Paper summary

Unsupervised Sound Separation Using Mixture of Mixtures

→ ในการทำ wav-sepatation ปกติ จะทำตามรูปที่ 1.(a)

ใช้ SNR-loss

→ เราต้องเรียนรู้จาก clean-source เท่านั้น

→ โดยใช้ Permutation-matrix ในการเทรน model (P-matrix)

examaple P-matrix

$$P_{1 \Leftrightarrow 2} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

swap $1 \Leftrightarrow 2$:

P-mat นั้น คือ I และ swap roll

→ ส่วนที่ paper นี้ต้องการจะทำคือ

→ ต้องการทำ separate sound จาก mixsound 2 sound ตามรูป 1(b)

→ จะใช้หลักการเดียวกับรูปแบบปกติ

โดย A-mat (mixing-mat)

ใช้รูปแบบเดียวกับ P-mat

(แต่เราต้องคิดทุก Combination)

$O(2^M)$

→ โดยที่เงื่อนไขในการแยก นี้คือ

ห้ามเกิดเหตุการณ์

Signal A $\propto$ Signal B

เพราะ model มันจะเป็น signal รวมกัน

แล้วแยกไม่ได้

โดยอธิบายละเอียดได้จาก

session 4.2 col 3

→ จะเห็นได้ว่า mat B ที่ทำให้

$s^* = B\hat{s}$, $A = A^* B$ (A^* คือ optimal A)

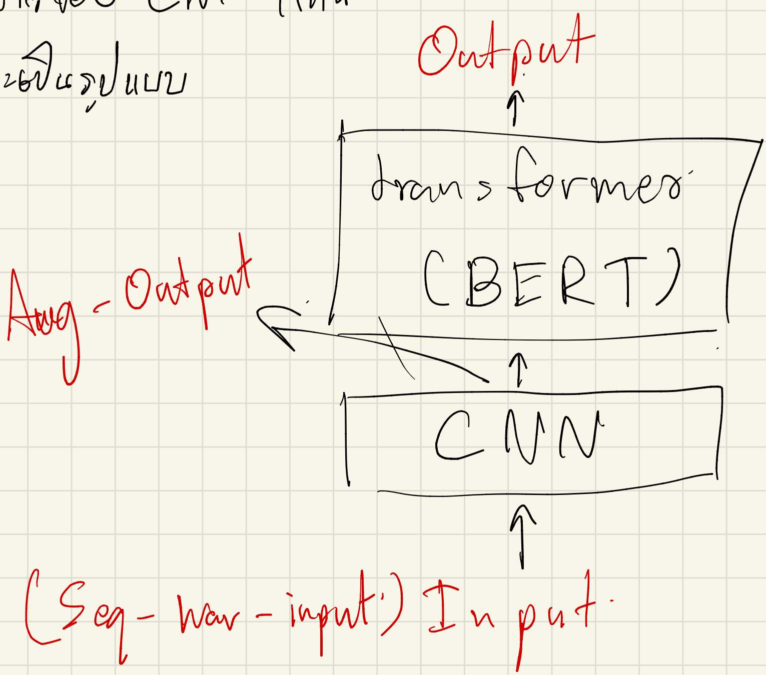
โดย B นี้สามารถมีได้หลายตัว ไม่ unique.

Wav2Vec

→ ต้องการทำ representation ของ wav-sound

→ เนื่องจาก ข้อมูลของ sound มีไว้ ข้อมูลที่
และความเกี่ยวข้องของเสียงนั้น จะไม่ ควรเกี่ยวข้อง
ในระยะไกลมาก ดังนั้นการ จึงไม่ต้องใช้
Seq-encoding (sin(...) ⇒ cos(...) ⇒) [ดูใน - Attention is all you need]
เปลี่ยนเป็นตัวของ CNN แทน

→ model จะเป็นรูปแบบ



⇒ Loss ในการทำ representation

1. Quantization module
   - → output แบบ softmax ได้ (Avg-Output)
   - → เป็นการบอก class แบบเชื่อว่า

$p_{q_0 v}$

$\Downarrow$

L_d

2. Contrastive loss ⇒ L_m
   - → เปรียบเทียบความเหมือนของ output

3. $L_f = L_2$- loss ตาม encode-CNN
   - → ใช้ว่าใน image L_2-loss Avg
     มีใช้ผลในการช่วยให้ข้อมูลเท่า ๆ กลุ่มกัน

Fast differentiable Sorting and Ranking

→ การ sorting กับ ranking เป็นการทำ algo ปกติ ใช้เวลา $O(n\log n)$

→ ปกติ การ sorting, ranking เป็น non-differentiable → ไม่เหมาะกับ ML

→ ใน paper ก่อนหน้านี้ สามารถทำการ sorting ที่ differentiable แต่ยังกินเวลา $O(n^2)$ แต่ ranking → ยังเป็น non-differentiable อยู่

→ สิ่งที่ paper นี้ต้องการทำคือการ ทำให้ ต้องการ ทำ sorting, ranking ที่ differentiable โดยที่ยังใช้ $O(n\log n)$

→ จะหา ปัญหา sorting, กับ ranking ให้เป็น formular ต่อเนื่องกัน ในการเปลี่ยน input

→ การเปลี่ยน ปัญหา sorting และ ranking ต้องเป็นแบบ ของ linear programing (LP)

parameter in this paper

ตัวแปร ~~ในpaper~~

$\theta \Rightarrow$ parameters $\Rightarrow \theta = g(x)$ model ก่อน sort ← input model

$\sigma \Rightarrow$ permutation $\sigma = (\sigma_1, \dots, \sigma_n)$

argsort

$\pi \Rightarrow$ inverse ของ $\sigma \Rightarrow \pi := \sigma^{-1}$

โดยที่ $\sigma(\theta) = (\sigma_1(\theta), \dots, \sigma_n(\theta))$ (จากมากไปน้อย)

และ $\theta_{\sigma_1(\theta)} \geq \dots \geq \theta_{\sigma_n(\theta)}$

$s(\theta) := \theta_{\sigma(\theta)}$ (sorting)

$r(\theta) := \sigma^{-1}(\theta)$

example $\theta_3 \geq \theta_1 \geq \theta_2$

จะได้ $\sigma(\theta) = (3, 1, 2)$, $s(\theta) = (\theta_3, \theta_1, \theta_2)$,

$r(\theta) = (2, 1, 3)$

ให้ $\Sigma_n \Rightarrow$ setของ Permutation $(1, \dots, n)$

อย่างเช่น $n=3$

$$\Sigma_3 = \{(1,2,3), (1,3,2), (2,1,3), (2,3,1), (3,1,2), (3,2,1)\}$$

$\mathcal{P}(\theta) \Rightarrow$ set Permutation $(\theta_1, \dots, \theta_n)$

เช่น $\theta_{ex} = (\theta_1, \theta_2, \theta_3)$

$$\mathcal{P}(\theta_{ex}) = \{(\theta_1,\theta_2,\theta_3), (\theta_1,\theta_3,\theta_2), (\theta_2,\theta_1,\theta_3), (\theta_2,\theta_3,\theta_1), (\theta_3,\theta_1,\theta_2), (\theta_3,\theta_2,\theta_1)\}$$

$$\boxed{\rho := (n, n-1, \dots, 1)}$$

เราจะสามารถหา $\sigma(\theta), r(\theta)$ มาจาก LP ตาม $\boxed{\text{Lemma 1}}$

$$\sigma(\theta) = \underset{\sigma \in \Sigma}{\arg\max} \langle \theta_\sigma, \rho \rangle$$

inner-product

$$r(\theta) = \underset{\pi \in \Sigma}{\arg\max} \langle \theta, \rho_\pi \rangle$$

$(1, 2, \dots, n)$

และ

$$S(\theta) = \underset{y \in P(\theta)}{\arg\max} \langle y, \rho \rangle$$

$$r(\theta) = \underset{y \in P(\rho)}{\arg\max} \langle y, -\theta \rangle$$

→ เนื่องจากปัญหาที่เรา แปลงเป็นรูปแบบ ของ LP

→ คุณสมบัติ ของ LP นี้คือ คำตอบที่ได้ จะอยู่ใน vertex (จุดมุม) ของ convex hull (จาก LP ในของ ม.ปลาย)

→ ตามรูป Figure 1

Differentiability a.e. of sorting

→ เราจะได้ ถ้า $\sigma(\theta)$ unique at θ.

$s(\theta) = \theta_{\sigma(\theta)}$ จะ dif ได้ โดยได้ Jacobian mat เป็น Permutation mat.

