LSTM Regression Assignment - Solutions

February 12, 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.*

[1]:
```
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.layers.core import Activation
from keras.utils import pad_sequences
```

```
2023-03-18 15:46:41.687045: I tensorflow/core/platform/cpu_feature_guard.cc:193]
This TensorFlow binary is optimized with oneAPI Deep Neural Network Library
(oneDNN) to use the following CPU instructions in performance-critical
operations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate
compiler flags.
2023-03-18 15:46:41.947516: E tensorflow/stream_executor/cuda/cuda_blas.cc:2981]
Unable to register cuBLAS factory: Attempting to register factory for plugin
cuBLAS when one has already been registered
```

[ ]:
```
#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?

[2]:
```
#Load your data into a pandas dataframe here
df = pd.read_csv("household_power_clean.csv")
```

[3]:
```
#create your training and validation sets here
```

```
#assign size for data subset
df_size = round(len(df)/100)

#take random data subset
start = np.random.choice(range(0,len(df)-df_size))
df_small = df.iloc[start:start+df_size].reset_index()

#split data subset 80/20 for train/validation
split_point = round(len(df_small)*0.8)
train_df = df_small.iloc[:split_point]
val_df = df_small.iloc[split_point:]
```

[4]:
```
#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

[18]:
```
seq_arrays = []
seq_labs = []
```

[19]:
```
# we'll start out with a 30 minute input sequence and a five 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 start in range(0,len(train_df)-seq_length-ph):
    seq_arrays.append(train_df[feat_cols].iloc[start:start+seq_length].
 ↪to_numpy())
    seq_labs.append(train_df['Global_active_power'].
 ↪iloc[start+seq_length+ph-1]) # subtract 1 for zero-indexing

#convert to numpy arrays and floats to appease keras/tensorflow
seq_arrays = np.array(seq_arrays, dtype = object).astype(np.float32)
```

```
seq_labs = np.array(seq_labs, dtype = object).astype(np.float32)
```

[7]:
```
assert(seq_arrays.shape == (len(train_df)-seq_length-ph,seq_length,␣
 ↪len(feat_cols)))
assert(seq_labs.shape == (len(train_df)-seq_length-ph,))
```

[8]:
```
seq_arrays.shape
```

[8]:
```
(16359, 30, 1)
```

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

A: These assert statements help to double check that our code outputs arrays of the correct shape/size. We use assert statements in our code to ensure that it is functioning as we expect it to–this helps with debugging during the development process and can be used to produce more mature exception-handling code before putting the code into production.

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

[9]:
```
# define path to save model
model_path = 'LSTM_model1.h5'

# build the network
nb_features = len(feat_cols) #number of features included in the training data
nb_out = 1 #expected output length

model = Sequential()

#add first LSTM layer
model.add(LSTM(
         input_shape=(seq_length, nb_features), #shape of input layer
         units=5, #number of hidden units
         return_sequences=True))
model.add(Dropout(0.2)) #dropout for regularization

# add second LSTM layer
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())
```

```
2023-03-18 15:47:06.217964: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:980] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2023-03-18 15:47:06.236364: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:980] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2023-03-18 15:47:06.236777: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:980] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2023-03-18 15:47:06.237498: I tensorflow/core/platform/cpu_feature_guard.cc:193]
This TensorFlow binary is optimized with oneAPI Deep Neural Network Library
(oneDNN) to use the following CPU instructions in performance-critical
operations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate
compiler flags.
2023-03-18 15:47:06.238654: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:980] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2023-03-18 15:47:06.238773: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:980] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2023-03-18 15:47:06.238869: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:980] successful NUMA node
```

```
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2023-03-18 15:47:06.657456: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:980] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2023-03-18 15:47:06.657816: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:980] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2023-03-18 15:47:06.657931: I
tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:980] successful NUMA node
read from SysFS had negative value (-1), but there must be at least one NUMA
node, so returning NUMA node zero
2023-03-18 15:47:06.658039: I
tensorflow/core/common_runtime/gpu/gpu_device.cc:1616] Created device
/job:localhost/replica:0/task:0/device:GPU:0 with 2100 MB memory:  -> device: 0,
name: NVIDIA GeForce RTX 3050 Ti Laptop GPU, pci bus id: 0000:01:00.0, compute
capability: 8.6

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
Trainable params: 252
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/100

2023-03-18 15:47:09.395888: I tensorflow/stream_executor/cuda/cuda_dnn.cc:384]
Loaded cuDNN version 8100
2023-03-18 15:47:10.317174: I tensorflow/stream_executor/cuda/cuda_blas.cc:1614]
TensorFloat-32 will be used for the matrix multiplication. This will only be
logged once.
```

