Contents

Workshop website (with videos): `https://simons.berkeley.edu/workshops/schedule/3749`

1 Machine Learning from Verbal User Instruction (Tom Mitchell, Carnegie Mellon University)

Abstract: Unlike traditional machine learning methods, humans often learn from natural language instruction. As users become increasingly accustomed to interacting with computer devices using speech, their interest in instructing these devices in natural language is likely to grow.

We present our Learning by Instruction Agent (LIA), an intelligent personal agent that users can teach to perform new action sequences to achieve new commands, using solely natural language interaction. LIA uses a CCG semantic parser to ground the semantics of each command in terms of primitive executable procedures defined in terms of the sensors and effectors of the agent. When LIA is given a natural language command that does not understand, it prompts the user to explain how to achieve the command through a sequence of steps, also specified in natural language. As a result of the instruction episode, LIA learns two types of knowledge: (1) a new procedure defined in terms of previously understood commands, and (2) an extension to its natural language lexicon that enables it to understand and execute the new command across multiple contexts (e.g., having been taught how to "forward an email to Alice," LIA can correctly interpret the command "forward this email to Bob." This talk will present LIA, plus ongoing research to extend it to include demonstration-based learning, and theoretical questions raised by such learning from instruction.

Background reading: `http://ai2-website.s3.amazonaws.com/publications/LearnByInst.pdf`

To think about the future of machine learning, look at how humans learn that machines don't. ML has been overfocused on one paradigm.

Suppose there is a sensor-effector system (like a phone) you would like to teach through human instruction. Sensors include mike, camera, GPS, web pages, incoming message, news... Effectors include vibrate, speak word, display message, seach web, invoke app...

If you could teach your phone things, what would you teach? Most things we say now are instantaneous: what is the weather?

We can have conditional strategies monitored all the time. Ex. Google has: Whenever I have to catch a flight, alert me to head to the airport so that I'll arrive an hour before flight time.

But there are lots of other things I want:

- Whenever it snows at night, set my alarm 30 minutes earlier.
- If somebody sends me a picture containing my mom, email it to her.
- Whenever a student sends email about a homework question, forward it to the right teaching assistant.

These are all in the sensor-effectuator closer of the phone: depending on things the phone can sense and actions the phone can do.

We assume

- set of sensors, effectors
- primitive language capability (language lexicon able to invoke each sensor, effector)
- general learning mechanisms.

We want the system to learn

- langauge: lexicon entries to understand new phrases. ("drop a note to Bill") The system might not know the phrase at first, but it should learn it.
- Capabilities: new sensors and effectors defined in terms of existing.
- Representations: new concepts and instances linked to instance.

  Ex. define concept "workshop". One instance is Interactive Learning 2017.

  This becomes another sensor. "Whenever I'm attending the workshop,..."

I'll show a prototype which allows teaching new commands.

1.1 Teaching new commands

Philosophy:

- Teaching the agent should be like teaching another human: it will remember.
- Anyone should be able to teach. Anyone can become a programmer!
- Verbal instruction and demonstration
- Anytime.

If the system doesn't know what something means, "I don't know how to do that, can you teach me what it means?"

- "Tell Tom to come to the meeting a few minutes early."
- Instruct by breaking into substeps.
- "I'm trying to generalize to other similar commands."
- "Tell Bob to bring some coffee." This is now comprehendable!

The system learned new language competence, and learned a procedure.

What it means to understand a command means to create the code to make the command execute.

The generalization comes out of inherent structure of the semantic parsing, and some heuristics to make it work as well as possible.

If the generalization was not perfect, how to correct? No good answer yet...

This system right now deals when unknown semantics have to do with phrasings and verbs (tell). It didn't know "tell" before; now it can show the semantics of "tell".

Meta-conversations: remember that time when I told you to tell Bob to bring coffee...

CCG (combinatory categorical grammar) parsing maps natural language to logical forms. "Prepare an email to Bill" becomes (doSeq (createEmail) (set recipient Bill)).

Parsing steps combine sentence parts both syntactically and semantically: Adjective Noun is a NP, where the logical form of the NP is Adjective(Noun).

"Set the subject to time to go."

- subject: FieldName, becomes (lambda x (getMutableFieldByFIeldName x)).
- time to go: StringV
- to: absorb phrase to right PP_StringV.
- End: (setFieldFromString (getMutableFieldByFieldName subject) (stringValue "time to go")).

Grammar rules can be ambiguous: we say this "can" be parsed a certain way rather than it will.

There are 2 things to learn: Learn new parsing rules and right control strategies to control the parse. Ex. "set" makes it more likely to parse the RHS as a string.

In practice, the number of potential parses increases rapidly as vocabulary.

We learn loglinearly and use beam search, keep 15 most valued parses. If you make complex sentences, the beam is not wide enough.

In the example, the algorithm learned the semantics of "tell".

Generalization algorithm: parameterize any text span whose semantic parse is a subtree of the full sentence semantic parse (replace by variable, ex. arg0="Tom", arg1="I am late").

We don't have things like yet: "Inform" is a synonym of "tell".

Everything is symbolic so far. Logical forms do have to be executable. SA: Embeddings could inform probabilities.

What about sending email vs. text? (Synonyms in action space.) It doesn't have a notion of goals, subgoals, planning. This would be a much richer substrate.

We ran experiments on Mechanical Turk. Create a new email and set the recipient to the current email's sender. Set recipient to Mom. I can't. Set recipient to Mom's email. This is a clarification that teaches the machine.

Subject taught agent new commands, saved on average 52 subcommands. Only 41% completed the task.

No cumulation in experiment. People can have own dialects (ex. send email vs. text). How about customizing from fully loaded sytem?

1.2 Teaching conditional strategies to automatically invoke commands

If (sensor value) then (command).

1. If I get new email from...: this is easily defined in language, "Test whether value of the from: field equals..."

   Declarative.

2. Concept definable as active sensing procedure. "If it snows at night..." Open weather app, read current conditions, equal to snow?

   Procedural.

3. Concept not definable in closed form, or procedurally. "If I get spam email..." No closed declarative or procedural definition. Generalize from examples with user instruction.

Typical user instruction is "it's probably spam if it's trying to sell me something."

The meaning has to be grounded in observable properties in email. Our approach is to jointly learn meaning of these natural language statements, (boolean) values for individual emails, and a classifier based on these values plus observed features.

MTurkers generated emails in the following classes.: contact, employee, event... Other MTurkers provided advice on classifying them.

The F1 score of the learned classifier is higher with sentences provided by users.

1.3 Teaching by demonstration

SUGILITE: Creating multimodal smartphone automation by demonstration.

Ex. "Order a Cappucino." Demonstrate using any third party apps. Determine parameter using voice command. Analyze app's user interface. Click on behalf of user.

1.4 Future work

Future work:

- Integrate verbal instruction with demonstration
- Share learning across agent (phones)
- Incremental refinement ("if it snows" to "if it snows lots")
- mixed initiative teachine dialogs
- If (sensor) then (effector) because (goal). This gives us a way to teach the system reasoning!
- Combine statistical, verbal, demonstration, and experiments.

Theory needed:

- Sample complexity impact from different types of instructions.

  Communcation is not necessarily a label, but another type of instruction: demonstrations with or without commentary, suggested new features...

  There is an effectively infinite set of features defined by a grammar. In ML we usually assume we have a nice set of features.

- Theory of agent architectures: what are sufficient initial capabilities to assure learnability of everything in the sensor-effector closure?

  What is the value of a good curriculum?

  What are sufficient conditions to assure non-damaging learning?

In one sentence: because now computers can understand what we're saying to some degree, there is this huge new underexplored area of verbal user instruction. Opening it up we get into interesting practical and theoretical questions, ex. incompleteness of verbal instruction.

Android platform easier than iOS. Toby Li: track every keystroke on every application.

Inverse question: how machine can teach humans to teach, interact in way it can understand. Student saying "that wasn't very good". I didn't understand your command because... there are many Bobs, I couldn't parse this last part...

Speech recognition is a bottleneck. We use google speech. One problem is that you get back the whole things; we want to track in real time the utterance. There are other speech systems at CMU (finer grained) but they are not as globally competent. Is there a way to make speech recognition better with grammar/parsing?

Gather vague and informative signals? Attention, etc. OpenSmile.

Part of the experiment: given human understanding, can they break it into pieces? Maybe it doesn't have to work as well as human understanding.

2 Machine teaching in interactive learning, Jerry Zhu, University of Wisconsin-Madison

Abstract: If machine learning is to discover knowledge from data, then machine teaching is an inverse problem to pass the knowledge on. More precisely, given a learning algorithm and a target model, the goal of machine teaching is to construct an optimal (e.g. the smallest) training set from which the algorithm will learn the target model. I will discuss several aspects of machine teaching that connect to interactive machine learning.

What do we really want from interactivity?

Imagine we're doing binary classification, $x \in [0,1]$, label $y \in \{-1,1\}$, the hypothesis space is threshold classifiers $\mathcal{H} = \{\theta \in [0,1] : \widehat{y} = \mathbb{1}_{x \geq \theta}\}$.

Contrast 3 types of teaching. Assume noiseless.

1. PAC (passive) learning: sample iid. A learner gets a batch of iid samples. With large probability
$$|\widehat{\theta} - \theta^*| = O(n^{-1}) \leq \varepsilon \tag{1}$$
$$n \geq O(\varepsilon^{-1}). \tag{2}$$
We can do much better with active learning.

2. Active learning: the learner picks query x; human oracle answers $y = \theta^*(x)$. Do binary search,
$$|\widehat{\theta} - \theta^*| = O(2^{-n}) \leq \varepsilon \tag{3}$$
$$n \geq O(\lg(\varepsilon^{-1})). \tag{4}$$

3. An ideal human teacher: Teacher knows the learner and designs an optimal training set. $n = 2$ for all $\varepsilon > 0$ by giving 2 opposite items ε away from each other.

1. Machine teaching: what can we expect from an ideal teacher?

2. The real world is not ideal.

2.1 Humans are teachers, not annotators. What can an ideal teacher do?

We make strong assumptions.

1. Teacher knows target model $\theta^* \in \mathcal{H}$.

2. Teacher can give training set D but not θ^* to learner.

   - Constructive teaching (can lie) $D \in \mathbb{D} = \bigcup_{n=1}^{\infty} (X \times Y)^n$.
   - Constructive teaching (honest) $D \in \mathbb{D} = (\bigcup_{n=1}^{\infty} X^n, Y = \theta^*(X))$.

- Pool-based teaching $D \in \mathbb{D} = 2^{\{(x_i,y_i)\}_{1:N}}$. Can only select examples from a pool.

3. Teacher knows learning algorithm/estimator/student $A : \mathbb{D} \to 2^{\mathcal{H}}$.
   - Ex. version space learner $A(D) = \{\theta \in \mathcal{H} : \theta(x_i) = y_i, 1 \le i \le n\}$.
   - Ex. regularized empirical risk minimizer $A(D) = \operatorname{argmin}_\theta \sum_{i=1}^n \ell(\theta, x_i, y_i) + \lambda \|\theta\|$.

The teacher is solving a problem in the space of training sets. Objective is size of training set.

$$\min_{D\in\mathbb{D},\theta^*\in A(D)} \|D\|_0 .$$

This is like the "inverse" of machine learning, find something in $A^{-1}(\{\theta^*\})$. An alternative view is from coding: the message is θ^*, the decoder is A, the language is $\mathbb{D}$. It's a strange decoder, maps the training set to the model.

A special case is classical optimal teaching (Goldman, Kearns 95). Further restrictions: A is a version space learner, often for $X, \mathbb{D}, \mathcal{H}$ finite.

The teaching dimension is

$$\min_{D\in\mathbb{D},\theta^*\in A(D)} \|D\|_0 \tag{5}$$

$$TD(\mathcal{H}) := \sup_{\theta^*\in\mathcal{H}} TD(\theta^*) \tag{6}$$

I'll move to other definitions depending on the learning algorithm.

Example 2.1: Think of hard-margin SVM. We want to teach a target model—a hyperplane. The teacher can do constructive teaching (pick arbitrary items). What's the smallest training set?

You can do it with 2. Place the examples such that the hyperplane is the perpendicular bisector of the hyperplane. So $TD = 2$ and $VC = d + 1$.

Example 2.2: Teach a d-dimension Gaussian to the maximum likelihood estimator.

In d dimension, choose tetrahedron vertices, $TD = d + 1$.

To teach linear learners (ridge regression, soft SVM, logistic regression), $A(D) = \operatorname{argmin}_{\theta\in\mathbb{R}^d} \sum_{i=1}^n \ell(\theta^T x_i,$ $\frac{\lambda}{2}\|\theta\|_2^2$.

TD depends on the loss function. For just learning the boundary, TD dimension is 1 for squared, hinge, and logistic loss: you can teach by a single example even with one positive example. This is because the student has a regularizer, which acts as a negative pull from the training example.

For ridge regression, the singleton set is

$$x_1 = a\theta^*, \quad y_1 = \frac{\lambda + \|x_1\|^2}{a}.$$

You can take any $a \neq 0$. Ex. to teach $\lambda = 1$ student the target $\theta^* = 1$, the teacher lies: $x_1 = 1, y_1 = 2$. The teacher needs to lie because the student is shrinking the estimate. Cancel out the regularizer.

The student doesn't know it's being taught; it just follows its algorithm. (What if the student expects to be taught?)

TD is like the "speed of light". Unavoidable effort in interactive machine learning: $n \geq TD$. Ideal teacher achieves $n = TD$. This can be much faster than active learning (ex. 2 vs. $\lg\left(\frac{1}{\varepsilon}\right)$). We must allow teacher-initiated items, unlike active learning. (You can't allow the computer to pick the items.)

We can also define learner risk, teacher effort, constraints (minimize subject to tolerance, or subject to budget). There is a Lagrangian form.

Suppose the teacher is restricted to a pool of items. Let the pool be n uniform items from $[-1, 1]$. Take the innermost pair and learner puts the boundary in the middle. Giving the whole pool achieves $\frac{1}{n}$. Because the learner is special, we can achieve $\frac{1}{n^2}$. Pick the most symmetric pair! Whp,

$$|\widehat{\theta} - \theta^*| = O(n^{-2}).$$

This is not training set reduction nor sample compression.

This is boring.

2.2 Most humans are not ideal teachers

I will continue using the 1-D example.

How do real humans teach? (Cover story: they choose their own θ^*. We told them what the learner is doing; we're not sure they understood it.) (One failure mode is when the teacher doesn't know what the student is doing.) (Instruction: please give the smallest training set.) People seemed to try to cover the hypothesis space.

1. 30% move linearly in space.
2. 40% crazy.
3. Few people gave 2 examples. Often they start with 2 examples, but add more to make sure.

The mixed-initiative algorithm: from $i = 1$ to TD, let human give (x, y) pair, to allow ideal human teachers. Afterwards the computer takes control and runs active learning starting from D until completion. (Good teachers should be allowed to do their thing. If the human isn't good at teaching, running active learning is good.)

Examples of teachers:

1. Seed teacher: provides one point per positive region.

   The concept class of interval can be hard to learn because it can be narrow. (You need random search. A seed teacher warm-starts the active learner, and reduces to binary search.)

2. Naive teacher: can be arbitrarily bad. (Ex. only give positive examples.)

Fallback guarantee: waste at most TD examples. $TD \leq AL$, so you get factor of 2 at most.

1. Human teachers: a significant fraction were unable to teach the concept (ex. only provide positive examples).
2. Active learning: 14 by binary.
3. Mixed initiative: preserve smart teachers, force those who can't succeed to be around 14–16.

Two ideas to help:

1. Control with mixed-initiative learning.
2. Educate with analogues: automatically generated optimal training sets for arbitrary $\theta' \in \mathcal{H}$: If threshold was \$19000, show \$19000 acceptable, \$19001 unacceptable.
3. Translate the teacher.

   Ex. Don't know the teacher's regularization constant. If human assumes wrong λ^w, then the learner learns wrong θ.

   If translator-in-the-middle knows λ^w, λ^*, it can translates examples $\widetilde{x} = \frac{xy}{x^2+\lambda^w}$.

   The translator is the meta-algorithm.

   This is not realistic, but may be useful when we know roughly what the teacher is teaching (ex. mixture of Gaussians), while in reality the student is running something else (ex. SVM).

`http://pages.cs.wisc.edu/~jerryzhu/machineteaching/`

Recruit people with teaching experience? We asked if they were teachers or parents, but didn't see correlation. Tasks may be too artificial.

Role of adaptivity?

Imperfect memory?

Build from psychology: models human might assume of human student?

3 Interactively Learning Robot Objective Functions (Anca Dragan, UC Berkeley)

Abstract: Inverse Reinforcement Learning enables robots to learn objective functions by observing expert behavior, but implicitly assumes that the robot is a passive observer and that the person is an uninterested expert. In this talk, I will share our recent works in making this learning process interactive, where humans teach rather than provide expert demonstrations, and robots act to actively gather information to showcase what they've learned.

Formally specifying objective functions is hard. Ex. asking a vacuum to suck dust: the robot pours dust on the floor and sucks it.

How do we get a robot to learn objective functions that are implicit in people's heads but hard to write down?

A robot car can drive according to passenger preferences, and predict how other drivers will drive.

The framework is inverse reinforcement learning. The person implicitly knows parameters of θ but can't explicate or write it down, but can demonstrate behavior according to θ:

$$\mathbb{P}(u|\theta, x_0) \propto e^{U_\theta(x_0,u)}.$$

The action can be the person acting or operating the robot.

Arobot can take this evidence to update its belief about θ.

$$b'(\theta) \propto P(u|\theta, x_0)b(\theta)$$

This is implicitly making 2 assumptions we don't want to make.

1. The robot is a passive observer during the learning phase.

   Robots are actuated agents. Can we leverage this to improve learning?

   Desiderata 1: Robots should leverage their action to improve learning.

2. We're asking people to behave like uninterested experts, optimally with respect to θ.

   This is like the machine having to learn gymnastics from learning Gabby Douglas (Olympic level); instead it's easier to learn from a coach.

We formulate learning as a cooperative 2-player game.

This gives the person an interest in the robot's learning.

In cooperative inverse RL, the human has an objective function $U_\theta(x_0, u_R, u_H)$ and the robot has an objective function $U_R(x_0, u_R, u_H)$. Go a step further and equate the utility function. The robot acts to improve estimation and the human has an incentive for the robot to improve its estimation, learn the right θ.

The challenge is tractability: how to solve this?

Inverse RL is one approximation: fix the human's policy to expert demonstration π_H. The robot assumes no more human actions (it won't get more information) and chooses the best response $\pi_R = br(\pi_H)$.

Setting it up as the game allows us to think about other approximations that take us closer to the desiderata.

First: how to model as a game.

In autonomous driving, estimate θ because we assume the person acts according to it. It's important to be able to do this as an ongoing interaction because people drive in different ways: aggressive, distracted, defensive, attentive.

