Project 1: Unsupervised Machine Learning for Chemistry Data using Restricted Boltzmann Machines

Data science motivation

Data-compression is a central problem in science; scientists are always trying to figure out ways to represent things in a more efficient, compact package. It turns out that machine learning is one of the best tools out there for the job. In the data science and machine learning community, the MNIST dataset [1] is the most well-known dataset for machine learning algorithms. The data consists of hand-written digits like in Fig. 1. A machine learning algorithm reads in all of these hand-written digits as binary (0's and 1's) data, assigning a 0 to pixels of the image that are white and a 1 to pixels of the image that are black. After an unsupervised machine learning algorithm has trained itself on this massive data set of hand-written digits, it offers itself as a compact "digit generator"; a small file containing all the information required in order to generate *new* instances of hand-written digits (i.e. hand-written digits that look like a real human drew them, but actually a machine generated). See Fig. 2.

In unsupervised learning, *generative models* can be used to learn the probability distribution underlying a dataset, and can generate new instances of the data. If you are unfamiliar with machine learning using generative models such as the restricted Boltzmann machine used in the rest of this project, you are encouraged to begin with the MNIST folder. It contains a `python` script to load MNIST data. There also exists another good database for machine learning algorithm benchmarks called the "Bars and Stripes" database. In the BarsAndStripes folder, there is a `python` script for loading this data.

[1] http://yann.lecun.com/exdb/mnist/

Figure 1: Examples of hand-written digits in the MNIST database.

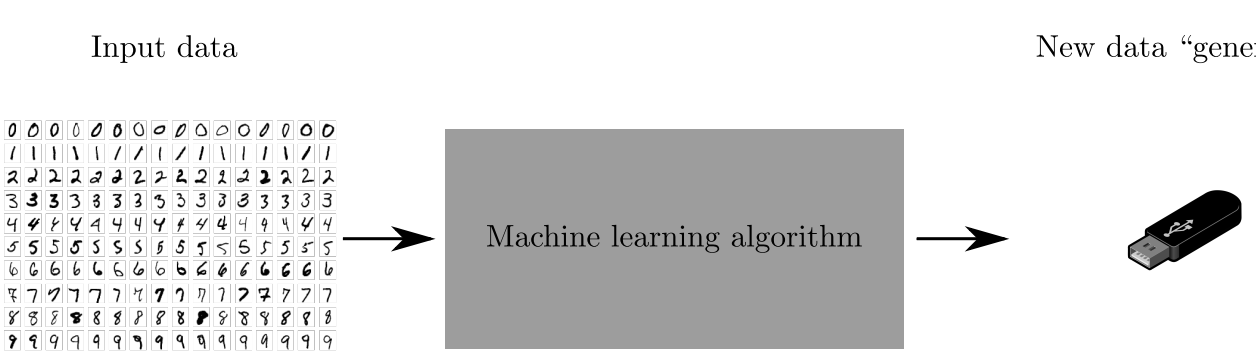

Figure 2: Flow chart of the machine learning process.

Chemistry & physics motivation

Why are machine learning algorithms important in the context of physics and chemistry? Physicists and chemists are faced with an extremely tough firewall when they want to perform numerical simulations known as "the curse of dimensionality"; problems we want to solve scale exponentially in terms of computational resources. Picture this: you want to send your friend information about one apple via email. The information may include the apple's color, its taste, if there are bruises and where the bruises are on the apple, etc. You've gathered all the information needed about the apple into a text file that ends up taking up 500Kb of data. You email this to your friend without any problems.

Now, let's say your friend asks you to send information about 10 different apples. It may take quite a while to document everything, but you have the time to do it. At the end of the documentation process, the file takes up 1GB. You probably can't send it via email any longer, so you send it as a Google-drive link. Now, what about 50 apples? Not only is it going to take an extremely long time to document all of the details about each apple, you can't even store all of this information on a 500GB hard-drive! It turns out that the amount of memory and time to perform this task for N apples is approximately 2^N Megabytes and 2^N minutes [2]. This is precisely the "curse of dimensionality" that physicists and chemists face daily. If we want to simulate things in nature, they typically scale exponentially. Machine learning algorithms can compress these problems we want to solve in such a way that squashes this curse of dimensionality into something much more manageable.

Your tasks

This week, you'll train an RBM to learn information about the H_2 molecule and Rydberg atoms. This is exactly like the MNIST example that was mentioned previously, but instead of hand-written digits we will use data from atoms and molecules. See Fig. 3. You'll also see how machine learning algorithms address the aforementioned "curse of dimensionality".

Task #1

In this task, you will train an RBM to reconstruct how the potential energy stored in the H_2 molecule varies as the distance between the H atoms changes. You will familiarize yourself with working with RBMs in the process.

[2] 2^{50} minutes is about 21421231 *centuries*, and 2^{50} Megabytes is 1125899900000 Gigabytes...

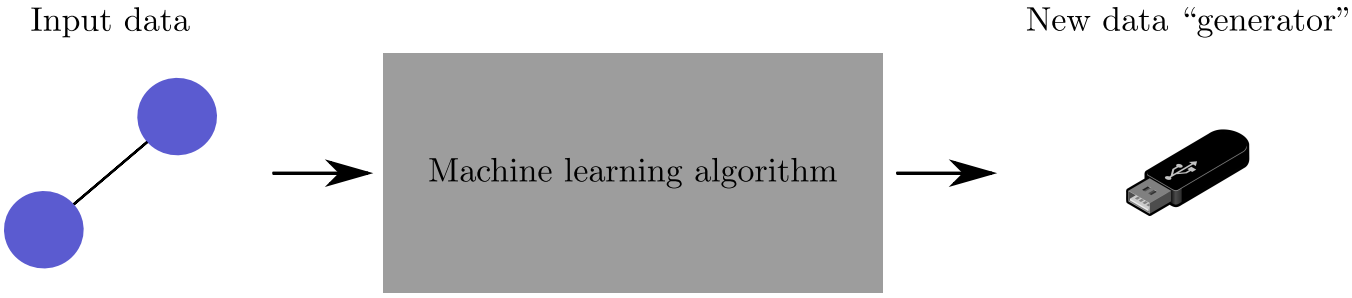

Figure 3: Flow chart of the machine learning process for this week's tasks.

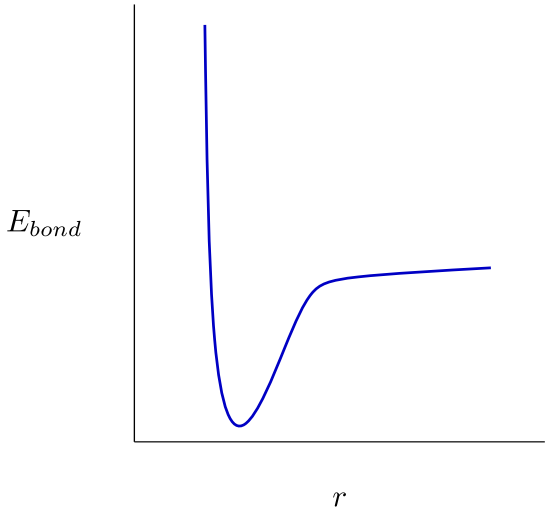

Figure 4: The general form of a molecule's potential energy curve as a function of bond length.

The chemical bond is what forces atoms in a molecule to stay close enough together. If we were to try and take two atoms that are chemically bonded and rip them apart, we would be met with incredible resistance from the retaliation of the bond. On the flip side, there is a repulsive force between atoms that are bonded that ensures that the atoms don't get too close to each other. This balance between the repulsive and attractive forces determines the distance that separates atoms in a chemical bond (called *bond length*). If we were to plot the potential energy stored in a molecule, E_{bond}, as a function of the two atoms' separation, r, it would look something like the blue or red curve in Fig. 4. It's at the very minimum point on the curves in Fig. 4 where the attractive and repulsive forces are in equilibrium with each other.

The code provided to you in `Task1.ipynb` currently only trains an RBM on H_2 data for a specific value of r and calculates the energy from samples generated from the trained RBM. Since we need input data to train the RBM, you're going to need more data for many values of r in order to reproduce a plot that looks like Fig. 4. In the `H2_data/` folder you can find the binary input data you need in files called `R_<r value>_samples.txt`.

E_{bond} can be calculated using the `energy` function in `H2_energy_calculator.py`[3]. The `energy` function takes three arguments:

[3]For more information on how this is calculated, see Ref. [1], Eq. (37). We are also using Sections 2.3.1 and 2.3.2 in Ref. [2] to aid in calculating the energy as a Monte Carlo estimator.

- `samples`: the samples generated from the trained RBM.
- `coeff`: parameters required to calculate E_{bond} at a given r. All of these parameters for every r value can be found in `H2_coefficients.txt` in the `H2_data` folder. The file's first column are the r values, and the remaining 7 numbers in each row are the parameters required to calculate the energy. You can just pass these values into the `energy` function as a `numpy` array.
- `RBM_wvfn`: This is precisely the "compressed" entity that contains all of the information about the problem we want to solve. Don't worry too much about the details, but this is required to calculate E_{bond}.

