Michael Ferko

Dr. Jacek Zurada

ECE 614 Deep Learning

14 February 2020

Lab 1: Regression

Regression with Multiple Linear Perceptron (MLP) Modeling of the Saddle and Ackley Functions

Objective:

To understand and implement nonlinear regression with multilayer perceptrons. Note that some lines of inactive code lines are preceded with # in preparation for other select tasks.

Tasks:

1. Run the code as given. Do this cell by cell by clicking the Run button. Perform two complete runs, one for the simple z function, another for z being the ackley function.

1a. Code Run 1: Simple Z function ($z = x^2 - y^2$)

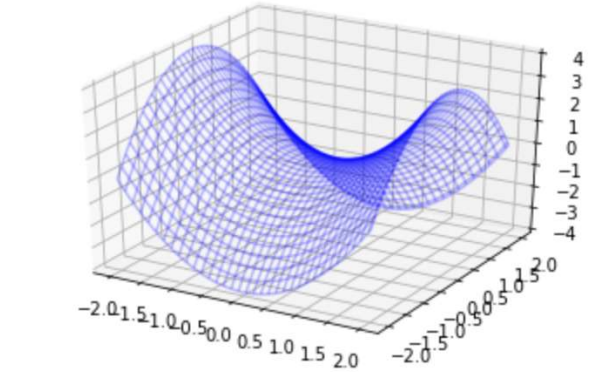


Figure 1: Data Generation of Hyperbolic Paraboloid Mesh grid

```
xycol (10201, 2)
zcol (10201, 1)
```

Figure 2: Data Reshaping into column arrays for neural processing

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 10)                30
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 11
=================================================================
Total params: 41
Trainable params: 41
Non-trainable params: 0
_________________________________________________________________
```

Figure 3: Model setup of Neural Network into a Sequence

```
10201/10201 [==============================] - 1s 138us/step - loss: 1.0326
Epoch 2/10
10201/10201 [==============================] - 1s 103us/step - loss: 0.0212
Epoch 3/10
10201/10201 [==============================] - 1s 97us/step - loss: 0.0100
Epoch 4/10
10201/10201 [==============================] - 1s 99us/step - loss: 0.0084
Epoch 5/10
10201/10201 [==============================] - 1s 99us/step - loss: 0.0064
Epoch 6/10
10201/10201 [==============================] - 1s 98us/step - loss: 0.0066
Epoch 7/10
10201/10201 [==============================] - 1s 100us/step - loss: 0.0057
Epoch 8/10
10201/10201 [==============================] - 1s 102us/step - loss: 0.0061
Epoch 9/10
10201/10201 [==============================] - 1s 103us/step - loss: 0.0055
Epoch 10/10
10201/10201 [==============================] - 1s 102us/step - loss: 0.0053
```

Figure 4: Model Training progression for each Epoch

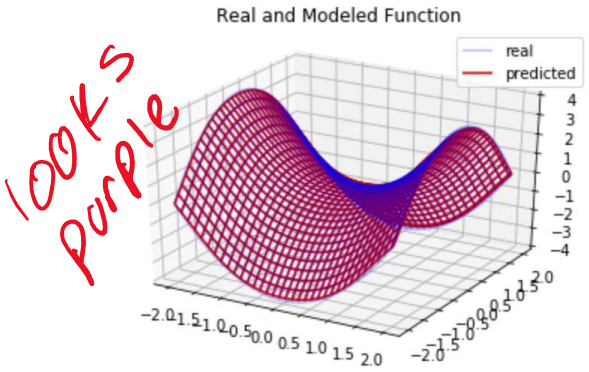


Figure 5(a): 3D Mesh grid Real vs Predicted

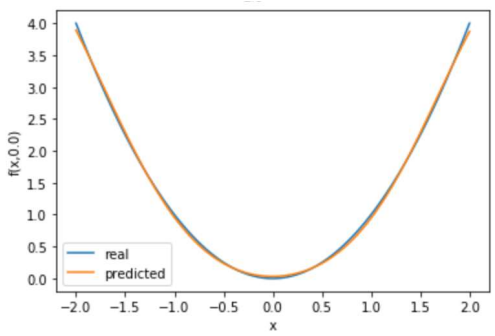


Figure 5(b): 2D section at y = 0

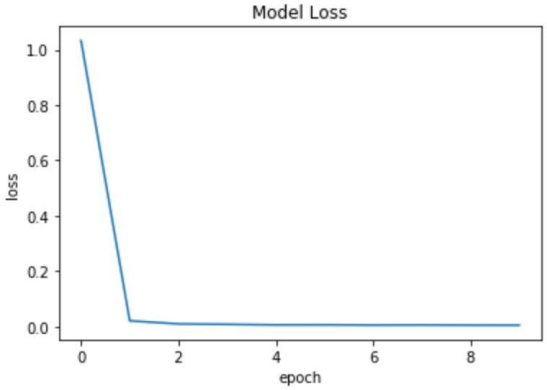


Figure 5(c): Model Loss vs Epoch

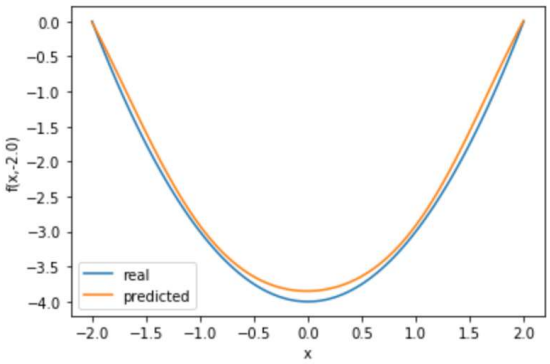


Figure 5(d): 2D section at y = -2

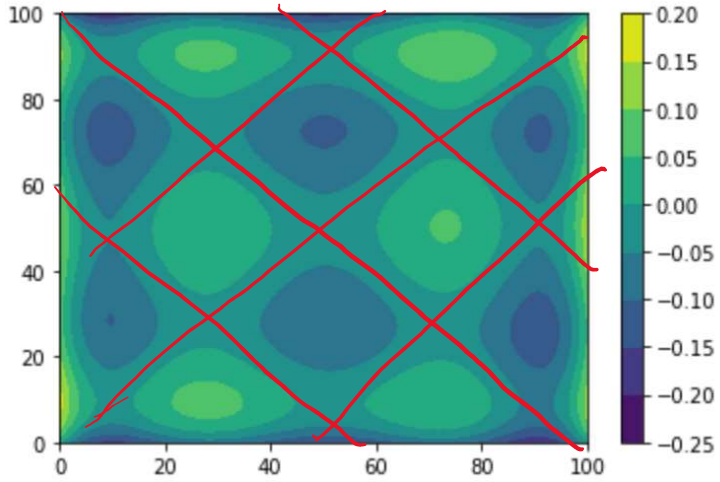


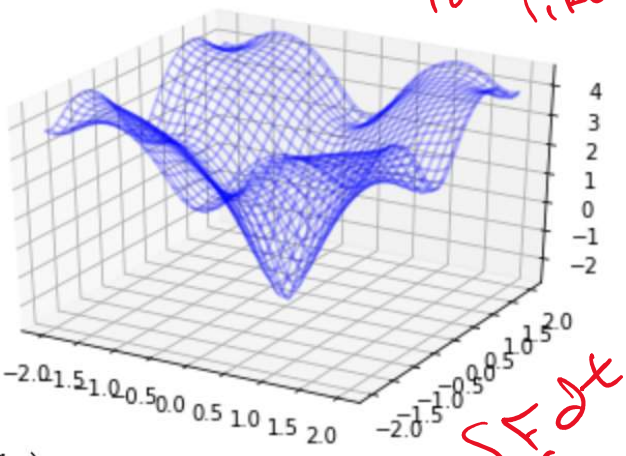
Figure 5(e): Topological Error Contour

1b. Code Run 2: Ackley function ($z = -20e^{-0.2\sqrt{0.5(x^2+y^2)}} - e^{0.5(\cos \pi x + cos \pi y)} + 20$)

Switched comment out of call line in Data Generation to switch from the simple z function to the Ackley function as shown in Figure 6. Anytime hereafter in which the Ackley has been used, it may be assumed by the reader that this comment change has been made.

```
#z  = simple_function(xv, yv)
z  = ackley_function(xv, yv)
```

Figure 6: Comment out Switch




```
x (101,)
xv (101, 101)
y (101,)
yv (101, 101)
```

Figure 7: Data Generation of Ackley Mesh grid

```
xycol (10201, 2)
zcol (10201, 1)
```

Figure 8: Data Reshaping into column arrays for neural processing

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_5 (Dense)              (None, 10)                30
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 11
=================================================================
Total params: 41
Trainable params: 41
Non-trainable params: 0
_________________________________________________________________
```

Figure 9: Model setup of Neural Network into a Sequence

```
Epoch 1/10
10201/10201 [==============================] - 1s 120us/step - loss: 0.8627
Epoch 2/10
10201/10201 [==============================] - 1s 111us/step - loss: 0.1015
Epoch 3/10
10201/10201 [==============================] - 1s 106us/step - loss: 0.0529
Epoch 4/10
10201/10201 [==============================] - 1s 107us/step - loss: 0.0386
Epoch 5/10
10201/10201 [==============================] - 1s 104us/step - loss: 0.0349
Epoch 6/10
10201/10201 [==============================] - 1s 101us/step - loss: 0.0327
Epoch 7/10
10201/10201 [==============================] - 1s 103us/step - loss: 0.0310
Epoch 8/10
10201/10201 [==============================] - 1s 103us/step - loss: 0.0291
Epoch 9/10
10201/10201 [==============================] - 1s 100us/step - loss: 0.0270
Epoch 10/10
10201/10201 [==============================] - 1s 97us/step - loss: 0.0245
```

