The speed of learning

CS440/ECE448 Lecture 34

Image CC-BY 2.0 https://commons.wikimedia.org/wiki/File:France-001675_-_Maze_(15291950978).jpg

Mark Hasegawa-Johnson, 4/2024

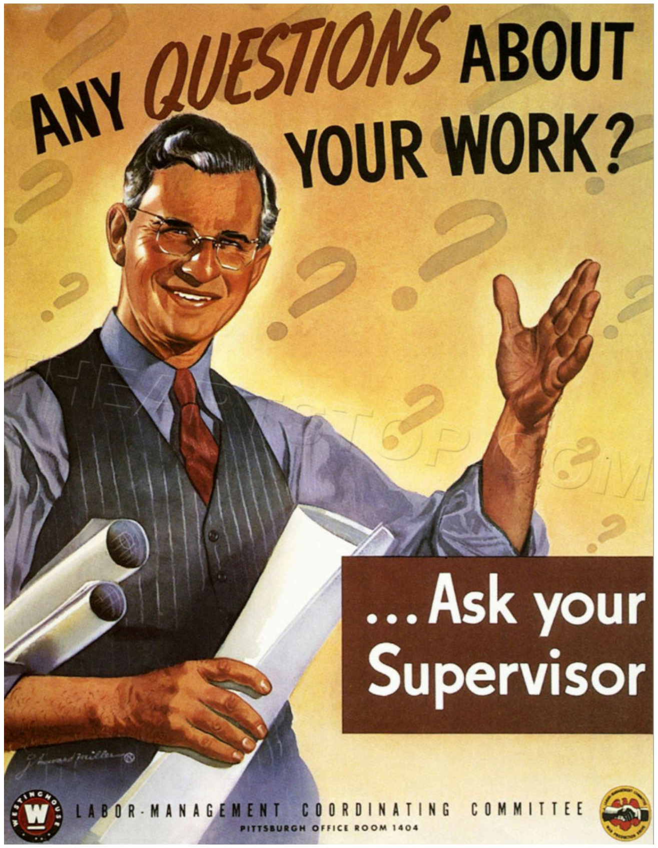
Outline

- Supervised learning
  - Imitation learning
- Unsupervised learning
  - Self-supervised learning
- Reinforcement learning
  - Experience replay buffer
  - Proximal policy gradient

Supervised learning

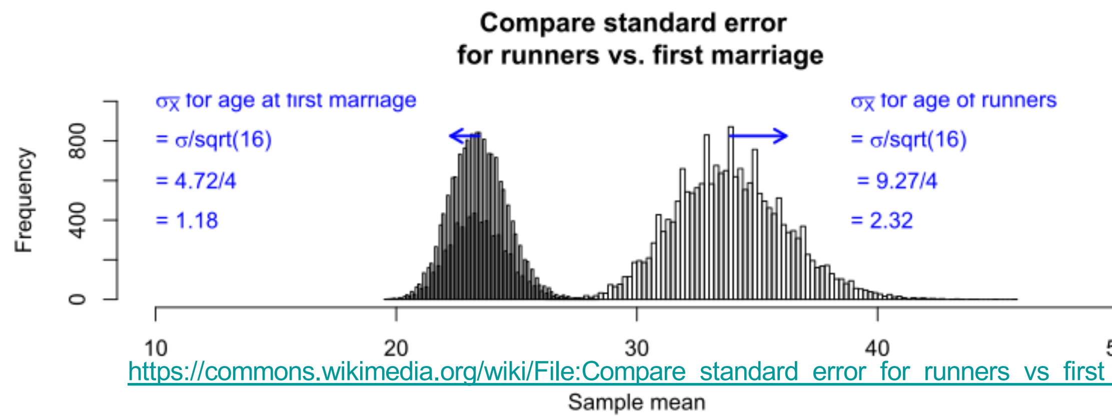
"Supervised" means that the learner is given a training database of paired examples, $\mathcal{D} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$, and is expected to learn the relationship between X and Y

- Linear or nonlinear regression: learn $f(X) = E[Y|X]$
- Linear or nonlinear classifier, or naïve Bayes: learn $f(X) = \underset{Y}{\operatorname{argmax}} P(Y|X)$



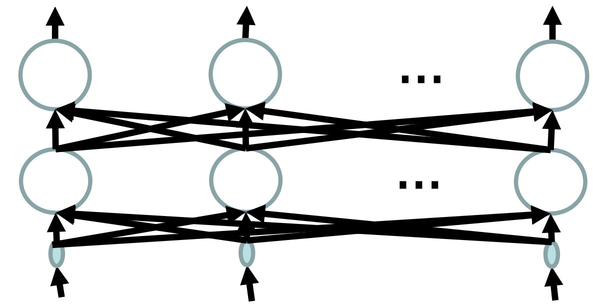
Public Domain, https://commons.wikimedia.org/w/index.php?curid=19686492

Standard error



- Suppose Y has mean μ, standard deviation σ
- The linear regression estimate of $E[Y]$ is $M = \frac{1}{n}\sum_{i=1}^{n} Y_i$
- M is a random variable, with $E[M] = \mu$, $\text{stdev}(M) = \frac{\sigma}{\sqrt{n}}$
- $\frac{\sigma}{\sqrt{n}}$ is called the "standard error" of this estimator. We can think of $\frac{1}{\sqrt{n}}$ as the rate at which this algorithm learns μ.

Neural nets



- Andrew Barron showed that the error rate of a neural net is

$$E = \mathcal{O}\left\{\frac{1}{N} + \frac{N}{n}\right\}$$

- N is the number of hidden nodes
- n is the number of training tokens
- By using $N = \sqrt{n}$ hidden nodes, we get

$$E = \mathcal{O}\left\{\frac{1}{\sqrt{n}}\right\}$$

Summary so far

- In many types of supervised learning, the error rate drops at a rate of $\frac{1}{\sqrt{n}}$ for n training tokens
- That's not too bad! Can we use that to teach a robot how to walk?

Imitation learning

- In some applications, you cannot bootstrap yourself from random policies
  - High-dimensional state and action spaces where most random trajectories fail miserably
  - Expensive to evaluate policies in the physical world, especially in cases of failure
- **Solution:** learn to imitate sample trajectories or demonstrations
  - This is also helpful when there is no natural reward formulation

Imitation learning

- $\boldsymbol{s}_t$ = a representation of the state of the environment at time t (can be a real-valued vector)
- a_t= the action that a human actor performed in response to this state (discrete)
- $f_k(\boldsymbol{s}_t)$ = k^{th} element in the softmax output of a neural network, given $\boldsymbol{s}_t$ as the input
- Training criterion: train the neural network to minimize

$$\mathcal{L} = -\log f_{a_t}(\boldsymbol{s}_t)$$

Outline

- Supervised learning
  - Imitation learning
- Unsupervised learning
  - Self-supervised learning
- Reinforcement learning
  - Experience replay buffer
  - Proximal policy gradient

Unsupervised learning

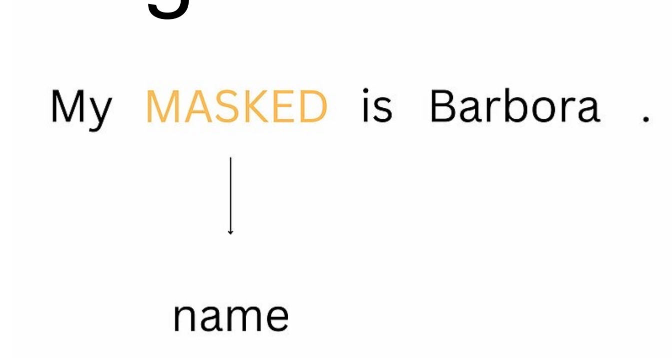
"Unsupervised" means that the learner is given only the observations, $\mathcal{D} = \{x_1, \dots, x_n\}$, and no labels (no Y).

- Hidden Markov model: learn to represent $P(X)$ using a hidden Y
- Skipgram and CBOW: learn a model of $P(X_{t+d}|X_t)$ or $P(X_t|X_{t+d})$

Self-supervised learning

"Self-supervised" means that we treat one part of X as the observation, and a different part as the label, then use a supervised method to learn the relationship. Examples:

