EV peak demand optimization

Abhilash

Table of Content:

In the previous assignment, we did the data pre processing pipeline to the point where we had the total power consumption for every hour using the following steps.

- We fixed the connectionTime and created a new column endTime as the min('disconnectTime', 'doneChargingTime') to account for whichever happens first. This is done since whichever event happens first determines when the power delivered to the vehicle stopped.
- Then we computed power delivered per second by dividing the 'kWhDeliverd' by the total seconds in the window(connectionTime to endTime)
- Then for each hour starting from the first date, we computed the total power consumed in that hour.
  -For first hour- Power_per_second*(3600-seconds(60*minute))- Every subsequent hour- Power_per_second*3600 Last Hour- Power_per_second*((60*minute)*second)
- Then we collated for each 1 hour window the total power consumed by all the users put together by aggregating the values computed in the last step hour wise.

A. Data Preprocessing

In this assignment, the added functionality is the ability to push the charging by 1 hour. Now for each hour window, we can push a certain amount of power consumed onto the next hour. We will need to optimize these values pushed onto the next hour such that a certain cost function achieves its minimum value.

First we compute how much of this power can be pushed into the next window. For this we will need to compute how much power consumption would happen in the pushed window and compare with the old power consumption to calculate how much can be pushed.

To calculate power consumption in the pushed window, we update the connectionTime by 1 hour. For computing the end time there is a caveat. If the charge is ending without being disconnected, it means the charge finishing time can be pushed. But if the user is disconnecting the plug before done time, this is a fixed event which cannot be pushed. To compute this, we set the endTime to be min('disconnecttime','doneChargingTime+1'). The rest of the power calculations happen similarly in the previous step.

Now we have 2 power time frames for each hour showing the power consumption with push and without push. If the power is decreasing in the pushed window for the following hour, it is pushable. But if the power is increasing, we would be increasing the load on the system, going against optimization.

Comparing the power values, the pushed window shows the power load if all the power consumption was pushed by exactly 1 hour in the starting. This is the maximum push. The optimal value would be somewhere between max value in each window and 0(indicating no push). For this we subtract the new power values by old values and convert the negative values to 0(since these indicate power load increase). This gives us D_max having a pushable load for each time window t. Hence the pushable load would be anywhere between 0 and the max value that can be pushed. Point to note here is that for a time window t, only the charging whose start time occurs in that window can be pushed.

Here the edge case that comes out is when the connection time and disconnect time is less that 1 hour. That is the charging and disconnecting happened within the 1 hour window that we were trying to push. We handle this by removing all the values who fall in this category in the pushed timeframe. Since the initial time frame will have all the power values and the new timeframe has none of this, it denotes the whole charging can be avoided, decreasing effective load on the grid. The case where doneTime is within one hour window is inherently handled by the addition of our 1 in the min('disconnecTime','doneChargingTime+1').

Now that we have Dmax, the max power that can be pushed in each window, for each t we need to find the value of D(t) as part of the optimization. This needs to confer to the cost functions and equations outlined next.

B. Optimization - Formulating the problem and Solution

The problem gives out that
With
P1- The max power consumed in November after optimisation
P2- The max power consumed in December after optimisation

Where,

$$P(t) = P_0(t) - D(t) + D(t-1).$$

We need to minimize-

$$\min\{C(x) = P_1 + P_2 + \alpha \sum_{t=1}^{1464} D(t)\}$$

That is, we need to minimize the net Peak overload over November and December along with the cost associated with delaying the charging by 1 hour for all the time intervals where delay happens.

For this we will do the gradient optimization with respect to the cost function that is given. That is we will be finding D(t) for each t by a grading descent on the cost function.

