

Topics in Reinforcement Learning:
Lessons from AlphaZero for
(Sub)Optimal Control and Discrete Optimization

Arizona State University
Course CSE 691, Spring 2023

Links to Class Notes, Videolectures, and Slides at
<http://web.mit.edu/dimitrib/www/RLbook.html>

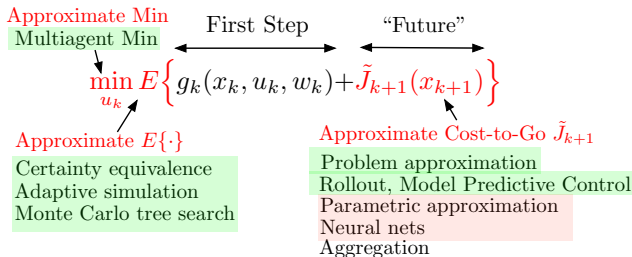
Dimitri P. Bertsekas
dbertsek@asu.edu

Lecture 11

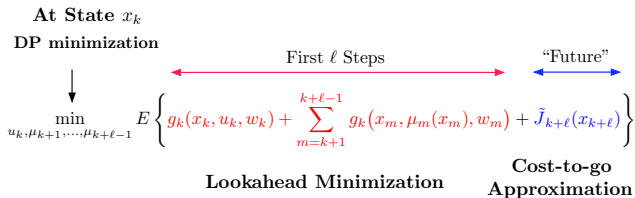
More on off-line training, parametric architectures, and
their use in approximate value and policy iteration
Aggregation - A different type of parametric architecture

- 1 Review of Off-Line Training with Parametric Architectures
- 2 Off-Line Training in Finite Horizon DP
- 3 Infinite Horizon - Approximate Policy Iteration
- 4 Introduction to Aggregation
- 5 Aggregation with Representative States: A Form of Discretization/Interpolation

Recall Approximation in Value Space

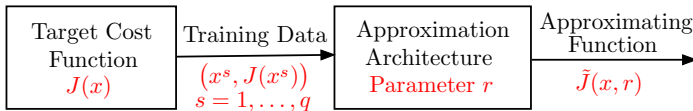


ONE-STEP LOOKAHEAD



MULTISTEP LOOKAHEAD

Parametric Approximation of a Target Cost Function



TRAINING CAN BE DONE WITH SPECIALIZED OPTIMIZATION SOFTWARE
SUCH AS
GRADIENT-LIKE METHODS OR OTHER LEAST SQUARES METHODS

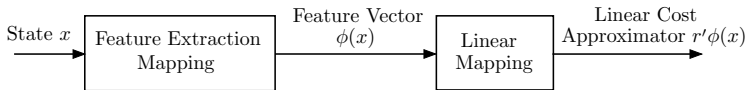
Cost Function Parametric Approximation Generalities

- We select a class of functions $\tilde{J}(x, r)$ that depend on x and a **vector** $r = (r_1, \dots, r_m)$ of m “tunable” **scalar parameters**.
- We adjust r to change \tilde{J} and “match” the training data from the target function.
- Architectures are called **linear or nonlinear**, if $\tilde{J}(x, r)$ is linear or nonlinear in r .
- Architectures are **feature-based** if they depend on x via a feature vector $\phi(x)$ that captures “major characteristics” of x ,

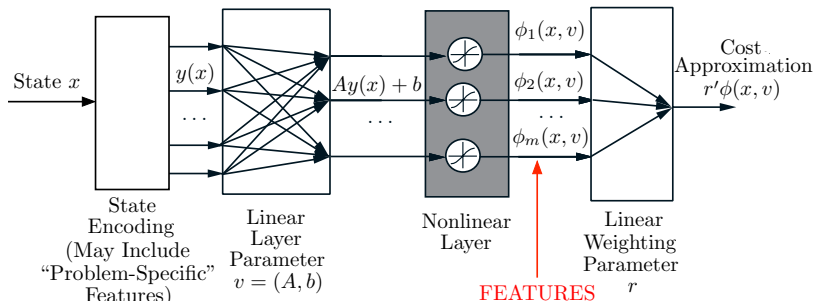
$$\tilde{J}(x, r) = \hat{J}(\phi(x), r),$$

where \hat{J} is some function. Intuitive idea: **Features capture dominant nonlinearities**.

