Midterm Exam #2 Review

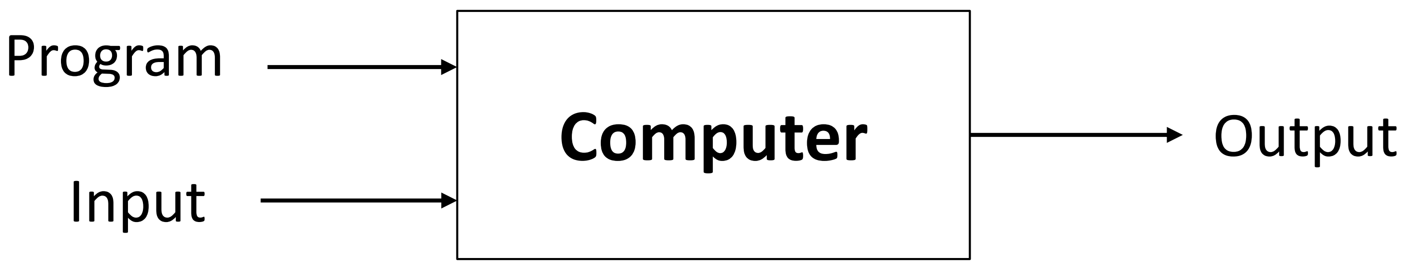
Janardhan Rao (Jana) Doppa

School of EECS, Washington State University

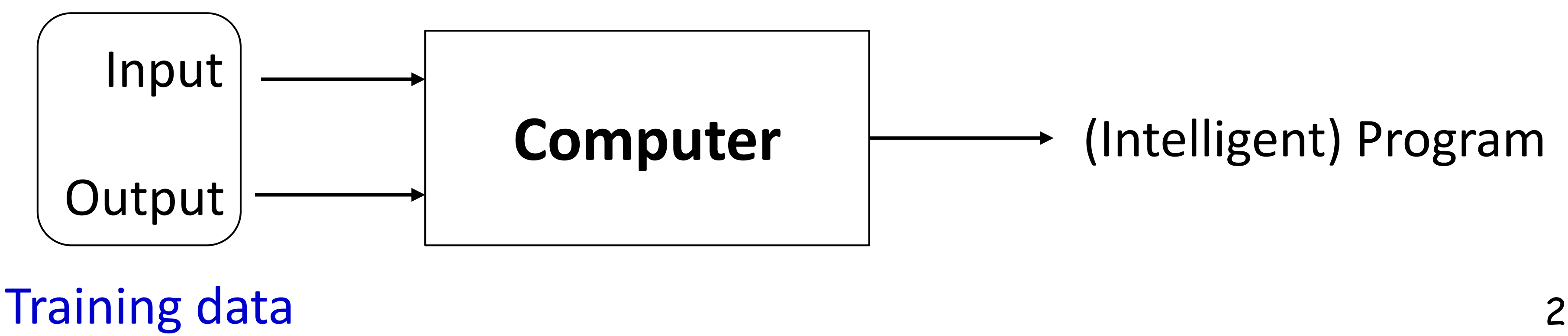
What is Machine Learning?

- **Machine learning = Automating Automation**

Traditional Programming



Machine Learning



Learning Paradigms

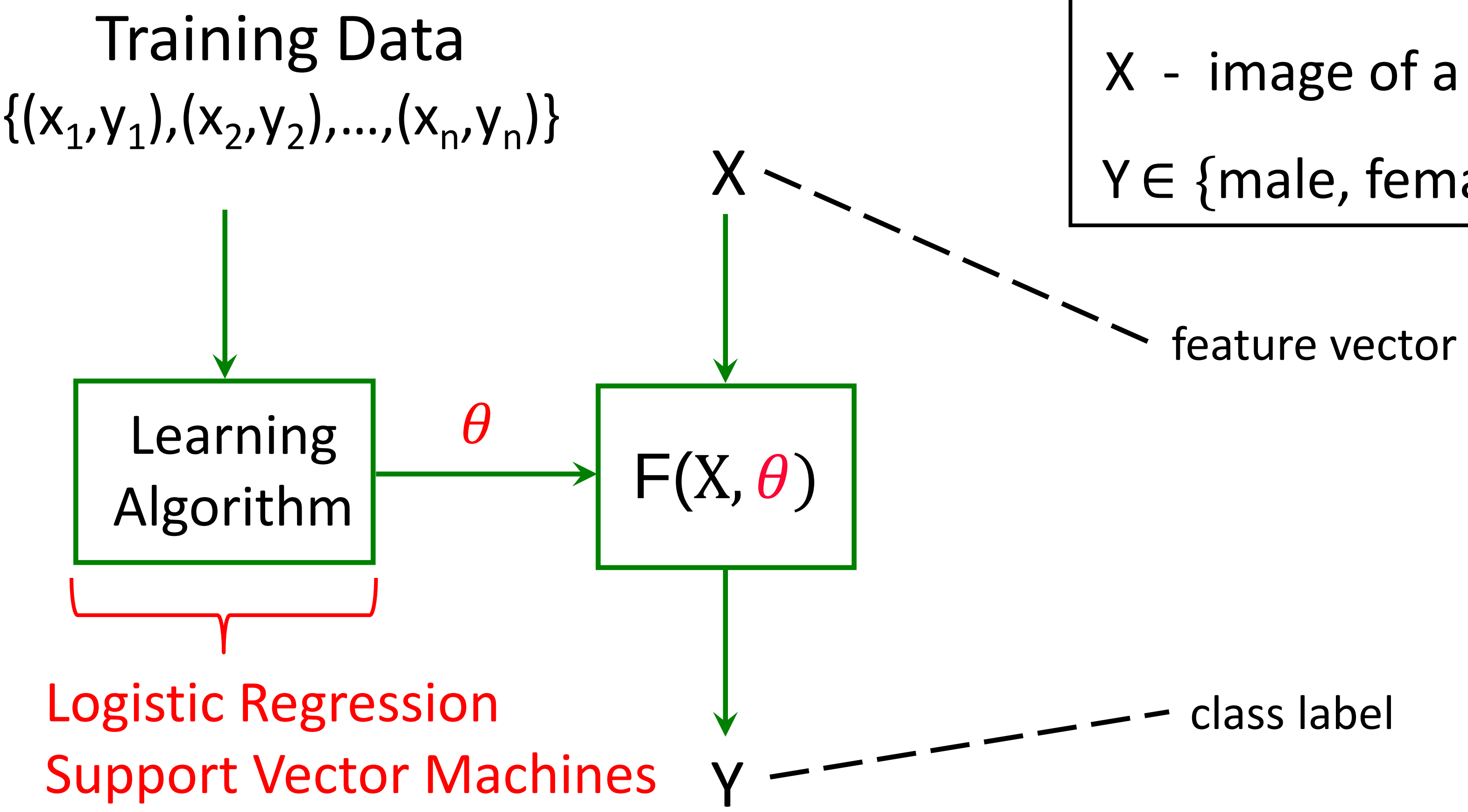
- **Supervised Learning –** Good coverage
- **Semi-Supervised Learning –** No coverage
- **Unsupervised Learning –** Limited coverage
- **Active Learning –** High level overview
- **Reinforcement Learning –** Good coverage

Supervised Learning for Simple Outputs

Example problem:

X - image of a face

Y ∈ {male, female}



Logistic Regression
Support Vector Machines
K Nearest Neighbor
Decision Trees
Neural Networks

Overview of ML Algorithms

- There are lot of machine learning algorithms

- Every machine learning algorithm has three components
  - **Representation**
  - **Evaluation**
  - **Optimization**

Online Learning

- **Online learning**
  - Iterative game between teacher and learner
- **Design principles of online learning**
  - Trade-off amount of change (conservative) and reduction in loss (corrective)
- **Online learning algorithms**
  - Perceptron (fixed learning rate for all examples)
  - Passive-Aggressive (fixed learning rate for each example)
  - Confidence-weighted classifier (fixed learning for each feature and each example)

Support Vector Machine (SVM) Classifier

- **Maximum margin classification:** Linear hyperplane that maximizes margin
- **Support vectors:** a small number of training examples are important to construct the classifier
- **Non-separable data**
  - **Soft-margin formulation**: relax the constraints
  - **Kernel-trick**: implicit mapping of data into high-dimensional space without any additional computational burden
- **Kernelizing online learning algorithms**
  - **Primal:** update weights directly
  - **Dual:** update the coefficients of the training examples

Non-Parametric Classifiers

- **K-nearest neighbor classification**
  - Flexible hypothesis: infinitely complex
  - Classification time is high (NN computation)
  - Doesn’t work very well in high dimensions
- **Decision tree classification**
  - Hierarchical partitioning of the input space into axis parallel rectangles
  - ID3 decision tree construction: greedy procedure based on information gain heuristic
  - Prone to over-fitting, but can be mitigated by pruning based on the validation data

