CMP\_SC: 8770 Neural Networks

By: Mikey Joyce

Pawprint: MPJYKY

Due: February 29th, 2024

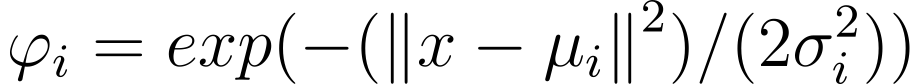
1. Technical Description:

General Introduction:

In this project, we were asked to implement a supervised learning neural network(s). There are multiple goals that are being asked of us. The initial goals are to test out the neural network on easily visualizable datasets known as sanity check datasets. These are data sets that are easy to tell what class is which so it is easy to understand why a neural network may not be performing well if it cannot even accomplish the easiest datasets. We are to analyze the performance of our network utilizing qualitative cues (does the network look like it is performing well when you look at the results on a plot) and by utilizing quantitative ques (compute some metrics for the performance of a neural network to determine if it is properly learning). After it has been assessed that the neural network can perform well on the sanity check data sets, we have been asked to now utilize the networks on a more complex dataset to see if we can make a neural network that provides real world value. Many options were given to us, such as the MNIST dataset, but while computer vision is extremely intriguing and fun, it doesn’t really apply to clinical glaucomatous progression analysis that I engage in. The dataset that I am going to analyze is known as the Indianapolis Glaucoma Progression Study (IGPS). It is a glaucoma clinical study designed to track progression of glaucoma over time, because of this all of the participants within the study have some form of glaucoma. What makes this study unique compared to other clinical glaucoma studies is that the participants of the study had measurements observed every 6 months for the duration of multiple years. One shortcoming of the data contained is the small number of participants. There are roughly 114 participants in the entire study, with only 69 of the patients with attendance at each visit (45 patients did not have consistent attendance over time). The strength of this clinical study is the number of measurements that were taken for the patients. As the goal was to analyze glaucomatous progression, the individuals conducting the study were able to use expensive machines that look at specifics of what doctors believe is contributing to glaucoma. One of the machines that is present within the Indianapolis Glaucoma Progression Study is known as Optical Coherence Tomography (OCT). The Optical Coherence Tomography essentially takes images of the eye that allow ophthalmologists to look at the thickness of the retinal nerve fiber layer (RNFL) and other parts of the optic nerve. Allowing for the imaging of the optic nerve allows assistance for ophthalmologists to classify glaucoma and to keep tabs on a patient’s progression. Unfortunately, this imaging data is not available to me, which could’ve presented a good opportunity to use the images to create a convolutional neural network to classify types of glaucoma, or even predict the progression of a patients’ glaucoma status. But since the data that we do have is quite small, convolutional neural networks (CNN) are out of the picture (also CNNs are strong for imaging, and we have shown that is not my goal currently). A small neural network needs to be created so that there are no issues learning a good model that can accomplish what needs to be accomplished. Even a multi-layer perceptron (MLP) can easily get to have a large number of trainable parameters. I also implemented a multi-layer perceptron in Intro to Computational Intelligence and wanted to try something new to me to ensure that my machine learning toolbox is growing. The last network that we learned about up to this project is the radial basis function (RBF) network. This network seemed quite intriguing and applicable to my problems for a number of reasons. The first, and probably strongest reason for me to implement a radial basis function network is due to the initialization of parameters of this network requiring some form of unsupervised learning. Much of the work that I have already done, has been implementing algorithms such as the fuzzy C-means (FCM) clustering algorithm. The clusters that were generated by FCM for the Indianapolis Glaucoma Progression Study have already been verified to have significance with respect to glaucomatous progression. Because of this, it is natural to want to utilize a radial basis function network where we can pass the already verified centroids to the network as an initialization, which should give the network an advantage when predicting glaucomatous progression versus other networks (MLPs, CNNs, etc.) with access to the same small dataset. Another reason that the radial basis function networks seem like a strong fit for my problem is due to their minimal size. It is a one hidden layer network with a small number of nodes contained within. Since, it is a small dataset, making the smallest network that works was the most desirable, because I do not believe a large network would be able to accomplish anything due to these data constraints. As Dr. Anderson outlined in lecture, in this day and age it is easy to get carried away with making massive networks that are over complicated for the problem at hand, which is something I wanted to avoid. Because of these reasons, I believe the explainability of implementing a radial basis function network is higher than the other options I had at hand. So, in this report we will dive deep into what a radial basis function neural network actually is and will attempt to use the RBF to predict progression with respect to glaucoma.

