Solving Initial Value Problems for ODE and PDE using RBF-basis

Freek De Haas **Yukiya Saito** **Jessie Fu**

Abstract

Among the many ways to solve Initial Value Problems (IVP), the most popular methods such as Finite Element Method and Finite Difference Method suffer from the curse of dimensionality. We use an alternate mesh-free approach that elevates solvers from this constraint by approximating the solution using a linear combination of RBF-basis functions. We use this approach on ODE and PDE example problems, including the Schrodinger equation. To improve learning of the parameters, we implement a -never tried before- adaptive sampling technique based on Metropolis Hastings algorithm and conclude that this substantially improves the convergence rate.

1 Introduction

Partial differential equations (PDE's) are ubiquitous in applied and fundamental science. Second order PDE's, where the highest order derivative is a second order derivative, can model a wide variety of natural phenomena such as fluid dynamics (Navier Stokes equation), the state function of a quantum-mechanical system (Schrodinger equation) and a continuous time, continuous-state markov chain (backward- and/or forward Kolmogorov equation or Fokker Planck equation).

Some of these PDE's have an analytical solution but it is usually only under very restricted and not practical circumstances that this solution can be written down in a closed form (e.g. the Laplace equation on a rectangular domain). Most PDE's, however, cannot be solved analytically and so will have to be dealt with by numerical techniques such as Finite Element (FEM), Difference (FDM) and Volume Methods (FVM). These classical numerical techniques solve initial value problems (IVP) on the vertices of a domain-mesh which turns the problem into a linear algebra problem. The resulting approximation of the solution, although often accurate, is therefore discrete. Although these techniques are very accurate and quite reliable (especially FEM), they suffer the *'curse of dimensionality'*. As the dimensionality of the domain increases on which the PDE is to be solved, the number of mesh points grows exponentially ($O(e^d)$) which becomes rapidly computationally intractable.

An alternative way to deal with high dimensional PDE's is by decomposing the solution using a Radial Basis Function (RBF) network. This approach is mesh-free and lets the user compute an approximate solution as a sum of 'simple' functions. However, for complicated PDEs or the shapes of the functions, the computational cost can be very expensive. Therefore, in order to facilitate efficient training of the network, it is important to pick training points which contribute to the rapid minimization of the loss.

In this report, we explore different implementations and ways to fit an RBF network and use a never tried before adaptive sampling technique to more accurately fit the approximate function.

2 Related Works

Methods to solve IVP that do not rely on a discretisation scheme often take the approach of approximation by using simpler functions. Chen and Leng (2011) used an RBF network to approximate the highest order derivative in a given problem and implemented an adaptive training algorithm to do so (Chen et al. [2011]). Their adaptive training algorithm relied on a chosen critical threshold of the loss, which is used to either add or remove nodes from the network. Other methods for approximation that have been tried include a dense net (Berg and Nyström [2018]), a deep net (Sirignano and Spiliopoulos [2018]) and other types of RBF functions like gaussians (Lagaris et al. [2000]). However none of these approaches have used an adaptive sampling methods.

As for the sampling method, there have been several studies that discuss importance sampling method to accelerate the training of deep neural networks. Zhao and Zhang [2015] discusses improvement of the convergence rate in the case of proximal stochastic gradient descent (prox-SGD) by reducing the variance of the stochastic gradient using importance sampling from the distribution proportional to the nuclear norm of the gradient of the loss function. This sampling method using the gradient of the loss function is also adopted in Alain et al. [2015]. Apart from this method, sampling from the prediction variance and ranking the training data points according to the last computed loss were investigated in Chang et al. [2017] and Loshchilov and Hutter [2015], respectively.

3 Methods

A Radial Basis Function (RBF) network can be seen as a fully connected neural network which projects points in $\mathbb{R}^d$ to points in $\mathbb{R}$. The network consists of a d-node input layer, followed by a hidden RBF layer with n-nodes and finally projects to a single output node (Figure 1). The hidden nodes in the RBF layer compute a nonlinear transformation of their input of the form $\phi(\|x - c_i\|)$. Often $\phi()$ is chosen to be a Gaussian $\phi(r) = e^{-(\varepsilon r)^2}$, however here we work mostly with Multiquadratic functions which have the form $\phi(r) = \sqrt{1 + (\varepsilon r)^2}$ (following the literature: Chen et al. [2011], Mai-Duy and Tran-Cong [2001]).

