NO.

DATE

CS182 Note & Review

Optimization: Implicite Regulation of GD; SGD; and Momentum

(x, y) data, need model $f_\theta(x) \to y$

$\hat{\theta} = \arg\min_\theta \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(y_i - f_\theta(x_i))$

Assume $\mathcal{L}(\vec{w}) = \|X\vec{w} - y\|_2^2$, then $\vec{w}^* = (X^TX)^{-1}X^Ty$

$\vec{w_{t+1}} = \vec{w_t} - \eta \nabla_w \mathcal{L}(w, \vec{x_{train}}, \vec{y_{train}})$ (GD)

$\vec{w_{t+1}} - \vec{w_t} = -2\eta X^T(Xw - y)$ ($(Xw-y)^T(Xw-y)$ 对 w 导)

Want to rewrite into: $\vec{w_{t+1}} - \vec{w^*} = ? \cdot (\vec{w_t} - \vec{w_*})$

以衡量 $\vec{w_t} \to \vec{w^*}$ 距离在下降/上升/如何变换

$\vec{w_{t+1}} - \vec{w^*} = (I - 2\eta X^TX)(\vec{w_t} - \vec{w^*})$

∴ We want $(I - 2\eta X^TX)^n$ converge when $n \uparrow$

Consider SVD for $I - 2\eta X^TX$, then the largest eigenvalue in Σ shouldn't be greater than 1

∴ $|1 - 2\eta\lambda_{max}(X^TX)| < 1$ $\quad \eta < \frac{1}{\lambda_{max}(X^TX)}$

⇒ 学习率 (or step size) 不可过大!

☆ 同时也暗示着: singular values & spread of singular values matter

Why? 不妨 $X = U\Sigma V^T$ 代入 $\vec{w_*}$ (with ridge considered!)

$\vec{w_*} = (V\Sigma^T\cancel{U^TU}\Sigma V^T + \lambda I)^{-1} V\Sigma^TU^T\vec{y}$

$= V(\Sigma^T\Sigma + \lambda I)^{-1}\cancel{V^TV}\Sigma^TU^T\vec{y}$

$= V(\Sigma^T\Sigma + \lambda I)\Sigma^TU^T\vec{y}$

$= V\begin{bmatrix} \sigma_1^2+\lambda & & \\ & \sigma_2^2+\lambda & \\ & & \ddots \end{bmatrix}\Sigma^TU^T\vec{y} = \sum_{i=1}^{n} \vec{v_i}\frac{\sigma_i}{\sigma_i^2+\lambda}\vec{u_i}^T\cdot\vec{y}$

可见 $\lambda \ll \sigma_i$ 则等于不正则化; $\lambda \gg \sigma_i$, 全为0, 优化目标只与 $\|w\|_2^2$ 有关了

因此提供了从奇异/特征值解释正则化的方向

带有Ridge正则化项的优化称为"岭回归"

NO.

DATE

※: Regularization ⟺ Weight Decay

在GD中, $\vec{w}_{t+1} = \vec{w}_t - 2\eta X^T(X\vec{w}_t - \vec{y})$. remains in the span of data

GD for ridge: $\vec{w}_{t+1} = (1-2\eta\lambda)\vec{w}_t - 2\eta X^T(X\vec{w}_t - \vec{y})$ ※

weight decay

奇异/特征值的威力还可更直观一些:

$\vec{w}_{t+1} = \vec{w}_t - 2\eta V\Sigma^T U^T(X\vec{w}_t - \vec{y})$ 左右同乘 V^T: 令 $\begin{cases} V^T\vec{y} = \vec{\tilde{y}} \\ \vec{\tilde{w}} = V\vec{w} \end{cases}$

$\vec{\tilde{w}}_{t+1} = \vec{\tilde{w}}_t - 2\eta\Sigma^T(\Sigma\vec{\tilde{w}}_t - \vec{\tilde{y}})$, $\tilde{w}_{t+1,i} = \tilde{w}_{t,i} - 2\eta\sigma_i[\sigma_i\cdot\tilde{w}_{t,i} - \tilde{y}_i]$

可见 σ_i 越小, i-th 维度越不重要, 更新越慢

这可视为一种 "early stopping", 因为GD在奇异值大的方向上对数据拟合, 拟合好之后, 奇异值小的几乎不再拟合

从岭回归的 $\frac{\sigma_i}{\sigma_i+\lambda}$, 若 σ_i 很小, 防止原本的 $\frac{1}{\sigma_i}$ 会很大; 到GD中的 σ_i 越小, 对应方向更新速率越小。上述两种都认为是 Implicit Regulation

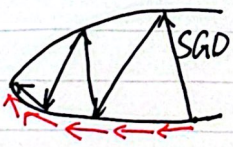
SGD: GD对凸函数很好, 但也很 expensive, 因为一次更新要全部数据点都 inference 一变。

Add some noise to gradient direction! In expectation, it descends:

$\theta_{t+1} = \theta_t - \eta\frac{1}{B}\sum_{i=1}^{B}\nabla f_i(\theta_t)$ B: mini batch, batch size

SGD 既 work, 还有利于跳出 local minima & saddle points, 且 less cost!

但训练中的 oscillation 会难搞:



How to smooth it?

如 batch size ↑, low-pass filter ···

此及 Momentum!

Momentum: $\begin{cases} \vec{w}_{k+1} = \vec{w}_k - \eta\vec{\pi}_{k+1} \\ \vec{\pi}_{k+1} = \beta\vec{\pi}_k + (1-\beta)\nabla f(\vec{w}_k), \ \pi_0 = 0, \ \beta\in[0,1] \end{cases}$

相当于梯度作damping!

⇒ Add history of 梯度, 并延 history 时间 decay weight exponentially

$\pi_n = \beta^{n-1}(1-\beta)\nabla f(w_0) + \cdots = \sum_{i=1}^{n}\beta^{i-1}(1-\beta)\nabla f(w_{n-i})$

NO.

DATE

Adam (Who is Adam?)

GD在 singular value 大的方向上，step 更大。优点在于 Implicit Regularization

缺点在于：η，learning rate，非常难选，因为要兼顾奇异值大的方向，梯度大，反之很小 $\Rightarrow$ normalize the gradients

对梯度向量除以模长，使之变为单位 vector

$$\vec{w}_{t+1} = \vec{w}_t - \eta \frac{\nabla_{\vec{w}} \mathcal{L}(\vec{w})}{\|\nabla_{\vec{w}} \mathcal{L}(\vec{w})\|_2 + \varepsilon}$$ → 2范数且开正根

但这不是"方向奇异值越大，变化越快"的最优算法！$\Rightarrow$ 用完不要梯度大小信息！

Instead: $\vec{m}_t \leftarrow \nabla_{\vec{w}} \mathcal{L}(\vec{w})$ $\vec{v}_t \leftarrow (\nabla_{\vec{w}} \mathcal{L}(\vec{w}))^2$ ↓ element-wise square

$$\vec{w}_{t+1} = \vec{w}_t - \eta \frac{\vec{m}_{t+1}}{\sqrt{\vec{v}_{t+1}} + \varepsilon}$$ → $\varepsilon > 0$, avoid division by zero

而 $\frac{\vec{m}_{t+1}}{\sqrt{\vec{v}_{t+1}}} = \mathrm{sgn}(\nabla_{\vec{w}} \mathcal{L}(\vec{w}))$，"sign SGD"

但也有 issue: Even we are close, it keeps overshooting / bouncing

$\Rightarrow$ smooth

$$\begin{cases} \vec{m}_{t+1} = \beta \vec{m}_t + (1-\beta) \nabla_{\vec{w}} \mathcal{L}(\vec{w}_t) \\ \vec{v}_{t+1} = \delta \vec{v}_t + (1-\delta) [\nabla_{\vec{w}} \mathcal{L}(\vec{w}_t)]^2 \\ \vec{w}_{t+1} = \vec{w}_t - \eta \frac{\vec{m}_{t+1}}{\sqrt{\vec{v}_{t+1}} + \varepsilon} \end{cases}$$

且 $\begin{cases} \tilde{\vec{m}}_{t+1} = \frac{\vec{m}_{t+1}}{1-\beta^{t+1}} \\ \tilde{\vec{v}}_{t+1} = \frac{\vec{v}_{t+1}}{1-\delta^{t+1}} \end{cases}$，Adam: $\vec{w}_{t+1} = \vec{w}_t - \eta \frac{\tilde{\vec{m}}_{t+1}}{\sqrt{\tilde{\vec{v}}_{t+1}} + \varepsilon}$

Adam 常用于 SGD，故 $\tilde{\vec{m}}$，$\tilde{\vec{v}}$ 由 mini-batch 计算得到

Normalization (统一化，非归一化) helps with exploding / vanishing gradients

But: size of the step is largely dictated by the learning rate.

而 AdamW: Adam with weight decay:

$$\vec{w}_{t+1} = (1-\lambda) \vec{w}_t - \eta \frac{\tilde{\vec{m}}_{t+1}}{\sqrt{\tilde{\vec{v}}_{t+1}} + \varepsilon}$$

NO.

