ILLINOIS INSTITUTE OF TECHNOLOGY

College of Science

Neural NILM: Deep Neural Networks Applied to Energy Disaggregation

Presented by:
Yuzhe Lim

What is the usage of NILM?

NILM (also called energy disaggregation) is a computational technique for estimating the power demand of individual appliances from a single meter which measures the combined demand of multiple appliances

help operators to manage the grid

to identify faulty appliance

to survey appliance usage behaviour

help users reduce their energy consumption

Data-preprocessing

Since machine learning (or deep learning) is data-driven, data-preprocessing is one of the essential parts.

Data is recorded separately and each channel represent an independent appliance

The 5 appliances used are kettle, fridge, microwave, dishwasher and washing machine, alongside the channel which represents the aggregate power data.

Work done

Accessing and processing data (existing tools for ingesting NILM data)	Adapting python code and existing model architectures to this data (data in, class label out)
• Importing data and Synchronizing • Extract activations • Generate real sample • Standardization of dataset	• Resnet model (residual neural network)

Importing data and Synchronizing

CONCATENATE ALL 6 CHANNELS MENTIONED INTO A SINGLE DATA FRAME.

TIMESTAMP IS ALIGNED USING FORWARD-FILLING AND ZERO FILLING TO MAKE IT IN A FREQUENCY OF 6 SEC.

DATA VISUALIZATION WAS PERFORMED TO CHECK THE ACCURACY OF RESULT AND TO UNDERSTAND THE DATA

Merge Method

```python
import glob
import pandas as pd
import time

path =r'C:/Users/Yu Zhe/Desktop/Presentation examples/' # Path to dataset
allFiles = glob.glob(path + "/*.dat") #Data file name
appliances = ['Kettle', 'Microwave', 'Laptop', 'TV'] #Array of appliances
i = 0 #Counter for appliances array

#Initialize the frame with timestamp (TS) as Index column
frame1 = pd.DataFrame(columns=['TS'])

#Loop to load dataset into a single frame
for file_ in allFiles:

    start = time.time()
    df = pd.read_csv(file_,delimiter = ' ', names = ['TS', file_[46:55]])
    print("Reading", i, " passed")
    frame1 = frame1.merge(df, on='TS', how='outer', sort =True)
    print("Merging", i, " passed")

    end = time.time()
    print("Time used: ", end - start)
    i = i+1

end = int(frame1['TS'][len(frame1['TS'])-1])
start = int(frame1['TS'][0])
frame1.set_index('TS')
print(frame1)
```

```
Reading 0  passed
Merging 0  passed
Time used:  0.013447999954223633
Reading 1  passed
Merging 1  passed
Time used:  0.015265703201293945
Reading 2  passed
Merging 2  passed
Time used:  0.009664058685302734
Reading 3  passed
Merging 3  passed
Time used:  0.011224746704101562
            TS  channel_1  channel_5  channel_6  channel_8
0   1303132929     224.19        NaN        NaN        NaN
1   1303132930     225.57        NaN        NaN        NaN
2   1303132931     226.09        NaN        NaN        NaN
3   1303132932     222.74        NaN        NaN        NaN
4   1303132933     222.20        6.0        0.0       21.0
5   1303132934     222.11        NaN        NaN        NaN
6   1303132935     223.14        NaN        NaN        NaN
7   1303132936     223.17        6.0        0.0       21.0
8   1303132937     222.25        NaN        NaN        NaN
9   1303132938     222.64        NaN        NaN        NaN
10  1303132939     221.88        NaN        NaN        NaN
11  1303132940     223.60        6.0        0.0       22.0
12  1303132941     222.21        NaN        NaN        NaN
13  1303132942     222.82        NaN        NaN        NaN
14  1303132943     222.91        6.0        1.0       21.0
15  1303132944     222.81        NaN        NaN        NaN
16  1303132945     221.64        NaN        NaN        NaN
17  1303132946     222.94        6.0        0.0       21.0
18  1303132947     222.43        NaN        NaN        NaN
19  1303132948     221.98        NaN        NaN        NaN
20  1303132949     222.30        NaN        NaN        NaN
21  1303132950     222.90        6.0        0.0       21.0
22  1303132951     222.56        NaN        NaN        NaN
```

Join Method (Lesser time consumption)

```python
import glob
import pandas as pd

path =r'C:/Users/Yu Zhe/Desktop/Presentation examples/' # Path to dataset
allFiles = glob.glob(path + "/*.dat") #Data file name
i = 0 #Counter for appliances array

#Initialize the frame with timestamp (TS) as Index column
frame1 = pd.DataFrame()
#frame1.set_index('TS')

#Loop to load dataset into a single frame
for file_ in allFiles:
    start = time.time()
    df = pd.read_csv(file_,delimiter = ' ', names = ['TS', file_[46:55]],
                     header = None)#appliances[i]])

    print("Reading", i, " passed")
    frame1 = frame1.join(df.set_index('TS'), how='outer')
    print("Joining", i, " passed")

    end = time.time()
    print("Time used: ", end - start)

    i = i+1


#df.set_index('TS')
print(frame1)
```

```
Reading 0  passed
Joining 0  passed
Time used:  0.006638288497924805
Reading 1  passed
Joining 1  passed
Time used:  0.009936094284057617
Reading 2  passed
Joining 2  passed
Time used:  0.010226249694824219
Reading 3  passed
Joining 3  passed
Time used:  0.009761333465576172
            channel_1  channel_5  channel_6  channel_8
TS
1303132929     224.19        NaN        NaN        NaN
1303132930     225.57        NaN        NaN        NaN
1303132931     226.09        NaN        NaN        NaN
1303132932     222.74        NaN        NaN        NaN
1303132933     222.20        6.0        0.0       21.0
1303132934     222.11        NaN        NaN        NaN
1303132935     223.14        NaN        NaN        NaN
1303132936     223.17        6.0        0.0       21.0
1303132937     222.25        NaN        NaN        NaN
1303132938     222.64        NaN        NaN        NaN
1303132939     221.88        NaN        NaN        NaN
1303132940     223.60        6.0        0.0       22.0
1303132941     222.21        NaN        NaN        NaN
1303132942     222.82        NaN        NaN        NaN
1303132943     222.91        6.0        1.0       21.0
1303132944     222.81        NaN        NaN        NaN
1303132945     221.64        NaN        NaN        NaN
1303132946     222.94        6.0        0.0       21.0
1303132947     222.43        NaN        NaN        NaN
1303132948     221.98        NaN        NaN        NaN
1303132949     222.30        NaN        NaN        NaN
```

Extract activations

Dataset source:

REDD (Reference Energy Disaggregation Dataset)	UK-DALE (UK Domestic Appliance Level Electricity)