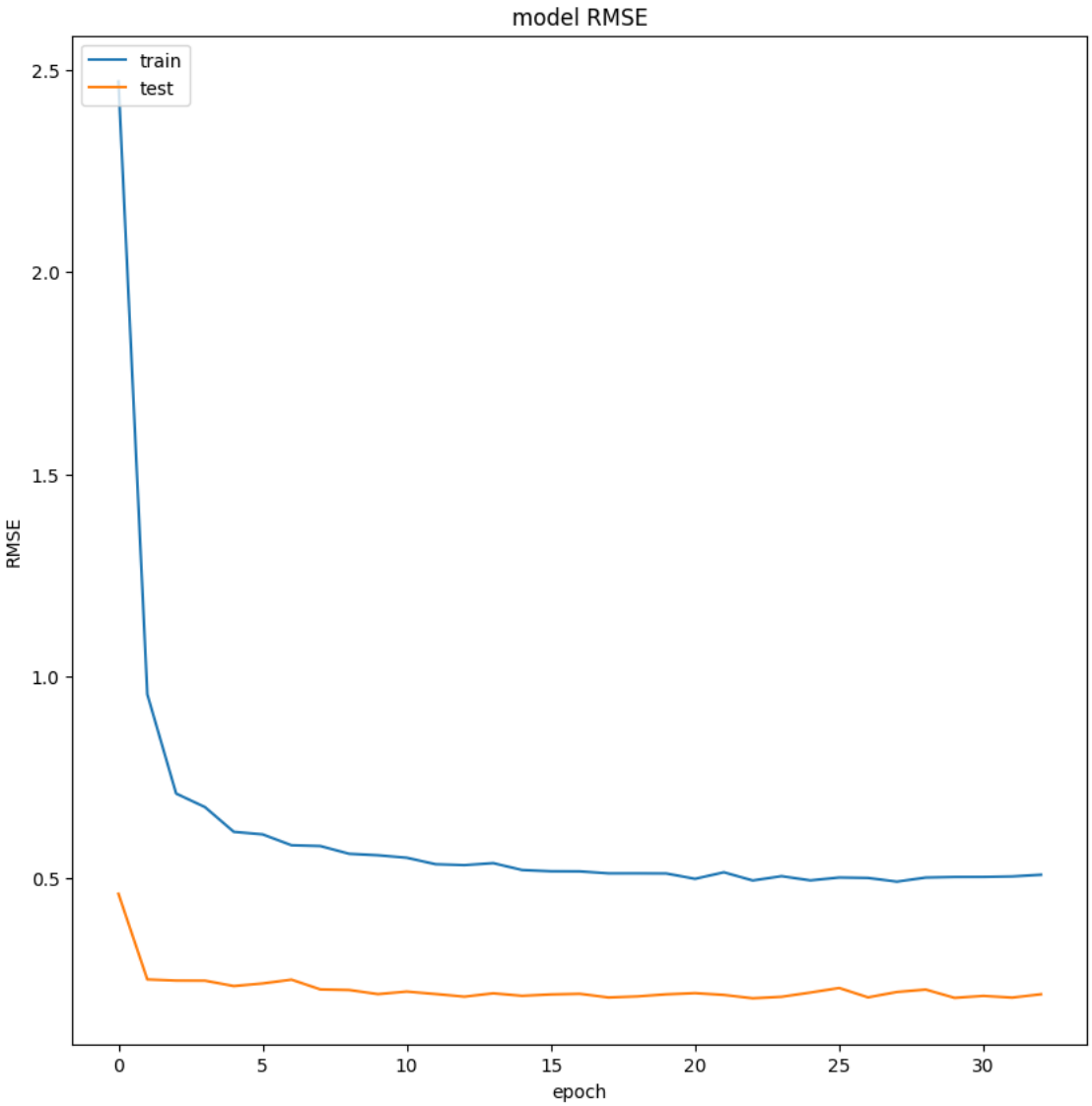
```
32/32 - 4s - loss: 2.4709 - mse: 2.4709 - val_loss: 0.4623 - val_mse: 0.4623 -
4s/epoch - 118ms/step
Epoch 2/100
32/32 - 0s - loss: 0.9556 - mse: 0.9556 - val_loss: 0.2505 - val_mse: 0.2505 -
142ms/epoch - 4ms/step
Epoch 3/100
32/32 - 0s - loss: 0.7101 - mse: 0.7101 - val_loss: 0.2477 - val_mse: 0.2477 -
136ms/epoch - 4ms/step
Epoch 4/100
32/32 - 0s - loss: 0.6766 - mse: 0.6766 - val_loss: 0.2474 - val_mse: 0.2474 -
137ms/epoch - 4ms/step
Epoch 5/100
32/32 - 0s - loss: 0.6154 - mse: 0.6154 - val_loss: 0.2341 - val_mse: 0.2341 -
129ms/epoch - 4ms/step
Epoch 6/100
32/32 - 0s - loss: 0.6092 - mse: 0.6092 - val_loss: 0.2405 - val_mse: 0.2405 -
117ms/epoch - 4ms/step
Epoch 7/100
32/32 - 0s - loss: 0.5823 - mse: 0.5823 - val_loss: 0.2500 - val_mse: 0.2500 -
118ms/epoch - 4ms/step
Epoch 8/100
32/32 - 0s - loss: 0.5803 - mse: 0.5803 - val_loss: 0.2259 - val_mse: 0.2259 -
127ms/epoch - 4ms/step
Epoch 9/100
32/32 - 0s - loss: 0.5612 - mse: 0.5612 - val_loss: 0.2243 - val_mse: 0.2243 -
136ms/epoch - 4ms/step
Epoch 10/100
32/32 - 0s - loss: 0.5575 - mse: 0.5575 - val_loss: 0.2142 - val_mse: 0.2142 -
131ms/epoch - 4ms/step
Epoch 11/100
32/32 - 0s - loss: 0.5514 - mse: 0.5514 - val_loss: 0.2204 - val_mse: 0.2204 -
117ms/epoch - 4ms/step
Epoch 12/100
32/32 - 0s - loss: 0.5353 - mse: 0.5353 - val_loss: 0.2144 - val_mse: 0.2144 -
122ms/epoch - 4ms/step
Epoch 13/100
32/32 - 0s - loss: 0.5331 - mse: 0.5331 - val_loss: 0.2079 - val_mse: 0.2079 -
134ms/epoch - 4ms/step
Epoch 14/100
32/32 - 0s - loss: 0.5380 - mse: 0.5380 - val_loss: 0.2161 - val_mse: 0.2161 -
117ms/epoch - 4ms/step
Epoch 15/100
32/32 - 0s - loss: 0.5211 - mse: 0.5211 - val_loss: 0.2099 - val_mse: 0.2099 -
118ms/epoch - 4ms/step
Epoch 16/100
32/32 - 0s - loss: 0.5179 - mse: 0.5179 - val_loss: 0.2133 - val_mse: 0.2133 -
126ms/epoch - 4ms/step
Epoch 17/100
```

