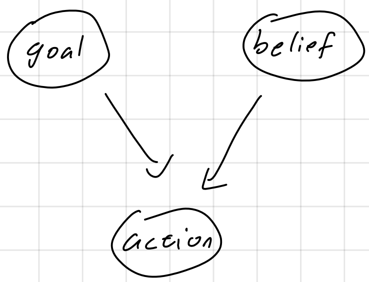
limitation in LLMs.

- language reasoning. → Human: internal world model
- Embodied reasoning. → Human: strategic planning → · internal model to predict states.
  · simulation of alternative plans.
  · access outcomes to refine /pick the best.
- social reasoning. (understand human)

↓

Theory of mind: reasoning above hidden mental variables



Human conduct model-based reasoning based on world models and agents.

World Model in Humans

- Perceiving physical properties
- Predicting dynamics
- Model-based control/planning. → human tool use: can learn tool through a few trials.

world model as state transistion probability.

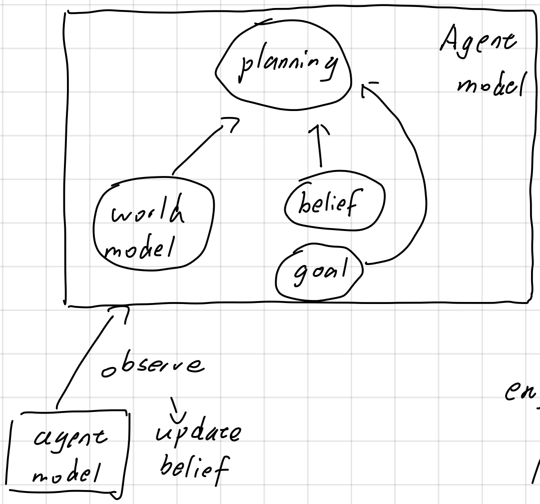
causal relationship between action and state change.

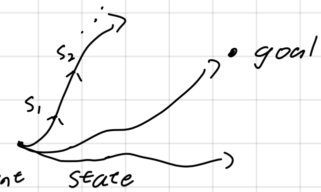
$$P(s' \mid s, a)$$

next state, current state, action.

Agent model in human:

- strength
- goal
- relationship
- moral judgment
- beliefs.





(simulate plan via world model).

ex) Alpha Go

MCTS with a known world model.

level-1 agent model:

- agent as a observer for observing level-0 agent

model-based theory of mind:

$$P(\text{mind} \mid \text{state}, \text{actions}) \propto \underbrace{p(\text{actions} \mid \text{state}, \text{mind})}_{\text{level-0 agent}} P(\text{mind})$$

understand the mind through state, action

Behavior Prediction

$$P(\text{future action} \mid \text{state}, \text{mind}).$$

Human-Ai interaction:

$$\pi(\text{action}_{AI} \mid \text{state}, \text{mind}_{AI}, \text{mind}_{human}).$$

Recursive social reasoning:

level-2 agent

↓ observe

level-1 agent

⋮

Experience:

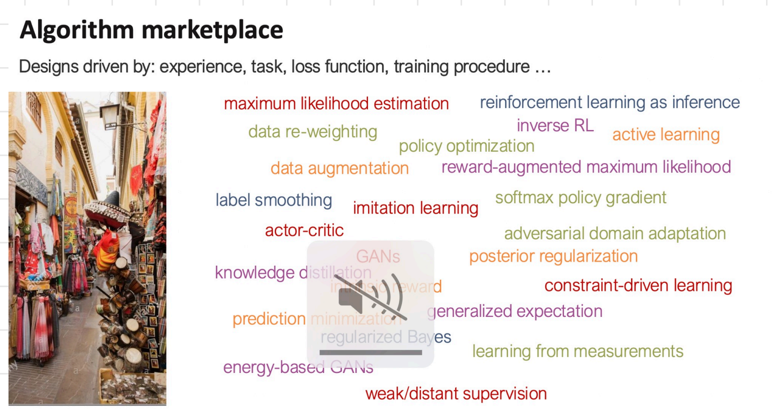
- Data example

1. language reasoning
   - math, logic, ...
2. space/time
   - vision, audio, sensors, ...
3. multi-agency
   - adversary, collaboration ...
4. law of nature
   - biology, ...

- Rules / Constraints
- Knowledge graphs
- Rewards
- Auxiliary agents
- Adversaries
- Master classes.
- ...

problems with few label:

- Privacy, security
- Expensive to collect/annotate
  - controllable content generation.
- Difficult/expertise-demanding to annotate.
- Specific domain



ML Solution:

loss experience.

$$\min \mathcal{L}(\theta, \mathcal{E})$$

optimization solver model architecture

unified, standardized ML principles

A "standardized formalism" of ML

$$\min_{q,\theta} -H + D - E$$

Uncertainty Divergence Experience

LLM

NLP before 2017:

Automated understanding and generation of natural language

- Name Entity Recognition.

  ex) $\underbrace{\text{Adam driver}}_{\text{person}}$ was born in $\underbrace{\text{San Diego.}}_{\text{city.}}$

- Sentiment Analysis.

Since 2017.

Every year, model size increases by 10x.

- google Bert → understanding
- Open AI GPT → generative
- GPT-4 → mixture of experts model.

Language Model:

next word prediction given context.

$P(w_i \mid w_1, \cdots, w_{i-1})$.

↓

probabilty dirstribution

↓

sample from distribution → (sampling decoding).
→ (greedy decoding).
→ (top-k decoding).

↓

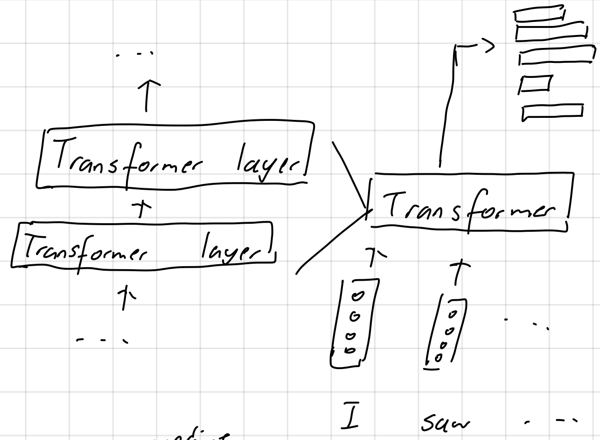
generate.

Implementation → Transformer

↓

Attention mechanisms. ex)

$p(* \mid I \text{ saw a cat on a})$



Training of ML :

Self-supervised learning :

- predict the future from the past $\longrightarrow$ predict
- predict the future from the recent past $\longrightarrow$ predict
- predict the past from the present. predict $\longleftarrow$
- predict the top from the bottom.

(predict any part of the past, present or future percepts from whatever information is available).

Motivation: • successfully learning to predict everything from everything else would result in the accumulation of lots of background knowledge about how the world works.

• Mass amount of data.

SSL in Language Model.

- sentence :

$$y = (y_1, y_2, \cdots, y_T)$$

$$P_\theta(y) = \prod_{t=1}^{T} P_\theta(y_t \mid y_{1:t-1}).$$

Training :

given data example y^*,

minimizes negative log-likelihood of the data.

$$\min_\theta L_{mle} = -\log p_\theta(y^*) = -\prod_{t=1}^{T} P_\theta(y^* \mid y^*_{1:t-1}).$$

- A transformer-based LM with 125M to 175B parameters

- Trained on massive text data.

