6.036 Notes Spring 2024

① Types of learning

supervised
- regression
- classification

unsupervised
- clustering
- density estimation (probability distribution)
- dimensionality reduction

sequence : mapping from input sequence to output sequence

reinforcement : maps states to optimal actions based on rewards

structural error - bad model

estimation error - lack of data

Loss

$$0\text{-}1 \text{ Loss} = \begin{cases} 0 & g = a \\ 1 & \text{guess } g \neq \text{answer } a \end{cases}$$

$$\text{Squared Loss} = (g-a)^2$$

$$\text{Absolute Loss} = |g-a|$$

$$\text{Asymmetric Loss} \begin{cases} x & \text{if } g=1 \text{ and } a=0 \\ y & \text{if } g=0 \text{ and } a=1 \\ 0 & \text{else} \end{cases}$$

to prioritize some mistakes over others

usually we want to minimize expected loss

Matrix Derivatives

$\partial y/\partial x$ is 1×1 scalar

$\partial y/\partial \underset{n\times 1}{\mathbf{x}}$ is $\begin{bmatrix} \partial y/\partial x_1 \\ \vdots \\ \partial y/\partial x_n \end{bmatrix}$ $n\times 1$ vector

$\partial \underset{m\times 1}{\mathbf{y}}/\partial x$ is $\begin{bmatrix} \partial y_1/\partial x & \partial y_2/\partial x & \cdots & \partial y_m/\partial x \end{bmatrix}$ $1\times m$ vector

$\partial \mathbf{y}/\partial \mathbf{x}$ is $\begin{bmatrix} \partial y_1/\partial x_1 & \partial y_2/\partial x_1 & \cdots & \partial y_m/\partial x_1 \\ \vdots & & & \vdots \\ \partial y_1/\partial x_n & & & \partial y_m/\partial x_n \end{bmatrix}$ $n\times m$ matrix

$$\frac{\partial x}{\partial A}_{n\times m} := \begin{bmatrix} \partial x/\partial A_{11} & \partial x/\partial A_{12} & \cdots & \partial x/\partial A_{1m} \\ \vdots & & & \vdots \\ \partial x/\partial A_{n1} & & \cdots & \partial x/\partial A_{nm} \end{bmatrix}_{n\times m \text{ matrix}}$$

$$\frac{\partial a}{\partial x} = 0 \quad \leftarrow a \text{ unrelated to } x \qquad \frac{\partial x}{\partial x} = 1 \qquad \frac{\partial Ax}{\partial x} = A^T \qquad \frac{\partial x^T A}{\partial x} = A$$

$$\frac{\partial vu}{\partial x} = v\frac{\partial u}{\partial x} + \frac{\partial v}{\partial x}u^T \quad \leftarrow \text{Product Rule.}$$

$$\frac{\partial g(u)}{\partial x} = \frac{\partial g(u)}{\partial u} \cdot \frac{\partial u}{\partial x} \quad \leftarrow \text{Chain Rule}$$

$$\frac{\partial u^T v}{\partial x} = \frac{\partial u}{\partial x}v + \frac{\partial v}{\partial x}u \qquad \frac{\partial u^T}{\partial x} = \left(\frac{\partial u}{\partial x}\right)^T$$

(2) **linear regression**

incorporate θ_0 into here via extra coordinate

$$J(\theta) = \frac{1}{n}\sum_{i=1}^{n}\left(\theta^T x^{(i)} - y^{(i)}\right)^2$$

$$\nabla_\theta J = \frac{2}{n}\tilde{X}^T(\tilde{X}\theta - \tilde{Y}) \Rightarrow \text{set to } 0$$

$$\Rightarrow \boxed{\theta = (\tilde{X}^T\tilde{X})^{-1}\tilde{X}^T\tilde{Y}}$$

$$\tilde{X} = n\begin{bmatrix} \text{one point} \\ \text{all first coord.} \end{bmatrix}_{d}$$

$$\tilde{Y} = n\begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(n)} \end{bmatrix} \text{ actual}$$

$$\theta = d\begin{bmatrix} \theta_1 \\ \vdots \\ \theta_d \end{bmatrix} \text{ prediction parameters}$$

