Meta-learning

Gautam Shroff

Spring 2021

Outline

Weeks 1 and 2 (HWs 1,2,3, Q1)

1. Meta-learning: Basic Techniques, Some Theory and Extensions

Week 3 (HW 4, Q2)

2. Meta-Learning for Transfer, Continual and Reinforcement Learning

Week 4 (Q3)

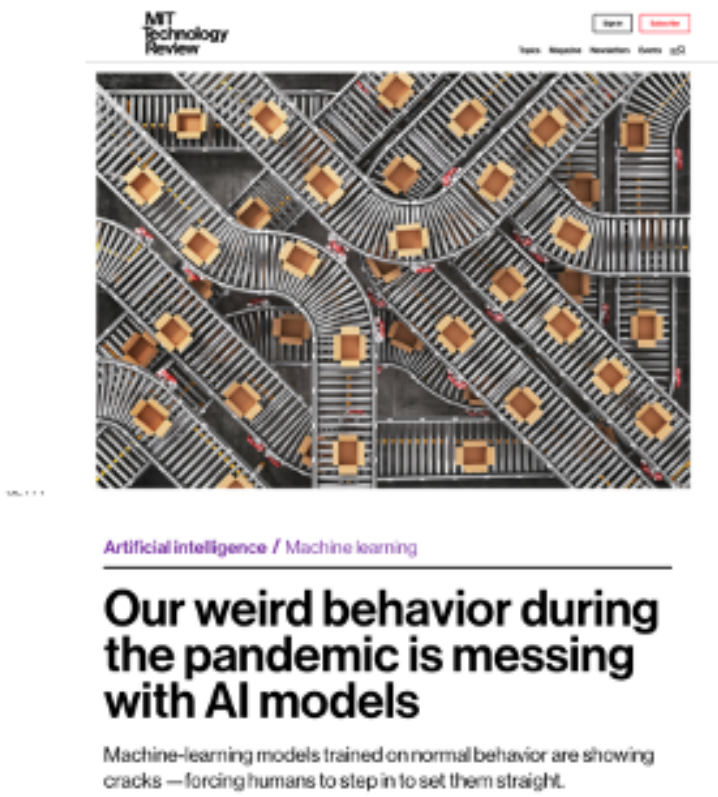
3. Learning from almost No Data
4. Selected applications of Meta-learning

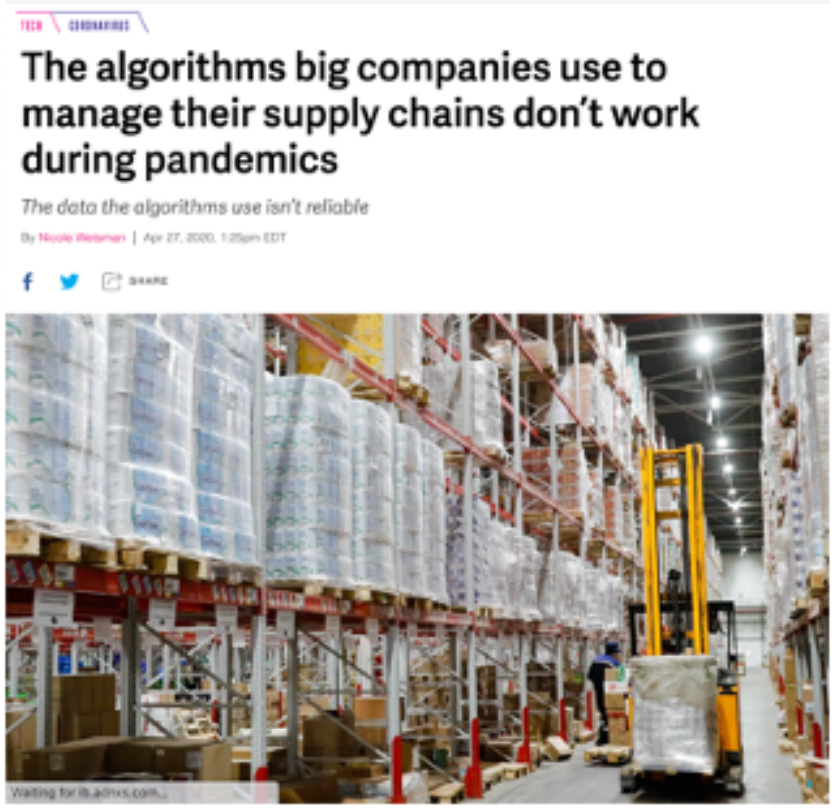
Week 5 (Q4)

5. Causality, Disentangled Representations and Causal RL

Motivation 1: Learning from Less Data

Motivation 2: A Changing World





1. Basics of Meta-learning

Machine Learning

A supervised learning *task* T is defined by the following:

Given 'training' data: $D = \{\mathrm{x}_i, y_i\}$ from some distribution $\mathcal{D}$,
'learn' a 'model' $y = f_{\mathrm{w}}(\mathrm{x})$, with parameters w that are chosen so as to:
minimise some loss $\mathcal{L}(\mathrm{w}, D)$, e.g., squared error: $\sum_i \|y_i - f_{\mathrm{w}}(\mathrm{x_i})\|^2$. or
negative log-likelihood: $\sum_i \sum_c y_i^c \log f_{\mathrm{w}}^c(\mathrm{x_i})$

Ideally the model should 'generalise well', so that the *expected loss* for 'test data' sampled from $\mathcal{D}$ is also small.

Machine-learning Examples

1. Linear regression:
$$y = f_{\mathrm{w}}(\mathrm{x}) = \mathrm{x}^T\mathrm{w} = [\mathrm{w}^T\mathrm{x}]^T, \mathrm{x}, \mathrm{w} \in \mathbb{R}^N$$

2. Linear classifier:
$$y = f_{\mathrm{W}}(\mathrm{x}) = \mathtt{softmax}([\mathrm{W}\mathrm{x}]^T), \mathrm{W} \in \mathbb{R}^{CxN}$$

2. Deep learning:
$$y = f_{\mathrm{w}}(\mathrm{x}) = [g(W_n \ldots g(W_2(g(W_1\mathrm{x}))\ldots)]^T$$

"multi-layer perceptron (MLP) with non-linearity g, e.g., $g = ReLU()$"

loss $\mathcal{L}(\{W_i\}, D)$ minimised by e.g. 'stochastic gradient descent'

gradient of *loss* w.r.t. parameters $\{W_i\}$ computed using 'back-propagation' = 'chain-rule for derivatives together with some dynamic programming to reuse intermediate results'

Meta-learning: Motivation & 'Definition'

Many, possibly different, tasks T, are experienced (by humans/machines).

What is a 'task'?
$D = \{D_{Train}, D_{Test}\} \sim \mathcal{D}$, a data distribution; and loss $\mathcal{L}$
where $D_{Train} = (X_{Train}, y_{Train})$ and $D_{Test} = (X_{Test}, y_{Test})$. So a task $T = (D, \mathcal{L})$. We assume a *distribution* $\mathcal{T}$ over such tasks.

Meta-learning, or 'learning to learn', attempts to learn from learning experiences across tasks, so as to learn 'better' on future tasks.
'Better' could mean with greater accuracy and/or with smaller amounts of data and/or faster exploration of functional forms of f.

Learning to discover better learning 'architectures' (f) is called **AutoML**
Learning across different data distributions is called **transfer learning**
Learning to learn faster and with less data is called **few-shot learning**
There are other (non meta-learning) techniques for few-shot/transfer learning: e.g. deductive learning/program synthesis

Learning from prior tasks via Pre-training

Reference - *Greedy layer-wise pre-training*

Layer-wise pre-training, e.g. train using layer 1 plus an output layer only, throw away output layer and use output of layer 1 to train layer 2 ... and so on. Target can be supervised, i.e., to predict labels, or unsupervised, i.e., to reconstruct the input (either directly or from a noisy version).

Reference - *Why Does Unsupervised Pre-training Help Deep Learning?*

Helps initialise weights to a region close to a local optima so that later training ('fine tuning') does better.

Reference - *How transferable are features in deep neural networks?*

Transferring weights from lower layers of one network to another, to see how general are the features being learned.

However, none of these actually explored meta-learning using pre-trained networks, i.e., whether training on many different datasets or tasks helps with new tasks and/or learning from less data.

