Adaptive Fuzzy Rule Learning with Non-Linear Fitting

Vignesh Krishnakumar, Sahitya Potluri, Abhiram Ravi

Indian Institute of Technology, Madras

Abstract. In this project, we experiment with Fuzzy Rule Learning for approximating functions over compact sets in highly non-linear systems. After studying and implementing existing approaches for learning fuzzy rules from example data on different Fuzzy Rule-based Systems, we propose a novel and adaptable extension to the Radial-basis fuzzy function approximation scheme by adding higher degrees of freedom with respect to its parametrization. We also propose an algorithm to perform this approximation and show that there is a substantial reduction in approximation error.

1 Motivation

Fuzzy logic is a tool for embedding human knowledge in a structured fashion into efficient algorithms. It is considered as a logical system that provides approximate reasoning rather than exact reasoning. Fuzzy logic systems can be used to design intelligent systems on the basis of knowledge expressed in human language. It combines both symbolic and numeric information.

Fuzzy models combine the knowledge of the system being modelled with the human expertise. With increasing complexity of the systems, constructing models from the underlying knowledge has become a very difficult task. Also, with increasing complexity of functions in practical applications, making accurate and adaptable approximations from data is a major concern. Hence, in this project, we explore methods of constructing and analyzing fuzzy models in a supervised setting.

2 Background

In this section, we present the basic ideas related to Fuzzy systems and Fuzzy Inference Engines. We also summarize the different types of *Fuzzy Rule-based Systems* that are practically employed for inference tasks.

2.1 Fuzzy inference

The following picture depicts a systematic method of how a fuzzy inference engine works.

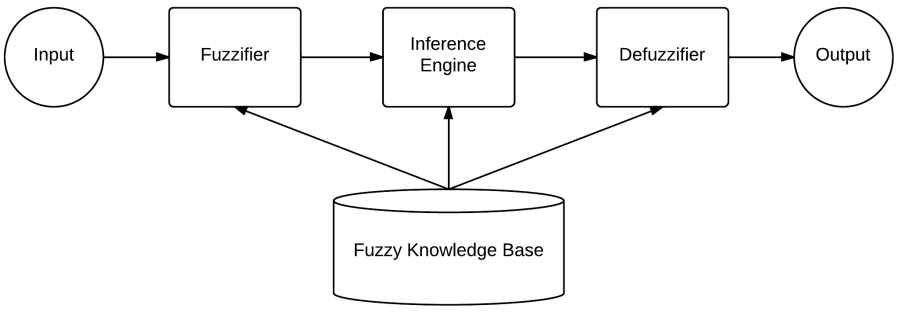


Fig. 1: Illustration of Fuzzy Inference

– *Fuzzifier* : Converts the crisp input to a linguistic variable (high/low etc) using the membership functions stored in the fuzzy knowledge base.
– *Inference Engine* : It converts the fuzzy input to a fuzzy output using If-Then fuzzy rules.
– *Defuzzifier* : Converts the fuzzy output to a crisp value using membership functions similar to the ones used in the Fuzzification step.

2.2 Fuzzy Rule-based Systems

There are three types of FRBS namely *Mamdani* Fuzzy models, *Sugeno* Fuzzy Models, *Tsukamoto* Fuzzy models. The main difference between them is the types of inputs and outputs.

– **Mamdani FRBS** :
  • If X_1 is A_1 and .... X_n is A_n, then Y is B. In this X_i is a real valued input, A_i is a fuzzy linguistic variable for input while Y is an output variable, B is fuzzy linguistic variable for output.
  • Example : If x is small, then y is large.
– **TSK FRBS** :
  • If X_1 is A_1 and .... X_n is A_n, then Y is $f(X_1, ..., X_n)$. Here f is a crisp function which is generally polynomial in X_i's.
  • Example : If x is small and y is small, then z = x + y - 1.
– **Tsukamoto FRBS** :
  • The consequent of each fuzzy rule is represented by a fuzzy set with a monotonical membership function.
  • Example : If x is small, then y is C_1.

