In [1]:
```
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
import numpy as np
import matplotlib.pyplot as plt
from nn_visualise import *
```

Section 1: 2nd ODE

Target: solve

$$y'' = f(x)$$

with

$$y(0) = 1, y(1) = 1$$

Solving the problem in the interval

$$[0, 1]$$

with step N=100, so

$$\Delta x = 1/99$$

With loss function:

$$L = \sum_{j=1}^{99} (\widehat{y''}(x_j) - f(x_j))^2 + \lambda(\widehat{y(1)} - y(1))^2 + \lambda(\widehat{y(0)} - y(0))^2$$

where

$$\widehat{y''}, \hat{y}$$

are outputs from the NN

In [2]:
```
x = np.linspace(0,1,100)
def f(x):
    return -2
```

Construction of the NN

In [3]:
```
class Model(nn.Module):
    def __init__(self):
        super().__init__()

        # Inputs to hidden layer linear transformation
        self.input = nn.Linear(1, 6)

        #self.hidden_1 = nn.Linear(32, 32)
        #self.hidden_2= nn.Linear(32, 32)

        self.output = nn.Linear(6,1)

        # Define sigmoid activation and softmax output
        self.sigmoid = nn.Sigmoid()
```

```
        self.softmax = nn.Softmax(dim=1)
    def forward(self, x):
        # Pass the input tensor through each of our operations
        x = self.input(x)
        x = self.sigmoid(x)

        #x = self.hidden_1(x)
        #x = self.sigmoid(x)
        #x = self.hidden_2(x)
        #x = self.sigmoid(x)

        x = self.output(x)
        return x
```

Section 2: Full batch gradient descent

In each time we call the back propagation, we first calculate the full batch lost (lost summed on all 100 samples)

$$L = \sum_{j=1}^{99} (\widehat{y''}(x_j) - f(x_j))^2 + \lambda(\widehat{y(1)} - y(1))^2 + \lambda(\widehat{y(0)} - y(0))^2$$

here we set

$$\lambda = 1$$

Some definitions worth mentioning:

1. Epoch: one epoch = forward feed of all samples
2. Batch: divide full sample into smaller batches
3. Iteration: In each iteration, we call bp on one batch

For example, if we divide the 100 samples in to 5 batches, then each batch has 20 samples.\ Batch size = 4 One epoch = 100 samples So we need 5 iterations to complete one epoch.

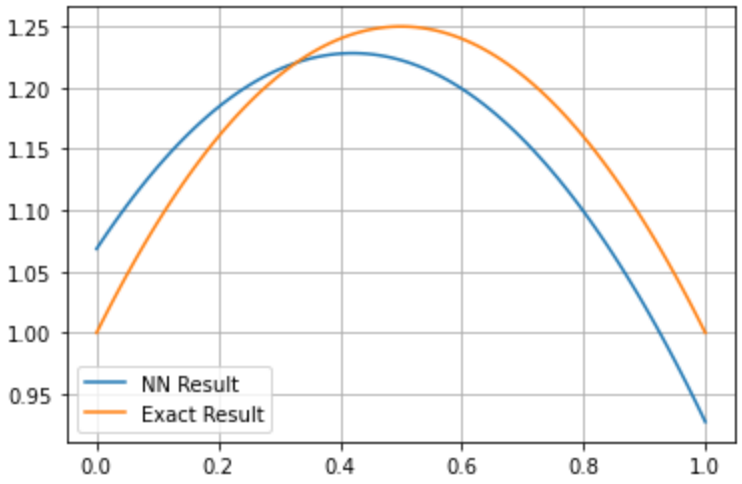
We will cover:\ 1.Full batch gradient descent: batch size = 100\ 2.SGD: batch size = 1\ 3.Batch SGD: batch size = 5 or 10 or 20

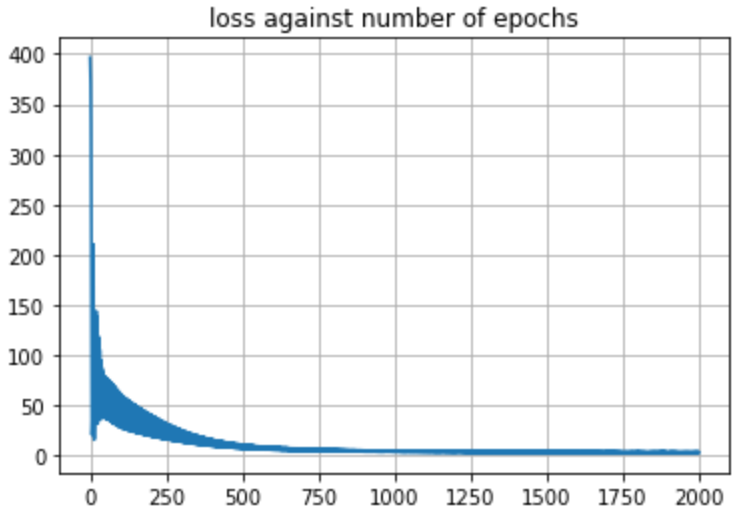
In [4]:
```
torch.manual_seed(1024)

model_full_batch = Model()
model_full_batch.train()
optimizer = optim.SGD(model_full_batch.parameters(), lr=0.01)

num_epochs =2000
h = 0.01 # step size for finite difference approximation
loss_list_full_batch = []

for i in range(num_epochs):
    loss = torch.tensor([[0.]])
    optimizer.zero_grad()
    for x in np.linspace(0,1,100):
        x= float(x)
        y_pred = model_full_batch.forward(torch.tensor([[x]]))
        y_minus_h = model_full_batch.forward(torch.tensor([[x-h]]))
        y_plus_h = model_full_batch.forward(torch.tensor([[x+h]]))
        y_2nd_deri= (y_plus_h-2*y_pred+y_minus_h)/(h**2)
        y_1st_deri = (y_plus_h-y_minus_h)/(2*h)
```

```
        if x==0:
            loss += 2*F.mse_loss(y_pred, torch.tensor([[float(1)]]))
        elif x==1:
            loss+=2*F.mse_loss(y_pred, torch.tensor([[float(1)]]))
        else:
            loss += F.mse_loss(y_2nd_deri, torch.tensor([[float(f(x))]]))
    loss.backward()
    optimizer.step()
    loss_list_full_batch.append(loss.detach().numpy()[0][0])
```

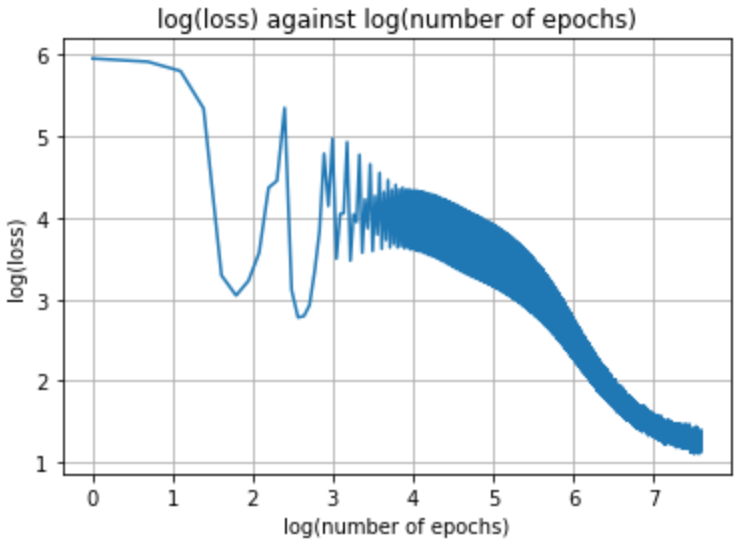
In [ ]:

In [5]:
```
plot_nn_ode_result(model_full_batch,num_epochs,loss_list_full_batch)
```






```
C:\Users\haoyang\OneDrive\MSc Courses\NNPDE_casestudy\nn_visualise.py:27: RuntimeWarnin
g: divide by zero encountered in log
  plt.plot(np.log(range(num_epochs)),np.log(np.array(loss_list)))
```



Section 3: Batch gradient descent

In each iteration, we use the loss function:

$$L = \sum_{j=1}^{n} (\widehat{y''}(x_j) - f(x_j))^2 + \lambda(\widehat{y(1)} - y(1))^2 + \lambda(\widehat{y(0)} - y(0))^2$$

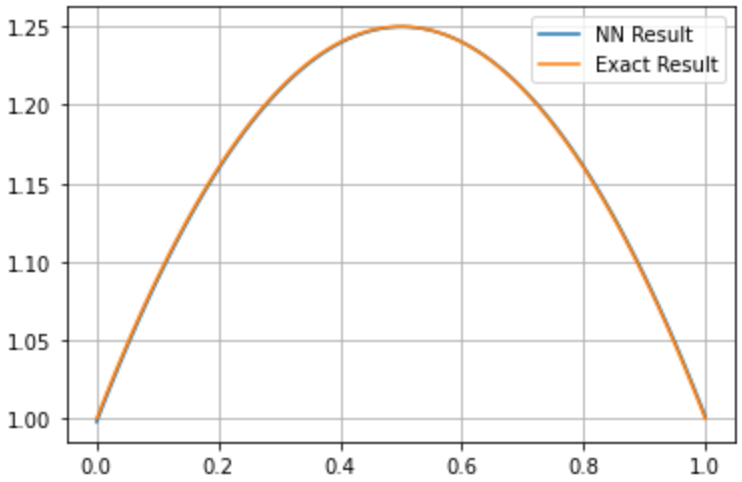
where n = batch size.\ n = $\frac{fullsamplesize}{iteration}$

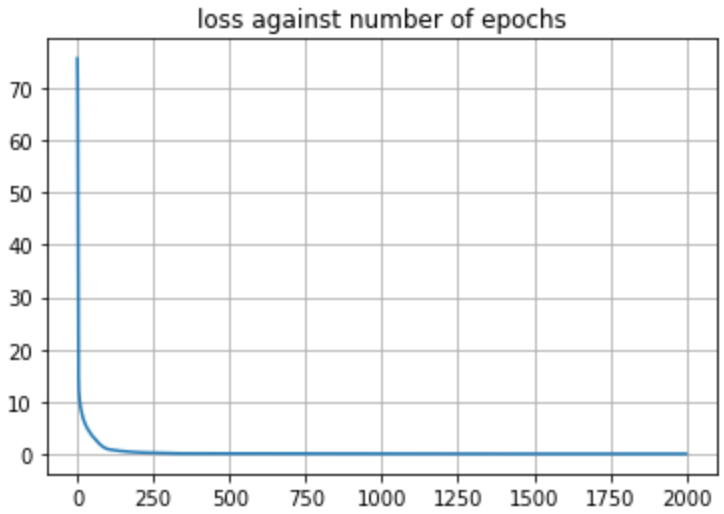
Try iteration=5 (divide sample in to 5 slices)

In [6]:
```
torch.manual_seed(1024)

model_batch_gd = Model()
model_batch_gd.train()
optimizer = optim.SGD(model_batch_gd.parameters(), lr=0.01)

#num_epochs =10
h = 0.01 # step size for finite difference approximation
loss_list_batch_grad_desc = []
iteration =5

for i in range(num_epochs):

    x_space =np.split(np.linspace(0,1,100),iteration)

    for i in range(iteration):
        loss = torch.tensor([[0.]])
        optimizer.zero_grad()
        for x in x_space[i]:
            x= float(x)
            y_pred = model_batch_gd.forward(torch.tensor([[x]]))
            y_minus_h = model_batch_gd.forward(torch.tensor([[x-h]]))
            y_plus_h = model_batch_gd.forward(torch.tensor([[x+h]]))
            y_2nd_deri= (y_plus_h-2*y_pred+y_minus_h)/(h**2)
            y_1st_deri = (y_plus_h-y_minus_h)/(2*h)

            if x==0:
                loss += 2*F.mse_loss(y_pred, torch.tensor([[float(1)]]))
            elif x==1:
                loss+=2*F.mse_loss(y_pred, torch.tensor([[float(1)]]))
            else:
```

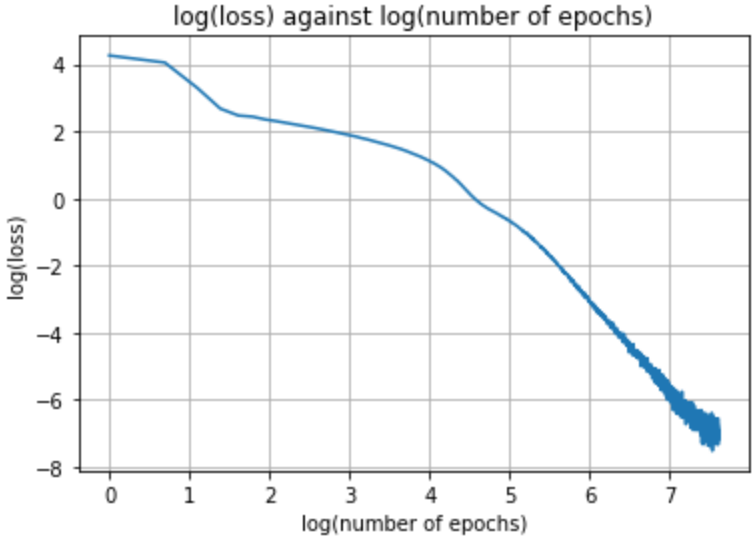
```
                loss += F.mse_loss(y_2nd_deri, torch.tensor([[float(f(x))]]))
        loss.backward()
        optimizer.step()

    loss_list_batch_grad_desc.append(loss.detach().numpy()[0][0])
```

In [7]:
```
plot_nn_ode_result(model_batch_gd,num_epochs,loss_list_batch_grad_desc)
```



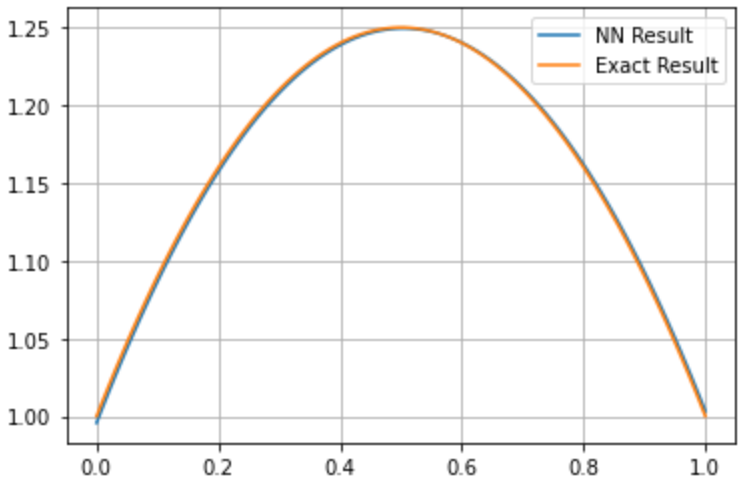


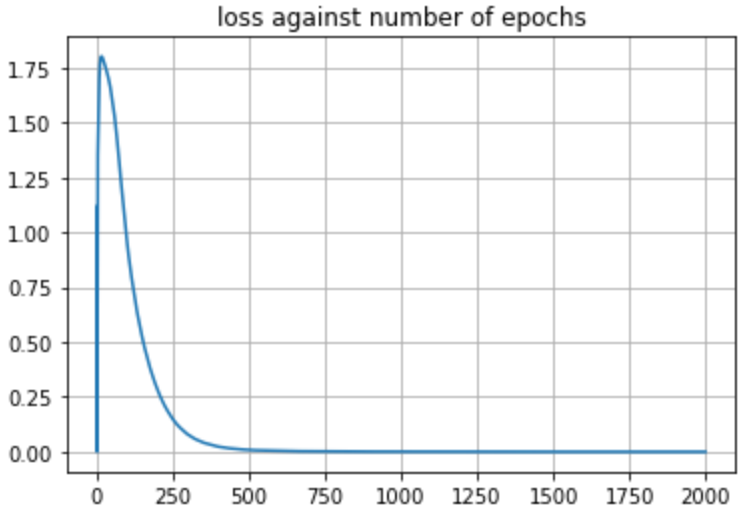

```
C:\Users\haoyang\OneDrive\MSc Courses\NNPDE_casestudy\nn_visualise.py:27: RuntimeWarnin
g: divide by zero encountered in log
  plt.plot(np.log(range(num_epochs)),np.log(np.array(loss_list)))
```



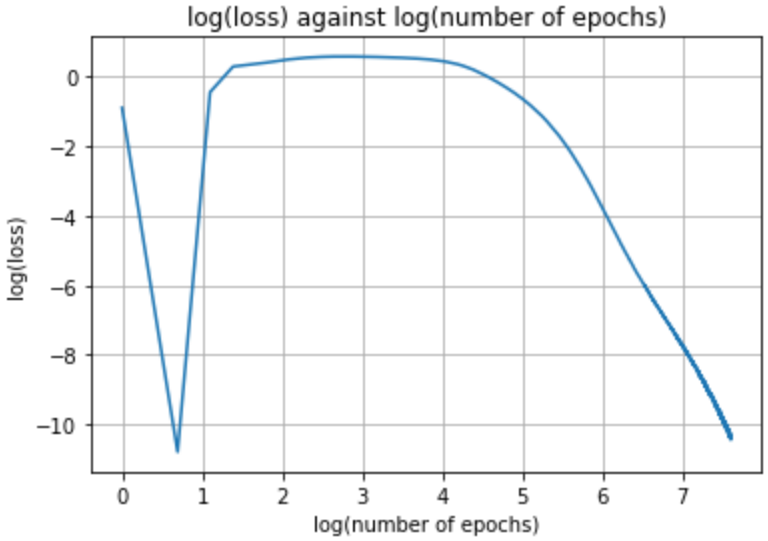
Try iteration=100 (each iteration uses only one sample)

In [8]:
```
torch.manual_seed(1024)

model_batch_gd = Model()
model_batch_gd.train()
optimizer = optim.SGD(model_batch_gd.parameters(), lr=0.01)

#num_epochs =10
h = 0.01 # step size for finite difference approximation
loss_list_batch_grad_desc = []
iteration =100

for i in range(num_epochs):

    x_space =np.split(np.linspace(0,1,100),iteration)

    for i in range(iteration):
        loss = torch.tensor([[0.]])
        optimizer.zero_grad()
        for x in x_space[i]:
            x= float(x)
            y_pred = model_batch_gd.forward(torch.tensor([[x]]))
            y_minus_h = model_batch_gd.forward(torch.tensor([[x-h]]))
            y_plus_h = model_batch_gd.forward(torch.tensor([[x+h]]))
            y_2nd_deri= (y_plus_h-2*y_pred+y_minus_h)/(h**2)
            y_1st_deri = (y_plus_h-y_minus_h)/(2*h)

            if x==0:
                loss += 2*F.mse_loss(y_pred, torch.tensor([[float(1)]]))
            elif x==1:
                loss+=2*F.mse_loss(y_pred, torch.tensor([[float(1)]]))
            else:
                loss += F.mse_loss(y_2nd_deri, torch.tensor([[float(f(x))]]))
        loss.backward()
        optimizer.step()

    loss_list_batch_grad_desc.append(loss.detach().numpy()[0][0])

plot_nn_ode_result(model_batch_gd,num_epochs,loss_list_batch_grad_desc)
```






```
C:\Users\haoyang\OneDrive\MSc Courses\NNPDE_casestudy\nn_visualise.py:27: RuntimeWarnin
g: divide by zero encountered in log
  plt.plot(np.log(range(num_epochs)),np.log(np.array(loss_list)))
```



Section 4: SGD

In each iteration, we use the loss function:

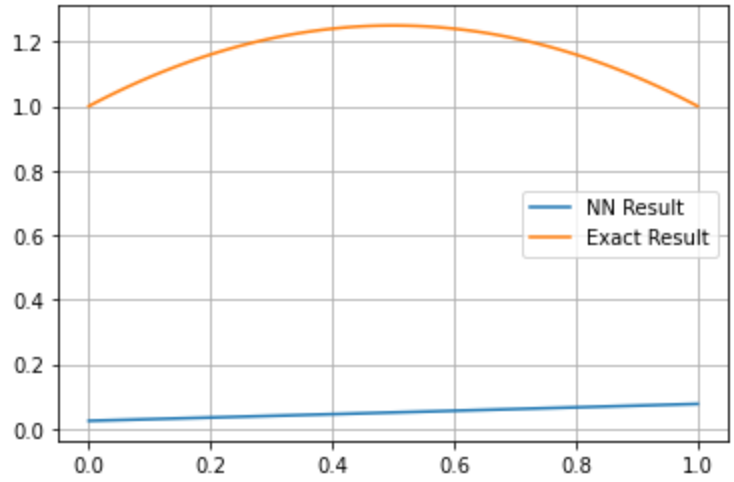
$$L = (\widehat{y''}(x_j) - f(x_j))^2 + \lambda(\widehat{y(1)} - y(1))^2 + \lambda(\widehat{y(0)} - y(0))^2$$