↓

regularize by **ridge regression**

$$\Rightarrow \theta = (\tilde{X}^T\tilde{X} + n\lambda I)^{-1}\tilde{X}^T\tilde{Y}$$

for

$$J_{ridge}(\theta,\theta_0) = \frac{1}{n}\sum_{i=1}^{n}\left(\theta^T x^{(i)} - y^{(i)}\right)^2 + \lambda\|\theta\|^2$$

$\Rightarrow$ avoids singularities.

cross validation

- break D into k batches D_k
  - → for each k, train using D/D_k and validate on D_k.

return $\frac{1}{k}\sum_{i=1}^{k}$ [error using model trained with D/D_k on D_k]

(3) gradient descent

move in direction opposed to gradient, η at a time.

when not improving more than ε per step, STOP.

→ many dimensions.

$$\nabla_\theta f = \begin{bmatrix} \partial f/\partial \theta_1 \\ \vdots \\ \partial f/\partial \theta_m \end{bmatrix}$$

$$\theta^{(t)} = \theta^{(t-1)} - \eta \nabla_\theta f(\theta^{(t-1)})$$

STOP at $|f(\theta^{(t)}) - f(\theta^{(t-1)})| < \varepsilon$.

for linear regression

$$\theta^{(t)} = \theta^{(t-1)} - \eta \underbrace{\frac{2}{n}\sum_{i=1}^{n}\left(\left[\theta^{(t-1)}\right]^T x^{(i)} - y^{(i)}\right)x^{(i)}}_{\sum \nabla_\theta J}$$

for ridge regression

$$J_{ridge}(\theta, \theta_0) = \frac{1}{n}\sum_{i=1}^{n}\left(\theta^T x^{(i)} + \theta_0 - y^{(i)}\right) + \lambda\|\theta\|^2$$

$$\nabla_\theta J_{ridge}(\theta, \theta_0) = \frac{2}{n}\sum_{i=1}^{n}\left(\theta^T x^{(i)} + \theta_0 - y^{(i)}\right)x^{(i)} + 2\lambda\theta$$

update θ, θ_0 separately:

$$\frac{\partial J_{ridge}(\theta, \theta_0)}{\partial \theta_0} = \frac{2}{n}\sum_{i=1}^{n}\left(\theta^T x^{(i)} + \theta_0 - y^{(i)}\right)$$

stochastic gradient descent

$f(\theta) = \sum_{i=1}^{n} f_i(\theta)$ — objective

pick a random parameter i to change

$$f_i \Rightarrow \theta^{(t)} = \theta^{(t-1)} - \eta(t) \nabla_\theta f_i(\theta^{(t-1)})$$

mini-batch: random sample size $\lceil \frac{n}{k} \rceil$ from n data points

if f convex, $\eta(t)$ satisfies $\sum_{t=1}^{\infty} \eta(t) = \infty$

$\sum_{t=1}^{\infty} \eta^2(t) < \infty$,

stochastic gradient descent converges with probability 1 to optimal θ.

④ linear classifier → chooses 1 or 0.

$$E_n(h) = \frac{1}{n} \sum_{i=1}^{n} \begin{cases} 1 & h(x^{(i)}) \neq y^{(i)} \\ 0 & \text{otherwise} \end{cases}$$

training error

usually, $h(x; \theta, \theta_0) = \text{sign}(\theta^T x + \theta_0)$.

↳ but usually NP-hard to find θ, θ_0

sigmoid $\sigma(z) = \frac{1}{1+e^{-z}}$

can predict 1 for $\sigma(z) > 0.5$, 0 otherwise

where $z = \theta^T x + \theta_0$

want to maximize

$$\prod_{i=1}^{n} \begin{cases} g^{(i)} & \text{if } y^{(i)} = 1 \\ 1 - g^{(i)} & \text{otherwise} \end{cases}$$

→ $\mathbb{P}$(both 1) ✓

→ $\mathbb{P}$(both 0) ✓

} right

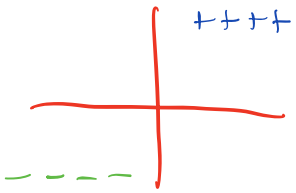
$$= \prod_{i=1}^{n} g^{(i)\,y^{(i)}} \left(1-g^{(i)}\right)^{(1-y^{(i)})}$$

maximized when $\log\left(\prod_{i=1}^{n} g^{(i)\,y^{(i)}} \left(1-g^{(i)}\right)^{(1-y^{(i)})} \right)$ is maximized

$$\Downarrow$$

i.e. minimize $\left[\sum_{i=1}^{n} -y^{(i)} \log g^{(i)} - (1-y^{(i)}) \log\left(1-g^{(i)}\right) \right] = \sum_{i=1}^{n} \mathcal{L}_{nll}\left(g^{(i)}, y^{(i)}\right)$

must have regularization → otherwise $\theta \to \infty$ is optimal in



case.

$$J_{lr}(\theta, \theta_0; \mathcal{D}) = \left(\frac{1}{n} \sum_{i=1}^{n} \mathcal{L}_{nll}\left(\sigma(\theta^T x^{(i)} + \theta_0), y^{(i)}\right) \right) + \lambda \|\theta\|^2$$

$$\nabla_\theta J_{lr}(\theta, \theta_0) = \frac{1}{n} \sum_{i=1}^{n} \left(g^{(i)} - y^{(i)} \right) x^{(i)} + 2\lambda\theta$$

$$\frac{\partial J_{lr}(\theta, \theta_0)}{\partial \theta_0} = \frac{1}{n} \sum_{i=1}^{n} \left(g^{(i)} - y^{(i)} \right)$$

↳ gradient descent!! possible since NLL (negative log likelihood) function is convex.

multiple classes (k, instead of 2)

- training label = one-hot vector = all 0's but 1 at i^{th} spot if in class i.
- finding θ, θ_0 s.t. $z = \theta^T x + \theta_0$ then

$$y = h(x; \theta, \theta_0) = \text{softmax}(z) = \begin{bmatrix} \exp(z_1) / \sum \exp(z_i) \\ \vdots \\ \exp(z_k) / \sum \exp(z_i) \end{bmatrix}$$

$$\mathcal{L}_{nllm}(g, y) = -\sum_{i=1}^{k} y_i \cdot \log(g_k)$$

↑ probability distribution vector

when evaluating accuracy, we check

$$A(h;D) = 1 - \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}_{0,1}\left(g^{(i)}, y^{(i)}\right)$$

↑ no probability deterministic

(5) Feature Representation

$[x] \rightarrow [x, x^2, x^3]$ ← gives more possibility for separability.

$[x, y, z] \rightarrow [x^2, xy, y^2, yz, z^2, xz, x, y, z]$

can also encode using dataset D with n points $x^{(1)}, x^{(2)}, \ldots, x^{(n)}$

$$\phi([x_1, x_2, \ldots, x_d]) \mapsto \left[f_{x^{(1)}}(x), f_{x^{(2)}}(x), \ldots, f_{x^{(n)}}(x)\right]^T, \quad f_p(x) = e^{-\beta\|p-x\|^2}$$

(normalizer)

discrete encodings

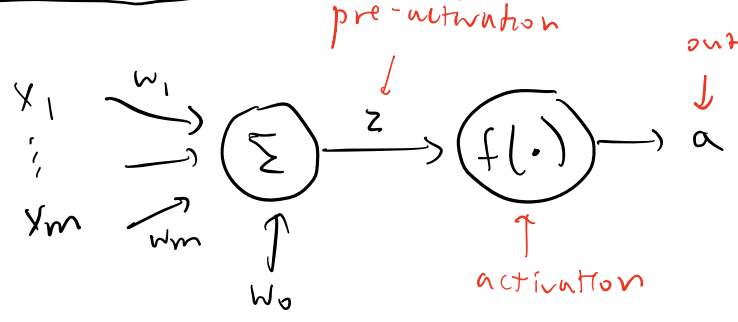
- ordering + mapping → numeric encoding
- ordering but no natural mapping → thermometer
- factoring ex) model + year → factor into two features, encode each one
- no ordering/categorical → one-hot
- binary code (00, 01, 10, 11) is usually BAD.

normalization can help: $\phi(x) = \frac{x-\bar{x}}{\sigma}$

(0, 0, 0, 0)
(1, 0, 0, 0)
(1, 1, 0, 0)
(1, 1, 1, 0)
(1, 1, 1, 1)

text → bag of words (one-hot for each word)

