CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM

Độc Lập – Tự Do – Hạnh Phúc

TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

BÁO CÁO MÔN HỌC PROJECT 3

Đề tài: Dự đoán phụ tải

sử dụng mạng nơ-ron nhân tạo MLP

Họ và tên: Nguyễn Huy Hoàng

MSSV: 20173132

Lớp: KHMT-05 K62

GVHD: Nguyễn Nhất Hải

Đặng Hoàng Anh

Hà Nội, tháng 9 – 2021

Lời cam kết

Tôi xin cam đoan đây là phần nghiên cứu và thể hiện đồ án môn học của riêng tôi, không sao chép các đồ án khác, nếu sai tôi xin chịu hoàn toàn trách nhiệm và chịu mọi kỷ luật của khoa/viện và nhà trường đề ra.

Lời cảm ơn

Em xin chân thành cảm ơn thầy giáo hướng dẫn Nguyễn Nhất Hải và Đặng Hoàng Anh vì sự giúp đỡ và dìu dắt tận tình của các thầy trong suốt quá trình em thực hiện đồ án.

Em cũng xin gửi lời cảm ơn đến các thầy, cô giáo trong viện CNTT&TT, trường đại học Bách Khoa Hà Nội, những nhà giáo đã truyền dạy cho em những kiến thức quý báu trong suốt những năm học vừa qua.

Cuối cùng, xin cảm ơn gia đình và bè bạn đã tạo điều kiện và luôn động viên tôi hoàn thành đồ án này.

Do thời gian hoàn thành đồ án có hạn cho nên những suy nghĩ cũng như sự thể hiện ý đồ không tránh khỏi có những khiếm khuyết. Em rất mong được sự động viên và đóng góp ý kiến của các thầy cô giáo.

Sinh viên thể hiện

Nguyễn Huy Hoàng

Mục lục

1. Giới thiệu đề tài

1.1. Đặt vấn đề

Vấn đề về năng lượng vẫn luôn là vấn đề được cả thế giới quan tâm. Trong đó điện năng là phần quan trọng nhất. Năm 2017, trong nhóm các nước thuộc OECD, công nghiệp là ngành tiêu thụ điện năng nhiều nhất, chiếm 32.2% điện năng tiêu thụ. Tuy nhiên lượng điện tiêu thụ của dân cư và ngành thương mại dịch vụ thấp hơn không đáng kể, tương ứng chiếm 30.8% và 31.7% [1]. Điện năng tiêu thụ bởi dân cư chiếm tỉ trọng không hề nhỏ, nếu như có thể tiết kiệm được lượng điện năng này, thì sẽ có lợi ích to lớn về kinh tế, môi trường.

Trong đề tài này, chúng tôi nghiên cứu phát triển một thiết bị giám sát việc sử dụng điện năng tại mỗi hộ gia đình, từ đó có thể giúp cho các hộ dễ dàng giám sát sự tiêu thụ điện năng của các thiết bị trong nhà, dẫn tới định hướng để tiết kiệm điện. Trước đây, người ta sử dụng phương pháp lắp đặt từng thiết bị đo, bộ cảm biến vào từng thiết bị điện trong nhà để thực hiện việc giám sát. Các cảm biến sẽ gửi thông tin về một thiết bị trung tâm để người dùng có thể theo dõi. Phương pháp này có ưu điểm là độ tin cậy, chính xác cao, tuy nhiên nhược điểm là chi phí lắp đặt cao, quá trình vận hành phức tạp, vì ta phải lắp từng cảm biến vào từng thiết bị. Để khắc phục điều này, người ta nghiên cứu một phương pháp khác, tên tiếng Anh là "Nonintrusive load monitoring" (NILM). Dịch ra tiếng Việt nghĩa là "Giám sát tải không xâm nhập". Phương pháp thực hiện bằng cách thay vì gắn cảm biến vào từng thiết bị, thì người ta chỉ gắn một bộ đo vào bảng điện chính của căn nhà. Bộ đo có nhiệm vụ đo các thông số về hiệu điện thế, cường độ dòng điện, hệ số công suất, công suất tiêu thụ,… Số liệu này được gửi đến bộ xử lí, sử dụng các phương pháp học máy (Machine Learning) để phân tích số liệu và xác định mức độ tiêu thụ của từng thiết bị. Phương pháp này hiện tại còn hạn chế về độ chính xác, vì thế mà vẫn đang được nghiên cứu. Tuy nhiên nếu thành công và ứng dụng rộng rãi được, thì đây sẽ là một cách tiếp cận tối ưu cho việc giám sát phụ tải.

1.2. Mục tiêu và phạm vi của đề tài

Trong phạm vi đồ án môn học, đề tài này chỉ đi vào nghiên cứu thuật toán học máy để thực hiện dự đoán, xác định mức độ tiêu thụ điện năng của từng thiết bị trong nhà. Dữ liệu đo đạc của các thiết bị điện đã có từ trước và được thầy Đặng Hoàng Anh cung cấp. Việc lựa chọn một thuật toán tối ưu và nhúng nó vào bộ xử lí để tạo ra một hệ thống hoàn chỉnh sẽ là công việc của các nghiên cứu sau này.

Cụ thể, thuật toán học máy được nghiên cứu là mô hình mạng nơ-ron nhân tạo Multilayer Perceptron (MLP). Chúng tôi đầu bằng việc tìm hiểu cấu tạo của mạng, cách nó hoạt động ra sao, sử dụng thư viện Sklearn để xây dựng ra mạng MLP, chạy thử với bộ dữ liệu, tìm hiểu sâu về bên trong thuật toán và tự triển khai mạng MLP mà không dùng thư viện.

Mục tiêu của đề tài phần lớn là để sinh viên làm quen với học máy, hiểu được cách xây dựng mạng nơ-ron và nắm được cách tự triển khai mô hình mạng nơ-ron vào bài toán thực tế.

1.3. Giải pháp thực hiện

Tóm tắt các bước đã làm trong quá trình thực hiện đề tài:

1. Đọc 2 bài báo [2] và [3]
2. Phân tích dữ liệu hiện có
3. Cài đặt theo bài báo 1, sử dụng thư viện sklearn
4. Cài đặt theo bài báo 2, sử dụng thư viện sklearn
5. Tìm hiểu và cài đặt mô hình MLP không sử dụng thư viện
6. Thay đổi mô hình theo hướng dự đoán phần trăm sử dụng thay vì trạng thái on/off

Cụ thể từng bước sẽ được trình bày trong các phần sau của báo cáo.

Mã nguồn, tài liệu, notebook đã được gửi kèm cùng với báo cáo.

1.4. Bố cục báo cáo

Báo cáo được chia thành 6 phần chính như sau:

1. Giới thiệu đề tài
2. Công nghệ sử dụng
3. Mô tả bài toán
4. Quá trình triển khai
5. Các giải pháp và đóng góp nổi bật
6. Kết luận và hướng phát triển

2. Công nghệ sử dụng

2.1. Ngôn ngữ lập trình

Python là ngôn ngữ lập trình được chọn để hoàn thành đề tài này. Hầu hết các dự án liên quan đến trí tuệ nhân tạo và học máy (AI, ML) đều lấy Python làm ngôn ngữ lập trình chính, vì những lý do sau:

- Python có cấu trúc cú pháp ngắn gọn, dễ đọc. Các thuật toán AI, ML đa phần đều phức tạp, vì thế một ngôn ngữ lập trình có cú pháp đơn giản sẽ làm cho code của cả hệ thống dễ đọc dễ hiểu hơn. Từ đó khiến cho việc phát triển hệ thống cũng đơn giản hơn. Lập trình viên sẽ tập trung được thời gian, trí lực vào việc giải quyết bài toán của Học máy thay vì gặp rắc rối với kỹ thuật của ngôn ngữ.
- Python có nhiều thư viện và framework. Python có một kho công nghệ phong phú bao gồm rất nhiều thư viện cho trí tuệ nhân tạo và học máy. Keras, Tensorflow, Scikit-learn, Numpy, Scipy, Pandas, Seaborn, … là những thư viện hàng đầu. Với những thư viện có sẵn, các lập trình viên có thể phát triển sản phẩm của mình nhanh hơn. Quan trọng là kết quả mang lại lợi ích gì cho việc phát triển kinh doanh, còn khách hàng không quan tâm nó được viết ra như thế nào.

Về hiệu năng xử lí cũng như khả năng nhúng vào các thiết bị phần cứng, Python không làm tốt bằng C/C++. Tuy nhiên vì trong phạm vi đề tài, để cho ngôn ngữ lập trình không trở thành rào cản, chúng tôi vẫn chọn Python. Trong tương lai, khi phát triển được thuật toán đến một mức chấp nhận được, giải quyết được bài toán thực tế, khi đó ta mới nên chuyển sang ngôn ngữ C/C++ để có thể nhúng được thuật toán vào trong thiết bị.

2.2. Môi trường

Nếu phần mềm có nhiều mã nguồn phức tạp (nhiều file, thư mục, nhiều class, các class kế thừa nhau,…) thì một môi trường phát triển tích hợp (IDE) là cần thiết. Tuy nhiên đó là tính chất của các dự án công nghệ phần mềm, ví dụ như làm web, làm game, … Còn đề tài này của chúng ta chưa cần nhiều mã nguồn nên chúng tôi chọn một môi trường khác là Jupiter Notebook.

Jupiter là một công cụ mã nguồn mở miễn phí với mục đích nhắm đến khoa học dữ liệu và giáo dục. Jupiter Notebook là công cụ cho phép bạn đưa cả code Python và các thành phần văn bản

phức tạp như hình ảnh, công thức, video, biểu thức,… vào trong cùng một file giúp cho việc trình bày trở nên dễ hiểu, giống như một file trình chiếu nhưng lại có thể thực hiện chạy code trên đó. Các file notebook này có thể được chia sẻ với người khác để mọi người có thể nắm được ý tưởng của nhau một cách dễ dàng.

2.3. Thư viện

Các thư viện được sử dụng ở đây đều là những thư viện hàng đầu được sử dụng về AI, ML. Hầu hết những người học Python đều phải biết.

- **Pandas**. Pandas là một thư viện Python cung cấp các cấu trúc dữ liệu nhanh, mạnh mẽ, linh hoạt. Trong đề tài này thì Pandas được sử dụng để đọc dữ liệu file .csv là chính.
- **Numpy**. Numpy là một thư viện lõi phục vụ cho khoa học máy tính của Python, hỗ trợ cho việc tính toán các mảng nhiều chiều, có kích thước lớn với các hàm đã được tối ưu áp dụng lên các mảng nhiều chiều đó. Numpy đặc biệt hữu ích khi thực hiện các hàm liên quan đến đại số tuyến tính. Nếu sử dụng cấu trúc dữ liệu list có sẵn của Python rồi dùng vòng for để duyệt và tính toán trên từng phần tử, thì không thể đáp ứng được yêu cầu về hiệu năng. Việc sử dụng vòng lặp trong Python tất nhiên với số lượng ít vòng lặp thì không sao, nhưng nếu số vòng lặp lên đến hàng triệu, thì ta có thể thấy list trong Python có hiệu năng cực tệ. Các thuật toán AI, ML thực hiện tính toán rất nhiều trên mảng, ma trận, vì thế yêu cầu về hiệu năng rất cao (hay nói cách khác là thời gian tính toán phải nhanh). Numpy được viết trên C/C++ nên có khả năng tính toán rất tốt.
- **Matplotlib**. Để thực hiện các suy luận thống kê cần thiết, cần phải trực quan hoá dữ liệu của bạn và Matplotlib là một trong những giải pháp như vậy cho người dùng Python. Nó là một thư viện vẽ đồ thị rất mạnh mẽ hữu ích cho những nguòi làm việc với Python và Numpy. Module được sử dụng nhiều nhất của Matplotlib là Pyplot cung cấp giao diện như MATLAB nhưng thay vào đó, nó sử dụng Python và nó là mã nguồn mở.
- **Scikit-learn**. (Sklearn) là thư viện mạnh mẽ nhất dành cho các thuật toán học máy được viết trên ngôn ngữ Python. Thư viện cung cấp một tập các công cụ xử lý các bài toán machine learning và statistical modeling gồm: classification, regression, clustering, và dimensionality reduction. Thư viện được cấp phép bản quyền chuẩn FreeBSD và chạy được trên nhiều nền tảng Linux. Scikit-learn được sử dụng như một tài liệu để học tập. Trong đề tài này chúng tôi chỉ sử dụng một thành phần trong Sklearn đó là mạng MLP.

2.4. Cài đặt môi trường, ngôn ngữ, thư viện

Để cài đặt môi trường, ngôn ngữ, thư viện chúng ta có thể cài đặt riêng rẽ từng thành phần, theo hướng dẫn trên trang chủ của chúng. Tuy nhiên một cách nhanh nhất mà chúng tôi sử dụng đó là sử dụng nền tảng Anaconda.

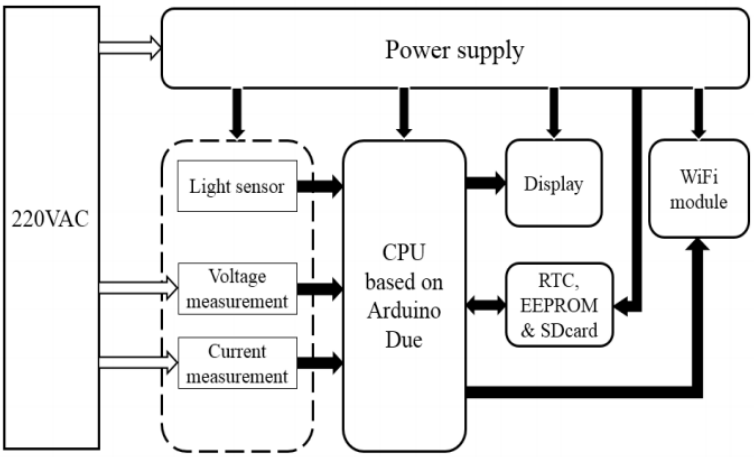
Anaconda là nền tảng (platform) mã nguồn mở về Khoa học dữ liệu (Data Science) trên Python thông dụng nhất hiện nay. Với hơn 6 triệu người dùng, Anaconda là cách nhanh nhất và dễ nhất để học Khoa học dữ liệu với Python hoặc R trên Windows, Linux và Mac OS X.

Chỉ cần lên trang chủ của anaconda và tải về (bản miễn phí) rồi cài đặt là ta đã có đầy đủ Jupyter Notebook, Python, và các thư viện cần thiết.

3. Mô tả bài toán

3.1. Kiến trúc hệ thống

Việc thiết kế và phát triển phần cứng không thuộc phạm vi của đề tài. Tuy nhiên chúng tôi vẫn đề cập ở đây để bạn đọc có một cái nhìn đầy đủ hơn về bài toán.



Kiến trúc phần cứng LMD

Hình trên là một sơ đồ tổng quan về thiết kế phần cứng của bộ giám sát tải (Load Monitoring Device – LMD). Bởi vì LMD được cài đặt vào trong bảng điện của căn nhà, nên nguồn điện sẽ được chuyển đổi từ dòng điện 220VAC và cung cấp cho thiết bị hoạt động. LMD bao gồm ba kênh đo lường đó là cảm biến ánh sáng, bộ đo hiệu điện thế và bộ đo cường độ dòng điện. Các thông số đo được sẽ được gửi tới CPU được thiết kế dựa trên Arduino Due, được lưu trữ trên bộ nhớ EEPROM và hiển thị lên màn hình LCD. Ngoài ra dữ liệu sẽ được gửi tới máy tính thông qua Wifi module.

3.2. Mô tả bộ dữ liệu

Trong đề tài này, chúng tôi đã có sẵn dữ liệu đo thực tế từ một căn hộ. Trong một dự án khác, thầy Đặng Hoàng Anh và cộng sự đã gắn các thiết bị đo vào các thiết bị điện trong một căn hộ. Dữ liệu được gửi về trung tâm thường xuyên và hệ thống này vẫn hoạt động cho đến thời điểm này. Công việc của chúng tôi là nghiên cứu thuật toán dựa trên bộ dữ liệu mẫu này.

Bộ dữ liệu mẫu mà chúng tôi sử dụng bao gồm 4 thông tin: cường độ dòng điện tiêu thụ, hiệu điện thế tiêu thụ, công suất tiêu thụ và hệ số công suất của 7 thiết bị:

- Chiếu sáng (light)
- Ổ cắm khách + bếp + ngủ (socket)
- Nóng lạnh (heater)
- Điều hoà 1 & 2 (aircond1)
- Điều hoà 3 (aircond2)
- Điều hoà phòng khách (aircond3)
- Bếp từ (indcooker)

Tất cả bao gồm 31680 bản ghi. Mỗi bản ghi là số liệu đo trong một phút, bắt đầu từ ngày 15/7/2020.

Time	Chiếu sáng	Ổ cắm khách + bếp + ngủ	Nóng lạnh	Điều hoà 1 & 2	Điều hoà 3	Điều hoà phòng khách	Bếp từ
7/15/2020 0:00	0	216	0	244	0	631	1
7/15/2020 0:01	0	210	0	286	0	631	3.5
7/15/2020 0:02	0	206	0	314	0	633	6
7/15/2020 0:03	0	206	0	254	0	632	3.5
7/15/2020 0:04	0	160	0	274	0	629	1
7/15/2020 0:05	0	115	0	342	0	628	1
7/15/2020 0:06	0	115	0	263	0	629	1
7/15/2020 0:07	0	115	0	217	0	631	4

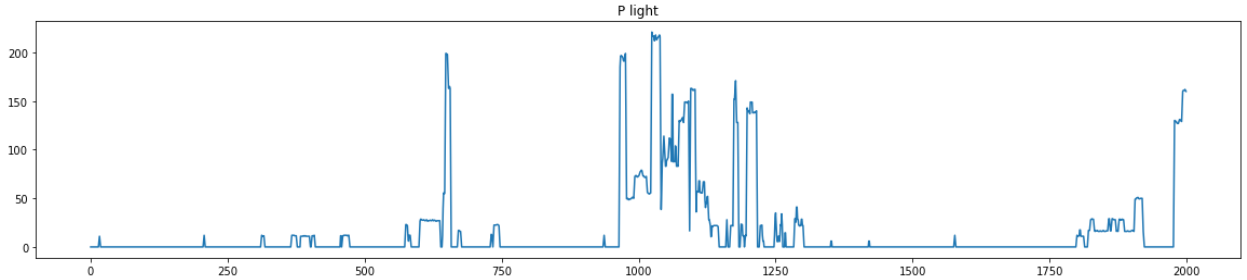
Một vài điểm dữ liệu của tập dữ liệu đo công suất tiêu thụ P. (U, I, cosphi tương tự)

Một thông số quan trọng của các thiết bị điện khi sử dụng đó là công suất phản kháng Q. Tuy nhiên Q chưa có trong bộ dữ liệu, mà ta sẽ phải tính gián tiếp thông qua P và cosphi, dựa theo công thức:

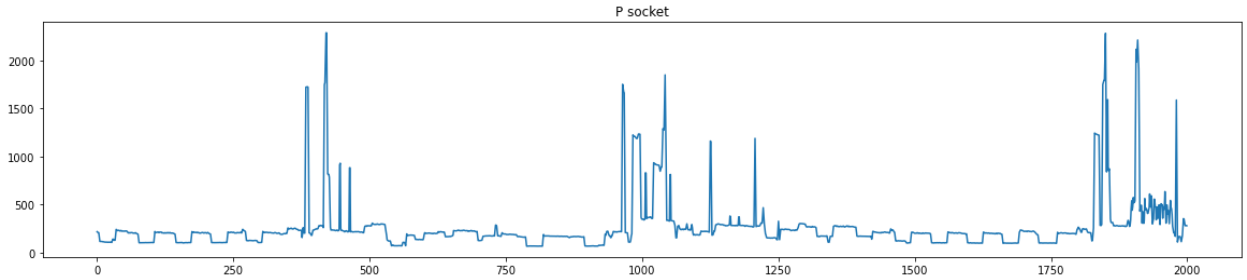
$$Q = \tan(arccos(cosphi)) * P$$

3.2.1. Trực quan hoá dữ liệu

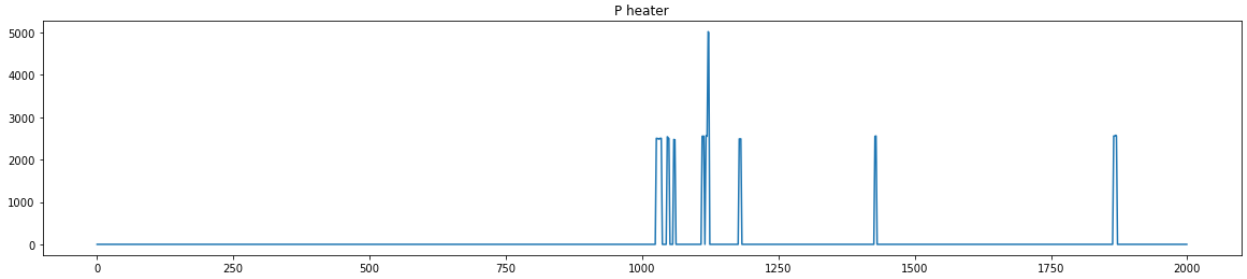
Dưới đây là một số hình ảnh đồ thị bảng đo công suất của từng thiết bị, để cho bạn đọc có một cái nhìn trực quan. Các đồ thị chỉ vẽ 2000 điểm dữ liệu đầu tiên, và chỉ thể hiện cho công suất tiêu thụ. Các thông số khác (U, I, cosphi) nếu vẽ ra cũng sẽ có cái nhìn tương tự.



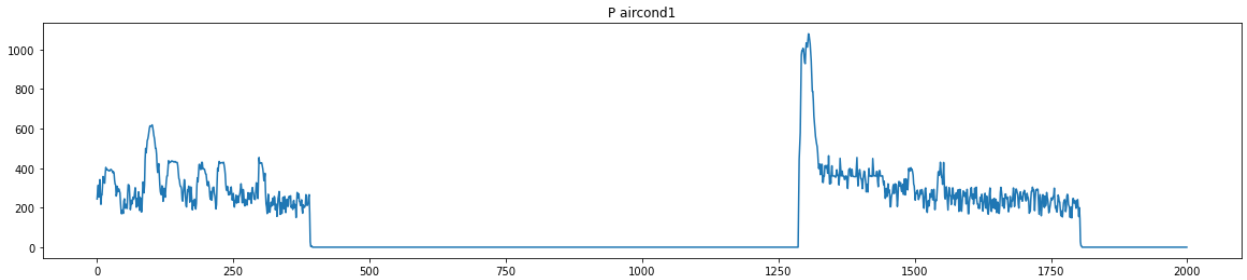
Công suất tiêu thụ của thiết bị chiếu sáng (2000 phút đầu tiên)



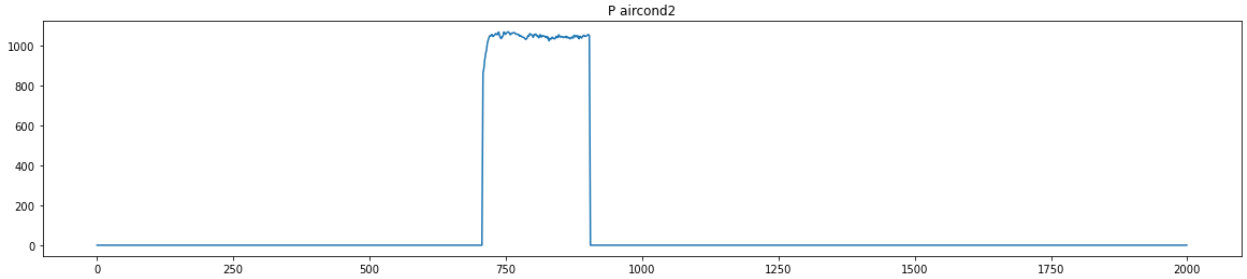
Công suất tiêu thụ của ổ cắm khách + bếp + ngủ (2000 phút đầu tiên)



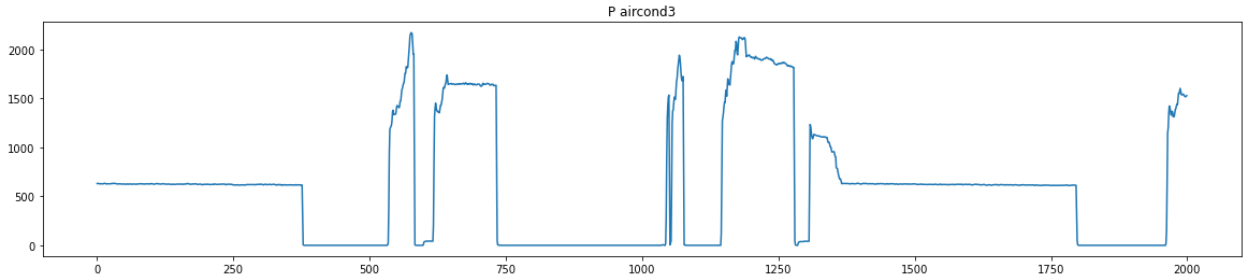
Công suất tiêu thụ của bình nóng lạnh (2000 phút đầu tiên)



