NM 100
1923 - 2014

MCET Dr. MAHALINGAM
COLLEGE OF ENGINEERING AND TECHNOLOGY
Udumalai Road, Pollachi, Coimbatore District 642003
Estd. 1998 | AICTE Approved | Affiliated to Anna University
An Autonomous Institution

NAAC A++ GRADE
Cycle 2 (2018-2023)
The Highest Grade

19EICN1701 - Introduction to Machine Learning
(UNIT 3 - Session 1)

UNIT 3 : SUPERVISED LEARNING
TOPIC : Linear and Logistic Regression

OUTCOMES

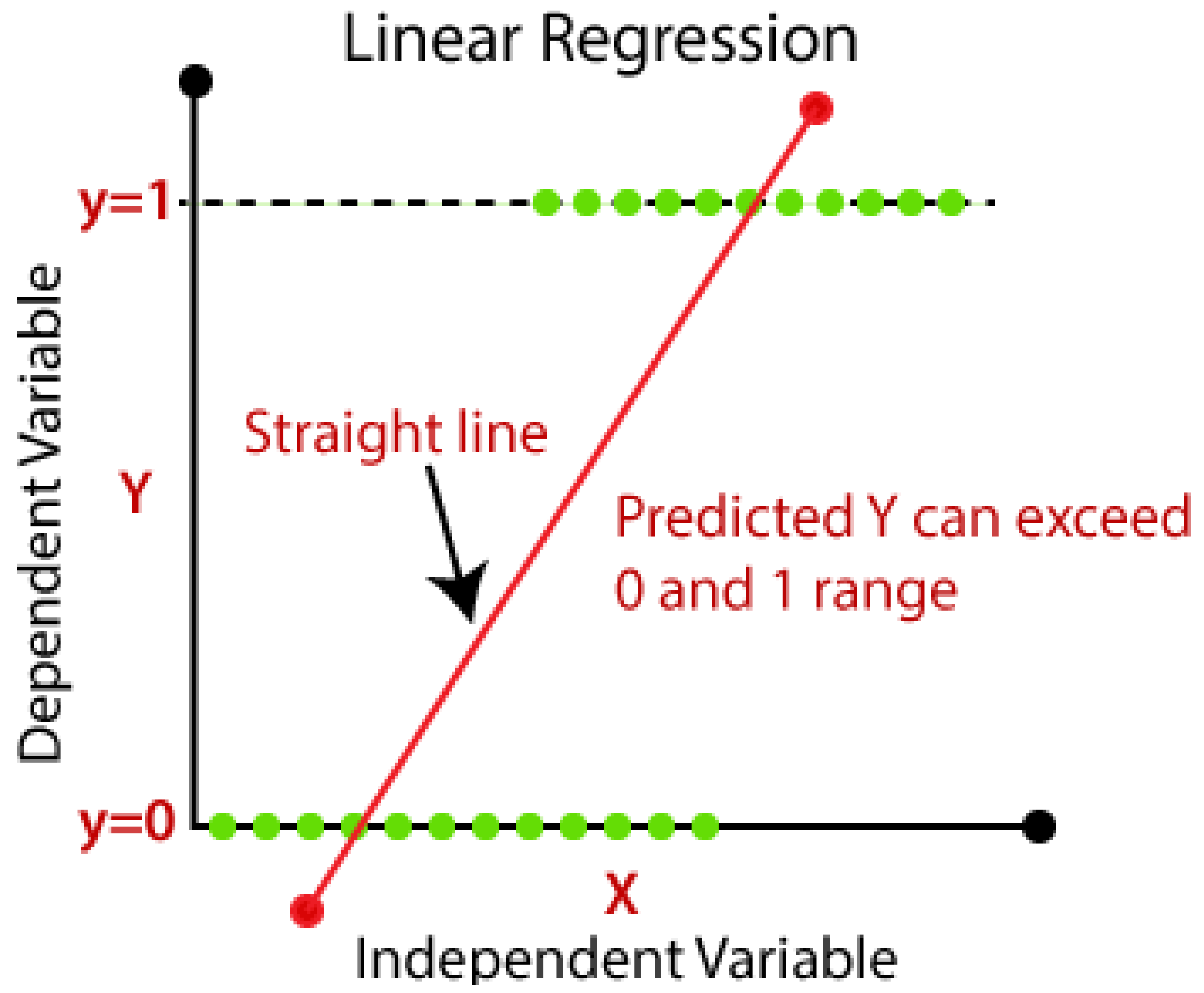
- **Course Outcome :**
  - Interpret the supervised learning techniques for classification
- **Session Outcome :**
  - Apply Linear and Logistic Regression on sample data

RECAP OF PREVIOUS SESSION

Unit II – Introduction

Linear Regression

- supervised learning technique
- used for solving Regression problems

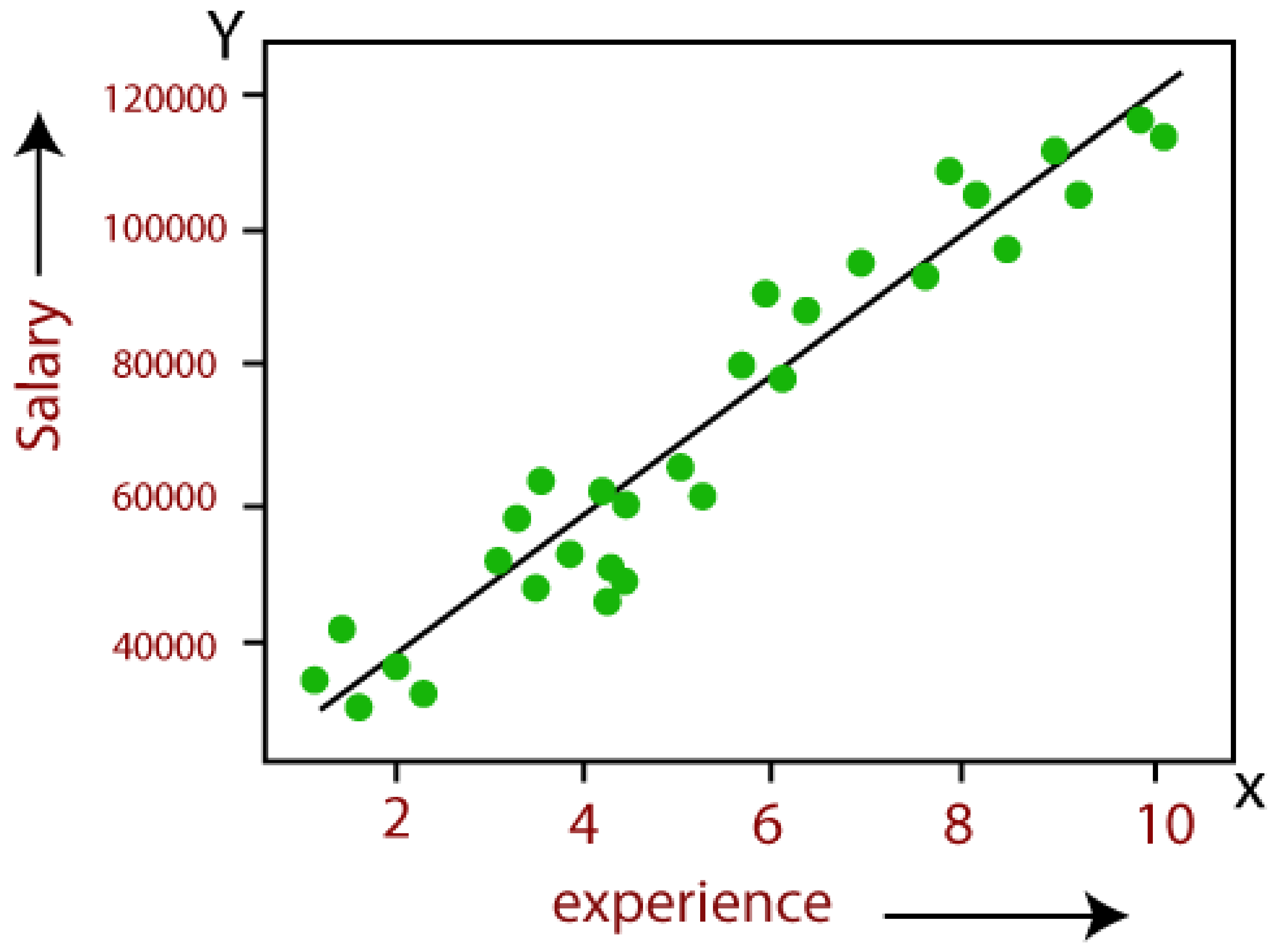


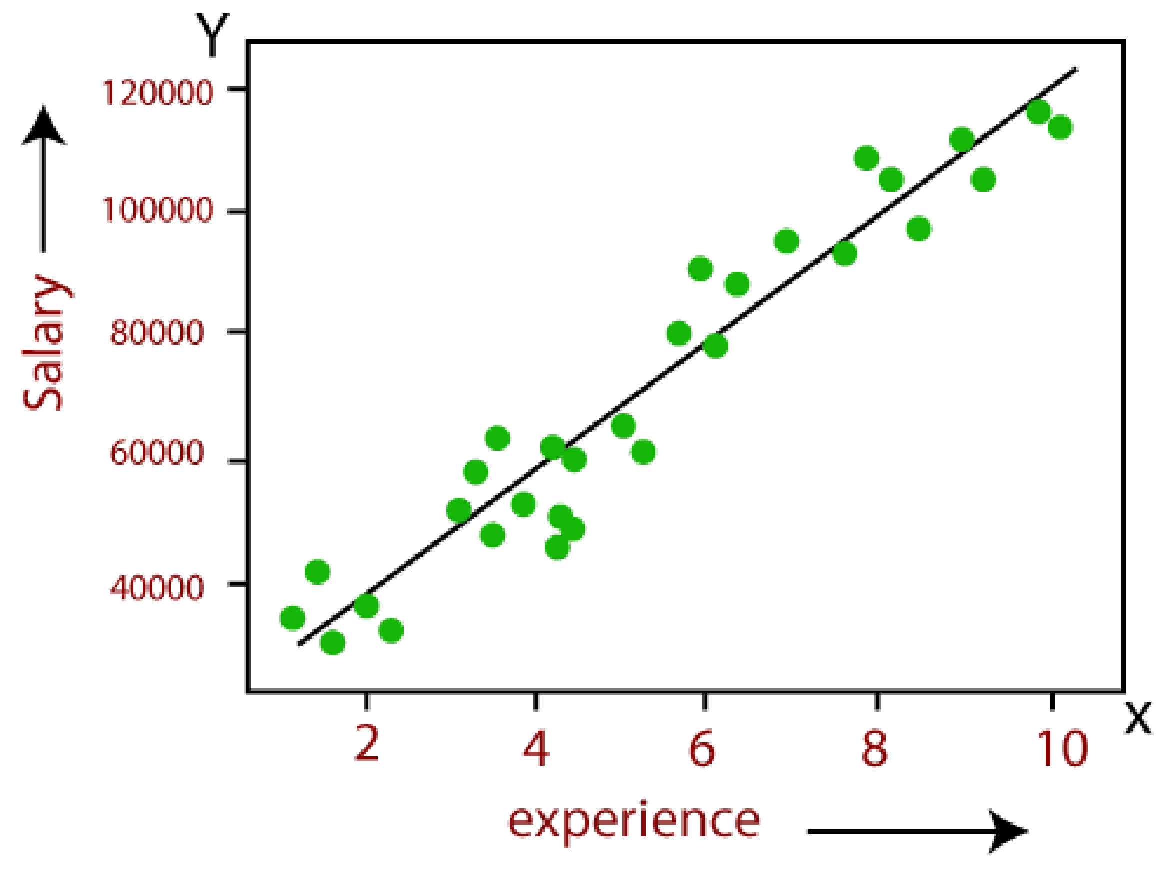
Linear Regression

- Linear Regression is one of the most simple ML algorithm
- Used for predicting the continuous dependent variable with the help of independent variables
- To find the that can accurately predict the output

- IF single independent variable is used for prediction then it is called Simple Linear Regression and if there are more than two independent variables then such regression is called as Multiple Linear Regression.

The output for Linear regression should only be the continuous values such as price, age, salary, etc.





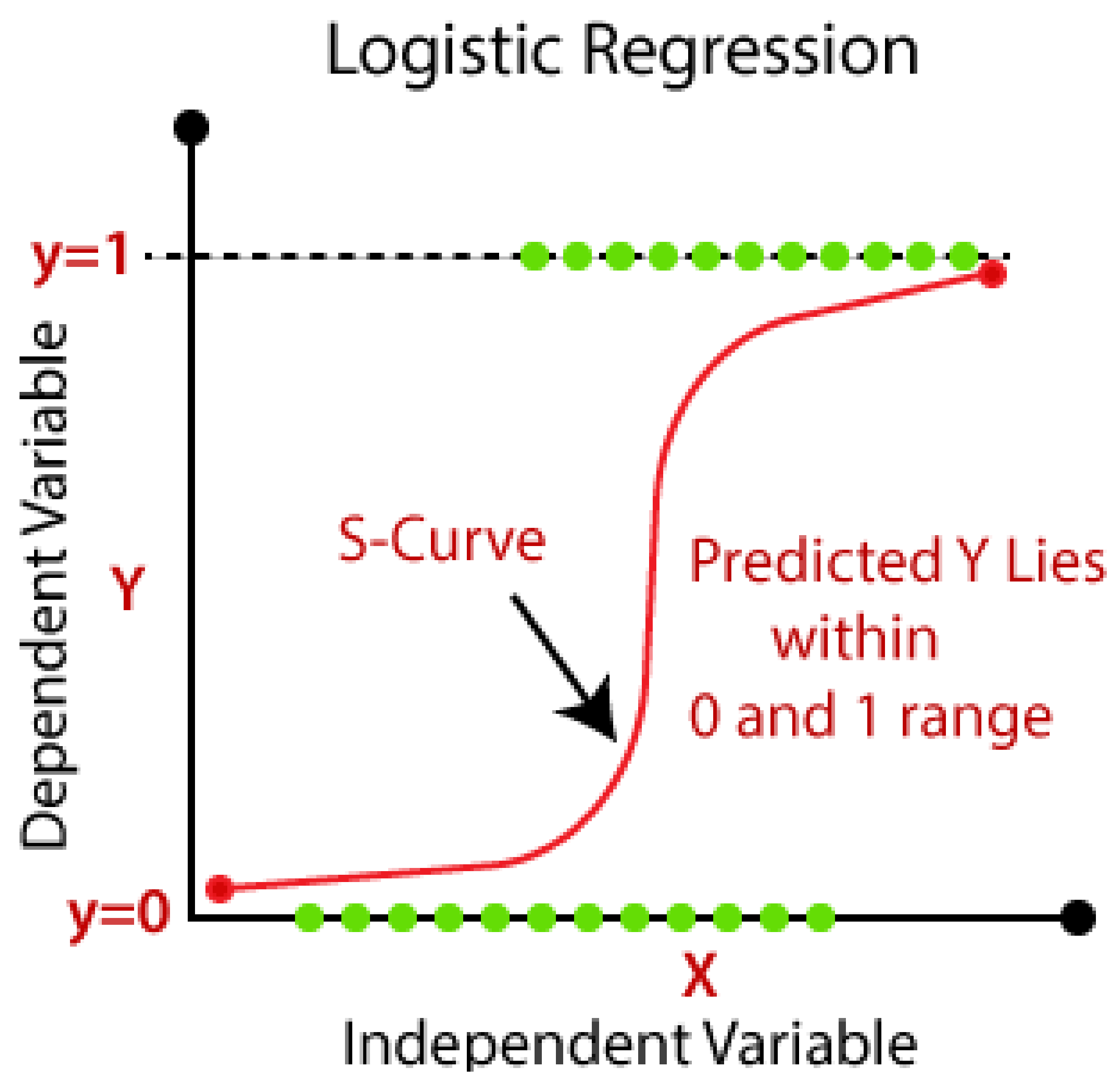
In above image the dependent variable is on Y-axis (salary) and independent variable is on x-axis(experience). The regression line can be written as:

