ASR Assignment 2022-23

S1852913, S2446971

I. Introduction

Automatic Speech Recognition (ASR) is a particularly critical area of research in the field of Artificial Intelligence thanks to it's use in facilitating human-computer interaction through language models. The primary goal of ASR is to identify and convert spoken language into written text, thus finding particular use in areas such as transcription services, voice assistants and healthcare amongst others. It is an area of research which has benefited from significant advancements in recent years, however due to the very complex nature of speech and speech signals it remains one of the hardest tasks in the field of Artificial Intelligence.

This report presents a comprehensive approach to improving an ASR system by implementing various methods to enhance speech recognition accuracy while simultaneously optimising computational efficiency. We measure accuracy in terms of Word Error Rate (WER),

$$\textbf{WER} = \frac{\text{Substitutions + Deletions + Insertions}}{\text{Number of words in reference transcription}}$$

a widely-used metric that quantifies the difference between the recognized and reference text. Here, Substitutions (S) refers to situations where substituting a transcribed word for another would result in the desired output. Deletions (D) and Insertions (I) each refer to adding or removing a selected desired word from the transcribed reference to obtain the desired output.

As an example, we may have:

Reference		Pickled	of	Peter	Peppers
Action	$\downarrow$ I	$\downarrow$ S	$\downarrow$	$\downarrow$ D	$\downarrow$
Desired	a	Peck	of		Peppers

This example would have a WER of $\frac{3}{4}$ = 0.75 as there is 1 of each of an Insertion, Substitution and Deletion and only 4 words were transcribed. We note that, unlike some other error rates which only take values in the range (0,1), WER can take any positive real value, namely if very few words are transcribed and multiple additional Insertions are required this may indeed lead to a WER significantly above 1.

Additionally, we assess computational efficiency through a combination of metrics. Firstly, 'Decode Time' refers to the time taken to obtain the likeliest sequence of hidden states given an observation (input audio signal). 'Backtrace Time' refers to the time taken to recover the best sequence of hidden states by following the backpointers once a forward pass is completed. Finally 'Forward Computations' Refers to the total number of calculations performed for a single forward pass of the Viterbi Algorithm [1] which we will use. We will also analyse the memory requirements in terms of the number of states and arcs in the Weighted Finite State Transducer (WFST).

Our research is divided into four main parts. Firstly, we train a baseline model using a Viterbi decoder with a WFST designed to recognise a sequence of words from a vocabulary. We use a WFST as it allows us to compactly represent and manipulate a possible search space for our speech recognition task. Meanwhile, the Viterbi algorithm can be briefly described as a dynamic programming algorithm used to calculate the most likely sequence of hidden states in a WFST given an observation.

In the second part of our study, we improve the performance of our system through the implementation of different states as well as careful parameter tuning. These adjustments include using transitions to each word based on unigram word probabilities, the introduction of silent states as well as varying the transition and final probabilities in an attempt at achieving better recognition performance in the form of a low WER.

The third part of our work involves the implementation of two different pruning strategies on the Viterbi decoder. Through pruning we aim to reduce the search space and thus improve computational efficiency while sacrificing a minimal amount of accuracy.

Finally, we delve into advanced topics to further enhance our ASR system's performance. These topics use the following approaches: Bigrams and Back-off, in combination with silent states. Once again with the goal of improving upon all our previous work with our model. Our full set of results can be viewed in Table 1 in the appendix (A).

It is worth noting that probabilities and weights will be used interchangeably, as the probabilities we are adding to the arcs, are later translated into weights within the WFST.

II. Task 1 – Initial systems

We will be training a model to recognise speech from the famous tongue-twister *'peter piper picked a peck of pickled peppers where's the peck of pickled peppers peter piper picked'*. Due to the limited datasize of our audio recordings (317) we decided to use all the data when tuning our parameters, while this may lead to overfitting we felt it was the best way to get the most out of our model.

We begin by implementing a baseline WFST, with self-loop, transition, and final state probabilities of 0.5. Since all arcs had the same probabilities, the probability of traversing to the next state in a sequence at any timestep was the same as staying in the current state, while from the start state moving to any state is equiprobable. We maintained the number of phone states as 3 as is common practice in speech recognition, splitting the phone into it's initial, middle and final parts.

Our initial analysis was done following the proposed metrics: Average Word Error Rate, Decoding time, Backtrace time, and Memory. Except the latter, all of these metrics were calculated by averaging over all the audio files in the class recording folder. These were our results:

- **WER (S, D, I)**: 0.7328 (S: 2.20, D: 0.25, I: 2.76)
- **Average Decoding Time**: 2.28414 with an average of 111970.17 forward counts per audio
- **Average Backtrace Time**: 0.00028583
- **Memory**: 230 arcs and 116 states

Additionally, we provide a metric showing the average WER for each audio length (**in words**). This could give some insight into how the current model is performing on varying audio lengths.

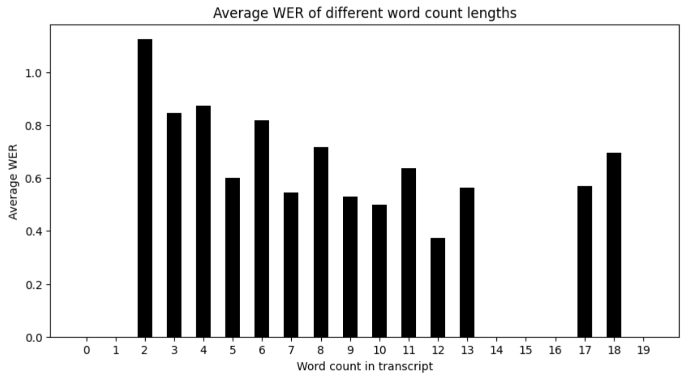

Fig. 1. :Graph showing the Average WER for different word count lengths in audio files

Fig. 1 highlights a trend of a slight decrease in average WER as the number of words in the transcript increases, given our limited dataset it is difficult to extrapolate this generally. Nonetheless, intuitively, as the computation of WER divides by the Number of words in the reference transcription we believe this may well generalise.

III. Task 2 – System tuning

A. Adding Unigram probabilities

Hypothesis: Adding unigram probabilities will reduce the WER because the model will move to words according to how salient they are, producing more accurate transcriptions of the audio. This will have no effect on the other metrics as there is no change to the memory.

Procedure: There are multiple ways to calculate the unigram probabilities for this exercise. We chose to calculate them through our given passage: *Peter Piper picked a peck of pickled peppers. Where's the peck of pickled peppers Peter Piper picked?*.

To calculate the unigram probabilities we identify each unique word in the passage, count their appearances and then divide it by the total number of words. For example, *peck* appears twice and the total number of words is 17 giving a unigram probability of 2/17. The unigram probability is allocated respectively to each arc that goes from the start state to each word in the WFST described in the baseline model.

Results: The results for unigram WFST when including these probabilities are the following:

- **WER (S, D, I)**: 0.704257195 (2.16, 0.29, 2.58)
- **Average Decoding Time**: 2.29282 with an average of 111970.17 forward counts
- **Average Backtrace Time**: 0.00024069
- **Memory**: 230 arcs and 116 states

Discussion: The WER had a significant decrease in WER from the baseline method. This effect is caused by a better representation of the word occurrences, which in turn makes the model choose the next word more accurately each time. However, this method does not take into account the silent breaks a person takes when concatenating words in speech. In one of the following exercises, where we include the silent state and unigram probabilities, we calculate these in a different, and more realistic manner, thanks to inclusion of the silent states.

In terms of the time metrics, the backtrace shows similar results as the baseline method. The decoding time on the other hand, seems to have slightly decreased, but the forward computations have remained exactly the same. If the forward computations has not changed and nor has the decoding algorithm, then the decrease in decoding time can be dismissed and can be explained by a change in computing power.

B. Adding Silent states

Hypothesis: Including silent states to the baseline method will reduce the WER as it makes a more accurate representation of how people speak, allowing for transition through the silent states when there are silent breaks in the audios. This will also increase the memory, and therefore the forward computations and decode time - due to the addition of extra states and arcs compared to previous model.

Procedure: To decide the structure of the silent states we followed the specification which recommended an ergodic structure for the 2nd,3rd, and 4th state. Additionally, we included arcs from these states to the final state to allow situations in which the silence between words is short. The silent states are included in the WFST as if it was another word, where the start state is connected to the first silent state through an arc, and the final state is connected back to the start state. The isolated zoomed structure can be visualized in the following image.

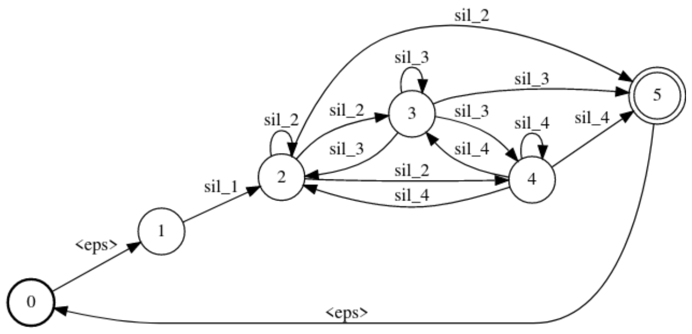

Fig. 2. Hidden Markov Model showing the structure of the silent states included in the WFST with silent states

This structure was added to the baseline method as if it was another word. The parameters were kept the same - self loop, transition, and final probabilities were given values of 0.5 . This includes those for the silent states.

Results:

- **WER (S, D, I)**: 0.5366 (2.14, 0.28, 1.57)
- **Average Decoding Time**: 2.3479 with an average of 119343.81 forward counts
- **Average Backtrace Time**: 0.0002248
- **Memory**: 245 arcs and 121 states

Discussion: In this implementation we can see the effect of including silent states, the WER has decreased from 0.7328 (baseline) to 0.5366. The silent states act as buffer states where after each word the model can wait until the next acoustic signal hits and it can transverse to a phone state. This is a more realistic representation as the words are not concatenated one straight after the other without silent pauses.

Again, there is no significant difference in the average backtrace time. There seems to be a slight increase in the average decoding time, and average forward counts compared to the previous experiment. This is caused by the increase in the number of total states.

C. Combining Silent states and Unigram Probabilities

Hypothesis: Adding silent states to the baseline model, and transition to the words or silent states with unigram probabilities will decrease the WER substantially as this produces the most accurate representation of human speech out of the previous models. The other metrics will be similar to those of the baseline with silent states.

Procedure: To combine the silent states with the unigram probabilities we modified the way in which the unigram probabilities are calculated. We still used the passage, but now we include each space between words as a *silent word*. So, now the total number of words becomes 33, and there are now 16 *silent words*. This way the probability of moving from the start state to the silent state is the most likely. Again, to maintain consistency the probabilities for self loops and transitions is kept at 0.5.

Results:

- **Average Word Error Rate (WER)**: 0.5139
- **Average Decoding Time**: 2.3472 with an average of 119343.81 forward counts
- **Average Backtrace Time**: 0.0002155
- **Memory**: 245 arcs and 121 states

Discussion: As we can see from the results the WER has decreased from our previous best result (0.5366), where we only had the silent states, to 0.5139. This combination of unigram probabilities and silent states makes a more accurate representation of when to choose which word to transverse next.

D. Weight Tuning

Hypothesis: The weights can be tuned as the current 0.5 probability for each possible action (self loop, transition, end sequence) is a wrong depiction of how the audios were recorded. Given that people speak at different paces, and have different accents, there must be a specific proportion of probabilities that represents them better.

Procedure: For this task we initially considered tuning the transition weights and the final weights. The former takes the transition from one state to the next, and the latter the probability of ending the sequence once in a final state. Performing a grid search with the full dataset would be very expensive as one run could take approximately 15 minutes. Therefore, to check if the effect of changing the transition weight and final weight is significant we selected 20 random audio samples from the set. The selection of these audios is the reason why the WER that will be observed in the next graph is different to those seen in the previous section. The chosen audios are possibly harder for our model and therefore it shows larger WERs.

The chosen transition weight in this experiment is given to every arc that transitions from one state to another. Similarly the final weight is awarded to every final state *f.set_final(final_state, final_weight)*. The self loop is maintained with the same weight as previous experiments: 0.5. As explained before, there are three possible actions to take in a state, self loop, transition, and be final. The probability of choosing one is given by the weights, by maintaining the self loop probability, we can check the proportions of these three actions. The following graph shows our initial experiment showing how the WER varied with different final and transition weights.

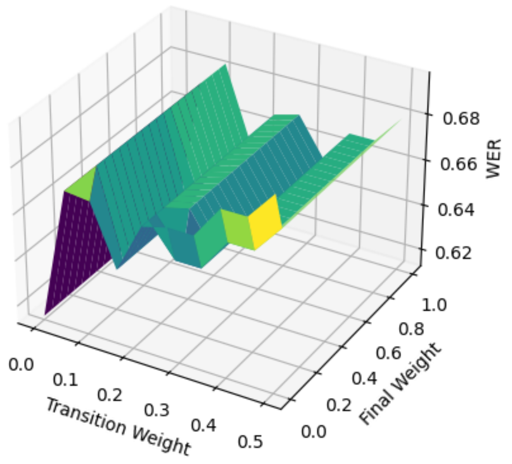

Fig. 3. Graph showing the Average WER for different transition and final probabilities, with a constant self loop probability of 0.5

From this graph it is clear that the final weights do not have any effect on the WER. We believe this is because all final weights are equal and therefore they are all equally likely to end in a final state. Therefore, to see a difference we must change the final weight of certain final states.

We performed a test where we only changed the final probability of the final silent state, and left the other final states at a constant probability. We expected to see an effect on the WER as the final weight was changed, but there seemed to be no effect. To ensure that this was not because we were using too small (or large) values we attempted extreme values, and again these did not produce any difference. From this experiment we can only conclude that for our best model (unigram silent) the final state probability has no effect on the Word Error Rate.

Given that the final weights have not produced significant difference in the WER, we followed our investigation by only testing the transition weight, and leaving the final weights to have equal probabilities.

Results: The following figure shows our results in tuning the WFST weights:

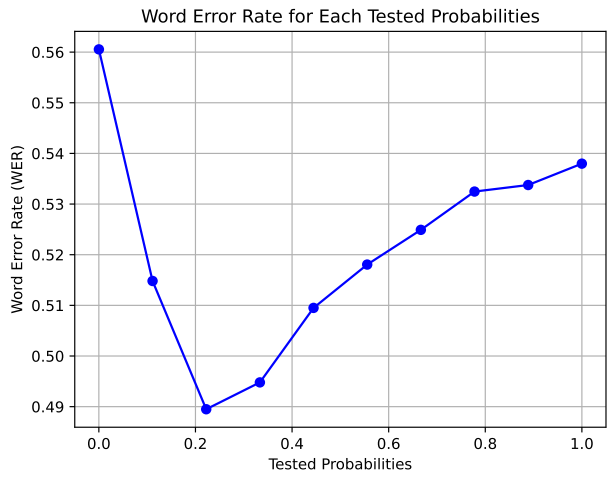

Fig. 4. Logarithmic graph showing tested Transition Weights vs obtained WER

The best WER was achieved by a transition probability of 0.2625. This optimized model has the following metrics:

- **WER (S, D, I)**: 0.4854 (2.05, 0.39, 1.23)
- **Average Decoding Time**: 2.562605 with an average of 119343.81 forward counts
- **Average Backtrace Time**: 0.000258661666
- **Memory**: 245 arcs and 121 states

Discussion: We observe, as expected, that the WER converges to 1 as the Transition Weight is decreased exponentially. In this situation, the transition weight is so low that there is little to no transition between states and so the model directly does not output any words. It therefore requires the full size of the reference transcription in insertions, providing a WER of 1.

The transition probability of 0.2625 entails that the model performs best when it is double as likely to perform a self loop than to transition to the next state. Additionally, we also say that the average WER for different word count lengths was more levelled compared to that of the baseline analysis - meaning this model is able to generalize much better to our set of audios.

Conclusions: To summarize, in this section we tested the effect of adding unigram probabilities and adding silent states to the baseline model. We observed that both of these features decreased the WER compared to the baseline model. The addition of silent states improved the accuracy with a 27% decrease in WER, compared to the baseline. There was a cost in including silent states, this was seen in an increase in memory and decode time - as this model had more states and arcs. We then tested the combination of unigram probabilities with silent states, which produced the best model with a 25% decrease in WER. We also observed that changing final state probabilities had no effect on the accuracy of the model, and therefore from now on in this investigation we will not consider it another parameter in our tuning.

IV. Task 3 - Pruning

To motivate how our pruning algorithms (losely based on Beam Search [2]) work, we briefly explain the idea behind how probabilities of path sequences are computed along a path within a WFST.

We write the probability of the first output ($y^{<1>}$) given an input sentence x as $P(y^{<1>}|x)$, furthermore the probability of a second output ($y^{<2>}$) at the second timestep must inherently depend not only on the input sentence x but also on the previous output $y^{<1>}$ having been emitted at the previous timestep. As we represent the probability of a second output ($y^{<2>}$) being output such that the initial path of a sentence follows $y^{<1>}$, $y^{<2>}$ by $P(y^{<2>}|x, y^{<1>})$. Then, from Baye's Rule the probability of (only) these two outputs together would be given by:

$$P(y^{<1>}, y^{<2>}|x) = P(y^{<1>}|x)P(y^{<2>}|x, y^{<1>})$$

This can be extrapolated for further words and states along the WFST until it reaches a final state and returns the predicted output sequence of words given the original input x.

Pruning is thus an efficient technique to improve on the speed and memory requirements of our Viterbi Decoder as it can allow us to limit which combinations of paths ($y^{<1>}, ..., y^{<n>}$), where $y^{<n>}$ denotes the end of an outputted sequence, should be explored earlier on. Pruning allows us to only retain a certain (fixed or relative) number of best partial solutions as potential candidates for the optimal path through our WFST. We use algorithms to efficiently prune the search space and reduce the number of possible paths to be considered. This not only improves the speed of the decoder but also significantly lowers the memory requirements, as less space is needed to store the partial paths. Pruning generally requires some form of parameter tuning as strict pruning (for example greedy pruning) may prematurely prune optimal paths and thus increase the WER of our model at the expense of a low decode time.

We propose two different pruning methods, one which is percentage based, thus computing a fixed percentage of possible paths at each time step (this compunds between timesteps) and a second one based on thresholding which is adaptive (in the sense that there is no fixed metric to determine how many paths will be pruned at a given timestep).

A. *Percentage Pruning method*

Inspired by the beam search algorithm we modified the pruning selection to retain paths based on probabilities rather than a fixed number. This Percentage Pruning technique maintains the top percentage states at each timestep (measured by their probability) and sets the rest to a probability of 0.

For example, if we have a percentage pruning threshold of 20% and in the current time step there are 100 states, we will explore paths through only the 20 states with highest probability and 'prune' the rest.

The graph below shows the effect the percentage pruning threshold has on the WER and the decode time.

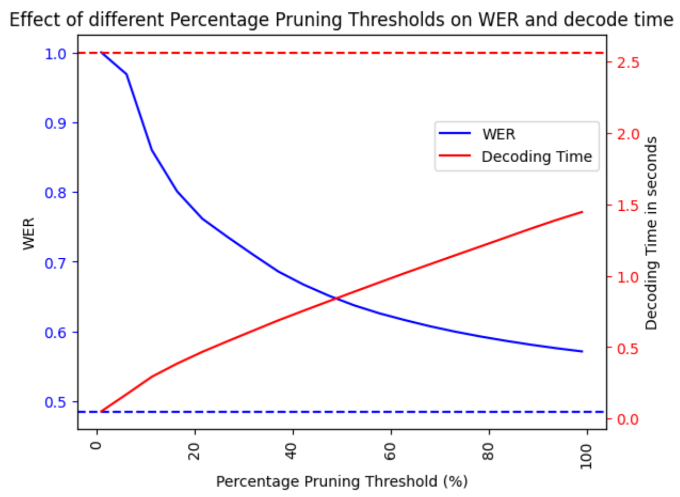

Fig. 5. Graph showing WER and decode time on different pruning thresholds with percentage pruning strategy

It is worth noting that we did not compute the threshold of 100% or 0% to build this graph, the range was from 0.1% to 99.9%, a percentage of 0 would mean not propagating any state and therefore no path reaches the end of observations. On the other hand, having a threshold of 100% would be like running the model without pruning - this limit is shown with two asymptotes in the graph.

From the graph in figure 5 we can see that the WER follows a negative logarithmic curve, whereas the decoding time has a positive linear trend. It gets exponentially harder for the WER to improve as we increase the threshold. The gap between the WER and decode time lines with their respective asymptotes can be explained by exploding nature of the pruning strategy. By exploding nature we mean that despite having a large threshold such as 99.9%, if the number of states is large enough, we will always set a number of states to zero probability at each time step. These states will not be propagated and in turn it will prevent their propagation. This way, even by just pruning one initial state, the number of forward computations and states are reduced significantly. The only way to reach the asymptote values is by having a 100% threshold. The graph presented in Fig. 5 shows these results.

B. Threshold Based Pruning

We implement a second pruning technique based on a threshold to contrast our previous pruning technique. With Fig.6 showing our results. Here, the threshold dictates the maximum allowed difference between a potential path's cost and the minimum cost among all the states at time 't'. Unlikely paths (with a difference in cost above the threshold) are pruned in favour of decreasing computational time.

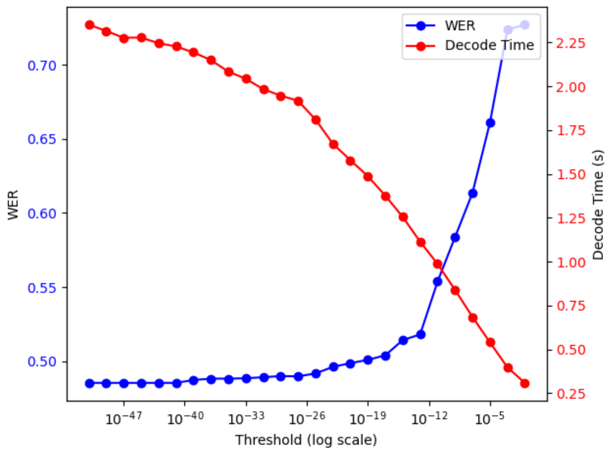

Fig. 6. Graph showing the Average WER and average Decode Time for different thresholds in logspace

This method has the added benefit that it is adaptive between timesteps, unlike our method above which would select

The justification for this method of pruning is to reduce the search space and computational complexity during the decoding process. By pruning paths with a cost difference greater than the threshold, only the more promising paths are retained, which speeds up the overall computation. This method is based on the observation that, in many real-world applications, low-probability paths are unlikely to become the best path in later steps and can be safely discarded.

We represent the minimum cost (i.e., the highest probability in log space) among all states at time t. By comparing the cost of each potential path with the minimum cost at that time step, this algorithm can determine whether the path is significantly less likely than the current best path. If the difference between the cost of the current path and the minimum cost is greater than the threshold, the path is pruned with its probability set to zero and thus not considered in further time steps.

$$V_j(t) - min\ cost > -\log(threshold)$$

Where $V_j(t)$ is given by $V_j(t) = \max_i V_i(t-1)a_{ij}b_j(x_t)$ and is the accumulated Viterbi cost of the most probable partial path in state j at time t (in negative log-likelihood form) and $min\ cost$ is the minimum cost across all paths at time step t (the path with the highest probability) across all N states given by $min\ cost = \min(V_j(0), ..., V_j(N-1))$. Like other pruning methods, threshold based pruning offers a trade-off between computational efficiency and the accuracy of the decoded sequence. By adjusting the threshold value, one can control the degree of pruning: a higher threshold value will result in less pruning and potentially more accurate results, while a lower threshold value will result in more aggressive pruning and faster computation at the cost of potential loss of accuracy. However when running our experiments, given the limited training size, we decided it would be more beneficial to maintain a lower WER even if it was at the expense of a (potentially) lower Decode Time. As such we chose the threshold $(1e-41)$ that minimized the Decode Time while retaining our optimal WER ($\tilde{0}.48538$), we note if our lexicon was larger, such as thousands of words, we would likely choose to weigh the importance of Decode Time higher and thus sacrifice a low WER for the benefit of a reduced Decode Time as we would expect decode time to increase exponentially with added words. As this pruning strategy yielded better results, we chose to progress using only this method of pruning.

V. Task 4 - Advanced Methods

A. Further Optimisations

In order to further improve the efficiency of our decoder and experiments we used a combination of two different WFST operations: Determinisation and Minimisation. Determinisation transforms a non-deterministic WFST into a deterministic one, where for a given input label each arc has no more than a single output transition. This can merge equivalent paths in the WFST and can thus simplify the search space by reducing the number of arcs (and states) and thus improving decoding efficiency. Minimisation in turn transforms the transducer by finding equivalent states in the WFST and merging them, in essence transforming the transducer to an equivalent version of itself with the fewest possible states and transitions while retaining it's whole overall behaviour. If it is indeed possible to reduce (minimise) the transducer then this should improve decoding efficiency by again reducing the search space and reduce memory requirements. The results of incorporating Determinise alone as well as Determinise followed by Minimise are show on the following table:

Model	Decode Time	Avg. No. Forward Computations
No Det. or Min.	2.1989	107803.33
With Det.	1.7163	84453.85
With Det. & Min.	1.636	79966.553

We note a significant difference in Decode Time when implementing Determinisation giving a decrease of 21.95%. , and while Minimising cannot do too much without compromising equivalence it still provided an improvement (which with a larger dataset would be even more noticeable). Our final results combining every optimisation metric until now were:

- **WER (S,D,I) :** 0.48537 (2.05, 0.39, 1.23)
- **Avg. Decoding Time:** 1.6363349 with an average of 79966.553 forward counts
- **Average Backtrace Time**: 0.000237
- **Memory**: 192 arcs and 92 states

B. Bigram model

Hypothesis: The Bigram will produce really good results in terms of WER, however not better than the unigram silent model, which we believe to be a more accurate representation of human speech. This model will also increase substantially in memory, and therefore in forward computations and decode time.

Procedure: The first step to build the bigram model is calculating the bigram probabilities. For the unigram model the probabilities were calculated based on the passage: *'Peter Piper picked a peck of pickled peppers. Where's the peck of pickled peppers Peter Piper picked?'*. Now, however, we can no longer rely on the passage to calculate the bigram probabilities as there were many transcripts that had word combinations that could not be seen in the passage. For example, one transcript is: *'where's the peppers'*, in this example the bigram *'the peppers'* would have a probability of 0 if we calculate the probabilities using the passage - because there are no counts of the consecutive words *'the peppers'* appearing.

Therefore, for this task we calculated the bigram probabilities based on the transcripts of the full audios set, with the following formula:

$$P(\text{word}_2|\ \text{word}_1) = \frac{\text{Count}(\text{word}_1, \text{word}_2)}{\text{Count}(\text{word}_1)}$$

Additionally, if we want to include silent states in the model we cannot calculate the probability of moving to the silent states by counting bigrams that include the space character (silent word). This would be the same as calculating the unigram probabilities, because after each word there is a space character. We will however address an alternative method to include silent states into our bigram model in a later section.

The structure of our WFST bigram model takes the same form as that of a unigram, but includes another set of the same words to which the first words are connect with bigram probabilities, and from which arcs are drawn to return to the first words with bigram probabilities.

To test our results and make them comparable to the baseline and the rest of models we used the same parameter values (0.5).

Results:

- **WER (S, D, I)**: 0.6322 (1.84, 0.31, 2.31)
- **Average Decoding Time**: 4.9611837 with an average of 287142.6 forward counts
- **Average Backtrace Time**: 0.000276
- **Memory**: 604 arcs and 231 states

Discussion: As expected, the decode time, memory and forward counts increased significantly. The obtained WER of 0.632199 is a good result, as it is an improvement from the unigram model, but we expected a lower WER. We believe this model has the same problem as the unigram, where there are no buffer states where the model can transverse to when there are silent breaks in the audios. Another possible improvement would be to tune the weights, which is the task we describe in the following section.

C. Bigram model Tuned

Hypothesis: We can continue to improve the bigram model by tuning the transition weights, as 0.5 for the transition probability is too high. In the unigram model, we observed that the transition probability giving the lowest WER is approximately 0.26. We would expect the bigram model to have a lower transition probability to transverse more through the self loops, in order to account for the absence of silent states.

Procedure: Following from the previous experiment we decided to tune the bigram model focusing on the transition probabilities. Using the same procedure as before, we calculated the WER of the model with different transition probabilities on the full audio set. Figure 7 shows our results:

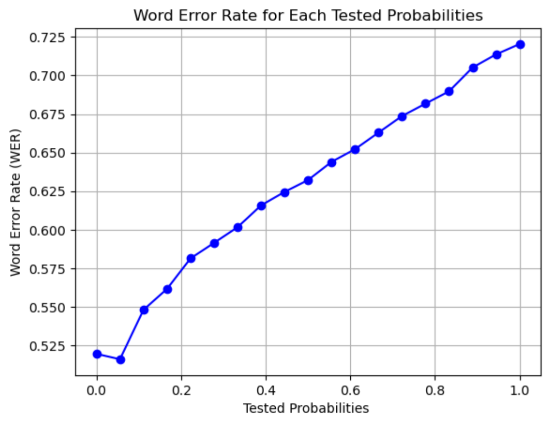

Fig. 7. Graph showing parameter tuning of the transition probability to optimise WER in our Bigram Model

We observed a clear trend in the bigram model, where lower transition probabilities (or higher costs) produce the best results in our training data. This drop in the transition probabilities leads to a better performing model, compared to the unigram silent model (without tuning). In the previous best tuned model we had a transition probability of 0.2625, with a self loop probability of 0.5. This time with the bigram model, with the same self loop probability, our transition probability is 0.0556, which is much lower. This means that, in order to tune this model, we must take a self loop arc ̃10 times more often than a transition arc. This difference between the unigram silent and bigram model, can be explained by the absence of silent states. Since the bigram model does not have silent states it has no buffer states in which to stay when there is a 'silent break', so instead it relies on self-loops and remains in the same state.

Results: The following results were obtained with a transition probability of **0.0556444**, this was the tested transition probability that yielded the best Average WER for our Bigram Model.

- **WER (S, D, I)**: 0.5162 (1.80, 0.53, 1.43)
- **Average Decoding Time**: 5.9279714 with an average of 287142.6 forward counts
- **Average Backtrace Time**: 0.000272
- **Memory**: 604 arcs and 231 states

Discussion: As we explained in our hypothesis the ideal transition probability had to be lower than those seen previously, as the absence of silent states must be accounted for with a higher self loop probability - or lower transition probability in this case. The WER decreased slightly and produced a competitive result compared to the previously seen. We saw no change in the time and memory metrics, as the model is exactly the same.

D. Bigram model with Silent States + Tuning

Hypothesis: We expect the bigram model to improve in WER by adding silent states, as we are now providing the model with buffer states to stay in when there are silent breaks in the recordings. The addition of silent states comes with the cost of having larger memory, and therefore higher decode time.

Procedure: As explained previously, there are ways in which to add silent states to a bigram model. We propose to add them by forcing each word to pass through a number of silent states. This would involve adding the silent states structure described in 2 at the end of every word - this way the final state of every word has an arc to the silent state structure. Each word is attached to its own silent structure, as otherwise it could lose its bigram history.

Using the silent states structure described in Fig. 2 may not be ideal, given that we are forcing the same structure to each word. Therefore we decided to experiment with different numbers of silent states to see if this could help. The WER and Decode Time were considered as the key metrics to evaluate the best model. In this case, Decode Time will be extremely relevant, as we will be increasing the memory by adding silent states, and models with more states require increasingly many forward computations. Fig. 8 shows the different WERs and Decode Times achieved by different numbers of silent states.

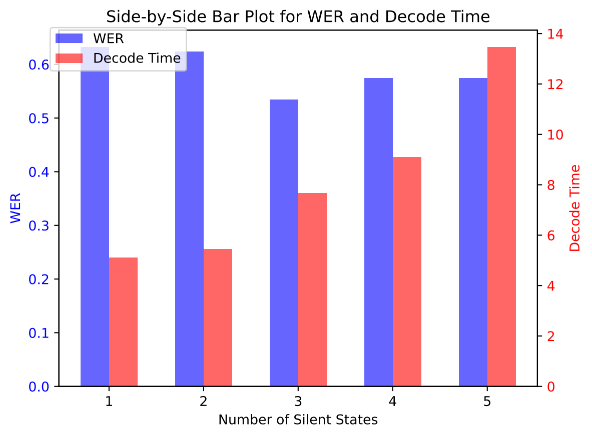

Fig. 8. Graph showing the WER and Decode time of different tested Bigram Models with different number of Silent States

As we expected, models with a higher number of silent states have a larger memory (states and arcs) and so have a (what looks to be) exponentially growing larger Decode Time for the Viterbi algorithm. We observed that having 3 silent states produced the lowest WER 0.533905, with a 2.2 seconds of decode time higher than the model with 2 silent states. We chose to go forward with the 3-silent state bigram model (in part due to Decode Time being a limited constraint due to our small dataset) and tune the transition probabilities as we did before in other tasks.

Results: After tuning the 3-silent state bigram model, we found that the best transition probability was 0.667.

- **WER (S, D, I)**: 0.5309 (1.65, 1.27, 0.99)
- **Average Decoding Time**: 6.8883257 with an average of 384157.6 forward counts
- **Average Backtrace Time**: 0.000311
- **Memory**: 814 arcs and 371 states

Discussion: This resulted in the highest transition probability we have had for a tuned model before. We can explain this increase in the desired transition probability by the increase in the number of states per word. This is the model with the largest number of states, 371 - caused by the addition of a silent structure at the end of every word in the WFST. To go through all the necessary states, given the recording features, the model has to go through more transition arcs than it has gone before with other models. Therefore, the transition probability must be increased in order to see an improvement in accuracy. We saw that the bigram model did not improve by adding silent states at the end of every word, contrary to what we expected. The tuned model required a higher transition probability, this possibly made the model produce more words as the arcs that returned the words were reached faster (were not delayed by self loops) - we suspect this because the deletion rate increased.

E. Bigram model with Back-Off

Hypothesis: By including back-off into the bigram model we expect a similar performance to the bigram model, possibly starting worse in terms of WER, without sacrificing too much decode time. There won't be a significant increase in decode time, because the memory is only slightly increased and therefore there will only be a small increase in forward computations.

Procedure: In this model, the words are also connected to a back-off state where their history is *forgotten*. This means that after reaching the back-off state, the arcs leaving from it can go to any of the words, but with the unigram probability of that word (rather than the bigram probability). Through this implementation we allow the model to forget the previous word history in certain paths.

Results:

- **WER (S, D, I)**: 0.4994 (1.53, 1.55, 0.65)
- **Average Decoding Time**: 5.0014453 with an average of 306810.0 forward counts
- **Average Backtrace Time**: 0.000253
- **Memory**: 644 arcs and 232 states

Discussion: We did not expect back-off to have a positive impact on the WER reducing it from 0.5162 (tuned bigram) to 0.4994. This could be caused by the possibility to *forget* history in certain word sequences, which allows for the model to turn to unigram probabilities when bigram probabilities are less favourable.

F. Bigram model with Silent States and Back-Off

Finally, as our last contribution in this final section we decided to attempt joining the silent bigram model with a back-off state.

Hypothesis: Given the previous effect of adding silent states to the bigram model, we do not expect this combination to improve the performance of bigram with back-off, however we do believe that it will improve that of the bigram with silent. We can expect this model to have the highest decode time, and the largest number of forward computations, due to its increase in states and arcs.

Procedure: Taking the the structure of our Bigram model with silent states, we will add the back-off state in the same manner as we did to produce the bigram model with back-off. This will, again, make the back-off state receive arcs from all words in the bigram model, and have outgoing arcs leaving from it with unigram probabilities.

Results:

- **WER (S, D, I)**: 0.5623 (1.30, 2.79, 0.27)
- **Average Decoding Time**: 7.528830 with an average of 306810.0 forward counts
- **Average Backtrace Time**: 0.000307
- **Memory**: 854 arcs and 372 states

Discussion: The WER has improved compared to the bigram model, but it does not improve the performance compared to the bigram silent, or bigram with back-off. As expected the decode time increased substantially due to the addition of the back-off arcs and silent states.

VI. Final Review

We attempted to improve the baseline model by taking into account different metrics, mainly: WER, Decode time, and memory. We proposed small changes to the baseline module that significantly improved its performance. Adding unigram probabilities to move from the start state to the starting phone of each word improved the WER by 0.03 points, without sacrificing any running time or memory. We then included silent states, which allowed the model to have buffer states in which to stay when there was silence in the audios. This improved the WER by a large margin - approximately 0.1 compared to the baseline. This however, increased the decode time slightly by 0.2 seconds.

The combination of these two previous features produced the best model after being tuned. Its WER performance is of 0.4854, decode time of only 2.42 seconds, and a memory holding 245 arcs and 121 states. Through tuning we saw the importance of having appropriate weights for self loop and transtioning, as it is their proportion that dictates how the model is traversed. Our top performing model was enhanced by adding determinzation and minimization which significantly lowered the decoding time, without sacrificing any WER. We also tested this model on two different pruning techniques which were able to lower the decoding time substantially but found one of them sacrificed too much of the WER accuracy.

Finally, in the last task we attempted to surpass this model by building more complex and advanced ones. We constructed several bigram models, some including silent states and back-off and also their combinations. We found these models to be nearly as powerful as that of the unigram with silent states, but were not able to compete with the unigram silent decoding time and memory - which were much lower. A full compact list of results can be viewed in the Appendix A.

Appendix

Model	Transition Probabilities	WER (S, D, I)	Avg. Decode Time	Avg. Forward Computations	Backtrace Time	Memory (Arcs, States)
Baseline	0.5	0.7328 (2.20, 0.25, 2.76)	2.1925789	111’970.2	2.48e-4	(230, 116)
Unigram	0.5	0.7043 (2.16, 0.29, 2.58)	2.2928156	111’970.2	2.40e-4	(230, 116)
Silent	0.5	0.5366 (2.14, 0.28, 1.57)	2.4041704	119’343.8	2.25e-4	(245, 121)
Unigram + Silent (US)	0.2625	0.5139 (2.08, 0.33, 1.43)	2.362049	119’343.8	2.58e-4	(245, 121)
US Tuned (UST)	0.2625	0.4854 (2.05, 0.39, 1.23)	2.420398	119’343.8	2.54e-4	(245, 121)
UST Pruned (USTP)	0.2625	0.4854 (2.05, 0.39, 1.23)	2.198931	107’803.3	2.25e-4	(245, 121)
USTP (with Det.)	0.2625	0.4854 (2.05, 0.39, 1.23)	1.7163247	84’453.8	2.11e-4	(201, 95)
USTP (with Det .& Min.)	0.2625	0.4857 (2.05, 0.39, 1.23)	1.6363349	79’966.6	2.37e-4	(192, 92)
Bigram	0.5	0.6322 (1.84, 0.31, 2.31)	4.9611837	287’142.6	2.76e-4	(604, 231)
Bigram Tuned	0.055644	0.5162 (1.80, 0.53, 1.43)	4.9279714	287’142.6	2.72e-4	(604, 231)
Bigram Silent	0.5	0.5339 (1.68, 1.27, 0.99)	7.3873396	384’157.6	3.12e-4	(814, 371)
Bigram Silent Tuned	0.66633	0.5309 (1.65, 1.27, 0.99)	6.8883257	384’157.6	3.11e-4	(814, 371)
Bigram Backoff	0.5	0.4994 (1.53, 1.55, 0.65)	5.0014453	306’810.0	2.53e-4	(644, 232)
Bigram Silent Backoff	0.5	0.5623 (1.30, 2.79, 0.27)	7.5288330	403’968.0	3.07e-4	(854, 372)

TABLE I
TABLE SHOWING THE FULL COLLECTION OF OUR RESULTS - WITH WORD ERROR RATE (WER) FOLLOWED BY (SUBSTITUTIONS, DELETIONS, INSERTIONS). AVG. DECODE TIME AND BACKTRACE TIME ARE GIVEN IN SECONDS. IT MUST BE NOTED THAT THE SELF LOOP PROBABILITY AND FINAL PROBABILITY WERE KEPT AT 0.5 FOR THESE EXPERIMENTS, AS EXPLAINED IN TASK II

REFERENCES

[1] G. D. Forney, “The viterbi algorithm,” *Proceedings of the IEEE*, vol. 61, no. 3, pp. 268–278, 1973.

[2] C. Tillmann and H. Ney, “Word reordering and a dynamic programming beam search algorithm for statistical machine translation,” *Computational linguistics*, vol. 29, no. 1, pp. 97–133, 2003.