DATE

Feature perspective and Taylor Expansion

Network from another perspective: $f(\vec{x}, \vec{\theta})$ Δ: f可视为 loss function

And further regard it as a function of $\vec{\theta}$:

Consider $f: R^m \to R$ (like regression / binary classification)

Taylor Expansion from current x, θ: change, 且负!! omit.

$$f(x, \theta_0+\Delta\theta) = f(x, \theta_0) + \left[\left\langle \nabla_\theta f\Big|_{\theta=\theta_0}, \Delta\theta \right\rangle\right] + [\cdots \text{higher term}]$$

$\nabla_\theta f|_{\theta=\theta_0}$ 为 f 在 θ_0 时对 θ 的梯度，$\langle\cdot,\cdot\rangle$ 为内积

这个近似的核心前提：$\Delta\theta$ 很小！ (converge fast). hold true

因此陷入 dilemma: 欲 $\Delta\theta$ 大以 $\langle \nabla_\theta f|_{\theta=\theta_0}, \Delta\theta\rangle$ 太小，又 $\Delta\theta$ 须小以泰勒近似

⇒优化问题: want: $\|\Delta\theta\|_? \le \eta$，可以是2范数等。这里假设 $?=\infty$

i.e., $\|\Delta\theta\|_\infty = \max_j |\Delta\theta[j]|$

在此优化约束下，优化: $\underset{\|\Delta\theta\|_\infty \le \eta}{\text{argmin}} \frac{1}{n}\sum_{i=1}^{n} \left\langle \nabla_\theta f\Big|_{\substack{x=x_i \\ \theta=\theta_0}}, \Delta\theta \right\rangle$

$$= \underset{\|\Delta\theta\|_\infty \le \eta}{\text{argmin}} \left\langle \frac{1}{n}\sum_{i=1}^{n} \nabla_\theta f\Big|_{\substack{x=x_i \\ \theta=\theta_0}}, \Delta\theta \right\rangle \quad \text{内积拆开}$$

$$= \underset{\forall j\, |\Delta\theta[j]| \le \eta}{\text{argmin}} \sum_j \left[\left(\frac{1}{n}\sum_{i=1}^{n} \nabla_\theta f\Big|_{\substack{x=x_i \\ \theta=\theta_0}}\right)[j] \cdot \Delta\theta[j] \right]$$

⇒ notation: $\nabla_{\theta_0} f_{batch}$

可见，可分别优化: $\underset{-\eta \le \Delta\theta[j] \le \eta}{\text{argmin}} \nabla_{\theta_0} f_{batch}[j] \cdot \Delta\theta[j]$

$\nabla_{\theta_0} f_{batch}$ 与 $\Delta\theta$ 无关，可见 $\Delta\theta[j]$ 取值实则很简单.

$$\Delta\theta = \begin{cases} +\eta & \text{if } \nabla_{\theta_0} f_{batch} < 0 \\ -\eta & \text{if } \nabla_{\theta_0} f_{batch} > 0 \end{cases} \Rightarrow \vec{\Delta\theta}^* = -\eta\, \text{sign}(\nabla_\theta f_{batch})$$

这就是 Adam 算法的核心！

NO.

DATE

Initialization & Optimizer

标准(归一)化 (standardization) 在ML / DL中很重要!

- 不同feature, 不同 numerical magnitude (else large features dominate)

Consider ReLU: scalar case:

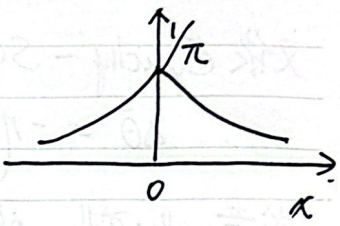
elbow: $-\frac{b}{w}$, say, $b \sim N(0,1)$ $w \sim N(0,1)$

则 $-\frac{b}{w}$ ~ Cauchy Distribution

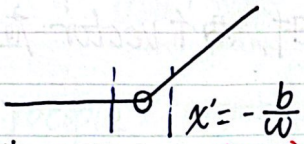
Δ: 柯西分布定义: x_0: 位置参数, 峰值中心位置; γ: 尺度参数, 分布离散程度

$$f(x) = \frac{1}{\pi \gamma \left[1 + \left(\frac{x-x_0}{\gamma}\right)^2\right]}$$

当 $x_0 = 0$, $\gamma = 1$ 时, 得标准柯西分布 $\Rightarrow f(x) = \frac{1}{\pi(x^2+1)}$



Recall ReLU中:



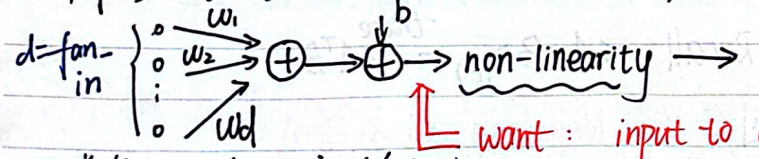
若是input 分布在蓝区间中, 能充分利用 non-linearity

故若 input 不归一化, 则模型表达能力严重下降

故 input 归一化很重要, 对应地, model参数初始化也很重要!

How to initialize? 若0初始化, gradients are zero, weights never move

所以 how to do? 注意到, 上一层的 activated output 是下一层输入



want: input to it is "standardized"

则考虑: b: 0初始化*; $w_i \sim N(0, 1/d)$ ⟹ Xavier 初始化

Why? $\mathrm{var}\left[\sum_i w_i h_i\right] = E\left[(\sum w_i h_i)^2\right] - \left[E(\sum w_i h_i)\right]^2 \to 0$

$$= \sum_{i=1}^{d} E(w_i^2)\, E(h_i^2) \quad \text{(due to i.i.d)}$$

2倍!

还有He初始化: Relu导致方差减半 ⇒ w的初始化预先把权重方差放大

$w_i \sim N(0, 2/d)$

* bias初始化不关0初始化, 还可以:
- use small number
- b合并至W, 视为d+1维, 然后全都 Xavier ($\sim N(0, 1/d+1)$)

↳ 当然, 它更受欢迎

NO.
DATE

* 通过调整不同层的初始化方差和学习率缩放，来解耦"特征学习"动态与"Lazy training"动态。在一个小模型上找到最优学习率，且可直接应用于一个任意宽的大模型上（小宽度→大宽度）

注意到：初始化时，$W(b)$ 采用随机 initialization，而在 Adam 中，因为 Taylor 近似视角，$\Delta\theta$ 不能很大

⇒ 意味着 models barely change from random initialization

这被视为一种 lazy training（NTK, Neural Tangent Kernel），且随机初始化不应视为能总是 work ⇒ 即神经网络的 power 还未全部使用

在之前的"$\Delta\theta$ 不能很大"中，用 ∞ 范数 — 若 L_2 范数呢？

$\|\Delta\vec{\theta}\|_2 \le \eta$，欲 $\underset{\|\Delta\theta\|_2\le\eta}{\operatorname{argmin}} \langle \nabla_{\vec{\theta}}\mathcal{L}(\vec{\theta}), \Delta\vec{\theta}\rangle$

注意到：$\langle x, y\rangle = x^T y = \|x\|_2\|y\|_2\cos\varphi$，$\cos\varphi$ 应尽可能 -1

又依 Cauchy-Schwartz：$|\langle\cdot,\cdot\rangle|$ 最大时，两个 vector 应对齐（align）

$$\therefore \Delta\vec{\theta} = -\eta\frac{\nabla_{\theta}\mathcal{L}(\vec{\theta})}{\|\nabla_{\theta}\mathcal{L}(\vec{\theta})\|_2}$$

或者，$\|\Delta\vec{\theta}\|_2$ 的约束以正则化体现：$\underset{\vec{\theta}}{\operatorname{argmin}} \underbrace{\langle \nabla_{\theta}\mathcal{L}(\vec{\theta}), \Delta\theta\rangle + \lambda\|\Delta\theta\|_2^2}_{g(\theta),\ \text{convex}}$

$$\nabla g(\Delta\theta) = \nabla_{\theta}\mathcal{L}(\vec{\theta}) + 2\lambda\Delta\vec{\theta},$$

$$\Delta\vec{\theta}^* = -\underbrace{\frac{1}{2\lambda}}_{\text{step size}}\nabla_{\theta}\mathcal{L}(\vec{\theta}) \rightarrow \text{gradient descent}\ (\text{只考虑一层})$$

之前的 $\vec{\theta}$ 总是 vector，但真正的 network 不只一层 ⇒ matrix!

$\underset{\|\Delta W\|\le\eta}{\operatorname{argmin}} \langle \nabla_W\mathcal{L}(W), \Delta W\rangle$，Recall $\langle A, B\rangle_{(F)} \overset{\text{trace}}{=} (A^T B)$.

设 $\nabla_W\mathcal{L}(W) = U\Sigma V^T$，则依 Von Neumann 迹不等式：

$$\Delta W^* = -\eta U V^T$$

这便是 Shampoo 优化器！同过上述三例，we have a recipe*:

- Given a norm, make an optimizer 七 - examples for $\|\cdot\|_2$, $\|\cdot\|_\infty$

（下一讲内容预告）

Additional: RMSnorm: $\|x\|_{RMS} = \frac{1}{\sqrt{d_{in}}}\|x\|_2$ 诱导范数

Matrix version: Recall: induced Matrix-Norm: $\|A\|_{\alpha\to\beta} = \max_{\|x\|_\alpha=1}\|Ax\|_\beta$ *

$\therefore$ RMS→RMS Norm: $\|A\|_{RMS\to RMS} = \max_{\|x\|_{RMS}=1}\|Ax\|_{RMS} = \max_{\|x\|_2=\sqrt{d_1}}\frac{1}{\sqrt{d_2}}\|Ax\|_2 = \frac{\sqrt{d_1}}{\sqrt{d_2}}\max_{\|x\|_2=1}\|Ax\|_2$

* 谱范数（Spectral）：衡量一个矩阵能对另一个向量产生的最大拉伸程度

↳ 其实就是诱导范数 of L_2

△: $\|A\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|A_{ij}|^2} = \sqrt{tr(A^TA)} = \sqrt{\sum_{i=1}^{\min(m,n)}\sigma_i^2}$

*: RMS→RMS Norm is rescaling of the spectral norm

No.

Date

RMS Norm & Maximal update normalization (μP)

Recall, induced matrix norm: for matrix, the def of its spetrum norm (like L_2)

$$\|A\|_{\alpha\to\beta} = \max_{\|\vec{x}\|_\alpha=1} \|A\vec{x}\|_\beta$$

↓ for vector, we know their norm!

RMS→RMS Norm ↔ Motivated by Xavier Initialization:

$$\|A\|_{RMS\to RMS} = \max_{\|\vec{x}\|_{RMS}=1} \|A\vec{x}\|_{RMS}$$

And $\|\vec{x}\|_{RMS} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}x_i^2}$, $RMSNorm(\vec{x}) = \frac{\vec{x}}{\|\vec{x}\|_{RMS}}$

$$= \frac{1}{\sqrt{n}}\|\vec{x}\|_2 \quad *$$

✡: $\vec{x}\in\mathbb{R}^{d_{in}}$, $A\in\mathbb{R}^{d_{out}\times d_{in}}$

$$\therefore \|A\|_{RMS\to RMS} = \sqrt{\frac{d_{in}}{d_{out}}}\max_{\|\vec{x}\|_2=1}\|A\vec{x}\|_2 = \sqrt{\frac{d_{in}}{d_{out}}}\|A\|_2$$

Recall: optimizer recepie: $\operatorname{argmin}_{\|\Delta W\|\le\eta} \langle \nabla_W\mathcal{L}(W), \Delta W\rangle$

→ choose norm for → choose η ⇒ get optimizer

In optimizer like Adam, we've tried $\|\cdot\|_\infty$ and $\|\cdot\|_2$. what about $\|\cdot\|_{RMS}$?

$\Delta W\in\mathbb{R}^{d_1\times d_2}$, then $\|\Delta W\|_{RMS}\le\eta \Rightarrow \|\Delta W\|_2 \le \sqrt{\frac{d_1}{d_2}}\cdot\eta$

Then we can adjust the learning rate dynamically

Different Approach: Commit to SignSGD/Adam:

$$W_{t+1} = W_t - \eta\,\operatorname{sign}(\nabla_W\mathcal{L}(W))$$

Rethink about step size!! A fixed η is not reasonable

Cause in different layers of different archietectures, W are different!

Want η adjust in accordance with W shape ⇒ RMS Norm can help

↳ Remains in span of minibatch data

If SGD, $\nabla_W\mathcal{L}(W)$ is low rank, suppose $\nabla_W\mathcal{L}(W) = \sigma\vec{u}\vec{v}^T$ (SVD)

Then $\|\nabla_W\mathcal{L}\|_2 = \sigma$, $\|\nabla_W\mathcal{L}\|_F = \sigma$, def $\operatorname{sign}(\nabla_W\mathcal{L}(W)) = S$

$\therefore S = \operatorname{sign}(\sigma\vec{u}\vec{v}^T) =^* \operatorname{sign}(\vec{u})\cdot\operatorname{sign}(\vec{v})^T \Rightarrow rank(S)=1$

*: $\operatorname{sign}(mn) = \operatorname{sign}(m)\cdot\operatorname{sign}(n)$

$\Rightarrow \|\nabla_W\mathcal{L}\|_2^2 = \sigma^2$ $\|\nabla_W\mathcal{L}\|_F^2 = tr(\sigma^2\vec{u}\vec{v}^T\vec{v}\vec{u}^T) = \sigma^2$

$W_{t+1} = W_t - \underbrace{\eta\cdot S}_{\Delta W}$, we want ΔW small to ensure taylor approximation

KOKUYO

No.

Date

Now we choose $\|\Delta W\|_{RMS\to RMS} \le \delta$ ← step size threshold

$\therefore \eta \cdot \sqrt{\frac{d_{in}}{d_{out}}} \|S\|_2 \le \delta$ ← learning rate , $\|S\|_2$?, S has all +1 or -1, and rank is 1!

Intuitively and in fact, $\|S\|_2 = \sqrt{d_{in} d_{out}}$, $\eta \le \frac{1}{d_{in}} \delta$

∴ If using RMS norm, learning rate can scale with layer!

⇒ Adam / signSGD lr can scale as $\frac{1}{d_{in}}$ to maintain $\|\Delta W\|_{RMS\to RMS} \le \delta$

Essence: μP optimization: maximal update parameterization

μP: How to change hyperparameter when NN width changes?

Can we optimize hyperparameter on smaller NN and 'transfer' to larger ones?

E.g. $\frac{1}{\sqrt{n}}(X_1 + \cdots + X_n) \overset{n\to\infty}{\sim} N(0,1)$, $X_i \sim N(0,1)$. but:

$\frac{1}{n}(X_1 + \cdots + X_n) \overset{n\to\infty}{\sim} 0$ $1\cdot(X_1 + \cdots + X_n) \overset{n\to\infty}{\sim} \infty$

$\frac{1}{\sqrt{n}}$ is the right scaling factor!

So for $\eta \le \frac{1}{d_{in}} \cdot \delta$, $1/d_{in}$ is the right scaling

for : $\vec{h}_{l-1} \to [W_l] \to \oplus b \to [\text{non-li}] \to \vec{h}_l$, we want $\|\vec{h}_l\|_{RMS} = 1$

⇒ $\|\vec{h}_l\|_2 = \sqrt{d_{out}}$, and suppose don't care bias & non-li:

$\|\vec{h}_l\|_2 = \|W_l\|_2 \|\vec{h}_{l-1}\|_2$, $\|W_l\|_2 = \Theta(\sqrt{\frac{d_{out}}{d_{in}}})$ ⇒ $\|W_l\|_{RMS} = \Theta(1)$

⇒ fixed RMS norm as we started with!!

⇒ If $\|W_l\|_{RMS\to RMS} = \Theta(1)$, $\|\vec{h}_l\|_{RMS} = \Theta(1)$, then $\|\vec{h}_l\|_{RMS} = \Theta(1)$

This 'zero-shot hyperparameter transfer' is good! μP!

For more info, please refer to HW3 Topic 2

No.

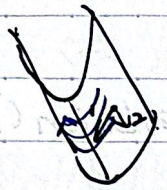
Date

Muon

Shampoo Optimizer: if $\nabla_W \mathcal{L}(W) = U\Sigma V^T$, then:

$$W_{t+1} = W_t - \eta UV^T.$$

Why discard Σ? It can be confusing!



For example, direction 1's gradient is large, and W can be busy swinging (like blue line), but pay less attention to direction 2.

But SVD can be expensive! Muon: if we can find:

$G_t = U\Sigma V^T$, $p(G_t) = Up(\Sigma)V^T$, $p^n(G_t) = Up^n(\Sigma)V^T = UV^T$

This method is called Newton-Schulz

$$\therefore \text{Muon}: \begin{cases} B_t = \mu B_{t-1} + \nabla_W \mathcal{L}(W) \\ O_t = \text{Newton Schulz}(B_t) \\ W_t = W_{t-1} - \eta O_t \end{cases}$$

No.

Date

CNN & GNN

* Architecture Introduction Order: MLP → CNN → GNN → RNN → Transformer

Basic Data Form: Images: $h \times w \times 3$ (RGB)

Overall: CNN

Key Ideas: (Expressivity) convolutional filters called "kernels"

① Locality → convolution* ② Invariance / Equivalence / Symmetries

③ Hierachical Structure ⤷⇒ weight sharing & data augmentation (Train)

⇒ Depth & Multi-Resolution

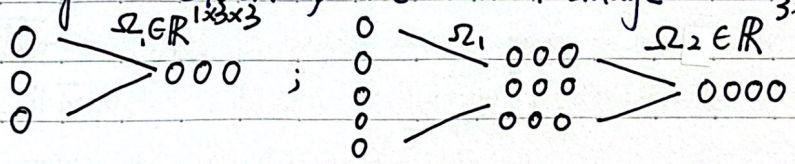
Key Ideas: (Work more Effectively)

① Normalization Layers ② Dropout ③ Residual / Skip Connection

Convolution: In DL, we don't flip !! And can have a bias.

- For kernels, weights are not shared across channels. Elementwise multiplication and sum, followed by a ReLU.
- Also: If wanting more channels at the output ⇒ repeat with fresh weights & bias
- Padding: "0", "same", "mirror" or no padding

Zero Padding ⇒ Black frame around Image

- Reception Field: 

Suppose. $K\times K$ conv with ℓ layer, one position in the final layer can see the information of range up to: $\frac{K+1}{2}\ell - 1$

- Down Sampling via Stride i
- Pooling: stride / subscript (取固定位置), maxpooling; mean pooling
- $K\times K$ conv with Cin channel input and Cout outputs:

⇒ Weights: K^2 Cin Cout Bias: Cout (#)

- Backprop w.r.t. weights & bias? PyTorch will handle it.

每处算完后的 $\frac{\partial L}{\partial w_i}$, 最后累加起来 ($Conv(X,W) = X_1 \otimes W + X_2 \otimes W \cdots$, $\frac{\partial L}{\partial W} = \sum_i \frac{\partial L}{\partial w_i}|_{X_i}$

⇔ (Scale of $\frac{\partial L}{\partial W}$ change with the number of terms in $\sum_i$)

- Data Augmentation: Robustness

Date

· Normalization Layers for Stability and Effectiveness

Idea: RMSNorm: d-dim $\vec{h_i} \xrightarrow{\text{RMSNorm}} \tilde{h_i}$, $\|\tilde{h_i}\|_{RMS} = 1$

$\therefore \tilde{h_i} = \frac{\vec{h_i}}{s}$, where $s = \sqrt{\frac{1}{d}\sum_j h_i[j]^2 + \varepsilon}$, s.t.:

$\Leftrightarrow \tilde{h_i} = \sqrt{d}\frac{\vec{h_i}}{\|\vec{h_i}\|_2 + \varepsilon} = \vec{h_i} / (\|\vec{h_i}\|_{RMS} + \varepsilon)$ ↙ 满足

Batch Norm: Average Over Space & Batch

Input: (bs, channel, H, W): 对每一个channel，计算这一个channel在整个批次中的均值&方差

$m = \frac{1}{|B|}\sum_{i \in B} h_{in}$, $s = \sqrt{\frac{1}{|B|}\sum_{i \in B}(h_{in} - m)^2}$, $h_{out}[i,j] = \gamma\left(\frac{h_{in}[i,j] - m}{s + \varepsilon}\right) + \delta$

where γ and δ are learnable, initialized with $\gamma = 1$ $\delta = 0$

But for test/inference time: Batchnorm don't have a batch. Need m&s!

