Classic \(K\)-means clustering

- Define \(s_n\) as the one-hot encoding of the discrete variable representing the index of the nearest cluster center for \(x_n\).
- It is also useful to think of a matrix \(S\) with entries \(s_{nk}\), holding the one-hot vectors in its rows.
- Assume we knew the cluster assignments \(s_n\) for each point \(x_n\).
- The \(K\)-means objective function measures the average distance between points \(x\) and their representatives:

\[
J = \sum_{n=1}^{N} \sum_{k=1}^{K} s_{nk} \| x_n - \mu_k \|^2
\]

- Learning amounts to finding both the prototypes \(\mu_k\) and the assignments \(s_n\) for each point, so as to minimize \(J\).
- This seems like a tricky optimization problem, because the \(s_n\) are discrete and the \(\mu_k\) are continuous.
- But learning gets easy if we decouple learning the \(s_n\) from learning the \(\mu_k\).
- This gives rise to a block coordinate-descent method, which is a special case of the EM-algorithm for training mixtures of Gaussians.

- \(K\)-means is traditionally a clustering algorithm.
- **Learning**: Fit \(K\) prototypes \(\mu_k\) (the rows of some matrix, \(W\)) to training data-points \(x_n\).
- **Inference**: Given a point, find the nearest prototype.
**Classic K-means clustering**

\[
J = \sum_{n=1}^{N} \sum_{k=1}^{K} s_{nk} \| \mathbf{x}_n - \mu_k \|^2
\]

Finding the optimal \( s_n \)
- Given the \( \mu_k \), we can optimize all the \( s_n \) independently, because the objective is just the sum over \( n \).
- But the squared error will be smallest if we set \( s_{nk} = 1 \) for whichever \( \mu_k \) is closest.
- Formally, to optimize all \( s_n \), given the set of \( \mu_k \), set:
  \[
  s_{nk} = \begin{cases} 
    1 & \text{if } k = \arg \min_j \| \mathbf{x}_n - \mu_j \|^2 \\
    0 & \text{otherwise.}
  \end{cases}
  \]

**Finding the optimal \( \mu_k \)**
- Given \( S \), \( J \) is a quadratic function of \( \mu_k \) which we can minimize by setting the derivative to zero:
  \[
  2 \sum_{n=1}^{N} s_{nk} (\mathbf{x}_n - \mu_k) = 0
  \]
- Solving for \( \mu_k \) yields:
  \[
  \mu_k = \frac{\sum_{n} s_{nk} \mathbf{x}_n}{\sum_{n} s_{nk}}
  \]
- This solution has a simple interpretation: Set \( \mu_k \) to the mean of all points currently assigned to cluster \( k \).

**Learning amounts to iterating inference for the \( s_n \), and adapting the parameters \( \mu_k \) until there are no more changes.**
- This training procedure always converges: \( J \) is positive, and every step either decreases it or leaves it unchanged.
- But there can be local minima.
- One way to deal with this is to try multiple runs with different initializations for the parameters \( \mu_k \) and to pick the solution with the lowest final cost.

*Classic K-means example (\( K = 2 \))*

This and most of the following images from: (Bishop, 2006)
The value of $J$ as learning progresses

The probably most important property of K-means is that it distributes cluster-centers in space, such that the cluster center density is roughly proportional to the data density. So it will resolve high-density regions well, at the cost of low-density regions.

K-means inference

- Given the trained model, we can infer the cluster-center for a new test-data point $x$ not seen during training, by finding the nearest $\mu_k$ like during training:

$$s_k(x) = \begin{cases} 1 & \text{if } k = \arg \min_j \| x_n - \mu_j \|^2 \\ 0 & \text{otherwise.} \end{cases}$$

- The set of all $K$ prototypes $\mu_k$ is called codebook.
- Clustering and K-means are also known as vector quantization.

K-means via online learning

- The reconstruction error for training point $x$ may be written

$$E(W) = \frac{1}{2} (x - w_{s(x)})^2$$

- Its gradient is

$$\frac{\partial E(W)}{\partial w_i} = -(x - w_{s(x)}) \delta_{s(x),i}$$

- So we can use the online learning rule:

$$w_{s(x)} \leftarrow w_{s(x)} + \eta (x - w_{s(x)})$$

- (Here, it is easier to think of $s(x)$ as index rather than one-hot vector.)
Geometry of online $K$-means

$\Delta w_s(x) = \eta(x - w_s(x))$

moves the winning weight vector towards the observation.

Online $K$-means and Hebbian learning

- We can interpret the online $k$-means updates as:
  - **Hebb-rule** + competition + unlearning
- To this end write the update as
  $\Delta w_k = \eta \delta_{ks(x)}(x - w_k)$

where

$$\delta_{ks(x)} = \begin{cases} 1 & \text{if } s(x) = k \\ 0 & \text{else} \end{cases}$$

is the “post-synaptic activity” determined by competition (“winner takes all” rule).
- There are two learning terms:

Hebbian $K$-means in 9 lines of code

```python
import numpy

def kmeans(W, X, numepochs, learningrate=0.01, batchsize=100):
    X2 = (X**2).sum(1)[:, None]
    for epoch in range(numepochs):
        for i in range(0, X.shape[0], batchsize):
            D = -2*numpy.dot(W, X[i:i+batchsize, :].T) + (W**2).sum(1)[:, None] + X2[i:i+batchsize].T
            S = (D==D.min(0)[None, :]).astype("float").T
            W += learningrate * (numpy.dot(S.T, X[i:i+batchsize, :]) - S.sum(0)[:, None] * W)
    return W
```

$K$-means and Hebbian learning

1. A Hebbian term:
   $$\delta_{ks(x)}x$$

2. An “unlearning” term:
   $$-\delta_{ks(x)}w_k$$

- The positive term decreases the energy near the data.
- The unlearning term increases the energy everywhere.
- “Hebb-rule + competition + unlearning” are present (not surprisingly) in a wide variety of learning algorithms, including contrastive divergence learning for RBMs.
K-means features learned from natural image patches

Self-organizing maps

- We obtain the self-organizing map (SOM) aka Kohonen network by changing the k-means update from
  \[ \Delta w_k = \eta \delta_{ks} (x - w_k) \]
  into
  \[ \Delta w_k = \eta h_{ks} (x - w_k) \]
  where \( h_{kj} \) is some smooth neighborhood function, that will let hiddens near the winning hidden learn, too.
- This requires hiddens to be arranged in space in some way (commonly 2-D).

The K-means energy function

- We can think of K-means as an instance of energy based learning, by defining the energy function
  \[ E(x) = \| x - W \text{wta} (W^T x) \|^2 = \| x - w_{s(x)} \|^2 \]
- Since far from the cluster-centers the energy goes to \( \infty \), K-means has then low energy everywhere.
- In other words, K-means simply doesn’t have the capacity to produce an arbitrary energy surface with low energy far away from data.
The K-means energy function

- If we define the unnormalized probability for a point as
  \[ q(x_n) = \exp(-E(x_n)) \]
  we obtain a density model which is just the superposition of \( K \) bumps.
- So the probability goes to zero far from any cluster center.
- Unlike RBMs and many other probabilistic models, there is no need to lower density away from the data in this model. Instead, the density will be low “by design”.
- (LeCun, 2006)

Winner-takes-all and lateral interactions

- The winner-takes-all function may be defined as
  \[ \text{wta}(x) = \text{onehot}( \arg \min_k \| x - w_k \|^2 ) \]
  where \( w_k \) is a column of \( W \).
- The squared distance can be written as
  \[ \| x - w_k \|^2 = x^T x + w_k^T w_k - 2w_k^T x \]
  If all \( w_k \) have the same norm, inference amounts to finding the hidden which maximizes
  \[ w_k^T x \]
  which is the usual “simple cell” response.

Lateral interactions

- Since \( \text{wta} \) is a function of all the hiddens, inference requires the hiddens to talk to each other.
- This is commonly referred to as lateral interactions.
- Typically, the interactions take the form of lateral inhibition.

End-stopping

- For the simple cell in the top middle, a linear model would predict the bottom right stimulus to give at least as large a response as the bottom left stimulus.
- But for actual neural responses it may give a weaker response.
- This effect is known as end-stopping, and it may be counted as evidence for lateral inhibition.

from: Natural Image Statistics (Hyvarinen, Hurri, Hoyer; 2009)
Gaussian mixture models

If we add prior probabilities $\pi_k$ for each cluster and interpret each cluster as a Gaussian with its own variance, we get the Mixture of Gaussians:

$$p(x) = \sum_k \pi_k p(x|k) = \sum_k \pi_k N(x|\mu_k, \Sigma_k)$$

For $p(x)$ to be a probability distribution, we need $\sum_k \pi_k = 1$ and $\pi_k > 0$ $\forall k$

Thus, we may interpret the $\pi_k$ as probabilities themselves.

This motivates introducing latent variables $s$ and re-writing the model, equivalently, in terms of two distributions $p(s)$ and $p(s|x)$:

$$p(x) = \sum_s p(s)p(x|s)$$

where

$$p(s) = \prod_{k=1}^K \pi_{s_k}$$

is a discrete distribution ($s$ is in one-hot encoding), and

$$p(x|s_k = 1) = N(x|\mu_k, \Sigma_k)$$

is a conditional Gaussian distribution.
Gaussian mixture models

- The model is a generative model, where we first draw a mixture component from a discrete distribution, and then we draw the observation from a Gaussian, whose parameters depend on the component.
- To compute how likely a given observation $x_n$ is to come from a particular mixture component, use Bayes’ rule:
  \[
p(s_n|x_n) = \frac{p(x_n|s_n)p(s_n)}{\sum_{s}p(x_n|s)p(s)}
\]
- $p(s_{nk} = 1|x_n)$ is called responsibility of mixture component $k$.
- The Gaussian mixture is like a “soft” version of $K$-means.

The EM algorithm

- Instead of optimizing $L$, we will now optimize the lower bound $\mathcal{L}$ with respect to both the original parameters and the newly introduced auxiliary variables $q(s)$.
- To avoid clutter, it is convenient write this as
  \[
  \mathcal{L} = \sum_{nk} q_{nk} \log p(x_n|s_n = k)p(s_n = k) - \sum_{nk} q_{nk} \log q_{nk}
  \]
  using the abbreviation $q_{nk} = q(s_n = k)$
- The first term of $\mathcal{L}$ is the expectation of $\log p(x_n, s_n)$ with respect to $q(s_n)$. It is commonly referred to as “expected complete log-likelihood”.
- This is the only term that depends on the model parameters.
- For learning, set derivatives to zero:
  \[
  \partial \mathcal{L} / \partial \mu_k = \sum_n q_{nk} \sum_k (x_n - \mu_k) = 0
  \Rightarrow \mu_k = \frac{\sum_n q_{nk} x_n}{\sum_n q_{nk}}
  \]
- Similarly, one can derive
  \[
  \Sigma_k = \frac{\sum_n q_{nk} (x_n - \mu_k)(x_n - \mu_k)^T}{\sum_n q_{nk}}
  \]
  and
  \[
  \pi_k = p(s_k) = \frac{\sum_n q_{nk}}{N}
  \]

Mixture models are usually trained using the EM algorithm (though one could use gradient descent).

Given training data $\{x_n\}$, we can write

\[
L := \sum_n \log p(x_n) = \sum_n \log \sum_{s_n} p(x_n|s_n)p(s_n)
= \sum_n \log \sum_{s_n} q(s_n) \frac{p(x_n|s_n)p(s_n)}{q(s_n)}
\geq \sum_n \sum_{s_n} q(s_n) \log \frac{p(x_n|s_n)p(s_n)}{q(s_n)}
:= \mathcal{L}
\]
where we use Jensen’s inequality:

$\log \sum_i a_i b_i \geq \sum_i a_i \log b_i$ if $\forall i : a_i > 0$ and $\sum_i a_i = 1$
The EM algorithm

- But what about the auxiliary variables $q_{nk}$?
- We rewrite $\mathcal{L}$ once more in a different way:

$$\mathcal{L} = \sum_n \sum_{s_n} q(s_n) \log \frac{p(x_n | s_n) p(s_n)}{q(s_n)}$$

$$= \sum_n \sum_{s_n} q(s_n) \log \frac{p(s_n | x_n) p(x_n)}{q(s_n)}$$

$$= \sum_n \sum_{s_n} q(s_n) \log \frac{p(s_n | x_n)}{q(s_n)} + \sum_n \sum_{s_n} q(s_n) \log p(x_n)$$

$$= -\sum_n \text{KL}(q(s_n) \| p(s_n | x_n)) + \mathcal{L}$$

EM algorithm summary

1. E-step: Evaluate the posteriors $p(s_n | x_n)$.
2. M-step: Optimize $\mathcal{L}$ with respect to the model parameters, keeping $q(s_n) = p(s_n | x_n)$ fixed.

- The E-step computes the expected complete log-likelihood. It amounts to evaluating the responsibilities $p(s_n | x_n)$ for each point.
- The M-step maximizes the expected complete log-likelihood. In a Gaussian mixture, this amounts to setting parameters to responsibility-weighted sums.

EM optimizes a sequence of lower bounds

- Since the KL divergence is non-negative, setting $q(s_n) = p(s_n | x_n)$ will make the bound $\mathcal{L}$ on $L$ tight!
- But $p(s_n | x_n)$ is easy to compute, using Bayes’ rule.
- We already know how to optimize $\mathcal{L}$ with respect to the model parameters. So we can repeatedly compute (by inferring $q_{nk}$), and then optimize, a tight lower bound on $L$. 

EM algorithm summary

1. E-step: Evaluate the posteriors $p(s_n | x_n)$.
2. M-step: Optimize $\mathcal{L}$ with respect to the model parameters, keeping $q(s_n) = p(s_n | x_n)$ fixed.
Example: Training a Gaussian mixture with EM

EM algorithm comments

- The EM algorithm can be applied to many latent variable models, not just mixtures of Gaussians.
- The E-step and the M-step have to be derived individually for each model, but the view from the lower bound $\mathcal{L}$ of the log-likelihood is always the same.
- One of the first models that deployed EM was the Hidden Markov Model.
- For many other models, computing $p(s|x)$ is not tractable. In this case, it is still common to deploy a variation of EM, where we only improve the KL-divergence in the E-step rather than finding the exact posterior.