[San Jose State University Special AI Lecture Series VII - ML Fundamentals/Vibe Coding]
ML Theory Meets Practice - Frameworks, Backpropagation & Live Coding

Sunghee Yun

Co-Founder & CTO @ Erudio Bio, Inc.
Co-Founder & CEO @ Erudio Bio Korea, Inc.
Leader of Silicon Valley Privacy-Preserving AI Forum (K-PAI)
CGO / Global Managing Partner @ LULUMEDIC
Global Leadership Initiative Fellow @ Salzburg Global Seminar
Visiting Professor & Advisory Professor @ Sogang Univ. & DGIST

About Speaker

- *Co-Founder & CTO @ Erudio Bio, Inc., San Jose & Novato, CA, USA* 2023 ∼
- *Co-Founder & CEO @ Erudio Bio Korea, Inc., Korea* 2025 ∼
- *Leader of Silicon Valley Privacy-Preserving AI Forum (K-PAI), CA, USA* 2024 ∼
- *CGO / Global Managing Partner @ LULUMEDIC, Seoul, Korea* 2025 ∼
- *KFAS-Salzburg Global Leadership Fellow @ Salzburg Global Seminar, Austria* 2024 ∼
- *Adjunct Professor, EE Department @ Sogang University, Seoul, Korea* 2020 ∼
- *Advisory Professor, EECS Department @ DGIST, Korea* 2020 ∼
- *AI-Korean Medicine Integration Initiative Task Force Member @ The Association of Korean Medicine, Seoul, Korea* 2025 ∼
- *Director of AI Semiconductor @ K-BioX, CA, USA* 2025 ∼
- Global Advisory Board Member @ Innovative Future Brain-Inspired Intelligence System Semiconductor of Sogang University, Korea 2020 ∼
- Technology Consultant @ Gerson Lehrman Gruop (GLG), NY, USA 2022 ∼
- Chief Business Development Officer @ WeStory.ai, Cupertino, CA, USA 2025 ∼
- Advisor @ CryptoLab, Inc., Seoul, Korea 2025 ∼

- Co-Founder & CTO / Head of Global R&D / Chief Applied Scientist / Senior Fellow @ Gauss Labs, Inc., Palo Alto, CA, USA 2020 $\sim$ 2023
- Senior Applied Scientist @ Amazon.com, Inc., Vancouver, BC, Canada 2017 $\sim$ 2020
- Principal Engineer @ Software R&D Center, Samsung Electronics 2016 $\sim$ 2017
- Principal Engineer @ Strategic Marketing & Sales, Memory Business 2015 $\sim$ 2016
- Principal Engineer @ DT Team, DRAM Development, Samsung 2012 $\sim$ 2015
- Senior Engineer @ CAE Team, Memory Business, Samsung, Korea 2005 $\sim$ 2012
- PhD - Electrical Engineering @ Stanford University, CA, USA 2001 $\sim$ 2004
- Development Engineer @ Voyan, Santa Clara, CA, USA 2000 $\sim$ 2001
- MS - Electrical Engineering @ Stanford University, CA, USA 1998 $\sim$ 1999
- BS - Electrical & Computer Engineering @ Seoul National University 1994 $\sim$ 1998

Highlight of Career Journey

- BS in Electrical Engineering (EE) @ Seoul National University
- MS & PhD in Electronics Engineering (EE) @ Stanford University
  - *Convex Optimization - Theory, Algorithms & Software*
  - advisor - *Prof. Stephen P. Boyd*
- Principal Engineer @ Samsung Semiconductor, Inc.
  - *AI & Convex Optimization*
  - collaboration with *DRAM/NAND Design/Manufacturing/Test Teams*
- Senior Applied Scientist @ Amazon.com, Inc.
  - *e-Commerce AIs* - anomaly detection, deep RL, and recommender system
  - *Jeff Bezos's project - drove $200M* in sales via Amazon Mobile Shopping App
- *Co-Founder & CTO / Global R&D Head & Chief Applied Scientist* @ Gauss Labs, Inc.
- *Co-Founder & CTO* @ Erudio Bio, Inc.
- *Co-Founder & CEO* @ Erudio Bio Korea, Inc.

Unpacking AI

- ML Basics - 5
  - Estimation, regression, and inference
  - Machine learning (ML) - basics & formulations
  - Deep learning (DL) - training, stochastic gradient descent (SGD), backpropagation
  - AI in practice
- Reinforcement Learning - 47
  - Markov decision process (MDP), Bellman equations, Bellman optimality equations
  - Dynamic programming (DP), Monte Carlo methods, temporal-difference learning
  - Modern reinforcement learning
  - Alpha Go & modern RL applications
  - LLM & RL, RL evolution
- Vibe coding demo!
- Selected references - 109
- References - 111

ML Basics

Estimation, Regression, and Inference

The optimal estimator

- estimation problem
  - for two random variables $X \in \mathbf{R}^n$ & $Y \in \mathbf{R}^m$
  - design *estimator or predictor* $g : \mathbf{R}^n \to \mathbf{R}^m$ to make $g(X)$ *as close as possible* to Y
- when *closeness* measured by mean-square-error (MSE), *the optimal solution* exists

$$g^*(x) = \mathbf{E}(Y|X = x)$$

Proof of optimality

$$
\begin{array}{rcl}
\mathop{\mathbf{E}}_{X,Y}(g(X) - g^*(X))^T(g^*(X) - Y) & = & \mathop{\mathbf{E}}_{X} \mathop{\mathbf{E}}_{Y}((g(X) - g^*(X))^T(g^*(X) - Y)|X) \\
& = & \mathop{\mathbf{E}}_{X}((g(X) - g^*(X))^T \mathop{\mathbf{E}}_{Y}(g^*(X) - Y)|X) \\
& = & 0
\end{array}
$$

hence

$$
\begin{array}{rl}
\multicolumn{2}{l}{\mathbf{E}\,\|g(X) - Y\|_2^2 = \mathbf{E}\,\|g(X) - g^*(X) + g^*(X) - Y\|_2^2} \\
= & \mathbf{E}\,\|g(X) - g^*(X)\|_2^2 + \mathbf{E}\,\|g^*(X) - Y\|_2^2 + 2\,\mathbf{E}(g(X) - g^*(X))^T(g^*(X) - Y) \\
= & \mathbf{E}\,\|g(X) - g^*(X)\|_2^2 + \mathbf{E}\,\|g^*(X) - Y\|_2^2 \\
\geq & \mathbf{E}\,\|g^*(X) - Y\|_2^2
\end{array}
$$

Regression

- in most cases, *not* possible to obtain g^* (unless, $e.g.$, full knowledge of join PDF)

- regression problem

  – given data set $D = \{(x_1, y_1), \ldots, (x_N, y_N)\} \subset \mathbf{R}^n \times \mathbf{R}^m$

  – find $g : \mathbf{R}^n \to \mathbf{R}^m$ to make $g(X)$ *as close as possible* to Y

- given certain regression method, regressor depends on dataset D

$$g(\cdot; D)$$

Bias & variance

assuming $\mathcal{D}$ is random variable for dataset D

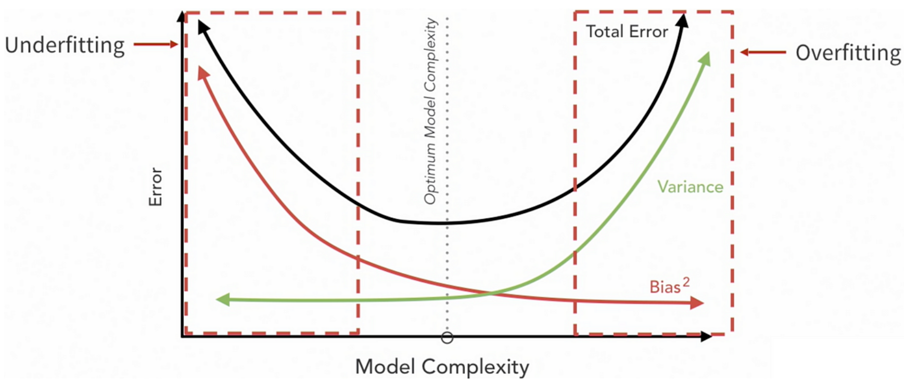
- estimation MSE is

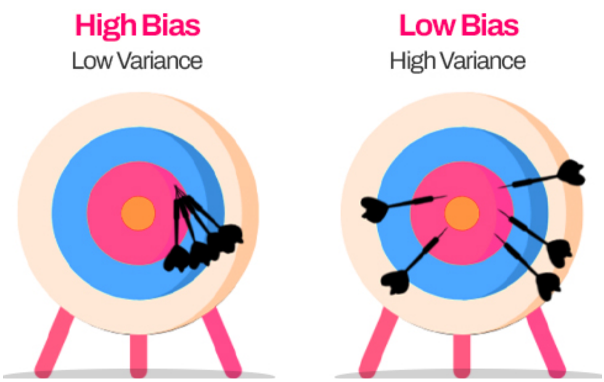
$$
\begin{aligned}
&\mathop{\mathbf{E}}_{X,Y,\mathcal{D}} \|g(X;\mathcal{D}) - Y\|_2^2 \\
&= \underbrace{\mathop{\mathbf{E}}_{X,\mathcal{D}} \|g(X;\mathcal{D}) - \mathop{\mathbf{E}}_{\mathcal{D}} g(X;\mathcal{D})\|_2^2}_{\text{variance}} + \underbrace{\mathop{\mathbf{E}}_{X} \|\mathop{\mathbf{E}}_{\mathcal{D}} g(X;\mathcal{D}) - g^*(X)\|_2^2}_{\text{bias}} + \underbrace{\mathop{\mathbf{E}}_{X,Y} \|g^*(X) - Y\|_2^2}_{\text{noise}} \\
&= \underbrace{\mathop{\mathbf{E}}_{X,\mathcal{D}} \|g(X;\mathcal{D}) - \mathop{\mathbf{E}}_{\mathcal{D}} g(X;\mathcal{D})\|_2^2}_{\text{variance}} + \underbrace{\mathop{\mathbf{E}}_{X,Y} \|\mathop{\mathbf{E}}_{\mathcal{D}} g(X;\mathcal{D}) - Y\|_2^2}_{\text{bias + noise}}
\end{aligned}
$$

- bias & variance
  - *bias* measures how good model is in average
  - *variance* measures how much model varies depending on dataset it is trained on
- *noise* cannot be reduced even with the optimal predictor

Model choice & hyperparameter optimization

- want to choose model or modeling method to make both bias & variance low
  - (too) complex models have low bias, but high variance
  - (too) simple models have low variance, but high bias
- usually solved by *hyperparameter optimization*
  - sometimes called *hyperparameter tuning*





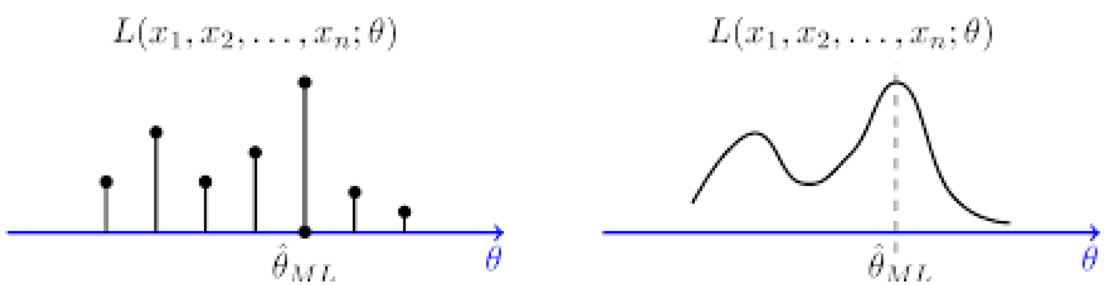
MLE

- maximum likelihood estimation (MLE)
  - assume parameterized distribution of $X \in \mathbf{R}^n$ by $\theta \in \Theta$ - $p(x;\theta)$
  - find θ maximizing *likelihood function*

$$p(x_1,\ldots,x_N;\theta) = \prod_{i=1}^{N} p(x_i;\theta)$$

- MLE solution

$$\hat{\theta}_{\mathrm{MLE}} = \operatorname*{argmax}_{\theta\in\Theta} \prod_{i=1}^{N} p(x_i;\theta)$$



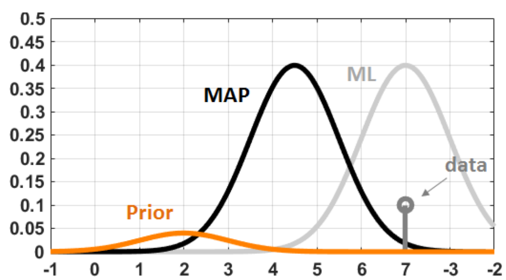
MAP estimation

- maximum a posteriori (MAP) estimation
  - assume *prior knowledge* of θ - $p(\theta)$
  - assume parameterized distribution of $X \in \mathbf{R}^n$ by θ - $p(x|\theta)$
  - find θ maximizing *posteriori probability*

$$p(\theta|x_1, \ldots, x_N)$$

  - Bayes' theorem implies $p(\theta|x_1, \ldots, x_N) \propto p(\theta) \prod_{i=1}^N p(x_i|\theta)$
- MAP solution

$$\hat{\theta}_{\mathrm{MAP}} = \operatorname*{argmax}_{\theta \in \Theta} p(\theta) \prod_{i=1}^N p(x_i|\theta)$$



Bayesian inference

- both MLE & MAP estimation are *point estimations*
- Bayesian inference
  - updates *prior distribution* by replacing it with posterior distribution
- conjugate prior
  - if prior can be further parameterized by hyperparameter α and posterior is in same probability distribution family, both prior and posterior called *conjugate distributions*, prior called *conjugate prior*

