AI-VVO (Artificial Intelligence Volt-VAR Optimization)

FINAL REPORT

sdmay22-36

Client/Advisor: Dr. Gelli Ravikumar

Jaden Alamsya
Evan Dinnon
William Dulaney
Demetrius Christou
Derrick Vang
Megan Phinney
Rachel Owens

sdmay22-36@iastate.edu
http://sdmay22-36.sd.ece.iastate.edu/

Table of Contents

Abstract

This project aims to solve the longstanding problem of volt-VAR control in smart grid systems with a high penetration of Distributed Energy Resources (DERs). The advent of residential solar and other forms of DERs have led to problems with voltage profiles at consumers' loads. Machine learning algorithms can be used to negate these problems by optimizing the real power delivered to the consumers while minimizing the reactive power transferred by engaging control mechanisms in a SCADA-connected smart grid. We have created an application that displays a map of the SCADA-connected smart grid, runs simulations in OpenDSS based upon changes in control mechanisms, and pushes the results to the backend to be processed by the machine learning application and displayed by the front end. This application is intended to be used by grid operators and power companies to improve power factor and reduce costs. By optimizing our algorithm to push the reactive power delivered to all buses as near to zero as possible, we have created a machine learning solution to the volt-VAR control problem.

1. Project Design

1.1. Evolution from 491

From our design in 491, as we began the implementation process, we realized that there were some deficiencies with our design. One design deficiency that we aimed to rectify was that we weren't storing nodal relationships in the backend for display on the front-end grid visualization map.

1.2. Functional Requirements

The functional requirements on this project were the following:

- Map-based grid display that shows interconnection of nodes in the smart grid

- Create an algorithms page on the frontend GUI display
- Deploy the application using Docker containerization
- Implementation of a time-series database for storage of grid values in a backend server
- Machine learning algorithm to engage the SCADA-connected devices in order to optimize power delivery and reduce reactive power flows

1.3. Non-functional Requirements

- Aesthetically pleasing GUI and frontend design
- Consistent styling in code files to ensure readability
- Large quantity of comments in code and script files

1.4. Engineering Standards

For the project, we utilized many different industry standards to ensure our product was in keeping with current industry standards and expectations.

- IEEE 26514-2010 Requirements for Designers and Developers of User Documentation - This standard applies to our project as there will be a lot of programming done and a proper plan as to how we should design our project will be important. Documentation is also very important to our project since this project has been running for multiple semesters and proper documentation will help future groups improve upon our work.
- IEEE 1250-2018 IEEE Guide for Identifying and Improving Voltage Quality in Power Systems - This applies to us since we need to be sure that our simulated values for our voltages are realistic to use in the real world otherwise our simulation would not be very useful.

- IEEE 1854-2019 IEEE Trial-Use Guide for Smart Distribution Applications - These standards give our project a clear definition of the components and functions it needs to be successful.
- IEEE 1028-2008 IEEE Standard for Software Reviews and Audits - This will help us review and check code before it is pushed to try to limit the amount of bugs that get to the main branch of the code base.
- IEEE 1547-2018 IEEE Standard for Interconnection and Interoperability of Distributed Energy Resources with Associated Electric Power Systems Interfaces

1.5. Constraints

The constraints are the fact that we had to use scripting to visualize data, plots and analytics. There were computing constraints for training the Deep Q network. Another constraint was that our team needed to utilize and build on the previous team's codebase. An additional constraint was that the VMs had resource limitations such as processor core count and the quantity of volatile memory available.

1.6. Security Concerns and Countermeasures

All development was done through Docker in order to assuage any security concerns. Docker is a secure containerization platform that can be used for streamlined, software development solutions. All development was done on secure VMs and Linux was utilized since it is a significantly more secure platform than Windows.

