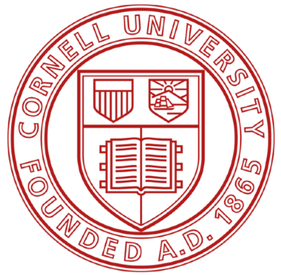
Lecture 3: Gaussian mixture models

CS 5788: Introduction to Generative Models



Reminders

- PS1 out (due Feb. 10)
- Office hours start this week
- Due to travel, Thursday's lecture will be a recording.
  - It is a neural net review, for those who would like to catch up.
  - CNNs, MLPs, Backprop (not covered: transformers and attention).

Recall: Multivariate Gaussian

Probability density function (pdf):

$$p_\theta(x) = \frac{1}{\sqrt{(2\pi)^k \det(\boldsymbol{\Sigma})}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right)$$

Parameters θ: mean $\boldsymbol{\mu}$, covariance matrix $\boldsymbol{\Sigma}$.

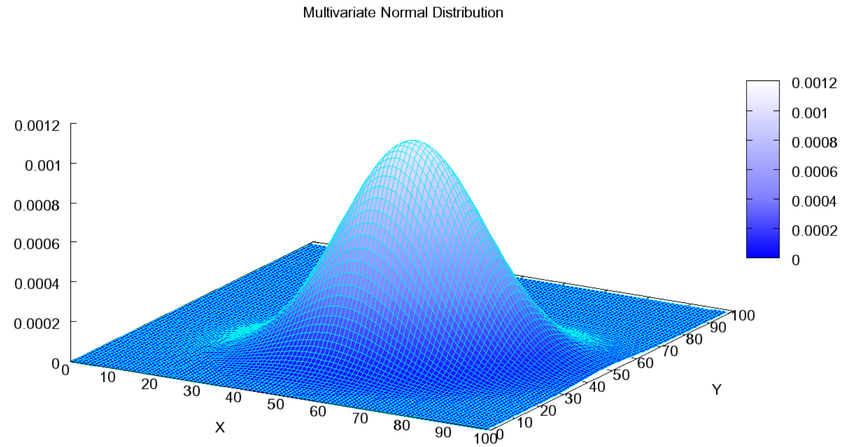


Figure source: Wikipedia

Recall: maximum likelihood estimation (MLE)

Goal: Find best parameters θ.

How do we quantify this? One option is **maximum likelihood estimation**.

Suppose we have a sample of points $x_1, x_2, \ldots, x_N$ from the distribution. We can find θ by maximizing their likelihood under the model:

$$\text{argmax}_\theta \prod_{i=1}^{N} p_\theta(x_i)$$

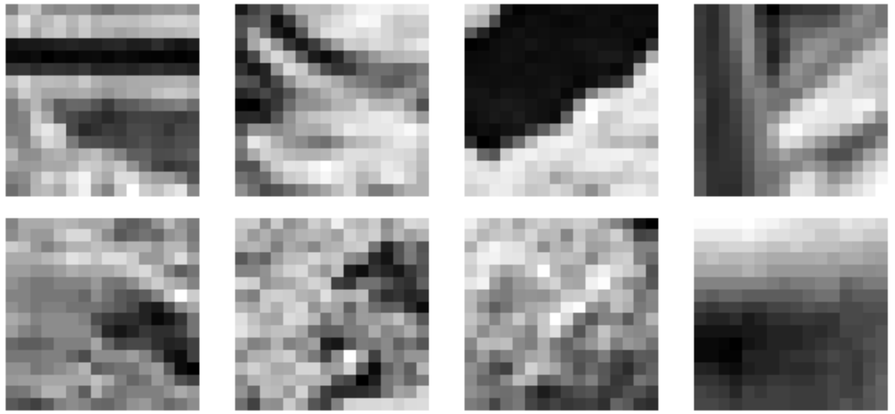
Assuming data is **iid** (independent and identically distributed).

$$= \text{argmax}_\theta \sum_{i=1}^{N} \log(p_\theta(x_i))$$

Equivalent to *minimizing* the **negative log likelihood:** $\text{NLL}(\theta) = -\sum_{i=1}^{N} \log(p_\theta(x_i))$

Recall: Fitting a multivariable Gaussian to image patches

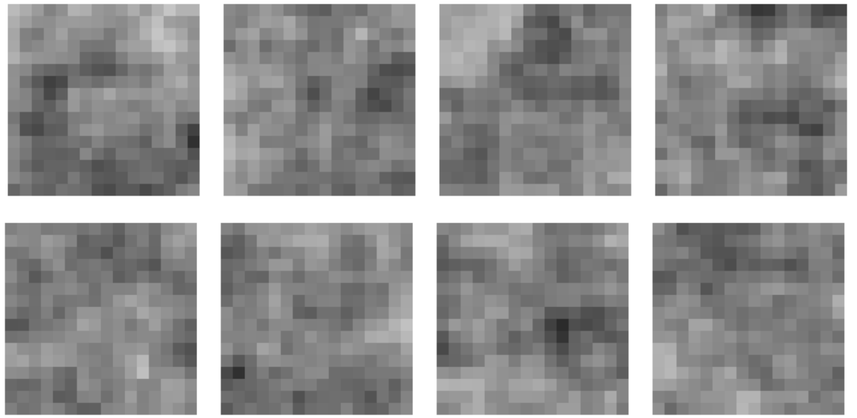
Training image patches (16×16)



Flatten each patch to get a 256-dimensional vector.

$$\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$$

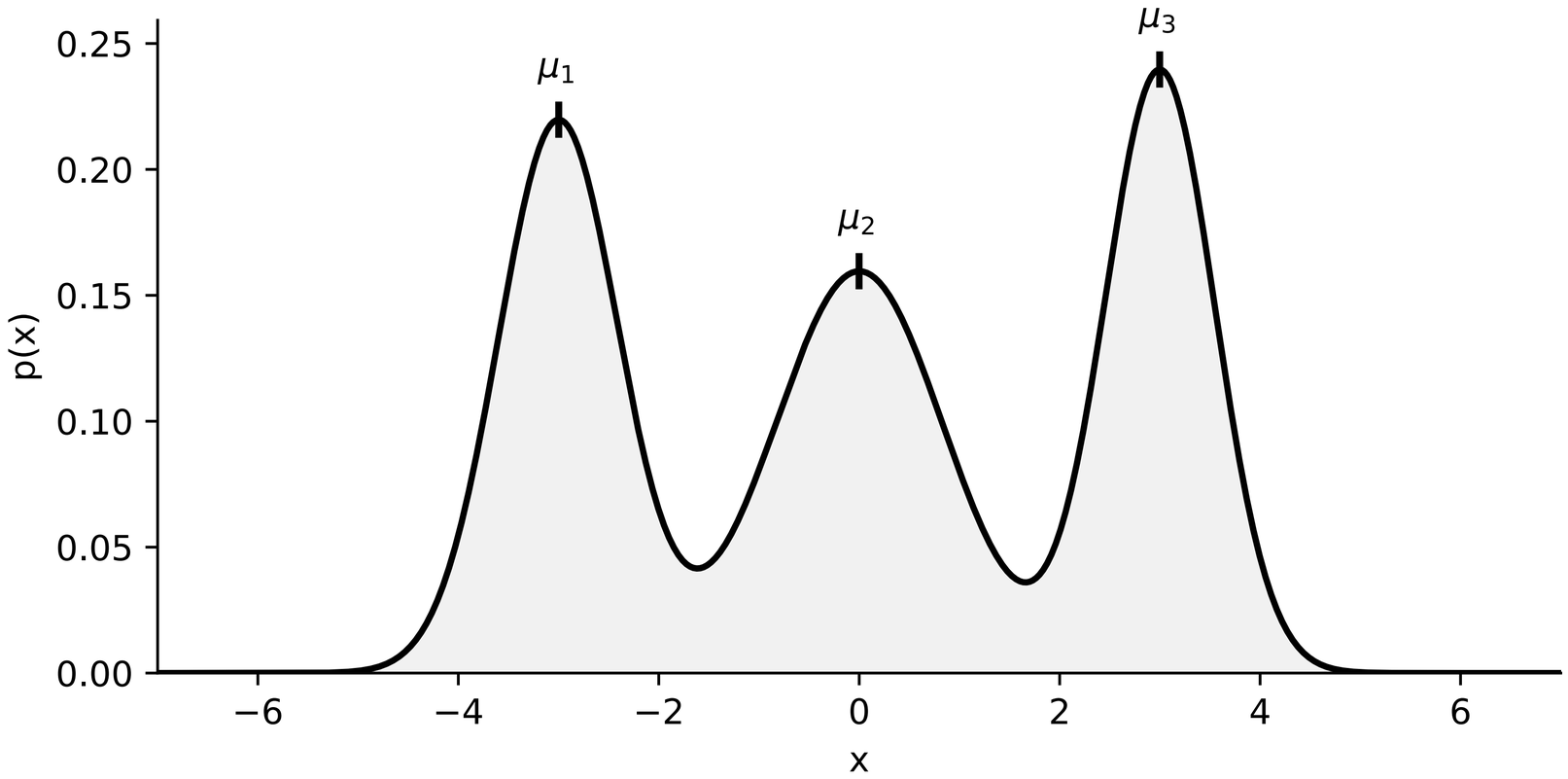
Samples from Gaussian



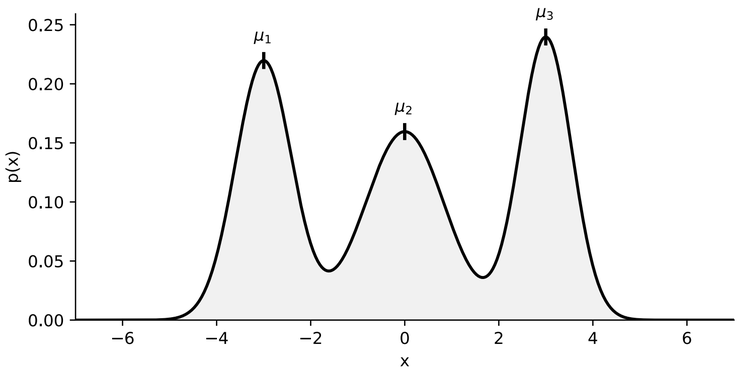
(not so great!)

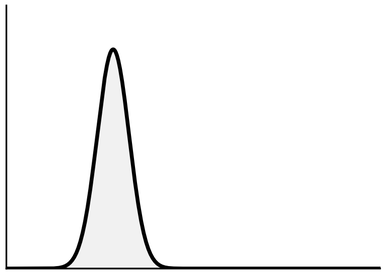
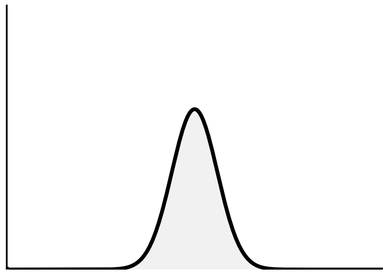
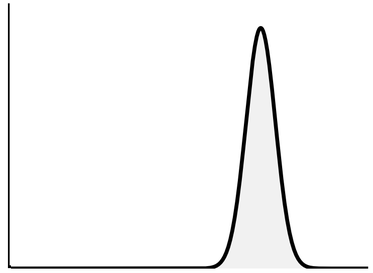
How can we improve this?

Gaussian mixture model



Gaussian mixture model



$= \pi_1 \times$  $+\pi_2 \times$  $+\pi_3 \times$ 

Mixing distributions:
(more generally)

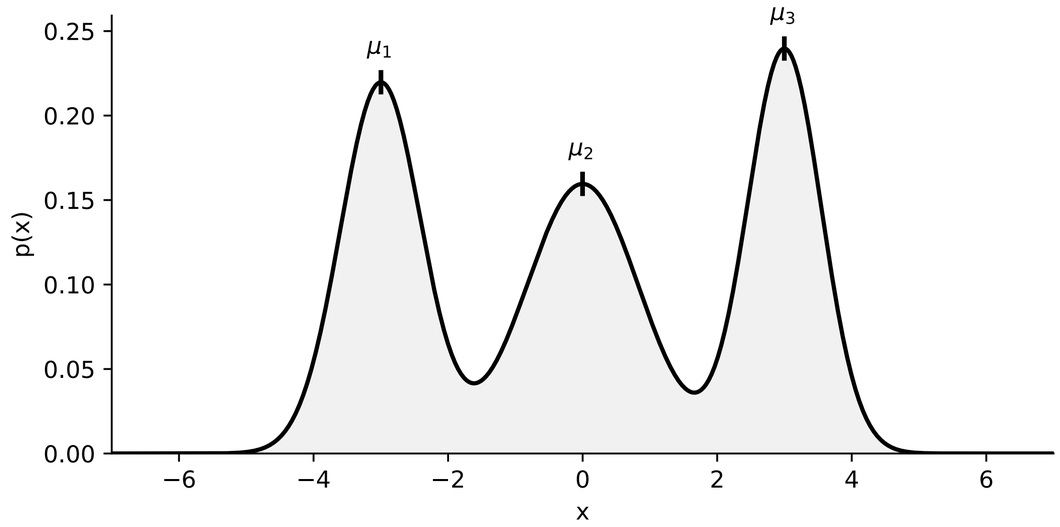
$$p_\theta(\mathbf{x}) = \sum_{i=1}^{K} \pi_i q_i(\mathbf{x}; \theta)$$ for non-negative weights $$\sum_i \pi_i = 1$$

Gaussian mixture model (GMM)

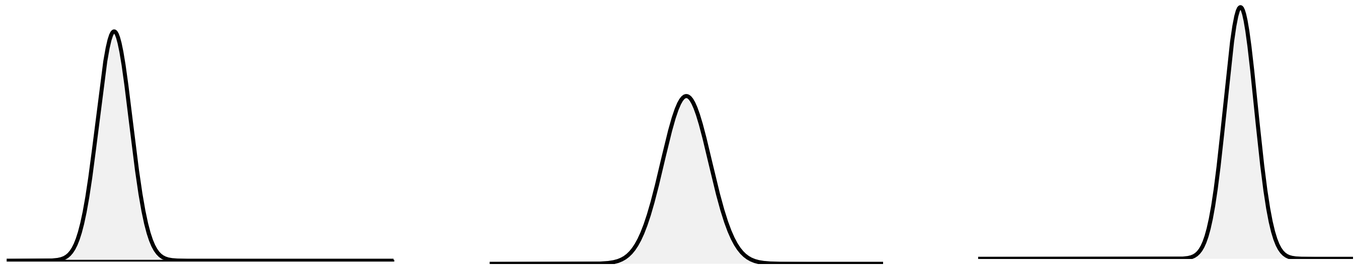
A useful type of mixture model that mixes multiple Gaussian distributions.

Density function: $p_\theta(\mathbf{x}) = \sum_{i=1}^{K} \pi_i \mathcal{N}(\mathbf{x}; \boldsymbol{\mu_i}, \boldsymbol{\Sigma}_i)$

where π_i is the probability of choosing mixture component i, and $\mathcal{N}$ is the Gaussian pdf with mixture-specific mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$.



Sampling from a GMM

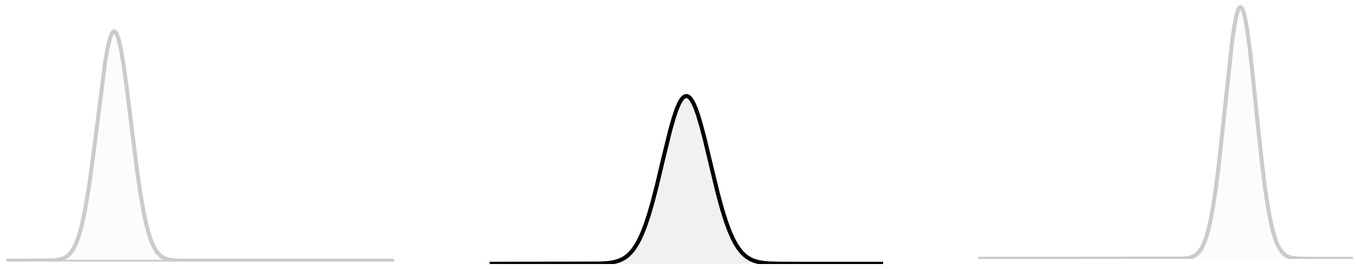


- Step #1: select mixture component:

$$z \sim \text{Categorical}(\pi_1, \pi_2, \ldots, \pi_K)$$

i.e., $p(z = 1) = \pi_1$, $p(z = 2) = \pi_2$, …, $p(z = K) = \pi_K$

Sampling from a GMM



- Step #1: select mixture component:

$$z \sim \text{Categorical}(\pi_1, \pi_2, \ldots, \pi_K)$$

- Step #2: sample from the chosen component

$$\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}_z, \boldsymbol{\Sigma}_z)$$

GMM as a latent variable model

- The variable z is the identity of the mixture component that generated the example. We call it a **latent variable** because we never directly observe it.
- Latent variables allow us to create more powerful generative models.
- They allow us to capture complex dependencies between variables.
- We'll see many other types of latent variable models in this course.

Adapted from R. Grosse

How do we learn a GMM?

How do we learn a GMM?

- We already know how to fit the parameters for a Gaussian. Can we take advantage of that?
- Idea #1: let's use an alternating optimization (similar to k-means).

do we want to make a hard assignment?

Loop:

- Assign each point $\mathbf{x}_i$ to a component z_i.
- Fit parameters $(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ using all examples assigned to component k.

How do we learn a GMM?

- Idea #2: a more principled alternating optimization that uses "soft" assignments.

Loop:

- For each point $\mathbf{x}_i$, compute $p(z_i = k \mid \mathbf{x}_i)$.
- Fit parameters $(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ using all examples "softly assigned" to component k using $p_\theta(z_i = k \mid \mathbf{x}_i)$

Let's make this idea more precise.

Learning a GMM

- Consider the *responsibility* of component k for example $\mathbf{x}_i$

$$r_{ik} = p_\theta(z_i = k \mid \mathbf{x}_i) = \frac{\pi_k \mathcal{N}(\mathbf{x}_i; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\mathbf{x}_i; \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}$$

- Let's perform maximum likelihood estimation:

$$\text{argmax}_\theta \sum_{i=1}^{N} \log(p_\theta(\mathbf{x}_i))$$

using these responsibilities as "soft assignments".

Learning a GMM

$$\log(p_\theta(\mathbf{x}_i)) = \log \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_i; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

$$= \log \sum_{k=1}^{K} r_{ik} \pi_k \frac{\mathcal{N}(\mathbf{x}_i; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{r_{ik}}$$

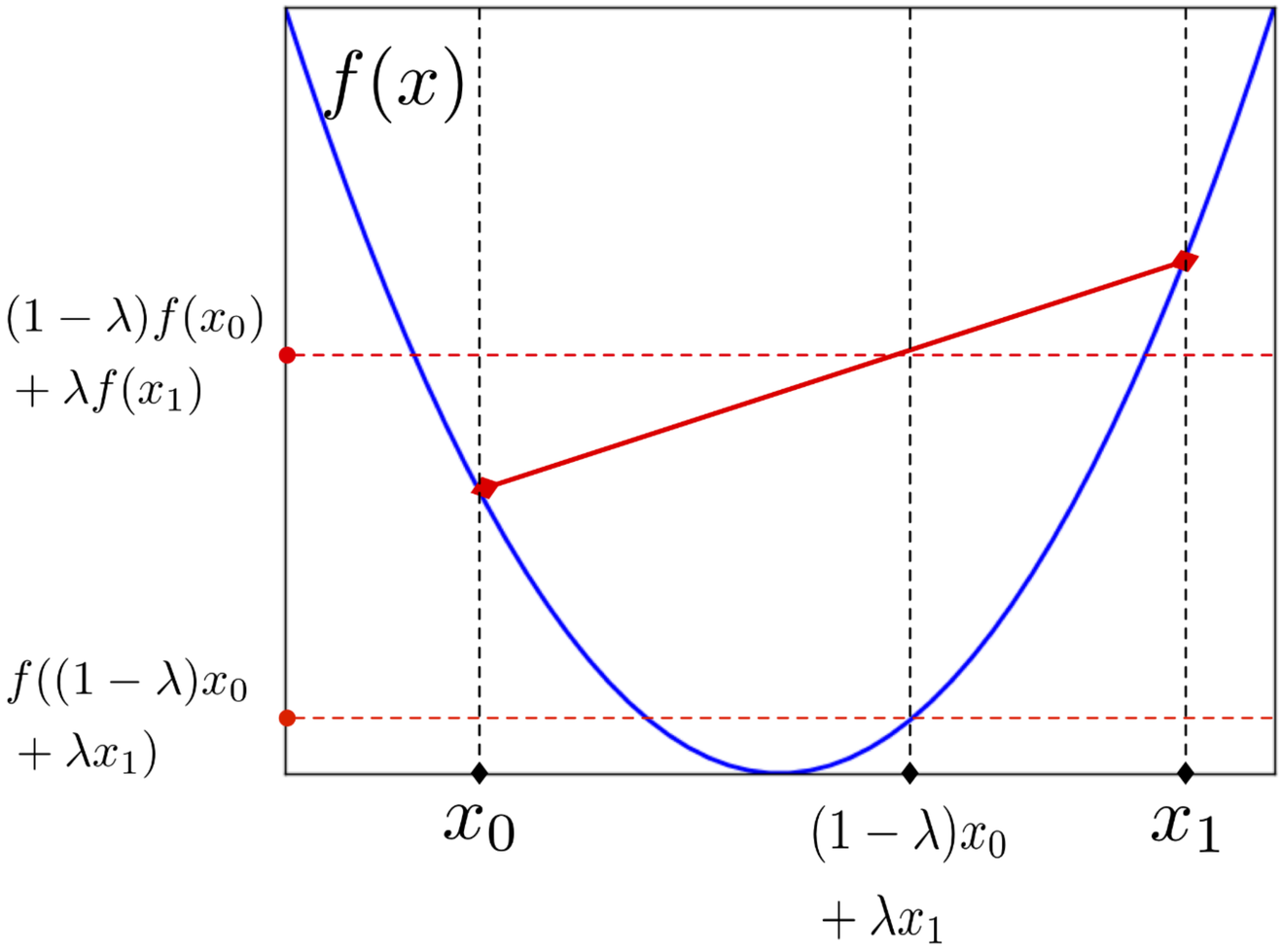
Jensen's inequality for concave functions:

$$\log \sum_{k=1}^{K} w_k c_k \geq \sum_{k=1}^{K} w_k \log c_k,$$

when $\sum_{k=1}^{K} w_k = 1.$

Hard to work with, since $\log$ is on outside of sum.

Jensen's inequality for convex functions



Source: R. Grosse

Learning a GMM

$$\log(p_\theta(\mathbf{x}_i)) = \log \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_i; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

$$= \log \sum_{k=1}^{K} r_{ik} \pi_k \frac{\mathcal{N}(\mathbf{x}_i; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{r_{ik}}$$

(w_k: r_{ik}, c_k: $\frac{\mathcal{N}(\mathbf{x}_i; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{r_{ik}}$)

$$\geq \sum_{k=1}^{K} r_{ik} \log \pi_k \frac{\mathcal{N}(\mathbf{x}_i; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{r_{ik}}$$

Jensen's inequality:

$$\log \sum_{k=1}^{K} w_k c_k \geq \sum_{k=1}^{K} w_k \log c_k,$$

when $\sum_{k=1}^{K} w_k = 1.$

Works for any non-negative r_{ik} vector that sums up to 1!

But it can be shown that equality holds when r_{ik} are the responsibilities $p(z_i = k \mid \mathbf{x}_i)$!

Learning a GMM

$$\left(\boldsymbol{\pi}, \{\boldsymbol{\mu}_i\}_{i=1}^K, \{\boldsymbol{\Sigma}_i\}_{i=1}^K\right)$$

$$\log(p_\theta(\mathbf{x}_i)) \geq L(\mathbf{r}, \boldsymbol{\theta}) = \sum_{k=1}^{K} r_{ik} \log \pi_k \frac{\mathcal{N}(\mathbf{x}_i; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{r_{ik}}$$

This is called an **evidence lower bound** (we'll discuss this more in a future class)

How do we maximize $L(\mathbf{r}, \boldsymbol{\theta})$ w.r.t. $\boldsymbol{\theta}$?

→ Basically just MLE estimation for Gaussian (but with some extra weights r_{ik}).

→ This gives us a new set of parameters θ'

How do we maximize $L(\mathbf{r}, \boldsymbol{\theta})$ w.r.t. $\mathbf{r}$?

→ Set $\mathbf{r}$ to be the responsibilities!

Learning a GMM

$$\log(p_\theta(\mathbf{x}_i)) \geq L(\mathbf{r}, \boldsymbol{\theta}) = \sum_{k=1}^{K} r_{ik} \log \pi_k \frac{\mathcal{N}(\mathbf{x}_i; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{r_{ik}} \quad = \sum_{k=1}^{K} r_{ik} \log \frac{p_\theta(\mathbf{z} = k, \mathbf{x}_i)}{r_{ik}}$$

How do we maximize $L(\mathbf{r}, \boldsymbol{\theta})$ w.r.t. $\mathbf{r}$?

If we set $r_{ik} = p_\theta(\mathbf{z} = k \mid \mathbf{x}_i)$, then

$$L(\mathbf{r}, \boldsymbol{\theta}) = \sum_{k=1}^{K} p_\theta(\mathbf{z} = k \mid \mathbf{x}_i) \log \frac{p_\theta(\mathbf{z} = k, \mathbf{x}_i)}{p_\theta(\mathbf{z} = k \mid \mathbf{x}_i)} \qquad \text{using } p_\theta(\mathbf{z} \mid \mathbf{x}) = \frac{p_\theta(\mathbf{z}, \mathbf{x})}{p_\theta(\mathbf{x})}$$

$$= \sum_{k=1}^{K} p_\theta(\mathbf{z} = k \mid \mathbf{x}_i) \log p_\theta(\mathbf{x}_i) \quad = \log p_\theta(\mathbf{x}_i)$$

This makes the bound tight. Setting to be the responsibilities maximizes L.

The EM algorithm for GMMs

Goal: solve for the model parameters $\boldsymbol{\theta}$

Loop:

- Compute the responsibilities $\mathbf{r}^{(t)} = p_{\theta^{(t)}}(\mathbf{z} = k \mid \mathbf{x}_i)$ [E step]
- Solve $\boldsymbol{\theta}^{(t+1)} = \text{argmax}_{\boldsymbol{\theta}} L(\mathbf{r}^{(t)}, \boldsymbol{\theta})$ [M step]

- Each step increases (or doesn't decrease) the bound L.
- This is an example of the **Expectation Maximization (EM)** algorithm.
- **E step**: compute expected latent assignments.
- **M step**: maximize log likelihood using those latent assignments.
- We won't discuss the fully general version in this class, but it is a very useful idea for fitting probabilistic models.

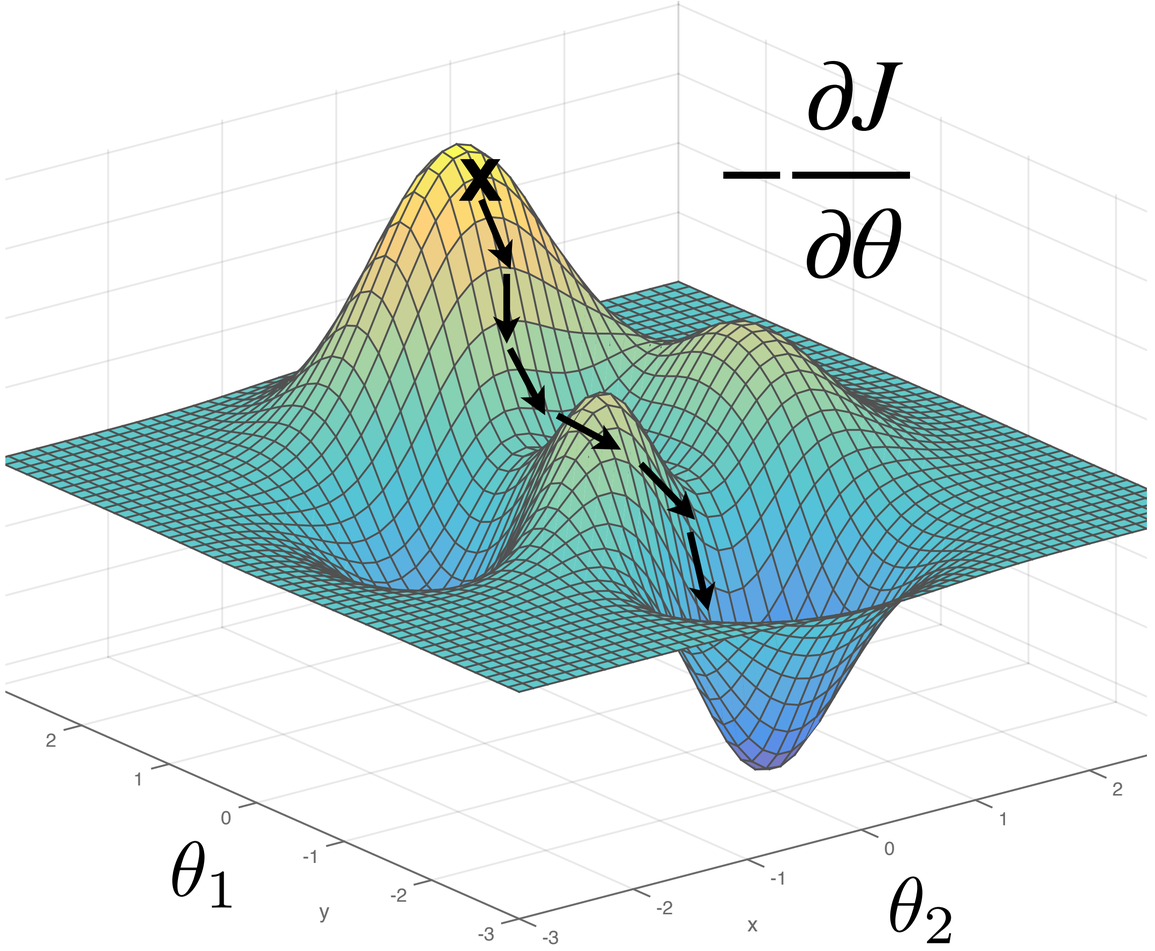
Learning a GMM by gradient descent

- One alternative approach (closer to what we'll do in the rest of the class) is to learn the GMM's parameters by **gradient descent**.
- Instead of maximizing $\sum_{i=1}^{N} \log p_\theta(\mathbf{x}_i)$, *minimize* $J(\theta) = \sum_{i=1}^{N} -\log p_\theta(\mathbf{x}_i)$.

Review: stochastic gradient descent

Recall: Learning parameters using gradient descent

$J(\theta)$



$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla J(\theta)$$

$$\theta^* = \arg\min_{\theta} J(\theta)$$

Source: Isola, Torralba, Freeman

Batch gradient descent

Loss function:

$$J(\theta) = \frac{1}{N} \sum_{i=1}^{N} L(x_i, \theta)$$

Its gradient is the sum of gradients for each example:

$$\nabla J(\theta) = \frac{1}{N} \sum_{i=1}^{N} \nabla L(x_i, \theta)$$

Problem: requires iterating over every training example each gradient step!

Can we speed this up?

Stochastic gradient descent

This is just an average!

$$\nabla J(\theta) = \frac{1}{N} \sum_{i=1}^{N} \nabla L(x_i, \theta)$$

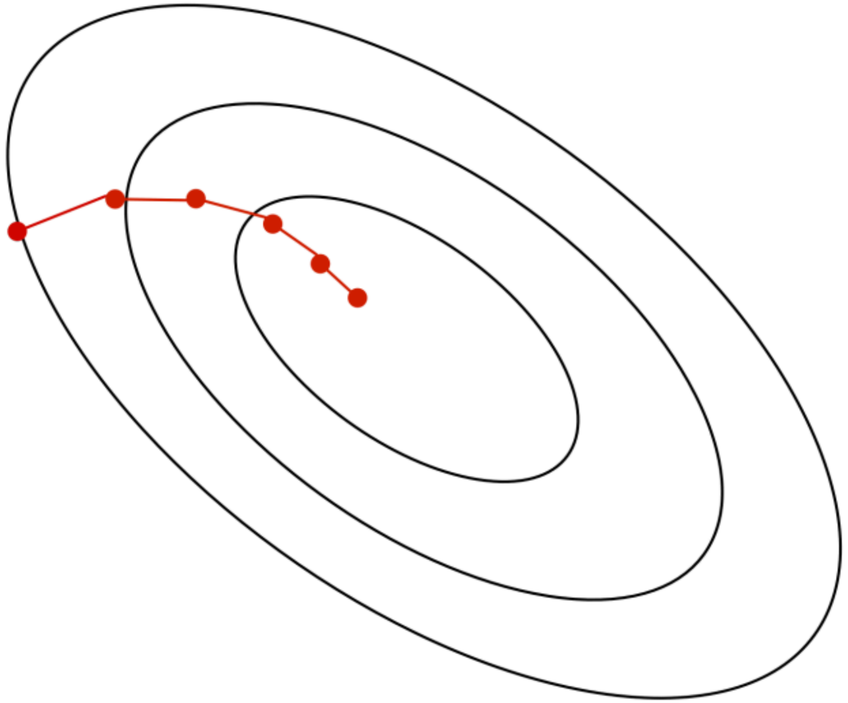
We know from statistics that we can estimate the average of a full "population" from a *sample*.

$$\nabla J(\theta) \approx \frac{1}{|B|} \sum_{i \in B} \nabla L(x_i, \theta)$$

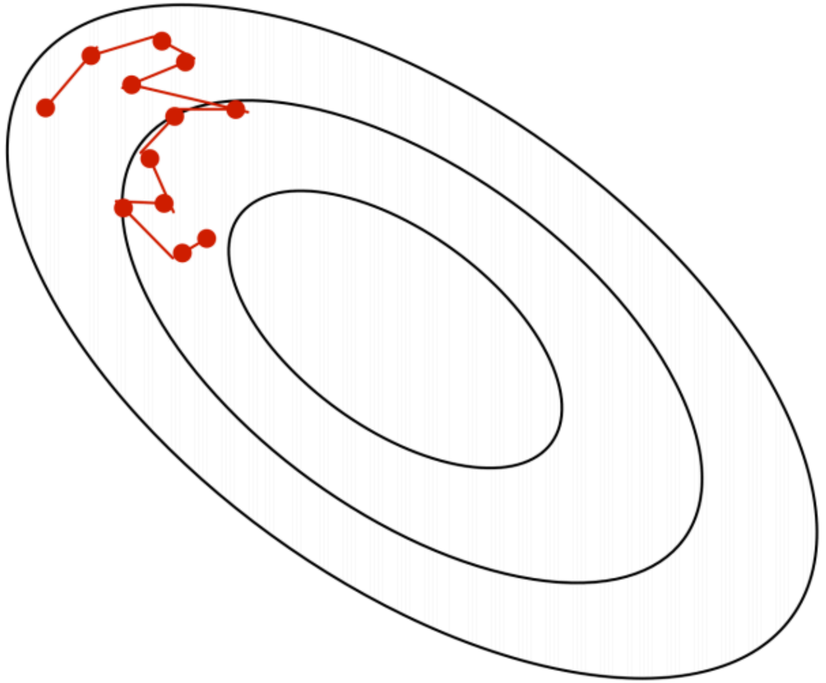
where B is a **minibatch:** a random subset of examples.

This is called **stochastic gradient descent (SGD)**.

Stochastic gradient descent



Batch gradient descent

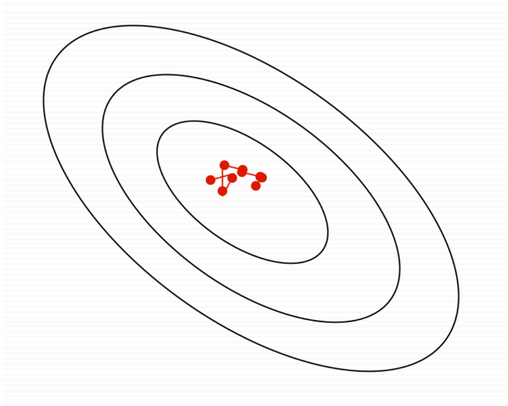


Stochastic gradient descent

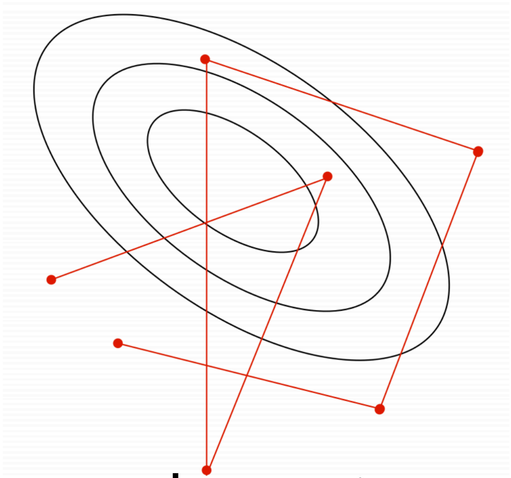
Source: R. Grosse

Learning rate

- Sensitive to the learning rate: $\theta^{t+1} = \theta^t - \eta_t \nabla_\theta J(\theta)\Big|_{\theta=\theta^t}$



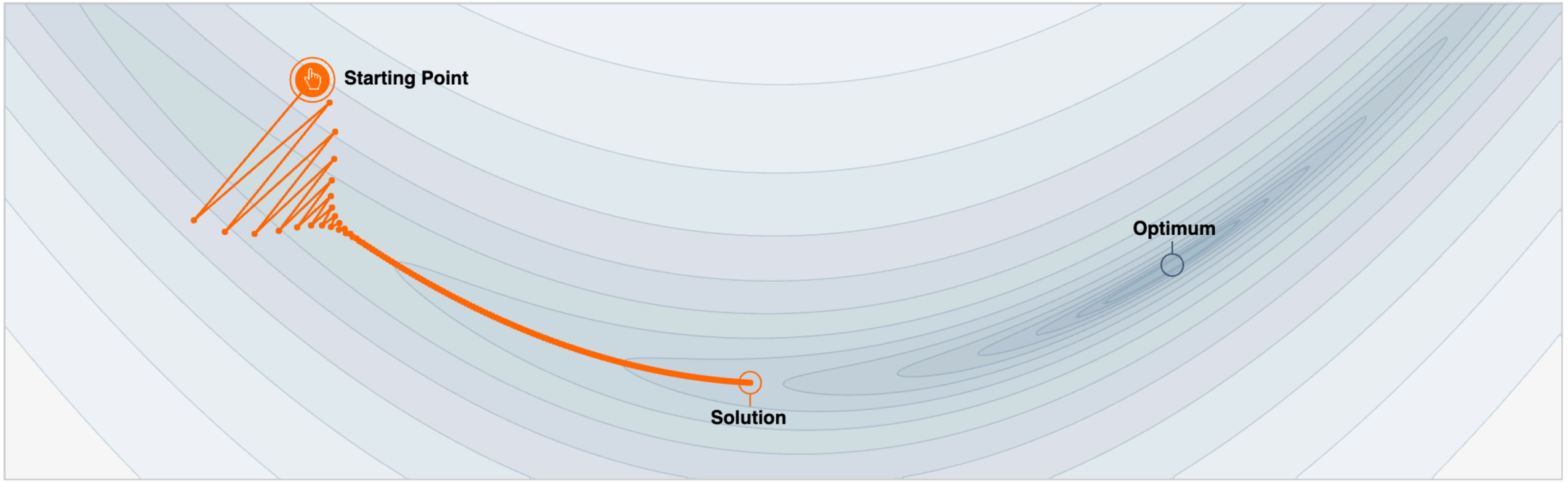
Small learning rate



Large learning rate

Figure source: R. Grosse

Dealing with oscillations



Source: https://distill.pub/2017/momentum

Momentum

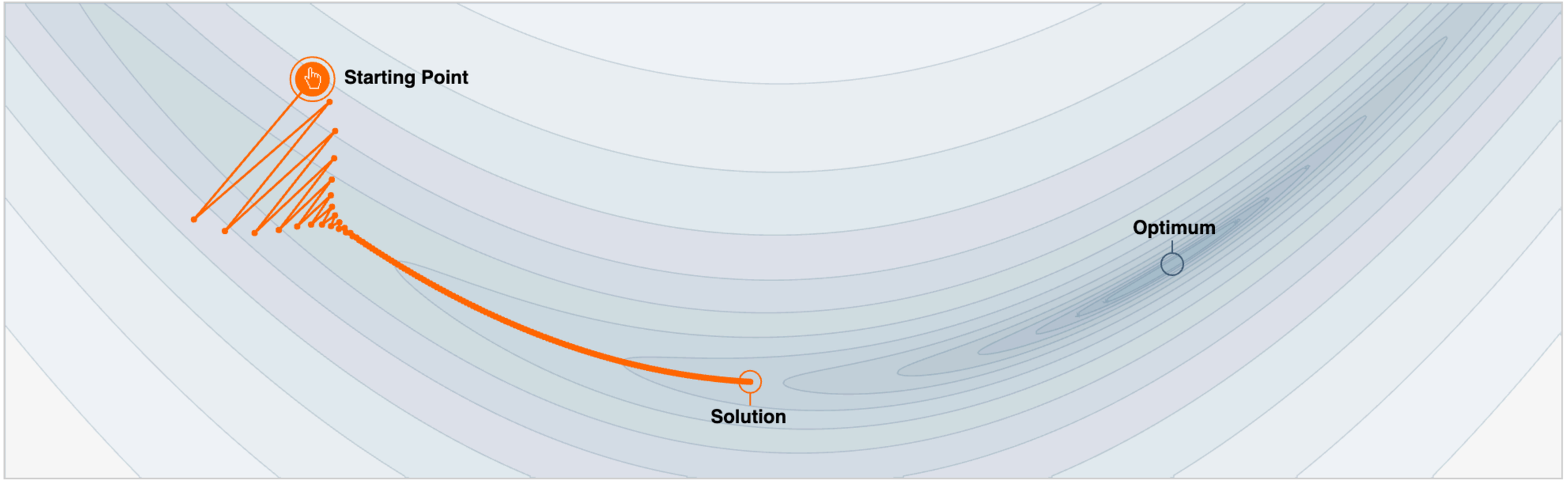
- Also known as the “heavy ball momentum”

$$\mathbf{v}^{(k+1)} \leftarrow \beta \mathbf{v}^{(k)} - \eta \frac{\partial J}{\partial \mathbf{W}}(\mathbf{W}^{(k)})$$

$$\mathbf{W}^{(k+1)} \leftarrow \mathbf{W}^{(k)} + \mathbf{v}^{(k+1)}$$

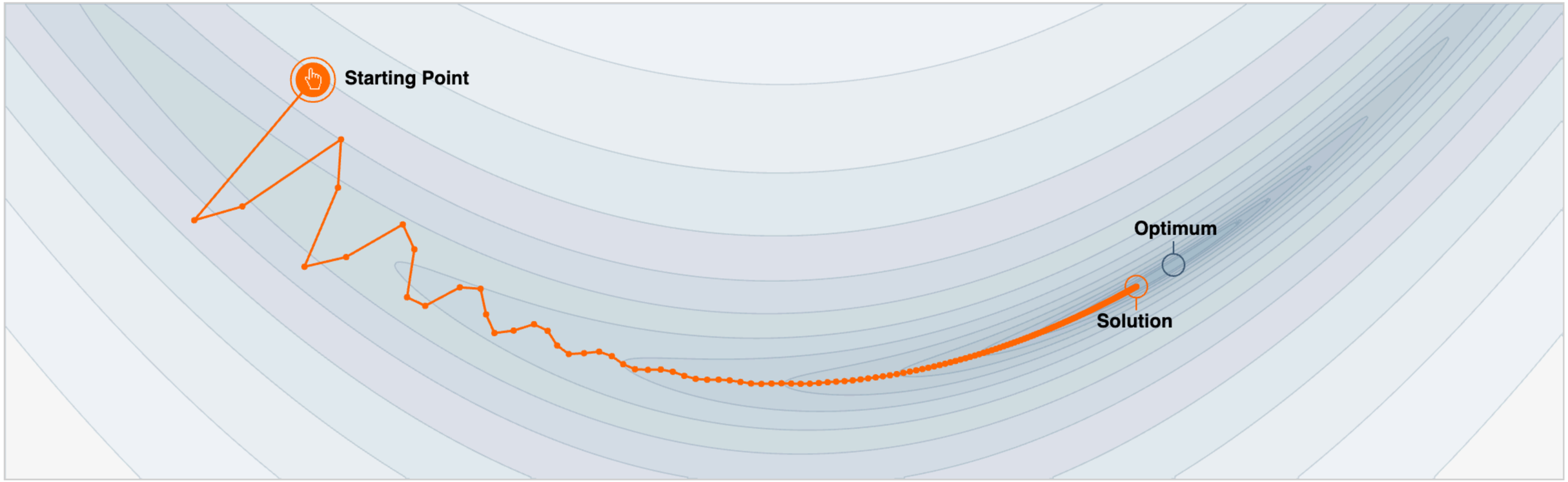
- Remember previous gradients and “build up speed”, setting $0 \leq \beta < 1$
- Reduces oscillations in high curvature regions, and picks up speed in when the loss surface is nearly flat.

Dealing with oscillations (no momentum)



Source: https://distill.pub/2017/momentum

Dealing with oscillations (with momentum)



$\beta = 0.99$

Source: https://distill.pub/2017/momentum

Learning multivariate Gaussian using SGD

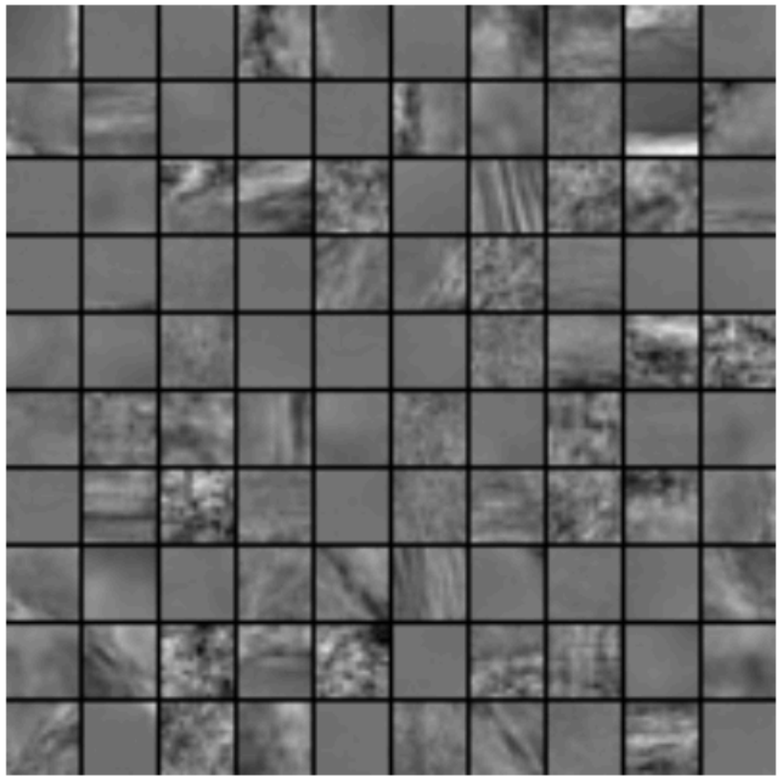
```
mu = randn(d)
A = randn((d, d)) # Matrix representing the covariance

for x in loader: # x: single example
  # Create covariance matrix
  cov = A.T @ A
  # Add regularization
  cov = cov + eps * eye(d)
  log_Z =  -0.5 * (d * log(2 * np.pi) + log(det(cov)))
  mahalanobis_distance = (x - mu).T @ inv(cov) @ (x - mu)

  # Compute the NLL
  loss = -(log_Z + -0.5 * mahalanobis_distance)

  # Perform an SGD update
  loss.backward()
```

Implementation notes for SGD with GMMs

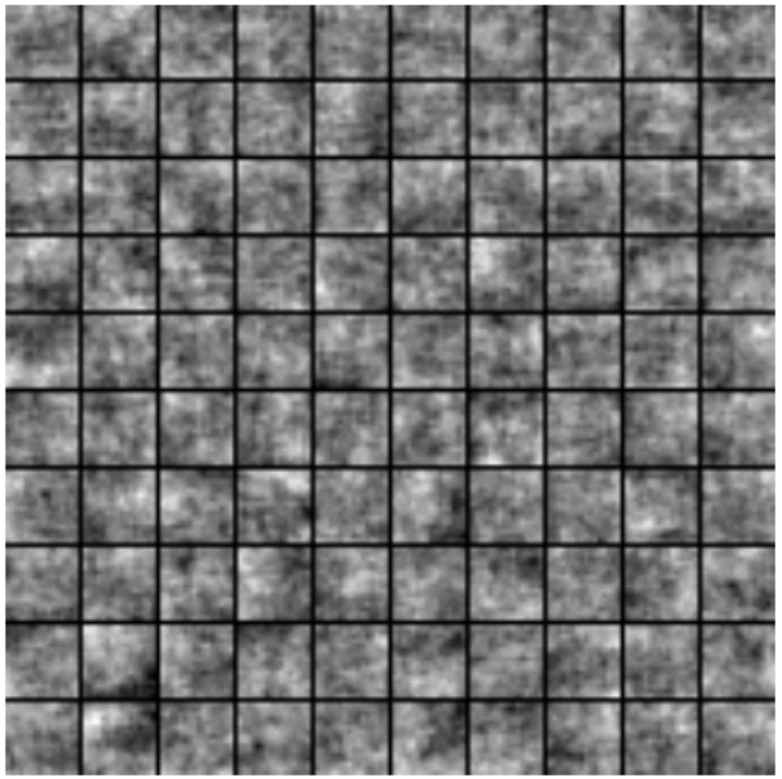
- Represent covariance matrix $\boldsymbol{\Sigma} = AA^\top$ to ensure it's positive semidefinite.
- Regularize the covariance matrix by adding a small value to its diagonal: $\boldsymbol{\Sigma}' = \boldsymbol{\Sigma} + \epsilon I$
  - This avoids numerical issues and degeneracies, e.g., $\det(\boldsymbol{\Sigma})$ approaches 0
- Represent the mixture weight *logits* $\mathbf{w}$. Take a softmax to ensure that they sum to one: $\pi_i = \dfrac{\exp(w_i)}{\sum_{j=1}^{K} \exp(w_j)}$

Case study: GMMs for image patches

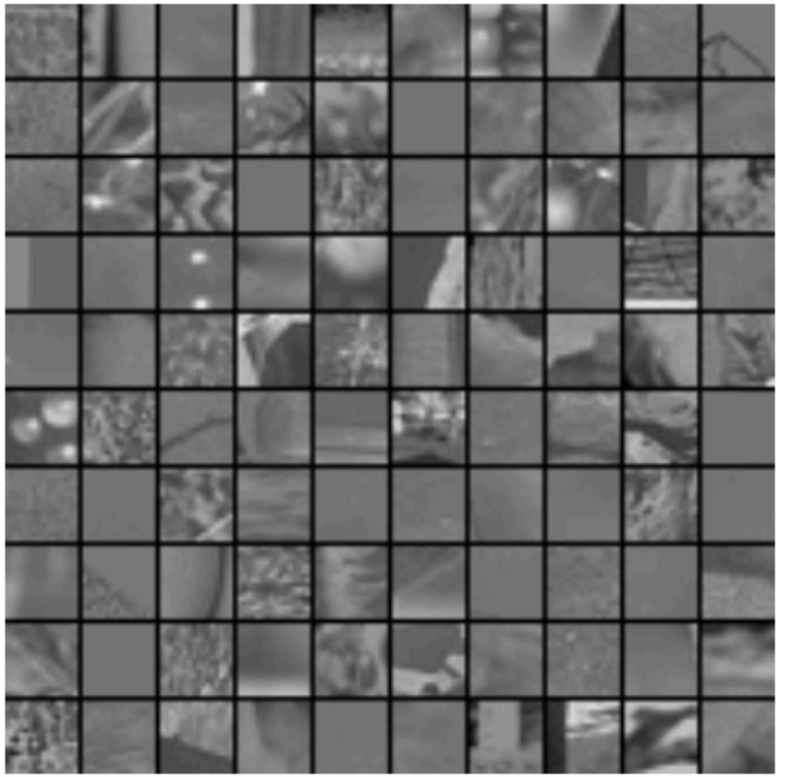
Based on paper by Zoran & Weiss, 2012.



A



B



C

Match the patches to the model:

1. Gaussian
2. Real patches
3. GMM

Source: [Zoran & Weiss, 2012]

What does this model learn?

Visualization tool: eigenvectors

For a zero-mean scalar Gaussian, you can sample points:

$$x = \sigma z, \text{ where } z \sim \mathcal{N}(0, 1)$$

Analogously, for a zero-mean multivariate Gaussian:

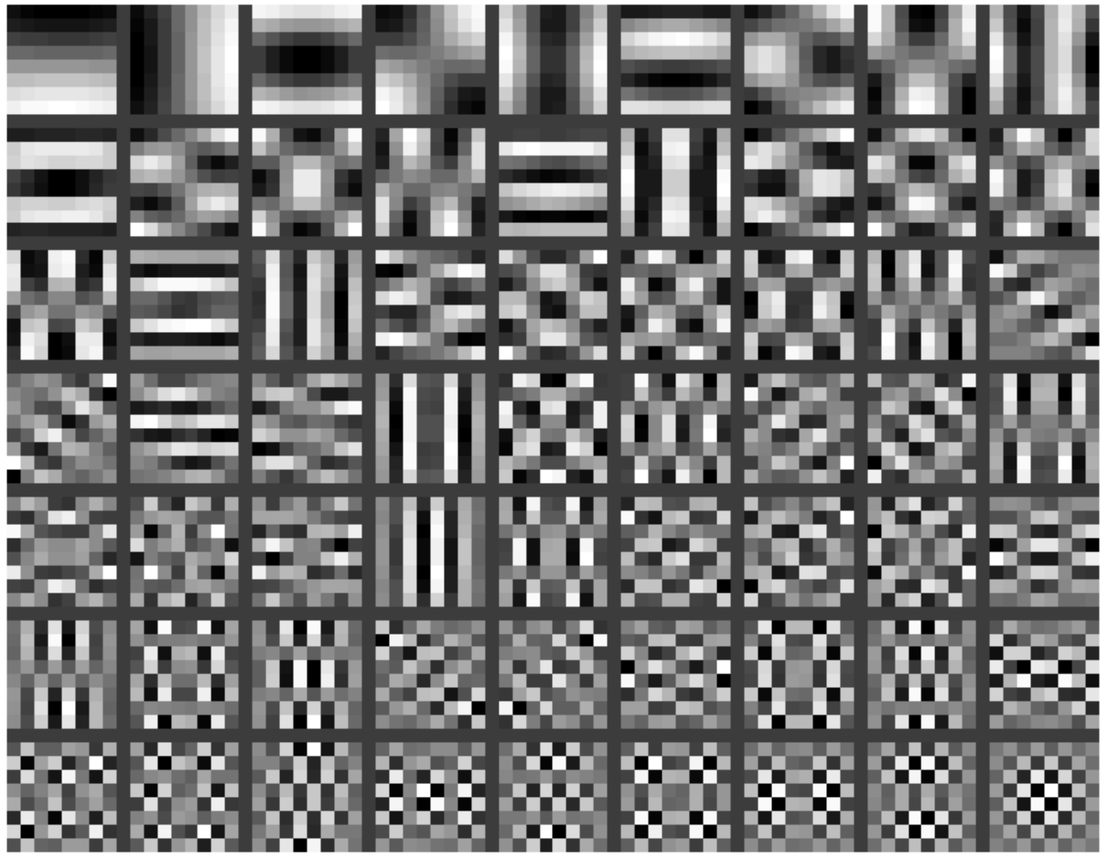
$$\mathbf{x} = \mathbf{V}\mathbf{D}^{\frac{1}{2}}\mathbf{z}, \text{ where } \mathbf{z} \sim \mathcal{N}(0, I)$$

and $\mathbf{\Sigma} = \mathbf{V}\mathbf{D}\mathbf{V}^\top$ is the eigendecomposition.

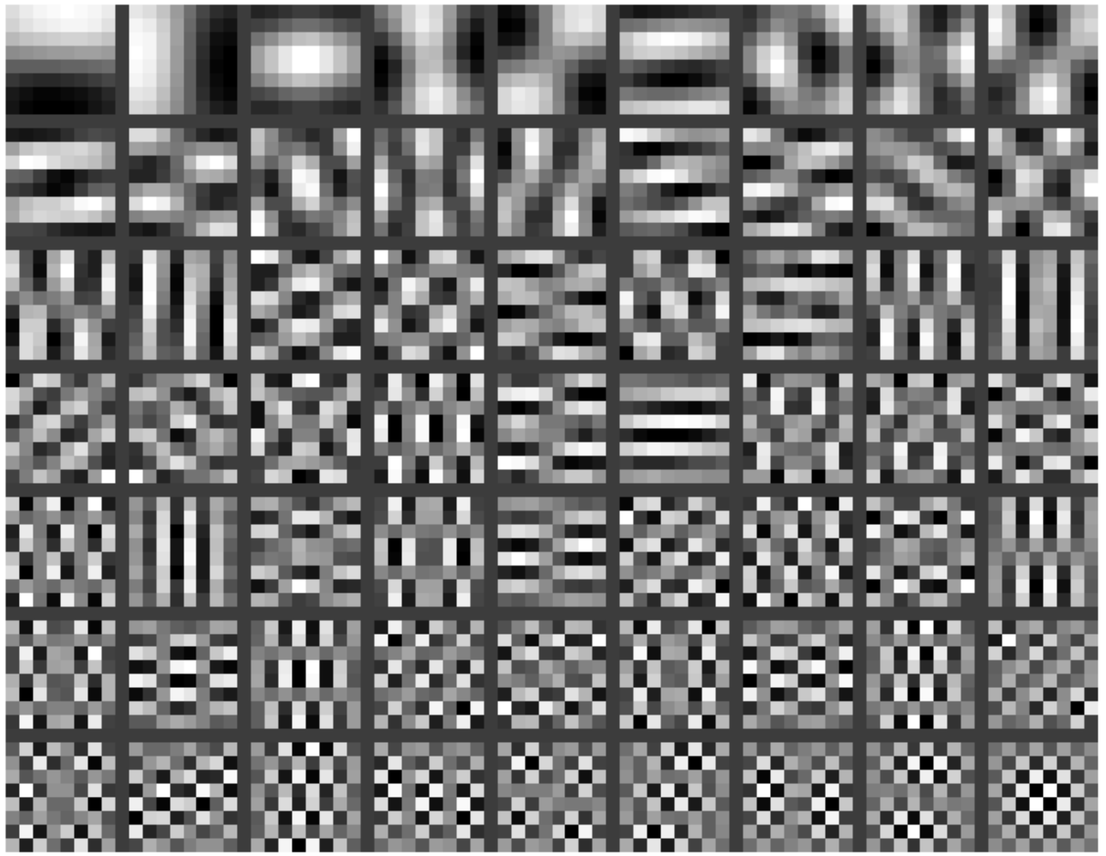
- Eigenvectors capture structure of image patches.
- The eigenvectors with high eigenvalues dominate the appearance of the image patches.

Eigenvectors for 2-component GMM

$\pi_1 = 0.5611$



$\pi_2 = 0.4389$

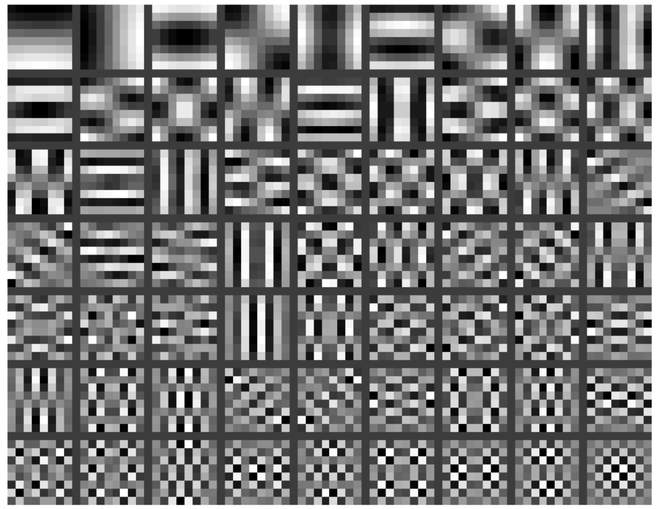


Source: [Zoran & Weiss, 2012]

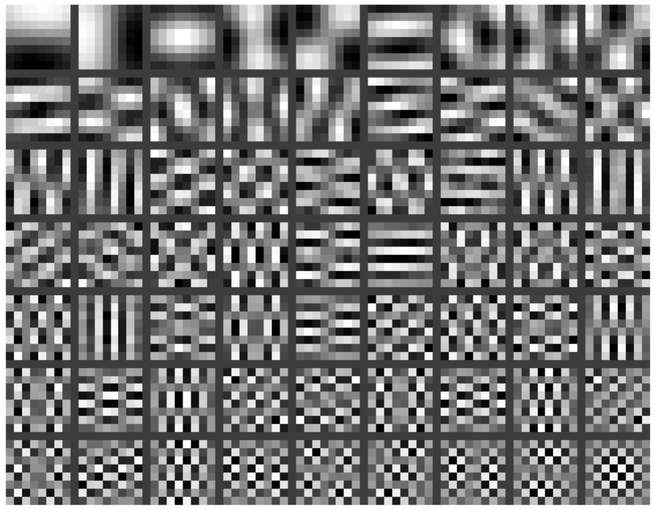
Eigenvectors for 2-component GMM

Eigenvectors for $\boldsymbol{\Sigma}_k$

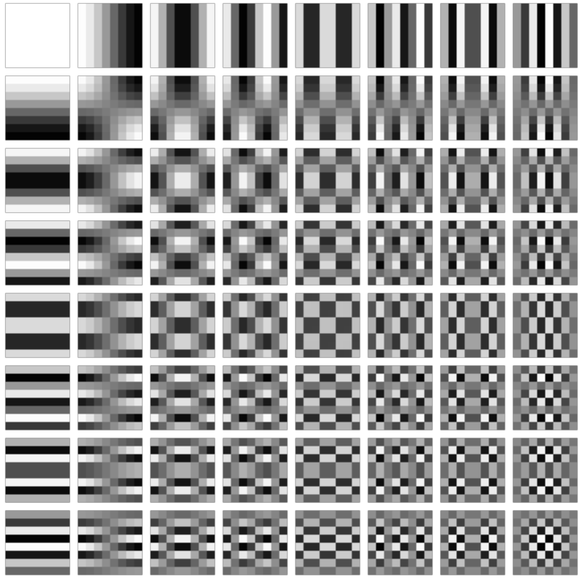
$\pi_1 = 0.5611$



$\pi_2 = 0.4389$

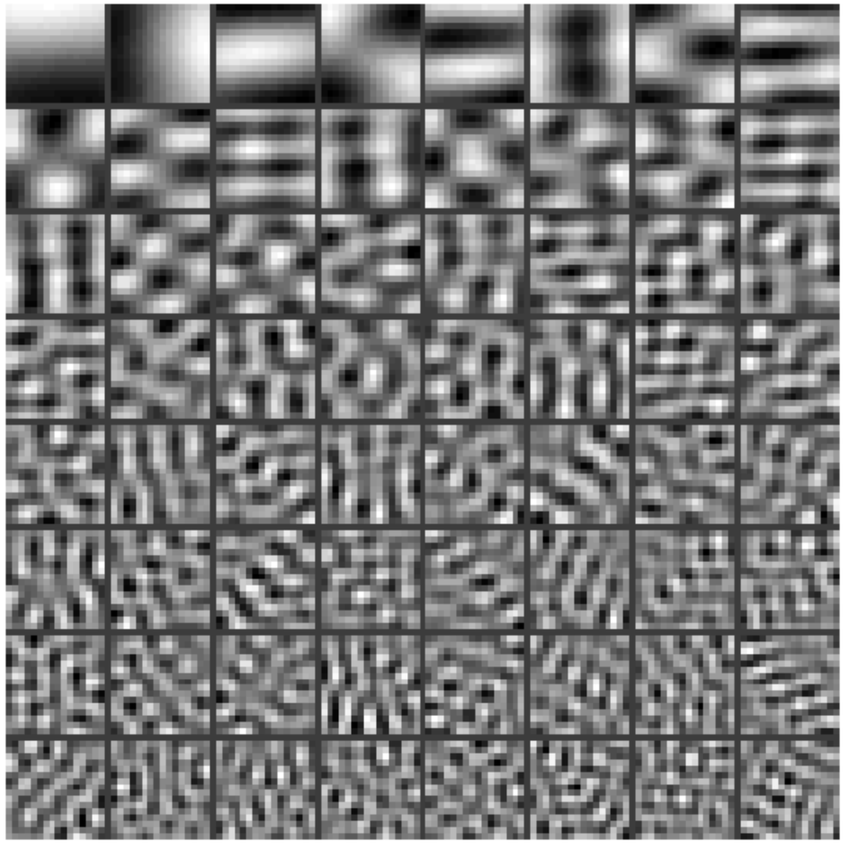


Discrete cosine transform
(similar to Fourier transform)

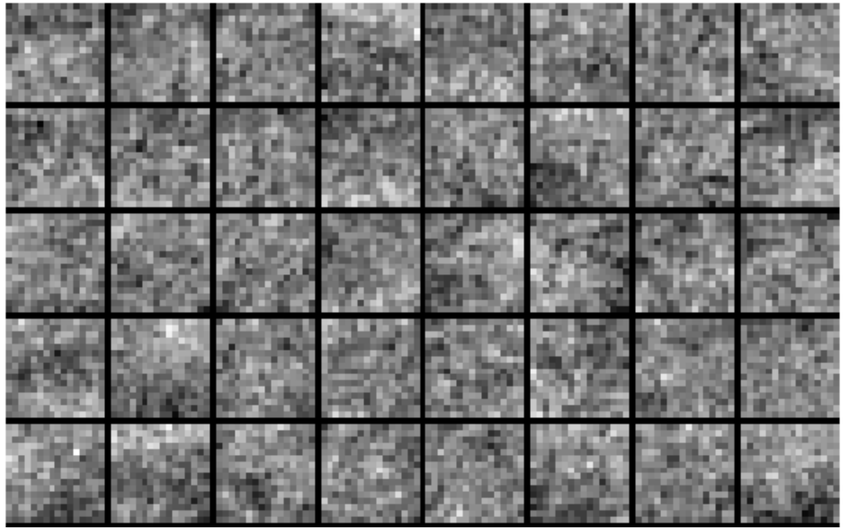


Used in JPEG compression

Source: Wikipedia

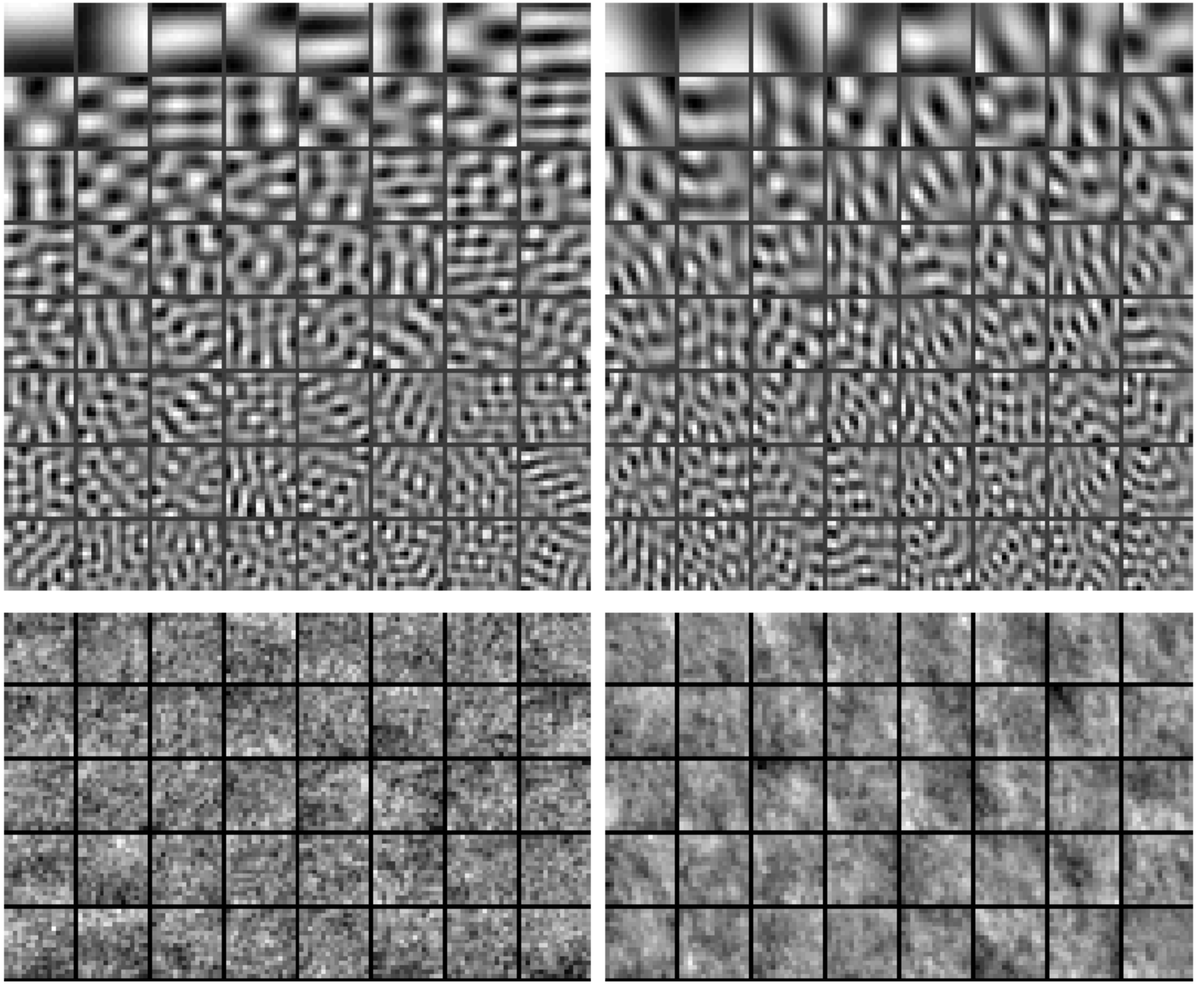


Eigenvectors for component k

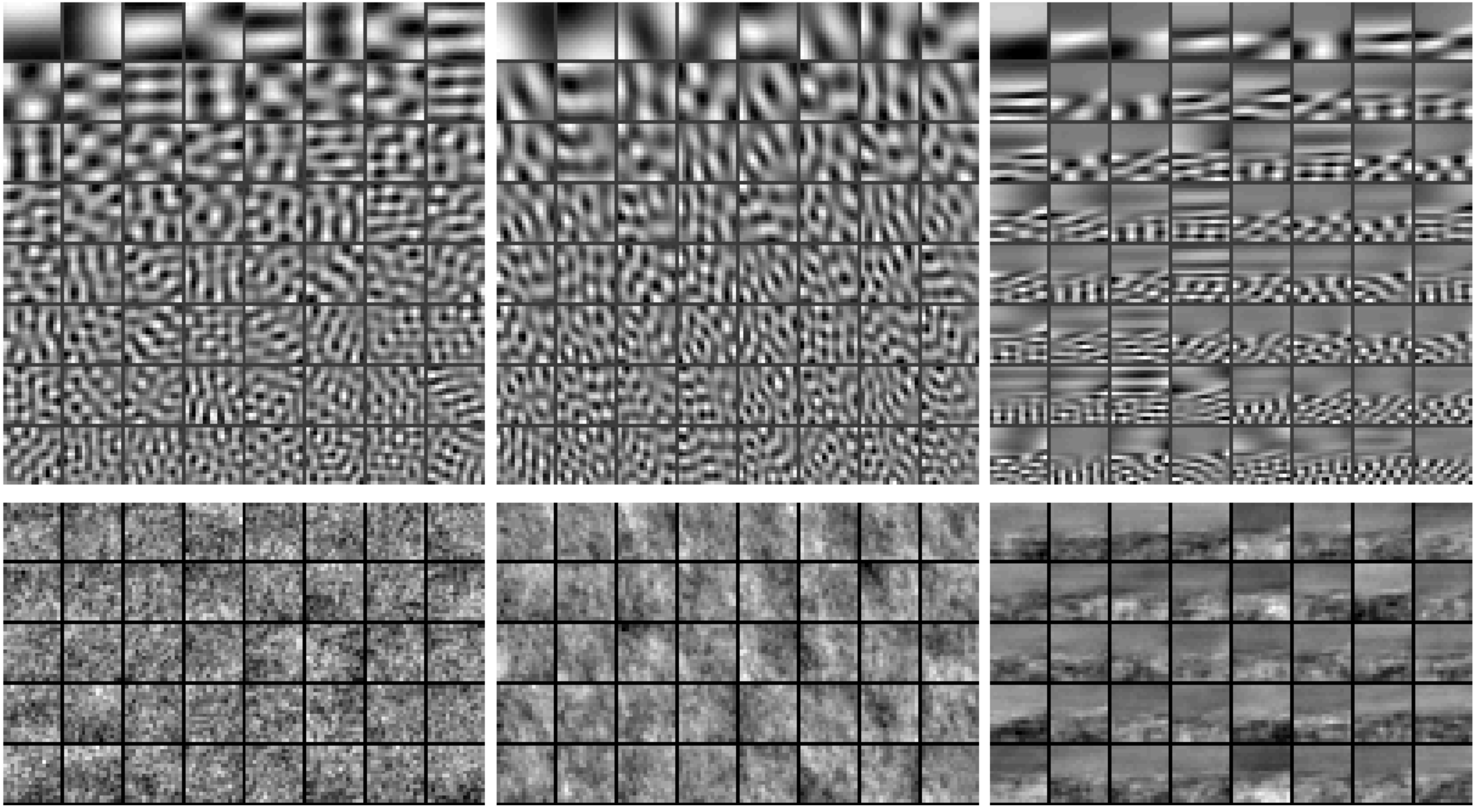


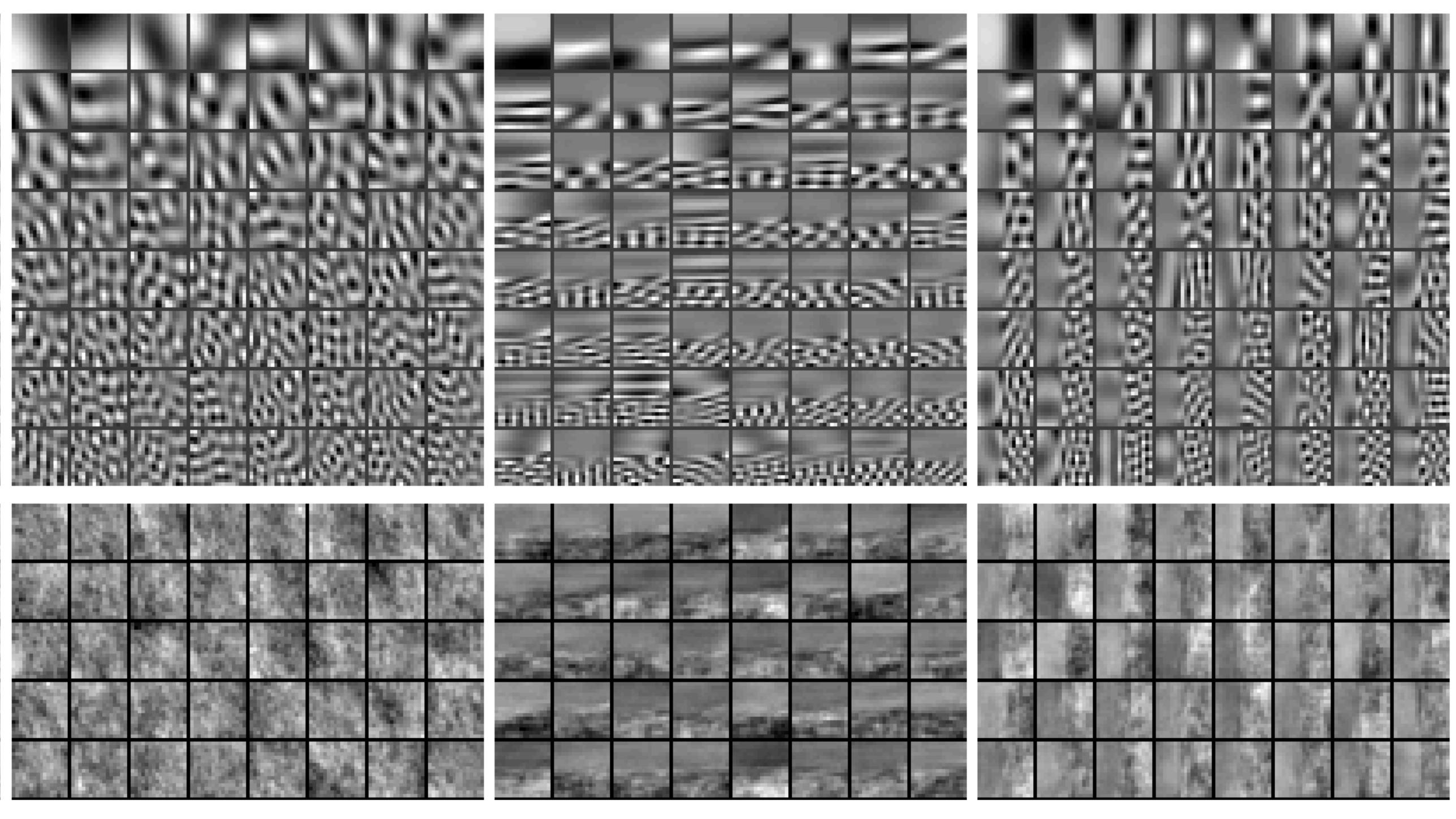
Samples from component k

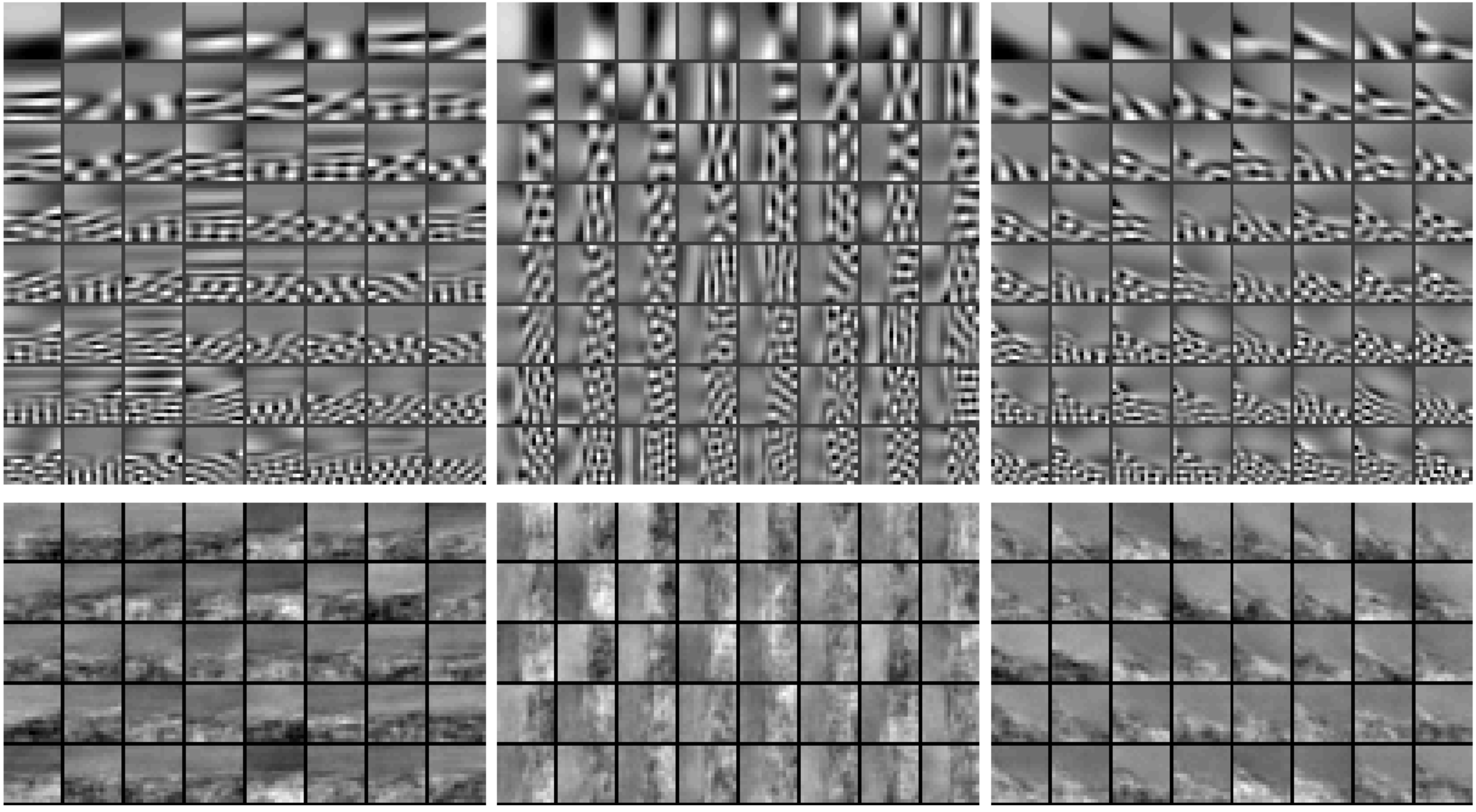
Source: [Zoran & Weiss, 2012]



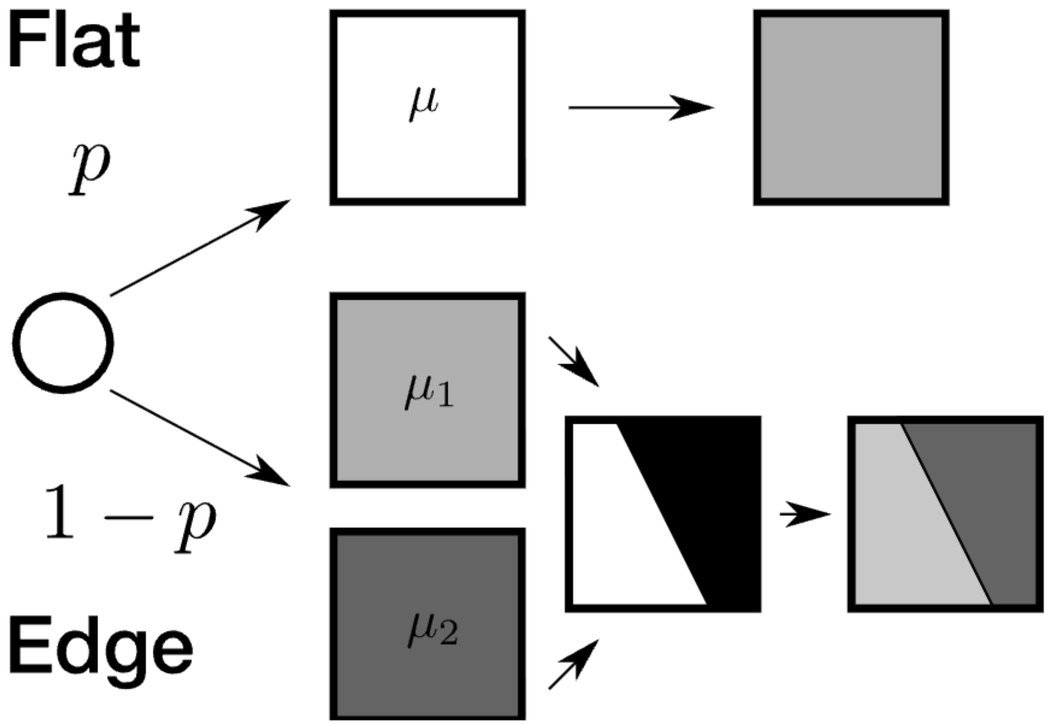
Source: [Zoran & Weiss, 2012]





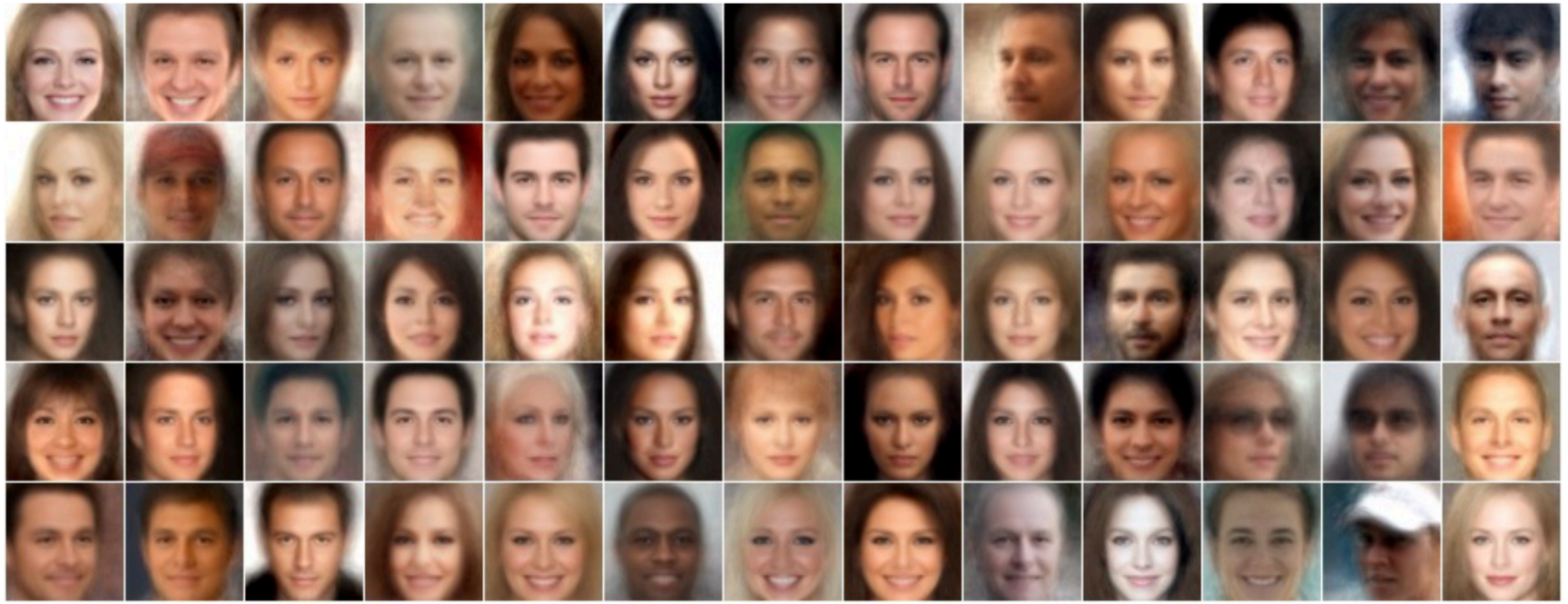


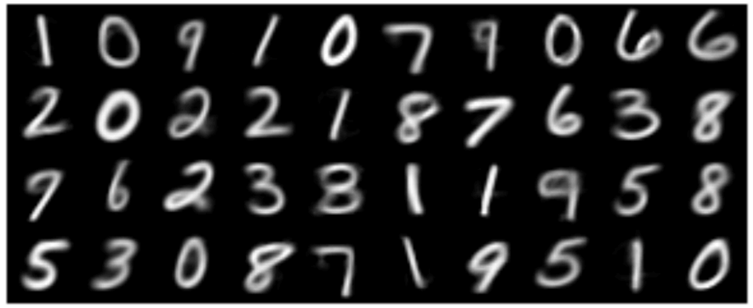
Dead leaf image generation model



Source: [Zoran & Weiss, 2012]

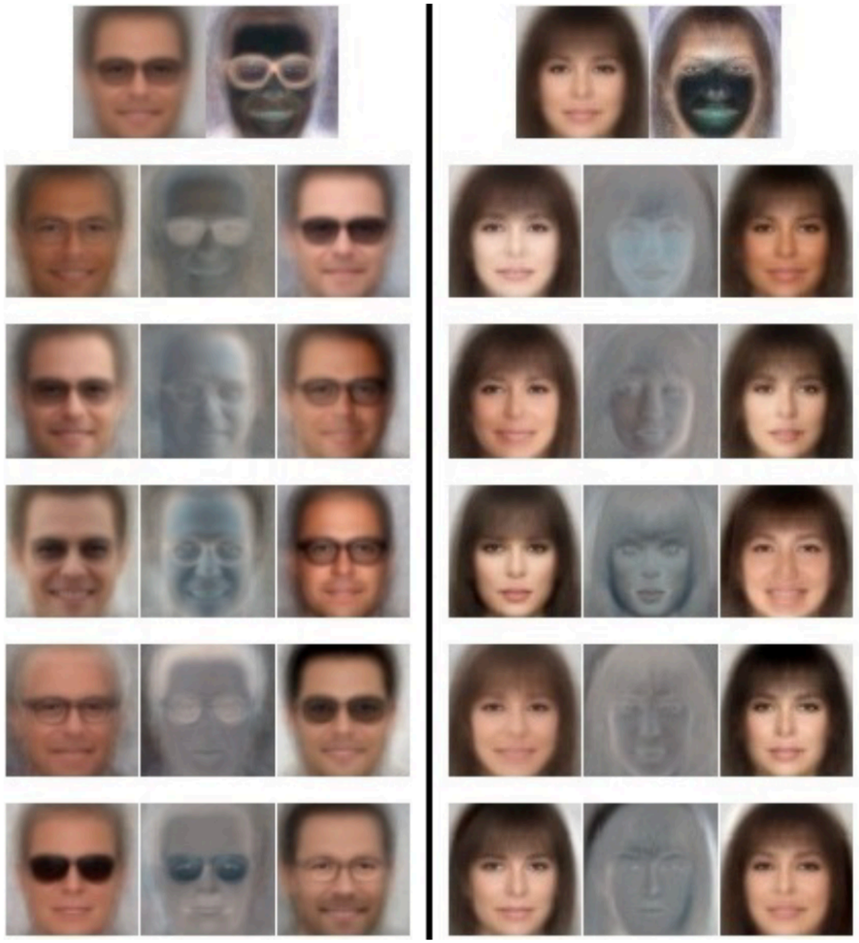
Scaling up GMMs to full images





Source: [Richardson & Weiss, 2018]

Scaling up GMMs to full images



Learned mixture components

Source: [Richardson & Weiss, 2018]

Image denoising

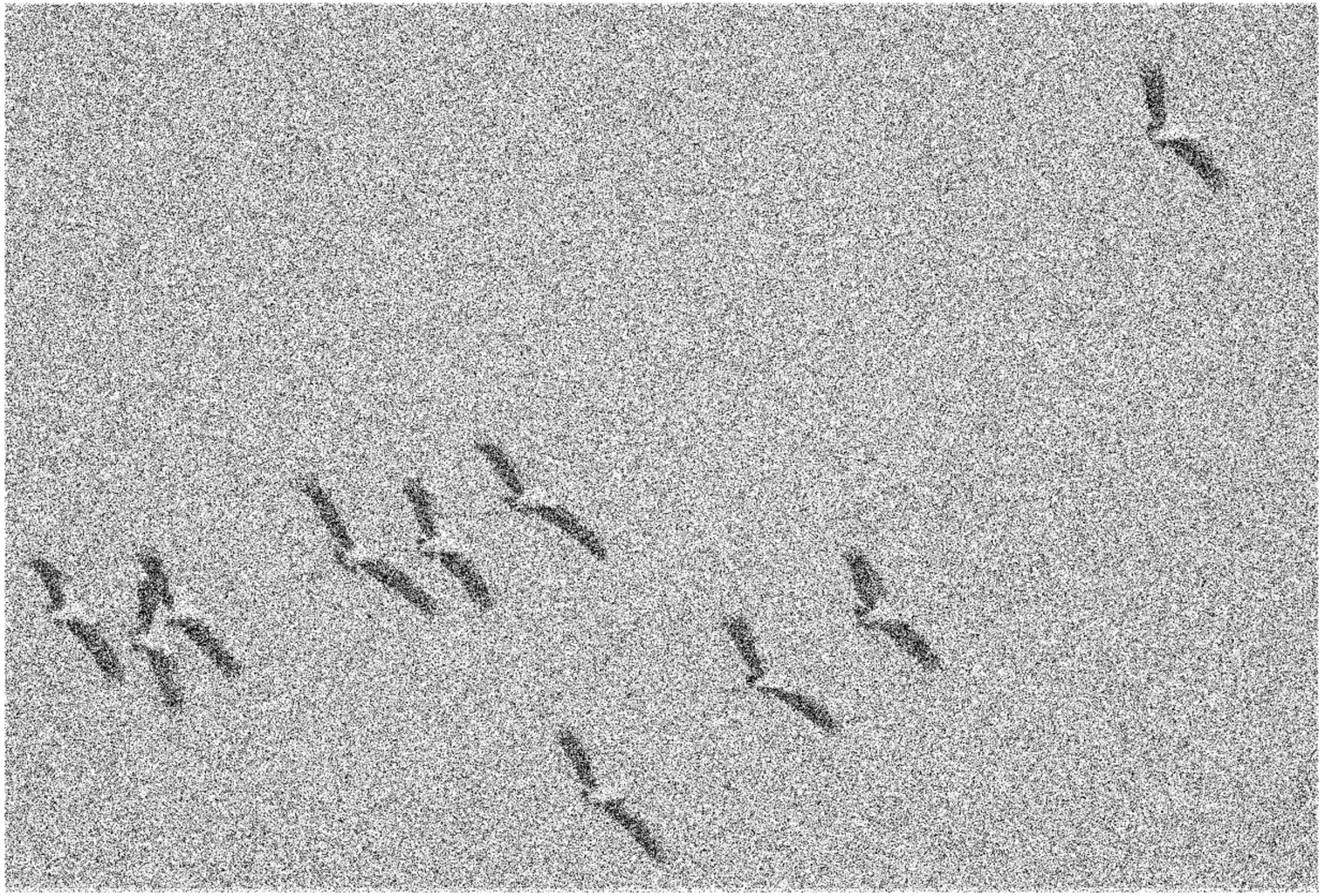
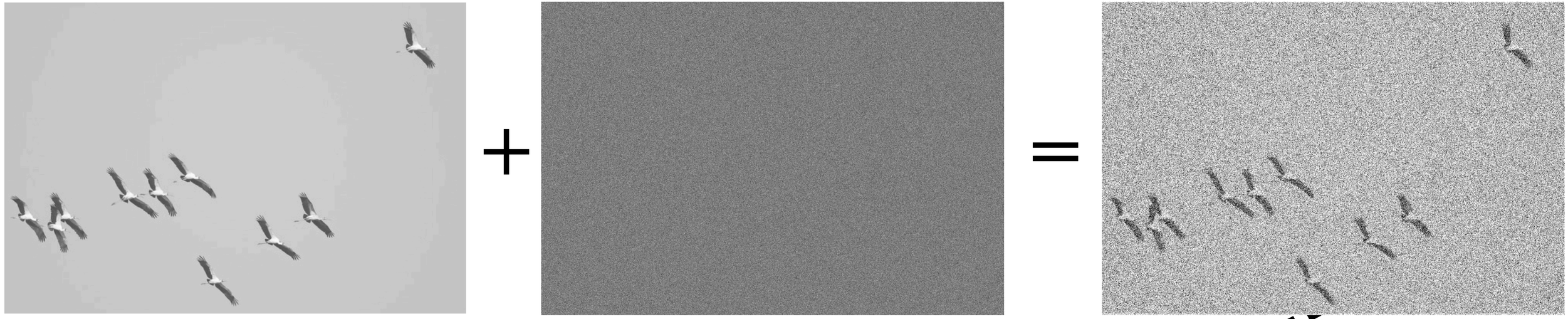


Photo by Fredo Durand

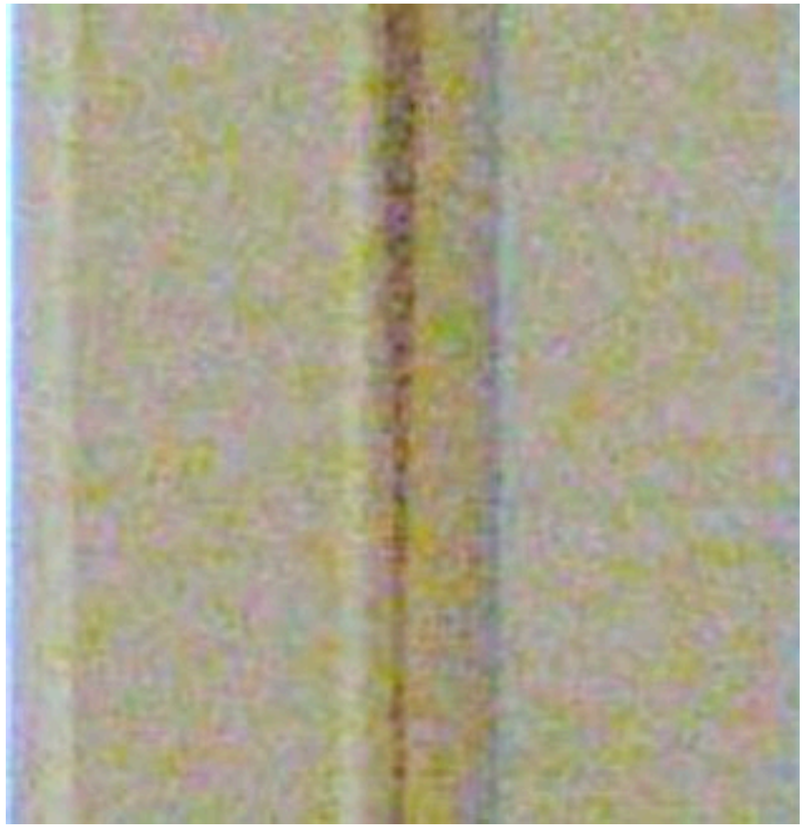
Application: image denoising

image + noise = noisy image



Goal: recover the original image

Image denoising problem



In practice: low light photography, “dead” pixels, etc.

Image source: wikipedia

Denoising using a prior

$$p(\mathbf{X}_{clean} \mid \mathbf{X}_{noisy}) = \frac{p(\mathbf{X}_{noisy} \mid \mathbf{X}_{clean})p(\mathbf{X}_{clean})}{p(\mathbf{X}_{noisy})}$$ by Bayes rule

We're solving for $\mathbf{X}_{clean}$, which is equivalent to *minimizing*:

$$L(\mathbf{X}_{clean}) = -\log p(\mathbf{X}_{noisy} \mid \mathbf{X}_{clean}) - \log p(\mathbf{X}_{clean})$$

What are good choices for these two terms?

Denoising using a GMM image prior

$$L(\hat{\mathbf{X}}) \approx \|\hat{\mathbf{X}} - \mathbf{X}_{noisy}\|^2 - \lambda \, \mathrm{EPLL}(\hat{\mathbf{X}}),$$

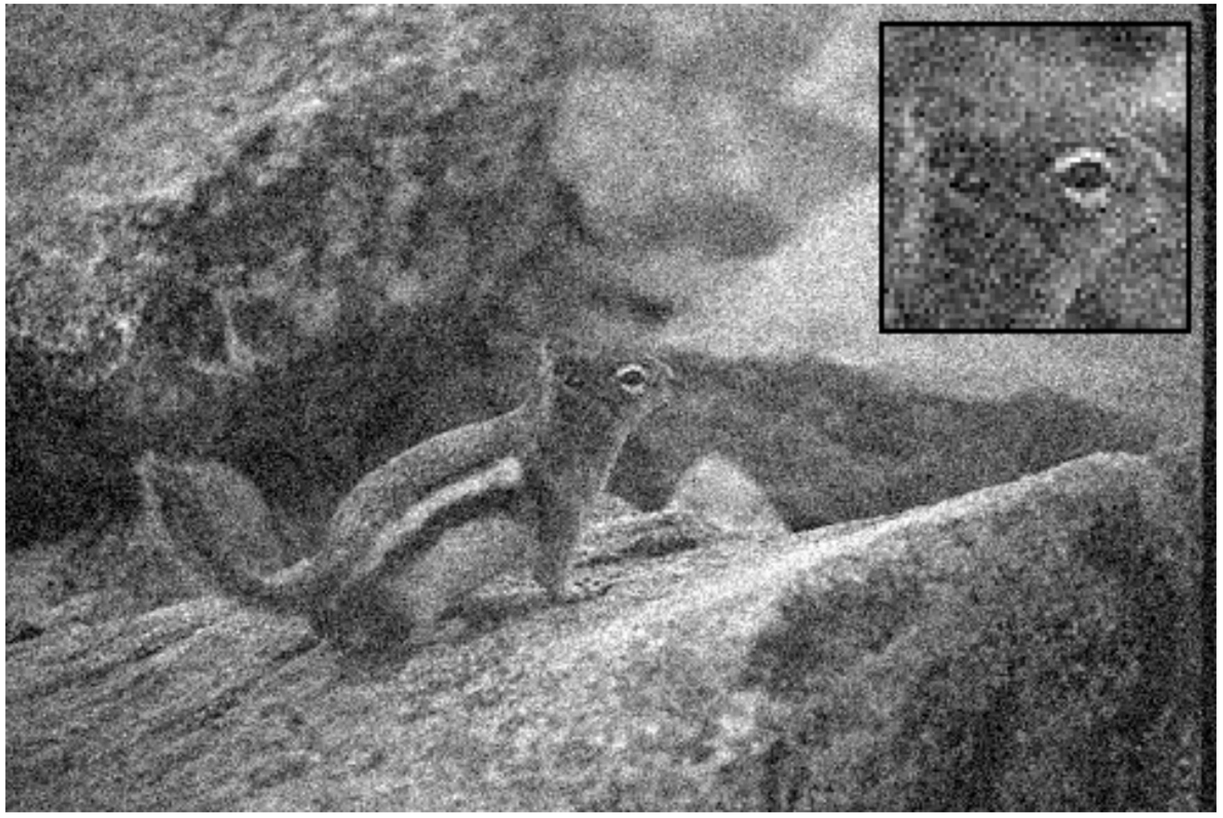
- First term (the likelihood) is equivalent to Gaussian centered on $\mathbf{X}_{noisy}$.
- Since we don't have a full image prior, let's use the "expected log patch likelihood": the average log likelihood our GMM assigns to each patch in the image:

$$\mathrm{EPLL}(\mathbf{X}) = \frac{1}{|\,P(\mathbf{X})\,|} \sum_{\mathbf{x} \in P(\mathbf{X})} \log p_\theta(\mathbf{x})$$

How do we solve for $\hat{\mathbf{X}}$? Gradient descent!

Source: [Zoran & Weiss, 2012]

Image denoising with a GMM



Input image

Denoised image

Next class: recorded neural net review