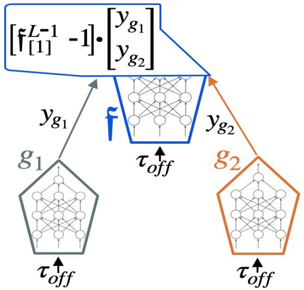
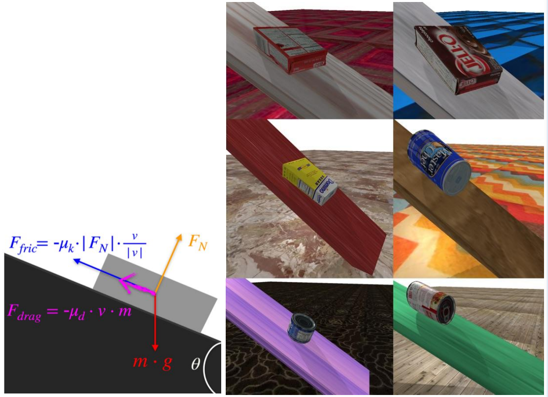
Analytic Manifold Learning
for Modular Latent Space Transfer

Rika Antonova

KTH, Stockholm, Sweden

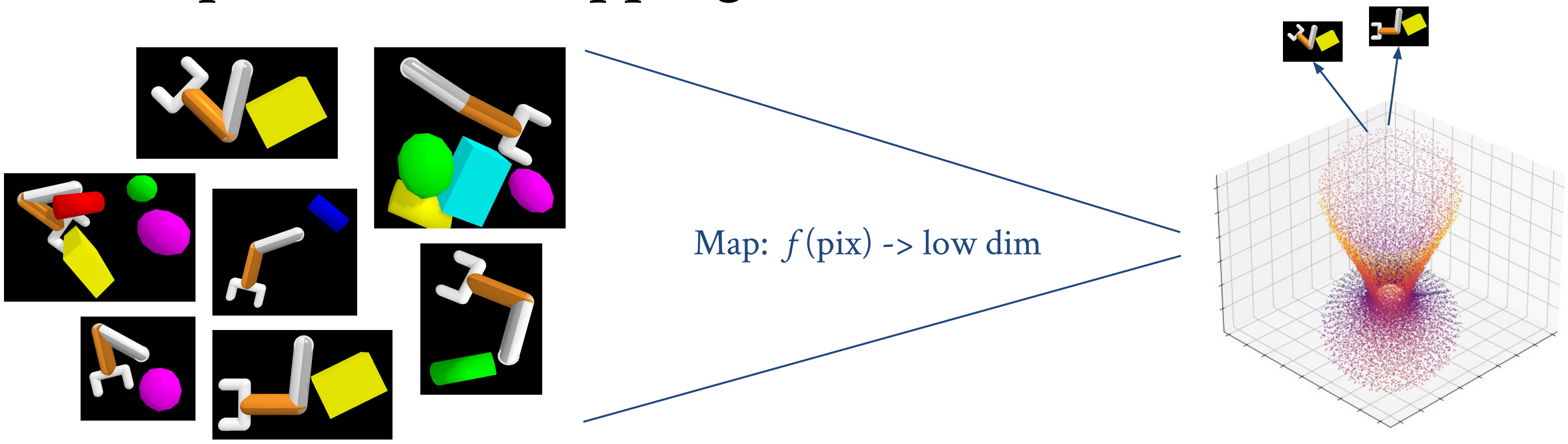




Analytic Manifold Learning: Unifying and Evaluating Representations for Continuous Control.

R Antonova, M Maydanskiy, D Kragic, S Devlin, K Hofmann. arxiv.org/abs/2006.08718

Latent Spaces: The 'Mapping' View

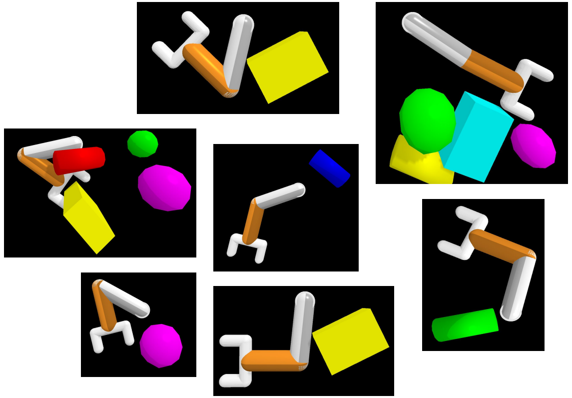


A common view of discovering a latent submanifold is to learn a mapping from high- to low-dim space with some desirable properties

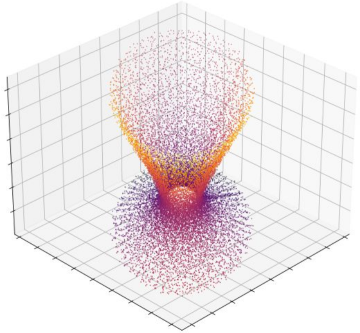
=> the submanifold is represented as an image of some map

Latent Spaces: The ‘Mapping’ View

source domain



Map: f (pix) -> low dim

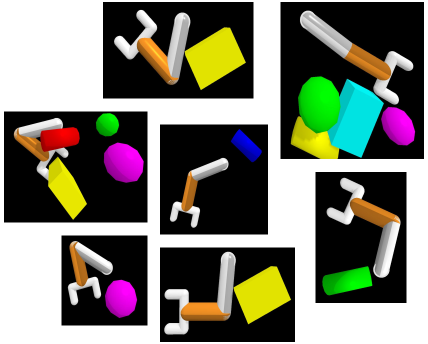


target domain

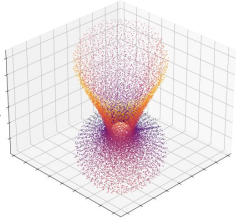
Map: f (‘real’ pix) -> low dim

Latent Spaces: The 'Mapping' View

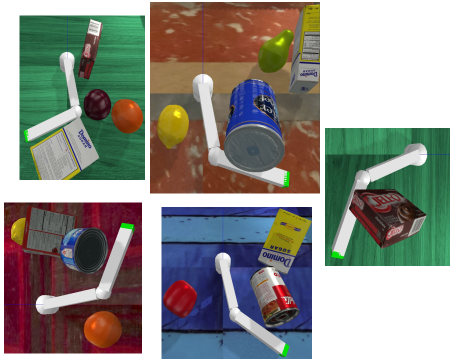
source domain



Map: f(pix) -> low dim



target domain



Map: f('real' pix) -> low dim

If target domain data is expensive to get, then need to reuse info from the source domain

But how?