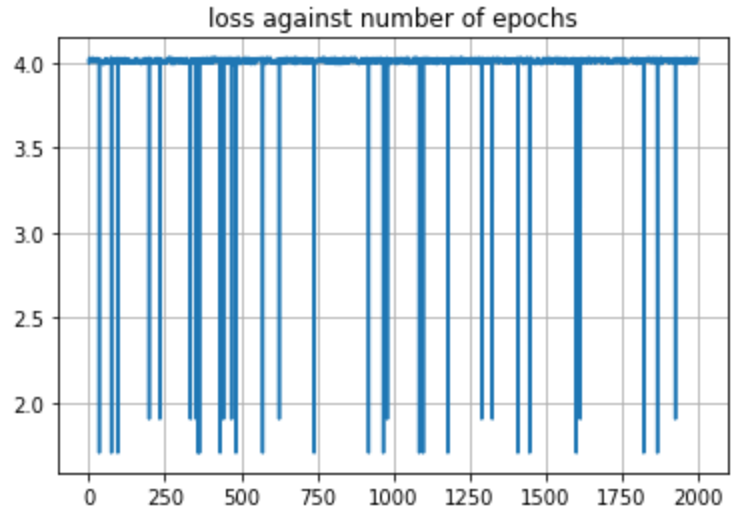
where n = batch size=1. In each iteration, one x_j is picked randomly from 0 to 1

In [9]:
```
torch.manual_seed(1024)

model_sgd = Model()
model_sgd.train()
optimizer = optim.SGD(model_batch_gd.parameters(), lr=0.01)

#num_epochs =100
h = 0.01 # step size for finite difference approximation
loss_list_sgd = []
iteration =100


for i in range(num_epochs):
```

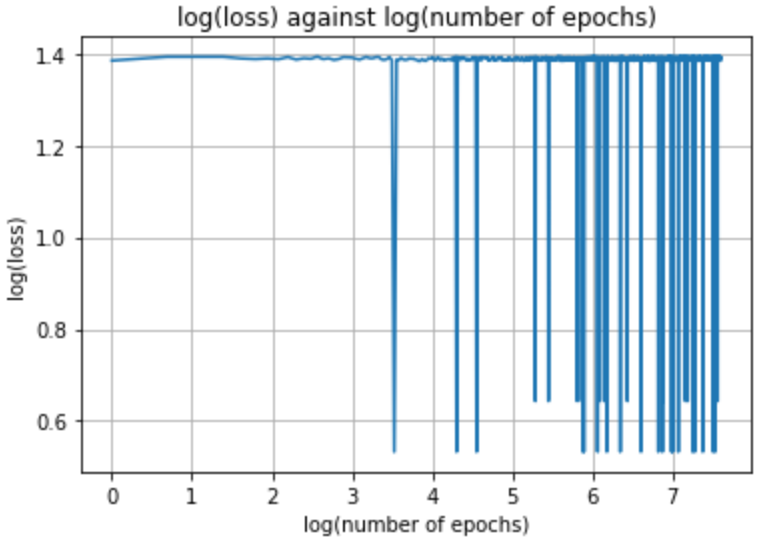
```
    for i in range(iteration):
        loss = torch.tensor([[0.]])
        optimizer.zero_grad()
        random_pick = np.random.randint(0,100)
        x = np.linspace(0,1,100)[random_pick]
        x= float(x)
        y_pred = model_sgd.forward(torch.tensor([[x]]))
        y_minus_h = model_sgd.forward(torch.tensor([[x-h]]))
        y_plus_h = model_sgd.forward(torch.tensor([[x+h]]))
        y_2nd_deri= (y_plus_h-2*y_pred+y_minus_h)/(h**2)
        y_1st_deri = (y_plus_h-y_minus_h)/(2*h)

        if x==0:
            loss += 2*F.mse_loss(y_pred, torch.tensor([[float(1)]]))
        elif x==1:
            loss+=2*F.mse_loss(y_pred, torch.tensor([[float(1)]]))
        else:
            loss += F.mse_loss(y_2nd_deri, torch.tensor([[float(f(x))]]))
        loss.backward()
        optimizer.step()

    loss_list_sgd.append(loss.detach().numpy()[0][0])
```

In [10]:
```
plot_nn_ode_result(model_sgd,num_epochs,loss_list_sgd)
```






```
C:\Users\haoyang\OneDrive\MSc Courses\NNPDE_casestudy\nn_visualise.py:27: RuntimeWarnin
g: divide by zero encountered in log
  plt.plot(np.log(range(num_epochs)),np.log(np.array(loss_list)))
```


log(loss) against log(number of epochs)
log(loss)
log(number of epochs)
1.4
1.2
1.0
0.8
0.6
0
1
2
3
4
5
6
7

In [ ]:

In [ ]:

In [ ]: