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.

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.*

```
import keras
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import os
import tensorflow as ten
# Setting seed for reproducibility
```

```
np.random.seed(1234)
PYTHONHASHSEED = 0

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

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

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?

```
#Load your data into a pandas dataframe here
from google.colab import files
uploaded = files.upload()
df = pd.read_csv(list(uploaded.keys())[0])
```

Choose Files cleanpower_csv (2)

cleanpower_csv (2).csv(text/csv) - 3309000 bytes, last modified: 1/26/2026 - 100% done

```
Saving cleanpower_csv (2).csv to cleanpower_csv (2).csv
```

```
#create your training and validation sets here
#assign size for data subset
#use 1% data set
subset_size = int(len(df) * 0.01)

#take random data subset
df = df.sort_values('Datetime').reset_index(drop=True)
df_subset = df.iloc[:subset_size]
#split data subset 80/20 for train/validation
train_df=df_subset.iloc[:int(len(df_subset) * 0.8)]
val_df = df_subset.iloc[int(len(df_subset) * 0.8):]
```

```
df.head()
```

	Date	Time	Global_active_power	Global_reactive_power	Voltage	Glol
0	2006-12-16	17:24:00	4.216	0.418	234.84	
1	2006-12-16	18:24:00	3.452	0.000	235.20	
2	2006-12-16	19:24:00	3.262	0.052	232.64	
3	2006-12-16	20:24:00	3.286	0.000	232.31	
4	2006-12-16	21:24:00	3.410	0.054	236.91	

Next steps: Generate code with df New interactive sheet

```
#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

```
seq_arrays = []
seq_labs = []
```

```


seq_length = 30
ph = 5
feat_cols = ['Global_active_power']

#create list of sequence length GAP readings

seq_arrays=[]
seq_labs=[]
num_rows = len(train_df)
for start in range(0, num_rows- seq_length-ph):
    seq_arrays.append(train_df[feat_cols].iloc[start:start+seq_length].t
    seq_labs.append(train_df['Global_active_power'].iloc[start+seq_lengt

#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)
```

```
assert(seq_arrays.shape == (len(train_df)-seq_length-ph,seq_length, len(
assert(seq_labs.shape == (len(train_df)-seq_length-ph,))
```

```
seq_arrays.shape
```

```
(237, 30, 1)
```

The function of the asserts function is to very assumptions at the start of the runtime. This is to catch the mistakes early ensuring the model trains valid data.

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

A:

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

```
# define path to save model
model_path = 'LSTM_model1.keras'

# build the network
nb_features = seq_arrays.shape[2]
nb_out =1

model = Sequential()

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

# 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=['m

print(model.summary())

# fit the network
history = model.fit(seq_arrays ,seq_labs, epochs=100, batch_size=500, va
          callbacks = [keras.callbacks.EarlyStopping(monitor='val_loss',
                       keras.callbacks.ModelCheckpoint(model_path,monito
          )

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

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/rnn/rnn.py:199:
  super().__init__(**kwargs)
Model: "sequential_1"
```

