DSP 2019 Spring HW1 Discrete HMM

B05901105 陳矩翰

- Environment:
  $ uname -a
  Linux t480 5.0.4-200.fc29.x86_64 #1 SMP Mon Mar 25 02:27:33 UTC 2019 x86_64 x86_64 x86_64 GNU/Linux
- compiler: g++ and gcc. Note that POSIX threading support is needed on the system.
  - gcc -v:
    Using built-in specs.
    COLLECT_GCC=gcc
    COLLECT_LTO_WRAPPER=/usr/libexec/gcc/x86_64-redhat-linux/8/lto-wrapper
    OFFLOAD_TARGET_NAMES=nvptx-none
    OFFLOAD_TARGET_DEFAULT=1
    Target: x86_64-redhat-linux
    Configured with: ../configure --enable-bootstrap --enable-languages=c,c++,fortran,objc,obj-c++,ada,go,lto --prefix=/usr --mandir=/usr/share/man --infodir=/usr/share/info --with-bugurl=http://bugzilla.redhat.com/bugzilla --enable-shared --enable-threads=posix --enable-checking=release --enable-multilib --with-system-zlib --enable-__cxa_atexit --disable-libunwind-exceptions --enable-gnu-unique-object --enable-linker-build-id --with-gcc-major-version-only --with-linker-hash-style=gnu --enable-plugin --enable-initfini-array --with-isl --enable-libmpx --enable-offload-targets=nvptx-none --without-cuda-driver --enable-gnu-indirect-function --enable-cet --with-tune=generic --with-arch_32=i686 --build=x86_64-redhat-linux
    Thread model: posix
    gcc version 8.3.1 20190223 (Red Hat 8.3.1-2) (GCC)
- how to train?
  $ chmod u+x run_train.sh; make clean; make; bash run_train.sh
- how to test?
  Please refer to the homework PDF.
- Some perks of this implementation:
  - 比較直覺性的實作可能會在每個 gamma 跟 epsilon 的 entry 都重新用 summation 做一次 P(O|lambda)，個人在這個作業的 P(O|lambda)都是使用該 observation sequence O 產生的 alpha 的最後一個 column 之 summation 作為其值，因此對於每個 input sequence O 來說可以省下 Θ(m*n)這麼多的計算量，其中 m 是 HMM 的 hidden state 數量、n 是 input sequence O 的長度，並且在實際背後代表的意義上應該也是較為精確的。
  - 在 training 以及 testing 的部份都有使用 POSIX pthreads 以進行加速:
    traing 時，針對一個 input sequence O，很明顯地，alpha 跟 beta 可以同時計算而不會有 data-racing 問題，同理 gamma 跟 epsilon 的累計、Baum Welch 最後計算新的(π', A', B')的部份也都可以平行化運算，因此在 training 的程式中這三個部份都是平行處理的，alpha/beta 開兩條 thread、gamma/epsilon 開兩條、B-W 開三條;
    testing 時，因為我們是針對每一個 HMM 進行運算、取值，對於 testing data 檔案只是唯讀操作，因此在這個程式中有多少個 HMM 就一次開了多少個 thread。
  - 有對每個 O 的長度進行 normalization，因此雖然這個作業測資的 O 長度固定都在 50，本程式可以處理檔案中有不同長度的測資的情況，並且儘管長度可能不同，其對結果的貢獻都是大致相同的。
  - 另，本程式也能應對不同 hidden state 總數、不同 possible observation count from a hidden state 的 initial HMM，例如可以丟有 8 個 hidden state 的 HMM、以及共有 ABCDEFGHI 的測資給這支程式訓練，在測試的時候也可以有不同 hidden state 數量的 HMM 一起進行處理。
- Some defects of this implementation:
  - 沒有對 HMM 的 observation probability may degenerate to 0 的預防處理，因此當 train iteration 過高可能會讓那個 model 直接壞掉。
  - run_train.sh 的結果其正確率並不高，僅 0.5932。
  - 測資只能是 char 型別，並且其必須大於等於 A，否則會產生記憶體存取越位。