Automatic Speech Recognition using OpenFST

Abstract

Automatic Speech Recognition (ASR) is a process transform the speech signal into text transcription, without necessarily understanding the meaning of what was spoken (Renals1 & Hain, 2010). It has many real life applications, e.g. telephone-based services, Siri, which enables human-machine communication (Yu & Deng, 2015). Weighted finite state transducers (WFST) plays an important role for WFST-based-ASR system in the terms of recognising patterns of the words (Mohri et al., 2008). In this paper, we will show how the performance of a ASR system based on OpenFST (Allauzen et al., 2007) is affected using different hyper-parameters in WFST. Also, we will show how the performance of ASR system is affected by applying Pruning method and using a different language model (Bigram).

1. Introduction

Automatic Speech Recognition is a process transform audio representation to the textual representation. This technology is recently developed rapidly and has many real applications in telephone based services, such as dialing, airline reservation, bank transaction and price quotation (Silva et al., 2012). ASR improves both Human-Human communication and Human-Machine communication. For example, it could improve Human-Human Communication in a way by integrating ASR with Machine Translation and Text to Speech technologies, enables two people who speak different langugae to communicate. Also, ASR could be integrated with NLP, enables machine understanding human's speech (Yu & Deng, 2015).

A finite-state transducer (FST) is a finite automaton whose state transitions are labeled with both input and output symbols (Mohri, 1997). A path through the transducer encodes the a mapping from an input to output. A weighted finite-state transducer (WFST) puts weights on transition addition to input and output. For a ASR system based on WFST, WFST provides essential components for the system, including hidden Markov Models, n-gram statistical- models and lexicons (Mohri et al., 2008)... A finite-state transducer (FST) is a finite automaton whose state transitions are labeled with both input and output symbols (Mohri, 1997). A path through the transducer encodes the a mapping from an input to output. A weighted finite-state transducer (WFST) puts weights on transition addition to input and output, an example of WFST shown in Figure 1. The figure 1

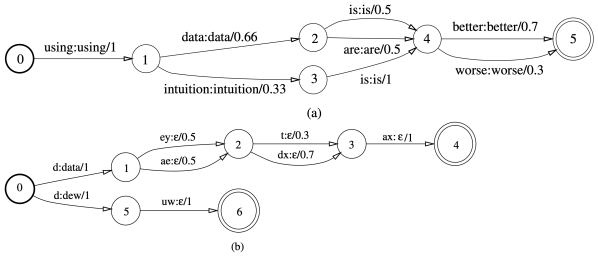

Figure 1: Weighted finite-state transducer examples.The input label i, the output label o, and weight w of a transition are marked on the corresponding directed arc by i : o/w.

shows WFST with weight in likelihood (ranged from 0 to 1), but the weight we will use in this paper is negative log likelihood (ranged from 0 to INF), to avoid underflow in probabilities when calculating the total weight of a path. For a ASR system based on WFST, WFST provides essential components for the system, including hidden Markov Models (HMM), context-dependency model, pronunciation lexicon and word-level grammar (Mohri et al., 2008). The input and output sequence of each transducer is shown in table 1:

	TRANSDUCER	INPUT SEQUENCE	OUTPUT SEQUENCE
G	WORD-LEVEL GRAMMAR	WORDS	WORDS
L	PRONOUNCIATION LEXICON	PHONES	WORDS
C	CONTEXT-DEPENDENCY	CD PHONES	PHONES
H	HMM	HMM STATES	CD PHONES

Table 1: a table showing different component represent as WFSTs, with their input sequence and output sequence. By combing transducers HCLG, it results in a transducer that maps from HMM states to word sequence

The transducers could be composited to produce a WFST that maps from HMM states to word sequence (Mohri et al., 2008). We could the Viterbi algorithm to find the most probable HMM state sequence given observations, and use the WFST to map the HMM state sequence to word sequence (Gales et al., 2008).

In this work, we will firstly build up a naive WFST-based ASR system using OpenFST (Allauzen et al., 2007). Then we will see the effect of each hyper-parameters on the ASR system, including self-loop probability, final probability, transition probability using unigram word probability and adding silence states to the WFST. Also, we will see how the performance of the system changes after pruning. Lastly,

we will use a bigram language model instead of unigram language model, and test the performance of the system using the bigram language model.

2. Dataset and Task

The dataset we use in our experiment is a small collection of the recordings from ASR students on this year's course. The recordings comprise short utterance with words randomly selected from "Piper picked a peck of pickled peppers. Where's the peck of pickled peppers Peter Piper picked?" (/group/teaching/asr/labs/recordings). In total, there are 2177 words in reference transcriptions.

As I have mentioned in section 1, there are 4 tasks in this paper. The first task is building a initial system of ASR. The system is not weighted, meaning each transition between state in WFST is equally probable. Then we will adjust the hyper-paramets to see the effects of them on the system's performance. The hyper-parameters we will adjust are: self-loop probability, final probability, transition probability and adding silence state to the system. Later, we will prune the system using beam Search, ignoring the low probability hypothesis. Lastly, we will see the effect of using Bigram language model on the system instead of unigram language at the beginning.

We will evaluate the performance of the ASR system through few aspects. The first metric is the Word Error Rate (WER) (Jurafsky & Martin, 2009):

$$WER = \frac{S + D + I}{N} \tag{1}$$

where S,D and I stands for substitutions,deletions and insertions respectively, and N is the number of words in reference transcription. In this work, N = 2177.

We will also evaluate the performance of the ASR by measuring the speed of the Viterbi decoder in the terms of time taken for decoding and backtracing.

Lastly, we will measure the memory required for the decoder in the terms of number of arcs and states in the WFST.

3. Task 1 - Initial systems

We will firstly build up an initial system as a baseline for our ASR system. As I have mentioned in section 1, we need to composite HCLG transducers together to produce a WFST that could map from HMM state sequence input to words sequence output. Traditionally, for H transducer, we model each phone using 3 HMM states with a left-to-right topology and with self loops (Renals1 & Hain, 2010). The 3 states enforce the minimum duration of each phone. As we are using unigram language model, we do not have worry about C transducer as it is just a identical mapping which could be skipped. For L transducers, we produce a WFST that could recognise every word in lexicon, which maps phones to words. As we are no grammar for our practice, G transducer is not required. Hence, we indeed only have two transducers (H and L) that maps from HMM sequence input to words sequence output.

Below shows how our H∘C WFST looks like:

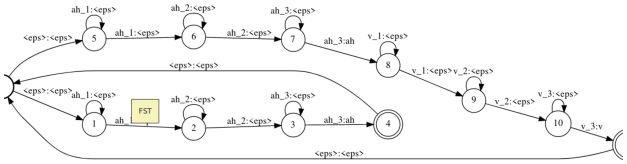

Figure 2: A figure shows how our H∘C WFST looks like, for phones of words "cat" and "a". The weight is not added.

After few naive tests of different threshold values, we will test the performance of the system start decreasing from the threshold of $0.5 * w_t^{best}$, without dropping drastically the WER.

Also, as the initial system is just a baseline for our tasks later, we do not put weights on the transducers. Hence, each transition is equally probable. The performance of baseline model is shown below:

S	D	I	WER
694	329	670	77.77%

Table 2: a table showing the accuracy of the baseline model in term of WER in 4 s.f. , where S stands for substituion, D stands for deletion, I stands for Insertion

D.T mean	B.T Mean	FC mean
8.1	0.0058	114810

Table 3: a table, from left to right respectively, showing the mean time for decoding time taken, the mean time for backtracing taken, The total number of forward computation for decoding and mean number of forward computation, for the baseline model

The baseline model has WER around 77.77%, which is quite high as we are dealing small vocabulary. substitution error is quite high, which means that the output word sequence are not correct. The insertion is also quite high, meaning there should be more words in the output which are not present. This is normal, as we have not set the weight for the transducers yet. The output words are not mainly relied on the previously defined observation probability. Now, we will adjust the hyper paremeters of the WFST to improve its performance.

4. Task 2: System turning

The self-looping and transition in the HMM model aligns phones with the observed audio frames. Different self-looping and transition probability will cause different alignment which largely affect the performance of the ASR sys-

WFST	No. States	No. Arcs	Sum
L ∘ G	28	37	65
H ∘ C	116	230	346

Table 4: a table showing the number of state, number of arcs and sum of the states and arcs for Transducer H ∘ C that maps HMM states to phones and Transducer L ∘ G that maps from phone to words, in baseline model

tem. Final probability puts weights on state being as final states, as different words would have likelihood appears as the last words.

Firstly, we will set up the weight in the H∘C transducers. We will use the unigram distribution of words in reference transcription to set up the transition probability. The distribution of words shown below:

Distribution: {where's: 146, peter: 267, piper: 219, peck: 219, peppers: 252, pickled: 246, picked: 258, a: 184, the: 165, of: 221}

Similily, we will use the distribution of final words to set up the final probability. The distribution of final words shown below:

DistributionFinal:{where's: 3, piper: 26, a: 4, peck: 17, peppers: 45, picked: 46, pickled: 15, peter: 16, the: 3, of: 5 }

We also set up the self loop probability and the transition probability to the next HMM state with probability negative log 0.5. All the probability is summed to one for leaving any state in the H∘C transducers.

We will firstly increase the probability of the self loop by 0.1 each step until the WER starts to decrease. If WER decreased at the first place, then we will decrease the probability by 0.1 each step from 0.5, until WER decreased again. Then, we will adjust the final probability. By inspection of final words distribution, some of words are unluckily to be in the final states. We will decrease the probability of unlikely words being in final state by multiple the frequent words distribution by a magnitude.

Also, we will increase the probability of the most frequent words being in final states by multiplying the same magnitude. Test the different magnitude to see how the allocating more probability to the frequent words affect the performance of the system.

We will not adjust the transition probability after the initial set up, as I think using the distribution is a quite reasonable approach.

Lastly, we will add additional silence states to the WFST. The silence states are modelled similarly to the way how the phones are modelled. This is because naturally we have pauses between words when we speak, hence we think the silence states are essential for improving the performance of the system.

5. Task 3: Pruning

Pruning is the technique of cutting off search the state we have established that this partial solution cannot be extended into the solution that we want. As exact search is not possible for large vocabulary tasks, we use pruning as a solution for computation issues (Mohri et al., 2008). As our task is not large vocabulary tasks, we think pruning should have small effect on our system. We will test how the pruning method actually affect our system using different threshold.

For pruning, we will remove the states with weight that are outside the threshold of minimum weight for next time step. we will mainly follow the principle of algorithm present in figure 3.

Algorithm 21 prune(S, A, t)

$w_t^{best} \leftarrow \{\bigoplus_{s \in S} \alpha(t, s)\} \oplus \left\{\bigoplus_{\langle e, j\rangle \in A} \alpha(t, e, j)\right\}$
$w_t^{th} \leftarrow \gamma \otimes w_t^{best}$
for each $s \in S$ **do**
  if $\alpha(t, s) \oplus w_t^{th} = w_t^{th}$ **then**
    Erase(S, s)
  end if
end for
for each $\langle e, j\rangle \in A$ **do**
  if $\alpha(t, e, j) \oplus w_t^{th} = w_t^{th}$ **then**
    Erase($A, \langle e, j\rangle$)
  end if
end for
return $\langle S, A\rangle$

Figure 3: Psudocode for pruning algorithm showing how to prune the states for searching. w_t^{best} is the weight of the best path in time t, and γ is the threshold for prunning

After few naive tests of different threshold values, we will test the performance of the system start decreasing from the threshold of $0.5 * w_t^{best}$, without dropping drastically the WER.

