Intro + Context

```
Approaches to Anomaly Detection

Pre-Processing

Approaches
    1 -- AE

    2 -- clustering [ outliers in the embedding space ]
        non DL methods [ e.g. isolation forest ]

    3 -- context model /?/ internal model [ predict future output t_1 + t
_N ]
        LSTM


Ensembling + [ GTC Lab ]
```

Target Audience + Background

In [1]:
```
import sys; sys.path.append('../utils')
```

In [2]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

from tight_plot import tight_plot
from customcolors import *
import nnViz
```

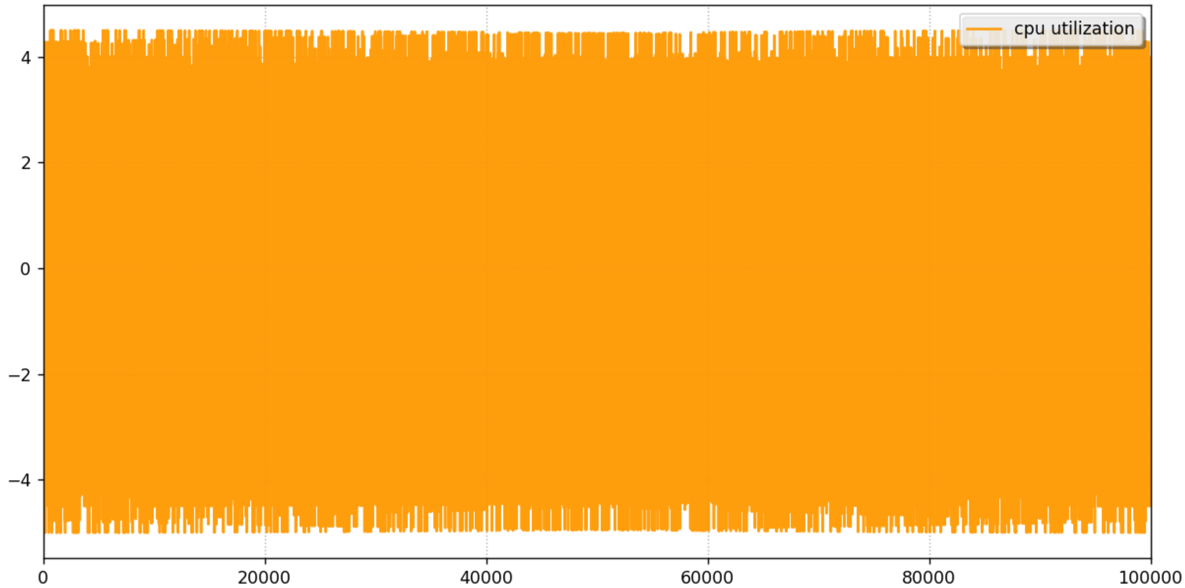
In [3]:
```
%matplotlib notebook
```

Generate Synthetic Data

In [4]:
```
numSyntheticSamples = 100000
syntheticDataRange = np.linspace( 0, np.pi*numSyntheticSamples/10.0, numSynthetic
syntheticData = np.sin( syntheticDataRange ) \
                    * np.random.randint( 1, 10, size = numSyntheticSamples ) * 1/
                    + np.random.randint( -1, 1, size = numSyntheticSamples ) * 1/
```

In [5]:
```
if syntheticData.ndim < 2:
    syntheticData = np.expand_dims(syntheticData, axis=1)
```

In [6]:
```
tight_plot(syntheticData, rawDataColor,'cpu utilization')
```



Define Model Hyper-parameters

In [7]:
```
# Define hyperparameters

# of samples per sensor for the micro model [sliding window of ~2.5 hrs]
hParams = {}

hParams['windowSamples'] = 30
hParams['bottleNeckUnits'] = 5
hParams['nSensors'] = 1
hParams['overlapPercentage'] = .99
hParams['advanceSamples'] = ( hParams['windowSamples'] - int( np.floor( hParams['
```

Split into train and test set (.25 test data)

In [8]:
```
def train_test_split (x, testDataRatio = .25, trainDataAtStart = True):
    assert x.ndim > 1
    if trainDataAtStart:
        splitIndex = int( ( 1.0 - testDataRatio) * x.shape[0] )

        xTrain = x[ 0:splitIndex, :]
        xTest = x[ splitIndex:, :]
    else:
        splitIndex = int( testDataRatio * x.shape[0] )
        xTest = x[ 0:splitIndex, :]
        xTrain = x[ splitIndex:, :]

    return xTrain, xTest
```

In [9]:
```
trainSplit, testSplit = train_test_split( syntheticData )
```

Normalize data (0 mean, unit standard deviation)

In [10]:
```
# find normalization statistics
trainMeans = np.mean(trainSplit, axis=0)
trainSTDevs = np.std(trainSplit, axis=0)
print(trainMeans); print(trainSTDevs)

# normalize [ in place / overwrite ]
normalizedTrainData = (trainSplit - trainMeans) / (trainSTDevs + .0001)
normalizedTestData = (testSplit - trainMeans) / (trainSTDevs + .0001)
```

```
[-0.251098]
[2.00798495]
```

Generate Overlapping Windows

In [11]:
```
def reshape_into_shuffled_data_windows ( x, windowSize, advanceSamples ):
    nWindows = int( np.floor( (x.shape[0] - windowSize)/(advanceSamples*1.0) ) )
    # shuffle indexes
    shuffledWindowInds = np.arange(nWindows)
    np.random.shuffle(shuffledWindowInds)

    nSensors = x.shape[1]
    outputMatrix = np.zeros((nWindows, windowSize * nSensors))

    # update data matrix on a row by row basis (choosing shuffled windows per row
    for iWindow in range(nWindows):
        startIndex = shuffledWindowInds[iWindow] * advanceSamples
        endIndex = startIndex + windowSize

        # flatten/interleave sensor values
        for iSensor in range(nSensors):
            outputMatrix[iWindow, iSensor::nSensors] = x[startIndex:endIndex, iSe

    return outputMatrix, shuffledWindowInds
```

In [12]:
```
trainMatrix, trainShuffledWindowInds = reshape_into_shuffled_data_windows(normali
testMatrix, testShuffledWindowInds = reshape_into_shuffled_data_windows(normalize

plt.figure()
plt.plot(trainMatrix[204,:], color = rawDataColor)
plt.title('example training input')
```

Out[12]:
```
<matplotlib.text.Text at 0x1904f84aa90>
```

ML/DL Imports

In [13]:
```
from keras.models import Sequential, Model
from keras.layers import Dense
from keras import metrics
from keras.callbacks import EarlyStopping
from keras.callbacks import ModelCheckpoint
```

```
C:\ProgramData\Anaconda3\envs\tfGPU\lib\site-packages\h5py\__init__.py:34: Futu
reWarning: Conversion of the second argument of issubdtype from `float` to `np.
floating` is deprecated. In future, it will be treated as `np.float64 == np.dty
pe(float).type`.
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
```

Model Architecture

In [14]:

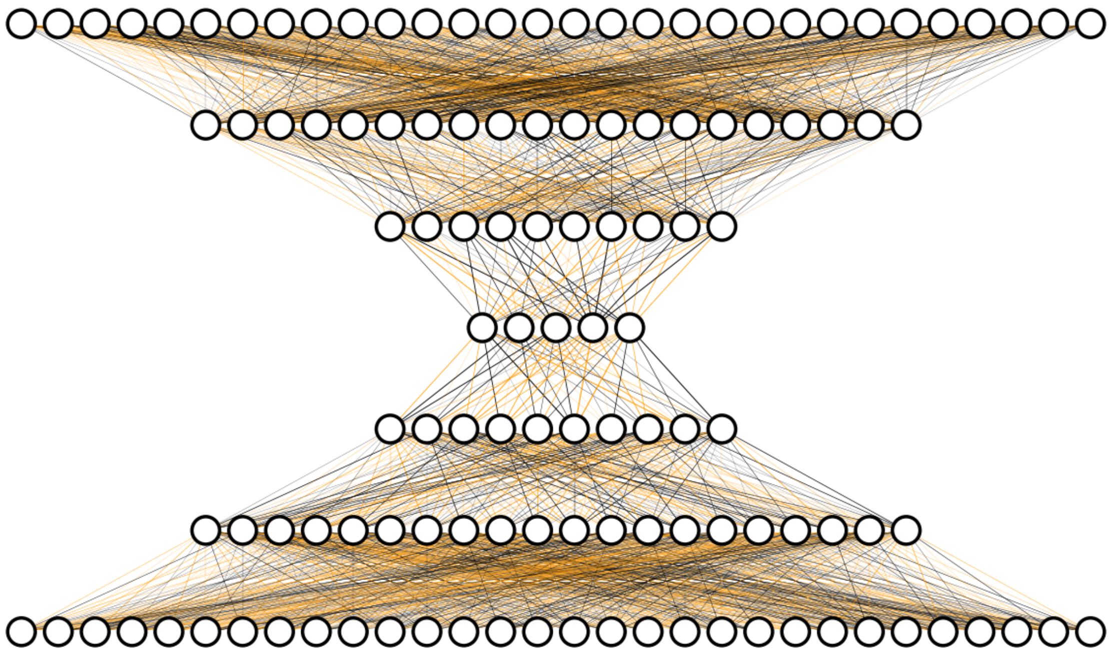
```
hParams['inputOutputDimensionality'] = int( hParams['windowSamples'] * hParams['n
assert hParams['inputOutputDimensionality'] == trainMatrix.shape[1]
```

In [15]:

```
# Define model
model = Sequential()


model.add( Dense( 20, input_dim = hParams['inputOutputDimensionality'], activatio

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

model.add( Dense( hParams['bottleNeckUnits'], activation = 'sigmoid'))

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

model.add( Dense( 20, activation = 'sigmoid'))

model.add( Dense( hParams['inputOutputDimensionality'], activation = 'linear',))


