Using Lisp to Teach a Computer to Write

This post was originally written in Chinese and published on CSDN, https://blog.csdn.net/lzw_java/article/details/1159

Most of the code and ideas are based on “Ansi Common Lisp”P138~P141.

Problem: Given an English text, how can a computer generate random but readable text based on it? For example:

> The National Venture Capital Association estimates that wealth associated with a deal a big spending by regulations that will spend one another’s main reason these projects .

This is random text generated by a computer after learning some of Paul Graham’s articles. It extends into a sentence from the word “Venture”. Surprisingly, the text is often readable.

Algorithm: Record the words that appear after each word and the number of times they appear. For example, if “I leave”appears 5 times in the original text and “I want”appears 3 times, and “I”doesn’t appear before any other word, then when generating random text, when “I”is encountered, there is a 5/8 probability of choosing “leave”as the next word. If “leave”is chosen, then check which words have appeared after “leave” and repeat the process.

Now, let’s solve the problem using Lisp.

Lisp’s symbol type can record various strings and punctuation marks well, so we will use it for recording. We will use the built-in hashtable to create a list:

(defparameter *words* (make-hash-table :size 10000))

How do we create the list?

```
(let ((prev '|.|))
  (defun see (sym)
    (let ((pair (assoc sym (gethash prev *words*))))
      (if pair
      (incf (cdr pair))
      (push (cons sym 1) (gethash prev *words*))))
    (setf prev sym)))
```

The current word is the keyword, and the assoc-list is the value under that keyword. For example, under “I” we have ((|leave| . 5) (|want| . 3)). If the word doesn’t exist, then push (word . 1).

How do we randomly select a word?

```
(defun random-word (word ht)
  (let* ((choices (gethash word ht))
     (x (random (reduce #'+ choices :key #'cdr))))
    (dolist (pair choices)
      (decf x (cdr pair))
      (if (minusp x)
      (return (car pair))))))
```

Here, the reduce function is cleverly used.

Now, let's think about how to extend a given word into a sentence on both sides?

1) Reverse the text to get a reversed list, i.e., "I leave, I want"becomes "leave I, want I".
2) Reverse the hashtable to get another hashtable, where the later word is the key, and the possible preceding words and their counts form an assoc-list.
3) Try your luck, start extending the sentence from a punctuation mark until the given word appears.

I used the second method:

```
(defparameter *r-words* (make-hash-table :size 10000))

(defun push-words (w1 w2 n)
  (push (cons w2 n) (gethash w1 *r-words*)))

(defun get-reversed-words () ; a cat -> cat a
  (maphash #'(lambda (k lst)
           (dolist (pair lst)
         (push-words (car pair) k (cdr pair))))
       *words*))
```

Traverse the original hashtable, and then insert each pair of words into another hashtable with their order reversed. Here is the code for automatically generating bidirectional extended sentences:

```
(defparameter *words* (make-hash-table :size 10000))
(defconstant maxword 100)
(defparameter nwords 0)
(defconstant debug nil)
(let ((prev '|.|))
  (defun see (sym)
    (incf nwords)
    (let ((pair (assoc sym (gethash prev *words*))))
```

```
      (if pair
      (incf (cdr pair))
      (push (cons sym 1) (gethash prev *words*))))
    (setf prev sym)))

(defun check-punc (c) ; char to symbol
  (case c
    (#\. '|.|) (#\, '|,|)
    (#\; '|;|) (#\? '|?|)
    (#\: '|:|) (#\! '|!|)))

(defun read-text (pathname)
  (with-open-file (str pathname :direction :input)
    (let ((buf (make-string maxword))
      (pos 0))
      (do ((c (read-char str nil 'eof)
          (read-char str nil 'eof)))
      ((eql c 'eof))
    (if (or (alpha-char-p c)
        (eql c #\'))
        (progn
          (setf (char buf pos) c)
          (incf pos))
        (progn
          (unless (zerop pos)
        (see (intern (subseq buf 0 pos)))
        (setf pos 0))
          (let ((punc (check-punc c)))
        (if punc
            (see punc)))))))))

(defun print-ht (ht)
  (maphash #'(lambda (k v)
       (format t "~A ~A~%" k v))
        ht))

(defparameter *r-words* (make-hash-table :size 10000))

(defun push-words (w1 w2 n)
  (push (cons w2 n) (gethash w1 *r-words*)))
```

```
(defun get-reversed-words () ; a cat -> cat a
  (maphash #'(lambda (k lst)
           (dolist (pair lst)
         (push-words (car pair) k (cdr pair))))
       *words*))

(defun print-a-word (word ht)
  (maphash #'(lambda (k lst)
           (if (eql k word)
           (format t "~A ~A~%" k lst)))
       ht))

(if debug
    (print-a-word '|leave| *r-words*))

(defun punc-p (sym) ; symbol to char, nil when fails.
  (check-punc (char (symbol-name sym) 0)))

(defun random-word (word ht)
  (let* ((choices (gethash word ht))
     (x (random (reduce #'+ choices :key #'cdr))))
    (dolist (pair choices)
      (decf x (cdr pair))
      (if (minusp x)
      (return (car pair))))))

(defun gen-former (word str)
  (let ((last (random-word word *r-words*)))
    (if (not (punc-p last))
    (progn
      (gen-former last str)
      (format str "~A " last)))))

(defun gen-latter (word str)
  (let ((next (random-word word *words*)))
    (format str "~A " next)
    (if (not (punc-p next))
        (gen-latter next str))))
```

```
;(gen-latter '|leave| t)

(defun get-a-word (ht) ; get a random word
  (let ((x (random nwords)))
    (maphash #'(lambda (k v)
         (dolist (pair v)
           (decf x (cdr pair))
           (if (minusp x)
               (return-from get-a-word (car pair)))))
         ht)))
;(get-a-word *words*)
(defun gen-sentence (word str)
  (gen-former word str)
  (format str "~A " word)
  (gen-latter word str))

(defun test ()
  (setf nwords 0)
  (read-text "essay.txt")
  (get-reversed-words)
  (let ((word (get-a-word *words*)))
    (print word)
    (gen-sentence word t)))
(test)
```