$$y= a_0+a_1x+ \varepsilon$$

Where, a_0 and a_1 are the coefficients and ε is the error term

Logistic Regression

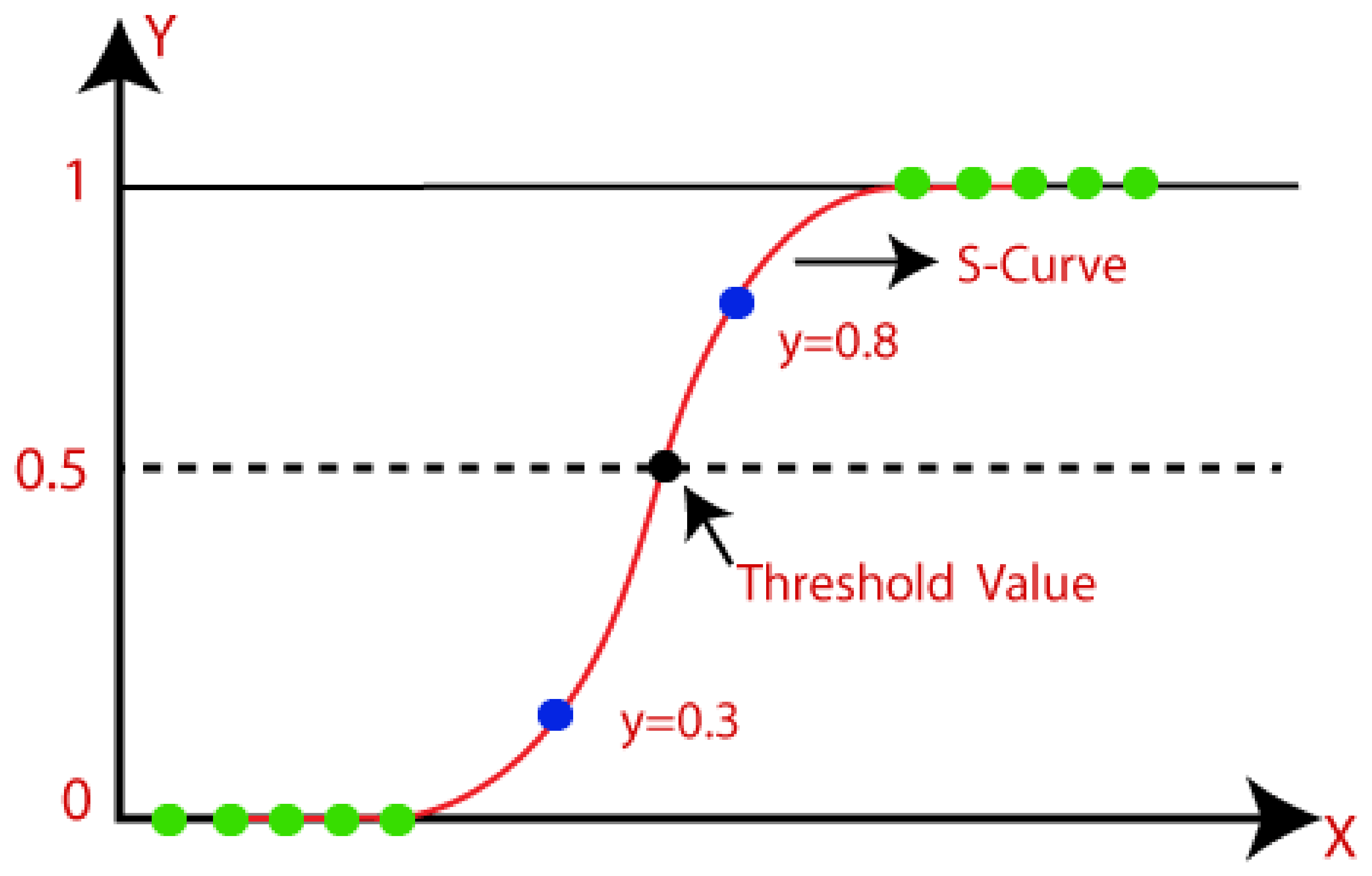
- Most popular ML algorithm under Supervised Learning techniques
- Used for Classification as well as for Regression problems



- used to predict the categorical dependent variable with the help of independent variables
- output of Logistic Regression problem can be only between the 0 and 1
- used where the probabilities between two classes is required
- Eg:whether it will rain today or not, either 0 or 1, true or false etc
- based on the concept of Maximum Likelihood estimation

Pass the weighted sum of inputs through an activation function that can map values in between 0 and 1.

Such activation function is known as **sigmoid function** and the curve obtained is called as sigmoid curve or S-curve.



$$\log\left[\frac{y}{1-y}\right] = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \cdots + b_nx_n$$

Difference b/w Linear and Logistic Regression

Linear Regression	Logistic Regression
Linear regression is used to predict the continuous dependent variable using a given set of independent variables.	Logistic Regression is used to predict the categorical dependent variable using a given set of independent variables.
Linear Regression is used for solving Regression problem.	Logistic regression is used for solving Classification problems.
In Linear regression, we predict the value of continuous variables.	In logistic Regression, we predict the values of categorical variables.
In linear regression, we find the best fit line, by which we can easily predict the output.	In Logistic Regression, we find the S-curve by which we can classify the samples.

Difference b/w Linear and Logistic Regression

Linear Regression	Logistic Regression
Least square estimation method is used for estimation of accuracy.	Maximum likelihood estimation method is used for estimation of accuracy.
The output for Linear Regression must be a continuous value, such as price, age, etc.	The output of Logistic Regression must be a Categorical value such as 0 or 1, Yes or No, etc.
In Linear regression, it is required that relationship between dependent variable and independent variable must be linear.	In Logistic regression, it is not required to have the linear relationship between the dependent and independent variable.
In linear regression, there may be collinearity between the independent variables.	In logistic regression, there should not be collinearity between the independent variable.

REFERENCES

- Aurélien Géron, “Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow”, Second edition, O‘Reilly Media, Inc,2019.

19EICN1701 - Introduction to Machine Learning

(UNIT 3 - Session 2)

UNIT 3 : SUPERVISED LEARNING

TOPIC : Eigenvectors and Eigenvalues

OUTCOMES

- **Course Outcome :**
  - Interpret the supervised learning techniques for classification
- **Session Outcome :**
  - Discuss on Eigen Values and Eigen vectors

Definition

The eigenvectors $\mathbf{x}$ and eigenvalues λ of a matrix A satisfy

$A\mathbf{x} = \lambda\mathbf{x}$

If A is an n x n matrix, then $\mathbf{x}$ is an n x 1 vector, and λ is a constant.

The equation can be rewritten as $(A - \lambda I)\,\mathbf{x} = 0$, where I is the
n x n identity matrix.

Computing Eigenvalues

Since **x** is required to be nonzero, the eigenvalues must satisfy

$\det(A - \lambda I) = 0$

which is called the *characteristic equation*. Solving it for values of λ gives the eigenvalues of matrix A.

2 X 2 Example

$$A = \begin{bmatrix} 1 & -2 \\ 3 & -4 \end{bmatrix} \qquad \text{so } A - \lambda I = \begin{bmatrix} 1 - \lambda & -2 \\ 3 & -4 - \lambda \end{bmatrix}$$

$\det(A - \lambda I) = (1 - \lambda)(-4 - \lambda) - (3)(-2)$

$= \lambda^2 + 3\lambda + 2$

Set $\lambda^2 + 3\lambda + 2$ to 0

Then = $\lambda = (-3 \;\; +/- \;\; \text{sqrt}(9-8))/2$

So the two values of λ are -1 and -2.

Finding the Eigenvectors

Once you have the eigenvalues, you can plug them into the equation A**x** = λ**x** to find the corresponding sets of eigenvectors **x**.

$$\begin{bmatrix} 1 & -2 \\ 3 & -4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = -1 \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \text{ so}$$

$$x_1 - 2x_2 = -x_1$$
$$3x_1 - 4x_2 = -x_2$$

$$\boxed{\begin{array}{l} (1)\;\; 2x_1 - 2x_2 = 0 \\ (2)\;\; 3x_1 - 3x_2 = 0 \end{array}}$$

These equations are not independent. If you multiply (2) by 2/3, you get (1).

The simplest form of (1) and (2) is $x_1 - x_2 = 0$, or just $x_1 = x_2$.

Since $x_1 = x_2$, we can represent all eigenvectors for eigenvalue -1 as multiples of a simple basis vector:

$$E = t \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$, where t is a parameter.

So $[1\ 1]^T$, $[4\ 4]^T$, $[3000\ 3000]^T$ are all possible eigenvectors for eigenvalue -1.

For the second eigenvalue (-2) we get

$$\begin{bmatrix} 1 & -2 \\ 3 & -4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = -2 \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$ so

$$x_1 - 2x_2 = -2x_1$$
$$3x_1 - 4x_2 = -2x_2$$

$$\boxed{\begin{array}{l} (1)\ \ 3x_1 - 2x_2 = 0 \\ (2)\ \ 3x_1 - 2x_2 = 0 \end{array}}$$

so eigenvectors are of the form t $\begin{bmatrix} 2 \\ 3 \end{bmatrix}$.

Generalization for 2 X 2 Matrices

If $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ then $\lambda = \dfrac{(a + d) +/- \text{sqrt}[(a+d)^2 - 4(ad - bc)]}{2}$

The discriminant (the part under the square root), can be simplified to get sqrt$[(a-d)^2 + 4bc]$.

If b = c, this becomes sqrt$[(a-d)^2 + (2b)^2]$
Since the discriminant is the sum of 2 squares, it has real roots.

We will be seeing some 2 x 2 matrices where indeed b = c,
so we'll be guaranteed a real-valued solution for the eigenvalues.

Another observation we will use:

For 2 x 2 matrix $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$,

$\lambda_1 + \lambda_2 = a + d$, which is called trace(A)
and
$\lambda_1\lambda_2 = ad - bc$, which is called det(A).

Finally, zero is an eigenvalue of A if and only if
A is singular and det(A) = 0.

REFERENCES

- Aurélien Géron, “Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow”, Second edition, O‘Reilly Media, Inc,2019.

19EICN1701 - Introduction to Machine Learning
(UNIT 3 - Session 3)

UNIT 3 : SUPERVISED LEARNING
TOPIC : Naïve Bayes Classifier

OUTCOMES

- **Course Outcome :**
    - Interpret the supervised learning techniques for classification
- **Session Outcome :**
    - Describe Naïve Bayes Classifier

Naïve Bayes Classifier Algorithm

- supervised learning algorithm
- based on **Bayes theorem**
- used for solving classification problems
- **It is a probabilistic classifier**
- used in *text classification* that includes a high-dimensional training dataset
- **spam filtration, Sentimental analysis, and classifying articles**.

Bayes' Theorem

- used to determine the probability of a hypothesis with prior knowledge

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

- P(A|B) is Posterior probability: Probability of hypothesis A on the observed event B.
- P(B|A) is Likelihood probability: Probability of the evidence given that the probability of a hypothesis is true
- P(A) is Prior Probability: Probability of hypothesis before observing the evidence.
- P(B) is Marginal Probability: Probability of Evidence.

Working of Naïve Bayes' Classifier

Suppose we have a dataset of **weather conditions** and corresponding target variable **"Play“**

1. Convert the given dataset into frequency tables.
2. Generate Likelihood table by finding the probabilities of given features.
3. Now, use Bayes theorem to calculate the posterior probability

- **Problem**: If the weather is sunny, then the Player should play or not?

Given Data	Outlook	Play
0	Rainy	Yes
1	Sunny	Yes
2	Overcast	Yes
3	Overcast	Yes
4	Sunny	No
5	Rainy	Yes
6	Sunny	Yes
7	Overcast	Yes
8	Rainy	No
9	Sunny	No
10	Sunny	Yes
11	Rainy	No
12	Overcast	Yes
13	Overcast	Yes

Likelihood table weather condition

Weather	No	Yes	
Overcast	0	5	5/14= 0.35
Rainy	2	2	4/14=0.29
Sunny	2	3	5/14=0.35
All	4/14=0.29	10/14=0.71	

Applying Bayes'theorem:

P(Yes|Sunny)= P(Sunny|Yes)*P(Yes)/P(Sunny)

P(Sunny|Yes)= 3/10= 0.3

P(Sunny)= 0.35

P(Yes)=0.71

So P(Yes|Sunny) = 0.3*0.71/0.35= **0.60**

P(No|Sunny)= P(Sunny|No)*P(No)/P(Sunny)

P(Sunny|NO)= 2/4=0.5

P(No)= 0.29

P(Sunny)= 0.35

So P(No|Sunny)= 0.5*0.29/0.35 = 0.41

$$P(Yes|Sunny) > P(No|Sunny)$$

For multiple attribute

- today = (Sunny, Hot, Normal, False)

$$P(Yes|today) = \frac{P(SunnyOutlook|Yes)P(HotTemperature|Yes)P(NormalHumidity|Yes)P(NoWind|Yes)P(Yes)}{P(today)}$$

$$P(No|today) = \frac{P(SunnyOutlook|No)P(HotTemperature|No)P(NormalHumidity|No)P(NoWind|No)P(No)}{P(today)}$$

Advantages of Naïve Bayes Classifier

- Naïve Bayes is one of the fast and easy ML algorithms to predict a class of datasets.
- It can be used for Binary as well as Multi-class Classifications.
- It performs well in Multi-class predictions as compared to the other Algorithms.
- It is the most popular choice for **text classification problems**.

Disadvantages of Naïve Bayes Classifier

- Naive Bayes assumes that all features are independent or unrelated, so it cannot learn the relationship between features.

Applications of Naïve Bayes Classifier

- It is used for **Credit Scoring**.
- It is used in **medical data classification**.
- It can be used in **real-time predictions** because Naïve Bayes Classifier is an eager learner.
- It is used in Text classification such as **Spam filtering** and **Sentiment analysis**.

REFERENCES

- Aurélien Géron, “Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow”, Second edition, O‘Reilly Media, Inc,2019.

