Multilayer Neural Network và Backpropagation

1. Giới thiệu

Multilayer Neural Network

Một neural network với nhiều hơn 2 layers còn được gọi là multilayer neural network, multilayer perceptrons (MLPs), deep feedforward network hoặc feedforward neural network. Feedforward được hiểu là dữ liệu đi thẳng từ đầu vào tới đầu ra và không quay lại ở điểm nào.

Một multilayer network là một neural network có nhiều layer, làm nhiệm vụ xấp xỉ mối quan hệ giữa các cặp (x, y) trong tập huấn luyện bằng hàm số có dạng:

$$y \approx g^{(L)}(g^{(L-1)}(...(g^{(2)}(g^{(1)}(x))))$$

Trong đó, layer thứ nhất đóng vai trò như hàm a1 = g1(x); layer thứ 2 đóng vai trò như hàm a2 = g2(g1(x)) = g2(a1) = f2(a1), … Tổng quát lên , ta được:

$g^{(l)}(a^{(l-1)}) = f^{(l)}(W^{(l)T}a^{(l-1)} + b^{(l)})$ *với W, b là ma trận trọng số và vector bias, f(x) là hàm kích hoạt (activation function)*

Trong phần này, ta sẽ tách riêng phần bias và ma trận hệ số, do đó vector input x là vector không mở rộng.

Đầu ra dự đoán của một network (feedforward):

$a^{(0)} = x$
$z^{(l)} = W^{(l)T}a^{(l-1)} + b^{(l)}, l = 1, 2, …, L$
$a^{(l)} = f^{(l)}(z^{(l)}), l = 1, 2, …, L$
$y^{\wedge} = a^{(L)}$

Bước này gọi là feedfoeward vì cách tính được thực hiện từ đầu đến cuối của network. Hàm mất mát thỏa mãn đạt giá trị nhỏ nhất khi đầu ra này gần với đầu ra thực sự. Tùy vào bài toán mà ta có hàm mất mát phù hợp.

Layer

Ngoài input layer và output layer, 1 multilayer neural network có thể có một hoặc nhiều hidden layer ở giữa, được đánh số 1, 2, …, L-1 (L là số lượng layer trong 1 multilayer network, được tính bằng tổng số hidden layer + 1).

Unit

Mỗi node trong một layer được gọi là một unit. Unit ở input layer, hidden layer, output layer lần lượt được gọi là input unit, hidden unit, output unit. Đầu vào của một unit là z, đầu ra í hiệu là a (a = f(x) với f là activation function)

Weights và Biases

Có L ma trận trọng số cho 1 multilayer neural network có L layer. Các ma trận được kí hiệu $W^{(l)}$ (l = 1, 2, …, L) thể hiện kết nối từ layer thứ l – 1 tới layer thứ l, tương tự bias của layer thứ l được kí hiệu là $b^{(l)}$. Khi tối ưu một multilayer neural network, ta cần phải đi tìm các weight and bias này.

Activation function

Output của một layer (trừ input) được tính dựa bào công thức:

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

Với $f^{(l)}$ là một hàm kích hoạt phi tuyến

Ta có một vài nhận xét:

- Hàm sgn chỉ được sử dụng trong Perceptron, trong thực tế hàm sgn không được sử dụng vì nó có đạo hàm bằng 0 tại hầu hết các điểm.
- Hàm sigmoid: sigmoid(z) = 1/(1 + exp(-z)), hàm số nhận mọi giá trị thực và đầu ra bị giới hạn trong khoảng (0, 1).
- Hàm tanh: tanh(z) = (exp(z) - exp(-z))/(exp(z) + exp(-z)), hàm này có đầu ra chạy từ -1 đến 1, có tính chất zero-centered nên được sử dụng nhiều hơn sigmoid. Tuy nhiên khi đầu vào có giá trị lớn thì đạo hàm của sigmoid lẫn tanh đều gần 0, cho nên việc cập nhật các tham số khi sử dụng GD hầu như không có ý nghĩa,
- ReUL: f(z) = max(0, z), đây là hàm có tính đơn giản, đạo hàm bằng 0 tại các điểm < 0 và bằng 1 tại các điểm dương. hàm này và các biến thể của nó được sử dụng nhiều và cho kq tốt

2. Backpropagation

Phương pháp phổ biến để tối ưu multilayer neural network là GD, tuy nhiên để áp dụng được GD, ta phải tính được đạo hàm của hàm mất mát theo ma trận trọng số W bà vector bias b, trong thực tế, việc tính toán trực tiếp các giá trị này là không khả thi vì hàm mất mát không phụ thuộc trực tiếp vào các ma trận trọng số và vector bias. Để giải quyết vấn đề này, ta sử dụng phương pháp backpropagation tính đạo hàm ngược từ layer cuối về layer đầu.

