# Các bước tiền xử lý dữ liệu:

* **Mô tả dữ liệu train**:

Phân bố dữ liệu các class:

{'rec.sport.hockey': 600, 'soc.religion.christian': 599,

'rec.motorcycles': 598, 'sci.crypt': 594,

'sci.space': 593, 'rec.autos': 593,

'sci.med': 592, 'comp.os.ms-windows.misc': 591,

'rec.sport.baseball': 591, 'comp.windows.x': 590,

'sci.electronics': 588, 'comp.sys.ibm.pc.hardware': 588,

'misc.forsale': 583, 'comp.graphics': 579,

'comp.sys.mac.hardware': 567, 'talk.politics.mideast': 564,

'talk.politics.guns': 546, 'alt.atheism': 475,

'talk.politics.misc': 465, 'talk.religion.misc': 374}

* **Làm sạch dữ liệu:**

1. tolower text sau đó remove stopwords
2. loại bỏ các khoảng trống thừa, kí tự đặc biệt (sử dụng regular expression để hiểu nguyên tắc sau đó dùng gensim.utils.simple\_preprocess để biết cách dùng thư viện
3. đưa từ về từ gốc sử dụng nltk.stem.PorterStemmer

* **Biểu diễn dữ liệu:**

1. Bag of word:
   * 1. tạo word freq là 1 dict chứa từ: tần suất xuất hiện của từ nhằm mục đích bỏ qua những từ ít xuất hiện, từ lỗi để giảm chiều vector
     2. biểu diễn một sentence, mỗi word là 1 one hot vector sau đó cộng lại (vòng for trực tiếp dễ hình dung hơn)
     3. độ dài 1 vector bằng độ dài từ điển (25000) do bỏ các từ có tần suất xuất hiện trên cả bộ dữ liệu <4 (các từ này thường là từ lỗi)
2. tf-idf
3. tính tf là tần suất xuất hiện của từ trong câu: số lần suất hiện / tổng số từ
4. idf: log (số văn bản / số văn bản có chứa từ đó)
   * 1. (số văn bản chứa một từ đã được tính trước, biểu diễn dạng dict)
5. chọn df bằng 4, mục đích chỉ loại bỏ tối đa từ có khả năng cao bị lỗi

todo: average word embedding; doc2vec; tf-idf weighted word2vec + doc2vec

* **Kết quả:**
  + bộ từ điển 25000 nghìn từ
  + 5 từ có tần xuất xuất hiện cao nhất và tần suất tương ứng:

[('the', 0.6968944099378882),

 ('organ', 0.9627329192546584),

 ('line', 0.9971606033717835),

('subject', 1.0),

 ('from', 1.0)]

# B. Naive Bayes Classifier

**Các bước tiến hành:**

1. Tính xác suất của mỗi class trong toàn bộ tập dữ liệu
2. Tính xác suất một từ xuất hiện trong class cho trước
3. Predict: tính xác suất có điều kiện p(c/x) ...

**Kết quả:**

* + bộ train:

- accuracy = 0.949068322981366

* bộ test:

- accuracy = 0.954551512728242

*Giống hoàn toàn với MultinomialNB của scikit-learn (về độ chính xác, thời gian fit model ~ 1.5 phút, predict ~ 0.1ms / 1 observation)*

todo: **GaussianNB**,  **[BernoulliNB](https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.BernoulliNB.html" \l "sklearn.naive_bayes.BernoulliNB)**

# C. Logistic regression

## Lý thuyết:

* Mục tiêu tạo ra 1 nonlinear function giúp phân tách hai class
* Xác suất một điểm thuộc class 1 (có hai class 0 và 1):

**sigmoid(-W.T \* X + b)** *(\* là matrix multiplication)*

* Hàm mục tiêu: cross entropy

cost(Y, Y\_predict) = -( ∑ y \* log(y\_predict) + (1-y) \* log(1-y\_predict))

với Y là giá trị nhãn ban đầu, Y\_predict là giá trị dự đoán

y và y\_predict là giá trị tính cho một dữ liệu đầu vào.

* Đạo hàm hàm mục tiêu theo w ta được:

**dw = (y\_predict - y) \* x**

⇒ là một shallow nerural network, 1 hidden layer, activation là sigmoid đầu ra là giá trị xác suất một điểm được phân vào class “1”

## Implement

* + Khởi tạo W:

- Để tránh vanishing & exploding gradient descent ⇒ dùng random.randn(dimension\_of\_x, 1) để khởi tạo giá trị ngẫu nhiên (không dùng np.zeros())

- Append vào mỗi vector x giá trị 1 ⇒ bias

* + forward:

- tính các giá trị dự đoán Y\_predict = sigmoid(-W.T\*X)

* + backward:
    - tính dw = (Y\_predict - Y\_truth) \* X
    - cập nhật W = W - learning\_rate \* dW
  + điều kiện dừng:
    - giới hạn vòng lặp
    - giới hạn mức độ thay đổi của w

## Logistic regression cho multiclass classification:

Sử dụng kỹ thuật one-vs-rest, học trọng số wi xác định một điểm dữ liệu có thuộc class i hay không sau đó trả về class mà xác suất điểm thuộc class đó là lớn nhất.

## Tối ưu tốc độ

Sử dụng mini batch GD thay vì stochastic GD, tận dụng khả năng tính toán song song của numpy

1. Kết quả

* count vectorizer:
  + my model:
* train accuracy: 0.974001774622892
* test accuracy: 0.7083833133413301
  + sklearn model
* train accuracy: 0.99400239904038
* test accuracy: 0.711182193789151
* tfidf vectorizer:
* my model:
* train accuracy: 0.9961845607808341
* test accuracy:  0.8290017326402772
* sklearn model:
* train accuracy: 0.99174800354924
* test accuracy: 0.831134212981474

**Vấn đề:**

Dễ dàng nhận thấy các mô hình trên đều bị overfit (low bias - high variance) ⇒

* Sử dụng regularization để giảm độ phức tạp mô hình

⇒ kết quả kém một cách bất ngờ, khi tăng coef cho l2 thì độ chính xác của train và test đều giảm

* Giảm số epochs, tăng tolerance (early stop) ⇒ chất lượng model không ổn định, không tăng nhiều.
* Giảm chiều dữ liệu xuống còn 2000 (TruncateSVD or PCA)

⇒ đã thử giảm chiều dữ liệu, kết quả không cải thiện nhiều, vẫn overfit

Sử dụng softmax thay cho logistic ⇒ kết quả vẫn kém: 70% cho train và 60% cho test

**=> NO IDEA - todo**

* Tốc độ thuật toán chạy rất chậm so với sklearn: 20 phút cho 100 epochs

trong khi tốc độ của sklearn là ~3 phút

* đã thử sử dụng mini-batch, không khả quan so với 3 phút của thư viện.

# D. KMeans

## Lý thuyết

* 1. Mục tiêu: phân các điểm dữ liệu thành k cụm (k là tham số đầu vào) sao cho các điểm trong cùng một cụm có nhiều điểm tương động, giống nhau và các điểm khác cụm có sự phân biệt, khác nhau.
  2. Phương pháp KMeans:
     1. Sử dụng khoảng cách giữa hai điểm để đánh giá sự giống nhau giữa hai điểm dữ liệu (càng gần nhau thì càng giống nhau) thông thường dùng Norm k.
     2. Implement: sử dụng phương pháp tối ưu lặp:
        1. khởi tạo ngẫu nhiên k cluster centers
        2. tính lại các nhãn cho các điểm (lấy nhãn theo center gần nhất)
        3. tính lại center của mỗi class bằng cách tính trung bình mỗi thành phần của các điểm thuộc class đó
        4. lặp lại 2 tới khi nhãn của các điểm không còn thay đổi
  3. các phương pháp phân cụm khác

*DBSCAN, Mean Shift, Spectral Clustering, Mixture of Gaussians...*

## Implement

* Implement theo các bước của thuật toán, vectorize with numpy for faster implementation

## Kết quả

* my model:
  + train purity: 0.4200532386867791
  + test purity: 0.351192856190857
  + time: 180.50415587425232 s
* sklearn model:
  + train purity: 0.4200532386867791
  + test purity: 0.351192856190857
  + time: 180.50415587425232 s

## Nguyên nhân

Sau khi sử dụng t-SNE giảm chiều dữ liệu về 3 chiều có thể dễ dàng quan sát phân bố của các class không ở dạng “cầu” mà hơi “phẳng” từ đó việc dựa vào khoảng cách hai điểm dữ liệu để xác định xem hai điểm đó có thuộc cùng class hay không là chưa hợp lý... Nếu dùng các phương pháp phân cụm thì sử dụng các phương pháp density based (ví dụ DBSCAN) sẽ khả quan hơn. Và  điều đó cũng lý giải các phương pháp phân loại sử dụng siêu phẳng phân tách (đơn giản như PLA) cũng có thể cho kết quả tốt hơn trong bài toán này.

# E. SVM and Neural Network with Torch

## SVM

1. kernel = poly:
   * + train accuracy: 0.9990239574090506
     + test accuracy: 0.5913634546181528

* training time: 300s

1. kernel = rbf, C = 5:
   * + train accuracy: 0.9965394853593611
     + test accuracy 0.8260695721711315
     + training time: 260s
2. kernel = linear
   * + train accuracy: 0.9899733806566104
     + test accuracy: 0.8295348527255765
     + training time: 120s

## Neural network

1. Softmax regression
   * 1. Layer (type)               Output Shape         Param #
     2. ================================================================
     3. Linear-1            [-1, 49776, 20]         497,780
     4. Sigmoid-2            [-1, 49776, 20]               0
     5. Softmax-3            [-1, 49776, 20]               0
     6. ================================================================
     7. Total params: 497,780
     8. Trainable params: 497,780
     9. Non-trainable params: 0
     10. ----------------------------------------------------------------
     11. Input size (MB): 4725.74
     12. Forward/backward pass size (MB): 22.79
     13. Params size (MB): 1.90
     14. Estimated Total Size (MB): 4750.43
     15. ------------------------------------------------------

Training time: 1s / 1epoch, convergence after 50 epoch

Train accuracy: 0.9840283939662822

Test accuracy: 0.7954151672664268

1. Other test model

        Layer (type)               Output Shape         Param #

================================================================

            Linear-1           [-1, 49776, 200]       4,977,800

              ReLU-2           [-1, 49776, 200]               0

           Dropout-3           [-1, 49776, 200]               0

            Linear-4           [-1, 49776, 100]          20,100

              ReLU-5           [-1, 49776, 100]               0

           Dropout-6           [-1, 49776, 100]               0

            Linear-7            [-1, 49776, 20]           2,020

           Sigmoid-8            [-1, 49776, 20]               0

           Softmax-9            [-1, 49776, 20]               0

================================================================

Total params: 4,999,920

Trainable params: 4,999,920

Non-trainable params: 0

----------------------------------------------------------------

Input size (MB): 4725.74

Forward/backward pass size (MB): 364.57

Params size (MB): 19.07

Estimated Total Size (MB): 5109.39

------------------------------------------------------------

Training time: 1.2 s / 1 epoch, converenge after 20 epoch

Train accuracy: 0.9991126885536823

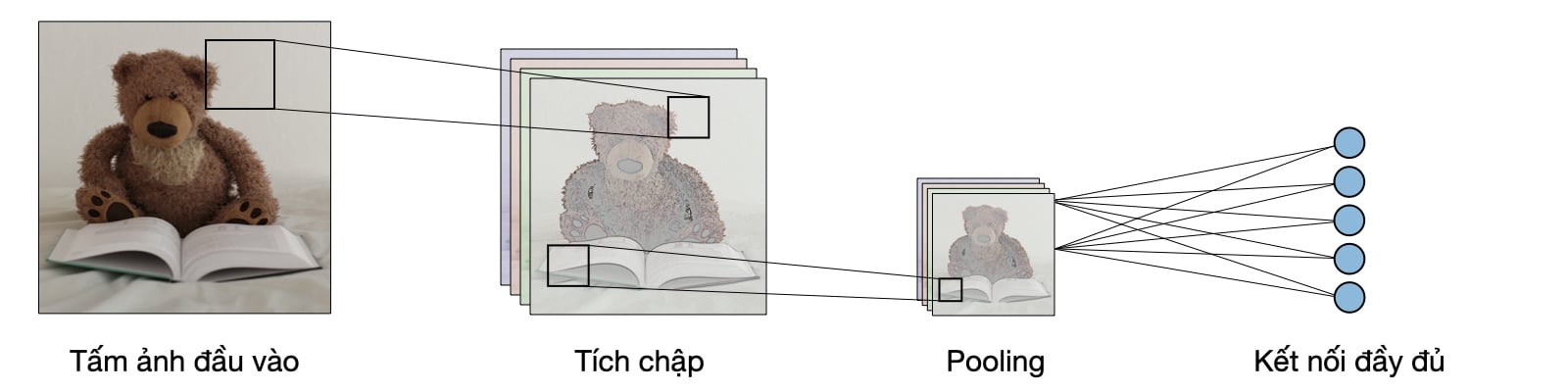
Test accuracy: 0.8178062108489937

* 1. Todo
     + 1. Lstm, gru...
       2. Bert, glove, fastext

# F. CNN and RNN

## 1. Lý thuyết

a. CNN (convolution Neural Network)

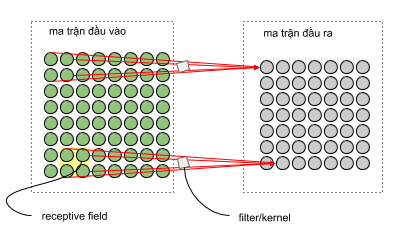


Mô tả mạng CNN (Nguồn: [Stanford University](https://stanford.edu/~shervine/teaching/cs-230/illustrations/architecture-cnn-vi.jpeg?6b63c596f802c81d7cff2028ae6572ab))

#### Convolution Layer

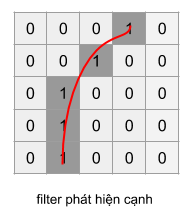
Convolution layer là lớp quan trọng nhất và cũng là lớp đầu tiên của của mô hình CNN. Lớp này có chức năng chính là phát hiện các đặc trưng có tính không gian hiệu quả. Trong tầng này có 4 đối tượng chính là: ma trận đầu vào, bộ **filters**, và **receptive field**, **feature map**. Conv layer nhận đầu vào là một ma trận 3 chiều và một bộ filters cần phải học. Bộ filters này sẽ trượt qua từng vị trí trên bức ảnh để tính tích chập (convolution) giữa bộ filter và phần tương ứng trên bức ảnh. Phần tưng ứng này trên bức ảnh gọi là receptive field, tức là vùng mà một neuron có thể nhìn thấy để đưa ra quyết định, và mà trận cho ra bới quá trình này được gọi là feature map.

Với ví dụ ở bên dưới, dữ liệu đầu vào ở là ma trận có kích thước 8x8x1, một bộ filter có kích thước 2x2x1, feature map có kích thước 7x7x1. Mỗi giá trị ở feature map được tính bằng tổng của tích các phần tử tương ứng của bộ filter 2x2x1 với receptive field trên ảnh. Và để tính tất cả các giá trị cho feature map, ta sẽ phải trượt filter từ trái sáng phải, từ trên xuống dưới.



#### Tầng convolution như là feature detector

Tầng convolution có chức năng chính là phát hiện đặc trưng cụ thể của bức ảnh. Những đặc trưng này bao gồm đặc trưng cơ bản là góc,cạnh, màu sắc, hoặc đặc trưng phức tạp hơn như texture của ảnh. Vì bộ filter quét qua toàn bộ bức ảnh, nên những đặc trưng này có thể nằm ở vị trí bất kì trong bức ảnh, cho dù ảnh bị xoáy trái/phải thì những đặc trưng này vẫn bị phát hiện.



Ví dụ filter detect edge

Chúng ta thấy rằng, chỉ ở những vị trí trên bức ảnh có dạng góc như đặc trưng ở filter thì mới có giá trị lớn trên feature map, những vị trí còn lại sẽ cho giá trị thấp hơn. Điều này có nghĩa là, filter đã phát hiện thành công một dạng góc/cạnh trên dự liệu đầu vào. Tập hơn nhiều bộ filters sẽ cho phép phát hiện được nhiều loại đặc trưng khác nhau,và giúp định danh được đối tượng.

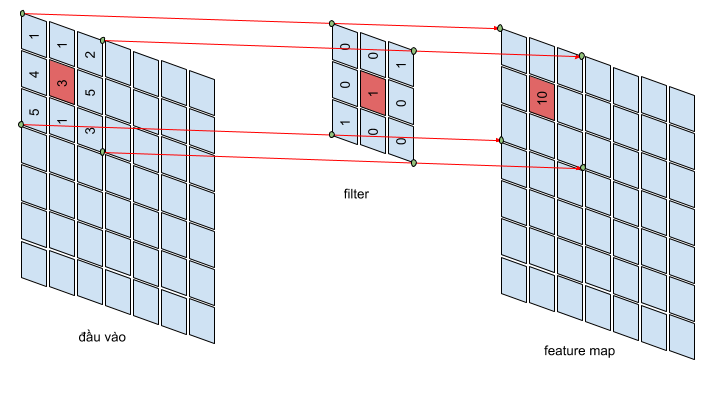
#### Các tham số của tầng convolution: Kích thước bộ filter, stride và padding

Kích thước bộ filter là một trong những tham số quan trọng nhất của tầng convolution. Kích thước này tỉ lệ thuận với số tham số cần học tại mỗi tầng convolution và là tham số quyết định receptive field của tầng này. Kích thước phổ biến nhất của bộ filter là 3x3.

Kích thước filter nhỏ được ưu tiên lựa chọn thay kích thước lớn vì những lý do sau đây.

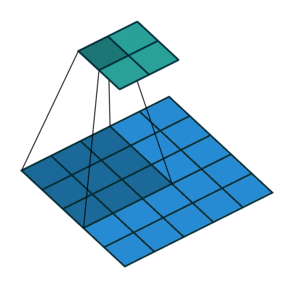
| **Filter Nhỏ** | **Filter Lớn** |
| --- | --- |
| Kích thước nhỏ thì mỗi lần nhìn được một vùng nhỏ các pixel | Receptive field lớn |
| Rút trích được đặc trưng có tính cục bộ cao | Các đặc trưng có tính tổng quát hơn |
| Phát hiện được các đặc trưng nhỏ hơn | Bắt được những phần cơ bản của bức ảnh |
| Đặc trưng rút trích được sẽ đa dạng, hữu ích hơn ở tầng sau | Thông ít rút trích được ít đa dạng |
| Giảm kích thước ảnh chậm hơn, do đó cho phép mạng sâu hơn | Giảm kích thước ảnh nhanh, do đó chỉ cho phép mạng nông |
| Ít trọng số hơn, chia sẻ trọng số tốt hơn | Chia sẽ trọng số ít ý nghĩa hơn |

Kích thước filter của tầng convolution hầu hết đều là số lẻ, ví dụ như 3x3 hay 5x5. Với kích thước filter lẻ, các giá trị của feature map sẽ xác định một tâm điểm ở tầng phía trước. Nếu các bạn chọn filter có kích thước 2x2, 4x4 thì chúng ta sẽ gặp khó khăn khi muốn tìm vị trí tương ứng của các giá trị feature map trên không gian ảnh.



Ở những trường hợp đặt biệt như filter có kích thước 1x1, hay có kích thước bằng với ma trận đầu vào, tầng convolution có ý nghĩa rất thú vị. Khi có kích thước 1x1, tầng convolution xem mỗi điểm như một đặc trưng riêng biệt, có chức năng giảm chiều (tăng chiều) khi số lượt feature map ở tầng sau nhỏ hơn (lớn hơn) tầng trước. Filter 1x1 đã được sử dụng trong kiến trúc mạng phổ biến như Inception networks. Trong khi đó, filter với kích thước bằng ảnh đầu vào, tầng convolution có chức năng y hệt fully connected layer.

Ngoài ra, có một tham số đặc biệt là stride, thể hiện số pixel bạn cần phải dịch chuyển mỗi khi trượt bộ filter qua bức ảnh. Ở ví dụ bên dưới, với tham số stride bằng 2, bộ filter sẽ dịch chuyển 2 pixel mỗi lần áp dụng phép convolution.



Khi ta áp dụng phép convolution thì ma trận đầu vào sẽ có nhỏ dần đi, do đó số layer của mô hình CNN sẽ bị giới hạn, và không thể xậy đựng deep nets mong muốn. Để giải quyết tình trạng này, các bạn cần padding vào ma trận đầu vào để đảm bảo kích thước đầu ra sau mỗi tầng convolution là không đổi. Do đó có thể xậy dựng được mô hình với số tầng convolution lớn tùy ý. Một cách đơn giản và phổ biến nhất để padding là sử dụng hàng số 0, ngoài ra các ta cũng có thể sử dụng reflection padding hay là symmetric padding.

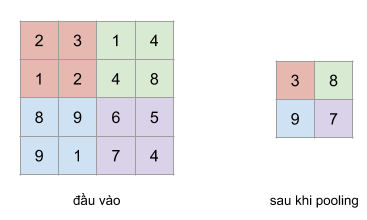
#### Nonlinear Layer

ReLU (Rectified Linear Units, f = max(0, x)) là hàm kích hoạt phổ biến nhất cho CNN tại thời điểm của bài viết, được giới thiệu bởi Geoffrey E. Hinton năm 2010. Trước khi hàm ReLU được áp dụng thì những hàm như sigmoid hay tanh mới là những hàm được sử dụng phổ biến. Hàm ReLU được ưa chuộng vì tính toán đơn giản, giúp hạn chế tình trạng vanishing gradient, và cũng cho kết quả tốt hơn. ReLU cũng như những hàm kích hoạt khác, được đặt ngay sau tầng convolution, ReLU sẽ gán những giá trị âm bằng 0 và giữ nguyên giá trị của đầu vào khi lớn hơn 0.

ReLU cũng có một số vấn đề tiềm ẩn như không có đạo hàm tại điểm 0, giá trị của hàm ReLU có thể lớn đến vô cùng và nếu chúng ta không khởi tạo trọng số cẩn thận, hoặc khởi tạo learning rate quá lớn thì những neuron ở tầng này sẽ rơi vào trạng thái chết, tức là luôn có giá trị < 0.

#### Pooling Layer

Sau hàm kích hoạt, thông thường chúng ta sử dụng tầng pooling. Một số loại pooling layer phổ biến như là max-pooling, average pooling, với chức năng chính là giảm chiều của tầng trước đó. Với một pooling có kích thước 2x2, ta sẽ phải trược filter 2x2 này trên những vùng ảnh có kích thước tương tự rồi sau đó tính max, hay average cho vùng ảnh đó.



Ý tương đằng sau tầng pooling là vị trí tuyết đối của những đặc trưng trong không gian ảnh không còn cần cần thiết, thay vào đó vị trí tương đối giữ các đặc trưng đã đủ để phân loại đối tượng. Hơn giảm tầng pooling có khả năng giảm chiều cực kì nhiều, làm hạn chế overfit, và giảm thời gian huấn luyện tốt.

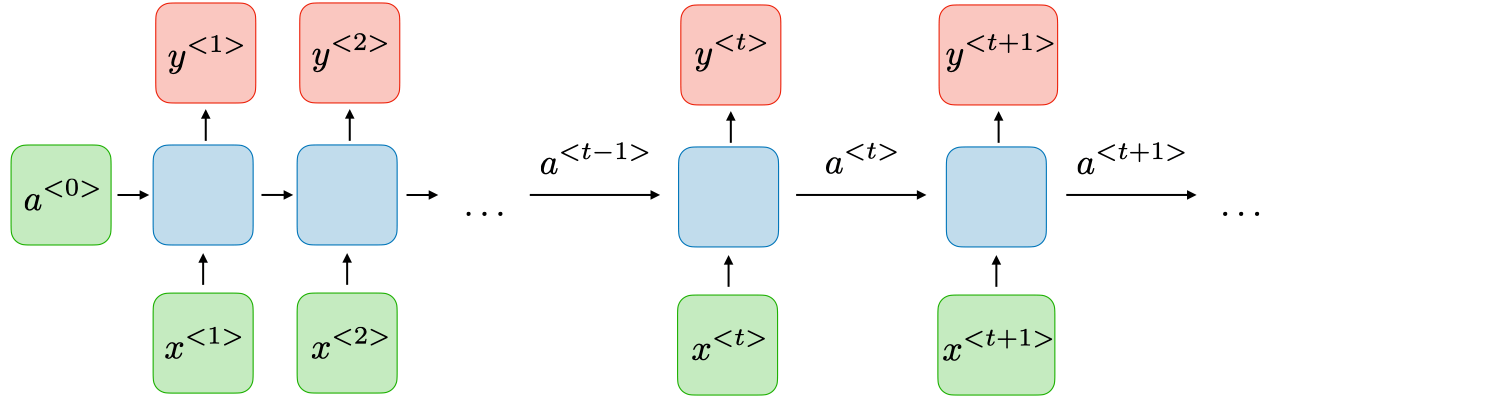
#### Fully Connected Layer

Tầng cuối cùng của mô hình CNN trong bài toán phân loại ảnh là tầng fully connected layer. Tầng này có chức năng chuyển ma trận đặc trưng ở tầng trước thành vector chứa xác suất của các đối tượng cần được dự đoán. Ví dụ, trong bài toán phân loại số viết tay MNIST có 10 lớp tương ứng 10 số từ 0-1, tầng fully connected layer sẽ chuyển ma trận đặc trưng của tầng trước thành vector có 10 chiều thể hiện xác suất của 10 lớp tương ứng.

Và cuối cùng, quá trình huấn luyện mô hình CNN cho bài toán phân loại ảnh cũng tương tự như huấn luyện các mô hình khác. Chúng ta cần có hàm độ lỗi để tính sai số giữ dự đoán của mô hình và nhãn chính xác, cũng như sử dụng thuật toán backpropagation cho quá trình cập nhật trọng số.

1. RNN (Recurrent Neural Network)

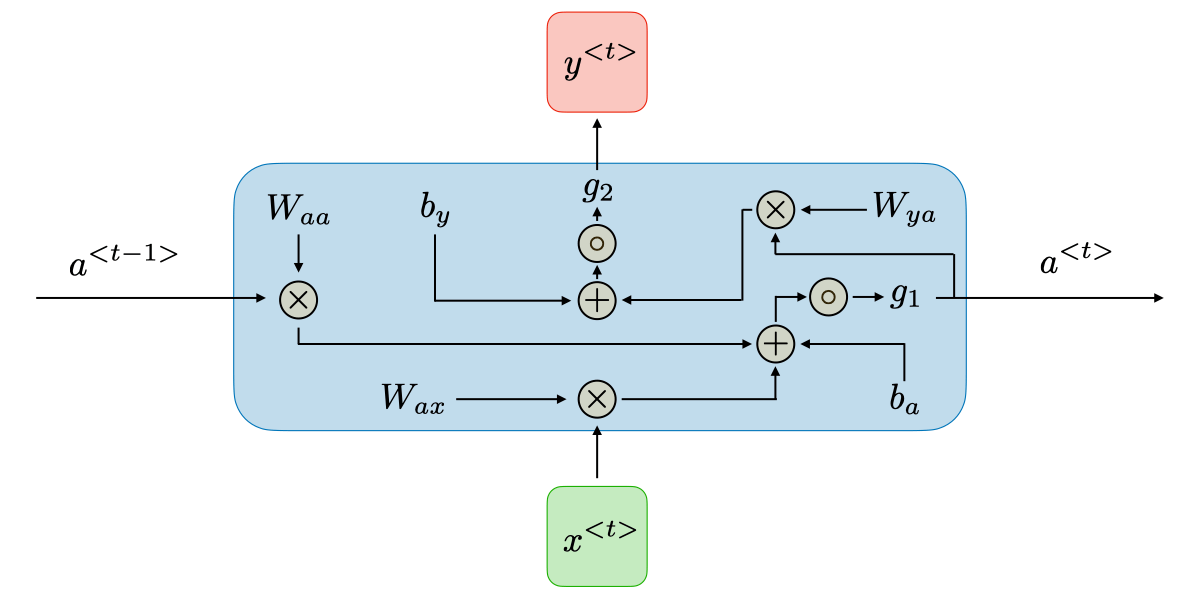
**Kiến trúc của một mạng RNN truyền thống** Các mạng neural hồi quy, còn được biến đến như là RNNs, là một lớp của mạng neural cho phép đầu ra được sử dụng như đầu vào trong khi có các trạng thái ẩn. Thông thường là như sau:



1. Tại mỗi bước *t*, giá trị kích hoạt  *a*<*t*> và đầu ra *y*<*t*> được biểu diễn như sau:

*a*<*t*>=*g*1​(*Waa*​*a*<*t*−1>+*Wax*​*x*<*t*>+*ba*​)​ và *y*<*t*>=*g*2​(*Wya*​*a*<*t*>+*by*​)​

​với *Waa*​,*Wya*​,*ba*​,*by*​ là các hệ số được chia sẻ tạm thời và *g*1​,*g*2​ là các hàm kích hoạt.



1. Ưu và nhược điểm của một kiến trúc RNN thông thường được tổng kết ở bảng dưới đây:

|  |  |
| --- | --- |
| **Ưu điểm** | **Hạn chế** |
| • Khả năng xử lí đầu vào với bất kì độ dài nào • Kích cỡ mô hình không tăng theo kích cỡ đầu vào • Quá trình tính toán sử dụng các thông tin cũ • Trọng số được chia sẻ trong suốt thời gian | • Tính toán chậm • Khó để truy cập các thông tin từ một khoảng thời gian dài trước đây • Không thể xem xét bất kì đầu vào sau này nào cho trạng thái hiện tại |

**Ứng dụng của RNNs**Các mô hình RNN hầu như được sử dụng trong lĩnh vực xử lí ngôn ngữ tự nhiên và ghi nhận tiếng nói. Các ứng dụng khác được tổng kết trong bảng dưới đây:

|  |  |  |
| --- | --- | --- |
| **Các loại RNN** | **Hình minh hoạ** | **Ví dụ** |
| Một-Một *Tx*​=*Ty*​=1 |  | Mạng neural truyền thống |
| Một-nhiều *Tx*​=1,*Ty*​>1 |  | Sinh nhạc |
| Nhiều-một *Tx*​>1,*Ty*​=1 |  | Phân loại ý kiến |
| Nhiều-nhiều *Tx*​=*Ty*​ |  | Ghi nhận thực thể tên |
| Nhiều-nhiều *Tx*​​=*Ty*​ |  | Dịch máy |

**Hàm mất mát**Trong trường hợp của mạng neural hồi quy, hàm mất mát L của tất cả các bước thời gian được định nghĩa dựa theo mất mát ở mọi thời điểm như sau:

L(*y*​, *y\_predict*)=*t*=1∑*Ty*​​L(*y*​<*t*>,*y\_predict*<*t*>)​

**Lan truyền ngược theo thời gian**Lan truyền ngược được hoàn thành ở mỗi một thời điểm cụ thể. Ở bước T*T*, đạo hàm của hàm mất mát \mathcal{L}L với ma trận trọng số W*W* được biểu diễn như sau:

A picture containing object, clock, red, holding

Description automatically generated

## 2. Thực hành

a. CNN

- Input: sử dụng lại vector tfidf của văn bản đã xử lý từ trước, giảm chiều về 1000 sử dụng TruncateSVD

- Kiến trúc mạng: Conv1d + ReLu + Conv1d + ReLu + MaxPool1d +

Conv1d + MaxPool1d + Linear + Softmax

Sau 100 epochs, 182s thu được kết quả:

* + - Train accuracy: 0.91952085181
    - Test accuracy: 0.65973610555

b. RNN

- Input: biểu diễn một văn bản đầu vào dưới dạnh vector với giá trị thứ j của vector là index của từ tương ứng trong từ điển. Từ trong bộ test không có trong từ điển 🡺 “US0”, khoảng trắng thêm vào câu để tạo vector có độ dài bằng nhau (lấy độ dài một vector đại diện cho đoạn text là 80 – trung bình độ dài của bộ train)

- Kiến trúc mạng: Embedding + Dropout + RNN + Linear + Softmax

Layer Embedding: đóng vai trò bảng lookup, mapping một từ với vector Embedding tương ứng

Sau khoảng 100 epochs, 1800s thu được kết quả:

* + - Train accuracy: 0.91493174061
    - Test accuracy: 0.67612954818
* Nhận xét:
  + - 1. Với một mạng CNN đơn giản, việc giảm chiều dữ liệu về 1000 chiều làm mất mát nhiều thông tin (bộ nhớ giới hạn nên cần giảm chiều), sử dụng vector tfidf là chưa được phù hợp (recommend matrix embedding) model bị overfit => cần thêm dropout
      2. Với RNN truyền thống, dữ liệu đầu vào sử dụng lại phần đã tiền xử lý là chưa hợp lý, nên lấy thêm thông tin từ dữ liệu gốc ví dụ từ loại (không stemming), từ viết thường hay viết hoa, các dấu câu cũng có ý nghĩa. Việc sử dụng câu với độ dài cố định cũng không cần thiết (trong trường hợp có đủ bộ nhớ)

Cần xử lý overfit.

Link code github: <https://github.com/trongtuyen99/text_preprocessing>

Link data github: <https://github.com/trongtuyen99/20news_goup_data>

source code xem tại: ./src và ./notebooks