NM 100
1923 - 2014

MCET Dr. MAHALINGAM
COLLEGE OF ENGINEERING AND TECHNOLOGY
Udumalai Road, Pollachi, Coimbatore District 642003
Estd. 1998 | AICTE Approved | Affiliated to Anna University
An Autonomous Institution

NAAC A++ GRADE
Cycle 2 (2018-2023)
The Highest Grade

19EICN1701 - Introduction to Machine Learning
(UNIT 3 - Session 4)

UNIT 3 : SUPERVISED LEARNING
TOPIC : Gradient Descent - **Optimization for ML**

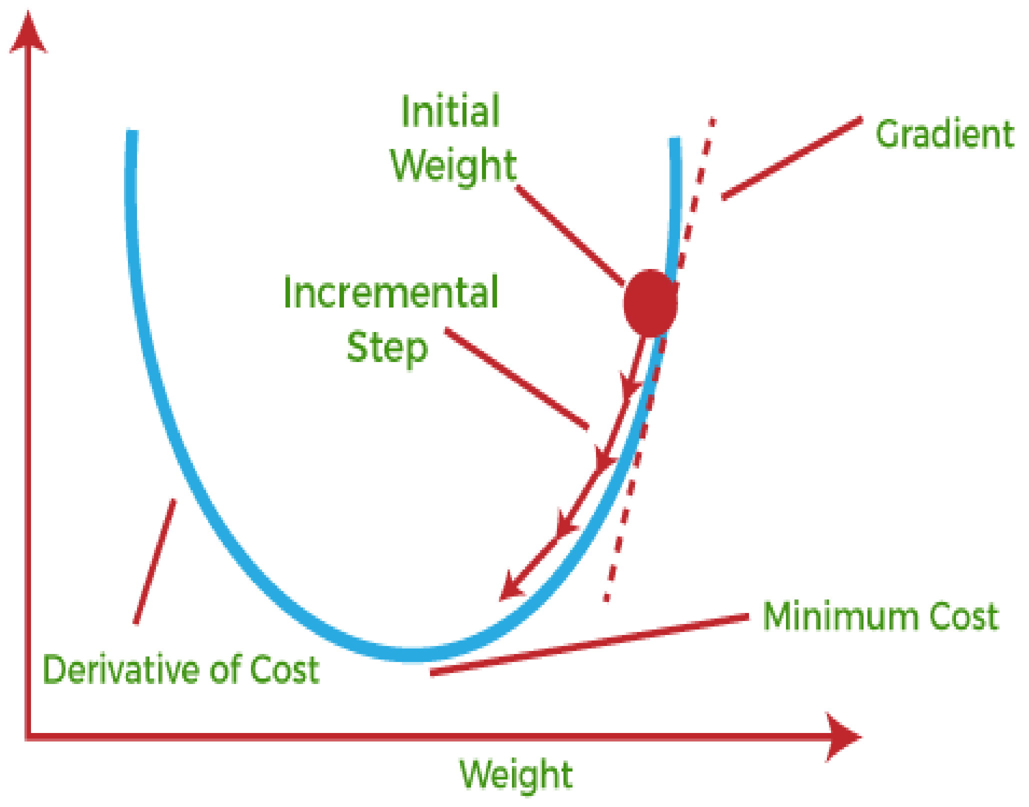
OUTCOMES

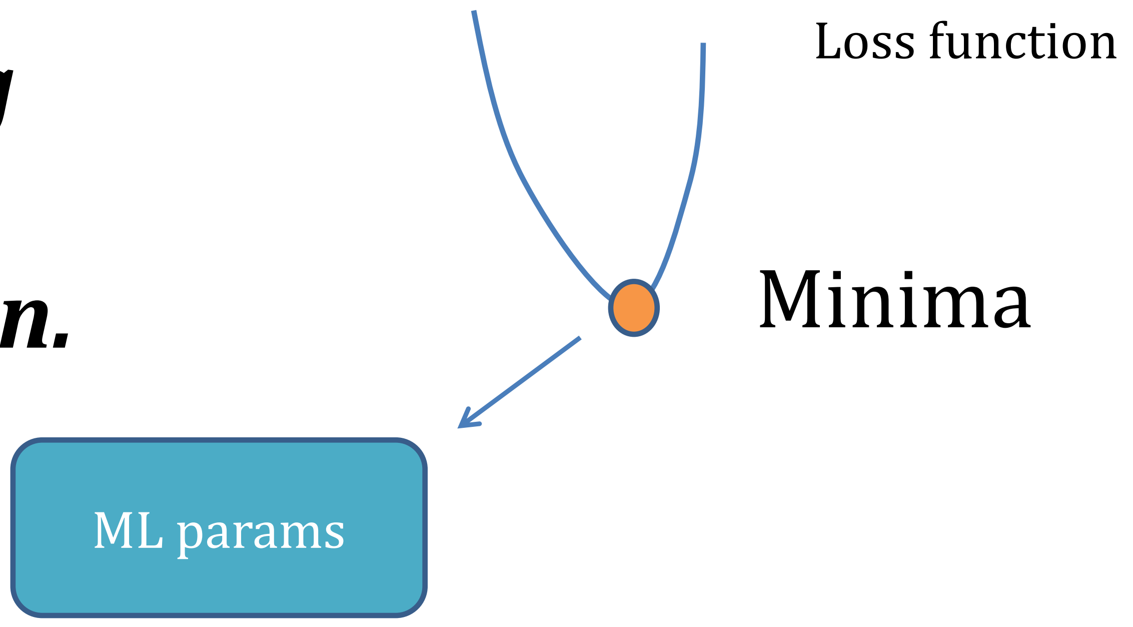
- **Course Outcome :**
  - Interpret the supervised learning techniques for classification
- **Session Outcome :**
  - Illustrate Gradient Descent model

Gradient Descent in Machine Learning

- Gradient Descent is known as one of the most commonly used optimization algorithms to train machine learning models by means of minimizing errors between actual and expected results.

- ***Gradient Descent is defined as one of the most commonly used iterative optimization algorithms of machine learning to train the machine learning and deep learning models. It helps in finding the local minimum of a function.***



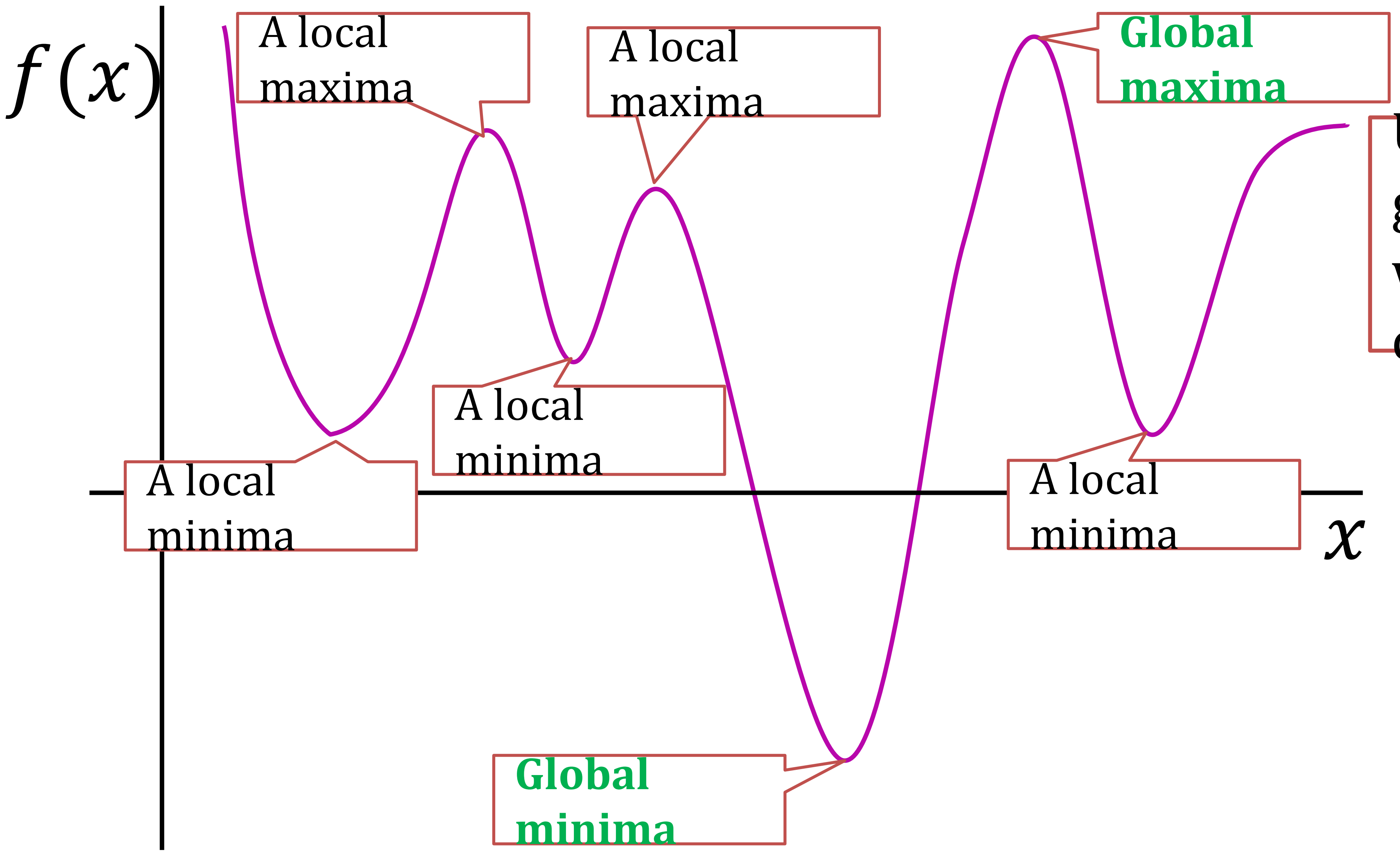


Functions and

The objective function of the ML problem we are solving (e.g., squared loss for regression)

Assume unconstrained for now, i.e., just a real-valued number/vector

- Many ML problems require us to optimize a function f of some variable(s) x
- For simplicity, assume f is a scalar-valued function of a scalar x ($f: \mathbb{R} \to \mathbb{R}$)



Usually interested in global optima but often want to find local optima, too

For deep learning models, often the local optima are what we can find (and they usually suffice) – more later

Will see what these are later

- Any function has one/more optima (maxima, minima), and maybe saddle points
- Finding the optima or saddles requires derivatives/gradients of the function

Derivatives

Will sometimes use $f'(x)$ to denote the derivative

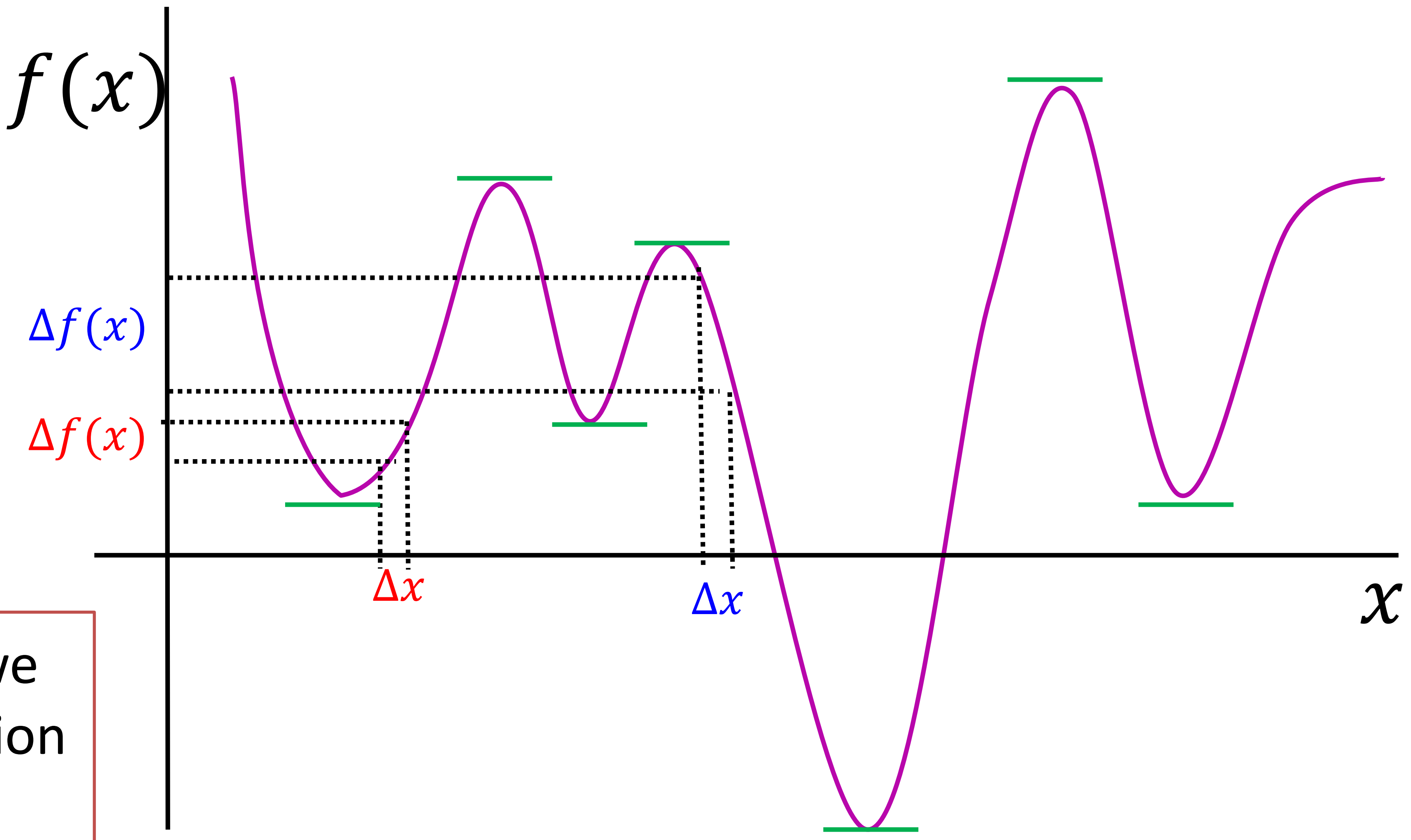
$f(x)$

- Magnitude of derivative at a point is the rate of change of the func at that point

$$\frac{df(x)}{dx} = \lim_{\Delta x \to 0} \frac{\Delta f(x)}{\Delta x}$$

Sign is also important: Positive derivative means f is increasing at x if we increase the value of x by a very small amount; negative derivative means it is decreasing

Understanding how f changes its value as we change x is helpful to understand optimization (minimization/maximization) algorithms



- Derivative becomes zero at stationary points (optima or saddle points)
  - The function becomes "flat" ($\Delta f(x) = 0$ if we change x by a very little at such points)
  - These are the points where the function has its maxima/minima (unless they are saddles)

Rules of Derivatives

Some basic rules of taking derivatives

- **Sum Rule:** $\left(f(x)+g(x)\right)' = f'(x)+g'(x)$
- **Scaling Rule:** $\left(a \cdot f(x)\right)' = a \cdot f'(x)$ if a is not a function of x
- **Product Rule:** $\left(f(x) \cdot g(x)\right)' = f'(x) \cdot g(x) + g'(x) \cdot f(x)$
- **Quotient Rule:** $(f(x)/g(x))' = \left(f'(x) \cdot g(x) - g'(x)f(x)\right)/\left(g(x)\right)^2$
- **Chain Rule:** $\left(f\left(g(x)\right)\right)' \overset{\text{def}}{=} (f \circ g)'(x) = f'\left(g(x)\right) \cdot g'(x)$

