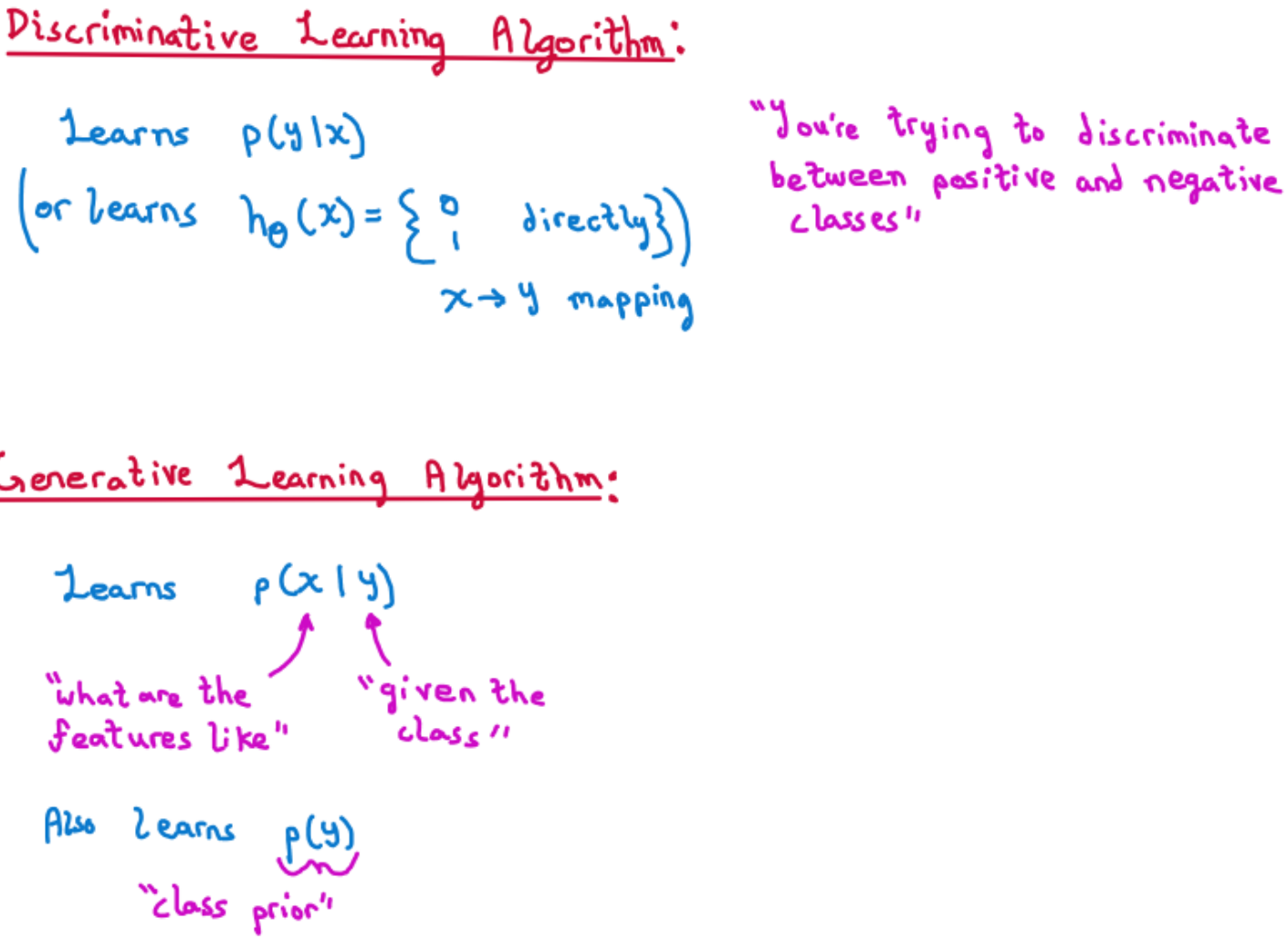
Lecture 5 [GDA & Naive Bayes]

Generative Learning Algorithms:

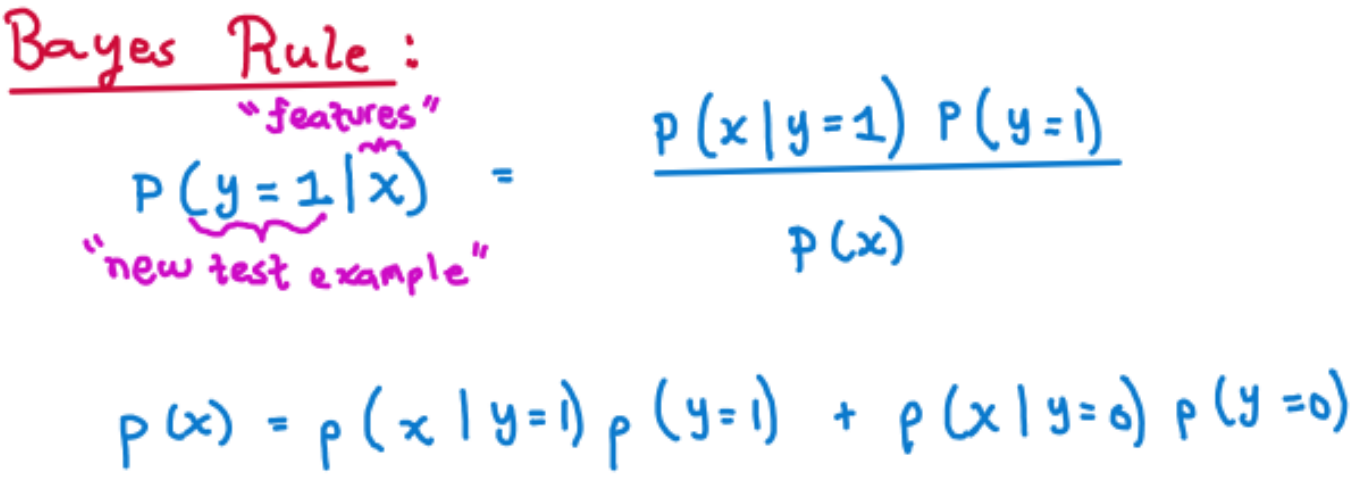
Logistic Regression is really searching for a decision boundary that separates positive and negative examples

Rather than looking at two classes and trying to find the separation (such as with a Discriminative learning algorithm like Logistic Regression which would use ***Gradient Descent*** to search for a line that separates the positive and negative examples), instead the algorithm is going to look at the classes one at a time and try to build a model for what each class looks like

Rather than looking at both classes simultaneously and searching for a way to separate them, a *Generative Learning Algorithm* instead builds a model of what each of the classes looks like (kind of almost in isolation), and then at test time it evaluates a new example against the models and tries to see which of the models it matches more closely against



Using ***Bayes Rule***, if you can build a model (calculate numbers for both of the following quantities) for p(x|y) and for p(y), when you have a new test example with features x, you can then calculate the chance of y being equal to 1

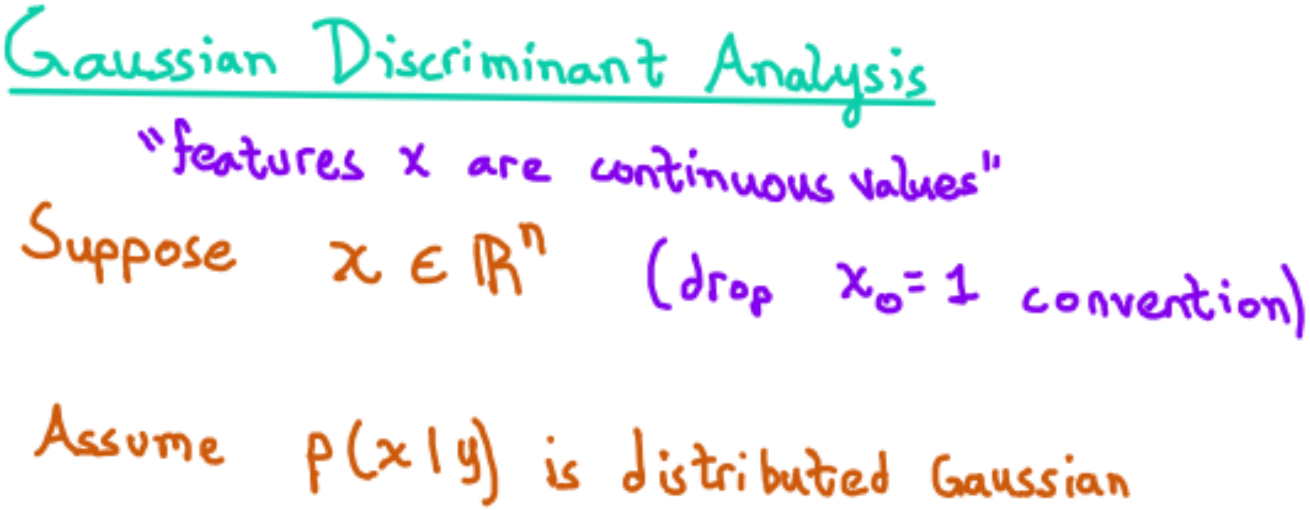


If you've learned both p(x|y) and p(y) , you can plug them into ***Bayes Rule*** to calculate p(y=1|x)

This is the framework we'll use to build ***Generative Learning Algorithms***

Two examples of Generative Learning Algorithms are one for ***Continous Value Features*** (which is used for things like tumor classification) and one for ***Discrete Features*** (which you can use for building something like an email spam filter)

Gaussian Discriminant Analysis (GDA):



In other words, conditioned on the tumors being malignant, the distribution of the features(size, cell adhesion, etc) is Gaussian

Multivariate Gaussian Distribution:

The Gaussian is a bell-shaped curve

A multivariate gaussian is the generalization of this bell-shaped curve over a 1-dimensional random variable to multiple random variables at the same time; to vector value random variables rather than a uni-variate random variable

Gaussian Discriminant Analysis

"features x are continuous values"

Suppose $x \in \mathbb{R}^n$ (drop $x_0 = 1$ convention)

Assume $p(x|y)$ is distributed Gaussian

If: $z \sim \mathcal{N}(\vec{\mu}, \Sigma)$ $z \in \mathbb{R}^n$

("is distributed", "Gaussian", "mean vector mu", "co-variance matrix sigma")

$\vec{\mu} \in \mathbb{R}^n$, $\Sigma \in \mathbb{R}^{n \times n}$

"probability density function" "for a Gaussian"

$$p(z) = \frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)\right)$$

"expected value"

$$E[z] = \mu$$

"co-variance"

$$\text{Cov}(z) = E[(z-\mu)(z-\mu)^T]$$

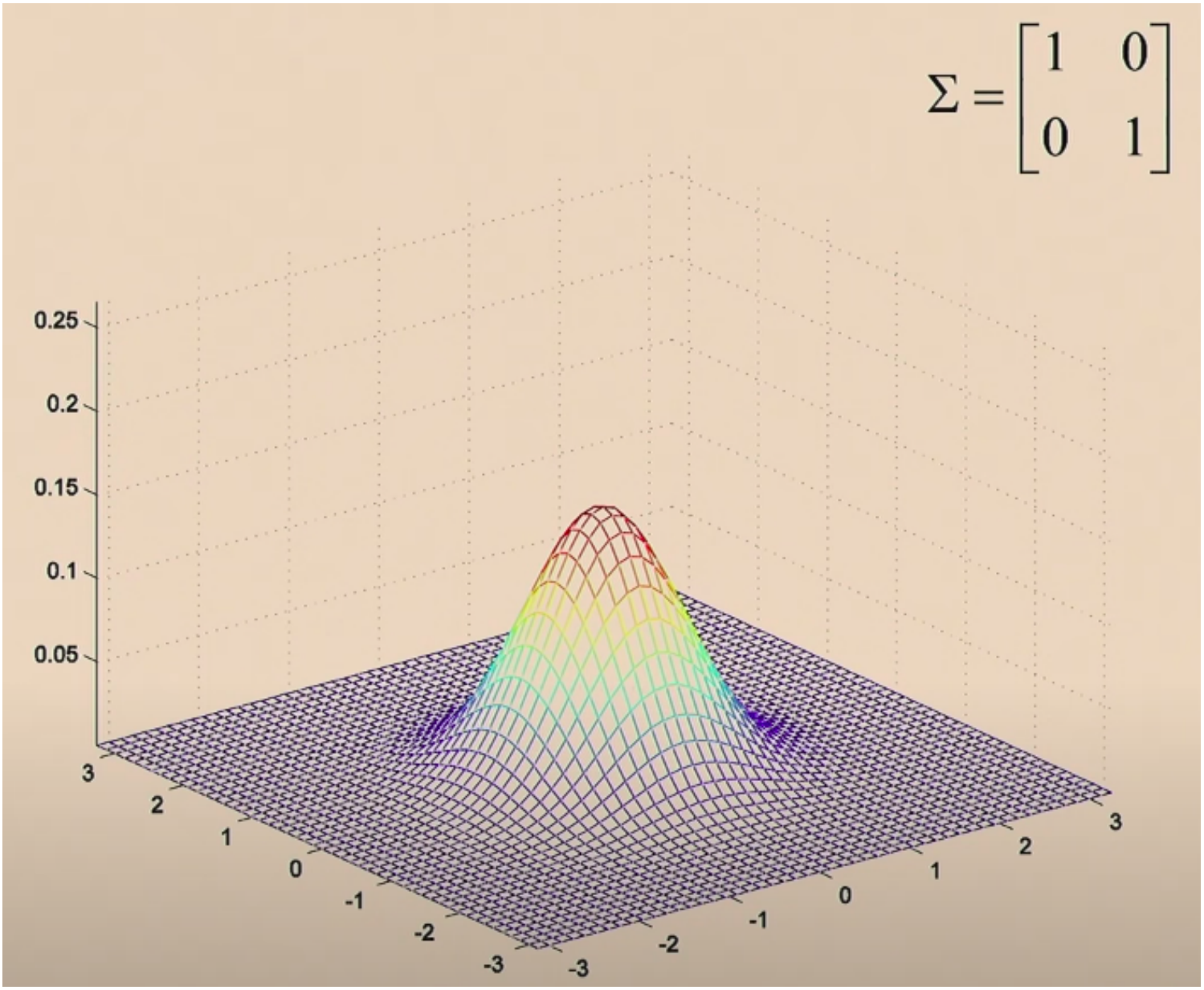
$$= Ezz^T - (Ez)(Ez)^T$$

$E[z] = Ez$

"You can omit brackets"

The multivariate Gaussian Density has two parameters, **μ** and **Σ**. They control the mean and the variance of this density

Visual Example:



This is a 2-dimensional Gaussian bump and for now the mean parameter is set to 0

μ is the 2-dimensional paramter [0. 0] which is why this Gaussian bump is centered at 0, and the co-variance matrix Σ is identity matrix

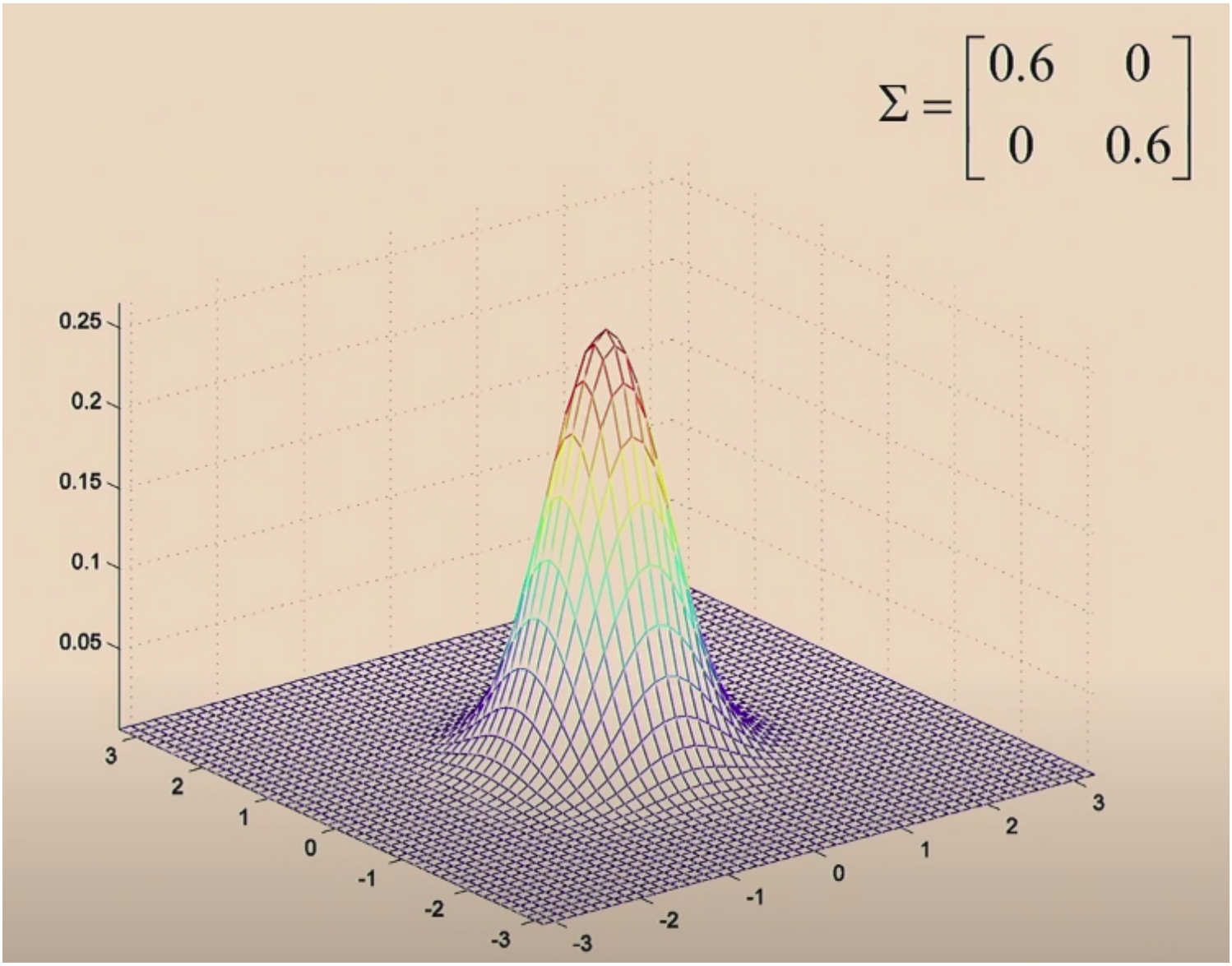
This is also called the ***Standard Gaussian Distribution***, which means mean 0 and the co-variance equals to the identity

To shrink the co-variance matrix, multiply it by a number less than 1. That should shrink the variance; reduce the variability of distributions