Figure 10: Model Training progression for each Epoch

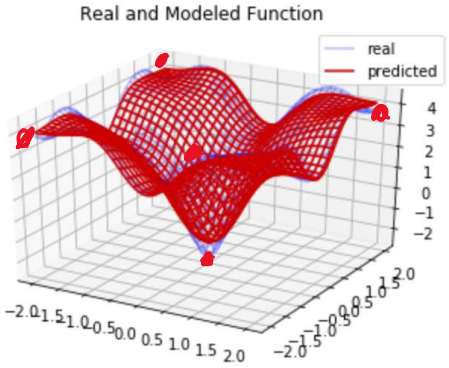


Figure 11(a): 3D Mesh grid Real vs Predicted

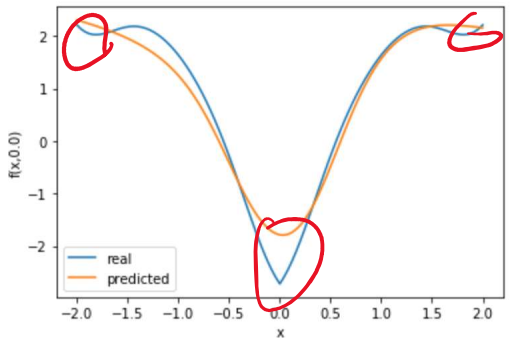


Figure 11(b): 2D section at y = 0

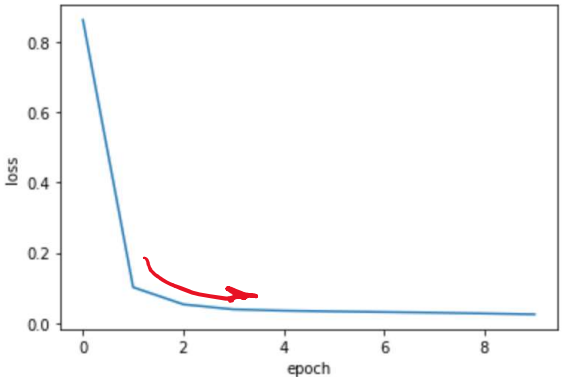


Figure 11(c): Model Loss vs Epoch

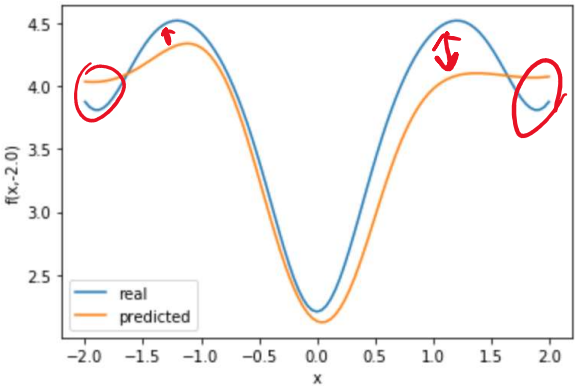


Figure 11(d): 2D section at y = -2

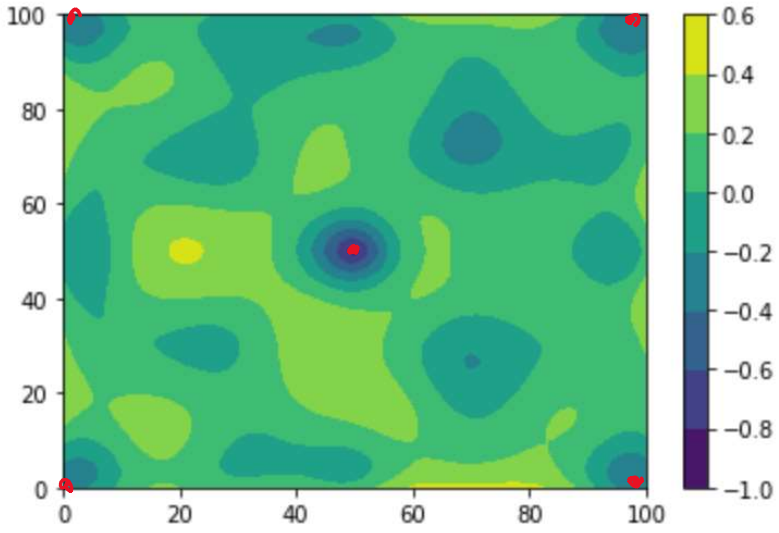


Figure 11(e): Topological Error Contour

2. In the Model Setup code block, change the activation functions of the hidden layer to 'relu' and train both models for the two z functions. How accurate is the regression for each z compared with Task 1 (example of the relu function is below in Appendix A).

2a. Change Activation Function:

Under Model Setup, the activation function was changed to relu:

```
model.add(Dense(10, activation='relu', input_shape=(2,)))
```

Figure 12: Activation function change

2b. Code Run 1: Simple Z function ($z = x^2 - y^2$)

For the sake of simplicity, only visualizations will be shown for simple observable differences between hidden layer activation functions.

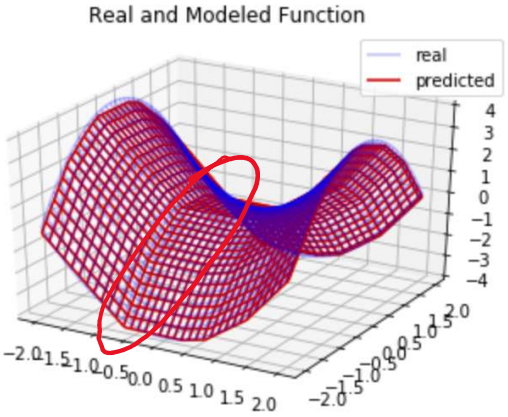


Figure 13(a): 3D Mesh grid Real vs Predicted

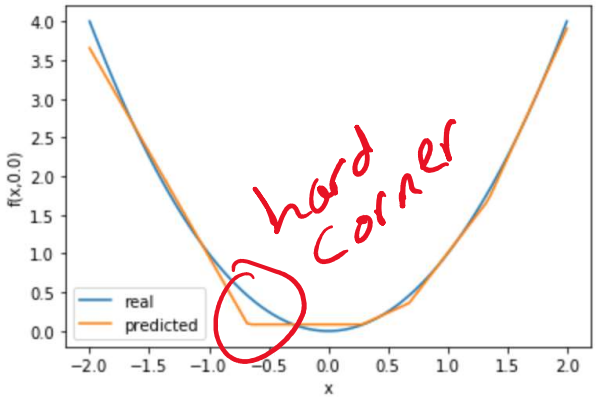


Figure 13(b): 2D section at y = 0

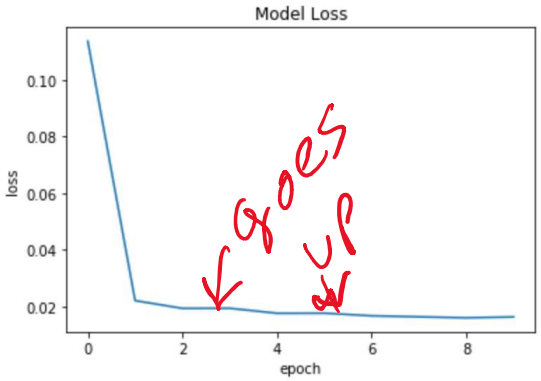


Figure 13(c): Model Loss vs Epoch

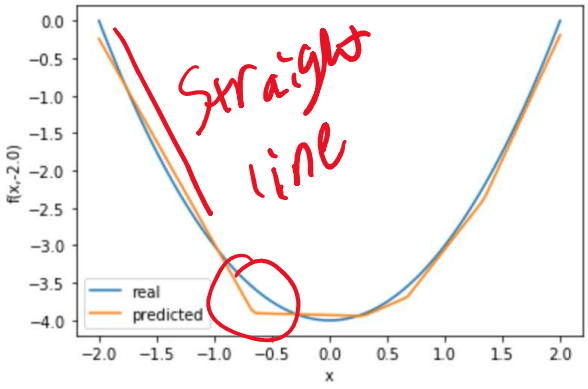


Figure 13(d): 2D section at y = -2

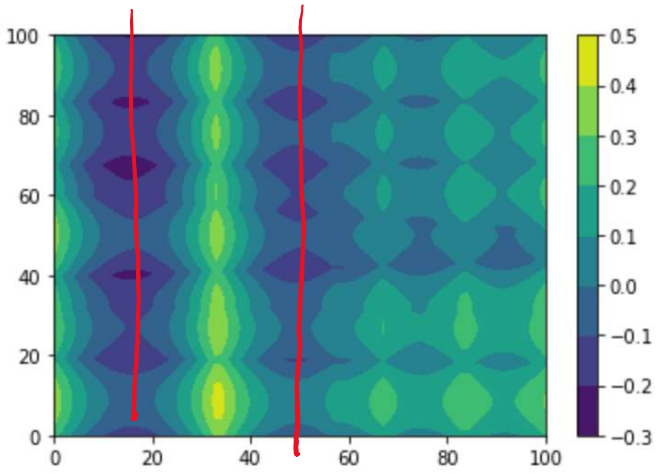


Figure 13(e): Topological Error Contour

2c. Code Run 2: Ackley function ($z = -20e^{-0.2\sqrt{0.5(x^2+y^2)}} - e^{0.5(\cos \pi x + cos\pi\ )} + 20$)

