LSTM_Regression_Assignment

February 3, 2024

1 Long Short Term Memory Networks for IoT Prediction

RNNs and LSTM models are very popular neural network architectures when working with sequential data, since they both carry some "memory" of previous inputs when predicting the next output. In this assignment we will continue to work with the Household Electricity Consumption dataset and use an LSTM model to predict the Global Active Power (GAP) from a sequence of previous GAP readings. You will build one model following the directions in this notebook closely, then you will be asked to make changes to that original model and analyze the effects that they had on the model performance. You will also be asked to compare the performance of your LSTM model to the linear regression predictor that you built in last week's assignment.

1.1 General Assignment Instructions

These instructions are included in every assignment, to remind you of the coding standards for the class. Feel free to delete this cell after reading it.

One sign of mature code is conforming to a style guide. We recommend the Google Python Style Guide. If you use a different style guide, please include a cell with a link.

Your code should be relatively easy-to-read, sensibly commented, and clean. Writing code is a messy process, so please be sure to edit your final submission. Remove any cells that are not needed or parts of cells that contain unnecessary code. Remove inessential `import` statements and make sure that all such statements are moved into the designated cell.

When you save your notebook as a pdf, make sure that all cell output is visible (even error messages) as this will aid your instructor in grading your work.

Make use of non-code cells for written commentary. These cells should be grammatical and clearly written. In some of these cells you will have questions to answer. The questions will be marked by a "Q:" and will have a corresponding "A:" spot for you. *Make sure to answer every question marked with a `Q:` for full credit.*

[37]:
```
import keras
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import os

# Setting seed for reproducibility
np.random.seed(1234)
```

```
PYTHONHASHSEED = 0

from sklearn import preprocessing
from sklearn.metrics import confusion_matrix, recall_score, precision_score
from sklearn.model_selection import train_test_split
from keras.models import Sequential,load_model
from keras.layers import Dense, Dropout, LSTM
from keras.utils import pad_sequences
from keras.layers import LSTM, Dense, Dropout, Activation
from sklearn.model_selection import train_test_split
```

[ ]:
```
#use this cell to import additional libraries or define helper functions
```

1.2 Load and prepare your data

We'll once again be using the cleaned household electricity consumption data from the previous two assignments. I recommend saving your dataset by running df.to_csv("filename") at the end of assignment 2 so that you don't have to re-do your cleaning steps. If you are not confident in your own cleaning steps, you may ask your instructor for a cleaned version of the data. You will not be graded on the cleaning steps in this assignment, but some functions may not work if you use the raw data.

Unlike when using Linear Regression to make our predictions for Global Active Power (GAP), LSTM requires that we have a pre-trained model when our predictive software is shipped (the ability to iterate on the model after it's put into production is another question for another day). Thus, we will train the model on a segment of our data and then measure its performance on simulated streaming data another segment of the data. Our dataset is very large, so for speed's sake, we will limit ourselves to 1% of the entire dataset.

TODO: Import your data, select the a random 1% of the dataset, and then split it 80/20 into training and validation sets (the test split will come from the training data as part of the tensorflow LSTM model call). HINT: Think carefully about how you do your train/validation split–does it make sense to randomize the data?

[3]:
```
#Load your data into a pandas dataframe here
df = pd.read_csv('household_power_clean.csv')

df.head()
```

[3]:

	Unnamed: 0	Date	Time	Global_active_power	\
0	0	2006-12-16	17:24:00	4.216	
1	1	2006-12-16	17:25:00	5.360	
2	2	2006-12-16	17:26:00	5.374	
3	3	2006-12-16	17:27:00	5.388	
4	4	2006-12-16	17:28:00	3.666	

	Global_reactive_power	Voltage	Global_intensity	Sub_metering_1	\
0	0.418	234.84	18.4	0.0	

```
1                  0.436   233.63              23.0             0.0
2                  0.498   233.29              23.0             0.0
3                  0.502   233.74              23.0             0.0
4                  0.528   235.68              15.8             0.0

   Sub_metering_2  Sub_metering_3             Datetime  gap_monthly  \
0             1.0            17.0  2006-12-16 17:24:00          NaN
1             1.0            16.0  2006-12-16 17:25:00          NaN
2             2.0            17.0  2006-12-16 17:26:00          NaN
3             1.0            17.0  2006-12-16 17:27:00          NaN
4             1.0            17.0  2006-12-16 17:28:00          NaN

   grp_monthly  v_monthly  gi_monthly
0          NaN        NaN         NaN
1          NaN        NaN         NaN
2          NaN        NaN         NaN
3          NaN        NaN         NaN
4          NaN        NaN         NaN
```