Ta có:

$$\frac{\partial J}{\partial w_{ij}^{(L)}} = \frac{\partial J}{\partial z_j^{(L)}} . \frac{\partial z_j^{(L)}}{\partial w_{ij}^{(L)}} = e_j^{(L)} a_i^{(L-1)}$$

trong đó $e_j^{(L)} = \frac{\partial J}{\partial z_j^{(L)}}$ thường là một đại lượng *không quá khó để tính toán* và $\frac{\partial z_j^{(L)}}{\partial w_{ij}^{(L)}} = a_i^{(L-1)}$ vì $z_j^{(L)} = \mathbf{w}_j^{(L)T} \mathbf{a}^{(L-1)} + b_j^{(L)}$. Tương tự, đạo hàm của hàm mất mát theo bias của layer cuối cùng là

$$\frac{\partial J}{\partial b_j^{(L)}} = \frac{\partial J}{\partial z_j^{(L)}} . \frac{\partial z_j^{(L)}}{\partial b_j^{(L)}} = e_j^{(L)}$$

Bằng cách quy nap ngược từ cuối lên, ở một lớp l thấp hơn ta có thể tính được:

$$\frac{\partial J}{\partial w_{ij}^{(l)}} = \frac{\partial J}{\partial z_j^{(l)}} . \frac{\partial z_j^{(l)}}{\partial w_{ij}^{(l)}} = e_j^{(l)} a_i^{(l-1)}$$

với

$$\begin{aligned} e_j^{(l)} &= \frac{\partial J}{\partial z_j^{(l)}} = \frac{\partial J}{\partial a_j^{(l)}} . \frac{\partial a_j^{(l)}}{\partial z_j^{(l)}} \\ &= \left(\sum_{k=1}^{d^{(l+1)}} \frac{\partial J}{\partial z_k^{(l+1)}} . \frac{\partial z_k^{(l+1)}}{\partial a_j^{(l)}} \right) f^{(l)\prime}(z_j^{(l)}) = \left(\sum_{k=1}^{d^{(l+1)}} e_k^{(l+1)} w_{jk}^{(l+1)} \right) f^{(l)\prime}(z_j^{(l)}) \end{aligned}$$

Thuật toán Backpropagation tới $w_{ij}^{(l)}$, $b_i^{(l)}$

1. *Bước feedforward: Với 1 giá trị đầu vào* $\mathbf{x}$*, tính giá trị đầu ra của network, trong quá trình tính toán, lưu lại các giá trị activation* $\mathbf{a}^{(l)}$ *tại mỗi layer.*
2. *Với mỗi unit* j *ở output layer, tính*

$$e_j^{(L)} = \frac{\partial J}{\partial z_j^{(L)}}; \quad \frac{\partial J}{\partial w_{ij}^{(L)}} = a_i^{(L-1)} e_j^{(L)}; \quad \frac{\partial J}{\partial b_j^{(L)}} = e_j^{(L)}$$

3. *Với* $l = L-1, L-2, ..., 1$*, tính:*

$$e_j^{(l)} = \left(\mathbf{w}_{j:}^{(l+1)} \mathbf{e}^{(l+1)}\right) f'(z_j^{(l)})$$

4. *Cập nhật đạo hàm cho từng hệ số*

$$\frac{\partial J}{\partial w_{ij}^{(l)}} = a_i^{(l-1)} e_j^{(l)}; \quad \frac{\partial J}{\partial b_j^{(l)}} = e_j^{(l)}$$

Thuật toán Backpropagation tới $\mathbf{W}^{(l)}$ và vector bias $\mathbf{b}^{(l)}$

