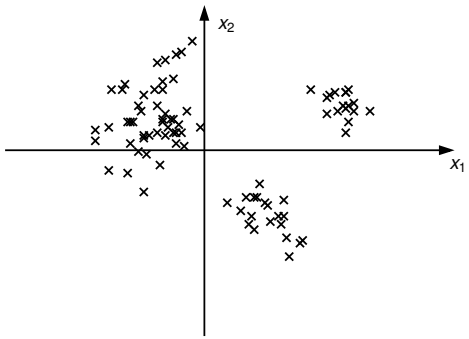
Machine learning for vision

Fall 2015

Roland Memisevic

Lecture 8, February 24, 2015

Review of classic (GOF) K-means clustering



- K-means is traditionally a clustering algorithm.
- **Learning:** Fit K prototypes $\boldsymbol{w}_k$ (the rows of some matrix, W) to training data-points $\boldsymbol{x}_n$.
- **Inference:** Given a point, find the nearest prototype.

Classic K-means clustering

- Define $\boldsymbol{s}_n$ as the one-hot encoding of the discrete variable representing the index of the nearest cluster center for $\boldsymbol{x}_n$.
- 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 $\boldsymbol{s}_n$ for each point $\boldsymbol{x}_n$.
- The K-means objective function measures the *average distance between points* $\boldsymbol{x}$ *and their representatives*:

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} s_{nk} \|\boldsymbol{x}_n - \boldsymbol{w}_k\|^2$$

Classic K-means clustering

- *Learning* amounts to finding *both* the prototypes $\boldsymbol{w}_k$ *and* the assignments $\boldsymbol{s}_n$ for each point, so as to minimize J.
- This seems like a tricky optimization problem, because the $\boldsymbol{s}_n$ are discrete and the $\boldsymbol{w}_k$ are continuous.
- But learning gets easy (*on paper*) if we decouple learning the $\boldsymbol{s}_n$ from learning the $\boldsymbol{w}_k$.
- In practice, it may be much better not to do this but to train online instead, as we will see later.

Classic K-means clustering

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} s_{nk} \|\boldsymbol{x}_n - \boldsymbol{w}_k\|^2$$

Finding the optimal $\boldsymbol{s}_n$

- Given the $\boldsymbol{w}_k$, we can optimize all the $\boldsymbol{s}_n$ 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 $\boldsymbol{w}_k$ is *closest*.
- Formally, to optimize all $\boldsymbol{s}_n$, given the set of $\boldsymbol{w}_k$, set:

$$s_{nk} = \begin{cases} 1 & \text{if } k = \arg\min_j \|\boldsymbol{x}_n - \boldsymbol{w}_j\|^2 \\ 0 & \text{otherwise.} \end{cases}$$

Classic K-means clustering

Finding the optimal $\boldsymbol{w}_k$

- Given S, J is a quadratic function of $\boldsymbol{w}_k$ which we can minimize by setting the derivative to zero:

$$2 \sum_{n=1}^{N} s_{nk}(\boldsymbol{x}_n - \boldsymbol{w}_k) = 0$$

- Solving for $\boldsymbol{w}_k$ yields:

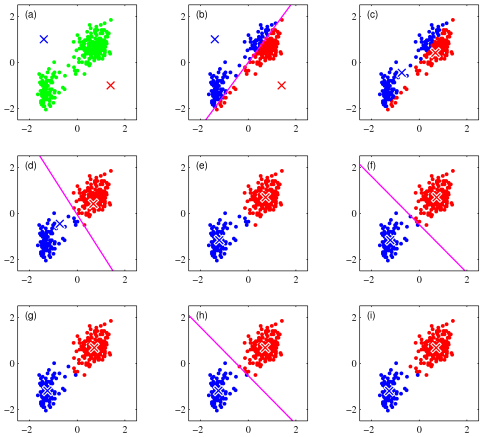
$$\boldsymbol{w}_k = \frac{\sum_n s_{nk} \boldsymbol{x}_n}{\sum_n s_{nk}}$$

- This solution has a simple interpretation: Set $\boldsymbol{w}_k$ to the mean of all points currently assigned to cluster k.

Classic K-means clustering

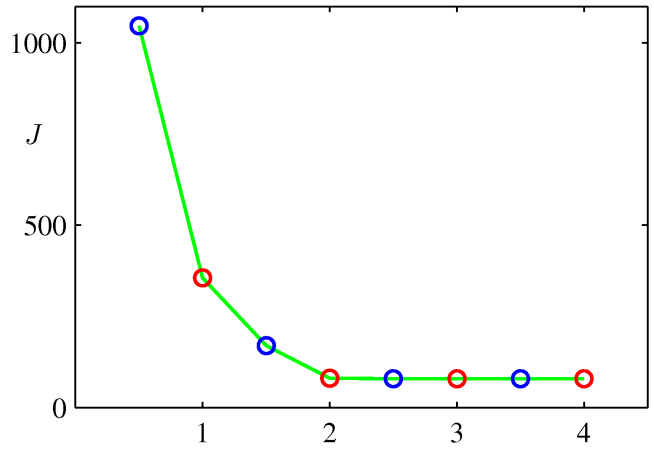
- Learning amounts to iterating inference for the $\boldsymbol{s}_n$, and adapting the parameters $\boldsymbol{w}_k$ until there are no more changes.
- This training procedure always converges: J is positive, and every step either decreases it or leaves it unchanged.
- But there can be local minima.
- One way to deal with this is to try multiple runs with different initializations for the parameters $\boldsymbol{w_k}$ 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



Classic K-means clustering

end of review

K-means inference

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

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

- The set of all K prototypes $\boldsymbol{w}_k$ is called *codebook*.
- Clustering and K-means are also known as *vector quantization*.

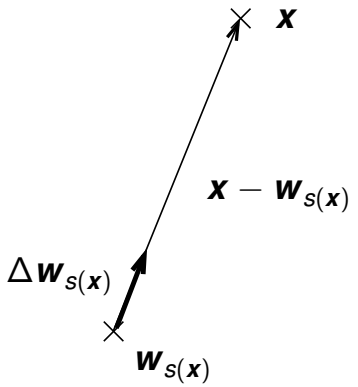
Why is K-means useful?

- Since K-means "tiles" space into locally constant regions, an arbitrary non-linear function (up to the tiling resolution, K) can be represented with a subsequent linear layer.
- K-means distributes cluster-centers in space, such that their 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 via online learning

- The reconstruction error for training point $\boldsymbol{x}$ may be written
$$E(W) = \frac{1}{2}\left(\boldsymbol{x} - \boldsymbol{w}_{s(\boldsymbol{x})}\right)^2$$
- Its gradient is
$$\frac{\partial E(W)}{\partial \boldsymbol{w}_i} = -\left(\boldsymbol{x} - \boldsymbol{w}_{s(\boldsymbol{x})}\right)\delta_{s(\boldsymbol{x}),i}$$
- So we can use the online learning rule:
$$\boldsymbol{w}_{s(\boldsymbol{x})} \leftarrow \boldsymbol{w}_{s(\boldsymbol{x})} + \eta\left(\boldsymbol{x} - \boldsymbol{w}_{s(\boldsymbol{x})}\right)$$
- (Here, it is easier to think of $s(x)$ as index rather than one-hot vector.)

Geometry of online K-means



- $\Delta \boldsymbol{w}_{s(\boldsymbol{x})} = \eta(\boldsymbol{x} - w_{s(\boldsymbol{x})})$ 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
$$\Delta \boldsymbol{w}_k = \eta \delta_{ks(\boldsymbol{x})}\left(\boldsymbol{x} - \boldsymbol{w}_k\right)$$
  where
$$\delta_{ks(\boldsymbol{x})} = \begin{cases} 1 & \text{if } s(\boldsymbol{x}) = k \\ 0 & \text{else} \end{cases}$$
  is the "post-synaptic activity" determined by competition ("winner takes all" rule)
- There are two learning terms:

K-means and Hebbian learning

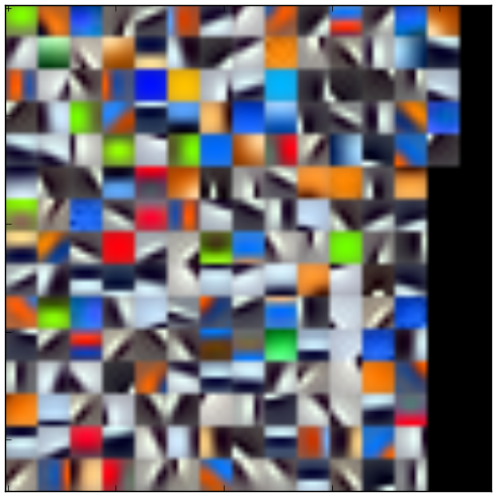
1. A Hebbian term:
$$\delta_{ks(\boldsymbol{x})}\boldsymbol{x}$$
2. An "unlearning" term:
$$-\delta_{ks(\boldsymbol{x})}\boldsymbol{w}_k$$

- 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.

Hebbian *K*-means in 9 lines of code

```
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 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

$$\Delta \boldsymbol{w}_k = \eta \delta_{k s(\boldsymbol{x})} \left(\boldsymbol{x} - \boldsymbol{w}_k \right)$$

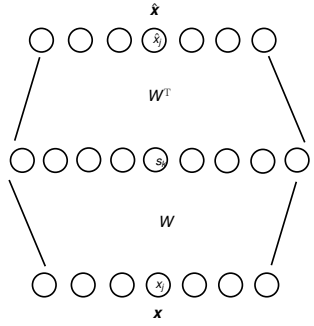
  into

$$\Delta \boldsymbol{w}_k = \eta h_{k s(\boldsymbol{x})} \left(\boldsymbol{x} - \boldsymbol{w}_k \right)$$

  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).

K-means as autoencoder

$$\begin{aligned} \boldsymbol{s} &= \mathrm{wta}(W^{\mathrm{T}} \boldsymbol{x}) \\ \hat{\boldsymbol{x}} &= W \boldsymbol{s} \end{aligned}$$



- The cost is $\sum_i \| \boldsymbol{x}_i - W \mathrm{wta}\left(W^{\mathrm{T}} \boldsymbol{x}_i \right) \|^2$
- wta = "winner takes all"
- Weights are "tied": Recognition weights are the transpose of generative weights.

The K-means energy function

- We can think of K-means as energy based learning, if we define the energy function as

$$E(\boldsymbol{x}) = \|\boldsymbol{x} - W\mathrm{wta}\left(W^{\mathrm{T}}\boldsymbol{x}\right)\|^2 = \|\boldsymbol{x} - \boldsymbol{w}_{s(\boldsymbol{x})}\|^2$$

- Since far from the cluster-centers the energy goes to ∞, K-means has low energy everywhere else.
- In other words, K-means 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 un-normalized probability for a point as

$$q(\boldsymbol{x}_n) = \exp(-E(\boldsymbol{x}_n))$$

  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 probabilistic models, there is no need to lower density away from the data in this model: "Negative updates are *built in*."
- (LeCun, 2006)

Winner-takes-all and lateral interactions

- The winner-takes-all function may be defined as

$$\mathrm{wta}(\boldsymbol{x}) = \mathrm{onehot}\left(\arg\min_k \|\boldsymbol{x} - \boldsymbol{w}_k\|^2\right)$$

  where $\boldsymbol{w}_k$ is a column of W.
- The squared distance can be written as

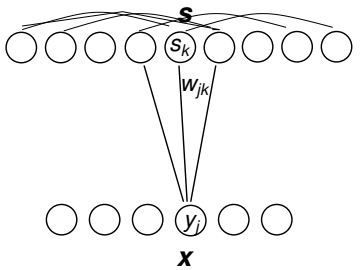
$$\|\boldsymbol{x} - \boldsymbol{w}_k\|^2 = \boldsymbol{x}^{\mathrm{T}}\boldsymbol{x} + \boldsymbol{w}_k^{\mathrm{T}}\boldsymbol{w}_k - 2\boldsymbol{w}_k^{\mathrm{T}}\boldsymbol{x}$$

- If all $\boldsymbol{w}_k$ have the same norm, inference amounts to finding the hidden unit which maximizes

$$\boldsymbol{w}_k^{\mathrm{T}}\boldsymbol{x}$$

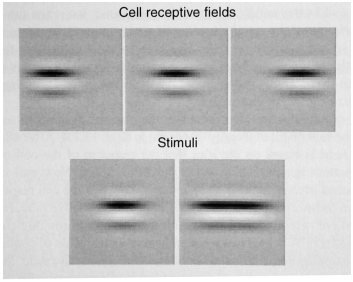
  which is the usual "simple cell" response plus competition.

Lateral interactions



- Since 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*.
- To induce competition, the interactions need to be *inhibitory*.

End-stopping



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

- 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.