[Figure 1 about here.]

An alternative representation for this network can be written down in closed form:

$$f(x) = \sum_{i=1}^{n} w_i \phi_i(\|x - c_i\|) \tag{1}$$

Depending on the number of hidden nodes n, a radial basis function network is a universal approximator, which means that it can represent a wide variety of interesting functions when given appropriate parameters (Park and Sandberg [1991]). We use this fact and the fact that the sum of radial basis functions are easy to integrate, to approximate initial value problems (IVP) for simple ordinary differential equations.

3.1 RBF decomposition, learning w

In this approach we use n radial basis functions of the form $\phi_i(\|x - c_i\|) = \sqrt{1 + \beta \|x - c_i\|^2}$ where $i \in [1, 2, ..., n-1, n]$. To approximate a continuous function $f(x)$ we then decompose this function using a linear combination of the given RBF basis and solve for the weights w_i in Equation 1. We fix the cenetroids c_i and the variance parameter β and convert the approximation into a system of linear equations in w_i. We use chosen centroids of the radial basis as the training data to obtain n equations with n unknowns. The training data is collected from an equidistant grid of the domain resulting in the following system of linear equations:

$$A \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{bmatrix} \tag{2}$$

, where the elements of A are $a_{ij} = \phi(\|x_i - x_j\|)$. We can solve this equation using standard techniques which we do in Matlab.

ODE example

Here we look at a simple ODE, which has the form: $\frac{dy}{dx} = 6\pi(\sin(2\pi x)/(4\pi) - \sin(6\pi x)/(12\pi))$ on the interval $x \in]0, 1[$. We first approximate the right hand side using Equation 2 to solve for w_i. We then use the known integrals of ϕ to obtain the solution $y(x)$.

$$\begin{aligned} \frac{dy}{dx} &= \sum_{i=1}^{n} w_i \sqrt{1 + \beta \left\| x - c_i \right\|^2} \\ y(x) &= \int_0^1 \frac{dy}{dx} dx = \sum_{i=1}^{n} w_i \frac{\sqrt{\beta}(x - c_i)\sqrt{1 + \beta \left\| x - c_i \right\|^2} + \sinh^1(\beta \left\| x - c_i \right\|)}{2\sqrt{\beta}} \end{aligned} \tag{3}$$

Algorithm 1 RBF decomposition

```
1: procedure RBF DECOMPOSITION
2:     c ← Initialization                        ▷ Pick the centeroids from an equidistant grid
3:     $\beta$ ← Initialization                      ▷ Choose a fixed $\beta$
4:     $D$ ← squareform(pdist(c,'euclidean'))     ▷ Compute euclidean distance matrix
5:     $A \leftarrow \sqrt{1 + \beta D^2}$                  ▷ Compute matrix A from Equation 2
6:     $B$ ← Initialization from $f(x)$
7:     $W$ ← linsolve$(A, B)$                     ▷ Solve linear system for $w$
8:     Use $W$ and Equation 3 to obtain $y(x)$
```

PDE example

We used the same technique to obtain a solution to the following first order PDE:

$$\begin{aligned} 2\frac{dy}{dx_1} + \frac{dy}{dx_2} &= \sin x_1 \quad \Omega \in]0, 1[\\ y(0, x2) &= \cos x_2 \end{aligned} \tag{4}$$

.

Here we take an alternate approach by decomposing the true solution $y(x_1, x_2)$ in radial basis functions and estimating the w parameters directly by solving the appropriate system of linear equations. The system of linear equations not only includes points in the interior but also boundary conditions:

$$\begin{bmatrix} LA \\ \dots \\ A \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} = \begin{bmatrix} g \\ \dots \\ p_1 \end{bmatrix} \tag{5}$$

where LA encodes the $n \times n$ matrix associated with the interior points and A is an $n_{bc} \times n$ matrix to satisfy the boundary conditions. n_{bc} stands for the number of boundary points chosen to fit the linear system.