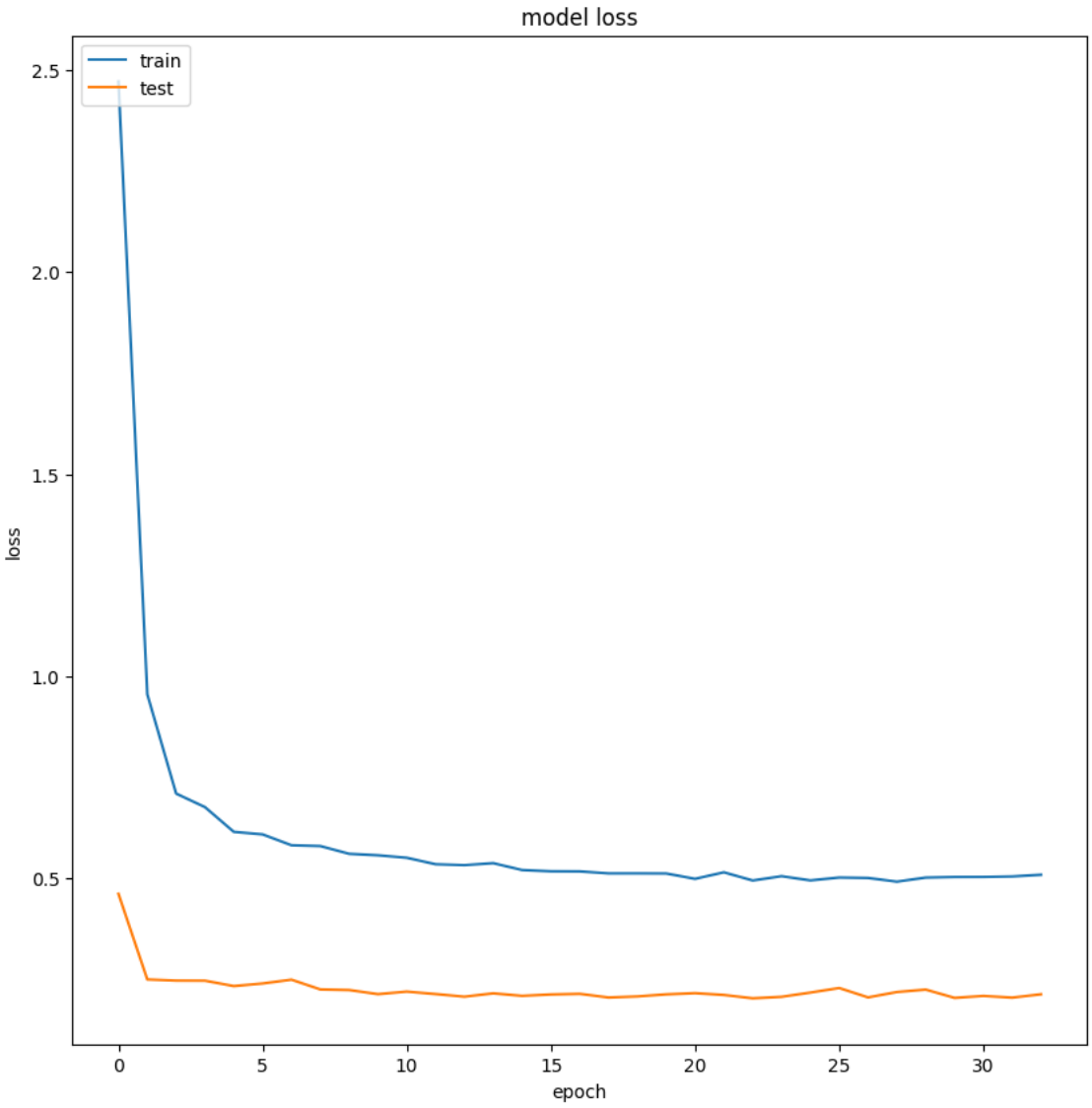
```
32/32 - 0s - loss: 0.5176 - mse: 0.5176 - val_loss: 0.2149 - val_mse: 0.2149 -
129ms/epoch - 4ms/step
Epoch 18/100
32/32 - 0s - loss: 0.5127 - mse: 0.5127 - val_loss: 0.2059 - val_mse: 0.2059 -
137ms/epoch - 4ms/step
Epoch 19/100
32/32 - 0s - loss: 0.5128 - mse: 0.5128 - val_loss: 0.2085 - val_mse: 0.2085 -
119ms/epoch - 4ms/step
Epoch 20/100
32/32 - 0s - loss: 0.5125 - mse: 0.5125 - val_loss: 0.2136 - val_mse: 0.2136 -
115ms/epoch - 4ms/step
Epoch 21/100
32/32 - 0s - loss: 0.4992 - mse: 0.4992 - val_loss: 0.2166 - val_mse: 0.2166 -
117ms/epoch - 4ms/step
Epoch 22/100
32/32 - 0s - loss: 0.5154 - mse: 0.5154 - val_loss: 0.2121 - val_mse: 0.2121 -
118ms/epoch - 4ms/step
Epoch 23/100
32/32 - 0s - loss: 0.4951 - mse: 0.4951 - val_loss: 0.2039 - val_mse: 0.2039 -
130ms/epoch - 4ms/step
Epoch 24/100
32/32 - 0s - loss: 0.5058 - mse: 0.5058 - val_loss: 0.2076 - val_mse: 0.2076 -
123ms/epoch - 4ms/step
Epoch 25/100
32/32 - 0s - loss: 0.4953 - mse: 0.4953 - val_loss: 0.2180 - val_mse: 0.2180 -
125ms/epoch - 4ms/step
Epoch 26/100
32/32 - 0s - loss: 0.5025 - mse: 0.5025 - val_loss: 0.2293 - val_mse: 0.2293 -
118ms/epoch - 4ms/step
Epoch 27/100
32/32 - 0s - loss: 0.5014 - mse: 0.5014 - val_loss: 0.2061 - val_mse: 0.2061 -
119ms/epoch - 4ms/step
Epoch 28/100
32/32 - 0s - loss: 0.4922 - mse: 0.4922 - val_loss: 0.2194 - val_mse: 0.2194 -
118ms/epoch - 4ms/step
Epoch 29/100
32/32 - 0s - loss: 0.5024 - mse: 0.5024 - val_loss: 0.2254 - val_mse: 0.2254 -
126ms/epoch - 4ms/step
Epoch 30/100
32/32 - 0s - loss: 0.5038 - mse: 0.5038 - val_loss: 0.2049 - val_mse: 0.2049 -
126ms/epoch - 4ms/step
Epoch 31/100
32/32 - 0s - loss: 0.5039 - mse: 0.5039 - val_loss: 0.2097 - val_mse: 0.2097 -
116ms/epoch - 4ms/step
Epoch 32/100
32/32 - 0s - loss: 0.5051 - mse: 0.5051 - val_loss: 0.2056 - val_mse: 0.2056 -
118ms/epoch - 4ms/step
Epoch 33/100
```

```
32/32 - 0s - loss: 0.5093 - mse: 0.5093 - val_loss: 0.2138 - val_mse: 0.2138 -
117ms/epoch - 4ms/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

[10]:
```
# summarize history for RMSE
fig_acc = plt.figure(figsize=(10, 10))
plt.plot(history.history['mse'])
plt.plot(history.history['val_mse'])
plt.title('model RMSE')
plt.ylabel('RMSE')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
fig_acc.savefig("LSTM_rmse1.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 RMSE
train
test
RMSE
epoch
2.5
2.0
1.5
1.0
0.5
0
5
10
15
20
25
30