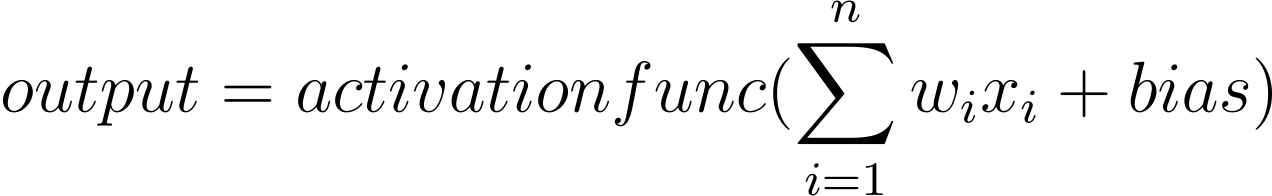
Radial Basis Function Networks:

What is a radial basis function neural network? In my learning of machine learning, I have come up with this analogy for different machine learning methods. Imagine machine learning as a concept of different toys, with each algorithm being a new toy. Many algorithms have very specific outlines for how they should be implemented, this is essentially the design of the toy. A clustering algorithm could be compared to something like a chess game, chess has very specific instructions for how it is played, but it is still non-deterministic. For example, the person who wins is never guaranteed, i.e., on each clustering run, you may get a different result. Another example that supports clustering algorithms being like a chess game, is each different clustering algorithm could be like a chess opening, where some chess openings give the player different advantages that could affect the outcome of a game, similar to how FCM gives the user the advantage of uncertainty over K-means, which could affect the outcome of the clustering. Neural networks however are not like chess at all, there are an infinite amount of blueprints that could easily be imagined and implemented. Because of this, neural networks are more like Legos. As someone who plays with Legos has different types of blocks that have different properties which allows them to essentially make an infinite amount of toys, or different scenes/objects utilizing the infinite combinations of different blocks. The opportunities of Legos, like neural networks, are infinite. You can make a small toy such as a simple house, or you could recreate an entire scene from Star Wars. In neural networks you can create an infinite combination of networks that could accomplish many different goals, just like the Lego builder. To figure out what type of network to build in this infinite sea of possibilities we need access to the same two items a Lego builder needs, an end goal and the schematics of our building blocks. It has already been outlined that we would like to utilize our neural network to predict progression with respect to glaucoma, so the goal portion is covered. What we have not yet discussed are the different building blocks, or Lego bricks, that we have access to when we use a radial basis function network (akin to purchasing a Lego set at the store). If we “purchased” a radial basis function network, what bricks would we get? The first building block we have is the input layer of course as the data needs to get loaded into the network somehow. Next, we have the hidden layer that is comprised of a number of Gaussian nodes to perform a transformation on the data that was loaded in the input layer. The number of Gaussian nodes that are utilized is up to the builder, however it is wise to have some meaning behind the number of Gaussian nodes contained within the hidden layer.



*Equation 1.1: Gaussian radial basis function equation*

Above is the equation for the Gaussian radial basis function kernel. Here we can see we are taking our data points and calculating their distance to the center of the Gaussian kernel. We then divide this by negative one half multiplied by the covariance associated with our kernel squared. This allowed us to take the data from the input layer and apply a non-linear transformation to it. We now take the transformed data to the last block, which is the output layer. The output layer consists of a number of perceptrons that the builder can utilize. The number of perceptrons that should be implemented should revolve around the size of the hidden layer and the problem at hand. Haykin mentions in the book that, “typically the size of the output layer is much smaller than that of the hidden layer” (Haykin 239). For a binary classification problem, a single perceptron is all that is necessary. A single perceptron itself consists of connections coming from the hidden layer that have weights associated with each connection. These weights are then multiplied by our transformed data points and then aggregated through a summing procedure. We then inject a bias term into our signal to get the output of the perceptron. Lastly, with the output signal of the perceptron, we can utilize something called an activation function which aids the network in avoiding non-linearities by mapping the values to an expected range. Below is the equation of the perceptron just described.