Word Embeding:

- Conventional word embeding
  - Word2vec, Glove
  - A pre-trained matrix, each row is an embedding vector

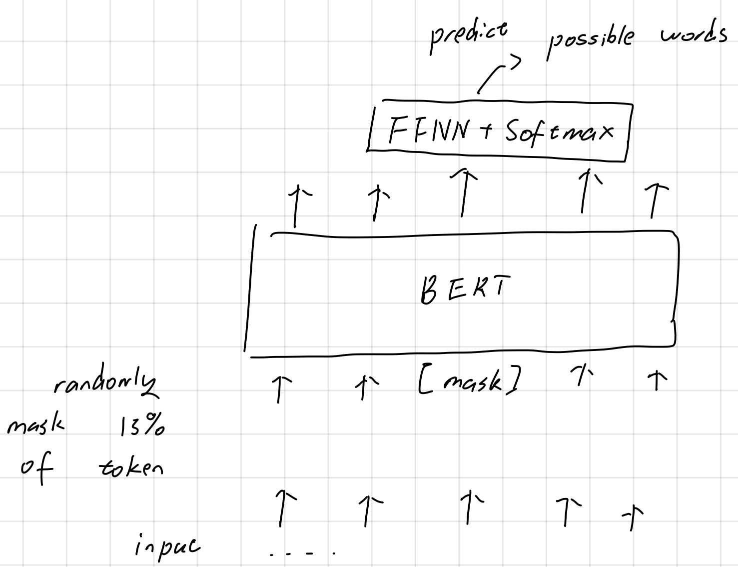
corpus → $\boxed{\text{Word2vec}}$ → embedding matrix

- Problem: word embedding are applied in a context free maner
- Solution: train contextual representations on text corpus.

↓

BERT: a bidirectional model to extract contextual word embedding

- Training procedure
  - Masked language model (masked LM).
    - masks some percent of words from the input and has to reconstruct those words from context.



(use the output of the masked word's position to predict the masked word).

More solutions:

80% time mask

10% time random word

10% time keep same

- Two sentence task:
  - To understand relationship between sentences:

- Predict likelihood that sentence B belongs after sentence A.

- BERT: Downstream Fine-tuning:
  - BERT for sentence classification.

Masked Autoencoder (MAE).

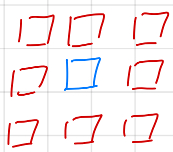
Masked images → Encoder → decoder → Target

↑ small subset of visible patches (Encoder)

↑ after pre-training, discard Decoder and use encoder for patten recognition.

Relative Positioning:

Train network to predict relative position of two regions in the same image.



← 8 possible location.

↑ classifier

↑ CNN ↑ CNN

pre-train CNN using self-sceppervision

1. Input image
2. Extract region proposals
3. Compute CNN features.

- Colorization:

  Train network to predict pixel colour from a monochrome input

SSL from videos:

- Video Sequence Order
  - Sequential Verification
- Video Direction
  - Predict if video playing forwards or backwards.

- Video Tracking:
  - Given a color video, colorize all frame of a gray scale version using a reference frame.

Enhancing LLM

limitation:
- Lack World and Agent knowledge → Need richer learning mechanism
- Inefficiency of the language modality ⟶ Need multi-modal capabilities

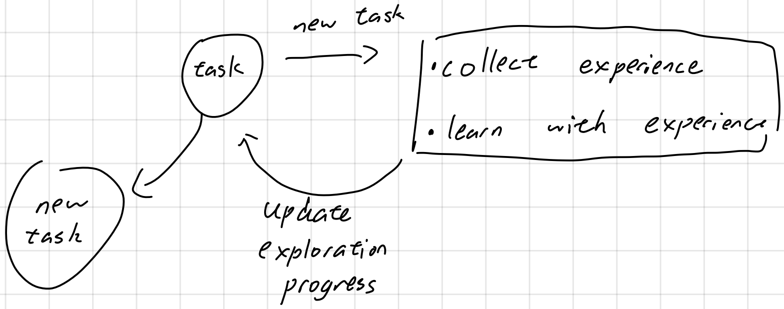
- Richer learning mechanisms
  - learning with embodied experience
    - Where to get
    - How to get
    - How to learn

- Embodied simulators
  - Household activities
  - Touchdown
  - Minecraft
  - OS
  - Simulated websites

- How to get:
  - Goal-oriented
    - Collecting experiences by completing a given task

MCTS $\xrightarrow[\text{into training data}]{\text{convert experience}}$ Question & Answer (planning)

- Auto curriculum
  - proposing new tasks automatically by prompting GPT-4 to generate new tasks.



- Random Exploration:

- Finetuning LMs with the experiences
- But also want to preserve the original language capabilities of LMs.
  - instead of overfitting
  - Continual learning with EWC (Elastic Weight Consolidation).

$$F_{i,i} = \frac{1}{n} \sum_{i=1}^{N} \left(\frac{\partial L_u^{(i)}}{\partial \theta^*_{u,i}} \right)^2$$

$$L(\theta) = L_V(\theta) + \lambda \sum_i F_{i,i} (\theta_i - \theta^*_{u,i})^2.$$

Conventional finetuning objective

regularizer to preserve important weights.

- Updating external memory
  - instead of changing LM parameters

Multi-modal Capabilities

existing limitation

- can understand images / cannot generate images
- can do interleaved generation of image and text
  - however generated images are not consistent → Lack internal world model.

- Video diffusion model → generates video given actions
  - predict future frame given previous frame and action.

    $P(s_t \mid s_{t-1}, a_{t-1})$.

- Text-to-Video generation.
  - ex) Sora

⇒ Need a more general world model.

- integrate different spaces
- generalist capability
- real time control.

Latent-space Reasoning

↓

which fuses information of different observed modalities.

- Multi-level latent space
  - immediate next moves
  - mid-term and long-term planning and thought experiments.

↓

How to learn a good latent space?

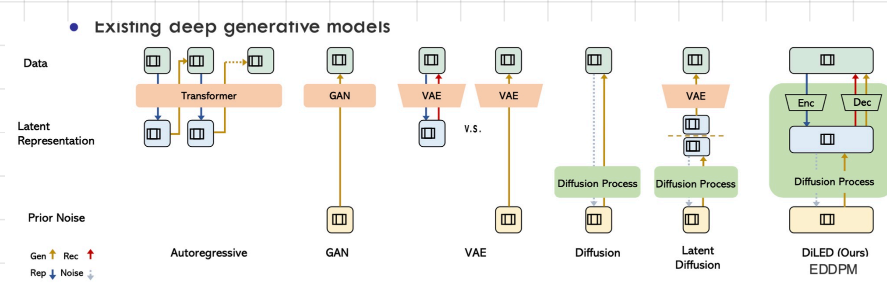
- Compact and well-structed representation of the world, realistic generation.

  consistent reconstruction.

Examples of latent-space

- Autoregressive
- GAN
- VAE
- Diffusion
- Latent diffusion.
- DiLED
  (EDDPM).

→



- No Free Lunch Theorem:
  - No single model is universally best-performing algorithm for all problems.
  - All algorithms perform equally well when their performance is average across all problems.

Unsupervised Learning

probability:

sample space: space of all possible outcomes

- $p(x, y)$ joint probability

- $p(y|x) = \dfrac{p(x, y)}{p(x)}$

- $E[f(x)] = \sum_x f(x)p(x)$ or $= \int_x f(x)p(x)\,dx$

- $p(x) = \sum_y p(x,y)$

- $p(x_1) = \sum_{x_2} \cdots \sum_{x_N} p(x_1, \cdots x_N)$.

- $p(x,y) = p(y|x)p(x)$
- $p(x_1, \cdots, x_N) = p(x_1)\, p(x_2|x_1) \cdots p(x_N|x_1, \cdots x_{N-1})$.

- Bayes rule:

$$p(\theta|D) = \frac{p(D|\theta)\,p(\theta)}{p(D)}$$

likelihood

prior belief

posterior belief

- $p(x, y) = p(x)p(y)$
- $p(x,y|z) = p(x|z)\,p(y|z)$

- Gaussian Distribution

- Multinomial Distribution

Entropy :

Shannon entropy $H(p) = -\sum_x p(x) \log p(x)$

average level of "information"

KL Divergence : measure closeness of two distributions $p(x)$ and $q(x)$

$$KL(q(x) \| p(x)) = \sum_x q(x) \log \frac{q(x)}{p(x)}$$

also the relative entropy.

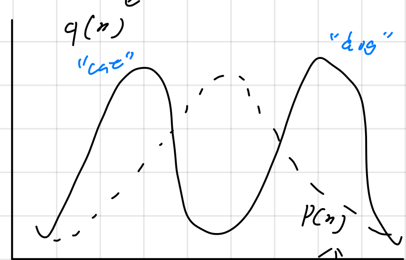
- $KL \geq 0$
- Not a true "distance"
  - Not commutative $KL(p\|q) \neq KL(q\|p)$.
  - Not satisfy triangle inequality

q high, p high $\Rightarrow$ low KL

q high, p low $\Rightarrow$ high KL.

If q is low, low KL regardless of p.

fixing this distribution.



see how close $p(x)$ is to $q(x)$

min KL is to approach $p(x)$ to $q(x)$.

- Supervised learning :

learn $p_\theta(x)$

observe full data.

MLE : $\min_\theta -E_{x \sim \tilde{p}(x)}[\log p_\theta(x)]$

$\downarrow$

MLE is minimizing KL between data and model distribution.

$$KL(\tilde{p}(x) \| p_\theta(x)) = -E_{\tilde{p}(x)}[\log p_\theta(x)] + H(\tilde{p}(x))$$

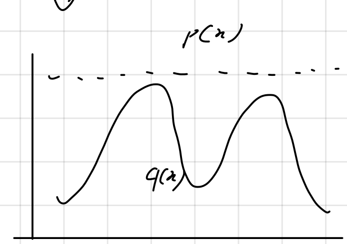
$$= \sum_x \tilde{p}(x) \log \frac{\tilde{p}(x)}{p_\theta(x)}$$

$$= \sum_x \tilde{p}(x) \log \tilde{p}(x) - \sum_x \tilde{p}(x) \log p_\theta(x)$$

$$= H(\tilde{p}(x)) \quad \underbrace{- E_{\tilde{p}(x)}[\log p_\theta(x)]}_{\text{cross entropy}}$$

$KL(p(x) \| q(x))$

$\Downarrow$



VAE mode seeking

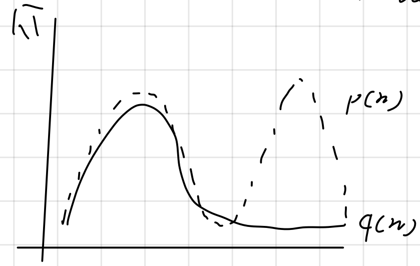
Unsupervised Learning

- observed variables x
- latent variables z.

want to learn $p_\theta(x, z)$.

GAN $KL(q(x) \| p(x))$

mode - missing



latent variable :

- Discrete latent variable can be used for partition / cluster data into subgroups.
- Continuous latent variable can used for

ex) Gaussian Mixture Models.

a mixture of K Gaussian components

$$p(x_n | \mu, \Sigma) = \sum_k \underbrace{\pi_k}_{\text{mixture proportion}} \underbrace{N(x_n | \mu_k, \Sigma_k)}_{\text{mixture component}}.$$

- z as latent class indicator

$$p(z_n) = multi(z_n ; \pi) = \prod_k (\pi_k)^{z_n^k}$$

- X as conditional gaussian variable with class specific mean/covariance

$$p(x_n | z_n^k = 1, \mu, \Sigma)$$

$$= \frac{1}{(2\pi)^{m/2} |\Sigma_k|^{\frac{1}{2}}} \exp\{ \cdots \}$$

- likelihood

taking the expectation.

mixture proportion , weight of gaussian distribution

$$p(x_n | \mu ; \Sigma) = \sum_k p(z^k = 1 | \pi)\, p(x_n | z^k = 1, \mu, \Sigma)$$

$$= \sum_k \pi_k N(x | \mu_k, \Sigma_k).$$

given different gaussian distribution

how data fits in

MLE: $\ell(\theta; D)$ ↙ observed data (supervised).

$$= \log \prod_n p(z_n, x_n)$$

$$= \log \prod_n p(z_n \mid \pi)\, p(x_n \mid z_n; \mu, \sigma)$$

$$= \sum_n \log \prod_k \pi_k^{z_n^k} + \sum_n \log \prod_k N(x_n; \mu_k, \sigma)^{z_n^k}$$

$$= \sum_n \sum_k z_n^k \log \pi_k - \sum_n \sum_k z_n^k \frac{1}{2\sigma^2} (x_n - \mu_k)^2 + C$$

to find unknown parameter

$\hat{\pi}_k$

$\hat{\mu}_k$

$\hat{\sigma}_k$

In unobserved data case:

Incomplete (or marginal) log likelihood:

with z unobserved, our objective becomes log of a marginal probability:

$$\ell(\theta, x) = \log p(x \mid \theta) = \log \sum_z p(x, z \mid \theta)$$

hard to learn

marginalize out z

$$\frac{\partial p(x_n \mid \mu, \Sigma, \pi)}{\partial \pi} = \frac{\partial \frac{p(x_n \mid \mu, \Sigma)}{\partial \pi}}{p(x_n \mid \mu, \Sigma, \pi)}$$

θ_z

GMM: $\log p(x_n \mid \mu, \Sigma) = \log \sum_k p(z^k = 1 \mid \pi)\, p(x_n \mid z^k = 1, \mu, \Sigma)$.

Complete log likelihood of "known latent" model

$p(A,B) = p(A)\, p(B \mid A)$

$$\ell_c(\theta; x, z) = \log p(x, z \mid \theta) = \log p(z \mid \theta)\, p(x \mid z, \theta)$$

$$= \log p(z \mid \theta_z) + \log p(x \mid z, \theta_x).$$

GMM: $\log p(x_n, z_n \mid \mu, \Sigma) = \log p(z_n \mid \pi)\, p(x_n \mid z_n, \mu\, \Sigma)$

Expectation Maximization (EM)

Intuition:

supervised MLE is easy : $\max_\theta \ell_c(\theta; x, z) = \log p(x, z|\theta)$

unsupervised MLE is hard : $\max_\theta \ell(\theta; x) = \log p(x|\theta) = \log \sum_z p(x, z|\theta)$

EM.

supervised MLE

M-step: $\max_\theta E_{q(z|x)} [\log p(x, z|\theta)]$ pretend we also observe z.

distribution that we "imagine"

E-step: $q(z|x) = p(z|x, \theta)$ can't observe q, estimate it.

using the current parameter, estimate the q distribution.

posterior distribution.

EM:

(optimize q distribution)

E-step: $q^{t+1}(z|x) = p(z|x, \theta^t)$

· similar idea GAN $\begin{cases} \text{generator} \\ \text{discriminator.} \end{cases}$

iterative process

M-step: $\max_\theta E_{q^{t+1}(z|x)} [\log p(x, z|\theta)]$

(optimize model parameter)

(coordinated descent)

Formally,

· For any distribution $q(z|x)$, define $\underbrace{\text{expected complete log likelihood}}$.

marginal

$$E_q[\ell_c(\theta; x, z)] = \sum_z q(z|x) \underbrace{\log p(x, z|\theta)}_{\text{supervised}} = E_{q(z|x)}[\log p(x, z|\theta)]$$

· a deterministic function of θ

· inherit the factorizability of $\ell_c(\theta; x, z)$

· use as the surrogate objective

Maximizing this yields a maximizer of likelihood?

$$\underbrace{\ell(\theta; x)}_{\substack{\text{marginal} \\ \text{unsupervised}}} \geq \underbrace{E_q[\ell_c(\theta; x, z)] + H(q)}$$

EM optimize this lower bound

which in turns optimize this

proof of inequality:

$$\ell(\theta;x) \geq E_q[\ell_c(\theta;x,z)] + H(q).$$

Note:

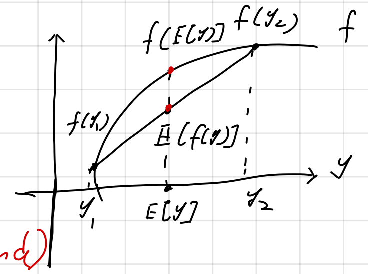
Jensen's inequality

convex:

$$E_{p(y)}[f(y)] \geq f(E_{p(y)}[y]).$$

concave:

$$E_{p(y)}[f(y)] \leq f(E_{p(y)}[y]).$$



$$\ell(\theta;x) = \log p(x|\theta)$$

$$= \log \sum_z P(x|z,\theta)$$

$$= \log \sum_z q(z|x) \frac{p(x,z|\theta)}{q(z|x)}$$

log function concave

$$= \log E_{q(z|x)}\left[\frac{p(x,z|\theta)}{q(z|x)}\right]$$

$$\geq E_{q(z|x)}\left[\log \frac{p(x,z|\theta)}{q(z|x)}\right]$$ (Evidence lower Bound) ELBO

$$= E_{q(z|x)}[\log p(x,z|\theta) - \log q(z|x)]$$

$$= E_{q(z|x)}[\log p(x,z|\theta)] - E_{q(z|x)}[\log q(z|x)] \quad . \quad \square.$$

Thus, conclusion,

$$\ell(\theta;x) \geq E_q[\ell_c(\theta;x,z)] + H(q) = E_{q(z|x)}\left[\log \frac{p(x,z|\theta)}{q(z|x)}\right] \quad (\text{ELBO})$$

. We can show that (HW).

$\Uparrow$ b/c KL ≥ 0.

ideal distribution; estimated distribution given parameters.

$$\ell(\theta;x) = \underbrace{E_{q(z|x)}\left[\log \frac{p(x,z|\theta)}{q(z|x)}\right]}_{\substack{(\text{ELBO}) \\ \text{M-step}}} + \underbrace{KL(q(z|x) \,||\, p(z|x,\theta))}_{\text{E-step}}.$$

Back to EM

. For fixed data x, define a functional called the (variational) free energy

$$F(q,\theta) = -E_q[\ell_c(\theta;x,z)] - H(q) \geq -\ell(\theta,x)$$

function of function

$$\min F(q,\theta)$$

Thus, EM is coordinated descent on F:

蛋生鸡
鸡生蛋 algorithm

E-step: $q^{t+1} = \arg\min_q F(q, \theta^t)$

M-step: $\theta^{t+1} = \arg\min_\theta F(q^{t+1}, \theta^t)$

(guest lecture)

Pre-training through language modeling.

- $P_\theta(w_t | w_{1:t-1})$, prob. distribution of next word given previous context.

high quality pre-training data

⇓

Internet.

- however, web-data is noisy, dirty, biased.
- copyright and usage constraint.
- data is contaminated with auto-generated text.

↓

training on these data will cause model collapse.

Diversity of data matter.

① pre-training

↓ ← limited data adapt to task.

② Fine-tuning

SGD intuition:

- approach general pre-training loss
- approach local min fine-tuning loss

Full fine-tuning → update all parameters.

Parameter-efficient fine-tuning → update a few existing /new parameters.
(less overfitting).

↓

by injecting a adapter layers into original network, keep other parameters frozen.
(randomly initialized).

prefix finetuning -> learning a small continuous task-specific vector to each transformer block, while keeping the pre-trained LM frozen.

prompt fine-tuning -> A single "soft prompt" prepended to the embedding input.

low-rank Adaptation (LoRA) -> update the $\underbrace{\text{low-rank representation}}^{\text{of full weight.}}$ parameters.
learn a low-rank "diff" between pre-trained and fine-tuned

Pre-training:

Encoder -> bidirectional, condition on the future context.
Autoencoder model, masked language model.

Encode & decoder -> seq 2 seq model

Decoder only -> Autoregressive model, Left-to-right language model.

Encoder: text reconstruction -> encode info. from both bidirection

Prompting:

- complete a sentence
- prompting
  ↓
  inference
- ask output in format.
- Few-shot prompting
  - provide a few example together with the tasks in prompting
- Add "role", "name", "content" as JSON.
- In-context learning
  - order of example matters

- label coverage matters
- label balance matters.

- Replacing correct label with random label will barely hurt the accuracy in in-context learning.
- Chain-of-Throught
  - get the model to explain its reasoning in output
- Zero-shot / Few-shot)- CoT → add a prompt to encourage model to reason ("let's think step by step").
- Structing output as program code (reasoning through coding)

Prompt engineering:

- Manual engineering: format
  ↓
  format of prompt matters in task accuracy.
- Paraphrase a existing prompt
- Reason + Action (ReAct)
- Persona-based prompting
  - role-playing — adapt to user
  - personalization. — adapt to environment.
- self-refinement prompting:

EM continued:

E-step: minimization of $F(q, \theta)$ w.r.t. q.

Optimal solution: $q^{t+1} = \arg\min_q F(q, \theta^t) = p(z|x, \theta^t)$

(the posterior distribution over the latent variables given the data and current parameters).

hint: $l(\theta; x) = E_{q(z|x)}\left[\log \frac{p(x,z|\theta)}{q(z|x)}\right] + KL(q(z|x) \,\|\, p(z|x,\theta))$.

$$= -F(q,\theta) + \underbrace{KL(q(z|x) \,\|\, p(z|x,\theta))}$$

approaches 0, $q(z|x) = p(z|x,\theta)$ as possible.

$\min l(\theta, x)$

M step: minimization of $F(q,\theta)$ w.r.t. θ.

$$F(q,\theta) = -\underbrace{E_q[l_c(\theta; x, z)]}_{\text{expected complete log likelihood}} - H(q) \geq -l(\theta; x).$$

$\Downarrow$

$$\text{so, } \theta^{t+1} = \underset{\theta}{\operatorname{argmax}}\, E_q[l_c(\theta; x, z)]$$

given as estimate

$$= \underset{\theta}{\operatorname{argmax}} \sum_z q^{t+1}(z|x) \underbrace{\log p(x,z|\theta)}_{\text{solve as supervised.}}$$

Gaussian Mixture Model (GMM)

E-step: posterior of z_n given the current estimate of the parameters

$$p(z^k=1 | x, \theta^t) = \frac{p(z^k=1)\,p(x|z^k=1)}{p(x)}$$

$$= \frac{p(z^k=1)\,p(x|z^k=1)}{\sum_{j=1}^{k} p(z^j=1)\,p(x|z^j=1)}$$

$$= \frac{\pi_k N(x|\mu_k, \Sigma_k)}{\sum_{j=1}^{k} \pi_j N(x|\mu_j, \Sigma_j)}$$

mixture proportion

given parameter.

$$= \gamma_k$$

M-step: computing the parameter given current estimate of z_n

$$q^{t+1}(z^k|x) = p(z^k|x, \theta^t) = \gamma^k$$

$$\theta^{t+1} = \arg\max_{\theta} \sum_k q^{t+1}(z^k=1|x) \log p(x, z^k=1|\theta)$$

$$E_{q^{t+1}}[\log p(x, z|\theta)]$$

$$= \sum_k \gamma_k (\log p(z^k=1|\theta) + \log P(x|z^k=1;\theta)).$$

$$= \sum_k \gamma_k \log \pi_k + \sum_k \gamma_k \log N(x; \mu_k, \Sigma_k).$$

we can maximize by solving parameter directly

EM for GMM:

- initialize μ_k, Σ_k, π_k
- iterate until convergence:

  E-step.

  M-step.

EM on LLM reasoning

$$E_{q(y|x)}[\log p(o=1|x,y)] - KL[q(y|x) \| p_\theta(y|x)].$$

$$= E_{q(y|x)} \log p(o=1|x,y) - \log q(y|x) + \log p_\theta(y|x)$$

$< - E_q \log q(y|x) \sim q^*(y|x)$

$$= -KL(q(y|x) \| p(o=1|x,y) p_\theta(y|x))$$

$q^*(y|x) = p(o=1|x,y)\, p_\theta(y|x).$

reweighting the answer

M-step.

E-step

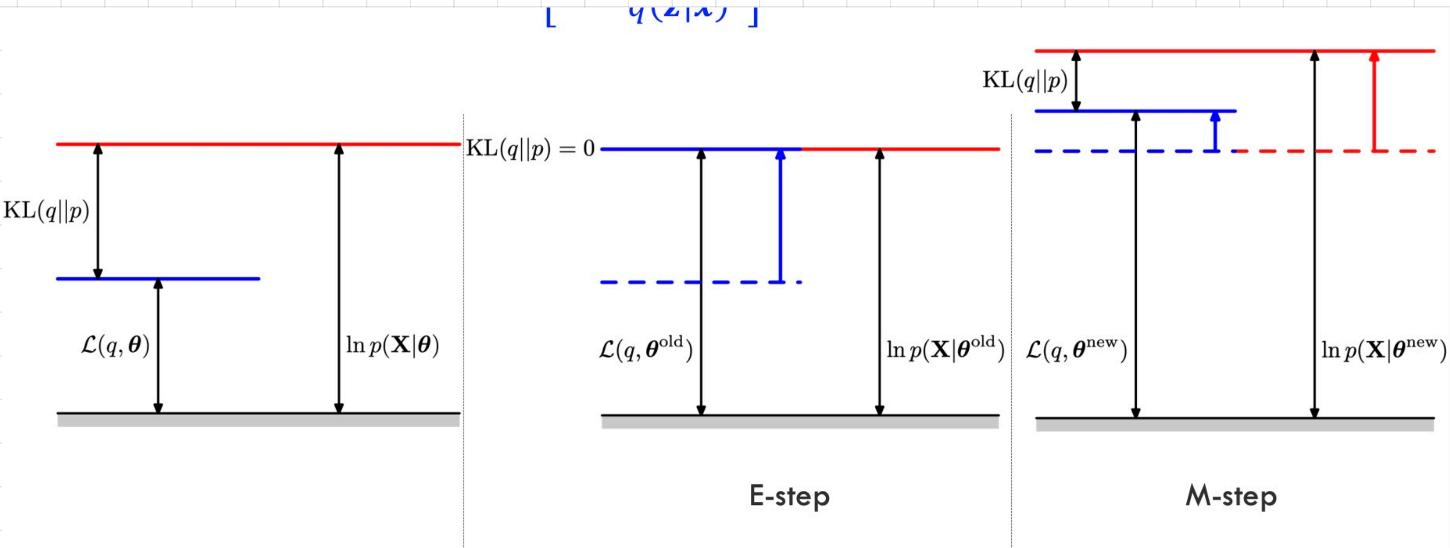
- Generate data

M-step

- optimise θ with positive data.

Replace PPO with EM.

Each EM iteration guarantees to improve the likelihood.



[PRML, Chap 9.4]

5

limitation: need to compute $p(z|x,\theta)$ in E-step.

$$\frac{p(z,x|\theta^t)}{\underbrace{\sum_z p(z,x|\theta^t)}}$$

might be hard to compute.
especially if it is continuous.

$\Rightarrow$ Variational inference.

· Observed variable x

latent variable z.

· Variational Bayes:

used to approximate the posterior distribution over the latent variables.

$P(A,B) = P(A)\,P(B|A)$

$$\boxed{p(z|x,\theta) = \frac{p(z,x|\theta)}{\sum_z p(z,x|\theta)}}$$

$P(z,x|\theta) = P(x|\theta)\,P(z|x,\theta)$

$\downarrow$

$\sum_z P(z,x|\theta)$

in E-M, we assume $q(z|x)$ can be any distribution.

& E-step shows the optimal $q(z|x)$ is the posterior distribution

idea:

1. choose a family of distribution over latent variable $z_{1:m}$ with its own set of variational parameters v.

$q(z_{1:m}|v)$

2. Then we find the setting of the parameters that makes our approximation q closest to the posterior distribution.

$\Rightarrow$ optimization problem.

3. use q with the fitted parameter in place of posterior,

$\Rightarrow$

$$\min_q KL(\underbrace{q(z|x,v)}_{\text{approximation}} \,||\, \underbrace{p(z|x)}_{\text{posterior}}).$$

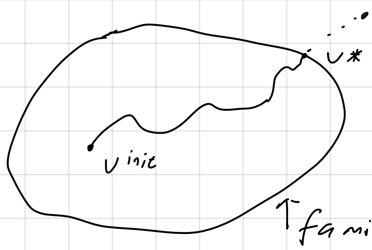
how $\Rightarrow$ $\underbrace{\ell(\theta,x)}_{\text{constant}} + E_{q(z|x)}\left[\log \frac{p(x,z|\theta)}{q(z|x)}\right] = -KL(q(z|x)\,||\,p(z|x,\theta))$.

$\downarrow$

$$\arg\max E_{q(z|x,v)}\left[\log \frac{p(x,z|\theta)}{q(z|x,\theta)}\right]$$

$$= \arg\max_v \mathbb{E}_{q(z|x,v)}\left[\log p(x, z|\theta)\right] - \mathbb{E}_{q(z|x,v)}\left[\log q(z|x,v)\right].$$

idea:



using EM:

$$q(z|x, v^*)$$
$$= \min_v KL(q(z|x,v)\|p(z|x,\theta^t))$$
$$= \min_v F(q(z|x,v), \theta^t) + \text{const.}$$

Q: How to choose variational family of $q(z|x,v)$?

- factorized distribution → mean field VI
- mixture of gaussian distribution → black-box VI
- neural-based distribution → variational autoencoder (VAEs).

Mean field VI:

Assume variational distribution over latent variable factorizes as:

$$q(z) = q(z_1 \cdots z_m) = \prod_{i=1}^{m} q(z_j).$$

independent z_j

"local variational approximation".

ex) Bayesian mixture of gaussians.

Assume mean-field $q(u_{1:k}, z_{1:n}) = \prod_k q(u_k) \prod_i q(z_i)$.

For each data example $i \in D$.

update local variational distribution → E-step

update global distribution & parameters. → M-step.

Until ELBO converges.

Or Stochastic VI

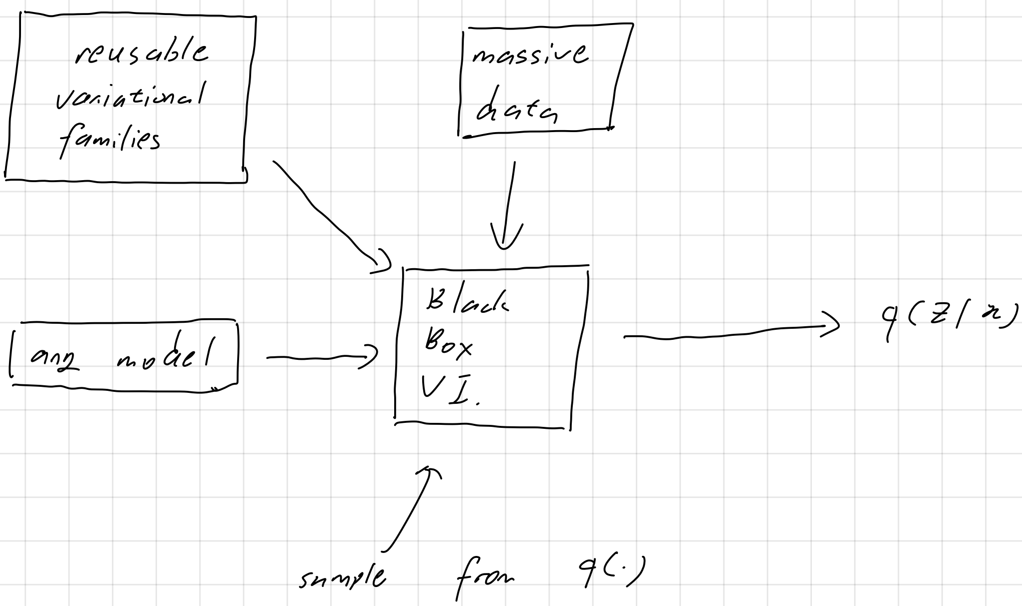
where we sampe data example randomly and update $q(z_i)$.

update global $q(\mu_k)$ with natural gradient asscent.

Black-box VI

limitation: we have to define update rules

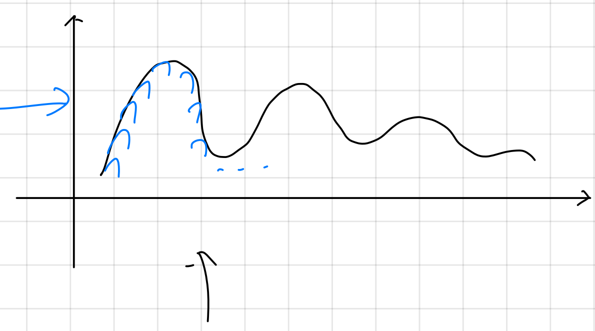
How can we use VI with any model:



form noisy gradient.

· Variational distributin $q_\lambda(z|x)$ with parameter λ.

gaussian mixture distribution as universal approximator



ELBO:

$$L(\lambda) = E_{q(z|\lambda)}[\log p(x,z)] - E_{q(z|\lambda)}[\log q(z|\lambda)].$$

Why deep neural network good?

=> inductive bias

Bottleneck: approximate well only with enough components.

inpractical.

- compute exact gradient is not feasible.

$$L = E_{q_\lambda(z)}[f_\lambda(\lambda)].$$

$\downarrow$

- Score function
- Reparametrization trick

=> Loss : $L = E_{q_\lambda(z)}[f_\lambda(z)]. = \int q_\lambda(z) f_\lambda(z)$

assume we can express the distribution $q_\lambda(z)$ with a transformation.

$$\begin{aligned} &\epsilon \sim s(\epsilon) \\ &z = t(\epsilon, \lambda) \end{aligned} \quad \Leftrightarrow \quad z \sim q(z|\lambda).$$

ex).

$$\begin{aligned} &\epsilon \sim Normal(0,1). \\ &z = \epsilon\sigma + \mu. \end{aligned} \quad \Leftrightarrow \quad z \sim Normal(\mu, \sigma^2).$$

=> After reparameterization :

$$L = E_{\epsilon \sim s(\epsilon)}[f_\lambda(z(\epsilon, \lambda))].$$

$\Downarrow$ chain rule.

Original distribution

reparametrized term.

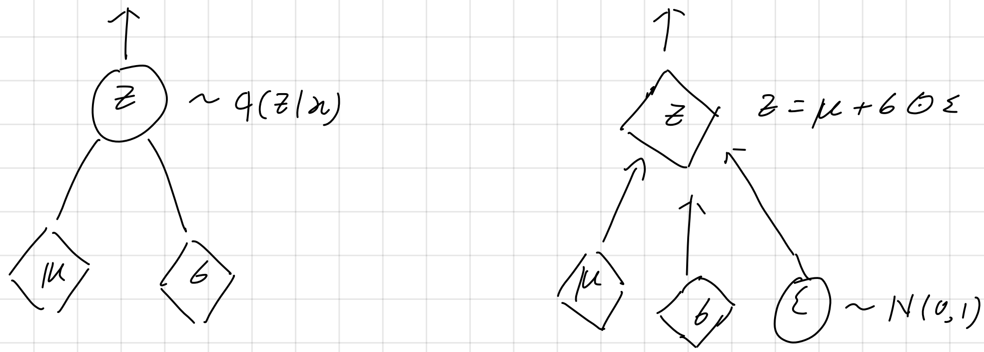
so, gradient $\nabla_\lambda L = E_{\epsilon \sim s(\epsilon)}[\nabla_z f_\lambda(z) \nabla_\lambda t(\epsilon, \lambda)].$

$$= \frac{1}{N} \sum_i \nabla_{z_i} f_\lambda(z_i) \nabla_\lambda z_i \quad \leftarrow \text{MC Simulation.}$$

Pro: low variance of gradient estimate

Con: Not all distribution cane be reparameterized.

$$\nabla_\lambda L = E_{\epsilon \sim s(\epsilon)}\left[\nabla_z[\log p(x,z) - \log q(z)] \nabla_\lambda t(\epsilon, \lambda)\right].$$



VAE

- Variational inference
- Variational distribution parameterized as neural networks.
- reparameterization

· Model: $p_\theta(x, z) = p_\theta(x|z)p(z)$.

generative model

prior (gaussian).

Assume variational distribution $q_\phi(z|x)$.

a gaussian distribution parameterized as deep neural network.

(probabilistic encoder).

ELBO:

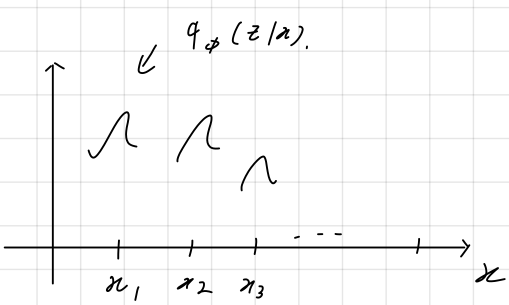
$$L(\theta, \phi; x) = E_{q_\phi(z|x)}[\log p_\theta(x, z)] + H(q_\phi(z|x)).$$

$$= \underbrace{E_{q_\phi(z|x)}[\log p_\theta(x|z)]}_{\text{reconstruction}} - \underbrace{KL(q_\phi(z|x) \| p(z))}_{\text{divergence from prior.}}.$$

generator

both gaussian distribution.

↓ analytical form



• reparameterization:

$$[\mu;\sigma] = f_\phi(x) \quad \text{(a neural network)}$$

$$z = \mu + \sigma \odot \epsilon, \quad \epsilon \sim N(0,1).$$

$$\nabla_\phi L = E_{\epsilon\sim N(0,1)}\left[\nabla_z[\log p_\theta(x,z) - \log q_\phi(z|x)]\,\nabla_\phi z(\epsilon,\phi)\right].$$

$$\nabla_\theta L = E_{q_\phi(z|x)}\left[\nabla_\theta \log p_\theta(x,z)\right].$$

Network:

$\boxed{x}$ input data

$\mu_{z|x}$ $\quad$ $\Sigma_{z|x}$ $\quad$ } encoder network $p_\theta(x|z)$

sample $z|x \sim N(\mu_{z|x}, \Sigma_{z|x})$

$\boxed{z}$ ← latent space representation

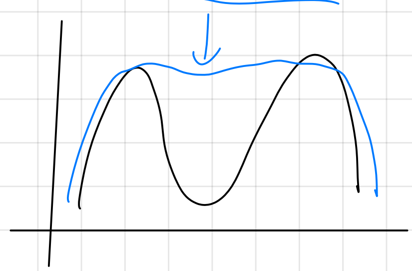
$\mu_{x|z}$ $\quad$ $\Sigma_{x|z}$ $\quad$ } decoder network. $q_\phi(z|x)$.

sample $x|z$ from $x|z \sim N(\mu_{x|z}, \Sigma_{x|z})$

$\boxed{\hat{x}}$

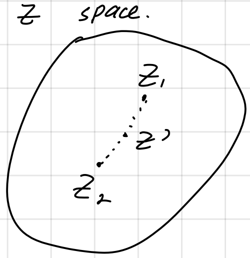
Con: sample blurrier and lower quality

$\underbrace{KL(p_{data} \,|\, p_\theta)}$



VAE for text: multinominal gaussian

z space.



z_1 = sentence 1

z_2 = sentence 2

generate z?.

Variational distribution as an inference model $q_\phi(z|x)$.

Amortized Variational Inference:

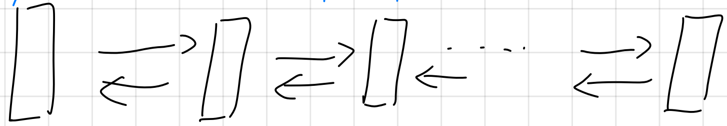
- Amortized the cost of inference by learning a single data-dependent inference model.

for quick inference.

Diffusion Model

sample data $p(x_0)$ → turn to noise

(Clean sample). $q(x_t|x_{t-1})$. (Pure noise).

$P_0(x_0)$  $P_T(x_T) \sim N(0,1)$

reverse / denoising process turn into data ← sample noise $P_T(x_T)$

$P_\phi(x_{t-1}|x_t)$

Better than VAE b/c
- decompose into many steps (2000 steps)
- denosing as supervised learning (ground true data exists).
- ...

DSC 190 DL.

Concept learning through probabilistic program induction.

expence: massive data examples.

Human-level concept learning: require much less data. richer representation.

Bayesian Program Induction → to mimic human-level learning.

↳ The program: a generative model of handwritten characters.

generate type: Primitives given

sample : from original data

↓

subpart : extract and vary

↓

part : combine

↓

object template (relation) : attached along

↓ (motor variance / starting location)

examplars : trajectory.

primitives

↓

program (heuristic).

↓

constrained search space. parameter.

↓

don't need much data to train.

constrained searching :

Bayesian inference: $\text{program}^* = \underset{\text{program}}{\operatorname{argmax}} \; P(\text{program} \mid \text{data})$.

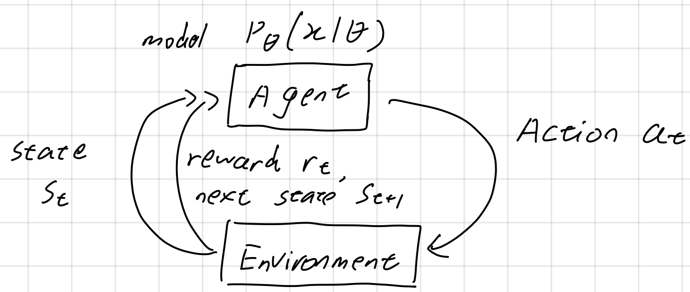
posterior

$$= \underset{\text{program}}{\operatorname{argmax}} \; \frac{P(\text{data} \mid \text{program}) \cdot P(\text{program})}{P(\text{data})}$$

Reinforcement Learning → Post-training for LLM

RLHF

take actions that maximize rewards.



level of supervision ↓

→ SL

→ RL

→ USL

ex)

State: angle & position

Action: torque applied on joints

Reward: 1 at each time step upright + forward movement.

sparse reward

↓

Reward shaping

MDP

$s_t \to a_t$

markov property: current state completely characterises the state of the world.

$(S, A, R, \mathbb{P}, \gamma)$ $\quad r_t \sim r(s_t, a_t) \in \mathbb{R}$

set of possible states | set of possible actions | distribution of reward given pair | transition probability | discount factor.

$\Rightarrow P(s_{t+1} \mid s_t, a_t)$.

to simulate uncertainty in the future for reward. (exponential decay).

At $t=0$, initial state $s_0 \sim P(s_0)$.

- action a_t
- sample reward $r_t \sim R(\cdot \mid s_t, a_t)$
- sample next state $s_{t+1} \sim P(\cdot \mid s_t, a_t)$
- receive reward r_t and next state s_{t+1}

Policy π is a function from S to A that specifies what action to take in each state.

Objective: find policy π^* that maximizes cumulative discounted reward $\sum_{t \geq 0} \gamma^t r_t$.

example) Grid-world: action {right, left, up, down}
reach from one grid to the other
negative "reward" for each transition.

Want to find optimal policy π^*, how to handle randomness
Maximizes expected sum of rewards

$$\pi^* = \arg\max_{\pi} E\left[\sum_{t \geq 0} \gamma^t r_t \mid \pi\right] \quad \text{with} \quad \begin{aligned} & s_0 \sim p(s_0) \\ & a_t \sim \pi(\cdot \mid s_t) \\ & s_{t+1} \sim p(\cdot \mid s_t, a_t). \end{aligned}$$

Following a policy result in sample trajectory: $s_0, a_0, r_0, s_1, a_1, r_1, \ldots$

Value Function

at state s, the expected cumulative reward from following policy from state s.

$$V^{\pi}(s) = E\left[\sum_{t\geq 0} \gamma^t r_t \mid s_0 = s, \pi\right].$$

assume following a policy.

Q-value function: (value func for state-action pair).

$$Q^{\pi}(s,a) = E\left[\sum_{t\geq 0} \gamma^t r_t \mid s_0 = s, a_0 = a, \pi\right].$$

(future expected cumulative reward)

taking action a in state s, the following the policy

expanded form:

$$Q^{\pi}(s,a) = \underset{\substack{\pi(a|s) \\ p(s'|a,s)}}{E}\left[\underset{s_0,a_0}{r_0} + \gamma \boxed{r_1 + \gamma^2 \cdot r_2 + \cdots}\right]$$

s_{t+1}, a_{t+1} another Q-value

$$= \underset{\substack{\pi(a|s) \\ p(s'|a,s)}}{E}\left[r_t + \gamma Q^{\pi}(s_{t+1}, a_{t+1})\right]$$

Bellman Equation: → also follows recursive fashion

optimal Q-value function Q^* maximum expected cumulative reward

$$Q^*(s,a) = \max_{\pi} E\left[\sum_{t\geq 0} \gamma^t r_t \mid s_0 = s, a_0 = a, \pi\right]$$

satisfies

future value

$$Q^*(s,a) = E_{s'\sim\varepsilon}\left[r + \gamma \max Q^*(s',a') \mid s,a\right].$$

intuition: if optimal state-action values for the next time-step $Q^*(s',a')$ are known,

then the optimal strategy is to take action that max expected value of

$$r + \gamma Q^*(s',a').$$

$$\begin{cases} a^* = \arg\max Q^*(s,a) \\ \pi^*(s|a^*) \end{cases}$$

take best action in any state as specified by Q^*

optimal policy

Value iteration algorithm: use bellman equation as an iterative update:

update

$$Q_{i+1}(s,a) = E[r + \gamma \max_{a'} Q_i(s',a') \mid s,a]$$

plug in

Q_i converge to Q^* as $i \to \infty$.

Problem:

Not scalable. must compute all state - action pair.

small scale → Table-based value iteration.

	a_1	a_2
s_1	. .	-
s_2	.	. .
s_3	.	. .

Solution: a function approximator to estimate $Q(s,a)$ (neural network)

$Q^{\theta}(s,a) \to Q$

Deep Q network (Alpha Go).

Q-learning

$Q(s,a;\theta) \approx Q^*(s,a)$ function approximator to estimate action-value function.

Forward pass:

experience database

surprived task.

loss function: $L_i(\theta_i) = E_{s,a \sim p(\cdot)}[(y_i - Q(s,a;\theta_i))^2]$

where $y_i = E_{s' \sim \varepsilon}[r + \gamma \max_{a'} Q'(s',a';\theta_{i-1}) \mid s,a]$

bellman eq as ground true

Backward pass:

learns a function to satisfies bellman equation.

Gradient update:

$$\nabla_{\theta_i} L_i(\theta_i) = E_{s,a\sim p(\cdot);\, s'\sim\varepsilon}\left[r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i) \nabla_{\theta_i} Q(s, a; \theta_i)\right]$$

ex) Atari Game:

$Q(s, a, \theta)$.

FC-4 (Q-val) → 4 actions

↑

FC-256

↑

32 4×4 conv

↑

16 8×8 conv

↑

game screen

(all states)

(what if actions is continuous)
ex) control

↓

pass in action as well

↓

enumerate all possible a for all state
(huge space to search).

Training Q-network trick

- Eexperience Replay

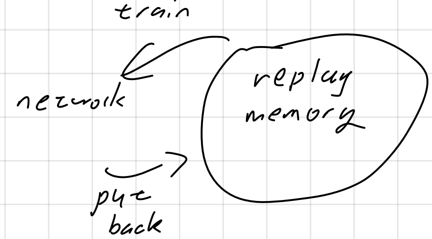
learning from batches of consecutive samples is problematic.

- sample are correlated
- bad feedback loop.

⇒ - Continually update a replay memory table of transitions

(s_t, a_t, r_t, s_{t+1})

- train Q network on random minibatches of transitions from replay memory



Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N — replay memory + Q network
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$ ← initialize state
  for $t = 1, T$ **do**
    exploration / exploitation.
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$ greedy that maximize Q value.
    execute action:
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$ → store data
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$ → sample from experience replay and do GD.

$$\text{Set } y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$$

    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

Derive policy : $a^* = \arg\max_a Q_\theta(s, a)$.

Policy Gradient:
- on-policy
- policy-based

define a class of parameterized policies: $\Pi = \{\pi_\theta, \theta \in \mathbb{R}^m\}$.

for each policy, define its value.

policy based

$$J(\theta) = E\left[\sum_{t \geq 0} \gamma^t r_t \,\middle|\, \pi_\theta\right]$$

Want to find optimal policy $\theta^* = \arg\max_\theta J(\theta)$

Gradient descent on policy parameter.

$$\Rightarrow J(\theta) = E_{\tau \sim p(\tau;\theta)}[r(\tau)]$$

$$= \int_\tau \underbrace{r(\tau)}_{\text{reward of trajectory}} p(\tau;\theta)\, d\tau$$

$$\nabla_\theta J(\theta) = \int_\tau r(\tau) \nabla_\theta p(\tau;\theta) d\tau.$$

use trick:

$$\nabla_\theta p(\tau;\theta) = p(\tau;\theta) \frac{\nabla_\theta p(\tau;\theta)}{p(\tau;\theta)} = p(\tau;\theta) \nabla_\theta \log p(\tau;\theta).$$

$$\nabla_\theta J(\theta) = \int_\tau \left(r(\tau) \nabla_\theta \log p(\tau;\theta)\right) p(\tau;\theta) d\tau.$$

$$= \underbrace{E_{\tau \sim p(\tau;\theta)}}_{\text{policy based}} \left[\underbrace{r(\tau) \nabla_\theta \log p(\tau;\theta)}_{\text{estimate with: Monte Carlo sampling}} \right]$$

$$p(\tau;\theta) = \prod_{t \geq 0} p(s_{t+1} \mid s_t, a_t) \pi_\theta(a_t \mid s_t)$$

$$\log p(\tau;\theta) = \sum_{t \geq 0} \cancel{\log p(s_{t+1} \mid s_t, a_t)} + \log \pi_\theta(a_t \mid s_t).$$

$$\nabla_\theta \log p(\tau;\theta) = \sum_{t \geq 0} \nabla_\theta \log \pi_\theta(a_t \mid s_t)$$ (doesn't depend on transition probability).

Intuition:

if $r(\tau)$ is high, push up probabilities of actions seen.

if $r(\tau)$ is low, push down probabilities of actions seen.

In expectation, if a trajectory is good, then all its actions were good, then it averages out.

RL for LLM

(Autoregressive) text generation model:

sentence $y = (y_0 \cdots y_T)$

$\uparrow$

trajectory, $\tau = s_0, a_0, r_0, s_1, a_1, \cdots$

logits

$$\pi_\theta (y_t | y_{<t}) = \text{softmax}(f_\theta (y_t | y_{<t})).$$

action state

policy $\pi_\theta (a_t | s_t)$

- Reward $r_t = r(s_t, a_t)$

  $\Downarrow$ sparse, $r_t = 0$ for $t < T$. (until sequence ends).

- General RL objective: maximum cumulative reward $J(\pi) = E_{\tau \sim \pi} \left[\sum_{t=0}^{T} \gamma^t r_t \right]$.
- Q function: expected future reward of taking action a_t is state s_t.

$$Q^\pi (s_t, a_t) = E_\pi \left[\sum_{t'=t}^{T} \gamma^{t'} r_{t'} \mid s_t, a_t \right]$$

GPT3.5 $\rightarrow$ ChatGPT:

(supervised fine-tunning & RLHF).

1. Collect demonstration data & train a supervised policy

① prompt sampled from prompt dataset.

② labeler demonstrate desired output

③ data used to fine-tuned LLM (supervised learning).

2. Collect comparison data & train a reward model.

$\Downarrow$

LLM output

labeler ranks output $\} \rightarrow$ used to train our reward model.

3. Optimize policy against reward model.

new prompt sampled from dataset.

↓

policy generates output

↓

reward model calculates a reward for the output.

update policy using PPO/DPO...