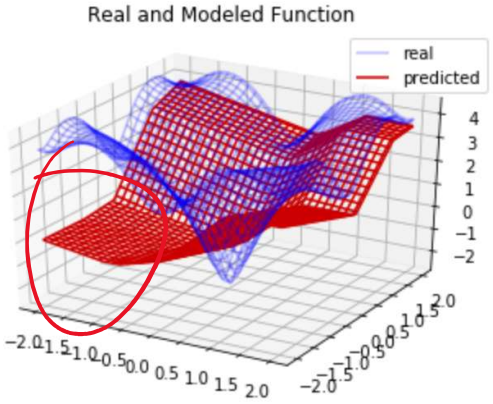


Figure 14(a): 3D Mesh grid Real vs Predicted

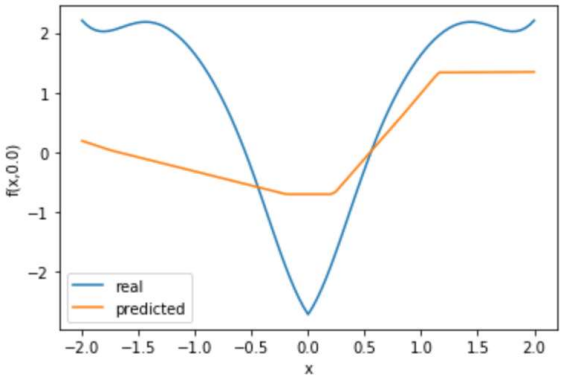


Figure 14(b): 2D section at y = 0

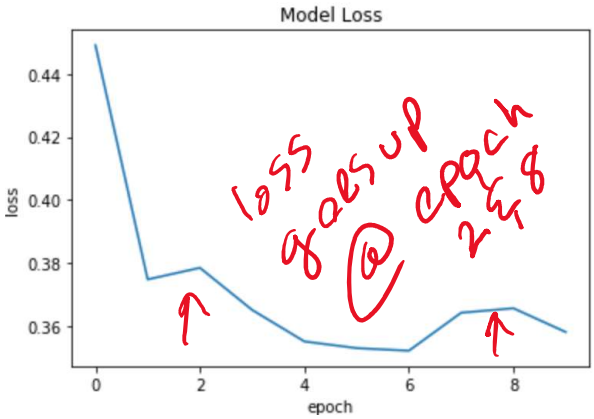


Figure 14(c): Model Loss vs Epoch

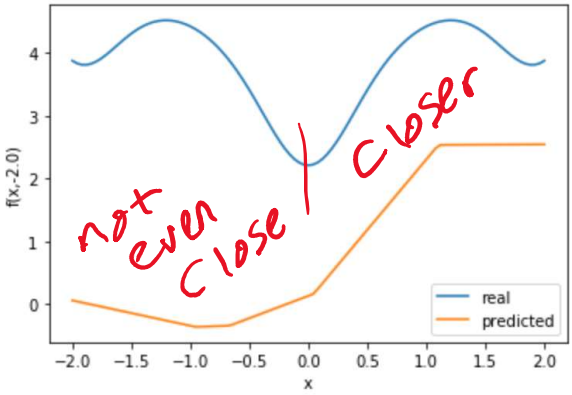


Figure 14(d): 2D section at y = -2

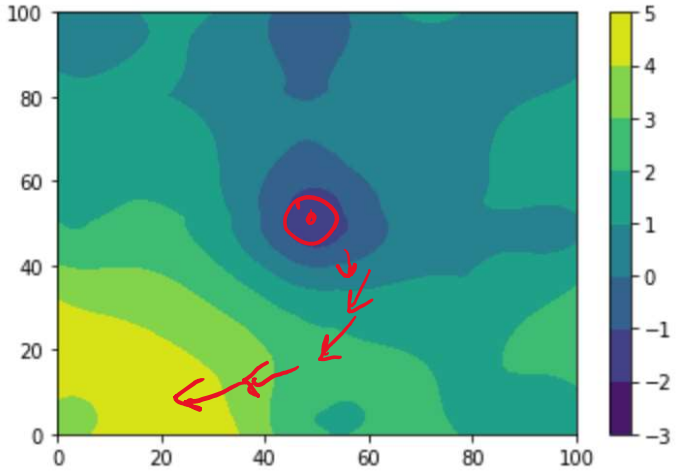


Figure 14(e): Topological Error Contour

2d. Comparison of Sigmoid vs ReLU Activation Functions:

The Sigmoid Activation Function (AF) is much more accurate than the ReLU AF for both z functions. The predicted z function, by the sigmoid AF hidden Layer, follows the real z function more closely for both z functions. Figures 5 and 11 show the sigmoid AF (for simple z and Ackley z functions respectively) whereas Figures 13 and 14 show the ReLU AF (for simple z and Ackley z functions respectively).

Between Figures 5 and 13 the main difference is the ReLU AF shapes the simple z function to have corners and straight lines to regress among the points that make up the simple z function real function. So the predicted z function looks trapezoidal. And the error is not as spread among the 3D object in Figure 5(d). The error is more focused linearly across the x-axis in Figure 13(d).

The obvious differences are between Figures 11 and 14 where the predicted z function is at least shaped like the real z function for the cross sections in Figures 11(b), 11(d). But the, again Straight lines and corners, predicted z function doesn't even look like the real function and doesn't even touch the real function in 14(b), 14(d) respectively. The Model Loss seems to increase at epochs 2 and 8 as shown in Figure 14(c). Again, the Topological error changes from a 3D well-spaced error to a centralized y=x linear shift.

3. In the Model Setup code block reverse the activation function back to sigmoid and set the number of neurons of the hidden layer from 10 to 100. Train the new models for both z. How do regression results compare to the regression in Task 1? Do not worry if results are poor. Sometimes a model can be too complex for a given task.

3a. Number of neurons in hidden layer set to 100:

```
model.add(Dense(100, activation='sigmoid', input_shape=(2,))
```

Figure 15: Changes Made for 100 Neurons Sigmoid AF

```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_15 (Dense)             (None, 100)               300
_________________________________________________________________
dense_16 (Dense)             (None, 1)                 101
=================================================================
Total params: 401
Trainable params: 401
Non-trainable params: 0
_________________________________________________________________
```

Figure 16: Model Setup Table

```
Epoch 1/10
10201/10201 [==============================] - 1s 120us/step - loss: 9.4327
Epoch 2/10
10201/10201 [==============================] - 1s 106us/step - loss: 1.7781
Epoch 3/10
10201/10201 [==============================] - 1s 100us/step - loss: 1.5419
Epoch 4/10
10201/10201 [==============================] - 1s 102us/step - loss: 1.5422
Epoch 5/10
10201/10201 [==============================] - 1s 101us/step - loss: 1.5419
Epoch 6/10
10201/10201 [==============================] - 1s 101us/step - loss: 1.5403
Epoch 7/10
10201/10201 [==============================] - 1s 101us/step - loss: 1.5375
Epoch 8/10
10201/10201 [==============================] - 1s 97us/step - loss: 1.5380
Epoch 9/10
10201/10201 [==============================] - 1s 101us/step - loss: 1.5354
Epoch 10/10
10201/10201 [==============================] - 1s 100us/step - loss: 1.5355
```

Figure 17: Model Training for 100 Neurons

3b. Code Run 1: Simple Z function ($z = x^2 - y^2$)

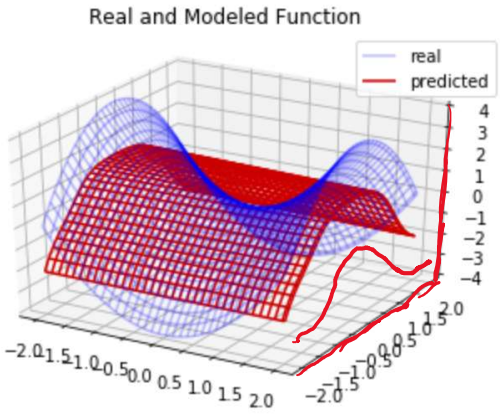


Figure 18(a): 3D Mesh grid Real vs Predicted

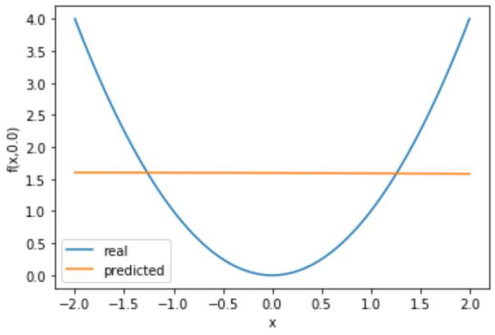


Figure 18(b): 2D section at y = 0

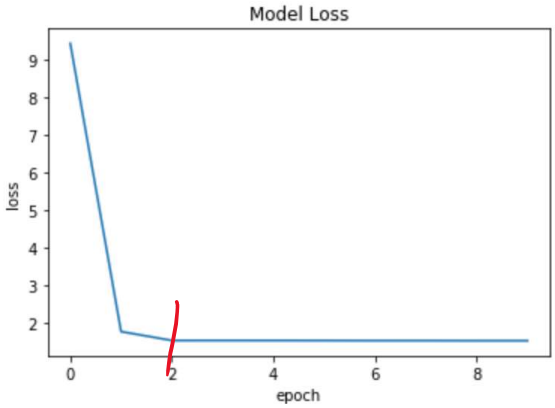


Figure 18(c): Model Loss vs Epoch

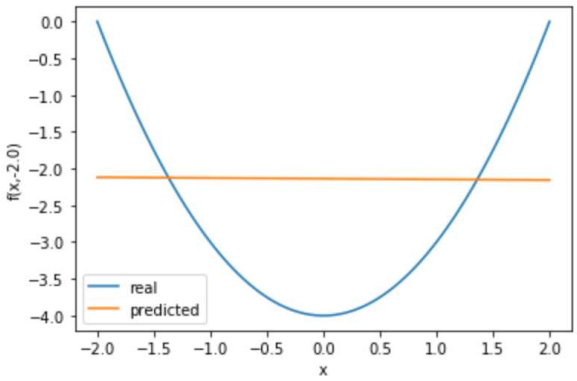


Figure 18(d): 2D section at y = -2

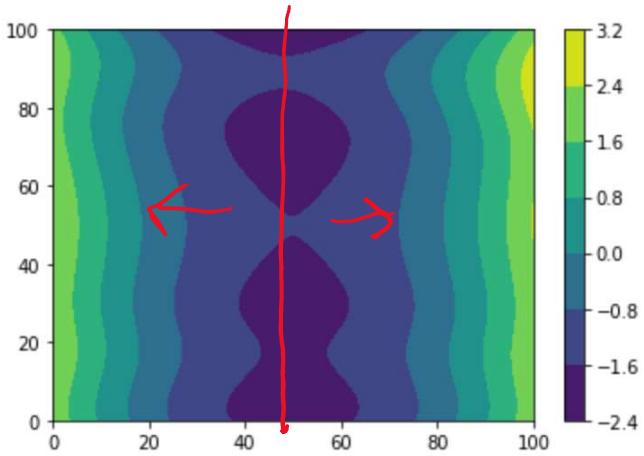


Figure 18(e): Topological Error Contour

3c. Code Run 2: Ackley function ($z = -20e^{-0.2\sqrt{0.5(x^2+y^2)}} - e^{0.5(\cos \pi x + cos \pi y)} + 20$)

```
10201/10201 [==============================] - 1s 113us/step - loss: 25.9450
Epoch 2/10
10201/10201 [==============================] - 1s 96us/step - loss: 2.0369
Epoch 3/10
10201/10201 [==============================] - 1s 90us/step - loss: 2.0055
Epoch 4/10
10201/10201 [==============================] - 1s 91us/step - loss: 1.9920
Epoch 5/10
10201/10201 [==============================] - 1s 90us/step - loss: 1.9876
Epoch 6/10
10201/10201 [==============================] - 1s 96us/step - loss: 1.9777
Epoch 7/10
10201/10201 [==============================] - 1s 97us/step - loss: 1.9718
Epoch 8/10
10201/10201 [==============================] - 1s 102us/step - loss: 1.9736
Epoch 9/10
10201/10201 [==============================] - 1s 102us/step - loss: 1.9743
Epoch 10/10
10201/10201 [==============================] - 1s 96us/step - loss: 1.9661
```

Figure 19: Model Training for 100 Neurons

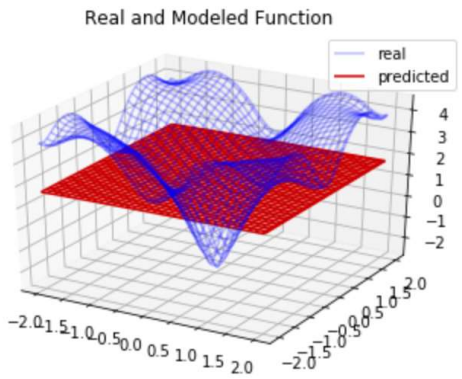


Figure 20(a): 3D Mesh grid Real vs Predicted

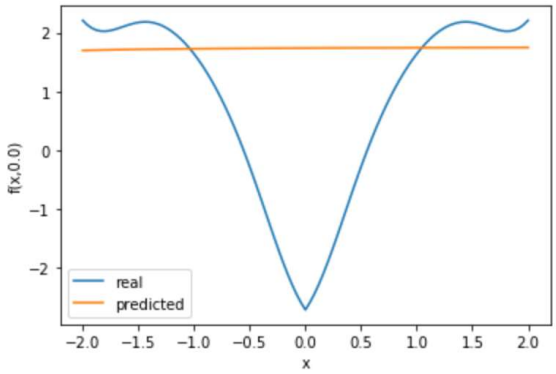


Figure 20(b): 2D section at y = 0

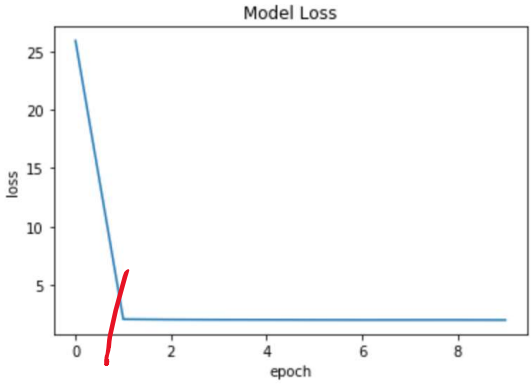


Figure 20(c): Model Loss vs Epoch

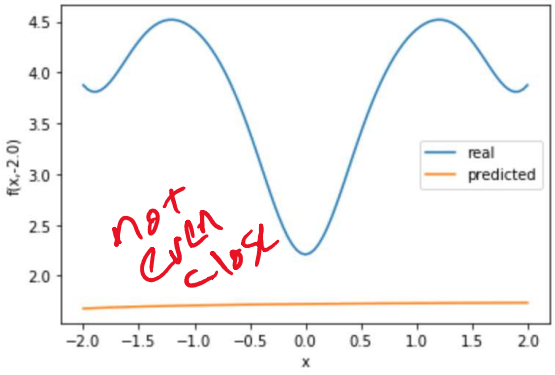


Figure 20(d): 2D section at y = -2

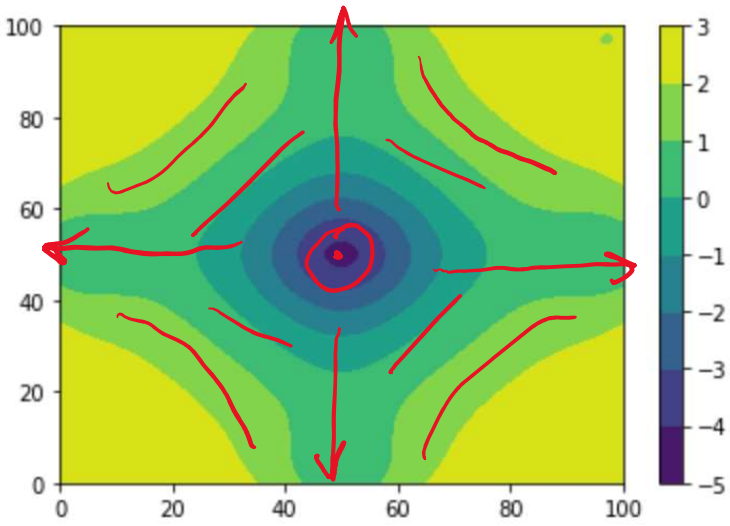


Figure 20(e): Topological Error Contour

3d. Compare Results to Results from Task 1:

From Figure 5(a) and 18(a) there is a huge difference between the predicted z for the simple z function. The predicted z follows the saddle like structure in 5(a) whereas the predicted z of 18(a) looks to be a 3D generalization of the overall structure of the real saddle hyperbolic paraboloid (looks like a Pringles Chip). Figure 20(a) however has a z predicted that is just a simple plane. The structure is too complex for 100 neurons. It's like if you under-sampled a continuous time signal and only got the bits of the continuous time signal that were flat, where there is still a lot of different important points that are missed between sample points by the general quantized structure of the digital procession.

The x-z cross sectional plane looks more similar in 18(a) than the x-y cross sectional planes of 18(b), 18(d). The predicted z cross sections in 18(b), 18(d) look nothing like the real z cross sections, whereas the cross sections of z predicted and z real in 5(b), 5(d) have a similar $y = x^2$ functional line. The same can be said about Figure 20(b), 20(d) where the x-y cross sectional planes of the z predicted lines just don't look anything like the z real lines.

Figures 18(c) and 20(c) seem to stop at a high loss of 2 at the 2nd epoch and doesn't get any better. Whereas the loss in Figure 5(c) seems to still have a descent to lower loss with increasing epoch.

Figure 18(d) has a central error along x=0 (mesh point = 50) with slightly less error expanding toward the high bounds of the range of x $[-2 \leq x \leq 2]$ or in mesh points $[0 \leq x \leq 100]$. Figure 20(d) has a central error at the point (x,y)=(0,0) and spreads to lower error in a cross shaped fashion along x=0 and y=0 with error contours looking like negative exponentials on the outside and diamond shaped structures along the cross alignment closer to the central high error. The error contours seem to progress toward a circulation around the point (x,y)=(0,0). In Figure 5(d) the error is spread along diagonal checkered sections of high and low error where the predicted z over and underestimates the real z several times. Very closely mimicking the real z.

4. Change the number of neurons in the first layer back to 10. Add another Dense layer with 10 neurons before the output layer and keep the sigmoid function. Train this new model for 25 epochs and visualize results.

4a. Two dense layers of 10 neurons and 25 epochs:

```
model.add(Dense(10, activation='sigmoid', input_shape=(2,)))
model.add(Dense(10, activation='sigmoid',))
```

Figure 21: Changes Made for Another Dense Layer of 10 Neurons

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 10)                30
_________________________________________________________________
dense_2 (Dense)              (None, 10)                110
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 11
=================================================================
Total params: 151
Trainable params: 151
Non-trainable params: 0
_________________________________________________________________
```

Figure 22: Model Setup Table

```
history = model.fit(xy_col, z_col, epochs=25, batch_size=10, verbose=1)
```

Figure 23: Changes Made for 25 Epochs

```
10201/10201 [==============================] - 1s 122us/step - loss: 14.3941
Epoch 2/25
10201/10201 [==============================] - 1s 105us/step - loss: 1.4946
Epoch 3/25
10201/10201 [==============================] - 1s 103us/step - loss: 0.1653
Epoch 4/25
10201/10201 [==============================] - 1s 103us/step - loss: 0.0726
Epoch 5/25
10201/10201 [==============================] - 1s 106us/step - loss: 0.0289
Epoch 6/25
10201/10201 [==============================] - 1s 101us/step - loss: 0.0191
Epoch 7/25
10201/10201 [==============================] - 1s 106us/step - loss: 0.0170
Epoch 8/25
10201/10201 [==============================] - 1s 99us/step - loss: 0.0162
Epoch 9/25
10201/10201 [==============================] - 1s 103us/step - loss: 0.0155
Epoch 10/25
10201/10201 [==============================] - 1s 99us/step - loss: 0.0093
Epoch 11/25
10201/10201 [==============================] - 1s 106us/step - loss: 0.0104
Epoch 12/25
10201/10201 [==============================] - 1s 106us/step - loss: 0.0107
Epoch 13/25
10201/10201 [==============================] - 1s 105us/step - loss: 0.0080
Epoch 14/25
10201/10201 [==============================] - 1s 104us/step - loss: 0.0090
Epoch 15/25
```

Figure 24: Model Training for Epochs 1-14