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 adds a dummy value to the input sequence until it reaches the required sequence length. These dummy values can be added to the beginning or end of the input sequence with the "padding" argument, and the default setting (padding the beginning of the sentence) makes the most sense for our use case since we want the most recent reading to occur at the end of the input sequence.

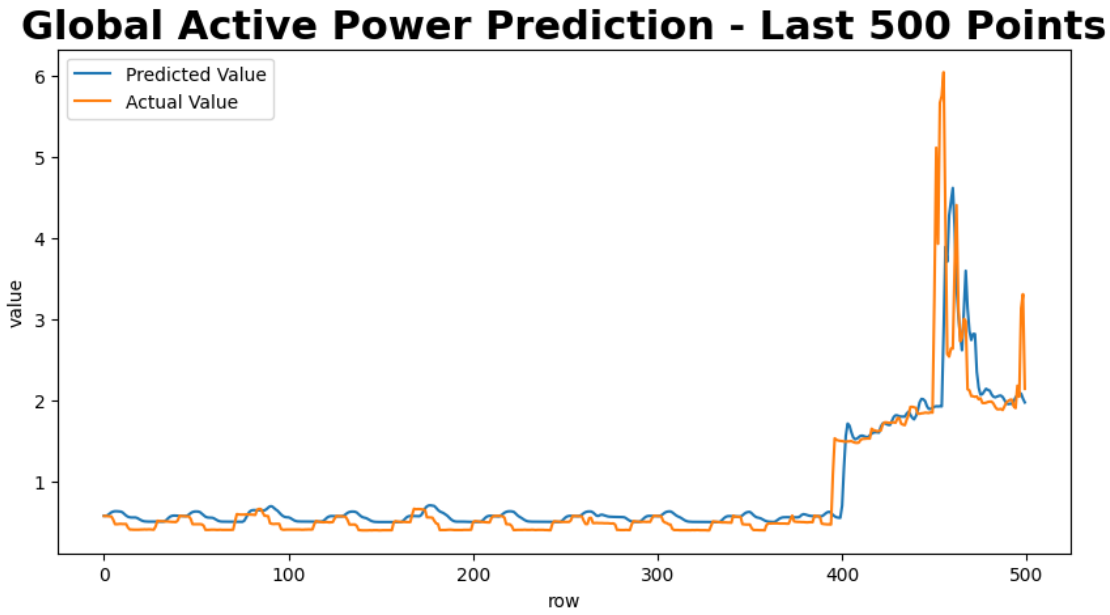
[11]:
```
val_arrays = []
val_labs = []

#create list of GAP readings starting with a minimum of two readings
for end in range(2, len(val_df)-ph):
    #add short sequences until we reach the sequence length
    if end < seq_length:
        val_arrays.append(val_df[feat_cols][0:end].to_numpy())
        val_labs.append(val_df['Global_active_power'][end+ph-1])
    #add sequences of seq_length once we have enough data
    else:
        val_arrays.append(val_df[feat_cols][end-seq_length:end].to_numpy())
        val_labs.append(val_df['Global_active_power'][end+ph-1])

# use the pad_sequences function on your input sequences
# remember that we will later want our datatype to be np.float32
val_arrays = pad_sequences(val_arrays, maxlen = seq_length, dtype = np.float32)

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

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

[12]:
```
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 the predicted data vs. the actual data
# we will limit our plot to the first 200 predictions for better visualization
fig_verify = plt.figure(figsize=(10, 5))
plt.plot(y_pred_test[-500:], label = 'Predicted Value')
plt.plot(y_true_test[-500:], 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")
```

```
128/128 - 0s - loss: 0.3373 - mse: 0.3373 - 221ms/epoch - 2ms/step

MSE: 0.3373478055000305
128/128 [==============================] - 1s 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: This model performs quite well. The predicted value follows the actual value quite closely on the validation set, and the MSE is relatively low (when you consider the range of the values in the data). We can also see that the loss and test accuracy curves are smooth lines that converge fairly quickly, and that the train and test curves follow each other without crossing. The smooth curves indicate that the model updates during training were consistently following the gradient. If the train set values had continued to improve while the test values got worse, this would indicate overfitting, possibly due to training the model too long or due to having too complicated of a model–this is not shown for our model.

Even though this model has performed quite well, we could still improve it by using a different architecture, adding variables to our input arrays, or changing hyperparameters such as the learning rate, sequence length, or predictive horizon.

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: I'm going to use a Wavenet-like architecture, where I add several convolutional layers to the model before I run it through an LSTM layer. This architecture has been particularly successful on audio data inputs, and while our data isn't quite as noisy/cyclical as audio data, it's an interesting model that is intended to be used on high-frequency sequential data, so I want to see how it compares to a more standard LSTM architecture.

[13]:
```
# play with your ideas for optimization here
from keras.layers import Conv1D, BatchNormalization

# define path to save model
model_path = 'LSTM_model2.h5'

# build the network
nb_features = len(feat_cols) #number of features included in the training data
nb_out = 1 #expected output length

model = Sequential()

#add first LSTM layer
model.add(Conv1D(32,
                 input_shape=(seq_length, nb_features),
                 kernel_size=2, padding="causal",
                 activation="relu"))
model.add(BatchNormalization())
model.add(Conv1D(48,
                 kernel_size=2,
                 padding="causal",
                 activation="relu",
                 dilation_rate=2))
```

```
model.add(BatchNormalization())
model.add(Conv1D(64,
                 kernel_size=2,
                 padding="causal",
                 activation="relu",
                 dilation_rate=4))
model.add(BatchNormalization())
model.add(LSTM(100))
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())
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d (Conv1D)             (None, 30, 32)            96

 batch_normalization (BatchN  (None, 30, 32)           128
 ormalization)

 conv1d_1 (Conv1D)           (None, 30, 48)            3120

 batch_normalization_1 (Batc  (None, 30, 48)           192
 hNormalization)

 conv1d_2 (Conv1D)           (None, 30, 64)            6208

 batch_normalization_2 (Batc  (None, 30, 64)           256
 hNormalization)

 lstm_2 (LSTM)               (None, 100)               66000

 dense_1 (Dense)             (None, 1)                 101

 activation_1 (Activation)   (None, 1)                 0

=================================================================
Total params: 76,101
Trainable params: 75,813
Non-trainable params: 288
_________________________________________________________________
None
```

[14]:
```
# fit the network
history = model.fit(seq_arrays, seq_labs, epochs=100, batch_size=100,␣
↪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())
```

```
Epoch 1/100
```

