Learning to Stop While Learning to Predict Supplement Materials

Yu,Linlin

March 2021

Contents

1 Derivation of Oracle Time Distribution(KKT Simple Intro)

Maximize $\mathcal{J}_{\beta-VAE}(\theta, q_\phi; \mathbf{x}, \mathbf{y})$ is equivalent to minimize $\mathcal{L}(\theta, q_\phi; \mathbf{x}, \mathbf{y})$
Minimize:

$$\begin{aligned}\mathcal{L}(\theta, q_\phi; \mathbf{x}, \mathbf{y}) &= \mathbb{E}_{q_\phi}\ell(\mathbf{y}, \mathbf{x}_t; \theta) - \beta H(q_\phi) \\ &= \sum_{t=1}^{T}[-q_\phi(t)\log p_\theta(\mathbf{y}|t, \mathbf{x}) + \beta q_\phi(t)\log q_\phi(t)]\end{aligned}$$

we want to get the optimal value of $q_\phi(t)$, while $0 < p_\theta(\mathbf{y}|t, \mathbf{x}) < 1$ is a constant related to t for $q_\phi(t)$. Then the optimize problem is as follows:

$$\textbf{Minimize:} \sum_{t=1}^{T}[-q_\phi(t)\log p_\theta(\mathbf{y}|t, \mathbf{x}) + \beta q_\phi(t)\log q_\phi(t)]$$

where:

$$\sum_{t=1}^{T} q_\phi(t) = 1$$
$$0 \le q_\phi(t) \le 1$$

Simply, we use q_t, p_t denote $q_\phi(t)$ and $p_\theta(\mathbf{y}|t, \mathbf{x})$ respectively.
It is an convex optimization problem. Hessian matrix is positive definite matrix and the constraints are linear function. It has a global minimum
Use KKT condition to solve this problem :

$$L(q_t, \lambda, \mu, \sigma) = \sum_{t=1}^{T}[-q_t\log p_t + \beta q_t \log q_t] + \lambda(\sum_{t=1}^{T} q_t - 1) + \sum_{t=1}^{T}(-\mu_t q_t) + \sum_{t=1}^{T}\sigma_t(q_t - 1)$$

KKT conditions are: Primal feasibility, Dual feasibility, Complementary Slackness

$$\nabla_{q_t} L = 0 \tag{1}$$
$$\sum_{t=1}^{T} q_t - 1 = 0 \tag{2}$$
$$-q_t \le 0, \mu_t \ge 0, \mu_t q_t = 0 \tag{3}$$
$$q_t - 1 \le 0, \sigma_t \ge 0, \sigma_t(q_t - 1) = 0 \tag{4}$$
$$\tag{5}$$

For Equation 4:

$$\frac{\alpha L(q_t, \lambda, \mu, \sigma)}{\alpha q_t} = -\log p_t + \beta \log q_t + \beta + \lambda - \mu_t + \sigma_t = 0$$
$$q_t = \exp(\frac{\mu_t - \sigma_t - \beta - \lambda}{\beta}) p_t^{\frac{1}{\beta}}$$

Actually, $\mu_t = 0$ and $\sigma_t = 0$ and use the equation 5, we can get the oracle close form distribution.

$$q_t = \frac{p_t^{\frac{1}{\beta}}}{\sum_{t=1}^{T} p_t^{\frac{1}{\beta}}}$$

Introduction to KKT conditions

This is inspired by https://personal.utdallas.edu/ nicholas.ruozzi/cs6375/2021sp/lects/Lecture$_{4L}M_and_{K}ernel.$
Consider an optimization problem:

$$\begin{aligned} minimize \quad & f_0(x) \\ subject \quad to \quad & f_i(x) \leq 0 \quad i = 1, \cdots, m \\ & h_i(x) = 0 \quad i = 1, \cdots, p \end{aligned}$$

$f_0(x)$ is no necessarily convex and constraints do not need to be linear. We defined the Lagrangian:

$$L(x, \lambda, \nu) = f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{i=1}^{p} \nu_i h_i(x) \tag{6}$$

The Lagrangian form incorporate constraints into a new objective function. $\lambda \geq 0$ and ν are vectors of Lagrangian multipliers,which can be thought of as enforcing constraints.
Then the primal problem becomes:

$$\inf_x \sup_{\lambda \geq 0, \nu} L(x, \lambda, \nu) \tag{7}$$

We can create a dual problem:

$$\sup_{\lambda \geq 0, \nu} \inf_x L(x, \lambda, \nu) \tag{8}$$

If $\inf_x \sup_{\lambda \geq 0, \nu} L(x, \lambda, \nu) = \sup_{\lambda \geq 0, \nu} \inf_x L(x, \lambda, \nu)$, this is called a strong duality.
For any optimization problem with differentiable objective and constraint functions for which strong duality obtains, any pair of primal and dual optimal points must satisfy the KKT conditions:

- Stationarity: $\Delta L(x, \lambda, \nu) = 0$
- Complementary Slackness: $\lambda_i f_i(x) = 0$ for all $i = 1, \ldots, m$
- Primal feasibility: $f_i(x) \leq 0$ for all $i = 1, , m$ and $h_i(x) = 0$ for all $i = 1, \ldots, p$
- Dual feasibility: $\lambda \geq 0$ for all $i = 1, , m$

2 Proof of A.2 (Equivalence between the Reverse KL and Max-EM RL)

$$\min_{\phi} \mathbf{KL}(q_\phi(t)||q^*_\theta(t|\mathbf{y},\mathbf{x})) \tag{12}$$

$$= \min_{\phi} -\sum_{t=1}^{T} q_\phi(t) \log \underline{q^*_\theta(t|\mathbf{y},\mathbf{x})} - H(q_\phi) \tag{13}$$

$$= \min_{\phi} -\sum_{t=1}^{T} q_\phi(t) \log \underline{p_\theta(\mathbf{y}|t,\mathbf{x})^{\frac{1}{\beta}}} - H(q_\phi) \tag{14}$$

$$+ \sum_{t=1}^{T} q_\phi(t) \log \underline{\sum_{\tau=1}^{T} p_\theta(\mathbf{y}|\tau,\mathbf{x})^{\frac{1}{\beta}}} \tag{15}$$

$$= \min_{\phi} -\sum_{t=1}^{T} q_\phi(t) \log p_\theta(\mathbf{y}|t,\mathbf{x})^{\frac{1}{\beta}} - H(q_\phi) \tag{16}$$

$$+ \sum_{t=1}^{T} q_\phi(t) \underline{C(\mathbf{x},\mathbf{y})} \tag{17}$$

$$= \min_{\phi} -\sum_{t=1}^{T} q_\phi(t) \log p_\theta(\mathbf{y}|t,\mathbf{x})^{\frac{1}{\beta}} - H(q_\phi) \tag{18}$$

$$+ \underline{C(\mathbf{x},\mathbf{y})} \tag{19}$$

$$= \min_{\phi} -\sum_{t=1}^{T} q_\phi(t) \log p_\theta(\mathbf{y}|t,\mathbf{x})^{\frac{1}{\beta}} - H(q_\phi) \tag{20}$$

$$= \underline{\max_{\phi}} \sum_{t=1}^{T} q_\phi(t) \log p_\theta(\mathbf{y}|t,\mathbf{x})^{\frac{1}{\beta}} + H(q_\phi) \tag{21}$$

$$= \max_{\phi} \sum_{t=1}^{T} -q_\phi(t) \frac{1}{\beta} \ell(\mathbf{y},\mathbf{x}_t;\theta) + H(q_\phi) \tag{22}$$

$$= \max_{\phi} \mathbb{E}_{t\sim q_\phi}[-\frac{1}{\beta}\ell(\mathbf{y},\mathbf{x}_t;\theta) - \log q_\phi(t)] \tag{23}$$

Now we need to build the connection between the RL and this probability model. In a RL model, we have:

- State S: s_t
- Action A: a_t
- Reward $R(\cdot|s_t,a_t)$
- State transition Probability: $P(s_{t+1}|s_t,a_t)$
- Policy: $\pi(a_t|s_t)$

For this model, the mapping is as follows

- State S: x_t
- Action A: $a_t \sim \pi_t = \pi_\phi(\mathbf{x}, \mathbf{x}_t)$, actually we can ignore the initial input x.
- Reward $R(\cdot|x_t, a_t)$

$$r(\mathbf{x}_t, a_t; \mathbf{y}) := \begin{cases} -\frac{1}{\beta}\ell(\mathbf{y}, \mathbf{x}_t; \theta), a_t = 1(i.e. stop) \\ 0, a_t = 0(i.e. continue) \end{cases}$$

- State transition Probability: $P(x_{t+1}|x_t, a_t)$

$$P(\mathbf{x}_{t+1}|\mathbf{x}_t, a_t) := \begin{cases} 1, \mathbf{x}_{t+1} = \mathcal{F}_\theta(\mathbf{x}_t) \quad and \quad a_t = 0 \\ 0, else. \end{cases}$$

- Policy: $\pi_\phi(a_t|s_t)$

$$p(a_t|x_t) = \pi_t, where \quad \pi_t \sim \pi_\phi(x_t) \tag{9}$$

  ($\pi(t) \in [0, 1]$ is the probability of stopping at t and π_ϕ is the parametrized policy.)

Now, Let's talk about the trajectory. Suppose a trajectory:

$$(x, x_1, a_1, x_2, a_2, \cdots, x_t, a_t) \tag{10}$$

In this model, $a_t = 1$ while $a_\tau = 0$ where $1 \leq \tau < t$. Also, $x_1, x_2, \cdots, x_t$ are fixed. Therefore, a trajectory can be simply written as $a_{1:t} = (a_1 = 0, a_2 = 0, \cdots, a_t = 1)$. The probability of a trajectory is as follows:

$$p(a_{1:t}) = q_\phi(t) \tag{11}$$

The reward for a trajectory $a_{1:t}$:

$$\begin{aligned} r(a_{1:t}) &= \sum_{\tau=1}^{t} r(x_\tau, a_\tau; y) \\ &= r(x_t, a_t; y) \\ &= -\frac{1}{\beta}\ell(\mathbf{y}, \mathbf{x}_t; \theta) \end{aligned} \tag{12}$$

$t \sim q_\phi$ in equation (23) is equal to a trajectory distribution $a_{1:t} \sim q_\phi(t)$, where $q_\phi(t)$ is determined by the policy π_ϕ.

$$
\begin{aligned}
q_\phi(t) &= p(a_{1:t}) \\
&= \prod_{\tau=1}^{t} \pi_\tau(a_\tau|x_\tau) p(x_{\tau+1}|x_\tau, a_\tau) \\
&= \prod_{\tau=1}^{t} \pi_\tau(a_\tau|x_\tau)
\end{aligned} \tag{13}
$$

Then we can derive (25) from (24):

$$
\max_\phi \mathbb{E}_{t\sim q_\phi}[-\frac{1}{\beta}\ell(\mathbf{y}, \mathbf{x}_t; \theta) - \log q_\phi(t)] \tag{23}
$$

$$
= \max_\phi \mathbb{E}_{\pi_\phi} \sum_{\tau=1}^{t} [r(\mathbf{x}_\tau, a_\tau; \mathbf{y}) - \log \pi_\tau(a_\tau|\mathbf{x}, \mathbf{x}_\tau)] \tag{25}
$$

To derive (26) from (25), we need to prove the equivalence:

$$
\begin{aligned}
&\sum_{\tau=1}^{t} -\log \pi_\tau(a_\tau|\mathbf{x}, \mathbf{x}_\tau) \\
&= \sum_{\tau=1}^{t} H(\pi(\tau))
\end{aligned} \tag{14}
$$

Actually, the connection between Maximum entropy Reinforcement Learning and variational Bayes model comes from an interesting framework "control as inference". This framework analogizes Maximum entropy Reinforcement Learning as inference in a graphical model and has been used to justify many recent contributions in Inverse RL and Soft Q-learning.
In the original paper, there is also a derivation from $-\log q_\theta(a_t|s_t)$ to $H(q_\theta(\cdot|s_t))$, where q_θ also denotes the probability, which is conditioned on θ and θ can be weights in neural network.

$$
H(q_\theta(\cdot|s_t)) = -q_\theta(\cdot|s_t) \log q_\theta(\cdot|s_t) = \mathbb{E}_{q_\theta}(-\log q_\theta(\cdot|s_t)) \tag{15}
$$

