Microsoft

Research Faculty Summit 2018

Systems | Fueling future disruptions

Microsoft

Neural networks and Bayes Rule

Geoff Gordon
Research Director, MSR Montreal
Joint work with Wen Sun and others

The “right answer” for inference

- Bayes rule
  - As implemented in graphical models
  - But, too expensive

- If we could do it, benefit: each node/edge has semantics
  - Helps model design, interpretation

OTOH, deep nets

- Efficient inference = simple matrix ops, fixed nonlinearities
- Efficient training = SGD FTW
- Not much semantics, but fast and successful

Can we get best of both worlds?

- Design deep nets that look more like graphical models (or vice versa)
- Want a model format that is both practical and "semantic"
- Take advantage of semantics for interpretation, model design, expressiveness, …
- Take advantage of SGD for performance on big problems

RNNs are Bayes nets already (sort of)

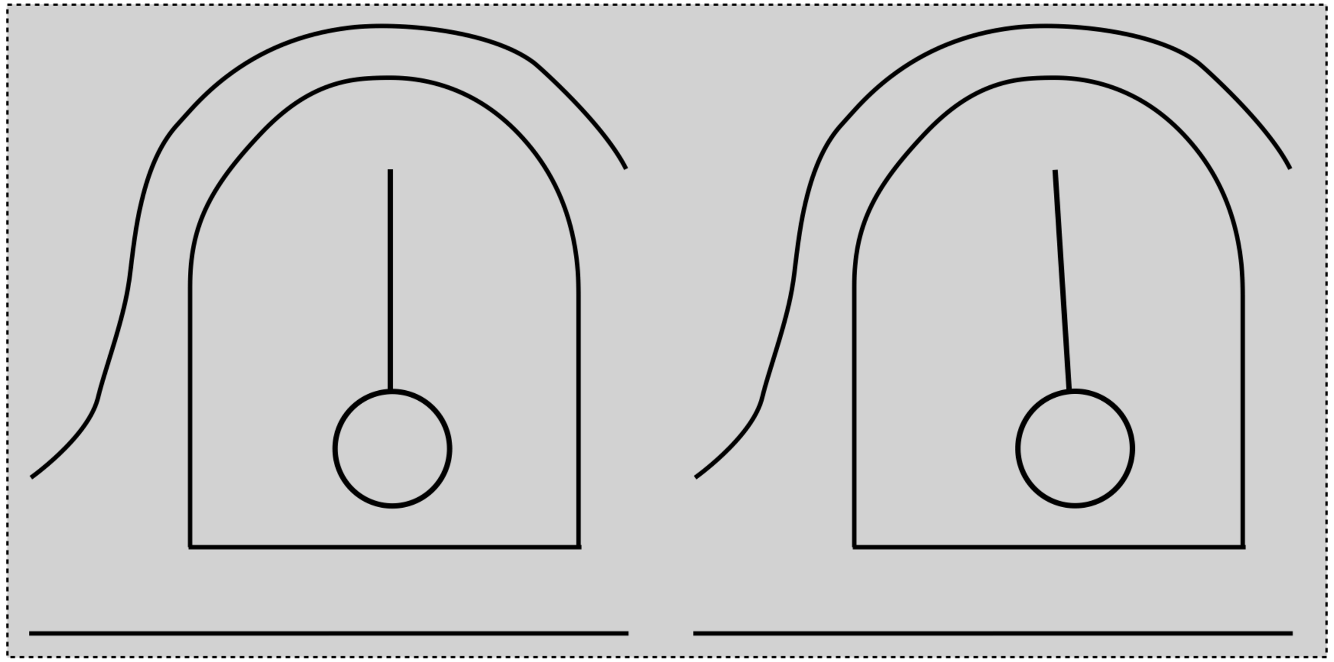
- Any RNN has to do approximate Bayesian inference (if it wants low loss)
- At each t, represents P(*future* | *history*) implicitly
  - E.g., can sample by rolling out
- Update rule has to implement approximate conditioning

Make implicit representation explicit

- In addition to predicting immediate next observation from latent state s_t,
  - Predict richer statistics of future
  - E.g., mean and covariance of observation features over next few steps
  - E.g., how many steps until we next see a 1
  - …
- If we use enough features, predictions are a 1:1 map from latent state
  - And therefore from predicted P(*future* | *history*)

- Called “predictive state”
  - A transformation of latent state to predictions about observables

