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 [1]:
```
import os
import tensorflow as tf
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 [2]:
```
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.
```

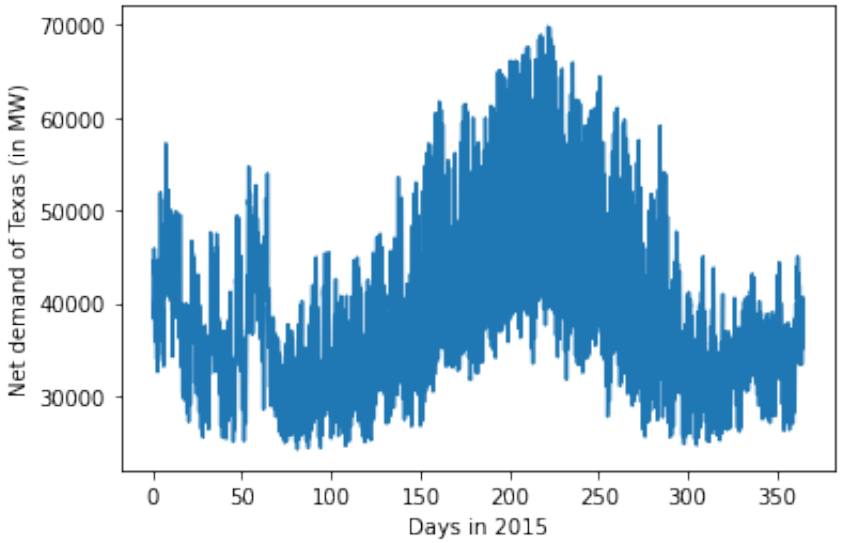
Understand the data.

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 [3]:
```
fig = plt.figure()

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

Out[3]:
```
Text(0, 0.5, 'Net demand of Texas (in MW)')
```



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

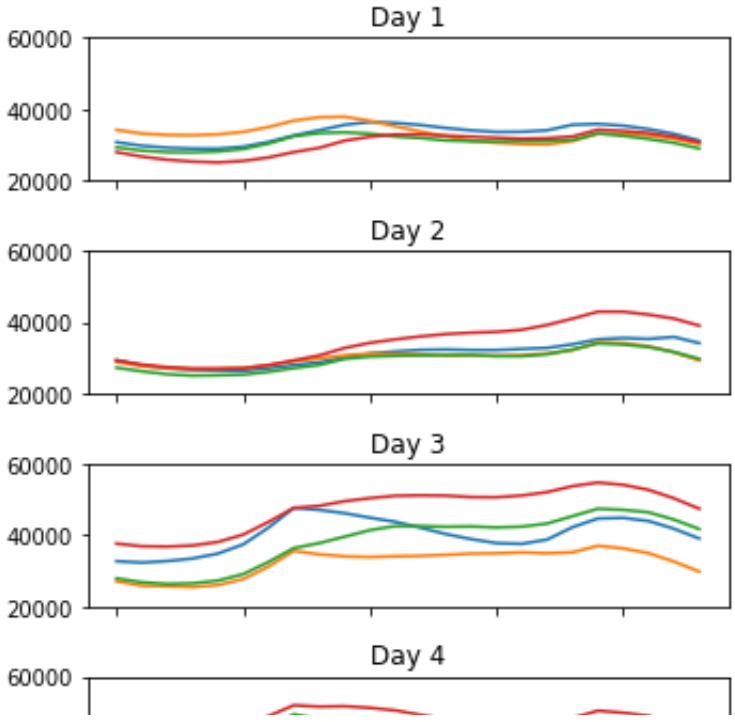
Question (5 points). From the above plot, what can you infer about the climate of Texas? What would you expect if you plotted the same in Illinois?

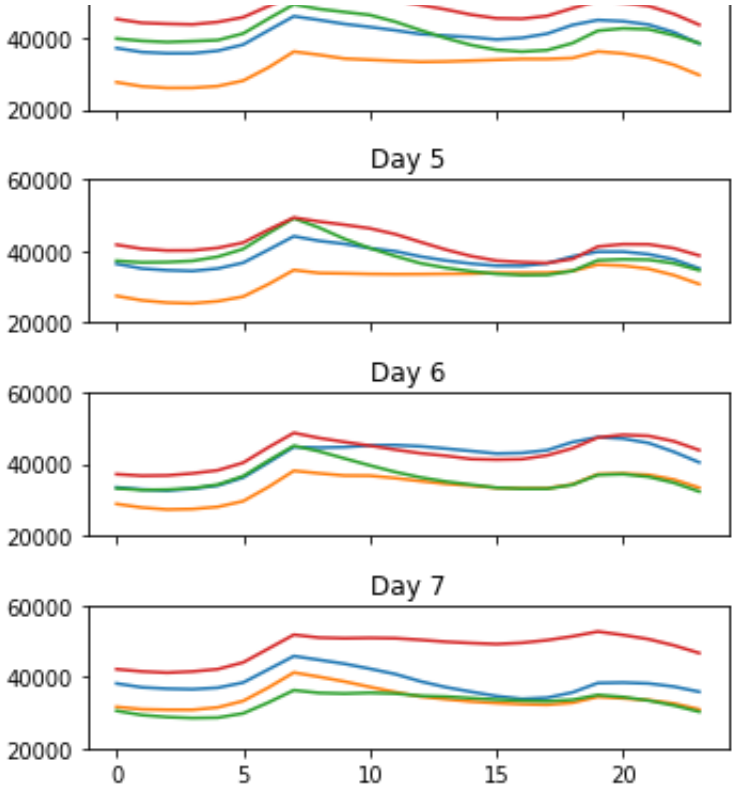
Your answer. As you can see above plot, climate of Texas is changes along the season such that between 150 days and 250 days, which is about summer season, is frequently hot as expected which results the dnet demand of Texas increases. I would expect pretty much same result based on the plot in Illinois, since as far as I know, we share very similar climate change.

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

In [4]:
```
# Plot the load data of the same day of the week over several weeks.

dayStart = 30
numberOfWeeks = 4

DayOfWeek = ['Monday','Tuesday','Wednesday','Thursday','Friday','Satur
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
                                                         (dayStart + 7
                                                         'ERCOT Load,
    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.
```





Question (5 points). Can you find any discernible change in the load profiles of different days of the week?

Your answer. As you can see, Day 3 which is Monday until Day 7 which is Friday, we see pretty much similar shapes of graph plot that the load profiles imply similar patterns of usage. However, Day 1 and 2 which are Saturday and Sunday, weekend days, the graph look quite different from weekdays. The discernible change in the load profiles is shown between Day 2 and Day 3 when the weekend ends and the week days start.

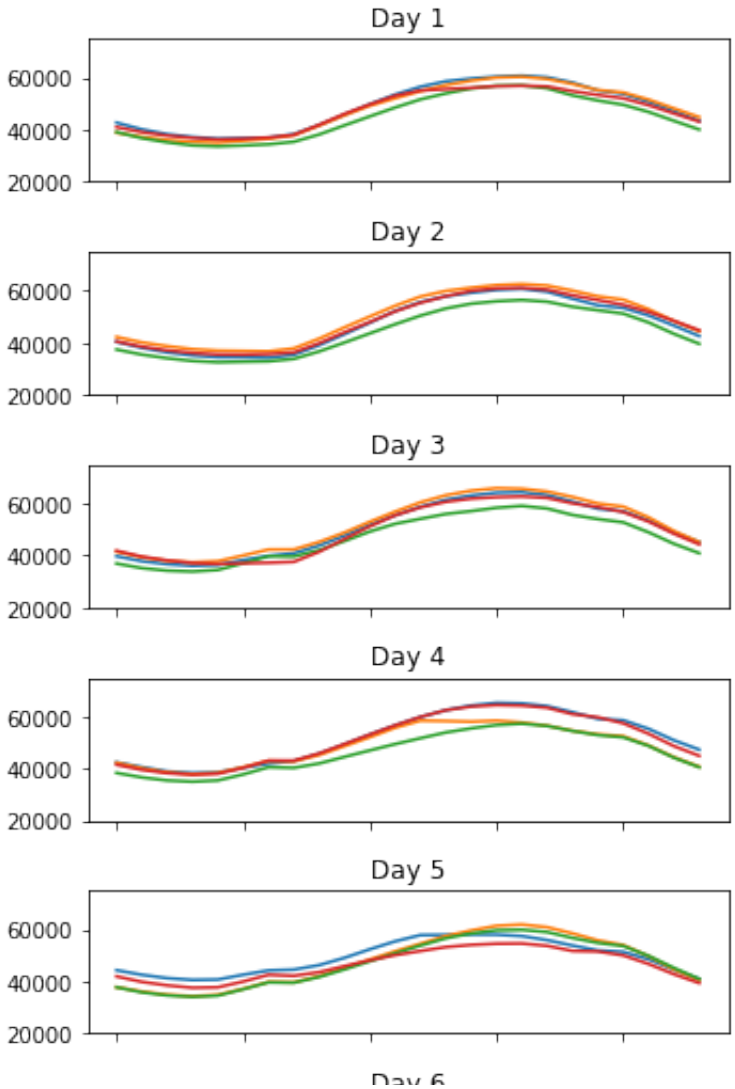
Question (15 points). Redo the above exercise for the months of August and September. Make 'Day 1' correspond to August 15th. What do you observe differently? Do your observations agree with Q1?

