Data Adaptive Dialog Systems

by
David Attwater
Alexandra Auckland
Jonathan Bloom
Brian Budd
Lizanne Kaiser
Peter Krogh
Dan O'Sullivan
Mark Stallings
David Suendermann
Jason Williams

Table of Contents

Introduction

Most production dialog systems are hand-written to achieve a specific goal. Once the system is deployed, a designer or speech engineer usually makes improvements by tuning the grammars, prompts, and call-flows using data such as utterance recordings, whole-call recordings, usage logs, etc. This style of "human-in-the-loop" tuning is widespread, and it is generally accepted to help build better systems. However, the scope of these data-driving improvements is limited by the cost of the work involved. Further, once the system is deployed, it is entirely static – despite interacting with millions of callers a day, most dialog systems make no improvements on their own.

Techniques are emerging which will enable a dialog system to learn or adapt its behavior automatically, with minimal or no human intervention. This paper explores various data-adaptive techniques and discusses the pros and cons for their use in production dialog systems.

Data Adapted Grammars

The tuning and adaption of grammars using data gathered in the field is an established example of a data-adaptive solution. A dialog system is deployed to users, and speech data is then collected and transcribed. Grammar performance is established against real speech data, then optimized also with data. This is a tool-assisted process performed by a speech scientist and is a good example of supervised goal-driven optimization using data. The goals, chosen by the scientist, may include minimizing false rejections, minimizing substitution errors, or maximizing grammar coverage. It will usually be a combination of such goals. The field data is used to empirically discover more optimal conditions, and then the improved solution is returned to the field [1].

Hand-written grammars can be further extended using data adaptive methods. Following an initial grammar design, transcribed data can then be used to transform the grammar into a statistical grammar. Such grammars are trained using data rather than hand-written rules. They can recognize phrases that had not been anticipated by a designer. During transcription of data, semantic tags (meanings) are added to the transcriptions to represent the intent of the utterance, a process called *annotation*. The relationship between language and intent is then learned from the data, further extending the ability of the system to generalize when a previously unseen utterance is presented. New intentions may also be discovered through this process, and new dialog transitions or alternate dialog strategies may be added to handle them. By using data to discover these new transitions, those that are naturally more frequent will emerge before those that are less frequent. This will lead to the adaptation of the system towards actual caller behaviors rather than just those anticipated by a designer.

By way of example consider the following dialogue fragment:

> *System: Say the reason for your call.*
>
> *Caller: I'm calling to pay my bill.*
>
> *System: You want to get a copy of your bill?*

Caller: No, I want to pay my bill.

Such mixed-initiative behavior can be and should be anticipated by a designer, but the designer is unlikely to be able to anticipate all the language used and the frequency with which certain misunderstandings occur. The initial confirmation dialog will likely contain yes/no grammars and strategies to deal with responses other than yes or no. These initial strategies will be effective, but they may be less efficient than a data adapted solution that deals with the mixed initiative response directly.

The use of data adaptive approaches can discover dominant cases where adaptation serves the caller better. At this point the yes/no grammar that was formerly shared with other contexts will gain a life of its own as it is adapted to its specific context.

Here's how we might go about performing this adaptation:

1. Determine whether there are enough transcribed utterances to reliably train a new context.
2. Train a specific grammar to this context.
3. Identify whether there are new dialog transitions opened up by this new context.

In addition to improving recognition grammars, this tuning cycle is also commonly used to set confidence score rejection thresholds, end-pointing sensitivities, and time-out settings. For example, the speech engineer might decide that no more than 3% of utterances may be falsely accepted and 10% falsely rejected, and set the confidence score rejection threshold accordingly. Automatic metrics can also be used – for example, maximizing the fraction of cases where the recognizer does the "correct" thing: correctly accepting in-grammar speech and correctly rejecting out-of-grammar speech.

A key question regarding the data-driven grammar training and tuning approach is – what are the costs of transcription and annotation? and - will the newly adapted context give enough benefit to make it worth it? New tools and methods which allow transcription and annotation to be generated semi-automatically suggest that up to 1 million utterances per month per person can be produced meeting high quality standards. The resulting grammars, specific to their context, show significantly improved accuracy beyond the normal tuning cycle.

Data Adapted Prompts

During the design process, designers often agonize over the best prompt phrasing, intonation, or pausing. These details have a tremendous impact on success, and although some guidelines do exist, the best choice for each dialog situation is rarely obvious. For example:

- If there is a history of noisy interruptions in this call, we might want to ask the caller whether they are on a hands-free phone or in a noisy environment. Should this be done after the first, second, or a later speech error? If the caller has a history of speech errors, should this be stated at the beginning of the call? When should barge-in be switched off (if at all)?
- If the user has trouble with the system, often the prompts are streamlined and tapered. Also an operator might be offered. What are the right elements of the tapered prompts? When should an operator be offered?

- It is often thought to be desirable to adjust the language of a prompt to match the language spoken by the caller. Does this improve task completion?

One obvious approach is to deploy a small collection of promising alternatives to users, and listen to calls to see what works best. While this is sometimes done, these studies are rare because they are expensive and time-consuming. Moreover, prompts are *inter-related*: prompt A in dialog state 1 may work well when used in conjunction with prompt X in dialog state 10, but not when used with prompt Y in dialog state 10. To test different prompts in different dialog states, it is often necessary to try *all combinations* of all prompts. This rapidly becomes too expensive for evaluation with a human in the loop.

A potential way forward is *data-adapted prompting.* For each dialog state, a designer specifies a set of prompt alternatives – perhaps different phrasings, pausing, intonations, speaking rate, etc. Each alternative is a reasonable choice, but the design team can't safely say which is the best. As calls arrive at a dialog state, prompts are chosen at random. Then each call is *automatically scored* based on a carefully chosen numeric goal – for example, reflecting successful task completion, number of turns, dialog duration, etc. Statistics are computed for the success of each prompt (and each combination of prompts) and the best prompts are selected. The approach is based on a well-founded body of mathematics called *reinforcement learning*, which is a parallel branch to the *supervised learning* used by the speech recognition engine. All of this has been tested in limited deployments, and consistently found to outperform hand-built systems [2] [3].

This approach opens the door to tailoring prompts to take into account much more dialog history. For example, automatic evaluation might find that it makes sense to use a more directed prompt if there was a speech error in the previous dialog state, but a shorter prompt if not. Specifying all these alternatives by hand is a daunting task as the number of different dialog histories is astronomical, and many of the choices would be arbitrary.

This approach also allows the system to *track changes in the user population – to* continuously adapt over the lifetime of the system. As the caller population evolves (perhaps as the mix of new and existing customer changes), some prompts which were initially discarded may become better alternatives. The system can be configured to occasionally re-try all of the prompts in its repertoire, to verify whether they might now be a better fit. At any point new prompts can be injected and evaluated – if they are better, they will be adopted; if not, they will be discarded.

Perhaps in the future, it would even be possible for a system to devise and test new prompt variants by itself (or suggest them to the designer for inclusion).

Unlike data-adapted grammars, data-adapted prompts have to be tested with real users in the loop. The accuracy of data-adapted grammars can be tested and verified off-line with a collection of representative utterances. This can be done because installing a new grammar doesn't substantively change user behavior. However, by design, changing prompts *does* change user behavior – and this can't be tested off-line.

Data Adapted Call Flows

Data-adapted prompting is limited to decisions within a dialog state – the transitions between dialog states are still fixed. The *data-adapted call-flow* approach generalizes this idea to the call flow: the designer specifies

multiple possible transitions; the dialog system explores these different transitions with callers to determine which dialog flow works the best.

For example, perhaps a dialog system needs to collect 3 pieces of information, but it is not clear which should be collected first. Each ordering could be included as a possible transition, and the best ordering can then be learned from data.

These alternative transitions allow many variations of a call flow to be explored. For example, a designer could suggest different ways of branching between dialog states based on features from:

- Speech recognition: confidence scores, number of words spoken, pauses, etc.
- History from the dialog: how many speech errors have been encountered, length of the dialog to the present moment, number of states re-visited
- External knowledge about the user: recent calls, activities during those calls, interactions through other channels such as a website
- World knowledge: service outages, mergers, weather-related events

It is often difficult to know precisely how to use all of this information; the data-adaptive call flow approach tackles this by allowing the designer to suggest several reasonable methods for branching, and then the learning finds which one is optimal. An attractive property of this process is that the different choices for prompts and branches are explored *in conjunction.* In other words, the learning algorithm does not attempt to find the best choices in isolation, but rather it seeks the *combination* of alternatives that performs best.

As an illustration, consider the following dialog design problem for a name-dialing system. This system asks for a name and then forwards the call – but doesn't have phone numbers for every name. In this setting, a high confidence might be suitable before forwarding a call ("Calling David Attwater, just a moment"); a medium confidence might be suitable when more information will be gathered before forwarding the call ("Brian Budd – cell phone or office phone?"); and a low confidence might be sufficient when a number is missing ("I thought you said Jason Williams, but I don't have a number for that name. Say a name, or feel free to hang up."). In each case there are different *whole-dialog costs* to misrecognitions – for example, in the first case, the cost of a false-accept is very high, but in the last case, there cost of a false-accept is very low – in fact there is an *additional cost* to confirming the name before saying there is no number available. (Work in the research community has found that the local confidence thresholds which optimizes overall task success are often not intuitive!)

So it seems likely that this call-flow could benefit from using different confidence score thresholds for different branches, but exactly how these parameters should be set is challenging for a designer to do *a priori.* Traditional off-line tuning – which sets one threshold for making accept/reject decisions for all recognition results based on a local measure – doesn't take into account different costs associated with different dialog branches. The data-adaptive call flow approach provides a way forward: different settings can be trialed and the combination of settings that maximizes the whole-dialog measure, like task success, can be retained.

Data Adapted User Models

Another way in which usage data can be used to automatically improve dialog systems is to learn probability models of users' behaviors and preferences. These models can help overcome some of the uncertainty introduced by speech recognition errors.

Recall that speech recognition outputs a list of hypotheses called an *N-Best list*, where the best guess is listed first, and alternatives are listed further down the list. In many difficult recognition tasks the correct answer is contained further down the list.

One way of using the information on the list is to consider all of the entries as hypotheses for what the user said, but to re-weight them according another source of information – a *user model*. For example, suppose a caller to a US-based airline says "Austin", but this is misrecognized as "Dresden" (Germany) with medium confidence, with "Austin" somewhere further down the N-Best list. In the absence of any other information, "Dresden" is the single best guess. However, if the system had access to a *user model* – in this case, a table of *prior probabilities* for each city – the alternatives on the N-Best list could be re-weighted. In other words, the recognition result and these prior probabilities are viewed *jointly* to determine that Austin is most likely when *both* are taken into account, even though Austin is not the top recognition result. This process can be extended to learn preferences over whole combinations of slot values, such as which destinations *starting from Austin,* and other aspects of user preference and behavior. In its most general form, this approach can be extended across many dialog states to synthesize information from the user model together with *all* of the N-Best lists observed in the entire dialog, making maximal use of all available information to improve whole-dialog accuracy [4].

In practice, when recognition confidence is high, these models are dominated by the recognition result and are effectively ignored. However, when recognition confidence is medium, these approaches can dramatically improve task completion.

Of course this whole approach relies on having a good *user model* in the form of a table of prior probabilities. These probabilities are readily estimated from usage data, such as which cities are ultimately ticketed in each call. At first a single model for all callers can be learned; over time, the models can be tailored to users from different regions (callers from New York are more likely to book flights starting in New York), or even tailored to particular users (Jason often flies to London to visit family but David often flies to New York for work).

So, Why Don't We Do All This Stuff?

The techniques outlined above have obvious merits. They all have novel aspects but are based on well understood principles known for a long time to those in the fields of speech and dialog research. Why is it that they are not widespread?

The answers are various but fall into three primary categories – cost benefit, design predictability, and tool support.

First, with increased complexity comes increased cost. This can be in the form of licenses, IVR port density, time to market, maintenance, professional services, or software development. The key benefits for many

commercial dialog systems are derived from just a small number of dominant paths. More complex or expensive solutions need to demonstrate incremental benefit over the simpler solutions.

Second, many commercial dialog systems sit on top of complex business processes that have real commercial risk if they fail. Stakeholders responsible for the business performance of a solution need to be able to verify the dialog system satisfies their business goals *before deployment to real users*. Two of the approaches above – data adaptive prompting and call-flows – increase the potential number of user experiences. Even though all possible paths can be theoretically anticipated in advance, there may be many more permutations than with a traditional system. There may in fact be too many variants for a single mind to weigh during the design and testing phase.

Tool support is thus closely tied to this increased scope of testing. Tools will be required to help design and test solutions. They need to provide core abstractions but at the same time be highly customizable to allow innovation. These tools will help the designer and tester anticipate all of the important sequences of events in the emergent solution. Automated verification tools could be an aid to this to ensure that content specified in the design is indeed in the implementation and has appropriate test coverage. To support all of these needs, design abstractions and well tested patterns will also become important for confident deployment of a data-adaptive solution.

The hallmarks of a successful data adaptive solution will therefore be:

- Gives clear incremental benefits over simpler solutions for the dominant interactions
- Has behavior that can be understood and tested by key stakeholders
- Has tool support which allows designers to anticipate and test interaction patterns

References

[1] D. Suendermann, J. Liscombe, K. Evanini, K. Dayanidhi, and R. Pieraccini, “From Rule-Based to Statistical Grammars: Continuous Improvement of Large-Scale Spoken Dialog Systems,” in Proc. of the ICASSP, Taipei, Taiwan, 2009.

[2] T. Paek and R. Pieraccini, “Automating Spoken Dialogue Management Design Using Machine Learning: An Industry Perspective,” *Speech Communication*, vol. 50, no. 8-9, 2008.

[3] D O'Sullivan, “Using an Adaptive Voice User Interface to Increase Caller Utilization in Automated Voice Systems.” White Paper. Interactive Digital. 2009.

[4] J. Williams, “Exploiting the ASR N-Best by Tracking Multiple Dialog State Hypotheses,” in Proc. of the Interspeech, Brisbane, Australia, 2008.