1. **Initialization**:
   - $D(t) = 0$ for $t = 1$ to 1488 (initialization)
   - Set α to a predefined value.
   - Define $D_{\max}(t)$ for each t.
2. **Compute Cost Function**:
   - Compute $P1$ and $P2$ based on the maximum consumption in the first and second months, respectively.
   - Compute the cost function $C(x)$ using:
     $C(x) = P1 + P2 + \alpha \sum_{t=1}^{1488} D(t)$
3. **Compute Gradients**:
   - Compute the gradient of the cost function with respect to each $D(t)$:
     $$\frac{\partial C}{\partial D(t)} = \begin{cases} -1 + \alpha & \text{if } t = t_1 \text{ or } t = t_2 \\ \alpha & \text{otherwise} \end{cases}$$
4. **Update Parameters**:
   - Update $D(t)$ using gradient descent with the maximum value constraint:
     $D(t) := \text{clip}(D(t) - \eta \times \frac{\partial C}{\partial D(t)}, 0, D_{\max}(t))$
5. **Repeat**:
   - Repeat steps 2-4 until convergence.

C. Coding the solution

1. **Initialization**:
   - The algorithm begins by initializing the variables including `D`, `alpha`, `learning_rate`, and `iterations`.
   - `D` is initialized as an array randomly between 1 to 10, representing the consumption at each hour `t`.
   - `alpha` is a regularization parameter.
   - `learning_rate` determines the step size in the gradient descent update.
   - `iterations` specifies the number of iterations to perform.

2. **Compute Cost Function**:
   - The `compute_cost` function calculates the cost function, which represents the objective to minimize.
   - It computes the consumption `P` at each hour using the current `D`.
   - It then calculates the maximum consumption in each month (`P1` and `P2`) and returns the sum of these values plus a regularization term.

3. **Compute Gradients:**
   - The `compute_gradients` function calculates the gradient of the cost function with respect to each element of `D`.
   - It computes the consumption `P` at each hour using the current `D`.
   - It finds the hours (`t1` and `t2`) with the highest consumption in each month.
   - It updates the gradients for these hours with the corresponding adjustments, including the regularization term.

4. **Update Parameters:**
   - The `update_parameters` function updates the `D` values using the calculated gradients and the specified learning rate.
   - It performs a gradient descent step by subtracting the product of the gradients and learning rate from the current `D`.
   - Additionally, it clips the updated `D` values to ensure they remain within specified constraints (minimum 0 and maximum `D_max`).

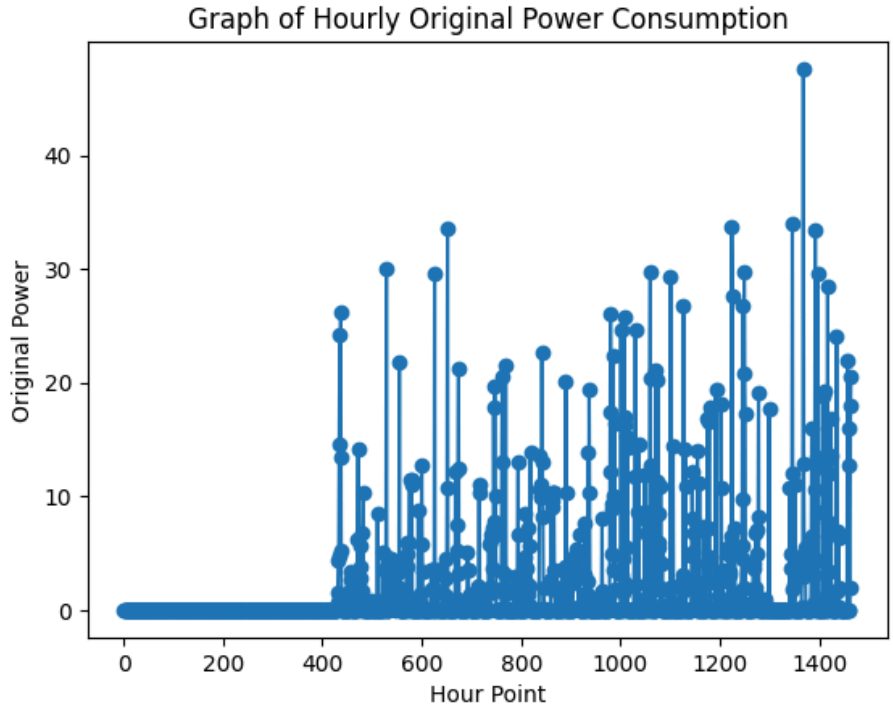
5. **Repeat**:
   - The algorithm iterates over a fixed number of iterations.
   - In each iteration, it computes the cost function, calculates gradients, updates parameters, and repeats until convergence or until the specified number of iterations is reached.
   - The optimization process aims to minimize the cost function by adjusting the `D` values.

