Automatic Speech Recognition 2025-26: Assignment

Peter Bell

11 February 2026

1 Introduction

In this assignment you will carry out speech recognition experiments in Python using the WFST structures and Viterbi decoder that you will have written in the Labs. You will experiment with methods to improve speech recognition accuracy and to make the recognition process more computationally efficient. You will work with a collection of recordings made by students on this year's course, and the accompanying transcriptions.

You will need to submit a report summarising your experiments and findings, together with the Python code you wrote to carry out your experiments. Feel free to base your code on solutions provided for the lab exercises; however, if you are happy with your own solutions you may find it easier to extend your own code instead.

Working in pairs

You can work in pairs for all parts of this assignments: by working with another student, you can discuss ideas and solve coding problems together, as well as collaborating on the writing. You need only submit one set of code and one report per pair.

You may discuss any aspects of the assignment with your partner and divide up the tasks however you wish; but we encourage you to collaborate on each part rather than doing a strict division of tasks, as this will enable better learning for both of you. You may also discuss high-level concepts and general programming questions with others in the class; however you may NOT share code, designs or the results of experiments directly with other groups, and reports should not be shared at all.

Please remember the good scholarly practice requirements of the University regarding work for credit. You can find guidance at the School page https://informatics.ed.ac.uk/taught-students/all-students/your-studies/academic-misconduct This also has links to the relevant University pages.

Note that you are required to take reasonable measures to protect your assessed work from unauthorised access. For example, if you put any such work on a public repository then you must set access permissions appropriately (permitting access only to yourself and your partner).

2 Coursework submission

The submission deadline is Wednesday, 25 March at 12:00. Your coursework submission is complete only if you or your partner submit both your report and Python code. The late policy for the assignment are specified on Learn in the "Coursework Planner". Guidance on late submissions is at https://informatics.ed.ac.uk/taught-students/all-students/your-studies/late-coursework-extension-requests. Please get in touch with the Informatics Teaching Office well ahead of the deadline if you have any uncertainty about the late submission rules.

Your report should be either a PDF (.pdf) or MS Word (.doc(x)) document. Code can be in the form of plain Python (.py) files, or as a Jupyter notebook (.ipynb).

We encourage you to submit your code in a single file, with inline comments to indicate which parts were used to run particular experiments. If you have based your work on the published solutions to the labs, you do not need to explicitly mark this in your code. You need not submit any code files supplied as part of the labs or assignment that have not been modified by you or your partner.

Your report and code should clearly contain both yours and your partner's exam numbers. Please do NOT include your names in any files.

You will submit your files using the GradeScope tool on Learn.

3 Assignment specifications

3.1 Data and resources

You will experiment with on a small collection of recordings from ASR students on this year's course. Recordings comprise short utterances with words randomly selected from "Peter Piper picked a peck of pickled peppers. Where's the peck of pickled peppers Peter Piper picked?" (You can ignore all casing and punctuation).

If you choose to contribute your recordings to a communal pool, you will be able to access these, and the accompanying transcriptions on DICE at `/group/teaching/asr/labs/recordings`. Please do not copy the files from this location. We plan for the collection of recordings to be finalised by Mon 23 February.

For this assignment, we have provided:

- a pronunciation dictionary, `lexicon.txt`, and phone list, `phonelist.txt`, for all words in the vocabulary;
- a pre-trained neural network used to compute pseudo-likelihoods $p(x_t|q_t = j)$ for each frame of a recording, where the states j are labelled with strings corresponding to monophone states (eg. "ax_2" is the second state of the "ax" phone;
- a module, `observation_model.py`, to simplify the computation of the observation probabilities. Note that when this module is first imported, the neural network is loaded into memory. When the `load_audio()` function is called, all observation probabilities for all frames of the utterances are pre-computed and stored in memory;
- a module, `wer.py`, to compute counts of the number of insertions, deletions and substitutions between ASR output and a reference transcription;
- a Jupyter notebook, `assignment.ipynb` that gives examples of the provided code and sets up a basic template you could use

Most of these files have been provided for earlier labs. All can be found in the repository https://github.com/yiwang454/asr_assignment Note that you do not need to work in the notebook for the assignment if you prefer to use plain Python.

3.2 Evaluation and analysis

In your experimental work and report, you should evaluate the performance of your ASR systems in a number of ways.

1. Measure the *accuracy* of the output using the standard Word Error Rate metric. This is computed as
$$\text{WER} = \frac{S + D + I}{N}$$
where S, D and I are counts of the total number of substitutions, deletions, insertions respectively, and N is the number of words in the references transcription. You should compute a single score over all utterances by summing the counts per utterance.

2. Estimate the *speed* of your Viterbi decoder by measuring the time taken to run the `decode()` and `backtrace()` functions (we are not prescriptive about the way you do this, provided your method gives consistent results).

   You should also count the number of *forward computations* required to perform the decoding: that is, every time you need to compute the likelihood along an arc in your WFST. This will be closely correlated with decoding speed.

3. Compute a measure of the *memory* required for your decoder by providing counts of number of states and arcs in the WFST.

Note: in this assignment, the actual memory usage and decoding times will by design be very low due to the very small vocabulary. The aim of the analysis is to get you to think about the issues of ASR system design when the vocabulary becomes much larger.

3.3 Tasks

Task 1 – Initial systems [20 marks]

Perform baseline experiments on the collection of recordings using your Viterbi decoder from the Labs along with a WFST designed to recognise any sequence of words from the vocabulary[1]. Analyse your experiments according to WER and the other measures described in 3.2, and briefly comment on your findings.

If you have doubts about your system performance at this stage, you are able to discuss your results with the course teaching staff and may also post your initial WER scores on Piazza to sanity-check them against other students' figures.

Task 2 – System tuning [20 marks]

It is likely that your WER scores from Task 1 are very high – find out why this might be. Can you adjust parameters of your WFST to improve your results? You might experiment with varying the self-loop probabilities, final probabilities, and using transitions to each word based on unigram word probabilities. Describe what you observe.

Investigate the effect of adding an optional silence state between words in your WFST to handle the case where there are pauses between words in the recording. The provided observation model can compute the observation probability for silence using the label "sil_N" for $N = (1, 2, \ldots 5)$. (This has been trained using a five-state left-to-right topology where states 1 and 5 are initial and final states respectively, and states 2, 3 and 4 have an ergodic structure. *You do not need to replicate this topology exactly*).

Task 3 – Pruning [20 marks]

Implement a form of pruning in your Viterbi algorithm to avoid propagating computations along unlikely paths. Experiment with different pruning thesholds, and, if you like, different pruning strategies. Discuss the effect on the accuracy and speed of your decoder.

[1] If your Viterbi decoder from the labs outputs phonemes instead of words, you might find it easier to modify your HMM structures to directly output words – or you could use WFST composition instead.

Task 4 – Advanced topics [40 marks]

Investigate methods to further enhance your system. You should experiment with:

- **Improving the efficency of the decoder**, using for example a tree-structured lexicon, optionally with language model look-ahead. You might alternatively opt to further improve your implementation of the Viterbi algorithm.
- **Improving the grammar**, using an n-gram language model, or any other grammar you might think is more appropriate for the task.

In both tasks, feel free to implement your approach directly in your own WFST creation code, or take advantage of WFST operations such as composition and determinisation. Lab 5 is likely to be useful for this. You can combine elements of both options if you prefer, but this is not necessary for full marks.

When designing a grammar, you might consider estimating language model parameters from the collection of recordings (though this is not the only possible approach to the task). If there are enough recordings, you could do this by splitting the data and performing a cross validation approach. However, you will not be penalised for estimating parameters on the test set directly, if you acknowledge possible limitations of this approach in your report.

3.4 Report

Keep the length of your report between 4 and 8 pages including figures, tables, and references. If possible, a two-column format is preferred. Results of experiments should be well summarised using figures or tables. You should not only show the results, but also explain your experiments and findings, and give discussions. Marking will be based on the contents and quality of experiments, presentation, and discussions. The functionality of your code will be assessed, but *your coding style will not.*

Please read the following instructions and write a "scientific report".

- Results of experiments should be efficiently summarised using figures or tables (but not both if possible) - avoid using a separate graph/table for each experiment.
- Figures/tables should be numbered and captioned.
- Conditions of experiments and methods that are different from your baseline system should be shown clearly and concisely - consider using a table for example.
- Show results even if there was no improvement. It is important to discuss/analyse why there was no improvement.