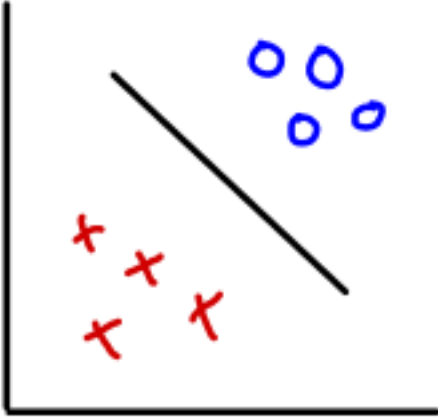


Lecture 14 [Expectation-Maximization Algorithms]

Unsupervised Learning:

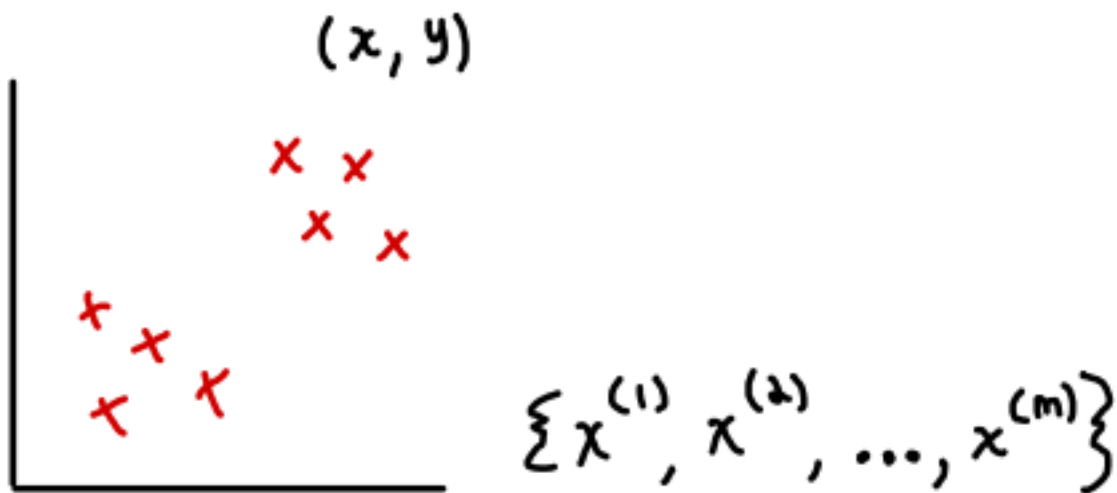
We've spent a lot of time on **supervised learning algorithms** including advice on how to apply **supervised learning algorithms** in which you'd have positive and negative examples and you run **logistic regression** or **SVM** or something to find the **decision boundary** between them



In **unsupervised learning**, you're given unlabeled data:

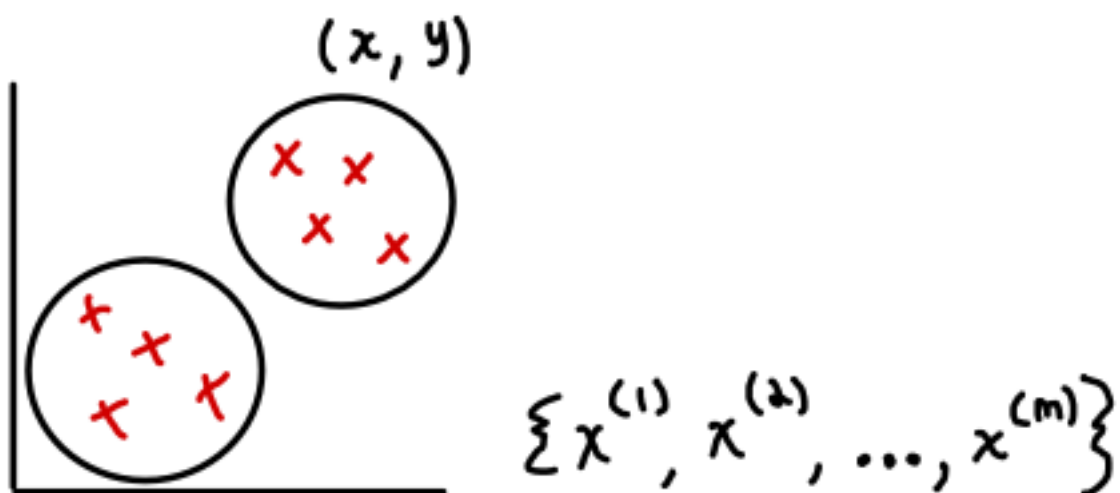


So rather than being given data with **x** and **y**, you're given only **x**, so your **training set** now looks like this:



And you're asked to find something interesting about the data

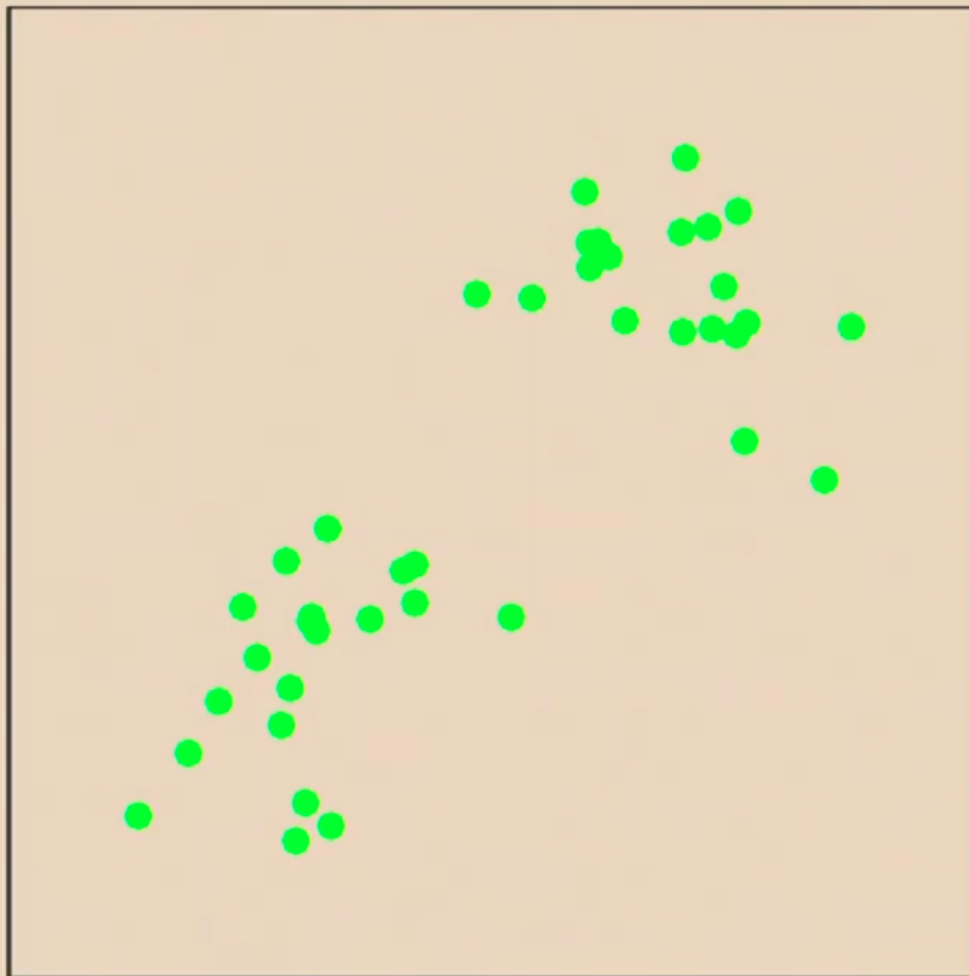
The first **unsupervised learning algorithm** we'll talk about is **clustering** in which, given a dataset like this, hopefully we can have an algorithm that can figure out that this dataset has two separate clusters:



One of the most common uses of clustering is **market segmentation**. If you have a website, selling things online, you have a huge database of many different users, and run clustering to decide what are the different **market segments**

So there may be people of a certain age range of a certain gender, people of a different age range, different level of education, people that live in the east coast vs the west coast vs other parts of the country, etc, but by clustering, you can group people into different groups

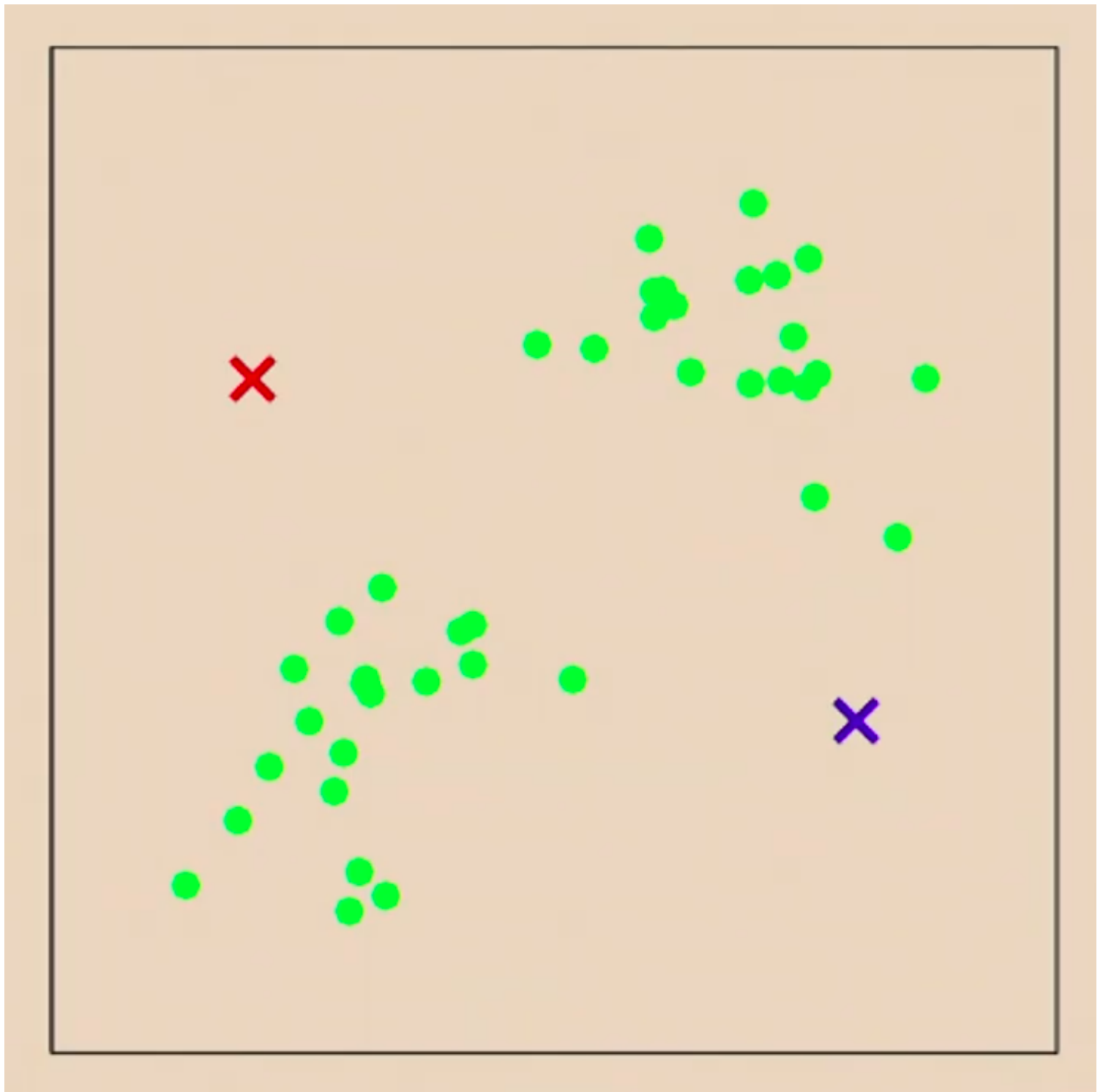
We'll go through an animation of what the most commonly used **clustering algorithm**, called **K-means clustering**, does. Let's say you're given a dataset like this:



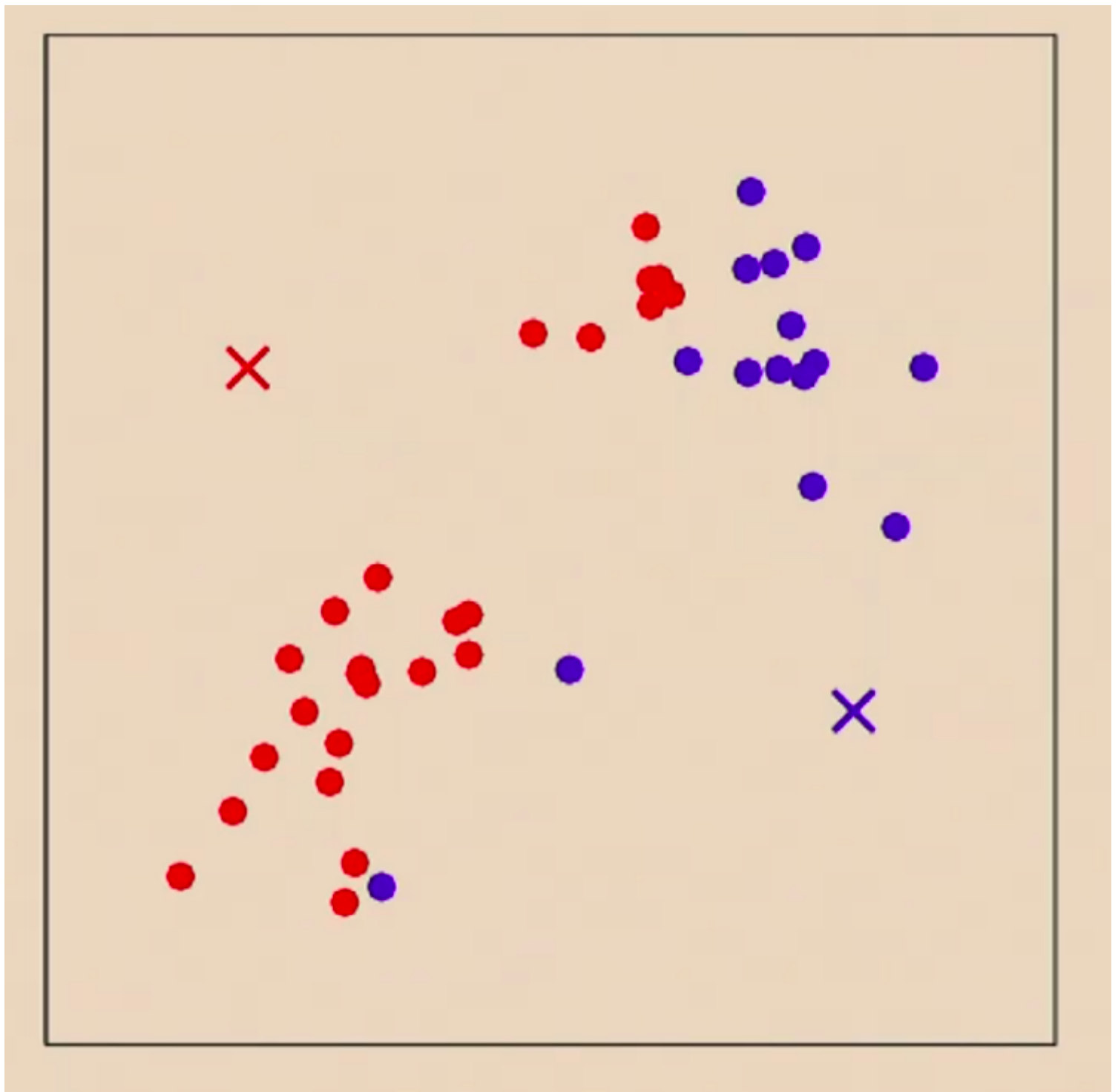
[Source: Michael Jordan]

All these are unlabeled examples, and we want an algorithm to try to find the two clusters here

The first step of ***k-means*** is to pick two points, denoted by the two crosses (called ***cluster centroids***), and the ***cluster centroids*** are your best guess for where are the centers of the two clusters you're trying to find

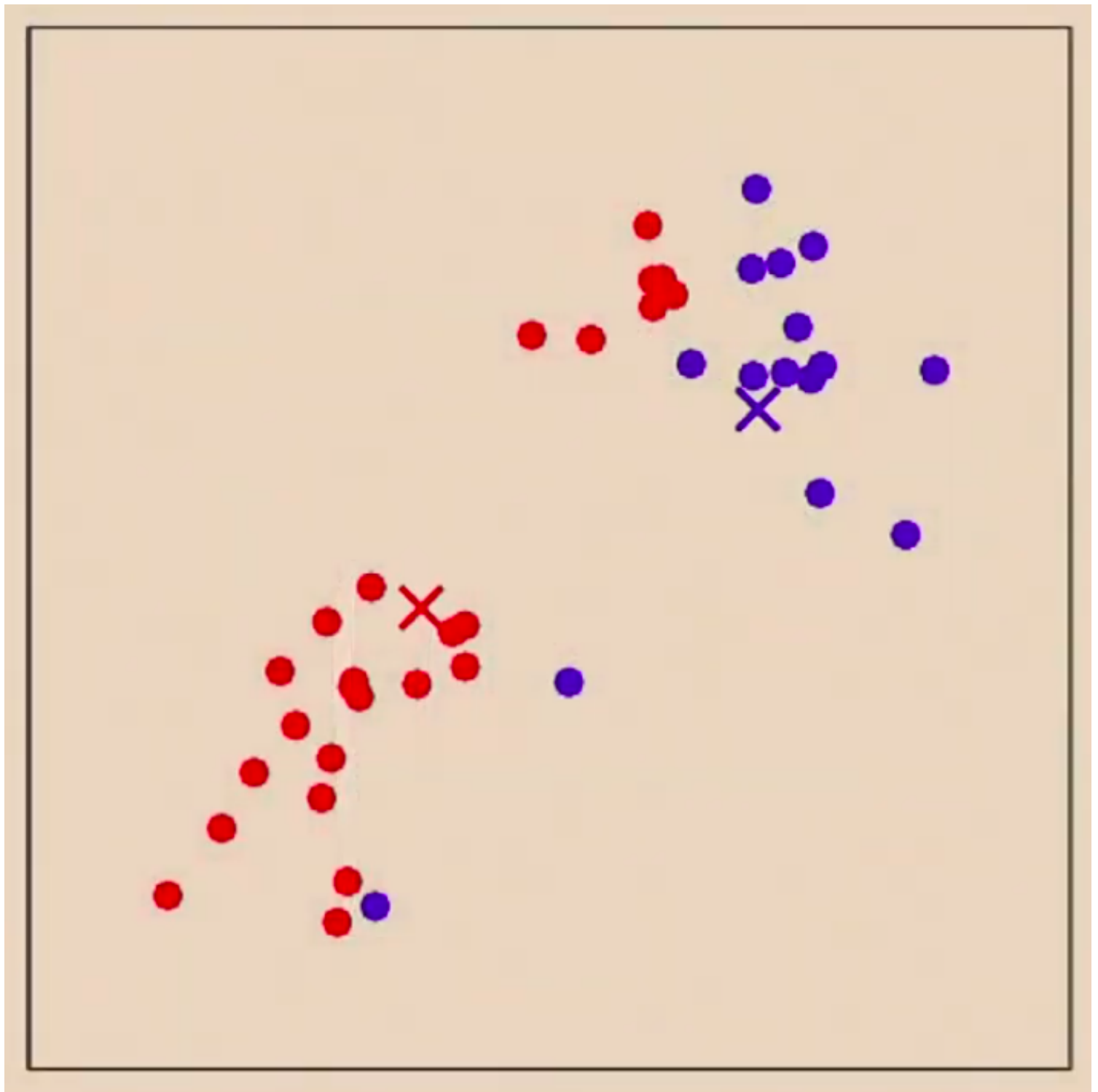


K-means is an iterative algorithm and you repeatedly do two things. The first thing is you go through each of your training examples (the **green** dots) and for each of them, you colour them either **red** or **blue** depending on which is the closer **cluster centroid**