*Equation 1.2: The output layer perceptron*

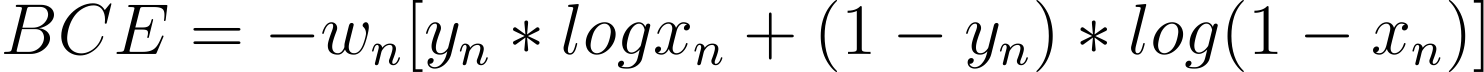
Since we have discussed the main infrastructure of our network, here is a diagram of one of the RBF networks I implemented:

*A diagram of a machine learning process

Description automatically generated*

*Figure 1.1: Radial basis function neural network with 3 radial basis functions in the hidden layer and a single perceptron with a sigmoid activation function in the output layer*

Now that we have a set infrastructure, there are couple of building blocks that need to be discussed that can allow this infrastructure to be useful, like applying icing on the cake. We need to give our RBF neural network a goal to achieve and a means to achieve this goal. For instance, a goal that our network is trying to accomplish when we say we want to predict progression with respect to glaucoma, it is trying to flip a switch for the next time step based on the current data to classify if this patient is going to experience glaucoma progression. To allow our network to achieve a goal like this we could utilize the Binary Cross Entropy loss function.



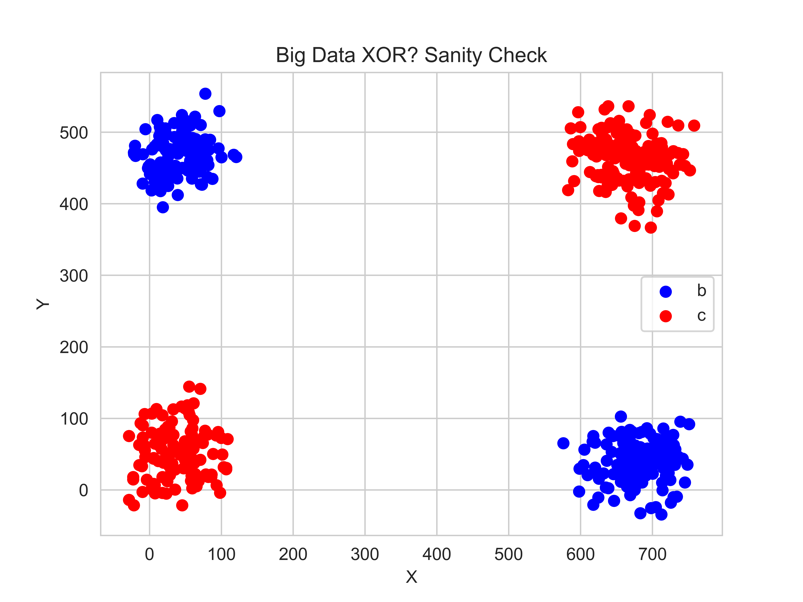
*Equation 1.3: Binary cross entropy PyTorch implementation*

After the forward pass through the network the error is calculated using the binary cross entropy equation (equation 1.3). Then backpropagation is to be performed where we compute the gradients of the loss with respect to the trainable parameters in the network. Refer to Fig 1.1, the trainable parameters within this RBF network are the centroids and standard deviations of the actual Gaussian nodes, the weight vectors that connect the Gaussian nodes to the perceptron, and the injected bias signal. The last block is to apply optimization algorithms over the gradients to tune those tunable parameters in the network to give us the best network to achieve our goal utilizing the outlined infrastructure. The goal of the optimization algorithms are to minimize the objective function that is given to the network by picking new values for the parameters contained within the network. RBF neural networks can be optimized in many ways. The way I opted to optimize this neural network was utilizing a two-step process. The first step was to utilize FCM to establish the centroids of the Gaussian nodes within the RBF network, these establish centroids are not the be changed during the following step. After this I experiment utilizing the adaptive moment estimation optimization algorithm to tune the parameters of the standard deviation of the Gaussian nodes, the weights connecting the Gaussian nodes to the perceptron, and lastly the injected bias. We have covered the necessary building blocks to build a radial basis function neural network, now it is time to implement the network and run some experiments to see if it was correctly implemented, and to understand what types of problems these small networks can solve.

1. Experiments and Results:

Sanity Check:

The purpose of the sanity check in this project is to ensure that the code we implemented is performing properly by visualizing and assessing its performance on a simple dataset. I created a couple of datasets and experiments pertaining to this to ensure that the RBF network that I implemented is working properly. The first experiment that I conducted was creating a XOR like dataset, except instead of having only 4 points like the XOR has, there are 4 different point clouds in the corners of the dataset corresponding to two different classes. I was able to create these sanity check datasets utilizing a Python library called “drawdata” where you can load this library into a Jupyter notebook and draw point clouds yourself. This first dataset I decided to name the big data XOR, and it can be visualized like so:



*Fig 2.1: “Big Data XOR” sanity check dataset visualized*

As we can see there are two different classes contained within the dataset. Class “b” and class “c”. This is a pretty simple dataset clearly and it should be no problem for the RBF neural network to solve. Since we can see there are clearly four different point clouds, we can cheat when creating this first RBF network and feed the cluster count hyperparameter to it as K=4. FCM will fit the dataset with respect to this cluster count and the four different centroids will be found. This is the first step into optimizing the RBF network. Here is the following output of what FCM does when K=4, I am demonstrating this now just as proof that my FCM implementation is sound.

A screen shot of a graph

Description automatically generated

*Fig 2.2: FCM output on “Big Data XOR” when K=4*

It is clear based on this visualization that FCM found the different point clouds with no issues. Let us build the RBF now:

A screenshot of a computer code

Description automatically generated

*Fig 2.3: First RBF created to solve the “Big Data XOR”*

In Figure 2.3, the first RBF was defined. In the first line of code that you can see, we are initializing the RBF network. There are five different necessary *initialization parameters* that need to be understood to properly tune this network. The first initialization parameter is the number of Gaussian nodes that the network will create, this essentially defines the size of the hidden layer. We selected K=4, you can verify this claim when looking at the output of the cell. In the output, there are four different *network parameters* contained within the RBF. The second network parameter in the output are the centroids that the FCM clustering algorithm found for us, as we can see there are four different entries, which is expected since we initialized K=4. The second initialization parameter that we see in the first line of code is the data that you want to pass to FCM to help find the centroids. Theoretically it is best to give it as much of the training data as possible, but there are definitely reasons why you don’t want to do that. For example, if you already have verified clusters that you want to utilize that were found using only a small subset of the data, it is probably best to only pass that subset of the data rather than the entire training set. The third initialization parameter of my RBF network is the starting sigma value. This parameter could be tuned to help give the network a head start at the beginning, but here we just initialize it as a sigma of 1. The fourth initialization parameter that can be seen on the first line of code is the “mode.” This mode has three different values: mode=0 is having one shared covariance (sigma) value amongst all of the Gaussian nodes in the hidden layer, mode=1 is having one covariance value for each Gaussian node in the hidden layer, and mode=2 is having a full covariance matrix for each Gaussian node in the hidden layer. On this run, we can see mode 0 is selected which is giving a shared sigma for each node in the hidden layer. This covariance mode is going to be something that I will be experimenting with, as it is a unique aspect to the RBF network and should be explored to fully understand the RBF network. Lastly, there is a fifth initialization parameter, the number of perceptrons we have in the output layer. Since this is a binary classification problem, we can just utilize a single perceptron to solve this. Now I will explain the four network parameters that we see in the output. The first network parameter is the shared covariance, with its initialization at 1.0. The second network parameter contains the centroids that FCM found for the “Big Data XOR” dataset. The third network parameter contains the weights that connect the hidden layer nodes to the output perceptron(s). The fourth network parameter is the bias that we inject into the signal. The last two building blocks that will allow this network to be trained are the loss function and the optimization algorithm specifications. Since this is a binary classification problem the loss function that is utilized is the binary cross entropy loss function (equation 1.3). The optimization algorithm that was utilized was the adaptive moment estimation algorithm with a learning rate of 0.01. Now we can train this RBF since it has been defined.

A graph with a line

Description automatically generated

*Fig 2.4: RBF BCE Training Loss Over 31 Epochs*

Here we can see a nice pretty curve that shows the network is learning throughout the training cycle (for the most part). The reason 31 epochs was selected was due to the “convergence” of the network. I outlined in the training cycle that if the BCE loss dips below 0.1 then stop training. This may not be the wisest way to define convergence, but I was just trying to avoid running the training for too long, as on some runs it would start unlearning, especially on the harder experiments. This clean loss curve demonstrates that this problem that we outlined for the RBF was not hard for our network as it was easily able to find new parameters to improve itself and didn’t take a step backwards or steps through a plateau (well it stepped through a plateau at the start). Let’s look at some qualitative and quantitative measures for the performance on the test data.