Pre-training 2: time-series & NLP; how to pre-train better

Reference - *TimeNet: Pre-trained deep recurrent neural network for time series classification*

Timenet: sequence-to-sequence LSTM model trained to reconstruct diverse time-series; embeddings used as features to train SVM. Classifiers using Timenet embeddings beat traditional and deep baseline classifiers.

Reference - *ConvTimeNet: A Pre-trained Deep Convolutional Neural Network for Time Series Classification*

Conv-Timenet: convolutional (in-time) network trained on diverse classification tasks transfers to new data (with fine-tuning) beating baselines as well as itself trained from scratch on target dataset.

Reference - *Improving Language Understanding by Generative Pre-Training*

Transformer network trained for *language modelling*; pre-trained layers used for a variety of NLP tasks and beating baselines.

Reference - *Domain Adaptive Transfer Learning with Specialist Models*

Selecting data to pre-train on using importance sampling improves over using all data for pre-training.

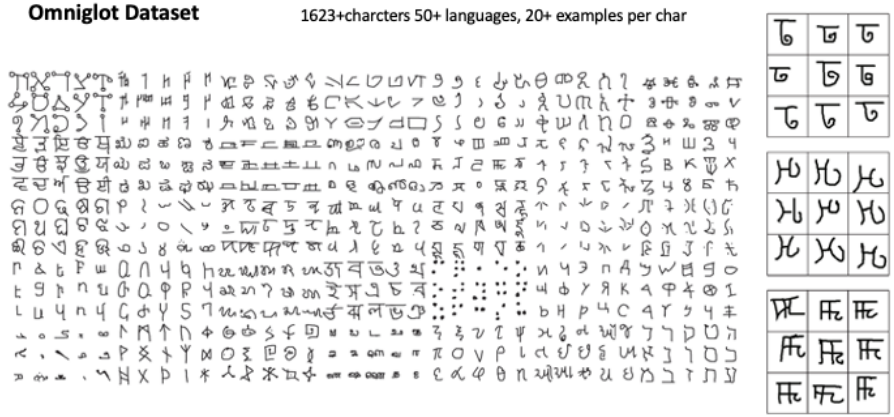
Early work in meta-learning and few-shot learning

Reference - *Schimduber's "Godel Machine"*

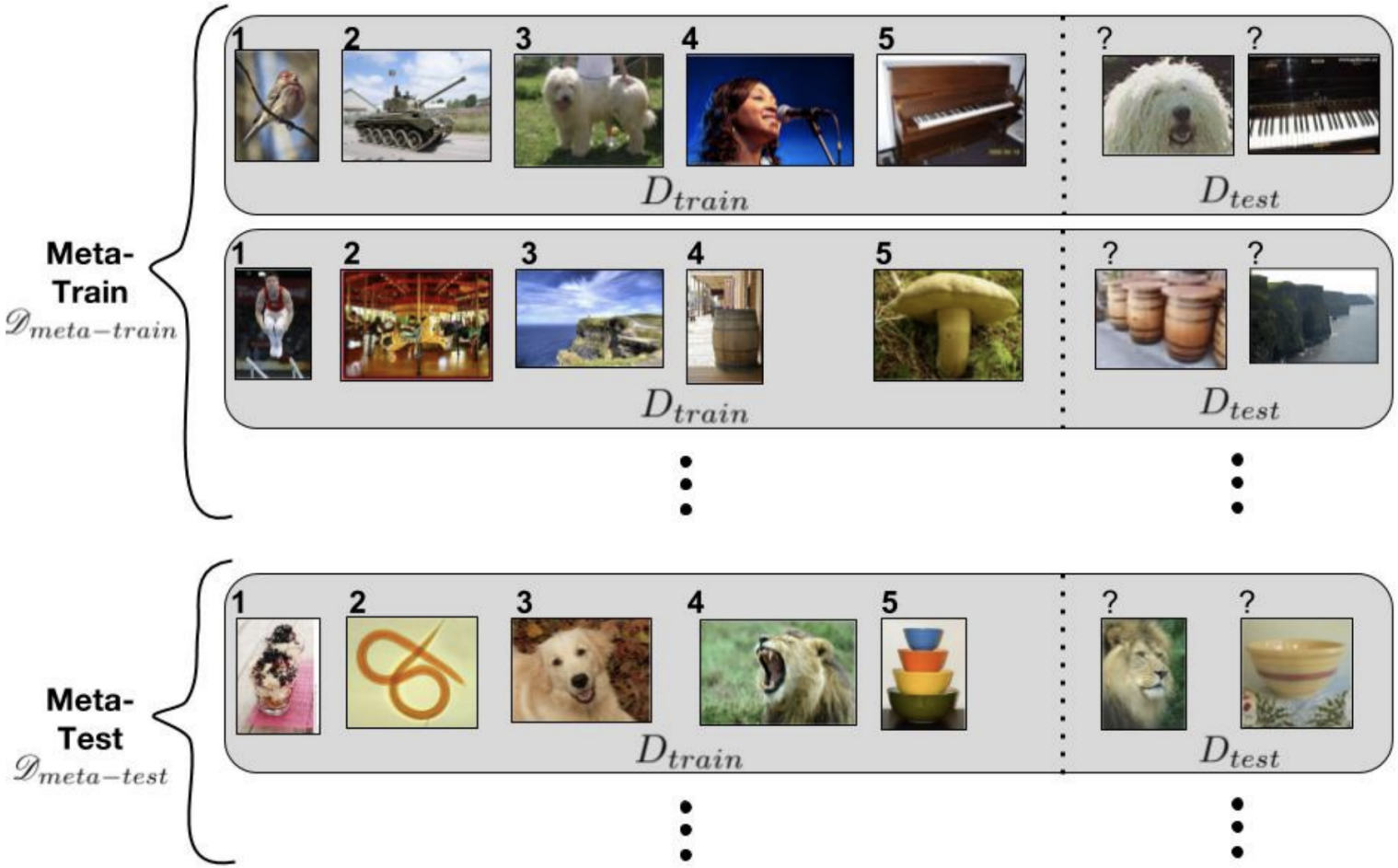
Architecture for a general-purpose meta-learner, where the inner learner could be a neural network or even a Turing machine. Key idea of optimisation across tasks introduced.

Reference - *Siamese Neural Networks for One-shot Image Recognition*

Use a 'siamese' network to learn a distance metric between images, and use it for few-shot nearest-neighbour classification.



Definitions: via Few-shot Learning



Key Idea in Meta-learning

Machine learning via empirical risk minimisation:

$$\mathrm{w} = \mathrm{argmin}_{\mathrm{w}}\, \mathbb{E}_{D_{Train} \sim \mathcal{D}} \mathcal{L}(f_{\mathrm{w}}, D_{Train})$$

and *hope* that

$$\mathbb{E}_{D_{Test} \sim \mathcal{D}} \mathcal{L}(f_{\mathrm{w}}, D_{Test}) \text{ is also small.}$$

Loss $\mathcal{L}$ depends on **problem**: classification NLL, regression MSE, RL ...

Key idea in meta-learning:

If the objective is to minimise $\mathbb{E}_{D_{Test} \sim \mathcal{D}} \mathcal{L}(f_{\mathrm{w}}, D_{Test})$,

Why not minimise it directly?

Instead of $\bar{y} \leftarrow f_{\mathrm{w}}(x)$, use $f : \underline{\bar{y} \leftarrow f_{\mathrm{w}}(x, D_{Train})}$.

Generic Meta-learning Algorithm

Given task distribution $\mathcal{T}$, meta-learner f_{w}

for $e \leftarrow 1$ to N_{epochs} **do**
  for $i \leftarrow 1$ to N_{steps} **do**
    Sample $\{D_{Train}, D_{Test}\} \sim \mathcal{T}$
    $\Delta_i = \mathcal{L}(f_{\mathrm{w}}(D_{Test}, D_{Train}))$
  end
  $\mathrm{w} \leftarrow \mathrm{w} - \eta \nabla \sum_i \Delta_i$
end

Algorithm 1: MetaLearn

Different kinds of Meta-learners

1. **Optimization-based** meta-learners:
   f_w *adapts* a base learner g_w.

