ASR CW2

s2264670, s2277042

Edinburgh University

1. ABSTRACT

In this report, we will describe our results in CW2. First, we created a simple baseline that is similar to the model shown in Lab 3, 4. Then, in each section, we improved our model by adding some changes regarding both the structure of the WFST and the Viterbi algorithm. We calculated Word Error Rate (WER), the speed of our decoding and backtracing functions, and the number of states and arcs in our WFST in each experiment. The data we used for the experiments are 354 samples of recordings provided for the coursework. In the following sections, we briefly describe our algorithm and changes from our baseline model and discuss our results with some figures.

2. TASK1

2.1. Initial system

We made an ASR class that provides a template to combine a wfst and a decoder class that recognizes speech from all of the audio files in a given path. It carries out performance tests, which include measuring substitution, deletion, and insertion counts, number of words, and WER rate for accuracy, decoding, backtracing time, and number of forwarding computations for speed, and number of states and arcs for memory. ASR class contains functions that print each result from the performance tests.

2.2. Baseline

We made a baseline WFST class and a baseline decoder class based on the previous lab exercises with minor changes. First, we created a baseline WFST that uses the given lexicon. Each phone in the lexicon has 3 states, and we used the unigram model as a baseline. Also, in this section, we didn't set any weight in all states or arcs so that our decoder moves along possible paths with the same probability.
Next, we created a baseline ViterbiDecoder that traverses through the given WFST and computes the sum of the negative log-likelihood of transition probabilities emission probabilities using the solution from Lab3 and 4.
Table1 and Table2 show our baseline results. From these results, we can see that our baseline has large WER caused by lots of insertions. We thought this is because our model doesn't have silence states thus putting a word in silence moments.

Method	WER	Decoding	Backtracing
Baseline	0.888	5.21	1.09×10^{-3}
+Final Probability	0.885	7.10	1.72×10^{-3}
+Self Loop(0.1)	2.36	5.09	8.40×10^{-4}
+Self Loop(0.4)	0.955	5.10	8.00×10^{-4}
+Self Loop(0.9)	0.728	6.40	1.12×10^{-3}
+Silence	0.467	7.31	9.20×10^{-4}
+Unigram	0.831	7.12	9.14×10^{-4}

Table 1. The results for the baseline and improvement.

Method	WER	Substitution	Deletion	Insertion
Baseline	0.888	2.163	0.248	4.096
+Self Loop(0.9)	0.728	2.194	0.448	2.746
+Silence	0.467	1.930	0.628	1.251

Table 2. Detailed evaluation

3. TASK2

In this section, we add some probability features which were omitted in the baseline model in 2.2 to WFST. We show all our results in Table 1 where we added only 1 change to the baseline model. Therefore, each 1 line shows how a particular change improves our results. Here, we don't change the structure (States, Arcs) in WFST so we don't show the results relating to the number of arcs and states.

3.1. Self loop probability

Self-loop probability expresses the probability to remain in the same state. We did 10 experiments changing that probability from 0.1 to 0.9 and checked how

self-loop probability affects our model's accuracy. The results of WER with some self-loop probability (p=0.1, 0.4, 0.9) are shown in Table 1. Also Figure 1 shows all our WER results when we change self-loop probability from 0.1 to 0.9.

From the results in Figure 1, we can see that WER

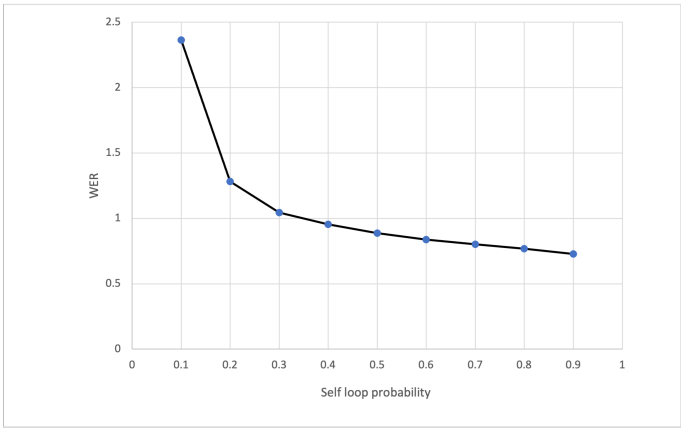

Fig. 1. Self loop probability

decreases as we increase self-loop probability. Given the result in Table 2, we thought this is because the baseline model cause error mainly due to inserting too many words, and the self-loop model reduces this insertion by staying at each state longer.

3.2. Final probability

Final probability stands for the probability to come to an end in each state. We calculated this probability by checking every given transcript and counting the number of each word that is located in the last position in a transcription. Our final probability is shown below.

```
{
     'a': 0.0028,
     'of': 0.011,
     'peck': 0.076,
     'peppers': 0.371,
     'peter': 0.101,
     'picked': 0.281,
     'pickled': 0.045,
     'piper': 0.098,
     'the': 0.005,
     "where's": 0.005
}
```

Listing 1. Our final probability

This final probability expresses that recordings are the most likely to come to an end with "peppers".
Table 1 includes our results using this final probability model. Compared to the baseline model, we can see that the final probability model doesn't bring clear improvement in WER. Unlike the model in section 3.4, this final probability model doesn't have silence states, and this model needs to fill in all timestamps with a high probability word. Therefore, at the last of sentences when recorders are about to stop recording, this model tends to suggest short words, for example, "a" or "the". We thought that is the reason why the final probability effect was cancelled out and this model doesn't bring much improvement.

3.3. Unigram-based transition probability

As a baseline, we adopted the unigram model with each phone having 3 states. Here, we calculated the unigram probability by counting each word's appearance in given transcriptions. The list below shows this unigram probability.

```
{
     'a': 0.061,
     'of': 0.105,
     'peck': 0.111,
     'peppers': 0.132,
     'peter': 0.133,
     'picked': 0.110,
     'pickled': 0.115,
     'piper': 0.114,
     'the': 0.062,
     "where's": 0.053
}
```

Listing 2. Our unigram probability

From the results shown in Table 1, we can see that this unigram probability model doesn't bring much improvement to our baseline model. We thought this is because unigram probability is almost equal ($p \simeq 0.10$) to each word as we can see in Listing 2. Most students read some words which are randomly picked up from "Peter Piper picked a peck of pickled peppers. Where's the peck of pickled peppers Peter Piper picked?", which makes unigram probability almost equal.

3.4. Silence State

In this section, we added 5 silence states to our unigram model. Following the given instruction, we adopted an ergodic structure for silence states. This structure is shown in figure 4.

By using these 5 states, every time we go back to the

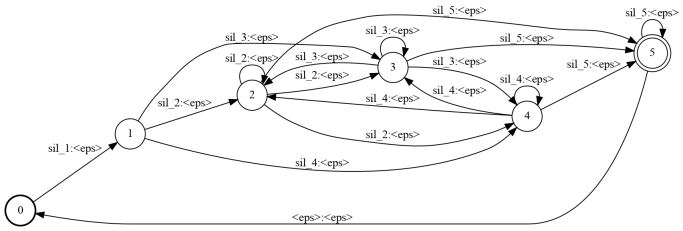

Fig. 2. Silence States

start state, we can move to silence states for a certain time probabilistically.

Table 1 and Table 2 shows that these silence states improve our baseline the best. As we can see the great decrease in Insertion in Table 2, the reason would be because silence states contribute to reducing our many insertions.

4. TASK3

In this section, we made a pruning decoder class using a sort of beam search which replaces the baseline Viterbi decoder. The main difference from the original decoder is that it can prune the path in different methods. The class receives hyperparameters which decide the pruning method and the value of the pruning threshold. Two methods used here are acoustic pruning and histogram pruning, which are referred from the following reference. [ON00] In this experiment, different threshold values are tried in the two methods. The baseline WFST is used for both decoders, so they have the same structure of the same number of states and arcs. Detailed values are explained in the subsections and the results are shown in Table 3.

4.1. Acoustic pruning

Acoustic pruning eliminates hypotheses that score below a certain threshold at each time step. The threshold is computed via a given threshold hyperparameter. The acoustic pruning decoder receives a threshold parameter between 0 and 1, which is multiplied by the maximum probability at that time step and that value is set as a threshold.

The results of the experiments on the threshold value ranging from 1e-1 to 1e-9 decreasing by one-tenth are shown in Table 3. As the threshold decreases, a smaller ratio value is multiplied by the maximum probability each time step. Only the states that have possibilities larger than the previously calculated threshold value at time t-1 are processed to compute the next Viterbi value at time t. Consequently, a lower threshold yields more states to be searched, leading to more forward counts thus slower decoding and backtracing time. However, the benefit is that exploring more states ensures higher accuracy, lower WER.

Meanwhile, the WER results of the threshold from 1e-1 to 1e-6 are counter-intuitive, in the sense that the error rate increases as the threshold decreases. This happens because setting a big threshold parameter makes the state search size too small, not big enough to derive a solution arriving at the end. In this case, the decoder predicts nothing, so the WER rate is always calculated as 1 = (0 substitution + n deletions + 0 insertions) / (n words). Thus, as there are more cases with no predictions produced, the WER rate gets closer to 1 than the intended value.

4.2. Histogram pruning

Histogram pruning is similar to acoustic pruning in the sense that it removes the hypothesis at each time step that is unlikely to happen. However, the main difference lies in the way of computing the threshold. While acoustic pruning limits a boundary of the value to overcome, histogram pruning limits the number of total hypotheses to be processed. The size of the processed set is decided via the threshold hyperparameter. The threshold is a ratio between 0 and 1, and it is multiplied by the total number of states, which works as the limit of the number of states saved to be computed at each time step.

In the experiment, the threshold value is changed from 0.1 to 0.9 by a 0.1 difference. The threshold value works as the ratio of how many states out of the whole should be explored. When the value is 1, all of the states are searched and it is the same as not pruning the paths. The results in table 3 show that as the threshold value gets larger, more states are explored, having more forward counts, as well as longer decoding and backtracing time. The WER rate increases and then decreases, for the same reason, explained in the previous subsection.

Method	Thres	WER	Decode	Backtrace	Forward
Baseline		0.888	5.21	1.09×10^{-3}	1.06×10^{5}
Acoustic	1e-1	0.960	2.18	6.33×10^{-4}	6.61×10^{3}
	1e-2	0.968	2.31	6.87×10^{-4}	9.61×10^{3}
	1e-3	0.975	2.41	7.60×10^{-4}	1.26×10^{4}
	1e-4	0.998	2.53	7.23×10^{-4}	1.58×10^{4}
	1e-5	1.030	2.70	8.80×10^{-4}	1.90×10^{4}
	1e-6	1.031	2.95	8.47×10^{-4}	2.26×10^{4}
	1e-7	1.000	3.09	9.43×10^{-4}	2.68×10^{4}
	1e-8	0.972	3.28	8.44×10^{-4}	3.15×10^{4}
	1e-9	0.951	3.43	8.94×10^{-4}	3.62×10^{4}
Histogram	0.1	1.003	1.93	7.14×10^{-4}	1.07×10^{4}
	0.2	1.080	2.36	1.02×10^{-3}	2.21×10^{4}
	0.3	1.021	2.64	9.27×10^{-4}	3.29×10^{4}
	0.4	0.937	3.04	1.06×10^{-3}	4.46×10^{4}
	0.5	0.926	3.44	1.27×10^{-3}	5.61×10^{4}
	0.6	0.912	3.76	1.25×10^{-3}	6.66×10^{4}
	0.7	0.906	4.16	1.39×10^{-3}	7.79×10^{4}
	0.8	0.888	4.45	1.48×10^{-3}	8.77×10^{4}
	0.9	0.887	4.94	1.70×10^{-3}	9.81×10^{4}

Table 3. The results of pruning.

5. TASK4

5.1. Bigram model

When we are using the unigram model, we can move to any words or phones according to a given probability. However, when we follow our grammar, each word or phone is affected a lot by the previous word or state. Therefore, we calculated bigram probability using the nltk[21]library and nested dictionary. The list below shows some bigram probabilities. Here, the first level key expresses 1st word, and the second level key means the following word.

```
{'eps':
    {'of': 0.0254,
     'piper': 0.0592,
     'peck': 0.0478,
     'pickled': 0.0478,
     "where's": 0.2197,
     'peter': 0.3859,
     'peppers': 0.0478,
     'a': 0.0619,
     'picked': 0.04225,
     'the': 0.0619},
 'of':
    {'of': 0.01337,
     'a': 0.03678,
     'peck': 0.03010,
     'eps': 0.01337,
     'peppers': 0.1237,
     'pickled': 0.6354,
     'the': 0.026755,
     'peter': 0.076923,
     'picked': 0.02341,
     'piper': 0.02006},
 'a':
    {'peter': 0.039772,
     'eps': 0.00568,
     'a': 0.00568,
     'piper': 0.03409,
     "where's": 0.011363,
     'peck': 0.7784,
     'pickled': 0.05113,
     'peppers': 0.02840,
     'of': 0.01136,
     'picked': 0.03409},
     ...
}
```

Listing 3. Our bigram probability

Method	WER	Arcs	Decoding	Backtracing	Forward
Baseline	0.888	230	5.21	1.09×10^{-3}	1.06×10^{5}
Unigram	0.831	230	7.12	9.14×10^{-4}	1.06×10^{5}
Bigram	0.775	316	5.32	7.01×10^{-4}	1.06×10^{5}

Table 4. The effects of WFST architecture

Table 5 below shows the results regarding 1.WER and 2.Efficiency about each WFST architecture and probability.

The results in Table 5 indicate Bigram model is better than the unigram model in terms of accuracy. As you can see in List 3, the bigram model has unevenly distributed probability distributions, unlike the unigram model. For example, List 3 shows 'pickled' is most likely to appear when given 'of', whose pair appear in the given sentence. Therefore, we can assume students picked up a "chunk of words", not a single word randomly from given a sentence, which makes the bigram model more useful than the unigram model.

5.2. Tree structured lexicon

While maintaining the semantics of the lexicon, changing the syntax of the lexicon to the simpler version benefits in having lower memory and decreased search

Method	WER	Arc	State
Baseline	0.888	230	116
Tree	0.831	183	84

Table 5. The results of tree structured lexicon

time. In the coursework, a tree-structured lexicon as shown in figure 3 is used to replace the original lexicon. Tree-structured lexicon shares the intermediate path that has the same phone sequences in common. The states searching specific phone sequence are not redundant and the number of the states are reduced. Since the search size is smaller, a faster search is possible. The results in table 5 show relevant results having less number of arcs and states, shorter decoding and backtracing time, as well as decreased forward counts.

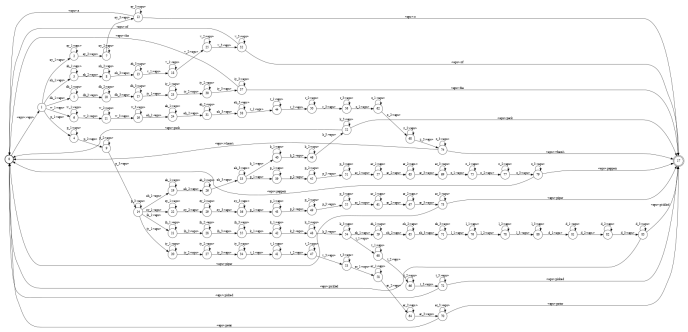

Fig. 3. Tree structured lexicon

References

[ON00] S. Ortmanns and H. Ney. “The time-conditioned approach in dynamic programming search for LVCSR”. In: *IEEE Transactions on Speech and Audio Processing* 8.6 (2000), pp. 676–687. DOI: 10.1109/89.876301.

[21] *nltk*. [Online; accessed 30-March-2022]. 2021. URL: https://www.nltk.org/.