In [1]:

```python
import xarray as xr
import pandas as pd
import numpy as np
from tensorflow import keras
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from tensorflow.keras.layers import *
from tensorflow.keras.optimizers import SGD, Adam
from tensorflow.keras.models import Sequential
from tensorflow.keras.utils import to_categorical
import tensorflow as tf
from tensorflow.keras import backend as K
```

In [2]:

```python
temp_dir = r"C:\Users\Muji\Documents\Columbia\Courses\Earth and Environmental Engineeri
evap_dir = r"C:\Users\Muji\Documents\Columbia\Courses\Earth and Environmental Engineeri
ssr_dir = r"C:\Users\Muji\Documents\Columbia\Courses\Earth and Environmental Engineerin
albedo_dir = r"C:\Users\Muji\Documents\Columbia\Courses\Earth and Environmental Enginee
co2_dir = r"C:\Users\Muji\Documents\Columbia\Courses\Earth and Environmental Engineerin


DS_temp = xr.open_dataset(temp_dir)
DS_evap = xr.open_dataset(evap_dir)
DS_ssr = xr.open_dataset(ssr_dir)
DS_albedo = xr.open_dataset(albedo_dir)
co2_array = np.array(pd.read_csv(co2_dir))
```

In [3]:

```python
time_range = []
for year in range(1973,2021):
    for month in range(1,13):
        for day in [1,15]:
            for hour in ['00:00:00','12:00:00']:
                time_range.append('{}-{}-{} {}'.format(year,month,day,hour))
time_len = len(time_range)
```

In [4]:

```python
#Computing the arctic average for each variable for every hour
temp_avg = []
albedo_avg = []
evap_avg = []
ssr_avg = []
co2_avg = []

for time in time_range:
    da_temp = DS_temp.sel(time = "{}".format(time))
    T_time = da_temp.mean().to_array()
    temp_avg.append(float(T_time)-273.15)

    da_albedo = DS_albedo.sel(time = "{}".format(time))
    albedo_time = da_albedo.mean().to_array()
    albedo_avg.append(float(albedo_time))

    da_evap = DS_evap.sel(time = "{}".format(time))
    evap_time = da_evap.mean().to_array()
    evap_avg.append(float(evap_time))
```

```
    da_ssr = DS_ssr.sel(time = "{}".format(time))
    ssr_time = da_ssr.mean().to_array()
    ssr_avg.append(float(ssr_time))

for i in range(time_len):
    co2_avg.append(float(co2_array[i]))
```

In [5]:

```
temp_avg = temp_avg/(np.std(temp_avg))
albedo_avg = albedo_avg/(np.std(albedo_avg))
evap_avg = evap_avg/(np.std(evap_avg))
ssr_avg = ssr_avg/(np.std(ssr_avg))
co2_std = np.std(co2_avg)
co2_avg = co2_avg/co2_std
```

In [6]:

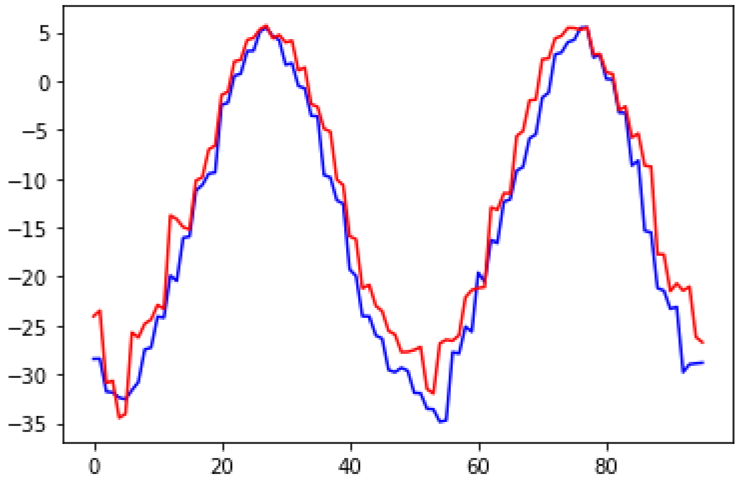
```
# Display training progress by printing a single dot for each completed epoch
class PrintDot(keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs):
        if epoch % 100 == 0: print('')
        print('.', end='')

# Function to plot how the model is doing during training
# Visualize the model's training progress using the stats stored in the history object.
# We want to use this data to determine how long to train before the model stops making
def plot_history_accuracy(history):
    plt.figure()
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.plot(history.epoch, np.array(history.history['accuracy']),
           label='Train accuracy')
    plt.plot(history.epoch, np.array(history.history['val_accuracy']),
           label = 'Val accuracy')
    plt.legend()

def plot_history_mae(history):
    plt.figure()
    plt.xlabel('Epoch')
    plt.ylabel('Mean Abs Error')
    plt.plot(history.epoch, np.array(history.history['mae']),
           label='Train Loss')
    plt.plot(history.epoch, np.array(history.history['val_mae']),
           label = 'Val loss')
    plt.legend()
```