Probabilistic Classifiers

- **Logistic Regression classifier**
  - Represent probability distribution as a parametric sigmoid function
  - Learn parameters via maximum likelihood estimation
  - Closely related to the Perceptron classifier
- **Naïve Bayes classifier**
  - Learns P(Y) and P(X|Y)
  - **Assumption:** each feature is independent from one another given the class label
  - Drastically reduces the number of parameters: improves computational and sample efficiency
  - Lot of success in real-world applications (e.g., text classification)

Ensemble Methods

- **Meta Learning:** combine multiple classifiers into a single one to improve the performance
- **Bagging** (Bootstrap AGGregatING)
  - Learn multiple classifiers by subsampling the training data
  - Classify new examples via majority vote
- **Boosting**
  - Combining multiple simple rules to come up with a highly accurate rule (or classifier)
  - Iteratively modify the weights of the training examples based on their hardness (i.e., misclassified by previous rules of thumb)
  - Classify new examples via weighted majority of the rules
  - **AdaBoost:** a concrete algorithm for boosting

Bias and Variance Decomposition

- **Under-fitting:** Models with too few parameters can perform poorly
- **Over-fitting:** Models with too many parameters can perform poorly
- Generalization error can be decomposed into Bias and Variance
  - Simple (inflexible) models will have high bias and Complex (flexible) models will have high variance
  - **Bias:** measures the accuracy or quality of the algorithm
  - **Variance:** measures the precision or specificity of match
  - **Tradeoff:** low bias => high variance and vice versa
- Bagging reduces variance and Boosting reduces bias

Applying ML in Practice

- **Over-fitting**
  - Training accuracy is much higher than the testing accuracy
  - Model explains the training set very well, but poor generalization
  - More training data and less features will help
- **Under-fitting**
  - Both training and testing accuracies are very low
  - Model cannot represent the target concept well enough
  - More features and using an expressive model will help
- Error analysis and ablation analysis will help debug and improve end-to-end systems
- Ensembles (e.g., random forests) works really well
- Learning == Generalization

Unsupervised Learning

- **Setting:** we are only provided with examples without specifying the labels and we want to cluster the data into groups
- Define a **distance measure** between input examples
  - Symmetric, positivity and self-similarity, satisfy triangle inequality
- **Hierarchical clustering algorithms**
  - Bottom up (agglomerative) and top down (divisive)
- **Partition clustering algorithms**
  - K-Means and Mixture of Gaussians (model based clustering)
  - Expectation Maximization (EM) algorithm for learning from hidden (missing) data
  - K-Means == Hard EM; and GMMs == Soft EM

Reinforcement Learning (1)

- **Problem:** Learning to Act (take decisions) by interacting with a system (world) to maximize the cumulative reward
- World is modeled as a **Markov Decision Process (MDP)**
  - Finite states, finite actions, stochastic transition function, and bounded real-valued reward function
- **Assumptions**
  - First-order Markovian dynamics
  - State-dependent reward
  - Stationary dynamics
  - Full observability
- Solution: policies ("plans" for MDPs)

Reinforcement Learning (2)

- **Non-stationary policy**
  - π:S x T → A; π(s,t) tells us what action to take at state s when there are t stages-to-go
  - Need when we are given a finite planning horizon H
- **Stationary policy**
  - π:S → A; π(s) is action to do at state s (regardless of time)
  - Need when we want to continue taking actions indefinitely
- **Value of a policy** π at state **s**
  - Depends on immediate reward, but also what you achieve subsequently by following that policy

$$V_{\pi}^{k}(s) = E\,[\sum_{t=0}^{k} R^{t} \mid \pi, s\,]$$

$$= E\,[\sum_{t=0}^{k} R(s^{t}) \mid a^{t} = \pi(s^{t}, k-t), s^{0} = s\,]$$

RL Algorithms: Big Picture

Planning with known model (MDP)

- **Policy evaluation**:
  - Given an MDP and a (non)stationary policy π
  - Compute finite-horizon value function $V_{\pi}^{k}(s)$ for any k
- **Policy optimization**:
  - Given an MDP and a horizon H
  - Compute the optimal finite-horizon policy
  - Equivalent to computing optimal value function (value iteration)

Planning with unknown model (MDP)

- **Policy evaluation**:
  - Given a stationary policy π, compute the value of policy
  - Passive RL: direct estimation, ADP, TD methods
- **Policy optimization:**
  - Compute the optimal policy
  - Active RL –- ADP, TD, and Q learning

Finite-Horizon: Policy Evaluation

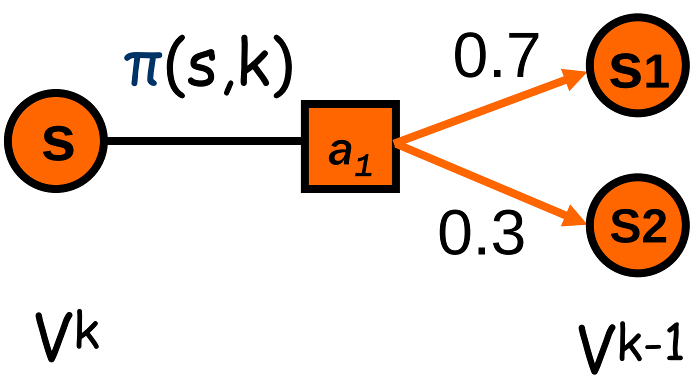
- Can use dynamic programming to compute $V_\pi^k(s)$
  - Markov property is critical for this

(k=0) $$V_\pi^0(s) = R(s), \quad \forall s$$

(k>0) $$V_\pi^k(s) = R(s) + \sum_{s'} T(s, \pi(s,k), s') \cdot V_\pi^{k-1}(s'), \quad \forall s$$

immediate reward

expected future payoff
with *k*-1 stages to go



Finite Horizon: Policy Optimization

- Markov property allows exploitation of DP principle for optimal policy construction
  - no need to enumerate $|A|^{Hn}$ possible policies

- Value Iteration

Bellman backup

$$V^0(s) = R(s), \quad \forall s$$

$$\boxed{V^k(s) = R(s) + \max_a \sum_{s'} T(s,a,s') \cdot V^{k-1}(s')}$$

$$\pi^*(s,k) = \arg\max_a \sum_{s'} T(s,a,s') \cdot V^{k-1}(s')$$

V^k is optimal k-stage-to-go value function

Π*(s,k) is optimal k-stage-to-go policy

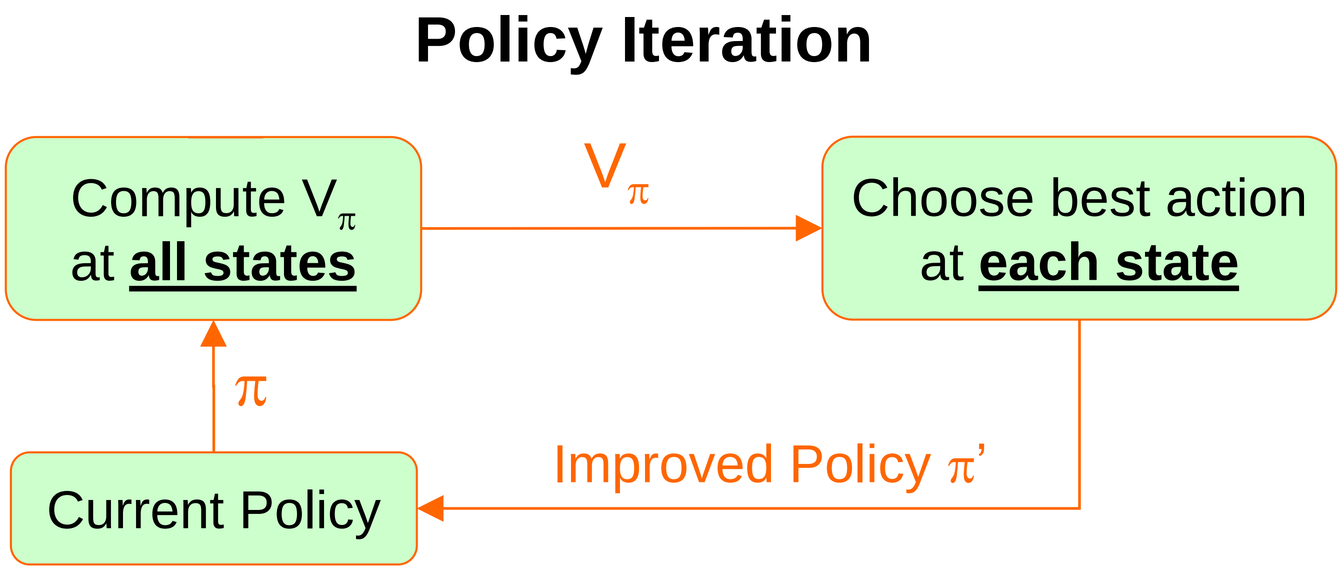
Passive RL: Policy Evaluation w/ unknown MDP

- Monte-Carlo Direct Estimation (model free)
  - Simple to implement
  - Each update is fast
  - Does not exploit Bellman constraints
  - Converges slowly
- Adaptive Dynamic Programming (model based)
  - Harder to implement
  - Each update is a full policy evaluation (expensive)
  - Fully exploits Bellman constraints
  - Fast convergence (in terms of updates)
- Temporal Difference Learning (model free)
  - Update speed and implementation similiar to direct estimation
  - Partially exploits Bellman constraints---adjusts state to ‘agree’ with observed successor
    - Not ***all*** possible successors as in ADP
  - Convergence in between direct estimation and ADP

Active RL: Policy Optimization w/ unknown MDP

- **Exploration vs. Exploitation trade-off**
  - **Exploitation**: To try to get reward. We exploit our current knowledge to get a payoff.
  - **Exploration**: Get more information about the world. How do we know if there is not a pot of gold around the corner.
- **Basic intuition behind most approaches**
  - Explore more when knowledge is weak. Exploit more as we gain knowledge.
- **Exploration policy**
  - We want a policy that is greedy in the limit of infinite exploration (GLIE)
- **ADP-based (model based) RL**
  - Solve for optimal policy given the current model. Take action according to exploration policy. Update model based on new observation. Repeat.
- **TD-based (model based) RL**
  - Start with initial value function. Take action according to exploration policy. Update model based on new observation. Perform TD update to get new value function. Repeat.
- **Q-Learning (model free) RL**
  - Start with initial Q values. Take action according to exploration policy. Perform TD update to get new Q values. Repeat.

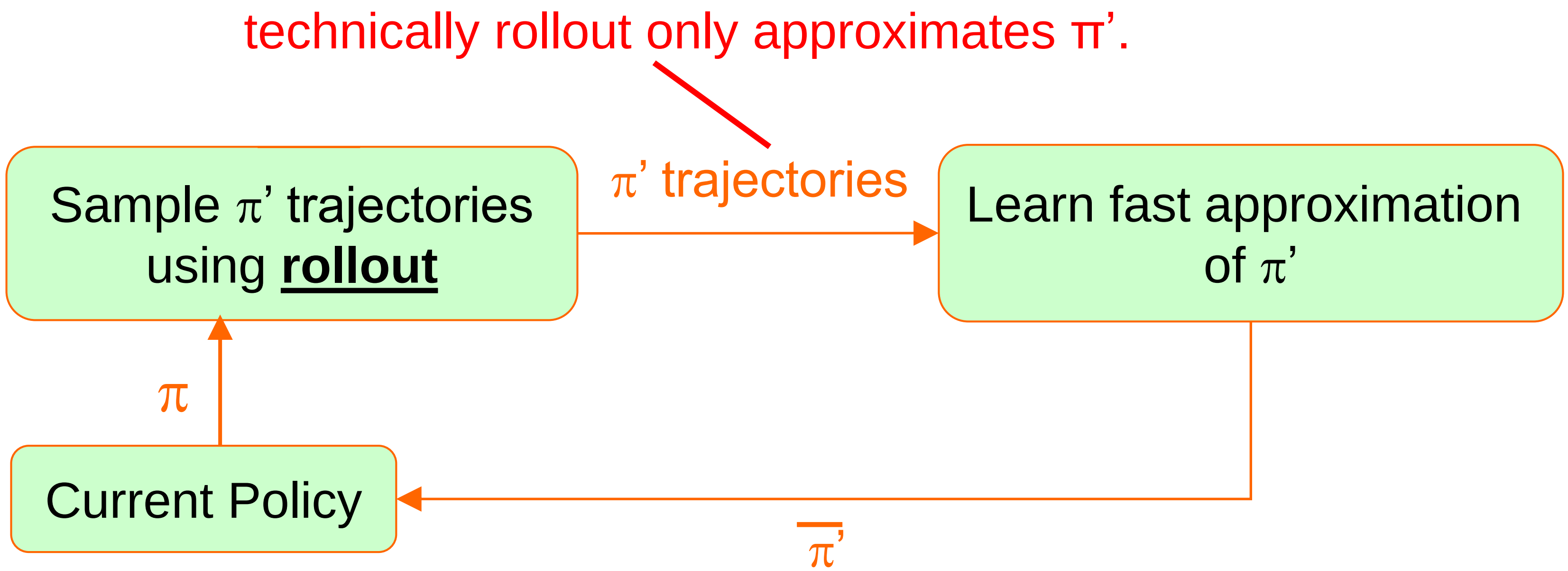
Approximate Policy Iteration for Large MDPs



