Competitive Mixtures of Simple Neurons

Karthik Sridharan Matthew J. Beal Venu Govindaraju

{ks236,mbeal,govind}@cse.buffalo.edu
Department of Computer Science and Engineering
State University of New York at Buffalo
Buffalo, NY 14260-2000, USA

Abstract

We propose a competitive finite mixture of neurons (or perceptrons) for solving binary classification problems. Our classifier includes a prior for the weights between different neurons such that it prefers mixture models made up from neurons having classification boundaries as orthogonal to each other as possible. We derive an EM algorithm for learning the mixing proportions and weights of each neuron, consisting of an exact E step and a partial M step, and show that our model covers the regions of high posterior probability in weight space and tends to reduce overfitting. We demonstrate the way in which our mixture classifier works using a toy 2-dimensional data set, showing the effective use of strategically positioned components in the mixture. We further compare its performance against SVMs and one-hidden-layer neural networks on four real-world data sets from the UCI repository, and show that even a relatively small number of neurons with appopriate competitive priors can achieve superior classification accuracies on held-out test data.

1 Introduction

One of the main challenges in the problem of classification is to find a hypothesis that does not overfit the training data; there is always a trade-off between training and test accuracies. High training accuracy is often indicative of an overfitted classifier that may lead to poor generalization performance on test data. An effective way to alleviate the problem of overfitting is to combine the classification results of several classifiers. A Bayes-optimal rule for combining the results of several classifiers is to take a linear weighted sum of their predictions, weighting by the posterior probability of each classifier having generated the training data set. Due to the fact that even for moderate dimensionalities of parameter space it is difficult to sample uniformly over the parameters' posterior distribution, this weighted prediction is often approximated using a very large set of bagged or bootstrap-trained classifiers [1]. An alternative to such bagged estimates is to use Markov chain Monte Carlo methods, in which a trajectory is simulated through the parameter space that converges to a stationary distribution that is the true posterior distribution [8]. However, both these methods are computationally intensive.

We propose a solution where we use much fewer finite combinations of simple neurons that each form simple linear boundaries, and combine them in a mixture model formalism to model various parts of the data. Indeed such a mixture model has been used before for classification [5]. However, in constrast to standard mixture model approaches, we introduce a penalty into the cost function of each neuron in the form of a squared cosine term between the weights of other neurons, such that the model learns solutions that are not only good in terms of the classification performance, but are also encouraged to be different from each other. Our algorithm can be thought of as a mixture of experts model with competitive penalties between the experts (as shown in Figure 1). We use an EM algorithm [2] for learning the weights of each of the neurons and the mixing proportions between them. Since we treat the loss function of a neuron as a negative log probability, we automatically have a probabilistic interpretation and hence can evaluate posterior responsibilities of neurons for data given priors on each of them.

2 Preliminaries

Consider a binary classification problem with feature space $\mathcal{X}$ and binary target space $\mathcal{T} = \{0, 1\}$. Given n training samples $\{\mathbf{x}_1, ..., \mathbf{x}_n\}$ from space $\mathcal{X}$ with binary labels $\{t_1, ..., t_n\} \in \mathcal{T}$, our task is to find a function $f : \mathcal{X} \to \mathcal{T}$ mapping a given input to a binary target classification. If we consider a single neuron having a logistic output function, the function $f(\mathbf{x}; \mathbf{w}) = (1 + e^{-\mathbf{w}^\top \mathbf{x} + b})^{-1}$ is thresholded to obtain the classification, where $\mathbf{w}$ is the set of weights for the neuron. One possible general error function that can be minimized during the training of the neuron is

$$E_\alpha(\mathbf{w}) = \frac{\alpha}{2} \sum_{i=1}^{n} (t_i - f(\mathbf{x}_i; \mathbf{w}))^2 \, .$$

If we assume that this loss function is the negative log probability of the data, then by this interpretation we are in fact

modeling each data point i the random variable $t_i - f(\mathbf{x}_i; \mathbf{w})$ as a Gaussian distribution with mean zero and precision α.

3 Mixture of Neurons

Consider k neurons with weights given by $\mathbf{w}_1, ..., \mathbf{w}_k$. Note that without loss of generality we can model a bias into the neuron by appending a bias weight and augmenting the inputs with an extra dimension that is fixed to a value of 1. Each neuron in the mixture attempts to reduce its total squared prediction error over all the data, but we also add a term in the cost function that explicitly pushes each neuron's parameters away from the remaining neurons. Thus our final classifier is the expected output of a set of classifiers, each trained to classify well, but at the same time as different from each other as possible. This overall cost function for the classifier modeling the ith data point is given by

$$E_i = \frac{\alpha}{2} \sum_{j=1}^{k} (t_i - f(\mathbf{x}_i; \mathbf{w}_j))^2 + \frac{\beta}{2} \sum_{j=1}^{k} \sum_{l=1, l \neq j}^{k} \cos^2(\mathbf{w}_j, \mathbf{w}_l) \tag{1}$$

where the squared cosine term is employed to penalized similar weights in different neurons. This term is important to break the symmetry between the k classifiers, since without it the weights of all the neurons would converge to the same value (modulo undersirable local minima), which is clearly an uninteresting predictive ensemble. By minimizing the inner product, in weight space we are effectively making the neuron weights as orthogonal to each other as possible thus covering the posterior more thoroughly. In data space, since the angle between any two weight vectors is the angle between the corresponding separating hyperplanes, reducing the squared cosine term causes a splaying of the separating hyperplanes to the widest possible angles whilst also striving for good classification capability. If we consider this cost function as the negative log probability of the datum, not only are we trying to model the prediction error with a Gaussian distribution with precision α and mean 0, but we are also including a prior stating that we believe that the cosine of the weight of each neuron with respect to other neurons is also from a Gaussian distribution with mean 0 (orthogonal) and precision β.

Now let $\Theta_w = \{\mathbf{w}_1, \ldots, \mathbf{w}_k\}$ denote the setting of the collection of weights for the k neurons. Let the hidden indicator variable h_i taking on value j denote the ith datum being modeled by the jth neuron with probability π_j, then the probability of observing datum $(\mathbf{x}_i, t_i)$ given the parameters and indicator $h_i = j$ is $P(\mathbf{x}_i, t_i, \Theta_w | h_i = j) \propto e^{-\frac{\alpha}{2}(t_i - f(\mathbf{x}_i; \mathbf{w}_j))^2 + \frac{\beta}{2} \sum_{l=1, l \neq j}^{k} \cos^2(w_j, w_l)} = e^{-E_{ij}}$.

3.1 EM Algorithm

We now derive in brief an EM algorithm [2] to estimate the parameters of our model. The EM algorithm we consider uses a batch update of weights, i.e. all the instances are considered together and the parameters of all the neurons are updated simultaneously. The probability of observing n i.i.d. training samples $\{\mathbf{x}_i, t_i\}_{i=1}^n$ given the weights Θ_w and mixing proportions $\boldsymbol{\pi}$ is given by

$$P(\{\mathbf{x}_i, t_i\}_{i=1}^n | \Theta_w, \boldsymbol{\pi}) = \prod_{i=1}^{n} \sum_{j=1}^{k} P(h_i = j | \boldsymbol{\pi}) P(\mathbf{x}_i, t_i | h_i = j, \Theta_w)$$

where each h_i is some hidden variable distributed according to the prior mixing proportions $\boldsymbol{\pi}$. Taking the logarithm of this probability, and introducing a set of variational distributions $Q = \{Q_i(h_i)\}_{i=1}^n$ over the hidden indicators for each of the data points, using Jensen's inequality we obtain a lower bound on the likelihood of the parameters Θ_w for the model, which we denote $\mathcal{F}(\Theta_w, \{Q_i(h_i)\})$

$$\begin{aligned} \mathcal{L}(\Theta_w) &\equiv \log P(\mathbf{x}_1, ..., \mathbf{x}_n, t_1, ..., t_n | \Theta_w) \\ &\geq \sum_{i=1}^{n} \sum_{h_i=1}^{k} Q_i(h_i) \log \left(\frac{P(h_i) P(\mathbf{x}_i, t_i | h_i, \Theta_w)}{Q_i(h_i)} \right) \\ &\equiv \mathcal{F}(Q, \Theta_w) \, . \end{aligned} \tag{2}$$

Therefore we have $\mathcal{L}(\Theta_w) \geq \mathcal{F}(Q, \Theta_w)$, and since we cannot directly optimize $\mathcal{L}(\Theta_w)$ we instead optimize $\mathcal{F}(Q, \Theta_w)$ and this guarantees that the negative log-likelihood of the data always decreases. The EM algorithm alternately optimizes the distributions $\{Q_i\}$ and the weights $\Theta_w = \{\mathbf{w}_1, \ldots, \mathbf{w}_k\}$ keeping one constant while optimizing the other, thus performing a co-ordinate ascent.

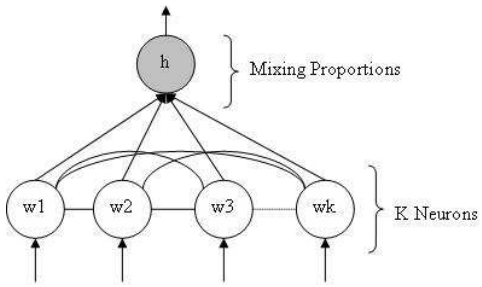


Figure 1. Model of the competitive classifier.

E step: In the E step we optimize $\mathcal{F}(Q, \Theta_w)$ with respect to each $Q_i(h_i)$. Since $Q_i(\cdot)$ is a probability distribution we use a Lagrage multiplier λ_i to enforce normalization. Taking derivatives of $\mathcal{F}$ with respect to each $Q_i(h_i)$ yields

$$\frac{\partial \mathcal{F}(Q, \Theta_w)}{\partial Q_i(h_i)} = \log P(\mathbf{x}_i, t_i, h_i | \Theta_w) - \log Q_i(h_i) + 1 - \lambda_i \, ,$$

which upon setting to zero, finding the extremum, and solving for λ_i, yields

$$Q_i(h_i = j) = \frac{\pi_j \, P(\mathbf{x}_i, t_i | h_i = j, \Theta_w)}{\sum_{l=1}^{k} \pi_l \, P(\mathbf{x}_i, t_i | h_i = l, \Theta_w)} \, , \tag{3}$$

where $\pi_j = P(h_i = j | \Theta_w)$ is the prior mixing proportion associated with neuron j.

M Step: From Equation (2) we have that

$$\mathcal{F}(Q,\Theta_w) = \sum_{i=1}^{n}\sum_{j=1}^{k} Q_i(j)\log\left(\frac{\pi_j P(\mathbf{x}_i,t_i|h_i=j,\Theta_w)}{Q_i(j)}\right)$$

In the M step we maximize $\mathcal{F}(Q,\Theta_w)$ with respect to each of the weights $\mathbf{w}_j$ and the priors π_j. To find the optimal prior $\boldsymbol{\pi}$ we differentiate with respect to each of its elements π_j while enforcing normalization with a Lagrange multiplier λ', resulting in $\sum_{i=1}^{n}\frac{Q_i(j)}{\pi_j} + \lambda' = 0 \Rightarrow \pi_j = -\frac{\sum_{i=1}^{n} Q_i(j)}{\lambda'}$. From normalization, $\lambda' = -\sum_{j=1}^{k}\sum_{i=1}^{n} Q_i(j) = -n$, so to maximize $\mathcal{F}(Q_i,\Theta_w)$ we set

$$\pi_j = \frac{\sum_{i=1}^{n} Q_i(j)}{n} \tag{4}$$

Now to maximize $\mathcal{F}(Q_i,\Theta_w)$ with respect to each of the weights $\mathbf{w}_j$ we use gradient ascent. The gradient of $\mathcal{F}(Q,\Theta_w)$ w.r.t. each of the weights $\mathbf{w}_j$ is given by

$$\begin{aligned}\frac{\partial\mathcal{F}(Q,\Theta_w)}{\partial w_j} &= \sum_{i=1}^{n} Q_i(j)\frac{\partial\log P(\mathbf{x}_i,t_i|h_i=j,\Theta_w))}{\partial\mathbf{w}_j}\\ &= -\sum_{i=1}^{n} Q_i(j)\frac{\partial E_{i,j}}{\partial\mathbf{w}_j} + c'\ ,\end{aligned}$$

where c' is some constant. With learning rate η, the jth neuron is updated following the negative gradient:

$$\begin{aligned}\Delta\mathbf{w}_j = \eta\sum_{i=1}^{n} Q_i(j)(t_i - f(\mathbf{x}_i;\mathbf{w}_j))(1-f(\mathbf{x}_i;\mathbf{w}_j))f(\mathbf{x}_i;\mathbf{w}_j)\mathbf{x}_i\\ -\eta\beta\sum_{l\neq j}\frac{(\mathbf{w}_j)^\top\mathbf{w}_l}{|\mathbf{w}_j||\mathbf{w}_l|}\frac{\mathbf{w}_l - \mathbf{w}_l\frac{\mathbf{w}_j\mathbf{w}_l^\top}{|\mathbf{w}_j|^2}}{|\mathbf{w}_j||\mathbf{w}_l|}\ .\end{aligned} \tag{5}$$

we follow the negative gradient, the M step is partial.

4 Performance Evaluation

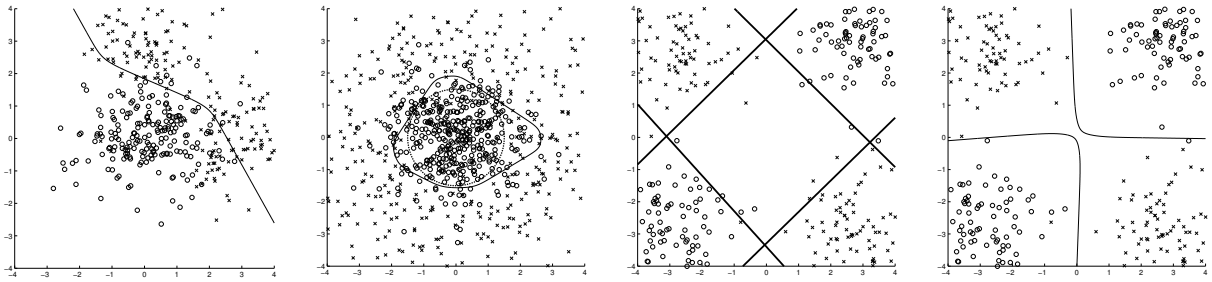
We demonstrate the working of our proposed method on some 2-d toy examples. Figure 2(a) shows the classification of 2-d data in which class 1 is a Gaussian with mean zero and unit spherical covariance and class 2 is made of 2 Gaussians with unit spherical covariances and means $(2,0)$ and $(0,2)$. This data set is successfully classified using mixture of just two neurons. Figure 2(b) shows a binary classification problem where class 1 consists of points drawn from a Gaussian of mean 0 and unit spherical covariance and is surrounded in a circular fashion by class 2 points. The Bayes boundary is shown as a circle of radius 1.5, as well as the boundary obtained using our proposed approach with just 4 neurons; we have effectively used the 4 neurons to form a closed boundary resembling the required boundary.

Lastly, one of the compelling reasons to design neural networks with hidden layers is to be able to classify the XOR function. We show that we can achieve this with a mixture of 4 neurons. Figure 2(c) shows the hyperplanes laid down by the 4 neurons and Figure 2(d) shows the decision boundary formed by this mixture of neurons. We see that the simple mixture of neurons is able to discern even this nonlinear boundary well.

To test the performance of the proposed mixture of neurons, we used four 2-class real datasets from the UCI Machine Learning repository [3]: PIMA, WDBC, ION and BUPA (see Table 1 for dimensionalities and numbers of instances). In all the experiments we used randomly selected 60% of the data for training and 40% for testing. Table 1 summarizes the accuracies of the proposed method against Gaussian-kernel SVMs, polynomial-kernel SVMs and a simple backpropagation neural network. The results are averaged over 20 trials and in each trial the same training and testing data was given to all four classifiers. First, we see that our method with competition between classifiers ($\beta > 0$) always beats a simple combination of classifiers ($\beta = 0$). We also see that the proposed method outperforms neural networks in all cases, and significantly beats polynomial- and Gaussian-kernel SVMs in (a different) 3 out of the 4 data sets. Further, we note the stability of our method as evidenced in its consistently low standard error. Finally we note that all algorithms had their hyperparameters individually tuned to report their best possible results: in our proposed method α and β parameters were set to 0.03 and 0.01 respectively. According to (1), α and β trade off the costs of classification and orthogonality of classifier boundaries, respectively, and this ratio of $3:1$ was found to be optimal. For the experiments above, a mixture of 16 neurons was found to be effective (through cross-validation).

5 Related Work

In [6] a hierarchical mixture of experts is proposed, in which a probabilistic formulation of the experts is trained using an EM algorithm. A precursor to the work in [6] is a model described in [5], which is most similar to our proposed method. The authors even suggest the idea of using of a cost function that introduces competition between the classifiers, but do not elaborate on this or perform such experiments. In our approach we do indeed find that our cosine squared cost function does help in creating a competition between the classifiers. We have also determined that there is an intriguing link between our proposed classifier and the margin-based classifiers such as that proposed in [4]. This link becomes clear when we use the proposed approach for a linearly separable classification problem where, apart from the mean squared error term, we use the cosine squared term to update only the bias term of the weight of the classifiers. In this case all the neurons have approximately the same orientation and the cosine squared term for the bias pushes the neurons as far



(a) (b) (c) (d)

Figure 2. Demonstrative synthetic examples (see text for key)

Table 1. Comparison of proposed model with state-of-the-art approaches

Dataset	dim.	instances	*Back-prop*	*SVM (Gaussian)*	*SVM (polynomial)*	*Mix. of Neurons* **no** *competition*	*Mix. of Neurons* **with** *competition*
PIMA	8	768	64.36 ± 0.45	69.12 ± 0.58	75.50 ± 0.39	76.74 ± 0.47	77.69 ± 0.18
WDBC	39	569	96.67 ± 0.15	97.04 ± 0.15	93.42 ± 0.36	97.24 ± 0.21	97.68 ± 0.17
ION	33	353	86.86 ± 0.69	87.93 ± 2.48	85.00 ± 0.67	87.00 ± 0.33	87.79 ± 0.78
BUPA	6	345	62.61 ± 0.82	67.25 ± 0.77	71.01 ± 0.70	70.22 ± 0.52	70.72 ± 0.72

Note: errors quoted are the standard errors of the mean.

away from each other as possible, while classifying the data correctly. Therefore the extremal hyperplanes (neurons) arrange themselves in a way so as to increase the gap (linear margin) between them. When all the terms of the weights of the classifier are updated according to the cosine squared function, the weight terms tend to be as different from each other as possible yet classifying the data accurately. Thus the model has similar behavior to mixture of SVM classifiers like that proposed in [7]. Therefore, an extension to our proposed method is to directly formulate the method as a mixture of margin classifiers and use quadratic programming to find the optimal parameter setting.

6 Conclusion

We have presented a probabilistic mixture of neurons for binary classification that can tackle the problem of overfitting by combining results of neurons that are as orthogonal to each other as possible yet each strives to do well on classification. We have shown the effective performance of the proposed method on both synthetic and real data, and have shown for the most part superior performance over neural networks and two types of SVMs. One phenomenon we observe while performing classification of 2-d data is that occasionally some neurons end up putting hyperplanes at unlikely locations where no data is found just to cancel out the effects of other neurons' errors. This is mainly because the decision of a single neuron is uniformly weighed all across the hyperplane. Hence certain neurons seem to be sacrificed to compensate for errors far away from actual data. One way to counter this problem is to have a distribution to weigh the classification across the hyperplane such that near the data the hyperplanes are weighted more and away from the data their weight is less thus decreasing their confidence about classification away from data points.

References

[1] L. Breiman. Bagging predictors. *Mach. Learn.*, 24(2):123–140, 1996.

[2] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–38, 1977.

[3] C. B. D.J. Newman, S. Hettich and C. Merz. UCI repository of machine learning databases, 1998.

[4] Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *ICML*, pages 148–156, 1996.

[5] R. A. Jacobs, M. I. Jordan, S. Nowlan, and G. Hinton. Adaptive mixture of local experts. *Neural Comp.*, 3(1):79–87, 1991.

[6] M. I. Jordan and R. A. Jacobs. Hierarchical mixtures of experts and the EM algorithm. *Neural Comp.*, 6(2):181–214, 1994.

[7] J. T.-Y. Kwok. Support vector mixture for classification and regression problems. In *ICPR*, volume 14, page 255, Washington, DC, USA, 1998. IEEE Computer Society.

[8] R. M. Neal. *Bayesian Learning for Neural Networks*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1996.