Đại học Bách Khoa Hà Nội
Viện Công nghệ thông tin và Truyền thông

ĐỒ ÁN MÔN HỌC

Phân loại văn bản tin tức

Môn: Học máy
Mã lớp: 95090

Giảng viên hướng dẫn: **TS. Nguyễn Nhật Quang**

Nhóm thực hiện

Phan Ngọc Lân - 20142505
Nguyễn Duy Mạnh - 20142857

Hà Nội, 5/2017

Mục lục

Phân công công việc

Họ và tên	MSSV	Công việc
Phan Ngọc Lân	20142505	Cài đặt mạng neuron và demo. Báo cáo phần lý thuyết mạng hồi quy, cài đặt và kết quả.
Nguyễn Duy Mạnh	20142857	Lọc, chỉnh sửa và tiền xử lý dữ liệu. Báo cáo phần lý thuyết mạng neuron và các tập dữ liệu.

1. Bài toán

1.1. Mô tả và yêu cầu

Phân loại tin tức là bài toán có ứng dụng rộng rãi không chỉ trong các ứng dụng và dịch vụ tin tức (news aggregator), mà còn có ảnh hưởng trong việc sàng lọc dữ liệu hay tìm kiếm thông tin nói chung. Việc phân loại dặc biệt hữu ích trong các hệ thống thu thập tin tức, trong đó các tin có thể đến từ nhiều nguồn với các cách phân loại khác nhau, không rõ ràng hoặc thậm chí không có phân loại trước.

Đầu vào : 1 văn bản tin tức (bao gồm tiêu đề và nội dung) và 1 số các chủ đề có sẵn.

Đầu ra : Chủ đề phù hợp nhất với văn bản được đưa vào.

Ví dụ : Với tập chủ đề (Sports, World, Business, Sci/Tech),

Input	Output
Tiêu đề : Fears for T N pension after talks **Nội dung :** Unions representing workers at Turner Newall say they are 'disappointed' after talks with stricken parent firm Federal Mogul.	Business
Tiêu đề : The Race is On: Second Private Team Sets Launch Date for Human Spaceflight (SPACE.com) **Nội dung :** SPACE.com - TORONTO, Canada -- A second team of rocketeers competing for the 36.10 million Ansari X Prize, a contest for privately funded suborbital space flight, has officially announced the first launch date for its manned rocket.	Sci/Tech

Bài toán tương tự nhưng khác với bài toán xây dựng chủ đề (topic modeling), trong đó không có 1 tập chủ đề xác định mà thay vào đó lời giải bao gồm 1 tập các chủ đề được tạo ra với số lượng cho trước.

Lời giải cần thỏa mãn 1 số yêu cầu:

- Có độ chính xác cao.
- Có khả năng thích ứng cao với các văn bản mới.
- Thời gian dự đoán tương đối tốt.

1.2. Một số cách tiếp cận phổ biến

Cách tiếp cận phổ biến nhất với bài toán là sử dụng các giải thuật dựa trên xác suất, với điển hình là giải thuật Naive Bayes [1]. SVM[2] cũng là 1 giải pháp thường được sử dụng.

Một cách tiếp cận thứ hai sử dụng các kĩ thuật xử lý ngôn ngữ tự nhiên như các bộ phân tích cú pháp [3].

Một số nghiên cứu cũng có sử dụng đến mạng neuron và Deep Learning.

1.3. Hướng giải quyết

Nhóm quyết định sử dụng một mô hình mạng neuron để giải quyết bài toán, với ưu điểm là khả năng chịu nhiễu và làm việc với dữ liệu thô tốt, đồng thời có thời gian chạy ngắn sau giai đoạn huấn luyện. Cụ thể, mô hình sẽ tích hợp mạng hồi quy để tiến hành phân loại.

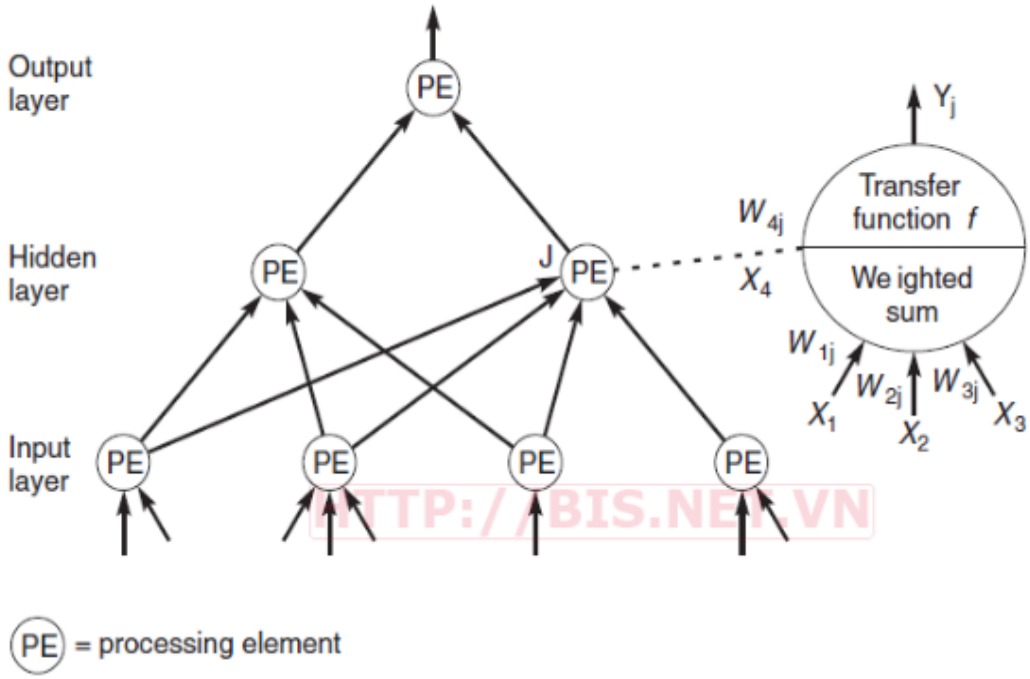
Để cài đặt hệ thống, nhóm sử dụng ngôn ngữ lập trình Python và Keras [4], một thư viện Deep Learning mã nguồn mở. Keras cho phép sử dụng 1 trong 2 nền tảng tính toán là TensorFlow (Google) và Theano (cả 2 đều hỗ trợ tăng tốc bằng GPU), cài đặt sẵn các dạng neuron dưới dạng các lớp có tính module hóa cao và độ tối ưu tốt. Quá trình cài đặt thuật toán do đó khá ngắn gọn, cho phép nhóm tập trung vào thiết kế mô hình và xử lý dữ liệu.

2. Mạng neuron - Mạng hồi quy

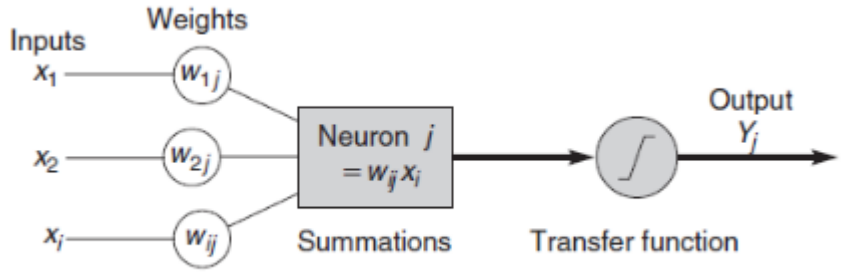
2.1. Mạng neuron (Artificial Neural Network – ANN)

Mạng nơron nhân tạo (Artificial Neural Network- ANN) là mô hình xử lý thông tin được mô phỏng dựa trên hoạt động của hệ thống thần kinh của sinh vật, bao gồm số lượng lớn các Neural được gắn kết để xử lý thông tin. ANN giống như bộ não con người, được học bởi kinh nghiệm (thông qua huấn luyện), có khả năng lưu giữ những kinh nghiệm hiểu biết (tri thức) và sử dụng những tri thức đó trong việc dự đoán các dữ liệu chưa biết (unseen data).

Các ứng dụng của mạng neuron được sử dụng trong rất nhiều lĩnh vực như điện, điện tử, kinh tế, quân sự,… để giải quyết các bài toán có độ phức tạp và đòi hỏi có độ chính xác cao như điều khiển tự động, khai phá dữ liệu, nhận dạng,…



Hình 1: Kiến trúc cơ bản của mạng neuron



Hình 2: Quá trình xử lý thông tin của 1 neuron

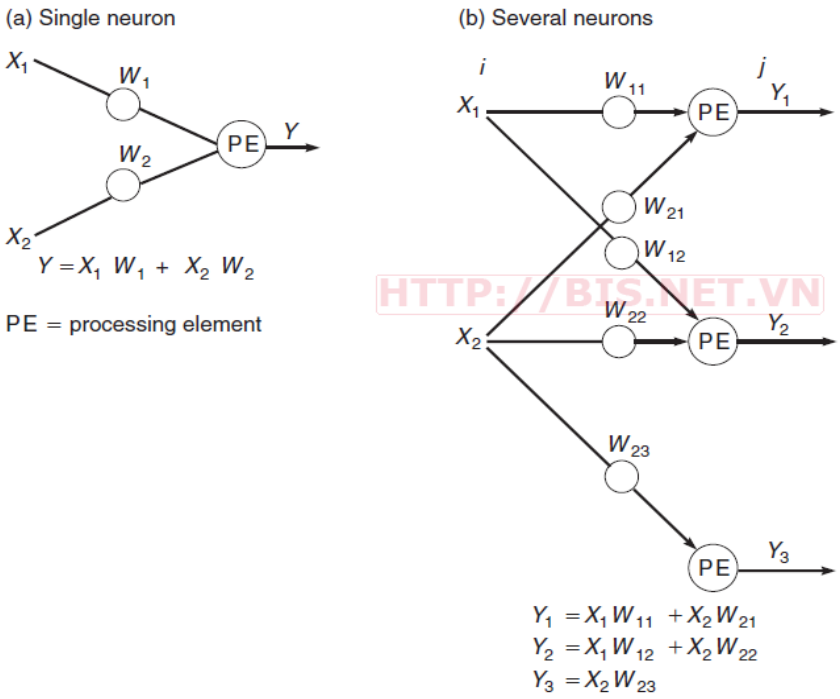
- **Inputs:** Mỗi Input tương ứng với 1 thuộc tính (attribute) của dữ liệu (patterns). Ví dụ như trong ứng dụng của ngân hàng xem xét có chấp nhận cho khách hàng vay tiền hay không thì mỗi Input là một thuộc tính của khách hàng như thu nhập, nghề nghiệp, tuổi, số con,…
- **Output:** Kết quả của một ANN là một giải pháp cho một vấn đề, ví dụ như với bài toán xem xét chấp nhận cho khách hàng vay tiền hay không thì output là yes (cho vay) hoặc no (không cho vay).
- **Connection Weights (Trọng số liên kết) :** Đây là thành phần rất quan trọng của một ANN, nó thể hiện mức độ quan trọng (độ mạnh) của dữ liệu đầu vào đối với quá trình xử lý thông tin (quá trình chuyển đổi dữ liệu từ Layer này sang layer khác). Quá trình học (Learning Processing) của ANN thực ra là quá trình điều chỉnh các trọng số (Weight) của các input data để có được kết quả mong muốn.
- **Summation Function (Hàm tổng):** Tính tổng trọng số của tất cả các input được đưa vào mỗi neuron(phần tử xử lý PE). Hàm tổng của một neurondối với n input được tính theo công thức sau:

$$Y = \sum_{i=1}^{n} X_i W_i$$

Hàm tổng đối với nhiều neuron trong cùng một Layer:[5]

$$Yj = \sum_{i=1}^{n} X_i W_{ij}$$

Giải thuật phổ biến để huấn luyện các mạng neuron có giám sát là lan truyền ngược (back-propagation).



Giải thuật học lan truyền ngược tìm kiếm một vectơ các trọng số (weights vector) giúp cực tiểu hóa lỗi tổng thể của hệ thống đối với tập học. [6]

Giải thuật BP bao gồm 2 giai đoạn (bước):

- Giai đoạn lan truyền tiến tín hiệu (Signal forward): các tín hiệu đầu vào (vectơ các giá trị đầu vào) được lan truyền tiến từ tầng đầu vào đến tầng đầu ra (đi qua các tầng ẩn)
- Giai đoạn lan truyền ngược lỗi (Error backward):
  - Căn cứ vào giá trị đầu ra mong muốn của vectơ đầu vào, hệ thống tính toán giá trị lỗi .
  - Bắt đầu từ tầng đầu ra, giá trị lỗi được lan truyền ngược qua mạng, từ tầng này qua tầng khác (phía trước), cho đến tầng đầu vào.
  - Việc lan truyền ngược lỗi (error back-propagation) được thực hiện thông qua việc tính toán (một cách truy hồi) giá trị gradient cục bộ của mỗi nơ-ron

Lan truyền tiến:

Đối với mỗi ví dụ học x:

Vectơ đầu vào x được lan truyền từ tầng đầu vào đến tầng đầu ra

Mạng sẽ sinh ra một giá trị đầu ra thực tế (actual output) Out (là một vectơ của các giá trị Outi , i=1..n).

Đối với một vectơ đầu vào x, một nơ-ron zq ở tầng ẩn sẽ nhận được giá trị đầu vào tổng thể (net input) bằng:

$$Net_q = \sum_{j=1}^{m} w_{qj} x_j$$

Và sinh ra một giá trị đầu ra (cục bộ) bằng:

$$Out_q = f(Net_q) = f\left(\sum_{j=1}^{m} w_{qj} x_j\right)$$

trong đó f(.) là hàm tác động (activation function) của nơ-ron zq

Giá trị đầu vào tổng thể (net input) của nơ-ron yi ở tầng đầu ra:

$$Net_i = \sum_{q=1}^{l} w_{iq} Out_q = \sum_{q=1}^{l} w_{iq} f\left(\sum_{j=1}^{m} w_{qj} x_j\right)$$

Nơ-ron yi sinh ra giá trị đầu ra (là một giá trị đầu ra của mạng)

$$Out_i = f(Net_i) = f\left(\sum_{q=1}^{l} w_{iq} Out_q\right) = f\left(\sum_{q=1}^{l} w_{iq} f\left(\sum_{j=1}^{m} w_{qj} x_j\right)\right)$$

Vectơ các giá trị đầu ra Outi (i=1..n) chính là giá trị đầu ra thực tế của mạng, đối với vectơ đầu vào x.

Lan truyền ngược:

Đối với mỗi ví dụ học x:

Các tín hiệu lỗi (error signals) do sự khác biết giữa giá trị đầu ra mong muốn d và giá trị đầu ra thực tế Out được tính toán

Các tín hiệu lỗi này được lan truyền ngược (back-propagated) từ tầng đầu ra tới các tầng phía trước, để cập nhật các trọng số (weights)

Để xét các tín hiệu lỗi và việc lan truyền ngược của chúng, cần định nghĩa một hàm đánh giá lỗi:

$$E(w) = \frac{1}{2}\sum_{i=1}^{n}(d_i - Out_i)^2 = \frac{1}{2}\sum_{i=1}^{n}[d_i - f(Net_i)]^2$$

$$= \frac{1}{2}\sum_{i=1}^{n}\left[d_i - f\left(\sum_{q=1}^{l} w_{iq} Out_q\right)\right]^2$$

Theo phương pháp gradient-descent, các trọng số của các liên kết từ tầng ẩn tới tầng đầu ra được cập nhật bởi:

$$\Delta w_{iq} = -\eta \frac{\partial E}{\partial w_{iq}}$$

Sử dụng quy tắc chuỗi đạo hàm đối với $\partial E/\partial w_{iq}$, ta có:

$$\Delta w_{iq} = -\eta \left[\frac{\partial E}{\partial Out_i}\right]\left[\frac{\partial Out_i}{\partial Net_i}\right]\left[\frac{\partial Net_i}{\partial w_{iq}}\right] = \eta\left[d_i - Out_i\right]\left[f'(Net_i)\right]\left[Out_q\right] = \eta\delta_i Out_q$$

δ_i là tín hiệu lỗi (error signal) của nơ-ron y_i ở tầng đầu ra:

$$\delta_i = -\frac{\partial E}{\partial Net_i} = -\left[\frac{\partial E}{\partial Out_i}\right]\left[\frac{\partial Out_i}{\partial Net_i}\right] = \left[d_i - Out_i\right]\left[f'(Net_i)\right]$$

Để cập nhật các trọng số của các liên kết từ tầng đầu vào tới tầng ẩn, chúng ta cũng áp dụng phương pháp gradient-descent và quy tắc chuỗi đạo hàm:

$$\Delta w_{qj} = -\eta \frac{\partial E}{\partial w_{qj}} = -\eta \left[\frac{\partial E}{\partial Out_q}\right]\left[\frac{\partial Out_q}{\partial Net_q}\right]\left[\frac{\partial Net_q}{\partial w_{qj}}\right]$$

Từ công thức tính hàm lỗi E(w), ta thấy rằng mỗi thành phần lỗi $(d_i - y_i)$ (i=1..n) là một hàm của Out_q

$$E(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{n}\left[d_i - f\left(\sum_{q=1}^{l} w_{iq} Out_q\right)\right]^2$$

Áp dụng quy tắc chuỗi đạo hàm, ta có:

$$\Delta w_{qj} = \eta \sum_{i=1}^{n}\left[\left(d_i - Out_i\right) f'(Net_i) w_{iq}\right] f'(Net_q) x_j$$

$$= \eta \sum_{i=1}^{n}\left[\delta_i w_{iq}\right] f'(Net_q) x_j = \eta \delta_q x_j$$

$$\delta_q = -\frac{\partial E}{\partial Net_q} = -\left[\frac{\partial E}{\partial Out_q}\right]\left[\frac{\partial Out_q}{\partial Net_q}\right] = f'(Net_q)\sum_{i=1}^{n}\delta_i w_{iq}$$

Trong đó Net_q là đầu vào tổng thể (net input) của nơ-ron z_q ở tầng ẩn.