This is the gradient descent algorithm that we are going to use for both offline and online solutions. Later, we also compare the results for various values of alpha, learning rate, and the number of iterations.

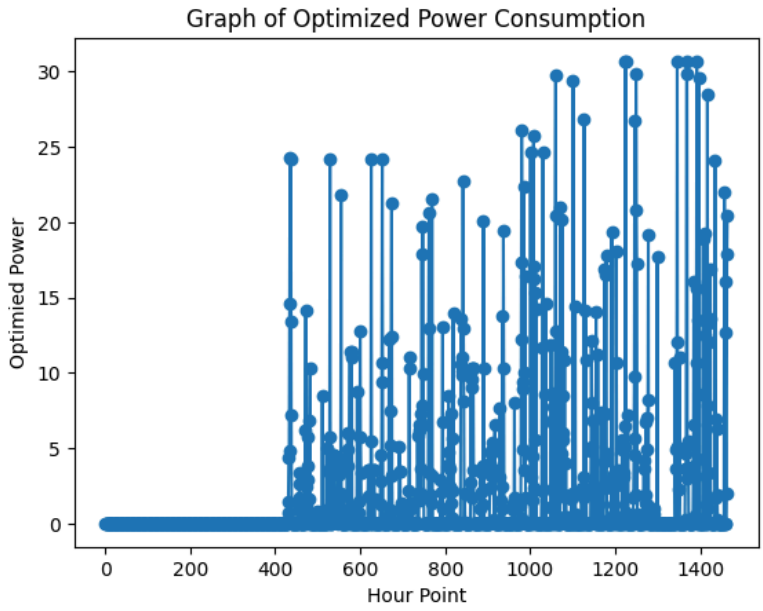
D. Offline Solution

For now, we chose alpha as 0.1, learning rate as 0.01, and the number of iterations are 3M. The **cost** we got is around 56.

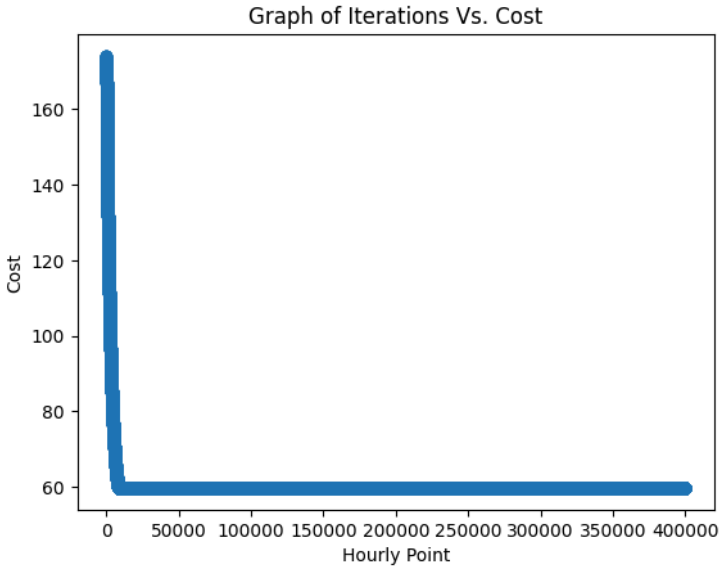
Below is the diagram for Original Power Consumption for the months November and December.



Below is the for Optimized Power Consumption for the months November and December.



We can have a look at how cost is changing with respect to the number of iterations.

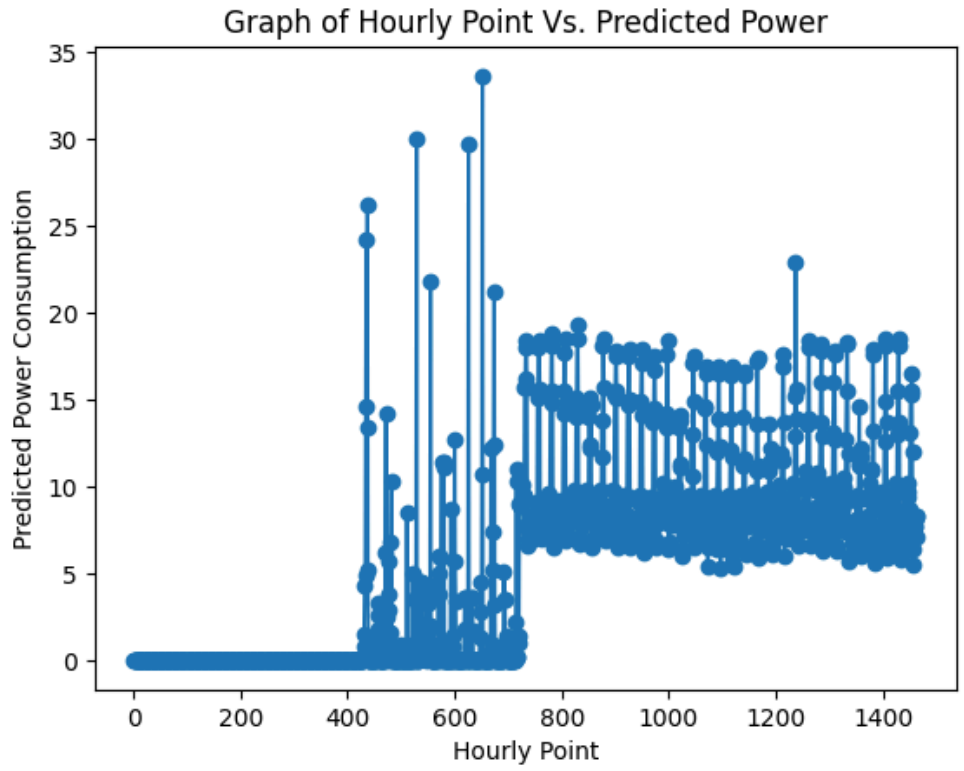


From the above figure, we can see that the cost is gradually decreasing as the number of iterations increase. It could even decrease more by increasing the number of iterations. However, considering the time that our algorithm takes it is not feasible to increase the iterations.

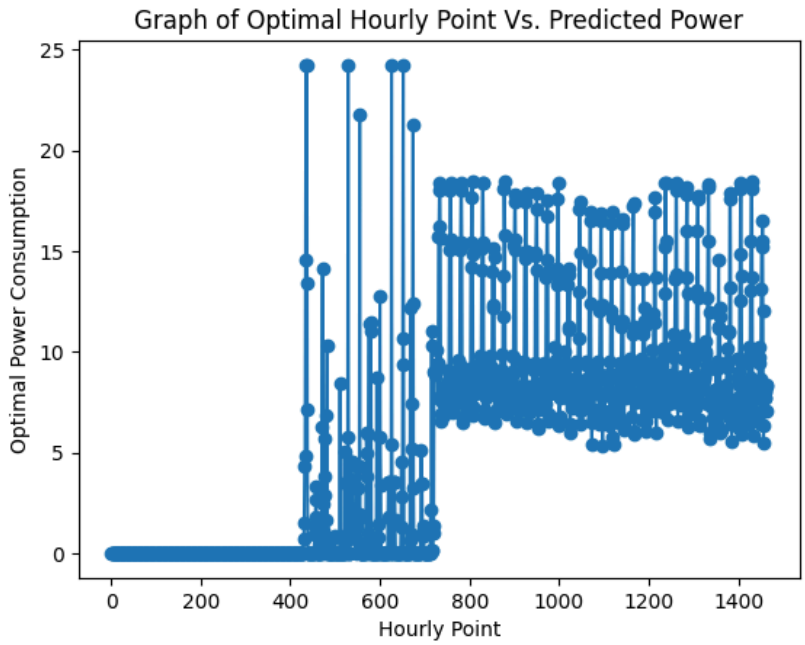
E. Online Solution

For the online solution, we used the predicted values of december. As described in the first report, we used GradientBoost Regressor to predict the December values. We should the same algorithm that is Gradient Descent algorithm and the same parameter such as alpha = 0.1, learning rate = 0.01, and iterations = 3M. The cost we got is around 45, which is lesser than the offline solution.

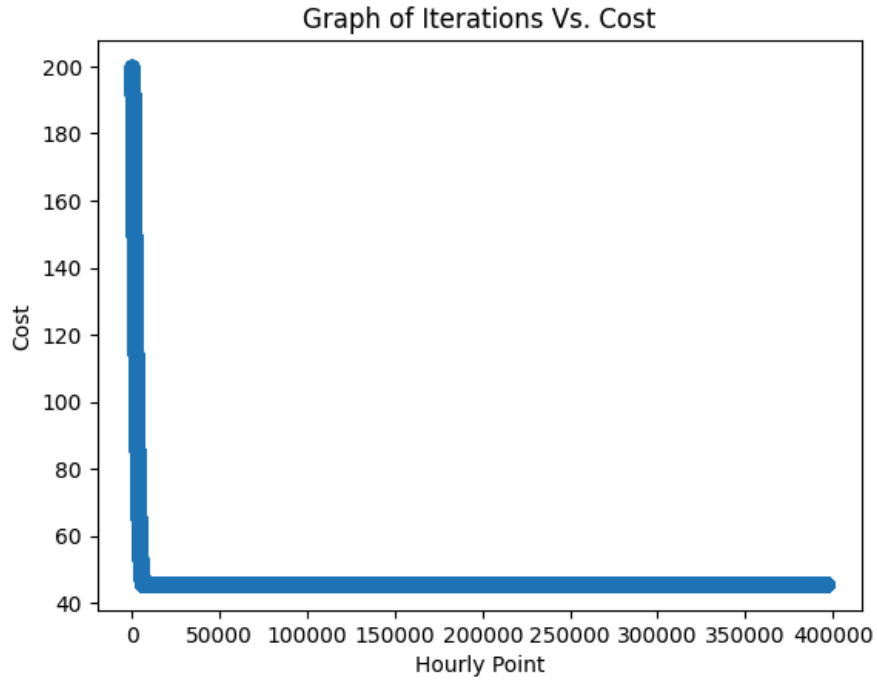
Below is the diagram for Predicted Power Consumption for the months November and December.



Below is the diagram for Optimized Power Consumption for the predicted months of November and December.



We can have a look at how cost is changing with respect to the number of iterations.



F. Offline Cost Vs. Online Cost Ratio

Offline cost : 56
Online cost : 45

The ration between the online and offline is 45/56 ~ 0.8. This is more visualized and compared in the section F.

G. Compare the results from the two items above and explain the differences in detail.

For offline:

1. Before optimization
   Peak Power value in November- 33.6
   Peak Power value in December- 47.5
   Total cost - 81.1
2. After Optimization
   Peak Power value in November-24.23
   Peak power value in December-30.67
   Total cost- 56
   Net Cost Reduction- 81.1- 56= 31.1

For online:

1. Before optimization
   Peak Power value in November- 33.6
   Peak Power value in December-22.85
   Total cost - 56.45
2. After Optimization
   Peak Power value in November-24.22
   Peak power value in December-18.42
   Total cost- 45.6
   Net Cost Reduction - 56.45-45.6= 10.85

Comparing the two sets of results, we see the following observations

1. The peak values are much more pronounced in the offline solution than in the online solution. This phenomenon is explained as the real time data would always have extreme values, but the fitted model will generally not have these extreme values and would try to predict the values in the possible range.
2. The more important observation we see is that the cost reduction in offline is so much more than the cost reduction in online solution. This is because the power distribution of

real time data is very anomalous not confirming much to set patters. But in the online one, confirming to our observation one, the power distribution would be much more uniform since the predictions are generated according to a set of patterns and rules. Hence not much load distribution is possible here resulting in lesser scope for optimization.

H. Comparison of Varying Parameters

Solution \ Alpha	0.01	0.1	1	2
Offline	55.3	56	81.19	81.32
Online	42.38	45.2	56.12	56.45
Online-Offline Ratio	0.766	0.807	0.69	0.6941

Observations:

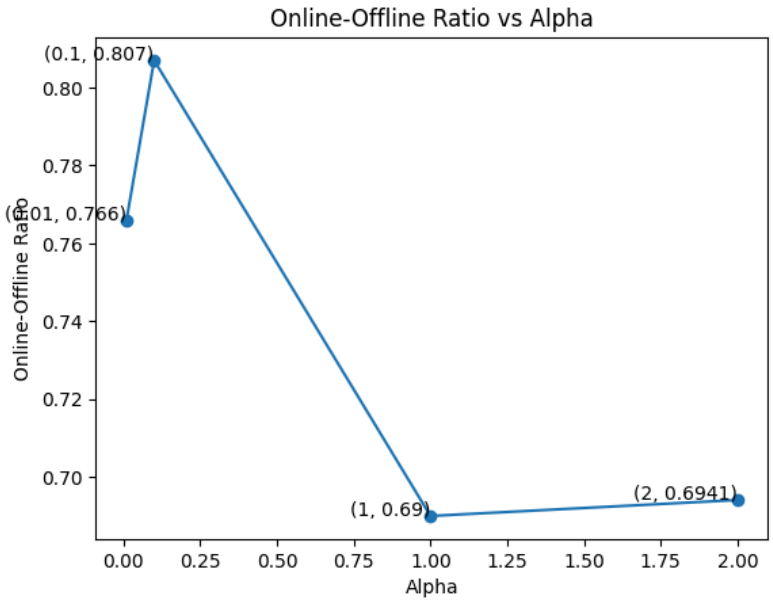
1. **Effect of Alpha on Offline and Online Costs**:
   - As alpha increases from 0.01 to 2, both offline and online costs increase.
   - This indicates that higher values of alpha lead to higher costs in both offline and online scenarios.

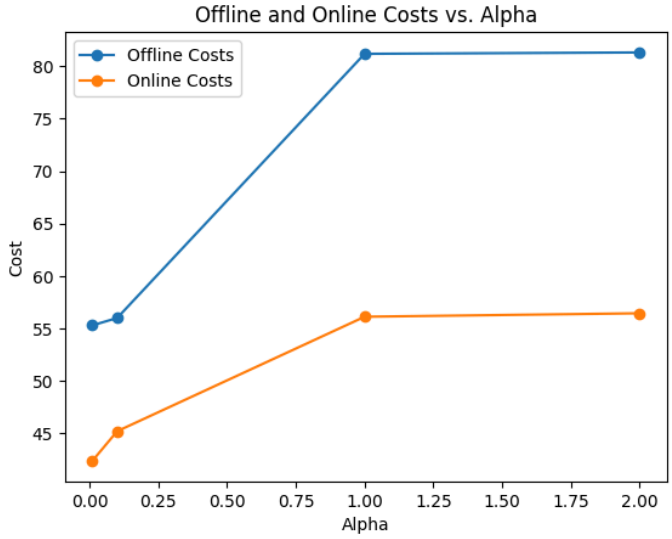
2. **Online-Offline Cost Ratio:**
   - The online-offline cost ratio decreases as alpha increases.
   - This suggests that while both offline and online costs increase with higher alpha, the relative increase in online costs compared to offline costs decreases.

3. **Sensitivity to Alpha**:
   - The ratio between online and offline costs (online-offline ratio) is more sensitive to changes in alpha at lower values (0.01 and 0.1) compared to higher values (1 and 2).
   - This indicates that changes in alpha have a greater impact on the relative costs of online and offline systems at lower alpha values.

4. **Threshold Effect**:
   - There seems to be a threshold effect around alpha = 1, where the rate of increase in both offline and online costs slows down compared to lower alpha values.
   - Beyond alpha = 1, the increases in costs become marginal.

Below are a few graphs from the above table.





I. Bibliography

1. Offline and Online Electric Vehicle Charging Scheduling With V2V Energy Transfer | IEEE Journals & Magazine | IEEE Xplore
2. How to implement a gradient descent in Python to find a local minimum ? - GeeksforGeeks

3. Gradient Descent in Python. When you venture into machine learning… | by Sagar Mainkar | Towards Data Science
4. Peak Shaving in Energy Storage: Balancing Demand, Savings, and Sustainability (powermag.com)

(Note: We have submitted our code in BrightSpace, please check.)