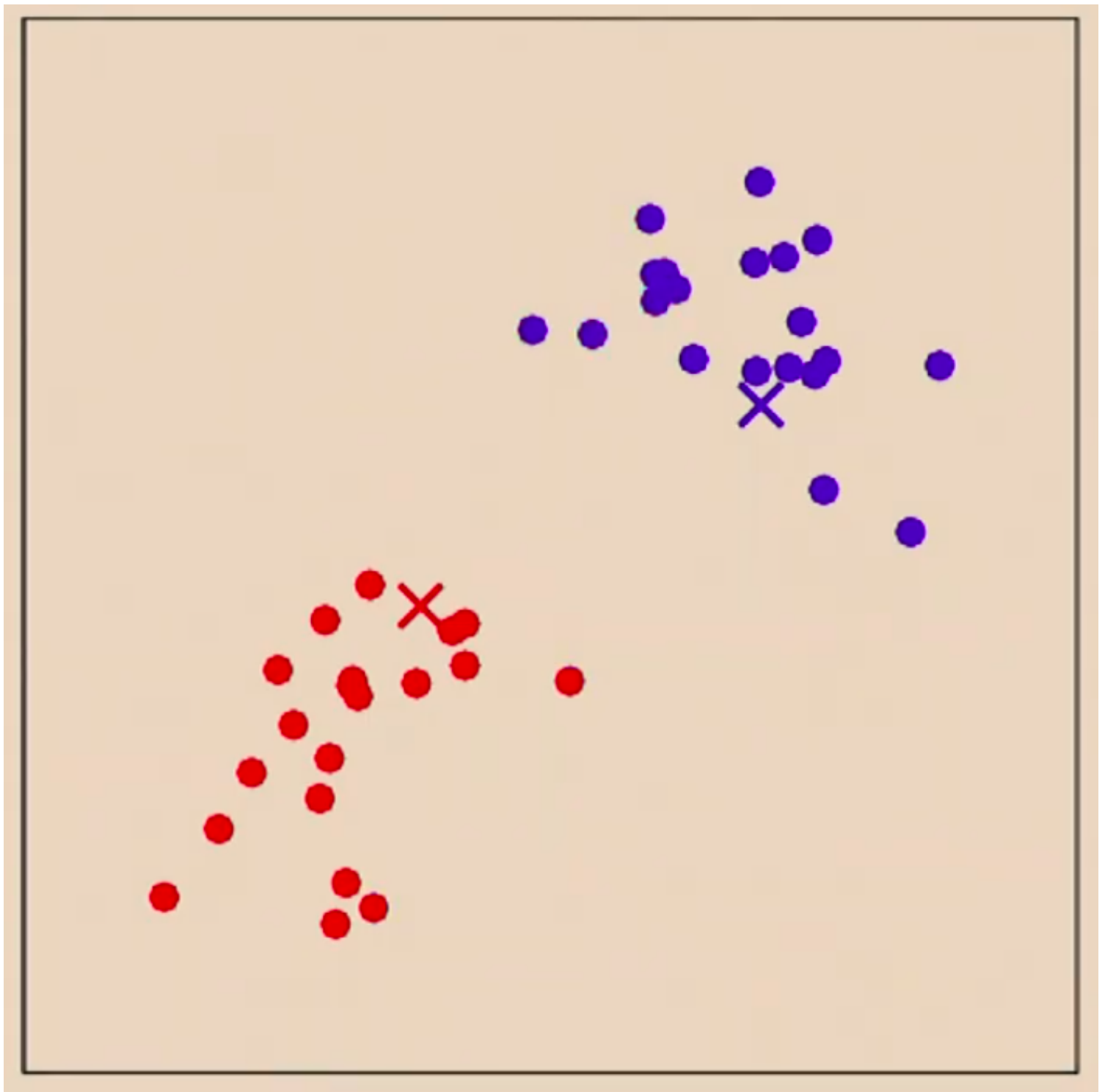
Here we've taken every dot and coloured it in **red** or **blue** depending on which **cluster centroid** it's closer to

The second thing you do is to look at all the **blue** dots and compute the average. Just find the mean of all the **blue** dots and move the **blue cluster centroid** there. Similarly, look at all the **red** dots, looking only at the **red** dots, and find the mean of all the **red** dots and move your **red cluster centroid** there

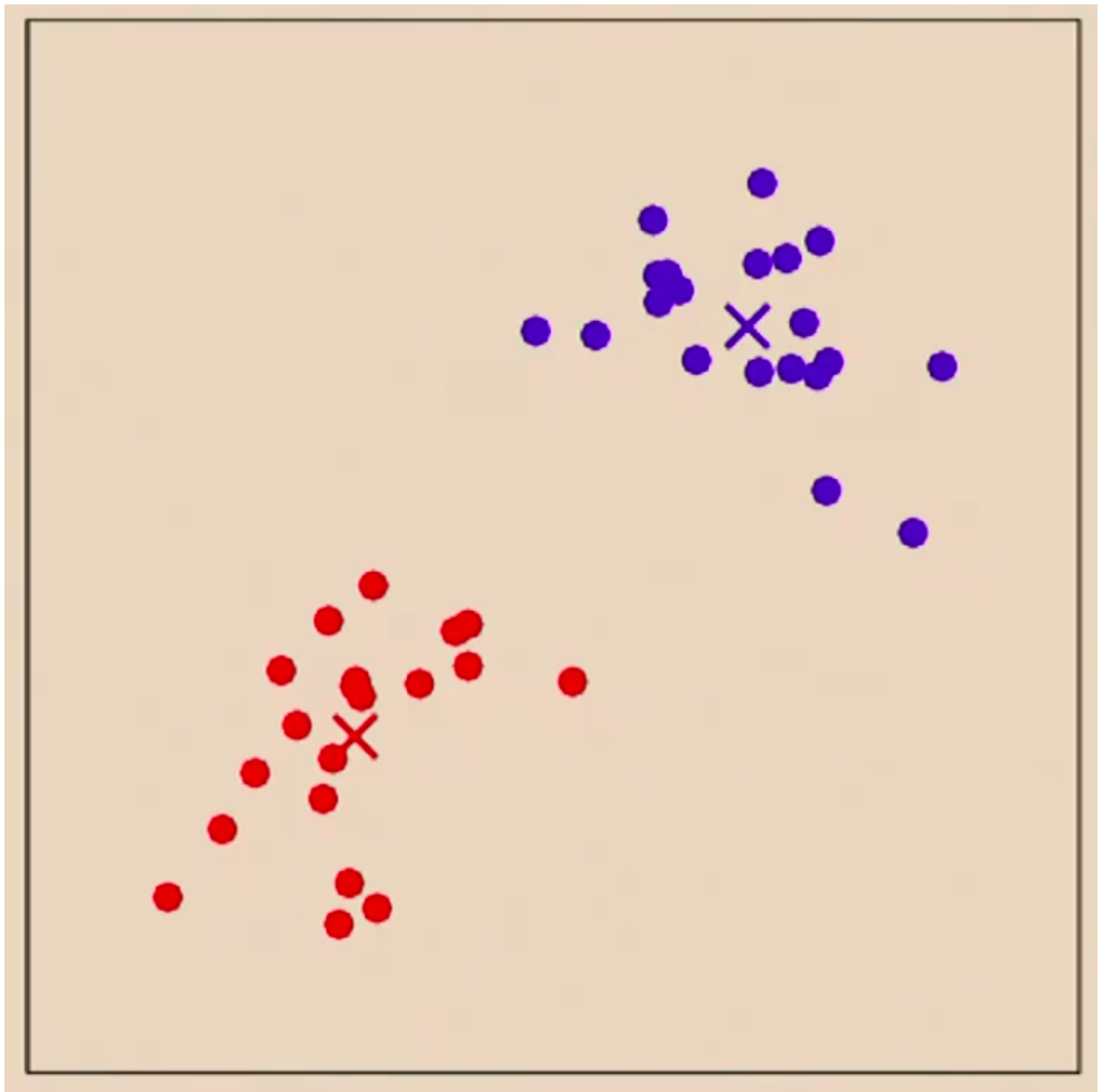


This is just a standard arithmetic average

Then you repeat again where you look at each of the dots and colour it either **red** or **blue** depending on which **cluster centroid** is closer



Then the second part of the algorithm was to look at the **blue** dots and find the mean, look at the **red** dots and find the mean, and then move the **cluster centroids** over to that mean



If you keep running the algorithm, nothing changes. So the algorithm has converged. If you look at this picture and you repeatedly colour each point **red** or **blue** depending on which **cluster centroid** is closer, nothing changes. And if you repeatedly look at each of the two clusters of coloured dots and compute the mean and move the clusters there, nothing changes. So this algorithm has converged even if you keep on running these two steps

K-Means clustering:

This is a clustering algorithm and specifically this is a ***k-means clustering algorithm***

Your dataset now does not come with any labels. In ***k-means***, **step (1)** is to initialize the **cluster centroids**:

Clustering (k-means):

$$\text{Data} : \{x^{(1)}, \dots, x^{(m)}\}$$

1.) Initialize the cluster centroids

$$\mu_1, \dots, \mu_k \in \mathbb{R}^n$$

randomly

This was a step where you plop down the **red** cross and the **blue** cross

In the animation, we did it as if we were just choosing these as random vectors. In practice, a good way (the most common way to select a random initial **cluster centroid** isn't quite what we saw) is to actually pick **k** examples out of your **training set** and just set the **cluster centroids** to be equal to **k** randomly chosen examples

In a low dimensional space like a **2-dimensional** plot that you can do on a diagram, it doesn't really matter, but when you work with very high dimensional datasets, the more common way to initialize these is to just pick **k** training examples and set the **cluster centroids** to be at exactly the location of those examples

Next, you repeat until convergence:

Clustering (k-means):

Data: $\{x^{(1)}, \dots, x^{(m)}\}$

1.) Initialize the cluster centroids

$$\mu_1, \dots, \mu_k \in \mathbb{R}^n$$

randomly

2.) Repeat until convergence:

(a) Set $c^{(i)} := \arg \min_j \|x^{(i)} - \mu_j\|_2$ ("color the points")

"L2 norm"

(b) For $j=1, \dots, k$,

("moves the cluster centroids")

$$\mu_j := \frac{\sum_{i=1}^m 1_{\{c^{(i)}=j\}} x^{(i)}}{\sum_{i=1}^m 1_{\{c^{(i)}=j\}}}$$

The two steps you would alternate between; the first one (**step (a)**) is to set $c^{(i)}$, for every value of i (for every example), equal to either **1** or **2** depending on whether that example $x^{(i)}$ is closer to **cluster centroid one** or **cluster centroid two**. So just take each point and colour it either **red** or **blue** and we represent that by setting $c^{(i)}$ equal to **1** or **2**, if you have two clusters (if $k = 2$)

Usually, it turns out that whether you use **L2 norm** or **L2 norm squared**, they give you the same answer because the algorithm is the same either way

By default, $\|x\|$ is the **L2 norm** of x if it's unspecified

Then for **step (b)**, take all the examples assigned to a certain cluster (cluster j) and set μ_j to be the average of all the points assigned to that cluster j

It turns out that this algorithm can be proven to converge. If you write this as a **cost function**:

$$J(c, \mu) = \sum_{i=1}^m \|x^{(i)} - \mu_{c^{(i)}}\|^2$$

The cost function for a certain set of assignments of points of examples to **cluster centroids** and for a certain set of positions of the **cluster centroids**:

"assignments" "centroids"

$$J(c, \mu) = \sum_{i=1}^n \|x^{(i)} - \mu_{c^{(i)}}\|^2$$

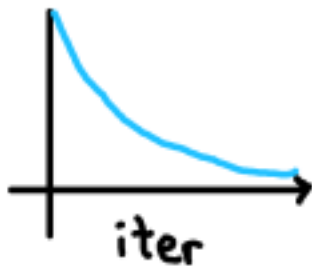
This cost here is the sum over your **training set** of what's the squared distance between each point and the **cluster centroid** that this is assigned to

On every iteration, **k-means** would drive this **cost function** down. Beyond a certain point, this **cost function** can't go any lower. It can't go below **0**, and so this shows that **k-means** must converge, or at least this function must converge because it's a strictly non-negative function that's going down on every iteration, so at some point it has to stop going down and then you could declare **k-means** to have converged

In practice, if you run **k-means** in a very very large dataset, then as you plot the number of iterations, **J** may go down:

"assignments" "centroids"

$$J(c, \mu) = \sum_{i=1}^n \|x^{(i)} - \mu_{c^{(i)}}\|^2$$



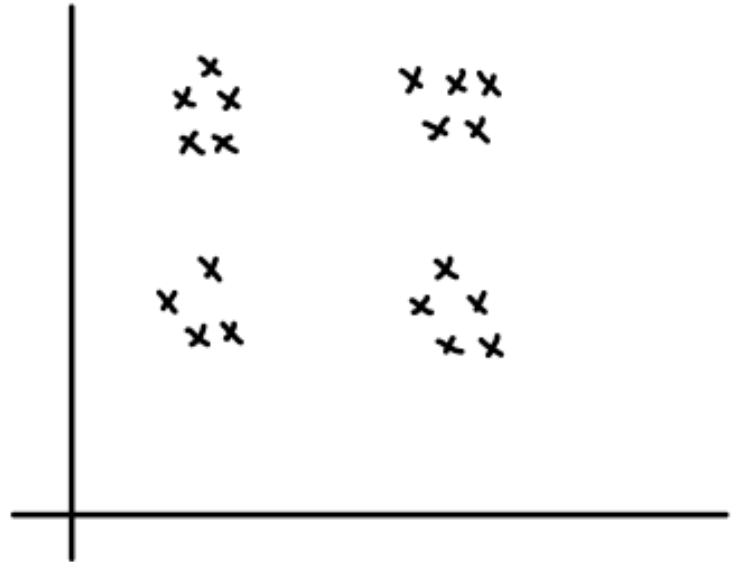
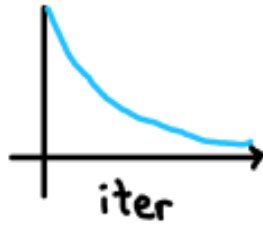
And just because of a lack of compute or lack of patience, you might just stop this running after a while if it's going down too slowly. That's sort of **k-means** in practice where maybe it hasn't totally converged so we just cut it off and call it good enough

How do you choose **k**?

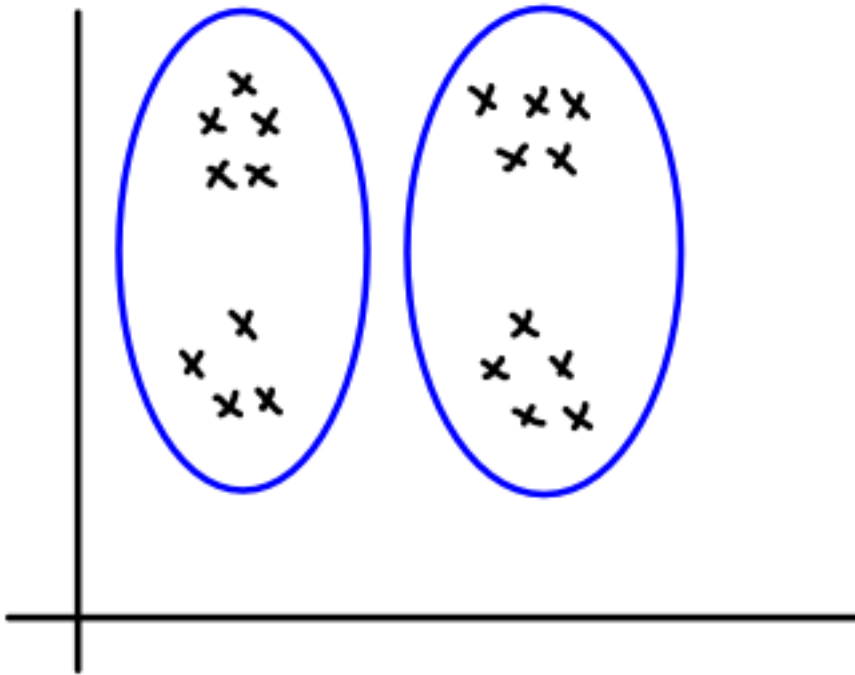
When we use **k-means**, we can still choose **k** by hand. In **unsupervised learning**, sometimes it's just ambiguous how many clusters there are

"assignments" "centroids"

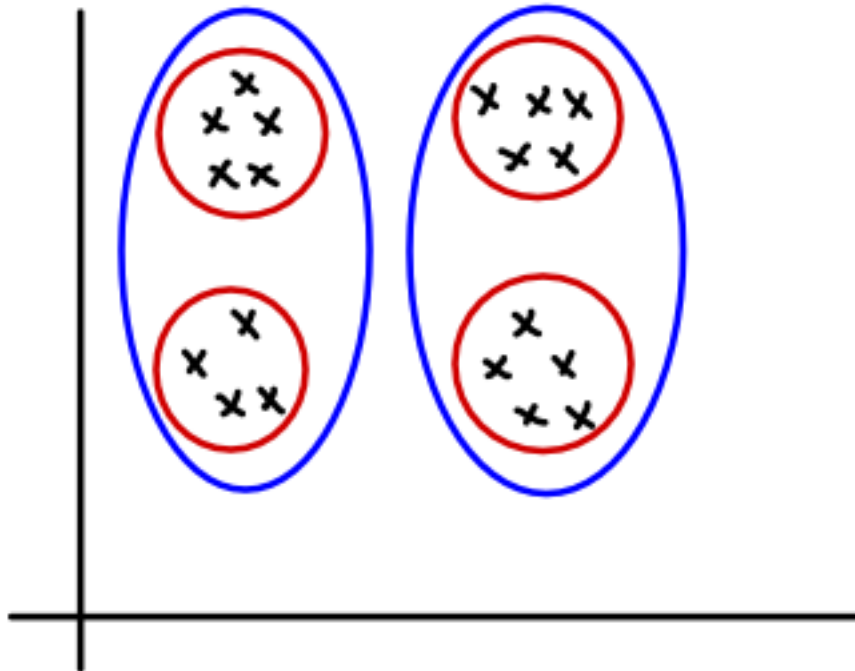
$$J(c, \mu) = \sum_{i=1}^m \|x^{(i)} - \mu_{c^{(i)}}\|^2$$



With this dataset, you may see two clusters:



You may also see four clusters:



And it's just inherently ambiguous what is the right number of clusters

There are some formulas you can find online, the criteria like **AIC** and **BIC** for automatically choosing the number of clusters. In practice, we tend not to use them because we usually look at the downstream application of what you actually want to use **k-means** for in order to make a decision on the number of clusters

For example, if you're doing a **market segmentation**, because your marketers want to design different marketing campaigns for different groups of users, then your marketers might have the bandwidth to design four separate marketing campaigns but not 100 marketing campaigns. So that would be a good reason to choose four clusters rather than 100 clusters

So if you look at the purpose of what you're doing this for, we usually pick the number of clusters, either manually or by looking at what you want to use **k-means clustering** for

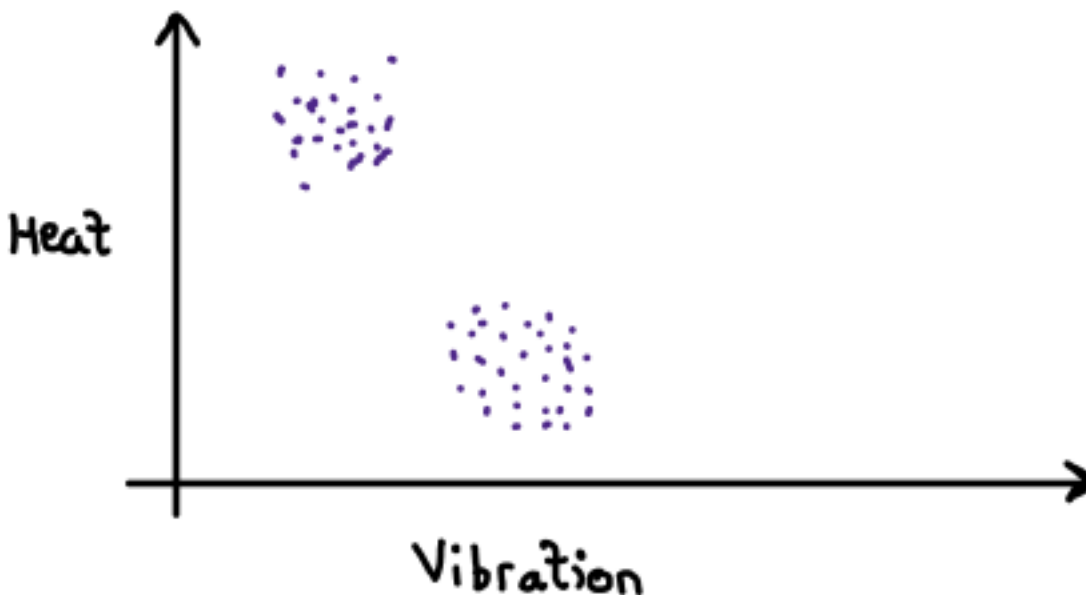
Will **k-means** get stuck on **local minima**?

Yes, **k-means** gets stuck on **local minima** sometimes. If you're worried about **local minima**, the thing you can do is run **k-means**, say 10 times or 100 times or 1,000 times, from different random initializations of the **cluster centroids**, and then run it, say 100 times, and then pick whichever run that resulted in the lowest value for this cost function **J**

There's a problem that seems closely related, but there's actually quite different ways to write the algorithms, which is **density estimation**

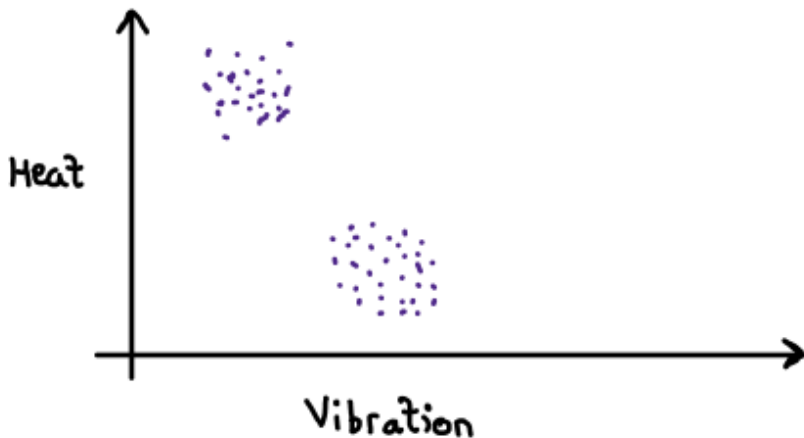
To motivate this with an example:

Density estimation:



Say, you have aircraft engines coming off the assembly line, and every time an aircraft engine comes off the assembly line, you measure some features of these engines. You measure some features about the vibration and you measure some features about the heat that the aircraft engine is producing. Let's say you get a dataset that looks like this (the above)

Density estimation:



Anomaly detection

The anomaly detection problem is if you get a new aircraft engine that comes off the assembly line, and if the vibration feature takes on this value and the heat feature takes on this value:

Density estimation:



Anomaly detection

Is that aircraft engine an anomalous one? Is it an unusual one?