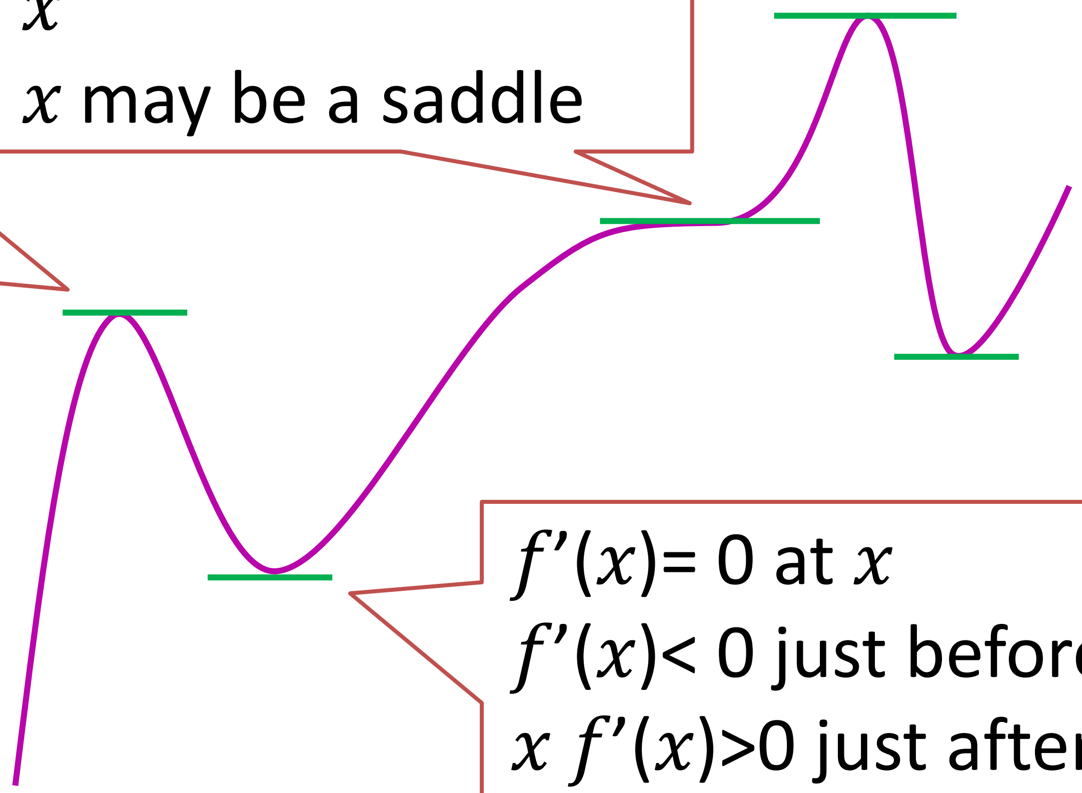
We already used some of these (sum, scaling and chain) when calculating the derivative for the linear regression model

Derivatives

- How the derivative itself changes tells us about the function's optima

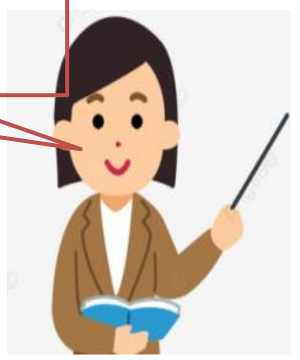
$f'(x)$= 0 at x,
$f'(x)$>0 just before x
$f'(x)$<0 just after x
x is a maxima

$f'(x)$= 0 at x
$f'(x)$= 0 just before x
$f'(x)$= 0 just after x
x may be a saddle



$f'(x)$= 0 at x
$f'(x)$< 0 just before x
$f'(x)$>0 just after x
x is a minima

$f'(x)$= 0 and $f''(x) < 0$
x is a maxima



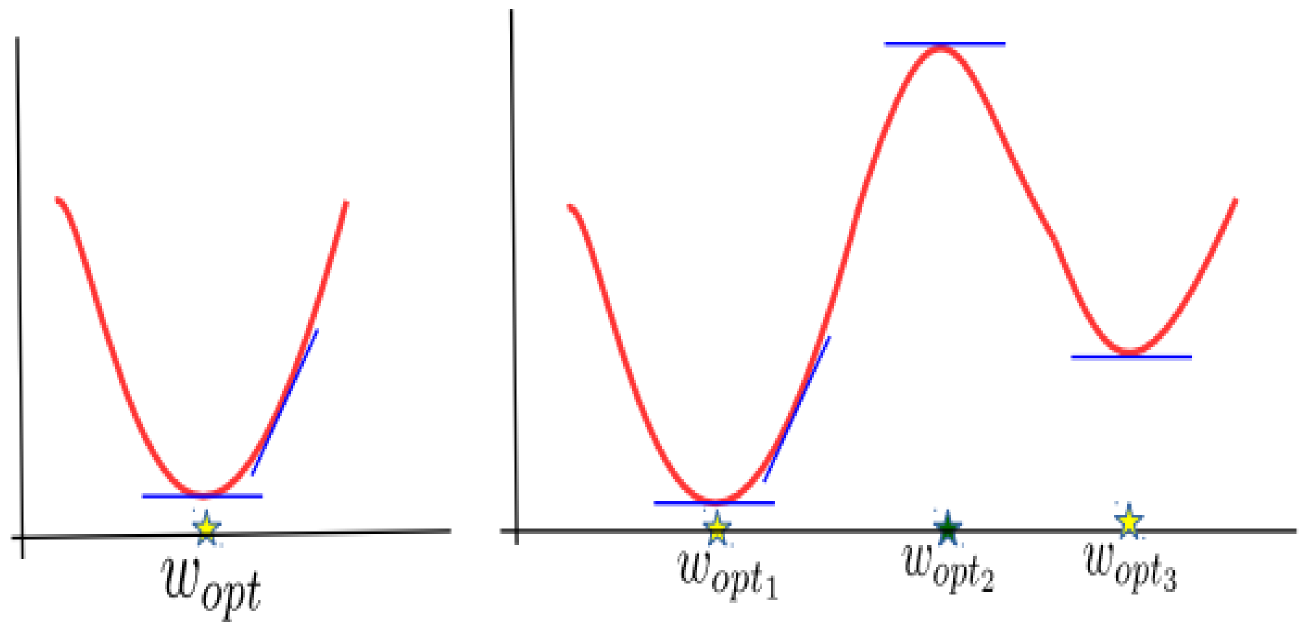
$f'(x)$= 0 and $f''(x) > 0$
x is a minima

$f'(x)$= 0 and $f''(x) = 0$
x may be a saddle. May need higher derivatives

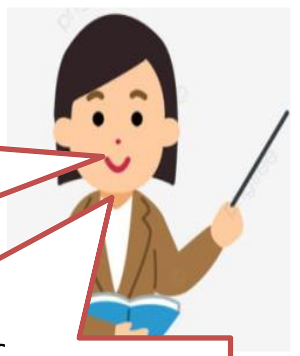
- The second derivative $f''(x)$ can provide this information

Optimization Using First-Order Optimality

- Very simple. Already used this approach for linear and ridge regression



Called "first order" since only gradient is used and gradient provides the first order info about the function being optimized



The approach works only for very simple problems where the objective is convex and there are no constraints on the values $\boldsymbol{w}$ can take

- First order optimality: The gradient $\boldsymbol{g}$ must be equal to zero at the optima

$$\boldsymbol{g} = \nabla_{\boldsymbol{w}}[L(\boldsymbol{w})] = \mathbf{0}$$

- Sometimes, setting $\boldsymbol{g} = \mathbf{0}$ and solving for $\boldsymbol{w}$ gives a closed form solution

- If closed form solution is not available, the gradient vector $\boldsymbol{g}$ can still be used in iterative optimization algos, like gradient descent

Optimization via Gradient Descent

Can I used this approach to solve maximization problems?

For max. problems we can use gradient ascent

$$w^{(t+1)} = w^{(t)} + \eta_t g^{(t)}$$

Iterative since it requires several steps/iterations to find the optimal solution

Fact: Gradient gives the direction of **steepest change** in function's value

Will move in the direction of the gradient

For convex functions, GD will converge to the global minima

Good initialization needed for non-convex functions

The learning rate very imp. Should be set carefully (fixed or chosen adaptively). Will discuss some strategies later

Gradient Descent

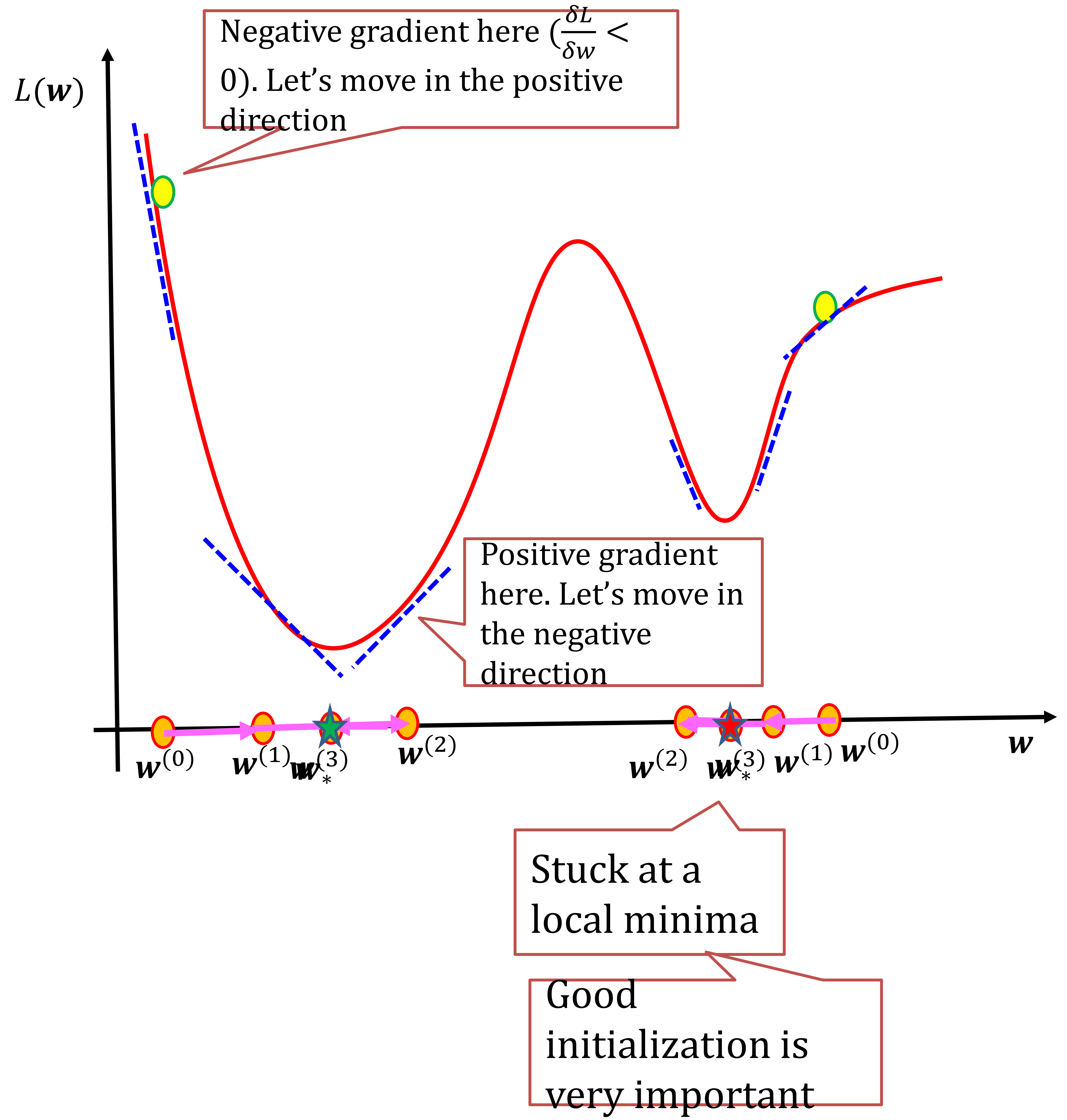
- Initialize $\boldsymbol{w}$ as $\boldsymbol{w}^{(0)}$
- For iteration $t = 0,1,2, \ldots$ (or until convergence)
  - Calculate the gradient $\boldsymbol{g}^{(t)}$ using the current iterates $\boldsymbol{w}^{(t)}$
  - Set the learning rate η_t
  - Move in the opposite direction of gradient

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta_t \boldsymbol{g}^{(t)}$$

Will see the justification shortly

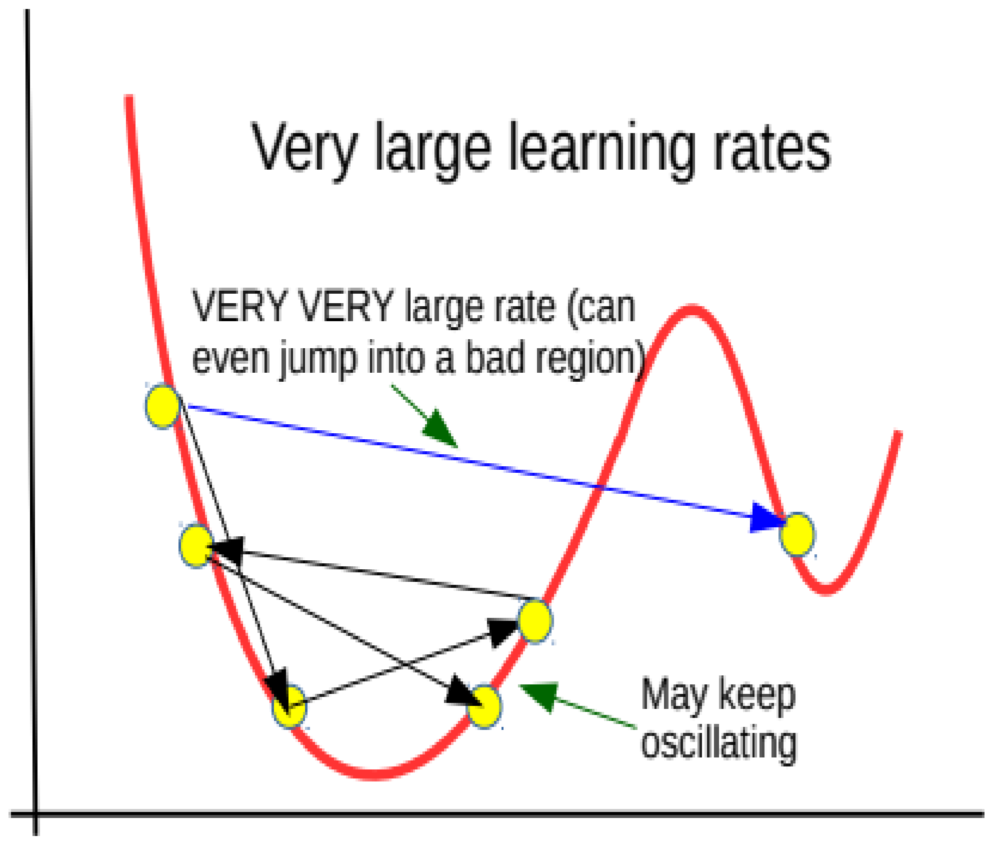
Sometimes may be tricky to to assess convergence? Will see some methods later