[7]:
```python
#create your training and validation sets here

#assign size for data subset
# subset of 1000 samples
subset_size = 1000
df_subset = df.sample(n=subset_size, random_state=42)


#take random data subset
df_subset = df.sample(n=subset_size, random_state=42)


#split data subset 80/20 for train/validation
train_df, val_df = train_test_split(df_subset, test_size=0.2, random_state=42)


print(f"Training set: {train_df}")
print(f"Validation set: {val_df}")
```

```
Training set:          Unnamed: 0        Date      Time  Global_active_power  \
571901       575837  2008-01-20  14:41:00                1.470
1098247     1102318  2009-01-20  05:22:00                0.400
2019949     2045928  2010-11-06  12:12:00                1.996
813599       817542  2008-07-06  11:06:00                0.298
1333538     1340986  2009-07-04  23:10:00                1.132
…             …         …        …                    …
247737       251471  2007-06-09  08:35:00                1.474
271268       275042  2007-06-25  17:26:00                0.322
919308       923255  2008-09-17  20:59:00                2.594
```

```
1807576     1821088  2010-06-03  08:52:00                1.776
517827       521761  2007-12-14  01:25:00                0.314

         Global_reactive_power  Voltage  Global_intensity  Sub_metering_1  \
571901                   0.000   241.01               6.2             0.0
1098247                  0.192   246.29               1.8             0.0
2019949                  0.340   234.78               8.6             0.0
813599                   0.000   240.13               1.4             0.0
1333538                  0.248   242.38               4.8             0.0
…                        …       …                    …               …
247737                   0.206   236.77               6.2             0.0
271268                   0.244   242.20               1.6             0.0
919308                   0.276   239.60              11.2             0.0
1807576                  0.174   238.49               7.4             1.0
517827                   0.086   245.04               1.4             0.0

         Sub_metering_2  Sub_metering_3             Datetime  gap_monthly  \
571901              0.0            18.0  2008-01-20 14:41:00     1.787800
1098247             0.0             0.0  2009-01-20 05:22:00     0.470200
2019949             1.0            17.0  2010-11-06 12:12:00     2.060000
813599              0.0             1.0  2008-07-06 11:06:00     0.820467
1333538             0.0             1.0  2009-07-04 23:10:00     1.844067
…                   …               …                    …        …
247737              1.0            17.0  2007-06-09 08:35:00     0.686000
271268              1.0             0.0  2007-06-25 17:26:00     0.391200
919308              0.0            18.0  2008-09-17 20:59:00     3.126067
1807576             1.0            18.0  2010-06-03 08:52:00     2.535800
517827              0.0             0.0  2007-12-14 01:25:00     0.287867

         grp_monthly   v_monthly  gi_monthly
571901      0.167867  239.885000    7.466667
1098247     0.081333  245.057667    2.000000
2019949     0.291733  235.103000    8.813333
813599      0.148200  240.582667    3.500000
1333538     0.155600  241.634333    7.753333
…              …         …          …
247737      0.069000  237.978667    2.920000
271268      0.257133  243.393333    1.960000
919308      0.226133  238.235667   13.246667
1807576     0.137667  238.067667   10.660000
517827      0.064000  244.434000    1.246667

[800 rows x 15 columns]
Validation set:          Unnamed: 0        Date      Time  Global_active_power
\
250177       253949  2007-06-11  01:53:00                0.192
1080461     1084531  2009-01-07  20:55:00                2.398
51980         51985  2007-01-21  19:49:00                3.728
```

```
665753       669693  2008-03-25  18:57:00                1.042
1822517     1836030  2010-06-13  17:54:00                3.790
…              …         …        …                     …
1030972     1034971  2008-12-04  10:55:00                1.340
1086715     1090785  2009-01-12  05:09:00                0.364
3592           3592  2006-12-19  05:16:00                0.242
1663743     1675226  2010-02-22  01:50:00                0.252
1769000     1782511  2010-05-07  13:55:00                1.462

         Global_reactive_power  Voltage  Global_intensity  Sub_metering_1  \
250177                   0.070   240.42               0.8             0.0
1080461                  0.056   242.72               9.8             0.0
51980                    0.182   232.29              16.0             0.0
665753                   0.356   238.85               4.6             0.0
1822517                  0.154   238.87              15.8             0.0
…                            …       …                 …               …
1030972                  0.064   240.81               5.6             0.0
1086715                  0.198   249.80               1.6             0.0
3592                     0.000   244.06               1.0             0.0
1663743                  0.048   242.13               1.0             0.0
1769000                  0.892   237.70               7.6             0.0

         Sub_metering_2  Sub_metering_3             Datetime  gap_monthly  \
250177              0.0             0.0  2007-06-11 01:53:00     0.181600
1080461             2.0            18.0  2009-01-07 20:55:00     2.993133
51980              18.0            17.0  2007-01-21 19:49:00     5.918400
665753              1.0             0.0  2008-03-25 18:57:00     0.634000
1822517            38.0            18.0  2010-06-13 17:54:00     1.583200
…                     …               …                    …           …
1030972             0.0            18.0  2008-12-04 10:55:00     2.388867
1086715             1.0             0.0  2009-01-12 05:09:00     0.481467
3592                0.0             0.0  2006-12-19 05:16:00     0.317933
1663743             0.0             1.0  2010-02-22 01:50:00     0.340867
1769000             0.0             0.0  2010-05-07 13:55:00     0.522267

         grp_monthly   v_monthly  gi_monthly
250177      0.056933  242.017667    0.786667
1080461     0.185933  242.346000   12.366667
51980       0.239400  227.962667   26.053333
665753      0.188200  242.050667    2.860000
1822517     0.114400  241.815000    6.753333
…              …         …          …
1030972     0.068733  241.013333    9.913333
1086715     0.182333  249.537333    2.060000
3592        0.082000  243.849333    1.446667
1663743     0.095400  241.770333    1.413333
1769000     0.124467  238.077000    2.360000
```

```
[200 rows x 15 columns]
```

[8]:
```
#reset the indices for cleanliness
train_df = train_df.reset_index()
val_df = val_df.reset_index()
```

Next we need to create our input and output sequences. In the lab session this week, we used an LSTM model to make a binary prediction, but LSTM models are very flexible in what they can output: we can also use them to predict a single real-numbered output (we can even use them to predict a sequence of outputs). Here we will train a model to predict a single real-numbered output such that we can compare our model directly to the linear regression model from last week.

