References:

1. [Understanding LSTM Networks -- colah's blog](https://colah.github.io/posts/2015-08-Understanding-LSTMs/?authuser=0)
2. [Language Model + RNN.pdf - Google Drive](https://drive.google.com/file/d/1xPbLXHJHTvSSPLAf7sKSIw68vhVjajVs/view)
3. [nc.dvi (jku.at)](https://www.bioinf.jku.at/publications/older/2604.pdf)

Video học:

1. [LSTM làm việc như thế nào? (youtube.com)](https://www.youtube.com/watch?v=yLZqQMPKXnc)
2. [Giải thích Mạng LSTM dễ hiểu và lập trình với Pytorch và Tensorflow (youtube.com)](https://www.youtube.com/watch?v=YcRPPy3EiJs)
3. [Text Generation in Deep Learning with Keras: Fundamentals (youtube.com)](https://www.youtube.com/watch?v=Ait1_xNmxII&list=PLQflnv_s49v9QOres0xwKyu21Ai-Gi3Eu)
4. [HCMUS - Summer - AI - Machine Learning part 2 - RNN, LSTM (youtube.com)](https://www.youtube.com/watch?v=PJKtY4WWwbQ)
5. [Sequence Models Complete Course (youtube.com)](https://www.youtube.com/watch?v=S7oA5C43Rbc)

“LSTM prevent the vanishing gradient problem”

Source: [Bài 14: Long short term memory (LSTM) | Deep Learning cơ bản (nttuan8.com)](https://nttuan8.com/bai-14-long-short-term-memory-lstm/)

Ví dụ dễ hiểu:

* Đoạn đầu tiên “Mặt trời mọc ở hướng …”, ta có thể chỉ sử dụng các từ trước trong câu để đoán là đông.
* Tuy nhiên, với đoạn, “Tôi là người Việt Nam. Tôi đang sống ở nước ngoài. Tôi có thể nói trôi chảy tiếng …” thì rõ ràng là chỉ sử dụng từ trong câu đấy hoặc câu trước là không thể dự đoán được từ cần điền là Việt.

=> Ta cần các thông tin từ state ở trước đó rất xa => cần long term memory điều mà RNN không làm được.

=> Cần một mô hình mới để giải quyết vấn đề này

=> Long short term memory (LSTM) ra đời.

LSTM (Long Short Term Memory)

|  | h(t-1): lịch sử phía trước  x(t): từ hiện tại  Từ C(t) đi qua rất nhiều bước để học được h(t).  (Simple RNN thì học ngay từ lịch sử phía trước và từ hiện tại)  ? Liệu có cần dùng hết lịch sử phía trước hoặc dùng hết thông tin các từ hiện tại hay không |
| --- | --- |
|  | Ví dụ: Tôi đang ăn. Mẹ tôi đang nấu cơm.  Xuất hiện token mang tính chất kết thúc (dấu “.”)  => LSTM chọn lọc thông tin qua:   * Forget (gate): lấy …% câu ở trước. * Input (gate): lấy từ “Mẹ” ở câu hiện tại. * Output (gate) … %   => Kiểm soát qua 3 cổng: forget, input, output. Sử dụng hàm sigmoid => giá trị chạy từ [0,1]. |
|  | f(t): % forget   * forget 60% => học 40%. * học qua hàm sigmoid.   i(t): input  o(t): output |
|  |  |
|  |  |
|  |  |
|  | |
|  | |
|  | |
|  | |
|  | |