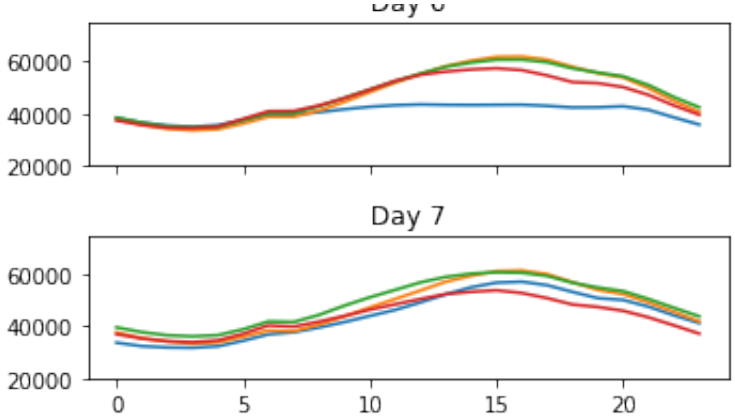
Your answer (comments here, code below). Yes, it still shows the similar pattern as we have observed in Q1.

In [40]:
```
# Modify the following code

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

dayStart = 226
numberOfWeeks = 4

DayOfWeek = ['Monday','Tuesday','Wednesday','Thursday','Friday','Satur
```

```
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
                                                         (dayStart + 7
                                                         'ERCOT Load,
    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.
```





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 [6]:
```
daysToTrainOn = [-7, -2, -1]
rangeOfDays = range(-np.min(daysToTrainOn), numberOfDays)

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

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 [7]:
```
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)

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
```

```
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.

Recall that TensorFlow defines a computation graph where the weights and biases associated with the NN are variables. The goal is to optimize the weights and biases of the NN to minimize prediction error using data.

To define the computation graph, create the inputs and outputs as 'placeholders'. The algorithm only expects them to be specified at the time of computation. The first element of the shape attribute for both inputs and outputs are 'None'. This means that they are left unspecified, and will be provided at runtime. It will help in batch training for prediction, where the size of the batch will determine this value. Batch training is useful because training the NN with one data point at a time can be time consuming.

In this lab, we begin with a 'relu' activation. We additionally implement 'dropouts' that basically prevents certain paramters from updating in each round. This is known to prevent overfitting. The number'0.995' in the description below updates 99.5% of all weights, leaving out 0.5%.

Design the optimizer and the loss. For reporting the accuracy of prediction, we choose in this lab the idea of mean absolute error (MAE). For a data set, if the true values are scalars $y_1, \dots, y_m$ and the predictions are $\hat{y}_1, \dots, \hat{y}_m$, then its MAE is given by

$$MAE = \frac{1}{m} \sum |y_i - \hat{y}_i|.$$

If y and $\hat{y}$ are multidimensional, it computes the average across each coordinate of y and $\hat{y}$.