How to get? ① 拿 train epoch 最后一个 m&s

② Run an extra epoch of training with no grad update

③ Just learn m&s using held-out data

· Dropout "Data Augmentation" inside our network during training

Idea: Zero out random activation inside net ⇒ Encourage Internal Representation to be redundant (Typical for MLP)

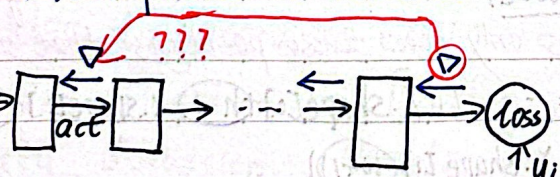
Training do dropout, but Inference? 若仅是放弃(zero out)通常不 work!

因为会 mess up scale

⇒ Solution: Scale activations by p. (Replace multiplication Noise by its mean)

(Done by PyTorch Dropout: Inference: $\tilde{h}[i] = \begin{cases} 0, & 1-p \\ \frac{h_i[i]}{p}, & p \end{cases}$ ☆

⇒ (不必 inference 时 scale back)



Gradients are dying!

May causing overfitting!

Desire: Create A "checkpoint superhighway" layer

Idea 1: Add links from output of all earlier layers to the input of a single

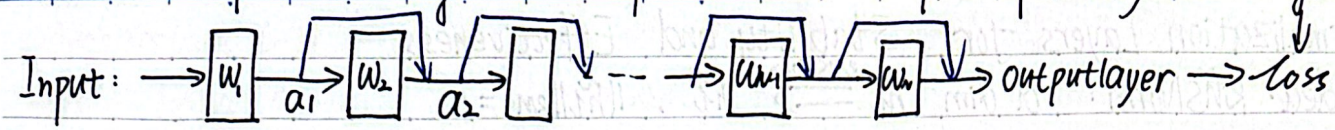
⇒ Blow up architectures! Not working!

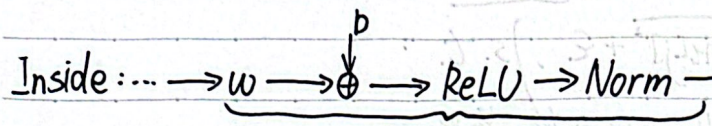
KOKUYO

No.

Date

Idea2: Skip Connecting: Add Input to the Output of a layer! y_i



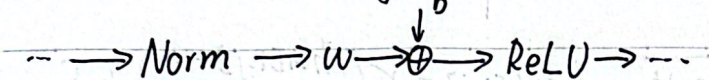


(W&b: Conv style or MLP style).

⇒ Neural Network: $f_1(f_2(\cdots f_L(x)\cdots))$

Now: Resnet Skip Connection $f_{i+1}(x) = f_i(x) + g_i[f_i(x)]$

Norm: Modern Default: Inside the block near the start



Pre-Norm Convention ⇒ stable training

But one challenge: if $f_i(x)$ change the dimension of input?

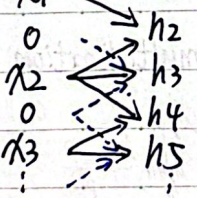
Simple addition will have dimension mis-match!

⇒ Need Something that changes shape, e.g., Linear Layer

△ Note: If entire block is just a maxpool downsampling or a subsampling, no skip connection needed for this block, since gradients can pass cleanly.

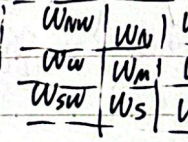
- △: Depthwise-Conv: Just treat each channel seperately as if it was a one-input-channel & one-output-channel conv. No weight sharing across channels.
- Upsampling? For pooling's inverse, we can zero or duplicate pad

⇒ Idea: Fill with zeros and use a learnable Transpose Conv

Tran Conv: 

, essentially: zero-fill upsample + Conv.

"→": no effect in a Conv Operation

3x3 example: 

give stride

$\vec{h}_{out} = \vec{b} + \sum_i W_i \vec{h}_{in_i}$, i ranges only over those positions that are neighbors

Code: h, w = K.shape → Tran Conv: Y = ... ((X.shape[0]+h-1), X.shape[1]+(w-1))

Conv: Y = torch.zeros((X.shape[0]-h+1, X.shape[1]-(w+1)))

for i in range(X.shape[0]):

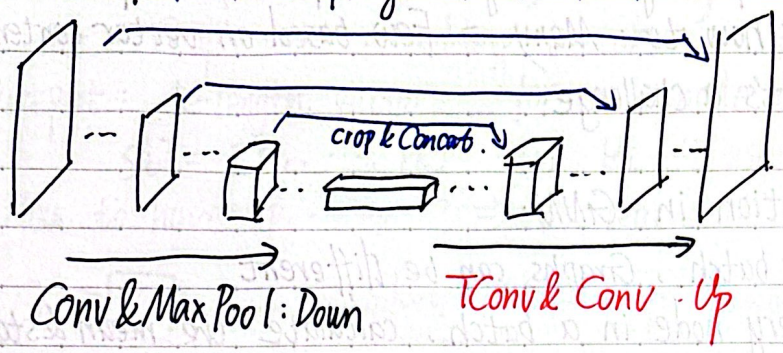
for j in range(X.shape[1]): → Tran Conv: Y[i:i+h, j:j+w] += X[i,j] * K *

Conv Y[i,j] = (X[i:i+h, j:j+w] * K).sum().

No.

Date

· With Up & DownSampling ⇒ U-net:



Conv & Max Pool : Down

TConv & Conv : Up

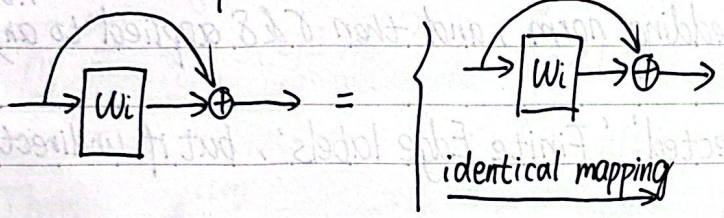
☆ 后半段 : Getting Details

前半 : Big Picture

Concat : Combine them !

Addition: Stochastic Depth Regularization

Idea : Dropout the entire block with some probability



It helps with overfitting and create functional redundancy

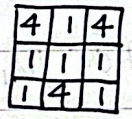
☆ Unlike dropout, no need to adjust anything during test time!

(Due to identical mapping & Presence of Normalization inside block)

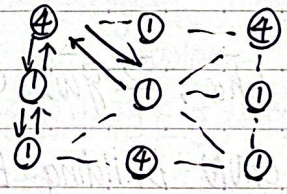
GNN : Graph Neural Networks (Generalizing CNNs)

Now: Image → General Graph

Image as a Graph ?



⇒



Edges are labeled with one of: {N, S, E, W, NE, SE, NW, SW}

Then, a 3×3 Conv can be viewed as receiving informations from neighbors, each of whom has their own labeled edge to me, multiplying by an edge-label-specific weight matrix, and adding up together with a special self-specific matrix (node's info & bias)

Core idea to generalize CNN ⇒ Every edge labeled from a discrete finite set

△: What does not generalize ? [Pooling, down/up sampling]*

Need discussion : Normalization

* Why do we do this ? ⇒ Grow Receptive Field Faster

Hack it ! Add a global node connected to Everything

But it can have side effect : ① blur away all fine-grained details

KOKUYO

② Oversquashing: info loss since squashing so many nodes' info into a finite embedding
So if wanting to downsample: how to: Many → Few based on vector contents (Like clustering) It's a challenge.

Another Challenge is Normalization in GNN:
batch norm can't work: In a batch, Graphs can be different
Layer norm can work: For every node in a batch, calculate the mean & std of all channels of this node and normalize
And then, use shared γ and δ (scale and shift) to process every node
* Easy choice: Just per-embedding norm, and then γ & δ applied to any node

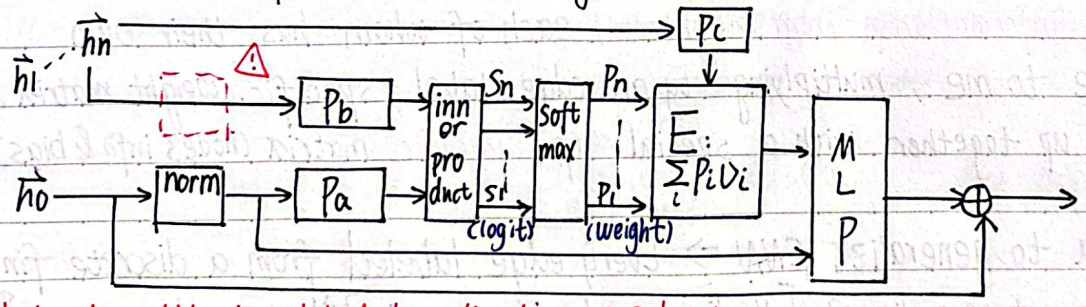
We have tackled senario of 'Directed' 'Finite Edge labels', but if undirected or infinite? Need to improve.
For a node to update, which information it will have:
① myself ② degree: # of neighbors ③ set of neighbors.
Also, for the info from neighbor set, our processing should be permutation-insensitive
So: Generalized Framework: For node with info h_0, update should be:

$$f_{w_1}(h_0, \sum_i g_{w_2}(h_i))$$

not necessarily Σ, can be Π, min/max mean, medium ……

More Generally: $f_{w_1}(\vec{h_0}, \Sigma \vec{g_{w_2}}(\vec{h_0}, \vec{h_i}))$ or $f_{w_1}(\vec{h_0}, \Sigma S_{w_2}(\vec{h_0}, \vec{h_i}) g_{w_3}(\vec{h_i}))$ (weight)

⇒ Natural Example: GNN Building Block



Obviously, this is related to attention mechanism

Back to challenge of Downsampling: Introduce: DiffPool
Idea: When we downsample, we get a smaller graph, but it's fully connected with probability on each edge

No.

Date

⇒ Make each point belong to every cluster, just with some probability

Input : Graph with embedding vectors

Output : k-length vector for each node with prob init:

$S_i^T = [p_1, \cdots, p_K]$ for $\forall i$ ← use softmax! (Differentiable)

How to have $p_1 \cdots p_K$? ⇒ Use learnable algorithm: GNN

But $\vec{S_i}^T$ aren't a graph. Not yet. What to do ?

Need New Vectors (Embeddings) for k clusters.

$\vec{h_j}^{(l+1)} = \sum_{i} S_i[j]\, \vec{h_i}^{(l)}$ ⇒ Let $A^{(l)} \in \mathbb{R}^{N \times N}$ ← original adjacency matrix (symmetric)

i ← pre-pooling node

↑ j-th cluster in layer $l+1$ (after pooling)

Then $\underbrace{A^{(l+1)}}_{\in \mathbb{R}^{k \times k}} = S^T A^{(l)} S \in \mathbb{R}^{N \times k}$ A_{ij} represents how strong the edge is between cluster i and j

⇒ Now : A new graph with k nodes and a new adjacency matrix

Now:

Can help get desirable properties* ← Auxiliary Loss

⚠ Not to immediately normalize these messages (Aij has stength info)

* Property 1: Have clusters respect graph topology

$\mathcal{L} = \|A - SS^T\|_F$ cause encourage : 原始图中相互连接的节点在簇分配上也应相似

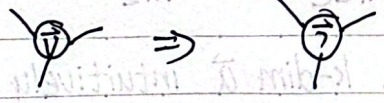
Property 2: Want to be close to hard cluster assignment

$\mathcal{L} = \frac{1}{N} \sum_i H(\vec{S_i}^T)$ ↑ Entropy (want assignment to be more deterministic)

Final thing : Held-out Data△ (Node-Level Problem)*

Approach 1: Excise entire nodes from the graph : but impact connectivity

2: Remove content from the node but leave it in



$\vec{?}$ are learnable, called mask

*: Like node classfication, tagging, etc

△: How to make vali/test dataset

KOKUYO

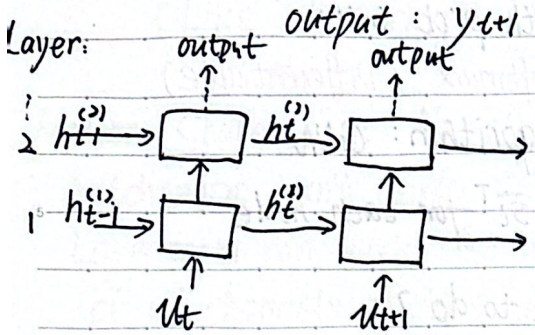
No.

Date

RNN / Self Supervision

Traditional RNN: $h_{t+1} = \sigma(W_h \cdot h_t + W_u (u_{t+1} + b)$

output: $y_{t+1} = c_j h_{t+1} + b_y$



Key problem: ① Gradients Exploding/Vanishing

② Non parellel Computation

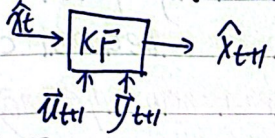
Challenges: Dependency Leaning

Recap on Kalman Filter: $\hat{x}_{t+1} = A\hat{x}_t + B_u \vec{u}_{t+1} + F(\vec{y}_{t+1} - C(A\hat{x}_t + B_u(\vec{u}_{t+1}) - D\vec{u}_{t+1})$

$= \tilde{A}\hat{x}_t + \tilde{B}\vec{u}_{t+1} + F(\vec{y}_{t+1})$

where $\tilde{A} = (I - FC)A \quad \tilde{B} = (I - FC)B_u - FD$

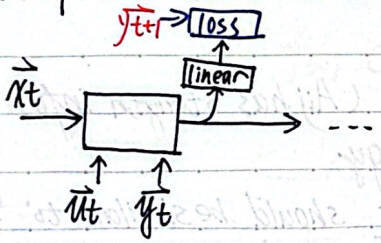
and $\begin{cases} F = PC^T(CPC^T + K_v)^{-1} \\ P = APA^T + K_w - APC^T(CPC^T + K_v)^{-1}CPA^T \leftarrow (P \text{ satisfy}) \end{cases}$

Anyway, essence is: 

, we can regard $\tilde{A}\ \tilde{B}\ F$: learnable

And $\vec{y}_{t+1}$: gt! For supervision!

But if we don't have label, how to supervise? Use next y!



This is self-supervision!

① We can learn a partial pattern that can be useful

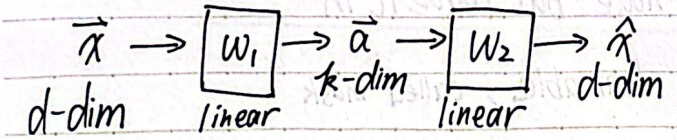
② Might need scaffolding(脚手架) parts of NN

③ Generic idea of "next-thing" prediction in causal sequence modeling

Step back: Consider PCA: Classical Solution:

$X = U\Sigma V^T = \sum \sigma_i \vec{u}_i \vec{v}_i^T$, For $\{\sigma_i\}$ select top-k

IF design following network:



to minimize $\|\vec{x} - \hat{x}\|_2^2$

then this k-dim $\vec{a}$ intuitively !! contains the most rich info to reconstruct

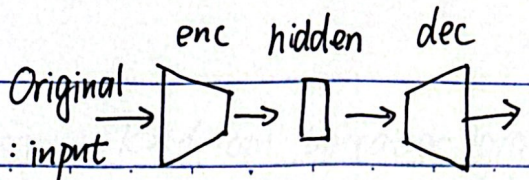
Has a name: auto-encoder

△ Traditional Perception: Decoder is scaffolding

No.

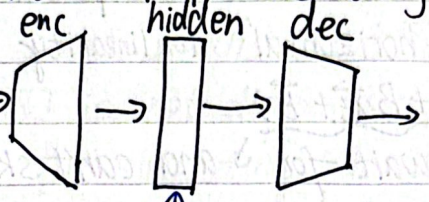
Date . .

Classic Autoencoder: input → enc → hidden → dec → output = original input



This hidden tensor contains rich feature though in low dimension!

Sparse Autoencoder:



can make hidden "bottleneck" even bigger than origial

with ⟹ Auxiliary loss in sparsity-seeking. eg l_1-l_1 penalty

Denoising ones: Above two with data augmentation.

Masked Autoencoder: 'Aug' is masking out some patches and reconstruct them

Besides PCA, another self-supervised ML task: clustering

⇔ Contrastive Learning. core idea: an example and its own augmentations should be in the same cluster. Different examples should be in dif cluster.

Basic Architecture:

$\vec{x}_j \to enc \to e_j$ } far apart

$\vec{x}_i \to enc \to e_i$ }

$Aug(\vec{x}_i) \to enc \to e_i'$ } close together

*: Note that in Teacher-Forcing: input is gt, not last prediction!

But for inferencing, input is the last prediction (auto-regressive)

No.

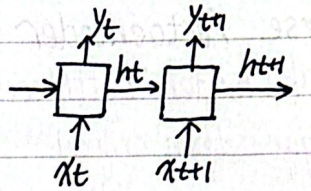
Date

State Space Model

Problem with traditional RNN: ① Hard to parallelize ② Hard to learn long-range dependency

For ①: Idea: Eliminate 'horizontal' nonlinearity

In RNN: $\vec{h_{t+1}} = \sigma(A\vec{h_t} + B\vec{x_t} + \vec{b})$



Nonlinearity! Have to wait for ↘ and can't skip

What if: $\vec{h_{t+1}} = A\vec{h_t} + B\vec{x_t}$?

Then Lets say: $\begin{cases} \vec{h_t} = A\vec{h_{t-1}} + B\vec{x_t} \\ \vec{y_t} = C\vec{h_t} + D\vec{x_t} \end{cases}$ ⇒ Linear-time invariant system

ABCD: Learnable

(A, B, C, D not dependent on time)

We can unroll $\vec{y_k}$! $\vec{y_k} = CA^kB\vec{x_0} + \cdots + CB\vec{x_k} + D\vec{x_t}$

Now $\vec{y_k}$ only depends on previous input! No on hidden states!

At time t, computation needs O(t); But if Max seq length is T, so time range from 1 to t, computation cost is $O(T^2)$

Can we improve? Yes with FFT: $O(T\log T)$

But what about expressive power? We can add vertical nonlinearity, aka, MLP (e.g.) on state-space layer.

So to summarize: Training is actually sort of convolution*

For inference: All past information is in the state. $\vec{h_t}$

In $\vec{y_k}$, we have CA^kB matrix. Can we speed up its computation?

For A^k, if A is diagonal, then A^k is easy to compute.

Now back to $\vec{h_t} = A\vec{h_{t-1}} + B\vec{x_t}$ and $\vec{y_t} = C\vec{h_t} + D\vec{x_t}$

Let's say: $\vec{y_t}$: d_y-dim; $\vec{x_t}$: d_x-dim; $\vec{h_t}$: d_h dim ("Memory")

$\vec{y_t} = \begin{bmatrix} y_t[1] \\ \vdots \\ y_t[d_y] \end{bmatrix}$, think of time-evolution sequence on a Component by component basis.

such sequences $y[k]$: $y_1[k]$, $y_2[k]$, ···, $y_t[k]$, and there are d_y

Regard $x[k]$ in a same way: Def $Seq(x[1])$: $x_1[1]\ x_2[1]\ \cdots\ x_t[1]$

Then an important formula is: in fact, Convolution

$$Seq(y[k]) = \sum_{l=1}^{d_x} F_{k,l} * Seq_l(x[l])$$

↑ Appropriate Filter

No.

Date

Can we simplify ? I mean: $k \times l$ can be rather large!

In imaging, kernels we use are Depthwise / Channel-Wise.

If A is diagonal, then $h_t[i] = \lambda_i h_{t-1}[i] + B\vec{x}_{t-1}$ channel wise

So the update of $h_t[i]$ is not dependent of $h_{t-1}[i]$ $(i \neq i)$ ⇒ ΔT

For B, block diagonal, for simulating 'locality'

Efficiency v.s. Efficacy Tradeoff

Larger $\vec{h}_t$, more compute, but also more memory / history tracking.