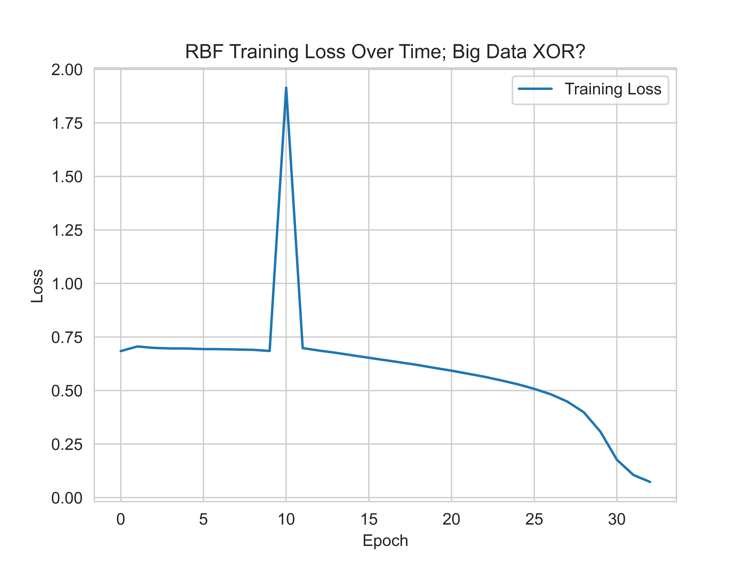
A graph with black dots

Description automatically generatedA blue squares with white text

Description automatically generated

*Fig 2.5 (left) & 2.6 (right): Prediction labels visualized; Confusion Matrix: TP=63, FP=0, FN=0, TN=47*

As we can see from the qualitative measure, which is visualizing the prediction labels on the test data on the actual 2D plane that the data exists on, our network seemed to have strong performance, accurately separating the classes from each other. From the quantitative measure, we see the same exact message conveyed, our RBF network perfectly is able to classify the data points from each other. So, it appears the implementation of the RBF is acting as expected, but there is something that we should explore right here. We utilized four different nodes in the hidden layer here, and each node had the same shared covariance. Since we have the ability to utilize a full covariance matrix, theoretically one RBF node could separate these classes (imagine an ellipsis that encompasses one of the classes within and the other class on the outside). The next RBF is going to have a single node in the hidden layer and the full covariance matrix mode is going to be switched on. The last initialization parameter that is going to be different from the first experiment is the covariance initialization being set at 0.75. I decided to do this because in the code, when initializing the full covariance matrix within the network, I utilize a random initialization function. I am essentially telling the network scale the random selections down a bit by selecting sigma=0.75.



*Fig 2.7: Training loss of RBF with single node hidden layer, “convergence” at 32 epochs*

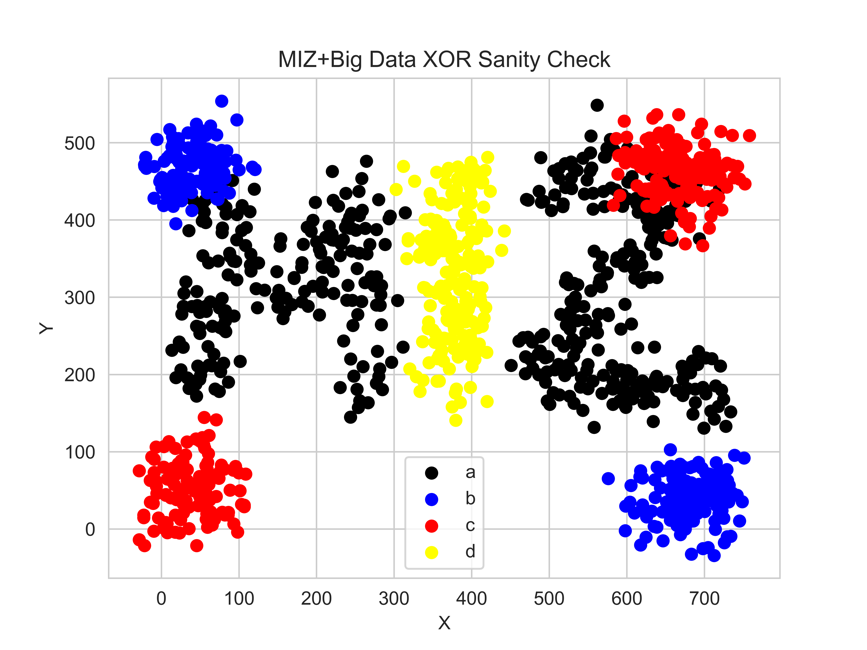
Below I visualize the predictions of this model on the data along with an ellipsis plotted that was painted utilizing the eigenvalues and eigenvectors of the covariance matrix that was optimized within the network. There is no need to show the confusion matrix on this experiment because it got perfect results, this experiment was mainly to prove that the full covariance mode for the RBF nodes works properly.