6. Task 4: Improving the grammar

Although our data is a collection of recordings of words randomly selected from "Peter Piper picked a peck of pickled peppers. Where's the peck of pickled peppers Peter Piper picked?", the distribution of final words shows that people do not intent to put words "where's, a ,of the" at the end of the sentence. We want to to test if using bigram language model could improve the performance of the system. Aslo, the effects of using bigram language on the peroformance could show us whether there is a pattern that how people tend to select the words.

For our implementation, as we have conditional probability for all words, no backoff methods would be used. We will just take the conditional distribution ground truth transcripts as the conditional transition probability between words. Figure 4 is a sight of how bigram model is implemented.

This WFST has HMM states sequence input and phone sequence output as before (H ∘ C). But it now has no back transition from words to start state. It has transition between final state of each words and first HMM state of each words, where transition has weight of conditional probability. For

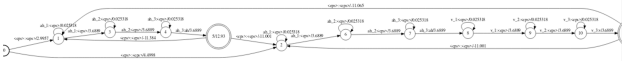

Figure 4: A figure shows how our H∘C WFST of bigram language model works for words "a" and "of". The silence states are not included.

the actual implementation, each state, including start state, is connected to silence state (from the state to silence state and back from silence state).

7. Experiments and discussion

In this section, we will show the results of different hyper parameters on the performance of the system. Also we will show the results of pruning and using bigram language model.

7.1. Weights Setup

The result after setting up the initial weight for WFST using words distribution in reference transcripts shown in below:

Model	S	D	I	WER
baseline	694	329	670	77.77%
After setup	678	357	603	75.24 %

Table 5: a table showing the WER of the baseline model and the model after setting up the initial weight in term of WER in 4 s.f. , where S stands for substituion, D stands for deletion, I stands for Insertion

Model	D.T mean	B.T Mean	FC mean
Baseline	8.1	0.0058	114810
After Setup	7.6	0.0056	114593

Table 6: a table, from left to right respectively, showing the mean time for decoding time taken, the mean time for back-tracing taken, The total number of forward computation for decoding and mean number of forward computation, for the baseline model and the model with setup the weight

As shown in table 8, the accuracy after setting up the initial weight has improved the WER by around 2%, which is really reasonable. There is small drop in both insertion errors and substitution errors, but with increase in deletion errors. It seems that the system tends to have short word output sequence. Interestingly, there is drop in mean time for computing the decoding. This could due to the different computing resources we had for the experiments, as there is no much drop in mean forward computation. The memory required is not shown as it is not necessary. There is no changes in the WFST.

Note that for the experiment results afterwards, the time spent and the memory required would not present if it is not necessary if they are not affected by the hyper parameters for the models.

7.2. Self loop probability

Below shows the result for different self loop probability on the model after the initial weight setup:

Model	S	D	I	WER
sl = 0.5	678	357	603	75.24 %
sl = 0.6	644	397	560	73.54%
sl = 0.7	635	413	510	71.57%
sl = 0.8	618	435	466	69.77%
sl = 0.9	611	457	416	68.17%
sl = 0.95	598	492	380	67.52%
sl = 0.975	595	520	350	67.29%

Table 7: a table showing the accuracy of the baseline model and the model after setting up the initial weight in term of WER in 4 s.f. , where S stands for substituion, D stands for deletion, I stands for Insertion

Note that the model with self loop probability = 0.5 is the model after the initial weight set up. As we can see from table 7, WER decrease as self loop (sl) probabilit increases, and the rate of decreasing decrease as the self loop probability closes to 1. As the sl probability increase, the substitution error and insertion error dropped significantly, but it increased the deletion error. This means that output words sequence is general shorter than before, with better accuracy. This is reasonable as self loop probability determine the duration of a state, and hence the duration of a word. As the duration of word increase, the total words generated would be decreased as number of time frame is fixed. We stopped at the sl = 0.975 as we think the WER decreased only small amount from sl =0.95, and it is about a point where it would increase WER for higher value sl. Hence, we will use sl = 0.975 for the following experiment.

7.3. Final probability

The result of adjusting the final probability shown below:

Model	S	D	I	WER
M=1	595	520	350	67.29%
M=100	593	520	342	66.83%
M=10000	592	521	330	66.28%

Table 8: a table showing the accuracy of the models with adding different weight on the more frequent words in term of WER in 4 s.f. , where S stands for substituion, D stands for deletion, I stands for Insertion

In theory, by adjusting final probability, it puts penalty on a state leaving a final state, which enforces states to stay in the states that are more likely to be final states. However,it seems that the decrease of WER is not significant

by adjusting final probability. We think this is due to the fact the accumulated weight through the states is still more weighed. We will use M =10000 for later experiment as it has the lowest WER.

7.4. Silence states

The results for model with silence states shown below:

Model	S	D	I	WER
without SIL	592	521	330	66.28%
with SIL	547	590	140	58.66%

Table 9: a table showing the accuracy of the models with or without silence state in term of WER in 4 s.f. , where S stands for substituion, D stands for deletion, I stands for Insertion

WFST	No. States	No. Arcs	Sum
H ∘ C old	116	230	346
H ∘ C with SIL	120	238	358

Table 10: a table showing the number of state, number of arcs and sum of the states and arcs for Transducer H ∘ C that maps HMM states to phones and Transducer L ∘ G that maps from phone to words, in the old model and the model with silence states

As shown in table 9, there is significant drop (about 8%) in WER with silence states. This is due to the significant drop in insertion errors and small drop in substitution errors, though deletion errors increased quite a lot. The significant drop in insertion errors meaning that the words sequence output is now much shorter than before, because now the words do not have to be output if the time frame is more likely to be in a silence state. As shown in table 10, adding silence states only increased the number of states by 4 (3 HMM state and 1 final state) and arc numbers by 8, while it improves the WER by 8 %. Hence, silence states are very useful for the system.

7.5. Pruning

Below shows result for prunning:

As shown in table 11, the accuracy generally increased as the threshold increases. This is reasonable as the true solution might be pruned and cause the WER to increase. There is one small bump from threshold = 0.4 w_t^{best} to threshold = 0.3 w_t^{best}. This is interesting and it might due to that the best path before pruning may not be the ground truth path, and the best pruned path is more accurate than the path before. Also, as shown in table 12, the mean time for decoding and mean number of froward computation decreases as the threshold decreases. This is because more pruned states are pruned with lower threshold and saves the computation cost.

Model	S	D	I	WER
Old	547	590	140	58.66%
TH=0.6 w_t^{best}	534	646	133	60.31%
TH=0.5 w_t^{best}	541	646	143	61.09%
TH=0.4 w_t^{best}	539	657	140	61.37%
TH=0.3 w_t^{best}	537	654	143	61.28%
TH=0.2 w_t^{best}	557	636	168	62.52%
TH=0.1 w_t^{best}	580	622	218	65.22%

Table 11: a table showing the accuracy of the models with different pruning threshold in term of WER in 4 s.f. , where S stands for substituion, D stands for deletion, I stands for Insertion

Model	D.T mean	B.T Mean	FC mean
Old	3.7	0.0022	118831
th=0.6 w_t^{best}	3.4	0.0022	110264
th=0.5 w_t^{best}	3.4	0.0022	109019
th=0.4 w_t^{best}	3.3	0.0022	107163
th=0.3 w_t^{best}	3.3	0.0022	106082
th=0.2 w_t^{best}	3.3	0.0022	104038
th=0.1 w_t^{best}	3.1	0.0024	99004

Table 12: a table, from left to right respectively, showing the mean time for decoding time taken, the mean time for backtracing taken, The total number of forward computation for decoding and mean number of forward computation, for the pruned model with differnt thresholds. MIN is the weight of the best path

Note that the mean time of decoding and backtracing for old model (the last model with silence states) decreased due to the fact that the experiments ran with different computation source. However, the backtracing time are about same through different threshold except TH = 0.1 w_t^{best}. We think this increase could be due to the noisy in the computation environment, as the pruning should have negligible effects on the backtracing for my implementation.

As the result, We think the pruning does not have much effect on the speed of the system without affecting the performance. There might be few reasons for it. Firstly, our task is a small vocabulary task, and there is no dependent relation between words (no transition between words), hence pruning states do not prune much arcs, which do not save much computations. Also, the time required to compute the best path and pruned out each state could relatively cost quite much. Overall, we think the pruning is unnecessary for our task, and we will not use pruning for the last task.

7.6. Bigram language model

Below shwos a result of bigram language model:

As the result shows in tables 2 14 15, the overall performance of the bigram model is worse than the unigram model. The bigram model has higher WER, slower speed and more memory requirements. This is the evident that

Model	S	D	I	WER
Unigram	547	590	140	58.66%
Bigram	605	489	221	60.40%

Table 13: a table showing the accuracy of the models with or without silence state in term of WER in 4 s.f. , where S stands for substituion, D stands for deletion, I stands for Insertion

Model	D.T Mean	B.T Mean	FC MEAN
Unigram	3.7	0.0022	118831
Bigram	5.1	0.0022	159853

Table 14: a table showing the accuracy of the models with different pruning threshold in term of WER in 4 s.f. , where S stands for substituion, D stands for deletion, I stands for Insertion

the words are randomly selected, meaning each words are more likely to having independent relation between each other rather than dependent relation. It is better to model our task using unigram model.

8. Conclusions & Further work

Overall, we have learned many things from this work. The most important thing we have learned is that how each of hyper paraemeters, pruning method and bigram language affect our system. The increase of the self loop probability increase the duration of words and hence decrease the insertion errors. Similarly, the silence state also decrease the insertion errors. Pruning and bigram language model are not working really well on our task, but it should become very useful when we modelling the real word tasks with large vocabulary.

For further work, we would like to investigate how to improving the efficiency of the decoder using tree-structured lexicon, optionally with language model look-ahead. Also, we may want to try different pruing strategy such as multi-pass search.

WFST	No. States	No. Arcs	Sum
H ∘ C Unigram	120	238	358
H ∘ C Bigram	160	408	568

Table 15: a table showing the number of state, number of arcs and sum of the states and arcs for Transducer H ∘ C that maps HMM states to phones and Transducer L ∘ G that maps from phone to words, in the old model and the model with silence states

References

Allauzen, Cyril, Riley, Michael, Schalkwyk, Johan, Skut, Wojciech, and Mohri, Mehryar. Openfst: A general and efficient weighted finite-state transducer library. In Holub, Jan and Žďárek, Jan (eds.), *Implementation and Application of Automata*, pp. 11–23, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. ISBN 978-3-540-76336-9.

Gales, Mark, Young, Steve, et al. The application of hidden markov models in speech recognition. *Foundations and Trends® in Signal Processing*, 1(3):195–304, 2008.

Jurafsky, Daniel and Martin, James H. *Speech and Language Processing (2Nd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2009. ISBN 0131873210.

Mohri, Mehryar. Finite-state transducers in language and speech processing. *Computational linguistics*, 23(2): 269–311, 1997.

Mohri, Mehryar, Pereira, Fernando, and Riley, Michael. Speech recognition with weighted finite-state transducers. In *Springer Handbook of Speech Processing*, pp. 559–584. Springer, 2008.

Renals1, Steve and Hain, Thomas. Speech recognition. 2010.

Silva, Diego F., de Souza, Vinícius M. A., Batista, Gustavo E. A. P. A., and Giusti, Rafael. Spoken digit recognition in portuguese using line spectral frequencies. *Lecture Notes in Computer Science*, pp. 241–250, 2012.

Yu, Dong and Deng, Li. *Introduction*, pp. 1–9. Springer London, London, 2015. ISBN 978-1-4471-5779-3. doi: 10.1007/978-1-4471-5779-3_1. URL https://doi.org/10.1007/978-1-4471-5779-3_1.