Lab 1: Day-ahead load prediction for ERCOT (Texas) markets.

In this lab, you train a neural network to predict 24-hour aggregate load from Texas for a day using history of demands. The goals for this lab are:

1. Load the data and analyze to find patterns.
2. Define a neural network for the regression. Try different number of layers, learning rates, linear v/s nonlinear regression, activation functions, number of epochs, etc.
3. Explore the effects of wind energy on load prediction.

In [3]:
```
import os
import tensorflow as tf
from tensorflow import keras
import numpy as np
import pandas as pd
import random
import datetime
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
import matplotlib.pyplot as plt

# The following line suppresses certain warnings.
os.environ['TF_CPP_MIN_LOG_LEVEL']='2'
```

Load the ERCOT data from 2015.

The load data is given in the column named 'ERCOT Load, MW' in the csv file provided.

In [5]:
```
year = 2015
dfDemand = pd.read_csv("ERCOT_Hourly_Wind_Output_" + str(year) + ".csv")

demands = dfDemand['ERCOT Load, MW']

# Count the number of days for which we have demand data.
numberOfDays = int(len(demands)/24)
print("Hourly demand data loaded for %d days." % numberOfDays)
```

```
Hourly demand data loaded for 365 days.
```

Understand the data.

It is always useful to get accustomed to the data you are trying to learn. Visualize it if you can.

Q1. How does load vary over the year in Texas?

In [1]:
```
fig = plt.figure()

plt.plot([hour/24 for hour in range(numberOfDays * 24)], demands.values)
plt.xlabel("Days in " + str(year))
plt.ylabel("Net demand of Texas (in MW)")
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [1], in <module>
----> 1 fig = plt.figure()
      3 plt.plot([hour/24 for hour in range(numberOfDays * 24)], demands.valu
es)
      4 plt.xlabel("Days in " + str(year))

NameError: name 'plt' is not defined
```

Fact. A significant portion of the demand is usually thermal, i.e., for air conditioners and heating systems.

Question (10 points). From the above plot, what can you infer about the climate of Texas? What would you expect if you plotted the same in Illinois?

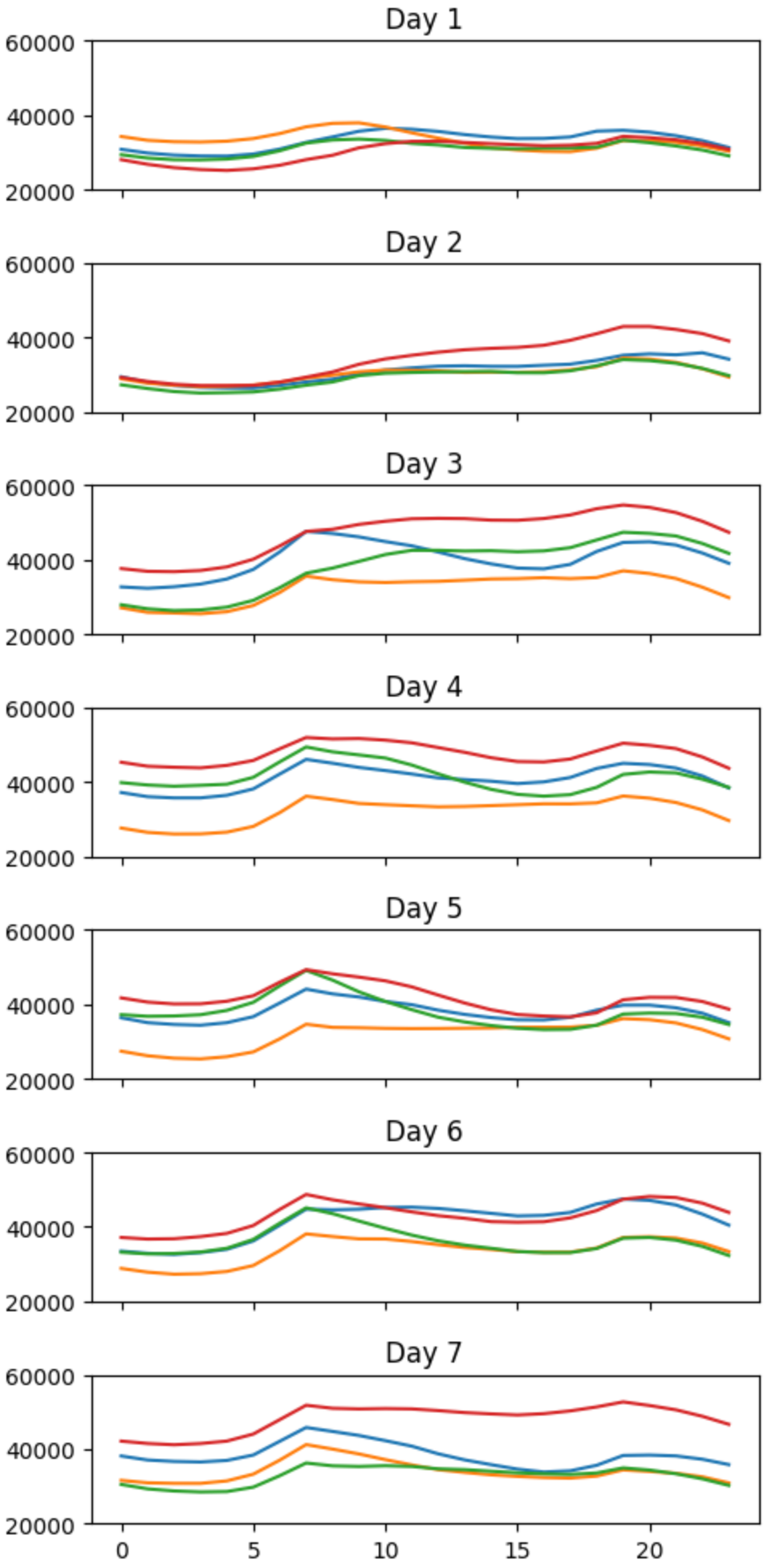
Your answer.