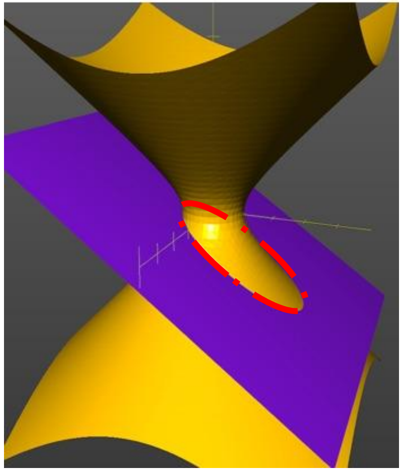
Fine-tune f?

- Catastrophic forgetting
- Bad local optima due to lack of random init
- Small learning rate to keep latent space structure

Latent Spaces: The 'Relations' View

Alternatively, a submanifold can be specified by describing all equations (relations) that have to hold of the points in the submanifold

=> the submanifold is represented as a null space of a set of functions



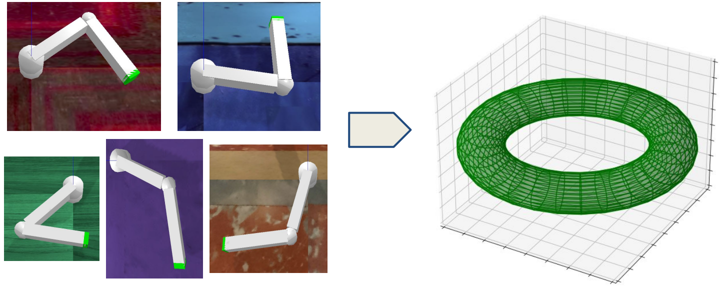
Example: representing an ellipse as an intersection of a hyperboloid and a plane:

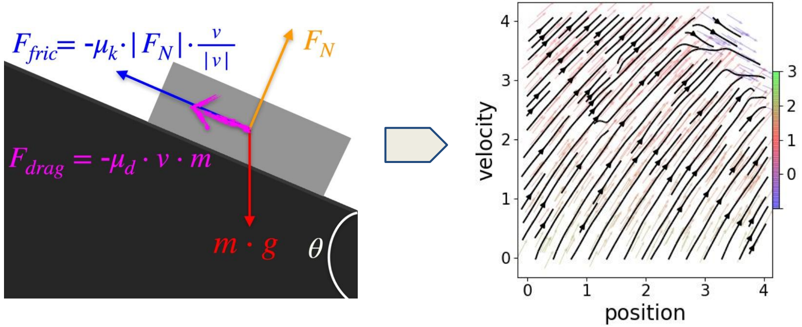
$$\begin{cases} x^2 + y^2 - 2z^2 - 1 = 0 & \text{relation } g_1 \\ y - 2z - 1 = 0 & \text{relation } g_2 \end{cases}$$

Latent Spaces: The 'Relations' View

The 'relations' view allows to encode the latent manifold directly

This is particularly useful for sim-to-real, since we can encode concisely the properties of the dynamical system in low-dim relations





We can then transfer this knowledge directly instead of needing photo-realistic rendering or struggling with fine-tuning

Latent Space Transfer

Previous work proposed using domain knowledge to structure the latent space during training

For example, imposing continuity between consecutive states:

$$L_{cont}(\mathcal{D}_x, \phi) = \mathbb{E}\left[||s_{t+1} - s_t||^2\right]$$

s_t is a low-dimensional or latent state, x_t is the corresponding high-dimensional state (e.g. RGB image), $D_x = \{x_t, x_{t+1}, \ldots\}$ & encoder $\phi(x) = s$

"State representation learning for control: An overview" T. Lesort, N. Diaz-Rodríguez, J.F. Goudou, D. Filliat. Neural Networks. 2018.

Such heuristics draw from intuition and prior knowledge, and it is tedious to manually incorporate a comprehensive set of these into the overall optimization

Latent Space Transfer with Analytic Manifold Learning

We take a broader perspective: we learn a set of relations that are non-linearly independent, and we define independence rigorously

Let $\mathbb{R}^N$ be the ambient space of possible latent state sequences τ

$$\tau = [s_1, a_1, s_2, a_2, ...]$$

Let $\mathcal{M}$ be the submanifold of actual state sequences that our dynamical system could generate (under any control policy)

Our goal is to capture the data submanifold by learning relations that have to hold for points in the submanifold

Analytic Manifold Learning : Mathematical Formulation

In linear algebra, a dependency is a linear combination of vectors with constant coefficients

In our nonlinear setting the analogous notion is that of *syzygy*

A collection of functions $\mathfrak{f}^{\ddagger} = \{f_1, \ldots, f_k\}$ is called a *syzygy* if $\sum_{j=0}^{k} f_j g_j$ is zero

If there is no syzygy $\mathfrak{f}^{\ddagger}$ s.t. $\sum_{j=0}^{k} f_j g_j = 0$, then $g_1, \ldots, g_k$ are independent

However, this notion of independence deems any g_1, g_2 dependent:
$g_1 \cdot g_2 - g_2 \cdot g_1 = 0$ holds for any g_1, g_2

Analytic Manifold Learning : Mathematical Formulation

Hence, we define restricted *syzygies*

Definition [Restricted Syzygy] : Restricted syzygy for relations $g_1, \ldots, g_k$ is a syzygy with the last entry f_k equal to -1, i.e. $\mathfrak{f} = \{f_1, \ldots, f_{k-1}, f_k = -1\}$ with $\sum_{j=1}^{k} f_j g_j = 0$.

Definition [Restricted Independence] : g_k is independent from $g_1, \ldots, g_{k-1}$ in a restricted sense if $\sum_{j=1}^{k} f_j g_j = 0$ implies $f_k \neq -1$, i.e. if there exists no restricted syzygy for $g_1, \ldots, g_k$.

Using these we can learn independent relations iteratively

Analytic Manifold Learning : Mathematical Formulation

Theorem [Restricted Independence] : When using real-analytic functions to approximate gs, the process of starting with a relation g_1 and iteratively adding new independent g_ks will terminate.

Definition [Strong Independence] : g_k is strongly independent from $g_1, \ldots, g_{k-1}$ if the equality $\sum_{j=1}^{k} f_j g_j = 0$ implies that f_k is expressible as $f_k = h_1 \cdot g_1 + \ldots + h_{k-1} \cdot g_{k-1}$.

Theorem [Strong Independence] : Suppose $g_1, \ldots, g_k$ is a sequence of analytic functions on B, each strongly independent of the previous ones. Denote by $\mathcal{M}_{\mathring{B}} = \{x \in \mathring{B} | g_j(x) = 0 \text{ for all } j\}$ the part of the learned data manifold lying in the interior of B. Then dimension of $\mathcal{M}_{\mathring{B}}$ is at most $N - k$.

Analytic Manifold Learning : Mathematical Formulation

We also give an alternative definition of independence via transversality

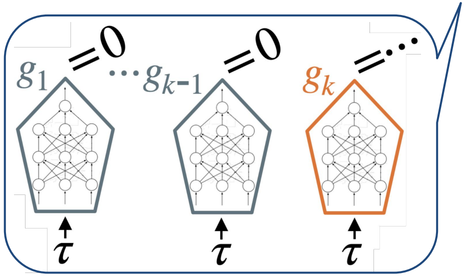
It ensures relations differ to first order and also yields guarantees on the dimensionality of the learned submanifold

Definition [Transversality] : If for all points $\tau^{(i)} \in \mathcal{M}$ the gradients of $g_1, .., g_k$ at τ, i.e. $v = \nabla_\tau g\big|_{\tau^{(i)}}$, are linearly independent, we say that g_k is transverse to the previous relations: $g_k \pitchfork g_1, \ldots, g_{k\text{-}1}$.

Lemma : For once differentiable $(g_1, .., g_k)$ s.t. H_{g_j}s are transverse along their common intersection H, this intersection H is a submanifold of $\mathbb{R}^N$ of dimension $N-k$.

Latent Space Transfer with Analytic Manifold Learning

Formulating the problem as learning analytic relations $g_1, .., g_k$ that cut out the latent data manifold allows us to use neural networks as function approximators



Analytic Manifold Learning : Training with Transversality

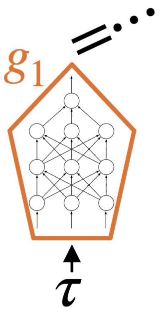
Algorithm 1 : Analytic Manifold Learning (AML)

$\{\tau^{(i)}\}_{i=1}^{d} \leftarrow$ rollouts from RL actors
train g_1 with loss $L = g_d(\tau) - \log \|v\|$ (Eq.1)
for $k = 2, 3, ...,$ **do**
  if *aiming_for_transversality* **then**
    train g_k with loss L_{tr} from Eq.2

Analytic Manifold Learning : Training with Transversality

Algorithm 1 : Analytic Manifold Learning (AML)

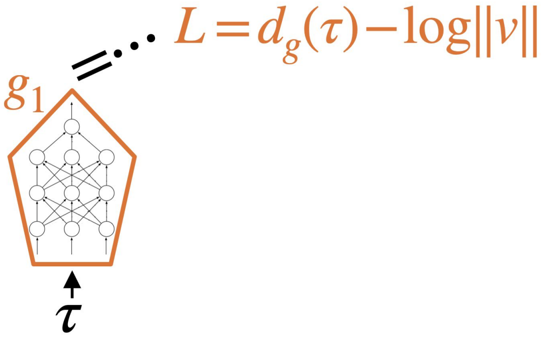
$\{\tau^{(i)}\}_{i=1}^{d} \leftarrow$ rollouts from RL actors
$\boxed{\text{train } g_1}$ with loss $L = g_d(\tau) - \log\|v\|$ (Eq.1)
for $k = 2, 3, \ldots,$ **do**
  if *aiming_for_transversality* **then**
    train g_k with loss L_{tr} from Eq.2



Analytic Manifold Learning : Training with Transversality

Algorithm 1 : Analytic Manifold Learning (AML)

$\{\tau^{(i)}\}_{i=1}^{d} \leftarrow$ rollouts from RL actors
train g_1 with loss $L = g_d(\tau) - \log \|v\|$ (Eq.1)
for $k = 2, 3, \ldots,$ **do**
  if *aiming_for_transversality* **then**
    train g_k with loss L_{tr} from Eq.2

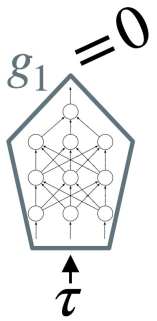


$$L(g) = d_g(\tau) - \log \|v\| \quad ; \quad d_g(\tau) = |g(\tau)| / \|v\| \qquad (1)$$

Analytic Manifold Learning : Training with Transversality

Algorithm 1 : Analytic Manifold Learning (AML)

```
{τ^(i)}_{i=1}^d ← rollouts from RL actors
train g_1 with loss L = g_d(τ) − log ||v||   (Eq.1)
for k = 2, 3, ..., do
    if aiming_for_transversality then
        train g_k with loss L_tr from Eq.2
```

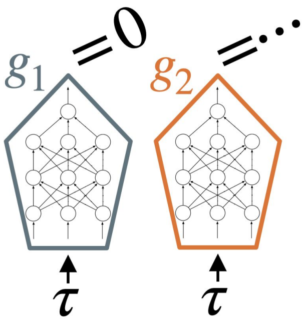


$$L(g) = d_g(\tau) - \log \|v\| \;\; ; \;\; v = \nabla_\tau g|_{\tau^{(i)}} \;\; ; \;\; d_g(\tau) = \frac{|g(\tau)|}{\|v\|} \quad (1)$$

Analytic Manifold Learning : Training with Transversality

Algorithm 1 : Analytic Manifold Learning (AML)

$\{\tau^{(i)}\}_{i=1}^{d} \leftarrow$ rollouts from RL actors
train g_1 with loss $L = g_d(\tau) - \log \|v\|$ (Eq.1)
for $k = 2, 3, \ldots,$ **do**
    if *aiming_for_transversality* **then**
        train g_k with loss L_{tr} from Eq.2

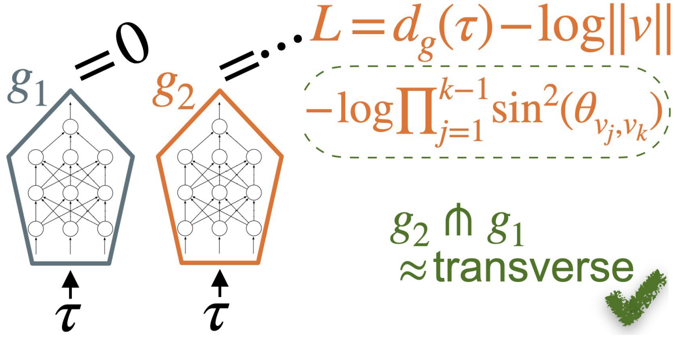


$$L(g) = d_g(\tau) - \log \|v\| \;\; ; \;\; v = \nabla_\tau g\big|_{\tau^{(i)}} \; ; \;\; d_g(\tau) = \frac{|g(\tau)|}{\|v\|} \quad (1)$$

Analytic Manifold Learning : Training with Transversality

Algorithm 1 : Analytic Manifold Learning (AML)

$\{\tau^{(i)}\}_{i=1}^{d} \leftarrow$ rollouts from RL actors
train g_1 with loss $L = g_d(\tau) - \log\|v\|$ (Eq.1)
for $k = 2, 3, \ldots,$ **do**
  if *aiming_for_transversality* **then**
    train g_k with loss L_{tr} from Eq.2



$$L(g) = d_g(\tau) - \log\|v\| \ ; \ v = \nabla_\tau g|_{\tau^{(i)}} \ ; \ d_g(\tau) = \frac{|g(\tau)|}{\|v\|} \quad (1)$$

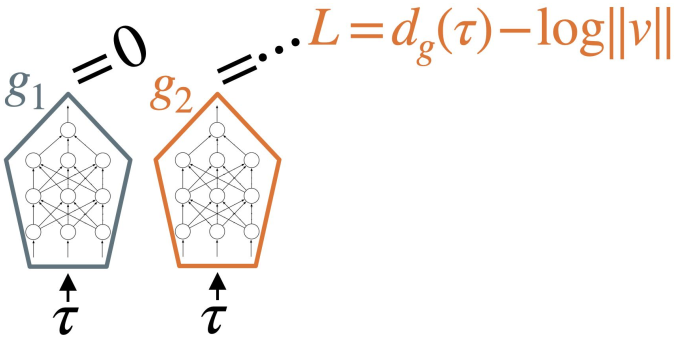
$$L_{tr}(g_k) = L(g_k) - \log\prod_{j=1}^{k-1}\sin^2\underbrace{(\theta_{v_j,v_k})}_{\text{angle}(v_j,v_k)} \quad (2)$$

Analytic Manifold Learning : Training with Syzygies

Algorithm 1 : Analytic Manifold Learning (AML)

$\{\tau^{(i)}\}_{i=1}^{d} \leftarrow$ rollouts from RL actors
train g_1 with loss $L = g_d(\tau) - \log \|v\|$ (Eq.1)
for $k = 2, 3, \ldots,$ **do**
...
  else // using syzygies
    train g_k with loss L from Eq.1
    for $j = 1, 2, \ldots,$ **do**
      generate $\tau_{off}, \tau_{off}^{test}$
      train $\mathfrak{f}_j$ with $L_{\mathfrak{f}} = |\mathfrak{f}_j(\tau_{off})|$
      if $\mathfrak{f}_j \neq 0$ *on* τ_{off}^{test} **then** break $//g_k \approx$indep.
      while $\mathfrak{f}_j(\tau_{off}^{test}) \approx 0$ **do**
        freeze $\mathfrak{f}_j$; train g_k with L_{syz} (Eq.3)

$$L(g) = d_g(\tau) - \log \|v\| \;;\; v = \nabla_\tau g|_{\tau^{(i)}} \;;\; d_g(\tau) = \frac{|g(\tau)|}{\|v\|} \quad (1)$$

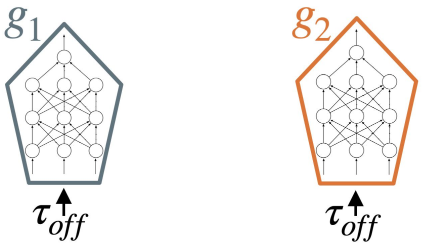


Analytic Manifold Learning : Training with Syzygies

Algorithm 1 : Analytic Manifold Learning (AML)

$\{\tau^{(i)}\}_{i=1}^{d} \leftarrow$ rollouts from RL actors
train g_1 with loss $L = g_d(\tau) - \log \|v\|$ (Eq.1)
for $k = 2, 3, \ldots,$ **do**
...
  else // using syzygies
    train g_k with loss L from Eq.1
    for $j = 1, 2, \ldots,$ **do**
      generate $\tau_{off}, \tau_{off}^{test}$
      train $\mathfrak{f}_j$ with $L_{\mathfrak{f}} = |\mathfrak{f}_j(\tau_{off})|$
      if $\mathfrak{f}_j \neq 0$ *on* τ_{off}^{test} **then** break //$g_k \approx$indep.
      while $\mathfrak{f}_j(\tau_{off}^{test}) \approx 0$ **do**
        freeze $\mathfrak{f}_j$; train g_k with L_{syz} (Eq.3)

Off-manifold data: $\tau_{off} = \{s_{off_t}, s_{off_{t+1}}, \ldots, s_{off_T}\}$

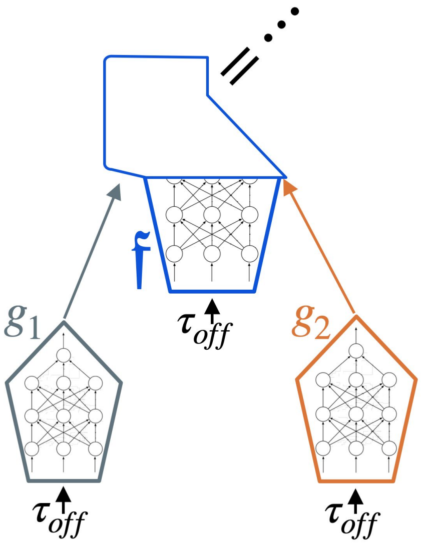


Analytic Manifold Learning : Training with Syzygies

Algorithm 1 : Analytic Manifold Learning (AML)

$\{\tau^{(i)}\}_{i=1}^{d} \leftarrow$ rollouts from RL actors
train g_1 with loss $L = g_d(\tau) - \log \|v\|$ (Eq.1)
for $k = 2, 3, \ldots,$ **do**
…
  else // using syzygies
    train g_k with loss L from Eq.1
    for $j = 1, 2, \ldots,$ **do**
      generate τ_{off}, τ_{off}^{test}
      train $\mathfrak{f}_j$ with $L_{\mathfrak{f}} = |\mathfrak{f}_j(\tau_{off})|$
      if $\mathfrak{f}_j \neq 0$ *on* τ_{off}^{test} **then** break //$g_k \approx$indep.
      while $\mathfrak{f}_j(\tau_{off}^{test}) \approx 0$ **do**
        freeze $\mathfrak{f}_j$; train g_k with L_{syz} (Eq.3)

Off-manifold data: $\tau_{off} = \{s_{off_t}, s_{off_{t+1}}, \ldots, s_{off_T}\}$



