

Learnable Planning and Cost

Expert demonstration
 real world reward.

Imitation Learning

- The explicit policy model, supervised learning (behavior cloning)

$$\hat{a} = f_{\theta}(x) \quad \mathcal{L} = \min_i \|a_i - \hat{a}\|_2^2 \quad \mathcal{L} = -\log \hat{a}_j$$

Imitation Learning

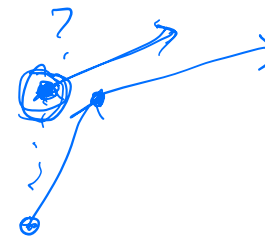
- The explicit policy model, supervised learning (behavior cloning)

$$\hat{a} = f_{\theta}(x) \quad \mathcal{L} = \min_i \|a_i - \hat{a}\|_2^2 \quad \mathcal{L} = -\log \hat{a}_j$$

- Energy-based (cost-based) approach

$$\tau^* = \operatorname{argmin}_{\tau} E(x, \tau)$$
$$p(\tau | x) = \frac{\exp(-E(x, \tau))}{\int_{\tau} \exp(-E(x, \tau))}$$

Imitation Learning



- The explicit policy model, supervised learning (behavior cloning)

$$\hat{a} = f_{\theta}(x) \quad \mathcal{L} = \min_i \|a_i - \hat{a}\|_2^2 \quad \mathcal{L} = -\log \hat{a}_j$$

- Energy-based (cost-based) approach

$$\tau^* = \operatorname{argmin}_{\tau} E(x, \tau)$$
$$p(\tau | x) = \frac{\exp(E(x, \tau))}{\int_{\tau} \exp(E(x, \tau))}$$

- Dataset Aggregation (DAGger)
 - Learned policy may deviate from experts
 - Need to collect more groundtruths

```
Initialize  $\mathcal{D} \leftarrow \emptyset$ .
Initialize  $\hat{\pi}_1$  to any policy in  $\Pi$ .
for  $i = 1$  to  $N$  do
  Let  $\pi_i = \beta_i \pi^* + (1 - \beta_i) \hat{\pi}_i$ .
  Sample  $T$ -step trajectories using  $\pi_i$ .
  Get dataset  $\mathcal{D}_i = \{(s, \pi^*(s))\}$  of visited states by  $\pi_i$ 
  and actions given by expert.
  Aggregate datasets:  $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_i$ .
  Train classifier  $\hat{\pi}_{i+1}$  on  $\mathcal{D}$ .
end for
Return best  $\hat{\pi}_i$  on validation.
```

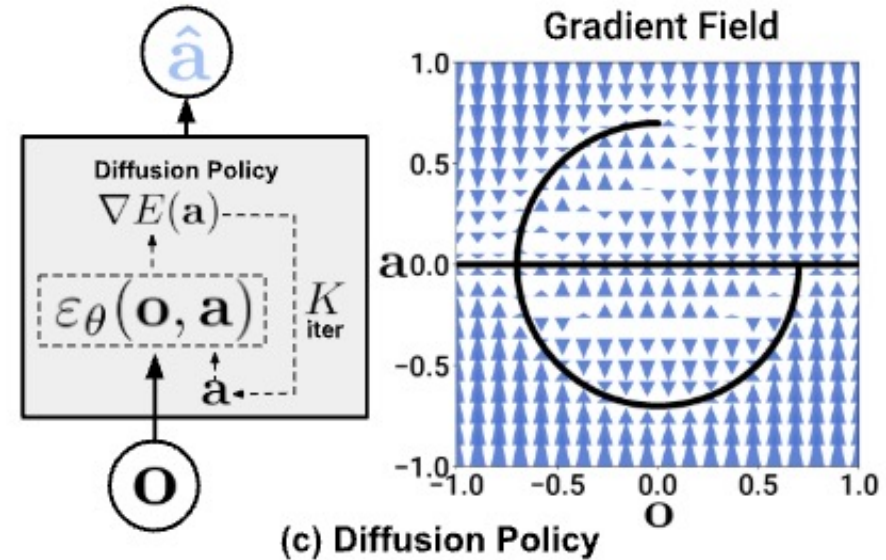
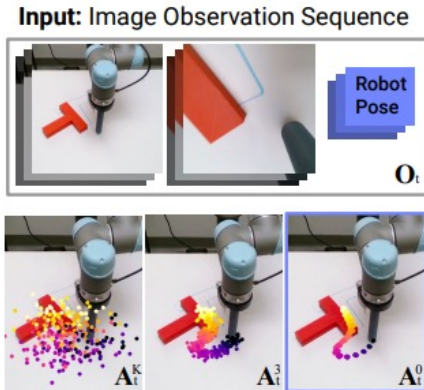
Algorithm 3.1: DAGGER Algorithm.

Direct Policy Learning from Diffusion

- Error prediction network is conditioned on observation features.

$$A_t^{k-1} = \alpha(A_t^k - \gamma \epsilon_\theta(O_t, A_t^k, k) + \mathcal{N}(0, \sigma^2 I)).$$

$$\mathcal{L} = MSE(\epsilon^k, \epsilon_\theta(O_t, A_t + \epsilon^k, k)).$$



Learning Cost

- Typically, we have some rough ideas on what the cost should look like
 - E.g. Avoid obstacle with squared distance barrier.

Learning Cost

- Typically, we have some rough ideas on what the cost should look like
 - E.g. Avoid obstacle with squared distance barrier.
- But there are costs that are implicitly defined.
 - E.g. Human comfort

Learning Cost

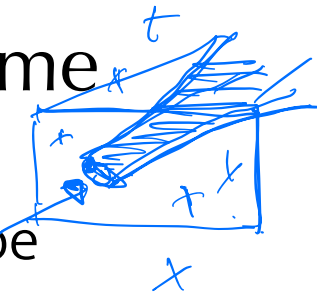
- Typically, we have some rough ideas on what the cost should look like
 - E.g. Avoid obstacle with squared distance barrier.
- But there are costs that are implicitly defined.
 - E.g. Human comfort
- Can be learned from human demonstrations (also called IRL)

↳ optimal traj wrt. cost?

Max-Margin Planning with Explicit Cost Volume

- If we have an explicit cost volume, the cost of a trajectory can be directly queried.

Max-Margin Planning with Explicit Cost Volume



- If we have an explicit cost volume, the cost of a trajectory can be directly queried.
- We can use the max-margin objective to make the groundtruth trajectory have lower costs.

$$\operatorname{argmin}_{\theta} \sum_{\{(\hat{x}_i^t, \hat{y}_i^t)\}_{i=1 \dots N}} \max_i \sum_{t=1}^T C_{\theta}^t[x_t, y_t] - C_{\theta}^t[\hat{x}_i^t, \hat{y}_i^t] + d_i^t$$

function learnable parameter.

Max-Margin Planning with Explicit Cost Volume

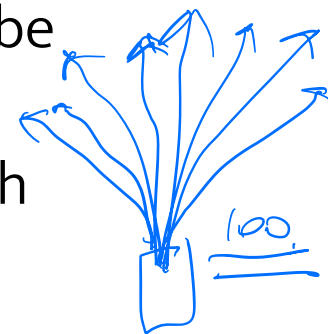
- If we have an explicit cost volume, the cost of a trajectory can be directly queried.
- We can use the max-margin objective to make the groundtruth trajectory have lower costs.
- Find the lowest cost trajectory among a batch of samples.

$$\operatorname{argmin}_{\theta} \sum_{\{(\hat{x}_i^t, \hat{y}_i^t)\}_{i=1 \dots N}} \max_i \sum_{t=1}^T C_{\theta}^t[x_t, y_t] - C_{\theta}^t[\hat{x}_i^t, \hat{y}_i^t] + d_i^t \rightarrow \text{task loss.}$$

Expert your traj.

Max-Margin Planning with Explicit Cost Volume

- If we have an explicit cost volume, the cost of a trajectory can be directly queried.
- We can use the max-margin objective to make the groundtruth trajectory have lower costs.
- Find the lowest cost trajectory among a batch of samples.
- Low-dimensional/known dynamics problems: External samplers



$$\underbrace{\text{planner}}_{\theta} \longleftrightarrow C[x, y, t]$$

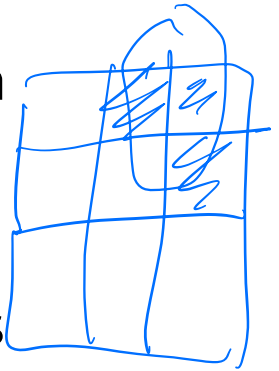
$$\underset{\theta}{\operatorname{argmin}} \sum_{\{(\hat{x}_i^t, \hat{y}_i^t)\}_{i=1 \dots N}} \max_i \sum_{t=1}^T C_{\theta}^t[x_t, y_t] - C_{\theta}^t[\hat{x}_i^t, \hat{y}_i^t] + d_i^t$$

samples of future traj.

your

Max-Margin Planning with Explicit Cost Volume

- If we have an explicit cost volume, the cost of a trajectory can be directly queried.
- We can use the max-margin objective to make the groundtruth trajectory have lower costs.
- Find the lowest cost trajectory among a batch of samples.
- Low-dimensional/known dynamics problems: External samplers
- In general, needs to perform optimization (e.g. DP)



$$\operatorname{argmin}_{\theta} \sum_{\{(\hat{x}_i^t, \hat{y}_i^t)\}_{i=1 \dots N}} \max_i \sum_{t=1}^T C_{\theta}^t[x_t, y_t] - C_{\theta}^t[\hat{x}_i^t, \hat{y}_i^t] + d_i^t$$

EBM Planning

- Energy-based framework also needs negative samples.

EBM Planning

- Energy-based framework also needs negative samples.
- “Pick” the groundtruth sample among others.

EBM Planning

- Energy-based framework also needs negative samples.
- “Pick” the groundtruth sample among others.
- If there isn’t an external sampler, we can either use autoregressive energy of sampling one dimension at a time, or gradient-based Langevin MCMC.

EBM Planning

- Energy-based framework also needs negative samples.
- “Pick” the groundtruth sample among others.
- If there isn’t an external sampler, we can either use autoregressive energy of sampling one dimension at a time, or gradient-based Langevin MCMC.

Langevin MCMC:
$$\tilde{\mathbf{y}}_i^k = \tilde{\mathbf{y}}_i^{k-1} - \lambda \left(\frac{1}{2} \nabla_{\mathbf{y}} E_{\theta}(\mathbf{x}_i, \mathbf{y}_i^{k-1}) + \omega^k \right), \omega^k \sim \mathcal{N}(0, \sigma).$$

EBM Planning

learnable energy.

- Energy-based framework also needs negative samples.
- “Pick” the groundtruth sample among others.
- If there isn't an external sampler, we can either use autoregressive energy of sampling one dimension at a time, or gradient-based Langevin MCMC.

Langevin MCMC: $\tilde{\mathbf{y}}_i^k = \tilde{\mathbf{y}}_i^{k-1} - \lambda \left(\frac{1}{2} \nabla_{\mathbf{y}} E_{\theta}(\mathbf{x}_i, \mathbf{y}_i^{k-1}) \right) + \omega^k, \omega^k \sim \mathcal{N}(0, \sigma)$.

optimization

noise

Loss: $\mathcal{L} = \sum_i -\log(p_{\theta}(\mathbf{y}_i | \mathbf{x}, \{\tilde{\mathbf{y}}_i\}_j))$

$$p_{\theta}(\mathbf{y}_i | \mathbf{x}, \{\tilde{\mathbf{y}}_i\}_j) = \frac{e^{-E_{\theta}(\mathbf{x}_i, \mathbf{y}_i)}}{e^{-E_{\theta}(\mathbf{x}_i, \mathbf{y}_i)} + \sum_j e^{-E_{\theta}(\mathbf{x}_i, \tilde{\mathbf{y}}_{i,j})}}$$

inference

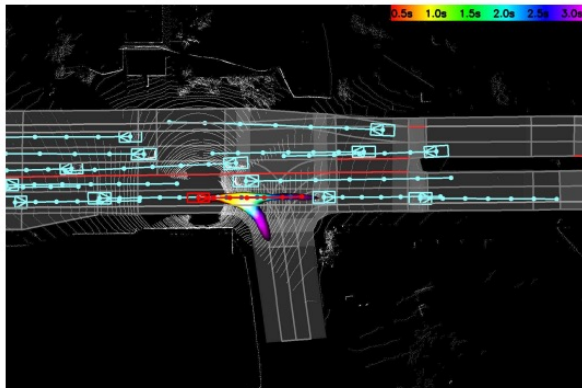
Non-Parametric Cost Volume for 2D Planning

$$C[x, y, t]$$

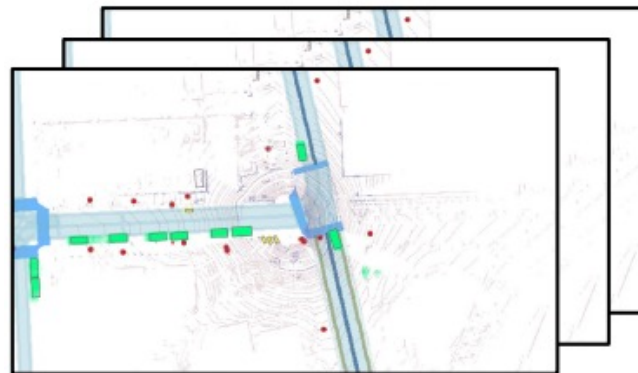
- Interpretability (both costs and planner inputs)

Compositionality.

Rasterization



Semantic Occupancy

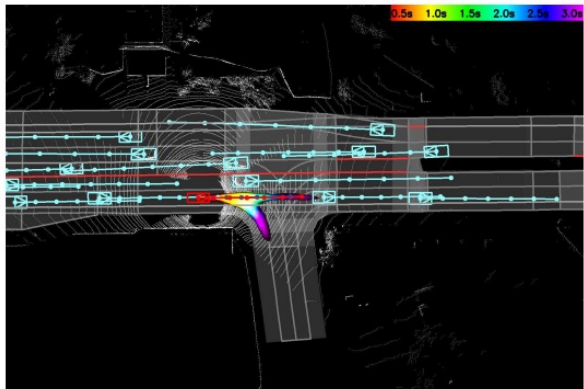


Non-Parametric Cost Volume for 2D Planning

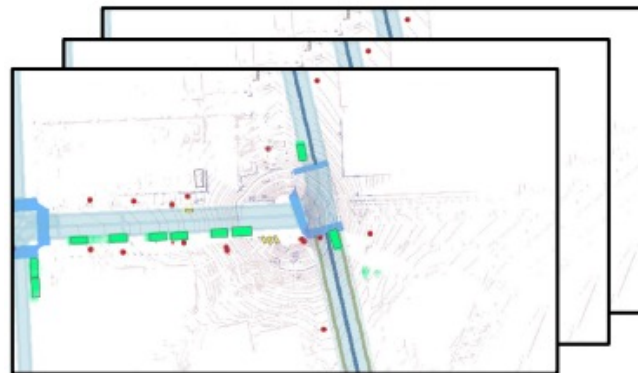
- Interpretability (both costs and planner inputs)
- Use spatial geometry to form cost from explicit objects

transformation of cost volume
→ equivariance.

Rasterization

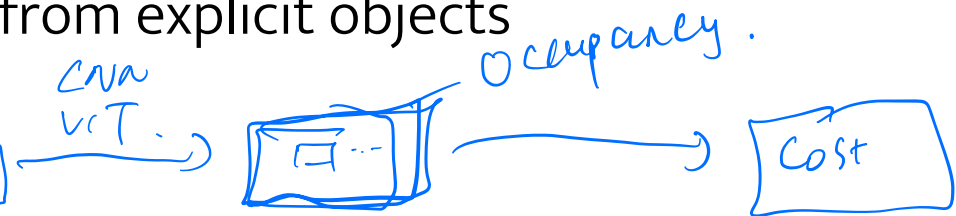


Semantic Occupancy

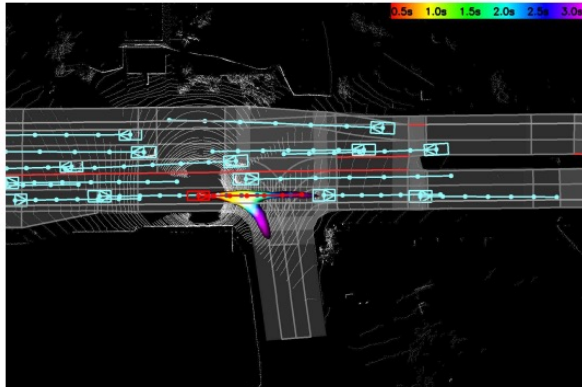


Non-Parametric Cost Volume for 2D Planning

- Interpretability (both costs and planner inputs)
- Use spatial geometry to form cost from explicit objects
- Predict spatial cost volume
 - Rasterize the scene for spatial inputs
 - Predict soft occupancy volumes (present and future)

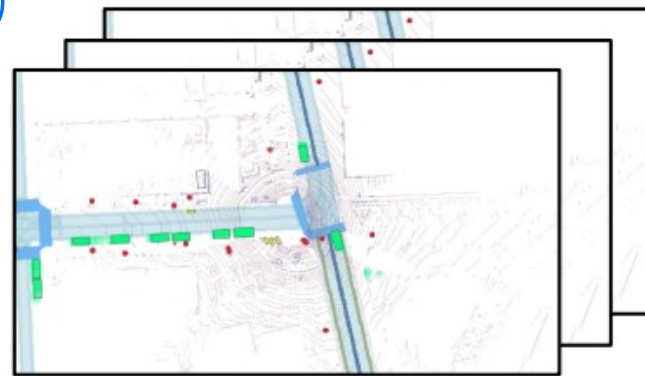


Rasterization



BEV

Semantic Occupancy

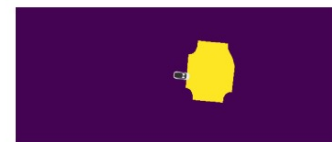


Learning Through Interpretable Predictions

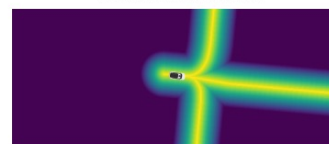
- Semantic occupancy, motion field, mapping, etc. as intermediate predictions.



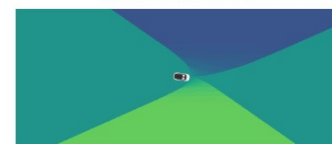
Drivable area



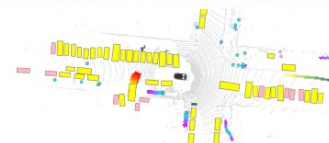
Intersections



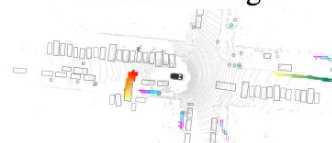
Reachable Distance Transform



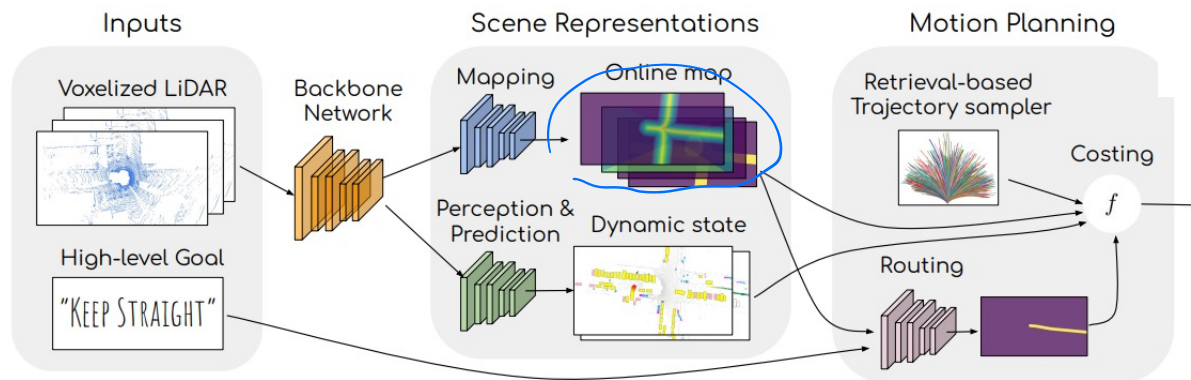
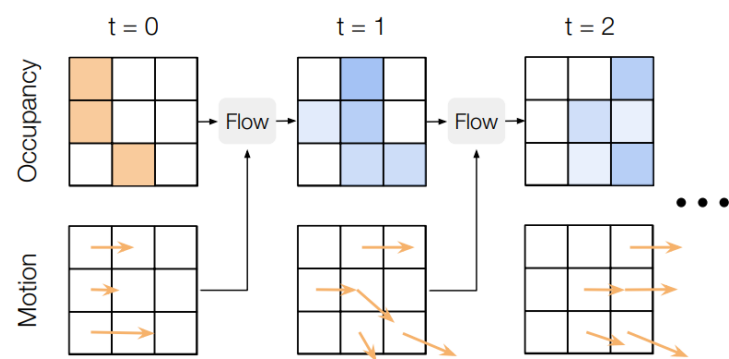
Reachable Angle



Occupancy

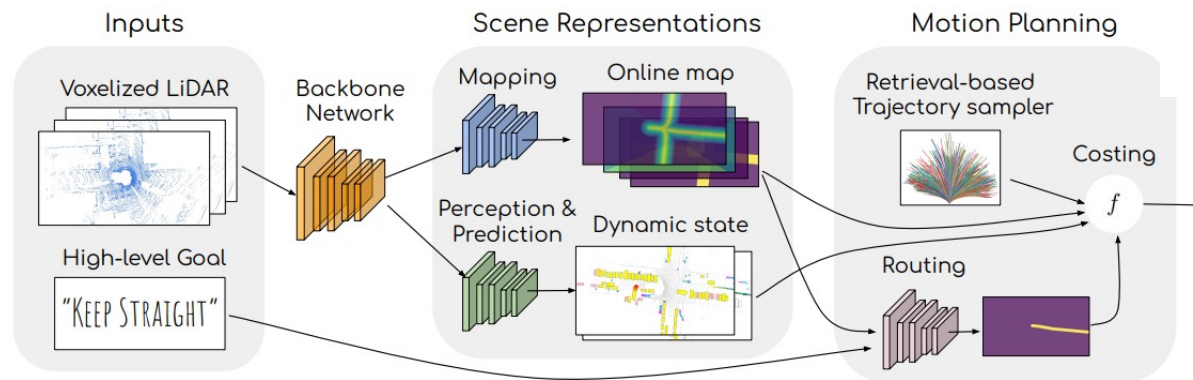
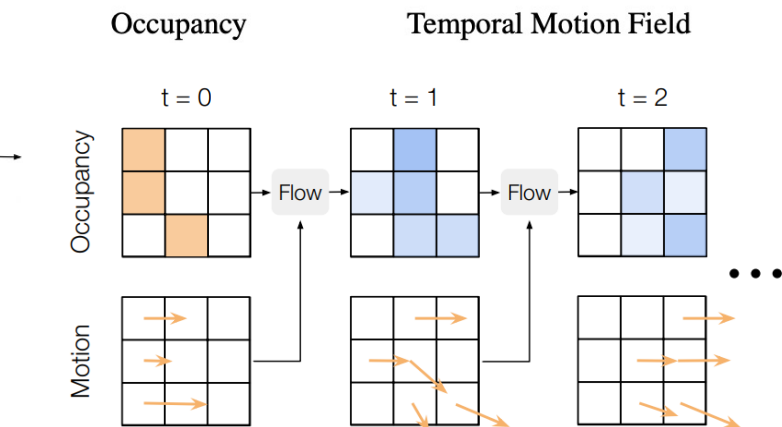
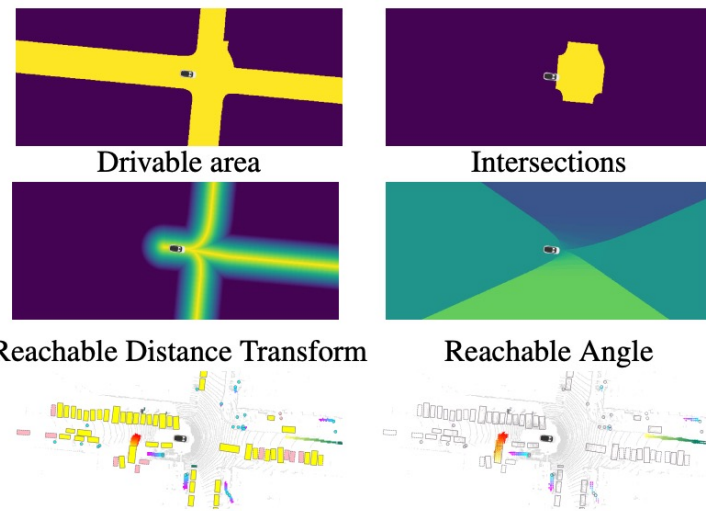


Temporal Motion Field



Learning Through Interpretable Predictions

- Semantic occupancy, motion field, mapping, etc. as intermediate predictions.
- Differentiable, supports end-to-end interpretable learning from perception to planning.



Value Iteration Networks

explicit optimization vs iterative network propagation.

- A network design for predicting cost volumes that are grounded from the classic value iteration algorithm.

Value Iteration Networks

- A network design for predicting cost volumes that are grounded from the classic value iteration algorithm.
- Classic VI algorithm:

Value Iteration Networks

- A network design for predicting cost volumes that are grounded from the classic value iteration algorithm.
- Classic VI algorithm:

$$V^*(s) = \max_{\pi} V^{\pi}(s)$$

↪ optimal policy.

Value Iteration Networks

- A network design for predicting cost volumes that are grounded from the classic value iteration algorithm.
- Classic VI algorithm:

$$V^*(s) = \max_{\pi} V^{\pi}(s)$$

$$V^{\pi}(s) = \mathbb{E}^{\pi} \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)$$

negative cost.

better policy better reward.

Value Iteration Networks

- A network design for predicting cost volumes that are grounded from the classic value iteration algorithm.
- Classic VI algorithm:

$$V^*(s) = \max_{\pi} V^{\pi}(s)$$

$$V^{\pi}(s) = \mathbb{E}^{\pi} \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)$$

$$Q_n(s, a) = R(s, a) + \gamma \sum_{s'} P(s'|s, a) V_n(s')$$

Bellman equation.

Value Iteration Networks

- A network design for predicting cost volumes that are grounded from the classic value iteration algorithm.
- Classic VI algorithm:

$$V^*(s) = \max_{\pi} V^{\pi}(s)$$

$$V^{\pi}(s) = \mathbb{E}^{\pi} \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)$$

$$Q_n(s, a) = R(s, a) + \gamma \sum_{s'} P(s'|s, a) V_n(s')$$

$$V_{n+1}(s) = \max_a Q_n(s, a)$$

Value Iteration Networks

- A network design for predicting cost volumes that are grounded from the classic value iteration algorithm.
- Classic VI algorithm:

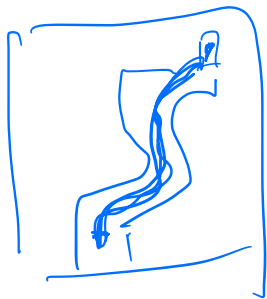
$$V^*(s) = \max_{\pi} V^{\pi}(s)$$

$$V^{\pi}(s) = \mathbb{E}^{\pi} \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)$$

$$\underline{Q_n(s, a)} = R(s, a) + \gamma \sum_{s'} P(s'|s, a) V_n(s')$$

$$V_{n+1}(s) = \max_a Q_n(s, a)$$

$$\pi^*(s) = \underline{\operatorname{argmax}_a} \underline{Q_{\infty}(s, a)}$$

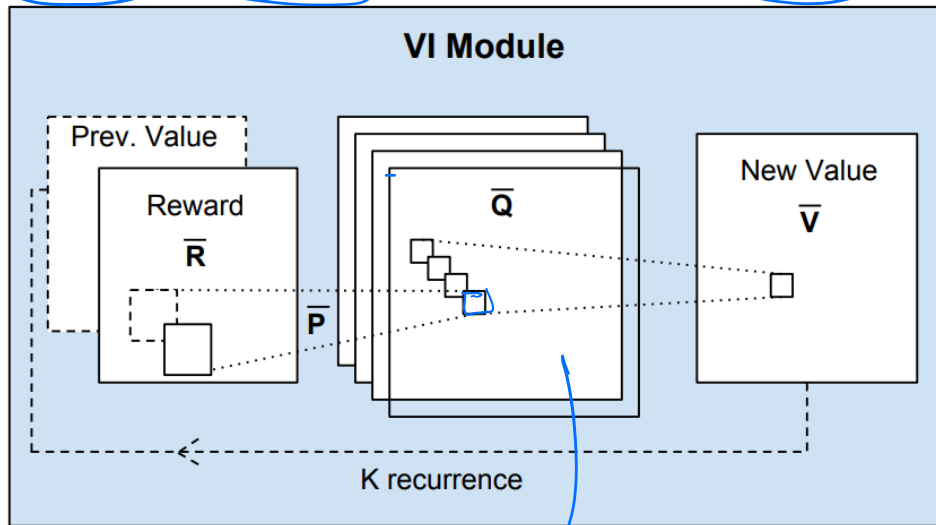
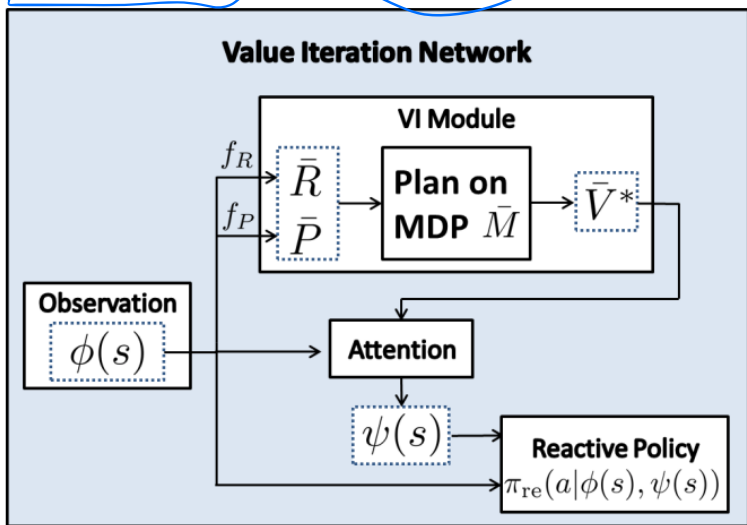


Value Iteration Networks

- Reward and previous value are fed into a network to generate Q of A channels. Transition matrix is convolutional kernel. Then Max-Pooling.

$$Q_n(s, a) = R(s, a) + \gamma \sum_{s'} P(s'|s, a) V_n(s') \quad V_{n+1}(s) = \max_a Q_n(s, a)$$

new information.
conv. kernel.
cost volume

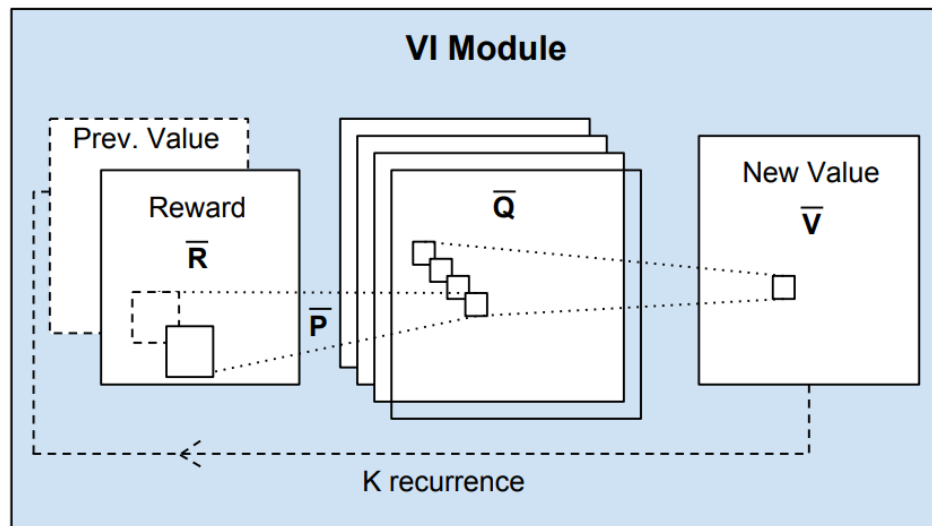
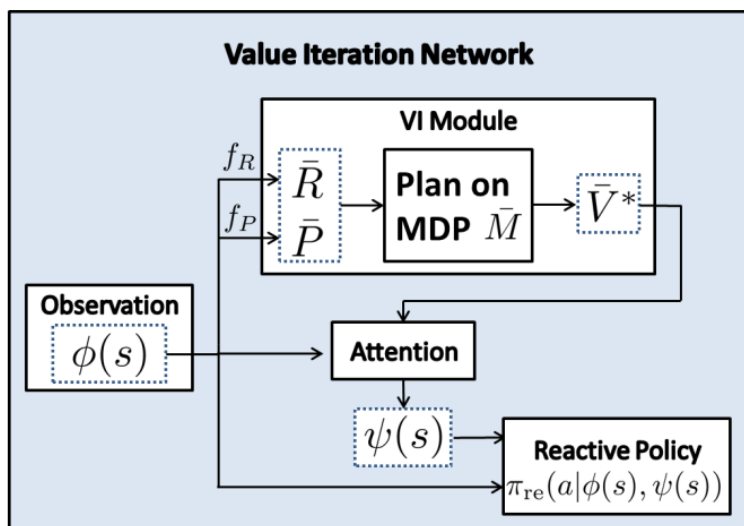


Cost volume

Value Iteration Networks

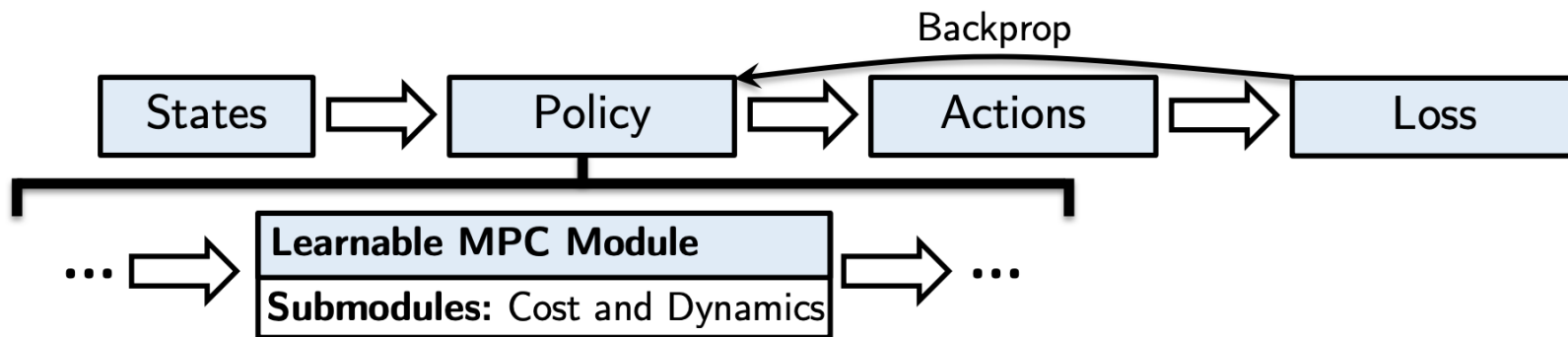
- Select the current state and choose an action from softmax.

$$\hat{a} \sim \text{softmax}_a(Q(s, a))$$



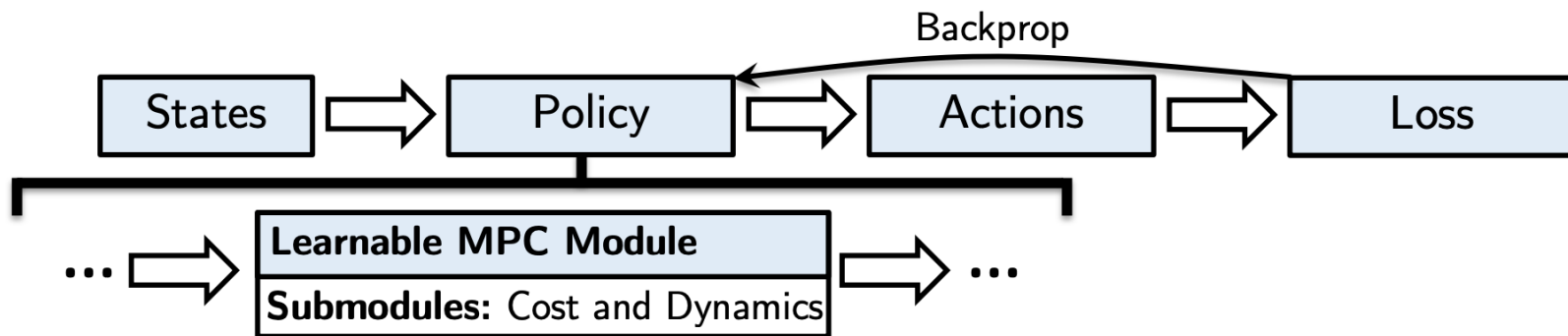
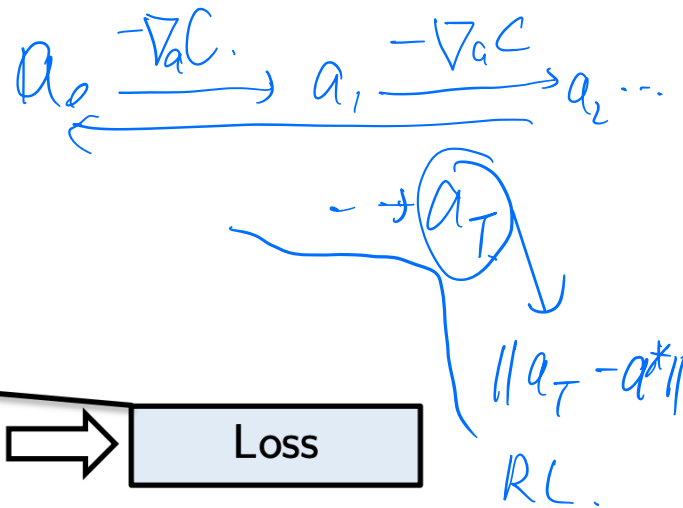
Backprop through Planning

- Treat planning as an end-to-end layer. Can be used for RL/Imitation.



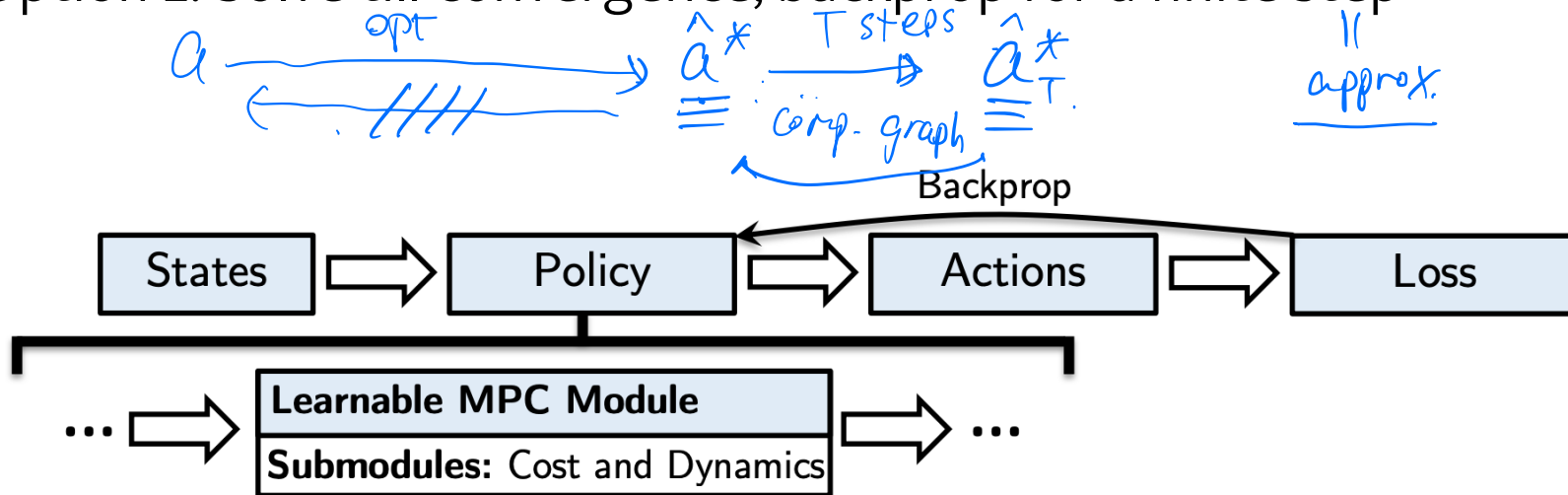
Backprop through Planning

- Treat planning as an end-to-end layer. Can be used for RL/Imitation.
- Option 1: Unrolling a finite number of steps



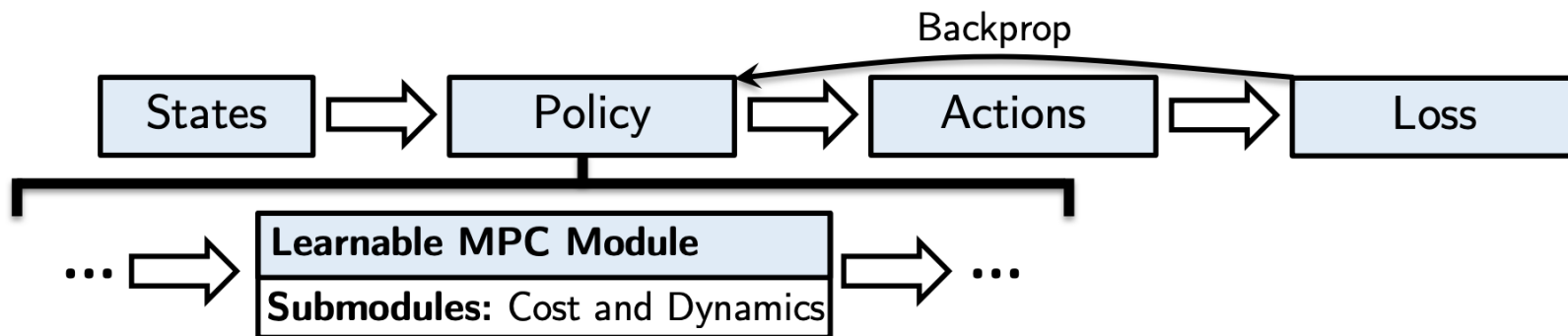
Backprop through Planning

- Treat planning as an end-to-end layer. Can be used for RL/Imitation.
- Option 1: Unrolling a finite number of steps
- Option 2: Solve till convergence, backprop for a finite step



Backprop through Planning

- Treat planning as an end-to-end layer. Can be used for RL/Imitation.
- Option 1: Unrolling a finite number of steps
- Option 2: Solve till convergence, backprop for a finite step
- Option 3: Converged at fixed point: Implicit differentiation



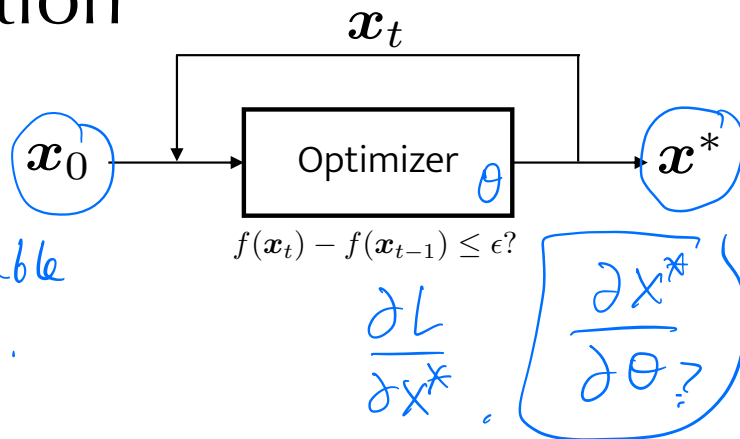
Implicit Differentiation

- Unconstrained case

$$\mathbf{x}^* = \operatorname{argmin}_{\mathbf{x}} f(\mathbf{x}; \boldsymbol{\theta}).$$

Cost \rightarrow

learnable
Cost

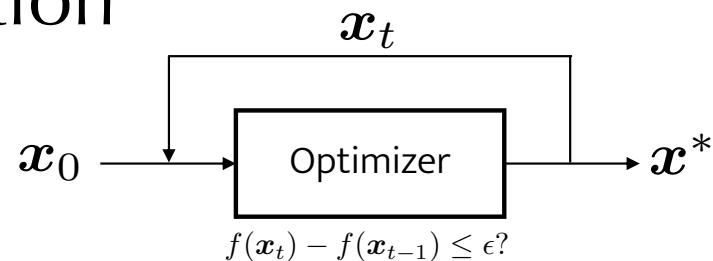


Implicit Differentiation

- Unconstrained case

$$\mathbf{x}^* = \underset{\mathbf{x}}{\operatorname{argmin}} f(\mathbf{x}; \boldsymbol{\theta}).$$

$$\mathbf{0} = \frac{d}{d\boldsymbol{\theta}} \underbrace{J_{f, \mathbf{x}^*}(\mathbf{x}^*; \boldsymbol{\theta})}$$



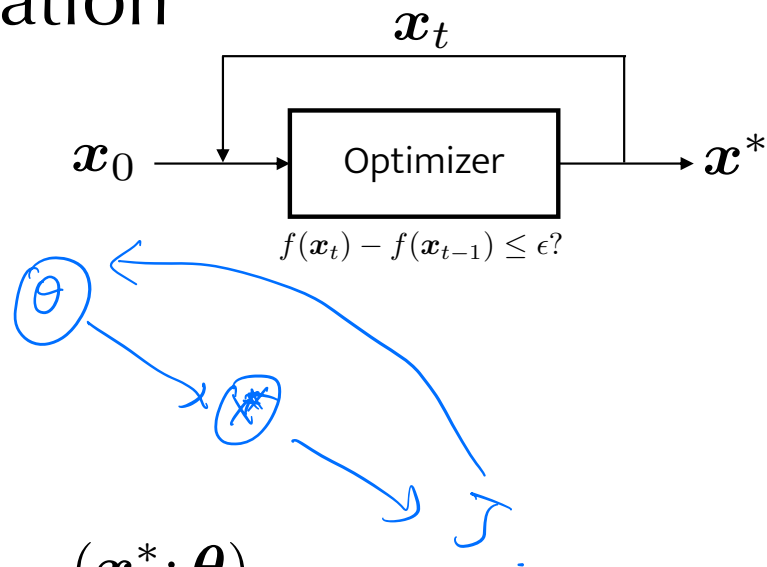
Implicit Differentiation

- Unconstrained case

$$\mathbf{x}^* = \operatorname{argmin}_{\mathbf{x}} f(\mathbf{x}; \boldsymbol{\theta}).$$

$$\mathbf{0} = \frac{d}{d\boldsymbol{\theta}} J_{f, \mathbf{x}^*}(\mathbf{x}^*; \boldsymbol{\theta})$$

$$\mathbf{0} = \frac{\partial}{\partial \mathbf{x}^*} J_{f, \mathbf{x}^*}(\mathbf{x}^*; \boldsymbol{\theta}) \frac{\partial \mathbf{x}^*}{\partial \boldsymbol{\theta}} + \frac{\partial}{\partial \boldsymbol{\theta}} J_{f, \mathbf{x}^*}(\mathbf{x}^*; \boldsymbol{\theta})$$



Implicit Differentiation

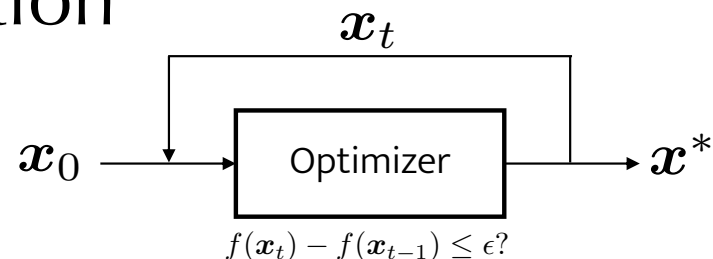
- Unconstrained case

$$\mathbf{x}^* = \underset{\mathbf{x}}{\operatorname{argmin}} f(\mathbf{x}; \boldsymbol{\theta}).$$

$$\mathbf{0} = \frac{d}{d\boldsymbol{\theta}} J_{f, \mathbf{x}^*}(\mathbf{x}^*; \boldsymbol{\theta})$$

$$\mathbf{0} = \frac{\partial}{\partial \mathbf{x}^*} J_{f, \mathbf{x}^*}(\mathbf{x}^*; \boldsymbol{\theta}) \frac{\partial \mathbf{x}^*}{\partial \boldsymbol{\theta}} + \frac{\partial}{\partial \boldsymbol{\theta}} J_{f, \mathbf{x}^*}(\mathbf{x}^*; \boldsymbol{\theta})$$

$$\mathbf{0} = H_{f, \mathbf{x}^*}(\mathbf{x}^*; \boldsymbol{\theta}) \frac{\partial \mathbf{x}^*}{\partial \boldsymbol{\theta}} + \frac{\partial}{\partial \boldsymbol{\theta}} J_{f, \mathbf{x}^*}(\mathbf{x}^*; \boldsymbol{\theta})$$



Implicit Differentiation

- Unconstrained case

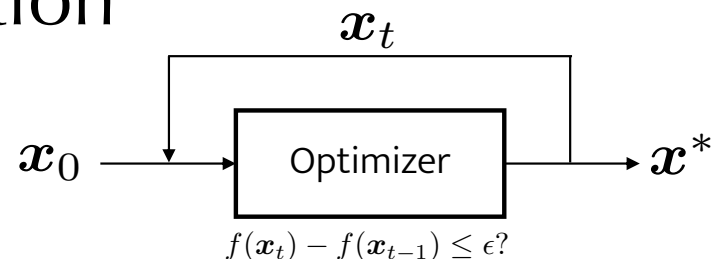
$$\mathbf{x}^* = \operatorname{argmin}_{\mathbf{x}} f(\mathbf{x}; \boldsymbol{\theta}).$$

$$\mathbf{0} = \frac{d}{d\boldsymbol{\theta}} J_{f, \mathbf{x}^*}(\mathbf{x}^*; \boldsymbol{\theta})$$

$$\mathbf{0} = \frac{\partial}{\partial \mathbf{x}^*} J_{f, \mathbf{x}^*}(\mathbf{x}^*; \boldsymbol{\theta}) \frac{\partial \mathbf{x}^*}{\partial \boldsymbol{\theta}} + \frac{\partial}{\partial \boldsymbol{\theta}} J_{f, \mathbf{x}^*}(\mathbf{x}^*; \boldsymbol{\theta})$$

$$\mathbf{0} = H_{f, \mathbf{x}^*}(\mathbf{x}^*; \boldsymbol{\theta}) \frac{\partial \mathbf{x}^*}{\partial \boldsymbol{\theta}} + \frac{\partial}{\partial \boldsymbol{\theta}} J_{f, \mathbf{x}^*}(\mathbf{x}^*; \boldsymbol{\theta})$$

$$\frac{\partial \mathbf{x}^*}{\partial \boldsymbol{\theta}} = H_{f, \mathbf{x}^*}(\mathbf{x}^*; \boldsymbol{\theta})^{-1} \frac{\partial}{\partial \boldsymbol{\theta}} J_{f, \mathbf{x}^*}(\mathbf{x}^*; \boldsymbol{\theta}).$$



Implicit Differentiation

- How to compute Hessian inverse vector product?

Implicit Differentiation

- How to compute Hessian inverse vector product?
- Conjugate gradient, solve $Ax = b$

$$x = A^{-1}b$$

$H^{-1}g$

↓

inv.

Conjugate Gradient Method

$$\mathbf{r}_0 := \mathbf{b} - \mathbf{A}\mathbf{x}_0$$

if \mathbf{r}_0 is sufficiently small, then return \mathbf{x}_0 as the result

$$\mathbf{p}_0 := \mathbf{r}_0$$

$$k := 0$$

repeat

$$\alpha_k := \frac{\mathbf{r}_k^\top \mathbf{r}_k}{\mathbf{p}_k^\top \mathbf{A} \mathbf{p}_k}$$

$$\mathbf{x}_{k+1} := \mathbf{x}_k + \alpha_k \mathbf{p}_k$$

$$\mathbf{r}_{k+1} := \mathbf{r}_k - \alpha_k \mathbf{A} \mathbf{p}_k$$

if \mathbf{r}_{k+1} is sufficiently small, then exit loop

$$\beta_k := \frac{\mathbf{r}_{k+1}^\top \mathbf{r}_{k+1}}{\mathbf{r}_k^\top \mathbf{r}_k}$$

$$\mathbf{p}_{k+1} := \mathbf{r}_{k+1} + \beta_k \mathbf{p}_k$$

$$k := k + 1$$

end repeat

return \mathbf{x}_{k+1} as the result

Implicit Differentiation

- How to compute Hessian inverse vector product?
- Conjugate gradient, solve $A\mathbf{x} = \mathbf{b}$
- Neumann series (finite truncation)

$$(I - A)^{-1} = \sum_{k=0}^{\infty} A^k.$$

- Same as backprop the last K steps (Option 2).
- Memory savings.

Conjugate Gradient Method

$$\mathbf{r}_0 := \mathbf{b} - \mathbf{A}\mathbf{x}_0$$

if \mathbf{r}_0 is sufficiently small, then return \mathbf{x}_0 as the result

$$\mathbf{p}_0 := \mathbf{r}_0$$

$$k := 0$$

repeat

$$\alpha_k := \frac{\mathbf{r}_k^\top \mathbf{r}_k}{\mathbf{p}_k^\top \mathbf{A} \mathbf{p}_k}$$

$$\mathbf{x}_{k+1} := \mathbf{x}_k + \alpha_k \mathbf{p}_k$$

$$\mathbf{r}_{k+1} := \mathbf{r}_k - \alpha_k \mathbf{A} \mathbf{p}_k$$

if \mathbf{r}_{k+1} is sufficiently small, then exit loop

$$\beta_k := \frac{\mathbf{r}_{k+1}^\top \mathbf{r}_{k+1}}{\mathbf{r}_k^\top \mathbf{r}_k}$$

$$\mathbf{p}_{k+1} := \mathbf{r}_{k+1} + \beta_k \mathbf{p}_k$$

$$k := k + 1$$

end repeat

return \mathbf{x}_{k+1} as the result

Differentiable LQR

- Now add linear equality constraints on the dynamics and initialization.

$$\tau_{1:T} = \{\underline{x}_t, u_t\}_{1:T}$$

State Control.

$$\begin{aligned} & \underset{\tau_{1:T}}{\operatorname{argmin}} \sum_{t=1}^T \frac{1}{2} \tau_t^\top C_t \tau_t + c_t^\top \tau_t \\ & \text{subject to } x_{t+1} = F_t \tau_t + f_t, x_1 = x_{\text{init}}. \end{aligned}$$

quadratic cost

linear dynamics

learnable. learnable.

Differentiable LQR

- Now add linear equality constraints on the dynamics and initialization.

$$\tau_{1:T} = \{x_t, u_t\}_{1:T}$$

- Chain rule: $\frac{\partial \ell}{\partial \theta} = \frac{\partial \ell}{\partial \tau_{1:T}^*} \frac{\partial \tau_{1:T}^*}{\partial \theta}$.

$$\operatorname{argmin}_{\tau_{1:T}} \sum_{t=1}^T \frac{1}{2} \tau_t^\top C_t \tau_t + c_t^\top \tau_t$$

$$\text{subject to } x_{t+1} = F_t \tau_t + f_t, x_1 = x_{\text{init}}.$$

Differentiable LQR

- Now add linear equality constraints on the dynamics and initialization.

$$\tau_{1:T} = \{x_t, u_t\}_{1:T}$$

- Chain rule: $\frac{\partial \ell}{\partial \theta} = \frac{\partial \ell}{\partial \tau_{1:T}^*} \frac{\partial \tau_{1:T}^*}{\partial \theta}$.

$$\operatorname{argmin}_{\tau_{1:T}} \sum_{t=1}^T \frac{1}{2} \tau_t^\top C_t \tau_t + c_t^\top \tau_t$$

subject to $x_{t+1} = F_t \tau_t + f_t, x_1 = x_{\text{init}}$.

General QP:

$$\mathbf{x}^* = \operatorname{argmin} \frac{1}{2} \mathbf{x}^\top Q \mathbf{x} + \mathbf{c}^\top \mathbf{x}$$

subject to $A\mathbf{x} = \mathbf{b}$.

Differentiable LQR

- Now add linear equality constraints on the dynamics and initialization.

$$\tau_{1:T} = \{x_t, u_t\}_{1:T}$$

- Chain rule: $\frac{\partial \ell}{\partial \theta} = \frac{\partial \ell}{\partial \tau_{1:T}^*} \frac{\partial \tau_{1:T}^*}{\partial \theta}$.
- KKT:

$$\operatorname{argmin}_{\tau_{1:T}} \sum_{t=1}^T \frac{1}{2} \tau_t^\top C_t \tau_t + c_t^\top \tau_t$$

subject to $x_{t+1} = F_t \tau_t + f_t, x_1 = x_{\text{init}}$.

General QP:

$$\mathbf{x}^* = \operatorname{argmin} \frac{1}{2} \mathbf{x}^\top Q \mathbf{x} + \mathbf{c}^\top \mathbf{x}$$

subject to $A\mathbf{x} = \mathbf{b}$.

Differentiable LQR

- Now add linear equality constraints on the dynamics and initialization.

$$\tau_{1:T} = \{x_t, u_t\}_{1:T}$$

- Chain rule: $\frac{\partial \ell}{\partial \theta} = \frac{\partial \ell}{\partial \tau_{1:T}^*} \frac{\partial \tau_{1:T}^*}{\partial \theta}$.
- KKT:

$$\begin{bmatrix} Q & A^\top \\ A & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{x}^* \\ \boldsymbol{\lambda}^* \end{bmatrix} = \begin{bmatrix} -\mathbf{c} \\ \mathbf{b} \end{bmatrix}$$

$$K \begin{bmatrix} \mathbf{x}^* \\ \boldsymbol{\lambda}^* \end{bmatrix} = \mathbf{v}.$$

$$\operatorname{argmin}_{\tau_{1:T}} \sum_{t=1}^T \frac{1}{2} \tau_t^\top C_t \tau_t + c_t^\top \tau_t$$

subject to $x_{t+1} = F_t \tau_t + f_t, x_1 = x_{\text{init}}$.

General QP:

$$\mathbf{x}^* = \operatorname{argmin} \frac{1}{2} \mathbf{x}^\top \underline{Q} \mathbf{x} + \underline{\mathbf{c}}^\top \mathbf{x}$$

subject to $\underline{A} \mathbf{x} = \underline{\mathbf{b}}$.

In classic LQR solver, the Riccati recursion solves this linear system.

Differentiable LQR

- Apply differentiation $\frac{d}{d\theta} \left(K \begin{bmatrix} \mathbf{x}^* \\ \lambda^* \end{bmatrix} \right) = \frac{dv}{d\theta}. \quad \frac{dK}{d\theta} \begin{bmatrix} \mathbf{x}^* \\ \lambda^* \end{bmatrix} + K \begin{bmatrix} \frac{d\mathbf{x}^*}{d\theta} \\ \frac{d\lambda^*}{d\theta} \end{bmatrix} = \frac{dv}{d\theta}$

Differentiable LQR

• Apply differentiation $\frac{d}{d\theta} \left(K \begin{bmatrix} \mathbf{x}^* \\ \lambda^* \end{bmatrix} \right) = \frac{dv}{d\theta} \cdot \frac{dK}{d\theta} \begin{bmatrix} \mathbf{x}^* \\ \lambda^* \end{bmatrix} + K \begin{bmatrix} \frac{d\mathbf{x}^*}{d\theta} \\ \frac{d\lambda^*}{d\theta} \end{bmatrix} = \frac{dv}{d\theta}$

$$K \begin{bmatrix} \frac{d\mathbf{x}^*}{d\theta} \\ \frac{d\lambda^*}{d\theta} \end{bmatrix} = K \begin{bmatrix} \frac{d\mathbf{x}^*}{d\mathbf{c}} & \frac{d\mathbf{x}^*}{d\mathbf{b}} & \frac{d\mathbf{x}^*}{dQ} & \frac{d\mathbf{x}^*}{dA} \\ \frac{d\lambda^*}{d\mathbf{c}} & \frac{d\lambda^*}{d\mathbf{b}} & \frac{d\lambda^*}{dQ} & \frac{d\lambda^*}{dA} \end{bmatrix} = \begin{bmatrix} -I & 0 & -\mathbf{x}^* & -\lambda^* \\ 0 & I & 0 & -\mathbf{x}^* \end{bmatrix}$$

Differentiable LQR

• Apply differentiation $\frac{d}{d\theta} \left(K \begin{bmatrix} \mathbf{x}^* \\ \lambda^* \end{bmatrix} \right) = \frac{dv}{d\theta} \cdot \frac{dK}{d\theta} \begin{bmatrix} \mathbf{x}^* \\ \lambda^* \end{bmatrix} + K \begin{bmatrix} \frac{d\mathbf{x}^*}{d\theta} \\ \frac{d\lambda^*}{d\theta} \end{bmatrix} = \frac{dv}{d\theta}$

$$K \begin{bmatrix} \frac{d\mathbf{x}^*}{d\theta} \\ \frac{d\lambda^*}{d\theta} \end{bmatrix} = K \begin{bmatrix} \frac{d\mathbf{x}^*}{d\mathbf{c}} & \frac{d\mathbf{x}^*}{d\mathbf{b}} & \frac{d\mathbf{x}^*}{dQ} & \frac{d\mathbf{x}^*}{dA} \\ \frac{d\lambda^*}{d\mathbf{c}} & \frac{d\lambda^*}{d\mathbf{b}} & \frac{d\lambda^*}{dQ} & \frac{d\lambda^*}{dA} \end{bmatrix} = \begin{bmatrix} -I & 0 & -\mathbf{x}^* & -\lambda^* \\ 0 & I & 0 & -\mathbf{x}^* \end{bmatrix}$$

$$K \frac{\partial \ell}{\partial \mathbf{z}^*} \begin{bmatrix} \frac{d\mathbf{x}^*}{d\mathbf{c}} & \frac{d\mathbf{x}^*}{d\mathbf{b}} \\ \frac{d\lambda^*}{d\mathbf{c}} & \frac{d\lambda^*}{d\mathbf{b}} \end{bmatrix} = \begin{bmatrix} -\frac{\partial \ell}{\partial \mathbf{x}^*} \\ \mathbf{0} \end{bmatrix} \quad K \mathbf{d}^* = \begin{bmatrix} -\frac{\partial \ell}{\partial \mathbf{x}^*} \\ \mathbf{0} \end{bmatrix}$$

Differentiable LQR

- Apply differentiation $\frac{d}{d\theta} \left(K \begin{bmatrix} \mathbf{x}^* \\ \lambda^* \end{bmatrix} \right) = \frac{dv}{d\theta}$. $\frac{dK}{d\theta} \begin{bmatrix} \mathbf{x}^* \\ \lambda^* \end{bmatrix} + K \begin{bmatrix} \frac{d\mathbf{x}^*}{d\theta} \\ \frac{d\lambda^*}{d\theta} \end{bmatrix} = \frac{dv}{d\theta}$

$$K \begin{bmatrix} \frac{d\mathbf{x}^*}{d\theta} \\ \frac{d\lambda^*}{d\theta} \end{bmatrix} = K \begin{bmatrix} \frac{d\mathbf{x}^*}{d\mathbf{c}} & \frac{d\mathbf{x}^*}{d\mathbf{b}} & \frac{d\mathbf{x}^*}{dQ} & \frac{d\mathbf{x}^*}{dA} \\ \frac{d\lambda^*}{d\mathbf{c}} & \frac{d\lambda^*}{d\mathbf{b}} & \frac{d\lambda^*}{dQ} & \frac{d\lambda^*}{dA} \end{bmatrix} = \begin{bmatrix} -I & 0 & -\mathbf{x}^* & -\lambda^* \\ 0 & I & 0 & -\mathbf{x}^* \end{bmatrix}$$

$$K \frac{\partial \ell}{\partial \mathbf{z}^*} \begin{bmatrix} \frac{d\mathbf{x}^*}{d\mathbf{c}} & \frac{d\mathbf{x}^*}{d\mathbf{b}} \\ \frac{d\lambda^*}{d\mathbf{c}} & \frac{d\lambda^*}{d\mathbf{b}} \end{bmatrix} = \begin{bmatrix} -\frac{\partial \ell}{\partial \mathbf{x}^*} \\ \mathbf{0} \end{bmatrix}$$

$K \mathbf{d}^* = \begin{bmatrix} -\frac{\partial \ell}{\partial \mathbf{x}^*} \\ \mathbf{0} \end{bmatrix}$

backward pass

- Equivalent QP:

$$\mathbf{d}^* = \underset{\mathbf{d}}{\operatorname{argmin}} \frac{1}{2} \mathbf{d}^\top Q \mathbf{d} + \frac{\partial \ell}{\partial \mathbf{x}^*}^\top \mathbf{d},$$

subject to $A\mathbf{d} = \mathbf{0}$.

Differentiable LQR

• Apply differentiation $\frac{d}{d\theta} \left(K \begin{bmatrix} \mathbf{x}^* \\ \boldsymbol{\lambda}^* \end{bmatrix} \right) = \frac{dv}{d\theta}$. $\frac{dK}{d\theta} \begin{bmatrix} \mathbf{x}^* \\ \boldsymbol{\lambda}^* \end{bmatrix} + K \begin{bmatrix} \frac{d\mathbf{x}^*}{d\theta} \\ \frac{d\boldsymbol{\lambda}^*}{d\theta} \end{bmatrix} = \frac{dv}{d\theta}$

$$K \begin{bmatrix} \frac{d\mathbf{x}^*}{d\theta} \\ \frac{d\boldsymbol{\lambda}^*}{d\theta} \end{bmatrix} = K \begin{bmatrix} \frac{d\mathbf{x}^*}{d\mathbf{c}} & \frac{d\mathbf{x}^*}{d\mathbf{b}} & \frac{d\mathbf{x}^*}{dQ} & \frac{d\mathbf{x}^*}{dA} \\ \frac{d\boldsymbol{\lambda}^*}{d\mathbf{c}} & \frac{d\boldsymbol{\lambda}^*}{d\mathbf{b}} & \frac{d\boldsymbol{\lambda}^*}{dQ} & \frac{d\boldsymbol{\lambda}^*}{dA} \end{bmatrix} = \begin{bmatrix} -I & 0 & -\mathbf{x}^* & -\boldsymbol{\lambda}^* \\ 0 & I & 0 & -\mathbf{x}^* \end{bmatrix}$$

$$K \frac{\partial \ell}{\partial \mathbf{z}^*} \begin{bmatrix} \frac{d\mathbf{x}^*}{d\mathbf{c}} & \frac{d\mathbf{x}^*}{d\mathbf{b}} \\ \frac{d\boldsymbol{\lambda}^*}{d\mathbf{c}} & \frac{d\boldsymbol{\lambda}^*}{d\mathbf{b}} \end{bmatrix} = \begin{bmatrix} -\frac{\partial \ell}{\partial \mathbf{x}^*} \\ \mathbf{0} \end{bmatrix} \quad K \mathbf{d}^* = \begin{bmatrix} -\frac{\partial \ell}{\partial \mathbf{x}^*} \\ \mathbf{0} \end{bmatrix}$$

- Equivalent QP:

$$\mathbf{d}^* = \underset{\mathbf{d}}{\operatorname{argmin}} \frac{1}{2} \mathbf{d}^\top Q \mathbf{d} + \frac{\partial \ell}{\partial \mathbf{x}^*}{}^\top \mathbf{d},$$

subject to $A\mathbf{d} = \mathbf{0}$.

$$\frac{\partial \ell}{\partial Q} = \frac{1}{2} (\mathbf{d}_x^* \otimes \mathbf{x}^* + \mathbf{x}^* \otimes \mathbf{d}_x^*)$$

$$\frac{\partial \ell}{\partial A} = \mathbf{d}_\lambda^* \otimes \mathbf{x}^* + \boldsymbol{\lambda}^* \otimes \mathbf{d}_x^*.$$

Differentiable LQR

- The backward pass can also be formulated as a LQR problem.
- Swap c to $\nabla_{\tau^*} \ell$ and f to 0.

Module 1 Differentiable LQR

(The LQR algorithm is defined in [appendix A](#))

Input: Initial state x_{init}

Parameters: $\theta = \{C, c, F, f\}$

Forward Pass:

- 1: $\tau_{1:T}^* = \text{LQR}_T(x_{\text{init}}; C, c, F, f)$
- 2: Compute $\lambda_{1:T}^*$ with (7)

▷ Solve (2)

Backward Pass:

- 1: $d_{\tau_{1:T}}^* = \text{LQR}_T(0; C, \nabla_{\tau^*} \ell, F, 0)$ ▷ Solve (9), ideally reusing the factorizations from the forward pass
- 2: Compute $d_{\lambda_{1:T}}^*$ with (7)
- 3: Compute the derivatives of ℓ with respect to C, c, F, f , and x_{init} with (8)

Differentiable MPC

- What about general MPC?

$$\operatorname{argmin}_{x_{1:T} \in \mathcal{X}, u_{1:T} \in \mathcal{U}} \sum_{t=1}^T C_t(x_t, u_t)$$

subject to $x_{t+1} = \underline{f}(x_t, u_t), x_1 = x_{\text{init}}$.

learnable cost.

Differentiable MPC

- What about general MPC?

$$\operatorname{argmin}_{x_{1:T} \in \mathcal{X}, u_{1:T} \in \mathcal{U}} \sum_{t=1}^T C_t(x_t, u_t)$$

subject to $x_{t+1} = f(x_t, u_t), x_1 = x_{\text{init}}$.

- Use Taylor expansion to approximate.

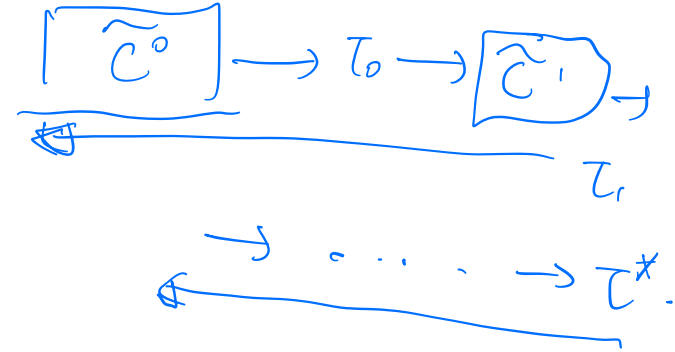
$$\underline{\tilde{C}}_{\theta,t}^i = C_{\theta,t}(\tau_t^i) + \underbrace{p_t^i}^e \top (\tau_t - \tau_t^i) + \frac{1}{2} (\tau_t - \tau_t^i) \top \underbrace{H_t^i}_{\circ} (\tau_t - \tau_t^i).$$

Differentiable MPC

- What about general MPC?

$$\operatorname{argmin}_{x_{1:T} \in \mathcal{X}, u_{1:T} \in \mathcal{U}} \sum_{t=1}^T C_t(x_t, u_t)$$

subject to $x_{t+1} = f(x_t, u_t), x_1 = x_{\text{init}}$.



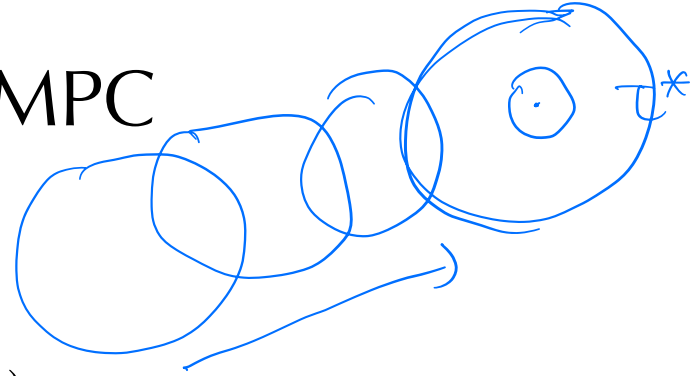
- Use Taylor expansion to approximate.

$$\tilde{C}_{\theta,t}^i = C_{\theta,t}(\tau_t^i) + p_t^{i\top} (\tau_t - \tau_t^i) + \frac{1}{2} (\tau_t - \tau_t^i)^\top H_t^i (\tau_t - \tau_t^i).$$

- Fixed point iteration

$$\tau^{i+1} = \operatorname{argmin}_{\tau} \sum_t^T \tilde{C}_t^i(\tau_t^i).$$

Differentiable MPC



- What about general MPC?

$$\operatorname{argmin}_{x_{1:T} \in \mathcal{X}, u_{1:T} \in \mathcal{U}} \sum_{t=1}^T C_t(x_t, u_t)$$

subject to $x_{t+1} = f(x_t, u_t), x_1 = x_{\text{init}}$.

- Use Taylor expansion to approximate.

$$\tilde{C}_{\theta,t}^i = C_{\theta,t}(\tau_t^i) + p_t^{i\top} (\tau_t - \tau_t^i) + \frac{1}{2} (\tau_t - \tau_t^i)^\top H_t^i (\tau_t - \tau_t^i).$$

- Fixed point iteration

$$\tau^{i+1} = \operatorname{argmin}_{\tau} \sum_t^T \tilde{C}_t^i(\tau_t^i).$$

- Backward only depends on final quadratic approximation.

Behavioral vs. Trajectory Planning

- Gradient-based optimization provides a locally optimized trajectory.

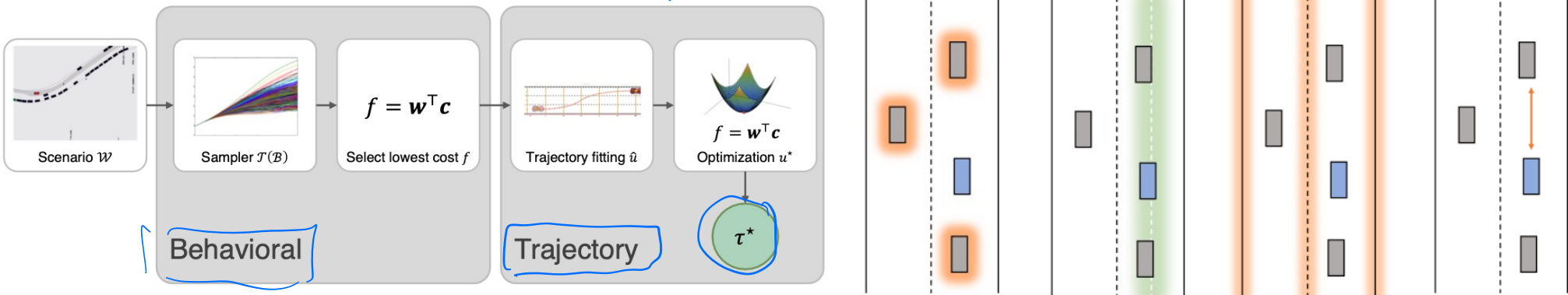
Higher level decision.

initial trajectory.

optimized traj.

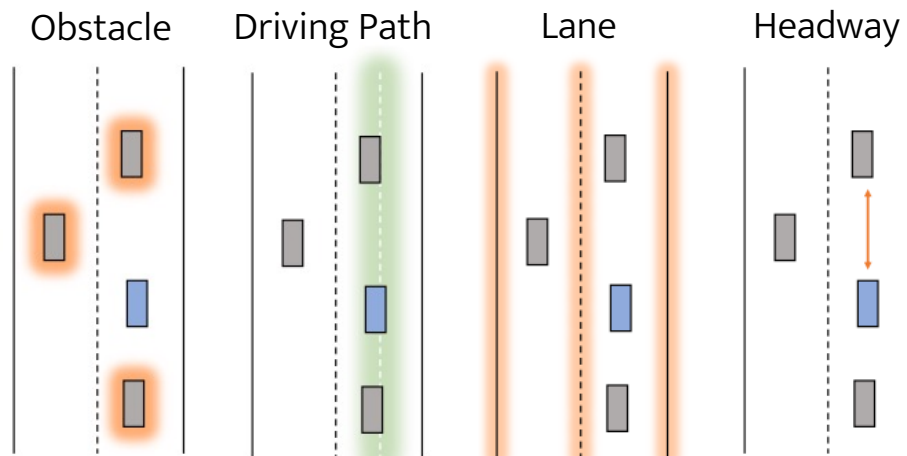
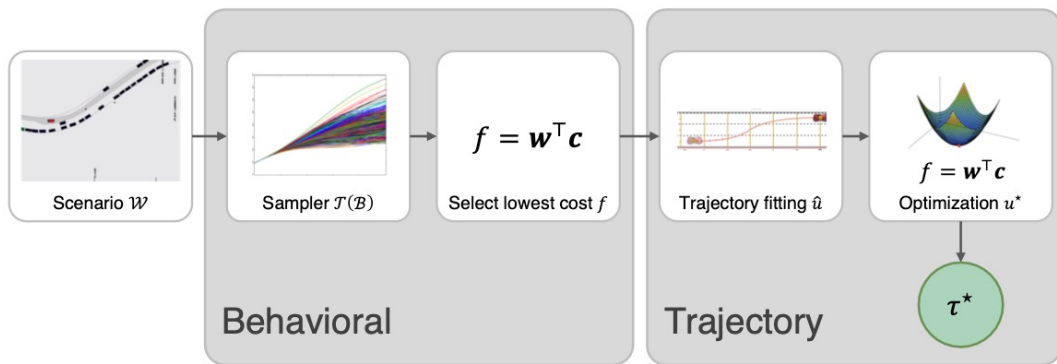


Obstacle Driving Path Lane Headway



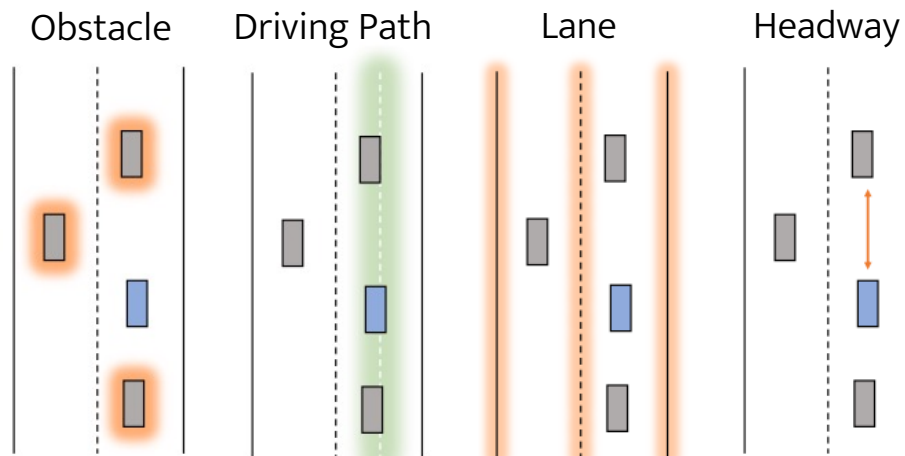
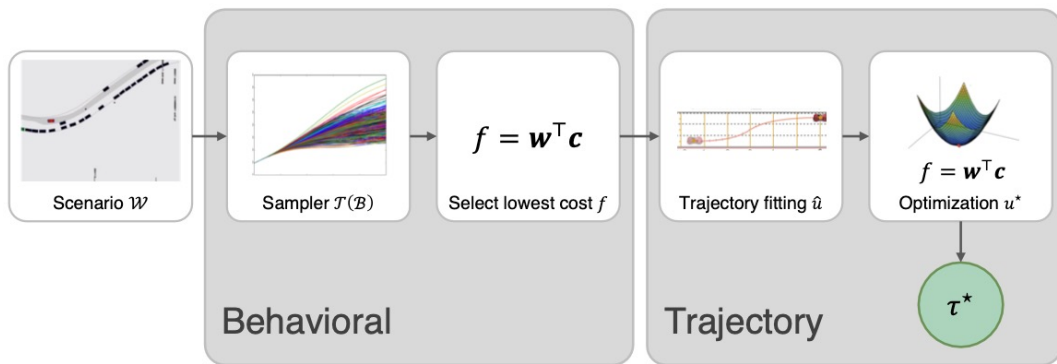
Behavioral vs. Trajectory Planning

- Gradient-based optimization provides a locally optimized trajectory.
- Samples may be needed for reasoning global structure.



Behavioral vs. Trajectory Planning

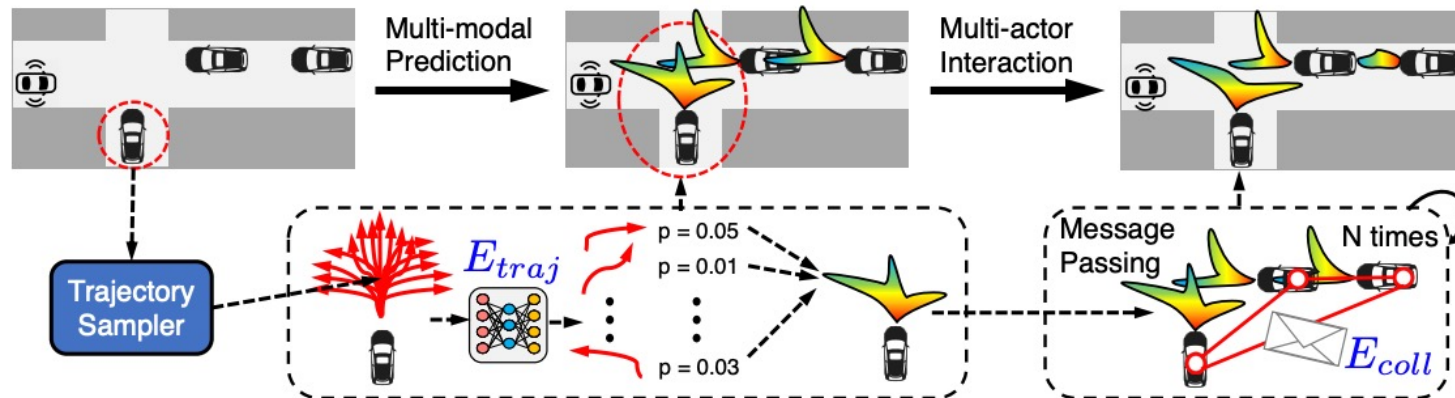
- Gradient-based optimization provides a locally optimized trajectory.
- Samples may be needed for reasoning global structure.
- Can learn together using the same learned costs.



Planning with Social Reasoning

- Jointly reason the future trajectories of multiple agents as an energy-based graphical model.

$$p(\mathbf{s}_1, \dots, \mathbf{s}_N \mid \mathbf{X}) = \frac{1}{Z} \exp(-E_{\theta}(\mathbf{s}_1, \dots, \mathbf{s}_N) \mid \mathbf{X})$$



Planning with Social Reasoning

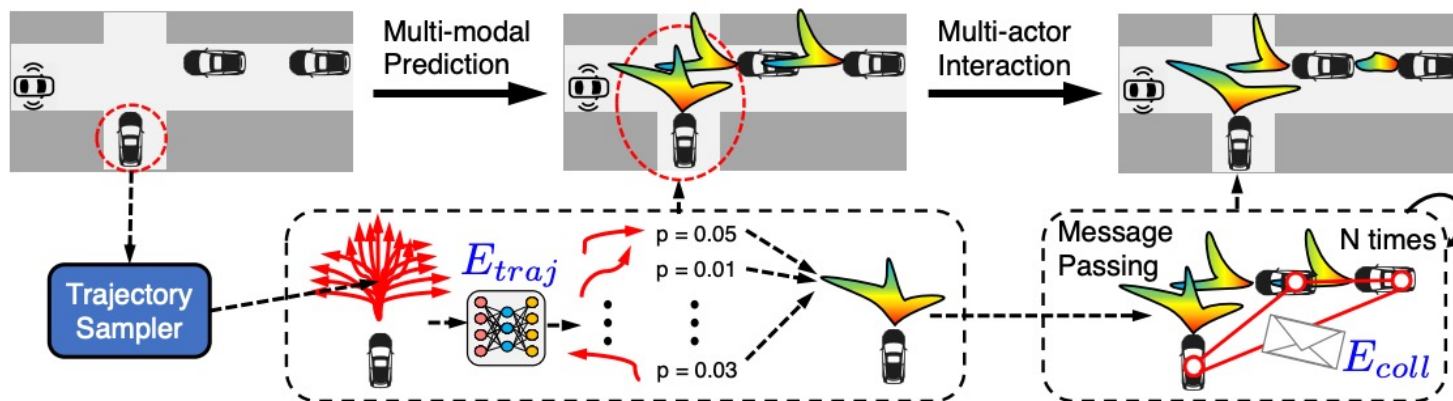
- Jointly reason the future trajectories of multiple agents as an energy-based graphical model.

$$p(\mathbf{s}_1, \dots, \mathbf{s}_N | \mathbf{X}) = \frac{1}{Z} \exp(-E_{\theta}(\mathbf{s}_1, \dots, \mathbf{s}_N) | \mathbf{X})$$

- Trajectory Goodness + Collision.

$$\sum_i E_{\theta}(s_i | X) + \sum_{i \neq j} E(s_i, s_j)$$

$$E(s_i, s_j) = \gamma \text{ if } \underline{s_i} \text{ collides } \underline{s_j}$$



Planning with Social Reasoning

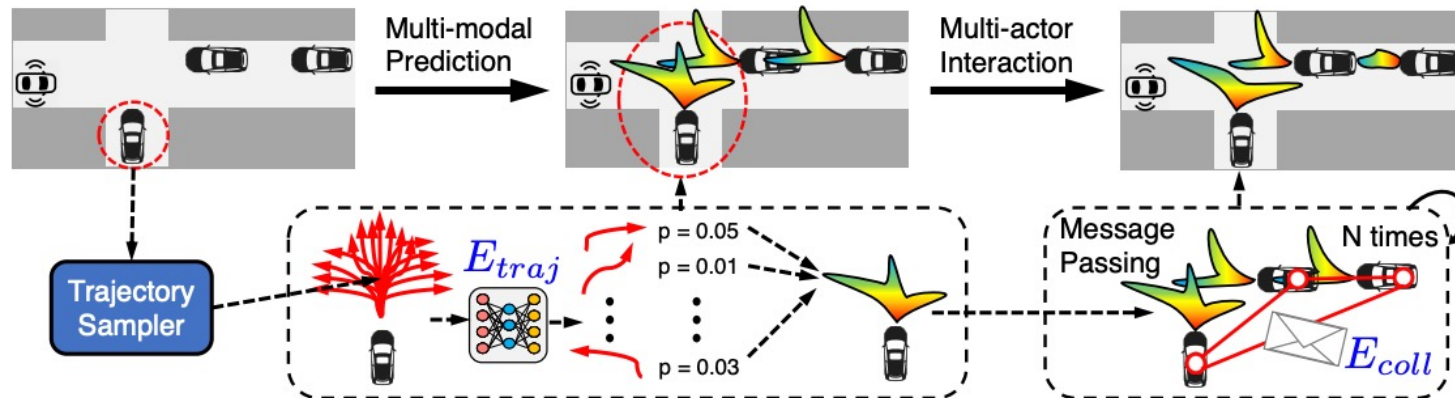
- Jointly reason the future trajectories of multiple agents as an energy-based graphical model.

$$p(\mathbf{s}_1, \dots, \mathbf{s}_N | \mathbf{X}) = \frac{1}{Z} \exp(-E_\theta(\mathbf{s}_1, \dots, \mathbf{s}_N) | \mathbf{X})$$

- Trajectory Goodness + Collision.
- Batch of trajectory samples.

$$\sum_i E_\theta(s_i | X) + \sum_{i \neq j} E(s_i, s_j)$$

$$E(\mathbf{s}_i, \mathbf{s}_j) = \gamma \text{ if } \mathbf{s}_i \text{ collides } \mathbf{s}_j$$



Planning with Social Reasoning

- Jointly reason the future trajectories of multiple agents as an energy-based graphical model.

$$p(\mathbf{s}_1, \dots, \mathbf{s}_N | \mathbf{X}) = \frac{1}{Z} \exp(-E_\theta(\mathbf{s}_1, \dots, \mathbf{s}_N) | \mathbf{X})$$

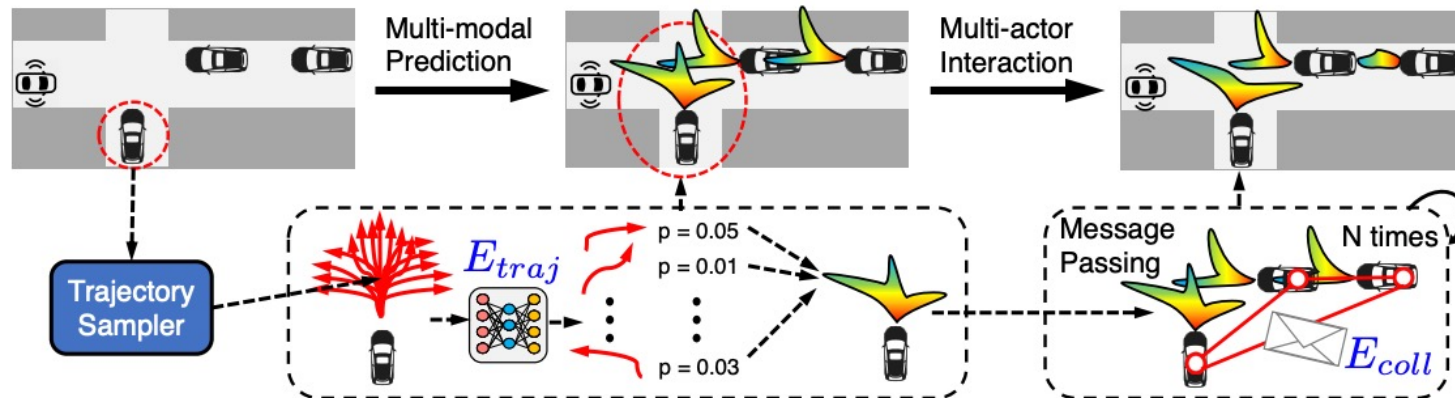
- Trajectory Goodness + Collision.

$$\sum_i E_\theta(s_i | X) + \sum_{i \neq j} E(s_i, s_j)$$

- Batch of trajectory samples.

$$E(\mathbf{s}_i, \mathbf{s}_j) = \gamma \text{ if } \mathbf{s}_i \text{ collides } \mathbf{s}_j$$

- Classification of groundtruth trajectory.



Summary: End-to-End Planning and Control

- Direct Policy Prediction
 - Condition perception features into the model
 - Use of diffusion models

Summary: End-to-End Planning and Control

- Direct Policy Prediction
 - Condition perception features into the model
 - Use of diffusion models
- Cost Learning (IRL) from Experts
 - Max-margin, max-entropy/EBM
 - Need negative samples →
 - Can be combined with efficient external samplers
 - Cost volume prediction: parametric + non-parametric

Summary: End-to-End Planning and Control

- Direct Policy Prediction
 - Condition perception features into the model
 - Use of diffusion models
- Cost Learning (IRL) from Experts
 - Max-margin, max-entropy/EBM
 - Need negative samples
 - Can be combined with efficient external samplers
 - Cost volume prediction: parametric + non-parametric
- Differentiable Planner
 - Backprop through local optimization, learnable cost
 - Can be memory efficient, implicit differentiation

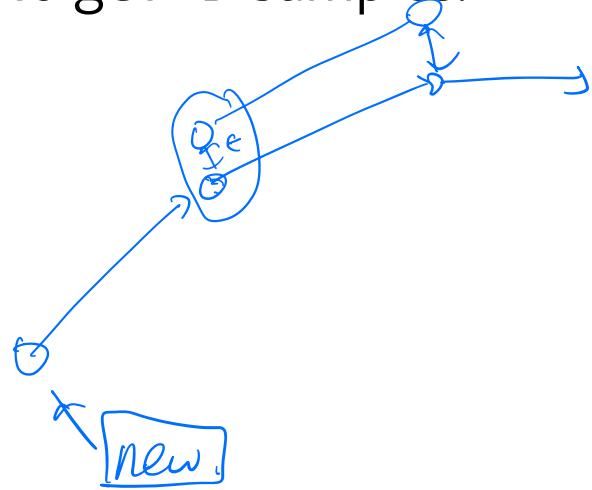
Module 5: Continual Learning

Why Continual Learning?

- The world is not a dataset that allows you to get IID samples.

Why Continual Learning?

- The world is not a dataset that allows you to get IID samples.
- The world keeps changing and evolving.

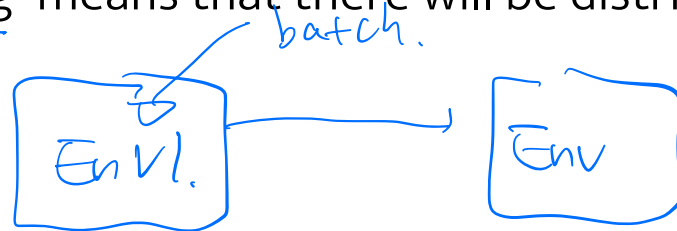


Why Continual Learning?

- The world is not a dataset that allows you to get IID samples.
- The world keeps changing and evolving.
- Online vs. Continual
 - Online means that samples arrive in a streaming / temporal partial order, but they may still come from a static distribution.

$$\theta_t = f(x_t, \theta_{t-1}) \quad x_{1:T} \sim \mathcal{X}$$

- Example: Online reinforcement learning, trajectory roll out is online, but the environment is the same.
- Continual learning means that there will be distribution shift.



Online ⊕ Continual.

What is Continual Learning?

- Distribution shift: Forgetting
 - Learning on A and then B, results in worse performance on A.

*retain knowledge.
What to retain?*

What is Continual Learning?

- Distribution shift: Forgetting
 - Learning on A and then B, results in worse performance on A.
- Multi-task learning: Forward transfer $(A) + (B)$
 - Learning Task A + B results in better learning in Task C compared to learning C alone.
 - Leverage the similarity between tasks.

What is Continual Learning?

- Distribution shift: Forgetting
 - Learning on A and then B, results in worse performance on A.
- Multi-task learning: Forward transfer
 - Learning Task A + B results in better learning in Task C compared to learning C alone.
 - Leverage the similarity between tasks.
- Compositionality
 - Learning A and B first, and then learning tasks with composed A+B.

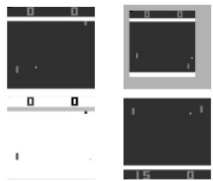
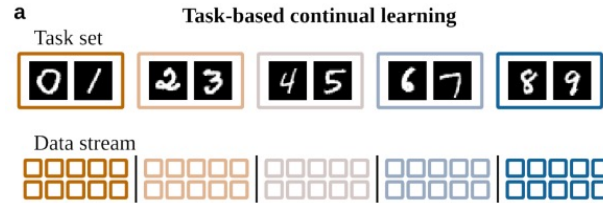
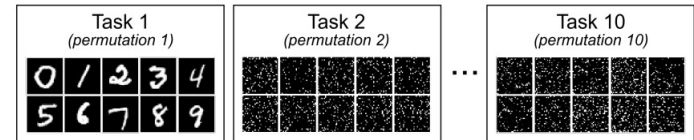
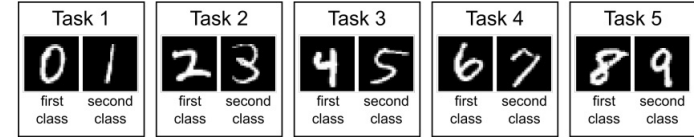
What is Continual Learning?

- Distribution shift: Forgetting
 - Learning on A and then B, results in worse performance on A.
- Multi-task learning: Forward transfer
 - Learning Task A + B results in better learning in Task C compared to learning C alone.
 - Leverage the similarity between tasks.
- Compositionality
 - Learning A and B first, and then learning tasks with composed A+B.
- Incremental/curriculum Learning
 - Learning A->B->C is easier than at random order 😊

Continual Learning

toy dataset

- Learning a sequence of tasks without looking back.



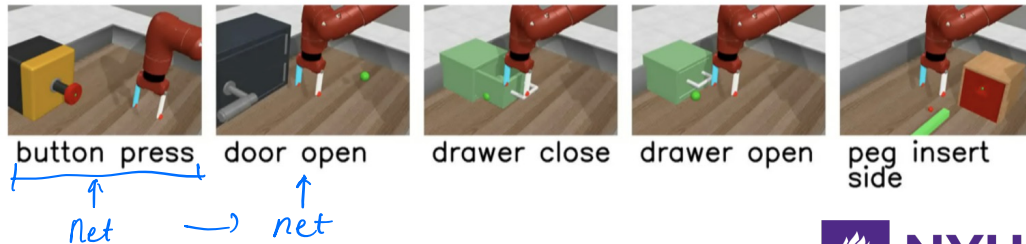
(a) Pong variants



(b) Labyrinth games

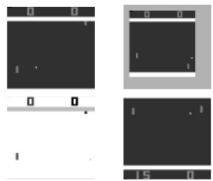
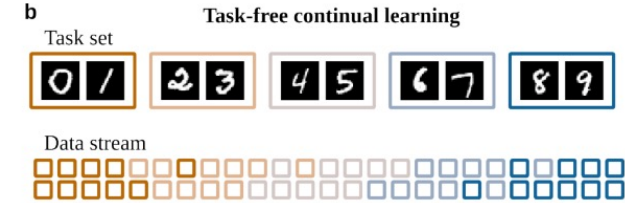
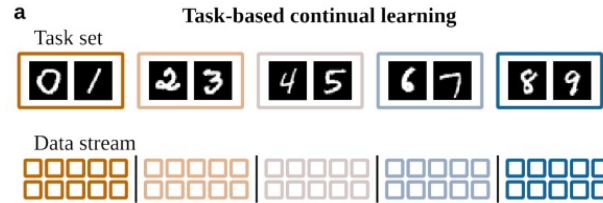
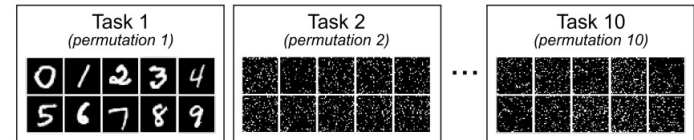
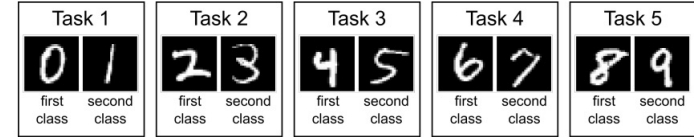


(c) Atari games



Continual Learning

- Learning a sequence of tasks without looking back.
- Goal is to do well on all of the tasks at the end.



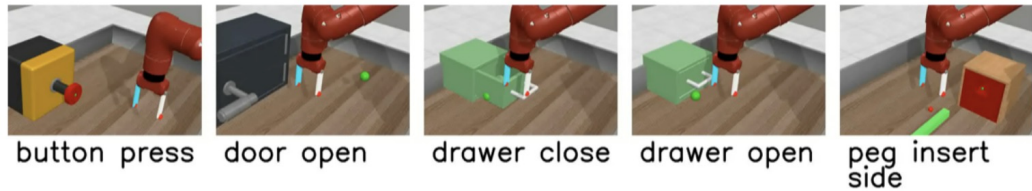
(a) Pong variants



(b) Labyrinth games

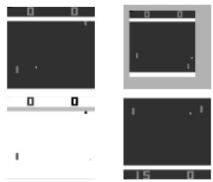
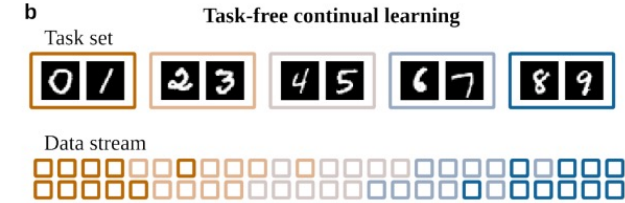
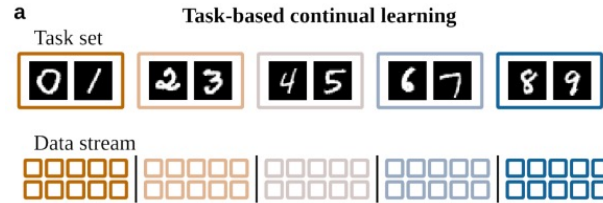
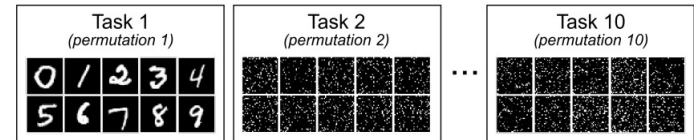
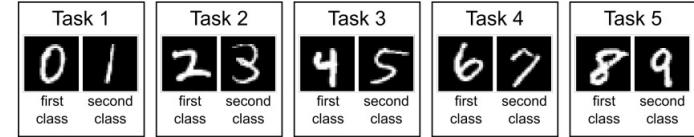


(c) Atari games

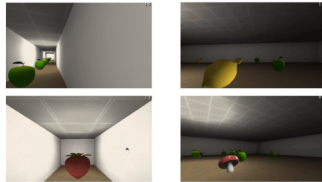


Continual Learning

- Learning a sequence of tasks without looking back.
- Goal is to do well on all of the tasks at the end.
- Task boundary ?



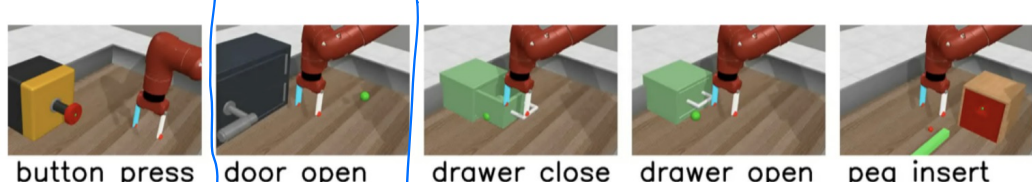
(a) Pong variants



(b) Labyrinth games



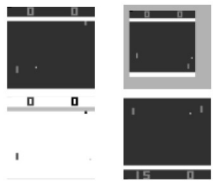
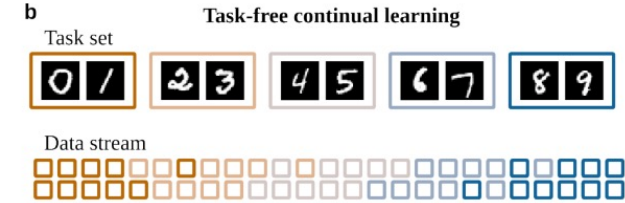
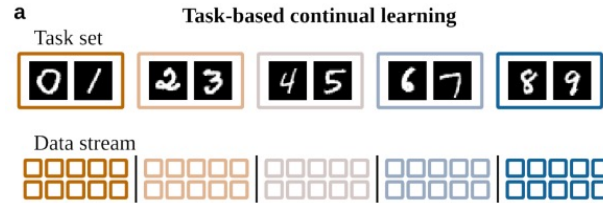
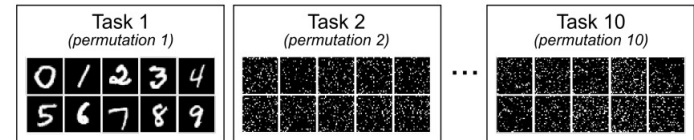
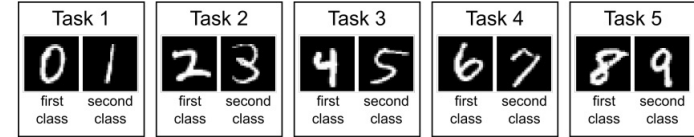
(c) Atari games



button press door open drawer close drawer open peg insert side

Continual Learning

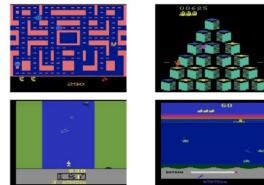
- Learning a sequence of tasks without looking back.
- Goal is to do well on all of the tasks at the end.
- Task boundary
- Memory constraints



(a) Pong variants



(b) Labyrinth games



(c) Atari games



button press



door open



drawer close

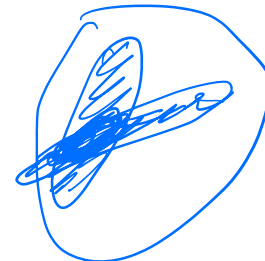


drawer open



peg insert side

Parameter Regularization



- Over-completeness Assumption. A multitude of models can reach equivalent performance.

$$\mathcal{S}_A = \{\theta \mid \ell_A(\theta) < \epsilon\}$$

$$\mathcal{S}_A \cap \mathcal{S}_B \neq \emptyset$$

Parameter Regularization

- Over-completeness Assumption. A multitude of models can reach equivalent performance.
- What is left is to efficiently find the intersection between A and B.

$$p(\theta \mid \mathcal{D}_A) = \mathcal{N}(\theta; \theta^*, \Sigma)$$

$$\mathcal{S}_A = \{\theta \mid \ell_A(\theta) < \epsilon\}$$

$$\mathcal{S}_A \cap \mathcal{S}_B \neq \emptyset$$

Parameter Regularization

- Over-completeness Assumption. A multitude of models can reach equivalent performance.

$$\mathcal{S}_A = \{\theta \mid \ell_A(\theta) < \epsilon\}$$

$$\mathcal{S}_A \cap \mathcal{S}_B \neq \emptyset$$

- What is left is to efficiently find the intersection between A and B.

$$p(\theta \mid \mathcal{D}_A) = \mathcal{N}(\theta; \theta^*, \Sigma)$$

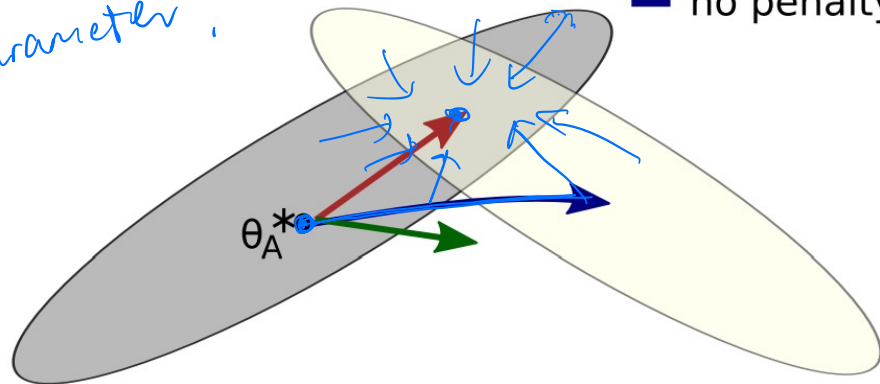
- Elastic Weight Consolidation (EWC):

$$\mathcal{L}(\theta) = \mathcal{L}_B(\theta) + \sum_i \frac{\lambda}{2} F_i (\theta_i - \theta_{A,i}^*)^2$$

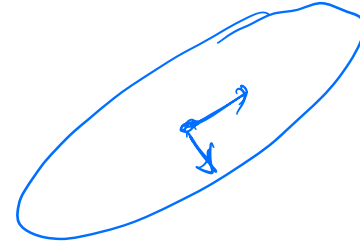
old parameters

strength per direction.

- Low error for task B
- Low error for task A
- EWC
- L₂
- no penalty

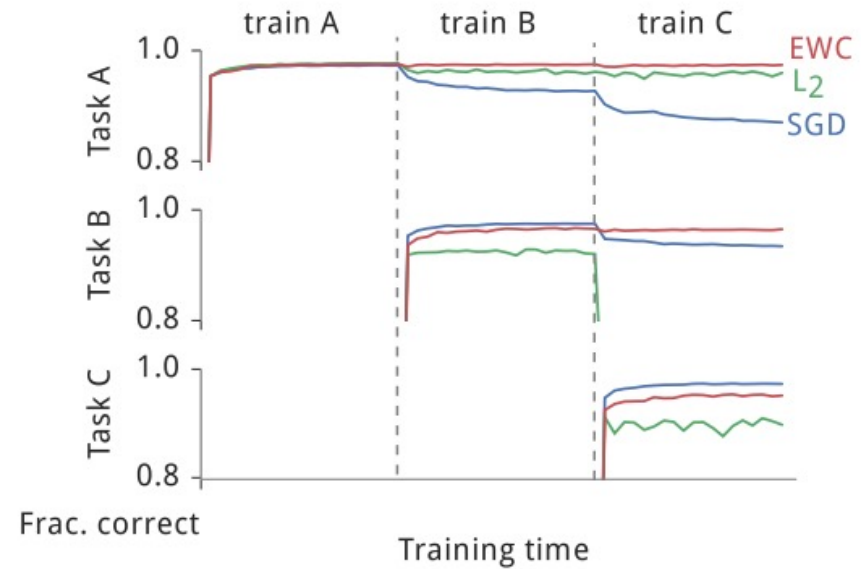


Computing Fisher



- At the end of each epoch, compute the gradient squared:

$$F_i = \left(\frac{d\mathcal{L}}{d\theta_i} \right)^2$$

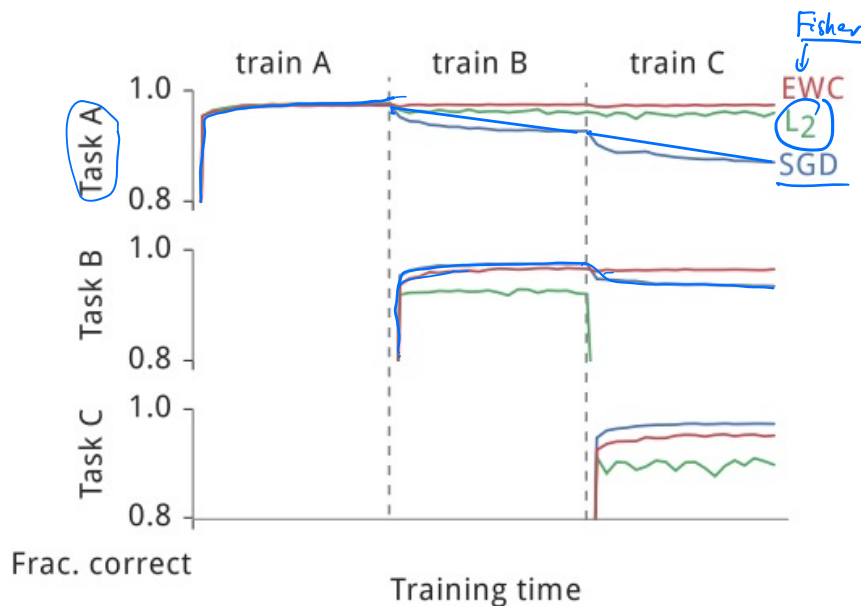


Computing Fisher

- At the end of each epoch, compute the gradient squared:

$$F_i = \left(\frac{d\mathcal{L}}{d\theta_i} \right)^2$$

- Measures the sensitivity on each parameter dimension.

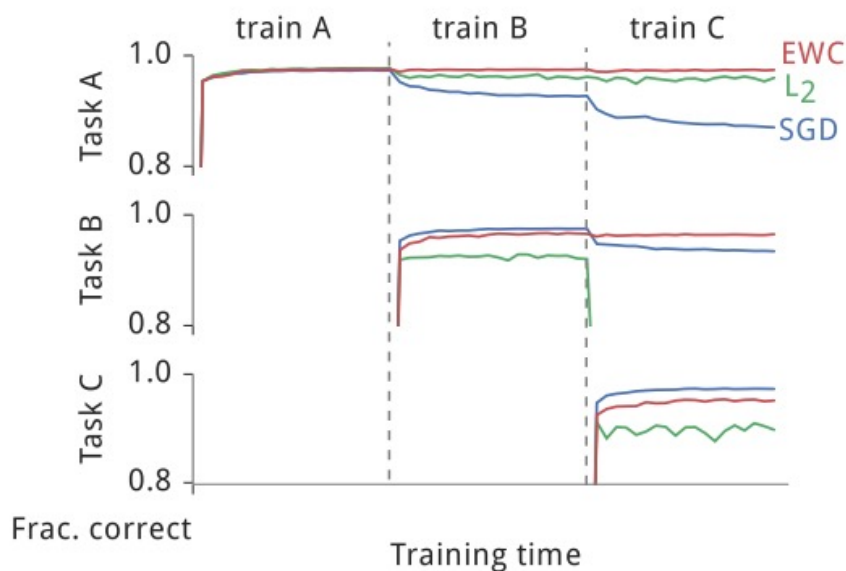


Computing Fisher

- At the end of each epoch, compute the gradient squared:

$$F_i = \left(\frac{d\mathcal{L}}{d\theta_i} \right)^2$$

- Measures the sensitivity on each parameter dimension.
- You can also accumulate an online estimate.



Variational Continual Learning (VCL)

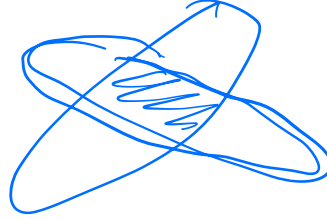
- Bayesian formulation:

$$p(\boldsymbol{\theta} \mid \mathcal{D}_{1:T}) \propto p(\boldsymbol{\theta}) \prod_{t=1}^T p(\mathcal{D}_t \mid \boldsymbol{\theta}) \propto p(\boldsymbol{\theta} \mid \mathcal{D}_{1:T-1}) p(\mathcal{D}_T \mid \boldsymbol{\theta}).$$

new posterior.

previous posterior.

final task.



Variational Continual Learning (VCL)

- Bayesian formulation:

$$p(\boldsymbol{\theta} \mid \mathcal{D}_{1:T}) \propto p(\boldsymbol{\theta}) \prod_{t=1}^T p(\mathcal{D}_t \mid \boldsymbol{\theta}) \propto p(\boldsymbol{\theta} \mid \mathcal{D}_{1:T-1}) p(\mathcal{D}_T \mid \boldsymbol{\theta}).$$

- Variational approach:

$$q_t(\boldsymbol{\theta}) = \operatorname{argmin}_{q \in \mathcal{Q}} \operatorname{KL} \left(q(\boldsymbol{\theta}) \parallel \frac{1}{Z_t} q_{t-1}(\boldsymbol{\theta}) p(\mathcal{D}_t \mid \boldsymbol{\theta}) \right).$$

Gaussian F_t .

Gaussian

F_{t-1}

Variational Continual Learning (VCL)

- Bayesian formulation:

$$p(\boldsymbol{\theta} \mid \mathcal{D}_{1:T}) \propto p(\boldsymbol{\theta}) \prod_{t=1}^T p(\mathcal{D}_t \mid \boldsymbol{\theta}) \propto p(\boldsymbol{\theta} \mid \mathcal{D}_{1:T-1})p(\mathcal{D}_T \mid \boldsymbol{\theta}).$$

- Variational approach:

$$q_t(\boldsymbol{\theta}) = \operatorname{argmin}_{q \in \mathcal{Q}} \operatorname{KL} \left(q(\boldsymbol{\theta}) \parallel \frac{1}{Z_t} q_{t-1}(\boldsymbol{\theta}) p(\mathcal{D}_t \mid \boldsymbol{\theta}) \right).$$

- Loss: $\mathcal{L}(q_t(\boldsymbol{\theta})) = \mathbb{E}_{\boldsymbol{\theta} \sim q_t(\boldsymbol{\theta})} \left[\underbrace{-\log p(\mathbf{y} \mid \mathbf{x}, \boldsymbol{\theta})}_{\text{data loss}} \right] + \underbrace{\operatorname{KL}(q_t(\boldsymbol{\theta}) \parallel q_{t-1}(\boldsymbol{\theta}))}_{\text{KL divergence}} \right).$

Variational Continual Learning (VCL)

- Bayesian formulation:

$$p(\boldsymbol{\theta} \mid \mathcal{D}_{1:T}) \propto p(\boldsymbol{\theta}) \prod_{t=1}^T p(\mathcal{D}_t \mid \boldsymbol{\theta}) \propto p(\boldsymbol{\theta} \mid \mathcal{D}_{1:T-1})p(\mathcal{D}_T \mid \boldsymbol{\theta}).$$

- Variational approach:

$$q_t(\boldsymbol{\theta}) = \operatorname{argmin}_{q \in \mathcal{Q}} \operatorname{KL} \left(q(\boldsymbol{\theta}) \parallel \frac{1}{Z_t} q_{t-1}(\boldsymbol{\theta}) p(\mathcal{D}_t \mid \boldsymbol{\theta}) \right).$$

- Loss: $\mathcal{L}(q_t(\boldsymbol{\theta})) = \mathbb{E}_{\boldsymbol{\theta} \sim q_t(\boldsymbol{\theta})} [-\log p(\mathbf{y} \mid \mathbf{x}, \boldsymbol{\theta})] + \operatorname{KL}(q_t(\boldsymbol{\theta}) \parallel q_{t-1}(\boldsymbol{\theta}))$.
 $q_t(\boldsymbol{\theta}) = \prod_{d=1}^D \mathcal{N}(\theta_{t,d}; \mu_{t,d}, \sigma_{t,d}^2)$.

Variational Continual Learning (VCL)

- Bayesian formulation:

$$p(\boldsymbol{\theta} \mid \mathcal{D}_{1:T}) \propto p(\boldsymbol{\theta}) \prod_{t=1}^T p(\mathcal{D}_t \mid \boldsymbol{\theta}) \propto p(\boldsymbol{\theta} \mid \mathcal{D}_{1:T-1})p(\mathcal{D}_T \mid \boldsymbol{\theta}).$$

- Variational approach:

$$q_t(\boldsymbol{\theta}) = \operatorname{argmin}_{q \in \mathcal{Q}} \operatorname{KL} \left(q(\boldsymbol{\theta}) \parallel \frac{1}{Z_t} q_{t-1}(\boldsymbol{\theta}) p(\mathcal{D}_t \mid \boldsymbol{\theta}) \right).$$

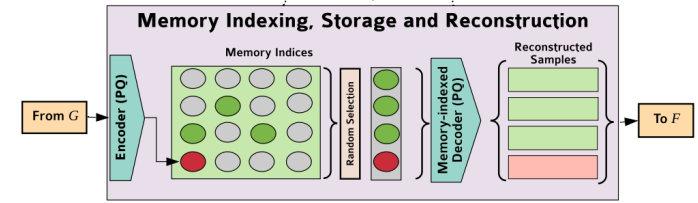
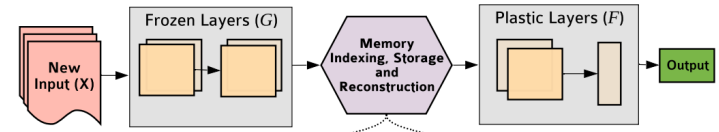
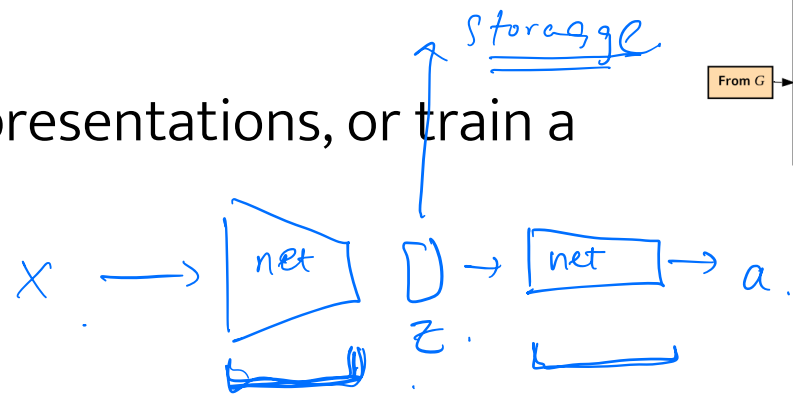
- Loss: $\mathcal{L}(q_t(\boldsymbol{\theta})) = \mathbb{E}_{\boldsymbol{\theta} \sim q_t(\boldsymbol{\theta})} [-\log p(\mathbf{y} \mid \mathbf{x}, \boldsymbol{\theta})] + \operatorname{KL}(q_t(\boldsymbol{\theta}) \parallel q_{t-1}(\boldsymbol{\theta}))$.
 $q_t(\boldsymbol{\theta}) = \prod_{d=1}^D \mathcal{N}(\theta_{t,d}; \mu_{t,d}, \sigma_{t,d}^2)$.

- Compare to EWC: Maintains uncertainty throughout training.

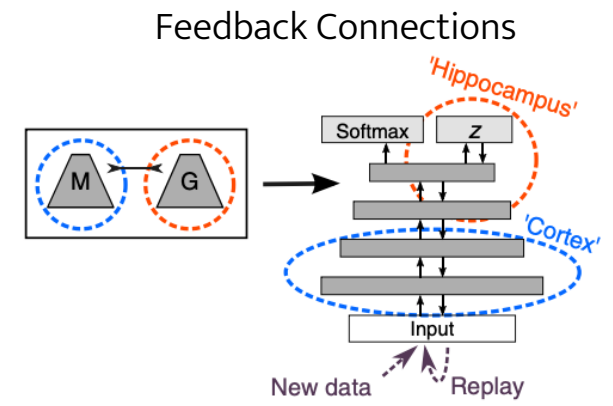
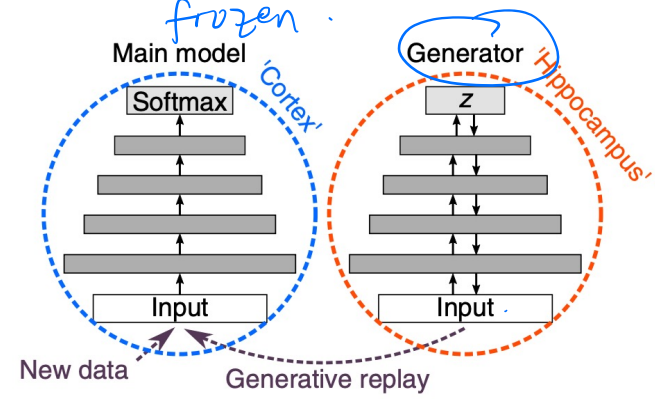
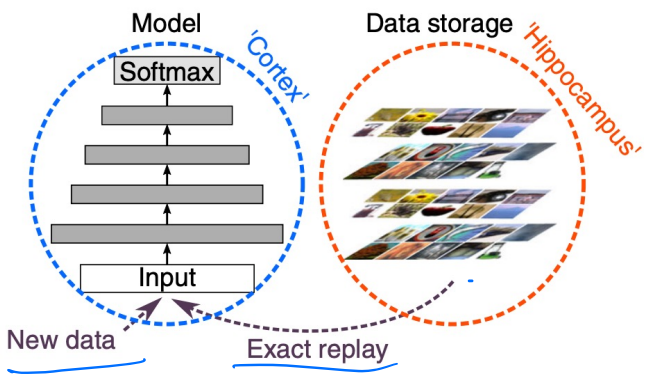
old input

Replay

- Store raw data, representations, or train a generative model



Storage rep.

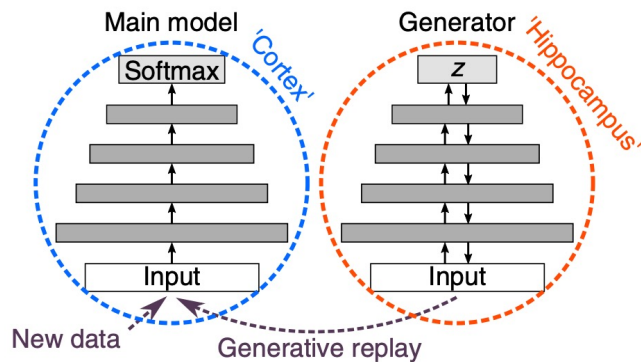
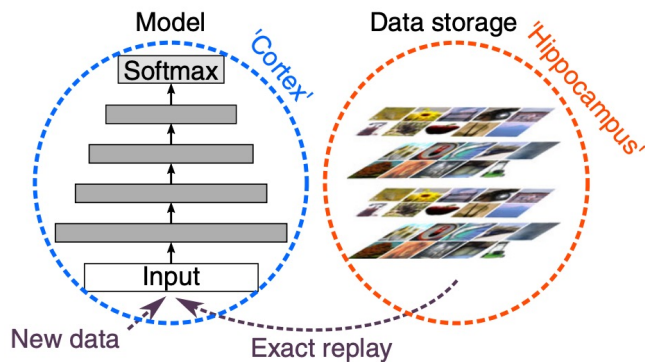
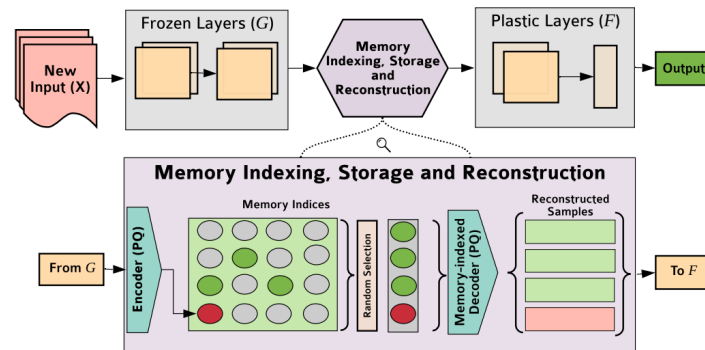


Rebuffi et al. iCaRL: Incremental Classifier and Representation Learning. CVPR 2017.
 van de Ven et al. Brain-inspired replay for continual learning with artificial neural networks. Nature communications 2020.
 Hayes et al. REMIND Your Neural Network to Prevent Catastrophic Forgetting. ECCV 2020.

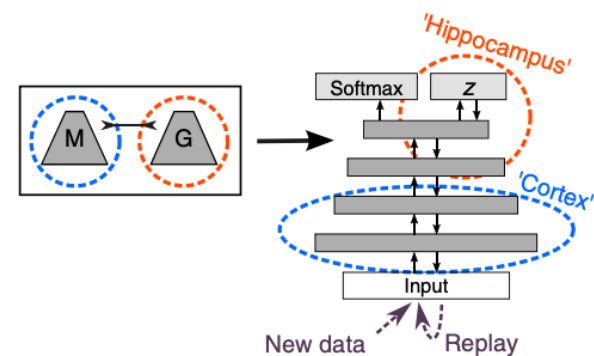
Replay

- Store raw data, representations, or train a generative model
- Coreset selection $\mu \leftarrow \frac{1}{n} \sum_{x \in X} \varphi(x)$

$$p_k \leftarrow \operatorname{argmin}_{x \in X} \left\| \mu - \frac{1}{k} \left[\varphi(x) + \sum_{j=1}^{k-1} \varphi(p_j) \right] \right\|.$$



Feedback Connections



Knowledge Distillation

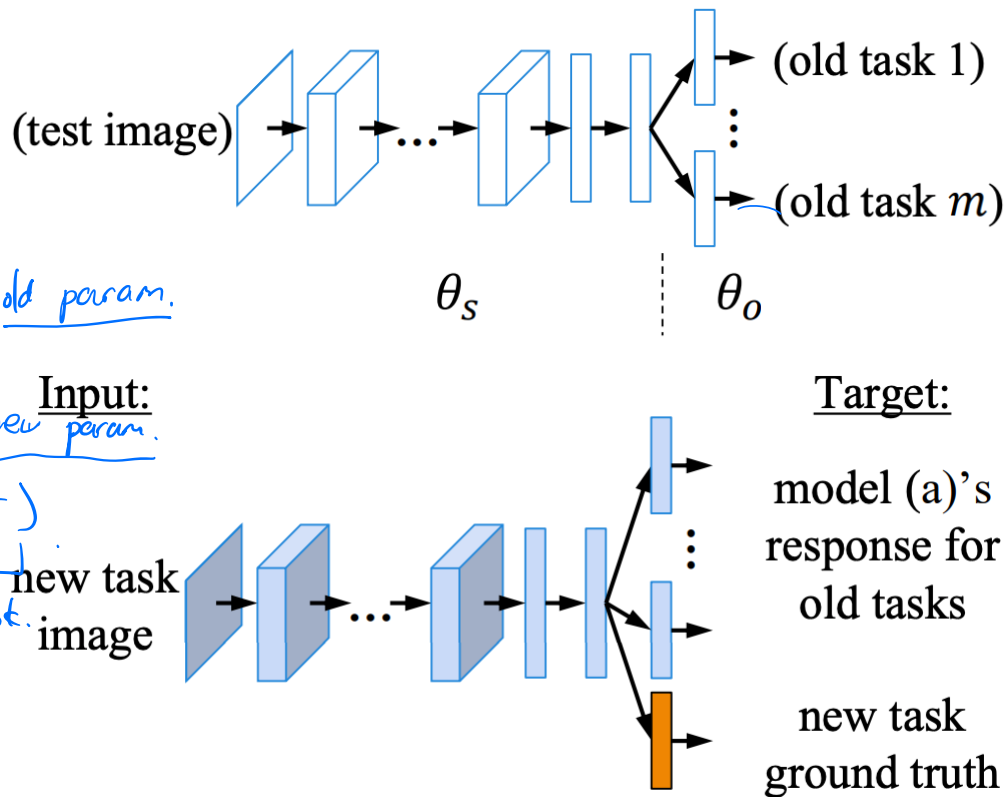
- Instead of saving the data points, we can also save the previous model checkpoint.

old output. $y_o = f(x_n; \theta_o)$. *new input* *old param.*

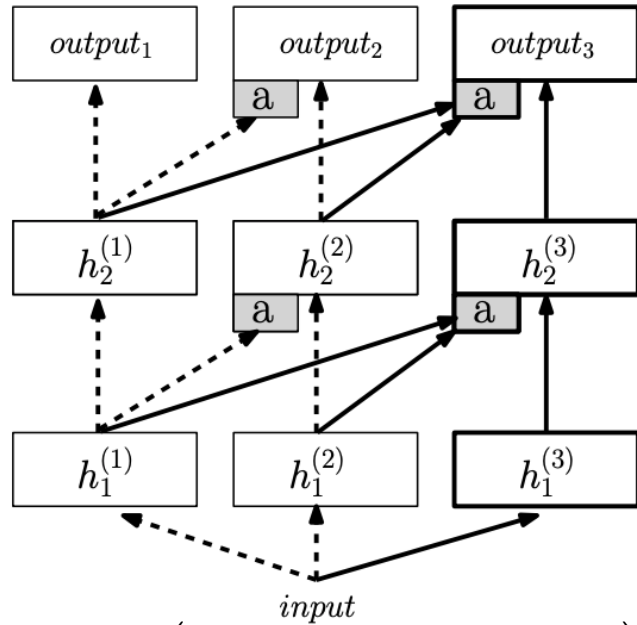
new output $\hat{y}_o = f(x_n; \theta_n)$. *new input* *new param.* **Input:**

$\mathcal{L}(y_o, \hat{y}_o) + \mathcal{L}(\hat{y}_o, y^*)$
prevent forgetting *learn new task.*

- Use new data points and old weights to “distill”

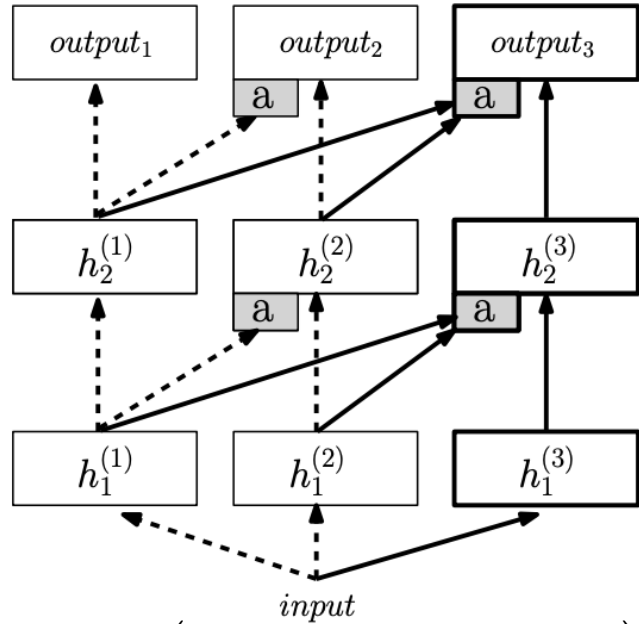


Architecture Expansion

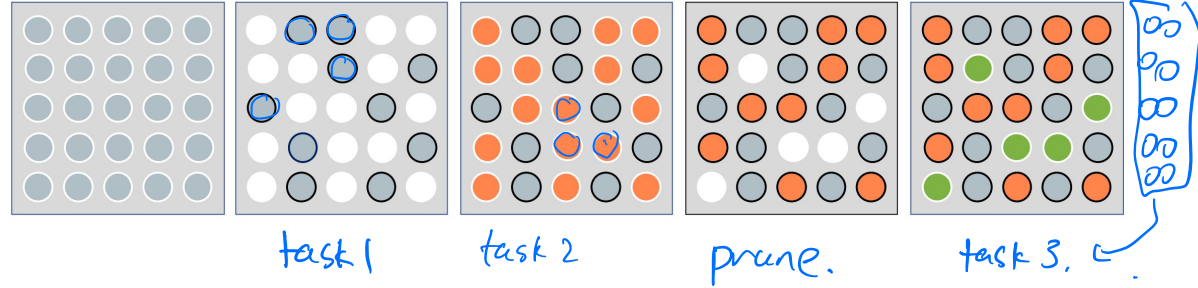


$$h_i^{(k)} = f \left(W_i^{(k)} h_{i-1}^{(k)} + \sum_{j < k} U_i^{(k:j)} h_{i-1}^{(j)} \right).$$

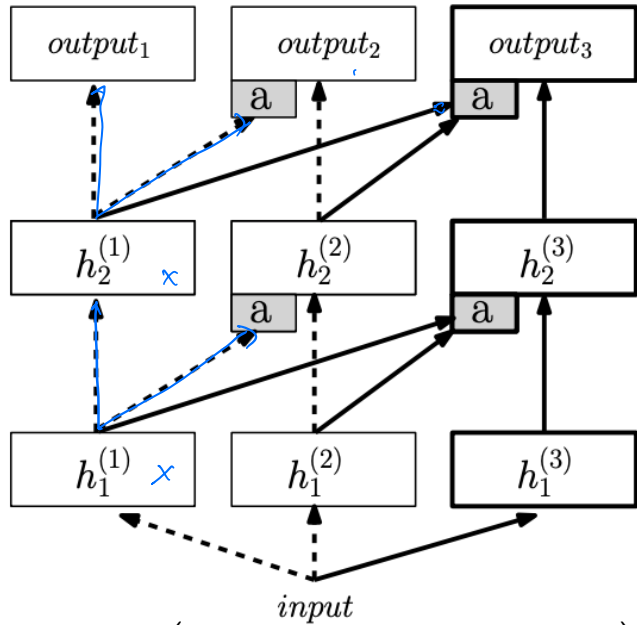
Architecture Expansion



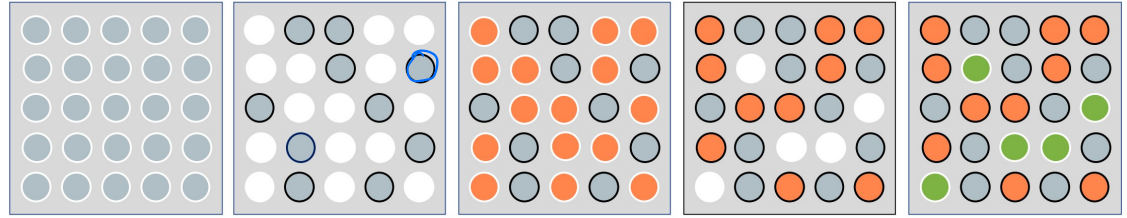
$$h_i^{(k)} = f \left(W_i^{(k)} h_{i-1}^{(k)} + \sum_{j < k} U_i^{(k:j)} h_{i-1}^{(j)} \right).$$



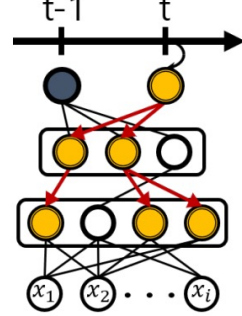
Architecture Expansion



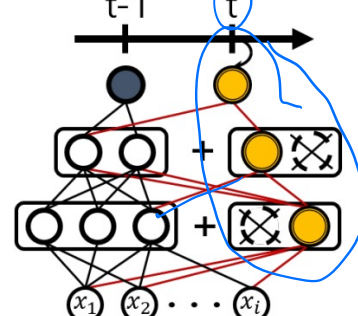
$$h_i^{(k)} = f \left(W_i^{(k)} h_{i-1}^{(k)} + \sum_{j < k} U_i^{(k:j)} h_{i-1}^{(j)} \right).$$



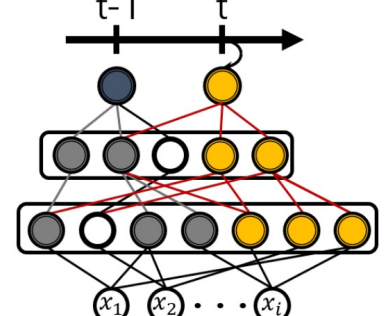
Selective Training



Dynamic Expansion



Network Split



Selective Training

$$\mathcal{L}(\mathbf{W}_L^t, \mathbf{W}_{1:L-1}^{t-1}, \mathcal{D}_t) + \mu \|\mathbf{W}_L^t\|_1$$

Dynamic Expansion

$$\mathcal{L}(\mathbf{W}_L^N, \mathbf{W}_L^{t-1}, \mathcal{D}_t) + \mu \|\mathbf{W}_L^N\|_1 + \gamma \sum_g \|\mathbf{W}_{l,g}^N\|_2$$

Group sparsity

Rusu et al. Progressive Neural Networks. NIPS 2016 Deep Learning Symposium.

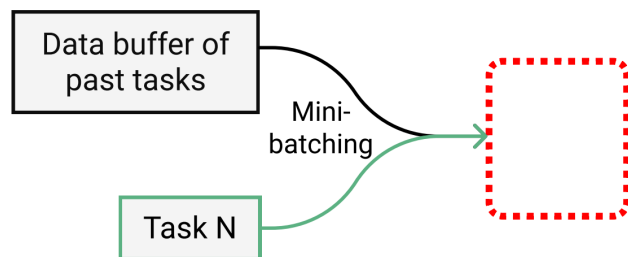
PackNet: Adding Multiple Tasks to a Single Network by Iterative Pruning. CVPR 2018.

Yoon et al. Lifelong Learning with Dynamically Expandable Networks. ICLR 2018.


Adapting Pretrained Models

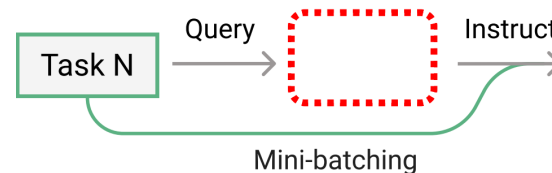
- Pretrained models have general knowledge that can be adapted to a continual stream of tasks.

Rehearsal-based methods:
Fine-tuning



Our method:
Prompt selection + tuning

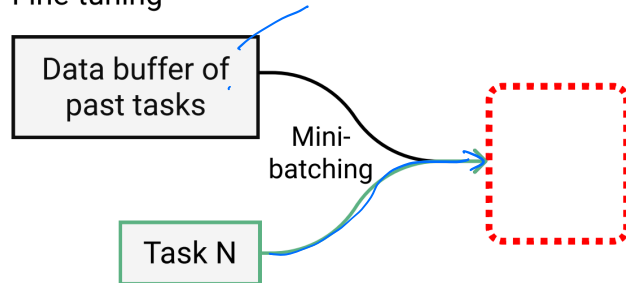
 : Trainable




Adapting Pretrained Models

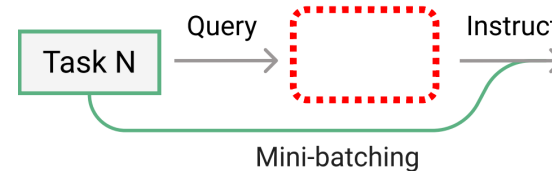
- Pretrained models have general knowledge that can be adapted to a continual stream of tasks.
- Learn adaptation parameters for each task and store these as “task embeddings.”

Rehearsal-based methods:
Fine-tuning



Our method:
Prompt selection + tuning

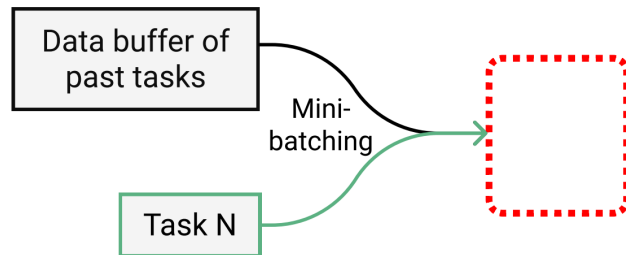
 : Trainable




Adapting Pretrained Models

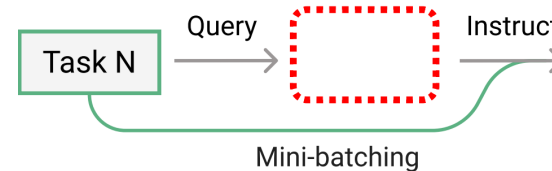
- Pretrained models have general knowledge that can be adapted to a continual stream of tasks.
- Learn adaptation parameters for each task and store these as “task embeddings.”
- Main model is frozen.

Rehearsal-based methods:
Fine-tuning



Our method:
Prompt selection + tuning

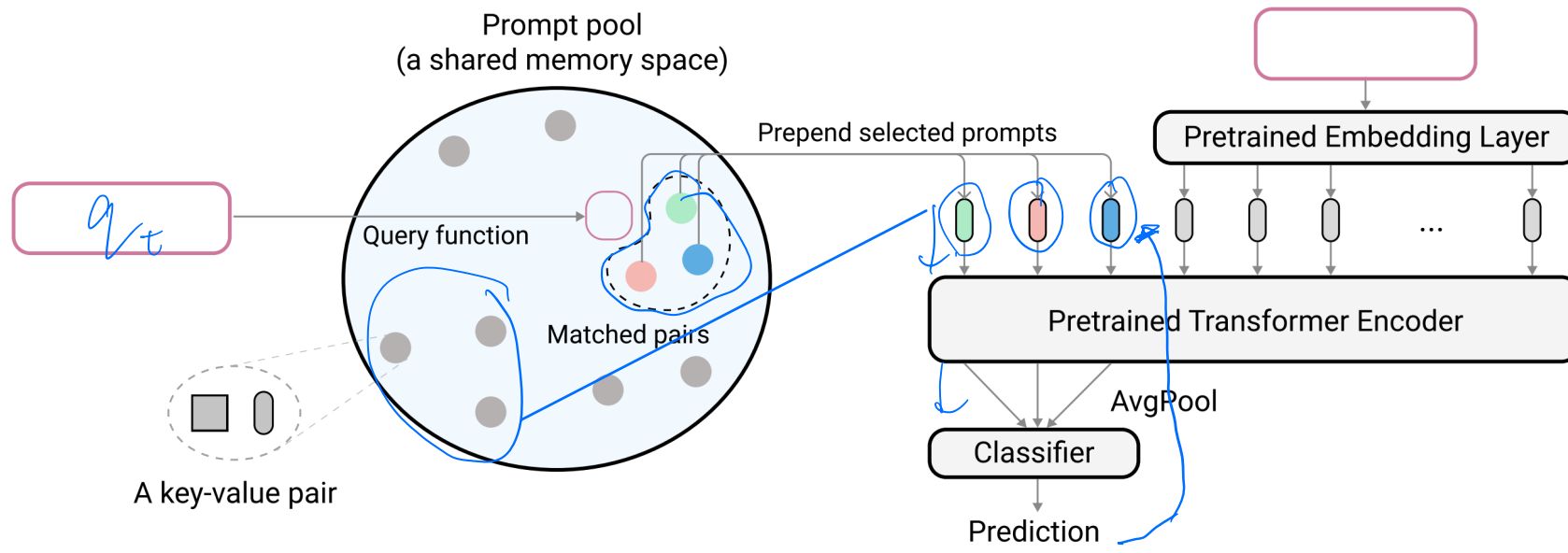
 : Trainable



Learning to Prompt

Prompt pool (slot memory)

$$\{(k_1, p_1), \dots, (k_M, p_M)\}$$



Learning to Prompt

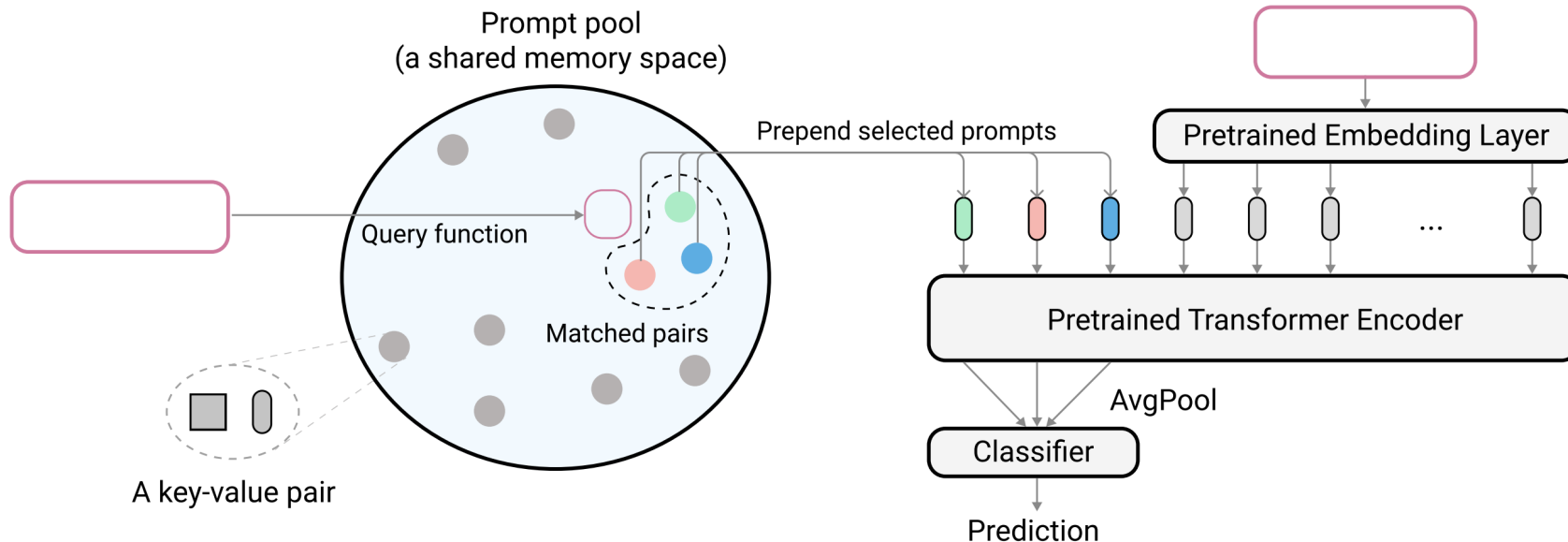
Prompt pool (slot memory)

$\{(k_1, p_1), \dots, (k_M, p_M)\}$

$$K_s = \operatorname{argmin}_S \sum_{i \in S} \gamma(q(x), k_i)$$

task. input

key each prompt.



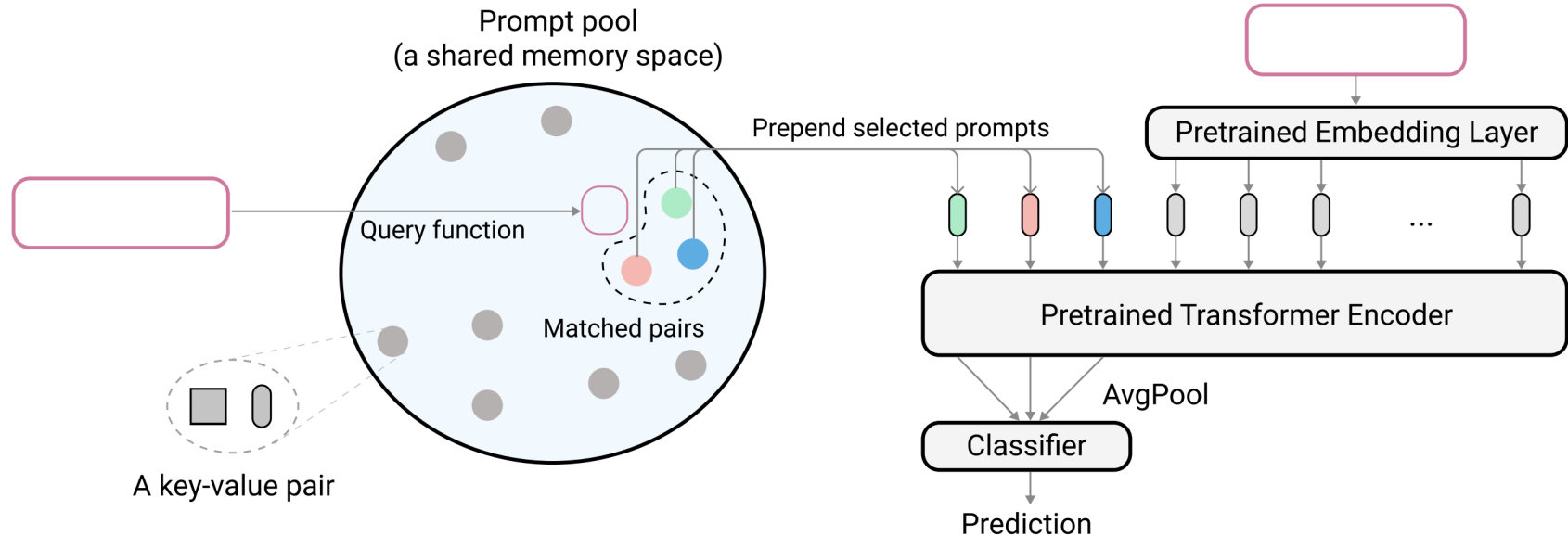
Learning to Prompt

Prompt pool (slot memory)

$\{(k_1, p_1), \dots, (k_M, p_M)\}$

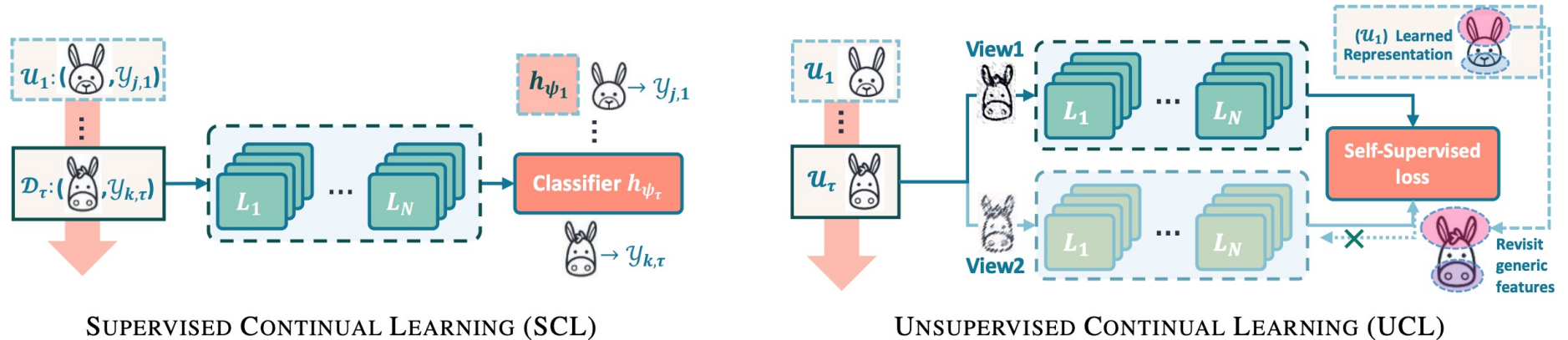
$$K_s = \operatorname{argmin}_{\mathcal{S}} \sum_{i \in \mathcal{S}} \gamma(q(x), k_i)$$

$$\min_{P, K, \phi} \mathcal{L}(g_{\phi}(x), y) + \lambda \sum_{i \in K_s} \gamma(q(x), k_i)$$



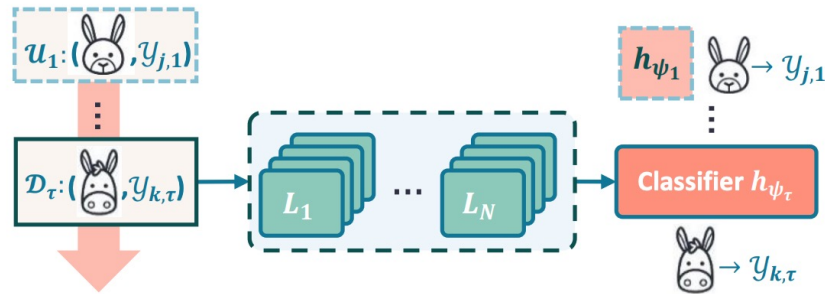
Continual Self-Supervised Learning

- Learning from a stream of unlabeled inputs.

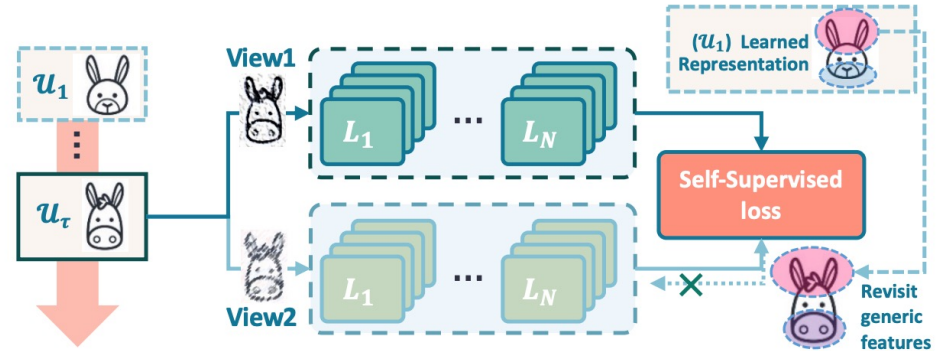


Continual Self-Supervised Learning

- Learning from a stream of unlabeled inputs.
- Bring SSL to the dynamic world.



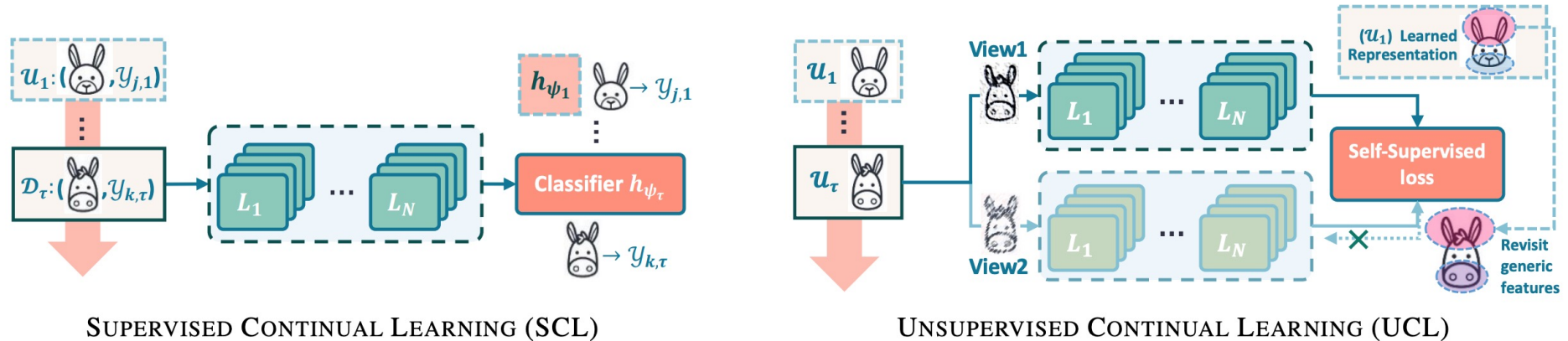
SUPERVISED CONTINUAL LEARNING (SCL)



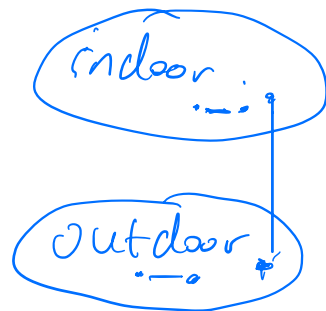
UNSUPERVISED CONTINUAL LEARNING (UCL)

Continual Self-Supervised Learning

- Learning from a stream of unlabeled inputs.
- Bring SSL to the dynamic world.
- SSL can still suffer from distributional shifts.

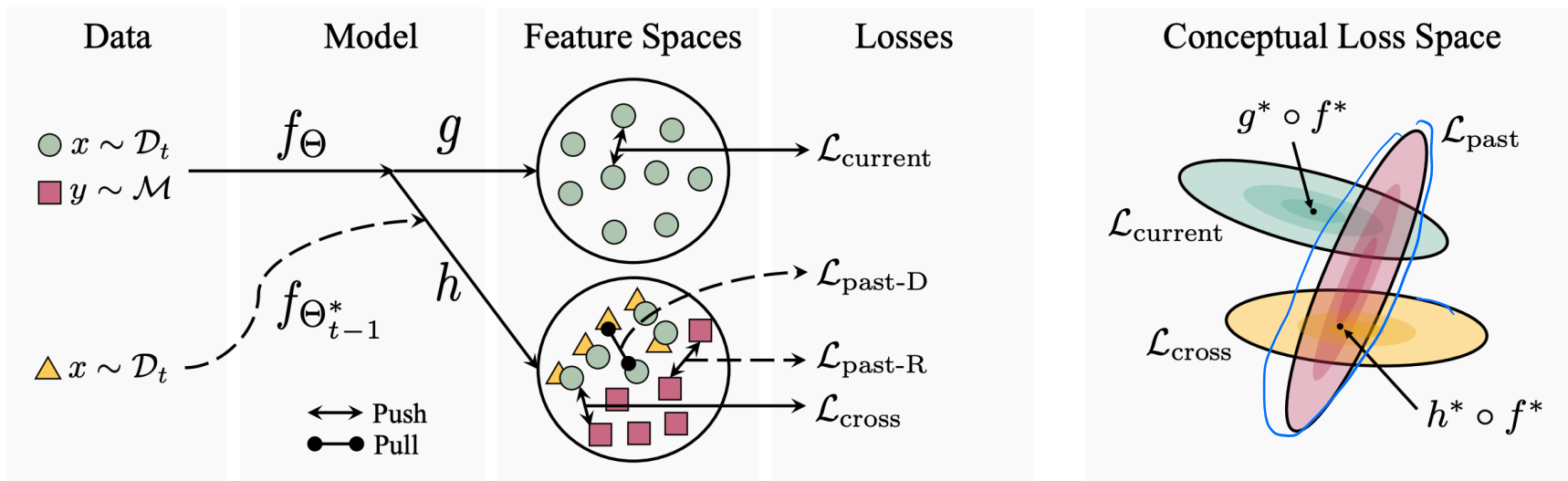


Continual Self-Supervised Learning



- Integrating learning objectives of both past and present.

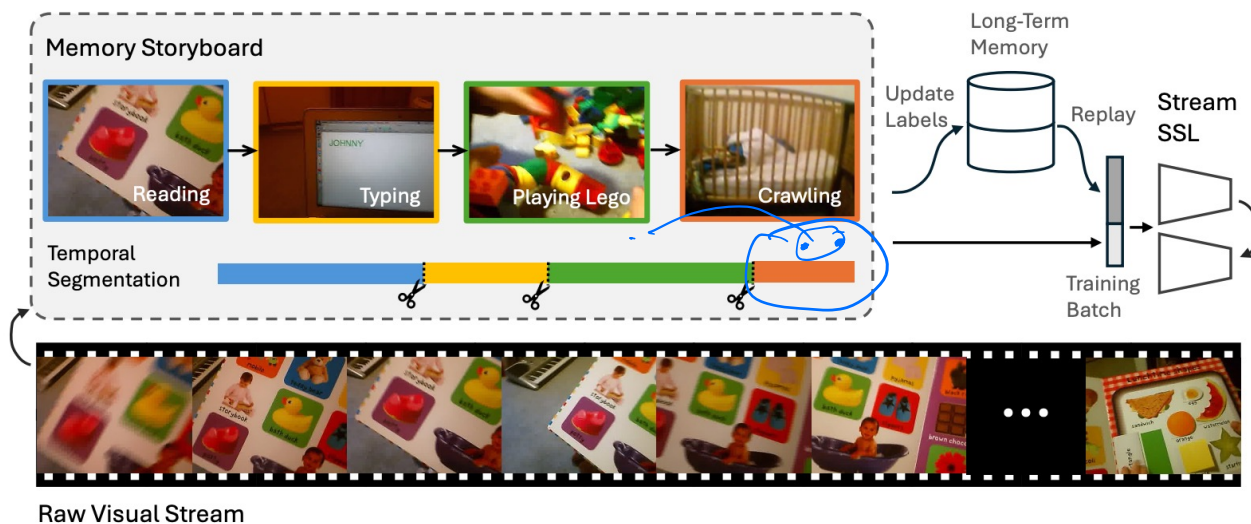
$$\mathcal{L}(X; g \circ f_t) + \mathcal{L}(X, X; h \circ f_{t-1}, h \circ f_t) + \mathcal{L}(Y; h \circ f_t) + \mathcal{L}(X, Y; h \circ f_t)$$



Continual Learning from Video Stream

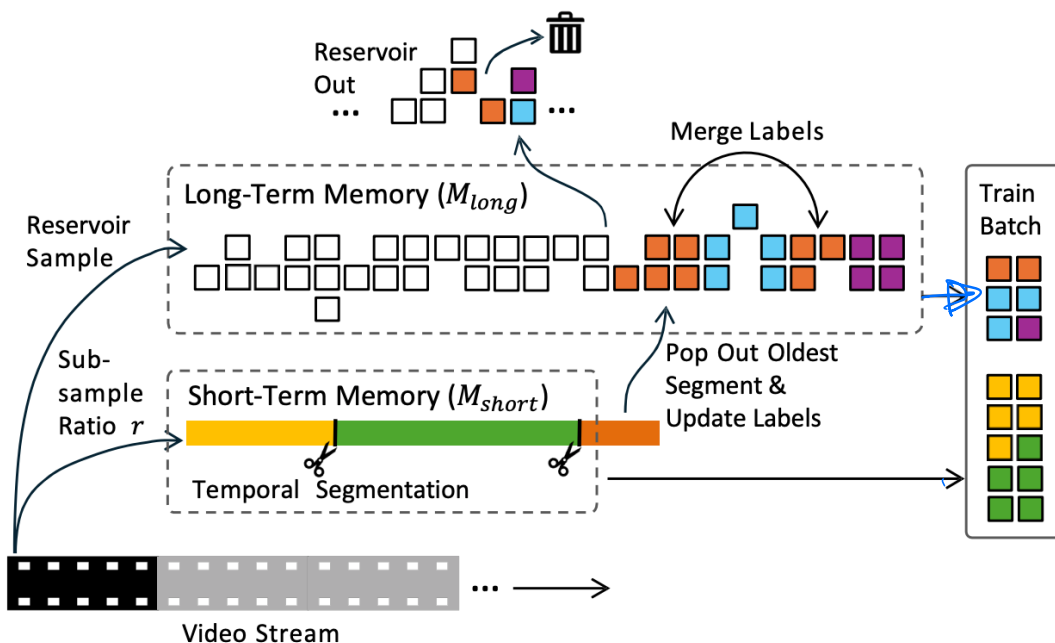
- Watching the full video once in a lifetime.
- Event segmentation to define “boundary.”

task boundary.



Multiple Timescale of Memory in Video Learning

- Micro Chunk: Atomic unit.
- **Semantic Segment:** Most relevant chunk for video-text alignment.
- **Event Segment:** High-level segment for event episodic memory.



Continual Learning and Memory

→ raw sample representation.

- Fragility of feedforward gradient descent of the entire networks

Continual Learning and Memory

- Fragility of feedforward gradient descent of the entire networks
- If we have representations ready, continual learning is just memorizing a sequence of new tasks.

Continual Learning and Memory

- Fragility of feedforward gradient descent of the entire networks
- If we have representations ready, continual learning is just memorizing a sequence of new tasks.
- In prompting approaches:
 - prompt pool = memory
 - pretrained network = representations

Continual Learning and Memory

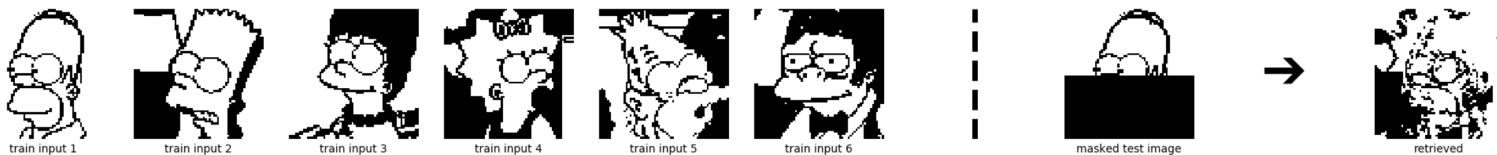
- Fragility of feedforward gradient descent of the entire networks
- If we have representations ready, continual learning is just memorizing a sequence of new tasks.
- In prompting approaches:
 - prompt pool = memory
 - pretrained network = representations
- But what if representations also need to be built sequentially?

Continual Learning and Memory

- Fragility of feedforward gradient descent of the entire networks
- If we have representations ready, continual learning is just memorizing a sequence of new tasks.
- In prompting approaches:
 - prompt pool = memory
 - pretrained network = representations
- But what if representations also need to be built sequentially?
- It's also plausible that representations are just “deeper memory.”

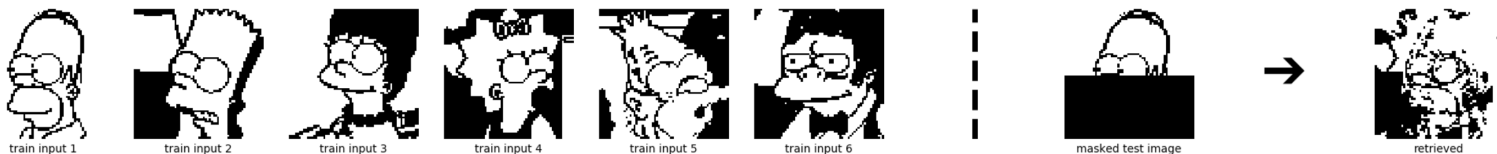
Associative Memory

- Memory aims to store content for easy retrieval



Associative Memory

- Memory aims to store content for easy retrieval
 - Associative memories (Hopfield Networks) can be viewed as energy-based models



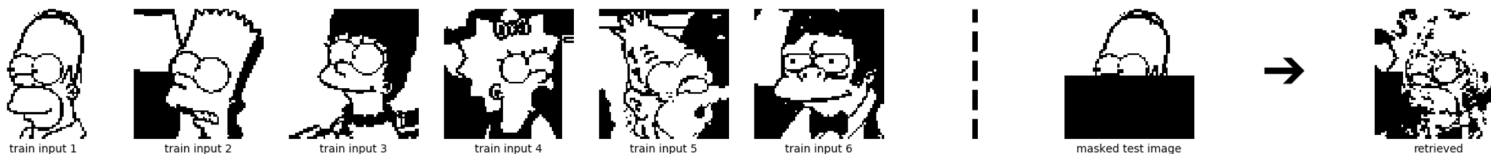
Associative Memory

- Memory aims to store content for easy retrieval
 - Associative memories (Hopfield Networks) can be viewed as energy-based models

$$E = -\frac{1}{2} \sum_{i,j=1}^N s_i W_{ij} s_j$$

$$W_{ij} = \sum_{k=1}^K \xi_i^k \xi_j^k$$

”Superposition” of k slots
Hebbian learning



Associative Memory

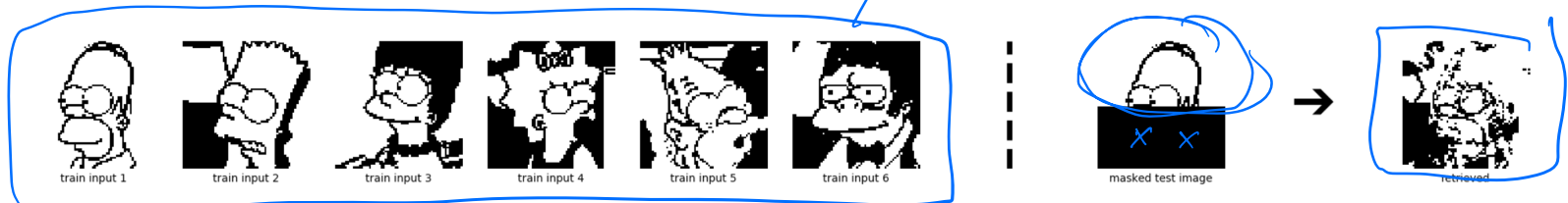
- Memory aims to store content for easy retrieval

- Associative memories (Hopfield Networks) can be viewed as energy-based models

$$E = -\frac{1}{2} \sum_{i,j=1}^N s_i W_{ij} s_j, \quad W_{ij} = \sum_{k=1}^K \xi_i^k \xi_j^k.$$

"Superposition" of k slots
Hebbian learning

- When presented with a new pattern the network should respond with a stored memory which most closely resembles the input.



Associative Memory

- Memory aims to store content for easy retrieval

- Associative memories (Hopfield Networks) can be viewed as energy-based models

$$E = -\frac{1}{2} \sum_{i,j=1}^N s_i W_{ij} s_j, \quad W_{ij} = \sum_{k=1}^K \xi_i^k \xi_j^k.$$

“Superposition” of k slots
Hebbian learning

- When presented with a new pattern the network should respond with a stored memory which most closely resembles the input.

- Retrieval: $s_i = \text{sign}(\sum_j W_{ij} s_j)$ Storage: $C \approx \frac{d}{2 \log(d)} = 0.14d.$



Relation to Transformers

- General form:

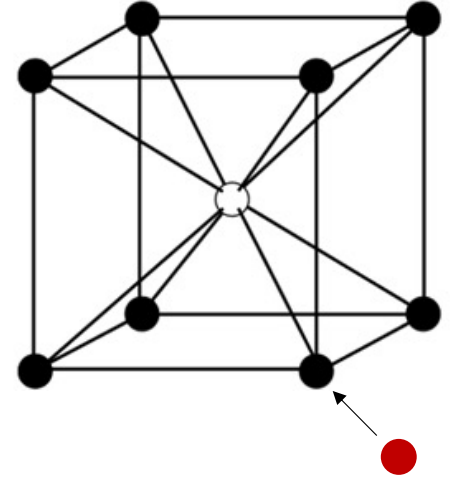
$$E = - \sum_k F\left(\sum_i \xi_i^k s_i\right).$$

Relation to Transformers

- General form:

$$E = - \sum_k F \left(\sum_i \xi_i^k s_i \right).$$

- When $F(z) = z^2$ it gives the classic HN.



$$\nabla_{s_i} E = - \sum_j W_{ij} s_j$$
$$s_i \leftarrow \text{sign} \left(\sum_j W_{ij} s_j \right)$$

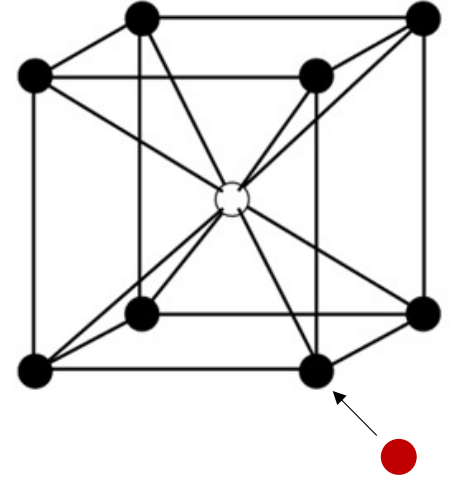
Relation to Transformers

- General form:

$$E = - \sum_k F\left(\sum_i \xi_i^k s_i\right).$$

- When $F(z) = z^2$ it gives the classic HN.
- Transformer-like attention operation:

$$\mathbf{Z} \leftarrow \text{softmax}(\beta \mathbf{X} \mathbf{W}_q \mathbf{W}_k^\top \mathbf{Y}^\top) \mathbf{Y}_i \mathbf{W}_v.$$



$$\nabla_{s_i} E = - \sum_j W_{ij} s_j$$
$$s_i \leftarrow \text{sign}\left(\sum_j W_{ij} s_j\right)$$

Relation to Transformers

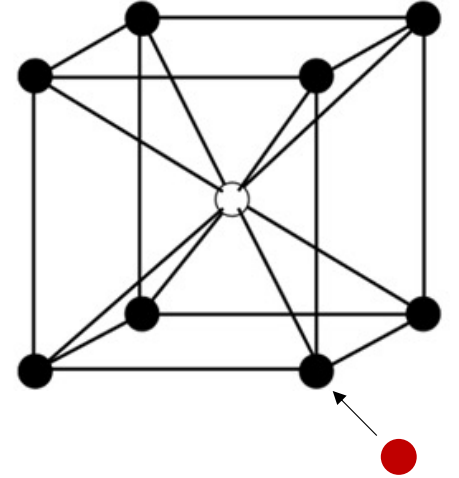
- General form:

$$E = - \sum_k F\left(\sum_i \xi_i^k s_i\right).$$

- When $F(z) = z^2$ it gives the classic HN.
- Transformer-like attention operation:

$$\mathbf{Z} \leftarrow \text{softmax}(\beta \mathbf{X} \mathbf{W}_q \mathbf{W}_k^\top \mathbf{Y}^\top) \mathbf{Y}_i \mathbf{W}_v.$$

$$\mathbf{S} \leftarrow \text{softmax}(\beta \mathbf{S} \mathbf{\Xi}^\top) \mathbf{\Xi}.$$

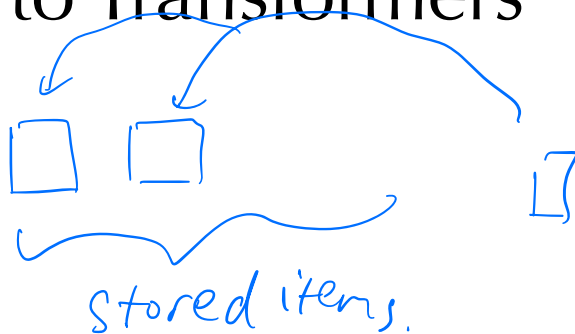


$$\nabla_{s_i} E = - \sum_j W_{ij} s_j$$
$$s_i \leftarrow \text{sign}\left(\sum_j W_{ij} s_j\right)$$

Relation to Transformers

- General form:

$$E = - \sum_k F\left(\sum_i \xi_{s_i}^k s_i\right).$$

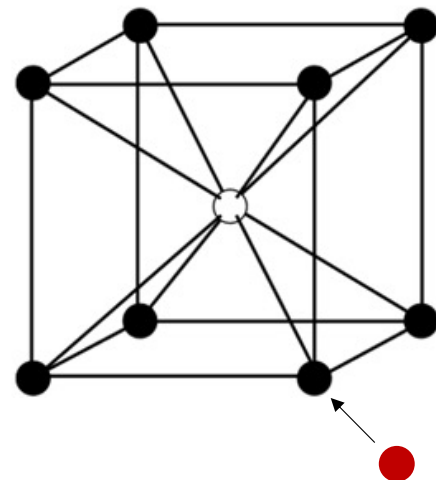


- When $F(z) = z^2$ it gives the classic HN.
- Transformer-like attention operation:

$$\mathbf{Z} \leftarrow \text{softmax}(\beta \mathbf{X} \mathbf{W}_q \mathbf{W}_k^\top \mathbf{Y}^\top) \mathbf{Y}_i \mathbf{W}_v.$$

$$\mathbf{S} \leftarrow \text{softmax}(\beta \mathbf{S} \mathbf{\Xi}^\top) \mathbf{\Xi}.$$

$$E = \underbrace{-\log \text{sumexp}(\beta, \mathbf{\Xi}^\top \mathbf{s})}_{\text{attention}} + \underbrace{\frac{1}{2} \mathbf{s}^\top \mathbf{s} + \beta^{-1} \log N + \frac{1}{2} M^2}_{\text{regularization}}.$$



$$\nabla_{s_i} E = - \sum_j W_{ij} s_j$$

$$s_i \leftarrow \text{sign}\left(\sum_j W_{ij} s_j\right)$$

Linear Attention as Memory

- Regular softmax attention

$$\text{Attention}(Q, K, V) = \text{softmax} \left(\frac{QK^T}{\sqrt{d_k}} \right) V$$

Linear Attention as Memory

- Regular softmax attention

$$\text{Attention}(Q, K, V) = \text{softmax} \left(\frac{QK^T}{\sqrt{d_k}} \right) V$$

- Problem: need to store a past tokens to compute the softmax.

Linear Attention as Memory

- Regular softmax attention

$$\text{Attention}(Q, K, V) = \text{softmax} \left(\frac{QK^T}{\sqrt{d_k}} \right) V$$

- Problem: need to store a past tokens to compute the softmax.

- Linear attention $Q(\underbrace{K^T V})$

Linear Attention as Memory

- Regular softmax attention

$$\text{Attention}(Q, K, V) = \text{softmax} \left(\frac{QK^T}{\sqrt{d_k}} \right) V$$

- Problem: need to store a past tokens to compute the softmax.
- Linear attention $Q(K^T V)$
- Store the KV matrix (fast weights).

Linear Attention as Memory

- Regular softmax attention

$$\text{Attention}(Q, K, V) = \text{softmax} \left(\frac{QK^T}{\sqrt{d_k}} \right) V$$

- Problem: need to store a past tokens to compute the softmax.

- Linear attention $Q(K^T V)$

- Store the KV matrix (fast weights).

- Write: $S_t = S_{t-1} + K_t V_t^T$

Linear Attention as Memory

- Regular softmax attention

$$\text{Attention}(Q, K, V) = \text{softmax} \left(\frac{QK^T}{\sqrt{d_k}} \right) V$$

- Problem: need to store a past tokens to compute the softmax.

- Linear attention $Q(K^T V)$

\downarrow
 ~~$d \times n$~~ $(n) \times (d)$

- Store the KV matrix (fast weights).

- Write: $S_t = S_{t-1} + K_t V_t^T$

$d \times d$

- Read: $O_t = Q_t S_t$

$O(n)$

More Update Rules

- **Saturation Problem:** The memory state continually accumulates information without bound.

$$S_t = S_{t-1} + K_t V_t^T$$

More Update Rules

- **Saturation Problem:** The memory state continually accumulates information without bound.

$$S_t = S_{t-1} + K_t V_t^T$$

- Pure Linear Attention lacks a mechanism to discard irrelevant information.

More Update Rules

- **Saturation Problem:** The memory state continually accumulates information without bound.

$$S_t = S_{t-1} + K_t V_t^T$$

- Pure Linear Attention lacks a mechanism to discard irrelevant information.

- Fixed Decay (Mamba, RWKV): $S_t = \gamma S_{t-1} + K_t V_t^T$

More Update Rules

- **Saturation Problem:** The memory state continually accumulates information without bound.

$$S_t = S_{t-1} + K_t V_t^T$$

- Pure Linear Attention lacks a mechanism to discard irrelevant information.

- Fixed Decay (Mamba, RWKV): $S_t = \gamma S_{t-1} + K_t V_t^T$

- Data-dependent Decay (GLA): $S_t = g_t \odot S_{t-1} + K_t V_t^T$

↓
gating

More Update Rules

- **Saturation Problem:** The memory state continually accumulates information without bound.

$$S_t = S_{t-1} + K_t V_t^T$$

- Pure Linear Attention lacks a mechanism to discard irrelevant information.

- Fixed Decay (Mamba, RWKV): $S_t = \gamma S_{t-1} + K_t V_t^T$

- Data-dependent Decay (GLA): $S_t = g_t \odot S_{t-1} + K_t V_t^T$

- DeltaNet:

$$S_t = S_{t-1} + E_t K_t^T \quad E_t = \underbrace{V_t - \hat{V}_t}_{\text{prediction error.}}$$

Test-Time Training

- We can also formulate memory update as a form of gradient descent.

$$\begin{aligned} \ell_t(W) &= \frac{1}{2} \|Wk_t - v_t\|^2 \\ \nabla_W \ell_t(W) &= (Wk_t - v_t)k_t^T \end{aligned}$$

Test-Time Training

- We can also formulate memory update as a form of gradient descent.

$$\ell_t(W) = \frac{1}{2} \|W k_t - v_t\|^2$$

$$\nabla_W \ell_t(W) = (W k_t - v_t) k_t^T$$

$$W_t = \underline{W_{t-1}} - \eta \underline{\nabla_W \ell_t(W_{t-1})}$$

$$W_t = W_{t-1} - \eta (\underline{W_{t-1} k_t - v_t}) k_t^T$$

$$W_t = W_{t-1} \oplus \eta (v_t - W_{t-1} k_t) k_t^T$$

Test-Time Training

- We can also formulate memory update as a form of gradient descent.

$$\ell_t(W) = \frac{1}{2} \|Wk_t - v_t\|^2$$

$$\nabla_W \ell_t(W) = (Wk_t - v_t)k_t^T$$

$$W_t = W_{t-1} - \eta \nabla_W \ell_t(W_{t-1})$$

$$W_t = W_{t-1} - \eta (W_{t-1}k_t - v_t)k_t^T$$

$$W_t = W_{t-1} + \eta (v_t - W_{t-1}k_t)k_t^T$$

DeltaNet $W_t = W_{t-1} + \eta E_t k_t^T$

Test-Time Training

- We can also form a deeper memory using MLP. Linear memory has the capacity upper bound.

$$f(x; W) = W_2(\text{GELU}(W_1 x))$$

$$\ell_t(W) = \frac{1}{2} \| \underset{\substack{\uparrow \\ \text{Current}}}{f(k_t; W)} - \underset{\substack{\uparrow \\ \text{value.}}}{v_t} \|^2.$$

Test-Time Training

- We can also form a deeper memory using MLP. Linear memory has the capacity upper bound.

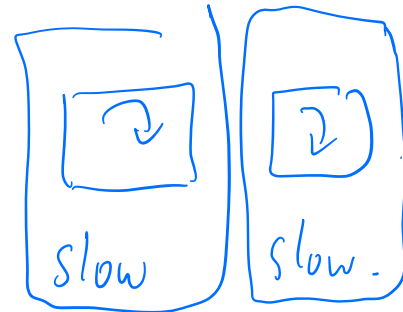
$$f(x; W) = W_2(\text{GELU}(W_1 x))$$

$$\ell_t(W) = \frac{1}{2} \|f(k_t; W) - v_t\|^2.$$

- The TTT View on CL: The context window is an online data stream:

$$\mathcal{D} = \{(k_1, v_1), (k_2, v_2), \dots, (k_N, v_N)\}$$

Test-Time Training



- We can also form a deeper memory using MLP. Linear memory has the capacity upper bound.

$$f(x; W) = W_2(\text{GELU}(W_1x))$$

$$\ell_t(W) = \frac{1}{2} \|f(k_t; W) - v_t\|^2.$$

- The TTT View on CL: The context window is an online data stream:

$$\mathcal{D} = \{(k_1, v_1), (k_2, v_2), \dots, (k_N, v_N)\}$$

- The Outer Model is meta-trained to project tokens into highly orthogonal key vectors to prevent forgetting.

Summary: Continual Learning

- Regularization, Distillation, Architecture Expansion/Isolation

Summary: Continual Learning

- Regularization, Distillation, Architecture Expansion/Isolation
- Frozen representation: prompt learning

Summary: Continual Learning

- Regularization, Distillation, Architecture Expansion/Isolation
- Frozen representation: prompt learning
- Combination with self-supervised learning

Summary: Continual Learning

- Regularization, Distillation, Architecture Expansion/Isolation
- Frozen representation: prompt learning
- Combination with self-supervised learning
- Integration of memory and representations

Summary: Continual Learning

- Regularization, Distillation, Architecture Expansion/Isolation
- Frozen representation: prompt learning
- Combination with self-supervised learning
- Integration of memory and representations
- Exploration of multimodal continual learning and planning from embodied environments. Adapt:
 - Changing visual scenes and statistics
 - World models, dynamics
 - Environment rewards

Summary: Continual Learning

- Regularization, Distillation, Architecture Expansion/Isolation
- Frozen representation: prompt learning
- Combination with self-supervised learning
- Integration of memory and representations
- Exploration of multimodal continual learning and planning from embodied environments. Adapt:
 - Changing visual scenes and statistics
 - World models, dynamics
 - Environment rewards
- Understand real-world structure. Online sequence perspective.