1. *Bước feedforward: Với một giá trị đầu vào* $\mathbf{x}$*, tính giá trị đầu ra của network, trong quá trình tính toán, lưu lại các activation* $\mathbf{a}^{(l)}$ *tại mỗi layer.*
2. *Với output layer, tính*

$$\mathbf{e}^{(L)} = \nabla_{\mathbf{z}^{(L)}} J \in \mathbb{R}^{d^{(L)}}; \ \nabla_{\mathbf{W}^{(L)}} J = \mathbf{a}^{(L-1)} \mathbf{e}^{(L)T} \in \mathbb{R}^{d^{(L-1)} \times d^{(L)}}; \ \nabla_{\mathbf{b}^{(L)}} J = \mathbf{e}^{(L)}$$

3. *Với* $l = L-1, L-2, ..., 1$*, tính:*

$$\mathbf{e}^{(l)} = \left(\mathbf{W}^{(l+1)} \mathbf{e}^{(l+1)}\right) \odot f'(\mathbf{z}^{(l)}) \in \mathbb{R}^{d^{(l)}}$$

trong đó $\odot$ *là element-wise product hay Hadamard product tức lấy từng thành phần của hai vector nhân với nhau để được vector kết quả.*

4. *Cập nhật đạo hàm cho các ma trận trọng số và vector bias:*

$$\nabla_{\mathbf{W}^{(l)}} J = \mathbf{a}^{(l-1)} \mathbf{e}^{(l)T} \in \mathbb{R}^{d^{(l-1)} \times d^{(l)}}; \quad \nabla_{\mathbf{b}^{(l)}} J = \mathbf{e}^{(l)}$$

Thuật toán Backpropagation tới $\mathbf{W}^{(l)}$ và vector bias $\mathbf{b}^{(l)}$ (mini-batch)

1. *Bước feedforward: Với toàn bộ dữ liệu (batch) hoặc một nhóm dữ liệu (mini-batch) đầu vào* $\mathbf{X}$*, tính giá trị đầu ra của network, trong quá trình tính toán, lưu lại các activation* $\mathbf{A}^{(l)}$ *tại mỗi layer. Mỗi cột của* $\mathbf{A}^{(l)}$ *tương ứng với một cột của* $\mathbf{X}$*, tức một điểm dữ liệu đầu vào.*
2. *Với output layer, tính*

$$\mathbf{E}^{(L)} = \nabla_{\mathbf{Z}^{(L)}} J; \quad \nabla_{\mathbf{W}^{(L)}} J = \mathbf{A}^{(L-1)} \mathbf{E}^{(L)T}; \quad \nabla_{\mathbf{b}^{(L)}} J = \sum_{n=1}^{N} \mathbf{e}_n^{(L)}$$

3. *Với* $l = L-1, L-2, ..., 1$*, tính:*

$$\mathbf{E}^{(l)} = \left(\mathbf{W}^{(l+1)} \mathbf{E}^{(l+1)}\right) \odot f'(\mathbf{Z}^{(l)})$$

trong đó $\odot$ *là element-wise product hay Hadamard product tức lấy từng thành phần của hai ma trận nhân với nhau để được ma trận kết quả.*

4. *Cập nhật đạo hàm cho ma trận trọng số và vector biases:*

$$\nabla_{\mathbf{W}^{(l)}} J = \mathbf{A}^{(l-1)} \mathbf{E}^{(l)T}; \quad \nabla_{\mathbf{b}^{(l)}} J = \sum_{n=1}^{N} \mathbf{e}_n^{(l)}$$

Tránh overfitting bằng weight decay

Với weight decay, hàm mất mát được cộng vào một lượng có dạng

$$\lambda R(\mathbf{W}) = \lambda \sum_{l=1}^{L} \|\mathbf{W}^{(l)}\|_F^2$$

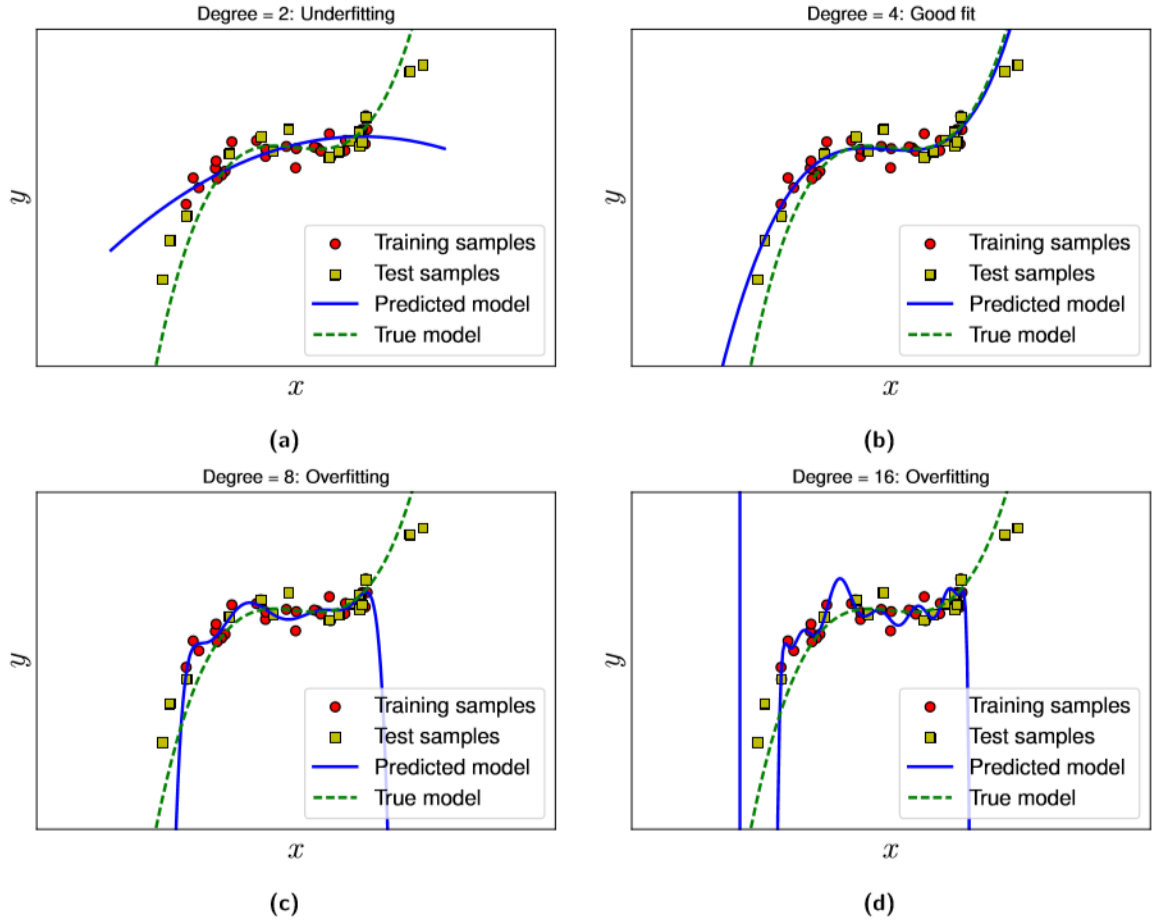
Khi làm việc với multilayer neural network, **bias hiếm khi được regularized (?)**, nên ta cần tách rời nó ra. (regularizaton: thay đổi để phù hợp) Việc tối thiểu các hệ số mô hình bằng số hạng regularization làm các thành phần của W không quá lớn, có nhiều thành phần gần về 0.

Overfitting

1. Giới thiệu

(generalization: tính tổng quát)

Trong các bài toán supervised learning, chúng ta phải đi tìm một mô hình ánh xạ các vector đầu vào thành các kết quả tương ứng trong training set, chính là hàm số f(x) sao cho $y_i \approx f(x_i)$. Thông thường, ta sẽ đi tìm các tham số mô hình của f sao cho sai số xấp xỉ là thấp nhất, mô hình càng fit càng tốt. Tuy nhiên, nếu mô hình quá fit với dữ liệu train thì nó đôi khi ko phát huy tác dụng dự đoán với tập test, hiện tượng này gọi là overfitting. Cái chúng ta cần là một mô hình có tính tổng quát áp dụng được với cả tập train lẫn tập test.



