encoder

January 4, 2021

[29]:
```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import matplotlib.pyplot as plt
```

[103]:
```
data = np.loadtxt('data_log_norm.csv')
params = np.loadtxt('params.csv')
data_noisy = data + 0.5*np.random.normal(size = data.shape)
```

1 Designing Encoder

[104]:
```
enc_inputs = keras.Input(shape=(128 , ))
x = layers.Dense(64 , activation='relu')(enc_inputs)
x = layers.Dense(64, activation='relu')(x)
enc_outputs = layers.Dense(32, activation='relu')(x)

x = layers.Dense(64 , activation='relu')(enc_outputs)
x =  layers.Dense(64 , activation='relu')(x)
x = layers.Dense(128)(x)
dec_outputs = layers.LeakyReLU(alpha = 0.7)(x)
```

[105]:
```
encoder = keras.Model(inputs=enc_inputs , outputs=enc_outputs , name =␣
 ↪'encoder')
encoder.summary()
```

```
Model: "encoder"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_14 (InputLayer)        [(None, 128)]             0
_________________________________________________________________
dense_54 (Dense)             (None, 64)                8256
_________________________________________________________________
dense_55 (Dense)             (None, 64)                4160
_________________________________________________________________
dense_56 (Dense)             (None, 32)                2080
```

```
=================================================================
Total params: 14,496
Trainable params: 14,496
Non-trainable params: 0
_________________________________________________________________
```

[106]:

```
auto_encoder = keras.Model(inputs=enc_inputs , outputs=dec_outputs , name =␣
 ↪'autoencoder')
auto_encoder.summary()
```

```
Model: "autoencoder"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_14 (InputLayer)        [(None, 128)]             0
_________________________________________________________________
dense_54 (Dense)             (None, 64)                8256
_________________________________________________________________
dense_55 (Dense)             (None, 64)                4160
_________________________________________________________________
dense_56 (Dense)             (None, 32)                2080
_________________________________________________________________
dense_57 (Dense)             (None, 64)                2112
_________________________________________________________________
dense_58 (Dense)             (None, 64)                4160
_________________________________________________________________
dense_59 (Dense)             (None, 128)               8320
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 128)               0
=================================================================
Total params: 29,088
Trainable params: 29,088
Non-trainable params: 0
_________________________________________________________________
```

[56]:

```
auto_encoder.compile(loss='mean_absolute_error',
                optimizer=tf.keras.optimizers.Adam(0.001))
```

[57]:

```
auto_encoder.fit(data , data , validation_split=0.3 , epochs = 10)
```

```
Epoch 1/10
2188/2188 [==============================] - 3s 1ms/step - loss: 0.0122 -
val_loss: 0.0033
Epoch 2/10
2188/2188 [==============================] - 3s 1ms/step - loss: 0.0050 -
val_loss: 0.0051
Epoch 3/10
2188/2188 [==============================] - 3s 1ms/step - loss: 0.0048 -
```

```
val_loss: 0.0051
Epoch 4/10
2188/2188 [==============================] - 4s 2ms/step - loss: 0.0043 -
val_loss: 0.0041
Epoch 5/10
2188/2188 [==============================] - 4s 2ms/step - loss: 0.0041 -
val_loss: 0.0022
Epoch 6/10
2188/2188 [==============================] - 4s 2ms/step - loss: 0.0038 -
val_loss: 0.0050
Epoch 7/10
2188/2188 [==============================] - 4s 2ms/step - loss: 0.0037 -
val_loss: 0.0042
Epoch 8/10
2188/2188 [==============================] - 5s 2ms/step - loss: 0.0035 -
val_loss: 0.0032
Epoch 9/10
2188/2188 [==============================] - 4s 2ms/step - loss: 0.0034 -
val_loss: 0.0029
Epoch 10/10
2188/2188 [==============================] - 4s 2ms/step - loss: 0.0033 -
val_loss: 0.0053
```

[57]: <tensorflow.python.keras.callbacks.History at 0x7f6272afe6d8>

[107]:
```
dae = keras.Model(inputs=enc_inputs , outputs=dec_outputs , name =␣
↪'autoencoder')
dae.summary()
```

```
Model: "autoencoder"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_14 (InputLayer)        [(None, 128)]             0
_________________________________________________________________
dense_54 (Dense)             (None, 64)                8256
_________________________________________________________________
dense_55 (Dense)             (None, 64)                4160
_________________________________________________________________
dense_56 (Dense)             (None, 32)                2080
_________________________________________________________________
dense_57 (Dense)             (None, 64)                2112
_________________________________________________________________
dense_58 (Dense)             (None, 64)                4160
_________________________________________________________________
dense_59 (Dense)             (None, 128)               8320
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 128)               0
```

```
=================================================================
Total params: 29,088
Trainable params: 29,088
Non-trainable params: 0
_________________________________________________________________
```

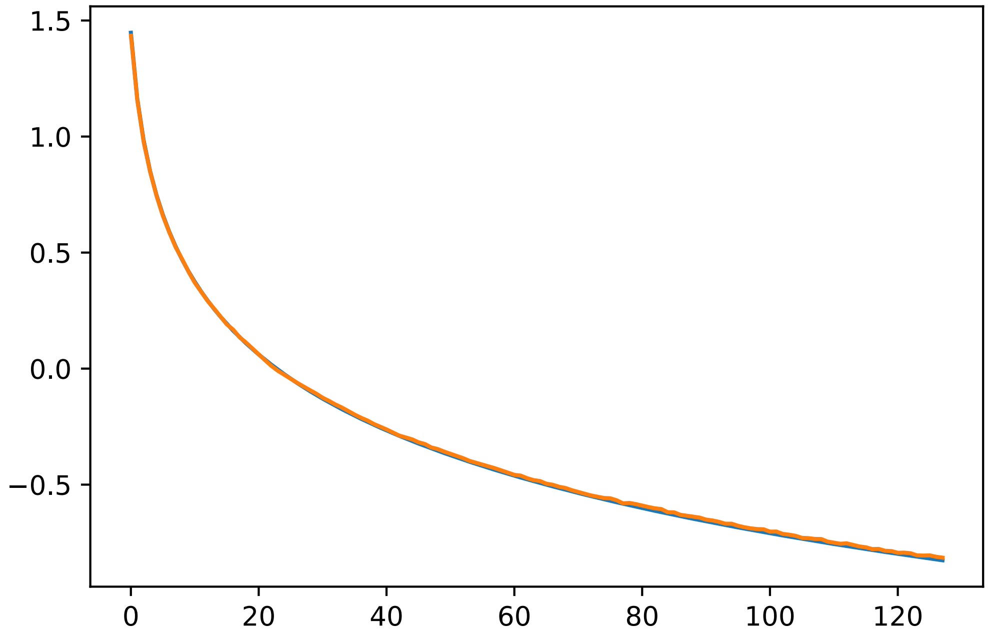
[108]:
```
dae.compile(loss='mean_absolute_error',
                optimizer=tf.keras.optimizers.Adam(0.001))
```