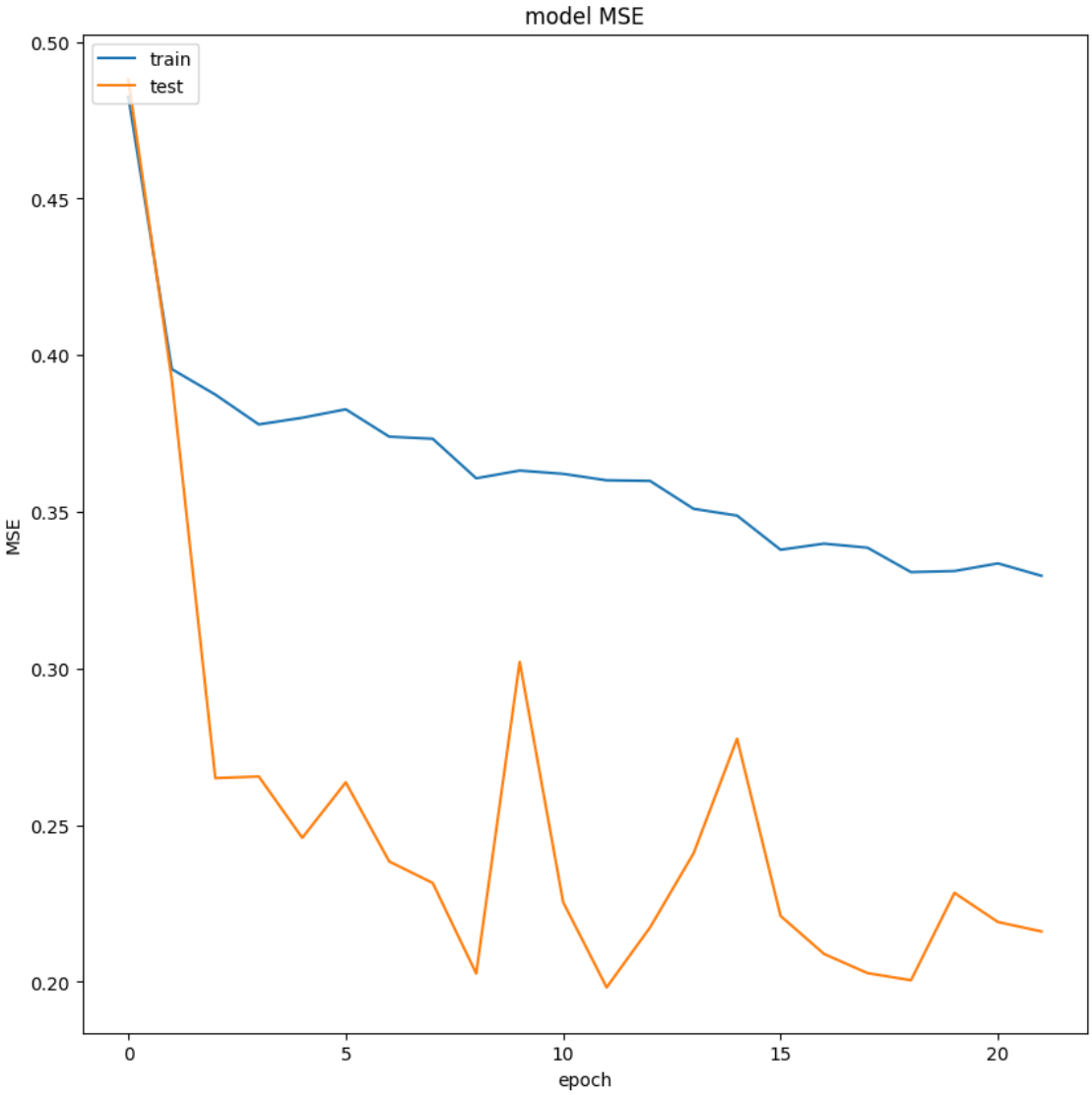
```
2023-03-18 15:49:28.091692: I
tensorflow/core/platform/default/subprocess.cc:304] Start cannot spawn child
process: No such file or directory
```