3.2 RBF decomposition, learning w, c and β

We also tried to use stochastic gradient descent algorithms to solve for the trainable parameters using keras and python. In this approach we used radial basis functions where the weights w, centroids c and the variances β are learned.

Here we initialize an RBF network with n hidden nodes and randomly sampling the centroids for each node between 0 and 1 and assign to each node a fixed β. Subsequently we use backpropagation to optimize the parameters c, β and w using RMSprop.

3.3 Adaptive Training Points Sampling

The previous methods described in Zhao and Zhang [2015], Alain et al. [2015] use the norm of the gradient of the loss function $\|\nabla\psi_i(\boldsymbol{w})\|_*$, however, as the number of parameters grow, the computational cost of the gradient can be negligible. In Loshchilov and Hutter [2015], the loss itself is used to rank the importance of the data points. The biggest difference between the present study and the previous studies is that, when approximating the solution of the differential equations, the user can generate data points so that the decrease of the loss is maximized. Therefore, we use the distribution of the loss to pick the training points.

3.3.1 The Loss

Let $f(x)$ be the function approximated by the RBF network and $f^*(x)$ be the true solution. Note that there are cases where the true solution does not have an analytic form. In this study, we investigate two losses: quadratic loss and square root of the quadratic loss. The quadratic loss is

$$\|f(x) - f^*(x)\|_2^2\,,$$

and the other loss is the square root of the quadratic loss:

$$\|f(x) - f^*(x)\|_2\,.$$

3.3.2 Sampling Algorithm

In order to simplify the problem, we assume that the true solution is known or can easily be evaluated for a specific point in the domain. In this case, since we know or can compute the distribution of the loss for any given point, we can use Metropolis-Hasting algorithm to generate the data points. The algorithm is summarized in Algorithm 2. This sampling is done after each 100 epochs of training.

Algorithm 2 Metropolis-Hasting Algorithm for Data Points Generation

```
1: procedure METROPOLIS-HASTING
2:     $\boldsymbol{x}_0 \leftarrow$ Initialization                ▷ Pick the center point of domain
3:     $\sigma \leftarrow$ Initialization                    ▷ ∼10% of domain
4:     $\boldsymbol{x}_{\text{temp}} \leftarrow \boldsymbol{x}_0$
5:     $X_{\text{gen}} \leftarrow$ An empty vector                  ▷ Store generated points
6:     for $i < i_{\max}$ do                   ▷ $i_{\max}$ = Number of generated points
7:         $\boldsymbol{x}_{\text{proposed}} \leftarrow \text{random.normal}(\boldsymbol{x}_{\text{temp}}, \sigma)$
8:         $r \leftarrow loss(\boldsymbol{x}_{\text{proposed}})/loss(\boldsymbol{x}_{\text{temp}})$
9:         $U \leftarrow \text{Uniform}(0,1)$
10:        if $U \leq r$ then
11:            $X_{\text{gen}}$.append($x_{\text{proposed}}$)
12:            $\boldsymbol{x}_{\text{temp}} \leftarrow \boldsymbol{x}_{\text{proposed}}$
13:        else
14:            $X_{\text{gen}}$.append($x_{\text{temp}}$)
15:            $\boldsymbol{x}_{\text{temp}} \leftarrow \boldsymbol{x}_{\text{temp}}$
16:    return $X_{\text{gen}}$
```

4 Results

4.1 RBF decomposition, learning w

The results of the initial value problems described in the methods are presented in Figure 2 and Figure 3. The ODE approximation is a nearly exact fit and predicts the function $y(x)$ accurately. With regard to the PDE problem, the approximation is not as good but still reasonable.

[Figure 2 about here.]

[Figure 3 about here.]

We also looked at the effect of β on the approximation. As β increases the width of the RBF functions becomes smaller and the approximation becomes more peaked.