model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_1 (Dense)              (None, 20)                620       
_________________________________________________________________
dense_2 (Dense)              (None, 10)                210       
_________________________________________________________________
dense_3 (Dense)              (None, 5)                 55        
_________________________________________________________________
dense_4 (Dense)              (None, 10)                60        
_________________________________________________________________
dense_5 (Dense)              (None, 20)                220       
_________________________________________________________________
dense_6 (Dense)              (None, 30)                630       
=================================================================
Total params: 1,795
Trainable params: 1,795
Non-trainable params: 0
_________________________________________________________________
```

In [16]:
```
plt.figure(figsize=(10,10))
plt.subplots_adjust( left = 0.01, right = 0.99, top = 0.99, bottom = 0.01, wspace
nnViz.visualize_model(model)
```



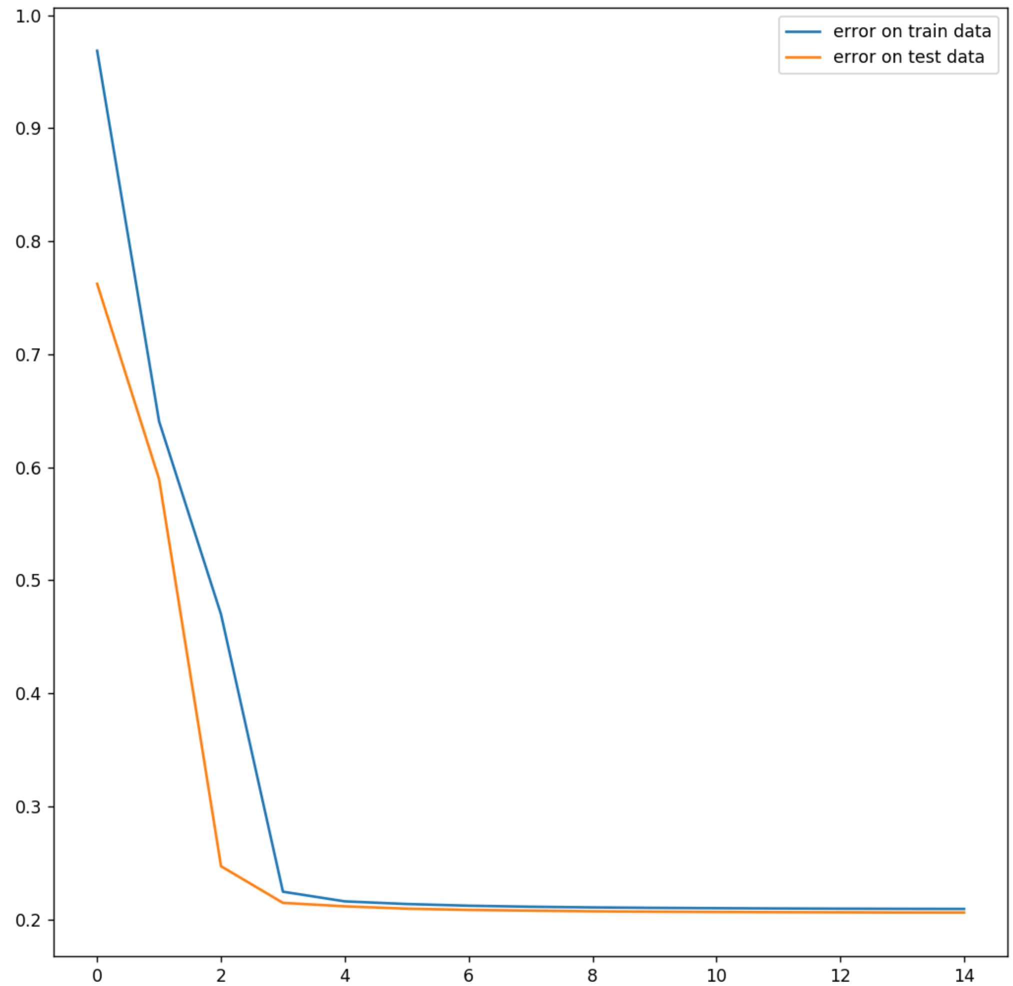
In [17]:
```
model.compile(optimizer = 'adam', loss = 'mse')
```

Train Model

In [18]:
```
early_stopping = EarlyStopping( monitor = 'val_loss', patience = 10)
checkpointer = ModelCheckpoint( filepath = 'synthetic_sin_weights_2.hdf5', verbos

'''
# keras version 2+
history = model.fit( trainMatrix, trainMatrix,
               batch_size = 256, epochs = 200,
               shuffle = True,
               callbacks = [early_stopping, checkpointer],
               validation_data = (testMatrix, testMatrix) )
'''

# keras version 1.2
history = model.fit( trainMatrix, trainMatrix,
               batch_size = 256, nb_epoch = 15,
               shuffle = True,
               callbacks = [early_stopping, checkpointer],
               validation_data = (testMatrix, testMatrix) )
```

```
val_loss: 0.7623
Epoch 2/15
73984/74970 [============================>.] - ETA: 0s - loss: 0.6412
Epoch 00002: val_loss improved from 0.76230 to 0.58942, saving model to synt
hetic_sin_weights_2.hdf5
74970/74970 [==============================] - 3s 35us/step - loss: 0.6406 -
val_loss: 0.5894
Epoch 3/15
73216/74970 [============================>.] - ETA: 0s - loss: 0.4750
Epoch 00003: val_loss improved from 0.58942 to 0.24690, saving model to synt
hetic_sin_weights_2.hdf5
74970/74970 [==============================] - 3s 36us/step - loss: 0.4699 -
val_loss: 0.2469
Epoch 4/15
74240/74970 [============================>.] - ETA: 0s - loss: 0.2245
Epoch 00004: val_loss improved from 0.24690 to 0.21448, saving model to synt
hetic_sin_weights_2.hdf5
74970/74970 [==============================] - 3s 34us/step - loss: 0.2244 -
val_loss: 0.2145
Epoch 5/15
```

In [19]:
```
plt.figure( figsize = (10,10) )
plt.plot( history.history['loss'] )
plt.plot( history.history['val_loss'] )
plt.legend(['error on train data', 'error on test data'])
```



Out[19]: <matplotlib.legend.Legend at 0x190ab7cc780>

Load Best Weights [on validation data]

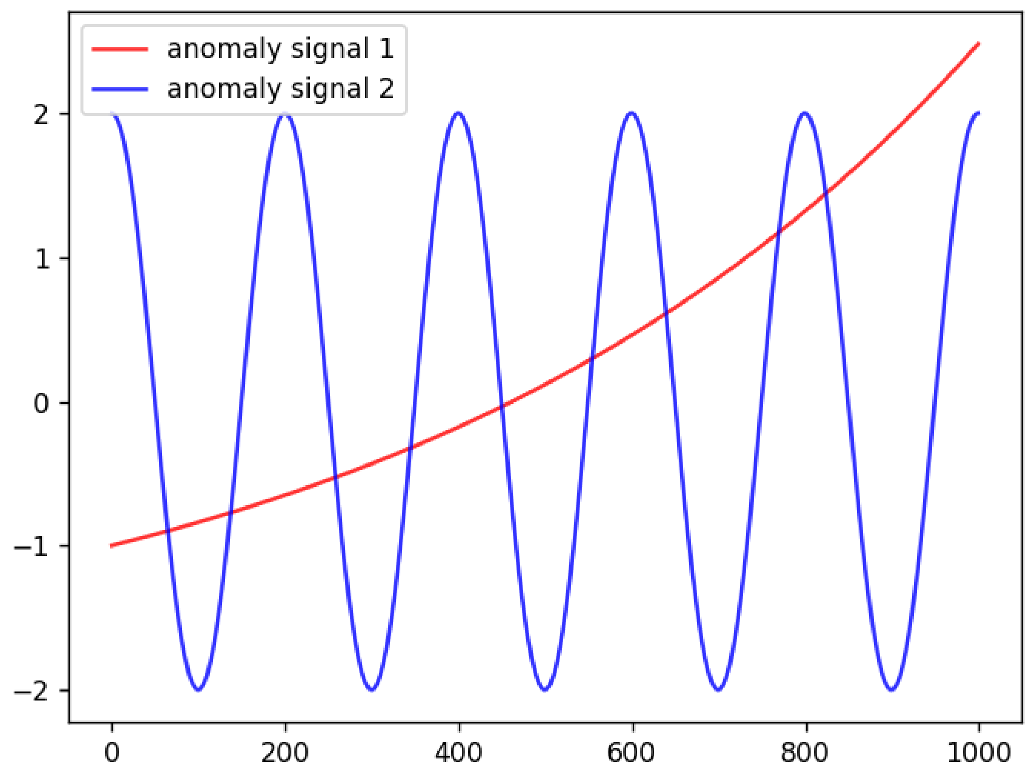
In [20]:
```
model.load_weights("synthetic_sin_weights_2.hdf5")
model.compile(optimizer = 'adam', loss = 'mse') # need to recompile model to be a
```

In [ ]:

Create Anomalies, and Mix/Insert into Training

Data

In [21]:
```
anomalySignal1 = np.exp(np.linspace(0, 1.5, 1000)) - 2
anomalySignal2 = np.cos(np.linspace(0,2*np.pi * 5, 1000)) * 2

anomalySignal1 = np.expand_dims(anomalySignal1, axis=1)
anomalySignal2 = np.expand_dims(anomalySignal2, axis=1)

plt.figure()

plt.plot(anomalySignal1, color = anomalyColor1)
plt.plot(anomalySignal2, color = anomalyColor2)

plt.legend(['anomaly signal 1', 'anomaly signal 2'])
```



Out[21]:
```
<matplotlib.legend.Legend at 0x190b01535f8>
```

In [22]:
```
startIndex = 0
endIndex = 3000
anomalyLen1 = len(anomalySignal1)
anomalyLen2 = len(anomalySignal1)
```

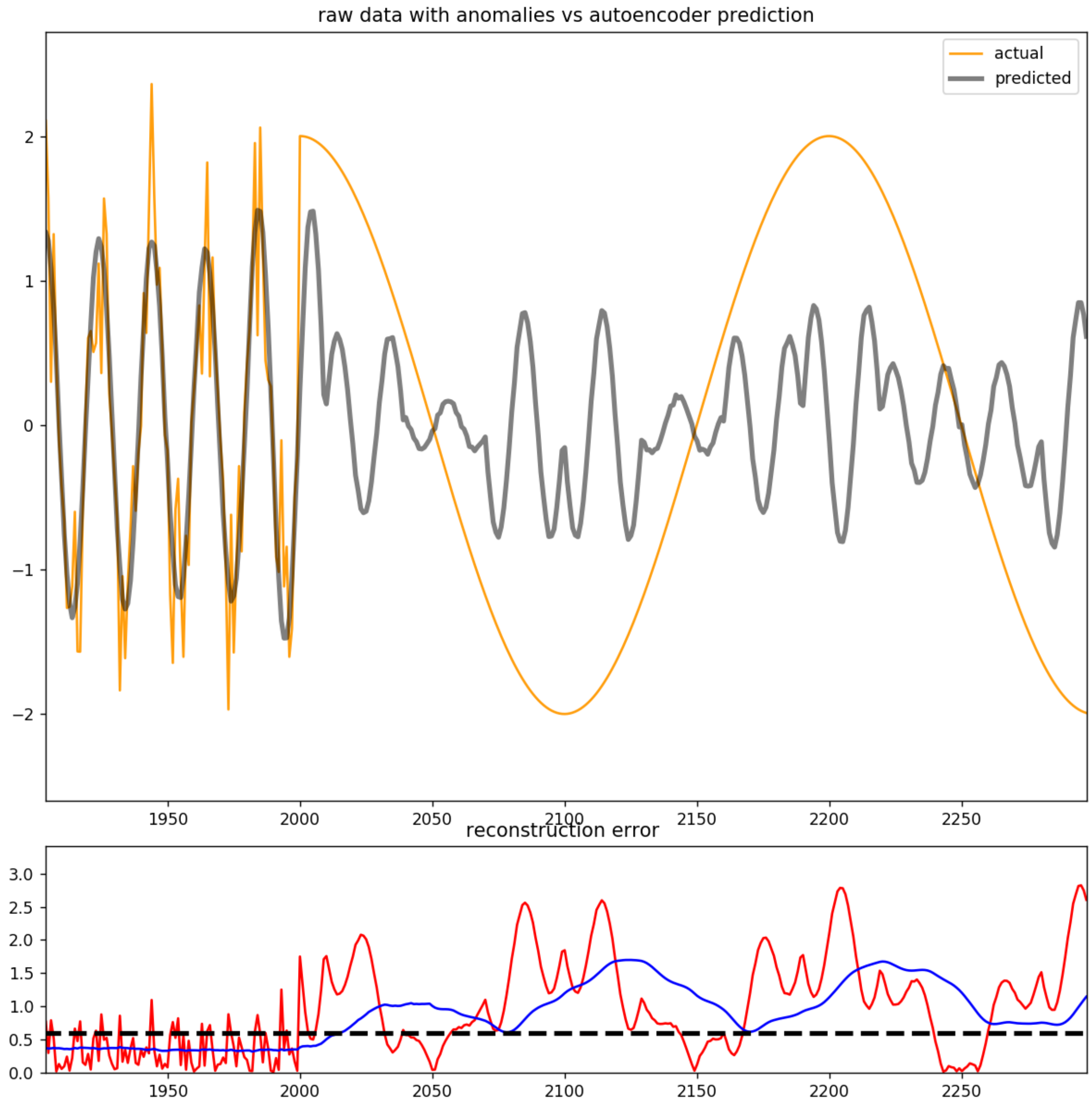
In [23]:
```
targetData = normalizedTestData[startIndex:endIndex]

