Deep learning.

representation learning; end-to-end learning → params in each layer are trained w.r.t. overall loss.

input → learned feature extractor → learned features → learned classifier → label.

Big Model + Large Datasets + Enough Compute.

good at scaling → the ability to work better w/ more data

inductive bias vs learning.

Machine Learning Review:

① Supervised learning. Model → Loss fn → optimizer.

- output probability → entire distribution over all instead of output themselves. / probabilistic models

$$\underbrace{\text{Softmax}(f_\theta(x))_c}_{\text{make output positive}} = \frac{\exp[f_\theta(x)]_c}{\sum_{i=1}^{k}\exp[f_\theta(x)]_i} = P_\theta(y=c|x)$$ (logits) — k possible labels.

- Maximum likelihood Estimation. MLE * consistency.

$D = \{(x_1,y_1),(x_2,y_2),\cdots,(x_N,y_N)\}$. $\theta_{MLE} = \underset{\theta\in\Theta}{\text{argmax}}\ P(D|\theta) = \underset{\theta\in\Theta}{\text{argmax}} \prod_{i=1}^{N} P(x_i) \underbrace{P_\theta(y_i|x_i)}_{P_\theta(x_i,y_i)}$

Negative Log Likelihood:

$\underset{\theta\in\Theta}{\text{argmax}} \sum_{i=1}^{N} \log P(x_i) + \log P_\theta(y_i|x_i) = \underset{\theta\in\Theta}{\text{argmax}} \sum_{i=1}^{N} \log P_\theta(y_i|x_i) \Rightarrow \ell(\theta; x_i, y_i) = \underset{\theta\in\Theta}{\text{argmin}} \sum_{i=1}^{N} -\log P_\theta(y_i|x_i)$

Cross Entropy Loss: (Maximizing log likelihood ≈ minimizing cross-entropy). ($\log P(x)$: const. w.r.t. θ)

$H(p,q) = -\sum_x p(x)\log q(x) = E_p[-\log q(x)]$.

$H(p_{data}, p_\theta) = E_{p_{data}}[-\log p_\theta(x,y)] = E_{p_{data}}[-\log P(x) - \log p_\theta(y|x)] \approx \sum_{i=1}^{N} -\log(\underbrace{P(x_i)}_{\text{const. w.r.t. }\theta} - \log P_\theta(y_i|x_i)$

- Iterative optimization. / gradient based optimization.

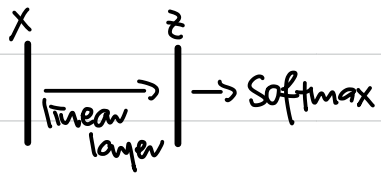
$\theta \leftarrow \theta - \alpha\nabla_\theta \frac{1}{N}\sum_{i=1}^{N} \ell(\theta; x_i, y_i)$

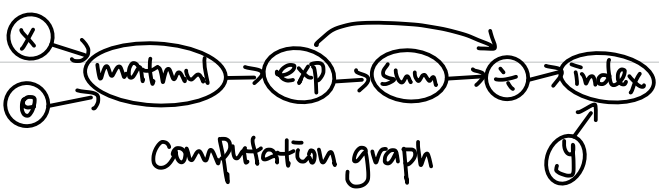
0-1 loss fn is not differentiable.

- logistic regression

$\ell(\theta; x, y) = -\log P_\theta(y|x) = -\log \text{Softmax}([f_\theta(x)]_c)$ ($f_\theta(x) = \theta^T x$)

$\theta \leftarrow \theta - \alpha\Delta_\theta \frac{1}{N}\sum_{i=1}^{N} \ell(\theta; x_i, y_i)$





Computation graph

- θ COULD be a $d \times (k-1)$ matrix.

Example: $k=2$ $P_\theta(y=1|x) = \text{Softmax}([\theta_{:,0}, \theta_{:,1}]^\top x)_1 = \frac{\exp \theta^\top x}{\exp \theta^\top x + 1}$

$\ell(\theta; x_i, y_i) = (1-y_i)(\log\{\exp \theta^\top x + 1\}) - y_i(\theta^\top x - \log\{\exp \theta^\top x + 1\})$.

$\nabla_\theta \ell(\theta; x_i, y_i) = \left(\frac{\exp \theta^\top x}{\exp \theta^\top x + 1} - y_i\right) x$.

- true risk $R(\theta) = E[\ell(\theta; X, y)]$ whole dataset.
  empirical risk $\hat{R}(\theta) = \frac{1}{N}\sum_{i=1}^{N} \ell(\theta; x_i, y_i)$ ERM. training set.

$R(\theta) \not\approx \hat{R}(\theta)$ cant reuse the training data.

high true risk, low empirical risk. → overfitting. too small dataset / too powerful model

high true risk, high empirical risk → underfitting. weak model / bad optimization.

in general, true risk ≥ empirical risk.

- model class: the set of all possible fns that the model can represent via different params.

capacity = |model class|.

training set. – learning θ. test underfitting. optimization.
validation set. – estimating $R(\theta)$. test overfitting.

test set – only use once.

combat overfitting. even underfitting

l_2 norm = MSE = $\frac{1}{n}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$ l_1 norm = MAE = $\frac{1}{n}\sum_{i=1}^{N}|y_i - \hat{y}_i|$

- regularizer: anything we add to the loss fn / optimization that NOT depend on data.
  → inductive bias

Maximum a posteriori (MAP.)

$\sum_{i=1}^{N} -\log P(y_i|x_i, \theta) - \log p(\theta)$. $p(\theta) = N(\theta; 0, \sigma^2 I)$, so $-\log p(\theta) = \sum_{i=1}^{D} \frac{1}{2}\frac{\theta_i^2}{\sigma^2} + \text{const.} = \lambda \|\theta\|_2^2$ where $\lambda = \frac{1}{2\sigma^2}$

$\Downarrow$

$\sum_{i=1}^{N} -\log P(y_i|x_i, \theta) + \lambda\|\theta\|_2^2$ l_2 regularization. = Ridge regression.

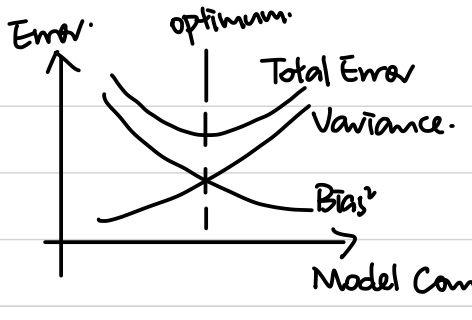
hyperparameter, "weight decay". Smaller λ → smoother.