Trên đây là ví dụ cho hiện tượng overfitting, với degree = 8 or 16, mô hình dự đoán đi qua hầu hết các điểm trong tập training nhưng so với true model thì hoàn toàn không khớp, k mô tả được xu hướng của dữ liệu. Với degree = 4, ta thấy mô hình có tính tổng quát, xấp xỉ được cả tập training và test data. Tuy nhiên khi degree = 2 thì mô hình dự đoán có xu hướng đi xuống trong khi dữ liệu đnag hướng đi lên, chưa đủ để fit bộ training, ta gọi là underfitting

→ Overfitting là hiện tượng mô hình tìm được quá khớp với dữ liệu huấn luyện, khi tập training bị nhiễu thì mô hình này không áp dụng tốt cho các tập dữ liệu ngoài tập training do mất đi tính tổng quát. Overfitting thường xảy ra khi tập huấn luyện quá nhỏ hoặc độ phức tạp cảu mô hình quá cao

Đánh giá chất lượng của của mô hình

Dựa trên 2 đại lượng là training error và test error.

- Training error: Là mức độ sai khác giữa đầu ra thực sự và đầu ra dự đoán của mô hình áp dụng lên training data. Hàm mất mát này có thêm một thừa số 1/N để tính giá trị trung bình trên mỗi điểm dữ liệu. Với các bài toán regression, thường được xác định bởi MSE (mean squared error)

$$\text{training error} = \frac{1}{2N_{\text{train}}} \sum_{\text{training set}} \|\mathbf{y} - \hat{\mathbf{y}}\|_2^2$$

- Test error: Tương tự như trên, nhưng mô hình được áp dụng cho test data.

$$\text{test error} = \frac{1}{2N_{\text{test}}} \sum_{\text{test set}} \|\mathbf{y} - \hat{\mathbf{y}}\|_2^2$$

Một mô hình được gọi là tốt nếu training error và test error đều thấp, nếu training error thấp và rest error cao gọi là overfitting, còn nếu traning error và test error đều cao gọi là umderfitting. TH training error cao nhưng test error thấp rất hiếm khi xảy ra.

Để tránh overfitting, ta sẽ tìm hiểu một số kỹ thuật phổ biến

2. Validation

2.1 Validation

Một mô hình được gọi là tốt nếu cả training error và test error đêu thấp, tuy nhiên khi xây dựng mô hình ta chỉ dựa vào training data thì làm sao tính được test error. Cách đơn giản nhất là trích ra từ training set một tập con nhỏ và thực hiện đánh giá trên tập con này như là test data. Tập con nhỏ được tách ra này gọi là validation data, training set mới lúc này là phần còn lại của training set.

Lúc này ta có tới 3 đại lượng để đánh giá là training error, validation error và test error. Ta cần tìm một mô hình sao cho training error và validation error nhỏ, từ đó ta dự đoán test error cũng nhỏ

2.2 Cross-validation (k-fold cross-validation)

Trong trường hợp ta có rất ít dữ liệu trong training set. Nếu lấy quá nhiều dữ liệu trong traing set làm validation set thì lượng dữ liệu còn lại k đủ để dựng mô hình. Hiện tượng overfitting có thể xảy ra. Ta dùng cross-validation để giải quyết vấn đề này.

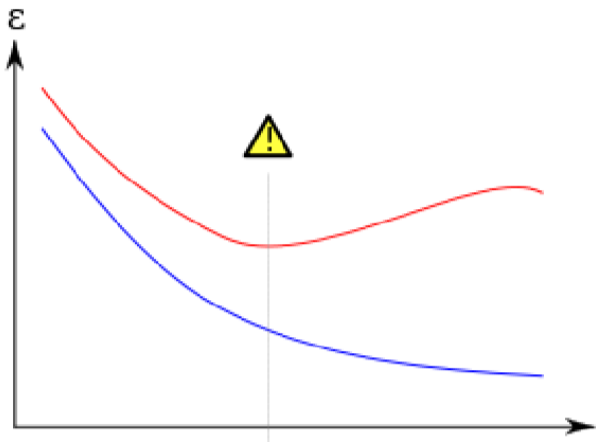
Chia nhỏ training set ra thành k tập con không giao nhau, có kích thước gần bằng nhau. Tại mỗi lần chạy, một tập con được lấy ra để làm validation set, phần còn lại là training set. Mô hình cuối được dựng dựa vào trung bình của các training eror và validation error.

