Master IARFID
Reconocimiento de Escritura (RES)
Practical session
State-of-the-art HTR systems:Decoding

PRHLT-Group

Departamento de Sistemas Informáticos y Computación
Universitat Politècnica de València

May 7, 2020

Description

- Deep Neural Networks are the state-of-the-art technology for handwritten text recognition.
- We will made a complete HTR system based on neural networks.
- First, we have trained an optical model based on Deep Neural Networks with Connectionist Temporal Classification (CTC).
- Secondly, we will:
  - A Test the optical model without language model.
  - B Build a weighted finite state transductor to constrict the search space: TLG.fst
  - C Test the TLG model
  - D Get Word Graphs

A. Testing the optical model

1. Decoding (1m aprox.)

```
mkdir results

source RDNN-HTR-PY/bin/activate

pylaia-htr-decode-ctc --train_path models/Optical \
  --model Rodrigo.net --batch_size 3 --use_letters \
  --space " " data/lists/symbols_train.lst \
  data/feat/ data/lists/test.lst | \
sort -k1 > results/test_chars.hyp
```

2. From output chars to words

```
sed -e "s/\[//g" -e "s/\]//g" -e "s/'//g" -e "s/\.jpg//" \
results/test_chars.hyp |\
awk -F", " '{for (i=1; i<=NF; i++)
  printf("%s",$i); printf("\n")}' > results/test_words.hyp
```

A. Testing the optical model

3. Getting the test references

```
awk -v TEST_LIST=data/lists/test.lst \
  'BEGIN{
     while ((getline < TEST_LIST) > 0){TL[$1]=1}
  }{
     if ($1 in TL) {
        printf("%s ",$1);
        for (i=2; i<=NF; i++)
            printf("%s",$i);
        printf("\n")
     }
  }' data/text/transcriptions_char.txt | \
sed 's/<space>/ /g' > data/text/test.ref
```

A. Testing the optical model

4. Matching hypothesis and references

```
awk -v TEST_LIST=data/text/test.ref '
BEGIN{
 while ((getline < TEST_LIST) > 0)
    TL[$1] = $0

} {
  if ($1 in TL) {
    N=split(TL[$1],DIC)
    for (i=2; i<NF; i++) printf("%s ", $i)
    printf("%s#",$NF);
    for (i=2; i<N; i++) printf("%s ",DIC[i])
    printf("%s\n",DIC[N]);
  }
}' results/test_words.hyp > results/test_words_hyp2ref
```

A. Testing the optical model

5. Getting the tasas evaluating tool

```
wget --no-check-certificate \
http://www.prhlt.upv.es/~mpastorg/RES/tasas.tgz

tar xvzf tasas.tgz
cd tasas
gcc -O3 -o tasas tasas.c
cd ..
```

6. Getting scores

```
WER=`tasas/tasas -f "#" -s " " results/test_words_hyp2ref`
CER=`tasas/tasas -f "#" results/test_words_hyp2ref`

echo "CER="$CER" WER="$WER
```

output –> CER=1.968 WER=7.687

B. Preparing the search graph

- A search graph is built with three WFSTs (T, L and G) compiled independently and combined as follows:

$$S = T \circ \min(\det(L \circ G))$$

  T, L and G are the token, lexicon and grammar WFSTs respectively, whereas $\circ$, det and min denote composition, determination and minimization, respectively. The determination and minimization operations are needed to compress the search space, yielding a faster decoding.

- We shall need some scripts

```
wget --no-check-certificate \
http://www.prhlt.upv.es/~mpastorg/RES/scripts.tgz

tar xvzf scripts.tgz
```

B. Preparing the search graph: T.fst

1. Adding special symbols to the trained graphical symbols

```
mkdir models/WFST
awk 'BEGIN{
     cont=0;
     print "<eps>",cont++;
     print "<ctc>",cont++;
     print "<blk>",cont++; getline; getline;
}{ print $1,cont++; }
END{
     print "#0",cont++;
     print "#1",cont++;
     print "#2",cont++;
     print "#3",cont++;
}' data/lists/symbols_train.lst > models/WFST/tokensMap.txt
```