→ แต่ ranking ต้อง dif ไม่ได้อยู่ (discrete space)

เราจะแก้ปัญหา ที่ 2 พร้อมกัน โดยใช้ z, w เป็นตัวแปรในการแก้

ปัญหา $s(\theta) \Rightarrow (z, w) = (\rho, \theta)$

และ $r(\theta) \Rightarrow (z, w) = (-\theta, \rho)$

โดยการ optimation นี้ เราจะใช้ regularization เพื่อเพิ่มความง่ายต่อการ optimization.

ตามสมการ

P_Q และ P_E

$\Downarrow$ sorting $\Downarrow$ ranking

→ ส่วนที่ 2 ที่ต้องการ ช่วยในการ optimization
มีเรื่อง soft operator (with φ-regulation)

$$S_\varepsilon \underline{\varphi}(\theta) : P_{\varepsilon\varphi}(\rho, \theta) = P_\varphi(\rho/\varepsilon, \theta)$$

$$r_\varepsilon \varphi(\theta) = P_\varphi(-\theta, \rho) = P_\varphi(-\theta/\varepsilon, \rho)$$

เราจะได้ส่วนที่ 2 จะได้คำตอบ นั่นเราจะได้
คำตอบ ที่ project มาจาก optimal
โดยที่ เมื่อเรา convert กับ on projection
นั้นจะได้คำตอบจริง

ผลดีของ soft-operator
นั้นจะได้ตาม Figure 3

~~ผลดี~~ $\boxed{\text{สิ่งที่ได้และต้องทำ}}$
→ Convexification

→ On tuning ε

→ relation to linear assignment formulation

→ ปัญหาที่ตอนต้นมาเนี่ยสามารถหาคำตอบ
ในรูปของ isotonic optimization

→ กรณีที่มันเป็น isotonic

→ สามารถเขียน formulation

ของ solution ในรูปแบบ exact sol.

ได้โดย

ได้ $\quad$ Sol ของ $(\theta_1, \dots, \theta_n) = f(\theta_1, \text{Sol}(\theta_1, \dots, \theta_{n-1}))$

→ อ่านต่อใน paper

Beyond Accuracy : NLP CheckList

→ ปัญหาในการ test task NLP ต่างๆ นั้นคือ
เพราะ: เชื่อใน test-set เกินไป
test-set อาจไม่ได้มีความสอดคล้องกับการใช้จริง
ทำให้เรา over-estimat model perfomance
ของมัน (test-set จำกัด และอาจมี distibution
ไม่สอดคล้องกับการใช้จริง)

→ จากมุมมองของ SE (software engineer)
จะเห็นว่าเราไม่สามารถทำ test-case ที่ดีพอ
→ ใน paper นี้จึงได้ออกแบบ
(<u>checklist</u>) เป็นเครื่องมือในการทำ test-case
ให้กับ model NLP ต่างๆ

→ เราสามารถทำ test-case ได้แบบ A, B, C
ตาม Figure 1
(รายละเอียดอ่าน paper) Figuer 2

→ ทำให้สรุปผลของ model ได้ตาม

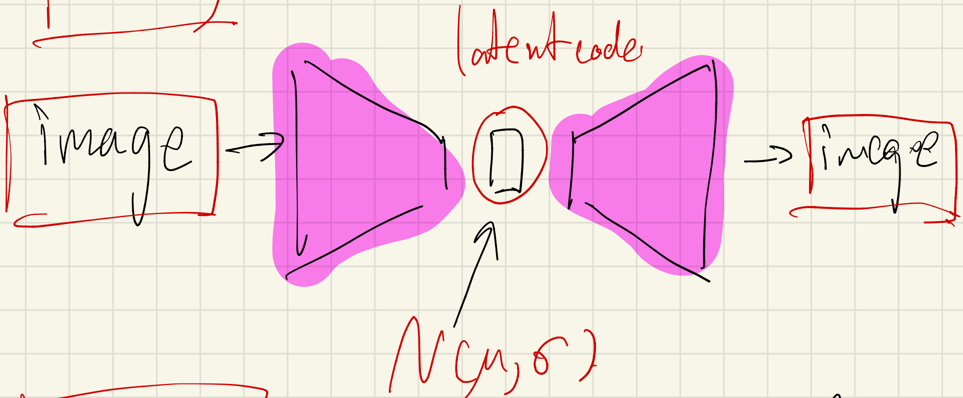
table 1, table 2, table 3

จะเห็นว่า model NLP หลายตัวก็มีปัญหา

Closed-Form Factorization of Latent Semantics in GAN:

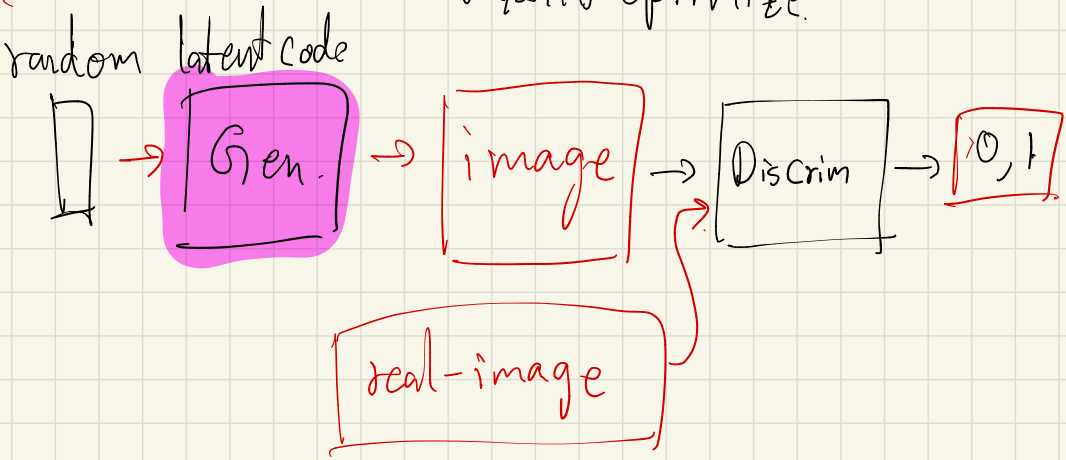
→ การทำ model genaration มีรูปแบบอยู่ 2 แบบ คือ VAE, GAN

VAE



GAN

→ อาจจะมี Loss ต่างๆ ที่ไม่ออก, ง่าย ต่อการ optimize



→ โดยเนื่องจาก VAE นั้น สามารถ ทำ vector แบบ separate ในการ แบ่งส่วนต่างๆ ได้ดีมาก ได้ (ตาม β-VAE)

→ แต่ตามจริง GAN ก็พอทำได้ แต่เนื่องจาก model GAN ไม่ได้อยู่ใน formulation ที่ สามารถ ได้ง่าย แต่พอทำได้ ตาม Figure 1

เราจะสามารถ หา element ของ vector ที่อยู่ไปได้

→ รูปแบบของการ Generation ภาพจาก GAN ตามรูป Figure 2

* [มีอธิบาย เพิ่มเติมในการต่อที่แหล่ง]

ตัวของ AdaIN และ Style GAN

→ ref form จาก paper $\boxed{\text{AdaIN}}$

ในการทำ style transfer นี้

จาก Batch Normalization $x \in \mathbb{R}^{N \times C \times H \times W}$

$$BN(x) = \gamma\left(\frac{x - \mu(x)}{\sigma(x)}\right) + \beta$$

Fig A1 ตามสมการ

Instance Normalization

$$IN(x) = \gamma\left(\frac{x - \mu(x)}{\sigma(x)}\right) + \beta$$

เปลี่ยน γ, β ตามสมการ

Fig A2

CIN : Conditional Instance Normalization

$$CIN(x;s) = \gamma^s \left(\frac{x-\mu(x)}{\sigma(x)}\right) + \beta^s$$

เราสามารถเปลี่ยนค่า γ^s, β^s ในการทำ style-tran ต่างๆ ในการ Generate ภาพ

→ แต่ว่า เราก็ต้องหา γ^s, β^s ในแต่ละ class

$(\mu(x), \sigma(x)) \Rightarrow$ คำนวณ ตาม $IN(x)$ เหมือนกัน

ต่อเรื่อง style GAN

ตาม Figure B1 [ใส่ γ^s, β^s เป็น layer ๆ]

ต่อ Closed-Form GAN

จาก $I = G'(FC(z)) \triangleq G'(y)$

โดย $y = FC(z)$

และ ∴ $z' = z + \alpha\eta$ ∴ โดย η นั้นคือ element vector

∴ Assumetion ในการหา η

ถ้า $\alpha\eta$ เปลี่ยนมาก y จะต้องเปลี่ยนมาก

$\Downarrow$

proof eq 3, 4, 5, 6, 7 $\Rightarrow$ โดยที่

$$y = FC(z) = Az + b$$

จะได้ว่า η นั้นคือ eigen-vector ของ A^TA

โดย sorting ตาม λ ของ A^TA

(นั่นคือ PCA, single value decomposition)

ผลดูตามใน paper

Adversarial Latent Autoencoder

→ AE สามารถทำ representation ให้กับ
(Auto-encoder) Latent vector ได้ดี

→ แต่ มักไม่ได้ เรื่องการ blur ของภาพ

→ GAN → การทำ Gen ได้ดี แต่ มี representation
ที่ไม่ค่อยดี

∴ พยายามเอา 2 ตัว AE & GAN
เป็น ALAE (Auto Latent Auto-encoder)
ตามรูป Figure 1
(ในการทำ Gen, Inferent)

ตัว GAN เราแบ่ง

$$G_{new} = G \circ F \quad \text{และ} \quad D_{new} = D \circ E$$

~~ซึ่ง~~ ตาม Figure 1 มีการ Gen

→ space ในส่วนที่ทำการช่วยในการ Inference
เป็น space เดียวกัน

→ ข้อดี

→ ไม่จำเป็นต้องใช้การ training ใหม่
ในการ Gen รูปแบบที่คล้ายกัน

ข้อ ex Figure 9

→ สามารถทำแบบ styleGAN
(style ALAE) ตาม Figure 2

[ผลดู ตาม paper จริง]

→ On learning Set of Symmetric Elements

(ต่อ จาก DSS (deepset paper)

$\boxed{\text{ผลที่ใช้ และได้ dss-layer ขึ้นมา}}$

→ ทวนความ เรื่อง deep-set paper.

Q → deepset ต้องการสร้าง model ที่มีการ

invar-ในการ swap-input.

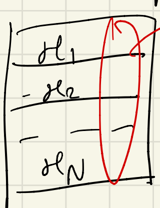
$f(x_1, x_2) = f(x_2, x_1)$ เพื่อเรียนรู้ความเป็น set

ของตัว input ของ model ออกมาด้วย

→ บทสรุป ของ มัน จะอยู่ใน รูปแบบ ของ summation

$$\text{Deepset}(\{x_1, \ldots, x_N\}) = \text{MLP}\left(\frac{1}{N}\sum_{i=1}^{N} \text{MLP}(x_i)\right)$$

→ แต่สิ่งที่ paper นี้ต้องการ นำเสนอ



มีการ invar ในบางส่วนของ แนวข้างต่อ

ชื่อ: ไทย Layer → ไปตามแบบรูป DSS-layer

→ ตามรูป Figure 1.

→ example ปัญหา ที่อยากใช้ envonment แบบนี้

→ มีหลายรูป ~~ที่~~ ⇒ (รูปถ่าย หลายมุม)

⇓

สร้างรูป 3D ออกมา.

⇒ de-blur ⇒ รูป blur ~~หลาย~~ หลายรูป ⇒ สร้างรูปที่ไม่ blur.

key word Inv

$$\boxed{A} \rightarrow f$$

$$\boxed{B} \rightarrow f$$

เหมือน

equi-var

$$f(g\,\boxed{A}) = g\,f(\boxed{A})$$

→

→ โจทย์เรื่อง Symmetric Groups เปิด link youtube

$$S_n = \{ (1, 2, 3, \ldots, n),\ (1, 3, 2, 4, \ldots, n) \}$$

$n!$

group ของการ permutation

Symm

→ แปะ link wiki เรื่อง Group theory ด้วย

S

~~สรุป~~

Charactoric paper ML.

① show data → learn fn

② in-side (ใน) data ⇒ เสริมความสามารถ fn

ใส่ Hypothesis บางอย่าง

ใน eq(1)

$$L(X_1) = L_1^{H}(X$$

จำกัดความ symmetry บางอย่าง

L^H จะมีคุณสมบัติ พิเศษ $\boxed{H \leq S_d}$

(H) บางแบบที่สลับได้

H คือ บางแบบ

เปลี่ยน $L \Rightarrow L^H$ ก็พอแล้ว

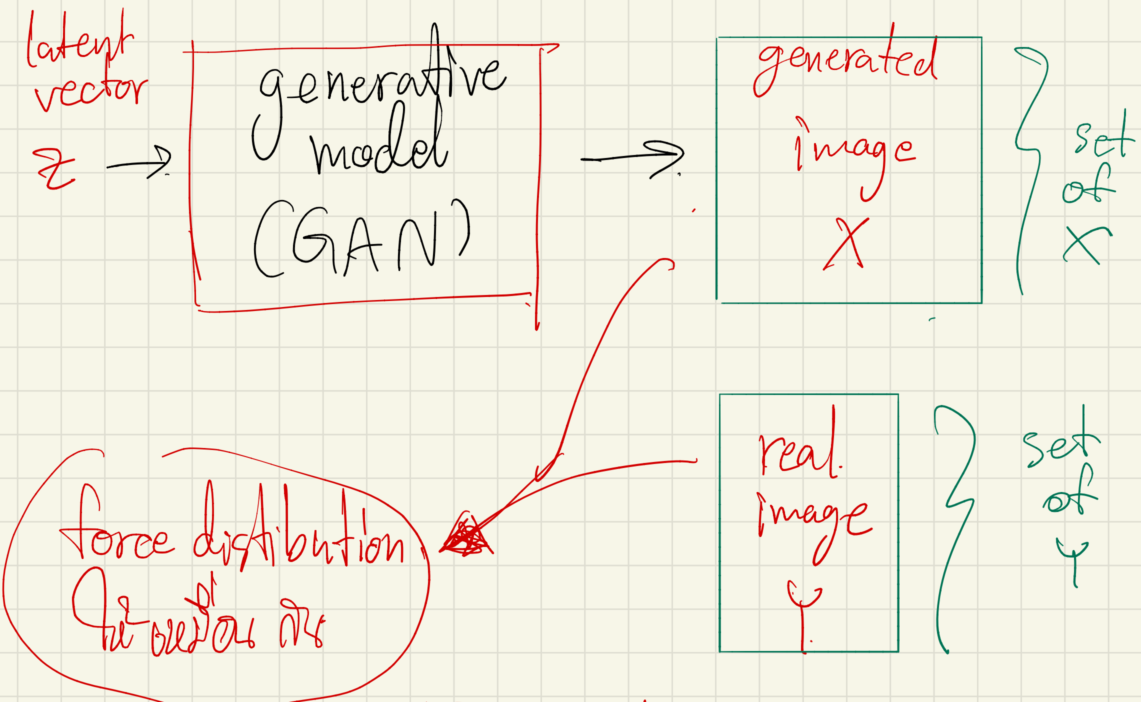
→ ที่เหลือเป็น ทฤษฎี ง่ายมาก → video

อธิบายเพิ่ม

Kernel Mean Matching for Content Addressability of GAN

→ paper นี้ต้องการนำเสนอ วิธีการใช้ GAN ในการ generate ภาพ ที่แม่น

→ โดย



โดยใช้ MMD

Figure 1

MMD (Maximum Mean Discrepancy)

→ มีทฤษฎีที่ว่า P, Q distibution

$\|\mu_P - \mu_Q\|_H$ ← main fold.

⇓

แค่ mean

ไม่ใช่ mean

~~แต่~~ แค่ ตัวแทน

แต่อาจจะมีหลาย moment

(จาก moment gen f^n)

(infite-dim)

→ การ force นี้ จะสามารถกำหนด weight ได้ ตาม eq (1;2)

→ แล้ว เรา จะได้ ความสัมพันธ์ ของมันได้ว่า

eq (3)

$$Y_n^* = \operatorname*{argmin}_{\{y_1, \dots, y_n\}} \widehat{MMD}(X_m, Y_n, w)$$

~~จากสมการ เราจะ~~ $\therefore$

จากสมการ eq (1,2,3,4,5)

จะเห็นว่า เราต้องการ kernel บางอย่างเพื่อ force สิ่งที่เราต้องการ

โดยที่เราเลือก kernel ของเราคือ

$$K(x,y) := k(E(x), E(y))$$

$E \Rightarrow$ คือ feature extractor (VGG, ...)

$k \Rightarrow$ non-linear kernel [Guassian kernel, Polynomial, ...]

ผลที่ได้ก็จะเห็นตาม

Figure 2, 3, 4, 5

Making Convolution Network Shift-Invar Again

→ แนวคิดจาก signal processing

→ การ stride มากกว่า 1. เป็นเหมือนการ downsampling สัญญาณ ⇒ ความถี่อาจจะเกิน ความถี่ Nyquist. (

→ เกิดการเสีย signal (aliasing)

~~ตาม~~ มุมมอง signal processing.

→ [Low-pass] → [down sample] → use.

⇓ ใส่ใน CNN ด้วย

Blurpool (Conv-layer)

(ภาพจะลื่นเนียนขึ้น, video)

⇒ มุมมองของ CV (Figure 5)

→ จากการ stride ลงผลต่อการ model
(shift) จาก Figure 1

example signal [0, 0, 1, 1, 0, 0, 1, 1]

ถ้าสิ่งที่ทำ maxpool-stride 2

→ shift = 0 ⇒ ได้ [0, 1, 0, 1] (Figure 4)

shift = 1 ⇒ ได้ [1, 1, 1, 1] → ผิด

สาเหตุ จากด้านเทคนิค (มุมมอง signal processing)

⇒ การเปลี่ยน layer → Figure 2

ดู: Figure 6, 7

Unsupervised Translation of Programming Language

→ เป็นโจทย์ที่พยายแปลง ภาษา programing จาก A → B

→ เช่น การ เขียน ให้ โค้ดที่ผ่านสามารถทำงานกับ Compiler ได้เหมือนเดิม

→ model รูปแบบที่ใช้ จะเป็น XML (Microsoft) (Tranformer)

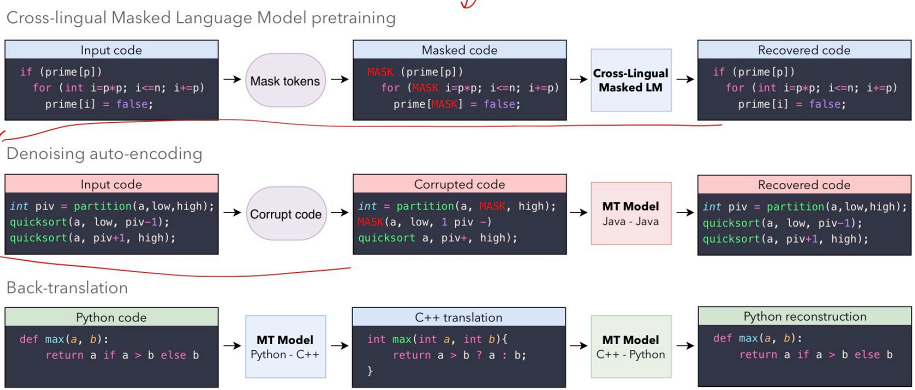
และจะใช้ รูปแบบ การ pretrain 3 แบบ หลัก ๆ

→ 1). Cross-lingual Masked (ใช้ Mask ค่า แล้วแปลงกลับ เป็น ตัวเดิม)

2). Denoising auto-encoding
→ ใส่ noise ค่าผิด, หรือ เปลี่ยน Mask
→ กลับมา ค่าเดิม

3) Back-translation (แปลไปกลับเลย เหมือนเดิม

Mask = Bert



recover

Back MT

Figure 1: **Illustration of the three principles of unsupervised machine translation used by our approach.** The first principle initializes the model with cross-lingual masked language model pretraining. As a result, pieces of code that express the same instructions are mapped to the same representation, regardless of the programming language. Denoising auto-encoding, the second principle, trains the decoder to always generate valid sequences, even when fed with noisy data, and increases the encoder robustness to input noise. Back-translation, the last principle, allows the model to generate parallel data which can be used for training. Whenever the Python → C++ model becomes better, it generates more accurate data for the C++ → Python model, and vice versa. Figure 5 in the appendix provides a representation of the cross-lingual embeddings we obtain after training.

pretain model Anchor point

The cross-lingual nature of the resulting model comes from the significant number of common tokens (anchor points) that exist across languages. In the context of English-French translation, the anchor points consists essentially of digits and city and people names. In programming languages, these anchor points come from common keywords (e.g. for, while, if, try), and also digits

~~ข้อ~~ <u>เสริม</u>

จากการทดลองเกี่ยวกับ MT หลายๆ นั้น
การ Augment ด้วย Back-translation นั้นสามารถ
ทำการ Augment ได้ดีที่สุด

→ อาจจะมีเหตุผลจากการ Force ด้วย
translation จริงๆ

→ ทำให้ Back-translation ดูเหมือนการ
Force ด้วยการ supervised training
อีกส่วนหนึ่ง (มีความใกล้กับความเป็น supervised)

Analysis.

ในการทำ MT นั้นจะมีสิ่งที่น่าสนใจในการ
แปลงจากภาษา A → B ได้ดี (หลายภาษา)

$A \to B$, $A \to C$, $A \to D$

จะมี Anchor point ในการช่วยในการแปล

Anchor point ⇒ คือ คำที่ใช้ร่วมในภาษา หลาย
ภาษา เช่น ชื่อสถานที่, ชื่อคน etc.

rare word มาก

โดย ในการทำ ของ programming นั้น มักมีการใช้
for, if, while, try ⟶ ร่วมกันอยู่แล้ว
(Anchor point สามารถพบได้ทั่วไป)

→ อาจจะแก้ปัญหา แก้ rare word โดยใช้ Bi pair encoding (BPE)
(ไม่ใช่เรื่องใหม่)

วัดผล ถึงงานนี้จะเป็นงาน ใน ของ MT
แต่การวัดโดยใช้ Bleu score นั้น ไม่ค่อย เหมาะสม
เนื่องจาก Bleu เน้นวัดในการวัดรอบข้าง เท่านั้น
ซึ่ง โดย ความเป็น programming จะมี structer ที่ชัดเจน
อยู่แล้ว จึง ไม่น่าจะเหมาะ

→ ผลลัพธ์ จึงต้องมาวัดผลโดยการใช้ compiler ว่า
สามารถทำงานได้หรือไม่

→ ส่วนที่ยาก จริงๆ ของงานนี้ ที่ผ่านมาที่ได้ยาก

python → Java , python → C++

เนื่องจากมันเป็นการ แปลง จาก ภาษาที่มี type dynamic (python)

เป็น ภาษาที่มี type static (C, C++, Java)

Dynamic, static คือ ดูว่าเราต้อง fix type int, str, float หรือไม่ ?

Pixel-Adaptive Convolution NN.

→ model ที่นำเสนอนี้ จะเป็นตามรูป Figure 1

→ โดยแนวคิดคือ อยากให้ weight ใน Conv นั้นสามารถ เปลี่ยนค่าไปตามช่วงต่างๆได้

→ แต่ model ได้ทำอย่างนี้โดย model จะใช้ Input "f" ที่บางอย่าง ที่ Raw แล้วมาสร้าง guideline ให้กับตัว Conv โดยผ่าน kernel บางอย่าง (ใน paper ใช้ตัวของ gaussian kernel

→ เป็นแนวคิดคล้ายกับการสร้าง Attention มาจาก input f

$$K(f_1,f_2) = \exp\left(-\frac{1}{2}\|f_1-f_2\|_2^2\right)$$

(เป็นตัวย่อยของ rbf

$$k(x_1,x_2) = \exp\left(-\gamma\|x_1-x_2\|^2\right) = \exp\left(-\frac{1}{2}\frac{\|x_1-x_2\|^2}{\sigma^2}\right)$$

→ Task ที่ทำมาเลือกใช้ เป็น task upsampling

โดย upsampling depth pic (ทำ upsampling ภาพ) และใช้ guideline เป็นรูป ของที่ ขยาย (ใช้แนวคิด จากรูปปกติ)

เปรียบเทียบ การใช้ kernel หลายแบบ

ใช้ model Figure 2

io Gallo[2], Erik Learned-Miller[1], and Jan Kautz[2]

erst [2]NVIDIA

original CNN

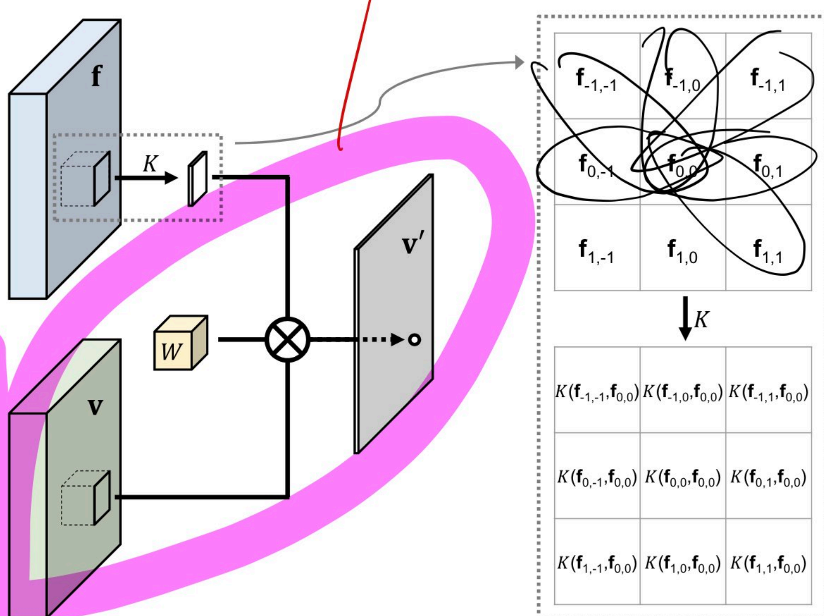


Figure 1: **Pixel-Adaptive Convolution.** PAC modifies a standard convolution on an input $\mathbf{v}$ by modifying the spatially invariant filter $\mathbf{W}$ with an adapting kernel K. The adapting kernel is constructed using either pre-defined or learned features $\mathbf{f}$. $\otimes$ denotes element-wise multiplication of matrices followed by a summation. Only one output channel is shown for the illustration.