TODO: Create a nested list structure for the training data, with a sequence of GAP measurements as the input and the GAP measurement at your predictive horizon as your expected output

[32]:
```
seq_arrays = []
seq_labs = []
```

[33]:
```
# we'll start out with a 30 minute input sequence and a 5 minute predictive␣
 ↪horizon
# we don't need to work in seconds this time, since we'll just use the indices␣
 ↪instead of a unix timestamp
seq_length = 30
ph = 5

feat_cols = ['Global_active_power']

#create list of sequence length GAP readings
for i in range(len(train_df) - seq_length - ph + 1):
    seq_arrays.append(train_df[feat_cols].iloc[i:i+seq_length].values)
    seq_labs.append(train_df[feat_cols].iloc[i+seq_length+ph-1].values)

# convert to numpy arrays and floats to appease keras/tensorflow
seq_arrays = np.array(seq_arrays).astype('float32')
seq_labs = np.array(seq_labs).reshape(-1).astype('float32')  # ensure labels␣
 ↪are a flat array
```

[34]:
```
assert(seq_arrays.shape == (len(train_df)-seq_length-ph+1, seq_length,␣
 ↪len(feat_cols)))
assert(seq_labs.shape == (len(train_df)-seq_length-ph+1,))
```

[35]:
```
seq_arrays.shape
```

[35]:
```
(766, 30, 1)
```

Q: What is the function of the assert statements in the above cell? Why do we use assertions in our code?

A: the assert statements verify that the code is formatted correctly, the assert statement passes if the statement evaluates to true. To help clarify, see the print statement below

1.3 Model Training

We will begin with a model architecture very similar to the model we built in the lab session. We will have two LSTM layers, with 5 and 3 hidden units respectively, and we will apply dropout after each LSTM layer. However, we will use a LINEAR final layer and MSE for our loss function, since our output is continuous instead of binary.

TODO: Fill in all values marked with a ?? in the cell below

[36]:
```
print(seq_arrays.shape == (len(train_df)-seq_length-ph+1, seq_length,␣
 ↪len(feat_cols)))
print(seq_labs.shape == (len(train_df)-seq_length-ph+1,))
```

```
True
True
```

[38]:
```
# define path to save model
model_path = 'LSTM_model1.h5'

# build the network
nb_features = seq_arrays.shape[2]  # Should be the number of features (1 if␣
 ↪'Global_active_power' is the only feature)
nb_out = 1  # Single real-numbered output


model = Sequential()

# add first LSTM layer with 5 hidden units
model.add(LSTM(
         input_shape=(seq_length, nb_features),
         units=5,
         return_sequences=True))
model.add(Dropout(0.2))

# add second LSTM layer with 3 hidden units
model.add(LSTM(
          units=3,
          return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(units=nb_out))
model.add(Activation('linear'))
optimizer = keras.optimizers.Adam(learning_rate = 0.01)
model.compile(loss='mean_squared_error', optimizer=optimizer,metrics=['mse'])

print(model.summary())
```

```
# fit the network
history = model.fit(seq_arrays, seq_labs, epochs=100, batch_size=500,␣
 ↪validation_split=0.05, verbose=2,
          callbacks = [keras.callbacks.EarlyStopping(monitor='val_loss',␣
 ↪min_delta=0, patience=10, verbose=0, mode='min'),
                       keras.callbacks.
 ↪ModelCheckpoint(model_path,monitor='val_loss', save_best_only=True,␣
 ↪mode='min', verbose=0)]
          )

# list all data in history
print(history.history.keys())
```

