Introduction to Developing Neural Networks

SESUG 2025

David B. Horvath, MS, CCP

Presenter

David B. Horvath, CCP, MS

The Author can be contacted at:
504 Longbotham Drive, Aston PA 19014-2502, USA
Phone: 1-610-859-8826
Email: dhorvath@cobs.com (mailto:dhorvath@cobs.com)
Web: http://www.cobs.com/ (http://www.cobs.com/)
LI: http://www.linkedin.com/in/dbhorvath (http://www.linkedin.com/in/dbhorvath)

Abstract

Neural nets have been discussed in in SAS regional and international conferences since last century, This session will be a bit different in that it starts with the obligatory basic background and then goes into actual code. We will discuss what Machine Learning can and cannot solve. There will be more discussion of the how results are produced and less about the overall concepts.

- What is a neuron?
- How does a neural net model a human neuron?
- How does that work with data?
- What does it NOT solve? Discuss the biggest problems with data.
- Examples

1. Working data
2. Why'd we get that result?
3. More Working data
4. Why'd we get **that** result?
5. Reviewing the code

My Background

- David is an IT Professional who has worked with various platforms since the 1980's with a variety of development and analysis tools.

- He has presented at PhilaSUG, SESUG, and SGF previously and has presented workshops and seminars in Australia, France, the US, Canada, and Oxford England (about the British Author Nevil Shute).
- He holds an undergraduate degree in Computer and Information Sciences from Temple University and a Masters in Organizational Dynamics from UPENN, and is pursuing a Masters in Data Sciences. He achieved the Certified Computing Professional designation with honors.
- Most of his career has been in consulting (although recently he has been in-house) in the Philadelphia PA area. He is currently in Data Analytics "Engineering" at a Regional Bank.
- He has several books to his credit (none SAS related) and is an Adjunct Instructor covering IT topics

Preliminaries

Import the packages we'll use and set up some functions to handle plotting

In [1]:

```
import matplotlib.pyplot as plt
import matplotlib as mpl
from matplotlib import cm
import numpy as np
from sklearn.metrics import mean_squared_error
import pandas as pd
%matplotlib inline
mpl.rc('axes', labelsize=12)
mpl.rc('xtick', labelsize=12)
mpl.rc('ytick', labelsize=12)
import os
import tensorflow as tf
from tensorflow import keras
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split


# Where to save the figures
PROJECT_ROOT_DIR = "."
FOLDER = "figures"
IMAGES_PATH = os.path.join(PROJECT_ROOT_DIR, FOLDER)
os.makedirs(IMAGES_PATH, exist_ok=True)

def save_fig(fig_id, tight_layout=True, fig_extension="png", resolution=300
    path = os.path.join(IMAGES_PATH, fig_id + "." + fig_extension)
    print("Saving figure", fig_id)
    if tight_layout:
        plt.tight_layout()
    plt.savefig(path, format=fig_extension, dpi=resolution)

def plotscatter3D(fit_x, fit_y, fit_z, scat_x, scat_y, scat_z ):

# Set up output filename
    if figure_name == "" : figure_out = "default"
    else : figure_out = figure_name

# Now set the axis values and set the size of the figure and prepare for fo
    fixed_axes = [-2, 2, -2, 2, -2, 2]

    fig = plt.figure(figsize = (12, 10))

    ax = [fig.add_subplot(221, projection = '3d'),
      fig.add_subplot(222, projection = '3d') ,
      fig.add_subplot(223, projection = '3d') ,
      fig.add_subplot(224, projection = '3d') ]

# Set the orientations of each subplot
    ax[0].view_init(0, 90) # upper left
    ax[1].view_init(40 ,0) # upper right
    ax[2].view_init(20, 90) # lower left
    ax[3].view_init(90, 90) # lower right

# And, finally draw the individual subplots with labels, limits, scatter (o
    for i in range (0, 4) :
        ax[i].set_xlabel("$X$", fontsize = 10)
        ax[i].set_ylabel("$Y$", fontsize = 10)
        ax[i].set_zlabel("$Z$", fontsize = 10)
```

```
        ax[i].set_xlim(fixed_axes[0:2])
        ax[i].set_ylim(fixed_axes[2:4])
        ax[i].set_zlim(fixed_axes[4:6])
        ax[i].scatter3D (scat_x, scat_y, scat_z, "b.")
        if fit_x[0] != None  : ax[i].scatter3D (fit_x, fit_y, fit_z, "r.")


    save_fig(figure_out)
    plt.show()


    return
```