Since most of the demand is thermal based, the demand is most likely due to heaters and AC. This means the peak in the graph around 150-250 is AC the summer and the demand near the beginning is from the heater in the winter. The summer looks long and the winter looks short, so from the graph you can infer that Texas has long summers and short winters. Illinois has short summers and long winters, so a graph for Illinois would have a smaller range for the summer portion and a longer portion for the winter.

Q2. How does day of week affect the load profiles?

In [7]:
```
# Plot the load data of the same day of the week over several weeks.

dayStart = 30
numberOfWeeks = 4

DayOfWeek = ['Monday','Tuesday','Wednesday','Thursday','Friday','Saturday','Su
print("The first day in the first plot is Jan 31, " + str(year) + ".")
print("Day 1", "was a", DayOfWeek[datetime.date(year, 1, 31).weekday()] + ".")

fig, axs = plt.subplots(7, 1, sharex=True, figsize=(5,10))
axs = axs.ravel()

for dayInFirstWeek in range(7):
    for week in range(numberOfWeeks):

        axs[dayInFirstWeek].plot(range(24), dfDemand.loc[(dayStart + 7 * week
                                                         (dayStart + 7 * week
                                                         'ERCOT Load, MW'].val
    axs[dayInFirstWeek].set_ylim(bottom=20000, top=60000)
    axs[dayInFirstWeek].set_title("Day " + str(dayInFirstWeek + 1))

fig.tight_layout()
plt.show()
```

```
The first day in the first plot is Jan 31, 2015.
Day 1 was a Saturday.
```

