poisson_1d

August 5, 2020

1 Modelling poisson using PINN

Author: Manu Jayadharan
Written as part of FlowNet package, a TensorFlow based neural network package to solve fluid flow PDEs.

Solving the poisson equation $-\Delta u = f$ using a physics informed neural network

1.1 1D problem poisson problem

1.1.1 Manufactured solution

We use $u = 3sin(4x)$ for $x \in [-1, 1]$

1.1.2 Importing packages

[665]:
```
import numpy as np
import tensorflow as tf
from tensorflow import keras
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

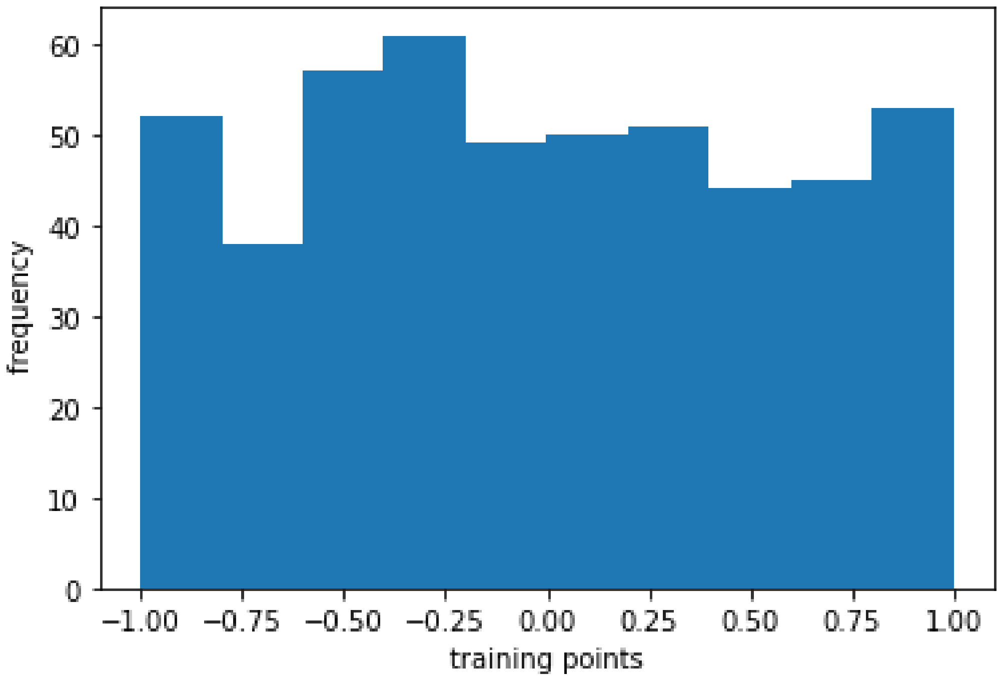
1.2 Manufacturing data for trainig

[666]:
```
np.random.seed(123)
X_tr_pde = np.random.uniform(-1,1,500).reshape(500,1)
```

Plotting histogram of randomly selected points to make sure they are uniformly distributed

[609]:
```
plt.hist(X_tr_pde)
plt.xlabel("training points")
plt.ylabel("frequency ")
```

[609]:
```
Text(0, 0.5, 'frequency ')
```



[610]:
```
Y_tr_pde = np.zeros((X_tr_pde.shape[0],1))
```

[611]:
```
Y_tr_pde = np.concatenate([Y_tr_pde,np.zeros((Y_tr_pde.shape[0],1))],axis=1)
Y_tr_pde.shape
```

[611]:
```
(500, 2)
```

[572]:
```
X_tr_Dr_bc_left = -1*np.ones(200).reshape(200,1)
X_tr_Dr_bc_right = 1*np.ones(200).reshape(200,1)
X_bc = np.concatenate([X_tr_Dr_bc_left,X_tr_Dr_bc_right],axis=0)