Definition of ‘appliance activation’: The power drawn by a single appliance over one complete cycle of appliance

Raw Data

1303132974	0.001303132978	0.001303132981	0.001303132984	0.001303132988	0.001303133002	0.001303133006	0.001303133009	0.001303133013
1303133027	0.001303133030	0.001303133034	0.001303133037	0.001303133040	0.001303133044	0.001303133048	0.001303133056	0.001303133060
1303133074	0.001303133077	0.001303133081	0.001303133084	0.001303133088	0.001303133091	0.001303133095	0.001303133098	0.001303133101
1303133120	0.001303133124	0.001303133127	0.001303133131	0.001303133134	0.001303133137	0.001303133141	0.001303133144	0.001303133148
1303133162	0.001303133165	0.001303133169	0.001303133177	0.001303133181	0.001303133184	0.001303133188	0.001303133191	0.001303133195
1303133208	0.001303133212	0.001303133215	0.001303133219	0.001303133222	0.001303133226	0.001303133229	0.001303133238	0.001303133241
1303133255	0.001303133259	0.001303133262	0.001303133266	0.001303133269	0.001303133272	0.001303133276	0.001303133279	0.001303133283
1303133302	0.001303133305	0.001303133309	0.001303133312	0.001303133316	0.001303133319	0.001303133323	0.001303133326	0.001303133330
1303133343	0.001303133347	0.001303133356	0.001303133359	0.001303133363	0.001303133366	0.001303133369	0.001303133373	0.001303133376
1303133390	0.001303133393	0.001303133397	0.001303133400	0.001303133404	0.001303133407	0.001303133416	0.001303133420	0.001303133423
1303133437	0.001303133440	0.001303133444	0.001303133447	0.001303133451	0.001303133454	0.001303133458	0.001303133461	0.001303133465
1303133483	0.001303133487	0.001303133490	0.001303133494	0.001303133497	0.001303133501	0.001303133504	0.001303133508	0.001303133511
1303133525	0.001303133528	0.001303133537	0.001303133540	0.001303133544	0.001303133547	0.001303133551	0.001303133554	0.001303133558
1303133572	0.001303133575	0.001303133578	0.001303133582	0.001303133585	0.001303133589	0.001303133597	0.001303133601	0.001303133604
1303133618	0.001303133622	0.001303133625	0.001303133628	0.001303133632	0.001303133635	0.001303133639	0.001303133642	0.001303133646
1303133665	0.001303133668	0.001303133672	0.001303133675	0.001303133679	0.001303133682	0.001303133686	0.001303133689	0.001303133693
1303133707	0.001303133716	0.001303133719	0.001303133722	0.001303133726	0.001303133729	0.001303133733	0.001303133736	0.001303133740
1303133754	0.001303133757	0.001303133761	0.001303133764	0.001303133768	0.001303133776	0.001303133780	0.001303133783	0.001303133787
1303133801	0.001303133804	0.001303133807	0.001303133811	0.001303133814	0.001303133818	0.001303133821	0.001303133825	0.001303133828
1303133848	0.001303133851	0.001303133855	0.001303133858	0.001303133862	0.001303133865	0.001303133869	0.001303133872	0.001303133875
1303133889	0.001303133898	0.001303133902	0.001303133905	0.001303133908	0.001303133912	0.001303133915	0.001303133919	0.001303133922
1303133936	0.001303133939	0.001303133943	0.001303133946	0.001303133955	0.001303133958	0.001303133962	0.001303133965	0.001303133969
1303133983	0.001303133986	0.001303133990	0.001303133993	0.001303133997	0.001303134000	0.001303134003	0.001303134007	0.001303134016
1303134030	0.001303134034	0.001303134038	0.001303134041	0.001303134045	0.001303134049	0.001303134052	0.001303134056	0.001303134060
1303134080	0.001303134084	0.001303134087	0.001303134091	0.001303134095	0.001303134098	0.001303134102	0.001303134106	0.001303134110
1303134124	0.001303134128	0.001303134137	0.001303134140	0.001303134144	0.001303134147	0.001303134151	0.001303134154	0.001303134158
1303134173	0.001303134176	0.001303134180	0.001303134184	0.001303134188	0.001303134196	0.001303134200	0.001303134203	0.001303134207
1303134220	0.001303134224	0.001303134227	0.001303134231	0.001303134235	0.001303134238	0.001303134241	0.001303134245	0.001303134248
1303134268	0.001303134271	0.001303134275	0.001303134278	0.001303134282	0.001303134285	0.001303134288	0.001303134292	0.001303134295
1303134309	0.001303134318	0.001303134321	0.001303134325	0.001303134328	0.001303134332	0.001303134335	0.001303134339	0.001303134342
1303134356	0.001303134359	0.001303134363	0.001303134366	0.001303134375	0.001303134379	0.001303134382	0.001303134386	0.001303134389

Timestamp is aligned in a frequency of 6 sec.

→ All houses record aggregate apparent mains power once every 6 seconds whereas the active and reactive mains power were recorded once a second (voltage and current)
→ The 1 second active mains power is downsampled to 6 seconds to align with the submetered data and used as the real aggregate data from these houses

```
#Set the index into a frequency of 6 seconds
import numpy as np

end = int(frame1.index[len(frame1)-1])
start = int(frame1.index[0])
print("start",start)
print("end",end)
l = [np.int64(i) for i in np.arange(start,end+6,6)]

print("Index:", l)
nframe1 = frame1.reindex(l)
nframe1
```

TS	channel_1	channel_5	channel_6	channel_8
1303132929	224.19	NaN	NaN	NaN
1303132935	223.14	NaN	NaN	NaN
1303132941	222.21	NaN	NaN	NaN
1303132947	222.43	NaN	NaN	NaN
1303132953	222.96	6.0	0.0	21.0
1303132959	225.66	NaN	NaN	NaN
1303132965	223.43	NaN	NaN	NaN
1303132971	227.29	6.0	1.0	21.0
1303132977	225.22	NaN	NaN	NaN
1303132983	226.28	NaN	NaN	NaN
1303132989	225.78	NaN	NaN	NaN
1303132995	226.56	NaN	NaN	NaN
1303133001	225.16	NaN	NaN	NaN
1303133007	225.87	NaN	NaN	NaN
1303133013	225.72	6.0	0.0	22.0
1303133019	NaN	NaN	NaN	NaN
1303133025	NaN	NaN	NaN	NaN

Forward-filling

Any gaps in appliance data shorter than 3 minu
are assumed to be due to RF (Radio frequency)
issues and so are filled by forward-filling

In [30]:
```
frame1.fillna(method='ffill')
```

Out[30]:

TS	channel_1	channel_5	channel_6	channel_8
1303132929	224.19	NaN	NaN	NaN
1303132930	225.57	NaN	NaN	NaN
1303132931	226.09	NaN	NaN	NaN
1303132932	222.74	NaN	NaN	NaN
1303132933	222.20	6.0	0.0	21.0
1303132934	222.11	6.0	0.0	21.0
1303132935	223.14	6.0	0.0	21.0
1303132936	223.17	6.0	0.0	21.0
1303132937	222.25	6.0	0.0	21.0
1303132938	222.64	6.0	0.0	21.0
1303132939	221.88	6.0	0.0	21.0
1303132940	223.60	6.0	0.0	22.0
1303132941	222.21	6.0	0.0	22.0
1303132942	222.82	6.0	0.0	22.0
1303132943	222.91	6.0	1.0	21.0
1303132944	222.81	6.0	1.0	21.0

Zero-filling

Any gaps longer than 3 minutes are assumed to be due to the appliance and meter being switched off and so are filled with zeros.

