Title: Machine Learning Lecture

Speakers: Mohamed Hibat Allah

Collection: Machine Learning 2023/24

Date: April 22, 2024 - 3:15 PM

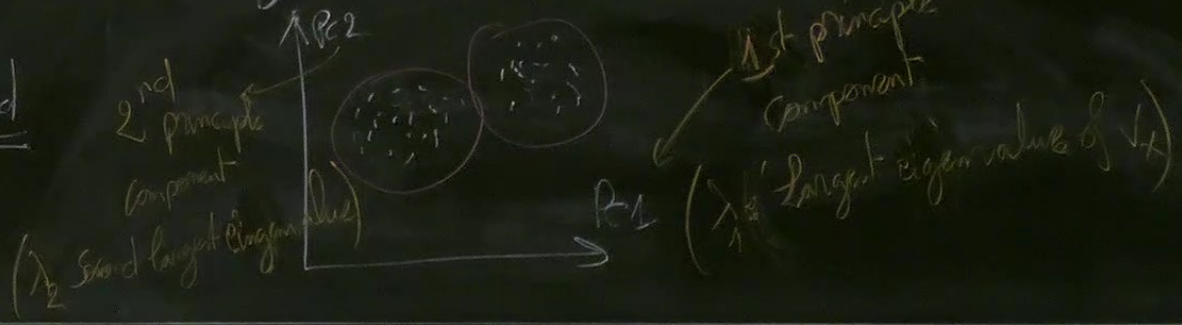
URL: https://pirsa.org/24040057

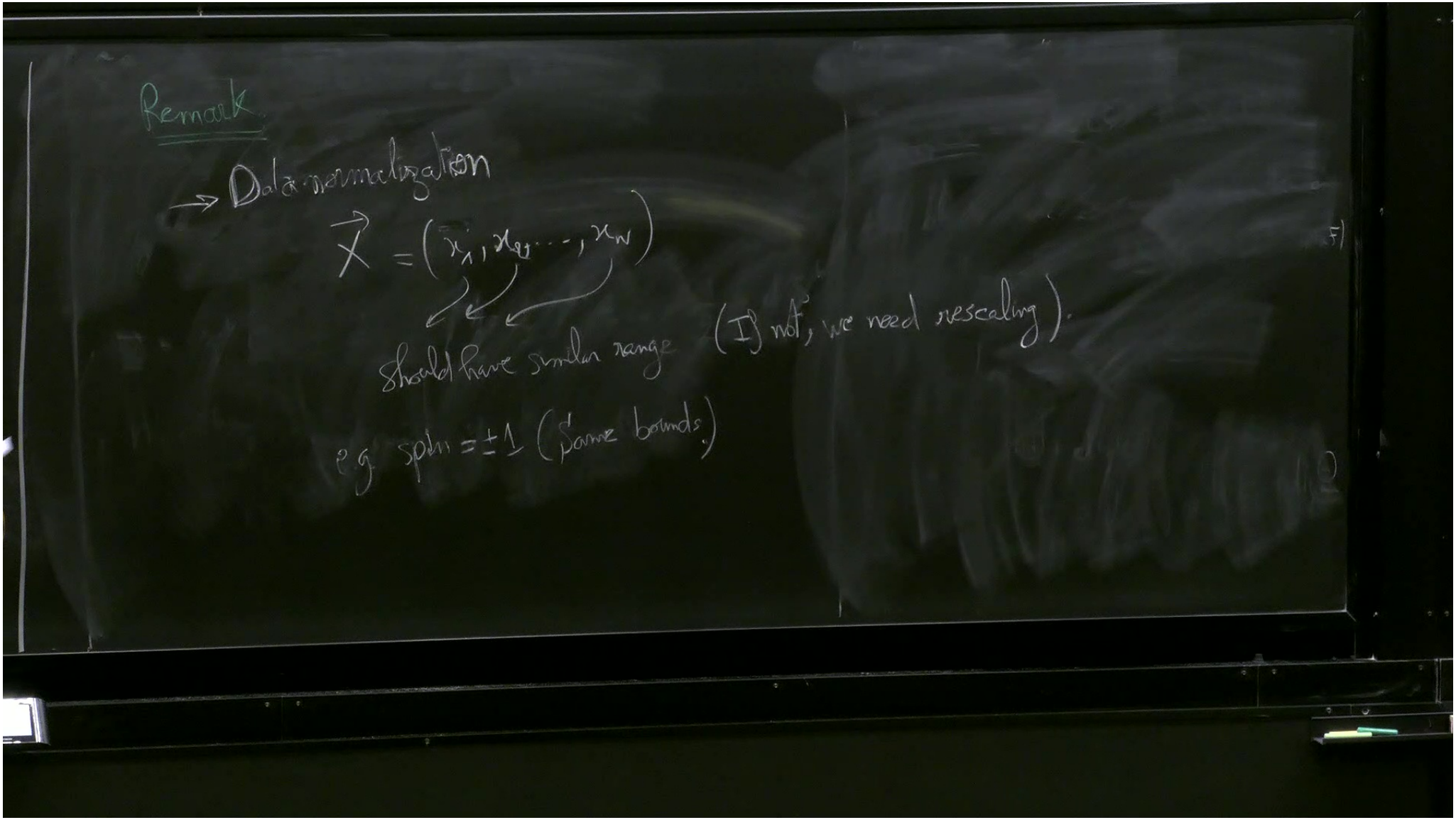
Lecture 9

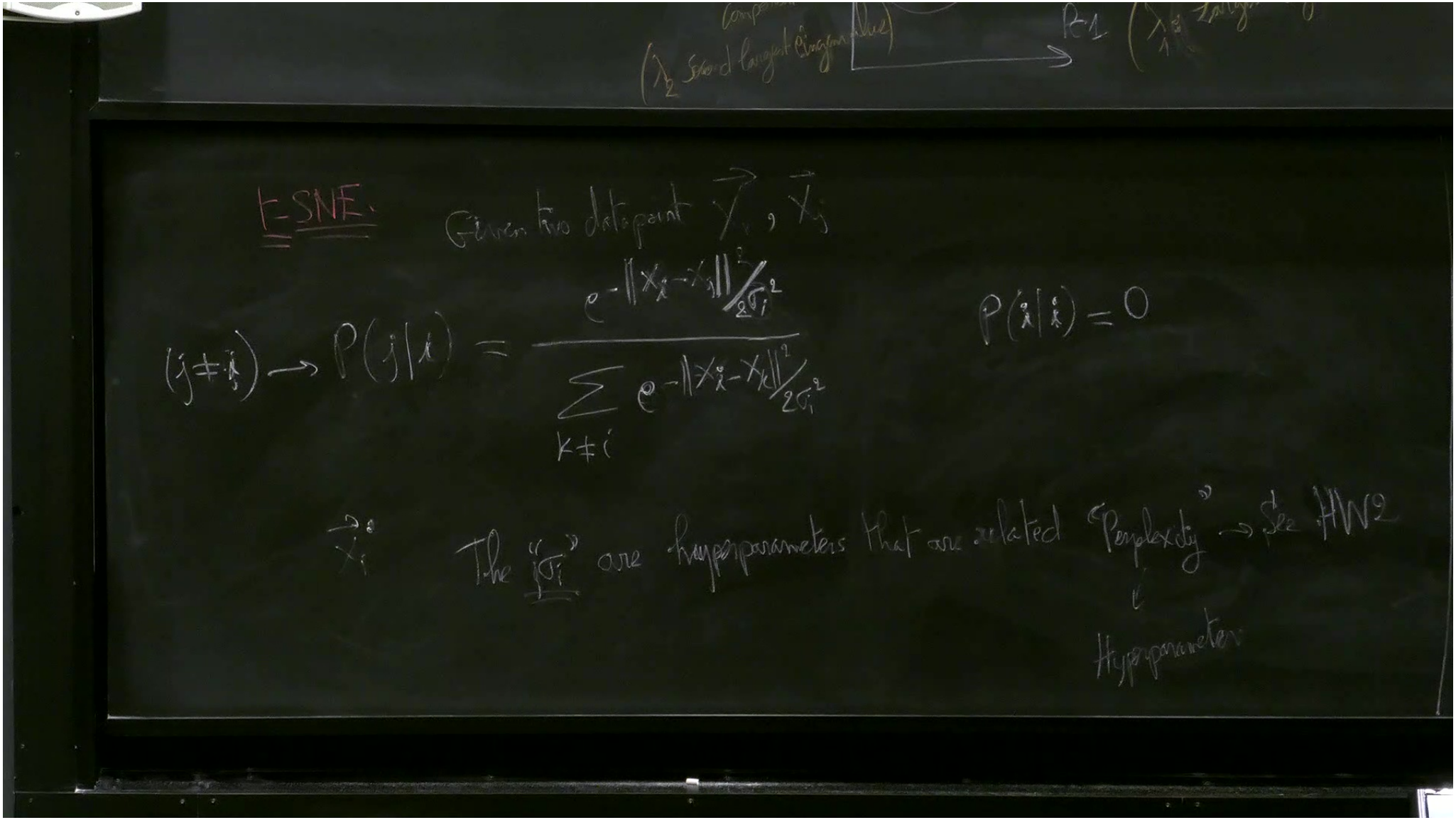
Last time:

↳ Principle Component analysis (PCA)

↳ Linear method



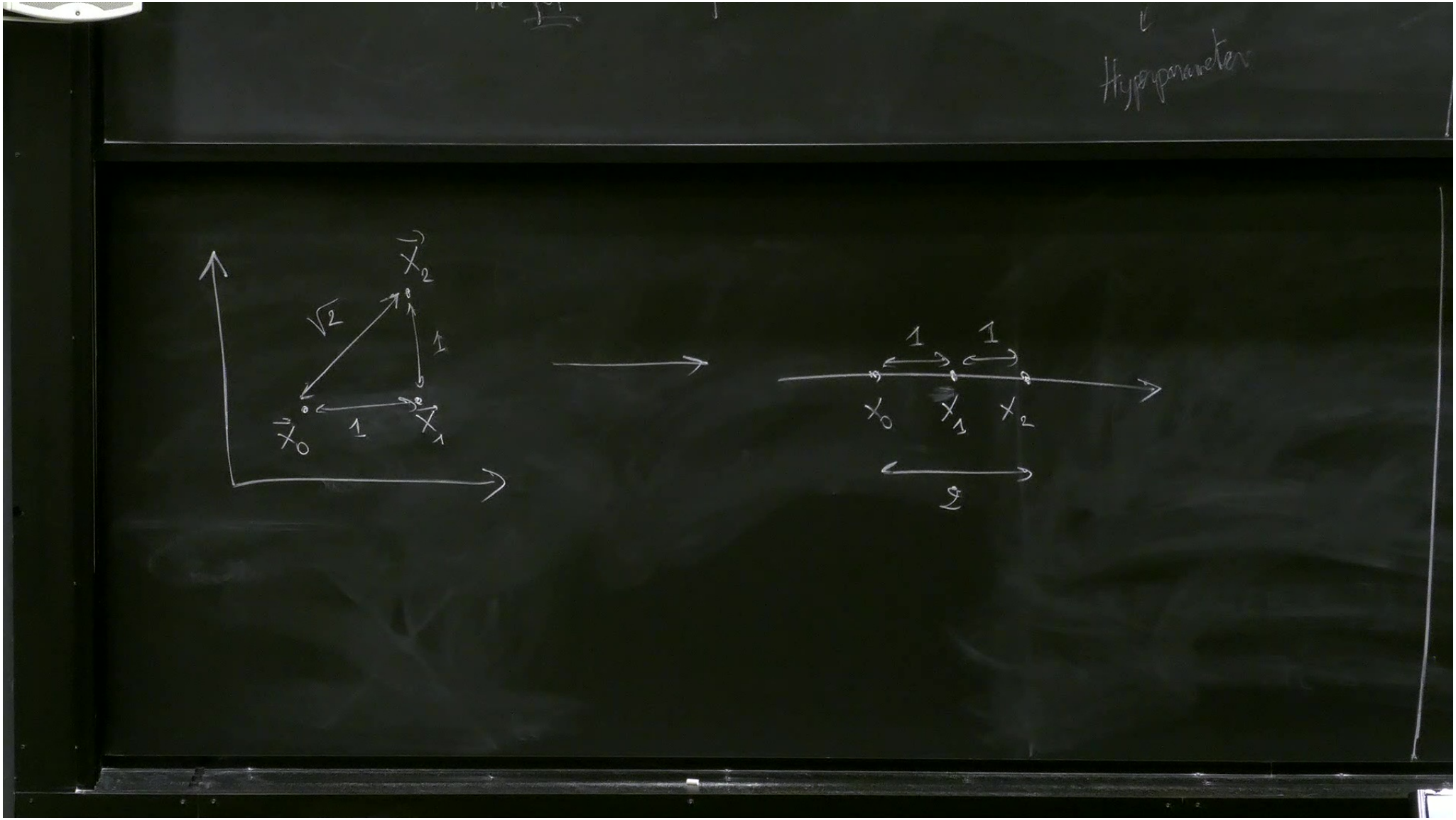

Remark
→ Data normalization
$\vec{X} = (x_1, x_2, \cdots, x_N)$
should have similar range (If not, we need rescaling).
e.g. spin = ±1 (same bounds)

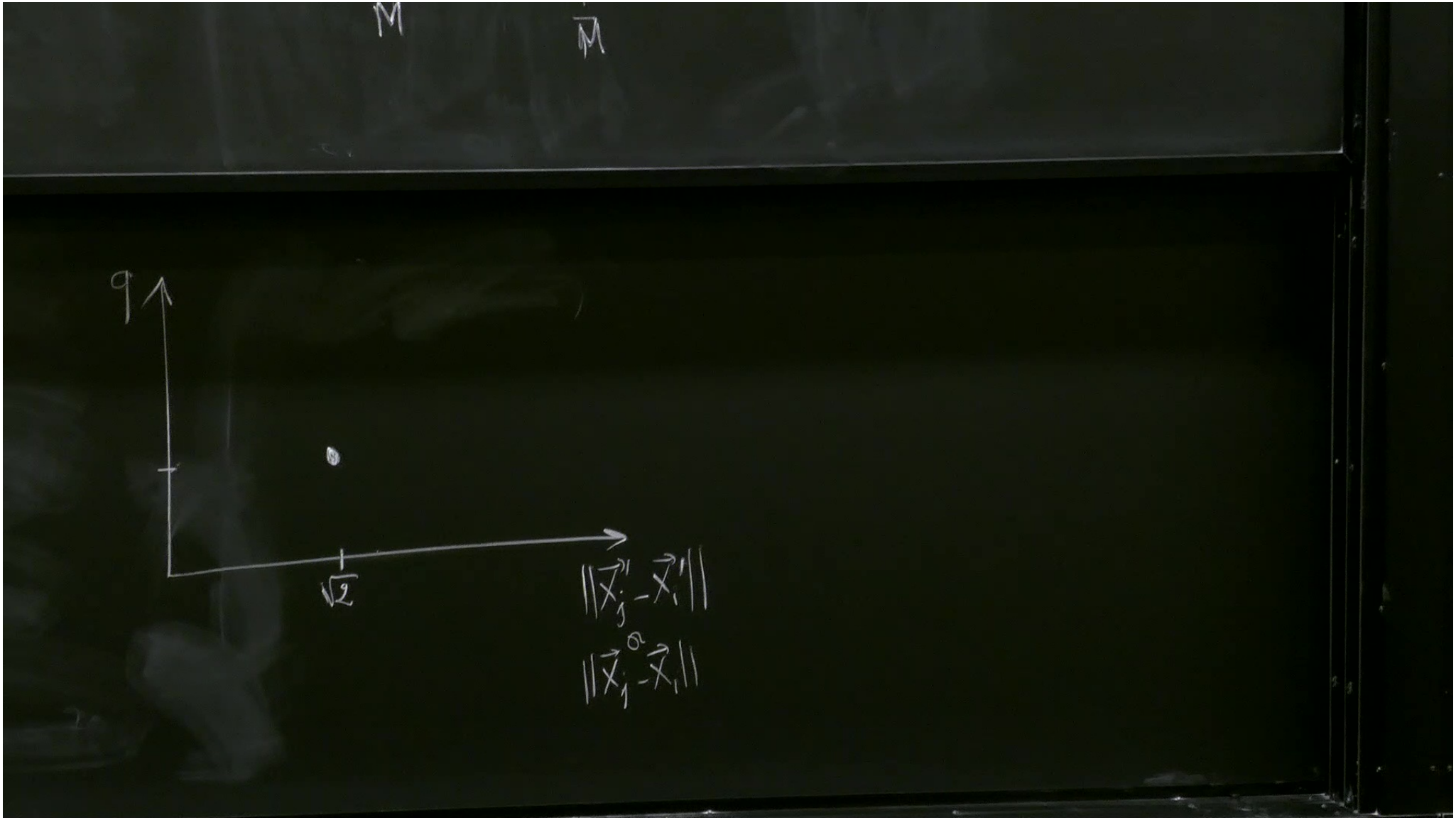

t-SNE
Given two datapoint $\vec{x}_i$, $\vec{x}_j$
$(j \neq i) \rightarrow P(j|i) = \frac{e^{-\|x_i - x_j\|^2 / 2\sigma_i^2}}{\sum_{k \neq i} e^{-\|x_i - x_k\|^2 / 2\sigma_i^2}}$
$P(i|i) = 0$
$\vec{x}_i$
The "σ_i" are hyperparameters that are related "Perplexity" → See HW2
&
Hyperparameter

Goal of t-SNE: find a lower dimensional representation $\vec{x}' \in \mathbb{R}^{N'}$ that preserves similarity distribution $p(i,j)$

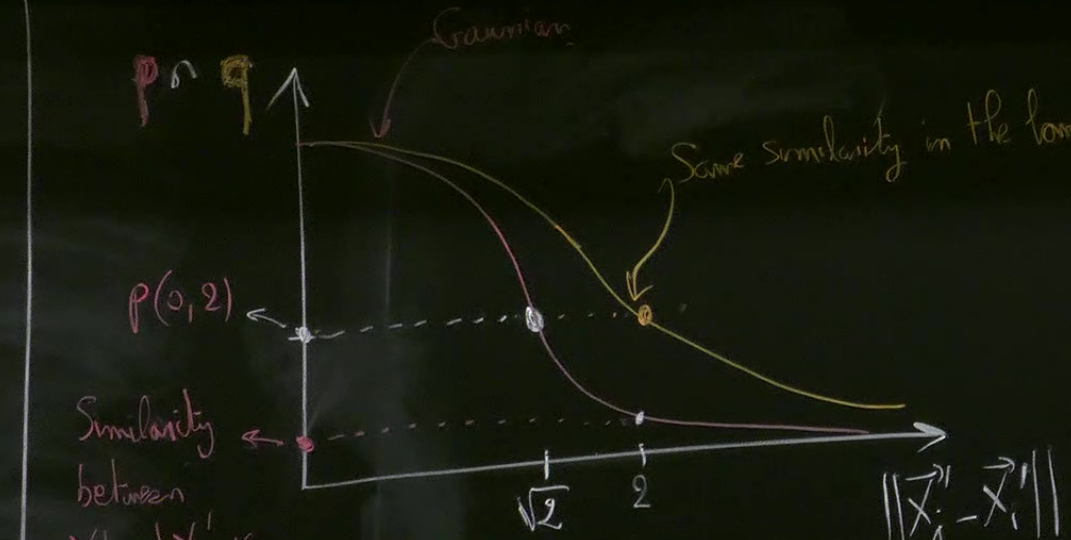
$$q(j|i) = \frac{f(\|\vec{x}_i' - \vec{x}_j'\|)}{\sum_{k \neq i} f(\|\vec{x}_i' - \vec{x}_k'\|)} \quad (i \neq j)$$

$q(i,j)$ defined in a similar way compared $p(i,j)$


Hyperparameter
x_2
$\sqrt{2}$
1
1
x_0
x_1
1
1
x_0
x_1
x_2
Σ

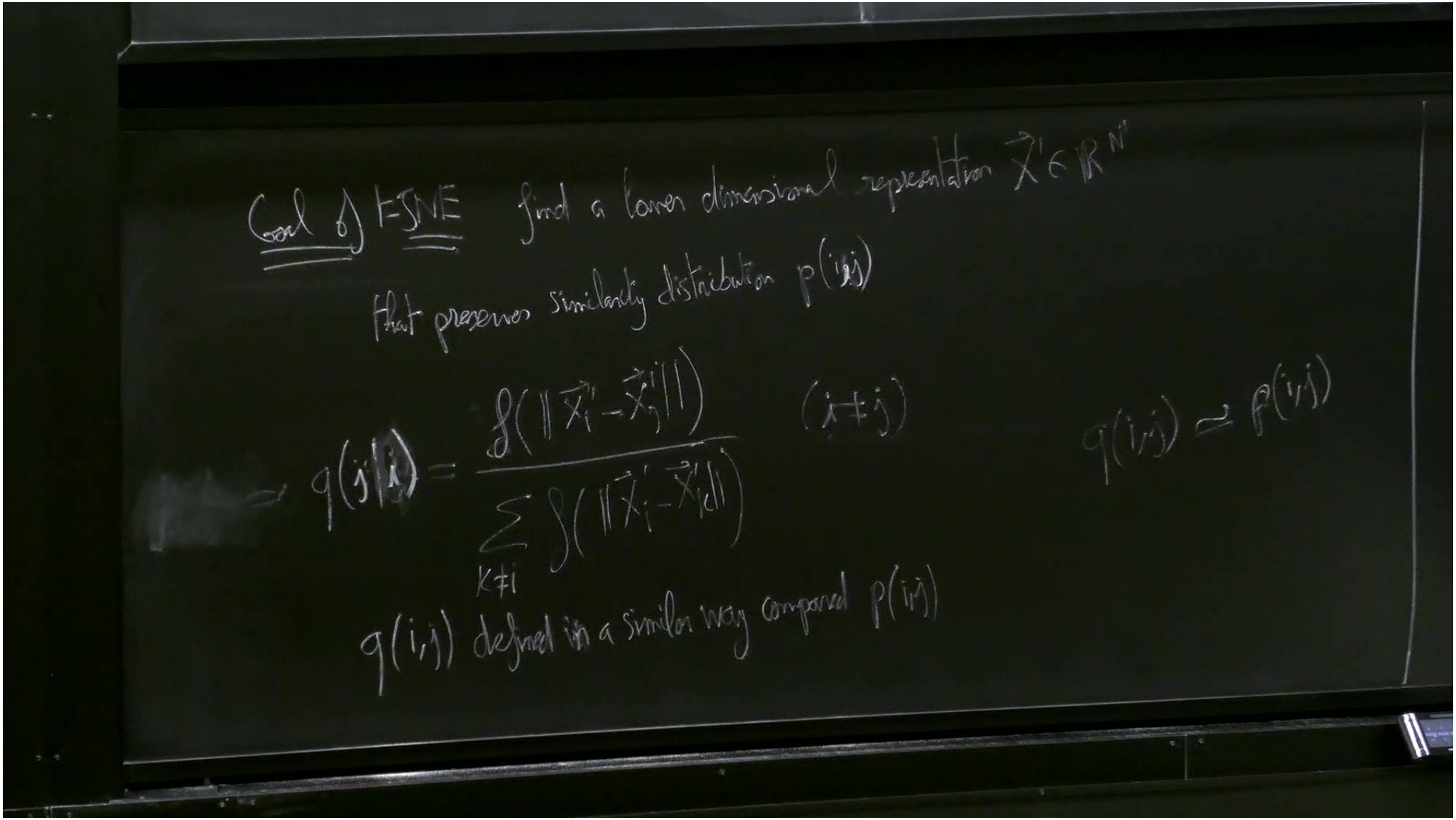

M
M
q
$\sqrt{2}$
$\|\vec{x}_j' - \vec{x}_i'\|$
$\|\vec{x}_j^{\,0} - \vec{x}_i\|$

M $\bar{M}$



$$f(z) = \frac{1}{1+z^2}$$

(t-Student distribution)


Goal of t-SNE find a lower dimensional representation $\vec{x}' \in \mathbb{R}^{N'}$
that preserves similarity distribution $p(i,j)$
$q(j|i) = \frac{f(\|\vec{x}_i' - \vec{x}_j'\|)}{\sum_{k \neq i} f(\|\vec{x}_i' - \vec{x}_k'\|)}$ $(i \neq j)$
$q(i,j) \sim p(i,j)$
$q(i,j)$ defined in a similar way compared $p(i,j)$

To find a lower-dimensional representation $D' = \{\vec{x}'_i\}$: iteratively update the points $\vec{x}'_i$ using gradient descent

$\vec{x}'_i$: Trainable parameter.

$$C(P|Q) = D_{KL}(P\|Q) = \sum_{i,j=1}^{N} P(i,j) \log\left(\frac{P(i,j)}{Q(i,j)}\right)$$

Kullback-Leibler (KL) divergence

To find a lower-dimensional representation $D' = \{\vec{x}'_i\}$: iteratively updated

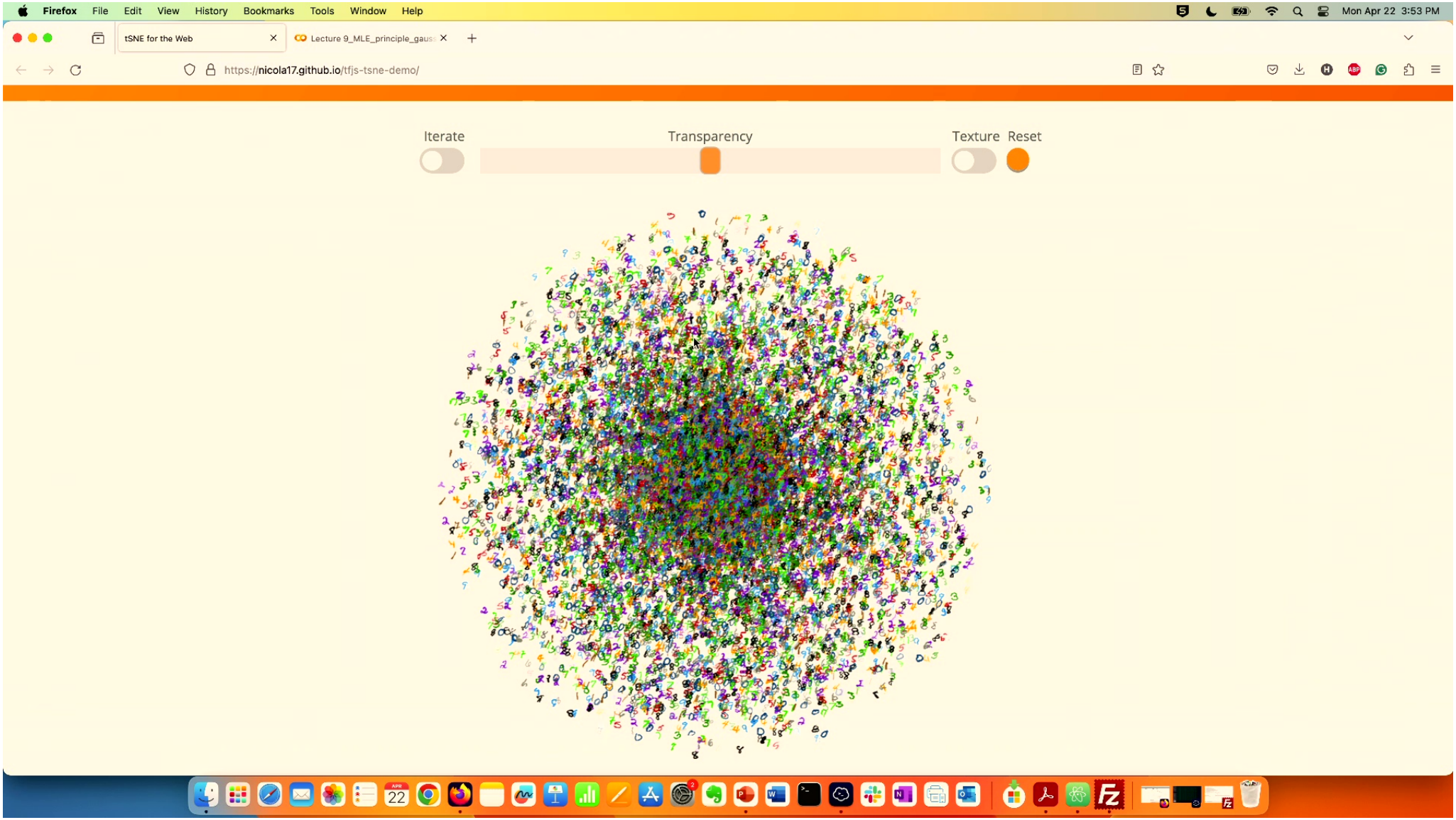
the points $\vec{x}'_i$ using gradient descent

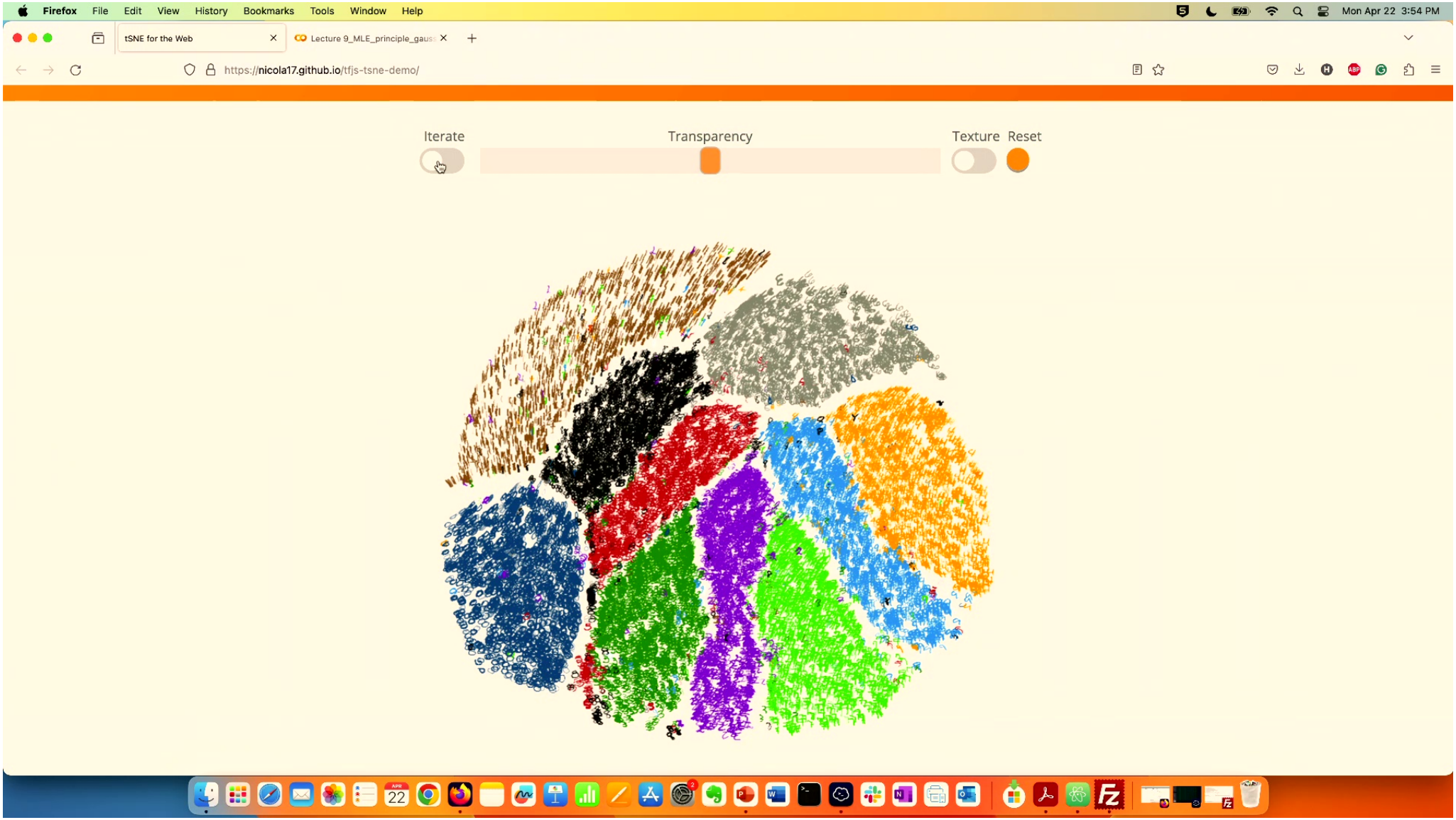
$\vec{x}'_i$: Trainable parameter.

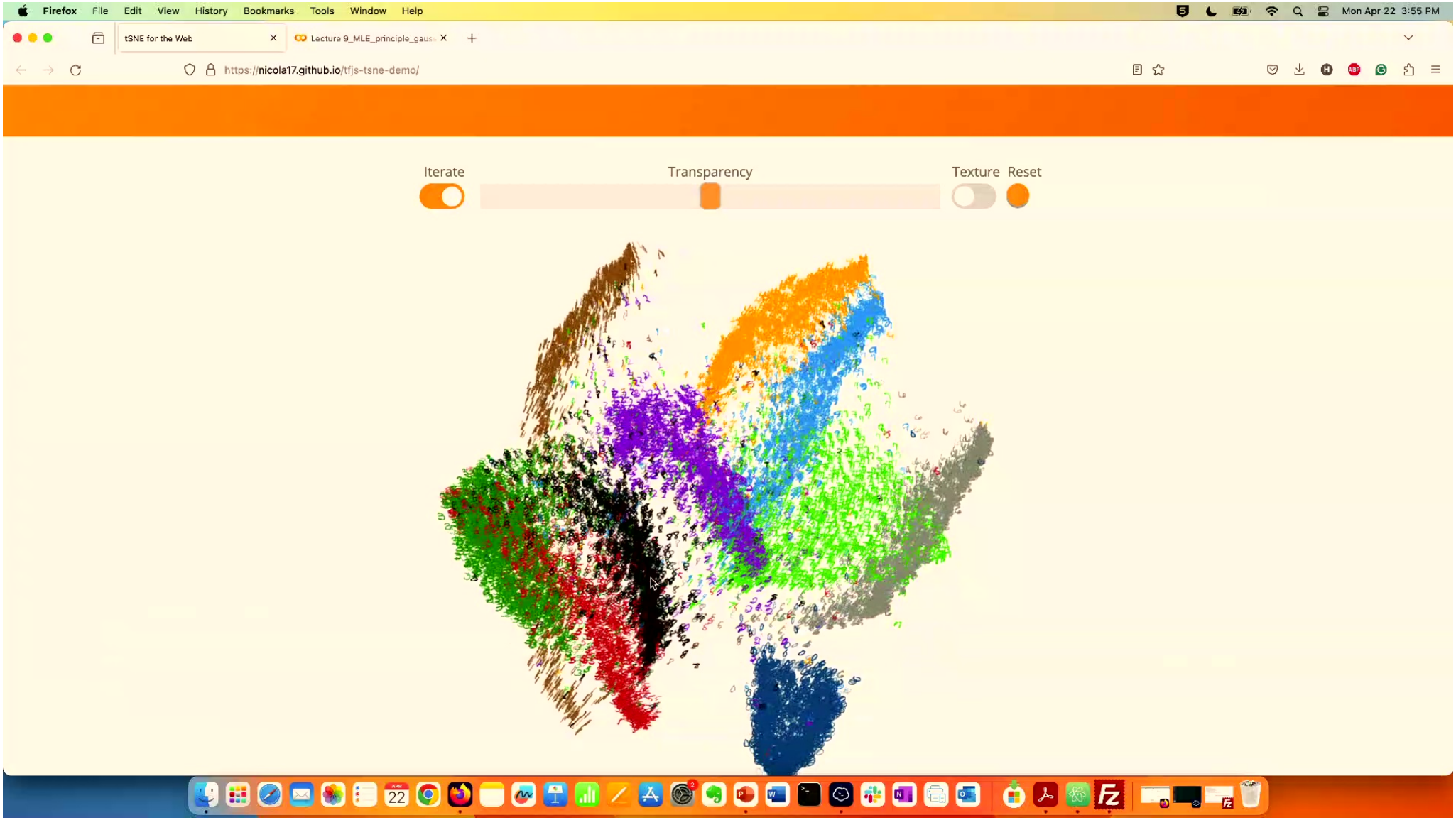
$$C(P\|Q) = D_{KL}(P\|Q) = \sum_{i,j=1}^{N} P(i,j) \log\left(\frac{P(i,j)}{Q(i,j)}\right)$$

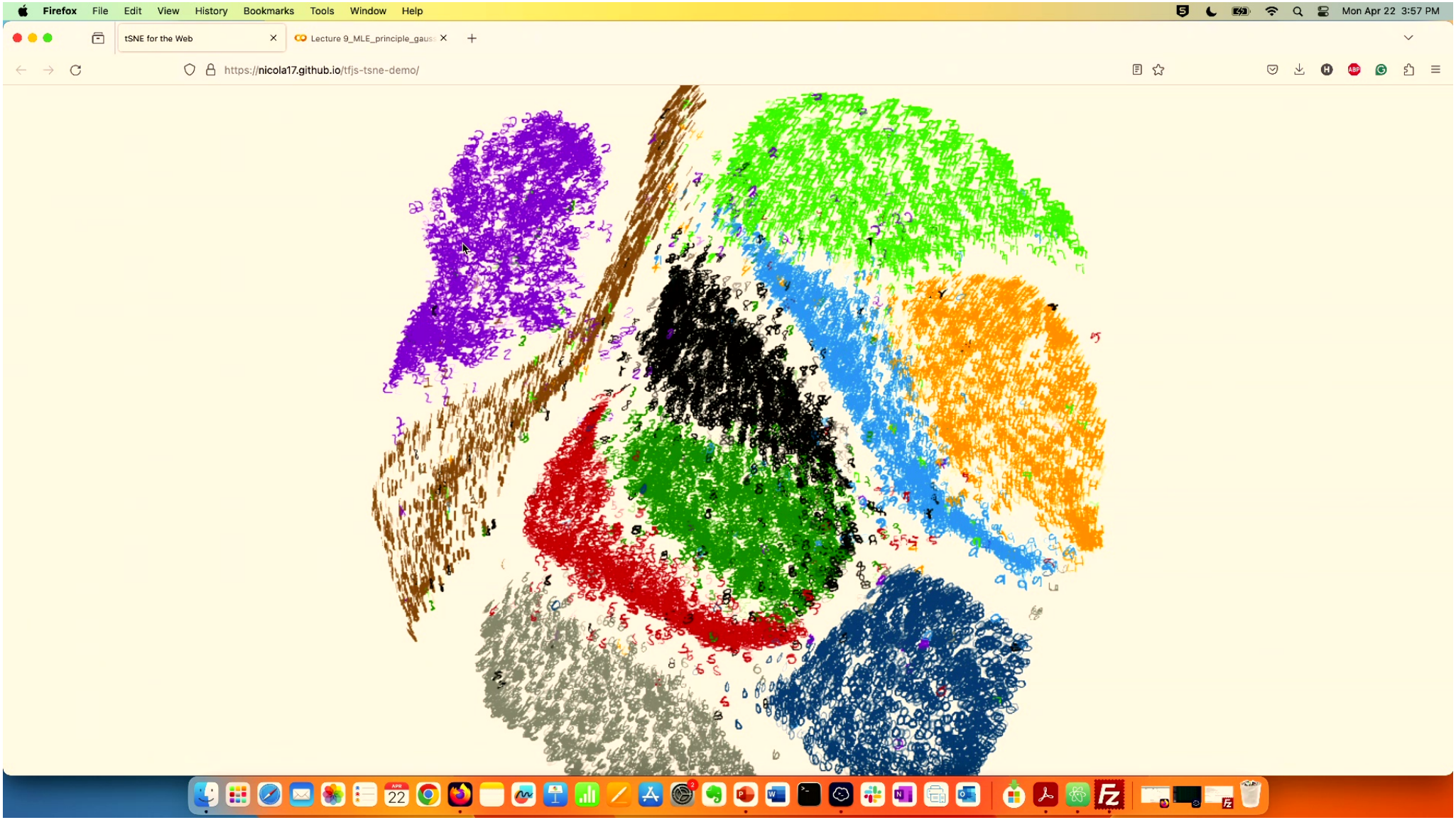
Kullback-Leibler (KL) divergence

$0 \log(0) = 0$


Firefox File Edit View History Bookmarks Tools Window Help
Mon Apr 22 3:53 PM
tSNE for the Web
Lecture 9_MLE_principle_gaus
https://nicola17.github.io/tfjs-tsne-demo/
Iterate
Transparency
Texture
Reset


Firefox File Edit View History Bookmarks Tools Window Help
Mon Apr 22 3:54 PM
tSNE for the Web
Lecture 9_MLE_principle_gaus
https://nicola17.github.io/tfjs-tsne-demo/
Iterate
Transparency
Texture
Reset


Firefox File Edit View History Bookmarks Tools Window Help
Mon Apr 22 3:55 PM
tSNE for the Web
Lecture 9_MLE_principle_gaus
https://nicola17.github.io/tfjs-tsne-demo/
Iterate
Transparency
Texture
Reset


Firefox File Edit View History Bookmarks Tools Window Help
Mon Apr 22 3:57 PM
tSNE for the Web
Lecture 9_MLE_principle_gaus
https://nicola17.github.io/tfjs-tsne-demo/

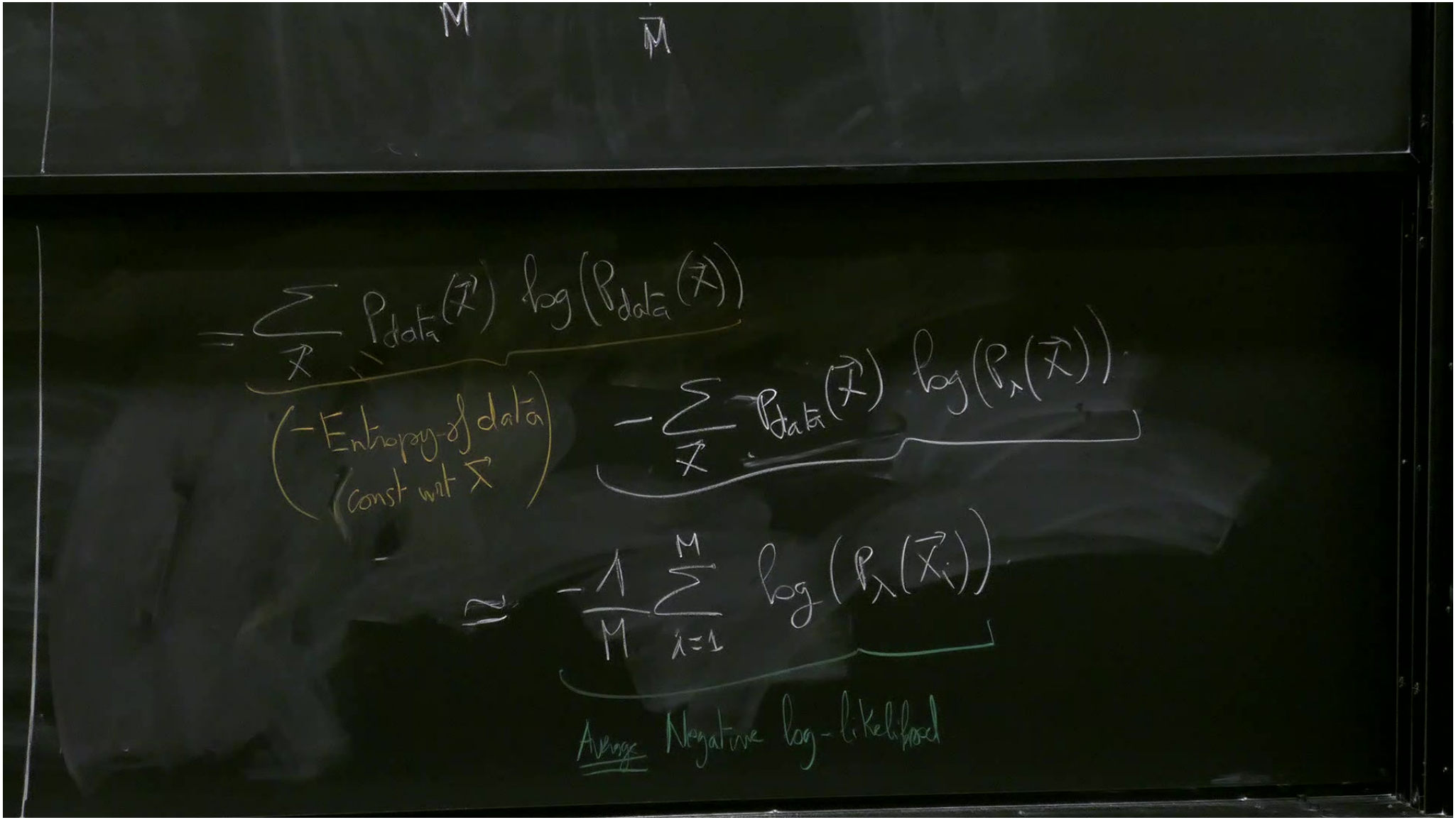
Hyperparameter

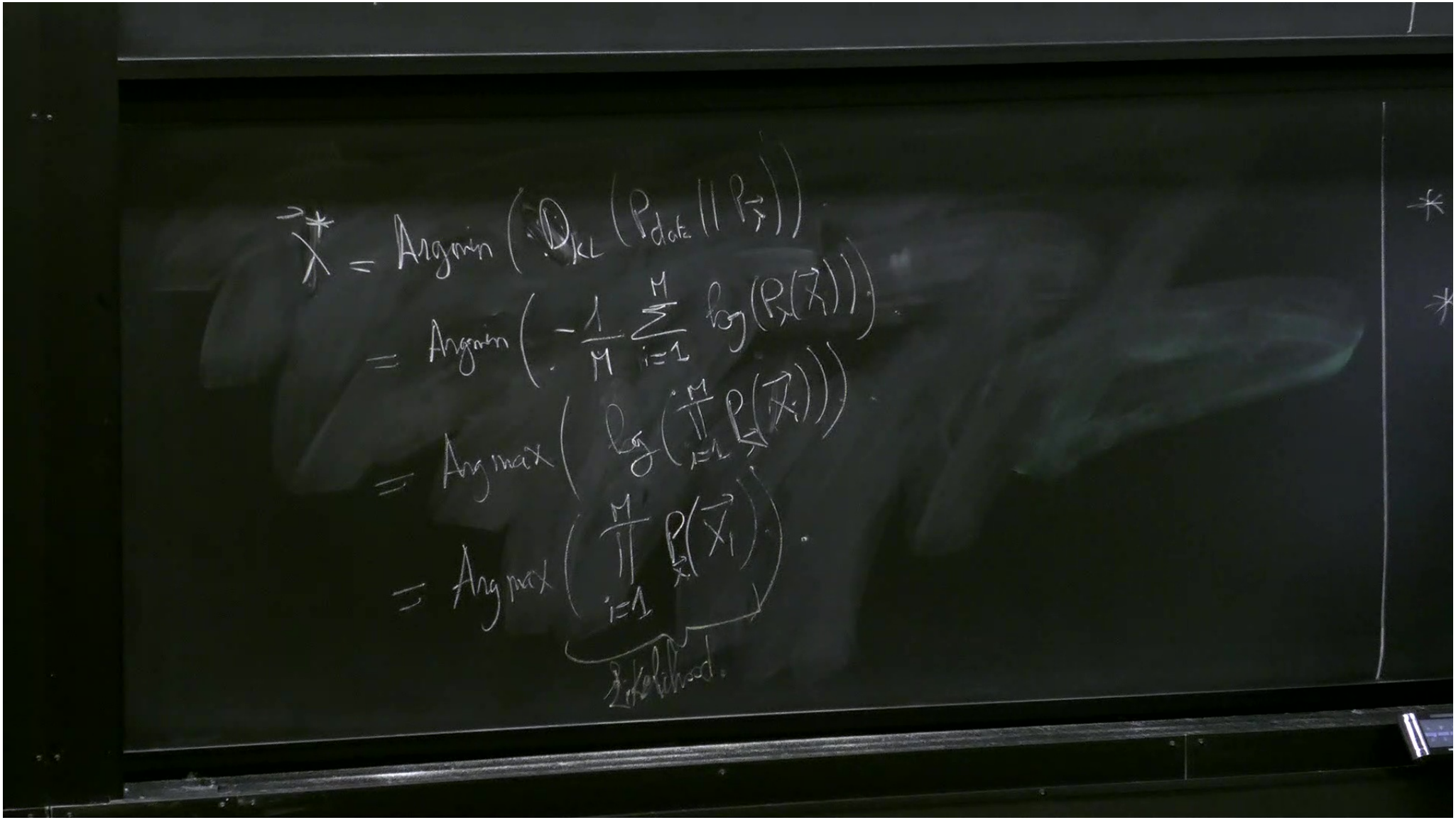
Generative modelling:

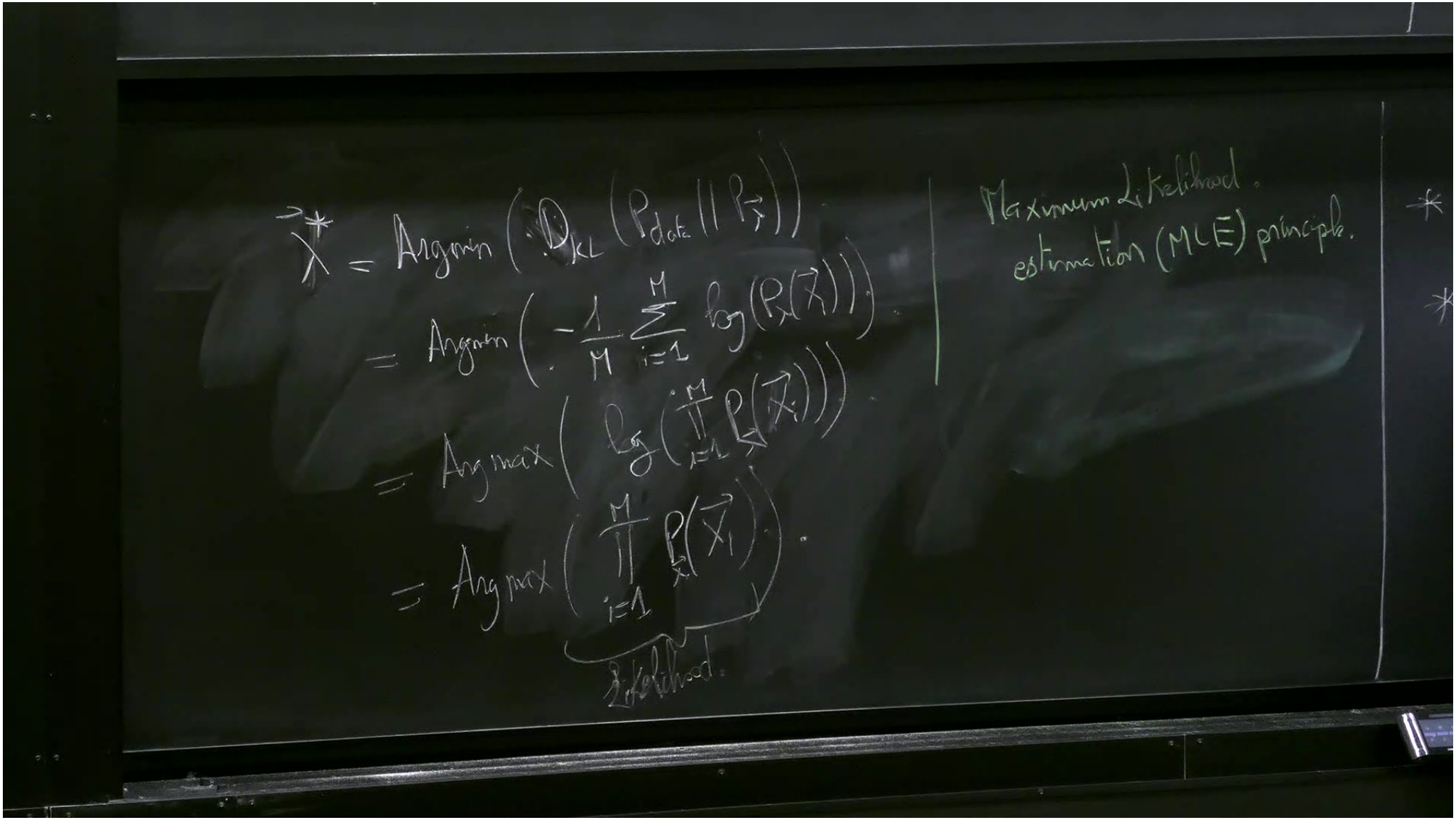
↳ Given datapoints $\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_M$ coming from $p_{data}(\vec{x})$

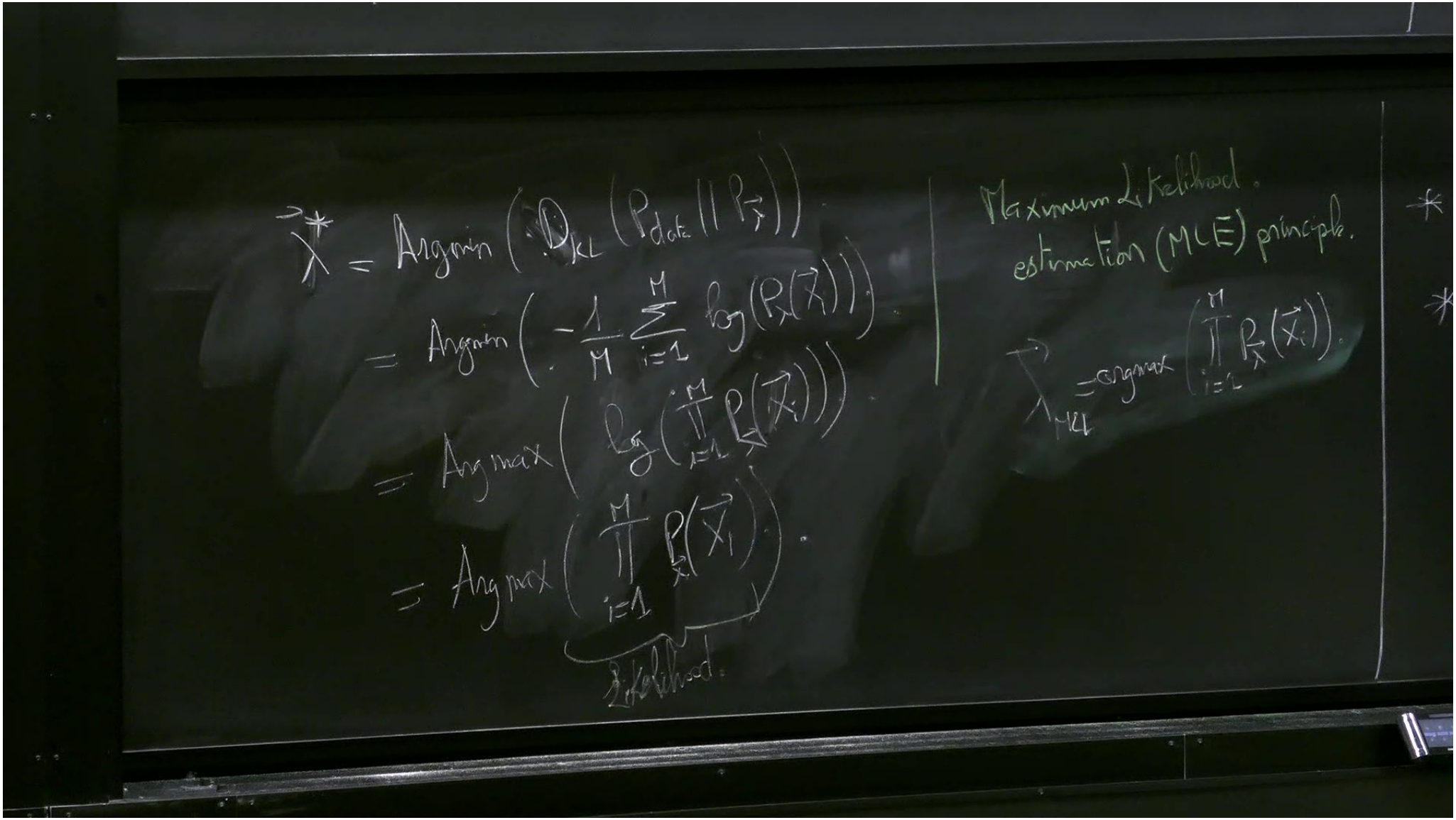
Goal: find parameters $\vec{\lambda}$ of a model $p_{\vec{\lambda}}(\vec{x}) \simeq p_{data}(\vec{x})$

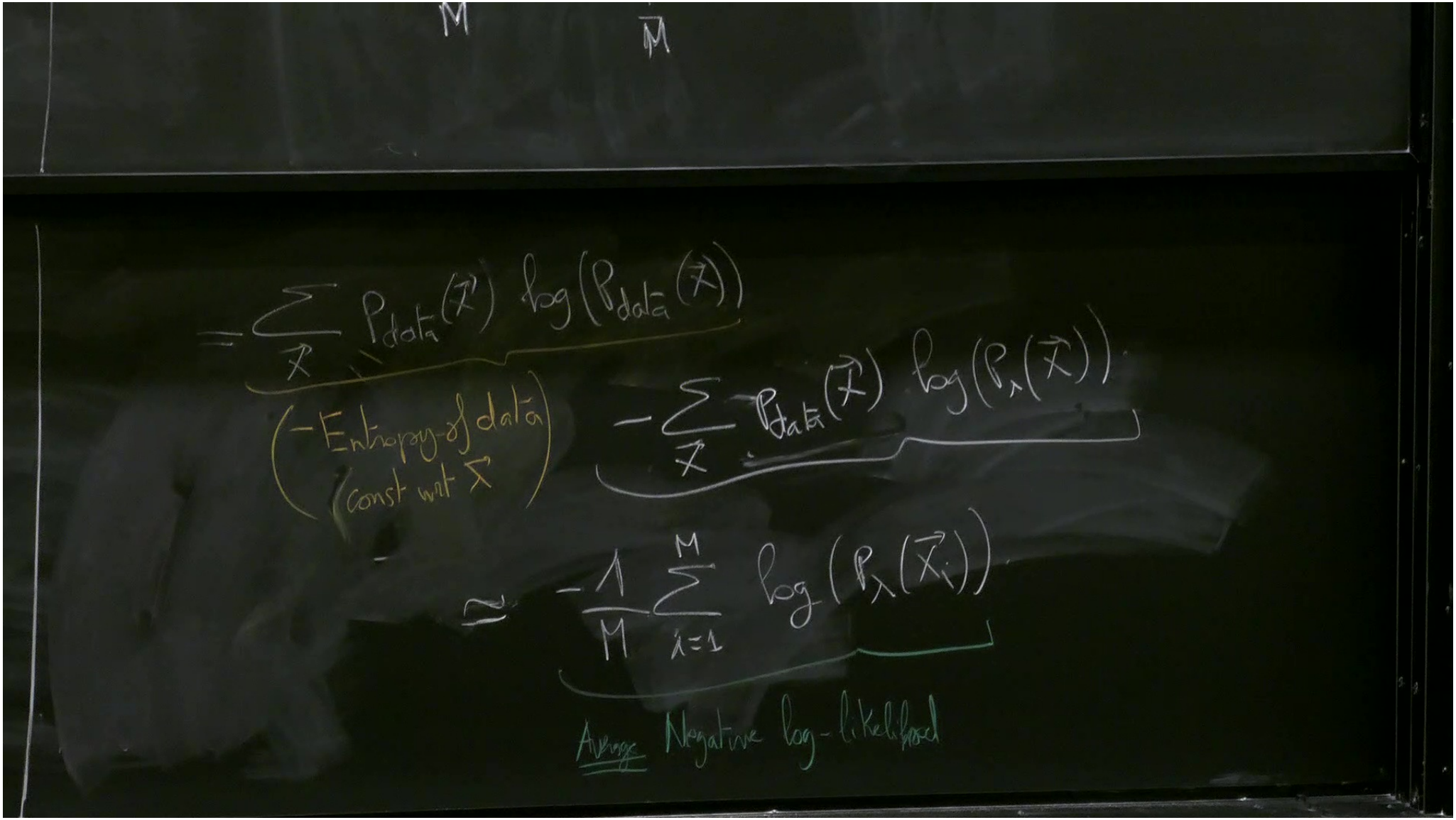
$$D_{KL}(p_{data} \,||\, p_{\vec{\lambda}}) = \sum_{\vec{x}} p_{data}(\vec{x}) \log\left(\frac{p_{data}(\vec{x})}{p_{\vec{\lambda}}(\vec{x})}\right)$$

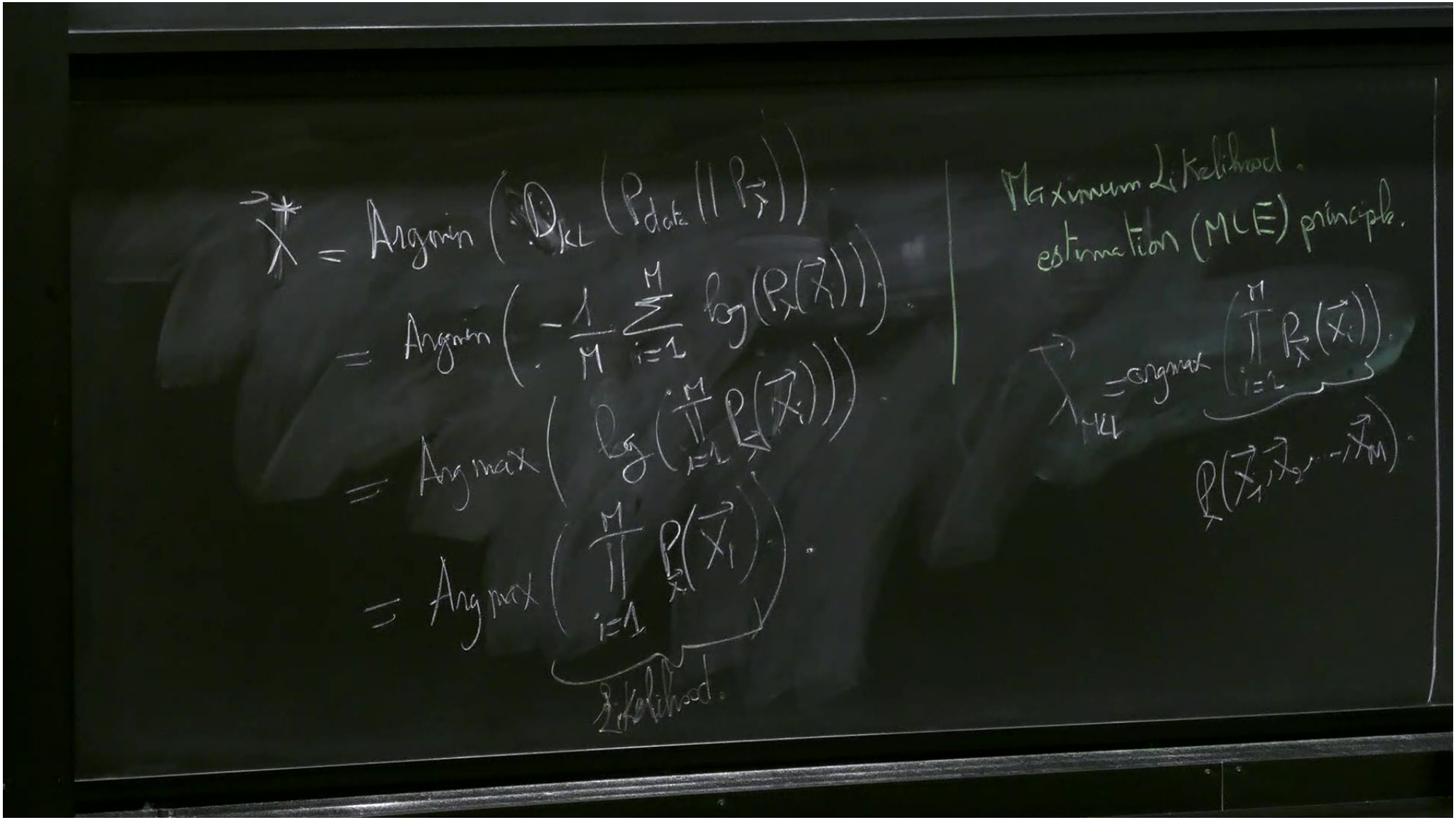

M
M
$= \sum_{\vec{x}} P_{data}(\vec{x}) \log(P_{data}(\vec{x}))$
(- Entropy of data
const wrt $\vec{x}$)
$- \sum_{\vec{x}} P_{data}(\vec{x}) \log(P_\lambda(\vec{x}))$
$\simeq -\frac{1}{M} \sum_{i=1}^{M} \log(P_\lambda(\vec{x}_i))$
Average Negative log-likelihood


$\vec{\lambda}^* = \text{Argmin}\left(D_{KL}(P_{data} \,||\, P_{\vec{\lambda}})\right)$
$= \text{Argmin}\left(-\frac{1}{M}\sum_{i=1}^{M} \log(P_{\vec{\lambda}}(\vec{X}_i))\right)$
$= \text{Argmax}\left(\log\left(\prod_{i=1}^{M} P_{\vec{\lambda}}(\vec{X}_i)\right)\right)$
$= \text{Argmax}\left(\prod_{i=1}^{M} P_{\vec{\lambda}}(\vec{X}_i)\right)$
Likelihood.


$\vec{\lambda}^* = \text{Argmin}\left(D_{KL}(P_{data} \,||\, P_{\vec{\lambda}})\right)$
$= \text{Argmin}\left(-\frac{1}{M}\sum_{i=1}^{M} \log(P_{\vec{\lambda}}(\vec{X}_i))\right)$
$= \text{Argmax}\left(\log\left(\prod_{i=1}^{M} P_{\vec{\lambda}}(\vec{X}_i)\right)\right)$
$= \text{Argmax}\left(\prod_{i=1}^{M} P_{\vec{\lambda}}(\vec{X}_i)\right)$
Likelihood.
Maximum Likelihood estimation (MLE) principle.


$\vec{\lambda}^* = \text{Argmin}\left(D_{KL}(P_{data} \,||\, P_{\vec{\lambda}})\right)$
$= \text{Argmin}\left(-\frac{1}{M}\sum_{i=1}^{M} \log(P_{\vec{\lambda}}(\vec{X}_i))\right)$
$= \text{Argmax}\left(\log\left(\prod_{i=1}^{M} P_{\vec{\lambda}}(\vec{X}_i)\right)\right)$
$= \text{Argmax}\left(\prod_{i=1}^{M} P_{\vec{\lambda}}(\vec{X}_i)\right)$
Likelihood.
Maximum Likelihood estimation (MLE) principle.
$\vec{\lambda}_{MLE} = \text{argmax}\left(\prod_{i=1}^{M} P_{\vec{\lambda}}(\vec{X}_i)\right)$


M
M
$= \sum_{\vec{x}} P_{data}(\vec{x}) \log\left(P_{data}(\vec{x})\right)$
$\left(-\text{Entropy of data}\right)$
const wrt Σ
$- \sum_{\vec{x}} P_{data}(\vec{x}) \log\left(P_\lambda(\vec{x})\right)$
$\simeq -\frac{1}{M} \sum_{i=1}^{M} \log\left(P_\lambda(\vec{x}_i)\right)$
Average Negative log-likelihood


$\vec{\lambda}^* = \text{Argmin}\left(D_{KL}(P_{data} \,||\, P_{\vec{\lambda}})\right)$
$= \text{Argmin}\left(-\frac{1}{M}\sum_{i=1}^{M} \log(P_{\vec{\lambda}}(\vec{x}_i))\right)$
$= \text{Argmax}\left(\log\left(\prod_{i=1}^{M} P_{\vec{\lambda}}(\vec{x}_i)\right)\right)$
$= \text{Argmax}\left(\prod_{i=1}^{M} P_{\vec{\lambda}}(\vec{x}_i)\right)$
Likelihood.
Maximum Likelihood estimation (MLE) principle.
$\vec{\lambda}_{MLE} = \text{argmax}\left(\prod_{i=1}^{M} P_{\vec{\lambda}}(\vec{x}_i)\right)$
$P(\vec{x}_1, \vec{x}_2, \ldots \vec{x}_M)$

Lecture 9_MLE_principle_gaussian_distribution.ipynb

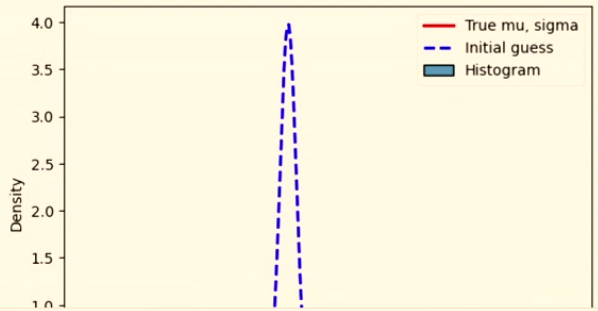
```
[ ] import numpy as np
    import matplotlib.pyplot as plt
    import seaborn as sns
    from scipy.stats import norm

    # True parameters
    mu_true = 1
    sigma_true = 1

    # Generate random data
    np.random.seed(0)
    data = np.random.normal(mu_true, sigma_true, 100)

[ ] initial_guess = [0.5, 0.1] #Initial ansatz for mu and sigma

    sns.histplot(data, kde=False, stat="density", label='Histogram')
    x = np.linspace(-2, 4, 300)
    plt.plot(x, norm.pdf(x, mu_true, sigma_true), 'r-', lw=2, label='True mu, sigma')
    plt.plot(x, norm.pdf(x, initial_guess[0], initial_guess[1]), 'b--', lw=2, label='Initial guess')
    plt.legend()
    plt.show()
```



Connected to Python 3 Google Compute Engine backend

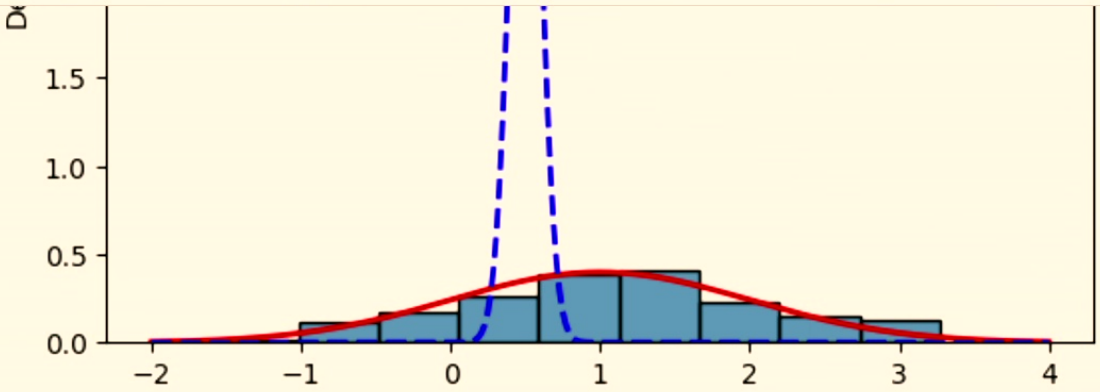
```
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import norm
```

```
# True parameters
mu_true = 1
sigma_true = 1

# Generate random data
np.random.seed(0)
data = np.random.normal(mu_true, sigma_true, 100)
```

```
initial_guess = [0.5, 0.1] #Initial ansatz for mu and sigma
```

```
sns.histplot(data, kde=False, stat="density", label='Histogram')
x = np.linspace(-2, 4, 300)
plt.plot(x, norm.pdf(x, mu_true, sigma_true), 'r-', lw=2, label='True mu, sigma')
plt.plot(x, norm.pdf(x, initial_guess[0], initial_guess[1]), 'b--', lw=2, label='Initial guess')
plt.legend()
plt.show()
```




```
from scipy.optimize import minimize

def neg_log_likelihood(params):
    mu, sigma = params
    return -np.sum(np.log(norm.pdf(data, mu, sigma)))

result = minimize(neg_log_likelihood, initial_guess, bounds=[(-1, 1), (0.1, 2)])
mle_mu, mle_sigma = result.x
```

```
sns.histplot(data, kde=False, stat="density", label='Histogram')
x = np.linspace(-2, 4, 300)
plt.plot(x, norm.pdf(x, mu_true, sigma_true), 'r-', lw=2, label='True mu, sigma')
plt.plot(x, norm.pdf(x, mle_mu, mle_sigma), 'b--', lw=2, label='MLE mu, sigma')
```

```
plt.plot(x, norm.pdf(x, mle_mu, mle_sigma), 'b--', lw=2, label='MLE mu, sigma')
plt.legend()
plt.show()
```