β	$\|y - \text{approx}\|_2$
1	395.2803
10	355.8473
100	355.3819
1000	654.3316
10000	737.6344
100000	752.3315

Table 1: As the variance parameter β increases, the radial basis functions become more 'peaked' shaped and the approximation becomes worse. The best approximation is achieved with a large variance (low β), which makes the approximate solution more smooth.

4.2 Data Points Sampling

To test the effect of the adaptive training data point sampling, we use the ODE

$$\frac{dy}{dx} = -100 \sin(10x), \tag{6}$$

where the solution is

$$y(x) = 10 \cos(10x) + const. \tag{7}$$

To solve this ODE, an RBF network with 1 layer of RBF with 1000 nodes are used. The detail of the method is explained in Section 3.2. The condition for this experiment is the following: `RMSprop` is used to minimize the mean squared error, the batch size is 50, and the total number of training epoch is 4000. As discussed in Section 3.3, the training data points sampling is done after each 100 epochs, therefore it is done 39 times.

An example of the sampled points is shown in Figure 4.

[Figure 4 about here.]

Figure 5 and 6 show the results of the training with and without the adaptive data points sampling. The detailed discussion on the performance is in Discussion section, but it can qualitatively be seen that the adaptive data point sampling improve the prediction.

[Figure 5 about here.]

[Figure 6 about here.]

5 Discussion

5.1 RBF decomposition learning w

Our results for the ODE example were very accurate and the accuracy increased when we increased the number of RBF functions (results not shown). The results were fairly insensitive to the hyperparameter β.

For the PDe example we looked at the effect of β on the error and found that an increase in β worsened our approximation significantly. This specific effect is likely caused by the fact that the true solution is quite smooth so it makes sense that we find that a small β results in the best approximation because the approximation will also be more smooth.

5.2 RBF decomposition learning w, c and β

We conclude from the approach where w, c and β are learnable parameters, that initialization is crucial for the algorithm to converge in a reasonable amount of time. The system seems especially sensitive to the initialization of β. Without being able to show any results, we have spend a significant amount of hours to try to get theses models to converge; often without success. It seems that the approach with fixed centroids and variances is superior.

5.3 Training Data Points Sampling

In Section 4.2, the results using the training data points sampling were shown. In this section, the quantitative comparison of the performance is discussed. The average of the error from the true solution is calculated by taking the average of the error on the 1000 linearly separated points. The process time for each method is also measured. The results are summarized in Table 2. It is clear that this sampling method improves the accuracy or the prediction. Although the process time is larger with the sampling, but it is not significant in this case. However, it is possible that the process increases for more complicated differential equations.

Table 2: Summary of the performance of each method.

	No sampling	Sampling with sqrt quad loss $\sigma = 0.12$	Sampling with sqrt quad loss $\sigma = 0.50$	Sampling with quadratic loss $\sigma = 0.12$	Sampling with quadratic loss $\sigma = 0.50$
Process Time [s]	382.6	450.8	512.8	494.9	465.6
Averaged Error	4.15	1.57	1.87	1.08	1.13

As discussed in Section 3.3, quadratic loss and its squared root were used.

[Figure 7 about here.]

6 Conclusion & Contribution

In this report, we approximated the solution to an IVP of an ODE and a PDE testcase with different methods and concluded that training with fixed centeroids and variances led to more consistent convergence and more accurate results.

Adaptive training data points sampling is applied to solving differential equations for the first time. It is shown that the picking training data points according to the distribution of the loss significantly improves the quality of the prediction, without adding excessive computational cost. This method could be further improved by more sophisticated method such as Hamiltonian MCMC. Also it is important to test the applicability of the method in more complicated problems such as high-dimensional differential equations.

References

G. Alain, A. Lamb, C. Sankar, A. Courville, and Y. Bengio. Variance reduction in sgd by distributed importance sampling. *arXiv preprint arXiv:1511.06481*, 2015.

J. Berg and K. Nyström. A unified deep artificial neural network approach to partial differential equations in complex geometries. *Neurocomputing*, 317:28–41, 2018.

H.-S. Chang, E. Learned-Miller, and A. McCallum. Active bias: Training more accurate neural networks by emphasizing high variance samples. In *Advances in Neural Information Processing Systems*, pages 1002–1012, 2017.

H. Chen, L. Kong, and W.-J. Leng. Numerical solution of pdes via integrated radial basis function networks with adaptive training algorithm. *Applied Soft Computing*, 11(1):855–860, 2011.

I. E. Lagaris, A. C. Likas, and D. G. Papageorgiou. Neural-network methods for boundary value problems with irregular boundaries. *IEEE Transactions on Neural Networks*, 11(5):1041–1049, 2000.

I. Loshchilov and F. Hutter. Online batch selection for faster training of neural networks. *arXiv preprint arXiv:1511.06343*, 2015.

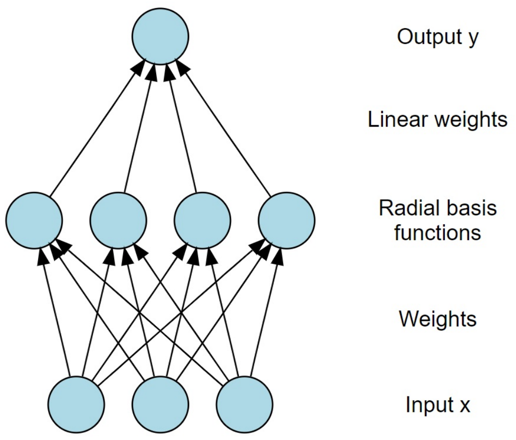
N. Mai-Duy and T. Tran-Cong. Numerical solution of differential equations using multiquadric radial basis function networks. *Neural Networks*, 14(2):185–199, 2001.

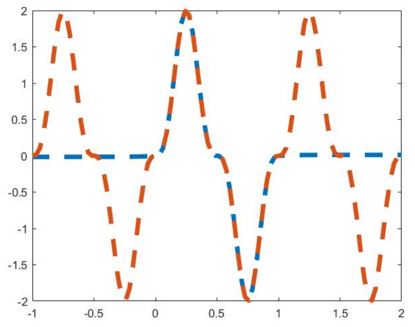
J. Park and I. W. Sandberg. Universal approximation using radial-basis-function networks. *Neural computation*, 3(2):246–257, 1991.

J. Sirignano and K. Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, 375:1339–1364, 2018.

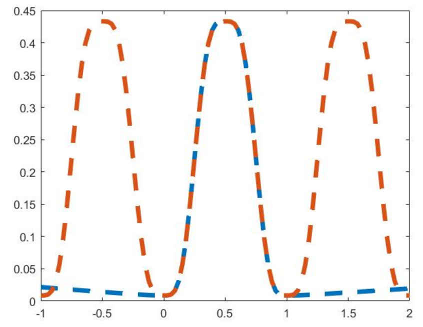
P. Zhao and T. Zhang. Stochastic optimization with importance sampling for regularized loss minimization. In *international conference on machine learning*, pages 1–9, 2015.

Figure 1: Diagram of RBF network. Source: Wikipedia



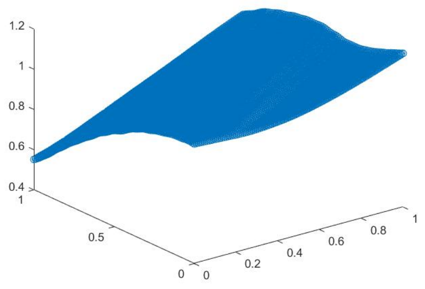


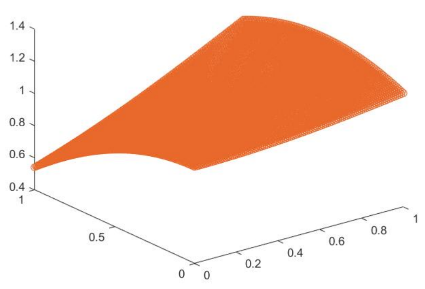
(a)



(b)