In [37]:
```
frame1.fillna(0)
```

Out[37]:

TS	channel_1	channel_5	channel_6	channel_8
1303132929	224.19	0.0	0.0	0.0
1303132930	225.57	0.0	0.0	0.0
1303132931	226.09	0.0	0.0	0.0
1303132932	222.74	0.0	0.0	0.0
1303132933	222.20	6.0	0.0	21.0
1303132934	222.11	6.0	0.0	21.0
1303132935	223.14	6.0	0.0	21.0
1303132936	223.17	6.0	0.0	21.0
1303132937	222.25	6.0	0.0	21.0
1303132938	222.64	6.0	0.0	21.0
1303132939	221.88	6.0	0.0	21.0
1303132940	223.60	6.0	0.0	22.0
1303132941	222.21	6.0	0.0	22.0
1303132942	222.82	6.0	0.0	22.0
1303132943	222.91	6.0	1.0	21.0
1303132944	222.81	6.0	1.0	21.0
1303132945	221.64	6.0	1.0	21.0

```python
def load_data(appliance_name, type='default'):
    if appliance_name == 'kettle':
        appliance_name = 'channel_5'
    path = 'dataset/'+appliance_name+ ".dat" # Path to dataset
    frame1 = pd.DataFrame()

    title = ['timestamp','appliance_power']
    df = pd.read_csv("dataset/channel_1.dat",sep = ' ', header = None, float_precision='round_trip', names=['timestamp','aggregate_power'])
    df2 = pd.read_csv(path, delimiter=' ', header=None, float_precision='round_trip', names=title)  # appliances[i]])

    frame1 = frame1.join(df.set_index('timestamp'), how='outer')
    frame1 = frame1.join(df2.set_index('timestamp'), how='outer')

    # Set the index into a frequency of 6 seconds
    end = int(frame1.index[len(frame1) - 1])
    start = int(frame1.index[0])
    print("start", start)
    print("end", end)

    l = [np.int64(i) for i in np.arange(start, end + 6, 6)]
    frame1 = frame1.reindex(l)

    # Forward filling
    frame1 = frame1.fillna(method='ffill')
    # Backward filling
    frame1 = frame1.fillna(method='bfill')
    frame1 = frame1.reset_index()
    columnsTitles = ["aggregate_power", "appliance_power", "timestamp"]
    frame1 = frame1.reindex(columns=columnsTitles)
    #print(frame1)
    return frame1
```

Extract activations

- Activations datapoints were extracted by finding strictly consecutive samples above the threshold power listed in the table.
- Any activations shorter than some threshold duration is then thrown away (to ignore spurious spikes).
- For more complex appliances such as washing machines whose power demand can drop below threshold for short periods during a cycle, NILMTK ignores short periods of sub-threshold power demand

Appliance	Max power (watts)	On power threshold (watts)	Min. on duration (secs)	Min. off duration (secs)
Kettle	3100	2000	12	0
Fridge	300	50	60	12
Washing m.	2500	20	1800	160
Microwave	3000	200	12	30
Dish washer	2500	10	1800	1800

Process of finding activation and non activation data points

Each set consist of data consist of 128 data points with continuous timestamp

Activation point dataset was made up of 64 data points before the activation point and 63 data points, including the activation point itself.

The other data point sets are activated(power) and non activated data points.

```python
def save_activation(appliance_name):
    df = load_data(appliance_name)
    df['timestamp'].astype(np.int64)
    df.insert(3, "end_time", df['timestamp'], True)
    df.insert(4, "index_house", 1, True)
    df.insert(5, "name_appliance", 'microwave', True)

    df.rename(columns={'timestamp':'start_time'}, inplace=True)
    columnsTitles = ["start_time", "end_time", "aggregate_power", "appliance_power", "index_house", "name_appliance"]
    df = df.reindex(columns=columnsTitles)

    #Disable warning
    pd.options.mode.chained_assignment = None  # default='warn'

    start = time.time()
    row_count = df.shape[0]
    for i in range (row_count):
        j =df['end_time'][i]
        df['end_time'][i] = j+6
    end = time.time()

    print("Time used: ", end - start)
    print(df)
    path2 = r'dataset/'  # Path to dataset
    df.to_csv(path2 + appliance_name + '_activation.csv', sep=",", index=False)
```