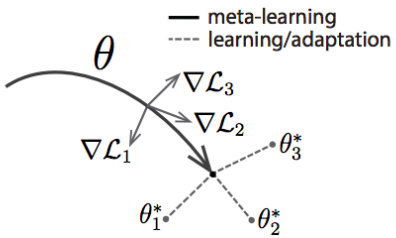
2. **Model-based** meta-learners:
   f_w is a (e.g., recurrent) network taking D_{Train} as input.

3. **Metric-based** meta-learners:
   f_w computes (learned) distances to elements of D_{Train}.

Optimisation-based Meta-learning

Reference - *Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks*

Choose model parameters w such that taking one or few gradient-steps on an unknown task (i.e., dataset) is maximally optimal: For each task, *adapt* $g_{\theta_0=\mathrm{w}}$ via gradient-step(s) using *test* loss on tasks, i.e., $\theta - \alpha\nabla_\theta\mathcal{L}_{\mathcal{T}_i}(g_\theta)$, over a number of tasks $\mathcal{T}_i$ sampled from a distribution. Note: f_w requires similar *adaptation* at (meta)-test time also. Also, as formulated MAML applies both to supervised as well as other ML tasks, e.g. reinforcement learning. Thus:



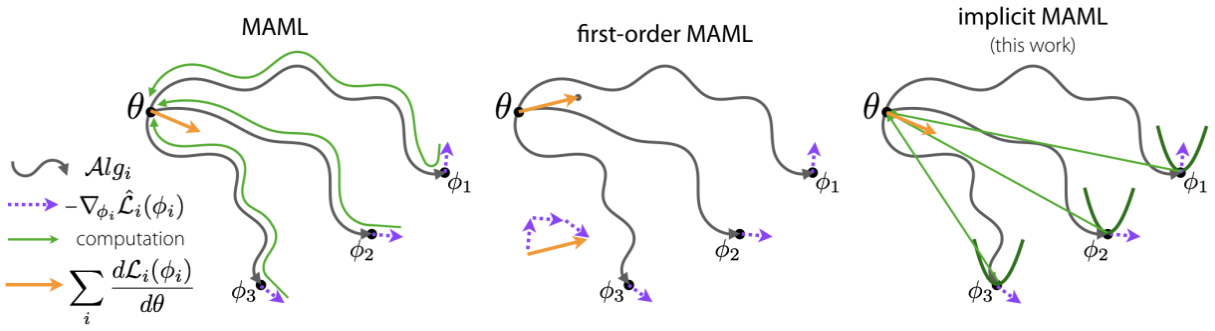
$$f_\mathrm{w}^{MAML}(x, D_{Train}, g) = g_{\mathrm{w}-\alpha\nabla\mathcal{L}(g_\mathrm{w}(D_{Train}))}(x) \equiv g_\phi(x)$$

In practice, more than one gradient steps are taken: *fast adaptation*.
Note that training f^{MAML}, i.e., optimizing for w across meta-training tasks, requires second-order derivatives, i.e., we need $\nabla_w\mathcal{L}(g_{\hat{\phi}}, D_{Test})$
$= (I - \alpha\nabla_w^2\mathcal{L}(g_w(D_{Train}))\nabla_\phi\mathcal{L}(g_\phi, D_{Test})|_{\phi=\hat{\phi}}$, where $\hat{\phi} = \mathrm{w} - \alpha\nabla\mathcal{L}(g_\mathrm{w}(D_{Train}))$

MAML, FO-MAML, and iMAML (Implicit layers)

References - *On First-Order Meta-Learning Algorithms* and *Meta-Learning with Implicit Gradients*

FO-MAML: $\mathrm{w} \leftarrow \mathrm{w} - \eta \sum_i \delta \hat{\mathrm{w}}_i$



iMAML: fully we minimize $G(\phi, \mathrm{w}) = \hat{\mathcal{L}}(\phi) + \frac{1}{2}\|\phi - \mathrm{w}\|^2$ fully, where $\hat{\mathcal{L}}$ denotes loss on D_{Train}. Let $\phi^*(\mathrm{w}) = \mathrm{argmin}_\phi\, G(\phi, \mathrm{w})$. For updating w we need $\nabla_{\mathrm{w}}\mathcal{L}(\phi^*) = \mathrm{d}_{\mathrm{w}}\phi^* \nabla_\phi \mathcal{L}(g_\phi)|_{\phi=\phi^*}$. To compute $\mathrm{d}_{\mathrm{w}}\phi^*$:
$\frac{dG}{d\phi} = \nabla_\phi \hat{\mathcal{L}}(\phi) + (\phi - \mathrm{w}) = 0$ at ϕ^* so $\phi^* = \mathrm{w} - \nabla_\phi \hat{\mathcal{L}}(\phi)|_{\phi=\phi*}$ Thus,

$$\frac{d\phi^*}{d\mathrm{w}} = \left(I + \nabla^2_\phi \hat{\mathcal{L}}(\phi)|_{\phi=\phi^*}\right)^{-1}$$

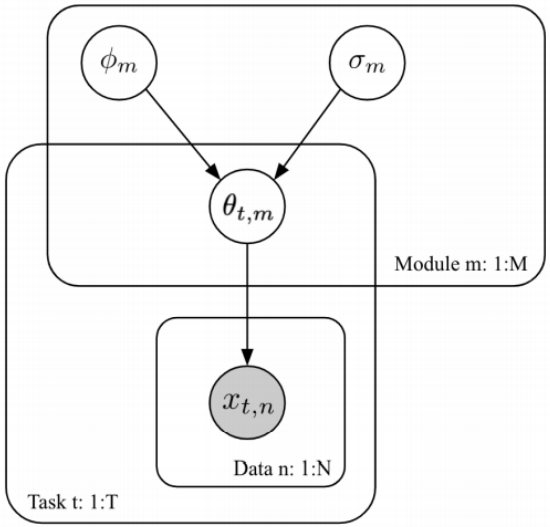
$$\mathrm{w} \leftarrow \mathrm{w} - \eta \sum_i \left(I + \nabla^2_\phi \hat{\mathcal{L}}_i(\phi)|_{\phi=\phi_i^*}\right)^{-1} \nabla_\phi \mathcal{L}_i(\phi)|_{\phi=\phi_i^*}$$

Modular Meta-learning: Variants of MAML

Reference - *Modular Meta-Learning with Shrinkage*

In general $\mathrm{w} = \{\theta_1 \dots \theta_M\}$ e.g., different layers of a network. Variants of MAML learn a prior for w that is adapted for each task; but do all layers need to adapt? E.g. if only one layer is adapted, perhaps it could be trained for many more steps per task without risk of over-fitting. This paper learns to *differently* adapt each layer: assuming each θ_m is normally distributed as $\mathcal{N}(\phi_m, \sigma_m^2)$. **Layers with small or zero σ_m^2 will not adapt.** To learn ϕ, σ^2 we take Bayesian view:

$$p(\mathrm{w}^{1:T}, \mathcal{D}|\phi, \sigma^2) = \prod_{t=1}^{T} \prod_{m=1}^{M} \mathcal{N}(\theta_m^t|\phi_m, \sigma_m^2) \prod_{t=1}^{T} p(\mathcal{D}_t|w_t)$$



using the MAML approach to update ϕ, σ^2:
the inner loop computes:

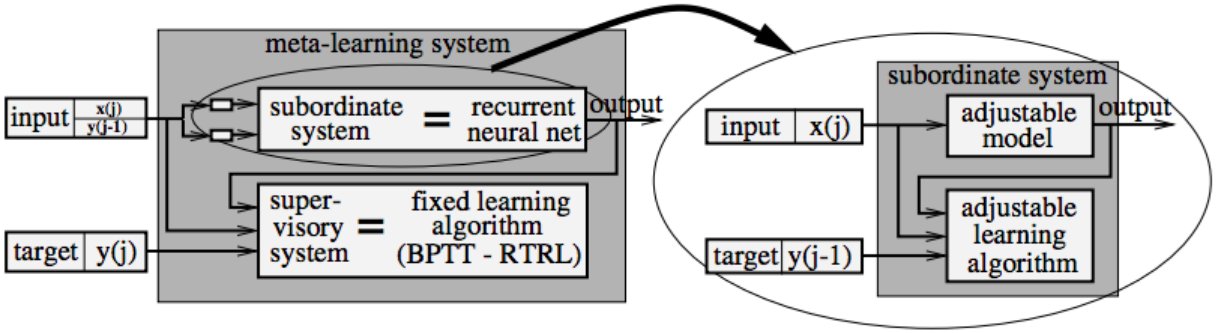
$$\hat{\theta}^t(\phi, \sigma^2) \equiv \underset{\mathrm{w}^t}{\mathrm{argmin}} \left[-\log p(\mathcal{D}_t^{Train}|\mathrm{w}^t) - \log p(\mathrm{w}^t|\phi, \sigma^2) \right]$$

& the outer loop minimizes $\frac{1}{T} \sum_{t=1}^{T} -\log p(\mathcal{D}_t^{Test}|\hat{\mathrm{w}}^t)$.

Model-based Meta-learning

Reference - *Learning to Learn using Gradient Descent*

Early paper that introduces the idea of inputting a *dataset* to a recurrent network as a sequence with labels. A 'supervisory procedure' - gradient descent - is used to train such a network: basically, an LSTM is used as a meta-learner taking datasets $D_k = \{(x_1^k, y_0^k) \dots (x_n^k, y_{n-1}^k)\}$ as input with targets $\{y_1^k \dots y_n^k\}$ The network learns to predict y_{n+1} for any new unlabeled labeled input $(x_{n+1}, \_)$



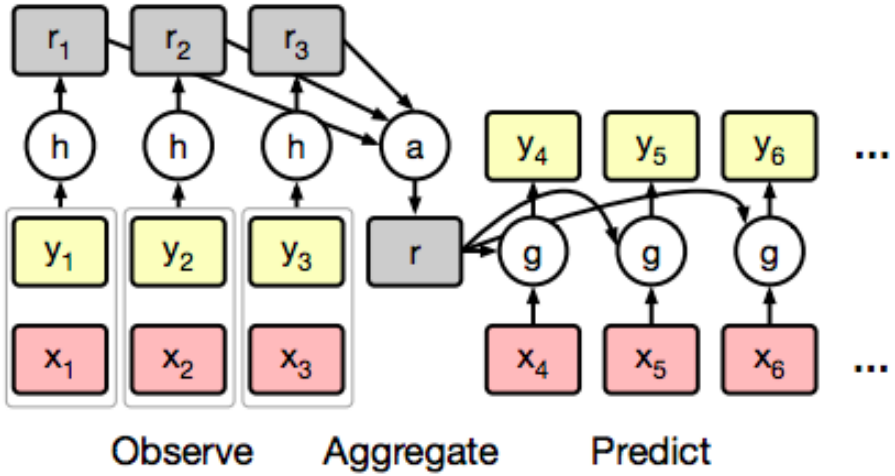
$h_t, c_t = LSTM([x_t, y_{t-1}], h_{t-1}, c_{t-1})$. Predict $y_i = g(x_i, LSTM([x_i, y_k], h_k, c_k))$, $k = i - 1$ for $x_i \in D_{Train}$, and $k = n$ for $x_i \in D_{Test}$

Update parameters of *LSTM and g* using gradient of loss over D_{Train}*

Model-based Meta-learning (cont) 1

Reference - *Contitional Neural Processes*

Training examples are passed through an MLP to generate representations; class-specific representations are aggregated and passed to a second classification MLP concatenated with query examples (from both test and train). Entire network is trained end-to-end on multiple tasks. Recent paper and seems to beat many baselines.

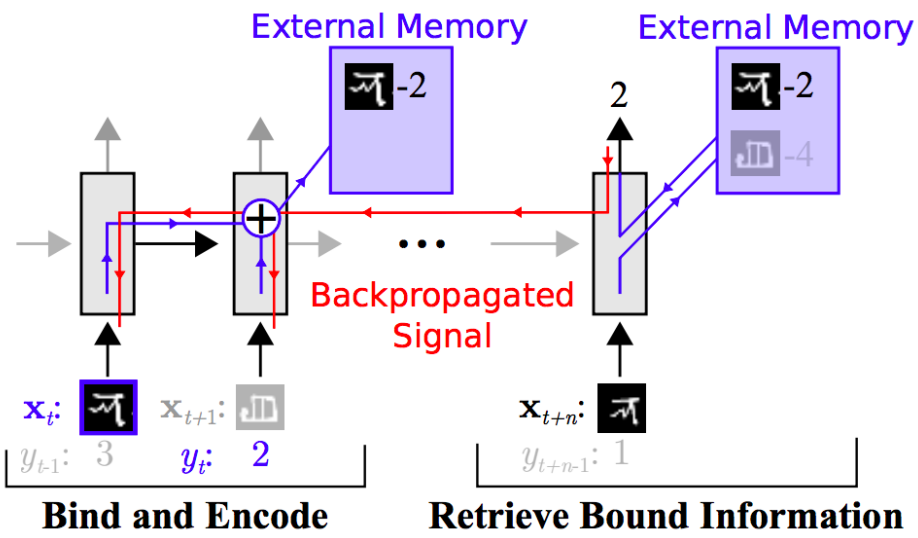


Model-based Meta-learning (cont) 2

Reference - *Meta-Learning with Memory-Augmented Neural Networks* *Theano reference implementation*

Datasets are presented as sequences, $\{(x_t, y_{t-1})\}$ as in the RNN-based approach.
A 'memory-augmented LSTM/FF network' is the meta-learner.

Network updates rows of a memory matrix M with read/write keys
$\mathrm{k}^{r/w} = \phi^{r/w}(z)^T$: ($z$ is 'cell state')
$M_t = M_{t-1} + \mathrm{w}^w \mathrm{k}_t^w$; w^w = write weights
Retrieved memory $r_t = \mathrm{w}^r M_t$ used to predict $\hat{y}_t$ using a feedforward layer.
(read weights $\mathrm{w}^r = Softmax(M_t \cdot k_t^r)$)
usage weights: $\mathrm{w}_t^u = \gamma \mathrm{w}_{t-1}^u + \mathrm{w}_t^r + \mathrm{w}_t^w$
used to compute w^w as follows:



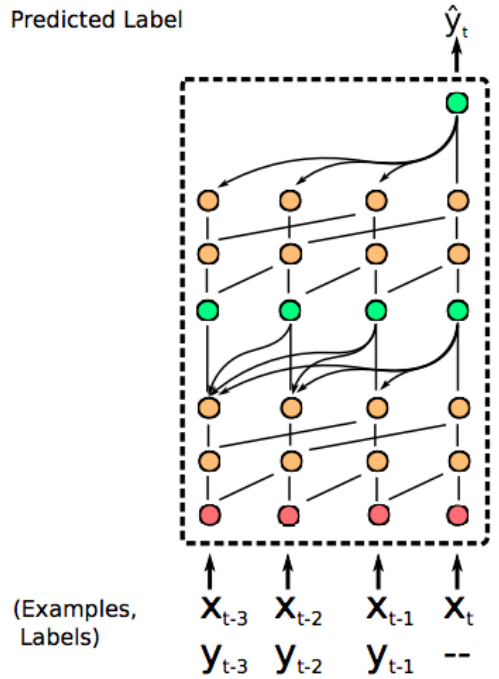
'least used' $w^{lu} = 1$ for t smallest elements of w^u, 0 otherwise
$\mathrm{w}_t^w = \delta w_{t-1}^r + (1-\delta)\mathrm{w}_{t-1}^{lu}$; prior to writing the least used row of M is zeroed.
Read/write to 'memory' - we can view this as a 'neural Turing machine'.

Model-based Meta-learning (cont) 3

Reference - *A Simple Neural Attentive Meta-learner*

Similar to Hochreiter's work above, i.e., dataset is passed as sequence but uses 1D convolutions and attention layers instead of an LSTM. Note: final prediction $y_{\texttt{test}}$ alone is used with target y to backpropagate errors, so train dataset is passed as $z^0 = \{(x_1, y_1) \ldots (x_{t-a}, y_{t-1}), (x_t, \_)\} \in \mathbb{R}^{d \times t}$ for any query x_t. Both train and test data for each task are used as queries for training.

Mix of 1D convolution $C()$ and attention $A()$ layers:
$u = \texttt{1Dconv}(z, 2^i); a = \tanh(u) * \sigma(u); C(z) = (z, a)$
$k = W_k z + b_k, q = W_q z + b_q, v = W_v z + b_v$
$p = Softmax(\frac{qk^T}{\sqrt{d}}); A(z) = (z, vp^T)$ $z_{in} \in \mathbb{R}^{d \times t}$
Note: convolutions and softmax are *causal*.
Using convolutions instead of RNNs is faster and easier to train. Attention is able to 'retrieve' inputs in position-independent manner.



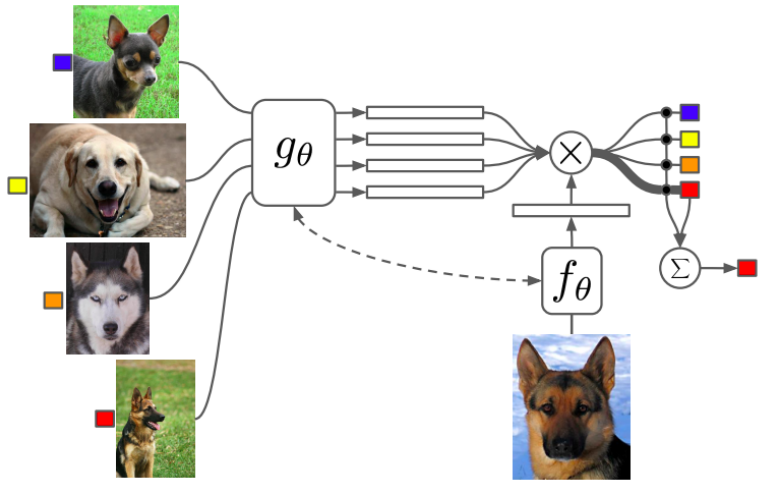
Metric-based Meta-learning

Reference - *Matching Networks for One Shot Learning*

First to introduce the training procedure for few-shot learning, i.e. sampling varieties of few-shot tasks and using test accuracies as an optimisation objective. The network computes a distance kernel ('attention kernel') combining LSTM and attention mechanisms over a given few-shot training set; thus the classifier (based on distance kernel) is 'non-parametric' in that it changes with training set.

$$\hat{y} = \sum_{i=1}^{k} a(\hat{x}, x_i) y_i \text{ where}$$
$$a(\hat{x}, x_i) = \frac{e^{c[f_\theta(\hat{x}), g_\theta(x_i)]}}{\sum_{i=1}^{k} e^{c[f_\theta(\hat{x}), g_\theta(x_i))]}}$$

Simple case: f, g are MLPs;
full-context case:
$g(x_i, S)$ takes training examples as input and feeds into $f(\hat{(}x), g(S))$.

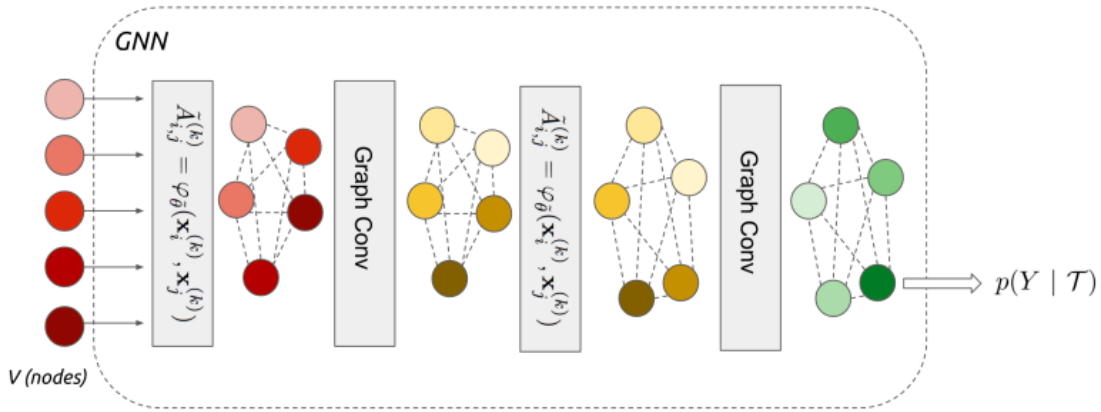


Metric-based Meta-learning (cont)

Reference - *Few-shot Learning with Graph Neural Networks*

$D_{Train} = (\{(x_i, y_i)\}), \{\tilde{x}_j\}; D_{Test} = \{\bar{x}_l\}$. D_{Train} has unlabeled examples $\tilde{x}_j$ for semi-supervised learning; y_is one-hot, and $\tilde{y}(), \bar{y}() = \frac{1}{C}$. D_{Train}, D_{Test} fed to GNN,

GNN is on a FC graph
$G(\{\mathrm{x} \in \mathbb{R}^d\} = \{(x, y)\}, \varphi)$
$\varphi^k(\mathrm{x}_i, \mathrm{x}_j) = MLP_{\theta^k}(|\mathrm{x}_i - \mathrm{x}_j|)$
GC: aggregate from neighbors & concatenate; $W^k, V^k \in \mathbb{R}^{d^{k+1} \times d^k}$:
$\mathrm{x}_i^{k+1} = \sum_{j \neq i} \varphi_{ij}^k(W^k \mathrm{x}_j, V^k \mathrm{x}_i)$

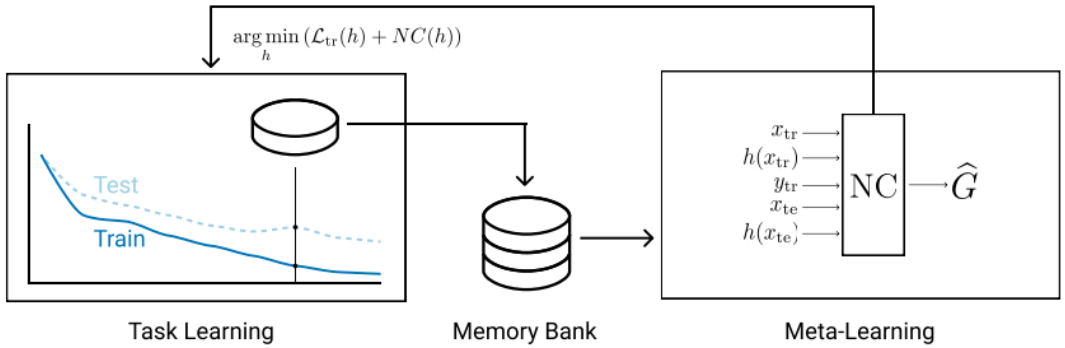


Finally $p(y|\mathcal{T}) = Softmax(\mathrm{x}^N)$ for all nodes, test and training. Network is trained using available training and test labels across many tasks - each task's data D_{Train}, D_{Test} is input as a graph, with possibly different number of examples. This is a mix of metric-based (due to φs) and model-based approaches.

Meta-learning to Predict Generalization

Reference - *Neural Complexity Measures*

During meta-learning tasks include train and test data: the gap between train/test loss is available. NC trains another network to predict this generalization gap

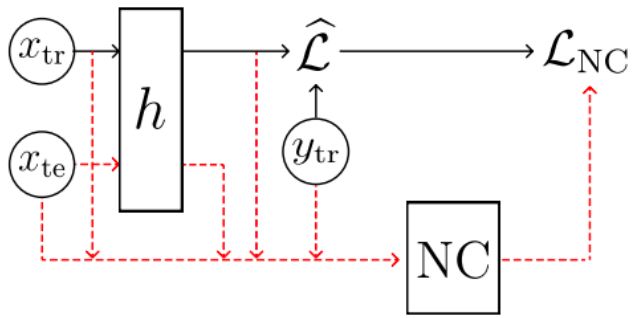


$$\mathcal{L}_{NC} \propto \|\mathcal{L}_{Test} - \mathcal{L}_{Train} - NC(H)\|$$

$$Q = f(X_{te}), K = f(X_{tr}), V = [K; Y_{tr}]$$

$$NC(H) \equiv NC(X_{tr}, X_{te}, Y_{tr}, h(X_{tr}), h(X_{te})) =$$

$$\frac{1}{m'} \sum_{i=1}^{m'} g(A); \text{ where } A = \frac{Softmax(QK^T)}{\sqrt{d}} V$$



NC-regularized loss: $\mathcal{L}_{reg} = \mathcal{L}_{Train} + \lambda NC(H)$; λ increased gradually over tasks.

2. Meta-Learning for Transfer, Continual learning and Reinforcement Learning

Transfer Learning: Domain Generalization/Adaptation & FSL

In transfer learning, distributions $\mathcal{D}_{Train} = (\mathcal{X}, \mathcal{Y})$ and $\mathcal{D}_{Test} = (\hat{\mathcal{X}}, \hat{\mathcal{Y}})$ can in general be from different *spaces*, e.g., images and text, etc., i.e., $\hat{\mathcal{X}} \neq \mathcal{X}$ or different goals, e.g., question-answering vs translation, i.e., $\hat{\mathcal{Y}} \neq \mathcal{Y}$.

In Domain Generalization, the spaces are the same but the distributions may differ: e.g. domain-shift where $p(\hat{x}) \neq p(x)$ (different styles of images - cartoon, photo, sketch etc.) or concept-shift where $p(\hat{y}|\hat{x}) \neq p(y|x)$ (changing customer behaviour) or class-incremental learning $p(\hat{y}) \neq p(y)$.

In Domain Adaptation: $D_{Train} \sim (\mathcal{D}_{Train}, \mathcal{D}_{Test})$, i.e., includes a few samples from the target distribution also.

Few-shot learning can be viewed as another special-case: when classes differ between meta-training and meta-testing so $p(\hat{y}) \neq p(y)$ (due to unseen classes), and/or $p(\hat{y}|\hat{x}) \neq p(y|x)$ (due to label re-mapping).

Domain Generalization vs Few-shot vs Continual Learning

1. Learning:
$$\mathrm{w} = \mathrm{argmin}_{\mathrm{w}}\, \mathbb{E}_{D_{Test}, D_{Train} \sim \mathcal{D}} \mathcal{L}(f_{\mathrm{w}}, D_{Test}, D_{Train})$$

2 Domain Generalization:
$$\mathrm{w} = \mathrm{argmin}_{\mathrm{w}}\, \mathbb{E}_{D_{Test} \sim \mathcal{D}_2, D_{Train} \sim \mathcal{D}_1} \mathcal{L}(f_{\mathrm{w}}, D_{Test}, D_{Train})$$

3. Few-shot Learning:
$$\mathrm{w} = \mathrm{argmin}_{\mathrm{w}}\, \mathbb{E}_{\mathcal{D} \sim \mathcal{T}} \mathbb{E}_{D_{Test}, D_{Train} \sim \mathcal{D}, |D_{Train}| \ll \delta} \mathcal{L}(f_{\mathrm{w}}, D_{Test}, D_{Train})$$

4. Continual Learning:
$$\mathrm{w} = \mathrm{argmin}_{\mathrm{w}}\, \sum_i \mathbb{E}_{\mathcal{D}^i \sim \mathcal{T}} \mathbb{E}_{D^i_{Test}, D^i_{Train} \sim \mathcal{D}^i} \mathcal{L}(f_{\mathrm{w}}, D^i_{Test}, D^i_{Train})$$

Meta-learning for Domain Generalization

Example: Classes = {'dog', 'elephant', 'giraffe', 'guitar', 'house', 'horse', 'person'} and Domains = { 'Photo', 'Art painting', 'Cartoon', 'Sketch'}

Reference - *Learning to Generalize: Meta-Learning for Domain Generalization*

Applies MAML idea to domain adaptation: Sample from train and test domains. Compute loss on train, take a gradient step to improve this, compute loss on test. Update initial parameters using gradient of linear combination of both these losses over many such iterations.

Sample D_{Train}, D_{Test} data from a set of train and test *domains*.

inner update: $\tilde{\theta} = \theta - \nabla_\theta \mathcal{L}(\theta, D_{Train})$

outer update: $\Delta\theta = -\eta \nabla_\theta \left[\mathcal{L}(\theta, D_{Train}) + \mathcal{L}(\tilde{\theta}, D_{Test}) \right]$

Note that unlike in vanilla MAML, D_{Train} and D_{Test} are from different domains, so the inner update will not learn on the training domains; hence we include the loss on D_{Train} in the outer update also.

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:

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

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

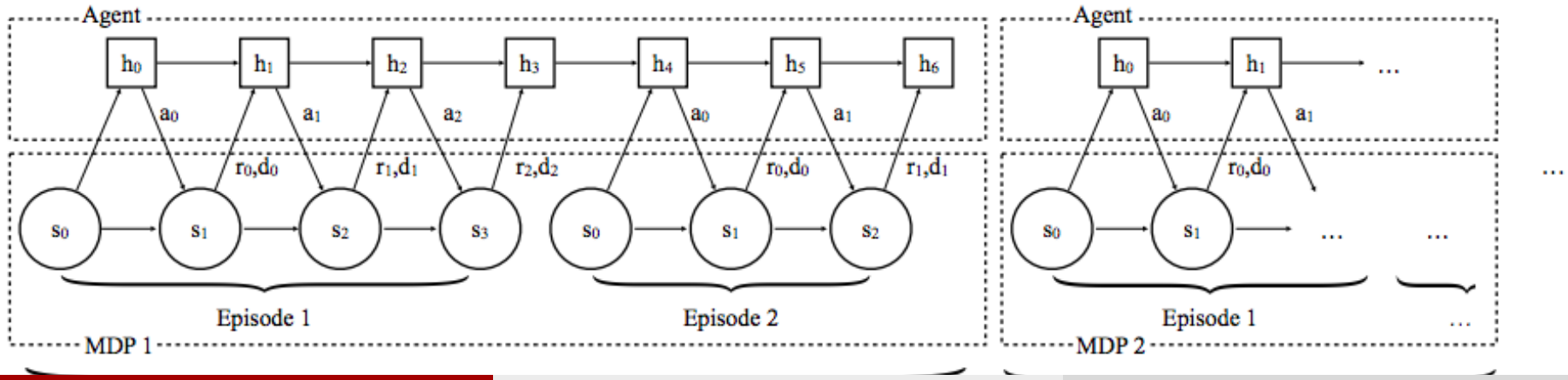
'REINFORCE': follow policy π_θ recording $r(\tau)$; update θ using $\nabla_\theta J(\theta)$.
Note: $\nabla_\theta \log \pi(\tau) = \sum_i \nabla_\theta \log \pi(a_i|s_i)$: available if π_θ is a NN.

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*

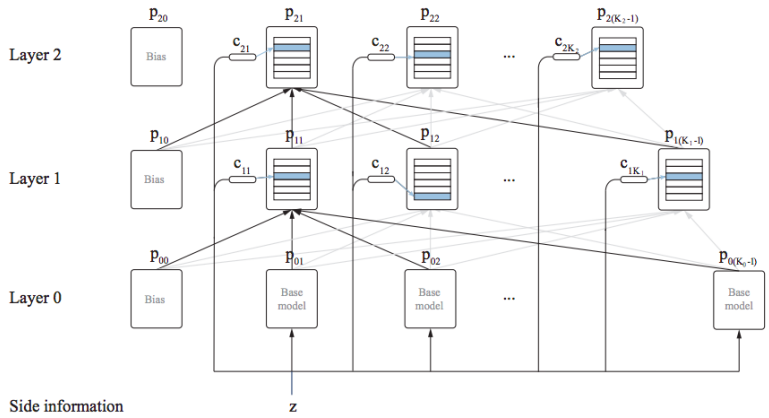
GLNs: Each neuron predicts the target directly, i.e., no back-propagation, via a **geometric mixture** of probabilities obtained from the previous layer. Lowest layer can be *random*, with say K_0 neurons. Each neuron in layer i has 2^d weight vectors w_{ikc} (of size K_{i-1}). Which weight vector to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

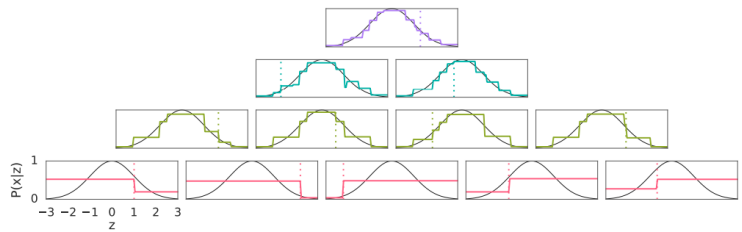
Outputs are: $p_{ik}(z) = \sigma(w_{ikc_{ik}(z)}^T \cdot \sigma^{-1}(\mathrm{p}_{i-1}(z)))$

& 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 $\sigma^{-1}(x) = \log \frac{x}{1-x}$

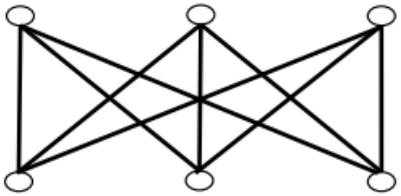




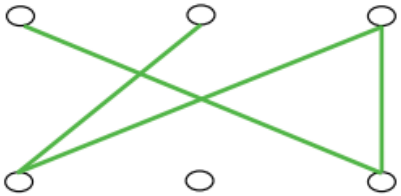
Continual Learning: Supermasks in Superposition

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

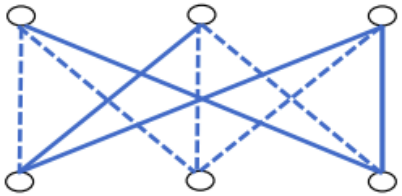
For each **edge** (u, v) with fixed **weight** w_{uv} assign a **score** s_{uv}



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



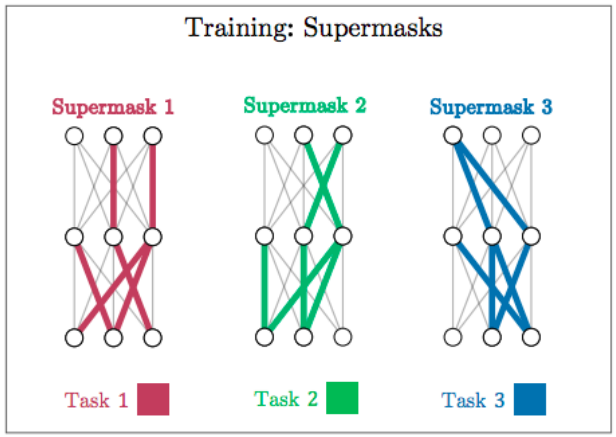
Backward: Update **all** the scores with the straight-through estimator

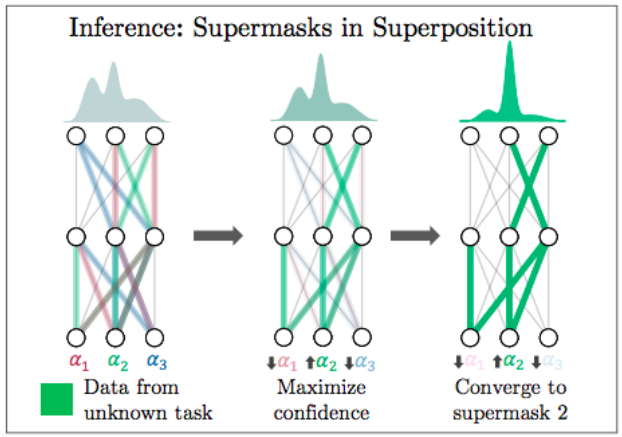


i.e. if the **weighted output** $w_{uv}z_u$ of node u is aligned with the negative gradient to v's **input** I_v, increase s_{uv}

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

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



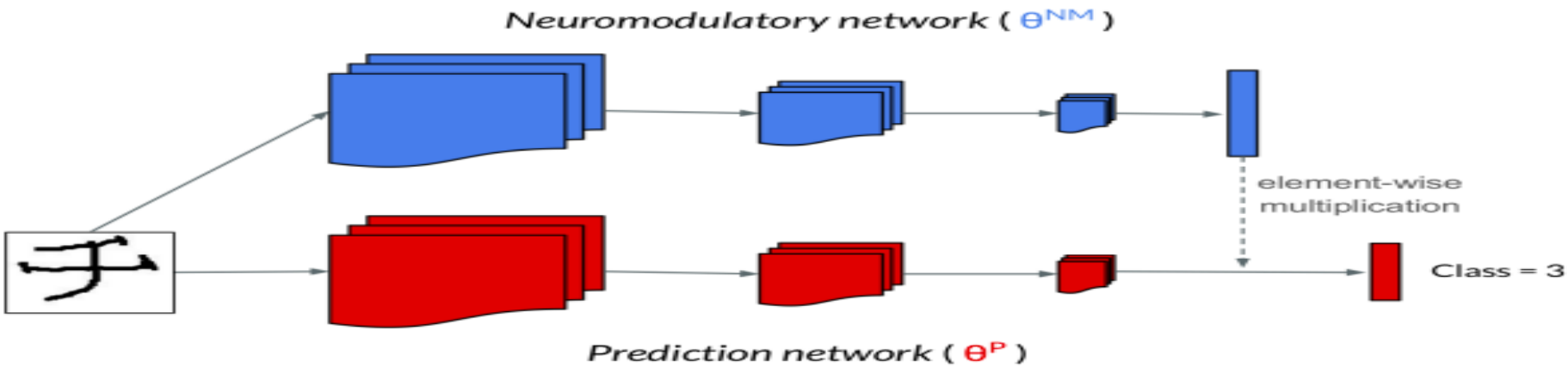


For future tasks adapt a mixture of masks by gradient descent on **entropy**: i.e., update $p(\alpha) = f\left(W \bigodot \sum_i \alpha_i M^i\right)$ via $\alpha \leftarrow \alpha - \eta \nabla_\alpha \mathcal{H}(p(\alpha))$, so low entropy, i.e., less 'confused' masks are preferred.

Continual Learning: Neuro-modulated Meta-learning

Reference - *Learning to Continually Learn*

(i) Outer-loop loss on meta-train test set and samples D_R from tasks seen so far
(ii) Multiplicative modulation of prediction network by a modulatory network.



Meta-training:

Sample D_{Train}, D_{Test}, D_R; $\theta_0^P = \theta^P$

for $i \in 0 \ldots k$,

$$\Delta\theta_i^P = -\beta \nabla_{\theta_i^P} \mathcal{L}(\theta^{NM}, \theta_i^P, D_{Train})$$

$$\Delta\theta^{NM,P} = -\alpha \nabla_{\theta^{NM,P}} \mathcal{L}(\theta^{NM}, \theta_k^P, D_{Test}, D_R)$$

Meta-testing:

$\mathcal{T}_{Test} -= (D_{Train}, D_{Test}) \sim \mathcal{T}$

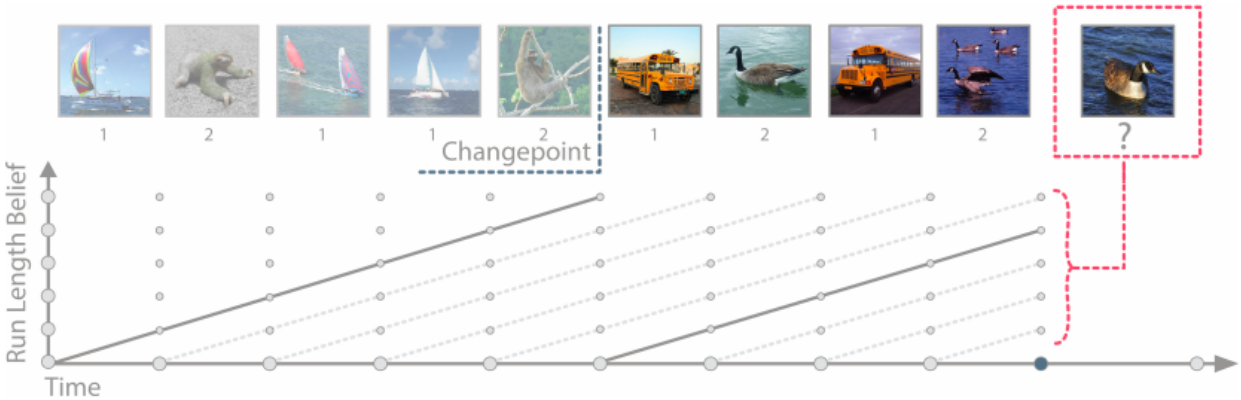
for $i \in 0 \ldots k$,

$$\Delta\theta^P = -\beta \nabla_{\theta^P} \mathcal{L}(\theta^{NM}, \theta^P, D_{Train})$$

Continual Meta-learning without task boundaries

Reference - *Continual Meta-learning without tasks.*

Task boundaries are unknown and change, but discretely and with probability λ. Algorithm keeps track of $b_t(r)$ its belief that, and $\eta_t[r]$ *adapted* parameters if, the current task has run for r steps, $\forall r \in \{0 \dots t-1\}$. [e.g., $\eta_t[r] =$ hidden representation of a model-based meta-learner using past r observations.]