Y_tr_bc = 3*np.sin(4*X_bc)
Y_tr_bc = np.concatenate([Y_tr_bc,np.ones((Y_tr_bc.shape[0],1))],axis=1)
```

1.2.1 Scaling the inputs(optional)

[612]:
```
# from sklearn.preprocessing import StandardScaler

# scaler = StandardScaler()
# X_tr_pde = scaler.fit_transform(X_tr_pde)
```

[613]:
```
X_tr = np.concatenate((X_tr_pde, X_bc), axis=0)
Y_tr = np.concatenate((Y_tr_pde, Y_tr_bc), axis=0)
```

[614]:
```
# from sklearn.preprocessing import StandardScaler

# scaler = StandardScaler()
# X_tr = scaler.fit_transform(X_tr)
# X_tr.std()
```

[615]:
```
(X_tr[0:3] +0.0056276)/4.520744138916567
```

[615]:
```
array([[ 0.0881638 ],
       [-0.0933682 ],
       [-0.11959745]])
```

1.3 Defining the NN model (custom Keras model)

- **Model specifications**: 7 layers: 1 input layer, 3 hidden layers with 20 neurons each, 1 dense intermediate layer, 1 gradient layer, 1 laplacian layer, 1 pde layer.
- Output is a list of two elements: value of function and value of the pde operator.
- mean squared error is used for finding the cost function.
- Specialized, sgd(stochastic gradient descenet) type optimizer is used: either nadam or adam.
- tanh activation functions are used.

[623]:
```
from tensorflow.keras import backend as K
class Poisson1d(tf.keras.Model):

    def __init__(self):
        super(Poisson1d, self).__init__()
#         self.batch_norm_ = keras.layers.BatchNormalization()
        self.flatten_input = keras.layers.Flatten()

        he_kernel_init = keras.initializers.he_uniform()

        self.dense_1 = keras.layers.Dense(20, activation="tanh",
                                          kernel_initializer=he_kernel_init,
                                          name="dense_1")
        self.dense_2 = keras.layers.Dense(20, activation="tanh",
                                          kernel_initializer=he_kernel_init,
                                         name="dense_2")
        self.dense_3 = keras.layers.Dense(20, activation="tanh",
                                          kernel_initializer=he_kernel_init,
                                         name="dense_3")
        self.dense_4 = keras.layers.Dense(1,
                                         name="dense_4")

    def findGrad(self,func,argm):
        return keras.layers.Lambda(lambda x: K.gradients(x[0],x[1])[0])␣
↪([func,argm])
```

```
    def findPdeLayer(self, pde_lhs, input_arg):
        return keras.layers.Lambda(lambda z: z[0] + 48*tf.sin(4*z[1]))␣
 ↪([pde_lhs, input_arg])

    def call(self, inputs):
#         layer_0 = self.batch_norm_input(inputs)
        layer_0 = self.flatten_input(inputs)
#         layer_0_1 = self.batch_norm_input(layer_0)
        layer_1 = self.dense_1(layer_0)
#         layer_2_0 = self.batch_norm_input(layer_1)
        layer_2 = self.dense_2(layer_1)
#         layer_3_0 = self.batch_norm_(layer_2)
        layer_3 = self.dense_3(layer_2)
        layer_4 = self.dense_4(layer_3)

        grad_layer = self.findGrad(layer_4, inputs)
        laplace_layer = self.findGrad(grad_layer, inputs)
        pde_layer = self.findPdeLayer(laplace_layer, inputs)

        return layer_4, pde_layer
```

1.3.1 Defining the loss functions

[628]:
```
#Loss coming from the boundary terms
def u_loss(y_true, y_pred):
    y_true_act = y_true[:,:-1]
    at_boundary = tf.cast(y_true[:,-1:,],bool)
    u_sq_error = (1/2)*tf.square(y_true_act-y_pred)
    return tf.where(at_boundary, u_sq_error, 0.)