```
Epoch 15/25
10201/10201 [==============================] - 1s 106us/step - loss: 0.0089
Epoch 16/25
10201/10201 [==============================] - 1s 106us/step - loss: 0.0105
Epoch 17/25
10201/10201 [==============================] - 1s 106us/step - loss: 0.0074
Epoch 18/25
10201/10201 [==============================] - 1s 105us/step - loss: 0.0083
Epoch 19/25
10201/10201 [==============================] - 1s 107us/step - loss: 0.0075
Epoch 20/25
10201/10201 [==============================] - 1s 106us/step - loss: 0.0070
Epoch 21/25
10201/10201 [==============================] - 1s 106us/step - loss: 0.0078
Epoch 22/25
10201/10201 [==============================] - 1s 106us/step - loss: 0.0076
Epoch 23/25
10201/10201 [==============================] - 1s 107us/step - loss: 0.0086
Epoch 24/25
10201/10201 [==============================] - 1s 107us/step - loss: 0.0061
Epoch 25/25
10201/10201 [==============================] - 1s 110us/step - loss: 0.0077
```

Figure 25: Model Training for Epochs 15-25

4b. Visualize Results:

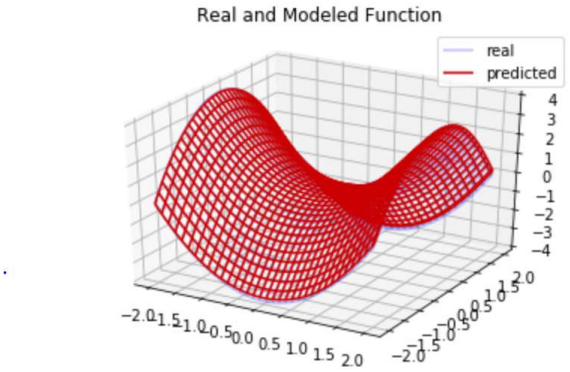


Figure 26(a): 3D Mesh grid Real vs Predicted

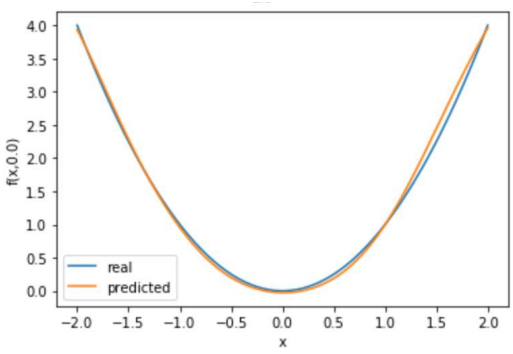


Figure 26(b): 2D section at y = 0

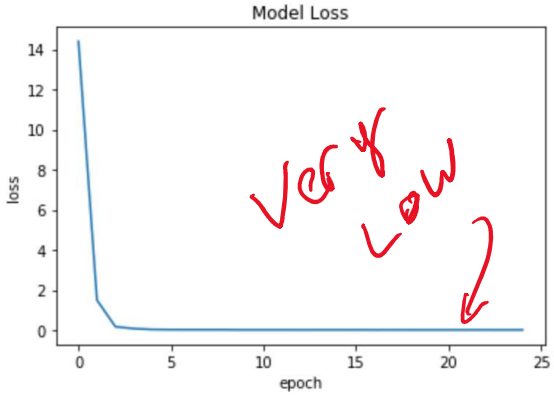


Figure 26(c): Model Loss vs Epoch

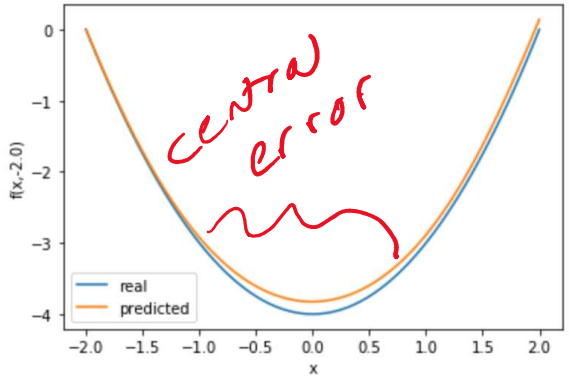


Figure 26(d): 2D section at y = -2

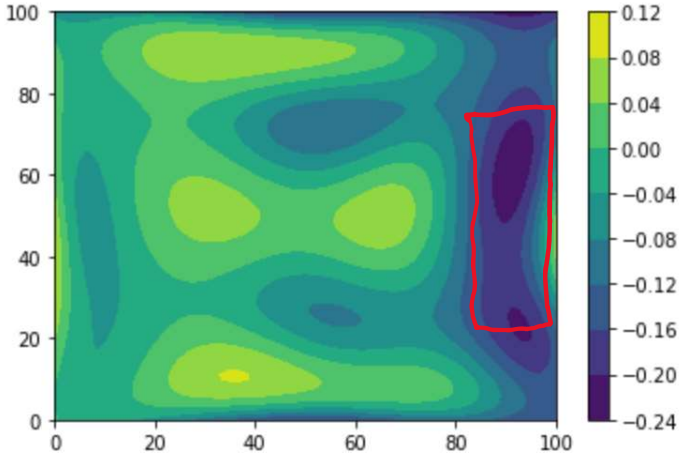


Figure 26(e): Topological Error Contour

Predicted z mesh grid seems very close to the real z in Figure 26(a). Cross sections are extremely close but get worse when closer to x=0. The loss looks really low after 25 epochs in Figure 26(c). The error seems centralized around middle of mesh y and x=90 in Figure 26 (e).

5. Change the activation function from sigmoid to tanh and rerun Task 2. How do the results compare?

5a. Change AF from sigmoid to tanh:

```
model.add(Dense(10, activation='tanh', input_shape=(2,)))
```

Figure 27: Changes made for tanh AF

5b. Return to Task 2:

Code Run 1: Simple Z function ($z = x^2 - y^2$)

For the sake of simplicity, only visualizations will be shown for simple observable differences between hidden layer activation functions.

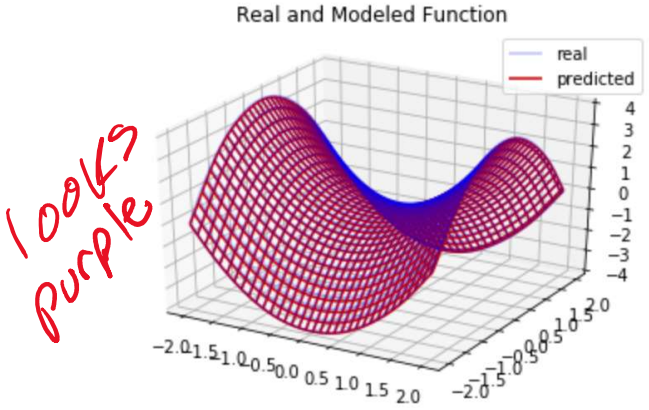


Figure 28(a): 3D Mesh grid Real vs Predicted

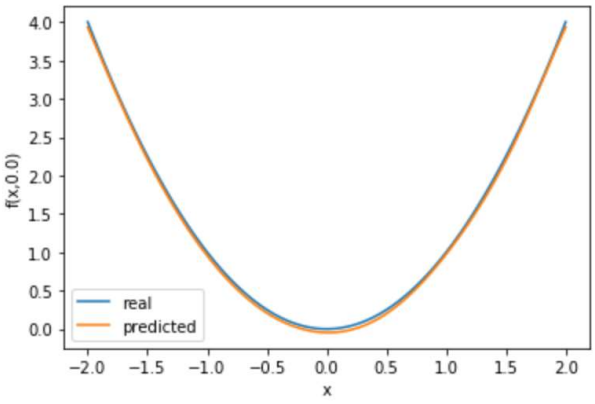


Figure 28(b): 2D section at y = 0

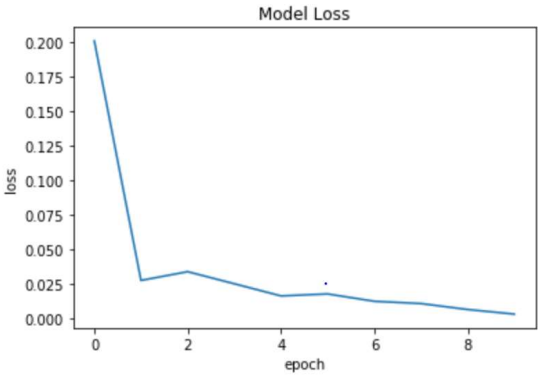


Figure 28(c): Model Loss vs Epoch

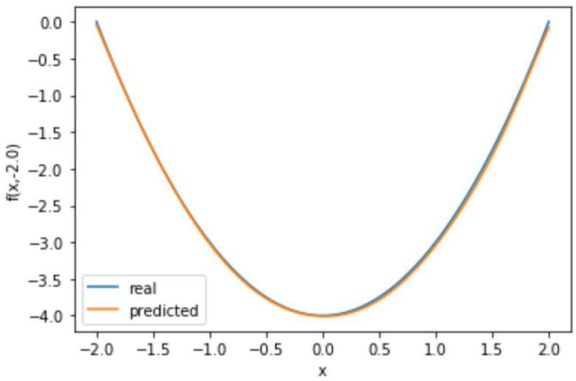


Figure 28(d): 2D section at y = -2

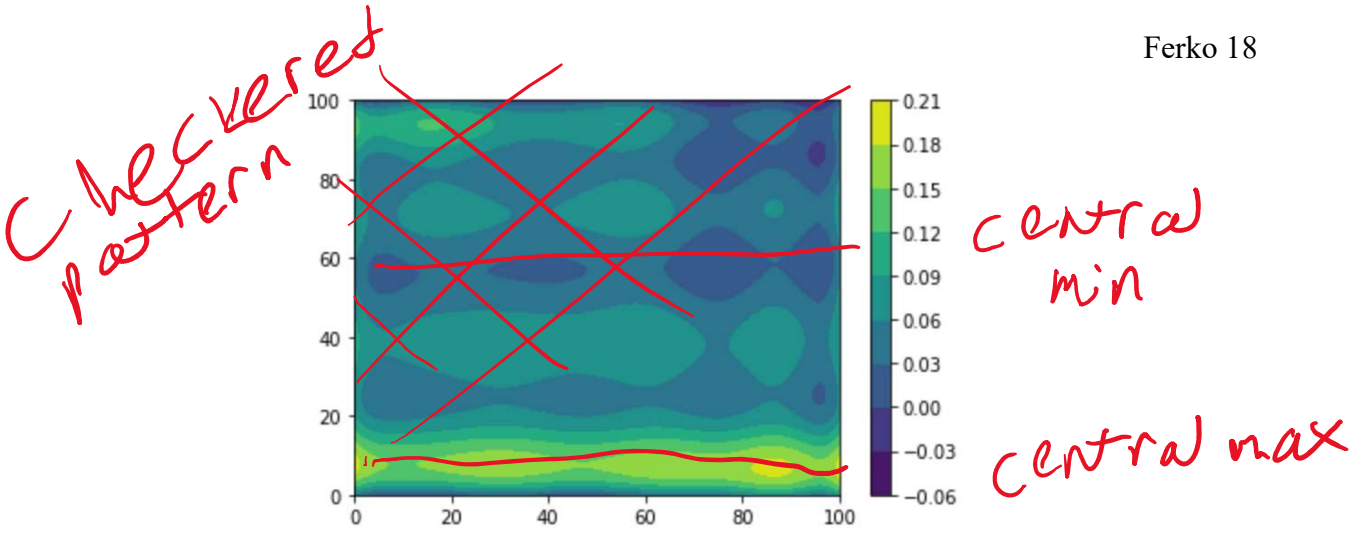


Figure 28(e): Topological Error Contour

Code Run 2: Ackley function ($z = -20e^{-0.2\sqrt{0.5(x^2+y^2)}} - e^{0.5(\cos \pi x + cos\pi y)}+20$)

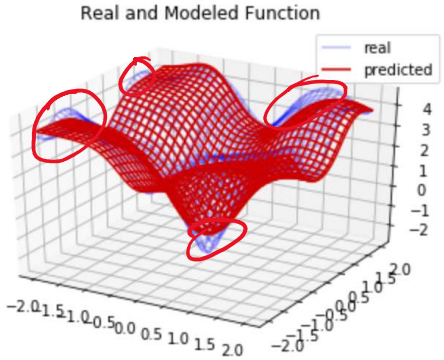


Figure 29(a): 3D Mesh grid Real vs Predicted

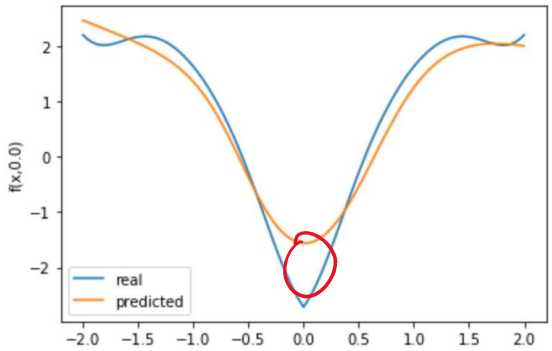


Figure 29(b): 2D section at y = 0

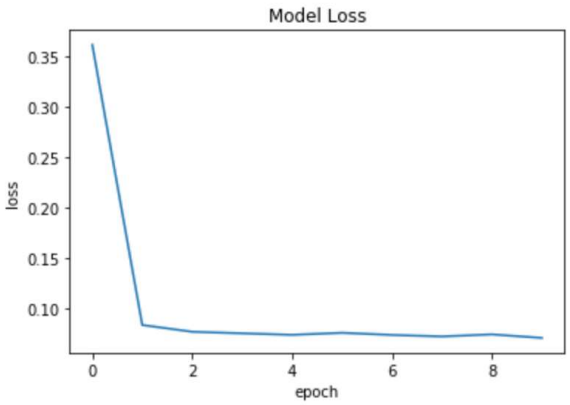


Figure 29(c): Model Loss vs Epoch

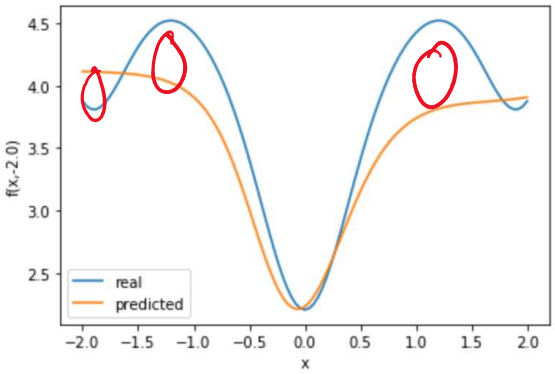


Figure 29(d): 2D section at y = -2

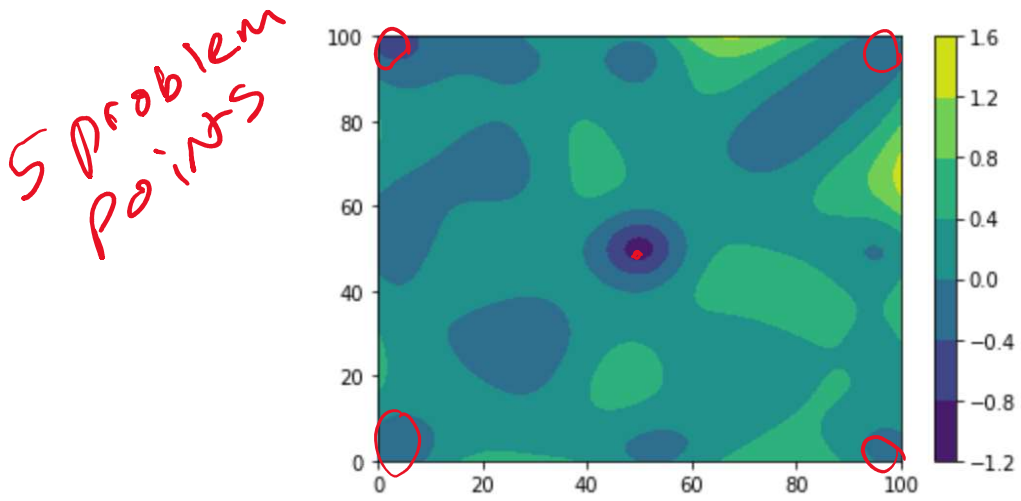


Figure 29(e): Topological Error Contour

Comparison of Sigmoid vs Tanh vs ReLU Activation Functions (AF):

The hyperbolic tangent (tanh) AF seems the most accurate among the three AF. Where accuracy (A_x *where* x *is an AF*) of the AFs are as follows: $A_{tanh} > A_{sigmoid} > A_{ReLU}$. Accuracy meaning the predicted z function has the lowest error or loss and could most closely mimic the real z function for supervised learning recognition of the real z pattern.

Figure 28(b), 28(d) show a very close match between the predicted and real z cross sections. The predicted z is so close that the color of the mesh grid 3D model looks purple because of the hue of blue and red so closely overlapping each other. There is a similar error diagonal checkered structure that was seen in Figure 5(e) to the checkered error structure in Figure 28(e), but the error seems to dissipate.

Again the Ackley function proposes similar prediction error issues. Much of the tanh AF Figure 29(a) – Figure 29(d) looks just like the sigmoid AF Figure 5(a) – Figure 5(d). The corners and center have huge gaps between the real and predicted. Predicted z values follow the real z Ackley function closely between the four corners an center (5 problem points).

6. Train the model with random noise uniformly distributed between [-0.1 and +0.1] superimposed on each z function in the data generation code block. The noise is only added for the training part, but the final model needs to be tested on the accurate z values as defined for each function.

6a. Add Random Noise Uniformly Distributed between [-0.1 and +0.1] superimposed:

Under imports a random import was added:

```
import random
```

Figure 30: Import Random Added

Under Data Generation noise is generated random and uniformly between [-0.1, +0.1] and added onto the z function both simple and Ackley function. The noisy z function is just for model training. Therefore, another z function z2 is defined for the normal simple z and Ackley z functions. And the z2 function is for the final model testing accurate z values:

```
noise = np.random.uniform(-0.1,0.1)
z  = simple_function(xv, yv) + noise
#z  = ackley_function(xv, yv) + noise
z2  = simple_function(xv, yv)
#z2  = ackley_function(xv, yv)
```

Figure 31: Noise Definition and Z Definition Change

```
#matplotlib inline

z_pred = model.predict(xy_col)

fig = plt.figure()

#Create subplot with 3d axis for graphing regression inputs and outputs
ax = fig.add_subplot(111, projection='3d')
ax.plot_wireframe(xv, yv, z2, colors=(0,0,1,0.25))

ax.plot_wireframe(xv, yv, z_pred.reshape(z2.shape[0],z2.shape[1]), colors=(0.8, 0, 0, 1))
ax.title.set_text('Real and Modeled Function')
ax.legend(['real', 'predicted'])

fig1 = plt.figure()
#Create subplot of of cross section along y=2
ax2 = fig1.add_subplot(111)
ax2.plot(xv[50,:], z2[50,:])

ax2.plot(xv[50,:], z_pred.reshape(z2.shape[0],z2.shape[1])[50,:])
ax2.legend(['real', 'predicted'])
plt.ylabel('f(x,'+str(y[50])+')'); plt.xlabel('x')

fig2 = plt.figure()
#Create subplot of of cross section along y=0
ax3 = fig2.add_subplot(111)
ax3.plot(xv[0,:], z2[0,:])

ax3.plot(xv[0,:], z_pred.reshape(z2.shape[0],z2.shape[1])[0,:])
ax3.legend(['real', 'predicted'])
plt.ylabel('f(x,'+str(y[0])+')'); plt.xlabel('x')
```

Figure 32(a): Changes Made for z2 in Final Model Under Visualization

```
fig4 = plt.figure()
#Create subplot of loss over training epochs.
ax4 = fig4.add_subplot(111)
ax4.plot(history.history['loss'])
ax4.title.set_text('Model Loss')
plt.ylabel('loss'); plt.xlabel('epoch')

fig5 = plt.figure()
z_pred=z_pred.reshape(z2.shape[0],z2.shape[1])
z2=z2.reshape(z2.shape[0],z2.shape[1])
#result = [[np.sqrt(np.abs((z2[i][j]**2) - (z_pred[i][j]**2)))  for j in range(len(z_pred[0]))] for i in range(len(z_pred[
result = [[(z2[i][j]) - (z_pred[i][j])  for j in range(len(z_pred[0]))] for i in range(len(z_pred[1]))]
cf = plt.contourf(result)
fig5.colorbar(cf)


