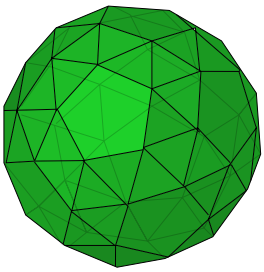
Projects in Mathematics and Applications

NEURAL NETWORK

Tuan Pham *, Long Nguyen †, Sang Ta ‡, Hoan Nguyen §

Ngày 27 tháng 8 năm 2018



* Trường Phổ Thông Năng Khiếu, TP.HCM
† Trường THPT Gia Định, TP.HCM
‡ Trường THPT Chuyên Hà Nội-Amsterdam
§ Trường THPT Chuyên Nguyễn Du, Đắk Lắk

Lời cảm ơn

Trước hết, chúng em xin cảm ơn ban tổ chức PiMA đã tạo ra một trại hè thật sự bổ ích cho các em học sinh trên khắp các tỉnh thành. Đây là một cơ hội rất tốt cho học sinh chúng em tiếp cận với những tri thức mới, hiểu biết hơn về machine learning, rèn luyện kĩ năng làm việc nhóm cũng như viết báo cáo. Đây là những kĩ năng rất cần thiết cho chúng em trong tương lai.

Bên cạnh đó, chũng em cũng chân thành cảm ơn các mentor là anh Trọng, chị Minh, anh Nhật cũng như anh Thế Anh đã hỗ trợ tụi em rất nhiều trong quá trình hoàn thiện dự án.

Sau cùng, chúng em muốn gửi lời cảm ơn đến các nhà tài trợ đã góp phần biến PiMA 2018 trở thành hiện thực.

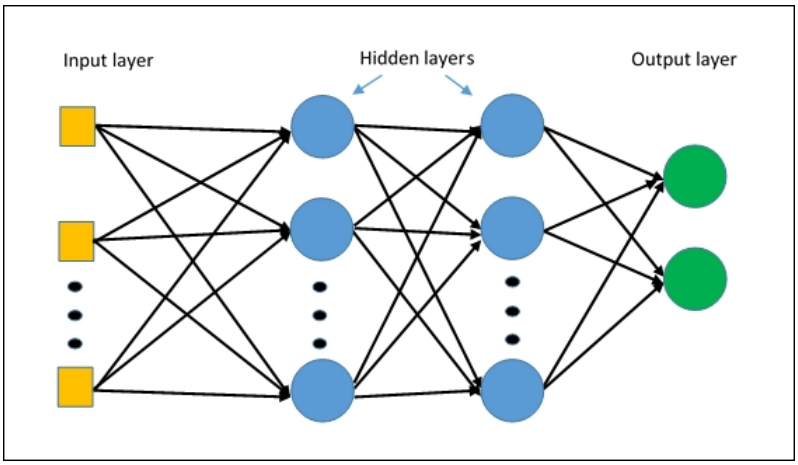
Tóm tắt nội dung

Trong bài báo cáo này, chúng tôi sẽ trình bày tổng quan về mạng neuron. Về lý thuyết, chúng ta sẽ nói về các các khái niệm sau đây. Thứ nhất, nền tảng xây dựng của mô hình mạng neuron – một perceptron unit, giới hạn của một unit và tiềm năng của một mạng neuron. Activation Function – các phép biến đổi chung để trả về một giá trị thuộc khoảng mong muốn. Loss Function – hàm mất mát thể hiện sự sai lệch của mô hình dự đoán với dữ liệu thực tế. Multilayer – mô hình gồm nhiều layer thể hiện sự đột phá của Machine Learning và khởi đầu của Deep Learning. Backpropogation – kĩ thuật tính gradient và bài toán tối ưu. Trong dự án của chúng tôi, chúng tôi sẽ dự đoán giá nhà dựa trên những đặc trưng chi tiết của dữ liệu có sẵn. Chúng tôi sẽ thiết kế một mạng neuron dự đoán về kết quả dựa trên những dữ liệu cho trước với sai số hợp lý.

1 Các ký hiệu

Trước khi đi vào chi tiết, xin được giới thiệu những kí hiệu sẽ dùng trong phần này:

1. Một Multi-layer Perceptron (MLP) bao gồm: input layers, output layers và có thêm nhiều hidden layers. (Hình minh hoạ)



Hình 1: Mô hình multilayer đơn giản

2. Một node trong một layer được gọi là một unit. Các unit được phân loại thành input unit, hidden unit, và output unit tương ứng với layer mà unit đó thuộc về.

3. Mỗi hidden layer nhận input là z và trả về output a (giá trị của hàm activation function với biến z. Đầu ra của unit thứ i trong layer thứ l được ký hiệu là $a_i^{(l)}$. Nếu gọi $d^{(l)}$ là số unit trong layer thứ l không tính bias thì vector output của layer thứ l là $\mathbf{a}^{(l)} \in \mathbb{R}^{d^{(l)}}$.

4. Ma trận trọng số (weights) ký hiệu là

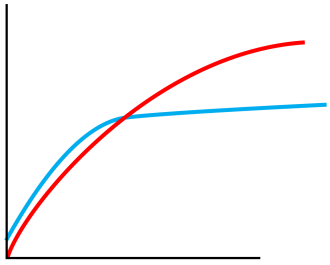
$$\mathbf{W}^{(l)} \in \mathbb{R}^{d^{(l-1)} \times d^{(l)}}, l = 1, 2, ..., L$$

trong đó $\mathbf{W^{(l)}}$ thể hiện các kết nối từ layer thứ $l-1$ tới layer thứ l (ta coi input layer là layer thứ 0). Cụ thể hơn, phần tử $w_{ij}^{(l)}$ thể hiện kết nối từ node thứ i của layer thứ $l-1$ tới nốt thứ j của layer thứ (l). Các biases của layer thứ l được kí hiệu là $\mathbf{b}^{(l)} \in \mathbb{R}^{d^{(l)}}$. Việc tối ưu một MLP cho một công việc nào đó đồng nghĩa với việc chúng ta cần đi tìm các weights và biases này.

Tập hợp các weights và biases lần lượt được ký hiệu là $\mathbf{W}$ và $\mathbf{b}$.

2 Giới thiệu tổng quan về Neural Network và Machine Learning

Deep Learning là một phân lớp nhỏ hơn của Machine Learning, sử dụng kĩ thuật mạng neuron. Cấu trúc của một mạng neuron được lấy ý tưởng từ cách hoạt động của mạng lưới neuron sinh học hoạt động trong não bộ: các neuron liên kết và truyền thông tin với nhau tạo thành một hệ thống – một hệ thần kinh. Neural network trong những năm hiện nay là một xu hướng và cũng là một đột phá trong các mô hình nghiên cứu về Machine Learning. Mạng neuron được xây dựng nên từ những perceptron unit lưu trữ và kế thừa thông tin từ những tập dữ liệu, bằng cách liên tục thử sai và tách các đặc trưng, mô hình có thể dự đoán được kết quả phù hợp của bài toán.

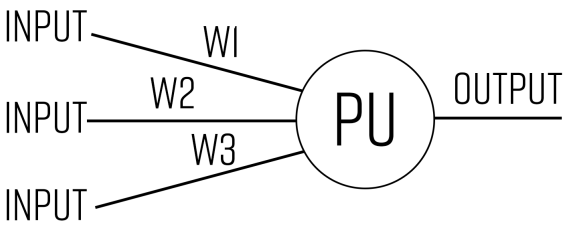


Hình 2: So sánh hiệu năng giữa hai thuật toán Machine Learning

Đường màu đỏ là hiệu năng của neural network. Đường màu xanh là hiệu năng của các thuật toán Machine Learning cổ điển.

3 Perceptron

Cấu trúc Neural Net Perceptron Units (PU) có cấu tạo tương tự với một neuron thần kinh trong não bộ.



Perceptron Unit hay còn gọi là một neuron là nền tảng để tạo nên một mô hình Neural network, mô hình được biểu diễn dưới dạng công thức:

$$\begin{aligned} y_i &= f_i(\sum_{j=1}^{n} w_{ij}x_j + b_i) \\ &= f_i(a_i) \end{aligned} \qquad (1)$$

Là một hàm trong một neuron trong mạng lưới. Được định nghĩa là tích của các input và bộ trọng số tại unit đấy và cộng thêm với ngưỡng (mức độ quan trọng của unit đấy - bias). Theo một cách toán học, mô hình chính được biểu diễn như (1). Trong đấy, y chính là đầu ra của hàm số, f là activation function (sẽ được đề cập ở mục sau), $\mathbf{w}$ là vector trọng số và $\mathbf{x}$ là đầu vào của bài toán, $\mathbf{b}$ chính là bias của mô hình.
Lưu ý rằng đưa về bài toán của NN, một mạng lưới các units theo từng layer thì đầu vào của nút ở layer hiện tại là đầu ra ở layer trước.
Trọng số trong mô hình trên chính là mức độ quan trọng, độ mạnh của mô hình thay đổi, cập nhật theo thời gian để khiến cho mô hình đạt được kết quả đúng nhất có thể.

4 Activation function - Hàm kích hoạt

4.1 Giới thiệu

Mô hình neural network có rất nhiều đặc trưng cơ bản, trong đó có Activation Function. Phần lớn các đơn vị trong neural network (mạng nơron) chuyển net input (mạng dữ liệu đầu vào) bằng cách sử dụng một hàm vô hướng (scalar-to-scarlar function) gọi là hàm kích hoạt, kết quả của hàm này là một giá trị gọi là mức độ kích hoạt của đơn vị (unit's activation). Activation function là một hàm kích hoạt xác định mức độ kích hoạt khác dựa trên mức độ kích hoạt hiện tại. Các hàm kích hoạt thường bị ép vào một khoảng giá trị xác định, do đó thường được gọi là các hàm bẹp (squashing).

4.2 Activation function

Mạng neural liên quan đến việc dạy một máy tính biết cách suy nghĩ và phân biệt dữ liệu. Việc dạy một mạng neural có 1 layer tương đương với việc giải ra ma trận tham số của nó. Hình dung bạn đang giải một hệ phương trình tuyến tính:

$$\mathbf{y} = \mathbf{W}^T\mathbf{x}$$

trong đó:
x ở đây là một vector input.
W là ma trận tham số, đại diện cho layer.
Để chuẩn hoá **y**, chúng ta cần một hàm f (gọi là activation function) khả vi, được dùng trong thuật toán dạy neuron. Mặt khác ta có công thức tính mỗi output của một unit:

$$a_i^{(l)} = f\left(\mathbf{w}_i^{(l)T}\mathbf{a}^{(l-1)} + \mathbf{b}_i^{(l)}\right)$$

Trong đó f là một activation function ở dạng phi tuyến (nonlinear). Công thức trên cũng được biểu diễn dưới dạng vector:

$$\mathbf{a}^{(l)} = f\left(\mathbf{W}^{(l)T}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}\right)$$

Một khái niệm có liên quan đến activation function là **element-wise**. Element-wise là việc áp dụng activation function f lên từng thành phần của một ma trận (vector). Các thành phần này sau đó sẽ được sắp xếp lại đúng theo thứ tự để được một ma trận có kích thước bằng với ma trận input.
Activation function là hàm nhận vector đầu vào, sau đó biến đổi để trả về vector đầu ra. Các hàm activation đơn giản gồm sgn, tanh, sigmoid, hay ReLU. Activation function cho phép ghi nhận được kết quả của dạng linear và nonlinear functions. (Hình minh họa)

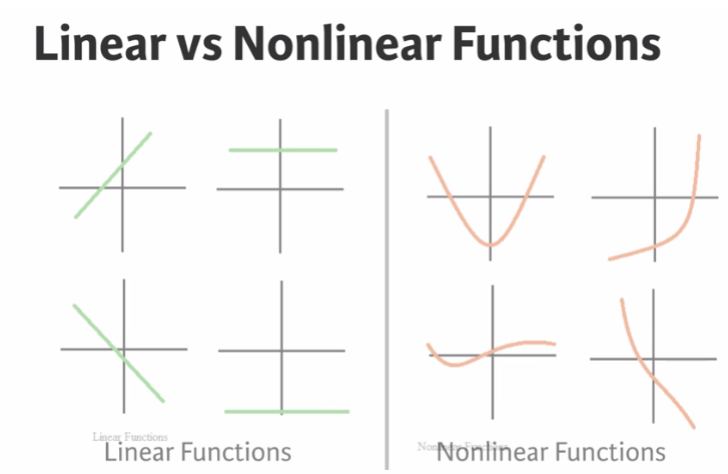
1. **Sign function (sgn)**

   Hàm sgn không được sử dụng trong Multi-layer Perceptron (MLP) mà chỉ được sử dụng trong Perceptron Learning Algorithm (PLA) vì nó có đạo hàm bằng không tại hầu hết các điểm và không có đạo hàm tại 0 khiến Gradient Descent không hoạt động.

2. **Sigmoid function**

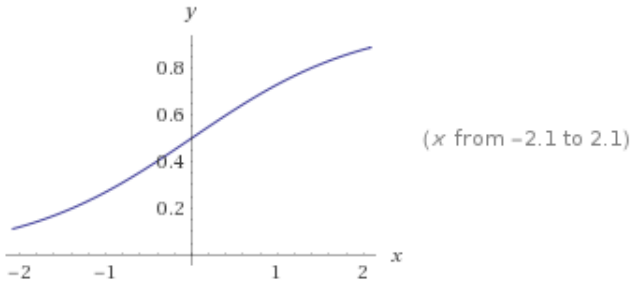
   Hàm sigmoid là một hàm toán học có đường cong hình chữ S hoặc đường cong sigmoid đặng trưng, có dạng:

$$S(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}$$



Hình 3: Hình dáng đồ thị của hàm tuyến tính và hàm phi tuyến

Đồ thị của hàm sigmoid như hình dưới đây: (Hình minh họa)



Hình 4: Hình dáng đồ thị hàm Sigmoid

Nhìn vào đồ thị ta thấy hàm số sigmoid là hàm đơn điệu, bị chặn và khả vi trên $(-1, 1)$

Với đầu vào nhỏ (rất âm), hàm số sẽ cho đầu ra gần với 0, tức là với $x \to -\infty$ thì giá trị hàm số tiến đến 0. Hàm số này được sử dụng nhiều trong quá khứ vì có đạo hàm rất đẹp. Gần đây, hàm số này ít khi được sử dụng vì có một nhược điểm cơ bản là khi đầu vào có trị tuyệt đối lớn (rất âm hoặc rất dương), gradient (đạo hàm) của hàm số này sẽ rất gần với 0 (nhược điểm này gọi là Sigmoid saturate and kill gradients), do đó dẫn đến việc các hệ số tương ứng với unit đang xét sẽ gần như không được cập nhật.

Ứng dụng của hàm Sigmoid:

Nhiều quá trình tự nhiên, chẳng hạn như các quy trình học tập phức tạp của hệ thống , thể hiện sự tiến triển từ những khởi đầu nhỏ giúp tăng tốc và tiến tới đỉnh điểm theo thời gian. Khi một mô hình toán học cụ thể bị thiếu, một hàm sigmoid thường được sử dụng.

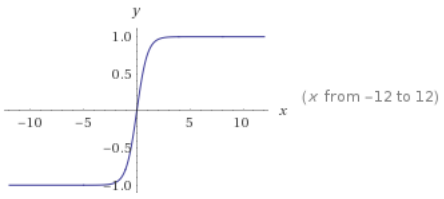
Mô hình van Genuchten-Gupta được dựa trên một đường cong S ngược và áp dụng cho phản ứng của năng suất cây trồng với độ mặn của đất. etc

3. **Tanh function**

Khác với hàm Sigmoid ít sử dụng thì **tanh** là một hàm tương tự thường được sử dụng và mang lại hiệu quả tốt hơn. Hàm tanh có bản chất là một hàm hyperbolic, có tính chất của một hàm lượng giác thông thường.

$$\tanh x = \frac{\sinh x}{\cosh x} = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1}$$

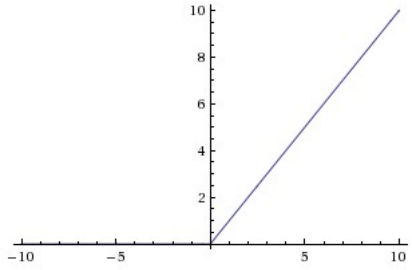
Đồ thị hàm tanh như sau



Hàm tanh cũng tương tự như hàm sigmoid đều có nhược điểm về việc gradient rất nhỏ với các đầu vào có trị tuyệt đối lớn.

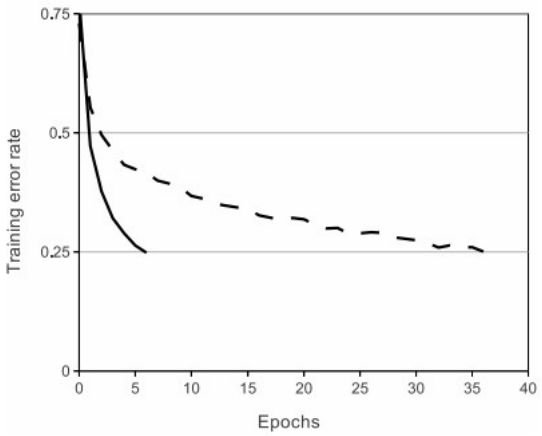
4. **ReLU**

ReLU (Rectified Linear Unit) được sử dụng rộng rãi gần đây vì tính đơn giản của nó. Công thức toán học của hàm ReLU: $f(s) = \max(0, s)$. Hàm ReLU có đồ thị như hình dưới đây:



Nhìn vào đồ thị ta thấy đạo hàm của nó bằng 0 tại các điểm âm, bằng 1 tại các điểm dương. Người ta chứng minh được ReLU giúp cho việc huấn luyện các multilayer neural network (MLP) và deep network (rất nhiều hidden layer) nhanh hơn rất nhiều so với hàm tanh.

Hình dưới đây so sánh sự hội tụ của hàm mất mát khi sử dụng hai hàm kích hoặc ReLU và tanh. Sự tăng tốc này được cho là vì ReLU được tính toán gần như tức thời và gradient của nó cũng được tính cực nhanh với gradient bằng 1 nếu đầu vào lớn hơn 0, bằng 0 nếu đầu vào nhỏ hơn 0.



Mặc dù hàm ReLU không có đạo hàm tại $s = 0$ trong thực nghiệm, người ta vẫn thường định nghĩa ReLU'$(0) = 0$ và khẳng định thêm rằng, xác suất để input của một unit bằng 0 là rất nhỏ.

Ngoài ra ReLU cũng có những biến thể khác như như leaky rectified linear unit (Leaky ReLU), parametric rectified linear unit (PReLU) và randomized leaky rectified linear units (RReLU). Thực tế là trước khi thiết kế ta không biết được hàm kích hoạt nào cho kết quả tốt nhất. Tuy vậy ta có thể bắt đầu bằng ReLU, nếu kết quả chưa khả quan thì ta thay thế bằng các biến thể của nó và so sánh để tìm ra kết quả tốt nhất.

5 Loss Function

5.1 Introduction

Sau đây, chúng ta sẽ tìm hiểu sơ khai về khái niệm Loss Function thông qua một ví dụ minh họa nho nhỏ.

Giả sử ta muốn phân loại (classify) một số đồ vật trong các bức ảnh. Giả sử tập hợp ảnh dùng cho máy học (training set) chỉ biểu thị 10 sự vật. Nói cách khác, mỗi ảnh trong tập này biểu thị 1 trong số 10 sự vật trên. Đối với con người chúng ta, việc này là quá đơn giản nhưng đối với máy tính thì chúng ta phải làm thế nào để nó có thể phân biệt được các sự vật trên. Và tất nhiên, khi đó ta phải dùng đến machine learning.

Như đã trình bày trong phần trên, trong machine learning, ta sẽ sử dụng tới một neural network. Trong mỗi neural network đó sẽ bao gồm các layer, mỗi layer gồm các node/unit. Mỗi một unit ở layer này sẽ được 'nối' với tất cả các unit của layer ngay trước đó. Những mối nối này chính là những trọng số ảnh đến đến activation(kết quả) của unit đó. $\mathbf{W}^T\mathbf{x}^{(l)} = \mathbf{x}^{(l+1)}$. Chỉ khi những trọng số này phù hợp thì mới cho ra được một kết quả mong muốn. Chính là ở bước này, khái niệm Loss Function được sinh ra để giải quyết vấn đề tìm bộ trọng số phù hợp.

Về cơ bản, ta có thể hiểu sơ qua rằng Loss Function là một hình phạt đối với máy tính mỗi khi dự đoán sai kết quả mong muốn. Cụ thể, nếu các activation (kết quả) của output layer càng 'sai lệch' so với kết quả mong muốn có sẵn (supervised learning) thì máy càng bị phạt nặng, tức là giá trị của Loss Function càng cao. Vậy bài toán sẽ trở thành cách để tối ưu (optimize) hàm Loss sao cho nhỏ nhất.