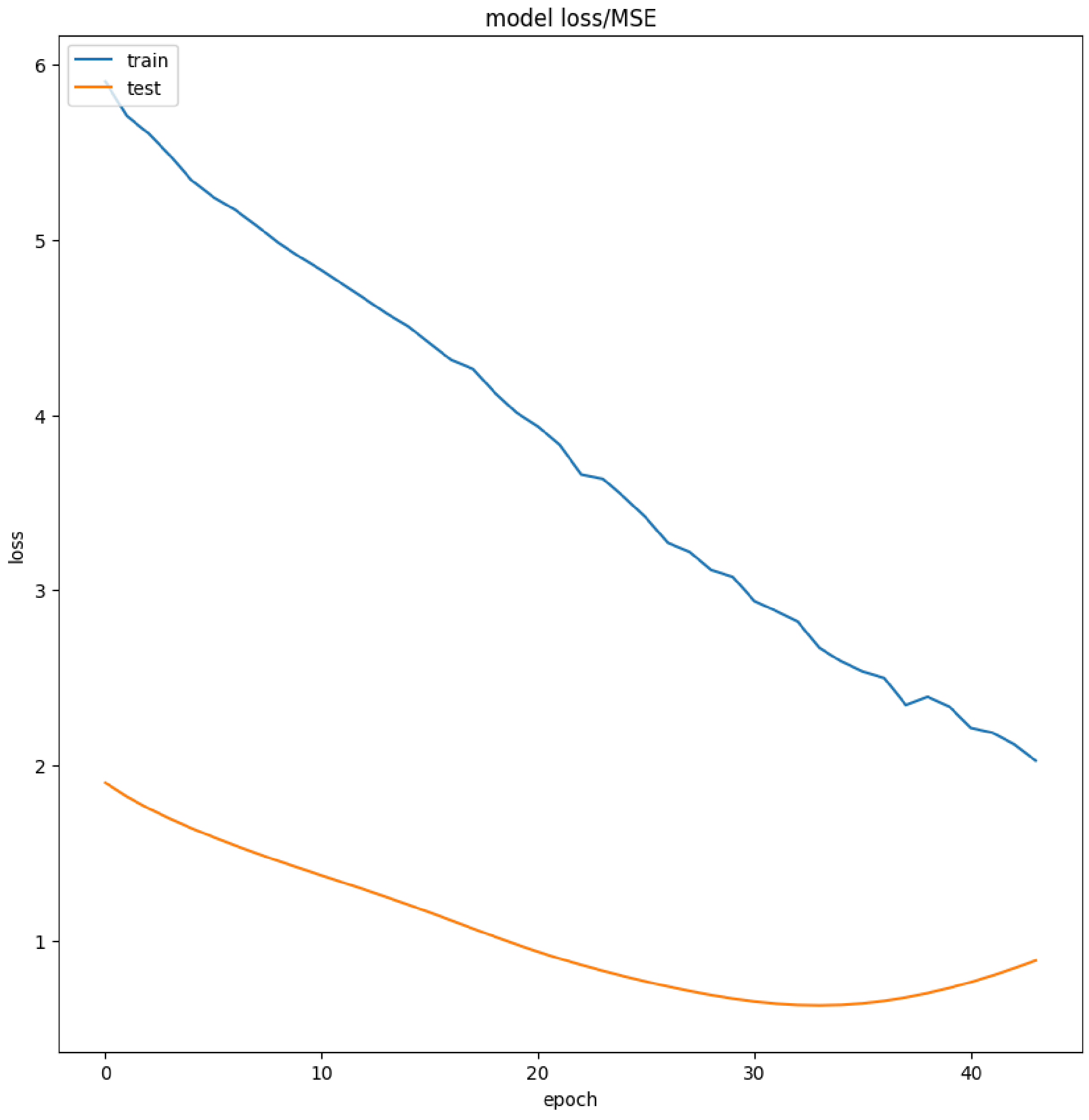
Layer (type)	Output Shape	Para
lstm_2 (LSTM)	(None, 30, 5)	1
dropout_2 (Dropout)	(None, 30, 5)	
lstm_3 (LSTM)	(None, 3)	1
dropout_3 (Dropout)	(None, 3)	
dense_1 (Dense)	(None, 1)	
activation_1 (Activation)	(None, 1)	

```
 Total params: 252 (1008.00 B)
 Trainable params: 252 (1008.00 B)
 Non-trainable params: 0 (0.00 B)
None
Epoch 1/100
1/1 - 5s - 5s/step - loss: 5.9047 - mse: 5.9047 - val_loss: 1.9007 - va
Epoch 2/100
1/1 - 0s - 144ms/step - loss: 5.7074 - mse: 5.7074 - val_loss: 1.8214 -
Epoch 3/100
1/1 - 0s - 150ms/step - loss: 5.6077 - mse: 5.6077 - val_loss: 1.7542 -
Epoch 4/100
1/1 - 0s - 149ms/step - loss: 5.4808 - mse: 5.4808 - val_loss: 1.6948 -
Epoch 5/100
1/1 - 0s - 148ms/step - loss: 5.3390 - mse: 5.3390 - val_loss: 1.6405 -
Epoch 6/100
1/1 - 0s - 169ms/step - loss: 5.2437 - mse: 5.2437 - val_loss: 1.5901 -
Epoch 7/100
1/1 - 0s - 148ms/step - loss: 5.1712 - mse: 5.1712 - val_loss: 1.5424 -
Epoch 8/100
1/1 - 0s - 146ms/step - loss: 5.0804 - mse: 5.0804 - val_loss: 1.4973 -
Epoch 9/100
1/1 - 0s - 146ms/step - loss: 4.9832 - mse: 4.9832 - val_loss: 1.4541 -
Epoch 10/100
1/1 - 0s - 143ms/step - loss: 4.9017 - mse: 4.9017 - val_loss: 1.4124 -
Epoch 11/100
1/1 - 0s - 146ms/step - loss: 4.8263 - mse: 4.8263 - val_loss: 1.3716 -
Epoch 12/100
1/1 - 0s - 162ms/step - loss: 4.7443 - mse: 4.7443 - val_loss: 1.3309 -
Epoch 13/100
1/1 - 0s - 145ms/step - loss: 4.6632 - mse: 4.6632 - val_loss: 1.2898 -
Epoch 14/100
1/1 - 0s - 152ms/step - loss: 4.5794 - mse: 4.5794 - val_loss: 1.2478 -
Epoch 15/100
1/1 - 0s - 145ms/step - loss: 4.5061 - mse: 4.5061 - val_loss: 1.2046 -
Epoch 16/100
1/1 - 0s - 148ms/step - loss: 4.4088 - mse: 4.4088 - val_loss: 1.1600 -
Epoch 17/100
1/1 - 0s - 143ms/step - loss: 4.3134 - mse: 4.3134 - val_loss: 1.1142 -
Epoch 18/100
```

```
1/1 - 0s - 149ms/step - loss: 4.2624 - mse: 4.2624 - val_loss: 1.0679 -
Epoch 19/100
1/1 - 0s - 164ms/step - loss: 4.1275 - mse: 4.1275 - val_loss: 1.0219 -
Epoch 20/100
1/1 - 0s - 150ms/step - loss: 4.0135 - mse: 4.0135 - val_loss: 0.9774 -
Epoch 21/100