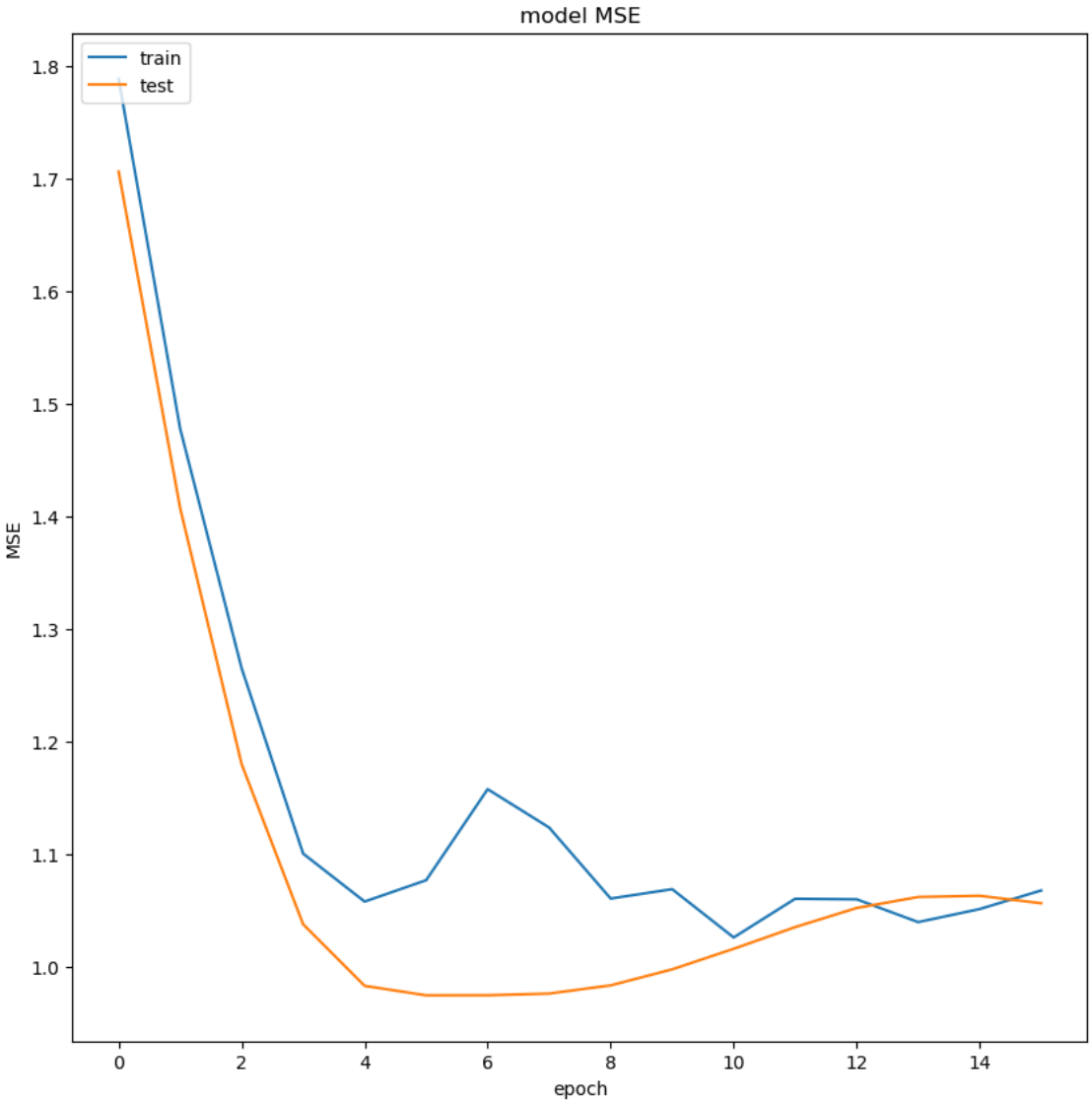
```
WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.Adam` runs
slowly on M1/M2 Macs, please use the legacy Keras optimizer instead, located at
`tf.keras.optimizers.legacy.Adam`.
```

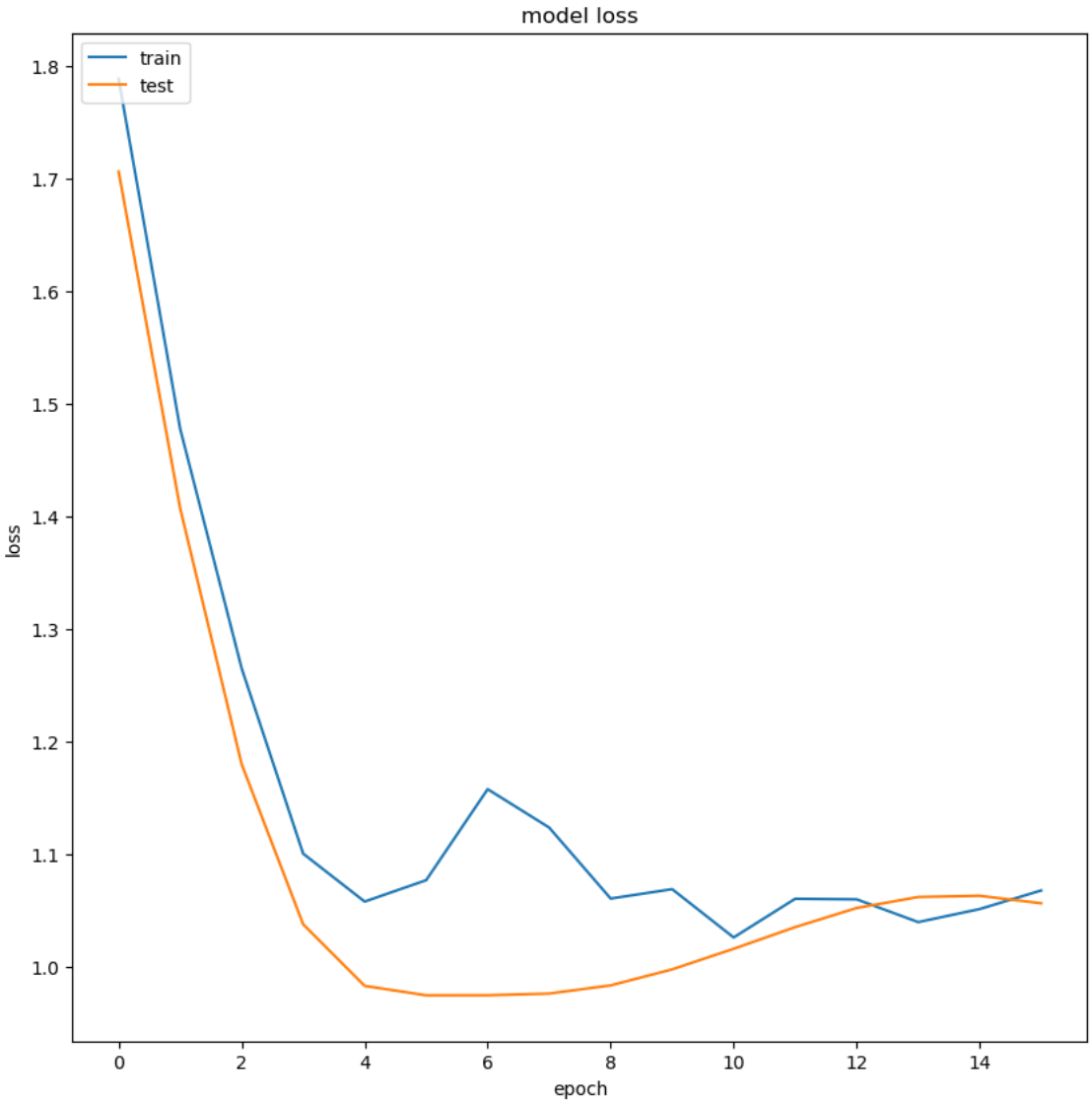
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 30, 5)             140

