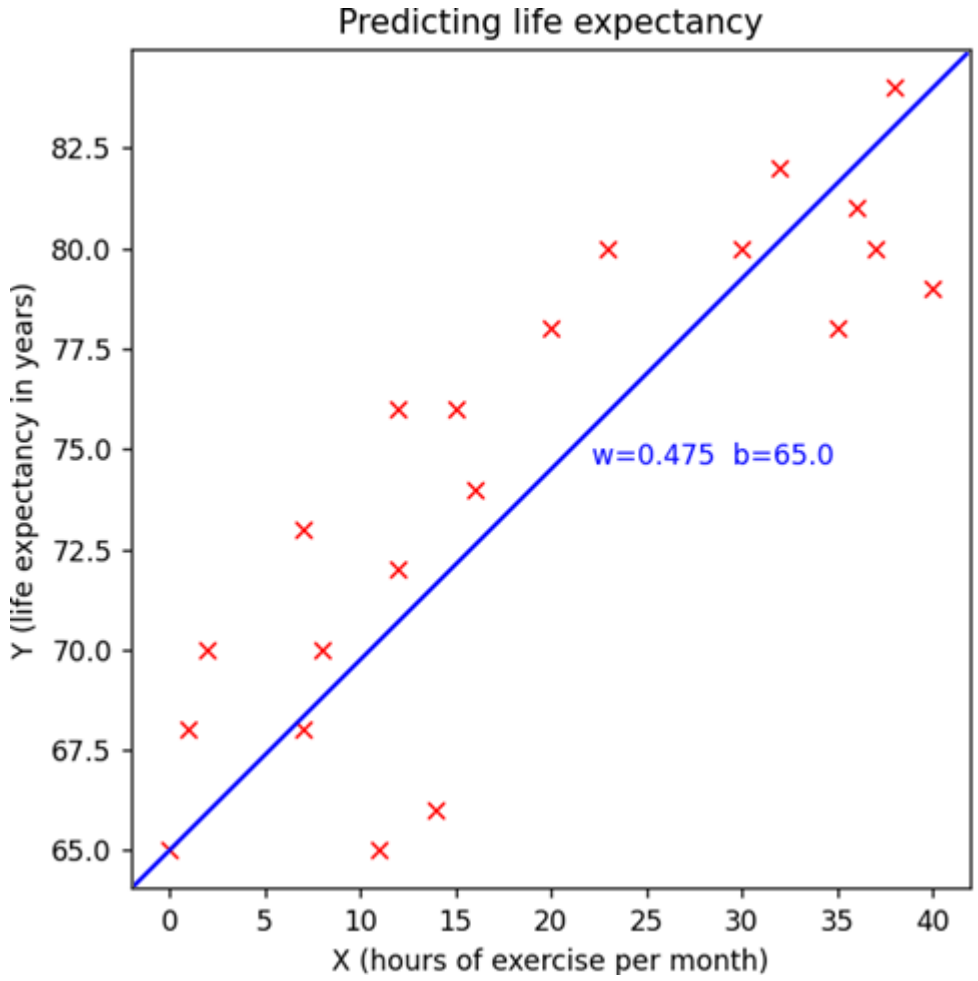
x → $f(x) = wx + b$ → $\hat{y}$

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.



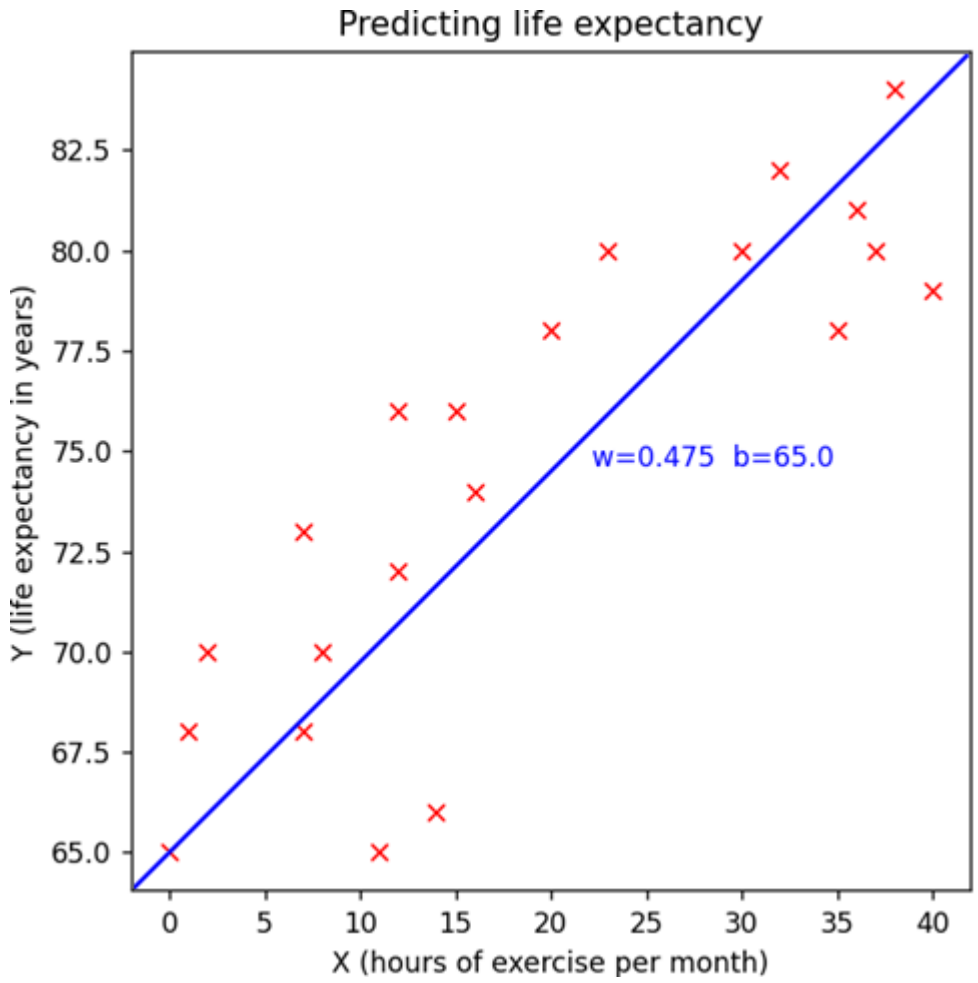
Linear Regression

Questions

What happens if we change w?

What about b?

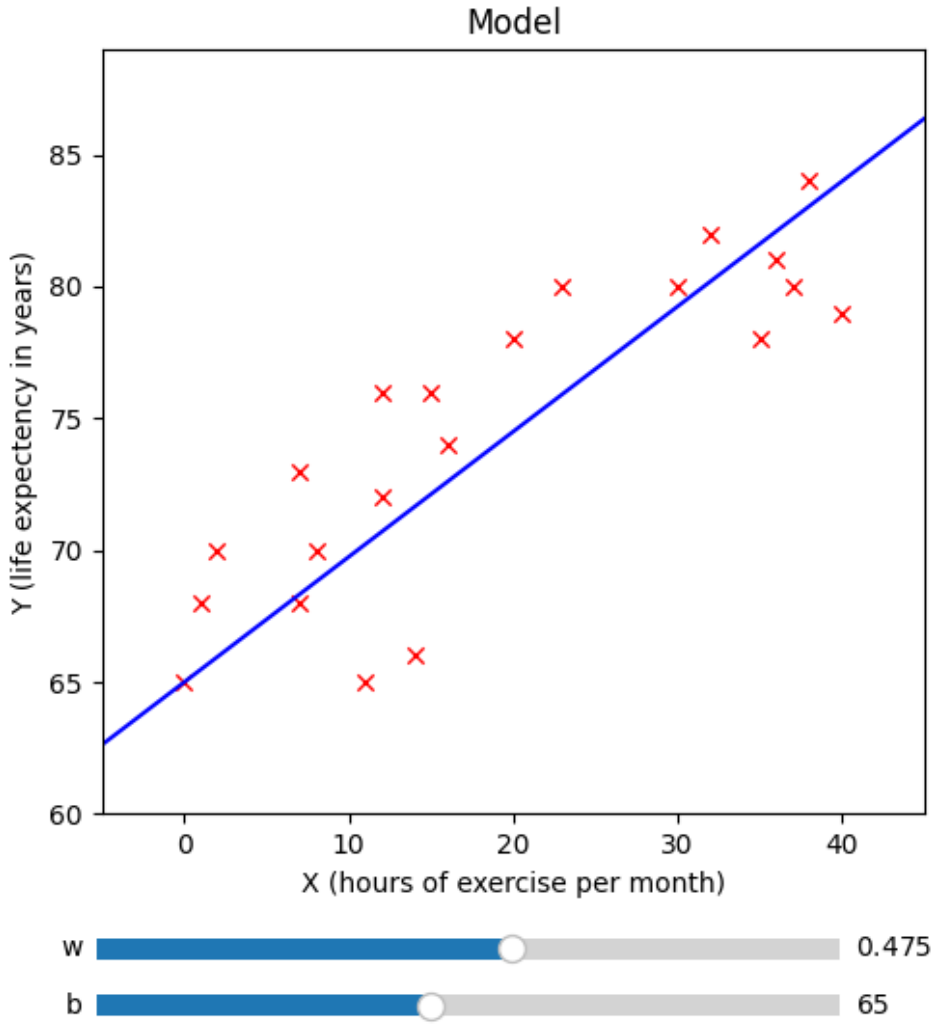
Can we change w and b automatically to obtain a good model?



Linear Regression

Exercise

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



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 move

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 – Squared error loss function

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

What are the benefits of this loss function?

Linear Regression – 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 way 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 – 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

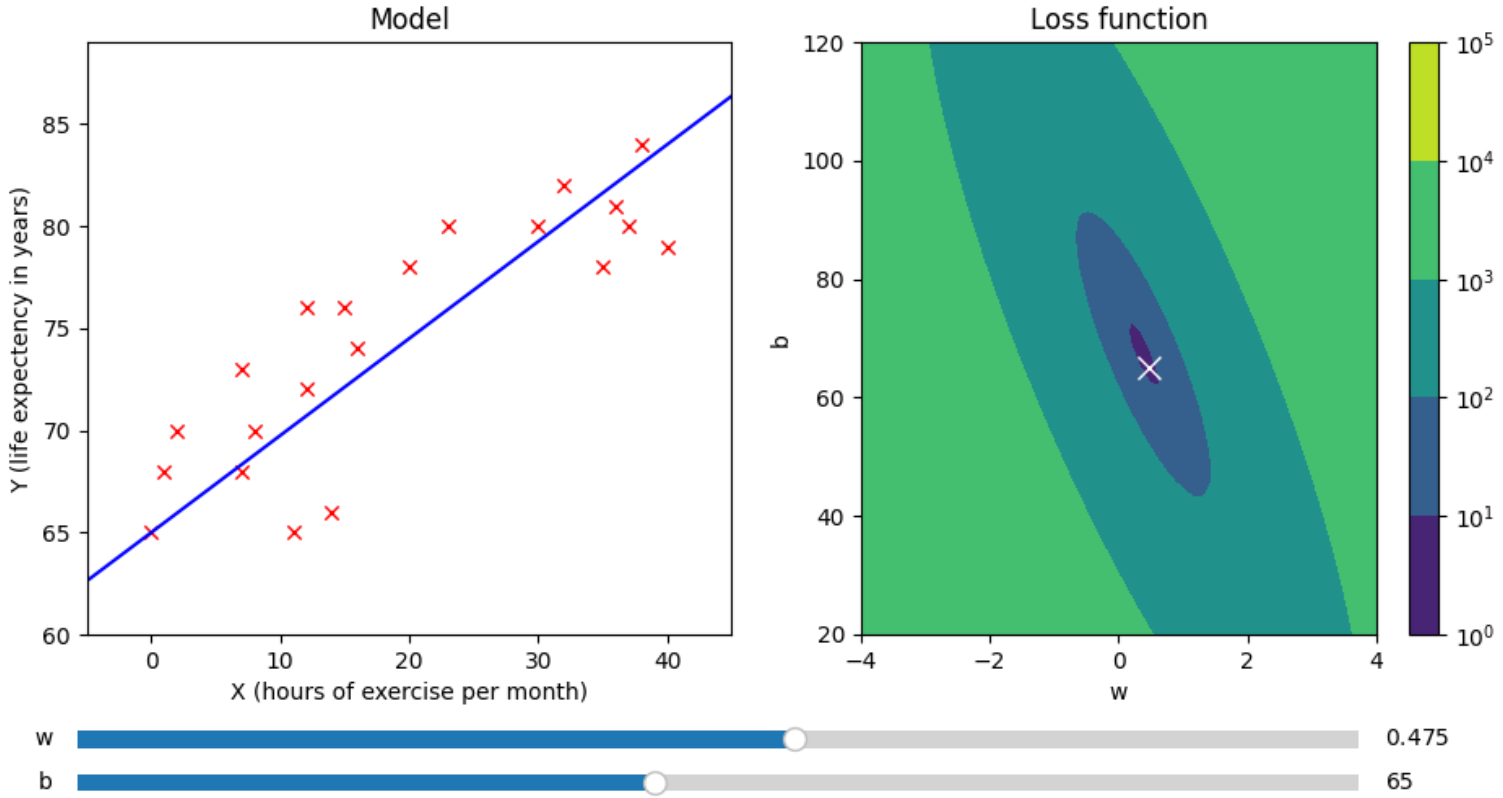
$$loss(w,b) = \frac{1}{2m}\sum_{i=1}^{m}\left(\hat{y}^{(i)} - y^{(i)}\right)^2$$

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

Linear Regression – Squared error loss function

Exercise

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



Linear Regression – Finding a good model

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

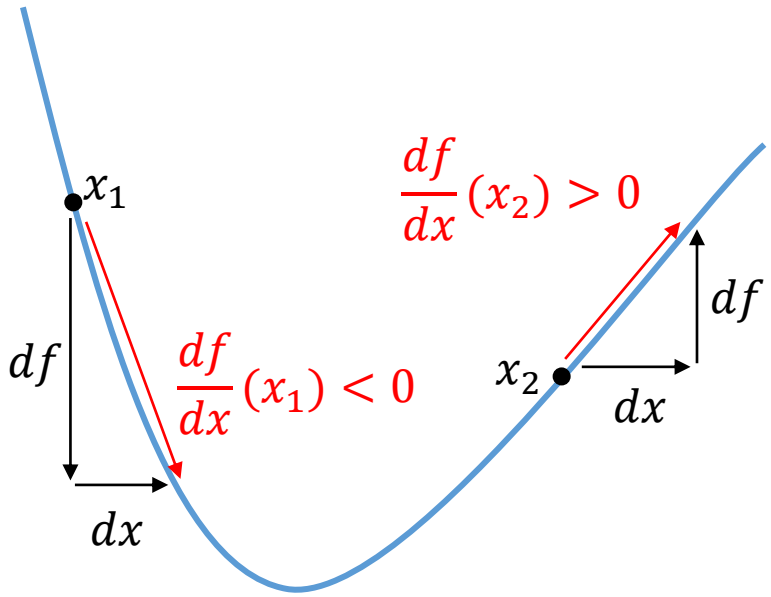
How can we find the parameters w and b automatically?

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

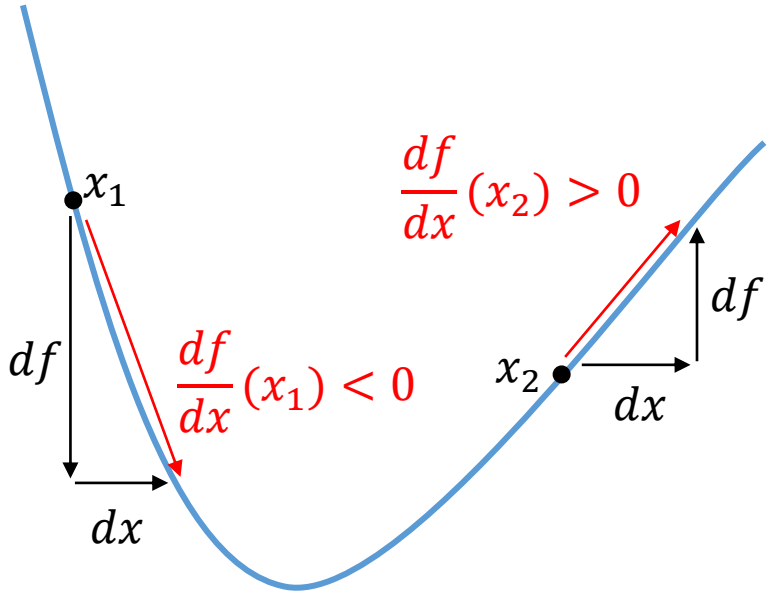
The derivative of a function f tells use how much $f(x)$ changes if we change x by a tiny amount



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



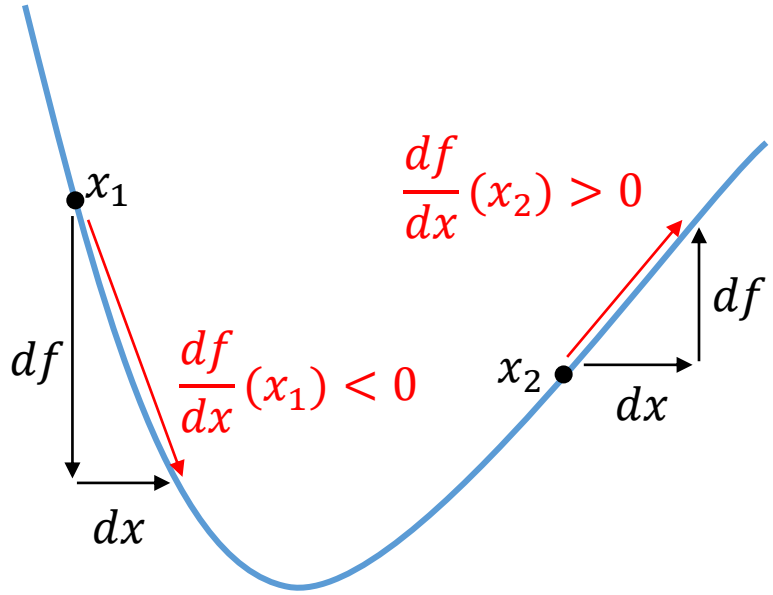
Linear Regression – Derivatives

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

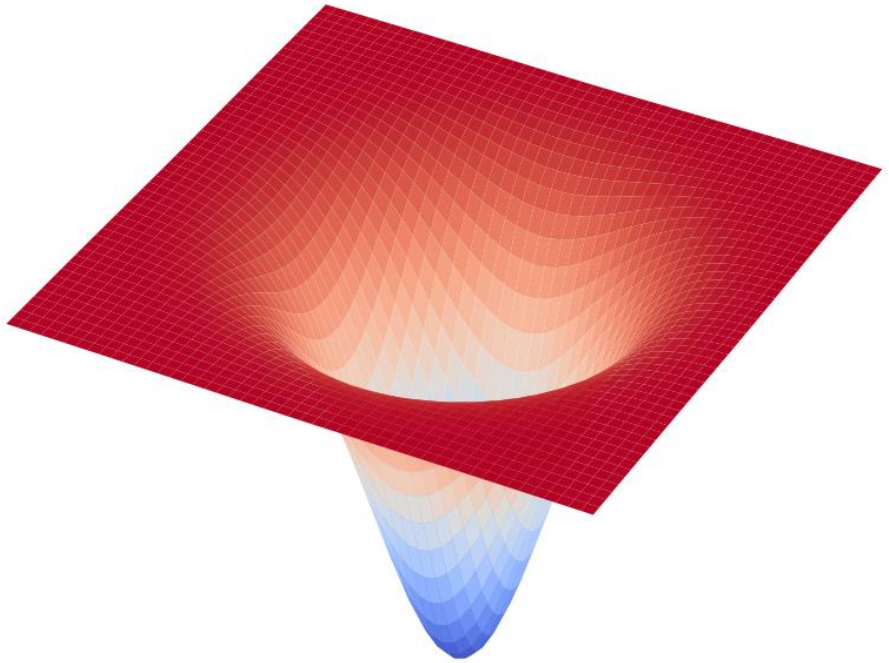
Changing x to $x + \frac{df}{dx}(x)$ will increase $f(x)$

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



Linear Regression – Finding a good model

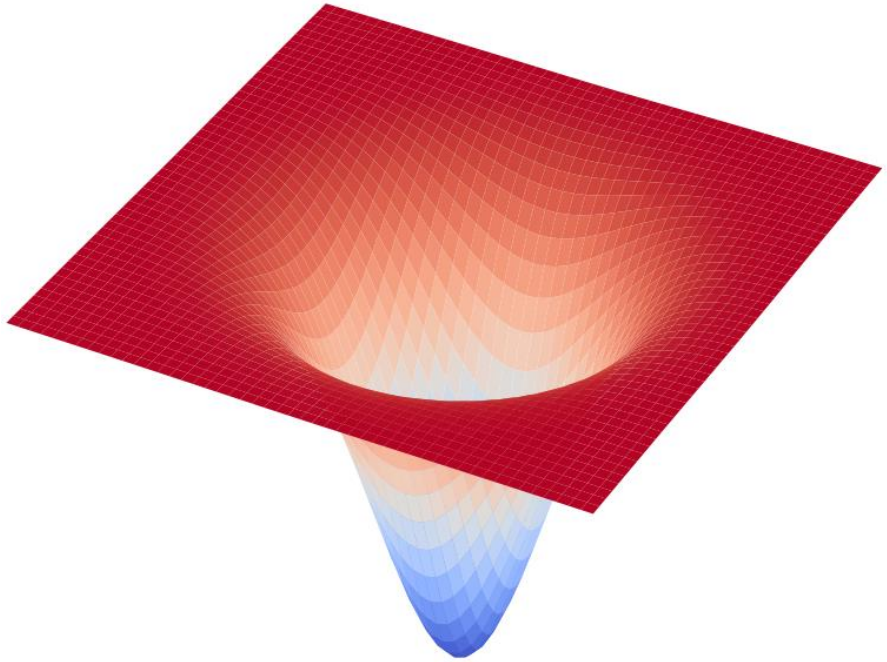
We can generalize the derivative from single to multiple function variables with partial derivatives



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



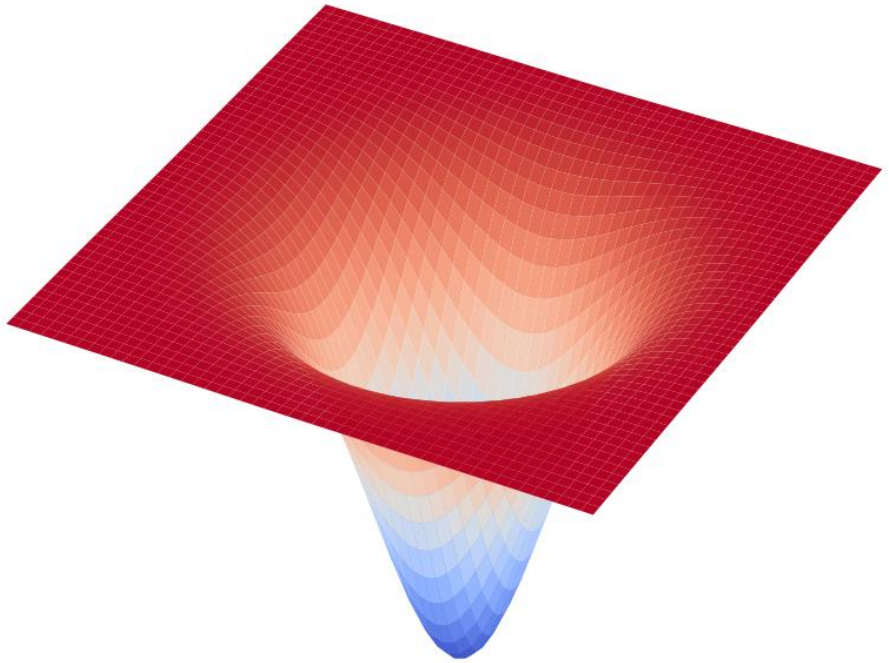
Linear Regression – Finding a good model

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

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

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



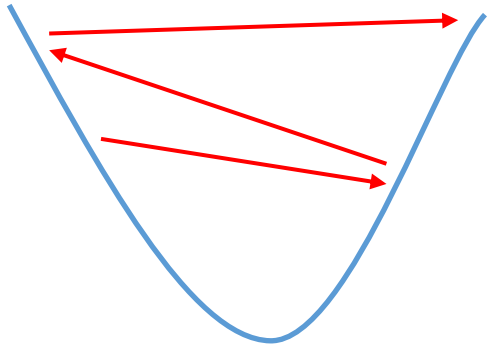
Linear Regression – Gradient Descent

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

Linear Regression – Gradient Descent

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

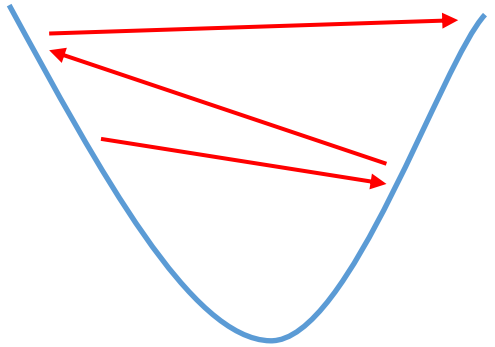
The gradient can be very large causing the algorithm to overshoot the minimum and to not converge



Linear Regression – Gradient Descent

This 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 α to reduce the gradient

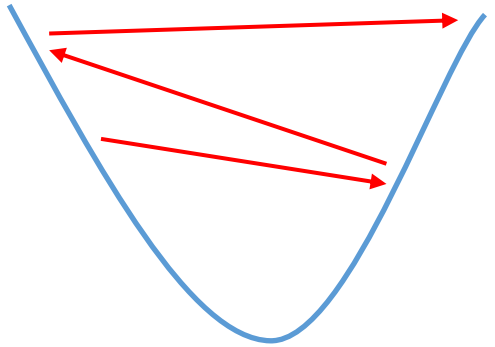
$$w := w - \alpha \frac{\partial}{\partial w} loss(w, b)$$

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

Linear Regression – Gradient Descent

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

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

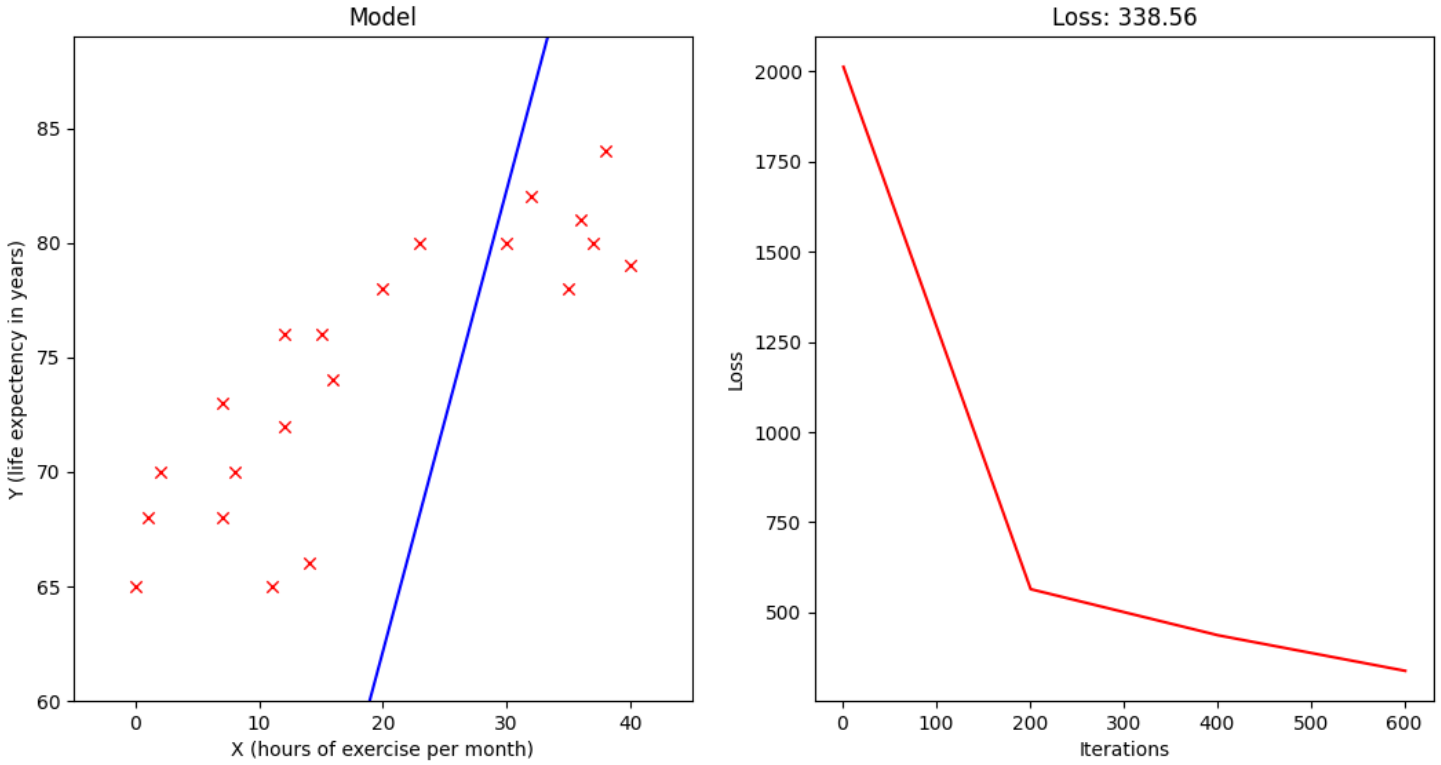
The algorithm's convergence speed will also be increased

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

Linear Regression – Gradient Descent

Exercise

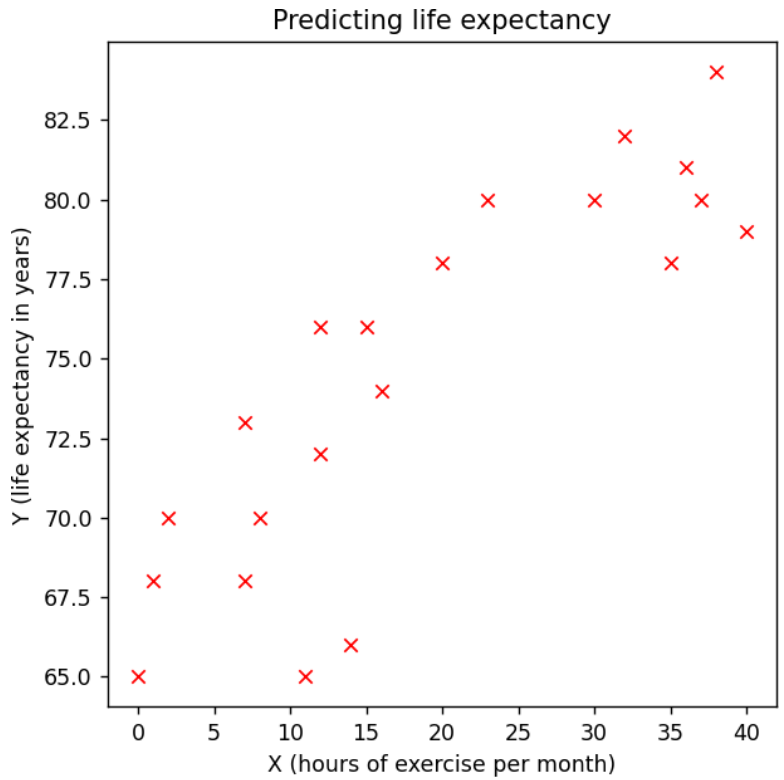
Implement gradient descent to automatically find good w and b parameters. Start with the code written in the **gradient_descent.py** script.



Lunch break

Discrete prediction

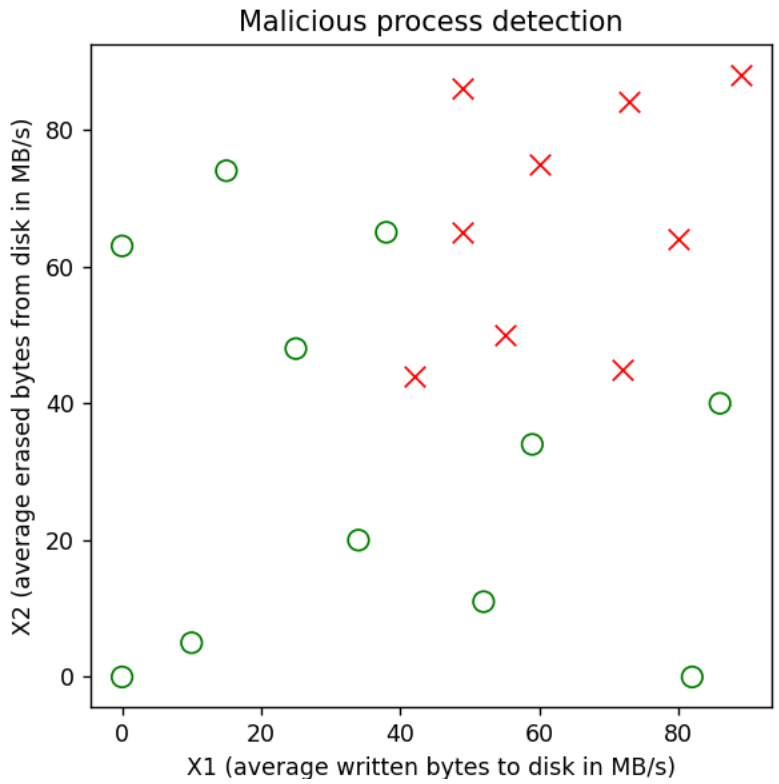
Linear Regression allows us to map features to continuous outputs



Discrete prediction

Linear Regression allows us to map features to continuous outputs

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

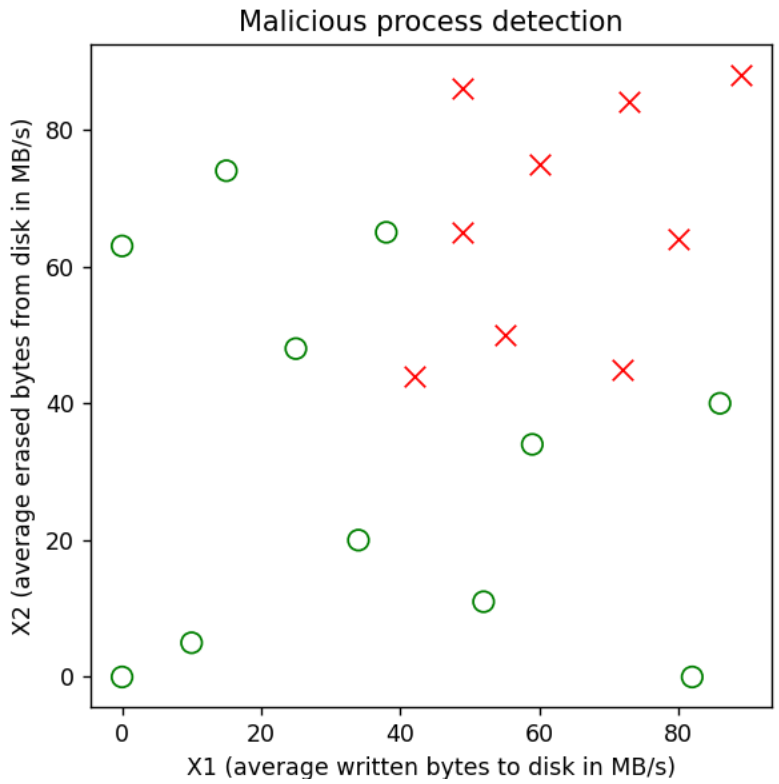


Discrete prediction

Linear Regression allows us to map features to continuous outputs

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

Logistic regression creates a boundary line to separate positives from negatives



Logistic Regression

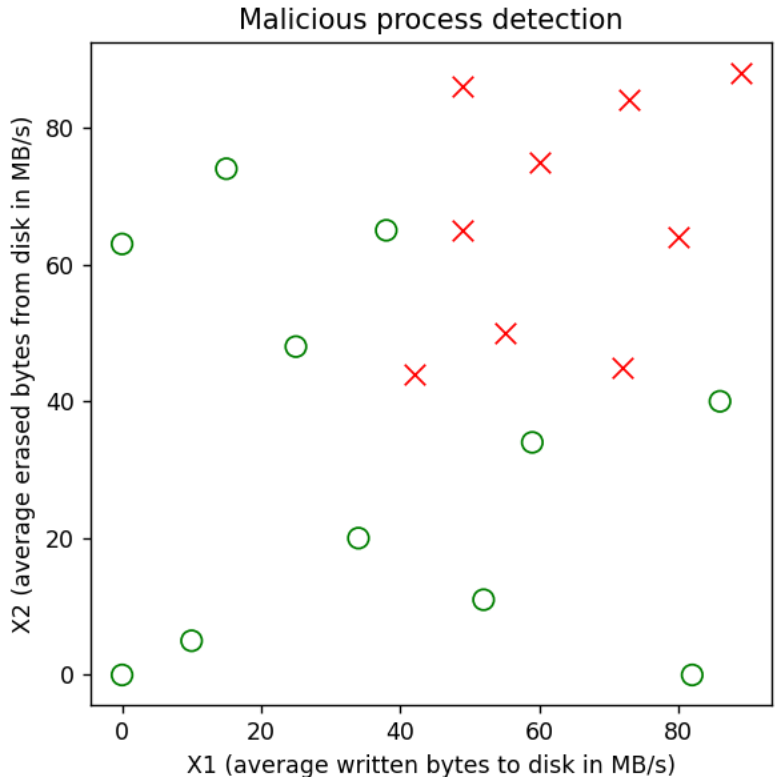
$X \rightarrow f(X) = w_1 x_1 + w_2 x_2 + b \rightarrow a(f(X)) \rightarrow \hat{y}$

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

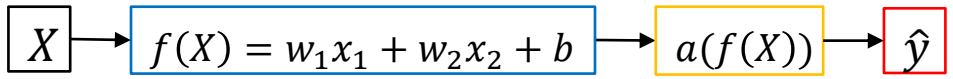
Calculate the "distance" between X and the decision boundary f

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

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



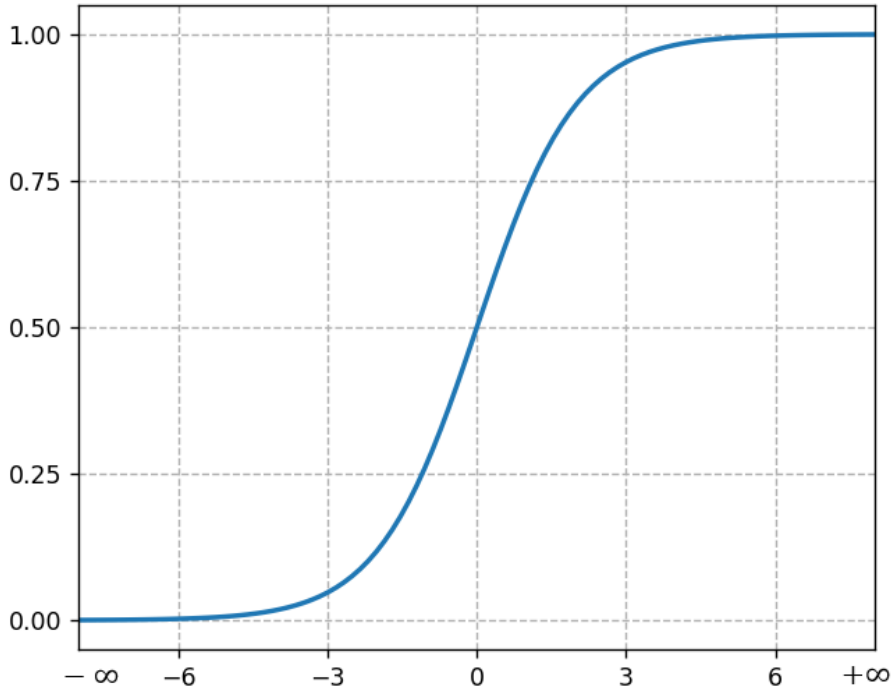
Logistic Regression



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 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 Squared Error loss function

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

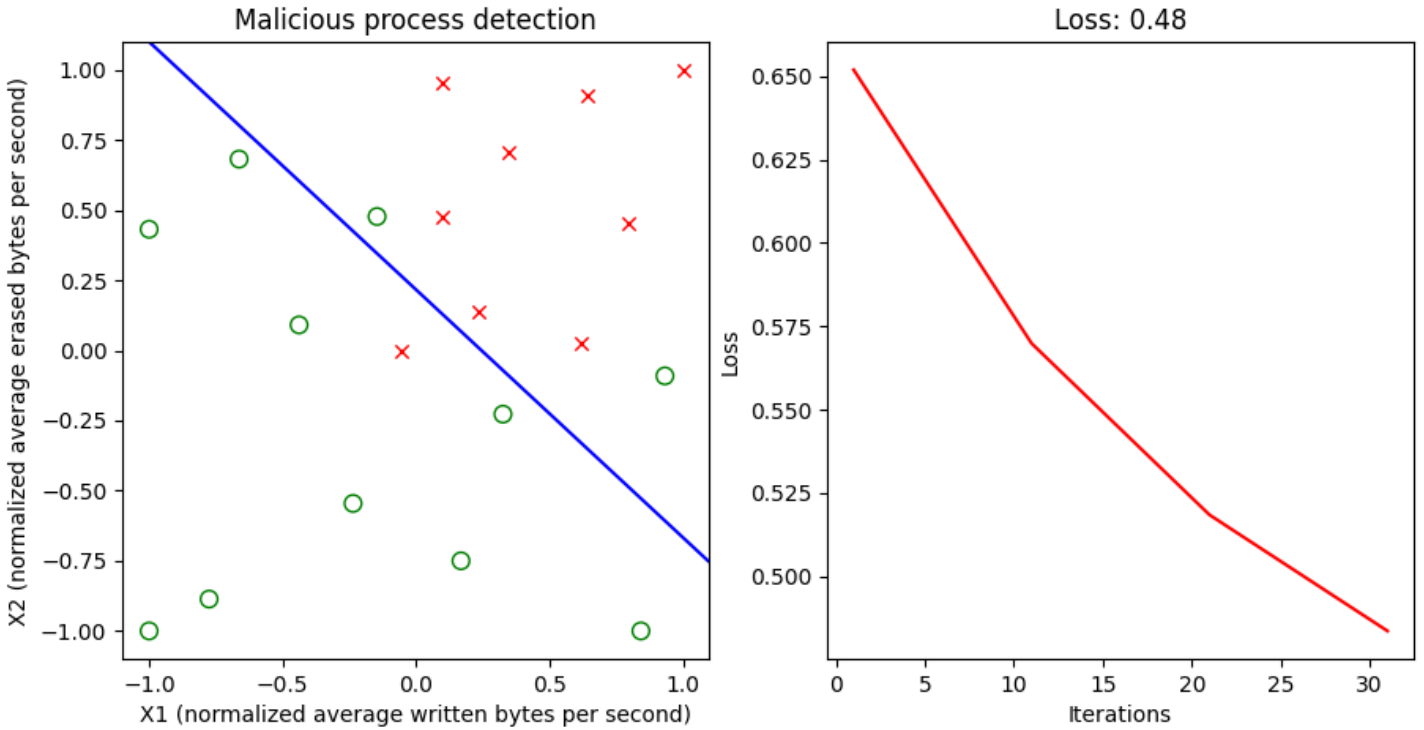
Logistic Regression Cross-Entropy (fancy name, right?) loss function

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

Logistic Regression – Gradient Descent

Exercise

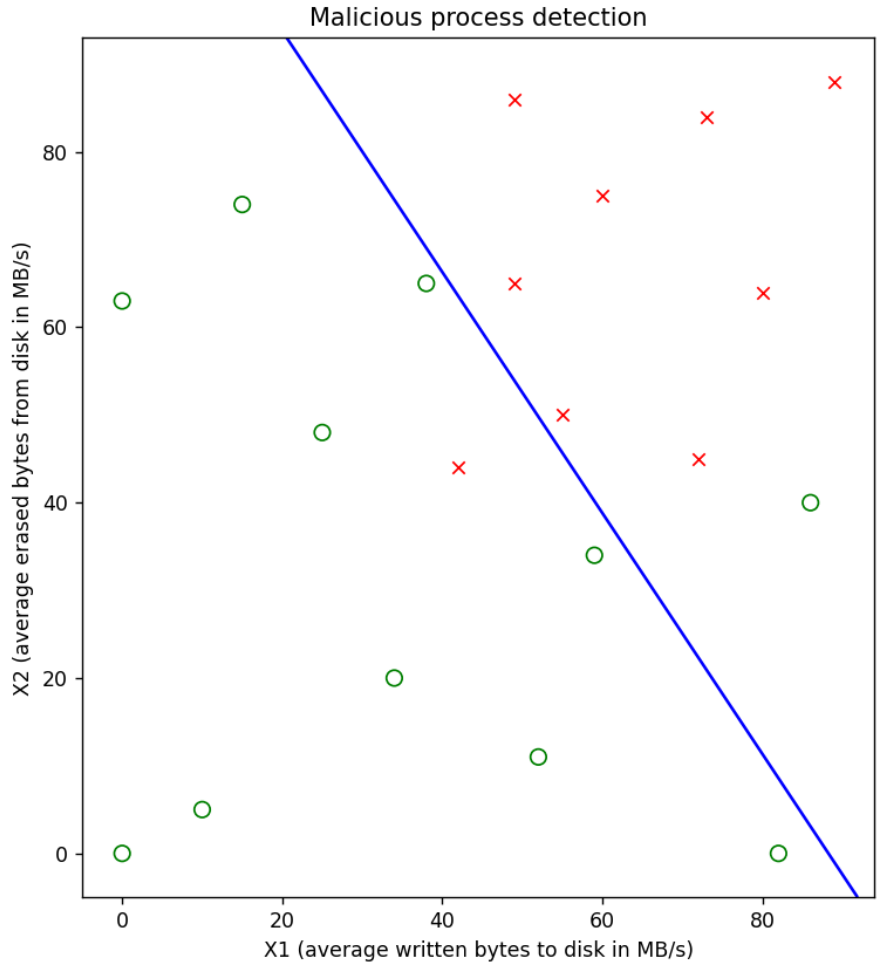
Implement gradient descent for Logistic Regression. Start with the code written in the **logistic_regression.py** script as base. Note that the data has been normalized for you.



Logistic Regression – Can we do better?

Our simple model doesn't fit the data well

We can add more features to obtain a better fit

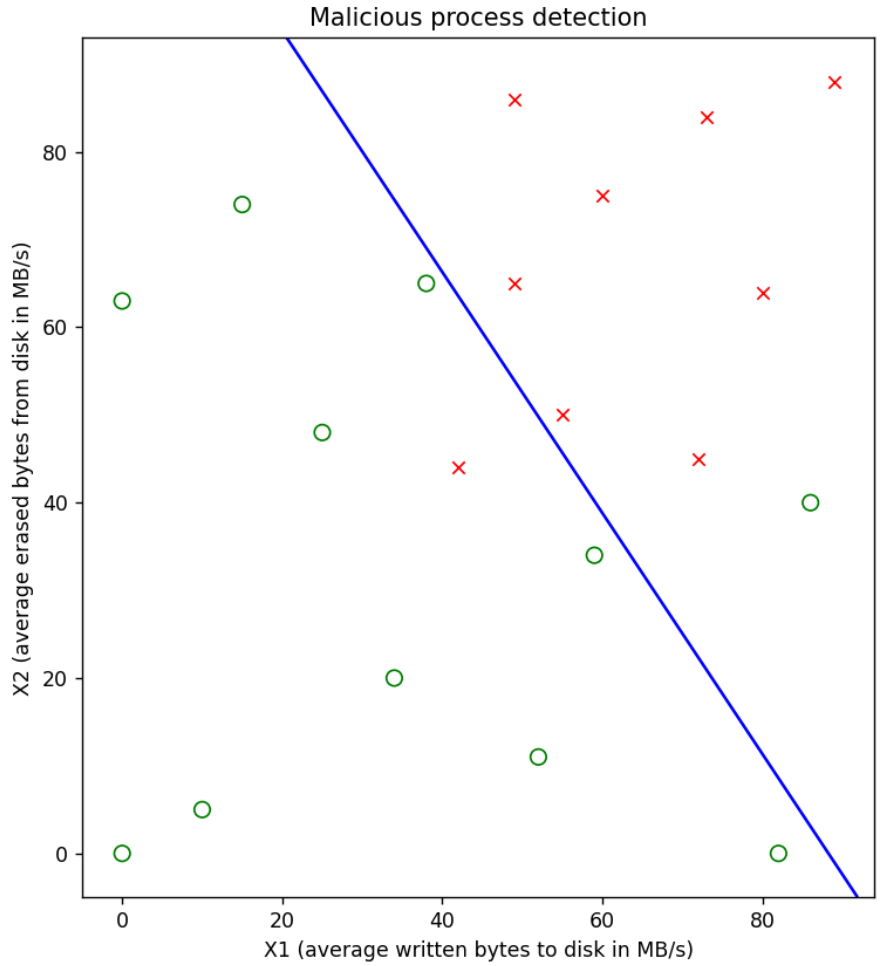


Logistic Regression – Can we do better?

Our simple model doesn't fit the data well

We can add more features to obtain a better fit

We are not adding new features such as the number of files created or erased but instead reuse the features x_1 and x_2 to keep the dimensionality to 2D to better visualize the data.



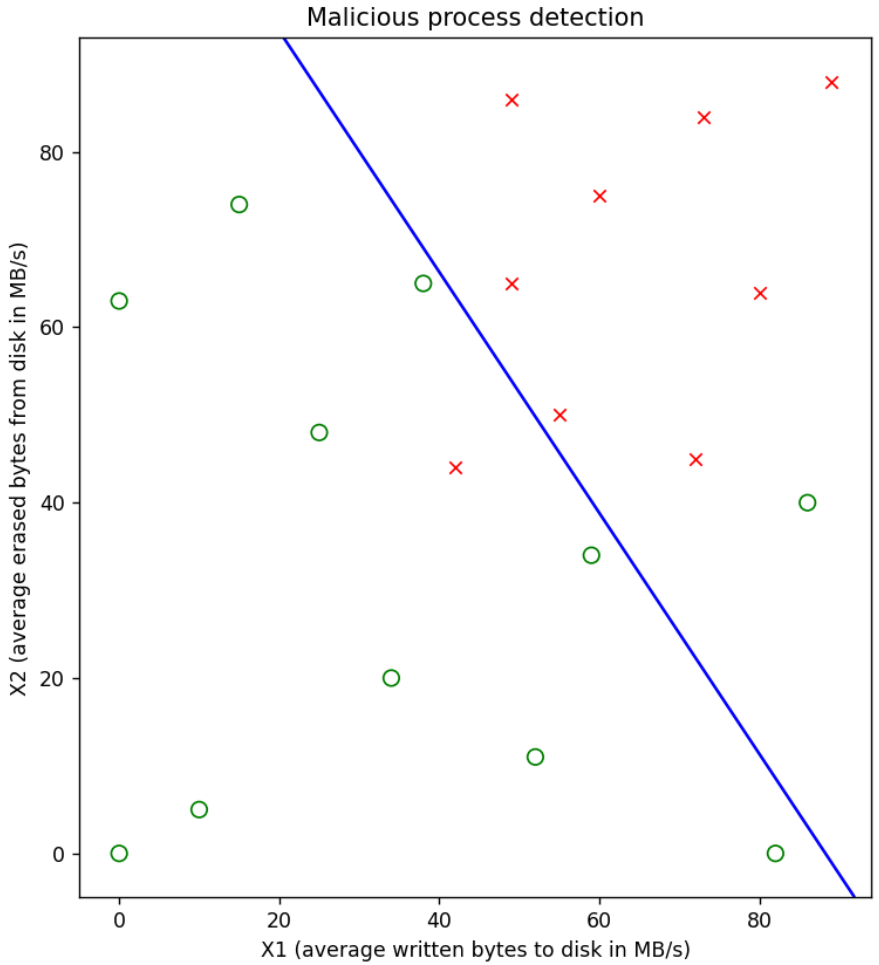
Logistic Regression – Can we do better?

Our simple model doesn't fit the data well

We can add more features to obtain a better fit

We are not adding new features such as the number of files created or erased but instead reuse the features x_1 and x_2 to keep the dimensionality to 2D to better visualize the data.

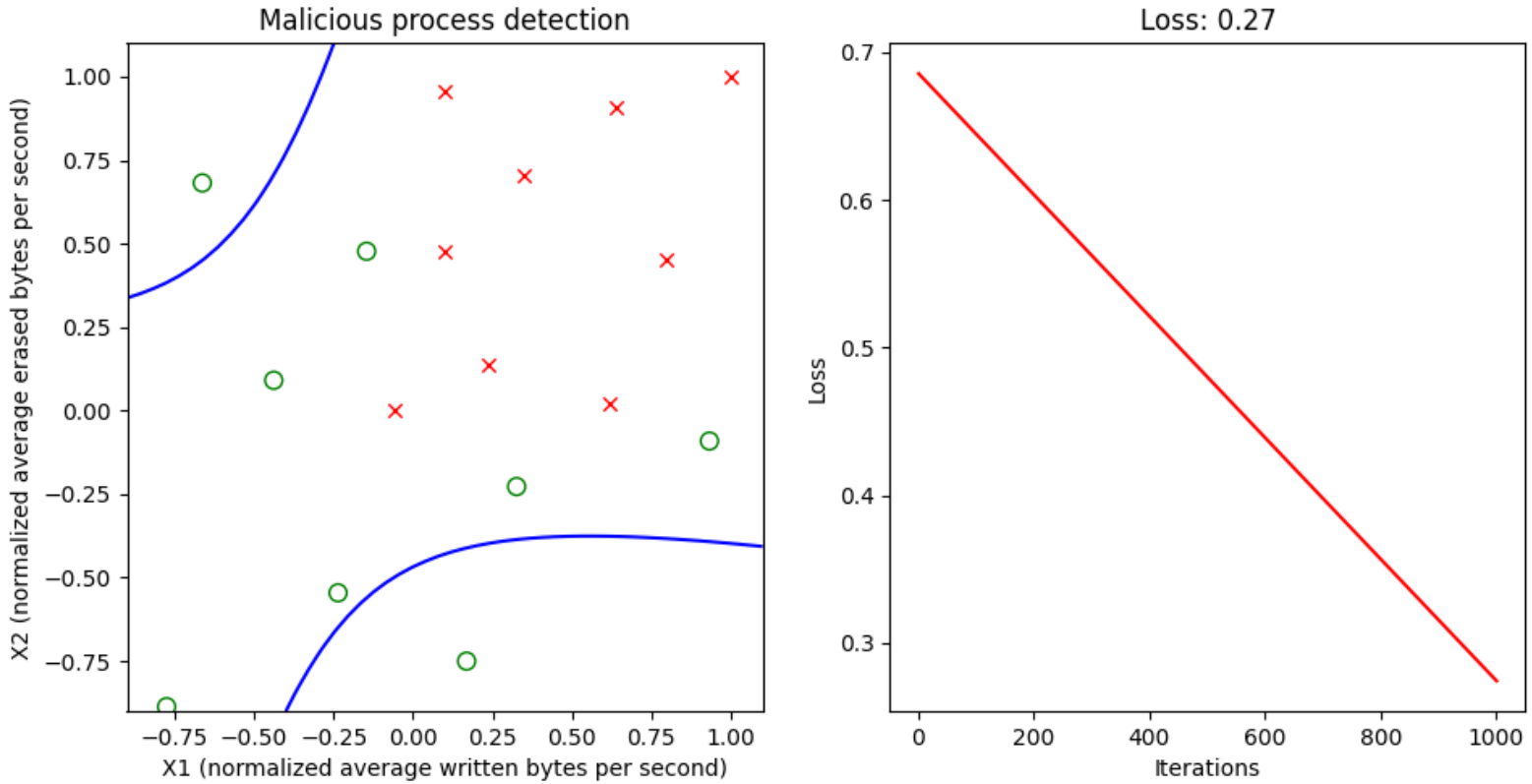
We are adding the features x_1^2, $x_1 x_2$ and x_2^2. Our boundary will be a polynomial and thus be more flexible than a simple line.



Logistic Polynomial Regression

Exercise

Implement logistic polynomial regression using the **polynomial.py** script to obtain a better fit of the data.



Polynomial Regression

Homework

Implement gradient descent with polynomial regression using the **polynomial.py** script file as the base to obtain a better fit of the life expectancy data set.

Congratulations!

You just finished day #1 of this course!

See you tomorrow for day #2!

Disadvantages of Linear and Logistic Regression

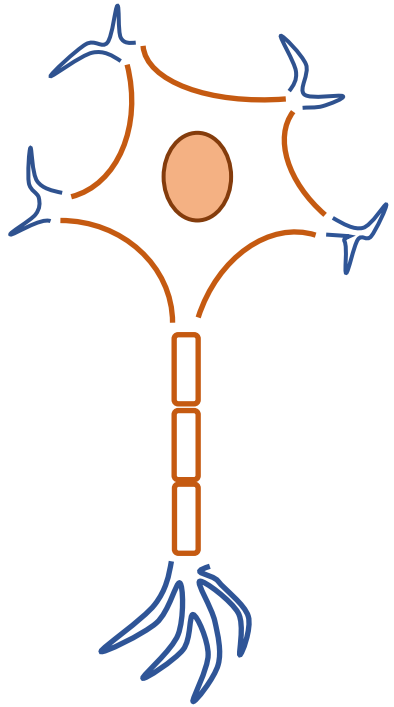
We must choose polynomial features manually for complex data sets (x_1^2, $x_1 x_2$ …)

How can we scale for more complex data sets?

This shortcoming is addressed by Artificial Neural Networks

Artificial Neural Networks

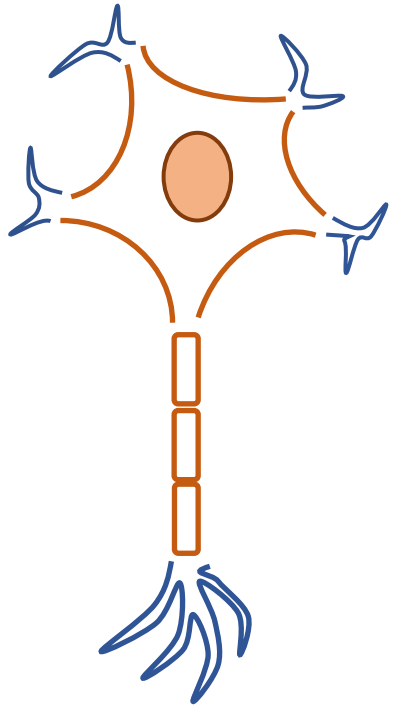
Artificial neural networks are inspired from biological neurons



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

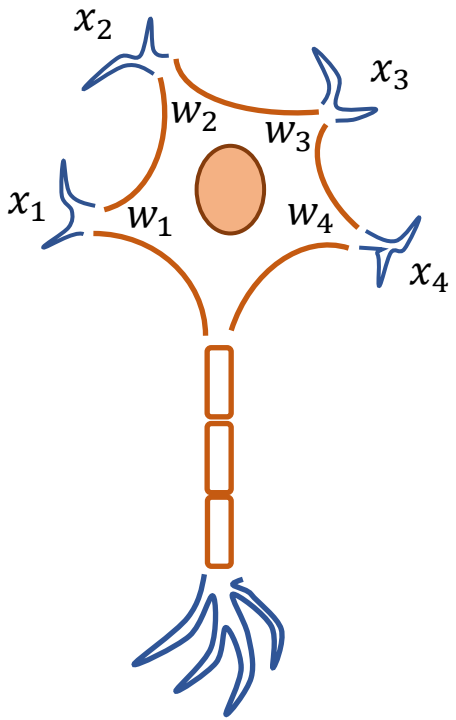


Artificial Neural Networks

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 Linear Regression model

Artificial Neural Networks – Layering

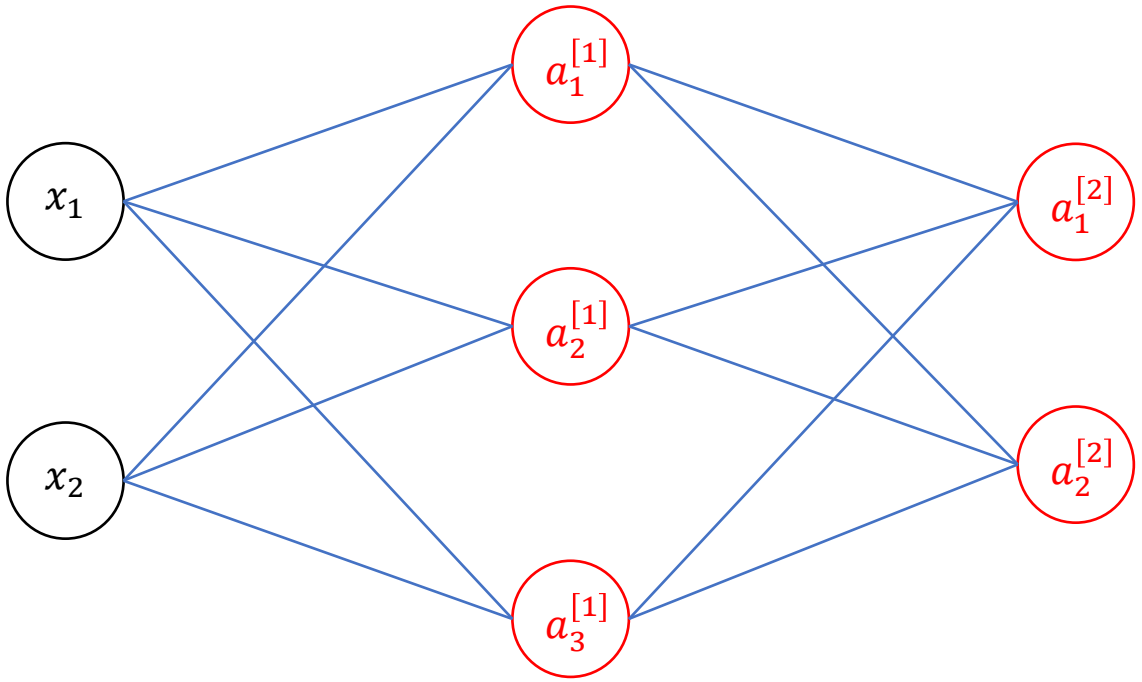
We can generalize the activation of an arbitrary layer l

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

$A^{[l]}$ 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

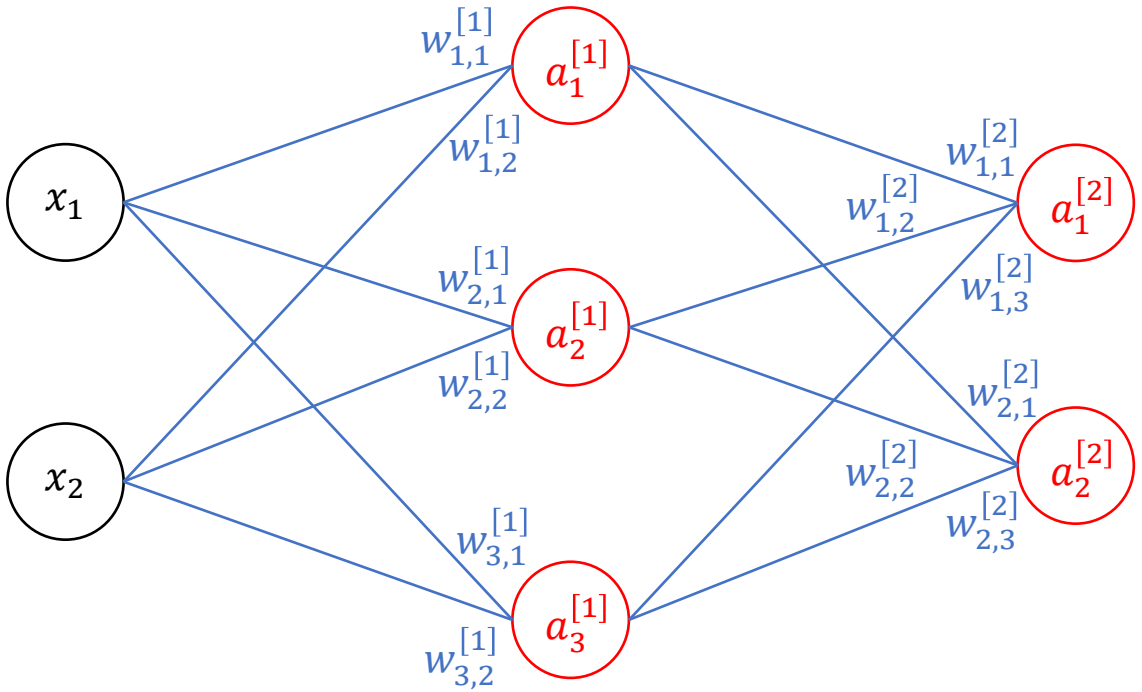
Note that $A^{[0]}$ is the feature vector X from the data set

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

$$A^{[1]} = a^{[1]}(W^{[1]}X \quad + B^{[1]})$$
$$A^{[2]} = a^{[2]}(W^{[2]}A^{[1]} + B^{[2]})$$
$$A^{[3]} = a^{[3]}(W^{[3]}A^{[2]} + B^{[3]})$$

Artificial Neural Networks – Forward Propagation

The Forward Propagation output depends on the type of the activation function for the last layer.

Artificial Neural Networks – Forward Propagation

The Forward Propagation output depends on the type of the activation function for the last layer.

If we want the neural network to output $\mathbb{R}^n$ vectors we can set the activation function to be the identity $a^{[L]}(Z)_i = z_i$. This is the generalization of Linear Regression for multiple outputs.

Artificial Neural Networks – Forward Propagation

The Forward Propagation output depends on the type of the activation function for the last layer.

If we want the neural network to output $\mathbb{R}^n$ vectors we can set the activation function to be the identity $a^{[L]}(Z)_i = z_i$. This is the generalization of Linear Regression for multiple outputs.

If we want the output 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 layer?

Artificial Neural Networks – Activation function

Why do we need activation functions for each 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 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.

Using the sigmoid activation for intermediate layers can lead to issues such as gradient saturation and slow convergence.

Artificial Neural Networks – Activation function

Why do we need activation functions for each 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.

Using the sigmoid activation for intermediate layers can lead to issues such as gradient saturation and slow convergence.

These issues are addressed by the $ReLU$ 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 Squared Error loss function for n continuous outputs…

$$loss(W_1, \dots, W_L, B_1, \dots, B_L) = \frac{1}{2m} \sum_{i=1}^{m} \sum_{j=1}^{n} \left(A_j^{[L](i)} - Y_j^{(i)} \right)^2$$

Artificial Neural Networks – Loss function

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

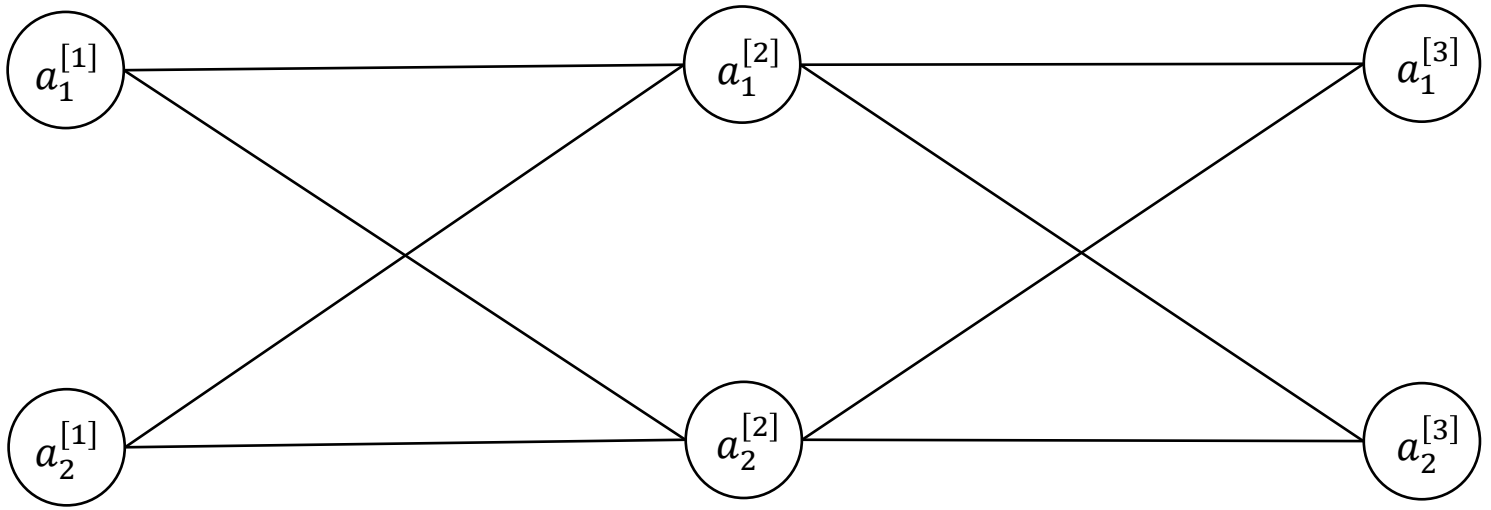
$$loss(W_1, \dots, W_L, B_1, \dots, B_L) = \frac{1}{2m} \sum_{i=1}^{m} \sum_{j=1}^{n} \left(A_j^{[L](i)} - Y_j^{(i)} \right)^2$$

And the Cross-Entropy loss function for n discrete outputs

$$loss(W_1, \dots, W_L, B_1, \dots, B_L) = -\frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{n} \left(1 - Y_j^{(i)}\right) \log\left(1 - A_j^{[L](i)}\right) + Y_j^{(i)} \log\left(A_j^{[L](i)}\right)$$

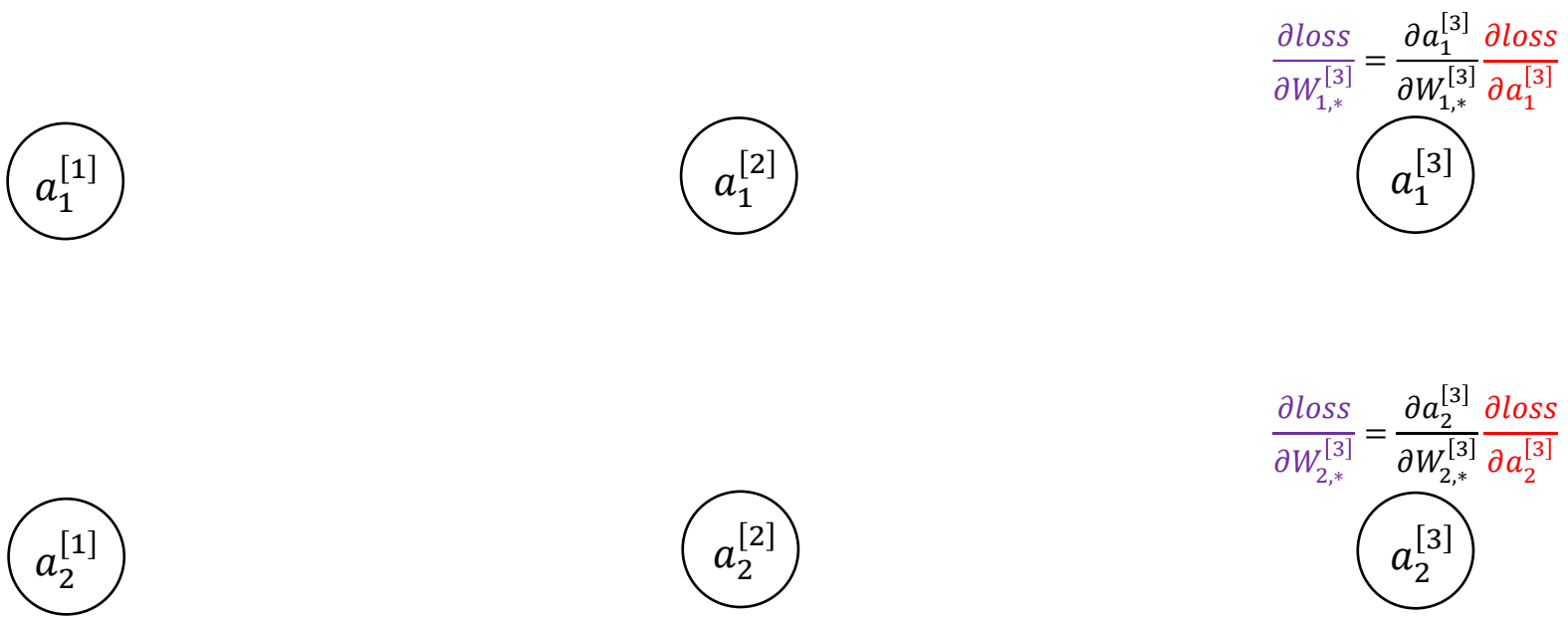
Artificial Neural Networks – Backpropagation

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



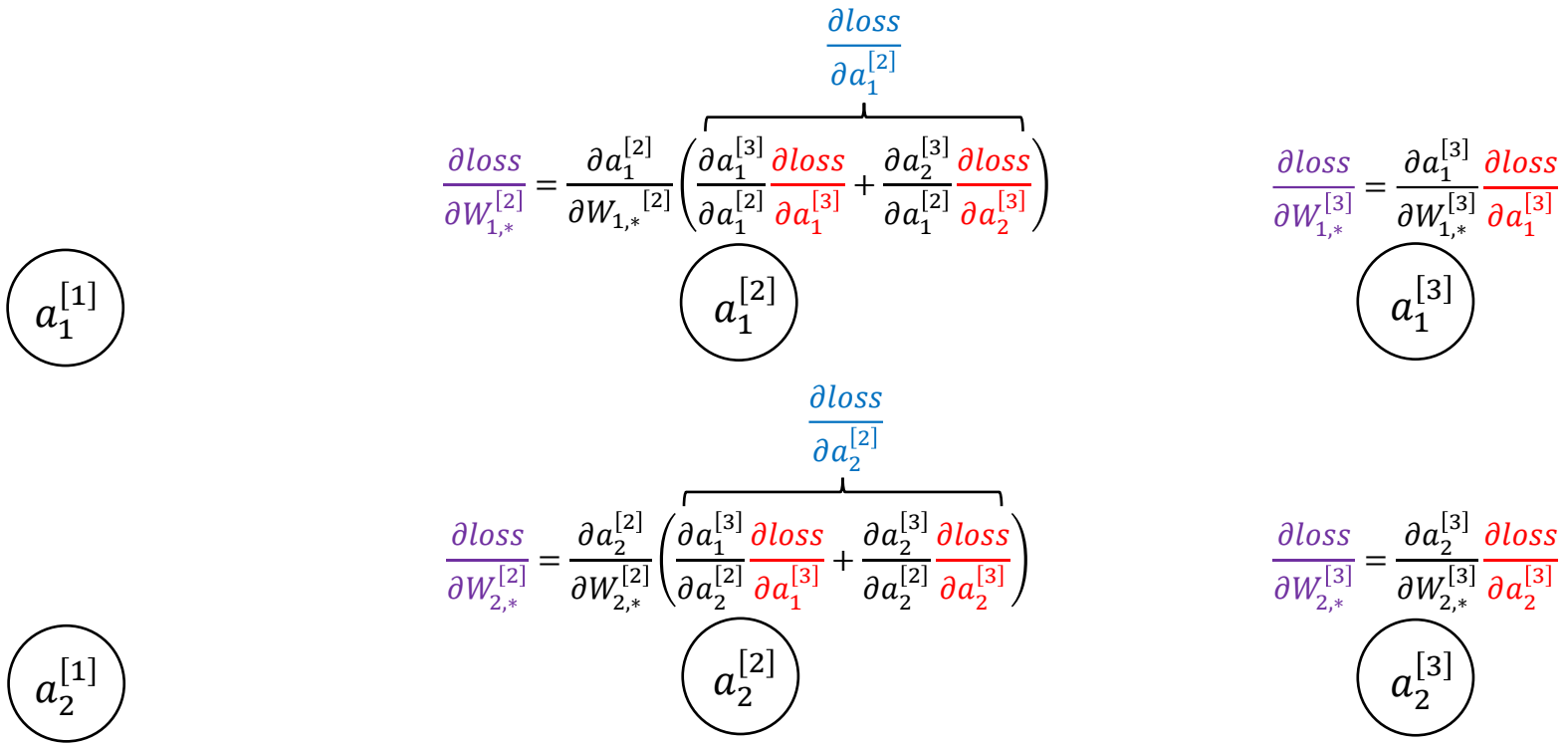
Artificial Neural Networks – Backpropagation

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



Artificial Neural Networks – Backpropagation

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



Artificial Neural Networks – Backpropagation

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

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

$a_1^{[1]}$

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

$a_1^{[2]}$

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

$a_1^{[3]}$

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

$a_2^{[1]}$

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

$a_2^{[2]}$

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

$a_2^{[3]}$

Artificial Neural Networks – Backpropagation

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

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

$a_1^{[1]}$

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

$a_1^{[2]}$

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

$a_1^{[3]}$

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

$a_2^{[1]}$

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

$a_2^{[2]}$

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

$a_2^{[3]}$

Lunch break

Final exercise

Implement and train an artificial neural network to recognize digits in images.

You can choose the difficulty of this exercise:

- You found this course difficult – make use of existing code inside the **neural.py** script to achieve your objective. You must implement only the **relu_activation**, **forward_propagation**, **calculate_loss** and **back_propagation** functions. Everything else is provided for you including tests.

- You found this course easy… and you can write code while you sleep – implement everything from scratch inside the **tough_neural.py** script to achieve your objective. Only the data set reading and draw functions are provided to you so you can focus on the machine learning aspect. You'll have to write your own tests.

Test your artificial neural network by classifying digits drawn by you or by others.

Congratulations!

You have finished the course!