The application of this is that as your aircraft engine comes off the assembly line, if you see a very unusual signature in terms of the vibrations and the heat the aircraft engine is generating, then probably something is wrong with this aircraft engine and you can have your team inspect it further or test it further before you ship the engine to an airplane maker and then have something go wrong in the air and there be a major accident or major disaster

One of the common ways to implement anomaly detection is the **model $p(\mathbf{x})$** which is, given all of these examples, can you model

what is the density from which \mathbf{x} was drawn?

So then, if $p(\mathbf{x})$ is very small (less than **epsilon** ϵ), then you flag an anomaly:

Density estimation:



Anomaly detection

Model $p(\mathbf{x})$

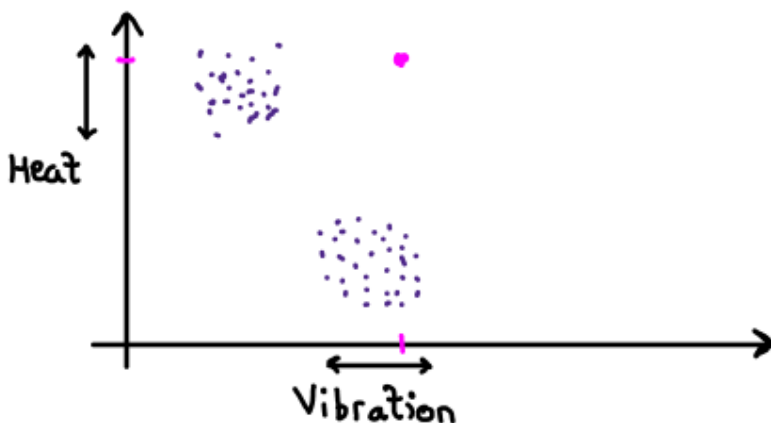
$p(\mathbf{x}) < \epsilon \rightarrow \text{anomaly}$

Anomaly detection is used for an inspection task like this. It's also used for telecommunications, computer security, etc. These are some of the applications of anomaly detection. A good way to do this is, given an unlabeled dataset, model $p(\mathbf{x})$, and then if you have very low probability samples, you flag that as a possible anomaly for further study

Given this dataset, how do you model this?

One interesting thing about this magenta dot is that neither the vibration nor the heat signature is actually out of range:

Density estimation:



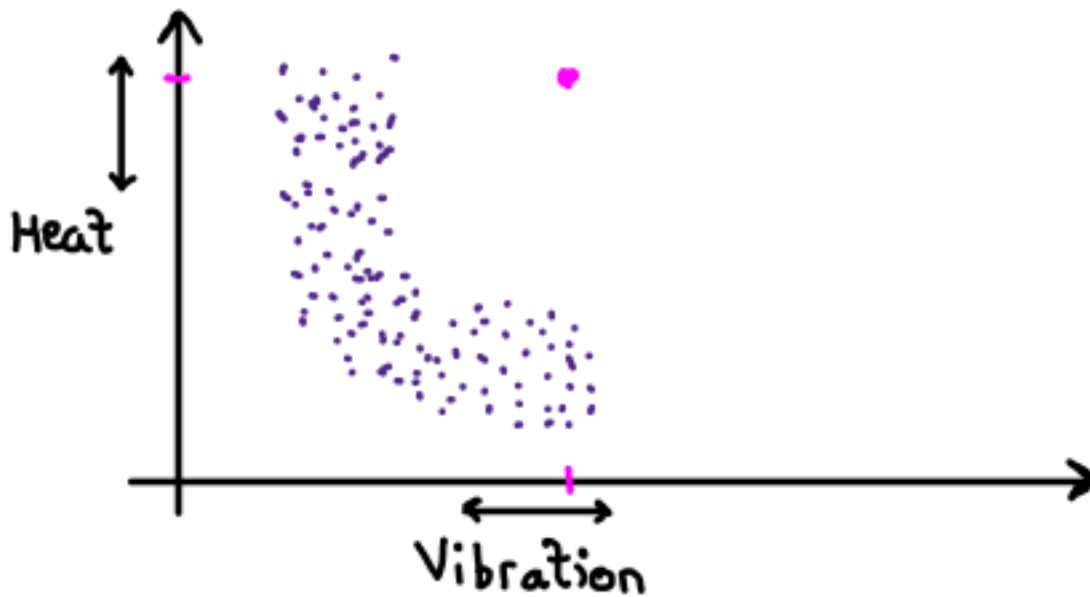
Anomaly detection

Model $p(\mathbf{x})$

$p(\mathbf{x}) < \epsilon \rightarrow \text{anomaly}$

There are a lot of aircraft engines with vibrations in that range and there are a lot of aircraft engines with heat in that range, so neither feature by itself is actually that unusual. It's actually the combination of the two that is unusual. And so, what we want to do is come up with an algorithm to model this

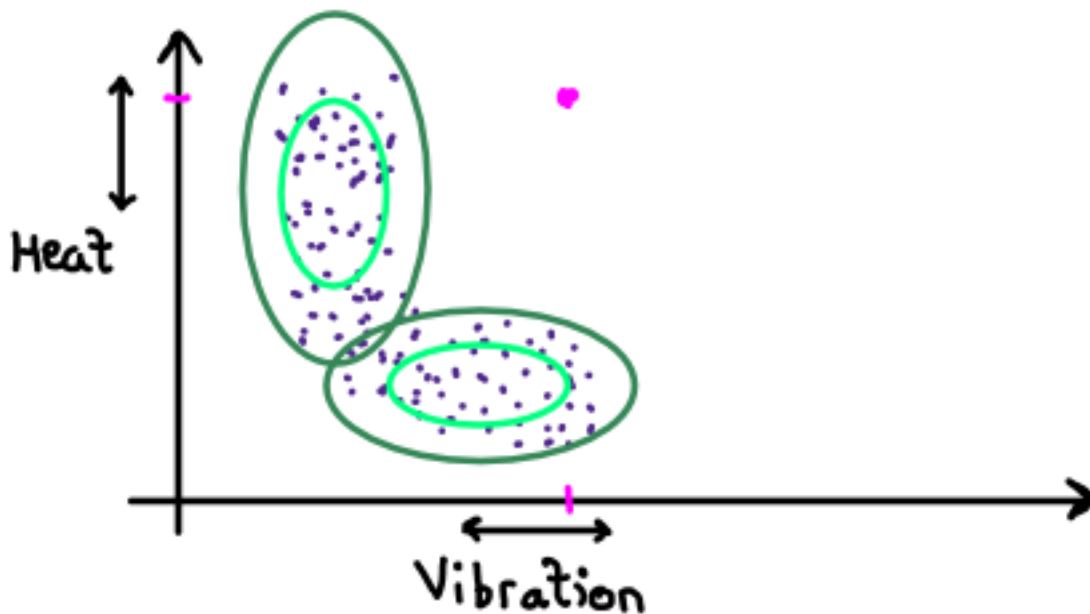
In fact, we'll come up with an algorithm that can even model a dataset with a **data density** that looks like this:



But how do you model $p(\mathbf{x})$ with the data coming from an **L shape**?

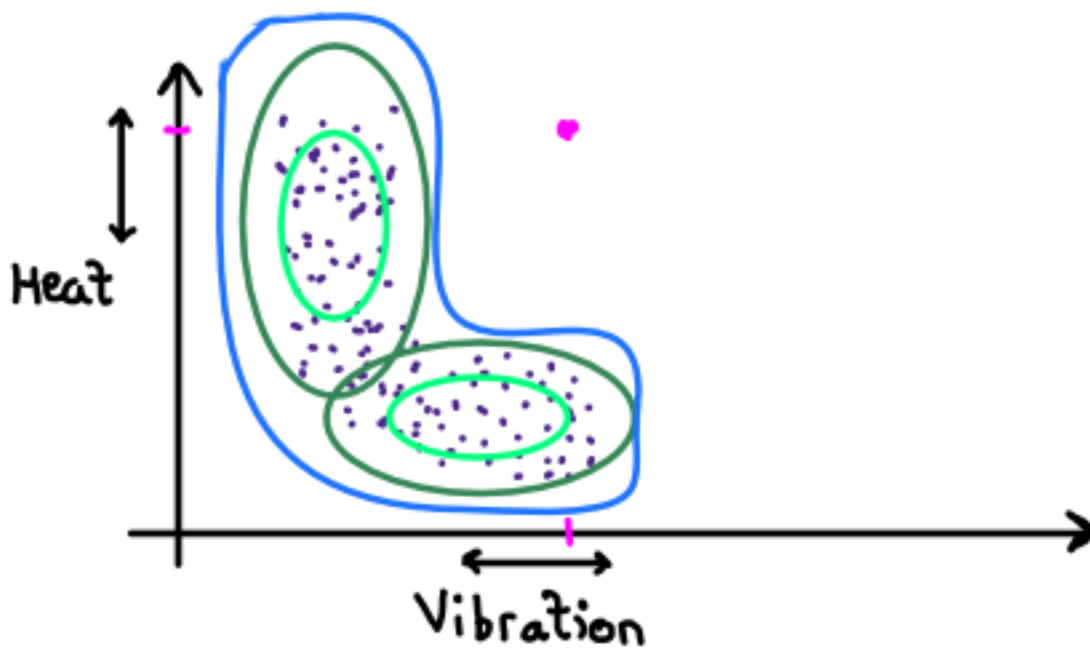
It turns out that there is no textbook distribution. If you look at a simple exponential family model types of distributions, there is no distribution for modeling very complex distributions like this

What we're going to talk about is the **mixture of Gaussians model** which, we look at data like this and say that it looks like this data actually comes from two **Gaussians**



There's one **Gaussian**, maybe there's one type of aircraft engine that is drawn from a **Gaussian** like the one below and a separate type of aircraft engine that's drawn from a **Gaussian** like that above

And this is why there's a lot of probability mass in this L-shaped region but very low probability outside that L-shaped region



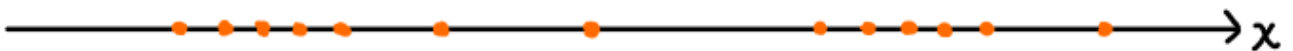
These ellipses we're drawing are the contours of these two **Gaussians**

What we'd like to do next is develop the **mixture of Gaussians model** which is useful for anomaly detection

In order to make the **mixture of Gaussians model** a bit easier to develop, we'll just use a **1-dimensional** example where \mathbf{x} is in \mathbf{R} . Let's say that we get a dataset that looks like this:

1-Dimensional example:

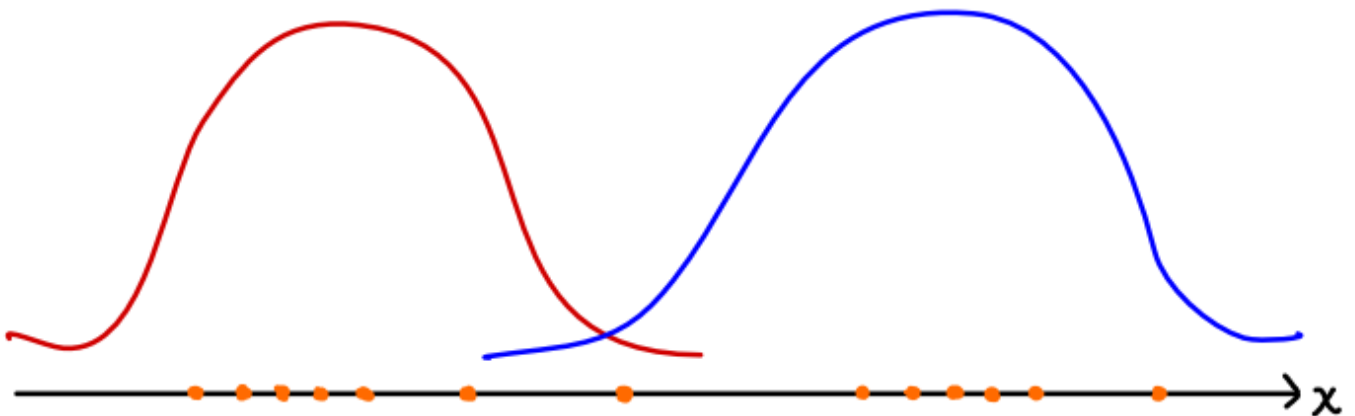
$$(x \in \mathbb{R})$$



So it looks like this data maybe comes from two **Gaussians**:

1-Dimensional example:

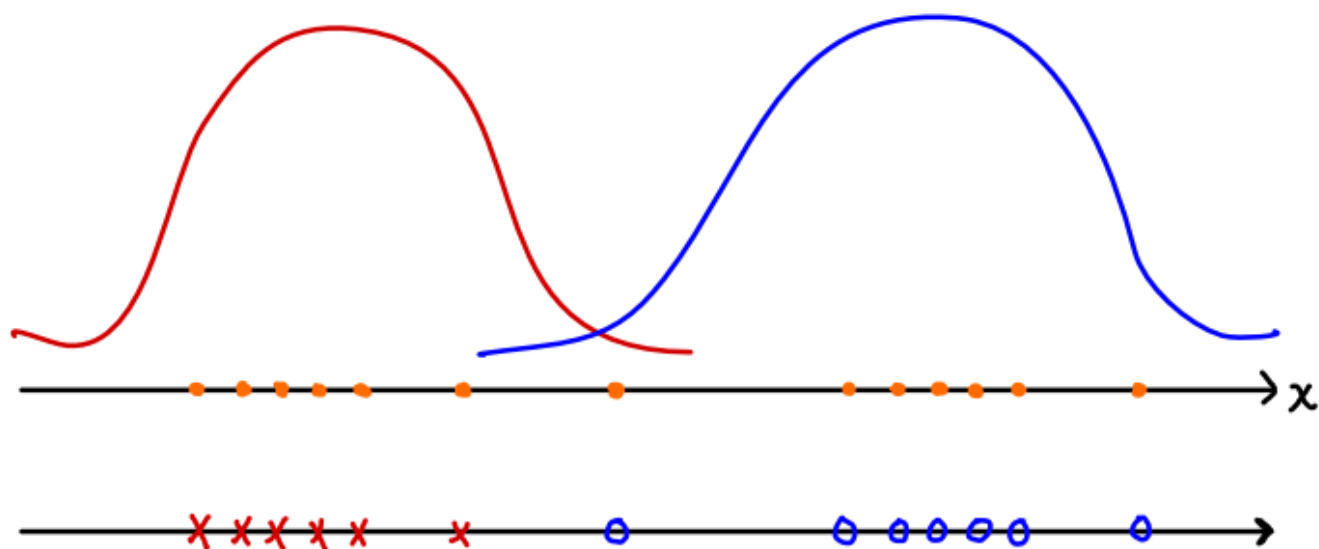
$$(x \in \mathbb{R})$$



If only we knew which of these examples had come from **Gaussian 1** (denoted by **X**'s) and which examples had come from **Gaussian 2** (denoted by **O**'s), then we'd just fit **Gaussian 1** to the **X**'s, fit **Gaussian 2** to the **O**'s, and then we'd be pretty much done:

1-Dimensional example:

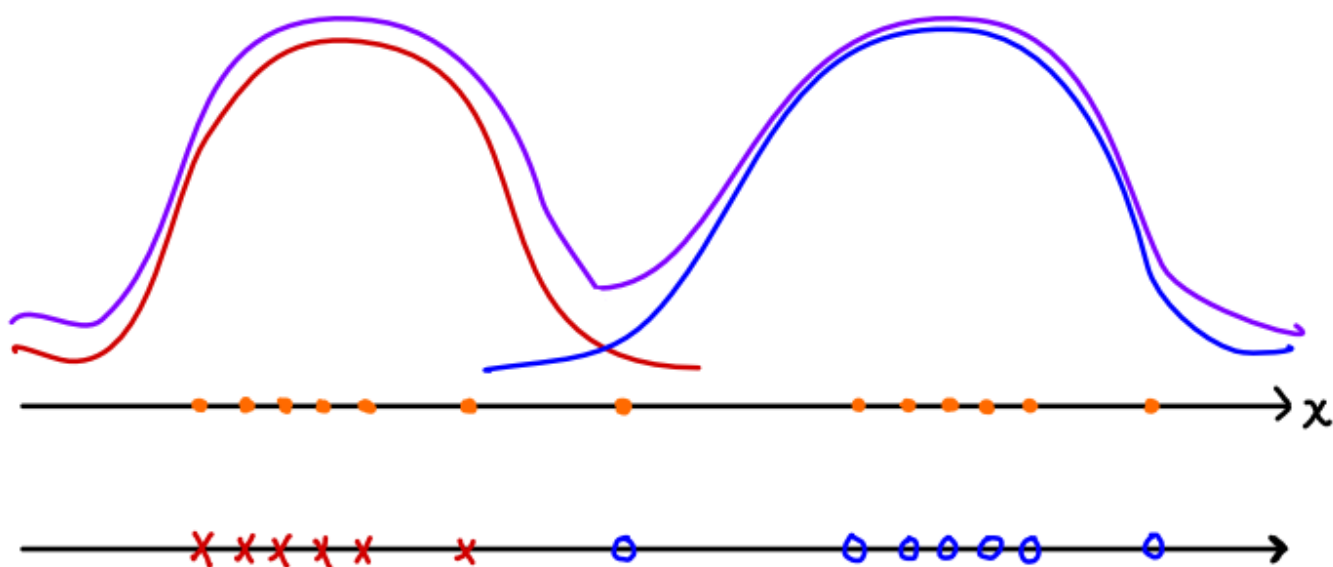
$$(x \in \mathbb{R})$$



These are the two **Gaussians** and so the overall **density** would be something like this:

1-Dimensional example:

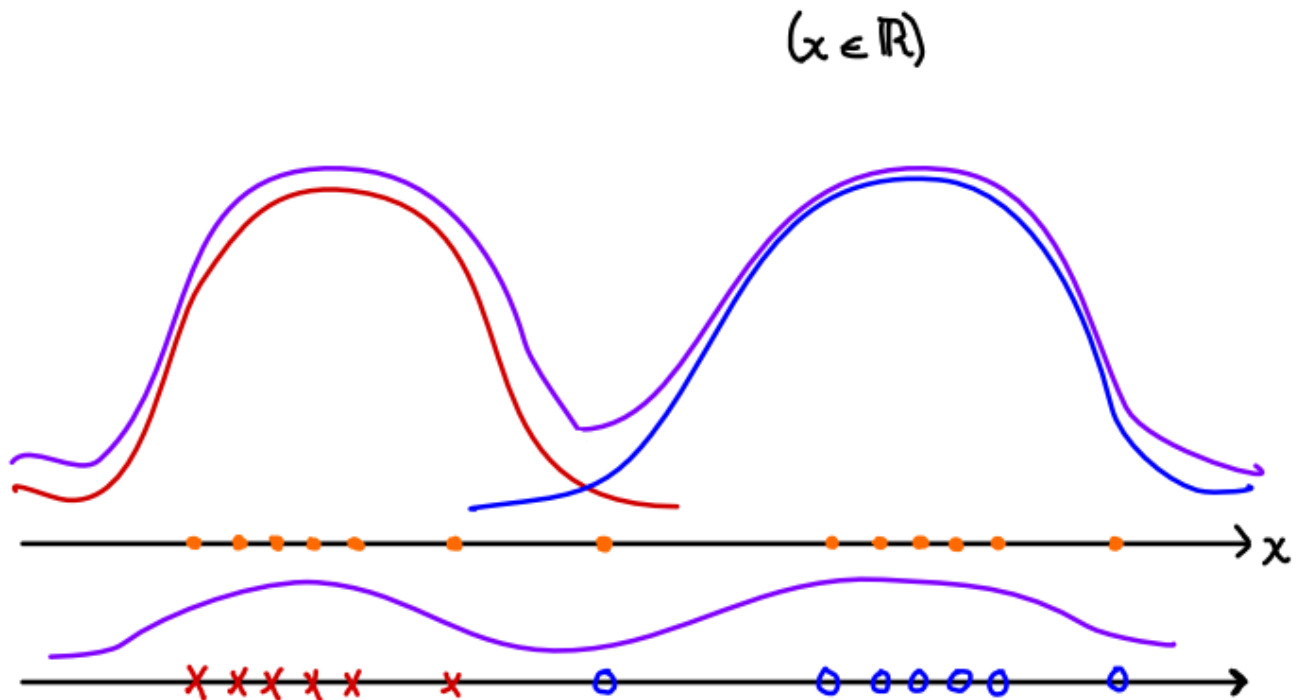
$$(x \in \mathbb{R})$$



That's the probability. A lot of probability mass on the left, a lot of probability mass on the right, and less probability mass in the middle

So the overall density, just to draw it again, would be something like this:

1-Dimensional example:



If you actually had these labels, if you knew that these examples came from **Gaussian 1** (X's) and those examples came from **Gaussian 2** (O's), then you can actually use an algorithm that's very similar to **GDA** (**Gaussian Discriminant Analysis**) to fit this model

The problem with this **density estimation problem** is, you just see this data, and maybe the data came from two different **Gaussians**, but you don't know which example actually came from which **Gaussian**

So, the **EM algorithm** (**Expectation-Maximization algorithm**) will allow us to fit a model despite not knowing which **Gaussian** each example had come from

Mixture of Gaussian:

We'll first write down the **mixture of Gaussians model** and then we'll describe the **EM algorithm** for this