Question (5 points). Insert a line of code for the output of layer 1 below (use the relu function)

In [47]:
```
# Insert your code where specified
nHidden = 150

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

# Define the weights, biases of the first layer.

# insert code
#tf.compat.v1.disable_eager_execution()
#inputNN = tf.compat.v1.placeholder(tf.float32, shape=[None, nDimX])
outputNN = tf.compat.v1.placeholder(tf.float32, shape=[None, nDimY])

W1 = tf.Variable(tf.random.truncated_normal(shape=[nDimX, nHidden]))
b1 = tf.Variable(tf.random.truncated_normal(shape=[nHidden]))
```

```
# Define the weights, biases of the second layer.

# insert code
W2 = tf.Variable(tf.random.truncated_normal(shape=[nHidden, nDimY]))
b2 = tf.Variable(tf.random.truncated_normal(shape=[nDimY]))

# Construct the neural network
@tf.function
def neuralNetworkModel(X):
    global W1, b1, W2, b2

    # Define the output of layer 1 (Z1 as a function of the input usin

    # insert code
    Z1 = tf.nn.relu(tf.matmul(X, W1) + b1)
    print(Z1)
    outputLayer1 = tf.nn.dropout(Z1, 0.005)
    print(outputLayer1)
    # Define the output of layer 2 (Z2 as a function of outputLayer1 u

    # insert code
    Z2 = tf.nn.relu(tf.matmul(outputLayer1, W2) + b2)
    print(Z2)

    outputLayer2 = tf.nn.dropout(Z2, 0.005)
    print(outputLayer2)

    return outputLayer2

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

# insert code
loss_fn = tf.losses.MeanSquaredError()
optimizer = tf.keras.optimizers.Adagrad(learning_rate=0.4)

# Define the metric (MAE - mean absolute error)

# insert code
mae = tf.metrics.MeanAbsoluteError()
```

Train the neural network.

Create the training module for the NN. Feed the training data in batches of size 'batchSize' and ask Tensorflow to run the function 'optimizer'. The number of batches, denoted by 'nBatches' is then given by the size of your training dataset divided by '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!

In [48]:
```
batchSize = 50
nBatches = int(nTrain/batchSize)
nEpochs = 2000

# Define the training scheme
def train(model, trainX, trainY):

    for epoch in range(nEpochs):
        lossEpoch = 0

        # In each epoch, use 'optimizer' to reduce the 'loss' over bat
        for n in range(nBatches):

            # Define the batch to train on.
            batchX = trainX[n * batchSize: (n + 1) * batchSize]
            batchY = trainY[n * batchSize: (n + 1) * batchSize]

            # Fit the batch data and compute the gradients
            with tf.GradientTape() as tape:
                prediction = model(batchX)
                loss = loss_fn(y_true=batchY, y_pred=prediction)

                # Print update
                lossEpoch = lossEpoch + loss.numpy()

                # Optimize the weights
                gradients = tape.gradient(loss, trainable_variables)
                optimizer.apply_gradients(zip(gradients, trainable_var

        print("Epoch: %d, Loss: = %1.1f" % (epoch + 1, lossEpoch))


print("Started the training module.")

# insert code

train(neuralNetworkModel, trainX, trainY)

print("Training process completed.")
```

```
print("Training process completed.")
```

```
5,
       34158, 36936, 37137, 36398, 34725, 32272]), array([27896, 26
809, 26323, 26500, 27262, 29075, 32482, 36372, 37756,
       39554, 41396, 42526, 42534, 42340, 42429, 42146, 42376, 4324
2,
       45314, 47382, 47084, 46387, 44363, 41708]), array([28886, 27
700, 27159, 26786, 26901, 27187, 27939, 29049, 29847,
       30742, 31219, 31142, 31024, 30597, 30671, 30608, 30770, 3127
2,
       32136, 34447, 34162, 33287, 31582, 29341]), array([43153, 40
742, 39028, 38206, 38348, 39738, 40656, 42719, 46086,
       50022, 54091, 57732, 60915, 63480, 65341, 66027, 66138, 6561
0,
       63925, 60857, 59299, 55984, 51643, 47187]), array([43425, 42
876, 43022, 43541, 44626, 46748, 50759, 53979, 51445,
       48389, 45992, 43753, 41378, 39422, 37789, 36190, 35449, 3522
1,
       36224, 39468, 40323, 40731, 40158, 38739]), array([35949, 34
581, 33939, 33723, 33935, 35415, 38818, 42030, 40843,
```

Test the accuracy of prediction via NN.

Here, you report the mean absolute error of your predictions over the 'testX' dataset. Finally, plot the actual demand profile versus the predicted demand profile for a few days from the test data.

In [49]:
```
    predictedY = neuralNetworkModel(testX)
    mae.update_state(y_true=testY, y_pred=predictedY)
    maeOfPrediction = mae.result().numpy()
    print("Mean absolute error of forecast = ", maeOfPrediction)
```

```
       37914, 36602, 35547, 34889, 34456, 34084, 33862, 33936, 3424
6,
       37383, 37849, 37493, 36985, 35111, 32775]), array([43683, 41
372, 39952, 39127, 39346, 40732, 41717, 43569, 46899,
       50686, 54585, 58031, 60993, 63577, 65472, 66141, 66001, 6546
6,
       63402, 60511, 58952, 55720, 51494, 47149]), array([41810, 39
453, 37838, 36907, 37102, 38461, 39389, 41528, 44589,
       48143, 51940, 55372, 58331, 60974, 62733, 64198, 64672, 6388
9,
       61827, 58960, 57172, 53972, 49610, 45138]), array([29473, 28
205, 27389, 27158, 27877, 30409, 33851, 33630, 34157,
       34719, 35080, 34988, 35036, 35205, 35454, 35878, 36205, 3633
0,
       35984, 36592, 36893, 34846, 31614, 29098]), array([36593, 34
406, 32776, 31730, 31252, 31642, 31704, 34488, 38216,
       41962, 45700, 48770, 51033, 52825, 54165, 55101, 55427, 5474
9,
       53037, 50552, 49305, 46890, 43210, 39652]), array([36540, 34
```

Question (5 points). Comment whether your MAE is high or low.

Hint. Compare the mean absolute error to the maximum demands.

Your answer.

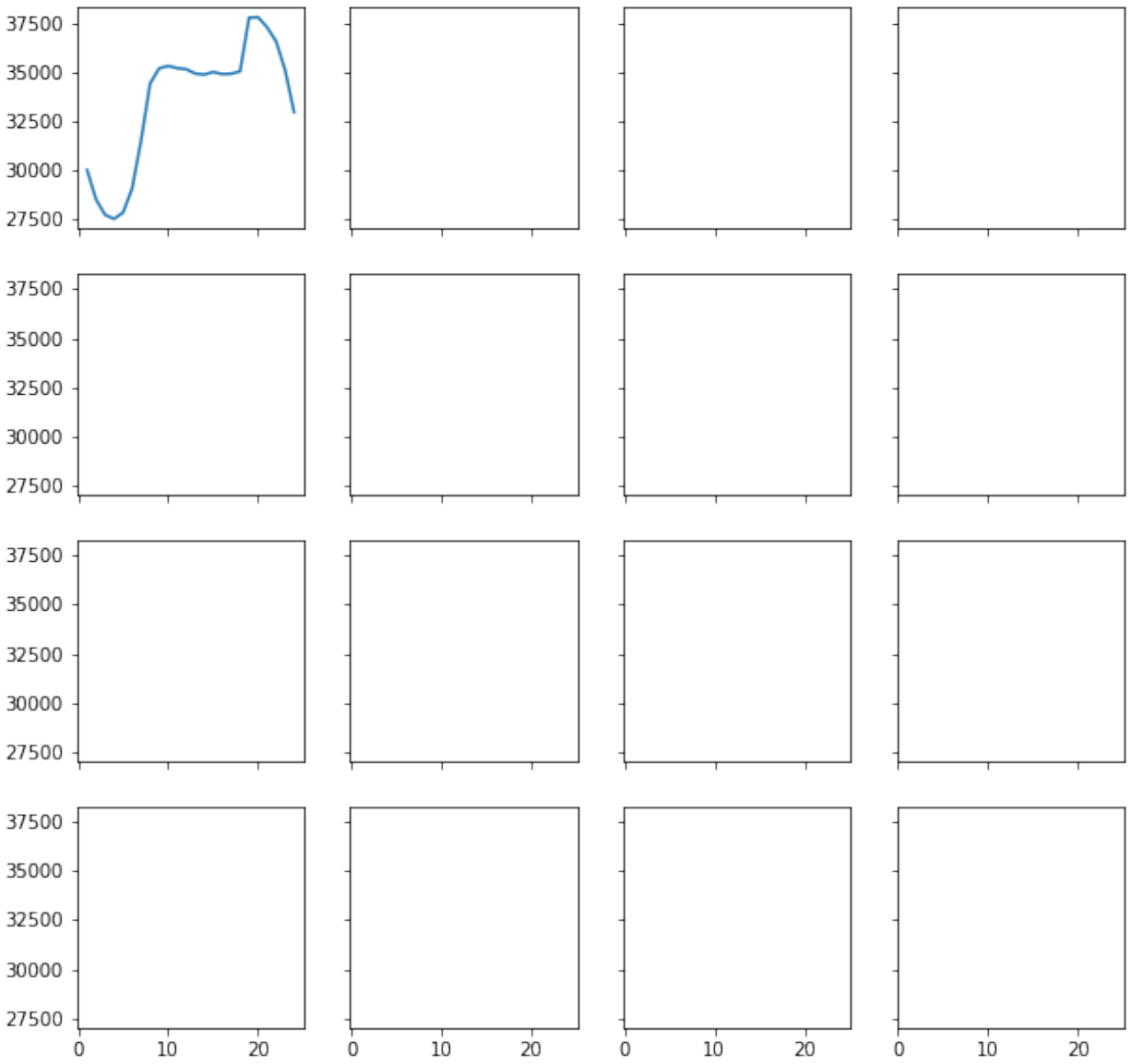
Let us visualize the results.

In [50]:
```
    # Plot the predicted load and compare against the actual load from
    assert(nTest >= 16)
    days = random.sample(range(nTest), 16)

    fig, axs = plt.subplots(4, 4, sharex=True, sharey=True, figsize=(1
    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='Predicte
```

```
        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', rotati

    plt.show()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-50-e98407a627c8> in <module>
     11
     12     l1 = axs[dd].plot(range(1, 25), testYDay, label='Measured')
---> 13     l2 = axs[dd].plot(range(1, 25), predictedYDay, label='Predicted')
     14
     15     axs[dd].set_ylim(bottom=0, top=75000)

/opt/anaconda3/envs/myenv/lib/python3.7/site-packages/matplotlib/axes/_axes.py in plot(self, scalex, scaley, data, *args, **kwargs)
   1741         """
   1742         kwargs = cbook.normalize_kwargs(kwargs, mlines.Line2D)
-> 1743         lines = [*self._get_lines(*args, data=data, **kwargs)]
   1744         for line in lines:
   1745             self.add_line(line)

/opt/anaconda3/envs/myenv/lib/python3.7/site-packages/matplotlib/axes/_base.py in __call__(self, data, *args, **kwargs)
    271                 this += args[0],
    272                 args = args[1:]
--> 273             yield from self._plot_args(this, kwargs)
    274
    275     def get_next_color(self):

/opt/anaconda3/envs/myenv/lib/python3.7/site-packages/matplotlib/axes/_base.py in _plot_args(self, tup, kwargs)
    387         if len(tup) == 2:
    388             x = _check_1d(tup[0])
--> 389             y = _check_1d(tup[-1])
    390         else:
    391             x, y = index_of(tup[-1])

/opt/anaconda3/envs/myenv/lib/python3.7/site-packages/matplotlib/cbook/__init__.py in _check_1d(x)
   1316                     message='Support for multi-dimensional in
```

```
dexing')
   1317
-> 1318                 ndim = x[:, None].ndim
   1319                 # we have definitely hit a pandas index or se
ries object
   1320                 # cast to a numpy array.

AttributeError: 'Tensor' object has no attribute 'ndim'
```



Question (20 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.

Question (20 points). Fix the number of echos to your favorite one, and then explore how the number of neurons affects the accuracy and speed of training. Start with 6 , and increase it to 12, 24, 48, 100, and more. Make comments based on your observations. As an engineer, what is your favorite number of neurons, and why?

Your answer.

Question (30 points). Fix the number of epochs and neurons to your favorite ones. Then, add another layer to the network. Discuss what your observe in terms of speed and accuracy. If the training becomes too slow, you may alter the number of epochs/neurons.

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

In [ ]:

The effect of wind energy (bonus).

In [25]:
```
#Let's check the raw data
dfDemand = pd.read_csv("ERCOT_Hourly_Wind_Output_" + str(year) + ".csv
dfDemand[:]
```

Out[25]:

	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? Write down your MAE before and after considering wind energy.

Your answer (comments here, code below).

In [ ]:
```

```