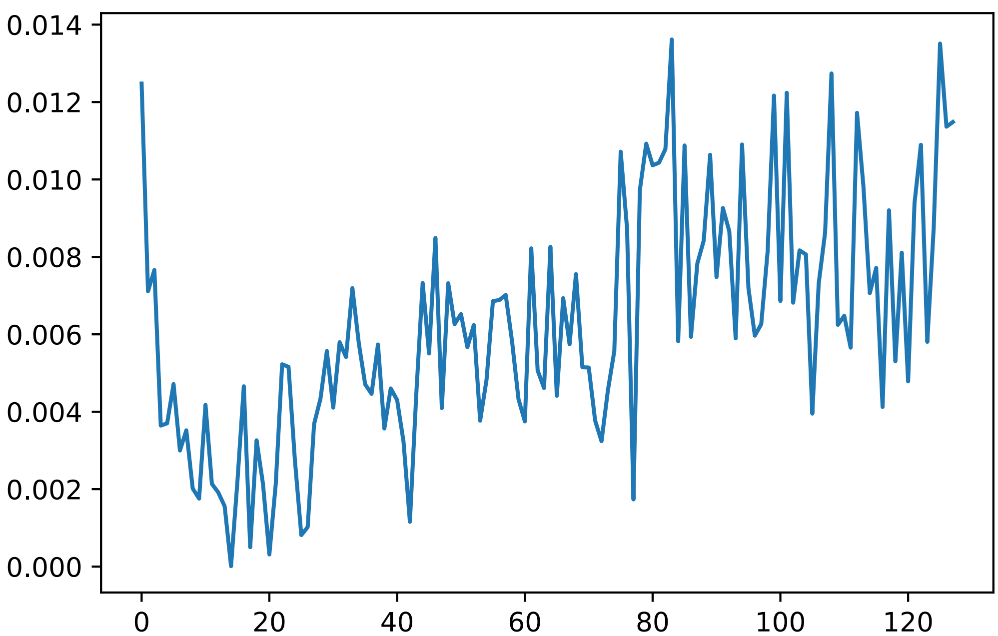
[109]:
```
dae_history = dae.fit(data_noisy , data , validation_split=0.3 , epochs = 10)
```

```
Epoch 1/10
2188/2188 [==============================] - 3s 1ms/step - loss: 0.0507 -
val_loss: 0.0441
Epoch 2/10
2188/2188 [==============================] - 3s 1ms/step - loss: 0.0424 -
val_loss: 0.0445
Epoch 3/10
2188/2188 [==============================] - 4s 2ms/step - loss: 0.0416 -
val_loss: 0.0414
Epoch 4/10
2188/2188 [==============================] - 4s 2ms/step - loss: 0.0413 -
val_loss: 0.0415
Epoch 5/10
2188/2188 [==============================] - 4s 2ms/step - loss: 0.0409 -
val_loss: 0.0421
Epoch 6/10
2188/2188 [==============================] - 4s 2ms/step - loss: 0.0406 -
val_loss: 0.0435
Epoch 7/10
2188/2188 [==============================] - 4s 2ms/step - loss: 0.0402 -
val_loss: 0.0415
Epoch 8/10
2188/2188 [==============================] - 4s 2ms/step - loss: 0.0400 -
val_loss: 0.0418
Epoch 9/10
2188/2188 [==============================] - 4s 2ms/step - loss: 0.0396 -
val_loss: 0.0414
Epoch 10/10
2188/2188 [==============================] - 4s 2ms/step - loss: 0.0395 -
val_loss: 0.0412
```

[59]:
```
d_pred = auto_encoder.predict(data[10:11])

plt.plot(data[10:11][0])
plt.plot(d_pred[0])
plt.show()
```

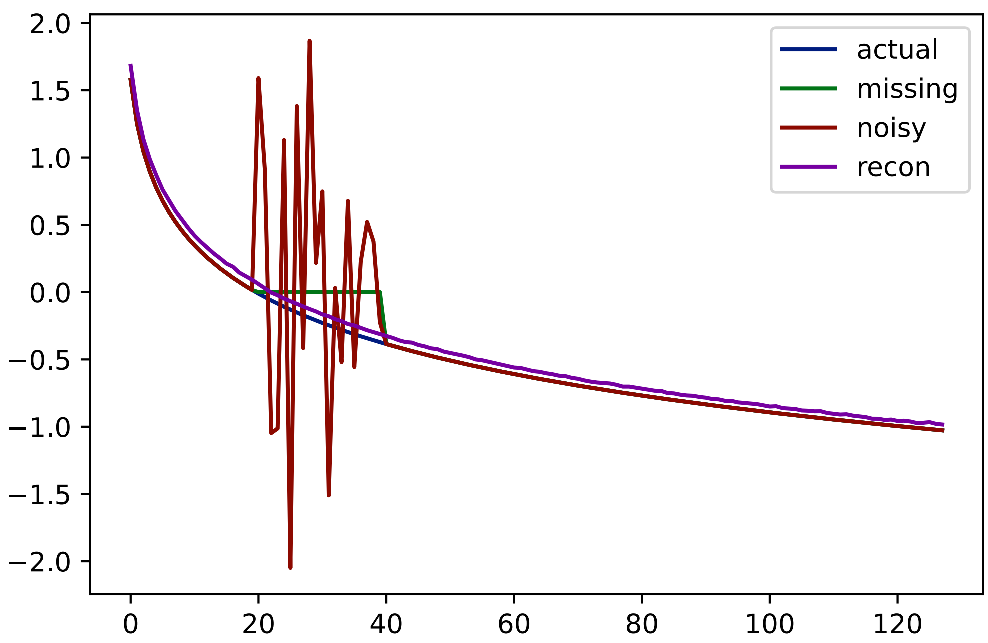
```
plt.plot(abs((data[10:11][0]-d_pred[0])))
plt.show()
```