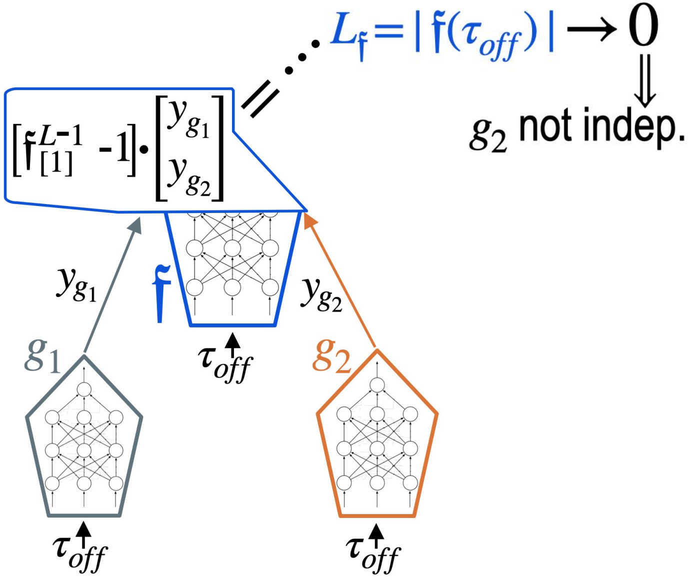
Analytic Manifold Learning : Training with Syzygies

Algorithm 1 : Analytic Manifold Learning (AML)

$\{\tau^{(i)}\}_{i=1}^{d} \leftarrow$ rollouts from RL actors
train g_1 with loss $L = g_d(\tau) - \log \|v\|$ (Eq.1)
for $k = 2, 3, \ldots,$ **do**
…
else // using syzygies
train g_k with loss L from Eq.1
for $j = 1, 2, \ldots,$ **do**
generate $\tau_{off}, \tau_{off}^{test}$
train $\mathfrak{f}_j$ with $L_{\mathfrak{f}} = |\mathfrak{f}_j(\tau_{off})|$
if $\mathfrak{f}_j \neq 0$ *on* τ_{off}^{test} **then** break //$g_k \approx$indep.
while $\mathfrak{f}_j(\tau_{off}^{test}) \approx 0$ **do**
freeze $\mathfrak{f}_j$; train g_k with L_{syz} (Eq.3)

Off-manifold data: $\tau_{off} = \{s_{off_t}, s_{off_{t+1}}, \ldots, s_{off_T}\}$

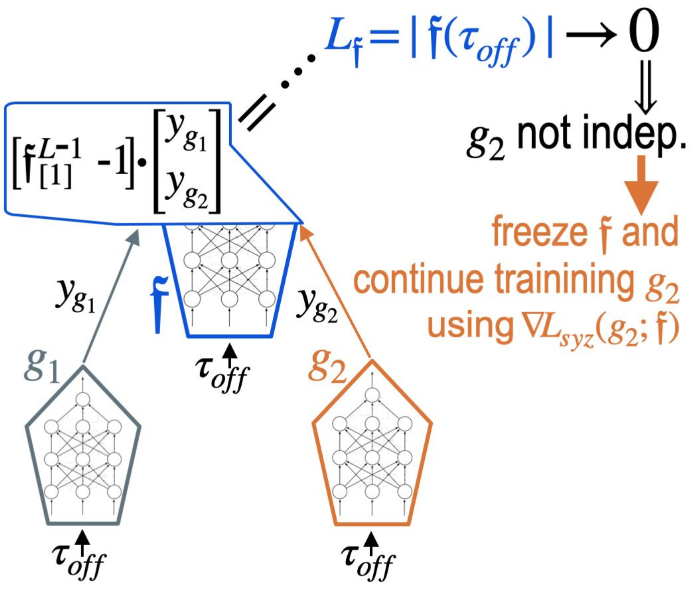
$$\nabla L_{syz}(g_k; \mathfrak{f}) = \nabla L(g_k) - \nabla_{g_k}\Big[\big|\mathfrak{f}(\tau_{off}, g_1, \ldots, g_k)\big|\Big] \quad (3)$$



Analytic Manifold Learning : Training with Syzygies

Algorithm 1 : Analytic Manifold Learning (AML)

```
{τ^(i)}_{i=1}^d ← rollouts from RL actors
train g_1 with loss L = g_d(τ) − log ||v||   (Eq.1)
for k = 2, 3, ..., do
    ...
    else // using syzygies
        train g_k with loss L from Eq.1
        for j = 1, 2, ..., do
            generate τ_off , τ_off^test
            train 𝔣_j with L_𝔣 = |𝔣_j(τ_off)|
            if 𝔣_j ≠ 0 on τ_off^test then break //g_k≈indep.
            while 𝔣_j(τ_off^test) ≈ 0 do
                freeze 𝔣_j; train g_k with L_syz (Eq.3)
```



Off-manifold data: $\tau_{off} = \{s_{off_t}, s_{off_{t+1}}, \ldots, s_{off_T}\}$

$$\nabla L_{syz}(g_k; \mathfrak{f}) = \nabla L(g_k) - \nabla_{g_k}\left[\left|\mathfrak{f}(\tau_{off}, g_1, \ldots, g_k)\right|\right] \quad (3)$$

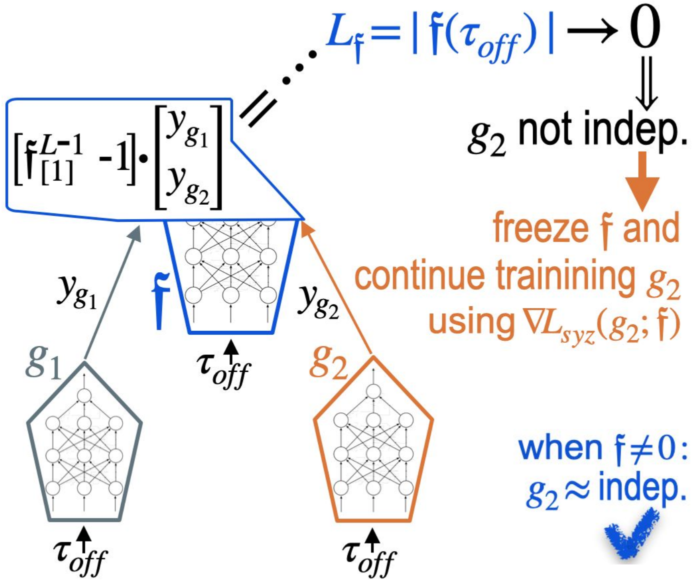
Analytic Manifold Learning : Training with Syzygies

Algorithm 1 : Analytic Manifold Learning (AML)

$\{\tau^{(i)}\}_{i=1}^{d} \leftarrow$ rollouts from RL actors
train g_1 with loss $L = g_d(\tau) - \log\|v\|$ (Eq.1)
for $k = 2, 3, \ldots,$ **do**
…
  else // using syzygies
    train g_k with loss L from Eq.1
    for $j = 1, 2, \ldots,$ **do**
      generate $\tau_{off}, \tau_{off}^{test}$
      train $\mathfrak{f}_j$ with $L_{\mathfrak{f}} = |\mathfrak{f}_j(\tau_{off})|$
      if $\mathfrak{f}_j \neq 0$ *on* τ_{off}^{test} **then** break //g_k≈indep.
      while $\mathfrak{f}_j(\tau_{off}^{test}) \approx 0$ **do**
        freeze $\mathfrak{f}_j$; train g_k with L_{syz} (Eq.3)

Off-manifold data: $\tau_{off} = \{s_{off_t}, s_{off_{t+1}}, \ldots, s_{off_T}\}$

$$\nabla L_{syz}(g_k; \mathfrak{f}) = \nabla L(g_k) - \nabla_{g_k}\left[\left|\mathfrak{f}(\tau_{off}, g_1, \ldots, g_k)\right|\right] \quad (3)$$



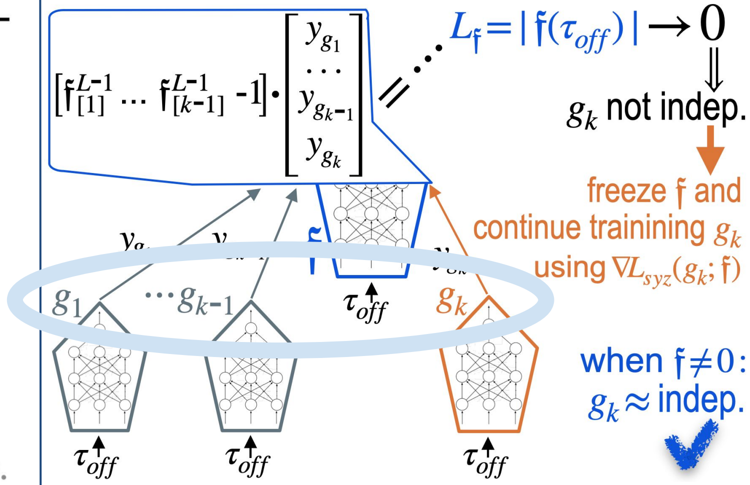
Analytic Manifold Learning : Training with Syzygies

Algorithm 1 : Analytic Manifold Learning (AML)

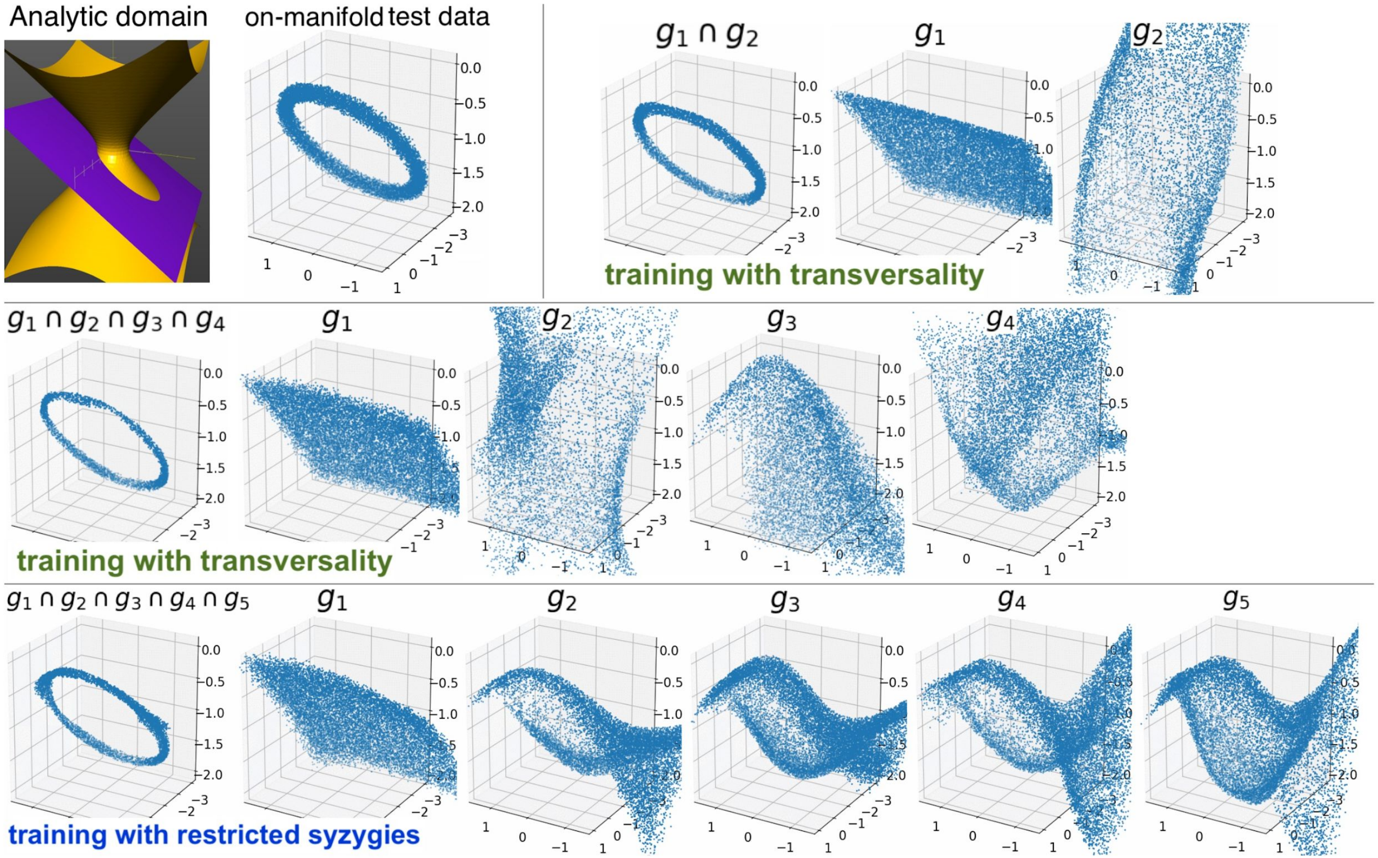
$\{\tau^{(i)}\}_{i=1}^{d} \leftarrow$ rollouts from RL actors
train g_1 with loss $L = g_d(\tau) - \log \|v\|$ (Eq.1)
for $k = 2, 3, \ldots,$ **do**
…
else // using syzygies
train g_k with loss L from Eq.1
for $j = 1, 2, \ldots,$ **do**
generate $\tau_{off}, \tau_{off}^{test}$
train $\mathfrak{f}_j$ with $L_{\mathfrak{f}} = |\mathfrak{f}_j(\tau_{off})|$
if $\mathfrak{f}_j \neq 0$ *on* τ_{off}^{test} **then** break //$g_k \approx$indep.
while $\mathfrak{f}_j(\tau_{off}^{test}) \approx 0$ **do**
freeze $\mathfrak{f}_j$; train g_k with L_{syz} (Eq.3)