the optimal gradient direction for parameters differs at each pixel. However, due to the spatial sharing nature of convoweights $\mathbf{W} \in \mathbb{R}^{\dots}$ can be written as

ตำแหน่ง

input ของ pixel

$$\mathbf{v}'_i = \sum_{j \in \Omega(i)} \mathbf{W}\left[\mathbf{p}_i - \mathbf{p}_j\right]\mathbf{v}_j + \mathbf{b} \qquad (1$$

CNN-weight

Original ConV

$$\mathbf{v}'_i = \sum_{j \in \Omega(i)} K\left(\mathbf{f}_i, \mathbf{f}_j\right) \mathbf{W}\left[\mathbf{p}_i - \mathbf{p}_j\right] \mathbf{v}_j + \mathbf{b} \quad (3)$$

new PAC

where K is a kernel function that has a fixed parametric form such as Gaussian: $K(\mathbf{f}_i, \mathbf{f}_j) = \exp(-\frac{1}{2}(\mathbf{f}_i - \mathbf{f}_j)^\intercal(\mathbf{f}_i - \mathbf{f}_j))$. Since K has a pre-defined form and is not param-

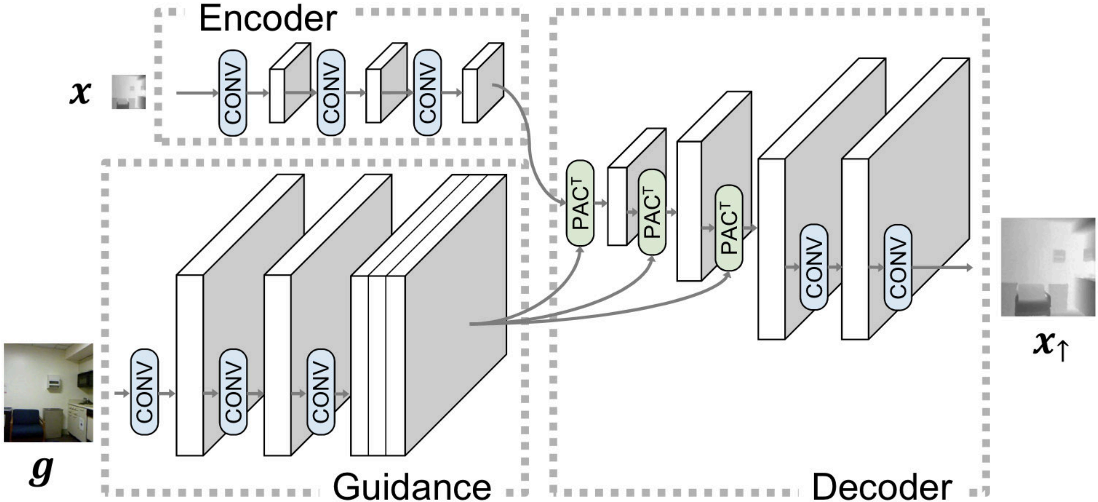


Figure 2: **Joint upsampling with PAC.** Network architecture showing encoder, guidance and decoder components. Features from the guidance branch are used to adapt PAC$^{\mathsf{T}}$ kernels that are applied on the encoder output resulting in upsampled signal.

ต่อ สามารถ ทำ model มาใช้ ที่ผลดีกับ semetic segmentation ด้วย โดยใช้ CRF (CV ใช้ เทคนิคนี้ สมัยก่อน ในการทำ โดยใช้ kernel ที่ดีขึ้นเข้าไป ในตัวของ Full-CRF ได้เยอะ ในส่วน

Full-CRF นี้ เป็น paper แยก ไปอ่านแยก (เป็น CRF แต่คิด ทั้งหมด ทุก pixel

→ โดย ผลจะได้ ตามรูป Figure 5

→ สิ่งที่ไม่ค่อยดี ของ paper นี้ คือ base line ที่ใช้ ที่ใช้ยังไม่ สามารถเทียบผลกับ state of the art จริงๆ เท่านั้น ที่สามารถทำได้ดีในปีนั้น มากๆแล้ว

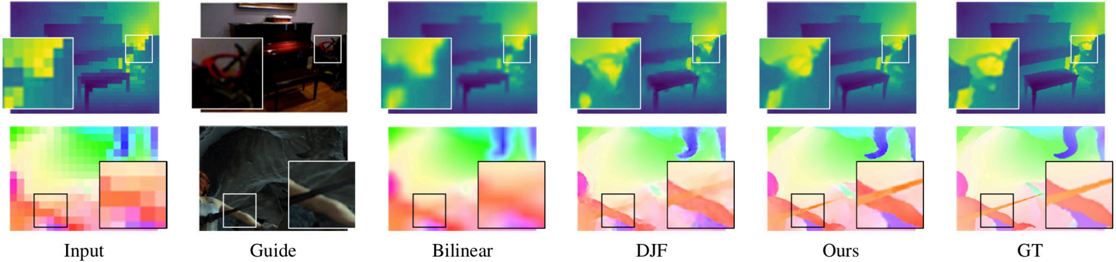


Figure 3: **Deep joint upsampling.** Results of different methods for 16× joint depth upsampling (top row) and 16× joint optical flow upsampling (bottom row). Our method produces results that have more details and are more faithful to the edges in the guidance image.

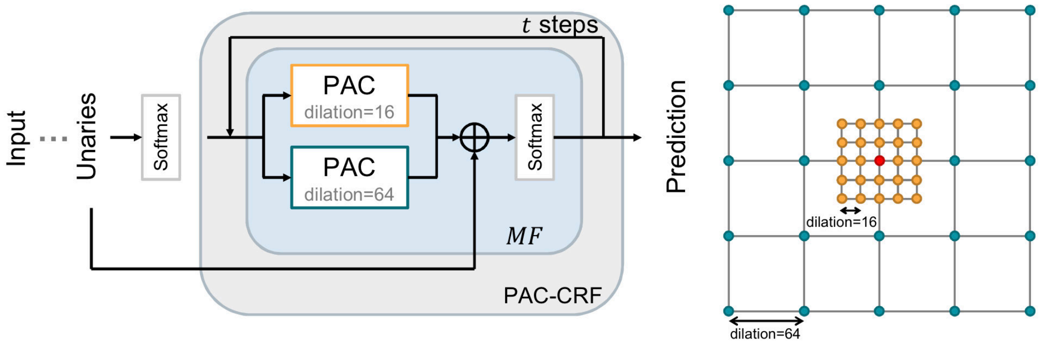


Figure 4: **PAC-CRF.** Illustration of inputs, outputs and the operations in each mean-field (MF) step of PAC-CRF inference. Also shown is the coverage of two 5×5 PAC filters, with dilation factors 16 and 64 respectively.

$$K(\mathbf{f}_i, \mathbf{f}_j) = w_1 \exp\left\{-\frac{\|\mathbf{p}_i - \mathbf{p}_j\|^2}{2\theta_\alpha^2} - \frac{\|I_i - I_j\|^2}{2\theta_\beta^2}\right\} + w_2 \exp\left\{-\frac{\|\mathbf{p}_i - \mathbf{p}_j\|^2}{2\theta_\gamma^2}\right\} \tag{4}$$

where $w_1, w_2, \theta_\alpha, \theta_\beta, \theta_\gamma$ are model parameters, and are typically found by a grid-search. Then, inference in Full-CRF amounts to maximizing the following Gibbs distribution: $P(\mathbf{l}|I) = \exp(-\sum_i \psi_u(l_i) - \sum_{i<j} \psi_p(l_i, l_j))$, $\mathbf{l} = (l_1, l_2, ..., l_n)$. Exact inference of Full-CRF is hard, and [25] relies on mean-field approximation which is optimizing for an approximate distribution $Q(\mathbf{l}) = \prod_i Q_i(l_i)$ by minimizing the KL-divergence between $P(\mathbf{l}|I)$ and the mean-field approximation $Q(\mathbf{l})$. This leads to the following mean-field (MF) inference step that updates marginal distributions Q_i iteratively for $t = 0, 1, ...$:

$$Q_i^{(t+1)}(l) \leftarrow \frac{1}{Z_i} \exp\Big\{ - \psi_u(l) - \sum_{l' \in \mathcal{L}} \mu(l, l') \sum_{j \neq i} K(\mathbf{f}_i, \mathbf{f}_j) Q_j^{(t)}(l') \Big\} \tag{5}$$

PAC-CRF. In PAC-CRF, we define pairwise connections over fixed windows Ω^k around each pixel instead of dense connections: $\sum_k \sum_i \sum_{j \in \Omega^k(i)} \psi_p^k(l_i, l_j | I)$, where the k-th pairwise potential is defined as

$$\psi_p^k(l_i, l_j | I) = K^k(\mathbf{f}_i, \mathbf{f}_j) \mathbf{W}^k_{l_j l_i}[\mathbf{p}_j - \mathbf{p}_i] \tag{6}$$

Here $\Omega^k(\cdot)$ specifies the pairwise connection pattern of the k-th pairwise potential originated from each pixel, and K^k is a fixed Gaussian kernel. Intuitively, this formulation allows the label compatibility transform μ in Full-CRF to be modeled by $\mathbf{W}$, and to vary across different spatial offsets. Similar derivation as in Full-CRF yields the following iterative MF update rule (see appendix for more details):

$$Q_i^{(t+1)}(l) \leftarrow \frac{1}{Z_i} \exp \Bigg\{ -\psi_u(l) -$$

$$\sum_k \underbrace{\sum_{l' \in \mathcal{L}} \sum_{j \in \Omega^k(i)} K^k(\mathbf{f}_i, \mathbf{f}_j) \mathbf{W}^k_{l' l}[\mathbf{p}_j - \mathbf{p}_i] Q_j^{(t)}(l')}_{\text{PAC}} \Bigg\} \tag{7}$$

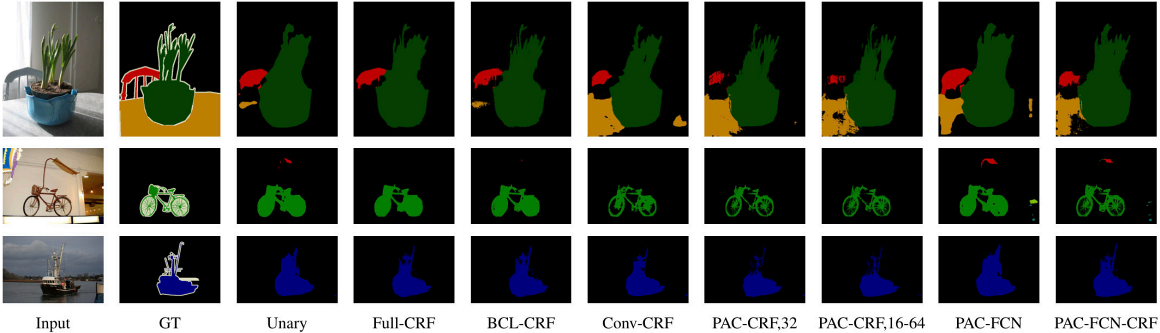


Figure 5: **Semantic segmentation with PAC-CRF and PAC-FCN.** We show three examples from the validation set. Compared to Full-CRF [25], BCL-CRF [21], and Conv-CRF [41], PAC-CRF can recover finer details faithful to the boundaries in the RGB inputs.

Outlier Exposure with Confidence Control for Out-of-Distribution Detection

ใน ปกติ ในการ ทำ model classification
ต้อง จะ fit ให้ สิบ ข้อมูล ใน class ที่ต้องการ
model ต้อง กำหนด และ ทำงาน ได้ ไม่ มาก ถ้า หาก model
มี dataset ที่ ไม่ มี class

$x \to$ model $\xrightarrow{\text{(softmax)}}$ y (class)

OOD $\to$ model $\to$ มั่วมาก (noisy)

→ สิ่งที่ เขา นำเสนอ คือ วิธีการ train รูปแบบใหม่
ที่เป็นการเปลี่ยน loss ของ model

→ ทำได้ทั้งใน image, text classification

setting

ปกติ แบ่ง Data เป็น

Data → train
Data → val
Data → train

สิ่งที่ mode แบ่งและ เพิ่ม OOD

→ train - train
→ train - val
+ OOD

ในการ train model จะได้ obj ตามรูป eq.1

และสามารถแปลงให้อยู่ แบบ Largian

ได้ตาม สมการที่ 2. ⇒ ~~[illegible]~~ แต่ว่าทำให้ผลิตยาก เลยเปลี่ยน

obj → พจน์ train-train

formulation ใหม่

λ_1 → พจน์ train - val

(เทียบกับ A_{tr})

Acc - ที่ลดลงไป

λ_2 → force เพื่อจับ OOD

cross-entropy (train-train)

$$\underset{\boldsymbol{\theta}}{\text{minimize}} \quad \mathbb{E}_{(x,y)\sim D_{in}}[\mathcal{L}_{CE}(f_{\boldsymbol{\theta}}(x), y)]$$

acc - model

$$\text{subject to} \quad \mathbb{E}_{x\sim D_{in}}\left[\max_{l=1,\ldots,K}\left(\frac{e^{z_l}}{\sum_{j=1}^{K} e^{z_j}}\right)\right] = A_{tr}$$

~~cros~~ (train-val)

$$\max_{l=1,\ldots,K}\left(\frac{e^{z_l}}{\sum_{j=1}^{K} e^{z_j}}\right) = \frac{1}{K}, \; \forall x^{(i)} \sim D_{out}^{OE}$$

OOD

(

where $\mathcal{L}_{CE}$ is the cross-entropy loss function and K is the number of classes available in D_{in}. Even though the constrained optimization problem (1) can be used for training various classification models, for clarity we limit our discussion to deep neural networks. Let $\mathbf{z}$ denote the vector representation of the example $x^{(i)}$ in the feature space produced by the last layer of the deep neural network (DNN) and let A_{tr} be the training accuracy of the DNN. Observe that the optimization problem (1) minimizes the cross entropy loss function subject to two additional constraints. The first constraint forces the average maximum prediction probabilities calculated by the softmax layer towards the training accuracy of the DNN for examples sampled from D_{in}, while the second constraint forces the maximum probability calculated by the softmax layer towards $\frac{1}{K}$ for all examples sampled from the probability distribution D_{out}^{OE}. In other words, the first constraint makes the DNN predict examples from known classes with an average confidence close to its training accuracy, while the second constraint forces the DNN to be highly uncertain for examples of classes it has never seen before by producing a uniform distribution at the output for examples sampled from the probability distribution D_{out}^{OE}. It is also worth noting that the first constraint of (1) uses the training accuracy of the neural network A_{tr} which is not available in general. To handle this issue, one can train a neural network by only minimizing the cross-entropy loss function for a few number of epochs in order to estimate A_{tr} and then fine-tune it using (1).

$$
\begin{aligned}
\underset{\boldsymbol{\theta}}{\text{minimize}}\ & \mathbb{E}_{(x,y)\sim D_{in}}\left[\mathcal{L}_{CE}(f_{\boldsymbol{\theta}}(x), y)\right] \\
& + \lambda_1 \left(A_{tr} - \mathbb{E}_{x\sim D_{in}}\left[\max_{l=1,\ldots,K} \left(\frac{e^{z_l}}{\sum_{j=1}^{K} e^{z_j}} \right) \right] \right) \\
& + \lambda_2 \sum_{x^{(i)}\sim D_{out}^{OE}} \left(\frac{1}{K} - \underline{\max_{l=1,\ldots,K} \left(\frac{e^{z_l}}{\sum_{j=1}^{K} e^{z_j}} \right)} \right)
\end{aligned}
\quad (2)
$$

grage form.

แทน
เป็น ไม่ม

l_2-norm

train-val

$$
\begin{aligned}
\underset{\boldsymbol{\theta}}{\text{minimize}} \ & \mathbb{E}_{(x,y)\sim D_{in}}[\mathcal{L}_{CE}(f_{\boldsymbol{\theta}}(x), y)] \\
& + \lambda_1 \left(A_{tr} - \mathbb{E}_{x\sim D_{in}} \left[\max_{l=1,\ldots,K} \left(\frac{e^{z_l}}{\sum_{j=1}^{K} e^{z_j}} \right) \right] \right)^2 \\
& + \lambda_2 \sum_{x^{(i)}\sim D_{out}^{OE}} \sum_{l=1}^{K} \left| \frac{1}{K} - \frac{e^{z_l}}{\sum_{j=1}^{K} e^{z_j}} \right|
\end{aligned} \tag{3}
$$

ไม่ต้องรู้ outlier α

น่าจะเกิด outlier

OOD

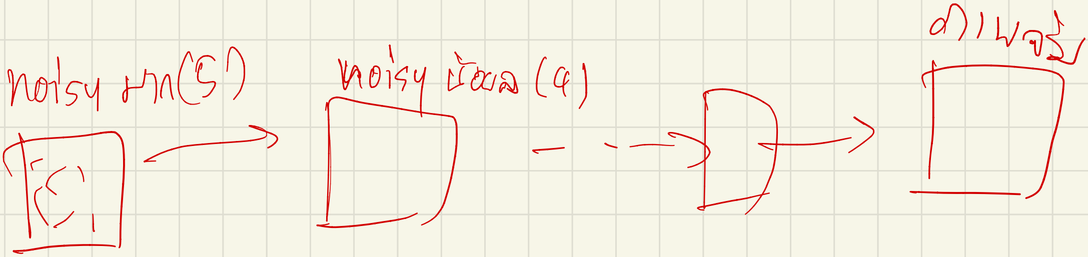
l_1

D_{in}	λ_1	λ_2	FPR95↓	AUROC↑	AUPR↑	Test Accuracy(D_{in})
CIFAR-10	-	-	34.94	89.27	59.16	94.65
	-	✓	8.87	96.72	77.65	92.72
	✓	✓	6.56	98.40	93.08	93.86
CIFAR-100	-	-	62.66	73.11	30.05	75.73
	-	✓	26.75	91.59	68.27	71.29
	✓	✓	28.89	91.80	71.50	73.14

Denoising Diffusion Probabilistic Models.

→ เป็น model generate ภาพ

แนวคิดในเรื่องของ thermodynamic เข้ามาเกี่ยวข้อง



มีแนวคิดจาก paper.

Deep Unsupervised Learning using Non-equilibrium Thermodynamics.

⇓

จาก paper ในแนว physics พยายามอธิบายว่าถ้าเราเก็บการแพร่นั้น เราพอที่จะสามารถ reconstruct มันออกมาเป็นอย่างเดิมได้ จาก ภาพปกติ กับ noisy data สุดๆ

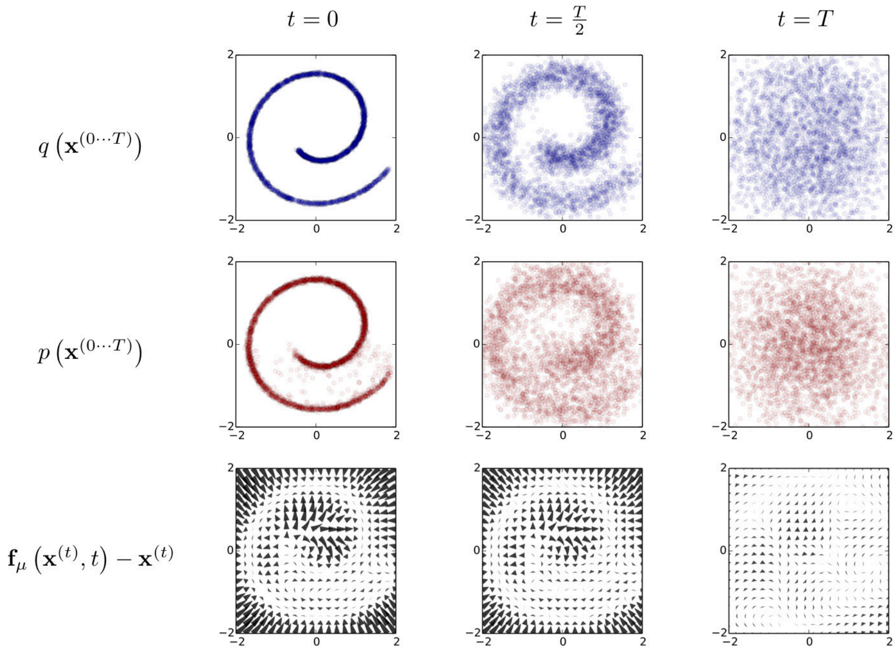


Figure 1. The proposed modeling framework trained on 2-d swiss roll data. The top row shows time slices from the forward trajectory $q\left(\mathbf{x}^{(0\cdots T)}\right)$. The data distribution (left) undergoes Gaussian diffusion, which gradually transforms it into an identity-covariance Gaussian (right). The middle row shows the corresponding time slices from the trained reverse trajectory $p\left(\mathbf{x}^{(0\cdots T)}\right)$. An identity-covariance Gaussian (right) undergoes a Gaussian diffusion process with learned mean and covariance functions, and is gradually transformed back into the data distribution (left). The bottom row shows the drift term, $\mathbf{f}_\mu\left(\mathbf{x}^{(t)}, t\right) - \mathbf{x}^{(t)}$, for the same reverse diffusion process.

เปิดตามรูปด้านบน

ถ้าเราคิดว่าการแพร่ในเวลาสั้นๆ เหลือเกิน process

ของแบบนี้ เราสามารถย้อนตัวเริ่มต้นขึ้นมาได้

→ สิ่งนั้น paper คือใช้แนวคิดในการทำลาย

โดยการใส่ noise เข้าไป แต่เลือกที่จะใส่

ทีละ step น้อยๆ ตามรูป

→ part ของ Algorithm

จะแบ่งเป็น forward part

(ทำลายภาพ)

→ reverse part

(reconstruction)

อารมณ์จะเป็น

TENET

555+

→ ข้อดีของการแยกเป็นส่วนย่อย

↳ เราสามารถคิด Conditional

prop จากปลายทั้ง 2 แล้วหา

ระหว่างกลางได้

→ มีความ stable ! (จากตัว paper

ไม่มีข้อพิสูจน์)

$q(x^0)$ ต้องการสร้ we have

ต้องการสร้าง

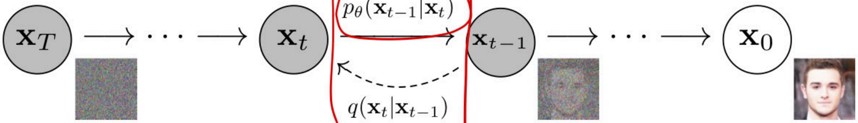


Figure 2: The directed graphical model considered in this work.

This paper presents progress in diffusion probabilistic models [50]. A diffusion probabilistic model (which we will call a "diffusion model" for brevity) is a parameterized Markov chain trained using variational inference to produce samples matching the data after finite time. Transitions of this chain are learned to reverse a diffusion process, which is a Markov chain that gradually adds noise to the data in the opposite direction of sampling until signal is destroyed. When the diffusion consists of small amounts of Gaussian noise, it is sufficient to set the sampling chain transitions to conditional Gaussians too, allowing for a particularly simple neural network parameterization.

→ สามารถใช้ model - Autoencoder (U-net) ได้

+ form คณิตศาสตร์ ค่อนข้าง ยากมากๆ

→ มีข้อดีเรื่องการหา ภาพตรงกลาง โดย ไม่ใช้การ เดาสุ่ม แต่เราสามารถ reconstruction สู่ใน manifold ของภาพได้ !! wow

→ การ train มาแบ่งเป็นส่วนแยกย่อย โดยใช้ model เดียวกัน และใช้เรื่อง time encoding (Attention as all you need)

Bad

→ สิ่งที่ใช้แบ่ง 1000 time ⇒ train โคตรนาน

→ destruction 100 up → โคตรนาน

→ แถมคือ ไปสู่ original of life (OMG!!)

Scalable Gradients for Stochastic Differential Eq.

สามารถดูเพิ่มใน youtube วิดีโอของ MIT

→ Ito calculus

จะใช้ตัว random เป็น B_r

→ เป็น calculus ในรูปแบบที่อธิบาย stochastic process

$$X = \mu \overset{\text{เวลา}}{t} + \sigma B_r$$

σB_r → random var ที่ทำให้ stochastic

μt ⇓ หมายถึง mean path ที่ควร

σB_r → สื่อถึง var ที่เกิดขึ้นในแต่ละเวลา

$$\boxed{X = \mu(x,t)\,t + \sigma(x,t)\,B_r}$$

⇓ รูปแบบของการ ตั้ง stochastic eq.

โดยการ differential ใน Ito calculus จะมี เพิ่ม ตัวของ $(dB_r)^2 = dt$ ⇒ อธิบายต่อ

⇑ assume ว่ามี

โดย สมการ ของ differential equation ทั่วไป

$$\frac{dX}{dt} = f(X) \Rightarrow \text{เขียน form เป็น } dX = f(X)\,dt$$

∴ โดยถ้าเขียนเป็นรูป Ito calculus จะเป็น form

$$dX = \underbrace{\mu(X,t)}_{\text{mean}}dt + \underbrace{\sigma(X,t)}_{\text{Var}}\underbrace{dB_t}_{\text{random time-inter}}$$

คำตอบของ สมการจริงเป็น

$$X = x(0) + \int \mu(x,t)\,dt + \sum_{i=1}^{m}\int_0^t$$

เพราะ time-invar ทุกตัวเลย start ที่ 0 เหมือนกันได้

⇒ โดยสิ่งที่ paper นี้ต้องการ คือการหา $\frac{\partial J}{\partial \theta}$ ต่อจากไป

เพราะต้องได้ $\frac{\partial J}{\partial \theta}$ → ใช้ Back prop ต่อได้ใน DL

⇒ โดยเราจะหาโดยใช้ adjoint method

(วิธีเดียวกับ Neural ODE)

แต่ถ้าคิดปกติจะต้องใช้ Memory มาก

→ Stratonovich Integral

→ ต้องไปดูว่าต้องการเขียนต่อ

→ หลักการ random

NG Boost

→ ใช้ Nerual gradient ในการ training model

→ model ต้องการ train ให้ output มีความเป็น probability ⇒ เหมาะกับพวก Bio or Med ที่ไม่มีความแน่นอน จริงๆ

→ Nerual grad
คือการ train model ใน main fold ของ space ของเมตริกไม่ใช่ของ model weight
(คล้ายกับการ add aug-task)

→ test กับ score (เทียบเท่า loss f^n) ในการวัด การ train 2 แบบ คือ MLE ($\mathcal{L} = -\log P$)
กับ CPRS ($C = \int_{-\infty}^{\infty} (P(x) - P_a(x))^2 \, dx$)

→ $\nabla_N g = I_\theta \nabla g$

original

matrix space

→ ใช้ เสริมการ boosting

boosting → มีหลาย weight model ใช้ปรับแบบเดียวกัน

→ หาตรงตา grad แล้ว train หลายตัว พร้อมกัน

(XG boost) ใช้ tree (Random forrest)

Shapley

Asymmetric Shapley value

~~Asymmetric Shapley~~

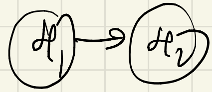
→ ไปดู Blog Shapley value

→ เป็นการคำนวณหาความ impact ของ input ในการ training model

∴ แต่การให้ weight มัน symmetric

แต่ในบางอย่าง input มันมี causal graph

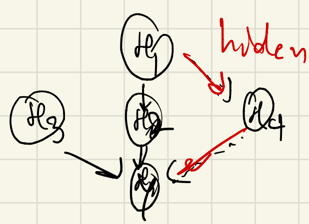
ตามเวลา เช่น x_1 ⇒ แต่งงานไหม



x_2 → สถานะสมรส

ใน ASV → จะให้น้ำหนักไปที่ต้นเหตุมากกว่า

แล้วก็เพิ่มการ check บางอย่างที่มี hidden causal graph



→ และ: สำคัญ input ที่ พอดี ไส่

What if NN had SVD (ข้อเสีย paper)

ไม่ test ว่าใช้ algo นี้แล้วผลลัพธ์ ต่างจากดีกว่า วิธีปกติขนาดไหน

⇒ ทำไมถึงต้องเป็น SVD

SVD ⇒ สามารถทำให้ model สามารถคำนวณได้เร็วขึ้น

∴ exampl solve $Ax=b$ ⇒ แก้หา x

∴ ~~แก้~~โดย row-operation ⇒ $O(n^3)$

แต่ถ้า $A = UDV^T$ (SVD-form)

$$\begin{bmatrix} UU^T = I \\ V^TV = I \end{bmatrix} \quad \therefore UDV^T x = b$$

$$x = \underbrace{V D^{-1} U^T b}_{O(n^2)\ \ O(n)\ \ O(n^2)}$$

ลดเหลือแค่ $O(n^2) < O(n^3)$

การจัด form SVD สามารถทำให้ระบบทำงานได้เร็วขึ้น

จัดรูปได้ทุกครั้งเหรอ, จะสามารถทำงานได้เร็ว

∴ การที่ model SVD ⇒ $A = UDV^T x$

การทำ grad descent $U_{new} = U - \Delta U$, $D_{new} = D - \Delta D$, $V_{new} = V - \Delta V$

การปรับแบบ grad ปกติไม่สามารถ การันตี

ให้ U, V orthogonal ได้ (D เป็น diagonal grad ไม่มีปัญหา)

∴ เขาต้องใช้วิธีใหม่ ใช้หลัก House holder matrix

เขาให้ $U = \prod_{i=1}^{d} H_i$ โดย $H_i = I - 2\frac{vv^T}{\|v\|^2}$

∴ ถ้าจะแบบนี้ การันตี ทุก grad descent

โดยจำนวน parameter เท่าเดิม

โดยใช้ ทฤษฎี บทนี้การแปลง ∴ สามารถ ทำงานได้ดีขึ้น

(paper ใช้ Auto grad (ของจริงใช้ pytorch, tensorflow ได้)

ดูต่อใน paper ดีกว่า

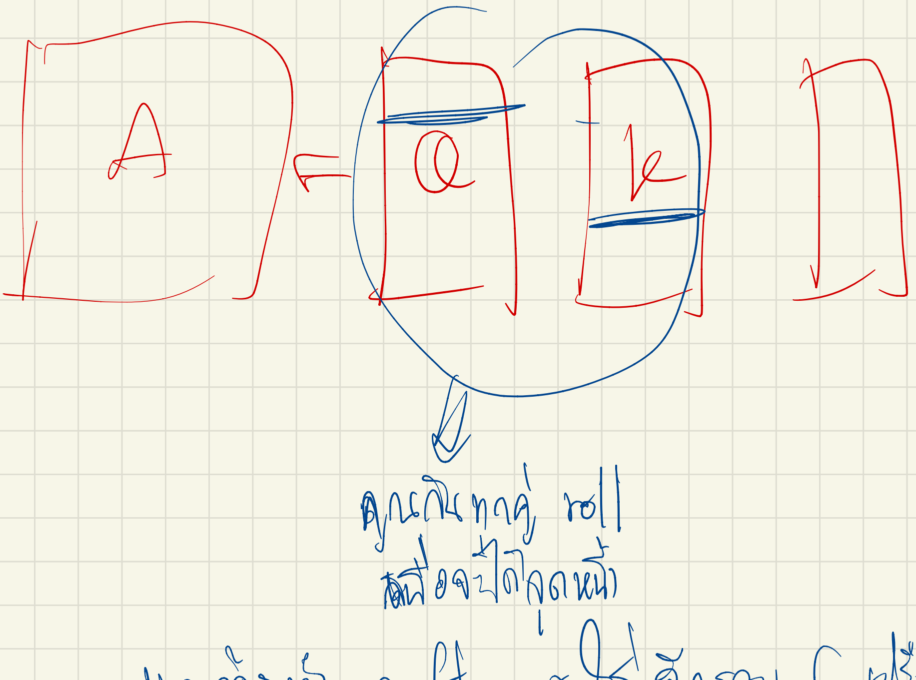
rethink performer

⇒ ปกติ tranformer model มัน เป็นที่นิยมมาก

⇒ แต่ tranform มัน ค่อนข้างหนัก เปลืองมาก
ใส่ส่วนของ

[Multi-Head self-Attention]

ใช้ หาค่าต้องตา



คูณกันทุกคู่ roll
ต้องจบไปได้ทุกหน้า

แต่ต้องทำ softmax ใส่ อีกรอบ (เปลืองอีก)

∴ การคิดนี้ จริงๆมีเหตุผล

∴ มี paper หลายตัว พยายามแก้ตรงนี้

⇒ แต่ ∴ ต้องขอ [paper นี้บอกว่าอันนี้ไม่มี bias
แต่ของเล่าไม่ได้ ?]

∴ paper นี้ใช้ หลักของ kernel แบบ random

(สุ่มตามสมการใน paper)

~~แต่~~ เพราะสิ่งที่ Q, K นั้นในการสร้าง attention-map

~~นั้น~~ มันเหมือนกันกับการเลือกค่าผ่านตัวของ

symilality พวกนั้น

∴ การใช้ ~~#~~ map ได้ kernel นั้น สามารถใช้ kernel

คิดออกมาแต่ได้เลย

(ผลไม่ค่อยจะดีเท่าไร) ค่าดีหรือไม่ดี เทียบกับ transformer

∴ แต่ประหยัด memory ได้เยอะมาก

(crash ในจุดที่ไกลกว่า transfomer มาก)