Univariate Radial Basis Function Layers: Brain-inspired Deep Neural Layers for Low-Dimensional Inputs

Basavasagar Patil * INRIA Grenoble, LJK, UGA
Xavier Alameda-Pineda INRIA Grenoble, LJK, UGA
Chris Reinke INRIA Grenoble, LJK, UGA

Abstract

Deep Neural Networks (DNNs) became the standard tool for function approximation with most of the introduced architectures being developed for high-dimensional input data. However, many real-world problems have low-dimensional inputs for which standard Multi-Layer Perceptrons (MLPs) are the default choice. An investigation into specialized architectures is missing. We propose a novel DNN layer called Univariate Radial Basis Function (U-RBF) layer as an alternative. Similar to sensory neurons in the brain, the U-RBF layer processes each individual input dimension with a population of neurons whose activations depend on different preferred input values. We verify its effectiveness compared to MLPs in low-dimensional function regressions and reinforcement learning tasks. The results show that the U-RBF is especially advantageous when the target function becomes complex and difficult to approximate.

1 Introduction

Neural networks became the fundamental tool for function approximation solving a wide range of problems, from natural language processing [1] to computer vision [2] and control [3]. Although a multi-layer perceptron (MLP) with a single large hidden layer can theoretically approximate any function [4], training such simple models even with several layers is often not successful. Different architectures have been investigated allowing the training of networks often depending on their input data type. Existing approaches were mainly developed for high-dimensional inputs such as Convolutional Neural Networks (CNNs) for images [5, 6] or Transformers for sequential data such as sentences [7].

However, many important datasets and tasks are not high-dimensional. Financial datasets for classification or regression such as fraud detection [8] usually have low-dimensional inputs. For example, the credit approval dataset for classification of the UCI Machine Learning Repository [9] consists only of 15 attributes (continuous and binary). Also, several real-world control tasks are low-dimensional such as in the field of energy systems [10, 11]. For example, the task of controlling the battery usage in hybrid cars can depend only on a three-dimensional continuous input [12].

So far, not many research efforts have been devoted to specialized deep neural architectures for low-dimensional input data, and standard MLPs are generally used by default. In this paper, we investigate a variant of Radial Basis Function (RBF) networks [13], which we name Univariate-RBF (U-RBF) layers, with the aim to improve trainability for low-dimensional input data. Our approach is inspired by the population coding and the tuning curve stimulus encoding theory from neuroscience [14]. According to these theories, some neurons encode low-dimensional continuous stimuli by having a bell-shaped tuning curve where their activity peaks at a preferred stimuli value.

*corresponding author email: sagarbasava8@gmail.com
Code is available at `https://github.com/bkpcoding/urbf`

Several neurons (or groups of neurons) encode a single stimulus dimension by having their preferred values span over the stimulus value range (Fig. 1). One such example is the processing of movement directions in the medial temporal visual area [15] where different populations of neurons are responsive to different directions of movement. Similar to their neuroscientific inspiration a unit in a U-RBF layer encodes a single continuous input ($x \in \mathbb{R}$) with several neurons ($y \in \mathbb{R}^n$) each having a Gaussian activation function peaking at a given input value μ with spread σ. The preferred input value (μ) is different for each of the n U-RBF neurons (and possibly their spread too). These two parameters are adaptable per neuron using backpropagation and gradient descent, recalling the adaptability of monkeys' sensory neurons when trained on specific tasks [16].

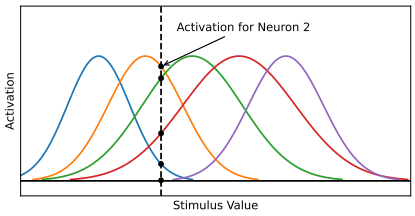


Fig. 1: Several neurons encode a single continuous input by having Gaussian-shaped activity curves that peak at different receptive values.

Our contributions in this paper are two-fold:

- First, we introduce a new neural network layer, called U-RBF, for function approximation with low-dimensional inputs and prove its universal function approximation capability.
- Second, we show the efficiency of the U-RBF layer in two function approximation domains: supervised learning and reinforcement learning.

2 Background & Related Work

2.1 Function Approximation

Function approximation aims to represent potentially complex and unknown functions $y = f(x)$ ($x \in \mathbb{R}^n$, $y \in \mathbb{R}^m$) by a model $y = \tilde{f}_\theta(x)$. Models are often based on parameters $\theta \in \mathbb{R}^d$. Many classical methods have been investigated to solve this problem ranging from linear regression [17], polynomial regression [18], random forests [19], AdaBoost [20], to support vector machines [21]. However, these methods often struggle with high-dimensional input data such as images. For these cases, deep neural networks (DNNs) [22] such as CNNs [5, 6] started to outperform these methods by allowing their training with large amounts of parameters through GPUs. As a result, DNNs became the research and industry standard for high-dimensional data. On the other hand, for low-dimensional data, DNNs often reach only a similar performance to traditional methods [23–25]. However, the relative ease with which neural networks can be adapted to different tasks and the abundance of tools for their development make them an interesting choice also for low-dimensional problems. Increasing their efficacy for such problems is therefore beneficial to the larger machine learning community.

2.2 RBF Networks

RBF networks have been used for function approximation and provide the basis for the proposed U-RBF layer. RBF networks are feedforward networks with a hidden layer. For an input $\boldsymbol{x} \in \mathbb{R}^n$, their j^{th} output is given by:

$$f_j(\boldsymbol{x}) = \sum_{k=1}^{K} w_{jk} \mathcal{G}(\|\boldsymbol{x} - \boldsymbol{c}_k\|, \sigma_k), \qquad \text{for } j = 1, 2, \ldots, J, \tag{1}$$

where $\mathcal{G}(u, v) = \exp(-u^2/2v^2)$ is a Gaussian kernel, J represents the number of outputs, K represents the number of Gaussian kernels in the layer, and the w_{jk}'s are scalar weights. The centre and the standard deviation of the k^{th} RBF kernel are represented as $\boldsymbol{c}_k \in \mathbb{R}^D$ and $\sigma_k > 0$.

RBF networks were introduced as adaptive networks for multivariable functional interpolation [13] and have seen widespread adaptation since Park and Sandberg [26, 27] proved that they are universal function approximators, i.e. an RBF network with one hidden layer and certain assumptions regarding their kernel functions is capable of approximating any function to a required amount of accuracy. The kernel function plays an important role, and several kernel functions have been proposed [28]. However, the Gaussian kernel is the most widely used. An important characteristic of a Gaussian

kernel is that its activation decreases monotonically with the distance from its centre point $\boldsymbol{c}_k$, with the decrease in the magnitude controlled by its spread σ_k, often termed as width. This property allows them to act as local approximators.