Figure 2: **a**: Here we show in orange the right hand side to be approximated: $6\pi(\sin(2\pi x)/(4\pi) - \sin(6\pi x)/(12\pi))$ and in blue the approximation on the interval $]0,1[$. **b**: Here it is shown the solution to the IVP where orange represents the true solution and blue the approximation using RBF decomposition on the interval $]0,1[$


(a)


(b)

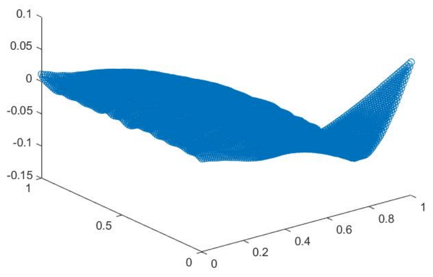

(c)

Figure 3: **a**: The approximate solution using RBF with $\beta = 300$, nInterior = 900 (30×30) and nBC = 300 (Boundary Condition). **b**: The true solution $y(x_1, x_2) = 0.5(1 + 2\cos(0.5(2x_2 - x_1)) - \cos(x_1)$. **c**: The error true $-$ approximation.

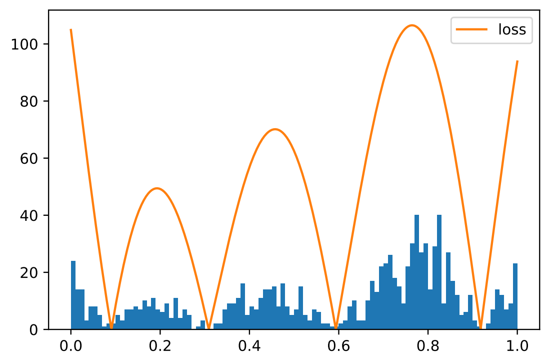
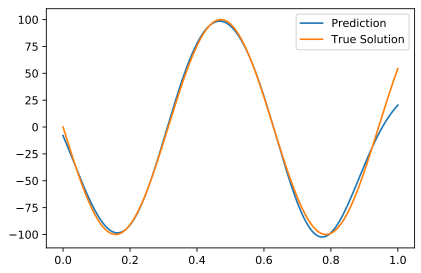


Figure 4:   An example of the sampled training data points (histogram), sampled from the distribution of the square root of the quadratic loss (orange line).

(a) 

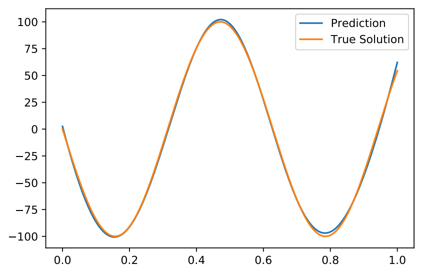
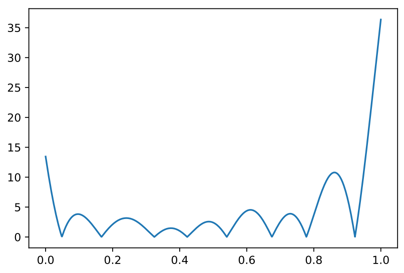
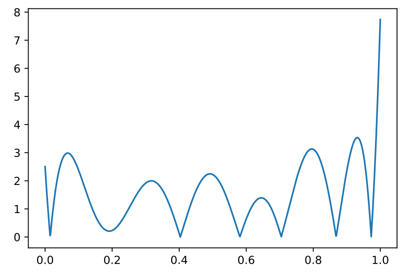
(b) 

Figure 5: **(a)** Results of the fit without adaptive training points sampling. **(b)** Results of the fit with the adaptive training points sampling using the square root of the quadratic loss.



(a)



(b)

Figure 6: **(a)** Discrepancy of the prediction from the true solution, without adaptive training points sampling. **(b)** Discrepancy of the solution from the true solution, with the adaptive training points sampling.

