Tài liệu giải thích

# Môi trường

Vì tính chất của nghiên cứu cần phân tích dữ liệu, vẽ biểu đồ, xem bảng biểu, … nhiều hơn là quản lý source code, nên em sử dụng Jupiter Notebook.

## Cách cài đặt

Cài đặt bộ [Anaconda](https://www.anaconda.com/products/individual) là sẽ đầy đủ môi trường, thư viện.

Bộ công cụ này bao gồm môi trường Python, phần mềm Jupiter Notebook và tất cả các thư viện thường được sử dụng trong giới AI-ML. (pandas, numpy, sklearn, …)

# Tập dữ liệu ban đầu

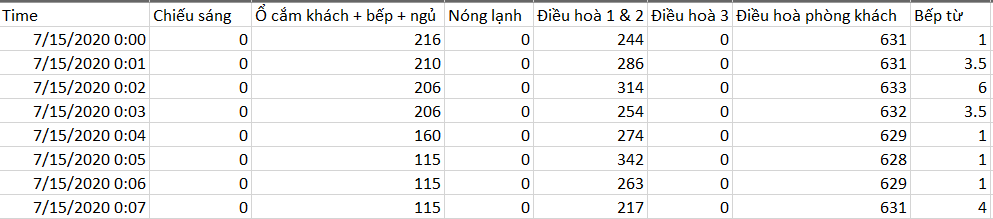
Tập dữ liệu bao gồm 4 file (trong thư mục ./data)

* Công suất tiêu thụ (W.csv)
* Điện áp tiêu thụ (V.csv)
* Dòng điện tiêu thụ (A.csv)
* Hệ số công suất (cosphi.csv)

Tất cả bao gồm 31680 bản ghi. Mỗi bản ghi là số liệu đo trong 1 phút, bắt đầu từ ngày 15/7/2020.

Thông tin ghi lại của 7 thiết bị:

* Chiếu sáng
* Ổ cắm khách + bếp + ngủ
* Nóng lạnh
* Điều hoà 1 & 2
* Điều hoà 3
* Điều hoà phòng khách
* Bếp từ

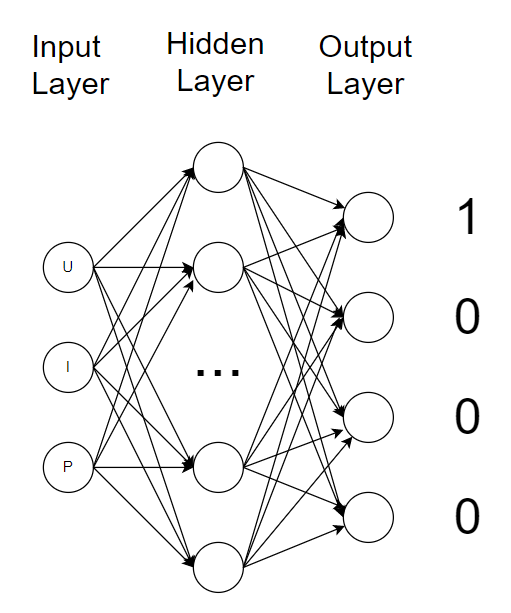


Một vài điểm dữ liệu của tập dữ liệu đo công suất tiêu thụ P. (U, I, cosphi tương tự)

# Mô hình làm theo bài báo 1

Bài báo 1: Design a System for Monitoring and Identifying Appliance and Home Activity using Machine Learning Algorithm

Ý tưởng: Sử dụng MLP với vector đầu vào x bao gồm [U, I, P] dự đoán xem những thiết bị nào đang bật/tắt.



Kiến trúc mạng gồm 3 lớp.

* Input layer gồm 3 nơ-ron tương ứng với 3 tham số đầu vào U, I, P
* Hidden layer gồm k nơ-ron (k sẽ được tìm sau một số lần chạy thực nghiệm, lấy giá trị k tối ưu. Với tập dữ liệu này thì k = 30 là tối ưu, k < 20 sẽ cho chất lượng mô hình tệ. k > 50 không có nhiều khác biệt)
* Output layer có số nơ-ron bằng với số trường hợp xảy ra. Ví dụ có 2 thiết bị thì có 4 trường hợp có thể xảy ra (như trên hình). Có n thiết bị thì có 2^n trường hợp có thể xảy ra.

Cụ thể từng bước triển khai đã được note trong file jupiter notebook: “Bai bao 1.ipynb”

# Mô hình làm theo bài báo 2

Bài báo 2: AI system for Monitoring states and Power consumption of HouseHold Applicances

Ý tưởng: Mỗi khi thiết bị bật hoặc tắt sẽ gây ra thay đổi và . Đo đạc tính toán hai thông số này để dự đoán xem thiết bị nào vừa mới bật hoặc tắt.

Kiến trúc mạng gồm 3 lớp:

* Input layer gồm 2 nơ ron tương ứng với và
* Hidden layer gồm k nơ ron (k cũng được xác định thông qua thực nghiệm)
* Output layer gồm 2 \* n nơ ron (n = số thiết bị). Mỗi thiết bị có 2 trạng thái bật/tắt, vì vậy cần one-hot vector có độ dài 2\*n để xác định thiết bị nào vừa bật/tắt. Ví dụ [1, 0, 0, 0] nghĩa là thiết bị 1 vừa mới bật. [0, 1, 0, 0] nghĩa là thiết bị 1 vừa mới tắt.

Cụ thể từng bước triển khai đã được note trong file “Bai bao2.ipynb

# Mô hình kết hợp

Ý tưởng: Mô hình giống với bài báo 1, chỉ khác là thêm 2 thông số và vào vector đầu vào.

Với mỗi điểm dữ liệu, tính toán U tổng, I tổng, P tổng, Q tổng, và.

Trong đó

= Pthời\_điểm\_t – Pthời\_điểm\_t-1,

# Giải thích tham số mô hình MLP

MLPClasifier của sklearn có các tham số sau:

**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: Ảnh có chứa văn bản

Mô tả được tạo tự động L2: Ảnh có chứa văn bản

Mô tả được tạo tự động

**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’

# Các bước cần thiết để xây dựng mạng nơ ron trong bài báo

## Phân tích dữ liệu

Dữ liệu như thế nào, tính chất ra sao => lựa chọn mô hình phù hợp

Mô hình đã chọn: MLP

## Lựa chọn kiến trúc mạng

Dựa vào data, suy ra được input layer và output layer có bao nhiêu unit.

Tuy nhiên để xác định số tầng và số unit của từng tầng hidden layer thì có một trong những cách sau:

* Dựa vào thực nghiệm
  + Nói chung không có công thức nào để xác định trực tiếp số layer hay số node trong mỗi layer sao cho tối ưu. Chỉ có thể dựa vào thực nghiệm
* Dựa vào trực giác
  + Nếu bài toán của chúng ta phức tạp thì có thể nó sẽ cần nhiều tầng.
* 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.

[[reference](https://machinelearningmastery.com/how-to-configure-the-number-of-layers-and-nodes-in-a-neural-network/)]

## Khởi tạo trọng số

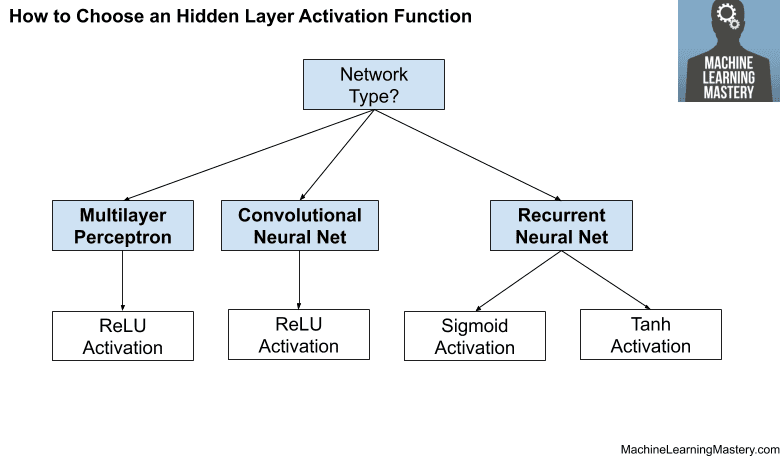
* Không được khởi tạo tất cả trọng số đều bằng 0, vì như vậy mạng nơ-ron sẽ không hoạt động.
* Không nên khởi tạo trọng số quá lớn, vì như vậy khi áp dụng vào activation function (sigmoid hay tanh thì sẽ cho ra kết quả rất gần biên, làm chậm quá trình học.
* Khởi tạo ngẫu nhiên
  + Ảnh có chứa văn bản

    Mô tả được tạo tự động
* Khởi tạo bias = 0 là tốt nhất.

## Chọn activation function

* ReLU
  + Hàm relu được sử dụng nhiều nhất vì độ đơn giản và hiệu quả của nó
  + Mô hình hội tụ nhanh hơn
  + F(x) = Max(0, x)
* Sigmoid
  + F(x) = 1 / (1 + e-x­)
  + Còn gọi là hàm logistic
  + Đưa đầu ra về khoảng 0, 1
* Tanh
  + Đưa đầu ra về khoảng -1, 1

Các hidden layer phải có cùng activation function



**Chọn activation function cho output layer?**

* Linear
* Logistic (sigmoid)
  + Đưa output về khoảng 0, 1
* Softmax
  + Tổng các output bằng 1

Cách chọn?

* Regression: One node, linear activation
* Binary classification: One node, sigmoid activation
* Multiclass Classification: One node per class, softmax activation
* Multilabel Classification: One node per class, sigmoid activation

## Xây dựng bộ giải (trình tối ưu hoá)

* 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

## Xác định hàm lỗi

* Mean Square Error
  + Ảnh có chứa văn bản

    Mô tả được tạo tự động
* Cross-Entropy Loss (or Log Loss)
  + Ảnh có chứa văn bản

    Mô tả được tạo tự động

## Lan truyền tiến

Tính giá trị cho các nút từ tầng input đến tầng output, dựa vào bộ trọng số và activation function.

## Lan truyền ngược

[[reference](https://www.coursera.org/lecture/neural-networks-deep-learning/backpropagation-intuition-optional-6dDj7)]

## Xây dựng hàm đánh giá

Tính toán độ chính xác của mô hình