Considerable research has been dedicated to the selection of centres for Gaussian kernels [29–33]. A naive approach is to chooses the centres of RBF randomly, suggesting that it helps with regularization [34]. However, such approaches are unsustainable for large datasets. Meyer et al. [35] developed a widely used method to choose RBF centres by using clustering algorithms to determine the cluster patterns in the input and select a sample from each cluster as a centre for RBF neurons. In our work, we treat the centres as parameters and learn them using gradient updates.

Different variants of RBF networks have seen wide use in supervised learning tasks, involving application areas such as biology/medicine [36–38] and control systems [39, 40]. One of the more closely related works to our work is from Jiang et al. [41]. They propose a multilayer RBF-MLP network in which there are alternating RBF layers in between MLP layers. However, they do not compare their network performance on reinforcement learning tasks nor do they study the effects of the proposed networks in conjunction with CNNs or other network architectures. Reinforcement learning which aims to solve multistep sequential decision-making problems is one of the applications that we explore in our paper. However, the only application of Radial Basis Networks in Reinforcement Learning so far is by Asadi et al. [42] and Barreto et al. [43]. In the former, the authors use the RBF network as an output layer for continuous control using the DQN algorithm. In the latter, the authors introduce a variant of the gradient descent algorithm called the Restricted Gradient Descent (RGD) algorithm, in which they adapt the width of the RBF units and also add new units based on the approximation error in the value function. But as far as we know, there has been no previous work exploring the particular architecture proposed by us in reinforcement learning. While most of the existing work requires significant modification of either the training algorithm or the network architecture, we aim to provide a new modular layer, which can be plugged into any suitable regression or reinforcement learning tasks.

3 Univariate Radial Basis Function (U-RBF) Layer

This section introduces the uni-variate layer (Fig. 2) which is derived from RBF networks (1). From here on, the classical RBF network structure will be referred to as M-RBF layer, short for Multivariate-Radial Basis Function layer, as it uses multivariate Gaussian kernels. In contrast, the uni-variate layer uses 1-dimensional Gaussian kernels.

The U-RBF layer consists of D RBF units, one for each dimension of the input $\boldsymbol{x} \in \mathbb{R}^D$. The individual input dimensions x_d are remapped into a higher-dimensional representation using K_d 1-dimensional Gaussian kernels per RBF unit. The layers output dimension is $J = \sum_{d=1}^{D} K_d$, where $J = KD$ if the number of kernels is the same for all RBF units. This high-dimensional representation

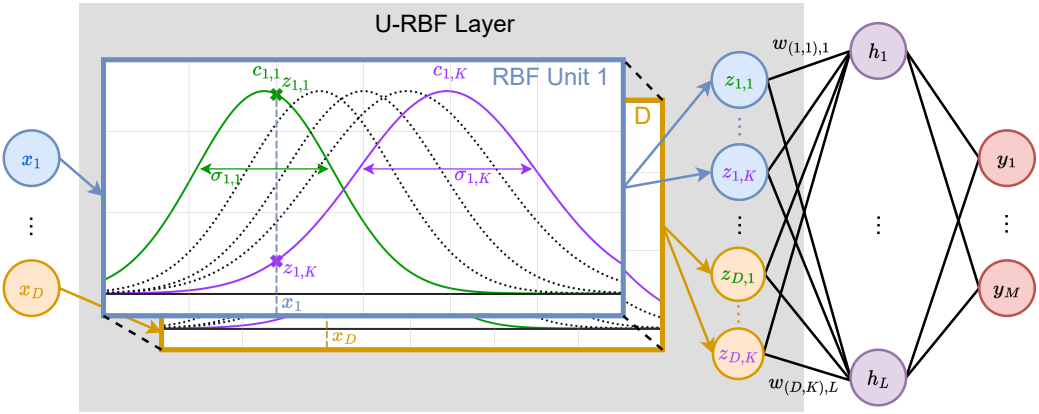


Fig. 2: The U-RBF layer.

is fed into a fully connected layer with L output neurons, indexed by l:

$$h_l(\boldsymbol{x}) = \sum_{d=1}^{D} \sum_{k=1}^{K_d} w_{(d,k),l}\, z_{d,k}, \quad \text{with} \quad z_{d,k} = \mathcal{G}(x_d - c_{d,k}, \sigma_{d,k})\,, \tag{2}$$

where the high-dimensional representation is denoted by $z_{d,k}$ with a double index, so as to map it to each of the D input dimensions (RBF units) and K_d Gaussian kernels. The centers $c_{d,k}$ (and potentially the spreads $\sigma_{d,k}$) per RBF unit are different so that their activation differs over the values inside the value range of their input dimension x_d. The centers and spreads can be either defined or learned by gradient descent with backpropagation together with the weights $w_{(d,k),l}$. The number of Gaussian kernels per input (RBF unit) is a hyperparameter named as Number of Neurons Per Input (NNPI). In summary, the U-RBF layer projects each dimension of an input to a higher dimensional space via its RBF units, where the projection depends on the centres and spreads of its Gaussian kernels.

3.1 Universal Function Approximation

A U-RBF layer that is followed by a hidden layer is a Universal Function Approximator, i.e. it has the capacity to accurately approximate any complex function to an arbitrary degree of precision. More formally:

Theorem. *Let $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$ be a set of distinct points in $\mathbb{R}^D$ and let $f : \mathbb{R}^D \to \mathbb{R}^L$ be any arbitrary function. Then there is a function $g : \mathbb{R}^D \to \mathbb{R}^L$ with U-RBF layer with $K \geq 2$ Gaussian kernels with different kernels, such that $f(\boldsymbol{x}) = g(\boldsymbol{x})$, for all $x \in \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$.*

The proof is given in Appendix A. The theorem provides the theoretical justification that U-RBF networks, similar to MLPs, have the representation power to capture any functional relationship, given sufficient parameters. This is a fundamental requirement for the network to be applicable across different tasks and problem domains.

4 Experiments

We evaluated the U-RBF layer on two key problem domains: function regression and reinforcement learning. The experiments aim to validate the effectiveness of U-RBF networks compared to baseline MLPs and traditional RBF networks across different conditions with comparable model sizes. We systematically vary two key factors that impact learning performance:

1. Complexity of the target function or environment.
2. Dimensionality of the input space.

The experiments show that the U-RBF layer has an improved performance as complexity increases, especially for low-dimensional inputs.