Time used: 298.92010951042175

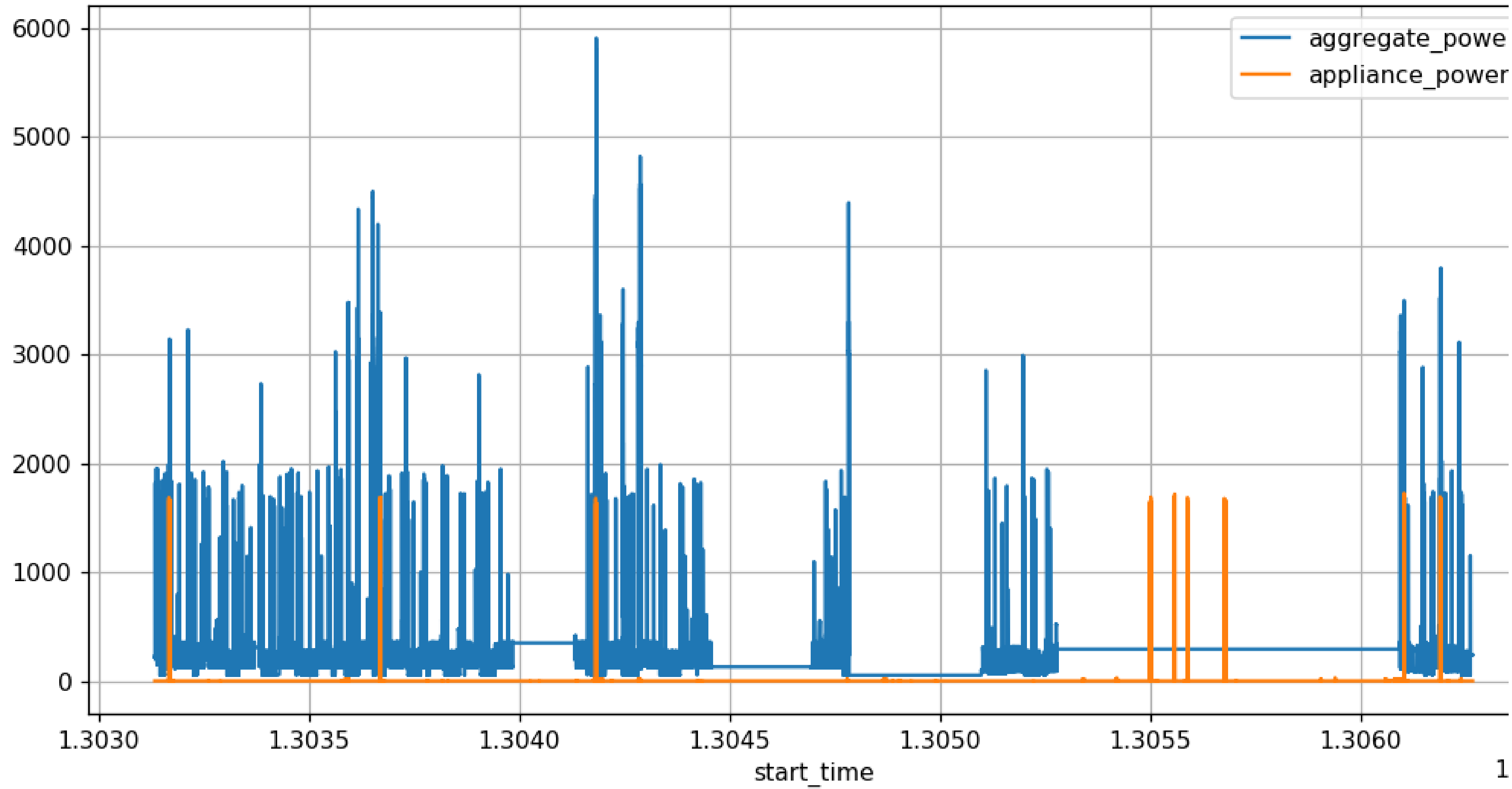
	start_time	end_time	aggregate_power	appliance_power	index_house	name_appliance
0	1303132929	1303132935	224.19	0.0	1	oven
1	1303132935	1303132941	223.14	0.0	1	oven
2	1303132941	1303132947	222.21	0.0	1	oven
3	1303132947	1303132953	222.43	0.0	1	oven
4	1303132953	1303132959	222.96	0.0	1	oven
5	1303132959	1303132965	225.66	0.0	1	oven
6	1303132965	1303132971	223.43	0.0	1	oven
7	1303132971	1303132977	227.29	0.0	1	oven
8	1303132977	1303132983	225.22	0.0	1	oven
9	1303132983	1303132989	226.28	0.0	1	oven
10	1303132989	1303132995	225.78	0.0	1	oven
11	1303132995	1303133001	226.56	0.0	1	oven
12	1303133001	1303133007	225.16	0.0	1	oven
13	1303133007	1303133013	225.87	0.0	1	oven
14	1303133013	1303133019	225.72	0.0	1	oven
15	1303133019	1303133025	226.24	0.0	1	oven
16	1303133025	1303133031	227.60	0.0	1	oven
17	1303133031	1303133037	227.07	0.0	1	oven
18	1303133037	1303133043	225.21	0.0	1	oven
19	1303133043	1303133049	227.14	0.0	1	oven
20	1303133049	1303133055	226.11	0.0	1	oven
21	1303133055	1303133061	222.22	0.0	1	oven
22	1303133061	1303133067	222.75	0.0	1	oven
23	1303133067	1303133073	222.94	0.0	1	oven
24	1303133073	1303133079	223.46	0.0	1	oven
25	1303133079	1303133085	224.04	0.0	1	oven
26	1303133085	1303133091	222.48	0.0	1	oven
27	1303133091	1303133097	221.27	0.0	1	oven
28	1303133097	1303133103	221.26	0.0	1	oven
29	1303133103	1303133109	221.22	0.0	1	oven
...	...	...	...	...	...	...
522320	1306266849	1306266855	239.07	0.0	1	oven
522321	1306266855	1306266861	239.07	0.0	1	oven
522322	1306266861	1306266867	239.07	0.0	1	oven

Load Activation plot

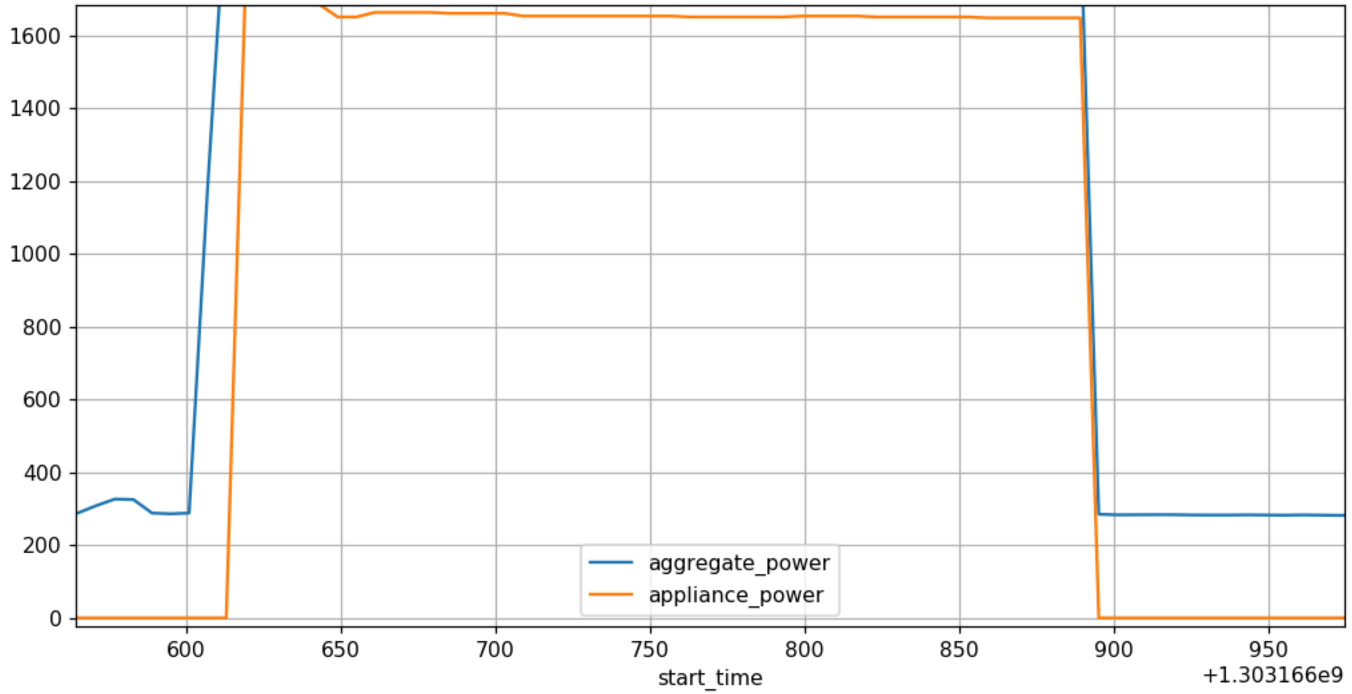
In [29]:
```
%matplotlib notebook
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv("D:/Users/Yu Zhe/Source/Repos/nilm/Yuzhe/dataset/channel_3_activation.csv", sep=',', header=0, float_precision=
#plt.plot(x=df['start_time'], y=[df['aggregate_power'], df['appliance_power']], grid=True)
df.plot(x='start_time', y=['aggregate_power', 'appliance_power'],grid=True)
```

In [2]:
```
print("Number of activations =", len(activations))
```

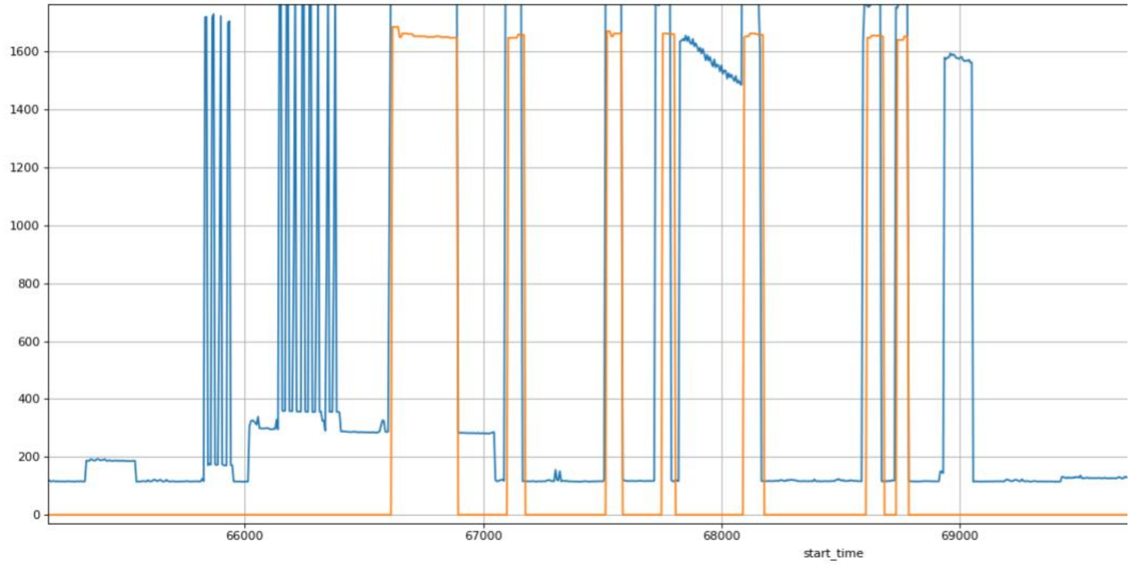
```
Number of activations = 16
```

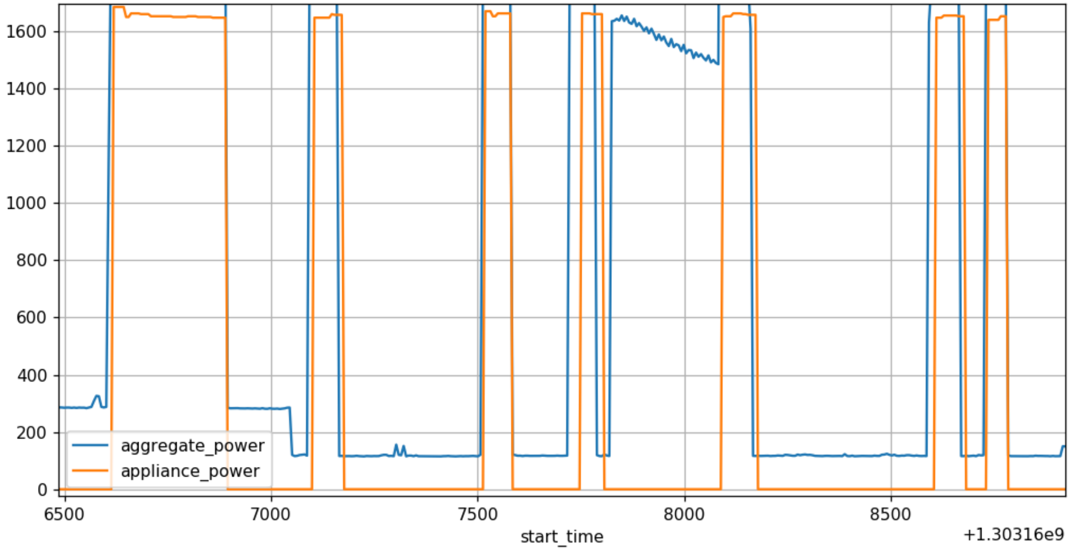


6000
5000
4000
3000
2000
1000
0
aggregate_powe
appliance_power
1.3030
1.3035
1.3040
1.3045
1.3050
1.3055
1.3060
start_time
1



Select windows of real aggregate data





Activation Points

start_time	end_time	aggregate	appliance_power
1303165953	1303165959	115.37	0
1303165959	1303165965	115.48	0
1303165965	1303165971	114.91	0
1303165971	1303165977	115.41	0
1303165977	1303165983	115.04	0
1303165983	1303165989	115.2	0
1303165989	1303165995	114.5	0
1303165995	1303166001	114.9	0
1303166001	1303166007	114.73	0
1303166007	1303166013	115.11	0
1303166013	1303166019	114.9	0
1303166019	1303166025	305.07	0
1303166025	1303166031	323.29	0
1303166031	1303166037	326.63	0
1303166037	1303166043	323.59	0
1303166043	1303166049	318.28	0
1303166049	1303166055	311.34	0
1303166055	1303166061	339.1	0
1303166061	1303166067	299.56	0
1303166067	1303166073	298.24	0
1303166073	1303166079	298.9	0
1303166079	1303166085	297.6	0
1303166085	1303166091	298.9	0
1303166091	1303166097	299.42	0
1303166097	1303166103	299.63	0
1303166103	1303166109	296.49	0
1303166109	1303166115	295.39	0
1303166115	1303166121	295.42	0
1303166121	1303166127	298.19	0
1303166127	1303166133	296.06	0
1303166133	1303166139	328.69	0
1303166139	1303166145	295.28	0
1303166145	1303166151	1889.81	0
1303166151	1303166157	1890.98	0
1303166157	1303166163	359.53	0
1303166163	1303166169	358.33	0
1303166169	1303166175	359.67	0
1303166175	1303166181	1898.2	0
1303166181	1303166187	1883.84	0
1303166187	1303166193	357.88	0
1303166193	1303166199	358.46	0
1303166199	1303166205	357.99	0

Standby

start_time	end_time	aggreg	applia e_power
1303132929	1303132935	224.19	0
1303132935	1303132941	223.14	0
1303132941	1303132947	222.21	0
1303132947	1303132953	222.43	0
1303132953	1303132959	222.96	0
1303132959	1303132965	225.66	0
1303132965	1303132971	223.43	0
1303132971	1303132977	227.29	0
1303132977	1303132983	225.22	0
1303132983	1303132989	226.28	0
1303132989	1303132995	225.78	0
1303132995	1303133001	226.56	0
1303133001	1303133007	225.16	0
1303133007	1303133013	225.87	0
1303133013	1303133019	225.72	0
1303133019	1303133025	226.24	0
1303133025	1303133031	227.6	0
1303133031	1303133037	227.07	0
1303133037	1303133043	225.21	0
1303133043	1303133049	227.14	0
1303133049	1303133055	226.11	0
1303133055	1303133061	222.22	0
1303133061	1303133067	222.75	0
1303133067	1303133073	222.94	0
1303133073	1303133079	223.46	0
1303133079	1303133085	224.04	0
1303133085	1303133091	222.48	0
1303133091	1303133097	221.27	0
1303133097	1303133103	221.26	0
1303133103	1303133109	221.22	0
1303133109	1303133115	221.41	0
1303133115	1303133121	222.99	0
1303133121	1303133127	223.4	0
1303133127	1303133133	222.9	0
1303133133	1303133139	218.88	0
1303133139	1303133145	218.49	0
1303133145	1303133151	219.15	0
1303133151	1303133157	217.25	0
1303133157	1303133163	218.35	0
1303133163	1303133169	217.9	0
1303133169	1303133175	218.96	0
1303133175	1303133181	218.08	0

Ongoing

start_time	end_time	aggregate	appliance_power
1.303E+09	1303166727	1947.41	1652.5
1.303E+09	1303166733	1946.23	1652.5
1.303E+09	1303166739	1946.69	1652.5
1.303E+09	1303166745	1947.26	1652.5
1.303E+09	1303166751	1946.91	1652.5
1.303E+09	1303166757	1945.99	1652.5
1.303E+09	1303166763	1944.16	1652.5
1.303E+09	1303166769	1944.27	1650
1.303E+09	1303166775	1946.82	1650
1.303E+09	1303166781	1944.02	1650
1.303E+09	1303166787	1944.08	1650
1.303E+09	1303166793	1946.56	1650
1.303E+09	1303166799	1948.53	1650
1.303E+09	1303166805	1947.4	1652.5
1.303E+09	1303166811	1944.57	1652.5
1.303E+09	1303166817	1982.16	1652.5
1.303E+09	1303166823	1985.66	1652.5
1.303E+09	1303166829	1972.39	1650
1.303E+09	1303166835	1948.07	1650
1.303E+09	1303166841	1946.93	1650
1.303E+09	1303166847	1986.67	1650
1.303E+09	1303166853	1945.41	1650
1.303E+09	1303166859	1945.64	1650
1.303E+09	1303166865	1943.42	1647.5
1.303E+09	1303166871	1943.53	1647.5
1.303E+09	1303166877	1947	1647.5
1.303E+09	1303166883	1941.17	1647.5
1.303E+09	1303166889	1940.26	1647.5
1.303E+09	1303166895	1943.19	1647.5
1.303E+09	1303166901	284.61	0
1.303E+09	1303166907	282.94	0
1.303E+09	1303166913	283.17	0
1.303E+09	1303166919	283.39	0
1.303E+09	1303166925	283.38	0
1.303E+09	1303166931	282.61	0
1.303E+09	1303166937	282.4	0
1.303E+09	1303166943	282.38	0
1.303E+09	1303166949	282.84	0
1.303E+09	1303166955	282.41	0
1.303E+09	1303166961	282	0
1.303E+09	1303166967	282.64	0
1.303E+09	1303166973	282.11	0

Generate real & synthetic samples

Both synthetic aggregate data and real aggregate data will be trained by the model at a ratio of 50:50

Training on a mix of synthetic and real aggregate data rather than just real data appears to improve the net's ability to generalize to unseen houses.

Synthetic data acts as a regularizer

The net's ability to generalize to unseen houses.

Validation and testing we use only real data

```python
def generate_real_sample(appliance_name):
    path = r'dataset/tobe_std_'
    df = pd.read_csv(path+appliance_name+'.csv', delimiter=',')  # appliances[i]])
    #df = pd.read_csv(r'test.csv', delimiter=',')  # appliances[i]])

    seq_length = 128
    data_set = {}
    for i in range(seq_length):
        data_set['aggregate_power_' + str(i)] = []
        data_set['appliance_power_' + str(i)] = []
        data_set['timestamp_' + str(i)] = []
    data_set['house_name'] = []

    set=0
    total_set = int(len(df.index)/128)

    for j in range(total_set):
        for i in range(seq_length):
            data_set['aggregate_power_' + str(i)].append(df['aggregate_power'][i+set])
            data_set['appliance_power_' + str(i)].append(df['appliance_power'][i+set])
            data_set['timestamp_' + str(i)].append(df['timestamp'][i+set])
        data_set['house_name'].append(df['house_name'][i+set])
        set = set+128

    #print(data_set)
    df2 = pd.DataFrame(data_set)
    df2.to_csv('dataset/dataset_' + appliance_name + '.csv', sep=' ', index=False)
```

	aggregate_power_0	appliance_power_0	timestamp_0	aggregate_power_1	appliance_power_1	timestamp_1	...	appliance_power_126	timestamp_126	aggregate_power_127	appliance_power_127	timestamp_127	house_name
0	115.37	0.0	1303165953	115.48	0.0	1303165959	...	1652.5	1303166709	1945.13	1652.5	1303166715	house_1
1	116.55	0.0	1303167489	115.86	0.0	1303167495	...	0.0	1303168245	119.80	0.0	1303168251	house_1
2	118.23	0.0	1303168257	117.03	0.0	1303168263	...	0.0	1303169013	1566.67	0.0	1303169019	house_1
3	87.39	0.0	1303667457	87.36	0.0	1303667463	...	0.0	1303668213	254.68	0.0	1303668219	house_1
4	252.51	0.0	1303668225	250.96	0.0	1303668231	...	1630.0	1303668981	1706.02	1630.0	1303668987	house_1
5	1703.89	0.0	1303668993	1713.70	0.0	1303668999	...	1682.5	1303669749	3280.42	1682.5	1303669755	house_1
6	55.42	0.0	1304181249	55.32	0.0	1304181255	...	10.0	1304182005	2725.49	0.0	1304182011	house_1
7	2732.33	0.0	1304182017	2054.98	0.0	1304182023	...	0.0	1304182773	4189.66	0.0	1304182779	house_1
8	4181.66	0.0	1304182785	5029.05	0.0	1304182791	...	0.0	1304183541	295.88	0.0	1304183547	house_1
9	113.88	0.0	1306103511	113.22	0.0	1306103517	...	1667.5	1306104267	3445.55	1667.5	1306104273	house_1
10	322.19	0.0	1306188759	324.26	0.0	1306188765	...	1630.0	1306189515	2013.30	1632.5	1306189521	house_1
11	565.85	0.0	1306190295	567.89	0.0	1306190301	...	1660.0	1306191051	2133.34	1660.0	1306191057	house_1
12	224.19	0.0	1303132929	223.14	0.0	1303132935	...	0.0	1303133685	220.93	0.0	1303133691	house_1
13	219.51	0.0	1303133697	220.37	0.0	1303133703	...	0.0	1303134453	211.99	0.0	1303134459	house_1
14	209.90	0.0	1303134465	211.22	0.0	1303134471	...	0.0	1303135221	204.98	0.0	1303135227	house_1
15	203.78	0.0	1303135233	203.74	0.0	1303135239	...	0.0	1303135989	201.31	0.0	1303135995	house_1

Synthetic aggregate data

- Extract a set of appliance activations for five appliances across all training houses.
- To create a single sequence of synthetic data, start with two vectors of zeros:
  - Input to the net
  - Input the target
  - The length of each vector = ‘window width’ of data for the network
- Decide whether or not to add an activation of that class to the training sequence.
  - 50% chance that the target appliance will appear in the sequence
  - 25% chance for each other ‘distractor’ appliance
- For each selected appliance class, an appliance activation is randomly selected and then add that activation on the input vector to a random location. Distractor appliances can appear anywhere in the sequence.

Standardization of dataset

The mean of each sequence is subtracted from the sequence to give each sequence a mean of zero.

Every input sequence is divided by the standard deviation of a random sample of the training set.

Targets are divided by a hand-coded 'maximum power demand' for each appliance

```python
def standardize_dataset(appliance_name):
    df = pd.read_csv(r'dataset/dataset_'+ appliance_name + '.csv', sep="\s+")

    seq_length = math.ceil(APPLIANCE_CONFIG[appliance_name]['window_width'] / SAMPLE_WIDTH)

    # Standardisation
    print('standardize', appliance_name)
    # get std of random sample
    sample = df.sample()
    aggregate_seq_sample = sample[['aggregate_power_' + str(i) for i in range(seq_length)]]
    aggregate_seq_sample = np.array(
        [aggregate_seq_sample['aggregate_power_' + str(i)].tolist()[0] for i in range(seq_length)])
    aggregate_seq_sample = aggregate_seq_sample - aggregate_seq_sample.mean()
    # print(aggregate_seq_sample, len(aggregate_seq_sample), np.std(aggregate_seq_sample))
    sample_std = np.std(aggregate_seq_sample)

    for i in range(seq_length):
        print(round(100 * i / seq_length / 2, 1), '%')
        i = str(i)
        new_column = pd.Series((df['aggregate_power_' + i] - df['aggregate_power_' + i].mean()) / sample_std,
                               name='aggregate_power_' + i)
        df.update(new_column)

    max_power = APPLIANCE_CONFIG[appliance_name]['max_power']

    for i in range(seq_length):
        print(round(100 * i / seq_length / 2, 1), '%')
        i = str(i)
        new_column = pd.Series(df['appliance_power_' + i] / max_power, name='appliance_power_' + i)
        df.update(new_column)
    print(df)
    df.to_csv(r'dataset/standardized_dataset_' + appliance_name + '.csv', sep=' ', index=False)
```

Standardized dataset samples

	aggregate_power_0	appliance_power_0	timestamp_0	aggregate_power_1	appliance_power_1	timestamp_1	...	appliance_power_126	timestamp_126	aggregate_power_127	appliance_power_127	timestamp_127	house_name
0	-0.942171	0.000323	1376118933	-0.932542	0.000323	1376118939	...	0.000323	1376119689	-0.447197	0.000323	1376119695	house_1
1	4.068014	0.955806	1368211882	4.215402	0.957097	1368211888	...	0.000323	1368212638	-0.350437	0.000323	1368212644	house_1
2	-0.335372	0.000645	1405247791	-0.324104	0.000645	1405247797	...	0.000645	1405248547	0.089082	0.000645	1405248553	house_1
3	2.567418	0.000323	1364813404	2.555727	0.000323	1364813410	...	0.000323	1364814160	3.011552	0.000323	1364814166	house_1
4	-0.956930	0.000323	1380206187	-0.948942	0.000323	1380206193	...	0.000323	1380206943	-0.342237	0.000323	1380206949	house_1
5	-0.610892	0.000323	1396794098	-0.597983	0.000323	1396794104	...	0.000323	1396794854	-0.137238	0.000323	1396794860	house_1
6	-0.783091	0.000323	1406110199	-0.773462	0.000323	1406110205	...	0.000323	1406110955	-0.197918	0.000323	1406110961	house_1
7	-0.551852	0.000323	1399811475	-0.586503	0.000323	1399811481	...	0.750968	1399812231	-0.035558	0.752903	1399812237	house_1
8	-0.676491	0.000323	1452445281	-0.666863	0.000323	1452445287	...	0.000323	1452446037	-0.447197	0.000000	1452446043	house_1
9	-0.799491	0.000323	1405510929	-0.778382	0.000323	1405510935	...	0.000323	1405511685	-0.342237	0.000323	1405511691	house_1
10	3.018417	0.765806	1390034325	2.991966	0.757742	1390034331	...	0.000323	1390035081	-0.273357	0.000323	1390035087	house_1

Split data

```
def load_test_train_data(appliance_name, type='default'):
    df = pd.read_csv(PREPOCESSED_DATA_DIR + '/standardized_dataset_' + appliance_name + '.csv', sep="\s+")
    seq_length = math.ceil(APPLIANCE_CONFIG[appliance_name]['window_width'] / SAMPLE_WIDTH)
    df_input = df[['aggregate_power_' + str(i) for i in range(seq_length)]]
    df_target = df[['appliance_power_' + str(i) for i in range(seq_length)]]
    print(df_target)
    X_train, X_test, y_train, y_test = train_test_split(df_input, df_target, test_size=1 / (1 +
    TRAIN_TEST_RATIO), random_state=42)

    return X_train, X_test, y_train, y_test
```

- The number of appliance training activations is show in Table 1
- The number of testing activations is shown in Table 2
- The specific houses used for training and testing is shown in Table 3

Table 1: Number of training activations per house.

	1	2	3	4	5
Kettle	2836	543	44	716	176
Fridge	16 336	3526	0	4681	1488
Washing machine	530	53	0	0	51
Microwave	3266	387	0	0	28
Dish washer	197	98	0	23	0

Table 2: Number of testing activations per house.

	1	2	3	4	5
Kettle	54	29	40	50	18
Fridge	168	277	0	145	140
Washing machine	10	4	0	0	2
Microwave	90	9	0	0	4
Dish washer	3	7	0	3	

Table 3: Houses used for training and testing.

	Training	Testing
Kettle	1, 2, 3, 4	5
Fridge	1, 2, 4	5
Washing machine	1, 5	2
Microwave	1, 2	5
Dish washer	1, 2	5

Machine Learning

- Train dataset is feed into DAE model to train the model
- Test dataset is being used to test generalization of dataset and accuracy of model

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_1 (Conv1D)            (None, 128, 8)            40
_________________________________________________________________
flatten_1 (Flatten)          (None, 1024)              0
_________________________________________________________________
dropout_1 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_1 (Dense)              (None, 1024)              1049600
_________________________________________________________________
dropout_2 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               131200
_________________________________________________________________
dropout_3 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 1024)              132096
_________________________________________________________________
dropout_4 (Dropout)          (None, 1024)              0
_________________________________________________________________
reshape_1 (Reshape)          (None, 128, 8)            0
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 128, 1)            33
=================================================================
Total params: 1,312,969
Trainable params: 1,312,969
Non-trainable params: 0
_________________________________________________________________
loading from model_kettle_1_20epo.hdf5
2019-04-30 00:51:24.701659: I tensorflow/core/platform/cpu_feature_guard.cc:141] You
MAE is 31.999124495367255
```

1. Input (length determined by appliance duration)
2. 1D conv (filter size=4, stride=1, number of filters=8, activation function=linear, border mode=valid)
3. Fully connected (N=(sequence length - 3) $\times$ 8, activation function=ReLU)
4. Fully connected (N=128; activation function=ReLU)
5. Fully connected (N=(sequence length - 3) $\times$ 8, activation function=ReLU)
6. 1D conv (filter size=4, stride=1, number of filters=1, activation function=linear, border mode=valid)

Resnet model

```
Training Data:
(8966, 128, 1)
(8966, 128)
Test Data:
(997, 128, 1)
(997, 128)
WARNING:tensorflow:From C:\Users\Yu Zhe\AppData\Local\Programs\Python\Python37\lib\site-packages\tensorflow\pyt
nd will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
================================================================================================
input_1 (InputLayer)            (None, 128, 1)       0
________________________________________________________________________________________________
zero_padding1d_1 (ZeroPadding1D (None, 134, 1)       0           input_1[0][0]
________________________________________________________________________________________________
conv1 (Conv1D)                  (None, 64, 64)       512         zero_padding1d_1[0][0]
________________________________________________________________________________________________
bn_conv1 (BatchNormalization)   (None, 64, 64)       256         conv1[0][0]
________________________________________________________________________________________________
activation_1 (Activation)       (None, 64, 64)       0           bn_conv1[0][0]
________________________________________________________________________________________________
max_pooling1d_1 (MaxPooling1D)  (None, 31, 64)       0           activation_1[0][0]
________________________________________________________________________________________________
res2a_branch2a (Conv1D)         (None, 31, 64)       4160        max_pooling1d_1[0][0]
________________________________________________________________________________________________
bn2a_branch2a (BatchNormalizati (None, 31, 64)       256         res2a_branch2a[0][0]
________________________________________________________________________________________________
activation_2 (Activation)       (None, 31, 64)       0           bn2a_branch2a[0][0]
________________________________________________________________________________________________
res2a_branch2b (Conv1D)         (None, 31, 64)       12352       activation_2[0][0]
```

```
                                                                 activation_46[0][0]
________________________________________________________________________________________________
activation_49 (Activation)      (None, 4, 2048)      0           add_16[0][0]
________________________________________________________________________________________________
avg_pool (AveragePooling1D)     (None, 1, 2048)      0           activation_49[0][0]
________________________________________________________________________________________________
flatten_1 (Flatten)             (None, 2048)         0           avg_pool[0][0]
________________________________________________________________________________________________
predictions (Dense)             (None, 128)          262272      flatten_1[0][0]
================================================================================================
Total params: 16,296,192
Trainable params: 16,243,072
Non-trainable params: 53,120
________________________________________________________________________________________________
WARNING:tensorflow:From C:\Users\Yu Zhe\AppData\Local\Programs\Python\Python37\lib\site-packages\tensorflow\python\ops\math_ops.py:30
d in a future version.
Instructions for updating:
Use tf.cast instead.
Train on 8966 samples, validate on 997 samples
Epoch 1/20
2019-05-01 10:37:32.189863: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports instructions that this TensorFlow
8966/8966 [==============================] - 401s 45ms/step - loss: 0.0399 - acc: 0.7369 - val_loss: 0.0100 - val_acc: 0.7954

Epoch 00001: val_loss improved from inf to 0.00999, saving model to model_kettle_1_20epo.hdf5
Epoch 2/20
8966/8966 [==============================] - 383s 43ms/step - loss: 0.0113 - acc: 0.7777 - val_loss: 0.0100 - val_acc: 0.7954

Epoch 00002: val_loss did not improve from 0.00999
```

Fact: Resnet is an artificial neural network of a kind that builds on constructs known from pyramidal cells in the cerebral cortex. Residual neural networks do this by utilizing skip connections, or short-cuts to jump over some layers. Typical ResNet models are implemented with single-layer skips

References

- Kelly, J., & Knottenbelt, W. (2015). Neural NILM: Deep Neural Networks Applied to Energy Disaggregation. *Proceedings of the 2nd ACM International Conference on Embedded Systems for Energy-Efficient Built Environments - BuildSys 15*. doi:10.1145/2821650.2821672

Thank you