Carbon Cut Technical Writeup

Arunim Agarwal[1], Owen Frausto[1], Malo Lahogue[2], and Rob Santos[3]

[1]University of Pennsylvania
[2]MIT
[3]Boston University

Keywords:
Artificial Intelligence
Climate Change
Decarbonization
Prediction

Technical Mission To develop an open-source resource for making high-quality building decarbonization characteristic predictions from simple and common data points. This writeup serves as documentation for our project, and a guide for future work. This project was started as part of the WattCarbon Challenge at the MIT Energy & Climate Hackathon in November 2023.

1. Approach

Train a machine learning architecture on the Department of Energy's ResStock dataset to learn:
- Given a load shape, what are the predicted carbon reductions for ten different prespecified decarbonization schemes?
- Given a load shape, estimate key characteristics about the house that are useful in the energy transition (square footage, EV usage, window type, etc.)

2. Data

We are using two parts of the ResStock dataset. We restricted ourselves to half the Massachusetts dataset to have reasonable data processing time, as the full US dataset weighs several terabytes.

On one side, we extracted the data about the loadshapes. The input was a time series with 15 min intervals, giving the moment-by-moment electric and gas consumption. These then were transformed into a matrix with two columns, electricity and gas, with rows corresponding to time, one per hour (averaged the energy consumption). The matrix X_1 is used as input to both our models.

On the other side, we extract metadata on the building. There is one file per upgrade, one row per building. On one side we created a vector X_2 which concatenates information on `['climate_zone', 'county', 'state', 'city', 'building_type']` (In order to combine tabular data with a machine learning scheme, we binary encoded all of the tabular data vectors). We then created labels Y_1, a 42-dim array, which has the CO_2 emissions for both electric and gas. Lastly, we created Y_2 concatenating more extensive data about the housings: `['sqft', 'nb_bedrooms', 'nb_floors','age', 'cooling_type', 'heating_type', 'nb_windows', 'pv_size', 'electric_vehicle']`.

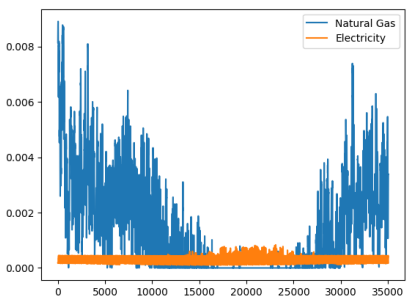

FIGURE 1. Example input data curve: timeseries data for electricity and gas.

3. Prediction Problem

We trained a two layer convolutional neural network (CNN) to process the two load curves and perform feature extraction. We flatten the output and append relevant metadata (simple information like the climate region and house type), which we then pass to a four layer feed forward neural network:

```
class CarbonReductionCNN(nn.Module):
    def __init__(self):
        super(CarbonReductionCNN, self).__init__()
        self.conv1 = nn.Conv1d(in_channels=2,
    out_channels=64, kernel_size=5)
        # Pooling Size
        self.p = 2
        self.pool = nn.MaxPool1d(kernel_size=self.p)
        self.conv2 = nn.Conv1d(in_channels=64,
    out_channels=128, kernel_size=5)
        # Adjust the input size based on your data
        self.fc1 = nn.Linear(128 * (((n // self.p - self.
    p) // self.p) - self.p) + m, 256)
        self.fc2 = nn.Linear(256, 128)
 self.fc3 = nn.Linear(64, 128)
        self.output_layer = nn.Linear(42, t)


    def forward(self, x1, x2):
        x = self.pool(F.relu(self.conv1(x1)))
        x = self.pool(F.relu(self.conv2(x)))
        x = torch.flatten(x, 1)
        x = F.relu(self.fc1(torch.hstack([x, x2])))
        x = F.relu(self.fc2(x))
 x = F.relu(self.fc3(x))
        x = self.output_layer(x)
        return x
```

4. Model Training

We implemented the model in PyTorch and trained using an Nvidia V100 GPU via Google Colabs Cloud Computing service.

Our model's performance was evaluated based on its ability to accurately predict the outcomes of the test data, which was not used during the training phase.

Before training, the dataset was randomly divided into a training set and a test set. The training set was used to train the model, while the test set was used to evaluate its performance. Data preprocessing steps, such as normalization and handling of missing values, were applied to ensure the quality of training, as well as task-specific issues tackled earlier in this writeup.

The model was trained over 50 epochs, and we found that both the

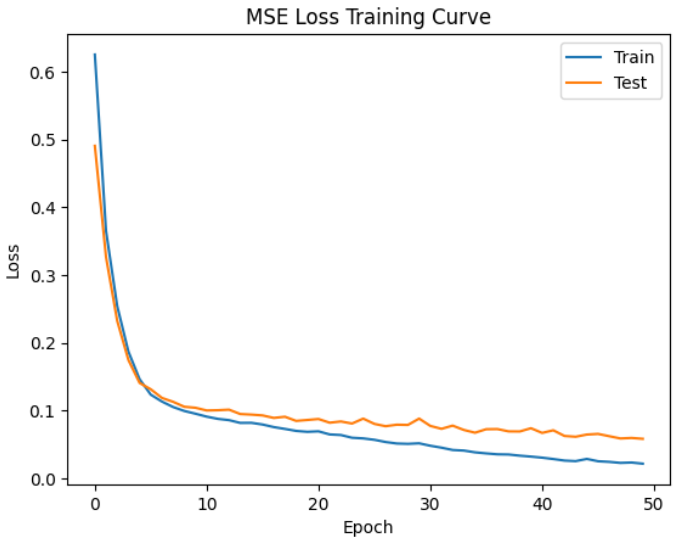

FIGURE 2. Loss curve.

mean-squared error and median percent accuracy converged to reasonable values (0.067 and 11% respectively) over the course of training. These results are promising, especially given the relatively small sample size which arose as a result of computing and memory constraints.

5. Estimation Problem

In order to extract information from the submitted curves, we use the pre-trained CNN layers from our previous step to perform compression/feature extraction on our load curves. By passing the load vectors of each house in the dataset through this CNN, we get representations in feature space that we can compare future representations to. Specifically, we use K-Nearest Neighbors (KNN) to find the most similar outputs and then aggregate the associated metadata. The challenge here lies in finding an optimal n for the clustering of each of the features that we want to estimate, and while we have code for the estimation algorithm, we haven't implemented a training step or finished connecting it to our data pipeline. The code is:

```
# Extract features from the model for each 'h' vector in the list of dictionaries
model_features = []
ground_truth_labels = []

with torch.no_grad():
    for sample in reference_houses:
  # The unsqueeze treats it as a batch of size 1
        h_vector = torch.tensor(sample['h']).unsqueeze(0)
        features = model(h_vector)
        model_features.append(features.numpy())
        ground_truth_labels.append(sample['m'].numpy())

model_features = np.vstack(model_features)
ground_truth_labels = np.array(ground_truth_labels)
print(model_features.shape)

# Use KNeighborsRegressor for k-nearest neighbors regression
knn_regressor = KNeighborsRegressor(n_neighbors=5)  # We need to train this (k)
knn_regressor.fit(model_features, ground_truth_labels)

# Extract features from the model for the new input
with torch.no_grad():
    new_features = model(x1.unsqueeze(0)).numpy()

# Use k-nearest neighbors regression to predict the associated parameter
predicted_parameter = knn_regressor.predict(new_features)
```

6. Open Source

We'd like this to be an accessible tool to enable the energy transition. To do this, we're providing links to all of our code:

- Data Loading and Prediction Model
- Estimation Model
- Trained Weights for Prediction Model (.pt weights file)
- Proccessed Dataset (X1, X2, Y)

We should note that we're still actively working and experimenting in the shared colab notebooks, so the model architechtures may not line up perfectly with what's outlined in this paper.
We think an impactful forward step is deploying this model as a Python package, lowering the bar of technical expertise and computational resources required to assess decarbonization options substantially for everyone. We hope that this approach would also be able to catalyze further improvements in the open-source carbon impact prediction ecosystem.

7. Areas of Future Work

There are several promising areas of future work, several promising avenues present themselves. First, we'd like to expand our research to include the ComStock dataset, which offers a wealth of information on commercial buildings that could significantly broaden the scope of our analysis and predictions. Secondly, we want to enhance the robustness of our model by training it on the entire country's data (or at least a sampled of it). This would increase the volume of data at our disposal and allow a more comprehensive and representative sample, improving the reliability and generalizability of our findings. Lastly, we want to incorporate pricing models into our current framework to predict the cost of implementing upgrades. This will provide valuable insights into the financial implications of different upgrade scenarios, aiding stakeholders in making informed, cost-effective decisions. These areas promise to provide a more holistic, wide-ranging, and practical understanding of our research subject.