Mixture of Gaussian model:

Suppose there's a latent (hidden/unobserved) random variable z , and $x^{(i)}, z^{(i)}$ are

distributed $P(x^{(i)}, z^{(i)}) = P(x^{(i)} | z^{(i)}) P(z^{(i)})$

where $z^{(i)} \overset{\text{"is"}}{\sim} \text{Multinomial}(\phi)$ $z \in \{1, \dots, k\}$

$x^{(i)} | z^{(i)} = j \sim \mathcal{N}(\mu_j, \Sigma_j)$
"Gaussian distribution" "mean" "covariance"

So, let's imagine that there's some hidden (or latent) random variable z , and the term *latent* just means hidden or unobserved (it means that it exists but you don't get to see the value directly), and $x^{(i)}$ and $z^{(i)}$ have this **joint distribution**

This is very very similar to the model we saw in **Gaussian Discriminant Analysis (GDA)**, but $z^{(i)}$ is **multinomial** with some set of parameters ϕ

For a mixture of two **Gaussians**, this would just be **Bernoulli** with two values, but if it were a mixture of **k Gaussians** then z can take on values from **1** through **k**, and if it was two **Gaussians** it'll just be **Bernoulli**

Then, once you know that one example comes from **Gaussian** number j , then x conditioned that $z^{(i)} = j$, that is drawn from a **Gaussian distribution** with some mean and some covariance Σ

The two unimportant ways this is different than **GDA** is: **(1)** We've set z to be one of **k** values instead of one of **2** values, and **GDA (Gaussian Discriminant Analysis)**, we had the labels y take on one of **2** values. **(2)** We have Σ_j instead of Σ . So by convention, when we fit **mixture of Gaussians models**, we let each **Gaussian** have its own covariance matrix Σ but you can actually force it to be the same if you want. But these are the trivial differences

The most significant difference is that, in **Gaussian Discriminant Analysis (GDA)**, we had labeled examples $(x^{(i)}, y^{(i)})$ where y was observed

$$\begin{array}{c} (x^{(i)}, y^{(i)}) \\ \uparrow \\ \text{observed} \end{array}$$

The main difference between this and **Gaussian Discriminant Analysis** is now we have replaced that with this latent (hidden) random variable $z^{(i)}$ that you do not get to see in the **training set**

If only we knew the value of the $z^{(i)}$'s, which we don't, but only if we did we could use **maximum likelihood estimation (MLE)** to estimate everything. So we would write the **log likelihood** of the parameters:

If we knew the $z^{(i)}$'s, we can use MLE:

$$\ell(\phi, \mu, \Sigma) = \sum_{i=1}^m \log p(x^{(i)}, z^{(i)}; \phi, \mu, \Sigma)$$

$$\phi_j = \frac{1}{m} \sum_{i=1}^m 1\{z^{(i)} = j\}$$

$$\mu_j = \frac{\sum_{i=1}^m 1\{z^{(i)} = j\} x^{(i)}}{\sum_{i=1}^m 1\{z^{(i)} = j\}}$$

$$\Sigma_j = \dots \leftarrow \text{"in the lecture notes"}$$

So if only you knew the values of the $z^{(i)}$'s then you could use **maximum likelihood estimates** and this is what you get. These two are exactly the formulas we had for **Gaussian Discriminant Analysis** except we've replaced y with z and then there's some other formula for Σ that's written in the lecture notes

But the reason we can't use these formulas is we don't actually know what are the values of z . So what we will do in the **EM algorithm** is two steps

EM (Expectation Maximization) algorithm:

In the first step, we will guess the value of the \mathbf{z} 's and in the second step we will use these equations using the values of the \mathbf{z} 's we just guessed

Sometimes in machine learning there is something that's called a **bootstrap procedure** where you get something that runs an algorithm using your guesses and then run the algorithm again

So, the **EM algorithm** has two steps

The **E-step (Expectation step)** is :

EM (Expectation Maximization) :

Expectation - step (Guess value of $z^{(i)}$'s):

$$\text{Set } w_j^{(i)} = P(z^{(i)} = j \mid x^{(i)}; \phi, \mu, \Sigma)$$

So $w_j^{(i)}$ is going to be the probability that $\mathbf{z}^{(i)} = \mathbf{j}$ given all the parameters

And much as we did with **Generative learning algorithms**, we used **Bayes' rule** to estimate the probability of \mathbf{y} given \mathbf{x} . So to compute this, you use a similar **Bayes' rule** type of calculation:

EM (Expectation Maximization):

Expectation-step (Guess value of $z^{(i)}$'s):

$$\text{Set } w_j^{(i)} = P(z^{(i)} = j \mid x^{(i)}; \phi, \mu, \Sigma)$$

$$= \frac{P(x^{(i)} \mid z^{(i)} = j) P(z^{(i)} = j)}{\sum_{l=1}^k P(x^{(i)} \mid z^{(i)} = l) P(z^{(i)} = l)}$$

$P(z^{(i)} = j)$ would be ϕ_j , which is just a **Bernoulli probability**. Remember that z is **multinomial**, so z is **multinomial** with parameters ϕ . The parameters ϕ for **multinomial distributions** tell you "what's the chance of z being 1, 2, 3, 4, and so on up to k , and so the chance of $z^{(i)}$ being equal to j is just ϕ_j . It's just (readed ?) off one of the parameters in your **multinomial probability** for the odds of z being different values. And similarly, the terms in the denominator :

Expectation-step (Guess value of $z^{(i)}$'s):

$$\text{Set } w_j^{(i)} = P(z^{(i)} = j \mid x^{(i)}; \phi, \mu, \Sigma)$$

$$= \frac{P(x^{(i)} \mid z^{(i)} = j) P(z^{(i)} = j)}{\sum_{l=1}^k P(x^{(i)} \mid z^{(i)} = l) P(z^{(i)} = l)}$$

"comes from a Gaussian density" $\rightarrow \mathcal{N}(\mu_j, \Sigma_j)$

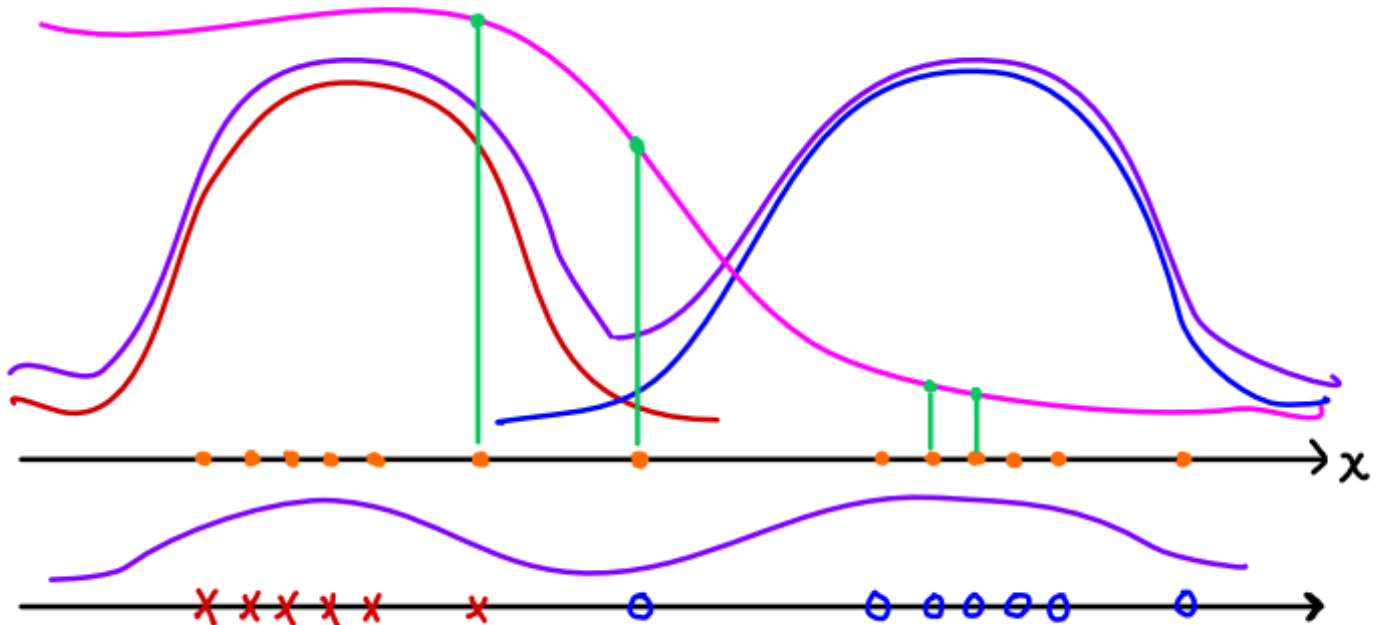
$z \sim \text{Multinomial}(\phi) \rightarrow \phi_j$

$$= \frac{1}{(2\pi)^{n/2} |\Sigma_j|^{1/2}} \exp \left(-\frac{1}{2} (x^{(i)} - \mu_j)^T \Sigma_j^{-1} (x^{(i)} - \mu_j) \right)$$

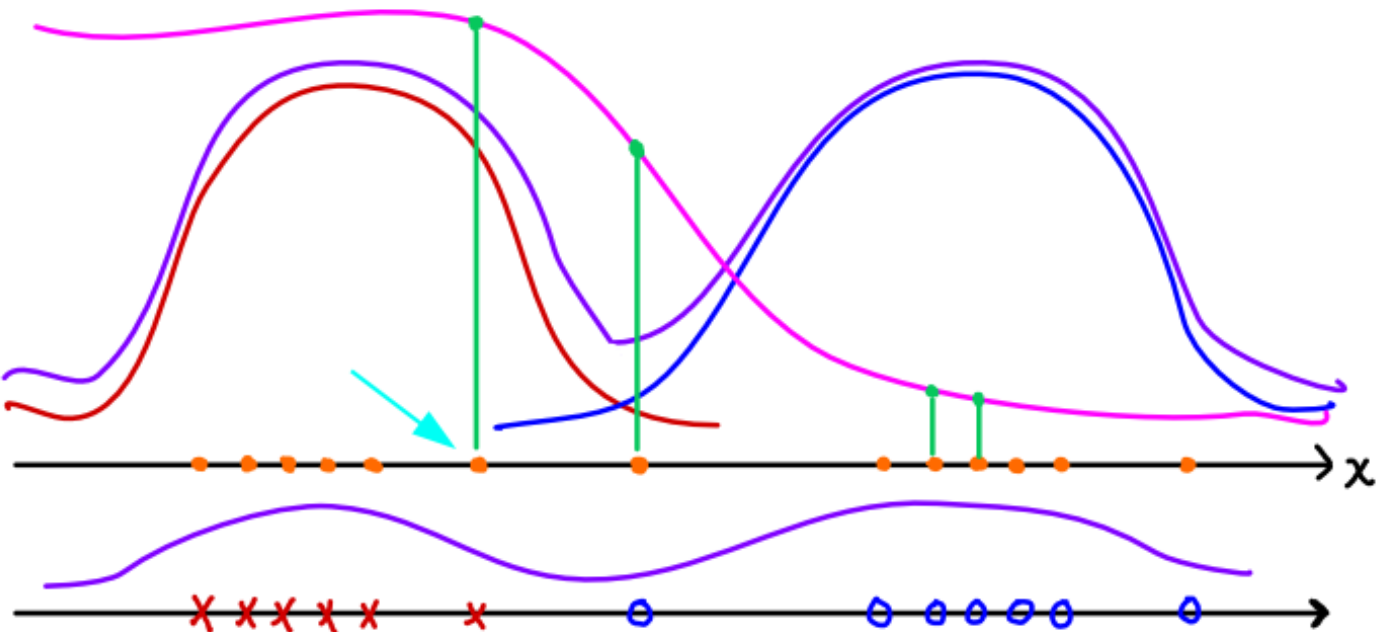
The **orange** term is from Gaussian and the **pink** term is from the **multinomial probability** that you have for z

That's how you plug in all of these numbers and use **Bayes rule** (use this equation) to compute, given the position of all these **Gaussians**, what the chance is of $w_j^{(i)}$ taking on a certain value

To make this concrete, if you were to scan from right to left, you'd get a **sigmoid function** like this. So $w_j^{(i)}$ is just the height of this **sigmoid**; it's just a chance of each of these examples coming from either the $z=1$ or $z=0$, and then you store all of these numbers in the variables $w_j^{(i)}$



So $w_j^{(i)}$ is just to compute the **posterior chance** of this example coming from the left **Gaussian** vs the right **Gaussian**:



That's the **E-step**; you compute the $w_j^{(i)}$ for every single training example i

In the **E-step** we're trying to guess the values of the **z**'s; we need to figure what's the probability of **z** being **1, 2, 3, 4**, up to **k** was stored here)

In the **M-step**, what we're going to do is use the formulas we have for **maximum likelihood estimation (MLE)** (compare these with the equations we had above)

Maximization - step:

$$\phi_j := \frac{1}{N} \sum_{i=1}^N w_j^{(i)}$$

$$\mu_j = \frac{\sum_{i=1}^N w_j^{(i)} x^{(i)}}{\sum_{i=1}^N w_j^{(i)}}$$

$$\Sigma_j = \dots$$

These equations are a lot like the equations above except that instead of indicator $z^{(i)}=j$, we replaced it with $w_j^{(i)}$ (which by the way is the **expected value** of this **indicator function**)

Maximization - step:

$$\phi_j := \frac{1}{n} \sum_{i=1}^n w_j^{(i)}$$

$$\mu_j = \frac{\sum_{i=1}^n w_j^{(i)} x^{(i)}}{\sum_{i=1}^n w_j^{(i)}}$$

$$\Sigma_j = \dots$$

$$1\{z^{(i)} = j\}$$

↓

$$w_j^{(i)} = E[1\{z^{(i)} = j\}]$$

↖ "expected value"

Because the **expected value** of an **indicator function** is just equal to the probability of $z^{(i)}=j$ being true

One intuition of this **mixture of Gaussians algorithm** is that it's a little bit like **k-means** but with **soft assignment**. So in **k-means**, in the first step we would take each point and just assign it to one of the **k cluster centroids** that is even a little bit closer to one **cluster centroid** than another. Even if it was just a little bit closer to one **cluster centroid** than another, **k-means** will just make what's called a **hard assignment**, meaning whatever **cluster centroid** it's closer to, we just assigned it **100%** of that **cluster centroid**

EM implements a softer way of assigning points to the different **cluster centroids** because instead of just picking the one closest **Gaussian** center and assigning it there, it uses these probabilities and gives it a weighting in terms of how much is assigned to **Gaussian 1** vs **Gaussian 2**. And then second, it updates the **means** accordingly (sum over all the $x^{(i)}$'s to the extent they're assigned to that **cluster centroid**, divided by the number of examples assigned to that **cluster centroid**)

So that's one intuition between **EM** and **k-means**. When you run this algorithm, it turns out that this algorithm will converge with some caveats and this will find a pretty decent estimate of the parameters (of fitting a **mixture of two Gaussians model**)

After you're fitting the **EM algorithm**, you now have a joint density of a **P(x,z)**, and so the density for **x** is just:

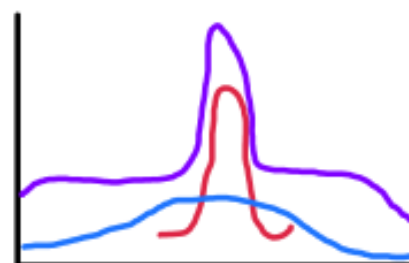
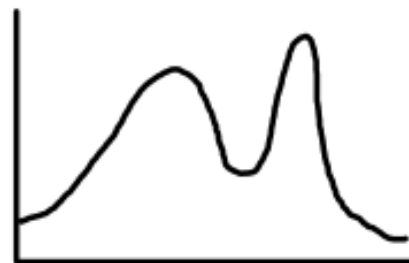
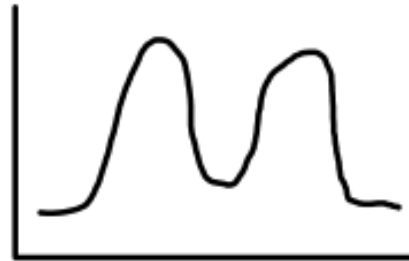
$$P(x, z)$$

$$P(x) = \sum_z P(x, z)$$

So a **mixture of Gaussians** can fit distributions that look like this:

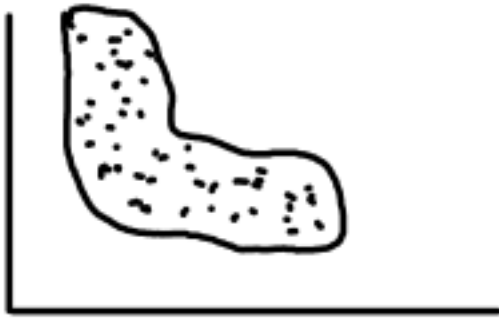
$$P(x, z)$$

$$P(x) = \sum_z P(x, z)$$



These are **mixtures of two Gaussians**, so this gives you a very rich family of models to fit very complicated distributions. A **mixture of two Gaussians** can actually fit a model of different things and a **mixture of more than two Gaussians** can fit even richer models

And so by doing this, you can now model $\mathbf{p}(\mathbf{x})$ for many complicated densities, including this one:

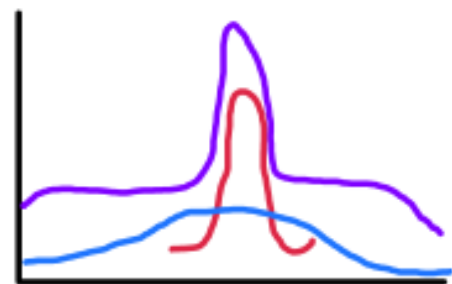
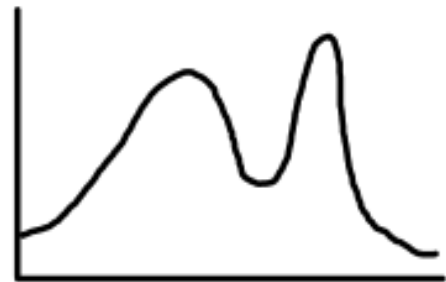


This will allow you to fit a probability density function that puts a lot of probability models on a region that looks like this

When you have a new example, you can evaluate $\mathbf{p}(\mathbf{x})$ and if $\mathbf{p}(\mathbf{x})$ is large, then you can say it looks okay, and if $\mathbf{p}(\mathbf{x})$ is less than ϵ , you can flag an anomaly and say “take another look at this engine”

$$P(x, z)$$

$$P(x) = \sum_z P(x, z)$$



$$P(x)$$

$$P(x) > \epsilon - \text{ok}$$

$$P(x) < \epsilon - \text{anomaly}$$

↑
"epsilon"

If your goal is, given a model $P(x, z; \theta)$, to maximize $P(x^{(i)}; \theta)$:

$$P(x, z; \theta)$$

$$\arg \max_{\theta} \prod_{i=1}^m P(x^{(i)}; \theta)$$

This is what **maximum likelihood** is supposed to do, that **EM** is exactly trying to do that

This is actually the weighting you assign to a certain **Gaussian**:

Expectation-step (Guess value of $z^{(i)}$'s):

$$\text{Set } w_j^{(i)} = P(z^{(i)} = j | x^{(i)}; \phi, \mu, \Sigma)$$

$$= \frac{P(x^{(i)} | z^{(i)} = j) P(z^{(i)} = j)}{\sum_{l=1}^K P(x^{(i)} | z^{(i)} = l) P(z^{(i)} = l)}$$

"comes from a Gaussian density" $\rightarrow N(\mu_j, \Sigma_j)$
 $z \sim \text{Multinomial}(\phi) \rightarrow \phi_j$

$$= \frac{1}{(2\pi)^{n/2} |\Sigma_j|^{1/2}} \exp \left(-\frac{1}{2} (x^{(i)} - \mu_j)^T \Sigma_j^{-1} (x^{(i)} - \mu_j) \right)$$

One way to think of this is:

$w_j^{(i)}$ is how much $x^{(i)}$ is assigned to the μ_j Gaussian

So $w_j^{(i)}$ is the strength of how strongly you want to assign that training example $x^{(i)}$ to that cluster or to that particular **Gaussian**. This is a number between **0** and **1**. The strength of all the assignments, and every point is assigned with a total strength equal to **1** because all these properties must sum up to **1**, so we're going to take this point and assign it **0.8** to a more closer **Gaussian** and **0.2** to a more distant **Gaussian**, for example, and this is our guess that there's an **80%** chance that it came with that **Gaussian** and a **20%** chance it came with the second **Gaussian**

When you're running the **EM algorithm**, you never know what the true values of **z** are. You're given a dataset, so you're only told the **x**'s, and as far as we know, these airplane engines were generated off two different **Gaussians**. Maybe there are two separate assembly processes; one from **plant #1**, one from **plant #2**, and maybe they actually operate a little bit differently but by the time the two suppliers of aircraft engines get to you, they've been mixed together. And so, you can't tell anymore which aircraft engine came from **plant #1** and which aircraft engine came from **plant #2**, and you don't even know that there are two **plants**. You just see the stream of aircraft engines. You're hypothesizing that there are two types

So, in every iteration of **EM**, you're taking each aircraft engine and guessing the chance it came from **plant #1** and **plant #2**. That's the **E-step**. And then in the **M-step**, you look at all the engines that you're kind of guessing were generated by **process 1**, and you update your **Gaussian** to be a better model for all the things that you kind of think were generated by **process 1**. And if there's something that you're absolutely sure came from **process 1**, then it has a weight of **1** (close to **1**) and if you think something has a **10%** of coming from **process 1**, then that example is given a lower weight and now you update the **mean** for that **Gaussian**