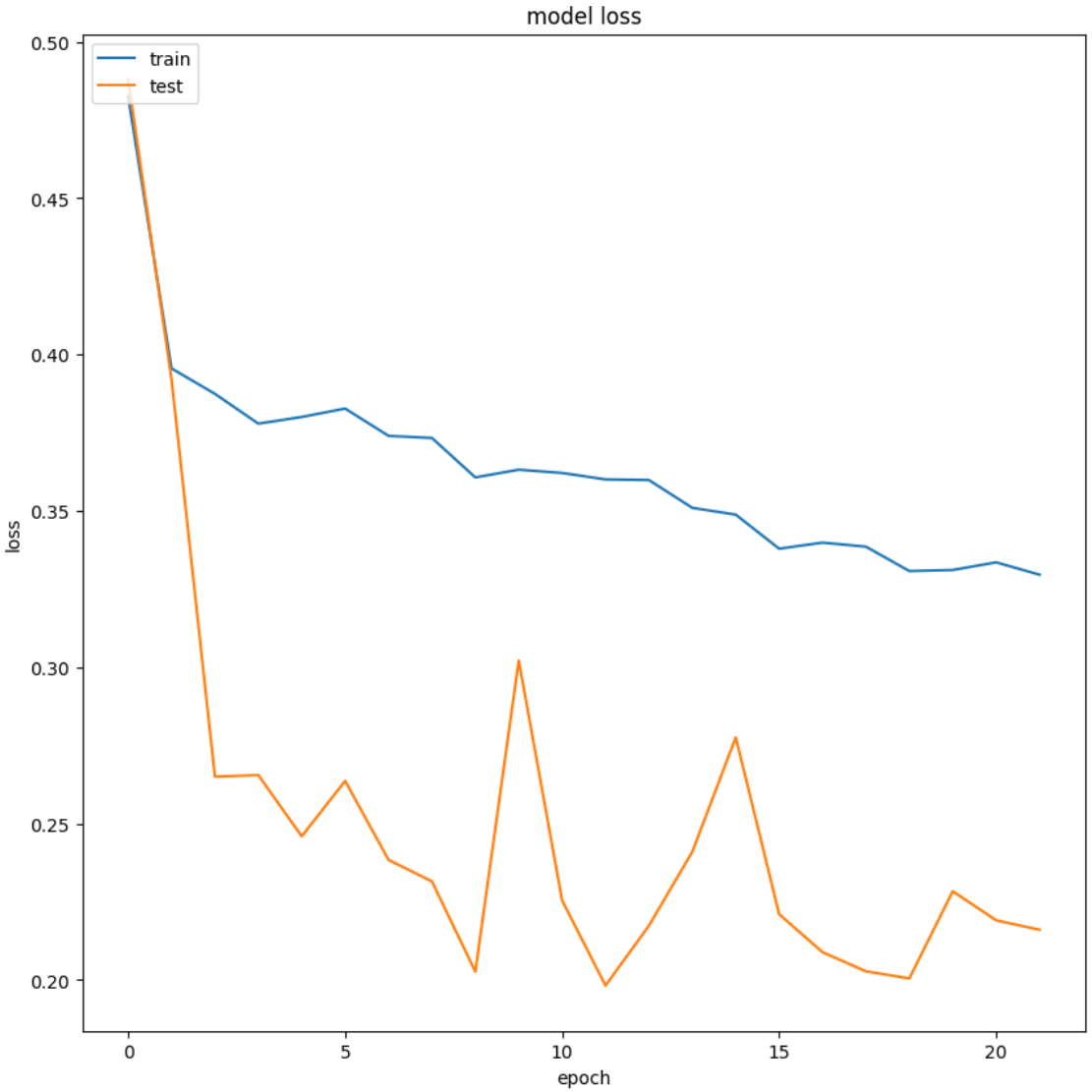
```
156/156 - 4s - loss: 0.4825 - mse: 0.4825 - val_loss: 0.4880 - val_mse: 0.4880 -
4s/epoch - 24ms/step
Epoch 2/100
156/156 - 1s - loss: 0.3956 - mse: 0.3956 - val_loss: 0.3921 - val_mse: 0.3921 -
865ms/epoch - 6ms/step
Epoch 3/100
156/156 - 1s - loss: 0.3874 - mse: 0.3874 - val_loss: 0.2650 - val_mse: 0.2650 -
683ms/epoch - 4ms/step
Epoch 4/100
156/156 - 1s - loss: 0.3779 - mse: 0.3779 - val_loss: 0.2656 - val_mse: 0.2656 -
911ms/epoch - 6ms/step
Epoch 5/100
156/156 - 1s - loss: 0.3801 - mse: 0.3801 - val_loss: 0.2460 - val_mse: 0.2460 -
1s/epoch - 6ms/step
Epoch 6/100
156/156 - 1s - loss: 0.3827 - mse: 0.3827 - val_loss: 0.2637 - val_mse: 0.2637 -
938ms/epoch - 6ms/step
Epoch 7/100
156/156 - 1s - loss: 0.3740 - mse: 0.3740 - val_loss: 0.2384 - val_mse: 0.2384 -
744ms/epoch - 5ms/step
Epoch 8/100
156/156 - 1s - loss: 0.3733 - mse: 0.3733 - val_loss: 0.2316 - val_mse: 0.2316 -
711ms/epoch - 5ms/step
Epoch 9/100
156/156 - 1s - loss: 0.3607 - mse: 0.3607 - val_loss: 0.2027 - val_mse: 0.2027 -
736ms/epoch - 5ms/step
Epoch 10/100
156/156 - 1s - loss: 0.3632 - mse: 0.3632 - val_loss: 0.3022 - val_mse: 0.3022 -
831ms/epoch - 5ms/step
```

```
Epoch 11/100
156/156 - 1s - loss: 0.3621 - mse: 0.3621 - val_loss: 0.2255 - val_mse: 0.2255 -
654ms/epoch - 4ms/step
Epoch 12/100
