CNN

idea: treat convolutional matrix as params (weights) to be learnt.

W_{in} = input width, K_W = kernel width, P = padding, S = stride → hyperparameters.

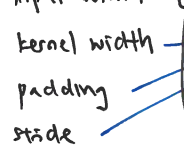
$W_{out} = \left\lfloor \frac{W_{in} - K_W + 2P}{S} \right\rfloor + 1$

parameter sharing (using same set of weights) → detect same features at diff positions.

for r < Hout:
  for c < Wout:
    for i,j < K:
      Y[r,c] += X[r+i, c+j] * F[i,j]

RNN

$x_t \in \mathbb{R}^3$, $h_t \in \mathbb{R}^4$, y_t & $\hat{y}_t \in \mathbb{R}^2$, $W_{hx} \in \mathbb{R}^{4\times 3}$, $W_{hy} \in \mathbb{R}^{2\times 4}$, $W_{hh} \in \mathbb{R}^{4\times 4}$



$z_t = W_{hh} h_{t-1} + W_{hx} x_t$ (only looks at prev hidden layer.)

$h_t = \psi(z_t)$, ψ identity

$o_t = W_{hy} h_t$, $\hat{y}_t = \text{soft}(o_t)$

$J_t = -\sum_{i=1}^{2} y_{t,i} \log(\hat{y}_{t,i})$, $J = J_1 + J_2 + J_3$

ancestral sampling: randomly pick edge w prob.

$\frac{\partial J}{\partial o_t} = \hat{y}_t - y_t$ $\left(\frac{\partial \hat{y}_i}{\partial o_t} = \hat{y}_i(\mathbb{I}[t=i] - \hat{y}_t),\ \frac{\partial J}{\partial \hat{y}_i} = -\frac{y_i}{\hat{y}_i} \Rightarrow \frac{\partial J}{\partial o_t} = \sum_i \frac{\partial J}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial o_t}\right)$

greedy decoding: linear in max path len, select edge w lower neg log prob.

$\frac{\partial J}{\partial h_i} = \sum_{t=i}^{3} (W_{hh}^T)^{t-i} W_{hy}^T \frac{\partial J}{\partial o_t}$, $\frac{\partial J}{\partial W_{hh}} = \sum_{i=1}^{3} \frac{\partial J}{\partial h_i}\frac{\partial h_i}{\partial W_{hh}}$ → $h_{i-1} + \sum_{j=1}^{i-1} W_{hh}^j h_{i-1-j}$

→ seq info, incorp info from prev step to current & previous inputs of var length. → h_t

RNN LM: convert all prev words to fixed len vector & def dist. $p(w_t | f_\theta(w_{t-1}, \dots, w_1))$ (structures designed to)

↳ $p(w_1, \dots, w_T) = p(w_1|h_1) \cdots p(w_T|h_T)$ → choose next word according to trained to max prob of seq. of tokens. predict next token in seq.

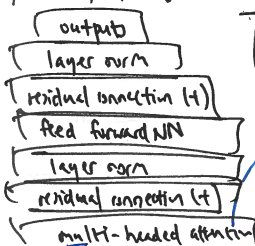
Transformers

problem: ↑ network depth, ↑ internal cov. shift (small change in low layers → amplify → higher)

changes in dist of network activation due to update in network params during training → can make training process slower/less stable bc each layer must continuously adapt to new dist.

fix: layer normalization. $a \to [b = f(a)] \to b$ to $a \to [b' = f(a)] \to [b = b' + a]$

problem: perf. degradation → residual connections.



transformer layer

looks back at hidden vectors of curr & prev timestep.

$S_{t,i} = \frac{K_i^T q_t}{\sqrt{d_k}}$

attention: query = XW_q, key = XW_k, value = XW_v

score $S = QK^T/\sqrt{d_k}$, attention $A = \text{softmax}(S)$

output $X' = AV$. applied row-wise

↳ $x'_t = \sum_{j=1}^{4} a_{t,j} v_j$ & if x_i, x_j swap, $a_{t,i}, a_{t,j}$ swap too

+ tokenization * cache keys & values.

batching: padding and truncation. discard queries/similarity score/attn weights

masking e.g. $\begin{pmatrix} 0 & -\infty & -\infty \\ 0 & 0 & -\infty \\ \dots & & \end{pmatrix}$ to maintain causality (prevent access of future tokens). not to enhance them. $v_j = W_v^T x_j$

aspects of image ⇒ enhance ability to understand complex dependencies (in texts like NLP) → each head pay attention to distinct input → model captures positional info better

indiv. losses.

training transformer LM: 1 sentence = 1 training ex. $J = \log p(\vec{w}) = \sum_i \log p(w_i|h_i)$

Kernel: a matrix/tensor of weights that is slid over image tensor. at each position (dictated by hyperparams such as padding/stride/kernel size), perform element-wise mult. of kernel & underlying part of img. tensor, sum resulting entries to obtain output val.

stride defines how many pixels the kernel moves @ each step as it passes over the rows/cols of image tensor. ↑ val of stride reduce computational power but lose more info (limit upside of models accuracy)

↓ stride preserves size of input img & can id more fine grained features, but ↑ computational cost.

padding surrounds image tensor w rows/cols of zero, by adding appropriate amt, we can ensure output shape = input shape and allow every pixel to be included in convolution. also, padding helps filters focus on corner just as much as middle (pass over corner multi times not just once).

downsampling reduces output dimensionality (e.g. mean/max pooling), appropriate for larger images where we want to reduce time/multiyield overfitting by making input to future convolutions less complex.

upsampling increase output dimensionality when we want output ≠ of conv to be larger (e.g. to match input img in dim) (e.g. assign label to each pixel of output & custom to input).

RL

MDP = $(S, A, T, R, \gamma, s_0)$, $T: S\times A \times S \to [0,1]$, $R: S\times A\times S \to \mathbb{R}$. ($p(s'|s,a)$ assumes s' only determined by s & a)

$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \dots$, $r_t = R(s_t, a_t, s_{t+1})$, then discounted payoff = $r_0 + \gamma r_1 + \gamma^2 r_2 + \dots$

policy $\pi: S \to A$, optimal value function $V^*(s) = \max_{a\in A} \sum_{s'\in S} p(s'|s,a)(R(s,a,s') + \gamma V^*(s'))$ → M

expected discounted future value if taking action a in state s s.t. $V^*(s) = \max_{a\in A} Q^*(s,a)$.

- exploration: take action to visit (state, action) pairs prev. unseen. → only no better than random
- exploitation: take action to visit (s,a) pairs known to have high reward. → only never discover optimal.

learning $V^\pi(s)$: fixed pt. iteration on Bellman eq. $V^\pi(s) = \mathbb{E}[r + \gamma V^\pi(s')]$. $V^\pi_{\phi_{t+1}}(s) \approx r + \gamma V^\pi_{\phi_t}(s')$.

similarly, $Q^\pi(s_t, a_t) \approx r_t + \gamma Q^\pi(s_{t+1}, a_{t+1})$, then switch to $\pi_{gr}(s) = \arg\max_a Q^\pi(s,a)$] SARSA.

↳ good if Q-vals in lookup table (plicy improvement guaranteed to ↑ reward). complexity idpt of stochastic policy gradient method. ↳ too aggressive if NN used to estimate Q-val: find plicy params θ that maximizes $\mathbb{E}_{\pi_\theta}[r_1 + \dots + r_T]$ → value iteration (instead)

Policy Gradient Theorem: observe trajectory $(s_1^n, a_1^n, r_1^n, \dots, s_T^n, a_T^n, r_T^n)$ by following $\pi_\theta(a_t^n | s_t^n)$. $O(|A||S|^2 + |S|^3) \to$ policy itr. $O(|A||S|^2 + |S|) \to$ value itr.

depends on future traj.

$Q_t^n = \sum_{i=t}^{T} r_i^n$ (empirical total reward starting from step t trajectory.