Approximate policy iteration:

- Only computes values and improved action at some states.
- Uses those to infer a fast, compact policy over all states.

Approximate Policy Iteration



1. Generate trajectories of rollout policy
   (starting state of each trajectory is drawn from initial state distribution I)
2. “Learn a fast approximation” of rollout policy
3. Loop to step 1 using the learned policy as the base policy

Hyper-parameter Search via Bayesian Optimization

- **Build a surrogate statistical model** based on past computational experiments
  - Assumption is that surrogate model is cheap to evaluate
- **Intelligently select the next experiment** (candidate solution) **using the statistical model**
  - Trade-off exploration and exploitation
  - Exploration corresponds to selecting candidates for which the statistical model is not confident (high variance)
  - Exploitation corresponds to selecting candidates for which the statistical model is highly confident (high mean)

Hyper-parameter Search via Bayesian Optimization

- Initialize statistical model F
- Repeat the following steps for several iterations
  - Select the next candidate (say x) by optimizing the acquisition function $A(x)$
  - Run experiment with candidate x to compute its quality y
  - Update the statistical model F based on the new training example (x, y)
  - Update the best uncovered solution so far (say x_{best})

Active Learning

- Active learning is a label-efficient learning strategy
- Intelligently selects the examples based on their informativeness
- **Query Selection Strategies**
  - Uncertainty Sampling
  - Query By Committee (QBC)

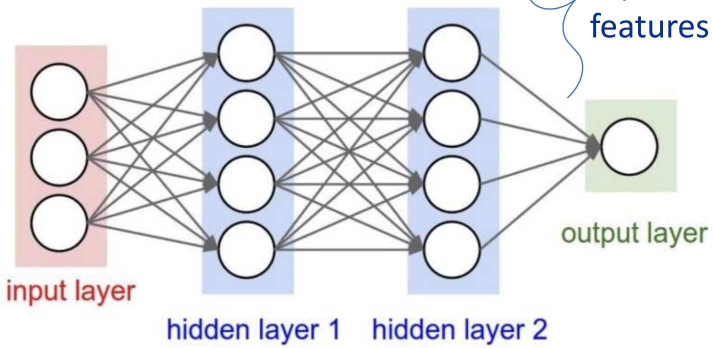
Deep Learning: Differentiable Programming Paradigm

- The user writes a differentiable program
- Use training data to optimize the parameters of this program so that the program behaves as desired

Different ways to think about a neural network

Automatic Feature Learning

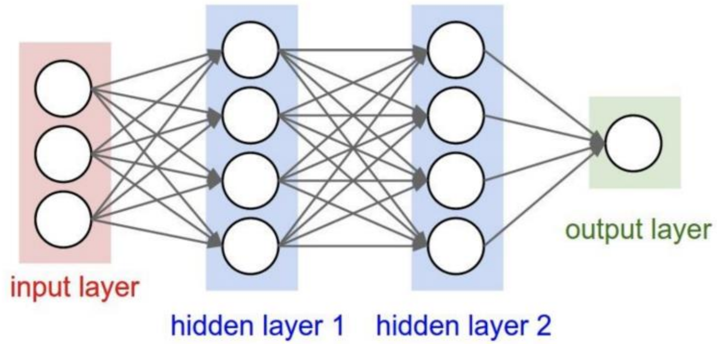
Classifier layer on
top of learned
features



Layers learning
representative
features

Different ways to think about a neural network

Function Approximator



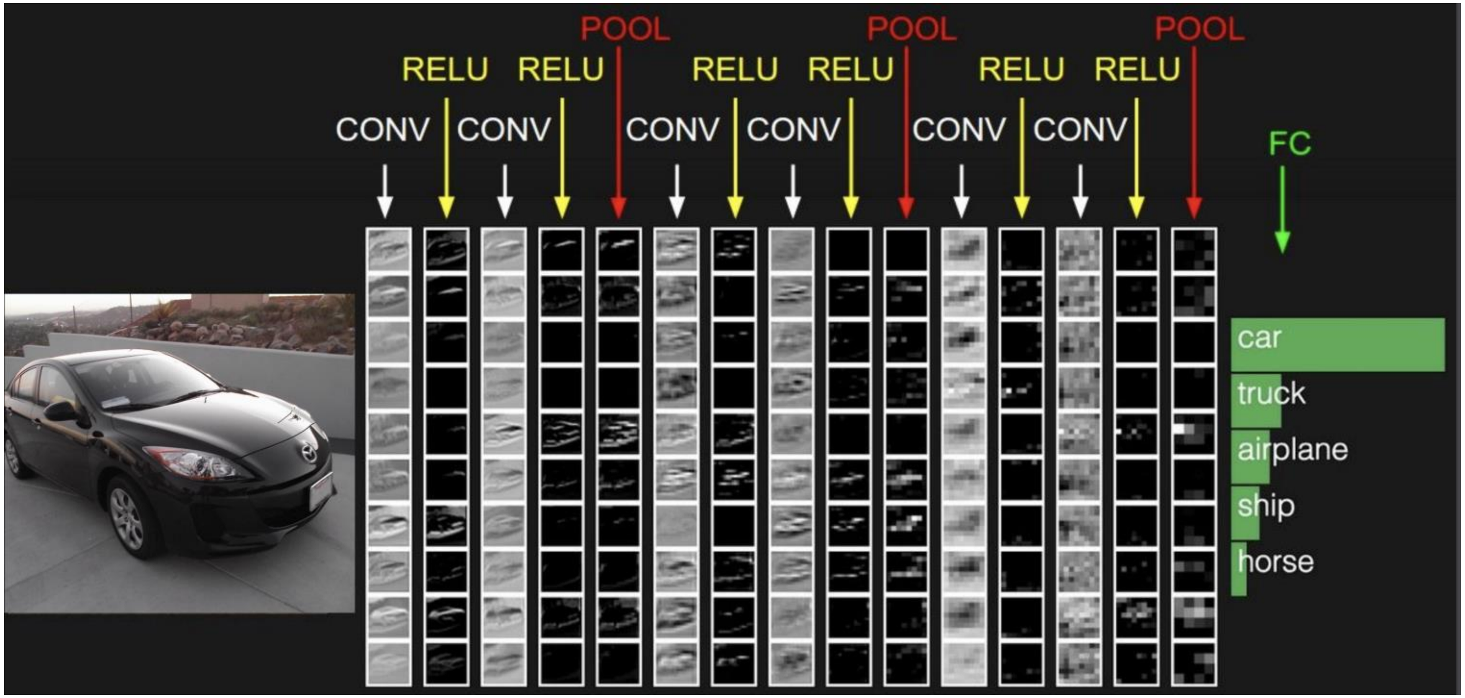
$x \longrightarrow f \longrightarrow y$

$$f\,(x) = \sigma[W_2\,[\sigma(W_1 x + b_1)] + b_2]$$

Inductive Biases and Architectures

- Similarly, Neural networks are a great tool!
- But they are very generic
- We need to add inductive biases in these models
- How do we do that?
- Different types of architectures!
- For e.g. Convolutional architecture for vision
- For e.g. Recurrent architecture for sequences, …

This is what a CNN looks like



We will talk about each layer in detail shortly!

Learning Theory

- **Sample complexity:** How many training examples are needed for a learner to construct (with high probability) a highly accurate concept?
- **Computational complexity:** How much computational resources (time and space) are needed for a learner to construct (with high probability) a highly accurate concept?
- **PAC Model** (Leslie Valiant got a Turing Award!)
  - Only requires a Probably Approximately Correct (PAC) concept: learn a decent approximation most of the time
  - Requires polynomial sample complexity and computational complexity

PAC Learning

- The only reasonable expectation of a learner is that with *high probability* it learns a *close approximation* to the target concept

- In the PAC model, we specify two parameters, ϵ and δ, and require that with probability at least $(1-\delta)$ a system learn a concept with error at most ϵ

- How to prove PAC learnability?
  - First, prove sample complexity of learning a target concept *h** using a hypothesis space *H* is polynomial
  - Second, prove that the learner can train on a polynomial-sized data set in polynomial time