156/156 - 1s - loss: 0.3601 - mse: 0.3601 - val_loss: 0.1982 - val_mse: 0.1982 -
669ms/epoch - 4ms/step
Epoch 13/100
156/156 - 1s - loss: 0.3599 - mse: 0.3599 - val_loss: 0.2174 - val_mse: 0.2174 -
933ms/epoch - 6ms/step
Epoch 14/100
156/156 - 1s - loss: 0.3510 - mse: 0.3510 - val_loss: 0.2410 - val_mse: 0.2410 -
723ms/epoch - 5ms/step
Epoch 15/100
156/156 - 1s - loss: 0.3488 - mse: 0.3488 - val_loss: 0.2776 - val_mse: 0.2776 -
806ms/epoch - 5ms/step
Epoch 16/100
156/156 - 1s - loss: 0.3379 - mse: 0.3379 - val_loss: 0.2211 - val_mse: 0.2211 -
693ms/epoch - 4ms/step
Epoch 17/100
156/156 - 1s - loss: 0.3399 - mse: 0.3399 - val_loss: 0.2089 - val_mse: 0.2089 -
842ms/epoch - 5ms/step
Epoch 18/100
156/156 - 1s - loss: 0.3386 - mse: 0.3386 - val_loss: 0.2028 - val_mse: 0.2028 -
751ms/epoch - 5ms/step
Epoch 19/100
156/156 - 1s - loss: 0.3308 - mse: 0.3308 - val_loss: 0.2005 - val_mse: 0.2005 -
751ms/epoch - 5ms/step
Epoch 20/100
156/156 - 1s - loss: 0.3311 - mse: 0.3311 - val_loss: 0.2284 - val_mse: 0.2284 -
909ms/epoch - 6ms/step
Epoch 21/100
156/156 - 1s - loss: 0.3336 - mse: 0.3336 - val_loss: 0.2191 - val_mse: 0.2191 -
733ms/epoch - 5ms/step
Epoch 22/100
156/156 - 1s - loss: 0.3296 - mse: 0.3296 - val_loss: 0.2161 - val_mse: 0.2161 -
1s/epoch - 7ms/step
dict_keys(['loss', 'mse', 'val_loss', 'val_mse'])
```

[15]:
```python
# 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_rmse2.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_loss2.png")
```

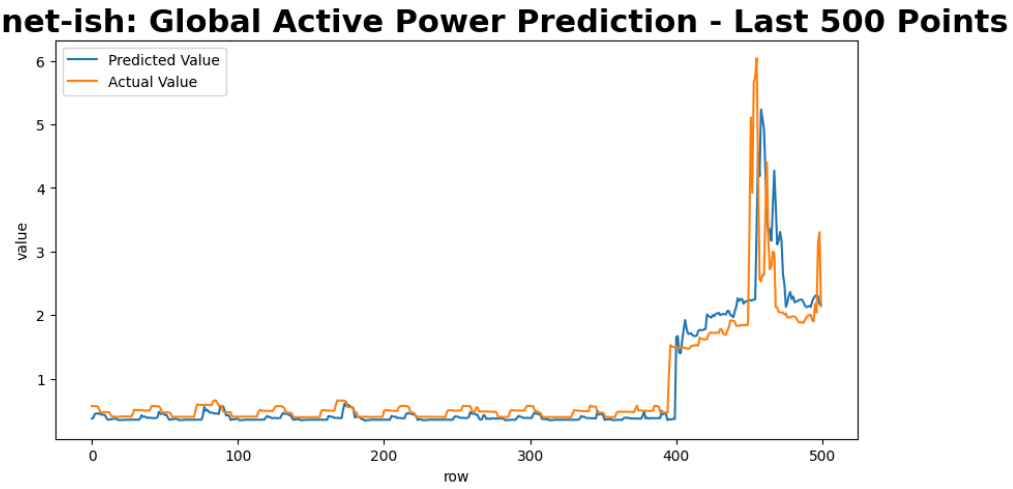




[16]:
```
# show me how one or two of your different models perform
# using the code from the "Validating our model" section above
scores_test = model.evaluate(val_arrays, val_labs, verbose=2)
print('\nMSE: {}'.format(scores_test[1]))

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 the predicted data vs. the actual data
# we will limit our plot to the first 200 predictions for better visualization
```

```
fig_verify = plt.figure(figsize=(10, 5))
plt.plot(y_pred_test[-500:], label = 'Predicted Value')
plt.plot(y_true_test[-500:], label = 'Actual Value')
plt.title('Wavenet-ish: 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")
```

```
128/128 - 0s - loss: 0.3408 - mse: 0.3408 - 290ms/epoch - 2ms/step

MSE: 0.3407917320728302
128/128 [==============================] - 0s 1ms/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: This model also performed quite well, though it hardly improved the MSE from the much smaller and more basic LSTM model. The model did train much more quickly, taking only 22 epochs instead of 32, but the loss and accuracy curves for this model were not nearly as smooth. This may indicate that the model is overly complex for the dataset, which shouldn't be too surprising given that this model has about 5x the parameters of our earlier model and the number of sequences in our training data. The next step would be to re-train either of these models with much more data and to see if the performance holds. Ideally we'd use the entire 3-month dataset, though that would take quite a bit of compute and time to train. I'd also like to see how an LSTM performs when applied to a moving average, like I did on the linear regression assignment.

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: The LSTM models built here performed about as well as the linear regression models did (though not as well as the model that worked with the moving average data). A device that predicts the GAP in a house is not likely to have strict physical limitations (e.g. it's not a wearable device that needs to be small), so it's realistic that we could use a deep learning method that would require more resources. However, for very similar performance it usually makes sense to use a simpler model, so I would probably choose to implement the linear regression model for this application.