- A **linear feature-based architecture**: $\tilde{J}(x, r) = \sum_{\ell=1}^m r_{\ell} \phi_{\ell}(x) = r' \phi(x)$, where r_{ℓ} and $\phi_{\ell}(x)$ are the ℓ th components of r and $\phi(x)$.



Neural Nets: An Architecture that Automatically Constructs Features



Given a set of state-cost training pairs (x^s, β^s) , $s = 1, \dots, q$, the parameters of the neural network (A, b, r) are obtained by solving the training problem

$$\min_{A, b, r} \sum_{s=1}^q \left(\sum_{\ell=1}^m r_{\ell} \sigma((Ay(x^s) + b)_{\ell}) - \beta^s \right)^2$$

- **Incremental (backpropagation) methods** play a critical role.
- **Universal approximation**; with large enough size, we can approximate "anything."
- **Deep neural network advantage**; overparametrization helps.

Finite Horizon Sequential DP Approximation - Parametric Approximation at Every Stage (Also Called **Fitted Value Iteration**)

Train cost approximations $\tilde{J}_N, \tilde{J}_{N-1} \dots, \tilde{J}_0$, sequentially going backwards

- Start with $\tilde{J}_N = g_N$
- Given a cost-to-go approximation \tilde{J}_{k+1} , we **use one-step lookahead to construct a large number of state-cost pairs** (x_k^s, β_k^s) , $s = 1, \dots, q$, where

$$\beta_k^s = \min_{u \in U_k(x_k^s)} E \left\{ g(x_k^s, u, w_k) + \tilde{J}_{k+1}(f_k(x_k^s, u, w_k), r_{k+1}) \right\}, \quad s = 1, \dots, q$$

- We “train” an architecture \tilde{J}_k on the training set (x_k^s, β_k^s) , $s = 1, \dots, q$.
- **Each sample involves minimization of an expected value $E\{\cdot\}$**

Typical approach: We minimize over r_k

$$\sum_{s=1}^q (\tilde{J}_k(x_k^s, r_k) - \beta_k^s)^2 \quad (+ \text{regularization})$$

Important advantage: Can be combined with on-line play/approximation in value space, so **the Newton step interpretation applies**. However, $\min_u E\{\cdot\}$ operation complicates the collection of samples.

- Consider sequential DP approximation of Q -factor parametric approximations

$$\tilde{Q}_k(x_k, u_k, r_k) \approx E \left\{ g_k(x_k, u_k, w_k) + \min_{u \in U_{k+1}(x_{k+1})} \tilde{Q}_{k+1}(x_{k+1}, u, r_{k+1}) \right\}$$

- We obtain $\tilde{Q}_k(x_k, u_k, r_k)$ by training with many pairs $((x_k^s, u_k^s), \beta_k^s)$, where β_k^s is a sample of the approximate Q -factor of (x_k^s, u_k^s) .
- A mathematical trick: The order of $E\{\cdot\}$ and \min have been reversed. Each β_k^s can use a few-samples approximation of the expected value $E\{\cdot\}$.
- Samples β_k^s can be obtained in model-free fashion. Sufficient to have a simulator that generates state-control-cost-next state random samples

$$((x_k, u_k), (g_k(x_k, u_k, w_k), x_{k+1}))$$

- Having computed r_k , the one-step lookahead control can be obtained on-line as

$$\tilde{\mu}_k(x_k) \in \arg \min_{u \in U_k(x_k)} \tilde{Q}_k(x_k, u, r_k)$$

without the need of a model or expected value calculations.

- Important advantage: The on-line calculation of the control is simplified.
- However, the Newton step property is lost. Also on-line replanning is lost.
- To address these issues: Use approximation in value space with

$$\tilde{J}_{k+1}(x_{k+1}) = (\text{ or } \approx) \min_u \tilde{Q}_{k+1}(x_{k+1}, u, r_{k+1})$$

Should we Approximate Q-Factors or Q-Factor Differences?