- Bias–Variance Decomposition.

$E[(f_{\theta(D)}(x') - y')^2] = (\bar{f}(x') - f(x'))^2 + E[(f_{\theta(D)}(x') - \bar{f}(x'))^2] + \sigma^2$ where $\bar{f}(x') = E[f_{\theta(D)}(x')]$.

bias. ↓ Variance ↓ irreducible error

overfitting: learn different params for similar training set. → High Variance
underfitting: learn similar params for different training set. → High Bias

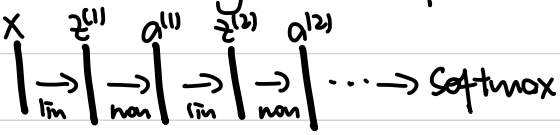


We don't fear overfitting.

Neural Network.

- can represent nonlinear fns. Add nonlinearity = Activation fn. σ (elementwise on linear layer output)

Ex. $\tanh(z)$. $\text{sigmoid}(z) = \frac{1}{\exp -z + 1}$, $\text{ReLU}(z) = \max(0, z)$.



$W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)}, \dots, W^{(L)}, b^{(L)}, W^{(\text{final})}, b^{(\text{final})}$ L hidden layers.

$f_\theta(x) = \text{softmax}(A^{\text{final}}(\sigma(A^{(L)}(\dots \sigma(A^{(1)}x))\dots)))$

$A^{(i)}(v) = W^{(i)}v + b^{(i)}$

- Calculating gradients — finite differences $\frac{\partial f}{\partial x_i} \approx \lim_{\varepsilon \to 0} \frac{f(x+\varepsilon e_i) - f(x-\varepsilon e_i)}{2\varepsilon}$

  backpropagation (matrix-vector products).

Example: 2-layer

$z^{(1)} = W^{(1)}x_i + b^{(1)}$ $\quad a^{(1)} = \sigma(z^{(1)})$ $\quad z^{(2)} = W^{(2)}a^{(1)} + b^{(2)}$ $\quad a^{(2)} = \sigma(z^{(2)})$ $\quad z = W^{\text{final}}a^{(2)} + b^{\text{final}}$

$p_\theta(y_i | x_i) = \frac{\exp z}{\sum \exp z}$ (vector / number) $\quad \log p_\theta(y_i | x_i) = z_{y_i} - \log \sum \exp z$ $\quad \ell(\theta; x_i, y_i) = \log \sum \exp z - z_{y_i}$

$\nabla_z \ell = \frac{\exp z}{\sum \exp z} - e_{y_i}$ (one-hot vector) $\quad \nabla_{a^{(2)}} \ell = \frac{dz}{da^{(2)}} \nabla_z \ell = W^{\text{final}\,T} \nabla_z \ell$.

$\nabla_{W^{\text{final}}} \ell = \frac{dz}{dW^{\text{final}}} \nabla_z \ell = (\nabla_z \ell) a^{(2)T}$. $\quad \nabla_{b^{\text{final}}} \ell = \frac{dz}{db^{\text{final}}} \nabla_z \ell = \nabla_z \ell$.

$\nabla_{z^{(2)}} \ell = \frac{da^{(2)}}{dz^{(2)}} \nabla_{a^{(2)}} \ell = [\sigma'(z_1^{(2)}) \cdots \sigma'(z_{d_2}^{(2)})] \nabla_{a^{(2)}} \ell$. $\quad \nabla_{a^{(1)}} \ell = \frac{dz^{(2)}}{da^{(1)}} \nabla_{z^{(2)}} \ell = W^{(2)T} \nabla_{z^{(2)}} \ell$.

$\nabla_{W^{(2)}} \ell = \frac{dz^{(2)}}{dW^{(2)}} \nabla_{z^{(2)}} \ell = (\nabla_{z^{(2)}} \ell) a^{(1)T}$ $\quad \nabla_{b^{(2)}} \ell = \frac{dz^{(2)}}{db^{(2)}} \nabla_{z^{(2)}} \ell = \nabla_{z^{(2)}} \ell$.

Jacobian $J_{ij} = \frac{\partial f_i}{\partial x_j}$ $\quad$ Hessian $(H f)_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j}$

Auto Diff

> Jacobian-vector product JVP (forward)

build one Jacobian column at a time.

$\frac{\partial f}{\partial x_1} = Jf(x)e_1$ ($e_1 \in \mathbb{R}^n$), $\frac{\partial f}{\partial x_2} = Jf(x)e_2$, $\cdots$, $\frac{\partial f}{\partial x_n} = Jf(x)e_n$.

$(a \to b) \to (a, Ta) \to (b, Tb)$

$O(1)$ times the FLOPs $O(n(m_1m_3 + m_3m_2 + m_2m_1 + m_1n))$ $\sim 2\times$ memory.

> Vector-Jacobian Product VJP (backward)

build one Jacobian row at a time.

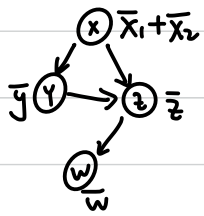
$\nabla f_1(x) = e_1^T Jf(x)$, $\nabla f_2(x) = e_2^T Jf(x)$, $\cdots$, $\nabla f_m(x) = e_m^T Jf(x)$

$(a \to b) \to a \to (b, CT\ b \multimap CT\ a)$

$O(1)$ times the FLOPs $O(m(m_1m_3 + m_3m_2 + m_2m_1 + m_1n))$ $O(\text{depth})$ memory.

< Primitives
Composites

```
def f(x):
    y = sin(x)   z = x * y   w = cos(z)
    return w.
```



x appears twice.

Optimization

we don't really care about reaching the optimum.

- critical point → global optimum / local optimum / saddle point / everywhere.
- Stochastic Gradient Descent (SGD).

For each epoch, shuffle the data, construct mini batches B from consecutive data points.

compute $\nabla_\theta \frac{1}{B} \sum_{i=1}^{B} l(\theta; x_i, y_i)$

learning Rate: high → may cause oscillation / divergence. Low → may stop learning too early.