A graph with a blue circle

Description automatically generated

*Fig 2.8: RBF with one node in the hidden layer predicts on the Big Data XOR dataset along with plotting the eigenvector/eigenvalue ellipsis corresponding to the full covariance matrix for the single Gaussian node*

Now there is one more experiment I would like to conduct on the sanity check. This experiment is going to be a multiclass classification problem. I created a new dataset that is essentially the merging of two different datasets that I made. The first dataset is the big data XOR that I showed above, it is merged with a second dataset I called MIZ (you will see why), the title of this merged dataset is “MIZ big data XOR,” here is a visualization:



*Fig 2.9: MIZ Big Data XOR dataset*

The reason I am running this experiment is for two reasons. Reason number one is to show that my RBF implementation can handle problems that require more than one perceptron in the output layer. This will become useful if I ever want to do multiclass classification or a multivariate regression with this RBF in the future. The second reason for this experiment is to see if I can trick my RBF neural network to make wrong predictions, this is why I made sure that some of the classes are overlapping over each other. Here is what the network looks like:

A screenshot of a computer

Description automatically generated

*Fig 2.10: RBF setup for Miz Big Data XOR dataset*

The reason I wanted to show the network I crafted for this experiment in detail is to outline that my infrastructure can solve more than just binary classification. As we can see here, I told the network to have 4 output perceptrons because there are 4 target classes, I probably could’ve utilized only 2 output perceptrons, but I decided on 4 for ease of training. If you look at the 3rd network parameter in the output of the cell, you can see that there are four different weights, and the 4th network parameter there are four different biases. This is expected because we asked the network to create four different perceptrons in the output layer. Another major difference is the change of loss function to be the cross entropy loss function, which allows for multiclass classification. Lastly, it is worth it to note that on this run, I changed the covariance mode to be one value for each centroid, mainly because I haven’t shown this mode yet (this is also the mode that Dr. Anderson included in his RBF implementation on the Github). Let’s train!

A graph with a line

Description automatically generated

*Fig 2.11: RBF #3 loss through 50 epochs*

A graph with black and red dots

Description automatically generated

*Fig 2.12: RBF #3 predictions over MIZ big data XOR*

A diagram of a confusion matrix

Description automatically generated

*Fig 2.13: RBF #3 confusion matrix over MIZ big data XOR (~75% accuracy)*

As we can see from the training the model didn’t converge based on the criterion that I had outlined earlier. We can see in the confusion matrix that was produced (and through the visualization of the predictions) that the RBF did pretty well on the XOR datapoints, but when it got to the MIZ data, it really couldn’t tell the difference between those two classes. My first theory for why this is, is due to FCM’s initializations of the centers not being good enough, but after switching the centers to be able to become a tunable parameter within the network the performance of the network was similar. I decided to switch the mode to be one shared sigma for each node in the hidden layer and increase the learning rate to 0.1. I thought it was peculiar that giving the network a shared sigma value aided in distinguishing the MIZ classes. I honestly can’t think of a reason why giving the RBF less specifications to fitting the data makes it fit the data better. What we can see here is that it does struggle on one of the XOR classes a bit more, which was a hypothesis I had going into this specific experiment.