Gradient Descent: An Illustration

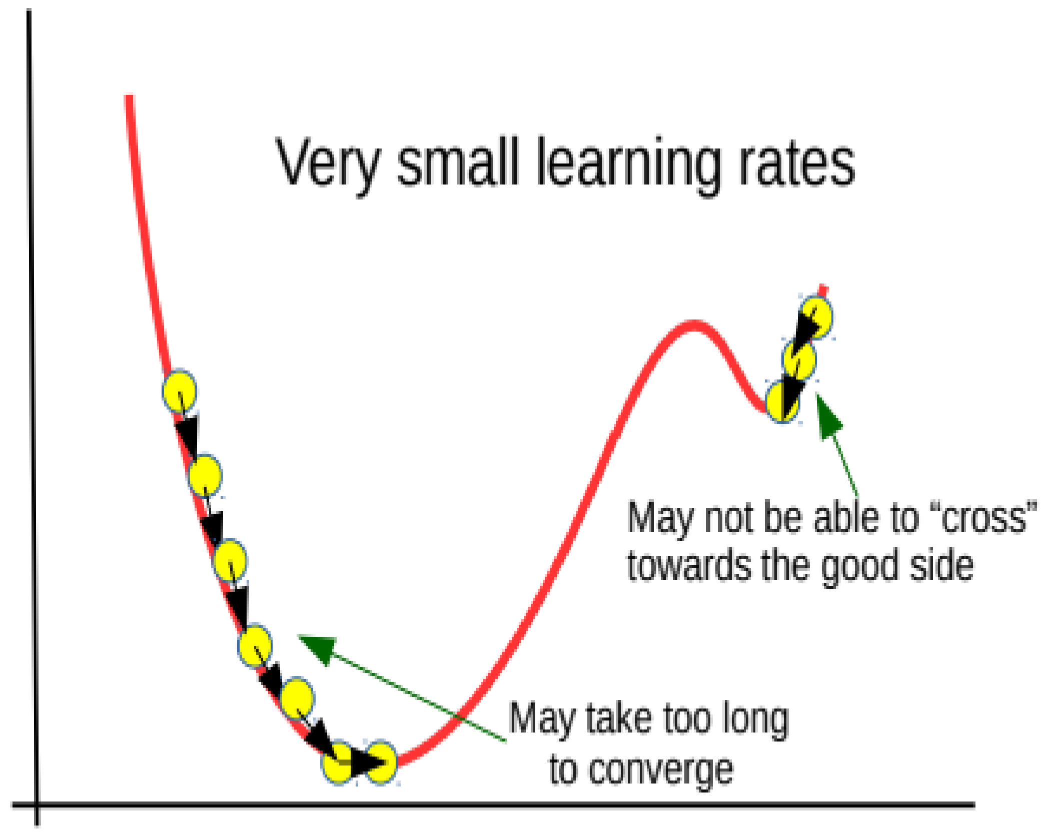


Negative gradient here ($\frac{\delta L}{\delta w} < 0$). Let's move in the positive direction
$L(\boldsymbol{w})$
Positive gradient here. Let's move in the negative direction
$\boldsymbol{w}^{(0)}$ $\boldsymbol{w}^{(1)}$ $\boldsymbol{w}^{(3)}$ $\boldsymbol{w}_*$ $\boldsymbol{w}^{(2)}$
$\boldsymbol{w}^{(2)}$ $\boldsymbol{w}^{(3)}$ $\boldsymbol{w}_*$ $\boldsymbol{w}^{(1)}$ $\boldsymbol{w}^{(0)}$
$\boldsymbol{w}$
Stuck at a local minima
Good initialization is very important

Learning rate is very importar



Very large learning rates
VERY VERY large rate (can even jump into a bad region)
May keep oscillating



Very small learning rates
May not be able to "cross" towards the good side
May take too long to converge

Types of gradient Descent:

- **Batch Gradient Descent:** This is a type of gradient descent which **processes all the training examples** for each iteration of gradient descent. But if the number of training examples is large, then batch gradient descent is computationally **very expensive**. Hence if the number of training examples is large, then batch gradient descent is not preferred. Instead, we prefer to use stochastic gradient descent or **mini-batch gradient descent**.

- **Stochastic Gradient Descent:** This is a type of gradient descent which **processes 1 training example per iteration.** Hence, the parameters are being updated even after one iteration in which only a single example has been processed. Hence this is quite faster than batch gradient descent. But again, when the number of training examples is large, even then it processes only one example which can be additional overhead for the system as the **number of iterations will be quite large.**

- **Mini Batch gradient descent:** This is a type of gradient descent which works faster than both batch gradient descent and stochastic gradient descent. Here *b* examples where $b<m$ are processed per iteration. So even if the number of training examples is large, it is processed in batches of b training examples in one go. Thus, it works for larger training examples and that too with lesser number of iterations.

- Gradient descent algorithm does not work for all functions. There are two specific requirements. A function has to be:
  - **differentiable**
  - **convex**

REFERENCES

- Aurélien Géron, “Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow”, Second edition, O‘Reilly Media, Inc,2019.

OUTCOMES

- **Course Outcome :**
  - Interpret the supervised learning techniques for classification
- **Session Outcome :**
  - Discuss on Maximum Likely hood and Minimum Description Length

Minimum Description Length Principle

- 'MDL' is a method for inductive inference…

– machine learning

– pattern recognition

– statistics

- …based on ideas from **data compression** (information theory)

- In contrast to most other methods, MDL automatically deals with overfitting, arguably the central problem in machine learning and statistics

Minimum Description Length Principle

- MDL is based on the correspondence between ‘regularity’ and ‘compression’
- The more you are able to compress a sequence of data, the more regularity you have detected in the data
- Example:
  00100100100100100100100100100100100::::001
  01011011100100111010001010101::::010

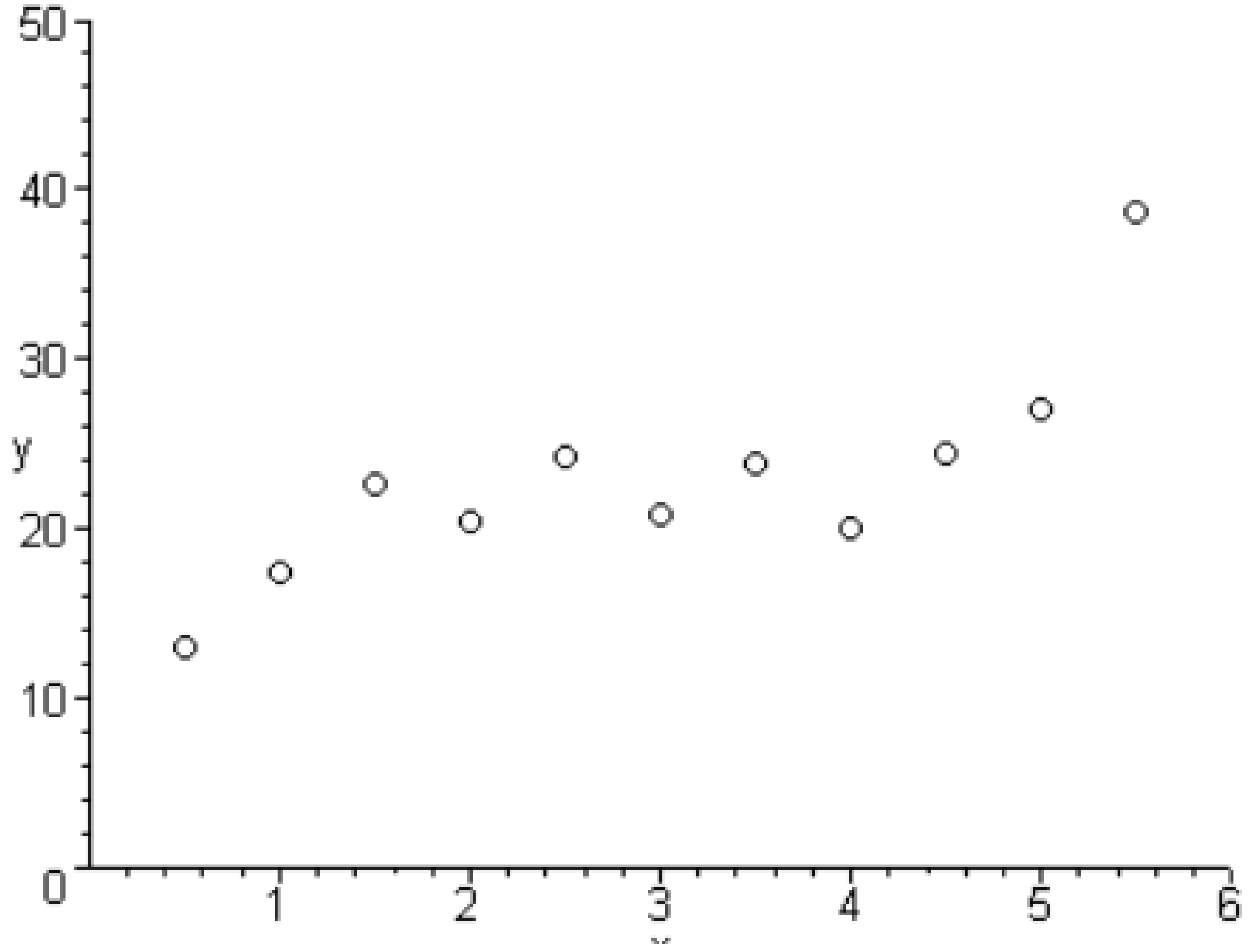
Model Selection/Overfitting

- Given data D and hypothesis spaces/models , which model best explains the data ?

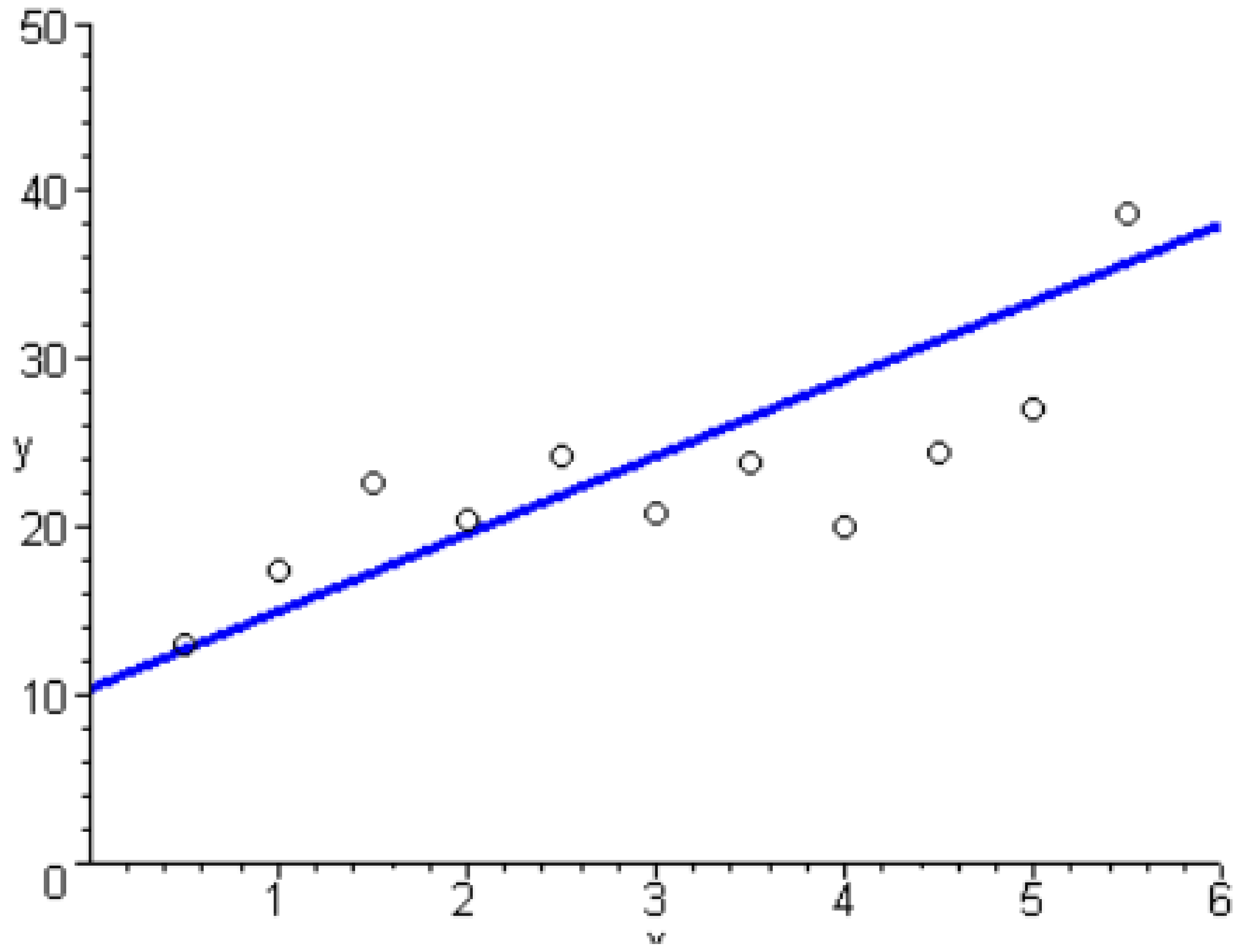
– Need to take into account
  - Complexity of models
  - Error (minus Goodness-of-fit)

– Example:
  - Selecting the degree of a polynomial in regression
  - Sum of squared errors

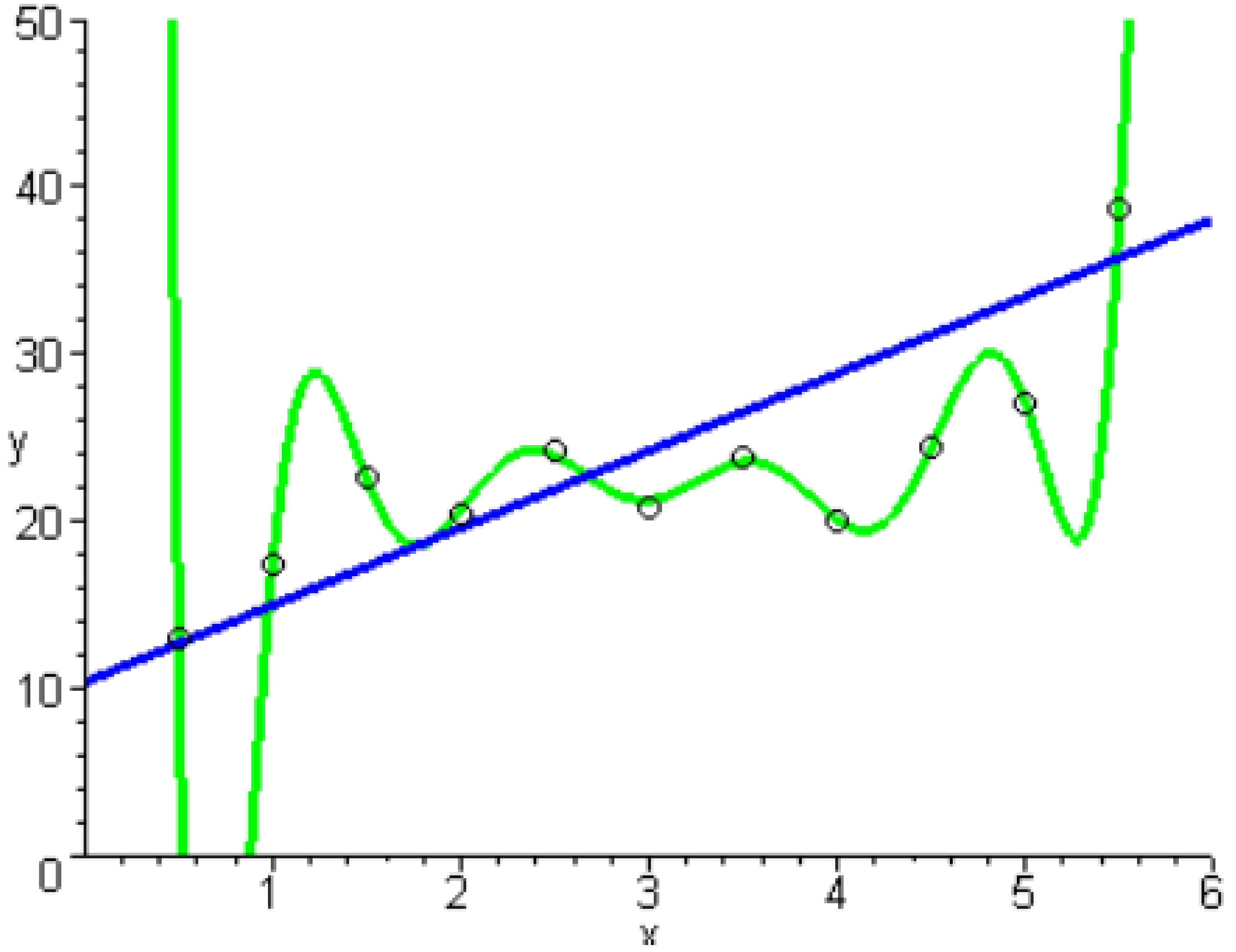
Example: Regression



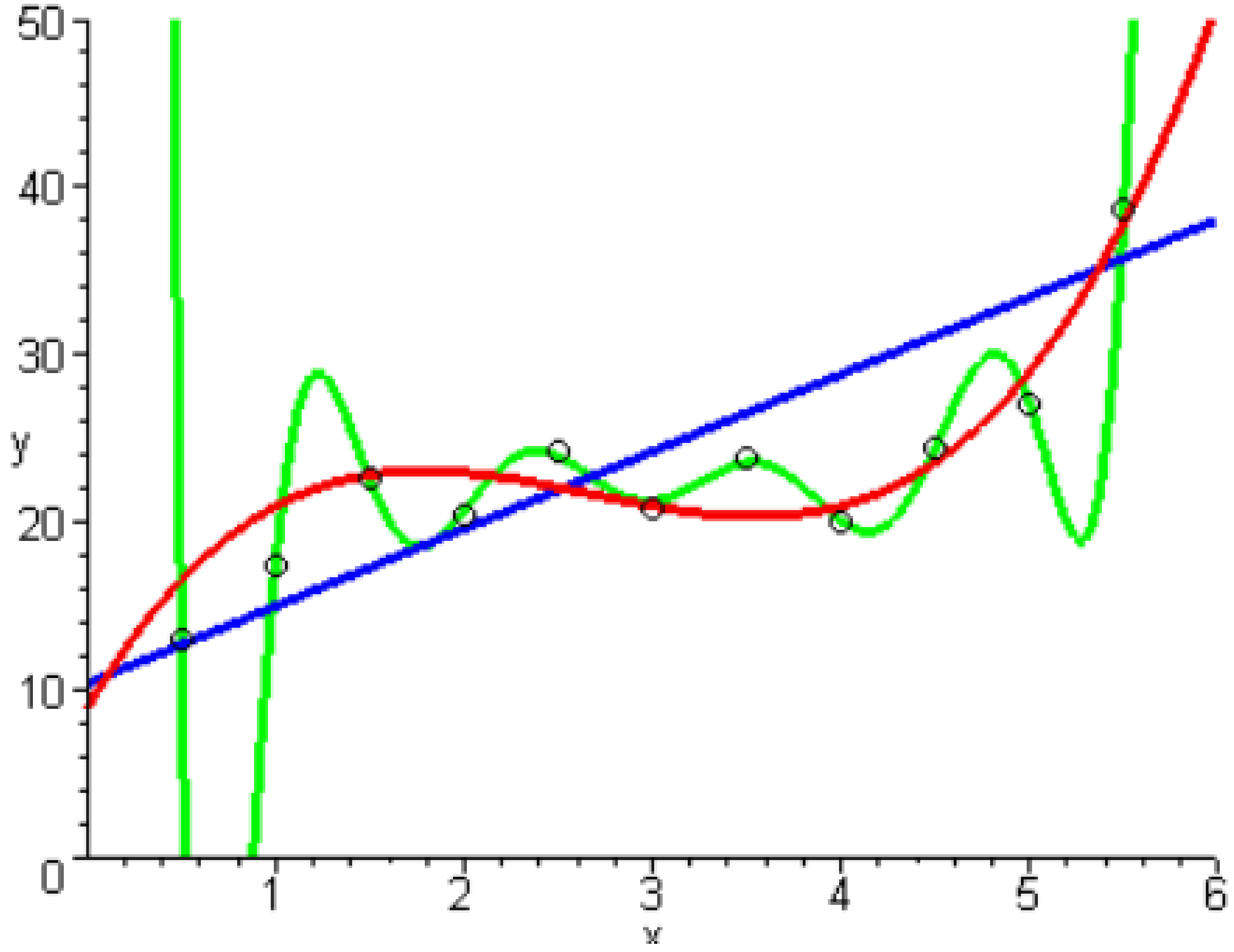
Example: Regression



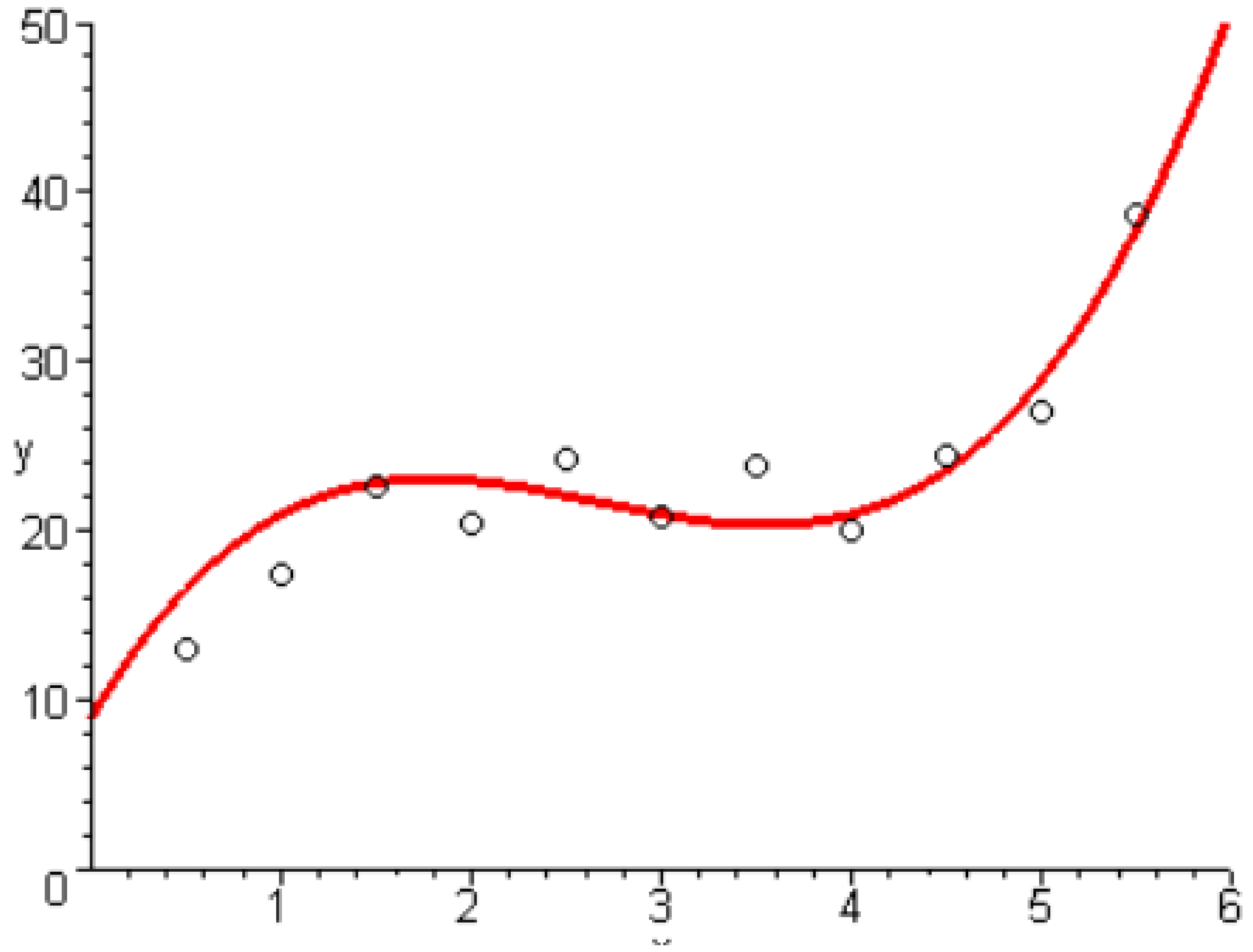
Example: Regression



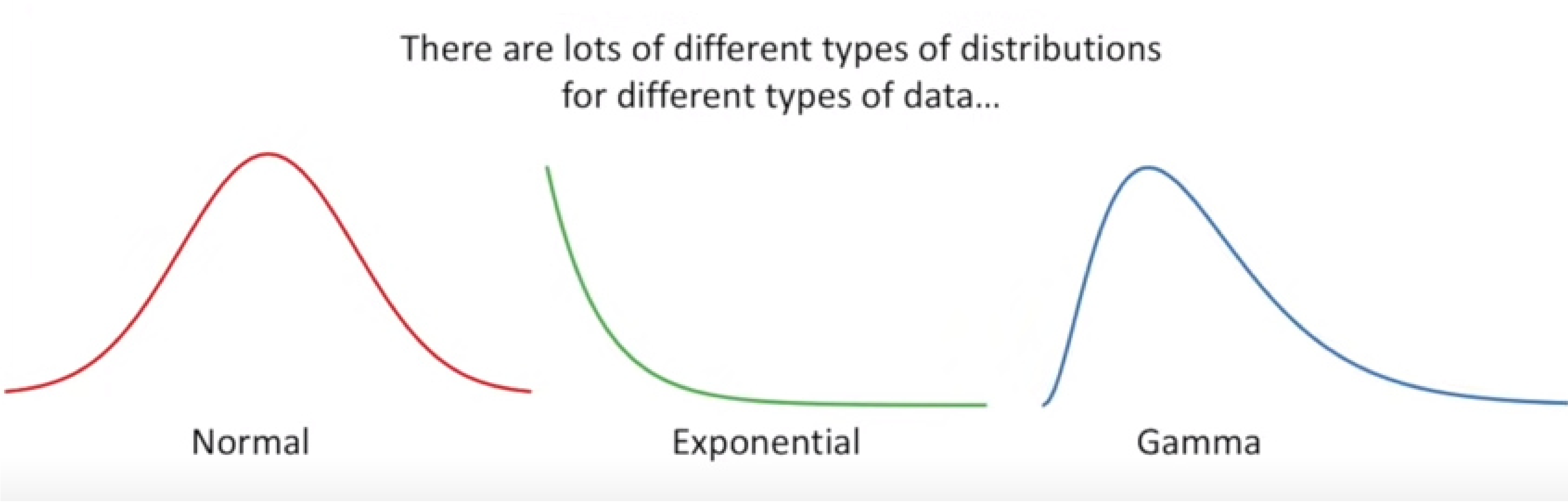
Example: Regression



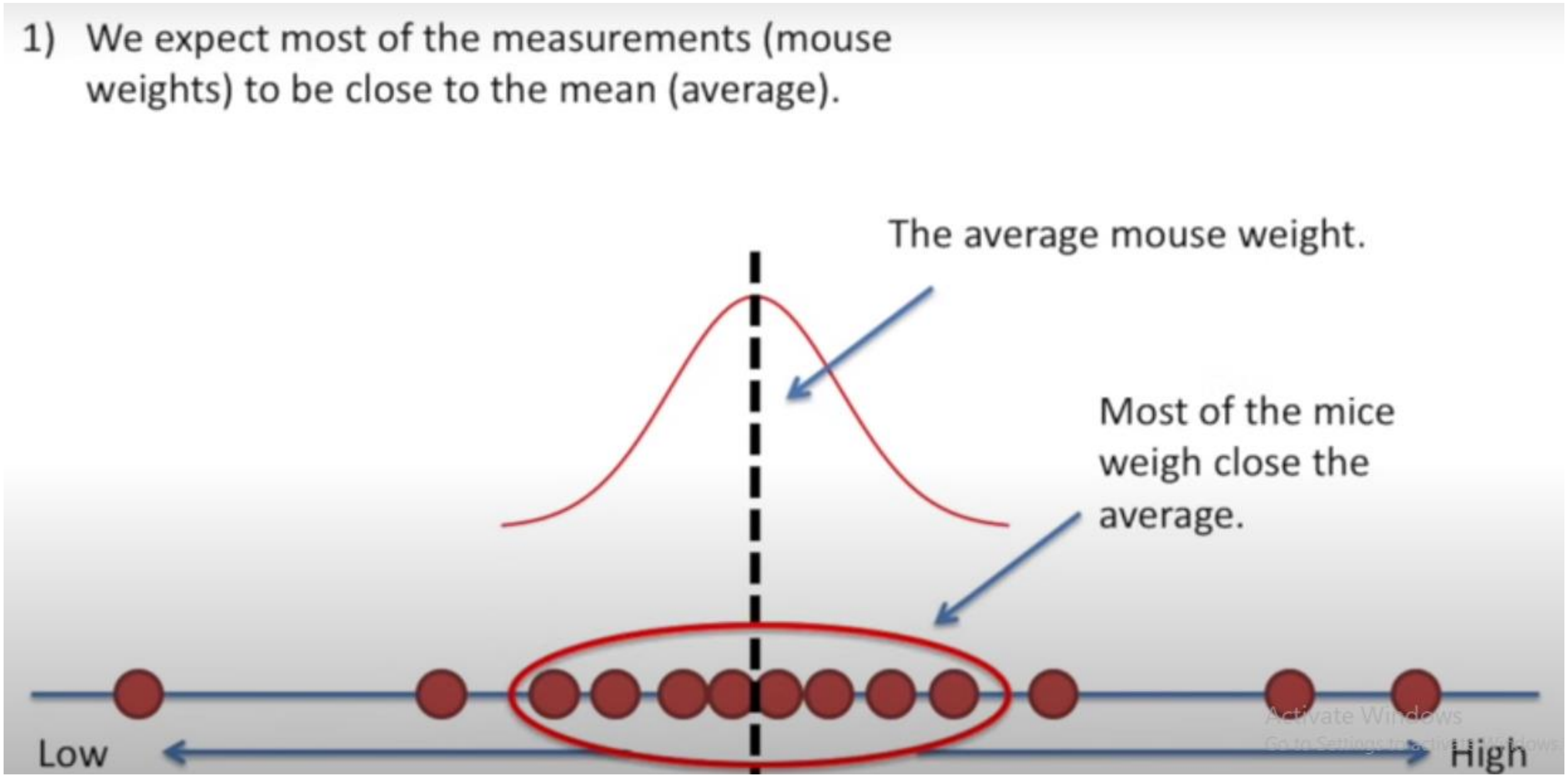
Example: Regression



Distribution


There are lots of different types of distributions
for different types of data...