A fuzzy system approximates a function by covering the graph with fuzzy patches (fuzzy sets) and taking a weighted average of patches that overlap. This approximation gets better as the fuzzy patches grow in number and shrink in size. In this project, we implemented a basic method to generate fuzzy rules from numerical data and then improvised it to capture the intricacies of the underlying function.

3 Related Work

In this section, we present some of the existing approaches for performing fuzzy rule learning.[4] have looked at learning fuzzy rules from examples and follow a Mamdani-like fuzzy rule based system. [3] has neatly presented the connection between radial basis function approximators and fuzzy rule learners. [1] have looked at learning fuzzy rules by fitting ellipsoids onto the function surface, and then using the corresponding projections along the coordinate axes to learn the widths of the membership functions.[2] have presented a theoretical analysis for solving the non-linear optimization that arises during radial basis function approximation using Gaussian kernels.

4 Methodology

In this section, we describe the various approaches that we have implemented and analyzed. The basic idea is the following. Given a set of training data points, we would like to learn a set of fuzzy rules that map fuzzy inputs to outputs (fuzzy/crisp), and establish a defuzzification process if the output of the rules are also fuzzy memberships.

4.1 Mamdani Based Approach

We'll discuss this approach for the two variable input case which can be extended to any number of variables.

$$\text{Input} = (x_{i1}, x_{i2}, y_i)\ \forall i = 1 \text{ to } N$$

x_1 and x_2 denote input variables, y denotes the output variable.

The approach used involves the following steps.

- **Divide the input and output space into fuzzy regions.**

  Divide the domain of each variable into a certain number of regions each of which correspond to a fuzzy membership function. The shape of each membership function is triangular (can be modelled as any mathematical function). One vertex lies at the centre of the region with a membership value 1 and the other two vertices lie at the centres of the two neighbouring regions with membership values zero.

- **Generate fuzzy rules from given data pairs.**

  We'll generate one rule for each input-output pair and resolve the conflicts later. Determine the membership of each of x_{i1}, x_{i2}, y to different fuzzy regions. Assign each of the variables to the fuzzy set with maximum membership value. Obtain a rule for each data point based on the above fuzzy linguistic input and output values obtained in the previous step.
  Example : If x_1 is B_1 and x_2 is C_1 , then y is B_1. Here B_1, C_1 are the fuzzy linguistic values.

- **Generate a degree to each rule.**

  Since we are generating a rule for each data point, it is highly probable that there will be conflicting rules. This step resolves the conflicts arising among the rules that are fired at any current scenario. We resolve this by assiging a degree to each rule and selecting the one with maximum degree when conflict arises. The degree assigned to each rule is the product of the maximum membership values of the variables that generated it. If x_1 is A and x_1 is B , then y is C. D(Rule) $= m_A(x_1)m_B(x_2)m_C(y)$ If we know apriori some information on how useful a given data point is, we can incorporate that information also in assigning a degree to each rule. D(Rule) $= m_A(x_1)m_B(x_2)m_C(y)m^k$ where m^k is the usefulness of the k^{th} data point that generated this rule.

- **Create a combined fuzzy rule base.**

  A combined fuzzy rule base contains rules generated from numerical data and the linguistic rules. Each linguistic rule also has a degree that is assigned by human expert. This degree depicts the expert's belief on how important the rule could be. In our case, the combined fuzzy rule base is a two dimensional matrix (here two corresponds to the number of input variables) with rows as the number of fuzzy sets described on x_1 and columns as the number of fuzzy sets described on x_2. If there is more than one rule in one box of this fuzzy rule base, we use the rule with the maximum degree. In this way both numerical and linguistic information are incorporated into a common framework which is the combined fuzzy rule base.

