Reinforcement Learning Primer

Markov Decision Process (informally): agent taking actions $\in \{a_i\}$ in an environment while observing states $\in \{s_i\}$ and receiving rewards $\{r_i\}$, and seeking to maximize cumulative reward $r(\tau) = \sum_i r_i$ along trajectories $\{(s_i, a_i, r_i)\}$ that evolve probabilistically: $p(s_{i+1}|s_i, a_i)$.
Deep) Reinforcement Learning: agent learns a policy $a = \pi_\theta(s)$ so as to maximize its expected cumulative reward:

$$J(\theta) = \mathop{\mathbb{E}}_{\tau \sim \pi_\theta} r(\tau) = \int \pi_\theta(\tau) r(\tau) d\tau$$

$$\nabla_\theta J(\theta) = \int \nabla_\theta \pi_\theta r(\tau) d\tau = \int \pi_\theta \nabla_\theta \log \pi_\theta r(\tau) d\tau = \mathop{\mathbb{E}}_{\tau \sim \pi_\theta} r(\tau) \nabla_\theta \log \pi_\theta$$

'REINFORCE': follow policy π_θ recording $r(\tau)$ then update θ using $\nabla_\theta J(\theta)$. Note: $\nabla_\theta \log \pi_\theta = \sum_i \nabla_\theta \pi(a_i|s_i)$, and available if π_θ is a NN.

Normal RL $\quad \pi_\theta(s) \to \begin{bmatrix} \vdots \\ p(a_i) \\ \vdots \end{bmatrix}$

MAML for RL

ω

Sample MDP

initialize $\pi_{\theta = \omega}$

act according to $\pi \to \tau \to R_\phi(\tau)$

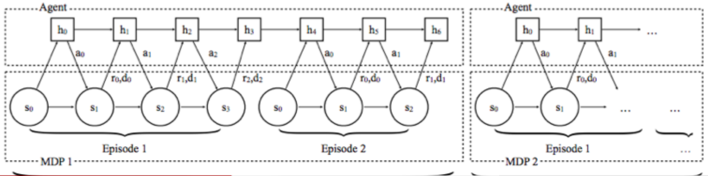
∇J_θ

Meta-learning for Reinforcement Learning

Since REINFORCE is based on gradient-based optimization of $J(\theta)$, both MAML-like and Model-based Meta-learning methods apply *directly*, as also demonstrated in those papers.

Reference - *Learning to Reinforcement Learn* and - *RL^2 Fast Reinforcement Learning via Slow Reinforcement Learning*

RL tasks are sampled from a distribution over MDPs, and are presented as sequences, as in model-based meta-learning. Instead of y_{i-1}, here a_{i-1}, r_{i-1} are passed along with s_i. First paper uses advantage AAC and second uses TRPO, to train an RNN as the deep-RL network; otherwise similar. Hidden state is re-set every time a new episode starts (first paper) / when a new MDP is sampled (second paper, better).



One gradient step

$\theta \rightarrow \theta'$] adapt

$R_{\theta'}(\tau')$

$J(w) = \sum R_{\theta'_i}$

$\nabla_w J(w)$

$R(\tau)$

s_i, r_{i-1}, a_{i-1}

Continual Learning

learn learn learn faster

$T_1, T_2, T_3 \dots T_k \quad T_{k+1}$

$L(f(T_1))$

f

Forgetting ↓

Neg. Backward Transfer ↓

Pos. Forward Transfer ↑

Continual Learning: Definition, & Gated Linear Networks

Continual learning: (a) avoid "forgetting" previously learned tasks, and also (b) improve as new tasks are learned. Online: single pass over data/tasks.

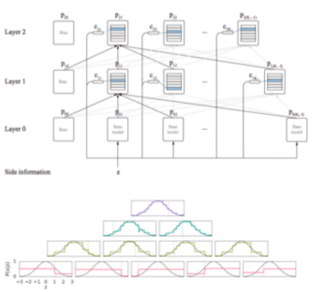
References - *Gated Linear Networks* and *A Combinatorial Perspective on Transfer Learning*

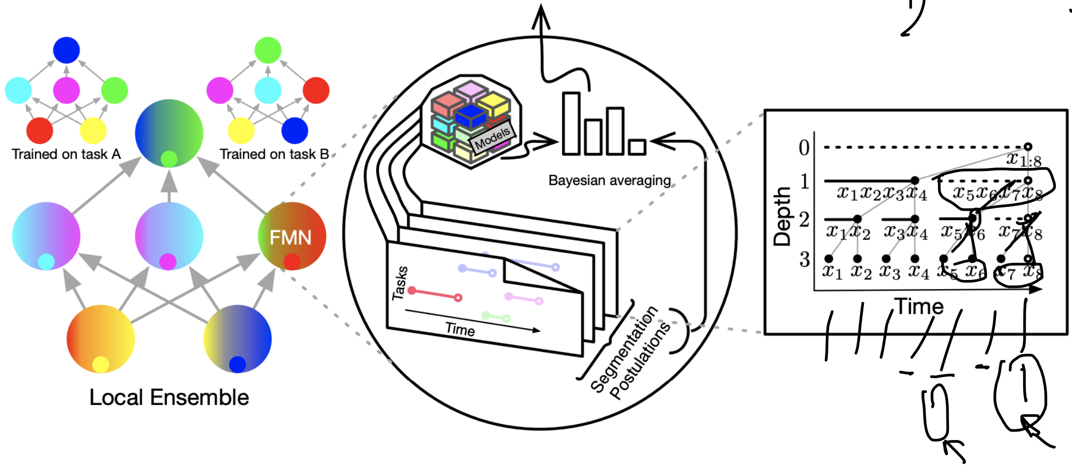
GLNs: Each neuron predicts the target directly, i.e., no back-propagation, via a **geometric mixture** of probabilities obtained from the previous layer. Lowest layer can be *random*, with say K_0 neurons. Each neuron in layer K_j has 2^d weight vectors w_{ikc} (of size K_{j-1}). Which weight vector to use is determined by a hash $c_{ik}(z)$ ('half-space gating') of the input z.

Outputs are: $p_{ik}(z) = \sigma(w^T_{ikc_{ik}(z)} \cdot \sigma^{-1}(\mathrm{p}_{i-1}(z))))$

& updated as: $\Delta w_{ikc_{ik}(z)} = \eta(p_{ik} - y)\sigma^{-1}(p_{i-a})$

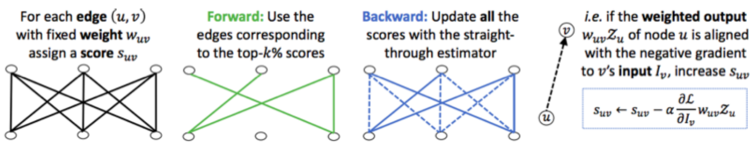
Note: $\sigma(x) = \frac{1}{1+e^{-x}}$ and $\sigma^{-1}(x) = \log \frac{x}{1-x}$





Continual Learning: Supermasks in Superposition

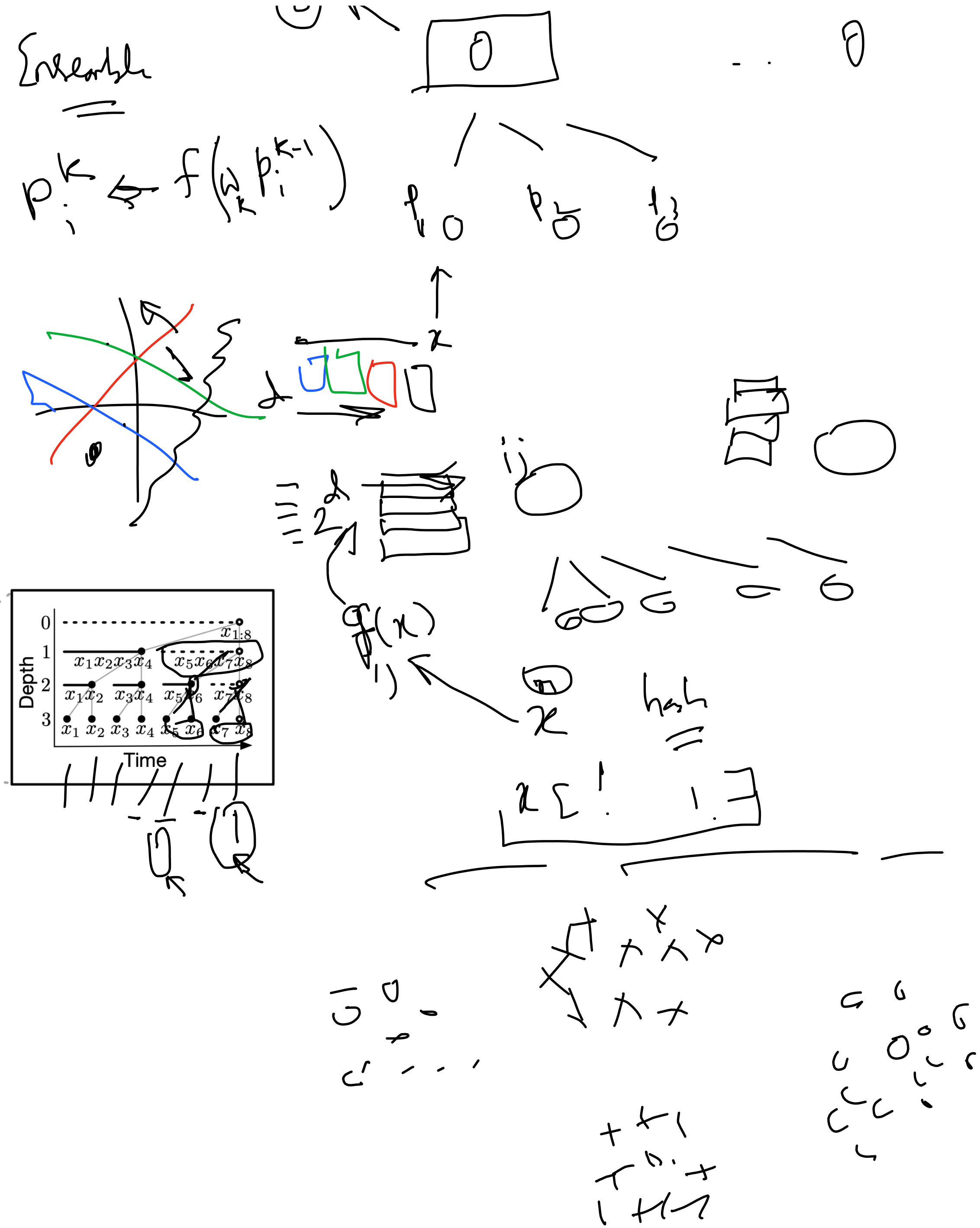
References - *Supermasks in Superposition* and *What's Hidden in a Randomly Weighted Neural Network?*



During training learn a mask M per task, using

Ensemble

$p_i^k \Leftarrow f(w_k p_i^{k-1})$



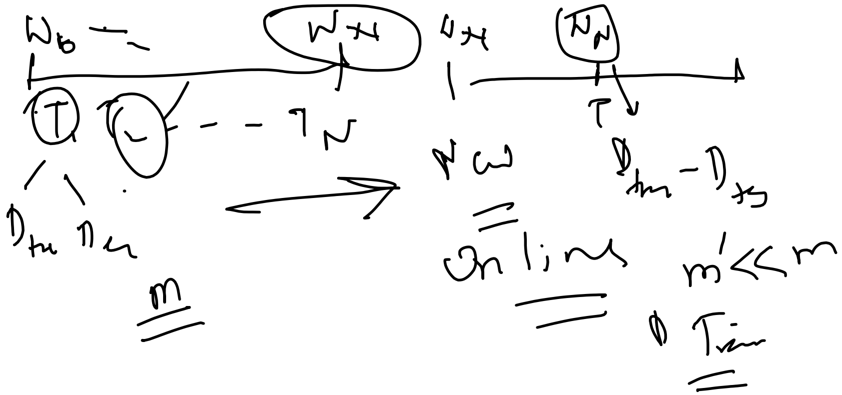
g(x)

x

hash

mask M per task, using top $k\%$ scores as above; network outputs $p = f(x, W \odot M)$.

For future tasks adapt a mixture of masks by gradient descent on **entropy**: i.e., update $p(\alpha) = f\left(W \odot \sum_i \alpha_i M^i\right)$ via $\alpha \leftarrow \alpha - \eta \nabla_\alpha \mathcal{H}(p(\alpha))$, so low entropy, i.e., less 'confused' masks are preferred.



Meta-learning for Continual Learning

Let A_i^k = training accuracy on task k after encountering i tasks, and T_i^k the 'time' to 'learn' task k. Then $CF = \sum_{i>N} \sum_{k<N} (A_k^k - A_i^k)$ is a measure of "forgetting"; and $FT = \sum_{t<N} \sum_{s>N} (T_t^t - T_s^s)$ measures "forward transfer".

In a meta-learning setup for continual learning, each task has a train and test set. The first N tasks are considered meta-training and the rest meta-testing. During meta-training, all tasks are available; in meta-testing, only the most recent task is available. If $\hat{T}_i^k$ the time to learn on the test-set for task k after seeing i tasks, then $CT_i == \sum_{k<n}(\hat{T}_i^k - \hat{T}_k^k)$ for $i > N$ also measures 'forgetting'. Further, if $\hat{A}_k^k$ is the **test**-set accuracy $G = \sum_{k>N} \hat{A}_N^k$ for $k > N$ measures generalization from meta-learning. Often, continual learning *trajectories* CT_i, $CF_i = \sum_{k<N}(A_k^k - A_i^k)$, $FT_i = \sum_{t<N}(T_t^t - T_i^i)$ and $\hat{A}_N^i$, for $i > N$ are measured.