Nhược điểm: Số lượng mô hình cần huấn luyện tỉ lệ thuận với k, lượng tham số thường lớn, đôi khi là số thực nên khó thực hiện.

3. Regularization

3.1 Early stop

Các thuật toán thường lặp cho tới khi hàm mất mát hội tụ để tìm nghiệm, việc này đôi khi gây ra overfitting, để tránh TH này, ta dừng thuật toán trước khi nó hội tụ. Vấn đề dừng khi nào là hợp lí?



Hình 8.3: Early stopping. Đường màu xanh là training error, màu đỏ là validation error. Trục x thể hiện số lượng vòng lặp, trục y là giá trị error. Thuật toán huấn luyện dừng lại tại vòng lặp mà validation error đạt giá trị nhỏ nhất (Nguồn: Over-fitting – Wikipedia).

Trong khi huấn luyện, ta tính toán cả training error và validation error. Nếu Training error có xu hướng giảm nhưng validation error có xu hướng tăng thì ta nên dừng.

3.2 Thêm số hạng vào hàm mất mát

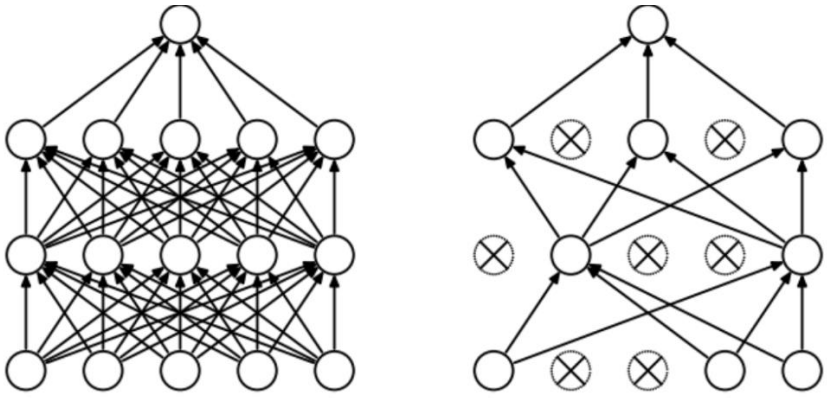
Số hạng này thường dùng để đánh giá độ phức tạp của mô hình, giá trị lớn thể hiện mô hình càng phức tạp. Hàm mất mát mới (regularization loss function) được định nghĩa:

$$L_{reg}(\theta) = L(\theta) + \lambda R(\theta)$$

Với *θ: các tham số trong mô hình*
L(θ): hàm mất mát phụ thuộc vào training set và θ
R(θ): số hạng regularization phụ thuộc vào θ
λ: tham số regularization

Có 2 hàm regularization phổ biến là l_1 norm và l_2 norm regularization. L_1 norm regularization giúp cho mô hình ít bị nhiễu hơn so với l_2 norm regularization. Tuy nhiên, đạo hàm của l_1 norm không xác định tại 0, tốn thời gian tìm nghiệm hơn trong khi l_2 norm có đạo hàm xác định tại mỗi điểm. Việc sử dụng l_2 regularization còn được gọi là **weight decay**

4. Dropout



(a) Standard Neural Net (b) After applying dropout.

Dropout là cách thức mà chúng ta giả định một phần các unit bị ẩn đi trong quá trình training, làm cho mô hình dựng được đơn giản hơn, tránh trường hợp model dựng được quá phức tạp để fit được tất cả điểm dữ liệu trên tập training

Trong Dropout, ta có hệ số p, là sô % lượng node trong 1 layer, mỗi layer có thể có hệ số p khác nhau. Tại mỗi bước trong quá trình training, ta chỉ giữ lại 1 – p số node của layer đó để training.

Nên lựa chọn p ở khoảng [0.2, 0.5]. Nếu p quá nhỏ thì không có tác dụng chống overfitting, nếu p quá lớn thì hầu như layer đó bị loại bỏ