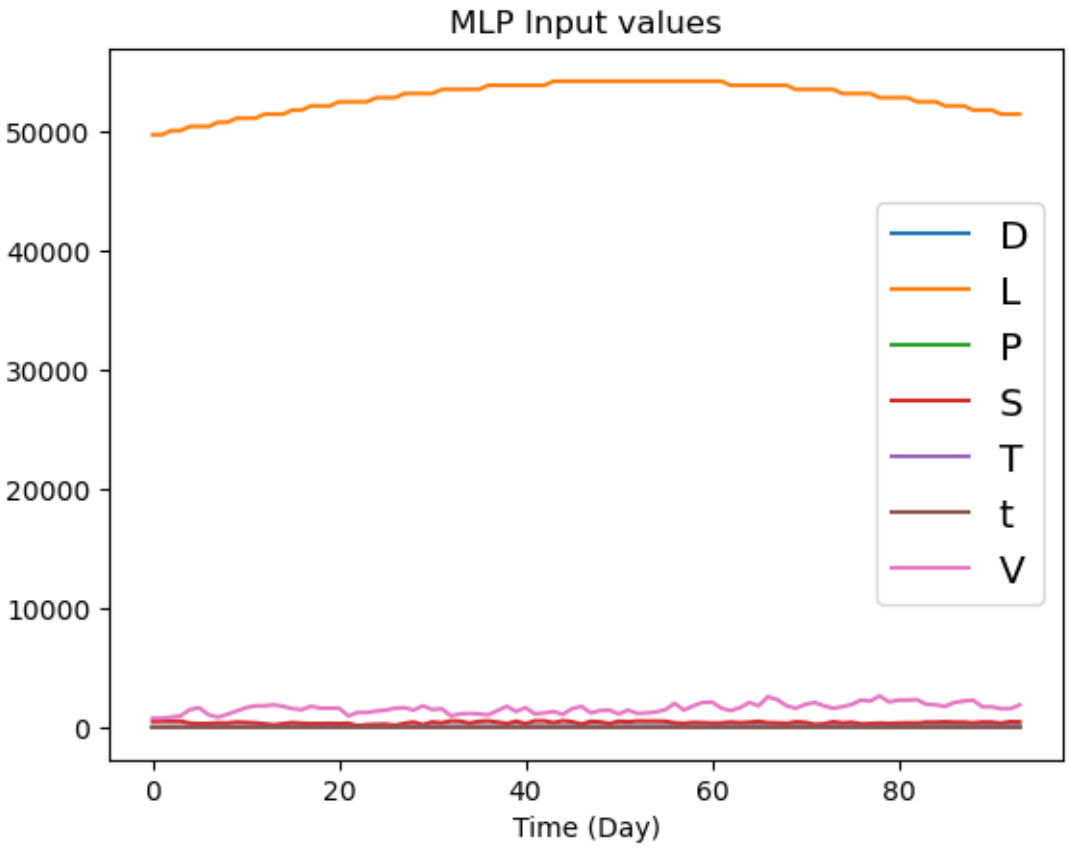
MLP model

November 24, 2024

[8]:
```
# first neural network with keras tutorial
import tensorflow as tf
from tensorflow import keras
from numpy import loadtxt
from keras.models import Sequential
from keras.layers import Dense
#from keras.wrappers.scikit_learn import KerasRegressor
from scikeras.wrappers import KerasClassifier, KerasRegressor
from sklearn.model_selection import cross_val_score
from sklearn.datasets import make_regression
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_absolute_error
from numpy import asarray
from numpy import unique
from numpy import argmax
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
import numpy as np
from tensorflow.keras.utils import plot_model
```

[12]:
```
# load the dataset
dataset = loadtxt('Data/Datas.csv', delimiter=',')
```

[14]:
```
#Plot the input
X = dataset[:,0:7]
y = dataset[:,7]
plt.plot(X)
plt.xlabel('Time (Day)')
#plt.ylabel('Crop Yield')
plt.title('MLP Input values')
plt.legend("DLPSTtV", fontsize="x-large")
plt.show()
```