(6) Neural Network



for one neuron want to minimize

$$J(w, w_0) = \sum_i \mathcal{L}\left(NN(x^{(i)}, w, w_0), y^{(i)}\right)$$

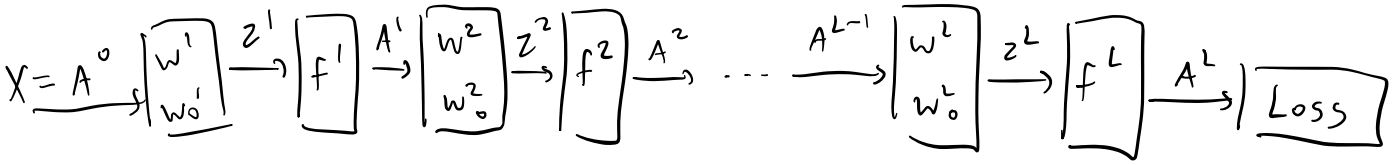
many neurons per layer

using $A = f(Z) = f(W^T X + W_0)$

activation; W^T: $(m \times n$ matrix$)^T$; W_0: $n \times 1$ vector

inputs to ℓ^{th} layer $= m^\ell$

outputs to ℓ^{th} layer $= n^\ell = m^{\ell+1}$



activation

$\text{step}(z) = \begin{cases} 0 & \text{if } z<0 \\ 1 & \text{if } z \geq 0 \end{cases}$ $\quad \text{ReLU}(z) = \begin{cases} 0 & \text{if } z<0 \\ z & \text{if } z \geq 0 \end{cases}$ $\quad \sigma(z)$ $\quad \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$ $\quad \text{softmax}(z)$

back-propagation

do forward pass, then calculate "blame"

$$\frac{\partial \text{loss}}{\partial W^\ell} = A^{\ell-1} \left(\frac{\partial \text{loss}}{\partial Z^\ell} \right)^T$$

$m^\ell \times n^\ell \quad m^\ell \times 1 \quad 1 \times n^\ell$

$\partial \text{loss}/\partial A^\ell =$ if $\ell < L$ then $\partial Z^{\ell+1}/\partial A^\ell \cdot \partial \text{loss}/\partial Z^{\ell+1}$ else $\partial \text{loss}/\partial A^L$

$\partial \text{loss}/\partial Z^\ell = \partial A^\ell/\partial Z^\ell \cdot \partial \text{loss}/\partial A^\ell$

$$\frac{\partial \text{loss}}{\partial Z^\ell} = \frac{\partial A^\ell}{\partial Z^\ell} \cdot \frac{\partial Z^{\ell+1}}{\partial A^\ell} \cdot \frac{\partial A^{\ell+1}}{\partial Z^{\ell+1}} \cdot \cdots \cdot \frac{\partial A^L}{\partial Z^L} \cdot \frac{\partial \text{loss}}{\partial A^L}$$

$$= \frac{\partial A^\ell}{\partial Z^\ell} \cdot W^{\ell+1} \cdot \frac{\partial A^{\ell+1}}{\partial Z^{\ell+1}} \cdot \cdots \cdot W^L \cdot \frac{\partial \text{loss}}{\partial A^L}$$

$\partial \text{loss}/\partial W^\ell = A^{\ell-1} \cdot (\partial \text{loss}/\partial Z^\ell)^T$

$\partial \text{loss}/\partial W_0^\ell = \partial \text{loss}/\partial Z^\ell$

$W^\ell = W^\ell - \eta(t)\, \partial \text{loss}/\partial W^\ell$

$W_0^\ell = W_0^\ell - \eta(t)\, \partial \text{loss}/\partial W_0^\ell$

parameter optimization

- batches
- adaptive step-size

regularization

- early stopping
- ridge regression
  - → negative feedback on $\|W\|^2$
- add random noise to training data
- normalize data
- dropout
  - → for each j, consider a_j^ℓ's for each ℓ and set to 0 with probability p ($= \frac{1}{2}$ usual) per training iteration

