Deep Learning for Texts

Lê Hồng Phương

<phuonglh@vnu.edu.vn>
College of Science
Vietnam National University, Hanoi

August 19, 2016

Content

Content

Content

Content

Content

Outline

Computational Linguistics

"*Human knowledge is expressed in language. So computational linguistics is very important*" – Mark Steedman, ACL President Address, 2007.

- Use computer to process natural language
- Example 1: Machine Translation (MT)
  - 1946, concentrated on Russian → English
  - Considerable resources of USA and European countries, but limited performance
  - Underlying theoretical difficulties of the task had been underestimated.
  - *Today, there is still no MT system that produces fully automatic high-quality translations.*

Computational Linguistics

Some good results...

Computational Linguistics

Some bad results...

Computational Linguistics

But probably, there will not be for some time!

Computational Linguistics

Example 2: Analysis and synthesis of spoken language:

- Speech understanding and speech generation
- Diverse applications:
  - text-to-speech systems for the blind
  - inquiry systems for train or plane connections, banking
  - office dictation systems

Computational Linguistics

Example 3: The Winograd Schema Challenge:

1. The customer walked into the bank and stabbed one of the tellers. He was immediately taken to the emergency room.
    - Who was taken to the emergency room?
    - The customer / the teller
2. The customer walked into the bank and stabbed one of the tellers. He was immediately taken to the police station.
    - Who was taken to the police station?
    - The customer / the teller

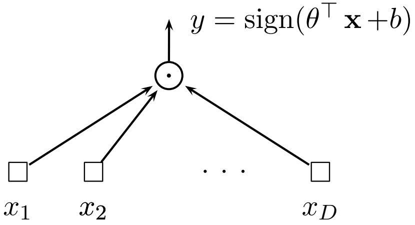
Job Market

- Research groups in universities, governmental research labs, large enterprises.
- In recent years, demand for computational linguists has risen due to the increase of
  - language technology products in the Internet;
  - intelligent systems with access to linguistic means

Outline

Linked Neurons

Perceptron Model

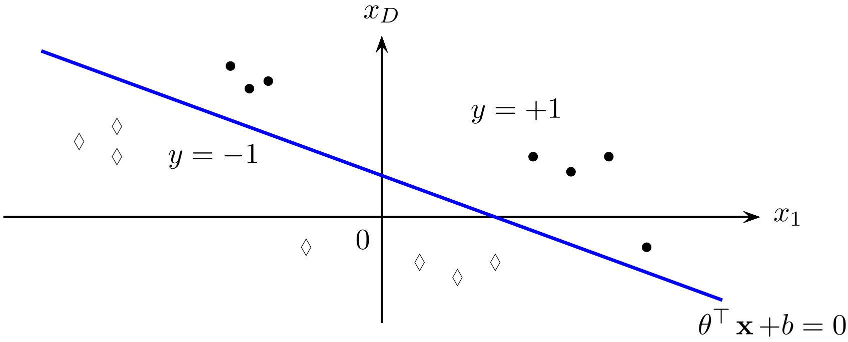


- The most simple ANN with only one neuron (unit), proposed in 1957 by Frank Rosenblatt.
- It is a linear classification model, where the linear function to prediction the class of each datum $\mathbf{x}$ defined as:

$$y = \begin{cases} +1 & \text{if } \theta^\top \mathbf{x} + b > 0 \\ -1 & \text{otherwise} \end{cases}$$

Perceptron Model

Each perceptron separates a space $\mathcal{X}$ into two halves by hyperplane $\theta^\top \mathbf{x} + b$.



Perceptron Model

- Add the intercept feature $x_0 \equiv 1$ and intercept parameter θ_0, the decision boundary is

$$h_\theta(\mathbf{x}) = \text{sign}(\theta_0 + \theta_1 x_1 + \cdots + \theta_D x_D) = \text{sign}(\theta^\top \mathbf{x})$$

- The parameter vector of the model:

$$\theta = \begin{pmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_D \end{pmatrix} \in \mathbb{R}^{D+1}.$$

Parameter Estimation

- We are given a training set $\{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_N, y_N)\}$.
- We would like to find θ that minimizes the *training error*:

$$\widehat{E}(\theta) = \frac{1}{N}\sum_{i=1}^{N}\left[1 - \delta(y_i, h_\theta(\mathbf{x}_i))\right] = \frac{1}{N}\sum_{i=1}^{N} L(y_i, h_\theta(\mathbf{x}_i)),$$

  where
  - $\delta(y, y') = 1$ if $y = y'$ and 0 otherwise;
  - $L(y_i, h_\theta(\mathbf{x}_i))$ is *zero-one* loss.
- What would be a reasonable algorithm for setting the θ?

Parameter Estimation

- Idea: We can just incrementally adjust the parameters so as to correct any mistakes that the corresponding classifier makes.
- Such an algorithm would reduce the training error that counts the mistakes.
- The simplest algorithm of this type is *the perceptron update rule*.

Parameter Estimation

- We consider each training example one by one, cycling through all the examples, and adjust the parameters according to

$$\theta' \leftarrow \theta + y_i \, \mathbf{x}_i \;\text{ if } y_i \neq h_\theta(\mathbf{x}_i).$$

- That is, the parameter vector is changed only if we make a mistake.
- These updates tend to correct the mistakes.

Parameter Estimation

- When we make a mistake

$$\text{sign}(\theta^T \mathbf{x}_i) \neq y_i \Rightarrow y_i(\theta^T \mathbf{x}_i) < 0.$$

- The updated parameters are given by

$$\theta' = \theta + y_i \mathbf{x}_i$$

- If we consider classifying the same example after the update, then

$$\begin{aligned} y_i \theta'^T \mathbf{x}_i &= y_i(\theta + y_i \mathbf{x}_i)^T \mathbf{x}_i \\ &= y_i \theta^T \mathbf{x}_i + y_i^2 \mathbf{x}_i^T \mathbf{x}_i \\ &= y_i \theta^T \mathbf{x}_i + \| \mathbf{x}_i \|^2. \end{aligned}$$

- That is, the value of $y_i \theta^T \mathbf{x}_i$ *increases* as a result of the update (become *more positive* or *more correct*).

Parameter Estimation

Algorithm 1: Perceptron Algorithm

Data: $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_N, y_N)$, $y_i \in \{-1, +1\}$

Result: θ

$\theta \leftarrow 0$;

for $t \leftarrow 1$ *to* T **do**

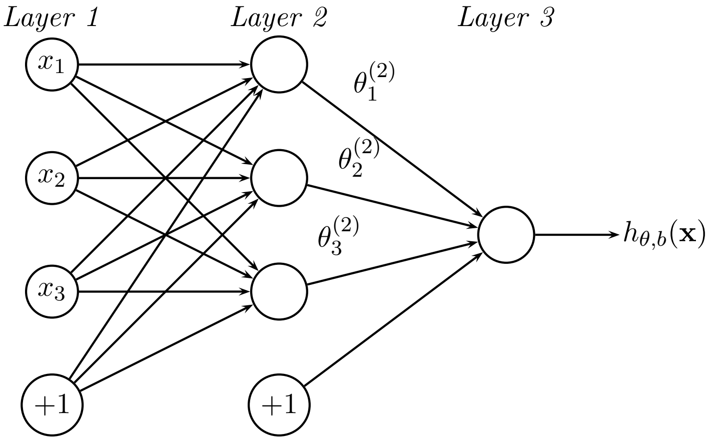
  for $i \leftarrow 1$ *to* N **do**

    $\widehat{y}_i \leftarrow h_\theta(\mathbf{x}_i)$;

    if $\widehat{y}_i \neq y_i$ **then**

      $\theta \leftarrow \theta + y_i \, \mathbf{x}_i$;

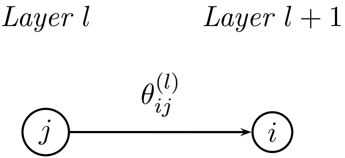
Multi-layer Perceptron



- Many perceptrons stacked into layers.
- Fancy name: Artificial Neural Networks (ANN)

Multi-layer Perceptron

- Let n be the number of layers ($n = 3$ in the previous ANN).
- Let L_l denote the l-th layer; L_1 is input layer, L_n is output layer.
- Parameters: $(\theta, b) = (\theta^{(1)}, b^{(1)}, \theta^{(2)}, b^{(2)})$ where $\theta_{ij}^{(l)}$ represents the parameter associated with the arc from neuron j of layer l to neuron i of layer $l+1$.



- $b_i^{(l)}$ is the bias term of neuron i in layer l.

Multi-layer Perceptron

The ANN above has the following parameters:

$$\theta^{(1)} = \begin{pmatrix} \theta_{11}^{(1)} & \theta_{12}^{(1)} & \theta_{13}^{(1)} \\ \theta_{21}^{(1)} & \theta_{22}^{(1)} & \theta_{23}^{(1)} \\ \theta_{31}^{(1)} & \theta_{32}^{(1)} & \theta_{33}^{(1)} \end{pmatrix} \qquad \theta^{(2)} = \begin{pmatrix} \theta_{11}^{(2)} & \theta_{12}^{(2)} & \theta_{13}^{(2)} \end{pmatrix}$$

$$b^{(1)} = \begin{pmatrix} b_1^{(1)} \\ b_2^{(1)} \\ b_3^{(1)} \end{pmatrix} \qquad b^{(2)} = \begin{pmatrix} b_1^{(2)} \end{pmatrix}.$$

Multi-layer Perceptron

- We call $a_i^{(l)}$ *activation* (which means output value) of neuron i in layer l.
- If $l = 1$ then $a_i^{(1)} \equiv x_i$.
- The ANN computes an output value as follows:

$$
\begin{aligned}
a_i^{(1)} &= x_i, \qquad \forall i = 1, 2, 3; \\
a_1^{(2)} &= f\left(\theta_{11}^{(1)} a_1^{(1)} + \theta_{12}^{(1)} a_2^{(1)} + \theta_{13}^{(1)} a_3^{(1)} + b_1^{(1)}\right) \\
a_2^{(2)} &= f\left(\theta_{21}^{(1)} a_1^{(1)} + \theta_{22}^{(1)} a_2^{(1)} + \theta_{23}^{(1)} a_3^{(1)} + b_2^{(1)}\right) \\
a_3^{(2)} &= f\left(\theta_{31}^{(1)} a_1^{(1)} + \theta_{32}^{(1)} a_2^{(1)} + \theta_{33}^{(1)} a_3^{(1)} + b_3^{(1)}\right) \\
a_1^{(3)} &= f\left(\theta_{11}^{(2)} a_1^{(2)} + \theta_{12}^{(2)} a_2^{(2)} + \theta_{13}^{(2)} a_3^{(2)} + b_1^{(2)}\right).
\end{aligned}
$$

where $f(\cdot)$ is an *activation function*.

Multi-layer Perceptron

- Denote $z_i^{(l+1)} = \sum_{j=1}^{3} \theta_{ij}^{(l)} a_j^{(l)} + b_i^{(l)}$, then $a_i^{(l)} = f(z_i^{(l)})$.
- If we extend f to work with vectors:

$$f((z_1, z_2, z_3)) = (f(z_1), f(z_2), f(z_3))$$

then the activation can be computed compactly by matrix operations:

$$\begin{aligned} z^{(2)} &= \theta^{(1)} a^{(1)} + b^{(1)} \\ a^{(2)} &= f\left(z^{(2)}\right) \\ z^{(3)} &= \theta^{(2)} a^{(2)} + b^{(2)} \\ h_{\theta,b}(\mathbf{x}) &= a^{(3)} = f\left(z^{(3)}\right). \end{aligned}$$

Multi-layer Perceptron

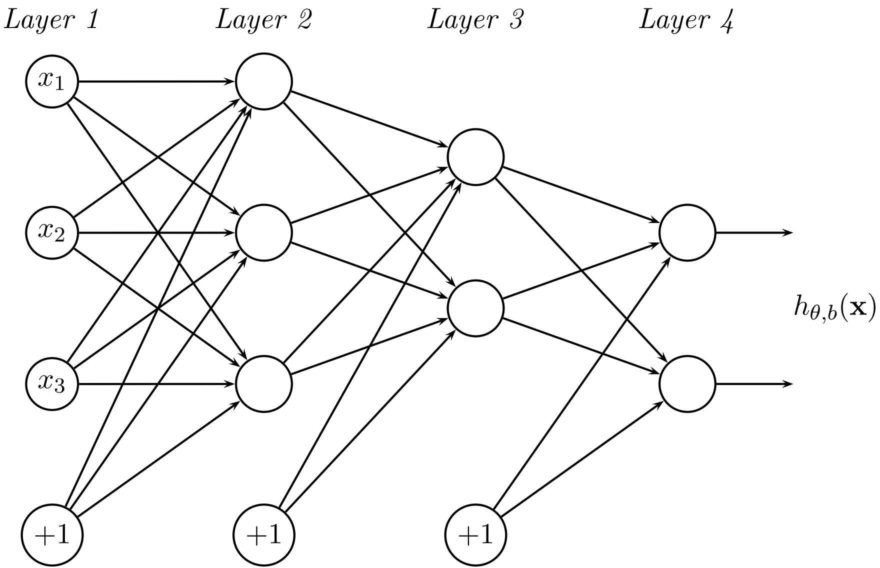
- In a NN with n layers, activations of layer $l+1$ are computed from those of layer l:

$$z^{(l+1)} = \theta^{(l)} a^{(l)} + b^{(l)}$$
$$a^{(l+1)} = f(z^{(l)}).$$

- The final output:

$$h_{\theta,b}(\mathbf{x}) = f\left(z^{(n)}\right).$$

Multi-layer Perceptron



Activation Functions

Commonly used nonlinear activation functions:

- Sigmoid/logistic function:

$$f(z) = \frac{1}{1 + e^{-z}}$$

- Rectifier function:

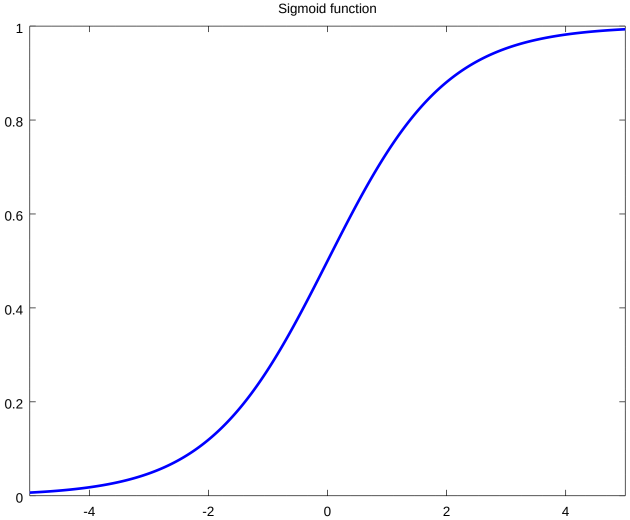
$$f(z) = \max\{0, z\}$$

This activation function has been argued to be more biologically plausible than the logistic function. A smooth approximation to the rectifier is

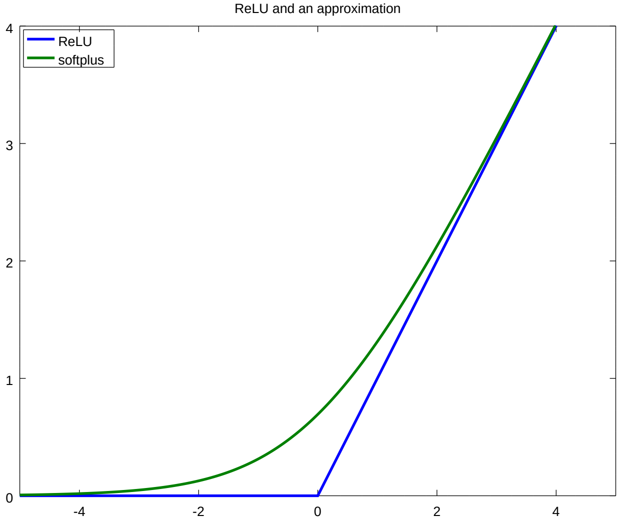
$$f(z) = \ln(1 + e^{z})$$

Note that its derivative is the logistic function.

Sigmoid Activation Function



ReLU Activation Function



Training a MLP

- Suppose that the training dataset has N examples:

$$\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_N, y_N)\}.$$

- A MLP can be trained by using an optimization algorithm.
- For each example $(\mathbf{x}, y)$, denote its associated loss function as $J(\mathbf{x}, y; \theta, b)$. The overall loss function is

$$J(\theta, b) = \frac{1}{N}\sum_{i=1}^{N} J(\mathbf{x}_i, y_i; \theta, b) + \underbrace{\frac{\lambda}{2N}\sum_{l=1}^{n-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(\theta_{ji}^{(l)}\right)^2}_{\text{regularization term}}$$

where s_l is the number of units in layer l.

Loss Function

Two widely used loss functions:

1. Squared error:

$$J(\mathbf{x}, y; \theta, b) = \frac{1}{2}\|y - h_{\theta,b}(\mathbf{x})\|^2.$$

2. Cross-entropy:

$$J(\mathbf{x}, y; \theta, b) = -\left[y \log(h_{\theta,b}(\mathbf{x})) + (1-y)\log(1 - h_{\theta,b}(\mathbf{x}))\right],$$

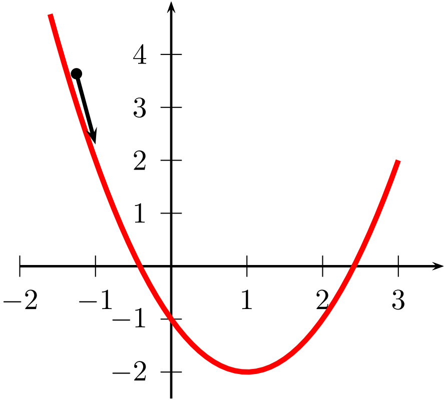
where $y \in \{0, 1\}$.

Gradient Descent Algorithm

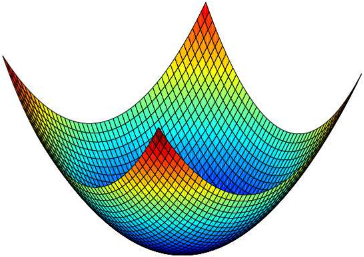
- Training the model is to find values of parameters θ, b minimizing the loss function:

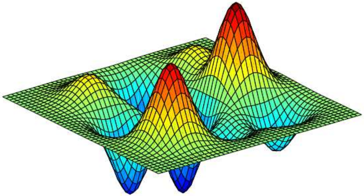
$$J(\theta, b) \to \min.$$

- The most simple optimization algorithm is Gradient Descent.



Gradient Descent Algorithm





- Since $J(\theta, b)$ is not a convex function, the optimal value may not be the globally optimal one.
- However in practice, the gradient descent algorithm is usually able to find a good model if the parameters are initialized properly.

Gradient Descent Algorithm

In each iteration, the gradient descent algorithm updates parameters θ, b as follows:

$$\theta_{ij}^{(l)} = \theta_{ij}^{(l)} - \alpha \frac{\partial}{\partial \theta_{ij}^{(l)}} J(\theta, b)$$

$$b_i^{(l)} = b_i^{(l)} - \alpha \frac{\partial}{\partial b_i^{(l)}} J(\theta, b),$$

where α is a learning rate.

Gradient Descent Algorithm

We have

$$\frac{\partial}{\partial \theta_{ij}^{(l)}} J(\theta, b) = \frac{1}{N} \left[\sum_{i=1}^{N} \frac{\partial}{\partial \theta_{ij}^{(l)}} J(\mathbf{x}_i, y_i; \theta, b) + \lambda \theta_{ij}^{(l)} \right]$$

$$\frac{\partial}{\partial b_{i}^{(l)}} J(\theta, b) = \frac{1}{N} \sum_{i=1}^{N} \frac{\partial}{\partial b_{i}^{(l)}} J(\mathbf{x}_i, y_i; \theta, b).$$

Here, we need to compute partial derivatives

$$\frac{\partial}{\partial \theta_{ij}^{(l)}} J(\mathbf{x}_i, y_i; \theta, b), \quad \frac{\partial}{\partial b_{i}^{(l)}} J(\mathbf{x}_i, y_i; \theta, b)$$

How can we compute efficiently these partial derivatives? By using the back-propagation algorithm.

Outline

Thuật toán lan truyền ngược

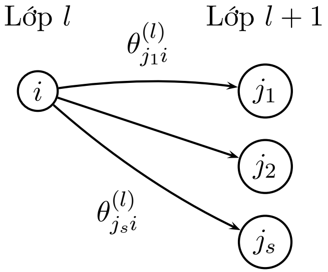
- Trước tiên, với mỗi dữ liệu $(\mathbf{x}, y)$, ta tính toán tiến qua mạng nơ-ron để tìm mọi kích hoạt, gồm cả giá trị ra $h_{\theta,b}(\mathbf{x})$.
- Với mỗi đơn vị i của lớp l, ta tính một giá trị gọi là sai số $\varepsilon_i^{(l)}$, đo phần đóng góp của đơn vị đó vào tổng sai số của đầu ra.
- Với lớp ra $l = n$, ta có thể trực tiếp tính được $\varepsilon_i^{(n)}$ với mọi đơn vị i của lớp ra bằng cách tính độ lệch của kích hoạt tại đơn vị i đó so với giá trị đúng. Cụ thể là, với mọi $i = 1, 2, \dots, s_n$:

$$\begin{aligned}
\varepsilon_i^{(n)} &= \frac{\partial}{\partial z_i^{(n)}} \frac{1}{2} \|y - f(z_i^{(n)})\|^2 \\
&= -(y_i - f(z_i^{(n)})) f'(z_i^{(n)}) \\
&= -(y_i - a_i^{(n)}) a_i^{(n)} (1 - a_i^{(n)}).
\end{aligned}$$

Thuật toán lan truyền ngược

- Với mỗi đơn vị ẩn, $\varepsilon_i^{(l)}$ được xác định là trung bình có trọng trên các sai số của các đơn vị của lớp tiếp theo có sử dụng đơn vị này để làm đầu vào.

$$\varepsilon_i^{(l)} = \left(\sum_{j=1}^{s_{l+1}} \theta_{ji}^{(l)} \varepsilon_j^{(l+1)} \right) f'(z_i^{(l)}).$$



Thuật toán lan truyền ngược

1. Tính toán tiến, tính mọi kích hoạt của các lớp $L_2, L_3 \dots, L_n$.
2. Với mỗi đơn vị ra i của lớp ra L_n, tính

$$\varepsilon_i^{(n)} = -(y_i - a_i^{(n)})a_i^{(n)}(1 - a_i^{(n)}).$$

3. Tính các sai số theo thứ tự ngược: với mọi lớp $l = n-1, \dots, 2$ và với mọi đơn vị i của lớp l, tính

$$\varepsilon_i^{(l)} = \left(\sum_{j=1}^{s_{l+1}} \theta_{ji}^{(l)} \varepsilon_j^{(l+1)} \right) f'(z_i^{(l)}).$$

4. Tính các đạo hàm riêng cần tìm như sau:

$$\frac{\partial}{\partial \theta_{ij}^{(l)}} J(\mathbf{x}, y; \theta, b) = \varepsilon_i^{(l+1)} a_j^{(l)}$$

$$\frac{\partial}{\partial b_i^{(l)}} J(\mathbf{x}, y; \theta, b) = \varepsilon_i^{(l+1)}.$$

Thuật toán lan truyền ngược

- Ta có thể biểu diễn thuật toán trên ngắn gọn hơn thông qua các phép toán trên ma trận.
- Kí hiệu $\bullet$ là toán tử nhân từng phần tử của các véc-tơ, định nghĩa như sau:[1]

$$\mathbf{x} = (x_1, \dots, x_D), \mathbf{y} = (y_1, \dots, y_D) \Rightarrow \mathbf{x} \bullet \mathbf{y} = (x_1 y_1, x_2 y_2, \dots, x_D y_D).$$

- Tương tự, ta mở rộng các hàm $f(\cdot), f'(\cdot)$ cho từng thành phần của véc-tơ. Ví dụ:

$$f(\mathbf{x}) = (f(x_1), f(x_2), \dots, f(x_D))$$
$$f'(\mathbf{x}) = \left(\frac{\partial}{\partial x_1} f(x_1), \frac{\partial}{\partial x_2} f(x_2), \dots, \frac{\partial}{\partial x_D} f(x_D) \right).$$

[1] Trong Matlab/Octave thì $\bullet$ là phép toán ".∗", còn gọi là tích Hadamard.

Thuật toán lan truyền ngược

1. Thực hiện tính toán tiến, tính mọi kích hoạt của các lớp $L_2, L_3 \ldots$ cho tới lớp ra L_n:
$$z^{(l+1)} = \theta^{(l)} a^{(l)} + b^{(l)}$$
$$a^{(l+1)} = f(z^{(l)}).$$
2. Với lớp ra L_n, tính
$$\varepsilon^{(n)} = -(y - a^{(n)}) \bullet f'(z^{(n)}).$$
3. Với mọi lớp $l = n-1, n-2, \ldots, 2$, tính
$$\varepsilon^{(l)} = \left((\theta^{(l)})^T \varepsilon^{(l+1)}\right) \bullet f'(z^{(l)}).$$
4. Tính các đạo hàm riêng cần tìm như sau:
$$\frac{\partial}{\partial \theta^{(l)}} J(\mathbf{x}, y; \theta, b) = \varepsilon^{(l+1)} \left(a^{(l)}\right)^T$$
$$\frac{\partial}{\partial b^{(l)}} J(\mathbf{x}, y; \theta, b) = \varepsilon^{(l+1)}.$$

Gradient Descent Algorithm

Algorithm 2: Thuật toán giảm gradient huấn luyện mạng nơ-ron

for $l = 1$ *to* n **do**

  $\nabla\theta^{(l)} \leftarrow 0; \quad \nabla b^{(l)} \leftarrow 0;$

for $i = 1$ *to* N **do**

  Tính $\frac{\partial}{\partial\theta^{(l)}} J(\mathbf{x}_i, y_i; \theta, b)$ và $\frac{\partial}{\partial b^{(l)}} J(\mathbf{x}_i, y_i; \theta, b)$;

  $\nabla\theta^{(l)} \leftarrow \nabla\theta^{(l)} + \frac{\partial}{\partial\theta^{(l)}} J(\mathbf{x}_i, y_i; \theta, b);$

  $\nabla b^{(l)} \leftarrow \nabla b^{(l)} + \frac{\partial}{\partial b^{(l)}} J(\mathbf{x}_i, y_i; \theta, b);$

$\theta^{(l)} \leftarrow \theta^{(l)} - \alpha \left(\frac{1}{N}\nabla\theta^{(l)} + \frac{\lambda}{N}\theta^{(l)}\right);$

$b^{(l)} \leftarrow b^{(l)} - \alpha \left(\frac{1}{N}\nabla b^{(l)}\right);$

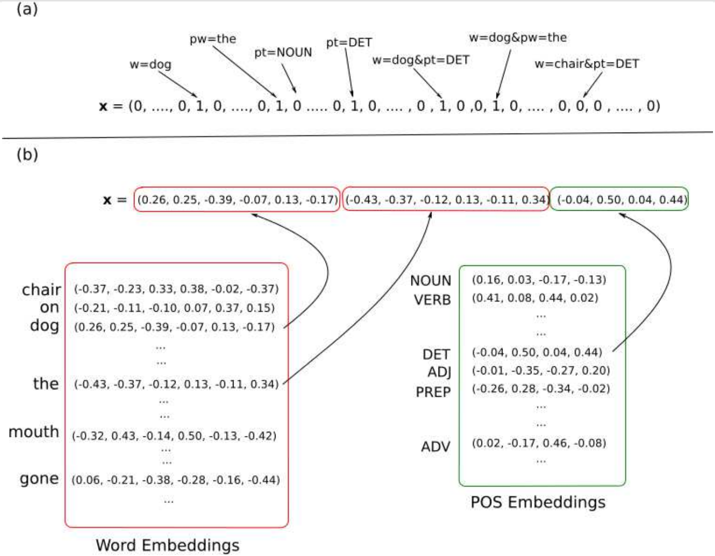
Kí hiệu $\nabla\theta^{(l)}$ là ma trận gradient của $\theta^{(l)}$ (cùng số chiều với $\theta^{(l)}$) và $\nabla b^{(l)}$ là véc-tơ gradient của $b^{(l)}$ (cùng số chiều với $b^{(l)}$).

Outline

Distributed Word Representations

- One-hot vector representation: $\vec{v}_w = (0, 0, \ldots, 1, \ldots, 0, 0) \in \{0, 1\}^{|\mathcal{V}|}$, where $|\mathcal{V}|$ is the size of a dictionary $\mathcal{V}$.
- $\mathcal{V}$ is large (e.g., 100K)
- Try to represent w in a vector space of much lower dimension, $\vec{v}_w \in \mathbb{R}^d$ (e.g., $D = 300$).

Distributed Word Representations

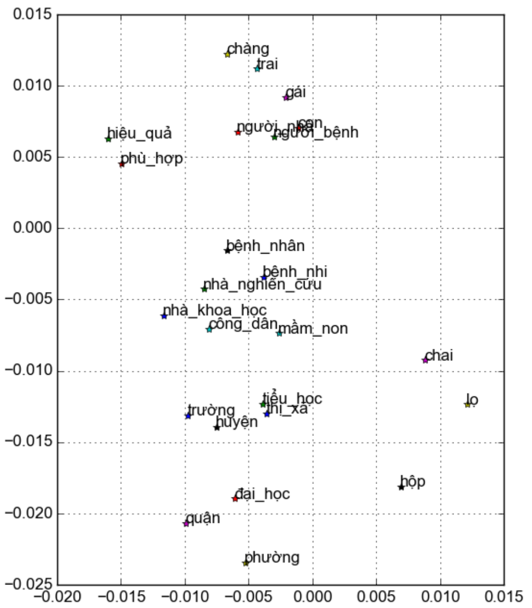


(Yoav Goldberg, 2015)

Distributed Word Representations

- Word vectors are essentially *feature extractors* that encode *semantic features* of words in their dimensions.
- Semantically close words are likewise close (in Euclidean or cosine distance) in the lower dimensional vector space.

Distributed Word Representations



Distributed Representation Models

- CBOW model[2]
- Skip-gram model[3]
- Global Vector (GloVe)[4]

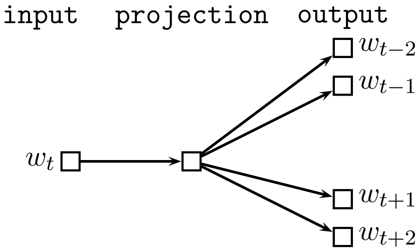
[2] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *Proceedings of Workshop at ICLR*, Scottsdale, Arizona, USA, 2013

[3] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in Neural Information Processing Systems 26*, C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger, Eds. Curran Associates, Inc., 2013, pp. 3111–3119

[4] J. Pennington, R. Socher, and C. Manning, "Glove: Global vectors for word representation," in *Proceedings of EMNLP*, Doha, Qatar, 2014, pp. 1532–1543

Skip-gram Model

- A sliding window approach, looking at a sequence of $2k + 1$ words.
  - The middle word is called the *focus word* or *central word*.
  - The k words to each side are the *contexts*.
- Prediction of surrounding words given the current word, that is to model $P(c|w)$.
- This approach is referred to as a *skip-gram* model.



Skip-gram Model

- Skip-gram seeks to represent each word w and each context c as a d-dimensional vector $\vec{w}$ and $\vec{c}$.
- Intuitively, it *maximizes* a function of the product $\langle \vec{w}, \vec{c} \rangle$ for (w, c) pairs in the training set and *minimizes* it for negative examples (w, c_N).
- The negative examples are created by randomly corrupting observed (w, c) pairs (*negative sampling*).
- The model draws k contexts from the empirical unigram distribution $\widehat{P}(c)$ which is *smoothed*.

Skip-gram Model – Technical Details

- Maximize the average conditional log probability

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{j=-c}^{c}\log p(w_{t+j}|w_t),$$

where $\{w_i : i \in T\}$ is the whole training set, w_t is the central word and the w_{t+j} are on either side of the context.

- The conditional probabilities are defined by the softmax function

$$p(a|b) = \frac{\exp(o_a^\top i_b)}{\sum_{w\in\mathcal{V}} \exp(o_w^\top i_b)},$$

where i_w and o_w are the input and output vector of w respectively, and $\mathcal{V}$ is the vocabulary.

Skip-gram Model – Technical Details

- For computational efficiency, Mikolov's training code approximates the softmax function by the hierarchical softmax, as defined in
  - F. Morin and Y. Bengio, "Hierarchical probabilistic neural network language model," in *Proceedings of AISTATS*, Barbados, 2005, pp. 246–252
- The hierarchical softmax is built on a binary Huffman tree with one word at each leaf node.

Skip-gram Model – Technical Details

- The conditional probabilities are calculated as follows:

$$p(a|b) = \prod_{i=1}^{l} p(d_i(a)|d_1(a)...d_{i-1}(a), b),$$

  where l is the path length from the root to the node a, and $d_i(a)$ is the decision at step i on the path:
  - 0 if the next node is the left child of the current node
  - 1 if it is the right child
- If the tree is balanced, the hierarchical softmax only needs to compute around $\log_2 |\mathcal{V}|$ nodes in the tree, while the true softmax requires computing over all $|\mathcal{V}|$ words.
- This technique is used for learning word vectors from huge data sets with *billions* of words, and with *millions* of words in the vocabulary.

Skip-gram Model

- Skip-gram model has been recently shown to be equivalent to an implicit *matrix factorization* method[5] where its objective function achieves its optimal value when

$$\langle \vec{w}, \vec{c} \rangle = \mathrm{PMI}(w, c) - \log k,$$

where the PMI measures the association between the word w and the context c:

$$\mathrm{PMI}(w, c) = \log \frac{\widehat{P}(w, c)}{\widehat{P}(w)\widehat{P}(c)}.$$

[5] O. Levy, Y. Goldberg, and I. Dagan, "Improving distributional similarity with lessons learned from word embeddings," *Transaction of the ACL*, vol. 3, pp. 211–225, 2015

GloVe Model

- Similar to the Skip-gram model, GloVe is a local context window method but it has the advantages of the global matrix factorization method.
- The main idea of GloVe is to use word-word occurrence counts to estimate the co-occurrence probabilities rather than the probabilities by themselves.
- Let P_{ij} denote the probability that word j appear in the context of word i; $\vec{w}_i \in \mathbb{R}^d$ and $\vec{w}_j \in \mathbb{R}^d$ denote the word vectors of word i and word j respectively. It is shown that

$$\vec{w}_i^\top \vec{w}_j = \log(P_{ij}) = \log(C_{ij}) - \log(C_i),$$

where C_{ij} is the number of times word j occurs in the context of word i.

GloVe Model

- It turns out that GloVe is a *global log-bilinear* regression model.
- Finding word vectors is equivalent to solving a weighted least-squares regression model with the cost function:

$$J = \sum_{i,j=1}^{|\mathcal{V}|} f(C_{ij})(\vec{w}_i^\top \vec{w}_j + b_i + b_j - \log(C_{ij}))^2,$$

  where b_i and b_j are additional bias terms and $f(C_{ij})$ is a weighting function.
- A class of weighting functions which are found to work well can be parameterized as

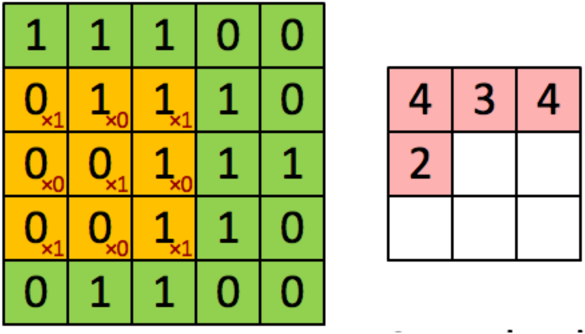
$$f(x) = \begin{cases} \left(\frac{x}{x_{\max}}\right)^\alpha & \text{if } x < x_{\max} \\ 1 & \text{otherwise} \end{cases}$$

Outline

Convolutional Neural Networks – CNN

- A CNN is a feed-forward neural network *with convolution layers interleaved with pooling layers.*
- In a *convolution layer*, a small region of data (a small square of image, a text phrase) at every location is converted to a low-dimensional vector (an embedding).
  - The *embedding function* is shared among all the locations, so that useful features can be detected irrespective of their locations.
- In a *pooling layer*, the region embeddings are aggregated to a global vector (representing an image, a document) by taking component-wise maximum or average
  - max pooling / average pooling.
- Map-Reduce approach!

Convolutional Neural Networks – CNN



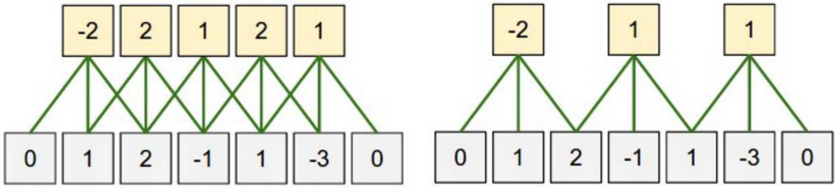
- The sliding window is called a *kernel*, *filter* or *feature detector*.

Convolutional Neural Networks – CNN

- Originally developed for image processing, CNN models have subsequently been shown to be effective for NLP and have achieved excellent results in:
  - semantic parsing (Yih et al., ACL 2014)
  - search query retrieval (Shen et al., WWW 2014)
  - sentence modelling (Kalchbrenner et al., ACL 2014)
  - sentence classification (Y. Kim, EMNLP 2014)
  - text classification (Zhang et al., NIPS 2015)
  - other traditional NLP tasks (Collobert et al., JMLR 2011)

Stride Size

- Stride size is a hyperparameter of CNN which defines by how much we want to shift our filter at each step.
- Stride sizes of 1 and 2 applied to 1-dimensional input:[6]

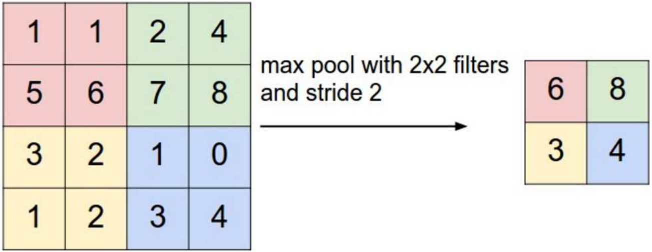


- The larger is stride size, the fewer applications of the filter and smaller output size are.

[6] http://cs231n.github.io/convolutional-networks/

Pooling Layers

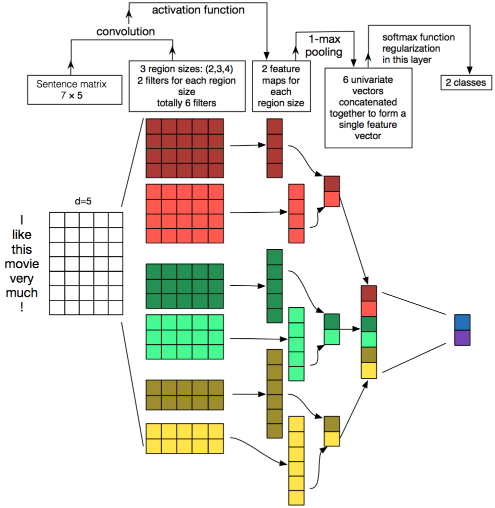
- Pooling layers are a key aspect of CNN, which are applied after the convolution layers.
- Pooling layers *subsample* their input.
- We can either pool over a window or over the complete output.



Why Pooling?

- Pooling provides a fixed size output matrix, which is typically required for classification.
    - 10K filters → max pooling → 10K-dimensional output, regardless of the size of the filters, or the size of the input.
- Pooling reduces the output dimensionality but keeps the most “salient” information (feature detection)
- Pooling provides basic invariance to shifting and rotation, which is useful in image recognition.
- However, max pooling loses global information about locality of features, just like a bag of n-grams model.

CNN for NLP


activation function
convolution
1-max pooling
softmax function regularization in this layer
Sentence matrix 7 × 5
3 region sizes: (2,3,4) 2 filters for each region size totally 6 filters
2 feature maps for each region size
6 univariate vectors concatenated together to form a single feature vector
2 classes
d=5
I
like
this
movie
very
much
!

Convolutional Module – Technical Details

A simple 1-d convolution:

- A discrete input function: $g(x) : [1, l] \to \mathbb{R}$
- A discrete kernel function: $f(x) : [1, k] \to \mathbb{R}$
- The convolution between $f(x)$ and $g(x)$ with stride d is defined as:

$$h(y) : [1, (l - k + 1)/d] \to \mathbb{R}$$

$$h(y) = \sum_{x=1}^{k} f(x) \cdot g(y \cdot d - x + c),$$

  where $c = k - d + 1$ is an offset constant.
- A set of kernel functions $f_{ij}(x), \forall i = 1, 2, \dots, m$ and $\forall j = 1, 2, \dots, n$, which are called *weights*.
- g_i are input features, h_j are output features.

Max-Pooling Module – Technical Details

- A discrete input function: $g(x) : [1, l] \to \mathbb{R}$
- Max-pooling function is defined as

$$h(y) : [1, (l - k + 1)/d] \to \mathbb{R}$$
$$h(y) = \max_{x=1}^{k} g(y \cdot d - x + c),$$

where $c = k - d + 1$ is an offset constant.

Building a CNN Architecture

There are many hyperparameters to choose:

- Input representations (one-hot, distributed)
- Number of layers
- Number and size of convolution filters
- Pooling strategies (max, average, other)
- Activation functions (ReLU, sigmoid, tanh)
- Regularization methods (dropout?)

Outline

Sentence Classification

- Y. Kim[7] reports experiments with CNN trained on top of pre-trained word vectors for sentence-level classification tasks.
- CNN achieved excellent results on multiple benchmarks, improved upon the state of the art on 4 out of 7 tasks, including sentiment analysis and question classification.

[7] Y. Kim, "Convolutional neural networks for sentence classification," in *Proceedings of EMNLP*. Doha, Quatar: ACL, 2014, pp. 1746–1751

Sentence Classification

Model	MR	SST-1	SST-2	Subj	TREC	CR	MPQA
CNN-rand	76.1	45.0	82.7	89.6	91.2	79.8	83.4
CNN-static	81.0	45.5	86.8	93.0	92.8	84.7	**89.6**
CNN-non-static	**81.5**	48.0	87.2	93.4	93.6	84.3	89.5
CNN-multichannel	81.1	47.4	**88.1**	93.2	92.2	**85.0**	89.4
RAE (Socher et al., 2011)	77.7	43.2	82.4	–	–	–	86.4
MV-RNN (Socher et al., 2012)	79.0	44.4	82.9	–	–	–	–
RNTN (Socher et al., 2013)	–	45.7	85.4	–	–	–	–
DCNN (Kalchbrenner et al., 2014)	–	48.5	86.8	–	93.0	–	–
Paragraph-Vec (Le and Mikolov, 2014)	–	**48.7**	87.8	–	–	–	–
CCAE (Hermann and Blunsom, 2013)	77.8	–	–	–	–	–	87.2
Sent-Parser (Dong et al., 2014)	79.5	–	–	–	–	–	86.3
NBSVM (Wang and Manning, 2012)	79.4	–	–	93.2	–	81.8	86.3
MNB (Wang and Manning, 2012)	79.0	–	–	**93.6**	–	80.0	86.3
G-Dropout (Wang and Manning, 2013)	79.0	–	–	93.4	–	82.1	86.1
F-Dropout (Wang and Manning, 2013)	79.1	–	–	**93.6**	–	81.9	86.3
Tree-CRF (Nakagawa et al., 2010)	77.3	–	–	–	–	81.4	86.1
CRF-PR (Yang and Cardie, 2014)	–	–	–	–	–	82.7	–
SVM_S (Silva et al., 2011)	–	–	–	–	**95.0**	–	–

Character-level CNN for Text Classification

- Zhang et al.[8] presents an empirical exploration on the use of character-level CNN for text classification.
- Performance of the model depends on many factors: dataset size, choice of alphabet, etc.
- Datasets:

Dataset	Classes	Train Samples	Test Samples	Epoch Size
AG's News	4	120,000	7,600	5,000
Sogou News	5	450,000	60,000	5,000
DBPedia	14	560,000	70,000	5,000
Yelp Review Polarity	2	560,000	38,000	5,000
Yelp Review Full	5	650,000	50,000	5,000
Yahoo! Answers	10	1,400,000	60,000	10,000
Amazon Review Full	5	3,000,000	650,000	30,000
Amazon Review Polarity	2	3,600,000	400,000	30,000

[8] X. Zhang, J. Zhao, and Y. LeCun, "Character-level convolutional networks for text classification," in *Proceedings of NIPS*, Montreal, Canada, 2015

Character-level CNN for Text Classification

Table 4: Testing errors of all the models. Numbers are in percentage. "Lg" stands for "large" and "Sm" stands for "small". "w2v" is an abbreviation for "word2vec", and "Lk" for "lookup table". "Th" stands for thesaurus. ConvNets labeled "Full" are those that distinguish between lower and upper letters

Model	AG	Sogou	DBP.	Yelp P.	Yelp F.	Yah. A.	Amz. F.	Amz. P.
BoW	11.19	7.15	3.39	7.76	42.01	31.11	45.36	9.60
BoW TFIDF	10.36	6.55	2.63	6.34	40.14	28.96	44.74	9.00
ngrams	7.96	2.92	1.37	**4.36**	43.74	31.53	45.73	7.98
ngrams TFIDF	**7.64**	**2.81**	**1.31**	4.56	45.20	31.49	47.56	8.46
Bag-of-means	**16.91**	**10.79**	**9.55**	**12.67**	**47.46**	**39.45**	**55.87**	**18.39**
LSTM	13.94	4.82	1.45	5.26	41.83	29.16	40.57	6.10
Lg. w2v Conv.	9.92	4.39	1.42	4.60	40.16	31.97	44.40	5.88
Sm. w2v Conv.	11.35	4.54	1.71	5.56	42.13	31.50	42.59	6.00
Lg. w2v Conv. Th.	9.91	-	1.37	4.63	39.58	31.23	43.75	5.80
Sm. w2v Conv. Th.	10.88	-	1.53	5.36	41.09	29.86	42.50	5.63
Lg. Lk. Conv.	8.55	4.95	1.72	4.89	40.52	29.06	45.95	5.84
Sm. Lk. Conv.	10.87	4.93	1.85	5.54	41.41	30.02	43.66	5.85
Lg. Lk. Conv. Th.	8.93	-	1.58	5.03	40.52	28.84	42.39	5.52
Sm. Lk. Conv. Th.	9.12	-	1.77	5.37	41.17	28.92	43.19	5.51
Lg. Full Conv.	9.85	8.80	1.66	5.25	38.40	29.90	40.89	5.78
Sm. Full Conv.	11.59	8.95	1.89	5.67	38.82	30.01	40.88	5.78
Lg. Full Conv. Th.	9.51	-	1.55	4.88	38.04	29.58	40.54	5.51
Sm. Full Conv. Th.	10.89	-	1.69	5.42	**37.95**	29.90	40.53	5.66
Lg. Conv.	12.82	4.88	1.73	5.89	39.62	29.55	41.31	5.51
Sm. Conv.	15.65	8.65	1.98	6.53	40.84	29.84	40.53	5.50
Lg. Conv. Th.	13.39	-	1.60	5.82	39.30	**28.80**	40.45	**4.93**
Sm. Conv. Th.	14.80	-	1.85	6.49	40.16	29.84	**40.43**	5.67

Outline

Relation Extraction

- Learning to extract semantic relations between entity pairs from text
- Many applications:
  - information extraction
  - knowledge base population
  - question answering
- Example:
  - `In the morning, the President traveled to Detroit` → `travelTo(President, Detroit)`
  - `Yesterday, New York based Foo Inc. announced their acquisition of Bar Corp.` → `mergeBetween(Foo Inc., Bar Corp., date)`
- Two subtasks: Relation extraction (RE) and relation classification (RC)

Relation Extraction

- Datasets:
  - SemEval-2010 Task 8 dataset for RC
  - ACE 2005 dataset for RE
- Class distribution:

ACE 2005 (87,512)		SemEval 2010 (10,717)	
Relation	%	Relation	%
ORG-AFF	2.8	Cause-Effect	12.4
PER-SOC	1.2	Component-Whole	11.7
ART	1.0	Entity-Destination	10.6
PART-WHOLE	1.4	Entity-Origin	9.1
GEN-AFF	1.1	Product-Producer	8.8
PHYS	2.1	Member-Collection	8.6
Other	**90.4**	Message-Topic	8.4
		Content-Container	6.8
		Instrument-Agency	6.2
		Other	**17.4**

Relation Extraction

Performance of Relation Extraction systems[9]

System	P	R	F
Words	54.95	43.73	48.69
Words-WC-Wed	50.10	44.47	47.11
Words-HM-Wed	57.01	55.74	56.36
Our CNN	71.25	53.91	**61.32**

CNN outperforms significantly 3 baseline systems.

[9] T. H. Nguyen and R. Grishman, "Relation extraction: Perspective from convolutional neural networks," in *Proceedings of NAACL Workshop on Vector Space Modeling for NLP*, Denver, Colorado, USA, 2015

Relation Classification

Classifier	Feature Sets	F
SVM	POS, WordNet, morphological features, thesauri, Google *n*-grams	77.7
MaxEnt	POS, WordNet, morphological features, noun compound system, thesauri, Google *n*-grams	77.6
SVM	POS, WordNet, morphological features, dependency parse, Levin classes, PropBank, FrameNet, NomLex-Plus, Google *n*-grams, paraphrases, TextRunner	82.2
CNN	-	**82.8**

CNN does *not use any supervised or manual features* such as POS, WordNet, dependency parse, etc.

Outline

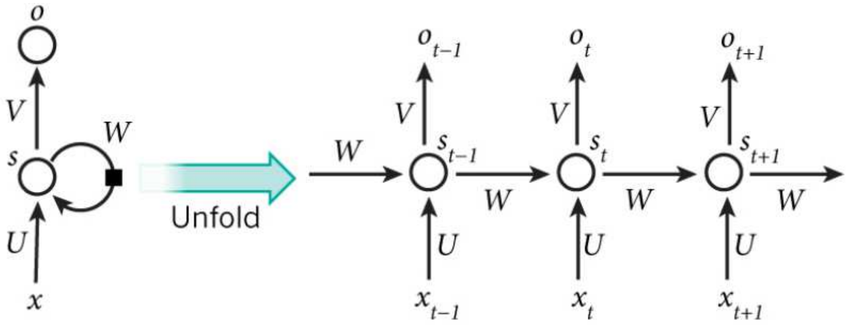
Recurrent Neural Networks – RNN

Recently, RNNs have shown great success in many NLP tasks:

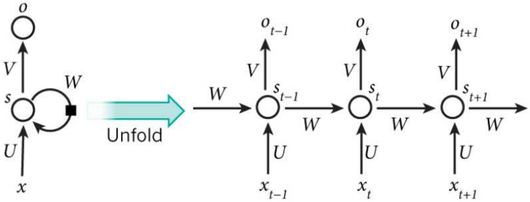
- Language modelling and text generation
- Machine translation
- Speech recognition
- Generating image descriptions

Recurrent Neural Networks – RNN

- The idea behind RNN is to make use of sequential information.
  - We can better predict the next word in a sentence if we know which words came before it.
- RNNs are called *recurrent* because they perform the same task for every element of a sequence.



Recurrent Neural Networks – RNN



- x_t is the input at time step t (one-hot vector / word embedding)
- s_t is the hidden state at time step t, which is caculated using the previous hidden state and the input at the current step:

$$s_t = \tanh(Ux_t + Ws_{t-1})$$

- o_t is the output at step t:

$$o_t = \text{softmax}(Vs_t)$$

Recurrent Neural Networks – RNN

- Assume that we have a vocabulary of 10K words, and a hidden layer size of 100 dimensions.
- Then we have,

$$
\begin{aligned}
x_t &\in \mathbb{R}^{10000} \\
o_t &\in \mathbb{R}^{10000} \\
s_t &\in \mathbb{R}^{100} \\
U &\in \mathbb{R}^{100\times 10000} \\
V &\in \mathbb{R}^{10000\times 100} \\
W &\in \mathbb{R}^{100\times 100}
\end{aligned}
$$

  where U, V and W are parameters of the network we want to learn from data.
- Total number of parameters $= 2,010,000$.

Training RNN

- The most common way to train a RNN is to use Stochastic Gradient Descent (SGD).
- Cross-entropy loss function on a training set:

$$L(y, o) = -\frac{1}{N} \sum_{n=1}^{N} y_n \log o_n$$

- We need to calculate the gradients:

$$\frac{\partial L}{\partial U}, \quad \frac{\partial L}{\partial V}, \quad \frac{\partial L}{\partial W}.$$

- These gradients are computed by using the *back-propagation through time*[10] algorithm, a slightly modified version of the back-propagation algorithm.

[10] P. J. Werbos, "Backpropagation through time: What it does and how to do it," in *Proceedings of the IEEE*, vol. 78, no. 10, 1990, pp. 1550–1560

Training RNN – The Vanishing Gradient Problem

- RNNs have difficulties learning *long-range dependencies* because the gradient values from "far away" steps become zero.
  - `I grew up in France. I speak fluent` *`French`*`.`
- The paper of Pascanu et al.[11] explains in detail the *vanishing* and *exploding* gradient problems when training RNNs.
- A few ways to combat the vanishing gradient problem:
  - Use a proper initialization of the W matrix
  - Use regularization techniques (like dropout)
  - Use ReLU activation functions instead of sigmoid or tanh functions
  - More popular solution: use Long Short Term Memory (LSTM) or Gated Recurrent Unit (GRU) architectures.

[11] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *Proceedings of ICML*, Atlanta, Georgia, USA, 2013

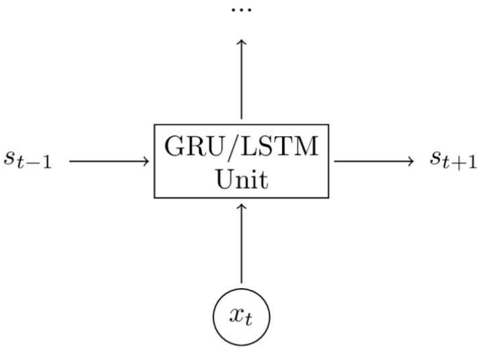
Long-Short Term Memory – LSTM

- LSTMs were first proposed in 1997.[12] They are the most widely used models in DL for NLP today.
- LSTMs use a *gating* mechanism to combat the vanishing gradients.[13]
- GRUs are a simpler variant of LSTMs, first used in 2014.

[12] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997

[13] `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`

Long-Short Term Memory – LSTM



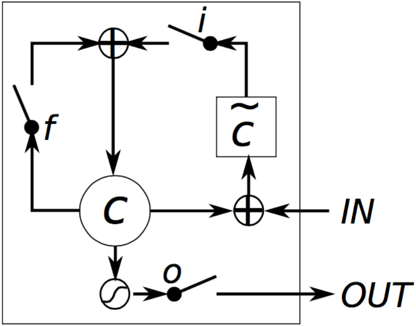
- A LSTM layer is just another way to compute the hidden state.
- Recall: a vanila RNN computes the hidden state as

$$s_t = \tanh(Ux_t + Ws_{t-1})$$

Long-Short Term Memory – LSTM

How LSTM calculates a hidden state s_t:

$$i = \sigma(U^i x_t + W^i s_{t-1})$$
$$f = \sigma(U^f x_t + W^f s_{t-1})$$
$$o = \sigma(U^o x_t + W^o s_{t-1})$$
$$g = \tanh(U^g x_t + W^g s_{t-1})$$
$$c_t = c_{t-1} \cdot f + g \cdot i$$
$$s_t = \tanh(c_t) \cdot o$$



σ is the sigmoid function, which squashes the values in the range $[0, 1]$. Two special cases:

- 0: let nothing through
- 1: let everything though

Outline

Generating Image Description

"man in black shirt is playing guitar."

"two young girls are playing with lego toy."

(http://cs.stanford.edu/people/karpathy/deepimagesent/)

Generating Image Description

“black and white dog jumps over bar.”

“woman is holding bunch of bananas.”

Outline

Language Modelling and Generating Text

- Given a sequence of words we want to predict the probability of each word given the previous words.
- Language models allow us to measure how likely a sentence is
  - an important input for machine translation and speech recognition: high-probability sentences are typically correct
- We get a generative model, which allows us to generate new text by sampling from the output probabilities.

Language Modelling and Generating Text

Samples from the Wikipedia model:

The meaning of life is the tradition of the ancient human reproduction: it is less favorable to the good boy for when to remove her bigger. In the show's agreement unanimously resurfaced. The wild pasteured with consistent street forests were incorporated by the 15th century BE. In 1996 the primary rapford undergoes an effort that the reserve conditioning, written into Jewish cities, sleepers to incorporate the .St Eurasia that activates the population. Mar??a Nationale, Kelli, Zedlat-Dukastoe, Florendon, Ptu's thought is. To adapt in most parts of North America, the dynamic fairy Dan please believes, the free speech are much related to the

(Extracted from[14])

[14] I. Sutskever, J. Martens, and G. Hinton, "Generating text with recurrent neural networks," in *Proceedings of ICML*, Washington, USA, 2011

Language Modelling and Generating Text

Samples from the ML model:

Recurrent network with the Stiefel information for logistic regression methods Along with either of the algorithms previously (two or more skewprecision) is more similar to the model with the same average mismatched graph. Though this task is to be studied under the reward transform, such as (c) and (C) from the training set, based on target activities for articles a ? 2(6) and (4.3). The PHDPic (PDB) matrix of cav'va using the three relevant information contains for tieming measurements. Moreover, because of the therap tor, the aim is to improve the score to the best patch randomly, but for each initially four data sets. As shown in Figure 11, it is more than 100 steps, we used

Language Modelling and Generating Text

Samples from the VietTreebank model:

Khi phát_hiện của anh <unk> vẫn là ĐD “ nhầm tảng ” , không ít nơi nào để làm_ăn tại trung_tâm **xã** <unk>, **huyện** Phước_Sơn, **tỉnh** Ia_Mơ loại bị bắt cá chết , đoạn xúc ào_ào bắn trong tầm bờ tưới .
Nghe những bóng người Trung_Hoa <unk> đỏ trong rừng tìm ra ầm_ầm giày của liệt_sĩ VN (Mỹ dân_tộc và con ngược miền Bắc nát để thi_công từ 1998 đến TP Phật_giáo đã bắt_đầu cung) nên những vòng 15 - 4 ngả biển .

(Extracted from Nguyễn Văn Khánh’s thesis, VNU-Coltech 2016)

Outline

Summary

- Deep Learning is based on a set of algorithms that attempt to model high-level abstractions in data using deep neural networks.
- Deep Learning can replace hand-crafted features with efficient unsupervised or semi-supervised feature learning, and hierarchical feature extraction.
- Various DL architectures (MLP, CNN, RNN) which have been successfully applied in many fields (CV, ASR, NLP).
- Deep Learning has been shown to produce state-of-the-art results in many NLP tasks.