TRUYỀN NGƯỢC

Nguyen Minh Tuan

Hanoi University of Science and Technology

06-2022

Tóm tắt nội dung

Hiện nay, mạng nơ ron nhân tạo là một trong những công nghệ mới nhất, được ứng dụng vô cùng rộng rãi trên rất nhiều lĩnh vực. Trong bài toán tối ưu hoá một mạng nơ ron, việc tính được vi phân của hàm mất mát tương ứng với các trọng số trong mạng là rất quan trọng. Truyền ngược là một phương pháp được sử dụng để làm điều này một cách cực kì hiệu quả. Bài viết dưới đây sẽ giới thiệu và chỉ ra cách sử dụng và chứng minh tính đúng đắn của phương pháp này.Tuy truyền ngược không chỉ được áp dụng trong học sâu, bài viết này sẽ chỉ tập trung vào truyền ngược trong mạng nơ ron nhân tạo.

1 Mạng Nơ ron nhân tạo là gì?

Mạng Nơ ron nhân tạo có thể được coi như một ánh xạ (khá phức tạp):

$$\hat{y} = f(\varphi(W^{(L)}\varphi(W^{(L-1)}...\varphi(W^{(1)}x))))$$

nói cách khác, đưa vào một vector đầu vào x (hay còn được gọi là $z^{(1)}$ hay $a^{(1)}$), mạng nơ ron nhân tạo sẽ biến đổi tuyến tính nó để trở thành vector $z^{(2)}$: $z^{(2)} = Wx + b$ (tuy nhiên ta thường thêm 1 vào vector x để tượng trưng cho giá trị b) rồi sử dụng một hàm kích hoạt φ để thu được giá trị $a^{(2)}$: $a^{(2)} = \varphi(z^{(2)})$, rồi lại tiếp tục biến đổi tuyến tính vector $a^{(2)}$ thành vector $z^{(3)}$, sử dụng φ để tính giá trị $a^{(3)}$ rồi cứ tiếp tục như thế cho tới khi ta thu được $z^{(L)}$ ở lớp cuối cùng của mạng. Khi này, ta sử dụng f để biến đổi $z^{(L)}$ thành vector đầu ra $\hat{y}$. Lưu ý rằng rất nhiều khi f và φ không giống nhau.

- f là hàm kích hoạt ở lớp cuối cùng của mạng.
- φ là hàm kích hoạt ở các lớp ẩn (là các lớp ngoài lớp đầu vào và đầu ra).
- $x = \begin{pmatrix} x_1 \\ x_2 \\ . \\ . \\ . \end{pmatrix}$ là vector đầu vào.
- $\hat{y} = \begin{pmatrix} \hat{y}_1 \\ \hat{y}_2 \\ . \\ . \\ . \end{pmatrix}$ là vector đầu ra.
- $L \in N^*$ là tổng số lượng lớp
- $W^{(l)} = \begin{pmatrix} w_{10}^{(l)} & w_{11}^{(l)} & . & . & . & w_{1m}^{(l)} \\ . & . & . & & & . \\ . & . & & . & & . \\ . & . & & & . & . \\ w_{n0}^{(l)} & w_{n1}^{(l)} & . & . & . & w_{nm}^{(l)} \end{pmatrix}$ là ma trận trọng số giữa lớp thứ $(l-1)$ và lớp thứ (l).

trong đó:

- $w_{ij}^{(l)}$ là trọng số giữa nút thứ j ở lớp $(l-1)$ và nút thứ i ở lớp (l)

- m là số lượng nút trong lớp $(l-1)$

- n là số lượng nút trong lớp (l)

2 Truyền ngược

Một mạng Nơ ron nhân tạo thực ra chỉ là một ánh xạ rất phức tạp. Mục đích của ta khi nói "huấn luyện mô hình" là tìm ra các trọng số thích hợp nhất để tối ưu giá trị của $\hat{y}$ sao cho nó gần giá trị thực y nhất có thể rồi dùng để dự đoán các giá trị y khi biết được x. Một cách chính xác hơn, ta cần tìm các trọng số thích hợp để tối thiểu hoá một hàm mất mát tượng trưng cho sự sai lệch giữa các giá trị mà ta dự đoán và giá trị thực. Có rất nhiều hàm mất mát khác nhau, trong đó các hàm mất mát hay được dùng là:

$$C = \begin{cases} \sum_{i=1}^{N} \frac{1}{2} (y_i - \hat{y}_i)^2 & \text{với các mô hình hồi quy} \\ \sum_{i=1}^{N} -y. \log \hat{y} - (1-y). \log(1-\hat{y}) & \text{với các mô hình phân loại} \end{cases}$$

trong đó N là số lượng điểm dữ liệu dùng để huấn luyện.