In [13]:

```
plt.plot(temp_avg[:96],"b")
plt.plot(temp_avg[-96:],"r")
```

Out[13]:

```
[<matplotlib.lines.Line2D at 0x209527a7088>]
```



Date to date regression

In [105…

```
X = []
Y = []
for i in range(time_len):
    X.append(np.array([temp_avg[i],albedo_avg[i],evap_avg[i],ssr_avg[i],co2_avg[i]]))
    Y.append(np.array([temp_avg[i],albedo_avg[i],evap_avg[i],ssr_avg[i]]))

x = np.array(X[:-1])
y = np.array(Y[1:])

X_train,X_test,Y_train,Y_test = train_test_split(x,y,test_size = 0.2)
```

In [106…

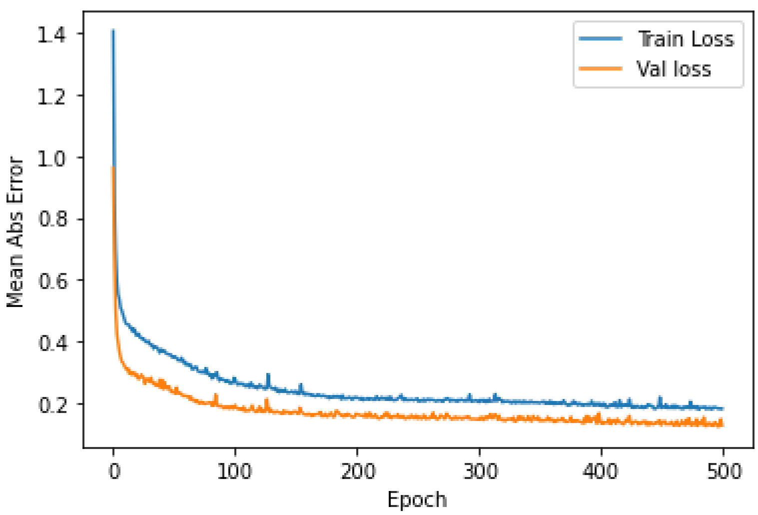
```
model_nn_single = keras.Sequential([
    keras.layers.Dense(25, input_shape=((5,)), activation=tf.nn.relu),
    keras.layers.Dense(25, activation=tf.nn.relu),
    keras.layers.Dropout(0.1),
    keras.layers.Dense(4)])

model_nn_single.compile(loss='mse',optimizer='adam',metrics=['mae'])

model_nn_single.summary()
```

```
Model: "sequential_24"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_72 (Dense)             (None, 25)                150       
_________________________________________________________________
dense_73 (Dense)             (None, 25)                650       
_________________________________________________________________
dropout_5 (Dropout)          (None, 25)                0         
_________________________________________________________________
dense_74 (Dense)             (None, 4)                 104       
=================================================================
Total params: 904
Trainable params: 904
Non-trainable params: 0
_________________________________________________________________
```

In [107…

```
# If you train too long, you are prone to over-fitting
# this prevents the model from generalizing to data it has never seen before
# early stopping is one way to go about this
# The patience parameter is the amount of epochs to check for improvement
early_stop = keras.callbacks.EarlyStopping(monitor='val_loss', patience=50)

K.set_value(model_nn_single.optimizer.learning_rate, 0.001)

# Store training stats
history = model_nn_single.fit(X_train, Y_train, epochs=500,
                    validation_split=0.2, verbose=0,
                    callbacks=[early_stop, PrintDot()])

plot_history_mae(history)

[loss, mae] = model_nn_single.evaluate(X_test, Y_test, verbose=0)
print("Testing set Mean Abs Error: {}".format(mae ))
```