(learning curve: epoch vs Loss)

learning rate schedule: Linear decay $\alpha_t = \alpha_{initial} \cdot (1 - \frac{t}{\text{max\_steps}})$

Cosine decay $\alpha_t = \alpha_{initial} \cdot 0.5 \cdot [1 + \cos(\pi + \frac{t}{\text{max\_steps}})]$

linear warmup for large $\alpha_{initial}$.

$\alpha_{initial} = 0.001$ (need tuning) $k\alpha_{initial} \to kB$.

- **Momentum.** "remember" the steps before

$\theta \leftarrow \theta - \alpha g$. before: $g \leftarrow \nabla_\theta \frac{1}{N}\sum_{i=1}^{N} \ell(\theta; x_i, y_i)$ now: $g \leftarrow \nabla_\theta \frac{1}{N}\sum_{i=1}^{N} \ell(\theta; x_i, y_i) + \mu g$. ($\mu$: 0.9)

exponential moving average

- **Nesterov's accelerated gradient.**

$\theta \leftarrow \theta - \alpha g$. $g \leftarrow \nabla_\theta \frac{1}{N}\sum_{i=1}^{N} \ell(\theta - \alpha\mu g; x_i, y_i) + \mu g$.

- **Adam**

(combine Momentum and second moment adjustment. Normalize the magnitude).

$\theta \leftarrow \theta - \alpha g$ Momentum: $m \leftarrow (1-\beta_1)\nabla_\theta \ell + \beta_1 m$ (β_1: 0.9)

$E(X^2)$ second moment estimate: $v \leftarrow (1-\beta_2)(\nabla_\theta \ell)^2 + \beta_2 v$. ($\beta_2$: 0.999)

detail - bias correction: $\hat{m} = m/(1-\beta_1^t)$ $\hat{v} = v/(1-\beta_2^t)$

$g = \hat{m}/(\sqrt{\hat{v}} + \varepsilon)$ (ε: 10^{-8})

- **Adaptive learning rate** (rescale separately for each dimension).

$\theta_k^{t+1} = \theta_k^t - \frac{\alpha}{\sqrt{s_k^t + \varepsilon}} \nabla_{\theta_k} L(\theta^t)$

RMSprop $s_k^t = \beta s_k^{t-1} + (1-\beta)|\nabla_{\theta_k} L(\theta^t)|^2$ moving average.

Adagrad $s_k^t = s_k^{t-1} + \beta|\nabla_{\theta_k} L(\theta^t)|^2$ sum. → s^t monotonically increases.

- **Weight decay** $\theta \leftarrow (1-\lambda)\theta$.

for SGD, weight decay = l_2-regularization.

for Adam, either weight decay (AdamW) or l_2-regularization.

building_blocks

- **Input/Standardization.** (output / Normalization) Preprocessing step / circularize the loss landscape. Conti. only for each dimension d, $\mu_d = \frac{1}{N}\sum_{i=1}^{N} x_d$ $\sigma_d = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_d - \mu_d)^2}$ (channel if image)

- **normalizing activations.**

Batch Normalization BN: → normalizing $z^{(l)}$ or $a^{(l)}$ by Batches. * large B * i.i.d in B.

= add a BN layer before / after activation.

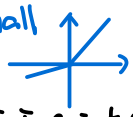
after normalization, each dimension $* \gamma_d' + \beta_d'$. (scale, shift)

< train mode

test mode. → use statistics computed in train mode. on one test sample.

* aggregated.

⑤ Leaky ReLU(x) = $\begin{cases} \alpha x & x<0 \\ x & x=0 \end{cases}$ α is small (convex)



⑥ Exponential linear unit ELU(x) = $\begin{cases} \alpha(e^x-1) & x<0 \\ x & x\geq 0 \end{cases}$ (convex)

enables higher α → faster training; training stability ↑; requires large B.

Layer Normalization. LN.

operates on single data points. / by feature dimension, shared across dimension.

also learnable scales and shifts

- nonlinearity.

① ReLU(v) = max{0, v}. (convex) $\nabla_v \text{ReLU}(v) = \text{diag}(\mathbb{1}[v>0]) \leq 1$ no gradient update for negative values.

gradient ≤ 1.

④ $\tanh(z) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

② Sigmoid(v) = $\frac{1}{1+\exp(-v)} = \frac{\exp(v)}{\exp(v)+1}$ (convex + concave) $\nabla \text{Sigmoid}(v) = \text{diag}[\text{sigmoid}(v) \odot (1-\text{sigmoid}(v))] \leq \frac{1}{4}$

very small gradients for large values.

Gaussian Error linear units ③ GELU(v) = $v \odot \Phi(v)$ (Φ: cdf of $N(0,1)$)

- skip connections = residual connections.

$a^{(l)} = \sigma(z^{(l)}) + a^{(l-1)}$ instead of $a^{(l)} = \sigma(z^{(l)})$ better loss landscape.

- weight initialization. $x_j \sim N(0,1)$ $E[z_i^2] = \sum_j E[(w^{(l)}_{ij})^2] E[x_j^2] = d\sigma_w^2$

for every linear layer, $W_{ij} \sim N(0, \sigma_w^2)$ $\sigma_w^2 = \frac{1}{d}$ → din $\text{Var}(X) = E[X^2] - (E[X])^2$

- dropout. * inductive bias

training only; each iteration, randomly zero out p of W_{ij}.

- ensembles

variance ↓, overfitting ↓. uncertainty quantification.

in practice: all, majority.

Convolutional Networks. * linear operator.