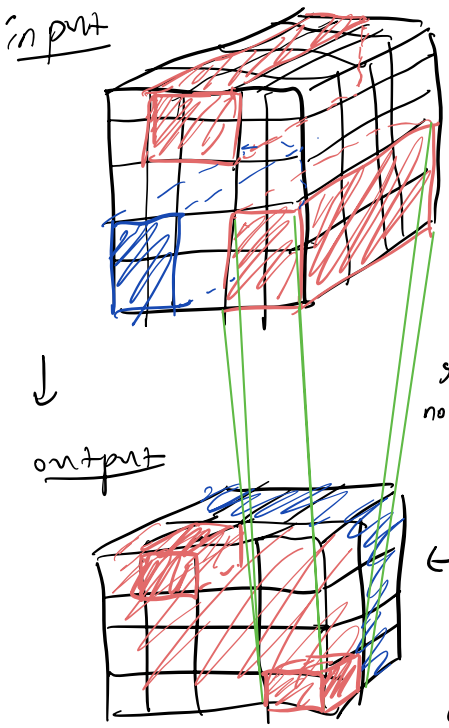
7 Convolutional Neural Networks

spatial locality – convolutional layer

translational invariance – max pooling layer

filters

starting with $n \times n$ image, separate into $n \times n \times 3$ based on RGB.



moves thru input, convolving with each involved pixel.

- stride determines step size
- padding on outside (with 0 typically) to generate similar image size

a layer ℓ is defined by

- number of filters m^ℓ
- filter size $k^\ell \times k^\ell \times m^{\ell-1}$ + 1 bias per filter
- stride s^ℓ
- input size $n^{\ell-1} \times n^{\ell-1} \times m^{\ell-1}$
- padding p^ℓ on every side

→ OUTPUT is $n^\ell \times n^\ell \times m^\ell$

↳ $n^\ell = \lceil (n^{\ell-1} + 2p^\ell - (k^\ell - 1))/s^\ell \rceil$

max pooling – take max of all cells in range

usually: input → (convolution + ReLU + pooling) → fully-connected → softmax.
many times

backpropagation

for something like $Z_i^\ell = W^{\ell T} A_{[i-\lfloor k/2 \rfloor : i+\lfloor k/2 \rfloor]}$,

i^{th} column of $\frac{\partial Z^\ell}{\partial W^\ell} = A_{[i-\lfloor k/2 \rfloor : i+\lfloor k/2 \rfloor]}$ ← $k \times 1$ column vector

$$\nabla_{(x,y)} \max(x, y) = \begin{cases} (1,0) & x > y \\ (0,1) & y > x \end{cases}$$

(8) Transformers

Semantic vector embeddings (word2vec): $v_{paris} - v_{france} \approx v_{rome} - v_{italy}$

$$p(k|q) = \text{softmax}([q^T k_1, q^T k_2, \dots, q^T k_n])$$

(q: query; $k_1, k_2, \dots, k_n$: other keys)

↝ roughly higher softmax for closer keys

$$\text{attention}(q, k, v) = \sum_k p(k|q)\, v(k)$$

(q: query; k: key; v: value; $\sum_k$: all other keys; $v(k)$: respective value)

mask can be simply which tokens are used in computation

Formally, L transformer blocks

transformer block: $\mathbb{R}^{n \times d} \to \mathbb{R}^{n \times d}$

params: n = # tokens, d = token dimension, θ = model parameters

each token $x^{(i)} \in \mathbb{R}^{d \times 1}$

$Q(x^{(i)})$ (query) $K(x^{(i)})$ (key) $V(x^{(i)})$ (value)

learnable weight matrices

$W_{h,q}$ $W_{h,k}$ $W_{h,v} \in \mathbb{R}^{d \times d_k}$

① attention weights

$$\alpha_{ij} = \text{softmax}_j\left(\frac{Q^T(x^{(i)})\, K(x^{(j)})}{\sqrt{d_k}}\right)$$