1/1 - 0s - 148ms/step - loss: 3.9340 - mse: 3.9340 - val_loss: 0.9355 -
Epoch 22/100
1/1 - 0s - 149ms/step - loss: 3.8298 - mse: 3.8298 - val_loss: 0.8966 -
Epoch 23/100
1/1 - 0s - 143ms/step - loss: 3.6604 - mse: 3.6604 - val_loss: 0.8605 -
Epoch 24/100
1/1 - 0s - 147ms/step - loss: 3.6356 - mse: 3.6356 - val_loss: 0.8267 -
Epoch 25/100
1/1 - 0s - 142ms/step - loss: 3.5289 - mse: 3.5289 - val_loss: 0.7950 -
Epoch 26/100
1/1 - 0s - 177ms/step - loss: 3.4123 - mse: 3.4123 - val_loss: 0.7653 -
Epoch 27/100
1/1 - 0s - 148ms/step - loss: 3.2701 - mse: 3.2701 - val_loss: 0.7374 -
Epoch 28/100
1/1 - 0s - 144ms/step - loss: 3.2186 - mse: 3.2186 - val_loss: 0.7117 -
Epoch 29/100
1/1 - 0s - 142ms/step - loss: 3.1153 - mse: 3.1153 - val_loss: 0.6886 -
Epoch 30/100
1/1 - 0s - 148ms/step - loss: 3.0759 - mse: 3.0759 - val_loss: 0.6686 -
Epoch 31/100
1/1 - 0s - 144ms/step - loss: 2.9379 - mse: 2.9379 - val_loss: 0.6522 -
Epoch 32/100
1/1 - 0s - 163ms/step - loss: 2.8810 - mse: 2.8810 - val_loss: 0.6401 -
Epoch 33/100
1/1 - 0s - 153ms/step - loss: 2.8205 - mse: 2.8205 - val_loss: 0.6327 -
Epoch 34/100
1/1 - 1s - 514ms/step - loss: 2.6712 - mse: 2.6712 - val_loss: 0.6303 -
Epoch 35/100
1/1 - 0s - 429ms/step - loss: 2.5939 - mse: 2.5939 - val_loss: 0.6332 -
Epoch 36/100
1/1 - 0s - 445ms/step - loss: 2.5357 - mse: 2.5357 - val_loss: 0.6416 -
Epoch 37/100
1/1 - 0s - 141ms/step - loss: 2.4987 - mse: 2.4987 - val_loss: 0.6555 -
Epoch 38/100
1/1 - 0s - 106ms/step - loss: 2.3447 - mse: 2.3447 - val_loss: 0.6749 -
Epoch 39/100
1/1 - 0s - 112ms/step - loss: 2.3918 - mse: 2.3918 - val_loss: 0.6994 -
Epoch 40/100
1/1 - 0s - 161ms/step - loss: 2.3356 - mse: 2.3356 - val_loss: 0.7286 -
Epoch 41/100
1/1 - 1s - 527ms/step - loss: 2.2137 - mse: 2.2137 - val_loss: 0.7623 -
Epoch 42/100
1/1 - 0s - 110ms/step - loss: 2.1861 - mse: 2.1861 - val_loss: 0.7999 -
Epoch 43/100
1/1 - 0s - 108ms/step - loss: 2.1204 - mse: 2.1204 - val_loss: 0.8412 -
Epoch 44/100
1/1 - 0s - 109ms/step - loss: 2.0265 - mse: 2.0265 - val_loss: 0.8855 -
dict_keys(['loss', 'mse', 'val_loss', 'val_mse'])
```