Một cách vô cùng đơn giản để tối thiểu hoá mô hình này là sử dụng gradient descent. Tuy nhiên, gradient descent (và rất nhiều thuật toán khác) yêu cầu ma trận Jacobi của hàm mất mát tương ứng với các ma trận trọng số: $\nabla_W^{(l)} C$ trong đó $l = 1, 2, .., L$, nói cách khác ta cần tính giá trị $\frac{\partial C}{\partial w_{ij}^{(l)}}$ cho tất cả phần tử $w_{ij}^{(l)}$ của ma trận W^l cho tất cả các lớp l.

Vậy ta làm việc này như thế nào? Tất nhiên ta có thể tính vi phân từng phần như bình thường. Tuy nhiên phương pháp này có một số vấn đề:

1. $\hat{y} = f(...)$ là một ánh xạ rất phức tạp. Lấy vi phân của ánh xạ này chắc chắn không hề đơn giản.

2. Có rất nhiều phép tính bị trùng lặp trong quá trình tính toán, khiến cho việc tính toán rất thiếu hiệu quả.

Một cách tiếp cận khác là sử dụng thuật toán truyền ngược: ta bắt đầu bằng việc tính vi phân của hàm mất mát tương ứng với lớp đầu ra (lớp cuối cùng) rồi dần dần tính ngược lại cho đến lớp đầu vào (lớp đầu tiên):

Ở trong phần này, m và n được sử dụng như số nút ở 2 lớp liền kề nhau. Ví dụ: m là số nút của lớp $(l-1)$ thì n sẽ là số nút ở lớp (l), hoặc khi m là số nút ở lớp (l) thì n sẽ là số nút ở lớp $(l+1)$.

Tính vi phân hàm mất mát tương ứng với lớp đầu ra vô cùng đơn giản bởi C có thể coi như một ánh xạ khá đơn giản của $\hat{y}$. Sau bước này, ta thu được $\nabla_{\hat{y}} C$.

Giờ ta sẽ bắt đầu tính ngược về:

Ta có

$$\frac{\partial C}{\partial w_{ij}^{(l)}} = \sum_{k=1}^{n} \frac{\partial C}{\partial z_k^{(l)}} \frac{\partial z_k^{(l)}}{\partial w_{ij}^{(l)}}$$

Nhưng

$$\frac{\partial z_k^{(l)}}{\partial w_{ij}^{(l)}} = \begin{cases} \frac{\partial \sum_{t=0}^{m} w_{kt}^{(l)} a_t^{(l-1)}}{\partial w_{ij}^{(l)}} = a_j & \text{khi } i = k \\ 0 & \text{khi } i \neq k. \end{cases}$$

$$\Rightarrow \frac{\partial C}{\partial w_{ij}^{(l)}} = a_j^{(l-1)} \frac{\partial z_i^{(l)}}{\partial w_{ij}^{(l)}}$$

$$\Rightarrow \frac{\partial C}{\partial w^{(l)}} = \frac{\partial C}{\partial z^{(l)}} (a^{(l-1)})^T$$

Đặt $\delta^{(l)} = \begin{pmatrix} \frac{\partial C}{\partial z_1^{(l)}} \\ \frac{\partial C}{\partial z_2^{(l)}} \\ . \\ . \\ . \end{pmatrix} \Rightarrow \frac{\partial C}{\partial w^{(l)}} = \delta^{(l)} a^{(l-1)T}$

Ta chỉ cần tính nốt $\delta^{(l)}$:

Ở lớp đầu ra, $\delta^{(L)} = \nabla_{\hat{y}} C$.