$u_t^n = \frac{d}{d\theta} \ln \pi_\theta(a_t^n | s_t^n) \in \mathbb{R}^d$ (action score vector, from autodiff) depends on action

d = # params in policy, so $\theta \in \mathbb{R}^d$ (per iteration)

even if env/state is not/partially observable.

$g^n = \sum_{t=1}^{T} Q_t^n u_t^n \in \mathbb{R}^d$ gradient estimate ⇒ unbiased estimate of $\frac{d}{d\theta} J(\theta)$.

reinforce algo: $\theta^{n+1} \leftarrow \theta^n + \eta g^n$, g^n from PGT. (rather than stochastic estimate $\frac{d}{d\theta} J(\theta)\big|_{\theta^n}$).

↳ failure modes: (1) cancellation (low SNR): total return Q can scale w horizon, and vary a lot due to randomness in policy/env, overall grad can be small (terms w opp signs).

(2) getting stuck: when plicy get too close to boundary of simplex, store vectors for unlikely actions get large, pmf of vary then get small, in the limit, 0 or ∞.

Actor Critic: $g_{AC} = \sum_{t=1}^{T} Q_\phi^\pi(s_t, a_t) u_t$ → $\mathbb{E}[Q_t | s_t, a_t]$: conditional expectation to dec. variance. this is generic (using Q^π). usually not implementable.

↳ $\mathbb{E}[u_t | s_t] = 0$, $g = \sum_t (Q_t - V_\phi^\pi(s_t)) u_t$: baseline: no bias, but higher variance (compared to Q_ϕ^π of AC).

Advantage Actor Critic: $A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$, $g = \sum_t A_t \nabla_\theta \ln P(a_t|s_t)$ value estimate

critic's target becomes: n-step returns! ↑

* Q-learning [online learning/deterministic transition] $Q(s,a) \leftarrow r + \gamma \max_{a'} Q(s',a')$ → $s' \sim p(s|s,a)$

[non-deterministic/temporal] difference error update] $Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a'))$

↳ take greedy action w.p. ε, random action o.w. → greediness param. learning rate α (trust param)

n-step returns $G_t^{(n)} = \sum_{k=0}^{n-1} \gamma^k r_{t+k} + \gamma^n V_\phi(s_{t+n})$, $A(s_t, a_t) \approx G_t^{(n)} - V_\phi(s_t)$ note: $Q(s_t, a_t) = r_t + \gamma V(s_{t+1})$ (adv. based on actual obs. rewards)

↳ (1) ↑ n reduce bias, increase variance, (2) fast backprop (n steps instead of 1), (3) better grad quality

few shot learning: $D = [(x_i, y_i)]_{i=1}^N$ with small N & v. large pre-trained model. e.g. technical/specific terminologies (adapt to)

A. supervised fine-tuning: use standard supervised obj, backprop to compute grad + std gradient optimizer, req. model weights

B. feed training ex as prompt, allow LLM to infer patterns in training ex during inference // transformer LM req. $O(N^2)$ time/space, N = |context| req. ~3× time & memory as forward prop.

Q-learning: model-free/no req. of $p(s'|s,a)$ or $r(s,a)$, but only guaranteed to conv if every state visited ∞ times/ slow conv, req. exp. strategy.

value-iteration: iteratively update $V^\pi(s)$ (M) then derive optimal policy $\pi^* = \text{argmax}_a$

policy-iteration: update $V^\pi(s)$ & $\pi_{new}(s)$, conv faster than value iteration. ↳ guarantees conv to opt. plicy, req. full model

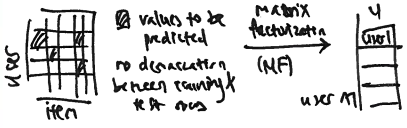
actor-critic: actor update π based on critic, critic estimates V, update rules use policy gradient. more sample efficient.

Recommender Systems

1. Content filtering: analyzes item attributes (features/metadata/keywords/category) & match them to users known preference, ex. Movie Genre proj. assesses access to side info ↗ doesn't work on new items w/o ratings
2. Collaborative filtering: analyzes user-item interaction patterns (ratings/clicks) to find similar users/items e.g. Netflix (incl. latent factor & neighborhood methods)

- Low-rank factorization: seek U, V $n \times k$ mats s.t. $\text{rank}(UV^T) \le k$.

if R has rank $> k$, cannot be represented exactly: minimize residual $E = R - UV^T$



each col is a latent factor U, V same # cols

$J_{ij}(U,V) = \frac{1}{2}(r_{ij} - u_i^T v_j)^2 + \frac{\lambda}{2}(\|u_i\|_2^2 + \|v_j\|_2^2)$

$\nabla_{u_i} J_{ij}(u_i, v) = -e_{ij} v_j + \lambda u_i$ (sim v_j)

(trick Cholesky) $\hat{r}_{ij} = 0$; $\cdot p_j + u_i \cdot 0$

$U = \begin{pmatrix} -u_1 - \\ \vdots \\ -u_m - \end{pmatrix}$, $V = \begin{pmatrix} | & & | \\ p_1 & \cdots & p_n \\ | & & | \end{pmatrix}$.

recommend based on proximity

if R fully obs. + no regul. optimal UV^T from both methods eg. $U = Q_k \Sigma_k$, $V = P_k$

unconstrained MF: [SGD] sample $(i,j) \sim Z$, compute $e_{ij} = r_{ij} - \vec{u}_i^T \vec{v}_j$, $\vec{u}_i \leftarrow \vec{u}_i - \eta \nabla_{\vec{u}_i} J_{ij}(\vec{u}_i, \vec{v})$, sim for $\vec{v}_j$

[Alternating Least Squares] $U = \text{argmin}_U \sum_{(i,j) \in \Omega} (r_{ij} - \vec{u}_i^T v_j)^2$ → $\frac{1}{2}\sum_{(i,j)\in\Omega}(r_{ij} - \vec{u}_i^T v_j)^2$ is convex, so can iterate for each of U, V, keeping other fixed

Singular Value Decomposition: $R = \underset{n\times n}{U} \underset{n\times d}{\Sigma} \underset{d\times d}{V^T}$, cols of U, V are orthonormal ($U^T U = I$), $R = \sum_{k=1}^{\text{rank}(R)} \sigma_k u_k \vec{v}_k^T$ → rank-1

→ sum to k: closest rank k matrix to R under (L2) Frobenius norm

constrained optimization [romance/no dislike] $U, V = \text{argmin}_{U,V} \frac{1}{2}\|R - UV^T\|_2^2$ ↗ train h_t on D_t, test on $D \setminus D_t$

Ensemble Methods

① bootstrap: resample D_t from D w/ replacement $|D|$ times, in expectation $\frac{1}{e}$ excluded

↳ returns accuracy est + avg. of sub accuracies, model ensemble. $\{h_t\}_{t=1}^T$, to predict for new x, return $\frac{1}{T}\sum_{t=1}^T h_t(\vec{x})$

T: hyperparam (↑, more ex, finer quantification of uncertainty) ↗ diversity of set of learned classifiers

bagging = bootstrap aggregating.

② column subsampling: use only subset of features for each classifier

↳ if base classifier DT, can resample on $\sqrt{\text{#features}}$ for every level, or each split — split feature randomization

diversity helps to cover posterior

↔ change hypothesis class for each bag.

random forest: bagging + col resampling + small DTs.] uses unweighted sum!

③ bagging for binary classification: $h_t : \mathbb{R}^d \to [-1, 1]$, vote if $\sum_t h_t(\vec{x}) \gtrless 0$ → voted classifier strictly more expressive than base. → classify above chance perf.

Boosting: if weak learner can get error $= \frac{1}{2} - \epsilon$, final boosted classifier can get 0 train & test on test (ϵ assumptions) → weighted vote $\sum_{t=1}^T \alpha_t h_t(\vec{x}) \gtrless 0$

A = col corresponds to h_t, mv to training ex $\vec{x}^{(i)}$, $\vec{a}^{(i)} = (h_t(\vec{x}^{(i)}))_{t=1}^{|A|}$, $\vec{a} \cdot \vec{a}$ → try to optimize loss that favors true margin, $y^{(i)} \vec{a}^{(i)} \alpha > 0 \ \forall i$

$\Delta\alpha = \frac{1}{2}\ln\frac{1-\epsilon}{\epsilon}$, $w_i \leftarrow w_i \cdot \begin{cases} e^{-\Delta\alpha}/Z & \text{if } y^{(i)} = h_t(\vec{x}^{(i)}) \\ e^{\Delta\alpha}/Z & \text{o/w} \end{cases}$ perceptron/SGD: $\alpha_t < 0, |\alpha|$ big. (weights $w_{t-1}^{(i)}$). training error

AdaBoost: initialize datapoint weights $w_0^{(i)} = \frac{1}{N}$, for $t = 1, \dots T$. train weak learner T by minimizing weighted (ϵ_t) $\epsilon_t = \sum_{i=1}^N w_{t-1}^{(i)} \mathbb{I}(y^{(i)} \neq h_t(\vec{x}^{(i)}))$, vote weight of $h_t = \alpha_t = \frac{1}{2}\ln\frac{1-\epsilon_t}{\epsilon_t}$, $w_t^{(i)} = \frac{w_{t-1}^{(i)}}{Z_t} e^{\mp\alpha_t}$, $\hat{y} = \text{sgn}\left(\sum_t \alpha_t h_t(\vec{x})\right)$

sps we fix M, train till all margins $\ge \gamma$, w.p. $1-\delta$, true error of final vote $\le$ training loss $+ O\left(\frac{A + \sqrt{\ln 1/\delta}}{\sqrt{N}}\right)$. where $\sum_t \alpha_t \le A$.

Autoencoders

$\min_{V,W} \sum_{i=1}^N \|VW\vec{x}^{(i)} - \vec{x}^{(i)}\|^2$. → solve for best q, keeping V fixed

OR: $\hat{X} = XWV^T$, collect latent vectors $u^{(i)} = W^T \vec{x}^{(i)}$, minimize $\sum_{j=1}^d (x_j^{(i)} - v_j^T u^{(i)})^2$

one training ex for each dim j of $x^{(i)}$

↳ v_j = feature vec for x_j, $x_j^{(i)}$: target output for ex. i, $u^{(i)}$: learnable regression weights.

minimum sol: $\vec{u}^{(i)} = W^T \vec{x}^{(i)}$, so to optimize linear autoencoder, find U, V, then find W as min $\|UV^T - X\|_F^2$ or $\sum(v_j^T u^{(i)} - x_j^{(i)})^2$ (collaborative filtering!)

if $k=1$ hidden dim, sol is $v_1 \in \mathbb{R}^d$ and $w_1 = \frac{v_1}{v_1^T v_1}$, to avoid scaling ambiguity, $\|v_1\| = 1$, $w_1 = v_1$

Principal Component Analysis

$v_1 = \text{argmin}_v \frac{1}{N}\sum_{i=1}^N \|x^{(i)} - (\vec{v}^T \vec{x}^{(i)})\vec{v}\|^2$ s.t. $\|\vec{v}\| = 1$

construct residuals $e_1^{(i)} = x^{(i)} - (v_1^T x^{(i)}) v_1$ ↗ in long dividing → equiv: maximizes $\|$ proj the variance

1. the 1st principal component is the unit vector that minimizes mean-sq reconstruction error
2. 2nd PC is unit vector that minimizes mean-sq rec. error $\&$ residuals while remaining orthogonal to v_1 $\max \frac{1}{N}\sum_i (v_1^T x^{(i)})^2$

$v_2 = \text{argmin}_v \frac{1}{N}\sum_{i=1}^N \|e_1^{(i)} - (v^T e_1^{(i)})v\|^2$ s.t. $\|v\| = 1$, $\vec{v}^T v_1 = 0$, $e_2^{(i)} = e_1^{(i)} - (v_2^T e_1^{(i)}) v_2$ ↳ kth PC.. orth... $v_1 \cdots v_{k-1}$

jth principal value = variance $\&$ proj. into jth PC, bc PCs orthog, sum $\&$ PV = var. $\&$ $X^{(i)}$

↳ frac of var explained by PC $= \frac{\frac{1}{N}\sum_i (v_j^T x^{(i)})^2}{\frac{1}{N}\sum_i (x^{(i)})^T x^{(i)}} = \frac{\text{var of proj}}{\text{var of } x^{(i)}} = \frac{v_j^T \Sigma v_j}{\text{cov matrix of } x^{(i)}}$, where $\Sigma = \frac{1}{N}\sum_i x^{(i)}(x^{(i)})^T$

diagonal: var of feature $x_j^{(i)}$

off-diag: cov $\&$ pair of feature.

eigenvalues $\lambda = \max_{\|\vec{v}\|=1} \vec{v}^T \Sigma \vec{v}$ is largest eigenval $\&$ Σ. (max over $v^T v_1 = 0$ for second.)

k-Means

vector-quantization: map pt to closest center in Voronoi diagram

→ discrete autoencoder $X \xrightarrow{VQ} \text{discrete hidden } Z \xrightarrow{V} \text{pred} = \text{center corr to } z$ ↗ implements clustering

train to minimize $MSE = \frac{1}{N}\sum_{i=1}^N \|x^{(i)} - \hat{x}^{(i)}\|^2$, $\hat{x} = v_z = Vx$ $VQ(x, V)$, $\hat{X} = ZV^T$, $V \in \mathbb{R}^{d \times k}$, $Z \in \{0,1\}^{N \times k}$ rows 1-hot.

[blocks coord descent] given feature vectors $D = \{x^{(1)}, \dots, x^{(N)}\}$, initialize matrix $\&$ centers V (each col v_j)

repeat: minimize wrt Z: $\forall i$, set $z^{(i)}$ to map $x^{(i)}$ to closest center, then minimize wrt V, $\forall j$ min MSE thm v_j

($\text{argmin}_{V,Z} \sum_{i=1}^N \|x^{(i)} - Vz^{(i)}\|^2$) $v_j = \text{mean}\{x^{(i)} | z^{(i)} k e_j\}$ ⇔ to its assigned pts equal sized Gaussians

$z = \underset{z \in \{e_1, \dots, e_k\}}{\text{argmin}} \|x - Vz\|_2^2$ → 1-hot. V = matrix $\&$ centers.

Initialization: (1) random, (2) furthest pt. heuristic, (3) k-means ++ ↗ Pr[each initial center in diff Gaussian]

1. may get stuck in poor local minima (if we did not pick one center per Gaussian = k-means output bad soln) $\propto \frac{k!}{k^k} \propto \frac{1}{e^k}$
2. pick first cluster pt c_1 randomly, pick each subseq c_j s.t. it is as far as possible from $c_1, \dots, c_{j-1}$ ok if data Gaussian but: outliers = problems.
3. let $D(x)$ be distance between pt x & nearest center, choose next center w.p. proptn $D(x)^2$. (1^{st} unif. random)

↳ THM: attains $O(\log k)$ approx to optimal k-means sol. in expectation. * kNN supervised, k-means unsup

PCA: factorize $X \in \mathbb{R}^{n \times m}$ as $Z^T U \approx X$, $Z \in \mathbb{R}^{n \times m}$, $U \in \mathbb{R}^{n \times m}$, rows $\&$ X contains dpts; rows $\&$ U the PC/prototypes $\hat{X} = Z^T U$.

- reducing num from U/Z takes variance associated w PC. ⇒ incr. recons. error (var minimized)
- rows of U eigenvectors → $UU^T = I$. (bc PCs unit vectors)
- goal of PCA to produce underlying structure to data that preserves largest amount of variance (output variable never provided)