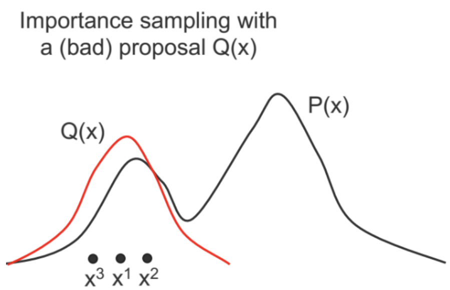
$$p(\theta; \alpha)$$

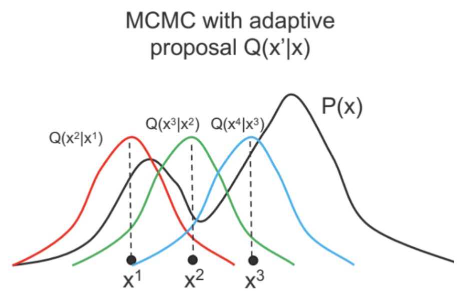
  - in this case, can update hyperparameter α, $i.e.$, find α^+ such that

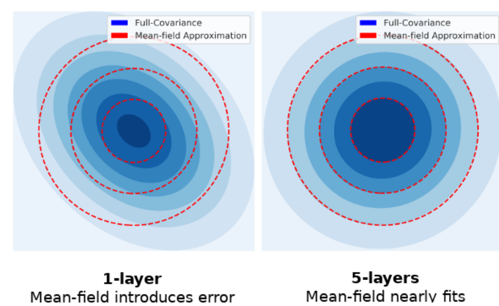
$$p(\theta; \alpha^+) = p(\theta | x_1, \ldots, x_N; \alpha) = \frac{p(\theta; \alpha) \prod_{i=1}^N p(x_i | \theta; \alpha)}{p(x_1, \ldots, x_N; \alpha)}$$

Bayesian algorithms & methods

- exact inference methods
  - conjugate priors - *e.g.*, Beta-Binomial, Normal-Normal, *etc.*
- Markov Chain Monte Carlo (MCMC)
  - Metropolis-Hastings algorithm, Gibbs sampling, Hamiltonian Monte Carlo (HMC)
- variational inference (VI)
  - mean field variational Bayes - assuming parameter independence for tractability
  - structured variational inference - maintaining dependencies & inference tractability
  - variational autoencoder (VAE) - NN-based VI for complex distributions



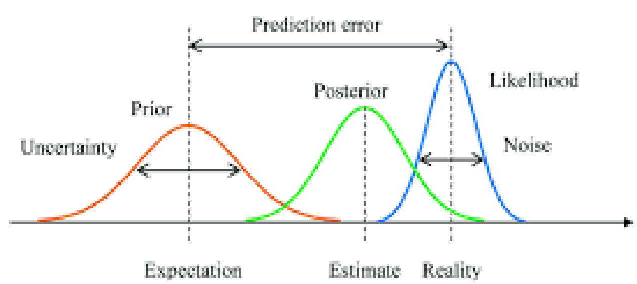




Pros & cons of Bayesian inference

- pros
  - principled uncertainty quantification - providing complete probability distributions
  - incorporates prior knowledge - allowing to formally include domain expertise, *etc.*
  - coherent framework - providing mathematically consistent approach
  - natural sequential learning - easily handles streaming data or online learning scenarios
  - interpretable results - outputs directly interpretable as probabilities
- cons
  - computational complexity - often requiring sophisticated sampling methods
  - prior sensitivity, scalability issues, implementation difficulty, slower inference, model selection challenges

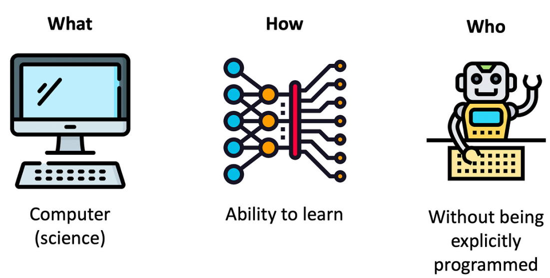
$$\overbrace{p(\theta \mid \mathrm{data})}^{\text{Posterior}} = \frac{\overbrace{p(\mathrm{data} \mid \theta)}^{\text{Likelihood}} \cdot \overbrace{p(\theta)}^{\text{Prior}}}{\underbrace{p(\mathrm{data})}_{\text{Normalization}}}$$

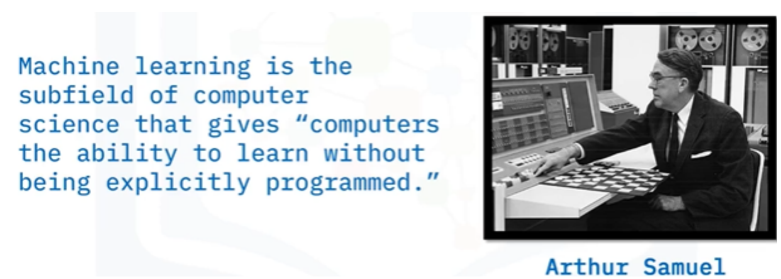


Machine Learning

Machine learning

- ML
  - subfield of computer science that
    "gives computers the ability to learn without being explicitly programmed."
    - Arthur Samuel (1959)
  - *not* magic, still less intelligent than humans for many cases
  - *numerically minimizes* certain (mathematical) loss function to (indirectly) solve *some statistically meaningful* problems





Two famous quotes and one non-famous quote

- Albert Einstein

  The grand aim of all science is to cover the greatest number of empirical facts by logical deduction from the smallest possible number of hypotheses or axioms.

- Alfred North Whitehead

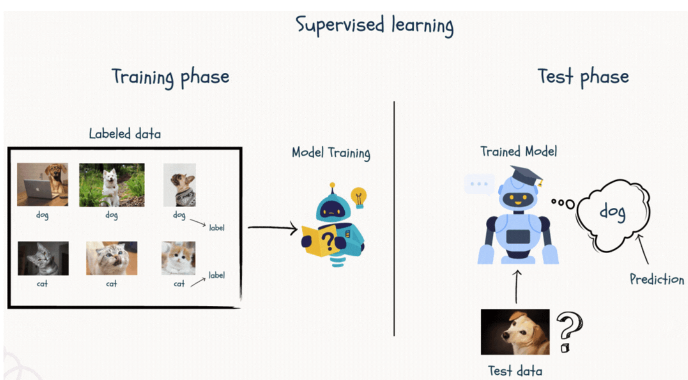
  Civilization advances by extending the number of important operations which we can perform without thinking about them. - Operations of thought are like cavalry charges in a battle – they are strictly limited in number, they require fresh horses, and must only be made at decisive moments.

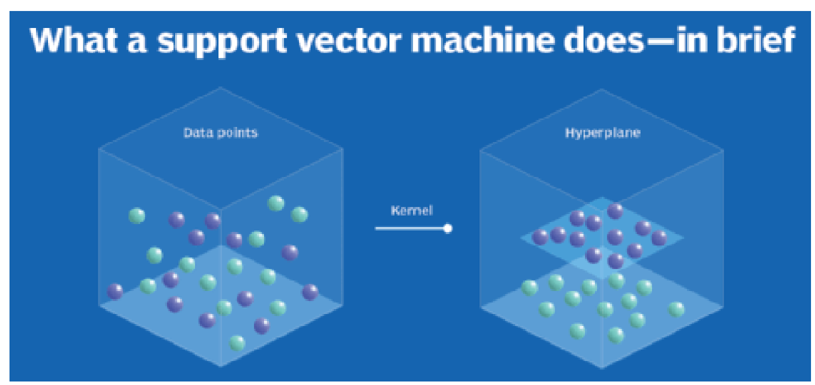
- Demis Hassabis

  . . . biology can be thought of as information processing system, albeit extraordinarily complex and dynamic one . . . just as mathematics turned out to be the right description language for physics, biology may turn out to be the perfect type of regime for the application of AI!

Supervised learning

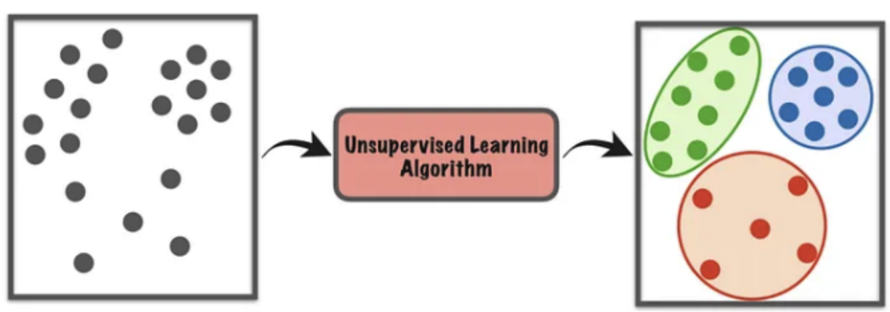
- most basic and widely used type of ML
- model is trained on dataset where correct output or “label” is provided for each input
- use cases
  - image classification, object detection, semantic segmentation
  - natural language processing (NLP) - text classification, sentiment analysis
  - predictive modeling, medical diagnosis
- algorithms
  - linear regression, logistic regression, decision trees, random forest
  - support vector machine (SVM), k-nearest neighbors (kNN)





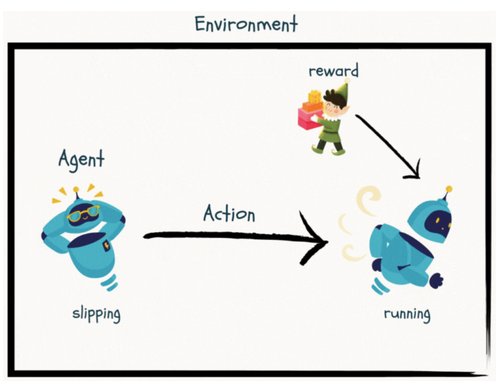
Unsupervised learning

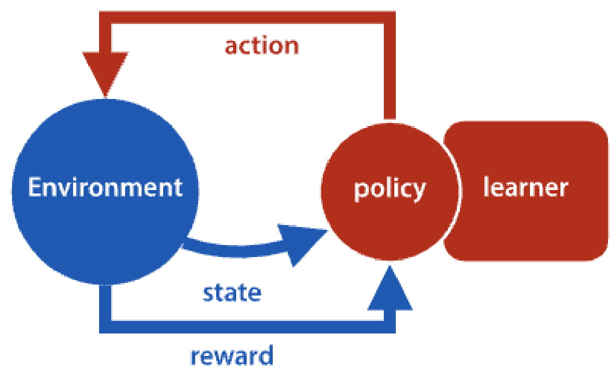
- model is given dataset without any labels or output
- model finds patterns & structure within data on its own
- use cases
  - clustering, dimensionality reduction
  - anomaly detection, generative models
- algorithms
  - k-means clutering, hierarchical clustering, principal component analysis (PCA)
  - t-distributed stochastic neighbor embedding (t-SNE)



Reinforcement learning

- (quite different from supervised & unsupervised learnings)
- model learns from consequences of its actions
  - model receives feedback on its performance; feedback called *reward*
  - uses that information to adjust its actions and improve its performance over time
- use cases
  - robotics, game playing, autonomous vehicles, industrial control
  - healthcare, finance
- algorithms
  - Q-learning, SARSA, DQN, A3C, policy gradient





ML Formulations

Loss minimization

- assume data set $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$ with $x^{(i)} \in \mathbf{R}^n$, $y^{(i)} \in \mathbf{R}^q$
- loss minimization is to solve

$$\text{minimize} \quad \frac{1}{m}\sum_{i=1}^{m} l(y^{(i)}, f(x^{(i)};\theta))$$

  where optimization variable is $\theta \in \mathbf{R}^p$, $f : \mathbf{R}^n \times \mathbf{R}^p \to \mathbf{R}^q$ is model function & $l : \mathbf{R}^q \times \mathbf{R}^q \to \mathbf{R}_+$ is loss function

  – find model with *smallest modeling error*

- loss function examples
  – Eucleadina norm (2-norm) - $\|y - \hat{y}\|_2^2$
  – 1-norm - $\|y - \hat{y}\|_1$
  – soft-max - $y^T \exp(\hat{y})/\mathbf{1}^T \exp(\hat{y})$

Statistical problem formulation

- assume data set $X_m = \{x^{(1)}, \ldots, x^{(m)}\}$
  - drawn independently from (true, but unknown) data generating distribution $p_{\mathrm{data}}(x)$
- maximum likelihood estimation (MLE) is to solve

$$\text{maximize} \quad p_{\mathrm{model}}(X;\theta) = \prod_{i=1}^m p_{\mathrm{model}}(x^{(i)};\theta)$$

  where optimization variable is θ

  - find *most plausible or likely model* that fits data
- equivalent (but more numerically tractable) formulation

$$\text{maximize} \quad \log p_{\mathrm{model}}(X;\theta) = \sum_{i=1}^m \log p_{\mathrm{model}}(x^{(i)};\theta)$$

MLE & KL divergence

- in information theory, Kullback-Leibler (KL) divergence defines distance between two probability distributions p & q

$$D_{\mathrm{KL}}(p\|q) = \mathop{\mathbf{E}}_{X\sim p} \log p(X)/q(X) = \int_{x\in\Omega} p(x)\log\frac{p(x)}{q(x)}\,dx$$

- KL divergence between data distribution p_{data} & model distribution p_{model} can be approximated by Monte Carlo method as

$$D_{\mathrm{KL}}(p_{\mathrm{data}}\|p_{\mathrm{model}}(\theta)) \simeq \frac{1}{m}\sum_{i=1}^{m}(\log p_{\mathrm{data}}(x^{(i)}) - \log p_{\mathrm{model}}(x^{(i)};\theta))$$

  where $x^{(i)}$ are drawn (of course) according to p_{data}

- hence *minimizing KL divergence is equivalent to solving MLE problem!*

Equivalence of MLE to MSE

- assume model is Gaussian, *i.e.*, $y \sim \mathcal{N}(g_\theta(x), \Sigma)$ ($g_\theta(x) \in \mathbf{R}^p$, $\Sigma \in \mathbf{S}^p_{++}$)

$$p(y|x;\theta) = \frac{1}{\sqrt{2\pi}^p |\Sigma|^{1/2}} \exp\left(-\frac{1}{2}\left(y - g_\theta(x)\right)^T \Sigma^{-1} \left(y - g_\theta(x)\right)\right)$$

- assuming that $\Sigma = \alpha I_p$, log-likelihood becomes

$$\begin{aligned}\sum_{i=1}^m \log p(x^{(i)}, y^{(i)};\theta) &= \sum_{i=1}^m \log p(y^{(i)}|x^{(i)};\theta) p(x^{(i)}) \\ &= -\sum_{i=1}^m \|y^{(i)} - g_\theta(x^{(i)})\|_2^2/2\alpha - \frac{pm}{2}\log(2\pi\alpha) + \sum_{i=1}^m \log p(x^{(i)})\end{aligned}$$

- hence *minimizing mean-square-error (MSE) is equivalent to solving MLE problem!*

Numerical optimization problem formulation

- (true) problem to solve

$$\text{minimize} \quad \mathbf{E}\, l(g_\theta(X), Y)$$

  - *impossible* to solve

- loss minimize formulation - surrogate problem to solve

$$\text{minimize} \quad f(\theta) = \frac{1}{m} \sum_{i=1}^m l(g_\theta(x^{(i)}), y^{(i)})$$

- formulation with regularization

$$\text{minimize} \quad f(\theta) = \frac{1}{m} \sum_{i=1}^m l(g_\theta(x^{(i)}), y^{(i)}) + \gamma r(\theta)$$

- stochastic gradient descent (SGD)

$$\theta^{k+1} = \theta^k - \alpha_k \nabla f(\theta^k)$$

Deep Learning

Deep learning (DL)

- machine learning using artificial neural networks with multiple layers for
  - automatically learning hierarchical representations of data
- key components
  - deep neural networks, hidden layers, backpropagation, activation functions
  - hierarchical feature learning, representation learning, end-to-end learning
- key breakthroughs enabling DL
  - massively available data, GPU computing, algorithmic advances

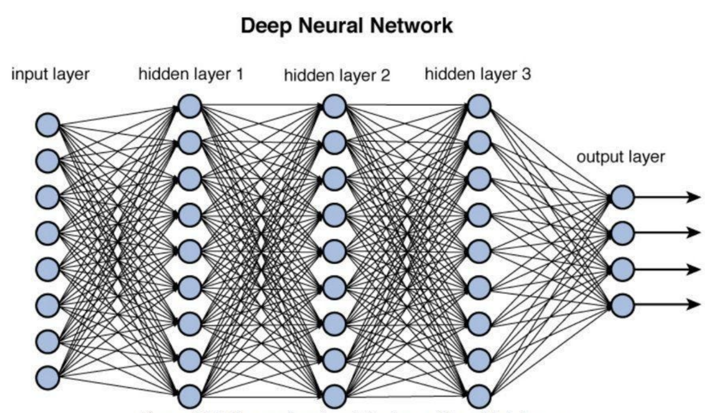
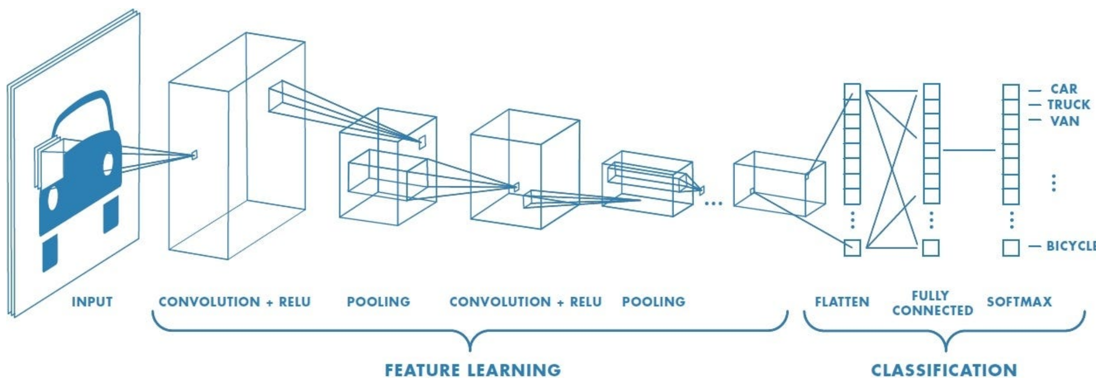


Figure 12.2 Deep network architecture with multiple layers.

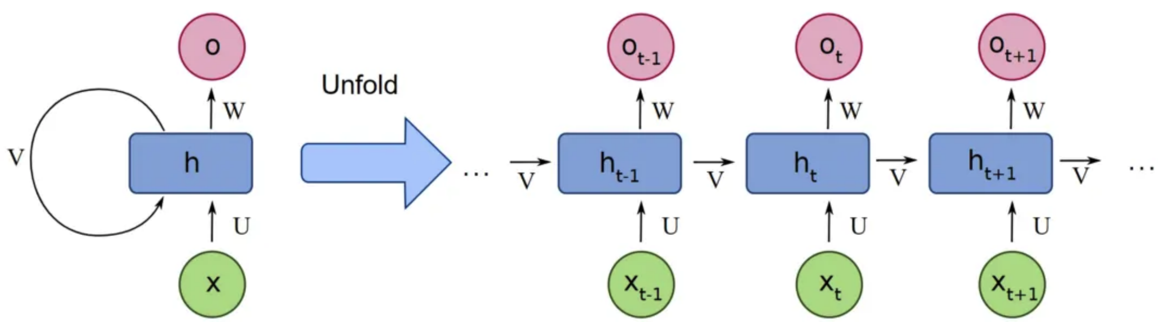
Convolutional neural network (CNN)

- specialized DL learning architecture designed for
  - processing grid-like data such as images
  - where spatial relationships between pixels matter
- key components
  - convolutional layers, pooling layers, activation functions, fully connected layers
- how it works
  - feature extraction, translation invariance, parameter sharing
- why it excels
  - local connectivity, hierarchical learning



Recurrent neural network (RNN)

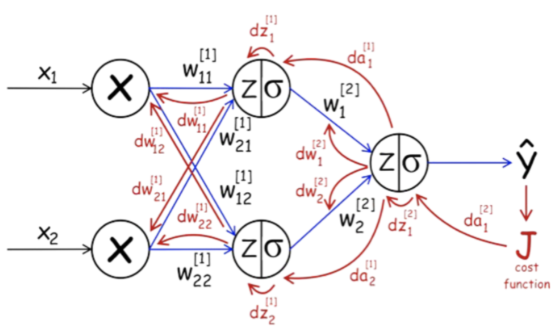
- neural network designed for
  - processing sequential data by maintaining memory of previous inputs
- key components
  - hidden states, recurrent connections, input/output layers, weight sharing
- how it works
  - sequential processing, memory mechanism, temporal dependencies
- why it excels
  - variable length input, context awareness, flexible architecture
- variants - long short-term memory (LSTM), gated recurrent unit (GRU)

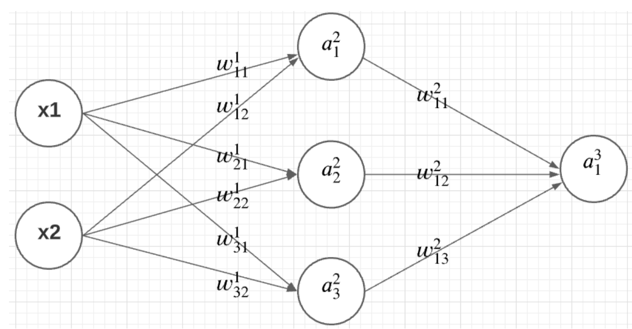


Training DNN using SGD

Notations

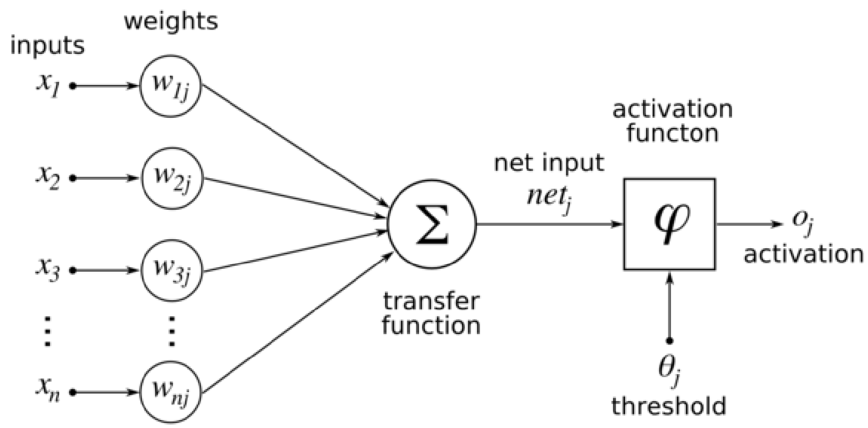
- p / q - dimension of input / output spaces
- $l : \mathbf{R}^q \times \mathbf{R}^q \to \mathbf{R}_+$ - loss function
- d - depth of neural network
- n_i $(1 \leq i \leq d)$ - number of perceptrons in ith layer
- $z^{[i]} \in \mathbf{R}^{n_i}$ - input to ith layer
- $o^{[i]} \in \mathbf{R}^{n_i}$ - output of ith layer
- $W^{[i]} \in \mathbf{R}^{n_i \times n_{i-1}}$ - weights of connections between $(i-1)$th and ith layer
- $w^{[i]} \in \mathbf{R}^{n_i \times n_{i-1}}$ - bias weights of ith layer
- $\phi^{[i]} : \mathbf{R}^{n_i} \to \mathbf{R}^{n_i}$ - activation functions of ith layer



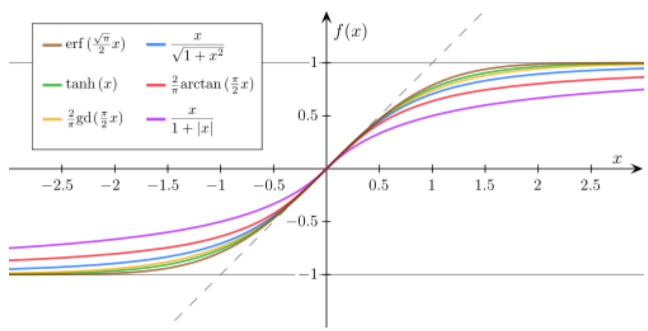


Basic unit & activation function

- basic unit



- activation function



Neural net equations

- modeling function for the (deep) neural network $g_\theta : \mathbf{R}^p \to \mathbf{R}^q$

$$g_\theta = \phi_\theta^{[d]} \circ \psi_\theta^{[d]} \circ \cdots \circ \phi_\theta^{[1]} \circ \psi_\theta^{[1]}$$

  or equivalently

$$g_\theta(x) = \phi_\theta^{[d]}(\psi_\theta^{[d]}(\cdots(\phi_\theta^{[1]}(\psi_\theta^{[1]}(x)))))$$

- for ith layer
  - output via (componentwise) activation function

$$o^{[i]} = \phi^{[i]}(z^{[i]}) \Leftrightarrow o_j^{[i]} = \phi_j^{[i]}(z_j^{[i]}) \quad (1 \leq j \leq n_i)$$

  - input via affine transformation $\psi^{[i]} : \mathbf{R}^{n_{i-1}} \to \mathbf{R}^{n_i}$

$$z^{[i]} = \psi^{[i]}(o^{[i-1]}) = W^{[i]} o^{[i-1]} + w^{[i]}$$

Stochastic gradient descent

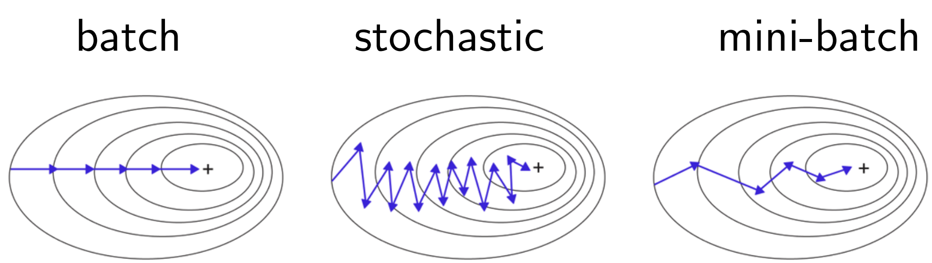
- ML training tries to minimize some loss function - $f(\theta)$ depends on (not only θ, but also) batch of data $(x^{(1)}, y^{(1)}), \ldots (x^{(m)}, y^{(m)})$

$$\text{minimize} \quad f(\theta)$$

- while exist hundreds of optimization methods solving this problem
  - *the only method used widely* is stochastic *gradient descent*!
- (stochastic) gradient descent

$$\theta^{k+1} = \theta^k - \alpha^k \nabla f(\theta^k)$$

- *backpropagation* is used to evaluate this (stochastic) *gradient* using *chain rule*



Chain rule

- suppose
  - two functions $f : \mathbf{R}^n \to \mathbf{R}^m$ & $g : \mathbf{R}^m \to \mathbf{R}$
  - Jacobian of f - $Df : \mathbf{R}^n \to \mathbf{R}^{m \times n}$
  - gradient of g - $\nabla g : \mathbf{R}^m \to \mathbf{R}^m$
- gradient of composite function $h = g \circ f$

$$\nabla h(\theta) = Df(\theta)^T \nabla g(f(\theta)) \in \mathbf{R}^n \quad \text{(using matrix-vector multiplication)}$$

in other words

$$\frac{\partial}{\partial \theta_i} h(\theta) = \sum_{j=1}^{m} \frac{\partial}{\partial \theta_i} f_j(\theta) \nabla_j g(f(\theta)) \quad \text{(scalar version)}$$