#Loss coming from the PDE constrain
def pde_loss(y_true, y_pred):
    y_true_act = y_true[:,:-1]
    at_boundary = tf.cast(y_true[:,-1:,],bool)
    #need to change this to just tf.square(y_pred) after pde constrain is added␣
 ↪to grad_layer
#     pde_sq_error = (1/2)*tf.square(y_true_act-y_pred)
    pde_sq_error = (1/2)*tf.square(y_pred)
    return tf.where(at_boundary,0.,pde_sq_error)
```

1.3.2 Instantiating and compiling the poisson_model

[633]:
```
poisson_NN = Poisson1d()
```

[634]:
```
poisson_NN.compile(loss=[u_loss,pde_loss],optimizer="adam")
```

[635]:

```
poisson_NN.fit(x=X_tr, y=Y_tr,epochs=100)
```

```
Epoch 1/100
29/29 [==============================] - 0s 8ms/step - loss: 259.7652 -
output_1_loss: 1.6987 - output_2_loss: 258.0666
Epoch 2/100
29/29 [==============================] - 0s 8ms/step - loss: 199.6366 -
output_1_loss: 3.7078 - output_2_loss: 195.9288
Epoch 3/100
29/29 [==============================] - 0s 4ms/step - loss: 167.5798 -
output_1_loss: 5.7012 - output_2_loss: 161.8786
Epoch 4/100
29/29 [==============================] - 0s 6ms/step - loss: 147.2352 -
output_1_loss: 6.9477 - output_2_loss: 140.2874
Epoch 5/100
29/29 [==============================] - 0s 6ms/step - loss: 127.5192 -
output_1_loss: 8.2436 - output_2_loss: 119.2756
Epoch 6/100
29/29 [==============================] - 0s 7ms/step - loss: 107.6846 -
output_1_loss: 9.3374 - output_2_loss: 98.3473
Epoch 7/100
29/29 [==============================] - 0s 7ms/step - loss: 94.4738 -
output_1_loss: 10.1162 - output_2_loss: 84.3576
Epoch 8/100
29/29 [==============================] - 0s 6ms/step - loss: 82.4624 -
output_1_loss: 10.6387 - output_2_loss: 71.8237
Epoch 9/100
29/29 [==============================] - 0s 3ms/step - loss: 69.6333 -
output_1_loss: 10.6447 - output_2_loss: 58.9887
Epoch 10/100
29/29 [==============================] - 0s 4ms/step - loss: 55.9477 -
output_1_loss: 10.3795 - output_2_loss: 45.5682
Epoch 11/100
29/29 [==============================] - 0s 9ms/step - loss: 42.3476 -
output_1_loss: 9.9878 - output_2_loss: 32.3598
Epoch 12/100
29/29 [==============================] - 0s 8ms/step - loss: 29.1121 -
output_1_loss: 9.2851 - output_2_loss: 19.8270
Epoch 13/100
29/29 [==============================] - 0s 8ms/step - loss: 19.3638 -
output_1_loss: 8.3920 - output_2_loss: 10.9718
Epoch 14/100
29/29 [==============================] - 1s 19ms/step - loss: 14.4945 -
output_1_loss: 7.5555 - output_2_loss: 6.9389
Epoch 15/100
29/29 [==============================] - 0s 8ms/step - loss: 12.6046 -
output_1_loss: 6.7571 - output_2_loss: 5.8475
```

```
Epoch 16/100
29/29 [==============================] - 0s 14ms/step - loss: 10.5798 -
output_1_loss: 5.9927 - output_2_loss: 4.5870
Epoch 17/100
29/29 [==============================] - 0s 6ms/step - loss: 9.1434 -
output_1_loss: 5.2864 - output_2_loss: 3.8570
Epoch 18/100
29/29 [==============================] - 0s 10ms/step - loss: 7.8388 -
output_1_loss: 4.6431 - output_2_loss: 3.1957
Epoch 19/100
29/29 [==============================] - 0s 7ms/step - loss: 6.9969 -
output_1_loss: 4.1099 - output_2_loss: 2.8870
Epoch 20/100
29/29 [==============================] - 0s 14ms/step - loss: 6.1960 -
output_1_loss: 3.6754 - output_2_loss: 2.5206
Epoch 21/100
29/29 [==============================] - 0s 9ms/step - loss: 5.5211 -
output_1_loss: 3.2757 - output_2_loss: 2.2454
Epoch 22/100
29/29 [==============================] - 0s 8ms/step - loss: 4.7912 -
output_1_loss: 2.8491 - output_2_loss: 1.9420
Epoch 23/100
29/29 [==============================] - 0s 8ms/step - loss: 4.2952 -
output_1_loss: 2.4328 - output_2_loss: 1.8624
Epoch 24/100
29/29 [==============================] - 0s 10ms/step - loss: 3.6080 -
output_1_loss: 2.0475 - output_2_loss: 1.5605
Epoch 25/100
29/29 [==============================] - 0s 12ms/step - loss: 2.9790 -
output_1_loss: 1.7081 - output_2_loss: 1.2709
Epoch 26/100
29/29 [==============================] - 1s 19ms/step - loss: 2.4628 -
output_1_loss: 1.4023 - output_2_loss: 1.0605
Epoch 27/100
29/29 [==============================] - 0s 11ms/step - loss: 2.0742 -
output_1_loss: 1.1864 - output_2_loss: 0.8878
Epoch 28/100
29/29 [==============================] - 0s 9ms/step - loss: 1.7676 -
output_1_loss: 1.0167 - output_2_loss: 0.7509
Epoch 29/100
29/29 [==============================] - 0s 9ms/step - loss: 1.5395 -
output_1_loss: 0.8897 - output_2_loss: 0.6498
Epoch 30/100
29/29 [==============================] - 0s 10ms/step - loss: 1.3731 -
output_1_loss: 0.7909 - output_2_loss: 0.5822
Epoch 31/100
29/29 [==============================] - 0s 8ms/step - loss: 1.2571 -
output_1_loss: 0.7032 - output_2_loss: 0.5539
```

```
Epoch 32/100
29/29 [==============================] - 0s 8ms/step - loss: 1.0477 -
output_1_loss: 0.6191 - output_2_loss: 0.4286
Epoch 33/100
29/29 [==============================] - 0s 10ms/step - loss: 0.9775 -
output_1_loss: 0.5610 - output_2_loss: 0.4165
Epoch 34/100
29/29 [==============================] - 0s 6ms/step - loss: 0.8646 -
output_1_loss: 0.5001 - output_2_loss: 0.3645
Epoch 35/100
29/29 [==============================] - 0s 7ms/step - loss: 0.8120 -
output_1_loss: 0.4500 - output_2_loss: 0.3620
Epoch 36/100
29/29 [==============================] - 0s 9ms/step - loss: 0.7423 -
output_1_loss: 0.4108 - output_2_loss: 0.3315
Epoch 37/100
29/29 [==============================] - 0s 13ms/step - loss: 0.6386 -
output_1_loss: 0.3692 - output_2_loss: 0.2693
Epoch 38/100
29/29 [==============================] - 0s 17ms/step - loss: 0.5728 -
output_1_loss: 0.3305 - output_2_loss: 0.2423
Epoch 39/100
29/29 [==============================] - 0s 12ms/step - loss: 0.5022 -
output_1_loss: 0.2914 - output_2_loss: 0.2108
Epoch 40/100
29/29 [==============================] - 0s 8ms/step - loss: 0.4589 -
output_1_loss: 0.2579 - output_2_loss: 0.2010
Epoch 41/100
29/29 [==============================] - 0s 12ms/step - loss: 0.4043 -
output_1_loss: 0.2306 - output_2_loss: 0.1737
Epoch 42/100
29/29 [==============================] - 0s 8ms/step - loss: 0.3789 -
output_1_loss: 0.2028 - output_2_loss: 0.1761
Epoch 43/100
29/29 [==============================] - 0s 8ms/step - loss: 0.3445 -
output_1_loss: 0.1821 - output_2_loss: 0.1624
Epoch 44/100
29/29 [==============================] - 1s 20ms/step - loss: 0.3695 -
output_1_loss: 0.1614 - output_2_loss: 0.2081
Epoch 45/100
29/29 [==============================] - 0s 8ms/step - loss: 0.2772 -
output_1_loss: 0.1431 - output_2_loss: 0.1341
Epoch 46/100
29/29 [==============================] - 0s 8ms/step - loss: 0.2570 -
output_1_loss: 0.1235 - output_2_loss: 0.1336
Epoch 47/100
29/29 [==============================] - 0s 9ms/step - loss: 0.2372 -
output_1_loss: 0.1082 - output_2_loss: 0.1290
```

```
Epoch 48/100
29/29 [==============================] - 0s 11ms/step - loss: 0.2071 -
output_1_loss: 0.0952 - output_2_loss: 0.1119
Epoch 49/100
29/29 [==============================] - 0s 16ms/step - loss: 0.1887 -
output_1_loss: 0.0816 - output_2_loss: 0.1071
Epoch 50/100
29/29 [==============================] - 0s 9ms/step - loss: 0.1826 -
output_1_loss: 0.0705 - output_2_loss: 0.1121
Epoch 51/100
29/29 [==============================] - 0s 8ms/step - loss: 0.1690 -
output_1_loss: 0.0628 - output_2_loss: 0.1063
Epoch 52/100
29/29 [==============================] - 0s 8ms/step - loss: 0.1603 -
output_1_loss: 0.0537 - output_2_loss: 0.1066
Epoch 53/100
29/29 [==============================] - 0s 13ms/step - loss: 0.1697 -
output_1_loss: 0.0455 - output_2_loss: 0.1243
Epoch 54/100
29/29 [==============================] - 0s 12ms/step - loss: 0.1362 -
output_1_loss: 0.0394 - output_2_loss: 0.0968
Epoch 55/100
29/29 [==============================] - 0s 14ms/step - loss: 0.1200 -
output_1_loss: 0.0342 - output_2_loss: 0.0857
Epoch 56/100
29/29 [==============================] - 0s 9ms/step - loss: 0.1060 -
output_1_loss: 0.0292 - output_2_loss: 0.0768
Epoch 57/100
29/29 [==============================] - 0s 11ms/step - loss: 0.1285 -
output_1_loss: 0.0248 - output_2_loss: 0.1037
Epoch 58/100
29/29 [==============================] - 0s 10ms/step - loss: 0.0924 -
output_1_loss: 0.0206 - output_2_loss: 0.0718
Epoch 59/100
29/29 [==============================] - 0s 13ms/step - loss: 0.0891 -
output_1_loss: 0.0174 - output_2_loss: 0.0717
Epoch 60/100
29/29 [==============================] - 0s 5ms/step - loss: 0.0812 -
output_1_loss: 0.0147 - output_2_loss: 0.0664
Epoch 61/100
29/29 [==============================] - 0s 9ms/step - loss: 0.0689 -
output_1_loss: 0.0127 - output_2_loss: 0.0562
Epoch 62/100
29/29 [==============================] - 0s 5ms/step - loss: 0.0796 -
output_1_loss: 0.0104 - output_2_loss: 0.0691
Epoch 63/100
29/29 [==============================] - 0s 10ms/step - loss: 0.0631 -
output_1_loss: 0.0085 - output_2_loss: 0.0546
```