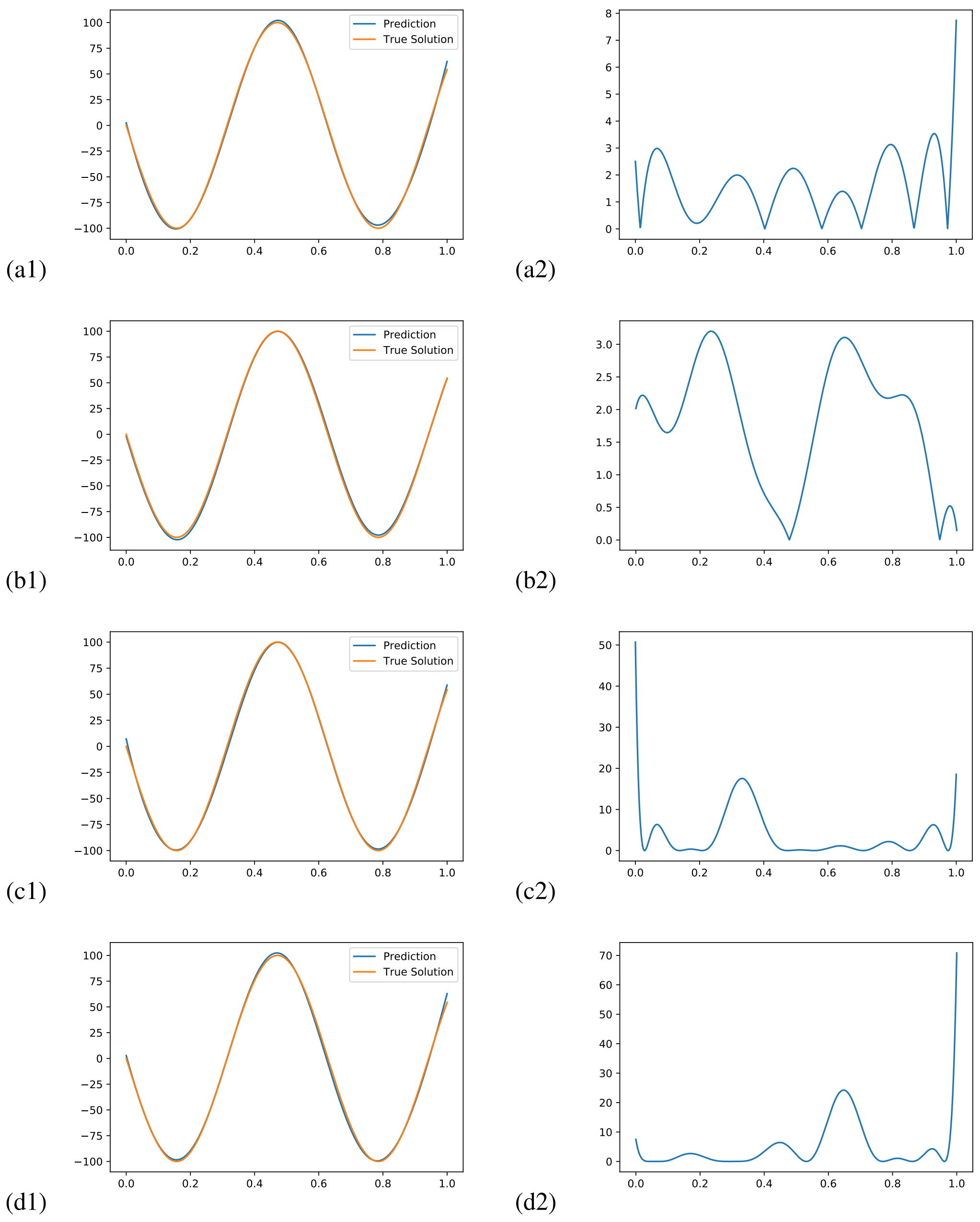


Figure 7: Comparison between different loss functions and sampling generation step-size. (a) and (b) use the squared root of the quadratic loss, and (c) and (d) use the quadratic loss. (a) and (c) use the step-size (variance of the normal distribution) of 12% of the domain size and (b) and (d) use the step-size of 50% of the domain size.