Họ và tên: Đặng Anh Tiến – 20520800
Lớp: CS431.N11

Bài tập LT12 - Kể chuyện về quá trình phát triển của các mạng học sâu trên dữ liệu văn bản

Bài làm

1. Vanilla RNN

Đối với dữ liệu dạng chuỗi (sequential data) – tức là dữ liệu bị thay đổi ý nghĩa khi thay đổi trình tự của chúng, các mô hình feedforward network truyền thống không thể xử lí được. Cần có một cơ chế có thể lưu lại dữ liệu trong quá khứ để hỗ trợ dự đoán cho tương lai.

Recurrent Neural Network (RNN) là một biến thể của FeedForward Network, có thể xử lí sequential data và được huấn luyện để lưu giữ các tri thức trong quá khứ. RNN có khái niệm về "**memory**" (trí nhớ) giúp lưu lại các thông tin (information) và trạng thái (state) từ các input trước để hỗ trợ tạo ra các chuỗi output kế tiếp.

Với mỗi input được lan truyền ở mỗi thời điểm, RNN kết hợp với hidden units ở thời điểm trước để tính giá trị **hidden units** và **output** ở thời điểm hiện tại và truyền hidden units sang các thời điểm tiếp theo. Sau đó thuật toán Backpropagation sẽ update weight theo từng thời điểm.

Có 3 dạng RNN: One to One, One to Many, Many to Many.

Ưu điểm:

- Có thể xử lí dữ liệu dạng chuỗi.
- Có thể xử lí input ở độ dài bất kì.
- Có khả năng "nhớ" các thông tin trong quá khứ.

Nhược điểm:

- Xử lí mỗi input tuần tự, không vận dụng triệt để tính toán song song của GPU.
- Không thể xử lí dựa vào các input đến từ tương lai.
- Hiện tượng Vanishing Gradient, chuỗi càng dài, cập nhật gradient không đáng kể.
- Khi xử lí các chuỗi dài, RNN gặp vấn đề việc "nhớ" các thông tin trong quá khứ.

Vấn đề xử lí input ở thời điểm tương lai có thể được giải quyết bởi một biến thể khác của RNN là Bidirectional RNN cho phép học thêm thông tin theo chiều ngược lại, kết hợp thông tin từ hai chiều để đưa ra các dự đoán.

2. Gated Recurrent Units/Long Short Term Memory

Vấn đề lớn nhất của Vanilla RNN là **long term dependency** (phụ thuộc dài) - không thể nhớ thông tin trong quá khứ tốt bởi hiện tượng Vanishing Gradient. Đối với dữ liệu chuỗi, tại một thời điểm, để đưa ra các dự đoán hiệu quả cần biết được các thông tin trong quá khứ nào cần được nhấn mạnh, cần được giữ lại, cần bị bỏ qua.

Cả GRU lẫn LSTM đều là cải tiến để khắc phục các vấn đề mà Vanilla RNN gặp phải. Bằng cách thêm cơ chế "**gate**" (cổng) cho phép quyết định thông tin nào được lưu giữ, thông tin nào sẽ được sử dụng trong tương lai tại mỗi thời điểm. Nhờ đó, nó có thể chuyển các thông tin cần thiết xuyên suốt chuỗi để đưa quyết định.

2.1. Long Short Term Memory(LSTM)

LSTM là một cải tiến của RNN có khả năng học chuỗi dài rất tốt. Sự thành công của LSTM là nhờ vào cơ chế Gate. Thay vì như Vanilla LSTM, tại mỗi time step chỉ xử lí dựa trên input tại thời điểm đó và hidden state từ thời điểm trước. Trong khi LSTM xử lí phức tạp hơn, mỗi time step sẽ lấy input từ 3 trạng thái khác nhau: input hiện tại, bộ nhớ ngắn hạn từ cell trước và cuối cùng là bộ nhớ dài hạn trong quá khứ. Các cell sử dụng cổng để điều chỉnh thông tin, quyết định thông tin được giữ lại, thông tin sẽ bị loại bỏ trước khi truyền thông tin dài hạn hoặc ngắn hạn cho các thời điểm kế tiếp. Có 3 gate được sử dụng trong LSTM: **Input Gate**, **Forget Gate** và **Output Gate**

- **Input Gate**: quyết định thông tin nào sẽ được lưu trữ trong "bộ nhớ" dài hạn. Đóng vai trò như bộ lọc các thông tin không hữu ích.
- **Forget Gate**: quyết định thông tin từ bộ nhớ dài hạn nào sẽ được giữ lại hoặc bị loại bỏ.
- **Output Gate:** sử dụng thông tin từ input hiện tại, thông tin từ từ bộ nhớ ngắn hạn ở thời điểm trước và bộ nhớ dài hạn vừa được tính toán để tạo ra bộ nhớ ngắn hạn mới – sẽ được truyền đi trong thời gian kế tiếp.

2.2.Gated Recurent Units (GRU)

Tương tự LSTM, GRU cũng là một cải tiến của Vanilla RNN khắc phục vấn đề phụ thuộc dài. GRU sử dụng cơ chế gate là **Update Gate** và **Reset Gate.**

- **Update Gate:** quyết định lượng thông tin ở trạng thái trước sẽ được truyền sang trạng thái kế tiếp. Điều này rất hiệu quả bởi vì mô hình có thể quyết định sao chép toàn bộ thông tin trong quá khứ và tránh được nguy cơ vanishing gradient.
- **Reset Gate:** quyết định lượng thông tin trong quá khứ cần bị loại bỏ, cũng có nghĩa là xác định tầm quan trọng của thông tin trong các trạng thái trước

3. Attention

Sequence to Sequence (Seq2Seq) là một loại kiến trúc RNN để xử lí các dạng bài toán phức tạp như Machine Translation, Question Answering. Seq2Seq là dạng mô hình được train để biến đổi chuỗi từ một domain này sang một domain khác (Ví dụ: Tiếng Anh -> Tiếng Việt, sinh ra câu trả lời từ câu hỏi ,…)

Sử dụng kiến trúc Encoder-Decoder, cả 2 đều sử dụng RNN/LSTM để xử lí chuỗi. Encoder có nhiệm vụ đọc toàn bộ chuỗi và tổng kết thông tin thành một vector. Decoder sử dụng vector này để tạo ra các dự đoán.

Seq2Seq bản chất cũng là 2 RNN/LSTM kết hợp với nhau nên cũng có các nhược điểm như long term dependency, vanishing gradient… Ngoài ra, Decoder tạo các dự đoán từ một vector tóm tắt là không đủ thông tin. Khi encoding, chuỗi có nhiều kích thước khác nhau, mang lượng thông tin khác nhau mà encode thành một vector như nhau có vẻ là không hợp lí. Dẫn đến Seq2Seq dựa trên RNN/LSTM đều có độ chính xác giảm đối với những chuỗi dài.

Attention là một cơ chế giúp khắc phục vấn đề phụ thuộc dài. Ý tưởng chính của Attention là cho phép Decoder đưa ra các dự đoán khi chỉ tập trung vào một vài input chính, bằng cách tính trọng số của các input để tìm ra input có đóng góp lớn nhất trong dự đoán tại mỗi time step.

Decoder đưa ra các dự đoán tại mỗi time step thực hiện các bước tính toán bao gồm: alignment scores, weights và context vector.

- **Alignment score:** Sử dụng hidden state ở time step hiện tại cùng với đầu ra của Decoder trước đó để tính toán score – cho biết input chuỗi đầu vào hiện tại sẽ căn chỉnh để đóng góp như thế nào cho output ở thời điểm hiện tại.
- **Weights:** được tính bằng cách áp dụng Softmax để đưa các alignment score về khoảng [0, 1].
- **Context Vector:** là vector sẽ được đưa vào Decoder ở mỗi time step, được tính bằng cách cộng có trọng số các input đầu vào và weights.

Các nghiên cứu cho thấy việc áp dụng Attention thực sự giúp cải thiện độ chính xác các mô hình Seq2Seq. Độ chính xác được cải thiện trên các chuỗi dài.

4. Transformer

Tuy Attention giúp Seq2Seq dự đoán tốt trên các chuỗi dài, tuy nhiên, việc vẫn sử dụng kiến trúc RNN vẫn là hạn chế của mô hình này. Bởi RNN không thể tận dụng triệt để tính toán song song của GPU, cũng như vanishing gradient vẫn còn là vấn đề chưa được giải quyết.

Trong bài báo "Attention is all you need", mô hình Transformer ra đời và như một vị cứu tinh đã khắc phục được các vấn đề này. Thật vậy, Attention là tất cả cần thiết cho mô hình này. Sử dụng cơ chế Multihead Attention, trong đó gồm nhiều FeedForward kết hợp với nhau thay vì các RNN như truyền thống. Việc sử dụng FeedForward giúp mô hình có thể tận dụng thành công khả năng tính toán song song của GPU. Trong các mô hình RNN, input được đưa vào tuần tự, FeedForward cho phép toàn bộ input được truyền vào cùng một lúc.

Self Attention là thành phần chính của Transformer, sử dụng 3 thành phần: Query, Key, Value để tính attention score. Giá trị này đo lường mức độ "tập trung" trên các input so với các thành phần khác. Tuy nhiên, do attention score được tính trên toàn bộ input cùng lúc, 2 câu có cùng số lượng từ nhưng khác thứ tự sẽ có kết attention score khác nhau. Multi-head Attention làm cho Self Attention có tính phân biệt mạnh mẽ hơn bằng cách xếp chồng nhiều lớp Self Attention lại với nhau kết hợp với FeedForward Network giúp tạo các context vector mạnh hơn.

Hầu hết các SOTA hiện nay trong lĩnh vực NLP đều sử dụng có sử dụng trong mô hình của mình. Một số kiến trúc SOTA hiện tại khác sử dụng một phần trong Encoder/Decoder của Transformer, có thể kể đến như BERT, GPT-2, XLNet, BART, Longformer,… Hầu hết các mô hình này đều có pretrained trên một data rất lớn, finetune các mô hình này cho các downstream task giúp cải thiện độ chính xác rất nhiều.

Tham khảo:

- https://machinelearningmastery.com/an-introduction-to-recurrent-neural-networks-and-the-math-that-powers-them/
- https://analyticsindiamag.com/lstm-vs-gru-in-recurrent-neural-network-a-comparative-study/
- https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21
- https://towardsdatascience.com/attention-is-all-you-need-discovering-the-transformer-paper-73e5ff5e0634
- https://jalammar.github.io/illustrated-transformer/