To compare controls at x , we only need Q-factor differences $\tilde{Q}(x, u) - \tilde{Q}(x, u')$

An example of what can happen if we approximate Q-factors:

- Scalar system and cost per stage:

$$x_{k+1} = x_k + \delta u_k, \quad g(x, u) = \delta(x^2 + u^2), \quad \delta > 0 \text{ is very small;}$$

think of discretization of continuous-time problem involving $dx(t)/dt = u(t)$

- Consider policy $\mu(x) = -2x$. Its cost function can be calculated to be

$$J_\mu(x) = \frac{5x^2}{4}(1 + \delta) + O(\delta^2), \quad \text{HUGE relative to } g(x, u)$$

Its Q-factor can be calculated to be

$$Q_\mu(x, u) = \frac{5x^2}{4} + \delta \left(\frac{9x^2}{4} + u^2 + \frac{5}{2}xu \right) + O(\delta^2)$$

- The important part for policy improvement is $\delta(u^2 + \frac{5}{2}xu)$. When $Q_\mu(x, u)$ is approximated by $\tilde{Q}_\mu(x, u; r)$, it will be dominated by $5x^2/4$ and will be “lost”
- If we approximate Q-factor differences this problem does not arise

A More General Issue: Disproportionate Terms in Q-Factor Calculations

Remedy: Subtract state-dependent constants from Q-factors ("**baselines**")

The constants subtracted should affect the offending terms

Example: Consider (truncated) rollout with policy μ and terminal cost function approximation, so $\tilde{J} \approx J_\mu$

- At x , we minimize over u

$$E\{g(x, u, w) + \tilde{J}(f(x, u, w))\}$$

- Question: **How to deal with $g(x, u, w)$ being tiny relative to $\tilde{J}(f(x, u, w))$?** This happens when we time-discretize continuous-time systems. Another case is when costs are "sparse" (e.g., all cost is incurred upon termination).
- A remedy: **Subtract $\tilde{J}(x)$ from $\tilde{J}(f(x, u, w))$.**

Other possibilities (see Sections 3.3.4, 3.3.5 of class notes)

- Learn directly the cost function differences $D_\mu(x, x') = J_\mu(x) - J_\mu(x')$ with an approximation architecture. This is known as **differential training**.
- Methods known as **advantage updating**. [Work with relative Q-factors, i.e., subtract the state-dependent baseline $\min_{u'} Q(x, u')$ from $Q(x, u)$.]

Exact PI in finite-state transition probability notation

- **Policy evaluation:** We compute the cost function J_μ of current policy μ and its Q -factors,

$$Q_\mu(i, u) = \sum_{j=1}^n p_{ij}(u) (g(i, u, j) + \alpha J_\mu(j)), \quad i = 1, \dots, n, \quad u \in U(i)$$

- **Policy improvement:** We compute the new policy $\bar{\mu}$ according to

$$\bar{\mu}(i) = \arg \min_{u \in U(i)} Q_\mu(i, u), \quad i = 1, \dots, n.$$

Approximate PI

- **Approximate policy evaluation:** Introduce a parametric architecture $\tilde{Q}_\mu(i, u, r)$. We determine r by generating a large number of training triplets (i^s, u^s, β^s) , $s = 1, \dots, q$, and using a least squares fit:

$$\bar{r} = \arg \min_r \sum_{s=1}^q (\tilde{Q}_\mu(i^s, u^s, r) - \beta^s)^2$$

- **Policy improvement:** We compute the new policy $\tilde{\mu}$ according to

$$\tilde{\mu}(i) = \arg \min_{u \in U(i)} \tilde{Q}_\mu(i, u, \bar{r}), \quad i = 1, \dots, n$$

Implementation Issues in Approximate Policy Iteration

BIG challenges to overcome - Rollout is a piece of cake by comparison

Architectural issues:

- To use a **linear feature-based architecture**, we need to have good features
- To use a **neural network**, we need to face harder training issues
- For problems with changing system parameters, we need **on-line replanning**, which may affect the architecture and/or waste the off-line training effort

Inadequate exploration issues:

- To evaluate a policy μ , we must simulate it, so samples of $J_\mu(x)$ are obtained starting from states x frequently visited by μ .
- **This underrepresents states x that are unlikely to occur under μ , and throws off the policy improvement.**
- Imperfect remedies to this include the use of **many short trajectories for generating samples**, and occasionally sample with an **"off-policy"** (a policy other than μ)

Oscillation issues: Policies tend to repeat in cycles

Fascinating phenomena may arise, like **"chattering"** (convergence in the space of parameters, but oscillation in the space of policies) - they do not arise in aggregation.