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

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



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 datapoints, 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:

```
val_arrays = []
val_labs = []

#create list of GAP readings starting with a minimum of two readings
num_rows=len(val_df)
for end in range(2, num_rows - ph):
    val_arrays.append(val_df[feat_cols].iloc[:end].to_numpy())
    val_labs.append(val_df['Global_active_power'].iloc[end + ph])

# use the pad_sequences function on your input sequences
# remember that we will later want our datatype to be np.float32
val_arrays =  ten.keras.utils.pad_sequences(val_arrays,maxlen=seq_length
)
#convert labels to numpy arrays and floats to appease keras/tensorflow
val_labs = np.array(val_labs, dtype = object).astype(np.float32)
```

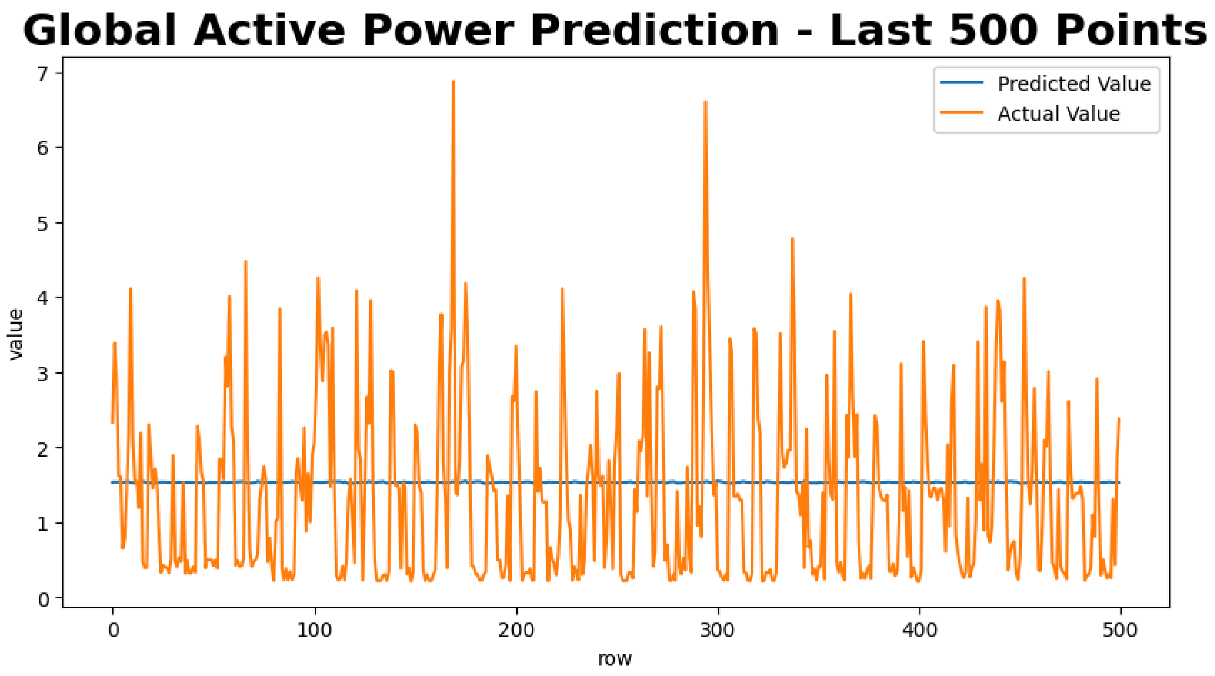
This function converts a list of many lengths into a 2d numpy array of shape(numsamples,num timesteps). The padding determins where pads values added pre or post.pre makes much more sense for us because it preserves more recent values. These values matter more for time series predictions.