$\frac{H+2p-k}{s}+1$ floor division

- filter [k, k, 0, I]; bias - [O]

input [I, H, W], output [O, H', W'] [H', W'] = ~~([H, W] + 2×pad − (k − 1))/stride~~

- Convolution → BN / LN → ReLU

[N, C, H, W] on N, H, W / on C, H, W

- Pooling — Max Pooling — nonlinear
  Average Pooling — linear

- Forward: $a^{(l)}[H, W, C] \to z^{(l+1)}[H', W', C']$ using filter $W^{(l+1)}[k, k, C', C]$

$z^{(l+1)}[i,j,k] = \sum_{a=0}^{k-1} \sum_{b=0}^{k-1} \sum_{c=0}^{C-1} w^{(l+1)}[a,b,k,c] a^{(l)}[i+a, j+b, c]$

Matrix-Vector Notation: $z^{(l+1)}[i,j] = \sum_{a=0}^{k-1} \sum_{b=0}^{k-1} W^{(l+1)}[a,b] a^{(l)}[i+a, j+b]$

(z: C'; W: $C' \times C$; a: C)

- Backward: $\frac{\partial z^{l+1}[i,j]}{\partial a^l[i+a,j+b]} = W^{l+1}[a,b]$ $\quad \frac{\partial L}{\partial a^l[i,j]} = \sum_{a=0}^{k-1}\sum_{b=0}^{k-1} W^{l+1}[a,b]\frac{dL}{dz^{l+1}[i-a,j-b]}$

$\frac{\partial z^{l+1}[i,j]}{\partial w^{l+1}[a,b]} = a^l[i+a,j+b]$ $\quad \frac{\partial L}{\partial w^{l+1}[a,b]} = \sum_{i=0}^{H'-1}\sum_{j=0}^{W'-1} a^l[i+a,j+b]\frac{\partial L}{\partial z^{l+1}[i,j]}$

Computer Vision.

↪ e.g. flip, crop inductive bias

data augmentations invariances

the 3R's of Vision: Recognition, Reconstruction, Reorganization.

↓ attach semantics labels to objects ↘ obtain 3D info ↘ partition image based on semantics info.

Image classification

Object Localization: determine the bounding box.

Object Detection: measured by mean-average-precision mAP. ; several objects. ; R-CNNs.

Semantics Segmentation: Label every pixel in the image; CNN, UNet.

Four challenges facing AI today: the limitations of supervision; static dataset AI VS embodied AI; perception disconnected from action; the semantic gap.

the six lessons: be multi-modal; be incremental; be physical; explore; be social; use language.

*: empirical / training loss

Style transfer.

image generation: freeze parameters; update image.

input random image ⟷ CNN ⟷ V ⟷ $\ell(V)$.

Not exactly the same. ← $\ell_c(V)=\frac{1}{2}\sum_{ij}(V_{ij}-C_{ij})^2$.

$G_{ij}=\sum_k V_{ik}V_{jk}$

$\ell_s(V)\propto\sum_{ij}(G_{ij}-G^S_{ij})^2$

$\ell=\alpha\ell_c+\beta\ell_s$ $\alpha/\beta=10^{-3}$.

output of intermediate conv layer.

(channel, height * width) 2D.

Style loss operates on multiple intermediate activations of CNN with different weights. (Not ℓ_c).

Recurrent Networks.

inputs: sequential; varying length. outputs: anything / nothing.

weight sharing. $a^{(t+1)}=\sigma(W[a^{(t)};x^{(t)}]+b)$. recurrent.

$a^{(0)}$ – independent from input.

$$*\begin{cases} h_t=\tanh(W_{hh}h_{t-1}+W_{hx}x_t+B_h) \\ y_t=W_{hy}h_t+B_y \end{cases}$$

sequence input, single output / sequence input, sequence output. < Autoregressive

gradient < explode – clip. vanish. – skip connection; LSTM.

$\frac{dL}{dw}=\frac{dL}{da^t}\frac{da^t}{dw}+\frac{dL}{da^{t-1}}\frac{da^{t-1}}{dw}+\cdots+\frac{dL}{da^1}\frac{da^1}{dw}$

↓ large / small ++. 乘 n 次 W.

bidirectional models

LSTM: $f_t=\sigma(X_tU^f+h_{t-1}W^f)$ forget Gate. < 1: remember 0: forget.

LT memory c + ST hidden h.

$i_t=\sigma(X_tU^i+h_{t-1}W^i)$ input/update Gate.

$o_t=\sigma(X_tU^o+h_{t-1}W^o)$ output Gate.

$\tilde{c}_t=\tanh(X_tU^g+h_{t-1}W^g)$.

memory. ↓

$C_t=f_t\circ C_{t-1}+i_t\circ\tilde{c}_t$

output.

$h_t=\tanh(C_t)\circ o_t$.

$$\frac{dJ}{dW^g}=\sum_{t=1}^{T}\frac{dJ}{dh^t}\frac{dh^t}{dW^g}=\sum_{t=1}^{T}\frac{dJ}{dh_t}\left(\frac{\partial h_t}{\partial C_t}\left(\frac{\partial C_t}{\partial W^g}+\frac{\partial C_t}{\partial C_{t-1}}\frac{\partial C_{t-1}}{\partial W^g}\right)+\frac{\partial h_t}{\partial h_{t-1}}\frac{\partial h_{t-1}}{\partial W^g}\right)$$

↑ *(=f_t)

- 1-1: classification, Semantic segmentation.
- 1-n: image captioning
- n-n: language translation.

Transformer.

Sequential output ("A bird flying over a body of water.") one step at a time.

Attention: (Luong)

X → top layers of Pretrained Conv Net → $\begin{bmatrix} c_1 \\ \vdots \\ c_L \end{bmatrix}$

Lower D. ⇒ Not pixels directly.

context matrix.

<START> → Model → y_1 "a"

Model → y_2 "bird"

q_2 want k that is most similar to q_2

$k_1 \to e_{1,2}=k_1^Tq_2$ … $k_L \to e_{L,2}=k_L^Tq_2$ → Softmax → $\alpha_{1,2}$ … $\alpha_{L,2}$

$=\sum_l \alpha_{l,2}V_l$.

Similarity score. ↓ Normalized.

key-value-query: $q_t = q(\text{model info})$. Not y. e.g. want subject.

(all (learned). $k_t = k(C_t)$ e.g. $(k_1, V_1) = (\text{subject, bird})$. $(k_2, V_2) = (\text{background, water})$.

$V_t = V(C_t)$

Self-Attention as a Layer: (handle sequential input). *linear. => processing whole sequence at once.

$d\text{-}q_t = q(X_t)$ $X_2 \to q_2$

$d\text{-}k_t = k(X_t)$ $k_1 \to e_{1,2} = k_1^T q_2$, $k_T \to e_{T,2} = k_T^T q_2$ $\to$ [softmax] $\to a_{1,2}, a_{T,2}$ $\quad a_2 = \sum_t a_{t,2} V_t$

$V_t = V(X_t)$

$X_1 \to a_1$, $X_T \to a_T$ (SA ... SA)

Scaled dot product attention: $\frac{e_{t,2}}{\sqrt{d}}$

Transformer Encoder.

$X_1, \dots, X_T \to$ feed forward (position-wise) $\to h_1^{(1)}, \dots, h_T^{(1)}$ $\to$ SA $\to a_1^{(1)}, \dots, a_T^{(1)}$ $\to$ feed forward $\to h_1^{(2)}, \dots, h_T^{(2)}$ $\to$ SA $\to a_1^{(2)}, \dots, a_T^{(2)}$ $\to \dots$ feed forward $\to z_1, \dots, z_T$.

e.g. representation, embeddings ... used for downstream tasks.

positional encoding to each $h_t^{(1)}$. (add d)

$d\text{-}p_t = \begin{bmatrix} \sin(t,d) \\ \cos(t,d) \\ \vdots \end{bmatrix}$ *relative position.

Multi-Head Attention: if want to capture multiple info at a time.

Independent heads -> concatenate to get a_2.

if $D = 512$ for one head, scale down $D = 64$ for q, k, V for 8 heads.

transformers: optimization is hard: BN, LN.

LN, dropout, skip connection.

forward: $X +$ SelfA.dropout(sublayer(SelfA.norm(X))). (feed forward / SA)

transformer decoder. < training - feed truth. testing - generate $y_1 \dots y_T$ -> feed back.

y_0 = <start> $\to$ ff $\to h_1^{(1)} \to$ SA $\to a_1^{(1)} \to \dots$ ff $\to y_1$

$y_{t-1} \to h_t^{(1)}, a_t^{(1)} \to y_t$

current time stamp.

Masked Attention: training. prevent from seeing the future.

e.g. $e_{T,2} = -\infty \Rightarrow a_{T,2} = 0$. by MA, we can train T tokens simultaneously.

① Original transformer

$[X_1, \dots, X_T] \to$ encoder $\to [z_1, \dots, z_T] \to$ decoder $\to [y_1, \dots, y_T]$ seq-to-seq model for translation.

② BERT. encoder only.

GLUE benchmark. (GLUE tasks).

pre-training: data+++; <CLS> <Sentence 1> <SEP> <Sentence 2>
fine tuning: data-i specific task.

word-level representation.
randomly mask 15% tokens → predict them.
Predict the order of pair sentence. (binary classifier).
sentence-level representation.

③ Vision transformers (ViTs). encoder only.

image → pitches as seq. + positional encoding (learned)
↗ non overlap. 16x16. ↗ no need to generalize.

Swin: shifted window to perform SA.

unsupervised ViT: train 2 ViTs to be similar.

masked autoencoder: MAE. mask 75% → encoder → 25% + 75% empty → "decoder"

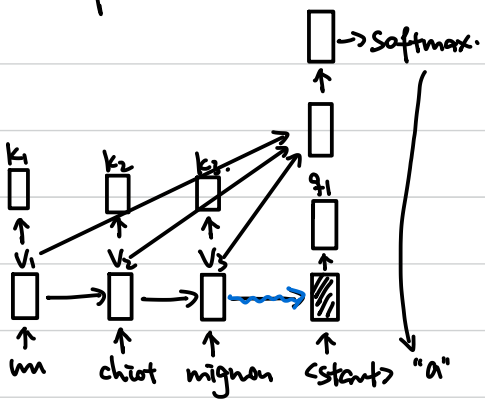
④ Reinforcement Learning. GPT-3 decoder only.

parameters+++; good at answering questions, summary.

Seq2seq. encoder-decoder Machine translation.

basic RNN(LSTM) seq2seq. > bottleneck problem.

Multilayer RNNs.



Cross Attention.

key, value from encoder. query from decoder.

IS

trained on CM; text input, text output.

learns the prompt along with the text.

Natural Language Processing.

M - size of vocab. T - max length of seq.

- greedy encoding. - extend single hypothesis. $O(MT)$.
- Beam Search. - extend multiple hypothesis. prune.

  every t, consider all possibilities, keep top k (largest p) only.

  e.g.t. low p to all words.

  prefer short; length normalization: $\frac{s(e)}{m}$. OR add bonus for extra word.

  $s(e) = \sum_{i=1}^{m} \log p(e_i | e_{1:i-1}, f)$;

Label Smoothing: 90%, 10%.

Byte-Pair Encoder. instead of <UNK>.

Back Translation.

<u>unsupervised pretraining.</u> (get embedding).

trained on some task, use hidden state as embedding.

ELMo: bidirectional LSTM, concatenate hidden states.

GPT: transformer-based <u>forward</u> language model (past context only); text generation.

BERT: (bidirectional models are inherently unidirectional.) => single transformer encoder

incorporate bi context. ; No masked attention.

T5: BERT style mask, but encoder-decoder; reform all to unified text-to-text format.

more general and flexible; multi-task learning.

(transformer over LSTM: model long-term dependencies and parallelization).

Distribution Shift.

when test set has a different distribution than training set.

- Risk.
- Empirical Risk (ERM). training only. generalization bounds. The big lie of ML.

Benchmark: ① ImageNet challenge test sets.

measure robustness under different DS; stress test; same classes.

② Wilds - different domains.

task of premise-hypothesis. contradiction, neutrality, entailment.

③ in NLP. Adversarial natural language inference (ANLI) dataset.

Against DS: ① training larger models on larger datasets.

② Data Augmentation.

-Mixup: element-wise convex combinations of data points. $(\lambda x_1+(1-\lambda)x_2, \lambda y_1+(1-\lambda)y_2)$ from ≠ images.

-AutoAugment: learns augmentation strategies.

-AugMix: mix random augmentation. autoaugmentation.

-PixMix: mix in separate image datasets

③ Alternative / additional training objectives.

ViT: masked autoencoding objective.

Anomalty: Anomaly score, generative model. (low p(x) - Anomaly). x

Detection. Simple baseline: Confidence = $\max_k p_\theta(y=k|x)$ Anomaly score = $-\max_k p_\theta(y=k|x)$.

(out-of-distribution). cannot detect adversarial examples.

Benchmark: train a model on one dataset, and treat any other datasets as anomalty.

e.g. ImageNet-22k, species.

binary classification problem. TP/TN/FP/FN.

Model calibration. - compare confidence against accuracy. $E[|P(\hat{Y}=Y|\hat{P}=p)-p|]$.

eg. 60% confidence rainy => rain 60% of time. $P(\hat{Y}=Y|\hat{P}=p)=p \ \forall p\in[0,1]$

① ensemble of models.

② (Single model) Add a softmax temperature T.

$\hat{p}(y=i|x)=\frac{\exp(f_i/T)}{\sum_{j=1}^{k}\exp(f_j/T)}$ T-hyperparameter. only calibration.

minimize negative log-likelihood.

> Expected Calibration Error (ECE).

group predictions into M bins. $B_m=\{\hat{y}_i,\hat{p}_i\}$ s.t. $\hat{p}_i\in(\frac{m-1}{M},\frac{m}{M})$.

$acc(B_m)=\frac{1}{|B_m|}\sum_{i\in B_m}1\{\hat{y}_i=y_i\}$ $conf(B_m)=\frac{1}{|B_m|}\sum_{i\in B_m}\hat{p}_i$

$ECE=\sum_{m=1}^{M}\frac{|B_m|}{n}|acc(B_m)-conf(B_m)|$.

> Maximum Calibration Error (MCE).

$MCE=\max_{m\in\{1,\dots,M\}}|acc(B_m)-conf(B_m)|$.

Robustness.

Handle DS w/o lots of data, domain knowledge.

① Domain Adaptation.

assumption: abundant data in Source Domain, few data from target domain.

Approaches: ① Importance weighting. ✗ (Similar ones get higher weights).

$$E_{target}[\ell(\theta; x, y)] = E_{source}\left[\frac{P_{target}(x,y)}{P_{source}(x,y)}\ell(\theta; x, y)\right].$$

unsupervised domain adaptation: no y from target domain

assume covariate shift: $P_{target}(y|x) = P_{source}(y|x)$.

② Invariant feature learning (want features < informative ⇒ predict y. / invariant → same distribution for S & T).

domain discriminators:

X ⟶ features ⟶ class label y.

gradient Reversal Layer ⟶ domain label d.

Invariant Risk Minimization (IRM) optimal classifiers are same.

directly make feature distribution similar

Correlation Alignment (CORAL). - Add a loss term.

② Subpopulation shift.

Assumption: Not only S≠T. Some domains are underrepresented in training data, but at test time, we want fairness / weight / worst-case ("Robustness").

in distribution (group) robustness. Adversary can only change distribution across domains, not distribution of classes inside domain. $\min_{\theta} \max_{p_1 \cdots p_D} \sum_{d=1} p_d E_d[\ell(\theta; x, y)]$.

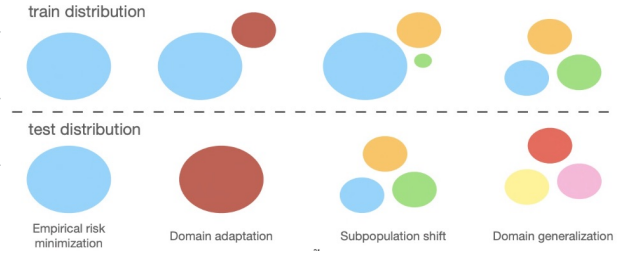
rebalancing the training data (upsampling / downsampling).

③ Domain Generalization. (One-shot domain Adaptation / Multi-Source domain Adaptation).

Assumption: >2 domains; No domain imbalance; will be given new domain at the test time ⇒ want to generalize.

Summary: different assumptions on training, expected test shift.

"domains" ⇒ share.



Test-time-adaption: change prediction at test time. e.g. threshold, rotation.

assume multiple test points available.

Adaptive model: $X \cdot P_X \rightarrow y$. ($P_X$: distribution of where X comes from.)

Not distribution shift.

Adversarial Robustness.

modern adversarial distortions: perceptible to human eyes but not perceptible to models.

ℓ_1 Norm: $\|V\|_1 = |V_1| + |V_2| + \cdots + |V_d|$

$\|V\|_2 = \sqrt{V_1^2 + \cdots + V_d^2}$

$\|V\|_\infty = \max\{|V_1|, |V_2|, \cdots, |V_d|\}$. greatest change among dimensions.

cumulative effect of many small changes made the adversary powerful. (change class of $\hat{y}$).

distortion budget ε s.t. $\|X_{adv} - X\|_p \leq \varepsilon$. ℓ_p attacks.

Adversary's goal: find distortion δ s.t. $X_{adv} = X + \underset{\delta : \|\delta\|_p \leq \varepsilon}{\arg\max}\, \ell(\theta; X+\delta, y)$.

Fast Gradient sign Method. (FGSM)

$X_{FGSM} = X + \varepsilon \operatorname{sign}(\nabla_X \ell(\theta; X, y))$ $\|X_{FGSM} - X\|_\infty = \varepsilon$. Single step GD.

Projected Gradient Descent (PGD)

Random initialize a perturbed image $\tilde{x} = X + n$, $n_i \sim \mathcal{U}[-\varepsilon, \varepsilon]$, initialize $\delta = 0$.

for $t = 1, \ldots, T$: $\delta \leftarrow \operatorname{clip}(\delta + \alpha \operatorname{sign}(\nabla_\delta \ell(\theta; \tilde{x} + \delta, y)), -\varepsilon, \varepsilon)$

$X_{PGD} = \tilde{x} + \delta$.

Method 1: Adversial Training. (AT). will reduce accuracy on non-adversarial examples.

sample batch $(X^{(1)}, y^{(1)}), (X^{(2)}, y^{(2)}) \ldots (X^{(B)}, y^{(B)})$ in training set.

Create $X^{(i)}_{adv}$ (OR $X^{(i)}_{PGD}$) from $X^{(i)}$ for all i.

optimize average training loss on them.

Method 2: More data.

Adversarial robustness scales slowly with dataset size.

✓ adversarial pretraining on larger dataset.

Method 3: Data Augmentation + adversial training + parameter exponential moving average => CutMix. (MixUp + CutOut).

untargeted attack. — maximize the loss $\arg\max_{\delta:\|\delta\|_p\le\varepsilon} \ell(\theta; x+\delta, y)$

targeted attack. — optimize examples to be misclassified as predetermined $\tilde{y}$.
$\arg\min_{\delta:\|\delta\|_p\le\varepsilon} \ell(\theta; x+\delta, \tilde{y})$.

"arms race" that defenders lose; within weeks.

transferability: models with different architecture. Real-world instantiation noise, sensor noise.

Smooth GELU instead of ReLU.

robustness to unforeseen attacks.

Deep generative Models.

① generation: synthesize new data points.
(conditional generation).

② Density Modeling.

③ Representation Learning. latent representation z.

Generative adversarial networks. (GANs) ①③ no need to compute $p_\theta(x)$.

generator G — synthesize data

vs.

discriminator D. — distinguish real data x from synthesized data $\tilde{x}$. (throw away afterwards).
(binary.)

$\min_G \max_D E_{x\sim p_{data}}[\log D(x)] + E_{z\sim p(z)}[\log(1-D(G(z)))]$. $\tilde{x}=G(z)$ w/ $p_G(x)$

optimal G^* under Bayes optimal D^* recreates the data distribution.

$D^*(x) = \frac{p(x)}{p(x)+p_G(x)} \Rightarrow \min_G E_{x\sim p(x)}[\log(\frac{p(x)}{p(x)+p_G(x)})] + E_{x\sim p_G(x)}[\log(\frac{p_G(\tilde{x})}{p(\tilde{x})+p_G(\tilde{x})})]$.

let $q(x) = \frac{p(x)+p_G(x)}{2}$ $\min_G \underbrace{E_{x\sim p(x)}[\log p(x) - \log q(x)]}_{D_{KL}(p(x)\|q(x))} + \underbrace{E_{\tilde{x}\sim p_G(\tilde{x})}[\log p_G(\tilde{x}) - \log q(\tilde{x})]}_{D_{KL}(p_G(x)\|q(x))} + \text{constant}$.

JS divergence between $p(x)$ and $p_G(\tilde{x})$. => generator distribution to close data distribution.

better architecture, modified objectives, bags of tricks, scaling up.

Conditional generation: class label OR "label-like" input (InfoGAN) to both G and D

cycleGan: 2G, 2D. turns image from one domain to another, reverse.

alternating taking GD on G and D. hard to evaluate (pretrained classifier)

In practice, care about whether D provides a useful signal to G. => restrict D's expressivity.

↑ Smooth: ① Lipschitz constraint. ② real-valued D. ③ Instance noise

$W(P_r, P_G) = \sup_{\|f\|_L \le 1} E_{x\sim P_r}[f(x)] - E_{x\sim P_G}[f(x)]$. f: 1-Lipschitz

① weight clipping. (each entry of weight $|\le \varepsilon$). K-Lipschitz.

② gradient penalty. $+\lambda(\|\nabla_x f(x)\|_2 - 1)^2$ ~1-Lipschitz.

③ spectral Normalization. ReLU → 1-Lipschitz. $k = \sup_{\|x\|_2=1} \|W_i x\|_2 = \sigma(W_i)$ (largest singular value of W_i) $W_i \leftarrow \frac{W_i}{\sigma(W_i)}$

Autoregressive Models ①, ②.

$P_\theta(x) = \prod_{i=1}^{d} P_\theta(x_i | x_{<i})$ masked convolution. / masked SA. pixels → mixture of logistic distributions

very slow.

Latent Variable Models

known: $p(z)$ (unit Gaussian), $p_\theta(x|z)$. intractable: $p(x) = \int p(z) p_\theta(x|z) dz$. $p(z|x)$.

model $p_\theta(z|x)$ by $q_\phi(z;x)$. ⇒ minimize KL divergence.

$D_{KL}[q_\phi \| p_\theta] = E_{q_\phi}[\log q_\phi(z;x) - \log p_\theta(z|x)] = E_{q_\phi}[\log q_\phi(z;x) - \log p_\theta(x;z)] + \log p_\theta(x)$.

$= E_{q_\phi}[-\log p_\theta(x|z) + \log q_\phi(z;x) - \log p(z)] + \log p_\theta(x) = \underbrace{-[E[\log p_\theta(x|z)] - D_{KL}(q_\phi \| p_z)]}_{\text{ELBO. lower bound}} + \underbrace{\log p_\theta(x)}_{\text{reconstruction + regularization}} \ge 0$ ($\times \phi$).

objective: $\arg\max_\theta \sum_{i=1}^{n} \log p_\theta(x_i) = \arg\max_\theta \sum_{i=1}^{n} \log\left(\sum_z p(z) p_\theta(x_i|z)\right)$.

by $\log E[x] \ge E[\log(x)]$, $\ge \sum_{i=1}^{n} E_{q(z|x_i)}[\log p(z) - \log q(z|x_i) + \log p_\theta(x_i|z)]$.

Re-parametrization Trick: $q_\phi(x) \sim N(0,1)$ $E_q(x^2) = E_p[(\theta+\varepsilon)^2]$. $\varepsilon \sim N(0,1)$. $p \sim N(0,1)$.

> Variational autoencoders (VAEs). ① ③ ② → ELBO. (tradeoff.)

observation model $p_\theta(x|z)$ (decoder), recognition model $q_\phi(z;x)$ (encoder) $\xrightarrow{\text{minimize}}$ ELBO

variational inference (VI) framework. keep both.

blurry. ← conditional independence assumption.

> deep autoencoders. (eg. denoising, bottleneck).

> masked autoencoders. ③. little ①.

Diffusion Models

fix forward process (x to z): add Gaussian noise, little by little ⇒ z (pure noise).

reverse process: (z to x).

maximizing ELBO.

e.g. DALL·E 2.

deep generative models
Self-supervised. > deep unsupervised learning. ↳ √: scaling up.
↳ target representation learning.

pretend a part of input is unknown and predict it. e.g. BERT; MAE

① leverage known structure of images. ② contrastive learning. ③ data augmentation.
leveraging spatial context; predicting rotations. 2 patches; relative pos

Contrastive Learning. learned representation should be close to "similar" $\ell = -\log \frac{\exp(z^T z_+ / \tau)}{\sum_{i=1}^{k} \exp(z^T z_i / \tau)}$ similar.

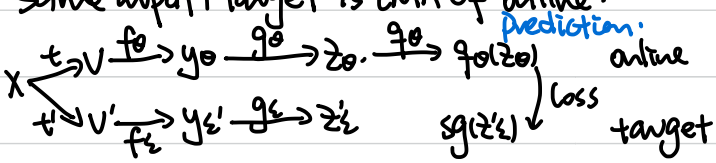
Contrastive predictive coding (CPC): similar -> comes from same input; extracting.