4.1 Function Regression Tasks

Function regression [44] aims to model the relationship between dependent variables (here $z \in \mathbb{R}$) and independent variables (here $x, y \in [-5, 5]$) that are continuous: $z = f(x, y)$. A DNN model $\tilde{f}_\theta$ with parameters θ approximates this relationship and is trained on the Mean Squared Error Loss (MSE):

$$\mathcal{L}(\theta) = \left(\tilde{f}_\theta(x, y) - f(x, y)\right)^2.$$

The U-RBF has been compared to MLPs and M-RBFs on two types of functions that model disturbances on a smooth xy-plane with increasing complexity.

Tasks & Procedure

The first type of function adds Gaussian-like curves with varying positions and variances on a linear plane in $x + y$ (Fig. 3a):

$$f_{\text{gauss}}(x, y) = x + y + \sum_{i=1}^{M} 3 \left[\exp\left(-\frac{(x - \mu_{i,x})^2}{2\sigma_i^2}\right) \cdot \exp\left(-\frac{(y - \mu_{i,y})^2}{2\sigma_i^2}\right)\right].$$

The number of Gaussian components ($M = \{0, 1, 3, 5\}$) determines the complexity level of the function.

The second function type evaluates how the networks handle discontinuities (Fig. 4a). Functions are composed of a linear plane ($x + y$) with random elevations added across different non-overlapping rectangular regions A_i:

$$f_{\text{disc}}(x, y) = x + y + \sum_{i=1}^{M} \mathbf{1}_{A_i}(x, y) h_i$$

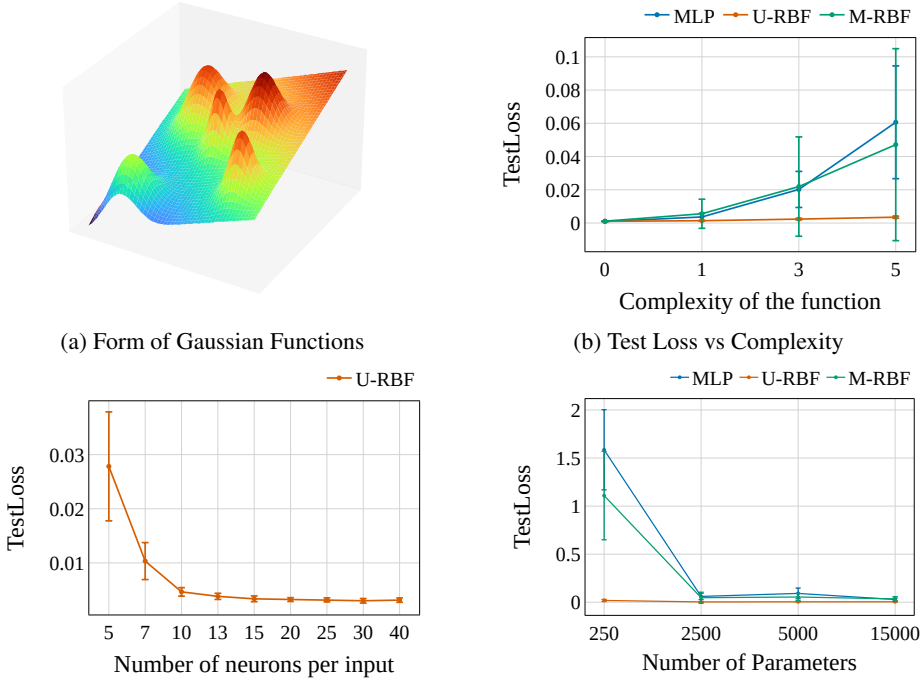
where $\mathbf{1}_{A_i}(x, y)$ is the indicator function being 1 if $(x, y) \in A_i$, otherwise 0 and h_i indicates a random height picked for the region. Similar to the Gaussian function, $M = \{0, 1, 3, 5\}$ is used to represent the number of discontinuities controlling the level of complexity. Details about the experimental procedure and the network architectures are provided in Appendix B.

Results

For the Gaussian functions, the U-RBF achieved on average a significantly lower test loss (Fig. 3b) than the MLP across the higher function complexities. This gap widened as more Gaussian components are added (Fig. 3c), demonstrating the U-RBF's advantage on functions with higher complexity. The M-RBF's performance was comparable to the MLP.

For the discontinuous functions, the U-RBF network reduced the test loss by approximately 50% compared to the MLP on the highest task complexity (Fig. 4b). The M-RBF performed poorly, with a test loss of 80% higher than the MLP.

The number of neurons per U-RBF input was a key factor, with performance plateauing at 20 neurons for the Gaussian functions and 40 for the discontinuous functions. However, the U-RBF achieved superior performance over MLPs even with fewer parameters (Fig. 3d, 4d), highlighting its parameter efficiency.



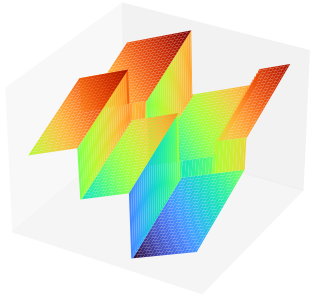
(a) Form of Gaussian Functions

(b) Test Loss vs Complexity

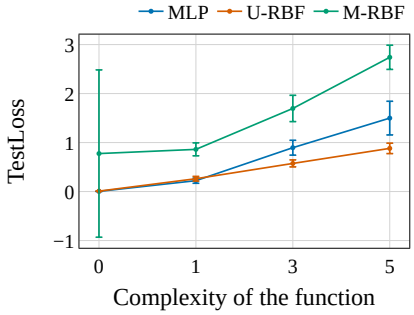
(c) Test Loss vs Number of Neurons

(d) Test Loss vs Number of Params

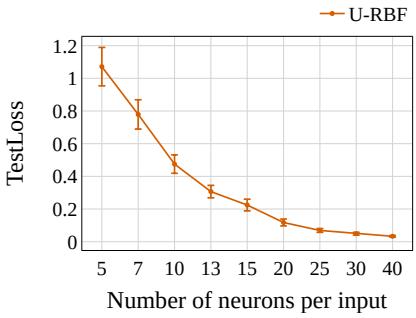
Fig. 3: For Gaussian functions (a) the U-RBF outperforms both MLP and M-RBF for functions of higher complexity (b). Its performance depends on the number of neurons per RBF unit requiring 20 for reaching its optimal performance. The U-RBF needs fewer network parameters than the MLP or M-RBF to approximate the function (d). Figures report the test loss with their mean and standard deviation over 30 repetitions.