Prepare The data

Read in the CSV, split into data and results, split off Train and Test, and then Scale the data

In [2]:

```
data = pd.read_csv("WUSS2024_ANN.csv")
# The "data" dataframe contains 3 columns -- an x, y, and z coordinate; x
# and z will become part
# of X_ and y will become y_ variables

arr = np.stack((data.iloc[:,0], data.iloc[:,-1]), axis = -1) # x and z

X_train, X_test, y_train, y_test = train_test_split(arr, data.iloc[:,1],
                                                    test_size = .20, random
scalerX = StandardScaler()
scalerY = StandardScaler()
X_train_scale = scalerX.fit_transform(X_train)
X_test_scale = scalerX.transform (X_test)

y_train = np.array(y_train).reshape(-1,1)
y_test = np.array(y_test).reshape(-1,1)

y_train_scale = scalerY.fit_transform(y_train)
y_train_scale = np.array(y_train_scale).reshape(-1)
y_test_scale = scalerY.transform(y_test)
y_test_scale = np.array(y_test_scale).reshape(-1)

np.random.seed(42)
tf.random.set_seed(42)
```

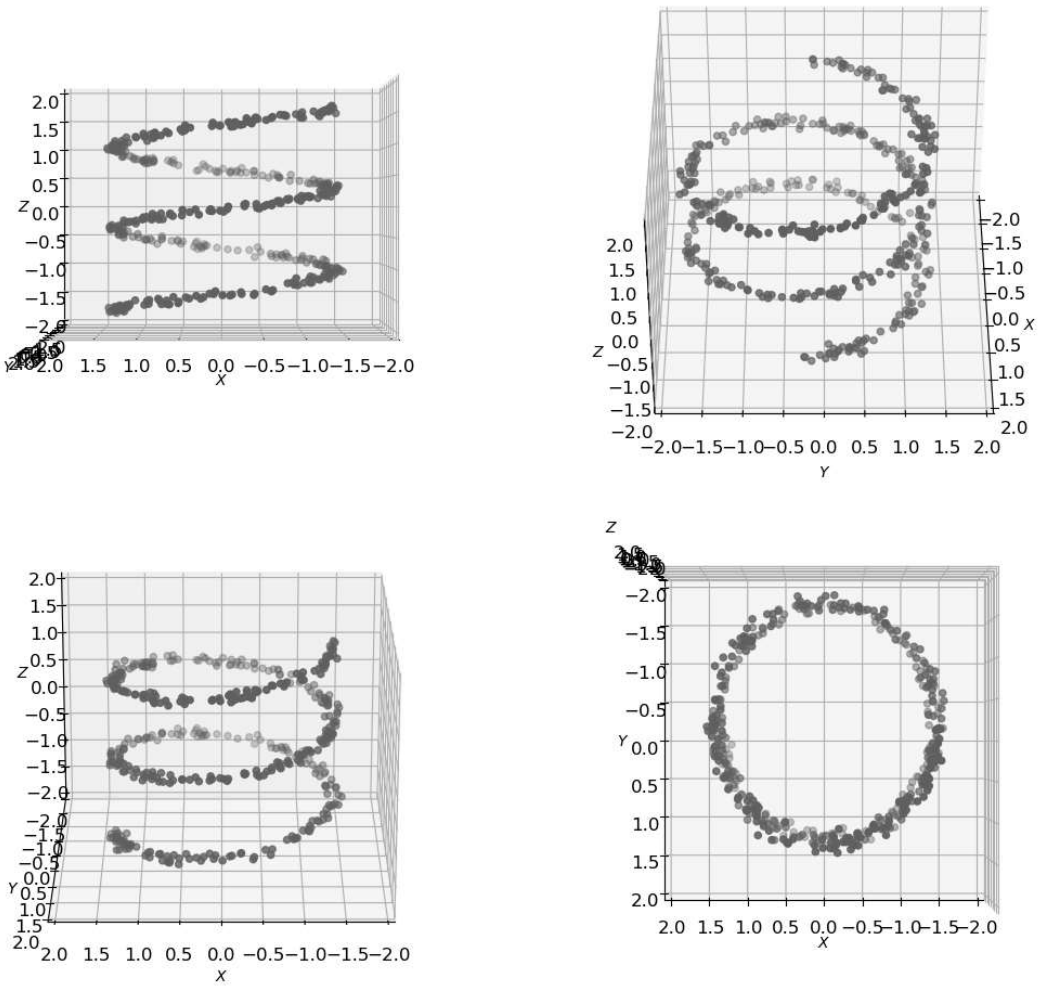
Plot Data

Plot the input data to get a feel for it

In [3]:
```
new_arr = np.stack((X_train_scale[:,0], y_train_scale, X_train_scale[:,1]),

train_df  = pd.DataFrame(new_arr, columns=['x', 'y', 'z'])
```

In [4]:
```
figure_name = "input_data_1"

plotscatter3D([None], None, None, X_train_scale[:,0], y_train_scale, X_trai
```

```
Saving figure input_data_1
```



In [ ]:
```

```

Explore 3D Sinusoidal Data with Artifical Neural Networks

In [5]:

```
tf.random.set_seed(42)

normalizer = tf.keras.layers.Normalization(axis = -1)
normalizer.adapt(np.array(X_train_scale))

model = keras.models.Sequential([
   normalizer,
#    keras.layers.Input( shape = X_train_scale.shape[1:]),
#    keras.layers.Dense(2, activation='relu', input_shape = X_train_scale.s
    keras.layers.Dense(150, activation='relu'),
    keras.layers.Dense(150, activation='relu'),
    keras.layers.Dense(150, activation='relu'),
    keras.layers.Dense(150, activation='relu'),
    keras.layers.Dense(150, activation='relu'),
    keras.layers.Dense(150, activation='relu'),
    keras.layers.Dense(150, activation='relu'),
    keras.layers.Dense(150, activation='relu'),
    keras.layers.Dense(150, activation='relu'),
    keras.layers.Dense(150, activation='relu'),
    keras.layers.Dense(1)
])

model.layers, model.summary()

model.compile(loss="mean_squared_error",
              optimizer=tf.keras.optimizers.SGD(learning_rate=4e-3)
              )
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 normalization (Normalizatio  (None, 2)                5
 n)

 dense (Dense)               (None, 150)               450

 dense_1 (Dense)             (None, 150)               22650

 dense_2 (Dense)             (None, 150)               22650

 dense_3 (Dense)             (None, 150)               22650

 dense_4 (Dense)             (None, 150)               22650

 dense_5 (Dense)             (None, 150)               22650

 dense_6 (Dense)             (None, 150)               22650

 dense_7 (Dense)             (None, 150)               22650

 dense_8 (Dense)             (None, 150)               22650

 dense_9 (Dense)             (None, 150)               22650

 dense_10 (Dense)            (None, 1)                 151

=================================================================
Total params: 204,456
Trainable params: 204,451
Non-trainable params: 5
_________________________________________________________________
```

In [6]:

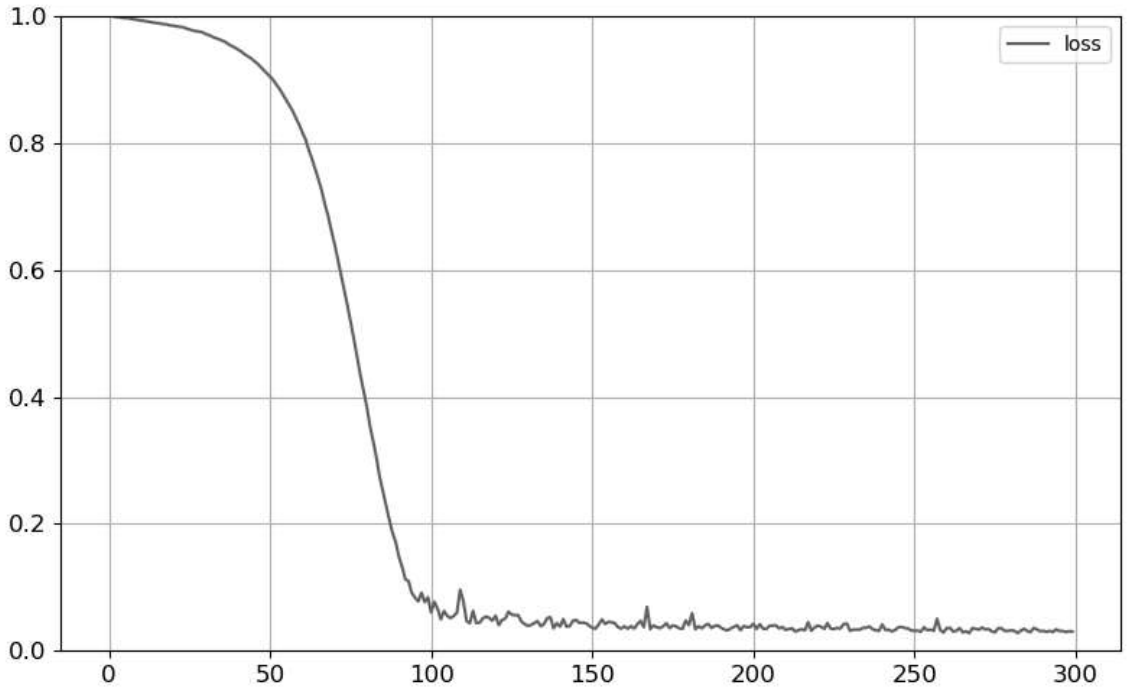
```
history = model.fit(X_train_scale, y_train_scale, epochs = 300)

history.history
```

```
Epoch 11/300
14/14 [==============================] - 0s 4ms/step - loss: 0.9931
Epoch 12/300
14/14 [==============================] - 0s 4ms/step - loss: 0.9923
Epoch 13/300
14/14 [==============================] - 0s 3ms/step - loss: 0.9915
Epoch 14/300
14/14 [==============================] - 0s 5ms/step - loss: 0.9906
Epoch 15/300
14/14 [==============================] - 0s 3ms/step - loss: 0.9897
Epoch 16/300
14/14 [==============================] - 0s 4ms/step - loss: 0.9891
Epoch 17/300
14/14 [==============================] - 0s 4ms/step - loss: 0.9881
Epoch 18/300
14/14 [==============================] - 0s 3ms/step - loss: 0.9875
Epoch 19/300
14/14 [==============================] - 0s 4ms/step - loss: 0.9869
Epoch 20/300
14/14 [==============================] - 0s 4ms/step - loss: 0.9853
```

In [7]:

```
pd.DataFrame(history.history).plot(figsize=(8, 5))
#plt.plot(pd.DataFrame(history.history))
plt.grid(True)
plt.gca().set_ylim(0, 1)
save_fig("keras_learning_curves_plot")
plt.show()

model.evaluate(X_train_scale, y_train_scale)
model.evaluate(X_test_scale, y_test_scale)
```

```
Saving figure keras_learning_curves_plot
```




```
14/14 [==============================] - 0s 3ms/step - loss: 0.0265
4/4 [==============================] - 0s 0s/step - loss: 0.0266
```

Out[7]:

```
0.026625096797943115
```

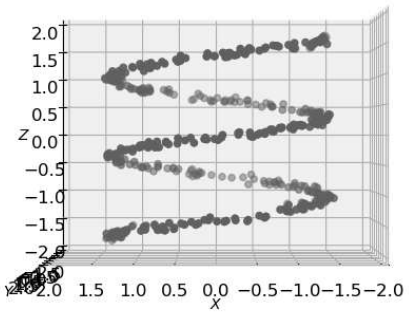
Plot Model Predictions for Training Set

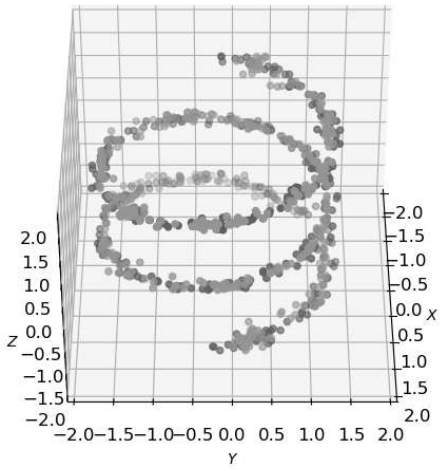
Plot predicted values based on the training data to show how closely the model learned.

In [8]:

```
#plotscatter3Ddata(fit_x, fit_y, fit_z, scat_x, scat_y, scat_z)
train_preds = model.predict(X_train_scale)

# I included the training set MSE and pure scatter plot version to get conf
# how close the predictions are.
train_val_error = mean_squared_error(y_train_scale, train_preds).round(4)
print ("MSE for training set", train_val_error)

figure_name = "3D_plot"
plotscatter3D(X_train_scale[:,0], train_preds[:,0], X_train_scale[:,1],
                  X_train_scale[:,0], y_train_scale, X_train_scale[:,1])


#train_preds
```

```
14/14 [==============================] - 0s 3ms/step
MSE for training set 0.0265
Saving figure 3D_plot
```

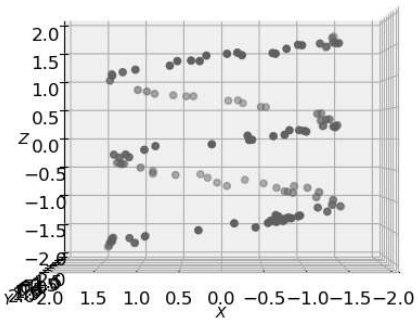


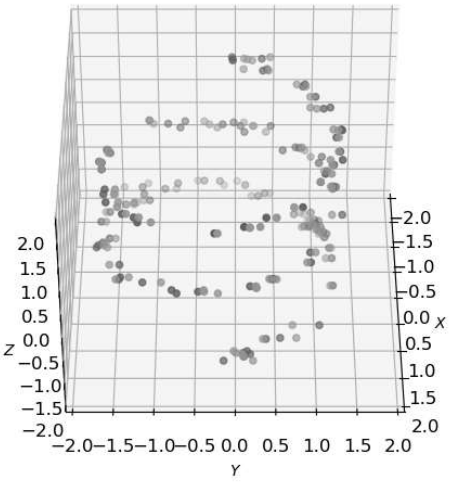


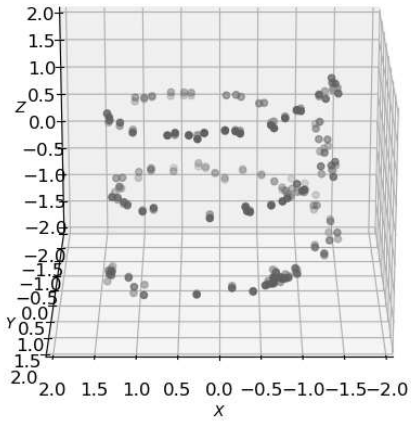
In [9]:

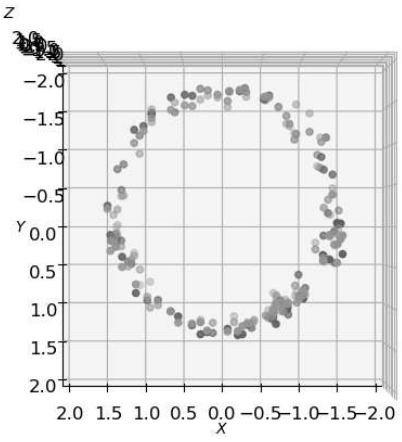
```
#plotscatter3Ddata(fit_x, fit_y, fit_z, scat_x, scat_y, scat_z)
test_preds = model.predict(X_test_scale)

# I included the training set MSE and pure scatter plot version to get conf
# how close the predictions are.
test_val_error = mean_squared_error(y_test_scale, test_preds).round(4)
print ("MSE for test set", test_val_error)

figure_name = "3D_plot test vs train"
plotscatter3D(X_test_scale[:,0], test_preds[:,0], X_test_scale[:,1],
                  X_test_scale[:,0], y_test_scale, X_test_scale[:,1])


#train_preds
```

```
4/4 [==============================] - 0s 7ms/step
MSE for test set 0.0266
Saving figure 3D_plot test vs train
```









Prepare Data using X and Y

In [10]:

```
#Reuse the same data dataframe
# The "data" dataframe contains 3 columns -- an x, y, and z coordinate; x
# and y will become part
# of X_ and z will become y_ variables

arr = np.stack((data.iloc[:,0], data.iloc[:,1]), axis = -1) # x and y

X_train, X_test, y_train, y_test = train_test_split(arr, data.iloc[:,-1],
                                                    test_size = .20, random
scalerX = StandardScaler()
scalerY = StandardScaler()
X_train_scale = scalerX.fit_transform(X_train)
X_test_scale = scalerX.transform (X_test)

y_train = np.array(y_train).reshape(-1,1)
y_test = np.array(y_test).reshape(-1,1)

y_train_scale = scalerY.fit_transform(y_train)
y_train_scale = np.array(y_train_scale).reshape(-1)
y_test_scale = scalerY.transform(y_test)
y_test_scale = np.array(y_test_scale).reshape(-1)

np.random.seed(42)
tf.random.set_seed(42)
```

Buld the Model with ANN again

In [11]:

```
tf.random.set_seed(42)

normalizer = tf.keras.layers.Normalization(axis = -1)
normalizer.adapt(np.array(X_train_scale))

model_bad = keras.models.Sequential([
   normalizer,
#    keras.layers.Input( shape = X_train_scale.shape[1:]),
#    keras.layers.Dense(2, activation='relu', input_shape = X_train_scale.s
    keras.layers.Dense(150, activation='relu'),
    keras.layers.Dense(150, activation='relu'),
    keras.layers.Dense(150, activation='relu'),
    keras.layers.Dense(150, activation='relu'),
    keras.layers.Dense(150, activation='relu'),
    keras.layers.Dense(150, activation='relu'),
    keras.layers.Dense(150, activation='relu'),
    keras.layers.Dense(150, activation='relu'),
    keras.layers.Dense(150, activation='relu'),
    keras.layers.Dense(150, activation='relu'),
    keras.layers.Dense(1)
])

model_bad.layers, model_bad.summary()

model_bad.compile(loss="mean_squared_error",
             optimizer=tf.keras.optimizers.SGD(learning_rate=4e-3)
             )
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 normalization_1 (Normalizat  (None, 2)                5
 ion)

 dense_11 (Dense)            (None, 150)               450

 dense_12 (Dense)            (None, 150)               22650

 dense_13 (Dense)            (None, 150)               22650

 dense_14 (Dense)            (None, 150)               22650

 dense_15 (Dense)            (None, 150)               22650

 dense_16 (Dense)            (None, 150)               22650

 dense_17 (Dense)            (None, 150)               22650

 dense_18 (Dense)            (None, 150)               22650

 dense_19 (Dense)            (None, 150)               22650

 dense_20 (Dense)            (None, 150)               22650

 dense_21 (Dense)            (None, 1)                 151

=================================================================
Total params: 204,456
Trainable params: 204,451
Non-trainable params: 5
_________________________________________________________________
```

In [12]:

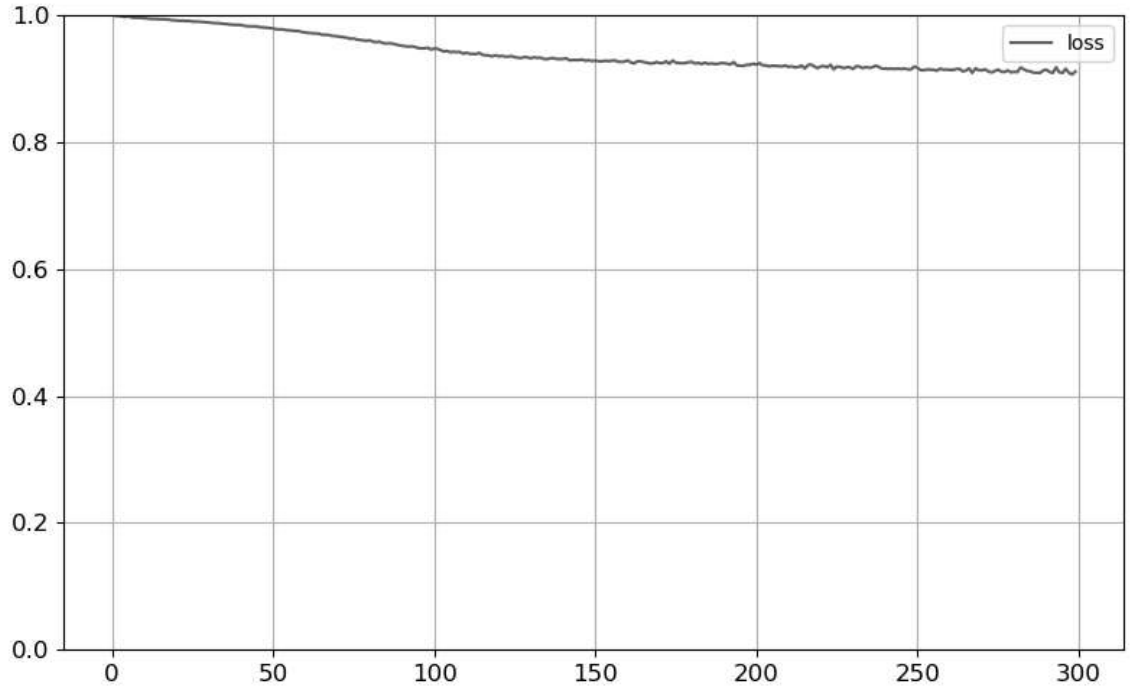
```
history = model_bad.fit(X_train_scale, y_train_scale, epochs = 300)

history.history
```

```
Epoch 1/300
14/14 [==============================] - 1s 5ms/step - loss: 1.0004
Epoch 2/300
14/14 [==============================] - 0s 3ms/step - loss: 0.9991
Epoch 3/300
14/14 [==============================] - 0s 4ms/step - loss: 0.9988
Epoch 4/300
14/14 [==============================] - 0s 4ms/step - loss: 0.9981
Epoch 5/300
14/14 [==============================] - 0s 4ms/step - loss: 0.9971
Epoch 6/300
14/14 [==============================] - 0s 4ms/step - loss: 0.9980
Epoch 7/300
14/14 [==============================] - 0s 3ms/step - loss: 0.9965
Epoch 8/300
14/14 [==============================] - 0s 4ms/step - loss: 0.9957
Epoch 9/300
14/14 [==============================] - 0s 4ms/step - loss: 0.9960
Epoch 10/300
```

In [13]:

```
pd.DataFrame(history.history).plot(figsize=(8, 5))
#plt.plot(pd.DataFrame(history.history))
plt.grid(True)
plt.gca().set_ylim(0, 1)
save_fig("keras_learning_curves_plot_whoops")
plt.show()

model_bad.evaluate(X_train_scale, y_train_scale)
model_bad.evaluate(X_test_scale, y_test_scale)
```

```
Saving figure keras_learning_curves_plot_whoops
```




```
14/14 [==============================] - 0s 1ms/step - loss: 0.9020
4/4 [==============================] - 0s 4ms/step - loss: 1.1314
```

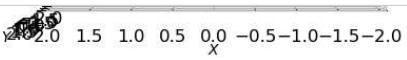
Out[13]:

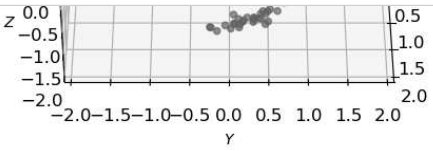
```
1.1314425468444824
```

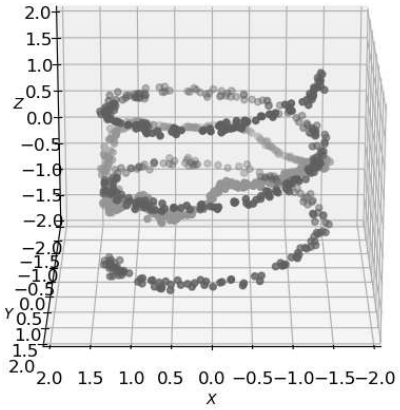
Plot Model Predictions for Training Set

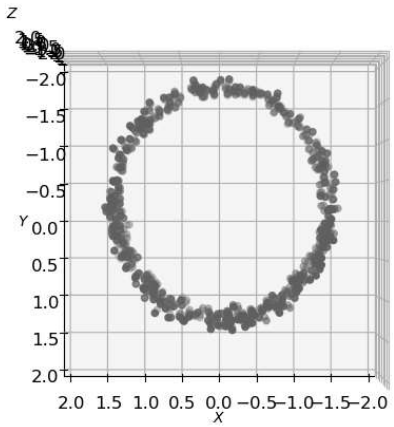
In [14]:

```
#plotscatter3Ddata(fit_x, fit_y, fit_z, scat_x, scat_y, scat_z)
train_preds = model_bad.predict(X_train_scale)

# I included the training set MSE and pure scatter plot version to get conf
# how close the predictions are.
train_val_error = mean_squared_error(y_train_scale, train_preds).round(4)
print ("MSE for training set", train_val_error)

figure_name = "3D_plot_whoops"
plotscatter3D(X_train_scale[:,0], X_train_scale[:,1], train_preds[:,0],
                  X_train_scale[:,0], X_train_scale[:,1], y_train_scale)


#train_preds
```





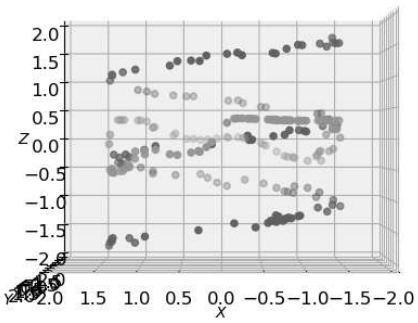


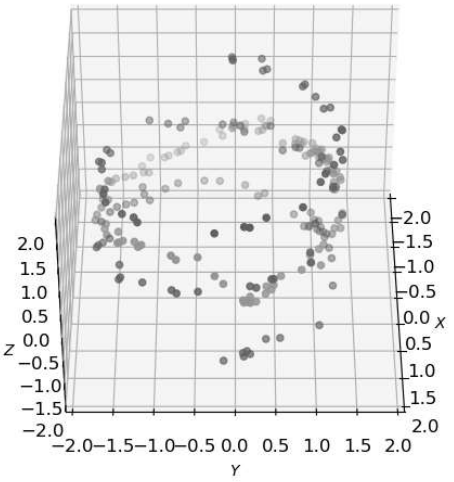


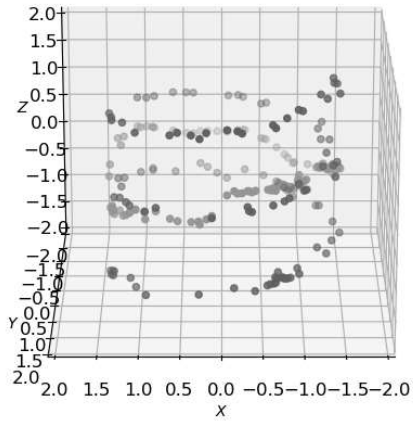
In [15]:

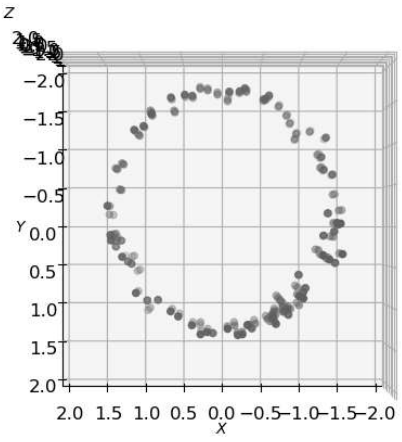
```
#plotscatter3Ddata(fit_x, fit_y, fit_z, scat_x, scat_y, scat_z)
test_preds = model_bad.predict(X_test_scale)

# I included the training set MSE and pure scatter plot version to get conf
# how close the predictions are.
train_val_error = mean_squared_error(y_test_scale, test_preds).round(4)
print ("MSE for test set", test_val_error)

figure_name = "3D_plot_whoops test"
plotscatter3D(X_test_scale[:,0], X_test_scale[:,1], test_preds[:,0],
                  X_test_scale[:,0], X_test_scale[:,1], y_test_scale)


#train_preds
```

```
4/4 [==============================] - 0s 4ms/step
MSE for test set 0.0266
Saving figure 3D_plot_whoops test
```









Conclusion

Neural Nets can be used to quickly develop models that have decent predictive capabilities. However, all the same issues with data quality and data element selection remain -- without

In [ ]: