Fallable Foundations for Artifical Agents

The standard picture of an ideal agent—a perfectly rational actor with clear probabilistic beliefs and indisputable preferences [20, 17]—is useful and compelling. Despite its many opponents over the years, and well-documented conceptual shortcomings [18, 5], the simplicity and power of this formalism have lead it to a place of cultural dominance among computer scientists. Today, uncertainty *is* probability and values *are* utilities. Still, not every aspect of cognition is most clearly described in terms of expected utility maximization. Expected utility maximizers are inflexible and brittle. They maintain a fully formed probabilistic picture of the world at all times, and are slaves to their own unchanging preferences. Do we really want to build powerful artificial agents that aspire to this paragon of thought (alone)?

Fortunately, the AI systems we build in practice do not embody our unrealistic rationality standards. Discriminative models (e.g., classifiers and policies) contain only a partial probabilistic description of the world. We train them by upating paramters incrementally, moving on from a training example long before we classifiy it properly—yet a Bayesian upate serves to establish a proposition with absolute certianty. We always use the same handful of objectives to train models, even though every utility function is equally valid. We augment these objectives with regularizers to prevent over-optimization, in a nod to a kind of epistemic modesty that an expected utility maximizer does not have. We inject noise, which means sometimes making "sub-optimal" decisions, but also makes for a more robust system overall [19]. (For example, auto-regressive language models are much worse during inference when sampling is done at temperature zero, always selecting the most probable next word.) Is this enormous gap between what an agent is in theory and what we do in practice simply a result of hacky engineering choices? Or perhaps, in light of a better understanding of how artificial agents (and indeed human minds) work, we ought to revisit the mathematical underpinnings of what an agent is in the first place.

My research develops conceptually rich, expressive, and clean mathematical foundations for agents, that aim to be simultaneously more human and more useful for describing modern AI systems. **The result of this project has been a surprisingly elegant unifying picture of agency quite different from the standard one.** In it, agents can be decomposed into sub-agents. They can hold inconsistent beliefs, and experience rich internal conflict. They are driven not by maximizing some arbitrary utility function, but rather by the pursuit of consistency (although in some cases, the two are the same) [3]. And, in the same stroke, this picture validates and explain a wide breadth of "ad-hoc" choices that we regularly make in building practical artifical agents. We will now see some of the piecs of this picture in finer detail.

Probabilistic Dependency Graphs (PDGs). In some cases, you might be able to articulate the gold standard for a rational belief: a complete joint distribution μ over relevant outcomes. But in other cases, you may have only have only partial probabilistic information about how some variables depend on others (after all, an explicit representation of such a distribution is exponentially large), Bayesian Networks (BNs) [12, 7] are a way of converting a collection $\mathcal{P}$ of local probabilistic relationships into a full joint distribution, in the special case where each variable is the target of precisely precisely one conditional distribution and the dependencies are acyclic. This requires only that the modeler assume the conditional independencies that arise by interpreting $\mathcal{P}$ causally.

But what if you aren't prepared to commit to this causal picture? Moreover, what if your local beliefs do not form a complete, acyclic picture of the world? Such epistemic situations are very common in real life. Suppose two different scientist perform experiments on Y, but control for different variables. Let $p(Y|X)$ and $q(Y|Z)$ be conditoinal probability distributions representing the respetive outcomes of these two experiments. It is not possible to form a BN that includes both p and q. Another way of getting a joint distribution is to simply multiply all the distributions together, and then renormalize. This approach yields another (arguably more general) kind of graphical model, called a factor graph [8]. But, in general, factor graphs have some conceptual problems. If X and Z turn out to be highly correlated and $p = q$ are the same, then one might hope to recover a joint probability μ that satisfies $\mu(Y|X) = p = q$. But the approach of multiplying p and q together yields $\mu(Y|X) \propto p(Y|X)^2$. In general, the components of a factor graph cannot be given an interpretation in isolation; they only have meaning relative to one another.

Probabilsitic Dependency Graphs (PDGs) [15] are new class of graphical models I constructed, that address these shortcomings in other models. **PDGs allow a modeler to specify arbitrary (even conflicting) probabilistic and causal information, and with any degree of confidence.** Compared to BNs, PDGs are far more flexible and expressive. Compared to FGs, they are much more interpertable, and retain a probabilistic interpretation in all cases. In other words, PDGs are more modular than existing graphical models; their components can be recombined easily and without loss of meaning. What's more, PDGs actually generalize Bayesian Networks (BNs) and factor graphs (FGs). The key to making this work is a principled information-theoretic scoring function that socres candidate distributions μ based on how compatible they are with the information in a PDG. In more detail, PDGs are given semantics by two scoring functions—one that measures compatibility with the observational probabilities, and one that measures fit to the graphical structure. Both functions are sums of relative entropies, weighted by confidence.

The observational scoring function is straightforward, yet the structural one, although almost identical in structure, is a little bit harder to get one's head around, especially if one isn't eager to buy analogies to thermodynamics. However, in the last few months, we have discovered a concrete interpretation of it for a very broad class of structures, in terms of structural causal models. This has lead us to a novel definition of what it means for a distribution to reflect the structure of a set of independent causal mechanisms along a directed hypergraph. Of course, this definition extends the independencies of a Bayesian Network, but it also cleanly captures functional dependencies, and very interesting results for cyclic models. In particular, it significantly undermines a standard argument against multivariate Shannon entropy.

Inference for BNs and FGs is NP-hard, and clearly PDG inference is at least as difficult. The only subclass for which inference is tractible is those BNs whose structure is "tree-like" (i.e., graphs with bounded treewidth) [2], and in this case, inference can be done in linear time. For PDGs more general, though, it was not obvious how to do inference at all (let alone efficiently), even long after we had a clear understanding of PDGs. It turns out that for PDGs with bounded tree-width, inference can also be done in polynomial time ($O(n^{2.89})$) [16]. The first key insight was that, with some care, finding a distribution μ that minimizes the PDG scoring function can be reduced to speical kind of convex program that uses *exponential cone constraints*, which was shown to be require polynomial time by Nesterov [11]. Unfortunately, the construction yields an exponentially large optimization problem. To get a polynomial algorithm, we also needed a series of additionl insights that allow us to express that optimization problem much more compactly for PDGs of bounded treewidth, and a careful analysis of approximation guarantees.

When we were first developing the theory of PDGs, our motivaitons were actually not among the benefits we have outlined so far. For me, one motivation was to bring BNs in line with category theory, and indeed PDGs do admit a far nicer presentation in those terms. But far and away the most important motivation, for both of us, was the ability to capture inconsistent beliefs. We wanted to desribe the process of resolving inconsistencies, which demanded a way to talk about them. Although we did not realize this at the time, our way of doing this extended the usefulness of PDGs far beyond specifying a joint probability distribution.

Loss as the Inconsistency of a PDG. Internal conflict is an important aspect of human thought. Yet we computer scientists build artificial agents using probabilistic models that, by construction, cannot represent conflicted beliefs. Inconsistency is clearly undesirable, but we stand to gain a lot by being able to represent it. A PDG's *degree of inconsistency* is the minimum value of its scoring function, over all joint distributions μ. For example, a Bayesian network has inconsistency zero, because there exists a distribution μ which simultaneously matches all of its constituent probababilities and also its structure. But in general, beliefs can conflict with one another.

To to take a simple example, one could have a PDG consisting of two different probabilites $p(X)$ and $q(X)$ over the same variable, and no joint distribution μ that can simultaneously satisfy both constraints, unless $p = q$. Intuitively, if p and q are close to one another, it is not very inconsistent to simultaneously hold both beliefs, but it becomes more inconsistent as p and q get further apart. The precise numerical value of this inconsistency depends not just on p and q, but also in the respective respective confidences we have in them. It turns out that, for different choices of confidence, the resulting inconsistency measure generates a large array of standard statistical divergences, including the Renyi divergences, foward and reverse KL divergence, and Chernoff divergences. The result is a unified epsitemic interpretation of all of these standard statistical divergences [13]. It also gives us a deep insight into relationships between them: properties such monotonicity of Renyi divergences, the data processing inequality, and many others, can all be viewed as simple consequences of a single intuitive fact: "more beliefs and higher confidence cannot make you any less inconsistent".

All of this is just for PDGs containing two beliefs over the same variable. In general, PDG inconsistency is a vast generalization of these divergences to arbitrary structured objects [13]. Almost uncannily, the appropriate loss function in every standard learning setting arises simply by writing down the relevant parts of the picture as a PDG, and then measureing the resulting inconsistency. The inconsistency of a classifier and labeled data its log accuracy. If instead the classifier specifies the mean of a Gaussian, the inconsistency is mean squared error. This recovers a well-known connection between priors and regularizers. It even gives a clean interpretation of notoriously difficult-to-understand variational objectives: the inconsistency of simultaneously believing an encode $e(Z|X)$, a decoder $d(X|Z)$, a prior $p(Z)$, and a sample x, is the ELBO (evidence lower bound), the standard loss function used to train a variational autoencoder [6]. The proof of the bound after which the ELBO is instructive and straightforward: it correpsonds directly to the fact that adding the encoder to the PDG can only increase inconsistency.

At first glance it may seem as though the minimum value of the PDG scoring function (i.e., the inconsistency) is just one aspect of the scoring function—but, perhaps surprisingly, the two are equivalent. The scoring function for a PDG M, applied to a joint distribution μ, is the inconsistency of the PDG obtained by adding μ (with high confidence) to M. Furthermore, calculating a PDG's degree of inconsistency and describing the corresponding optimal distribution are even more closely related than it might seem. It turns out that the ability to precisely calculating a PDG's degree of inconsistency is enough to do inference in essentially constant time [16]. In a sense, this means that it is no easer to know how exactly how inconsistent you are, than it is resolving all of your inconsistencies.

Since calculating a PDG's degree of inconsistency is so difficult in the general case, an obvious approach is to restrict one's attention to one small part of the model at a time, and resolve inconsistencies locally. This leads to an "approximate" algorithm for resolving inconsistencies bit-by-bit: focus on one small part of the graph, and resolve inconsistencies in it by controlling parameters of another small part. The result is a very general algorithm called Local Inconsistency Resolution [14]. Immediate special cases include the EM algorithm [10], message passing algorithms [9], adversarial training [1], and the adversarial process used to train generative adversarial networks (GANs) [4].

Confidence Functions. One key component of a PDG is a numerical degree of "confidence" in each piece of probabilistic information (and in the functional depednence suggested by each arc). There is a strong precedent for doing things the way we do—i.e., by taking weighted sums of evidence—but what exactly does confidence *mean* in contexts like this? Confidence is often viewed as a synonym for likelihood, but that's not what's happening here. While high confidence is difficult to distinguish from high probability, there is a sense in which low confidence can bet quite different from low probability. What does a weight of zero mean in the context of linear regression, or in the multiplicative weights algorithm? It doesn't mean that a given feature is *wrong*, but rather that it is not trusted. Suppose you are 60% sure that X is true, and then an untrusted party tells you X is true. Having no confidence in this statement is quite different from thinking that X is false, or that this statement is a lie. You simply don't trust it.

Although obvious in retrospect, this distinction can be difficult to see at first. I attribute this to the complete cultural dominance of probability, which is itself a coherent (and not unrelated) interpretation of the word "confidence", and shadows this other concept. The mathematical foundation for these ideas applies extremely broadly. It helps to explain why we update our learning models incrementally (i.e., with "low confidence"). In some sense, it genrealizes the ideas of Shafer's *Theory of Evidence* [18] to almost any belief state. These ideas have laid out in an unpublished paper entitled *Learning with Confidence* which I expect to submit in the coming months. The key feature of a confidene function is that it allows the learning of a statement φ to be continuously parameterized by confidence value $s \in [0, 1]$, where $s = 0$ leaves your beliefs unchanged, $s = 1$ is an idempotent update.

Mixture Languages. Confidence functions can be used as the basis of an inductive semantics for a programing language. There is an important distinction between a process whose state is continuous, and one that evolves continuously over time. There are many general-purpose programming abstractions for modeling processes with continuous state (e.g., real-valued variables and probabilistic programs), but programming languages for modeling of continuous time processes are quite domain-specific. This is because the correctness of such computations is not considered a matter of computer science—after all, the correctness of a physics engine is a matter of physics, and the correctness of an animation is a matter of taste. Yet continuous-time processes of every kind compose in the same (surprisingly unfamiliar) way: they can be "mixed". This is different from the dominant model of concurrency, in which parallel composition amounts to a non-deterministic interleaving of atomic instructions and is notoriously unintuitive. But for continuous processes—and in particular anything described by a confidence function—concurrency has a single clear and intuitive meaning: run them at the same time. Roughly speaking, this coincides with a uniform infinitesimal interleaving. In work that will be presented at the Languages for Inference (LAFI) workshop at POPL '24, Jialu Bao and I outline a hierarchy of operational semantics for mixture languages and show how to inductively define sequential and parallel composition, as well as conditionals in this setting.

One of the most interesting discoverie is a duality between *observe* and *sample* commands. Relative entropy is an asymmetric measure of discrepancy between two probability distributions, one representing a belief and the other representing reality. We give semantics to **observe** $p(Y|X)$ by allowing p to play the rule of belief, and to **sample** $Y \sim p \mid X$ by allowing it to play the role of reality. With these definitions, we find that an **observe** command amounts to multiplicative interpolation, while a **sample** command interpolates additively. Moreover, PDGs (and hence other graphical models) are mixtures of **observe** commands. Traditional probabilistic programs are sequences of **sample** commands, which are themselves mixtures of deterministic programs. The standard algorithms captured by local inconsistency resolution [14] can also be described with such a programming language.

Future Work

Going forwards, I would like to extend these mathematical foundations of fallable probabilistic agents to other domains. In addition, although much of what I have described is quite abstract, I would like to reintroduce a focus on application to my work in the future. Here are some of the ideas I would like to pursue next.

PDG-based Logic. There is a natural way to use PDG semantics to define a logical entailment relation. The resulting logic is quite unusual because it captures a lot of probabilitic reaosning, and gives meaningful results even if all relevant formulas are contradictions.

PDGs and Probabilistic Databases. PDGs are one way of storing probabilistic information. Another is a probabilistic database. Preliminary investigations suggest that PDGs can store similar kinds of information, but without making as many independence assumptions. It appears also that the structural aspects of PDGs can model quantifiers.

Applications for PDGs. Any situation in which there is a common loss function, PDG inconsistency reproduces it. But what about non-standard situations? Although I have several toy examples where this way of selecting a loss function is better than the current standard, I would like to find a setting people care about in which PDGs can provide a better loss than one currently in use.

References

[1] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Š rndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. Evasion attacks against machine learning at test time. In *Advanced Information Systems Engineering*, pages 387–402. Springer Berlin Heidelberg, 2013.

[2] Venkat Chandrasekaran, Nathan Srebro, and Prahladh Harsha. Complexity of inference in graphical models. *arXiv preprint arXiv:1206.3240*, 2012.

[3] Leon Festinger. Cognitive dissonance. *Scientific American*, 207(4):93–106, 1962.

[4] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples, 2014.

[5] Joseph Y Halpern. *Reasoning About Uncertainty.* MIT press, 2017.

[6] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2014.

[7] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques.* MIT press, 2009.

[8] Frank R Kschischang, Brendan J Frey, and H-A Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on information theory*, 47(2):498–519, 2001.

[9] Tom Minka. Divergence measures and message passing. Technical Report MSR-TR-2005–173, Microsoft Research, Cambridge, U.K., 2005.

[10] Radford M Neal and Geoffrey E Hinton. A view of the em algorithm that justifies incremental, sparse, and other variants. In *Learning in Graphical Models*, pages 355–368. Springer, 1998.

[11] Yu E Nesterov, Michael J Todd, and Yinyu Ye. Infeasible-start primal-dual methods and infeasibility detectors for nonlinear programming problems. Technical report, 1999.

[12] Judea Pearl. Bayesian networks. 2011.

[13] Oliver E Richardson. Loss as the inconsistency of a probabilistic dependency graph: Choose your model, not your loss function. *AISTATS '22*, 151, 2022.

[14] Oliver E Richardson. The local inconsistency resolution algorithm (workshop version), 2023. ICML '23 Workshops: Local Learning Workshop (LLW) and Structured Prediction in Generative Modeling (SPIGM).

[15] Oliver E Richardson and Joseph Y Halpern. Probabilistic dependency graphs. *AAAI '21*, 2021.

[16] Oliver E Richardson, Joseph Y Halpern, and Christopher De Sa. Inference for probabilistic dependency graphs. *UAI '23*, 2023.

[17] Leonard J Savage. The theory of statistical decision. *Journal of the American Statistical association*, 46(253):55–67, 1951.

[18] Glenn Shafer. *A Mathematical Theory of Evidence*, volume 42. Princeton university press, 1976.

[19] Daniel Smilkov, Nikhil Thorat, Been Kim, Fernanda Viégas, and Martin Wattenberg. Smoothgrad: removing noise by adding noise. *arXiv preprint arXiv:1706.03825*, 2017.

[20] John von Neumann and Oskar Morgenstern. Theory of games and economic behavior. *Princeton University Press*, 1953.