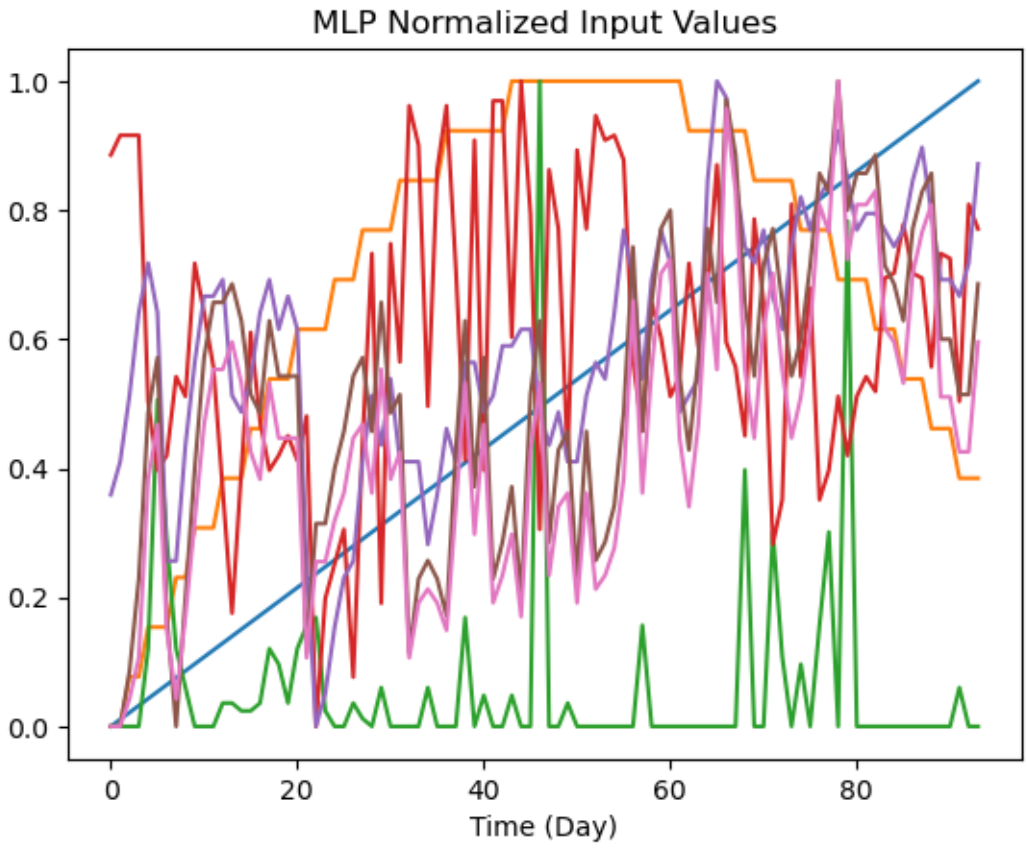


[16]:
```
# #ESSAI 3  -----OK
X = dataset[:,0:7]
y = dataset[:,7]
scalarX, scalarY = MinMaxScaler(feature_range=(0,1)),␣
 ↪MinMaxScaler(feature_range=(0,0.78272))
scalarX.fit(X)
scalarY.fit(y.reshape(94,1))
X = scalarX.transform(X)
y=np.array(y).reshape(94,1)
y = scalarY.transform(y)
```

[18]:
```
#print(y)
```

[20]:
```
#Plot the Normalized input
plt.plot(X)
plt.xlabel('Time (Day)')
#plt.ylabel('Crop Yield')
plt.title('MLP Normalized Input Values')
#plt.legend("DLPSTtV", fontsize="x-large")
#plt.legend(loc="upper left")
```

```
plt.show()
```



[22]:
```
# define the keras model
model = Sequential()
model.add(Dense(20, input_dim=7, kernel_initializer='normal',␣
↪activation='relu')) #kernel_initializer='normal'
model.add(Dense(1, kernel_initializer='normal', activation='linear')) #linear
print(model.summary())
```

```
H:\Anaconda\Lib\site-packages\keras\src\layers\core\dense.py:87: UserWarning: Do
not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #

```
 dense (Dense)                (None, 20)                160

 dense_1 (Dense)              (None, 1)                 21


 Total params: 181 (724.00 B)

 Trainable params: 181 (724.00 B)

 Non-trainable params: 0 (0.00 B)
```

None

[24]:
```
#Model optimization: SGD and Adam
sgd = tf.keras.optimizers.SGD(learning_rate=0.01, decay=0.0, momentum=0.7,␣
 ↪nesterov=False)
adam=tf.keras.optimizers.Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999,␣
 ↪epsilon=1e-8)
model.compile(optimizer=sgd, loss='mean_absolute_error')
```

```
H:\Anaconda\Lib\site-packages\keras\src\optimizers\base_optimizer.py:86:
UserWarning: Argument `decay` is no longer supported and will be ignored.
  warnings.warn(
```

[26]:
```
# fit the keras model on the dataset
history = model.fit(X, y, epochs=100, batch_size=4, verbose=2,␣
 ↪validation_split=0.20)
```

```
Epoch 1/100
19/19 - 0s - 21ms/step - loss: 0.0869 - val_loss: 0.6607
Epoch 2/100
19/19 - 0s - 2ms/step - loss: 0.0832 - val_loss: 0.6378
Epoch 3/100
19/19 - 0s - 2ms/step - loss: 0.0812 - val_loss: 0.6612
Epoch 4/100
19/19 - 0s - 2ms/step - loss: 0.0825 - val_loss: 0.6583
Epoch 5/100
19/19 - 0s - 2ms/step - loss: 0.0802 - val_loss: 0.6564
Epoch 6/100
19/19 - 0s - 2ms/step - loss: 0.0792 - val_loss: 0.6526
Epoch 7/100
19/19 - 0s - 2ms/step - loss: 0.0801 - val_loss: 0.6340
Epoch 8/100
19/19 - 0s - 2ms/step - loss: 0.0803 - val_loss: 0.6461
Epoch 9/100
19/19 - 0s - 2ms/step - loss: 0.0788 - val_loss: 0.6614
```

```
Epoch 10/100
19/19 - 0s - 2ms/step - loss: 0.0757 - val_loss: 0.5887
Epoch 11/100
19/19 - 0s - 2ms/step - loss: 0.0803 - val_loss: 0.6099
Epoch 12/100
19/19 - 0s - 2ms/step - loss: 0.0748 - val_loss: 0.6328
Epoch 13/100
19/19 - 0s - 2ms/step - loss: 0.0773 - val_loss: 0.6254
Epoch 14/100
19/19 - 0s - 2ms/step - loss: 0.0756 - val_loss: 0.6123
Epoch 15/100
19/19 - 0s - 2ms/step - loss: 0.0756 - val_loss: 0.5734
Epoch 16/100
19/19 - 0s - 2ms/step - loss: 0.0750 - val_loss: 0.5902
Epoch 17/100
19/19 - 0s - 2ms/step - loss: 0.0728 - val_loss: 0.6044
Epoch 18/100
19/19 - 0s - 2ms/step - loss: 0.0705 - val_loss: 0.5827
Epoch 19/100
19/19 - 0s - 2ms/step - loss: 0.0711 - val_loss: 0.5399
Epoch 20/100
19/19 - 0s - 2ms/step - loss: 0.0700 - val_loss: 0.5367
Epoch 21/100
19/19 - 0s - 2ms/step - loss: 0.0714 - val_loss: 0.5600
Epoch 22/100
19/19 - 0s - 2ms/step - loss: 0.0672 - val_loss: 0.5423
Epoch 23/100
19/19 - 0s - 2ms/step - loss: 0.0672 - val_loss: 0.5423
Epoch 24/100
19/19 - 0s - 2ms/step - loss: 0.0674 - val_loss: 0.5109
Epoch 25/100
19/19 - 0s - 2ms/step - loss: 0.0659 - val_loss: 0.5289
Epoch 26/100
19/19 - 0s - 2ms/step - loss: 0.0650 - val_loss: 0.5182
Epoch 27/100
19/19 - 0s - 2ms/step - loss: 0.0631 - val_loss: 0.5053
Epoch 28/100
19/19 - 0s - 2ms/step - loss: 0.0611 - val_loss: 0.4949
Epoch 29/100
19/19 - 0s - 2ms/step - loss: 0.0634 - val_loss: 0.4578
Epoch 30/100
19/19 - 0s - 2ms/step - loss: 0.0583 - val_loss: 0.4959
Epoch 31/100
19/19 - 0s - 2ms/step - loss: 0.0595 - val_loss: 0.4386
Epoch 32/100
19/19 - 0s - 2ms/step - loss: 0.0621 - val_loss: 0.4592
Epoch 33/100
19/19 - 0s - 2ms/step - loss: 0.0586 - val_loss: 0.4133
```