Implementation Details

2.1. Front-End

- **Overall Design**

  For the frontend design we implemented an About page containing information on the project, a Registration page for new users to set up accounts, and a map to overlay new grid data on top of. We also began setting up a new display of the new Machine Learning Algorithms output which is currently blank. We decided to use ReactJS for the frontend design. ReactJS is a component based frontend development language that worked nicely with the other tools we were using. The previous year's team set everything up in ReactJS meaning we did not have to refactor their work. We used axios libraries for corresponding with the backend. Axios is a promise based HTTP client that works well with ReactJS. It has a great support community and lots of documentation making setup simple. We used react-leaflet libraries for developing the map. React-leaflet worked great with ReactJS and allowed us to drop in a Map component.

- **About Page**

  This page includes some general information about the goal of the project, the way it can be used to achieve that goal, and information about our group.

- **Registration**

  The registration component utilizes Axios functions to create new users and post that data to the backend. The login page then uses that information by retrieving users and validating against user input. This allows new users to create new profiles and separate access to the application.

- **Map**

  For the map implementation we used OpenStreetMaps and integrated into our project using the react-leaflet library. React-leaflet had a lot of helpful functionality that allowed us to display grid data on top of the map. We used polylines for bus lines and custom SVG images for the nodes. We used axios functions to post and get data to and from the backend. Using the Axios GET functionality, we were able to fetch all the bus and node coordinate data and build lines and nodes to display on the map. We implemented the post functions on custom markers on the map that represented nodes and bus lines. Clicking on lines and nodes has different functionality, each sending and updating data in the backend.

- **Machine Learning Algorithm Page**

  The Machine Learning Algorithm page is not completed, but we have outlined the initial setup. Once the machine learning algorithms are ready to be pulled into the frontend and display data, the website will send a command to run the machine learning loop and display the capacitor state, transformer tap, and the switch state. This page will also utilize Axios functions to post and get data from the backend for display on the website.

2.2. Back-End

- **Overall Design**

  For our backend we simply expanded upon the previous team's work. The previous team already had a Django backend server up and running but it had some issues with connecting to the postgres server that holds user data. After

fixing any leftover bugs from the previous team we spent time expanding the api endpoint that django supported. We implemented various http endpoints that include getting power, current, voltage, node coordinates and, node connections to post requests like switch_update which will toggle the on/off value of a switch which affects the machine learning algorithm.

- **Influx Database**

  The influx database was created to hold the real time data about node voltages that was given to our team by our client. Since Influx specializes in holding real time data it was the obvious choice to hold onto our real time data. In order to streamline the data uploading process we created several python scripts that parse the data from the CSVs, clean the data then upload it to the influx database. That data can then be retrieved through get requests on the django backend and displayed on the frontend.

- **Neo4j Database**

  The Neo4j database was created to hold the node data representing the various bus nodes in the electrical grid and how they all connect to each other. Since Neo4j is a graph based database it was very easy to store and retrieve the data. Python scripts were also created in order to make the data upload process very easy. This data could again be retrieved from a get request on the django backend as a csv which will be parsed by the front-end team and displayed on the map page of the application.

- **OpenDss Server**

Due to technical constraints we were unable to get OpenDss to work while running on the linux VMs that the rest of the project has been tested on. This meant that a VM running Windows needed to be created in order for our Machine Learning team to use OpenDss for their algorithm. Since the output from opendss was needed to be displayed on the frontend we needed to create a server to get data from opendss to the Aivvo application. We ended up creating a simple python http server that would run on the windows VM and relay data back to the frontend. This allows quick and easy communication between the Machine Learning algorithm and helper scripts and the frontend in order to make for a responsive application.

2.3. Machine Learning Algorithm

- **Overall Design**

  For our machine learning algorithm we took a similar approach to the previous team in using a reinforcement learning algorithm. However, we opted to use a deep Q-learning algorithm instead of a more direct approach. Using Q-learning along with a neural network allows for faster convergence of our algorithm as well as greater accuracy. The two main components of our algorithm are the environment and the agent. These will be described in further detail below. To obtain real data from a true grid we utilized OpenDSS grid simulation software to test and run our algorithm on a real simulated grid.

- **Convolutional Neural Network**

  For the machine learning algorithm, all of the literature pointed to convolutional neural networks seeing the fastest performance gains in recent years. It was

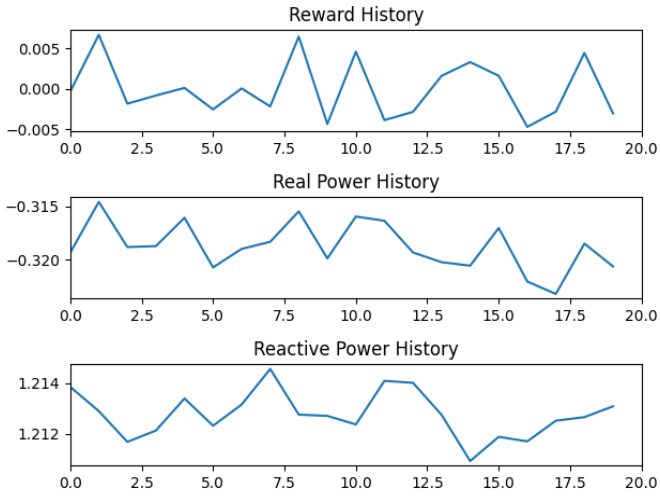
decided to pursue this approach over the previous team's Pandas implementation, since that was more limited in terms of speed and accuracy. Our team's PyTorch implementation of a convolutional neural network aims to solve the speed issues present in the Pandas traditional reinforcement learning approach taken by the previous team to work on the project.

- **Environment**

  A reinforcement learning environment contains the base definitions needed for training. The environment contains details on the action space, observation space, as well as how the agent class will step through. Most importantly, the environment defines the reinforcement learning algorithms reward function. For our algorithm, we defined the action space as the state of each capacitor bank, state of each switch, and the tap position of each regulator. This means that at each step of the training process our algorithm will change one or more of these values in an effort to increase the reward. In our case, the reward was defined by the change in reactive power and real power summed together. There was an emphasis placed on the real power as this was deemed to be of greater importance. A large positive reward indicates a "good" action. The observation space of the algorithm was defined as the real and reactive powers along with the current states of the action space. The environment at each step updates the OpenDSS script files corresponding to the action provided. It then will update the observation space by running OpenDSS on the new values.

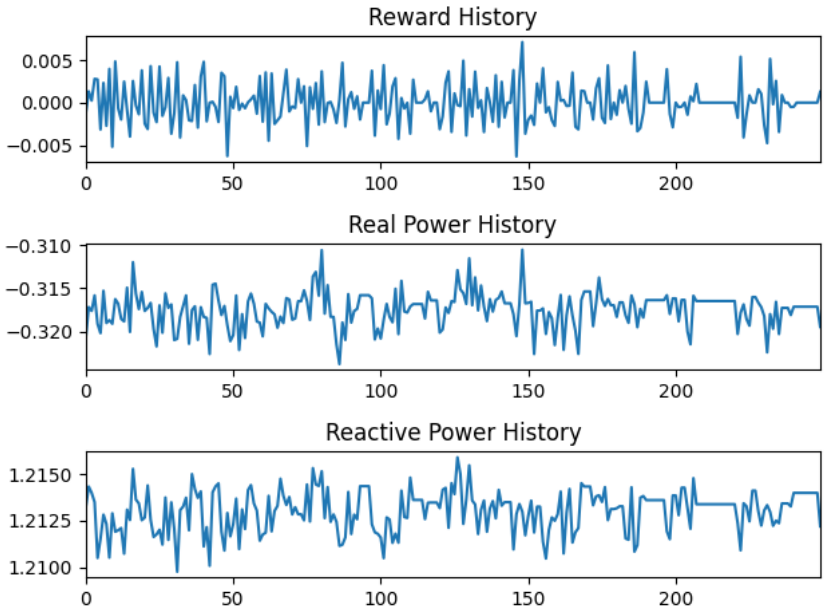
- **Agent**

The agent class is the brains of the reinforcement learning algorithm. The agent steps through the environment recording the reward and learning the best action to take. Initially, we started with a “brute force” algorithm for comparison. This algorithm simply takes random actions until the rewards sum to a given value or a certain number of iterations is reached. We used this to show the speed and efficiency that neural networks have to offer. Below shows a graph of the rewards, real, and reactive powers over each iteration.



From this we can see the sporadic change in reward after each iteration when random actions are taken. Moving on from this, we implemented Q-learning algorithm to more efficiently and effectively train the algorithm. Q-learning is a model-free reinforcement learning algorithm that finds an optimal action based on maximizing the expected reward. Deep Q-learning makes use of neural networks to optimize this algorithm. For our algorithm, we made use of the Pytorch library to build this algorithm. After running a number of steps through the environment, the algorithm is able to learn the probability that a certain action will provide a higher reward. We can see below that using Q-learning and neural networks we

can achieve greater results.We notice towards the end of the run the reward begins to flatten out.



The agent class is initiated by a call from the application webpage. Upon completion the environment data over the time spent learning is sent to the front-end to be displayed. We are then able to view the grid on the map with the now optimized environment.

Testing

2.4. Testing Process

We implemented testing schemes to test both the functional and non-functional components of the project. The functional requirements were easier to test because there are software packages available that allow unit testing of functional components of the project. The non-functional components such as design and aesthetics were tested by showing our client and advisor demos of our project during weekly meetings. Our client and advisor provided us with swift feedback

to direct the design in the correct direction. Overall our project focused on utilizing unit tests and integration tests in order to ensure we had a properly working application.

- **Unit Tests**

  Unit Tests make up a large portion of the tests we implemented in our project. That is because they are easy to set up and take relatively little time and resources to run. For example our Django backend uses many unit tests in order to make sure all of the functionality of the code works no matter what new features are done. It also allows our team to quickly find where bugs may be located all while preventing us from pushing code with many bugs to the main branch of the application. Our overall process for creating new unit tests was very straightforward. Whenever new features were added unit tests would be made so that each function that makes up the feature is tested to ensure that as many lines of code as possible were being tested.

- **Integration Testing**

  Integration testing made up the other portion of our testing scheme since it allowed us to test the application when it was actually running. Some of the integration tests would encompass multiple parts of the application at once since some tests on the backend would test the backends communication with both the frontend and machine learning aspects of the project. Looking at the backend as an example again when creating integration tests the process was very simple. Whenever a new api

endpoint needed to be created new integration tests were also made testing the results and response from the actual endpoint in order to test the different possible errors that could occur along with testing to make sure a successful request could be made to the server.

2.5. Testing Results

The testing overall helped to make sure that we had a highly functional codebase that was able to avoid major bugs. Testing also helped to make sure that multiple error paths were identified for example when the frontend sends a post request to the backend in an attempt to update a switch there are many places where the request could fail. It could fail early on when parsing the post data or it could fail at the very end when the updated data is being sent back to the frontend. Without testing the code it would be very hard to identify where all these possible error points are.

Context

In SCADA-connected smart grid systems, a high penetration of Distributed Energy Resources (DERs) is becoming commonplace for utilities. DERs are important in ushering an environmentally conscious and forward-thinking future, but they can cause voltage profile problems if not compensated for properly. This machine learning application reduces the reactive power transferred in smart grid systems by utilizing deep Q-learning. This improves the system's power factor and cost-effectiveness at delivering electric power. Volt-VAR control is a highly active area of research due to the fast adoption of DERs and distributed generation sources.

Appendices

Appendix I: Operation Manual

How to run the application:

Prerequisite:

- Clone the project from GitLab if you have not already

Running the Application:

- Open a new terminal instance
- **Navigate into the "sdmay22-36" project directory that you cloned**
- Run the command "sh aivvo-start.sh"
    - If you run into any errors run “reboot” in the terminal instance and wait for your VM to load
- **Navigate into the "frontend" directory**
- **Navigate into the "react_app" directory**
- Run the command "npm start"
    - If you run into an error run “npm install” to make sure you have the right packages installed to run the application

Stopping/Updating the Application:

- When back-end changes are made you need to navigate back to the **"sdmay22-36"** project directory and run “sh aivvo-stop.sh”, then run “git pull” to obtain the updated back-end
    - Again if you run into any errors run “reboot” in the terminal instance and wait for your VM to load

- Running “sh aivvo-stop.sh” in the **"sdmay22-36"** project directory will stop the application

Features:

About Page:

- This page includes some general information about the goal of the project, the way it can be used to achieve that goal, and information about our group.

Registration Page:

- Users can use this page to register as a new user with a username and password on the application

Update Password Page:

- Users can update/change their passwords for their profile on this page

Map-based Grid Display:

- Bus nodes
    - Every node is initially displayed onto our map with specific data shown as popups for each node. This feature can be selected to be shown or hidden.
- Bus connections
    - Every connection between buses is also initially displayed onto our map with specific data shown as popups for each connection. In connections associated with Switches, users can flip switches on or off. This feature can be selected to be shown or hidden.
- Node types

    - We were able to differentiate between 3 node types: Transformers, Switches and PV nodes. This feature allows users to view specific node types by selecting it to show all nodes of the desired type.
- Voltage Information
    - This feature displays every node's specific voltage information. Users can select this feature to be shown or hidden.
- Current Information
    - This feature displays every connection's specific current information. Users can select this feature to be shown or hidden.

Appendix II: Other Versions

Before sdmay22-36 worked on this project, sdmay21-24 contributed to this project. Their version utilized the Numpy and Pandas libraries in Python to implement a reinforcement learning algorithm. We explored expanding their approach, but found it inadequate to handle the large data sets required for such an application. We switched from Numpy to PyTorch as PyTorch is a far more modern and streamlined machine learning framework. Originally, there was an ASCII style map that showed every power bus and the nodal connections between them. Our new version utilized OpenStreetMaps so we could overlay all of the buses on an actual map with their real geographic locations. This provided a far more user-friendly experience in the application.

Appendix III: Other Considerations

Lessons Learned

Jaden Alamsya: Through this project, I learned much more about working with React. I also got a lot more experience working with a team on a larger scale project. I also learned the

importance of documentation and how the lack of it can completely halt the progress of a project. This is especially true when different teams need to pick it up to update or modify the project.

Demetrius Christou: I was able to learn a lot about python in general since all of the code I created was using python. I also learned about how Django works and how all the different http endpoints need to be set up to work properly. After setting up both the influx db and the Neo4j database as well as fixing the issues with the Postgres database I learned about the variations and minor differences in the different mainstream databases used by companies today. As well as strengthen my understanding of how a complex database works.

Evan Dinnon: While working on this project I was able to gain a better understanding on the practical applications of machine learning. I was also able to become more fluent in the python programming language. I also learned a lot about the different types of machine learning algorithms that exist and their use cases. Working on the reinforcement learning algorithm for our project allowed me to better understand neural networks and how they contribute to more efficient algorithms.

William Dulaney: Over the course of this project, I've learned about Python and how machine learning can synthesize a solution from a complex problem. I've reinforced my knowledge of power systems and smart grid systems by using software knowledge to solve the problem of Volt-VAR control. I was exposed to backend programming such as script writing and time-series databases such as InfluxDB. I learned how to specify a convolutional neural network in PyTorch and how to design an algorithm.

Rachel Owens: Over the past two semesters working on this project I learned a lot about working in a team and coordinating efforts. I learned how to work with ReactJS and many of its

features and libraries. I learned about implementing frontend maps. Additionally, I learned how to go through a design process and defend and validate my design decision**s.**

Megan Phinney: Throughout the course, I learned the importance of taking an insane amount of notes. It is difficult to remember different portions of a project and I constantly have to go back through weekly presentations to figure out what I'm forgetting. Now I know I can take my own notes so I can find important information easier.

Derrick Vang: I learned a lot about the React framework. I learned how to work better in a team environment and gained more development skills. I also learned how to coordinate with a client on their desires and expectations.