2.2. Mạng hồi quy (Recurrent Neural Network – RNN)

Mạng hồi quy là 1 mô hình mạng neuron đặc biệt, trong đó đầu vào của mỗi lớp RNN được chia thành các bước rời rạc (time-step). Mỗi neuron không xử lý toàn bộ đầu vào mà xử lý lần lượt trên từng time-step, với trạng thái của bước sau phụ thuộc vào bước trước. Đặc điểm này cho phép mạng hồi quy phản ánh quan hệ phụ thuộc giữa các phần tử trong một chuỗi, ví dụ giữa các từ trong 1 câu hay các khung hình trong 1 video. RNN do đó được thường được sử dụng để học trên các luồng input tuyến tính như văn bản, tín hiệu,... Một trong những thành công đầu tiên trong ứng dụng RNN là các mô hình dịch máy.

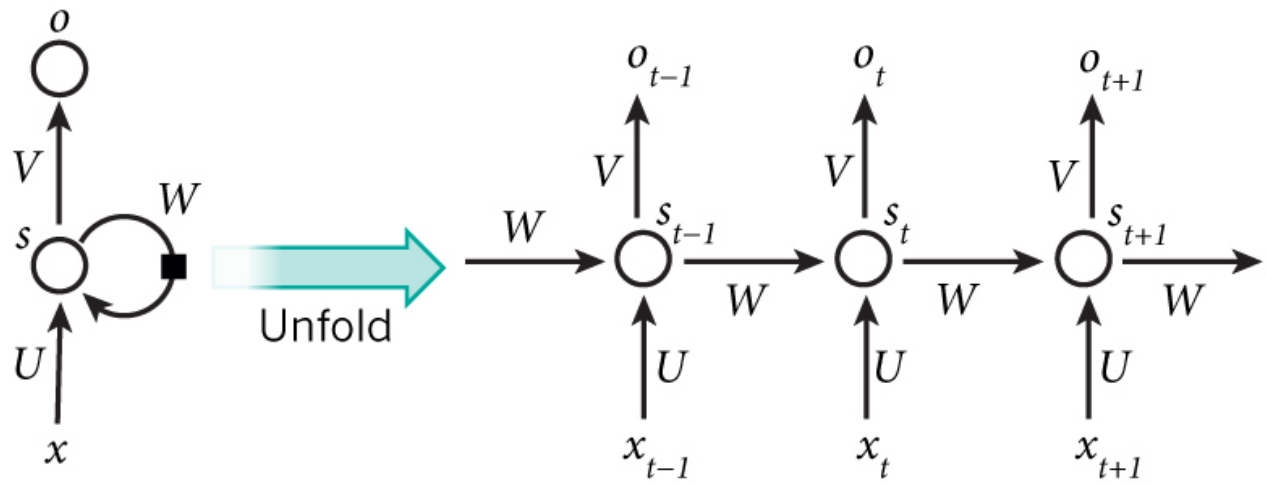
Một neuron (hay cell) của mạng hồi quy có 2 input và 2 output. Input bao gồm giá trị của ví dụ tại time-step tương ứng x_i và 1 vector "trạng thái" (hay hidden state) được truyền xuyên suốt các time-step. Giá trị ban đầu của vector này bằng 0. Tương tự như neuron thông thường, input của ví dụ được nhân với vector trọng số U. Hidden state cũng được nhân với trọng số riêng W. 2 giá trị thu được được cộng vào nhau và đưa qua 1 hàm kích hoạt (hidden activation) để được giá trị s_i. Giá trị này tiếp tục được nhân với trọng số ra V, sau đó lại được đưa qua hàm kích hoạt thứ hai (output activation) để được o_i. o_i trở thành đầu ra thấy được của time-step, còn s_i trở thành hidden state đưa vào time-step tiếp theo.

$$s_i = \sigma_g(Ux_i + W s_{i-1} + b_s)$$
$$o_i = \sigma_h(V s_i + b_o)$$

trong đó σ_g là hàm kích hoạt ẩn, σ_h là hàm kích hoạt đầu ra, b là các trọng số bias.

Trong cài đặt, σ_g thường được chọn là hàm tanh() hoặc sigmoid(). σ_h được chọn dựa trên output mong muốn của neuron.

Trên lý thuyết, một neuron có thể nhận vào các ví dụ có độ dài bất kỳ. Tuy nhiên, để tăng tốc độ tính toán, ta thường muốn gộp các ví dụ thành 1 ma trận và xử lý theo batch. Do đó khi cài đặt các ví dụ thường được thêm (pad) các kí tự đặc biệt để đạ t cùng độ dài.

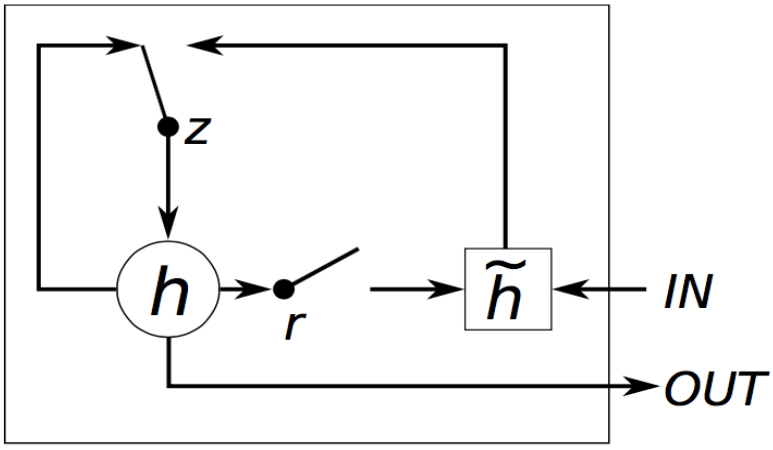


Hình 3: Mô hình 1 RNN Cell

Các vector trọng số U, W, V là mục tiêu học của mỗi neuron và là như nhau với mỗi time-step trong quá trình dự đoán. [7]

Để học các tham số của U, W, V, RNN sử dụng 1 biến thể của thuật toán lan truyền ngược là lan truyền ngược qua thời gian (Back-propagation Through Time – BPTT). BPTT coi các xử lý ở các time-step là các neuron có chung trọng số, hay nói cách khác, ta trải các time-step thành 1 mạng neuron. Sau đó, tương tự như back-propagation, ta áp dụng quy tắc chuỗi đạo hàm trên từng time-step, với lỗi truyền từ time-step cuối đến time-step đầu. Tuy nhiên, vì các trọng số được chia sẻ giữa các bước, nên ta lấy tổng gradient ở mỗi bước và update trọng số sau khi đi hết time-step đầu.

Trong Deep Learning, ta thường sử dụng 2 biến thể của mạng hồi quy là Long Short-Term Memory (LSTM) và Gated Recurrent Unit (GRU). Phần sau sẽ tập trung giải thích cơ chế hoạt động của GRU - mô hình được sử dụng trong báo cáo.



Hình 4: Mô hình 1 GRU Cell

Khác với 1 neuron hồi quy thông thường, neuron của GRU có output bằng hidden state (được kí hiệu h). Giá trị này được tính dựa trên input đầu vào của time-step, giá trị đầu ra của time-step trước và trọng số của 2 "cổng" update z và reset r. Công thức tính của h như sau:

$$z_i = \sigma_g(W_z x_i + U_z h_{i-1} + b_z)$$

$$r_i = \sigma_g(W_r x_i + U_r h_{i-1} + b_r)$$

$$h_i = z_i \circ h_{i-1} + (1 - z_i) \circ \sigma_h(W_h x_i + U_h(r_i \circ h_{i-1}) + b_h)$$

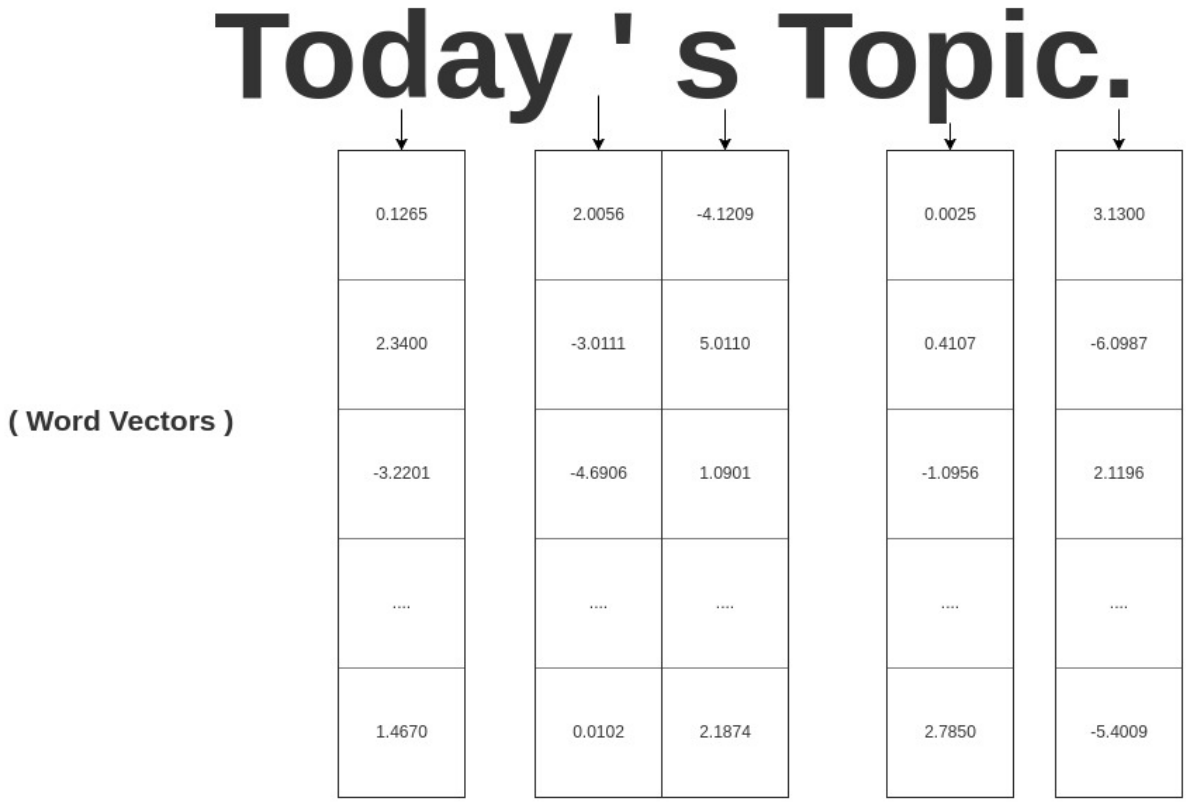
Cổng update z quyết định lượng thông tin trong hidden state được lưu giữ lại (z tăng → $z_i \circ h_{i-1}$ tăng). Ngược lại, cổng reset quyết định lượng thông tin trong hidden state được dùng để kết hợp với input. Nếu z có giá trị lớn và xấp xỉ 1 (σ_g thường mặc định là sigmoid), ta gần như giữ nguyên hidden state và bỏ qua input đầu vào. Đặc tính này sẽ ứng với các neuron học được các phụ thuộc "dài", giữa các từ đứng xa nhau trong câu. Ngược lại, với z nhỏ và r nhỏ, neuron gần như reset lại hidden state cũ, tuy nhiên lại kết hợp nó với input của time-step. Đây là các neuron học được các phụ thuộc "ngắn", giữa các từ đứng gần nhau trong câu.

Trong một mạng GRU, mỗi neuron sẽ học được một dạng dependency nhất định trong các trọng số W và U. Các thuộc tính được sinh ra do đó phản ánh rất tốt cả quan hệ ngữ nghĩa ngắn (phủ định, khẳng định,...) và dài (chủ đề, ngữ cảnh,...).

3. Thiết kế, cài đặt

3.1. Biểu diễn ví dụ

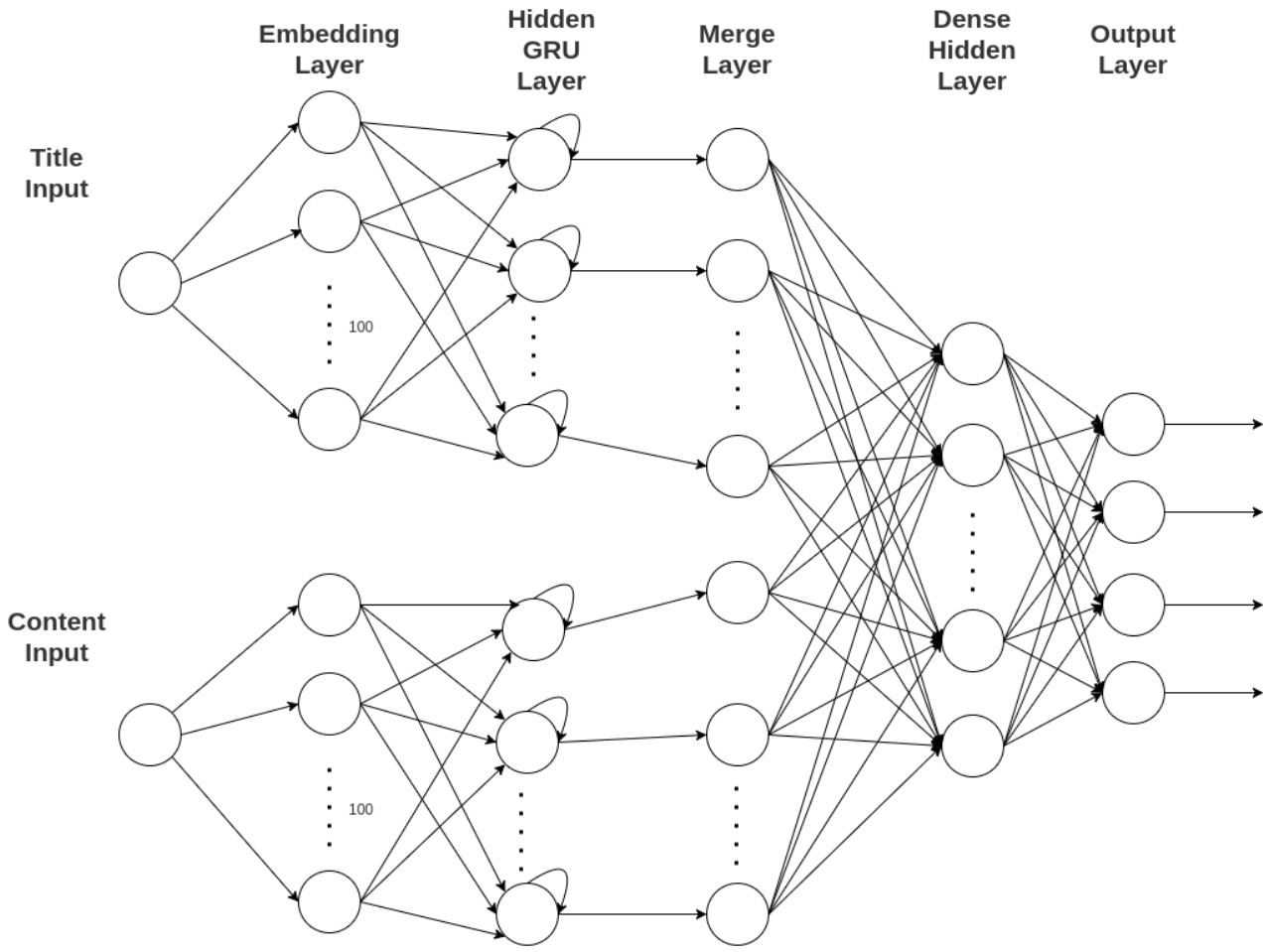
Đa số các mô hình phân loại văn bản biểu diễn một văn bản như một "túi từ" (bag-of-words), trong đó chỉ theo dõi số lượng của các từ trong từ điển trong văn bản. Tuy nhiên, để tận dụng khả năng của RNN, ta biểu diễn mỗi ví dụ dưới dạng 1 chuỗi các vector từ, trong đó thứ tự của các từ trong văn bản được duy trì.



Hình 5: Biểu diễn ví dụ dưới dạng các vector từ.

Các vector từ có thể phản ánh thứ tự của từ trong từ điển (1 vector hot-coded tại vị trí ứng với từ) hoặc được học bằng các thuật toán biểu diễn từ như Glove, word2vec hay fastText. Đồ án sử dụng biểu diễn 100 chiều học bằng thuật toán Glove[8] (Stanford) trên dữ liệu Wikipedia.

3.2. Kiến trúc mạng neuron



Hình 6: Kiến trúc mạng, với số chủ đề = 4.

Mạng neuron nhận vào 2 input: tiêu đề và nội dung của văn bản. Thay vì gộp chung vào thành 1 xâu, ta dùng tiêu đề như 1 tham số thứ 2, do tiêu đề thường phản ánh ngắn gọn nội dung của văn bản và có giá trị cao trong phân loại. Mỗi từ trong xâu đưa vào được biểu diễn dưới dạng 1 vector (đồ án sử dụng độ dài 100) sinh bởi 1 lớp Embedding. Lớp này thực chất là 1 tập các vector từ tương ứng với từ điển, $E(i) = W_i$, với W là ma trận trọng số của E. Trọng số ban đầu của E được lấy từ tập con của biểu diễn từ Glove và cũng được cập nhật trong quá trình huấn luyện.

Để thực hiện xử lý batch, ta buộc phải cố định độ dài của mỗi xâu. Do đó, nhóm sử dụng 1 ký tự đặc biệt MASK_TOKEN có thứ tự 0 để thể hiện kí tự rỗng.

Các vector được đưa vào 1 lớp mạng GRU và xử lý lần lượt như phần 2.2. Hàm kích hoạt output cho các neuron GRU là tanh(), hàm kích hoạt ẩn là sigmoid(). Khi gặp MASK_TOKEN, GRU sẽ tự động bỏ qua từ mà không thực hiện tính toán. Output của từ cuối cùng trong xâu được dùng làm vector biểu diễn cho cả xâu.[1]

1 Một cách tiếp cận khác để tạo biểu diễn xâu là sử dụng 1 vector trung bình của tất cả các từ trong câu sau khi đưa qua 1 lớp trọng số. Tuy nhiên phương pháp này trở nên phức tạp khi phải xử lý các

Sau khi có được 2 vector nội dung và tiêu đề, ta thực hiện ghép chúng lại để tạo thành vector văn bản D. Có thể coi vector này là 1 tập các feature mới được học bởi mạng GRU. Lớp GRU được dùng như một lớp mã hóa (encoder), tìm ra các đặc điểm cần cho phân loại.

Bài toán trở thành phân loại các văn bản với đặc điểm D. Ta sử dụng 1 mạng neuron nhân tạo để học cách phân loại các đặc điểm này, các lớp đều có hàm kích hoạt sigmoid (hệ thống cho phép tùy chỉnh độ sâu và kích cỡ đầu ra của từng lớp mạng). Cuối cùng, hệ thống coi mỗi văn bản chỉ có 1 chủ đề, do đó ta lấy đầu ra là 1 phân bố xác suất sinh bởi hàm kích hoạt softmax:

$$softmax(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

với z là vector độ dài K, j = 1...K

Softmax trả về 1 vector có tổng bằng 1, với tỉ lệ tương đối giữa các phần tử được bảo toàn. Do đó có thể coi kết quả thu được là 1 phân bố xác suất đúng của mỗi chủ đề trên văn bản đầu vào. Chủ đề có xác suất cao nhất được chọn làm chủ đề của văn bản.

Giá trị trọng số ở mỗi lớp được khởi tạo ngẫu nhiên trong khoảng giá trị được đề xuất bởi Glorot và Bengio năm 2010 [9]:

$$W \in \left[-\sqrt{\frac{6}{n_{inp} + n_{out}}}, \sqrt{\frac{6}{n_{inp} + n_{out}}} \right]$$

3.3. Tập dữ liệu

Đồ án sử dụng 3 tập dữ liệu: AG-News có độ lớn 120,000 (train) và 7,600 (test) với 4 nhãn, BBC có độ lớn 1,777 (train) và 447 (test) với 5 nhãn, và Reuters có độ lớn 7,769 (train) và 3,018 (test) với 86 nhãn.

Nhóm quy định định dạng chuẩn của các dataset theo dataset AG-News, trong đó mỗi dataset cần có 3 file:

- train.csv chứa các ví dụ của tập train, lưu trong 3 cột nhãn (số nguyên bắt đầu từ 1), tiêu đề và nội dung.
- test.csv chứa các ví dụ của tập test với định dạng tương tự.
- classes.txt chứa tên các nhãn tương ứng với thứ tự trong các file CSV, mỗi nhãn trên 1 dòng.

MASK_TOKEN được bỏ qua.

Dễ nhận thấy rằng tập AG-News có tỉ lệ rất không cân đối giữa 2 tập train và test. Do đó nhóm thực hiện phân bố lại tập này tuân theo stratified sampling và thu được 1 dataset với .

Tập dữ liệu Reuters được thêm vào trong giai đoạn cuối của đồ án và có nhiều chủ đề cho mỗi đoạn. Để tương thích với hệ thống, nhóm buộc phải lấy chủ đề đầu tiên của mỗi đoạn làm nhãn của đoạn đó. Điều này làm cho một số nhãn có số ví dụ rất ít hoặc không có ví dụ nào trong tập test. Đây là 1 hạn chế nhóm chưa kịp khắc phục trong tập dữ liệu và có ảnh hưởng đến mô hình học trên tập này.

Các tập huấn luyện được tách validation tuân theo stratified sampling với tỉ lệ 80:20 khi huấn luyện.

3.4. Quá trình huấn luyện

Hệ thống được huấn luyện offline trên 1 tập dữ liệu đã xác định trước. Bước đầu, các file đầu vào (tập train, test và tên các nhãn) được nạp vào bộ nhớ cùng với **1 phần** của tập vector Glove (nhóm sử dụng 40,000 từ đầu tiên trên tổng số 400,000). Tập vector được chèn thêm vector của kí tự MASK_TOKEN (một vector 0) và UNKNOWN_TOKEN tượng trưng cho các từ không có trong từ điển (một vector có giá trị ngẫu nhiên).

Các đoạn đi qua quá trình tách từ (tokenization) được thực hiện bởi thư viện Natural Language Toolkit (NLTK)[10]. Lý do ta không thực hiện tách từ theo dấu trắng là để tách các các dấu liền với từ (VD 'chào.') thành 1 từ riêng biệt. Sau đó, mỗi từ được thay thế bằng chỉ số của vector tương ứng trong tập vector. Hệ thống cũng in ra tỉ lệ UNKNOWN_TOKEN để tránh trường hợp từ điển quá nhỏ (tỉ lệ trong các huấn luyện của nhóm vào khoảng 2-5%). Khi kết thúc giai đoạn nạp dữ liệu, ta thu được một dãy vector từ, các ma trận số nguyên Xt_ và Xc_ (tiêu đề và nội dung) với kích cỡ n.s (n: số ví dụ trong tập, s: độ dài tối đa của 1 xâu) và vector nhãn y cho mỗi tập dữ liệu train, validation và test.

Sau khi nạp dữ liệu, hệ thống thực hiện khởi tạo mạng neuron và thiết lập cấu hình huấn luyện, bao gồm thuật toán huấn luyện và tốc độ học. Thuật toán huấn luyện được nhóm sử dụng là RMSProp [11]. RMSProp là 1 thuật toán học có thích nghi được sử dụng phổ biến với các mạng hồi quy. Tốc độ học ban đầu được nhóm lựa chọn là 0.001.

Trong mỗi lần huấn luyện, thuật toán chạy trên từng mini-batch của tập train. Sau mỗi lần đi hết tập train (1 epoch), ta đánh giá hàm mục tiêu và độ chính xác trên tập train và validation. Huấn luyện sẽ dừng lại khi:

- Hàm mục tiêu trên tập train hội tụ (giá trị bắt đầu tăng) hoặc
- Hàm mục tiêu trên tập validation bắt đầu phân kì, được nhóm quy ước là khi giá

trị hàm không cải thiện sau 5 vòng liên tiếp.

Sau mỗi vòng, mô hình được lưu lại chỉ khi có cải thiện hàm mục tiêu trong lần huấn luyện và độ chính xác validation trong các lần huấn luyện trước.

Khi kết thúc một lần huấn luyện, ta đánh giá độ chính xác của mô hình trên tập test. Các đồ thị hàm mục tiêu và độ chính xác cũng được lưu trong thư mục của mô hình. Đồ thị được vẽ sử dụng thư viện Matplotlib[12].

Với mỗi tập dữ liệu, nhóm thực hiện tối ưu theo số lượng neuron ở lớp kết nối đầy đủ bắt đầu từ số neuron bằng 1000 giảm dần. Các tham số còn lại được giữ cố định bao gồm:

- Tốc độ học ban đầu (LEARNING RATE): 0.001
- Số epoch tối đa (N_EPOCH): 200. Là tham số bắt buộc để huấn luyện sử dụng Keras. Nhóm chọn 1 giá trị rất cao để chắc chắn mô hình sẽ đạt điều kiện dừng trước khi hết số vòng lặp.
- Kích cỡ batch (BATCH_SIZE): 100.
- Kích cỡ từ điển (VOCABULARY_SIZE): 40,000 (60,000 cho tập Reuters). Tỉ lệ các từ không biết trong khoảng 3-6%.
- Độ dài tiêu đề và độ dài nội dung: tùy thuộc vào bộ dữ liệu.
  - BBC: 50, 400.
  - Reuters: 30, 150.
  - AG-News: 30, 50.
- Số neuron GRU của tiêu đề (TITLE_OUTPUT): 200.
- Số neuron GRU của nội dung (CONTENT_OUTPUT): 400.

3.5. Cài đặt chi tiết

3.5.1. Đọc và tiền xử lý dữ liệu

Nhóm cài đặt các hàm đọc và tiền dữ liệu trong script data_utils.py, bao gồm:

- load_embedding() nạp vào các vector từ Glove và đưa vào 1 ma trận. Đồng thời hàm cũng trả về 1 mapping từ-số hiệu (word_to_index) và số hiệu-từ (index-to-word) để thực hiện các chuyển đổi.
- read_csv() đọc file csv được truyền vào, thực hiện tách từ bằng thư viện NLTK và trả về danh sách các ví dụ. Mỗi ví dụ được biểu diễn dưới dạng 1 từ điển với 3 khóa 'class', 'title' và 'content'.

- strat_samples() nhận vào 1 danh sách văn bản được sinh bởi read_csv() và tách thành 2 tập tuân thủ theo stratified sampling với tỉ lệ bất kỳ.
- get_mat() thực hiện chuyển đổi danh sách các văn bản thành các ma trận để đưa vào huấn luyện.
- load_generic() thực hiện lần lượt các thao tác trên với một thư mục bất kỳ.
- load_ag_news(), load_bbc() và load_reuters() là các hàm bao cho load_generic().