So goal: want to make $\vec{h}_t$ as large as possible, while still being able to run. Now: Introduce: S4 model

In a LTI (Linear Time Invariance) System, use FFT.

Let $A = \begin{bmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_N \end{bmatrix}$, then how is λ related to system stability?

if $|\lambda| > 1$: Bad! But if $|\lambda| < 1$: history die out fast!

So for $\lambda \in \mathbb{R}$, $|\lambda| = 1 \Rightarrow \lambda = \pm 1$ we have two choices and far apart $(-1 \leftrightarrow 1)$

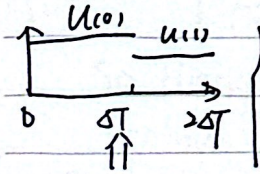
Solution: Complex eigenvalues! $\lambda = \exp(-\text{ReLU}(\lambda_R) + j\lambda_I)$ ⇓ GD hard

欲 A 特征值为复数且位于单位圆附近

And for initialization, important! Have to be mostly on or very close to unit circle with some inside unit circle. Use Hippo-based initialization.

Also: Freeze A with good initialization and only train the rest also seemed to work well!

S6/Mamba: Think of Discrete-time System as a [discretization] (Sampling) of a continuous time system:



$$\frac{d\vec{h}(t)}{dt} = A\vec{h}(t) + B\vec{u}(t)$$

$$\vec{y}(t) = C \cdot \vec{h}(t)$$

Δ: And to convert CT to DT. we need a sampling interval ΔT

Scalar case: $\dot{x}(t) = ax(t) + bu(t)$

Assume $u(t)$ is piecewise constant for time ΔT. (ΔT small: reasonable).

Note that: $a > 0$: explode; $a < 0 \Rightarrow$ decay to 0

$$x(\Delta T) = e^{a\Delta T} \cdot x(0) + \frac{b}{a}(e^{a\Delta T} - 1)u(0)$$

Assume $a < 0$: Then $\Delta T \to 0 \Rightarrow e^{a\Delta T} \approx 1 \Rightarrow$ Remember!

ΔT larg ⇒ $e^{a\Delta T} \to 0 \Rightarrow$ forget

KOKUYO

No.

Date

Moreover: $e^{a\Delta T} - 1 \approx a\Delta T$ (T small), then $\frac{b}{a}(e^{a\Delta T} - 1) \approx b\Delta T$.

$\Delta T \to 0 \Rightarrow b\Delta T \to 0 \Rightarrow$ input doesn't matter

ΔT large $\Rightarrow b\Delta T$ large $\Rightarrow$ input matters

So via changing ΔT, can change relative importance of past & future.

Idea: Choose ΔT based on input and learn it.

How to generate ΔT ? Softplus: $\text{Softplus}(x) = \ln(1+e^x)$

So: $\begin{cases} \vec{h_{k+1}} = A_k \vec{h_k} + B_k x_k \\ \vec{y_{k+1}} = C \vec{h_k} \end{cases}$ $A_k = \begin{bmatrix} e^{\lambda_1 \Delta T} & & \\ & e^{\lambda_2 \Delta T} & \\ & & \ddots \end{bmatrix}$ $B_k = \vec{b_k} = \vec{b}\Delta T$

Lastly: Initialization:

λI blocks for A: A few methods:

① $\lambda_n = -\frac{1}{2} + n i$

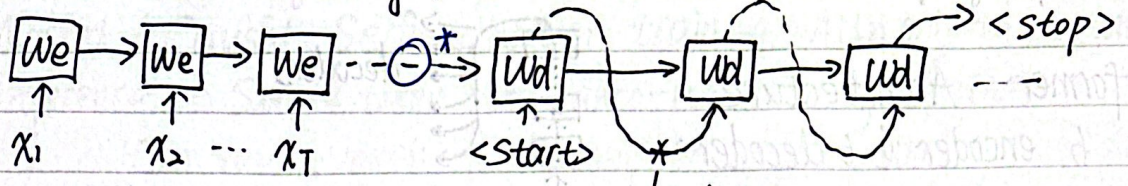
② $\lambda_n = -\frac{1}{2}$

③ $\lambda_n = -(n+1)$.

Date

Attention: [1] Intro to Attn

Sequence Problem using RNN (Enc-Dec Architecture)

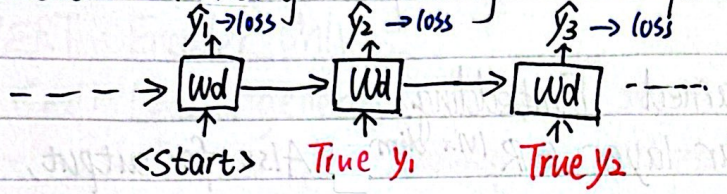


Loss: Cross-Entropy

→ With Sampling

* Bottleneck: Need to capture all of the input sequence in this bottleneck

Teacher-Forcing: Training technique for auto-regressive generation



But Ground Truth!

The input here is not autoregressive one

In U-net, we've met bottleneck. In U-net, solution is adding connections.

But for language: cannot set a topology of connects in advance!

Solution: Hash table, but need to be differentiable.

So we take weighted average: $\frac{1}{t}\sum_i \text{sim}(\vec{q}_n, \vec{k}_i)\cdot\vec{v}_i$

$\text{sim}\langle\vec{q}_n, \vec{k}_i\rangle$: inner product & softmax.

Key question: How to learn them? W_K, W_{q_n}, W_V!

There are many types of attention···

Cross-Attention: Keys + Values from encoder, Querys from decoder

Self-Attention: K, Q, V from decoder:

$$\begin{cases}\vec{q}_n = \beta_q + W_q\vec{x}_n \\ \vec{k}_m = \beta_K + W_K\vec{x}_m \\ \vec{v}_i = \beta_V + W_V\cdot\vec{x}_i\end{cases}$$

$\beta_{q,k,v}$ & $W_{q,k,v}$ are learnable

$$a[\vec{x}_m, \vec{x}_n] = \frac{\exp[\langle\vec{k}_m, \vec{q}_n\rangle]}{\sum_{j=1}^{N}\exp[\langle\vec{k}_j, \vec{q}_n\rangle]}$$

So Higher similarity ⇒ Higher attention

Scaled Self-Attention: $\frac{\exp[\langle\vec{k}_m, \vec{q}_n\rangle/\sqrt{D}]}{\sum_{j=1}^{N}\exp[\langle\vec{k}_j, \vec{q}_n\rangle/\sqrt{D}]}$ where $W_q, W_K \in \mathbb{R}^{D\times N}$

⇒ Prevent entries to softmax from getting too large

Multi-Head Attention: $X \in \mathbb{R}^{N\times D}$

Ω_c: linear transform

Q, K, V $\in \mathbb{R}^{N\times D/H}$ → Attention → Head$_i$ output $\in \mathbb{R}^{N\times D/H}$

Concatenation and Transform

$\Omega_c[S_a[X]^T, S_a[X]^T]^T$

Transpose to concat Vertically

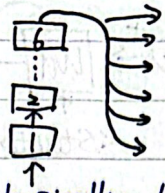
No.

Date

Multi-Query Attention: For each head: keys and values are the same, but change the query for each head. It can save memory!

[2] Transformer Architecture

Typically 6 encoder & 6 decoder:



And normalization layers are now typically before the attention block.

GPT-3: Decoder Only: Word embeddings are directly fed into a series of transformation layers with masked-self-attention.

[3] Embedding Layers.: Use learned Embedding.

For input, we need a linear layer $\in \mathbb{R}^{|V| \times dim}$; Also for output, we use linear layer to project to $|V|$ and softmax.

Like before, we may want $W\vec{x} = \vec{y}$, $\|\vec{y}\|_{RMS} = 1$, and since $\vec{x}$ are $\in \mathbb{R}^{|V|}$, one-hot, so we want each column of W to have RMS Norm 1.

[4] Positional Encoding:

① In paper, sinusoidal pattern is used, but now no longer used

② NoPE ③ Learned Relative Positional Encoding.

score $s_i = \frac{\langle \vec{q_t} \cdot \vec{k_i} \rangle}{\sqrt{d}}$, $P_{t,i} = \frac{e^{s_i}}{\sum_j e^{s_j}} \Rightarrow \frac{e^{s_i + (b_{t-i})}}{\sum_j e^{s_j + (b_{t-j})}}$ ← learn these!

④ RoPE - Modify $\vec{k} \cdot \vec{q_n}$ based on their absolute position so that the score depends on relative position.

$\langle \vec{k} \cdot \vec{q_n} \rangle = \|\vec{k}\|_2 \|\vec{q_n}\|_2 \cos\theta$ Rotation: $\begin{bmatrix} \cos w_t & -\sin w_t \\ \sin w_t & \cos w_t \end{bmatrix}$

So $\langle R_t \vec{q_t}, R_i \vec{k_i} \rangle = \vec{q_t}^T R_t^T R_i \vec{k_i}$

Rotate $-w_t$ Rotate w_i

So design $M = \begin{bmatrix} M_1 & & \\ & M_2 & \\ & & \ddots \\ & & & M_{d/2} \end{bmatrix}$ for $q_n, k \in \mathbb{R}^d$, $M_j = \begin{bmatrix} \cos(2\pi f_j \cdot t) & -\sin(2\pi f_j \cdot t) \\ \sin(2\pi f_j \cdot t) & -\cos(2\pi f_j \cdot t) \end{bmatrix}$

But f_j are different. Where f_j is fixed and stands for frequency

Different f_j freq → Different timescale. Low f_j channels → long range relative info.

Date

5 Back to GPT-3: Decoder-Only model:

- Focus: Generate the next token in a sequence.
- Masked / Causal Self attention training: Attn only to current and previous
- Inference: Sample token and auto-regressive generation
  - △: Can reuse much of the computation for attn
- Sampling Strategy: Beam search / Top-k sampling
- Size: 96 Transformer Layers ; 96 heads in self-attention layers

6 BERT: Encoder Only

Goal: Learn general info about statistics of language
And use fine-tuning to adapt to solve specific task

Self-Supervision For Pretrain: ① Masked Token Prediction

<cls>: Start token <Sep>: token to seperate sentences

② Sent A Sent B: Predict if Sent B immediately follows A

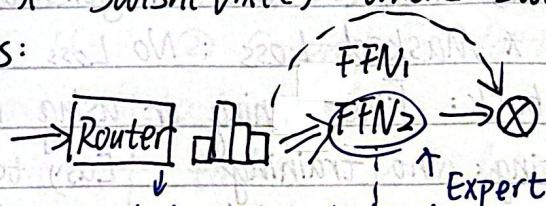
→ Predict this on the sep token.

7 MoE: First: Gated Linear Units:

$GLU(x) = (Wx+b) * \text{sigmoid}(Vx+c)$

$SwiGLU(x) = (Wx+b) * Swish(Vx+c)$ where $Swish(x) = x \cdot \text{sigmoid}(\beta x)$ (β: learnable)

So Mixture of Experts:



Router is designed to choose only one or few Experts to activate

Add +Norm

Determine which expert to handle

weighted average (only activate experts have non-zero weights)

Date

ICL / Prompting & Fine-tuning

GPT-style models: Recal Pretraining: Next-token Prediction

△: Data mix matters: 预训练数据组成比例重要!

☆: Teacher-Forcing (Input: GT, not sampled token)

And for inferencing: Autoregressive:

→ De-Embedder → Softmax (Temp*) → Sampler → next input token

*: $e^{\frac{1}{\pi}x[i]} / \sum_i e^{\frac{1}{\pi}x[i]}$, π: temp

GPT-3 (2020): With even bigger model, also get in-context learnings.

E.g. Red: Ruby; Blue: Saphire

But now: Can solve a learning problem with a few examples without any finetuning or gradient descent. ⇐> Pure Prompting.

Challenge: all training data fit in context.

Two core problem: ① How to find better prompt?

② How to build model that is more promptable? (更容易被提示)

⇕: Instruction Tuning & SFT

Input Prompt tokens and [do loss on generated response token]

* Masked Loss: No Loss on Question

But take one step back: Fine-tuning or using a pretrained model?

0) Pure prompting: no training Easy to do, but might not work as well

1). Linear Probing: treat model as embedder, for a feature extracter. Do classical ML to train a separate model.

2). Full fine-tuning: adjust the weights of the pretrained model.

Tip: Don't just initialize a random head better way is to initialize head to 0.

As for better prompt: Can use tools like DSPy

But if we don't want full fine-tuning, we can PEFT (参数高效微调), like softprompts and prefix tuning

· Soft Prompts: 不再在input中加离散prompt文本, 而是在embedding层加learnable参数(向量) If pre-prompt has length of 100, 则向embedder添加100个新的可学习列 *

No.

Date

*: 指的 Vocab 扩充: $W_{vocab}^{D\times V}$ (concat) $W_{vocab}^{D\times(V+100)}$.

And during training, $W^{D\times V}$ freeze, and $W_{new}^{D\times 100}$ (only) are learnable.
Keep GPT model parameters frozen as well.
这样 model 会让这100个向量来激发模型完成特定任务，而不是设计prompt

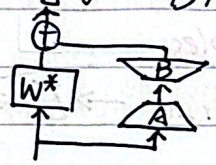
· Prefix Tuning: Soft Prompts 之上，还调整每一层 KV cache.

LoRA:

Idea: Why not do parameter-efficient finetuning beyond soft-prompt?
Soft Prompt & Prefix Tuning 虽高效，但并未改变权重矩阵。而 LoRA 允许以极小成本修改模型深层权重 (e.g. QKV & MLP)

In pretrained model, weight-matrices W_*: We update: $W = W_* + \Delta W$
where ΔW is in rank $r << rank(W_*)$
∴ Let $\Delta W = BA$, $B \in R^{W_{*in} \times r}$, $A \in R^{r \times W_{*out}}$

Now:



⚠ A: Xavier Init
B: Zero Init (训练开始时，模型行为与原来一致，避免噪声使模型崩溃)

Δ: learning rate of LoRA can be different from the one in pretraining

Meta-Learning: Learning to learn ⇒ 训 model 以让它擅长于被微调"
Question: If we have a new task, what's the best base model?
① Randomly Init → No !! ② General Foundation Model
③ MAML (Model-Agnostic Meta Learning)

Need:
a. Collection of tasks: Data & Loss
b. Approach of fine-tuning
c. Approach of eval

Key insight: Learning Process of SGD is like a RNN
Be precise: A task i has Training Data D_i, Training Loss L_i, Test Loss $\tilde{L}_i$. Want to train to do well on all tasks from this family.

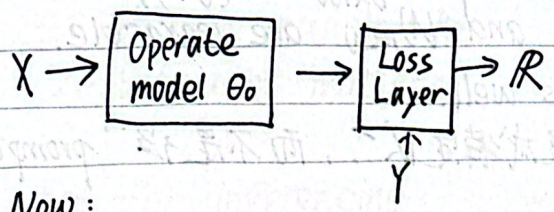
Assume we have a good initialization $\underline{\theta_0}$ (learnable parameter). How to use it?
1) Start model at θ_0
2) Do gradient descent steps with D, L to get θ_{final}
3) Evaluate θ_{Final} with held-out data $\tilde{D}$ and test loss $\tilde{L}$

KOKUYO

No.

Date

Fit these procedures (1,2,3) into our standard form:

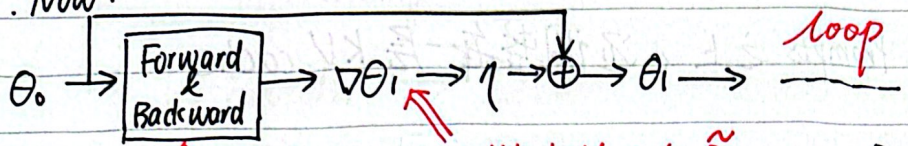


$X \leftrightarrow D, \tilde{D}$

$Y \leftrightarrow \tilde{D}_Y$

Loss Layer $\leftrightarrow \tilde{L}$

∴ Now:



↑ with training D with held-out $\tilde{D}$

△. To backprop through this, need to store θ_i history. But can gradient checkpoint to redo for/backward for this.

✡: MAML insight: That's just a deep-model. Can backprop through it

i.e., given a gradient $\nabla\theta_0$ on initial condition

Can take a step in that direction for a better condition

So we can: { outer-loop on tasks
inner-loop on batches/examples

Backprop

Task 1: $\theta_{0,0} \to \theta_{1,1} \to \theta_{1,2} \to \cdots \to \theta_{1,N_1} \to \tilde{L} \to R$

↓ apply

2: $\theta_{0,1} \to \theta_{2,0} \to \theta_{2,1} \to \cdots \to \theta_{2,N_2} \to \tilde{L} \to R$

3: ⋮

↓ $\theta_{0,final}$

Two variants: ① Reptile: Avoid backprop through backprop

⇒ Gradient $\approx \theta_{final} - \theta_{initial}$ (θ_0)

② ANIL/MetaOptnet/R2D2: Optimize for linear probes

E.g., for regression, have closed form formulas for head. Use closed form solution to compute W^*_{head}, and it is differentiable, so through it, gradients can back prop to feature extractor Φ.

△: Catastrophic Forgetting: When fine-tuning, model forgets how to do what it knew how to do.

Key Practical Solution: During fine-tuning, mix in some pre-training style, like 10%

△: Can be thought of a kind of overfitting

No.

Date

(VAE & Diffusion)

Generative model & {Post-training and test-time compute}

Generative Model:

Unconditional: [G] ← Randomness → New example

Conditional: Most practically useful

conditioning → [G] ← Randomness → New example

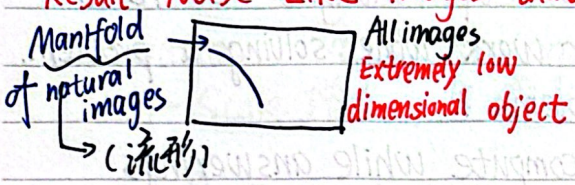
E.g. Prompt in LLM/Images/Video Generative Model

Ideas that don't work:

A) Use a classifier: Image: $\mathbb{R}^{50\times50\times3}$ → Classifier → $\mathbb{R}$ "cat score"

Try: Random Uniform Image followed by Gradient Ascent on the cat score.

Result: Noise-Like images that classifier confidently classifies as a cat



$\vec{x} \to [E] \xrightarrow{\vec{z}} [D] \to \hat{x}$

B) Use an autoencoder: labels are $\vec{x}_i$ itself, bottleneck in the middle.

Traditional Perspective: Decoder is scaffolding. 但推理时输入的 $\vec{z}$ 通常会得到模糊图像 ⇔ 训练时 $\vec{z}$ 分布与随机采样分布不同

VAE approach: 3 key ingredients:

1) make $\vec{z}$ random during training too.
2) Add a loss on distribution of $\vec{z}$
3) Make this work with SGD

$\vec{x} \to [E] \xrightarrow{\text{Distribution}} [\text{sample}] \xrightarrow{\vec{z}} [D] \to \hat{x} \to [\text{loss}] \to \mathbb{R}$

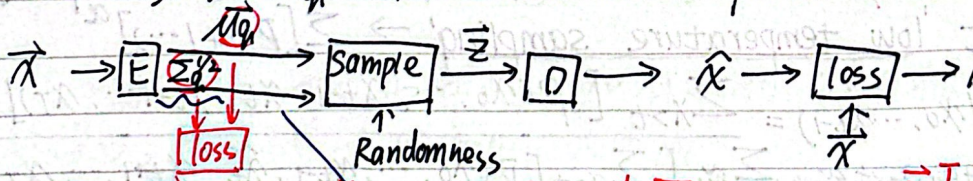
Desired Distribution → [loss] → $\mathbb{R}$; Randomness → sample ; $\vec{x}$ → loss

Loss on Distributions: KL Divergence: $KL(Q\|P) = \int Q(z) \ln \frac{Q(z)}{P(z)} dz$

Our choice for distribution: $N(0, I_k)$

But directly sample from $z \sim N(\mu, \Sigma)$ cuts gradients flow

$\Rightarrow z = \mu_q + \Sigma_q^{1/2} v$, v is a noice from standard normal distribution.