2 Adding Noise

[97]:
```
d_missing = np.copy(data[0:1])
d_missing[0][20:40] = np.zeros(20)
d_noisy = np.copy(d_missing)
d_noisy[0][20:40] = np.random.normal(size=20)
d_pred = auto_encoder.predict(d_missing)

plt.style.use('seaborn-dark-palette')
plt.plot(data[0:1][0] , label = 'actual')
plt.plot(d_missing[0] , label='missing')
plt.plot(d_noisy[0] , label='noisy')
plt.plot(d_pred[0] , label='recon')
plt.legend(['actual' , 'missing' , 'noisy' , 'recon'])
plt.show()
#plt.plot(abs((data[10:11][0]-d_pred[0])))
#plt.show()
```



[98]:
```
pred_model = keras.models.load_model('fc_model')
pred_model.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_21 (Dense)             (None, 64)                8256
```

```
_________________________________________________________________
dense_22 (Dense)             (None, 64)                4160
_________________________________________________________________
dense_23 (Dense)             (None, 2)                 130
=================================================================
Total params: 12,546
Trainable params: 12,546
Non-trainable params: 0
_________________________________________________________________
```

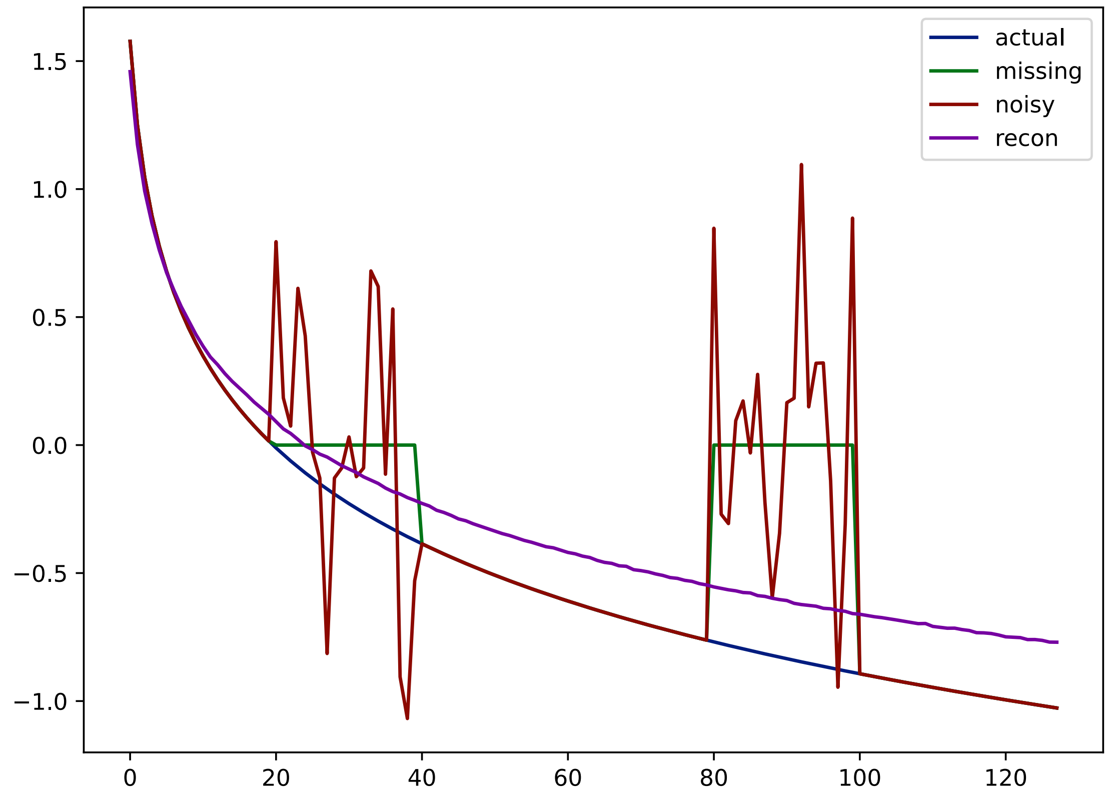
[101]:

```
proper_pred = pred_model.predict(data[0:1])
noisy_pred = pred_model.predict(d_pred)
missing_pred = pred_model.predict(d_missing)
true_p = params[0:1]
print(proper_pred)
print(missing_pred)
print(noisy_pred)
print(true_p)
```

```
[[2.2370088 1.7263913]]
[[2.324698  1.1990267]]
[[2.2958343 1.3359421]]
[[2.23685133 1.7316627 ]]
```

[146]:

```
d_missing = np.copy(data[0:1])
d_missing[0][20:40] = np.zeros(20)
d_missing[0][80:100] = np.zeros(20)
d_noisy = np.copy(d_missing)
d_noisy[0][20:40] = 0.5*np.random.normal(size=20)
d_noisy[0][80:100] = 0.5*np.random.normal(size=20)
d_pred = dae.predict(d_noisy)

plt.style.use('seaborn-dark-palette')
plt.figure(figsize=(8,6))
plt.plot(data[0:1][0] , label = 'actual')
plt.plot(d_missing[0] , label='missing')
plt.plot(d_noisy[0] , label='noisy')
plt.plot(d_pred[0] , label='recon')
plt.legend(['actual' , 'missing' , 'noisy' , 'recon'])
plt.show()
#plt.plot(abs((data[10:11][0]-d_pred[0])))
#plt.show()
```



