Collectives of local decision makers
Construction Of A Boundary Hunter

Daniel Braithwaite

Supervisor: Marcus Frean

June 29, 2017

Abstract

Training a Multi Layer Perceptron (MLP) Network to solve supervised learning problems is common practice. When a problem is composed of subtasks then using an MLP Network has poor learning. The paper Adaptive Mixtures of Local Experts [1] proposes a Mixture of Experts model to solve this which is used in practice. This report speculates that decoupling the expert networks will further improve learning. The model developed in this paper allows experts to be dynamically trained and added as needed, it is able to perform well in low dimensions but encounters problems with learning as the number grows.

Chapter 1

Introduction

Single or Multilayer Perceptron Networks are commonly trained to model and solve a supervised learning problem. Attempting to train these networks to perform different tasks on separate occasions, using a training algorithm such as Backpropergation will result in slow convergence and poor generalization. **Find Citation, Many papers say this but don't provide reference**.

If it is known prior to training that our problem is the composition of sub tasks, then a more suitable system should be used to avoid the previously discussed issues. In the paper Adaptive Mixtures of Local Experts [1] an idea was proposed to create systems consisting of expert networks, each responsible for learning a subset of the data. A gate, another neural network, takes examples and outputs the responsibility for each of the experts. Such a system is called a Mixtures of Experts (MoE) model.

During training of a MoE model the gating network is learning how it can best assign examples to the various experts. If a case is presented to the model which it gets wrong then the weight changes are localized to the gating network and the networks to which it assigned the example. Therefore experts which operate in different situations (regions of the data) do not directly interfere with each others weights, removing some dependency between the expert networks.

Consider that for a given example the responsibility of network j is given by $p_j = \frac{e_j^x}{\sum_i e_i^x}$. Equation 1.1 is the loss function used, which is a linear combination of the outputs from each expert.

$$E^c = -log \sum_i p_i^c e^{-\frac{1}{2}\|d^c - o_i^c\|^2} \tag{1.1}$$

Equation 1.2 shows the change in error with respect to the output of some expert network.

$$\frac{\partial E^c}{\partial o_i^c} = \left[\frac{p_i^c e^{-\frac{1}{2}\|d^c - o_i^c\|^2}}{\sum_j p_j^c e^{-\frac{1}{2}\|d^c - o_j^c\|^2}}\right] \cdot (d^c - o_i^c) \tag{1.2}$$

The term $\left[\frac{p_i^c e^{-\frac{1}{2}\|d^c - o_i^c\|^2}}{\sum_j p_j^c e^{-\frac{1}{2}\|d^c - o_j^c\|^2}}\right]$ is a Softmax function on the output of our expert networks. Consequently we see that $\frac{\partial E^c}{\partial o_i^c}$ takes into account network i's performance against all others. Demonstrating that there still exists dependency between the experts.

We speculate that further reducing and eliminating the dependency between expert networks will improve learning. To motivate this problems with the MoE model are identified and it is explained how reducing dependency could solve them.

Currently if using an MoE model the number of experts must be decided beforehand. Requiring either prior knowledge about the number of subtasks in the problem or a trial and error process to find the optimal amount. If the experts where completely independent then we could build our system sequentially by adding each one until we achieve our desired accuracy.

Consider a MoE model with only one expert. This will just act like a standard Perceptron Network, given the gate will have no choice but to assign all examples to the single expert, demonstrated by $p_1 = \frac{e^{x_1}}{\sum_i e^{x_i}} = \frac{e^{x_1}}{e^{x_1}} = 1$. Figure 1.1 is a concrete example of how this results in a poor classifier which does not truly model the data.

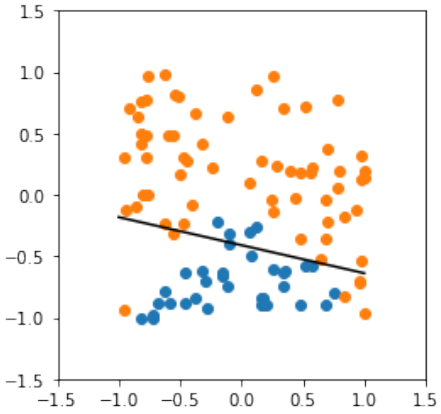

Figure 1.1: Example data for situation with line representing a trained perceptron

Shown in Fig 1.1, a single Perceptron (an MoE model with only one expert) trained on this data does not produce anything meaningful. The classifier achieves a low error but the decision boundary has not captured any structure of the data. If training a MoE model with no dependency between experts, then individually they should learn which parts of the data they are responsible for regardless of whether there are any other experts networks, consequently this would mean giving each one its own gate. The goal would be to train experts which identify the local features of the data, in Fig 1.1 these would be both sides of the chevron. With no communication between experts could end up with some learning the same information. Perhaps the minimal amount of communication between experts needed is that they stay out of each others way.

In the current MoE model the dependency between experts is a consequence of two things, the gating network and that the error is a combination of the output from all experts. To remove this dependency each expert should decide what it is responsible for, thus making it possible to train them separately using an error function which only considers their individual outputs.

To investigate this speculation, we developed a model based on the previous observations. Experts are trained independently and identify local features in the data. Because of this experts can be added or removed from the data as needed. These independent experts will be called boundary hunters

This paper reports on the development of our model and describes the path we followed to arrive at our final solution.

Chapter 2

Neuron Parametrisation

A perceptron is learning a decision boundary described by a hyperplane, defined by vector normal $w = [w_1, ..., w_n]$ to the plane and a bias b.

Definition 2.0.1. A Perceptron with vector $w = [w_1, ..., w_n]$ normal to the decision boundary and bias b has weighted sum for input $x = [x_1, ..., x_n]$ defined as

$$z = \sum_{i=1}^{n} w_i x_i$$

and output $a(z) = f(z)$
where $f(z) = 1$ indicates that x is of class 1 and $f(x) = 0$ class 0.

To define the boundary hunter it is necessary to be able to specify a region of interest, as discussed before they must be able to decide what examples they are interested in. A logical way to define this region is to give the perceptron a centre and specify a radius around it, anything inside this area it will care about. A perceptron's parameters wont allow this, so the parametrisation must be adapted.

2.1 Normal & Point Parametrisation

An equivalent definition for a perception can be formulated defined by a centre point and a hyperplane passing though it.

Definition 2.1.1. The **Normal & Point** parametrisation of a perceptron consists of the vector $w = [w_n, ..., w_n]$ normal to the hyperplane and $m = [m_n, ..., m_n]$ the point our hyperplane passes though.

On input $x = [x_1, ..., x_k]$ the weighted sum is defined as

$$z = \sum_{i=1}^{n} w_i \cdot (m_i - x_i)$$

and output $a(z) = f(z)$
where $f(z) = 1$ indicates that x is of class 1 and $f(x) = 0$ class 0.

Theorem 1. *The normal & point representation for a perceptron is equivalent to our standard definition*

Proof. Assume that both parametrisations are using the same activation, then it is enough to show that the weighted sums would be the same. Let $w = [w_1, ..., w_n], m = [m_1, ..., m_n]$ be the parameters for a normal and point perceptron.

$$\begin{aligned} z &= \sum_{i=1}^{n} w_i \cdot (m_i - x_i) \\ &= w \cdot [m_1 - x_1, ..., m_n - x_n] \\ &= w_1 \cdot (m_1 - x_1) + ... + w_n \cdot (m_k - x_k) \\ &= w_1 m_1 - w_1 x_1 .... w_n m_k - w_n x_n \\ &= (w_1 m_1 + ... + w_n m_n) - w_1 x_1 - ... - w_n x_n \end{aligned}$$

Now an equivalent standard perceptron can be defined, with bias term $b = w_1 m_1 + ... + w_n m_n$ and each component of $m^{'}$ as $m_i^{'} = -m_i$. □

Comparison To Perceptron

Figures 2.1 & 2.2 compare the two parametrisation experimentally to confirm the proof presented above, showing that both will converge to the same optimal solution. The centre of the normal and point perceptron is represented by the green dot in the graph.

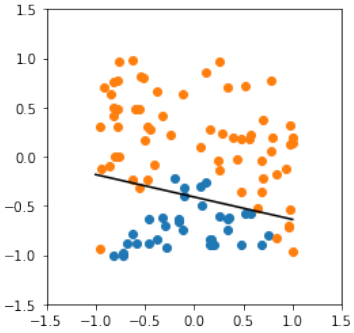

Figure 2.1: Standard Perceptron (SSE = 3.90)

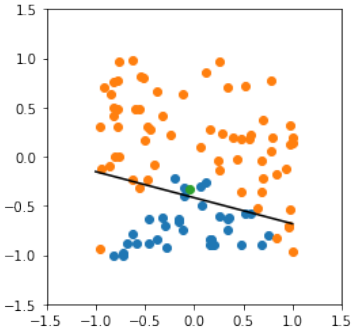

Figure 2.2: Modified Perceptron (SSE = 3.90)

Chapter 3

Boundary Hunter: First Attempt

3.1 Responsibility Function

The **Normal & Point Parametrization** has been shown to be equivalent to the regular Perceptron, it can now be used as a base to develop a boundary hunter. Before a suitable loss function can be designed the concept of responsibility must be further developed. Using the centre point plus a radius how can the boundary hunters responsibility for an example be quantified? Let r, $R(r, i)$ be denoted the radius and responsibility for example i.

A list of properties for R follows.

1. $R(r, i) \in [0, 1]$. This keeps the values constrained, a responsibility of 0 or 1 means we ether care or not retrospectively.
2. $R(r, i)$ must continuous and differentiable so our gradients work with Back-propagation
3. Any point inside or outside the area of interest should have responsibility close to 1 or 0 retrospectively.
4. Responsibility for points outside the area of interest should decrease as distance from area increases, this guarantees that as we move further away from the edge of our radius we get less interested

A logical choice for R is the logistic function, $f = 1 - \frac{1}{1+e^{-S(x-x_0)}}$ where S is the steepness. f meets requirements (1) - (4). The steepness can be thought of this as how rapidly our interest changes at the border of our region, a steep function means we rapidly move from caring to ignoring an example, where as a more gradual slope means it takes longer for us to loose interest.

The sigmoid function guarantees that any example will have $R(r,i) > \frac{1}{2}$, $R(r,i) < \frac{1}{2}$, $R(r,i) = \frac{1}{2}$ if it is inside, outside or on the border retrospectively. As $S \to \infty$ then the responsibility approaches a step function, being 1 for any example inside the boundary hunters region and 0 otherwise. If the steepness is to low then examples far away from the boundary hunter will have to much of an impact, on the other hand if its to high then the boundary hunter wont consider any example its not interested in. From inspecting the graphs for various steepnesses $S = 10$ is a goood place to start, the responsibility decays rappidly but slow enough to consider a range of points in/out side the area of interest.

$$R(r,i) = 1 - \frac{1}{1 + e^{-10(d_i - r)}} \tag{3.1}$$

where d_i is the distance example i is from the boundary hunters centre.

3.2 Loss Function Design: Attempt 1

For example i let the output of a boundary hunter be denoted by $\widehat{t_i}$. $\widehat{t_i}$ is the probability that example i has a class of 1, i.e. $p_{t_i=1} = \widehat{t_i}$ and $p_{t_i=0} = 1 - \widehat{t_i}$. If the true class is 1 then probability of being correct p_c is $p_{t_i=1}$, likewise if the true class is 0 then $p_c = p_{t_i=0}$. The boundary hunter should have a concept of certainty. If the given example has a high responsibility (i.e. is close to the boundary hunters centre) then the hunter should be certain its answer is correct, likewise a low responsibility means the example is far from the boundary hunters centre and therefore it should be uncertain about the correctness of its answer.

As uncertainty about the boundary hunters output increases t_i (probability of the examples true class being 1) approaches $\frac{1}{2}$, a fair coin toss, i.e. $R(r,i) \to 0$ then $\widehat{t_i} \to \frac{1}{2}$.

If the boundary hunter has a high responsibility for an example then the error is its accuracy at predicting the correct class, using the cross entropy this is quantified as $t_i log(\widehat{t_i}) + (t_i) log(1 - \widehat{t_i})$. If the responsibility for an example is low then its penalty for that instance is the error for predicting a fair coin toss, also using the cross entropy results with $log(\frac{1}{2})$. Equation 3.2 is the resulting loss function

$$L = -\frac{1}{N} \sum_{n=0}^{N} R(r,n)(t_n log(\widehat{t_n}) + (1 - t_n) log(1 - \widehat{t_n})) + (1 - R(r,n)) log(1/2) \tag{3.2}$$

3.2.1 Experimental Results

Figure 3.1 shows a boundary hunter trained with equations 3.1 & 3.2 for the loss and responsibility functions.