Double-click (or enter) to edit

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

```
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 500 predictions for better visuali
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=2
plt.ylabel('value')
plt.xlabel('row')
plt.legend()
plt.show()
fig_verify.savefig("model_regression_verify.png")
```

```
64/64 - 0s - 6ms/step - loss: 1.5169 - mse: 1.5169

MSE: 1.5168596506118774
64/64 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
```



The model performed

Double-click (or enter) to edit

The model performed poorly it didnt capture the spikes and fluctuations in the data. To improve this model there needs to be much more data as none of the patterns were acurrately captured. To improve this model the best thing we can do is work with more data. We can also add longer sequences. Another improvement would be to add more variables.

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:

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:

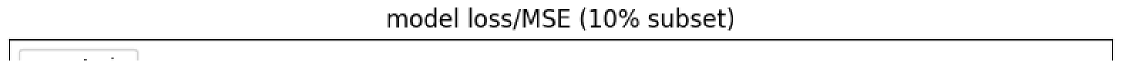
```
# play with your ideas for optimization here
# use 10% data set
subset_size = int(len(df) * 0.10)
```

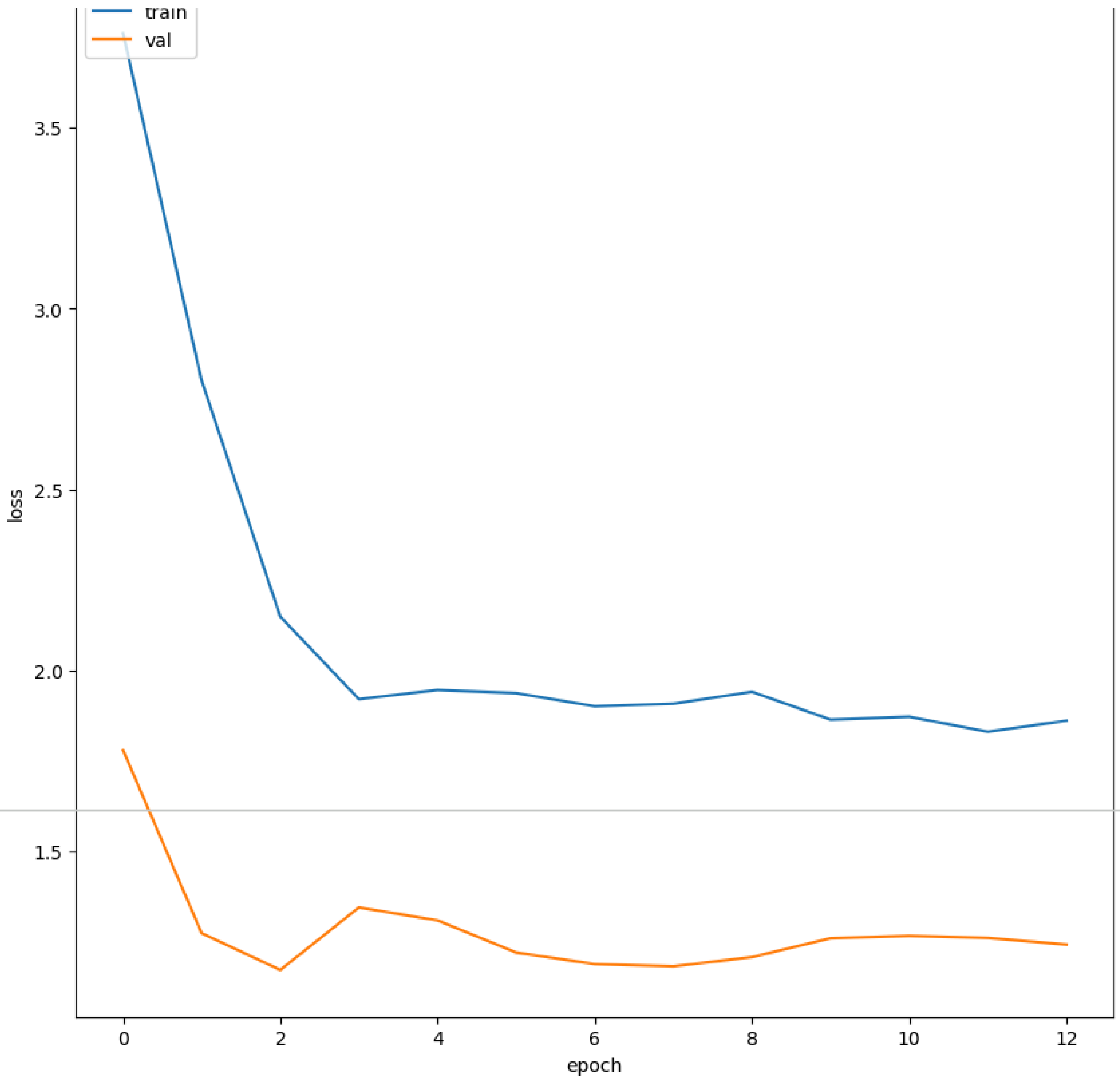
```
# keep time order (do NOT randomize for LSTM)
df = df.sort_values('Datetime').reset_index(drop=True)
df_subset = df.iloc[:subset_size]

# split data subset 80/20 for train/validation
train_df = df_subset.iloc[:int(len(df_subset) * 0.8)]
val_df = df_subset.iloc[int(len(df_subset) * 0.8):]

print("len(df):", len(df))
print("len(df_subset):", len(df_subset))
print("len(train_df):", len(train_df))
print("len(val_df):", len(val_df))

# create input and output sequences
seq_length = 30
ph = 5
feat_cols = ['Global_active_power']

seq_arrays = []
seq_labs = []

num_rows = len(train_df)
for start in range(0, num_rows - seq_length - ph):
    seq_arrays.append(train_df[feat_cols].iloc[start:start+seq_length].t
    seq_labs.append(train_df['Global_active_power'].iloc[start+seq_lengt

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

# make labels 2D so nb_out works
seq_labs = seq_labs.reshape(-1, 1).astype(np.float32)

print("seq_arrays.shape:", seq_arrays.shape)
print("seq_labs.shape:", seq_labs.shape)

# model training (same architecture, continuous output)
model_path = 'LSTM_model_10pct.keras'

nb_features = seq_arrays.shape[2]
nb_out = seq_labs.shape[1]

model = Sequential()

model.add(LSTM(
         input_shape=(seq_length, nb_features),
         units=5,
         return_sequences=True))
model.add(Dropout(0.2))

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=['

print(model.summary())

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, p
        keras.callbacks.ModelCheckpoint(model_path, monitor='val_loss',
    ]
)

print(history.history.keys())

fig_acc = plt.figure(figsize=(10, 10))
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss/MSE (10% subset)')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
fig_acc.savefig("LSTM_loss_10pct.png")

val_arrays = []
val_labs = []

num_rows = len(val_df)
for end in range(2, num_rows - ph):
    val_arrays.append(val_df[feat_cols].iloc[:end].to_numpy())
    val_labs.append(val_df['Global_active_power'].iloc[end + ph])

val_arrays = ten.keras.utils.pad_sequences(
    val_arrays,
    maxlen=seq_length,
    dtype='float32',
    padding='pre',
    truncating='pre',
    value=0.0
)

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

```
    val_labs = val_labs.reshape(-1, 1).astype(np.float32)

    print("val_arrays.shape:", val_arrays.shape)
    print("val_labs.shape:", val_labs.shape)

    scores_test = model.evaluate(val_arrays, val_labs, verbose=2)
    print("Validation MSE:", scores_test[1])

    y_pred = model.predict(val_arrays, batch_size=500)

    k = 500
    fig_pred = plt.figure(figsize=(12, 5))
    plt.plot(y_pred[-k:], label='Predicted Value')
    plt.plot(val_labs[-k:], label='Actual Value')
    plt.title('Global Active Power Prediction (Last 500 points) - 10% subset
    plt.legend()
    plt.show()
    fig_pred.savefig("LSTM_pred_10pct.png")
```

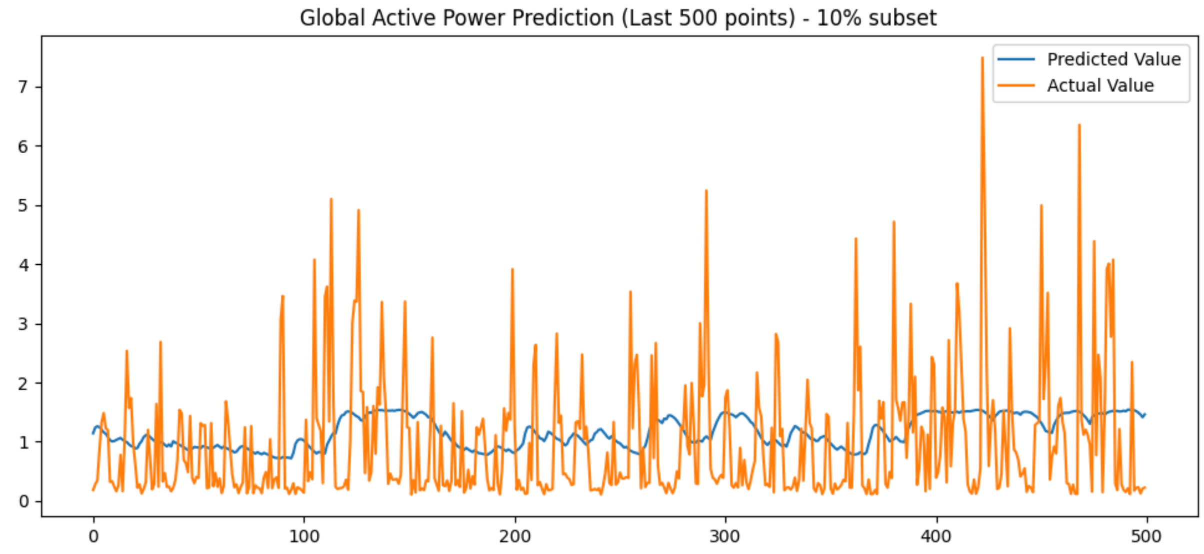
```
len(df): 34155
len(df_subset): 3415
len(train_df): 2732
len(val_df): 683
seq_arrays.shape: (2697, 30, 1)
seq_labs.shape: (2697, 1)
/usr/local/lib/python3.12/dist-packages/keras/src/layers/rnn/rnn.py:199
  super().__init__(**kwargs)
Model: "sequential_3"
```

```
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━
┃ Layer (type)                    ┃ Output Shape           ┃      Para
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━
│ lstm_6 (LSTM)                   │ (None, 30, 5)          │         1
├─────────────────────────────────┼────────────────────────┼──────────
│ dropout_6 (Dropout)             │ (None, 30, 5)          │
├─────────────────────────────────┼────────────────────────┼──────────
│ lstm_7 (LSTM)                   │ (None, 3)              │         1
├─────────────────────────────────┼────────────────────────┼──────────
│ dropout_7 (Dropout)             │ (None, 3)              │
├─────────────────────────────────┼────────────────────────┼──────────
│ dense_3 (Dense)                 │ (None, 1)              │
├─────────────────────────────────┼────────────────────────┼──────────
│ activation_3 (Activation)       │ (None, 1)              │
└─────────────────────────────────┴────────────────────────┴──────────

 Total params: 252 (1008.00 B)
 Trainable params: 252 (1008.00 B)
 Non-trainable params: 0 (0.00 B)
None
Epoch 1/100
6/6 - 4s - 636ms/step - loss: 3.7610 - mse: 3.7610 - val_loss: 1.7791 -
Epoch 2/100
6/6 - 0s - 48ms/step - loss: 2.8005 - mse: 2.8005 - val_loss: 1.2730 - v
Epoch 3/100
6/6 - 0s - 52ms/step - loss: 2.1482 - mse: 2.1482 - val_loss: 1.1716 - v
Epoch 4/100
6/6 - 1s - 114ms/step - loss: 1.9201 - mse: 1.9201 - val_loss: 1.3444 -
Epoch 5/100
6/6 - 0s - 67ms/step - loss: 1.9453 - mse: 1.9453 - val_loss: 1.3086 - v
Epoch 6/100
6/6 - 0s - 61ms/step - loss: 1.9367 - mse: 1.9367 - val_loss: 1.2197 - v
Epoch 7/100
6/6 - 0s - 69ms/step - loss: 1.9005 - mse: 1.9005 - val_loss: 1.1877 - v
Epoch 8/100
6/6 - 0s - 69ms/step - loss: 1.9074 - mse: 1.9074 - val_loss: 1.1814 - v
Epoch 9/100
6/6 - 0s - 50ms/step - loss: 1.9403 - mse: 1.9403 - val_loss: 1.2069 - v
Epoch 10/100
6/6 - 0s - 42ms/step - loss: 1.8637 - mse: 1.8637 - val_loss: 1.2591 - v
Epoch 11/100
6/6 - 0s - 41ms/step - loss: 1.8714 - mse: 1.8714 - val_loss: 1.2656 - v
Epoch 12/100
6/6 - 0s - 44ms/step - loss: 1.8301 - mse: 1.8301 - val_loss: 1.2600 - v
Epoch 13/100
6/6 - 0s - 42ms/step - loss: 1.8604 - mse: 1.8604 - val_loss: 1.2417 - v
dict_keys(['loss', 'mse', 'val_loss', 'val_mse'])
```





```
val_arrays.shape: (676, 30, 1)
val_labs.shape: (676, 1)
22/22 - 0s - 8ms/step - loss: 0.9970 - mse: 0.9970
Validation MSE: 0.9969973564147949
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 290ms/step
```



```
# play with your ideas for optimization here
# use 10% data set and change seq length from 30 to 50
subset_size = int(len(df) * 0.10)

# keep time order (do NOT randomize for LSTM)
df = df.sort_values('Datetime').reset_index(drop=True)
df_subset = df.iloc[:subset_size]

# split data subset 80/20 for train/validation
train_df = df_subset.iloc[:int(len(df_subset) * 0.8)]
val_df = df_subset.iloc[int(len(df_subset) * 0.8):]

print("len(df):", len(df))
print("len(df_subset):", len(df_subset))
print("len(train_df):", len(train_df))
print("len(val_df):", len(val_df))

# create input and output sequences
seq_length = 50
ph = 5
feat_cols = ['Global_active_power']

seq_arrays = []
seq_labs = []

num_rows = len(train_df)
for start in range(0, num_rows - seq_length - ph):
    seq_arrays.append(train_df[feat_cols].iloc[start:start+seq_length].t
    seq_labs.append(train_df['Global_active_power'].iloc[start+seq_lengt

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

# make labels 2D so nb_out works
seq_labs = seq_labs.reshape(-1, 1).astype(np.float32)

print("seq_arrays.shape:", seq_arrays.shape)
print("seq_labs.shape:", seq_labs.shape)

# model training (same architecture, continuous output)
model_path = 'LSTM_model_10pct.keras'

nb_features = seq_arrays.shape[2]
nb_out = seq_labs.shape[1]

model = Sequential()

model.add(LSTM(
         input_shape=(seq_length, nb_features),
         units=5,
         return_sequences=True))
model.add(Dropout(0.2))

model.add(LSTM(
          units=3,
          return_sequences=False))
model.add(Dropout(0.2))

model.add(Dense(units=nb_out))
model.add(Activation('linear'))

optimizer = ten.keras.optimizers.Adam(learning_rate=0.01)
model.compile(loss='mean_squared_error', optimizer=optimizer, metrics=['

print(model.summary())

history = model.fit(
    seq_arrays, seq_labs,
    epochs=100, batch_size=500, validation_split=0.05, verbose=2,
    callbacks=[
        ten.keras.callbacks.EarlyStopping(monitor='val_loss', min_delta=
        ten.keras.callbacks.ModelCheckpoint(model_path, monitor='val_los
    ]
)

print(history.history.keys())

fig_acc = plt.figure(figsize=(10, 10))
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss/MSE (10% subset)')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
fig_acc.savefig("LSTM_loss_10pct.png")

val_arrays = []
val_labs = []

num_rows = len(val_df)
for end in range(2, num_rows - ph):
    val_arrays.append(val_df[feat_cols].iloc[:end].to_numpy())
    val_labs.append(val_df['Global_active_power'].iloc[end + ph])

val_arrays = ten.keras.utils.pad_sequences(
    val_arrays,
    maxlen=seq_length,
    dtype='float32',
    padding='pre',
    truncating='pre',
    value=0.0
```

```
)

val_labs = np.array(val_labs, dtype=object).astype(np.float32)
val_labs = val_labs.reshape(-1, 1).astype(np.float32)

print("val_arrays.shape:", val_arrays.shape)
print("val_labs.shape:", val_labs.shape)

scores_test = model.evaluate(val_arrays, val_labs, verbose=2)
print("Validation MSE:", scores_test[1])

y_pred = model.predict(val_arrays, batch_size=500)

k = 500
fig_pred = plt.figure(figsize=(12, 5))
plt.plot(y_pred[-k:], label='Predicted Value')
plt.plot(val_labs[-k:], label='Actual Value')
plt.title('Global Active Power Prediction (Last 500 points) - 10% subset
plt.legend()
plt.show()
fig_pred.savefig("LSTM_pred_10pct.png")
```