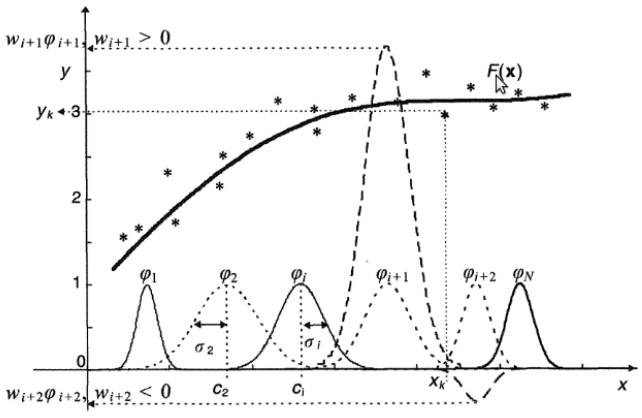
– **Determine a mapping based on the rules (Defuzzification)**

The crisp output for a given value of the input is the centroid of the resultant region. Output y = $\frac{\sum_{i=1}^{K} m^i_{O^i} \bar{y}^i}{\sum_{i=1}^{K} m^i_{O^i}}$ In the above formula, O^i denotes the output region of rules i, $\bar{y}^i$ denotes the centre of the region O^i, K is the number of fuzzy rules in the combined FRB and $m^i_{O^i}$ is the product of the memebership values in the antecedant of rule i. $m^i_{O^i} = m^i_{I_1}(x_1) m^i_{I_2}(x_2)$

4.2 Standard Radial Basis Function Approximation

This approach uses the fact that neural networks and fuzzy logic models are similar in various aspects.

– Learning of fuzzy rules from data and training of radial basis function network are mathematically equivalent.
– These two models can be used to approximate any continuous real valued function.



In the above figure, $*$ corresponds to the training data, φ are the basis functions, $F(x)$ is the non-linear approximating function.When the basis functions φ are fixed (i.e both mean and variance), the approximation is linear in the parameters (only weights here).

In the RBF framework, the approximationg function $F(x)$ takes the form as depicted below.

$$F(x) = \sum_{i=1}^{N} w_i \varphi_i(x, c_i)$$

w_i are the weights to be learned, c_i are the centres of the basis functions φ_i.

This problem is similar to the problem of learning fuzzy rules from a given data. In the setup described above, consider the gaussian basis functions as memebership functions to correlate the similarity between these two problems.

The optimal set of weights obtained by trying to minimize the squared error are the following.

$$\mathbf{w} = (\Phi^T \Phi)^{-1} \Phi^T \mathbf{y}$$

Here Φ is a matrix with rows as the number of radial basis functions and columns as the number of data points. It is a matrix whose values are φ_i's applied on the training data. y is a column vector with the output values of the entire training data.

We used the above equation to get the optimal set of parameters which are the weights given to various fuzzy sets in approximating the underlying function. More specifically, for a given input value, the output is obtained by considering a weighted combination of the membership values of the input to various fuzzy sets (gaussians here). This in some sense can be interpreted as the fuzzy rule we would be interested in learning.

4.3 Gaussian Basis Function Approximation with Non-linear Optimization

In the RBF approximation approach, suppose we also made the gaussian mean and variance parameters part of the training. This would result in the following non-linear optimization problem

$$\text{Minimize} \sum_{i=1}^{N} \left(y_i - \sum_{j=1}^{K} w_j \ exp\left(\frac{-(x_i - \mu_j)^2}{2\sigma_j^2} \right) \right)^2$$

where w_j, μ_j, and σ_j $j = 1...K$ are the parameters

In order to solve the above optimization problem, we propose to use a Gradient Descent algorithm. We first define

$$e_i = y_i - \sum_{j=1}^{K} w_j \ exp\left(\frac{-(x_i - \mu_j)^2}{2\sigma_j^2} \right)$$

$$E = \frac{1}{2} \sum_{i=1}^{N} e_i^2$$

$$C_j = 1/2\sigma_j^2$$

Differentiating the above objective function E with respect to each of the parameters, we get

$$\frac{\partial E}{\partial w_j} = -\sum_{i=1}^{N} e_i \ exp\left(-C_j (x_i - \mu_j)^2 \right)$$

$$\frac{\partial E}{\partial \mu_j} = -\sum_{i=1}^{N} 2 \ e_i \ C_i \ w_i \ exp\left(-C_j (x_i - \mu_j)^2 \right) \ (x_i - \mu_j)$$

$$\frac{\partial E}{\partial C_j} = \sum_{i=1}^{N} e_i \ w_i \ exp\left(-C_j (x_i - \mu_j)^2 \right) \ (x_i - \mu_j)^2$$

Algorithm 1 Gradient Descent Algorithm

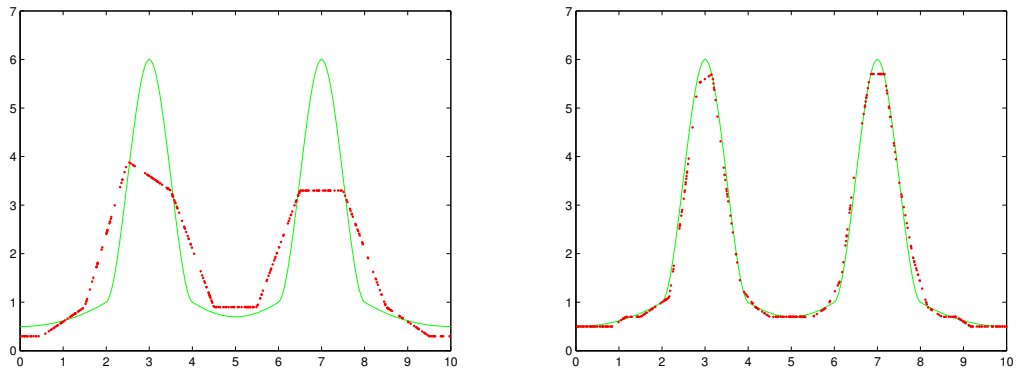
- Choose m = Number of Gaussians
- Perform kmeans on the training data to get m clusters
- Initialize the Gaussian centres to the cluster centres
- Let d be the average inter-cluster distance
- Initialize $C_i = M/d^2$
- Find w_i's using Pseudo Inverse
- while ($E \leq$ threshold)
  - $w_i \leftarrow w_i - \eta_w \frac{\partial E}{\partial w_i}$
  - $C_i \leftarrow C_i - \eta_C \frac{\partial E}{\partial C_i}$
  - $\mu_i \leftarrow \mu_i - \eta_\mu \frac{\partial E}{\partial \mu_i}$

5 Empirical Evaluation

In this section, we compare the performances of the approaches described in the previous section. We have implemented the algorithms in MATLAB. The basic setup is as follows. We first fix a *function* of our choice and restrict the domain of the function to a *compact set*. We then sample a large fixed size set of points uniformly from the functional graph. We now divide these points into a training-test split, in a 2:1 ratio. We now use the training data to learn the fuzzy rules, and then report the error and performance on the test data. The green curves represent the function that we are trying to approximate. The blue curves represent the fuzzy sets. The red dots represent the output of our approximation on the test data.

5.1 Mamdani Approach

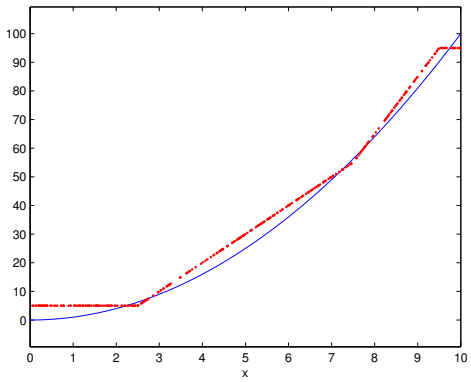
Here, we look at the performance on three different functions, as shown in the figures, we also report the Mean Squared error on the test data. Note that we have fixed the number of fuzzy input and output sets predefined before learning the rules.