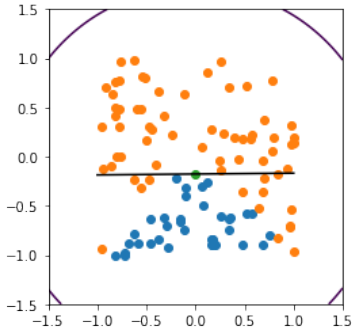

Figure 3.1: Boundary Hunter with 3.2 as loss and 3.1 as responsibility

The boundary hunter has learnt to care about all the data which is the opposite of the goal.

3.2.2 Discussion

Figure 3.1 shows the boundary hunter has a high responsibility for every example, reducing 3.2 to the standard cross entropy loss

$$L = -\frac{1}{N}\sum_{n=0}^{N} R(r,n)(t_n log(\widehat{t_n}) + (1-t_n) log(1-\widehat{t_n}))$$

The solution which the boundary hunter converged on allows miss classifications, indicating that the penalty for taking a guess and getting it wrong is to low. Ideally the boundary hunter would have a perfect classification accuracy and choose to ignore any data which compromises this.

3.3 Loss Function Design: Important Observation

Let l_i be the standard cross entropy loss for example i. Consider a simplified version of equation 3.2

$$L = \sum R(r,i) * l_i$$

When computing the gradients to update the free parameters, the partial derivative $\frac{\partial L}{\partial r}$ gives the change in loss with respect to the radius.

$$\begin{aligned}\frac{\partial L}{\partial r} &= \frac{\partial}{\partial r}\big[\sum R(r,i)\cdot l_i\big] \\ &= \sum R^{'}(r)\cdot l_i\end{aligned}$$

The quantity $R^{'}(r)$ is direction in which to move r with the goal of maximizing the responsibility function. l_i is always the same sign, consequently observe the quantity $\frac{\partial L}{\partial r}$, it does not represent how changing the radius changes the loss.

$$\frac{\partial L}{\partial r} = (l_1 + ... + l_n)\cdot R^{'}(r)$$

This gradient will move r in the direction which increases the responsibility function, a larger radius gives a larger responsibility, resulting in the radius constantly increasing.

This observation shows that to find a loss function to train boundary hunters simply scaling an existing loss with the responsibility is not a plausible solution.

3.4 Loss Function Design: Attempt 2

Consider the boundary hunter as a salesperson, who gets paid $\$B$ for selling something to someone who wants it (if the salesperson is responsible for this individual) and gets penalized $\$C$ dollars for selling something to someone who does not want it. Using this model, the salesperson is to position them selves and adjust their responsibility so that they are maximizing there profit.

$y^t(1-y)^{t-1}$, $y^{t-1}(1-y)^t$ is the probability the boundary hunter outputs a correct or wrong classification retrospectively, consequently giving equation 3.3 as a candidate loss function for a boundary hunter.

$$L = \sum_{i=0} R(r,i) * \big[Cy_i^{1-t_i}(1-y_i)^{t_i} - By_i^{t_i}(1-y_i)^{1-t_i}\big] \tag{3.3}$$

The differential $\frac{\partial L}{\partial r}$, $Cy_i^{1-t_i}(1-y_i)^{t_i} - By_i^{t_i}(1-y_i)^{1-t_i}$ can be both positive and negative, avoiding the issues discussed in section 3.3. If the boundary hunter is making a loss then L is negative, consequently $\frac{\partial L}{\partial r} = -\alpha\cdot R^{'}(r)$ thus decreasing the radius. If the boundary hunter is making a profit then $\frac{\partial L}{\partial r} = \alpha\cdot R^{'}(r)$ thus increasing the radius.

3.4.1 Experimental Results

Figure 3.2 shows the result of training a boundary hunter with equations 3.3 & 3.1 as the loss and responsibility functions. With a similar solution to using equation 3.2 as the loss function the boundary hunter includes every example in its area of interest.

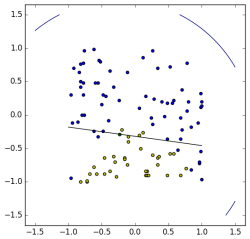

Figure 3.2: Boundary Hunter with (3.5) as loss and (3.2) as responsibility

The cost of miss classifying a few examples is outweighed by the reward of getting all other examples correct. This demonstrates that the parameter C (cost for incorrect classification) is to low. It is unclear the extent to which increasing C will change the optimal solutions, to get a good spread 1.3, 1.6, 1.9 are used as values for C

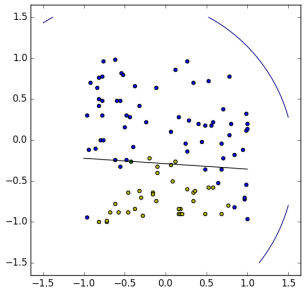

Figure 3.3: C = 1.3

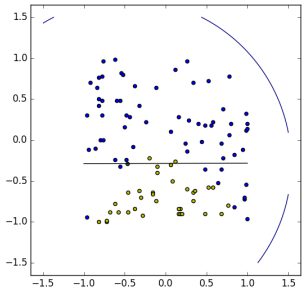

Figure 3.4: C = 1.6

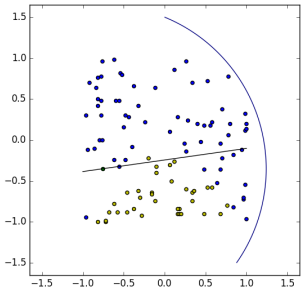

Figure 3.5: C = 1.9

Figures 3.3, 3.4 & 3.5 show that this only changes the hyperplane positioning, it does not stop the boundary hunter from being interested in all examples. If some reasonable restrictions are imposed on the radius (i.e. $0.3 \leq r \leq 0.8$), set C = 1.9 and adjust the responsibility function to be less steep (steepness of 5) then some better results are achieved, which are more in line with the goal.

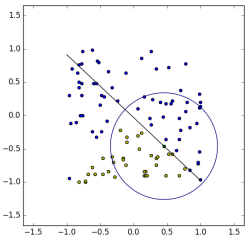

Figure 3.6

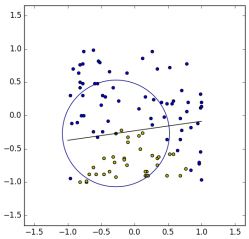

Figure 3.7

More often than not the boundary hunter converges on a solution like the one shown in figure 3.8, these do not achieve anything meaningful.

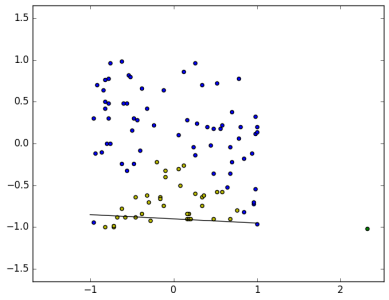

Figure 3.8

The solutions shown in figures 3.6 & 3.7 by inspection have lower error than figure 3.8, consequently the issue we are observing must be that of local optima.

The local optima become visible in figures 3.9, 3.10 & 3.11. These plots show the loss for each possible centre position of the boundary hunter. The hyperplane has been fixed so that it aligns with the left side of the chevron. The responsibility has a steepness of 5 and C = 1.9.

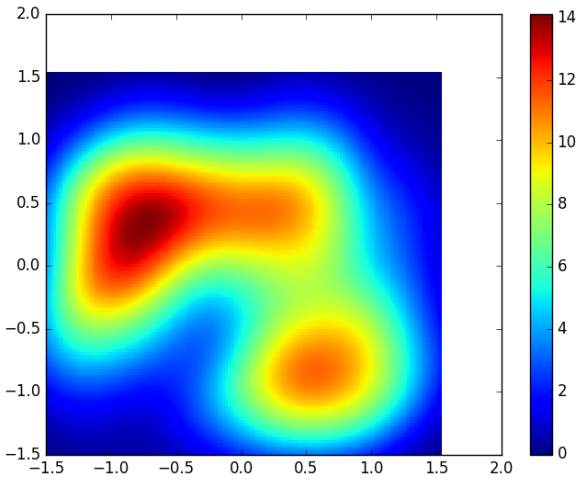

Figure 3.9: Radius as 0.3

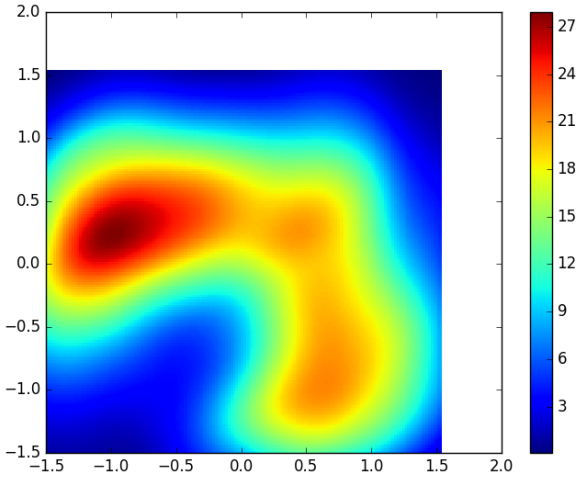

Figure 3.10: Radius as 0.8

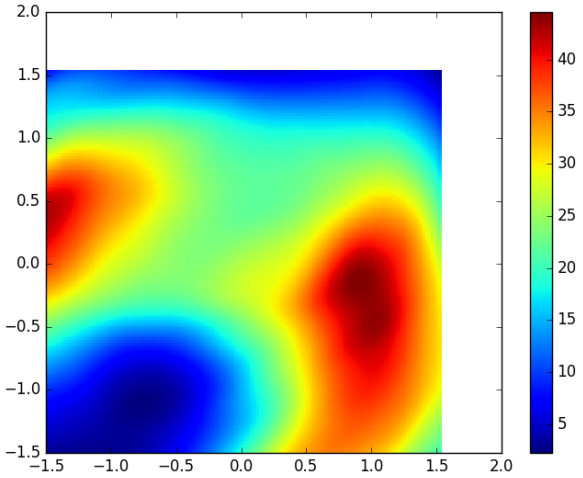

Figure 3.11: Radius as 2.0

Figures 3.9, 3.10 & 3.11 show that while there is an optima aligned with the left side of the chevron there are more optima at the edges of the examples, corresponding to figure 3.8

Outside the boundary hunter's area of interest the responsibility for any example rapidly approaches 0 as the distance from the centre increases. As

the boundary hunter moves away from all examples thus excluding all from its interest then error approaches 0. This seems to indicate that the responsibility function is to steep because the boundary hunter does not care enough about examples outside its interest.

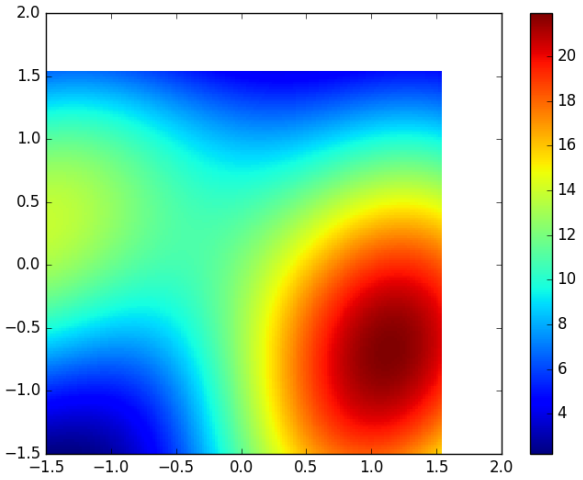

Figure 3.12: Radius as 0.3, with steepness as 0.5

Figure 3.12 demonstrates that even with a less steep responsibility function the same local optima issues exists.

3.5 Discussion

Based on these findings this reports concludes that trying to incorporate the responsibility into a loss function is, in essence a careful balancing act of rewards and penalty's for which there is possibly no solution. Consequently a new approach is needed.

Chapter 4

Radial Basis Function Networks

Radial Basis Function (RBF) Networks seem closely related to a boundary hunter. Before this connection can be made a brief overview of RBF Networks will be given by first defining an RBF Function.

Definition 4.0.1. If f is an **Radial Basis Function** then its output depends only on the distance from its input to some point c, called the centre of f.

$$f(x, c) = g(\|x - c\|) \tag{4.1}$$

Definition 4.0.2. A **Radial Bias Function Neuron** has activation a which is an RBF with centre c. On input x the output of an RBF Neuron is $a(x, c)$

Definition 4.0.3. A **Radial Bias Function Network** consists of three layers, input, hidden and output. The hidden layer consists of Radial Bias Function Neurons. There are no weights between the input and hidden layers, the example given to the RBF Network is passed directly to the RBF Neurons. The output layer consists of standard perceptrons.

Each RBF Neuron is measuring the similarity between its centre and the input vector presented to it, the closer the input is to the centre the closer the activation is to 1. Any RBF is sutible to be an activation, a commonly used one is based off the 1D Gussian

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Which can be simplified to give

$$a(x) = e^{-\beta \|x-c\|^2} \tag{4.2}$$

We use the β term to control how quickly the activation decays.

An intuitive view of an RBF Network is a number of RBF Neurons measure how similar a given input vector is to their centre, which gets reported to the output neurons. Each output neuron then use this information to make a decision about their activation by taking a weighted sum of the outputs from the RBF Neurons.

4.1 Training RBF Networks

The simplest method for training RBF Networks is to randomly initialize all parameters in the network and use gradient descent to optimize them.

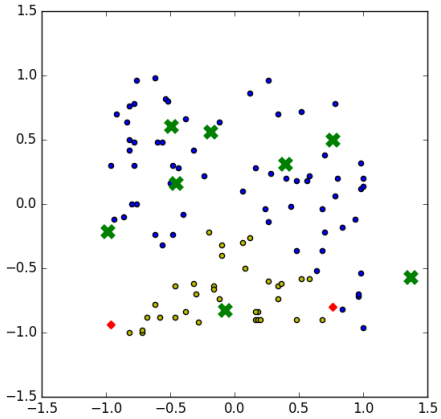

Figure 4.1

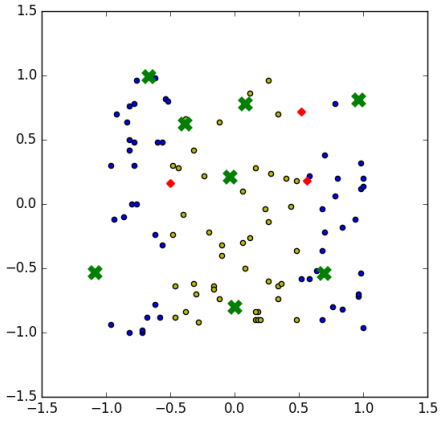

Figure 4.2

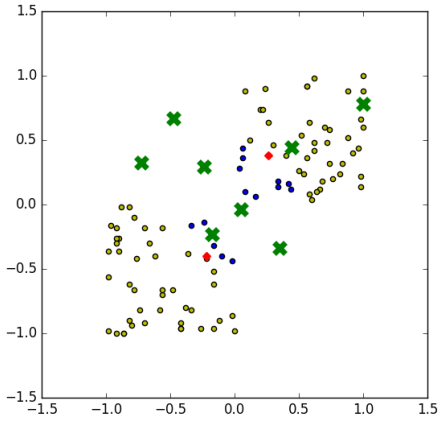

Figure 4.3

The green x's and red x's represent our centroids and the points we are getting incorrect retrospectively

Figures 4.1, 4.2 and 4.3 show RBF Networks trained over three different data sets. In each case the overall classifier has achieved a good model for the data, only getting a few examples wrong in each case.

4.2 Discussion

RBF Neurons do not make any decisions about the data, they only report on the similarity between the input vector and their centre. Consider the activation of an RBF Neuron as the certainty that the neuron is responsible about an example, a closer example means a higher certainty about responsibility. Now the output of an RBF Network can be thought of as a decision based off number of certainties about a given examples position.

This is what a boundary hunter should be doing, except instead of reporting the certainty of responsibility it will output its certainty of the examples class.

Chapter 5

RBF Based Boundary Hunters

Based on these RBF Networks, is it possible to construct a boundary hunter? In this chapter an activation function for an RBF Boundary Hunter Neuron is derived.

Consider that the output of an RBF Neuron is the uncertainty about whether it is responsible for the given example, as the point gets further away the neuron becomes less certain about our ownership and so the activation decreases in value. In the case of an RBF Boundary Hunter its activation reflects the neurons certainty of the examples class, as the distance from the boundary hunters centre increases then the neuron will be less certain of its classification.

Let f be the activation function for an RBF Boundary Hunter. As the belief that the example has a class of 1 increases $f \to 1$ and as the belief decreases $f \to 0$, so if $f = \frac{1}{2}$ then the neuron is completely uncertain about the classification. Consequently as the distance between the boundary hunter's centre and an example increases $f \to \frac{1}{2}$. The definition of an RBF Boundary Hunter can now be given.

Definition 5.0.1. A **RBF Boundary Hunter Neuron** (for binary classification) in $\mathbb{R}^n$ has $2n + 1$ free parameters. $\beta \in \mathbb{R}$, $\mathbf{c} \in \mathbb{R}^n$ and $\mathbf{n} \in \mathbb{R}^n$. $\beta, \mathbf{c}$ represent decay speed and centre of the RBF portion of our neuron. $\mathbf{n}$ defines a hyperplane which passes through our point $\mathbf{c}$. We define the activation as follows

$$a(x) = \frac{1}{2} + e^{-\beta\|x-c\|^2}\left(\frac{1}{1 + e^{-(\mathbf{n}\cdot(\mathbf{c}-x))}} - \frac{1}{2}\right) \tag{5.1}$$

5.0.1 Training RBF Boundary Hunters

Using the same method as before, randomly initializing all parameters and then training using gradient descent.

In figures 5.1, 5.2 and 5.3 green x's represent the centroids while the red x's the incorrect classifications.

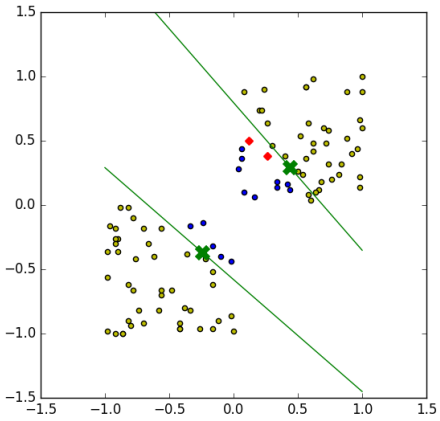

Figure 5.1

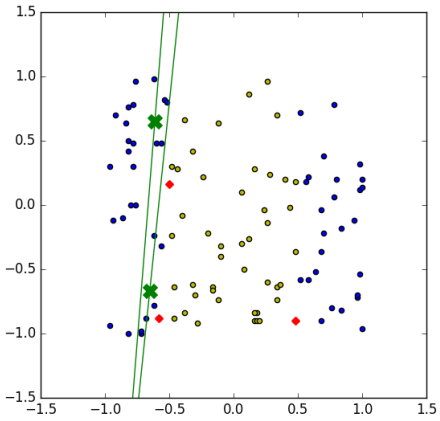

Figure 5.2

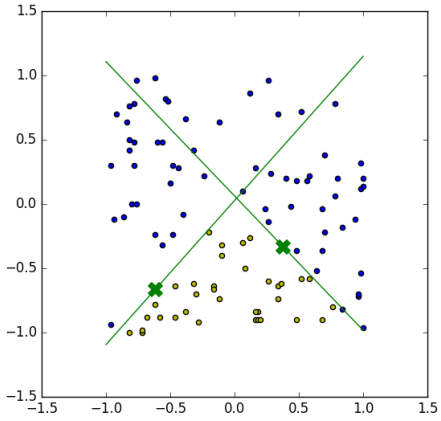

Figure 5.3

The hyperplanes are certainly being placed on boundaries in the data. Returning to the motivation for this project, what happens if a single RBF Boundary Hunter neuron is trained on the chevron data set ?

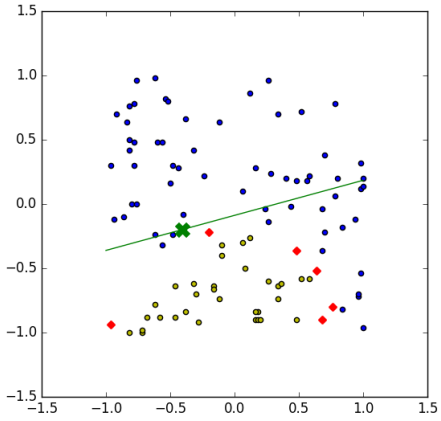

Figure 5.4

Figure 5.4 demonstrates that these RBF Boundary Hunter Networks still suffer from dependency between the neurons. If there was no dependency between the boundary hunters then the hyperplane in figure 5.4 would be one of the planes shown in 5.3, seeing as this is not the case then the two boundary hunters must be dependent on each other.

5.1 Gated Neurons

Moving away from the idea of RFB Neurons and generalizing the concept of a RBF Boundary Hunter Neuron to what will be called a **Gated Neuron**. This will allow different methods to be defined for computing the responsibility.

Definition 5.1.1. A **Gated Neuron** consists of two functions, a **gating function** g and prediction function f. g takes the neurons input and outputs our certainty about the classification for that point, if g is 0 then the neuron is completely unsure about the input and if g is 1 then it is certain. The Gated Neuron activation function is defined below

$$a(x) = \sigma(g(x) \cdot f(x)) \tag{5.2}$$

Note: Equation 5.2 is not equivalent to our previous activation 5.1. The gating operation has been moved inside the sigmoid function as this is simpler.

Then a Gated Neuron (GN) Network is a feed-foward network where the hidden neurons are Gated Neurons.

5.2 Hyperplane Gated Neuron

A hyperplane gated neuron can now be defined, the neuron has a second hyperplane which is used to divide the space into two parts, the ones its responsible for and the ones it is not. In a similar fashion to before this definition will include a centre point.

Definition 5.2.1. A **Hyperplane Gated Neuron** is a gated neuron with the following parameters and functions. A centre point c and two hyperplanes n and m passing through c. n is normal to the decision boundary and m is normal to a hyperplane which divides the plane into parts, the do and don't care sections. Therefore giving $g = \sigma(m \cdot (c - x))$ and $f = n \cdot (c - x)$. The resulting activation function is

$$a(x) = \sigma(\sigma(m \cdot (c - x)) \cdot (n \cdot (c - x))) \tag{5.3}$$

Which when in a GN Network and trained on the chevron data gives the following result

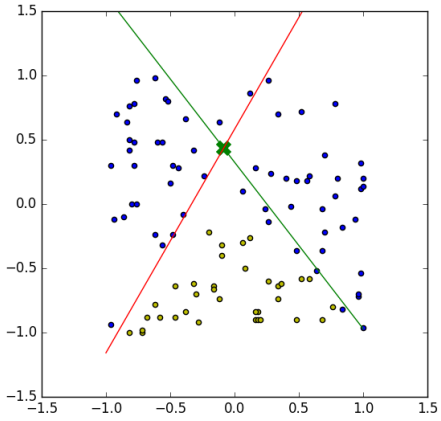

Figure 5.5
The red line represents the gating hyperplane and the green line represents the decision boundary

This is precisely what we are looking for in terms of classifying a chevron with only one boundary hunter.

5.2.1 Comparison To Perceptron

These boundary hunters have $3n$ free parameters, a little less than three times the number required for a regular perceptron, so a reasonable question to ask is what can be achieved with a three layer perceptron network?

Is it possible to achieve better performance with a standard Multi Layer Perceptron (MLP) Network containing 3 hidden neurons. This network configuration has been chosen as it contains a similar number of free parameters to the Hyperplane GN Network.

This comparison will be performed on the chevron dataset, using 10-Fold Cross Validation and computing a 95% confidence interval on the accuracies.

Network	Avg Training Error	Avg Test Error
Perceptron	0.00749882335194	0.0172678373009
Gated Neuron	0.0244635285134	0.0354204954447

Network	Training Error Conf Interval	Avg Test Error Conf Interval
Perceptron	(0.007, 0.008)	(0.007, 0.028)
Gated Neuron	(0.014, 0.035)	(0.012, 0.059)

These 95% confidence intervals show that there is a statistically significant difference between the training but not the test accuracies, so it can be concluded that the two methods have the same performance.

5.2.2 Redundancy

The definition of a Hyperplane Gated Neuron included a centre point, which is redundant. The point is specifying the intersection of our two lines which this is not needed. Figure 5.6 demonstrates that this point can be removed and the same results can be achieved but with fewer free parameters, only 2n + 2 to be precise.

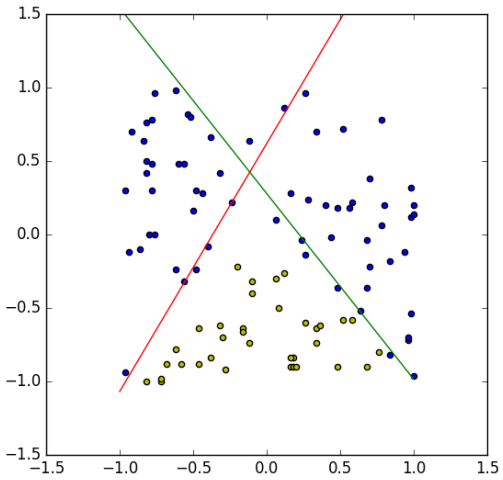

Figure 5.6
Hyperplane Gated Boundary Hunter without the centre point

Comparing the performance of these new hyperplane gated boundary hunters against a standard perceptron network with 2 hidden neurons gives the following.

Network	Avg Training Error	Avg Test Error
Perceptron	0.023	0.037
Gated Neuron	0.023	0.031

Network	Training Error Conf Interval	Avg Test Error Conf Interval
Perceptron	(0.010, 0.037)	(0.013, 0.061)
Gated Neuron	(0.017, 0.030)	(0.008, 0.053)

With this modification there is still no statistically significant difference between the performance of the two networks.

5.2.3 Discussion

Does this current solution solve the problem laid out at the beginning of this report. How is this solution different to the one proposed in Adaptive Mixtures of Experts [1]. The goal was to create neurons which act independently and learn to classify local features of the data.

Compared to the MoE model it would definitely appear that Hyperplane GN Networks have less dependency between the hidden neurons, each GN is responsible for deciding its interest which is not the case in the MoE model.

One thing to consider however is that a Hperlplane GN Network has an output layer, taking the classification from each of the boundary hunters and making the final decision about the examples class. The output layer reintroduces some dependency between the hidden neurons, demonstrated by figures 5.7 & 5.8.

In figure 5.7, the single boundary is unable to identify any sides of the rectangle boundary.

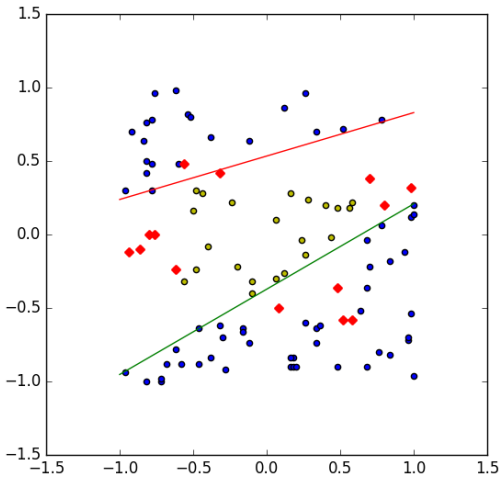

Figure 5.7
Rectangle data trained with a single boundary hunter

Figure 5.8 shows a Hyperplane GN Network with two boundary hunters trained on the same data. The hyperplanes align them selves with the rectangle boundary, showing that there is collaboration between the boundary hunters. If there was no cooperation then figure 5.7 would be equivalent to figure 5.8 with a boundary hunter removed.

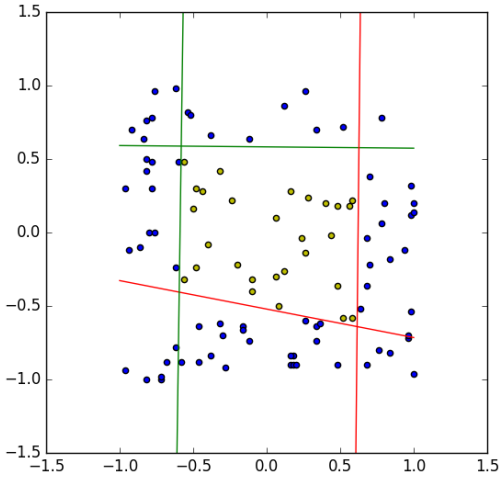

Figure 5.8

Rectangle data trained with two boundary hunter

Training these Hyperplane GN's individually would have no dependency between the boundary hunters but how will this effect the performance?

Chapter 6

Individual Hyperplane Gated Neurons

Can hyperplane gated neurons be trained individually and achieve similar results to when they are trained in a network. Consider the loss when using the cross entropy error function.

$$L = -\sum_{n=0}^{N} t_n log(\widehat{t_n}) + (1 - t_n) log(1 - \widehat{t_n})$$

For a specific example i, without loss of generality say $t_n = 1$, then $l_i = log(\widehat{t_n}) = log(\sigma(\sigma(m \cdot x) \cdot (n \cdot x)))$ which can be simplified by substituting $r = \sigma(m \cdot x)$, resulting with $l_i = log(\sigma(r \cdot (n \cdot x)))$. As $r \to 0$, $l_i \to log(\frac{1}{2})$, so L is equivalent to equation 3.2.

Consequently it is not surprising to see that training this set-up has the same results as section 3.2, figure 6.1 demonstrates this.

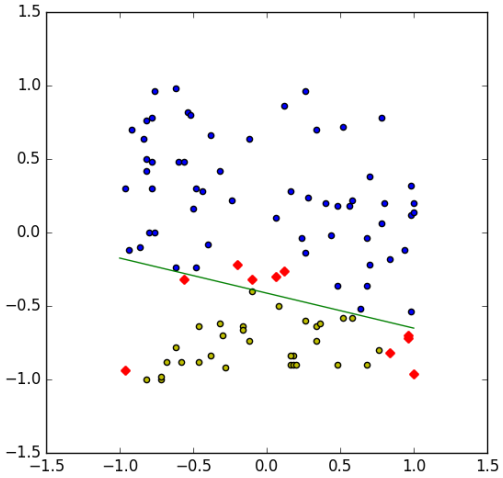

Figure 6.1
Chevron data trained with a lone boundary hunter

6.1 Biases

As the responsibility for example i approaches 0 $\widehat{t_i} \to \frac{1}{2}$. $\widehat{t_i}$ represents the probability that example i has a class of 1, when $\widehat{t_i} = \frac{1}{2}$ then it means that it is equally likely the true class to be 1 or 0. It is more likely that the probability of drawing an example with class 1 is not $\frac{1}{2}$.

