Future Directions

Jason L Hutchens
Centre for Intelligent Information Processing Systems
Department of Electrical & Electronic Engineering
The University of Western Australia
Nedlands WA, AUSTRALIA 6907

April 28, 1997

Abstract

It seems that whenever I seem to be making good progress in my research, I decide to move the target. This is partially due to the fact that experimental results change my perceptions of what I should be doing. It's also because people point out that my work is similar to what's being done in other fields; I read work in these fields and then apply what I've read to my project.

In this brief paper, I explain what I am currently doing, and where I want to go with it. In doing so, I state all of my assumptions, and what I instinctively feel my results will show. Additionally, I argue why the way I've chosen to do things is the Best Way.

1 Introduction

I've currently been working on GIMP. That stands for "The Grammatical Inference Modelling Package", and it's supposed to be a collection of routines which can be controlled through a simple interface, to make testing and

experimentation easier (rather than writing code to run an experiment, you just tell the program what it is you want to do).

At the moment, GIMP is capable of building n-gram language models using an alphabet of arbitrary symbols. It can evaluate the models using standard techniques, it can make predictions, and it can chunk.

I began to evaluate different chunking strategies, notably Mike's method and my method, and that's where I first ran into problems. In the next section I will describe Mike's chunking strategy and what its problems are; after that will describe how I intend to modify my system to avoid these problems.

2 Finding Chunks

To find chunks in arbitrary data we train our model using some of the data, and then we use this model to make predictions about the rest of the data. If our predictions aren't very good, their entropy will be high, and we conclude that we've stumbled upon a "chunk boundary". At the end of this process, we can add all the chunks we've found to the dictionary as new symbols. We then repeat until some stopping criteria is reached.

Here begin our problems: a threshold entropy is needed to find the chunk boundaries, but there's no way we can know what this threshold should be *a priori*; we may find zillions of chunks, but not all of them should be added to the dictionary, so we need some way of deciding which ones to add; to evaluate whether adding the chunks was a good idea or not requires two more passes over the corpora (one to train with the new chunks, one to evaluate the model).

To overcome this problem, I've toyed with the idea of making the system completely adaptive. I describe how I propose to do this is the following sections.

3 Adaptive Models

It would be really nice if our language modelling system was adaptive. Rather than making several passes over a corpus (to infer a model, evaluate the model, find new structure and discard bad generalisations), we would make a single pass over the corpus, performing all of these steps at every stage.

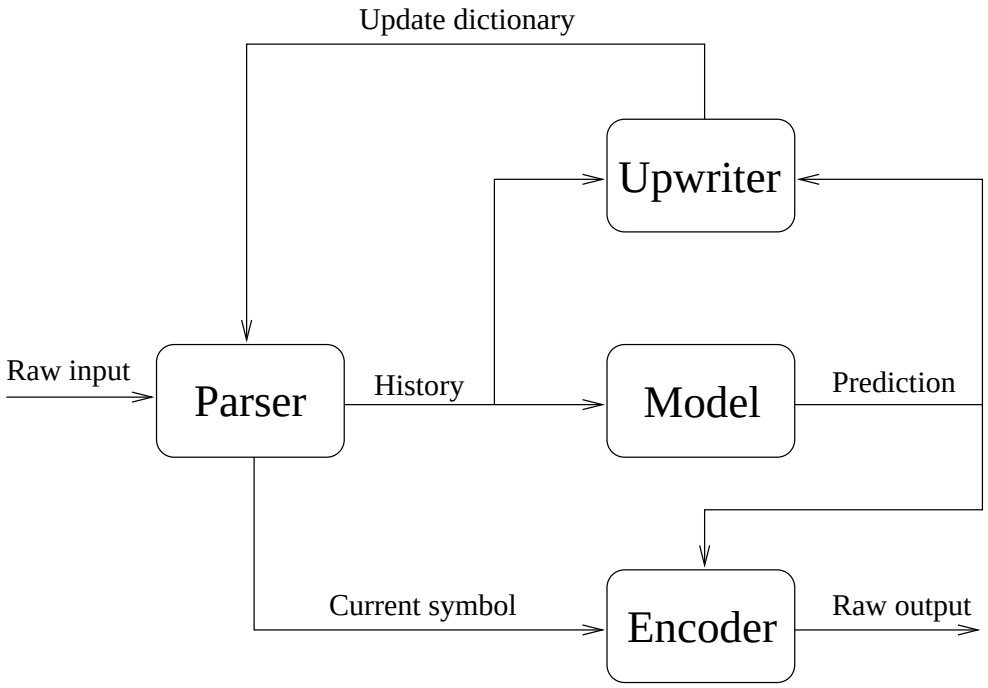

Figure 1: Adaptive language model

There are several good reasons for believing that this is the way to go: it allows us to use the model as a compression system, which has the added benefit of making model evaluation more meaningful; it makes the chunking process more akin to "real" learning as I will soon describe; it forces us to make minimal assumptions about the type of data we are likely to see.

Figure 1 is a schematic diagram of how I believe an adaptive language model should work. No assumptions are made about the data source, other than it produces a one-dimensional array of symbols. We must make assumptions

about these symbols, as the parser needs something to work with. Therefore, the dictionary must be initialised with the lowest-level symbols, which will most likely be bits or bytes. The model can be anything which makes a prediction about the next symbol, given the history. The encoder uses this prediction, along with the symbol which actually occurred, to compress the symbol stream. Finally, the upwriter uses predictions to find structure in the data, and it adds this structure to the dictionary. All of the components perform their tasks at every stage, which means we must develop efficient methods of doing things.

At the moment, we will consider the upwriter to be nothing more than a chunker. Traditional methods of doing chunking require multiple passes over a corpus, which isn't possible in an adaptive system. We therefore need to think about adaptive chunking.

3.1 Adaptive Chunking

I have thought about this problem, and think I have come up with a satisfactory solution. Rather than finding chunks by making passes over a corpus, we find chunks directly from the model. This is equivalent to the case where we chunk by making passes over a text, but discard all the chunks which are not $n-1$ symbols long (where n is the order of our model).

To put things simply, we chunk by looking at all of the contexts in our model , where each context is a string of $n-1$ symbols. This string may be chunked and added to the dictionary as a new symbol if it satisfies certain criteria. The entropy is a natural criteria to use, so if the entropy of the prediction we get if we use the context as the history is high enough, we chunk the context.

We can even attempt to think about why this technique is reasonable. If a context has a high entropy, that means the predictor is uncertain about what is coming next. There are two explanations for this uncertainty: the context isn't long enough, or the next symbol is independent of the history. We

assume that the first explanation is the correct one, and we add the context to the dictionary as a new symbol, so that the next time it is seen we will have a slightly longer context to make a prediction from.

This is analogous to using scaffolding to build a skyscraper—our "real" chunks are built up from smaller "chunkettes", which are used to learn higher-level structure but which become redundant when this higher-level structure is found. They will still exist in our model and in our dictionary, but they just won't be used very often. They effectively become "fossils".

I believe that this chunking method will find the same chunks as a static technique, because our chunking criteria is the same. The main advantage is that it's tremendously fast, which means we can look for chunks whenever we add a symbol to the model. This is because we can keep a sorted record of the entropy of each context, so we only need to investigate n possible contexts on each iteration (that is, the n contexts affected by the new symbol which we've just added to the model). So we would add a maximum of n chunks to the dictionary at each iteration. In fact, it may be sufficient to add only the most promising chunk, and then only if it's entropy exceeded some threshold.

Why is this method better than Mike's offline method? Because we don't need an entropy threshold, we have an intuitive way of limiting the number of chunks added to the dictionary, we train and chunk in parallel, evaluation is only done once (and on the fly at that), and we know what the stopping criteria is (it's when we reach the end of the corpus)!

4 Model Evaluation

Traditionally statistical language models have been evaluated by measuring their perplexity, which is a monotone function of their average information. Perplexity is better, though, because small improvements in the average information give large improvements in the perplexity, which means that you can impress the plebs a whole lot more.

The thing is, the average information is very nearly a measure of how good the predictor would be as a compressor. It's close, but not quite there, because it takes no account of how big the model is. If we really wanted to use the model as a compressor, we would have to transmit the model along with the compressed data, because traditional techniques are static (that is, the model is trained on one corpus, and evaluated on another).

In accordance with Occam's razor, we would like a measure which gives a tradeoff between really good but horrendously complicated models and not so good but simple models. Adaptive data compression gives us such a measure naturally. And because we are really truly compressing the data, we know our performance measure is valid (because we can uncompress the data and make sure it works), and we can compare our model with other models which may not even be statistical or adaptive.

5 Conclusion

In conclusion, I propose that my next major step in my project be adaptive language modelling. My adaptive chunking technique is similar to Mike's non-adaptive method, but it's a lot faster. Also, if we do things adaptively, we may come closer to modelling what's going on in biological brains. We may therefore end up developing a general learning system for one-dimensional discrete data rather than just a language modelling tool. That would be nice.