2. Get the tokens weighted finite state transductor T.fst

```
scripts/ctc_token_fst.py models/WFST/tokensMap.txt | \
 fstcompile --isymbols=models/WFST/tokensMap.txt \
 --osymbols=models/WFST/tokensMap.txt | \
 fstarcsort --sort_type=olabel > models/WFST/T.fst
```

B. Preparing the search graph: train the language model

3. Preparing the transcriptions

```
sed -e "s/\w*\^{\w*}//g" -e "s/\w*\[//g" -e "s/\]//g" \
    data/text/transcriptions.txt | \
awk -v MAP_UNITS=data/lists/unitsMap.lst \
    'BEGIN{ while ((getline < MAP_UNITS) > 0)
         MAP[$1] = $2
    }{
      printf ("%s ",$1)
      for(w=2; w<=NF;w++) {
        N=split(tolower($w),CHAR,"")
        for(i=1; i<=N; i++)
        if (CHAR[i] in MAP)
            if (CHAR[i] ~ /[,;.:?()]/)
                printf(" %s ", MAP[CHAR[i]])
            else
                printf("%s", MAP[CHAR[i]])
            printf (" ")
        }
        printf("\n")
    }' > data/text/transcr_words.txt
```

B. Preparing the search graph: train the language model

4. Getting the corpus training

```
awk -v TR_LIST=data/lists/train.lst \
    -v VAL_LIST=data/lists/val.lst ' BEGIN{
    while ((getline < TR_LIST) > 0) DICT[$1] = 1
    while ((getline < VAL_LIST) > 0) DICT[$1] = 1
  }{
    if ($1 in DICT) {
      $1="";
      N=split($0,CHAR,"")
      for(i=1; i<=N; i++){
        if (CHAR[i] ~ /[(),.:;?]/)
           printf(" %s ",CHAR[i])
        else
           printf("%s",CHAR[i])
      }
      printf ("\n")
    }
}' data/text/transcr_words.txt > data/text/train_words.txt
```

B. Preparing the search graph: L.fst

5. Getting the mapping from words to numbers

```
awk '{for (w=1; w<=NF; w++){print $w}}' \
     data/text/train_words.txt | sort -u | \
awk 'BEGIN{print "<eps> 0"; print "<s> "1; print "</s> 2"}
    {print $0, NR+2} END{ print "#0",NR+3; }' \
> models/WFST/wordsMap.txt
```

6. Build the lexicon

```
awk 'BEGIN{
    print "<s> <space> #1"; print "</s> <space> #2";
}{
    N=split($1,CHARS,"");
    if ($1 !~ /<s>/ && $1 !~ /<\/s>/ &&
        $1 !~ /<eps>/ && $1 !~ /#0/){
          printf("%s ", $1);
          for(i=1; i<=N; i++)
            printf("%s ",CHARS[i]);
          printf ("<space>\n")
    }
}' models/WFST/wordsMap.txt > models/WFST/lexicon.txt
```

B. Preparing the search graph: L.fst

7. Build the lexicon weighted finite state transductor L.fst

```
export PATH=$PATH:/home/usures/kaldi/src/lmbin:\
  /home/usures/eesen/src/fstbin/

token_disamb=`grep \#0 models/WFST/tokensMap.txt| \
    awk '{print $2}'`
word_disamb=`grep \#0 models/WFST/wordsMap.txt| \
    awk '{print $2}'`

scripts/make_lexicon_fst.pl \
    models/WFST/lexicon.txt 0.5 "<space>" '#'3| \
fstcompile --isymbols=models/WFST/tokensMap.txt \
          --osymbols=models/WFST/wordsMap.txt | \
fstaddselfloops "echo $token_disamb |" "echo $word_disamb |"| \
fstarcsort --sort_type=olabel > models/WFST/L.fst
```

B. Preparing the search graph: G.fst

8. Getting the 3-grams language model

```
ngram-count -order 3 -kndiscount -interpolate \
-text data/text/train_words.txt -lm models/3gram-words.lm
```

9. Build the weighted finite state transductor G.fst

```
arpa2fst models/3gram-words.lm | fstprint | \
sed -e 's/<eps>/\#0/' -e 's/<s>/<eps>/g' \
-e 's/<s>/<eps>/g' -e 's/<\/s>/<eps>/g'|\
fstcompile --isymbols=models/WFST/wordsMap.txt \
    --osymbols=models/WFST/wordsMap.txt |\
fstrmepsilon | fstarcsort --sort_type=ilabel > \
    models/WFST/G.fst
```

B. Preparing the search graph: LG and TLG compositions

10. Getting LG as $\min(\det(L \circ G))$

```
fsttablecompose models/WFST/L.fst models/WFST/G.fst |\
fstdeterminizestar --use-log=true |\
fstminimizeencoded |\
fstarcsort --sort_type=ilabel > models/WFST/LG.fst
```

11. Getting TLG as $T \circ \min(\det(L \circ G))$

```
fsttablecompose models/WFST/T.fst models/WFST/LG.fst > \
   models/WFST/TLG.fst
```

C. Testing the TLG model

1. Getting the confidences matrix

```
pylaia-htr-netout --show_progress_bar \
    --logging_level info --logging_also_to_stderr info \
    --logging_file CMs-crnn.log \
    --train_path ./models/Optical \
    --model_filename Rodrigo.net \
    --batch_size 40 \
    --output_transform log_softmax \
    --output_matrix ConfMats.ark \
    data/feat/ data/lists/test.lst
```

2. Testing language and lexicon restrictions (3m aprox.)

```
export PATH=$PATH:/home/usures/eesen/src/decoderbin

decode-faster --print-args=false \
    --beam=30.0 --max-active=5000 \
    --acoustic-scale=1.0 --allow-partial=true \
    --word_symbol-table=models/WFST/wordsMap.txt \
    models/WFST/TLG.fst ark:ConfMats.ark \
    ark,t:results/test_TLG_numbers.hyp
```

C. Testing the TLG model

3. Preparing the test references

```
awk -v TEST_LIST=data/lists/test.lst '
BEGIN{
   while ((getline < TEST_LIST) > 0) DICT[$1] = 1
}{
  if ($1 in DICT) {
    printf("%s ",$1)
    $1="";
    N=split($0,CHAR,"")
    for(i=1; i<=N; i++){
      if (CHAR[i] ~ /[(),.:;?]/)
         printf(" %s ",CHAR[i])
      else
         printf("%s",CHAR[i])
    }
    printf ("\n")
  }
}' data/text/transcr_words.txt | \
      sed "s/\s\s*/ /g" > data/text/test_words.ref
```

C. Testing the TLG model

4. Getting WER and CER

```
scripts/int2sym.pl -f 2-200 models/WFST/wordsMap.txt \
    results/test_TLG_numbers.hyp | \
awk -v TEST_LIST=data/text/test_words.ref 'BEGIN{
  while ((getline < TEST_LIST) > 0) DICT[$1] = $0

} {
  if ($1 in DICT) {
    N=split(DICT[$1],DIC)
    for (i=2; i<NF; i++) printf("%s ", $i)
    printf("%s#",$NF);
    for (i=2; i<N; i++) printf("%s ",DIC[i])
    printf("%s\n",DIC[N]);
  }
}' > results/test_TLG_hyp2ref

echo -e "WER= "`tasas/tasas -f "#" -s " " results/test_TLG_hyp2ref`
echo -e "CER= "`tasas/tasas -f "#" results/test_TLG_hyp2ref`
```

output -> CER = 6.087 WER = 17.962

D. Getting Word Graphs

1. Get the word graphs (25m apox.)

```
mkdir -p results/lattices/words

latgen-faster --beam=25.0 --max-active=5000 \
       --acoustic-scale=1.0 models/WFST/TLG.fst ark:ConfMats.ark \
       "ark,t:|gzip -c > results/lattices/test-word-lat.gz" \
        ark,t:results/lattices/test-3gram-words-1bestLat.hyp
```

2. Label the edges with words.

```
zcat results/lattices/test-word-lat.gz| \
perl scripts/int2sym.pl -f 3 models/WFST/wordsMap.txt \
    > results/lattices/test-word-lat
```

3. From Kali to HT format.

```
perl scripts/convert_slf.pl results/lattices/test-word-lat \
    results/lattices/words
```