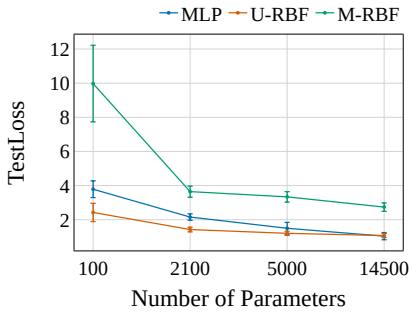
(a) Form of Discontinuous Functions



(b) Test Loss vs Complexity



(c) Test Loss vs Number of Neurons



(d) Test Loss vs Number of Params

Fig. 4: For discontinuous functions (a) the U-RBF outperforms both MLP and M-RBF for functions of higher complexity (b). Its performance depends on the number of neurons per RBF unit requiring 40 to reach its optimal performance. The U-RBF needs fewer network parameters than the MLP or M-RBF to approximate the function (d). Figures report the test loss with their mean and standard deviation over 30 repetitions.

In summary, the U-RBF significantly outperformed MLPs and M-RBF networks, especially as function complexity increased. The results align with the theoretical motivation for U-RBF layers being beneficial for complex low-dimensional data.

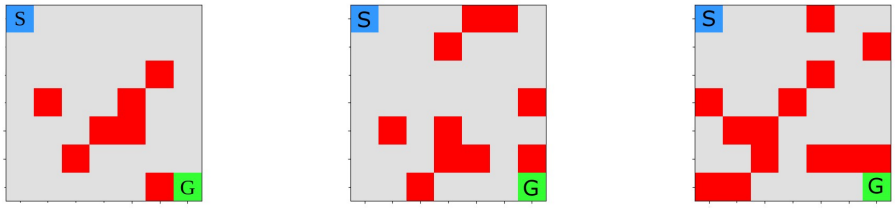
4.2 Reinforcement Learning Tasks

Reinforcement learning aims to solve multi-step decision problems [45] where an agent maximizes reward by interacting with its environment. RL problems are modeled as Markov Decision Processes (MDPs): $\mathcal{M} = (\mathcal{S}, \mathcal{A}, p, r, \gamma)$. $\mathcal{S}$ and $\mathcal{A}$ represent a set of states and actions. An agent interacting with the environment at time t, transitions from a state $s_t \in \mathcal{S}$ to $s_{t+1} \in \mathcal{S}$ by taking an action $a_t \in \mathcal{A}$, with the probability defined by the transition function $p(s_{t+1}|s_t, a_t)$. The agent's objective is to find a policy $\pi : \mathcal{S} \to \mathcal{A}$ that maximizes the expected return $G_t \equiv \sum_{i=0}^{\infty} \gamma^i r(s_{t+i}, a_{t+i}, s_{t+1+i})$, where $\gamma \in [0, 1)$ is the discount factor. One way to achieve this is through the use of a Q-value function, $Q^\pi(s, a) \equiv \mathbb{E}^\pi [G_t \mid s_t, a_t]$, where $\mathbb{E}^\pi [.]$ denotes the expectations over the transitions induced by π. The Q-value function allows the agent to choose the optimal action under the current policy, by choosing the action with the highest Q-value: $\pi(s) = \text{argmax}_a Q^\pi(s, a)$.

Deep Q-learning (DQN) is an algorithm combining deep learning and reinforcement learning [46] which approximates the optimal Q-function with a DNN. The DQN uses a replay buffer to store the agent's experience from each timestep, i.e. the observation (s_t, a_t, r_t, s_{t+1}) with the state, taken action, received reward, and the next state. The buffer allows the agent to sample a batch of experiences from the past to train the Q-approximation on according to the following loss:

$$\mathcal{L}(\theta) = \left(r_t + \gamma \max_{a'} \tilde{Q}_{\bar{\theta}}(s_{t+1}, a') - \tilde{Q}_\theta(s_t, a_t)\right)^2,$$

where $\tilde{Q}_{\bar{\theta}}$ is the approximated Q-function with frozen parameters that are not influenced by back-propagation.



(a) Level 1 (b) Level 2 (c) Level 3

Fig. 5: Examples of the maze task for the different complexity levels with 7, 10, and 13 pits (red). The agent starts in the blue cell and has to reach its goal in the green cell.

Tasks & Procedure

We generated maze environments with variable complexities (Fig. 5). All mazes are 9x9 gridworlds. The agent obtained +100 reward for reaching the goal and -100 if it fell into a pit. No intermediate rewards were given for steps and episodes ended after 100 steps. The mazes differed in the following aspects:

- Complexity: The number of obstacle pits has been 7, 10, or 13 for low, medium, or high complexity.
- State representations: Coordinates (x,y coordinates of the agent), Matrix encoding (a $9x9$ matrix representing empty cells, pits, agent location, and goal positions with numbers between 0 and 4, flattened to a vector of size 81), and Image pixels were tested.

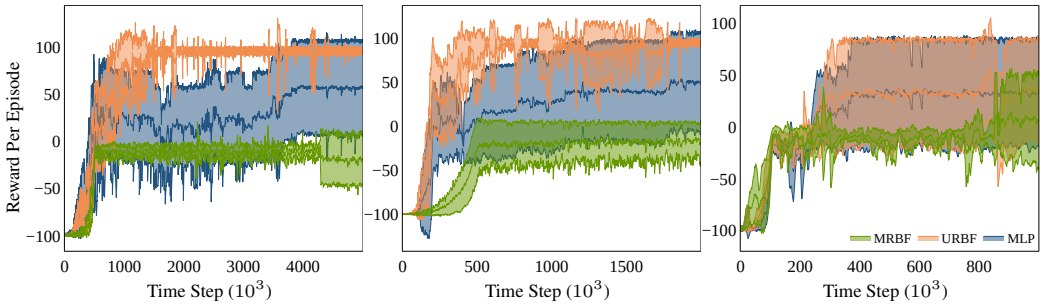
Architectures

DQN was used to learn the tasks using a DNN to approximate the Q-function. First the DNN encodes the state input into a low-dimensional latent representation. Afterwards, the Q-values are approximated based on this representation using either a MLP, M-RBF, or U-RBF network. For the Coordinate and Matrix inputs the input is encoded by 3-4 fully connected layers. For the Image input, several CNN layers compute the latent representation. The number of latent dimensions ("Lat" in Table 1) was varied between 8, 16, and 32 units to analyze the impact of the bottleneck capacity on the performance of the different network architectures. A detailed description of the procedure and architectures is provided in Appendix B.