All users react in almost the same way. If a robot just does inverse RL, it doesn't get a lot of information. If the robot tries to merge left, it allows the other person to go first. This is typical behavior; they're defensive. That's not what humans do. I have to stick myself in front of people and force them to allow me in.

"Google's driverless cars run into problem: cars with drivers." The human drivers kept inching forward looking for the advantage—paralyzing Google's robot.

We thought: what if instead of just observing actions, allow the robot to act, and the human's actions depend:

$$\mathbb{P}(u_H|u_R, \theta, x_0) \propto e^{U_\theta(x_0, u_R, u_H)}.$$

Incentivize information gain. Robot updates belief $b'(\theta) \propto P(u_H|u_R, \theta, x_0)b(\theta)$ and optimizes

$$u_R^* = \text{argmax}_{u_R} \mathop{\mathbb{E}}_{\theta}[H(b) - H(b')] = \text{argmin}_{u_R} \int H(b')b(\theta)\, d\theta..$$

Choose actions to aid learning. Trade off exploitation and information gain. Learning more about the other person helps make better decisions.

This is a POMDP but we can't solve them well in continuous state and action space.

This is used in navigation, localization, etc. We're doing it not over physical state but human internal state.

Simplification: Assume optimal observation (rather than draw from distribution)

$$u_H = \text{argmax}\, U_\theta(x_0, u_R, u_H).$$

where $b'(\theta) \propto \mathbb{P}(u_H|u_R, \theta, x_0)b(\theta)$. Use quasi-Newton method. Use implicit differentiation.

Replace integral by sum,

$$\text{argmin}_{u_R} \sum_{\theta} H(b')b(\theta).$$

Offline do clustering of user types. Identify type of user online.

When merging, the robot car nudges in. If the person is distracted, go back. If the person is attentive, merge. At an intersection, inch forward. See one behavior from distracted (continue) and another from attentive users (stop). If distracted, backs up.

Data from a simulator.

Information gathering improves estimation.

In order to interact properly, we can't treat them as obstacles; we have to interact with them.

The robot autonomously generates strategies that we normally hand-code.

Right-of-way? We haven't put in those constraint.

How does the robot do with a copy of itself? That would be different.

There's another side of learning how the passenger wants to drive; make queries to the expert. Generate trajectories, which would you prefer?

$$\text{argmax}_{x_0, u_R^1, u_R^2} \mathop{\mathbb{E}}_{\theta}[H(b) - H(b')].$$

The person (teacher) tries to figure out how the robot is learning, and responds differently from just an expert demonstration:

$$\pi_H \neq br(br(\pi_H)).$$

Ex. Robot doesn't know the relative value of peaks of the optimization function. Expert goes straight to the closest peak. The best response visits both, gives robot evidence that the other peak is also good.

There's a sweet spot: if the person is optimal, there's little you can do. If there's enough flexibility, people can take interesting teaching actions.

Looking at the problem this way speeds up learning.

Safety vs. information? Adapt tradeoff with safety over time. Reason about value of information you gain.

Other relaxations? Game makes a lot of assumptions about person having perfect knowledge. This is not the case. Person has own beliefs. I don't think iterating on best responses buys us that much.

4 Words, Pictures, and Common Sense: Learning by playing (Devi Parikh, Georgia Institute of Technology)

Abstract: Wouldn't it be nice if machines could understand content in images and communicate this understanding as effectively as humans? Such technology would be immensely powerful, be it for aiding a visually-impaired user navigate a world built by the sighted, assisting an analyst in extracting relevant information from a surveillance feed, educating a child playing a game on a touch screen, providing information to a spectator at an art gallery, or interacting with a robot. As computer vision and natural language processing techniques are maturing, we are closer to achieving this dream than we have ever been.

In this talk, I will present two ongoing thrusts in my lab that push the boundaries of AI capabilities at the intersection of vision, language, and commonsense reasoning.

Visual Question Answering (VQA): I will describe the task, our dataset (the largest and most complex of its kind), our model, and ongoing work for free-form and open-ended Visual Question Answering (VQA). Given an image and a natural language question about the image (e.g., "What kind of store is this?", "How many people are waiting in the queue?", "Is it safe to cross the street?"), the machines task is to automatically produce an accurate natural language answer ("bakery", "5", "Yes"). We have collected and recently released a dataset containing ¿250,000 images, ¿760,000 questions, and $\approx$ 10 million answers. Our dataset is enabling the next generation of AI systems, often based on deep learning techniques, for understanding images and language, and performing complex reasoning; in our lab and the community at large.

Learning Common Sense Through Visual Abstraction: Common sense is a key ingredient in building intelligent machines that make "human-like" decisions when performing tasks – be it automatically answering natural language questions, or understanding images and videos. How can machines learn this common sense? While some of this knowledge is explicitly stated in human-generated text (books, articles, blogs, etc.), much of this knowledge is unwritten. While unwritten, it is not unseen! The visual world around us is full of structure bound by commonsense laws. But machines today cannot learn common sense directly by observing our visual world because they cannot accurately perform detailed visual recognition in images and videos. This leads to a chicken-and-egg problem: we would like to learn common sense to allow machines to understand images accurately, but in order to learn common sense, we need accurate image parsing. We argue that the solution is to give up on photorealism. We

propose to leverage abstract scenes – cartoon scenes made from clip art by crowd sourced humans – to teach our machines common sense.

4.1 Common sense from cartoons

Classical computer vision: describe what is going on in images.

Hard: "A man is *rescued* from his truck that is hanging *dangerously* from a bridge."

This requires commonsense reasoning.

Where do we gather this common sense from? We can use all the text on the web.

There is reporting bias in what people write about online. Ex. people inhale 6 times more often than they exhale, and get murdered 17 times as often. People have heads to gallbladders 1085:1.

Is there a way to watch the world around us and use the structure to learn commonsense knowledge. Ex. two people conversing in front of a blackboard vs. two people standing in front of a blackboard. Gaze helps us differentiate.

This is hard:

- lack visual density: hard to find two images where just gaze differs.
- Annotations are expensive
- Why need annotations? Computer vision doesn't work well enough.

There is a chicken-and-egg problem: learn common sense vs. improve image understanding

Do we need photorealism? No, it lies in semantic content of scene. We introduce Mike and Jenny with variety of objects. They can have different expressions and poses. We can get people to create visual data for us.

"Mike fights off a bear by giving him a hotdog while Jenny runs away."

People make different images: consistent: Mike facing bear with hot dog, Jenny running in opposite direction.

We took 1000 classes, 10 MTurkers for each.

Each image is trivially annotated. Which visual features are important for semantic meaning? Which words correlate with specific visual features?

We have full control over density of sampling.

We can take an input description (fresh sentence, not in training set), turn them into tuples, and automatically generate images. (Zitnick, Parikh, Vanderwende, ICCV2013)

Based on sentences, update on potential of how likely objects are likely to be there. Ex. "raining" reduces probability of sun relative to prior.

Goal is to use this abstract world to learn commonsense knowledge that generalizes to realistic examples.

Another example (fine-grained interactions): Learning what it looks like to interact with each other. Ex. "Person 1 is dancing with Person 2", walking with, holding hands with, talking with, jumping over, etc. We keep track of all intermediate stages but haven't used it.

Could we use this as training data to help learn real-life interactions (zero-shot learning). We can do statistically significant above chance. You would get similar performance if you train on 2–3 real images vs. if you train on 50 generated images.

This is just a 2-D canvas, while real images have all of that. If you want to improve this, make 3-D canvas.

We initialize poses randomly, hope that people go to the local minimum rather than all converge to the same thing. For Mike and Jenny, we give access to a randomly generated sample of clip art.

4.2 Visual question answering

What color are her eyes? What is the mustache made of? Is this a vegetarian pizza? Recent years: improve 55 to 68%. Human accuracy is 83%.

There is a heavy language bias. "Do you see a...?" is right 87% of the time. When collecting the dataset, people are asking while looking at the image. People ask "Is there a clock?" more often when there is a clock. This hindered progress on binary question.

Sometimes there is a bias towards no. "Should this person be doing...?" The framing of the question gives away information about what the answer is! Problem with using as a benchmark!

We want to rectify this, by removing language priors.

"Is there a place to sit other than the floor?" Modify the image as little as possible to make it true. Now we have 2 complementary scenes which are globally very similar but differ in the answer. Guessing can't do well here.

Only if model answers correctly for both in pair does it get a point. Answering based on question alone you get 0 accuracy. Training on holistic features gets 3.2%. Training on balanced set gets 23.13%. Making the model more sophisticated (parsing, explicit alignment), this attention-based model gets 9.84%, 34.73% in the unbalanced/balanced case.

Here everything has (primary, verb, secondary object) structure. Attention model is learned from training data. It decides how sharp the attention map should be.

Summary:

- Learn by playing
- Fully annotated visual data.
- Allow full control over distribution and density of data. You're not at the mercy of the distribution of Google, flickr...

4.3 Reasoning by imagination

We have text where it helps to imagine the scene behind the text.

- man holds meal
- tree grows in table.

Fill-in-the-blank: Mike is having lunch when he sees a bear. (Mike tries to hide.)

Visual paraphrasing: Are these two descriptions describing the same scene.

- Jenny was going to throw her pie at Mike
- Jenny is very angry. Jenny is holding a pie.

Ex. For multiple choice, generate scene from each answer. Given text with correct answer and scene should score higher than incorrect ones. What's relevant are semantics. This helps.

Ex. Plausibility of "tree grows in table." Find support for description in images *and* database of text.

You should be able to learn from text embeddings, so similarity in new representation space matches.

4.4 Understanding visual humor

People rate humor.

Ask maching to: Recognize humor. Add humor. Remove humor.

Ask people to create funny and nonfunny scenes. What part of scene is funny? Replace with soemthing else so it stops being funny. Which objects are contributing to humor? Replace with something semantically meaningful. Ex. replace with butterflies... Indoor: potted plant.

Test: which scene is less funny.

Turing test: human's funny version vs. algorithm's funny version. Algorithm wins 28% of the time. When there is a lot of something the machine thinks it's funny.

Often we weren't laughing, so we ask them why it's funny. "This terrified woman's home is being invaded by mice while the cat sleeps."

Visual abstraction for...

- studying mappings between images and text.
- zero-shot learning
- studying image memorability, specificity, visual humor
- learn common sense
- Rich annotation modality: ask for descriptions, ask for scenes, show scene and ask for modifications, perturb scene and ask for descriptions.

Study high-level image understanding tasks without waiting for lower-level vision tasks to be resolved.

Further push learning by playing modality.

In images, "look at" and "eat" are close in visual space. These are not close in word embedding space! Word2vec is now informed by this! (Note: not symmetric.)

5 Stochastic variance reduction methods for policy evaluation (Lihong Li)

Abstract: Policy evaluation is a crucial step in many reinforcement-learning problems, which estimates a value function that predicts long-term value of states for a given policy. In this talk, we present stochastic variance reduction algorithms that learn value functions from a fixed dataset, which is shown to have (i) guaranteed linear convergence rate, and (ii) linear complexity (in both sample size and feature dimension), under the condition of linear function approximation and possibly off-policy learning as well as eligibility traces. In particular, we transform the policy evaluation problem into an empirical (quadratic) saddle-point problem and apply stochastic variance reduction methods in the primal-dual space. Interestingly, the algorithms converge linearly even when the quadratic saddle-point problem has only strong concavity but no strong convexity. Numerical experiments on random MDPs and on Mountain Car demonstrate improved performance of our algorithms.

This is not common in RL work because it uses recent advances in optimization. The techniques can be very powerful.

A robot takes actions a_t to affect the world and gets immediate reward r_t.

The value of a state is the expected long-term return from state s,

$$V^\pi(s) := \mathbb{E}\left[\sum_{t=1}^{\infty} \gamma^{t-1} r_t | s_1 = s\right].$$

In the end we care about policy optimization.

A crucial step is to evaluate a fixed policy (policy evaluation).

For convenience, drop π and a for the rest of the talk. We focus on PE with linear function approximation, and a batch setting where data is fixed (experience replay).

We showed first-order algorithms with linear convergence: total time complexity $O\left(nd \ln\left(\frac{1}{\varepsilon}\right)\right)$ where n is sample size, d is dimension, and ε is optimization precision. Previous complexity is $O(nd^2)$ or $O(n/\varepsilon)$.

Key technical ingredients are

- Saddle-point reformulation of policy evaluation. (Usually it's described as an fixed point of an iteration.) This seems to make it more complicated, but we can apply optimization.
- Stochastic variance reduction based on SDVRG and SAGA.
- Eigenvalue analysis of corresponding matrices.

All results can be extended to eligibility traces and off-policy RL.

5.1 Problem setup

We model as Markov reward process (there is no action) $M = \langle S, P, R\rangle$: Markov chain with reward function. S is finite set of states, $\mathbb{P}(s'|s)$ is transition probability, $R(s)$ is expected one-step reward.

Value funciton is $V(s) = \mathbb{E}[r_1 + \gamma r_2 + \gamma^2 r_3 + \cdots | s_1 = s]$. Bellman equation says

$$V = R + \gamma PV.$$

V is a fixed point to the linear operator $X \mapsto R + \gamma PX$.

In linear value function approximation, we want an approximation

$$\widehat{V}(s) = \phi(s)^T \theta$$

(or $\hat{V} = \Phi\theta$). Here $\phi(s)$ is a d-dimensional feature vector and $\theta \in \mathbb{R}^d$ has parameters to optimize.

The objective function: Data is a trajectory of n steps, $D = (s_1, r_1, \ldots, r_n, s_{n+1})$ with $\phi_t = \phi(s_t)$. Training target is not given, unlike regression problemn.

We use empirical MSPBE (mean square projected Bellman error)

$$\left\| \widehat{V} - \Pi(R + \gamma P\widehat{V}) \right\|_{\Xi}^2.$$

Precisely...

$$\theta^* = \text{argmin}_\theta \frac{1}{2} \|A\theta - b\|_{C^{-1}}^2 + \frac{\rho}{2} \|\theta\|^2 \tag{7}$$

$$A_t = \phi_t(\phi_t - \phi_{t+1})^T \tag{8}$$

$$A = \frac{1}{n} \sum_t A_t \tag{9}$$

$$b_t = r_t \phi_t \tag{10}$$

$$b = \frac{1}{n} \sum_t b_t \tag{11}$$

$$C_t = \phi_t \phi_t^T \tag{12}$$

$$C = \frac{1}{n} \sum_t C_t. \tag{13}$$

Important is that we have an objective function.

5.2 Saddle-point formulation and algorithms

The objective does not have the sum-over-data structure. It's not straightforward to decompose into individual examples. Direct optimization is not easy, often requiring $O(d^2)$ complexity. GTD2 (Sutton et al 2009) has $O(d)$ complexity but convergence is slow.

The conjugate function of $f(w) = \frac{1}{2} \|w\|_C^2$ is

$$f^*(u) := \sum (u^T w - f(w)) = \frac{1}{2} \|u\|_{C^{-1}}^2.$$

Rewrite MSPBE as

$$\min_\theta \left[\frac{1}{2} \|A\theta - b\|_{C^{-1}}^2 + \frac{\rho}{2} \|\theta\|^2 \right] = \min_\theta \max_w \underbrace{\left[w^T(b - A\theta) - \frac{1}{2} \|w\|_C^2 + \frac{\rho}{2} \|\theta\|^2 \right]}_{L(\theta, w)} \tag{14}$$

L is convex-concave, so the we can find the minimax. We can decompose

$$L(\theta, w) = \frac{1}{n} \sum_t L_t(\theta, w).$$

Use existing optimization, like primal dual batch gradient.

Initialize (θ, w). For $m = 1 : M$, update by gradient

$$\begin{pmatrix} \theta \\ w \end{pmatrix} - = \begin{pmatrix} \sigma_\theta & \\ & \sigma_w \end{pmatrix} B(\theta, w) \tag{15}$$

$$B(\theta, w) := \begin{pmatrix} \nabla_\theta L \\ -\nabla'_w L \end{pmatrix} = \begin{pmatrix} \rho\theta - A^T w \\ A\theta - b + Cw \end{pmatrix}. \tag{16}$$

Per-iteration cost is $O(nd)$. This enjoys linear convergence.

But is it not practical because it is linear in n.

We can do a stochastic version. Randomly sample a data point and compute a stochastic gradient.

$$\begin{pmatrix} \theta \\ w \end{pmatrix} - = \begin{pmatrix} \sigma_\theta & \\ & \sigma_w \end{pmatrix} B_{t_m}(\theta, w) \tag{17}$$

$$t \sim U[1, \ldots, n] \tag{18}$$

$$B(\theta, w) := \begin{pmatrix} \nabla_\theta L_t \\ -\nabla'_w L_t \end{pmatrix} = \begin{pmatrix} \rho\theta - A_t^T w \\ A\theta - b_t + C_t w \end{pmatrix}. \tag{19}$$

This recovers GTD2. Per-iteration cost is $O(d)$ but convergence is sublinear. The problem is that B_{t_m} has high variance. We reduce the gradient by SVRG

$$\begin{pmatrix} \theta \\ w \end{pmatrix} - = \begin{pmatrix} \sigma_\theta & \\ & \sigma_w \end{pmatrix} B_{t_m}(\theta, w, \widetilde{\theta}, \widetilde{w}) \tag{20}$$

$$t \sim U[1, \ldots, n] \tag{21}$$

$$B_t(\theta, w, \widetilde{\theta}, \widetilde{w}) = B_t(\theta, w) - B_t(\widetilde{\theta}, \widetilde{w}) + B(\widetilde{\theta}, \widetilde{w}). \tag{22}$$

We need to maintain $\{\widetilde{\theta}, \widetilde{w}, B(\widetilde{\theta}, \widetilde{w})\}$, update periodically. In the long run, $B_t(\theta, w, \widetilde{\theta}, \widetilde{w}) \approx B(\widetilde{\theta}, \widetilde{w})$. Per-batch is $O(d)$. onvergence is linear.

SAGA also reduces variance. Idea is similar but details are different. It doesn't need periodic batch graident updates, but has to maintain gradient for each datum (extra space $O(n)$).

Extend to off-policy PE (when data sampled from different distribution) and PE with eligibility traces (smooth the gap between MSPBE and MSE). The difference is how (A, b, C) are formed.

5.3 Complexity bounds

Summary:

1. LSTD (least squares temporal difference) $O(nd^2)$ (exact matrix inversion)

2. GTD2 $O\left(\frac{d\kappa_1}{\varepsilon}\right)$.

3. PDBG $O\left(nd\kappa_2 \ln\left(\frac{1}{\varepsilon}\right)\right)$.

4. SVRG/SAGA $O\left(nd\left(1+\frac{\kappa_3}{\ln}\left(\frac{1}{\varepsilon}\right)\right)\right)$].

The κ_i are algorithm-specific condition numbers.

$L(w,\theta) = w^T(b - A\theta) - \frac{1}{2}\|w\|_C^2 + \frac{\rho}{2}\|\theta\|^2$. Step sizes are $(\sigma_\theta, \sigma_w$. Let $\beta = \frac{\sigma_w}{\sigma_\theta}$. Optimality conditions give

$$\Delta_{m+1} = (I - \sigma_\theta G)\Delta_m \tag{23}$$

$$\Delta_m := \begin{pmatrix} \theta_m - \theta^* \\ \beta^{-0.54}(w_m - w^*) \end{pmatrix} \tag{24}$$

$$G = \begin{pmatrix} \rho I & -\sqrt{\beta}A^T \\ \sqrt{\beta}A & \beta C \end{pmatrix}. \tag{25}$$

Eigendecomp $G = Q\Lambda Q^{-1}$ is diagable for large enough β. Get

$$\Delta_{m+1} = (I - \sigma_\theta Q\Lambda Q^{-1})\Delta_m \tag{26}$$

$$Q^{-1}\Delta_{m+1} = (I - \sigma\Lambda)Q^{-1}\Delta_m. \tag{27}$$

Capture convergence by potential function $P_m := \|Q^{-1}\Delta_m\|^2$,

$$P_{m+1} \le \|I - \sigma_\theta\Lambda\|^2 P_m.$$

Proper σ_θ gives exponential decay of P_m. This gives linear convergence of θ_m to θ^*.

$$\|\theta_m - \theta^*\| \le \|Q\|^2 \left\|Q^{-1}\Delta_m\right\|^2 = O(2^{-m}).$$

We have to bound eigenvalues of Q.

5.4 Experiments

We compare

1. TD (experience replay)

2. GTD (experience replay)

3. PDBG

4. SVRG/SAGA

5. LSTD

We experiment on random MDPs (400 states, 200 actions, $\gamma = 0.95$, features $d = 200$, $n = 100000$) and mountain car (2d, 3 actions, $\gamma = 0.9$).

Previous work:

- stochastic variance reduction for convex optimization,
- saddle-point optimization (Balamurugan, Bach 2016).

  Require proximal mappings that are expensive to compute in PE, and strongly convex-concave objectives (we only require strong concavity in dual).
- Gradient-based TD (Sutton et al. 2009). This is derived from very different principles and have slow convergence.
- Incremental LSTD, unknown convergence rate.

5.5 Conclusion

We give a saddle-point formulation of batch policy evaluation, first-order algorithms with linear convergence rate, and have promising experimental results on benchmarks.

Future directions

- extend to nonlinear value-function approximation
- extend to control case (policy optimization)
- application of modern optimization techniques to RL.

6 Discussion 2/13/17

EBrunskill: We want to bridge different fields. RL and active learning communities are largely disjoint but they tackle common problems.

SDasgupta: Traditional divide between learning concepts (what is a giraffe) and skills (ride bicycle). Are there math formalisms which allow us to unite the two? This is why control-related work and conceptual active-learning work have been done separately.

JZhu: Relation between optimum teaching and RL: how do you teach a RL agent? If the learner is not a simple batch classifier, but something more complex, how does that change the problem formulation and solution?

R: Minimal contrast pair, smallest change to picture to change classification cf. learning with 2 examples.

JZ: What kind of learner takes that as important teaching examples?

SD: Representation teaching. Teaching doesn't have to be limited to examples and labels. Everyone understands what those mean. Features and explanations: one party doesn't understand the other any longer.

TMitchell: Tabular rasa assumptions in theory. Zhu's talk assumed the learner was a tabular rasa learner. How to frame theoretical questions that can be studied in this messy world where there are people. Tabular rasa is one of the big question marks: can we frame theory questions that are close enough to reality?

JZ: Baby step to weaken assumption is to give uncertainty to human student.

SD: Learning theory has focused on a single concept and the learner doesn't need to know anything else. In principle we can imagine a hierarchy, DAG of concepts; the learner is somewhere along the way. Teacher should understand what learner already has. Should be quite easy to formulate math model where there are many concepts and the learner knows some.

EB: cf. education, knowledge graph.

Issue when doing one-shot, minimal learning (esp. in interactions with human) is very little data. having a whole lot of experience. Policies of behavior are great when you have experience and terrible when you don't. There is disconnect between minimizing amount of data needed and having enough of it to optimize policy.

JZ: Human is not a great teacher.

When we explain to someone else, we make commonsense assumptions. We expect that you will spread info to rest of state space where it's applicable.

JZ: Educate the human teacher.

R: "Make a PB and J sandwich" game.

EB: cf. picking up dust.

SD: We have rich spaces of concept classes, how difficult we expect the problem to be. What are categories of agents we can learn to control easily, slightly more complicated... that would give a nice handle on how much linguistic communication is needed to be able to control these agents.

TM: Think of a policy of robot, say, actions in discrete space. It is a function. In that sense, there is no difference even though it feels like there are.

EB: Policy search: often formulate in terms of VC dimensions. dimension. There may be other forms.

SBen-David: View as hierarchies of agents. Hierarchies of behavior?

EB: cf. Anca. Control community. What assumptions do we make? Stochastic, 1-step approximation. Types of approximations.

SD: What about agnostic case? In the teaching cases, it seems more reasonable to assume there is a perfect concept. We are perfect categorizers. Of course there must be a perfect categorizer for zebra, antelope, because we do it.

: Can still have model misspecification problem.

JZ: Ideal teacher knows the world is complex, but I know you're limited (ex. linear classifier), and help you get the linear separated. Meta-level teacher: I want to tell you to expand your hypothesis space, can't just use linear classifier.

List of topics

- RL +/or AL
- Learning concepts vs. learning skills
- Optimum teaching and RL
- Representation teaching
- Theoretical framing, questions for nontabular rasa setting
- Min data needed for RL vs. typical data requirement

- Categoris of agents
- Imperfect concept classification, model misspecification

7 Interactive clustering (Pranjal Awasthi, Rutgers University)

Abstract: Clustering is typically studied in the unsupervised learning setting. But in many applications, such as personalized recommendations, one cannot reach the optimal clustering without interacting with the end user. In this talk, I will describe a recent framework for interactive clustering with human in the loop. The algorithm can interact with the human in stages and receive limited, potentially noisy feedback to improve the clustering. I will present our preliminary results in this model and mention open questions.

Most theoretical work on interaction is in supervised learning (PAC learning, active learning, membership queries, equivalence queries). This work looks at interaction for unsupervised learning. Can interaction be useful?

In clustering, given n objects, we want to partition them into k disjoint clusters. We want to design provably good algorithms.

The classical approach is to define an objective function like k-means

$$\sum_i \min_j \|x_i - c_j\|^2 .$$

This requires the objective function to actually be minimized with the actual clustering.

Another approach is to use a generative modeling approach: data is generated from a mixture of Gaussians, stochastic block models, etc. We have optimal recovery under these assumptions, but this is not robust; we can't say much if the modeling assumption is wrong.

Both approaches aim to remove ambiguity from the problem. Can we remove ambiguity with a human-in-the-loop? In many scenarios, the best clustering is ambiguous unless the end user is involved, ex. personalized recommendations.

Approaches to interactive clustering include:

1. constrained k-means/median (constrained given by users) (cf. semisupervised learning)
2. pairwise clustering (asking the user whether pairs are in the same cluster or not)

We give a different model.

- What can we do with minimal feedback?
- Can the model be made noise robust?
- Is the model practical?

Interactive clustering was introduced by Balcan and Blum 2008. This is inspired by equivalence query model for learning. There is an algorithm and teacher. The algorithm proposes function f_1, and gets a counterexample x such that $h_1(x) \neq h^*(x)$. Repeat until the algorithm proposes h^*.

We do this in the clustering setting. The algorithm proposes clustering $C_1, \ldots, C_k$, gets limited/coarse feedback, and repeats until it proposes the true clustering.

We want as few rounds of interaction as possible.

To specify this model, I need to say what the right feedback is. It should be

1. easy to provide form teacher's point of view
2. be informative

The two kinds of feedback are

1. split: Split(C_i) if C_i contains points from two or more clusters of the ground truth. The teacher does not have to say why.
2. merge: Merge(C_i, C_j) if C_i and C_j are subsets of the same cluster in the ground truth. (In particular they are pure.)

Note that "cannot link" can be interpreted as a split, but "must link" cannot be interpreted here.

The goal is to design an efficient algorithm that attains ground truth in small number of rounds.

- Given n points, we can cluster them in at most $n - k$ queries: start with each point in own cluster and merge points when merge feedback is issued.

  The teacher is just stepwise saying what the clustering is.

- Without further assumptions this is best possible.

- What if the clusters are nice?

Suppose that membership in cluster C_i^* can be decided by a boolean function $f_i()$, $f_i \in H$. For example, H can be linear separators, rectangles...

Theorem 7.1. *Suppose f_i's belong to class H. Then we can cluster using $O(kd \ln n)$ queries where $d = VCdim(H)$.*

The algorithm is similar to version space algorithms: The version space is all valid clusterings.

1. Start with $VS = \{\text{all valid clusterings}\}$.
2. Choose a clustering from VS and present it to the teacher. Choose by:
   (a) Find a maximal set of points S such that more than half of clusterings in VS put S in a single cluster.

   (b) Output S as a cluster. Repeat until points are clustered.

3. On feedback, delete all clusterings from VS that are inconsistent with feedback.

This works for every concept class. We can get efficient algorithms for specific function classes. The algorithm does not know k; it could generate more clusters.

Claim: After each request, we get rid of at least half of VS.

- On Split(C_i), delete all clusterings that put C_i in a single cluster.
- On Merge(C_i, C_j), delete all clusterings that do not put $C_i \cup C_j$ in a single cluster. (This must be more than half, otherwise $C_i \cup C_j$ is the maximal set.)

In practice, show random subsets of each cluster.

Another model: the ground truth $C_1^*, \ldots, C_k^*$ is stable. Assume average stability, which says that on average, points prefer their own clusters over other: For $i \neq j$, $A \subseteq C_i^*$, $A' \subseteq C_j^*$.

$$S_{avg}(A, C_i^* \backslash A) \geq S_{avg}(A, A')$$

Theorem 7.2. *If average stability holds, k queries suffice.*

Algorithm:

1. Construct average linkage tree T_{avg}. This is a minimum spanning tree using average distance (to merge children).
2. Present the root node as a single cluster.
3. On split feedback, replace cluster with children in T_{avg}.

Claim: C^* is laminar with respect to T_{avg}: each cluster is present as a subtree in T_{avg}. Otherwise average stability is violated. The teacher will never issue merge requests.

These results are nice but don't lead to practical algorithms.

- They don't account for noise in human resposne.
- They will present very bad clusterings to the user in order to make quick progress.

  We want to show clusterings in a smooth fashion.

Change the model:

- η-noisy merges: If the user asks to merge C_i, C_j, then $\geq 1 - \eta$ fraction of both the clusters belong together in the ground truth.
- Local algorithm: The algorithm cannot choose the initial clustering. After a round of feedback, the algorithm can onlhy modify the clusters involved in the feedback. This gives better user experience.

  This is useful if you already have a system in place, in use.

Measure progress in terms of how far it is from the actual clustering.

Theorem 7.3. *If C^* is stable, then we can find it using $O\left(\frac{\varepsilon_0+k}{1-\eta}\ln n\right)$-queries where ε_0 is the error of the initial clustering.*

Algorithm:

1. Construct average linkage tree T_{avg}.
2. On split(C_i), find the first node where C_i gets split and go to its children.
3. On merge(C_i, C_j), find the deepest node containing enough points from C_i, C_j and carve out the pure portion from this node.

Can we deal with extremely high noise?

- Unrestricted merges: if user asks to merge C_i, C_j then at least one point from both the clusters belongs together in ground truth.

Theorem 7.4. *If C^* is stable and request is chosen randomly, then can find it using $O((\varepsilon_0+k)^2\ln n)$ rounds with high probability.*

We chose image segmentation. This is not really the right problem, but it's easy to test. We posed this as a problem on Mechanical Turk. The user can point to regions to merge or split. This model has the potential to lead to good algorithms.

There are many technical problems:

- Algorithm is not scalable, requires average linkage tree.
- We need strong stability assumption to prove theorems.
- In other applications, presenting the entire clustering is not practical.
- What if user is willing to provide more feedback?

Main message: interaction can be useful/critical for unsupervised learning.

QA:

After each step, ask user: Are you happy with segmentation, or do you want to provide more feedback. Some workers were trying to fine-tune boundaries.

Show the same picture to multiple people: Did they come to the same answer? In a global sense, they come to the same answer. On boundaries there is some disagreement. In crowdsourcing scenario, take a vote?

Your algorithm wants clusters with respect to underlying similarity or distance. Are users happy with the metric itself? Is there a way to tell your algorithm that?

8 Crowdsourcing and machine learning (Adam Kalai, Microsoft Research New England)

Abstract: People understand many domains more deeply than today's machine learning systems. Having a good representation for a problem is crucial to the success of intelligent systems. In this talk, we discuss recent work and future opportunities for how humans can aid machine learning algorithms. Beyond simply labeling data, the crowd can help uncover the latent representation behind a problem. We discuss recent work on eliciting features using active learning as well as other aspects of crowdsourcing and machine learning, such as how crowdsourcing can help generate data, raise questions, and assist in more complex AI tasks.

The crowd is still useful today. One basic application is labeling data. They can caption, but can also label things much better and more efficiently.

Captioning the Obama picture: "Barack Obama plays a clever prank on someone." vs. the best neural net, "Barack Obama smiles and watches man weigh himself on locker room."

The crowd still understand the real world better than ML.

An example: word embedding capture semantics, Man:King::Woman:Queen.

There are gender biases in word embeddings.

- he:blue::she:pink (*)
- he:brother::she:sister
- he:doctor::she:nurse (*)
- he:programmer::she:homemaker (*)
- he:kidney stone::she:pregnancy
- he:realist::she:feminist
- he:kidney stone::she:burn

Question: which analogies reflect stereotypes? (*)

Demo: put in words and the algorithm finds related words split by gender.

8.1 Crowdsourcing judgments

There is a lot of work on getting labels from crowdsourcing. I call it "judgment" rather than labels because they might not agree. There is a common EM algorithm and ways to improve; I don't cover this.

People give different answers. Get people to build a classifier to estimate how much a person weights from a picture. Different people have different judgments. Suppose we have a budget: we can ask people questions. Asking other questions like "is pretty", "is male", "has good posture" helps predict weight. How many people should we ask for each question? Ex.

$$\widehat{y} = 20 + (\text{numeric weight}) - 15\,(\text{is pretty}) + 23\,(\text{is male}) + 7\,(\text{has good posture})\,.$$

Average weight estimate over 5 people, prettiness over 2 people, gender over 1 person, etc.

1. Choose representation for data. (Use crowd) It could be feature-based or similarity-based.
2. Label data set according to that representation (Use crowd)
3. Run ML algorithms on labeled data.

Make this whole process "crowdomatic."

8.2 Crowdsourcing representation

Input is images $X = \{x_i\}$. Crowdsource queries are: is x_i more like x_j or x_k?

This works for any dataset that the crowd can understand.

Ask: is floor tiling 1 more like tiling 2 or tiling 3? At the beginning randomly choose question; later choose them adaptively. We get an embedding of items in $\mathbb{R}^d$.

You can use this on any dataset, as way to search (shoppjng for products, or find researcher by face).

How about with letters, "is e more like s or u?" We benefit from diversity of crowd. Semantic features include vowel/constant (there is a large margin separator), tall/short, and alphabetical order

Nearest neighbors show reasonable results.

The adaptive algorithm is as follow. Ask Turks random triples. Then loop: fit K to all data so far, and ask Turk most informative triples.

We model by

$$p_{ijk}^K = \mathbb{P}\left(\text{person says } x_i \text{ more like } x_j \text{ than } x_k\right) = \frac{\delta_{ik}^K}{\delta_{ij}^K + \delta_{ik}^K}.$$

The MLE is nonconvex but in the realizable case the SGD will converge to the correct model.

We use probabilistic model and information gain to decide informativity. 60% more triples are required to achieve the same performance with random triples compared to adaptive triples.

"Closest to the margin" queries are those we know are 50/50. This is often useful.

A standard model is

$$\mathbb{P}\left(x_i \text{ is more like } x_j \text{ than } x_k\right) = \frac{e^{\alpha K_{ij}}}{e^{\alpha K_{ij}} + e^{\alpha K_{ik}}}.$$

This has convex MLE but most informative triples are "extreme", and not as useful as the probabilistic model.

In a dream world we can learn the matrix with $n \log n$ queries.

You can think of this as an online problem too.

Measuring information: Start with uniform prior over the tiles. Pretend A is an unknown tile. Which question could we ask (is A more like B or C) to get most important about which A is?

We did this on a lot of datasets.

Wilber et al. SNaCK 2015.

8.2.1 Feature elicitation

We lose a lot of information from just asking similarity. There are many reasons (both have glasses), etc. We ask the crowd to give us features. The features should be

- numerous,
- salient
- dense
- easy

Ex. dictionary for sign language. Standard image tagging gives constant features.

Give compare/contrast questions: Which one is different? Why is it different? (not helpful) What is common to the others? Vehicles (helpful).

We give 2/3 comparisons. What feature is common to two of the three examples? Ex. Two hands. 6-% responses on random triples were number of hands.

Use crowd to get features and label rest of images with features.

We don't compare a triple for which we already know 2/3 have a feature in common.

Pick uniformly a random "unresolved" triple and ask.

Without active learning, run into problem that people give synonym for feature (male, man).

Two models are hierarchical and independent. If you show your algorithm works in both cases, that's indication it's a reasonable algorithm.

At some point you don't have unresolved triples anymore.

There are generalists and specifists: ex. two birds and a car: "nature" vs. "bird".

Every time you ask an unresolved triple you find a new node on the tree.

For hierarchical/independent features, adaptive gives $d, \theta(d)$, while nonadaptive comparisons gives $\frac{d^3}{4}$, $2^{\Omega(d)}$.

Analysis of hiearchical case:

1. Every triple has a feature that distinguishes one.
2. Every feature has a triple for which it is the unique 2/3 feature.
3. By design, never rediscover a feature.

Nonadaptive algorithm requires $\Omega(d^3)$ expected computations because only 1 triple exposes the deepest feature to a generalist.

Ex. in sign language: 1/2 hands is general feature, thumb use is fine feature.

Programming with help of the crowd. We want computers to program for us. Given a natural naguage description "Get the last number" and/or training examples ("foo 12", "12"), ("55 bar", "55"). The crowd can give lots more unlabeled examples.

8.3 Conclusions

Machine learning steps:

0. Choose problem

1/2. Get data.

Help us come brainstorm interesting machine learning problems based on data set that we will create.

Help invent a fun game: guess gender from picture of handwriting. Guess eye color from picture of face with eyes closed.

Crowdsourcing can help ML find models and representations, make better use of resourcing. +ML can help us make progress on hard ML problems.

QA:

How many of the features are easy for a computer to learn?

Images are easy for crowd to understand, but you can also choose other data (audio, etc.).

9 Active Learning Beyond Label Feedback (Kamalika Chaudhuri, UC San Diego)

Abstract: An active learner is given a hypothesis class, a large set of unlabeled examples and the ability to interactively query labels of a subset of them; the learner's goal is to learn a hypothesis in the class that fits the data well by making as few label queries as possible. While active learning can yield considerable label savings in the realizable case – when there is a perfect hypothesis in the class that fits the data – the savings are not always as substantial when labels provided by the annotator may be noisy or biased. Thus an open question is whether more complex feedback can help active learning in the presence of noise. In this talk, I will present two separate feedback mechanisms, and talk about when they can help reduce the label complexity of active learning. The first is when labels are obtained from multiple annotators with slightly different labeling patterns; the second is when the annotator can say "I don't know" instead of providing an incorrect label.

To build an accurate classifier we need a lot of data. Unlabeled data is cheap but labels are expensive. We hope to use interaction to get around this.

We have x_i's and try to find a prediction rule to predict y from x using few label queries. By querying strategically we can get away with fewer labels.

The challenge is incorrect/noisy responses. Or we are in the agnostic setting: there are no assumptions on data distribution.

Active learning could be statistically inconsistent. Even if label budget is infinite, you converge to a local minimum.

Can other kinds of queries help active learning?

We use the Probably Approx. Correct (PAC) model

- Concept class X, samples (x_i, y_i) from data distribution D

- Ex. C =linear classifiers,
- Find $c \in C$ with low error $\mathbb{P}_{(x,y)\sim D}(c(x) \neq y)$.

We look at 2 versions:

- realizable: there is a perfect classifier, $\exists c^* \in C$, $c^*(x, y) = y$ for all $(x, y) \sim D$.
- agnostic: No assumptions on D.

This leads us to agnostic active learning: if the best $c \in C$ has error ν^*, we want to find a classifier with error $\leq \nu^* + \varepsilon$.

There are 3 types of algorithms.

1. disagreement-based active learning
2. margin/confidence-based active learning
3. clustering-based active learning.

We focus on disagreement-based active learning.

Algorithm:

1. Maintain candidate set V that contains best $c \in C$.
2. For unlabeled x, if $\exists c_1, c_2 \in V$ such that $c_1(x) \neq c_2(x)$, then x is in disagreement region, query x.

9.1 Weak and strong labelers

What happens if we have auxiliary information in the form of an oracle? Ex. doctor is the oracle, expensive but correct. Medical resident is a weak labeler, cheap and sometimes wrong.

We can make interactive label queries to oracle O or weak labeler W. We want to minimize label queries to O.

We want to find $c \in C$ with error $\leq \nu^* + \varepsilon$ on O.

Problem: The weak labeler may be biased. There can be a region where W gives the wrong answer. (You can't ask multiple times and average.)

[UBS12] make explicit assumptions on where W and O differ, close to decision boundaries.

[MCR14] give no explicit assumptions, but applies to online selective classification and robust regression.

We give general learning strategy from W and O with no explicit assumptions.

The main idea is to learn a difference classifier h to predict when O and W differ. Use h with standard active learning to decide if we should query O or W.

1. Draw $x_1, \ldots, x_m$.

2. For each x_i, query O and W. Set $y_{i,D} = 1$ if $y_{i,O} \neq y_{i,W}$.
3. Train difference classifier $h \in H$ on $\{(x_i, y_{i,D})\}$.
4. Run standard disagreement-based active learning.

Key observations:

1. Directly learning difference classifier may lead to inconsistent annotation on target task.

   Solution: train cost-sensitive difference classifier: constrain false negative (FN) rate to be very low.

   What is label complexity?

   The number of labels to train difference classifiers is $\approx \widetilde{O}\left(\frac{d'}{\varepsilon}\right)$ where $d' = VCdim(H)$ and ε =target error.

2. Let R be disagreement region of current confidence set.

   We don't need the difference classifier to work well outside R. Just train restricted to R.

   We need to learn a difference classifier with FN rate $\leq \frac{\varepsilon}{\mathbb{P}(R)}$. We need $\approx \widetilde{O}\left(\frac{d'\mathbb{P}(R)}{\varepsilon}\right)$ labels.

   Problem: R keeps changing, so we have to retrain.

   The full algorithm: let H be the difference concept class, $d' = VCdim(H)$. For epoch k, the target excess error is $\varepsilon_k \approx \frac{1}{2^k}$. Maintain confience set V_k, disagreement region $DIS(V_k)$. Draw samples, query O and W, and train difference classifier h.

   Run disagreement algorithm A to target excess error ε_k. When A queries x, if $h(x) = 1$ query O, else query W.

The total number of labels to train difference classifier is $\approx \widetilde{O}\left(\frac{d'\theta(\nu^*+\varepsilon)}{\varepsilon}\right)$ vs. the number of labels for active learning $\approx \widetilde{O}\left(\frac{d\sigma(\nu^*)^2}{\varepsilon^2}\right)$, $\sigma \approx \frac{\alpha(2\nu^*+\varepsilon, O(\varepsilon))}{2\nu^*+\varepsilon} \leq \theta$. The number of labels for disagreement based active learning is $\approx \widetilde{O}\left(\frac{d\theta(\nu^*)^2}{\varepsilon^2}\right)$.

We assume there is h in H such that

- low FN over disagreement region
- low positives.

9.2 Abstaining labelers

Labeler abstains on more difficult examples. Can we exploit abstentions to learn better?

Example: learn thresholds. The concept class C is thresholds on instance space $X = [0, 1]$. Suppose c^* is ground truth.

The learner can query any $x \in X$ (membership query). Responses can be $+, -, ?$ drawn from unknown $\mathbb{P}(Y|x)$.

Our goal is to find c such that $|c - c^*| \leq \varepsilon$ with minimum number of queries.

When can abstentions help?

We assume that close to the decision boundary, abstentions happen more often. This could give us information about the boundary.

The basic algorithm (assuming correct response) is binary search. Divide plausible interval containing c^* by 2 each query.

For noisy response; query multiple times and average to get ground truth label with high confidence. Make an adaptive number of queries

Make 3 queries at quartiles of interval.

Adaptively learn confidence, see whether we are confident in the label at any point, or if the abstention rate is increasing in some direction.

You can query multiple points and they are independent.

Why do we need 3? We need to determine which direction the abstention rate is increasing.

The algorithm is completely adaptive and statistically consistent as long as abstention rate does not decrease towards the boundary.

Ex. suppose abstention rate is $\mathbb{P}(Y = ?|x) = 1 - C_0|x - c^*|^\alpha$ and $\mathbb{P}(Y \neq c^*(x)) \leq \frac{1}{2} - C_1|x - c^*|^2$, $\alpha, \beta \geq 1$. The number of queries to get $|c - c^*| \leq \varepsilon$ is $O(\varepsilon^{-\alpha})$ with our method, $O(\varepsilon^{-\alpha-2\beta})$ using only labels.

Summary: abstentions may help if erate increases close to decision boundary. We have algorithms for thresholds and smooth boundary fragments. We have work in progress in the PAC model.

Conclusion: more complex feedback helps active learning under certain conditions. We need more sophisticated algorithms.

Q: If you have an initial classifier, and want to adapt to specific tastes? Domain adaptation. We are dealing with adaptation when labeling function changes. We don't deal with distribution changes.

Q: What happens in higher dimensions? We look at smooth boundaries. Break into multiple 1-D problems; it's a nonparametric problem.

Q: For 1-D, lower bound.

10 Active Learning for Multidimensional Experimental Spaces of Biological Responses (Robert Murphy, Carnegie Mellon University)

Abstract: The scale and complexity of biological systems makes biological research a fertile domain for active learning. This is because for complex biological systems, time and cost constraints make it infeasible to do all possible experiments. Previous applications of active learning in biology have been limited, and can be divided between retrospective studies, the goal of which is just to demonstrate the usefulness of active learning algorithms, and prospective studies, in which active learning is actually used to drive experimentation.

The latter ones are rare. Furthermore, past studies mostly considered unidimensional active learning, where only a single variable is explored (i.e., which member of a set of drugs is most active on a single target). The goal was to reduce cost for a study that would have been feasible but expensive if carried out exhaustively. However, most biological systems have multiple, interacting components, and thus require multidimensional active learning (e.g., choose which pairs of drugs and targets to test in order to model possible effects of multiple drugs on multiple targets). This is far more challenging, but the goal is not just to reduce cost but tackle problems not otherwise addressable. In this talk, I will describe both retrospective and prospective applications of multidimensional active learning to biological systems. Considerations discussed will include the choice of the modeling method, incorporation of prior information using similarity matrices, and how to know when a model is good enough to stop doing experiments.

The oracles aren't humans, but experiments.

In drug development we have big problems and little data. We have nowhere near the amount of data we need. This is the quintessential place where we need active learning.

- Diseases are complex/heterogeneous, can be affected by many variables.
- Drug effects can be very different depending on patient and disease.
- We would need a huge matrix with lots of variables...
- The biggest problem of drug development is not finding drugs that do what you want, but finding drugs that do what you want *and not other stuff*.

  We need to know the effect of drugs on many different things in the body, not just one thing.

"Genetic code" seems to imply there are rules. There aren't rules! Bio systems are complex systems without rules/laws. All we can do is build empirical models

Let's take a piece of that where we learn, for some system, learn a matrix or tensor that describes responses of systems as a function of different sets of variables.

Consider the case where we have many possible drugs (var 1) and targets (var 2). There are millions of drugs, 100,000 targets.

Constrast with the way things are done now: pick a particular target (protein important in cancer) and find drugs which block that target. There is active learning work to try and reduce the experiemnts.

But we didn't learn anything from protein 1 for protein 2.

People thought from 2 perspectives:

1. matrix factorization from whatever data you have
2. take data on drugs we know already to predict other drugs without doing experiments

Neither is satisfactory.

What kind of active learning problem is this?

We don't have unlabeled data—we have nothing. All we have is variables of all the drugs and targets. We have to do an experiment to get the response. We typically start

with very little data. If only a little data is missing, we can do matrix completion and predict. If we have all data on some drugs and not on others, we can do also do matrix completion/factorization.

The most common case is starting from nothing; most data is missing. We may still have some information to make predictions: features of the drugs or targets, measures of chemical similarities between drugs, similarity of pathways in the cell. When we do have the info, we often don't know how good it is.

What are we prediction? Most work predicts binary values, and sometimes a real values. It turns out for many responses, we can't think of it in those terms. There's not a single axis over which the response occurs. A drug could have totally different effects in different patients. We care about the class of response. Does it affect the heart, kidney, cell transport, etc.

Almost all work on active learning is done to test an approach using retrospective studies where all data is available. "It meets all the assumptions."

We assume fixed costs to experiments, but in general it could be variable; we also have an oracle accuracy problem.

We want to do "batch" experiments. Biologists tend to do 1000 experiments at a time for efficiency. We have equipment to do this.

Most work has been done with small spaces of drugs and targets.

We do standard uncertainty-based active learning. In all datasets which have only been analyzed in the framework "given 80%, predict other 20%", we found they were good candidates for active learning. Doing active learning up to 80% is much better.

We try to extend to something closer to the real problem. We took a subset of PubChem with 177 assays and 133 protein targets, 20000 compounds, $\approx 10^6$ experiments. Almost all of current matrix factorization approaches don't work efficiently on data this big.

Given features, make a LASSO model to predict the target, and vice versa. We have lots of row and column predictors, and average the results. This is the driver for the active learning.

We compare random search (random choice of experiments), QSAR model (model for each target), and active learning in terms of counts of discoveries.

We're starting to get the kind of results we want. With only 2.5% of the matrix covered, we identify 57% of responses!

Problem: This needs features to make predictions before we started. What do we do when outputs are multidimensional? We're now in the space where any phenotype is possible.

We measure features not of the drugs and targets, but of the image (cells expressing a target treated with drug), and make a representation of the possible phenotypes. We get clusters of phenotypes that are similar responses.

Do matrix factorization on phenotypes. Each experiment is assigned a phenotype by clustering (categorical). Group the drugs by which drugs are not provably dissimilar, potentially the same. Do the same thing for targets.

Now we predict the phenotypes for everyone. We assume same phenotypes with other members of the group. This gives a predictive model.

How to use this in active learning?

Which experiments explicitly test the assignment of a drug to a group. We try to break

the implicit factorization. Also take into account if we were to falsify one assignments, how many other things would have been falsified?

This is along the theme, trying to get a mutual information measurement, looking at the impact on future models without doing an explicit calculation.

Using that approach, we try to test prospectively: set up a robot to add drugs to cells, set up camera, computing pipeline... The computer decides the experiments to run next.

The experiment space is 48 proteins and 48 drugs where we know nothing about drugs or targets. But we don't know whether the model is right, so we duplicate all the drugs and targets, so we get a 96×96 space.

(Our group has automated image analysis. We have lots of experience with this: textures, etc. We presume we have a good unsupervised feature generator.)

Start with looking at the cells when no drugs are added, and then run; stop when we run out of money. We sampled 28%. This accounted for 78% of responses.

We ask: at each round how good of active learning was the model. We can't compare predictions because they're clustering dependent. We measure how close the estimated phenotype is.

[Demo: big matrix with experiments done, correct predictions, incorrect predictions. Why vertical strips of incorrect predictions? drugs that the model didn't learn well because the response of cell is very variable.]

After 28% of experiments, it is 92% accurate, 40% more accurage than random experiment. Almost all improvement came from duplication structure.

This is the first example of AL in biology where we didn't know phenotypes. It's the first real-world case!

One real issue is knowing when to stop active learning (assuming I still have money...). It's not talked about very often because so much active learning work is done retrospectively, when you know the whole model. In the real world the aim is to avoid during those experiments.

We tried a different approach. We can characterize/parameterize an experimental space using 2 parameters: sparseness of interactions (how often does a target respond to a drug), and how similar drugs and targets are to each other.

Simulated results: In the unique case you get no improvement. In the case where there are few responses (needle in haystack) you don't get improvement. Otherwise we get significant improvement in active learning.

Define a set of features of an active learning trajectory, like how consistency changed from step to step, how many I found so far... Learn regressors from those features to accuracy. If this is a fair parameterization of space, then I have a predictor of accuracy for experiments drawn from that space.

In the range that I care about ($\approx$ 90% accuracy), I can get accurate prediction. The prediction is conservative.

Doing the same thing I did before, but deciding to stop using this criterion, I get a comparable accuracy compared to just doing a fixed percentage.

Q: Adverse interaction is much more important, more important to catch? I flipped around between accuracy and area-under-curve (AUC). I could weight differently.

11 Interactive Language Learning from two extremes (Sida Wang, Stanford University)

Note: First part of talk overlaps with Percy Liang's talk `https://www.dropbox.com/s/cbwmt7i2o9p0ki0/simons_ml.pdf?dl=0` (§9.5 Interactive learning)

Towards the goal of creating more usable and adaptive language interfaces, we consider two extremes of the solution space – starting from scratch, and "naturalizing" a programming language.

The former is inspired by Wittgenstein's language games: a human wishes to accomplish some task (e.g., achieving a certain configuration of blocks), but can only communicate with a computer, who performs the actual actions (e.g., removing all red blocks). The computer initially knows nothing about language and therefore must learn it from scratch through interaction, while the human adapts to the computer's capabilities. We created a game called SHRDLURN in a blocks world, and analyze how 100 people interacted with the game.

On the other extreme, we seed the system with a core programming language and allow users to "naturalize" the core language incrementally by defining alternative syntax and increasingly complex concepts in terms of compositions of simpler ones. In a voxel world, we show that a community of users can simultaneously teach one system a diverse language and use it to build 240 complex voxel structures. Over the course of three days, these builders went from using only the core language to using the full naturalized language in 74.7% of the last 10K utterances.

Natural language understanding is the bottleneck. We want NL systems to learn from mistakes.

We are stuck when these systems misunderstand us. We want it to adapt to users, handle special domains and low resource languages where mamiliar words take on new meaning, and perform complex actions (ex. call Bob, hang up after 8 rings).

11.1 Learning language games from scratch

Language derives its meaning from use (Wittgenstein).

We consider a iterated, cooperative game between human and computer.

- The human has a goal and cannot perform actions, but can use language and provide feedback.
- The computer player does not know the goal, can perform the actions, but does not understand language.

The human must teach the computer a suitable language and adapt, and the computer must learn language quickly through interaction.

The setting is a blocks world. The human gives an utterance; the computer produces a ranked list of end results; the human chooses the desired result.

We user semantic parsing: actions are logical forms such as add(hascolor(red), cyan). Multiple actions can correspond to the same block configuration.

Use parsing freely: generate logical forms from the smallest to largest, score with a model, and use beam search.

Score logical forms with a loglinear model with features of the input and logical form (features of the input are arbitrary strings).

$$p_\theta(z|x) \propto \exp(\phi(x, z) \cdot \theta).$$

We don't have access to z, just the denotation,

$$p_\theta(y|x) = \sum_{z:Exec(z)=y} p_\theta(z|x).$$

Use a L1 penalty and update with AdaGrad.

Features include uni-, bi-, skip-grams for the utterance and tree-grams for the commands. Real features are cross-products.

We got 100 Turkers to play SHRDLURN, and got 10223 utterance (6 hours total). We gave minimal instructions. Measure performance by amount of scrolling needed.

Tasks are in order of difficulty.

Good players are consistent and match the computer action space.

Players adapt to the task by becoming more consistent and precise. Our full model gets 33.3% accuracy, 48.6% for top players.

Pragmatics makes learning quicker.

Findings:

- Our system learns from scratch quickly
- Learning pragmatics helpful
- Players adapt.

(Idea: let players design own teaching tasks.)

11.2 Learning concepts through definitions

Drawbacks of previous system:

1. Logical forms are simple.
2. Each user has a private language, no sharing. The system does not improve with more users
3. Exponential number of logical forms (Selection as a supervision signals cannot scale very well because the number of logical forms is exponential in length.)

Our solution is naturalization:

- Seed the system with a core programming language that ensures capability, defines action space, but is tedious. It is unambiguous and composes tractably.

- User augments the system by adding definitions like "3 by 4 square:= 3 red columns of height 4."

  The is the Montague approach to language: language derives its meaning through definition.

We do this in a voxel world, Voxelurn. Voxels are (x, y, z, color). Domain specific relations are directions.

The core language is designed to interpolate with NL but has usual programming language constructs. We avoid explicit variables with lambda DCS.

- controls: if, foreach, repeat, while
- lambda DCS for variable-free joins, set ops, etc. "has color yellow or color of has row 1"
- selection to avoid variables: select left of this.
- block-structured scoping.

Demo:

- "add palm tree".
  - Explain: "Add trunk at 3".
    - Add brown top 3 times.
  - Go to top and add green blocks.

Use probabilistic model to handle scoping. We can now use palm tree in other definitions and commands. "Row of 5 palm trees 5 spaces apart."

At first we have to do the bracketing, etc., but quickly it's getting powerful.

The model is now over derivations:

$$p_\theta(d|x, u) \propto \exp(\phi(d, x, u) \cdot \theta).$$

There is less collision, but we have to handle scoping choices.

A derivation describes how to derive the formula from the utterance. Ex. (loop 3 (add red left)).

We have features, like which rule you use. This turns CFG to PCFG. We have indicators: whether the rule is induced or core. This is a community project so we add features like social.author, social.friend (id of author), social.self (is rule authored by user?).

We user grammar induction: substitute matching parts. Input x: add red top times 3. Define as X: repeat 3 [add red top]. Derivation: (loop 3 (add red top)) and how it is derived. Substitute matching derivations by their categories.

There isn't a generic way to be correct because there is context dependence. We use the same scoring for formulas to determines which parts to abstract (Zettlemoyer, Collins 2005).

Can people actually do it? We got turkers to use the system and build structures.

Setup:

- qualifier: build a fixed structure
- post-qual: over 3 days build whatever they want.
- prizes for best structures in categories, ex. bridge, house, animal.
- prize for top h-index: a rule (and its author) gets a citation whenever it is used.

The users don't actually see the rules—if text matches then the computer proposes the derivation with the rule; if accepted, the rule gets a citation. A rule has as a property the person who proposed it.

Statistics: 70 workers qualified, 42 participated, 230 structures. 64K utterances, 36K accepts. Each leads to a datapoint labeled by derivation.

Is naturalization happening? Has the language evolved from the core to what the user wants? Measure by percent utterances in core or using induced rules.

The percent of utterances using induced rules is 58% at the end (up from 0). At the end, 77.9% of last 10k are accepted. Top users naturalized to different extends, but all increased.

A rough metric of expressive power is cumulative average of string.length in program / number of tokens in utterance. This is stable at 10 for core language. As time passes the ratio increases; this varies greatly by user. A jump is when the user defines a higher-level concept.

Modes of naturalization involve:

- short forms, like "l" for "move left"
- syntactic: pick different syntax. go down and right := go down; go right (sequencing)

  select orange := select has color orange.

  add red top 4 times := repeat 4 [add red top]
- higher level: add black block width 2 length 2 height 3, cube size 5.

Therer are 1113 cited rules. Top rules: left 3, select up, right, ..., go left, select right 2, etc.

You need to do simpler actions more frequently, so simpler actions everyone needs have high citation counts.

(If augment with these primitives, what more naturalization do you need? But this way you don't have to figure out alternate phrasings. We also have complex concepts that require 2 loops, 1 subconcept.)

The hope is many different versions of flowers; you can have "flower" have different definitions.

The goal is to bridge the gap in power. Naturalize a programming language to handle complex actions, share community learning to cover more variations, and be better for beginners.

Pidgin language?

We covered 2 extreme:

1. LLG: start from scratch, understand nothing, anything goes; user has private language; selection is supervision; features and learning from denotations do heavy lifting, language agnostic

2. NPL: start with programming language and power, user community has shared language, definition is supervision; grammar induction.

We tried to apply the same approach to a calendar setting. See blog post.
Q: What are backgrounds of turkers? Try out with schoolkids?
We did a survey about programming experience. 2/3 had no programming experience.
Important is the curriculum, sequence of tasks.
Code, experiments, demo: `shrdlurn.sidaw.xyz`.

12 Robot learning from motor-impaired teachers and task partners (Brenna Argall, Northwestern University)

Abstract: It is an irony that often the more severe a person's motor impairment, the more assistance they require and yet the less able they are to operate the very machines created provide this assistance. A primary aim of my lab is to address this confound by incorporating robotics autonomy and intelligence into assistive machines to offload some of the control burden from the user. However, robots which do not adapt to the variable needs of their users when providing physical assistance will struggle to achieve widespread adoption and acceptance. Not only are the physical abilities of the user very non-static and therefore so also is their desired or needed amount of assistance but how the user operates the robot too will change over time. The fact that there is always a human in the loop offers an opportunity: to learn from the human, transforming into a problem of robot learning from human teachers. Which raises a significant question: how will the machine learning algorithm behave when being instructed by teachers who not only are not machine learning or robotics experts, but moreover have motor impairments that influence the learning signals which are provided? This talk will overview a new area beginning to be explored by my research group: that of robot learning from motor-impaired teachers and task partners. Our goal is to transform robotics autonomy in rehabilitation by designing algorithms that treat the constraints imposed by motor-impairments as advantages, rather than limitations.

RIC (rehabilitation in Chicago). We have access to rich population of patients and to expertise of therapists, so we can focus on solutions deployable immediately. There is robotics in rehabilitation (wearable robots such as prosthesis), but clinically there is not higher-level autonomy.

Problem: Machines are difficult for people with motor impairments to control.

Power wheelchair is 2-D . If you can operate a joystick, it becomes a natural extension of your body. For headrest, you get 1-D at a time and the signal is discrete. To go faster you have to navigate the interface to change the power level. Straw-based interface. Take 2-D and parse into 1-D controls.

Robot arm. To just position end of arm is a 6-D problem. 2-D/3-D joystick only has a small part of control. If you need assistance from robotic arm, you don't have the ability to control the joystick! The more motor impaired you are, the less you are able to issue the more complex signal the machine requires.

Control challenges:

- limited interfaces
- motor impairments
- machine complexity
- users have unique and non-static abilities (people have different personal preferences. They can be rehabilitating, have a degeneratie condition, have fatigue/pain throughout the day, etc.)

To turn an assistive machine into an assistive robot,

- add sensing
- add computing and AI to reason what it senses

We prioritize customization and low cost for wheelchairs. We don't expect this to be covered by Medicaire/Medicaid anytime soon. A therapist has to justify every choice on the wheelchair to get it covered by insurance. When it shows up on reimbursement forms, I will consider it success.

We have modular software and hardware. Someone comes with power wheelchair and control interface (widely vetted and covered by insurance). We want to do the best job in software.

We draw from power on wheelchair to get computing, power, and electronics. Add RGD-B sensor, and maybe IR, ultrasonic sensors, IMU at additional cost and capability.

Customize through control sharing. Robot has an idea of what it should do. To reason about those 2 signals and combine into a single one is understudied and widely variable. It needs a lot of study. Ex. do too much for them or do something unanticipated, they reject the tech. This is the linchpin.

We introduce tunable knobs. It can't adapt too quickly (unanticipated) or slowly. We're gather teleoperation data to statistically analyze differences.

12.1 Sharing control between human and assistive robots

Autonomy needs a goal. One way is to forward project into the future (.5 second into the future) with fewest assumptions on what the human wants to do. Can detect places where human needs assistance, ex. doorways can be detected. Docking locations at table, anchor to circular place settings. Check for clearance and safety. If there is more than 1 candidate location, do intent inference. Ex. video with 2 doorways. Perceive goal, choose between goal.

Autonomy comes up with safe path that reaches the goal. This gives the autonomy-controlled signal. One way to decide is **filtering**. Cap user's control signal to not exceed autonomy control in either dimension. Look at how far, whether oriented in the direction, extent to which human signal agrees with going through the door. If human signal stops, then most of the time it stops. If really close, autonomy might take over. This is customizable.

This is not driverless cars—this is parking assist, etc. Can wheelchair tell human what its inferred goal is? Getting this right: some information better than none, but not too much. This is another research area.

Another paradigm is **blended**. Still check that the blended signal is safe. Ex. driving towards obstacle, the robot swerves to avoid.

Another way is **switch**: do blending, and take control when goal is unambiguous. Occasionally people want 100% autonomy.

There are so many different ways to formulate partial autonomy. Do people have preference; can they tell between them? We did a comparative study, 5 control paradigms, 2 interfaces, 2–4 sessions, 7 injured and 7 uninjured subjects. Task: 4-doorway traversals.

Manually operating the interface is painstakingly slow to align.

There's not a clear winner between the different paradigms (but they all do better than manual), but we see a clear preferences for most users.

- No single control paradigm is most preferred across subjects or performs best with statistical significance.

  Filtering is a clear loser (number of interactions, time). This is capping the speeds to not exceed autonomy.

  Preferences changed between sessions. We want to do a 30-session study.

  This doesn't tell us about so much about individual paradigms; it tells us it's good to give options.

  It's hard to be statistically significant; do case studies.

  Q: How to differentiate paradigms vs. implementations of control paradigms? Ex. "Wizard of Oz" where a human, rather than computer, infers intent. Human vs. autonomy is more different than difference between autonomous methods.

- Both performance and preference change with control interface. Performance differences decrease with increasing autonomy.

- Performance changes across sessions (learning familiarity factor).

  SCI switch preference 58% of the time with both interfaces. There is always a certain level of difficulty.

  Uninjured subjects switched much less with the joystick.

- Few differences between subject groups. Greatest differences are from command fluency. Zero differences are from distances to obstacle.

- We can model preferencec as a function of computable metrics. Not just time to completion, etc., but how the human is interacting. One metric is number of interface interactions. Fewer is better. Signal frequency, disfluency...

How to tune parameters? We explored with robot arm end-user customization. We had 4 SCI subjects and 13 uninjured subjects.

Many paradigms take the functional form of a piecewise linear function $0, \alpha c(x - x_0), \alpha$. We wanted people to customize the function. We had people verbally telling us how they wanted it to change (autonomy help faster, come in sooner...).

Experiment with robot arm: For manual operation, need to change control modes. With assistance, don't need to switch when the control mode is very informative about which object you're going to. Current research: Automatic mode switches to maximally inform autonomy. Confidence of intent inference.

Unlike in navigation, what to do when you have the goal object could be varied.

We're starting to consider voice commands. It's much more ambiguous. To give high-level goal it's better (go to kitchen). 20 years ago there was a commercial voice recognition that was dangerous. Ex. when people shout "stop" it didn't recognize the emergency tone of voice.

Pilot: "help me help you". Try to seek out disambiguating information.

Once we get to robust BCI, that will be a game-changer in these domains. What what I've heard, we're 15 years away. For operation a wheelchair you can use EEG signals because it's discrete. For arm control, people need an embedded BCI.

Estimating intent based on distance is not rich enough. It may just need to be more richly formulated.

Results:

- Difference between SCI and uninjured subjects diminish with assistance and are eliminated with customization.
- SCI customized for more assistance, uninjured for less assistance.
- User-directed optimization considers something more than standard cost functions. 7/34 customizations had greater completions, and 14/34 had more mode switches.

12.2 Robot learning

Future work is adaptations, robot learning from motor-impaired teachers. What does robot learning look like here? Tactile interface to provide corrections is not feasible here. You need a signal. Where the signals come from is a big question. How do we define something from scratch? Try to define 15-D control signal. Outfit my arm and hand.

Do we need new algorithms or reformulate existing ones.

Gold standard: What would be great is if operating the device had motor learning benefits. If people can't get to fully capable, use assistive machine. What's great if the machine can help with rehabilitation, encourage using it less.

Use shoulder movement. Can control 2-D; what about 6-D? Autonomy could be monitoring human performance. When people becomes more capable, unlock new dimension.

Motion calibration: move in all dimensions as much as they can. This defines variance; take dimensions where they have most movement for control. This encourages movement within full range of motor capability.

This uses robot machine learning to elicit human motor learning!

13 Reinforcement learning with rich observations (Alekh Agarwal, Microsoft Research New York)

Abstract: This talk considers a core question in reinforcement learning (RL): How can we tractably solve sequential decision making problems where the learning agent receives rich observations? We begin with a new model called Contextual Decision Processes (CDPs) for studying such problems, and show that it encompasses several prior setups to study RL such as MDPs and POMDPs. Several special cases of CDPs are, however, known to be provably intractable in their sample complexities. To overcome this challenge, we further propose a structural property of such processes, called the Bellman Rank. We find that the Bellman Rank of a CDP (and an associated class of functions) provides an intuitive measure of the hardness of a problem in terms of sample complexity and is small in several practical settings. In particular, we propose an algorithm, whose sample complexity scales with the Bellman Rank of the process, and is completely independent of the size of the observation space of the agent. We also show that our techniques are robust to our modeling assumptions, and make connections to several known results as well as highlight novel consequences of our results. This talk is based on joint work with Nan Jiang, Akshay Krishnamurthy, John Langford and Rob Schapire.

RL: Unlike in other ML problems, actions have consequences. Actions modify state, which influences what rewards you can pick up in the future.

Ex. rat in maze. When it stumbles on the cheese, it knows there is something to be had! Before, it had no information and had to act to obtain information. There may be no intermediate reward (ex. until you get the finish point in the maze). You can take as a state the position in the maze, or the visual perception. If the maze is large enough, stumbling on the cheese is difficult. Just having powerful perceptive abilities may not be sufficient to solve the RL problem well.

How to learn?

- Practice: Powerful modeling, simple exploration (Atari Deep RL)
- Theory: Sophisticated exploration in small-state MDPs, e.g. E^3, R-MAX algorithms.

  There is limited theory for rich observations.

Often the world is not a set of discrete cells; you perceive through various sensory inputs.

The goal is to close this gap: develop RL approaches guaranteed to learn an optimal policy with small number of samples despite rich observations.

We present a single formalism and algorithm that derives many RL models and solutions. Results:

- Small state MDP's
- Structured large-state MDP's (new PAC guarantes): We focus on this.
- Reactive POMDP's
- Reactive PSR's (new)

- LQR (continuous actions)

Key ideas:

- New measure of hardness of exploration: Bellman rank
- Algorithm with sample complexity scaling with this measure
- Applications in several RL settings.

A Markov decision process (MDP) is a sequential process.

- $x_1 \sim \Gamma_1$ initial distribution of state. Think of x as being continuous, high-dimensional.
- Take state a_1, observe $r_1(a_1)$.
- Induces distribution over next state $x_2 \sim \Gamma(x_1, a_1)$.

Assume

- episodic: H actions in trajectory.
- layered: distinct states at each level.
- Markovian: x_h only depends on (x_{h-1}, a_{h-1}), r_h on (x_h, a_h)

We maximize long-term reward using policies (mappings from states to actions)

$$\sum_{h=1}^{H} r_h(\pi(x_h)).$$

Ex. Consider a robotic agent navigating in a gridworld. A more realistic setting is when you get images from a camera, and we reason based on those.

Ex. Web search: User comes in with intent, issues query, receives ranked list of results, issues another query.

Existing results:

- Learn ε-optimal policy using poly$(|X|, A, H, \frac{1}{\varepsilon})$ samples.
- Small number of states necessary for learning.

There is a MDP with $|A|^H$ states where finding an ε-optimal policy requires $\Omega\left(\frac{|A|^H}{\varepsilon^2}\right)$ trajectories. Inuition: embed a bandit problem with $|A|^H$ arms in a tree.

A compact F is not sufficient for generalization in RL; gathering the right data has large sample complexity.

In large-state MDP's, there are too many unique states. We cannot reason separately for each state, so we need information sharing between similar states, generalization. This is typically done via value-function approximation.

Let Q^* be the optimal value function. It maps (x, a) pair to the long-term reward when you take action a in state x and follow the optimal policy thereafter.

$$\pi^*(x) = \text{argmax}_a \, Q^*(x, a).$$

Let's approximate it. Given a class F of functions $X \times A \to R$, find a good approximation to Q^* assuming $Q^* \in F$. The associated greedy policy is $\text{argmax}_a \, f(x, a)$.

The key intuition is to use a class F that generalizes well in supervised learning (small VC-dimension/Rademacher complexity/finite size).

Here is a solution sketch.

- Start with initial guess $f_1 \in F$ for Q^*.
- Act according to f_1 and collect trajectories, $(x_1, a_1, r_1, \ldots, x_H, a_H, r_H)$ where $a_h = \pi_{f_1}(x_h)$.
- Use to obtain better estimate $f_2 \in F$. (How to improve?)
- Repeat.

Use the Bellman equation

$$\varepsilon(f, \pi, h) = \mathbb{E}[f(x_h, a_h) - r_h - f(x_{h+1}, a_{h+1})]$$

where $a_1, \ldots, a_{h-1} \sim \pi$ and a_h, a_{h+1} are according to π_f (h is the number of steps). Think of this as a consistency check. For all π, $\varepsilon(Q^*, \pi, h) = 0$. This gives a test (1-sided certificate) for checking if $f \approx Q^*$.

To use the Bellman equations,

- given candidate $f \in \mathcal{F}$, check $\varepsilon(f, \pi, h)$ for all π, h.
- Reject f if $\varepsilon(f, \pi, h) \gg 0$ for any π, h.
- Restrict to $\pi = \pi_g$ for $g \in F$.

Say f if valid if it satisfies this. This is a necessary but not sufficient condition: it could be a bad policy and still be valid.

The challenge is that computing $\varepsilon(f, \pi, h)$ requires samples from π. Doing it for all π_g requires $O(|F|)$ samples.

We have no reason to expect data from one policy would help for another. There are too many functions in any interesting F, and data based on one f might not prove suboptimality for another.

Consider the $|F| \times |F|$ matrix

$$\varepsilon(F, h)_{f,g} = \varepsilon(f, \pi_g, h).$$

We have one such matrix at every level h.

Definition 13.1: The **Bellman rank** of a MDP is the rank of $\varepsilon(F, h)$.

Sample complexity will be polynomial in rank.

If matrix is low-rank, we can use information about a subset of entries to propagate information to other entries. This is bounded by

- number of states
- rank of transition matrix Γ
- number of "hidden" states (for reactive value functions that don't reason over histories).

It can be bounded by intrinsic structural parameters you don't necessarily observe but that represent the hardness of the problem.

For continuous state space (LQR) we argued that the Bellman rank is at most d^2. Our algorithm does not directly apply to continuous actions.

The number of cells in the grid is a good upper bound on the rank, even if observations are in a larger space.

Algorithm intuition: low Bellman rank gives concise basis for checking validity (exploration).

Challenge: we don't know the basis, just its existence.

Algorithm: Optimism Led Iterative Value-function Elimination (OLIVE)

- $F_0 = F$.
- For $t = 1, 2, \ldots,$
  - Choose f_t to maximize $\widehat{V} = \mathbb{E}_{x \sim \Gamma_1}[f(x, \pi_f(x))]$. Optimism under uncertainty. This is the guess for $V(\pi^*)$ if $f = Q^*$.
  - Collect trajectories using $\pi_t = \pi_{f_t}$.
  - If $V(\pi_t) \geq \widehat{V} - \varepsilon$, return π_t. This is checking our optimistic belief.
    Otherwise it's not Q^*. Eliminate it, and also:
  - Reject all f with large $\varepsilon(f, \pi_t, h)$ for any h (prune the possible solutions).
  - Set F_t to be the set of surviving f.

Theorem 13.2 (PAC guarantee)**.** *Suppose $Q^* \in F$. Suppose Bellman rank is at most M. OLIVE returns π, $V(\pi) \geq V(\pi^*) - \varepsilon$ and with probability $1 - \delta$, number of trajectories*

$$O\left(\frac{M^2 H^3 |A|^2 \ln\left(\frac{|F|}{\delta}\right)}{\varepsilon^2}\right)$$

This is probably suboptimal exponents on everything except ε.

This retains sample-efficiency for small-state MDPs, gives new results for several settings, and gives unifying treatment for sample-efficient RL.

Proof intuition (correctness):

- Algorithm always retains Q^*, terminates when $V(\pi_t) \geq V^* - \varepsilon$.

- Bellman error matrix has low rank.
- Each elimination step decreases rank by 1 if we check for $\varepsilon(f, \pi, h) = 0$. (For the noise-free case, this is just algebra.)
- Extension to noisy checking: ellipsoidal argument. Reduce the volume of Bellman error vectors by constant fraction each time.

Extensions:

- Do not require $Q^* \in F$ (agnostic setting).
  Find the value f with largest $V(\pi_f)$.
- Adapt to the knowledge of M.
- Allow errors in Bellman factorization and validity.
- Allow infinite classes F with low VC-like dimension.

Wrap-up:

- New structural condition for efficient exploration
- First sample-complexity results in broad setup called Contextual Decision Processes
- Algorithm robust to modeling assumptions
- Key open problem: computational efficiency. We rely on enumerating. We can't even make it efficient in tabular setting.

Contextual bandits is the special case where the Bellman rank is 1.

Q: Practically optimism is overkill. Can you apply to Thompson sampling, etc? It might operate by inducing a distribution on policies.

14 Robots learning from human teachers (Andrea Thomaz, UT Austin)

Abstract: Our research aims to computationally model mechanisms of human social learning in order to build robots and other machines that are intuitive for people to teach. We take Machine Learning interactions and redesign interfaces and algorithms to support the collection of learning input from end users instead of ML experts. This talk covers results on building models of reciprocal interactions, high-level task goal learning, low-level skill learning, and active learning interactions using anthropomorphic robot platforms.

My lab is called the Socially Intelligent Machines Lab. We want to get robots out in the world interacting with people!

A social robot is a robot who's functional goal involves interacting with people in an environment designed for humans.

Many robot successes are in structured environments, program once and repeat. Humans and human environments are dynamic, and pre-programming controllers is not an option.

One example is in manufacturing. We want robots to be flexible coworkers in manufacturing. A small factory may want robots learn to do new things.

Another is service robots. Every home is slightly different. Only the end-user knows what the robot should do.

How do we get end-users to come into our lab and teach our robots to do stuff? Can we change anything about the interaction, algorithm to make things mesh?

Ex. teaching Simon how to close a box. A standard approach is to get demonstration from a teacher as to what it's supposed to be doing. The whole thing was recorded; getting the whole trajectory is not ideal.

Instead get **keyframes**.The person is figuring out the kinematics; we're not recording all of it.

- Good teachers were better at using trajectories and they preferred it.
- Bad teachers were better at using keyframes and they preferred it.

Students have developed this in various domains. I focus on B's work on keyframe vs. demonstration.

Ask person: was that a good input? The person thinks it's an amazing demonstration. We have to build an algorithm that works with this terrible data!

Users are goal-oriented: Concentrate on achieving the skill more than how to exactly do it. We have 2 learning problems.

- Goal model (what to do): monitor execution.
- Action model, execution: how to do it.

Robot says "I see that it was successful." "I see I failed."

Build an action model from motor data, and build a goal model that monitors execution. Keyframes highlight salient points in the skill.

(What makes something a keyframe? Can you learn this?)

Motor data is pose with respect to some target object. There is a depth camera extracting features related to bounding box, surface normals. Take both feature vectors, and build different models. Use hidden Markov models to get canonical keyframes. The hidden states are the true keyframes. The emission/observation space is larger. Use Baum-Welch algorithm.

Add prior probabilities p and terminal probabilities ζ to the model.

When executing, take learned action model and sample a path from prior to terminal states. Favor paths with lengths close to demonstrated number of keyframes. Look at emission states. To generate the canonical trajectory, take the true means of emission spaces as keyframes. Use a 5th order spline.

Having decided the trajectory, how to use the goal model to monitor it? While executing, take a snapshot of visual space that coincides with each keyframe. How likely is that this

sequence represents a success or failure in the goal? We make sure that the last frame represents something in the terminal state.

How well does this work with data we get from people? We had 8 people come to teach closing the box and pouring.

"Start here, go here, go here, go here, end here." Execute action model 5 times.

Across people, "close the box" had 57.5% execution success but 90% monitoring success. Pouring had 75% execution succeess but 90% monitoring success.

Some teachers were better than others at teaching actions. How to bridge this gap?

Execute with some variance and monitor. Do the same kind of thing. Before we splined between canonical version. Now take multivariate gaussian as sampling mechanism. Perturb the 7-D poses.

(Can you use optimism? There's much more you can do to sample the space effectively.)

We want to sample further away from the mean when we are bored (to broaden the applicability of something that works, or explore to fix).

Each rollout of the skill is an execution and evaluation. In an episode do 5 rollouts. Put the positive examples back in and rebuild our new action model.

If a human watches the process, we can use the human's labels. This forgets the user data after having enough successful samples.

Depending on ratio of success over last few executions, expand how risky we're being in sampling behavior.

Experiment: first provide failing action model. Exploration fixed the model. You can get there faster by adaptive sampling. Can we do this with real people's data?

Putting it all together. Have teacher provide labels during execution and recognition (verbal feedback) 12 users teach 3 skills. Do 5 demonstrations, 3 executions. After 1st, 3rd, and 5th demo do executions. Watch 5 samples and hear goal output. The human gives feedback. The person leaves and do self-improvement for 9 more episodes.

"Show me what you learned." "I willl gladly." "How did I do?" "Try it yourself." "Here it goes." "How did I do? I think I succeeded."

There was a lot of variance in action models. Opening the box was hard.

Self-improvement helped.

The action space is very small so it's surprising that after this process you get a skill that consistently succeeds. It started with always failing, but after rollouts, it succeeds every time.

It's surprising little you have to change to get the model to work.

Summary: human seeded action models improved with human taught goal models.

Note we didn't give the robot any notion of the objective function!

Demonstrations are stand-in for the physics, the control equations.

One limitation is that there is a better optimal way that people don't understand because they don't understand the physics of the robot. We're starting from the action model, assume it's in the basin of attraction. If we don't assume this, you can try to glean from objective function the core constraints. It is encoded in info we have.

The goal is a trajectory vs. a state. Treating as state, I'm being agnostic. Trajectory helps by giving a sequence. We're assuming that something about the way you divided it up is important, salient in goal state. Some keyframes are superfluous, and you shouldn't

think of them as hard constraints.

When you look at data, if human teachers have variability, in first few frames have variability; converge at end? Use this to drive info you want to get from people—maybe we can throw away the constraint, not worry about how you start!

15 Robot Learning, Interaction and Reliable Autonomy (Sonia Chernova, Georgia Tech)

Robot autonomy and interactive learning (RAIL) lab.

We spend a lot of time thinking about the arrows. There are lots of questions about the interaction. From user to robot learning:

- How to provide demonstrations? The power we give to the user influences the performance of learning algorithm.
- What examples to give?
- How to interpret demonstrations?

This is a 2-way process. From robot to user:

- What to ask?
- When to interrupt?
- How to elicit information?

I'll talk about 3 things:

1. Series of projects on remotely controlling robots. Teachers are remote. This is part of the downward arrow.
2. Look at semantic reasoning and interpreting intent correctly.
3. Return arrow: figure out timing of asking question, interruptability.

15.1 Remote control

`robotwebtools.org`, `wiki.ros.org/rms`

Robot demonstration from data is starved for data. The main expense is time. We want to make our robot available through the web to make data collection easier. We made a browser interface. Anyone across the world can open a browser and have whatever control we give them.

User gets a camera view. (There are 7 cameras they can switch between.) They pick up objects. This is gamified. We give them points for picking up objects in a fixed amount of time. We learn 3-D object models using the Kinect camera, scanning laser.

Each example gives one view. If we take all examples and do iterative point cloud registration process we can build a 3-D model. The models can now be mapped.

The robot explores, evaluates, and ranks the grasps. At the end we get close to 100% performance.

If 2 users have different trajectories, averaging may not be good. We only remember grasp locations. There are attempts at clustering, etc. Here we are using planning.

We compared robotics graduate students and random crowd users. Statistically the crowd is indistinguishable from the graduate student! If we have access to more data, the benefit of crowdsourcing does filter to the robotic world!

A negative aspect of gamification: a truck would roll away. People avoided the truck because everything else was easier. (Should give more points!)

We had a goal of a complex domain. Open puzzle boxes, containers, etc. We had a major roadblock: the arrow marker requires 6 controls (3 for position, 3 for orientation, ring and arrow marker). Getting them to solve complicated tasks, even precision grasp took too long. The end we couldn't do learning because we recorded users' frustration.

We had to back up, look at better ways of getting data from users in this setting. We had an intermediate set of objects. Grasp interfaces include

- free positioning (6 degrees of freedom, ring and arrow marker)
- constrained positioning (3 degrees of freedom, reduced degree of freedom, control approach angle). This gives 3-D data. Most time is spent refining how to align. Arm plans to the grasp location.
- point and click (antipodal grasp identification and learning, AGILE): analyze surface normals of 3-D surface. Algorithm figures out the lip and has a classifier that predicts where the best locations are to place the gripper.

We went from something that was frustrating to use (6-D) to something much more user-friendly. The more degrees of freedom a user has to specify, the more clicks it takes to complete a task. As interfaces are simpler, the number of tasks completed goes up.

For shiny objects (reflective surface), vision fails; fall back on free positioning.

All these tasks are object manipulation. We want to think about objects themselves, at a higher level.

15.2 Semantic reasoning

Task adaptation through analogical reasoning. We took pairs of words, fed them into conceptnet, tried to parse subgraphs to understand relationships. We want to generate explanation: society is based on customs, and a game is based on rules.

This level of understanding is important.

We can do all this, but this is a closed system. Take away the words and put in something else. For robotics there is a whole set of planning domains where the plans are in regular English. Reading recipes/procedures online (ex. wikihow) and turn them into things a robot can do.

Task: pack schoolbag. "I couldn't find glue, can I substitute tape?" "I couldn't find a pencil, could I substitute a quill?" "No, that's not useful." "Could I substitute a pen." "I couldn't find an apple, could I substitute a banana?" "No, I don't like bananas."

Analogical reasoning can be used to substitute objects.

Also pull words from vision, use WordNet, ConceptNet, ShapeNet. Combine abstract knowledge with environment. Use this to create a "situated affordance network" which combines "quill is writing implement" with "there aren't any quills", "spatulas are generally in drawers," "this user puts spatulas in a giant cup". There are certain priors for locations of objects.

We also do word sense disambiguation. Use use Bayesian logic networks. Add transitive properties. This gives more flexibility. We get good inference so far, 85% accuracy in queries without situated information. We're pushing to incorporate observations into the model. Once we have this info, we want to look at the context of learning: if the human demonstrates something with cups, how do we generalize to similar objects?

Some information is from asking humans.

15.3 Classification of human interruptibility

Use gesture, activity recognition, etc. Interruptability is inverse of workload. We take this as granted as humans. We want to give this same power to robots.

Use person descriptors and context descriptors (ex. holding a cell phone to ear) to estimate interruptility. Categories:

0. No information. (ex. back to robot)
1. busy, should not interrupt
2. busy, may interrupt
3. not busy, unaware
4. not busy, aware of robot.

Traditionally people used HMMs. We use latent dynamic CRF. We took this from the gesture community. We tried different sets of features. Some are particularly noisy. HMMs struggle with noisy features. We compared HMMs, CRFs.

We are good at classifying most, but 3 can be confused with 2 and 4. 1 is recognized well.

Ground truth comes from labelers coming in and labeling from scale of 1 to 4.

Right now, we have lots of niche projects that look at subproblems separately. We hope to connect them in the future. Robot that's learning can figure out when to interrupt to ask for help, etc.

Humans are amazing at improvising—replace spoon with fork, etc. Robots adapt very little. How do we repair plans, policies, to allow greater generalization?

We try to avoid hand-coding. We're working on a project with tactile sensors to detect wood, plastic, metal... ShapeNet, ConceptNet, WordNet: information has been encoded for

other communities; we want to pull information from them. Data is overly general. The model says "a cup is probably ceramic, metal, and wood!"

Interruptability is framed as binary. Some tasks, you want to gauge how to interrupt the user. Scale is not only thing that informs interruptability. How urgent is the situation (ex. house on fire)? Smaller interruptions have less cost, etc.

Implicit features: use ambient human substitution patterns? People in cooking shows engage in substitution activities. It would be nice to learn from ambient observations—watch cooking shows, humans in environment. We have a partnership with vision group. This is a core perception problem. Observing from a distance is an excellent way to get more data. Right now they need to be in your face because of range of 3-D sensors.

16 Interactive Learning of Parsers from Weak Supervision (Luke Zettlemoyer, University of Washington)

Natural language parsers are widely used in applications such as question answering, information extraction and machine translation. In this talk, I will describe recent and ongoing work in the UW NLP group for learning CCG parsers that build relatively rich representations of the meaning of input texts. I will cover recent work on interactive approaches for improving both data collection and parsing algorithms. On the data side, we have introduced methods for gathering semantic supervision from non-expert annotators at a very large scale and using such supervision interactively, to label as little data as possible while still learning high quality models. For inference, we introduce new neural A* parsing algorithms that achieve state-of-the-art runtimes and accuracies, while also providing formal guarantees of optimality in inference, by learning to interactively focus on promising parts of the parsing search space. Finally, I will sketch some of our future directions where we aim to extend these ideas to build parsers that work well for any domain and any language, with as little engineering effort as possible.

We are motivated by detailed semantic analysis. We need to pull out all our tricks to get this to work generally.

Given a sentence, put it through a semantic parser, get meaning representation, put it through the executor, and get a response. If the response is good, then we are successful.

Ex. "How many people live in Seattle?" becomes a database query which returns an anwer.

Ex. "Go to the third junction and take a left."

There is some amount of syntactic and semantic information from each word. Build up compositionally. Getting data for these detailed analyses is painful. How much can you get from latent variable learning—ex. just from anwers?

We have done this for lots of applications: language to code, understanding cooking recipes,...

Challenge: gather data and learn from model from scratch in each case. Why can't we reuse parsers? I'll answer this in the context of converting a cooking recipe to a graph representing the flow of ingredients.

These are simple sentences. Shouldn't an off-the-shelf parser nail this? It was an utter failure. We could design an unsupervised learning method that does better.

What does "parsing" mean? I view parsing as syntactic and semantics. Syntax is prerequisite for semantics. The hope is to reuse the syntactic parser.

There is a simple reason why parsers don't work: they were trained on 1990's WSJ articles. There are something like 5 imperative sentences. It hasn't seen a verb in the first position. It turns those verbs into nouns.

We need to get data in new domains.

- Can we crowdsource semantics?
- Train latent models of syntax from this.
- Build fast and accurate parsers. A* search vaguely related to RL.
- Actively select which data to label.

16.1 Crowdsource semantics

Semantic role labeling (SRL): given sentence, find all verbs, find all arguments. "They increased the rend drastically this year." If you solve this, you've solved some percentage of other tasks.

There is FrameNet with a large set of possible levels, PropBank (predicate-specific rules)... Roughly speaking, to annotate this requires a degree in linguistics and takes months to do well. It's a data bottleneck issue.

For training parsers for the kinds of tasks we care about, can we do something easier?

I can train any native speaker to label data for me!

Given highlighted verb, ask question about verb. Question needs to include the verb, phrase needs to be from sentence.

"They increased the rent this year." Who increased something? They. When is something increased? This year.

This is as useful as the complex annotations. How to relate to FrameNet and PropBank? There is work on pooling data across approaches, mapping to same space.

We haven't made ontological distinctions. We have attachment information; that's the hardest part.

We worked on newswire, Wikipedia. Part-time freelancers from upwork.com, hourly rate $10.

What to do with these annotations?

What is success for broad coverage semantic parsing? Produce semantic represntation. Success if any sentence you hand, can produce right edges. In any domain, if you match it, 80%. Out of domain, 30%.

It's tricky. John denied the report. John refused to deny the report. John refused to confirm or deny the report. There can be an arbitrary long-distance relationship. Syntactic parsers such as Stanford NLP fail to give long-distance edges.

Coreference resolution, pronoun resolution...

I could analyze a sentence in isolation or in context of rest of document. Few models use context. No one knows how much context would help.

16.2 Latent models of syntax

Let's jointly model the semantic and syntactic part. It's notoriously difficult to get joint modeling to help. It's not clear why. Probably semantic is harder, going back and fixing syntax messes things up.

Our algorithm uses joint training. CCG dependencies, Lewis et al. 2015. Rather than work on spanning tree, do a detailed analysis with CCG. It does tree-structured analysis, build graph structure dependencies. "Wanted" means John wanted to confirm the report: "John" should connect to "confirm". CCG is thought to be harder to parse with.

The algorithm gives syntactic dependencies. Joint training is just a matter of labeling edges.

Learn latent CCG that recovers the SRL. Optimize for marginal likelihood. It worked well out of domain too.

CCG was designed by linguistics with semantic representations in mind. There are other kinds of semantics.

16.3 Fast and accurate parsers

Assume we have labeled parse trees. (We would like to work without output of the previous step, but for now we have higher requirements for now.)

We build parsers with global features, ex. feature for every subtree with some optimality guarantee. (Typically do greedy inference, and don't have guarantees.)

Global models (e.g. recursive NNs) break dynamic programs. Our approach: combine local and global models in A* parser. We get accurate models with formal guarantees.

Lee et al 2016, EMNLP best paper.

Go from sentences to tree-structured representations. It's useful to have the notion of a hypergraph: show derivation structure to build parse. Combine pieces: ((Fruit flies) (like bananas)).

There are in general exponentially many graphs. Each hyperedge is weighted with a score $g(\varepsilon)$, some function of edge: look at details in nodes (features on subtrees, etc.). The score for a derivation is the sum of scores of edges.

Collapse into one big hypergraph that represents all possible parses. We have sharing. Explore a small subset of it. Is this a tractable graph? We have complete parses as one whole node; there are exponentially many in sentence length. As defined this is expressive, but finding the best derivation is not tractable.

In practice there are 2 ways: approximate inference (greedy or beam search), or reranking (build simpler model, rescore according to more complicated model).

Redefine graph to further collapse node, assume a lot of equivalent. Ex. store root. Doing this cleverly, get polynomial number of nodes, and you get algorithms like CKY. These are local models, the dynamic programming signature. This is a tradeoff.

Recursive neural networks break dynamic programs! I want different features depending on how I got to a node.

Global model:

$$y^* = \text{argmax}_y \, g_{global}(y)$$

Intractable amax, expressive g_{global}. Vice versa for local.

Combined:

$$y^* = \text{argmax}_{y \in Y}(g_{local}(y) + g_{global}(y)).$$

A* parsing: Search space is partially built graphs (parses).

We need some guess about how good things could possibly be in the future. We have a scoring function giving exploration priority: how good is parse so far, upper bound on how good the rest could be (admissible A* heuristic). Search in the order of that function.

In the end, when you get a complete parse, everything you didn't expand is provably worse.

A* parsing has been done before. We have a particular way of doing this for CCG.

Depending on scoring, you need a different way of doing the bounding. One way: Entire parse scored by score of each word. (Supertag-factored A* parser.) This is a strong locality assumption but does state-of-the-art! (Lewis et al 2016) We get a trivial bound on how good everything else could be by taking maxes independently.

How to not make such strong assumptions and still get a bound? Use a fancier method: a neural net for scoring subtrees. Arbitrary scoring can care about any aspect. It's hard to bound.

We did a sneaky trick: bound it loosely.

Have a neural net compute some score, and constrain it to be negative. The loose bound is 0. This would work horribly because it's very loose. Observation: we had a local model that just couldn't make certain distinctions. Sum in with local heuristic.

$$g(y) = g_{local}(y) + g_{global}(y) \leq h_{local}(y) + 0.$$

The global part just corrects the local part.

It's a standard neural net architecture. We use $\log \circ \sigma$ activation. How to set up learning scheme?

We have a good local model. Correct mistakes in online method. Violation based loss: If you would have popped something off that didn't, it's a loss. Backprop through it.

$$L(A) = \sum_{t=1}^{T} \max_{y \in A_t} f(y) - \max_{y \in GOLD(A_t)} f(y).$$

Run through, collect all violations, do online update, go to next one.

Wrinkle: set up backprop to reuse gradients in hypergraph, do backprop on hypergraph.

Updates are aggressive, force the algorithm to be efficient. We got a new state-of-the-art result. Certificate of optimality is almost always there. There are exponentially many parses but this explores only 190 partial parses on average (after learning). It hones in on the right analysis! The average sentence length is 35.

Focusing is faster!

16.4 Actively select which data to label

Can we be more focused? For any particular sentence, can we pick just a few questions? Can we use parsers to make multiple-choice questions, use that signal?

"Pat ate the cake on the table that I baked last night." A parser often gets this wrong. World knowledge (commonsense) about what can be baked helps.

Can we use human judgments to improve parse?

Run the parser. It will be the local model. Give parses from n-best list. Look for confusions where the parser isn't sure. Based on confusions, generate a multiple-choice question. "What did someone bake?" 1. table 2. cake.

(If I had word embeddings, I could know bake and cake are more related.) We use state-of-the-art word embeddings but they don't magically solve the problem; you need commonsense. Get jugments and shove it back into the model.

Pipeline:

- CCG parser (candidate dependencies from n-best list)
- question generator (heuristic)
- crowdsourcing platforms
- reparse with constraints, ex. Cpos (bake→ cake)
- Re-parsed CCG dependency tree.

After you do reparsing, if the analysis is better, this provides a new labeled training example to train a better parser. The gains weren't big enough for us.

Only 10% data was changed. We need more coverage by the heuristic question generator.

Heuristics look at properties about CCG grammar. Come up with tuples. Collapse tuples. We have many different scores. Is the question grammatical, what is the entropy over results, etc. Combine all together. This drove annotation process.

People can give more than 1 correct answer.

This is fast and cheap. We get long-distance dependencies. Sometimes there are 2 attachments. Coreference is tricky: people give antecedents. (Semantically this is fine for end tasks. For improving CCG parse, this would be a disaster.)

We also had Turkers look at bio texts.

We only changed 10% of sentences; on those we had respectable gains.

If we can label sentences for 50c each (as in here), we can get magnitudes more data than we have right now.

Can we get the crowd to generate sentences? Algorithmically produce fake sentences? This is very difficult. If you change distribution of natural sentences, you trained on stuff not in natural text.

Curriculum learning—start with simple sentences? Trying to get a better and better parser, this is the way to go.

How to get to general domain parsers? One first step is to make a corpus with many domains. Get 20, 50 domains; do cross-validation... There hasn't been a focus on it. It's

also possible we don't have the representational capacity—it's hard to get commonsense knowledge inside; you may need external information.

Before we can get this massive dataset, I can provide a tool, turnkey: get anyone who understands the text, train with several thousand examples.

In vision there is a lot of focus on transfer learning, learn general-purpose representations and then fine-tune. Is there a similar story here? I don't know. My cynical view is that it's a data issue. People do use parsers but it tends to be a struggle.

17 Power of Active Sampling for Unsupervised Learning (Aarti Singh, Carnegie Mellon University)

Abstract: Most modern datasets are plagued with missing data or limited sample sizes. However, in many applications, we have control over the data sampling process such as which drug-gene interactions to record, which network routes to probe, which movies to rate, etc. Thus, we can ask the question ? what does the freedom to actively sample data in a feedback-driven manner buy us? Active learning tries to answer this question for supervised learning. In this talk, I will present work by my group on active sampling methods for several unsupervised learning problems such as matrix and tensor completion/approximation, column subset selection, learning structure of graphical models, reconstructing graph-structured signals, and clustering, as time permits. I will quantify the precise reduction in the amount of data needed to achieve a desired statistical error, as well as demonstrate that active sampling often also enables us to handle a larger class of models such as matrices with coherent row or column space, graphs with heterogeneous degree distributions, and clusters at finer resolutions, when compared to passive (non-feedback driven) sampling.

Despite the availability of big datasets, we can seldom hope to measure our systems everywhere and all the time. For example there is too much data in computer networks (too many node), plenoptic camera (record spectral characteristics, etc. of materials), and smart cities. When we have a limit on how much data we can collect, what do we do?

Differentiate

1. passive sampling: data is drawn uniformly at random, or selected prior to observing any data (experimental design). Ex. given graph of connections, choose where on the graph to sample.

2. active sampling: selective data drawn sequentially in a feedback-driven way. (For many applications we have control over what data to acquire.) Ex. select which nodes to ping, which biomedical experiments to do.

Tradeoffs in statistical learning are

1. statistical efficiency (error, risk, noise tolerance)

2. measurement efficiency

3. memory efficiency

4. computational efficiency
5. communication efficiency
6. model assumptions

How can active sampling enhance these tradeoffs?

Most of active learning has been in the context of supervised problems (classification, regression), whether to collect the label for a data point.

If there is (unknown) heterogeneity, active methods can learn about it using feedback and adapt the sampling. (If heterogeneity is known, you can just design the sampling at the beginning.) In heterogeneity, points near decision boundary are more important. Active sampling helps for piecewise smooth functions

What heterogeneities might exist and how to exploit them?

We talk about 3 problems and their heterogeneities.

1. Graphical model structure learning: varying node degrees.
2. Matrix, tensor completion and approximation, column subset selection: spiky columns or rows
3. Hierarchical clustering: multi-resolution clusters

17.1 Graphical model structure learning

We want to learn conditional independence relations between variables (ex. sensors, protein interaction network) using few samples/measurements.

Graphical models encode

$$\{i,j\} \notin E \iff X_i \perp X_j | X_{[p]\setminus\{i,j\}}.$$

Two nodes are not connected by an edge iff they sare independent given all others. Gaussian graphical models have $X_1, \ldots, X_p \sim N(0, \Sigma)$ iid.

$K = \Sigma^{-1}$ is sparse with zeros encoding absence of edges (conditional independencies).

Lower bound (Wang-Wainwright-Ramachandran10): Number of passive samples necessary is $np = \widetilde{\Omega}_{d_{max}}(d_{\max} p \ln p)$.

Upper bound (Meinhausen-Buhlmann06, Wainwright09): $np = I(d_{\max} p \ln p)$. Use node-wise lasso.

Passive sampling requires sampling from the joint distribution. Collecting these samples is expensive: there is a cost of measurement and synchronization. In practice, you may not be able to do this.

Active sampling: Select a subset and sample from the marginal,

$$\{X_{S_l}^{(i)}\}_{i=1}^{N_l} \sim N(0, \Sigma_{S_l})$$

Get $\sum_l N_l |S_l|$ samples.

First take a small number of samples from all variables, get an estimate of their neighborhoods. For the ones whose neighborhoods you learn, use half your budget to learn it and half to refine, then stop sampling from them.

Initialize NBDFound, Settled, $D_1, D_2 = \phi$, $l = 1$. Obtain $N = cl \ln p$ samples from $X_{[p]\setminus(\text{Settled})}$ and add to D_1¡ repeat for D_2. Sequentially for each $j \in [p]\setminus$ (NBDFound),

1. learn neighborhood: identify top $\leq l$ size neighborhood of j using D_1 via lasso
2. verify neighborhood: if given learnt neighborhood, the partial correlation (computed with D_2) of j with remaining nodes $\leq \xi$, add j to NBDFound.

For $j \in$ (NBDFound) if all neighbors of j are in NBDFound, add j to Settled.

You get more samples from the hub nodes.

Let $d_{\max}(j)$ be the max degree of any neighbor of node j. The number of active samples sufficient is

$$\sum_l N_l |S_l| = O\left(\sum_{j=1}^{p} d_{\max}(j) \ln p\right)$$

The number of samples of a node needed is adapted to its local degree. This is good for skewed degree distributions and minimizes need for synchronized measurements of variables.

In experiments we see improvement for power law (preferential attachment) graphs.

We have a lower bound with average degree (rather than average max degree of neighbor).

17.2 Active sampling for matrix completion

Complete or approximate a matrix by a low-rank matrix given feedback-driven choice of which entries to observe.

Usual assumption is that the matrix is both row and column incoherent. Active sampling helps when it is spread out in 1 direction, ex. column incoherent. For a k-dimensional subspace $U \subseteq \mathbb{R}^n$,

$$\mu(U) = \frac{n}{k} \max_{i \in [n]} \|P_U e_i\|_2^2 .$$

Here $\|P_U e_i\|_2$ is unnormalized leverage score of ith coordinate if U is top k principal components. Incoherence means that energy in the space is uniformly spread out amongst coordinates.

Lower bounds due to Candes-Tao10: The number of random entries necessary to recover $n \times n$ matrix of rank r with row and column space coherence bounded by μ scales as $\mu r \ln n$ per column.

Upper bound (Yudong-Chen15): the number of random entries sufficient is $\mu r (\ln n)^2$. Use nuclear norm regularization, $\min_X \|X\|_*$, convex surrogate.

What does active sampling do. Initialize subspace $A = \{0\}$.

1. Randomly draw m entries from column.
2. If column doesn't lie in space spanned by A, observe all entries, add to A. (Test using partially observed column, but fully observed subspace.)

3. Else complete it by projecting onto current A.

If $\mu \asymp \mu r(\ln\left(\frac{r}{\delta}\right))^2$ where column space has incoherence $\leq \mu$, then algorithm recovers wp $\geq 1-\delta$ using $n_2 m + r n_1$ entries ($\sim n\mu r(\ln r)^2$).

Measurement complexity goes from $\mu r(\ln n)^2$ to $\mu r(\ln r)^2$, independent of matrix size. Computational complexity goes from $n^2 r$ (doing things with singular values) to $(nr^2 + r^4)(\ln r)^2$ (sequential) If learns in 5 minutes rather than 2 hours! It does sequential column/row processing. Memory complexity goes from n^2 (which can be improved) to nr. The most incredible is that you've increased the class of matrices you can handle. Random sampling requires incoherent rows and columns; active sampling only requires incoherent rows OR columns. It focuses on harder examples.

Lower bound for passive sampling: number of passive entries necessary to recover $n \times n$ matrix of rank r with only column space coherence bounded by μ scales as n per column.

For row coherent matrices, random sampling requires n per column; we only require $\mu r(\ln r)^2$ per column.

What if matrix is not exactly low-rank? I'll show this in the context of column subset selection. Can I find columns that give the best rank k approximation? (Find features that are most representative.)

We get a a $1+3\varepsilon$ approximation with respect to best residual norm. A lot of matrix approximation results get additive error. The sample complexity is similar to passive matrix completion, but the error bound is relative. We only assume column space incoherence.

17.3 Hierarchical clustering

Cluster objects given feedback-driven choice of whichpairwise similarities to observe.

For hierarchical clusters, need $n \ln n$ similarities; for heterogeneous clusters, just need nk^2 similarities. "Break the $\sqrt{n}$ barrier for stochastic block models."

18 Leveraging Union of Subspace Structure to Improve Constrained Clustering (Laura Balzano, University of Michigan)

Abstract: Many clustering problems in computer vision and other contexts are also classification problems, where each cluster shares a meaningful label. Subspace clustering algorithms in particular are often applied to problems that fit this description, for example with face images or handwritten digits. While it is straightforward to request human input on these datasets, our goal is to reduce this input as much as possible. We present an algorithm for active query selection that allows us to leverage the union of subspace structure assumed in subspace clustering. The central step of the algorithm is in querying points of minimum margin between estimated subspaces; analogous to classifier margin, these lie near the decision boundary. This procedure can be used after any subspace clustering algorithm that outputs an affinity matrix and is capable of driving the clustering error down more quickly than other state-of-the-art active query algorithms on datasets with subspace structure. We

demonstrate the effectiveness of our algorithm on several benchmark datasets, and with a modest number of queries we see significant gains in clustering performance.

19 Talks 2/16/17

19.1 Corralling a Band of Bandit Algorithms (Haipeng Luo, Microsoft Research)

MSN gives personalized news recommendations, through contextual bandits. It looks at a user's profile and site information to select what it things you like. It uses user feedback to learn about preferences and generalize.

There are many contextual bandit algorithms. Which one should I use?

- No one single algorithm is guaranteed to be the best.
- Naive approach: try all and pick the best. This is inefficient, wasteful, and nonadaptive.
- Hope: create master algorithm that selects base algorithms automatically and adaptively on the fly, performing closely to the best in the long run.

In the full information setting, run the "expert" algorithm like Hedge. But in the bandit setting, we can't do this. At first it looks like a multi-armed bandit algorithm, can we use EXP3?

There is a serious flaw. The regret guarantee is only about the actual performance. The performance of base algorithms are significantly influenced due to lack of feedback.

Ex. If algorithm 1 does better on the first few runs, the master algorithm chooses algorithm 1, and algorithm 2 doesn't get the information it needs.

Right objective: Perform almost as well as the best base algorithm if ti was run on its own.

Difficulties:

- worse performance $\leftrightarrow$ less feedback
- requires better tradeoff between exploration and exploitation.

Previous work: EXP3 with higher uniform exploration (Maillard, Munos (2011)), but get $T^{\frac{2}{3}}$ regret.

We give a novel algorithm with more active and adaptive exploration, almost same regret as base algorithm

Two applications:

- exploit easy environments while keeping worst case robustness
- select correct model automatically.

General bandit problem: for $t = 1 : T$ do

- environment reveals side info $x_t \in X$
- Player picks $\theta_t \in \Theta$.
- environment decides loss function $f_t : \Theta \times X \to [0, 1]$.
- Player suffers and observes $f_t(\theta_t, x_t)$.

Contextual bandits: x is context, $\theta \in \Theta$ policy, $f_t(\theta, x)$ loss of arm $\theta(x)$.
Environment is iid adversarial or hybrid.
Pseudo-regret is difference between attained and best in hindsight.
Given M base algorithms B_i, giving suggestions θ_t^i, create master algorithm.
Suppose running B_i alone gives $REG_{B_i} \leq R_i(T)$. When running master with all base algorithms, we want

$$REG_M \leq O(\text{poly}(M)R_i(T)).$$

Ex. create one algorithm with worst-case guarantee, but can exploit the environment when easy.

This is impossible in general: the guarantee when run separately tells you nothing about what happens when run with master. We need more assumptions.

Typical strategy:

- sample base algorithm $i_t \sim p_t$.
- feed importance-weighted feedback to all B_i, $\frac{f_t(\theta_t, x_t)}{p_{t,i}} \mathbb{1}_{i=i_t}$.

Assume B_i ensures $REG_{B_i} \leq \mathbb{E}[\left(\max_t \frac{1}{p_{t,i}}\right)^{\alpha_i}] R_i(T)$. $p_{t,i}$ is the probability that the master algorithm picks t at step i. We want $REG_M \leq O(\text{poly}(M)R_i(T))$. EXP3 still doesn't work.

3 ingredients

- Special OMD (online mirror descent): want $\frac{1}{p_{t,j}}$ to be small.

  In EXP3 with Shannon entropy, $\frac{1}{p_{t,i}} \approx \exp(\eta \,(\text{loss}))$. Look at other entropies that make this smaller than exponential. Use log barrier as mirror map $-\frac{1}{\eta}\sum_i \ln p_i$ to get $\eta\,(\text{loss})$. This algorithm provides the least extreme weighting.

- Increasing learning rates schedule. We need to learn faster if a base algorithm has a low sampled probability.

  We don't want the master to converge or be overcommitted. Allow individual learning rates $\sum_i \frac{-\ln p_i}{\eta_i}$ and increase learning rate when $\frac{1}{p_{t,i}}$ is too large.

19.2 The End of Optimism? An Asymptotic Analysis of Finite-Armed Linear Bandits (Csaba Szepesvari, University of Alberta)

With Tor Lattimore, AISTATS 2017.

One standard design for sequential decision making under uncertainty is optimism.

I show that the two standard design principles for stochastic multi-armed bandits have serious drawbacks beyond the simplest case, and offer some alternatives.

Linear bandits: actions A, for each $x \in A$ reward distribution P_π. In each round choose $A_t \in A$, observe $Y_t \sim P_{A_t}$, maximize reward, total rounds n.

Ex. $A = [k]$ and $Y_t \sim B(\mu_{A_t})$ with $\mu \in [0,1]^k$.

Ex. $A \subseteq \mathbb{R}^d$ and $Y_t = \langle A_t, \theta \rangle + \eta_t$ with $\theta \in \mathbb{R}^d$ and η_t noise. Let $\mu_x = \langle x, \theta \rangle$, $\mu^* = \max_{x \in A} \mu_x$, immediate regret $\Delta_x = \mu^* - \mu_x$.

Regret is $R_n = n \max_{x \in A} \mu_x - \mathbb{E}\left[\sum_{t=1}^n \mu_{A_t}\right] = \mathbb{E}\left[\sum_{t=1}^n \Delta_{A_t}\right]$.

We concentrate on asymptotics. Strategy is consistent if $R_n = o(n^p)$ for $p > 0$ (subpolynomial).

How small can we make the regret? Optimism for linear bandits: in each round construct confidence set $C_t \subseteq \mathbb{R}^d$ such that $\theta \in C_t$ whp. Then choose

$$A_t = \operatorname{argmax}_{x \in A} \max_{\widetilde{\theta} \in C_t} \left\langle x, \widetilde{\theta} \right\rangle .$$

Whp,

$$\Delta_{A_t} \leq \left\langle A_t, \widetilde{\theta} \right\rangle - \langle A_t, \theta \rangle = \left\langle A_t, \widetilde{\theta} - \theta \right\rangle ,$$

width of confidence set in direction A_t. The width is decreased in this direction. Gram matrix summarizes information you know about unknown parameter.

The regret of OFUL is bounded by $O(d\sqrt{n \operatorname{poly} \log n})$. This almost matches lower bound $\Omega(d\sqrt{n})$ so it looks like we are done.

Worst-case obscures instance-dependent structure.

Lower bound: for any consistent strategy, confidence weights do not decrease too fast:

$$\limsup_{n \to \infty} (\ln n) \left\| x \right\|_{G_n^{-1}}^2 \leq \frac{\Delta_x^2}{2}$$

for all $x \in A$. Corollary: lower bound on regret

$$\limsup_{n \to \infty} \frac{R_n}{\ln n} \geq c(\theta, A)$$

where $c(\theta, A) = \inf_{\alpha \in [0,\infty)^k} \sum_{x \in A} \alpha(x) \Delta_x$ subject to $\left\| x \right\|_{H_n^{-1}}^2 \leq \frac{\Delta_x^2}{2}$, $H_\alpha = \sum_{x \in A} \alpha(x) x x^T$. It has to prove it knows which actions are suboptimal.

Upper bound: there exists a strategy that matches, $\leq$.

Why does optimism fail? Learning is slow once an action is not played; need $\Omega\left(\frac{\ln n}{\varepsilon^2}\right)$ plays of x to learn sub-optimal. Regret is $\Omega\left(\frac{\ln n}{\varepsilon}\right)$ compared to optimal $O(\ln n)$. The algorithm should reason about info gain about actions that have been shown suboptimal!

Thompson sampling fails the same way.

Algorithm:

1. Find barycentric spanner $B \subseteq A$. Choose $x \in B$ $\left\lceil \ln(x)^{\frac{1}{2}} \right\rceil$ times.
2. Anomaly detection: solve optimization problem to plan exploration. Loop as long as new observations are not too consistent with $\hat{\Delta}$.

3. Recover: switch to UCB

We need practical optimal algorithms, finite-time guarantees...
Infinite-action spaces?
Trading regret for information?

19.3 RL of Partially Observable Environments Using Spectral Methods (Kamyar Azizzadenesheli, UC Irvine)

RL: environment-agent interaction. At each step, environment gives state, agent chooses action, gets reward, get new state, etc. In RL the agent uses this information (reward, observation) to reinforce policy. Try to maximize reward. Evaluate performance by regret.

Fully observable models: playing maze observing map, playing video game with access to state of emulator, and navigation with access to map. Here state equals observation. This is MDP which has been studied a lot. UCRL: whp regret bounded by

$$Reg_N = \widetilde{O}(DY\sqrt{AN}),$$

$D = \max_{y,y'} \min_\pi \mathbb{E}[T(y \to y')|\overline{M}, \pi]$ (diameter, some measure of connectivity). This is linear in number of observations. In some environments, the number of observations might be large. In large MDP's, suffering from linear regret is not practical.

Structured MDP: the space of observations when navigating is large; the 2d location is enough. Solve in the smaller MDP!

Partially observable MDP: Ex. maze: Instead of observing whole game, observe window. In video game, just have access to screen. In self-driving, get sensory observation of environment.

Instead of a clean mapping, we have a messy mapping: given observation, we cannot infer hidden state.

Most problems in RL are POMDPs.

Randomly generate: 2 hidden, 4 observed, 2 actions: MDP algorithms fail.

Contextual MDP: We have sufficient information to infer hidden state. At some point the agent learns the mapping. In RL, at the beginning we don't have access to the mapping. What if we learn it?

If we partially learn the mapping, we can construct a smaller auxiliary MDP. How to even partially learn the mapping? We need to learn the latent structure of the model. Use the spectral method.

Initialize with policy. Use spectral method to cluster the contexts. The agent uses the optimism principle. Apply the new policy for a longer time, etc.

This algorithm reaches regret which is linear in the number of hidden states, not observed states. Connectivity is in terms of the hidden states.

Applying this to a simple MDP, $X = 4$, $Y = 20$, $A = 4$, we got better performance. The algorithm quickly converges to the true MDP.

POMDP: SM-UCRL-POMDP algorithm: Estimate parameters. Choose policy wrt model (optimism).

Experiment on GridWorld.

19.4 Safety in Exploration (Andreas Krause, ETH Zurich)

Trying to push RL closer to real world: driving, treating patients, etc. How do we think about safety in exploration? How can learning system autonomously explore while guaranteeing safety?

Think about optimizing reward function

$$\max_x f(x).$$

We can add a constraint, $g(x) \geq \tau$. If we know g we can restrict the set of actions to meet those constraints. What happens if we know neither f nor g? We interactively experiment with environment to learn both f and g from noisy data.

Simplifying assumption: Reward and constraint function are the same. Assume f is smooth, and you have a safe seed x. What can you hope to get? We want to explore to get to global optimum but maybe that is too much to ask. We care about reachable optimum, in the connected component of x in the feasible set.

Starting point is Bayesian optimization, useful paradigm for interactive learning. Endow function f with a prior, nonparametric like Gaussian process. It's important to keep track of uncertainty.

$$y_t = f(x_t) + \varepsilon_t.$$

Use a acquisition function. Mostly heuristic, but theory has been built up in recent rates.

Suppose we have confidence intervals around the function. Best must be where possible value is above the best lower bound. UCB$\geq$ best lower bound. Statistically certify that an action is safe.

Maximize acquisition function over certified safe domain.

This simple modification doesn't work, you get stuck in local optimum. You have no incentive to explore at the boundary!

Collect information not just to trade exploration/exploitation in area that's safe; collect info to certify a larger area is safe.

Maintain classification of decision set. Keep track of potential expander: points where there's a reasonable chance that if you make an observation there, you can certify a larger region is safe. Do uncertainty sampling in this region. Pick the most uncertainty. Keep expanding.

Theorem 19.1. *Under suitable conditions on the kernel and on f there is a function $T(\varepsilon, \delta)$ such that for any $\varepsilon, \delta > 0$, it holds wp $\geq 1 - \delta$ that SAFEOPT*

1. *never make unsafe decision*
2. *after at most $T(\varepsilon, \delta)$, it found an a ε-optimal reachable point*

Ex. quadrocopter, without crash into wall.

Extensions: beyond contextual bandits (NIPS16), combining virtual and physical experiments (ICRA17), guarantee stability of nonlinear dynamical systems (CDC16).

19.5 PAC Partially Observable RL (Emma Brunskill, Carnegie Mellon University)

Suppose the reward are a function of the state. Our goal is to maximize expected reward even though we can't directly observe the world. Policies are mappings from histories to actions.

In PAC, on all but the (sample complexity number of steps/decisions), the algorithm selects an action for the current state that is at least ε-optimal. We have a budget of mistakes.

A POMDP is a latent variable model.

Control research: identify system and then act.

Bayesian learning: We can be Bayesian about everything. If we can do full horizon planning, and maintain a distribution of worlds, we can choose the optimal tradeoff between exploration and exploitation. This is computationally intractable. Because of approximations, actual algorithms lose all guarantees.

We consider full-memory policies.

Prior work converts to a fully observable model. We can treat the history as a state, but the space is too large. This yields exponential bounds on data needed.

We leverage progress on latent variable model estimation. This provides finite sample accuracy bounds on estimated parameters.

But latent variable models haven't been used in control settings.

We can take the same action multiple times for many episodes and use tensor decomposition to learn HMM parameters.

Problem with tensor decompositions; The way it labels the hidden states, may not be the same across actions.

We transform POMDP into a special form of HMM to solve. States in the HMM are pairs of states. Now learn the HMM.

Each episode, select trandomly for 4 actions and then use any policy. Estimate transformed HMM parameters using tensor decomposision.

On all but a polynomial number of steps, take ε-optimal action. (The bound has large exponents. It depends on constants depending on matrices we don't know in advance.)

This is to my knowledge the first PAC algorithm.

Assumptions:

- nonzero probability of being in any state in 2 steps from start of episode. (Very fast mixing. Doesn't work for e.g. reachability.)
- Full rank models: transition model full rank per action, observation and reward matrices full column rank.

  (If you have large observation cases, this will almost always be true.)

One important class is exploration POMDPs. The state is static, and we want to gather information. Transition matrix is the identity.

Potential improvement:

- Reduce computational expense

- Consider more general settings
- Beyond explore than exploit (this is too conservatie)

Contributions:

- first PAC RL algorithm for POMDP
- applies to important subclass of POMDPs
- exponential improvement in sample complexity

19.6 I-SED: An Interactive Sound Event Detector (Bongjun Kim, Northwestern University)

A speech and language pathologist wants to analyze the relationship between a kid's language development and their listening environment.

Record a kid 16 hours a day for a month.

I sit down in the lab to find an interesting sound event.

In Audacity, do manual annotation. But the audio track is days/weeks long!

I want an automated annotation tool. TotalRecall, Sonic Visualizer, ASAnnotation, LENA. But the sound event I'm interested in is not in their model.

Problem:

1. Predetermined sound classes or acoustic features
2. Too unrealiable for mission critical tasks. (LENA agrees with human annotators only 76% of the time.)
3. We do not have enough labeled training examples of the particular sound class (even hard to search).
4. The audio is credential (medical data) and we need expert-level ground truth annotation, which precludes crowdsourcing.

We need a tool (interface) that speeds up manual annotation of audio, allows us to define a target sound class on-the-fly, and does not require any knowledge about machine learning and audio signal processing.

In I-SED, I'm still doing manual annotations, but I'm helped by a machine. Machine gives a suggestion (fast) and the human gives feedback (accurate).

1. Select sound event by selecting region.
2. System does segmentation and feature extraction, and highlights the n closest regions.

$$Rel(s) = \frac{d(s, s_n)}{d(s, s_n) + d(s, s_p)}$$

where s_n is nearest negatively labeled, s_p is nearest positively labeled.

3. User adjusts region boundaries and labels these as positive/negative.

4. Update feature weight and relevance score.

(Why just give n closest? My goal is not to train classifier, just to finish the annotation.)
Experiment: people found events faster interactively than manually.

19.7 Data Dependent Hierarchical Interactive Classification Learning (Shai Ben-David, Univeristy of Waterloo)

There is an unknown probability distribution $X \times \{0,1\}$. We get samples $S \sim P^m$, $(x_1, y_1), \ldots, (x_m, y_m)$. We want $h : X \to \{0,1\}$ that has high probability of success.

Instead of picking samples randomly, we pick which one to label.

In active annotation, we assume that for the input, we only see $(x_1, \ldots, x_m)$. The algorithm can access a label oracle: what is the label y_i? The output is a set of labels $(x_1, A(x_1)), \ldots, (x_n, A(x_n))$.

The goal of the algorithm is to make sure that A errs on at most ε fraction of the labels. The algorithm can always achieve this by querying all the points.

We want to minimize the number of oracle queries.

The Dasgupta-Hsu algorithm is based on hierchical clustering. Ask $\frac{1}{\varepsilon}$ queries. If homogeneous, guess that every point there is the same cluster. If heterogenous, split into children nodes and go down the tree.

Keep going down until all leaves are heterogeneous.

This is a luckiness bound. If you were lucky, you stopped ahead of bound and saved queries. See Dasgupta's webpage.

We want to give success guarantees based on some assumptions.

1. Probabilistic Lipschitzness:

$$\mathbb{P}_{x \sim X}[\exists y, x', y' \text{ s.t. } (x,y),(x',y') \text{ violates } \lambda\text{-Lipschitzness}] = \phi(\lambda).$$

2. For nondeterministic labeling we introduce a more subtle measure.

$$\Gamma(n, x) = \mathbb{P}[B(x,r) : |\eta_B - \frac{1}{2}| < \tau]$$

There are few balls with heterogeneity.

These parameters can be thought of as formalization of the cluster assumptions. From this we bound the number of queries.

19.8 Learning with Feature Feedback: From Theory to Practice (Stefanos Poulis, UC San Diego)

Joint work with Sanjoy Dasgupta.

We look at human feedback beyond the label. Ex. Highlight keyworks in document. There is a lot of practical but little theoretical work.

We want to understand and quantify the benefits. What can we learn, how quickly?

In any labeling situation there is vagueness in intent of labeler. Ex. Document about politics. They highlight "Obama" but they may be thinking about a lot of words which co-occur with Obama.

The human sees documents but the computer may have a different representation.

We study 2 models.

1. Probabilistic generalization of disjunctions (or functions): PDM

   Instance space $X = \mathbb{R}^d$. Intermediate (topic) space $\Theta = [0,1]^T$, label space $Y = [k]$.

   Assume each topic has a label in $[k]$ or is ?.

   $P = \{t : l(t) \neq ?\}$ is the set of predictive topics.

   Define a generative model

   (a) $\theta = (\theta_1, \ldots, \theta_T)$ topic representation

   (b) Pick $t \in P$ wp $\propto \theta_t$.

   (c) Label is $\ell(t)$.

   How to learn this?

   Learning a PDM is (NP-)hard! Feature feedback makes learning tractable.

   Initialize n_{ty} for times that t received label y.

   Repeat: receive next x. Receive label y and predictive words $w_1, \ldots, w_c$. S =select-topics$(x, w_1, \ldots)$. For $t \in S$, increment counter $n_{ty} += 1$.

   Labeling: Look at how pure a label is. If $n_{ty} \geq \lambda \sum_y n_{ty}$ then label, else ?.

   For any $t \in P$, the expected number of documents is $O(|P| \ln(Tk))$

2. Linear separators.

$$\widehat{w} = \text{argmin}_w \frac{1}{n} \sum_{i=1}^{n} \ell(w \cdot x_i, y_i) + \eta \|w\|^2 .$$

   Incorporate feature feedback into regularization norm.

   Reduce degree of regularization on predictive features.

   Take $\|x\|_A = \sqrt{x^T A x}$, $A_{jj} = \frac{1}{c}$ for predictive features j, $c > 1$, $A_{jj} = 1$ otherwise.

   This has statistically guarantees.

   Let $R = \{j \in [p] : w_i^* \neq 0\}$ denote relevant features of target w^*. Write any x in terms of relevant and other components, $x = (x_R, x_o)$.

   For the family $F = \{w : \|w\|_A \leq \|w^*\|_A\}$, $R_n(F) \leq \|w^*\| \max_x \sqrt{\left(\frac{1}{c}\|x_o\|_2^2 + \|x_R\|_2^2\right)} \sqrt{\frac{2}{n}}$.

These are 2 types of feedback, vague and clear feature feedback.

In practice, combining is useful.

Future work: discriminative feedback.

19.9 Active Nearest Neighbors in Changing Environments (Ruth Urner, Max Planck Institute for Intelligent Systems, Tuebingen)

I'll ask a different question. Typically we get train and test data from one distribution. Here, use the active algorithm to adapt to a changing environment.

Can we use the labeled data and active algorithm unlabeled data from the new distribution to learn? Ex. ask labels about pictures that are different from what it has seen.

We developed a new learning method ANDA: nearest neighbor query rule and nearest neighbor prediction.

(k, k')-query rule: look at point in unlabeled same, look at k'-nearest label. Do we have k labels? If we do, then we are good. Otherwise, query.

Abstract out the combinatorial property of the label that it chooses.

R is (k, k')-NN cover for T if $\forall x \in T$, $x \in R$ or there are k elements from R amount k' nearest neighbors of $T \cup R$.

Given labeled S, unlabeled T, k, k', find $T' \subseteq T$ such that $S \cup T'$ is (k, k')-NN-cover of T. Query labels of points in T'. Output $h^l_{S \cup T'}$ the k-NN classifier on $S \cup T'$.

Same guarantees as if T had been completely labeled. We can also bound the number of labeled queries.

Key lemma: Let R be (k, k')-NN-cover for T. For every x the distance to the k nearest labels (in R) is at most 3 times the distance to the $k' + 1$ nearest target points.

Labeled data behaves as if it comes from the distribution. Use NN to get guarantees.

Converge to Bayes optimal. Doesn't have further assumptions on type of shift.

Correctness of ANDA does not depend on relatedness assumptions of source and target marginals. However, the number of queries ANDA makes depends on a local relatedness measure. (How bad was shift between label and target distribution.)

Define weight ratio $\beta(B) = D_S(B)/D_T(B)$.

With large enough sample size, ANDA-S will not query any points $x \in T$ with $\beta(B_{Ck,T}(x)) > w$.

If source and target distributions are the same, it will not query at all. Correctness always holds and the algorithms queries as much as it needs.

In the limit of large sample size, ANDA will not make any queries in the support of the source distribution.

Lower bounds: If restricted to limited queries, then bound to fail. No DA learner with a fixed query budget (passive learner) is consistent.

What other types of classification methods can we get similar adaptation behavior.

Finding the min cover is NP-hard. Use a greedy method. Bounds still work.

19.10 Interactive Learning Opportunities for the Air Force (Lee Seversky, Air Force Research Laboratory)

I'll focus on how we're using interactive learning, and some of the challenges we've faced.

1. We're data-rich but label-scarce. We have diverse platforms and modalities, continuous collection, and data at scale.

2. We have complex tasks and dynamic environments. We have time constraints (changing mission requirements). Lots of times, the problem space is under-constrained, ill-posed (notify me before bad things happen). Context matters in planning and execution.

3. We're human-driven. People are highly-trained and specialized. Military has a strong organizational hierarchy. Trust and policy: a human has to be involved to some degree.

Why?

1. Objective: decision making at scale and speed: decrease human resource requirements, reduce uncertainty.

2. Move from human in the loop to human on the loop to human outside the loop.

How do we get outside the loop as quickly as possible.

Highlights

1. Active learning for autonomous sUAV swarms. Surveillance: detect, plan and sense "interesting" objects.

   Do active learning classification on the system. You don't have constant communication. What classes of objects do we care about. Selectively query about objects.

   Model swarm similarity for control model selection.

2. Active classification/detection: image, signal

3. Image captioning

4. Active sampling (configuration and experimentation)

   Parameter space is large.

5. Triplet similarity learning

6. Active learning with side info

7. Zero-shot learning

8. Active learning

9. Learning by example

Challenges:

1. label scarcity: Go to the crowd. Use AMT, but we are limited to working with toy and public/open data.

   Look at the AF "crowd". There are 767,681 people in the AF. But all have full-time responsibilities; this is a very expensive crowd.

   People don't like nonsense data combinations, questions that are too easy.

2. data sensitivity

   Task and data is only accessible to small groups of users.

   Go from source domain (turkable) to target domain (sensitive).

   Do inductive transfer learning. This requires labeled data in the target domain.

   What tasks maximize transfer?

   How to actively select source and target labels jointly?

   Minimize source and target info exchange?

   LIDAR classification: classify interesting and not interesting LIDAR regions. Interesting is a function of current interests.

3. costs of learning

   There are many additional costs: acquisition, transfer, pre-processing, computation, user expertise.

Pool-based active learning algorithm: did we need every unlabeled instance in D_u?
Spatial active multi-classification: reduce number of labels and instances.
Fill in gaps: interpolate over domain.
Advanced Learning Algorithm Development `tinyurl.com/h2d7lgb`