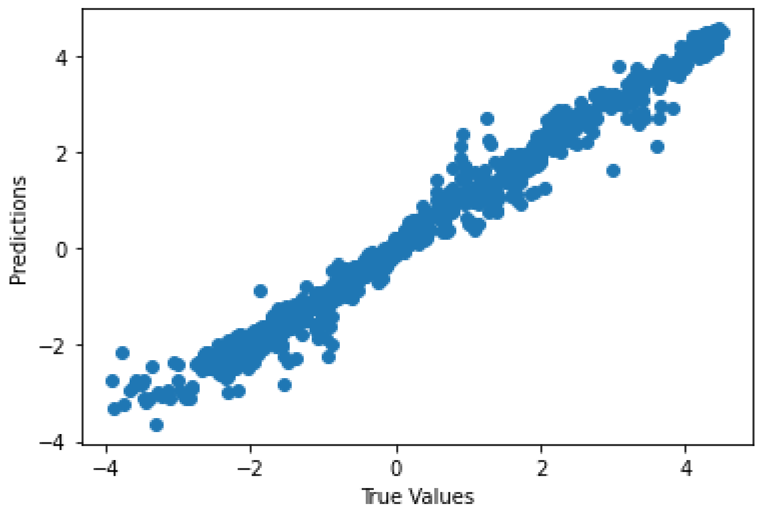
```
....................................................................................................

....................................................................................................

....................................................................................................

....................................................................................................

....................................................................................................
Testing set Mean Abs Error: 0.13784269988536835
```



In [108…

```
test_predictions = model_nn_single.predict(X_test).flatten()
test_labels = Y_test.flatten()

plt.scatter(Y_test, test_predictions)
plt.xlabel('True Values')
plt.ylabel('Predictions')
plt.plot()
```

Out[108… []



Multiple dates regression

In [24]:

```
length = 8
X = []
Y = []
for i in range(time_len-length):
    features = []
    for j in range(length):
        features.append(temp_avg[i+j])
        features.append(albedo_avg[i+j])
        features.append(evap_avg[i+j])
        features.append(ssr_avg[i+j])
        features.append(co2_avg[i+j])

    X.append(np.array(features))
    Y.append(np.array([temp_avg[i+length],albedo_avg[i+length],evap_avg[i+length],ssr_a

x = np.array(X)
y = np.array(Y)

X_train,X_test,Y_train,Y_test = train_test_split(x,y,test_size = 0.2)
```

In [25]:

```
model_nn_multiple = keras.Sequential([
    keras.layers.Dense(75, input_shape=((5*length,)), activation=tf.nn.relu),
    keras.layers.Dense(75, activation=tf.nn.relu),
    #keras.layers.Dropout(0.1),
    keras.layers.Dense(4)])

model_nn_multiple.compile(loss='mse',optimizer='adam',metrics=['mae'])

model_nn_multiple.summary()
```

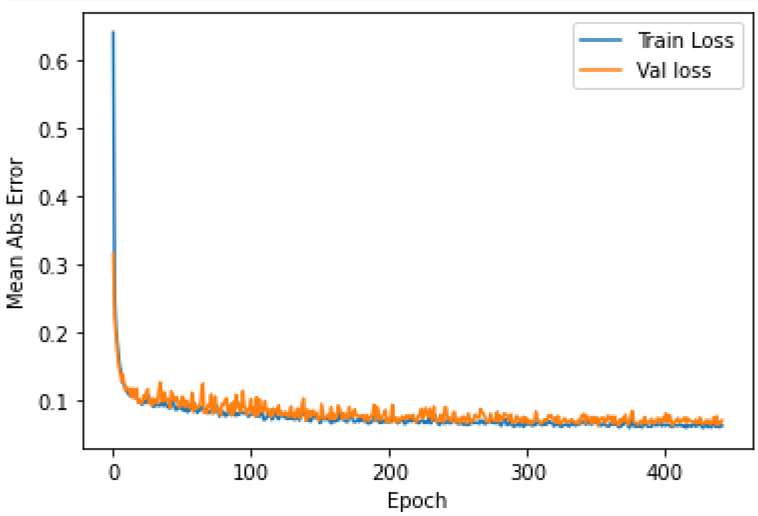
```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_6 (Dense)              (None, 75)                3075
_________________________________________________________________
dense_7 (Dense)              (None, 75)                5700
```

```
dense_8 (Dense)              (None, 4)                 304
=================================================================
Total params: 9,079
Trainable params: 9,079
Non-trainable params: 0

_________________________________________________________________
```