anomalyStartIndex_1 = 500
anomalyEndIndex_1 = anomalyStartIndex_1 + anomalyLen1

targetData[anomalyStartIndex_1:anomalyEndIndex_1] = anomalySignal1

anomalyStartIndex_2 = 2000
anomalyEndIndex_2 = anomalyStartIndex_2 + anomalyLen2

targetData[anomalyStartIndex_2:anomalyEndIndex_2] = anomalySignal2


anomalousInds_1 = np.arange(int(anomalyStartIndex_1/hParams['windowSamples']), in
anomalousInds_2 = np.arange(int(anomalyStartIndex_2/hParams['windowSamples']), in
```

Plot Target/Raw vs Predicted Data

In [24]:
```
from sliding_window_inference import windowed_predict, windowed_predict_bottlenec
predictedData = windowed_predict ( model, targetData, hParams['inputOutputDimensi
error = np.sqrt((targetData - predictedData)**2)
```

```
number of windows: 100
```

In [25]:
```
plt.figure( figsize = (10,10) )

ax1 = plt.subplot2grid((4, 1), (0, 0), rowspan=3)
ax2 = plt.subplot2grid((4, 1), (3, 0), rowspan=1, sharex=ax1)
plt.subplots_adjust( left = 0.05, right = 0.95, top = 0.95, bottom = 0.05, wspace

ax1.plot(targetData, color = rawDataColor)
ax1.plot(predictedData, color = contrastColor2, linewidth=3)
ax1.set_title('raw data with anomalies vs autoencoder prediction')
ax1.legend(['actual', 'predicted'])


ax2.autoscale(enable=True, axis='both', tight=True)
ax2.plot(error, 'r')
rollingMean = pd.rolling_mean (error, window=50)
ax2.plot(rollingMean, 'b')

threshLine = np.ones((len(rollingMean),1)) * .6
ax2.plot(threshLine, 'k--', linewidth = 3)
ax2.set_title('reconstruction error ')
```



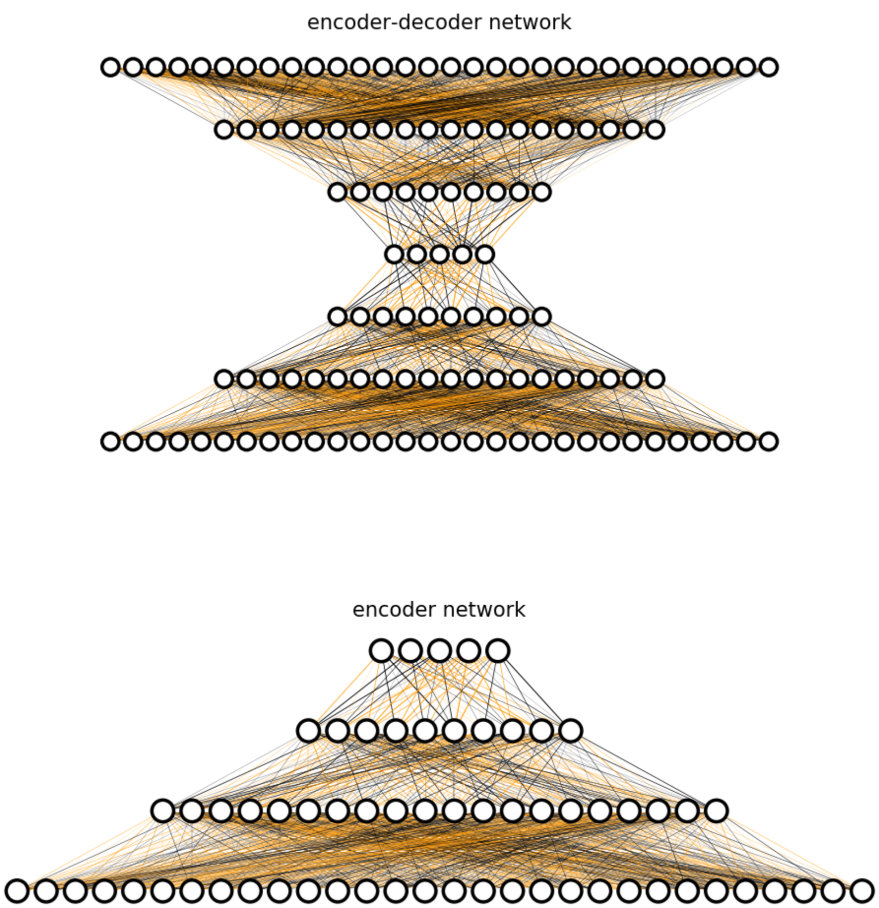

```
C:\ProgramData\Anaconda3\envs\tfGPU\lib\site-packages\ipykernel_launcher.py:15:
FutureWarning: pd.rolling_mean is deprecated for ndarrays and will be removed i
n a future version
  from ipykernel import kernelapp as app
```

Out[25]: `<matplotlib.text.Text at 0x190b8574668>`

In [ ]:

Remove Last Two Layers [focus on bottleneck activations]

In [26]:
```
plt.figure( figsize = (10, 10) )

plt.subplot(2,1,1)
nnViz.visualize_model(model)
plt.title('encoder-decoder network')