Problem im getting now is , since we are adding normal distribution with mean 0 and variance 1.0 hence in the prediction it just raises the value of the lower side,

we want the mean of the noise being added to follow the same trend as that of the data value only , so once we have found the likely data trend by denoising auto encoder , we can again 'add'(not multiply) this output (denoised output) to the noisy data , so the mean of distribution will get closer to the actual distribution. then call this as the new noisy data the once again pass this through denoiser ,

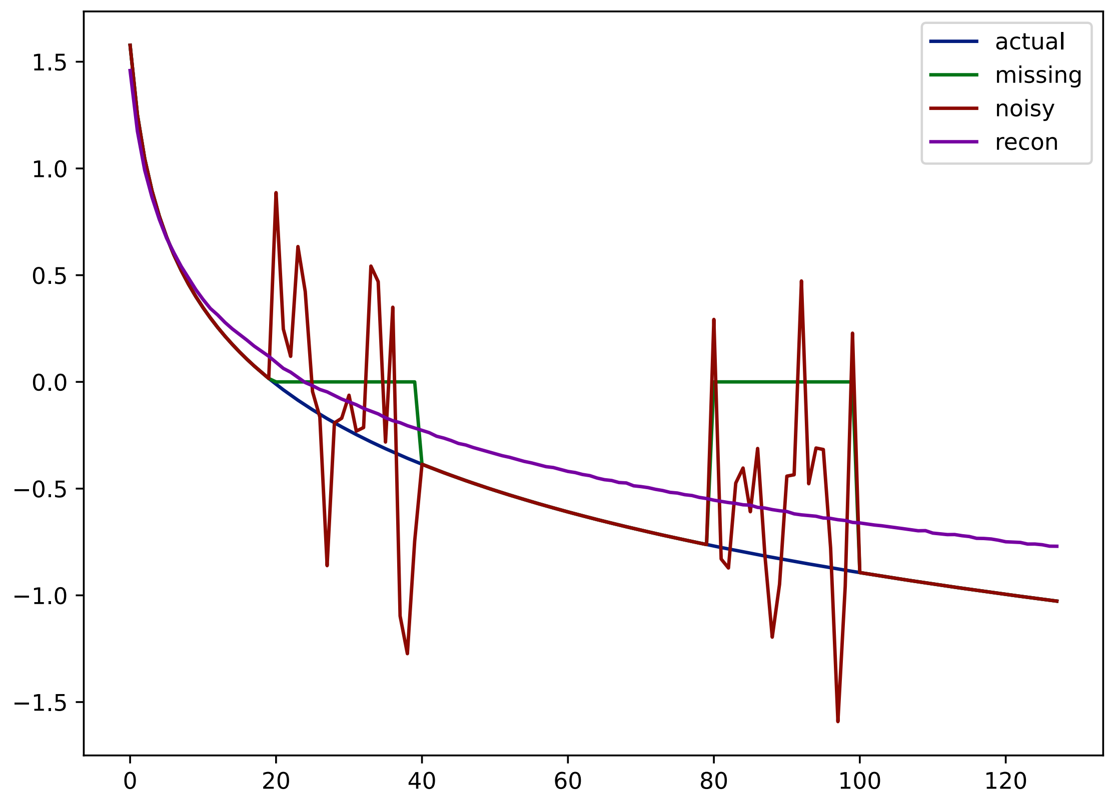
continue this loop so one solution maybe that ,

[134]:
```
proper_pred = pred_model.predict(data[0:1])
noisy_pred = pred_model.predict(d_pred)
missing_pred = pred_model.predict(d_missing)
true_p = params[0:1]

print('true_param:' , true_p)
print('Proper data prediction:' , proper_pred)
print('prediction on missing data:' , missing_pred)
print('prediction on reconstructed data:' , noisy_pred)
```

```
true_param: [[2.23685133 1.7316627 ]]
Proper data prediction: [[2.2370088 1.7263913]]
```

```
prediction on missing data: [[1.8855422 2.5612636]]
prediction on reconstructed data: [[1.8785461 1.6719625]]
```

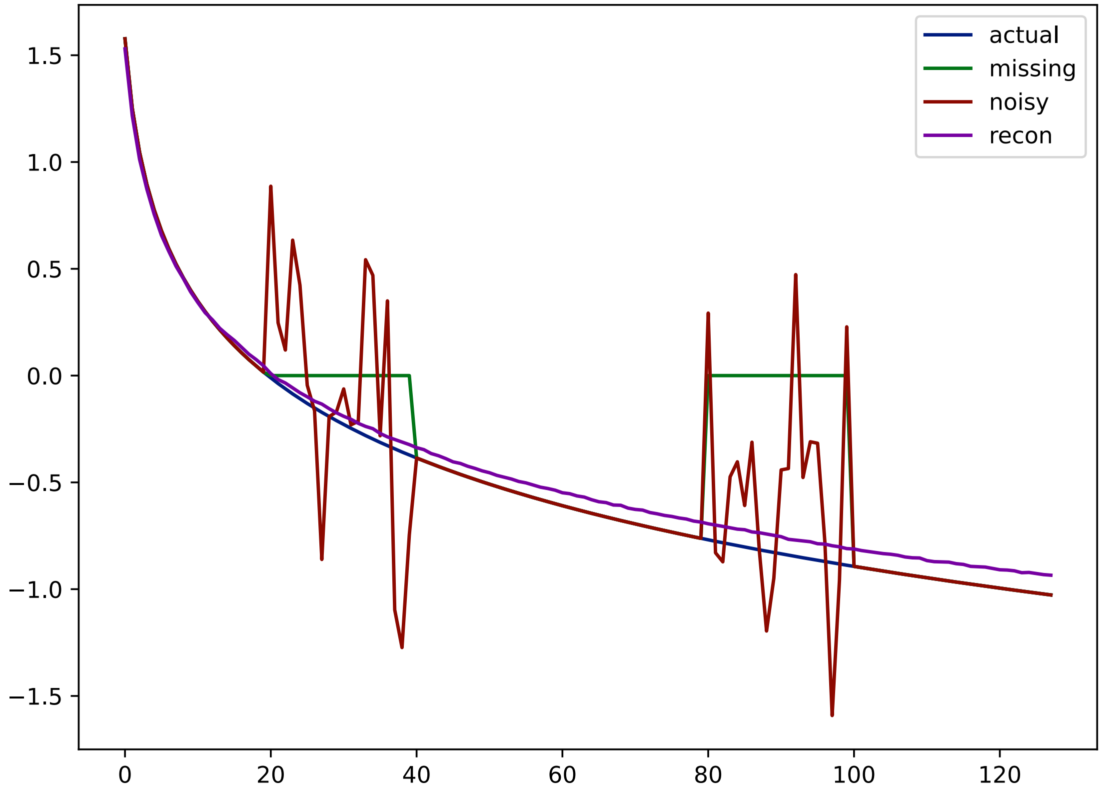
[147]:
```
d_noisy[0][20:40] = d_pred[0][20:40]+d_noisy[0][20:40]
d_noisy[0][80:100] = d_pred[0][80:100]+d_noisy[0][80:100]

plt.style.use('seaborn-dark-palette')
plt.figure(figsize=(8,6))
plt.plot(data[0:1][0] , label = 'actual')
plt.plot(d_missing[0] , label='missing')
plt.plot(d_noisy[0] , label='noisy')
plt.plot(d_pred[0] , label='recon')
plt.legend(['actual' , 'missing' , 'noisy' , 'recon'])
plt.show()
#plt.plot(abs((data[10:11][0]-d_pred[0])))
#plt.show()
```