plt.show()
```

Figure 32(b): Changes Made for z2 in Final Model Under Visualization

6b. Code Run 1: Simple Z function ($z = x^2 - y^2$)

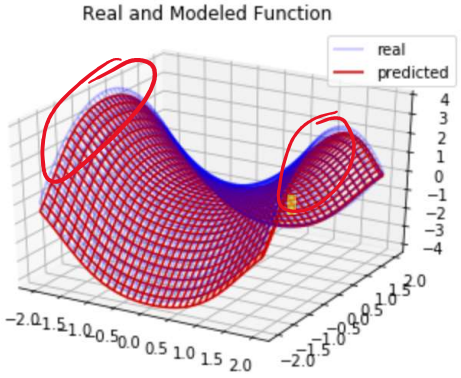


Figure 33(a): 3D Mesh grid Real vs Predicted

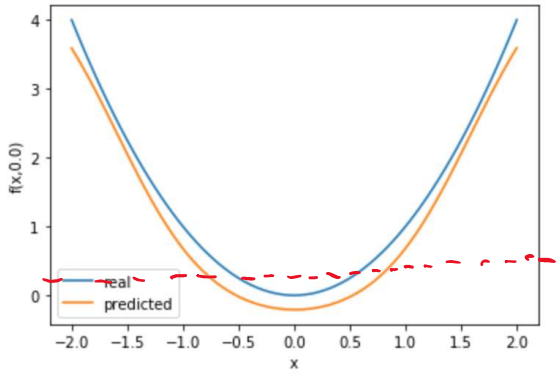


Figure 33(b): 2D section at y = 0

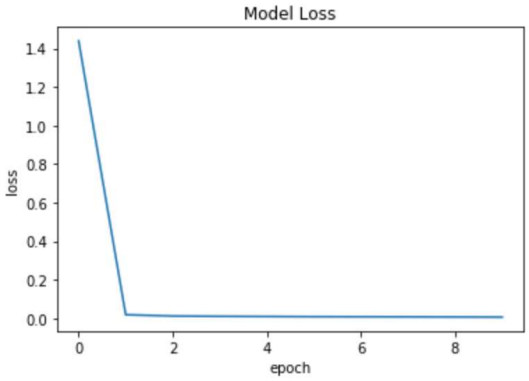


Figure 33(c): Model Loss vs Epoch

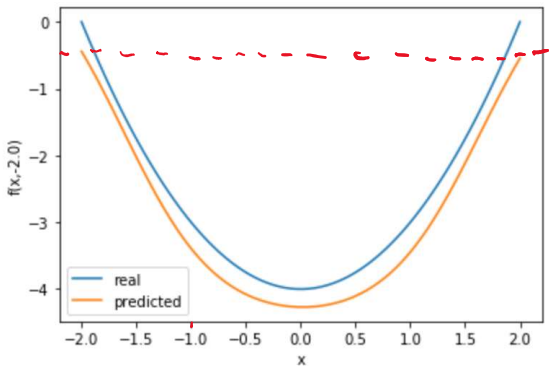


Figure 33(d): 2D section at y = -2

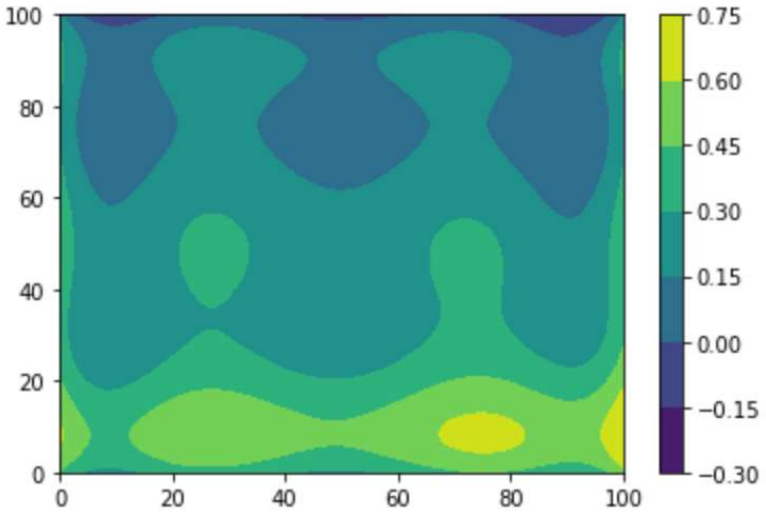


Figure 33(e): Topological Error Contour

6c. Code Run 2: Ackley function ($z = -20e^{-0.2\sqrt{0.5(x^2+y^2)}} - e^{0.5(\cos \pi x + cos \pi y)} + 20$)

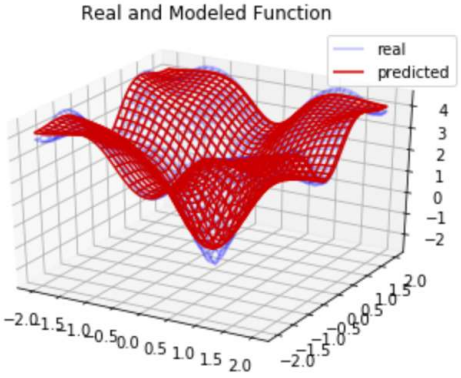


Figure 34(a): 3D Mesh grid Real vs Predicted

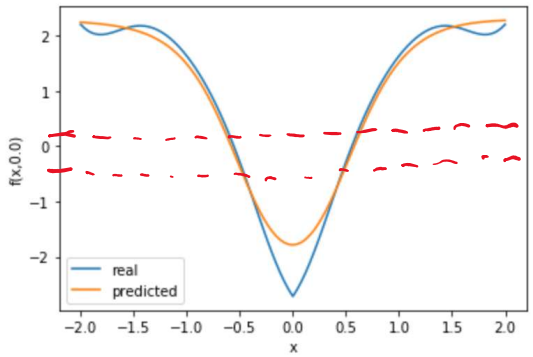


Figure 34(b): 2D section at y = 0

↓noise model training Shift Δz = -0.2

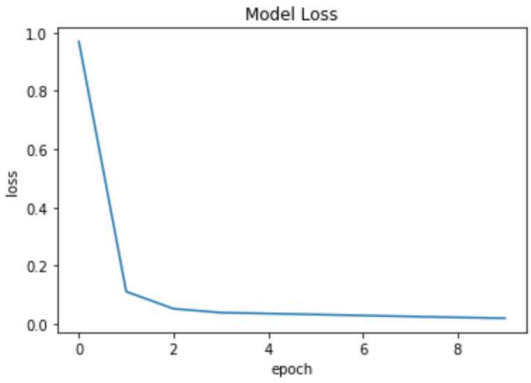


Figure 34(c): Model Loss vs Epoch

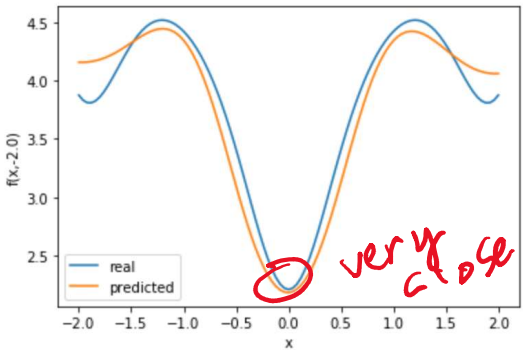


Figure 34(d): 2D section at y = -2

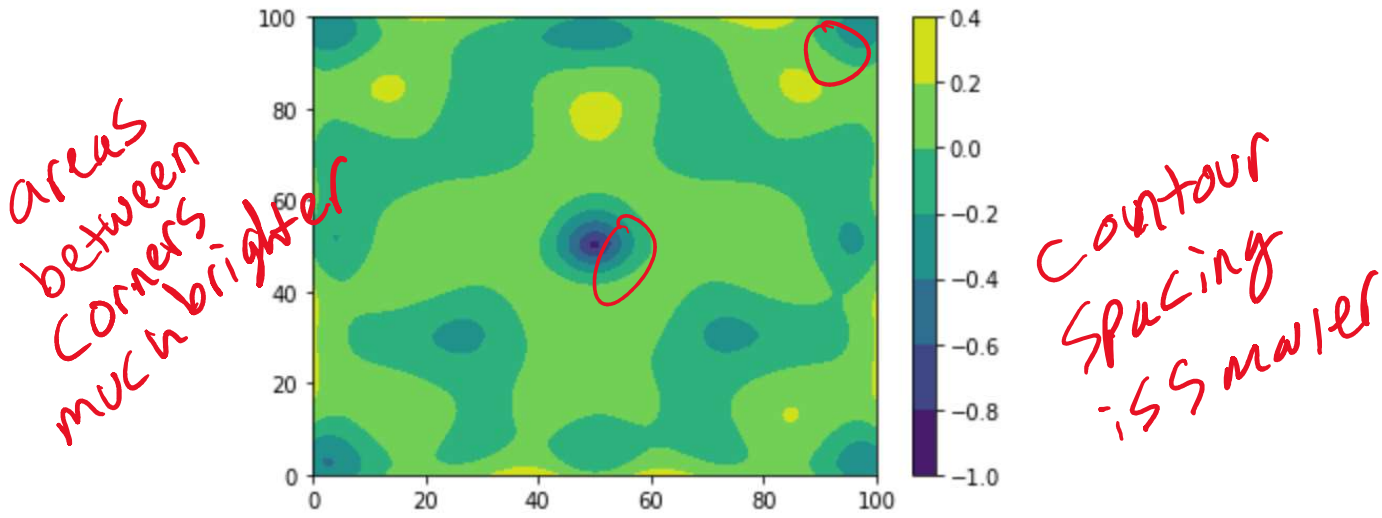


Figure 34(e): Topological Error Contour

The noise model training makes the predicted z values stray away from the real z function in the simple z function. An error increase may be observed by comparing Figure 5(e) to Figure 33(e). By drawing the noise range between [-0.1<z<+0.1] on Figure 33(b), 33(d) we can see the gap shift is equal to the noise range (i.e. $\Delta z = 0.2$).

In the case of the Ackley z function the opposite effect may be observed. Where the gap shift presented by the model training along the z axis helps predicted z values shift closer to the real z values. Figure 34(a) looks much closer on the corners than Figure 11(a). More well seen by the close contours on the error contour map of Figure 34(e). In Figures 34(b), 34(d) the predicted cross sections and real z cross sections are much closer! Figures 11(b), 11(d) don't utilize this model training gap shift and therefore the cross sections don't align nearly as well.

Extra Work:

For an extra experimentation I wanted to see if the gap shift worked for a different set of cross sections. I changed the points at which the cross sections were made to 2 and 0 and here are the results:

Extra Work a. Changes Made:

```
fig1 = plt.figure()
#Create subplot of of cross section along y=2
ax2 = fig1.add_subplot(111)
ax2.plot(xv[100,:], z2[100,:])

ax2.plot(xv[100,:], z_pred.reshape(z2.shape[0],z2.shape[1])[100,:])
ax2.legend(['real', 'predicted'])
plt.ylabel('f(x,'+str(y[100])+')'); plt.xlabel('x')

fig2 = plt.figure()
#Create subplot of of cross section along y=0
ax3 = fig2.add_subplot(111)
ax3.plot(xv[50,:], z2[50,:])

ax3.plot(xv[50,:], z_pred.reshape(z2.shape[0],z2.shape[1])[50,:])
ax3.legend(['real', 'predicted'])
plt.ylabel('f(x,'+str(y[50])+')'); plt.xlabel('x')
```

Figure 35: Coding Changes Made for Mesh Strings

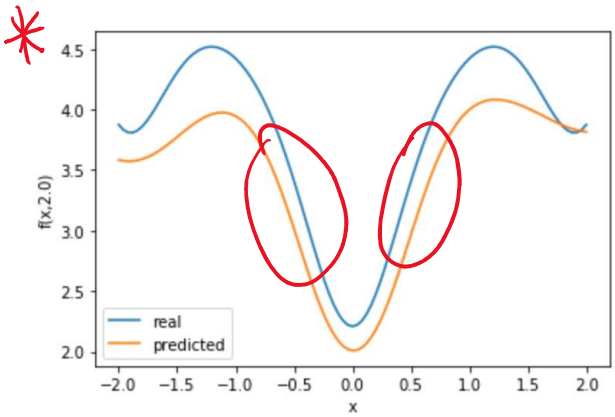


Figure 36(a): Cross Section at y=2

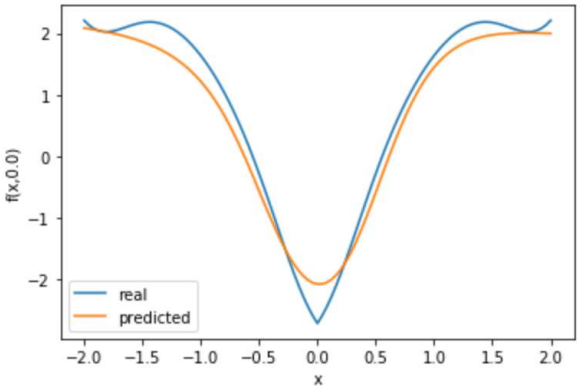


Figure 36(b): Cross Section at y=0

From Figure 36(a) the shift gap is worse at y=2 cross section. This means that as much as the noise helped in some cross sections, the noise worked against predicted z values and real z values coming closer together in other sections. This means that for all the other changes which have been made in Tasks 1 through 6 have been improving the model predictions in some cross-sectional areas, but then worsening the loss and error in some cross-sectional areas. To further prove this theory let us test the y=2 cross section for Task 1-5 adjustments.

Extra Work: Ackley z at y = 2 Effects

Extra Work: Task 1:

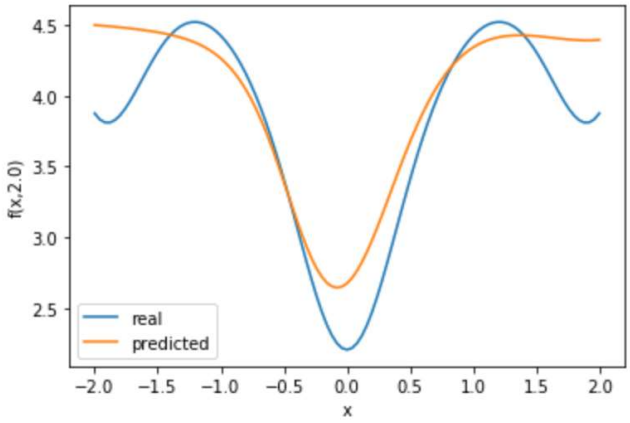


Figure 37: Cross Section at y=2

Extra Work: Task 2: ReLU

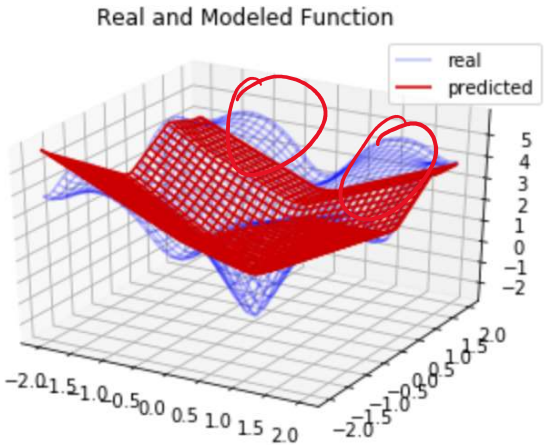


Figure 38(a): 3D Mesh grid Real vs Predicted

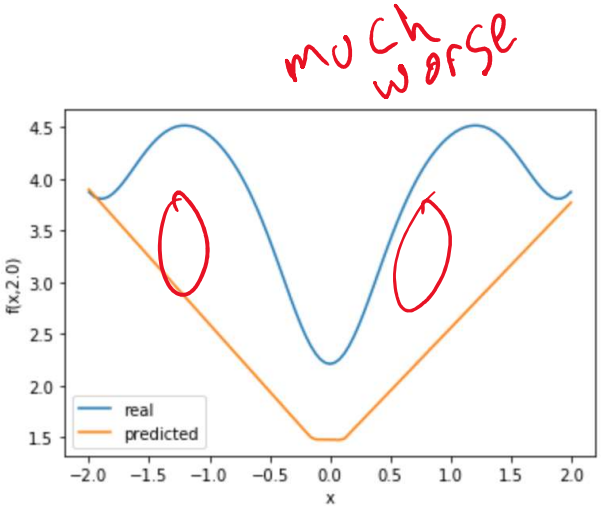


Figure 38(b): 2D section at y = 2

The ReLU AF works much worse for the y=2 2D Cross section, but the ReLU Af was bad by itself during this Task.

Extra Work: Task 3: 100 Neurons

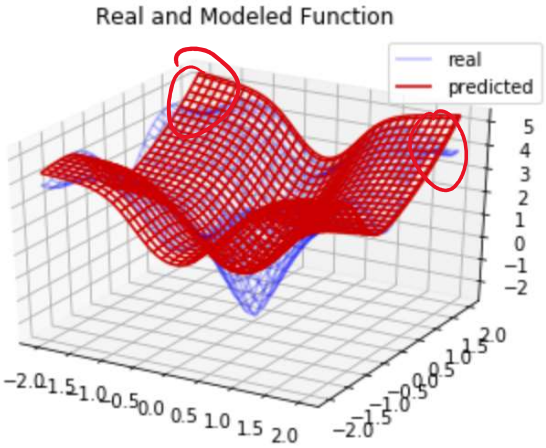


Figure 38(a): 3D Mesh grid Real vs Predicted

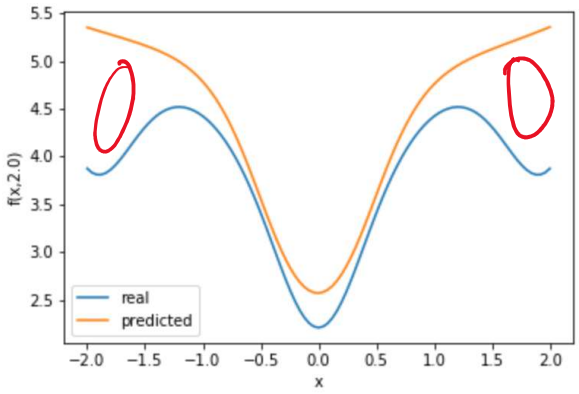


Figure 38(b): 2D section at y = 2

Again, the y=2 2D cross section has worse areas.

Extra Work: Task 4: 2 Dense Layers of 10 Neurons and 25 Epochs

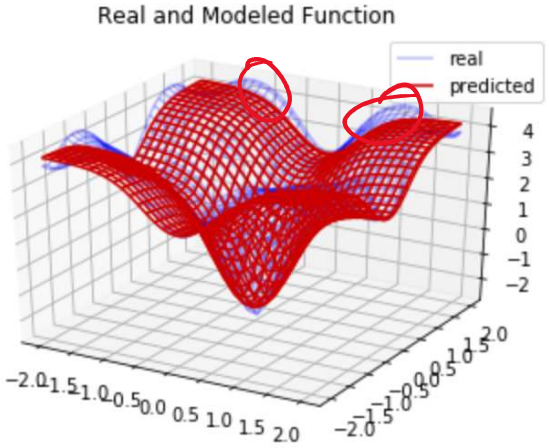


Figure 38(a): 3D Mesh grid Real vs Predicted

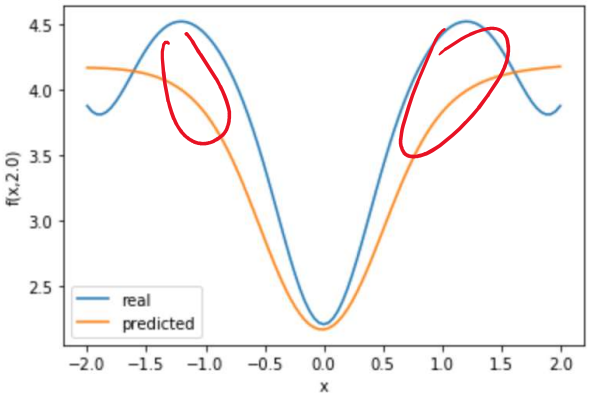


Figure 38(b): 2D section at y = 2

Again, the y=2 2D cross section has worse areas.

Extra Work: Task 5: 2 Dense Layers of 10 Neurons and 25 Epochs

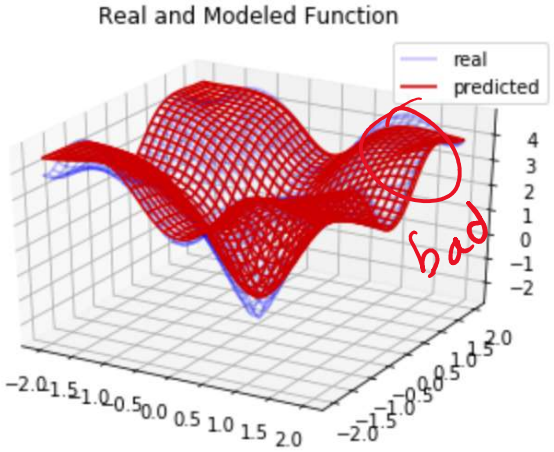


Figure 38(a): 3D Mesh grid Real vs Predicted

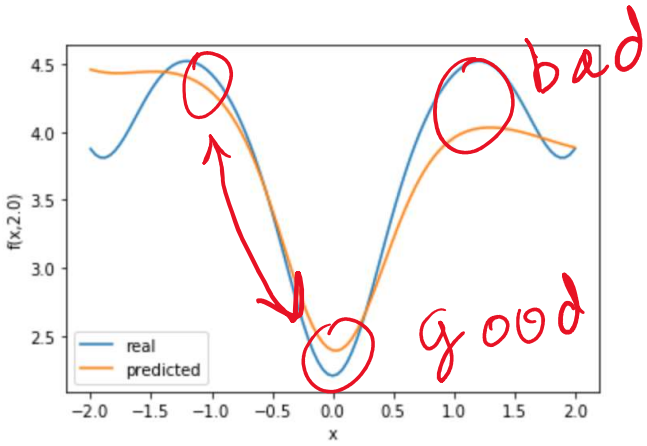


Figure 38(b): 2D section at y = 2

Again, the y=2 2D cross section has worse areas.

Therefore I propose that some 2D cross sections will be improved more than others. In Tasks 1-6 we analyzed ways to improve and worsen the 2D cross sections at y=-2 and y=0. By improving some cross sections, we worsen others. A lot of the time when improving by minimizing error we improve certain areas of the 3D model better than others. y=-2 and y=0 are improved vastly superior to the area y=2.