② $$y^{(i)} = \sum_{j=1}^{n} \alpha_{ij}^{(h)} V^{(h)}(x^{(j)})$$

③ $$u'^{(i)} = \sum_{h=1}^{H} W_{h,c}^T\, y^{(i)}$$

④ $$u^{(i)} = \text{LayerNorm}(x^{(i)} + u'^{(i)};\ \gamma_1, \beta_1)$$

⑤ $$z'^{(i)} = W_2^T\, \text{ReLU}(W_1^T u^{(i)})$$

⑥ Final $$z^{(i)} = \text{LayerNorm}(z'^{(i)} + u^{(i)};\ \gamma_2, \beta_2)$$

h is an index over each transformer head $(1, 2, \dots, H)$

$W_{h,c} \in \mathbb{R}^{d_k \times d}$ weight matrix for heads

$W_1 \in \mathbb{R}^{d \times m}$ $W_2 \in \mathbb{R}^{m \times d}$

(9) Non Parametric Methods

k-nearest neighbor: classify each point in region as majority vote of k nearest training points

decision trees: focus on CART/ID3 – axis-aligned and fit constant model in leaf

① regression

$I_m := \{ i \mid x^{(i)} \in R^m \}$

split up into m regions $R_1, R_2, \dots, R_m$

each region has output $O_1, O_2 \dots, O_m$ → classifier

↳ O_m = average$_{i \in I_m}$ $y^{(i)}$ ← actual values.

↳ Error $E_m = \sum_{i \in I_m} (y^{(i)} - O_m)^2$

minimize $\sum_{i=1}^{m} E_i + \lambda m$ ← avoid many regions

algorithm: keep breaking regions into smaller regions until all are $\leq k$ in size. divide region along single line minimizing

$$\sum_{i \in I_1} (y^{(i)} - y_1^{avg})^2 + \sum_{i \in I_2} (y^{(i)} - y_2^{avg})^2.$$

if tree is too large, "weakest-link" pruning to remove the bottom-level split minimizing $C_\alpha(T) = \sum_{m=1}^{|T|} E_m(T) + \alpha |T|$

② classification

O_m = majority$_{i \in I_m}$ $y^{(i)}$ $\quad E_m = \#\{ i \mid i \in I_m \text{ and } y^{(i)} \neq O_m \}$

empirical probability $\hat{p}_{m,k} = 1 - \frac{E_m}{N_m}$ ← $|I_m|$

possible metrics

misclassification error

$Q_m(T) = \frac{E_m}{N_m}$

Gini index

$Q_m(T) = \sum_k \hat{p}_{m,k}(1 - \hat{p}_{m,k})$ ← over all classes k

Entropy

$Q_m(T) = H(I_m) = -\sum_k \hat{p}_{m,k} \log_2 \hat{p}_{m,k}$

↓

minimize by maximizing information gain

$H(I_m) - \left(\frac{|I_1|}{N_m} \cdot H(I_1) + \frac{|I_2|}{N_m} \cdot H(I_2) \right)$

↑ fixed.

Bootstrap Aggregating (Bagging)

make multiple trees w/small samples, decide as average of trees' outputs

random forests

B bootstrap samples, tree T_b for each sample by recursively choosing the best out of d split points, return majority vote of trees.

(10) Markov Decision Processes

$\langle S, A, T, R, \gamma \rangle$

(S: states, A: actions)

$T : S \times A \times S \to \mathbb{R}$

$T(s, a, s') = \mathbb{P}(S_t = s' \mid S_{t-1} = s, A_{t-1} = a)$

$R : S \times A \to \mathbb{R}$ reward for taking action a at state S.

$\gamma \in [0, 1]$ = discount factor.

policy $\pi : S \to A$ specifies which action to take in each state

finite horizon value: $V_\pi^h(s) = R(s, \pi(s)) + \gamma \sum_{s'} T(s, \pi(s), s') V_\pi^{h-1}(s')$

$\pi_1 >_h \pi_2$ if $\forall s \in S$, $V_{\pi_1}^h(s) \geq V_{\pi_2}^h(s)$ and $\exists s \in S$ s.t. $V_{\pi_1}^h(s) > V_{\pi_2}^h(s)$.

"$>_h$" ↑ "better"

infinite horizon value: use $\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_t \mid \pi, s_0\right] = \mathbb{E}[R_0 + \gamma R_1 + \cdots \mid \pi, s_0]$

↑ random variable representing reward across all possible stochastic transitions at time t

Bellman Equation

$$V_\pi(s) = \mathbb{E}[R_0 \mid \pi, S_0 = s] + \gamma \mathbb{E}[R_1 + \gamma(R_2 + \gamma(R_3 + \cdots)) \mid \pi, S_0 = s]$$

$$= R(s, \pi(s)) + \gamma \sum_{s'} T(s, \pi(s), s') V_\pi(s')$$

finite policy finding: Q = "expected" value

$Q^0(s, a) = 0$

$Q^1(s, a) = R(s, a) + 0$

$Q^2(s, a) = R(s, a) + \gamma \sum_{s'} T(s, a, s') \max_{a'} Q^1(s', a')$

$\vdots$

$Q^h(s, a) = R(s, a) + \gamma \sum_{s'} T(s, a, s') \max_{a'} Q^{h-1}(s', a')$

$\pi_h^*(s, a) = \operatorname{argmax}_a Q^h(s, a)$

infinite policy finding:

$\exists$ optimal policy π^* s.t. $V_{\pi^*}(s) \geq V_\pi(s) \quad \forall s, \pi$.

$Q^*(s,a) = R(s,a) + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q^*(s',a') \qquad \pi^*(s,a) = \arg\max_a Q^*(s,a)$

↑ expected infinite-horizon discounted value of being in state s, action a, and doing π onwards

↳ update starting w/ initial $Q(s,a) = 0$

↳ return Q when $|Q^{new}(s,a) - Q^{old}(s,a)| < \varepsilon \quad \forall s \in S, a \in A$

result is that we get $\| V_{\pi_{Q_{new}}} - V_{\pi^*} \| < \varepsilon$. Also $\exists$ some $\varepsilon > 0$ s.t. $\| Q_{old} - Q_{new} \| < \varepsilon \Rightarrow \pi_{Q_{new}} = \pi^*$

(11) Reinforcement Learning

"Learning" an MDP without knowing R, T.

model-free method (Q-learning)

initialize all $Q(s,a) = 0$.

given (s_{old}, a, s_{new}, r), [pick a based on current $\max_{a'} Q(s,a')$]

$$Q_{new}(s_{old}, a) = (1-\alpha) Q_{old}(s_{old}, a) + \alpha [r + \gamma \max_{a'} Q(s_{new}, a')]$$

to address exploitation vs. exploration,

we can also pick $\arg\max_{a' \in A} Q(s,a')$ with $1-\varepsilon$, random action with ε.

deep Q-learning: minimize loss $\left(Q(s,a) - (r + \gamma \max_{a'} Q(s',a'))\right)^2$

using a neural network

↑ can lead to catastrophic forgetting ← can handle using replay buffer

fitted Q-learning: alternate between running Q-function's policy and training neural network to get Q

policy gradient: define some form of differentiable that can be representative of $Pr[\text{action } a \mid \text{state } s]$, then gradient descent

model-based RL

Solve the following MDP.

we know $R(s,a)$ when we experience it

set $\hat{T}(s,a,s') = \frac{\#(s,a,s') + 1}{\#(s,a) + |S|}$ → starts at $\frac{1}{|S|}$

(12) Unsupervised Learning

- no labels to learn.

clustering — given points, how to categorize in k groups ← hyperparam.

$$\min\left(\text{loss} = \sum_{j=1}^{k} \sum_{i=1}^{n} \mathbb{1}(y^{(i)}=j) \left\| x^{(i)} - \mu^{(j)} \right\|^2 \right)$$

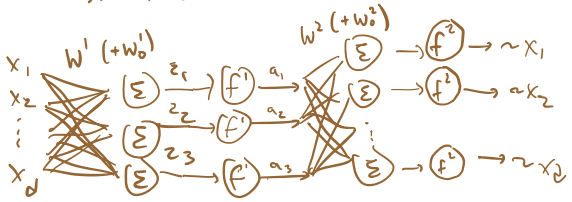
($\mu^{(j)}$: centroid of cluster)

potential metrics for evaluation
- consistency
- ground truth (if it exists)
- visualization
- interpretability

k-means: classify each point, by nearest neighbor
then centroids of each new cluster,
repeat

initial centroids / k both important choices.

autoencoders — encode vectors into shorter dimension (k), then decode (e.g. with linear/ReLU layers)



want small k

can minimize squared loss

$$\min_{W^1, W_0^1, W^2, W_0^2} \sum_{i=1}^{n} \left\| h(g(x^{(i)}; W^1, W_0^1); W^2, W_0^2) - x^{(i)} \right\|^2$$