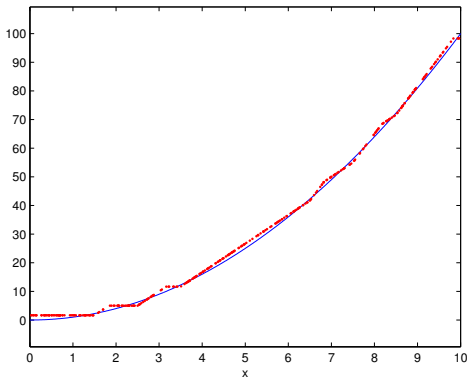
(a) 10 fuzzy input sets & 10 fuzzy output sets

(b) 30 fuzzy input sets & 30 fuzzy output sets

Fig. 2: Mamdani fuzzy function approximation for a spline with two peaks . The fuzzy sets (input/output) are triangular. We see that for 10 fuzzy sets, similar to the case with a single peak, the peaks here fail to be captured correctly. The number of fuzzy sets was increased to 30, and still the approximation is fairly inaccurate, and also has a lot of redundant information by maintaining several fuzzy sets for the flat regions.

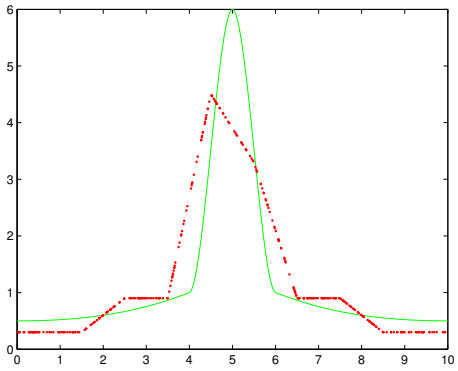


(a) 10 fuzzy input sets & 10 fuzzy output sets

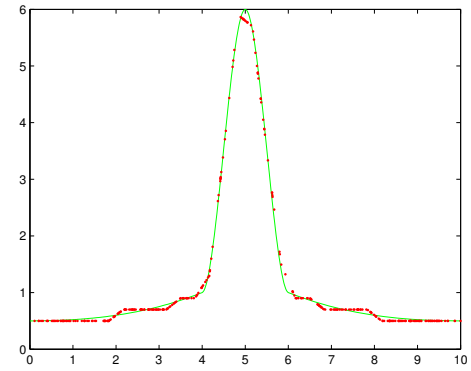


(b) 30 fuzzy input sets & 30 fuzzy output sets

Fig. 3: Mamdani fuzzy function approximation for $y = x^2$. The fuzzy sets (input/output) are triangular. We see that the approximation is better for a larger number of fuzzy sets.



(a) 10 fuzzy input sets & 10 fuzzy output sets

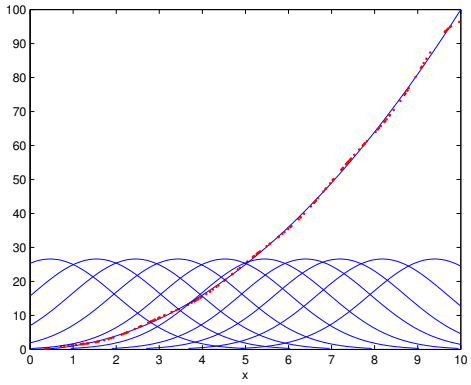


(b) 30 fuzzy input sets & 30 fuzzy output sets. We need these many fuzzy sets to even get a decent approximation!

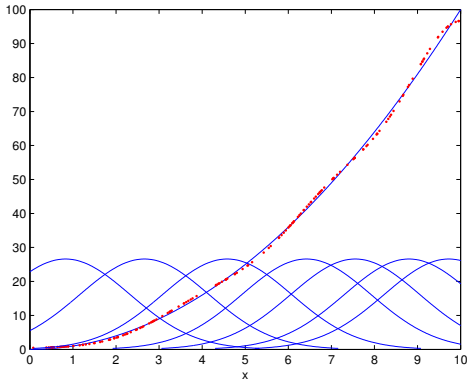
Fig. 4: Mamdani fuzzy function approximation for a spline with a single peak . The fuzzy sets (input/output) are triangular. We see that for 10 fuzzy sets, the peak fails to be captured correctly. The number of fuzzy sets had to be increased to 30, to even get a decent approximation. Also, note that there is a lot of redundant information with a large number of fuzzy sets on the two flat sides.

The main observation is that there if we fix the fuzzy sets explicitly before the training data is seen, we would need a lot of fuzzy sets if we want to capture the intricate parts of the functions being learnt. We now move forward to perform radial basis function approximation.

5.2 Standard Gaussian Basis Function approximation

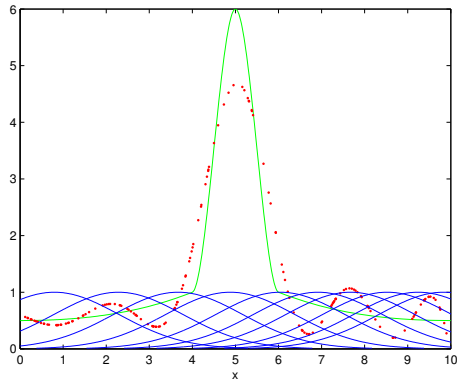


(a) 10 fuzzy input sets

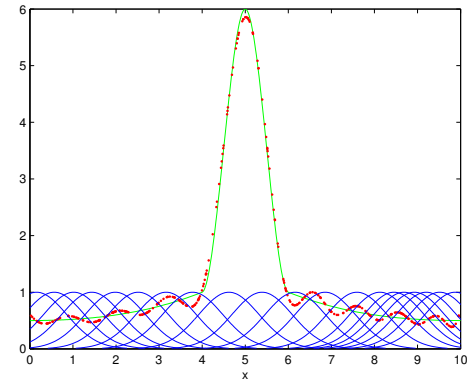


(b) 7 fuzzy input sets

Fig. 5: For $y = x^2$, we see that this approach shows better performance than the mamdani approach for the same number of fuzzy sets. We would like to experiment more and see if this behavior extrapolates for other complex functions as well.



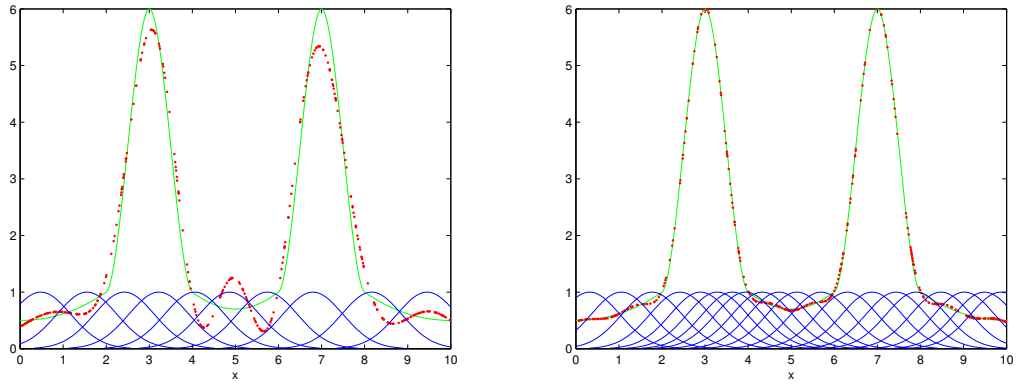
(a) 10 fuzzy input sets



(b) 20 fuzzy input sets

Fig. 6: RBF approximation for a spline with a single peak . The fuzzy sets (input/output) are gaussian, are are denoted in blue. We see that for 10 fuzzy sets, the peak fails to be captured correctly. The number of fuzzy sets had to be increased to 20, to even get a decent approximation. Also, note that there is a lot of redundant information with a large number of fuzzy sets on the two flat sides, in addition to the fact that these flat surfaces are not properly captured.

