Automatic Speech Recognition - Coursework 2

S2030247, S2620698

March 27, 2024

All experiments were performed using 329 recordings of utterances, using 12 words from a common lexicon. Each phone in each word is represented using a 3-state WFST - all WFSTs were created using the OpenFST Python library in a Jupyter notebook.

The performance is measured with 4 metrics, across all the utterances:

- The Word Error Rate (WER) metric, defined as

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

  where S = the number of substitutions, D = deletions, I = insertions and N = the number of words in the actual utterance. (When we say we calculate this across all utterances, we mean we sum up the S, D, I and N counts for all utterances.)

- The time taken to run the backtrace and decode operations in the Viterbi decoder (taken from Labs 3-4). This sometimes varied greatly when performing operations on different DICE machines, so some values may be inconsistent - we have taken care to ensure that this effect was minimized.

- The number of forward computations in performing the decoding.

- The number of states and arcs in the WFST.

With each experiment, we start from the baseline model, such that the only variable affecting the performance is the one we're actively changing.

1 Task 1 - Initial systems

The baseline WFST has an initial state with 12 outward arcs, each going to the start state for a different word in the lexicon, each with uniform probability 1/12. Every state has a self-loop arc with a probability of 0.1 and an arc transitioning to the next state with probability 0.9 - the transition to the final state in each 'word branch' has the word itself as the output symbol, while every other state has an <eps> output. The final state, with an associated final probability, then transitions back to the start state with probability 1, to allow for words in sequence. The topology of the WFST is shown in Figure 1.

This WFST, representing the transition probabilities of a Hidden Markov Model (HMM), is combined with the observation probabilities from a pre-trained model i.e. the probability of observing a phone symbol at time t, trained using the provided utterances. The Viterbi decoder then estimates the sequence of states with the maximum likelihood, given the observation sequence i.e. the states along the most likely path in the HMM given the observations. The recognised word sequence is then the output symbols along that path. The results from the baseline model, along with all others, can be seen in Table 1.

The WER is incredibly high at 144.88% - from the S, D, I counts, we can see that most of the errors were insertions (i.e. additional words). The root cause of this is probably to do with the fact that the current WFST assumes that there is no gap between speaking one word and speaking the next. As soon as a 'word branch' is done, it transitions back to the start state if there are more observations, meaning any gaps or moments of silence would potentially lead to more words being emitted.

The self-loop/transition probabilities add to this issue. If there is more than one frame corresponding to each state, then having the self-loop probability be as low as 0.1 would negatively affect the accuracy of the WFST, as paths reaching the word-emitting states more often would be considered more likely; this exacerbates the problem with the lack of any 'silent' states. An inaccurate observation model could also contribute to the high error rate, though this is not something we have any control over. With the 329 recordings we use, the total number of frames (measured using the 'wave' Python library) is 25017183, while the number of phones in all transcription times 3 (3 states per phone) is 27405 - therefore, each state HAS to correspond to multiple frames.

Additionally, from these results we can see that substitutions are the second most frequent type of error i.e. our model

Model param	WER / %	# S, D, I	Decode time / s	Backtrace time / s	# Forward comps.	# States, arcs
Baseline						
	144.88	605, 19, 2885	321.09	0.083	32307072	116, 230
Changing the self-loop probability						
0.2	103.47	618, 36, 1852	320.61	0.079	32307072	116, 230
0.5	75.63	607, 56, 1169	319.99	0.083	32307072	116, 230
0.8	66.06	583, 96, 921	320.58	0.085	32307072	116, 230
0.99	59.41	601, 203, 635	322.58	0.085	32307072	116, 230
Changing the final probability						
0.8	144.88	605, 19, 2885	320.97	0.087	32307072	116, 230
0.6	144.88	605, 19, 2885	320.17	0.079	32307072	116, 230
0.4	144.88	605, 19, 2885	320.83	0.079	32307072	116, 230
0.2	144.88	605, 19, 2885	317.27	0.073	32307072	116, 230
0.01	144.88	605, 19, 2885	317.97	0.074	32307072	116, 230
Adding the silence model						
	55.62	631, 56, 660	337.27	0.065	34324527	121, 245
Unigram probabilities for word transitions						
	140.67	605, 22, 2780	325.16	0.070	32307072	116, 230
Adding a pruning threshold						
0.05	184.52	792, 21, 3656	123.54	0.080	11007716	116, 230
0.1	207.39	926, 102, 3995	106.22	0.066	9058312	116, 230
0.15	243.39	1168, 163, 4564	91.95	0.079	7520122	116, 230
0.2	252.23	1183, 169, 4757	84.91	0.067	6856198	116, 230
0.25	260.49	1197, 206, 4906	79.92	0.072	6382580	116, 230
0.3	266.76	1191, 230, 5040	76.21	0.060	6037868	116, 230
Improving the lexicon - allowing for multiple pronunciations						
	146.70	591, 18, 2944	352.68	0.082	35097738	127, 252
Tree-structured lexicon						
	144.88	605, 19, 2885	248.56	0.056	24843253	84, 183
Bigram probabilities for word transitions						
	130.68	546, 25, 2594	364.33	0.087	32307072	117, 323

Table 1: Results for all experiments: the value of the relevant parameter is mentioned if necessary (e.g. in the 'changing self-loop prob.', the parameter is the self-loop probability value)

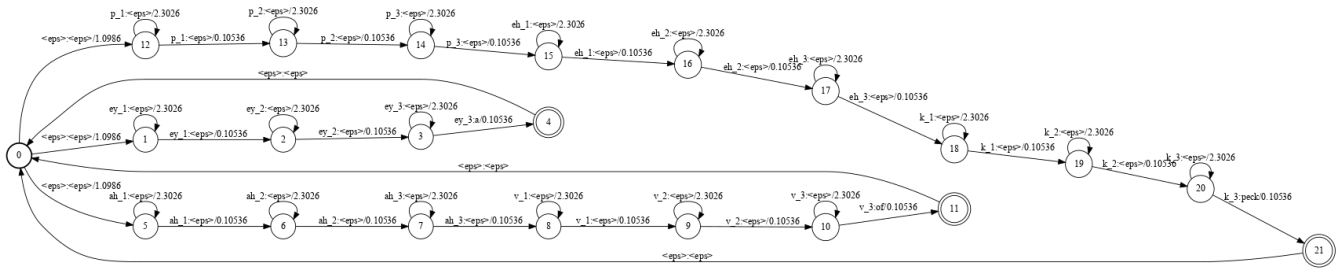

Figure 1: A smaller version of the 'baseline' WFST that only considers the first 3 words in the lexicon. The word label is emitted on the transition to the final state of each branch.

outright misidentifies the word. A potential contributor to this problem is that the start state assumes a uniform probability distribution for the transitions, i.e. each word is exactly as likely to occur as the others.

Another potential cause of substitutions is that the current parse_lexicon function (taken from the lab solutions) does not actually allow for multiple pronunciations of the same word, which the provided lexicon contains, e.g. “the” = “dh iy” or “dh ah” depending on whether it precedes a vowel. The current function stores the lexicon as a dictionary with the word as the key and the phones as a list value - for words with multiple pronunciations, only the second value is stored.

This may not be especially significant for the differing pronunciations of ‘a’ in the lexicon, but it may negatively affect the recognition of ‘the’ The pronunciation stored in the lexicon dictionary is “dh iy”, the pre-vowel pronunciation, but in terms of the words we have, that would only ever happen with “the a” or “the of”, which would never occur in an utterance (assuming perfect grammar from every speaker in which pronunciation to use).

2 Task 2 - System tuning

2.1 Self-loop probabilities

We begin here by increasing the self-loop probabilities - here, we adjust the transition probabilities to match as well, to keep the assumptions of HMMs in place. We then see if that improves the WER, going by the reasoning established in the previous section.

In our experimentation, we found that the only things that changed were the error rates (and, obviously, the S, D and I counts). The rest of the values stayed around the same as the baseline, as nothing about the core structure of the WFST or the Viterbi decoder had changed. As shown in the results, the WER performance keeps improving as the self-loop probability increases, even up to the unreasonably-high value of 0.99. Additionally, we can see that the number of deletions (i.e. words missing from the machine transcription) increases greatly alongside the self-loop probability, reaching 203 by the end.

This works because, by increasing the likelihood of paths that reach the final word-emitting states less often, we inadvertently address the problem of the WFST emitting words during periods of silence. However, the number of deletions increasing indicates that this is not the ideal method of improvement; the high self-loop probability reduces the number of words emitted **overall**, not just the extraneous ones produced during silence.

2.2 Final probabilities

The ‘final probability’, here, refers to the probability of being in the final state. As previously mentioned, the current WFST has the final probability set to 1, i.e. a cost of 0.

In this experiment, we try to decrease the final probability, to decrease the likelihood of longer paths that emit more words. We proceed in decrements of 0.2, stopping when we reach 0.01 (considered the absolute minimum, as setting it to 0 would clearly be incorrect); we then measure how it affects the system performance in terms of WER. Once again, all the other measures remain largely unchanged from the baseline, as no structural changes have been made to the WFST.

We can see that the WER and the S, D and I counts **do not change at all** from the baseline - changing the final probability had absolutely no effect. This is most likely because final probabilities don’t actually matter that much when picking the most likely path. It would add a per-word cost (negative log probability) to longer paths, but those longer paths are already more costly by virtue of passing through more states.

2.3 Silence state

To model the silence between words, we use a custom WFST - this is then included as a separate ‘word branch’ in the overall WFST. The structure of the WFST is similar to what’s described in the coursework specification, and can be seen on its own in Figure 2. The initial transition probability from the start state is then adjusted to account for this branch’s inclusion.

As can be seen from the results, this **greatly** improves the WER, and particularly the number of insertions. Additionally, the reduction in insertions does not correspond with as large an increase in other errors, unlike with the results when the self-loop probability was set to 0.99. Examining some of the utterances in Audacity, we note gaps of silence between the words, as well as large gaps at the beginning and end of the files. Having the silence state present reduces the number of extraneous words specifically during moments of silence, as the provided observation model accounts for ’silence’ inputs.

The number of substitutions has remained the same across all experiments so far, as none of the potential error sources have been resolved; we still assume a uniform probability for each individual word, we still use the slightly-incorrect lexicon implementation, and we still use the same potentially-inaccurate observation model.

In terms of the execution time, while the time for decoding has increased (understandable, due to the more complex WFST), the time for backtrace has actually decreased a little.

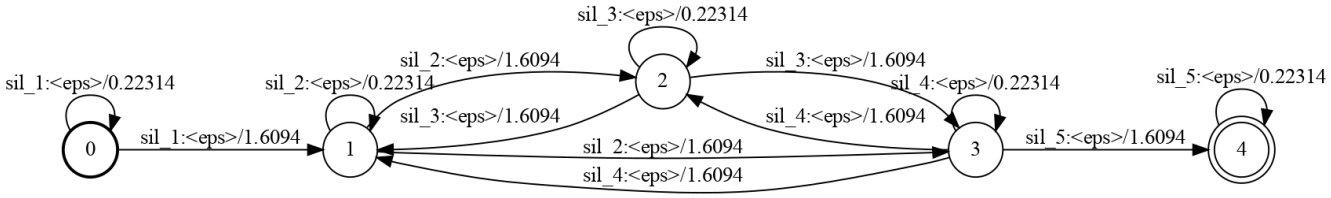

Figure 2: The 'silence' WFST - the middle 3 states follow an ergodic topology.

This might be because the inclusion of the silence branch has actually decreased the ambiguity in the model - the additional states and arcs provide more information that make it easier to distinguish the most likely path.

2.4 Word transition probabilities - unigram

With the unigram model, we use maximum likelihood estimation (MLE) to estimate the probability of each word occurring as follows:

$$P(x) = \frac{\text{No. of occurrences of x}}{\text{No. of occurrences of all words}} \tag{2}$$

We use the transcriptions of the same utterances that we test the model on. Normally, you would want to separate test and training data, but with only 329 occurrences and a limited vocabulary, the effect of this choice should be negligible. (Of course, this is different in real-world applications.) These probabilities are then used for the cost of the outgoing arcs from the start state to the word branches.

Checking the results in Table 1 though, we see that implementing this probability distribution doesn't actually result in much improvement; the WER has only decreased by 4.21%. This is probably down to the nature of the utterances themselves; given that they are random (or at least pseudo-random) sequences of words from a pre-selected vocabulary, looking at the words by themselves doesn't reveal much. Indeed, examining the unigram probabilities manually revealed that most words had a probability of around 0.1 to 0.15, with 3 notable exceptions. In a real-world scenario, however, implementing a unigram PD should have more positive effects on the model's accuracy.

3 Task 3 - Pruning

This was implemented by introducing a pruning threshold value in the forward_step function (and consequently the decode function) - for every state j at every time t, the forward_step function checks if the path cost is higher than the best cost plus the negative log probability of the threshold (the same as subtracting the threshold from the best probability). If it is, the path is not propagated.

We experiment with different pruning thresholds; we decided to keep the increment fairly small, as the initial test with 0.1 as the threshold significantly impacted the WER. We see that as the pruning threshold increases, so does the WER; however, the decode time significantly decreases with each step as well. This is to be expected; increasing the pruning threshold means more paths are pruned, reducing computational complexity, but it could also cause the algorithm to prune the optimal path and focus on a worse one.

4 Task 4 - Advanced topics

4.1 Improving the lexicon to allow multiple pronunciations

This involved a minor modification to the parse_lexicon function; if the word already exists, then a new entry is made with the key being the word plus "1" (e.g. "the1" or "a1"). The later WFST functions were then modified to make sure the branch emits the correct word symbol at the end. The WFST with the improved lexicon function can be seen in Figure 4.

Checking the results, however, we were surprised; including multiple pronunciations for words actually **increases** the word error rate by 1.98%. This may be because the words with multiple pronunciations don't actually occur too often; looking at the unigram probabilities, they occurred less frequently than almost any other word, the only exception being "where's". As such, the extra paths may just exacerbate the aforementioned problem with the lack of a silence state in the baseline (hence the higher I count), even though they do lower the number of substitutions.

4.2 Tree-structured lexicon

We use the OpenFST determinize function to create a tree-structured lexicon for our model - essentially, it creates an equivalent FST where no state has two outgoing arcs with the same input label, meaning all repeating phone states (e.g. the states for 'p', which occur in the branches for 'peppers', 'peter', 'picked', etc.) are combined into one. The effect can be seen in Figure 4.

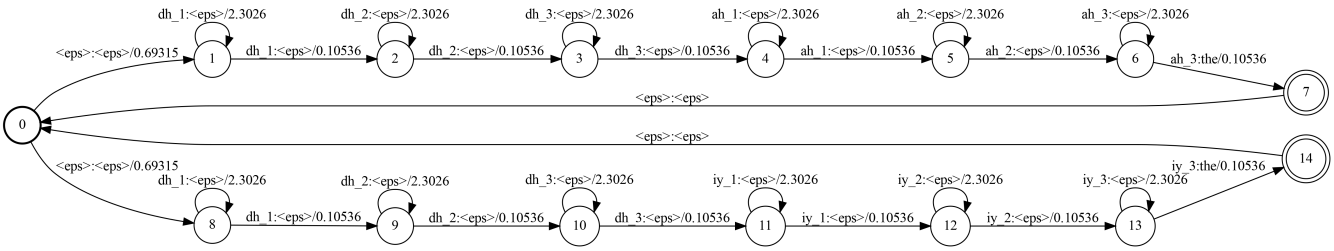

Figure 3: The results of the improved lexicon on the WFST, for both pronunciations of "the".