A graph with different colored dots

Description automatically generatedA diagram of a confusion matrix

Description automatically generated

*Fig 2.14 & 2.15: predictions visualized qualitatively and quantitatively for MIZ big data XOR RBF: ~85% accuracy*

Glaucomatous Progression Classification:

Here we will run a few experiments to determine if a radial basis function neural network can learn how to predict glaucomatous progression. First a note on the definition of glaucomatous progression. The way structural glaucomatous progression is defined based on the labels provided in my dataset are from six different variables contained within the data. The first equation that computes structural progression is:

*(mean\_RNFL\_thickness – BaselineRNFL)/BaselineRNFL <= 0.08*

This is essentially saying if the RNFL thickness (a measurement obtained from OCT imaging of the retina) decreases, then progression is happening. Here is the second equation that can turn the progression label on is:

*(cup\_disc\_horiz\_ratio – Baseline\_ratio)>=0.2 OR (cup\_disc\_vert\_ratio – Baseline\_ratio)>=0.2*

This equation is saying if the cup to disc ratio (another measurement obtained from OCT imaging of the retina) increases, then progression is happening. It is important to understand that the definition of progression is not something that is entirely agreed upon in the ophthalmology community as progression can be defined many different ways. Now earlier, I mentioned that the RBF neural network interested me for the application of our problem because part of the RBF training process can be influenced by having verified clusters, which would increase the explainability of my solution. So, because of this we are going to formulate the first experiment to utilize the 12 dimensional feature subset of our data that was used to obtain our verified clusters. Five of the features in this feature vector are simple measurements of the pressures within the eye and the heart rate of the patient. Using these five features, a simulation via a set of verified differential equations is run to produce seven synthetic features. Lastly, the labels of the feature vector are the binary structural progression labels that is experienced in the NEXT time step (we are trying to predict whether or not this progression is going to be present within the next 6 months essentially). Now, for the experiment, since the number of clusters that we have verified is 3, we are going to pass this to the RBF. Also, the IGPS study was taken over many different visits, and we will want to use much of this data, but it is important only to pass the first visit on initialization so that FCM utilizes only the datapoints that were present when creating the verified clusters (we want to make sure they are the same clusters). We are going to use a batch size of 6 and a learning rate of 0.001 with the Adam optimizer over 2000 epochs (I want the network to learn slowly after seeing that data many times) to do binary classification. I will test out this RBF utilizing all three of the covariance modes and compare their performance.

A graph with a line going up

Description automatically generatedA graph with a line going up

Description automatically generated

A graph with a line drawn on it

Description automatically generated

***Top Left****: one single sigma,* ***Top Right****: one sigma per RBF node,* ***Bottom:*** *full covariance matrix*

On the next page there are the 3 confusion matrices that are associated with the models trained above (i.e. 1st=one sigma, 2nd=one sigma/node, 3rd=full covar). Some scores are also provided for each model on the page following the confusion matrices.

A blue squares with white text

Description automatically generatedA diagram of a confusion matrix

Description automatically generated

A blue and white diagram

Description automatically generated

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Accuracy | Precision | Recall | F1 Score |
| Single sigma (mode=0) | 50.4% | 52.9% | 41.5% | 46.5% |
| One sigma per RBF node (mode=1) | 54.4% | 55.5% | 61.5% | 58.1% |
| Full covariance matrix (mode=2) | 53.6% | 54.1% | 70.7% | 61% |

From the loss curves, we can see the first two modes have a somewhat consistent learning rate, where they experienced a plateau mid training. While the last mode had quite a choppy error curve. When looking at the confusion matrices and their associated scores, the best accuracy came from mode=1. The scores on mode=2 seem to be decent, but I am unsure these scores are particularly good because when you look at the confusion matrix, it looks like it tried to predict more leaning on everyone who has glaucoma is getting worse, despite the distribution of the data being somewhat equal. Either way, the performance of all of the models are not really too much better than tossing a coin on this one. My theory of why the RBF was not finding a particularly good solution with this formulation is due to trying to utilize the same 12d feature set to obtain the same verified clusters. Because of this, I want to craft a smaller feature set that has variables that are more in tune with how we define what progression is in this dataset. Since, before we had verified clusters, using a new feature set is going to mean those verified clusters are no longer going to be a part of this experiment. This means I can’t just assume that K=3 is going to work, so this time we will train 30 different models from K=1🡪30 and pick the model that has the highest F1 score on the testing data. Also, because the last experiment the covariance mode=1 showed the most stable/promising results, I will select this mode for this experiment. All other hyperparameters are going to remain the same. The new feature set is going to include some of the same features from the initial 12d feature set while also bringing in 3 features from the OCT imaging machines that actually define progression, and 2 features that indicate higher risks of progression (age and acuity=what stage of glaucoma is the patient at), this feature set has 9 dimensions. Here is the results of the experiment:

A graph with a line

Description automatically generated

A diagram of a confusion matrix

Description automatically generated

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Accuracy | Precision | Recall | F1 |
| RBF; k=18 | 61.6% | 60.2% | 76.9% | 67.6% |

The model that was performed with the best F1 score was one with 18 different RBF nodes in the hidden layer. It was nice to see that we were able to obtain a score that was better than simply just flipping a coin. However, this is still underwhelming for being a reliable indication of whether someone is going to experience glaucomatous progression. I could do similar experiments, where I mess with the parameters and feature vector, however before I formalized my experiments, I was exploring different parameter setups, and no models could do better than this one. Well, there was one model that was able to get around a 70% accuracy, but that model had access to the current progression status, which I determined is kind of cheating, because if the model knows this person is already experiencing progression it makes it much easier to say with confidence they will experience progression in the next 6 months.

1. Reflection:

In this project, we explored the inner workings of a radial basis function neural network. We provided context to what the radial basis function network actually is, and how it could possibly be a good tool to solve a novel glaucoma problem. We then implemented the radial basis function in Python utilizing libraries such as PyTorch and NumPy, creating a nice module that had many different capabilities presented to the user. After creating a sanity check dataset, the RBF module was tested to ensure it was working properly. Strangely enough, the sanity check process took the longest because I was making sure everything was inline, with few bugs and instabilities, that way I could train my RBF in obscure ways if I wanted to (i.e. the sanity check where I had k=1). After doing the sanity check process, fixing bugs, and running formal experiments, we decided to try and tackle the novel problem of predicting future glaucomatous progression. The results of utilizing the RBF on the progression problem, were quite underwhelming, but it is expected that everything that you do in research might not have expected results.

For the sanity check experiments, everything did turn out the way that I thought it would. I particularly enjoyed the experiment where I was solving the XOR like problem and I utilized only 1 node within the hidden layer and showed the ellipsis that was generated from the eigenvalues/vectors of the covariance matrix associated with the hidden node.

For the experiments involving structural progression, I am not sure if they turned out exactly how I envisioned them. Obviously, I wanted them to succeed with high accuracies, but this was not something that I was entirely expecting as I am not too sure how much I trust the quality of some of these clinical databases. There could be various measurement errors involved with the measurements that could be adding immense amount of noise, it was entirely possible that there isn’t a set of weights for any radial basis function network that could solve the progression problem on that given dataset. I was hoping for decent results on the binary classification, because then I would’ve done a multiclass classification had the results been successful. I would’ve introduced the other form of progression which is functional progression (is the patient experiencing a loss of vision) and seen if we could predict that as well. But, since we couldn’t even predict the structural progression, which is calculated from data rather than life experience, I determined it was not a good idea to make the problem even harder than it already is. I was also a little underwhelmed that I wanted to make a small network with high explainability, but the best performing network I could make on this problem seemed to be a black box (high number of hidden nodes with no explainable significance).

While the radial basis function network seemed attractive at the start of this project for my problem, discovering its lack of ability to predict the glaucomatous progression allows us to understand possible reasons for this. Again, aside from possible noise due to the quality of data, which would limit predictive power, there could be inherent reasons the radial basis function network didn’t perform. Clustering, which the RBF is reliant (to some degree) on, thrives on separability in the data. It is very possible that even though my feature vectors were 12D and 9D, up in that space the “point” clouds that the data exists on aren’t as separable as we think, which could be limiting the RBF. Because of this, if I had more time, I would possibly be interested in implementing something like a siamese neural network, or just pairwise metric learning in general. I would create a feature vector with my data pairs that is exhaustive for the entire IGPS dataset, since I can afford this high computational cost due to having a small dataset. I think this could possibly work better than the RBF network, because this type of neural network could theoretically deal with the lack of separability between the progression vs non-progression classes that are present in the IGPS dataset. I am not too sure if this is something that could work, but it seems quite intriguing.