We can do an unbiased point estimate on this problem. Point estimation involves the use of an observed value to estimate a parameter. In this condition, $-\log q_\theta(a_t|s_t)$ is an observed value and can be used to estimate the parameter $H(q_\theta(\cdot|s_t))$.
Take flipping a coin as an example: the probability of getting a head is a random variable and follows a Bernoulli distribution: $x \sim Bernoulli(p)$. $\mathbb{E}(x) = p$. Now we need to estimate the parameter p. Although it is a very bad estimation, we can do an experiment. Heads up and $p = 1$, otherwise $p = 0$.
Back to this paper,

$$
H(\pi_\tau) = -\pi_\tau(a_\tau|x, x_\tau) \log \pi_\tau(a_\tau|x, x_\tau) = \mathbb{E}_{a_\tau\sim\pi_\tau} - \log \pi_\tau(a_\tau|x, x_\tau) \tag{16}
$$

Then we use an observed value π_τ to estimate the entropy of policy. We can derive (26).

$$\max_{\phi} \mathbb{E}_{\pi_\phi} \sum_{\tau=1}^{t} [r(\mathbf{x}_\tau, a_\tau; \mathbf{y}) - \log \pi_\tau(a_\tau|\mathbf{x}, \mathbf{x}_\tau] \tag{25}$$

$$= \max_{\phi} \mathbb{E}_{\pi_\phi} \sum_{\tau=1}^{t} [r(\mathbf{x}_\tau, a_\tau; \mathbf{y}) + H(\pi_\tau(\cdot|\mathbf{x}, \mathbf{x}_\tau)] \tag{26}$$

I am not quite sure that whether we can use point estimation in this model or not. I appreciate it if you can give me some suggestions.
From author's response, there should be $H(\pi_\tau(\cdot|\mathbf{x}, \mathbf{x}_\tau))$ to be more clear.

3 Derivation of Loss of VAE:

First, let's describe the terms (p denotes a true probability and q denotes a approximated probability):

- $x \sim \tilde{p}(x)$: the true data x follows a distribution $\tilde{p}(x)$ (we will never know it but it is fixed and constant) ;
- z: the latent variable. The prior distribution is $z \sim q(z)$ and the posterior distribution is $z \sim q(z|x)$;

We can use latent variable z to represent the true data distribution $\tilde{p}(x)$.

$$q(x) = \int q(x, z)dz = \int q(z)q(x|z)dz \tag{17}$$

A more simple way is to approximate the joint distribution of x, z:

- True joint distribution: $p(x, z) = \tilde{p}(x)p(z|x)$
- Approximate distribution: $q(x, z) = q(z)q(x|z)$, where $q(x|z)$ denotes the generalization ability.

We use the joint distribution $q(x, z)$ to approximate $p(x, z)$. Then we need to minimize the KL divergence between this two joint distributions.

$$KL(p(x,z)\|q(x,z)) \tag{18}$$
$$= \iint p(x,z)\log\frac{p(x,z)}{q(x,z)}dzdx \tag{19}$$
$$= \int_{x\sim\tilde{p}(x)}\int \tilde{p}(x)p(z|x)\log\frac{\tilde{p}(x)p(z|x)}{q(x,z)}dzdx \tag{20}$$
$$= \int_{x\sim\tilde{p}(x)}\tilde{p}(x)\int p(z|x)\log\frac{\tilde{p}(x)p(z|x)}{q(x,z)}dzdx \tag{21}$$
$$= \mathbb{E}_{x\sim\tilde{p}(x)}\int p(z|x)\log\frac{\tilde{p}(x)p(z|x)}{q(x,z)}dz \tag{22}$$
$$= \mathbb{E}_{x\sim\tilde{p}(x)}[\int p(z|x)\log\tilde{p}(x)dz + \int p(z|x)\log\frac{p(z|x)}{q(x,z)}dz] \tag{23}$$
$$= \mathbb{E}_{x\sim\tilde{p}(x)}[\log\tilde{p}(x)\int p(z|x)dz + \int p(z|x)\log\frac{p(z|x)}{q(x,z)}dz] \tag{24}$$
$$= \mathbb{E}_{x\sim\tilde{p}(x)}[\log\tilde{p}(x) + \int p(z|x)\log\frac{p(z|x)}{q(z)q(x|z)}dz] \tag{25}$$
$$= \mathbb{E}_{x\sim\tilde{p}(x)}[\log\tilde{p}(x) + \int p(z|x)\log\frac{p(z|x)}{q(z)}dz - \int p(z|x)\log q(x|z)dz] \tag{26}$$
$$= \mathbb{E}_{x\sim\tilde{p}(x)}[\log\tilde{p}(x) + KL(p(z|x)\|q(z)) - \mathbb{E}_{z\sim p(z|x)}\log q(x|z)] \tag{27}$$
$$= \mathbb{E}_{x\sim\tilde{p}(x)}\log\tilde{p}(x) + \mathbb{E}_{x\sim\tilde{p}(x)}[\mathbb{E}_{z\sim p(z|x)} - \log q(x|z) + KL(p(z|x)\|q(z))] \tag{28}$$
$$= \mathbb{E}_{x\sim\tilde{p}(x)}\log\tilde{p}(x) + \mathbb{E}_{x\sim\tilde{p}(x)}[\mathbb{E}_{z\sim p(z|x)} - \log q(x|z) + KL(p(z|x)\|q(z))] \tag{29}$$

The blue part $\mathbb{E}_{x\sim\tilde{p}(x)}\log\tilde{p}(x)$ is a constant and minimize the KL divergence is equivalence to minimize the red part. Therefore, the loss function is:

$$\mathbb{E}_{z\sim p(z|x)} - \log q(x|z) + KL(p(z|x)\|q(z)) \tag{30}$$

Mapping to this paper, t is the latent variable z, $p(t|x)$ is the prior distribution $q(z)$ and $q_\phi(t)$ is the posterior probability $p(z|x)$. Then we can derive equation 6(in the paper) regardless of β. This is inspired by https://spaces.ac.cn/archives/5343, which is a blog in Chinese.

4 Sparse Recovery Review

Task: Recover $\mathbf{x}^* \in \mathbb{R}^n$ from its noisy linear measurements $\mathbf{b}$.

$$\mathbf{b} = A\mathbf{x}^* + \epsilon \tag{31}$$

where:

- Given: $\mathbf{b} \in \mathbb{R}^m$ is the noisy linear measurement
- Parameter: $A \in \mathbb{R}^{m\times n}$ and $m << n$

- $\epsilon \in \mathbb{R}^m$
- Target: $\hat{y} = \mathbf{x}^* \in \mathbb{R}^n$

It becomes easier to solve when $\mathbf{x}^*$ is sparse (I don't know why and you can find many math proofs on the Internet if interesting). We want to get the sparse signal $\mathbf{x}^*$ from $\mathbf{b}$. Therefore, it is a little bit like sparse coding problem.
In the sparse coding problem, we want to get a sparse representation $\mathbf{x} \in \mathbb{R}^n$ of $\mathbf{b} \in \mathbb{R}^m$ using an over-complete dictionary $A \in \mathbb{R}^{m \times n}$ and $m << n$, such that

$$\mathbf{b} \approx \mathbf{A}\mathbf{x} \tag{32}$$

A popular approach is to model the problem as the LASSO formulation:

$$\underset{\mathbf{x} \in \mathbb{R}^n}{min} \frac{1}{2}\|\mathbf{b} - A\mathbf{x}\|_2^2 + \lambda\|\mathbf{x}\|_1^1 \tag{33}$$

The iterative ISTA algorithm is as Algorithm 1:

Algorithm 1 ISTA

Input: $\mathbf{x}^0$
Output: $\mathbf{x}^L$
1: **for** $l = 0, 1, \ldots, L-1$ **do**
2: $\quad \mathbf{x}^{l+1} = S_\lambda((\mathbf{I} - \frac{1}{\mu}\mathbf{A}^T\mathbf{A})\mathbf{x}^l + \frac{1}{\mu}\mathbf{A}^T\mathbf{b})$
3: **end for**

Let $\mathbf{A}_t^2 = \mathbf{I} - \frac{1}{\mu}\mathbf{A}^T\mathbf{A}$ and $\mathbf{A}_t^1 = \frac{1}{\mu}\mathbf{A}^T$, then the iterative equation becomes:

$$\mathbf{x}^{l+1} = S_\lambda(\mathbf{A}_t^2\mathbf{x}^l + \mathbf{A}_t^1\mathbf{b}) \tag{34}$$

where soft-theresholding (λ is a hyper parameter):

$$S_\lambda(x) = sign(x) \cdot max\{|x| - \lambda, 0\} \tag{35}$$

Unrolled LISTA algorithm: one iteration of ISTA executes a linear and then a non-linear operation and thus can be recast into a network layer; by stacking the layers together a deep network is formed. In this model, we optimize $\theta = \{\lambda_t, \mathbf{W}_t^1, \mathbf{W}_t^2\}$ with real datasets and don't need knowledge on the dictionary $\mathbf{W}$. However, we can compute the $\mathbf{W}_0^1, \mathbf{W}_0^2$ according to $\mathbf{A}$ using the ISTA algorithm as the initial weight, which can speed up the convergence.
The architecture is as Figure 1 .
In LISTA model, the learning parameters are

$$\theta = \{\lambda_t, \mathbf{W}_t^1, \mathbf{W}_t^2\} \tag{36}$$

and the T-th layer network with update steps:

$$\mathbf{x}_t = \eta_{\lambda_t/L}(\mathbf{W}_t^2\mathbf{x}_{t-1} + \mathbf{W}_t^1\mathbf{b}), \quad t = 1, \cdots, T \tag{37}$$

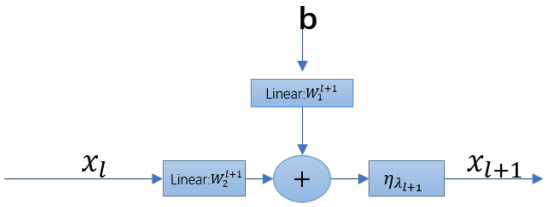

Figure 1: One Single Network Layer in LISTA

η_θ is the soft-thresholding function and L is usually taken as the largest eigenvalue of $\mathbf{A}^T\mathbf{A}$
The loss function is as follows(I don't find any information on the mentioned loss related to γ in the paper. If you find something, please let me know):

$$\ell(\mathbf{y}, \mathbf{x}_t; \theta) = \frac{1}{N}\sum_{n=1}^{N} \|\hat{\mathbf{y}}_t^n(b^n, \mathbf{x}_t^n; \theta) - \mathbf{y}^n\|_2^2 \tag{38}$$

The recovery performance is evaluated by NMSE (in dB), smaller is better:

$$NMSE(\hat{x}, x^*) = 10\log_{10}(\frac{\mathbb{E}\|\hat{x} - x^*\|_2^2}{\mathbb{E}\|x^*\|_2^2}) \tag{39}$$

The architecture is as Figure 2. The specifications are as follows:

- t: # of the stop layer ;
- x_t: the internal output of the predictive model;

$$\mathbf{x}_t = \eta_{\lambda_t/L}(\mathbf{W}_t^2\mathbf{x}_{t-1} + \mathbf{W}_t^1\mathbf{b}), \quad t = 1, \cdots, T \tag{40}$$

- y: the ground truth of recovery. $\hat{y}_t = x_t$: the predictive sparse for the $t-th$ layer;
- θ: parameters in the predictive model

$$\theta = \{\lambda_t, \mathbf{W}_t^1, \mathbf{W}_t^2\} \tag{41}$$

- ϕ: parameters in the policy network, specifically the weights in the shared Linear layers
- $p_\theta(y|t, x)$: the probability of t-layer's output is equal to the ground truth
- $\ell(y, x_t; \theta)$:

$$\ell(\mathbf{y}, \mathbf{x}_t; \theta) = \frac{1}{N}\sum_{n=1}^{N} \|\hat{\mathbf{y}}_t^n(b^n, \mathbf{x}_t^n; \theta) - \mathbf{y}^n\|_2^2 \tag{42}$$

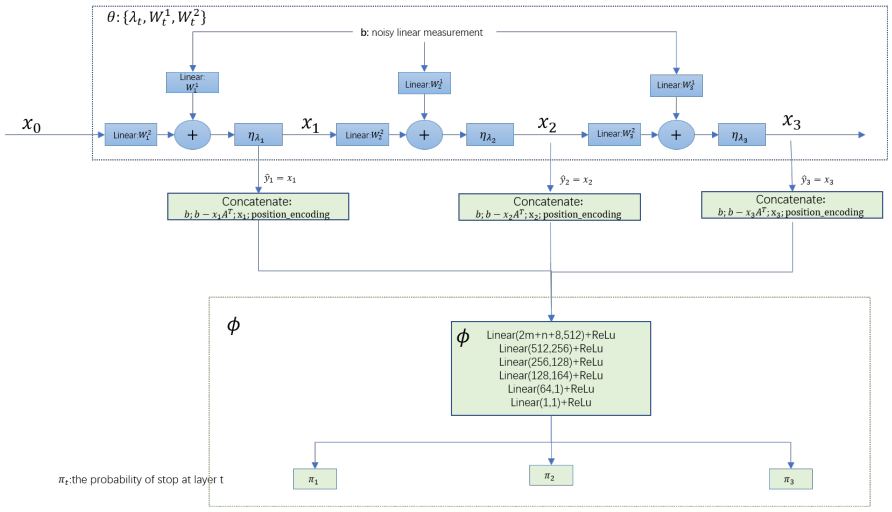

Figure 2: LISTA-stop

Q: Why the stop time is larger than 13/20 layer, while in SDN paper a good stop layer is much closer to front like 2/7 layer?
In the sparse recovery task, we use a simulation experiment. $x^* \in \mathbb{R}^{500}$ and it is a sparse vector where each of its entry to be non-zero following the Bernoulli distribution with $pb = 0.1$; $\epsilon \in \mathbb{R}^{250}$ is a Gaussian white noise. In the other hand, SDN paper uses image recognition task, whose input is 224 x 224 x 3 dimensions.
The difference between the tasks may be the reason why a SDN model can stop earlier than policy network.

5 Meta-Learning Review(Changbin)

Note: There is no too much details to explain how stop strategy is combined into MAML setting. The following description is based on their code implementation and my own understanding. Please let me know if you have any questions.
In addition to find the best layer for each sample in a deep neural network model, the l2stop strategy proposed in the paper also could be used to find the best number of iteration (gradient update steps) in the meta-learning setting, especially for the task-imbalanced meta learning.
In the original MAML setting, to achieve the fast adaptation, it use one or few gradient adaptation steps during the training. In practice, it uses 5 and 10 steps during meta-training and meta-testing stages for the miniImageNet experiment. In the MAML with stop strategy setting in the current paper, we predefine the total number of gradient adaptation steps, such as $T = 14$. And our goal is

to find the optimal gradient adaptation step among these 14 steps according to the current task.

Overview of MAML with stop. Figure.3 shows the overview of how the stop strategy is applied in the MAML setting during the training for each task (or a batch of tasks). In Figure. 3, Iter denotes the number of gradient adaptation steps (There are T steps in the figure). In practice, the model is always backbone (which is a CNN model with 4 convolutional layers, details could be found from MAML paper [1].) To explain it better, I plot the structure in a unrolling manner (from Iter:0 to Iter:T). For the backbone, the same color denotes the same parameters at that time. For example, in the Iter:0, after receiving the support set $\{X_s, y_s\}$, the backbone will update (the orange block). In the next iteration (Iter:1), the updated backbone (the orange block) will begin a new iteration.

For each iteration t,after recieving the support set, the model will output: $\{\ell_{tr}(t), Feat_{tr}(t), Pred_{tr}(t)\}$, where $\ell_{tr}(t)$ is the inner loss for this task, $Feat_{tr}(t)$ is the extracted features for support set, and $Pred_{tr}(t)$ is the predicted logits for support set.

After updating the backbone in the current iteration and receiving the query set, the model will output: $\{\ell_{ts}(t), Feat_{ts}(t), Pred_{ts}(t)\}$, where $\ell_{ts}(t)$ is the outer loss for this task, $Feat_{ts}(t)$ is the extracted features for query set, and $Pred_{ts}(t)$ is the predicted logits for query set.

Input of policy network. The input of the policy network would be:

- $\{(\ell_{tr}(t), Feat_{tr}(t), Pred_{tr}(t))|_{t=0}^{t=T}\}$
- $\{(Feat_{ts}(t), Pred_{ts}(t))|_{t=0}^{t=T}\}$
- the support set $\{X_s, y_s\}$ (images, and labels) and query samples X_q (only images, without label information)

It should be noted that the policy network will not take as input the label information of query set because there is no any label information when querying in practice. This is also the explanation why the outer loss $\ell_{ts}(t)$ is not included as the input.

Output of policy network. There are two parts in the output of policy network:

- The best iter, t_{pred}, namely the best number of gradient adaptation steps.
- The posterior distribution of stop time $pi(t)$.

Note: Figure. 3 is an example of one task during the training, which could be viewed as a batch of tasks as well in a straightforward way.

Structure of policy network. The structure of policy network used in the MAML setting is further explained in Figure. 4. Basically speaking, there would have a specific embedding function for each specific input of policy network. After embedding, the input of policy network will be concatenated into the final features. (please check the code for the detailed implementation of the embedding.)

The final features would be put into a network (basically a MLP) and generate scores (policy values). The score will be transformed into the stop time distribution $pi(t)$ and the predicted best number of gradient adaptation steps, t_{pred}.

How to update the policy network. In the implementation, the model (MAML with stop strategy) seems an end-to-end model generally speaking. For each epoch, it seems a 2-stage framework explained in the paper. The loss function is also different from the paper explanation. My thought is that this is because there are some simplification for the practice implementation as shown in Section 4.5 in the paper.

For each epoch, the final loss function is:

$$\ell_{ts}^* = \min_t \ell_{ts}(t;\theta) \tag{43}$$

$$\theta^*, \phi^* = \min_{\theta,\phi} pi(t^*;\phi) + \ell_{ts}^* \tag{44}$$

where θ is the meta-parameters in MAML, ϕ is the parameters in the policy network. $t^* = \arg\min \ell_{ts}(t)$.

$\ell_{ts}(t;\theta)$ is the outer loss function used in MAML:

$$\ell_{ts}(t;\theta) = \ell(\mathcal{A}lg_t(X_s;\theta), X_q) \tag{45}$$

where ℓ is the cross entropy. $\mathcal{A}lg_t(D_s;\theta)$ generates the updated model parameters (initialized as θ) after t steps according to support set $D_s = \{X_s, y_s\}$. Similarly, D_q is the query set.

Linlin: The specifications are as follows:

- t: # of gradient decent steps, $t^* = \arg\min \ell_{ts}(t)$;
- x_t: parameters in time step t;
- y: the ground truth of class label. $\hat{y}_t = x_t$: the predictive label of the model updated by the $t-th$ gradient decent step;
- θ: meta-parameters in MAML, specifically those weights in backbone;
- ϕ: parameters in the policy network;
- $p_\theta(y|t,x)$: the probability of predicted results generating from the model whose parameters has t's updated is equal to the ground truth
- $\ell(y, x_t;\theta)$: the experiment is an image recognition task and the final loss is entropy loss. The final target is to update initialize parameter using loss $\ell_{ts}(t;\theta)$, where

  $$\ell_{ts}(t;\theta) = \ell(\mathcal{A}lg_t(X_s;\theta), X_q) \tag{46}$$

  where ℓ is the cross entropy. $\mathcal{A}lg_t(D_s;\theta)$ generates the updated model parameters (initialized as θ) after t steps according to support set $D_s = \{X_s, y_s\}$. Similarly, D_q is the query set.

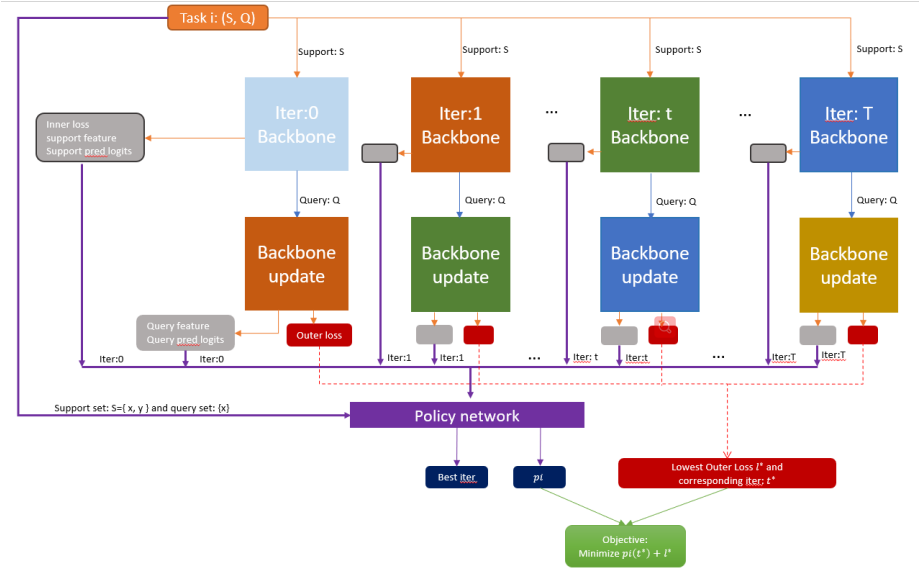

Figure 3: MAML with the optimal stop strategy

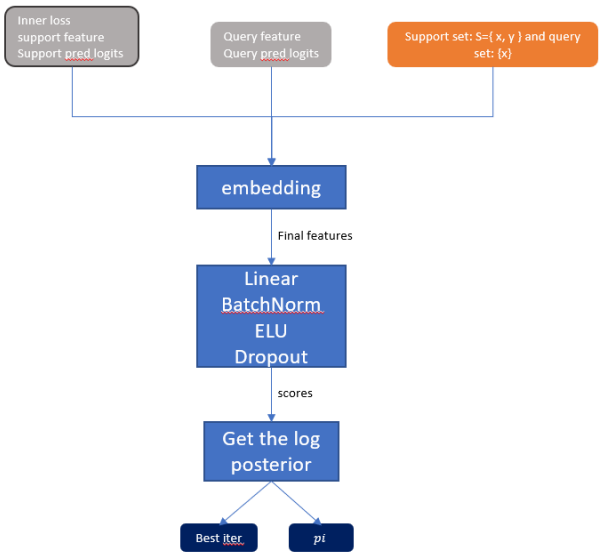

Figure 4: Policy Network used in the MAML setting

6 Image Denoising Review

Task: Recover a clean image $\mathbf{y}$ from a noisy observation $\mathbf{x}$.

$$\mathbf{x} = \mathbf{y} + \mathbf{v} \tag{47}$$

One common assumption is that $\mathbf{v}$ is additive white Gaussian noise (AWGN) with standard deviation σ

The most popular deep model for Image Dnoising Task is DnCNN. Modifying from VGG model, DnCNN uses residual learning and batch normalization layer, which can greatly benefit the CNN learning as they can not only speed up the training but also boost the denoising performance.

Traditional algorithm aims to learn a mapping function $\mathcal{F}(\mathbf{x}) = y$ to predict the latent clean image. For DnCNN, we adopt the residual learning formulation to train a residual mapping $\mathcal{R}(\mathbf{x}) \approx \mathbf{v}$ and then $\mathbf{y} = \mathbf{x} - \mathcal{R}(\mathbf{x})$.

Residual learning of CNN was originally proposed to solve the performance degradation problem in ResNet architecture, i.e., even the training accuracy begins to degrade along with the increasing of network depth. By assuming that the residual mapping is much easier to be learned than the original unreferenced mapping, residual network explicitly learns a residual mapping for a few stacked layers. With such a residual learning strategy, extremely deep CNN can be easily trained and improved accuracy has been achieved for image classification and object detection

The network shown in Fig. 1 can be used to train either the original mapping $\mathcal{F}(\mathbf{x})$ to predict $\mathbf{y}$ or the residual mapping $\mathcal{R}(\mathbf{x})$ to predict $\mathbf{v}$. According to ResNet, when the original mapping is more like an identity mapping, the residual mapping will be much easier to be optimized. Note that the noisy observation $\mathbf{x}$ is much more like the latent clean image $\mathbf{y}$ than the residual image $\mathbf{v}$ (especially when the noise level is low). Thus, $\mathcal{F}(\mathbf{x})$ would be more close to an identity mapping than $\mathcal{R}(\mathbf{x})$, and the residual learning formulation is more suitable for image denoising.

Formally, the averaged mean squared error between the desired residual images and estimated ones from noisy input

$$\ell(\theta) = \frac{1}{2N} \sum_{i=1}^{N} \|\mathcal{R}(\mathbf{x}_i; \theta) - (\mathbf{x}_i - \mathbf{y}_i)\|_2^2 \tag{48}$$

can be adopted as the loss function to learn the trainable parameters θ with $\{(\mathbf{y}_i - \mathbf{x}_i)\}_{i=1}^{N}$ represents N noisy-clean training pairs.

The detailed architecture is as Figure 5:

The hyperparameters are as Table 1 and Table 2.

	#layer		Output Dimension
DnCNN		Input	W x H x 3
	1	Conv+ReLu	W x H x 64
	2-16	Conv+BN+ReLu	W x H x 64
	17	Conv	W x H x 3

Table 1: Architecture of DnCNN

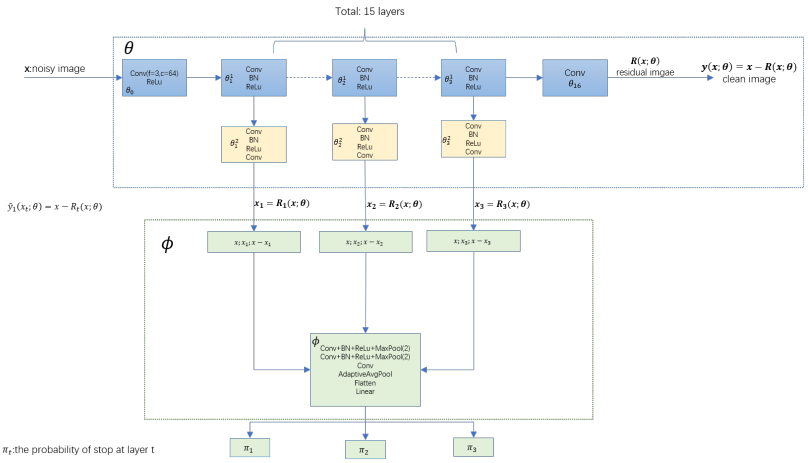

Figure 5: DnCNN-stop Model

Policy Network	#layer	Output Dimension
Sperate part	Input	W x H x 64
	Conv+BN+ReLu	W x H x 64
	Conv	W x H x 3
Cat Operation	$\mathbf{y}$; R($\mathbf{y}$;θ); ($\mathbf{y} - R(\mathbf{y};\theta)$)	W x H x 3
Shared part	conv2d+BN+ReLu+Maxpool2d(2)	W/2 x H/2 x 64
	conv2d+BN+ReLu+Maxpool2d(2)	W/4 x H/4 x 64
	conv2d	W/4 x H/4 x 64
	AdaptiveAvgPool2d(1)	1 x 1 x 64
	Flatten	64
	Linear	1

Table 2: Policy Network of DnCNN-stop

The performance is evaluated by the PSNR, which is used to represent the goodness of the reconstruction and lower is better.

- t:# of stop layer in the DnCNN model
- x_t: Internal output of predictive model: $x_t = f_{\theta_t^1}(x_{t-1})$; x_0 is the noisy image;
- y: the clean image and $y_t = x - f_{\theta_t^2}(x_t)$;
- θ: parameters in predictive model, $\theta = \{\theta^1, \theta^2\}$;
- ϕ: parameters in the policy network;
- $p_\theta(y|t, x)$: the probability of t-layer's prediction is equal to the ground truth

- $\ell(y, x_t; \theta)$:

$$\ell(y, x_t; \theta) = \frac{1}{2N} \sum_{i=1}^{N} \|y_t^i(\mathbf{x}_t^i; \theta) - (\mathbf{x}^i - \mathbf{y}_t^i)\|_2^2 \tag{49}$$

7 Image Recognition

Dataset: The Tiny ImageNet dataset consists of a subset of ImageNet images, re-sized at 64x64 pixels and RGB. There are 200 classes
For Comparion, we provide the detailed architecture of VGG-16-SDN is as table 3.

Table 3: VGG-16-SDN Model under the TinyImage Dataset

# layer	VGG-16	"Conv" Output dimension	IC	FR's Input/Output dimension	full's Input/ Output dimension
	input (64 ×64 × 3)		0/1		
1	conv3-64	64×64 × 64	0		
2	conv3-64	64×64 × 64	1	64×64 × 64/4 × 4 × 64	1024/200
	maxpool	32×32 × 64			
3	conv3-128	32×32 × 128	0		
4	conv3-128	32×32 × 128	1	32×32 × 128/4 × 4 × 128	2048/200
	maxpool	16×16 × 128			
5	conv3-256	16×16 × 256	0		
6	conv3-256	16×16 × 256	1	16×16 × 256/4 × 4 × 256	4096/200
7	conv3-256	16 ×16 × 258	1	16×16 × 256/4 × 4 × 256	4096/200
	maxpool	8×8 × 256			
8	conv3-512	8×8 × 512	0		
9	conv3-512	8×8 × 512	1	8×8 × 512/4 × 4 × 512	8192/200
10	conv3-512	8×8 × 512	1	8×8 × 512/4 × 4 × 512	8192/200
	maxpool	4×4 × 512			
11	conv3-512	4×4 × 512	0		
12	conv3-512	4×4 × 512	0		
13	conv3-512	4×4 × 512	0		
	maxpool	2×2 × 512			
	flatten	2048	0		
14	FC-4096	2048	0		
15	FC-4096	2048	0		
16	FC-1000	200	0		
	softmax	200			

Different architectures of policy network used in the image recognition in l2stop paper although the final results are not better than SDN results including "MulticlassNetImage", "MNIconfidence" and "imiconfidence". We take MulticlassNetImage as an example and the detailed architecture is as Figure 6:

1.) MulticlassNetImage

x_module
conv2d: 3×3, padding=1
BatchNorm2d
ReLU
AdaptiveAvgPool2d
Flatten()
Linear()

For each internal classifier:

last layer
Linear

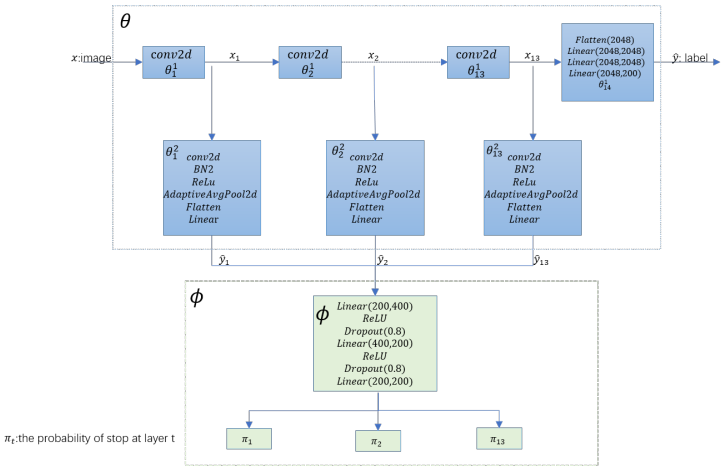

Figure 6: VGG-STOP Model

shared_module_fm
Linear(*, 400)
ReLU
Dropout(0.8)
Linear(400, 200)
ReLU
Dropout(0.8)

Policy Network structure (7 modules concatenated)
7 x_module
shared_module_fm
Last layer

The specifications:

- t: of internal layer in VGG-16 model;
- x_t: The internal prediction;
- y: the ground truth and $\hat{y}$ is the predictive label;
- θ: parameters in original predictive model, specifically is the parameters in VGG-16 model;
- ϕ: parameters in the policy network;
- $p_\theta(y|t,x)$: the probability of successfully predict the label in the t-th layer;
- $\ell(y, x_t; \theta)$:

$$\ell(y, x_t; \theta) = \frac{1}{2N} \sum_{i=1}^{N} -\log p_\theta(y|t,x) \tag{50}$$

8 Unrolled Algorithm Review

Unrolled algorithm: It was first proposed to develop fast neural network approximations for sparse coding. The main idea is to build a connection between the iterative algorithm and DNN to get a combined performance. Traditional methods are mainly designed by analyzing the physical processes and handcrafting, while DNN wants to model to automatically discover the pattern from the training set. Unrolled algorithms take the iterative algorithm as a prior knowledge and incorporate it into the neural networks, when using the strategy mapping one iteration to one layer. Passing through the network is equivalent to executing the iterative algorithm a finite number of times.

What are the advantages of unrolled algorithms over rolled algorithms?

I don't find any definitions of "rolled algorithm". The advantage of unrolled algorithms is compared with traditional iterative algorithms and deep neural networks. Traditional iterative algorithm

Advantage comparing with iterative methods:

Computation savings the number of layers in a deep network is typically much smaller than the number of iterations required in an iterative algorithm. Therefore, the inference cost of unrolled algorithms is smaller than iterative models.

Performance Improvement: we can get a better performance since we can get some coefficients or parameters which are difficult to hand-design. And also the training process can discover a better pattern than hand-analyzing. Dynamic parameters: in traditional iterative algorithms, some coefficients are fixed and all the iterative steps share the same value. However, in unrolled algorithm, it becomes the learning parameters that can be changing during the training process and values may vary across different layers, which extend the representation capacity.

Advantage comparing with DNN:

Interpretability: traditional iterative algorithms are usually highly interpretable because they are developed via modeling the physical processes underlying the problem and/or capturing prior domain knowledge. Unrolled algorithms also have this prior knowledge leading to a better interpretability.

Generalizability: DNN is overly dependent on the quantity and quality of training data available. However, in some areas such as medical imaging, data is relevant to privacy and security and hard to get enough. As a prior information guided deep network, unrolled algorithms can do a good job with less training data and have a better generalization ability.

9 Other questions

1. How to define the goodness of a meta-learning algorithm?

   For each task in the training, validation and testing stage, we have training examples and testing examples. We need to update the parameters both

in the learning algorithm such as an initial parameter value and the task-specific parameters such as the weights in a convolution network. During the training stage, one task denotes one training instance and can get one loss of the learning algorithm by updating the parameters in the learning algorithm. In contrast, one training example in one training task can get a loss for updating the task-specific parameters. In detail, we use the training examples in one training task to train the task-specific network and use the testing examples to get one loss for learning algorithm. All tasks in the training tasks can define the training loss of the learning algorithm. In the testing stage, we still need to use the training example in testing task to update the task-specific parameters, and used this trained model to test the testing samples and get the testing loss of the learning algorithms.

2. More information on MAML

   MAML automatically finds initialization parameters that it thinks the parameters are good for all tasks and still uses gradient descent learning framework.

   We define a loss function of the learning algorithm and the parameter is the initialization parameter ϕ. θ^n is the model parameters updated from ϕ using gradient descent in task n which apparently depends on ϕ. $\ell^n(\theta^n)$ is the loss of task n on the testing set of task n. Now we have N tasks and we use the same initialization parameters and get the loss using the final learned parameters for each task and do a summation. then it is the loss function of ϕ.

   In practice, MAML suppose that the training algorithm only updates parameters once in the training phase, It means the model learned ϕ and computes the gradient to get θ^n , which is considered as the final parameters. So it is simple to update θ^n with a learning rate , while updating θ with learning rate η.

   Why can we only consider updating once.There are four main reasons. The first one is fast and we need many tasks and each task has multiple examples to learn. The second reason is from the result, because it is good if we can get a good result with only one-time updating. In addition, we can still update multiple times when we use the model, which means in the test step, we can update multiple times to get a better performance. Finally, meta-learning is always mentioned with few-shot learning, which has limited data. In this situation, deeper layers result in an overfitting problem.

   Meta-learning is different from pre-training methods. The right figure in 25 is a good example. We use one dimension to denote the model parameters. These two green lines denote two different task's losses.This figure measures the relation between model parameters and loss of tasks. For MAML, we don't care about the performance of ϕ on the training task, however, what we care about is the performance of final parameters

n trained on ϕ. For example, this ϕ is not good for both task 1 and task 2, but it may be a good initialization parameter. Because, ϕ is an initialization and theta needs to update based on ϕ. so we can see for both task 1 and task 2, it can easily arrive at the optimal value according to the direction of gradient. In conclusion, we want to find a phi which can achieve good performance after training

3. Disentanglement:

   it can be simply thought of as a dimension reduction, such as PCA(Principal Component Analysis) and MF(Matrix Factorization). In the real world, the data always contains high-dimension information and we need to disentangle it to a low-dimensional representation to capture some patterns. For example, one photo of human faces contains multiple dimensions of information, such as the gentle skin color, hair color, hair length, emotion. What a classification model needs to get a representation to decide whether wearing a pair of glasses. So it needs to disentangle the information and many other relatively independent factors in separate dimensions. Such a disentangled representation is very beneficial to facial image generation.

   In beta-VAE, we have the loss:

$$\mathcal{J}_{\beta-VAE}(\theta, q_\phi; \mathbf{x}, \mathbf{y}) := \mathbb{E}_{q_\phi} \log p_\theta(\mathbf{y}|t, \mathbf{x}) - \beta \mathbf{KL}(q_\phi(t) \| p(t|\mathbf{x})) \tag{51}$$

   Where the first component is the reconstruction loss and the second component is latent capacity (the more different between the posterior and the prior, the more information latent variable has learnt). Beta is used to make a balance between these two components. Therefore a higher beta encourages more efficient latent encoding and further encourages the disentanglement.

References

[1] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, pages 1126–1135. PMLR, 2017.