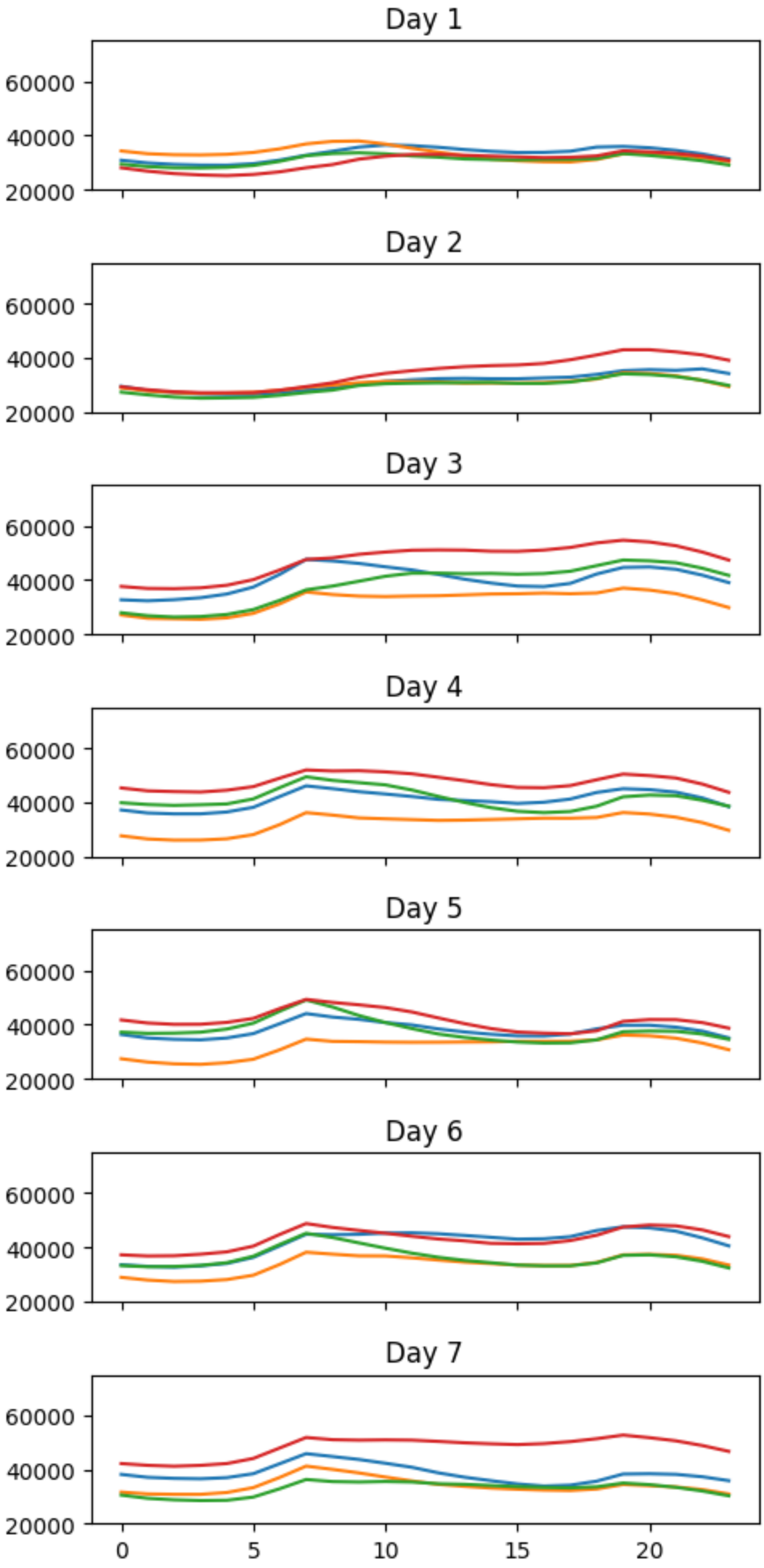

Day 1
Day 2
Day 3
Day 4
Day 5
Day 6
Day 7
60000
40000
20000
0
5
10
15
20

Question (15 points). Can you find any discernible change in the load profiles of different days of the week? Redo the above exercise for the months of August and September. Make 'Day 1' correspond to August 15th. What do you observe differently?

Your answer. From the graphs printed above, I saw that day 1-2 were more concentrated than the other days. I realized this might be have resulted in some sort of error. After corresponding day 1 to august 15, the results resulted in more of a concentrated graph where all the days looked more related to each other (no more big gaps between the lines).

In [8]:
```
# Modify the following code

# Plot the load data of the same day of the week over several weeks.

dayStart = 30
numberOfWeeks = 4

DayOfWeek = ['Monday','Tuesday','Wednesday','Thursday','Friday','Saturday','Su
print("The first day in the first plot is August 15, " + str(year) + ".")
print("Day 1", " was a ", DayOfWeek[datetime.date(year, 8, 15).weekday()] + ".

fig, axs = plt.subplots(7, 1, sharex=True, figsize=(5,10))
axs = axs.ravel()

for dayInFirstWeek in range(7):
    for week in range(numberOfWeeks):

        axs[dayInFirstWeek].plot(range(24), dfDemand.loc[(dayStart + 7 * week
                                                         (dayStart + 7 * week
                                                         'ERCOT Load, MW'].val
    axs[dayInFirstWeek].set_ylim(bottom=20000, top=75000)
    axs[dayInFirstWeek].set_title("Day " + str(dayInFirstWeek + 1))

fig.tight_layout()
plt.show()
```

```
The first day in the first plot is August 15, 2015.
Day 1  was a  Saturday.
```


Day 1
60000
40000
20000
Day 2
60000
40000
20000
Day 3
60000
40000
20000
Day 4
60000
40000
20000
Day 5
60000
40000
20000
Day 6
60000
40000
20000
Day 7
60000
40000
20000
0
5
10
15
20

Define the demand prediction module.

Use past demand profiles to predict demands a day in advance. We draw two conclusions from the above analysis:

1. Demand profiles have seasonal effects. Therefore, data from the past few days will help in predicting the demands tomorrow.
2. Demand profiles have weekly dependencies. Therefore, data from the same days but a week or two before can be useful in load prediction.

How much past data you want to train over depends on two considerations:

1. Which data in the past is useful in prediction?
2. How complex you want your training process to be? The more features of past data you want to train on, the more complex your neural network should be, and it will require more time to train it.

To strike a balance, use the demand profile from $d-7, d-2, d-1$ to predict the load profile of day d.

In [9]:
```
daysToTrainOn = [-7, -2, -1]
rangeOfDays = range(-np.min(daysToTrainOn), numberOfDays)

X = [np.concatenate([dfDemand.loc[(day + h) * 24: (day + h + 1) * 24 -1, 'ERCO
     for h in daysToTrainOn]) for day in rangeOfDays]
Y = [dfDemand.loc[day * 24: (day + 1) * 24 - 1, 'ERCOT Load, MW'].values.flatt
```

When you perform regression, it is often desirable to scale the inputs so that it has zero mean and unit variance. Other types of scaling are possible. Here, we cheat a little and scale both the training and test data together. Ideally, they should be scaled separately.

Split the data into two sets: training set and testing set. Train the neural network on the training set, and test how well it performs on the testing set. You should typically never sample from the training set to test your algorithms. The learnt model for prediction should work well on data that the algorithm has never encountered before.

The function 'train_test_split' helps you to split the data into two parts, where 'test_size' indicates the fraction of the data you want to test on.

In [10]:
```
X = preprocessing.StandardScaler().fit_transform(X)
trainX, testX, trainY, testY = train_test_split(X, Y, test_size=0.2)
trainX = trainX.astype(np.float32)
testX = testX.astype(np.float32)
trainY = np.array(trainY)
testY = np.array(testY)

print("Scaled and split the data into two parts:")

nTrain = np.shape(trainX)[0]
nTest = np.shape(testX)[0]

print("Neural network will train on data from %d days, and test on %d days." %
```

```
Scaled and split the data into two parts:
Neural network will train on data from 286 days, and test on 72 days.
```

Design the neural network (NN) for demand prediction with only one hidden layer.

Question (25 points). Insert code to design the NN and its optimizer (use the relu function)

In [15]:
```
nHidden = 150

# Store the dimension of each row of 'X' in 'nDimX' and that of 'Y' in 'nDimY'
nDimX = np.shape(trainX)[1]
nDimY = np.shape(trainY)[1]

# Construct the neural network using relu
inputs = keras.Input(shape=nDimX, name="input")
nn_layer = keras.layers.Dense(nHidden, activation="relu")(inputs)
outputs = keras.layers.Dense(nDimY, name="output")(nn_layer)

model = keras.Model(inputs=inputs, outputs=outputs)

# Define the loss function (MSE) and the optimizer (AdagradOptimizer).

# insert code
model.compile(
    optimizer= tf.optimizers.Adagrad(learning_rate=1),
    loss= 'mean_squared_error'
)
```

Train the neural network via Keras.

Create the training module for the NN.

Keras is a user-friendly framework to define, train and test neural networks. Check their page out for more details. https://keras.io/ (https://keras.io/)

Feed the training data in batches of size 'batchSize'.Usually, going through the training data once does not train your NN. You train over the same data multiple times. More precisely, train it 'nEpochs' times. It is similar to the idea that you never learn a material by reading through it once!

Question (20 points). Insert code to define the training module

In [12]:
```
batchSize = 50
nEpochs = 1000

# Train the model

print("Fit model on training data")

history = model.fit(
    trainX,
    trainY,
    batch_size = batchSize,
    epochs = nEpochs
)
```

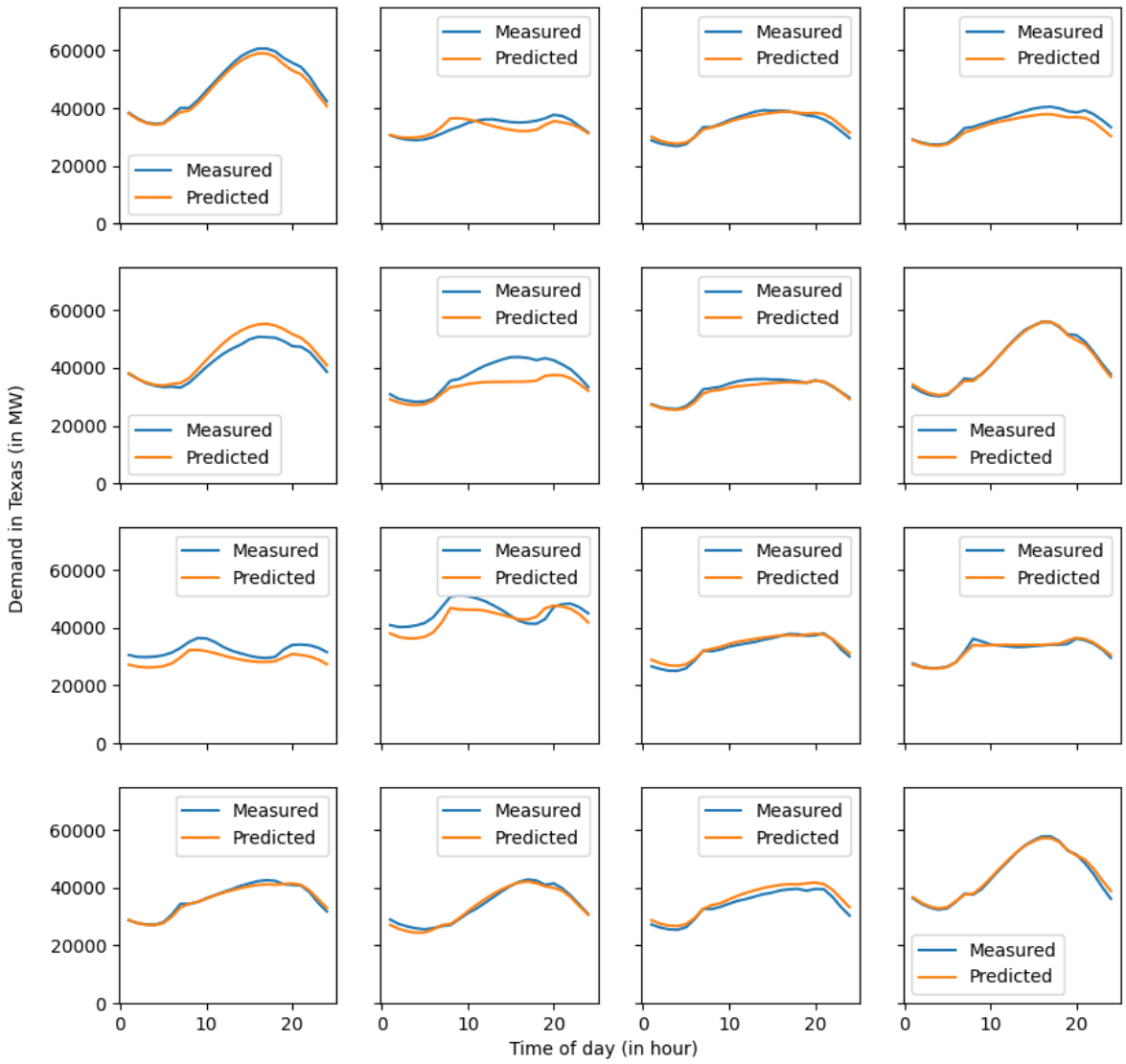
```
Epoch 22/1000
6/6 [==============================] - 0s 2ms/step - loss: 54623856.0000
Epoch 23/1000
6/6 [==============================] - 0s 2ms/step - loss: 53877172.0000
Epoch 24/1000
6/6 [==============================] - 0s 2ms/step - loss: 52039052.0000
Epoch 25/1000
6/6 [==============================] - 0s 3ms/step - loss: 51016768.0000
Epoch 26/1000
6/6 [==============================] - 0s 2ms/step - loss: 50052344.0000
Epoch 27/1000
6/6 [==============================] - 0s 3ms/step - loss: 50341440.0000
Epoch 28/1000
6/6 [==============================] - 0s 2ms/step - loss: 48663352.0000
Epoch 29/1000
6/6 [==============================] - 0s 2ms/step - loss: 48563476.0000
Epoch 30/1000
6/6 [==============================] - 0s 2ms/step - loss: 47569408.0000
Epoch 31/1000
6/6 [==============================] - 0s 2ms/step - loss: 46050968.0000
```