Aggregation within the Approximation in Value Space Framework

Approximate minimization

$$\min_{u \in U(i)} \sum_{j=1}^n p_{ij}(u) (g(i, u, j) + \alpha \tilde{J}(j))$$

First Step “Future”

Approximations:

Replace $E\{\cdot\}$ with nominal values
(certainty equivalence)

Adaptive simulation

Monte Carlo tree search

Computation of \tilde{J} :

Problem approximation

Rollout

Approximate PI

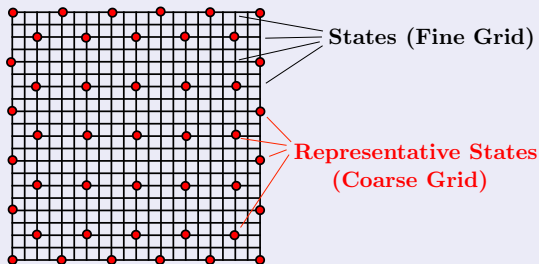
Parametric approximation

Aggregation

- Aggregation is a form of **problem approximation**. We approximate our DP problem with a “smaller/easier” version, which we solve optimally to obtain \tilde{J} .
- **Is related to feature-based parametric approximation** (e.g., when \tilde{J} is piecewise constant, the features are 0-1 set membership functions).
- Several versions: **finite horizon, multistep lookahead, multiagent, etc ...**
- **Can be combined with parametric approximation** (like a neural net) in two ways. Either **use the neural net to provide features**, or **add a local parametric correction** to a \tilde{J} obtained by a neural net (see the class notes).

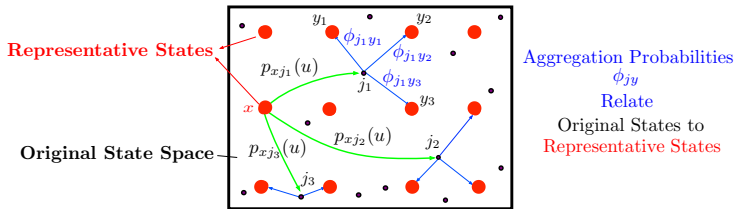
Illustration: A Simple Classical Example of Approximation

Approximate the state space with a coarse grid of states



- Introduce a “small” set of “representative” states to form a **coarse grid**.
- Approximate the original DP problem with a coarse-grid DP problem, called **aggregate problem** (need transition probs. and cost from rep. states to rep. states).
- Solve the aggregate problem by **exact DP**.
- “Extend” the **optimal cost function of the aggregate problem** to the original fine-grid DP problem, i.e., **use some form of interpolation**.
- For example extend the solution by a **nearest neighbor/piecewise constant scheme** (a fine grid state takes the cost value of the “nearest” coarse grid state).

Constructing the Aggregate Problem



- Introduce a finite subset of “representative states” $\mathcal{A} \subset \{1, \dots, n\}$. We denote them by x and y .
- Original system states j are related to rep. states $y \in \mathcal{A}$ with **aggregation probabilities** ϕ_{jy} (“weights” satisfying $\phi_{jy} \geq 0$, $\sum_{y \in \mathcal{A}} \phi_{jy} = 1$).
- Aggregation probabilities express “similarity” or “proximity” of original to rep. states. Can be viewed as **interpolation coefficients**.

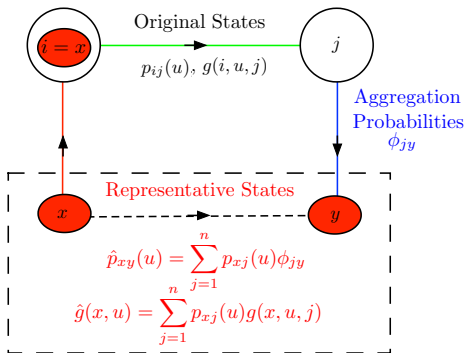
- **Aggregate problem dynamics**: Transition probabilities between rep. states x, y

$$\hat{p}_{xy}(u) = \sum_{i=1}^n p_{xj}(u) \phi_{jy}$$

- **Aggregate problem stage cost** at rep. state x under control u :