Table 1: Maze Results

		Coordinates Input			**Matrix Input**		
#Lat	Algo	Level 1	Level 2	Level 3	Level 1	Level 2	Level 3
	MLP	4.38 (2.01)	2.36 (2.21)	1.45 (2.04)	**8.13 (0.17)**	6.26 (3.2)	4.62 (4.00)
16	M-RBF	-6.94 (10.62)	-8.85 (21.31)	-4.67 (6.83)	0.07 (0.1)	-0.29 (0.45)	-0.18 (0.24)
	U-RBF	**6.67 (0.20)**	**6.41 (0.28)**	**4.83 (0.60)**	7.80 (0.36)	**8.00 (0.16)**	**6.82 (2.34)**
	MLP	4.98 (1.93)	4.50 (1.73)	2.28 (2.38)	**7.94 (0.37)**	7.31 (1.97)	6.02 (3.15)
32	M-RBF	-0.87(3.05)	-1.19(1.94)	-1.73(3.58)	0.1(0.13)	-0.20 (0.27)	-0.08 (0.11)
	U-RBF	**6.45 (0.64)**	**6.01 (0.49)**	**4.60 (1.79)**	7.02 (1.87)	**7.64 (0.37)**	**7.69 (0.40)**
	MLP	4.79 (1.71)	3.85 (2.26)	1.97 (1.83)	**7.83 (0.34)**	7.08 (2.50)	6.28 (3.19)
64	M-RBF	-1.01(2.55)	-1.39(2.56)	-0.83(0.84)	-0.08(0.15)	-0.27 (0.42)	-0.12 (0.10)
	U-RBF	**6.73 (0.18)**	**6.12 (0.54)**	**4.07 (1.89)**	7.83 (0.38)	**7.76 (0.26)**	**6.92 (2.17)**

		Image Input (reward $\times 10^1$)		
#Lat	Algo	Level 1	Level 2	Level 3
	MLP	-0.07 (0.09)	0.00 (0.07)	**-0.07 (0.13)**
8	M-RBF	**2.08 (2.82)**	**0.48 (1.38)**	-0.23 (0.43)
	U-RBF	0.18 (1.70)	-0.07 (0.09)	-0.27 (0.13)
	MLP	-0.04 (0.09)	1.09 (1.26)	-0.04 (0.09)
16	M-RBF	0.09 (0.45)	-0.14 (0.10)	-0.26 (0.15)
	U-RBF	**0.20 (0.40)**	**1.78 (3.61)**	**1.67 (3.61)**
	MLP	1.40 (2.91)	0.00 (0.00)	-0.16 (0.09)
32	M-RBF	-0.12 (0.04)	-0.12 (0.07)	-0.24 (0.13)
	U-RBF	**3.44 (4.01)**	**1.62 (3.58)**	**-0.07 (0.09)**



(a) Coordinates (b) Matrix (c) Image

Fig. 6: The URBF learns faster and shows less variance than the MLP or MRBF (best performing networks across the architectures).The curves provide the mean and standard deviation of the average reward per episode (10 runs, 5 in the case of Image input).

Results

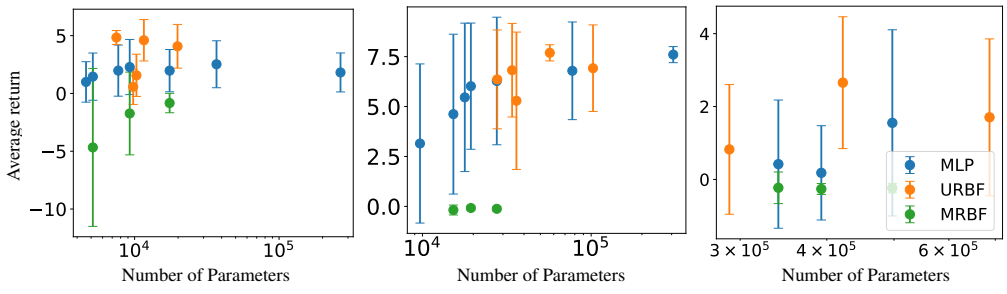
The networks are compared based on their learning performance in rewards per episode over training steps (Fig. 6) and the average reward per timestep over the entire training period (Table 1). This metric combines the notions of how well an agent learned to solve the environment and how fast it learned. Generally, the U-RBF outperforms the MLP and M-RBF in most conditions. For a low number of input dimensions (Coordinates), the U-RBF outperforms the MLP for all complexity levels. For the intermediate number of dimensions (Matrix), the MLP is able to outperform the U-RBF only on the lowest complexity level. For higher complexities (Levels 2 and 3), the U-RBF has the best performance. For Coordinates and Matrix inputs, the MLP struggles to reach the goal in almost half of the repetitions, whereas the U-RBF reaches the goal in all repetitions. In the case of the Image input, the difference between the U-RBF and MLPs is less strong. The M-RBF generally performs the worst except for the lower levels of the Image input with a small number of latents.

In terms of the number of parameters in comparison to the average return (Fig. 7), several MLP networks have fewer parameters than the U-RBF but also a lower performance. For networks that have a smiliar number of parameters, the U-RBF reaches generally a better performance over all conditions.

5 Discussion

5.1 When does the U-RBF improve performance over MLPs?

For both the function regression and the reinforcement learning tasks, the U-RBF provides an improvement over the traditional M-RBF. Compared to the MLP, there are three important factors



(a) Coordinates (b) Matrix (c) Image

Fig. 7: The U-RBF outperforms the MLP and M-RBF with a similar or a lower number of parameters in Coordinates and Image input spaces. The bars represent the mean and standard deviations of average reward per timestep (10 runs, 5 in the case of Image input) of all evaluated architecture variations per network type.

that determine when the U-RBF outperforms it.

Dimensionality of the input The maze task showed (Fig. 6) that the U-RBF is most appropriate for low-dimensional continuous inputs. It outperforms the MLP clearly for lower dimensional input conditions (Coordinates and Matrix) besides for the lowest complexity (Level 1) of Matrix inputs. Whereas, for higher dimensional inputs (Image) the difference between the MLP and U-RBF becomes less clear.

Target function complexity The U-RBF improves performance for higher complexities. For function regression, the MLP and U-RBF reach similar performances for the lowest complexity (Fig. 3b, 4b). With increasing complexity, the U-RBF outperforms the MLP. The maze task shows a similar trend where the U-RBF outperforms the MLP more consistently in tasks with higher complexity (Tbl. 1).

Number of network parameters Generally, the more network parameters (weights) the MLP or U-RBF have the better their performance is. The MLP is often able to reach the same performance as the U-RBF (Fig. 3a, 4a), but it requires much more parameters increasing its memory demand and computational load for training and inference.

5.2 What is the influence of the U-RBF hyperparameters?

The U-RBF has two hyperparameter settings.

Number of neurons per RBF unit A minimum number of neurons K_d are required to cover the entire domain of the input space and to represent their values in a distinguishable manner. Thus an insufficient number of neurons can contribute to poor performance of the network. However, once the required number of neurons is found the performance is plateauing for more neurons (Fig. 3c).

