The Role of Massively Multi-Task and Weak Supervision in Software 2.0

Alexander Ratner
Stanford University
ajratner@cs.stanford.edu

Braden Hancock
Stanford University
bradenjh@cs.stanford.edu

Christopher Ré
Stanford University
chrismre@cs.stanford.edu

ABSTRACT

Over the last several years, machine learning models have reached new levels of empirical performance across a broad range of domains. Driven both by accuracy improvements and deployment advantages, many organizations have begun to shift to learning-centered software stacks—a new mode that has been called *Software 2.0*. This approach holds the promise of radically accelerating the construction, maintenance, and deployment of software systems, and opens up a broad research agenda around changes to hardware, systems, and interaction models. However, these approaches require one critical and often prohibitively expensive ingredient: labeled training data.

We outline a vision for a Software 2.0 lifecycle centered around the idea that labeling training data can be *the* primary interface to Software 2.0 systems. In our envisioned approach, Software 2.0 stacks are programmed using *weak supervision*—i.e. noisier, programmatically-generated training data—which is specified at various levels of declarative abstraction and precision, and then combined using unsupervised statistical techniques. The codebase for Software 2.0 is also radically different: we envision training labels for tens or hundreds of different tasks across an organization combined in a *massively multitask* central model, leading to amortization of labeling costs and new models of software reuse and development. Finally, Software 2.0 stacks are deployed by using collected training labels to supervise commodity model architectures over different servable feature sets. We outline the components of this envisioned lifecycle, and provide an interim report on Snorkel, our prototype Software 2.0 system, based on our experiences working on problems ranging from ad fraud to medical diagnostics with some of the world's largest organizations.

ACM Reference Format:
Alexander Ratner, Braden Hancock, and Christopher Ré. 2018. The Role of Massively Multi-Task and Weak Supervision in Software 2.0. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

In the past several years, deep learning models have achieved a notable set of accomplishments, hitting an inflection point on many traditionally challenging tasks in image [11] and speech recognition [34]; beating human opponents at games such as Go [30]; and displaying a surprisingly flexible adaptation to a range of other problems and domains. In response, large corporations have invested billions of dollars in reinventing themselves as "AI-centric"; swaths of academic disciplines have flocked to incorporate machine learning into their research; and a wave of excitement about AI and ML has proliferated through the broader public sphere.

Much of the adoption of machine learning in industry, however, has in fact been driven by the potential development and deployment advantages of replacing traditional software stacks. Organizations in a range of domains have found that this *Software 2.0* approach is often easier to build, deploy, and re-use, for several emerging reasons [17, 28, 33]:

(1) *The Death of Feature Engineering:* The features of traditional ML models are notoriously difficult to develop, maintain, and deploy. Instead, in Software 2.0 approaches, modern machine learning models learn features automatically from data and achieve much higher accuracy on a range of tasks.
(2) *Adaptability and Modularity:* Software 2.0 approaches have the potential to be far more adaptable, both to new domains and to shifting performance requirements, due to the modular nature of modern model architectures.
(3) *Homogenous Deployment:* Finally, traditional "Software 1.0" stacks face deployment challenges such as varied execution environments, changing dependencies, and unpredictable memory allocation requirements. In the Software 2.0 approach, these black-box pieces of code are replaced with relatively homogeneous networks, with matrix multiplies effectively becoming the new JVM, a transportable set of "write once, run anywhere" operations with predictable runtimes and memory requirements.

However, all these prospective benefits of the Software 2.0 approach are contingent on a critical ingredient: *large labeled training sets*. Across organizations of all sizes, we have witnessed training data becoming the key bottleneck in developing Software 2.0 systems. For many of these, training data maintenance is a major capital expenditure—a critical asset that enables core organizational products, but unfortunately depreciates and becomes underutilized with time. In response to this, over the past three years we have developed Snorkel [23][1], a system for programmatically creating and modeling training data for tasks over arbitrary data types such as text, images, time series, financial data, and many others. Snorkel has now been used by over 35 major organizations, including active use in high-impact settings such as law enforcement, medical diagnostics, and consumer applications. These collaborations have given us a more nuanced view of three core challenges within the lifecycle of organization-scale Software 2.0 systems:

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

[1] snorkel.stanford.edu

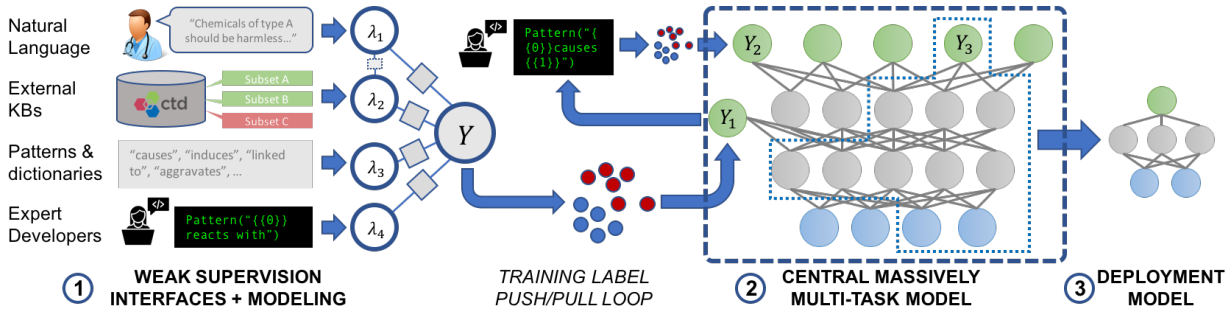

Figure 1: We envision a Software 2.0 pipeline consisting of (1) *building* training sets from weak supervision, provided via a stack of interfaces at different levels of abstraction; (2) *combining* training signals for tasks across an organization into a central *massively multitask* model, which allows developers to contribute and use task models via the simple interface of *labels*; and (3) *deploying* servable models by detaching tasks from the central model and distilling them into commodity edge models.

(1) **Diverse Sources of Supervision.** An increasingly prominent trend is the use of noisy, programmatically-generated training data—often termed *weak supervision*—to train Software 2.0 models. For example, developers might utilize external knowledge bases [20], ontologies, heuristic rules [23], patterns, noisy crowd labels [7], or even other classifiers to generate labels [24]. These sources have diverse accuracies, coverages, and correlations, and *modeling* their accuracies may be the way to improve accuracy of downstream models—but is a major technical challenge given the lack of ground truth. Additionally, providing accessible ways for non-programmers to provide weak supervision is another key challenge.

(2) **Multi-Task Problems.** Most real-world problems involve solving multiple related sub-tasks, each of which requires its own training datasets. Even weakly supervised data can still be expensive and time-consuming to label—often requiring the direct attention of expensive subject matter experts like doctors, engineers, and research scientists—and it depreciates when the modeling tasks change. We would ideally like ways to amortize the cost of generating training data across related tasks, and to foster new methods of data and label reuse.

(3) **Servable Deployment.** Often, the features of the data that are available at labeling time are different from the *servable* features that are available in a deployed setting. Moreover, various deployment settings may have different performance, memory, and latency requirements. We would like to be able to use available training labels to quickly deploy new models across different servable or edge settings.

Despite these challenges, we see training data as a valuable medium for collecting subject matter expert input from across an organization, storing and reusing it in a way that is agnostic to the particular models being used, and transferring it between modalities and deployment settings. To this end, we describe a vision for an end-to-end framework for creating, managing, and deploying Software 2.0 systems at organizational scale, which addresses the aforementioned challenges through three core components:

(1) **Weak Supervision as the Declarative Interface:** In our envisioned framework, developers provide weak supervision via interfaces at various layers of abstraction, from high-level interfaces requiring no programming knowledge to low-level ones offering complete expressive control. The generated training labels are then automatically denoised and combined using unsupervised statistical techniques [25].

(2) **Massively Multitask Models as the Central Codebase:** We then envision developers across an organization pooling their training labels in a central, *massively multitask (MMT)* model, that serves as the equivalent of a central codebase. Recently, *multitask* learning [5, 22, 29] has become an increasingly popular way of effectively pooling learned representations between tasks for improved performance. While current approaches consider a small, fixed set of hand-labeled training sets, we envision our central MMT model dynamically aggregating labels for tens to hundreds of different tasks across an organization. We envision this defining a new mode of software and label reuse across an organization, where each contributed task can both be immediately used in other applications, but also potentially raises the quality of many other tasks.

(3) **Servable Commodity Models for Edge Deployment:** Finally, building on the increasing availability of commodity, open-source model architectures, we envision a simplified *qualification* process where new model architectures are chosen for each deployment setting, and then supervised using predicted labels from the central MMT model. In this style of approach, developers have the flexibility to specify weak supervision over any range of expensive, slow, private, or otherwise *non-servable* features, then automatically use this to train *servable* models for edge deployment.

By making training signals easy to specify, combining them across tasks using a central, massively multitask model, and automatically deploying them to servable settings, we expect our envisioned framework to radically change and accelerate the way that developers interact with and *program* Software 2.0 systems. We now go over the envisioned framework, experiences with our initial prototypes for various components within our system Snorkel, and a running use case of a Software 2.0 fraud detection system at a large web company.

2 BUILDING WEAK SUPERVISION SYSTEMS: THE PROGRAMMING STACK OF SOFTWARE 2.0

To avoid the expensive and time-consuming process of hand-labeling training data, machine learning practitioners are increasingly turning to *weak supervision* [24] to create their training data for tasks involving text, image, video, time series, and other data, including at major organizations with substantial resources such as Google [8, 9, 18], Facebook [19], and Microsoft [14]. The idea of using programmatic sources of supervision builds on a long history of work in distant supervision [1, 13, 20] and crowdsourcing [3, 6, 7, 16, 35]. However, as more types of weak supervision sources are integrated, accounting for their accuracies, coverages, and correlations becomes increasingly important to achieve optimal results.

In response to this challenge, we developed Snorkel, a framework for creating and modeling weak supervision [23]. With Snorkel, instead of providing labels, users provide *labeling functions (LFs)*, programmatically label data based on patterns, heuristics, external knowledge bases, etc. Snorkel then addresses the challenge of uneven training source quality by automatically learning a statistical model of the labeling functions' accuracies [25] and correlation structure [2]. This learned model combines and reweights the labeling functions' labels, producing a set of probabilistic training labels. We have seen that this overall approach can boost downstream predictive accuracy by as much as 132% over prior heuristic approaches in information extraction applications [23], image classification [32], and a range of other problem types. We now introduce a running case study, based on real scenarios that we have observed over the course of our collaborations with industry partners:

Running Case Study: Real-Time Fraud Detection

A large organization, `WebCorp`*, wants to train a suite of real-time fraud detection systems. Ground truth data is not available in any sizable quantity, but various forms of weak supervision from subject matter experts across the organization are available; for example:*

(1) The fraud detection team has a legacy fraud prediction system for text data, composed of complex linguistic rules.
(2) A research scientist has built several graph prediction models over collected network statistics.
(3) Heuristic rules are automatically compiled monthly by the quality control team based on aggregated statistics.
(4) Common patterns observed in fraud instances can be verbally explained by non-programmer fraud experts.

`WebCorp` *would like to opportunistically bring all these available sources of signal to bear by using them to train a single, high-performance fraud detection model. However, doing this requires properly accounting for the variety in accuracy, correlations, coverage, and provenance of the sources.*

We envision Software 2.0 systems supporting a full stack of supervision interfaces (Fig. 2). Just as SQL and other higher-level declarative languages have made a vast range of algorithmic capabilities accessible to non-expert users, we see these higher-level supervision interfaces doing the same for Software 2.0, allowing non-experts to declaratively specify noisy sources of signal. Tools

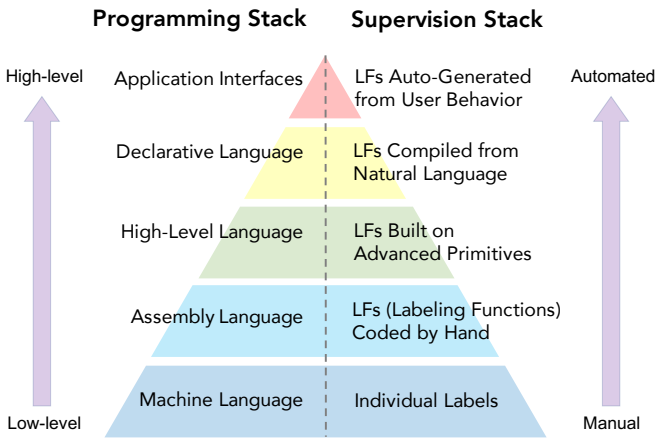

Figure 2: Just as higher-level programming languages magnify a user's algorithmic capabilities, we envision higher-level supervision interfaces that magnify a user's labeling capabilities. These higher-level inputs can be compiled into *labeling functions (LFs),* which in turn are used by systems like Snorkel to generate labeled training sets.

like Snorkel can then be viewed as supervision source *compilers*—converting higher-level inputs into lower level inputs (i.e., labels), which are then easily used to train a range of commodity models.

We have constructed a number of prototype systems exploring interfaces up and down this stack. In one recent project [32], we preprocess image data with unsupervised, pre-trained models (which tag bounding boxes, edges, geometric shapes, etc.), and then let users write higher-level labeling functions over these primitives. In another [10], we demonstrate how we can accept natural language explanations as inputs, which are then compiled via a semantic parser into sets of labeling functions. We are currently exploring directions such as automatic synthesis of labeling functions from high-level sketches and from "analyst exhaust" such as clickstreams and eye-tracker signals. We believe that by combining these and other approaches into a single coherent framework, we can create an accessible interface to Software 2.0 that harnesses the full spectrum of supervision signal available at an organization.

3 COMBINING TRAINING SIGNALS WITH MASSIVELY MULTITASK MODELS

Given the initial success of Software 2.0-style efforts, organizations are beginning to apply this approach to an increasingly large set of problems, which themselves are often comprised of multiple sub-tasks. This has led to a growing emphasis on amortizing costs and increasing *reuse* of training data across tasks in an attempt to avoid expensive *de novo* model construction. One popular approach for this is *transfer learning* [22][2], which is the strategy of training a model for a first task, and then repurposing it for a second task. A directly related approach is *multi-task learning* [5, 15, 29, 31], in which a model is trained on multiple tasks, learning a shared representation of the data which can improve with more tasks. In

[2]Which, notably, made it into Amazon's most recent shareholder's letter [4].

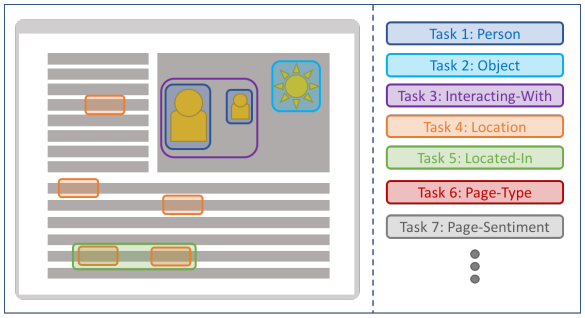

Figure 3: A mockup of an interface to the task predictions of a central *massively multitask (MMT)* model over multimodal webpage data. Here, developers can access the predicted labels for various tasks—for example, tagging objects and relations in text and image data, as well as classifying the webpage into higher-level categories—and in turn can use these to help supervise new tasks, which are then contributed back to the central MMT model. Each new task adds minimal additional training time, potentially benefits from and improves the quality of other existing tasks, and can immediately be used by other developers.

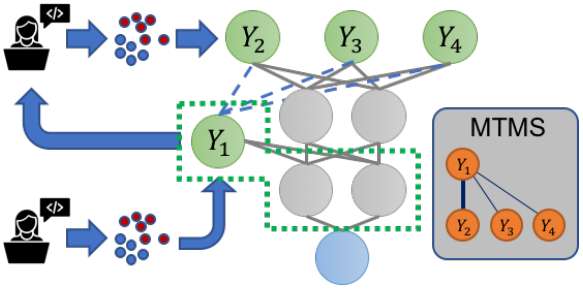

Figure 4: In the envisioned architecture, weak supervision for tens to hundreds of tasks (green) across an organization are used to supervise a central MMT model with shared representation layers (gray). This MMT model functions as the Software 2.0 equivalent of a central code repository, enabling new modes of software development and reuse. For example, a first developer might supervise a task Y_1 corresponding to low-level feature extraction (e.g., tagging entity mentions in text), which is then used by a second developer to generate training labels for a higher-level task Y_2 (e.g., extracting entity-entity relations from text). A *multitask management system (MTMS)* tracks dependencies between tasks and executes incremental retraining of the MMT model as necessary.

general, we observe these techniques being used in effective but one-off ways—as an engineer's trick to coax extra performance out of individual models—and almost exclusively on small, fixed sets of carefully hand-curated training sets. However, we see these trends pointing in the direction of a much larger paradigm shift in how developers program, use, and re-use both labeled data and software within an organization.

In our framework, we envision a central, *massively multitask (MMT) model* that collects tens to hundreds of weakly-supervised tasks from across an organization, enabling a new level of data reuse and fundamentally changing how developers *program* Software 2.0. This central "mother" model will be the Software 2.0 equivalent of a central organizational codebase, but with noteworthy paradigm-shifting benefits:

- *The Rising Tide of Multi-Task Supervision:* Adding tasks to the central MMT model has the potential to improve the performance of other tasks [5, 15]. We envision this leading to a virtuous cycle in which engineers are motivated to add their tasks to the central MMT model to gain performance benefits "for free", which in turn may boost the performance of other tasks in the model.
- *Simple, Label-Based APIs:* In a traditional codebase, a developer often must overcome non-trivial hurdles to use another developer's contributed code, due to arbitrary input and output signatures, dependencies, and performance requirements. In our envisioned MMT "codebase", regardless of how a task is specified or supervised, its input and output API is simply a set of labels (see Fig. 3). We envision developers being able to use labels from existing tasks in the MMT model to help specify and supervise new ones (see Fig. 4).
- *Continuous Versioning:* Unlike in a traditional codebase, updates to a given task in the MMT model can be smoothly propagated to other tasks if desired, rather than requiring a discrete upgrade event. Old versions of tasks or layers in the MMT model can even be phased out gradually by interpolating between model versions.
- *Software 2.0 Form Factor:* While a high volume of commits to a central Software 1.0 codebase might risk bloat and standards degradation, adding tasks to a central MMT model will generally require minimal modification to the existing network structure—maintining the same, relatively computationally homogeneous architecture.

Before addressing technical challenges to realizing this vision, we revisit our running case study to illustrate the potential impact of this envisioned approach:

Running Case Study: Multiple Related Tasks

`WebCorp`*'s fraud detection tasks cover a variety of modalities, such as text, images, and full webpages. Currently, each vertical is handled separately, and consists of distinct, individualized models and training sets for each task. However, many of them relying on shared sub-tasks, third-party tools (e.g., commodity taggers), external resources (e.g., blacklisted IP addresses), and more broadly, concern similar concepts.* `WebCorp` *would like to capitalize on those similarities both to improve performance and to reduce the overhead associated with updating training sets for each task as new failure modes in their models are discovered and exploited.*

In the our envisioned Software 2.0 framework, `WebCorp`*'s process might look something like this:*

- *A central MMT model is initiated as the central component of* `WebCorp`*'s fraud detection suite. A task is added for each of the*

fraud detection verticals of interest, as well as for related lower-level tasks like feature extractors, using the existing training labels as supervision.

- *After some time, the engineers in one team observe a new failure mode in their webpage classifier, which they correct by writing additional labeling functions that use outputs from other, lower-level tasks contributed to the MMT model by other teams.*
- `WebCorp` *decides to add a new vertical to its fraud detection suite; the pre-trained weights of the shared layers in the central MMT model provide a strong starting point, reducing the amount of labeled data and supervision resources required to reach the required quality for deployment.*

While both transfer and multi-task learning are established techniques, the *massive, weakly-supervised* setting we propose requires solving a set of novel and fundamental technical challenges. One of these is combining the noisy, conflicting, and potentially correlated *weak, multi-task supervision* that users will provide. To begin to study this challenge, we recently developed a prototype system building on Snorkel, Snorkel MeTaL [26][3]. In Snorkel MeTaL, users specify multi-task labeling functions that directly label–or logically imply–labels for multiple tasks, which are related via a user-specified task graph. For example, if we are training a fine-grained entity tagger that tags mentions of specific professionals, e.g. {Doctor, Lawyer, etc.}, organizations, e.g. {Hospital, Office, etc.}, and more, we might also want to utilize coarser-grained labels we have for a higher-level task like {Person, Organization, etc.}. In Snorkel MeTaL, we can use all these supervision sources together, learning their accuracies and then training an automatically-compiled multi-task network. Our initial work shows impressive gains over alternative approaches: 20.2 F1 points over traditional supervision, 6.8 points over a majority vote baseline, and 4.1 points over a non-multi-task-aware data programming approach [27]. More importantly, however, our initial user studies highlight the benefit of allowing developers to focus "locally" on single tasks in isolation—e.g. Is this a doctor or a lawyer mention?—and then later merging the various subtasks into a single model automatically. We believe this is an example of one of the benefits the new programming model our MMT framework may provide.

As next steps, we plan to focus on other key technical challenges of this setting, including incremental training and maintenance of the central MMT model, tracking of task-usage dependencies for finer-grained model updates, automatic learning of task relationship structures, and new measures of "task head" stability balanced with intermediate "torso" layer generalization.

4 DEPLOYING SERVABLE MODELS AUTOMATICALLY FROM DEVELOPMENT

One of the major drivers of Software 2.0 adoption is the appealingly homogeneous compositions of neural network architectures, with matrix multiplies effectively replacing the various compilers and guardrails needed for arbitrary, black-box Software 1.0 code [17]. Already, hardware is racing to meet this appealingly general design specification, with billions of dollars being spent to prepare next generation hardware to better support Software 2.0 systems [21].

[3]https://github.com/HazyResearch/metal

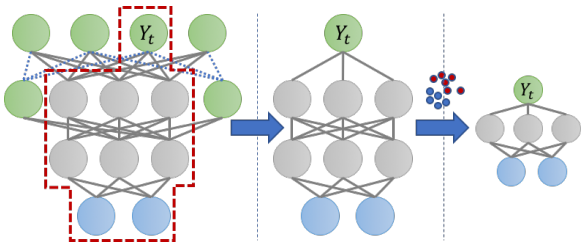

Figure 5: A model for task Y_t can be deployed in a simple *qualification* process: First, a copy of the full development MMT model (left) is made containing only the *task head* Y_t and relevant representation layers (middle). Second, this model—which may be over expensive or private *non-servable* data, or may be too large for edge settings—can then be used to train a commodity *deployment model* (right), which has the required performance specifications and runs over *servable* features.

Another complementary trend is the increasing availability—and *commodification*—of neural network architectures. Recent industry efforts to accelerate these trends include model zoos and standards such as ONNX[4], as well as supporting open source infrastructure such as TensorFlow and PyTorch. These trends afford flexibility: whereas traditional code takes effort to port to new modalities and deployment settings, here we can simply modify or swap out commodity model architectures, using the same training data labels.

We envision a simplified *qualification* process (Fig. 5) where new model architectures are chosen for each deployment setting, and then supervised using predicted labels from the central MMT model. This approach anchors on the core idea that many commodity model architectures—spanning a wide range of performance specifications and data types—can perform well enough if we have sufficiently large labeled training sets for them. Thus, we can take expensive and difficult-to-deploy Software 1.0 code, and/or our large central MMT model, and use these to label training data that can then be used to supervise a cheap commodity network for edge deployment (similar in process to model distillation techniques [12]).

One common scenario we have observed is when developers have access to a large set of features that are high-value but also costly to compute, slow, private, or otherwise *non-servable* in deployment settings. In this scenario, we can generate weak supervision—and train our MMT model—over the non-servable features, and then use this as supervision for a separate deployment model that operates over servable features. We return to our running case study:

Running Case Study: Training Servable Deployment Models
Thus far, `WebCorp`*'s engineers have assembled an organization-wide collection of training sets and models for many related fraud and risk-detection tasks. However, the majority of these labels and models operate over* non-servable *data such as monthly aggregate statistics, computationally expensive graph query models, and other similar features.*

[4]https://onnx.ai/

Instead, WebCorp *would like to deploy a model that operates—and predicts fraud or risk event types—over real-time streaming web data. In a prior approach, this would require repurposing an expert-level engineering team to develop an entirely new model from scratch. However, with our envisioned Software 2.0 system, a commodity architecture appropriate for this real-time deployment setting is automatically selected, and then trained using the existing weak supervision and models that had been built over the non-servable features.*

In Figure 5, we sketch a slightly more detailed schematic of a two-step pipeline in which a task Y_t is first separated out from the MMT model—by cloning the MMT model, and removing all the components specific to other tasks—and then used to train a new commodity deployment model that operates over servable features. We note that in this process, lineage information—e.g., how the task connected to other elements in the MMT model—can be used to help specify the deployment architecture. We believe that this lineage-informed model distillation and transfer will be a critical benefit of the proposed MMT-centric framework. Moreover, we note that the deployment model can be significantly smaller and simpler, focused primarily on inference at the edge with potentially a small capability for learning (e.g. for user fine-tuning).

5 CURRENT STATUS AND NEXT STEPS

Our envisioned system for programming and deploying Software 2.0 systems via training data management consists of three main stages: *building* training sets from diverse layers of weak supervision interfaces, *combining* training signal from multiple tasks in central, massively multitask models; and *deploying* models directly from this training signal over development data to servable models. For this first stage, we plan to continue "climbing up the stack" by providing increasingly high-level, minimal-effort interfaces for specifying weak supervision, and are currently exploring intersections with areas such as program synthesis and semantic parsing, and with input devices ranging from natural language parsers to eye trackers. For the second stage, our next steps are to explore the data management challenges of massively multi-task models at scale, such as incremental model maintenance and task relationship structure induction, and the new programming paradigms they lead to. Finally, we continue to work with several industry and medical collaborators to study ways of faster and more automatic model deployment from non-servable training data to servable modalities. We plan to continue studying the totality of this pipeline, and its associated technical challenges, within our open-source Software 2.0 framework, Snorkel.

ACKNOWLEDGMENTS

Thanks to Charles Srisuwananukorn, Feng Niu, Kunle Olukotun, Andrej Karpathy, and members of Hazy Research for their valuable feedback in numerous discussions.

REFERENCES

[1] E. Alfonseca, K. Filippova, J. Delort, and G. Garrido. 2012. Pattern learning for relation extraction with a hierarchical topic model. In *Proceedings of ACL: Short Papers*. ACL, 54–59.
[2] S. Bach, B. He, A. Ratner, and C. Ré. 2017. Learning the structure of generative models without labeled data. In *Proceedings of ICML*.
[3] D. Berend and A. Kontorovich. 2014. Consistency of weighted majority votes. In *Proceedings of NIPS*. 3446–3454.
[4] J. Bezos. 2017. Amazon Shareholder's Letter. https://www.sec.gov/Archives/edgar/data/1018724/000119312518121161/d456916dex991.htm.
[5] R. Caruna. 1993. Multitask learning: A knowledge-based source of inductive bias. In *Machine Learning: Proceedings of the Tenth International Conference*. 41–48.
[6] N. Dalvi, A. Dasgupta, R. Kumar, and V. Rastogi. 2013. Aggregating crowdsourced binary ratings. In *Proceedings of WWW*. ACM, 285–294.
[7] A. Dawid and A. Skene. 1979. Maximum likelihood estimation of observer error-rates using the EM algorithm. *Applied Statistics* (1979), 20–28.
[8] M. Dehghani, A. Severyn, S. Rothe, and J. Kamps. 2017. Learning to Learn from Weak Supervision by Full Supervision. *arXiv preprint arXiv:1711.11383* (2017).
[9] M. Dehghani, H. Zamani, A. Severyn, J. Kamps, and W. B. Croft. 2017. Neural ranking models with weak supervision. In *Proceedings of ACM SIGIR*. ACM, 65–74.
[10] B. Hancock, P. Varma, S. Wang, M. Bringmann, P. Liang, and C. Ré. 2018. Training Classifiers with Natural Language Explanations. (2018).
[11] K. He, X. Zhang, S. Ren, and J. Sun. 2016. Deep residual learning for image recognition. In *Proceedings of CVPR*. 770–778.
[12] G. Hinton, O. Vinyals, and J. Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* (2015).
[13] R. Hoffmann, C. Zhang, X. Ling, L. Zettlemoyer, and D. Weld. 2011. Knowledge-based weak supervision for information extraction of overlapping relations. In *Proceedings of ACL*. Association for Computational Linguistics, 541–550.
[14] Z. Jia, X. Huang, I. Eric, C. Chang, and Y. Xu. 2017. Constrained deep weak supervision for histopathology image segmentation. *IEEE transactions on medical imaging* 36, 11 (2017), 2376–2388.
[15] L. Kaiser, A. Gomez, N. Shazeer, A. Vaswani, N. Parmar, L. Jones, and J. Uszkoreit. 2017. One model to learn them all. *arXiv preprint arXiv:1706.05137* (2017).
[16] D. Karger, S. Oh, and D. Shah. 2011. Iterative learning for reliable crowdsourcing systems. In *Proceedings of NIPS*. 1953–1961.
[17] A. Karpathy. 2017. Software 2.0. https://medium.com/@karpathy/software-2-0-a64152b37c35.
[18] C. Liang, J. Berant, Q. Le, K. Forbus, and N. Lao. 2016. Neural symbolic machines: Learning semantic parsers on freebase with weak supervision. *arXiv preprint arXiv:1611.00020* (2016).
[19] D. Mahajan, R. Girshick, V. Ramanathan, K. He, M. Paluri, Y. Li, A. Bharambe, and L. van der Maaten. 2018. Exploring the Limits of Weakly Supervised Pretraining. *arXiv preprint arXiv:1805.00932* (2018).
[20] M. Mintz, S. Bills, R. Snow, and D. Jurafsky. 2009. Distant supervision for relation extraction without labeled data. In *Proceedings of ACL*. Association for Computational Linguistics, 1003–1011.
[21] K. Olukotun. 2018. Designing Computer Systems for Software 2.0. http://iscaconf.org/isca2018/docs/Kunle-ISCA-Keynote-2018.pdf.
[22] S. Pan, Q. Yang, et al. 2010. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering* 22, 10 (2010), 1345–1359.
[23] A.J. Ratner, S.H. Bach, H. Ehrenberg, J. Fries, S. Wu, and C. Ré. 2018. Snorkel: Rapid training data creation with weak supervision. In *VLDB*.
[24] A. Ratner, S. Bach, P. Varma, and C. Ré. 2017. Weak Supervision: The New Programming Paradigm for Machine Learning. https://hazyresearch.github.io/snorkel/blog/ws_blog_post.html.
[25] A. Ratner, C. De Sa, S. Wu, D. Selsam, and C. Ré. 2016. Data programming: Creating large training sets, quickly. In *Proceedings of NIPS*. 3567–3575.
[26] A. Ratner, B. Hancock, J. Dunnmon, R. Goldman, and C. Ré. 2018. Snorkel MeTaL: Weak Supervision for Multi-Task Learning. In *Proceedings of DEEM Workshop*. ACM, 3.
[27] A. Ratner, B. Hancock, J. Dunnmon, F. Sala, S. Pandey, and C. Ré. 2018. Training Complex Models with Multi-Task Weak Supervision. (2018).
[28] C. Ré. 2018. Software 2.0 and Snorkel: Beyond Hand-Labeled Data. In *Proceedings of SIGKDD*. ACM, 2876–2876.
[29] S. Ruder. 2017. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098* (2017).
[30] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, et al. 2017. Mastering the game of Go without human knowledge. *Nature* 550, 7676 (2017), 354.
[31] A. Søgaard and Y. Goldberg. 2016. Deep multi-task learning with low level tasks supervised at lower layers. In *Proceedings of ACL (Short Papers)*, Vol. 2. 231–235.
[32] P. Varma, B. He, P. Bajaj, N. Khandwala, I. Banerjee, D. Rubin, and C. Ré. 2017. Inferring Generative Model Structure with Static Analysis. In *Proceedings of NIPS*. 240–250.
[33] P. Warden. 2017. Deep Learning is Eating Software. https://petewarden.com/2017/11/13/deep-learning-is-eating-software/.
[34] W. Xiong, J. Droppo, X. Huang, F. Seide, M. Seltzer, A. Stolcke, D. Yu, and G. Zweig. 2017. The Microsoft 2016 conversational speech recognition system. In *Proceedings of ICASSP*. IEEE, 5255–5259.
[35] Y. Zhang, X. Chen, D. Zhou, and M. Jordan. 2014. Spectral methods meet EM: A provably optimal algorithm for crowdsourcing. In *Proceedings of NIPS*. 1260–1268.