**RNN**

Mạng nơ-ron hồi quy (RNN) là một mô hình học sâu được đào tạo để xử lý và chuyển đổi đầu vào dữ liệu tuần tự thành đầu ra dữ liệu tuần tự cụ thể. Dữ liệu tuần tự là dữ liệu, chẳng hạn như từ, câu hoặc dữ liệu chuỗi thời gian, trong đó các thành phần tuần tự tương quan với nhau dựa trên ngữ nghĩa phức tạp và quy tắc cú pháp. RNN là một hệ thống phần mềm gồm nhiều thành phần được kết nối với nhau theo cách con người thực hiện chuyển đổi dữ liệu tuần tự, chẳng hạn như dịch văn bản từ ngôn ngữ này sang ngôn ngữ khác.

RNN được tạo thành từ các nơ-ron: các nút xử lý dữ liệu kết hợp cùng nhau để thực hiện các tác vụ phức tạp. Các nơ-ron được tổ chức dưới dạng lớp đầu vào, đầu ra và ẩn. Lớp đầu vào nhận thông tin để xử lý và lớp đầu ra cung cấp kết quả. Quá trình xử lý dữ liệu, phân tích và dự đoán diễn ra trong lớp ẩn.

RNN hoạt động bằng cách lần lượt truyền dữ liệu tuần tự nhận được đến các lớp ẩn. Tuy nhiên, RNN cũng có quy trình làm việc tự lặp lại hay hồi quy: lớp ẩn có thể ghi nhớ và sử dụng các đầu vào trước đó cho các dự đoán trong tương lai trong một thành phần bộ nhớ ngắn hạn. Quy trình này sử dụng đầu vào hiện tại và bộ nhớ đã lưu trữ để dự đoán chuỗi tiếp theo.

A screenshot of a graph

Description automatically generated

Các kỹ sư máy học (ML) đào tạo các mạng nơ-ron sâu như RNN bằng cách cung cấp dữ liệu đào tạo cho mô hình và tinh chỉnh hiệu năng của mô hình. Trong ML, trọng số của nơ-ron là tín hiệu để xác định mức độ ảnh hưởng của thông tin đã học trong quá trình đào tạo khi dự đoán đầu ra. Mỗi lớp trong RNN đều có trọng số bằng nhau.

RNN có cấu trúc đơn giản nhất. Tại mỗi bước thời gian, nó chỉ có một trạng thái ẩn (hidden state) được cập nhật dựa trên đầu vào hiện tại và trạng thái ẩn từ bước trước đó. Tuy nhiên khả năng nhớ thông tin xa kém hiệu quả vì trạng thái ẩn liên tục bị ghi đè bởi thông tin mới.

RNN gặp phải vấn đề vanishing gradient (độ dốc biến mất) khi xử lý các chuỗi dài, làm giảm khả năng học và nhớ thông tin ở các bước thời gian xa.

Vanishing Gradient

Bài toán độ dốc biến mất là một điều kiện, trong đó độ dốc của mô hình đạt đến 0 trong quá trình đào tạo. Khi độ dốc biến mất, RNN không học hiệu quả từ dữ liệu đào tạo, dẫn đến chưa khớp. Mô hình chưa khớp không thể hoạt động tốt trong các ứng dụng thực tế vì các trọng số chưa được điều chỉnh thích hợp. RNN sẽ có rủi ro gặp phải vấn đề độ dốc cực lớn và biến mất khi xử lý các chuỗi dữ liệu dài.

Vanishing gradient xảy ra khi các mạng neuron sâu như RNN phải học các mối quan hệ từ dữ liệu tuần tự (chuỗi thời gian), và trong quá trình lan truyền ngược qua nhiều bước thời gian, các gradient dần trở nên quá nhỏ, làm cho việc cập nhật trọng số không hiệu quả. Điều này khiến RNN khó học được các thông tin quan trọng ở những bước xa hơn trong chuỗi.

Cụ thể:

Lan truyền ngược qua thời gian (Backpropagation Through Time - BPTT): Khi huấn luyện RNN, các gradient được lan truyền ngược qua nhiều bước thời gian. Nếu gradient quá nhỏ ở mỗi bước, sau nhiều bước, nó sẽ giảm xuống gần bằng 0, dẫn đến việc mô hình không học được gì hoặc rất chậm. Điều này gọi là vanishing gradient.

Hậu quả: Khi vanishing gradient xảy ra, các trọng số của các lớp đầu tiên (gần input) của mạng không được cập nhật hiệu quả, khiến mô hình gặp khó khăn trong việc học các phụ thuộc xa hoặc các mối quan hệ dài hạn trong dữ liệu chuỗi.

Giải pháp:

LSTM (Long Short-Term Memory) và GRU (Gated Recurrent Unit) là hai kiến trúc cải tiến của RNN được phát triển để giải quyết vấn đề vanishing gradient bằng cách sử dụng các cơ chế cổng (gates) để duy trì và kiểm soát thông tin qua nhiều bước thời gian.

Chuẩn hóa gradient (Gradient Clipping): Giới hạn độ lớn của gradient trong một khoảng nhất định để ngăn chúng không trở nên quá nhỏ hoặc quá lớn.

**LSTM**

LSTM là một phiên bản cải tiến của RNN, với cấu trúc phức tạp hơn. Nó bao gồm cell state (trạng thái ô) và ba cổng: Forget gate, Input gate, và Output gate. Các cổng này cho phép LSTM kiểm soát thông tin nào cần được lưu giữ, thông tin nào cần bị lãng quên, và thông tin nào cần xuất ra.

LSTM còn duy trì một trạng thái ẩn (hidden state) và một trạng thái bộ nhớ (cell state). Trạng thái bộ nhớ là nơi lưu trữ thông tin dài hạn, giúp điều chỉnh và giữ lại thông tin qua nhiều bước thời gian.

* Forget Gate: Quyết định thông tin nào trong cell state cần bị lãng quên hoặc giữ lại.
* Input Gate: Quyết định thông tin mới nào từ đầu vào sẽ được thêm vào cell state.
* Output Gate: Quyết định phần nào của cell state sẽ được sử dụng để tạo ra đầu ra của nơ-ron.

**GRU**

GRU là một phiên bản đơn giản hơn của LSTM. Nó chỉ sử dụng hai cổng chính: Update gate và Reset gate. GRU không có trạng thái bộ nhớ tách riêng, mà sử dụng trực tiếp trạng thái ẩn (hidden state) để duy trì thông tin.

* Update gate kiểm soát lượng thông tin từ trạng thái trước cần được giữ lại và cập nhật.
* Reset gate quyết định liệu thông tin từ trạng thái trước có bị lãng quên hay không.

**Sự phức tạp giữa LSTM và GRU**

* LSTM: Phức tạp hơn vì nó có nhiều cổng và duy trì thêm trạng thái bộ nhớ riêng. Điều này giúp LSTM linh hoạt hơn trong việc giữ lại thông tin dài hạn, nhưng cũng khiến nó tiêu tốn tài nguyên tính toán hơn.
* GRU: Đơn giản hơn do chỉ có hai cổng và không có trạng thái bộ nhớ riêng biệt. Điều này làm cho GRU nhanh hơn và dễ huấn luyện hơn trong một số trường hợp.

**Hiệu suất giữa LSTM và GRU**

* LSTM: Hiệu quả hơn khi xử lý các chuỗi dài và phức tạp, đặc biệt khi cần duy trì thông tin trong nhiều bước thời gian.
* GRU: Hiệu quả hơn với các chuỗi ngắn và dữ liệu có mối quan hệ đơn giản hơn, do tính chất đơn giản hơn về cấu trúc.

Autoencoder là mạng ANN có khả năng học hiệu quả các biểu diễn của dữ liệu đầu vào mà không cần nhãn, nói cách khác, giả sử từ một hình ảnh có thể tái tạo ra một bức ảnh có liên quan chặt chẽ với bức ảnh đầu vào đó. Đầu vào loại mạng neural này không có nhãn, tức là mạng có khả năng học không giám sát (Unsupervised Learning)

Đầu vào được mạng mã hóa để chỉ tập trung vào các đặc trưng quan trọng nhất, tùy vào bài toán cụ thể. Các biểu diễn (coding) thường có chiều nhỏ hơn so với input của Autoencoder, đó là lý do Autoencoder có thể dùng trong các bài toán giảm chiều dữ liệu hoặc trích xuất đặc trưng.

Bên cạnh đó, Autoencoder còn có thể được sử dụng để tạo ra các mô hình học tập trung (Generative learning models), ví dụ như huấn luyện một tập hợp các khuôn mặt để tạo ra các khuôn mặt mới.

Autoencoder bao gồm 3 phần chính

Encoder: Module có nhiệm vụ nén dữ liệu đầu vòa thành một biễu diễn được mã hóa (coding), thường nhỏ hơn một vài bậc so với dữ liệu đầu vào

Bottleneck: Module chứa các biểu diễn tri thức được nén (chính là output của Encoder), đây là phần quan trọng nhất của mạng bới nó mang đặc trưng của đầu vào, có thể dùng để tái tạo ảnh, lấy đặc trưng của ảnh, ….

Decoder: Module giúp mạng giải nén các biểu diễn tri thức và tái cấu trúc lại dữ liệu từ dạng mã hóa của nó, mô hình học dựa trên việc so sánh đầu ra của Decoder với đầu vào ban đầu (Input của Encoder)

Vanishing gradient là một vấn đề phổ biến trong quá trình huấn luyện các mạng nơ-ron sâu (deep neural networks), đặc biệt là trong các mạng nơ-ron hồi quy (RNN) và mạng nơ-ron tích chập (CNN) khi có nhiều lớp. Nó xảy ra khi các gradient (độ dốc) trở nên quá nhỏ trong quá trình backpropagation, làm cho việc cập nhật trọng số ở các lớp đầu tiên trở nên rất chậm hoặc thậm chí không diễn ra.

Cách thức xảy ra:

Trong quá trình huấn luyện một mạng nơ-ron, thuật toán backpropagation được sử dụng để tính toán các gradient, và dựa trên các gradient này để cập nhật các trọng số của mạng bằng cách sử dụng phương pháp gradient descent. Tuy nhiên, khi mạng có nhiều lớp hoặc mạng RNN có chuỗi dài, gradient có thể bị giảm rất nhỏ khi lan truyền ngược từ các lớp cuối cùng về các lớp đầu tiên. Điều này xảy ra đặc biệt khi các hàm kích hoạt như sigmoid hoặc tanh được sử dụng, vì chúng làm cho gradient trở nên rất nhỏ trong khoảng giá trị nhất định.

Hệ quả của vanishing gradient:

Học rất chậm: Các trọng số ở những lớp đầu tiên hoặc ở bước thời gian đầu trong RNN không được cập nhật hiệu quả do gradient quá nhỏ, khiến mô hình không thể học tốt hoặc học rất chậm từ dữ liệu.

Giảm hiệu suất của mạng: Vì các lớp ban đầu không được cập nhật đúng cách, mạng không thể học được các đặc trưng quan trọng từ dữ liệu đầu vào, ảnh hưởng đến toàn bộ hiệu suất của mạng.

Giải pháp cho vanishing gradient:

Sử dụng các hàm kích hoạt khác: Các hàm kích hoạt như ReLU (Rectified Linear Unit) thường được sử dụng để tránh vanishing gradient, vì ReLU không bị bão hòa như sigmoid hay tanh.

Các kiến trúc mạng đặc biệt: Các mô hình như LSTM (Long Short-Term Memory) và GRU (Gated Recurrent Unit) trong RNN được thiết kế đặc biệt để giải quyết vấn đề vanishing gradient.

Khởi tạo trọng số thông minh hơn: Các phương pháp khởi tạo trọng số như Xavier initialization hoặc He initialization giúp giảm bớt vấn đề vanishing gradient.

Tóm lại, vanishing gradient là một trong những vấn đề quan trọng cần giải quyết khi huấn luyện mạng nơ-ron sâu, đặc biệt trong các mạng có nhiều lớp hoặc chuỗi dài.

Backpropagation (lan truyền ngược) là một thuật toán được sử dụng để huấn luyện các mạng nơ-ron nhân tạo (Artificial Neural Networks) bằng cách tối ưu hóa các trọng số của mạng. Thuật toán này dựa trên phương pháp gradient descent, giúp giảm thiểu hàm mất mát (loss function) và cải thiện hiệu suất của mạng trong các tác vụ như phân loại hoặc dự đoán.

Các bước cơ bản của Backpropagation:

Forward pass (Lan truyền tiến):

Dữ liệu đầu vào đi qua các lớp của mạng nơ-ron, từ lớp đầu vào, qua các lớp ẩn, đến lớp đầu ra.

Tại lớp đầu ra, mạng sẽ đưa ra một dự đoán dựa trên trọng số hiện tại của các kết nối giữa các nơ-ron.

Sau đó, hàm mất mát (loss function) sẽ tính toán độ chênh lệch giữa giá trị dự đoán và giá trị thực tế (label). Ví dụ, trong bài toán phân loại, có thể sử dụng hàm mất mát như Cross-Entropy hoặc Mean Squared Error.

Tính toán lỗi (Error calculation):

Sau khi có được giá trị mất mát (error), quá trình backpropagation sẽ được sử dụng để truyền ngược lỗi này qua mạng nhằm cập nhật các trọng số.

Backward pass (Lan truyền ngược):

Dựa trên phương pháp gradient descent, backpropagation tính toán gradient (độ dốc) của hàm mất mát theo từng trọng số của mạng.

Gradient này cho biết mức độ ảnh hưởng của mỗi trọng số đến giá trị mất mát tổng thể.

Lỗi từ lớp đầu ra được truyền ngược trở lại các lớp ẩn và lớp đầu vào, điều chỉnh trọng số dựa trên gradient. Các trọng số này sẽ được điều chỉnh ngược dần từ các lớp cuối về lớp đầu.

Cập nhật trọng số:

Dựa trên gradient tính được, các trọng số của mạng được cập nhật theo hướng giảm giá trị của hàm mất mát.

Phương trình cập nhật trọng số phổ biến nhất là: A black background with white text

Description automatically generated

Vai trò của Backpropagation:

Backpropagation giúp tối ưu hóa mạng nơ-ron bằng cách điều chỉnh trọng số sao cho sai số giữa dự đoán của mạng và giá trị thực giảm dần. Nhờ đó, mạng có thể học được các mối quan hệ từ dữ liệu và dần dần cải thiện hiệu suất trong các bài toán như phân loại, hồi quy và nhận diện mẫu.

Vấn đề gặp phải:

Vanishing gradient: Như đã đề cập, khi các gradient quá nhỏ, việc cập nhật trọng số sẽ gặp khó khăn và quá trình học trở nên chậm chạp, đặc biệt trong các mạng sâu hoặc RNN.

Exploding gradient: Ngược lại với vanishing gradient, đôi khi gradient có thể trở nên quá lớn, gây ra hiện tượng cập nhật trọng số không ổn định.

Các giải pháp:

Sử dụng các hàm kích hoạt như ReLU thay vì Sigmoid hay Tanh để giảm vanishing gradient.

Các mô hình như LSTM và GRU có thể giải quyết vấn đề gradient trong các mạng RNN.

Gradient clipping là một kỹ thuật để giới hạn giá trị của gradient khi nó trở nên quá lớn.

Tóm lại, backpropagation là một phương pháp quan trọng giúp các mạng nơ-ron học cách điều chỉnh các trọng số một cách hiệu quả để giải quyết các bài toán phức tạp.

Trọng số (weights) trong mạng nơ-ron là các giá trị số được gán cho các kết nối giữa các nơ-ron ở các lớp khác nhau. Trọng số đại diện cho mức độ ảnh hưởng của một nơ-ron đối với nơ-ron tiếp theo trong mạng, và chúng là những tham số quan trọng mà mô hình học được trong quá trình huấn luyện.

Vai trò của trọng số:

Trọng số điều chỉnh đầu vào từ một nơ-ron ở lớp trước đến nơ-ron ở lớp sau. Chúng kiểm soát mức độ mà mỗi tín hiệu đầu vào đóng góp vào đầu ra cuối cùng của mạng.

Trong quá trình học, các trọng số được cập nhật liên tục dựa trên quá trình backpropagation và gradient descent để giảm sai số giữa đầu ra dự đoán và kết quả thực tế.

Cách thức hoạt động của trọng số:

Kết nối giữa các nơ-ron: Mỗi nơ-ron trong một lớp liên kết với một hoặc nhiều nơ-ron ở lớp tiếp theo thông qua các trọng số. Khi một tín hiệu đi qua mạng, đầu vào được nhân với trọng số của từng kết nối. Kết quả này sau đó được tổng hợp lại và đưa qua một hàm kích hoạt để tạo ra đầu ra cho nơ-ron ở lớp sau.

Biến đổi tín hiệu đầu vào: Trọng số đóng vai trò biến đổi tín hiệu đầu vào theo cách mà mô hình cần để dự đoán đúng. Chúng xác định xem nơ-ron tiếp theo sẽ kích hoạt mạnh hay yếu dựa trên giá trị trọng số.

Ví dụ:

Giả sử một mạng nơ-ron đơn giản với đầu vào x1,x2​ và hai trọng số tương ứng là w1,w2 Đầu ra của nơ-ron có thể được tính như sau:

A black background with white text

Description automatically generated

Kết quả này sau đó có thể được đưa qua một hàm kích hoạt như sigmoid hoặc ReLU để quyết định đầu ra cuối cùng.

Tầm quan trọng của trọng số trong quá trình học:

Huấn luyện mạng nơ-ron chính là quá trình tìm ra các giá trị trọng số tối ưu. Ban đầu, trọng số thường được khởi tạo ngẫu nhiên. Qua các bước huấn luyện (forward và backward pass), chúng được điều chỉnh dần dần để giảm thiểu sai số.

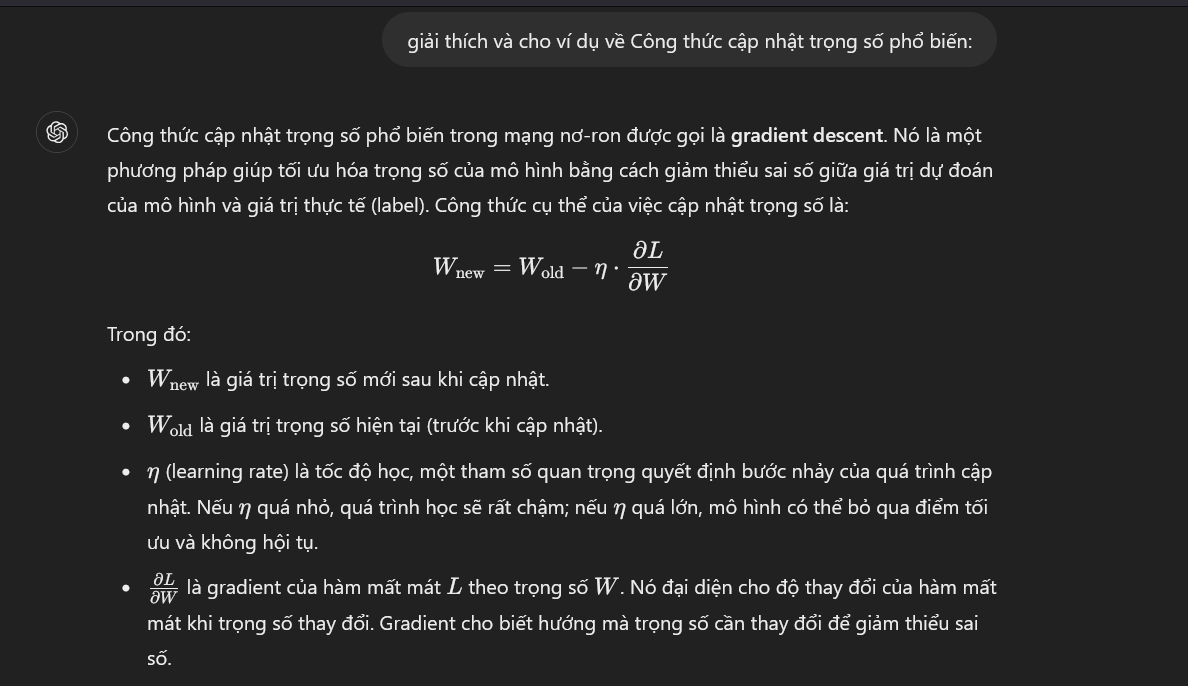
Trọng số giúp mô hình tìm hiểu các đặc trưng và mối quan hệ ẩn trong dữ liệu. Khi trọng số được điều chỉnh đúng cách, mô hình có thể dự đoán chính xác hoặc thực hiện tốt các tác vụ mà nó được huấn luyện.

Cập nhật trọng số:

Khi mạng học từ dữ liệu, các trọng số được cập nhật qua từng bước thông qua gradient descent. Trong backpropagation, gradient của hàm mất mát (loss function) được tính toán theo từng trọng số, từ đó điều chỉnh trọng số để giảm sai số.

Công thức cập nhật trọng số phổ biến: A black screen with white text

Description automatically generated

 A screenshot of a computer program

Description automatically generated A black screen with white text

Description automatically generated A black screen with white text

Description automatically generated