If you do that, the probability density function becomes taller

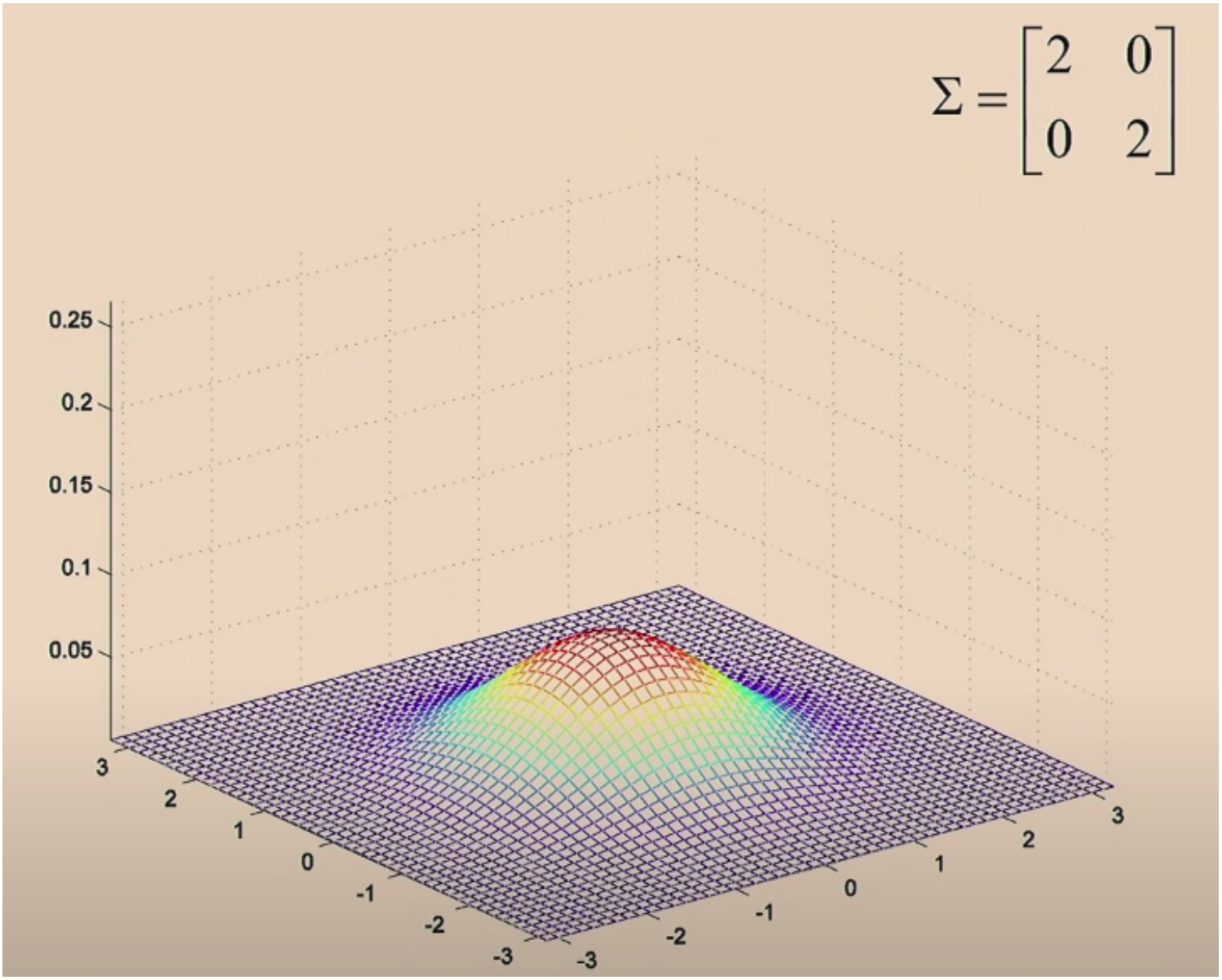
Since this is a probability density function, it always integrates to 1. The area under the curve is 1

So by reducing the co-variance from the identity to 0.6 times the identity, it reduces the spread of the Gaussian density and also makes it tall as a result

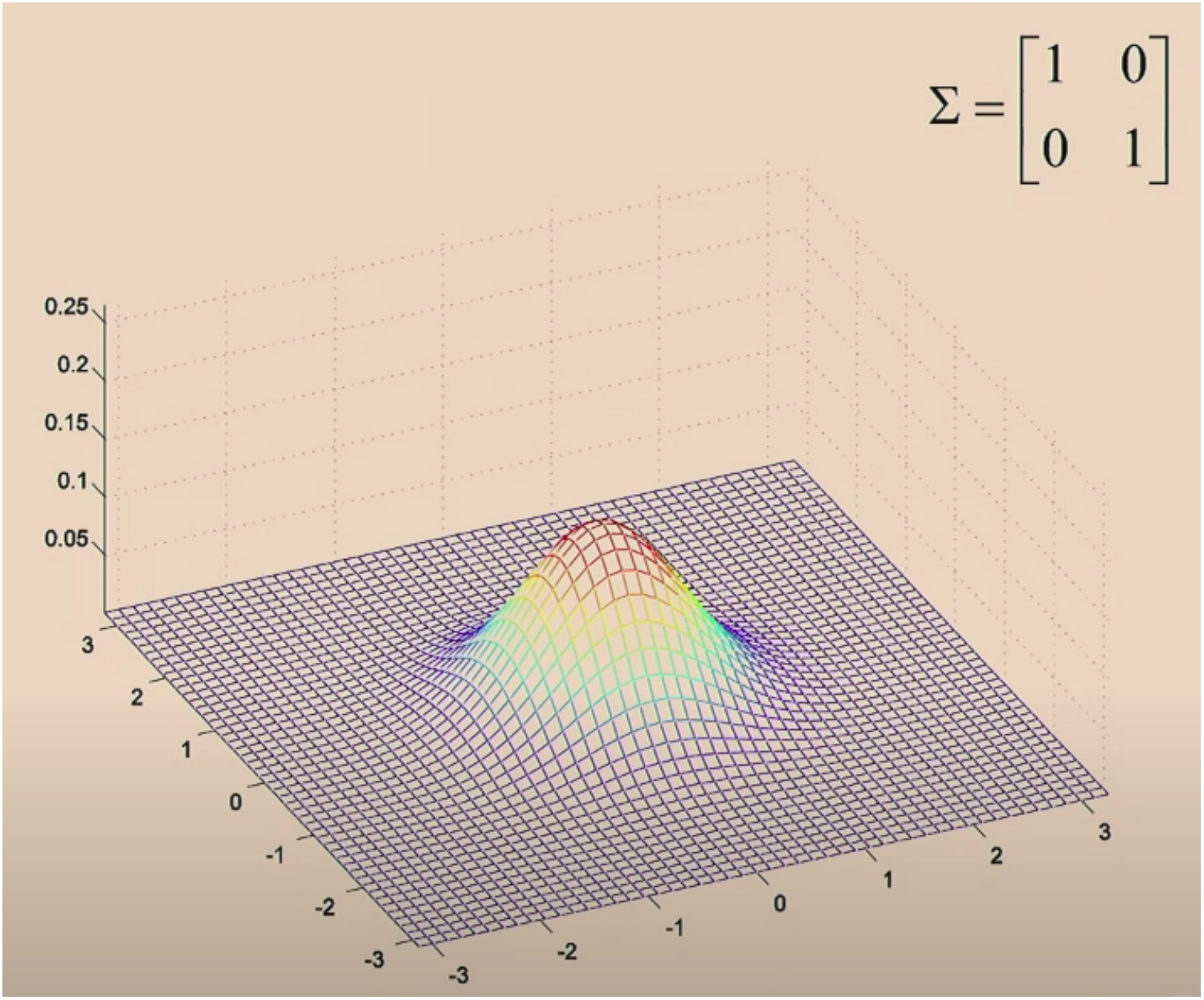


To make it wider, make the co-variance two times the identity

You will end up with a wider distribution where the values of the axes increases the variance of the density

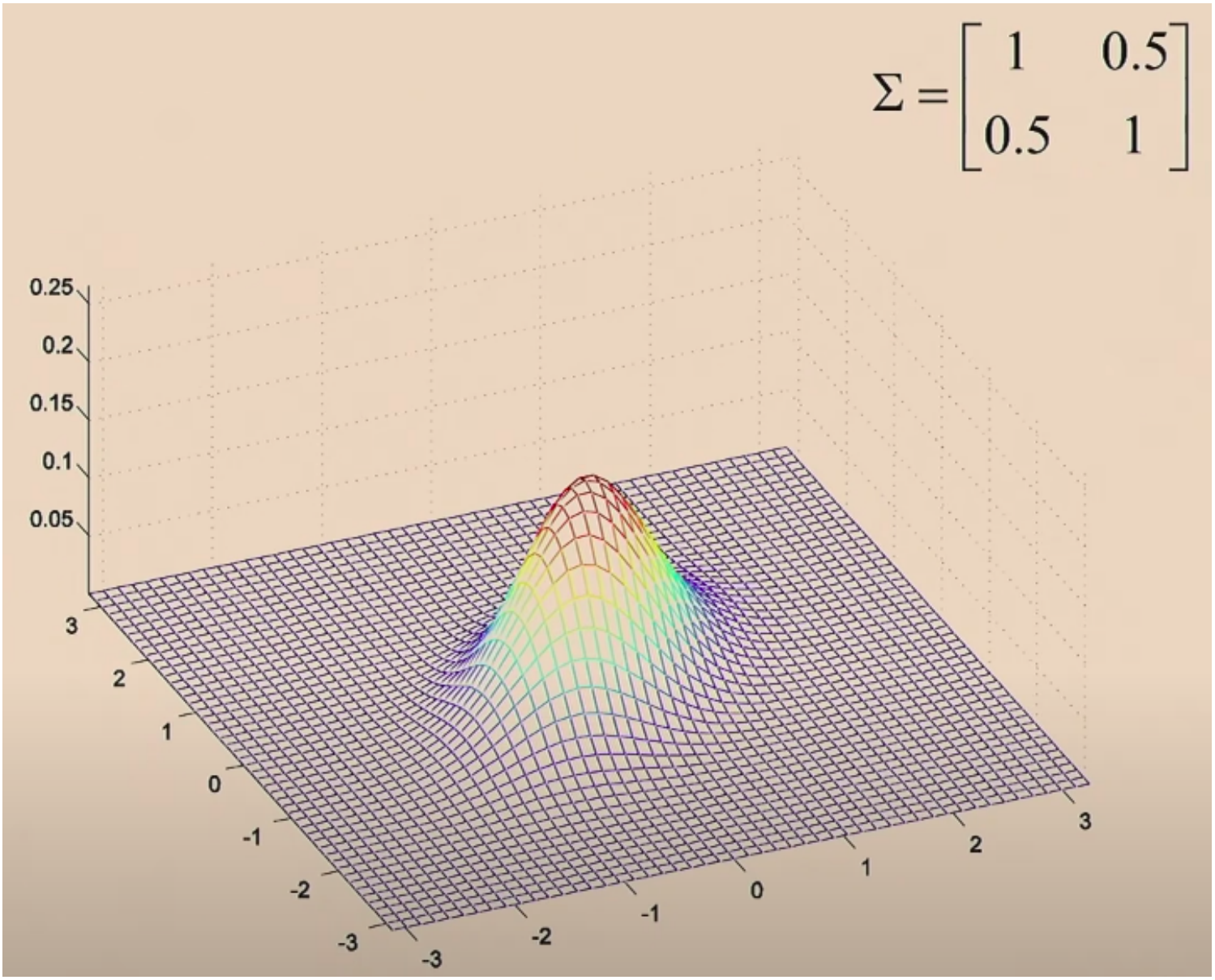


Back to the Standard Gaussian Distribution:



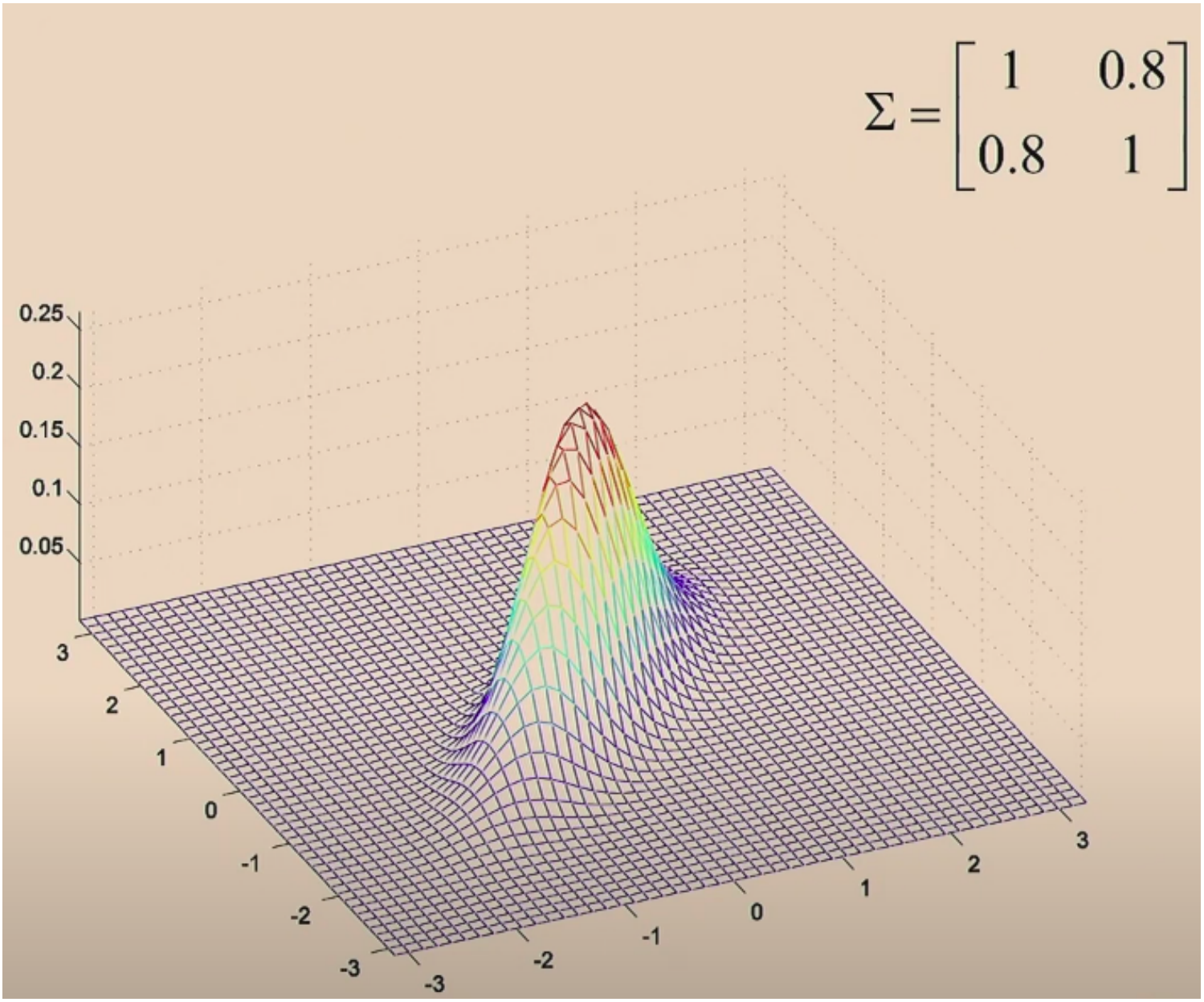
The off-diagonal entries are currently 0, so in this Gaussian density the off-diagonal entries are [0, 0]

Increasing it to 0.5 gives us:



The Guassian density goes from a round shape to a slightly narrower shape

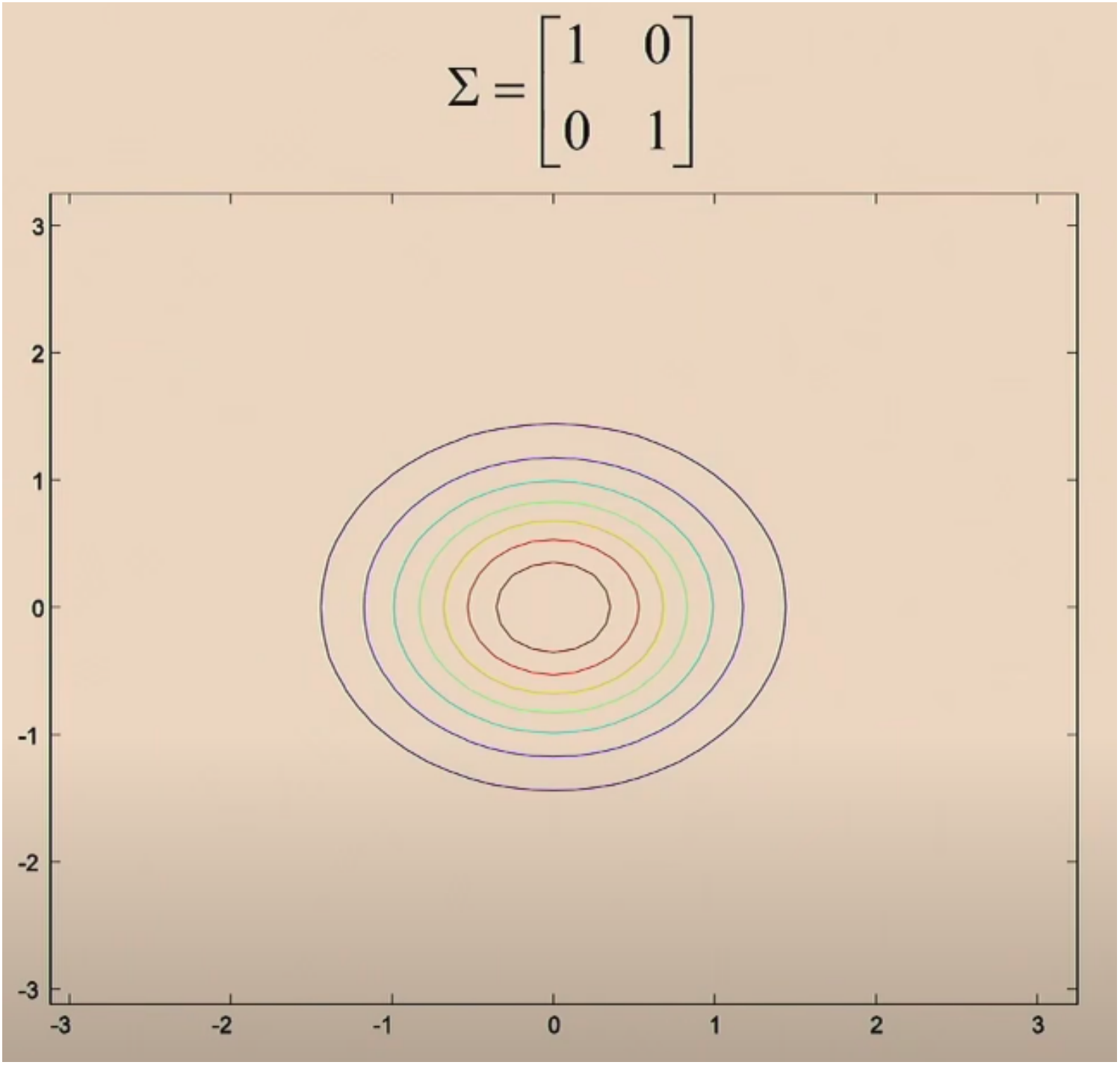
Increasing it further to [0.8, 0.8] gives us:



The density ends up looking like this where now, it's more likely that z1 and z2 (bottom axes) are now positively correlated

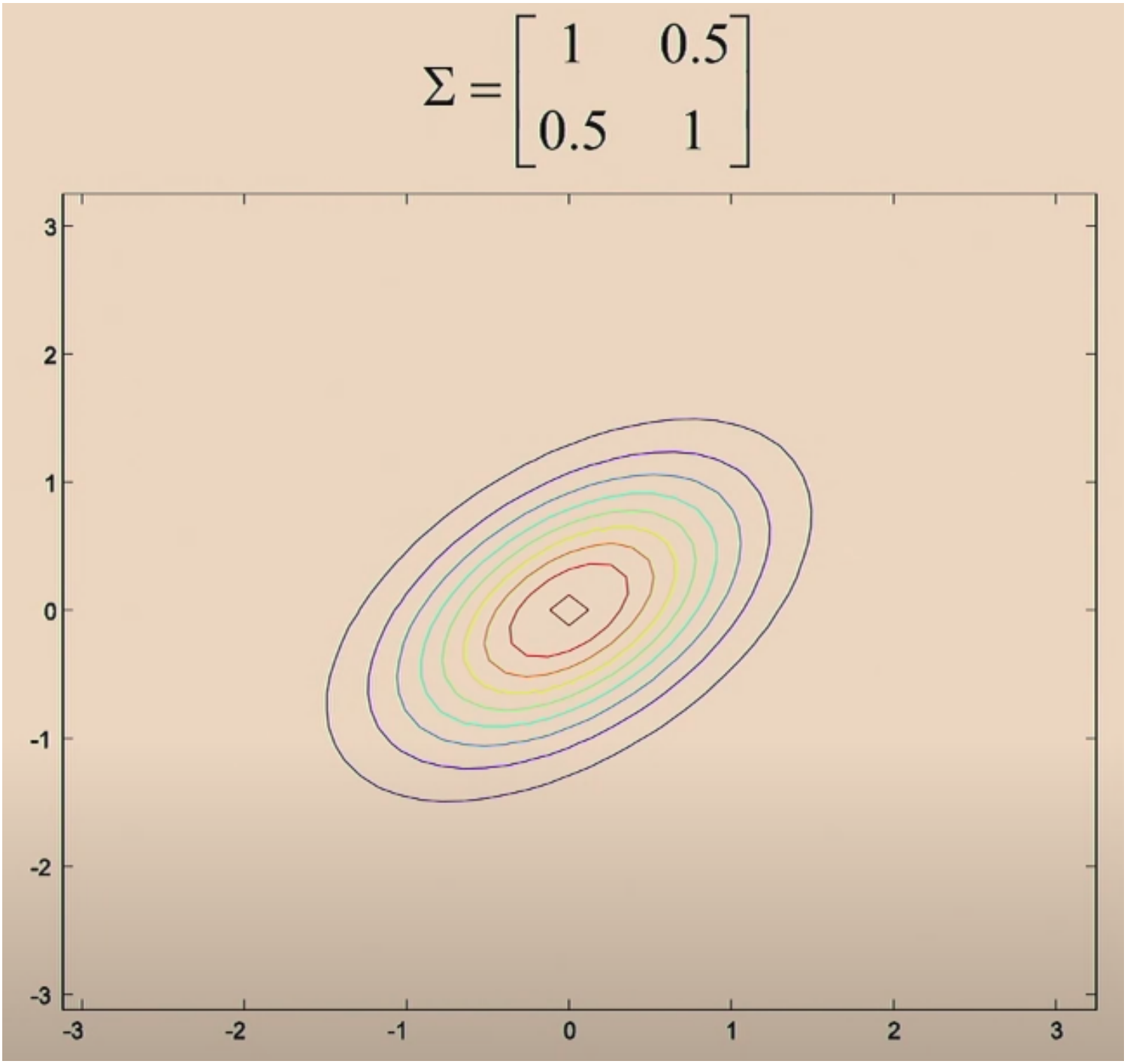
Now looking at contours of these Gaussian densities:

This is the contours of the Gaussian density when the co-variance matrix is the identity matrix:

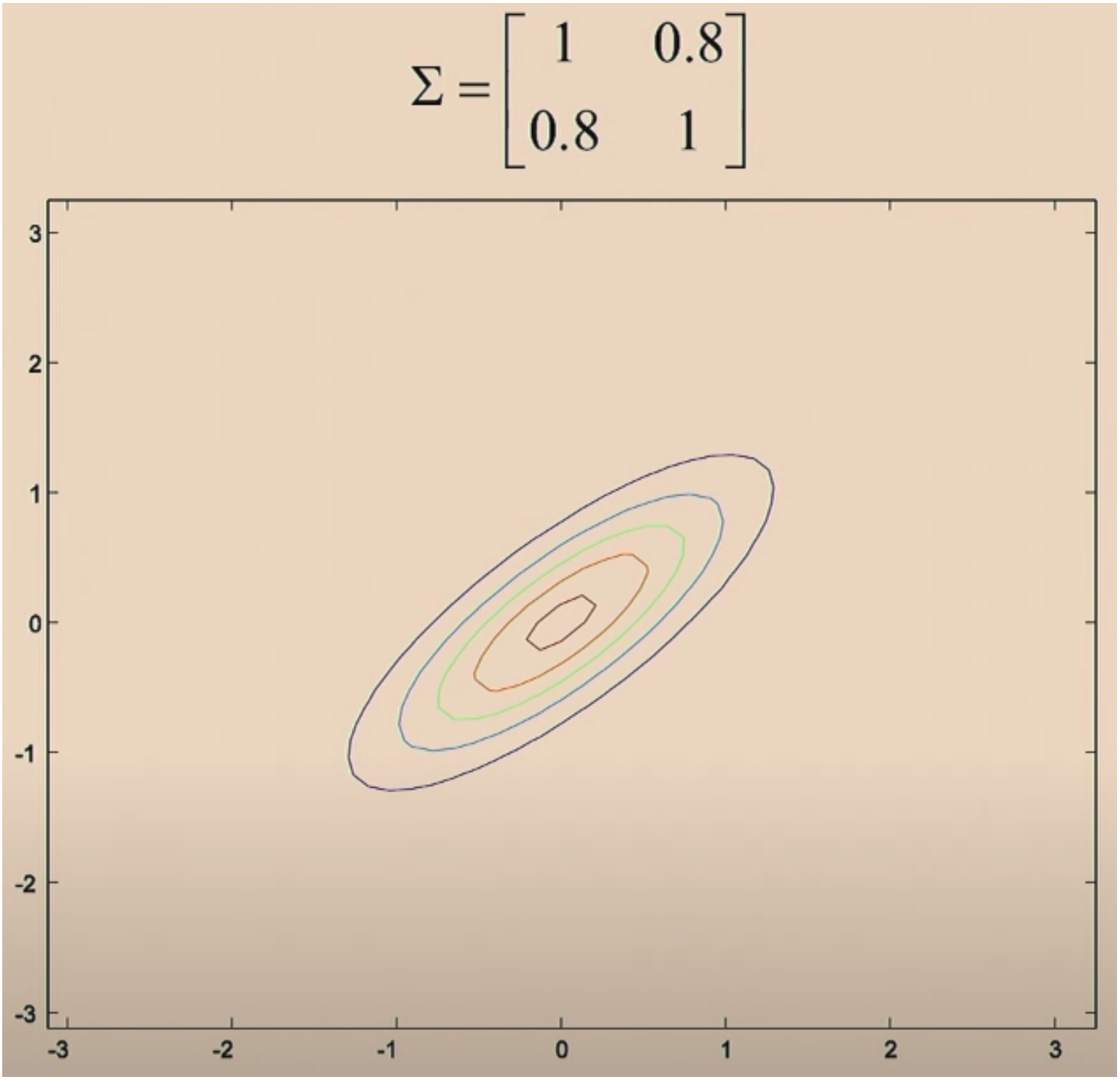


When the co-variance matrix is the identity matrix, z1 and z2 are uncorrelated and the contours of the Gaussian bump (of the Gaussian density) look like round circles

If you increase the off-diagonal then it looks like this:

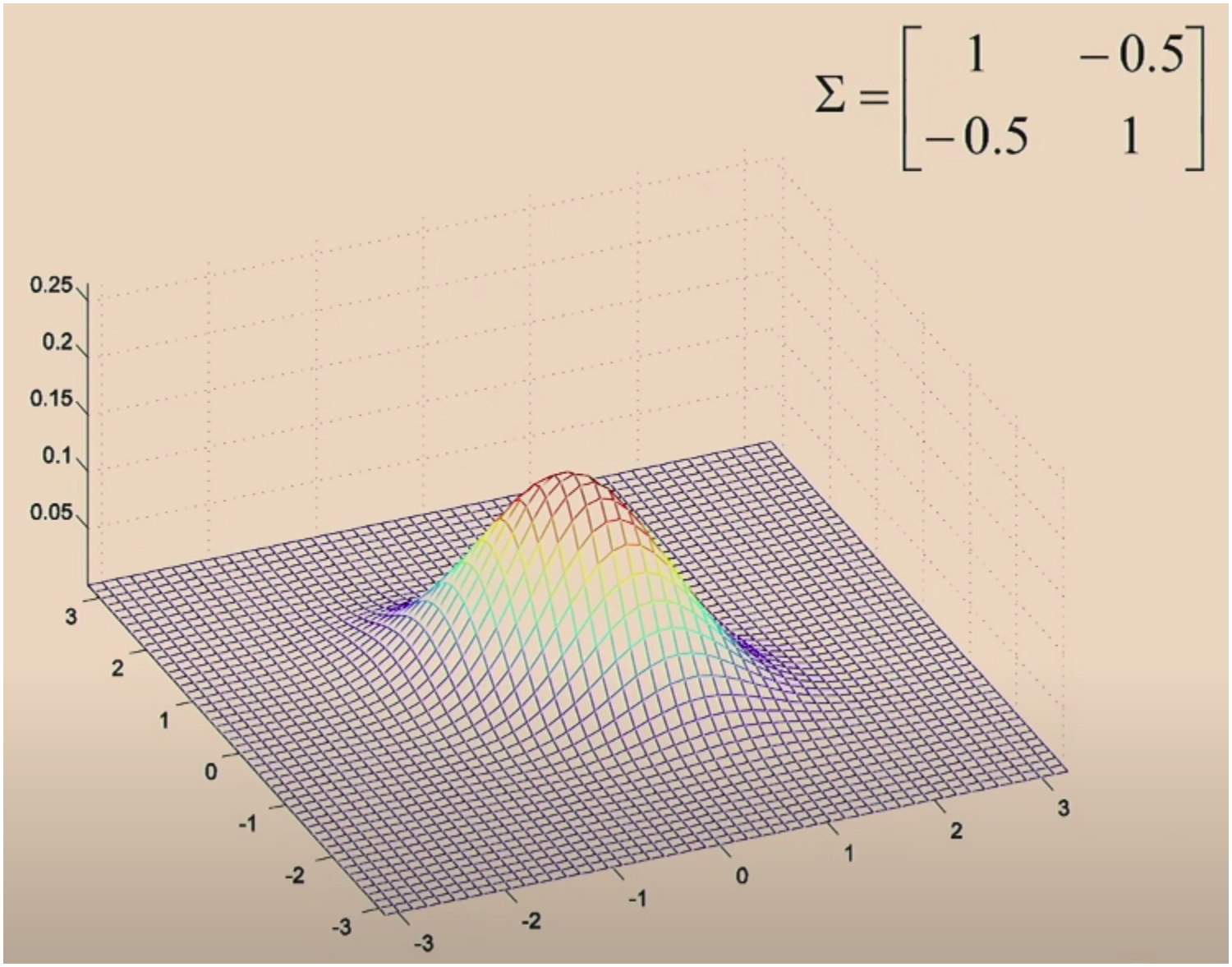


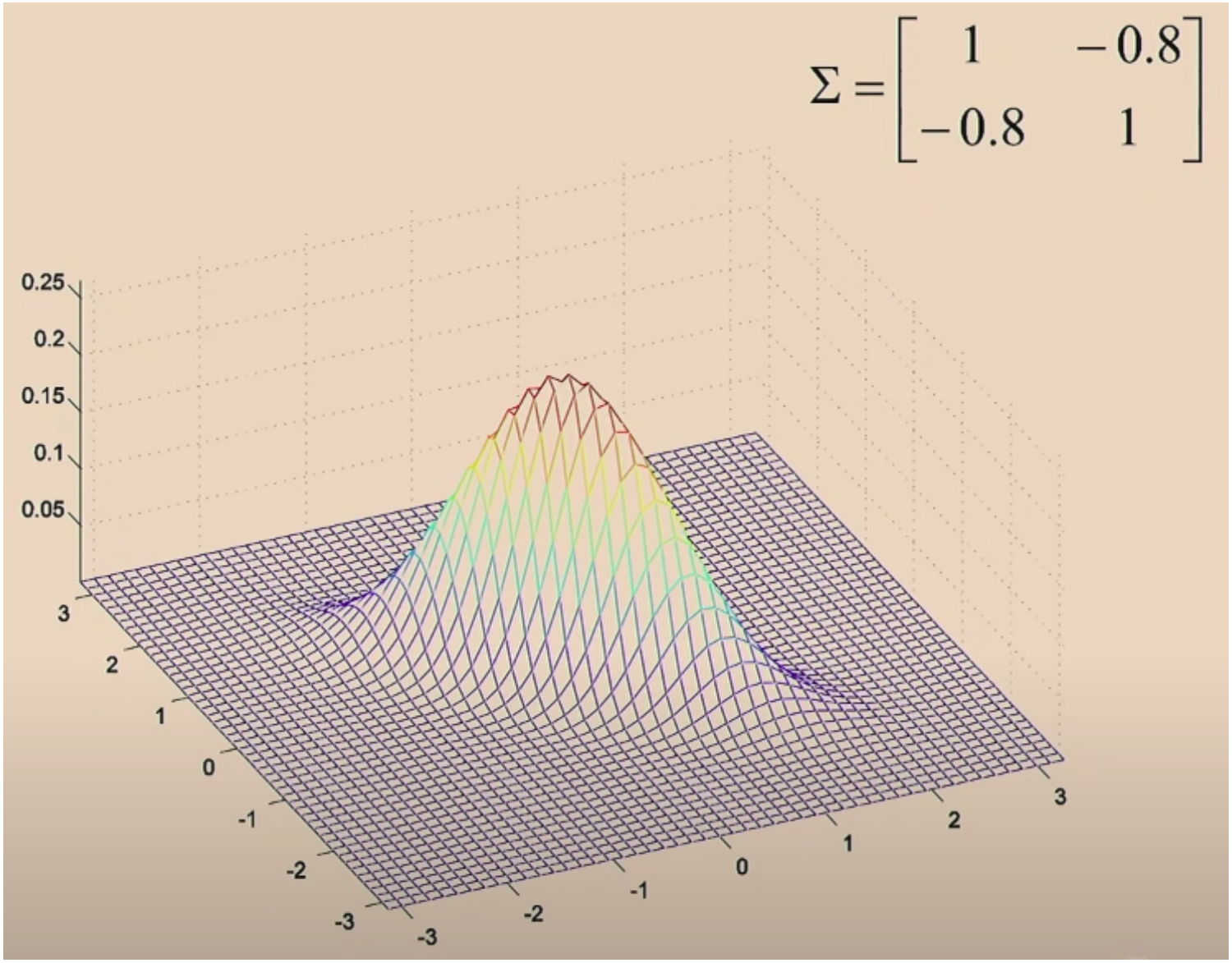
If you increase it further to [0.8, 0.8] then it looks like this:



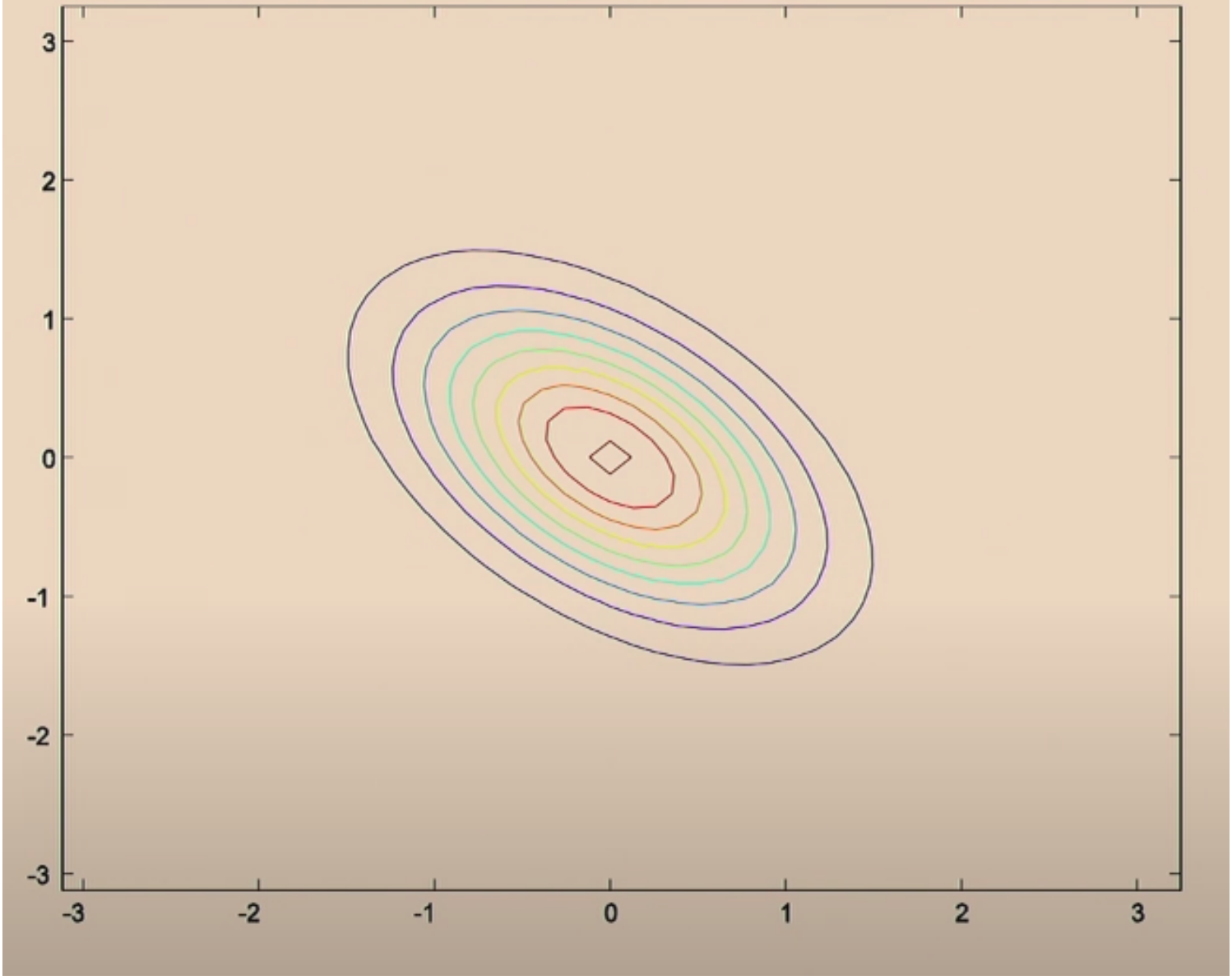
Now, most of the probability density function places value on z1 and z2 being positively correlated

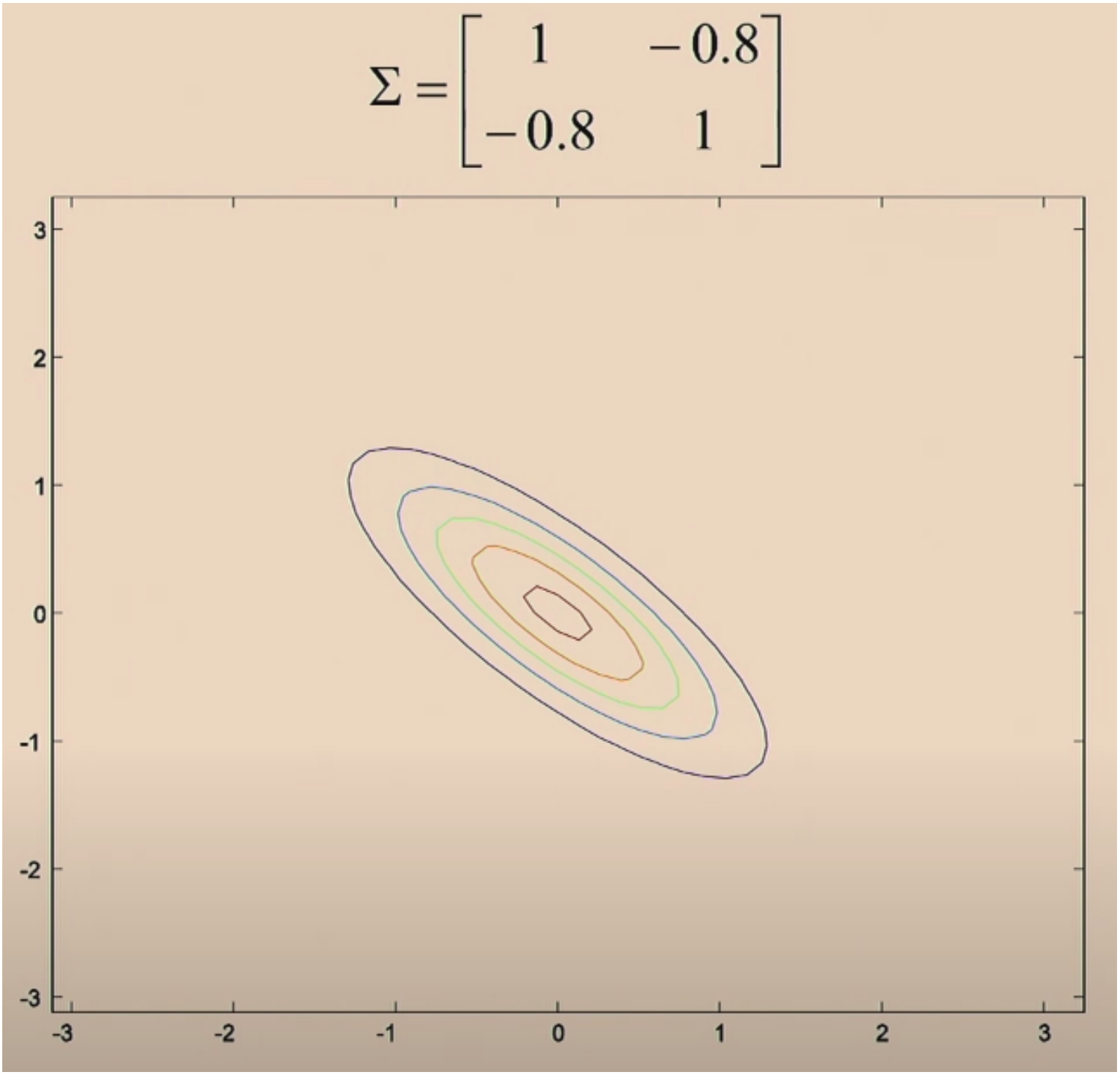
What happens if we set the off-diagonal elements to negative values?