 dropout (Dropout)           (None, 30, 5)             0

 lstm_1 (LSTM)               (None, 3)                 108

 dropout_1 (Dropout)         (None, 3)                 0

 dense (Dense)               (None, 1)                 4

 activation (Activation)     (None, 1)                 0

=================================================================
Total params: 252 (1008.00 Byte)
Trainable params: 252 (1008.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
Epoch 1/100
2/2 - 2s - loss: 1.7885 - mse: 1.7885 - val_loss: 1.7062 - val_mse: 1.7062 -
2s/epoch - 788ms/step
Epoch 2/100
2/2 - 0s - loss: 1.4779 - mse: 1.4779 - val_loss: 1.4074 - val_mse: 1.4074 -
47ms/epoch - 23ms/step
Epoch 3/100
2/2 - 0s - loss: 1.2651 - mse: 1.2651 - val_loss: 1.1800 - val_mse: 1.1800 -
44ms/epoch - 22ms/step
```

```
Epoch 4/100
2/2 - 0s - loss: 1.1006 - mse: 1.1006 - val_loss: 1.0378 - val_mse: 1.0378 -
41ms/epoch - 20ms/step
Epoch 5/100
2/2 - 0s - loss: 1.0580 - mse: 1.0580 - val_loss: 0.9831 - val_mse: 0.9831 -
38ms/epoch - 19ms/step
Epoch 6/100

/Users/williamkencel/anaconda3/lib/python3.11/site-
packages/keras/src/engine/training.py:3103: UserWarning: You are saving your
model as an HDF5 file via `model.save()`. This file format is considered legacy.
We recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')`.
  saving_api.save_model(

2/2 - 0s - loss: 1.0769 - mse: 1.0769 - val_loss: 0.9747 - val_mse: 0.9747 -
38ms/epoch - 19ms/step
Epoch 7/100
2/2 - 0s - loss: 1.1578 - mse: 1.1578 - val_loss: 0.9748 - val_mse: 0.9748 -
32ms/epoch - 16ms/step
Epoch 8/100
2/2 - 0s - loss: 1.1237 - mse: 1.1237 - val_loss: 0.9763 - val_mse: 0.9763 -
31ms/epoch - 15ms/step
Epoch 9/100
2/2 - 0s - loss: 1.0607 - mse: 1.0607 - val_loss: 0.9836 - val_mse: 0.9836 -
29ms/epoch - 14ms/step
Epoch 10/100
2/2 - 0s - loss: 1.0690 - mse: 1.0690 - val_loss: 0.9978 - val_mse: 0.9978 -
30ms/epoch - 15ms/step
Epoch 11/100
2/2 - 0s - loss: 1.0261 - mse: 1.0261 - val_loss: 1.0161 - val_mse: 1.0161 -
33ms/epoch - 17ms/step
Epoch 12/100
2/2 - 0s - loss: 1.0605 - mse: 1.0605 - val_loss: 1.0353 - val_mse: 1.0353 -
31ms/epoch - 15ms/step
Epoch 13/100
2/2 - 0s - loss: 1.0600 - mse: 1.0600 - val_loss: 1.0523 - val_mse: 1.0523 -
29ms/epoch - 14ms/step
Epoch 14/100
2/2 - 0s - loss: 1.0398 - mse: 1.0398 - val_loss: 1.0621 - val_mse: 1.0621 -
31ms/epoch - 16ms/step
Epoch 15/100
2/2 - 0s - loss: 1.0513 - mse: 1.0513 - val_loss: 1.0631 - val_mse: 1.0631 -
30ms/epoch - 15ms/step
Epoch 16/100
2/2 - 0s - loss: 1.0679 - mse: 1.0679 - val_loss: 1.0565 - val_mse: 1.0565 -
32ms/epoch - 16ms/step
dict_keys(['loss', 'mse', 'val_loss', 'val_mse'])
```

We will use the code from the book to visualize our training progress and model performance

[39]:
```
# summarize history for MSE
fig_acc = plt.figure(figsize=(10, 10))
plt.plot(history.history['mse'])
plt.plot(history.history['val_mse'])
plt.title('model MSE')
plt.ylabel('MSE')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
fig_acc.savefig("LSTM_mse1.png")