Say probability of drawing a class 1 example which the boundary hunter does not care about is p_1^c. The loss over all examples which the boundary hunter is not responsible for is

$$\begin{aligned} L_c &= -(n_1 log(p_1) + n_0 log(1-p_1)) \\ &= -(log(p_1^{n_1}) + log((1-p_1)^{n_0})) \\ &= -(log(p_1^{n_1}(1-p_1)^{n_0})) \end{aligned}$$

where n_a is the number of do not care examples of class a. Regardless of n_0, n_1 $p_1^{n_1}(1-p_1)^{n_0}$ has maximum value at $p_1 = \frac{1}{2}$ consequently L_c also attains maximum value.

$$-log(p_1^{n_1}(1-p_1)^{n_0}) \leq -log(\frac{1}{2}^{n_1}\frac{1}{2}^{n_0}) \tag{6.1}$$

Equation 6.1 demonstrates that using the true distribution of the do not care data will give a lower loss. Consequently it makes more sense for a boundary hunter to learn the value which they output in the case they don't care. This parameter will be called bias β.

Definition 6.1.1. A **Hyperplane Gated Neuron with Bias** has the same parameters as the same parameters as a Hyperplane GN with an added bias β. Similarly to before $g = \sigma(m \cdot (c - x))$ and $f = n \cdot (c - x)$. Equation 6.2 defines the activation of this neuron.

$$a(x) = \sigma(g(x) \cdot f(x) + (1 - g(x)) \cdot \beta) \tag{6.2}$$

Figure 6.2 shows a Hyperplane GN with Bias trained on the chevron data, the result is the same that was achieved with a single neuron Hyperplane GN Network.

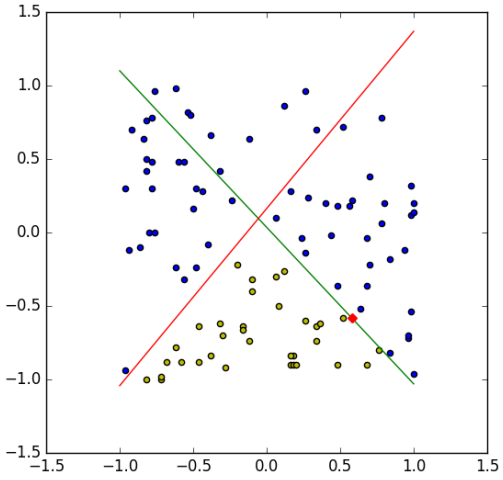

Figure 6.2
Chevron data trained with a single boundary hunter that has a bias which is learnt.

This is all very well but its very easy to learn the bias when the data has no noise. What would happen when training over the data in figure 6.3

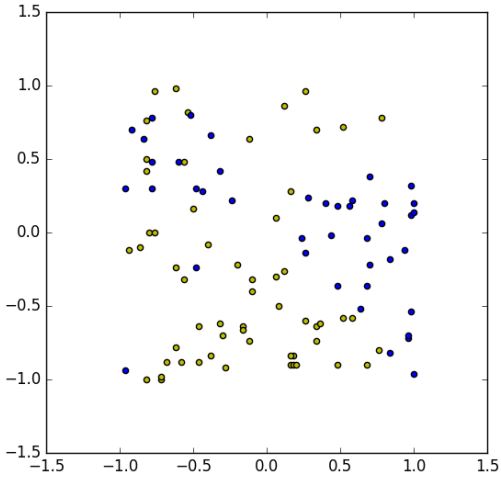

Figure 6.3

With added noise the chevron becomes much less clear and could easily be interpreted in other ways, and while classifying the non noisy side of the chevron is still an option its not clear whether this is the best option. Figure 6.4 shows that the boundary hunter is still able to identify the non noisy side of the chevron.

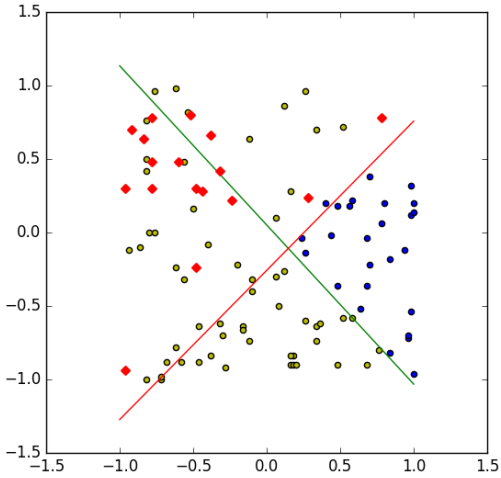

Figure 6.4

Chapter 7

Training in Higher Dimensions

All testing up to this point has been performed in 2D, useful for visualisation but any real world problem will generally occupy a higher dimension. To test the Hyperplane BH in n dimensions the vertices of an n dimensional hypercube will be generated and divided up by two hyperplanes, representing the caring and decision boundaries. Then a Hyperplane BH will attempt to learn the generated boundaries.

7.1 Results

Until $n = 7$ the Hyperplane BH is able to perform at least as well as the generated solution, the data which on the do not care side of the data has a randomly assigned class so it might be possible to perform better than the generated solution. At $n = 7$ it no longer ignores all the data it should.

Any future work would first have to understand what is causing this issue.

Chapter 8

Conclusion

Locating local features in the data and ignoring everything else, a seemingly simple enough problem quickly became a difficult balancing act of reward vs penalty. After attempting to design a loss which encouraged local learning we concluded that this was not a promising avenue to follow.

Using the idea of RBF Networks we where able to develop similar models of boundary hunters but this was not quite solving the problem we had in mind, the network structure meant there was some dependency between the hidden neurons. Removing the network and individually training these boundary hunter neurons did not initially have the results we wanted, after adding biases which where learned along side the weights we started to see the boundary hunters learning local features. Some tests on simple noisy data also proved promising.

Testing in higher dimensions revealed problems which where not seen on the 2D toy examples. Learning boundary hunters becomes difficult for $n > 7$. Consqeuently one remaining question is, what causes the diffcultys in learning when the number of dimentions increases?

This report has demonstrated that the concept of a boundary hunter is plausible but while there is difficulty learning in higher dimensions they can not be used in practice.

Bibliography

[1] Jacobs, R. A., Jordan, M. I., Nowlan, S. J., and Hinton, G. E. Adaptive mixtures of local experts. *Neural computation 3*, 1 (1991), 79–87.