Predictive state example



Predictive state example

$$\mathbb{E}(x_1 \mid s_1) = O s_1$$

$$\mathbb{E}(x_2 \mid s_1) = O T s_1$$

$$\mathbb{E}(x_3 \mid s_1) = O T^2 s_1$$

If this matrix has full column rank

$$\mathbb{E}\left(\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \middle| \; s_1\right) = \begin{bmatrix} O \\ OT \\ OT^2 \end{bmatrix} s_1$$

then s_1 is completely determined

so this vector is a state

Adding predictive state to an RNN

- … is an inductive bias
- … empirically helps prediction accuracy
- … but like all RNNs, serious worry about local optima

	Swimmer	HalfCheetah	Hopper	Walker2d	Walker2d†
TRPO	91.3 ± 25.5	330 ± 158	1103 ± 264	383 ± 96	1396 ± 396
TRPO + pred	$\mathbf{97.0 \pm 19.4}$	$\mathbf{372 \pm 143}$	$\mathbf{1195 \pm 272}$	$\mathbf{416 \pm 88}$	$\mathbf{1611 \pm 436}$
	$6.30\%^*$	$13.0\%^*$	$9.06\%^*$	$8.59\%^*$	$15.4\%^{**}$

Venkatraman et al. Predictive-State Decoders: Encoding the Future into Recurrent Networks. arXiv, 2018

Idea: bootstrap from supervised learning

- Empirically, many fewer worries about local optima for supervised learning
  - And theoretically, in simple cases (e.g., linear)
- We hope to borrow this property

- Hope: solve some supervised learning problems, get good weights for our deep net
  - then we can also run SGD to fine-tune these weights

Bootstrap outline

1. Predict future features directly from a fixed window of history
   - Supervised learning problem
   - But suboptimal: finite memory
2. Add [predicted future at time *t*] as input when predicting future for *t+1*
   - Chaining predictions allows infinite memory
   - To avoid introducing recurrence, use (fixed) predictions from a previous training iteration
   - Problem: training distribution changes across iterations
3. Fix the problem from step 2
   - imitation learning

Imitation for inference

- Inference is an RL problem (state = predictions so far, action = make another prediction conditioned on state, cost = sum of errors in predictions)
- Learning to do inference = finding a good policy
- Don’t need full RL: it’s much easier to imitate an “expert”
  - expert always gets its prediction from a labeled training set
- Which is good: unlike full RL, we can reduce imitation learning to supervised learning
  - via approximate policy iteration

(Exact) policy iteration

- Do at least once:
  - for all states s, actions a
    - calculate current total cost $Q^\pi(s, a)$, value $V^\pi(s) = E_{a\sim\pi(s)}[Q^\pi(s, a)]$, and (dis)advantage $A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s)$ // evaluate
  - choose $\pi^{new}(s) = \text{argmin}_a A^\pi(s, a)$ // improve

- Doesn't work in a real-size problem:
  - must sample (s, a) rather than iterating over all
  - can't calculate A^π exactly, must estimate somehow
  - can't choose new policy freely, must work in some hypothesis class

Approximate policy iteration (meta-algorithm)

- Do at least once:
  - estimate $A^{\pi}(s, a)$ // evaluate
  - update π^{new} to reduce $E_{new}[A^{\pi}(s, a)]$ // improve
- To instantiate: way to estimate $A^{\pi}(s, a)$, way to update π^{new}
  - also starting π, stopping criterion

Simple analysis of approximate policy iteration

- Guarantee: cost of π^{new} is $V^{\pi}(s_0)$ + T $E_{new}[A^{\pi}(s, a)]$
  - via performance difference lemma (simple proof: telescoping sum)
  - improvement when $E_{new}[A^{\pi}(s, a)] < 0$ (i.e., π improvable within hypothesis class, training succeeds)
- Difficulty: expectation is under distribution of (s, a) from π^{new} (not the distribution we used to collect data)
- Can we develop algorithms that guarantee improvement (w/ assumptions) despite this difficulty?
  - Yes…

DAgger

- Sample states according to expert policy
- Estimate A^{π} for all actions in current state (error to gold label)
- Generate training examples: $(s, a, A^{\pi}(s, a))$
- Train π^{new} by no-regret cost-sensitive classification
  - sadly, deep nets aren't no-regret

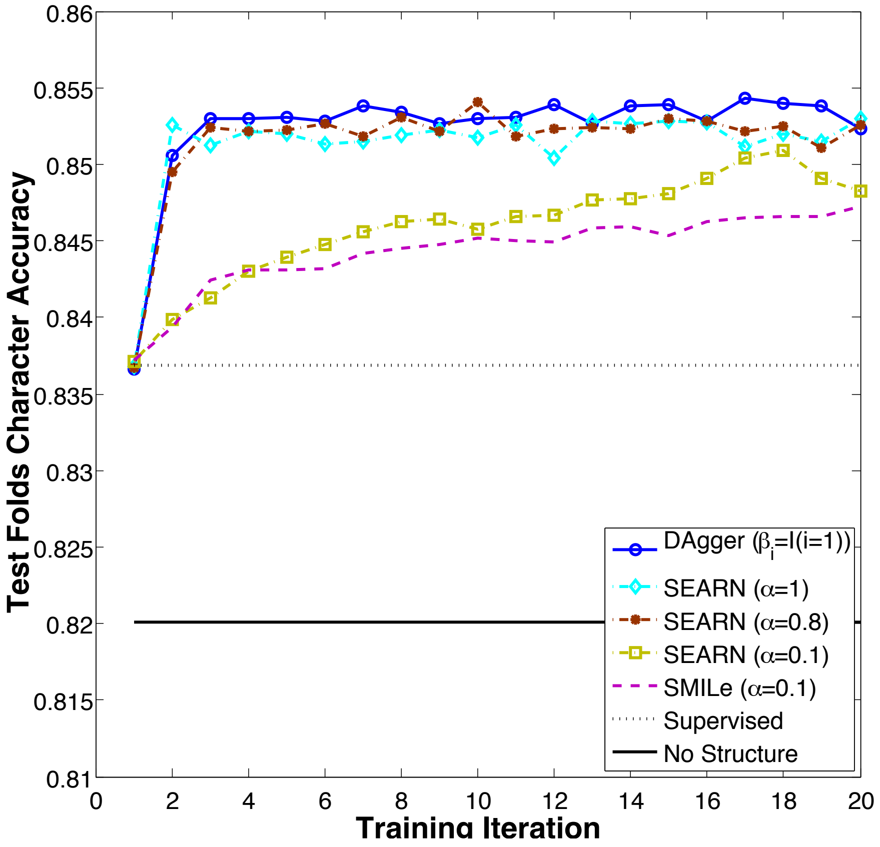
Ross, Gordon, Bagnell. A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. AISTATS, 2011

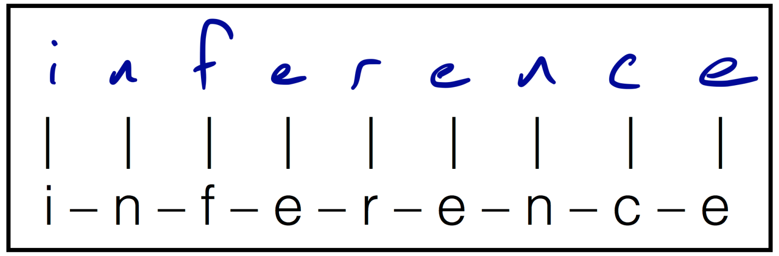
AggreVaTeD

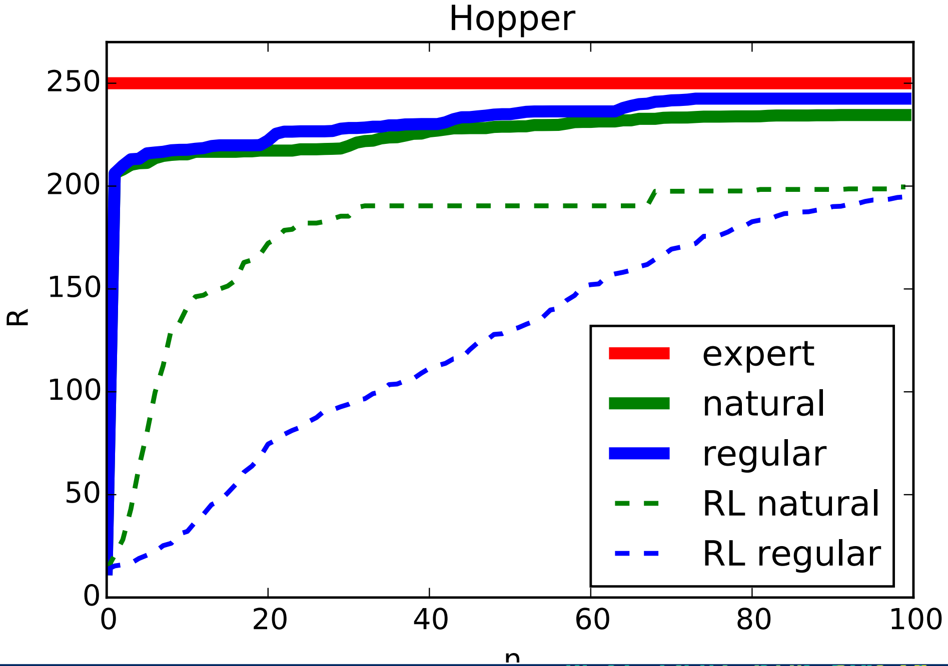
- Sample states according to expert policy
- Estimate A^{π} for all actions in current state (error to gold label)
- Update π^{new} by policy gradient (or natural gradient) to reduce cost
  - works for any differentiable policy, including deep nets

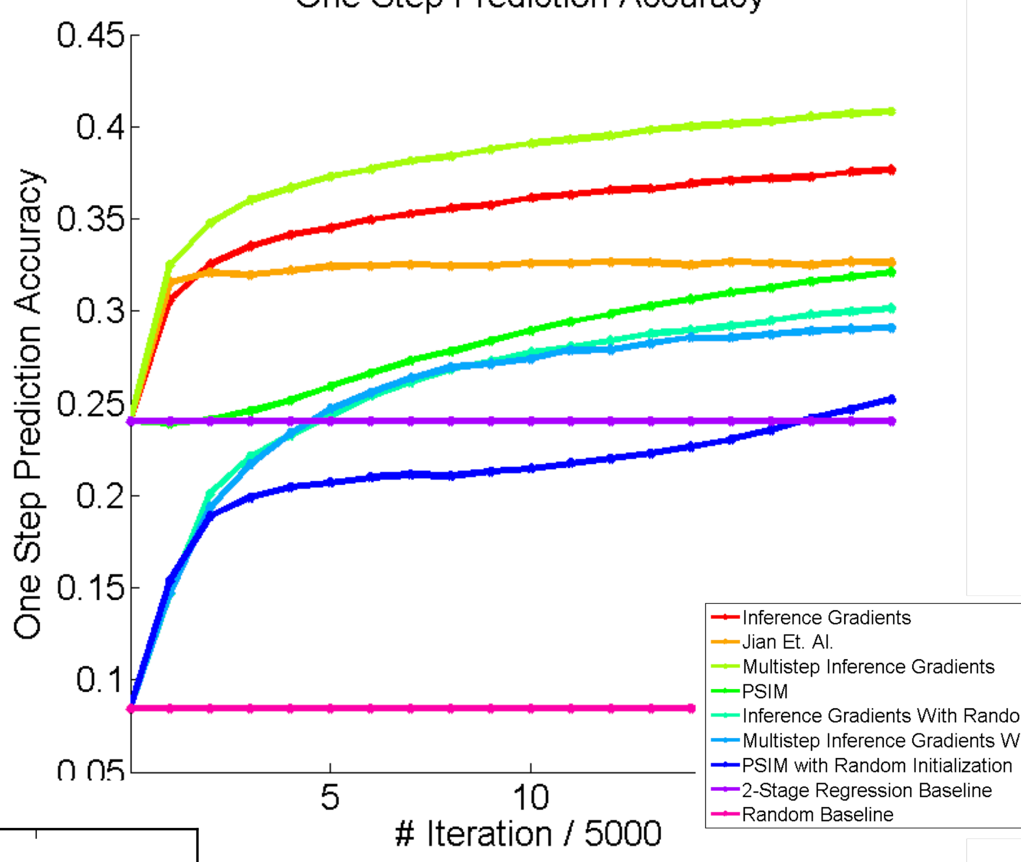
Sun et al. Deeply AggreVaTeD: Differentiable Imitation Learning for Sequential Prediction. arXiv, 2017.

Empirically, beats SGD w/ random init









Bonus: our network can explicitly encode Bayes rule

- Discrete observation x_t (as 1-hot vector)
- Choose future statistic ${}_t$ of the form $x_t \times \phi(x_{t+1:t+k})$
  - phi arbitrary, except should include a constant feature
- When predicting ${}_{t+1}$ from $\mathbb{E}(\psi_t)$ and x_t:
  - First layer: compute $x_t^T \mathbb{E}(\psi_t)$ then renormalize (using constant in ψ_t)
  - Remaining layers arbitrary
- Now can implement HMM learning and forward inference
  - use a single linear layer
  - If true model is an HMM, after learning, linear layer's parameters encode transition, observation probabilities

Thank you!

Microsoft