The determinization is incredibly effective in our case, as we use the vocabulary for a tongue twister which relies on repeated phones. While the determinized WFST performs identically in terms of WER to our baseline, as is to be expected, it has removed 32 states and 47 arcs. This results in the decode time being reduced by 22.5% and the number of forward computations going down by 23.1%. Using this tree-structured lexicon helps make the decoding process more computationally efficient, but there is a downside: this approach doesn't work when assigning unique probabilities to each word, since we don't actually know what word branch we're on based purely on the states and arcs. Using the tree structure may be especially beneficial when using the improved lexicon, as that results in more branches with shared phones.

4.3 Bigram language model

We create a bigram language model that considers the beginning and end of sentences as tokens: <s> and <\s>. This model is based off the word branch model, except it defines a common end state for all branches. At the end of each branch, outgoing arcs are added to the start of all other word branches, as well as the end state, depending on the bigram probability (i.e. the arc is weighted if the bigram probability exists, but is outright excluded if the probability is 0). These additions result in a WFST with 117 states and 323 arcs, an increase of 93 arcs. The bigram WFST for the first 3 words can be seen in 5

Unfortunately, the implementation of this model did not yield a significant improvement in the model's performance; it resulted in a WER reduction of 14.2%, which pales in comparison with the effect that the silence branch had, for example. The improvement that does exist is probably due to the existence of common bigrams in these seemingly-random utterances, such as "peck of" (p=0.71) or "a peck" (p=0.76), revealing a degree of human bias (since both bigrams were in the original tongue-twister).

Additionally, mostly reusing the branch structure paid off; while the increased number of arcs resulted in the decode time increasing by 43.24 seconds, the number of forward computations remained identical to that of the baseline model. In a real-life application, the use of a bigram model would most likely yield even better improvements to the WER.

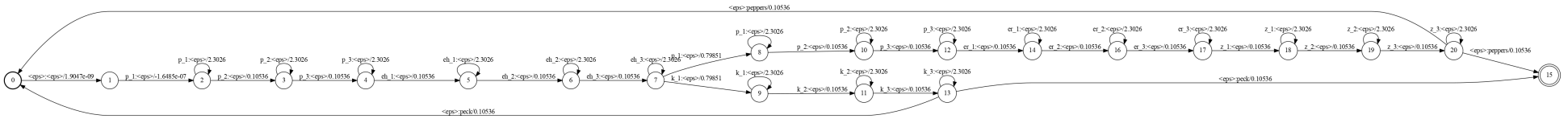

Figure 4: The tree-structured WFST for the words "peck" and "peppers".

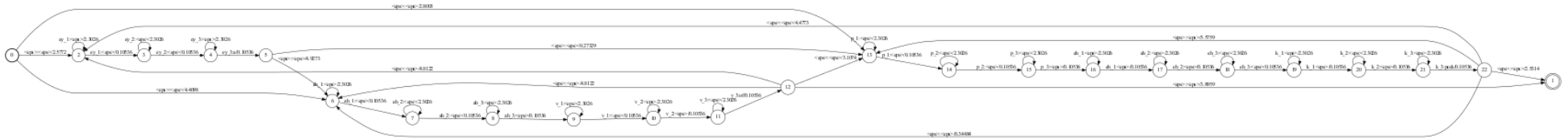

Figure 5: The bigram WFST for the first 3 words in the lexicon.