# summarize history for Loss
fig_acc = plt.figure(figsize=(10, 10))
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
fig_acc.savefig("LSTM_loss1.png")
```


model MSE
train
test
MSE
epoch
1.8
1.7
1.6
1.5
1.4
1.3
1.2
1.1
1.0
0
2
4
6
8
10
12
14



1.4 Validating our model

Now we need to create our simulated streaming validation set to test our model "in production". With our linear regression models, we were able to begin making predictions with only two data-points, but the LSTM model requires an input sequence of *seq_length* to make a prediction. We can get around this limitation by "padding" our inputs when they are too short.

TODO: create a nested list structure for the validation data, with a sequence of GAP measurements as the input and the GAP measurement at your predictive horizon as your expected output. Begin your predictions after only two GAP measurements are available, and check out this keras function to automatically pad sequences that are too short.

Q: Describe the pad_sequences function and how it manages sequences of variable

length. What does the "padding" argument determine, and which setting makes the most sense for our use case here?

A: The pad_sequences function equalizes the lengths of the sequence-based data by adding padding.

The "padding" argument determines if this padding is applied at the start ('pre') or end ('post') of sequences. We used "pre" because, for tasks involving RNNs like LSTMs, 'pre' padding is often used, especially for time series data where recent context is critical. This helps us ensure the model's focus on the latest input before making its prediction. For our use case of simulating a streaming scenario, 'pre' padding is preferred as it maintains the sequence's order and the integrity of the most recent observations.

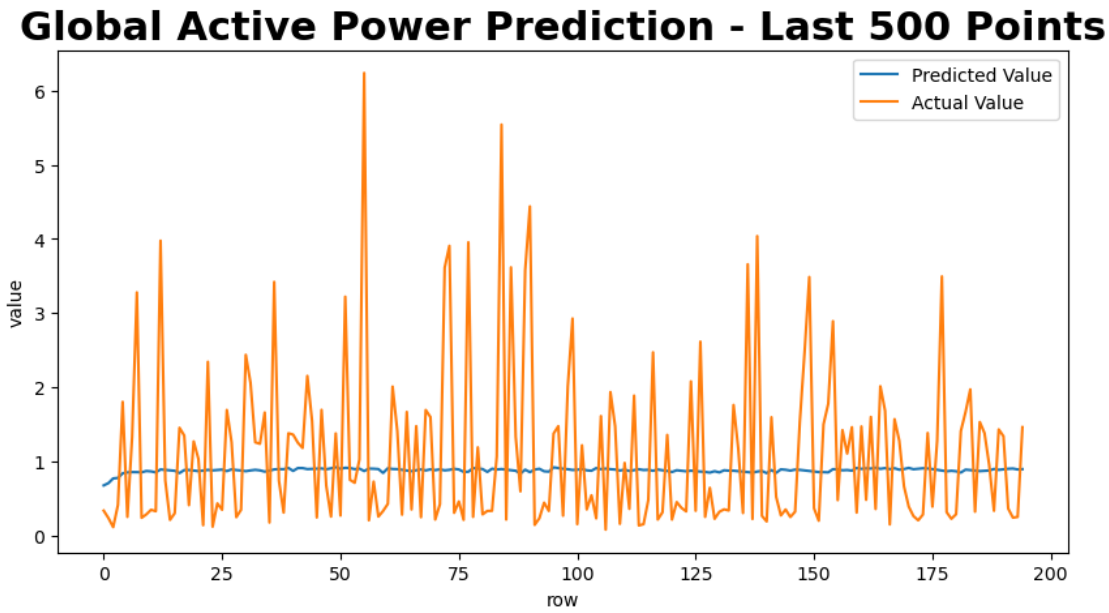
[42]:
```
val_arrays = []
val_labs = []

# create list of GAP readings starting with a minimum of two readings

# start after the second reading (index 1)
for i in range(1, len(val_df) - ph + 1):
    # each sequence should have at least 2 readings and a max of seq_length␣
 ↪readings
    input_seq = val_df[feat_cols].iloc[max(i - seq_length + 1, 0):i+1].values
    val_arrays.append(input_seq)
    # label is the GAP measurement ph steps after the current sequence end
    val_labs.append(val_df[feat_cols].iloc[i + ph - 1].values)

# use the pad_sequences function on input sequences - pad with zeros on the␣
 ↪left side ('pre') to reach seq_length
# we will later want our datatype to be np.float32
val_arrays = pad_sequences(val_arrays, maxlen=seq_length, dtype='float32',␣
 ↪padding='pre')


#convert labels to numpy arrays and floats to appease keras/tensorflow
val_labs = np.array(val_labs).astype('float32')
```

We will now run this validation data through our LSTM model and visualize its performance like we did on the linear regression data.

[46]:
```
scores_test = model.evaluate(val_arrays, val_labs, verbose=2)
print('\nMSE: {}'.format(scores_test[1]))

y_pred_test = model.predict(val_arrays)
y_true_test = val_labs

test_set = pd.DataFrame(y_pred_test)
test_set.to_csv('submit_test.csv', index = None)

# plot predicted data vs. the actual data
```

```
fig_verify = plt.figure(figsize=(10, 5))
plt.plot(y_pred_test[-900:], label = 'Predicted Value')
plt.plot(y_true_test[-900:], label = 'Actual Value')
plt.title('Global Active Power Prediction - Last 500 Points', fontsize=22,␣
 ↪fontweight='bold')
plt.ylabel('value')
plt.xlabel('row')
plt.legend()
plt.show()
fig_verify.savefig("model_regression_verify.png")
```

```
7/7 - 0s - loss: 1.2706 - mse: 1.2706 - 18ms/epoch - 3ms/step

MSE: 1.2706388235092163
7/7 [==============================] - 0s 2ms/step
```



Q: How did your model perform? What can you tell about the model from the loss curves? What could we do to try to improve the model?

A: it performed badly possibly because of overfitting or insufficient learning. To improve our model we could try increasing the model complexity by adding more hidden layers and train for more epochs

1.5 Model Optimization

Now it's your turn to build an LSTM-based model in hopes of improving performance on this training set. Changes that you might consider include:

- Add more variables to the input sequences

- Change the optimizer and/or adjust the learning rate
- Change the sequence length and/or the predictive horizon
- Change the number of hidden layers in each of the LSTM layers
- Change the model architecture altogether–think about adding convolutional layers, linear layers, additional regularization, creating embeddings for the input data, changing the loss function, etc.

There isn't any minimum performance increase or number of changes that need to be made, but I want to see that you have tried some different things. Remember that building and optimizing deep learning networks is an art and can be very difficult, so don't make yourself crazy trying to optimize for this assignment.

Q: What changes are you going to try with your model? Why do you think these changes could improve model performance?

A: add more hidden layers and train for more epochs

[50]:
```
# play with your ideas for optimization here

# define path to save model
model_path = 'LSTM_modeloptimized.h5'

# build the network
nb_features = seq_arrays.shape[2]  # Number of features
nb_out = 1  # Single real-numbered output

model = Sequential()

# add more hidden units to the first LSTM layer
model.add(LSTM(
         input_shape=(seq_length, nb_features),
         units=50,  # Increased from 5 to 50
         return_sequences=True))
model.add(Dropout(0.2))

# add more hidden units to the second LSTM layer
model.add(LSTM(
          units=20,  # Increased from 3 to 20
          return_sequences=False))
model.add(Dropout(0.2))

# output layer remains unchanged
model.add(Dense(units=nb_out))
model.add(Activation('linear'))

# use the Adam optimizer (legacy for M1 mac) with the default learning rate
optimizer = keras.optimizers.legacy.Adam()
model.compile(loss='mean_squared_error', optimizer=optimizer, metrics=['mse'])
```

```
print(model.summary())

# fit the network, but with more epochs
history = model.fit(seq_arrays, seq_labs, epochs=200,  # Increased from 100 to␣
 ↪200
                    batch_size=500, validation_split=0.05, verbose=2,
                    callbacks=[keras.callbacks.
 ↪EarlyStopping(monitor='val_loss', min_delta=0, patience=10, verbose=0,␣
 ↪mode='min'),
                               keras.callbacks.ModelCheckpoint(model_path,␣
 ↪monitor='val_loss', save_best_only=True, mode='min', verbose=0)]
                    )

# List all data in history
print(history.history.keys())
```

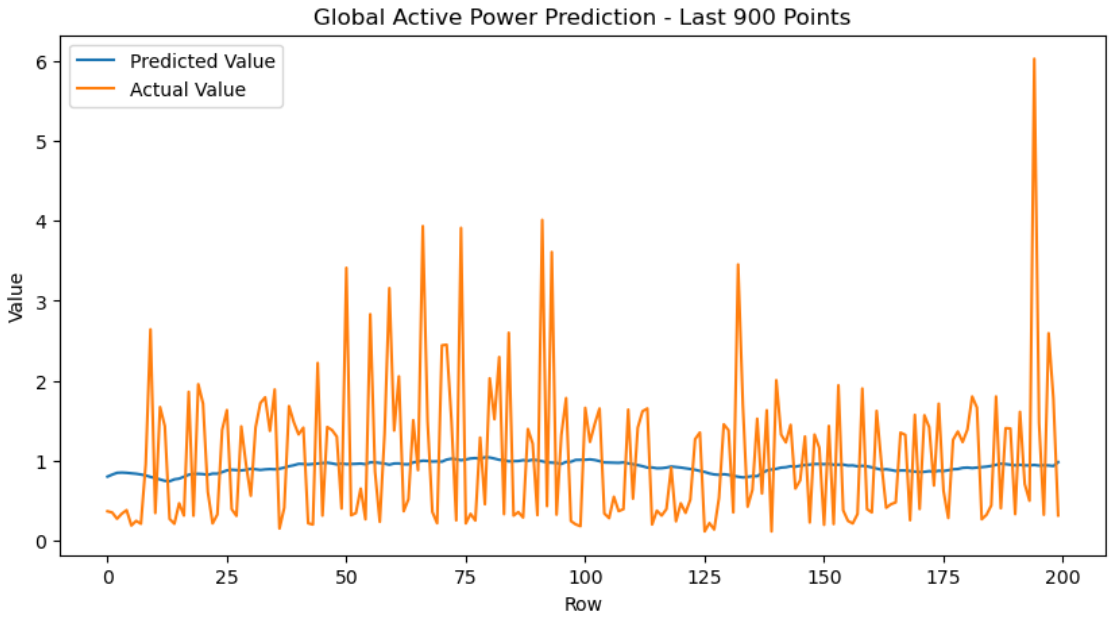
```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_6 (LSTM)               (None, 30, 50)            10400

 dropout_6 (Dropout)         (None, 30, 50)            0

 lstm_7 (LSTM)               (None, 20)                5680

 dropout_7 (Dropout)         (None, 20)                0

 dense_3 (Dense)             (None, 1)                 21

 activation_3 (Activation)   (None, 1)                 0

=================================================================
Total params: 16101 (62.89 KB)
Trainable params: 16101 (62.89 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
Epoch 1/200
2/2 - 2s - loss: 2.0144 - mse: 2.0144 - val_loss: 2.0839 - val_mse: 2.0839 -
2s/epoch - 831ms/step
Epoch 2/200
2/2 - 0s - loss: 1.7350 - mse: 1.7350 - val_loss: 1.8045 - val_mse: 1.8045 -
114ms/epoch - 57ms/step
Epoch 3/200
2/2 - 0s - loss: 1.5024 - mse: 1.5024 - val_loss: 1.5511 - val_mse: 1.5511 -
126ms/epoch - 63ms/step
```

```
Epoch 4/200
2/2 - 0s - loss: 1.3107 - mse: 1.3107 - val_loss: 1.3223 - val_mse: 1.3223 -
120ms/epoch - 60ms/step
Epoch 5/200
2/2 - 0s - loss: 1.1522 - mse: 1.1522 - val_loss: 1.1313 - val_mse: 1.1313 -
117ms/epoch - 58ms/step
Epoch 6/200
2/2 - 0s - loss: 1.0314 - mse: 1.0314 - val_loss: 1.0074 - val_mse: 1.0074 -
133ms/epoch - 67ms/step
Epoch 7/200
2/2 - 0s - loss: 1.0156 - mse: 1.0156 - val_loss: 0.9703 - val_mse: 0.9703 -
121ms/epoch - 60ms/step
Epoch 8/200
2/2 - 0s - loss: 1.0327 - mse: 1.0327 - val_loss: 0.9741 - val_mse: 0.9741 -
154ms/epoch - 77ms/step
Epoch 9/200
2/2 - 0s - loss: 1.0734 - mse: 1.0734 - val_loss: 0.9730 - val_mse: 0.9730 -
154ms/epoch - 77ms/step
Epoch 10/200
2/2 - 0s - loss: 1.0369 - mse: 1.0369 - val_loss: 0.9717 - val_mse: 0.9717 -
111ms/epoch - 56ms/step
Epoch 11/200
2/2 - 0s - loss: 1.0137 - mse: 1.0137 - val_loss: 0.9798 - val_mse: 0.9798 -
111ms/epoch - 56ms/step
Epoch 12/200
2/2 - 0s - loss: 0.9868 - mse: 0.9868 - val_loss: 0.9964 - val_mse: 0.9964 -
118ms/epoch - 59ms/step
Epoch 13/200
2/2 - 0s - loss: 1.0040 - mse: 1.0040 - val_loss: 1.0158 - val_mse: 1.0158 -
122ms/epoch - 61ms/step
Epoch 14/200
2/2 - 0s - loss: 0.9894 - mse: 0.9894 - val_loss: 1.0308 - val_mse: 1.0308 -
117ms/epoch - 58ms/step
Epoch 15/200
2/2 - 0s - loss: 0.9957 - mse: 0.9957 - val_loss: 1.0399 - val_mse: 1.0399 -
111ms/epoch - 55ms/step
Epoch 16/200
2/2 - 0s - loss: 1.0268 - mse: 1.0268 - val_loss: 1.0431 - val_mse: 1.0431 -
107ms/epoch - 54ms/step
Epoch 17/200
2/2 - 0s - loss: 0.9995 - mse: 0.9995 - val_loss: 1.0400 - val_mse: 1.0400 -
114ms/epoch - 57ms/step
dict_keys(['loss', 'mse', 'val_loss', 'val_mse'])
```

[53]:
```
# Evaluate the model on the validation set
scores_test = model.evaluate(seq_arrays, seq_labs, verbose=2)
print('\nMSE: {}'.format(scores_test[1]))
```

```
# Predict using the model on the validation set
y_pred_test = model.predict(seq_arrays)
y_true_test = seq_labs

# Save the prediction results to a CSV file
test_set = pd.DataFrame(y_pred_test)
test_set.to_csv('submit_test.csv', index = None)

# Plot the predicted data vs. the actual data for the last 900 points
fig_verify = plt.figure(figsize=(10, 5))
plt.plot(y_pred_test[-200:], label = 'Predicted Value')
plt.plot(y_true_test[-200:], label = 'Actual Value')
plt.title('Global Active Power Prediction - Last 900 Points')
plt.ylabel('Value')
plt.xlabel('Row')
plt.legend()
plt.show()

# Save the plot to an image file
fig_verify.savefig("model_regression_verify.png")
```

```
24/24 - 0s - loss: 0.9785 - mse: 0.9785 - 73ms/epoch - 3ms/step

MSE: 0.9785467982292175
24/24 [==============================] - 0s 2ms/step
```



Q: How did your model changes affect performance on the validation data? Why do you think they were/were not effective? If you were trying to optimize for production, what would you try next?

A: my changes didn't effect the model much. maybe the model still doesn't have enough capacity to capture the complexity of the data, possibly overfitting if we've now added too much capacity without proper regularization, or this could simply be because the additional features and data preprocessing were more critical than model architecture.

As next steps I would do more hyperparameter tuning and implement regularization techniques.

Q: How did the models that you built in this assignment compare to the linear regression model from last week? Think about model performance and other IoT device considerations; Which model would you choose to use in an IoT system that predicts GAP for a single household with a 5-minute predictive horizon, and why?

A: I did some research comparing LSTM models to linear regression for use in an IoT system. Sometimes the simpler linear regression is advantageous for its speed, lower computational demands, and easy interpretability, which are beneficial in resource-limited IoT devices. However, if complex temporal patterns in the data are crucial for accurate predictions, using an LSTM model may be more suitable for us, provided that our device has the necessary computational resources. The final decision on which model to deploy should weigh the performance gains against the computational costs and the specific accuracy needed for the IoT application.