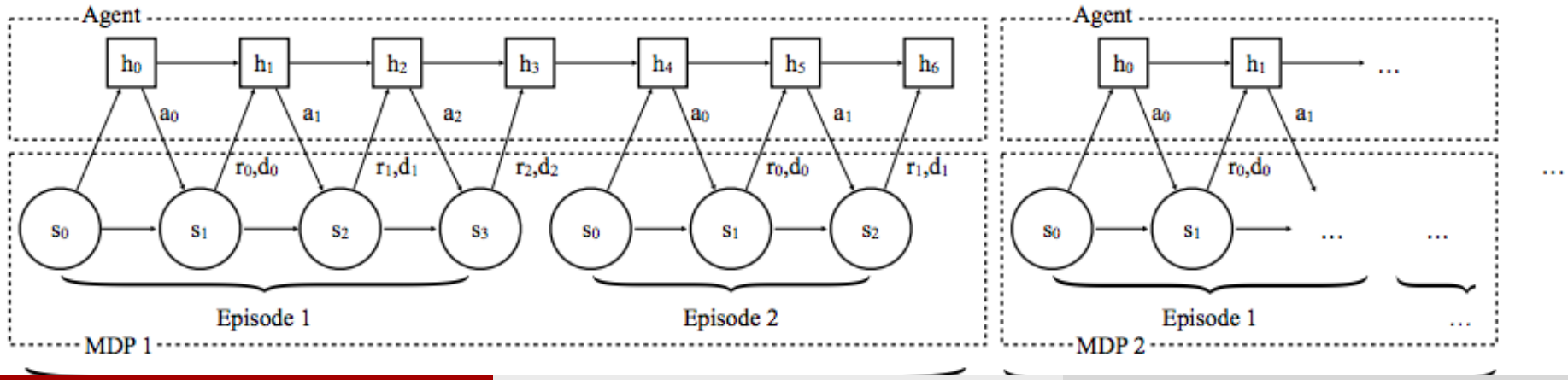
Meta-learning for Reinforcement Learning

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).



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.

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*

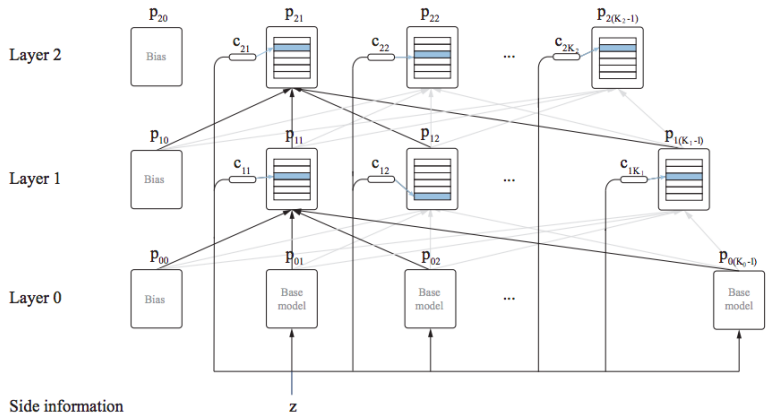
GLNs: Each neuron predicts the target directly, i.e., no back-propagation, via a **geometric mixture** of probabilities obtained from the previous layer. Lowest layer can be *random*, with say K_0 neurons. Each neuron in layer i has 2^d weight vectors w_{ikc} (of size K_{i-1}). Which weight vector to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

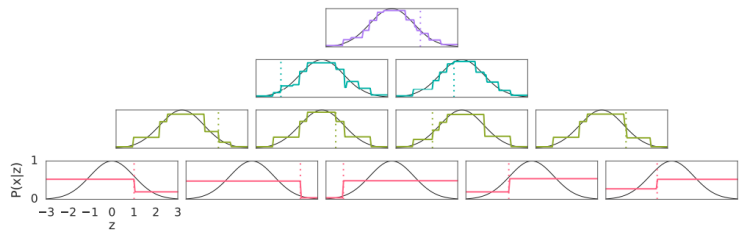
Outputs are: $p_{ik}(z) = \sigma(w_{ikc_{ik}(z)}^T \cdot \sigma^{-1}(\mathrm{p}_{i-1}(z)))$

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

For combinatorial generalization, a Bayesian mixture of previous layers' models from past tasks is chosen at each neuron.

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

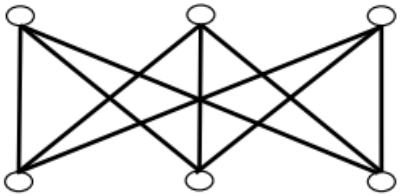




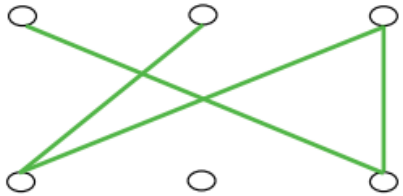
Continual Learning: Supermasks in Superposition

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

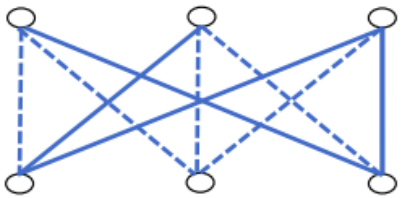
For each **edge** (u, v) with fixed **weight** w_{uv} assign a **score** s_{uv}



Forward: Use the edges corresponding to the top-k% scores



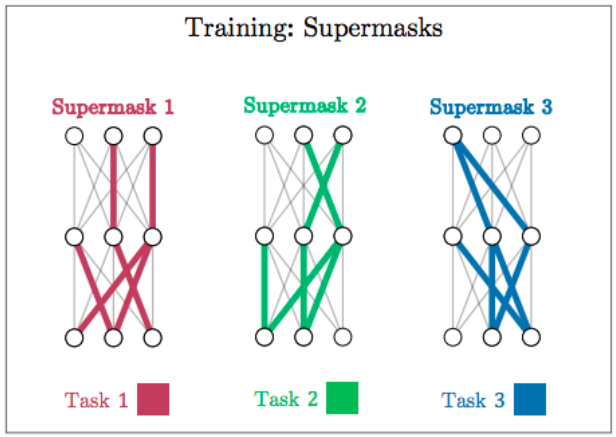
Backward: Update **all** the scores with the straight-through estimator

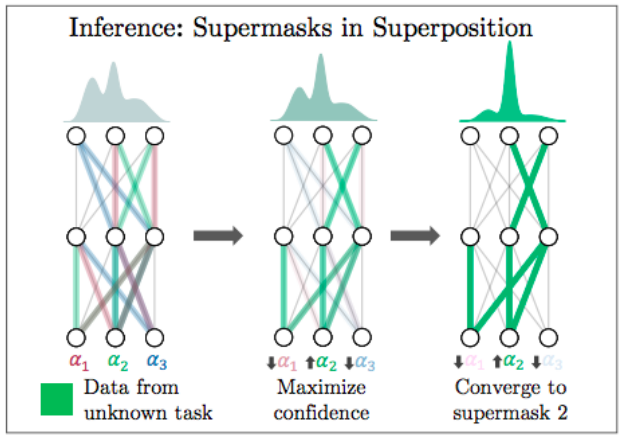


i.e. if the **weighted output** $w_{uv}z_u$ of node u is aligned with the negative gradient to v's **input** I_v, increase s_{uv}

$$s_{uv} \leftarrow s_{uv} - \alpha \frac{\partial \mathcal{L}}{\partial I_v} w_{uv} z_u$$

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





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