Loss function & its gradient

- assume cost function of deep neural network is

$$f(\theta) = \frac{1}{m} \sum_{k=1}^{m} l(g_\theta(x^{(k)}), y^{(k)}) = \frac{1}{m} \sum_{k=1}^{m} f_k(\theta)$$

where

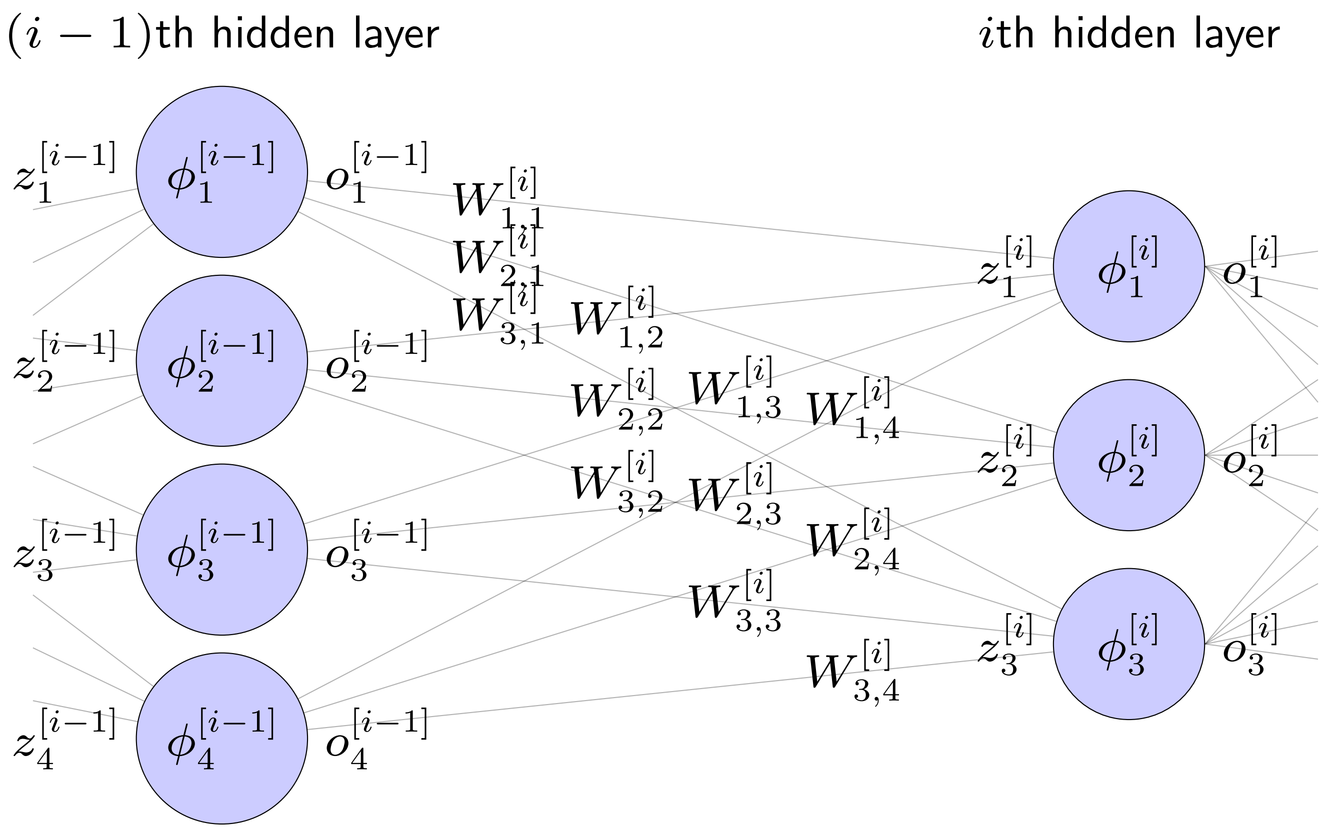
$$f_k(\theta) = l(g_\theta(x^{(k)}), y^{(k)})$$

- gradient is

$$m \nabla_\theta f(\theta) = \sum_{k=1}^{m} \nabla_\theta l(g_\theta(x^{(k)}), y^{(k)}) = \sum_{k=1}^{m} \nabla_\theta f_k(\theta)$$

  – $i.e.$, evaluate gradient $\nabla_\theta f_k(\theta)$ for each data point $(x^{(k)}, y^{(k)})$

Hidden layers



Backpropagation formula using chain rule

- for each data $(x^{(k)}, y^{(k)})$
  - via activation function

$$\frac{\partial}{\partial z_j^{[i]}} f_k(\theta) = \frac{\partial}{\partial o_j^{[i]}} f_k(\theta) {\phi_j^{[i]}}'(o_j^{[i]}) \quad \text{for } 1 \leq j \leq n_i \tag{1}$$

where ${\phi_j^{[i]}}'(o_j^{[i]})$ is derivative of activation function $\phi_j^{[i]}$ evaluated at $o_j^{[i]}$

  - via affine transformation

$$\begin{aligned}
\frac{\partial}{W_{j,l}^{[i]}} f_k(\theta) &= o_l^{[i-1]} \frac{\partial}{\partial z_j^{[i]}} f_k(\theta) \quad \text{for } 1 \leq j \leq n_i \;\&\; 1 \leq l \leq n_{i-1} && (2) \\
\frac{\partial}{\partial w_j^{[i]}} f_k(\theta) &= \frac{\partial}{\partial z_j^{[i]}} f_k(\theta) \quad \text{for } 1 \leq j \leq n_i && (3) \\
\frac{\partial}{\partial o_l^{[i-1]}} f_k(\theta) &= \sum_{j=1}^{n_i} W_{j,l}^{[i]} \frac{\partial}{\partial z_j^{[i]}} f_k(\theta) \quad \text{for } 1 \leq l \leq n_{i-1} && (4)
\end{aligned}$$

Backpropagation formula using matrix-vector multiplication

- for each data $(x^{(k)}, y^{(k)})$
  - via activation function

$$\nabla_{z^{[i]}} f_k(\theta) = D\phi^{[i]} \nabla_{o^{[i]}} f_k(\theta) \tag{5}$$

where $D\phi^{[i]} = \mathbf{diag}({\phi_1^{[i]}}'(o_1^{[i]}), \ldots, {\phi_{n_i}^{[i]}}'(o_{n_i}^{[i]}))$ is Jacobian of $\phi^{[i]}$ evaluated at $o^{[i]}$

  - via affine transformation

$$\begin{aligned} \nabla_{W^{[i]}} f_k(\theta) &= \nabla_{z^{[i]}} f_k(\theta) {o^{[i-1]}}^T \in \mathbf{R}^{n_i \times n_{i-1}} && (6) \\ \nabla_{w^{[i]}} f_k(\theta) &= \nabla_{z^{[i]}} f_k(\theta) \in \mathbf{R}^{n_i} && (7) \\ \nabla_{o^{[i-1]}} f_k(\theta) &= {W^{[i]}}^T \nabla_{z^{[i]}} f_k(\theta) \in \mathbf{R}^{n_{i-1}} && (8) \end{aligned}$$

Backpropagation formula using `Python` `numpy` package

- for each data $(x^{(k)}, y^{(k)})$
  - via activation function

$$\texttt{grad\_z = phi\_dir * grad\_o} \tag{9}$$

where `grad_z`, `phi_dir`, `grad_o` are 1d `numpy.ndarray` of size n_i

  - via affine transformation

$$\texttt{grad\_W = numpy.dot(grad\_z, val\_o.T)} \tag{10}$$

$$\texttt{grad\_w = grad\_z.copy()} \tag{11}$$

$$\texttt{grad\_o\_prev = numpy.dot(grad\_z, W)} \tag{12}$$

where `val_o`, `grad_w` are 1d `numpy.ndarray` of size n_i, `grad_o_prev` is 1d `numpy.ndarray` of size n_{i-1}, `grad_W` is 2d `numpy.ndarray` of shape (n_i, n_{i-1})

Gradient evaluation using backpropagation

- forward propagation - evaluate for each $(x^{(k)}, y^{(k)})$

$$g_\theta(x^{(k)}) = \phi_\theta^{[d]}(\psi_\theta^{[d]}(\cdots(\phi_\theta^{[1]}(\psi_\theta^{[1]}(x^{(k)})))))$$

- *backpropagation - evaluate partial derivatives backward*
  - evaluate gradient with respect to output of output layer $o^{[d]} = g_\theta(x^{(k)})$

$$\nabla_{o^{[d]}} f_k(\theta) = \nabla_{y_1} l(g_\theta(x^{(k)}), y^{(k)})$$

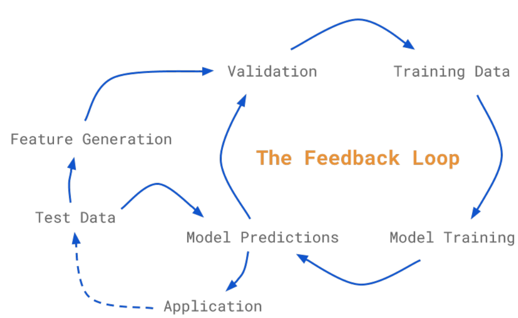
  - evaluate gradient with respect to input from that with respect to output using (1), or equivalently, using (5) $i.e.$, evaluate $\nabla_{z^{[i]}} f_k(\theta)$ from $\nabla_{o^{[i]}} f_k(\theta)$
  - evaluate gradient with respect to weights, bias, and intput of previous layer using (3), (4), & (2) or equivalently, using (7), (8), & (6) $i.e.$, evaluate $\nabla_{W^{[i]}} f_k(\theta)$, $\nabla_{w^{[i]}} f_k(\theta)$ & $\nabla_{o^{[i-1]}} f_k(\theta)$ from $\nabla_{z^{[i]}} f_k(\theta)$
  - repeat back to input layer to evaluate all

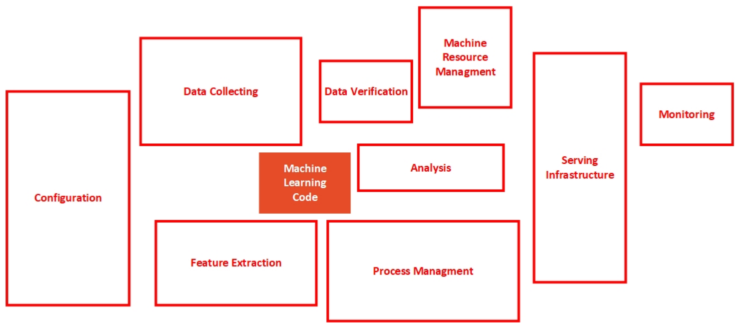
$$\nabla_{W^{[1]}} f_k(\theta), \nabla_{w^{[1]}} f_k(\theta), \ldots, \nabla_{W^{[d]}} f_k(\theta), \nabla_{w^{[d]}} f_k(\theta)$$

ML in Action

ML in practice

- define business problem - business objective, success metrics, establish baselines (early)
- data collection - data cleaning, validation & exploratory data analysis (EDA)
- feature engineering - based on domain expertise
- train/validation/test split - stratified sampling, chronological splits for time-series
- model selection or/and hyperparameter optimization
- monitoring, retraining & notification
- start simple, iterative fast (fail fast!), validate business impact - *e.g.*, A/B test

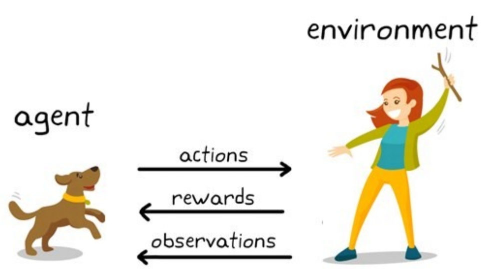


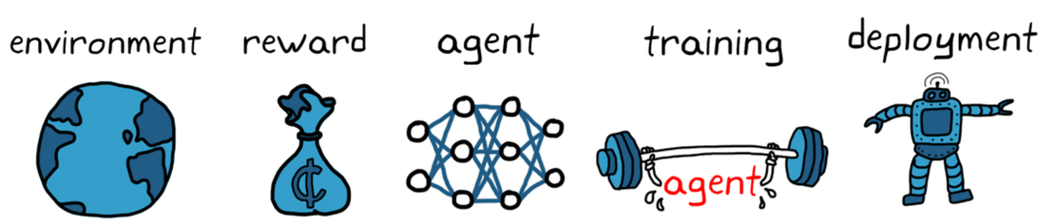


Reinforcement Learning

Reinforcement learning (RL)

- machine learning where agent learns how to take actions to achieve goal
  - by maximizing cumulative *reward*
  - while interacting with environment
- learning from interaction - foundational idea underlying all learning & intelligence
- differs from supervised learning
  - labeled input and output pairs *not* presented
  - sub-optimal actions need *not* be explicitly corrected
- focus is finding balance between exploration & exploitation





Why Deep RL?

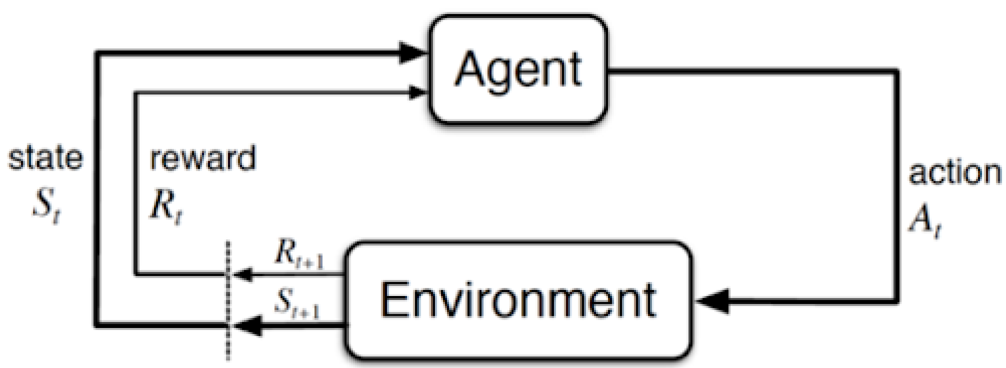
- Koray Kavukcuoglu (director of research at Deepmind) says

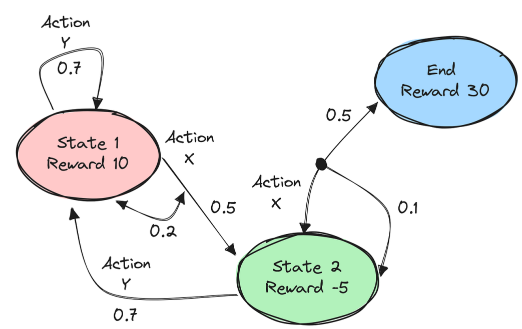
> If one of the goals we work for here is AI, then it is at the core of that. RL is a very general framework for learning sequential decision making tasks. And DL, on the other hand, is (of course) the best set of algorithms we have to learn representations. And combinations of these two different models is the best answer so far we have in terms of learning very good state representations of very challenging tasks that are not just for solving toy domains but actually to solve challenging real world problems.

MDP

Markov decision process (MDP)

- classical formulation of sequential decision making
  - actions influence not just immediate rewards, but also subsequent states, hence, involving delayed reward
  - need to trade-off immediate and delayed reward
- elements - *states*, *actions*, *reward*, and *return*
- agent interacts with environment
  - agent makes decision as to which action to take with knowledge of state it's in
  - action changes (state of) environment
  - agent receives reward

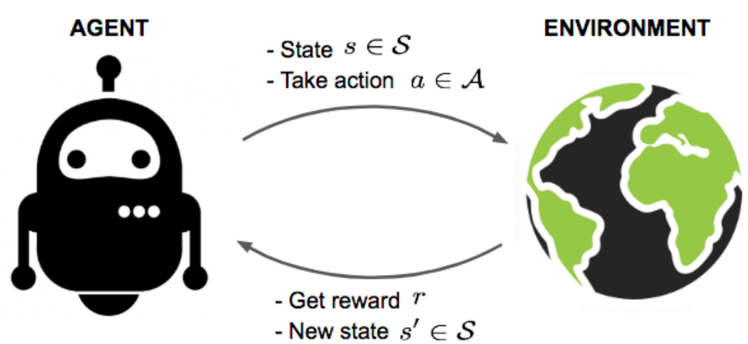


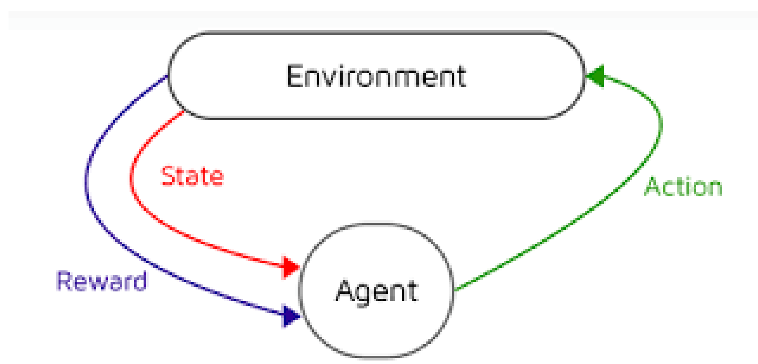


MDP & Markov property

- agent in *state* S_t takes *action* A_t at t
  - receives *reward* R_{t+1} (from environment)
  - environment transitions to state S_{t+1}
- sequence of random variables - $S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, S_3, A_3, \ldots$
- *Markov property* - $S_{t+1}, R_{t+1} | S_t, A_t, R_t, S_{t-1}, A_{t-1}, R_{t-1}, \ldots = S_{t+1}, R_{t+1} | S_t, A_t$
  - formally expressed (using PDF)

$$p\left(S_{t+1}, R_{t+1} \middle| S_t, A_t, R_t, S_{t-1}, A_{t-1}, R_{t-1}, \ldots\right) = p\left(S_{t+1}, R_{t+1} \middle| S_t, A_t\right)$$



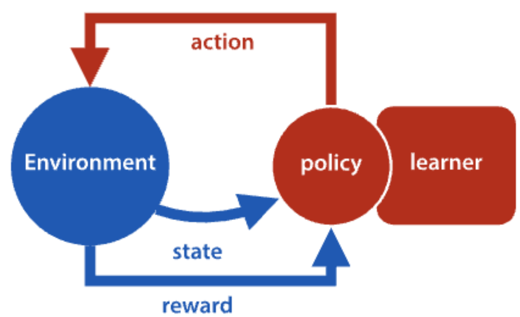


Policy & return

- *policy* - conditional probability of A_t given S_t

$$\pi(A|S) = p(A_t|S_t),$$

- *return* (at t) - $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k} = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots$
- $\gamma \in [0, 1]$ - *discount factor*
  - if $\gamma = 0$, myopic
  - if $\gamma = 1$, truly far-sighted
  - if $\gamma \in (0, 1)$, considers near-future rewards more importantly than those in far future



State value function & action value function

- *state value function* (sometimes referred to simply as *value function*)

$$v_\pi(s) = \mathop{\mathbf{E}}_{\pi,p} \{G_t |\, S_t = s\} = \mathop{\mathbf{E}}_{\pi,p} \left\{ \sum_{k=0}^{\infty} \gamma^k R_{t+k} \middle|\, S_t = s \right\}$$

  – function of state - expected return agent will get from s when following π

- *action value function* (sometimes referred to simply as *action function*)

$$q_\pi(s, a) = \mathop{\mathbf{E}}_{\pi,p} \{G_t |\, S_t = s, A_t = a\} = \mathop{\mathbf{E}}_{\pi,p} \left\{ \sum_{k=0}^{\infty} \gamma^k R_{t+k} \middle|\, S_t = s, A_t = a \right\}$$

  – function of state & action - expected return agent will get from s when agent takes a

- (most) RL algorithms (try to) maximize either of these functions - not maximizing immediate reward, but long-term return

Bellman

- Richard E. Bellman
  - introduced dynamic programming (DP) in 1953
  - proposed *Bellman equation* as necessary condition for optimality associated with DP

$$
\begin{aligned}
v_\pi(s) &\doteq \mathbb{E}_\pi[G_t \mid S_t = s] \\
&= \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right] \\
&= \mathbb{E}_\pi\left[R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \,\middle|\, S_t = s\right] \\
\overset{?}{\downarrow}\; &= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s', r|s, a) \left[r + \gamma \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \,\middle|\, S_{t+1} = s'\right]\right] \\
&= \sum_a \pi(a|s) \sum_{s', r} p(s', r|s, a) \Big[r + \gamma v_\pi(s')\Big], \quad \forall s \in \mathcal{S}, \qquad (3.12)
\end{aligned}
$$

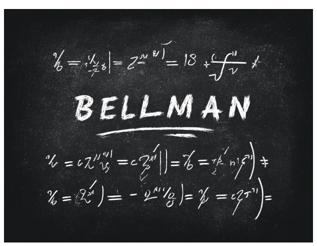
Bellman equations

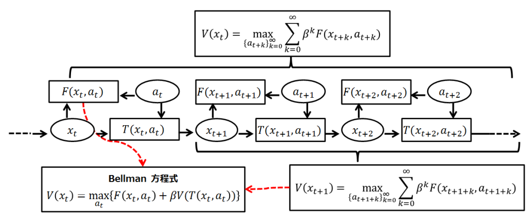
- *Bellman equation for state value function*

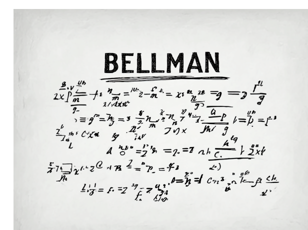
$$v_\pi(s) = \sum_a \pi(a|s) q_\pi(s,a) = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) \left(r + \gamma v_\pi(s')\right) \quad (13)$$

- *Bellman equation for action value function*

$$\begin{aligned} q_\pi(s,a) &= \sum_{s',r} p(s',r|s,a) \left(r + \gamma v_\pi(s')\right) \\ &= \sum_{s',r} p(s',r|s,a) \left(r + \gamma \sum_{a'} \pi(a'|s') q_\pi(s',a')\right) \end{aligned} \quad (14)$$







Bellman equation derviation - state value function

- Markov property implies
  - value functions only depend on current state & action taken
  - function value closely related to function values of next states
- these facts cleverly used to derive Bellman equations

$$
\begin{aligned}
v_\pi(s) &= \mathop{\mathbf{E}}_{\pi,p} \{G_t |\, S_t = s\} \\
&= \mathop{\mathbf{E}}_{A_t|S_t=s} \mathop{\mathbf{E}}_{\pi,p} \{G_t |\, S_t = s, A_t\} \\
&= \sum_a p(A_t = a|S_t = s) \mathop{\mathbf{E}}_{\pi,p} \{G_t |\, S_t = s, A_t = a\} \\
&= \sum_a \pi(a|s) \mathop{\mathbf{E}}_{\pi,p} \{G_t |\, S_t = s, A_t = a\} \\
&= \sum_a \pi(a|s) q_\pi(s, a) \qquad (15)
\end{aligned}
$$

Bellman equation derviation - action value function

$$
\begin{aligned}
q_\pi(s,a) &= \mathop{\mathbf{E}}_{\pi,p} \{ G_t |\, S_t = s, A_t = a \} \\
&= \mathop{\mathbf{E}}_{S_{t+1}, R_{t+1} | S_t = s, A_t = a} \mathop{\mathbf{E}}_{\pi,p} \{ G_t |\, S_t = s, A_t = a, S_{t+1}, R_{t+1} \} \\
&= \mathop{\mathbf{E}}_{S_{t+1}, R_{t+1} | S_t = s, A_t = a} \mathop{\mathbf{E}}_{\pi,p} \left\{ \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s, A_t = a, S_{t+1}, R_{t+1} \right\} \\
&= \mathop{\mathbf{E}}_{S_{t+1}, R_{t+1} | S_t = s, A_t = a} \mathop{\mathbf{E}}_{\pi,p} \left\{ R_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k R_{t+k+2} \middle| S_t = s, A_t = a, S_{t+1}, R_{t+1} \right\} \\
&= \sum_{s',r} p_{S_{t+1}, R_{t+1} | S_t, A_t}(s', r | s, a) \\
&\qquad \mathop{\mathbf{E}}_{\pi,p} \{ R_{t+1} + \gamma G_{t+1} |\, S_t = s, A_t = a, S_{t+1} = s', R_{t+1} = r \}
\end{aligned}
$$

$$
\begin{aligned}
&= \sum_{s',r} p_{S_{t+1},R_{t+1}|S_t,A_t}(s',r|s,a) \\
&\qquad \left(r + \gamma \mathop{\mathbf{E}}_{\pi,p} \left\{ G_{t+1} \middle| \, S_t = s, A_t = a, S_{t+1} = s', R_{t+1} = r \right\} \right) \\
&= \sum_{s',r} p_{S_{t+1},R_{t+1}|S_t,A_t}(s',r|s,a) \left(r + \gamma \mathop{\mathbf{E}}_{\pi,p} \left\{ G_{t+1} \middle| \, S_{t+1} = s' \right\} \right) \\
&= \sum_{s',r} p_{S_{t+1},R_{t+1}|S_t,A_t}(s',r|s,a) \left(r + \gamma v_\pi(s') \right) \qquad (16)
\end{aligned}
$$

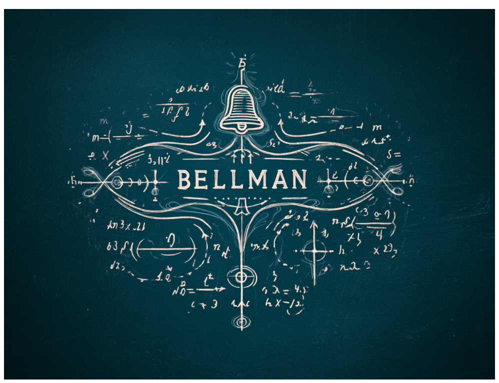
Optimal functions

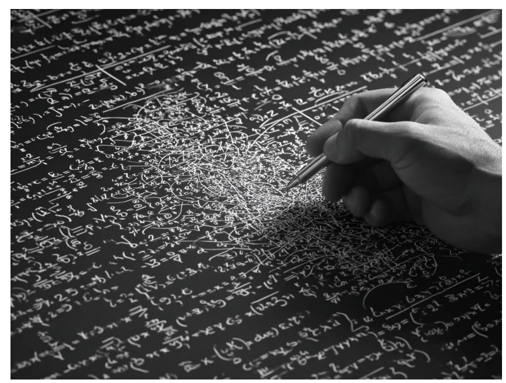
- define *optimal state-value function* as that of optimal policy π_*

$$v_*(s) = v_{\pi_*}(s) = \max_{\pi \in \Pi} v_\pi(s) \tag{17}$$

- (similarly) define *optimal action-value function* as that of π_*

$$q_*(s, a) = q_{\pi_*}(s, a) = \max_{\pi \in \Pi} q_\pi(s, a) \tag{18}$$





Bellman optimality equations

(17) & (18) with (15) & (16) imply

- *Bellman optimality equation for state value function*

$$v_*(s) = v_{\pi*}(a) = \max_{a \in \mathcal{A}} q_{\pi*}(s,a) = \max_{a \in \mathcal{A}} \sum_{s',r} p(s',r|s,a)\left(r + \gamma v_\pi(s')\right) \tag{19}$$

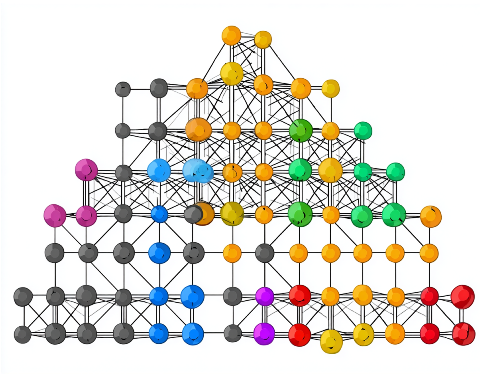
- *Bellman optimality equation for action value function*

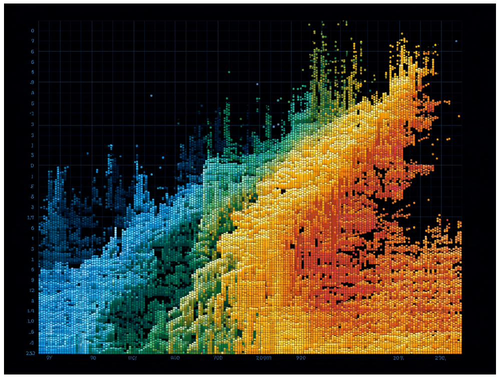
$$\begin{aligned} q_*(s,a) &= q_{\pi*}(s,a) = \sum_{s',r} p(s',r|s,a)\left(r + \gamma v_{\pi*}(s')\right) \\ &= \sum_{s',r} p(s',r|s,a)\left(r + \gamma \max_{a' \in \mathcal{A}} q_{\pi*}(s',a')\right) \end{aligned} \tag{20}$$

Dynamic Programming

Dynamic programming (DP)

- collection of algorithms to *compute optimal policies given perfect model of environment as MDP*
- provide *essential foundation for understanding of RL methods*
- all RL algorithms can be viewed as attempts to achieve much the same effect as DP
  - only with *less computation and without assuming perfect model of environment*
- key idea of RL in general
  - use of *value functions* to organize and structure search for good policies



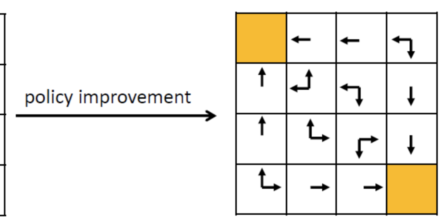


Policy evaluation (prediction)

- *policy evaluation* (in DP literature)
  - compute state-value function v_π for arbitrary policy π
  - also referred to as *prediction problem*
- existence and uniqueness of v_π guaranteed as long as either
  - $\gamma < 1$
  - eventual termination is guaranteed from all states under policy π
- policy evaluation algorithm uses fact that all state value functions satisfy Bellman equation (note resemblance to 13) - algorithm described in Table 1

$$v_{k+1}(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a) \left(r + \gamma v_k(s')\right)$$

0.0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0.0



Algorithm - iterative policy evaluation

Inputs: π, MDP
Algorithm parameters: $\theta > 0$ (small threshold determining accuracy of estimation)

Initialize $V(s) \in \mathbf{R}$ for all $s \in \mathcal{S}$ except that $V(\mathrm{terminal}) = 0$

Loop:
    $\Delta \leftarrow 0$
    For each $s \in \mathcal{S}$:
        $v \leftarrow V(s)$
        $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \left(r + \gamma V(s')\right)$
        $\Delta \leftarrow \max\{\Delta, |v - V(s)|\}$
until $\Delta < \theta$

Table 1: Iterative Policy Evaluation for estimating $V \sim v_\pi$

Policy iteration

- iterative process of improving policy to maximize value functions

- algorithm described in Table 2

Algorithm - policy iteration

Inputs: MDP
Algorithm parameters: $\theta > 0$ (small threshold determining accuracy of estimation)
1. Initialization
   $V(s) \in \mathbf{R}$ and $\pi(s) \in \mathcal{A}(s)$ for all $s \in \mathcal{S}$
2. Policy Evaluation
   Loop:
      $\Delta \leftarrow 0$
      For each $s \in \mathcal{S}$:
         $v \leftarrow V(s)$
         $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \left(r + \gamma V(s')\right)$
         $\Delta \leftarrow \max\{\Delta, |v - V(s)|\}$
   until $\Delta < \theta$
3. Policy Improvement
$u \leftarrow$ `true`
   For each $s \in \mathcal{S}$
      $b \leftarrow \pi(s)$
      $\pi(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \left(r + \gamma v_\pi(s')\right)$
      If $b \neq \pi(s)$, then $t \leftarrow$ `false`
   If u, then stop and return $V \sim v_*$ and $\pi \sim \pi_*$; else go to 2

Table 2: Policy Iteration (using iterative policy evaluation) for estimating $\pi \sim \pi_*$

Value iteration

- drawback to policy iteration
  - each iteration involves policy evaluation
- policy evaluation step can be truncated without losing convergence guarantees
- *value iteration*
  - policy evaluation is stopped after just one sweep by turning Bellman optimality equation (19) into update rule
  - can be written as simple update operation combining policy improvement and truncated policy evaluation steps

$$v_{k+1}(s) \leftarrow \max_{a \in \mathcal{A}} \sum_{s',r} p(s', r|s, a) \left(r + \gamma v_k(s')\right)$$

- (in-place version of) algorithm described in Table 3

Algorithm - value iteration

Inputs: MDP
Algorithm parameters: $\theta > 0$ (small threshold determining accuracy of estimation)

Initialize $V(s) \in$ **R** for all $s \in \mathcal{S}$ except that $V(\mathrm{terminal}) = 0$

Loop:
  $\Delta \leftarrow 0$
  For each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_{a \in \mathcal{A}(s)} \sum_{s',r} p(s',r|s,a) \left(r + \gamma V(s')\right)$
    $\Delta \leftarrow \max\{\Delta, |v - V(s)|\}$
until $\Delta < \theta$

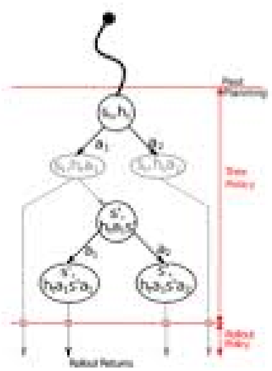
Output: deterministic policy π such that
  $\pi(s) = \mathrm{argmax}_{a \in \mathcal{A}(s)} \sum_{s',r} p(s',r|s,a) \left(r + \gamma V(s')\right)$

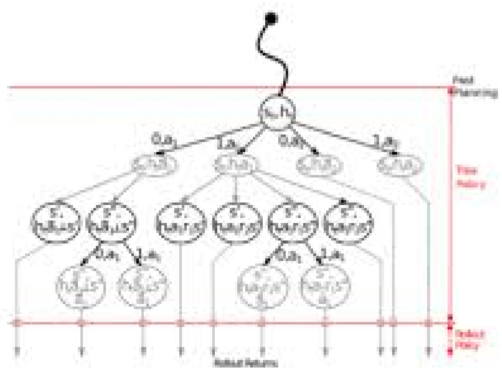
Table 3: Value Iteration for estimating $\pi \sim \pi_*$

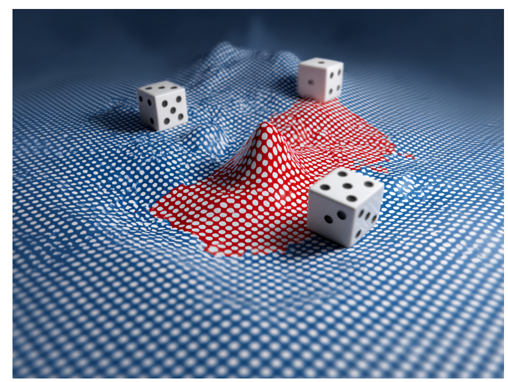
Monte Carlo Methods

Monte Carlo methods

- do not assume complete knowledge of environment
- require only experience sample sequences of states, actions & rewards
  - from *actual or simulated interaction with environment*
- require no prior knowledge of environment's dynamics
  - *not complete probability distributions* required for DP
  - yet can still attain optimal behavior
- simulation can be used







Monte Carlo prediction

- (simply) average returns observed after visits to each state
- Monte Carlo (MC) prediction methods - very similar but slightly different theoretical properties
  - *first-visit MC method* - most widely studied, dating back to 1940s
  - *every-visit MC method* - extends more naturally to function approximation and eligibility traces
- first-visit MC prediction algorithm described in Table 4

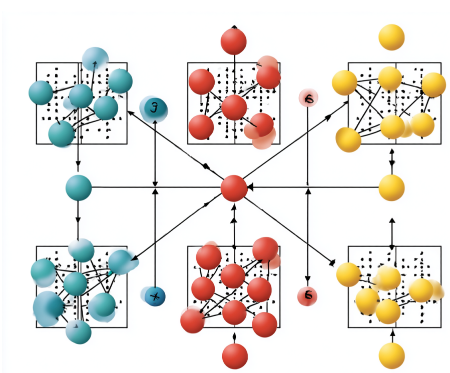
Algorithm - first-visit MC prediction

Inputs: π
 	
Initialize:
    $V(s) \in \mathbf{R}$ for all $s \in \mathcal{S}$
    $R(s) \leftarrow \texttt{list}()$ for all $s \in \mathcal{S}$
 	
Loop:
    Generate an episode following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$
    $G \leftarrow 0$
    Loop for each step of episode, $t + T - 1, T - 2, \ldots, 0$:
        $G \leftarrow \gamma G + R_{t+1}$
        If $S_t \not\in \{S_0, S_1, \ldots, S_{t-1}\}$:
            $R(S_t).\texttt{append}(G)$
            $V(S_t) \leftarrow R(S_t).\texttt{average}()$
Until a certain criterion is satisfied

Table 4: First-visit MC prediction for estimating $V \sim v_\pi$

Monte Carlo control

- proceed according to same pattern as DP, *i.e.*, according to idea of generalized policy iteration (GPI)
- maintain both approximate policy & approximate value functions
  - value functions repeatedly altered to more closely approximate value function for current policy
  - policy repeatedly improved with respect to current value function
- complete simple algorithm, called *Monte Carlo with Exploring Starts (ES)* described in Table 5



Algorithm - MC ES

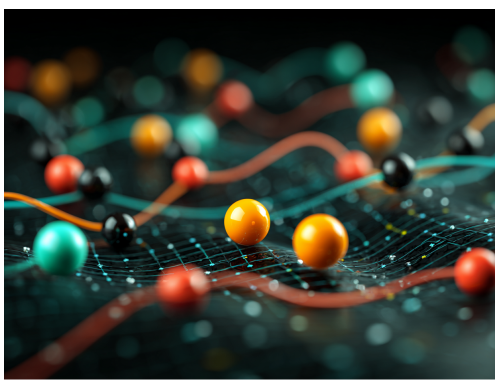
Initialize:
  $\pi(s) \in \mathcal{A}(s)$ for all $s \in \mathcal{S}$
  $Q(s,a) \in \mathbf{R}$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
  $R(s,a) \leftarrow \texttt{list}()$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$

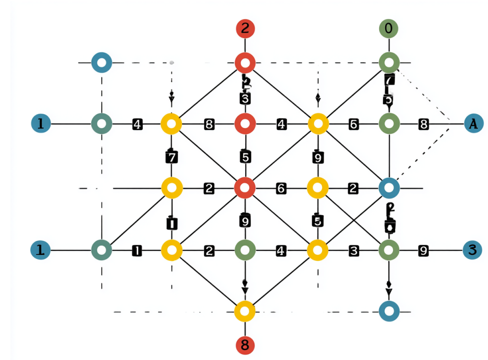
Loop:
  Choose $S_0 \in \mathcal{S}$, $A_0 \in \mathcal{A}(S_0)$ randomly such that all pairs have probability > 0
  Generate an episode from S_0, A_0 following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  Loop for each step of episode, $t + T - 1, T - 2, \ldots, 0$:
    $G \leftarrow \gamma G + R_{t+1}$
    If $S_t \not\in \{S_0, S_1, \ldots, S_{t-1}\}$:
      $R(S_t, A_t).\texttt{append}(G)$
      $Q(S_t, A_t) \leftarrow R(S_t, A_t).\texttt{average}()$
      $\pi(S_t) \leftarrow \mathrm{argmax}_{a \in \mathcal{A}(S_t)} Q(S_t, a)$
Until a certain criterion is satisfied

Table 5: MC ES for estimating $\pi \sim \pi_*$

Monte Carlo control without exploring starts

- want to avoid unlikely assumption of exploring starts
- only general way to ensure that all actions are selected infinitely often is for agent to continue to select them
- two approaches to ensure this
  - on-policy methods - attempt to evaluate or improve policy used to make decisions
  - off-policy methods - evaluate or improve policy different from used to generate data
- on-policy first-visit MC control using ϵ-greedy, not using unrealistic assumption of exploring starts, described in Table 6





Algorithm - on-policy first-visit MC control

Algorithm parameters: small $\epsilon > 0$

Initialize:
 $\pi(s) \in \mathcal{A}(s)$ for all $s \in \mathcal{S}$
 $Q(s, a) \in \mathbf{R}$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
 $R(s, a) \leftarrow \texttt{list}()$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$
Loop:
 Choose $S_0 \in \mathcal{S}$, $A_0 \in \mathcal{A}(S_0)$ randomly such that all pairs have probability > 0
 Generate an episode from S_0, A_0 following π: $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$
 $G \leftarrow 0$
 Loop for each step of episode, $t + T - 1, T - 2, \ldots, 0$:
  $G \leftarrow \gamma G + R_{t+1}$
  If $S_t \not\in \{S_0, S_1, \ldots, S_{t-1}\}$:
   $R(S_t, A_t).\texttt{append}(G)$
   $Q(S_t, A_t) \leftarrow R(S_t, A_t).\texttt{average}()$
   $A^* \leftarrow \mathrm{argmax}_{a \in \mathcal{A}(S_t)}$
   For all $a \in \mathcal{A}(S_t)$

$$\pi(a|S_t) \leftarrow \begin{cases} 1 - \epsilon + \epsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \epsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$$

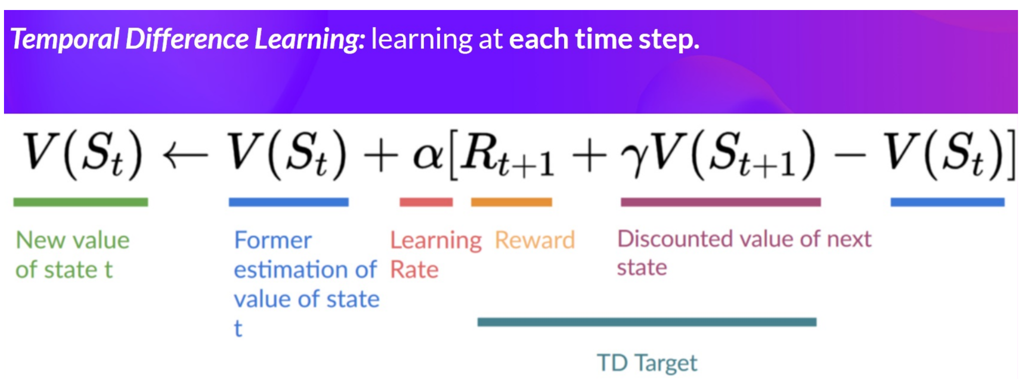
Until a certain criterion is satisfied

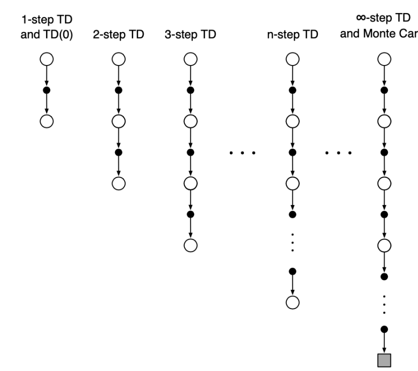
Table 6: On-policy first-visit MC control (for ϵ-soft policies) for estimating $\pi \sim \pi_*$

Temporal-difference Learning

Temporal-difference (TD) learning

- combination of MC ideas & DP ideas
  - like MC, learn directly from raw experience without model of environment's dynamics
  - like DP, update estimates based in part on other learned estimates, without waiting for final outcome - *they bootstrap*
- relationship between TD, DP & MC methods - recurring theme in theory of RL
- will start focusing on policy evaluation or prediction problem, $i.e.$, estimating v_π
- control problem (to find optimal policy)
  - DP, TD & MC methods all use some variation of generalized policy iteration (GPI)





TD prediction

- both TD & MC use experience to solve prediction problem
- simple every-visit MC method suitable for nonstationary environments

$$V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t)) = (1-\alpha)V(S_t) + \alpha G_t$$

- TD methods wait only until next time step
  - at $t+1$, form target and make update using reward R_{t+1} & estimate $V(S_{t+1})$
- TD(0) - one-step TD - simplest TD method

$$\begin{aligned} V(S_t) \quad \leftarrow \quad & V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)) \\ & = (1-\alpha)V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1})) \end{aligned} \tag{21}$$

  - TD(0) is special case of TD(λ) & n-step TD methods
- TD(0) described in Table 7 in procedural form

Algorithm - TD(0) for estimating v_π

Inputs: the policy π to be evaluated
Algorithm parameters: step size $\alpha \in (0, 1]$

Initialize:
    $V(s) \in$ **R** for all $s \in \mathcal{S}$ except that $V(\text{terminal}) = 0$

Loop for each episode:
    Initialize S
    Loop for each step of episode:
        $A \leftarrow$ action given by π for S
        Take action A, observe R, S'
        $V(S) \leftarrow (1 - \alpha)V(S) + \alpha(R + \gamma V(S'))$
        $S \leftarrow S'$
    until S is terminal
Until a certain criterion is satisfied

Table 7: TD(0) for estimating v_π.

TD error

- *TD error* - quantity in brackets in TD(0) update

$$\delta_t := R_{t+1} + \gamma V_t(S_{t+1}) - V_t(S_t) \tag{22}$$

  - difference between estimated value of S_t & better estimate $R_{t+1} + \gamma V(S_{t+1})$
  - arise in various forms throughout RL

- define *modified TD error*

$$\delta_t' := R_{t+1} + \gamma V_{t+1}(S_{t+1}) - V_t(S_t) \tag{23}$$

Monte Carlo error

- MC error
  - difference between return along path from t to terminal state & state-value function

$$G_t - V_t(S_t) = \sum_{k=t}^{T-1} \gamma^{k-t} \delta'_k = \sum_{k=0}^{T-t-1} \gamma^k \delta'_{k+t} \tag{24}$$

  - can be expressed as sum of discounted (modified) one-step TD errors.
- assuming that every V_t does not change during episode
  - δ_t coincides with δ'_t
  - hence, (24) becomes

$$G_t - V(S_t) = \sum_{k=t}^{T-1} \gamma^{k-t} \delta_k = \sum_{k=0}^{T-t-1} \gamma^k \delta_{k+t}. \tag{25}$$

MC error - derivation

- MC error

$$
\begin{aligned}
G_t - V_t(S_t) = R_{t+1} + \gamma G_{t+1} - V_t(S_t) & \\
= \quad & R_{t+1} + \gamma \left(G_{t+1} - V_{t+1}(S_{t+1}) + V_{t+1}(S_{t+1}) \right) - V_t(S_t) \\
= \quad & R_{t+1} + \gamma V_{t+1}(S_{t+1}) - V_t(S_t) + \gamma \left(G_{t+1} - V_{t+1}(S_{t+1}) \right) \\
= \quad & \delta'_t + \gamma \left(G_{t+1} - V_{t+1}(S_{t+1}) \right) \\
= \quad & \delta'_t + \gamma \delta'_{t+1} + \gamma^2 \left(G_{t+2} - V_{t+2}(S_{t+2}) \right) \\
= \quad & \delta'_t + \gamma \delta'_{t+1} + \gamma^2 \delta'_{t+2} + \cdots + \gamma^{T-t-2} \delta'_{T-2} + \gamma^{T-t-1} \left(G_{T-1} - V_{T-1}(S_{T-1}) \right) \\
= \quad & \delta'_t + \gamma \delta'_{t+1} + \gamma^2 \delta'_{t+2} + \cdots + \gamma^{T-t-2} \delta'_{T-2} + \gamma^{T-t-1} \left(R_T + \gamma V_T(S_T) - V_{T-1}(S_{T-1}) \right) \\
= \quad & \delta'_t + \gamma \delta'_{t+1} + \gamma^2 \delta'_{t+2} + \cdots + \gamma^{T-t-2} \delta'_{T-2} + \gamma^{T-t-1} \delta'_{T-1} \\
= \quad & \sum_{k=t}^{T-1} \gamma^{k-t} \delta'_k = \sum_{k=0}^{T-t-1} \gamma^k \delta'_{k+t}
\end{aligned}
$$

where fact that state-value function for terminal state, $V_{T-1}(S_T)$, is 0 is used

Sarsa - on-policy TD Control

- (as in all on-policy methods)
  - continually estimate q_π for behavior policy π
  - (at the same time) change π toward greediness with respect to q_π
- convergence properties depend on nature of policy's dependence on Q
  - examples of policies - ϵ-greedy or ϵ-soft
- converges with probability 1 to an optimal policy & optimal action-value function as long as
  - all state–action pairs are visited infinite number of times
  - policy converges in the limit to greedy policy
- algorithm is described in Table 8

Algorithm - sarsa for estimating $Q \sim q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$ and small $\epsilon > 0$

Initialize:
    $Q(s, a) \in$ **R** for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$ except $Q(\mathrm{terminal}, \cdot) = 0$

Loop for each episode:
    Initialize S
    Choose A from S using policy derived from Q ($e.g.$, ϵ-greedy)
    Loop for each step of episode:
        Take action A, observe R, S'
        Choose A' from S' using policy derived from Q ($e.g.$, ϵ-greedy)
        $Q(S, A) \leftarrow (1 - \alpha)Q(S, A) + \alpha(R + \gamma Q(S', A'))$
        $S \leftarrow S'$, $A \leftarrow A'$,
    until S is terminal
Until a certain criterion is satisfied

Table 8: Sarsa (on-policy TD control) for estimating $Q \sim q_*$

Q-learning - off-policy TD control

- development of off-policy TD control algorithm known as Q-learning (Watkins, 1989) - one of early breakthroughs in RL
- update defined by

$$
\begin{aligned}
Q(S_t, A_t) \quad \leftarrow \quad & Q(S_t, A_t) + \alpha \left(R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t) \right) \\
& = (1-\alpha) Q(S_t, A_t) + \alpha \left(R_{t+1} + \gamma \max_a Q(S_{t+1}, a) \right)
\end{aligned}
$$

- learned action-value function Q directly approximates optimal action-value function q_*, independent of policy being followed
  - dramatically simplifies analysis of algorithm & enabled early convergence proofs
- Q has been shown to converge with probability 1 to q_*
- algorithm described in Table 9

Algorithm - Q-learning for estimating $\pi \sim \pi_*$

Algorithm parameters: step size $\alpha \in (0,1]$ and small $\epsilon > 0$

Initialize:
    $Q(s,a) \in$ **R** for all $s \in \mathcal{S}$ and $a \in \mathcal{A}(s)$ except $Q(\mathrm{terminal}, \cdot) = 0$

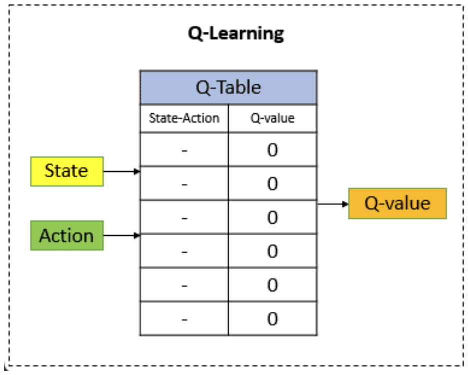
Loop for each episode:
    Initialize S
    Loop for each step of episode:
        Choose A from S using policy derived from Q ($e.g.$, ϵ-greedy)
        Take action A, observe R, S'
        $Q(S,A) \leftarrow (1-\alpha)Q(S,A) + \alpha(R + \gamma \max_{a \in \mathcal{A}(S')} Q(S',a))$
        $S \leftarrow S'$
    until S is terminal
Until a certain criterion is satisfied

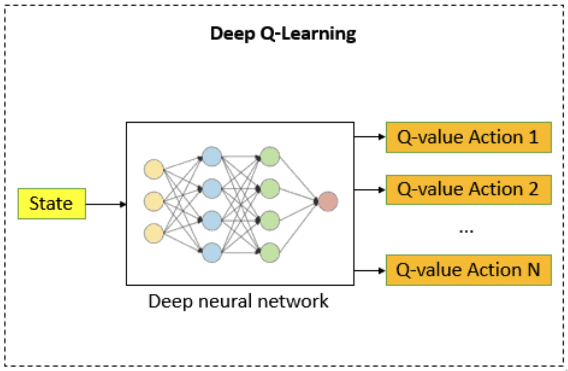
Table 9: Q-learning (off-policy TD control) for estimating $\pi \sim \pi_*$

Modern Reinforcement Learning

Deep Q-learning revolution

- problem with classical Q-learning
  - limited to small, discrete state spaces
  - Q-table becomes intractable for complex environments
  - cannot handle high-dimensional inputs, $e.g.$, images, continuous states
- deep Q-networks (DQN)
  - replace Q-table with deep neural network (DNN)
  - DNN approximates action-value function $Q(s, a)$
  - handle raw pixel inputs, continuous states
  - enables RL in complex environments, $e.g.$, Atari games, robotics

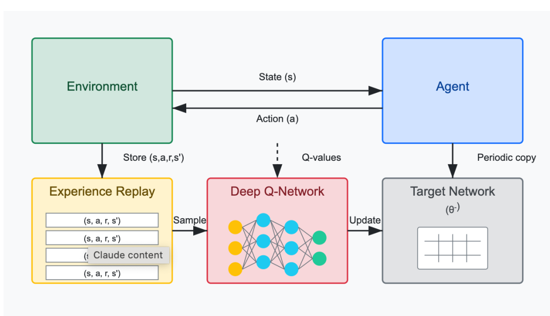


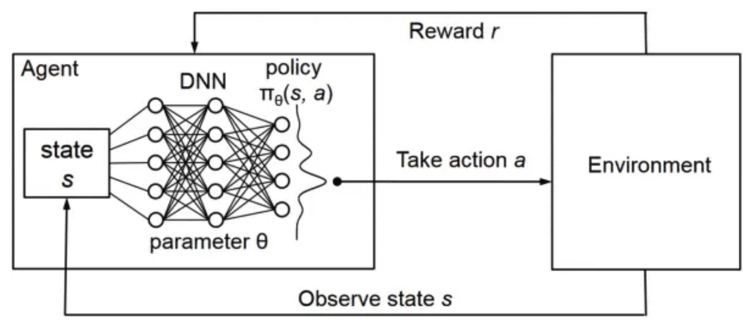


DQN architecture & key innovations

- experience replay
  - store transitions (s, a, r, s') in replay buffer & sample mini-batches for training
  - break correlation between consecutive samples to improve data efficiency and stability
- target network
  - separate target network for computing TD targets being updated periodically
  - reduce correlation between Q-values & targets to improve training stability
- DQN loss function

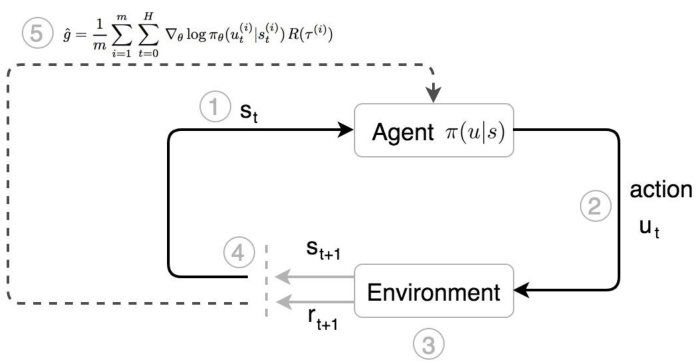
$$L(\theta) = \mathbf{E}((r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta))^2$$

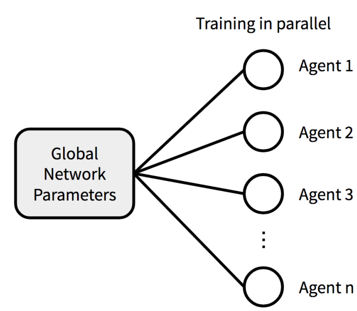




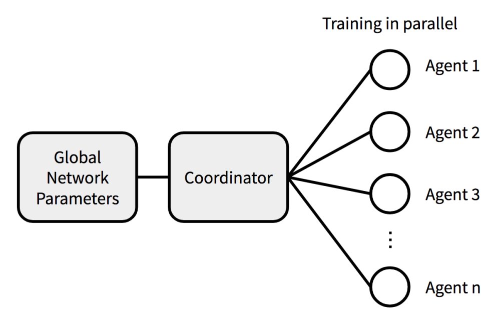
Policy gradient methods

- limitations of value-based approaches
  - indirect policy optimization
  - difficulty with continuous action spaces
  - may not find stochastic optimal policies
- policy gradient methods
  - direct policy optimization & natural handling of continuous actions
  - can learn stochastic policies & better convergence properties (in some cases)





A3C (Async)



A2C (Sync)

Policy gradient algorithm

- merit cuntion - $J(\theta) = \mathbf{E}(V(S_0)|\pi_\theta) = \mathbf{E}\left(\sum_{t=0}^{\infty} \gamma^t R_t \middle| \pi_\theta\right)$
- maximization problem formulation

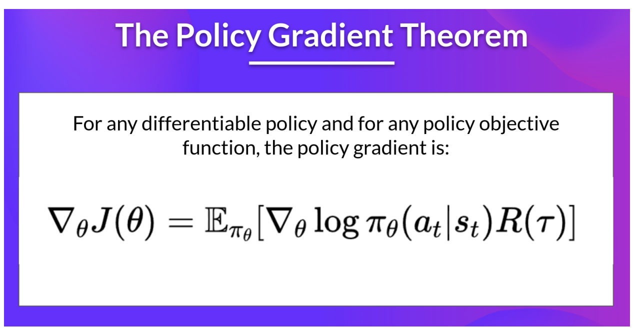
$$\begin{array}{ll} \text{maximize} & J(\theta) \\ \text{subject to} & \theta \in \Theta \end{array}$$

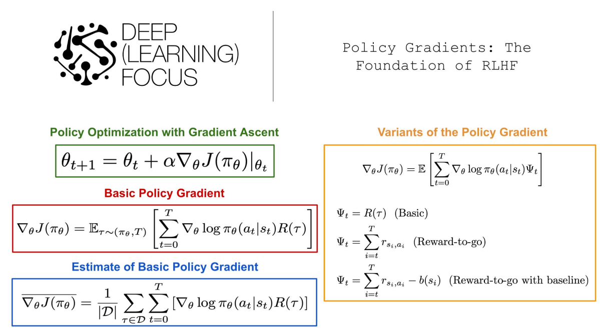
- REINFORCE algorithm

$$\theta^{k+1} = \theta^k + \alpha^k \nabla J(\theta^k)$$

where

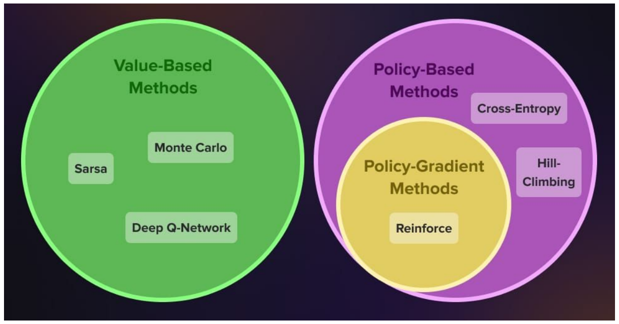
$$\nabla_\theta J(\theta) = \mathbf{E}(\nabla_\theta \log \pi(a|s;\theta) Q^\pi(s,a))$$

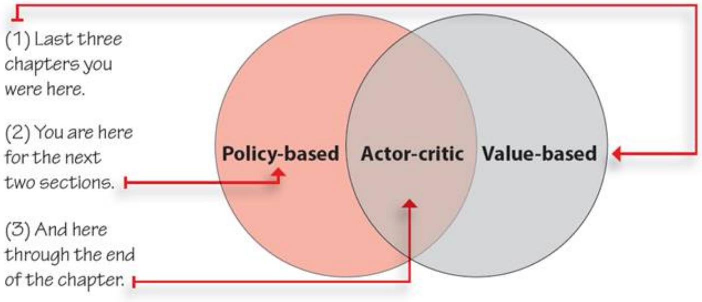




Q-learning vs policy gradients

- Q-learning
  - does *not* always work
  - usually *more sample-efficient* (when it works)
  - *challenge - exploration*
  - *no guarantee* for convergence
- policy gradients
  - *very general*, but suffers from *high variance*
  - requires *lots of samples*
  - converges to local minima of $J(\theta)$
  - *challenge - sample-efficiency*





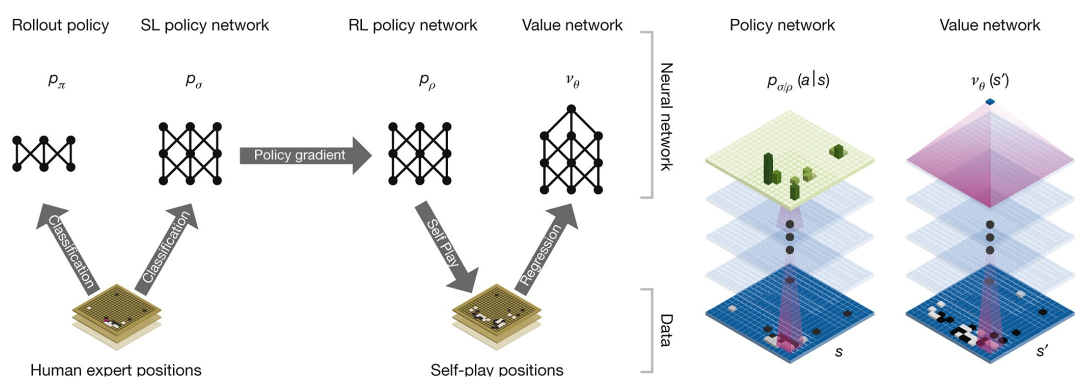
AlphaGo & AlphaGo Zero Technologies

AlphaGo

- Go board marked with 19×19 grid - $(19 \times 19)! = 361! \approx 1.44 \times 10^{768}$
  1437923258884890654832362511499863354754907538644755876127282765299227795534389618856841908003141196
  0714137944348905859683839682333043216077138088370565578796691924861827097800358990211005794501073330
  5079262777172275041226808677528136885057526541812043502150623466302643442673632627092764643302557772
  2695595343233942204301825548143785112222186834487969871267194205609533306413935710635197200721473378
  7338269803085351043174203653673779887217565513450041291061650506154496265581102824241428406627054585
  5623101563752892899924857388316647687165212001536218913733713768261861456295440900774337589490771443
  9917299937133680728459000034496420337066440853337001284286412654394495050773954560000000000000000000
  000000000000000000000000000000000000000000000000000000000000000000000
- deep reinforcement learning with Monte Carlo tree search
  - trained on thousands of years of Go game history
  - AlphaGo Zero learns by playing against itself
- development experience, insight, knowledge, know-how transferred to AlphaFold

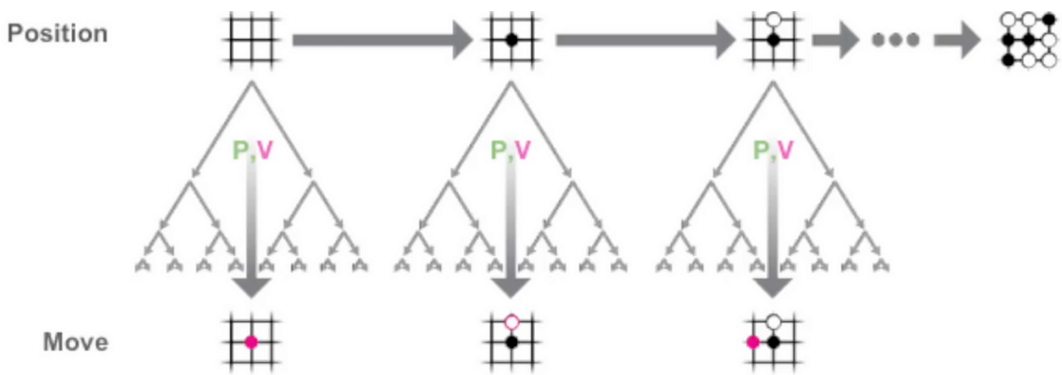
AlphaGo - hybrid approach - 2016

- components
  - policy network - predicts human expert moves
  - value network - evaluates board positions
  - Monte Carlo tree search (MCTS) - explores game tree
  - rollout policy - fast playouts for MCTS
- training process
  - supervised learning - train policy network on human games
  - RL - improve policy through self-play
  - regression - train value network on self-play positions



AlphaGo Zero - pure RL revolution - 2017

- breakthrough - no human knowledge
  - learns from scratch through self-play, no human game data or handcrafted features
  - much stronger than original AlphaGo
- simplified architecture
  - single neural network with two heads - policy head $\pi(a|s)$ & value head $v(s)$
- key innovations
  - residual NN - enable very deep networks
  - MCTS with NN - perfect integration
  - self-play curriculum - gradually increasing difficulty



Modern RL Applications & Industry Examples

Autonomous systems

- Waymo - Google
  - RL for trajectory planning and decision making
  - Simulation-based training with millions of scenarios
  - Integration with traditional planning algorithms
- Tesla Autopilot
  - RL for lane changes and complex driving scenarios
  - Real-world data collection and training

Gaming & entertainment

- OpenAI Five for playing Dota 2
  - complex multi-agent environment
  - long-term planning (45+ minute games)
- DeepMind AlphaStar for playing StarCraft II
  - league-based training, population-based methods
  - partial observability challenges
  - human-level performance

Robotics

- Boston Dynamics
  - RL for dynamic locomotion
  - sim-to-real transfer
  - robust control policies
- Covariant - warehouse automation
  - RL for robotic picking and manipulation
  - real-world deployment in warehouses
  - continuous learning from experience

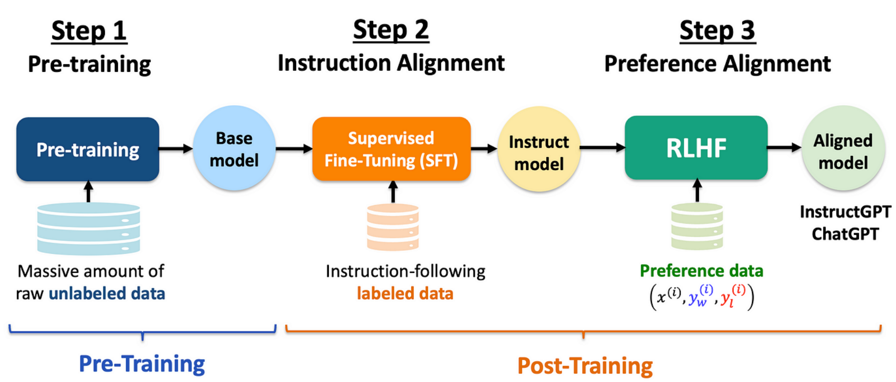
Finance & trading

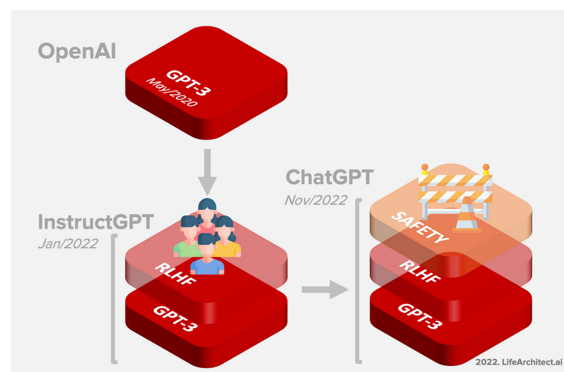
- JP Morgan Chase
  - algorithmic trading with RL
  - portfolio optimization
  - risk management
- Two Sigma, Renaissance Technologies
  - market making and execution
  - multi-agent trading environments

LLM & RL

RLHF - RL from human feedback

- ChatGPT, GPT-4 training pipelines
  - supervised fine-tuning - train on human demonstrations
  - reward model training - learn human preferences
- key components
  - reward model - predicts human preferences
  - KL penalty - prevents deviation from original model
  - constitutional AI - self-improvement through AI feedback





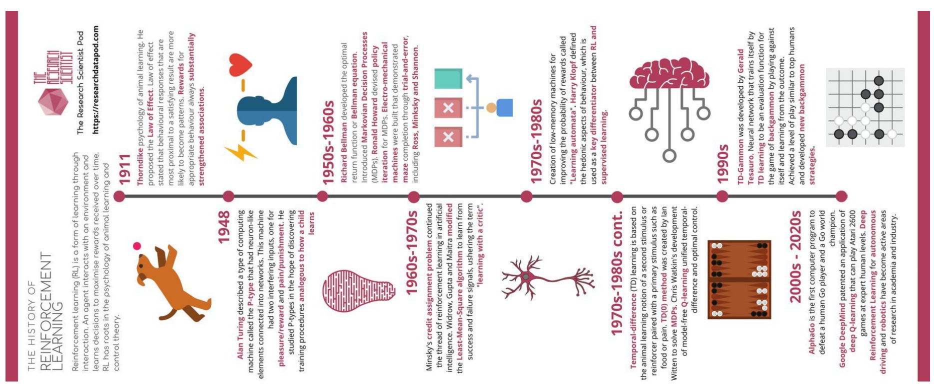
Applications in LLMs

- OpenAI - ChatGPT/GPT-4
  - RLHF for helpful, harmless, honest responses
  - massive scale PPO training
  - human preference learning
- Anthropic - Claude
  - constitutional AI methods
  - self-supervised preference learning
  - scalable oversight techniques

RL Evolution

Classical to modern RL

- key progressions
  - tabular $\rightarrow$ function approximation $\rightarrow$ DNN / model-free $\rightarrow$ model-based $\rightarrow$ hybrid
  - single agent $\rightarrow$ multi-agent $\rightarrow$ large-scale systems
- core principles *intact*
  - exploration vs exploitation trade-off / Bellman equations & Bellman optimality
  - policy improvement & evaluation / generalized policy iteration (GPI)
- modern additions
  - scale and compute power / human feedback integration
  - safety & robustness considerations / multi-modal and foundation models ($e.g.$, LLM)



Selected References & Sources

Selected references & sources

- Robert H. Kane "Quest for Meaning: Values, Ethics, and the Modern Experience" 2013
- Michael J. Sandel "Justice: What's the Right Thing to Do?" 2009
- Daniel Kahneman "Thinking, Fast and Slow" 2011
- Yuval Noah Harari "Sapiens: A Brief History of Humankind" 2014
- M. Shanahan "Talking About Large Language Models" 2022
- A.Y. Halevry, P. Norvig, and F. Pereira "Unreasonable Effectiveness of Data" 2009
- A. Vaswani, et al. "Attention is all you need" @ NeurIPS 2017
- S. Yin, et. al. "A Survey on Multimodal LLMs" 2023
- Chris Miller "Chip War: The Fight for the World's Most Critical Technology" 2022

- CEOs, CTOs, CFOs, COOs, CMOs & CCOs @ startup companies in Silicon Valley
- VCs on Sand Hill Road - Palo Alto, Menlo Park, Woodside in California, USA

References

References

[Bis06] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[HTF01] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2001.

[SB18] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Bradford Books, 2nd edition, 2018.

Thank You