5.2 Loss Function tổng quát

Một loss function sẽ cho chúng ta biết được độ tốt hay tồi của model (neural network) của chúng ta.
Cho một tập hợp các cặp số: $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$ Trong đó: $\mathbf{x}_i$ là một vector chứa các features của sự vật (của training set). y_i là giá trị (có thể là một vector, hay một số thực,. . .) mong muốn của $\mathbf{x}_i$. Nói cách khác, sau khi cho $\mathbf{x}_i$ cho máy học thì ta mong muốn thu được kết quả y_i. N: là số phần tử của training set. Ta kí hiệu $\mathcal{L}$ là Loss Function và định nghĩa hàm Loss Function tổng quát như sau:

$$\mathcal{L} = \frac{1}{N}\sum_i \mathcal{L}_i(f(x_i, \mathbf{W}), y_i)$$

Trong đó

- $\mathbf{W}$: bộ các weights.
- y_i : kết quả của Loss Function đối với điểm dữ liệu $\mathbf{x_i}$.
- f : activation function (một phép biến đổi ouput của từng node thành một đại lượng tùy vào mục đích sử dụng).

Do đó, Loss Function chính là trung bình của kết quả các Loss Function đối với tất cả các điểm dữ liệu trong training set.

5.3 Tối ưu hóa hàm Loss

Trong việc tối ưu các hàm Loss, một kĩ thuật mà ta hay sử dụng đó chính là Gradient Descent. Đối với những bài toán với các biến gồm 1 hay 2 chiều thì việc giải cực trị có thể đạt qua các kĩ năng phân tích (analytical) là khả thi, nhưng đối với các bài toán gồm nhiều chiều (như trong machine learning) thì các kĩ năng này hầu như vô ích. Đó chính là lí do ta dùng Gradient Descent. Vậy Gradient Descent là gì?
Ta có công thức tổng quát cho những bước cập nhật bộ trọng số như sau:

$$\mathbf{w} \leftarrow \mathbf{w} - \mathbf{e}\mathbf{J}\nabla_{\mathbf{w}}\mathcal{L}$$

Trước hết, về mặt toán học, Gradient là một vector chứa các đạo hàm của một hàm theo từng biến (đạo hàm riêng theo từng chiều). Kí hiệu là ∇f. Về ý nghĩa hình học, Gradient sẽ cho ta biết hướng của 'di chuyển' sao cho giá trị hàm tăng nhanh nhất theo mọi chiều. Do đó, $-\nabla f$ sẽ cho ta biết hướng đi sao cho giá trị của hàm giảm mạnh nhất. Chính ý tưởng này giúp ta xây dựng thuật toán để cập nhật các bước đi cho các bộ trọng số sao cho hàm Loss sẽ dần dần tiến tới cực tiểu.

5.4 Một số loại Loss Function

Đối với mỗi một kĩ thuật hồi quy (Regression) thì ta sẽ phải xây dựng một Loss Function riêng. Nói cách khác, ta đi tìm dạng của $\mathcal{L}$.

Linear Regression

Một cách để minh hoa (visualize) kĩ thuật này chính là phân loại những sự vật thành 2 loại (2 classes) bằng đường thẳng (trong 2D) hoặc các siêu phẳng trong các chiều lớn hơn. Do đó, kĩ thuật này thuộc loại 'binary classification'.
Linear regression là một kĩ thuật phân loại (classification) tuyến tính. Có nghĩa là các activation của layer này sẽ là một tổ hợp tuyến tính của các activation của layer trước. Cụ thể $\mathbf{x}^{(\mathbf{l}+\mathbf{1})} = \mathbf{W}^T\mathbf{x}^{(\mathbf{l})},\ l = 1, 2..., N$. (Đã gặp ở phần trước).
- Ở trường hợp này, các kết quả y_i của activation function chính bằng luôn tổ hợp tuyến tính của các activation x_i của layer trước. Nói cách khác, activation function ở đây chính là $f(y_i) = y_i = \mathbf{w}^T\mathbf{x_i}$ (activation function trả kết quả là chính nó).

Xây dựng hàm loss

Nếu ta có bộ label $\mathbf{y}^* = [y_1^*, y_2^*, ..., y_n^*]$ (desired output) và các input $\mathbf{x} = [x_1, x_2, ..., x_k]$ thì thông qua linear regression, ta tạo output là bộ $\mathbf{y} = [y_1, y_2, ..., y_n]$.
Khi đó nếu ta lấy $\|\mathbf{y} - \mathbf{y}^*\|$ ta sẽ tính được sự chênh lệch so với y của model. Ta có thể coi đây là Loss function của hàm này nhưng thay vì đó, ta lại biến đổi biểu thức trên một chút để được Loss Function mới là:

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2N}\|\mathbf{y}^* - \mathbf{y}\|_2^2$$

Do đó Loss Function tổng quát chính là tổng của những Loss đối với tất cả các điểm dữ liệu:

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2N}\sum_{i=1}^{N}|y_i^* - y_i|_2^2$$
$$= \frac{1}{2N}\left\|\mathbf{y}^* - \mathbf{w}^\mathbf{T}\mathbf{x}\right\|_2^2$$

Ta sử dụng dạng này bởi vì:
i) Biểu thức trên dễ dàng thực hiện đạo hàm và từ đó dễ lập trình được gradient của Loss Function mà ta sẽ nhắc tới sau này.
ii) Hệ số $\frac{1}{2}$ không ảnh hưởng gì tới loss function bởi vì các error mà ta tìm được thông qua loss function thực chất chỉ là giảm đi một nửa.

Để tối ưu hàm này, ta cần tìm bộ weight ($\mathbf{w}$) sao cho Gradient của của Loss Function theo $\mathbf{w}$ là bằng 0. Cụ thể: Theo tính chất đạo hàm trong giải tích đối với ma trận (không cần chứng minh):

$$\nabla_\mathbf{w}\mathcal{L} = \frac{1}{N}\mathbf{X}\left(\mathbf{X}^T\mathbf{w} - \mathbf{y}\right)$$

Gradient bằng không khi $\mathbf{X}\mathbf{X}^T\mathbf{w} = \mathbf{X}\mathbf{y}$.
Nếu ma trận $\mathbf{X}\mathbf{X}^T$ khả nghịch thì ta luôn tìm được kết quả của bộ weight, đó là $\mathbf{w} = \left(\mathbf{X}\mathbf{X}^T\right)^{-1}\mathbf{X}\mathbf{y}$.

Trong trường hợp $\mathbf{X}\mathbf{X}^T$ không khả nghịch, ta sẽ cần sử dụng tới khái niệm về ma trận giả nghịch đảo. Do đó, ta sẽ không đề cập tới trường hợp này ở đây.

Logistic regression

a) Tổng quan
Đôi khi trong data set của ta xuất hiện những outlier, khiến cho kĩ thuật Linear bị nhiễu, ảnh hưởng tới độ chính xác của model. Khi đó ta có một cách khắc phục là Logistic Regression.
Giống với Linear Regression, Logistic Regression là một kĩ thuật thuộc lớp các bài toán về binary classification.
Khác với Linear Regression, Logistic Regression có activation function như sau:

$$f(s) = \frac{1}{1+e^{-s}}$$

Trong đó $s = \mathbf{w}^T\mathbf{x}$. Hàm có dạng trên được gọi là Logistic/Sigmoid.
Nhận xét:

$$\lim_{s\to-\infty} f(s) = 0; \lim_{s\to+\infty} f(s) = 0$$

Do đó các output của hàm Sigmoid bị chặn trong khoảng $(0, 1)$. Điều này rất có lợi bởi hàm đã biến những activation của layer trước, thông qua Sigmoid function, thành những output chỉ nằm ở khoảng $(0, 1)$. Do đó, một cách để tưởng tượng: output của hàm này cho biết độ 'liên quan' hay xác xuất của một example trong training set. Đồng thời, nó cũng cho biết độ lớn hay chiều hướng của mối quan hệ giữa các input và output.
Vậy thì từ hàm trên, ta sẽ xây dựng Loss Function ra sao?

b) Xây dựng hàm Loss

Trong binary classification ta sẽ có 2 class mà ta cần phân biệt. Gọi y_i là output của điểm dữ liệu x_i thì ta giả sử khi $y_i = 1$ tượng trưng cho class thứ nhất còn khi $y_i = 0$ tượng trưng cho class thứ hai. Khi đó ta định nghĩa xác xuất để một example $\mathbf{x}$ nằm vào class 1 là $f(\mathbf{x}, \mathbf{w}^T)$. Do đó, xác xuất nằm vào class 2 sẽ là $1 - f(\mathbf{x}, \mathbf{w}^T)$. (Do chỉ có 2 class.) Cụ thể:

$$\begin{aligned} P(y_i = 1|\mathbf{x_i}; \mathbf{w}) &= f(\mathbf{w}^T\mathbf{x_i}) \\ P(y_i = 1|\mathbf{x_i}; \mathbf{w}) &= 1 - f(\mathbf{w}^T\mathbf{x_i}) \end{aligned}$$

Kí hiệu $z_i = f(\mathbf{w}^T\mathbf{x}_i)$ và viết gộp lại hai biểu thức trên ta có:

$$P(y_i|\mathbf{x_i}; \mathbf{w}) = z_i^{y_i}(1 - z_i)^{1-y_i}$$

Có thể thấy biểu thức trên chính là công thức phân phối nhị thức đối với khi $n = 1$ (tức là khi chỉ có hai biến cố). Xét toàn bộ training set với $\mathbf{X} = [\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_N}] \in \mathbb{R}^{d\times N}$ và $\mathbf{y} = [\mathbf{y_1}, \mathbf{y_2}, ..., \mathbf{y_N}]$, chúng ta cần tìm $\mathbf{w}$ để biểu thức sau đây đạt giá trị lớn nhất

$$P(\mathbf{y}|\mathbf{X}; \mathbf{w})$$

Khi đó do tính độc lập của các điểm dữ liệu (example) trong training set, ta tính được $P(\mathbf{y}|\mathbf{X}; \mathbf{w})$ theo quy tắc nhân như sau:

$$\begin{aligned} P(\mathbf{y}|\mathbf{X}; \mathbf{w}) &= \prod_{i=1}^{N} P(y_i|\mathbf{x_i}; \mathbf{w}) \\ &= \prod_{i=1}^{N} z_i^{y_i}(1 - z_i)^{1-y_i} \end{aligned}$$

Do ta cần xác xuất là sát với 1 nhất (tức là sát với sự chắc chắn) nên ta cần phải tìm $\mathbf{w}$ sao cho hàm P trên là lớn nhất. Tuy nhiên, việc này không hề đơn giản bởi vì thứ nhất, việc tối ưu của hàm tích rất khó (khó dùng trong gradient descent) và bởi vì nếu N quá lớn thì P sẽ cho kết quả là một số rất nhỏ dẫn đến sai số trong tính toán.
Cách khắc phục chính là lấy logarith tự nhiên của P. Cụ thể: Ta định nghĩa Loss Function luôn như sau:

$$\begin{aligned} J(\mathbf{w}) &= -\log P(\mathbf{y}|\mathbf{X}; \mathbf{w}) \\ &= -\sum_{i=1}^{N} (y_i \log z_i + (1 - y_i) \log(1 - z_i)) \end{aligned}$$

Để P lớn nhất thì J cần nhỏ nhất.

c) Tối ưu hàm Loss

Ta sẽ tối ưu hàm Loss bằng phương pháp gradient descent. Cụ thể ta sẽ đi tìm gradient của $\mathcal{L}$ theo từng trọng số (weight). w ở đây là bộ trọng số nối với một node.

$$\frac{\partial J}{\partial w_j} = \sum_{i=1}^{N} \frac{\partial J}{\partial z_i} \cdot \frac{\partial z_i}{\partial s_i} \cdot \frac{\partial s_i}{\partial w_j}$$

Ta sẽ tính biểu thức trên bằng cách tính từng đạo hàm riêng.

$$\frac{\partial J}{\partial z_i} = y_i \cdot \frac{1}{z_i} + (1 - y_i) \cdot \frac{1}{1 - z_i}$$

Vì chỉ có biểu thức đối với z_i bị đạo hàm.

$$z_i = f(s_i) = \frac{1}{1+\exp(-s_i)}$$

$$\frac{\partial z_i}{\partial s_i} = \frac{-\exp(-s_i)\cdot(-1)}{(1+\exp(-s_i))^2} = \frac{1}{1+\exp(-s_i)} \cdot \frac{\exp(-s_i)}{1+\exp(-s_i)} = z_i(1-z_i)$$

Vì

$$s_i = \mathbf{w}^T\mathbf{x_i}$$

$$\rightarrow \frac{\partial s_i}{\partial w_j} = x_i$$

Do đó

$$\begin{aligned}
\frac{\partial J}{\partial w_j} &= \sum_{i=1}^{N} \frac{y_i}{z_i} z_i(1-z_i)x_i - \frac{1-y_i}{1-z_i} z_i(1-z_i)x_i \\
&= -\sum_{n=1}^{N} y_i(1-z_i)x_i - (1-y_i)z_i x_i \\
&= -\sum_{n=1}^{N} x_i\left[y_i - y_i z_i - z_i + y_i z_i\right] \\
&= -\sum_{n=1}^{N} (y_i - z_i)x_i
\end{aligned}$$

Suy ra cách cập nhật của ta là:

$$\mathbf{w} = \mathbf{w} - \eta\nabla_{\mathbf{w}} J$$

5.4.1 Softmax Regression

a) Tổng quan

Khi giải các bài toán với nhiều class hơn thì ta không thể sử dụng được các kĩ thuật Binary Classification. Lúc đó, một kĩ thuật phổ biến đó chính là Softmax Regression.

Ta sẽ trình bày ý tưởng cơ bản của Softmax Regression thông qua ví dụ ở phần đầu của mục Loss Function này. Đánh dấu mỗi class trong 10 class trên lần lượt là $[C_1, C_2, C_3, ..., C_{10}]$. Xét một bộ output gồm 10 node $[y_1, y_2, y_3, ..., y_{10}]$, mỗi node sẽ là một kết quả biểu thị xác suất mà máy đoán tương ứng cho mỗi class. Giả sử ta mong muốn máy đoán được đúng một bức ảnh thuộc class C_k. Nếu máy đoán chuẩn 100% thì $y_k = 1$ và $y_k = 0$ với $i = 1, ..., k-1, k+1, ...n$.

Trên thực tế, máy tính không thể học được hoàn toàn chính xác như vậy và output sẽ là một bộ 10 xác suất của 10 class. Do những class này độc lập với nhau nên tổng của 10 xác suất phải bằng 1. Giá trị xác suất tương ứng với class nào càng lớn và gần 1 thì máy tính đoán ảnh đó thuộc vào class đó. Ví dụ với trường hợp trên, nếu $y_1 = 0.9$ thì khả năng cao là máy tính đã đoán chính xác.

Từ những suy luận trên, ta sẽ tìm một activation function để có thể tính được xác suất của từng class. Thông qua đó ta sẽ so sánh với những xác suất mon muốn và tìm ra được hàm Loss.

Ví dụ: Giả sử xác suất output là $y = [y_1, y_2, y_3, ..., y_{10}]$ với $y_1 + y_2 + ... + y_{10} = 1$ và $y^* = [1, 0, ..., 0]$ thì sai số sẽ là $\mathbf{y}^* - \mathbf{y}$. Tuy nhiên, đây vẫn chưa phải là Loss Function và ta sẽ

tìm được dưới đây.

b) Xây dựng hàm Loss

Ta sẽ định nghĩa activation funcion (a) như sau:

$$y_i = \frac{\exp(z_i)}{\sum_{j=1}^{C} \exp(z_j)}, \ \forall i = 1, 2, ..., C$$

Ý tưởng: mỗi một unit trong output layer, thông qua activation function, sẽ cho ra một xác suất. Khi sử dụng mô hình Softmax Regression, với mỗi đầu vào $\mathbf{x}$, ta sẽ có đầu ra dự đoán là $y = \text{softmax}(\mathbf{W}^T\mathbf{x})$. Trong đó đầu ra mong muốn là vector y^* được biểu diễn dưới dạng vector đơn vị.

Theo đây, ta xây dựng hàm Loss dựa trên sự sai lệch giữa $\mathbf{y}$ và $\mathbf{y}^*$.

Ta có thể nghĩ ngay tới hàm Loss giống với trường hợp trong Linear Regression, đó là

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2N} \|\mathbf{y} - \mathbf{y}^*\|_2^2$$

Tuy nhiên hàm này không hiệu quả đối với những bài toán về khoảng cách giữa các phân phối xác suất. Khi đó ta giới thiệu một hàm mới có tên là Cross Entropy:

$$H(\mathbf{p}, \mathbf{q}) = -\sum_{i=1}^{C} p_i \log q_i$$

Ở ví dụ của ta, hàm kia trở thành:

$$H(\mathbf{y}^*, \mathbf{y}) = -\sum_{i=1}^{C} y_i^* \log y_i$$

Vì H là một hàm số nhiều biến với một ràng buộc rằng $g(\mathbf{y}) = y_1 + y_2 + ... + y_n = 1$ với n là số class nên ta sẽ dùng tới phương pháp nhân tử Lagrange để tính giá trị nhỏ nhất của H.

Xét bài toán sau:

Cho $\mathbf{y}^* \in \mathbb{R}_+^n$ là một vector với các thành phần dương và tổng bằng 1. Bài toán tối ưu

$$\mathbf{y} = \text{argmin}_{\mathbf{y}} H(\mathbf{y}^*, \mathbf{y})$$

thỏa mãn

$$\sum_{i=1}^{n} y_i = 1; y_i > 0$$

có nghiệm $\mathbf{y} = \mathbf{y}^*$

Giải:

$$\mathcal{L}(y_1, y_2, ..., y_n, \lambda) = -\sum_{i=1}^{n} y_i^* \log(y_i) + \lambda(\sum_{i=1}^{n} y_i - 1)$$

Ta sẽ tính sao cho gradient của $\mathcal{L}$ bằng 0.00000000000

$$\nabla_{q_1,...,q_C,\lambda} \mathcal{L}(q_1, ..., q_C, \lambda) = 0 \Leftrightarrow \begin{cases} -\frac{p_i}{q_i} + \lambda = 0, i = 1, ..., C (1) \\ q_1 + q_2 + ... + q_C = 1 (2) \end{cases}$$

(1) đúng bởi vì khi ta đạo hàm cho y_i thì mọi biểu thức của y_j với j khác i đều tiêu giảm.

Từ phương trình thứ nhất ta có $p_i = \lambda q_i$. Vì vậy, $1 = \sum_{i=1}^{C} p_i = \lambda \sum_{i=1}^{C} q_i = \lambda \Rightarrow \lambda = 1$. Điều này tương đương với $q_i = p_i,\ \forall i = 1, ..., C$.
Chính vì vậy, thông qua cross-entropy, ta sẽ xây dựng được hàm loss function như sau:

$$\begin{aligned}\mathcal{L}(\mathbf{W}; \mathbf{X}, \mathbf{Y}) &= -\sum_{i=1}^{N}\sum_{j=1}^{C} y_{ji} \log(a_{ji}) \\ &= -\sum_{i=1}^{N}\sum_{j=1}^{C} y_{ji} \log \frac{\exp(\mathbf{w}_j^T \mathbf{x}_i)}{\sum_{k=1}^{C}(\mathbf{w}_k^T \mathbf{x}_i)}\end{aligned}$$

Trong đó C là số các class, còn N là số các điểm dữ liệu (example).
Ý tưởng: Cộng tổng các hàm Cross-Entropy của tất cả các điểm dữ liệu x trong training set. Ta sẽ tối ưu hàm trên như sau: Trước hết ta sẽ tính gradient tại một điểm dữ liệu, ví dụ là $x^{(i)}$. Ta biến đổi lại hàm Loss đối với điểm dữ liệu $x^{(i)}$ là:

$$\begin{aligned}J_i(\mathrm{W}) &= -\textstyle\sum_{j=1}^{C} y_{ji}^* \log(y_{(ji)}) \\ &= -\textstyle\sum_{j=1}^{C} y_{ji}^* \log\left(\frac{exp(\mathbf{w}_j^T \mathbf{x}_i)}{\sum_{k=1}^{C} \exp(\mathbf{w}_k^T \mathbf{x}_i)}\right) \\ &= -\textstyle\sum_{j=1}^{C} \left(y_{ji}^* \mathbf{w}_j^T \mathbf{x}_i - y_{ji}^* \log\left(\sum_{k=1}^{C} \exp(\mathbf{w}_k^T \mathbf{x}_i)\right)\right) \\ &= -\sum_{j=1}^{C} y_{ji} \mathbf{w}_j^T \mathbf{x}_i + \log\left(\sum_{i=1}^{C} \exp(\mathbf{w}_k^T \mathbf{x}_i)\right)\end{aligned}$$

Khi đó, gradient của nó là một vector chứa đạo hàm riêng theo các bộ weight nối với từng output node . Kí hiệu bộ weight này là $\mathrm{w_i}$ với $i = \overline{1, N}$, trong đó N là số class. Ta có:

$$\begin{aligned}\nabla_{\mathbf{w}_j} J_i(\mathbf{W}) &= -y_{ji}^* \mathrm{x}_i + \frac{\exp(\mathbf{w}_j^T \mathrm{x}_i)}{\sum_{k=1}^{C} \exp(\mathbf{w}_k^T \mathrm{x}_i)} \mathrm{x}_i \\ &= -y_{ji}^* \mathrm{x}_i + y_{ji} \mathrm{x}_i \\ &= \mathrm{x}_i (y_{ji} - y_{ji}^*) \\ &= e_{ji} \mathrm{x}_i \qquad\qquad (\text{do } e_{ji} = y_{ji} - y_{ji}^*)\end{aligned}$$

Do đó gradient đối với điểm dữ liệu $\mathrm{x}^{(\mathrm{i})}$ đối với tất cả các bộ weight sẽ là tổng của những Gradient như bên trên. Khi đó ta được:
$\nabla_{\mathbf{w}} J_i(\mathbf{W}) = \mathrm{x}_i [e_1, e_2, ..., e_{C_i}] = \mathrm{x}_i e_i^T$
Khi đó lặp lại các thao tác trên đối với tất cả các điểm dữ liệu, ta được gradient tổng quát sau:
$\nabla_{\mathbf{w}} J(\mathbf{W}) = \frac{1}{N} \sum_{i=1}^{N} \mathrm{x}_i \mathrm{e}_i^T = \frac{1}{N} \mathrm{X} \mathbf{E}^T$
Khi đó cách update các weight được thực hiện như sau:
$\mathbf{W} \leftarrow \mathbf{W} - \frac{\eta}{N} \mathbf{X} \mathbf{E}^T$
Với n là learning rate.