$\Sigma = \begin{bmatrix} 1 & -0.5 \\ -0.5 & 1 \end{bmatrix}$
0.25
0.2
0.15
0.1
0.05
3
2
1
0
-1
-2
-3
-3
-2
-1
0
1
2
3


$\Sigma = \begin{bmatrix} 1 & -0.8 \\ -0.8 & 1 \end{bmatrix}$
0.25
0.2
0.15
0.1
0.05
3
2
1
0
-1
-2
-3
-3
-2
-1
0
1
2
3

$$\Sigma = \begin{bmatrix} 1 & -0.5 \\ -0.5 & 1 \end{bmatrix}$$

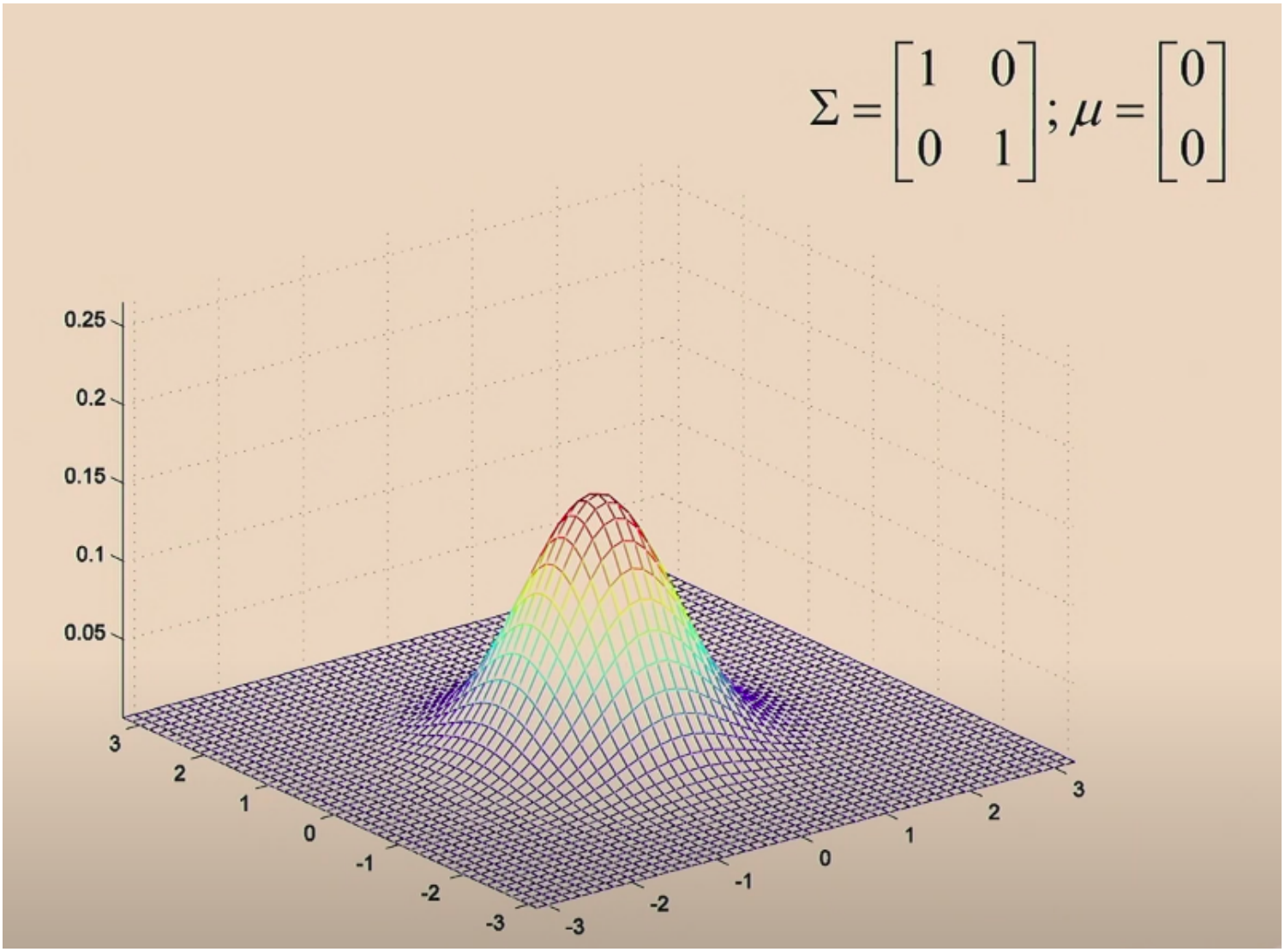




As you endow the two random variables with negative correlation, you end up with these types of probability density functions and contours (where it's now slanted the other way)

So now z1 and z2 have a negative correlation

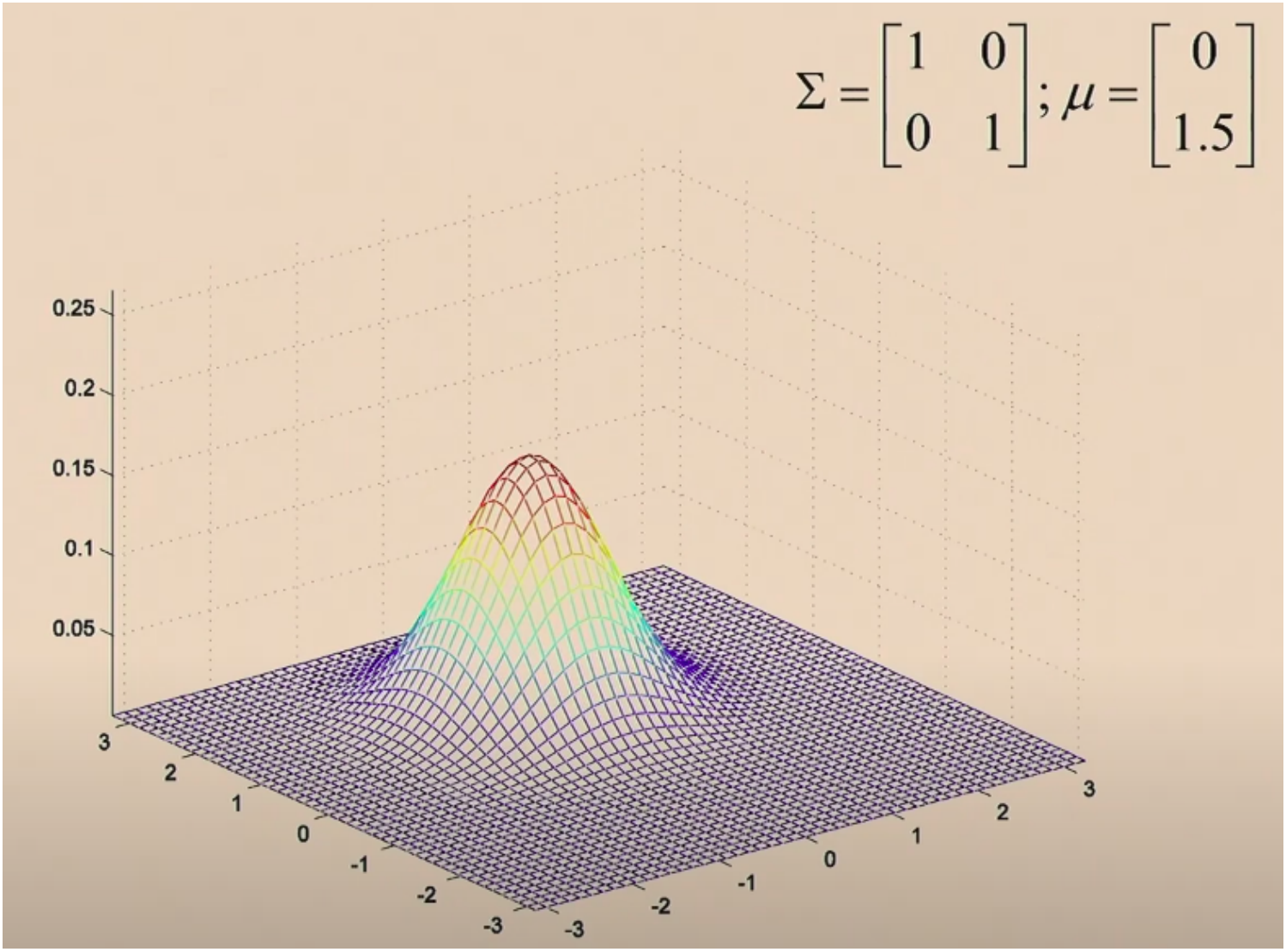
So far we've been keeping the mean vector as 0 and just varying the co-variance matrix:

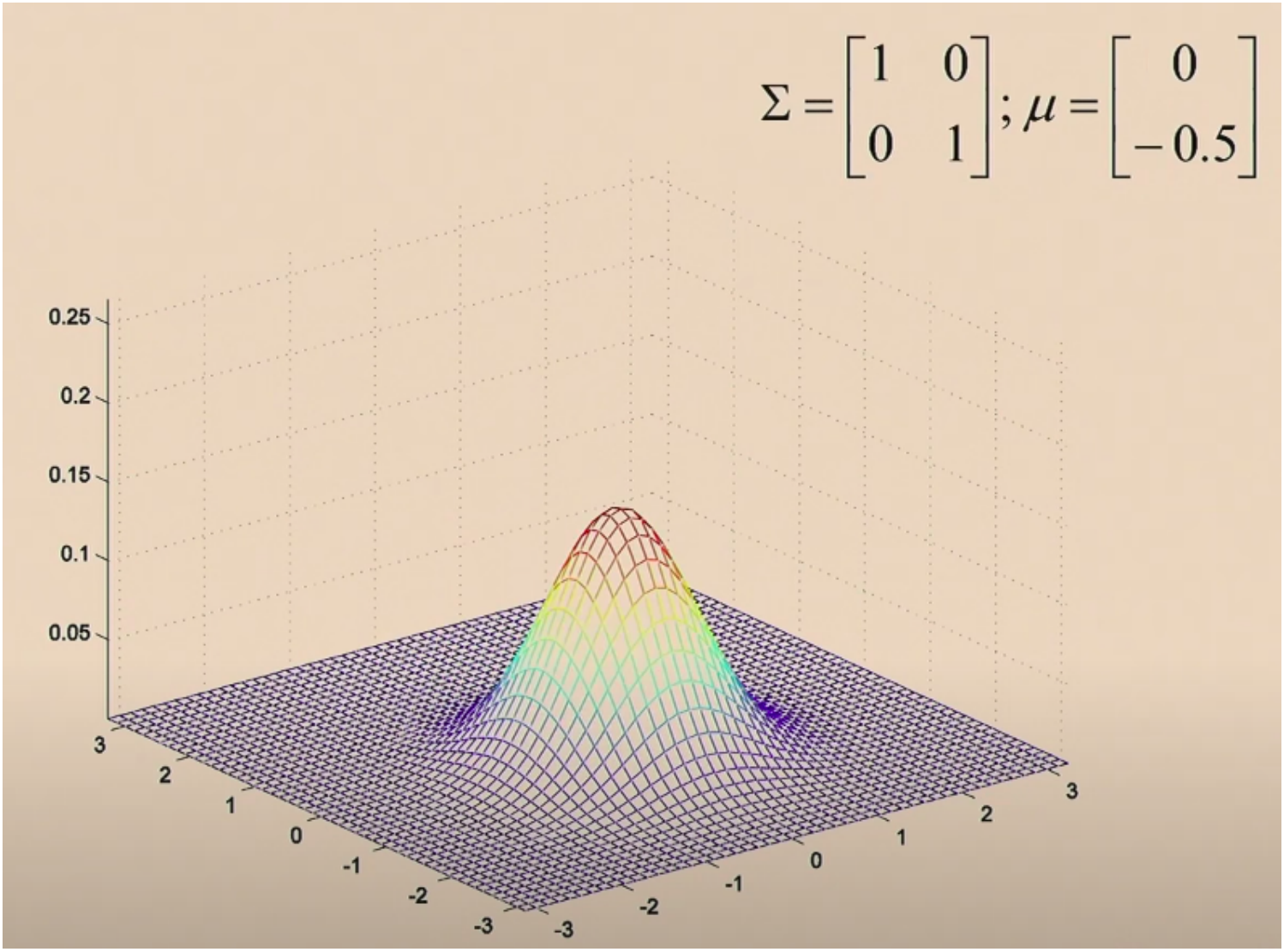


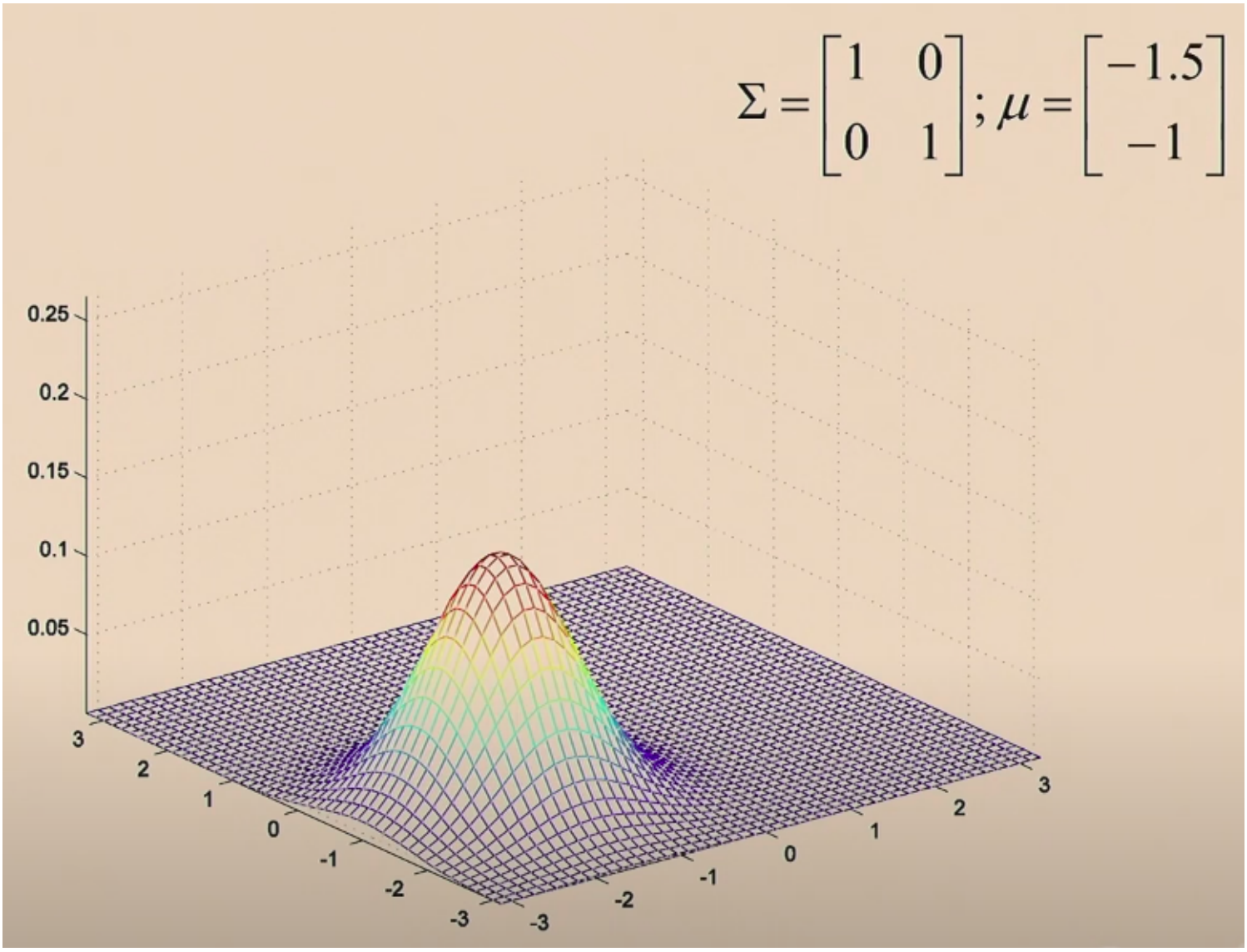
Every co-variance matrix is symmetric *#*

This Standard Gaussian has mean 0 (Gaussian bump is centered at [0, 0] because μ is [0, 0])

Moving μ around gives you:


$\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}; \mu = \begin{bmatrix} 0 \\ 1.5 \end{bmatrix}$
0.25
0.2
0.15
0.1
0.05
3
2
1
0
-1
-2
-3
-3
-2
-1
0
1
2
3


$\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}; \mu = \begin{bmatrix} 0 \\ -0.5 \end{bmatrix}$
0.25
0.2
0.15
0.1
0.05
3
2
1
0
-1
-2
-3
-3
-2
-1
0
1
2
3

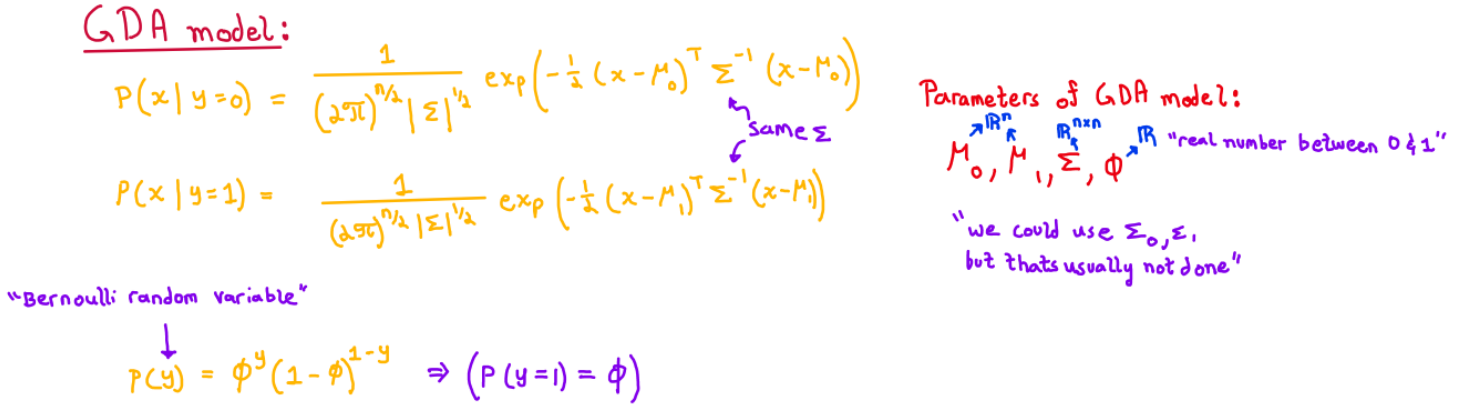


So by varying the value of μ, you could also shift the center of the Gaussian density around

As you vary the parameters, the mean and the co-variance matrix of the 2D Gaussian density, those are probability density functions you can get as a result of changing μ and Σ

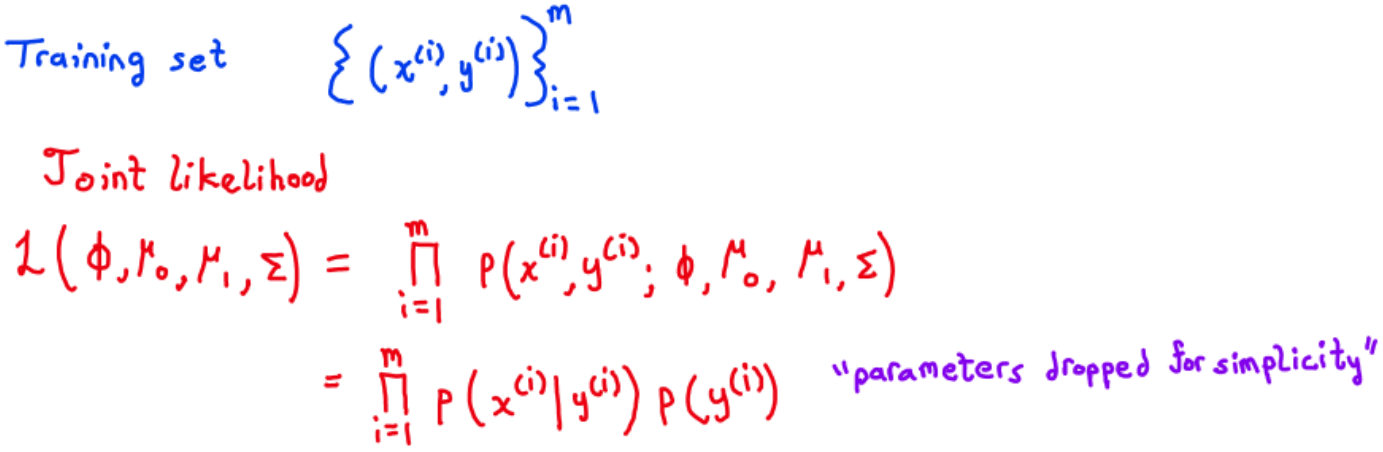
GDA Model:

We're going to assume that the two Gaussians, for the positive and negative classes, have the same co-variance matrix but they have different means. You don't have to make this assumption, but this is the way it's most commonly done



Fitting the parameters:

In order to fit these parameters, we're going to maximize the joint likelihood



Generative & Discriminative comparison:

The big difference between a *Generative Learning Algorithm* like this compared to a *Discriminative Learning Algorithm* is that the cost function you maximize is this ***Joint Likelihood*** shown above. Whereas for a Discriminative Learning algorithm we are maximizing the ***Conditional Likelihood***

Training set $\{(x^{(i)}, y^{(i)})\}_{i=1}^{m}$

Joint likelihood

$$L(\phi, \mu_0, \mu_1, \Sigma) = \prod_{i=1}^{m} p(x^{(i)}, y^{(i)}; \phi, \mu_0, \mu_1, \Sigma)$$

$$= \prod_{i=1}^{m} p(x^{(i)} | y^{(i)}) p(y^{(i)})$$ "parameters dropped for simplicity"

Discriminative:

Conditional likelihood

$$L(\theta) = \prod_{i=1}^{m} p(y^{(i)} | x^{(i)}; \theta)$$

So the big difference between these two cost functions is that for ***Logistic Regression*** or ***Linear Regression*** and ***Generalized Linear Models***, you were trying to choose parameters Θ that maximize p(y | x). But for ***Generative Learning Algorithms***, we're going to try to choose parameters that maximize p(x, y)

Maximum Likelihood Estimation:

$$\max_{\phi, \mu_0, \mu_1, \Sigma} \underbrace{l(\phi, \mu_0, \mu_1, \Sigma)}_{= \log L(\ldots)}$$

$$\phi = \frac{\sum_{i=1}^{m} y^{(i)}}{m} = \frac{\sum_{i=1}^{m} 1\{y^{(i)} = 1\}}{m}$$

$1\{\text{true}\} = 1$

$1\{\text{false}\} = 0$

Φ is the estimate of probability of y being equal to 1

This notation is an *indicator function* where indicator {$\mathbf{y^{(i)}}$ = 1} returns 0 or 1 depending on whether the thing inside is true

Maximum Likelihood Estimation:

$$\max_{\phi, \mu_0, \mu_1, \Sigma} \underbrace{l(\phi, \mu_0, \mu_1, \Sigma)}_{= \log \mathcal{L}(\ldots)}$$

$$\phi = \frac{\sum_{i=1}^{m} y^{(i)}}{m} = \frac{\sum_{i=1}^{m} 1\{y^{(i)} = 1\}}{m_1}$$

$1\{\text{true}\} = 1$

$1\{\text{false}\} = 0$

$$\mu_0 = \frac{\sum_{i=1}^{m} 1\{y^{(i)} = 0\} x^{(i)}}{\sum_{i=1}^{m} 1\{y^{(i)} = 0\}}$$

← sum of feature vectors for examples with $y=0$

← # of examples with $y=0$

The Maximum Likelihood Estimate of the mean of all of the features for the benign tumors can be found by taking all the benign tumors in your training set and just take the average. In other words, look at all of your negative examples and average their features

$$\phi = \frac{\sum_{i=1}^{m} y^{(i)}}{m} = \frac{\sum_{i=1}^{m} 1\{y^{(i)} = 1\}}{m_1}$$

$1\{\text{true}\} = 1$

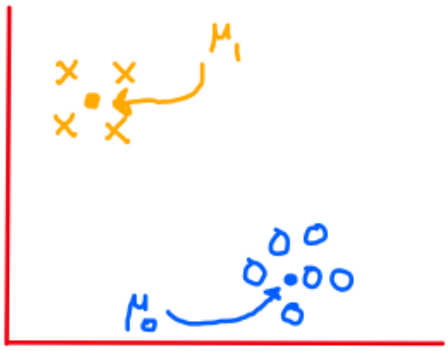
$1\{\text{false}\} = 0$

$$\mu_0 = \frac{\sum_{i=1}^{m} 1\{y^{(i)} = 0\} x^{(i)}}{\sum_{i=1}^{m} 1\{y^{(i)} = 0\}}$$

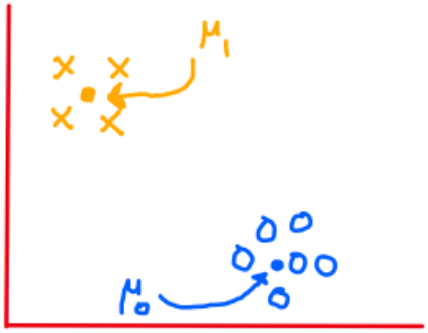
← sum of feature vectors for examples with $y=0$

← # of examples with $y=0$

$$\mu_1 = \frac{\sum_{i=1}^{m} 1\{y^{(i)} = 1\} x^{(i)}}{\sum_{i=1}^{m} 1\{y^{(i)} = 1\}}$$



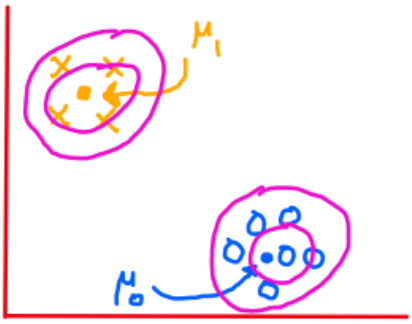
$$\mu_1 = \frac{\sum_{i=1}^{m} 1\{y^{(i)}=1\} x^{(i)}}{\sum_{i=1}^{m} 1\{y^{(i)}=1\}}$$



$$\Sigma = \frac{1}{m} \sum_{i=1}^{m} (x^{(i)} - \mu_{y^{(i)}})(x^{(i)} - \mu_{y^{(i)}})^T$$

The covariance matrix basicallly tries to fit contours to the ellipse (tries to fit a Gaussian to both of the positive and negative examples with their corresponding means, but you want one covariance matrix to both of theses)

$$\mu_1 = \frac{\sum_{i=1}^{m} 1\{y^{(i)}=1\} x^{(i)}}{\sum_{i=1}^{m} 1\{y^{(i)}=1\}}$$



$$\Sigma = \frac{1}{m} \sum_{i=1}^{m} (x^{(i)} - \mu_{y^{(i)}})(x^{(i)} - \mu_{y^{(i)}})^T$$

Having fit these parameters, if you want to make a prediction (given a new patient) how do you make a prediction for whether their tumor is malignant or benign?

If you want to predict the most likely class label, you choose:

Prediction:

$$\max_y \; p(y|x) = \max_y \frac{P(x|y)\,p(y)}{P(x)}$$

arg min/max:

$$\min_z (z-5)^2 = 0$$

$$\arg\min_z (z-5)^2 = 5$$

The **min** is the smallest possible value attained by the thing inside and the **arg min** is the value you need to plug in to achieve that smallest possible value

So the prediction you actually want to make, (if you want to output a value for y, you don't want to output a probability), you might choose a value of y that maximizes this

Prediction:

$$\arg\max_y \; p(y|x) = \arg\max_y \frac{P(x|y)\,p(y)}{P(x)}$$

The arg max p (y|x) would be either 0 or 1
y

Prediction:

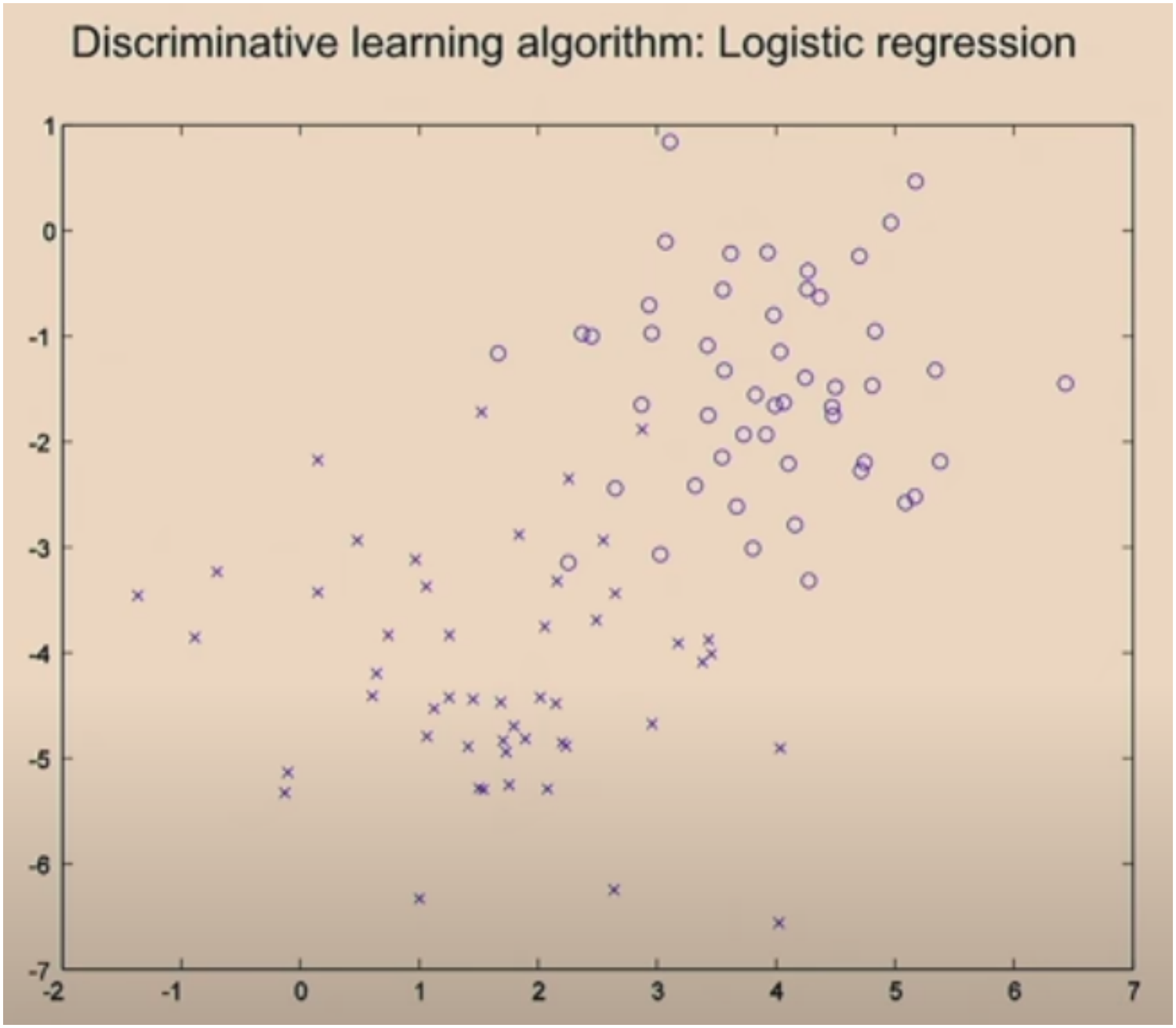
$$\arg\max_y p(y|x) = \arg\max_y \frac{p(x|y)\,p(y)}{p(x)}$$

← "constant value"

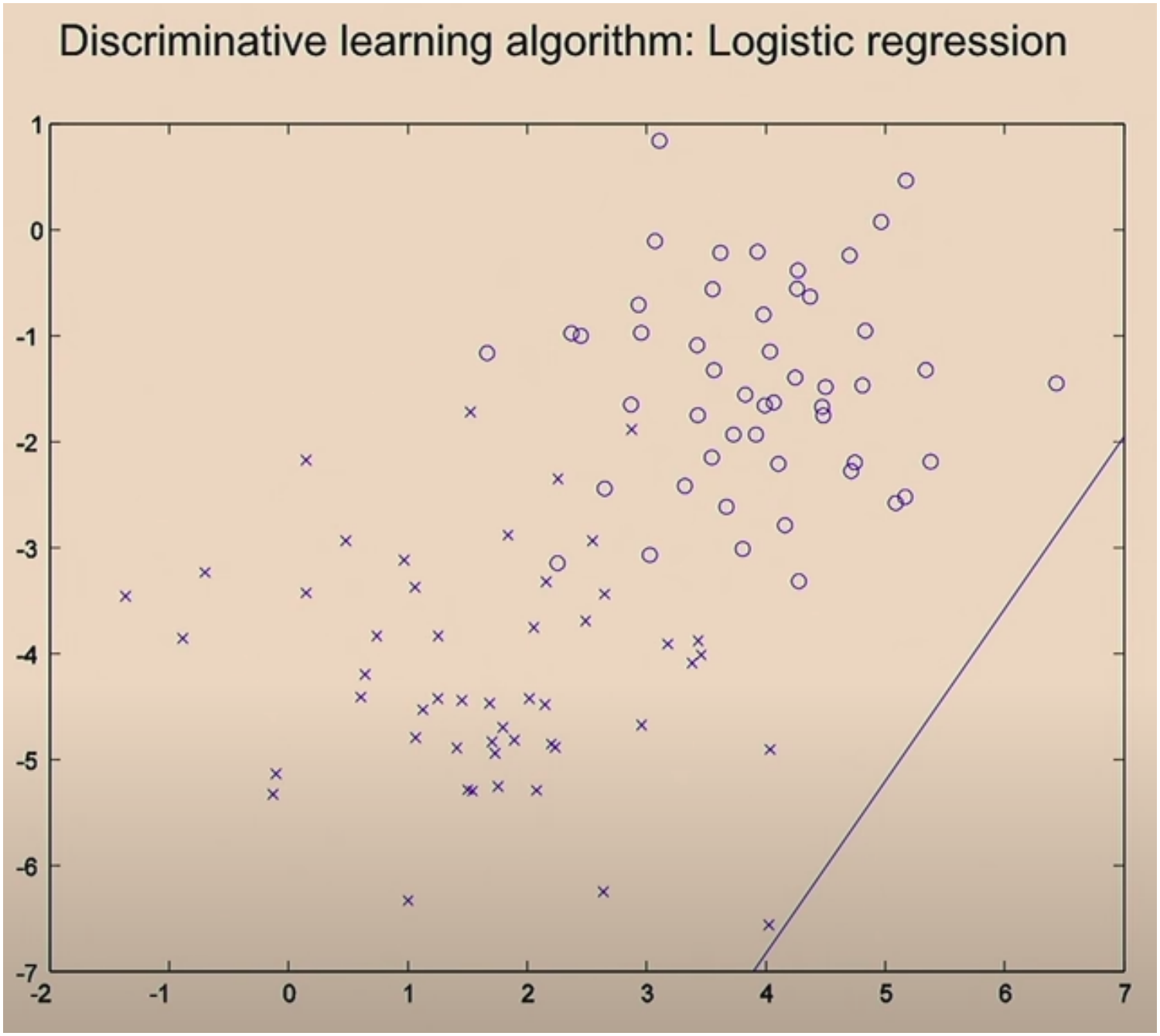
$$= \arg\max_y p(x|y)\,p(y)$$

When making predictions with Gaussian generative learning algorithms, sometimes to save on computation you don't bother to calculate the denominator if all you care about is to make a prediction, but if you'd actually need a probability, then you'd have to normalize the probability

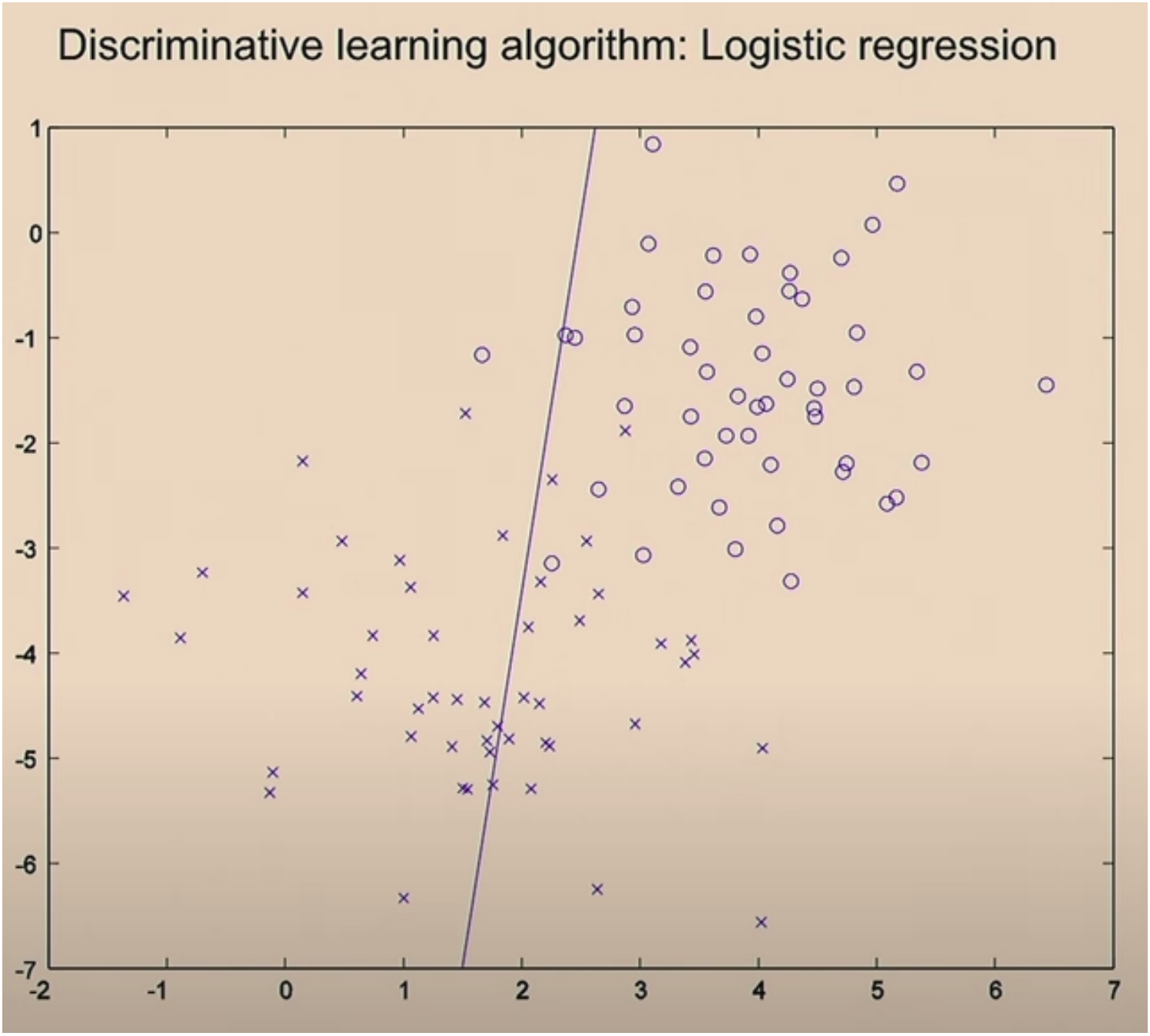
Visualizations:



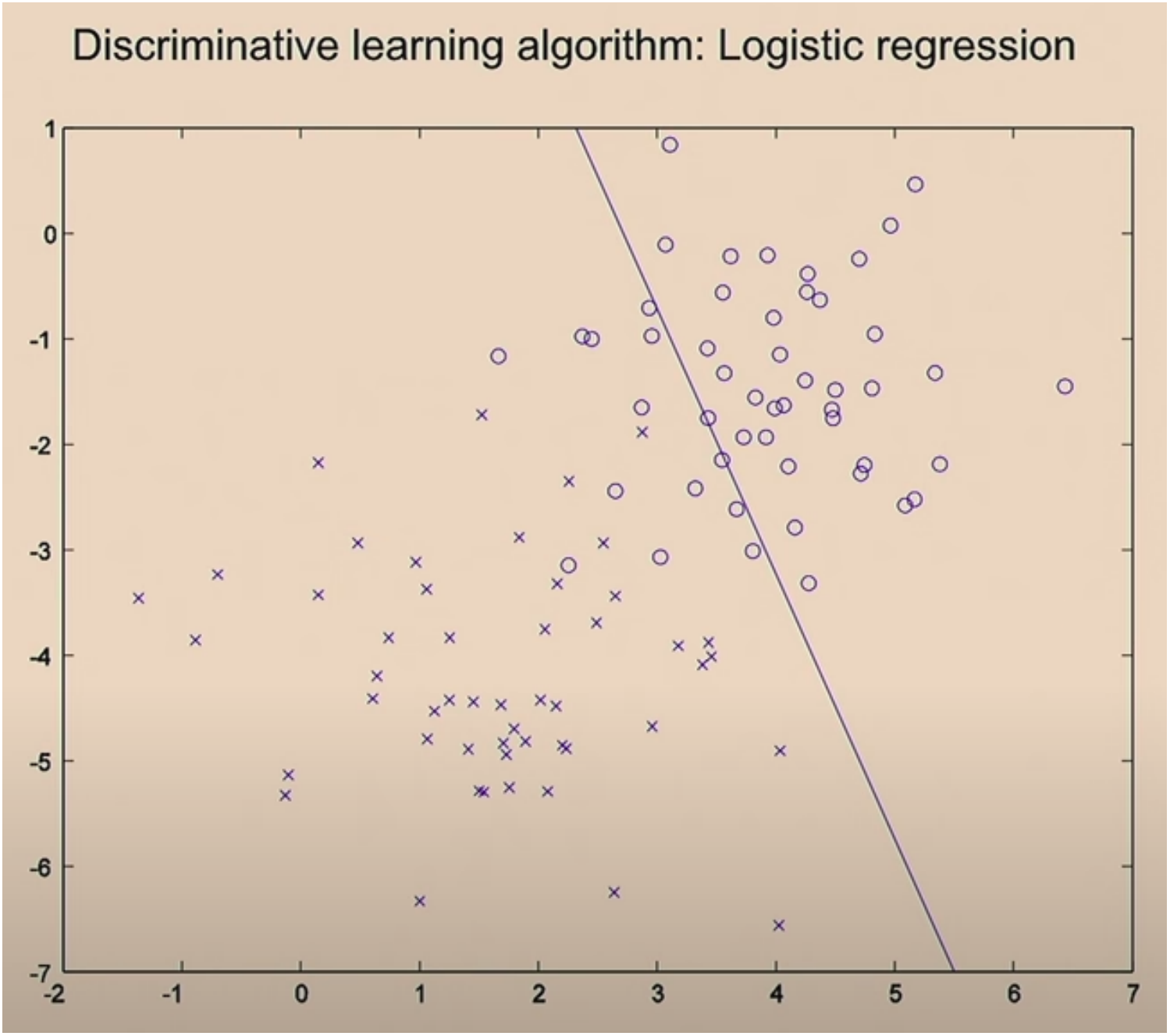
Initialize the parameters randomly:



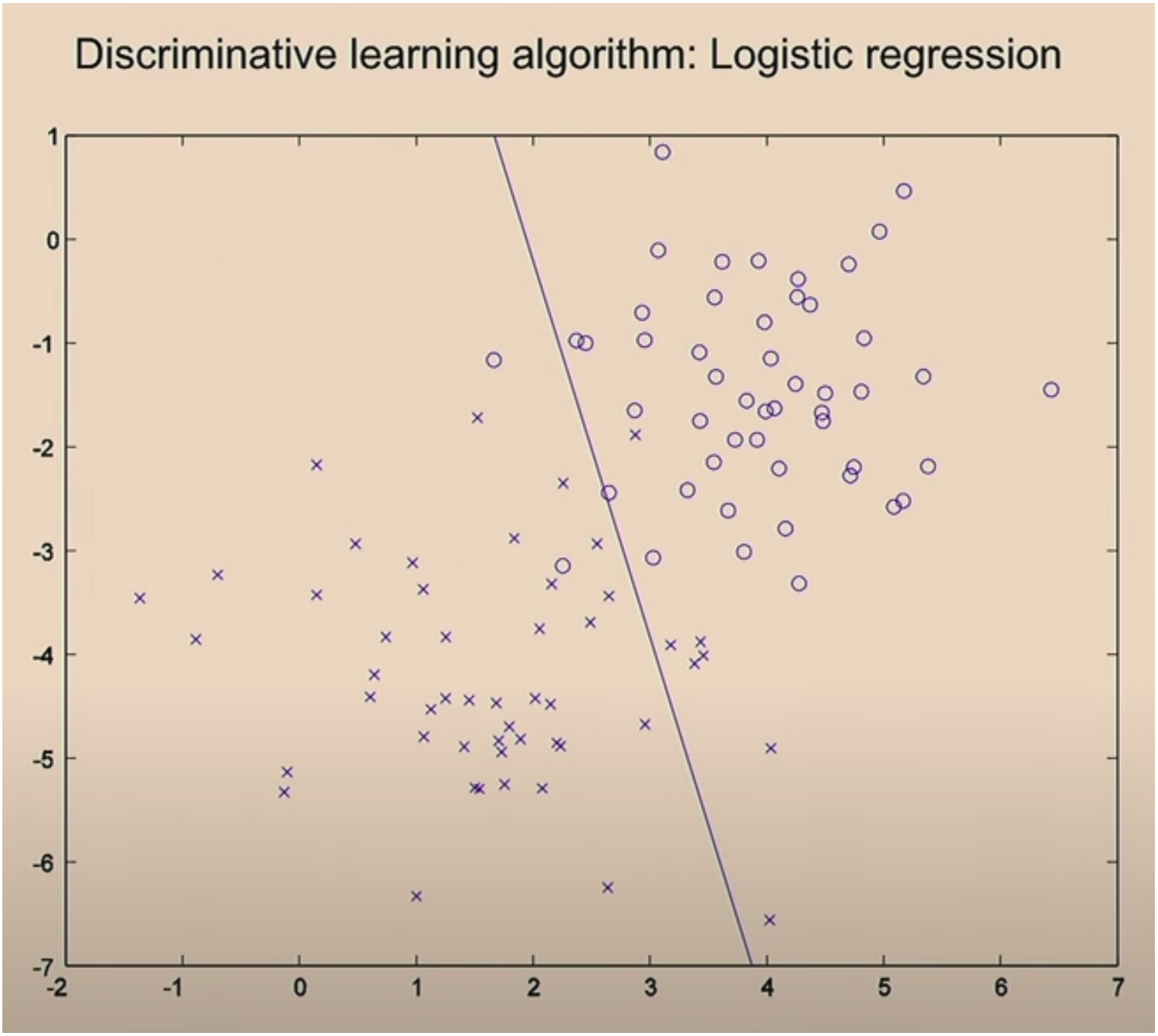
Running one iteration of gradient descent on the conditional likelihood (one iteration of logistic regression):



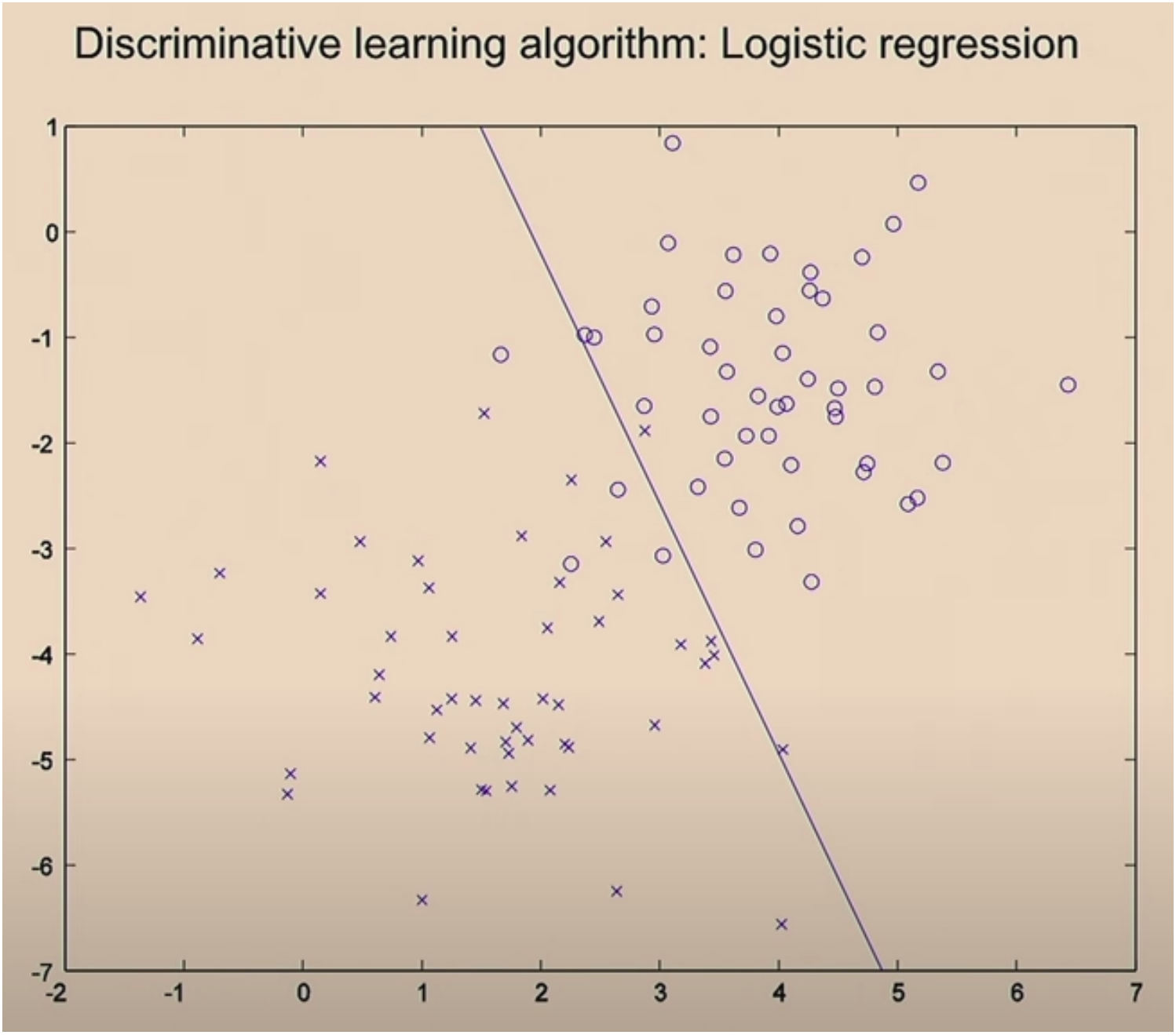
Two iterations:

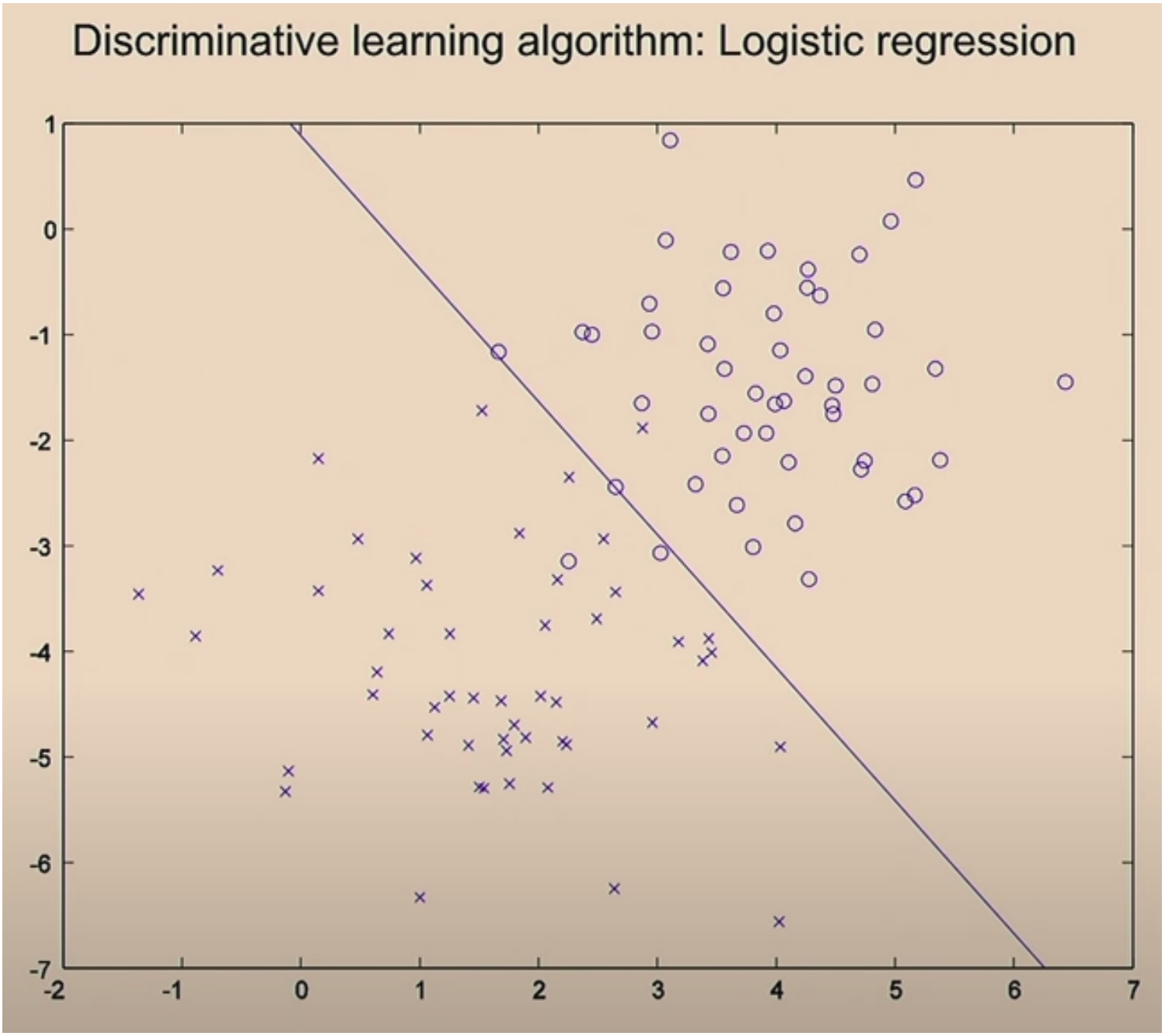


Three iterations:



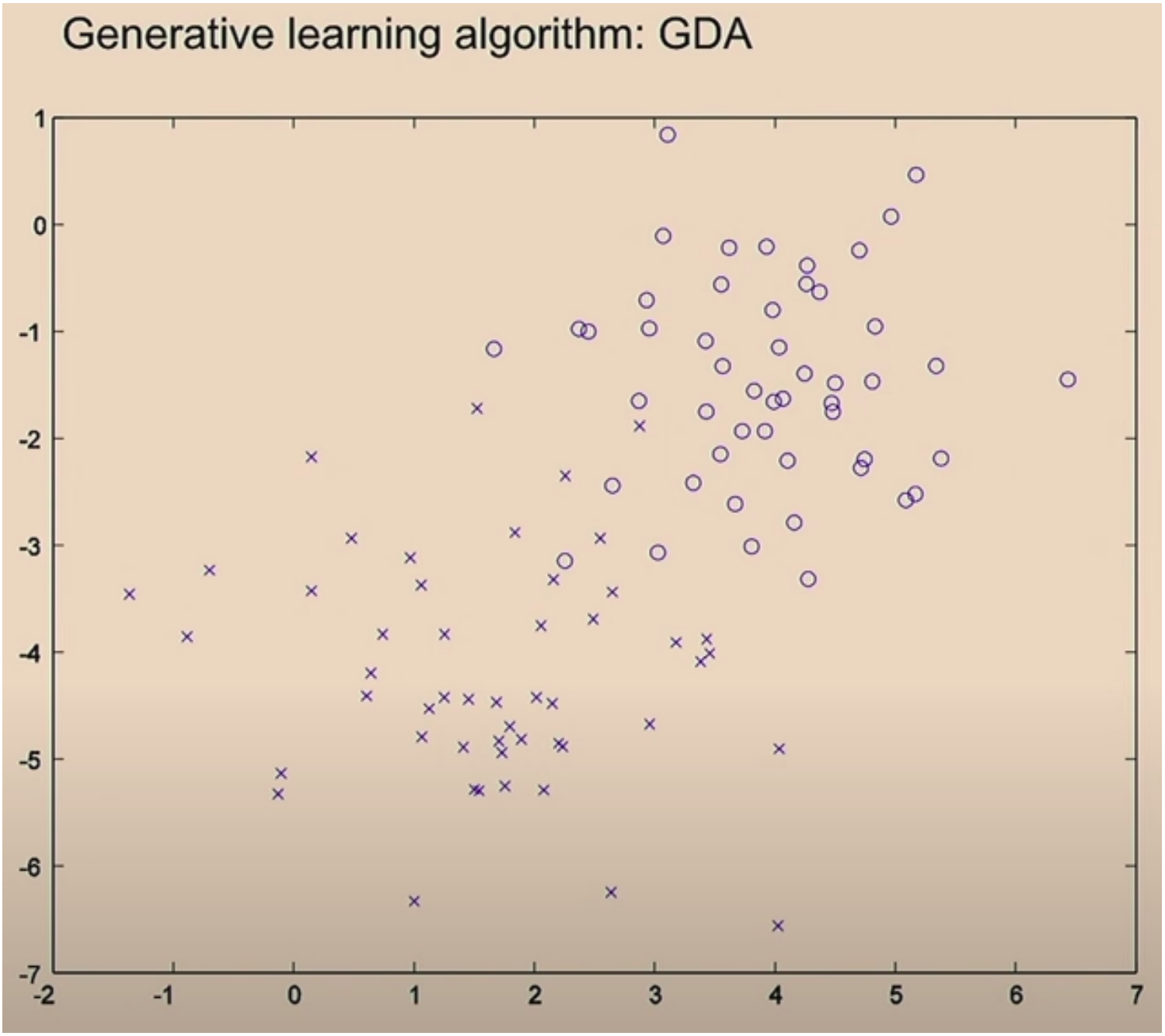
Four iterations:


Discriminative learning algorithm: Logistic regression
1
0
-1
-2
-3
-4
-5
-6
-7
-2
-1
0
1
2
3
4
5
6
7

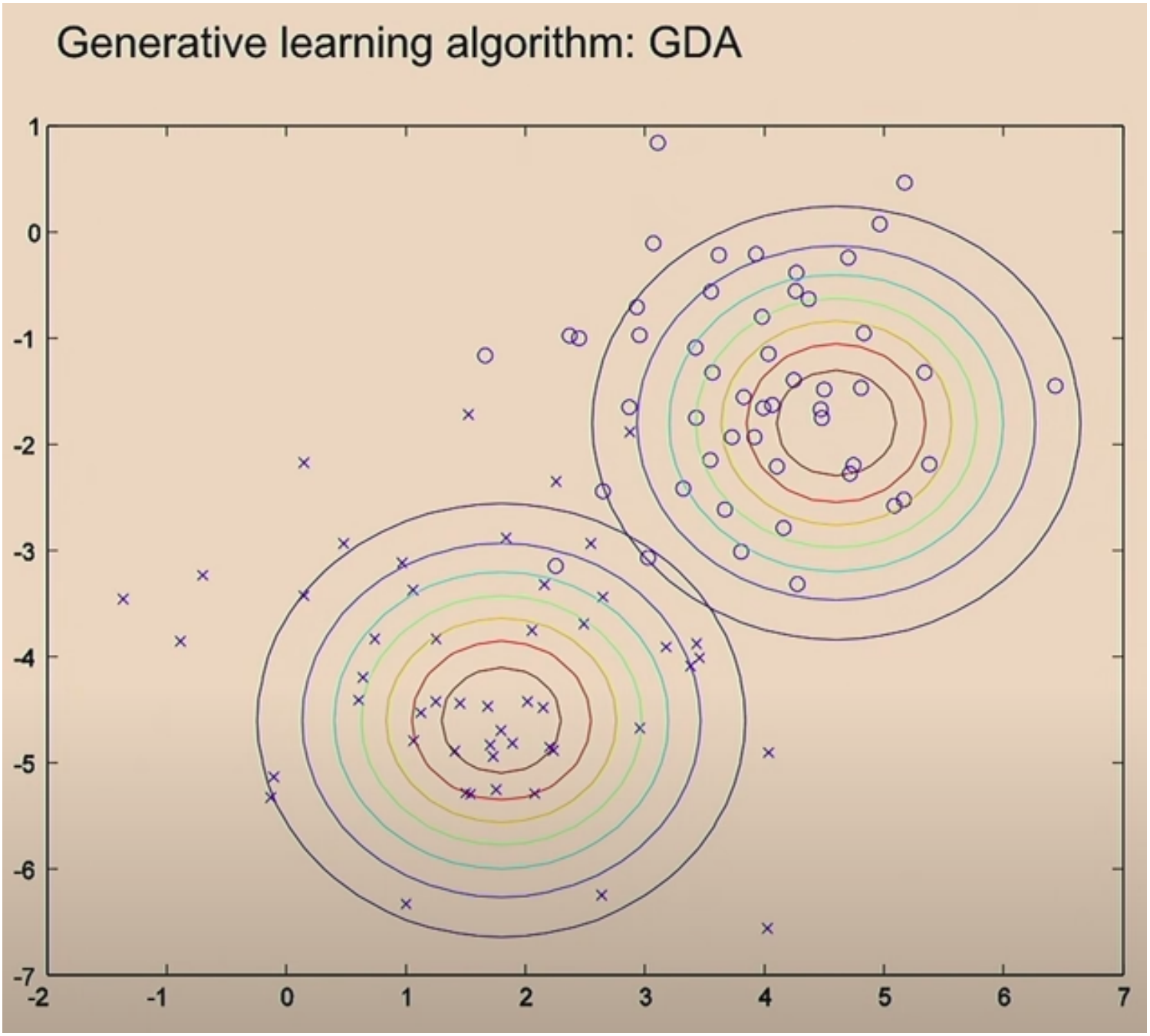


After about 20 iterations it will converge to that pretty decent discriminative boundary

Logistic Regression: Searching for a line that separates positive and negative examples

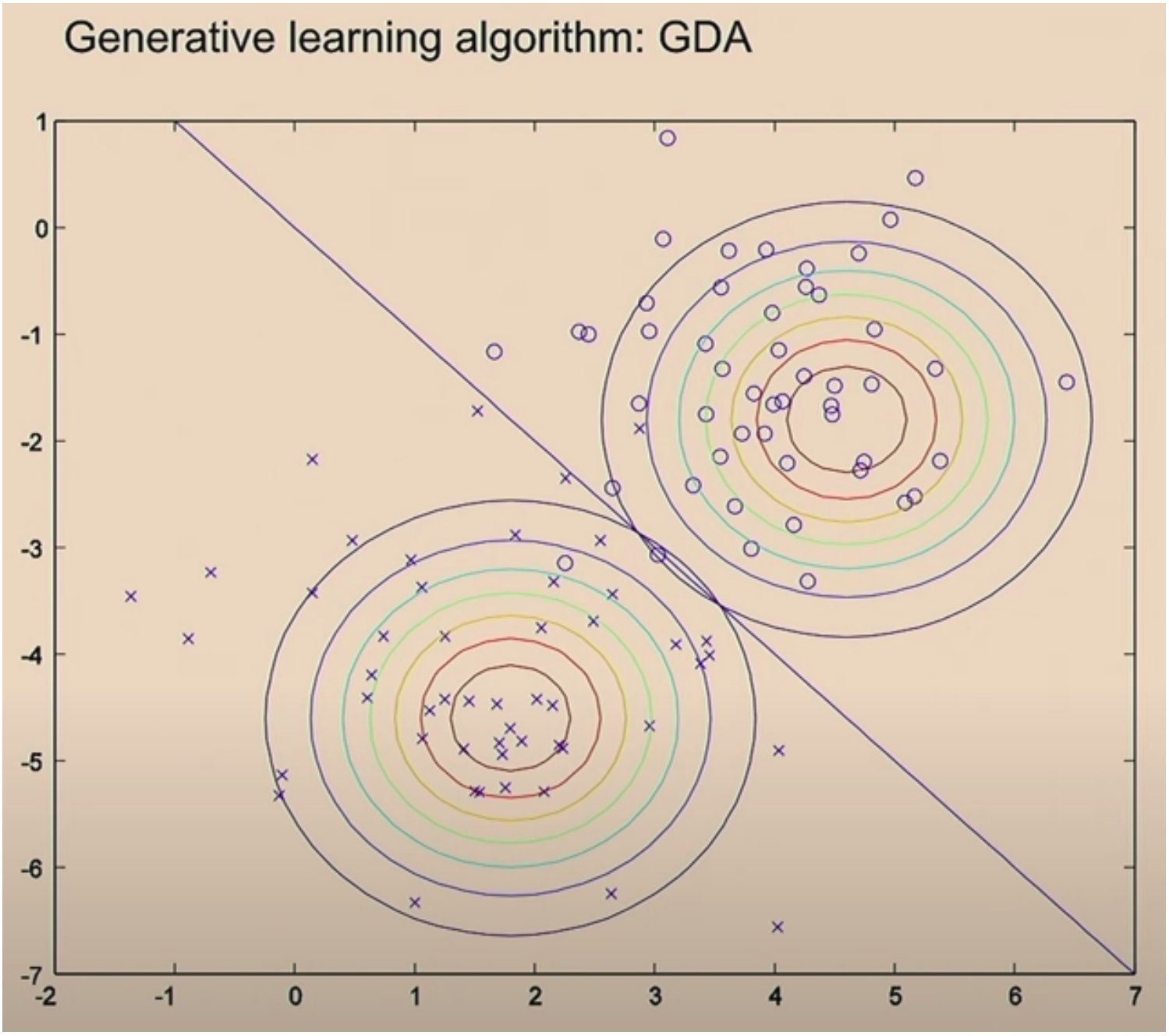


Generative learning algorithms fit with Gaussian discriminant analysis. What we'll do is fit Gaussians to the positive and negative examples



Initially described as looking at the two classes separately because we use the same covariance matrix Σ for the positive and negative classes, we actually don't quite look at them totally separately but we do fit two Gaussian densities to the positive and negative examples, and then what we do is, for each point try to decide what is its class label using Bayes' rule

It turns out that this implies the following decision boundary

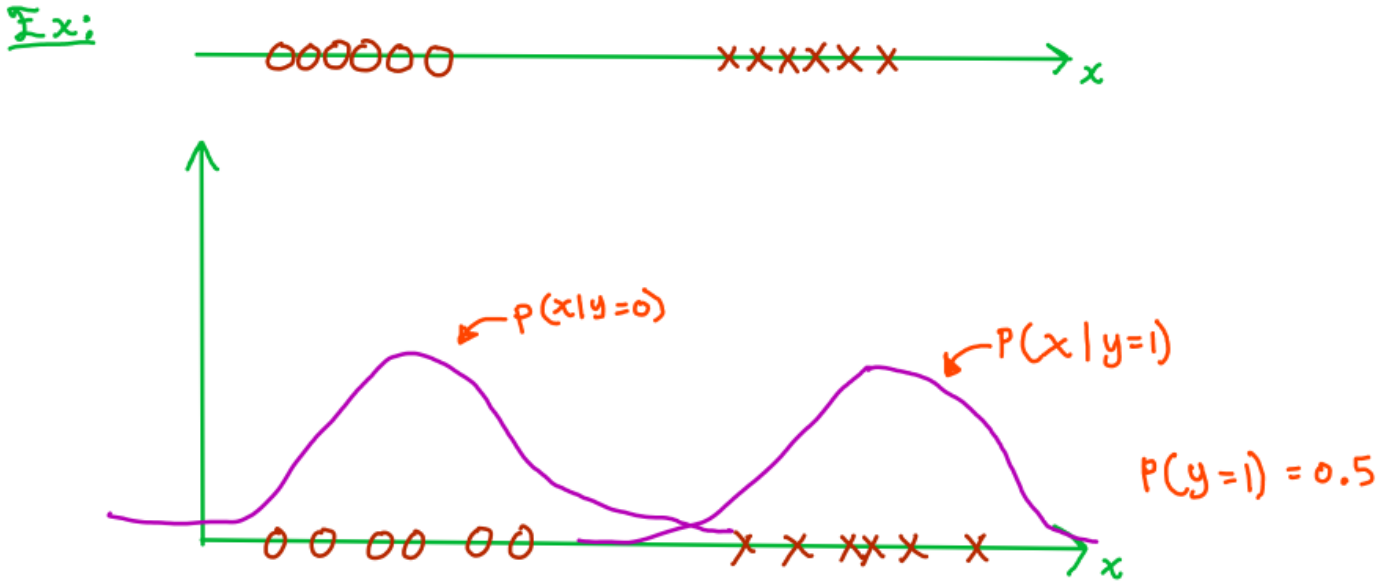

Generative learning algorithm: GDA
1
0
-1
-2
-3
-4
-5
-6
-7
-2
-1
0
1
2
3
4
5
6
7

Example:

With 1 feature, negative O and positive X

We'll see what ***Gaussian Discriminant Analysis*** will do on this dataset

Mapping this data to an x-axis gives us:

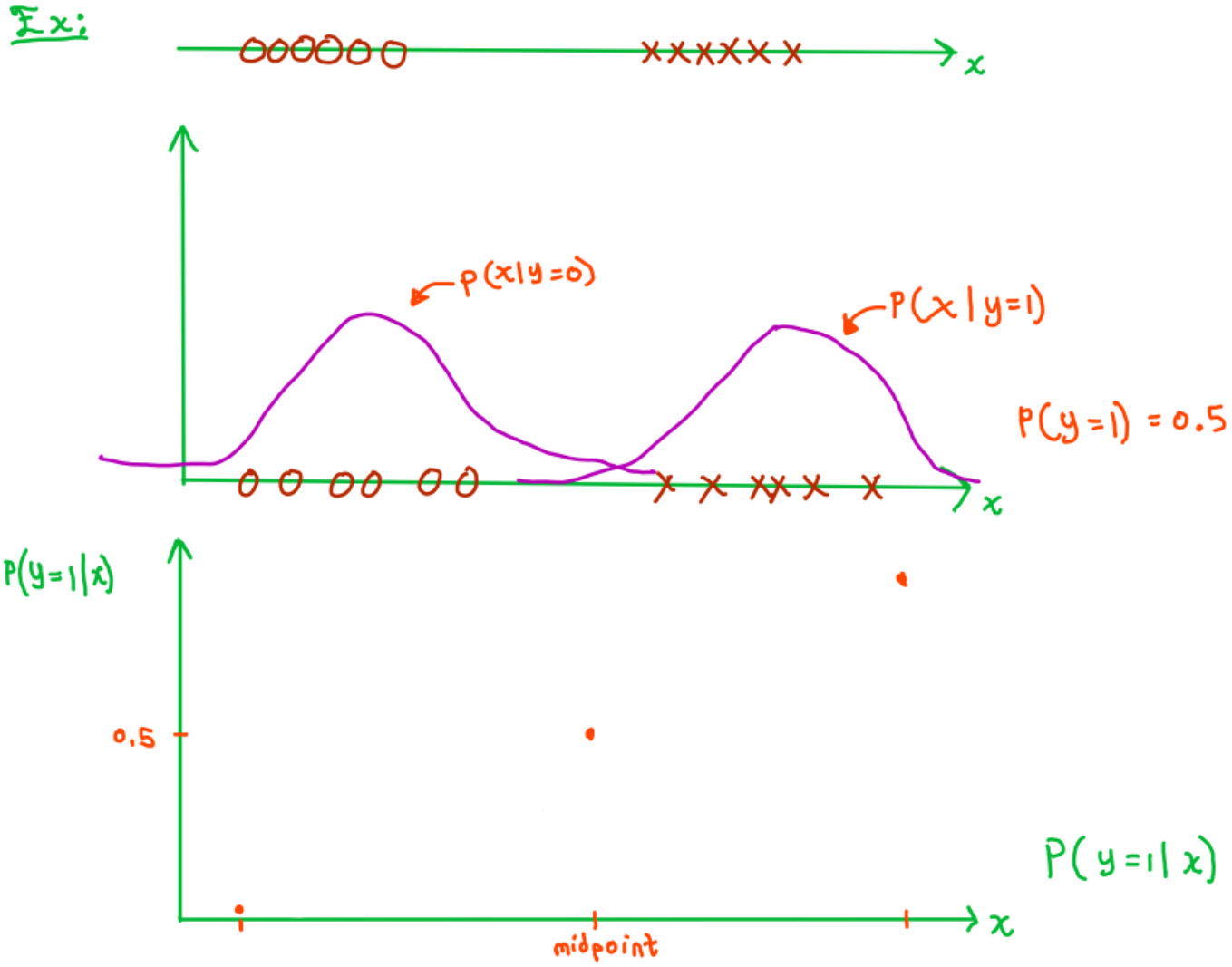


If you fit a Gaussian to each of these two data sets then you end up with the above Gaussians where the left bump is p(x|y=0) and the right bump is p(x|y=1)

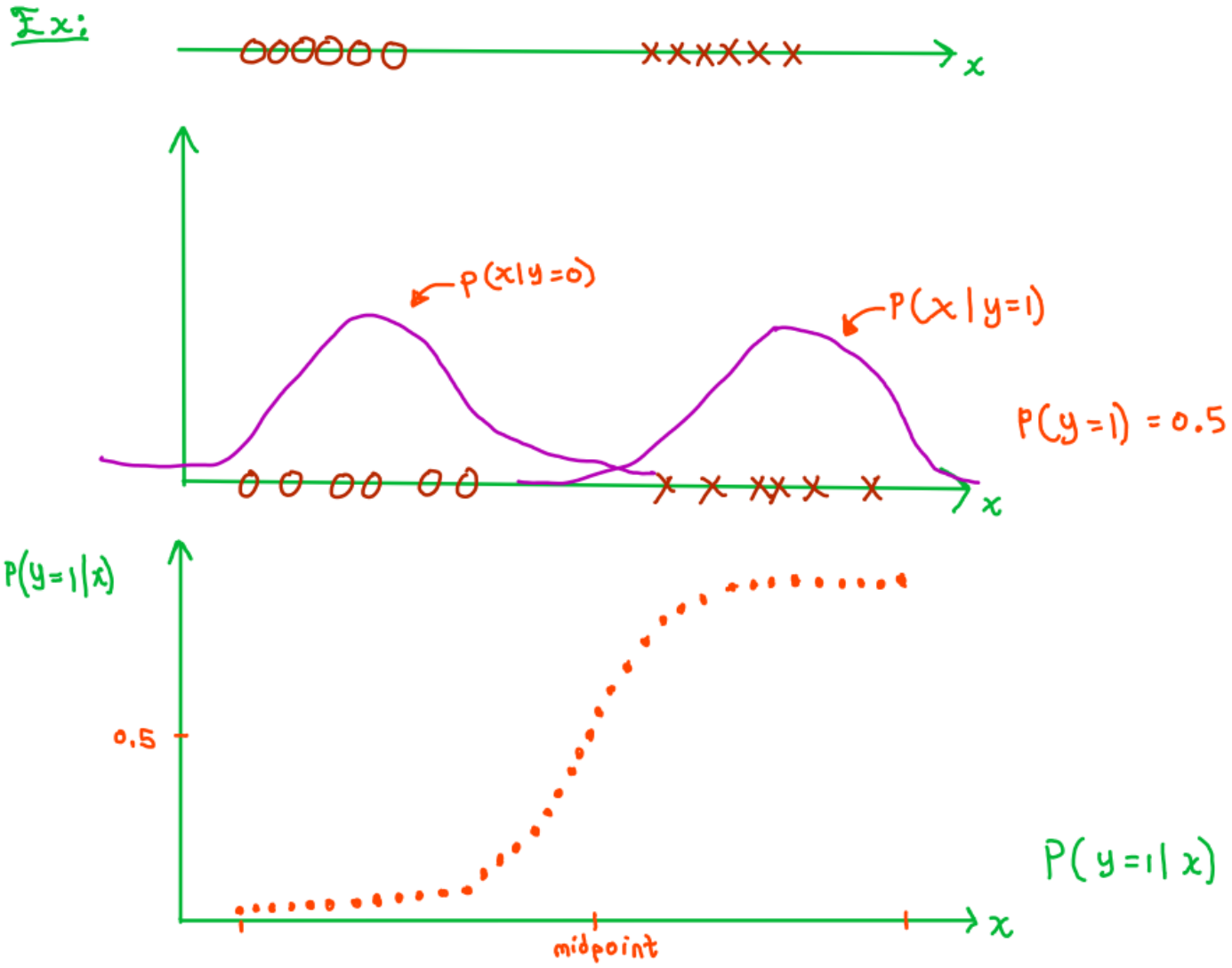
We set the same variance to the two Gaussians (you kind've model the Gaussian densities of what does this **class 0** look like and what does this **class 1** look like with two Gaussian bumps like this)

Because the dataset is split 50/50, p(y=1) = 0.5

Plotting p(y=1|x) for different values of X:

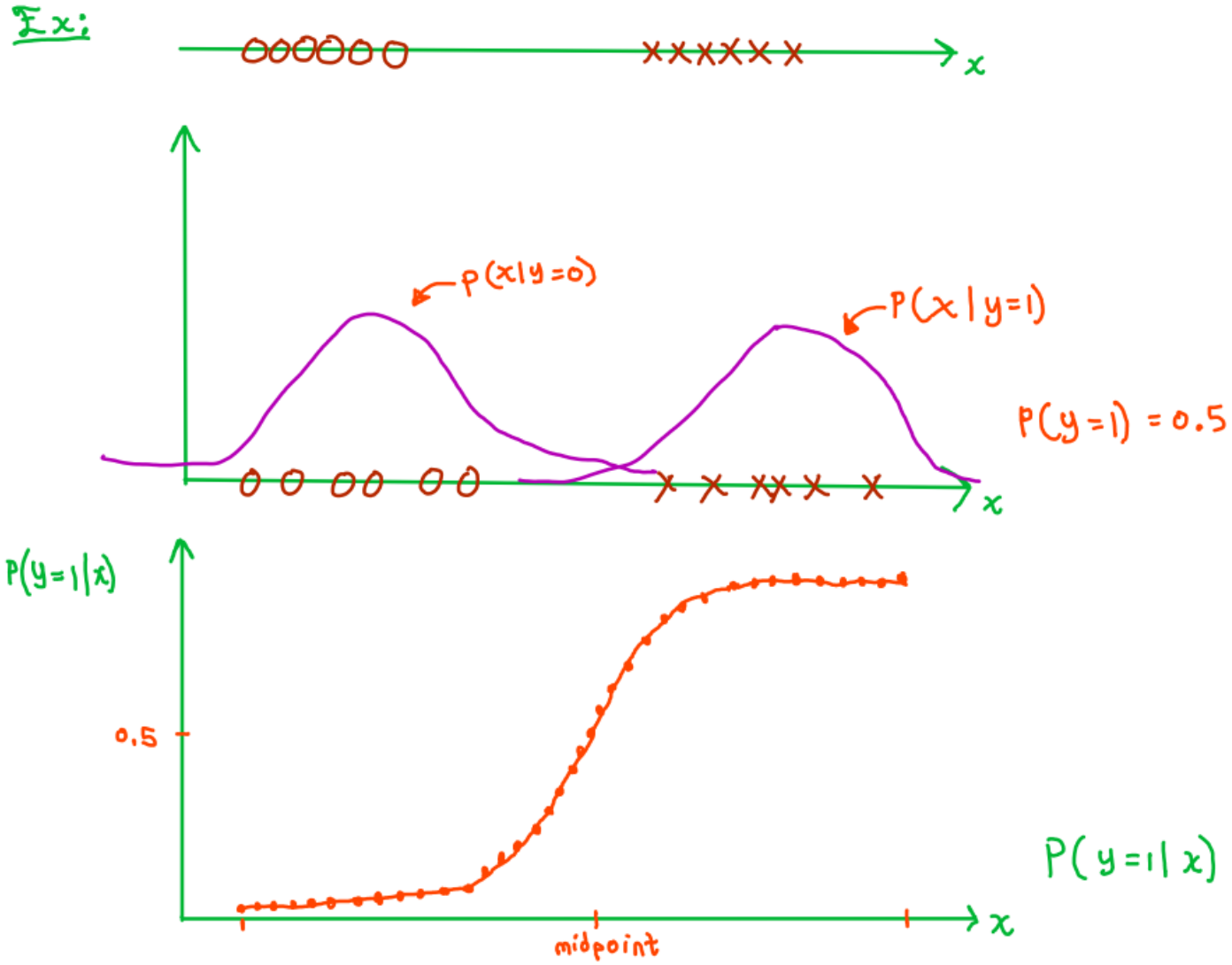


It turns out that if you repeat this exercise, sweeping from left to right for many many points on the X-axis, you find that for points far to the left, the chance of coming from the **y=1 class** is very small and as you approach the midpoint, it increases to **0.5** and it surpasses 0.5 and then beyond a certain point it becomes very very close to **1**



You do this exercise and actually just for every point, for a dense grid on the x-axis, evaluate this formula which will give you a number between 0 and 1 (the probability) and go ahead and plot the values, you get a curve like this

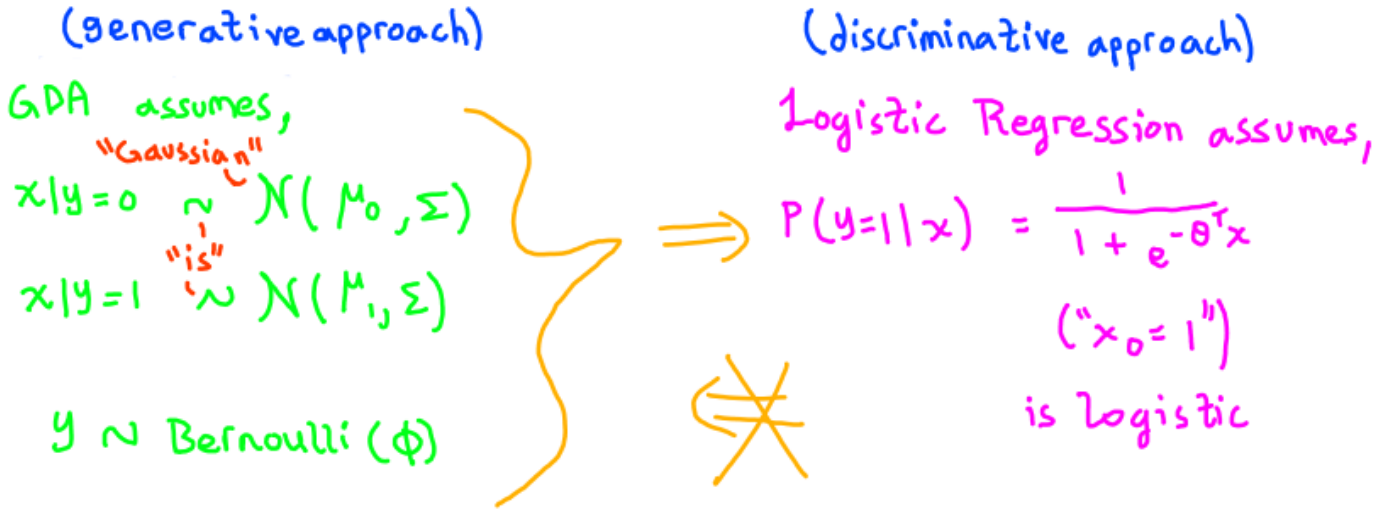
It turns out that if you connect the dots, then this is exactly a sigmoid function



The outcome of both ***Logistic Regression*** and ***Gaussian Discriminant Analysis*** actually ends up being a sigmoid function that calculates p(y=1|x)

But the specific choice of the parameters they end up choosing are quite different and, as seen in the visualizations, the two algorithms actually come up with two different decision boundaries

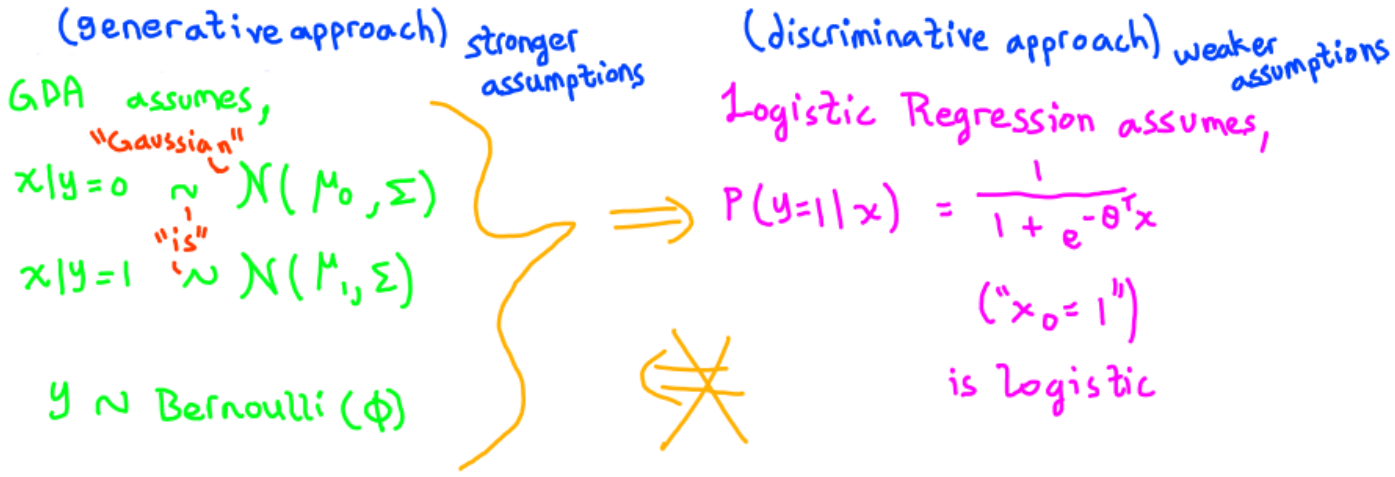
Let's discuss when a **Generative algorithm**, like **GDA**, is superior and when a **Discriminative algorithm**, like **Logistic Regression**, is superior:



The plotted sigmoid function above illustrates that the ***Generative approach*** set of assumptions implies that p(y=1|x) is governed by a logistic function

But it turns out that the implication in the opposite direction is not true. So if you assume p(y=1|x) is governed by logistic funtion by the sigmoid shape, this does not in any way, shape, or form assume that x|y=0, x|y=1 is Gaussian

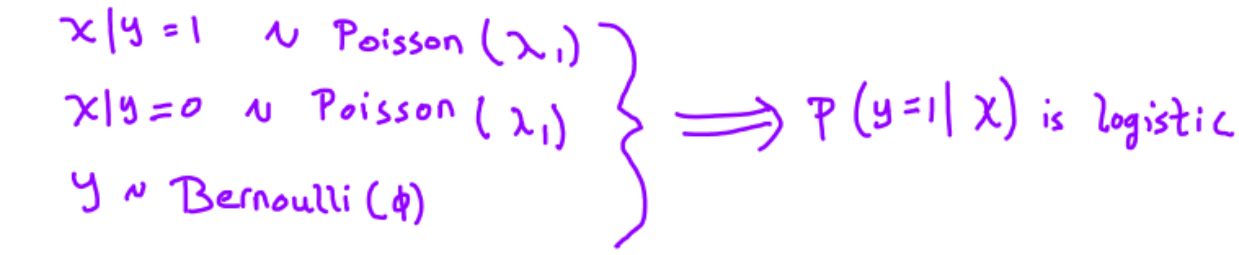
What this means is that GDA (the generative learning algorithm in this case) makes a stronger set of assumptions and Logistic Regression makes a weaker set of assumptions because you can prove the ***discriminative approach*** assumptions from the ***generative approach*** assumptions



What you see in a lot of learning algorithms is that if you make strongly modeling assumptions and if your modeling assumptions are moderately correct, then your model will do better because you're telling more information to the algorithm

So if indeed x|y is Gaussian, then GDA will do better because you're telling the algorithm x|y is Gaussian and so it can be more efficient. And so even if you have a very small dataset, if these assumptions are roughly correct, then GDA will do better. And the problem with GDA is if these assumptions turn out to be wrong, so if x|y is not at all Gaussian, then this might be a very bad set of assumptions to make. You might be trying to fit a Gaussian density to data that is not at all Gaussian and then GDA would do more poorly

Fun fact:



This is actually true for any generalized linear model where the difference between these two distributions varies only according to the natural parameter of the exponential family distribution

What this means is that if you don't know if your data is Gaussian or Poisson, if you're using Logistic Regression you don't need to worry about it, it'll work fine either way

Maybe you're fitting a model, binary classification model, to some data and you don't know if the data is Gaussian, Poisson, some other exponential family model, maybe you just don't know. But if you're fitting Logistic Regression it'll do fine under all of those scenarios

But if your data was actually Poisson but you assumed it was Gaussian, then your model might do quite poorly

The key high level principals you should take away from this is, if you make weaker assumptions as in Logistic Regression then your algorithm will be more robust to modeling assumptions such as accidentally assuming the data is Gaussian and it is not, but on the flip side, if you have a very small dataset then using a model that makes more assumptions will actually allow you to do better because by making more assumptions you're just telling the algorithm more truth about the world which will allow it to do better

#For problems where you have a lot of data you would probably use Logistic Regression because with more data you could overcome telling the algorithm less about the world

The algorithm has two sources of knowledge, one source of knowledge is what did you tell it (what are the assumptions you told it to make), and the second source of knowledge is learned from the data

One practical reason why you should still use algorithms like GDA (General Discriminant Analysis) or algorithms like this is that it's actually quite computationally efficient and it turns out computing mean and variances of covariance matrices is very efficient and the other reason is actually, apart from the assumptions type of benefit, there's no iterative process needed. So when you use these models, you're more motivated by computation and less by performance

*** This idea about “Do you make strong or weak assumptions?” is a general principle in machine learning ***

What happens when the co-variance matrices are different? It turns out that it still ends up being a logistic function but with a bunch of quadratic terms in the logistic function, so it's not a linear decision boundary anymore

Naive Bayes:

A Generative learning algorithm

The motivating example is email spam classification

The first question we will have given the email classification problem, **how do you represent it as a feature vector?**

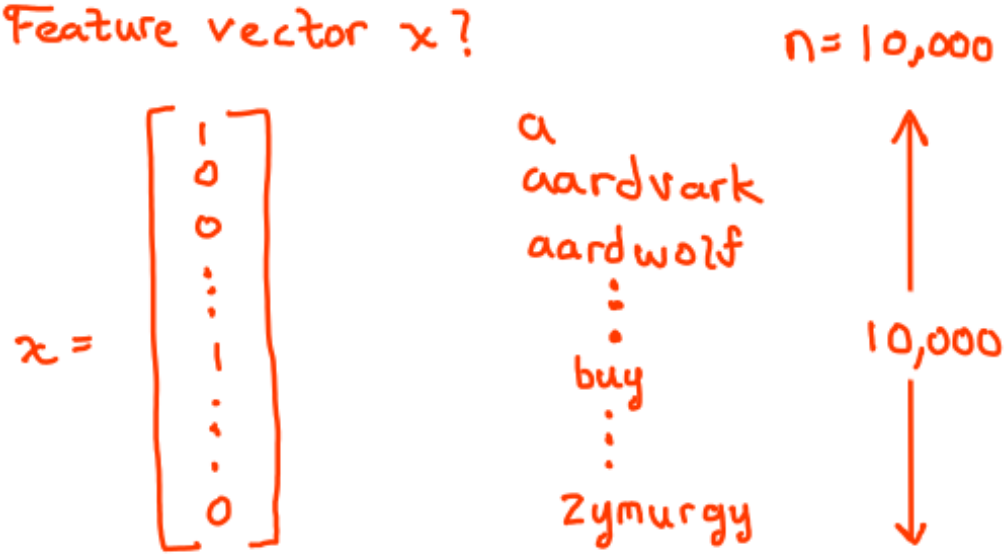
In **Naive Bayes** what we're going to do is take your email, take a piece of email, and first map it to a feature vector X and we'll do so as follows:

First, let's start with the English dictionary and make a list of all the words in the English dictionary. In practice what you do is not actually look at the dictionary but look at the top 10,000 words in your training set

And so, given an email, what we would like to do is then take this piece of text and represent it as a feature vector. And one way to do this is you can create a binary feature vector that puts a **1** if a word appears in the email and puts a **0** if it doesn't

Naive Bayes:

Feature vector x?

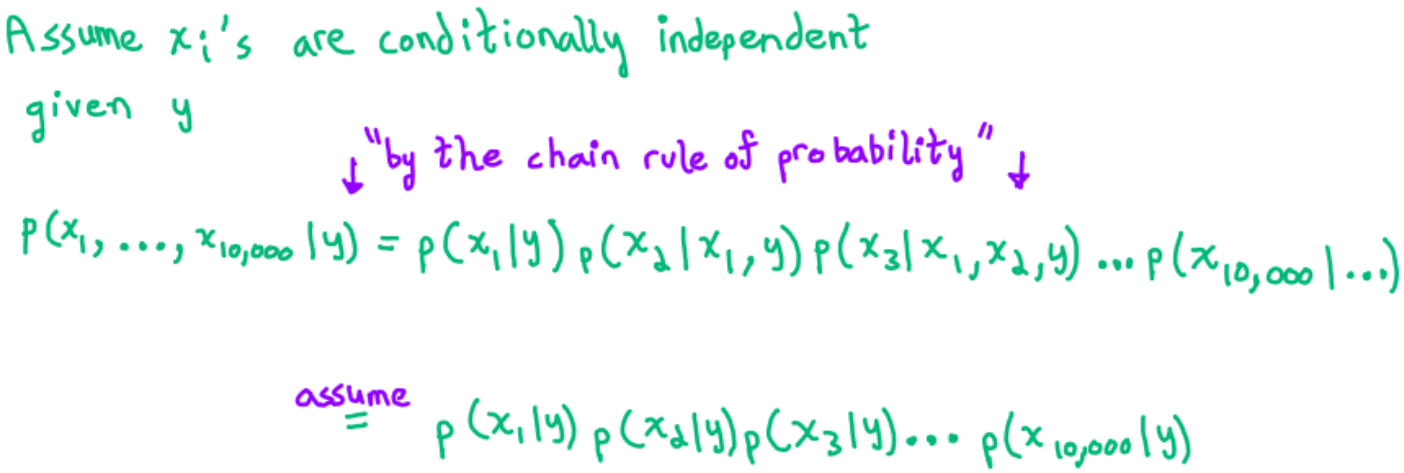


$x \in \{0,1\}^n$

$x_i = 1\{\text{word } i \text{ appears in email}\}$

Want to model $p(x|y)$, $p(y)$

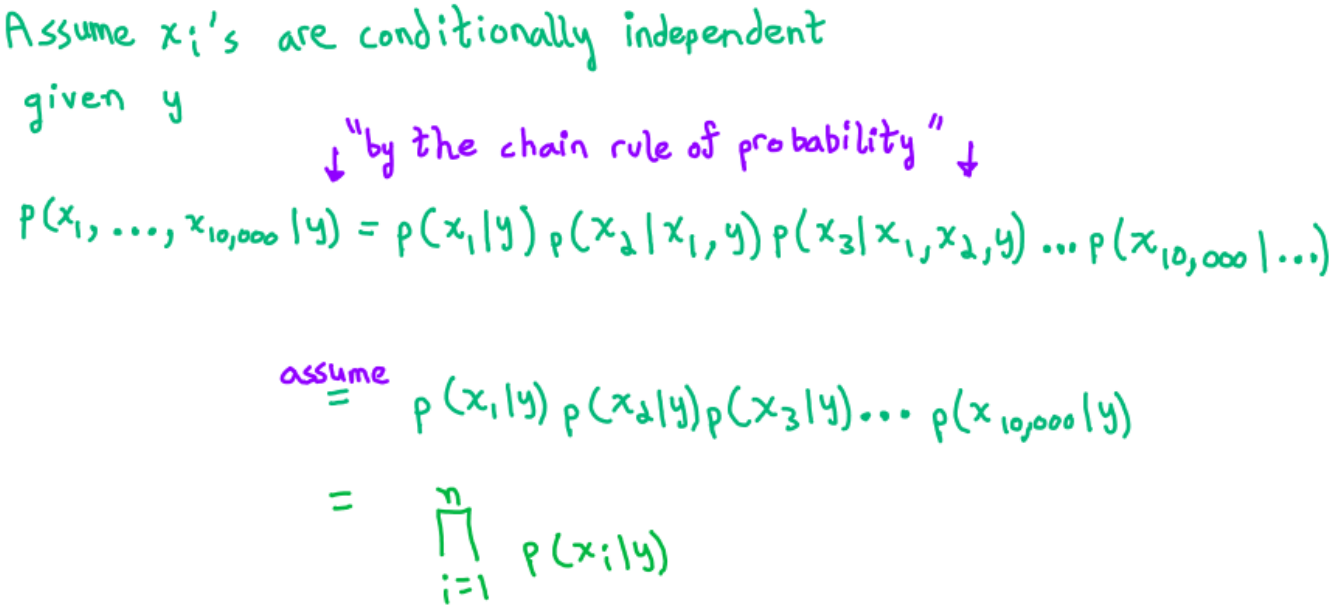
$2^{10,000}$ possible values of x



This assumption is called a ***conditional independence assumption***, also sometimes called the ***Naive Bayes assumption***, but you're assuming that so long as you know **y**, the chance of seeing the words “aardvark” in your email does not depend on whether the word “a” appears in your email

This is one of those assumptions that is definitely not a true assumption in that is just not a mathematically true assumption, just as sometimes your data isn't perfectly Gaussian

You just use this equation, that's all you need to derive Naive Bayes



Parameters:

$$\phi_{j|y=1} = P(x_j = 1 \mid y = 1)$$

"y=1 is spam"

y=0 not spam

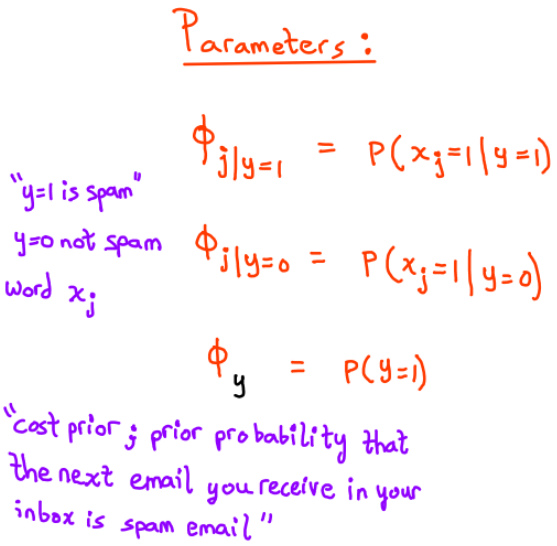
$$\phi_{j|y=0} = P(x_j = 1 \mid y = 0)$$

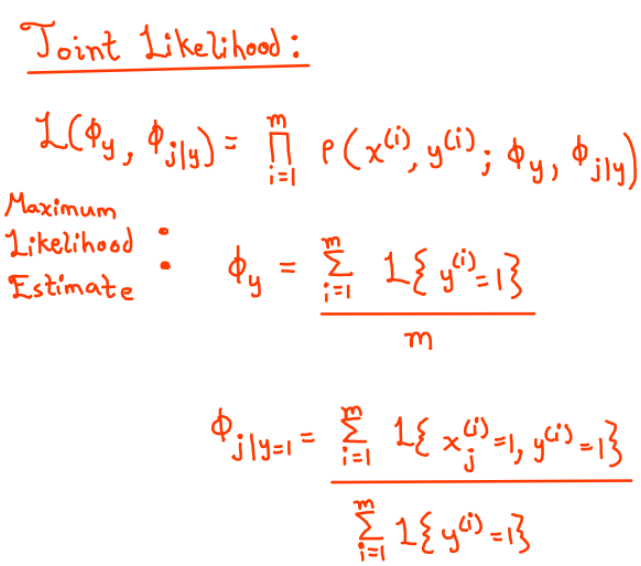
word x_j

$$\phi_y = P(y = 1)$$

"cost prior; prior probability that the next email you receive in your inbox is spam email"

To fit the parameters of this model, you would, similar to Gaussian Discriminant Analysis, write down the ***Joint Likelihood***





That's the ***Indicator function notation*** of writing out "Look through your training set, find all the spam emails and of all the spam email, i.e. examples of **y=1**, count up what fraction of them had **word j** in it

So you estimate that the chance of word **j** appearing in a spam email is just, we have all the spam emails in your training set, what fraction of them contain that word (what fraction of them had $\mathbf{x_j=1}$ for the word

It turns out that if you implement this algorithm, it will nearly work, but this is Naive Bayes for email spam classification

With one fix to this algorithm, this is actually a not too horrible spam classifier

It turns out that if you used Logistic Regression for spam classification, you'd do better than this almost all of the time, but this is a very efficient algorithm because estimating these parameters is just counting and then computing probabilities is just multiplying a bunch of numbers, so there's nothing iterative about this. So you can fit this model very efficiently and also keep on updating this model even as you get new data you can update this model very efficiently

But it turns out that the biggest problem with this algorithm is what happens if you get 0's in some of these equations