$KL(N(\mu_q, \Sigma_Q) \| N(0, I_k)) = \frac{1}{2} Tr(\Sigma_Q) + \vec{\mu_q}^T \vec{\mu_q} - k - \log(\det \Sigma_Q)$

$(\Sigma_Q = \Sigma_Q^{1/2} (\Sigma_Q^{1/2})^T)$ ⇔ Enforce semi-definite

KOKUYO

No.

Date

New tricks: 1) Using a KL Divergence Regularizer on a distribution

2). Treating Sampling in a way that allows gradients to just pass thrugh it.

3). Accepting the stochasticity of random noice in sampling as just more stochasticity in SGD-style optimization.

Post-training including RLVR: (RL with verifiable rewards)

Recall basic SFT for instruction following: Masked loss (on generated tokens, not questions)

For LLMs, one key advance was "chain-of-thought". So it is useful to have examples that show their work while solving a problem.

How to make a better at solving problems?

A) Be willing to spend more compute while answerings

B) Train it to be better

Test-time Compute: 0) Oldest Approach: Pure Promptings. "Think step by step"

1). Repeated generation. Generate N answers, and choose via majority vote

2). Sample better.

Δ: Observation on RLVR: empirically show better performance & generalization

但人们也注意到，RLVR生成的这些高质量答案，在base model (no RLVR)中其实也拥有比错误答案更高的概率

难题：若从原始模型 token-by-token sampling，一旦模型犯了一个小错，后面推理会完全崩溃，不知所措 (flounder)

How to solve it: a): old approach: beam search

b): Alternative: Don't sample from $P(x)$, sample from $(P(x))^{\alpha}$ $(\alpha>1)$

$\alpha>1$: lower temperature，会放大高概率选项，压制低概率项

This is not just 'low temperature sampling' → $\sum[p(x_t|\cdots)]^{\alpha}$

It's: $P_{desired}(x_t|x_0,\cdots x_{t-1}) = \frac{\sum_{x>t}[p(x_0,\cdots,x_{t-1},x_t,x_{t+1}\cdots,x_T)]^{\alpha}}{\sum_{\hat{x}_t}[\sum_{x>t}[p(x_0,\cdots,x_{t-1},\hat{x}_t,x_{t+1},\cdots,x_T)]^{\alpha}]}$

对整个序列联合概率作锐化

这被称为 power sampling，能更好地从全局角度筛选高质量的完整推理路径

Campus

No.

Date

To sample from $p(x)^{\alpha}$, in the paper, use MCMC sampling skill.

以前：模型一句话写完结束，写得烂也认

现在：模型先写个草稿，然后利用MCMC算法，反复地"回退"到句子的某个中间位置，尝试重新生成后缀

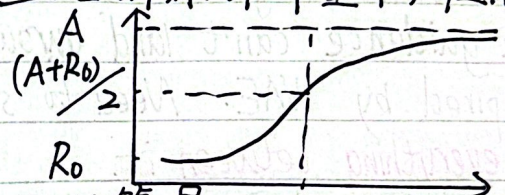
结果：只有当新生成的后缀让整句话的整体概率（经过α放大后）变得更高，才会保留修改。

RLVR: Reinforcement Learning with Verifiable Rewards

(Turn to 'train it better')

Observation: RL's scaling laws：随RL训练计算量↑，性能呈Log-Scale增长，且遵训曲线：

$$R_C = R_0 + \frac{A - R_0}{1 + (C_{mid}/C)^B}$$



C_{mid}：达到总增益50%时所需的计算量

Unpacking. $\nabla J_{ScaleRL}(\theta) = \mathbb{E}_{x\sim D,\ \{y_i\}_{i=1}^{G}\sim \pi_{gen}^{\theta_{old}}(\cdot|x)} \left[\frac{1}{\sum_{g=1}^{G}|y_g|} \sum_{i=1}^{G}\sum_{t=1}^{|y_i|} \text{sg}(\min(\rho_{i,t}, \varepsilon))\hat{A}_i^{norm} \log \pi_{train}^{\theta}(y_{i,t}) \right]$

Stop Gradient → 重要性

目标：用旧生成策略生成G个样本 ← token-level归一化 ← 当前策略生成$y_{i,t}$ token概率

where $\rho_{i,t} = \frac{\pi_{train}^{\theta}(y_{i,t})}{\pi_{gen}^{\theta_{old}}(y_{i,t})}$ 是Importance Sampling，因为π_{gen}生成数据，但π_{train}是更新的策略

$\hat{A}_i^{norm} = \hat{A}_i / \hat{A}_{std}$ 是优势函数，通常是 reward − baseline

故 Objective 本质：用旧π生成数据不断更新π_{train}，这样便不用更新一次π_{train}后再采样数据

上述式的重要基石：$\nabla_\theta \mathbb{E}_{y\sim p_\theta}[F(y)] = \mathbb{E}_{y\sim p_\theta}[F(y)\nabla_\theta(\log p_\theta)]$

即最大化期望回报，不需对奖励函数求导，也可对$\log p_\theta$求导并以$F(y)$作为权重 ($\hat{A} \Rightarrow F(y)$；$\log\pi \Rightarrow \log P$)

KL Regularization也可以帮忙。如有policy π_{ref}微调，希望是在$r(\cdot)$上多得分，同时不希望π policy "大变局"：

故 Objective: $\underset{\pi}{\text{argmax}}\ \mathbb{E}_{y\sim\pi(\cdot|x)}[r(x,y)] - \beta D_{KL}[\pi\|\pi_{ref}]$

如GRPO这个有KL Regularization

KOKUYO

No.

Date

Pretrain → SFT on Good Example ⇒ Generate → Get HF → Train RM → RL → Generate

RLHF (human feedback)：三步走：① SFT ② Reward Model：收集人类对 model output 的 Binary Comparison（人类不擅长给 real-value）。RM 被训练来预测哪个回答更好 ③使用 RM 为 rc·），do reinforcement learning, e.g.

$$\arg\max_{\pi} E[r(x,y)] - \beta D_{KL}[\pi \| \pi_{ref}]$$

↳ Direct Preference Optimization

DPO：直接在人类偏好数据上的对数似然优化，避免 RM 训练。定义奖励用 π_θ 隐式

$$L_{DPO}(\pi_\theta; \pi_{ref}) = E_{(x,y_w,y_l)\sim D}\left[\log \sigma\left(\beta \log \frac{\pi_\theta(y_w|x)}{\pi_{ref}(y_w|x)} - \beta \log \frac{\pi_\theta(y_l|x)}{\pi_{ref}(y_l|x)}\right)\right]$$

y_w: Winner 回答；y_l: Loser 回答 ↓ Sigmoid

Finally, back to generative model：diffusion. Recall that：Use a classifier as a guidance can't land on our desired manifold.

Inspired by VAE: Need to see noisy things during training, and why not everything between it.

Encoder: Forward Process: $x_t = \sqrt{1-\beta_t}\, x_{t-1} + \sqrt{\beta_t}\, \varepsilon_t$ where $\varepsilon_t \sim N(0, I)$ and β_t is a variance schedule.

Decoder: Predict the noise. The loss is MSE between ε_t & $\tilde{\mu}_t(x_t)$ Model.

Sampling Approach: ① DDPM (Denoising Diffusion Probabilistic Models)

关于 forward，其实可以一步写出 x_t & x_0 关系：

$$x_t = \sqrt{\bar{\alpha}_t}\, x_0 + \sqrt{1-\bar{\alpha}_t}\, \varepsilon, \quad \bar{\alpha}_t = \prod_{s=1}^{t}(1-\beta_s)$$

而 $q(x_{t-1}|x_t) = \frac{q(x_t|x_{t-1})\, q(x_{t-1})}{q(x_t)}$ 不可能

但 $q(x_{t-1}|x_t, x_0)$ 可以 通过 Bayes 得闭式解

$$q(x_{t-1}|x_t, x_0) = N(x_{t-1}; \tilde{\mu}_t(x_t, x_0), \tilde{\beta}_t I)$$

where $\tilde{\mu}_t(x_t, x_0) = \frac{\sqrt{\bar{\alpha}_{t-1}}\,\beta_t}{1-\bar{\alpha}_t} x_0 + \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t} x_t$

② DDIM: $x_{t-1} = \sqrt{\bar{\alpha}_{t-1}}\, \hat{x}_0 + \sqrt{1-\bar{\alpha}_{t-1}} \frac{x_t - \sqrt{\bar{\alpha}_t}\,\hat{x}_0}{\sqrt{1-\bar{\alpha}_t}} + \sigma_t \varepsilon_t$

若 $\sigma_t = 0$：

$$x_{t-1} = \sqrt{\bar{\alpha}_{t-1}}\left(\frac{x_t - \sqrt{1-\bar{\alpha}_t}\,\epsilon_\theta(x_t)}{\sqrt{\bar{\alpha}_t}}\right) + \sqrt{1-\bar{\alpha}_{t-1}}\,\epsilon_\theta(x_t)$$

DDPM ⇐

$x_1 \leftarrow N(0, \sigma_q^2)$

for $t = 1, (1-\Delta t), (1-2\Delta t), \cdots$

$\eta \leftarrow N(0, \sigma_q^2 \Delta t)$

$x_{t-\Delta t} \leftarrow f_\theta(x_t, t) + \eta$

DDIM ⇐

$\lambda \leftarrow \frac{\sqrt{t}}{\sqrt{t-\Delta t} + \sqrt{t}}$

$x_{t-\Delta t} \leftarrow x_t + \lambda f_\theta(x_t, t) - \lambda$

return x_0 图像

△：假设 $f_\theta(x_t, t)$ 预测 t 时去噪后