Off-manifold data: $\tau_{off} = \{s_{off_t}, s_{off_{t+1}}, \ldots, s_{off_T}\}$

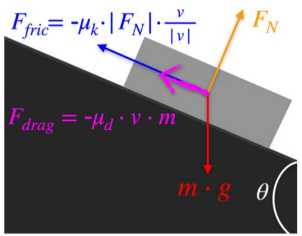
$$\nabla L_{syz}(g_k; \mathfrak{f}) = \nabla L(g_k) - \nabla_{g_k}\Big[\big|\mathfrak{f}(\tau_{off}, g_1, \ldots, g_k)\big|\Big] \quad (3)$$

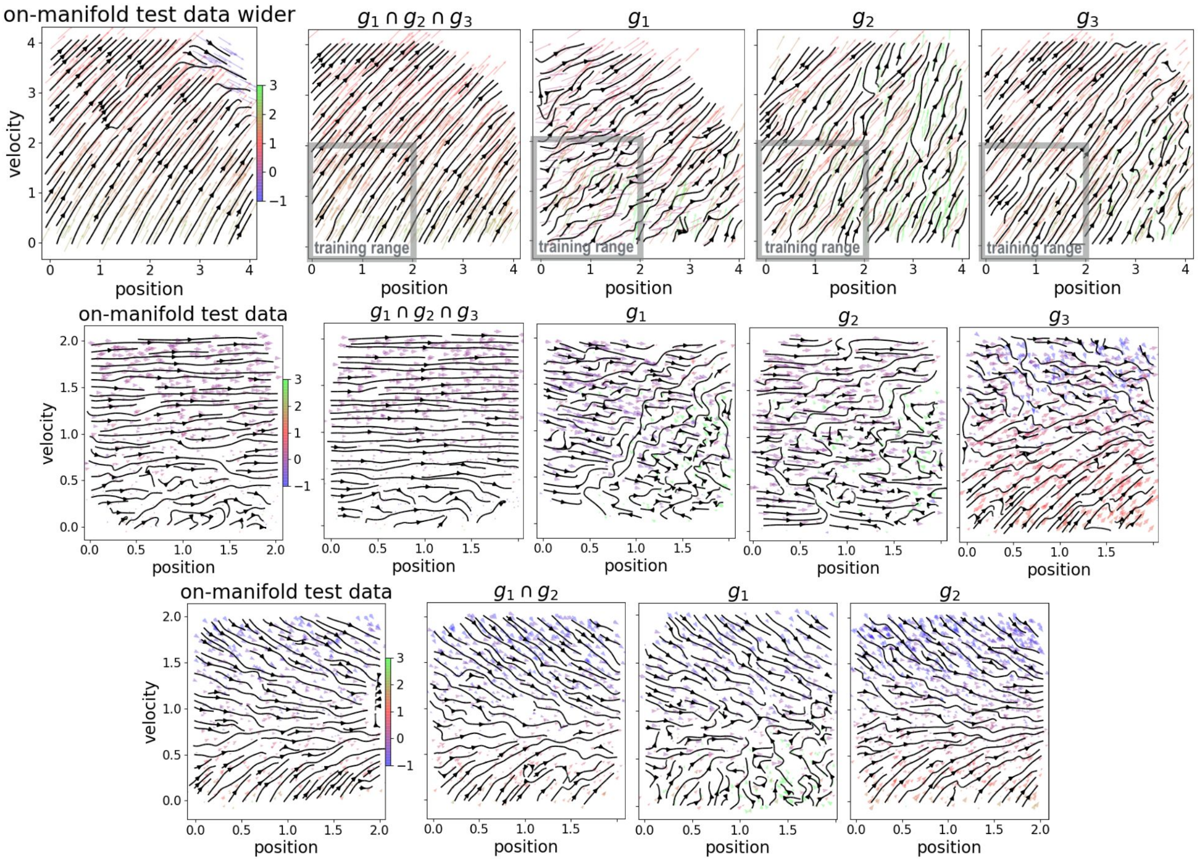


Analytic Manifold Learning : Modular Latent Spaces



Analytic Manifold Learning : Modular Latent Spaces



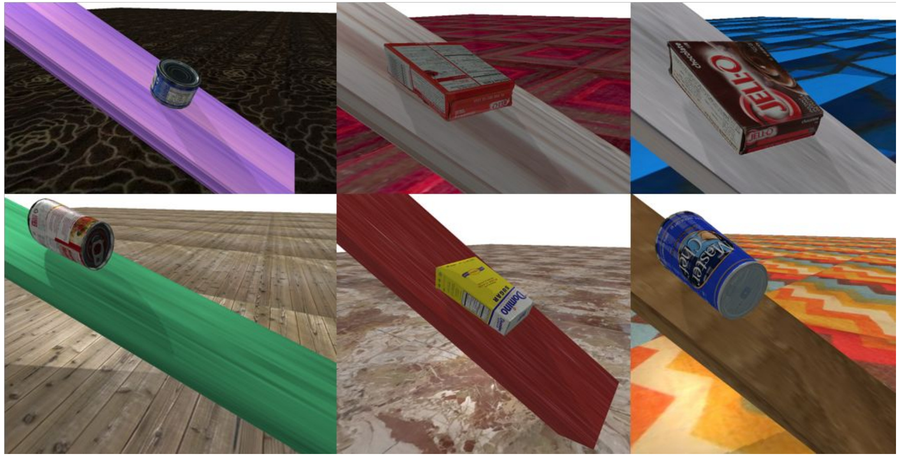


Analytic Manifold Learning : Latent Space Transfer

As baselines, we use two kinds of unsupervised learners: VAE and $PRED$: a sequential VAE that, given a sequence of frames $x_1, \ldots, x_t$, constructs a predictive sequence $x_1, \ldots, x_{t+k}$

We learn AML relations on simulation states of a domain with simple geometric shapes

Then, we train $PRED$ on the target *YCB-on-incline* domain



Analytic Manifold Learning : Latent Space Transfer

AML relations are imposed on the latent state of $PRED$ by augmenting the latent part of the loss as follows:

$$\mathcal{L}_{PRED}^{AML} = \mathbb{E}_{\substack{\tilde{\tau}_{1:T+L}\sim \\ q(\tau_{1:T+L}|x_{1:T})}} \Big[-\Big(\overbrace{\log p(x_{1:T+L}|\tilde{\tau}_{1:T+L}) - KL\big(q||\mathcal{N}(0,1)\big)}^{\text{standard } ELBO \text{ for } PRED \text{ version of } VAE} \Big) + \underbrace{\textstyle\sum_{k=1}^{K} \big|g_k(\tilde{\tau}_{1:T+L}, a_{1:T+L})\big|}_{\text{impose } AML \text{ relations}} \Big]$$

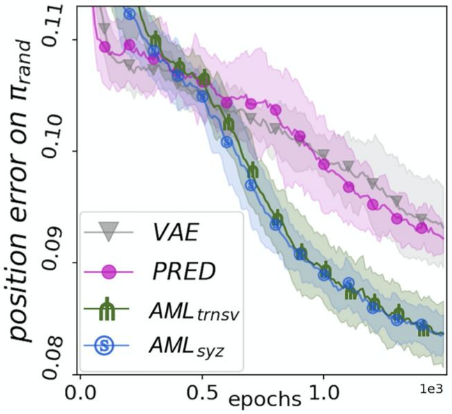
$\tilde{\tau}_{1:T+L}$ denotes a sample from approx. posterior $q(\tau_{1:T+L}|x_{1:T})$

$p(x_{1:T+L}|\tilde{\tau}_{1:T+L})$ denotes the likelihood for $PRED$

magenta color indicates that decoder outputs a predictive sequence $\hat{x}_{1:T+L}$ instead of a reconstruction $\hat{x}_{1:t}$

Analytic Manifold Learning : Latent Space Transfer

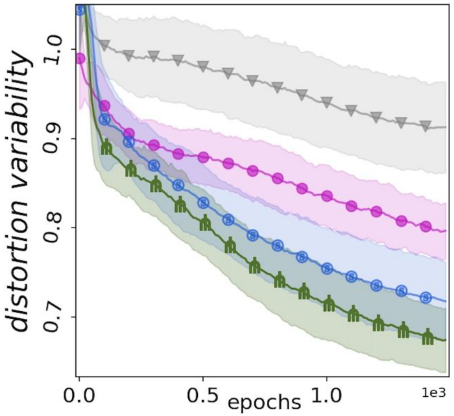
The resulting AML_{trnsv} (AML_{syz} with syzygies) gets a better latent state alignment for object position compared to VAE and $PRED$ without AML relations imposed



AML also lowers distortion of the encoder map, i.e. better preserves the geometry of the low-dimensional manifold

$$\rho_{distort} = \log \frac{d_{L2}(\phi_{enc}(x_1), \phi_{enc}(x_2))}{d_{L2}(\tau_1^{true}, \tau_2^{true})}$$

$\tau_1^{true}, \tau_2^{true}$ are the low-dim representations
x_1, x_2 are the corresponding pixel-based representations



Current & Future Work: Lifelong Learning with AML

KTH VETENSKAP OCH KONST
&
Microsoft Research

Consider an autonomous agent that has acquired useful skills appropriate to a given environment

e.g. a household robot doing a set of household chores

Suppose this agent is transported to an environment that has a different appearance, but similar latent rules/regularities/relations

e.g a robot is moved to a different country to perform similar household tasks

Current & Future Work: Lifelong Learning with AML

We would like the agent to:

- quickly adapt to new visual appearances

- leverage experience embedded in the latent space structure:
  rules/regularities/relations inferred from previous experiences

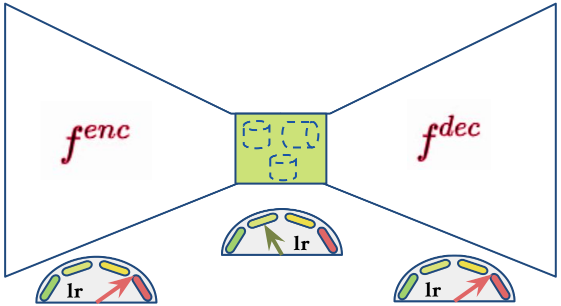
- learn to infer new relations to better reshape the latent space,
  and retain ability to quickly re-adapt to the original environment

Current & Future Work: Lifelong Learning with AML

KTH VETENSKAP OCH KONST & Microsoft Research

Imposing learned relations helps retain latent space structure when learning on the target domain

We can increase enc/dec learning rate =>
faster adaptation to changes in visual/high-dimensional aspects



Current & Future Work: Lifelong Learning with AML

To let the latent space evolve/adapt:

We could introduce weights for each imposed relation and adapt them (e.g. by propagating gradients through the weights)

We could suppress relations whose weights decay to zero and could gradually expand the set by learning new relations

$$w_1 \cdot g_1 + w_2 \cdot g_2 + w_3 \cdot g_3 + \ldots + w_k \cdot g_k$$

$$w_1 \cdot g_1 + \underline{w_2 \cdot g_2} + w_3 \cdot g_3 + \ldots + w_k \cdot g_k$$

$$w_1 \cdot g_1 + \qquad + w_3 \cdot g_3 + \ldots + w_k \cdot g_k$$

$$w_1 \cdot g_1 + \qquad + w_3 \cdot g_3 + \ldots + w_k \cdot g_k + \underline{w_{k+1} \cdot g_{k+1}}$$

Current & Future Work: Lifelong Learning with AML

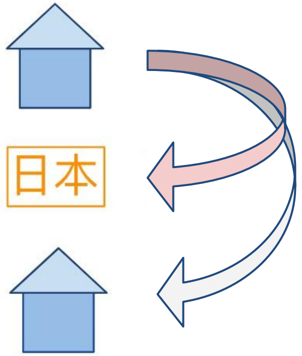
If we anticipate returning to the ‘old world’ but can’t store f^{enc} , f^{dec} :
(large NNs)

we could keep all relations (small NNs), even with weights ≈ 0 in the ‘new world’ and use the old set of relations+weights when we return to the ‘old world’

$$w_1 \cdot g_1 + w_2 \cdot g_2 + w_3 \cdot g_3 + \dots + w_k \cdot g_k$$

$$w_1 \cdot g_1 + 0 \cdot g_2 + w_3 \cdot g_3 + \dots + w_k \cdot g_k + w_{k+1} \cdot g_{k+1}$$

$$w_1 \cdot g_1 + w_2 \cdot g_2 + w_3 \cdot g_3 + \dots + w_k \cdot g_k$$



Future Work: Sim-to-real Hardware

My previous works focused on making sure the algorithms were designed to perform well on hardware, so I would like to ensure this for AML as well