Here's what we'd like you to do.

1. Load in data for all values of r in the `H2_data/` folder and train different RBMs for each r value.
2. With the trained RBM, generate samples from it in order to calculate E_{bond} from those samples. Then, plot E_{bond} for every r and save the image. Does it look like Fig. 4?

Take a second to appreciate what you just did. You successfully used machine learning (an RBM) to learn information about a H_2 molecule... Cool!

Task #2

Go back to the apples analogy that we talked about earlier for a second. It turns out that Task #1 is sort of like the 1-apple problem: it's easy to do just by brute force and we didn't really need a machine learning algorithm in the first place. In this task, you'll explore the value of machine learning in physics and chemistry.

In Task #1, we gave you plenty of binary input data so that you wouldn't run into any issues. On top of this, learning information about an H_2 molecule is actually quite easy for an RBM; we didn't require that many hidden units. In reality, the availability of data is a problem and the difficulty of learning information about certain problems isn't easy. If we were to obtain input data from a real experiment, the experiment may not be stable for long enough to obtain a lot of data. We'll have limited input data to train something like an RBM as a result. Even if we did have a very stable experiment and could obtain as much data as we wanted, we still might be faced with a hard learning problem. So, we'll need a lot of hidden units and the "compression" of the original problem won't be as much as we'd hoped.

Your input data for this task will be binary data from 100 atoms. Already, we've hit the "curse of dimensionality": the brute force problem at hand scales as 2^N in terms of computational resources, where N is the number of Rydberg atoms. To solve this problem via brute force, we'd only be able to go up to around $N = 12$!

The input data file is `Rydberg_data.txt`. To train an RBM on this data, we will look at evaluating the energy *during* training to see how well the RBM is doing and stop the training process once we see that it's done well enough (dubbed the "learning criterion"). The energy can be calculated using the `energy` function in `Rydberg_energy_calculator.py`. The learning criterion you'll use is $|E_{RBM} - E_{exact}| \leq 0.0001$ (i.e. train the RBM until this is satisfied), where $E_{exact} = -4.1203519096$. However, don't wait forever for the RBM to try and satisfy the learning criterion. Only wait 1000 training steps (epochs).

Here's what we'd like you to do.

1. We've given you a very large dataset in `Rydberg_data.txt`. So let's evaluate how "hard" the learning problem is (i.e. how many hidden units to reach our learning criterion). Use the entire dataset to determine the minimum number of hidden units required in order to obtain the learning criterion. The "size" of the compressed entity that the RBM spits out is the equivalent storage of $100 + n_h + n_h \times 100$ numbers, where n_h is the number of hidden units that you found (start with $n_h = 1$). How does this compare to 2^{100}?

2. Double the number of hidden units determined in the previous question and determine how many data points (i.e. the portion of the full dataset in `Rydberg_data.txt`) you need to reach the learning criterion. Start with 500 data points. Move up in increments of at least 100 (depending on how precise you want to be!). This will let experimentalists know the minimum amount of data required from their experiment!

Start with the code provided in `Task2.ipynb`. For those of you interested, we've intentionally left the RBM code in `RBM_helper.py` very generic and bare-bones. Use any means necessary to modify this code (or make your own!) and try and improve the results you previously got.

Further Challenges

Now you're on your own. Discuss ideas for expanding on the above tasks with your group. Some challenges to consider:

1. Additional data has been provided for LiH and BeH_2. [4] Consider an RBM reconstruction for these more complicated molecules. What is the fundamental difference between these and the systems studied in the above tasks?

2. Explore other unsupervised machine learning techniques besides RBMs, such as recurrent neural networks (RNNs).

3. What other quantum wavefunction datasets can you obtain? Is a "standard dataset" for machine learning quantum chemistry possible?

As the final outcome, your project should consider what you learned above in the context of quantum entrepreneurship. This task has been laid out in the Quantum Cohort Project Business Application. [5]

References

[1] Rongxin Xia, Teng Bian, and Sabre Kais. Electronic Structure Calculations and the Ising Hamiltonian. *arXiv:1706.00271 [quant-ph]*, June 2017.

[2] M. Beach, I. De Vlugt, A. Golubeva, P. Huembeli, B. Kulchytskyy, X. Luo, R. Melko, E. Merali, and G. Torlai. QuCumber: Wavefunction reconstruction with neural networks. *SciPost Physics*, 7(1):009, July 2019.

[4] CohortProject_2020/datasets/qubit_molecules/

[5] CohortProject_2020/Project_1_RBM_and_Tomography/Business_Application.md