```
Epoch 64/100
29/29 [==============================] - 0s 6ms/step - loss: 0.0613 -
output_1_loss: 0.0069 - output_2_loss: 0.0544
Epoch 65/100
29/29 [==============================] - 0s 8ms/step - loss: 0.0605 -
output_1_loss: 0.0054 - output_2_loss: 0.0551
Epoch 66/100
29/29 [==============================] - 0s 5ms/step - loss: 0.0541 -
output_1_loss: 0.0045 - output_2_loss: 0.0496
Epoch 67/100
29/29 [==============================] - 0s 14ms/step - loss: 0.0599 -
output_1_loss: 0.0035 - output_2_loss: 0.0564
Epoch 68/100
29/29 [==============================] - 0s 6ms/step - loss: 0.0621 -
output_1_loss: 0.0029 - output_2_loss: 0.0592
Epoch 69/100
29/29 [==============================] - 0s 12ms/step - loss: 0.0662 -
output_1_loss: 0.0025 - output_2_loss: 0.0637
Epoch 70/100
29/29 [==============================] - 0s 9ms/step - loss: 0.0576 -
output_1_loss: 0.0017 - output_2_loss: 0.0559
Epoch 71/100
29/29 [==============================] - 0s 6ms/step - loss: 0.0537 -
output_1_loss: 0.0015 - output_2_loss: 0.0522
Epoch 72/100
29/29 [==============================] - 0s 9ms/step - loss: 0.0597 -
output_1_loss: 0.0011 - output_2_loss: 0.0586
Epoch 73/100
29/29 [==============================] - 0s 10ms/step - loss: 0.0433 -
output_1_loss: 8.0166e-04 - output_2_loss: 0.0425
Epoch 74/100
29/29 [==============================] - 0s 13ms/step - loss: 0.0458 -
output_1_loss: 5.7144e-04 - output_2_loss: 0.0452
Epoch 75/100
29/29 [==============================] - 0s 9ms/step - loss: 0.0509 -
output_1_loss: 3.3017e-04 - output_2_loss: 0.0506
Epoch 76/100
29/29 [==============================] - 0s 16ms/step - loss: 0.0572 -
output_1_loss: 4.1536e-04 - output_2_loss: 0.0568
Epoch 77/100
29/29 [==============================] - 0s 11ms/step - loss: 0.0598 -
output_1_loss: 2.8450e-04 - output_2_loss: 0.0595
Epoch 78/100
29/29 [==============================] - 0s 5ms/step - loss: 0.0454 -
output_1_loss: 1.8232e-04 - output_2_loss: 0.0453
Epoch 79/100
29/29 [==============================] - 0s 6ms/step - loss: 0.0440 -
output_1_loss: 1.1054e-04 - output_2_loss: 0.0439
```

```
Epoch 80/100
29/29 [==============================] - 0s 9ms/step - loss: 0.0441 -
output_1_loss: 1.1150e-04 - output_2_loss: 0.0440
Epoch 81/100
29/29 [==============================] - 0s 11ms/step - loss: 0.0662 -
output_1_loss: 2.7556e-04 - output_2_loss: 0.0659
Epoch 82/100
29/29 [==============================] - 0s 8ms/step - loss: 0.0587 -
output_1_loss: 1.0060e-04 - output_2_loss: 0.0586
Epoch 83/100
29/29 [==============================] - 0s 9ms/step - loss: 0.0475 -
output_1_loss: 1.2469e-04 - output_2_loss: 0.0474
Epoch 84/100
29/29 [==============================] - 0s 12ms/step - loss: 0.0620 -
output_1_loss: 1.9952e-04 - output_2_loss: 0.0618
Epoch 85/100
29/29 [==============================] - 0s 7ms/step - loss: 0.0734 -
output_1_loss: 2.8111e-04 - output_2_loss: 0.0731
Epoch 86/100
29/29 [==============================] - 0s 9ms/step - loss: 0.0401 -
output_1_loss: 9.4932e-05 - output_2_loss: 0.0400
Epoch 87/100
29/29 [==============================] - 0s 10ms/step - loss: 0.0352 -
output_1_loss: 1.5613e-04 - output_2_loss: 0.0350
Epoch 88/100
29/29 [==============================] - 0s 7ms/step - loss: 0.0378 -
output_1_loss: 2.0924e-04 - output_2_loss: 0.0376
Epoch 89/100
29/29 [==============================] - 0s 7ms/step - loss: 0.0394 -
output_1_loss: 2.8825e-04 - output_2_loss: 0.0391
Epoch 90/100
29/29 [==============================] - 0s 8ms/step - loss: 0.0496 -
output_1_loss: 3.8716e-04 - output_2_loss: 0.0492
Epoch 91/100
29/29 [==============================] - 0s 7ms/step - loss: 0.0464 -
output_1_loss: 3.6435e-04 - output_2_loss: 0.0460
Epoch 92/100
29/29 [==============================] - 0s 7ms/step - loss: 0.0418 -
output_1_loss: 3.4827e-04 - output_2_loss: 0.0415
Epoch 93/100
29/29 [==============================] - 0s 7ms/step - loss: 0.0472 -
output_1_loss: 4.5195e-04 - output_2_loss: 0.0467
Epoch 94/100
29/29 [==============================] - 0s 10ms/step - loss: 0.0305 -
output_1_loss: 4.2057e-04 - output_2_loss: 0.0301
Epoch 95/100
29/29 [==============================] - 0s 7ms/step - loss: 0.0282 -
output_1_loss: 4.6182e-04 - output_2_loss: 0.0278
```

