Modeling Neural Populations with Mixture Models

Santiago M. Castro Dau

July 2022

Abstract

The central motivation for this work is to provide an alternative perspective on the online K-means algorithm which is known to be analogous to a Winner takes it all network. Furthermore we seek to improve said algorithm such that it becomes immune to the uneven membership assignment problem, where occasionally and depending on initialization one centroid is assigned responsibility over multiple clusters of data points while others are left without any.

We start by introducing Winner takes it all networks and their connection to the K-means algorithm. Then we introduce Mixture Models and show that the K-means algorithm can be seen as an instance of parameter inference of a Gaussian Mixture Model. By showing this equivalence we point out a natural way to get rid of uneven membership assignment.

Then, using a Poisson Mixture Model we formulate an algorithm that we think is more fitting to represent neural populations and their dynamics. Finally we do some simulations and demonstrate the potential of the proposed algorithm.

This work is very similar to [6] and [4] specially in terms of the theory behind it, but the proposed algorithm differs from those proposed in these papers. We argue that the framework used here and in the aforementioned papers provide a probabilistic interpretation of the underlying learning process.

Contents

1 Winner Take It All Networks

Winner Takes it All Networks (WTAn) are a special kind of artificial neural network (ANN) for classification. They consist of two fully connected layers, the input and the output layer.

Input layer

We will refer to the i^{th} value associated with the i^{th} input neuron with x_i, having in total M neurons in this layer. The entire input vector is denoted by ${}^{n}x$, where n denoted the index of the input, having in total N of these input vectors in one data set.

Output layer

Likewise we will refer to any of the K neurons in the output layer as neuron z_j. The goal is to optimally classify the inputs using the K output neurons.

Connections between layers

Associated with every connection between any x_i and z_j neurons we have the weight λ_{ji}. Every output neuron has a collection of weights associated to it denoted by $\lambda_{j:}$. We use Λ to describe all of the weights in matrix from where $\lambda_{j:}$'s are the rows of the matrix.

WTAn have been studied in computational neuroscience because they provide a simple model for a common connectivity pattern in the brain cortex, where neurons compete with each other for activation through lateral inhibition [4].

Lateral inhibition is modeled by making it so only one neuron is activated in the output layer per input. The winning neuron is the one with weights $\lambda_{j:}$ that are the closest, according to some measure of distance, to the n^{th} input vector x [1]. It is important to note that in these networks the inputs do not get multiplied by the weights like in most ANNs, but are simply compared to the inputs on every iteration in order to find the closest $\lambda_{j:}$.

Learning takes place through adjustments of the weights, Λ, every time an input is presented to the network (e.g in a n online fashion). The idea is that given an input pattern ${}^{n}x$, the associated weights $\lambda_{j:}$ of winning neuron z_j gets adjusted such that if the same or a similar input ${}^{n+1}x$ is presented to the network then z_j is again the winner, thereby successfully classifying the input.

Even thou it is convenient to think of all $\lambda_{j:}$ weights associated with winning neuron z_j to be updated as a whole in vector form, we can also think of the update as taking place individually for each element of λ_{ji}, associated with input neuron x_i and output neuron z_j. The weight updates then become akin to Hebbain learning where learning happens locally at each synapse.

[1]When its not important to indicate the index of the data point, x will be used instead of ${}^{n}x$

If we restrict the weights and the inputs to be normalized, such that the sum of the elements in the input vectors x and the weight vectors $\lambda_{j:}$ are equal to a constant α (1) (2), we can implement the learning by nudging the previous weight vectors $\lambda_{j:}$ in the direction of the input. This nudge would be a function of the difference between $\lambda_{j:}$ and input vector x (3). [2]

$$\begin{aligned}
\sum_{i=1}^{M} \lambda_{ji} &= \alpha && (1)\\
\sum_{i=1}^{M} x_i &= \alpha && (2)\\
\lambda_{j:}^{(t)} &\leftarrow \lambda_{j:}^{(t-1)} - \eta\epsilon'(x - \lambda_{j:}^{(t-1)}) && (3)
\end{aligned}$$

Here $^{(t)}$ indicates the index of the iteration in the context of an algorithm, and η is a small positive learning rate.

If we choose the squared L2 norm as our measure of distance $\epsilon(x - \lambda_{j:}^{(t-1)})$ we can take the derivative w.r.t $\lambda_{j:}$ and use the result as the update function $\epsilon'(x - \lambda_{j:}^{(t-1)})$.

$$\begin{aligned}
\epsilon(x - \lambda_{j:}^{(t-1)}) &= \frac{1}{2}||x - \lambda_{j:}^{(t-1)}||_2^2 && (4)\\
\frac{d}{d\lambda_{j:}^{(t-1)}}\epsilon(x - \lambda_{j:}^{(t-1)}) &= \frac{d}{d\lambda_{j:}^{(t-1)}}\frac{1}{2}||x - \lambda_{j:}^{(t-1)}||_2^2 && (5)\\
&= -(x - \lambda_{j:}^{(t-1)}) && (6)\\
&= \epsilon'(x - \lambda_{j:}) && (7)\\
\lambda_{j:}^{(t)} &\leftarrow \lambda_{j:}^{(t-1)} + \eta(x - \lambda_{j:}^{(t-1)}) && (8)
\end{aligned}$$

The result is an online version of the K-means algorithm (Algorithm 1), where upon an input, the closest centroid (here the weights $\lambda_{j:}$), gets pulled towards the coordinates of the input x (8).

A perfect equivalence with the K-means algorithm would require that all the inputs ${}^{n}x$ to be presented at once. In such case we would simply adjust our cost function to account for all the data points N' associated to the winning neuron z_j. By minimizing this quantity, applying the derivative, setting it to zero and solving for $\lambda_{j:}$ (9) - (11), we find that the update is the same as in the K-means algorithm (Algorithm 2), showing the equivalence between these two seemingly different clustering sachems.

[2]For more information on the biological plausibility and of normalization in biological neural circuits we refer the reader to [4].

```
Algorithm 1 Online K-means
    Step 0. Initialize parameters for each centroid.
        $\Lambda \leftarrow \text{init.}$
  Step 1. For every incoming $n$ data point find the closes cluster.
        ${}^{n}z_j^{(t)} = \underset{\lambda_{j:}}{\text{argmin}} \|{}^{n}x - \lambda_{j:}\|_2^2$
  Step 2. Update the winning centroid $z_j$
        $\lambda_{j:}^{(t)} \leftarrow \lambda_{j:}^{(t-1)} + \eta(x - \lambda_{j:}^{(t-1)})$
  Step 3. Repeat step 1 and 2 for every incoming input
```

$$\epsilon(X - \lambda_{j:}) = \frac{1}{2}\sum_{n=1}^{N'} ||{}^{n}x - \lambda_{j:}||_2^2 \tag{9}$$

$$\hat{\lambda_{j:}} = \underset{\lambda_{j:}}{\text{argmin}} \frac{1}{2}\sum_{n=1}^{N'} ||{}^{n}x - \lambda_{j:}||_2^2 \tag{10}$$

$$\hat{\lambda_{j:}} = \frac{1}{N'}\sum_{n=1}^{N'} ({}^{n}x) \tag{11}$$

```
Algorithm 2 K-means
    Step 0. Initialize parameters for each centroid.
        $\Lambda \leftarrow \text{init.}$
  Step 1. For all $N$ data points find the closes cluster and assign it membership
  over the data point.
        ${}^{n}z_j^{(t)} = \underset{\lambda_{j:}}{\text{argmin}} \|{}^{n}x - \lambda_{j:}\|_2^2$
  Step 2. Update the $K$ centroids, with the average of the coordinates of its
  children data points.
        $\lambda_{j:}^{(t+1)} = \frac{1}{N'}\sum_{n=1}^{N'} ({}^{n}x)$
  Step 3. Repeat step 1 and 2 until convergence
```

The vanilla WTAn as well as the K-means algorithm have the problem that because only the weights $\lambda_{j:}$ associated to the wining neuron z_j get updated, it is common for some centorids to be assigned two or more clusters of data points, while other centroids are left without “children” data points and hence never get updated. Here we call this issue uneven membership assignment.

This is partly due to the “hardness” of the assignment step, or in other words the fact that only one neuron can win and therefore represent a given input. By modeling centroids as being only partly responsible for a given data point, instead of belonging exclusively to one, we can bypass this problem. This idea of responsibility allows us to model multiple output neurons getting activated upon an input, which is a more realistic model of the dynamics in a population of neurons. In order to formally define this notion of responsibility we must first introduce mixture models.

2 Mixture Models

A mixture model is a probabilistic model for representing the presence of sub populations within an overall population, without requiring that an observed data set identifies the sub-population to which an individual observation belongs (e.g without labels). Usually mixture models are used to make statistical inferences about the properties of the sub-populations given only observations on the pooled population [9].

Just by reading this description we can already sense that mixture models might be a good fit for modeling the classification of patterns as in the WTAn, where we seek to differentiate between distinct input sub-populations. The idea is that each observations is distributed according to a mixture of components, where all the components belong to the same parametric family of distributions but with different parameters (12). The parameters of each component represent the distinct sub-populations and so we then seek to find the parameters that best explain our data in terms of some number of sub-populations. We specify a mixture model with he following notation (note the correspondence to the previous section).

- K is the number of mixture components, individual components indicated with subscript j. Sometimes we refer to components as clusters, mixtures, sub-populations, and further on as output neurons.
- N is the number of observations, individual observations indicated with left superscript n, where each observation ${}^{n}x \in R^{M}$ and ${}^{n}x_i$ is the i^{th} element of the n^{th} observation.
- w is the vector of mixture weights where w_j is the mixture weight of component j. Can also be expressed as the prior probability $P(z_j|\theta)$ of a random data point belonging to any of the K components. Note that $\sum_{j=1}^{k} w_j = 1$
- Λ is the matrix of parameters, where rows $\lambda_{j:}$ are the parameters that correspond to the j^{th} component. λ_{ji} denotes the the ji^{th} entree of Λ.

- ${}^{n}z \in R^{K}$ is the vector of probabilities where ${}^{n}z_j$ is RV representing the probability of data point n belonging to cluster j.
- We will use z_j without superscript as a marker variable indicating that we are referring only to the distribution of component j. For example $P(x|z_j, \theta)$. Indicates the probability that standpoint x came from the distribution of component j.
- ${}^{(t)}$ indicates the index of the iteration in the context of an algorithm.
- $\theta = \{w, \Lambda\}$ is the set of all parameters

The likelihood of one data point can be written as follows, where $P(x|z_j, \theta)$ is the probability density function of the j^{th} component.

$$\mathcal{L}(x : \theta) = \sum_{j=1}^{K} P(z_j|\theta) P(x|z_j, \theta) \tag{12}$$

$$\mathcal{L}(x : \theta) = \sum_{j=1}^{K} w_j P(x|z_j, \theta) \tag{13}$$

There is two main inference problems when dealing with mixture models. On one hand one would like to know the number and functional form (parametric family) of the components within a mixture. This is usually referred to as the system identification. On the other hand one would like to estimate the corresponding parameter values given a fixed number of components and their functional form. This is referred to as parameter estimation [9]. In this work we concern ourselves with he problem of parameter estimation, since we assume that the number of components and their underlying distributions is fixed. Furthermore we take Expectation Maximization as or method of choice for inference. [3]

In the context of mixture models, the EM can be seen as unsupervised learning algorithm. This is interesting because if it is applicable to learning in WTAs it loosely implies that it might be useful to reason about learning in the brain as an unsupervised learning procedure, where the hidden RVs are the stimuli which generate specific input pattern distributions (the observed RV) that the brain then categorizes as belonging to the stimuli or cause by tuning the weights of the synapses.

2.1 The EM algorithm

After initialization of the parameters, the EM starts by performing the expectation step (E-step), where expectation values for the membership variables of

[3]The reason behind this choice is that we will show that a constrained form of EM for Gaussian Mixture Models is equivalent to K-means.

each data point are computed, followed by and maximization step, or M step which completes one full iteration. These two steps are repeated until the parameters converge.

In the E step we seek to find the responsibility $\gamma_j({}^n x)$ of each cluster on each data point ${}^n x$, or in other words the probability that data point ${}^n x$ belongs to cluster j.

$$\begin{aligned} \gamma_j({}^n x) &= P(z_j|{}^n x, \theta) && (14) \\ &= \frac{P(z_j, {}^n x, \theta)}{P({}^n x, \theta)} && (15) \\ &= \frac{P({}^n x|z_j, \theta)P(z_j|\theta)}{P({}^n x|\theta)} && (16) \\ P({}^n x|\theta) &= \sum_k P({}^n x|\theta, z_k)P(z_k|\theta) && (17) \end{aligned}$$

We know from before that $P(z = j|\theta) = w_j$ are the prior probabilities and $P({}^n x|\theta, z = k)$ is the likelihood, which depends on the parametric family we decide on.

$$\gamma_j({}^n x) = \frac{w_j P({}^n x|\theta, z_k)}{\sum_{j=1}^{K} w_k P({}^n x|\theta, z_k)} \qquad (18)$$

Once we have all the γ's for each data point, we again have a fully annotated data set, namely the responsibilities and the coordinates of the data points. We can now move on to infer the parameters θ in the M step using the previously estimated γ's. For this we take the expression of the likelihood of the data given the parameters and we maximize its lower bound. The derivation is as follows.

We can rewrite equation (18) accounting for all N data points, applying the log and then Jensen's inequality to get the lower bound on the data likelihood [4]. We maximize this quantity (21) instead of the log likelihood directly simply because of the log likelihood is hard to evaluate.

$$\begin{aligned} \log \mathcal{L}(D:\theta) &\geq b(\theta) && (19) \\ \log \mathcal{L}(D:\theta) &= \sum_{n=1}^{N} \log \sum_{j=1}^{K} w_j P({}^n x|z_j, \theta) && (20) \\ b(\theta) &= \sum_{n=1}^{N} \sum_{j=1}^{K} \gamma_j({}^n x) \log \frac{w_j P({}^n x|z_j, \theta)}{\gamma_j({}^n x)} && (21) \end{aligned}$$

[4] For a more detailed derivation we refer the reader [5].

We can then expand the logarithm, distribute $\gamma_j({}^n x)$, and split the sum.

$$b(\theta) = \sum_{n=1}^{N}\sum_{j=1}^{K} \gamma_j({}^n x) \log w_j P({}^n x|z_j, \theta) - \sum_{n=1}^{N}\sum_{j=1}^{K} \gamma_j({}^n x) \log \gamma_j({}^n x) \tag{22}$$

Because the $\gamma_j({}^n x)$'s are already fixed from the E step we can concentrate only on the first term which we term $Q(\theta)$.

$$Q(\theta) = \sum_{n=1}^{N}\sum_{j=1}^{K} \gamma_j({}^n x) \log w_j P({}^n x|z_j, \theta) \tag{23}$$

To get an estimate for the our parameters θ we perform the partial derivatives with respect to each λ_{ji} and w_j, make it equal to 0, and solve for the corresponding parameter. This will look different depending on the parametric family we choose and also the assumptions we make (e.g if we uniform fixed priors or we have them fixed).

Algorithm 3 EM algorithm

```
    **Step 0.** Initialize parameters.

        $\theta \leftarrow \text{init.}$

  **E step.** Compute $\gamma_j({}^n x)$ for each $n$ data point and each component $j$.

        $\gamma_j({}^n x) = \frac{w_j P({}^n x|\theta, z_j)}{\sum_{j=1}^{K} w_k P({}^n x|\theta, z_j)}$

  **M-step.** Optimize the lower bound of the log likelihood $Q(\theta)$ to update $\theta$.

        $\frac{dQ(\theta)}{d\theta} = \frac{d}{d\theta} \sum_{n=1}^{N}\sum_{j=1}^{K} \gamma_j({}^n x) \log w_j P({}^n x|z_j, \theta)$

  **Step 3.** Repeat E and M step until $\theta$ converges.
```

2.2 Gaussian Mixture Models

In the following subsection we show that by considering the components of a mixture model to be Gaussian and by further constraining it, namely by enforcing fixed spherical variances in all components and fixed uniform mixture weights, we recover the K-means algorithm, which implies that we might be able to use the EM algorithm to device a more general procedure for learning with a WTAn. This constrained version of the EM algorithm is termed the Hard. We then show how by easing up these restrictions we obtain a similar version of K-means but with soft membership assignment, which naturally eases the uneven membership problem. This is called the Soft EM for Gaussian Mixture Models (GMMs).

2.2.1 The Soft EM for GMMs

We simply have to plug in our Gaussian assumption, e.g. $P({}^{n}x|\theta, z_j) = \mathcal{N}({}^{n}x|\theta, z_j)$. The parameters then become $\theta = \{w, \mu, \Sigma$, where each component j has its own collection of parameters $\{w_j, \mu_j, \Sigma_j\}$ [5].

Algorithm 4 Soft EM for GMMs

Step 0. Initialize parameters $\theta = \{w, \mu, \Sigma\}$.

$$\theta \leftarrow \text{init}.$$

E step. Compute $\gamma_j({}^{n}x)$ for each n data point and each component j.

$$\gamma_j({}^{n}x) = \frac{w_j \mathcal{N}({}^{n}x|\theta, z_j)}{\sum_{j=1}^{K} w_k \mathcal{N}({}^{n}x|\theta, z_k)}$$

M-step. Update θ.

$$w_j^{(t)} \leftarrow \frac{1}{n} \sum_{i=1}^{n} \gamma_j^{(t)}({}^{n}x)$$

$$\mu_j^{(t)} \leftarrow \frac{\sum_{i=1}^{n} \gamma_j^{(t)}({}^{n}x){}^{n}x}{\sum_{i=1}^{n} \gamma_j^{(t)}({}^{n}x)}$$

$$\Sigma_j^{(t)} \leftarrow \frac{\sum_{i=1}^{n} \gamma_j^{(t)}({}^{n}x)({}^{n}x - \mu_j^{(t)})({}^{n}x - \mu_j^{(t)})^T}{\sum_{i=1}^{n} \gamma_j^{(t)}({}^{n}x)}$$

Step 3. Repeat E and M step until θ converges.

2.2.2 The Hard EM for GMMs

For the Hard EM the E-step classifies each data point as belonging to only one of the clusters. Intuitively we choose the cluster with the highest $P(z_j|{}^{n}x, \theta)$, then we make $\gamma_j({}^{n}x) = 1$ for all but the wining component, and 0 for the rest. Then by enforcing fixed spherical variance in all components $(\Sigma_1, ..., \Sigma_K = I\sigma)$ and fixed uniform mixture weights $(\forall j, w_j = \frac{1}{K})$, we are only left with the updates of μ_j that simplify into the simple average of all the data points belonging to cluster j.

[5] Note that we assume each component to be multivariate Gaussian

We can also simplify the E-step as follows. As stated above we seek the cluster that maximizes $P(z_j|^n x, \theta)$.

$$^n z^{(t)} = \underset{j}{\text{argmax}} P(z_j|^n x, \theta^{(t-1)}) \tag{24}$$

We can decompose $P(z_j|^n x, \theta^{(t-1)})$ using Bayes rule to get the follwoing expression.

$$^n z^{(t)} = \underset{j}{\text{argmax}} \frac{1}{Z} P(z_j|\theta^{(t-1)}) P(^n x|z_j, \theta^{(t-1)}) \tag{25}$$

$\frac{1}{Z}$ is the normalization constant from Bayes rule which is also independent of z and therefor can be omitted in the maximization. By assuming that each cluster has the same size, $P(z|\theta^{(t-1)})$ to be constant we can take them out of the maximization argument.

$$^n z^{(t)} = \underset{j}{\text{argmax}} \frac{1}{\sqrt{(2\pi\sigma^2)^d}} \exp(\frac{-1}{2\sigma^2} \|^n x - \mu_j^{(t-1)}\|_2^2) \tag{26}$$

We can see again that the maximization is only over the squared 2-norm and we can also note that maximizing this quantity is equivalent to minimizing the 2-norm.

$$^n z^{(t)} = \underset{z}{\text{argmin}} \|^n x - \mu_j^{(t-1)}\|_2 \tag{27}$$

Thereby the E-step becomes to Step 1 of K-means (Algorithm 2).

We can now see that by considering constant variances and weights the Hard EM (Algorithm 5) performs exactly the same operations as the K-means (Algorithm 2) and is therefore equivalent.

By relaxing both the constant variances and weights constraint we allow the underlying cluster to have different sizes and elliptical shapes, which allowed us to capture richer structures in the data. Doing so often leads to enhanced performance specially when the clusters are overlapping.

Likewise, by using membership instead of hard assignments we account for the uncertainty of the assignments. This help poorly initialized clusters get updated, since even if they are far away from data points initially, they will still have a responsibility over them and will therefore move towards the data. Thus this approach resolves the uneven membership assignment issue that is encountered in K-means and vanilla WTAn.

```
Algorithm 5 Hard EM for GMMs
    Step 0. Initialize parameters $\theta = \mu$.
                $\mu \leftarrow \text{init}.$
  E step. Compute ${}^{n}z^{(t)}$ for each $n$ data point.
                ${}^{n}z^{(t)} = \underset{z}{\text{argmin}} \|{}^{n}x - \mu_j^{(t-1)}\|_2$
  M-step. Update $\mu$.
                $\mu_j^{(t)} = \frac{1}{N'} \sum_{n=1}^{N'} ({}^{n}x)$
  Step 3. Repeat E and M step until $\theta$ converges.
```

3 Optimization Via Gradient Accent

The EM formulated above updates the parameters by performing MLE over N data points. Because this involves a group of data points and we would like an algorithm that updates the parameters base only on one observations at a time (e.g. online), we turn towards gradient accent. By continuously take small steps in the direction of the gradient of the lower bound of the likelihood function $Q(\theta)$, one can eventually reach a local maximum of $Q(\theta)$ [7].

In the following we show this approach for the Gaussian and constrained Hard EM.

$$\mu_j^{(t)} = \mu_j^{(t-1)} + \eta \frac{\delta Q(\mu_j^{(t-1)})}{\delta \mu_j^{(t-1)}} \tag{28}$$

$$Q(\theta) = \sum_{n=1}^{N} \sum_{j=1}^{K} \gamma_j({}^{n}x) \log w_j P({}^{n}x|z_j, \theta) \tag{29}$$

Since in the Hard EM all $\gamma_j({}^{n}x)$ are 0 except for that the parent component, the variance is spherical, and we only have one data point ($N = 1$) we have that $Q(\theta)$ is equal to:

$$Q(\theta) = \log P(x|z_j, \theta) = \log w_j \mathcal{N}(x|z_j, \mu_j, \Sigma_j) \tag{30}$$

$$= \log w_j + \log \frac{1}{\sqrt{(2\pi\sigma^2)^d}} + \log \exp(\frac{-1}{2\sigma^2} \|x - \mu_j^{(t-1)}\|_2^2) \tag{31}$$

We can further remove positive constant multipliers because they don't affect the direction of the gradient and terms that don't include $\mu_j^{(t-1)}$, because they will be 0 anyway when we take the derivative.

$$Q(\theta) = -\|x - \mu_j^{(t-1)}\|_2^2 \tag{32}$$

$$\frac{\delta Q(\mu_j^{(t-1)})}{\delta \mu_j^{(t-1)}} = 2(x - \mu_j^{(t-1)}) \tag{33}$$

We can again remove the 2 because it does not affect the direction of the gradient. Plugging in this into equation (28) we obtain the following.

$$\mu_j^{(t)} = \mu_j^{(t-1)} + \eta'(x - \mu_j^{(t-1)}) \tag{34}$$

We see how this recovers the online K-means algorithm shown in the first section (Algorithm 1). Now that we have a way of deriving an update rule we can go back to a general mixture model, decide on the parametric family and derive and EM algorithm that can also be computed with a modified WTAn. Specifically we will choose this family to be the Poisson distribution.

4 EM for Poisson Mixture Models

Previously we chose the likelihood of the data $P(x|z,\theta)$ to be Gaussian but this was simply to show the equivalence with the K-means algorithm. It might make more sense to use the Poisson since we would can think of entrees in ${}^n x_i$ vectors to represent spiking events. This makes sense since a Poisson RV represents the number of occurrences or a random event withing a specific time frame. Furthermore, with normal distributions there is always a possibility that a number drawn from such distribution is negative which would not make sense if we decide to think of the inputs as spiking events.

4.1 E-step

$$\gamma_j({}^n x) = P(z_j|{}^n x, \theta) = \frac{P({}^n x|z_j,\theta)P(z_j|\theta)}{\sum_k P({}^n x|\theta, z_k)P(z_k|\theta)} \tag{18 revisited}$$

$P({}^n x|\theta, z_j) = \prod_{i=1}^{M} \text{Pois}({}^n x_i; \lambda_{ji})$, where λ_{ji} denotes the rate parameter of component j from element i of the n^{th} input. Note that this assumes that that the input from each neuron is generated form an independent Poisson, when in reality there might be strong correlation between the activities x_i. Here we limit ourselves to the independent case but perhaps there is a more fitting way of expressing this likelihood, maybe as a multivariate Poisson [3].

$$\mathrm{Pois}({}^n x_i; \lambda_{ji}) = Pr(X =^n x_i) = \frac{\lambda_{ji}^{{}^n x_i} e^{-\lambda_{ji}}}{{}^n x_i!} \tag{35}$$

$$\gamma_j({}^n x) = \frac{w_j \prod_{i=1}^{M} \mathrm{Pois}({}^n x_i; \lambda_{ji})}{\sum_{k=1}^{K} w_k \prod_{i=1}^{M} \mathrm{Pois}({}^n x_i; \lambda_{ki})} \tag{36}$$

$$= \frac{w_j \exp\left(\sum_{i=1}^{M} {}^n x_i \log \lambda_{ji} \right)}{\sum_{k=1}^{K} w_k \exp\left(\sum_{i=1}^{M} {}^n x_i \log \lambda_{ki} \right)} \tag{37}$$

$$= \frac{w_j \prod_{i=1}^{M} \lambda_{ji}^{{}^n x_i}}{\sum_{k=1}^{K} w_k \prod_{i=1}^{M} \lambda_{ki}^{{}^n x_i}} \tag{38}$$

4.2 M-step

$$Q(\theta) = \sum_{n=1}^{N} \sum_{j=1}^{K} \gamma_j({}^n x) \log w_j P({}^n x | z_j, \theta) \tag{39}$$

$$= \sum_{n=1}^{N} \sum_{j=1}^{K} \gamma_j({}^n x) \log w_j \prod_{i=1}^{M} \frac{\lambda_{ji}^{{}^n x_i} e^{-\lambda_{ji}}}{{}^n x_i!} \tag{40}$$

$$= \sum_{n=1}^{N} \sum_{j=1}^{K} \gamma_j({}^n x) \Bigg(\log w_j + \sum_{i=1}^{M} \log \frac{\lambda_{ji}^{{}^n x_i} e^{-\lambda_{ji}}}{{}^n x_i!} \Bigg) \tag{41}$$

The derivative of every term which does not involve the λ_{ji} is 0, thus we can ignore such terms.

$$\frac{\delta Q(\theta)}{\delta \lambda_{ji}} = \frac{\delta}{\delta \lambda_{ji}} \sum_{n=1}^{N} \gamma_j({}^n x) \Bigg(\log w_j + \log \lambda_{ji}^{{}^n x_i} + \log e^{-\lambda_{ji}} - \log({}^n x_i!) \Bigg) \tag{42}$$

$$= \frac{\delta}{\delta \lambda_{ji}} \sum_{n=1}^{N} \gamma_j({}^n x) \Bigg(\log w_j + {}^n x_i \log \lambda_{ji} - \lambda_{ji} - \log({}^n x_i!) \Bigg) \tag{43}$$

$$= \sum_{n=1}^{N} \gamma_j({}^n x) \left(\frac{{}^n x_i}{\lambda_{ji}} - 1 \right) \tag{44}$$

For the batch case the closed form MLE estimate for the new λ_{ji} and w_j are as follows.

$$\lambda_{ji} = \frac{\sum_{n=1}^{N} \gamma_j (n_i^x)^n x_i}{\sum_{n=1}^{n} \gamma_j(^n x_i)} \tag{45}$$

$$w_j = \frac{\sum_{n=1}^{N} \gamma_j(^n x_i)}{N} \tag{46}$$

Furthermore we can consider the update via gradient ascent in which case we get the following expressions.

$$\lambda_{ji}^{(t)} \leftarrow \lambda_{ji}^{(t-1)} + \eta \frac{\delta Q(\theta)}{\delta \lambda_{ji}} \tag{47}$$

$$\lambda_{ji}^{(t)} \leftarrow \lambda_{ji}^{(t-1)} + \eta \gamma_j(^n x) \left(\frac{^n x_i - \lambda_{ji}^{(t-1)}}{\lambda_{ji}^{(t-1)}} \right) \tag{48}$$

If we consider the w_j's to be fixed we need only update the centroids λ_{ji}. Note that in this case we kept the constant positive multiplier $\gamma_j(^n x)$ since it provides a way to modulate the strength of the learning depending on the degree of responsibility that the mixture j over element i of data point n.

Algorithm 6 Online EM for a PMM

Step 0. Initialize parameters.

$$\Lambda \leftarrow \text{init.}$$

E step. Compute $\gamma_j(^n x)$ for each n data point and each component j.

$$\gamma_j(^n x) = \frac{w_j \exp\left(\sum_{i=1}^{M} {^n x_i} \log \lambda_{ji} \right)}{\sum_{k=1}^{K} w_k \exp\left(\sum_{i=1}^{M} {^n x_i} \log \lambda_{ki} \right)}$$

M-step. Optimize the lower bound of the log likelihood $Q(\theta)$ to update Λ.

$$\lambda_{ji}^{(t)} \leftarrow \lambda_{ji}^{(t-1)} + \eta \gamma_j(^n x) \left(\frac{^n x_i - \lambda_{ji}^{(t-1)}}{\lambda_{ji}^{(t-1)}} \right)$$

Step 3. Repeat E and M step at every input.

5 Poisson WTAn

Its not difficult to see that the Online EM for a PMM (Algorithm 6) is not too different from the online K-means (Algorithm 1). Instead of neurons in the output layer having all or non responses we have a normalized output $\gamma_j({}^n x)$ as a weighted soft-max function (37). This is a different and perhaps more accurate way of modeling the response of a population to an input, specially when considering lateral inhibition. The learning rule is again not too different, the only difference being that the change in the coefficient is the ratio of the difference and the old parameter, scaled by the normalized output $\gamma_j({}^n x)$ (48).

In sum, the proposed WTAn would consist of two step iterations upon input, and the learning would take place locally (each synapse implements their own adjustments separately), just as in the original version of the WTAn.

6 Discussion

By showing that a WTA is equivalent to an online version of the EM algorithm we also show that it shares the same theoretical assurances, namely that algorithm maximizes the lower bound of the likelihood. But perhaps even more importantly, it gives us a probabilistic framework which we can use to reason about the learning.

Specifically, the mixture weights $w_j = P(z_j|\theta)$ can be thought of as the overall probability that a random data point belongs to any given component. Here we propose that such probability can be seen as the homeostatic neuronal activity of the output neurons, e.g. the probability that a neuron will get activated, regardless of the input. While our model does not learn these parameters, we set them a priori as hyper-parameters.

Furthermore $\gamma_j({}^n x) = P(z_j|{}^n x, \theta)$ can be view as the same probability but conditional on the input or alternatively, the percentage of activity that output neuron j has relative to all of the output neurons. In a biological context it would be natural that the intensity with which the output neuron reacts to a stimuli affects the strength of the learning, that is the magnitude of the change in the synaptic weights (48).

It would also makes sense that the mixing weights w_j are taken into account for the computation $\gamma_j({}^n x)$, the relative activation of output neuron j (note that the mixing weights are only present in the computation of the $\gamma_j({}^n x)$).

Naturally this normalized activity $\gamma_j({}^n x)$ is where the lateral inhibition of the output neurons is modeled, such that the overall activity is a function of the activity of all of the neurons in the output layer. This division operation could be implemented approximately with shunting inhibition of example, where the other neurons in the output layer set the conductance such that the gain and the summation dynamics of neuron z_j are is determined by the activity of all other neurons [2]. This is due to the fact that the output voltage of a postsynaptioc neuron can be modeled as being inversely proportional to the conductance (set by lateral shunting inhibition) and directly proportional to the driving current resulting from the temporal and spatial summation of inhibitory and expiatory inputs coming from the input layer. Alternative ways of activity normalization have been proposed including synaptic scaling and feed-forward inhibition [1] [8]. It is not clear how exactly a ensemble of biological neurons would implement exactly the operation to compute $\gamma_j(^n x)$.

We further note that the proposed algorithm consists only of computing the normalized output of each neuron and then updating the synaptic weights, which as you can see from equation (48) only involves the quantities that are local to the synapse, and is thus quite Habbian like.

Perhaps a word of caution since it does not make sense to think of λ_{ji} as synaptic weights, but rather in this context they can bee seen as the guess made by neuron j for the rate parameter of the Poisson process that originates the activity from input neuron i, which gets pushed up and down depending on the how responsive output neuron j is. Perhaps there is away to show that the change in synaptic weights is repeatable to parameter λ_{ji}.

In the following section we provide the results from the simulations of the proposed algorithm. The code in this available in this gitHub repository.

7 Simulations

hola :)

References

[1] Matteo Carandini and David J. Heeger. "Normalization as a canonical neural computation". In: *Nature Reviews Neuroscience* 13.1 (2012), pp. 51–62. DOI: 10.1038/nrn3136. URL: https://doi.org/10.1038/nrn3136.

[2] Matteo Carandini and David J. Heeger. "Summation and Division by Neurons in Primate Visual Cortex". In: *Science* 264.5163 (1994), pp. 1333–1336. DOI: 10.1126/science.8191289. eprint: https://www.science.org/doi/pdf/10.1126/science.8191289. URL: https://www.science.org/doi/abs/10.1126/science.8191289.

[3] David I. Inouye et al. "A review of multivariate distributions for count data derived from the Poisson distribution". In: *WIREs Computational Statistics* 9.3 (2017), e1398. DOI: https://doi.org/10.1002/wics.1398. eprint: https://wires.onlinelibrary.wiley.com/doi/pdf/10.1002/wics.1398. URL: https://wires.onlinelibrary.wiley.com/doi/abs/10.1002/wics.1398.

[4] Christian Keck, Cristina Savin, and Jörg Lücke. "Feedforward Inhibition and Synaptic Scaling – Two Sides of the Same Coin?" In: *PLOS Computational Biology* 8.3 (Mar. 2012), pp. 1–15. DOI: 10.1371/journal.pcbi.1002432. URL: https://doi.org/10.1371/journal.pcbi.1002432.

[5] Brandon Malone. *Expectation-Maximization for Estimating Parameters for a Mixture of Poissons*. Feb. 2014.

[6] Timoleon Moraitis et al. “SoftHebb: Bayesian inference in unsupervised Hebbian soft winner-take-all networks”. In: *CoRR* abs/2107.05747 (2021). arXiv: `2107.05747`. URL: `https://arxiv.org/abs/2107.05747`.

[7] Chris Piech. *Gradient Ascent*. Nov. 2018.

[8] Frédéric Pouille et al. “Input normalization by global feedforward inhibition expands cortical dynamic range”. In: *Nature Neuroscience* 12.12 (2009), pp. 1577–1585. DOI: `10.1038/nn.2441`. URL: `https://doi.org/10.1038/nn.2441`.

[9] Wikipedia. *Mixture model — Wikipedia, The Free Encyclopedia*. `http://en.wikipedia.org/w/index.php?title=Mixture%20model&oldid=1082352648`. [Online; accessed 28-June-2022]. 2022.