Massachusetts Institute of Technology

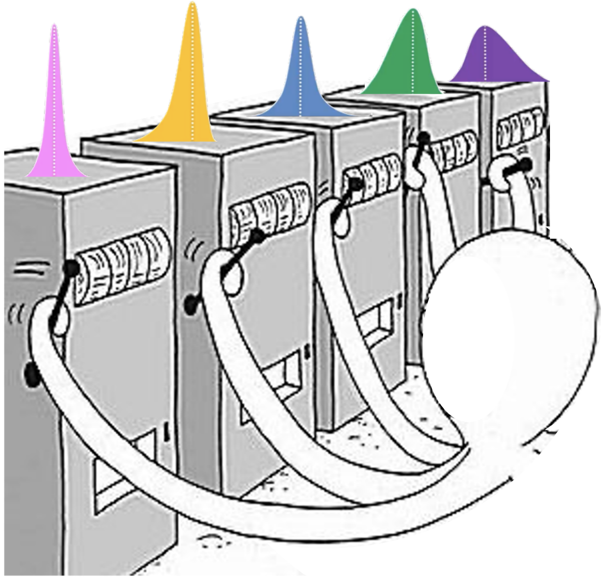
Meta Learning

MIT
Iddo Drori, Fall 2020

Multi-Armed Bandit

Stateless

- Action: pull one of k arms
- Reward for pulling that arm



at each time step t:

choose action a_t among k actions

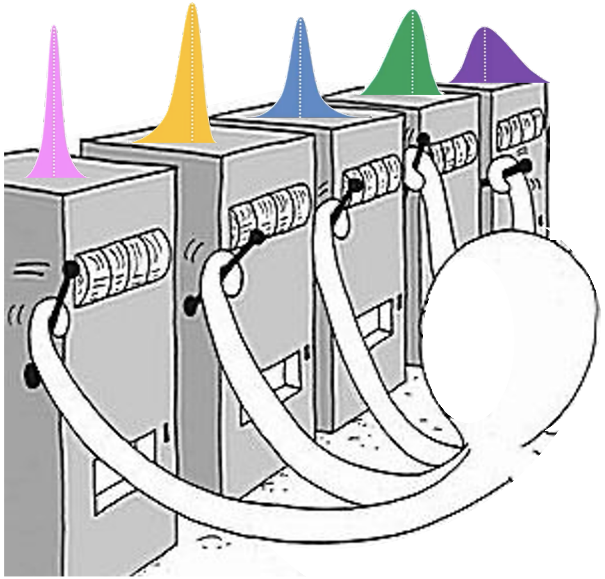
receive reward r_t for taking action a_t

- Taking action a is pulling arm i which gives reward $r(a)$ with probability p_i
- Probabilities distributions $p_1,\dots,p_k$ are unknown
- Goal is to maximize total expected return

Multi-Armed Bandit

- Value of action a is expected reward: $Q^*(a) = E[r_t \mid a_t = a]$
  we don't know the action values

- Estimate value of action a at time t: $Q_t(a)$
- For example keep current mean reward for each action



Greedy Action

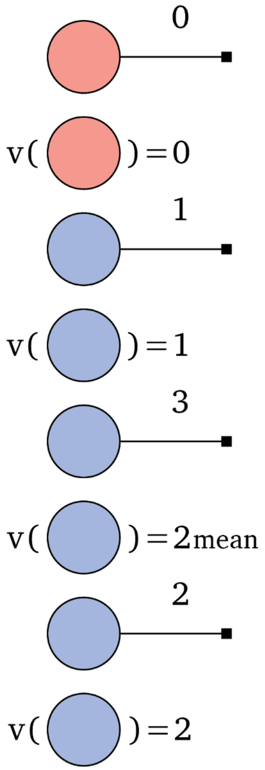
- A greedy action takes the best estimate at time t, exploiting knowledge

  $a_t = \text{argmax\_a } Q_t(a)$

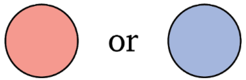
- For example choose action with largest mean reward.

- A non-greedy action is exploring.

Greedy Action Selection



Which to choose next?



ε-Greedy

- Behave greedily most of the time:

with probability ε choose random action
with probability $1-\varepsilon$ choose greedy action

for each each action a **do**
$Q(a) = 0$
$N(a) = 0$ number of times action is chosen
for each time step **do**
$a = \underset{a}{\operatorname{argmax}} Q(a)$ with probability $1-\epsilon$ and random action with probability ϵ.
$N(a) = N(a) + 1$
$Q(a) = Q(a) + (r(a) - Q(a))/N(a)$

Upper Confidence Bound (UCB)

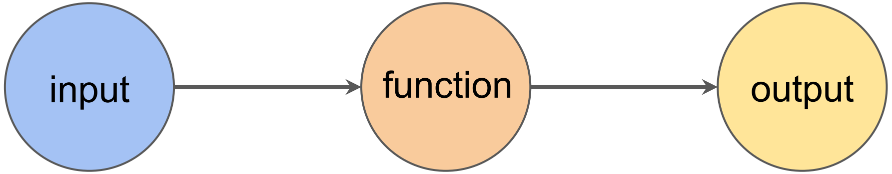
- Optimism in face of uncertainty
- Use both mean and variance of reward

$$\underset{a}{\operatorname{argmax}}(\mu(r(a)) + \epsilon\sigma(r(a)))$$

- Finite-time Analysis of the Multiarmed Bandit Problem, Auer et al, Machine Learning, 2012
- Used in Monte Carlo tree search (MCTS), in expert iteration and AlphaZero.

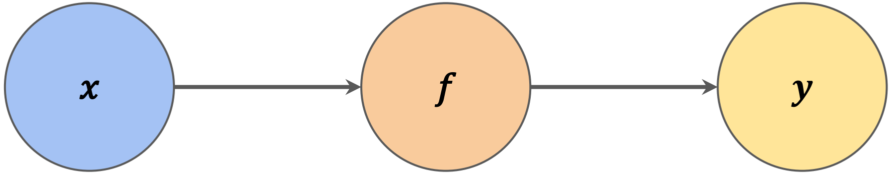
Observation

- Input $\mathbf{x}_{dx1}$
- Function f
- Output y_{1x1}



Observation

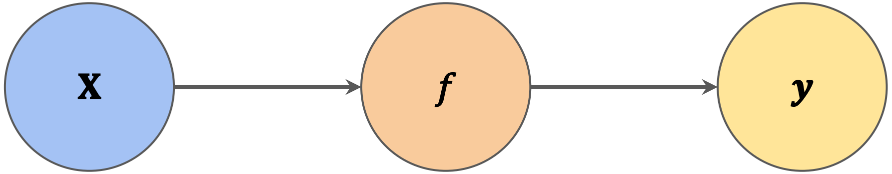
- Input $\boldsymbol{x}_{\text{dx1}}$
- Function $\boldsymbol{f}$
- Output $\boldsymbol{y}_{\text{1x1}}$



$$\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x})$$

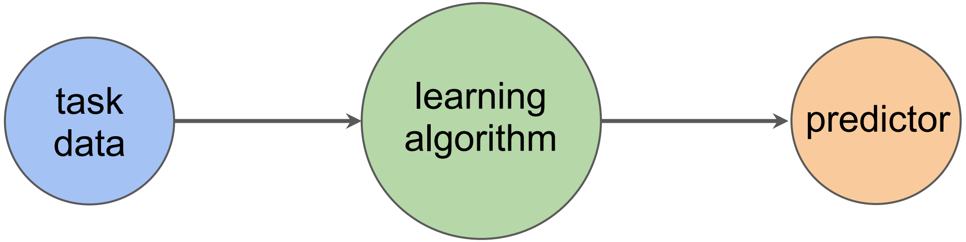
Observations

- Input $\mathbf{X}_{dxm}$
- Function f
- Output y_{mx1}

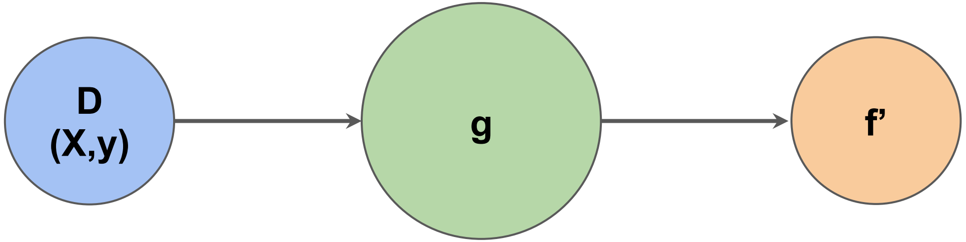


$$y = f(X)$$

Supervised Learning

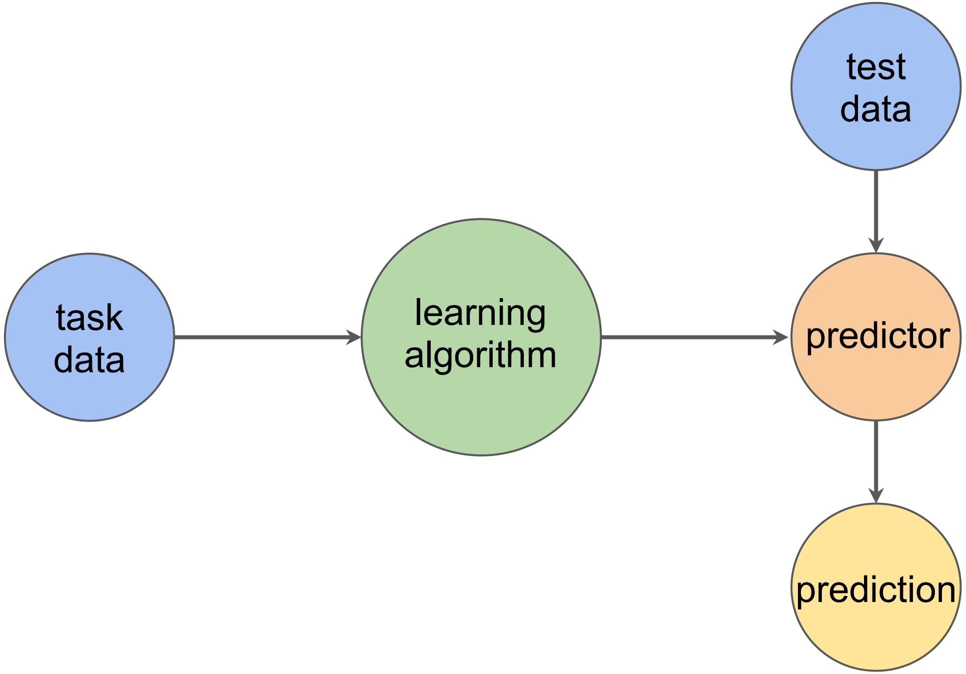


Supervised Learning

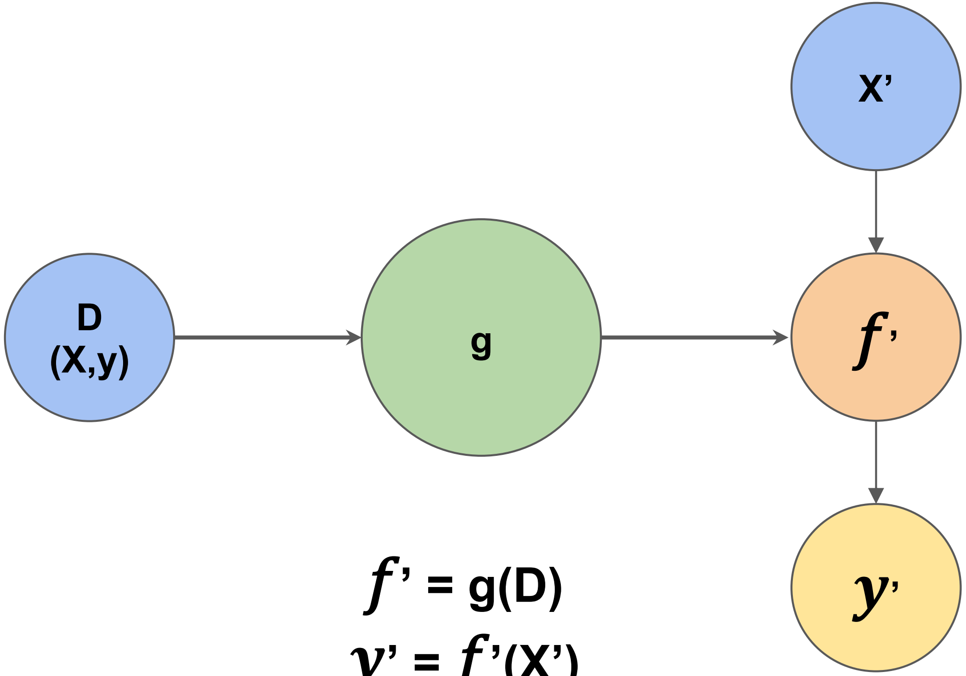


$f' = g(D)$

Supervised Learning



Supervised Learning

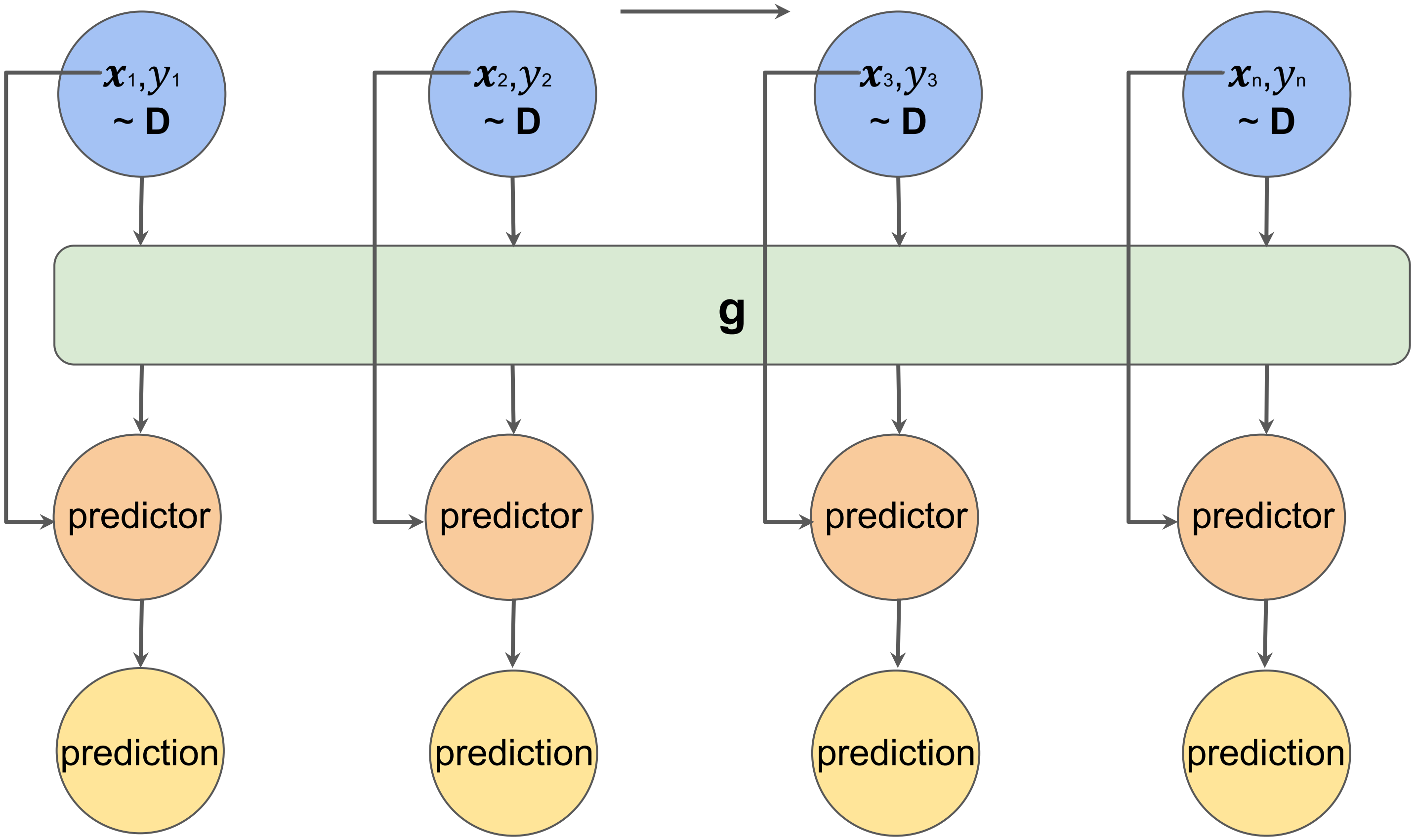


$$f' = g(D)$$
$$y' = f'(X')$$

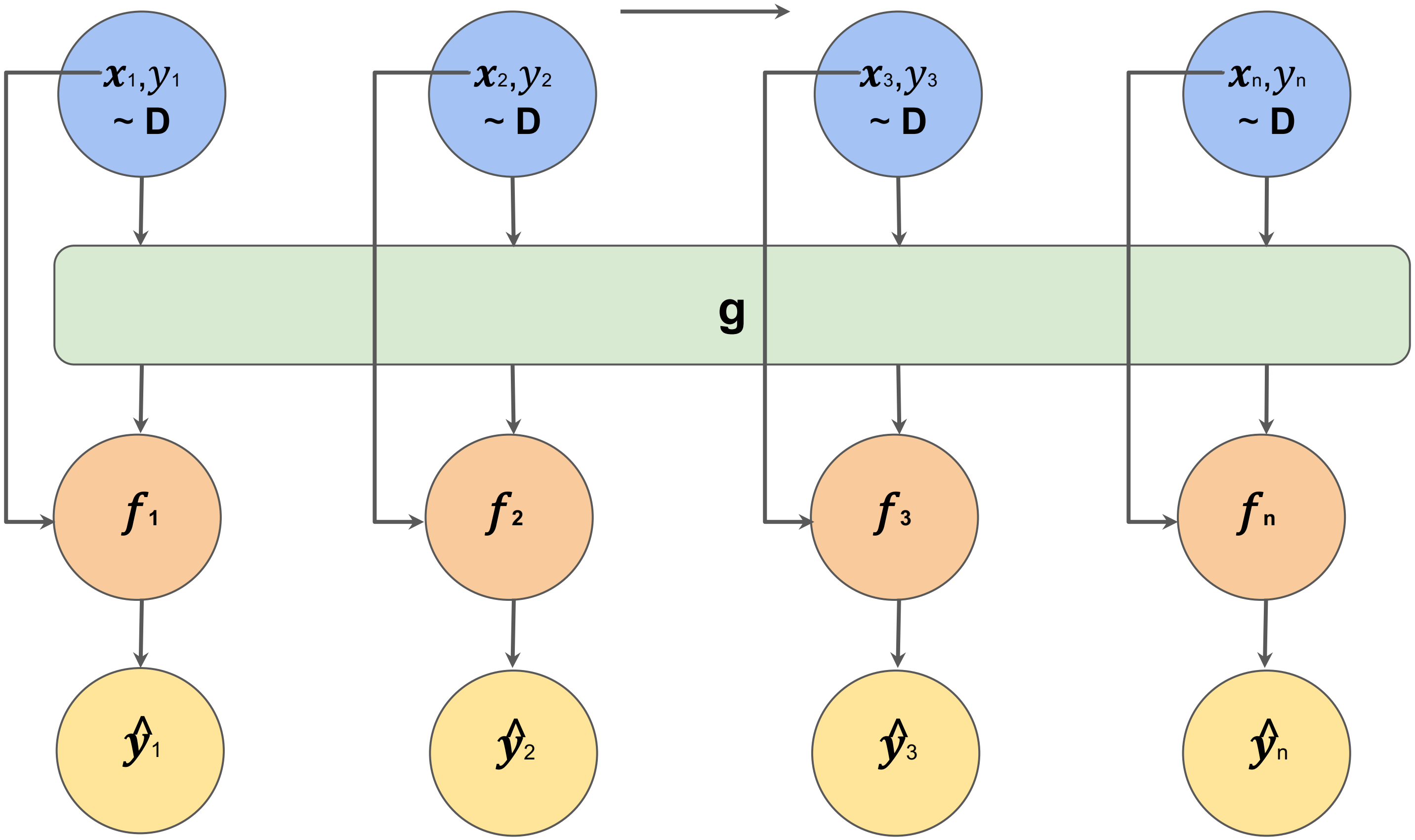
Online Learning

- Learning algorithm **g** interacts with oracle multiple rounds
- **g** holds a classifier f
- For each round
  - Oracle selects a pair $(\boldsymbol{x}, y)$
  - Learning algorithm **g** presented with sample $\boldsymbol{x}$
  - **g** outputs prediction $\hat{y} = \boldsymbol{f}(\boldsymbol{x})$
  - Oracle reveals ground truth label y
  - **g** changes f if mistaken: $y \neq \hat{y}$

Online Learning



Online Learning



Online Learning

- Goal of learning algorithm **g** is to make as few mistakes
- Oracle may be adversary

Perceptron Algorithm

- init $\boldsymbol{\theta} = \mathbf{0}$, $\boldsymbol{\theta}_0 = 0$
- for t = 1..T

  changed = false

  – for i = 1..n
    - given example $\boldsymbol{x}_i$
    - if prediction was a mistake $y_i(\boldsymbol{\theta}^T\boldsymbol{x}_i + \boldsymbol{\theta}_0) \leqslant 0$
      – then update $\boldsymbol{\theta} = \boldsymbol{\theta} + y_i\boldsymbol{x}_i$, $\boldsymbol{\theta}_0 = \boldsymbol{\theta}_0 + y_i$
      – changed = true

  – if not changed then break

Winnow Algorithm

- Boolean feature vector $\boldsymbol{x}$ in $\{0,1\}^n$
- Weights $\boldsymbol{w} = (\boldsymbol{w}_1,..,\boldsymbol{w}_n)$ initialized to 1's
- Given example $\boldsymbol{x}$ then if $\boldsymbol{w}^T\boldsymbol{x} > t$ predict 1 otherwise 0
- If mistakenly predicted 1 then set $\boldsymbol{w}_i = 0$ for all i s.t. $\boldsymbol{x}_i = 1$
- If mistakenly predicted 0 then set $\boldsymbol{w}_i = 2\boldsymbol{w}_i$ for all i s.t. $\boldsymbol{x}_i = 0$

Halving Algorithm

- For each round
  - Each expert i = 1..N makes a prediction $\hat{y}_i = f_i(x)$
  - Take majority vote of correct experts until round
  - Oracle reveals ground truth label y

- Algorithm with as few mistakes as possible
- If there is a perfect expert then at most logN mistakes

Online Learning to Batch Learning

- Given online learning algorithm **g**
- Examples S of pairs $(\boldsymbol{x}, y)$
- Repeat
  - for each pair $(\boldsymbol{x}, y)$ in S
    - predict $\hat{y}$ using **g**
    - If $y \neq \hat{y}$ remove pair from S
- Until no mistake is made

Continual (Lifelong, Sequential) Learning

- Neural network classification: training and testing generalization, static, often training from scratch
- Continual learning: incremental, dynamic

- Neural networks have catastrophic forgetting of old concepts when learning new concepts
- Humans gradually forget, not completely forget all at once

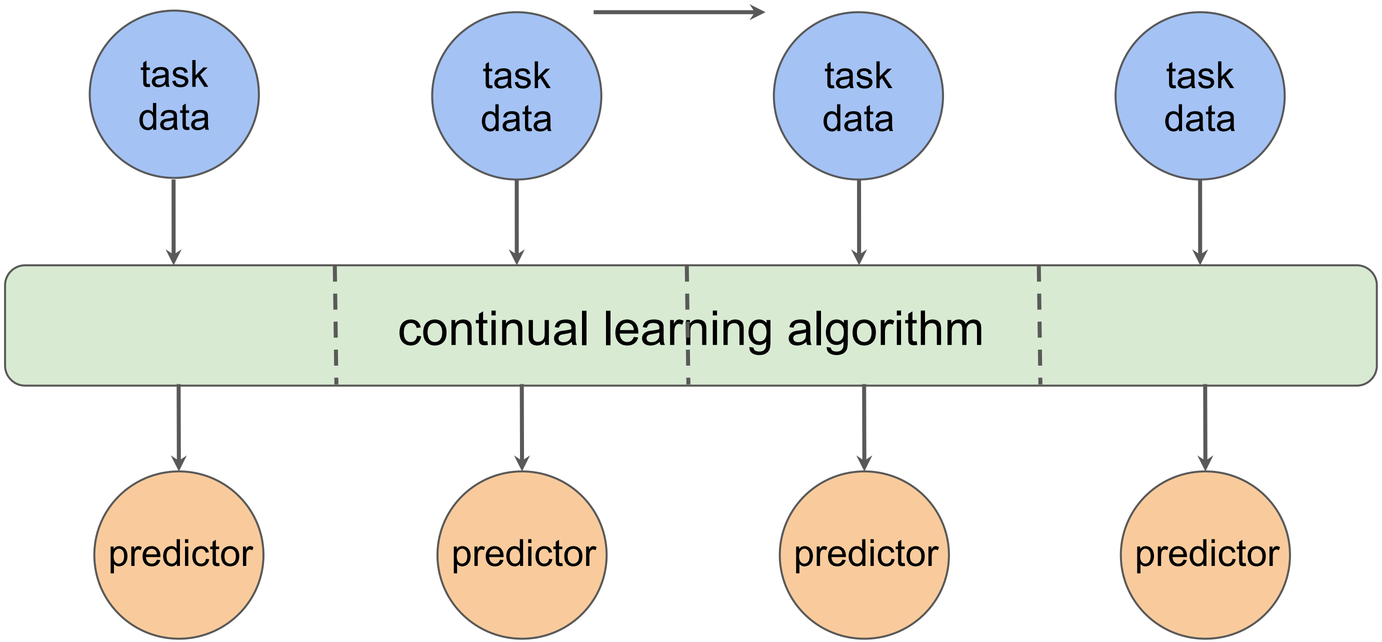
Stability vs. Plasticity

- Learning algorithm should preserve what it has learned: stability

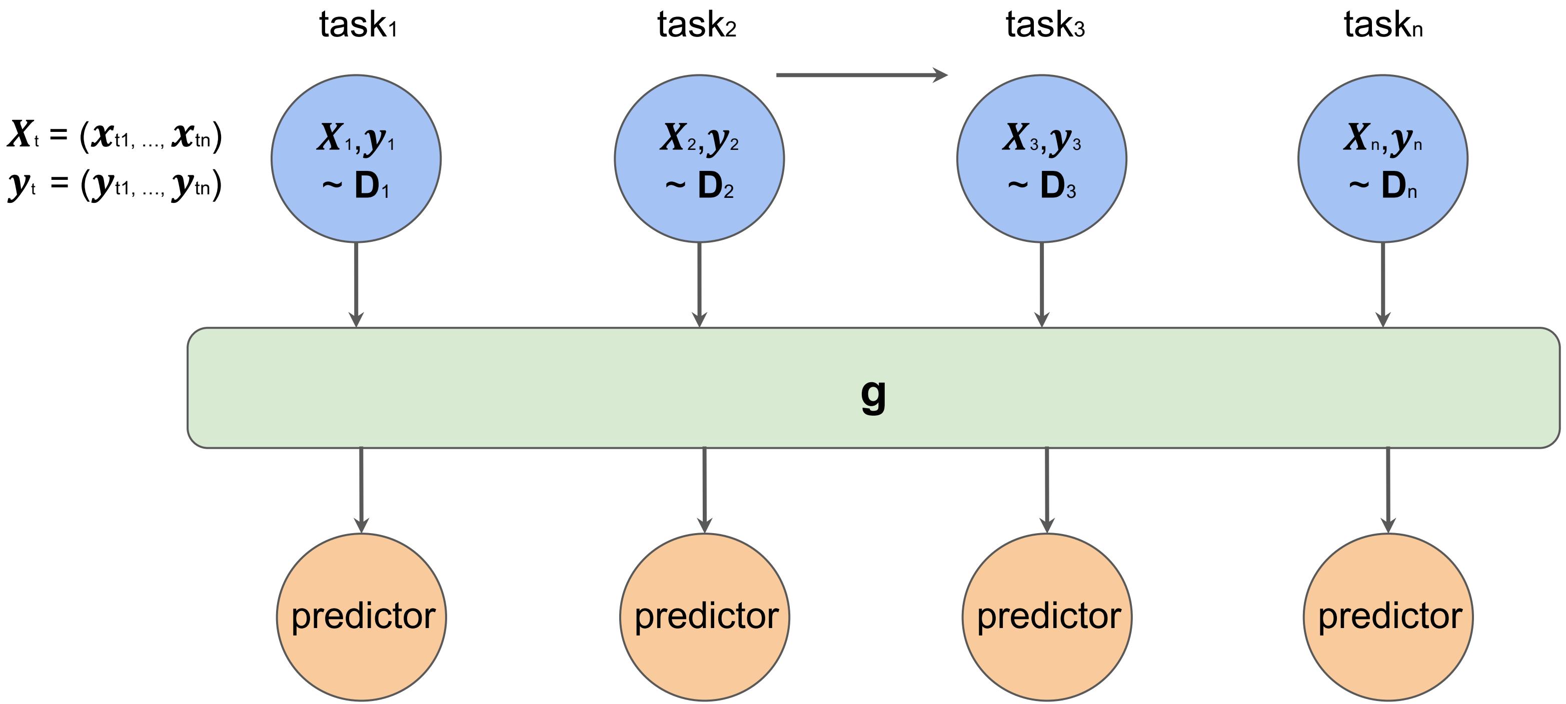
- Learning algorithm should quickly learn a new task: plasticity

- Goal: effectively update neural network with new information over time

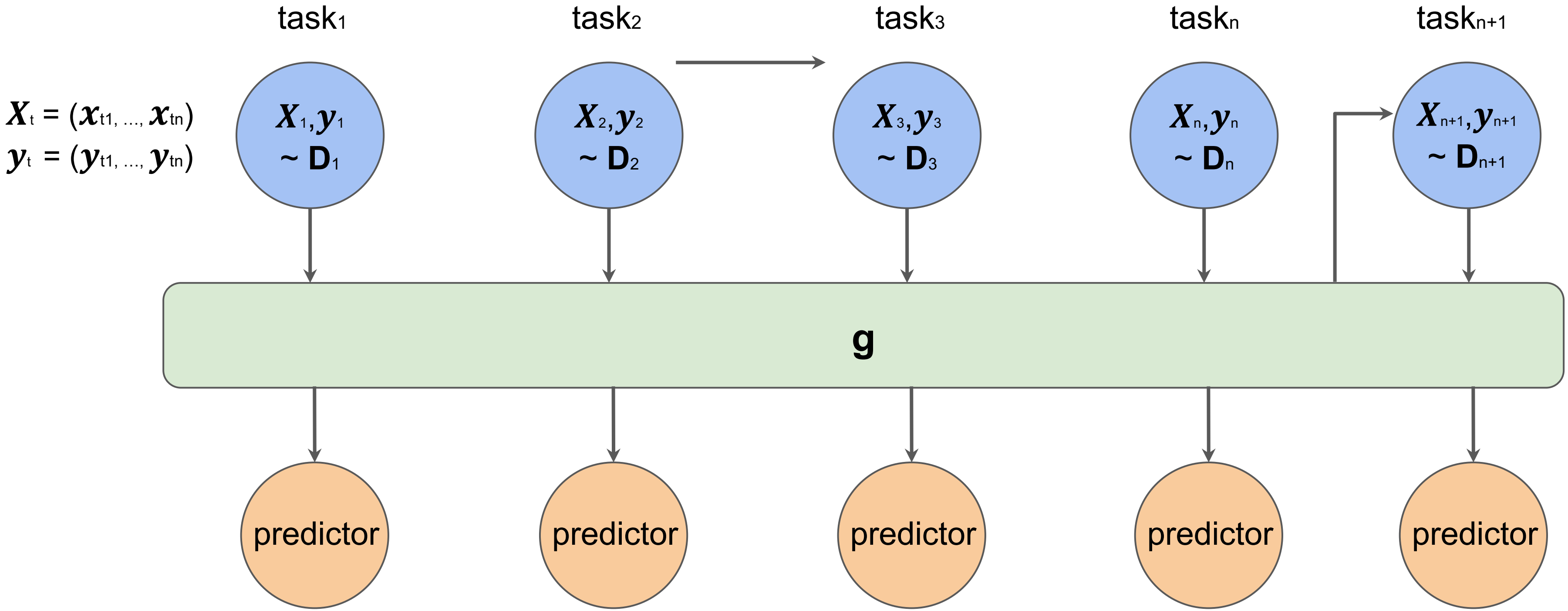
Continual (Lifelong) Learning



Continual (Lifelong) Learning



Learning New Tasks



Human Learning on the Job

- 10% formal education and training
- 70% on the job learning
- 20% observation of others
- Open world, self-supervision
- Discover new tasks and continually learn them

Learning on the job: Online lifelong and continual learning, Liu 2020.

Continual (Lifelong, Sequential) Learning

- Endless data stream
- Goal is to learn without catastrophic forgetting

  performance on previously learned task should not significantly degrade over time when learning new tasks

Backward Transfer

- Influence that learning task t has on performance on previous task k < t:

$$1/(t-1)\sum_{i=1..t-1} (R_{ti} - R_{ii})$$

  R_{ij} is test accuracy of model on task j after observing last sample from task i

- Positive backward transfer: learning task t increases performance on preceding task k.
- Negative backward transfer: learning task t decreases performance on preceding task k.
- Catastrophic forgetting: large negative backward transfer.

Forward Transfer

- Influence that learning task t has on performance of future task k > t.

$$1/(t-1)\sum_{i=2..t}(R_{i-1,i} - b_i)$$

b_i is test accuracy for task i at initialization

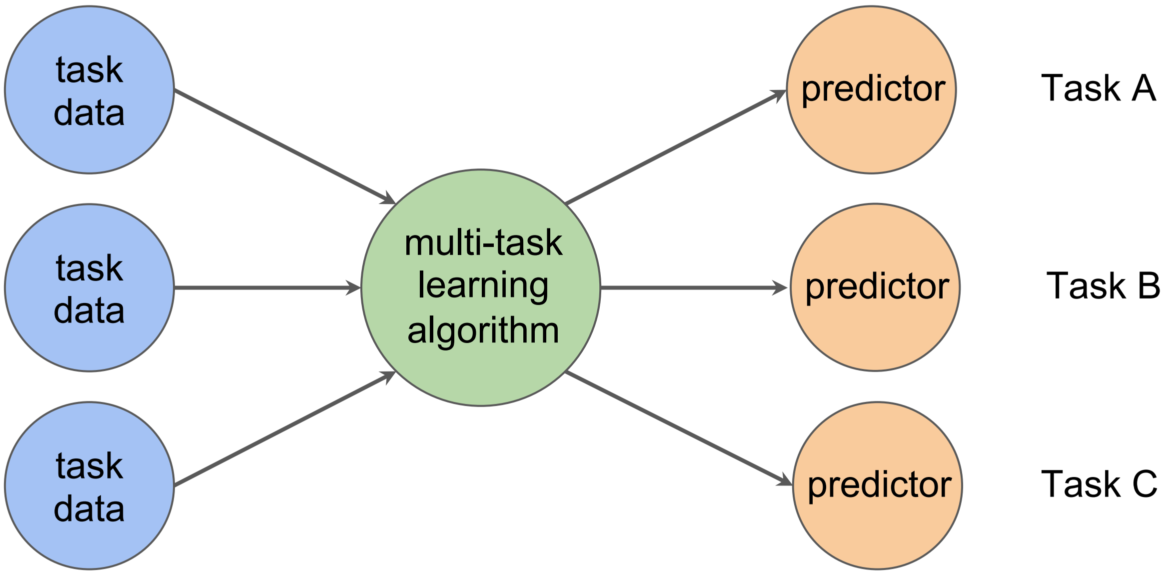
- Positive forward transfer: possible when model is able to perform zero-shot learning. For example, using structure available in task descriptors

Continual (Lifelong, Sequential) Learning

- Sequential tasks with batches of data
- Optimizing for a new task directly results in catastrophic forgetting

Multi-Task Learning

- Storing all past examples reduces continual learning problem to multi-task problem



Continual (Lifelong, Sequential) Learning

- Perform well on all previous tasks t optimizing:

$$\sum_{t=1..T} E_{X_t, y_t \sim D_t} (\mathcal{L}(f_t(X_t, \boldsymbol{\theta}), y_t))$$

  y_t is a vector $y_{t1},...,y_{tn}$

- For current task t

$$1/n \sum_{i=1..n} \mathcal{L}(f(x_i, \boldsymbol{\theta}), y_i)$$

Replay Methods

- Store past samples or generate pseudo samples
- Store sample subset of observations, for example representing each class
- Reuse for training when learning new task, interleave training on new and memorized data

Reservoir Sampling

- Stream (x_i, y_i) for $i=1..n$
- Memory size m
- For i = 1..n
  - If memory is not filled then store (x_i, y_i)
  - Else sample u ~ Uniform(0,1)
  - If u <= m/i
    - Replace random stored instance of class $c == y_i$ with (x_i, y_i)

    Else ignore (x_i, y_i)

Reservoir Sampling

- Stores iid subset of stream
- Problem: if stream is imbalanced then memory is imbalanced
- Forgetting underrepresented classes quickly
- Solution: take into account labels of observed instances
- Store underrepresented classes, iid subset of each class

Class Balanced Reservoir Sampling

- Stream (x_i,y_i) for i=1..n
- Memory size m
- For i = 1..n
  - If memory is not filled then store (x_i,y_i)
  - Else
    - If $c == y_i$ has not been largest class

      Overwrite a random instance from largest class with (x_i,y_i)
    - Else m_c=# stored instances of class $c == y_i$, n_c = total # of stream instance of class $c == y_i$
    - sample u ~ Uniform(0,1)
    - If $u <= m_c/n_c$
      - Replace random stored instance of class $c == y_i$ with (x_i,y_i)

      Else ignore (x_i,y_i)

Online continual learning from imbalanced data, Chrysakis et al, 2020.

Average Gradient Episodic Memory

- Store subset of observed examples of each task
- Minimize loss on current task while treating average losses of episodic memories of previous tasks as inequality constraint
- Avoid increasing, allow decreasing previous average loss, allows positive backward transfer. Learning task t objective:

$$\min_{\boldsymbol{\theta}} \mathcal{L}(f_{\boldsymbol{\theta}}, Dt) \text{ s.t. } \mathcal{L}(f_{\boldsymbol{\theta}}, M) \leqslant \mathcal{L}(f_{\boldsymbol{\theta}_{t-1}}, M)$$

where $M = \cup M_k$ for all $k < t$

$f_{\boldsymbol{\theta}_{t-1}}$ is the network trained until task t-1

- Increase in loss of previous tasks: computing angle between their loss gradient vector and proposed update

Efficient lifelong learning with A-GEM, Chaudhry et al, 2019

Regularization Methods for Continual Learning

- Add regularization terms in loss: penalize changes in network parameters when learning new task

- For example, per-parameter regularization parameters

Neural Network Training

- Optimizing parameters: finding most probable values given data D
- Bayes rule, log, rearranging terms

$$\log p(\boldsymbol{\theta}|D) = \log p(D|\boldsymbol{\theta}) + \log p(\boldsymbol{\theta}) - \log p(D)$$

$$L(\boldsymbol{\theta}) = -\log p(D|\boldsymbol{\theta})$$

Regularization for Continual Learning

- Data split into independent parts D_1 and D_2 for tasks:

$$\log p(\boldsymbol{\theta}|D) = \log p(D_2|\boldsymbol{\theta}) + \log p(\boldsymbol{\theta}|D_1) - \log p(D_2)$$

$L_2(\boldsymbol{\theta}) = -\log p(D_2|\boldsymbol{\theta})$, all information of task 1 in posterior $\log p(\boldsymbol{\theta}|D_1)$, approximate

- Estimate distribution over model parameters, use as prior when learning from new data, penalize changes to important parameters
- Remember old tasks: slow learning on weights for old tasks

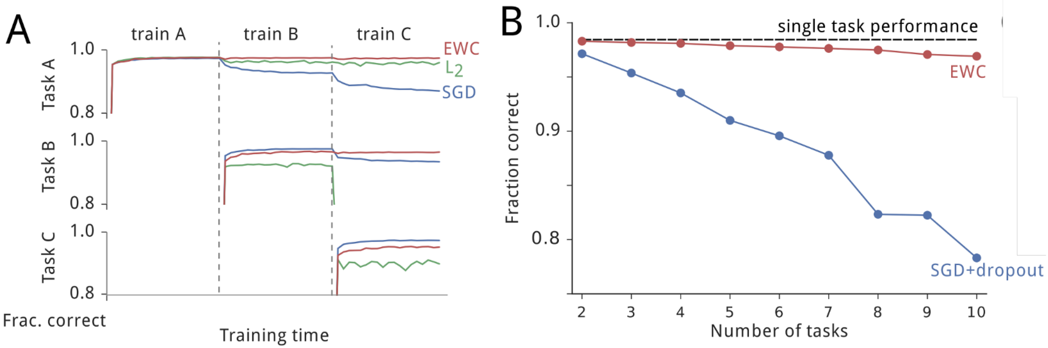


Figure source: Overcoming catastrophic forgetting in neural networks, Kirkpatrick et al, PNAS 2017

Elastic Weight Consolidation

- Given approximation

$$\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) = \mathcal{L}_2(\boldsymbol{\theta}) + \sum_i \alpha/2\, \boldsymbol{f}_i(\boldsymbol{\theta}_i - \boldsymbol{\theta}^*_{1i})$$

- $\mathcal{L}_2(\boldsymbol{\theta})$: loss of task 2 only
- α: importance of old task compared with new task
- i: each parameter label

Source: Overcoming catastrophic forgetting in neural networks, Kirkpatrick et al, PNAS 2017

Iterative Pruning and Retraining

- Exploit redundancies in DNNs to free up parameters that are used to learn new task

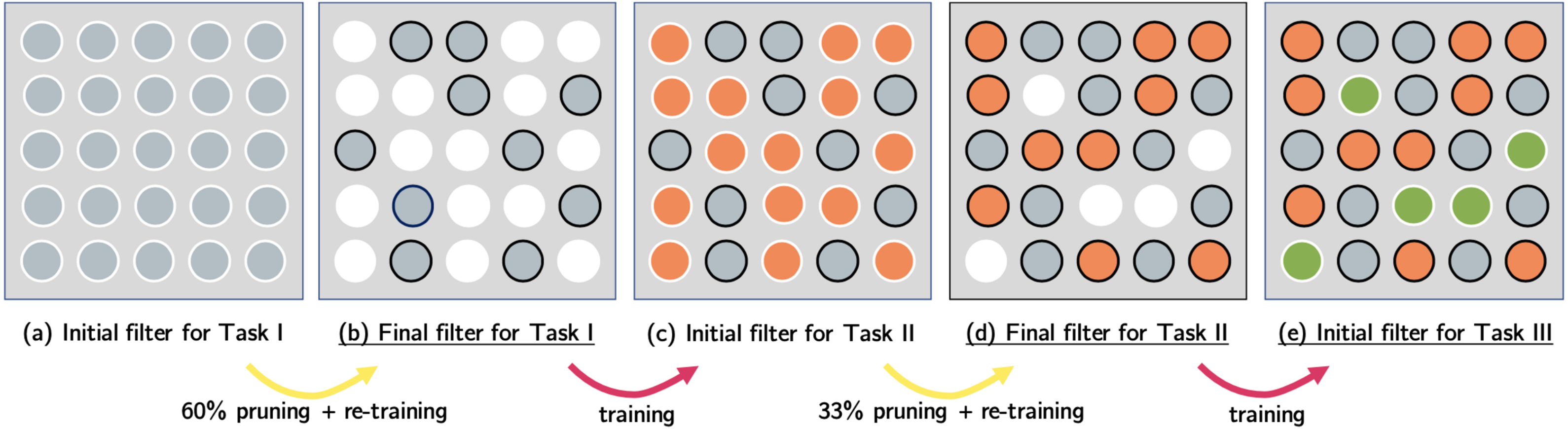


Figure source: PackNet: Adding Multiple Tasks to a Single Network by Iterative Pruning, Mallya and Lazebnik, 2018

Iterative Pruning and Retraining

- Training:
- Initial training of network for task 1 learns dense filter
- Pruning and re-training results in sparse filter for task 1: white circles 0 weights, task 1 weights in gray remain fixed and are not pruned.
- Pruned weights are updated for task 2: filter for task 2 shares weights learned for task 1.
- Pruning and re-training results in filter used for evaluating task 2, weights for task 2 (orange) are fixed. Process continues until running out of pruned weights

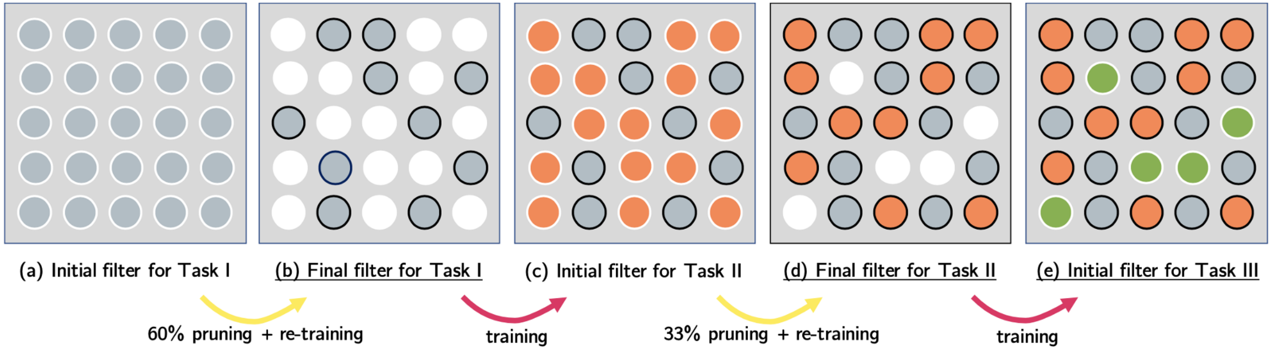


Figure source: PackNet: Adding Multiple Tasks to a Single Network by Iterative Pruning, Mallya and Lazebnik, 2018

Iterative Pruning and Retraining

- Testing: appropriate masks applied depending on task to replicate filters learned for task

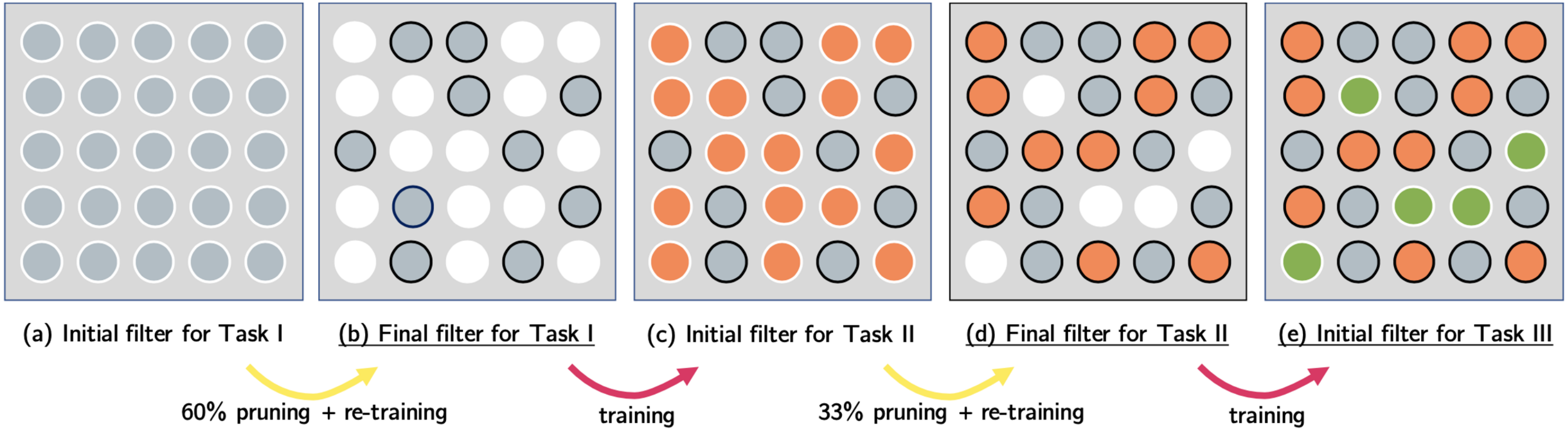


Figure source: PackNet: Adding Multiple Tasks to a Single Network by Iterative Pruning, Mallya and Lazebnik, 2018

Self-Supervised Learning

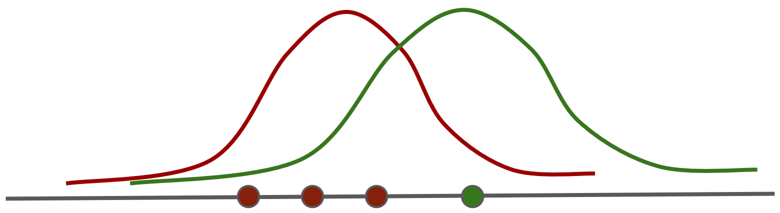
- Transform unlabeled images
- Label images according to transformation
- Train network with parameters (θb, θs)

  θb shared backbone parameters, θs self-supervision parameters
- Transfer learning using θb fine tuning for main task (θb, θm)

  θm main task parameters

Distribution Shift

- Training and test distributions are often different



Test-Time Training

- Unlabeled test sample x provides hint about test distribution
- Allow model parameters θ to depend on test sample x
- Learn from a single sample using self-supervision

Source: Test-time training with self-supervision for generalization under distribution shifts, Sun et al, 2020

Test-Time Training

- Training

$$\min_{\theta_b,\theta_m,\theta_s} \mathcal{L}_m(X, \mathbf{y}; \theta_b, \theta_m) + \mathcal{L}_s(X, \mathbf{y_s}; \theta_b, \theta_s)$$

- Testing: initialize with $\theta = (\theta_b, \theta_s)$ from training

$$\min_{\theta_b,\theta_s} \mathcal{L}_s(x, y_s; \theta_b, \theta_s)$$

- Prediction: $\theta(x) = \theta^*_b, \theta_m$, where θ^*_b is from testing
- Online prediction: $\theta(x_t)$ initialized with $\theta(x_{t-1})$, allows using information in $x_1,..,x_t$

Source: Test-time training with self-supervision for generalization under distribution shifts, Sun et al, 2020

Test-Time Training

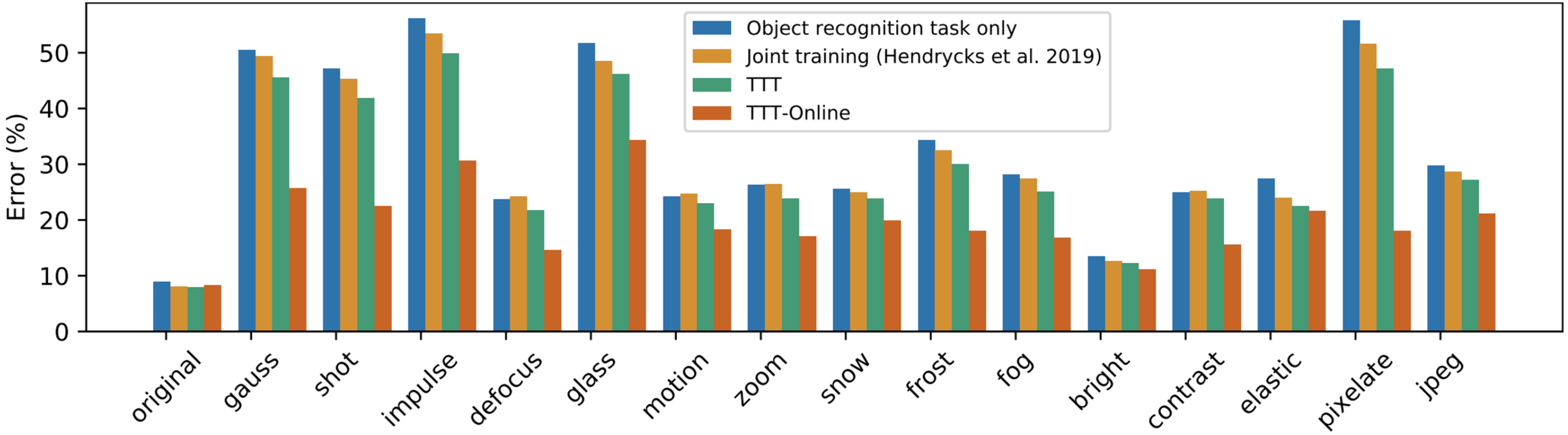


Figure source: Test-time training with self-supervision for generalization under distribution shifts, Sun et al, 2020

Test-Time Training

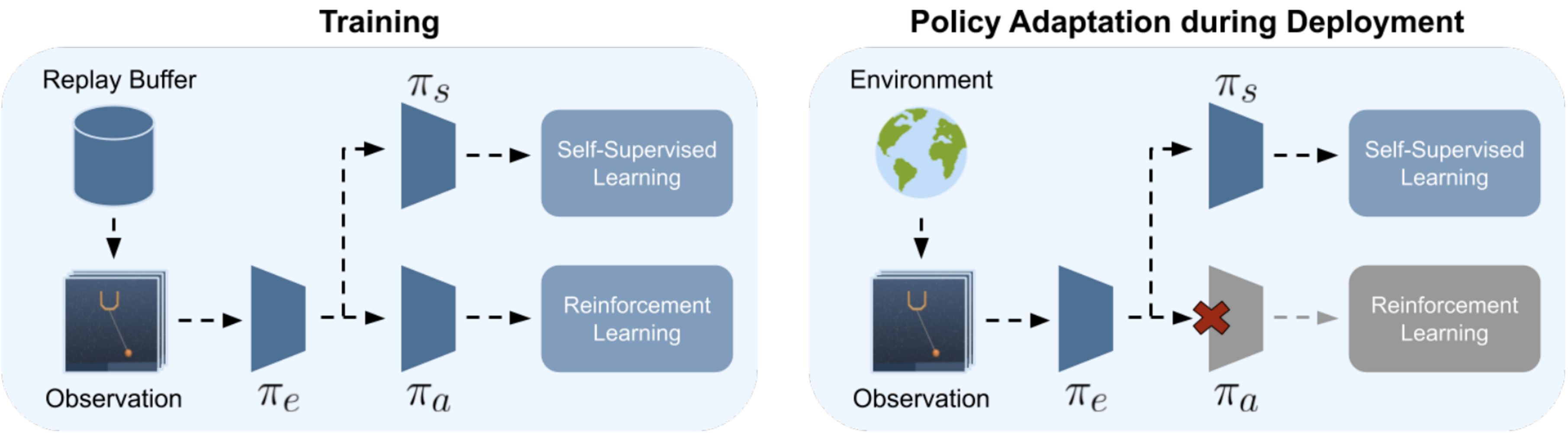


Figure source: Self-supervised policy adaptation during deployment, Hansen et al, 2020

Lifelong Latent Actor-Critic

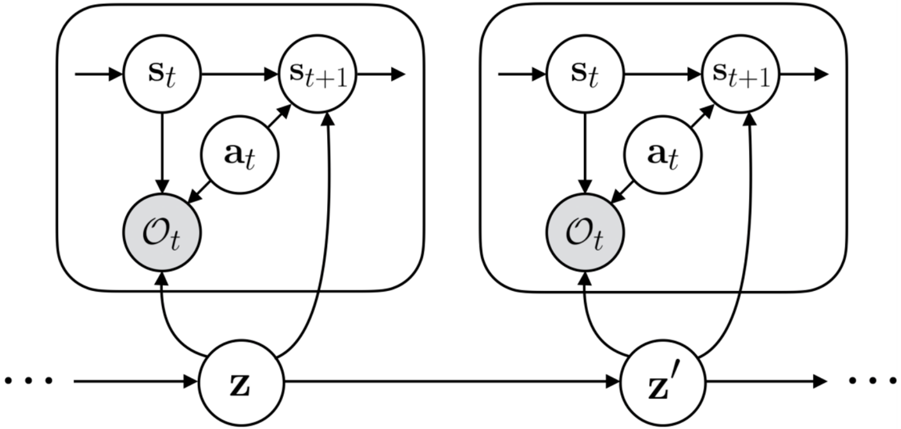


Figure source: Deep reinforcement learning amidst lifelong non-stationarity, Xie et al, 2020

Open-Ended Reinforcement Learning

- Grow population of environment-agent pairs
- Continually produce new and increasingly complex environments using mutations, random perturbations
- Train agents that learn to solve them
- Goal switching
- Compute distance between environments

Source: Enhanced POET: Open-ended reinforcement learning through unbounded invention of learning challenges and their solutions, Wang et al 2020

Self-Modeling

1. Robot recorded action-sensation pairs
2. Deep learning self-model consistent with data
3. Self-model used for planning of separate tasks
4. Emulate damage and morphological change
5. Adapting self-model from new data
6. Continual tasks

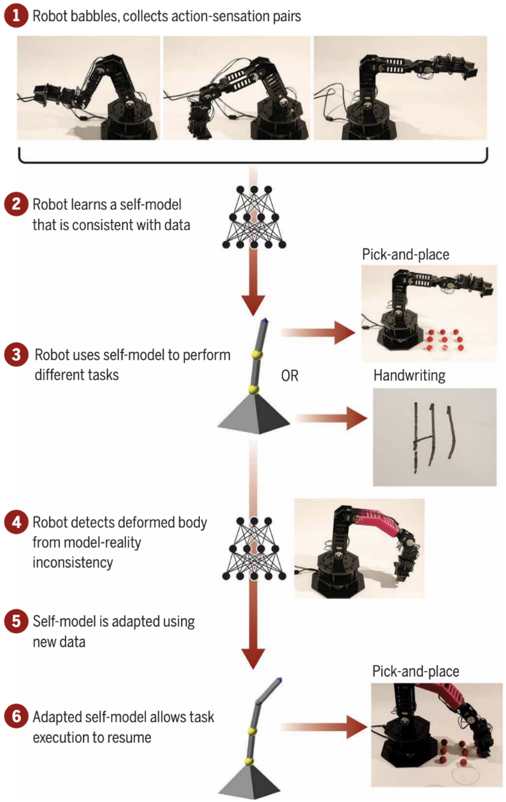


Figure source: Task-agnostic self-modeling machines, Kwiatkowski and Lipson, Science Robotics 2019

Meta Learning

MIT

Iddo Drori, Fall 2020