Let us visualize the results.

Question (5 points). Usine the NN to predict on test data

In [13]:
```
# Output the accuracy of the regressor on the test data.

predictedY = model.predict(testX)

# Plot the predicted load and compare against the actual load from the test da
assert(nTest >= 16)
days = random.sample(range(nTest), 16)

fig, axs = plt.subplots(4, 4, sharex=True, sharey=True, figsize=(10,10))
axs = axs.ravel()


for dd, day in enumerate(days):
    testYDay = testY[day]
    predictedYDay = predictedY[day]

    l1 = axs[dd].plot(range(1, 25), testYDay, label='Measured')
    l2 = axs[dd].plot(range(1, 25), predictedYDay, label='Predicted')

    axs[dd].set_ylim(bottom=0, top=75000)
    axs[dd].legend()

fig.text(0.5, 0.07, 'Time of day (in hour)', ha='center')
fig.text(0.04, 0.5, 'Demand in Texas (in MW)', va='center', rotation='vertical

plt.show()
```

```
3/3 [==============================] - 0s 2ms/step
```



Question (10 points). Explore how the number of epochs affects the accuracy and speed of training. Start with 10 epochs, and increase it to 100, 1000, 5000, 10000, and maybe more (do not exceed 20000 unless you have a powerful computer, you are only required to do up to 10000 for this lab). Make comments based on your observations. As an engineer, what is your favorite number of epochs, and why?

Your answer. When the epochs increase, the accuracy and performance increase as well. While the accuracy and performace increase, the cost of calculating all the results increase as well. This makes sense as more data in general reuqires more reqsources. As an engineer, I prefer 1000 epochs as it has the best speed and the best resources based on the other results.

Question (15 points). Fix the number of epochs to your favorite one. Then, add another layer to the network. Discuss what your observe in terms of speed and accuracy.

Your answer (comments here, code below). Your code should show the results for the case with an additional hidden layer. Go back to the codes above for the 1 layer case and run it again for the same number of epochs/neurons

After adding additional code to enchance the accuracy, I realized the accuracy and measure

The effect of wind energy (bonus).

In [10]:
```
#Let's check the raw data
dfDemand = pd.read_csv("ERCOT_Hourly_Wind_Output_" + str(year) + ".csv")
dfDemand[:]
```

Out[10]:

	time-date stamp	Date	ERCOT Load, MW	Total Wind Output, MW	Total Wind Installed, MW	Wind Output, % of Load	Wind Output, % of Installed	1-hr MW change	1-hr % change
0	1/1/15 0:00	1-Jan	39932	871	12730	2.2	6.8	NaN	NaN
1	1/1/15 1:00	1-Jan	39134	724	12730	1.8	5.7	-147.0	-16.9
2	1/1/15 2:00	1-Jan	38560	596	12730	1.5	4.7	-127.0	-17.6
3	1/1/15 3:00	1-Jan	38334	486	12730	1.3	3.8	-110.0	-18.5
4	1/1/15 4:00	1-Jan	38392	651	12730	1.7	5.1	165.0	33.8
...	...	...	...	...	...	...	...	...	...
8755	12/31/15 19:00	31-Dec	39909	3825	16170	9.6	23.7	484.0	14.5
8756	12/31/15 20:00	31-Dec	38737	4626	16170	11.9	28.6	801.0	20.9
8757	12/31/15 21:00	31-Dec	37588	4958	16170	13.2	30.7	332.0	7.2
8758	12/31/15 22:00	31-Dec	36356	4699	16170	12.9	29.1	-259.0	-5.2
8759	12/31/15 23:00	31-Dec	35150	4313	16170	12.3	26.7	-386.0	-8.2

8760 rows × 9 columns

Note that in addition to the load data, we have some wind data!

Question (20 points). Subtract the wind data from the load, and redo the above experiment and observe how does wind energy affect the forecasting process. How does the accuracy change? Why?

Your answer (comments here, code below). The wind energy leaded to a result of a decrease in accuracy. I believe this accuracy resulted in the fact that wind is one of the most important factors in terms of energy. Because of this, wind effects the energy changes a great difference and removing this factor changes the performance a great deal. This resulted in the overall decrease in accuracy.

In [17]:
```
dfDemand.columns
dfDemand_clear = dfDemand.drop(columns=['Total Wind Output, MW','Total Wind In
dfDemand_clear

daysToTrainOn = [-7, -2, -1]
rangeOfDays = range(-np.min(daysToTrainOn), numberOfDays)

X = [np.concatenate([dfDemand_clear.loc[(day + h) * 24: (day + h + 1) * 24 -1,
     for h in daysToTrainOn]) for day in rangeOfDays]
Y = [dfDemand_clear.loc[day * 24: (day + 1) * 24 - 1, 'ERCOT Load, MW'].values

X = preprocessing.StandardScaler().fit_transform(X)
trainX, testX, trainY, testY = train_test_split(X, Y, test_size=0.2)

trainX = trainX.astype(np.float32)
testX = testX.astype(np.float32)
trainY = np.array(trainY)
testY = np.array(testY)

nHidden = 150

# Store the dimension of each row of 'X' in 'nDimX' and that of 'Y' in 'nDimY'
nDimX = np.shape(trainX)[1]
nDimY = np.shape(trainY)[1]

# Construct the neural network using relu
inputs = keras.Input(shape=nDimX, name="input")
nn_layer = keras.layers.Dense(nHidden, activation="relu")(inputs)
outputs = keras.layers.Dense(nDimY, name="output")(nn_layer)

model = keras.Model(inputs=inputs, outputs=outputs)

# Define the loss function (MSE) and the optimizer (AdagradOptimizer).

model.compile(
    optimizer=tf.optimizers.Adagrad(learning_rate=1),
    loss='mean_squared_error'
)

# Train the model

print("Fit model on training data")

history = model.fit(
    trainX,
    trainY,
    batch_size = 50,
    epochs = 1000
)
# Output the accuracy of the regressor on the test data.

predictedY = model.predict(testX)

# Plot the predicted load and compare against the actual load from the test da
assert(nTest >= 16)
days = random.sample(range(nTest), 16)
```

```
fig, axs = plt.subplots(4, 4, sharex=True, sharey=True, figsize=(10,10))
axs = axs.ravel()


for dd, day in enumerate(days):
    testYDay = testY[day]
    predictedYDay = predictedY[day]

    l1 = axs[dd].plot(range(1, 25), testYDay, label='Measured')
    l2 = axs[dd].plot(range(1, 25), predictedYDay, label='Predicted')

    axs[dd].set_ylim(bottom=0, top=75000)
    axs[dd].legend()

fig.text(0.5, 0.07, 'Time of day (in hour)', ha='center')
fig.text(0.04, 0.5, 'Demand in Texas (in MW)', va='center', rotation='vertical

plt.show()
```

```
Epoch 25/1000
6/6 [==============================] - 0s 3ms/step - loss: 54598476.0000
Epoch 26/1000
6/6 [==============================] - 0s 6ms/step - loss: 52175096.0000
Epoch 27/1000
6/6 [==============================] - 0s 2ms/step - loss: 51873428.0000
Epoch 28/1000
6/6 [==============================] - 0s 3ms/step - loss: 51984996.0000
Epoch 29/1000
6/6 [==============================] - 0s 2ms/step - loss: 50300604.0000
Epoch 30/1000
6/6 [==============================] - 0s 3ms/step - loss: 49123688.0000
Epoch 31/1000
6/6 [==============================] - 0s 2ms/step - loss: 48474356.0000
Epoch 32/1000
6/6 [==============================] - 0s 3ms/step - loss: 46869108.0000
Epoch 33/1000
6/6 [==============================] - 0s 3ms/step - loss: 47478484.0000
Epoch 34/1000
6/6 [==============================] - 0s 3ms/step - loss: 46321100.0000
```