plt.subplot(2,1,2)
model.pop(); model.pop(); model.pop()
nnViz.visualize_model(model)
plt.title('encoder network')
```



Out[26]: <matplotlib.text.Text at 0x190bb99af28>

In [27]:
```
model.compile(optimizer = 'adam', loss = 'mse')
```

In [28]:
```
bottleNeckSize = hParams['bottleNeckUnits'] #model.layers[-1].get_config()['units
bottleneckActivations = windowed_predict_bottleneck_activation (model, targetData
```

number of windows: 100

PCA

In [29]:
```
from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
pca = PCA(n_components = 3)
PCA_bottleneckActivations = pca.fit_transform(bottleneckActivations)
```

Interactive Interpretation

In [30]:
```
# idea plot all points in gray -- those above threshold as ... & [ make pickable
```

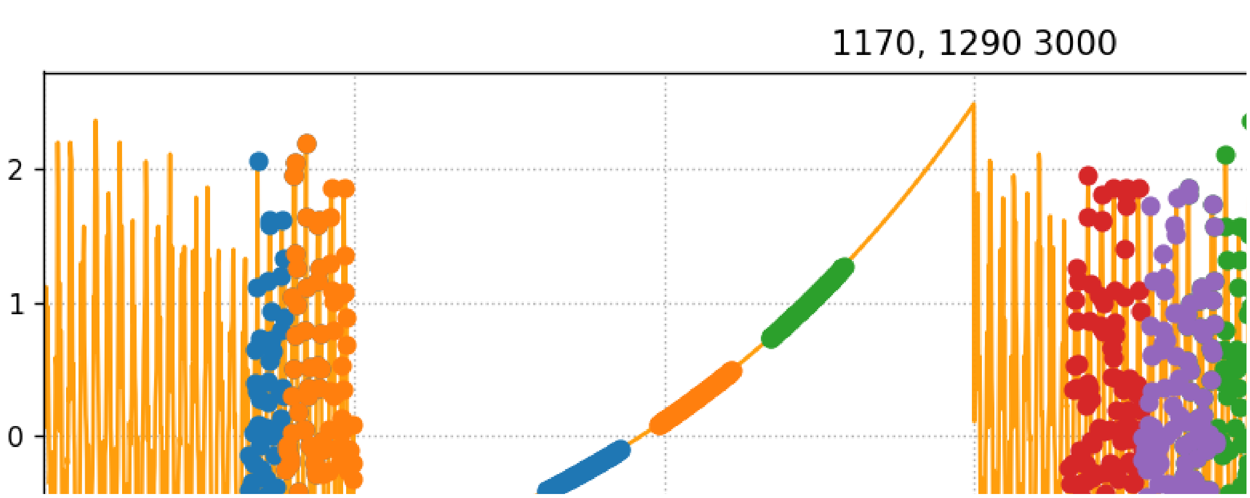
In [31]:
```
#!sudo pip3 install mpld3
```

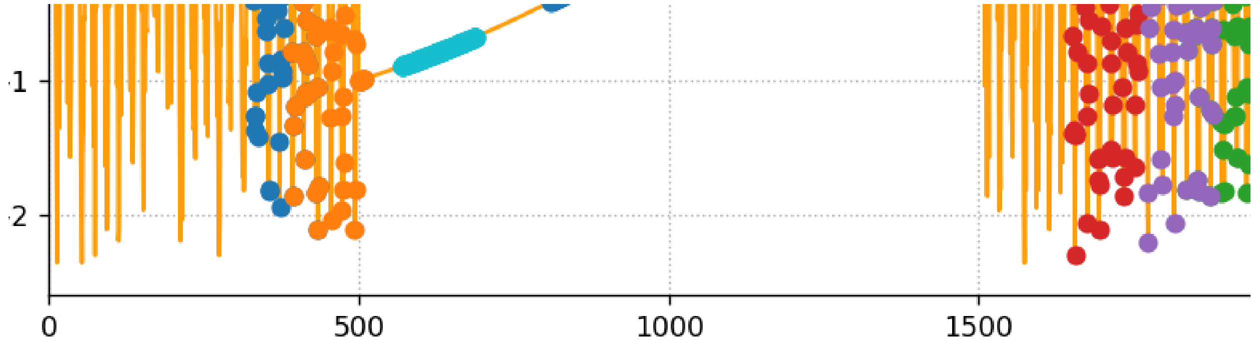
In [32]:
```
import mpld3
from mpld3 import plugins, utils
from mpl_toolkits.mplot3d import Axes3D
%matplotlib ipympl
# https://github.com/matplotlib/jupyter-matplotlib
```

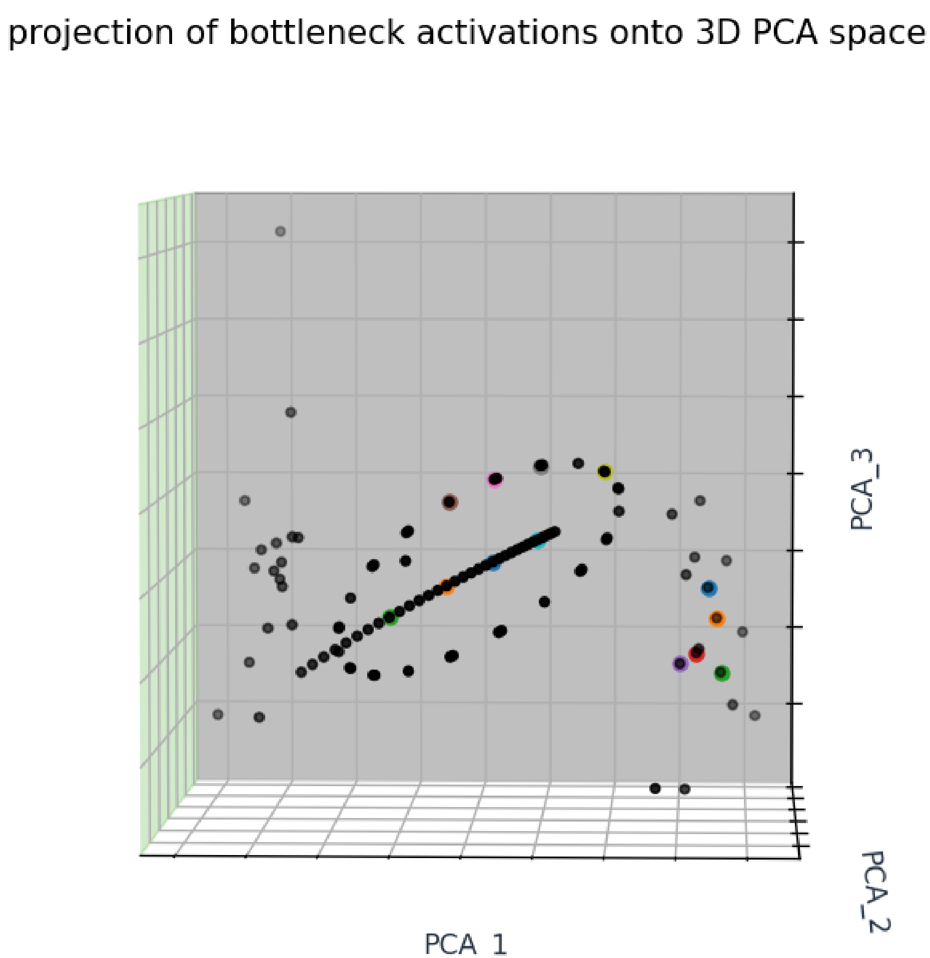
Warning: Cannot change to a different GUI toolkit: ipympl. Using notebook inste
ad.

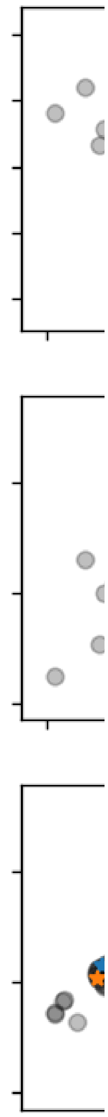
In [33]:
```
fig = plt.figure(figsize=(10,10))

plt.subplots_adjust( left = 0.02, right = 0.95, top = 0.95, bottom = 0.01, wspace

ax1 = plt.subplot2grid((5, 5), (0, 0), rowspan=2, colspan=5)
ax1.plot( range(len(targetData)), targetData, color = rawDataColor )
ax1.autoscale(enable=True, axis='x', tight=True)
ax1.legend(['raw data'], loc='upper right', shadow=True)
ax1.grid(linestyle='dotted')

ax2 = plt.subplot2grid((5, 5), (2, 0), rowspan=3, colspan=3, projection='3d')
ax2.scatter(PCA_bottleneckActivations[:, 0], PCA_bottleneckActivations[:,1], PCA_
ax2.set_xticklabels([]); ax2.set_yticklabels([]); ax2.set_zticklabels([]);
ax2.set_title('projection of bottleneck activations onto 3D PCA space')
ax2.set_xlabel('PCA_1', color=amazonSquidInk)
ax2.set_ylabel('PCA_2', color=amazonSquidInk)
ax2.set_zlabel('PCA_3', color=amazonSquidInk)

ax2.w_xaxis.set_pane_color(rawDataColor2); ax2.w_yaxis.set_pane_color(contrastCol

ax3 = plt.subplot2grid((5, 5), (2, 3), rowspan=1, colspan=2)
ax3.plot ( PCA_bottleneckActivations[:,0], PCA_bottleneckActivations[:,1], 'o', c
ax3.set_xticklabels([]); ax3.set_yticklabels([])
ax3.set_xlabel('PCA_1', color=amazonSquidInk); ax3.xaxis.set_label_coords(0.5, 0.
ax3.set_ylabel('PCA_2', color=amazonSquidInk); ax3.yaxis.set_label_coords(1.05, 0

ax4 = plt.subplot2grid((5, 5), (3, 3), rowspan=1, colspan=2)
ax4.plot ( PCA_bottleneckActivations[:,0], PCA_bottleneckActivations[:,2], 'o', c
ax4.set_xticklabels([]); ax4.set_yticklabels([])
ax4.set_xlabel('PCA_1', color=amazonSquidInk); ax4.xaxis.set_label_coords(0.5, 0.
ax4.set_ylabel('PCA_3', color=amazonSquidInk); ax4.yaxis.set_label_coords(1.05, 0

ax5 = plt.subplot2grid((5, 5), (4, 3), rowspan=1, colspan=2)
ax5.plot ( PCA_bottleneckActivations[:,1], PCA_bottleneckActivations[:,2], 'o', c
ax5.set_xticklabels([]); ax5.set_yticklabels([])
ax5.set_xlabel('PCA_2', color=amazonSquidInk); ax5.xaxis.set_label_coords(0.5, 0.
ax5.set_ylabel('PCA_3', color=amazonSquidInk); ax5.yaxis.set_label_coords(1.05, 0
```

Figure 1




1
2
0
500
1000
1500


projection of bottleneck activations onto 3D PCA space
PCA_3
PCA_2
PCA_1



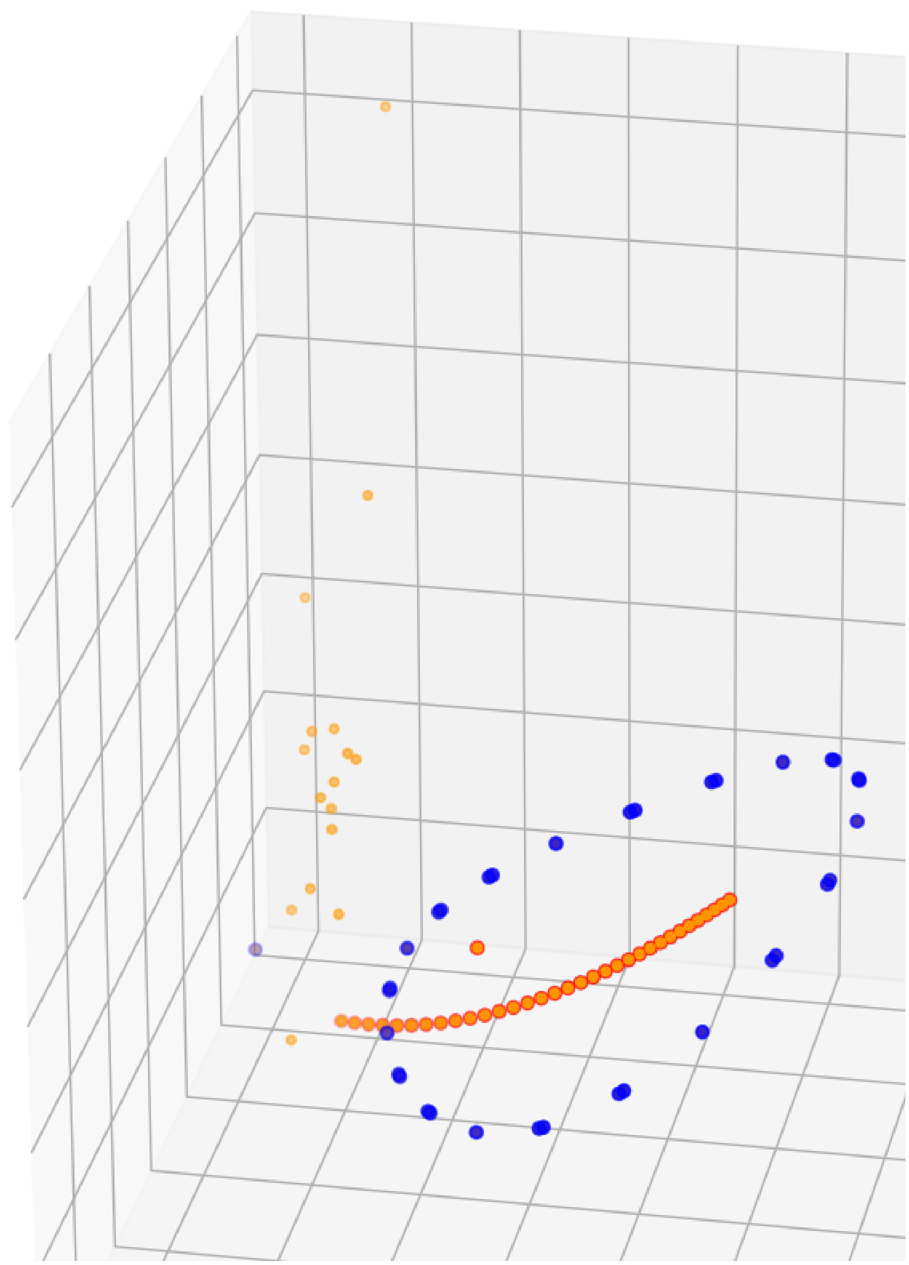
In [34]:
```
def updatePlots ( ind ):
    if len(ind) > 0:
        targetInd = ind[0]
        lowerBound = max( 0, int((targetInd-2) * hParams['inputOutputDimensionali
        upperBound = min( len(targetData), int((targetInd + 2) * hParams['inputOu

        strOut = str(lowerBound) + ', ' + str(upperBound) + ' ' + str(len(targetD
        ax1.set_title(strOut)

        ax1.plot( list(range( lowerBound, upperBound)), targetData[ lowerBound:up
        ax2.scatter(PCA_bottleneckActivations[targetInd, 0], PCA_bottleneckActiva

        ax3.plot ( PCA_bottleneckActivations[targetInd,0], PCA_bottleneckActivati
        ax4.plot ( PCA_bottleneckActivations[targetInd,0], PCA_bottleneckActivati
        ax5.plot ( PCA_bottleneckActivations[targetInd,1], PCA_bottleneckActivati
```

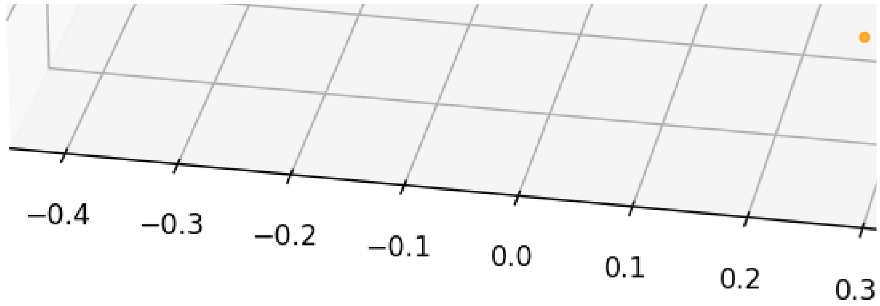
In [35]:
```
def on_pick(event):
    if event.mouseevent.inaxes != ax2:
        eventArtist = event.artist
        xdata, ydata = eventArtist.get_data()
        ind = event.ind
        updatePlots(ind)

cid = fig.canvas.mpl_connect('pick_event', on_pick)
```

End Result

In [36]:
```
fig = plt.figure( figsize = (10, 10) )
ax = fig.add_subplot(111, projection='3d')
plt.subplots_adjust( left = 0.02, right = 0.95, top = 0.95, bottom = 0.01, wspace


ax.scatter(PCA_bottleneckActivations[:, 0], PCA_bottleneckActivations[:,1], PCA_b
ax.scatter(PCA_bottleneckActivations[anomalousInds_1, 0], PCA_bottleneckActivatio
ax.scatter(PCA_bottleneckActivations[anomalousInds_2, 0], PCA_bottleneckActivatio
```

Figure 2






```
Out[36]: <mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x19041be9748>
```

Appendix - Alternative Dimensionality Reduction Methods

tSNE 3D

In [ ]:
```
from sklearn.manifold import TSNE

embeddedBottleneckActivations = TSNE(n_components = 3, perplexity = 6, learning_r
```

In [ ]:
```
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure(figsize=(10,10))
ax = fig.add_subplot(111, projection='3d')

ax.plot(embeddedBottleneckActivations[:,0], embeddedBottleneckActivations[:,1], e
ax.plot(embeddedBottleneckActivations[anomalousInds_1,0], embeddedBottleneckActiv
ax.plot(embeddedBottleneckActivations[anomalousInds_2,0], embeddedBottleneckActiv
```

In [ ]: