SAPIENZA
UNIVERSITÀ DI ROMA

Smart Houses

Machine Learning Project Report

Deborah Dore

1994616

dore.1994616@studenti.uniroma1.it

Vincenzo Imperati

1834930

imperati.1834930@studenti.uniroma1.it

Academic Year 2021-2022

Contents

Abstract

This paper proposes an hour-ahead Demand Response algorithm for Home Energy Management System. To deal with the uncertainty in future prices, a steady price prediction model based on recurrent neural network is presented. In cooperation with forecasted future prices, multi-agent reinforcement learning is adopted to make optimal decisions for different home appliances in a decentralized manner.

1 Introduction

Lately, improving electricity use in homes is becoming more and more critical. Enhancing electricity usage is possible through the *Demand Response* algorithm that allows users to manage their energy consumption correctly, reply better to peak demand or electricity market offers and utilise the energy resources more efficiently.
This work proposes an hour-ahead Demand Response algorithm for Home Energy Management System (HEMS) that uses machine learning techniques, considering electricity costs and client's dissatisfaction.
Specifically, because of the inherent nature in hour-ahead electricity price market, the customer accesses only one price for the current hour. To deal with the uncertainty in future prices, a stable price forecasting model is presented, which is implemented by a Long Short-Term Memory (LSTM). A particular Recurrent Neural Network capable of learning long-term dependencies.
Price's prediction has become an essential argument in today's electrical engineering, and numerous methods have been tested for its implementation. The LSTM method is easy and efficient to implement, showing good predictions due to the nature of the prices: a time series strictly correlated.
Figure 1 shows the detailed DR algorithm that combines the long short-term memory and multi-agent reinforcement learning. Every hour, the HEMS receives the hour-ahead price, and uses the LSTM to predict the future prices (twelve hours forecasted). In cooperation with the forecasted future prices, MARL is adopted to make optimal decisions for different appliances in a decentralized manner, to minimize the user energy bill and degree of discomfort. Here, each appliance has an agent, and RL is used for decision-making in the context of uncertainty regarding the price information and load demand of the appliances.

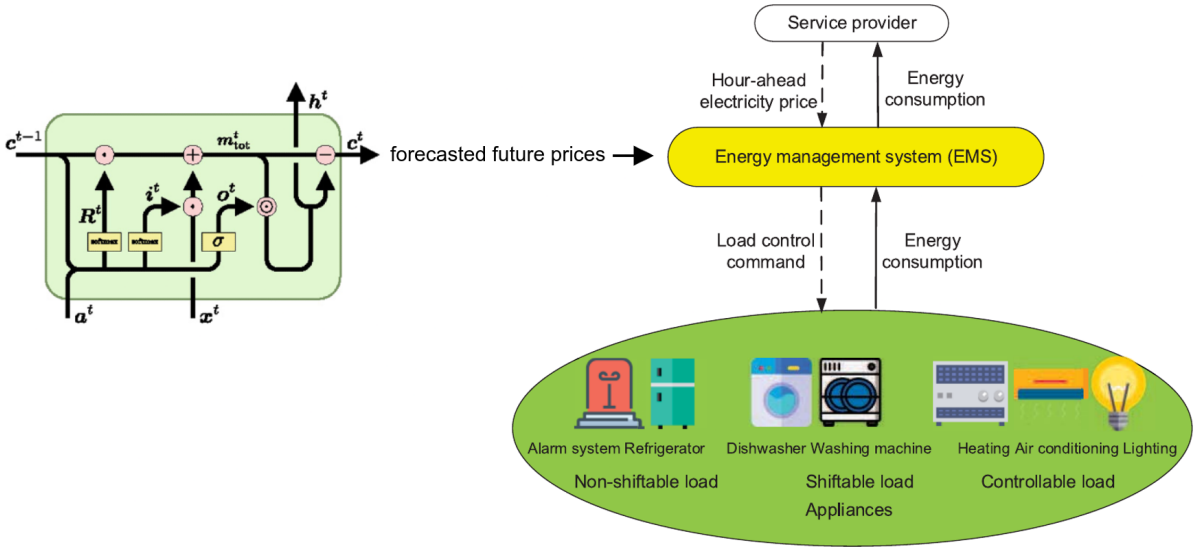


Figure 1: HEMS architecture

2 First Phase: Long Short Term Memory Model

2.1 Data Processing

Nordpoolspot [4] provides the energy price data used by the implemented HEMS for the demand response algorithm. The dataset has two columns: *starting*, which contains a timestamp denoting the price's day and hour, and *energy_market_price*, which holds the relative price.

starting	energy_market_price

Table 1: Representation of the dataset *energy.60.csv*

The actual structure of the dataset does not make it ready-to-use for a neural network model. Our goal is to forecast prices over the next 12 hours. We may train a model using prices from the previous hours, specifically, the previous 50 hours, to accomplish this.
One crucial step before creating the new dataset is to verify the presence of nan values and remove them if so. The Sklearn library's k-Nearest Neighbors function was used to fill in missing values. The mean value from n nearest neighbors discovered in the training set is used to impute missing values for each sample. If the qualities that neither sample lacks are similar, the samples are similar.

```
imputer = KNNImputer(n_neighbors=2, weights='uniform')
transformed_data =
↪  pd.DataFrame(imputer.fit_transform(df[['energy_market_price']]),
↪  columns=['energy_market_price'])
```

The imputation was done with two neighbors, and the weighting technique was uniform: all points in each neighborhood were weighted equally. We do not have to worry about outliers because the house's energy prices and consumption records have already been checked and cleaned, and our data was collected from there.
We can now begin constructing the dataset that will be used to train the model. To accomplish so, we developed a simple algorithm that generates a CSV file with 64 columns: a timestamp, 50 energy prices from the previous 50 hours, the current price we are looking at, and then the prices for the next 12 hours.

```
def create_dataset(energy, nn_datas):
    energy_market_price = []
    with open(energy, "r") as energy_60_file, open(nn_datas, "w") as
    ↪  nn_datas_file:
        reader = csv.reader(energy_60_file)
        writer = csv.writer(nn_datas_file)
        headers = ['timestamp']
        headers.extend(['energy_price_ahead_' + str(n) for n in range(50, 0, -1)])
        headers.extend(['energy_price_forward_' + str(n) for n in range(0, 13)])
        writer.writerow(headers)
        timestamp = []
        for index, row in islice(enumerate(reader), 1, None):
            energy_market_price.append(row[1])
```

```
            timestamp.append(row[0])
            if index > 62:
                new_line = [timestamp[index - 13]]
                new_line.extend(energy_market_price[index - 63: index])
                writer.writerow(new_line)
```

2.2 The model

2.2.1 Definition

Long Short Term Memory, **LSTM**, is a type of recurrent neural network that can handle long-term dependencies. We decided to use the LSTM model because of its ability to preserve long-term dependencies. Furthermore, it also reduces the problem of the vanishing gradient along the temporal line.
Its architecture is similar to an RNN cell. In an RNN, the structure is a chain of repeating simple modules of a neural network.

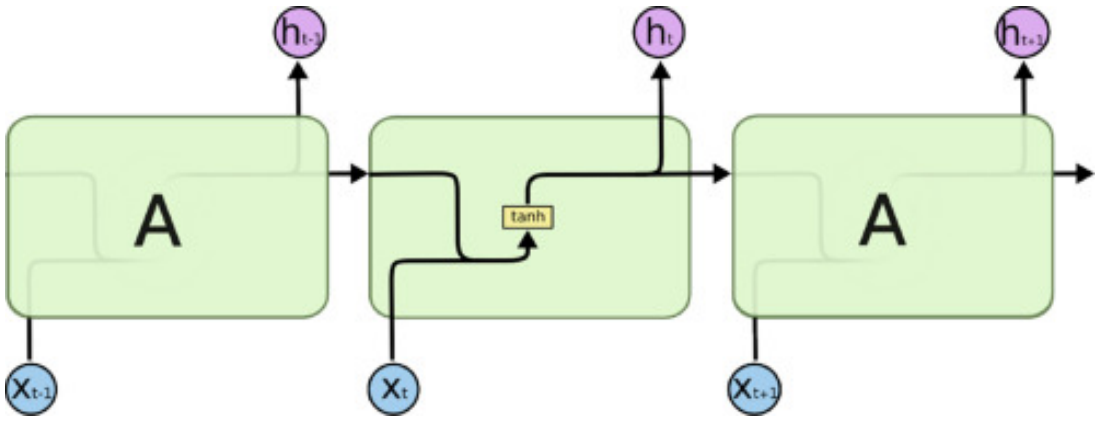


Figure 2: RNN architecture

In an LSTM, the repeating module has a different structure. It is composed of four interacting neural network layers.

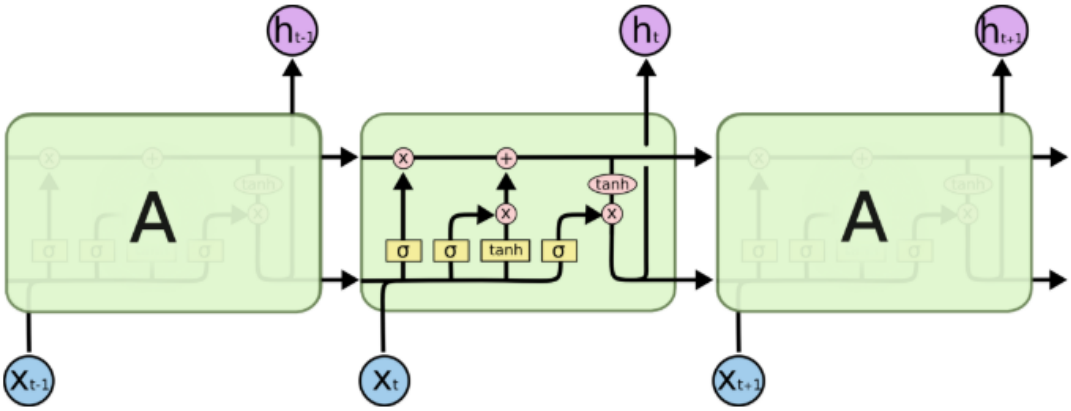


Figure 3: LSTM architecture

Inside an LSTM cell, we can distinguish between:

- **Cell State**, which reduces the problem of short memory in an RNN. It maintains a vector C_t which dimension is the same as the hidden state h_t. Inside the cell state, the information can be forgotten using the **Forget Operator (X)** but also added using the **Input (+) Operator**.

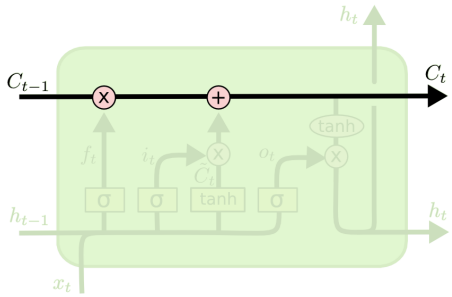
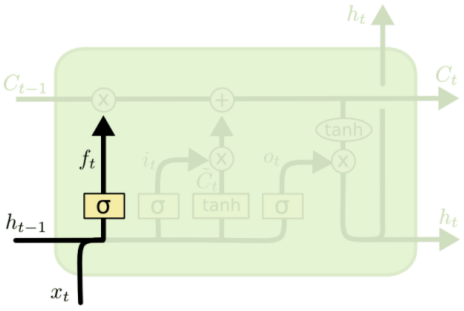


Figure 4: Cell State of an LSTM

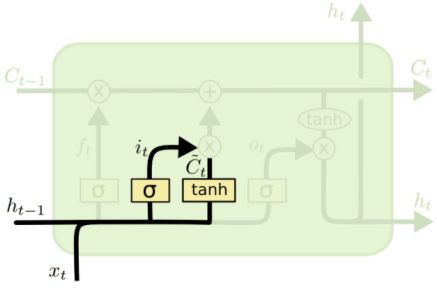
- **Forget Gate** is a sigmoid layer that decides what should be forgotten about the previous state. It takes h_{t-1} and x_t as input and outputs a value between 0 and 1 where 0 is *completely keep this value* and 1 is *completely forget about this value.*



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Figure 5: Forget Gate of a LSTM cell

- **Input Gate** comprises a sigmoid layer that decides which values to update and a tahn layer that determines the amount to add/subtract from these values.

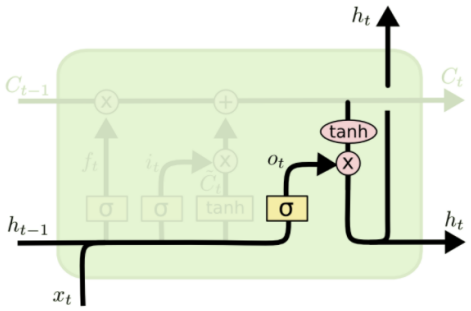


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Figure 6: Input Gate of a LSTM cell

- **Output Gate** decides what goes to the next layer and the next state based on our cell state, which was scaled to [-1,1] using the tahn. The output is computed using

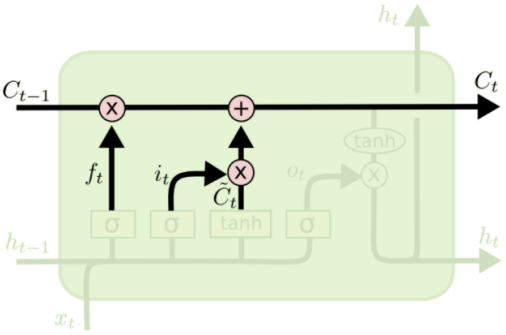
the sigmoid function between the scaled cell state output and the current hidden state.



$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Figure 7: Output Gate of a LSTM cell

Nevertheless, how is the cell state updated? We multiply the old state C_{t-1} by f_t, the output of the Forget Gate. Then we add $i_t * \tilde{C}_i$, which was the output of the Input Gate. We obtain the new cell state C_t.



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Figure 8: Updating the cell state in a LSTM

The applications of an LSTM are countless: we used it in time series prediction for which is most suited.

2.2.2 Implementation

We choose Keras as a library to implement our model. *Keras* is an open-source software library that provides a Python interface for artificial neural networks. It acts as an interface for the Tensorflow library. Using Keras, we were able to define each layer of our LSTM model in the following way:

```
model = Sequential()
model.add(LSTM(units=50, return_sequences=True, input_shape=(50, 1)))
model.add(Dropout(0.5))
model.add(LSTM(units=50))
model.add(Dropout(0.5))
model.add(Dense(12))
model.compile(optimizer='adam', loss="mse", metrics=['mse', 'mae', 'mape'])
```

The first layer incorporates 50 units and takes 50 features as input. Those features represent the prices of the energy for the previous 50 hours. A dropout layer was added after each LSTM layer to avoid over-fitting. The hidden layer is composed of an LSTM layer with 50 units. The last layer is a Dense one with 12 outputs for the prices of the following 12 hours to be predicted.
The model was compiled using the *Adam Optimizer*, a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments. The function to minimize is the Mean Square Error, the average squared difference between the estimated values and the actual value. The compile function will also calculate the Mean Absolute Error, the distance between the predicted and actual value, and the Mean Absolute Percentage Error. This model is a relative-simple implementation used in the first phases of the research (it will be changed afterward using hyper-parameter search) to understand if the LSTM is a valid prediction model for our data.

2.2.3 Training and Testing

For the training and testing part, the dataset was split into train (70%) and testing (30%), without shuffling. Our dataset is a strictly correlated time series, so shuffling the dataset would have meant losing this connection.
Each set was then scaled in the range [0,1] using the MinMaxScaler provided by the Sklearn library. Scaling the dataset makes it easy for the model to learn and understand the problem.

```
scaler = MinMaxScaler(feature_range=(0, 1))

scaled_x = (pd.DataFrame(
    df['timestamp']).join(pd.DataFrame(scaler.fit_transform(x)))).to_numpy()
scaled_y = scaler.fit_transform(y)

x_train, x_test, y_train, y_test = train_test_split(scaled_x, scaled_y,
    shuffle=False, random_state=42, test_size=0.3)
```

The model was trained using 20 epochs and a batch size of 32. Additionally, it was given the size of the *validation dataset* (20% of training dataset). The validation dataset can provide some helpful information to optimize hyperparameters. It can also be an indicator of overfitting.

```
model.fit(x_train[:, 1:].astype('float64'), y_train, epochs=20, batch_size=32,
    verbose=2, validation_split=0.2, callbacks=[callback])
```

After the training, we can plot the *loss* and the *validation loss* through the epochs:

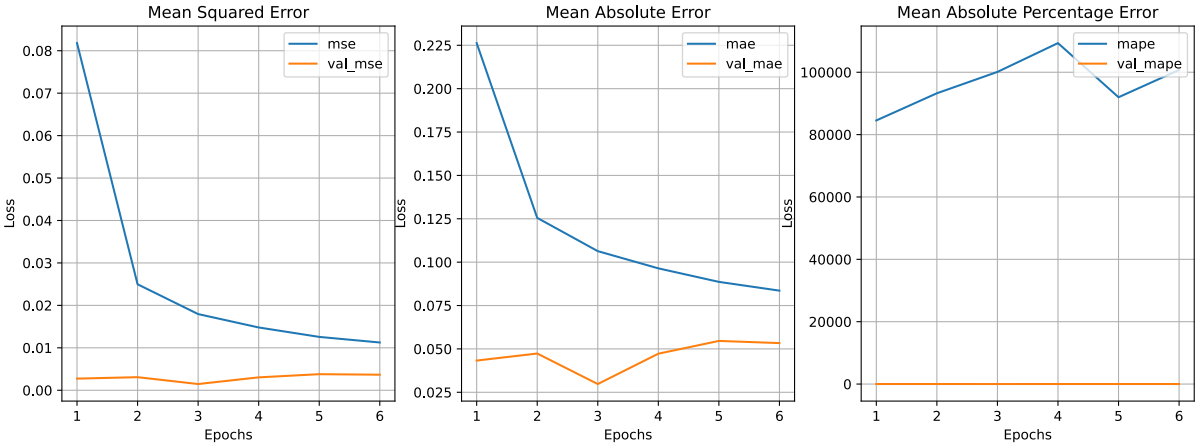


Figure 9: Loss and Validation Loss through the epochs

As we can see from Figure 9, the *loss* decreases as the epochs increases. In this case, the validation loss is lower than the training loss: it indicates that the validation dataset is easier for the model to predict than the training dataset. The model was then used to predict the price for the 12 hours ahead of using the custom function. The result was saved unscaled.

2.2.4 Evaluation

Using the test set, we were able to see graphically how much our predictions differ from the ground truth.

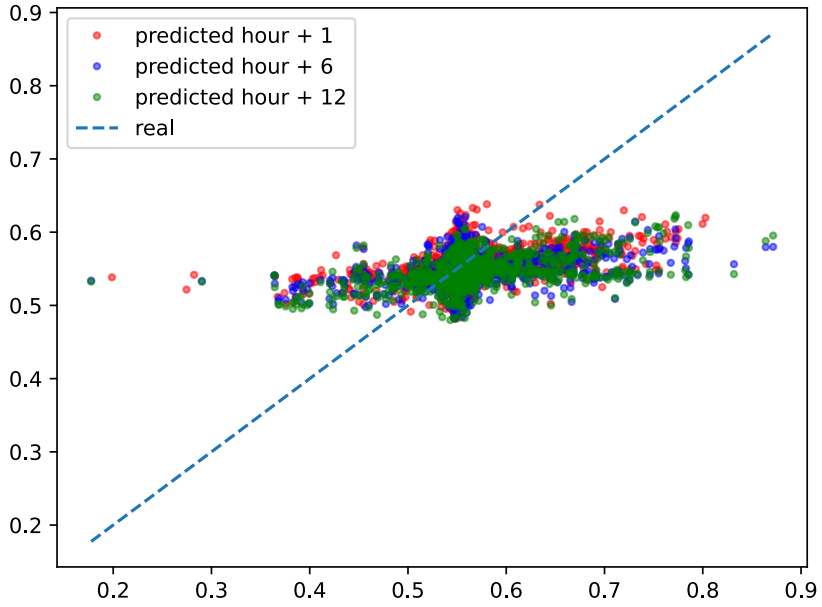


Figure 10: Comparison between the real values and the predicted values

As we can see from Figure 10, the prediction does not strictly follow the actual values. They are concentrated in one point, between 0.5 and 0.6. This means that the LSTM nearly always predicts the same value. However, if we analyze Figure 9, the mean squared error used as loss function in the model is decreasing.
As we said before, since this is a time series strictly correlated, we cannot do k-fold cross-validation to obtain a more general evaluation. Another evaluation method used on every model is the Root Mean Squared Error: a frequently used measure of the differences between predicted and actual values.

```
rms = np.sqrt(mean_squared_error(y_test, preds))
```

The result for the first model was 0.059002225233545286. The lower the RMSE, the better the model is able to fit a dataset. In conclusion, the LSTM model is a good predictor and an ideal choice to solve our problem.

2.2.5 Hyper-parameter tuning

Not yet satisfied with the result, we decided to use the Keras Tuner framework to perform a better hyperparameter search to create a model with better performance. The KerasTuner is an easy hyperparameter optimization framework that finds the best hyperparameter for a given model. It provides many different algorithms to conduct the hyperparameter search: we used the Hyperband.
The first step was to define which parameters could the tuner set:

- The number of units in the first LSTM layer
- The Dropout rate in the first Dropout layer
- Number of additional LSTM layer with their units and additional Dropout layer with their rate
- The activation function (relu or sigmoid) in the last Dense layer
- The learning rate of the model

```
def hypermodel_builder(hp):

    model = Sequential()
    model.add(LSTM(units=hp.Int('units_first_layer', min_value=32, max_value=512,
    ↪  step=32), return_sequences=True, input_shape=(50, 1)))

    model.add(Dropout(hp.Float('Dropout_rate_first_layer', min_value=0.1,
    ↪  max_value=0.5, step=0.1)))

    for i in range(hp.Int('n_additional_layers', 1, 4)):
        model.add(LSTM(units=hp.Int('units_add_layer', min_value=32,
        ↪  max_value=512, step=32), return_sequences=True))

        for j in range(hp.Int('n_additional_dropout_layers', 0, 1)):
            model.add(Dropout(hp.Float('Dropout_rate_add_layer', min_value=0.1,
            ↪  max_value=0.5, step=0.1)))
```

```
13
14      model.add(Flatten())
15
16      model.add(Dense(12, activation=hp.Choice('dense_activation', values=['relu',
        ↪  'sigmoid'], default='relu')))
17
18      model.compile(optimizer=Adam(lr=hp.Choice('learning_rate', values=[1e-2, 1e-3,
        ↪  1e-4])), loss=['mean_squared_error'], metrics=['mse', 'mae', 'mape',
        ↪  RootMeanSquaredError()])
19
20      return model
```

Once that we've defined the model, we can run the tuner. The KerasTuner takes as input:

- The model
- The objective function to minimize
- The maximum number of epochs to be tried
- A factor used to calculate the number of trials
- The directory in which to save the results

```
1 tuner = kt.Hyperband(hypermodel_builder, kt.Objective("root_mean_squared_error",
  ↪  direction="min"), max_epochs=30, factor=3, directory='output',
  ↪  project_name='LSTM 3.0')
```

Now everything is ready to start the tuner. We have added a callback to stop the tuner when the validation loss increases for three models in a row.

```
1 callback = EarlyStopping(monitor='val_loss', patience=3,
  ↪  restore_best_weights=True)
2
3 tuner.search(x_train[:, 1:].astype('float64'), y_train, validation_split=0.2,
  ↪  callbacks=[callback])
```

The trials went on for 3 hours; each trial lasted from a few seconds to 10 minutes. In the end, the best model was chosen with these characteristics:

- 384 units in the first LSTM layer
- 0.30000000000000004 dropout rate for the first Dropout layer
- 2 additional LSTM layer with 320 units
- 0 additional Dropout layer
- The sigmoid function as activation function
- 0.001 learning rate with the Adam optimizer

- 30 epochs

Once the model is defined, we can train it using the training dataset and plot the predictions against the actual values:

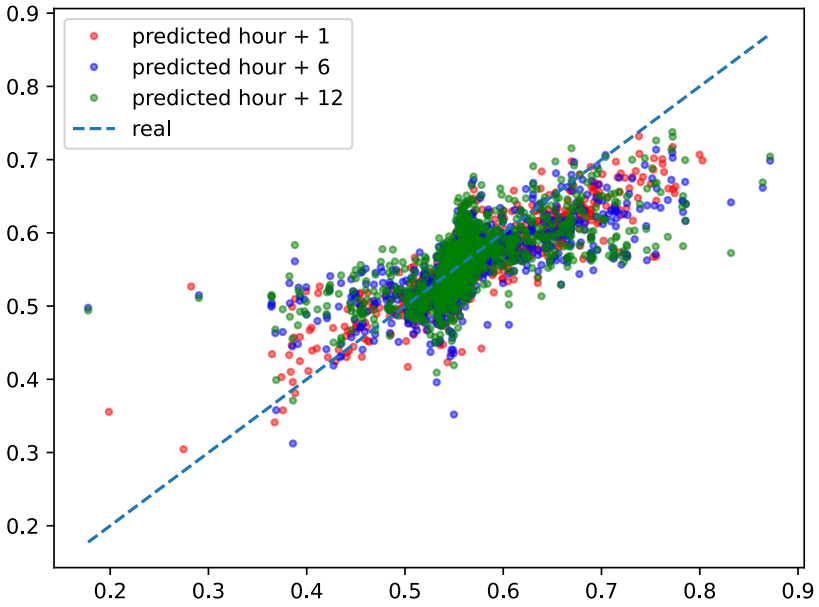


Figure 11: Comparison between the real values and the predicted values

As we can see from Figure 11, the newly trained model is much more accurate in its prediction than the previous one. The next hour energy price is more precise than the following hours: we can clearly say that the accuracy decreases over time. This phenomenon is expected and also experienced in the previous model.
Also for this model, we can plot the *loss* and the *validation loss* through the epochs:

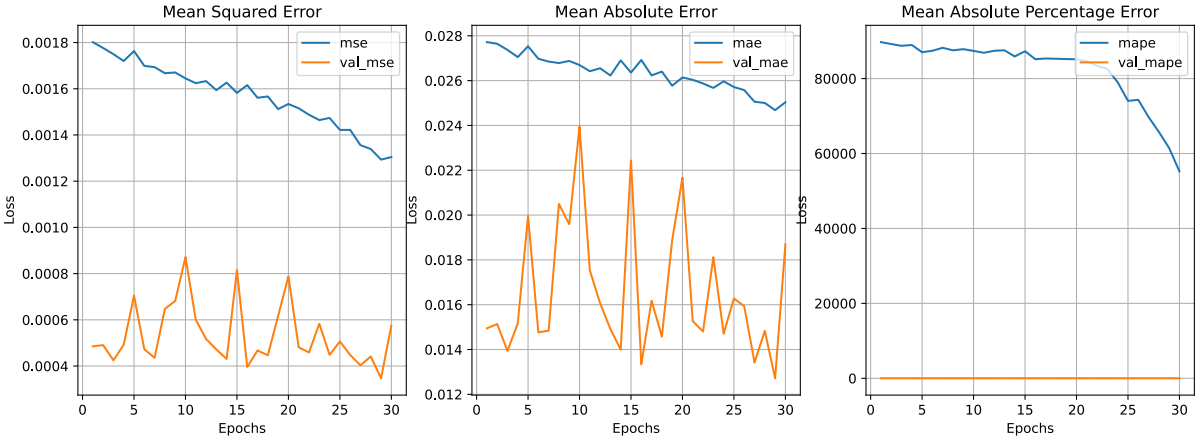


Figure 12: Loss and Validation Loss through the epochs using the hypermodel

As shown in Figure 12, there are some spikes in the validation loss.These spikes are standard and happen when the validation dataset is too small. However, the small size of the overall dataset does not allow us to use a more extensive validation dataset.
The plot also shows that the loss decreases through the epochs, indicating that the model is learning. The RMSE for this model was 0.042919002838567366, slightly lower than the previous model.
As Table 2 displays, the performance of the hyper model is slightly better than the performance of the base model. The tuner was able to find a model whose loss is lower than the one we have constructed before. Consequently, this model is more promising than the previous one, and its predictions will be used to perform the reinforcement learning stage.

Table 2: Summary Table

	Base Model	**Hyper Model**
Root Mean Squared Error	0.059002	0.042919
Mean Squared Error	0.003481	0.001842
Mean Absolute Error	0.040115	0.029098
Mean Absolute Percentage Error	0.070619	0.052231

3 Second Phase: Reinforcement Learning

3.1 Definition

Reinforcement Learning (RL) permits taking adequate actions to maximise the reward in a particular situation. Figure 13 shows the general architecture of an RL. An **agent** and its environment interact through a sequence of discrete temporal phases and, at each phase, the agent selects an action to transmit to the environment. Consequently, the agent obtains a **reward** and the environment changes in a **new state**. RL tries to build a map between the actions and the state to maximise total rewards based on the agent's knowledge, obtained through direct unsupervised interaction with the environment.
Thanks to the models' essence and to the fact that we do not need previous knowledge of the domain, RL has become a powerful tool to optimise the energetic system's control that has to face continuous adjustments; for example, intermitted availability of renewable resources, dynamic electricity prices and changes in the amounts of the energy consumptions.

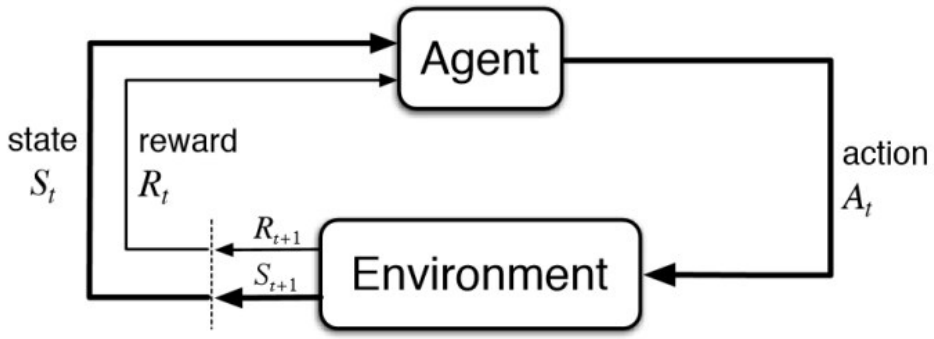


Figure 13: Setting of the Reinforcement Learning

3.1.1 Advantages of Reinforcement Learning

There are many advantages to using an RL to obtain an optimal decision process.
The Reinforcement Learning has the following properties:

Free of models The agents do not require a predefined rule or preliminary knowledge of selecting an action. Instead, it finds optimal actions directly learning while interacting with the environment.

Adaptive The agent can autonomously acquire optimal decisions in a form that can be adapted to different devices, considering the uncertainty and the flexibility of the EMS.

Coincided The entire calculation is based on a look-up table that is much more easily applied in the real world than traditional optimization methods.

3.1.2 Multi-Agent Reinforcement Learning

Multi-Agent Reinforcement Learning (MARL) is a complication of Reinforcement Learning and involves **numerous agents** that work in the same environment and often collaborate to achieve a final objective. The functioning of every single agent is similar to the one described in reinforcement learning. The characteristic, in this case, is the capacity of the environment to provide the required information to each agent; that is, the reward for the action performed and the new state of the environment as a result of that action.

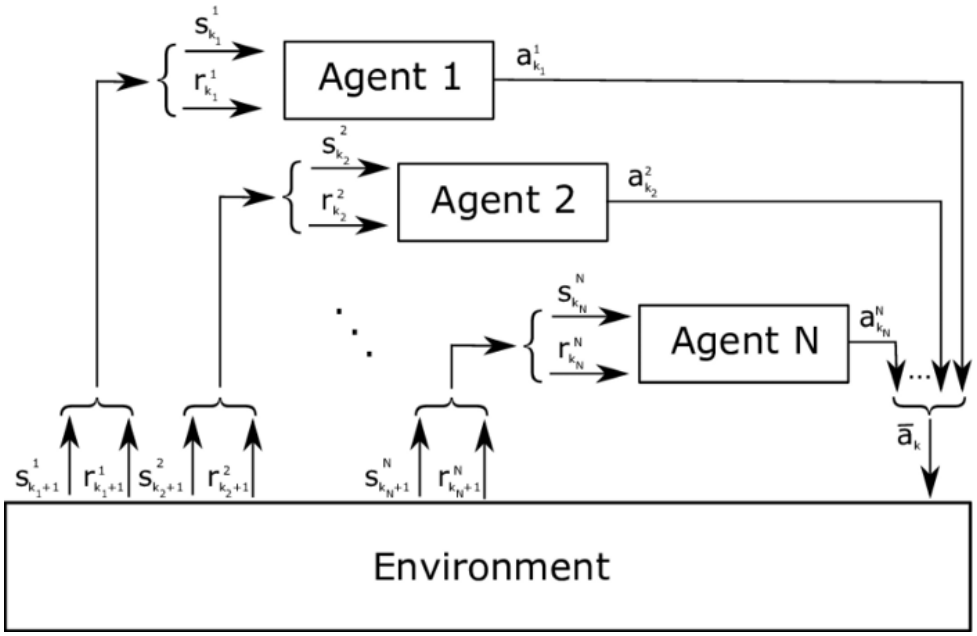


Figure 14: Setting of the Multi Agent Reinforcement Learning

3.2 Problem Formulation

3.2.1 Decision Making with MARL

To perform the optimal action, the RL problem is modeled as a discrete-time horizon through Markov Decision Making (MDP), which exhibits the Markov property that the state transitions depend only on the current state and the current action taken and, thus are independent of all previous environmental states and agents actions.

The **key elements** in this model include:

- a discrete hour h;
- an action a_h;
- a state s_h;
- a reward $r(s_h, a_h)$.

We use υ to denote the policy mapping states to actions, i.e., $\upsilon : a_h = \upsilon(s_h)$. The objective of this RL problem is to discover an **optimal policy** for every state s_h in order that the

selected action a_h maximizes the reward $r(s_h, a_h)$.
To obtain the optimal policy we use **Q-learning**. The mechanism behind Q-learning is the assigning of a Q-value $Q(s_h, a_h)$ to each state-action pair at hour h and updating of this value to each iteration, in order to optimize the result. The optimal Q-value $Q^*_\upsilon(sh, ah)$, denotes the maximum discounted future reward $r(s_h, a_h)$ when taking action a_h at state s_h, while continuing to follow the optimal policy υ, which satisfies the **Bellman equation** below:

$$Q^*_\upsilon(s_h, a_h) = r(s_h, a_h) + \gamma \cdot \max Q(s_{h+1}, a_{h+1}) \tag{1}$$

In which $\gamma \in [0, 1]$ is a discounting factor indicating the relative importance of future versus current rewards.
Each hour, an agent performs an action, and the Q-value of the corresponding cell is updated based on the Bellman equation, Eq. 1, as follows:

$$Q(s_h, a_h) \leftarrow Q(s_h, a_h) + \theta[r(s_h, a_h) + \gamma \cdot \max Q(s_{h+1}, a_{h+1}) - Q(s_h, a_h)] \tag{2}$$

In which $\theta \in [0, 1]$ is a learning rate representing to what degree the new overrides the old Q-values.
In a **multi-agent context**, each agent of the residential appliance acts independently to identify its optimal policy. During the learning process, each agent maintains its own Q-values, and reaches a policy based solely on the effects occurring in the environment caused by its own actions. Each policy is implemented as a separate Q-learning process with its own state space. When each agent reaches the optimal Q-value, all agents have obtained the maximum reward, meaning that the sum of the rewards is also at a maximum, and the system has reached the **global optimal Q-value**.
In the problem described, each household appliance has its own agent. The individual agents of the system can be modeled in the three types described below.

3.2.2 Non-shiftable load

Non-shiftable loads have critical demand requests that must be met during the energy distribution process, e.g., the **refrigerators** (REFGs) or certain alarm systems for security. Once a non-shiftable load begins operation, it **must work continuously**, and cannot be scheduled. The energy consumption of such loads is always equal to the energy demand:

$$E^{non}_{n,h} = e^{non}_{n,h} \tag{3}$$

The cost of this kind appliance is just the electricity bill of energy consumption. So, the utility function of a non-shiftable appliance n is:

$$U^{non}_{n,h} = P_h \cdot E^{non}_{n,h} \tag{4}$$

where P_h indicates the electricity price at hour h.

3.2.3 Shiftable load

Shiftable loads can **schedule their energy demand** to off-peak hours when the prices are low in the schedule horizon, so that not only peak energy usage is avoided, but also the energy bill is reduced. For example, the **washing machines** (WMs) usually operate

during the working period $[T_{n,ini}, T_{n,end}]$ and **must work continuously for a duration** $T_{n,ne}$, but the time of operation can be shifted from high electricity price periods to low price periods. Shiftable loads have two available operating points, "on" and "off". The energy consumption of such loads is the follow:

$$E_{n,h}^{shift} = I_{n,h} \cdot e_{n,h}^{shift} \tag{5}$$

where $I_{n,h}$ is a binary variable: $I_{n,h} = 1$ if the appliance works at hour h; otherwise $I_{n,h} = 0$. For this kind of appliance, there are two types of cost: the electricity bill of consuming energy, and the dissatisfaction of waiting time for a device to begin and then complete its operation. Thus, the utility function of a shiftable appliance n is:

$$U_{n,h}^{shift} = P_h \cdot E_{n,h}^{shift} + k_n \cdot (T_{n,w} - T_{n,ini}) \tag{6}$$

$$T_{n,ini} \leq T_{n,w} \leq [T_{n,end} - T_{n,ne}] \tag{7}$$

$$T_{n,ne} \leq T_{n,end} - T_{n,ini} \tag{8}$$

where $T_{n,w}$ denotes the operation starting time, so the waiting time is $T_{n,w} - T_{n,ini}$, and k_n is a device-dependent coefficient. In Eq. 6, the first term represents the electricity cost and the second term denotes the cost of waiting time.

3.2.4 Controllable Load

The controllable loads can be operated with a **flexible power consumption** between the minimum power demand and maximum power demand denoted by $e_{n,min}$ and $e_{n,max}$ respectively, i.e., the **air conditioners** (ACs) can regulate their power consumption from $e_{n,min}$ to $e_{n,max}$ in response to price changes. The energy consumption of such loads is the follow:

$$E_{n,h}^{con} = e_{n,h}^{con} \tag{9}$$

$$e_{n,min} \leq e_{n,h}^{con} \leq e_{n,max} \tag{10}$$

The objective of this kind of appliance should be to minimize the customer electricity bill by decreasing the power demand during the scheduling slots, however, the reduced power can cause dissatisfaction for the user. Hence, the utility function of a controllable appliance n is:

$$U_{n,h}^{con} = P_h \cdot E_{n,h}^{con} + \beta_n \cdot (E_{n,h}^{con} - e_{n,max})^2 \tag{11}$$

where β_n is a device-dependent dissatisfaction cost parameter. The first term denotes the electricity cost, and the second term indicates the customer dissatisfaction cost, which is defined by a quadratic function.

3.2.5 Objective Function

The objective function of the user is to not only minimize the electricity cost, but also minimize the dissatisfaction cost that can be expressed as follows:

$$\min \sum_{n=1}^{N} \sum_{h=1}^{H} \left\{ \left(1-\rho\right) \cdot P_h \cdot \left(E_{n,h}^{non} + E_{n,h}^{shift} + E_{n,h}^{con}\right) + \rho \cdot \left[k_n \cdot \left(T_{n,w} - T_{n,ini}\right) + \beta_n \cdot \left(E_{n,h}^{con} - e_{n,max}\right)^2 \right] \right\} \tag{12}$$

where the first term denotes the electricity cost, and the second term indicates the dissatisfaction cost. ρ is a balance parameter for achieving the trade-off between the electricity and dissatisfaction costs, which is determined by the user preference.

3.3 Problem Implementation

As an example of device integration in the HEMS, we discuss the integration of a plug-in electric vehicle in the formulated MARL model. The general implementation of the MARL model is then illustrated, and it is available at the following GitHub repository [3].

3.3.1 Integration of a plug-in electric vehicle

A **plug-in electric vehicle** (PEV) includes an internal battery that can be charged externally, such as at a charging station or by plugging it into a standard power outlet. Battery capacity has increased at an exponential rate over time, providing PEV drivers more and more autonomy.
To **integrate the charge of the PEV into the HEMS**, a strategy for managing the charge of a PEV using a modular power supply is presented below. As a result, it's critical to understand how to model the PEV using the three types of household appliances listed above (non-shiftable load, shiftable load, controllable load).

3.3.2 Types of modeling of the PEV

It is possible to model the PEV as if it were a charging battery connected to the home's electrical system. This means that the agent connected with the PEV takes into account some aspects that have yet to be established, such as the battery's **state of charge** and its **maximum capacity**. As a result, the agent in charge of deciding how to charge the PEV battery must adhere to the following constraints:

$$0 \leq SOC_{b,h} \leq C_b \tag{13}$$

$$0 \leq E_{b,h} \leq C_b - SOC_{b,h-1} \tag{14}$$

$$SOC_{b,h} = SOC_{b,h-1} + E_{b,h-1} \tag{15}$$

where b is the battery of the PEV, $SOC_{b,h}$ and C_b are respectively the state of charge of the PEV battery at the end of hour h and the maximum capacity of the PEV battery. $E_{b,h}$ is the power consumption demand of b at hour h.
All the models of the PEV battery examined are listed below.

Non-shiftable Load Because it can be charging as soon as it is connected to the home and stop charging when its state of charge reaches its maximum capacity or when the PEV is unplugged from home, the PEV battery can be modeled as a non-shiftable load. Taking into account the battery restrictions, battery power consumption as a non-movable load is as follows:

$$E_{b,h}^{non} = \min(e_{b,h}^{non}, C_b - SOC_{b,h-1}) \tag{16}$$

The utility function of the battery as a non-shiftable load is therefore as follows:

$$U_{b,h}^{non} = P_h \cdot E_{b,h}^{non} \tag{17}$$

This form of modeling ensures that the battery charge takes precedence, but it does not enable decisions to be taken to help lower the home's demand peaks. Given the burden that a PEV charge places on a home, it's a good idea to attempt modeling its batteries in a different approach.

Shiftable Load The PEV battery can complete its charge by scheduling it in the time interval $[T_{n,ini}, T_{n,end}]$. Battery power consumption as a shiftable load, taking note of the battery constraints, is as follows:

$$E_{b,h}^{shift} = I_{b,h} \cdot \min(e_{b,h}^{shift}, C_b - SOC_{b,h-1}) \tag{18}$$

The utility function of the battery as a shiftable load is therefore as follows:

$$U_{b,h}^{shift} = P_h \cdot E_{b,h}^{shift} + k_b \cdot (T_{b,w} - T_{b,ini}) \tag{19}$$

$$T_{b,ini} \leq T_{b,w} \leq [T_{b,end} - T_{b,ne}] \tag{20}$$

$$T_{b,ne} = \min\left(\left\lceil \frac{C_b - SOC_{b,h-1}}{e_{b,h}^{shift}} \right\rceil, T_{b,end} - T_{b,ini}\right) \tag{21}$$

Note that $T_{b,ne}$ is properly defined when the battery's state of charge and maximum capacity are known.
Shiftable load modeling assures that the battery is charged similarly to non-shiftable load modeling. The charge, on the other hand, does not take precedence because the shiftable load schedules the hours it will operate in order to charge the battery in the most efficient manner possible according to the MARL algorithm.
This modeling implies that $T_{b,end}$ is defined at hour $T_{b,ini}$ and that the battery must be connected to the home, in the worst case, throughout the entire interval $[T_{b,ini}, T_{b,end}]$ for the shiftable load decision-making process to function properly. As a result, it is possible that the PEV will be unavailable for a length of time longer than $T_{b,ne}$. This criticality is minimized by increasing the value k_b, but the energy savings are reduced.

If we are willing to accept this inefficiency in exchange for lower electricity bills, you can model the PEV battery in a different way (shiftable and divisible load), drawing inspiration from the shiftable load and using the possibility of dividing the energy demand of the battery; which it can do un-like shiftable loads like the washing machine.

Shiftable and Divisible Load The following type of modeling of the PEV battery makes sense to exist thanks to the possibility, by the battery, of programming and then postponing its charge, but also and above all thanks to the possibility of **dividing the charge into several distinct intervals**. This last property does not belong to the movable loads which, by definition, once started must operate continuously for the duration $T_{b,ne}$. This observation, therefore, allows modeling the PEV battery as a displaceable and divisible load. The whole availability of the time interval $[T_{b,ini}, T_{b,end}]$ is taken into account in this modeling.
The following type of modeling of the PEV battery makes sense to exist thanks to the possibility, on the part of the battery, to program and therefore postpone its charge, but also and above all thanks to the possibility of **dividing the charge into several distinct intervals**. This last property does not belong to the movable loads which, by definition, once started must operate continuously for the duration $T_{b,ne}$. This observation, therefore, allows modeling the PEV battery as a displaceable and divisible load.
This form of decision-making process does not involve multi-agent reinforcement learning, but rather a *naive* algorithm that depends simply on LSTM's predicted future prices. The algorithm's goal is to charge the battery during the hours of the interval $[T_{b,ini}, T_{b,end}]$ when electricity prices are lower than projected in the future. In this case is a must to know the energy consumption T_{ne} and the interval $[T_{b,ini}, T_{b,end}]$, and the complete availability of interval $[T_{b,ini}, T_{b,end}]$ is assumed.

$$E_{b,h}^{naif} = I_{b,h} \cdot \min(e_{b,h}^{naif}, C_b - SOC_{b,h-1}) \tag{22}$$

$$T_{b,ne} = \min\left(\left\lceil \frac{C_b - SOC_{b,h-1}}{e_{b,h}^{naif}} \right\rceil, T_{b,end} - T_{b,ini}\right) \tag{23}$$

Modeling the PEV battery as a shiftable and divisible load is an excellent choice when you want to reduce the cost of the bill and tolerate the unavailability of the PEV during $[T_{b,ini}, T_{b,end}]$.

Controllable Load The last type taken into consideration is the one that allows the PEV battery to be modeled as a controllable load. This type has greater flexibility than shiftable loads as regards the modulation of the energy consumption request, but less flexibility than shiftable and divisible loads. Compared to the latter, however, this modeling has greater flexibility regarding the availability of the PEV. In fact, by modeling the PEV battery as a controllable load, the PEV is not subject to any unavailability. There remains the disservice of the user present in the utility function.
The power consumption of the battery as a controllable load, taking note of the battery constraints, is as follows:

$$E_{b,h}^{con} = e_{b,h}^{con} \tag{24}$$

$$0 \leq e_{b,h}^{con} \leq \min(e_{b,max}, C_b - SOC_{b,h-1}) \tag{25}$$

The utility function of the battery as a controllable load is therefore as follows:

$$U_{b,h}^{con} = P_h \cdot E_{b,h}^{con} + \beta_b \cdot (E_{b,h}^{con} - \min(e_{b,max}, C_b - SOC_{b,h-1}))^2 \tag{26}$$

3.3.3 Q-learning algorithm

Q-learning algorithm with linear convergence is used in the MARL model implementation. This entails repeating the learning process a certain number of times (episode). The algorithm that takes advantage of this convergence is as follows (Alg. 1):

Algorithm 1: Q-learning algorithm with linear convergence

```
Initialize Q(s, a);
for each episode do
    Initialize s to initial state;
    while s != final state do
        Choose a from s using ε-greedy policy;
        Take action a, observe reward r(s, a) and next state s';
        Q(s, a) ← Q(s, a) + θ[r(s, a) + γ · max Q(s', a') − Q(s, a)];
    end
end
```

3.3.4 Problem Implementation Structure

Figure 15: Q-Learning Classes Diagram

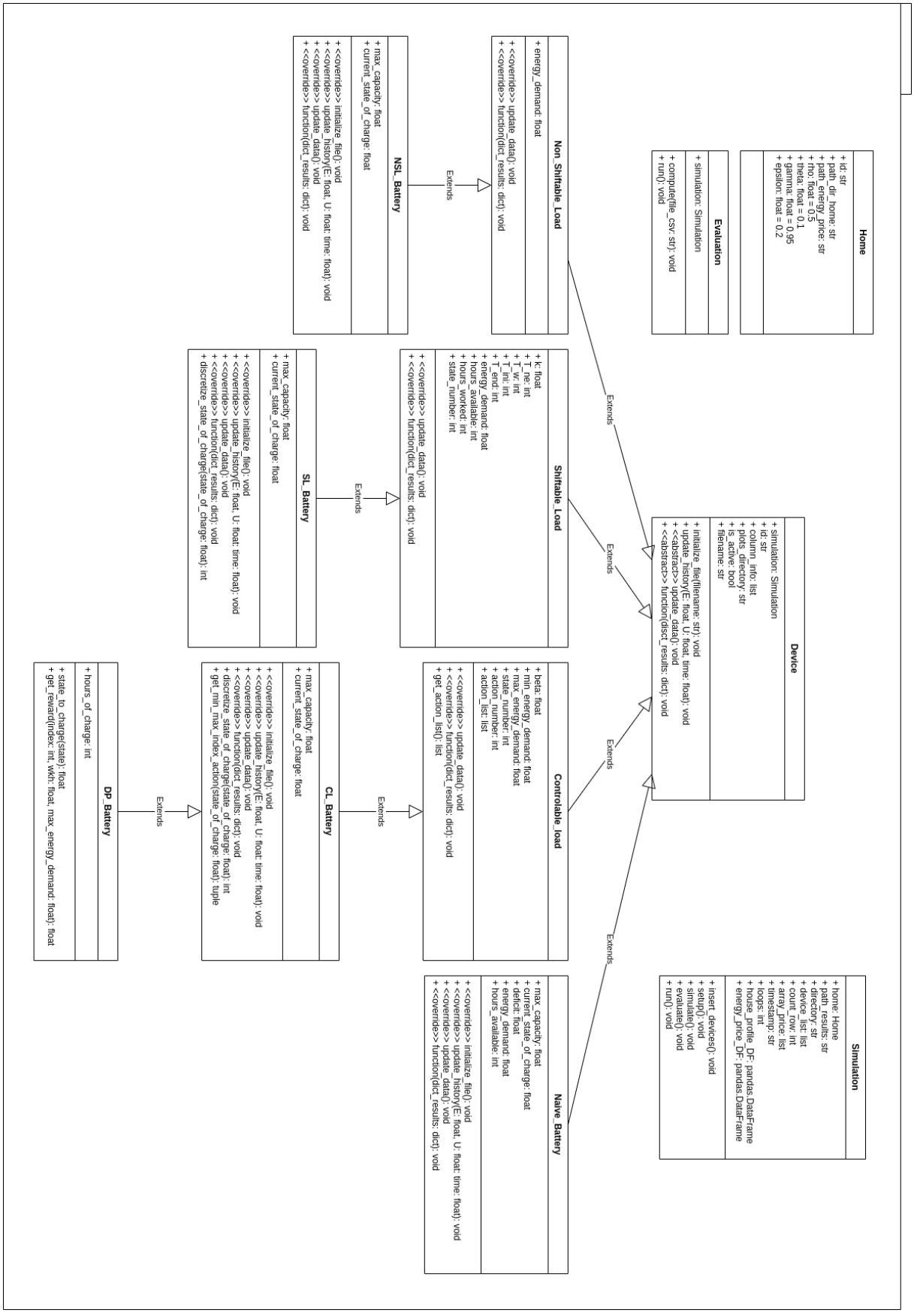


Figure 16: HEMS Classes Diagram

3.4 Evaluation

The Test-An-EV project [5] gathered information about the PEV's battery charge history. These are references to a model car [6]. The consumption data for the other devices managed by the HEMS are derived from the SmartHG project [7].
The objective of the evaluation is to analyze the results obtained by the algorithms that model the battery of the PEV. To do this, the HEMS described in this paper was carried out with the PEV integration.
The data described above are therefore simulated through the execution of the implemented HEMS.

Table 3: HEMS System Variables

ρ	θ	γ	ε
0.2	0.1	0.95	0.2

3.4.1 Evaluated Devices

The battery of the PEV examined was modeled with the types described in the Sec. 3.3.2. The Tab. 4 describes in detail all the models evaluated with their relative parameters. Of each model implemented, the parameters are as follows:

\# **states** number of states in models using MARL (Shiftable Battery and Controllable Battery models);

\# **actions** number of actions in models using MARL (Shiftable Battery and Controllable Battery models);

deficit indicates the terminal percentage of the battery capacity that the Naive Battery model will not fill to implement further energy savings;

β device-dependent cost-of-dissatisfaction parameter in the Controllable Battery model.

\# **epochs** number of epochs in models using MARL for linear convergence of Q-learning algorithm.

Table 4: PEV implementation models examined

ID	Typology	# states	# actions	deficit	β	epochs
NSL_Battery.0	Non-Shiftable Battery	-	-	-	-	-
Naive_Battery.0	Naive Battery	-	-	0%	-	-
Naive_Battery.1	Naive Battery	-	-	1%	-	-
Naive_Battery.2	Naive Battery	-	-	2%	-	-
CL_Battery.0	Controllable Battery	$24 \cdot 16$	4	-	0.00075	1000

3.4.2 Evaluation criteria

Four criteria are used to evaluate devices. These are calculated as the difference between the results obtained from the model with which the device to be evaluated is implemented and the results obtained from the same device, but without any algorithm. It should be noted that modeling a device in the absence of an algorithm is equivalent to modeling it as a non shiftable load. The valuation criteria are outlined in greater detail below.

Percentage difference in energy costs It represents the percentage difference between the cost of energy generated by the model with which the evaluated device is implemented and the cost of energy of the same device modeled without any algorithm. The formal phrase is as follows:

$$CE_b = \sum_{d=1}^{D} \sum_{h=1}^{H} P_{d,h} \cdot E_{b,d,h} \tag{27}$$

$$\Delta^{\%}_{CE_b} = \frac{CE_b^{non} - CE_b}{\max(CE_b^{non}, CE_b)} \cdot 100 \tag{28}$$

where CE_b denotes the cost of energy generated by the battery b as a result of the evaluated modeling; CE_b^{non} is the cost of energy generated by the same battery b as a result of modeling without an algorithm; $\Delta^{\%}_{CE_b}$ is the percentage difference in the cost of energy produced by the battery b as a result of the evaluated modeling.

Percentage difference in charged kWs It represents the percentage difference between the charged kWs by the model with which the evaluated device is implemented and the charged kWs of the same device modeled without any algorithm. The formal phrase is as follows:

$$KWC_b = \sum_{d=1}^{D} \sum_{h=1}^{H} E_{b,d,h} \tag{29}$$

$$\Delta^{\%}_{KWC_b} = \frac{KWC_b^{non} - KWC_b}{\max(KWC_b^{non}, KWC_b)} \cdot 100 \tag{30}$$

where KWC_b denotes the charged kWs by the battery b as a result of the evaluated modeling; KWC_b^{non} are the charged kWs by the same battery b as a result of modeling without an algorithm; $\Delta^{\%}_{KWC_b}$ is the percentage difference in charged kWs by the battery b as a result of the evaluated modeling.

Percentage difference in the average loading price It represents the percentage difference between the average loading price of the model with which the evaluated device is implemented and the cost of energy of the same device modeled without any algorithm. The formal phrase is as follows:

$$PMC_b = \frac{CE_b}{KWC_b} \tag{31}$$

$$\Delta^{\%}_{PMC_b} = \frac{PMC_b - PMC_b^{non}}{\max(PMC_b, PMC_b^{non})} \cdot 100 \tag{32}$$

where PMC_b denotes the average loading price of battery b as a result of the evaluated modeling; PMC_b^{non} is the average loading price of the same battery b as a result of modeling without an algorithm; $\Delta^{\%}_{PMC_b}$ is the percentage difference in the average loading price of battery b as a result of the evaluated modeling.

Percentage difference in average state of charge in output It represents the percentage difference between the average state of charge in output of the model with which the evaluated device is implemented and the average state of charge in output of the same device modeled without any algorithm. The formal phrase is as follows:

$$SOCMO_b = \frac{\sum_{p=1}^{P} SOC_{b,p}^{out}}{P} \tag{33}$$

$$\Delta^{\%}_{SOCMO_b} = \frac{SOCMO_b - SOCMO_b^{non}}{\max(SOCMO_b, SOCMO_b^{non})} \cdot 100 \tag{34}$$

where P is the number of plug-ins towards the home, made by the battery b; $SOCMO_b$ is the average SOC in output of the battery b as a result of the evaluated modeling; $SOCMO_b^{non}$ is the average SOC in output of the same battery b as a result of modeling without an algorithm; $\Delta^{\%}_{SOCMO_b}$ is the percentage difference in average state of charge in output of the battery b as a result of the evaluated modeling.

Score calculation It is possible to obtain a score that defines the overall performance of the model to be evaluated by using the four criteria described above. This will be useful for analyzing the evaluation results. Given a PEV's battery b, regardless of the type of modeling used, the formula for calculating the score is as follows:

$$Score = (1-\rho) \cdot (\Delta^{\%}_{CE_b} + \Delta^{\%}_{PMC_b}) - \rho \cdot (\Delta^{\%}_{KWC_b} + \Delta^{\%}_{SOCMO_b}) \tag{35}$$

3.4.3 Evaluation Results

This section examines the outcomes of the assessments performed on the various types analyzed, which shape the PEV battery.

Scenario with $\rho = 0.2$ In this scenario, the HEMS commits to reducing energy consumption costs by 80% while reducing user disservice by 20%. Tab. 5 summarizes the performance of the evaluated devices.

Table 5: Model evaluation results with $\rho = 0.2$

ID	$\Delta^{\%}_{CE_b}$	$\Delta^{\%}_{PMC_b}$	$\Delta^{\%}_{KWC_b}$	$\Delta^{\%}_{SOCMO_b}$	Score
Naive_Battery.2	33.38	7.41	-28.05	-16.12	23.80
CL_Battery.0	34.05	4.70	-30.80	-17.70	21.30
Naive_Battery.1	23.65	6.60	-18.26	-10.49	18.45
Naive_Battery.0	13.90	5.89	-8.51	-4.89	13.15
NSL_Battery.0	0	0	0	0	0

All devices perform better than NSL_Battery.0, which represents the device that does not use an algorithm to reduce energy consumption, with a propensity to save energy consumption compared to the user's disservice.
These results are in line with expectations. This scenario more tolerates the disruption caused by Naive Battery (with non-zero deficit parameter) and that caused by the Controllable Battery.
The Fig. 17 shows a charging cycle taken as an example.

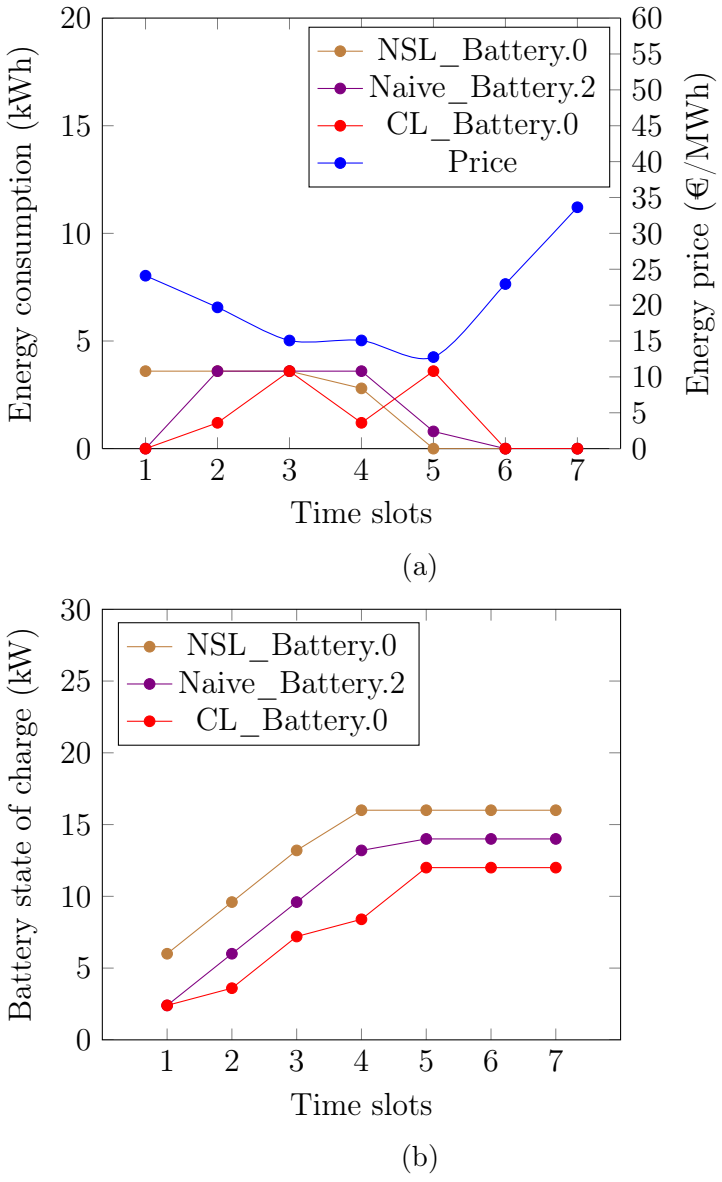


Figure 17: Battery PEV charge with $\rho = 0.2$

Here are some points to consider:

- NSL_Battery.0 charges the battery at full power as soon as the charge cycle begins, and continues to do so until the battery's state of charge reaches maximum capacity;
- Naive_Battery.2 charges in hours that are less expensive than the battery in terms of price;

- Due to the high cost of energy, CL_Battery.0 does not charge in the first time slot and modulates the demand for energy consumption in the following hours where the price is lower.

4 Conclusion

In this paper, we proposed an hour-ahead DR algorithm for HEMS, taking a machine learning approach with the aim of minimizing the user energy bill and degree of discomfort. Specifically, to overcome future price uncertainties, a steady price forecasting model based on LSTM is presented. In cooperation with the forecasted prices, multi-agent RL is adopted to make optimal decisions for different appliances. Simulation results showed that the proposed DR algorithm can handle the energy management and minimize the user energy bill and dissatisfaction costs.
As an example, the management of an PEV's decision-making processes has been integrated into the HEMS; one of the devices that most requires intelligent management of its energy consumption request. The evaluation results showed that the Naive Battery model is very versatile, but T_{end} must be known. If you don't know T_{end}, the only applicable model is the Controllable Battery model, which allows you to perform better than the Non-Shiftable Battery model in scenarios where the focus is on reducing energy costs.
Better models were discovered than the Non-Shiftable Battery model, which interpreted the PEV battery in the absence of a charging decision making algorithm. We can thus confirm that the elaborate has succeeded in developing models capable of better managing the charging process of a PEV at home, in order to achieve a more accurate and effective management of domestic electricity.
In the future, the PEV will be modeled using a Shiftable Load model and a model based on Dynamic Programming inspired by the knapsack problem. Other domestic devices will be implemented in order to be managed using the proposed MARL algorithm.

References

[1] *Understanding LSTM Networks*. 2021. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs.

[2] Renzhi Lu, Seung Ho Hong, and Mengmeng Yu. "Demand Response for Home Energy Management Using Reinforcement Learning and Artificial Neural Network". In: *IEEE Transactions on Smart Grid* 10.6 (2019), pp. 6629–6639. DOI: 10.1109/tsg.2019.2909266.

[3] *Machine Learning Project*. 2021. URL: https://github.com/deborahdore/Machine-Learning-Project.

[4] *Nord Pool*. URL: https://www.nordpoolgroup.com/Market-data1/Dayahead/Area-Prices/DK/Hourly/?view=table.

[5] *Test-An-EV*. URL: smarthg.di.uniroma1.it/Test-an-EV.

[6] *Mitsubishi i-MiEV*. URL: https://it.wikipedia.org/wiki/Mitsubishi_i-MiEV.

[7] *SmartHG: Energy Demand Aware Open Services for Smart Grid Intelligent Automation*. URL: http://smarthg.di.uniroma1.it.

[8] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[9] *Q-learning simulator*. URL: https://www.mladdict.com/q-learning-simulator.

[10] *pyqlearning Python library*. URL: https://pypi.org/project/pyqlearning.

[11] *scikit-learn: machine learning in python*. URL: https://scikit-learn.org.

[12] *Keras: the Python deep learning API*. URL: https://keras.io.

[13] *Machine Learning Project Repository on GitHub*. URL: https://github.com/deborahdore/Machine-Learning-Project.