```
Epoch 96/100
29/29 [==============================] - 0s 10ms/step - loss: 0.0415 -
output_1_loss: 3.9756e-04 - output_2_loss: 0.0411
Epoch 97/100
29/29 [==============================] - 0s 8ms/step - loss: 0.0256 -
output_1_loss: 4.7936e-04 - output_2_loss: 0.0251
Epoch 98/100
29/29 [==============================] - 0s 15ms/step - loss: 0.0375 -
output_1_loss: 5.0414e-04 - output_2_loss: 0.0370
Epoch 99/100
29/29 [==============================] - 0s 10ms/step - loss: 0.0350 -
output_1_loss: 4.9992e-04 - output_2_loss: 0.0345
Epoch 100/100
29/29 [==============================] - 0s 11ms/step - loss: 0.0268 -
output_1_loss: 5.2197e-04 - output_2_loss: 0.0262
```

[635]:
```
<tensorflow.python.keras.callbacks.History at 0x7f64b425cfa0>
```

1.4 Testing the trained network

[636]:
```python
X_test_st = np.random.uniform(-1,1,100).reshape(100,1)
```

1.4.1 Scaling the test set (only if the trainng data was scaled)

[637]:
```python
# #Scaling test set
# X_test_st_2 = scaler.transform(X_test_st)

#xtrain: mean, std: -0.005627660222786496 4.520744138916567
```

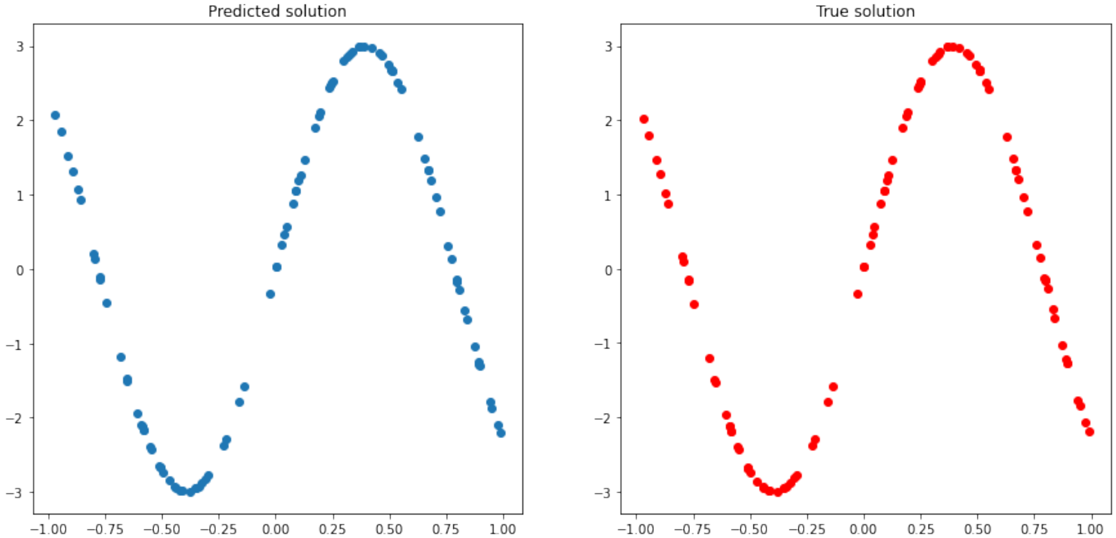
[638]:
```python
Y_test = poisson_NN.predict(X_test_st)
```

1.4.2 Plotting the true and predicted solutions

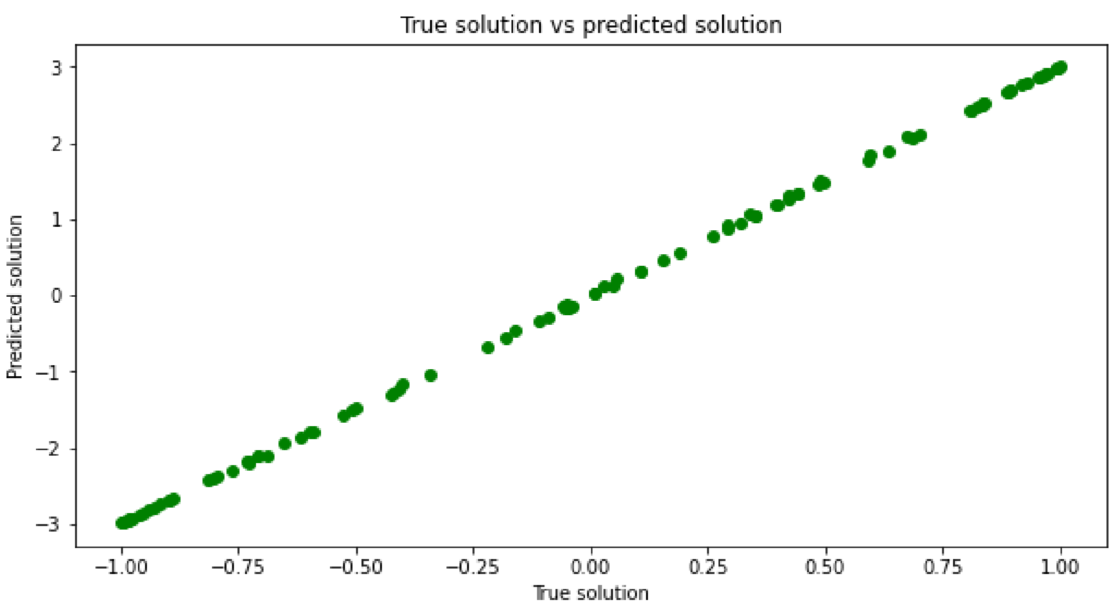
[662]:
```python
# fig, ax = plt.subplots(nrows=2,ncols=2, figsize=(10,10))
#plotting predicted solution
plt.figure(figsize=(15,7))
plt.subplot(1,2,1)
plt.scatter(X_test_st, Y_test[0][:,0])
plt.title("Predicted solution")

plt.subplot(1,2,2)
plt.scatter(X_test_st, 3*np.sin(4*X_test_st), c="r")
plt.title("True solution")
```

[662]:
```
Text(0.5, 1.0, 'True solution')
```



[664]:
```
#True vs predicted solution
plt.figure(figsize=(10,5))
plt.scatter(np.sin(4*X_test_st), Y_test[0][:,0], c="g")
plt.title("True solution vs predicted solution")
plt.xlabel("True solution")
plt.ylabel("Predicted solution")
plt.show()
```



1.4.3 Notes to be made

- For second order pde, some form of normalization is needed for convergence.
- If the the input data already comes normalized, there is no problem.
- If the data is not normalized, then we would want to some kind of normalization technique like batch normalization or normalization of incoming data.