In [26]:

```
# If you train too long, you are prone to over-fitting
# this prevents the model from generalizing to data it has never seen before
# early stopping is one way to go about this
# The patience parameter is the amount of epochs to check for improvement
early_stop = keras.callbacks.EarlyStopping(monitor='val_loss', patience=75)

K.set_value(model_nn_multiple.optimizer.learning_rate, 0.001)

# Store training stats
history = model_nn_multiple.fit(X_train, Y_train, epochs=1000,
                    validation_split=0.2, verbose=0,
                    callbacks=[early_stop, PrintDot()])

plot_history_mae(history)

[loss, mae] = model_nn_multiple.evaluate(X_test, Y_test, verbose=0)
print("Testing set Mean Abs Error: {}".format(mae ))
```

```
....................................................................................................

....................................................................................................

....................................................................................................

....................................................................................................

..........................................Testing set Mean Abs Error: 0.0633960366249084
5
```

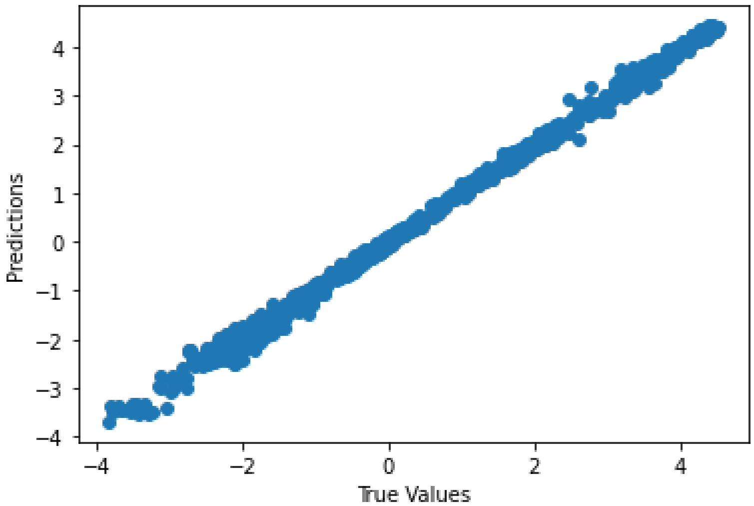


In [27]:

```
test_predictions = model_nn_multiple.predict(X_test).flatten()
test_labels = Y_test.flatten()

plt.scatter(Y_test, test_predictions)
```

```
plt.xlabel('True Values')
plt.ylabel('Predictions')
plt.plot()
```

Out[27]: []



LSTM

In [24]:

```
X = []
Y = []
length = 48
for i in range(0,time_len,length):
    batch = []
    for j in range(length-1):
        batch.append(np.array([temp_avg[i+j],albedo_avg[i+j],evap_avg[i+j],ssr_avg[
    X.append(batch)
    Y.append(np.array([temp_avg[i+length-1],albedo_avg[i+length-1],evap_avg[i+length-1]
    #Y.append(np.array([temp_avg[i+length-1]]))

x = np.array(X)
y = np.array(Y)
X_train,X_test,Y_train,Y_test = train_test_split(x,y,test_size = 0.2)
```

In [25]:

```
model_LSTM = keras.Sequential()
model_LSTM.add(LSTM(150,input_shape=(length-1,5),activation = tf.nn.tanh))
model_LSTM.add(Dense(120))
#model_LSTM.add(Dropout(0.1))
model_LSTM.add(Dense(4))

model_LSTM.compile(loss='mse',optimizer='adam',metrics=['mae'])

model_LSTM.summary()
```

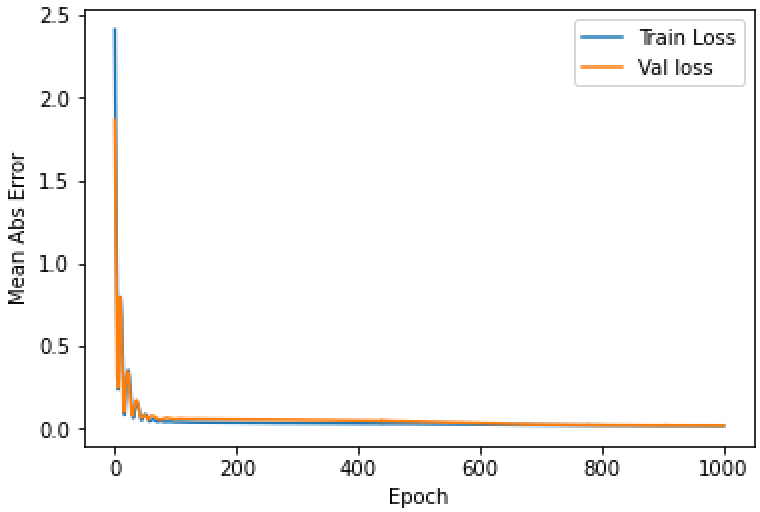
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 150)               93600
_________________________________________________________________
```

```
dense_2 (Dense)              (None, 120)               18120
_________________________________________________________________
dense_3 (Dense)              (None, 4)                 484
=================================================================
Total params: 112,204
Trainable params: 112,204
Non-trainable params: 0
_________________________________________________________________
```

In [26]:

```
# If you train too long, you are prone to over-fitting
# this prevents the model from generalizing to data it has never seen before
# early stopping is one way to go about this
# The patience parameter is the amount of epochs to check for improvement
early_stop = keras.callbacks.EarlyStopping(monitor='val_loss', patience=30)

K.set_value(model_LSTM.optimizer.learning_rate, 0.001)

# Store training stats
history = model_LSTM.fit(X_train, Y_train, epochs=1000,
                    validation_split=0.2, verbose=0,
                    callbacks=[early_stop, PrintDot()])

plot_history_mae(history)

[loss, mae] = model_LSTM.evaluate(X_test, Y_test, verbose=0)
print("Testing set Mean Abs Error: {}".format(mae ))
```

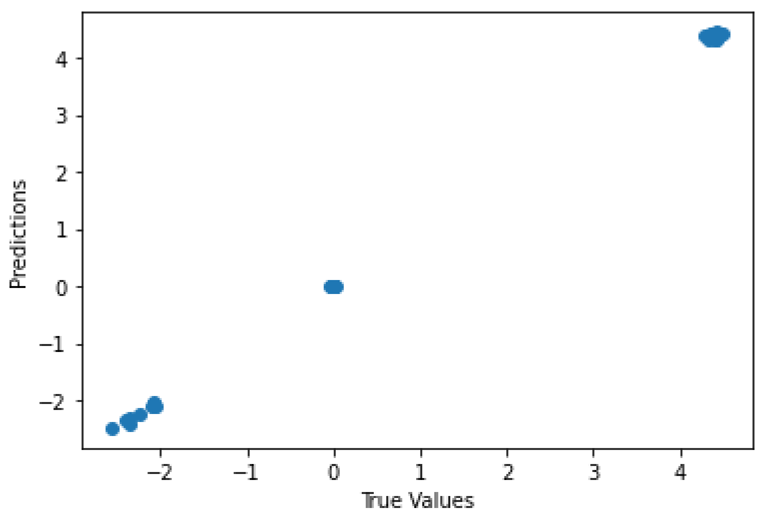
```
.......................................................................................

.......................................................................................

.......................................................................................

.......................................................................................

.......................................................................................

.......................................................................................

.......................................................................................

.......................................................................................

.......................................................................................

.......................................................................................
Testing set Mean Abs Error: 0.024045681580901146
```



In [27]:

```
test_predictions = model_LSTM.predict(X_test).flatten()
test_labels = Y_test.flatten()

plt.scatter(test_labels, test_predictions)
plt.xlabel('True Values')
plt.ylabel('Predictions')
plt.plot()
```

Out[27]: []



Predictions

In [28]:

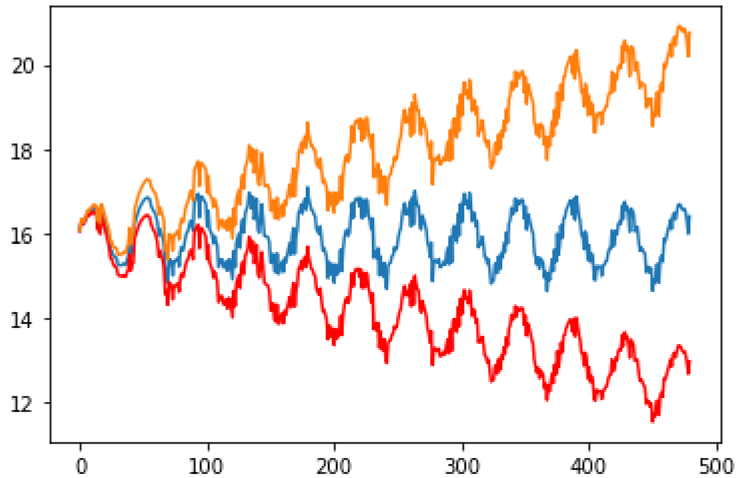
```
normal_array = np.array(pd.read_csv(r"C:\Users\Muji\Documents\Columbia\Courses\Earth an
exp_decreasing_array = np.array(pd.read_csv(r"C:\Users\Muji\Documents\Columbia\Courses\
exp_increasing_array = np.array(pd.read_csv(r"C:\Users\Muji\Documents\Columbia\Courses\

normal = []
exp_decreasing = []
exp_increasing = []
for i in range(len(normal_array)):
    normal.append(float(normal_array[i]))
    exp_decreasing.append(float(exp_decreasing_array[i]))
```

```
    exp_increasing.append(float(exp_increasing_array[i]))

normal = (normal/co2_std)[:]
exp_decreasing = (exp_decreasing/co2_std)[:]
exp_increasing = (exp_increasing/co2_std)[:]
```

In [29]:

```
plt.plot(normal)
plt.plot(exp_decreasing,"r")
plt.plot(exp_increasing)
```

Out[29]: [<matplotlib.lines.Line2D at 0x17b0c2d4a48>]



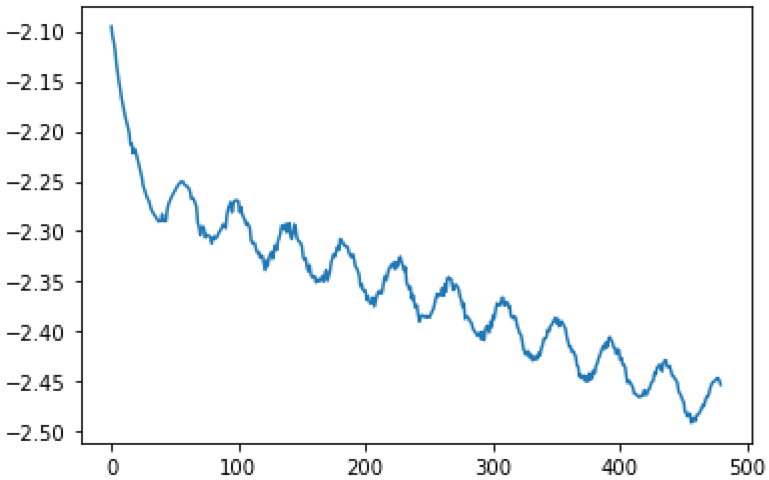
In [30]:

```
co2_scenario = exp_decreasing

current_values = []
for i in range(length-1,0,-1):
    current_values.append(np.array([temp_avg[-i],albedo_avg[-i],evap_avg[-i],ssr_avg[-i

prediction = []
for i in range(len(co2_scenario)):
    cv = np.array([current_values[:]])
    pred = list(model_LSTM.predict(cv)[0])
    prediction.append(pred)
    pred.append(co2_scenario[i])
    current_values.append(pred)
    current_values.pop(0)
```

In [31]:

```
prediction = np.array(prediction)
temp_pred = (prediction[:,0])
albedo_pred = (prediction[:,1])
evap_pred = (prediction[:,2])
ssr_pred = (prediction[:,3])

plt.plot(temp_pred)
#plt.plot(albedo_pred)
#plt.plot(evap_pred)
#plt.plot(ssr_pred)
#plt.plot(co2_scenario)
```

Out[31]: [<matplotlib.lines.Line2D at 0x17b0b1df288>]



In [ ]: