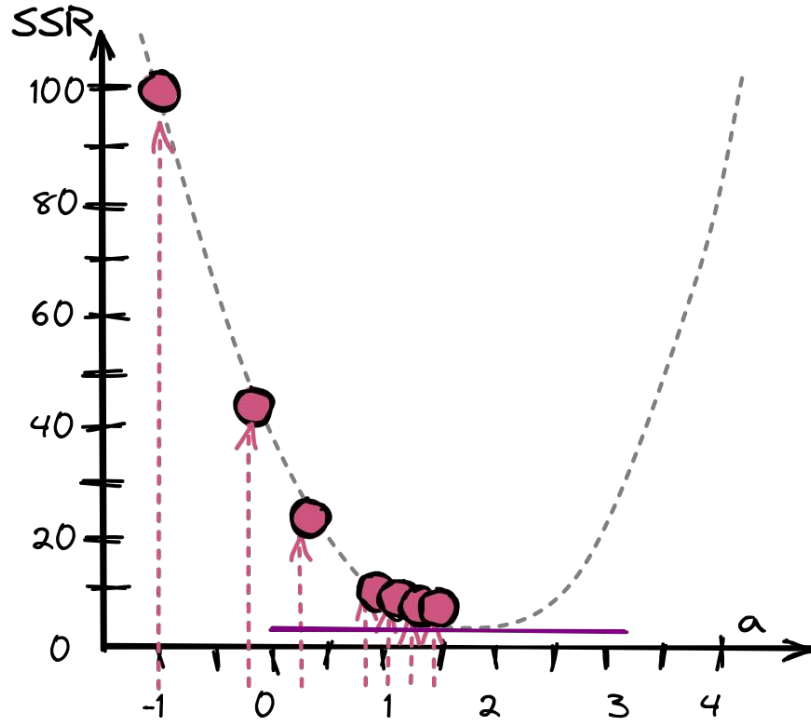


Neural Networks

Part 2

Week 16



Middlesex University Dubai; CST4050 Fall21;
Instructor: Ivan Reznikov

Plan

- Chain Rule
- Gradient Descent
- Backpropagation

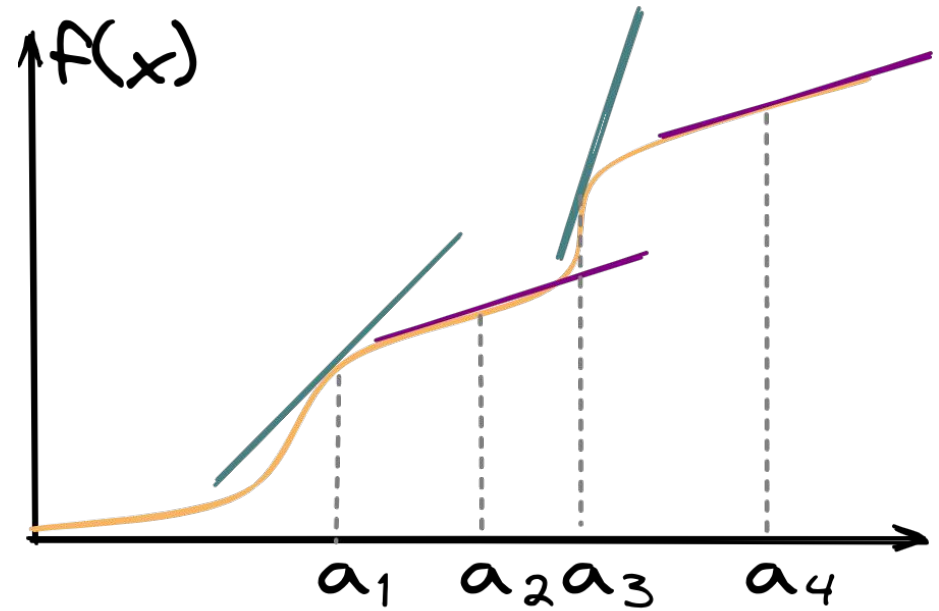
Reminder: Derivatives

Derivative is a measure of the rate at which the value $f(x)$ changes with respect to the change of the variable a .

The derivative of the function $f(x)$, evaluated at $x=a$ gives the slope of the curve at $x=a$.

Speed is the derivative of distance with respect to time:

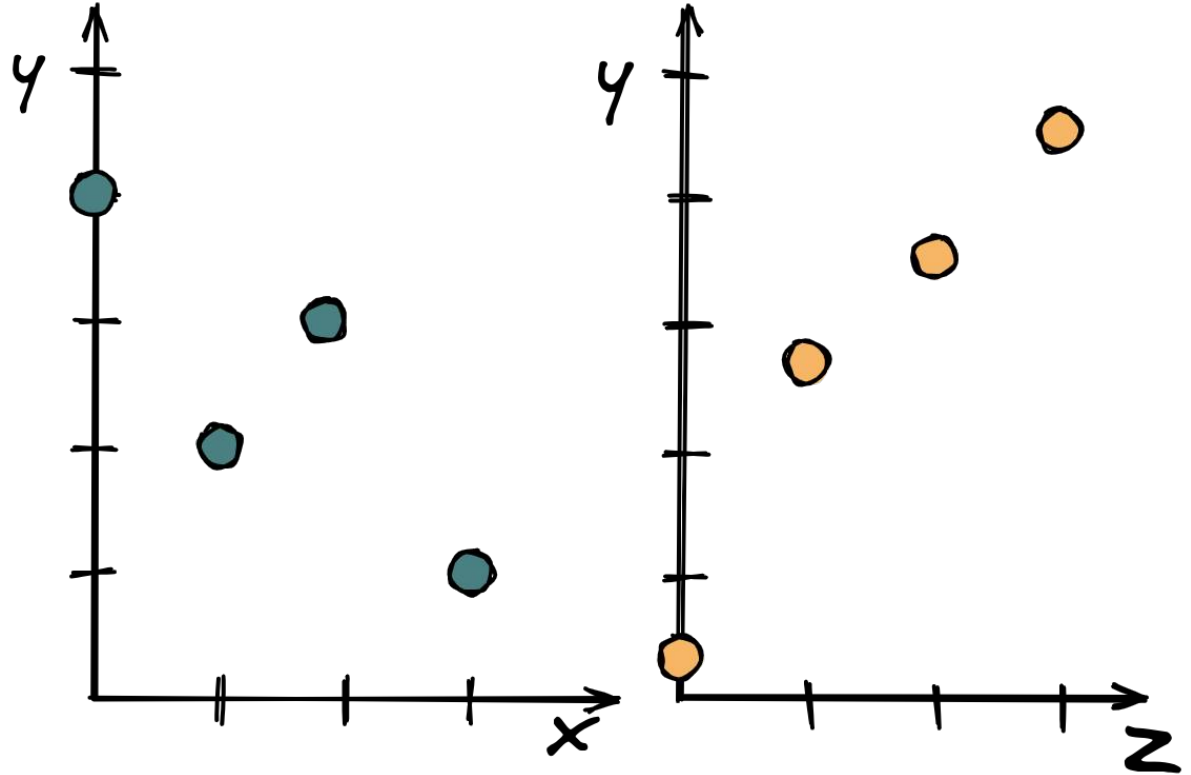
$$\text{speed} = \frac{d(\text{distance})}{d(\text{time})}$$



Data: Case1

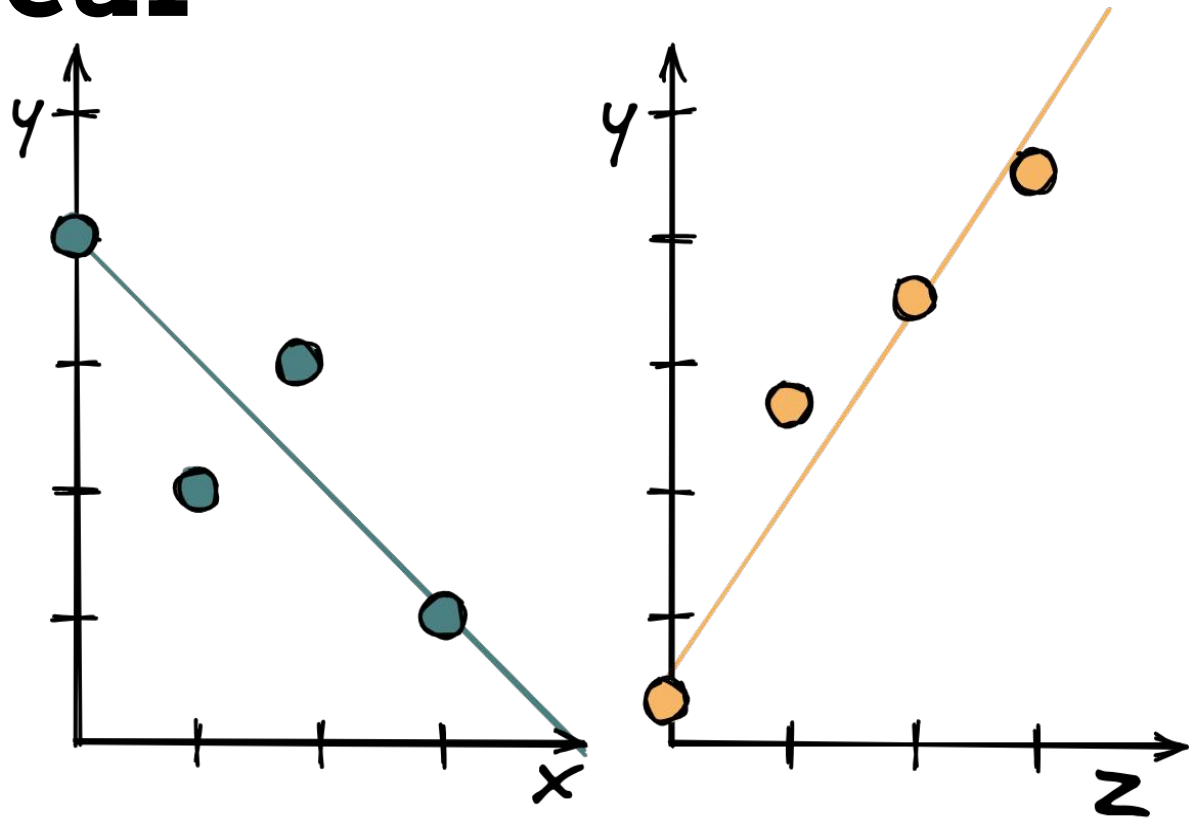
Let's imagine we have some variable x that affects variable y . Our variable y is positively correlated with variable z .

Our target is to figure out how the changes in x will affect z value.



Linear + Linear

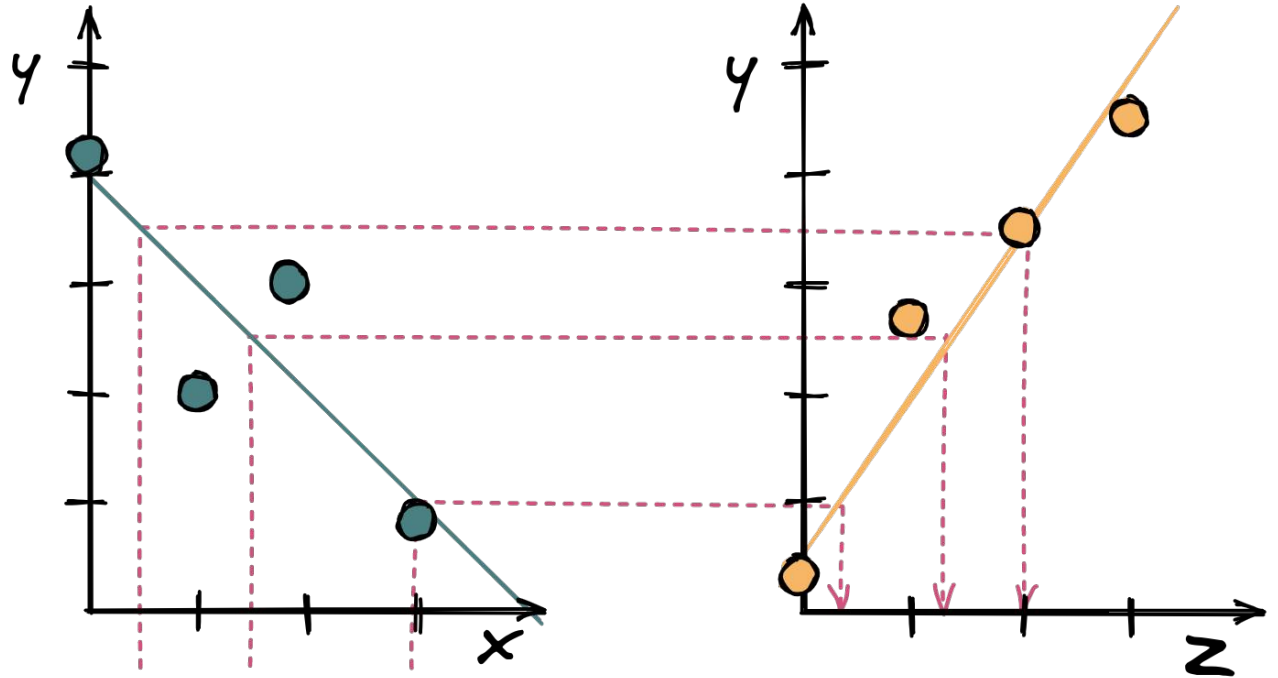
We'll start with linear dependencies between x - y and y - z pairs.



x-y-z linear dependencies

If we assume that z value can be affected by x value through y , we can define the relationship between x and z .

Moreover, we can calculate the derivative of z with respect to x . This will allow us to predict how z will change if x has changed.

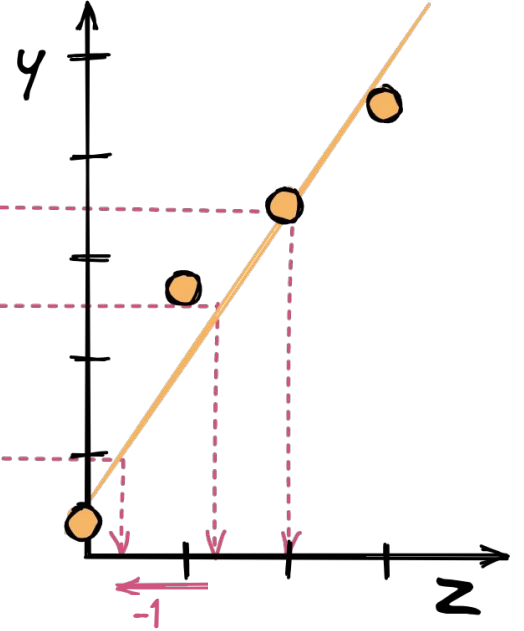
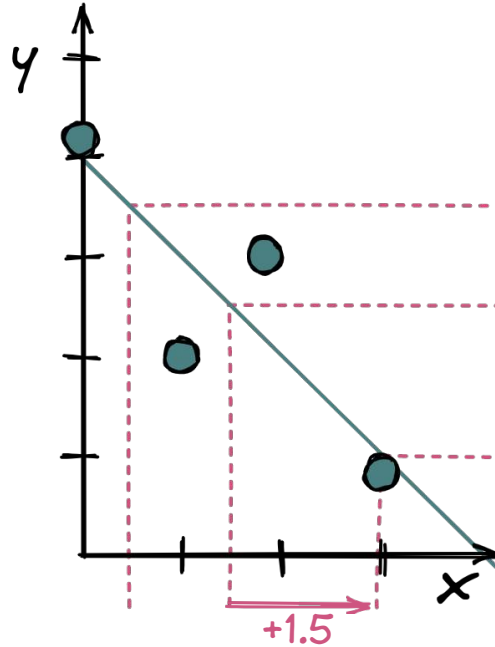


Chain Rule

$$\frac{dx}{dy} = -1 \quad \frac{dy}{dx} = -1$$

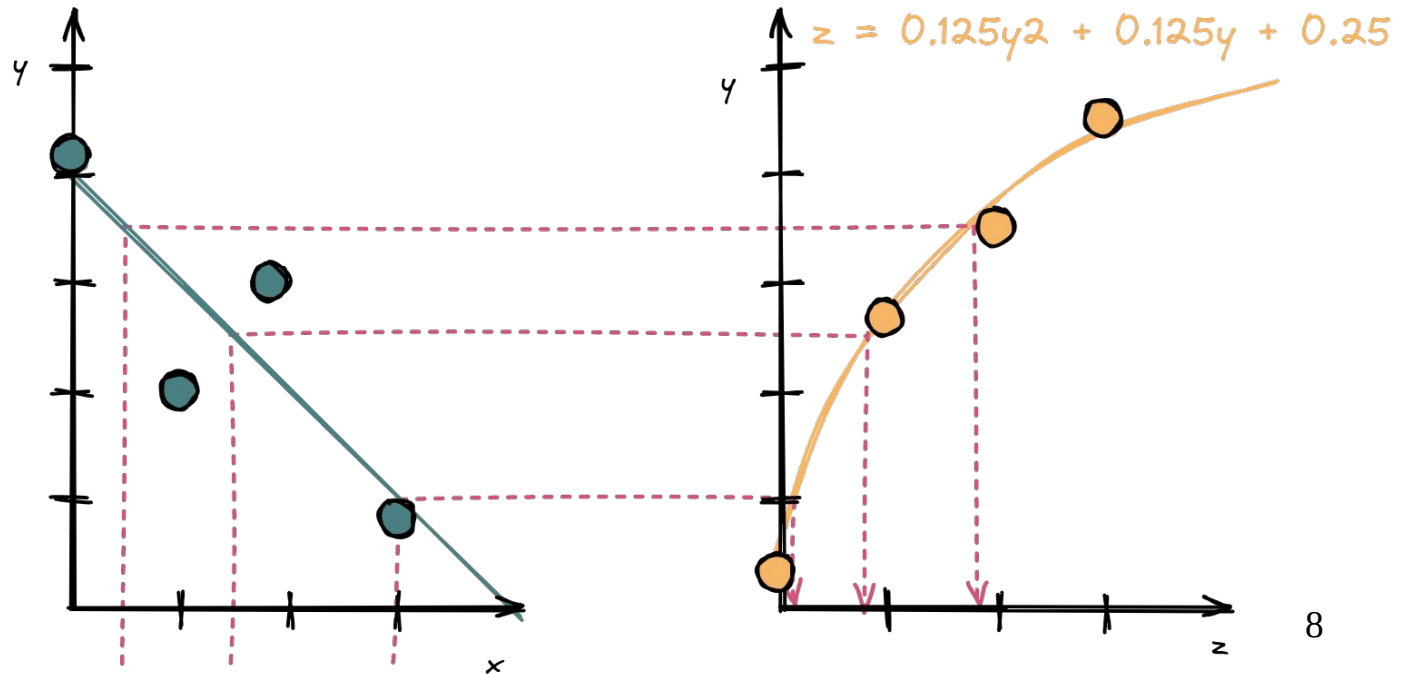
$$\frac{dz}{dy} = \frac{2}{3} \quad \frac{dy}{dz} = \frac{3}{2}$$

$$\frac{dz}{dx} = \frac{dy}{dx} \times \frac{dz}{dy} = -1 \times \frac{2}{3} = -\frac{2}{3}$$



x-y-z non-linear dependencies

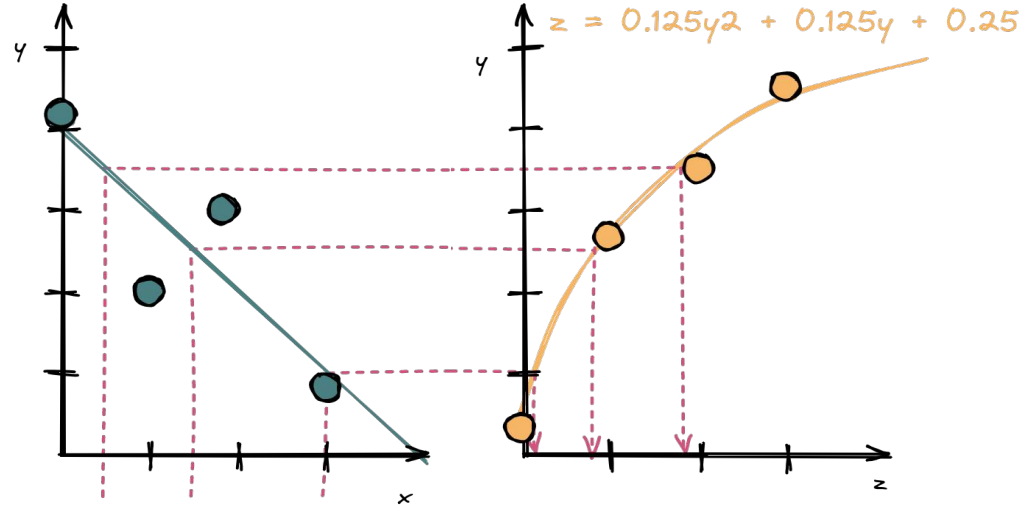
Now let's take a look at a case, where we'll have a non-linear dependency between y-z pair.



Chain Rule

$$f(x) = x^k \rightarrow f'(x) = kx^{k-1}$$

$$x^1 = x \quad x^0 = 1$$



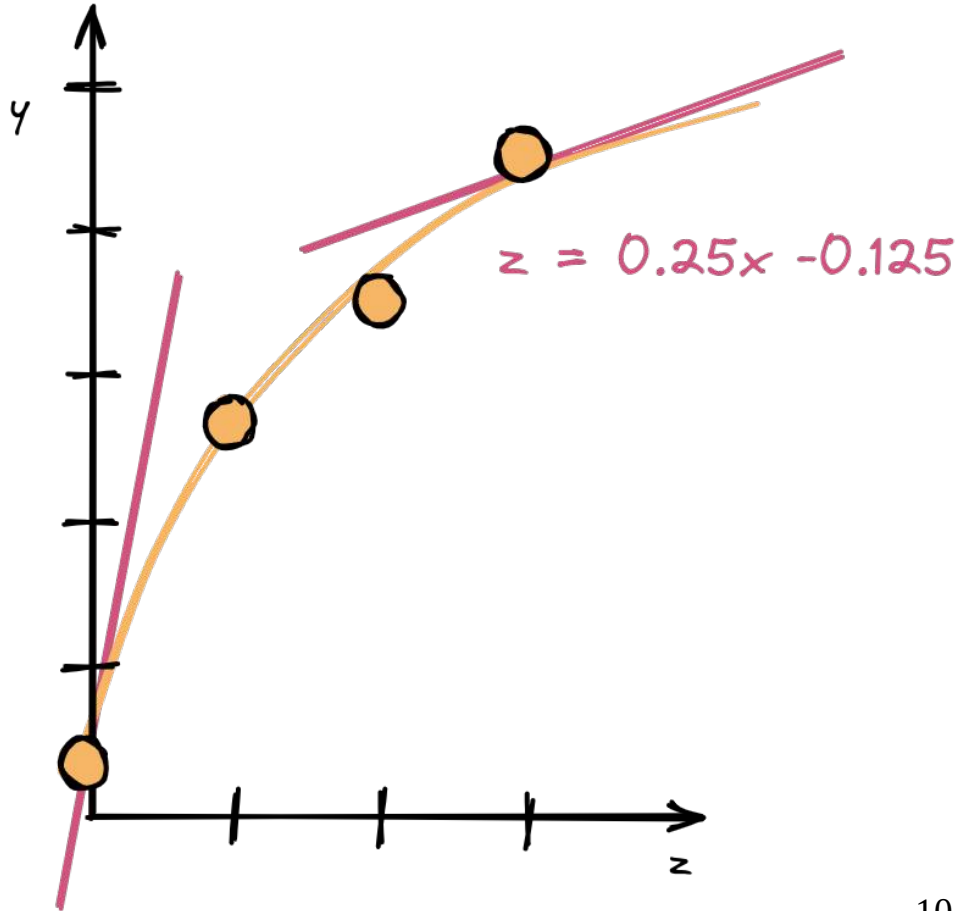
$$\begin{aligned} \frac{dz}{dy} &= f'(0.125y^2 + 0.125y + 0.25(y^0)) = \\ &= 2 \times 0.125y^{2-1} + 1 \times 0.125^{1-1} + 0 \times 0.25y^{-1} \\ &= 2 \times 0.125y + 0.125 + 0 = 0.25y + 0.125 \end{aligned}$$

$$\begin{aligned} \frac{dz}{dx} &= \frac{dy}{dx} \times \frac{dz}{dy} = -1 \times (0.25y + 0.125) = -0.25y - 0.125 \\ &= -1 \times (0.25(-x)) - 0.125 = \underline{0.25x - 0.125} \end{aligned}$$

x-z dependency

As mentioned previously, the equation of the derivative describes the slope of the function per any point.

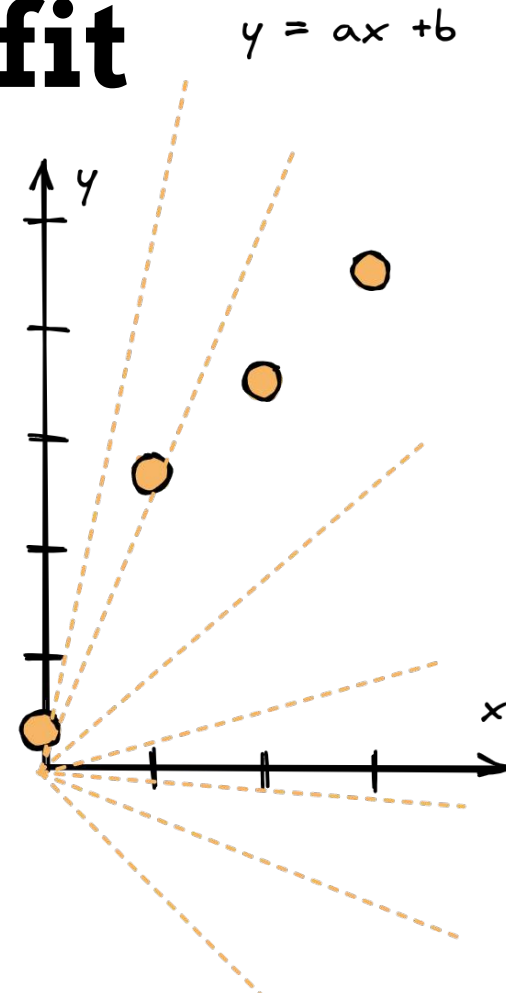
This is true, even if the point (x) is not present on the graph.



Case2: Finding best fit

For simplicity, we'll stick with the same data we've been using in the previous example.

Our current task is to find the best linear fit, given the intercept (b) = 0. Basically, we're looking for the optimal a coefficient in the equation $y = ax$

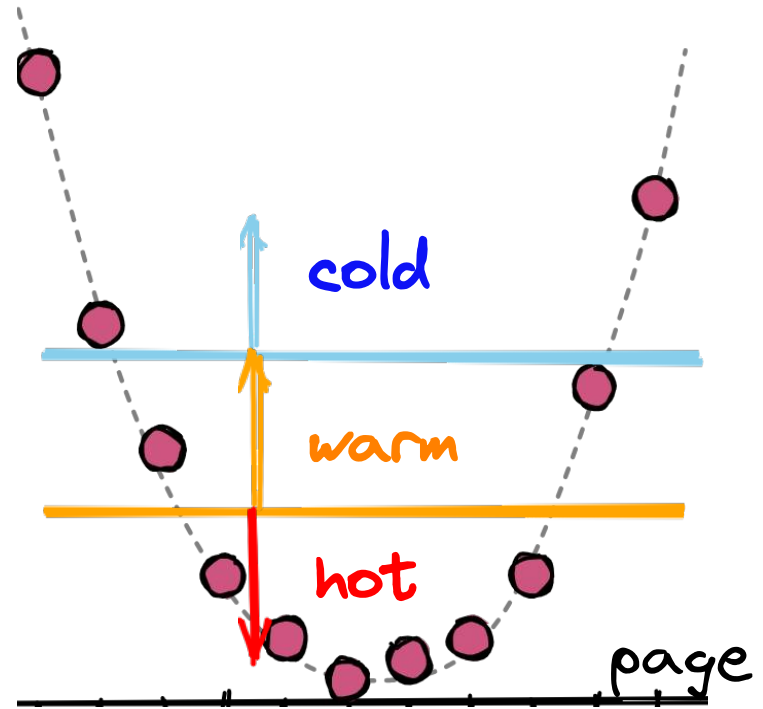


Finding best fit: Strategy1

Our first strategy might be calculating some a values, hoping to obtain a curve similar to the one in the picture.

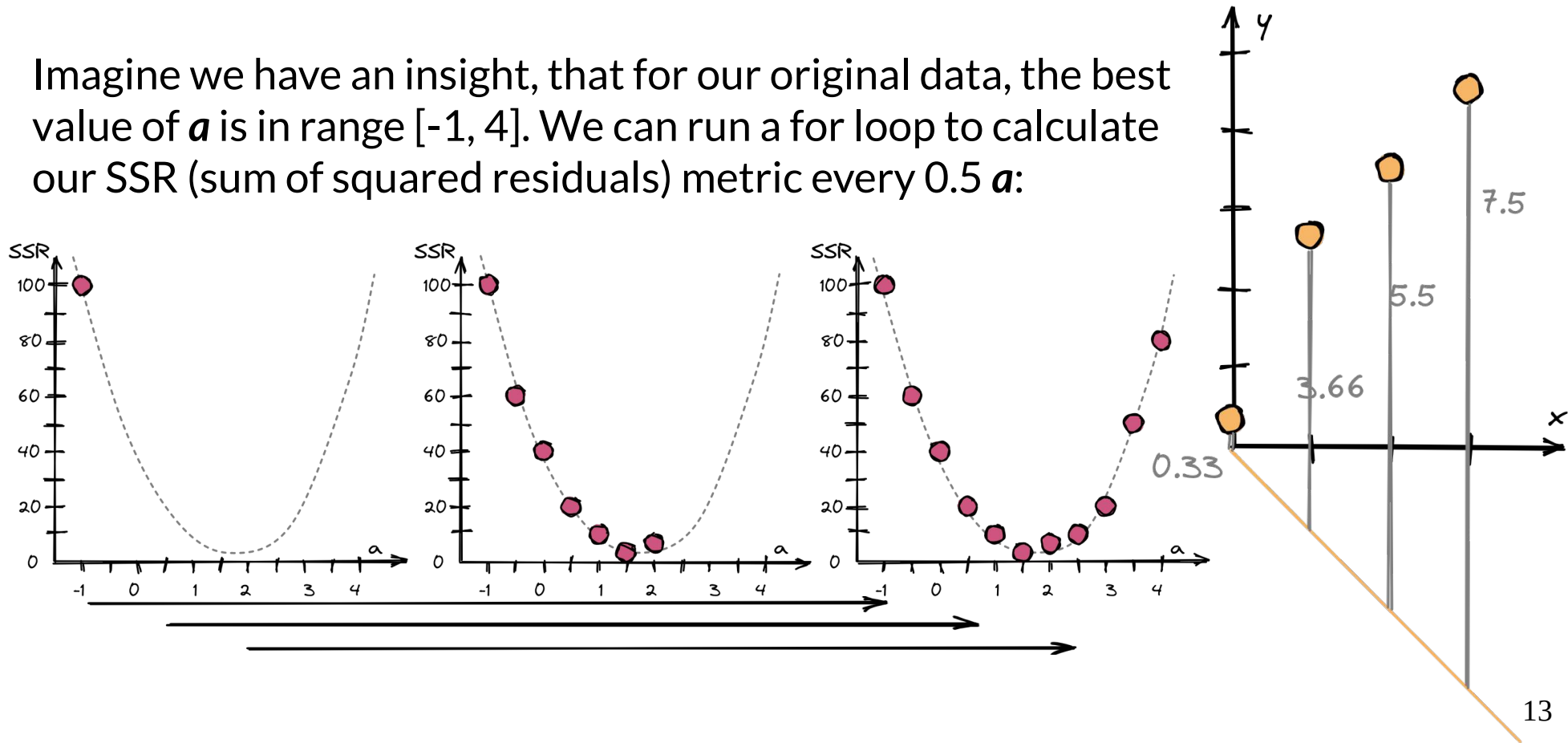
Imagine you're playing a hot and cold game, where you need to guess the book's page. Also, you don't know what book it is.

We can run a for loop every 10-20 pages, each time measuring the hotness of our guess. After the for loop is finished, we might consider running another for loop within a much smaller range. Now, we may even guess the correct page.



Finding best fit: Strategy1

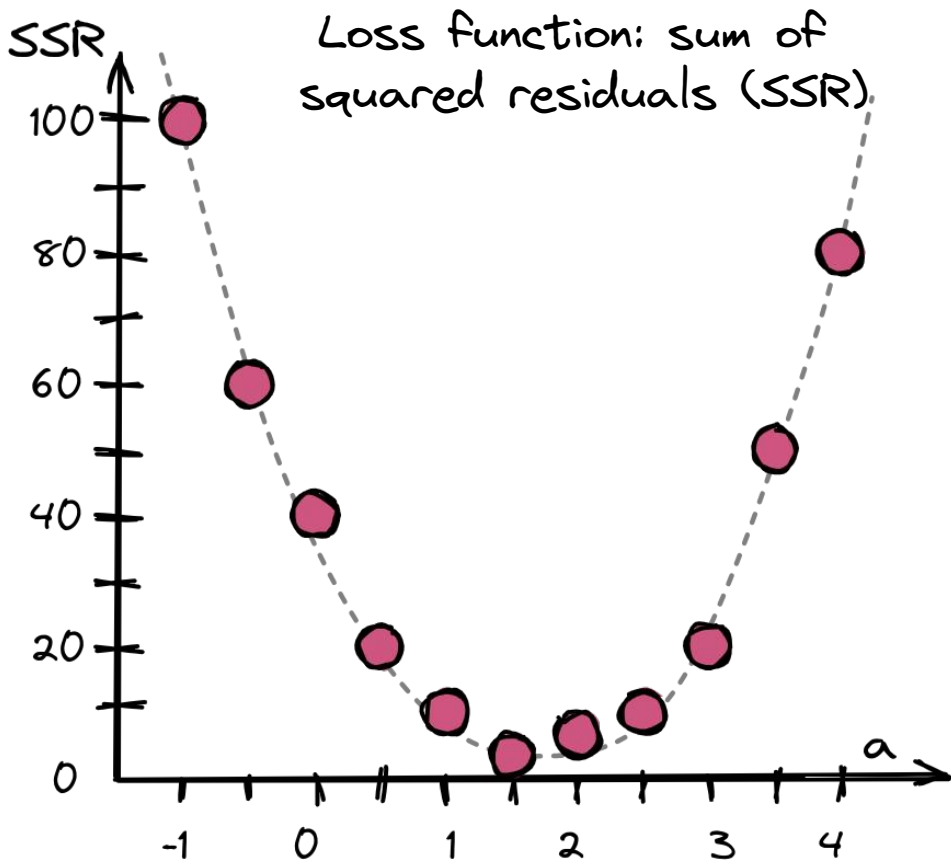
Imagine we have an insight, that for our original data, the best value of a is in range $[-1, 4]$. We can run a for loop to calculate our SSR (sum of squared residuals) metric every $0.5 a$:



Finding best fit: Strategy1

Our solution is not very effective:

- we've measured extra a values
- we haven't found the absolute minimum
- the given parabola is the best-case scenario. It happened because we had an insight about a range value. Without it, we may have spent much more time.

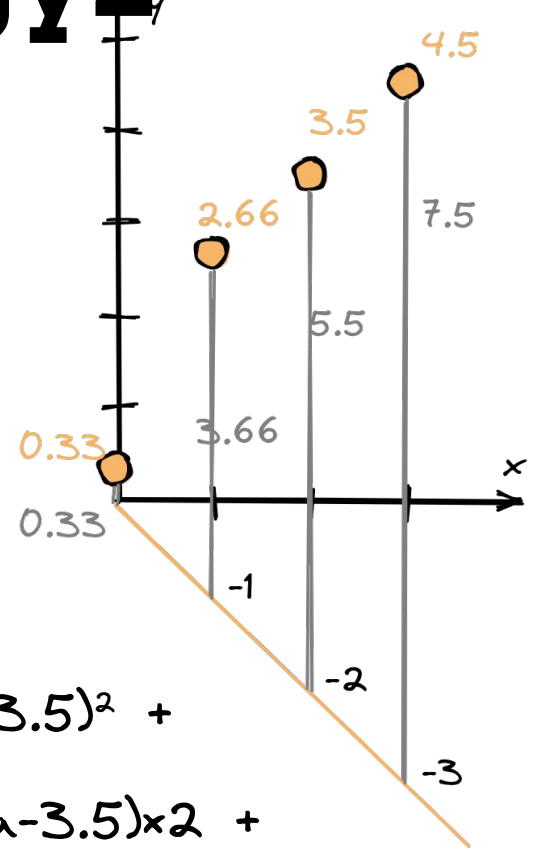


Finding best fit: Strategy2

Our second strategy will be based on measuring the acceleration of a decrease. Let's calculate our SSR:

$$\begin{aligned} SSR &= (ax_1 - 0.33)^2 + (ax_2 - 2.66)^2 + \\ &+ (ax_3 - 3.5)^2 + (ax_4 - 4.5)^2 = \\ &= (a \times 0 - 0.33)^2 + (a \times 1 - 2.66)^2 + \\ &+ (a \times 2 - 3.5)^2 + (a \times 3 - 4.5)^2 = \\ &= -0.33^2 + (a - 2.66)^2 + (2a - 3.5)^2 + (3a - 4.5)^2 \end{aligned}$$

$$\begin{aligned} \frac{d}{da} &= \frac{d}{da} (-0.33)^2 + \frac{d}{da} (a - 2.66)^2 + \frac{d}{da} (2a - 3.5)^2 + \\ &+ \frac{d}{da} (3a - 4.5)^2 = 0 + 2 \times (a - 2.66) \times 1 + 2 \times (2a - 3.5) \times 2 + \\ &= 2 \times (3a - 4.5) \times 3 = 2a - 2 \times 2.66 + 4 \times 2a = \\ &= -4 \times 3.5 + 6 \times 3a - 6 \times 4.5 = \underline{28a - 46.32} \end{aligned}$$



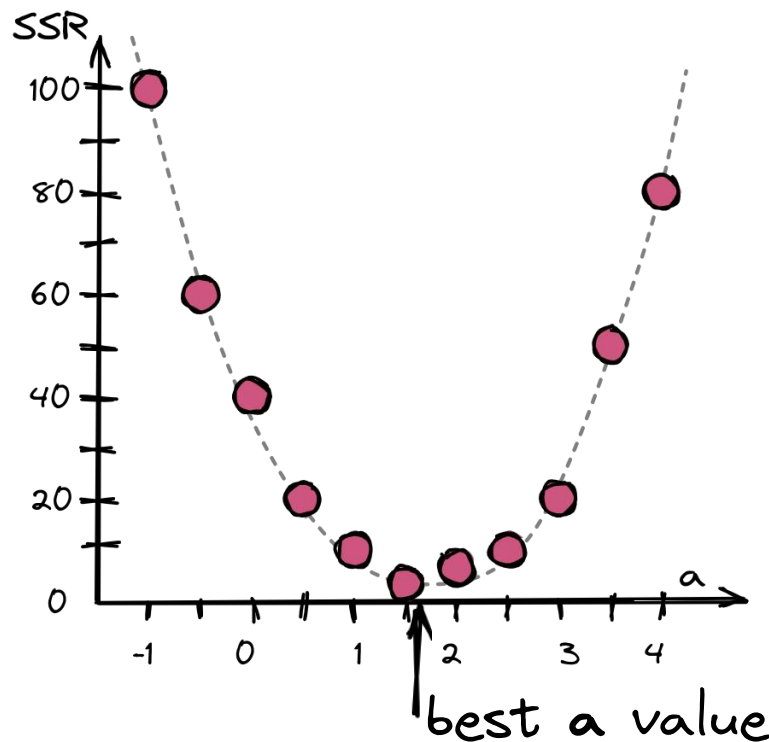
Finding best fit: Strategy2

Our derivative is quite simple: $28a - 46.32$

From a math perspective our solution can be found as:

$$\frac{d}{da} = 0 \rightarrow 28a - 46.32 = 0 \rightarrow a = 1.654$$

Using ordinary least squares: $a = 1.654$



But what if our derivative function isn't that simple?

Gradient Descent: Step1

$$28a - 46.32$$

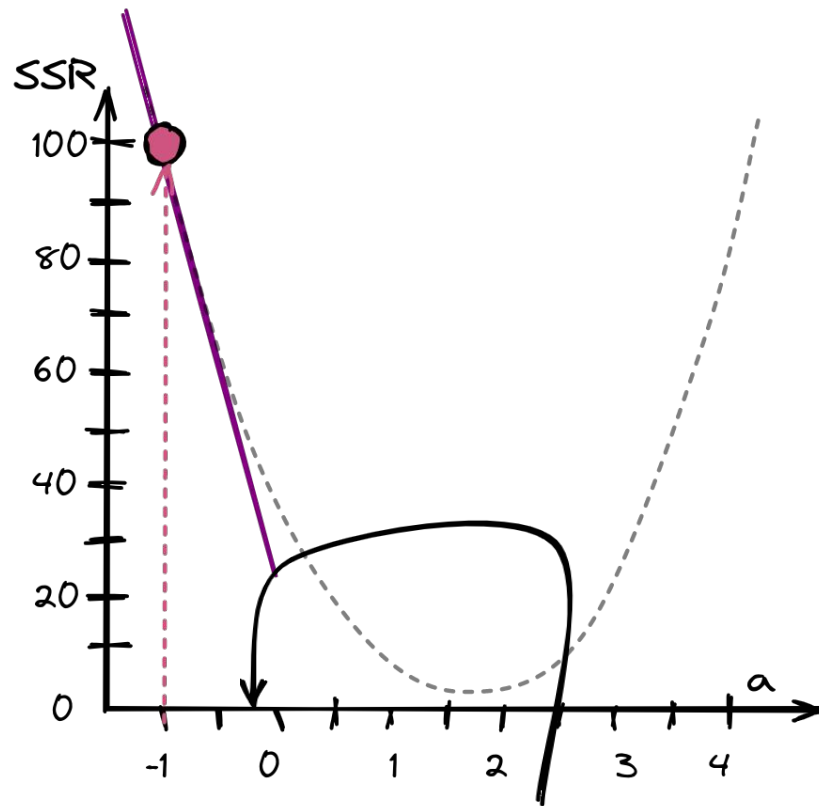
As, our $a = -1$, our tangent value:

$$28a - 46.32 = -74.32$$

In order to calculate our next a to check, we will require *learning rate* - how quick are we accelerating. Let our learning rate for this example = 0.01:

$$\text{step_size} = -74.32 \times 0.01 = -0.7432$$

$$\text{updated_a} = \text{old_a} - \text{step_size} = -1 - (-0.7432) = -0.2568$$



Gradient Descent: Step2,3, ...

Our derivative: $28a - 46.32$

current a value = -0.2568

Step2:

$$28 \times (-0.2568) - 46.32 = -53.51$$

$$\text{step_size} = -53.51 \times 0.01 = -0.5351$$

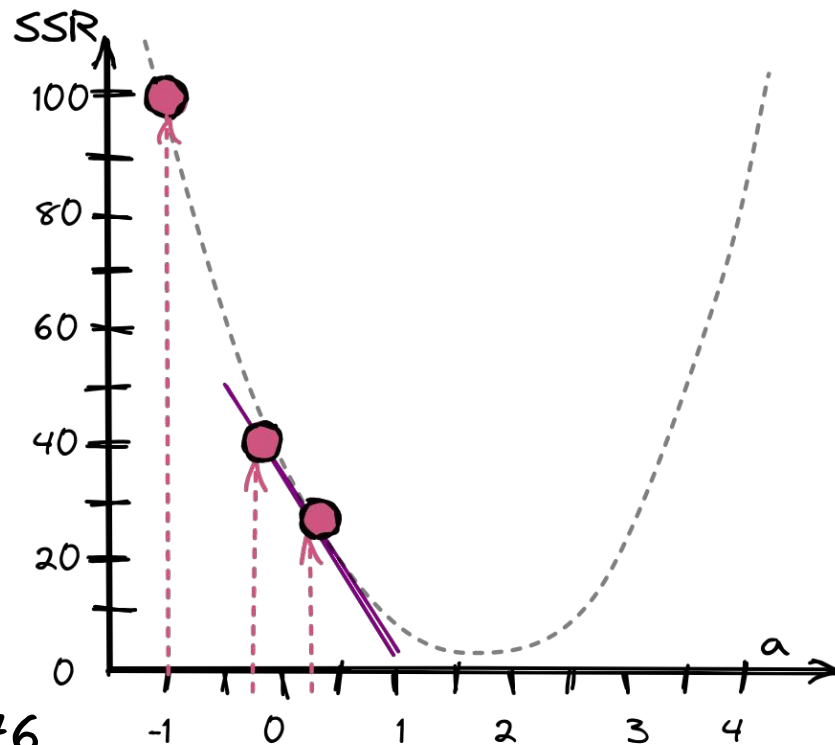
$$\text{new slope} = -0.2568 - (-0.5351) = 0.2783$$

Step3:

$$28 \times 0.2783 - 46.32 = -38.53$$

$$\text{step_size} = -38.53 \times 0.01 = -0.3853$$

$$\text{new slope} = 0.2783 - (-0.3853) = 0.663576$$

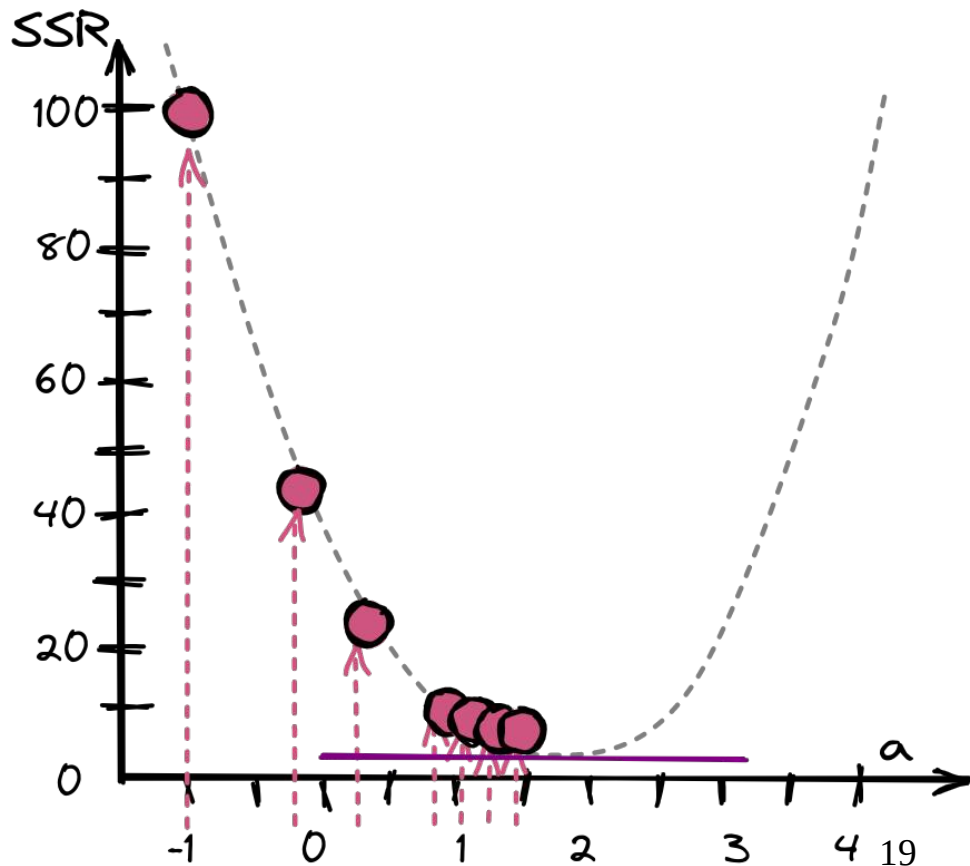


Gradient Descent: StepN

As one can notice, we're closing the gap to the best value with gradient descent much more efficiently. The drawback is that this continuous process, in theory, might never end.

To avoid such cases, the number of steps or the update difference is set.


If we increase the learning rate, the number of steps will decrease, but there is always a risk of "missing" a minimum.



Gradient Descent: Learning rate



Learning rate = 0.01

step	a	difference
1	-0.2568	-0.7432
2	0.2783	-0.5351
3	0.6636	-0.3853
4	0.941	-0.2774
5	1.1407	-0.1997
6	1.2845	-0.1438
7	1.388	-0.1035
8	1.4626	-0.0745
9	1.5163	-0.0537
10	1.5549	-0.0386



Learning rate = 0.025

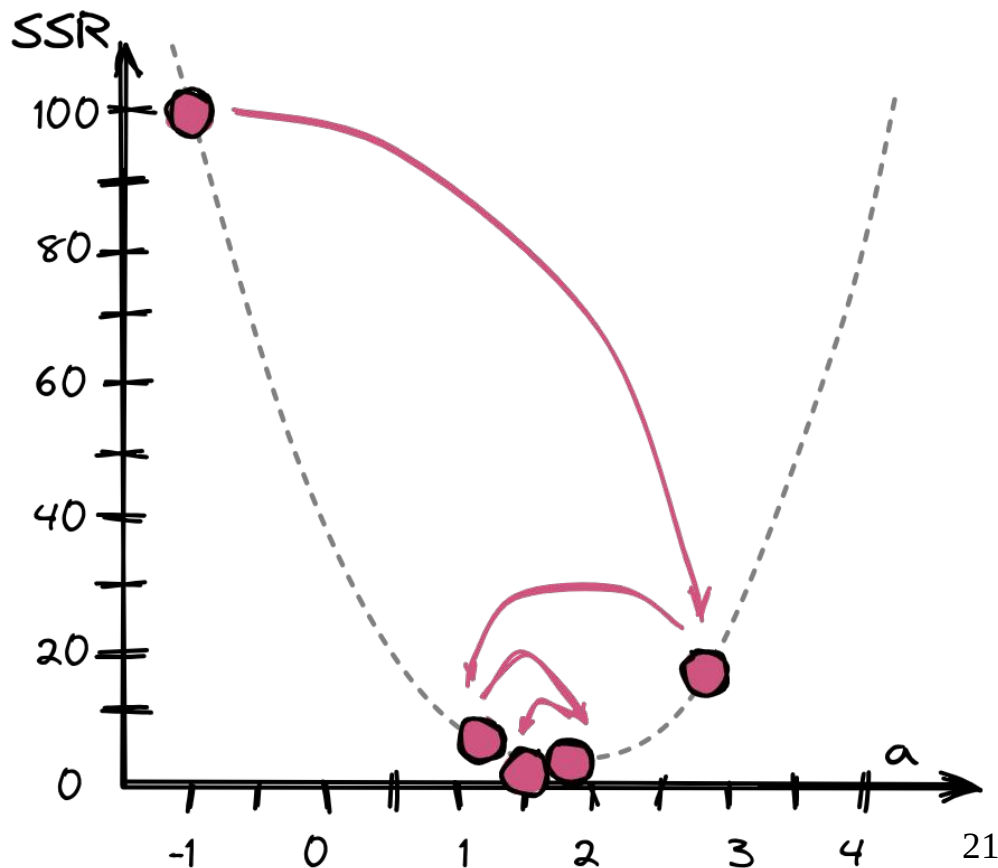
step	a	difference
1	0.858	-1.858
2	1.4154	-0.5574
3	1.5826	-0.1672
4	1.6328	-0.0502
5	1.6478	-0.015
6	1.6524	-0.0045
7	1.6537	-0.0014
8	1.6541	-0.0004
9	1.6542	-0.0001
10	1.6543	0



Gradient Descent: Learning rate

Learning rate = 0.05

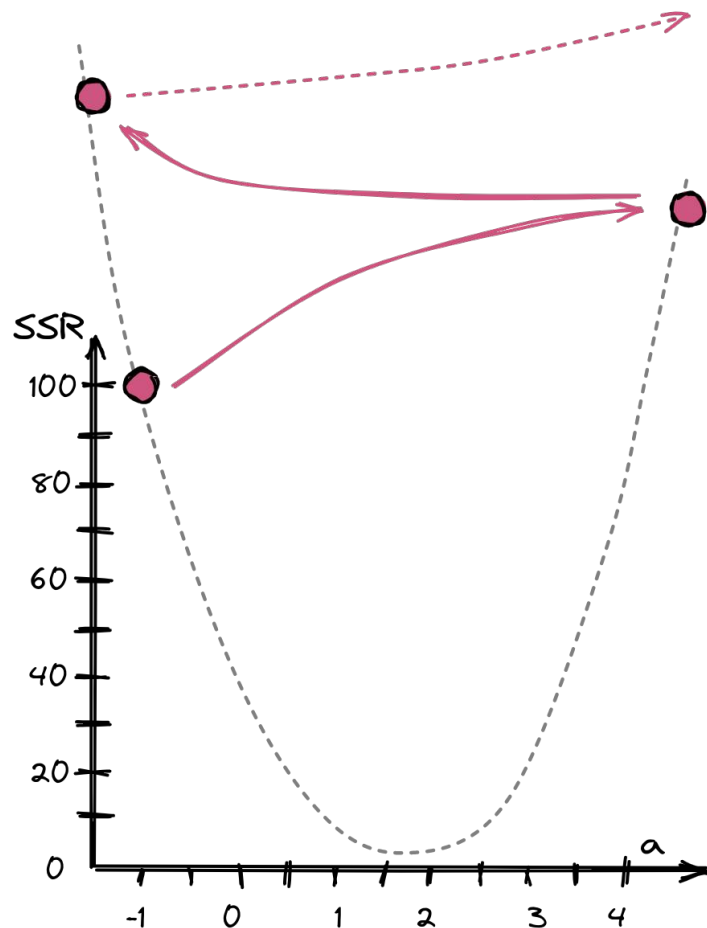
step	a	difference	
1	2.716	-3.716	⊗
2	1.2296	1.4864	⊗
3	1.8242	-0.5946	⊗
4	1.5863	0.2378	⊗
5	1.6815	-0.0951	⊗
6	1.6434	0.0381	⊗
7	1.6586	-0.0152	⊗
8	1.6525	0.0061	⊗
9	1.655	-0.0024	⊗
10	1.654	0.001	⊗



Gradient Descent: Learning rate

Learning rate = 0.075

step	a	difference
1	4.574	-5.574
2	-1.5574	6.1314
3	5.1871	-6.7445
4	-2.2319	7.419
5	5.929	-8.1609
6	-3.0479	8.977
7	6.8267	-9.8747
8	-4.0354	10.8621
9	7.913	-11.9484
10	-5.2302	13.1432



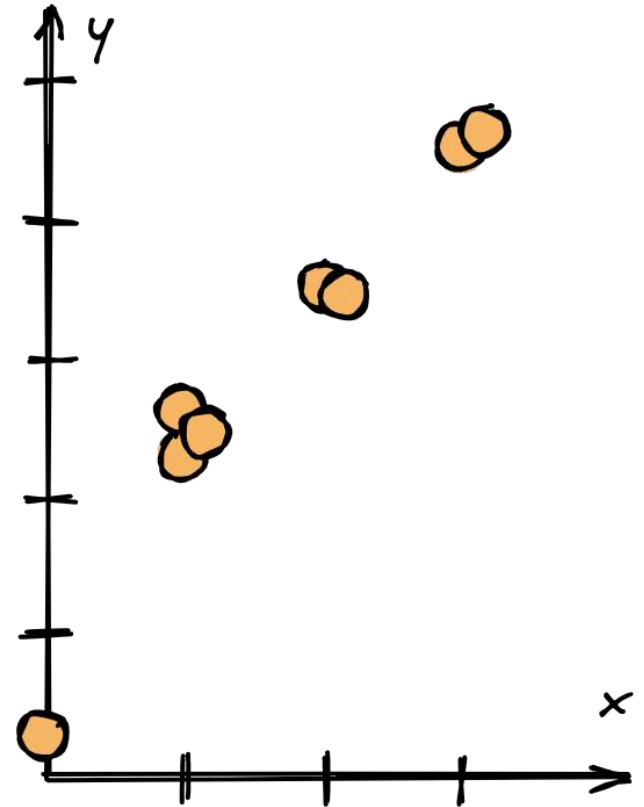
Stochastic Gradient Descent

Our "dataset" consists of only 1 feature column and 4 data rows. Let's imagine we have 1000 rows, which results in solving 1000 derivatives for each step.

If we had at 20 columns, that would increase the number of calculations, depending on the complexity, at least 20 times.

Having 500 steps, we might end up solving **10.000.000** equations per run.

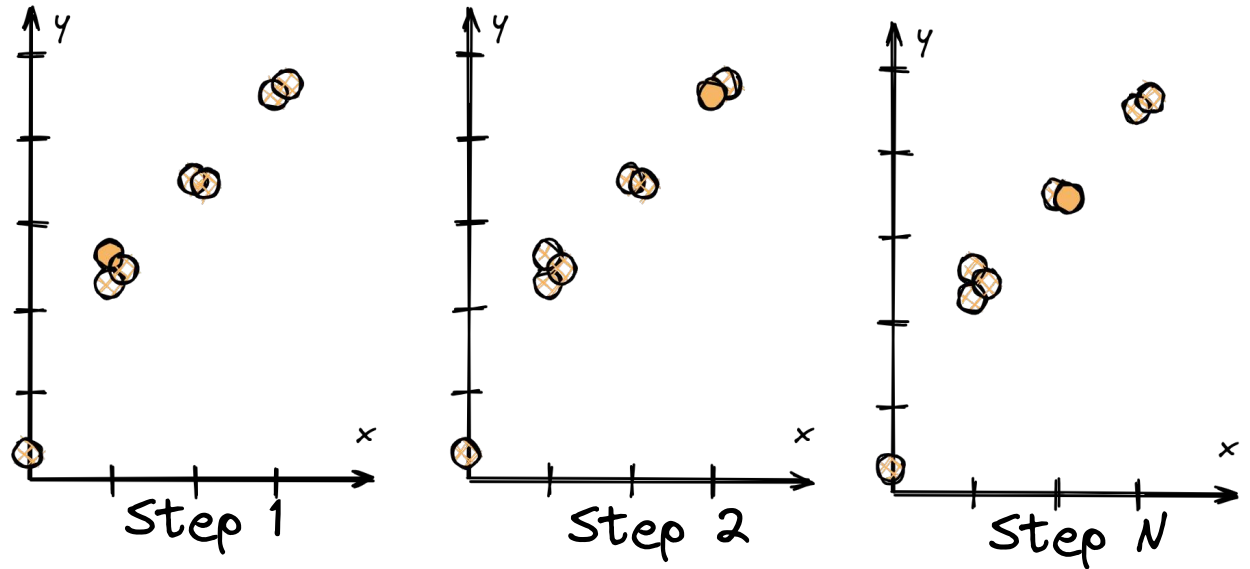
For the next example, we'll add 4 new data points.



Stochastic Gradient Descent

Stochastic gradient descent is a strategy to decrease the number of calculations without significant reducing the performance.

The "single point" idea is to use just one data point from all available data per step.

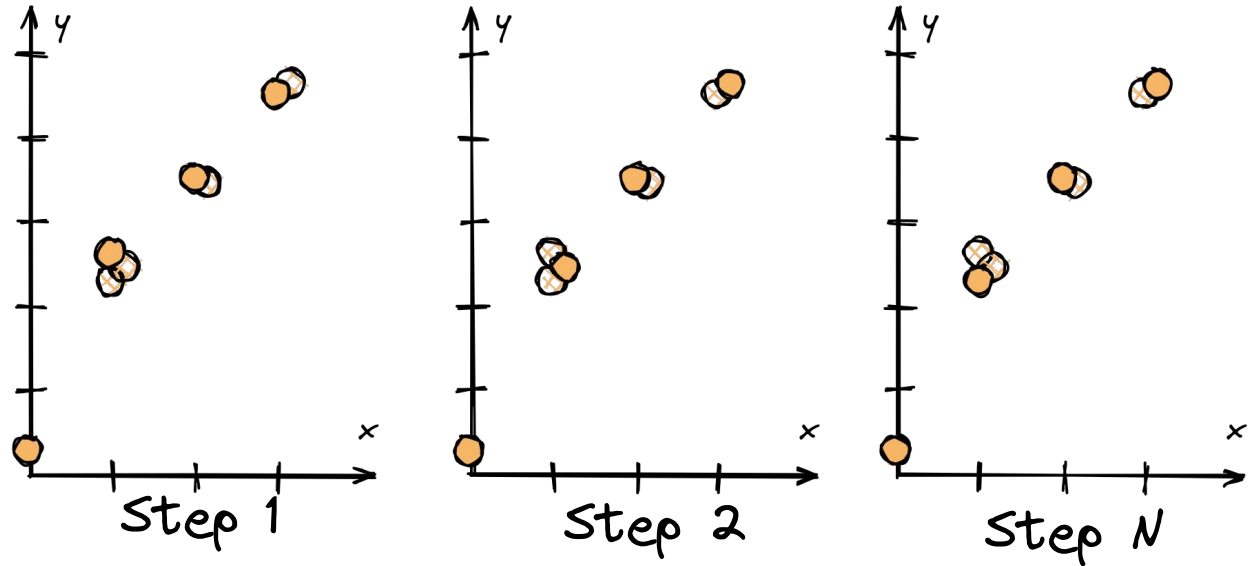


The "single point" strategy results in a more significant number of steps required but decreases the number of calculations per step.

Stochastic Gradient Descent

The "mini-batch" idea is to divide data on "clusters" and use one data point from each cluster per step.

In this case, we'll decrease the number of calculations per step without significantly increasing the number of steps required to complete the run.



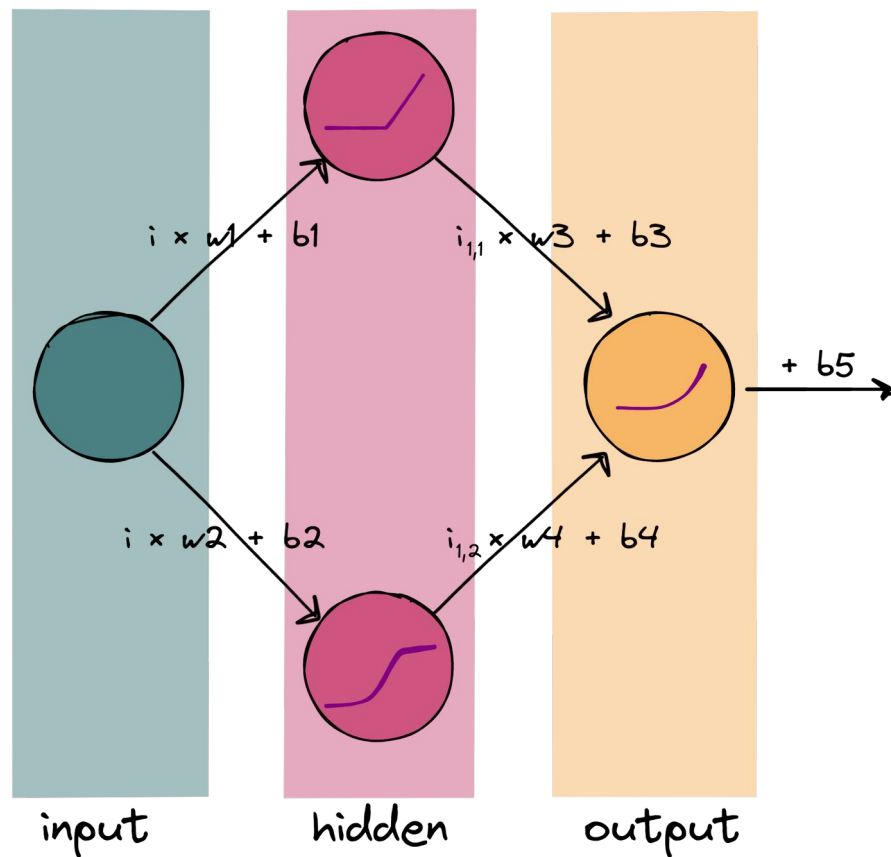
Gradient Descent

Our gradient descent algorithm will look as follows:

- Step1. Take the derivative of our loss function (SSR in our case) for each parameter (a in our case)
- Step2. Pick a first random value for all parameters (-1 for a in our case)
- Step3. Calculate the gradient
- Step4. Calculate the subsequent step size as the product of slope and learning rate
- Step5. Update parameter values
- Step6. If the maximum number of steps isn't reached or updated parameters can't be distinguished from the un-updated, perform Step3

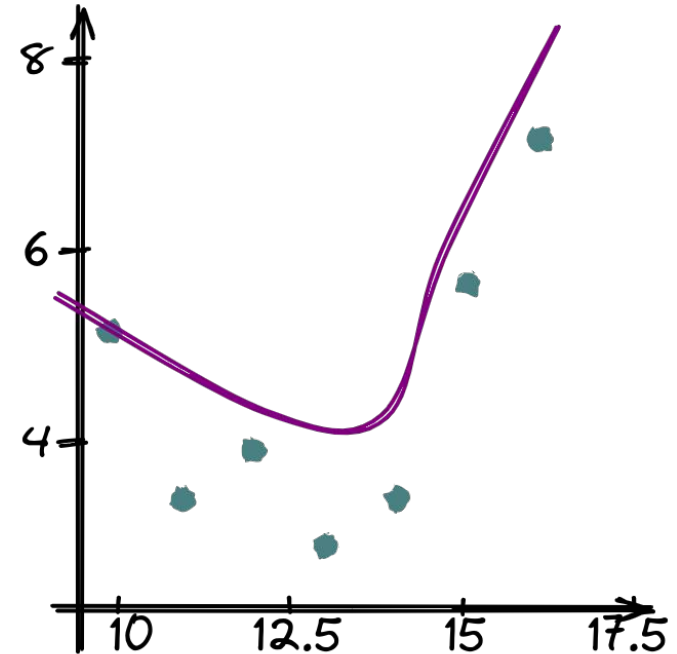
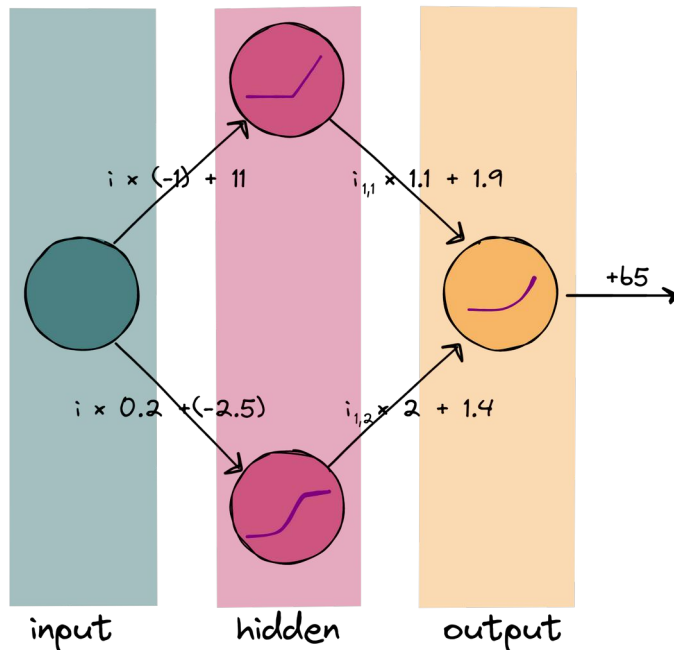
Backpropagation

Gradient descent is highly used to update weight and biases. Let's look at how the last bias b_5 is updated and will slowly go backward. Updating network parameters in such a fashion is called backpropagation.



Updating bias

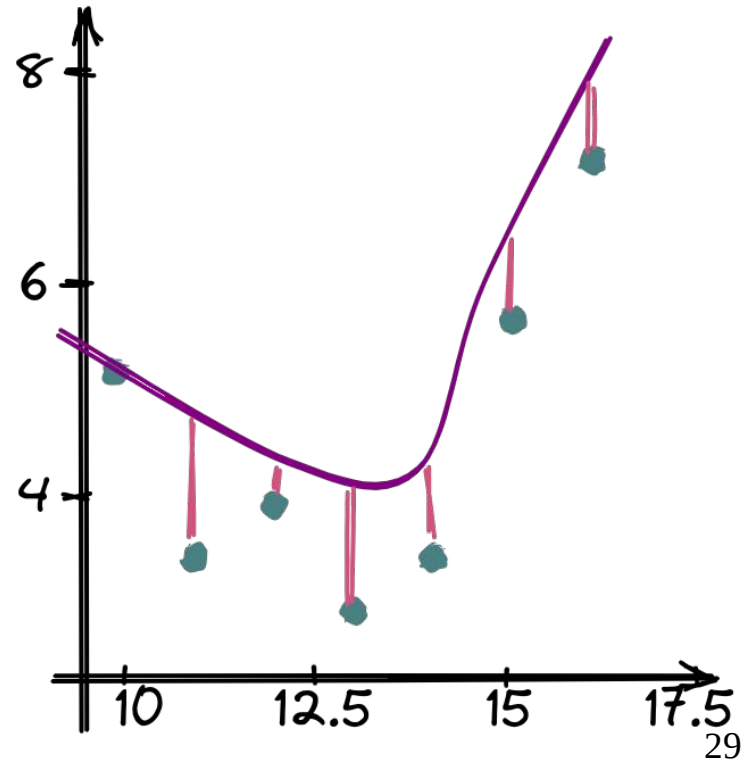
On the right is our fitting curve for the data generated by our neural network. As one may notice, the predicted curve has a high bias. Let's reduce it by updating b_5 .



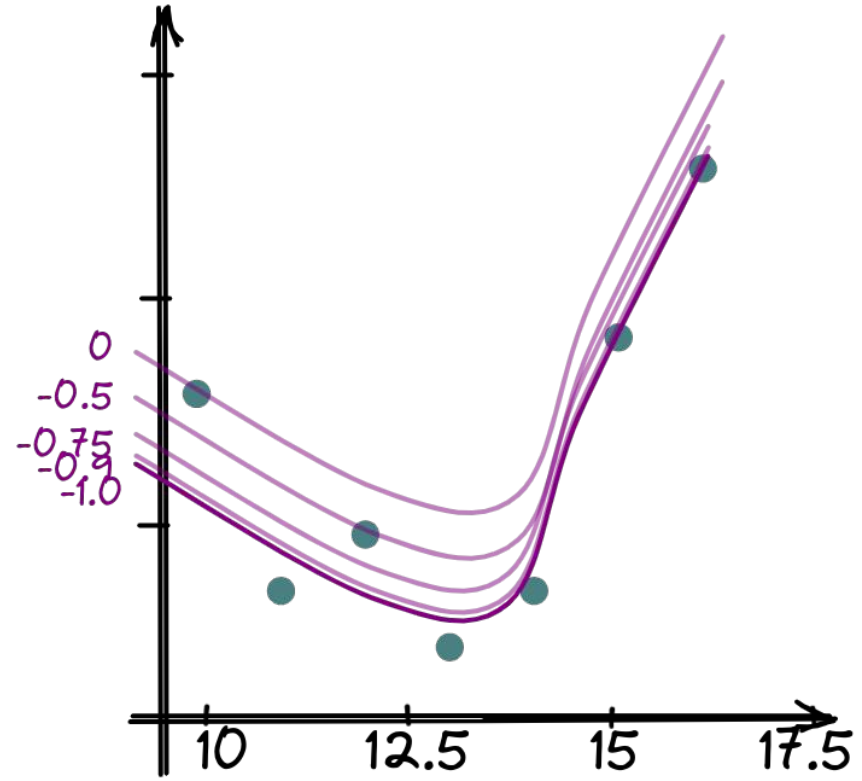
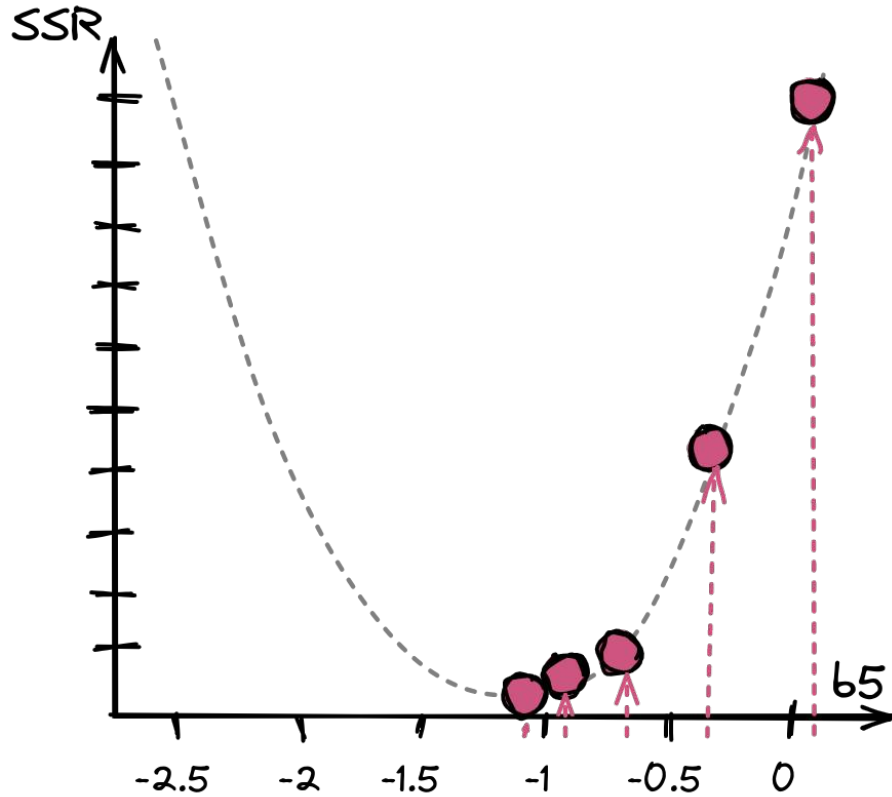
Updating bias

Similarly to the previous case, we can use the sum of squared residuals as the loss function. Finding the best fit is something we are already comfortable with.

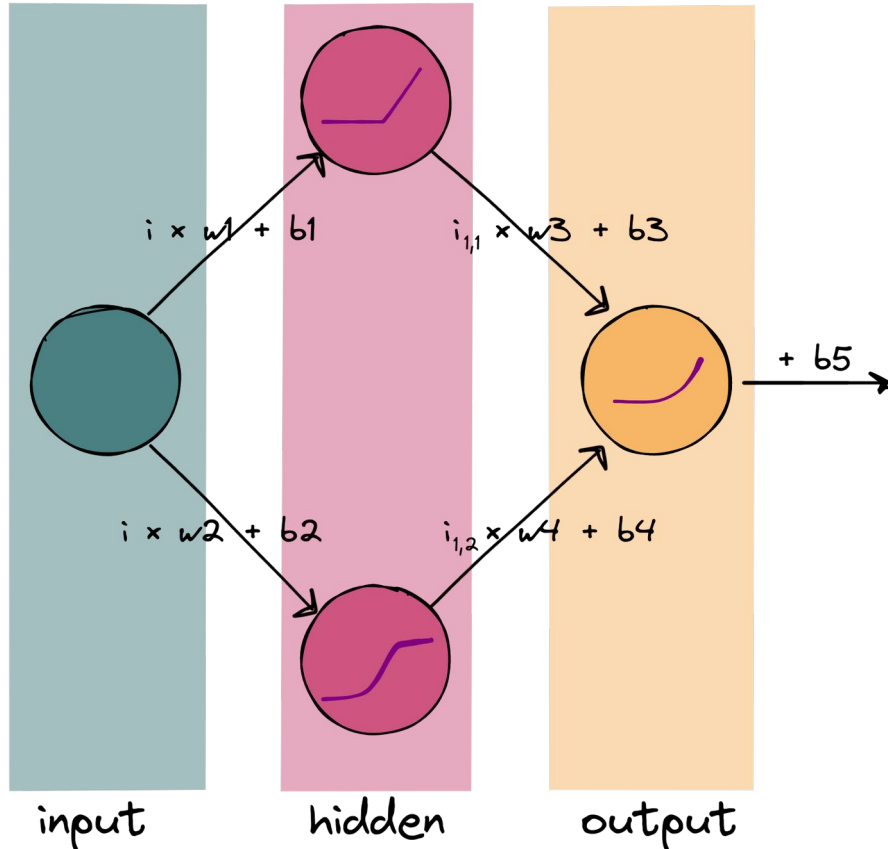
For this purpose, we'll use gradient descent.



Updating bias through GD



Updating bias



In order to calculate the gradient, we need to find the following derivative:

$$\frac{d(SSR)}{d(b5)}$$

As

$$SSR = \sum (True_i - Pred_i)^2$$

$$Pred_i = \text{softplus}(i_{1,1} \times w3 + b3 + i_{1,2} \times w4 + b4) + b5$$

Using the chain rule:

$$\frac{d(SSR)}{d(b5)} = \frac{d(SSR)}{d(Pred)} \times \frac{d(Pred)}{d(b5)}$$

Updating bias

$$\frac{d(SSR)}{d(b_5)} = \frac{d(SSR)}{d(Pred)} \times \frac{d(Pred)}{d(b_5)}$$

$$SSR = \sum (True_i - Pred_i)^2 \Rightarrow \frac{d(SSR)}{d(Pred)} = -2 \times \sum (True_i - Pred_i)$$

$$Pred_i = \text{softplus}(i_{i,1} \times w_3 + b_3 + i_{i,2} \times w_4 + b_4) + b_5 \Rightarrow$$

$$\frac{d(Pred)}{d(b_5)} = \frac{d(\text{softplus}(i_{i,1} \times w_3 + b_3 + i_{i,2} \times w_4 + b_4) + b_5)}{d(b_5)} =$$

$$= \frac{d(0 + b_5)}{d(b_5)} = 1$$

$$\frac{d(SSR)}{d(b_5)} = -2 \times \sum (True_i - Pred_i) \times 1$$

Updating weights and biases

$$\frac{d(\text{SSR})}{d(b_5)} = \frac{d(\text{SSR})}{d(\text{Pred})} \times \frac{d(\text{Pred})}{d(b_5)}$$

$$\frac{d(\text{SSR})}{d(b_3)} = \frac{d(\text{SSR})}{d(\text{Pred})} \times \frac{d(\text{Pred})}{d(b_3)}$$

$$\frac{d(\text{SSR})}{d(w_3)} = \frac{d(\text{SSR})}{d(\text{Pred})} \times \frac{d(\text{Pred})}{d(w_3)}$$

$$\frac{d(\text{SSR})}{d(b_1)} = \frac{d(\text{SSR})}{d(\text{Pred})} \times \frac{d(\text{Pred})}{d(i_{1,1})} \times \frac{d(i_{1,1})}{d(b_1)}$$

$$\frac{d(\text{SSR})}{d(w_1)} = \frac{d(\text{SSR})}{d(\text{Pred})} \times \frac{d(\text{Pred})}{d(i_{1,1})} \times \frac{d(i_{1,1})}{d(w_1)}$$