Bài toán sinh từ

| Data Preprocessing | from tensorflow.keras.preprocessing.sequence import pad\_sequences  from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout, Bidirectional  from tensorflow.keras.preprocessing.text import Tokenizer  from tensorflow.keras.models import Sequential  from tensorflow.keras.optimizers import Adam  from tensorflow.keras import regularizers  import tensorflow.keras.utils as ku  import numpy as np | thư viện Keras từ tf cung cấp các class, function xử lý text, xây dựng model LSTM. |
| --- | --- | --- |
| Tokenizer.  Tạo các input sequences. | => | * tạo một bộ tokenizer để ánh xạ các từ trong dữ liệu văn bản thành các số nguyên * fit\_on\_texts(corpus) giúp tokenizer học từ vựng trong corpus ban đầu. * texts\_to\_sequences(): chuyển thành danh sách số nguyên. * Với mỗi câu, tạo ra nhiều chuỗi nhỏ hơn gọi là n-gram sequence.   Ví dụ, với câu "despite of wrinkles",  => có các chuỗi con như  ['despite'], ['despite', 'of'], ['despite', 'of', 'wrinkles'].  => Tất cả các chuỗi này được lưu vào input\_sequences. |
| Padding cho input sequences để lấy nhãn theo chiều dài nhất | # pad sequences  max\_sequence\_len = max([len(x) for x in input\_sequences])  input\_sequences = np.array(pad\_sequences(input\_sequences, maxlen=max\_sequence\_len, padding='pre')) | pad\_sequences sẽ chuẩn hóa độ dài của tất cả các input\_sequences sao cho chúng có cùng độ dài. Bất kỳ sequence nào ngắn hơn sẽ được thêm padding (các số 0) vào phía trước (padding='pre').  Ví dụ padding: |
| Chia thành các nhãn | # create predictors and label  predictors, label = input\_sequences[:,:-1],input\_sequences[:,-1]  print(label)  label = ku.to\_categorical(label, num\_classes=total\_words) | cắt từ trên xuống để chia thành nhãn    từ cuối cùng sẽ là label: **input\_sequences[:,-1]**   * label: * predictor: |
| Xây dựng mô hình LSTM | model = Sequential()  model.add(Embedding(total\_words, 100, input\_length=max\_sequence\_len-1))  model.add(Bidirectional(LSTM(150, return\_sequences = True)))  model.add(Dropout(0.2))  model.add(LSTM(100))  model.add(Dense(total\_words/2, activation='relu', kernel\_regularizer=regularizers.l2(0.01)))  model.add(Dense(total\_words, activation='softmax'))  model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])  print(model.summary()) | **Embedding layer**: Chuyển đổi các số nguyên đại diện cho từ thành vector có kích thước 100 chiều.  **Bidirectional LSTM**: LSTM 150 đơn vị chạy hai chiều (từ trái sang phải và từ phải sang trái). Điều này giúp mô hình hiểu rõ ngữ cảnh hơn.  **Dropout**: 20% các neurons sẽ bị loại bỏ ngẫu nhiên để tránh overfitting.  **LSTM 100 đơn vị**: Một lớp LSTM thông thường với 100 đơn vị.  **Dense layer**: Kết nối toàn bộ với hàm relu và kernel\_regularizer để ngăn chặn overfitting.  **Output layer (Dense với softmax)**: Đây là lớp cuối cùng với softmax, cho phép mô hình dự đoán xác suất của các từ tiếp theo.  **Compile model**: Sử dụng categorical\_crossentropy làm loss function và Adam làm optimizer. |
| Training | history = model.fit(predictors, label, epochs=100, verbose=1) |  |
| Text Generation | for \_ in range(next\_words):  token\_list = tokenizer.texts\_to\_sequences([test\_seq])[0]  token\_list = pad\_sequences([token\_list], maxlen=max\_sequence\_len-1, padding='pre')  predicted = model.predict\_classes(token\_list, verbose=0)  output\_word = ""  for word, index in tokenizer.word\_index.items():  if index == predicted:  output\_word = word  break  test\_seq += " " + output\_word | sinh ra từ tiếp theo, rồi lại đưa tất cả các từ vào model để sinh ra tiếp từ tiếp theo.  Mỗi lần như vậy, chuyển test\_seq thành danh sách số nguyên, thêm padding, và dùng model để dự đoán từ tiếp theo. |
| Text Generation (cont.) | Tham khảo các tổ chức của code này: [Vietnamese-Text-Generator/function/generator.py at main · Narius2030/Vietnamese-Text-Generator (github.com)](https://github.com/Narius2030/Vietnamese-Text-Generator/blob/main/function/generator.py) | |