We'll go over the more formal derivation of the **EM algorithm** that doesn't rely on this "hand-wavy" argument of "let's guess the **z**'s and use **maximum likelihood** with the guessed values"

Derivation of EM:

In order to lead up to that derivation, we'll describe **Jensen's inequality**

Jensen's inequality:

Let f be a convex function

(e.g. $f''(x) > 0$)

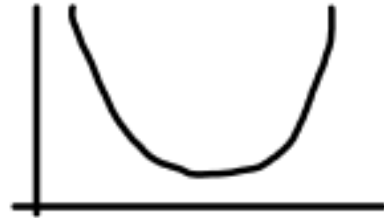


To do **EM**, we're actually going to need a concave function

Jensen's inequality:

Let f be a convex function

(e.g. $f''(x) > 0$)

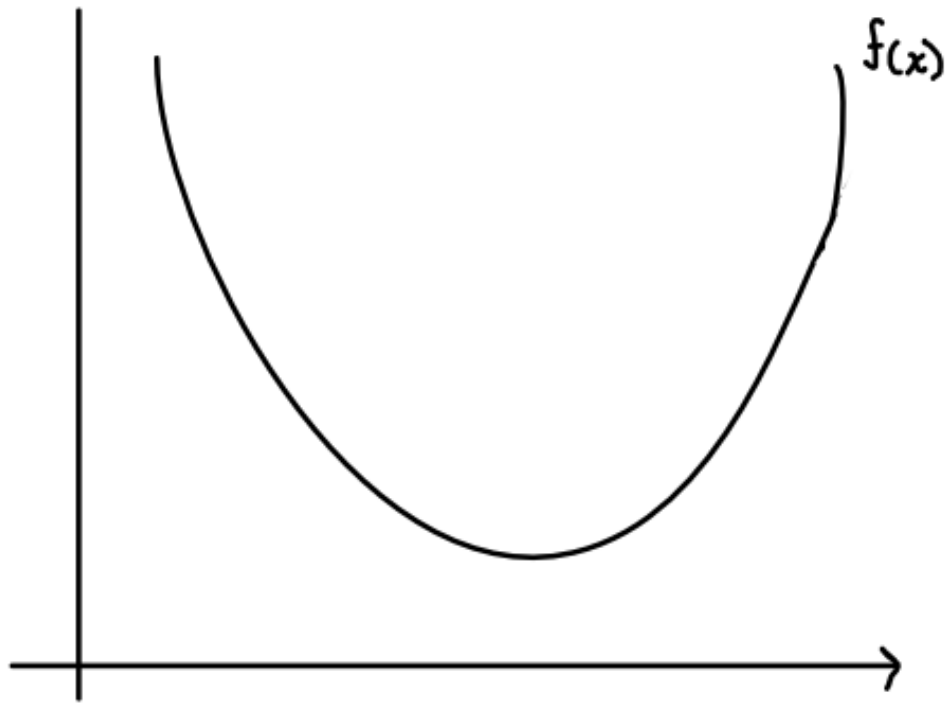


Let x be a random variable

$$\text{Then } f(E x) \leq E [f(x)]$$

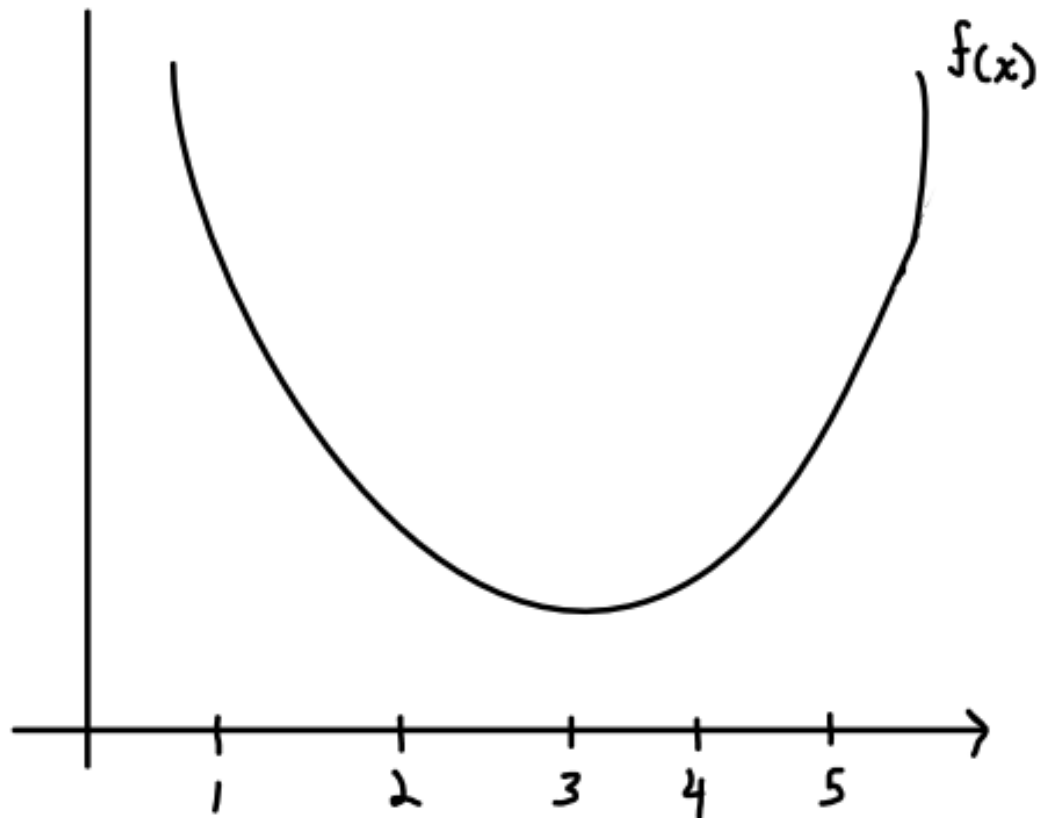
Here's an example:

Example



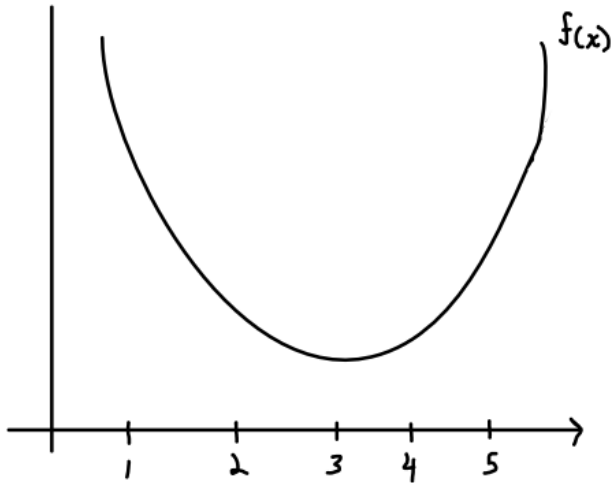
And let's say that these are the values:

Example



And just for illustration, suppose that x is equal to:

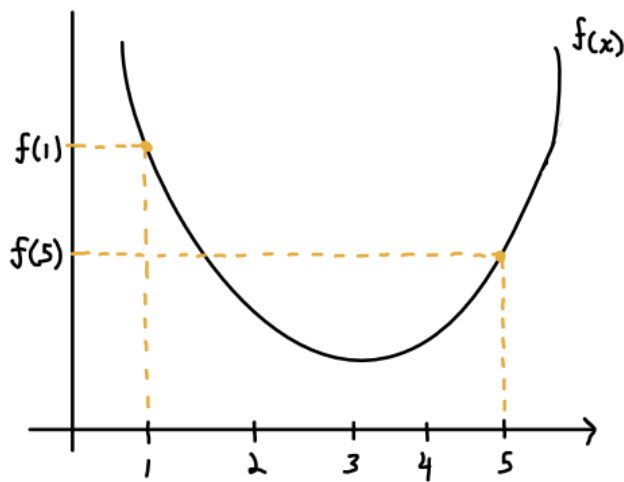
Example



$$x = \begin{cases} 1 & \text{with probability } \frac{1}{2} \\ 5 & \text{with probability } \frac{1}{2} \end{cases}$$

Then, here is $f(1)$ and $f(5)$:

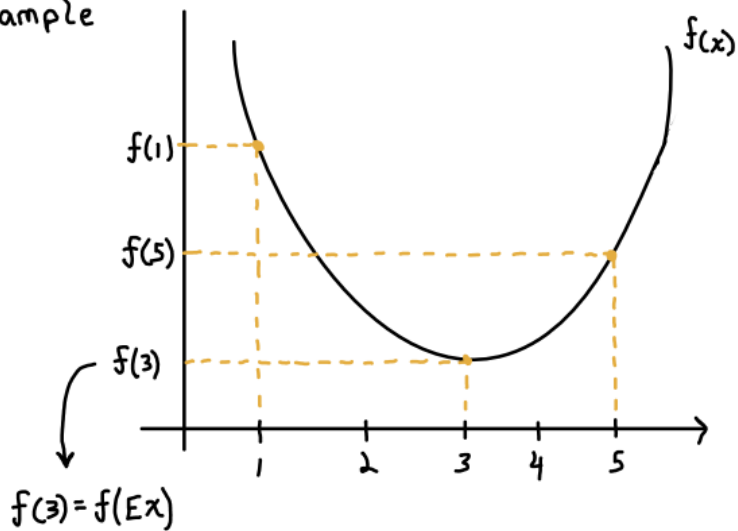
Example



$$x = \begin{cases} 1 & \text{with probability } \frac{1}{2} \\ 5 & \text{with probability } \frac{1}{2} \end{cases}$$

Here is $f(3)$, and $f(3)$ is f of the **expected value** of x :

Example



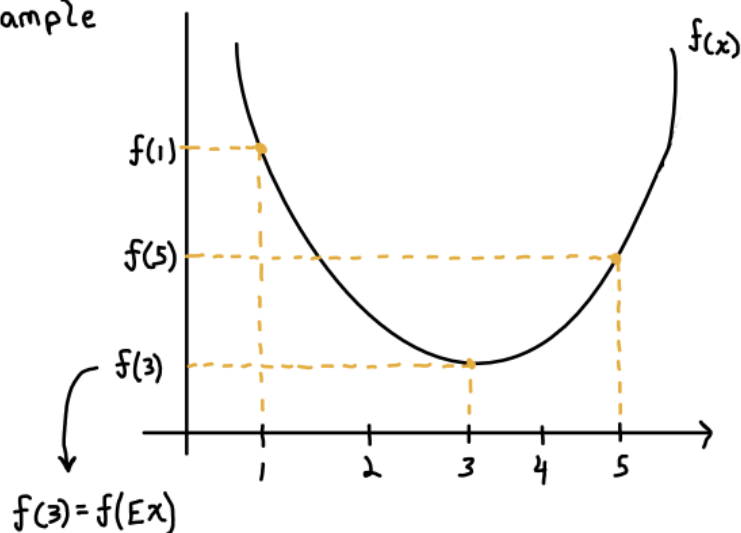
$$x = \begin{cases} 1 & \text{with probability } \frac{1}{2} \\ 5 & \text{with probability } \frac{1}{2} \end{cases}$$

$$E[X] = E x = 3$$

So, the **expected value** of x is the average of x , is equal to **3**. And so the **$f(\text{expected value of } x)$** is equal to that value (**3, $f(3)$**)

Whereas the **$E[f(x)]$** is the **mean** of **$f(1)$** and **$f(5)$** :

Example



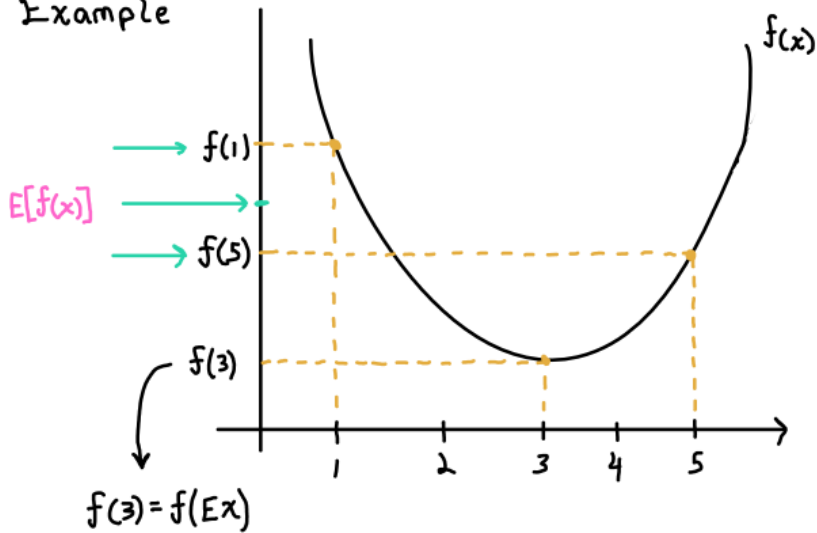
$$x = \begin{cases} 1 & \text{with probability } \frac{1}{2} \\ 5 & \text{with probability } \frac{1}{2} \end{cases}$$

$$E[X] = E x = 3$$

$$E[f(x)] = \frac{1}{2} f(1) + \frac{1}{2} f(5)$$

So the **expected value** of **$f(x)$** , **$f(x)$** is a **50%** chance of being **$f(1)$** and a **50%** chance of being **$f(5)$** , is equal to this value in the middle:

Example



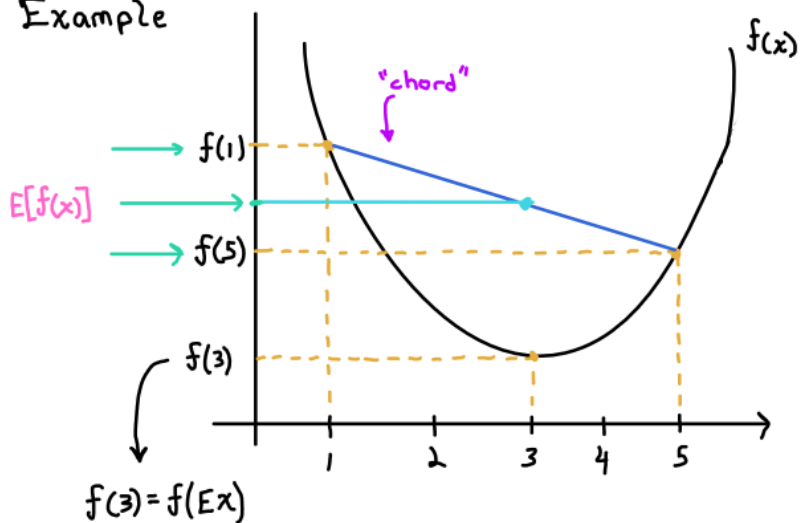
$$X = \begin{cases} 1 & \text{with probability } \frac{1}{2} \\ 5 & \text{with probability } \frac{1}{2} \end{cases}$$

$$E[X] = E[X] = 3$$

$$E[f(X)] = \frac{1}{2} f(1) + \frac{1}{2} f(5)$$

And so in this example, the **expected value** of $f(x)$ is greater than $f(\text{expected value of } x)$, as predicted by **Jensen's inequality**

Example



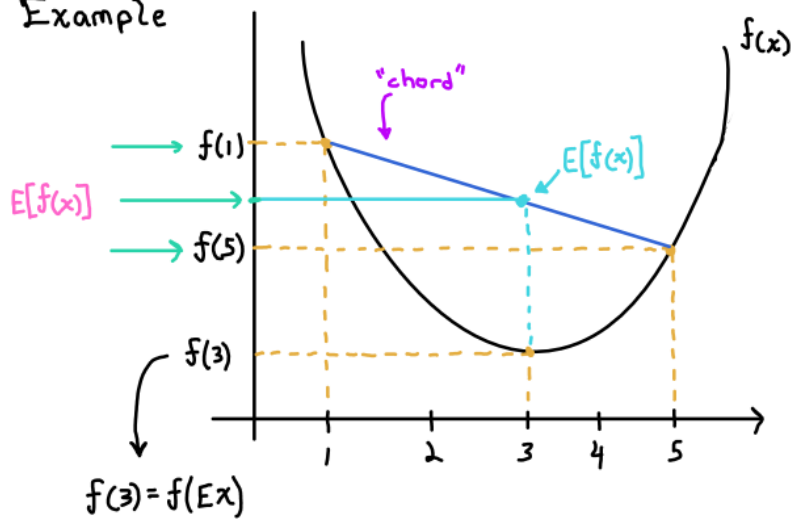
$$X = \begin{cases} 1 & \text{with probability } \frac{1}{2} \\ 5 & \text{with probability } \frac{1}{2} \end{cases}$$

$$E[X] = E[X] = 3$$

$$E[f(X)] = \frac{1}{2} f(1) + \frac{1}{2} f(5)$$

If you draw a line that connects these two, then the midpoint of this line (called a **chord**) is the height of **expected value of $f(x)$** :

Example



$$f(3) = f(E[x])$$

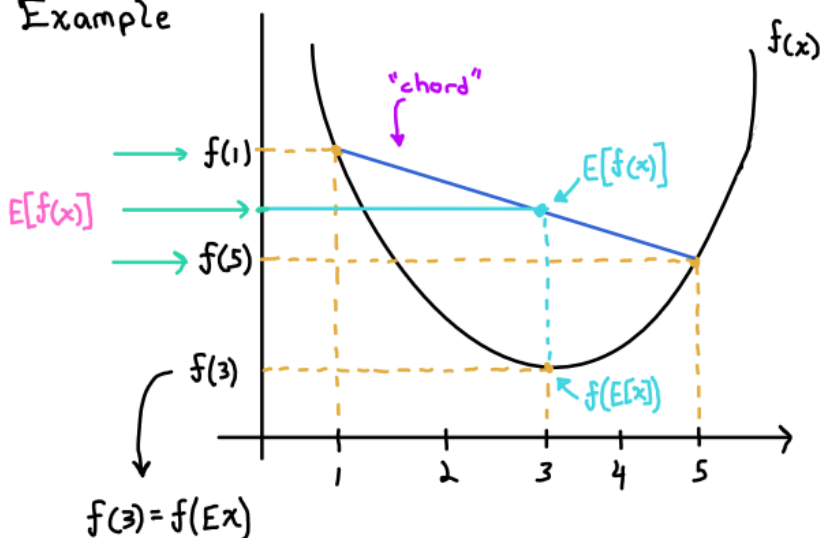
$$E[f(x)] = \frac{1}{2} f(1) + \frac{1}{2} f(5)$$

$$x = \begin{cases} 1 & \text{with probability } \frac{1}{2} \\ 5 & \text{with probability } \frac{1}{2} \end{cases}$$

$$E[X] = E[x] = 3$$

And this point is **f(expected value of x)**:

Example



$$f(3) = f(E[x])$$

$$E[f(x)] = \frac{1}{2} f(1) + \frac{1}{2} f(5)$$

$$x = \begin{cases} 1 & \text{with probability } \frac{1}{2} \\ 5 & \text{with probability } \frac{1}{2} \end{cases}$$

$$E[X] = E[x] = 3$$

And in any convex function, if you draw any chord, **$E[f(x)]$** is always higher than **$f(E[x])$** , which is another way of seeing why **Jensen's inequality** holds true

One addendum:

Jensen's inequality:

Let f be a convex function

(e.g. $f''(x) > 0$)



Let x be a random variable

$$\text{Then } f(E[x]) \leq E[f(x)]$$

Further, if $f''(x) > 0$ (f is strictly convex)

then

$$E[f(x)] = f(E[x]) \iff x \text{ is a constant}$$

($x = E[x]$ with probability 1)

A straight line is also a convex function, but this addendum is saying that if f is a strictly convex function, meaning basically it's not a straight line, then the only way for the left- and right-hand sides to be equal is if x is a constant. Meaning it's a random variable that always takes on the same value

Jensen's inequality says that left-hand side is always less than or equal to the right-hand side, and the only way it's equal is if x is a random variable that always takes on the same value

In advanced probability, the technical way of saying x is a constant is x is equal to $E[x]$ with probability 1

One more addendum to this is that the form of **Jensen's inequality** we are going to use is actually a form for a concave function (a concave function is just a negative of a convex function):

Let f be a concave function

(e.g. $f''(x) < 0$)



Let x be a random variable

$$\text{Then } f(E[x]) \geq E[f(x)]$$

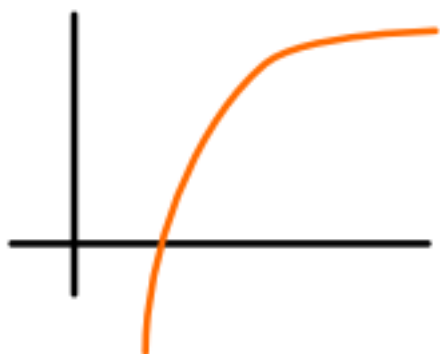
Further, if $f''(x) < 0$ (f is strictly concave)

then

$$E[f(x)] = f(E[x]) \iff x \text{ is a constant}$$

($x = E[x]$ with probability 1)

So the form of **Jensen's inequality** we're going to use is actually the concave form of **Jensen's inequality** and we're going to apply it to the log function. Log x looks like this:



So that's a concave function

Here's the **density estimation problem** (**density estimation** means you want to estimate $p(\mathbf{x})$):

Density Estimation problem:

Have a model for $P(\mathbf{x}, \mathbf{z}; \theta)$

only observe \mathbf{x} $\{\mathbf{x}^{(i)}, \dots, \mathbf{x}^{(m)}\}$

$$\ell(\theta) = \sum_{i=1}^m \log P(\mathbf{x}^{(i)}; \theta)$$

$$= \sum_{i=1}^m \log \sum_{\mathbf{z}^{(i)}} P(\mathbf{x}^{(i)}, \mathbf{z}^{(i)}; \theta)$$

Instead of writing out μ , Σ , and Φ like we did for the **mixture of Gaussians**, we're just going to capture all the parameters we have (whatever the parameters are) in one variable, θ

We only observe \mathbf{x} , so our **training set** looks like this

The **log likelihood** of the parameters θ is equal to the sum over your training examples, and this in turn is the log of sum over $\mathbf{z}^{(i)}$'s $P(\mathbf{x}^{(i)}, \mathbf{z}^{(i)}; \theta)$

Density Estimation problem:

Have a model for $P(x, z; \theta)$

only observe $x \quad \{x^{(i)}, \dots, x^{(m)}\}$

$$\ell(\theta) = \sum_{i=1}^m \log \underline{P(x^{(i)}; \theta)}$$

$$= \sum_{i=1}^m \log \sum_{z^{(i)}} \underline{P(x^{(i)}, z^{(i)}; \theta)}$$

↑
"joint distribution"

Because $P(x^{(i)}; \theta)$ is just taking the joint distribution and marginalizing out $z^{(i)}$

And so what we want is the **maximum likelihood estimation** which is defined the value of θ that maximizes this **log likelihood**

Density Estimation problem:

Have a model for $P(x, z; \theta)$

only observe x $\{x^{(i)}, \dots, x^{(m)}\}$

$$\ell(\theta) = \sum_{i=1}^m \log \underline{P(x^{(i)}; \theta)}$$

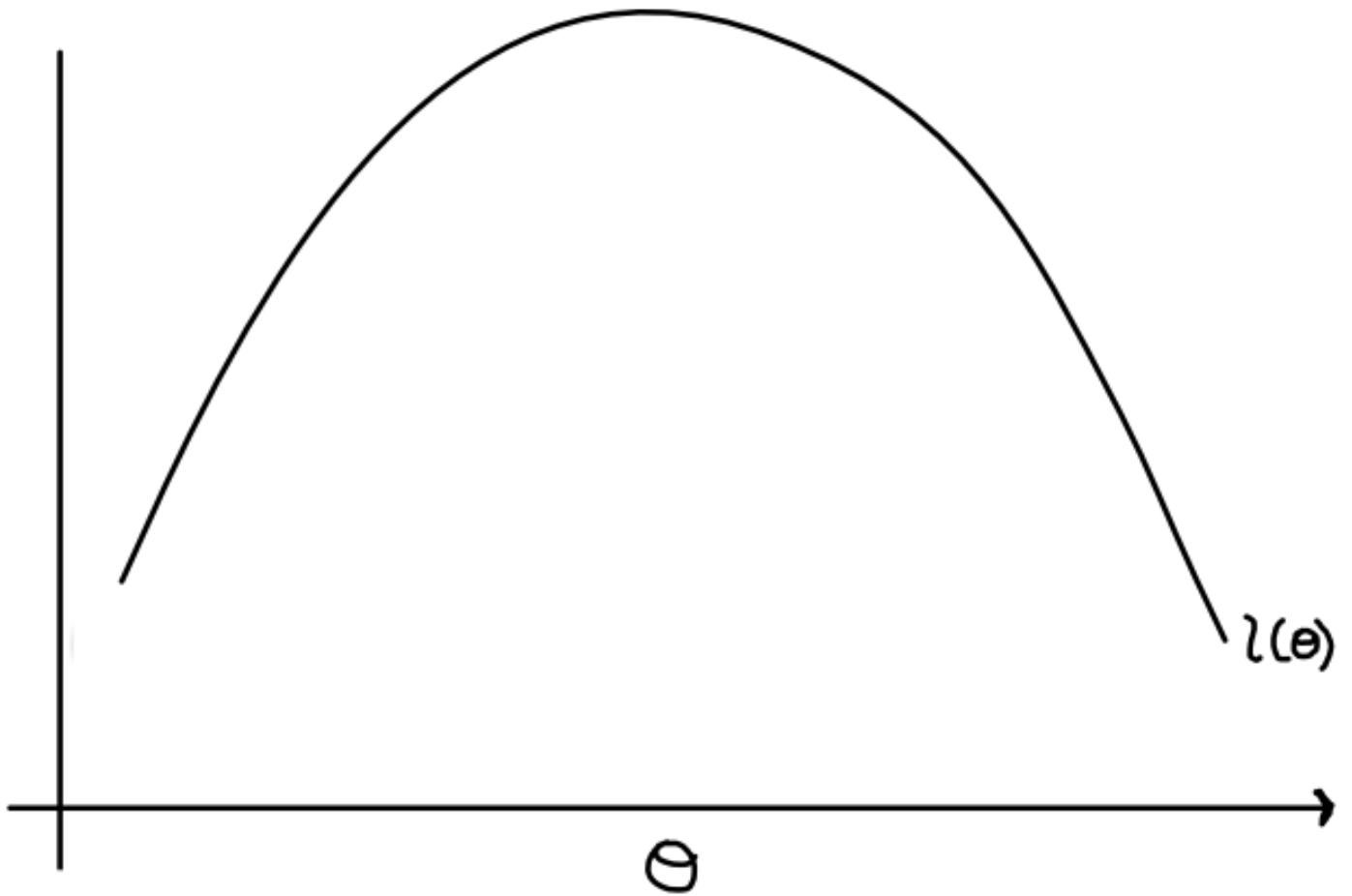
$$= \sum_{i=1}^m \log \sum_{z^{(i)}} \underline{P(x^{(i)}, z^{(i)}; \theta)}$$

↑
"joint distribution"

We want: $\arg \max_{\theta} \ell(\theta)$

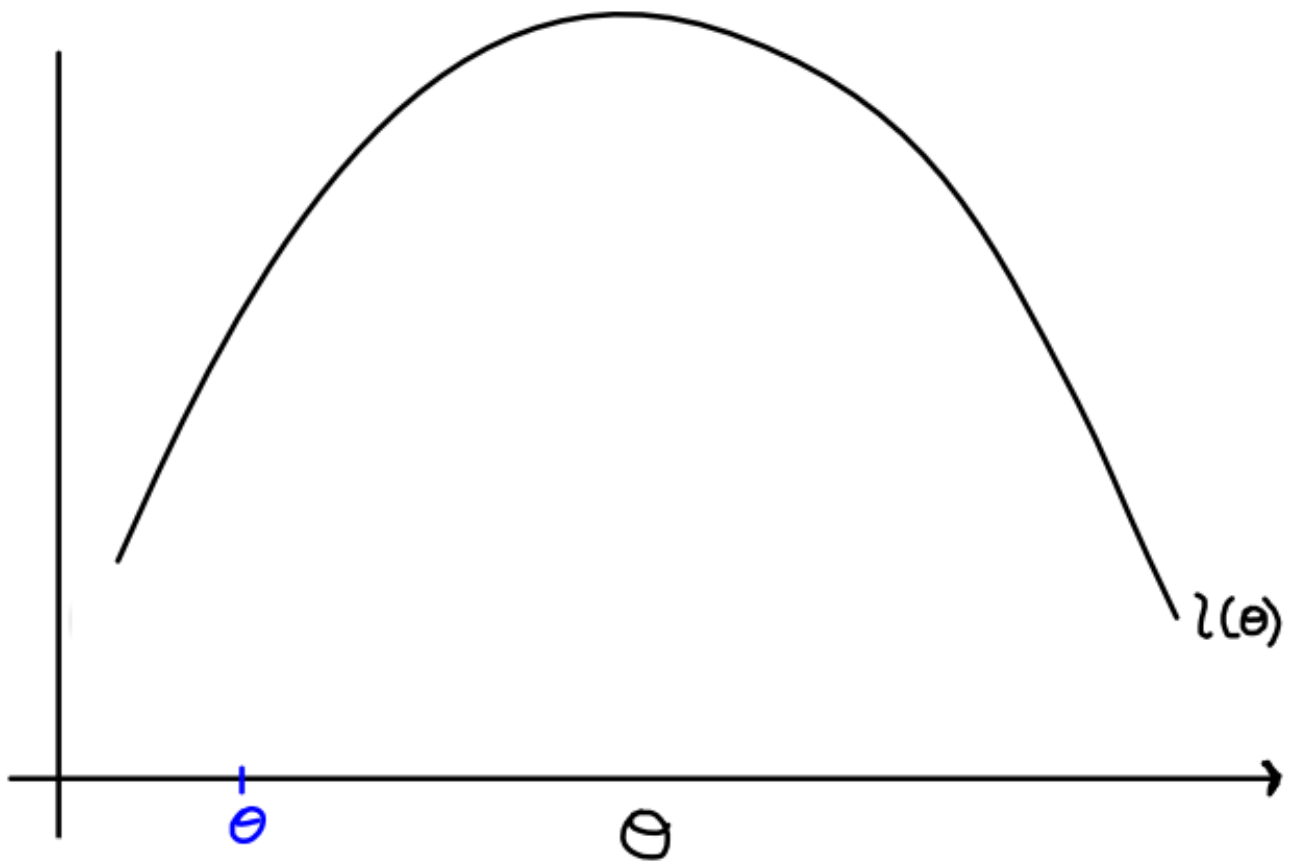
What we'd like to do is derive an algorithm, which will turn out to be an **EM algorithm**, as an iterative algorithm for finding the **maximum likelihood estimates** of the parameters θ

We'll draw a picture of that to keep in mind as we go through the math, which is:

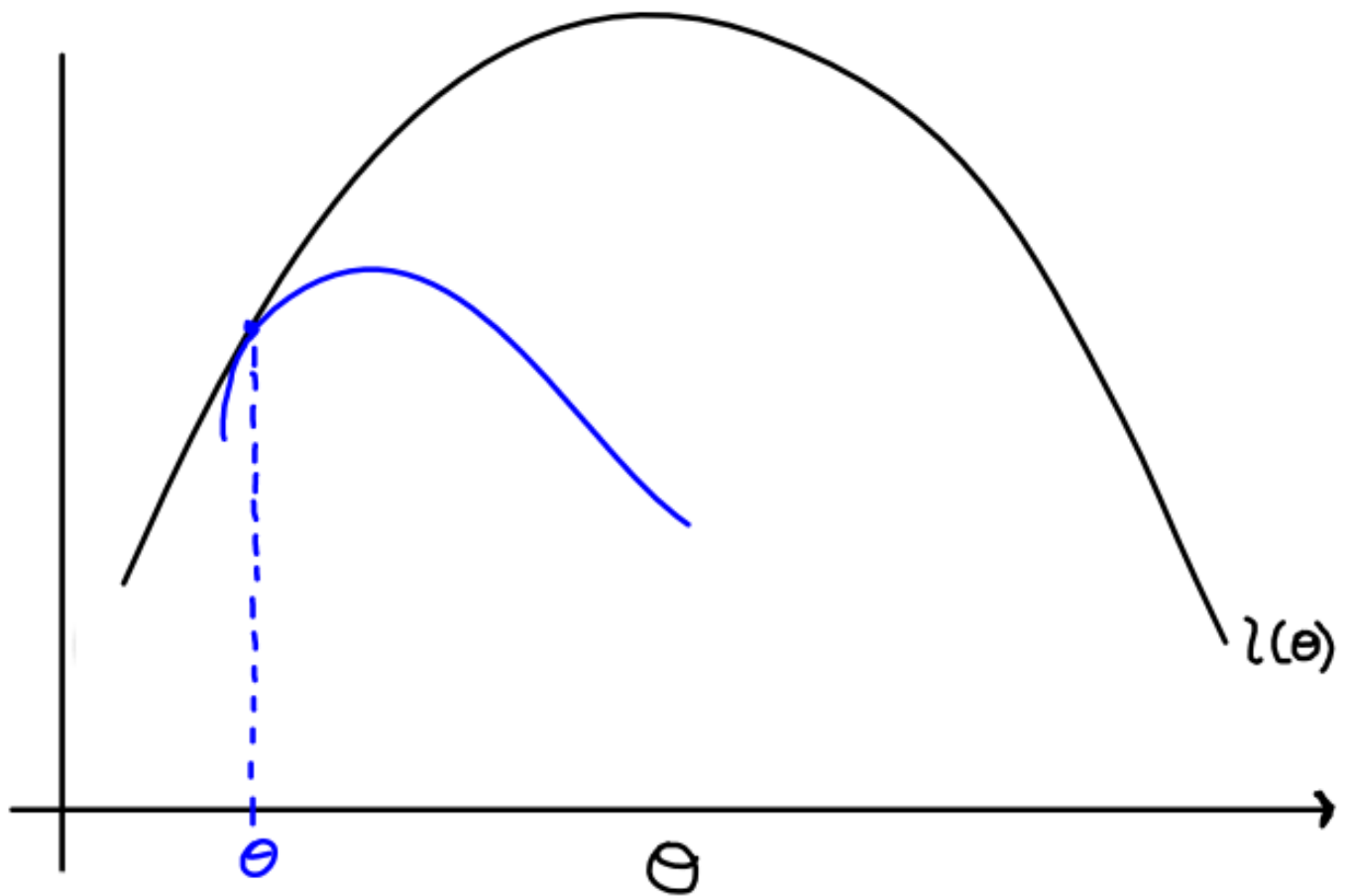


The horizontal axis is the space of possible values of the parameters θ and so there's some function $l(\theta)$ that you try to maximize

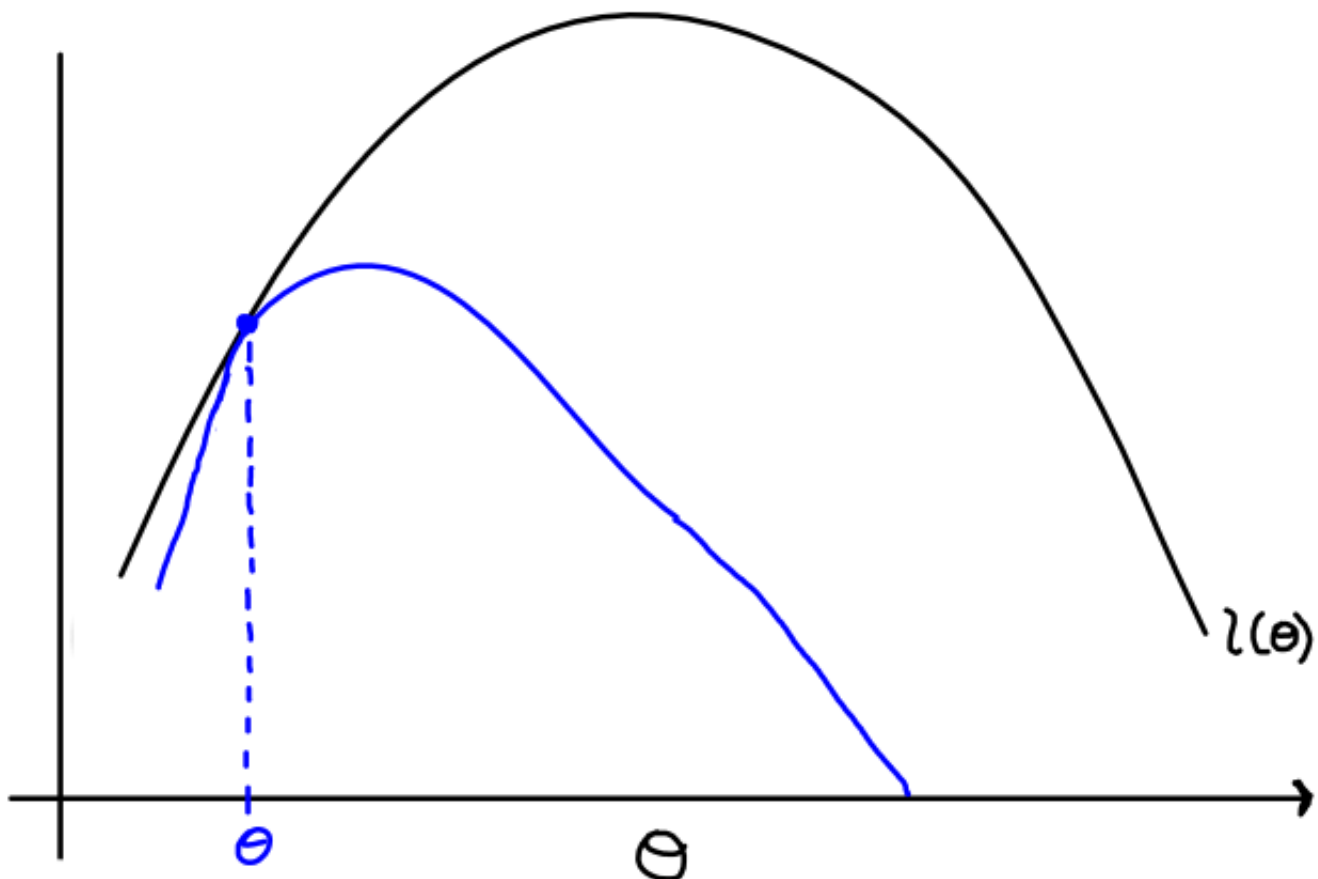
What **EM** does is, let's say you (randomly) initialize θ as some value (similar to the ***k-means cluster centroids*** where you just randomly initialize your μ 's with a ***mixture of Gaussians***):



What the **EM algorithm** does is in the **E-step**, we're going to construct a lower bound, shown in **blue** here, for the **log likelihood**:



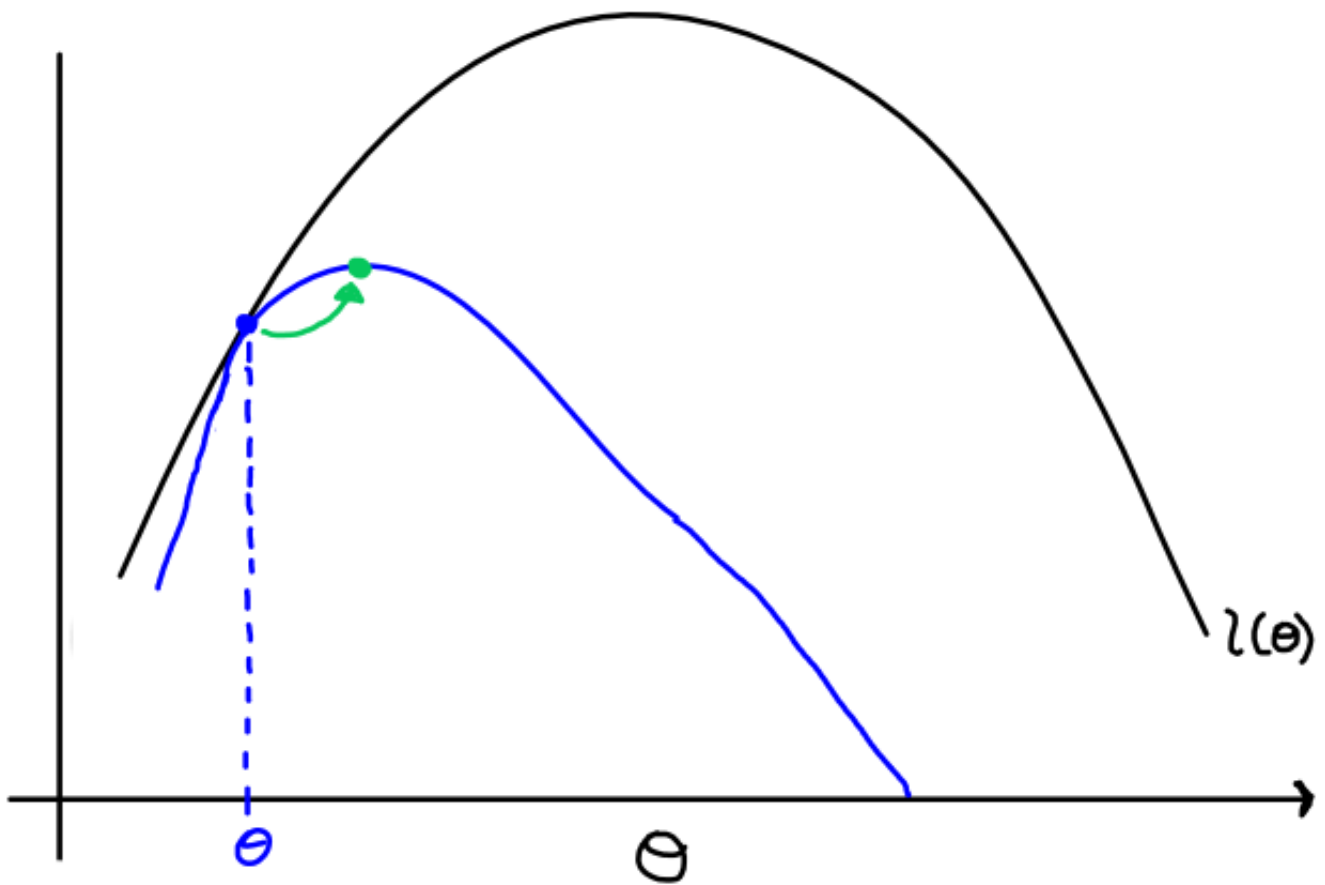
And this lower bound (**blue curve**) has two properties. **(1)** It is a lower bound. So everywhere you look over all values of θ , the **blue curve** lies below the **black curve**. **(2)** The second property that the **blue curve** has is that it is equal to the **black curve** at the current value of θ



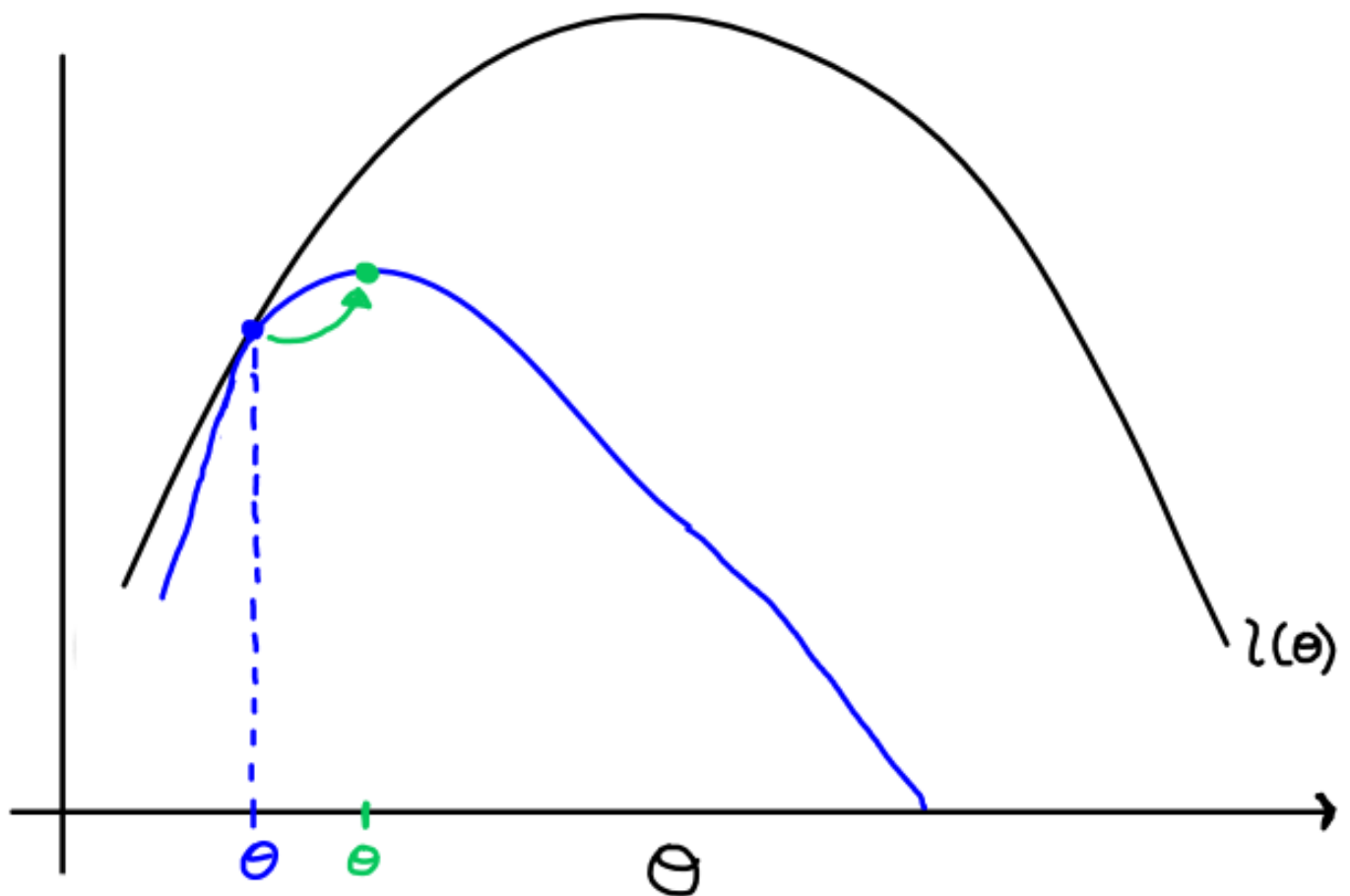
So, what the **E-step** does is construct the lower bound that looks like this

We want the **blue curve** to be equal to the **black curve** at the old value of θ , so we'll use that addendum to *Jensen's inequality* when we derive that. So the **E-step** is to draw out the **blue curve**

What the **M-step** does is it takes the **blue curve** and then it finds the maximum:

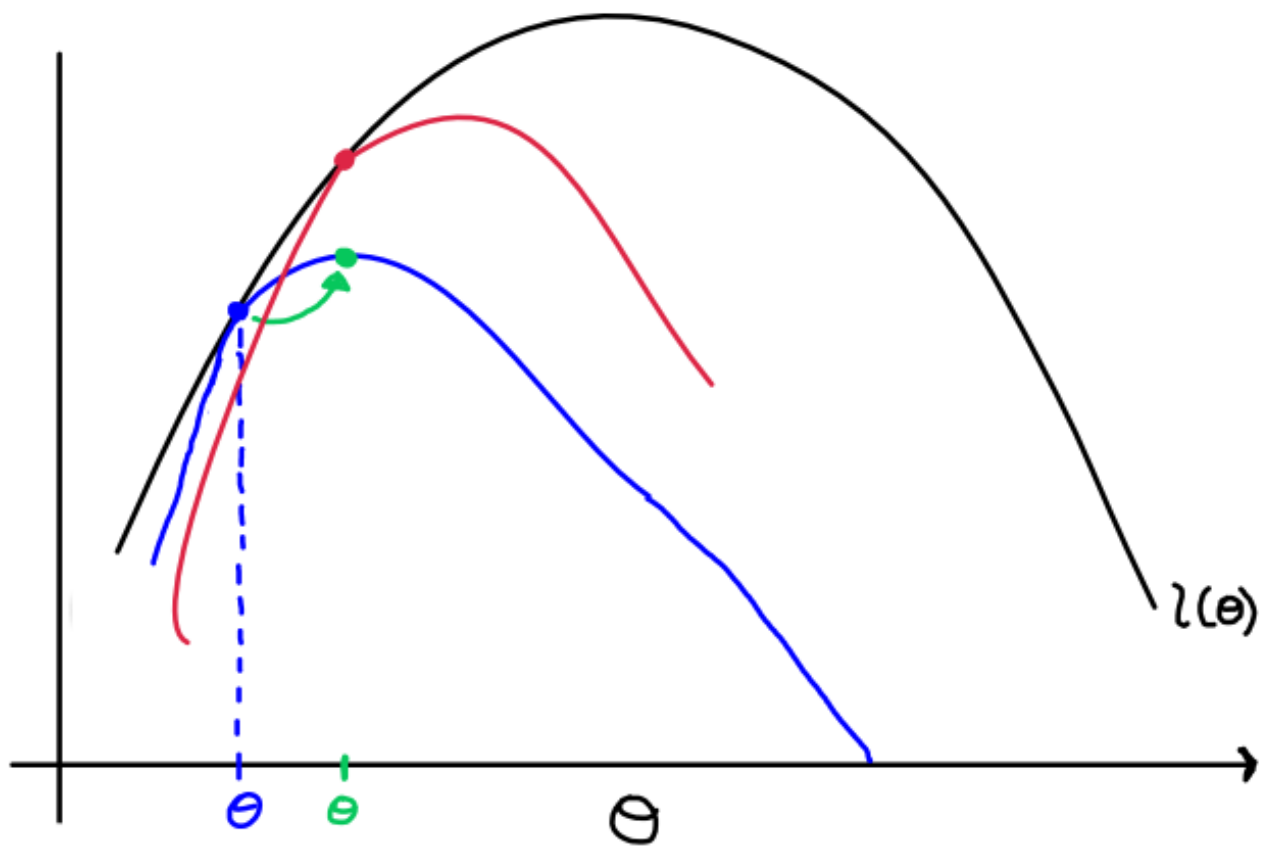


And one step of **EM** will then move θ from this blue value to this green value:

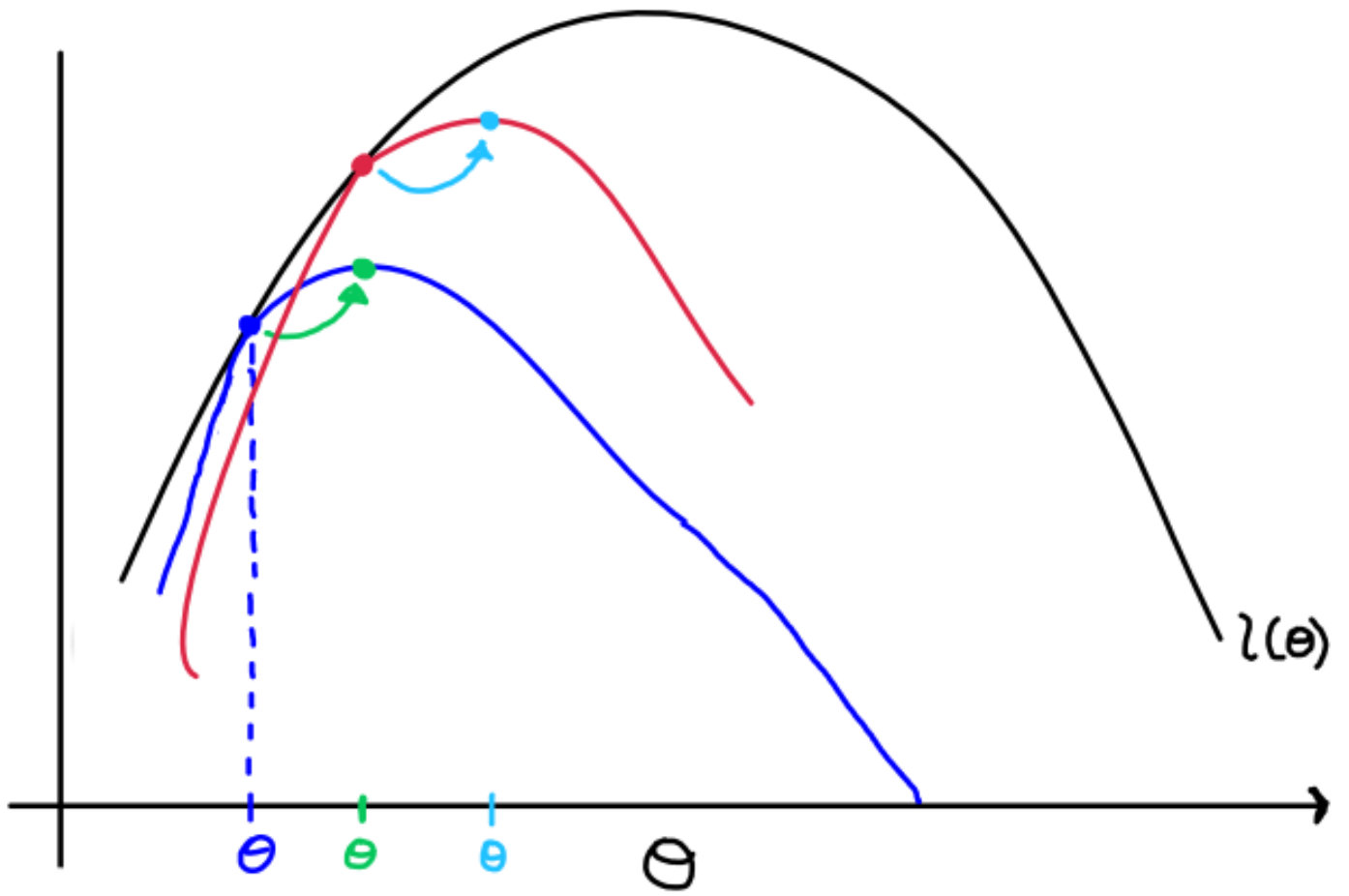


So, the **E-step** constructs the **blue curve** and the **M-step** finds the maximum of the **blue curve**, and this is one iteration of **EM**

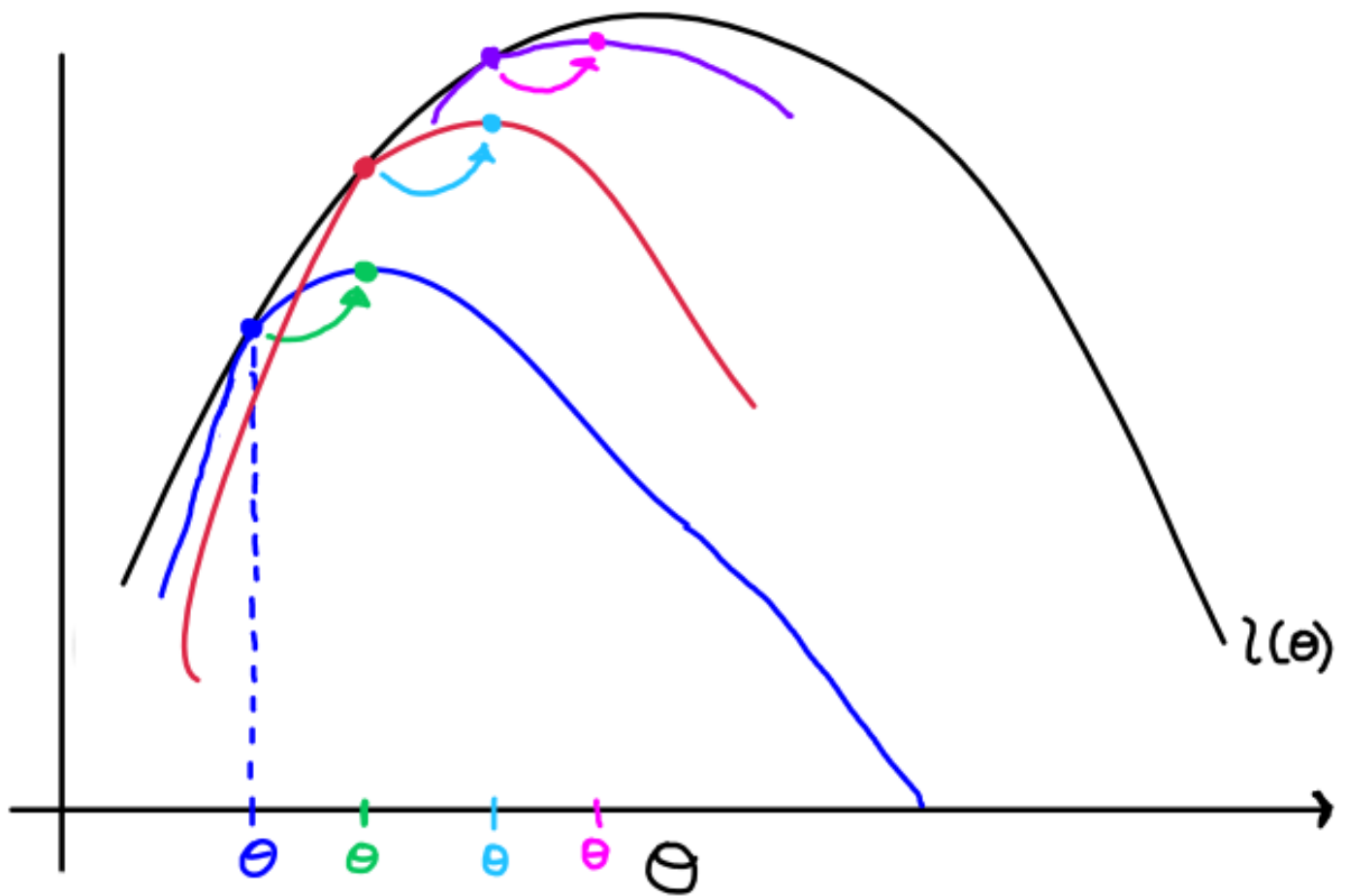
The second iteration of **EM**, now that we're at this **green** value of θ , is we'll construct a new lower bound and again the values are equal at this new value of θ . That's the **E-step**:



And then the **M-step** will maximize this **red curve**:

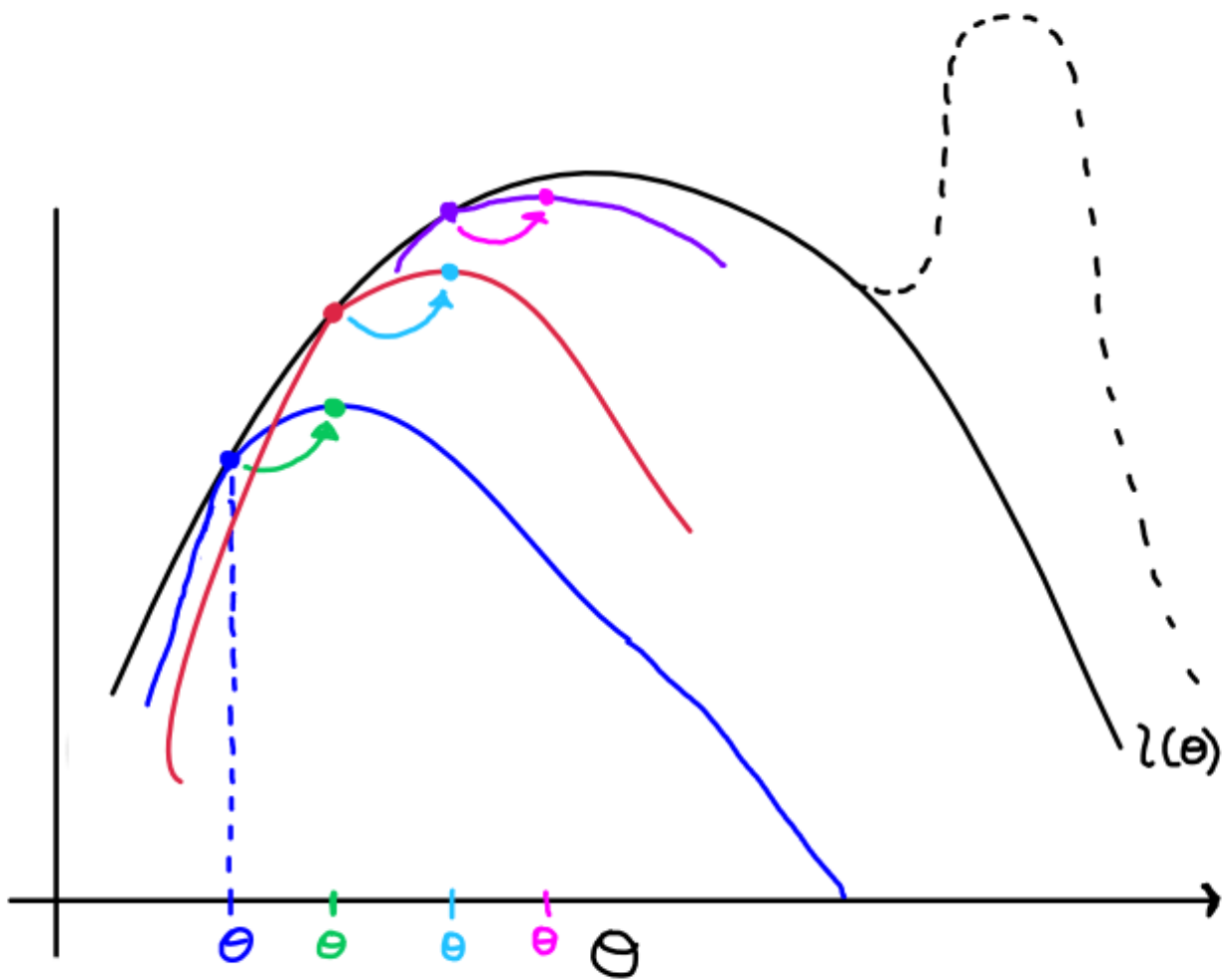


And so on:

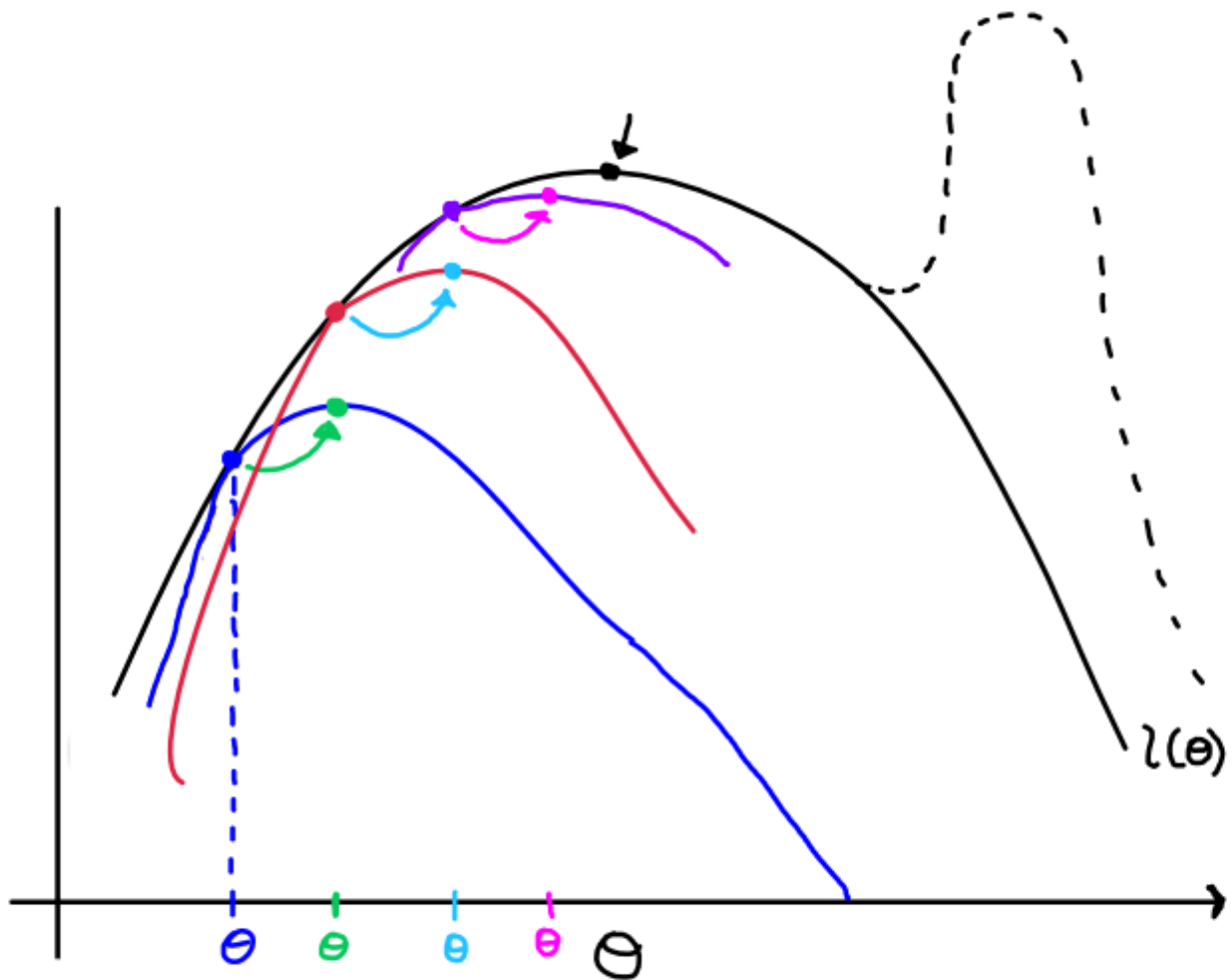


You can kind of tell that as you keep running **EM**, this is constantly trying to increase $l(\theta)$ (trying to increase the **log likelihood**) until it converges to a **local optimum**

The **EM algorithm** does converge only to **local optimum**, so if there was another even bigger curve there, then it may never find its way over to that better **optimum**:



But the **EM algorithm**, by repeatedly doing this, will hopefully converge to a pretty good **local optimum**:



We'll write out how we do that. We've already said that our goal is to find the parameters θ that maximizes this:

$$\max_{\theta} \sum_i \log P(x^{(i)}; \theta)$$

And so that equation we said is just now:

$$\max_{\theta} \sum_i \log P(x^{(i)}; \theta)$$

$$= \sum_i \log \sum_{z^{(i)}} P(x^{(i)}, z^{(i)}; \theta)$$

So this is just what we had written down previously

What we're going to do next is multiply and divide by this:

$$\max_{\theta} \sum_i \log P(x^{(i)}; \theta)$$

$$= \sum_i \log \sum_{z^{(i)}} P(x^{(i)}, z^{(i)}; \theta)$$

$$= \sum_i \log \sum_{z^{(i)}} Q_i(z^{(i)}) \frac{P(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})}$$

(where $Q_i(z^{(i)})$ is a probability distribution
i.e. $\left(\sum_{z^{(i)}} Q_i(z^{(i)}) \right) = 1$)

So we're going to multiply and divide by some **probability distribution**, and we'll decide later how to come up with this probability distribution Q_i

We're allowed to construct a probability distribution and multiply and divide by the same thing

Then:

$$\max_{\theta} \sum_i \log P(x^{(i)}; \theta)$$

$$= \sum_i \log \sum_{z^{(i)}} P(x^{(i)}, z^{(i)}; \theta)$$

$$= \sum_i \log \sum_{z^{(i)}} Q_i(z^{(i)}) \left[\frac{P(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} \right]$$

(where $Q_i(z^{(i)})$ is a probability distribution
i.e. $\left(\sum_{z^{(i)}} Q_i(z^{(i)}) \right) = 1$)

$$= \sum_i \log E_{z^{(i)} \sim Q_i} \left[\frac{P(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} \right]$$

So the way you compute the **expected value** of some function of $z^{(i)}$ is you sum over all the possible values of $z^{(i)}$, of the probability of $z^{(i)}$ times whatever that function is. So, this equation is just the **expected value** with respect to $z^{(i)}$ drawn from that Q_i distribution of that thing in the **orange** square brackets

Now, using the concave form of **Jensen's inequality**, we have that this is greater than or equal to:

$$\max_{\theta} \sum_i \log P(x^{(i)}; \theta)$$

$$= \sum_i \log \sum_{z^{(i)}} P(x^{(i)}, z^{(i)}; \theta)$$

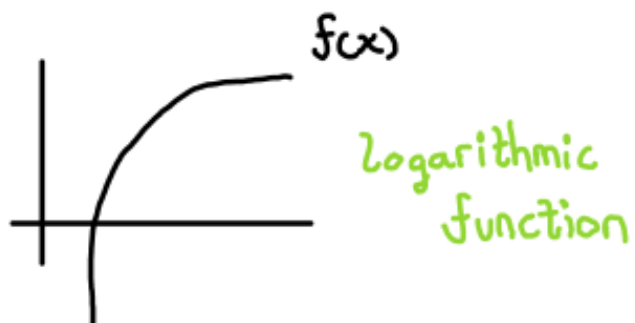
$$= \sum_i \log \sum_{z^{(i)}} Q_i(z^{(i)}) \left[\frac{P(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} \right]$$

(where $Q_i(z^{(i)})$ is a probability distribution
i.e. $\left(\sum_{z^{(i)}} Q_i(z^{(i)}) \right) = 1$)

$$= \sum_i \log E_{z^{(i)} \sim Q_i} \left[\frac{P(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} \right] \geq \sum_i E_{z^{(i)} \sim Q_i} \left[\log \frac{P(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} \right]$$

So this is a form of **Jensen's inequality** where:

$$f(E[x]) \geq E[f(x)]$$



So the **log function** is a **concave function** that looks like that

And so, using the form of Jensen's inequality with the signs reversed, **$f(E[x]) \geq E[f(x)]$** . So you get **log of expectation \geq expectation of the log**

Finally, we'll take this **expectation** and unpack it one more time:

$$\max_{\theta} \sum_i \log P(x^{(i)}; \theta) \quad \text{--- } \mathcal{L}(\theta)$$

$$= \sum_i \log \sum_{z^{(i)}} P(x^{(i)}, z^{(i)}; \theta)$$

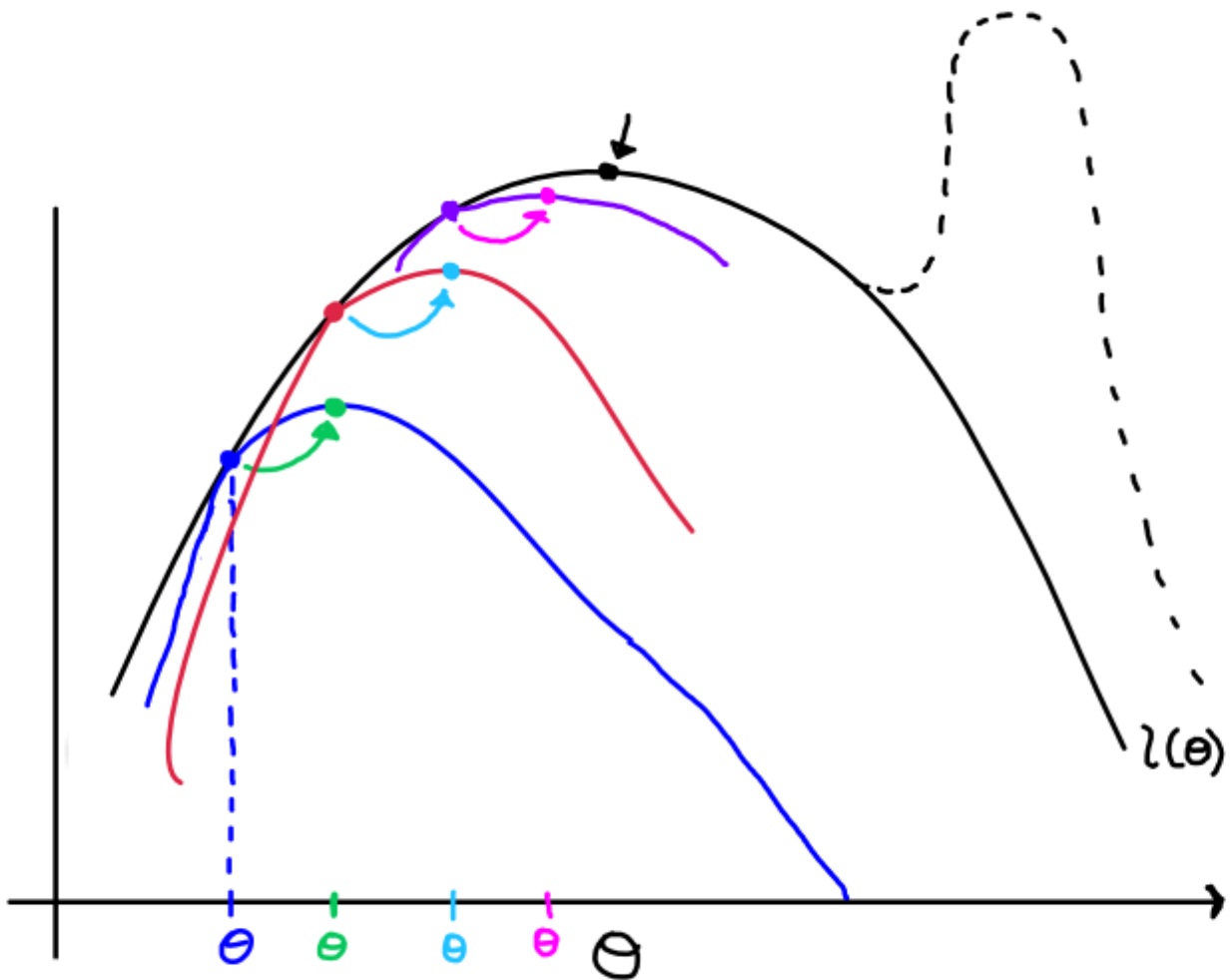
$$= \sum_i \log \sum_{z^{(i)}} Q_i(z^{(i)}) \left[\frac{P(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} \right]$$

(where $Q_i(z^{(i)})$ is a probability distribution
i.e. $\left(\sum_{z^{(i)}} Q_i(z^{(i)}) \right) = 1$)

$$= \sum_i \log E_{z^{(i)} \sim Q_i} \left[\frac{P(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} \right] \geq \sum_i E_{z^{(i)} \sim Q_i} \left[\log \frac{P(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} \right]$$

$$\rightarrow = \sum_i \sum_{z^{(i)}} Q_i(z^{(i)}) \log \frac{P(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})}$$

If you remember this picture:



What we wanted to do was to construct a function (construct this **blue curve** that's a lower bound for the **black curve**)

And if you view this formula here as a function of θ :

$$\rightarrow = \sum_i \sum_{z^{(i)}} Q_i(z^{(i)}) \log \frac{P(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})}$$

Your \mathbf{x} is just your data and \mathbf{z} is a variable you sum over. So this whole thing is the function of θ because \mathbf{x} 's are fixed, \mathbf{z} is just something you sum over, so this whole formula here is a function of the parameters θ

And what we've shown is that this formula here is a lower bound for the **log likelihood** for this:

$$\max_{\theta} \sum_i \log P(x^{(i)}; \theta) \quad \text{--- } \ell(\theta)$$

Let's say that z takes on values from **1** through **10**, so say you roll a 10-sided dice, and you want to compute the **expected value** of some function $g(z)$. Then the **expected value** of $g(z)$ is the sum over all the possible values of z of the probability that you get that z , times $g(z)$:

$$z \in \{1, \dots, 10\}$$

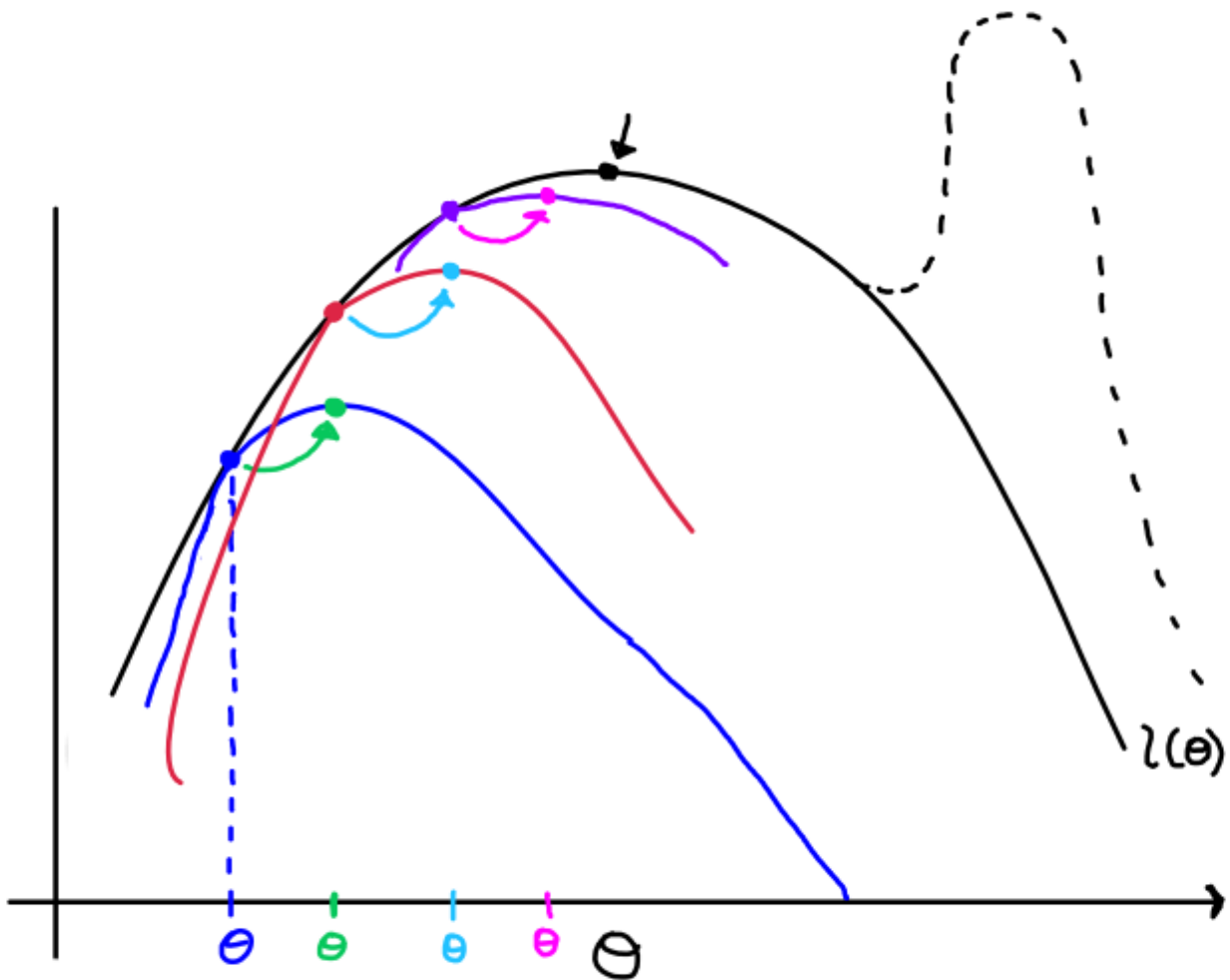
$$E[g(z)] = \sum_z \overbrace{P(z) g(z)}^{Q_i(z)}$$

$$E[z] = \sum_z P(z) z$$

So that's what the expected value is of a function of a random variable. And the **expected value** of z is the average of the random variable

In the notation that we have, the probability of z taking on different values is denoted by $Q_i(z)$

One of the things we want when constructing this **black** lower bound:



is we want that **blue** lower bound to be equal to the **black** function at this point (**blue** point)

This is actually how you guarantee that when you optimize the **blue** function, by improving on the **blue** function, you're improving on the **black** function. So we want this lower bound to be tight (the two functions be equal, tangent to each other)

So in other words, we want this inequality to hold with equality:

$$= \sum_i \log E_{z^{(i)} \sim Q_i} \left[\frac{P(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} \right] \geq \sum_i E_{z^{(i)} \sim Q_i} \left[\log \frac{P(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} \right]$$

$$\rightarrow = \sum_i \sum_{z^{(i)}} Q_i(z^{(i)}) \log \frac{P(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})}$$

So we want the left hand side and the right hand side to be equal for the current value of θ

On a given iteration of EM (with the current parameters equal to θ), we want:

$$\log E_{z^{(i)} \sim Q_i} \left[\frac{P(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} \right] = E_{z^{(i)} \sim Q_i} \left[\log \frac{P(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} \right]$$

We want the left and right hand sides to be equal to each other because that's what it means for the lower bound to be tight (for the **blue** curve to be exactly touching the **black** curve as we construct that lower bound)

For this to be true, we need the random variable inside to be a constant:

For this to hold, we need

$$\frac{P(x^{(i)}, z^{(i)})}{Q_i(z^{(i)})} = \text{constant}$$

Meaning that, no matter what value of $z^{(i)}$ you plug in, this should evaluate to the same value

In other words, the ratio between the numerator and denominator must be the same

So far, we have not yet specified how we choose this distribution for $z^{(i)}$. So far the only constraint we have is that Q_i has to be a probability distribution over $z^{(i)}$, but you could choose whatever distribution you want for $z^{(i)}$

It turns out that we can set $Q_i(z^{(i)})$ to be proportional to $P(x^{(i)}, z^{(i)}; \theta)$:

$$\text{Set } Q_i(z^{(i)}) \propto P(x^{(i)}, z^{(i)}; \theta)$$

↑
"to be proportional to"

This means that for any value of z (z indicates whether it is from **Gaussian 1** or **Gaussian 2**), the chance of **Gaussian 1** is proportional to the chance of **Gaussian 1** vs **Gaussian 2**. Whether $z^{(i)}$ takes on **1** or **2** is proportional to this

The Q_i 's need to sum to **1**, so one way to ensure that this is proportional to the right hand side is to just take the right hand side and normalize it to sum to **1**, and after a couple steps (that are in the lecture notes), you can show that this results in sending $Q_i(z^{(i)})$ to be equal to that posterior probability:

$$\sum_{z^{(i)}} Q_i(z^{(i)}) = 1$$

$$Q_i(z^{(i)}) = \frac{P(x^{(i)}, z^{(i)}; \theta)}{\sum_{z^{(i)}} P(x^{(i)}, z^{(i)}; \theta)}$$

...

$$= P(z^{(i)} | x^{(i)}; \theta)$$

If you want this to be a constant:

$$\frac{P(x^{(i)}, z^{(i)})}{Q_i(z^{(i)})} = \text{constant}$$

Meaning, whether you plugged in $z^{(i)} = 1$ or $z^{(i)} = 2$ or whatever, this evaluates to the same constant, the only way to do that is to make sure the numerator and denominator are proportional to each other:

$$\text{Set } Q_i(z^{(i)}) \propto P(x^{(i)}, z^{(i)}; \theta)$$

↑
"to be proportional to"

And because $Q_i(z^{(i)})$ is a density that must sum to **1**, one way to make sure they're proportional is to just set this to be equal to the right hand side but normalize the sum to **1**

To summarize, this gives us the **EM algorithm**:

E-step: ↙ $w_j^{(i)}$

$$\text{Set } Q_i(z^{(i)}) := P(z^{(i)} | x^{(i)}; \theta)$$

M-step:

$$\theta := \arg \max_{\theta} \sum_i \sum_{z^{(i)}} Q_i(z^{(i)}) \log \frac{P(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})}$$

In the **E-step**, we're going to set $Q_i(z^{(i)})$ to be equal to that, and previously this was the $w_j^{(i)}$'s. So previously we were storing these probabilities in variables you call $w_j^{(i)}$'s

In the **M-step**, we're going to take that lower bound that we constructed, which is this function, and maximize it with respect to θ


Remember, in the **M-step**, we constructed this thing on the right hand side as a lower bound for the **log likelihood**. And so, for the fixed value of **Q**, you can maximize this with respect to **Θ** and that updates the **Θ** (maximizing the **blue** lower bound). That's what the **M-step** does

And if you iterate these two steps, then you find that this should converge to a **local optima**


The obvious question, why don't we try to maximize the **log likelihood** $l(\Theta)$ directly?:

$$\max_{\Theta} l(\Theta)$$

It turns out that if you take the **mixture of Gaussians model**, try to take derivatives of this, and set derivatives equal to **0**, there's no known way to solve for the value of **Θ** that maximizes the **log likelihood**. But you find that for the **mixture of Gaussians model** and for many models, including **factor analysis**, if you actually plug in the **mixture of Gaussians model** for **P** and take derivatives, set derivatives equal to **0**, and solve, you will be able to find an analytic solution to maximize this **M-step** and it'll be exactly what we had worked out in the early derivation of the **EM algorithm**

E-step: 

$$\text{Set } Q_i(z^{(i)}) := P(z^{(i)} | x^{(i)}; \Theta)$$

M-step: 

$$\Theta := \arg \max_{\Theta} \sum_i \sum_{z^{(i)}} Q_i(z^{(i)}) \log \frac{P(x^{(i)}, z^{(i)}; \Theta)}{Q_i(z^{(i)})}$$

This derivation shows that the **EM algorithm** is a **maximum likelihood estimation algorithm** with optimization solved by constructing lower bounds and optimizing lower bounds