In this section, we look at the performance on same three functions, assuming gaussian basis with fixed variances. We also fix the number of gaussians before hand. By *fixed*, we mean that these values are fixed before the training data is seen. To evaluate the centers, we perform k-means on the x coordinate of the training data.



(a) 10 fuzzy input sets

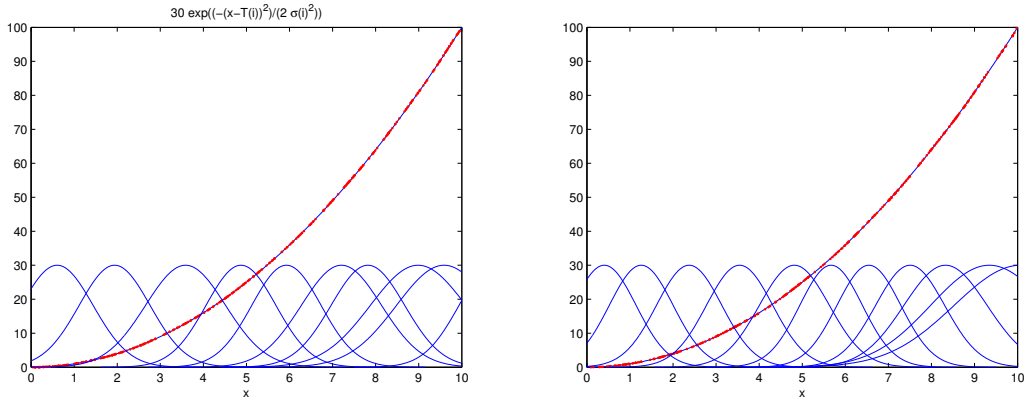
(b) 20 fuzzy input sets

Fig. 7: RBF approximation for a spline with two peaks . Similar to the one peak case, we see that for 10 fuzzy sets, the peak fails to be captured correctly. The number of fuzzy sets had to be increased to 20, to get a good approximation. Also, note that the flat surfaces are better captured with this increase in number of fuzzy sets.

The main observation here is that fixing the means and variances of the gaussians (given a fixed number of gaussians) before hand, although the performance shown is good for fairly simple functions, for highly non-linear functions with regions of high information (peaks), this method shows suboptimal performance, or rather, no substantial improvement over the ordinary mamdani approach.

5.3 Gaussian Basis Function approximation with Non-linear optimization

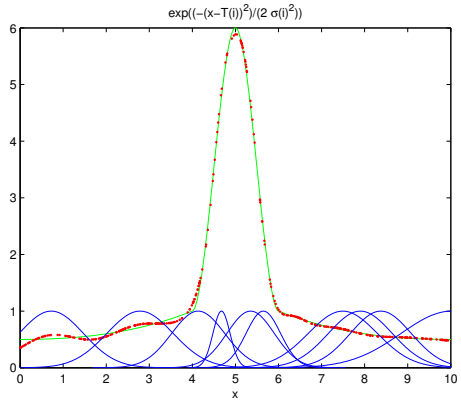
In this section, we look at the performance of our proposed gradient descent algorithm for updating the weights, means and variances for the gaussian fuzzy sets. We only fix the number of gaussians in this case.



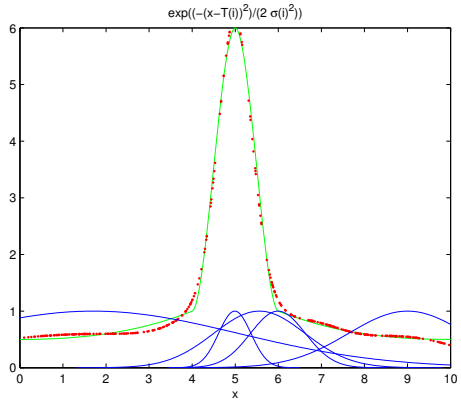
(a) 10 fuzzy input sets

(b) 12 fuzzy input sets

Fig. 8: For $y = x^2$, we see that the performance is as good as standard RBF, which is expected.

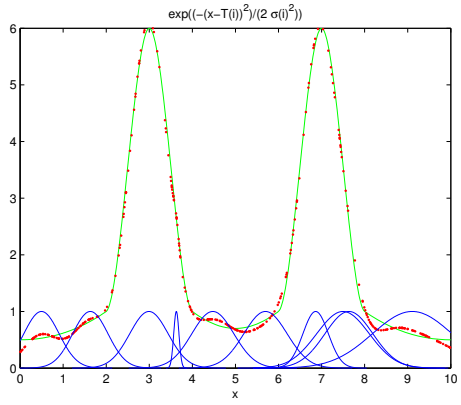


(a) 10 fuzzy input sets

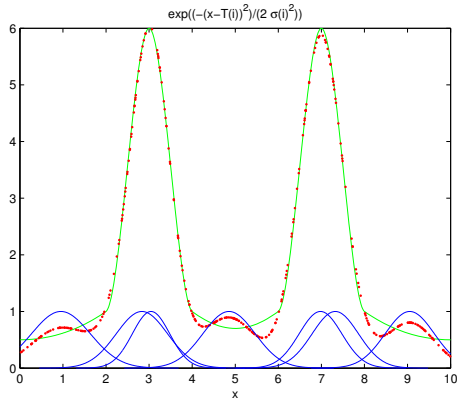


(b) 5 fuzzy input sets

Fig. 9: RBF approximation with non linear optimization for a spline with a single peak . We observe that the performance has substantially increased, for the same number of fuzzy sets. More specifically, we observe that the variances of the gaussians are adjusted in a way that captures the parts of the function with high information density. We also observe that the redundant information in the system has reduced, since the flat areas of the function now correspond to fuzzy sets with high variance.



(a) 10 fuzzy input sets



(b) 7 fuzzy input sets

Fig. 10: RBF approximation with non linear optimization for a spline with a two peaks . We observe a similar substantial decrease in error. Gaussians with low variance capture the *peak* features of the spline.

6 Discussion

By varying width, number of fuzzy sets and their centers, we add more degrees of freedom. By adding these extra degrees of freedom, and solving a non-linear optimization problem, we obtain a stronger approximation to more complex functions. The power here comes in the flexibility of defining the fuzzy sets. Parts of the function that carry higher *information,* are better captured using fuzzy sets with smaller variance, with the means of the fuzzy sets being simultaneously learnt. We also observe that the Mamdani based rule mappings, where fuzzy inputs are mapped to fuzzy outputs are outperformed by rules that map fuzzy

inputs to crisp outputs, when it comes to capturing intricate features of complex functions. Given that the model is parameterized, the system can be adapted easily to new data. The process is also very dynamic in the sense that, suppose we were later given additional training points, we can continue gradient descent from where we left off, and obtain a new set of parameters incrementally without substantial computational effort.

7 Conclusion

The key takeaway from the project is that by adding greater degree of freedom to the fuzzy sets while performing function approximation, we get better estimates given the same *number* of fuzzy input sets. Thus, we have shown that fuzzy learning is a novel and adaptable solution to function approximation over compact sets, and is competent with existing function approximation techniques.

References

1. J. A. Dickerson and B. Kosko. Fuzzy function approximation with ellipsoidal rules. *Trans. Sys. Man Cyber. Part B*, 26(4):542–560, August 1996.
2. T. Hangelbroek and A. Ron. Nonlinear Approximation Using Gaussian Kernels. *ArXiv e-prints*, November 2009.
3. Vojislav Kecman. *Learning and Soft Computing: Support Vector Machines, Neural Networks, and Fuzzy Logic Models.* MIT Press, Cambridge, MA, USA, 2001.
4. L.-X. Wang and J.M. Mendel. Generating fuzzy rules by learning from examples. *Systems, Man and Cybernetics, IEEE Transactions on*, 22(6):1414–1427, Nov 1992.