```
Epoch 34/100
19/19 - 0s - 2ms/step - loss: 0.0564 - val_loss: 0.4585
Epoch 35/100
19/19 - 0s - 2ms/step - loss: 0.0575 - val_loss: 0.4086
Epoch 36/100
19/19 - 0s - 2ms/step - loss: 0.0574 - val_loss: 0.4264
Epoch 37/100
19/19 - 0s - 2ms/step - loss: 0.0537 - val_loss: 0.4183
Epoch 38/100
19/19 - 0s - 2ms/step - loss: 0.0538 - val_loss: 0.4093
Epoch 39/100
19/19 - 0s - 2ms/step - loss: 0.0499 - val_loss: 0.3848
Epoch 40/100
19/19 - 0s - 2ms/step - loss: 0.0504 - val_loss: 0.3914
Epoch 41/100
19/19 - 0s - 2ms/step - loss: 0.0455 - val_loss: 0.3474
Epoch 42/100
19/19 - 0s - 2ms/step - loss: 0.0467 - val_loss: 0.3564
Epoch 43/100
19/19 - 0s - 2ms/step - loss: 0.0468 - val_loss: 0.3317
Epoch 44/100
19/19 - 0s - 2ms/step - loss: 0.0466 - val_loss: 0.3052
Epoch 45/100
19/19 - 0s - 2ms/step - loss: 0.0416 - val_loss: 0.2978
Epoch 46/100
19/19 - 0s - 2ms/step - loss: 0.0395 - val_loss: 0.3397
Epoch 47/100
19/19 - 0s - 2ms/step - loss: 0.0408 - val_loss: 0.2803
Epoch 48/100
19/19 - 0s - 2ms/step - loss: 0.0347 - val_loss: 0.2519
Epoch 49/100
19/19 - 0s - 2ms/step - loss: 0.0379 - val_loss: 0.2776
Epoch 50/100
19/19 - 0s - 2ms/step - loss: 0.0313 - val_loss: 0.2161
Epoch 51/100
19/19 - 0s - 2ms/step - loss: 0.0335 - val_loss: 0.2504
Epoch 52/100
19/19 - 0s - 2ms/step - loss: 0.0334 - val_loss: 0.2641
Epoch 53/100
19/19 - 0s - 2ms/step - loss: 0.0351 - val_loss: 0.2134
Epoch 54/100
19/19 - 0s - 2ms/step - loss: 0.0298 - val_loss: 0.2297
Epoch 55/100
19/19 - 0s - 2ms/step - loss: 0.0327 - val_loss: 0.1788
Epoch 56/100
19/19 - 0s - 2ms/step - loss: 0.0268 - val_loss: 0.1944
Epoch 57/100
19/19 - 0s - 2ms/step - loss: 0.0276 - val_loss: 0.1974
```

```
Epoch 58/100
19/19 - 0s - 2ms/step - loss: 0.0238 - val_loss: 0.1727
Epoch 59/100
19/19 - 0s - 2ms/step - loss: 0.0252 - val_loss: 0.1306
Epoch 60/100
19/19 - 0s - 2ms/step - loss: 0.0304 - val_loss: 0.0915
Epoch 61/100
19/19 - 0s - 2ms/step - loss: 0.0288 - val_loss: 0.1655
Epoch 62/100
19/19 - 0s - 2ms/step - loss: 0.0276 - val_loss: 0.1175
Epoch 63/100
19/19 - 0s - 2ms/step - loss: 0.0222 - val_loss: 0.1412
Epoch 64/100
19/19 - 0s - 2ms/step - loss: 0.0239 - val_loss: 0.1187
Epoch 65/100
19/19 - 0s - 2ms/step - loss: 0.0244 - val_loss: 0.0937
Epoch 66/100
19/19 - 0s - 2ms/step - loss: 0.0270 - val_loss: 0.0645
Epoch 67/100
19/19 - 0s - 2ms/step - loss: 0.0224 - val_loss: 0.0757
Epoch 68/100
19/19 - 0s - 2ms/step - loss: 0.0253 - val_loss: 0.1132
Epoch 69/100
19/19 - 0s - 2ms/step - loss: 0.0236 - val_loss: 0.0833
Epoch 70/100
19/19 - 0s - 2ms/step - loss: 0.0187 - val_loss: 0.0771
Epoch 71/100
19/19 - 0s - 2ms/step - loss: 0.0229 - val_loss: 0.0470
Epoch 72/100
19/19 - 0s - 2ms/step - loss: 0.0196 - val_loss: 0.0516
Epoch 73/100
19/19 - 0s - 2ms/step - loss: 0.0207 - val_loss: 0.0409
Epoch 74/100
19/19 - 0s - 2ms/step - loss: 0.0146 - val_loss: 0.0824
Epoch 75/100
19/19 - 0s - 2ms/step - loss: 0.0192 - val_loss: 0.0697
Epoch 76/100
19/19 - 0s - 2ms/step - loss: 0.0174 - val_loss: 0.0390
Epoch 77/100
19/19 - 0s - 2ms/step - loss: 0.0151 - val_loss: 0.0957
Epoch 78/100
19/19 - 0s - 2ms/step - loss: 0.0222 - val_loss: 0.0437
Epoch 79/100
19/19 - 0s - 2ms/step - loss: 0.0157 - val_loss: 0.0273
Epoch 80/100
19/19 - 0s - 2ms/step - loss: 0.0151 - val_loss: 0.0298
Epoch 81/100
19/19 - 0s - 2ms/step - loss: 0.0166 - val_loss: 0.0298
```

```
Epoch 82/100
19/19 - 0s - 2ms/step - loss: 0.0154 - val_loss: 0.0322
Epoch 83/100
19/19 - 0s - 2ms/step - loss: 0.0126 - val_loss: 0.0320
Epoch 84/100
19/19 - 0s - 2ms/step - loss: 0.0194 - val_loss: 0.0322
Epoch 85/100
19/19 - 0s - 2ms/step - loss: 0.0162 - val_loss: 0.0504
Epoch 86/100
19/19 - 0s - 2ms/step - loss: 0.0171 - val_loss: 0.0371
Epoch 87/100
19/19 - 0s - 2ms/step - loss: 0.0109 - val_loss: 0.0390
Epoch 88/100
19/19 - 0s - 2ms/step - loss: 0.0157 - val_loss: 0.0409
Epoch 89/100
19/19 - 0s - 3ms/step - loss: 0.0203 - val_loss: 0.0723
Epoch 90/100
19/19 - 0s - 2ms/step - loss: 0.0170 - val_loss: 0.0717
Epoch 91/100
19/19 - 0s - 2ms/step - loss: 0.0225 - val_loss: 0.0417
Epoch 92/100
19/19 - 0s - 2ms/step - loss: 0.0154 - val_loss: 0.0433
Epoch 93/100
19/19 - 0s - 2ms/step - loss: 0.0140 - val_loss: 0.0423
Epoch 94/100
19/19 - 0s - 3ms/step - loss: 0.0124 - val_loss: 0.0436
Epoch 95/100
19/19 - 0s - 2ms/step - loss: 0.0114 - val_loss: 0.0453
Epoch 96/100
19/19 - 0s - 2ms/step - loss: 0.0139 - val_loss: 0.0481
Epoch 97/100
19/19 - 0s - 2ms/step - loss: 0.0112 - val_loss: 0.0659
Epoch 98/100
19/19 - 0s - 2ms/step - loss: 0.0148 - val_loss: 0.0503
Epoch 99/100
19/19 - 0s - 2ms/step - loss: 0.0183 - val_loss: 0.0462
Epoch 100/100
19/19 - 0s - 2ms/step - loss: 0.0102 - val_loss: 0.0449
```

[27]:
```
# evaluate on test set
yhat = model.predict(X)
error = mean_absolute_error(y, yhat)
print('MAE: %.5f' % error)
```

```
3/3             0s 10ms/step
MAE: 0.02008
```

[30]:
```
# Print the optimized output
print(yhat)
```

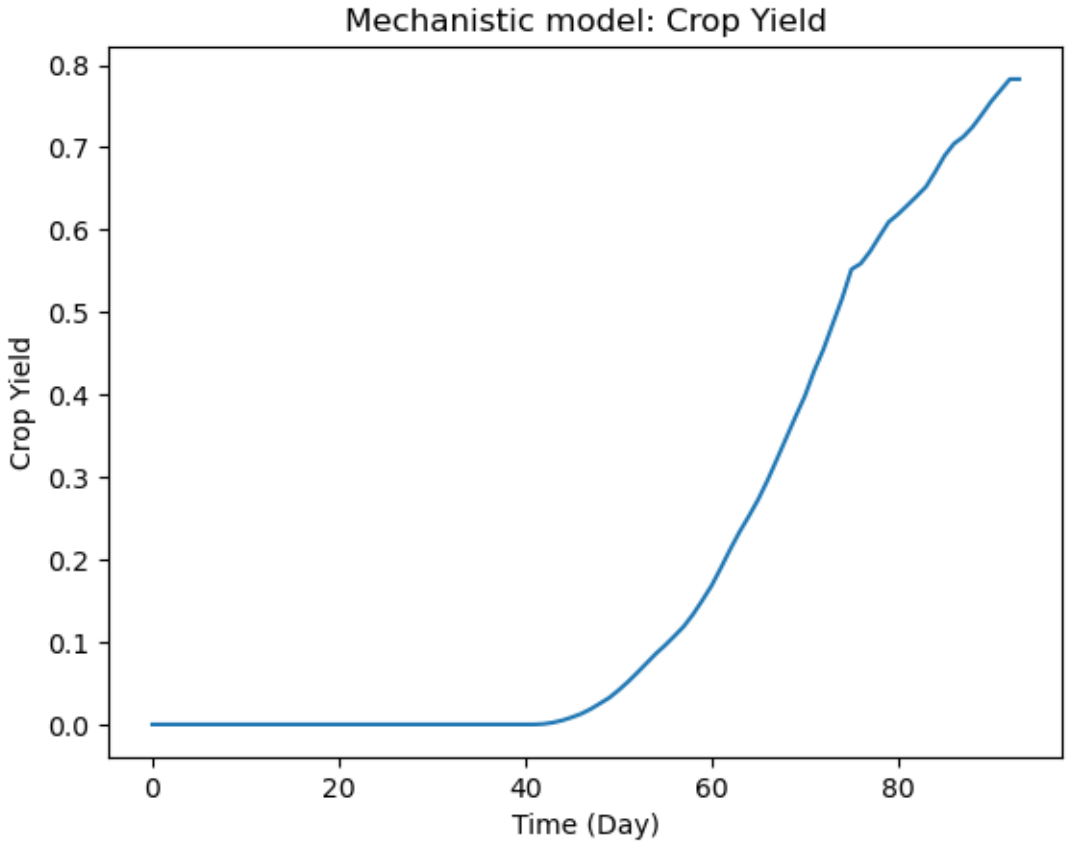
```
[[-1.11396117e-02]
 [-1.11093717e-02]
 [-1.51323555e-02]
 [-1.72573328e-02]
 [-1.41054327e-02]
 [-8.99567921e-03]
 [-8.51512607e-03]
 [-1.08254859e-02]
 [-1.15159499e-02]
 [-1.99161321e-02]
 [-1.67961828e-02]
 [-1.36569599e-02]
 [-1.46750668e-02]
 [-8.58695246e-03]
 [-1.17522562e-02]
 [-1.84617117e-02]
 [-1.39588723e-02]
 [-1.73861794e-02]
 [-1.55560402e-02]
 [-1.36332558e-02]
 [-1.76540539e-02]
 [-1.57579556e-02]
 [-7.24771339e-03]
 [-9.06979758e-03]
 [-1.42262997e-02]
 [-1.31365145e-02]
 [-8.12972616e-03]
 [-1.72025636e-02]
 [-1.91317275e-02]
 [-9.86721646e-03]
 [-1.53977601e-02]
 [-1.66625418e-02]
 [-2.00560689e-02]
 [-1.74076408e-02]
 [-8.97183362e-03]
 [-1.27481585e-02]
 [-1.82625465e-02]
 [-1.29628489e-02]
 [-6.79512043e-03]
 [-1.18311616e-02]
 [-8.57102685e-04]
 [-1.55781209e-03]
 [ 3.05404793e-03]
 [ 5.35357650e-03]
 [ 3.18841636e-03]
```

```
[ 1.89476553e-03]
[ 1.32568190e-02]
[ 1.69496723e-02]
[ 2.05585621e-02]
[ 4.43564951e-02]
[ 3.46327163e-02]
[ 4.09592725e-02]
[ 5.85997775e-02]
[ 6.58571273e-02]
[ 7.78955892e-02]
[ 8.80481079e-02]
[ 9.91897509e-02]
[ 1.34965733e-01]
[ 1.21457905e-01]
[ 1.37685940e-01]
[ 1.53169990e-01]
[ 1.60135746e-01]
[ 2.02462554e-01]
[ 2.23868668e-01]
[ 2.42980286e-01]
[ 2.67012715e-01]
[ 2.76860476e-01]
[ 2.89877236e-01]
[ 3.29719007e-01]
[ 3.46010417e-01]
[ 3.67073536e-01]
[ 4.09731776e-01]
[ 4.24061388e-01]
[ 4.01998371e-01]
[ 4.97112274e-01]
[ 4.71025914e-01]
[ 5.20090580e-01]
[ 5.34959495e-01]
[ 5.58252096e-01]
[ 5.95879674e-01]
[ 5.80807626e-01]
[ 5.93746424e-01]
[ 6.41727388e-01]
[ 6.48824155e-01]
[ 6.60293400e-01]
[ 7.07741082e-01]
[ 7.26662397e-01]
[ 7.41359651e-01]
[ 7.87304103e-01]
[ 7.94272006e-01]
[ 8.08243811e-01]
[ 8.90216172e-01]
[ 8.71594906e-01]
```

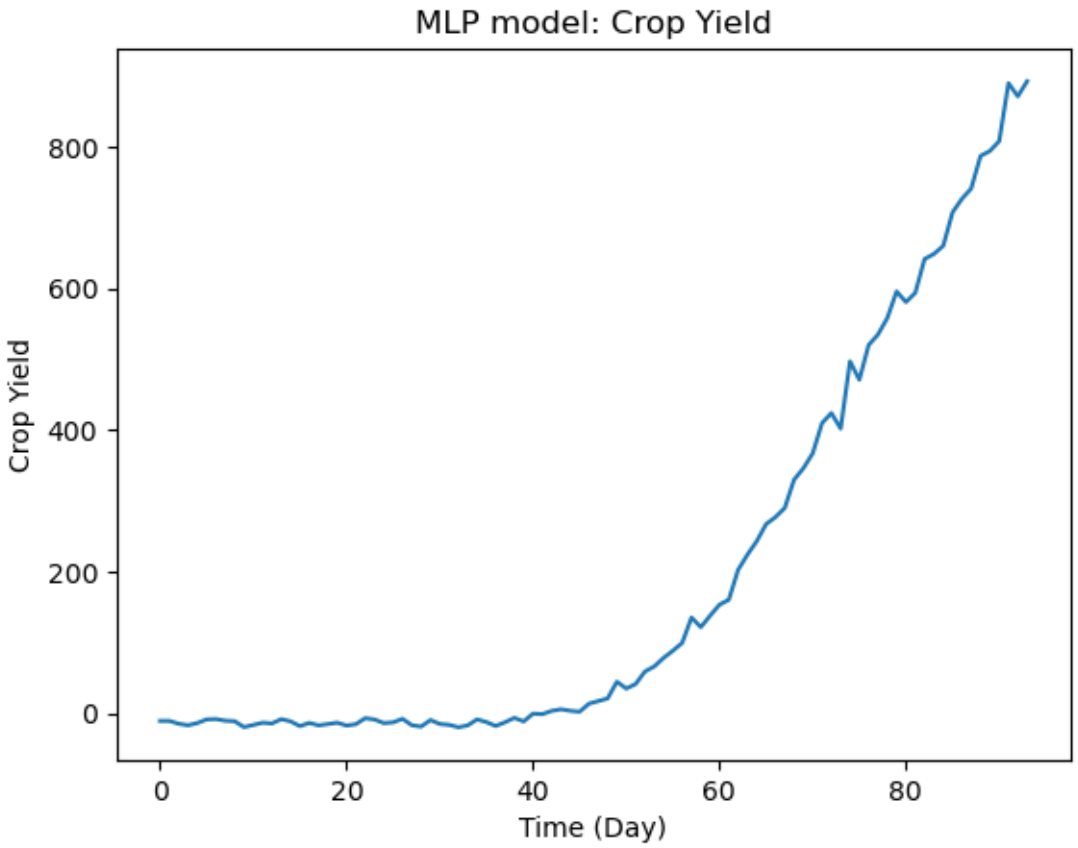
```
 [ 8.92973244e-01]]
```

[32]:
```
#print(y)
```

[34]:
```
plt.plot(y)
plt.xlabel('Time (Day)')
plt.ylabel('Crop Yield')
plt.title('Mechanistic model: Crop Yield')
plt.show()
```



[36]:
```
# plot the optimized output
plt.plot(yhat*1000)
plt.xlabel('Time (Day)')
plt.ylabel('Crop Yield')
plt.title('MLP model: Crop Yield')
plt.show()
```


MLP model: Crop Yield
Crop Yield
800
600
400
200
0
0
20
40
60
80
Time (Day)

[ ]: