Modeling Neural Populations with Mixture Models

Santiago M. Castro Dau

Semester project for the Cortical Computation Group,
Institute of Neuroinformatics,
University of Zürich and ETH Zürich

June 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 [7] and [5] 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 denotes the index of the input, having in total N of these input vectors in one data set (Figure 1).

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 (Figure 1).

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 (Figure 1).

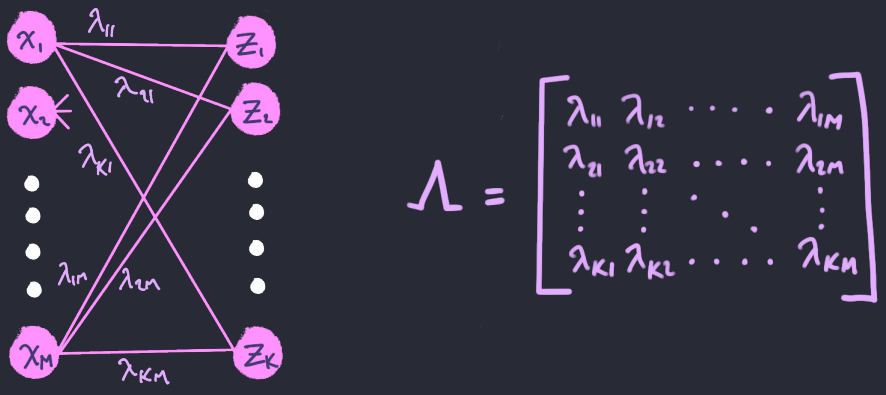


Figure 1: Schematic representation of the WTAn described in this section.

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 [5].

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 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 get 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.

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]

$$\sum_{i=1}^{M} \lambda_{ji} = \alpha \tag{1}$$

$$\sum_{i=1}^{M} x_i = \alpha \tag{2}$$

$$\lambda_{j:}^{(t)} \leftarrow \lambda_{j:}^{(t-1)} - \eta\epsilon'(x - \lambda_{j:}^{(t-1)}) \tag{3}$$

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 $\epsilon'(x - \lambda_{j:}^{(t-1)})$ (4) - (8).

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

[2] For more information on the biological plausibility and of normalization in biological neural circuits we refer the reader to [5].

$$\epsilon(x - \lambda_{j:}^{(t-1)}) = \frac{1}{2}||x - \lambda_{j:}^{(t-1)}||_2^2 \tag{4}$$

$$\frac{d}{d\lambda_{j:}^{(t-1)}}\epsilon(x - \lambda_{j:}^{(t-1)}) = \frac{d}{d\lambda_{j:}^{(t-1)}}\frac{1}{2}||x - \lambda_{j:}^{(t-1)}||_2^2 \tag{5}$$

$$= -(x - \lambda_{j:}^{(t-1)}) \tag{6}$$

$$= \epsilon'(x - \lambda_{j:}) \tag{7}$$

$$\lambda_{j:}^{(t)} \leftarrow \lambda_{j:}^{(t-1)} + \eta(x - \lambda_{j:}^{(t-1)}) \tag{8}$$

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).

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 closest 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

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 schemes.

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

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

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

Algorithm 2 K-means

Step 0. Initialize parameters for each centroid.

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

Step 1. For all N data points find the closest 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 centroids 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 (Figure 2).

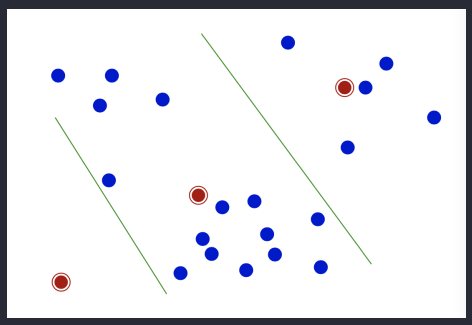


Figure 2: Schematic representation uneven membership assignment.

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 [10].

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 observation 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 subpopulations 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 the 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 ${}^{n}x \in \mathbb{R}^{M}$ 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 ji^{th} entree of Λ.
- ${}^{n}z \in \mathbb{R}^{K}$ 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 data point x came from the distribution of component j, e.g. the probability density function of the j^{th} component.
- ${}^{(t)}$ 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.

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

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

There are 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 [10]. In this work we concern ourselves with the problem of parameter estimation, since we assume that the number of components and their underlying distributions are fixed. Furthermore we take Expectation Maximization (EM) algorithm as or method of choice for inference. [3]

In the context of mixture models, the EM can be seen as a 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 induce specific input pattern distributions 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) followed by a maximization step (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.

$$\gamma_j({}^n x) = P(z_j|\,{}^n x,\theta) \tag{14}$$

$$= \frac{P(z_j,{}^n x,\theta)}{P({}^n x,\theta)} \tag{15}$$

$$= \frac{P({}^n x|z_j,\theta)P(z_j|\theta)}{P({}^n x|\theta)} \tag{16}$$

$$P({}^n x|\theta) = \sum_k P({}^n x|\theta,z_k)P(z_k|\theta) \tag{17}$$

[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.

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.

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

Once we have all the γ's for each data point, we have a fully annotated data set, and 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 (12) 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.

$$\log \mathcal{L}(D : \theta) \geq b(\theta) \tag{19}$$

$$\log \mathcal{L}(D : \theta) = \sum_{n=1}^{N} \log \sum_{j=1}^{K} w_j P({}^n x|z_j, \theta) \tag{20}$$

$$b(\theta) = \sum_{n=1}^{N} \sum_{j=1}^{K} \gamma_j({}^n x) \log \frac{w_j P({}^n x|z_j, \theta)}{\gamma_j({}^n x)} \tag{21}$$

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 parameters θ we perform the partial derivatives with respect to each individual parameter (e.g. λ_{ji} and w_j), make it equal to 0, and

[4] For a more detailed derivation we refer the reader to [6].

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 choose uniform 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 θ.

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

Step 3. Repeat E and M-step until θ 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 EM for Gaussian Mixture Models (GMMs). We start by deriving the unconstrained version of the EM which yields 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 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).

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

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

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.

we make $\gamma_j({}^n x) = 1$ for all but the winning 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.

These modifications already yield the Hard EM for GMMs but one can also derive it as follows to show its equivalence to K-means. As stated above we seek the cluster that maximizes $P(z_j | {}^n x, \theta)$ in the E-step.

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

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 therefore 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.

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

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.

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

We see the E-step becomes Step 1 of K-means (Algorithm 2). Similarly by performing MLE on the lower bound of the likelihood with these constraints one obtains Step 2 of the K-means algorithm for the M-step (derivation not shown).

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.

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

M-step. Update μ.

$$ \mu_j^{(t)} = \frac{1}{N'} \sum_{n=1}^{N'} ({}^{n}x) $$

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

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 clusters to have different sizes and elliptical shapes, which allows 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 helps poorly initialized clusters to 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. However performance will still be highly dependent on initialization.

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 based only on one observation at a time (e.g. online), we turn towards gradient accent. By continuously taking 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 [8].

In the following derivation we show this approach for a constrained GMM (the Hard EM setting).

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

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

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 in the next step.

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

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

We can again remove the 2 because it does not affect the direction of the gradient. Plugging this into the 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 online update rule we can go back to a general mixture model, decide on a different parametric family and derive an EM algorithm that can also be computed with a 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 sense to use the Poisson if we can think of entrees ${}^n x_i$ as the number of spiking events. This makes sense since a Poisson RV represents the number of occurrences or a random event within 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 the i^{th} element 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, perhaps as a multivariate Poisson [4].

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

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

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

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

4.2 M-step

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

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

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

The derivative of every term which does not involve λ_{ji} is 0, thus we can ignore such terms (43).

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

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

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

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

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

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

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 data point ${}^n x$.

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.

It's 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 none 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 (compare (48) and (8)). In sum, we propose a modified version of the WTAn with the operations described in algorithm 4 [6].

5 Discussion

By showing that a WTAn 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. Biological mechanisms for activity normalization have been proposed and studied [2] [1] [9], however it is not clear how or if an ensemble of biological neurons could implement the operation to compute $\gamma_j({}^n x)$ described in equation (37).

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

It is important to clarify that it does not make sense to think of λ_{ji} as synaptic weights, but rather as the rate at which neuron i fires at neuron j, where each

[6] The pseudo-codes are missing the normalization of the inputs, which is rather a processioning step, and the normalization of the λ_{ji} w.r.t. $\lambda_{j:}$ after each M-step.

neuron j can have a different guess for the rate λ_{ji} [7]. When both λ_{ji} and x_i are big, the more neuron j will get activated, and when there is a discrepancy between these two, the λ_{ji} will be pushed up or down to match the input.

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

6 Simulations

6.1 Simulations specifications

We tested our algorithm on the MNIST data-set [3]. We adjusted the weights through 30 epochs of the training split and tested the classification performance on the testing split. Since every image is 28 by 28 pixels the input dimension (number of nodes in the input layer) is 784 and we chose the output dimension to be 50 in order to capture heterogeneity across digits.

Each training and testing image was normalized, and scaled in compliance with equations (1) and (2). After scaling, we summed 1 to every pixel to avoid numerical problems. We chose the previously mentioned scaling factor $\alpha = 784$, but since we added 1 to every pixel the overall intensity in every image came out to be 1568, which was the normalization factor we used for the $\lambda_{j\cdot}$'s after each M-step.

The λ_{ji} were initialized according to the mean pixel value across the processed training split plus its standard deviation weighted by numbers drawn from a uniform random distribution. Finally the images were flattened to a 1-dimensional tensor to facilitate matrix operations.

The simulations were run in two scenarios, uniform fixed mixing weights and log-normal. The sections below show the results for the uniform mixing weight scenario and we discuss the impact of changing the mixing weights in a subsequent subsection.

For more implementation details please see the simulations code.

6.2 Evolution of the λ_{ji}

In figure 3 we can see the evolution of the $\lambda_{j\cdot}$ in heatmap form after each of the 30 epochs, starting from the top left, going to the right before going down to the next row.

The first thing that catches the eye is that individual neurons display a generative field that resembles a digit. This is a nice result as it indicates that each

[7] Although perhaps there is away to show that the synaptic weights is somehow connected to parameter λ_{ji}.

Figure 3: Heatmaps of the generative field of the first neuron after each of the 30 epochs.

neuron is the steward of a generative field corresponding to one of the 10 classes. These results are in agreement with the results in [5].

The second one is that perhaps 30 epochs is an overkill since it seems that already after the first one the $\lambda_{j:}$ have settled down into one of the digits. In the next figure you can see the generative fields for each one of the 50 neurons after 30 epochs of training.

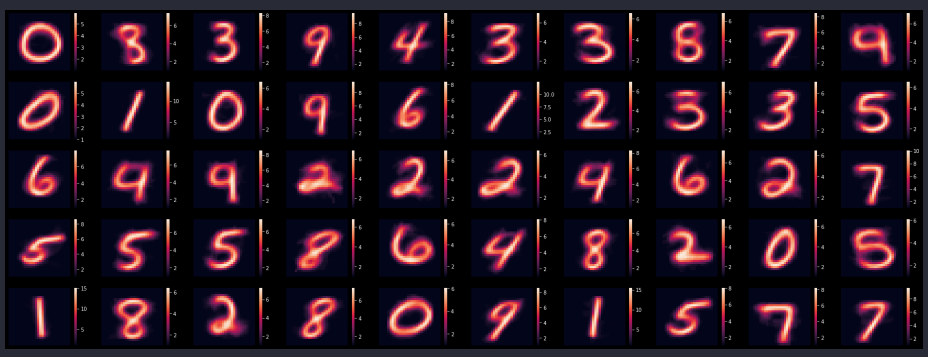


Figure 4: Heatmaps of the generative field of all neurons after 30 epochs.

We can see that there are multiple neurons representing one single digit, which perhaps helps capture heterogeneity in the typography. For the following classification performance analysis we manually annotated each one of the neurons according to the digit they display in their generative field.

6.3 Classification Performance

The next figure shows the performance of the trained algorithm in predicting the correct class of the testing images. For each image we computed the $\gamma_j({}^n x)$ vector, and counted as on-probability the value of the $\gamma_j({}^n x)$'s that corresponded to the label of the input, and off-probability the $\gamma_j({}^n x)$ of all the other neurons. We summed these on and off probabilities for each neuron for the whole testing

split. Then we aggregated all of the on-probability mass for neurons representing the same digit and the same for the off mass. These quantities are then represented in the bar chart below.

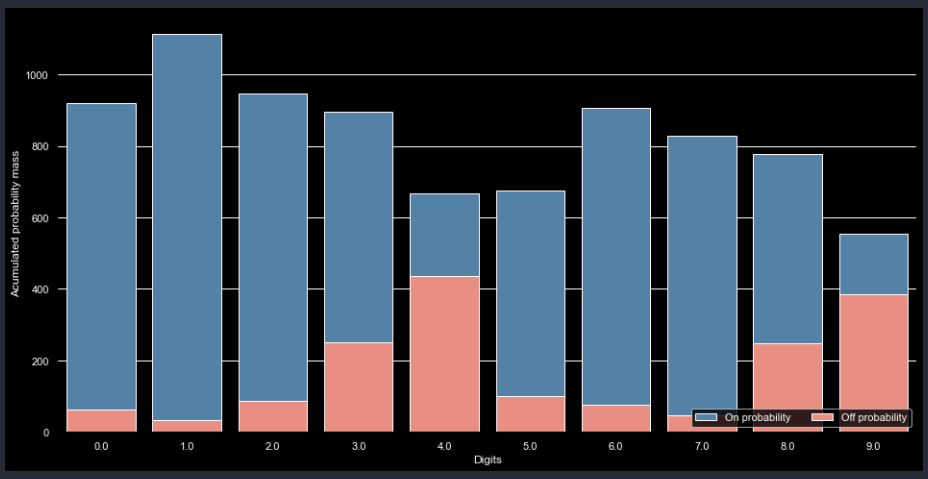
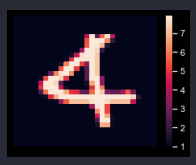


Figure 5: Off and on probability of predictions for every image on the testing split.

We can see that there is a significant amount of off-probability specially for digits 4 and 9 which are often very similar. In fact when you look at the number of images that are classified correctly according to the maximally excited neuron, only around 80% of the test images are classified correctly.

To further investigate why the performance is not better even when the generative fields show neat digits we displayed examples where the input was not classified correctly along with the top 10 maximal excited neurons (Figure 6). It seems that because the $\gamma_j({}^n x)$ only cares about overlap between the generative field and the input there can be miss classifications when the input overlaps more with a different digit.

```
Label of input:  4
Heatmap of input image:
```



```
Probability of image belonging to the first 10 components:
```

```
[1.0000e+00, 8.4539e-23, 1.7772e-23, 3.1410e-32, 1.1038e-33,
3.9651e-46, 3.0196e-49, 4.7079e-61, 4.6908e-64, 5.2946e-66]

Top 10 generative fields:
```

Figure 6: Display of code showing the top 10 generative fields for a miss classified image.

We can also note that the probabilities for all subsequent neurons are usually quite low with the maximal $\gamma_j({}^n x)$ often approximating 1. This might be a numerical issue that needs further investigation. Ideally we would like our network to faithfully represent uncertainty in the input, spreading more evenly the probability mass along the output neurons, specially the ones with similar generative fields.

6.4 Effect of the Mixing Weights on Classification Performance

The performance is very similar, perhaps slightly lower but would need further investigation [8]. Likewise, the evolution of the weights and the resulting generative fields are very similar to the ones with the simulations where the mixing weights were uniform (images not shown).

6.5 Future Perspectives

In relation to the simulations and the theoretical framework presented before we present a list of further improvements to this work.

- The weights need not be kept constant, they could be updated just as the λ_{ji}'s. We hypothesize that specially when there is a large class imbalance, updating the mixing weights would have a strong impact on performance.
- We can view the mixing weights as representing the prior on the class size; perhaps the algorithm learns to compensate for class imbalance by grouping together output neurons that share similar generative fields. However, it is not clear whether this is actually happening in our simulations.
- As mentioned before the uncertainty in the prediction is not very well represented. Achieving an accurate representation would be a great way to improve performance.

[8] There is more variation in the performance between different runs of the simulations than variation between the uniform weights and log normal weights simulation.

- It would be interesting to explore whether one could perform more complex downstream tasks other than classification using this framework. One example of such a task would be the addition of two numbers, where the input layer contains digits that come from two distinct underlying distributions and the output layer somehow represents the result of the sum.
- It would also be interesting to explore the implications of assuming a mixture model to begin with or alternative parametric families.
- Finally the independence assumption for the derivation of the E-step in the Poisson EM could also have a large impact on performance, but also on model complexity.

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] Li Deng. "The mnist database of handwritten digit images for machine learning research". In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 141–142.

[4] 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.

[5] 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.

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

[7] 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.

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

[9] 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.

[10] 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.