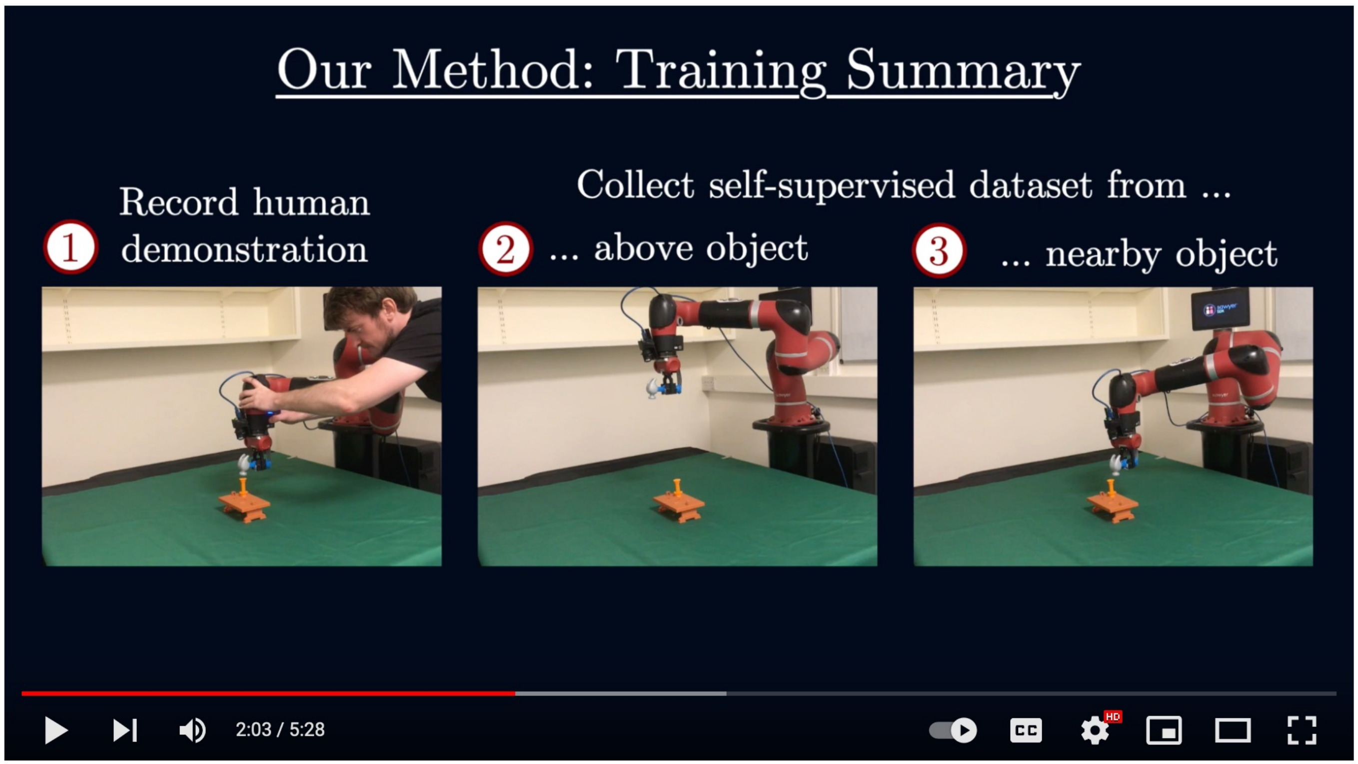
- Skip-gram and CBOW: learn $P(X_{t+d}|X_t)$ or $P(X_t|X_{t+d})$
- Autoregressive language model: learn $P(X_t|X_{t-d}, \dots, X_{t-1})$
- Masked language model: learn to use context to predict the masked words in a sentence or the masked pixels in an image



Does unsupervised learning converge?

- Yes! Error drops as $\frac{1}{\sqrt{n}}$, and n can be huge, because unlabeled data are very cheap! (The whole internet!)
- But: It only converges to a representation that works well for the unsupervised task (e.g., CBOW).
- Pre-training+Fine-tuning:
  1. **Pre-training:** Use unsupervised learning (lots of data) to train the first layers of a neural net
  2. **Fine-tuning:** Add one more layer, and use supervised learning (small dataset) to learn the output-layer weights, and adjust weights of the rest of the network

Example: Coarse-to-Fine Imitation Learning



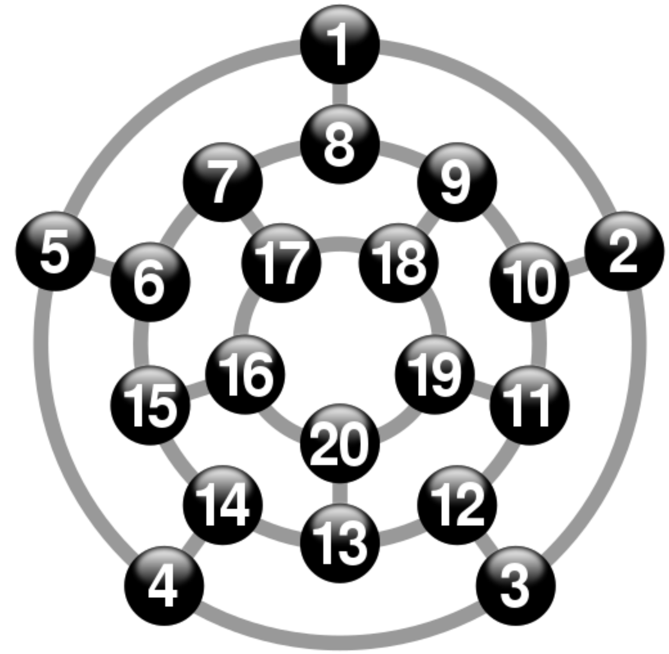
Edward Johns, [Coarse-to-Fine Imitation Learning: Robot Manipulation from a Single Demonstration](), 2021.

Outline

- Supervised learning
  - Imitation learning
- Unsupervised learning
  - Self-supervised learning
- Reinforcement learning
  - Experience replay buffer
  - Proximal policy gradient

Does reinforcement learning converge?

- You are in room 1, with three choices…
  - You are in room 8, with three choices…
    - You are in room 9, with three choices…
- Any room d steps from room 1 is explored once every b^d times you play the game
- …so the error is $\frac{1}{\sqrt{n/b^d}}$!



Try the quiz!

Try the quiz!

https://us.prairielearn.com/pl/course_instance/147925/assessment/2418125

One solution: Experience replay buffer

- Rollout:
  - Take action a_t according to current policy
  - Store experience $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$ in *experience replay buffer*

$(s_t, a_t, r_t, s_{t+1}, a_{t+1})$

$(s_1, a_1, r_1, s_2, a_2)$
$(s_2, a_2, r_2, s_3, a_3)$
...

$\mathcal{D}$ =sampled so that all combinations of (s_t, a_t) are in the minibatch

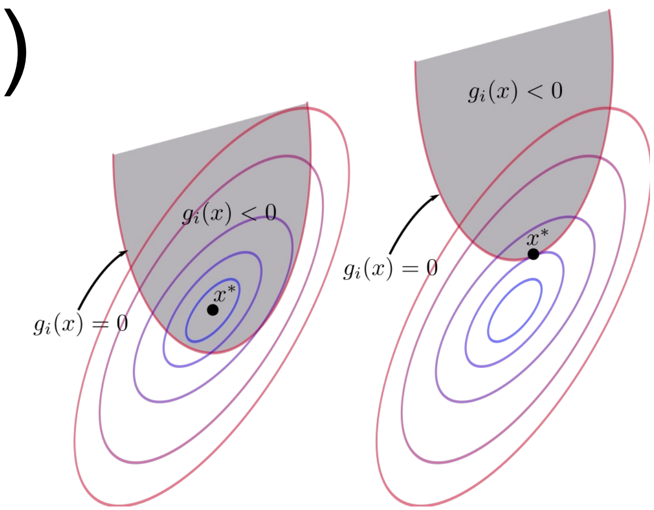
- Learning:
  - Sample a minibatch, $\mathcal{D}$, so that all combinations of (s_t, a_t) are in the minibatch
  - Train $\pi(s)$ to maximize utility

Another solution: Proximal policy optimization (PPO)

1. Use unsupervised and/or supervised learning (autoregressive language model, imitation learning, etc) to learn an initial policy $\pi_0(s)$
2. Use rollout to generate lots of trajectories, τ
3. Let humans reward the good trajectories, punish the bad ones, resulting in a utility estimate

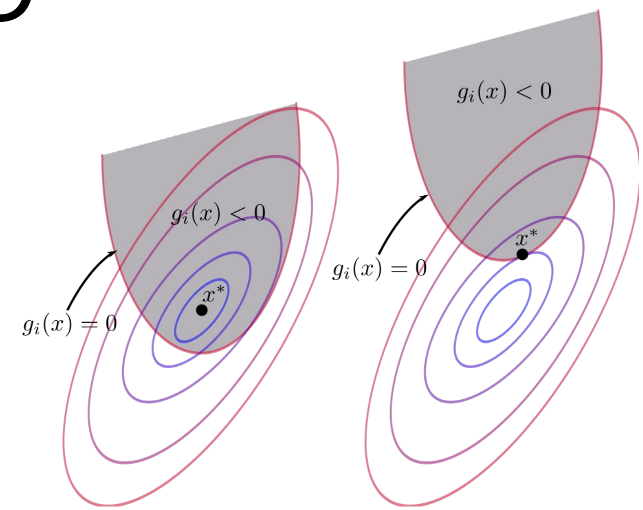
$$\frac{\partial u}{\partial \pi} = E\left[v(\tau)\frac{\partial \ln P(\tau)}{\partial \pi}\right]$$

1. Find π that
   1. Maximizes the utility, while also attempting to…
   2. …keep π from getting too far away from π_0.



Advantages of PPO

- Assumes that human rewards are good at predicting the value of small changes to the policy
- Assumes that big changes to the policy are undesirable, because the unsupervised & supervised training has learned a good starting policy



Outline

- Supervised learning learns $P(Y|X)$
  - Standard error: $M = \frac{1}{n}\sum_{i=1}^{n} Y_i$, $\text{stdev}(M) = \frac{\sigma}{\sqrt{n}}$
- Unsupervised learning learns $P(X)$
  - Self-supervised learning: skipgram, CBOW, autoregressive or masked
- Reinforcement learning: Any room d steps from the start is explored once every b^d times you play the game. Solutions include:
  - Experience replay buffer
  - Proximal policy gradient