see above plot , now the noise mean seems to follow the likely distribution , lets do denoising again on this modified noise data

[148]:
```
d_pred = dae.predict(d_noisy)
```

```
plt.style.use('seaborn-dark-palette')
plt.figure(figsize=(8,6))
plt.plot(data[0:1][0] , label = 'actual')
plt.plot(d_missing[0] , label='missing')
plt.plot(d_noisy[0] , label='noisy')
plt.plot(d_pred[0] , label='recon')
plt.legend(['actual' , 'missing' , 'noisy' , 'recon'])
plt.show()
```



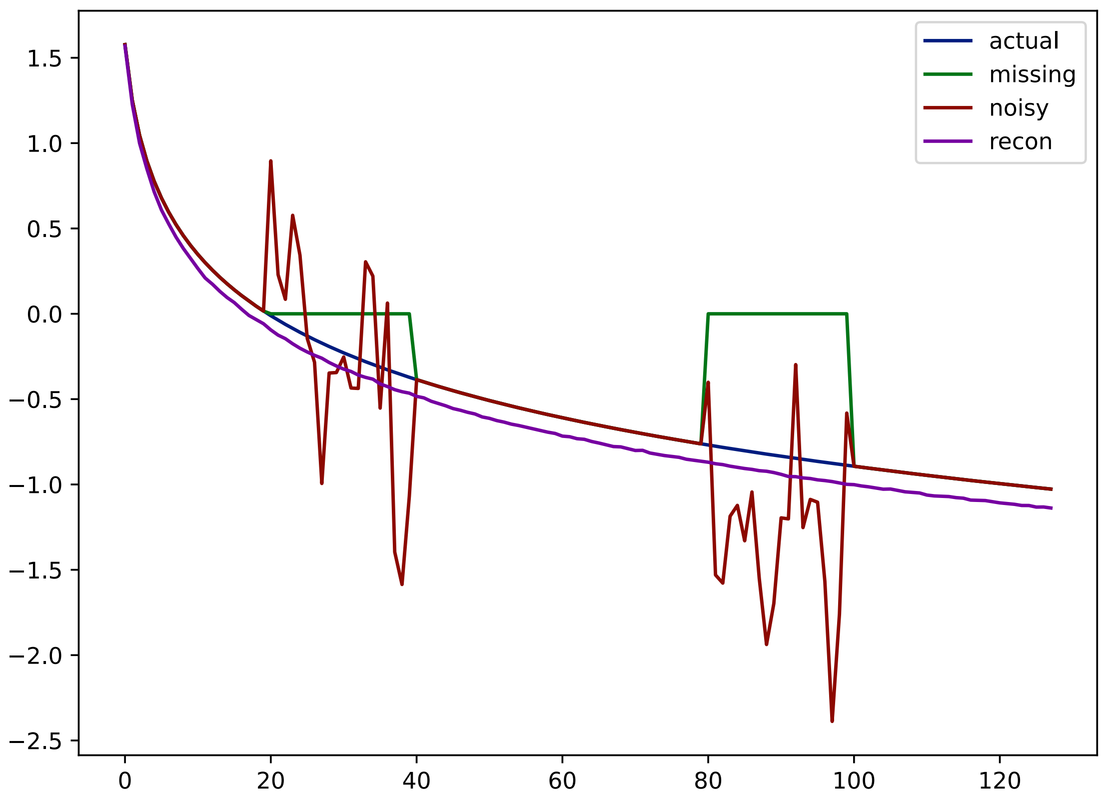
see the above plot , now the denoised output seems more closer to the true data , lets see the parameters prediction values now

[149]:
```
proper_pred = pred_model.predict(data[0:1])
noisy_pred = pred_model.predict(d_pred)
missing_pred = pred_model.predict(d_missing)
true_p = params[0:1]

print('true_param:' , true_p)
print('Proper data prediction:' , proper_pred)
print('prediction on missing data:' , missing_pred)
print('prediction on reconstructed data:' , noisy_pred)
```

```
true_param: [[2.23685133 1.7316627 ]]
Proper data prediction: [[2.2370088 1.7263913]]
prediction on missing data: [[1.8855422 2.5612636]]
prediction on reconstructed data: [[2.0948975 1.8685734]]
```

Now compare this with earlier predictions,now its much much better lets run this loop a few more times

[150]:
```python
d_noisy[0][20:40] = d_pred[0][20:40]+d_noisy[0][20:40]
d_noisy[0][80:100] = d_pred[0][80:100]+d_noisy[0][80:100]

d_pred = dae.predict(d_noisy)

plt.style.use('seaborn-dark-palette')
plt.figure(figsize=(8,6))
plt.plot(data[0:1][0] , label = 'actual')
plt.plot(d_missing[0] , label='missing')
plt.plot(d_noisy[0] , label='noisy')
plt.plot(d_pred[0] , label='recon')
plt.legend(['actual' , 'missing' , 'noisy' , 'recon'])
plt.show()
#plt.plot(abs((data[10:11][0]-d_pred[0])))
#plt.show()
```

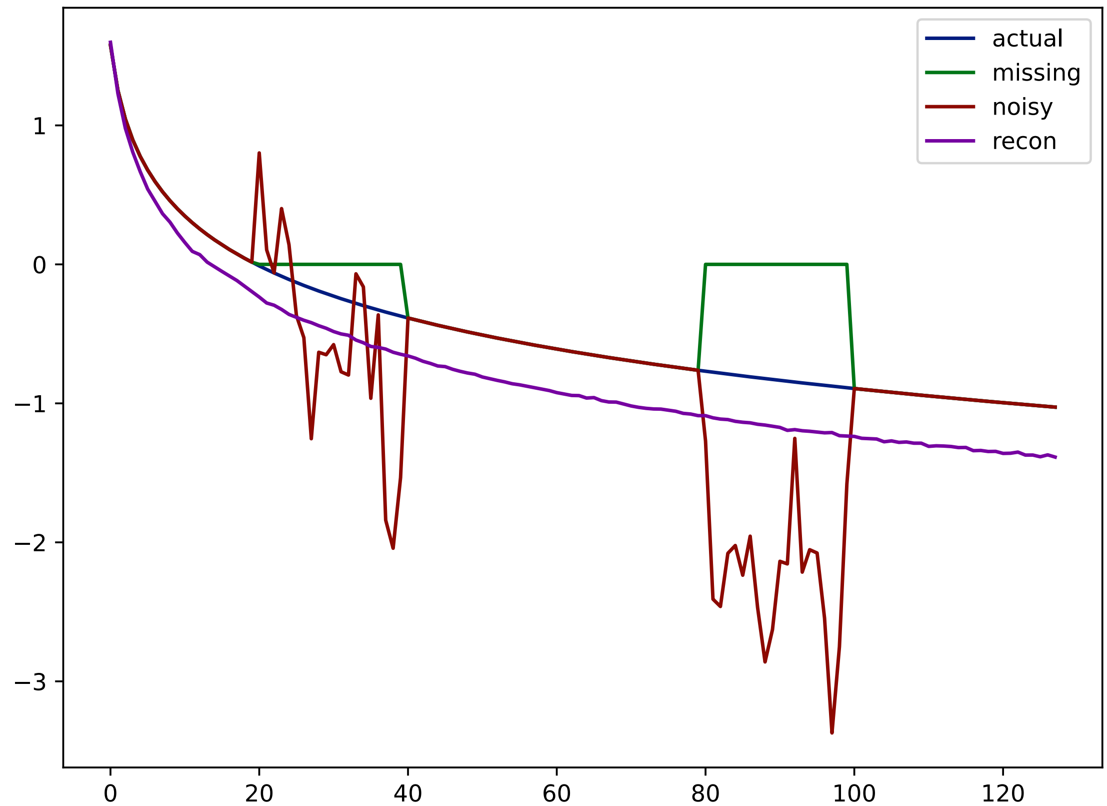


[151]:
```
proper_pred = pred_model.predict(data[0:1])
noisy_pred = pred_model.predict(d_pred)
missing_pred = pred_model.predict(d_missing)
true_p = params[0:1]

print('true_param:' , true_p)
print('Proper data prediction:' , proper_pred)
print('prediction on missing data:' , missing_pred)
print('prediction on reconstructed data:' , noisy_pred)
```

```
true_param: [[2.23685133 1.7316627 ]]
Proper data prediction: [[2.2370088 1.7263913]]
prediction on missing data: [[1.8855422 2.5612636]]
prediction on reconstructed data: [[2.2958224 2.0246103]]
```

Ahhh , now we have a problem , too much is getting subtracted , maybe addition is not a good idea , but it sure works for one loop

[152]:
```
d_noisy[0][20:40] = d_pred[0][20:40]+d_noisy[0][20:40]
d_noisy[0][80:100] = d_pred[0][80:100]+d_noisy[0][80:100]

d_pred = dae.predict(d_noisy)

plt.style.use('seaborn-dark-palette')
plt.figure(figsize=(8,6))
plt.plot(data[0:1][0] , label = 'actual')
plt.plot(d_missing[0] , label='missing')
plt.plot(d_noisy[0] , label='noisy')
plt.plot(d_pred[0] , label='recon')
plt.legend(['actual' , 'missing' , 'noisy' , 'recon'])
plt.show()
#plt.plot(abs((data[10:11][0]-d_pred[0])))
#plt.show()
```



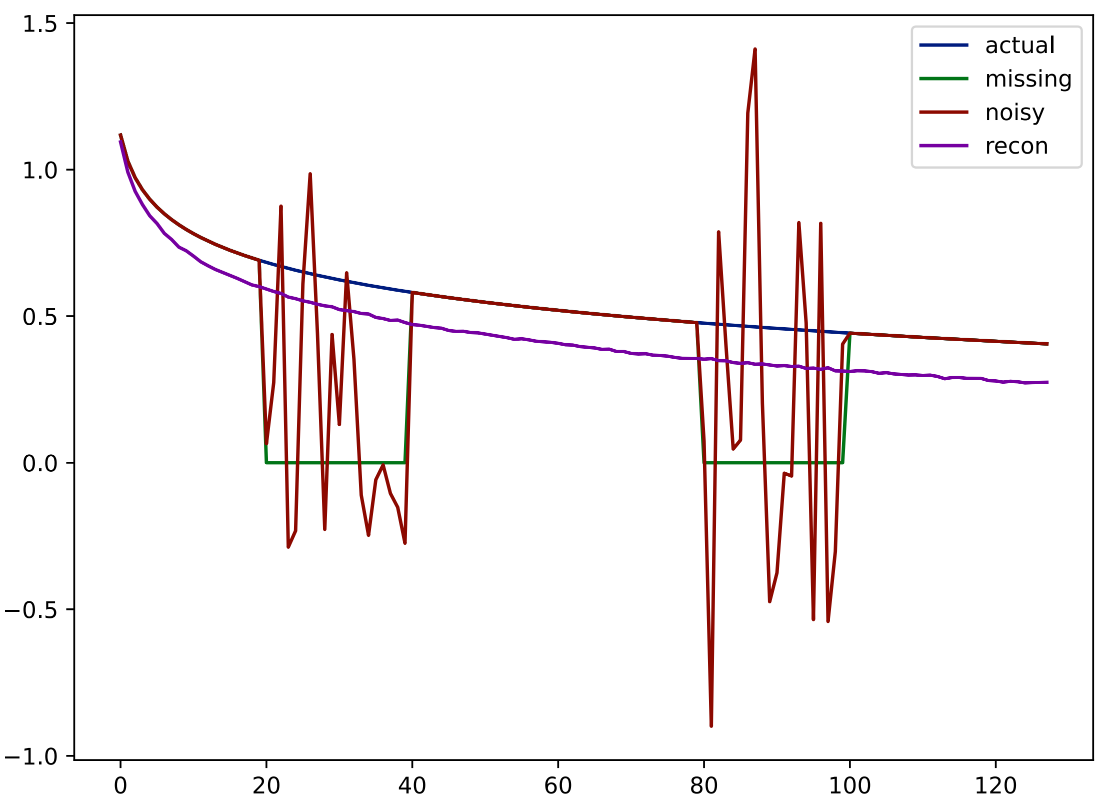
[153]:
```
proper_pred = pred_model.predict(data[0:1])
noisy_pred = pred_model.predict(d_pred)
missing_pred = pred_model.predict(d_missing)
true_p = params[0:1]

print('true_param:' , true_p)
print('Proper data prediction:' , proper_pred)
print('prediction on missing data:' , missing_pred)
print('prediction on reconstructed data:' , noisy_pred)
```

```
true_param: [[2.23685133 1.7316627 ]]
Proper data prediction: [[2.2370088 1.7263913]]
prediction on missing data: [[1.8855422 2.5612636]]
prediction on reconstructed data: [[2.5427022 2.2137847]]
```

Lets see the entire thing on some other data - point obviously , i wolud need to quantify this entire procedure for a more qualitative approach , like defining the loss for entire dataset (maybe)

[172]:
```
d_missing = np.copy(data[18:19])
d_missing[0][20:40] = np.zeros(20)
d_missing[0][80:100] = np.zeros(20)
d_noisy = np.copy(d_missing)
```

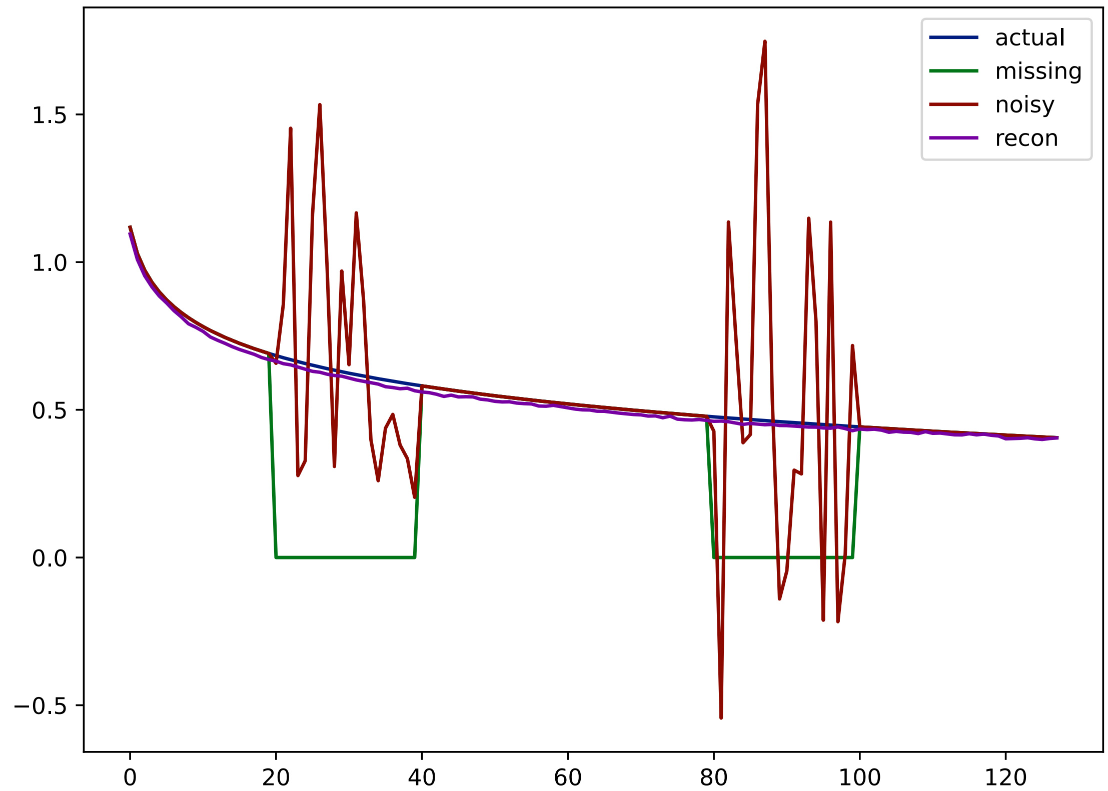
```
d_noisy[0][20:40] = 0.5*np.random.normal(size=20)
d_noisy[0][80:100] = 0.5*np.random.normal(size=20)
d_pred = auto_encoder.predict(d_noisy)

plt.style.use('seaborn-dark-palette')
plt.figure(figsize=(8,6))
plt.plot(data[18:19][0] , label = 'actual')
plt.plot(d_missing[0] , label='missing')
plt.plot(d_noisy[0] , label='noisy')
plt.plot(d_pred[0] , label='recon')
plt.legend(['actual' , 'missing' , 'noisy' , 'recon'])
plt.show()
#plt.plot(abs((data[10:11][0]-d_pred[0])))
#plt.show()
proper_pred = pred_model.predict(data[18:19])
noisy_pred = pred_model.predict(d_pred)
missing_pred = pred_model.predict(d_missing)
true_p = params[18:19]

print('true_param:' , true_p)
print('Proper data prediction:' , proper_pred)
print('prediction on missing data:' , missing_pred)
print('prediction on reconstructed data:' , noisy_pred)
```



```
true_param: [[0.61177407 0.69605106]]
Proper data prediction: [[0.6131964  0.70847195]]
prediction on missing data: [[0.61389947 7.914525  ]]
prediction on reconstructed data: [[0.70663166 1.4621823 ]]
```

[173]:

```
d_noisy[0][20:40] = d_pred[0][20:40]+d_noisy[0][20:40]
d_noisy[0][80:100] = d_pred[0][80:100]+d_noisy[0][80:100]

d_pred = auto_encoder.predict(d_noisy)

plt.style.use('seaborn-dark-palette')
plt.figure(figsize=(8,6))
plt.plot(data[18:19][0] , label = 'actual')
plt.plot(d_missing[0] , label='missing')
plt.plot(d_noisy[0] , label='noisy')
plt.plot(d_pred[0] , label='recon')
plt.legend(['actual' , 'missing' , 'noisy' , 'recon'])
plt.show()
#plt.plot(abs((data[10:11][0]-d_pred[0])))
#plt.show()
proper_pred = pred_model.predict(data[18:19])
noisy_pred = pred_model.predict(d_pred)
missing_pred = pred_model.predict(d_missing)
true_p = params[18:19]

print('true_param:' , true_p)
print('Proper data prediction:' , proper_pred)
print('prediction on missing data:' , missing_pred)
print('prediction on reconstructed data:' , noisy_pred)
```

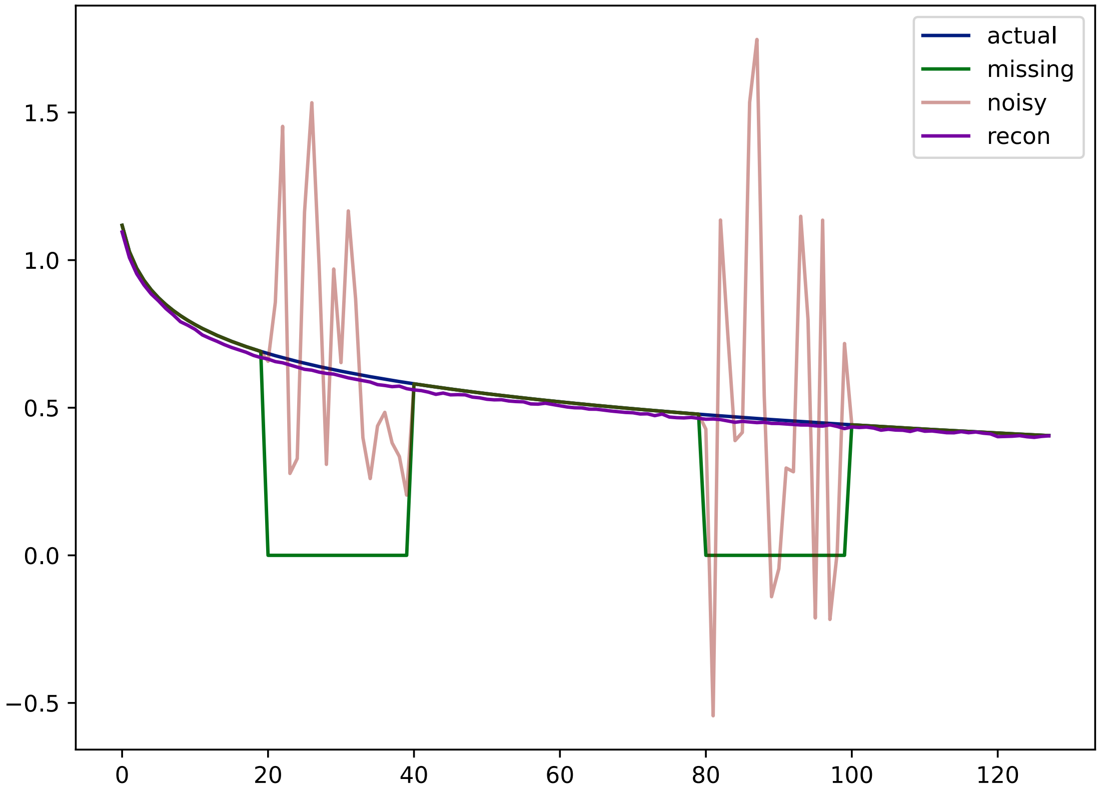



```
true_param: [[0.61177407 0.69605106]]
Proper data prediction: [[0.6131964  0.70847195]]
prediction on missing data: [[0.61389947 7.914525  ]]
prediction on reconstructed data: [[0.59474885 0.96093726]]
```

[174]:

```
plt.style.use('seaborn-dark-palette')
plt.figure(figsize=(8,6))
plt.plot(data[18:19][0] , label = 'actual')
plt.plot(d_missing[0] , label='missing')
plt.plot(d_noisy[0] , label='noisy' , alpha = 0.4)
plt.plot(d_pred[0] , label='recon')
plt.legend(['actual' , 'missing' , 'noisy' , 'recon'])
plt.show()
#plt.plot(abs((data[10:11][0]-d_pred[0])))
#plt.show()
proper_pred = pred_model.predict(data[18:19])
noisy_pred = pred_model.predict(d_pred)
missing_pred = pred_model.predict(d_missing)
true_p = params[18:19]

print('true_param:' , true_p)
print('Proper data prediction:' , proper_pred)
```

```
print('prediction on missing data:' , missing_pred)
print('prediction on reconstructed data:' , noisy_pred)
```




```
true_param: [[0.61177407 0.69605106]]
Proper data prediction: [[0.6131964  0.70847195]]
prediction on missing data: [[0.61389947 7.914525  ]]
prediction on reconstructed data: [[0.59474885 0.96093726]]
```

[ ]: