ks_net

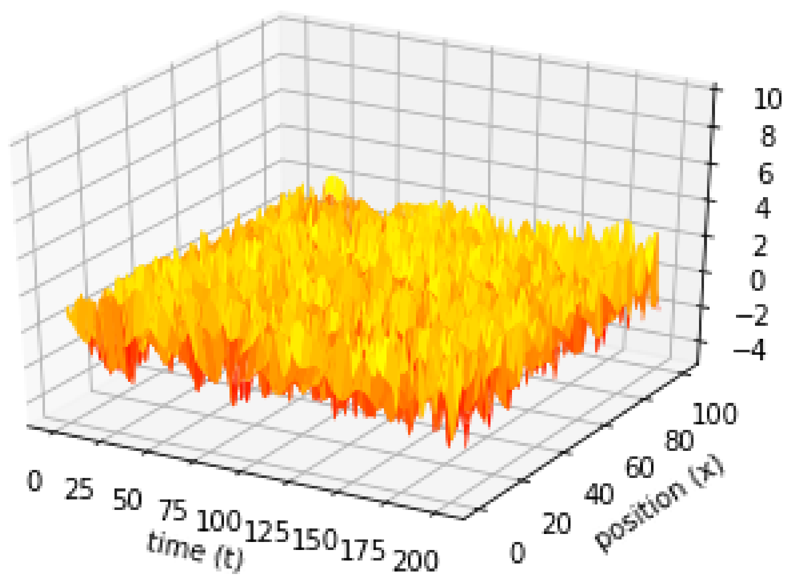
May 20, 2019

0.0.1 Generating Data

Below we choose 100 different starting points for the Lorenz system and calculate the trajectory for 8 seconds in time. We will use this to train our neural network. This will allow the network to act as a time-stepper : given a point (x,y,z) it can calculate the new point (x', y', z') dt seconds further in time.

[1]:
```
import matplotlib.pylab as plt
import numpy as np
import scipy.integrate
import scipy.io
from mpl_toolkits.mplot3d import Axes3D
import os

%matplotlib inline

max_train_size = 100 # max number of initial conditions to use for data
curr_train_size = 0


# Load data for this system
for f in os.scandir('./data/'):

    if curr_train_size >= max_train_size: break

    if 'kuramoto' in f.name and 'train' in f.name:
        data = scipy.io.loadmat(f.path)
        uu = data['uu']

        # Construct inputs and expected outputs (i.e. prediction dt into the␣
 ↪future)
        if curr_train_size == 0:
            inputs = uu.T[:-1, :]
            outputs = uu.T[1:, :]
        else:
            inputs = np.vstack((inputs, uu.T[:-1, :]))
            outputs = np.vstack((outputs, uu.T[1:, :]))

        curr_train_size += 1
```

```
# just a small glimpse of what the system looks like at one particular initial␣
 ↪condition
uu = data['uu']
x = data['x']
tt = data['tt']

fig = plt.figure(1)
ax = fig.gca(projection='3d')
ax.plot_surface(tt, x, uu, cmap='autumn')

plt.xlabel('time (t)')
plt.ylabel('position (x)')
ax.set_zlim(-5, 10)

plt.show(block=True)
```



0.0.2 Training A neural Network

Here we define a standard 3-layer feedforward neural network with 20 neurons in each layer

[23]:
```
from keras.models import Sequential
from keras.layers import Dense
```

```
from keras import regularizers

l1 = 1e-5  # L1 regularization of network weights\
l2 = 0 # L2 Regularization of network weights

# can define a custom activation function and pass it as a parameter with␣
 ↪'activation' as well.
model = Sequential()
model.add(Dense(500, activation='relu', kernel_regularizer=regularizers.
 ↪l1_l2(l1=l1, l2=l2), input_shape = (uu.shape[0],)))
model.add(Dense(500, kernel_regularizer=regularizers.l1_l2(l1=l1, l2=l2),␣
 ↪activation='relu'))
model.add(Dense(500, kernel_regularizer=regularizers.l1_l2(l1=l1, l2=l2),␣
 ↪activation='sigmoid'))
model.add(Dense(500, kernel_regularizer=regularizers.l1_l2(l1=l1, l2=l2),␣
 ↪activation='relu'))
model.add(Dense(500, kernel_regularizer=regularizers.l1_l2(l1=l1, l2=l2),␣
 ↪activation='relu'))
model.add(Dense(500, kernel_regularizer=regularizers.l1_l2(l1=l1, l2=l2),␣
 ↪activation='relu'))

model.add(Dense(uu.shape[0], activation='linear'))

model.compile(optimizer='adadelta', loss='mean_squared_error')
```

[ ]:

```
model.fit(inputs, outputs,
          epochs=100,
          batch_size=100,
          shuffle=True,
          validation_split = 0.2) # use 20 % of data as a validation dataset
```

0.0.3 Analyzing Training

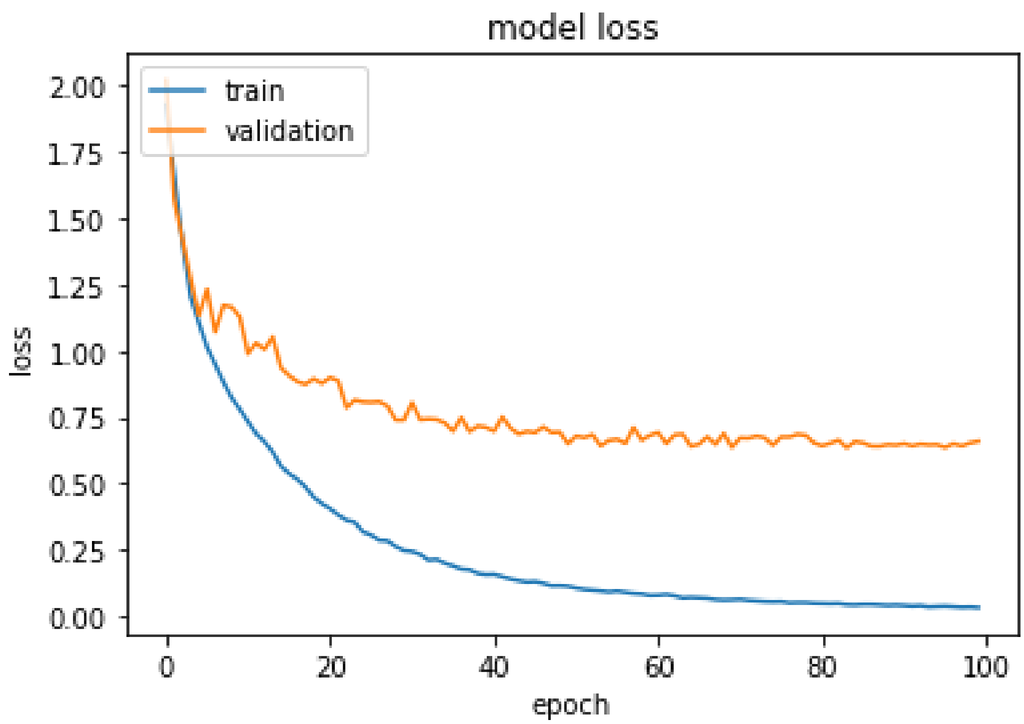
Analyze the loss of the neural net (on both the training and validation datasets) as a function of epochs

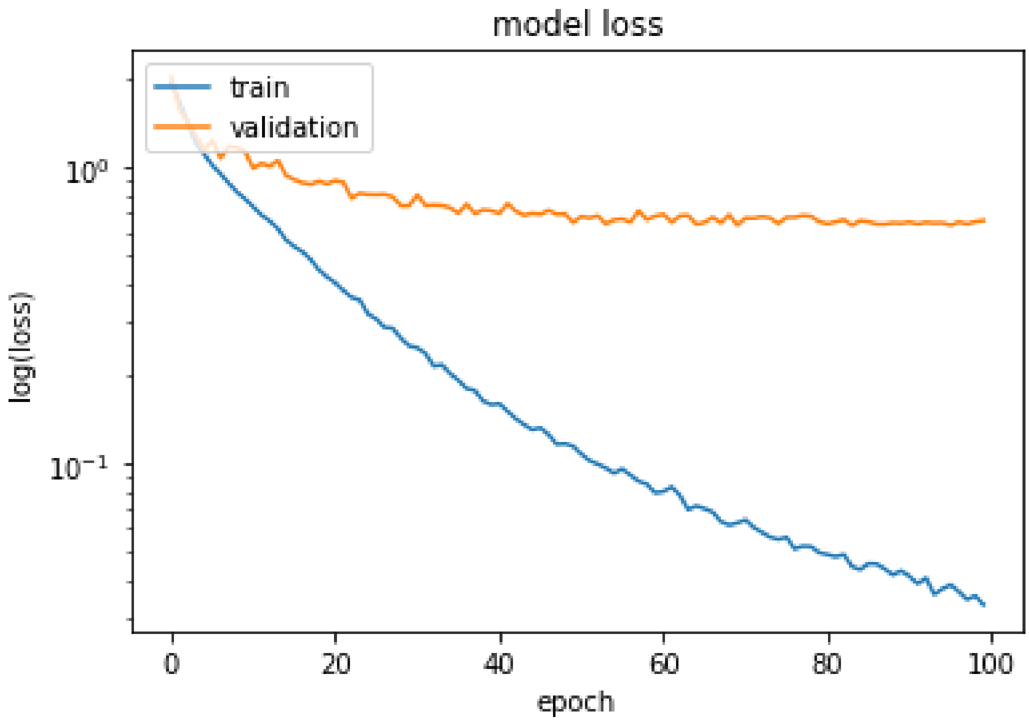
[25]:

```
%matplotlib inline

# summarize history for loss
plt.figure(2)
plt.plot(model.history.history['loss'])
plt.plot(model.history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
```

```
plt.figure(3)
plt.semilogy(model.history.history['loss'])
plt.semilogy(model.history.history['val_loss'])
plt.title('model loss')
plt.ylabel('log(loss)')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
```

[25]: `<matplotlib.legend.Legend at 0x7f5264614828>`



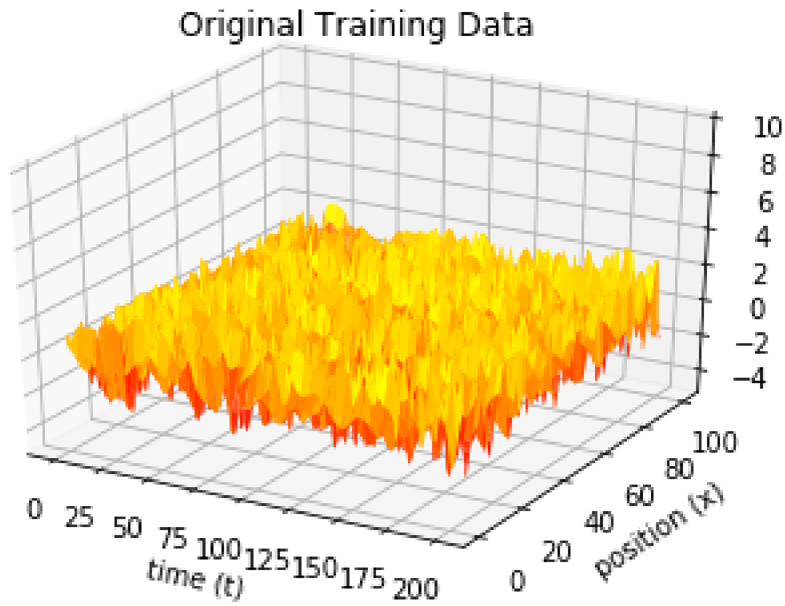


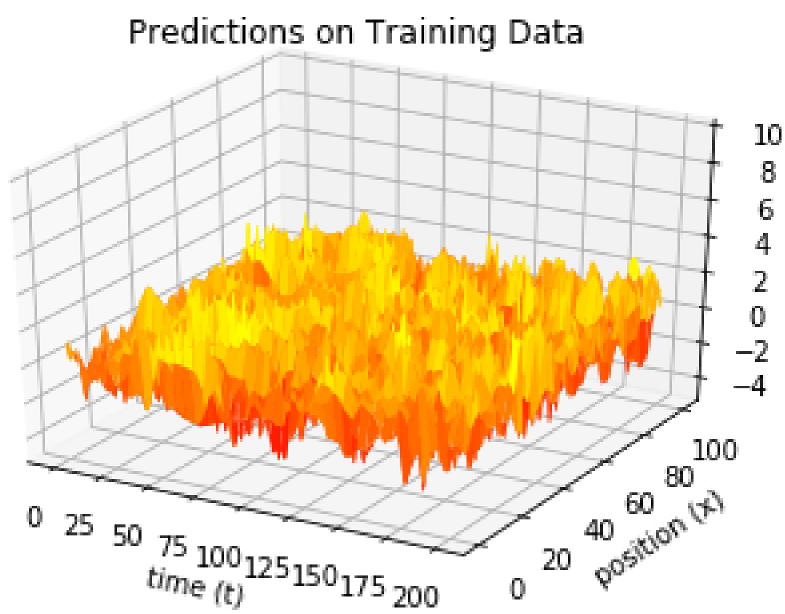
0.0.4 Test Performance of Neural Net

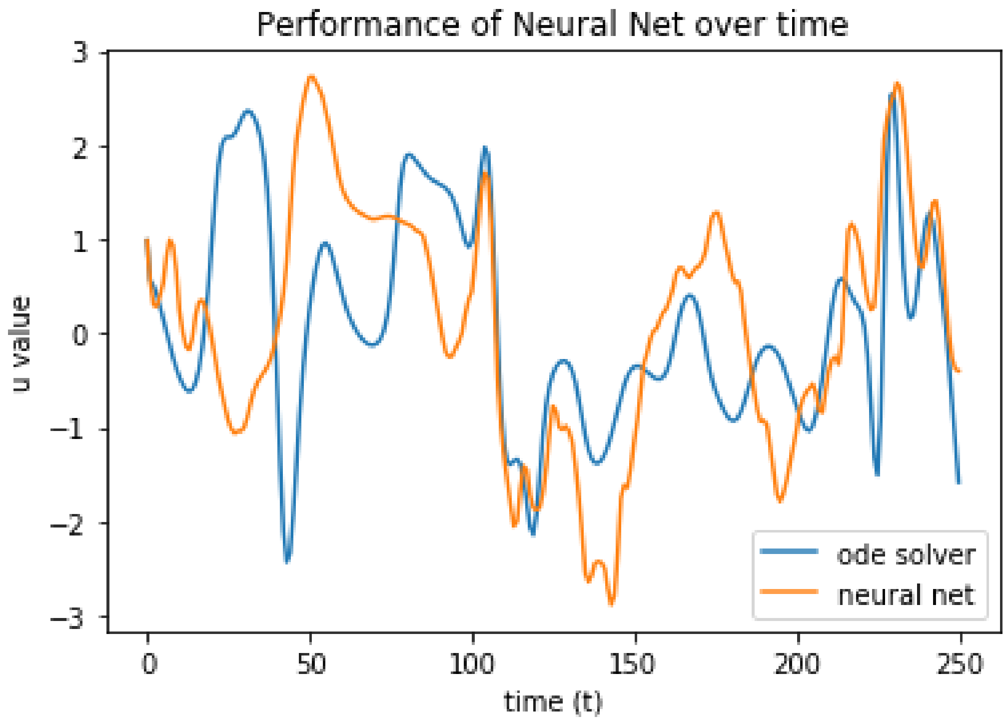
Here, we will generate new trajectories using a random starting condition. Using both an ode solver and our neural network, we will predict the trajectories of the points.

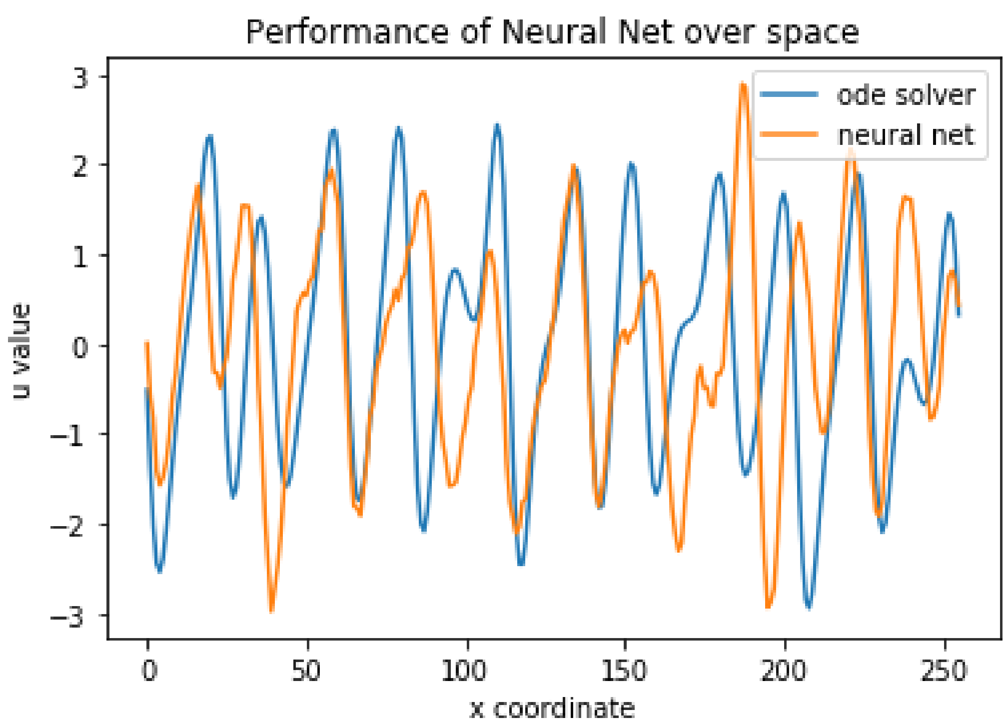
[44]:
```
%matplotlib inline

# performance on training data

y_NN = np.zeros(uu.shape)
y_NN[:, 0] = uu[:, 0]

for i in range(1, y_NN.shape[1]):
    y_NN[:, i] = model.predict(np.expand_dims(y_NN[:, i-1], axis=1).T)

# plot original training data
fig = plt.figure(4)
ax = fig.gca(projection='3d')
ax.plot_surface(tt, x, uu, cmap='autumn')
plt.title('Original Training Data')
plt.xlabel('time (t)')
plt.ylabel('position (x)')
ax.set_zlim(-5, 10)
```

```python
plt.show(block=True)

# plot predictions
fig = plt.figure(5)
ax = fig.gca(projection='3d')
ax.plot_surface(tt, x, y_NN, cmap='autumn')
plt.title('Predictions on Training Data')
plt.xlabel('time (t)')
plt.ylabel('position (x)')
ax.set_zlim(-5, 10)

plt.show(block=True)

# plot some overlaid trajectories for fixed x
x_coord = 0
plt.figure()
plt.plot(uu[x_coord, :])
plt.plot(y_NN[x_coord, :])
plt.title("Performance of Neural Net over time")
plt.xlabel('time (t)')
plt.ylabel('u value')
plt.legend(['ode solver', 'neural net'])
plt.show()

# plot some overlaid trajectories for fixed t
time = 40
plt.figure()
plt.plot(uu[:, time])
plt.plot(y_NN[:, time])
plt.title("Performance of Neural Net over space")
plt.xlabel('x coordinate')
plt.ylabel('u value')
plt.legend(['ode solver', 'neural net'])
plt.show()
```


Original Training Data
10
8
6
4
2
0
−2
−4
0
25
50
75
100
125
150
175
200
time (t)
0
20
40
60
80
100
position (x)


Predictions on Training Data
10
8
6
4
2
0
−2
−4
0
25
50
75
100
125
150
175
200
time (t)
0
20
40
60
80
100
position (x)

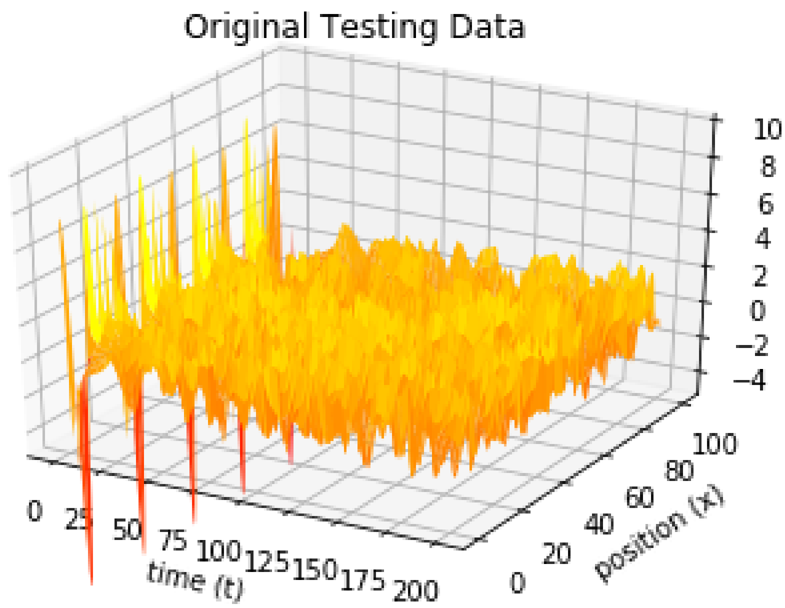

Performance of Neural Net over time
u value
time (t)
ode solver
neural net
3
2
1
0
−1
−2
−3
0
50
100
150
200
250

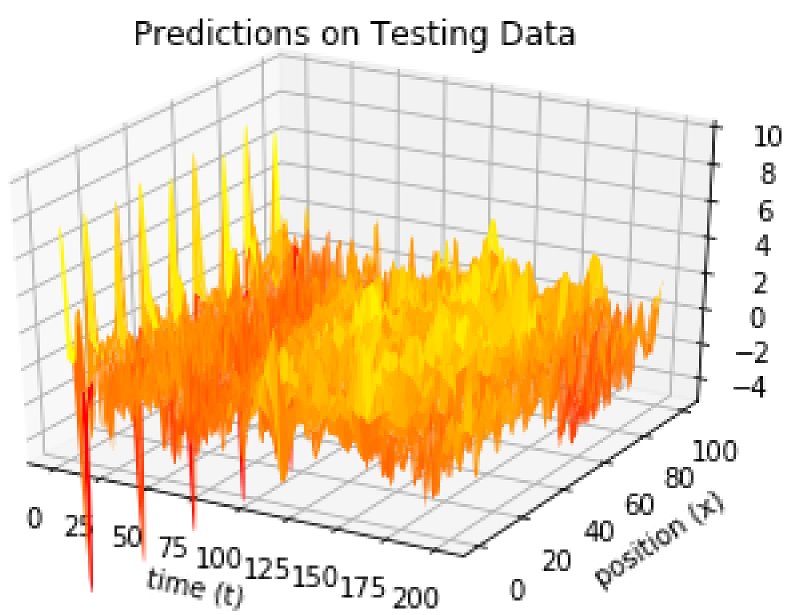

Performance of Neural Net over space
u value
x coordinate
ode solver
neural net
3
2
1
0
−1
−2
−3
0
50
100
150
200
250

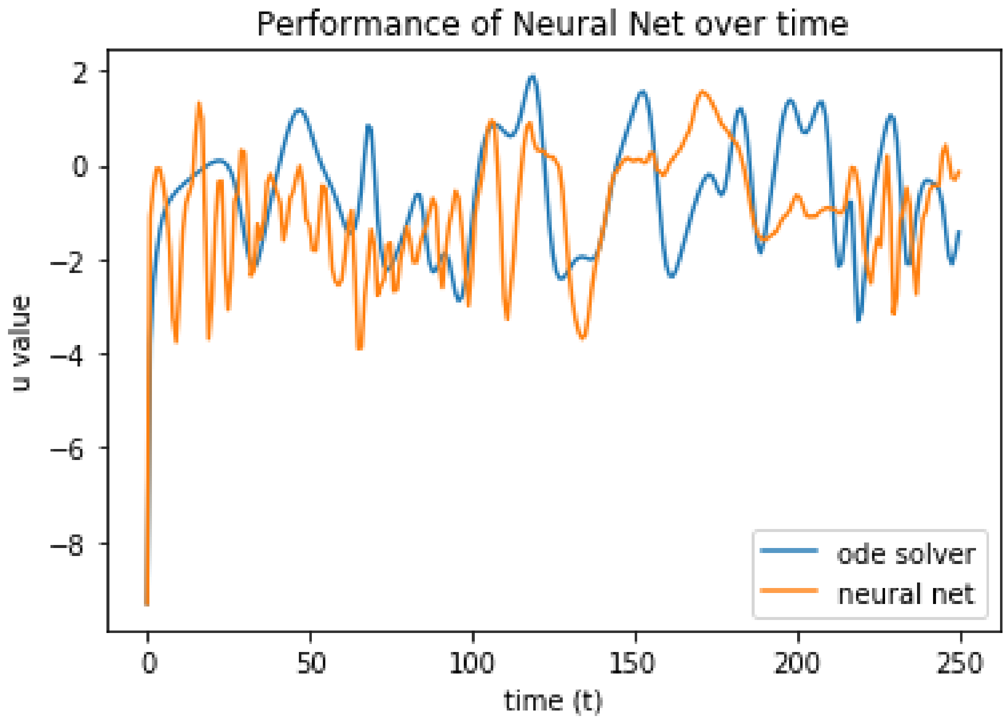
[54]:

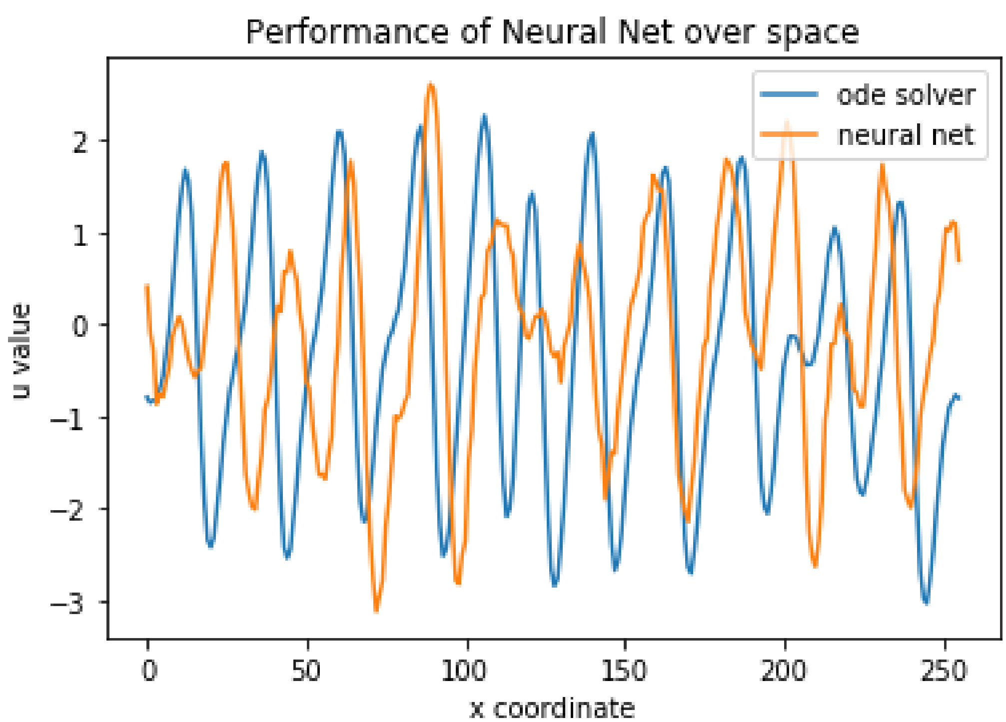
```
# LOAD TESTING DATA

test_data = scipy.io.loadmat('./data/kuramoto_sivishinky_test2.mat')
uu_test = test_data['uu']
# performance on testing data


y_NN = np.zeros(uu_test.shape)
y_NN[:, 0] = uu_test[:, 0]

for i in range(1, y_NN.shape[1]):
    y_NN[:, i] = model.predict(np.expand_dims(y_NN[:, i-1], axis=1).T)


fig = plt.figure(4)
ax = fig.gca(projection='3d')
ax.plot_surface(tt, x, uu_test, cmap='autumn')
plt.title('Original Testing Data')
plt.xlabel('time (t)')
plt.ylabel('position (x)')
ax.set_zlim(-5, 10)

plt.show(block=True)

fig = plt.figure(5)
ax = fig.gca(projection='3d')
ax.plot_surface(tt, x, y_NN, cmap='autumn')
plt.title('Predictions on Testing Data')
plt.xlabel('time (t)')
plt.ylabel('position (x)')
ax.set_zlim(-5, 10)

plt.show(block=True)

# plot some overlaid trajectories for fixed x
x_coord = 10
plt.figure()
plt.plot(uu_test[x_coord, :])
plt.plot(y_NN[x_coord, :])
plt.title("Performance of Neural Net over time")
plt.xlabel('time (t)')
plt.ylabel('u value')
plt.legend(['ode solver', 'neural net'])
plt.show()

# plot some overlaid trajectories for fixed t
time = 150
```

```
plt.figure()
plt.plot(uu_test[:, time])
plt.plot(y_NN[:, time])
plt.title("Performance of Neural Net over space")
plt.xlabel('x coordinate')
plt.ylabel('u value')
plt.legend(['ode solver', 'neural net'])
plt.show()
```






Performance of Neural Net over time
u value
time (t)
ode solver
neural net
2
0
−2
−4
−6
−8
0
50
100
150
200
250


Performance of Neural Net over space
u value
x coordinate
ode solver
neural net
2
1
0
−1
−2
−3
0
50
100
150
200
250

[ ]:

```

```

[ ]:

```

```