

# ECE/CS 559 - Neural Networks Lecture Notes:

## Unsupervised Learning

Erdem Koyuncu

This is also referred to as self-organization. We similarly have a training set. But, for each member of the training set, we do not have a correct output information. The NN is required to organize itself, by possibly detecting the similarities between the patterns (statistical properties) and arranges its outputs accordingly. A typical case is clustering, which may be considered as a form of classification.

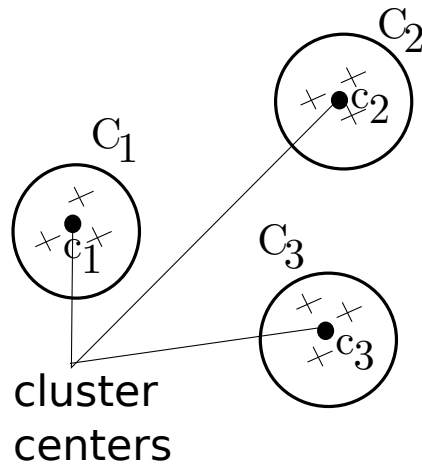


Figure 1: Clustering

- Suppose that the inputs will form clusters  $\mathcal{C}_i$  with cluster centers  $\mathbf{c}_i$ .
- If we know the cluster centers, we can then distinguish the clusters by distance: Given  $\mathbf{x} \in \mathcal{S}$ , we have  $\|\mathbf{x} - \mathbf{c}_i\| < \|\mathbf{x} - \mathbf{c}_j\|, \forall j \neq i \implies \mathbf{x} \in \mathcal{C}_i$ .
- Hence one way of looking at unsupervised learning is to find appropriate cluster centers for each clusters, hence detect the clusters in the given training set.
- Here, the cluster centers  $\mathbf{c}_i$  are also known as the quantization vectors.
- Unsupervised learning tries to minimize quantization error by finding appropriate cluster centers:
- If we have  $m$  clusters, the quantization error given  $\mathbf{c}_i, i = 1, \dots, m$  is given by

$$E = \sum_{\mathbf{x} \in \mathcal{S}} \min_i \|\mathbf{x} - \mathbf{c}_i\|^2$$

- The goal is then to minimize  $E$ , i.e. finding the appropriate cluster centers (quantization points)  $\mathbf{c}_i$  such that  $E$  is minimized.

# 1 $k$ -means algorithm

- In general, the problem of minimizing  $E$  can be solved iteratively. Let  $\mathcal{V}_i = \{\mathbf{x} : \|\mathbf{x} - \mathbf{c}_i\| \leq \|\mathbf{x} - \mathbf{c}_j\|\}$  denote the Voronoi cell for cluster center  $i$ , where ties are broken arbitrarily. Then, we have

$$E = \sum_{i=1}^m \sum_{\mathbf{x} \in \mathcal{V}_i} \|\mathbf{x} - \mathbf{c}_i\|^2 \quad (1)$$

This decomposition suggests the following algorithm:

- 1) Initialize cluster centers  $\mathbf{c}_1, \dots, \mathbf{c}_m$  (e.g., arbitrarily).
- 2) Until convergence (of the energy  $E$  or cluster centers)
  - 2.1) Calculate Voronoi cells  $\mathcal{V}_1, \dots, \mathcal{V}_m$  given cluster centers  $\mathbf{c}_1, \dots, \mathbf{c}_m$ .
  - 2.2) Set  $\mathbf{c}_i \leftarrow \min_{\mathbf{c}} \sum_{\mathbf{x} \in \mathcal{V}_i} \|\mathbf{x} - \mathbf{c}\|^2$ ,  $i = 1, \dots, m$ . In other words, we update each cluster center while considering the corresponding Voronoi cells to be fixed.

Note that  $\arg \min_{\mathbf{c}} \sum_{\mathbf{x} \in \mathcal{V}_i} \|\mathbf{x} - \mathbf{c}\|^2 = \frac{1}{|\mathcal{V}_i|} \sum_{\mathbf{x} \in \mathcal{V}_i} \mathbf{x}$ , i.e. the solution of the optimization problem is the geometric centroid of the voronoi region  $\mathcal{V}_i$ . Thus, instead of 2.2 above, I can equivalently write:

- 2.2) Set  $\mathbf{c}_i \leftarrow \frac{1}{|\mathcal{V}_i|} \sum_{\mathbf{x} \in \mathcal{V}_i} \mathbf{x}$ ,  $i = 1, \dots, m$ .

This is known as the  $k$ -means algorithm or the Lloyd algorithm. Each step of the algorithm provides a lower energy  $E$ , and thus the algorithm is guaranteed to converge in terms of the energy-function sense. The convergence of the cluster centers is another matter that we will not go into, but typically you may assume that you also have the convergence of the cluster centers.

It is possible to reinterpret the algorithm using neural networks, as we will show soon.

# 2 Hebbian Learning

Another important case we will look at is Hebbian learning.

- This learning rule was first proposed by D. Webb in 1940's. Hebb's idea was (1949):
- When an axon of a cell A is near enough to excite a cell B and repeatedly or persistently fires it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased.
- Nodes that tend to be either both positive or both negative at the same time have strong positive weights, while those that tend to be opposite have strong negative weights.
- Mathematically, we have  $\mathbf{w}_i(n+1) = \mathbf{w}_i(n) + \eta \mathbf{x} y_i(n)$ , where  $y_i(n)$  is the output of neuron  $i$  for training sample  $n$ . This is also called the activity product rule.
- This type of update leads to saturation problems. For example, suppose  $y_i(n) = \text{sgn}(\mathbf{x}^T \mathbf{w}_i(n))$ , with the initial condition  $\mathbf{w}_i(n) = \mathbf{1}$ , and consider a fixed input  $\mathbf{x} = \mathbf{1}$ . We obtain the sequence **1, 2, 4, 8**, with exponential divergence to infinity.
- So, sometimes, the update  $\mathbf{w}_i(n+1) = \alpha \mathbf{w}_i(n) + \eta \mathbf{x} y_i(n)$  is used, where  $\alpha < 1$ . For the previous example, we obtain  $1, 1+\alpha, 1+\alpha(1+\alpha), 1+\alpha(1+\alpha(1+\alpha)), \dots$ . Solution  $1+\alpha x = x$ ,  $x = \frac{1}{1-\alpha}$  (finite).
- Sometimes, this update is also used  $\mathbf{w}_i(n+1) = \mathbf{w}_i(n) + \eta(\mathbf{x} - \bar{\mathbf{x}})(y_i(n) - \bar{y}_i)$ . Here,  $\bar{\mathbf{x}}$  and  $\bar{y}_i$  are time averages of their respective quantities.
- Example: Suppose  $\mathcal{C}_1 = \left\{ \mathbf{x}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \mathbf{x}_2 = \begin{bmatrix} 0.9 \\ 1 \end{bmatrix}, \mathbf{x}_3 = \begin{bmatrix} 1 \\ 1.1 \end{bmatrix} \right\}$ , and  $\mathcal{C}_2 = \left\{ \mathbf{x}_4 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, \mathbf{x}_5 = \begin{bmatrix} 1.1 \\ -1 \end{bmatrix}, \mathbf{x}_6 = \begin{bmatrix} 1 \\ -1.1 \end{bmatrix} \right\}$ .  
The problem is to find a single neuron that can distinguish between the two classes. Although the example introduces the two classes separately, we assume we do not know the desired output for any given pattern. Obviously, if the desired output were given for all patterns, we could have used the perceptron training algorithm to solve the classification task.

- Here, we consider weights without any thresholding. Suppose the initial weights are  $\mathbf{w}(1) = \begin{bmatrix} 0.1 \\ 0.1 \end{bmatrix}$ . Note that given input  $\mathbf{x}_1$ , the output is  $\text{sgn}(\mathbf{x}_1^T \mathbf{w}(1)) = 1$ . Given input  $\mathbf{x}_5$  on the other hand, the output is  $\text{sgn}(\mathbf{x}_5^T \mathbf{w}(1)) = 1$ . However  $\mathbf{x}_1$  and  $\mathbf{x}_5$  were supposed to be on different classes. Hence, the initial weights cannot distinguish the two classes.
- Begin Epoch: 1. Suppose  $\eta = 1$ .
  - Feed  $\mathbf{x}_1$ , the output is  $\text{sgn}(\mathbf{x}_1^T \mathbf{w}(1)) = 1$ . Hence, we set  $\mathbf{w}(2) = \mathbf{w}(1) + 1 \times 1 \times \mathbf{x}_1 = \begin{bmatrix} 1.1 \\ 1.1 \end{bmatrix}$ .
  - Feed  $\mathbf{x}_2$ , the output is  $\text{sgn}(\mathbf{x}_2^T \mathbf{w}(2)) = 1$ . Hence, we set  $\mathbf{w}(3) = \mathbf{w}(2) + 1 \times 1 \times \mathbf{x}_2 = \begin{bmatrix} 2 \\ 2.1 \end{bmatrix}$ .
  - Feed  $\mathbf{x}_3$ , the output is  $\text{sgn}(\mathbf{x}_3^T \mathbf{w}(3)) = 1$ . Hence, we set  $\mathbf{w}(4) = \mathbf{w}(3) + 1 \times 1 \times \mathbf{x}_3 = \begin{bmatrix} 3 \\ 3.2 \end{bmatrix}$ .
  - Feed  $\mathbf{x}_4$ , the output is  $\text{sgn}(\mathbf{x}_4^T \mathbf{w}(4)) = -1$ . Hence, we set  $\mathbf{w}(5) = \mathbf{w}(4) - 1 \times 1 \times \mathbf{x}_4 = \begin{bmatrix} 2 \\ 4.2 \end{bmatrix}$ .
  - Feed  $\mathbf{x}_5$ , the output is  $\text{sgn}(\mathbf{x}_5^T \mathbf{w}(4)) = -1$ . Hence, we set  $\mathbf{w}(6) = \mathbf{w}(5) - 1 \times 1 \times \mathbf{x}_5 = \begin{bmatrix} 0.9 \\ 5.2 \end{bmatrix}$ .
  - Feed  $\mathbf{x}_6$ , the output is  $\text{sgn}(\mathbf{x}_6^T \mathbf{w}(4)) = -1$ . Hence, we set  $\mathbf{w}(7) = \mathbf{w}(6) - 1 \times 1 \times \mathbf{x}_6 = \begin{bmatrix} -0.1 \\ 6.3 \end{bmatrix}$ .
- We can proceed with more epochs like this, but let us stop at the end of Epoch 1. It can be verified that the final weight vector  $\mathbf{w}(7)$  provides an output of 1 for input patterns  $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$ , and it provides an output of  $-1$  for input patterns  $\mathbf{x}_4, \mathbf{x}_5, \mathbf{x}_6$ .

### 3 Competitive learning

- Also called the winner takes all learning rule.
- Output neurons compete around themselves to be active.
- Consider a one-layer network. Suppose we have  $m$  input neurons and  $n$  output neurons without any bias, and the activator function is identity. Let  $\mathbf{W} \in \mathbb{R}^{n \times m}$  denote the weight matrix with the weight going to output neuron  $i$  from neuron  $j$  denoted by  $w_{ij}$ . Then, given input  $\mathbf{x} \in \mathbb{R}^{m \times 1}$ , the induced local fields are  $\mathbf{v} = \mathbf{W}\mathbf{x} \in \mathbb{R}^{n \times 1}$  and the outputs are  $\mathbf{y} = \phi(\mathbf{v}) \in \mathbb{R}^{n \times 1}$ . The weights for the  $i$ th output neuron is then  $i$ th row  $\mathbf{w}_i$  of  $\mathbf{W}$ . We may also use the extended notation.
- After this, we look at the output neuron with the largest output. Due to the monotonicity of the activation function, this neuron is the one with the largest induced local field.
- Suppose that the  $i$ th neuron has the largest induced local field, i.e.  $v_i = \max_{j \in \{1, \dots, n\}} v_j$  according to some tie-breaking rule. Then, the  $i$ th neuron is declared the winning neuron, and its outputs are updated as  $\mathbf{w}_i(n+1) = \mathbf{w}_i(n) + \eta(\mathbf{x} - \mathbf{w}_i(n))$ . The others remain the same, i.e.  $\mathbf{w}_j(n+1) = \mathbf{w}_j(n)$ ,  $j \neq i$ . Here,  $\eta$  is the learning constant.
- If  $\|\mathbf{w}_i\| = \|\mathbf{w}_j\|$  for every  $i \neq j$ , then the winning neuron coincides with the neuron whose weight vector is closest to the input  $\mathbf{x}$  in terms of the Euclidean distance - prove this. So, roughly speaking: The learning rule has the effect of moving the input  $\mathbf{x}$  towards the direction of the winning neuron. The winner is more likely to win next time as well.
- Usually, for the success of the algorithm, most of the time input vectors and weights are normalized.

- Suppose that some input pattern  $\mathbf{x}$  is repeatedly applied, and at each time, neuron  $i$  wins. We have

$$\begin{aligned}
\mathbf{w}_i(2) &= \mathbf{w}_i(1) + \eta(\mathbf{x} - \mathbf{w}_i(1)) = (1 - \eta)\mathbf{w}_i(1) + \eta\mathbf{x} \\
\mathbf{w}_i(3) &= \mathbf{w}_i(2) + \eta(\mathbf{x} - \mathbf{w}_i(2)) = (1 - \eta)^2\mathbf{w}_i(1) + \eta(1 + (1 - \eta))\mathbf{x} \\
&\vdots \\
\mathbf{w}_i(n+1) &= (1 - \eta)^n\mathbf{w}_i(1) + \eta(1 + (1 - \eta) + \cdots + (1 - \eta)^{n-1})\mathbf{x} \\
&= (1 - \eta)^n\mathbf{w}_i(1) + (1 - (1 - \eta)^n)\mathbf{x}
\end{aligned}$$

If  $|\eta| < 1$ , then we have  $\mathbf{w}_i(n+1) \rightarrow \mathbf{x}$  as  $n \rightarrow \infty$ . Hence, if we apply patterns from some cluster  $\mathcal{C}$ , then (eventually) we expect the same neuron to win the competition, and that the weight of the winner neuron converges to the cluster center of  $\mathcal{C}$  (Typically, for this to precisely happen, the learning parameter should also go to zero as the number of epochs go to infinity. Otherwise, the winning neuron will oscillate around the cluster center of  $\mathcal{C}$ ).

Example: Say the learning parameter is 0.5 with  $\mathbf{x}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ ,  $\mathbf{x}_2 = \begin{bmatrix} 0.9 \\ 1 \end{bmatrix}$ ,  $\mathbf{x}_3 = \begin{bmatrix} 1 \\ 1.1 \end{bmatrix}$ ,  $\mathbf{x}_4 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$ ,  $\mathbf{x}_5 = \begin{bmatrix} 1.1 \\ -1 \end{bmatrix}$ ,  $\mathbf{x}_6 = \begin{bmatrix} 1 \\ -1.1 \end{bmatrix}$ .

Suppose  $\mathbf{w}_1(1) = \begin{bmatrix} 0.8 \\ 0 \end{bmatrix}$ ,  $\mathbf{w}_2(1) = \begin{bmatrix} 0.9 \\ 0.1 \end{bmatrix}$

$[\mathbf{w}_1(1)]^T \mathbf{x}_1 = 0.80$ ,  $[\mathbf{w}_2(1)]^T \mathbf{x}_1 = 1.00 \implies 2$  wins.

$[\mathbf{w}_1(1)]^T \mathbf{x}_2 = 0.72$ ,  $[\mathbf{w}_2(1)]^T \mathbf{x}_2 = 0.91 \implies 2$  wins.

$[\mathbf{w}_1(1)]^T \mathbf{x}_3 = 0.80$ ,  $[\mathbf{w}_2(1)]^T \mathbf{x}_3 = 1.01 \implies 2$  wins.

$[\mathbf{w}_1(1)]^T \mathbf{x}_4 = 0.80$ ,  $[\mathbf{w}_2(1)]^T \mathbf{x}_4 = 0.80 \implies$  Draw.

$[\mathbf{w}_1(1)]^T \mathbf{x}_5 = 0.88$ ,  $[\mathbf{w}_2(1)]^T \mathbf{x}_5 = 0.89 \implies 2$  wins.

$[\mathbf{w}_1(1)]^T \mathbf{x}_6 = 0.80$ ,  $[\mathbf{w}_2(1)]^T \mathbf{x}_6 = 0.79 \implies 1$  wins.

Epoch - 1

- $\mathbf{x}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ ,  $[\mathbf{w}_1(1)]^T \mathbf{x}_1 = 0.80$ ,  $[\mathbf{w}_2(1)]^T \mathbf{x}_1 = 1.00 \implies 2$  wins. Thus, we update

$$\mathbf{w}_1(2) = \mathbf{w}_1(1) = \begin{bmatrix} 0.8 \\ 0 \end{bmatrix},$$

$$\begin{aligned}
\mathbf{w}_2(2) &= \mathbf{w}_2(1) + \alpha(\mathbf{x}_1 - \mathbf{w}_2(1)) \\
&= \begin{bmatrix} 0.9 \\ 0.1 \end{bmatrix} + 0.5 \left( \begin{bmatrix} 1 \\ 1 \end{bmatrix} - \begin{bmatrix} 0.9 \\ 0.1 \end{bmatrix} \right) = \begin{bmatrix} 0.95 \\ 0.55 \end{bmatrix}.
\end{aligned}$$

- $\mathbf{x}_2 = \begin{bmatrix} 0.9 \\ 1 \end{bmatrix}$ ,  $[\mathbf{w}_1(2)]^T \mathbf{x}_2 = 0.72$ ,  $[\mathbf{w}_2(2)]^T \mathbf{x}_2 = 1.405 \implies 2$  wins. Thus, we have

$$\mathbf{w}_1(3) = \begin{bmatrix} 0.8 \\ 0 \end{bmatrix},$$

$$\mathbf{w}_2(3) = \begin{bmatrix} 0.925 \\ 0.775 \end{bmatrix}$$

- $\mathbf{x}_3 = \begin{bmatrix} 1 \\ 1.1 \end{bmatrix}$ ,  $(0.8, 1.7775)$  2 wins. Thus, we have

$$\mathbf{w}_1(4) = \begin{bmatrix} 0.8 \\ 0 \end{bmatrix},$$

$$\mathbf{w}_2(4) = \begin{bmatrix} 0.9625 \\ 0.9375 \end{bmatrix}$$

- $\mathbf{x}_4 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$ , : (0.8, 0.025) 1 wins. Thus, we have

$$\begin{bmatrix} 0.9 \\ -0.5 \end{bmatrix}, \begin{bmatrix} 0.9625 \\ 0.9375 \end{bmatrix}$$

- $\mathbf{x}_5 = \begin{bmatrix} 1.1 \\ -1 \end{bmatrix}$ , : (1.49, 0.12125) 1 wins. Thus, we have

$$\begin{bmatrix} 1 \\ -0.75 \end{bmatrix}, \begin{bmatrix} 0.9625 \\ 0.9375 \end{bmatrix}$$

- $\mathbf{x}_6 = \begin{bmatrix} 1 \\ -1.1 \end{bmatrix}$ , : (1.825, -0.06875) 1 wins. Thus, we have

$$\begin{bmatrix} 1 \\ -0.925 \end{bmatrix}, \begin{bmatrix} 0.9625 \\ 0.9375 \end{bmatrix}$$

The final weights can do the clustering as intuitively desired.

## 4 Batch competitive learning interpretation of $k$ -means

Note the decomposition  $\|\mathbf{x} - \mathbf{c}_i\|^2 = \|\mathbf{x}\|^2 - 2\mathbf{x}^T \mathbf{c}_i + \|\mathbf{c}_i\|^2$ . So, if we consider a network of  $m$  neurons, where the  $i$ th neuron has weights  $2\mathbf{c}_i$  and bias  $-\|\mathbf{c}_i\|^2$ , then the winning neuron (the one with the largest induced local field) coincides with the neuron for which  $\mathbf{c}_i$  is closest to  $\mathbf{x}$ . Consider showing all  $\mathbf{x} \in \mathcal{S}$ , where  $\mathcal{S}$  is the training set, and recording the winning neurons for each example shown. Equivalently, we are calculating the Voronoi regions  $\mathcal{V}_1, \dots, \mathcal{V}_m$  given the cluster centers  $\mathbf{c}_1, \dots, \mathbf{c}_n$ . At the end of the Epoch, we update the weights and the biases of each neuron according to the  $k$ -means update rule described previously, i.e. we set the weights of the  $i$ th neuron to be  $2\frac{1}{|\mathcal{V}_i|} \sum_{\mathbf{x} \in \mathcal{V}_i} \mathbf{x}$ , and the bias of the  $i$ th neuron to be  $-\|\frac{1}{|\mathcal{V}_i|} \sum_{\mathbf{x} \in \mathcal{V}_i} \mathbf{x}\|^2$ . This variant of a competitive batch learning rule coincides with the  $k$ -means algorithm.