3.5.2. Mô hình phân loại

Nhóm thiết lập 1 lớp Classifier để chứa mô hình phân loại. Ngoài mạng neuron và các trọng số, Classifier cần lưu các thông tin như từ điển, tên các nhãn,... để tiến hành phân loại. Các tham số như kích cỡ đầu ra, độ sâu mạng,... đều có thể được điều chỉnh. Hàm khởi tạo của lớp được định nghĩa như sau:

```
def __init__(self, word_vec, word_to_index, index_to_word, classes,
             title_output=128, content_output=512,
             dense_neurons=(1024, 256,), title_len=50, content_len=2000,
             weights=None, directory='.'):
    self.directory = directory
    self.word_to_index = word_to_index
    self.index_to_word = index_to_word
    self.title_len = title_len
    self.content_len = content_len
    self.word_vec = word_vec
    self.classes = classes
    self.title_output = title_output
    self.content_output = content_output
    self.dense_neurons = dense_neurons
    self.gru_regularize = gru_regularize
    self.dense_regularize = dense_regularize

    # Encode document's title
    title_inp = Input(shape=(title_len,), name='Title_Input')
    title_embed = Embedding(input_dim=np.size(word_vec, 0),
                            output_dim=np.size(word_vec, 1),
                            weights=[word_vec], mask_zero=True,
                            name='Title_Embedding')
    self.t_encoder = Sequential(name='Title_Encoder')
    self.t_encoder.add(title_embed)
    self.t_encoder.add(GRU(title_output, name='Title_GRU', consume_less='mem',
                           W_regularizer=WeightRegularizer(l2=gru_regularize)))
    title_vec = self.t_encoder(title_inp)
    # Encode document's content
    content_inp = Input(shape=(content_len,), name='Content_Input')
    content_embed = Embedding(input_dim=np.size(word_vec, 0),
                              output_dim=np.size(word_vec, 1),
                              weights=[word_vec], mask_zero=True,
                              name='Content_Embedding')
    self.c_encoder = Sequential(name='Content_Encoder')
    self.c_encoder.add(content_embed)
    self.c_encoder.add(GRU(content_output, name='Content_GRU',
```

```
                            consume_less='mem'))
    content_vec = self.c_encoder(content_inp)
    # Merge vectors to create output
    doc_vec = merge(inputs=[title_vec, content_vec], mode='concat')
    self.decoder = Sequential(name='Decoder')
    self.decoder.add(Dense(dense_neurons[0],
                    input_shape=(title_output + content_output,),
                    name='Dense_0', activation='hard_sigmoid'))
    for i, n in enumerate(dense_neurons[1:]):
        self.decoder.add
          (Dense(n, activation='hard_sigmoid', name='Dense_%s' % (i + 1)))

    self.decoder.add(Dense(len(classes), activation='softmax',
                    name='Dense_Output'))
    output = self.decoder(doc_vec)
    self.model = Model(input=[title_inp, content_inp], output=output,
                       name='Model')
    if weights is not None:
        self.model.load_weights(weights)
```

Chức năng dự đoán chủ đề được cài đặt dưới dạng 1 phương thức của Classifier, trả về nhãn được dự đoán cùng với phân bố xác suất của từng chủ đề:

```
def predict(self, title, content, verbose=0):
    t = nltk.word_tokenize(title.lower())
    Xt = [self.word_to_index[word] if word in self.word_to_index
          else self.word_to_index[utils.UNKNOWN_TOKEN] for word in t]
          [:self.title_len]
    c = nltk.word_tokenize(content.lower())
    Xc = [self.word_to_index[word] if word in self.word_to_index
          else self.word_to_index[utils.UNKNOWN_TOKEN] for word in c]
          [:self.content_len]
    Xt = utils.pad_vec(Xt, self.title_len)
    Xc = utils.pad_vec(Xc, self.content_len)
    probs = self.model.predict([Xt, Xc], verbose=verbose)
    pred = np.argmax(probs)
    return pred, probs
```

Keras hỗ trợ việc định nghĩa các hàm callback (hàm được gọi tại các thời điểm nhất định trong quá trình huấn luyện). Nhóm cài đặt 2 callback để lưu mô hình và đánh giá trên tập test.

```
class SaveCallback(Callback):
    def __init__(self, classifier, prev_val_acc):
        self.best_loss = 1000.0
        self.best_val_loss = 1000.0
        self.best_val_acc = prev_val_acc
        self.classifier = classifier
        super(SaveCallback, self).__init__()
    def on_epoch_end(self, epoch, logs=None):
        print()
        if logs['val_loss'] <= self.best_val_loss and logs['loss'] <=
           self.best_loss:
            self.classifier.save()
```

```
            self.best_val_loss = logs['val_loss']
            self.best_loss = logs['loss']
            self.best_val_acc = logs['val_acc']
            self.classifier.log('Save point:\nLoss: %s\tAccuracy: %s\n' %
                                (logs['loss'], logs['acc']) +
                                'Validation loss: %s\tValidation accuracy: %s\n'
                               % (logs['val_loss'], logs['val_acc']), out=False)
        elif logs['val_loss'] > self.best_val_loss:
            print('No improvement on validation loss. Skipping save...')
        elif logs['loss'] > self.best_loss:
            print('No improvement on loss. Skipping save...')

class TestCallback(Callback):
    def __init__(self, classifier, Xt, Xc, y):
        self.classifier = classifier
        self.Xt = Xt
        self.Xc = Xc
        self.y = y
        super(TestCallback, self).__init__()
    def on_train_end(self, logs=None):
        print('Evaluating on test set...')
        result = self.classifier.model.evaluate([self.Xt, self.Xc], self.y,
                                                batch_size=10)
        self.classifier.log('GRU_Regularizer: %s --- Dense Regularizer: %s' %
                            (self.classifier.gru_regularize,
self.classifier.dense_regularize))
        self.classifier.log('Test loss: %s --- Test acc: %s\n' % (result[0],
                            result[1]))
```

3.5.3. Huấn luyện và chạy mô hình

Phương thức compile() và train() của Classifier cho phép biên dịch và huấn luyện mạng neuron chứa trong đối tượng.

Khái niệm biên dịch của Keras được thừa hưởng từ nền tảng tính toán (backend) Theano và TensorFlow. Thao tác này thực hiện xây dựng và tối ưu đồ thị tính toán của mạng, để từ đó có thể thực hiện tính năng tính gradient và tối ưu tự động. Biên dịch cũng quyết định thuật toán tối ưu và tốc độ học được sử dụng.

Huấn luyện nhận vào các ví dụ và nhãn được sử dụng (dưới dạng các mảng NumPy [13] [14]) cùng với các cài đặt huấn luyện để huấn luyện mạng. compile() buộc phải được thực hiện trước train().

Script train.py được sử dụng để huấn luyện 1 mô hình mới dựa trên các cấu hình mạng.

Mô hình đã huấn luyện được nạp và sử dụng trong 2 demo của hệ thống. Việc dự đoán (phương thức predict()) không cần thực hiện compile().

3.5.4.Cấu hình huấn luyện

Để tiện lợi trong việc điều chỉnh cấu hình huấn luyện, nhóm sử dụng 1 file cấu hình settings.py để xác định các tham số mạng.

```
LEARNING_RATE = 0.001
N_EPOCH = 200
BATCH_SIZE = 100
VOCABULARY_SIZE = 40000
TITLE_LEN = 50
CONTENT_LEN = 400
TITLE_OUTPUT = 200
CONTENT_OUTPUT = 400
DENSE_NEURONS = [500]
DATASET = 'bbc'
```

3.5.5. Đánh giá mô hình

Để đánh giá chi tiết về độ chính xác của 1 mô hình đã huấn luyện, nhóm sử dụng phương thức evaluate() của Classifier, gọi qua evaluate.py để tính Precision, Recall và điểm F1 của từng nhãn trên tập dữ liệu test cũng như các giá trị trung bình. Ngoài ra, hệ thống cũng thực hiện đánh giá hàm mục tiêu và độ chính xác trong suốt quá trình huấn luyện.

3.5.6. Các hàm tiện ích

Nhóm xây dựng 1 số hàm tiện ích như tính precision, recall, điểm F1, vẽ đồ thị,... trong script utils.py. Việc lưu và nạp lại 1 mô hình đã huấn luyện được thực hiện trong 1 thư mục dành riêng cho mô hình đó. Các thông tin cần lưu được tổ chức như sau:

- Trọng số của mạng neuron được lưu trong weights.hdf5
- Các thông số cấu hình mạng (độ sâu, đầu ra,...) được lưu trong config.pkl
- Các giá trị từ điển (bao gồm vector từ) được lưu trong dictionary.npz
- Các ghi chép của hệ thống qua từng chu kì huấn luyện được lưu trong logs.txt và epochs.csv.
- Cấu hình hệ thống khi huấn luyện mô hình được lưu trong settings.py. Thay thế settings.py ở thư mục gốc bằng file này cho phép lặp lại quá trình huấn luyện của mô hình.
- Các đồ thị độ chính xác và hàm mục tiêu được lưu trong thư mục plots.

```
def save():
    f1 = self.directory + '/weights.hdf5'
    f2 = self.directory + '/config.pkl'
    f3 = self.directory + '/dictionary.npz'
    config = {'title_output': self.title_output,
              'content_output': self.content_output,
              'dense_neurons': self.dense_neurons,
              'title_len': self.title_len,
              'content_len': self.content_len,
              'classes': self.classes,
              'word_vec_dim': np.shape(self .word_vec)}
    self.model.save_weights(f1)
    pickle.dump(config, open(f2, 'wb'), pickle.HIGHEST_PROTOCOL)
    np.savez(f3, wit=self.word_to_index, itw=self.index_to_word,
```

```
            wv=self.word_vec)
    print('Saved model to %s.' % self.directory)
```

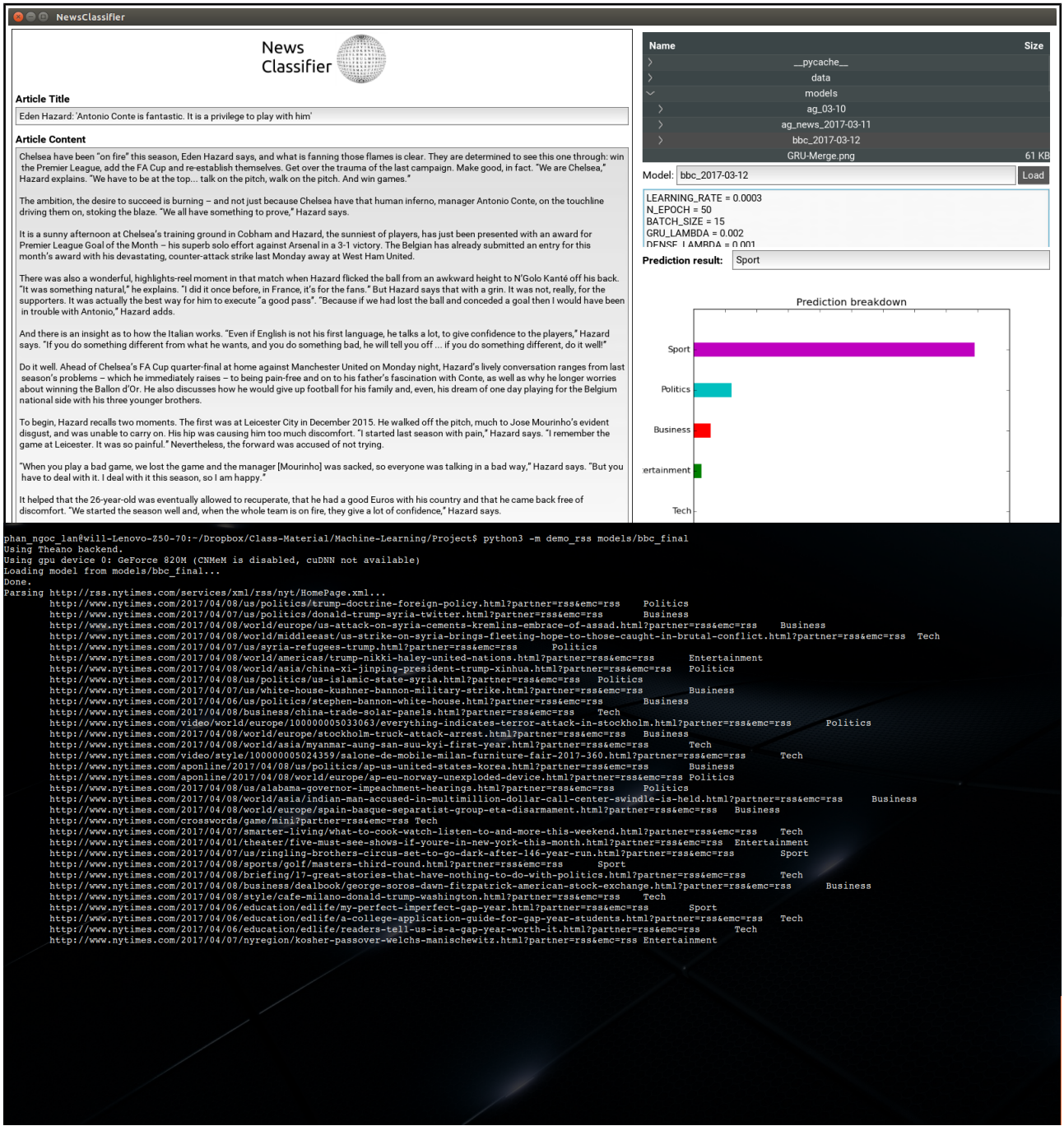
3.5.7. Các demo

Nhóm xây dựng 2 demo cho các mô hình được huấn luyện.

Demo thứ nhất cho phép nhập vào tiêu đề và nội dung của 1 văn bản tin tức bất kì, và trả về nhãn được dự đoán cũng như các xác suất tương ứng. Giao diện được xây dựng bằng framework Kivy.

Demo thứ hai lấy về các văn bản tin tức từ feed RSS của 1 số trang báo và phân loại từng bài báo dựa trên tiêu đề và phần mô tả (được coi như nội dung). Kết quả sau đó được lưu vào results.csv.

Hướng dẫn thực thi các file huấn luyện và demo được nêu trong README.md.



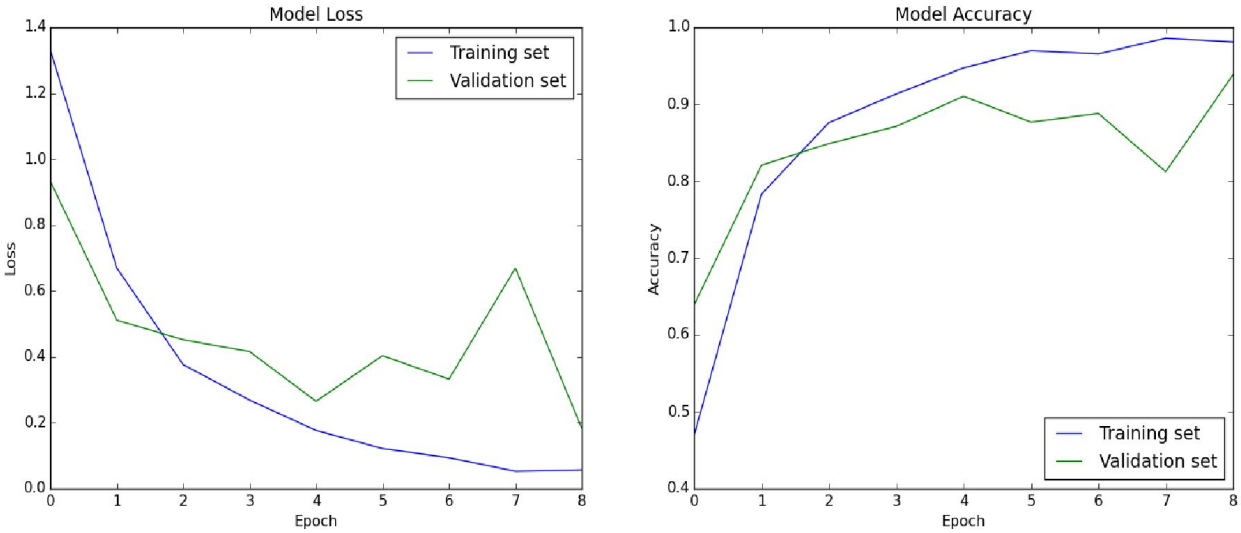
4. Kết quả

Kết quả trên từng tập dữ liệu được nêu trong bảng sau*:

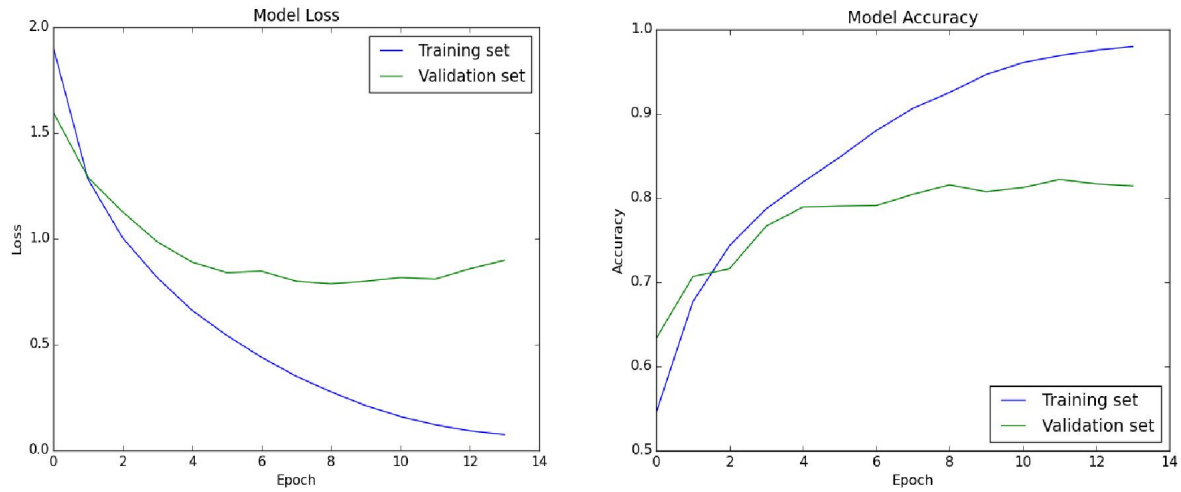
	AG-News	BBC	Reuters
Số lượng nhãn	4	5	86
Số neuron tối ưu	125	250	250
Hàm mục tiêu (train)	0.1702	0.0569	0.2767
Hàm mục tiêu (val)	0.2108	0.1832	0.7872
Hàm mục tiêu (test)	0.2176	0.2324	0.8119
Độ chính xác (train)	0.9400	0.9809	0.9252
Độ chính xác (val)	0.9263	0.9383	0.8156
Độ chính xác (test)	0.9232	0.9284	0.8021
Precision (test)	0.9240	0.9277	0.5649
Recall (test)	0.9232	0.9279	0.2965
Điểm F1 (test)	0.9232	0.9275	0.4494

** kết quả ứng với mô hình có độ chính xác validation cao nhất*

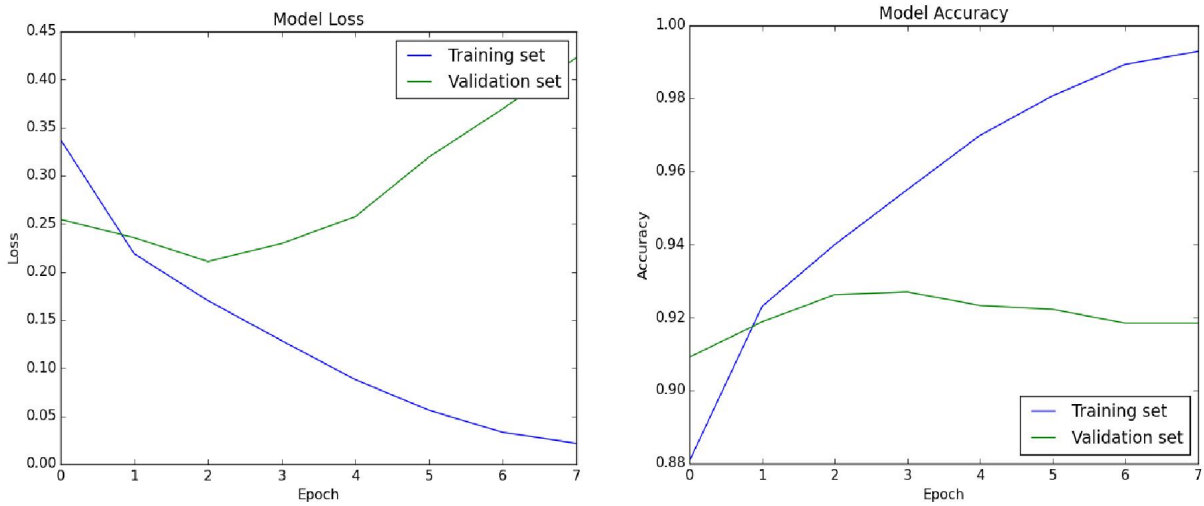
Đồ thị hàm mục tiêu và độ chính xác tại lần huấn luyện tối ưu:



Hình 7: Dataset BBC (0.002, 0.003)



Hình 8: Dataset Reuters (0.000, 0.003)



Hình 9: Dataset AG-News (0.00, 0.01)

Thời gian huấn luyện của hệ thống phụ thuộc vào kích cỡ của mạng được thiết lập trong settings.py và khả năng phần cứng của máy tính. Với cấu hình được nhóm lựa chọn chạy trên máy tính cá nhân có hỗ trợ GPU, một lần huấn luyện có thời gian chạy từ 16 phút đến 1 giờ.

Thời gian đoán nhận sau khi huấn luyện của mô hình là khá nhanh, thường trong khoảng 1-2s. Do đặc thù của backend Theano, lần đoán nhận đầu tiên có thể chậm hơn và có thể đến ~10s.

Nhóm cũng nhận thấy chênh lệch đáng kể giữa thời gian chạy khi có và không sử dụng hỗ trợ GPU. Sử dụng GPU cho phép hệ thống học nhanh gấp 3-4 lần, tuy nhiên không có chênh lệch đáng kể trong thời gian đoán nhận.

5. Kết luận và hướng phát triển

5.1. Khó khăn, tranh luận và hướng giải quyết

Một số khó khăn nhóm gặp phải bao gồm:

- Lựa chọn các tham số cho quá trình huấn luyện, đặc biệt là số lượng neuron cũng như số lớp. Do không đủ điều kiện thực hiện nhiều lần huấn luyện, nhóm chỉ lựa chọn số neuron tầng kết nối đầy đủ để thực hiện tối ưu. Các tham số cấu hình khác của mạng được tham khảo ở các nguồn đã trích dẫn hoạc được nhóm ước lượng.
- Tranh luận về việc sử dụng thêm các giải pháp chống overfit. Mặc dù có ý định ban đầu là sử dụng thêm các giải pháp như regularization để giảm overfit, nhóm nhận thấy qua các lần chạy thử chỉ có tập Reuters là có biểu hiện overfit, với chênh lệch độ chính xác khoảng 10%. Tập này cũng đồng thời có tỉ lệ nhiễu lỗi cao hơn 2 tập còn lại, do đó khó đảm bảo là có thể cải thiện kể cả với các kĩ thuật trên. Việc thêm vào các kĩ thuật này lại yêu cầu chạy thêm rất nhiều thử nghiệm để tìm các giá trị tối ưu cho tham số, do đó nhóm quyết định chưa thực hiện trong đồ án này và có thể quay lại trong tương lai khi có những bộ dữ liệu đa dạng hơn.
- Vấn đề khi vector hóa các kí tự đặc biệt. Do nhóm tự định nghĩa các từ MASK_TOKEN và UNKNOWN_TOKEN, vector của các từ này phải được chèn vào các vector từ được nạp từ GloVe. Với MASK_TOKEN, giá trị của vector không quan trọng do từ chỉ đóng vai trò chiếm đủ độ dài câu và sẽ được bỏ qua. Giá trị được nhóm chọn là 1 vector 0. Tuy nhiên, UNKNOWN_TOKEN được hệ thống xử lý và phần nào mang ngữ nghĩa. Giá trị được nhóm thống nhất chọn là 1 vector ngẫu nhiên có giá trị tương đối lớn để phân biệt với các từ khác trong từ điển.
- Đổi lại cho khả năng tăng tốc bằng GPU, việc sử dụng các nền tảng tính toán biểu tượng cản trở khá nhiều trong quá trình debug và xác nhận các kết quả trung gian, do việc tối ưu đồ thị tính toán.

5.2. Kết luận

Từ kết quả thu được, nhóm rút ra một số kết luận như sau:

- Mạng neuron hồi quy, cụ thể là GRU, có khả năng xử lý ngôn ngữ rất tốt, mô phỏng 1 phần quá trình đọc-hiểu của con người thay vì dựa vào các từ khóa.
- Mô hình hoạt động rất tốt với các văn bản ngắn hoặc trung bình, trong đó GRU dễ dàng học được sự liên kết giữa các từ từ đầu văn bản đến cuối văn bản.

- Biểu diễn câu bằng vector từ tuy lưu giữ được nhiều thông tin hơn về văn bản, nhưng ngược lại không thể tránh khỏi việc đưa vào nhiễu trong dữ liệu. Vì từ điển gần như không thể chứa tất cả các từ trong tất cả các ví dụ, ta buộc phải thay thế bằng kí tự UNKNOWN_TOKEN.
- Mô hình thể hiện rõ khả năng chịu nhiễu rất tốt của mạng neuron. Với số lượng từ UNKNOWN_TOKEN ở ngưỡng 3-6%, mô hình rất ít bị ảnh hưởng và không có dấu hiệu overfit. Tuy nhiên, ta cũng thấy số lượng ví dụ cần đạt 1 lượng nhất định để tránh overfit như trường hợp tập Reuters.
- Hệ thống có thể dễ dàng học các phân loại trên nhiều loại văn bản không phải tin tức hoặc chỉ có 1 input (bằng cách đặt TITLE_OUTPUT bằng 1 và chỉ truyền vào tiêu đề là xâu rỗng, hệ thống sẽ học cách bỏ qua hoàn toàn thông tin về tiêu đề).

5.3. Hướng phát triển

Nhóm có dự định phát triển hệ thống theo một số hướng như sau:

- Kết hợp mô hình với một thuật toán phân loại khác (có thể là SVM) và lựa chọn đáp án dựa trên mức tự tin của mỗi dự đoán.
- Xây dựng cơ chế để một mô hình đã huấn luyện có thể tiếp tục học online để cải thiện kết quả. Chủ yếu liên quan đến việc quyết định mức độ học với mỗi ví dụ học mới.
- Áp dụng một số kĩ thuật giảm overfit như Regularization, Dropout, Pooling,...
- Thử nghiệm mô hình với các tập dữ liệu ở các domain tương tự như phân loại tweet trên Twitter, phân loại lời bài hát,...

Tài liệu tham khảo

1: Zach Chase, Nicolas Genain, Orren Karniol-Tambour, Learning Multi-Label Topic Classification of News Articles,
2: Thorsten Joakims, Text Categorization with Support Vector Machines: Learning with Many Relevant Features,
3: Dennis Ramdass, Shreyes Seshasai, Document Classification for Newspaper Articles, 2009
4: Francois Chollet, Keras Documentation, , https://keras.io/
5: , Tổng quan về Mạng Neuron (Neural Network) - BIS, 2016, http://bis.net.vn/forums/t/482.aspx
6: Nguyễn Nhật Quang, Giáo trình Học Máy, 2016
7: Denny Britz, Recurrent Neural Networks Tutorial, Part 1 – Introduction to RNNs – WildML, , http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/
8: Jeffrey Pennington; Richard Socher; Christopher D. Manning, GloVe: Global Vectors for Word Representation, 2014
9: Xavier Glorot, Yoshua Bengio, Understanding the difficulty of training deep feedforward neural networks, 2010
10: NLTK Project, NLTK 3.0 documentation, , http://www.nltk.org/
11: Sebastian Ruder, An overview of gradient descent optimization algorithm, 2016, http://sebastianruder.com/optimizing-gradient-descent/index.html#rmsprop
12: John D. Hunter, Matplotlib: A 2D Graphics Enviroment, 2007
13: Eric Jones, Travis Olyphant, Pearu Peterson, et al , SciPy: Open Source Scientific Tools for Python, 2001, http://www.scipy.org
14: Stefan van der Walt, S. Chris Colbert, Gael Varoquaux, The NumPy Array: A Structure for Efficient Numerical Computation, 2011