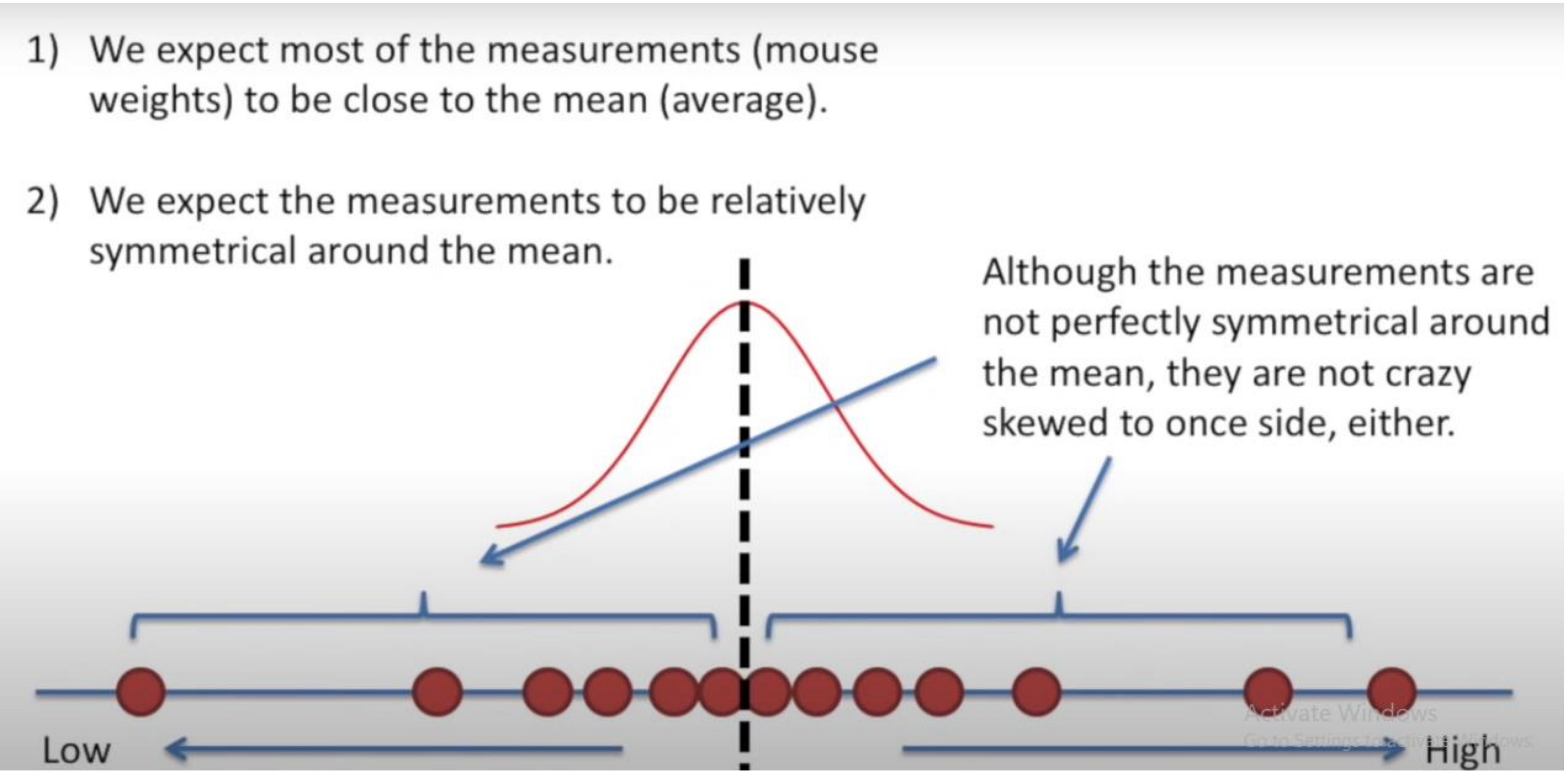
Normal Distribution



1) We expect most of the measurements (mouse weights) to be close to the mean (average).

The average mouse weight.
Most of the mice weigh close the average.
Low
High

Normal Distribution

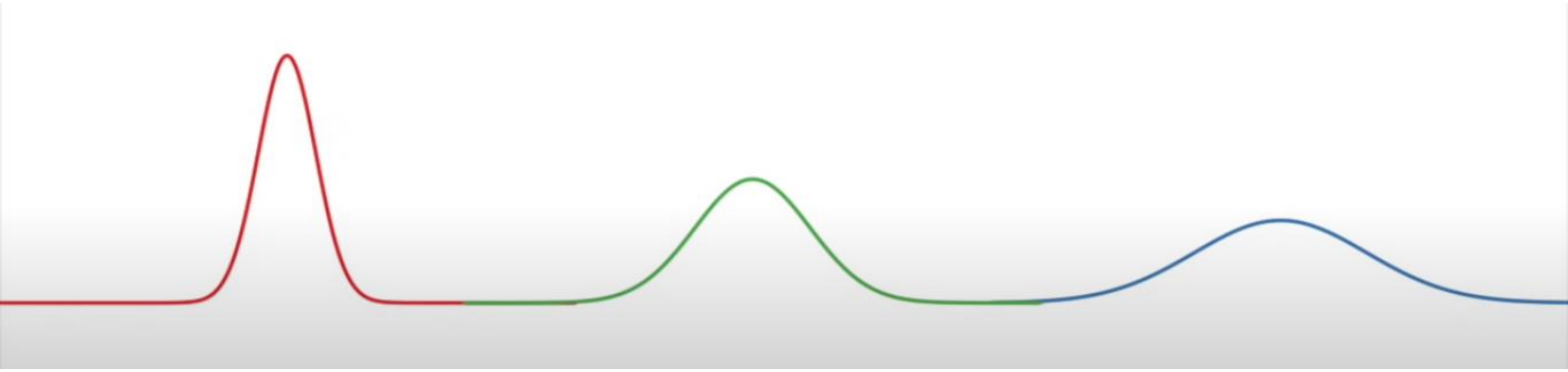


1) We expect most of the measurements (mouse weights) to be close to the mean (average).

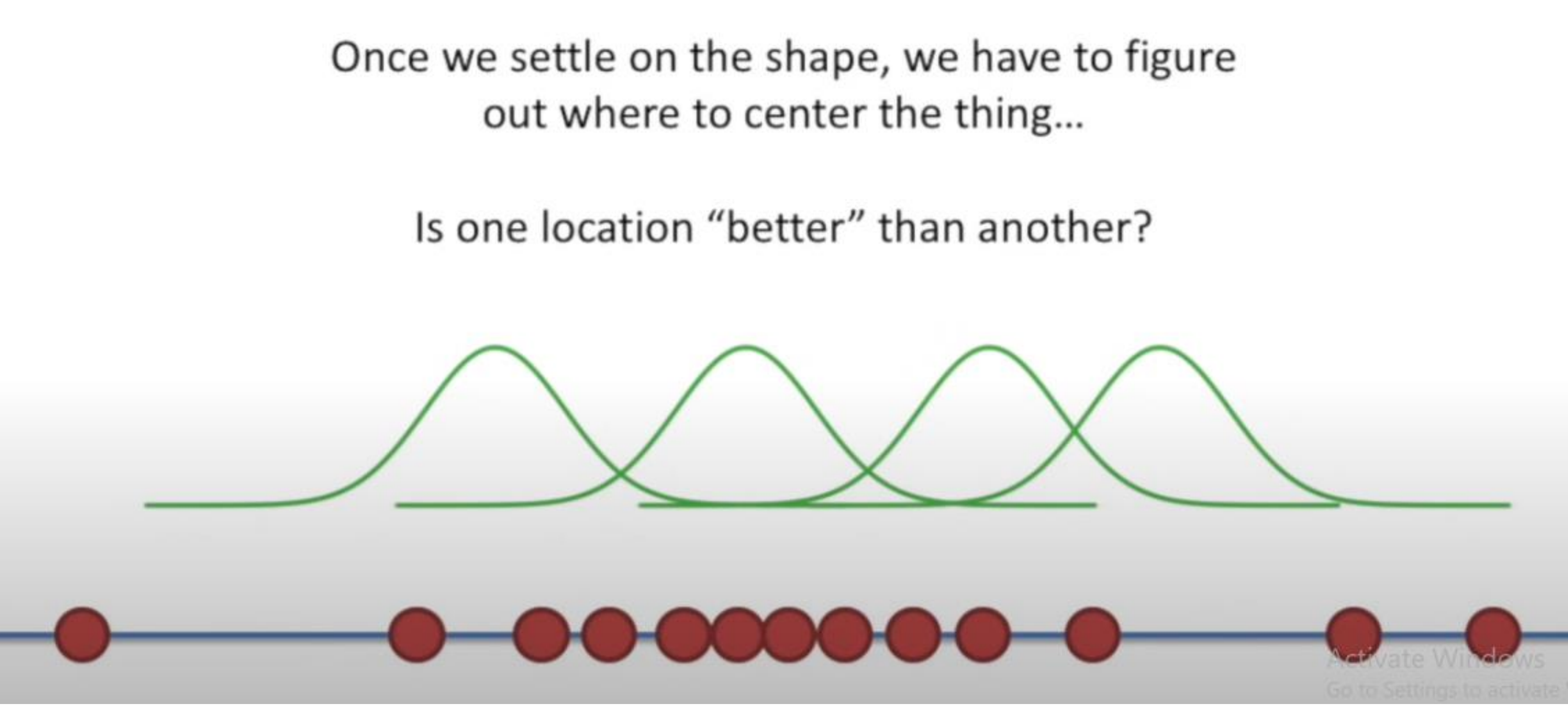
2) We expect the measurements to be relatively symmetrical around the mean.

Although the measurements are not perfectly symmetrical around the mean, they are not crazy skewed to once side, either.

- Normal distribution comes in all kinds of shapes and sizes
- Skinny, Medium or large boned

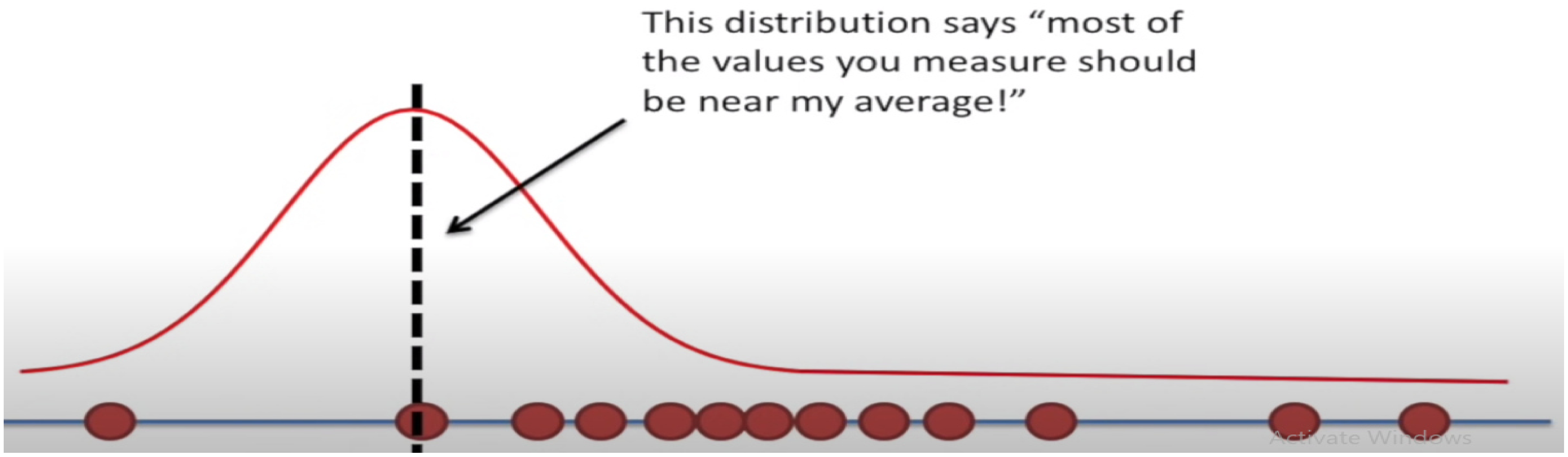


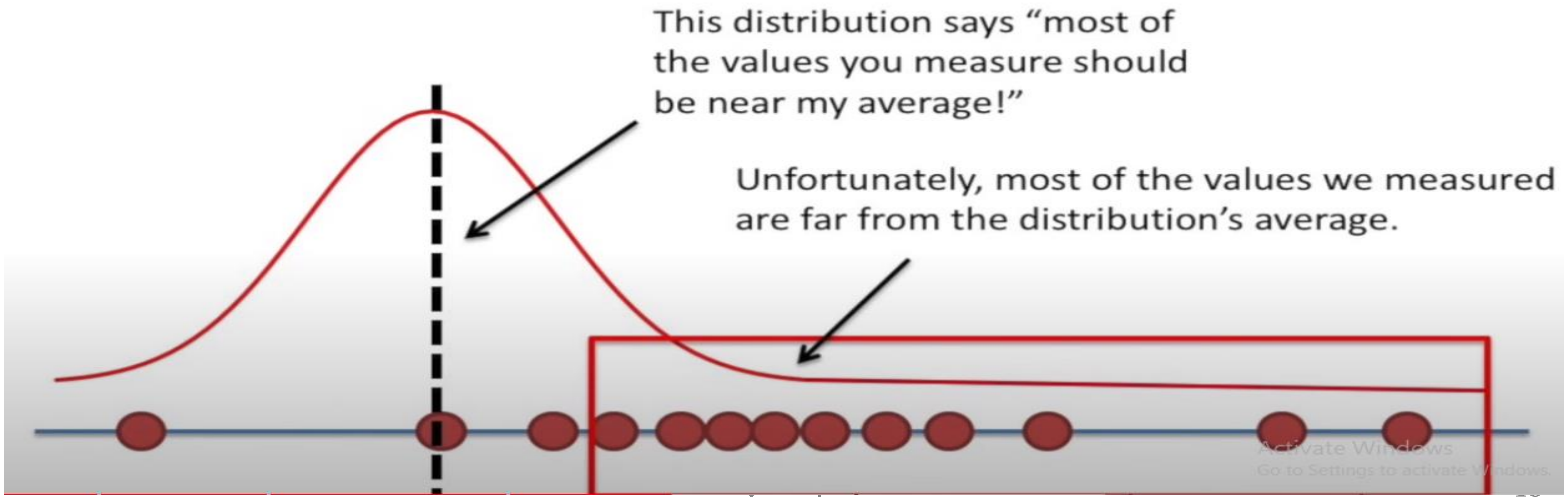
Location

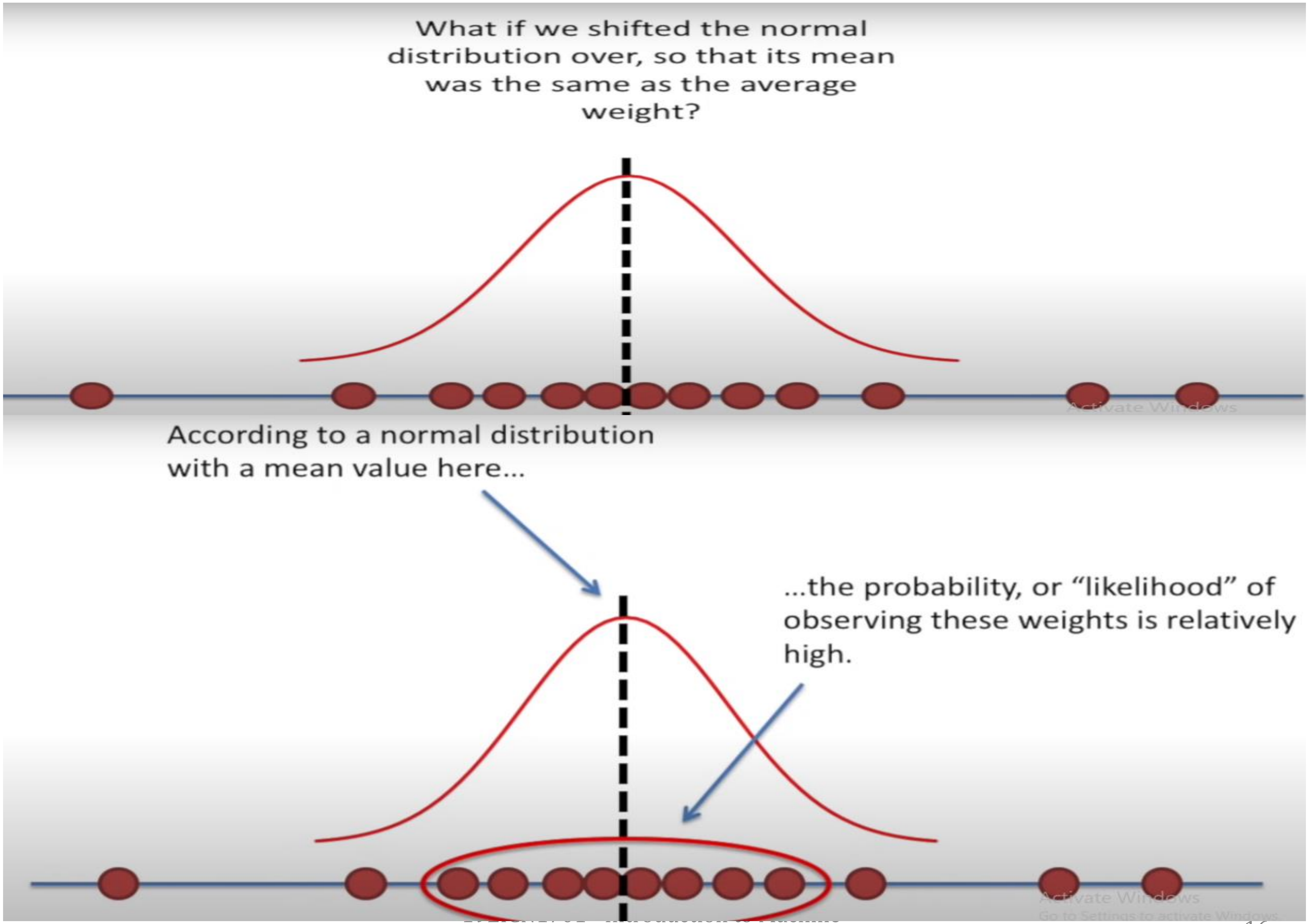


Once we settle on the shape, we have to figure out where to center the thing...

Is one location "better" than another?


This distribution says "most of the values you measure should be near my average!"


This distribution says "most of the values you measure should be near my average!"
Unfortunately, most of the values we measured are far from the distribution's average.

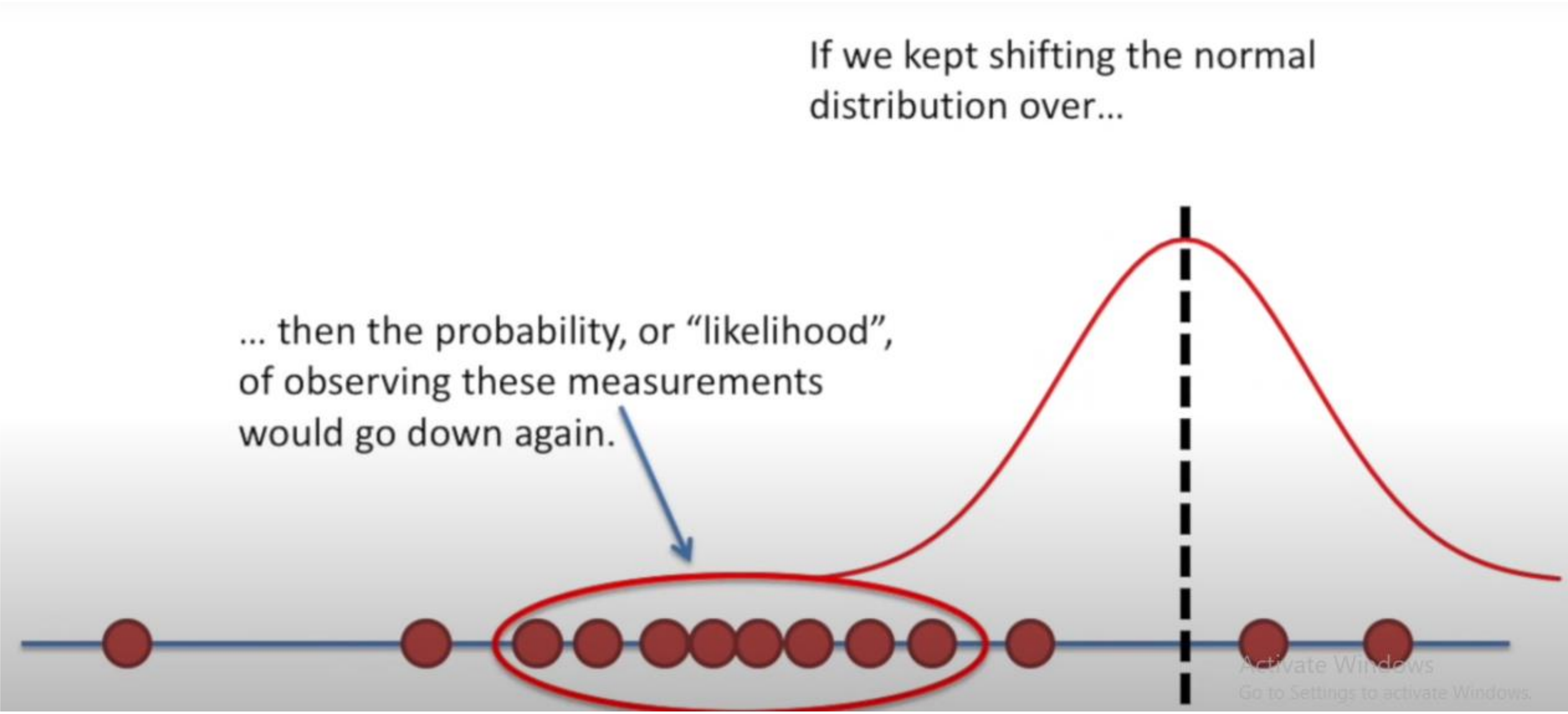


What if we shifted the normal distribution over, so that its mean was the same as the average weight?

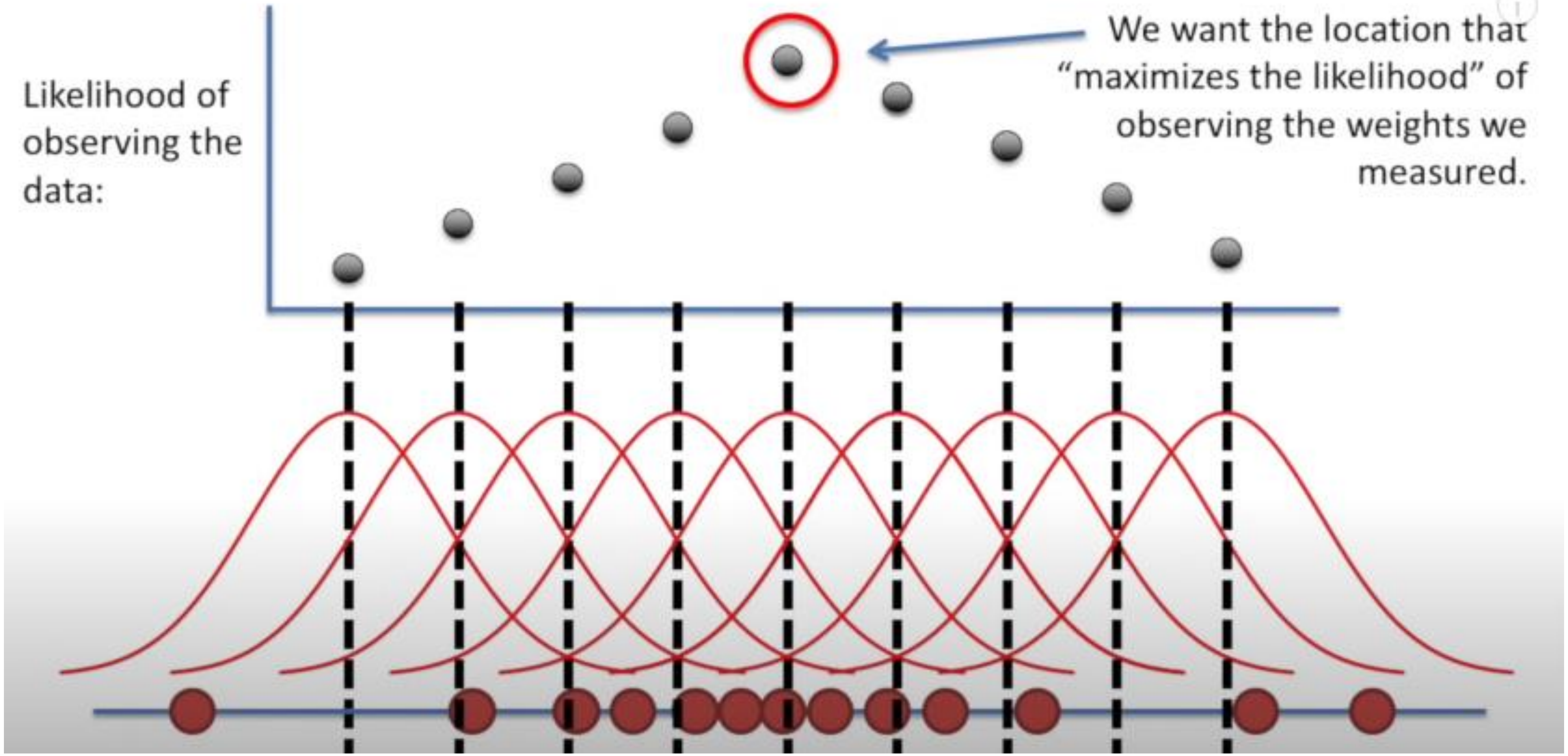
According to a normal distribution with a mean value here...

...the probability, or "likelihood" of observing these weights is relatively high.

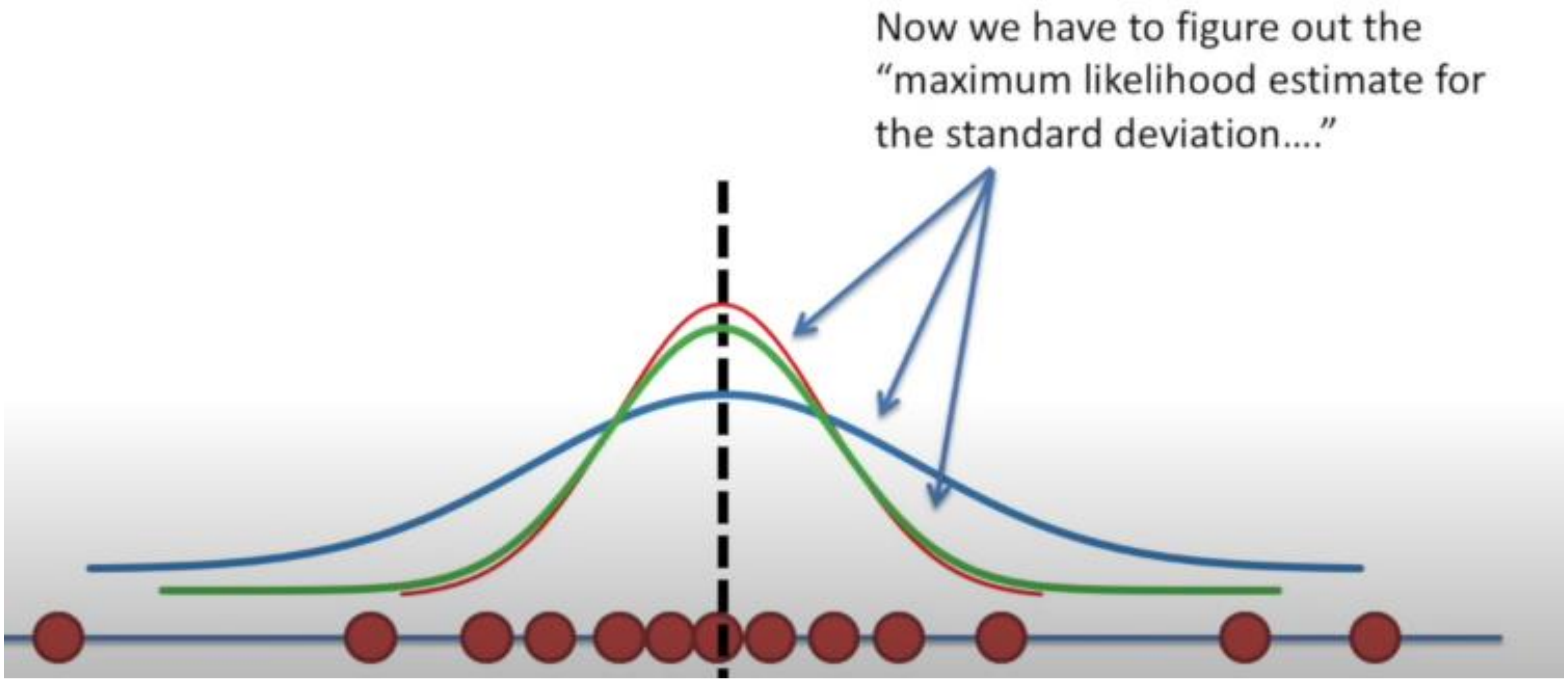
Try shifting the curve


If we kept shifting the normal distribution over...
... then the probability, or "likelihood", of observing these measurements would go down again.
Activate Windows
Go to Settings to activate Windows.

Maximum Likelihood


Likelihood of observing the data:
We want the location that "maximizes the likelihood" of observing the weights we measured.

Likelihood for SD


Now we have to figure out the "maximum likelihood estimate for the standard deviation...."

REFERENCES

- Aurélien Géron, “Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow”, Second edition, O‘Reilly Media, Inc,2019.
- https://www.youtube.com/watch?v=XepXtl9YKwc
- https://www.timvanerven.nl/assets/teaching/ml0708/slides/mlslides12-part1.pdf