6 Backpropagation - Multilayers neural network

6.1 Multilayers neural network

Layer

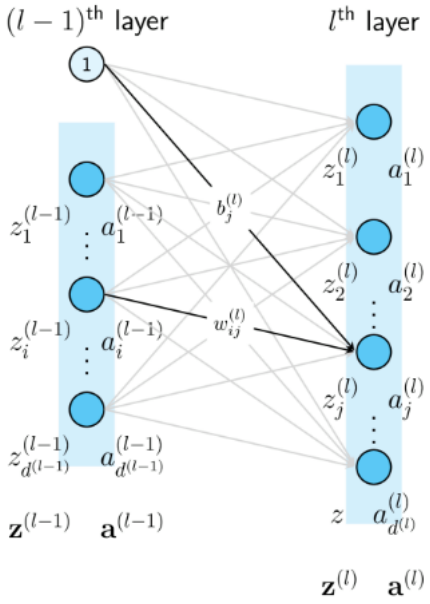
Ngoài input layer và output layer, mô hình còn có thể có nhiều hidden layers ở giữa. Người ta thường sử dụng chữ L để kí hiệu cho số các layers, được tính là số các hidden layer + 1 (output layer). Vì vậy, người ta thường gọi layer 1, layer 2,...cho đến layer cuối cùng là layer L.

Unit

Mỗi node hình tròn trong các layer thường được gọi là các unit. Các node trong input hay output layer được gọi là input unit và output unit.

6.2 Backpropagation

Với mục tiêu tối thiểu hàm Loss Function và đưa ra một mô hình tốt nhất, chúng ta thường sử dụng công cụ Gradient Descent. Một trong những thuật toán hiệu quả nhất để áp dụng Gradient Descent chính là Backpropagation.
Mục đích của Backpropagation là tính các đạo hàm $\frac{\partial \mathcal{L}}{\partial w}$ và $\frac{\partial \mathcal{L}}{\partial b}$ trên mọi weight và bias trong network.
Nhắc lại về mô hình multilayers network:



$$z_j^{(l)} = \sum_k w_{jk}^{(l)} \cdot a_k^{l-1} + b_j^l \quad (1)$$
$$a_{ij} = f^l(z_j^l) \quad (2)$$

Trong đó

$w_{jk}^{(l)}$: là weight của neuron thứ k trong layer (l-1) đặt vào neuron thứ j trong layer (l)

f^l : là activation function của layer thứ l

Các phương trình quan trọng của backpropagation

Để tính các $\frac{\partial \mathcal{L}}{\partial w}$ và $\frac{\partial \mathcal{L}}{\partial b}$ tại mọi điểm, ta cần đặt $\delta_j^l = \frac{\partial \mathcal{L}}{\partial z_j^l}$ được gọi là error của node thứ j trong layer l. Khi đó

$$\begin{aligned}\delta_j^L &= \frac{\partial \mathcal{L}}{\partial z_j^L} \\ &= \frac{\partial \mathcal{L}}{\partial a_j} \cdot f(z_j^l)' \quad \text{(dùng quy tắc đạo hàm của hàm hợp cho phương trình (2))}\end{aligned}$$

Khi đó, tổng quát hóa dưới dạng ma trận, ta có:

$$\delta^L = \nabla_a \mathcal{L} \bigodot f(\mathbf{z^L})' \quad (3.1)$$

Trong đó:

$\nabla_a \mathcal{L}$: là vector mà các phần tử là các đạo hàm riêng $\frac{\partial \mathcal{L}}{\partial a_j^L}$

$\bigodot$: là tích Hadamard (element-wise): $[s \bigodot t]_j = s_j \cdot t_j$

Để tính error trên node j của layer l:

$$\begin{aligned}\delta_j^l &= \frac{\partial \mathcal{L}}{\partial z_j^l} \\ &= \frac{\partial \mathcal{L}}{\partial a_j^{(l)}} \cdot \frac{\partial a_j^{(l)}}{\partial z_j^{(l)}} && \text{(Dùng quy tắc mắt xích cho phương trình (2))} \\ &= \left(\sum_k \frac{\partial \mathcal{L}}{\partial z_k^{(l+1)}} \frac{\partial z_k^{(l+1)}}{\partial a_j^{(l)}} \right) \cdot f'\left(z_j^l\right) && \text{(Các } z_k^{l+1} \text{ là các unit được nối với } a_j) \\ &= \left(\sum_k \delta_k^{l+1} w_{jk} \right) \cdot f'\left(z_j^l\right) \\ &= \left(\delta^{l+1} w_k^T \right) \cdot f'\left(z_j^l\right)\end{aligned}$$

Tổng quát hóa dưới dạng ma trận, ta có:

$$\delta^l = \delta^{l+1} w^T \bigodot f(z_j^l)' \quad (3.2)$$

Kết hợp (3.1) và (3.2), ta có thể dễ dàng tính toán tất cả các $\frac{\partial \mathcal{L}}{\partial \delta_l}$ trong toàn bộ mô hình

Để tính các $\frac{\partial \mathcal{L}}{\partial b}$ trong mô hình, ta có công thức

$$\frac{\partial \mathcal{L}}{\partial b_j^l} = \delta_j^l \quad (3.3)$$

Để tính các $\frac{\partial \mathcal{L}}{\partial w}$ trong mô hình, ta có công thức

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial w_{jk}^l} &= \frac{\partial \mathcal{L}}{\partial z_j^l} \cdot \frac{\partial z_j^l}{\partial w_{jk}^l} \quad \text{(Sử dụng quy tắc mắt xích cho phương trình (1))} \\ &= \frac{\partial \mathcal{L}}{\partial z_j^l} \cdot a_k^{l-1} \\ &= \delta_j^l \cdot a_k^{l-1} \qquad (3.4)\end{aligned}$$

TỔNG KẾT: BỐN PHƯƠNG TRÌNH CỦA BACKPROPAGATION LÀ:

$$\begin{aligned}
\delta_j^L &= \nabla_a \mathcal{L} \odot f'(z^L) \\
\delta^l &= \delta^{l+1} w^{(l+1)T} \odot f'(z_j^l) \\
\frac{\partial \mathcal{L}}{\partial b_j^l} &= \delta_j^l \\
\frac{\partial \mathcal{L}}{\partial w_j^l} &= \delta_j^l \cdot a_k^{l-1}
\end{aligned}$$

Ta thấy ở về phải của phương trình (3.1) các hệ số đều rất dễ tính. Vì vậy, ta sẽ tính được δ_j^L. Sau đó ta sẽ dùng phương trình (3.2) để tính ngược lên các δ_j^l với l= L-1,L-2,...,2. Các phương trình (3.3) và (3.4) cho ta hướng cập nhật các chỉ số weight và bias của toàn bộ mô hình của chúng ta.

Thuật toán backpropagation

1. Input: Nhập các dữ liệu input, tạo các hàm activation function,...

2. Feed forward: Với l=2,3,...L tính

$$\mathbf{z^l} = \mathbf{w^l} \cdot \mathbf{a^{l-1}} + \mathbf{b_l}$$

và

$$\mathbf{a_l} = f(\mathbf{z_j^l})$$

3. Output error δ^L: Tính

$$\delta_{\mathbf{j}^L} = \nabla_a \mathcal{L} \bigodot f'(\mathbf{z^L})$$

4. Backpropagate the error: kéo các error về tất cả các weight và bias

$$\delta^l = \delta^{l+1} (w^{(l+1)})^T \bigodot f'(z_j^l)$$

5. Output Gradient descent của hàm Loss được cho bởi

$$\frac{\partial \mathcal{L}}{\partial b_j^l} = \delta_j^l$$

và

$$\frac{\partial \mathcal{L}}{\partial w_j^l} = \delta_j^l \cdot a_k^{l-1}$$

6.2.1 Ý nghĩa của backpropagation

Backpropagation cho ta cách tính các $\frac{\partial \mathcal{L}}{\partial w}$ rất nhanh chóng bằng cách kéo hàm error $\frac{\partial \mathcal{L}}{\partial \delta}$ ngược về đằng trước. Vì vậy, mô hình gần như tính toán toàn bộ các gradient descent bằng một lần forward và một lần backward. Vì vậy, có thể nói rằng, backpropagation là một thuật toán rất hiệu quả cho công việc tính gradient descent của chúng ta, từ đó giúp chúng ta điều chỉnh các weight và bias của mô hình một cách thuận tiện và nhanh chóng.

7 Overfitting

Khi có data set, ta sẽ phân data set làm 2 phần (một cách ngẫu nhiên) gồm training set và test set. Training set là lượng dữ liệu để máy học và đưa ra một mô hình tốt nhất, sau đó chúng ta sẽ kiểm chứng chúng bằng test set trước khi đưa vào thực tiễn. Nhưng trong lúc cố gắng tìm mô hình tốt nhất cho bài toán, máy có thể bị mắc một lỗi là vô tình quan trọng một tính chất riêng nào đó của bộ data set. Nói cách khác, máy đã quá quan trọng việc xấp xỉ dữ liệu mà quên đi việc quan trọng hơn chính là tính tổng quát của mô hình, khiến cho mô hình không biểu diễn tốt các data bên ngoài. Đây là lỗi overfitting, một lỗi hay gặp trong neural network. Overfitting thường xảy ra trong các mô hình có lượng data ít (khi đó các data không

đủ tính tổng quát), hoặc mạng noron quá phức tạp (khi đó ta đã quá chú trọng việc xấp xỉ dữ liệu trong training set)

Một trong những đại lượng để tính toán độ overfitting
Training error: là sai số trên tập training set, được tính bằng công thức

$$\frac{1}{N_{\text{training set}}} \cdot L_{\text{training set}}$$

Test error: là sai số trên tập test set, được tính bằng công thức

$$\frac{1}{N_{\text{test set}}} \cdot L_{\text{test set}}$$

Lưu ý, cả 2 công thức đều cần nhân với đại lượng $\frac{1}{N}$ để tránh trường hợp so sánh khập khiễng vì có thể test set nhiều hơn, hay training set nhiều hơn. Nói cách khác, ta biết được một mô

hình bị overfit khi training error rất thấp mà test error cao. Một mô hình chỉ được coi là tốt nếu như training error và test error đều thấp.

Để tránh hiện tượng overfitting, người ta thường sử dụng 2 kĩ thuật là validation và regularization.

7.1 Validation

Ý tưởng của validation là sẽ trích từ tập training set ra một lượng data để riêng gọi là validiation set. Khi đó, bộ training data mới là bộ training data đã bỏ đi validation set. Lúc này bộ validation set sẽ đóng vai trò như bộ test trong mô hình của chúng ta. Chúng ta sẽ cố gắng tìm một mô hình cho cả training error và validation error nhỏ, khi đó, mô hình đó rất có thể làm cho test error nhỏ.

Dừng sớm

Dựa trên ý tưởng của validation, khi ta cố gắng làm cho training error nhỏ lại, thì validation error cũng sẽ nhỏ lại. Nhưng đến một lúc, khi ta đã quá chú trọng vào training set thì validation error rất có thể sẽ tăng lên. Khi đó, ta dùng ngay tại thời điểm mà validation error tăng lên. (insert hình)

7.2 Regularization

Ý tưởng chính của Regularization là chấp nhận hy sinh một chút độ chính xác của mô hình, nhưng giảm độ phức tạp của nó giúp tránh overfitting. Một số kỹ thuật thường dùng trong Regularization là L2 và L1 regularization. Ý nghĩa của cả 2 phương pháp là làm cho mô hình có khuynh hướng chọn các weight có giá trị nhỏ trong khi vẫn cố làm cho hàm Loss đạt cực tiểu.

7.2.1 L2 regularization

Thêm một đại lượng vào hàm Loss function (đại lượng nào được gọi là regularization term)

$$\mathcal{L} = \mathcal{L}_0 + \frac{\lambda}{2n} \cdot \sum_{\mathbf{w}} \|w_i\|^2$$

Trong phương trình trên, đại lượng được thêm vào là tổng bình phương của tất cả cái weights trong mạng neural của mình rồi nhân cho hệ số $\frac{\lambda}{2n}$. Trong đó λ được gọi là regularization parameters. $\mathcal{L}_0$ là hàm loss ban đầu, chưa dùng regularization
Khi đó

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial w_i} &= \frac{\partial \mathcal{L}_0}{\partial w_i} + \frac{\lambda}{n} \cdot w_i \\ \frac{\partial \mathcal{L}}{\partial b_j} &= \frac{\partial \mathcal{L}_0}{\partial b_j} \end{aligned}$$

Trong đó, $\frac{\partial \mathcal{L}_0}{\partial w_i}$ và $\frac{\partial \mathcal{L}_0}{\partial b_j}$ có thể tính được thông qua backpropagation, nên $\frac{\partial \mathcal{L}}{\partial w_i}$ và $\frac{\partial \mathcal{L}}{\partial b_j}$ đều có thể tính được
Theo đó, bước cập nhật của gradient descent sẽ là:

$$\begin{aligned} b_j &\leftarrow b_j - \eta \cdot \frac{\partial \mathcal{L}_0}{\partial b_j} \\ w_i &\leftarrow w_i - \eta \cdot \frac{\partial L_0}{\partial w_i} - \frac{\eta \cdot \lambda}{n} \\ &= w_i \cdot (1 - \frac{\eta \cdot \lambda}{n}) - \eta \cdot \frac{\partial L_0}{\partial w_i} \end{aligned}$$

Đại lượng $(1 - \frac{\eta \cdot \lambda}{n})$ có khuynh hướng làm cho các giá trị weight giảm xuống nên còn được gọi là weight decay. Hơn nữa, ta thấy, b sẽ không thay đổi so với mô hình không được regularization.

Ý nghĩa của L2 regularization: L2 regularization có ý nghĩa làm cho mô hình vừa có khuynh hướng chọn các weight có giá trị nhỏ vừa làm cho mô hình được hàm loss tối thiểu nhất có thể. Cụ thể hơn, L2 regularization sẽ giảm các weight không có ảnh hưởng quan trọng đến độ chính xác của mô hình, trong khi vẫn cố giữ nguyên các weight thiết yếu với mô hình

7.2.2 L1 regularization

Mô hình L1 regularization về cơ bản cũng giống như mô hình L2 regularization. Tuy nhiên, đại lượng thêm vào là $\frac{\lambda}{n} \cdot \sum_{\mathbf{w}} \|w_i\|$. Khi đó, ta có

$$\mathcal{L} = \mathcal{L}_0 + \frac{\lambda}{n} \cdot \sum_{w} \|w_i\|$$

Giống như L2 regulation, hàm loss sẽ phạt các weight lớn, làm cho mô hình sẽ có khuynh hướng chọn các weight có giá trị tuyệt đối nhỏ. Khi đó, ta tính đạo hàm của hàm loss theo biến weight.

$$\frac{\mathcal{L}}{w_i} = \frac{\mathcal{L}_0}{w_i} + \frac{\lambda}{n} \cdot \text{sgn}(w)$$

Trong đó hàm $\text{sgn}(x)$ sẽ bằng $+1$ nếu $w_i > 0$ và -1 nếu $w_i < 0$.
Bước cập nhật tiếp theo của chúng ta:

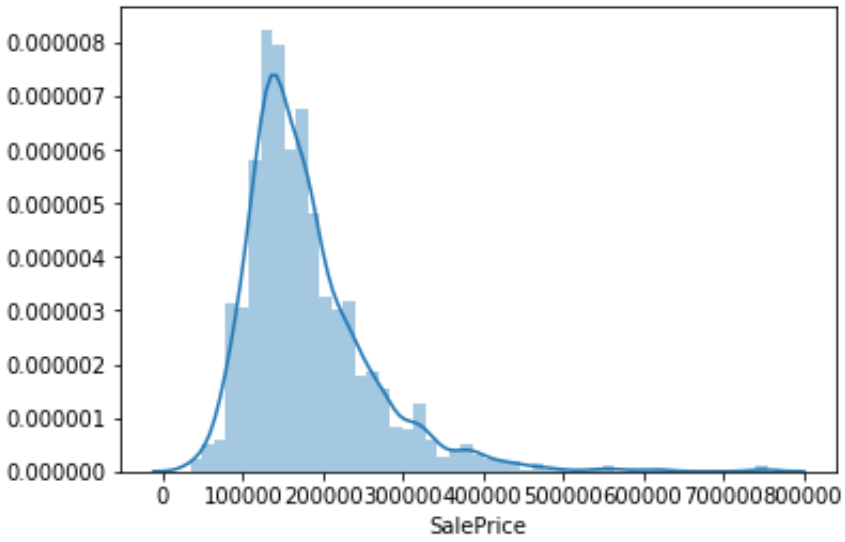
$$w_i \leftarrow w_i - \frac{\eta \cdot \lambda}{n} \cdot \text{sgn}(w_i) - \eta \cdot \frac{\partial \mathcal{L}_0}{\partial w_i}$$

Ý nghĩa của L1 regularization
So sánh với cách cập nhật của kỹ thuật L2 regularization ta thấy cả 2 kỹ thuật đều làm cho mô hình có khuynh hướng chọn các giá trị weight nhỏ hơn. Nhưng với L1, weights giảm đi một lượng là hằng số, còn với L2, weight giảm đi một lượng phụ thuộc vào **w**. Vì vậy, khi ||**w**|| lớn thì L2 giảm nhanh hơn L1, còn khi ||**w**|| nhỏ thì L1 giảm nhanh hơn L2. Khác với L2 một chút, L1 sẽ cố gắng giữ các weight có ảnh hưởng quan trọng đến độ chính xác của mô hình, đồng thời ép các weight còn lại rất gần 0.
Do hàm $\text{sgn}(x)$ có đạo hàm không xác định tại $x = 0$. Nhưng ta có thể khắc phục nhược điểm này bằng cách cho $\text{sgn}(0) = 0$ và không tính đạo hàm tại điểm đó (Vì khi weight đã bằng 0 rồi thì ta không thể giảm weight được nữa)

Ngoài ra, còn có một vài phương pháp khác giúp tránh overfitting như dropout,...

8 Bài toán và Phân tích data

Với những kiến thức và lí thuyết học được ở những phần trên, chúng tôi xây dựng một model để dự đoán giá nhà dựa trên những đặc trưng của ngôi nhà. Dataset mà ta sử dụng ở đây (nguồn: www.Kaggle.com) gồm có 80 đặc trưng. Tuy nhiên, ta chỉ giữ lại những đặc trưng biểu diễn dưới dạng số (continuous feature) bởi vì những đặc trưng phân loại (Categorical feature) được biểu diễn dưới dạng chữ. Trong bộ dataset của ta có 42 đặc trưng phân loại nên ta sẽ chỉ quan tâm tới 80-42=38 đặc trưng còn lại.
Trong bài toán đối với dự đoán giá nhà, trước hết chúng ta sẽ lập một biểu đồ để xem xét phân phối các giá nhà. Dựa vào biểu đồ dưới, ta thấy phân phối trông giống với một phân phối chuẩn (Guassian Distribution). Phần lớn các ngôi nhà đều có giá trong khoảng từ 10000-25000, và sau đó là một phần ít các nhà giá cao. Điều này được thể hiện rõ qua việc biểu đồ xuất hiện một cái đuôi về phía tay phải; còn những ngôi nhà giá rẻ thì lại tập trung về phía tay trái.

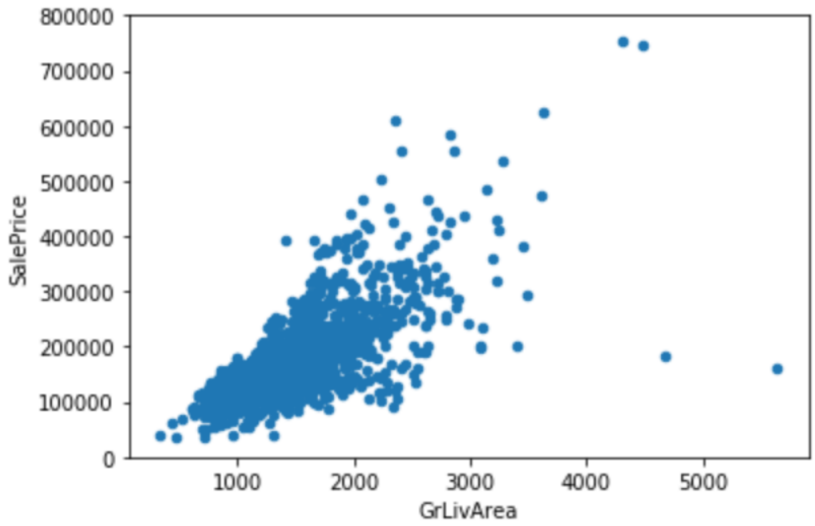


8.1 Phân tích rời rạc với một số đặc trưng

Tiếp theo, để tìm hiểu vì sao phân phối giá nhà lại có hình dạng như vậy, ta sẽ tìm hiểu mối liên hệ của giá nhà với một số đặc trưng khác (feature).

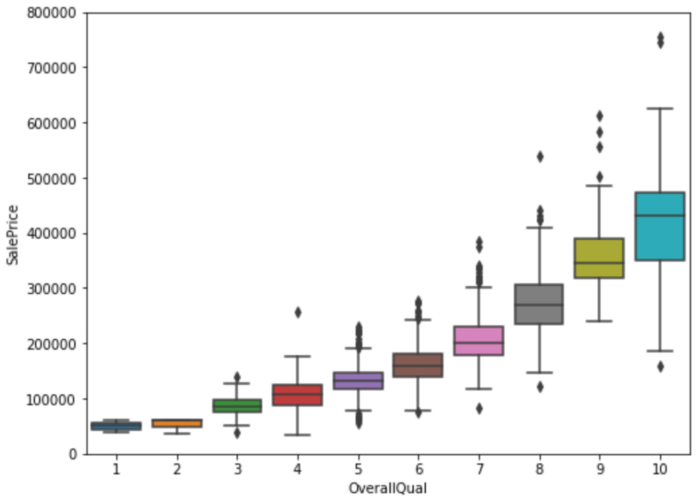
1. Đặc trưng được biểu hiện bằng số

   Giả sử biểu đồ (scatterplot) giữa giá nhà và diện tích sống sau đây:



Dựa trên biểu đồ, giá nhà và diện tích có một mối tương quan tuyến tính với nhau. Điều này hoàn toàn hợp lí bởi vì diện tích nhà càng rộng thì khả năng cao là giá nhà cũng sẽ cao.

2. Đặc trưng đặc trưng phân loại Ví dụ như mối quan hệ giữa giá nhà và chất lượng của nhà(được đánh giá theo một thang từ 1-10). Hãy xem biểu đồ hộp (box-plot) ở phía dưới:

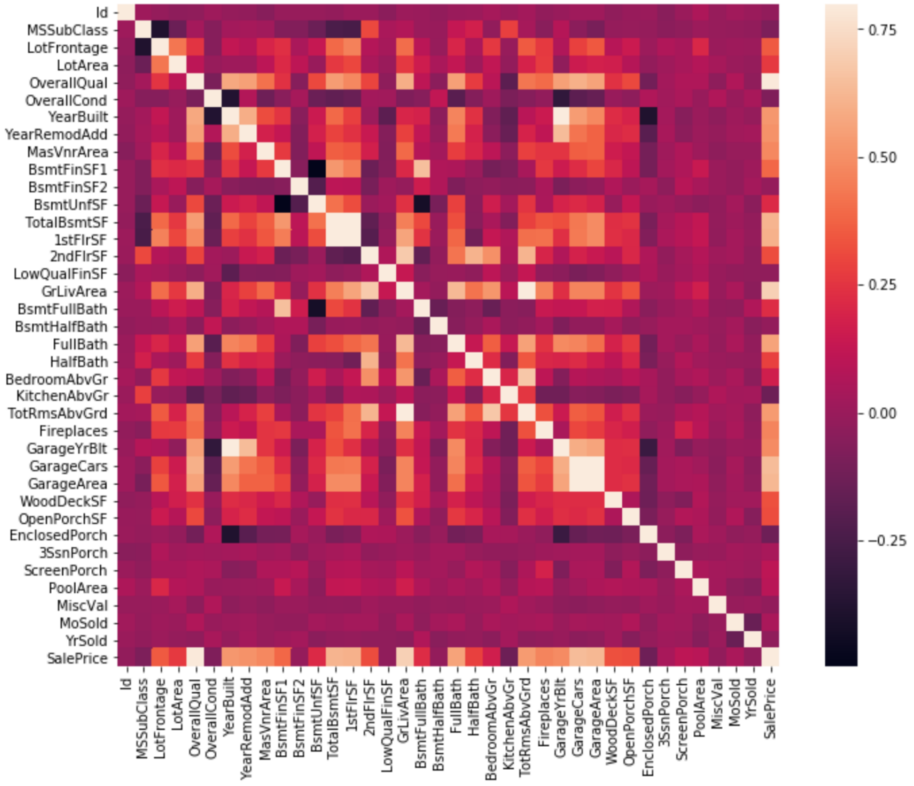


Dựa trên biểu đồ ta thấy giá nhà và chất lượng có mối tương quan đúng như dự đoán: chất lượng càng cao, nhà càng đắt. Không những vậy, ta có thể thấy giá nhà lớn nhất, nhỏ nhất, trung vị của từng chất lượng nhà cũng có mối quan hệ như vậy. Nếu để ý kĩ, ta còn thấy giá nhà tang theo lũy thừa nhưng không mạnh (not dramatic).

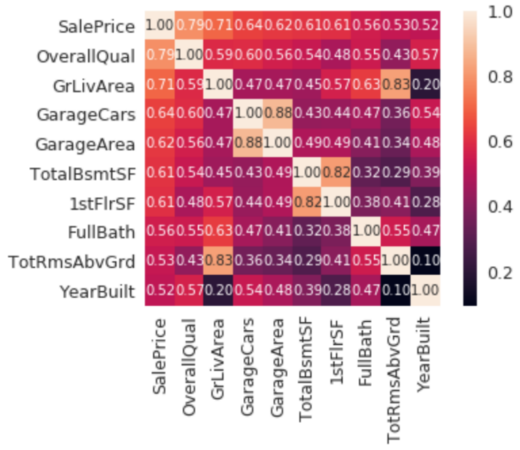
8.2 Biểu đồ mối liên hệ độ nóng (Correlation Heatmap)

Trong phần phân tích vừa rồi, ta mới chỉ, dựa theo kinh nghiệm, xét những đặc trưng mà ta nghĩ là có ảnh hưởng tới giá nhà. Đây là một cách phân tích rất chủ quan và ta

cần có một cách phân tích khách quan hơn cho những mối tương quan. Ta sẽ sử dụng tới Correlation Heatmap. Dưới đây là một heatmap thể hiện mỗi tương quan đôi một giữa những đặc trưng thể hiện được bằng số (continuous features) cũng như mối tương quna của từng đặc trưng này với giá nhà (Sale Price).



Ý tưởng cơ bản của biểu đồ này là ô vuông càng sáng thì mối tương quan giữa những feature là tọa độ của ô vuông càng mạnh. Dựa vào trực quan này, ta phát hiện thấy một số vùng tối rộng(ví dụ như vùng phía dưới hay bên tay phải của biểu đồ). Điều này có nghĩa là những đặc trưng thể hiện vùng tối này không ảnh hưởng gì nhiều tới những đặc trưng khác. Tất nhiên, một đặc trưng thì hoàn toàn tương quan (completely correlate) với chính nó nên là ta có thể thấy đường chéo của biểu đồ là rất sáng. Tiếp đó, ta sẽ loại bớt đi những đặc trưng không quan trọng này (insignificant feature) và vẽ lại một biểu đồ heatmap khác.



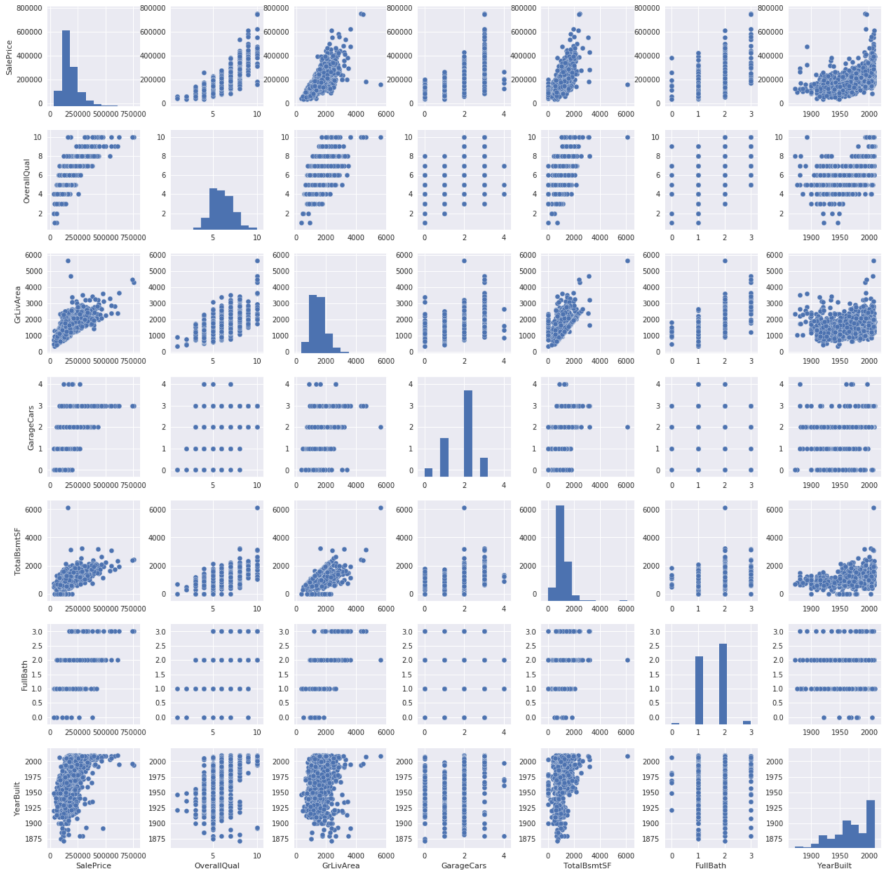
Ta đã sắp xếp những cột và hàng dựa trên độ tương quan với giá nhà (Sales Price) đối với những đặc trưng sau quá trình ''lọc'' đặc trưng trên.

Ta có một vài nhận xét sau:

- Đúng như ở phần trên, mối tương quan giữa giá nhà và diện tích (GrLivArea) hay chất lượng (OverallQual) là cao (sát với 1).

- Một số đặc trưng có mối quan hệ với giá nhà gần bằng nhau và mối tương quan như với nhau rất cao. Nhìn kĩ vào dữ liệu thì ta có thể thấy hiển nhiên. Ví dụ như đặc trưng diện tích nhà và diện tích sàn tầng 1. Những đặc trưng này có thể coi như gần giống nhau (cùng thể loại với nhau) và ta có thể lọc tiếp để tiếp tục phân tích dữ liệu.

Sau quá trình lọc dữ liệu, ta còn lại 7 đặc trưng đáng quan tâm nhất. Do số đặc trưng là ít, ta vẽ tất cả các biểu đồ (scatterplot) giữa đôi một những đặc trưng.



Thông qua những biểu đồ này, ta nhận ra một số mối tương quan thú vị:

- Một số những điểm dữ liệu trong biểu đồ mối quan hệ giữa Diện tích tầng hầm ('TotalBsmtSF') và diện tích nhà trên mặt đất (GrLiveArea) tạo thành những đường thẳng (tưởng tượng như những đường biên giới). Điều này được giải thích rằng diện tích hầm sẽ thường không vượt quá diện tích của nhà.

- Một phát hiện thú vị khác dựa vào biểu đồ ở góc dưới cùng phía tay trái. Ta có thể thấy rằng nhà càng mới thì giá càng cao. Ngoài ra, hình dạng tương đối của biểu đồ gần với một hàm lũy thừa nhẹ (hãy vận dụng trí tưởng tượng ở đây). Do đó, ta rút ra rằng giá nhà càng ngày càng có xu hướng tăng nhanh.

9 Tiền xử lý dữ liệu

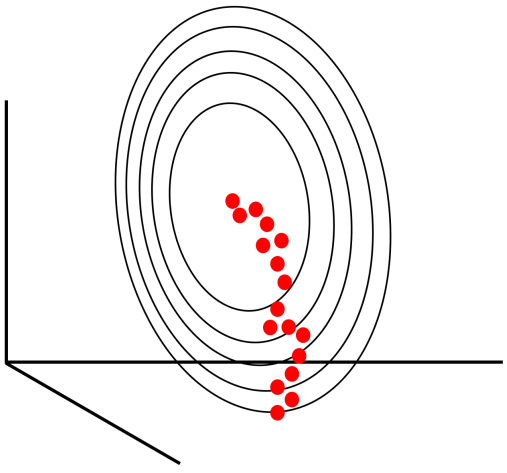
9.1 Outliers

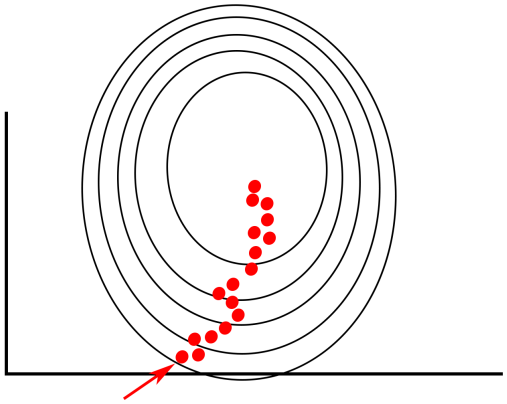
Một vấn đề ta thấy trong biểu đồ hộp (box plot) đó là xuất hiện một số điểm dữ liệu trông có vẻ như vượt quá mức khoảng 'bình thường' của những điểm dữ liệu khác. Đó có thể là những điểm quá lớn hoặc quá nhỏ hơn so với những điểm dữ liệu khác. Những điểm dữ liệu này khi cho máy học sẽ có khả năng gây nhiễu mô hình khiến những dự đoán của máy tính không được chính xác. Do đó chúng ta cần lọc để loại bỏ những điểm dữ liệu này.

9.2 Feature Scaling

Trước khi chúng ta tìm hiểu về vấn đề, ta cần hiểu một ý tưởng đó là khi các feature có quy mô (scale) gần gần bằng nhau thì thuật toán Gradient Descent sẽ diễn ra nhanh hơn. Nói cách khác, những bước đi của Gradient sẽ được thực hiện một cách 'trực tiếp' hơn về phía cực tiểu. Thật vậy, ta sẽ thấy rõ hơn nhận xét này thông qua một ví dụ đơn giản. Lấy ví dụ trong một bài toán dự đoán giá nhà, ta có 2 đặc trưng (ta lấy 2 đặc trưng để đơn giản hóa bài toán) để xử lí:
x_1 ($0 < x_1 < 2000$, đơn vị m^2)
x_2 ($0 < x_2 < 5$, đơn vị: *phong*)

Từ 2 đặc trưng trên, rõ ràng là quy mô của x_1 lớn hơn x_2 rất nhiều(vì $2000 >> 5$). Khi trình bày hình dạng của đường định mức (contour line) của các đặc trưng trên, ta sẽ thấy rằng các đường mức tạo thành những đường *elipse* mảnh và dài.(Xem dưới)





Điều này sẽ ảnh hưởng tới hiệu quả của Gradient Descent. Nhớ lại rằng Gradient (tam giác) sẽ cho ta đường đi ngắn nhất để đi tới cực đại. Nghĩa là đi ngược lại với Gradient sẽ cho ta đường đi ngắn nhất tới cực tiểu. Trong trường hợp với những đường định mức (contour lines) này, (âm tam giác) sẽ vuông góc với những đường mức và chỉ về hướng đi ngắn nhất. Do hình dạng của contour lines này, ta thấy rõ rằng Gradient mất rất nhiều bước để đạt tới cực tiểu, và do đó, mất nhiều thời gian xử lí. Cách để khắc phục là đưa quy mô của các đặc trưng gần bằng nhau. Ví dụ, trong trường hợp này, ta sẽ biến đổi các feature của nhà như sau:
$x_1' = \frac{x_1}{2000}$ với $0 < x_1' < 1$
$x_2' = \frac{x_2}{5}$ với $0 < x_2' < 1$

Nói cách khác, ta biến đổi giá trị của mỗi feature bằng cách chia giá trị đó cho khoảng (range). Về mặt dữ liệu, ta không hề thay đổi bản chất của dữ liệu mà thực chất chỉ biểu thị dữ liệu đó dưới một tỉ lệ (scale) mới. Do đó, ta có tên gọi kĩ thuật này là *Feature Scaling*. Sau thay đổi về đặc trưng này, ta thấy rằng các đường định mức tạo thành những đường elipse 'tròn' hơn và khi đó ta thấy (âm tam giác) sẽ 'hướng' trực tiếp hơn đến tới cực tiểu. Do đó, số bước cập nhật để tiến tới cực tiểu sẽ được rút ngắn.
Ngoài cách biến đổi đặc trưng như trên ta còn một cách nữa: chuẩn hóa trung bình (mean normalization). Cụ thể là như sau:

$$x_i^* = \frac{(x_i - \mu)}{\text{range}}$$

Trong đó
μ : trung bình các giá trị của mỗi feature của tất cả các điểm dữ liệu.
range : hiệu của giá trị lớn nhất và giá trị nhỏ nhất của mỗi feature.
Đặc điểm của cách này là đưa trung bình của tất cả giá trị của một feature về gần với 0. Tùy vào mục đích thì cách này sẽ hiệu quả hơn so với cách trên.

10 Thiết kế mô hình- Kết luận

Mô hình 1 : Bài toán của chúng tôi được tiếp cận bằng một mạng neuron gồm có 3 lớp (layer). Trong đó, có 1 lớp input, 1 lớp ẩn (hidden layer), và 1 lớp output. Lớp input gồm có 36 node, lớp ẩn gồm cũng gồm có 36 node, còn kết quả quả chúng ta là một output. Bằng activation reLu đơn giản nhưng đem lại kết quả khá tốt. Kết hợp với kĩ thuật Kfold dùng để chia data thành các tập validation ngẫu nhiên cùng các bước tiền xử lý, chúng tôi đã tính được RMSE trên validation với kết quả xấp xỉ 0.1 và kết quả trên test data là 0.3, chứng tỏ mô hình đủ hiệu quả với bài toán trên.

```
def init_model():

  # create model
  model = Sequential()
  model.add(Dense(36,input_dim=36,kernel_initializer='normal',activation='relu'))
  model.add(Dense(1, kernel_initializer='normal'))

   # Compile model
  model.compile(loss='mean_squared_error', optimizer='adam')
  return model

   estimator = KerasRegressor(build_fn=baseline_model,nb_epoch=100,batch_size=5,
       verbose=0)

   estimator.fit(train[col_train_bis], train[label])
```

Mô hình 2 : Bằng cách sử dụng nhiều hidden layers hơn (36,12,6) khiến model sâu hơn sử dụng các activation trên mô hình lần lượt là (reLu,sigmoid, tanh) kết hợp với việc chuẩn hóa dữ liệu (scalling input). Kết quả RMSE trên validation là 0.0047 test data xấp xỉ 0.17, chứng tỏ hiệu quả của mô hình tăng lên đáng kể.

```
def init_model():

  # create model
  model = Sequential()
  model.add(Dense(36,input_dim=36,kernel_initializer='normal',activation='relu'))
  model.add(Dense(12,kernel_initializer='normal',activation='sigmoid'))
  model.add(Dense(6,kernel_initializer='normal',activation='tanh'))
  model.add(Dense(1, kernel_initializer='normal'))

   # Compile model
  model.compile(loss='mean_squared_error', optimizer='adam')
  return model

   estimators = []
   estimators.append(('standardize', StandardScaler()))
   estimators.append(('mlp', KerasRegressor(build_fn=init_model, epochs=200,
       batch_size=5, verbose=0)))
   pipeline = Pipeline(estimators)

   pipeline.fit(train[col_train_bis], train[label])\\
```

11 Phụ lục

Đây là đoạn code Python mà team em đã sử dụng để mô phỏng
https://github.com/TrongDuongik/PiMA-team-7-NN

12 Hướng phát triển trong tương lai

Bằng cách tối ưu mô hình như tăng số layer để khả năng mô hình càng thêm mạnh mẽ hay kết hợp việc sử dụng regularization để giảm độ quá nhạy không cần thiết của mô hình. Từ đó số chiều đầu vào (số feature của input) có thể mở rộng mà mô hình mạng neuron của chúng ta vẫn có thể xử lí. Cụ thể, tùy vào số chiều của đầu vào, mô hình có thể tự điều chỉnh để đáp ứng và tạo ra kết quả vừa chính xác, vừa không bị ảnh hưởng bởi hiện tượng overfit.

Tài liệu

[1] Vũ Hữu Tiệp. *Machine Learning cơ bản*.

[2] Michael Nielsen. *Neural Networks and Deep Learning*. url: http://neuralnetworksanddeeplearnin
com/.

[3] Dean De Cock. *House Prices: Advanced Regression Techniques*. url: https://www.
kaggle.com/c/house-prices-advanced-regression-techniques.

[4] Pedro Marcelino. *Comprehensive data exploration with Python*. url: https://www.
kaggle.com/pmarcelino/comprehensive-data-exploration-with-python.