$$\hat{g}(x, u) = \sum_{i=1}^n p_{xj}(u) g(x, u, j)$$

The Aggregate Problem - A Reduced State Space DP Problem



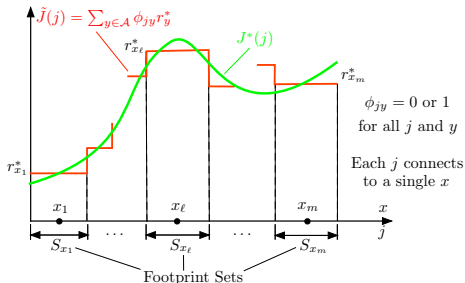
- If r_x^* , $x \in \mathcal{A}$, are the optimal costs of the aggregate problem, approximate the optimal cost function of the original problem by

$$\tilde{J}(j) = \sum_{y \in \mathcal{A}} \phi_{jy} r_y^*, \quad j = 1, \dots, n, \quad (\text{interpolation})$$

- Hard aggregation case:** $\phi_{jy} = 0$ or 1 for all j and y . Then $\tilde{J}(j)$ is **piecewise constant**: It is constant on each set

$$S_y = \{j \mid \phi_{jy} = 1\}, \quad y \in \mathcal{A}, \quad (\text{called the footprint of } y)$$

The Hard Aggregation Case ($\phi_{jy} = 0$ or 1 for all j, y)



The approximate cost fn $\tilde{J} = \sum_{y \in \mathcal{A}} \phi_{jy} r_y^*$ is constant at r_y^* within $S_y = \{j \mid \phi_{jy} = 1\}$.

Approximation error for the piecewise constant case ($\phi_{jy} = 0$ or 1 for all j, y)

Consider the footprint sets

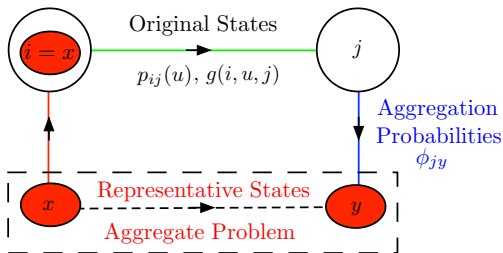
$$S_y = \{j \mid \phi_{jy} = 1\}, \quad y \in \mathcal{A}$$

Then the $(J^* - \tilde{J})$ error is small if J^* varies little within each S_y . In particular,

$$|J^*(j) - \tilde{J}(j)| \leq \frac{\epsilon}{1 - \alpha}, \quad j \in S_y, y \in \mathcal{A},$$

where $\epsilon = \max_{y \in \mathcal{A}} \max_{i, j \in S_y} |J^*(i) - J^*(j)|$ is the max variation of J^* within the S_y .

Solution of the Aggregate Problem



Data of aggregate problem (it is stochastic even if the original is deterministic)

$$\hat{p}_{xy}(u) = \sum_{j=1}^n p_{xj}(u) \phi_{jy}, \quad \hat{g}(x, u) = \sum_{j=1}^n p_{xj}(u) g(x, u, j), \quad \tilde{J}(j) = \sum_{y \in \mathcal{A}} \phi_{jy} r_y^*$$

Exact methods

Once the aggregate model is computed (i.e., its transition probs. and cost per stage), **any exact DP method can be used**: VI, PI, optimistic PI, or linear programming.

Model-free simulation methods

Given a simulator for the original problem, we can obtain a simulator for the aggregate problem. Then **use an (exact) model-free method** to solve the aggregate problem.

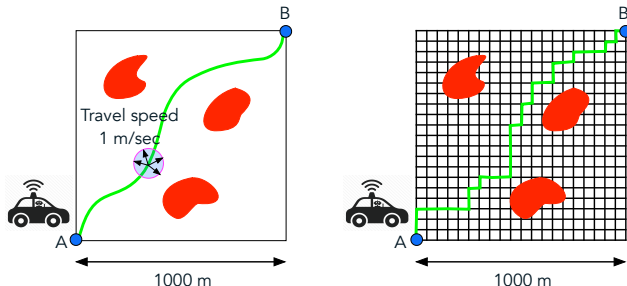
Continuous state space - discounted/bounded cost per stage model

- The rep. states approach **applies with no modification**.
- **The number of rep. states should be finite.**
- A simulation/model-free approach may still be used for the aggregate problem.
- We thus obtain **a general discretization method** for continuous-spaces discounted problems.
- Extension to continuous-state stochastic shortest path problems is more delicate mathematically.