Ở các lớp còn lại:

$$\frac{\partial C}{\partial z_i^{(l)}} = \sum_{j=1}^{n} \left(\frac{\partial C}{\partial z_j^{(l+1)}} \frac{\partial z_j^{(l+1)}}{\partial z_j^{(l)}} \right) = \sum_{j=1}^{n} \left(\frac{\partial C}{\partial z_j^{(l+1)}} \frac{\partial \sum_{k=0}^{m} w_{jk}^{(l+1)} a_k^{(l)}}{\partial z_i^{(l)}} \right)$$

Trong biểu thức $\sum_{k=0}^{m} w_{jk}^{(l+1)} a_k^{(l)}$, chỉ có một đại lượng phụ thuộc vào $z_i^{(l)}$, đó là $w_{ji}^{l+1} a_i^{(l)}$ và vi phân từng phần của nó tương ứng với $z_i^{(l)}$ là $w_{ji}^{(l+1)} \varphi'(z_i^{(l)})$.

$$\Rightarrow \frac{\partial C}{\partial z_i^{(l)}} = \sum_{j=1}^{n} \left(\frac{\partial C}{\partial z_j^{(l+1)}} w_{ji}^{(l+1)} \varphi'(z_i^{(l)}) \right)$$

$$\Rightarrow \delta^{(l)} = \begin{pmatrix} \sum_{j=1}^{n} \left(\frac{\partial C}{\partial z_j^{(l+1)}} w_{j1}^{(l+1)} \varphi'(z_1^{(l)}) \right) \\ \sum_{j=1}^{n} \left(\frac{\partial C}{\partial z_j^{(l+1)}} w_{j2}^{(l+1)} \varphi'(z_2^{(l)}) \right) \\ . \\ . \\ . \end{pmatrix} = \begin{pmatrix} \sum_{j=1}^{n} \left(\frac{\partial C}{\partial z_j^{(l+1)}} w_{j1}^{(l+1)} \right) \\ \sum_{j=1}^{n} \left(\frac{\partial C}{\partial z_j^{(l+1)}} w_{j2}^{(l+1)} \right) \\ . \\ . \\ . \end{pmatrix} \circ \begin{pmatrix} \varphi'(z_1^{(l)}) \\ \varphi'(z_2^{(l)}) \\ . \\ . \\ . \end{pmatrix}$$

trong đó $\circ$ là tích Hadamard, là phép nhân các phần tử tương ứng của 2 vector/ma trận với nhau.

Thêm vào đó:

$$\begin{pmatrix} \sum_{j=1}^{n}(\frac{\partial C}{\partial z_j^{(l+1)}} w_{j1}^{(l+1)}) \\ \sum_{j=1}^{n}(\frac{\partial C}{\partial z_j^{(l+1)}} w_{j2}^{(l+1)}) \\ . \\ . \\ . \end{pmatrix} = \begin{pmatrix} w_{11}^{(l+1)} & . & . & . & w_{n1}^{(l+1)} \\ . & . & & & . \\ . & & . & & . \\ . & & & . & . \\ w_{1m}^{(l+1)} & . & . & . & w_{nm}^{(l+1)} \end{pmatrix} \begin{pmatrix} \frac{\partial C}{\partial z_1^{(l+1)}} \\ \frac{\partial C}{\partial z_2^{(l+1)}} \\ . \\ . \\ . \end{pmatrix} = (W^{(l+1)})^T \delta^{(l+1)}$$

$$\Leftrightarrow \delta^{(l)} = \varphi'(z^{(l)}) \circ (W^{(l+1)})^T \delta^{(l+1)}$$

Như vậy ta có:

$$\nabla_{W^l} C = \delta^{(l)} (a^{(l-1)})^T$$

$$\delta^{(l)} = \varphi'(z^{(l)}) \circ (W^{(l+1)})^T \delta^{(l+1)}$$

Bước cuối cùng: tính giá trị của $\varphi'(z^{(l)})$ và $a^{(l)}$ ở mỗi lớp. Để làm điều này, ta chỉ cần cho điểm dữ liệu x qua mô hình, rồi lưu lại các giá trị $a^{(l)}$ và $\varphi'(z^{(l)})$ cho mỗi lớp l bắt đầu từ lớp đầu tiên. Sau khi thực hiện xong bước này, ta đã có được tất cả những gì cần thiết để tìm ma trận Jacobi của các ma trận trọng số của mô hình.

Tài liệu

[1] Goodfellow, Ian; Bengio, Yoshua; Courville, Aaron (2016) "6.5 Back-Propagation and Other Differentiation Algorithms". Deep Learning. MIT Press. pp. 200–220. ISBN 9780262035613.

[2] Mizutani, Eiji; Dreyfus, Stuart; Nishio, Kenichi (July 2000). "On derivation of MLP back-propagation from the Kelley-Bryson optimal-control gradient formula and its application". Proceedings of the IEEE International Joint Conference on Neural Networks.