Initial centers and spreads The initial settings of c and σ seem less important as they are adapted during the training. Nonetheless, future research should investigate if different random initialization methods might have an influence on learning performance.

5.3 Why does the U-RBF improve performance?

Similar to other deep learning architectures the theory behind U-RBFs is still lacking. At the moment we can only provide hypotheses why the U-RBF improves the performance over MLPs under certain conditions. Similar to SVMs and other kernel machines [47], the U-RBF projects its input in a higher dimensional space. They aim to unfold the non-linearity and thus be able to learn easier in the higher-dimensional space. The U-RBF neurons also provide a clustering method where the RBF neurons represent clusters for the input values to which they are most responsive to. Having clusters might help with the downstream learning task. However, a further investigation into the theoretical background behind U-RBFs is necessary.

6 Conclusion

This paper introduced and analyzed a novel Univariate Radial Basis Function layer for neural networks applied to low-dimensional continuous inputs. The theoretical proofs and experimental results on function approximation and reinforcement learning problems demonstrated the benefits of specializing model architectures over conventional MLPs for such data.

Limitations of this initial investigation include the need for more rigorous ablation studies, evaluation on real-world benchmarks, and elucidation of the specific mechanisms behind U-RBF's superior performance. There remains great opportunity for developing specialized deep learning architectures for small input spaces. By spurring further research, we hope this work catalyzes continued innovation in this important area.

Bibliography

[1] OpenAI, "Gpt-4 technical report," 2023.

[2] X. Zhai, A. Kolesnikov, N. Houlsby, and L. Beyer, "Scaling vision transformers," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12104–12113, 2022.

[3] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, *et al.*, "Mastering atari, go, chess and shogi by planning with a learned model," *Nature*, vol. 588, no. 7839, pp. 604–609, 2020.

[4] K. Hornik, M. B. Stinchcombe, and H. L. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, pp. 359–366, 1989.

[5] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[6] D. C. Ciresan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber, "Flexible, high performance convolutional neural networks for image classification," in *Twenty-second international joint conference on artificial intelligence*, Citeseer, 2011.

[7] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[8] K. G. Al-Hashedi and P. Magalingam, "Financial fraud detection applying data mining techniques: A comprehensive review from 2009 to 2019," *Computer Science Review*, vol. 40, p. 100402, 2021.

[9] A. U. Asuncion, "Uci machine learning repository, university of california, irvine, school of information and computer sciences," in *UCI*, 2007.

[10] Z. Zhang, D. Zhang, and R. C. Qiu, "Deep reinforcement learning for power system applications: An overview," *CSEE Journal of Power and Energy Systems*, vol. 6, no. 1, pp. 213–225, 2019.

[11] J. R. Vázquez-Canteli and Z. Nagy, "Reinforcement learning for demand response: A review of algorithms and modeling techniques," *Applied energy*, vol. 235, pp. 1072–1089, 2019.

[12] X. Qi, Y. Luo, G. Wu, K. Boriboonsomsin, and M. Barth, "Deep reinforcement learning enabled self-learning control for energy efficient driving," *Transportation Research Part C: Emerging Technologies*, vol. 99, pp. 67–81, 2019.

[13] D. S. Broomhead and D. Lowe, "Radial basis functions, multi-variable functional interpolation and adaptive networks," 1988.

[14] P. Dayan and L. F. Abbott, *Theoretical neuroscience: computational and mathematical modeling of neural systems*. MIT press, 2005.

[15] J. H. Maunsell and D. C. Van Essen, "Functional properties of neurons in middle temporal visual area of the macaque monkey. i. selectivity for stimulus direction, speed, and orientation," *Journal of neurophysiology*, vol. 49, no. 5, pp. 1127–1147, 1983.

[16] A. Schoups, R. Vogels, N. Qian, and G. Orban, "Practising orientation identification improves orientation coding in v1 neurons," *Nature*, vol. 412, no. 6846, pp. 549–553, 2001.

[17] S. F. Buck, "A method of estimation of missing values in multivariate data suitable for use with an electronic computer," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 22, no. 2, pp. 302–306, 1960.

[18] S. M. Stigler, "Gergonne's 1815 paper on the design and analysis of polynomial regression experiments," *Historia Mathematica*, vol. 1, pp. 431–439, 1974.

[19] L. Breiman, "Random forests," *Machine Learning*, vol. 45, pp. 5–32, 2001.

[20] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," in *European Conference on Computational Learning Theory*, 1997.

[21] H. Drucker, C. J. C. Burges, L. Kaufman, A. Smola, and V. N. Vapnik, "Support vector regression machines," in *NIPS*, 1996.

[22] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[23] A. Korotcov, V. Tkachenko, D. P. Russo, and S. Ekins, "Comparison of deep learning with multiple machine learning methods and metrics using diverse drug discovery datasets," in *Molecular Pharmaceutics 2017*, 2017.

[24] M.-C. Lee and T. Chang, "Comparison of support vector machine and back propagation neural network in evaluating the enterprise financial distress," *ArXiv*, vol. abs/1007.5133, 2010.
[25] S. Jiao, Y. Gao, J. Feng, T. Lei, and X. Yuan, "Does deep learning always outperform simple linear regression in optical imaging?," *Optics express*, vol. 28 3, pp. 3717–3731, 2019.
[26] J. Park and I. W. Sandberg, "Universal approximation using radial-basis-function networks," *Neural Computation*, vol. 3, pp. 246–257, 1991.
[27] J. Park and I. W. Sandberg, "Approximation and radial-basis-function networks," *Neural Computation*, vol. 5, pp. 305–316, 1993.
[28] G. A. Montazer, D. Giveki, M. Karami, and H. Rastegar, "Radial basis function neural networks : A review," *Computer Reviews Journal*, no. 1, pp. 52–74, 2018.
[29] Q. Gan, R. Subramanian, N. Sundararajan, and P. Saratchandran, "Design for centres of rbf neural networks for fast time-varying channel equalisation," *Electronics Letters*, vol. 32, pp. 2333–2334, 1996.
[30] R. Perfetti and E. Ricci, "Reduced complexity rbf classifiers with support vector centres and dynamic decay adjustment," *Neurocomputing*, vol. 69, pp. 2446–2450, 2006.
[31] K. Warwick, J. D. Mason, and E. L. Sutanto, "Centre selection for radial basis function networks," in *International Conference on Adaptive and Natural Computing Algorithms*, 1995.
[32] B. Burdsall and C. G. Giraud-Carrier, "Ga-rbf: A self-optimising rbf network," in *International Conference on Adaptive and Natural Computing Algorithms*, 1997.
[33] R. B. Bhatt and M. Gopal, "On the structure and initial parameter identification of gaussian rbf networks," *International journal of neural systems*, vol. 14 6, pp. 373–80, 2004.
[34] M. Kubat, "Neural networks: a comprehensive foundation by simon haykin, macmillan, 1994, isbn 0-02-352781-7.," *The Knowledge Engineering Review*, vol. 13, no. 4, pp. 409–412, 1999.
[35] B. J. Meyer, B. Harwood, and T. Drummond, "Nearest neighbour radial basis function solvers for deep neural networks," *ArXiv*, vol. abs/1705.09780, 2017.
[36] P. Venkatesan and S. Anitha, "Application of a radial basis function neural network for diagnosis of diabetes mellitus," *Current Science*, vol. 91, pp. 1195–1199, 2006.
[37] S. Liu, C. Li, Z. Lu, Y. Li, and Q. Lai, "A memristive rbf neural network and its application in unsupervised medical image segmentation," *The European Physical Journal Special Topics*, pp. 1–10, 2022.
[38] B. Liu, B. Tan, L. Huang, J. Wei, X. Mo, J. Zheng, and H. Luo, "Intelligent algorithm-based picture archiving and communication system of mri images and radiology information system-based medical informatization," *Contrast Media & Molecular Imaging*, vol. 2021, 2021.
[39] Z. Gu, H. Li, Y. Sun, and Y. Chen, "Fuzzy radius basis function neural network based vector control of permanent magnet synchronous motor," *2008 IEEE International Conference on Mechatronics and Automation*, pp. 224–229, 2008.
[40] K. K. Tan and K. Z. Tang, "Adaptive online correction and interpolation of quadrature encoder signals using radial basis functions," *IEEE Trans. Control. Syst. Technol.*, vol. 13, pp. 370–377, 2005.
[41] Q. Jiang, L. Zhu, C. Shu, and V. Sekar, "An efficient multilayer rbf neural network and its application to regression problems," *Neural Computing and Applications*, vol. 34, pp. 4133 – 4150, 2021.
[42] K. Asadi, R. E. Parr, G. D. Konidaris, and M. L. Littman, "Deep rbf value functions for continuous control," *ArXiv*, vol. abs/2002.01883, 2020.
[43] A. Barreto and C. W. Anderson, "Restricted gradient-descent algorithm for value-function approximation in reinforcement learning," *Artif. Intell.*, vol. 172, pp. 454–482, 2008.
[44] S. Lathuilière, P. Mesejo, X. Alameda-Pineda, and R. Horaud, "A comprehensive analysis of deep regression," *IEEE transactions on pattern analysis and machine intelligence*, vol. 42, no. 9, pp. 2065–2081, 2019.
[45] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.
[46] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *ArXiv*, vol. abs/1312.5602, 2013.
[47] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proceedings of the fifth annual workshop on Computational learning theory*, pp. 144–152, 1992.
[48] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014.
[49] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *International Conference on Machine Learning*, 2010.

[50] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2015.

Appendix A Proposition and Theorems

A critical theoretical foundation for the proposed U-RBF network architecture is establishing its universal function approximation capability. This section provides the formal analysis to prove U-RBF networks can approximate any continuous function over the input domain. We first define the function classes used to represent single hidden layer feedforward neural networks (Definitions 1 and 2). Theorem 1, adapted from [4], establishes the universal approximation property for this network class. We then prove a one-to-one mapping between the input space and U-RBF layer output space in Corollary 1. Finally, Theorem A leverages these results to formally demonstrate that augmenting the single hidden layer networks with a U-RBF layer maintains the universal function approximation property.

Definition 1. *For $J \in \mathbb{N}$, $\mathcal{A}^J$ is defined as the set of affine functions from $\mathbb{R}^J$ to $\mathbb{R}$, where an affine function is of the form $A(\boldsymbol{z}) = \boldsymbol{w}^\top \boldsymbol{z} + b$, $\boldsymbol{w}, \boldsymbol{z} \in \mathbb{R}^J$ and $b \in \mathbb{R}$.*

In the context of feedforward networks or MLPs, the affine function is the linear transformation of the input x using a weight vector plus an offset or bias. A non-linear transformation following an affine function is widely used in machine learning models. Thus we introduce one such class of functions in the next definition based on our current definition of affine functions.

Definition 2. *For any measurable function $\rho(\cdot)$ from $\mathbb{R}$ to $\mathbb{R}$ and $J \in \mathbb{N}$ let $\mathcal{B}^J(\rho)$ be the set of functions defined as:*

$$\mathcal{B}^J(\rho) = \left\{ f : \mathbb{R}^J \to \mathbb{R},\ f(\boldsymbol{z}) = \sum_{q=1}^{Q} \beta_q \rho\left(A_q(\boldsymbol{z})\right), \beta_q \in \mathbb{R}, A_q \in \mathcal{A}^J, Q \in \mathbb{N} \right\}.$$

This class of functions is used to represent single-layer feed-forward networks with input space $\mathbb{R}^J$ and output space $\mathbb{R}$. The β_q's represent scalar weights from the hidden layer to the output layer. Further from now, we will use this class of functions to represent our single-layered single-output feed-forward network.

Theorem 1. *Let $\{\boldsymbol{z}_1, \ldots, \boldsymbol{z}_N\}$ be a set of distinct points in $\mathbb{R}^J$ and let $g : \mathbb{R}^J \to \mathbb{R}$ be an arbitrary function. If ψ achieves 0 and 1 , then there is a function $f \in \mathcal{B}^J(\psi)$ with N hidden units such that $f\left(\boldsymbol{z}_i\right) = g\left(\boldsymbol{z}_n\right), n \in \{1, \ldots, N\}$.*

Theorem 1 taken from [4] proves the universal function approximation capability of feedforward neural networks. Although we only state the theorem for a single hidden layered network with a single output, the authors in the same paper generalize it to a multi-layer multi-output network. With the universal function approximation capability of a general feedforward network proven, we now move on to corollary 1 to prove one-to-one mapping from the input space to the intermediate space, which is the output of our U-RBF layer.

Corollary 1. *For any $K \geq 2$, $c_1, \ldots, c_K \in \mathbb{R}$ and $\sigma_1, \ldots, \sigma_K > 0$, the mapping $h : \mathbb{R} \to \mathbb{R}^K$ defined as:*

$$h(u) = \left(\exp\left(-\frac{(u-c_1)^2}{2\sigma_1^2}\right), \exp\left(-\frac{(u-c_2)^2}{2\sigma_2^2}\right), \ldots, \exp\left(-\frac{(u-c_K)^2}{2\sigma_K^2}\right) \right)$$

is a one-to-one correspondence for all $u \in \mathbb{R}$, if and only if $c_i \neq c_j$ for at least one pair of (i, j), $i \neq j$.

Proof. It is needed to prove that if $h(u) = h(v)$ then $u = v$. For the sake of simplicity and without loss of generality, let us assume $K = 2$. This implies for $h(u) = h(v)$:

$$\left(\exp\left(-\frac{(u-c_1)^2}{2\sigma_1^2}\right), \exp\left(-\frac{(u-c_2)^2}{2\sigma_2^2}\right) \right) = \left(\exp\left(-\frac{(v-c_1)^2}{2\sigma_1^2}\right), \exp\left(-\frac{(v-c_2)^2}{2\sigma_2^2}\right) \right)$$

Since $\exp(\cdot)$ is a monotonically increasing function we have:

$$(u - c_1)^2 = (v - c_1)^2 \quad \text{and} \quad (u - c_2)^2 = (v - c_2)^2. \tag{3}$$

Therefore, given that $c_1 \neq c_2$, the only possible solution is $u = v$. Thus, h is a one-to-one correspondence for all $K \geq 2$.

Corollary 1 proves one-to-one mapping from the input space to the higher dimensional space that U-RBF projects into. The corollary is essential for the universal approximation property proved in the next theorem, in the sense that it reflects a unique mapping from the input space to the output space for the U-RBF layer and thus provides the basis for further transformation in the following layers.

Theorem. *Let $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$ be a set of distinct points in $\mathbb{R}^D$ and let $f : \mathbb{R}^D \to \mathbb{R}^L$ be any arbitrary function. Then there is a function $g : \mathbb{R}^D \to \mathbb{R}^L$ with U-RBF layer with $K \geq 2$ Gaussian kernels with different kernels, such that $f(\boldsymbol{x}) = g(\boldsymbol{x})$, for all $x \in \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$.*

Proof. From Theorem 1 and Corollary 1, it is clear that if there is a one-to-one mapping from input space to a higher dimensional space using U-RBF, following which an MLP is applied to this space, then the network is able to approximate any function f. Although the theorem proves the universal approximation for a single output, a set of further tedious modifications in turn would make it applicable for multiple output networks with more than one hidden unit.

As it can be noticed, our approach mainly hinges on the point that there is no loss of information while projecting the input space into a higher dimensional space using the U-RBF layer. Using this along with the universal function approximation capability of a feedforward network, we conclude that in turn adding our proposed U-RBF layer to a feedforward network doesn't affect its universal function approximation capability.

Appendix B Experimental and Architectural Details

B.1 Experimental Details for Function Regression

The number of elevations $M = \{0, 1, 3, 5\}$ defines the function complexity. The level of elevation is chosen to be 3 for Gaussian functions, while for Discontinuous functions the elevation (h_i) is chosen randomly.

Each network was evaluated for 30 repetitions, each with randomly generated functions. For Gaussian functions with $\mu_{i,x/y} \sim \mathcal{U}[-5, 5]$, $\sigma_i \sim \mathcal{U}[0.4, 0.8]$ and for discontinues functions with $h_i \in \mathcal{U}[1, 10]$. The dataset for each repetition consists of 7500 sampled data points for the training set and 1000 for the test set. The training was run for 300 epochs with Adam [48] (learning rate: 0.0001) and a batch size of 256.

B.2 Architectural Details for Function Regression

We evaluated 4 different architectures for each network type. For the MLP the network depth varies from a single hidden layer to up to four hidden layers with the number of hidden units per hidden layer given in Table 2. For the U-RBF the architecture remains similar except that there is a U-RBF layer between the input and the first hidden layer, with the number of units in the layer decided by the Number of Neurons Per Input (NNPI). The value of NNPI is a hyperparameter and is optimized over the range of values $\{5, 7, 10, 13, 15, 20, 25, 30, 40\}$ to study its relative importance in the performance of the network.

Table 2: Architectures for Function Regression Experiments.

MLP	M-RBF	U-RBF
16	16	2*NNPI [a], 16
32, 64	32, 64	2*NNPI, 32, 64
32, 64, 128	32, 64, 128	2*NNPI, 32, 64, 128
32, 64, 128, 32	32, 64, 128, 32	2*NNPI, 32, 64, 128, 32

[a] Number of Neurons Per Input

The initial *ranges* for the U-RBF layer define the range over which the centres of the layer are spread equidistantly. For all experiments, the initial ranges are set to $[-5, +5]$. In the case of an RBF unit with 5 neurons, their centres will be at $\{-5, -2.5, 0, 2.5, 5\}$.

B.3 Experimental Details for Reinforcement Learning

Each DQN agent had a replay buffer of size $100,000$ with the learning starting after $30,000$ timesteps. The batch size was 64, with Adam [48] as the optimizer. The exploration was ϵ-greedy [45], with initial and final exploration rates being 1 and 0.02 and an exploration fraction (exploration rate is reduced per episode) of 0.1. Each agent and condition was evaluated on 10 repetitions (5 for *Image* inputs), each with a unique random seed.

B.4 Architectural Details for Reinforcement Learning

The architecture for the MLP network with Coordinates input and Matrix input is similar, with 3 layers of hidden units with the size of the first and the last hidden layer being 128 and the size of the middle layer either 16, 32, or 64. The architecture for the M-RBF is similar to the MLP but with an M-RBF layer replacing the middle layer with the same number of units. The architecture of the U-RBF consists of 4 layers with the initial two layers similar to the MLP but appended by an U-RBF layer after the second hidden layer. The last layer is a multilayer perceptron with the same number of neurons as MLP's last layer. All the networks use the ReLU [49] activation function. There is no activation function between the U-RBF and M-RBF layer and the layer after them.

The architecture for image inputs is a slightly modified version of the architecture in [50]. The inputs are RGB images with shape 84x84x3. The first layer convolves 32 filters of 8 x 8 with a stride of 4 before applying ReLU. The second layer convolves 64 filters of 4 x 4 with a stride of 2 followed by ReLU. The third layer convolves 64 filters of 3 x 3 with a stride of 1 followed by ReLU. The output from this layer is flattened. For the MLP it is then connected to a fully connected layer. For M-RBF's, it is connected to a M-RBF layer with a latent dimension $\in 8, 16, 32$. In the case of the U-RBF, an U-RBF layer is added at this point before connecting to the last layer. The final hidden layer has 512 hidden units. It is fully connected to the output layer with 4 neurons, one for each action.

The learning rate and ranges for U-RBF for all the experiments were $8e-4$ and [0, 8] respectively except for the case with an Image input, in which they were set to be $3e-4$ and [0, 12].