Discounted POMDP with a belief state formulation

- Discounted POMDP models with belief states, fit neatly into the continuous state discounted aggregation framework.
- **The aggregate/rep. states POMDP problem is a finite-state MDP** that can be solved for r^* with any (exact) model-based or model-free method (VI, PI, etc).
- The optimal aggregate cost r^* **yields an approximate cost function**
$$\tilde{J}(j) = \sum_{y \in \mathcal{A}} \phi_{jy} r_y^*$$
- \tilde{J} defines a one-step or multistep lookahead suboptimal control scheme for the original POMDP.

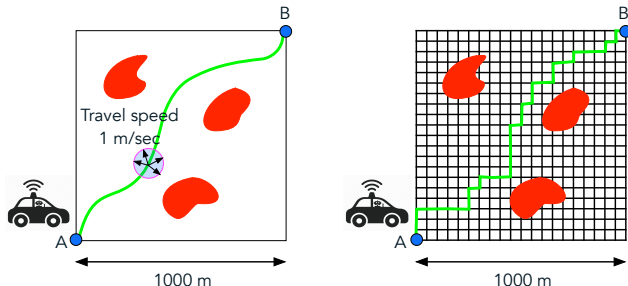
Continuous Control Space Discretization



An example: Discretizing Continuous Motion

- A self-driving car wants to drive from A to B through obstacles. Find the fastest route.
- Car speed is 1 m/sec in any direction.
- We discretize the space with a fine square grid. Suppose we restrict the directions of motion to horizontal and vertical.
- We solve the discretized shortest path problem as an approximation to the continuous shortest path problem.
- A challenge question: Is this a good approximation?

Answer to the Challenge Question



Discretizing Continuous Motion

- The discretization is **FLAWED**.
- **Example:** Assume all motion costs 1 per meter, and no obstacles.
- The continuous optimal solution (the straight A-to-B line) has length $\sqrt{2}$ kilometers.
- The discrete optimal solution has length 2 kilometers **regardless of how fine the discretization is**.
- The difficulty here is that the state space is discretized finely **but the control space is not**.
- This is not an issue in POMDP (the control space is finite).

The main difficulty with rep. states/discretization schemes:

- It may not be easy to find a set of rep. states and corresponding piecewise constant or linear functions that approximate well J^* .
- Too many rep. states may be required for good approximate costs $\tilde{J}(j)$.

Suppose we have a good feature vector $F(i)$: We discretize the feature space

- We introduce representative features that span adequately the feature space
- We aim for an aggregate problem whose states are the rep. features.
- This is a more complicated but also more flexible construction (see the class notes, Section 3.5).

We Reached the End of the Last Lecture for this Course

Some words of caution

- There are challenging implementation issues in all approaches that involve off-line training, and **no fool-proof methods**.
- Problem approximation and hand-crafted feature selection require **domain-specific knowledge**.
- **Training algorithms are not as reliable** as you might think by reading the literature.
- Approximate PI and approximation in policy space **do not deal well with on-line replanning** (approximation in value space does, and involves a **Newton step**).
- **Recognizing success or failure** can be a challenge!
- The RL successes in game contexts are spectacular, but they have benefited from **perfectly known and stable models** and **small number of controls** (per state).
- **Problems with partial state observation** remain a big challenge.

On the positive side

- **Approximation in value space and rollout are relatively simple and reliable.**
- **Massive computational power and distributed computation** are a source of hope.
- Silver lining: **We can begin to address practical problems of unimaginable difficulty!**
- There is **an exciting long journey ahead!**

Some old quotations ...

- **The book of the universe is written in the language of mathematics.** Galileo
- **Learning without thought is labor lost; thought without learning is perilous.**
Confucius (In the language of Confucius' day: learning \approx obtaining knowledge; thought \approx ideas on how to do things)
- **Many arts have been discovered through practice, empirically; for experience makes our life proceed deliberately, but inexperience unpredictably.** Plato
- **White cat or black cat it is a good cat if it catches mice.** Deng Xiaoping

... and some more recent ones

- **Machine learning is the new electricity.** Andrew Ng
(Electricity changed how the world operated. It upended transportation, manufacturing, agriculture and health care. AI is poised to have a similar impact.)
- **Machine learning is the new alchemy.** Ali Rahimi and Ben Recht
(We do not know why some algorithms work and others don't, nor do we have rigorous criteria for choosing one architecture over another ...)

Thank you for your attendance!

Good luck with your term papers and projects!