Similar: diff. augmentation of the same image. additional transformation representation.
SimCLR. on $g(\cdot)$ not $f(\cdot)$

Momentum Contrast (MoCo): queue of negatives (add representation from the latest mini batch.).
EMA; encourages similarity across minibatch.

tednical improvements => SimCLR. $\ell_{i,j} = -\log \frac{\exp(\text{sim}(z_i, z_j)/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{k \neq i} \exp(\text{sim}(z_i, z_k)/\tau)}$

Bootstrap your own latent (BYOL): online vs. target network; given two diff. augmentation of same input; target is EMA of online.



degenerate solution.

Self-distillation with no labels (DINO): student -> teacher's prob; "distillation": teacher knows sth. nothing.

Multimodal contrastive learning (CLIP): text encoder & image decoder; similar: (text, image).

Massive Models (transformer decoder language models).

Scaling laws: larger models require fewer samples. ; 8x model size, 5x dataset size.
(neural language models).
train large models, stop short after convergence.

GPT-3: few-shot learning. #examples.

Gopher: 280B. 2T->300B.

Chinchilla: varying hyperparameters; x model = x data. ; Massive Multitask Language Understanding (MMLU)

Megatron-Turing NLG: 530B. 339B->270B.

PaLM: 540B; 780B -> 780B.

chain-of-thought prompting. <10 ; self-consistency. (sampling + pick answer).
↓ in-context learning.
process to achieve answer.

Fine Tuning: update via gradient based optimization with small datasets.

↓

specializing code: Codex. AlphaCode.

scaling up model / dataset size; change architecture to encoder-decoder; fine tune on competition code; sample/filter sols.

Meta-Learning.

assumption: meta-training task & meta-test task drawn i.i.d. from same distribution.

Input: D^{tr}, X^{ts} Output: y^{ts}.

- Multi-task learning. – solve $T_1, \dots, T_T$ at once.
- transfer learning. – solve T_b after solving T_a, transfer knowledge.
- meta-learning – given data $T_1, \dots, T_n$, quickly solve T_{test}.] impractical to prior tasks.

T_i: support set / query set.

K-shot learning, N-way classification.

① Black-Box Meta-learning.

Parametrize learners as NN. + expressive. – optimization not consistent.

② Optimization-based Meta-learning.

embed optimization. – 2nd optimization compute. + consistent, positive inductive bias.

③ Non-parametric

+ expressive. compute

>1 shot: independently, aggregate by $c_n = \frac{1}{K} \sum_{(x_i, y_i) \in D_i^{tr}} 1(y=n) f_\theta(x)$. $P(y=n|x) = \frac{\exp(-d(f_\theta(x), c_n))}{\sum_{n'} \exp(d(f_\theta(x), c_{n'}))}$

meta-training: sample task T_i.

binary classification: Sample disjoint D_i^{tr}, D_i^{test} from D_i.

Compute $\phi_i \leftarrow f_\theta(D_i^{tr})$ $\hat{y}^{ts} = \sum_{x_k, y_k \in D^{tr}} f_\theta(x^{ts}, x_k) y_k$.

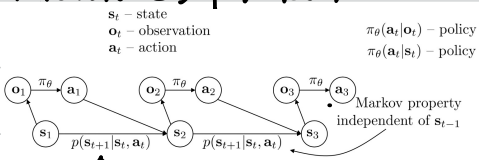
update θ by $\nabla_\theta L(\phi_i, D_i^{test})$. $\nabla_\theta L(\hat{y}^{ts}, y^{ts})$.

meta-testing: given D_j^{tr} from new task T_j.

N-way classification. Compute $\phi_j \leftarrow f_\theta(D_j^{tr})$.

Deep-learning.

Moravec's paradox.



transition operator / probability.

Behavioral cloning.

DAgger.

1. train $\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$ from human data $\mathcal{D} = \{\mathbf{o}_1, \mathbf{a}_1, \ldots, \mathbf{o}_N, \mathbf{a}_N\}$
2. run $\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$ to get dataset $\mathcal{D}_\pi = \{\mathbf{o}_1, \ldots, \mathbf{o}_M\}$
3. Ask human to label $\mathcal{D}_\pi$ with actions $\mathbf{a}_t$
4. Aggregate: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_\pi$

goal:

$$\underbrace{p_\theta(\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T)}_{p_\theta(\tau)} = p(\mathbf{s}_1) \prod_{t=1}^{T} \pi_\theta(\mathbf{a}_t|\mathbf{s}_t) p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$$

$$\theta^\star = \arg\max_\theta E_{\tau \sim p_\theta(\tau)} \left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right]$$

Q-learning:

RL objective: $\max_\pi \sum_{t=1}^{T} E_{\mathbf{s}_t, \mathbf{a}_t \sim \pi}[r(\mathbf{s}_t, \mathbf{a}_t)]$

Q-function: $Q^\pi(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{T} E_{\mathbf{s}_{t'}, \mathbf{a}_{t'} \sim \pi}[r(\mathbf{s}_{t'}, \mathbf{a}_{t'})|\mathbf{s}_t, \mathbf{a}_t]$

$\pi(\mathbf{a}|\mathbf{s}) = 1$ if $\mathbf{a} = \arg\max_{\mathbf{a}} Q^\pi(\mathbf{s}, \mathbf{a})$

$Q^\star(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \max_{\mathbf{a}'} Q^\star(\mathbf{s}', \mathbf{a}')$

enforce this equation at all states!

minimize $\sum_i (Q(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \max_{\mathbf{a}'_i} Q(\mathbf{s}'_i, \mathbf{a}'_i)])^2$

minimize $\sum_i (Q(\mathbf{s}_i, \mathbf{a}_i) - y_i)^2$

$Q(\mathbf{s}, \mathbf{a}) \leftarrow r(\mathbf{s}, \mathbf{a}) + \max_{\mathbf{a}'} Q(\mathbf{s}', \mathbf{a}')$

off-policy Q-learning:

1. collect dataset $\{(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)\}$ using some policy, add it to $\mathcal{B}$
2. sample a batch $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$ from $\mathcal{B}$ — $K\times$
3. minimize $\sum_i (Q(\mathbf{s}_i, \mathbf{a}_i) - [r(\mathbf{s}_i, \mathbf{a}_i) + \max_{\mathbf{a}'_i} Q(\mathbf{s}'_i, \mathbf{a}'_i)])^2$