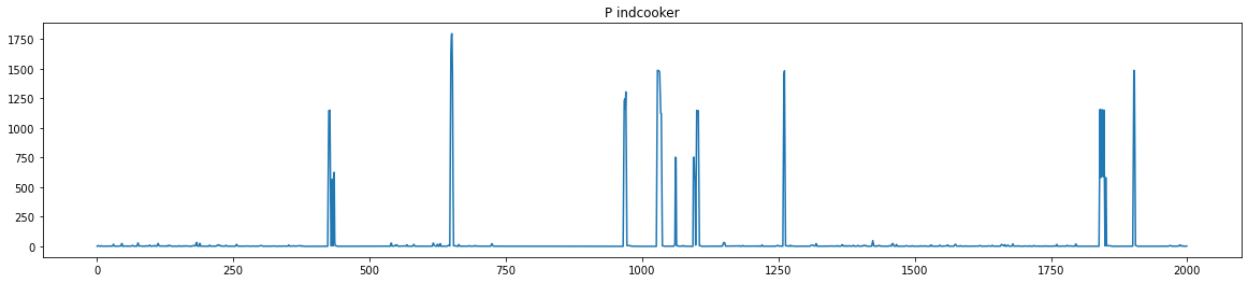
Công suất tiêu thụ của điều hoà 1 & 2 (2000 phút đầu tiên)



Công suất tiêu thụ của điều hoà 3 (2000 phút đầu tiên)



Công suất tiêu thụ của điều hoà phòng khách (2000 phút đầu tiên)



Công suất tiêu thụ của bếp từ (2000 phút đầu tiên)

3.3. Mô tả bài toán

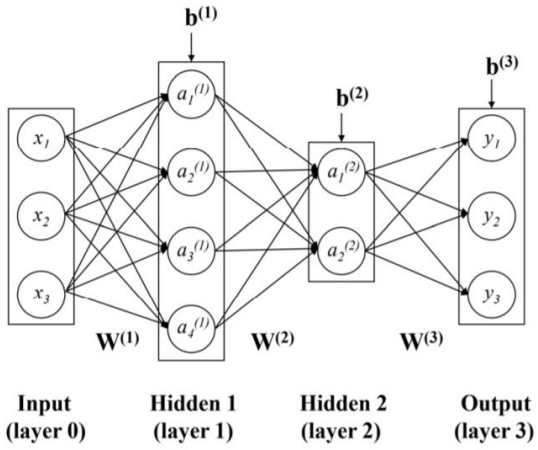
Khi hoạt động thực tế, thiết bị đo không đo riêng U, I, P, cosphi của từng thiết bị, mà chỉ đo ở bảng điện tổng của căn nhà. Nhiệm vụ của chúng ta là từ số liệu tổng đó, làm sao phân tích ra được trạng thái hoạt động của từng thiết bị trong căn nhà. Để mô phỏng dữ liệu đo ở bảng điện tổng, ta làm như sau:

- Đối với điện áp U: điện áp ở bảng điện tổng hay trên từng thiết bị là như nhau. Ta chỉ việc giữ nguyên.

- Đối với dòng điện I: cộng cường độ dòng điện tiêu thụ của từng thiết bị lại với nhau theo từng thời điểm t, ta được I tổng.
- Đối với công suất P: cộng công suất tiêu thụ của từng thiết bị lại với nhau theo từng thời điểm t, ta được P tổng.
- Đối với công suất phản kháng Q: cộng công suất phản kháng của từng thiết bị lại với nhau theo từng thời điểm t, ta được Q tổng.
- Đối với hệ số công suất cosphi: chưa sử dụng đến cosphi tổng.

Bài toán được chia làm hai dạng. Một là xây dựng mô hình MLP để xác định (dự đoán) trạng thái on/off của các thiết bị. Hai là xây dựng mô hình MLP để xác định (dự đoán) công suất tiêu thụ của các thiết bị.

Tại sao lại có hai dạng như vậy? Ban đầu chúng tôi tìm hiểu và làm theo hai bài báo [2] và [3]. Hai bài báo này đều xác định trạng thái on/off của các thiết bị. Sau khi đã nắm rõ được cách hoạt động của mô hình MLP, chúng tôi tự xây dựng mạng MLP không dùng thư viện, và tuỳ biến nó để có thể dự đoán được công suất tiêu thụ của từng thiết bị.



Minh hoạ mô hình mạng MLP

3.3.1. Xây dựng mô hình MLP để xác định trạng thái on/off của các thiết bị

Trong thực tế, một thiết bị điện có nhiều hơn hai trạng thái on và off. Ví dụ như bếp từ sẽ có thêm trạng thái standby. Tuy nhiên để cho đơn giản, ta coi như các thiết bị trong tập dữ liệu chỉ gồm hai trạng thái on/off. Đặt ra một ngưỡng công suất $threshold = 10W$, thiết bị nào đang hoạt động với công suất P > 10W sẽ được coi là đang ở trạng thái on, ngược lại là đang ở trạng thái off.

Từ tập dữ liệu, ta có thể xác định được các thiết bị (chiếu sáng, ổ cắm, nóng lạnh, điều hoà,…) đang ở trạng thái on hay off, đơn giản bởi vì ta có số liệu đo công suất của từng thiết bị theo thời gian. Từ đó có thể gán nhãn cho tập dữ liệu và triển khai được mô hình học có giám sát (supervised learning).

Mô hình MLP sẽ nhận đầu vào là các thông số U, I tổng, P tổng, Q tổng tại thời điểm t nào đó. Đầu ra là thiết bị nào đang on, thiết bị nào đang off. Như vậy trường hợp này thuộc lớp bài toán phân lớp (Classification).

3.3.2. Xây dựng mô hình MLP để xác định công suất tiêu thụ của các thiết bị

Mô hình MLP sẽ nhận đầu vào là các thông số U, I tổng, P tổng, Q tổng tại thời điểm t nào đó. Đầu ra sẽ là tỉ lệ công suất của từng thiết bị đối với P tổng. Trường hợp này thuộc lớp bài toán hồi quy (Regression).

Ví dụ tại thời điểm phút thứ 1000:

- P tổng = 3000W

Mô hình dự đoán ra:

- P light = 0.1
- P socket = 0.3
- P heater = 0.0
- P aircond1 = 0.5
- …

Nghĩa là thiết bị chiếu sáng đang chiếm 10% của P tổng, hay công suất hoạt động của thiết bị chiếu sáng đang là 300W. Tương tự, công suất hoạt động của thiết bị ổ cắm đang là 900W, nóng lạnh là 0W, aircond1 (viết tắt của điều hoà 1 & 2) là 1500W,… Tất nhiên mô hình cần phải đảm bảo tổng tỉ lệ bằng 1.

Việc gán nhãn cho tập dữ liệu trong trường hợp này cũng hết sức đơn giản. Nhãn sẽ là tập các tỉ lệ công suất của từng thiết bị đối với P tổng, tính bằng cách lấy P của từng thiết bị chia cho P tổng.

4. Quá trình triển khai

Quá trình triển khai thực hiện đề tài bao gồm các bước:

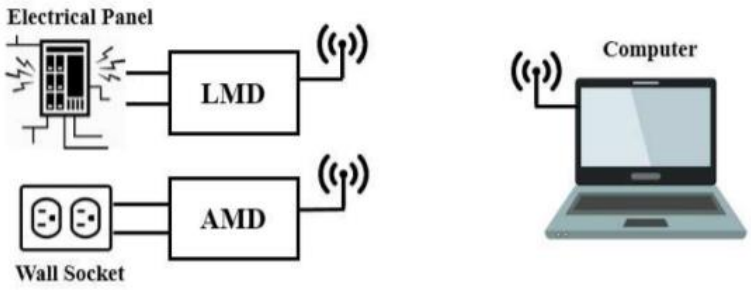
- Xây dựng theo phương pháp của bài báo thứ nhất [2]
- Xây dựng theo phương pháp của bài báo thứ hai [3]
- Kết hợp hai bài báo
- Đi sâu vào xây dựng mạng MLP
- Dự đoán công suất tiêu thụ của các thiết bị

Mục này sẽ đi vào chi tiết từng bước của quá trình triển khai thực hiện đề tài.

4.1. Xây dựng theo phương pháp của bài báo thứ nhất

4.1.1. Tóm tắt nội dung của bài báo thứ nhất

Tác giả đã đề xuất hai thiết bị là Load Monitoring Device (LMD) và Activity Monitoring Device (AMD). LMD chính là thiết bị liên quan đến đề tài này. Còn về phần thiết bị AMD chúng tôi bỏ qua.



Kiến trúc hệ thống

LMD là thiết bị được gắn vào bảng điện chính. Tác dụng của nó là đo và tính toán U, I, P, Q, cosphi, E_v. Số liệu được đưa đến module xử lí theo phương pháp học máy (mạng MLP) và cho ra kết quả là những thiết bị nào đang on/off. Đồng thời dữ liệu cũng được gửi về máy tính thông qua Wifi.

Mô hình MLP của tác giả đề xuất có kiến trúc như sau:

- Tầng input có 5 nơ-ron. Vector input bao gồm 5 thông số: U, I, P, cosphi, E_v

- Tầng hidden có 20 nơ-ron. Tác giả có thử nghiệm với số nơ-ron từ 0-100 cho tầng ẩn và kết quả 20 nơ-ron là tối ưu.
- Tầng output có 2^n nơ-ron. (n là số thiết bị).

Tác giả thực hiện thí nghiệm với 3 thiết bị: bóng đèn sợi đốt, ấm đun nước và máy sấy tóc. Mỗi thiết bị có hai trạng thái on/off, vì thế có 2^3=8 trường hợp của tầng đầu ra như sau:

Out-comes	1	2	3	4	5	6	7	8
y (Output vector)	$\begin{bmatrix}1\\0\\0\\0\\0\\0\\0\\0\end{bmatrix}$	$\begin{bmatrix}0\\1\\0\\0\\0\\0\\0\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\\1\\0\\0\\0\\0\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\\0\\1\\0\\0\\0\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\\0\\0\\1\\0\\0\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\\0\\0\\0\\1\\0\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\\0\\0\\0\\0\\1\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\1\end{bmatrix}$

Các giá trị đầu ra khả thi

Kết quả thực nghiệm:

- Tập dữ liệu: 2275 phần tử. 70% train, 30% test.
- Độ chính xác: 99.78%

4.1.2. Triển khai theo phương pháp của bài báo, sử dụng thư viện sklearn

4.1.2.1. Chuẩn bị dữ liệu

Sử dụng thư viện pandas đọc dữ liệu từ các file .csv vào thành các DataFrame. DataFrame là cấu trúc dữ liệu linh hoạt của Pandas, hình dung như dạng bảng, có thể convert sang dạng mảng Numpy một cách dễ dàng. Đặt tên cho các cột lần lượt là 'time', 'light', 'socket', 'heater', 'aircond1', 'aircond2', 'aircond3', 'indcooker'.

```
1.  import pandas as pd
2.
3.  # Power dataframe
4.  p_df = pd.read_csv('./data/W.csv',\
5.                     names=['time', 'light', 'socket', 'heater', 'aircond1', 'aircond2',
    'aircond3', 'indcooker'],\
6.                     header=0)
7.
8.  # Voltage dataframe
9.  u_df = pd.read_csv('./data/V.csv',\
```

```
10.                     names=['time', 'u'],\
11.                     header=0)
12.
13. # Current dataframe
14. i_df = pd.read_csv('./data/A.csv',\
15.                     names=['time', 'light', 'socket', 'heater', 'aircond1', 'aircond2',
    'aircond3', 'indcooker'],\
16.                     header=0)
17.
```

Ở đây ta chỉ sử dụng 3 thông số đầu vào là P, U, I. Tạm thời cosphi chưa sử dụng tới.

Khi sử dụng Jupyter Notebook ta có thể xem DataFrame dưới dạng bảng rất đẹp mắt và tiện lợi.

Bảng đo công suất tiêu thụ

```
p_df
```

	time	light	socket	heater	aircond1	aircond2	aircond3	indcooker
0	2020-07-15 00:00:00	0.0	216.0	0	244.0	0.0	631.0	1.0
1	2020-07-15 00:01:00	0.0	210.0	0	286.0	0.0	631.0	3.5
2	2020-07-15 00:02:00	0.0	206.0	0	314.0	0.0	633.0	6.0
3	2020-07-15 00:03:00	0.0	206.0	0	254.0	0.0	632.0	3.5
4	2020-07-15 00:04:00	0.0	160.0	0	274.0	0.0	629.0	1.0
...	...	...	...	...	...	...	...	...
31675	2020-08-05 23:55:00	0.0	236.0	0	183.0	0.0	0.0	1.0
31676	2020-08-05 23:56:00	0.0	238.0	0	174.0	0.0	0.0	1.0
31677	2020-08-05 23:57:00	0.0	243.0	0	199.0	0.0	0.0	1.0
31678	2020-08-05 23:58:00	0.0	241.0	0	229.0	0.0	0.0	1.0
31679	2020-08-05 23:59:00	0.0	246.0	0	245.0	0.0	0.0	1.0

31680 rows × 8 columns

Bảng đo điện áp tiêu thụ

```
u_df
```

	time	u
0	2020-07-15 00:00:00	231
1	2020-07-15 00:01:00	230
2	2020-07-15 00:02:00	230
3	2020-07-15 00:03:00	230
4	2020-07-15 00:04:00	230
...	...	...
31675	2020-08-05 23:55:00	230
31676	2020-08-05 23:56:00	230
31677	2020-08-05 23:57:00	231
31678	2020-08-05 23:58:00	231
31679	2020-08-05 23:59:00	232

31680 rows × 2 columns

Ta nên bắt đầu thử nghiệm với 2, 3 thiết bị trước, rồi tất cả sau. Để tiện cho việc này, ta tạo một mảng lưu danh sách các thiết bị được chọn để dự đoán.

```
1.  select_device = ['heater', 'indcooker']
2.  # select_device = ['heater', 'indcooker', 'aircond1']
3.  # select_device = ['heater', 'indcooker', 'aircond1', 'aircond2', 'aircond3']
4.  # select_device = ['heater', 'indcooker', 'aircond1', 'aircond2', 'aircond3', 'socket',
    'light']
5.
```

Tiếp theo ta tính tổng P và I (không cần tính U, vì U của các thiết bị là như nhau tại cùng thời điểm) để mô phỏng đó là giá trị mà thiết bị đo đo được từ bảng điện tổng. Tất nhiên chỉ tính trong các thiết bị đã được chọn.

Tính tổng P

```
p_df['sum'] = p_df[select_device].sum(axis=1)
p_df.head()
```

	time	light	socket	heater	aircond1	aircond2	aircond3	indcooker	sum
0	2020-07-15 00:00:00	0.0	216.0	0	244.0	0.0	631.0	1.0	1092.0
1	2020-07-15 00:01:00	0.0	210.0	0	286.0	0.0	631.0	3.5	1130.5
2	2020-07-15 00:02:00	0.0	206.0	0	314.0	0.0	633.0	6.0	1159.0
3	2020-07-15 00:03:00	0.0	206.0	0	254.0	0.0	632.0	3.5	1095.5
4	2020-07-15 00:04:00	0.0	160.0	0	274.0	0.0	629.0	1.0	1064.0

Tính tổng I

```
i_df['sum'] = i_df[select_device].sum(axis=1)
i_df.head()
```

	time	light	socket	heater	aircond1	aircond2	aircond3	indcooker	sum
0	2020-07-15 00:00:00	0.0	0.965	0.0	1.52	0.0	2.86	0.25	5.595
1	2020-07-15 00:01:00	0.0	0.955	0.0	1.53	0.0	2.86	0.25	5.595
2	2020-07-15 00:02:00	0.0	0.950	0.0	1.53	0.0	2.86	0.25	5.590
3	2020-07-15 00:03:00	0.0	0.950	0.0	1.53	0.0	2.87	0.25	5.600
4	2020-07-15 00:04:00	0.0	0.790	0.0	1.53	0.0	2.86	0.25	5.430

Tiếp theo cần phải gán nhãn cho dữ liệu. Dựa vào bảng đo công suất tiêu thụ, tại mỗi thời điểm t, thiết bị nào có công suất tiêu thụ lớn hơn $threshold$ thì xác định là thiết bị đó đang bật.

Có n thiết bị, mỗi thiết bị có 2 trạng thái là on và off nên sẽ có tổng số 2^n trường hợp, hay 2^n nhãn lớp.

Ví dụ có 5 thiết bị: A, B, C, D, E thì sẽ có $2^5 = 32$ nhãn lớp.

Giả sử tại một thời điểm t nào đó:

A đang bật => 1

B đang tắt => 0

C đang bật => 1

D đang bật => 1

E đang tắt => 0

Thì nhãn lớp sẽ là 10110 (ở cơ số 2) tương ứng với 22 (ở cơ số 10)

```
threshold = 10

def set_label(row):
    x = 0
    for i, name in enumerate(select_device):
        if row[name] > 10:
            x += 2**i
    return x

p_df['label'] = p_df.apply(set_label, axis=1)

p_df.sample(10)
```

	time	light	socket	heater	aircond1	aircond2	aircond3	indcooker	sum	label
1597	2020-07-16 02:37:00	0.0	101.0	0	241.0	0.0	620.0	1.0	963.0	52
14613	2020-07-25 03:33:00	0.0	257.0	0	0.0	484.0	32.0	1.0	774.0	56
1497	2020-07-16 00:57:00	0.0	209.0	0	364.0	0.0	627.0	1.0	1201.0	52
16834	2020-07-26 16:34:00	8.5	733.0	0	0.0	0.0	2025.0	1.0	2767.5	48
22261	2020-07-30 11:01:00	16.5	321.0	0	0.0	0.0	1344.0	2.5	1684.0	112
18776	2020-07-28 00:56:00	0.0	236.0	0	150.0	0.0	596.0	1.0	983.0	52
15682	2020-07-25 21:22:00	10.5	301.0	0	0.0	0.0	2049.0	1.0	2361.5	112
23578	2020-07-31 08:58:00	128.0	250.0	0	0.0	0.0	1107.0	1.5	1486.5	112
8371	2020-07-20 19:31:00	22.0	28.0	0	0.0	0.0	0.0	0.5	50.5	96
10122	2020-07-22 00:42:00	0.0	239.0	0	285.0	0.0	620.0	1.0	1145.0	52

Tiếp theo ta sẽ tách các cột tổng đã được tính trong các DataFrame ra để tạo thành một DataFrame mới, chính là dữ liệu sẽ cho vào mô hình để train và test.

Lấy các cột tổng ra cùng với label tương ứng

```
data = pd.DataFrame()
data['P'] = p_df['sum']
data['U'] = u_df['u']
data['I'] = i_df['sum']
data['Label'] = p_df['label']

data.sample(10)
```

	P	U	I	Label
1718	1137.0	228	542.0	52
4601	1015.0	230	545.0	52
9422	2155.0	223	988.0	48
18626	1741.0	225	825.0	52
3582	2296.0	223	1054.0	48
11856	270.5	233	204.0	36
4468	1209.5	229	625.0	52
13160	1102.0	230	532.0	52
4949	119.0	227	90.0	32
2686	2236.0	226	1035.5	112

Tiếp theo cần đưa dữ liệu về hai tập X và y ở dạng Numpy thì mới đưa được vào mô hình. X là một ma trận 2 chiều, có 3 cột P, U, I và mỗi dòng tương ứng với một điểm dữ liệu.

```
1.  X = data[['P', 'U', 'I']].to_numpy()
```

Tập nhãn lớp y cần đưa về dạng one-hot coding vector như trong bài báo.

Ví dụ:

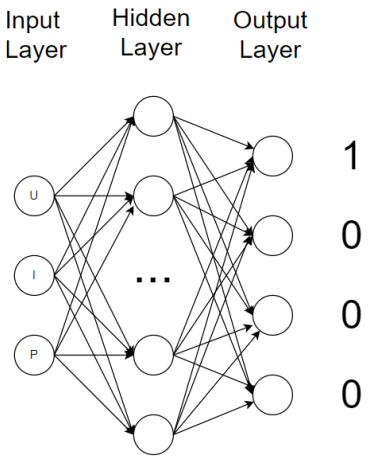
- label = 0 tương ứng với vector [1, 0, 0, ..., 0]
- label = 1 tương ứng với vector [0, 1, 0, ..., 0]
- label = 2 tương ứng với vector [0, 0, 1, ..., 0]
- ...
- label = $2^n - 1$ (n = số thiết bị) tương ứng với vector [0, 0, 0, ..., 1]

Số lượng nơ-ron ở tầng đầu ra = độ dài vector output = $2^n - 1$

```
1.  y = [[0 for i in range(2**len(select_device))] for j in range(len(data))]
2.  for i in range(len(data)):
```

```
3.      y[i][data['Label'][i]] = 1
4.
```

4.1.2.2. Đưa dữ liệu vào mô hình



Mô hình MLP

Thư viện sklearn cho phép ta tạo ra một mô hình mạng MLP Classifier chỉ với vài dòng lệnh

```
1.  clf = MLPClassifier(solver='adam',
2.                      alpha=1e-5,
3.                      hidden_layer_sizes=100,
4.                      random_state=1,
5.                      max_iter=1000,
6.                      verbose=True,
7.                      learning_rate='adaptive',
8.                      n_iter_no_change=20)
9.
```

Danh sách tham số và ý nghĩa của chúng bạn đọc vui lòng xem ở phần phụ lục, hoặc truy cập vào website documentation của sklearn để tìm hiểu thêm.

Tiếp theo chia tập dữ liệu làm hai phần, 70% cho train, 30% cho test. Tỷ lệ chia 70-30, lấy theo thứ tự, không xáo trộn (để dễ dàng so sánh kết quả với các giải thuật hoặc mô hình khác)

```
1.  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1,
    shuffle=False)
```

Thực hiện train mô hình:

```
1.  clf.fit(X_train, y_train)
```

Kiểm tra độ chính xác của mô hình:

```
1. clf.score(X_test, y_test)
```

4.1.2.3. Kết quả

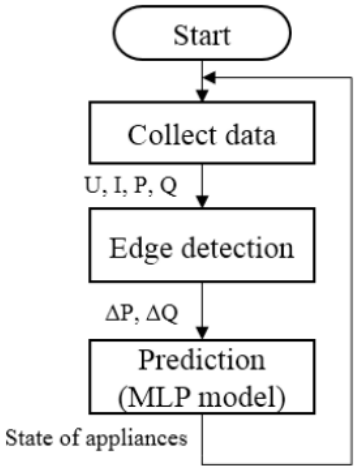
Độ chính xác khi đo với số lượng ít thiết bị thì cao, thấp dần khi tăng số thiết bị lên.

Select_device	Độ chính xác
['heater', 'indcooker']	0.985
['heater', 'indcooker', 'aircond1']	0.889
['heater', 'indcooker', 'aircond1', 'aircond2', 'aircond3']	0.597
['heater', 'indcooker', 'aircond1', 'aircond2', 'aircond3', 'socket', 'light']	0.187

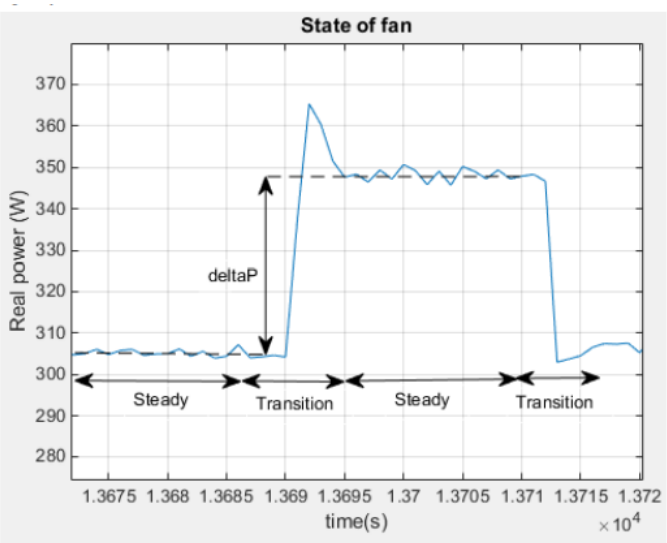
4.2. Xây dựng theo phương pháp của bài báo thứ hai

4.2.1. Tóm tắt nội dung bài báo thứ hai

Bài báo này có một hướng đi khác, đó là xác định hai đại lượng ΔP và ΔQ bằng thuật toán Edge Detection. Từ đó đưa ΔP và ΔQ vào mô hình để xác định thiết bị nào vừa mới bật hoặc tắt.



Sơ đồ luồng thuật toán dự đoán trạng thái thiết bị [3]



Phát hiện rising và falling edge [3]

Sau khi nhận được các độ đo, bộ xử lí chạy thuật toán edge detection để phát hiện sự thay đổi trạng thái của thiết bị, tính ra ΔP và ΔQ. Mỗi khi thiết bị bật hoặc tắt, đồ thị P và Q sẽ thay đổi đáng kể. Tác giả đặt một mức ngưỡng dao động tối thiểu là 15W cho P và 8VAr cho Q. Tức là khi P thay đổi lớn hơn 15W thì xác định là có thay đổi trạng thái, ngược lại thì không. Tương tự với Q.

Mô hình MLP tác giả đề xuất có kiến trúc như sau:

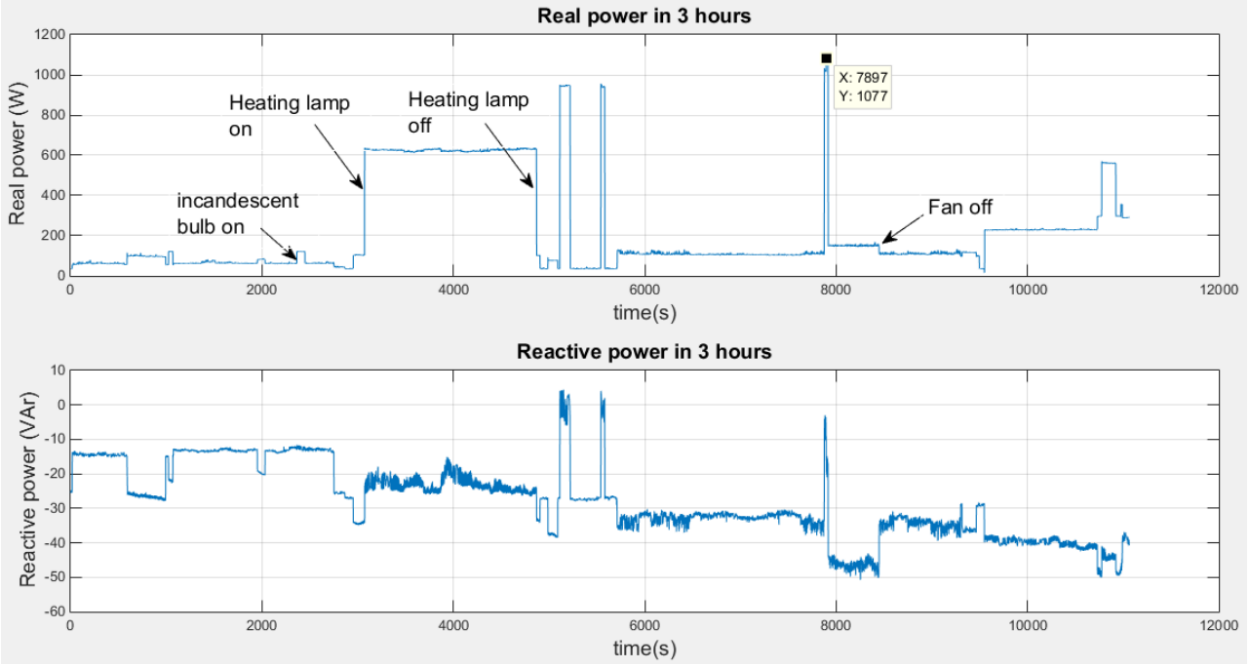
- Input layer: 2 nơ-ron tương ứng với ΔP và ΔQ
- Hidden layer: tác giả không đề cập
- Output layer: 32 nơ-ron. (bằng 2 * số thiết bị, vì mỗi thiết bị có 2 trạng thái bật và tắt, ở đây tác giả thực nghiệm với 12 thiết bị).

Table 2. Output vector **y**

Output	1	2	3	…	32
y (Output vector)	$\begin{bmatrix}1\\0\\0\\...\\0\end{bmatrix}$	$\begin{bmatrix}0\\1\\0\\...\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\\1\\...\\0\end{bmatrix}$	…	$\begin{bmatrix}0\\0\\0\\...\\1\end{bmatrix}$

Table 3. Device states correspond to each output

Output	State
1	Hair dryer (mode 1) is on
2	Hair dryer (mode 1) is off
3	Hair dryer (mode 2) is on
…	…
32	Heating bag is off



Đồ thị năng lượng tiêu thụ trong ngày thứ 4 [3]

Kết quả:

- Thu thập 215 điểm dữ liệu cho train và test.
  - Chia tỷ lệ 70-30
  - Độ chính xác trên tập test đạt 93.65%
- Triển khai thực tế trong 3 tuần tại một ngôi nhà
  - 766 điểm dữ liệu

- Độ chính xác 93.6%

4.2.2. Triển khai theo phương pháp của bài báo

4.2.2.1. Chuẩn bị dữ liệu

Ở đây chỉ sử dụng hai bảng P và cosphi. Đọc P và cosphi từ file .csv vào:

```
1.  # Power dataframe
2.  p_df = pd.read_csv('./data/W.csv',\
3.                         names=['time', 'light', 'socket', 'heater', 'aircond1', 'aircond2',
    'aircond3', 'indcooker'],\
4.                         header=0)
5.
6.  # Power factor dataframe
7.  pf_df = pd.read_csv('./data/cosphi.csv',\
8.                         names=['time', 'light', 'socket', 'heater', 'aircond1', 'aircond2',
    'aircond3', 'indcooker'],\
9.                         header=0)
10.
```

Bảng công suất tiêu thụ

```
p_df
```

	time	light	socket	heater	aircond1	aircond2	aircond3	indcooker
0	2020-07-15 00:00:00	0.0	216.0	0	244.0	0.0	631.0	1.0
1	2020-07-15 00:01:00	0.0	210.0	0	286.0	0.0	631.0	3.5
2	2020-07-15 00:02:00	0.0	206.0	0	314.0	0.0	633.0	6.0
3	2020-07-15 00:03:00	0.0	206.0	0	254.0	0.0	632.0	3.5
4	2020-07-15 00:04:00	0.0	160.0	0	274.0	0.0	629.0	1.0
...	...	...	...	...	...	...	...	...
31675	2020-08-05 23:55:00	0.0	236.0	0	183.0	0.0	0.0	1.0
31676	2020-08-05 23:56:00	0.0	238.0	0	174.0	0.0	0.0	1.0
31677	2020-08-05 23:57:00	0.0	243.0	0	199.0	0.0	0.0	1.0
31678	2020-08-05 23:58:00	0.0	241.0	0	229.0	0.0	0.0	1.0
31679	2020-08-05 23:59:00	0.0	246.0	0	245.0	0.0	0.0	1.0

31680 rows × 8 columns

Bảng hệ số công suất

```
pf_df
```

	time	light	socket	heater	aircond1	aircond2	aircond3	indcooker
0	2020-07-15 00:00:00	0.0	0.975	0.0	0.700	0.0	0.960	0.02
1	2020-07-15 00:01:00	0.0	0.955	0.0	0.815	0.0	0.960	0.06
2	2020-07-15 00:02:00	0.0	0.940	0.0	0.895	0.0	0.965	0.10
3	2020-07-15 00:03:00	0.0	0.940	0.0	0.720	0.0	0.955	0.06
4	2020-07-15 00:04:00	0.0	0.865	0.0	0.775	0.0	0.955	0.02
...	...	...	...	...	...	...	...	...
31675	2020-08-05 23:55:00	0.0	0.975	0.0	0.675	0.0	0.000	0.02
31676	2020-08-05 23:56:00	0.0	0.980	0.0	0.645	0.0	0.000	0.02
31677	2020-08-05 23:57:00	0.0	0.985	0.0	0.740	0.0	0.000	0.02
31678	2020-08-05 23:58:00	0.0	0.970	0.0	0.850	0.0	0.000	0.02
31679	2020-08-05 23:59:00	0.0	0.980	0.0	0.910	0.0	0.000	0.02

31680 rows × 8 columns

Tiếp theo tính Q (công suất phản kháng) dựa trên P và cosphi, áp dụng công thức:

$$Q = \tan(arccos(cosphi)) * P$$

```
# Q dataframe
q_df = pd.DataFrame()
q_df['time'] = p_df['time']
column_names = ['light', 'socket', 'heater', 'aircond1', 'aircond2', 'aircond3', 'indcooker']

for col_name in column_names:
    q_df[col_name] = np.tan(np.arccos(pf_df[col_name])) * p_df[col_name]

q_df
```

	time	light	socket	heater	aircond1	aircond2	aircond3	indcooker
0	2020-07-15 00:00:00	0.0	49.226923	0.0	248.929791	0.0	184.041667	49.989999
1	2020-07-15 00:01:00	0.0	65.222218	0.0	203.344581	0.0	184.041667	58.228239
2	2020-07-15 00:02:00	0.0	74.768016	0.0	156.496947	0.0	172.025051	59.699246
3	2020-07-15 00:03:00	0.0	74.768016	0.0	244.818628	0.0	196.287817	58.228239
4	2020-07-15 00:04:00	0.0	92.813292	0.0	223.428875	0.0	195.356071	49.989999
...	...	...	...	...	...	...	...	...
31675	2020-08-05 23:55:00	0.0	53.784971	0.0	200.030584	0.0	0.000000	49.989999
31676	2020-08-05 23:56:00	0.0	48.327961	0.0	206.151577	0.0	0.000000	49.989999
31677	2020-08-05 23:57:00	0.0	42.569243	0.0	180.876712	0.0	0.000000	49.989999
31678	2020-08-05 23:58:00	0.0	60.400293	0.0	141.921453	0.0	0.000000	49.989999
31679	2020-08-05 23:59:00	0.0	49.952431	0.0	111.625298	0.0	0.000000	49.989999

31680 rows × 8 columns

Tương tự cách làm với bài báo 1, ta chọn một số thiết bị:

```
select_device = ['heater', 'indcooker', 'aircond1']
```

Tính tổng P, Q để mô phỏng đó là giá trị đo được từ thiết bị đo:

```
p_sum = p_df[select_device].sum(axis=1).to_numpy()
q_sum = q_df[select_device].sum(axis=1).to_numpy()

```

Tại mỗi thời điểm t, tính $\Delta P = P[t] - P[t-1]$, $\Delta Q = Q[t] - Q[t-1]$

Lấy các điểm có delta P và delta Q ở ngưỡng nhất định: chỉ lấy các điểm có deltaP và deltaQ lớn hơn `min_delta_p`

`threshold` là ngưỡng công suất xác định thiết bị đang ở trạng thái bật hay tắt.

- Nhãn = 0 tương ứng với thiết bị 0 bật
- Nhãn = 1 tương ứng với thiết bị 0 tắt
- Nhãn = 2 tương ứng với thiết bị 1 bật
- Nhãn = 3 tương ứng với thiết bị 1 tắt
- ...
- Nhãn `2 * i` tương ứng với thiết bị `i` bật
- Nhãn `2 * i + 1` tương ứng với thiết bị `i` tắt

```
data = []
label = []
min_delta_p = 20
threshold = 10

# For each time t
for t in range(len(p_sum) - 1):

    # Calculate delta P and delta Q
    delta_p = p_sum[t+1] - p_sum[t]
    delta_q = q_sum[t+1] - q_sum[t]

    # Only consider if |delta P| > min_delta_p:
    if abs(delta_p) > min_delta_p:

        # For each device
        for i in range(len(select_device)):

            # Check if the device has just turned on or off, then add to dataset
            if p_df[select_device[i]].iloc[t] < threshold and p_df[select_device[i]].iloc[t+1] >= threshold:
                data.append([delta_p, delta_q])
                label.append(2*i)
                break
            elif p_df[select_device[i]].iloc[t] >= threshold and p_df[select_device[i]].iloc[t+1] < threshold:
                data.append([delta_p, delta_q])
```

```
                label.append(2*i+1)
                break

```

Đưa nhãn lớp về dạng one-hot vector

- Nhãn 0 tương ứng với [1, 0, 0, .. , 0]
- Nhãn 1 tương ứng với [0, 1, 0, .. , 0]
- ...
- Nhãn 2n - 1 tương ứng với [0, 0, 0, .., 1]

Độ dài vector = 2n (n là số thiết bị)

```
X = data

def one_hot(y):
    ret = [0 for i in range(len(select_device) * 2)]
    ret[y] = 1
    return ret

y = list(map(one_hot, label))

```

	delta P	delta Q	label
0	-146.5	-28.612323	[0, 0, 0, 0, 0, 1]
1	573.5	884.529521	[0, 0, 1, 0, 0, 0]
2	-574.0	-847.121836	[0, 0, 0, 1, 0, 0]
3	560.0	848.667388	[0, 0, 1, 0, 0, 0]
4	-560.0	-848.667388	[0, 0, 0, 1, 0, 0]
...	...	...	...
922	52.0	118.694324	[0, 0, 0, 0, 1, 0]
923	-63.5	-106.033716	[0, 0, 0, 0, 0, 1]
924	221.5	279.556311	[0, 0, 0, 0, 1, 0]
925	-217.5	-113.847320	[0, 0, 0, 0, 0, 1]
926	108.5	105.140511	[0, 0, 0, 0, 1, 0]

927 rows × 3 columns

Dữ liệu đã được chuẩn bị

4.2.2.2. Đưa dữ liệu vào mô hình

Chia tập train/test tương tự với bài báo 1:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)

```

Tạo mô hình bằng sklearn và thực hiện train:

```
from sklearn.neural_network import MLPClassifier
clf = MLPClassifier(solver='adam',
                    alpha=1e-5,
                    hidden_layer_sizes=30,
                    random_state=1,
                    max_iter=1000,
                    verbose=True,
                    learning_rate='adaptive',
                    n_iter_no_change=20)
clf.fit(X_train, y_train)
```

4.2.2.3. Kết quả

Kết quả kiểm tra trên tập test:

Select_device	Độ chính xác
['heater', 'indcooker']	0.900
['heater', 'indcooker', 'aircond1']	0.813
['heater', 'indcooker', 'aircond1', 'aircond2', 'aircond3']	0.697
['heater', 'indcooker', 'aircond1', 'aircond2', 'aircond3', 'socket', 'light']	0.474

4.3. Kết hợp hai bài báo

4.3.1. Ý tưởng

Ta sẽ sử dụng 6 thông số U, I, P, Q, $\Delta P, \Delta Q$ để dự đoán những thiết bị nào đang bật, những thiết bị nào đang tắt, tại một thời điểm t nào đó.

Kiến trúc mạng MLP:

- Tầng input: 6 nơ-ron tương ứng với 6 thông số: U, I, P, Q, $\Delta P, \Delta Q$
- Tầng hidden: 30 nơ-ron
- Tầng output: n nơ-ron (n = số thiết bị)

4.3.2. Triển khai

4.3.2.1. Chuẩn bị dữ liệu

Đọc các dữ liệu U, I, P, cosphi và tính bảng Q như các thao tác trước đây.

Tính tổng P, Q, I để mô phỏng dữ liệu đo thực tế.

Đánh nhãn cho tập dữ liệu:

Tại mỗi thời điểm t, tính $\Delta P = P[t] - P[t-1]$, $\Delta Q = Q[t] - Q[t-1]$

Xác định nhãn lớp bằng cách kiểm tra đồ thị công suất của từng thiết bị, nếu lớn hơn mức ngưỡng thì thiết bị đó là bật.

Rồi thêm vào tập dữ liệu

```
data = []
label = []
threshold = 10  # ngưỡng xác định thiết bị là bật > 10W
for i in range(1, len(p_sum)):
    delta_p = p_sum[i] - p_sum[i-1]
    delta_q = q_sum[i] - q_sum[i-1]
    data.append([u_sum[i], i_sum[i], p_sum[i], q_sum[i], delta_p, delta_q])
    x = 0
    for j, device_name in enumerate(select_device):
        if p_df[device_name].iloc[i] > threshold:
            x += 2**j

    label.append(x)

```

Đưa label về dạng vector 0, 1

Ví dụ có 5 thiết bị: A, B, C, D, E:

- A đang bật => 1
- B đang tắt => 0
- C đang bật => 1
- D đang bật => 1
- E đang tắt => 0

Thì nhãn lớp là vector [1,0,1,1,0]

```
# Hàm chuyển đổi nhãn lớp về dạng vector

```

```
# VD có 5 thiết bị
# tobase2(3) sẽ trả về [0, 0, 0, 1, 1]

def tobase2(n):
    length = len(select_device)
    ret = [0 for i in range(length)]
    i = length - 1
    while n > 0:
        ret[i] = n % 2
        n = n // 2
        i -= 1
    return ret

X = data
y = [tobase2(label[i]) for i in range(len(label))]

```

Dữ liệu đã sẵn sàng để đưa vào mô hình

	delta P	delta Q	P	Q	U	I	label
0	230	559.5	1130.5	510.836704	38.5	-21.351676	[0, 1, 0, 1, 0, 1, 0]
1	230	559.0	1159.0	462.989261	28.5	-47.847443	[0, 1, 0, 1, 0, 1, 0]
2	230	560.0	1095.5	574.102699	-63.5	111.113439	[0, 1, 0, 1, 0, 1, 0]
3	230	543.0	1064.0	561.588237	-31.5	-12.514463	[0, 1, 0, 1, 0, 1, 0]
4	230	525.5	1086.0	405.119945	22.0	-156.468291	[0, 1, 0, 1, 0, 1, 0]
...	...	...	...	...	...	...	...
31674	230	248.0	420.0	303.805554	-53.0	69.748154	[0, 0, 0, 1, 0, 1, 0]
31675	230	248.0	413.0	304.469537	-7.0	0.663983	[0, 0, 0, 1, 0, 1, 0]
31676	231	249.0	443.0	273.435954	30.0	-31.033583	[0, 0, 0, 1, 0, 1, 0]
31677	231	249.0	471.0	252.311746	28.0	-21.124208	[0, 0, 0, 1, 0, 1, 0]
31678	232	249.0	492.0	211.567727	21.0	-40.744019	[0, 0, 0, 1, 0, 1, 0]

31679 rows × 7 columns

4.3.2.2. Đưa dữ liệu vào mô hình

Chia tập train/test vẫn như cách chia cũ:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1,
shuffle=False)

```

Train:

```
from sklearn.neural_network import MLPClassifier
clf = MLPClassifier(solver='adam',
                    alpha=1e-5,
```

```
                    hidden_layer_sizes=30,
                    random_state=1,
                    max_iter=1000,
                    verbose=True,
                    learning_rate='adaptive',
                    n_iter_no_change=20)
clf.fit(X_train, y_train)

```

Kiểm tra độ chính xác của mô hình trên tập test

```
clf.score(X_test, y_test)
```

4.3.2.3. Kết quả

Select_device	Độ chính xác
['heater', 'indcooker']	0.988
['heater', 'indcooker', 'aircond1']	0.870
['heater', 'indcooker', 'aircond1', 'aircond2', 'aircond3']	0.527
['heater', 'indcooker', 'aircond1', 'aircond2', 'aircond3', 'socket', 'light']	0.195

4.4. Đi sâu vào xây dựng mạng MLP

Để xây dựng một mạng nơ-ron nhân tạo nói chung, mạng MLP nói riêng, cần phải thực hiện các bước sau đây:

- Lựa chọn kiến trúc mạng
- Khởi tạo trọng số
- Chọn activation function
- Xây dựng bộ giải
- Xác định hàm lỗi
- Lan truyền tiến
- Lan truyền ngược
- Xây dựng hàm đánh giá

Phần này sẽ đi làm chi tiết các bước này, để xây dựng một mạng MLP tổng quát (không gắn liền với bài toán).

4.4.1. Lựa chọn kiến trúc mạng

Kiến trúc mạng MLP luôn bao gồm 3 tầng: Input Layer, Hidden Layer và Output Layer. Việc lựa chọn kiến trúc mạng MLP chỉ đơn thuần là lựa chọn số tầng và số nơ-ron cho từng tầng của Hidden Layer. Còn tầng Input sẽ có số nút bằng với số đặc trưng của dữ liệu. Tầng Output có số nút bằng với số chiều của nhãn lớp.

Việc lựa chọn số tầng và số nút cho từng tầng trong Hidden Layer là khó khăn nhất. Có một số cách như sau [4]:

- Dựa vào thực nghiệm: Nói chung không có công thức nào để xác định số tầng/nút cho Hidden Layer một cách tối ưu. Chỉ có thể dựa vào thực nghiệm. Thử lại nhiều lần với các giá trị khác nhau để tìm ra tham số tối ưu.
- Dựa vào trực giác: Nếu bài toán của chúng ta đơn giản thì khả năng rất cao là sẽ tối ưu khi ta sử dụng một tầng. Còn nếu bài toán của chúng ta phức tạp thì có thể cần nhiều tầng mới có thể khớp với dữ liệu.
- Mượn ý tưởng: Dựa vào các nghiên cứu có trước.
- Tìm kiếm: Tạo một cơ chế tự động tìm và kiểm tra xem bao nhiêu layer, bao nhiêu node là tối ưu. Các chiến lược tìm kiếm: Random, grid, heuristic, exhaustive.

4.4.2. Khởi tạo trọng số

Có một nguyên tắc đó là không được khởi tạo tất cả trọng số **W** bằng 0. Vì như vậy mạng nơ ron sẽ không hoạt động được. (Vì khi đó tất cả các nút của mỗi tầng sẽ luôn có trọng số như nhau). Nên khởi tạo trọng số **W** bằng các giá trị ngẫu nhiên nhỏ. Tại sao không nên khởi tạo **W** lớn? Bởi vì khi sử dụng activation function là các hàm vd như sigmoid hay tanh, thì giá trị đầu vào lớn sẽ khiến đạo hàm tại các điểm đó gần như bằng 0, khiến cho sau nhiều vòng lặp mà mô hình không thay đổi là bao, dẫn tới mô hình khó hội tụ.

Việc khởi tạo bộ bias **b** bằng 0 thì không sao, có người cho rằng nên khởi tạo **b** bằng 0, cũng có người nói nên khởi tạo ngẫu nhiên. Cái này tuỳ thuộc vào bài toán và ta nên thử nghiệm từng trường hợp.

4.4.3. Chọn activation function

Tại sao lại cần activation function? Bởi vì nếu như không có activation function thì cả mạng nơ-ron cũng chẳng khác gì một mô hình tuyến tính. Việc lựa chọn activation function cũng là một vấn đề trong việc xây dựng mạng MLP. Việc chọn activation function cho mạng MLP bao gồm 2 phần: chọn activation function cho Hidden Layer và Output Layer.

Các activation function phổ biến bao gồm hàm relu, sigmoid, tanh, identity.

Hàm ReLU: $\boldsymbol{f(x) = \max(0, x)}$

Hàm ReLU (viết tắt của Rectangle Linear Unit), được sử dụng phổ biến nhất vì độ đơn giản và hiệu quả của nó. Việc sử dụng hàm Relu sẽ giúp mô hình hội tụ nhanh hơn, vì đạo hàm tại mọi điểm x > 0 luôn bằng 1.

Hàm Sigmoid: $\boldsymbol{f(x) = \frac{1}{1+e^{-x}}}$

Hàm sigmoid hay còn gọi là hàm logistic. Nó có tác dụng đưa giá trị đầu vào về khoảng (0,1). Hàm sigmoid thường hay được áp dụng ở tầng output.

Hàm Tanh: $\boldsymbol{f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}}$

Hàm tanh cũng có tính chất tương tự như hàm sigmoid, chỉ khác ở chỗ nó đưa giá trị đầu ra về khoảng (-1, 1)

Hàm Identity: $\boldsymbol{f(x) = x}$

Hàm Identity được sử dụng trong một số ít trường hợp. Đa phần sử dụng trong các bài toán hồi quy.

4.4.4. Xây dựng bộ giải

Có nhiều bộ giải có thể lựa chọn để xây dựng, với những ưu nhược điểm khác nhau:

- Gradient Descent
    - Dễ tính toán
    - Dễ hiểu
    - Dễ triển khai
    - Có thể bị mắc tại cực tiểu địa phương
    - Không phù hợp với bộ dữ liệu lớn (vì tốn bộ nhớ)

- Stochastic Gradient Descent
    - Cập nhật trọng số thường xuyên, mô hình hội tụ nhanh hơn
    - Cần ít bộ nhớ hơn
    - Phương sai cao trong các tham số mô hình
- Mini-batch Gradient Descent
    - Cập nhật trọng số thường xuyên và tham số ít phương sai hơn
    - Cần bộ nhớ trung bình
- Momentum
    - Giảm dao động và phương sai của các tham số
    - Hội tụ nhanh hơn Gradient Descent
    - Thêm một hyper paparameter cần phải hiệu chỉnh thủ công để có được độ chính xác cao.
- Nesterov Accelerated Gradient
    - Không bị kẹt tại cực tiểu địa phương
    - Thêm một hyper parameter
- Adagrad
    - Thay đổi tốc độ học cho từng tham số
    - Không cần điều chỉnh tốc độ học thủ công
    - Tính toán chi phí cao hơn
    - Tốc độ học luôn giảm dần có thể dẫn đến đóng băng
- AdaDelta
    - Giới hạn độ giảm của learning rate
    - Tính toán chi phí cao hơn
- Adam
    - Hội tụ nhanh
    - Chi phí tính toán cao

Chúng tôi lựa chọn xây dựng bộ giải Adam vì nó luôn được đánh giá cao. Adam là sự kết hợp của RMSProp và Momentum.

4.4.5. Xác định hàm lỗi

Hàm lỗi là hàm đánh giá độ sai lệch của mô hình đối với bộ dữ liệu train. Hàm lỗi càng nhỏ thì mô hình càng khớp với bộ dữ liệu. Bản chất của việc train mô hình cũng chính là việc tối ưu hàm lỗi.

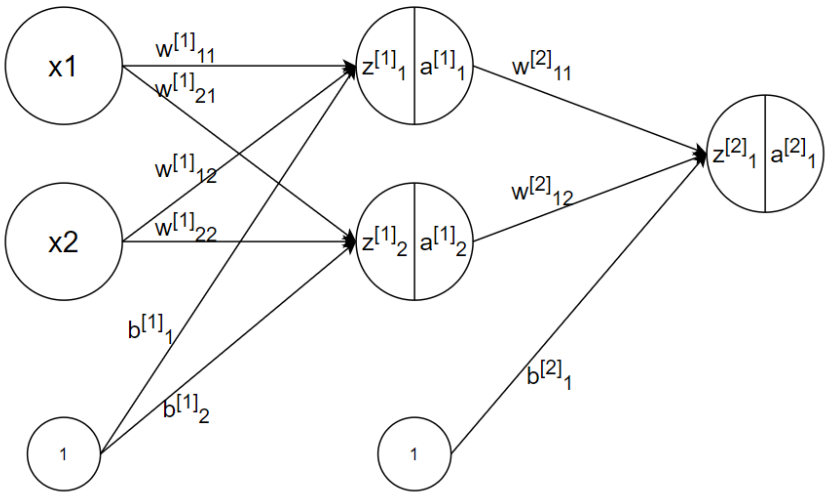
Đối với mạng MLP Classifier, hàm lỗi khuyên dùng là hàm Cross Entropy hay hàm Log Loss:

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i))$$

Còn đối với mạng MLP Regression, hàm lỗi khuyên dùng là hàm MSE:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2$$

4.4.6. Lan truyền tiến



Ví dụ mạng MLP đơn giản

Giả sử có một mạng MLP đơn giản gồm 3 tầng, input layer gồm 2 nút, hidden layer gồm 2 nút và output layer gồm 1 nút như hình vẽ. Giải thích ký hiệu:

x_1, x_2 là hai tín hiệu đầu vào, hay nói cách khác là vector đầu vào có 2 chiều.

Chỉ số đặt trong dấu ngoặc vuông thể hiện cho tầng thứ mấy. Tầng đầu vào là tầng [0].

w là trọng số kết nối các nơ ron

Ví dụ $w_{21}^{[1]}$ là trọng số kết nối nơ ron thứ nhất của tầng [0] với nơ ron thứ 2 của tầng [1].

b là bias, hình dung như liên kết kết nối nút có giá trị bằng 1 với nơ-ron.

z là giá trị tổng của các liên kết trỏ đến nút hiện tại.

Ví dụ trong hình, $z_1^{[1]} = x_1 * w_{11}^{[1]} + x_2 * w_{12}^{[1]} + b_1^{[1]}$

a là giá trị của *z* sau khi đã áp dụng activation function.

$$a = g(z)$$

Tuỳ trường hợp, hàm g có thể là hàm sigmoid, tanh, relu,…

$g^{[1]}$ là activation cho tầng 1, $g^{[2]}$ là activation cho tầng 2,…

Ký hiệu ma trận $z^{[1]} = [\ z_1^{[1]} \quad z_2^{[1]}\ ]$. Ký hiệu tương tự với w, b, a, x.

Lan truyền tiến là việc tính toán các nút trong mạng từ input đến hidden rồi đến output.

$$z^{[1]} = w^{[1]}.x + b^{[1]}$$

$$a^{[1]} = g^{[1]}\ (\ z^{[1]})$$

$$z^{[2]} = w^{[2]}.a^{[1]} + b^{[2]}$$

$$a^{[2]} = g^{[2]}(z^{[2]})$$

Trong đó (.) là phép nhân ma trận.

Kết quả $a^{[2]}$ là kết quả của tầng đầu ra, chính là output của mạng, người ta còn hay ký hiệu $a^{[2]} =$ ŷ. Trường hợp này đầu ra là vector có 1 chiều, thực tế có thể có nhiều chiều.

4.4.7. Lan truyền ngược

Sau khi có được output ŷ, ta đem so sánh nó với y (là giá trị đầu ra mong muốn, có sẵn trong tập dữ liệu), tính bằng hàm lỗi, ví dụ trong trường hợp sử dụng hàm log loss làm hàm lỗi:

$$L = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)}.\log\left(\hat{y}^{(i)}\right) + \left(1 - y^{(i)}\right).\log\left(1 - \hat{y}^{(i)}\right)$$

Trong đó m là tổng số ví dụ học. Kí hiệu (i) là thể hiện cho ví dụ học thứ i.

Việc tính giá trị của L không thật sự cần thiết trong pha lan truyền ngược, mà ta ghi lại giá trị của L sau nhiều lần lặp và theo dõi sự thay đổi của nó để đưa đến kết luận mô hình đã hội tụ hay chưa. Việc quan trọng hơn đó là tính đạo hàm của L.

Lan truyền ngược là việc tính đạo hàm của L theo từng biến, từ tầng cuối cùng đến tầng đầu tiên. Như ví dụ trong hình vẽ ở phần trước, ta cần tính đạo hàm của L theo $a^{[2]}$, rồi từ đó sử dụng công thức chuỗi đạo hàm (chain rule) để tính đạo hàm của L theo $z^{[2]}$. Từ đó có thể tính đạo hàm của L

theo $w^{[2]}$, đạo hàm của L theo $b^{[2]}$. Các giá trị đạo hàm này được sử dụng để cập nhật trọng số w và b.

Trong code, để ngắn gọn, người ta hay ký hiệu:

- Đạo hàm của L theo $a^{[2]}$ là "$da^{[2]}$"
- Đạo hàm của L theo $z^{[2]}$ là "$dz^{[2]}$"
- Tương tự với w, b, x

Các bước lan truyền ngược cụ thể như sau:

Tính "da$^{[2]}$" $= \frac{dL}{da^{[2]}} = \frac{dL}{d\hat{y}}$

Giá trị này phụ thuộc vào hàm lỗi là gì, vd hàm lỗi là hàm log loss thì:

$$\frac{dL}{d\hat{y}} = -\frac{y}{\hat{y}} - \frac{1-y}{1-\hat{y}}$$

Tính "$dz^{[2]}$" $= \frac{dL}{dz^{[2]}}$.

Áp dụng quy tắc chuỗi đạo hàm:

$$\text{"dz}^{[2]}\text{"} = \frac{dL}{dz^{[2]}} = \frac{dL}{da^{[2]}}.\frac{da^{[2]}}{dz^{[2]}} = "da^{[2]}".\frac{da^{[2]}}{dz^{[2]}}$$

"$da^{[2]}$" vừa tính được ở trên, còn $\frac{da^{[2]}}{dz^{[2]}}$ chính là đạo hàm của hàm activation function theo biến z.

Tức là $\frac{da^{[2]}}{dz^{[2]}} = g'(z^{[2]})$

Phụ thuộc vào hàm g là gì, ta tính được giá trị này. Ví dụ g là hàm sigmoid, thì:

$$g'\left(z^{[2]}\right) = z^{[2]}.(1 - z^{[2]})$$

Tính "$dw^{[2]}$" $= \frac{dL}{dw^{[2]}}$

Áp dụng quy tắc chuỗi đạo hàm:

$$"dw^{[2]}" = \frac{dL}{dw^{[2]}} = \frac{dL}{dz^{[2]}}.\frac{dz^{[2]}}{dw^{[2]}} = "dz^{[2]}".\frac{dz^{[2]}}{dw^{[2]}}$$

Mặt khác, theo pha lan truyền tiến thì

$$z^{[2]} = w^{[2]}.a^{[1]} + b^{[2]}$$

Vì thế:

$$\frac{dz^{[2]}}{dw^{[2]}} = a^{[1]}$$

Thay vào ta tính được "$dw^{[2]}$"

Giá trị "$dw^{[2]}$" sẽ được sử dụng để cập nhật bộ trọng số $w^{[2]}$:

$$w^{[2]} := w^{[2]} - \alpha."dw^{[2]}"$$

Trong đó α là learning rate – tốc độ học.

Tính "$\boldsymbol{db^{[2]}}" = \frac{\boldsymbol{dL}}{\boldsymbol{db^{[2]}}}$

Áp dụng quy tắc chuỗi đạo hàm:

$$"db^{[2]}" = \frac{dL}{db^{[2]}} = \frac{dL}{dz^{[2]}}.\frac{dz^{[2]}}{db^{[2]}} = "dz^{[2]}".\frac{dz^{[2]}}{db^{[2]}}$$

Mặt khác, theo pha lan truyền tiến thì

$$z^{[2]} = w^{[2]}.a^{[1]} + b^{[2]}$$

Vì thế:

$$\frac{dz^{[2]}}{db^{[2]}} = 1$$

Vậy

$$"db^{[2]}" = "dz^{[2]}"$$

Giá trị "$db^{[2]}$" sẽ được sử dụng để cập nhật bộ bias $b^{[2]}$:

$$b^{[2]} := b^{[2]} - \alpha."db^{[2]}"$$

Trong đó α là learning rate – tốc độ học.

Như vậy ta đã xong tầng [2], tiếp tục với tầng [1]:

Tính "da$^{[1]}$" $= \frac{dL}{da^{[1]}}$

Áp dụng quy tắc chuỗi đạo hàm:

$$"da^{[1]}" = \frac{dL}{da^{[1]}} = \frac{dL}{dz^{[2]}}.\frac{dz^{[2]}}{da^{[1]}} = "dz^{[2]}".\frac{dz^{[2]}}{da^{[1]}}$$

Mặt khác, theo pha lan truyền tiến thì

$$z^{[2]} = w^{[2]}.a^{[1]} + b^{[2]}$$

Vì thế:

$$\frac{dz^{[2]}}{da^{[1]}} = w^{[2]}$$

Thay vào ta tính được "$da^{[1]}$"

Tính "dz$^{[1]}$" $= \frac{dL}{dz^{[1]}}$

Việc này tương tự như tính "$dz^{[2]}$" đã được trình bày ở trên.

Tính "$dw^{[1]}$" $= \frac{dL}{dw^{[1]}}$

Việc này tương tự như tính "$dw^{[2]}$" đã được trình bày ở trên.

Ta cũng sử dụng "$dw^{[1]}$" để cập nhật trọng số $w^{[1]}$.

$$w^{[1]} := w^{[1]} - \alpha."dw^{[1]}"$$

Tính "$db^{[2]}$" $= \frac{dL}{db^{[2]}}$

Việc này tương tự như tính "$db^{[2]}$" đã được trình bày ở trên.

Ta cũng sử dụng "$db^{[1]}$" để cập nhật trọng số $b^{[1]}$.

$$b^{[1]} := b^{[1]} - \alpha."db^{[1]}"$$

4.4.8. Xây dựng hàm đánh giá

Hàm đánh giá cho ra kết quả là độ chính xác của mô hình trên một tập dữ liệu.

Đối với bài toán **classification** đơn giản là số ví dụ dự đoán đúng / tổng số ví dụ.

Còn đối với bài **regression**, có nhiều độ đo khác nhau để đánh giá. Ví dụ độ đo R^2:

$$R^2 = 1 - \frac{u}{v}$$

Trong đó:

$$u = \sum_{i=1}^{m} (y^{(i)} - \hat{y}^{(i)})^2$$

(m là số ví dụ học)

Và:

$$v = \sum_{i=1}^{m} \left(y^{(i)} - y_{mean}\right)^2$$

(y_{mean} là giá trị trung bình của các y trong m ví dụ học)

4.5. Dự đoán công suất tiêu thụ của các thiết bị

Sau khi đã xây dựng được mạng MLP theo các bước ở trên, chúng tôi đưa nó vào dự đoán công suất tiêu thụ của các thiết bị. Các tham số đầu vào bao gồm U, I, P, Q.

4.5.1. Chuẩn bị dữ liệu

Như thường lệ, ta đọc dữ liệu, chọn thiết bị, tính Q, tính P, Q, U, I tổng:

```
1.  import pandas as pd
2.  import numpy as np
3.  import matplotlib.pyplot as plt
4.
5.  plt.rcParams['figure.figsize'] = (20, 4)
6.
7.  # Power dataframe
8.  p_df = pd.read_csv('./data/W.csv',
9.                     names=['time', 'light', 'socket', 'heater', 'aircond1', 'aircond2',
    'aircond3', 'indcooker'],
10.                    header=0)
11. # Voltage dataframe
12. u_df = pd.read_csv('./data/V.csv',
```

```
                    names=['time', 'u'],
                    header=0)
# Current dataframe
i_df = pd.read_csv('./data/A.csv',
                    names=['time', 'light', 'socket', 'heater', 'aircond1', 'aircond2',
    'aircond3', 'indcooker'],
                    header=0)
# Power factor dataframe
pf_df = pd.read_csv('./data/cosphi.csv',
                     names=['time', 'light', 'socket', 'heater', 'aircond1', 'aircond2',
    'aircond3', 'indcooker'],
                     header=0)

# Reactive power dataframe
q_df = pd.DataFrame()
q_df['time'] = p_df['time']
column_names = ['light', 'socket', 'heater', 'aircond1', 'aircond2', 'aircond3',
    'indcooker']

# Calculate reactive power using P and cosphi
for col_name in column_names:
    q_df[col_name] = np.tan(np.arccos(pf_df[col_name])) * p_df[col_name]

select_device = ['heater', 'indcooker', 'aircond2']

# Calculate P, Q, U, I sum
p_sum = p_df[select_device].sum(axis=1).to_numpy()
q_sum = q_df[select_device].sum(axis=1).to_numpy()
u_sum = u_df['u'].to_numpy()  # không cần tính tổng U, nhưng cứ đặt là u_sum cho đồng nhất
i_sum = i_df[select_device].sum(axis=1).to_numpy()

```

Tính phần trăm P / P tổng của từng thiết bị

```
data = []
label = []
for t in range(0, len(p_sum)):
    data.append([u_sum[t], i_sum[t], p_sum[t], q_sum[t]])
    percent = []
    for j, device_name in enumerate(select_device):
        if p_sum[t] == 0:
            percent.append(0)
        else:
            percent.append(p_df[device_name].iloc[t] / p_sum[t])
    label.append(percent)

X = np.array(data)
y = np.array(label)

```

4.5.2. Đưa dữ liệu vào mô hình

```
model = MyMLPRegressor(
                    hidden_layer_sizes=(40,),
                    learning_rate=0.001,
                    random_state=0,
                    max_iter=100000,
                    n_iter_no_change=20,
                    tol=0.001,
```

```
                    hidden_activation='tanh',
                    output_activation='sigmoid',
                    solver='adam',
                    loss_func='mse',
                    batch_size=200)
model.fit(X, y)
```

Tính score R^2:

```
score = model.score(X, y)
```

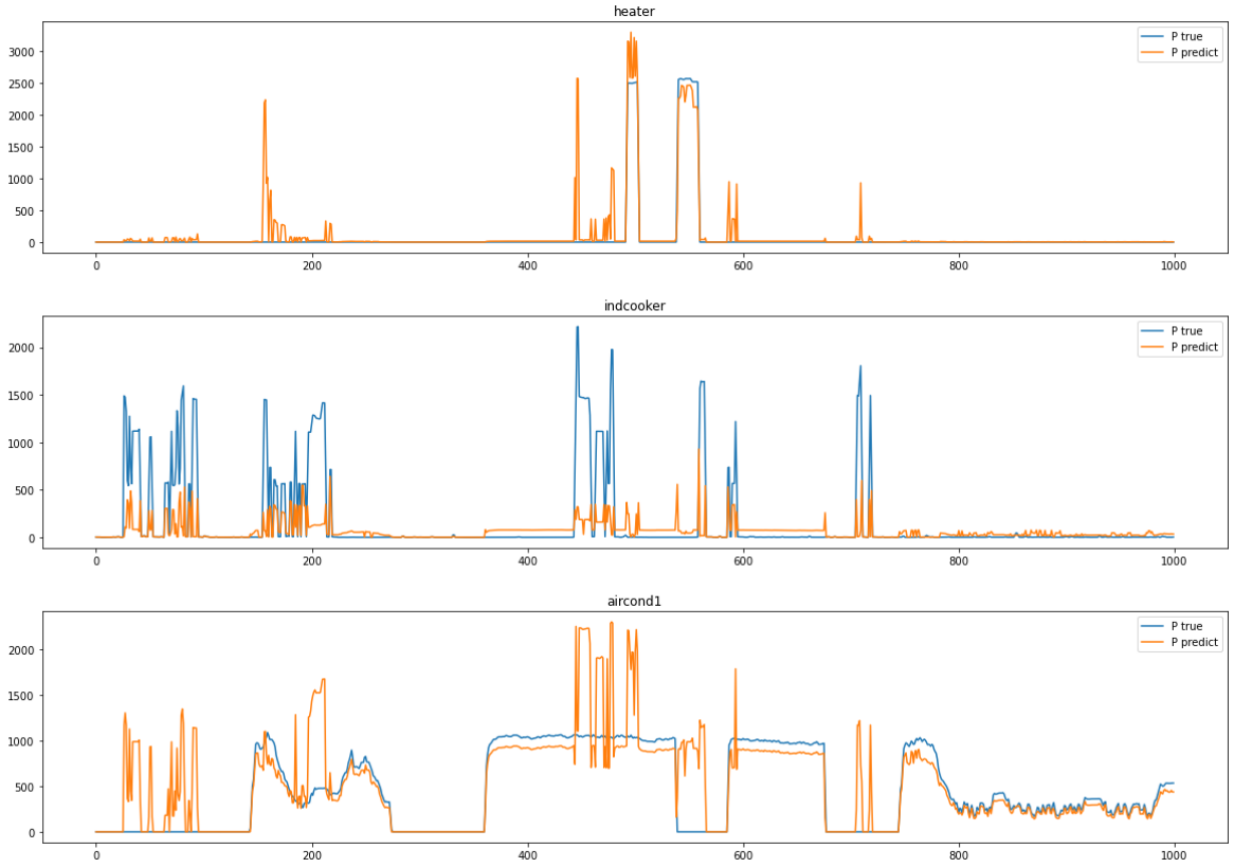
Tính công suất dự đoán và vẽ đồ thị:

```
yhat = model.predict(X)
a = 2000
b = 3000
for i in range(y.shape[1]):
    plt.plot(y[a:b, i] * p_sum[a:b])
    plt.plot(yhat[a:b, i] * p_sum[a:b])
    plt.title(select_device[i])
    plt.legend(['P true', 'P predict'])
    plt.show()

```

4.5.3. Kết quả

Select_device	R^2
['heater', 'indcooker']	0.580
['heater', 'indcooker', 'aircond1']	0.754
['heater', 'indcooker', 'aircond1', 'aircond2', 'aircond3']	0.386
['heater', 'indcooker', 'aircond1', 'aircond2', 'aircond3', 'socket', 'light']	0.593



Trường hợp [‘heater’, ‘indcooker’, ‘aircond1’] với $R^2 = 0.754$

5. Kết luận và hướng phát triển

5.1. Kết luận

Kết quả của việc xây dựng mô hình theo hướng hai bài báo và kết hợp đều khá tốt với bộ dữ liệu. Tuy nhiên kết quả khi dự đoán phần trăm công suất sử dụng của từng thiết bị không được tốt cho lắm. Nguyên nhân có thể là do việc lựa chọn thông số đầu vào.

Thông qua đề tài, sinh viên có thể hiểu hơn về cách hoạt động cũng như xây dựng mạng nơ-ron nhân tạo nói chung và mạng MLP nói riêng; biết cách áp dụng chúng vào bài toán thực tế, có thêm kinh nghiệm trong lĩnh vực học máy và trí tuệ nhân tạo.

5.2. Hướng phát triển

Chúng tôi sẽ tiếp tục nghiên cứu để hướng đến phát triển hoàn thiện sản phẩm như đã đề xuất. Mô hình mạng nơ-ron nhân tạo MLP là có khả thi, tuy nhiên chúng tôi sẽ mở rộng tìm hiểu những giải pháp khác cho bài toán NILM này.

Sau khi phát triển thuật toán đến một mức nhất định, chấp nhận được, chúng tôi sẽ tiến hành tích hợp hệ thống thành một sản phẩm hoàn chỉnh. Hi vọng một ngày nào đó những ý tưởng này không chỉ còn là trên giấy mà sẽ được áp dụng vào thực tế.

6. Phụ lục

MLPClasifier của sklearn có các tham số sau [6]:

hidden_layer_sizes : *tuple, length = n_layers - 2, default=(100,)*

Kích thước tầng ẩn. Phần tử thứ i đại diện cho số nơ-ron của tầng ẩn thứ i

activation : *{'identity', 'logistic', 'tanh', 'relu'}, default='relu'*

Hàm kích hoạt.

solver : *{'lbfgs', 'sgd', 'adam'}, default='adam'*

Bộ giải tối ưu hoá trọng số

- 'lbfgs': một bộ giải thuộc họ phương pháp quasi-Newton
- 'sgd': Stochastic Gradient Descent
- 'adam': Stochastic Gradient-base optimizer đề xuất bởi Kingma, Diederik, và Jimmy Ba. Viết tắt của Adaptive Moment Estimation

Bộ giải adam làm việc tốt trên tập dữ liệu lớn (hàng ngàn mẫu).

Bộ giải lbfgs làm việc tốt trên tập dữ liệu nhỏ

alpha : *float, default=0.0001*

L2 penalty (theo regularization term)

L1: $\sum_{i=1}^{n}(Y_i - \sum_{j=1}^{p} X_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$ Cost function

L2: $\sum_{i=1}^{n}(y_i - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$ Cost function

batch_size : *int, default='auto'*

Kích thước minibatch cho trình tối ưu hoá ngẫu nhiên. Nếu bộ giải là 'lbfgs' thì không sử dụng tham số này. Khi set là 'auto' thì batch_size = min(200, n_samples)

learning_rate : *{'constant', 'invscaling', 'adaptive'}, default='constant'*

- 'constant': luôn là hằng số được cho bởi tham số learning_rate_init
- 'invscaling': giảm dần tốc độ học theo thời gian t. Tốc độ học tính theo công thức

  Learning_rate = learning_rate_init / pow(t, power_t)
- 'adaptive': giữ learning_rate bằng hằng số learning_rate_init miễn là hàm lỗi tiếp tục giảm. Mỗi khi 2 epochs liên tiếp không giảm được ít nhất 'tol', learning rate sẽ được giảm đi 5 lần.

Chỉ sử dụng khi solver là 'sgd'

learning_rate_init : *double, default=0.001*

Giá trị khởi tạo của tốc độ học. Chỉ sử dụng khi solver là 'sgd' hoặc 'adam'

power_t : *double, default=0.5*

Số mũ cho inverse scaling learning rate. Sử dụng khi learning rate đặt là 'invscaling' và solver là 'sgd'

max_iter : *int, default=200*

Số bước lặp tối đa. Bộ giải sẽ lặp cho tới khi hội tụ (được xác định bởi 'tol') hoặc đạt số bước lặp tối đa. Đối với bộ giải 'sgd' và 'adam' thì lưu ý rằng tham số này xác định số epochs (mỗi điểm dữ liệu được sử dụng bao nhiêu lần), chứ không phải số bước gradient descent

shuffle : *bool, default=True*

Mỗi lần lặp có xáo trộn các mẫu hay không. Chỉ sử dụng khi solver = 'sgd' hoặc 'adam'

random_state : *int, RandomState instance, default=None*

Trạng thái ngẫu nhiên. Dùng để tạo ngẫu nhiên bộ trọng số và bias, và lấy mẫu ngẫu nhiên khi solver= 'sgd' hoặc 'adam'. Một random_state cố định thì những lần chạy khác nhau cho ra kết quả giống nhau.

tol : *float, default=1e-4*

Tolerence. Khi hàm lỗi không cải thiện ít nhất là tol cho n_iter_no_changed lần lặp liên tiếp, trừ khi learning_rate đặt là ‘adaptive’, sự hội tụ sẽ được xem như đạt được và dừng training.

verbose : *bool, default=False*

In trạng thái của quá trình training ra màn hình

warm_start : *bool, default=False*

Khi set là True thì sẽ dùng lại kết quả của lần gọi trước.

momentum : *float, default=0.9*

Quán tính khi update gradient descent. Giá trị thuộc khoảng [0,1]. Chỉ dùng khi solver= ‘sgd’

nesterovs_momentum : *bool default=True*

Sử dụng Nesterov’s momentum hay không. Chỉ dùng khi momentum > 0 và solver= ‘sgd’

early_stopping : *bool, default=False*

Khi đặt là True, sẽ chia 10% bộ dữ liệu train ra để validate. Sẽ dừng train khi điểm số không cải thiện ít nhất tol trong n_iter_no_changed số epochs liên tiếp.

validation_fraction : *float, default=0.1*

Tỉ lệ dữ liệu được chọn làm việc xác nhận early_stopping. Chỉ dùng khi early_stopping=True. Giá trị nằm giữa 0 và 1.

beta_1 : *float, default=0.9*

Tỉ lệ giảm luỹ thừa cho vector moment thứ nhất trong thuật toán adam. Nằm trong [0,1). Chỉ dùng khi solver= ‘adam’

beta_2 : *float, default=0.999*

Tỉ lệ giảm luỹ thừa cho vector moment thứ hai trong thuật toán adam. Nằm trong [0,1). Chỉ dùng khi solver= ‘adam’

epsilon : *float, default=1e-8*

Giá trị chống chia cho 0. Chỉ dùng khi solver = ‘adam’

n_iter_no_change : *int, default=10*

Số lần lặp tối đa mà mô hình không cải thiện được ít nhất tol. Dùng khi solver= ‘adam’ hoặc ‘sgd’

7. Tài liệu tham khảo

- [1] Báo cáo cập nhật ngành điện tháng 10-2019 https://www.vietinbank.vn/investmentbanking/resources/reports/102019-CTS-BCnganhdien.pdf
- [2] Nguyen, Nam & Duong, Viet. (2019). A System for Monitoring the Electric Usage of Home Appliances using Machine Learning Algorithms. 158-164. 10.1109/ACOMP.2019.00032.
- [3] V. H. Duong and N. H. Nguyen, "AI System for Monitoring States and Power Consumption of Household Appliances," 2020 IEEE Eighth International Conference on Communications and Electronics (ICCE), 2021, pp. 527-532, doi: 10.1109/ICCE48956.2021.9352110.
- [4] https://machinelearningmastery.com/how-to-configure-the-number-of-layers-and-nodes-in-a-neural-network/
- [5] Khoá học Neural network and Deep Learning của thầy Andrew Ng trên Coursera https://www.coursera.org/learn/neural-networks-deep-learning
- [6] sklearn.neural_network.MLPClassifier https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html#sklearn.neural_network.MLPClassifier