Adaptation: $p_\theta(\hat{y}_t|x_{1:t}, y_{1:t-1}) = \sum_{r=0}^{t-1} b_t(r) p(\hat{y}_t|x_t, \eta_{t-1}[r])$ & $l_t = NLL(\hat{y}_t, y_t)$
$\hat{b}_t(r) = p(y_t|x_{1:t}, \eta_{t-1}[r]) b_t(r)$, and $b_{t+1}(r)|_{r>0} = (1-\lambda)\hat{b}_t(r-1), b_{t+1}(0) = \lambda$
Update $\eta_t[r]$; and every k steps update θ using $\nabla_\theta \sum_{t-k}^{t} l_t$.

3. Learning from almost No Data: Meta-interpretive Program Synthesis

Learning from One Example using Logical Induction

Reference - *One-shot Information Extraction from Document Images using Neuro-Deductive Program Synthesis*

Learn a *program* to extract specific entities from document images that follow a template, from *one* sample. E.g. correspondence no., 186FDBC1802472 here:

```
TO
DNB NOR BANK ASA
TRADE FINANCE/GUARANTEE
DEPT. NO.0021
OSLO NORWAY
NO
```

```
Please Quote in all correspondence
186FDBC1802472
```

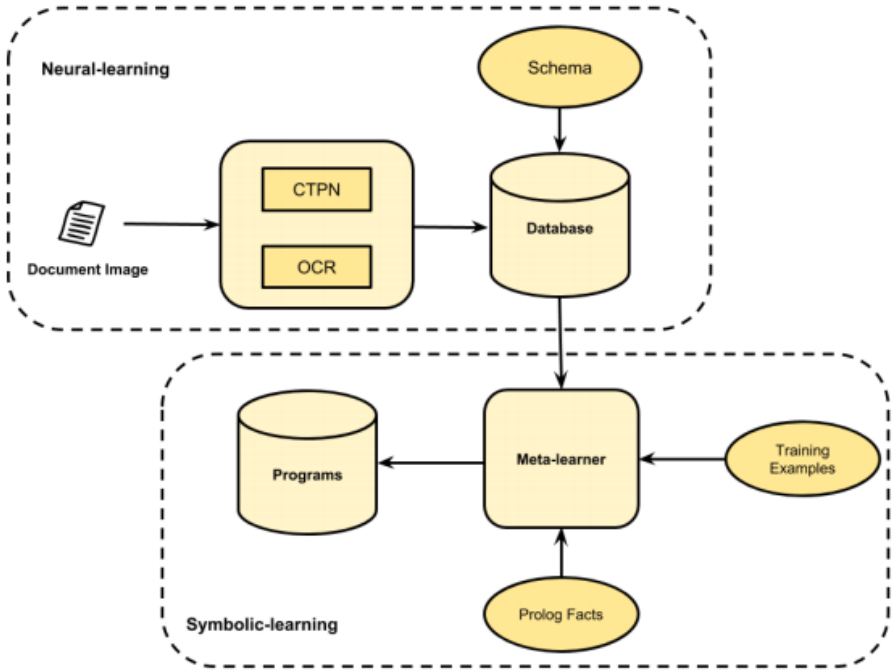
DL to extract to a database; then ILP.

A correct program for this example:

```
corr(A,B) :-
        has_keyword('Please', A,C),
        has_line_below(C,B).
```

But ILP will also find this incorrect one:

```
corr(A,B) :-
       has_keyword('Please',A,C),
       left_of('Please',C,D),
       right_of('ASA',D,C),
       has_line_below_word(C,B).
```

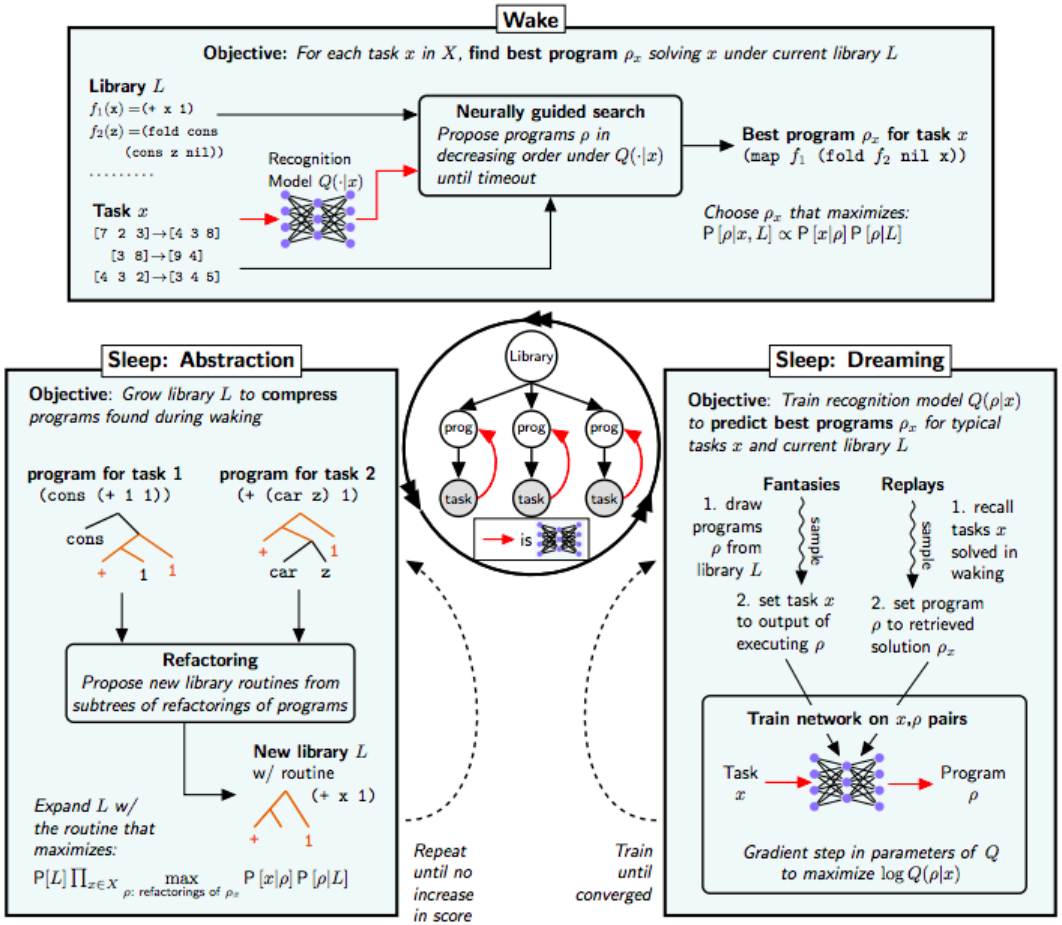


Key idea: perturb *all* entities in the DB to create a 'noisy clone' and feed both examples to ILP, which then learns the correct program.

Logical Meta-learning

Reference - *DreamCoder: Growing generalizable, interpretable knowledge with wake-sleep Bayesian program learning*

For input-output task x learn a program ρ_x using library of primitive operations L. $P(\rho) \sim l(\rho)$. 'Improve' library and learn faster as more tasks are solved.



Wake:
$\rho_x = \text{argmax}_\rho P(\rho|x, L) \propto P(x|\rho)P(\rho|L)$ for task x.

Sleep: $L = \text{argmax}_L \prod_x \max_{\rho=r(\rho_x)} P(x|\rho)P(\rho|L)$

Dream: update $Q_\theta(\rho|x)$: $\text{argmin}_\theta \|P(\rho|x, L) - Q_\theta(\rho|x)\|$

Key: efficient algorithm to search over (potentially exponential) refactorings $r(\rho_x)$.

4. Applications of Meta-learning

Selected Applications of Meta-learning

1. **E-commerce and Retail:** Recommendations systems as user-behavior changes, new items are introduced. Adapting to local users/items. Demand forecasting as user-behavior and economic indicators change; adapting to local conditions/products.
2. **Advertising:** Multi-channel attribution models that adapt to new inventory and content. Maximizing cost-per-conversion.
3. **Manufacturing Supply-chain:** Demand forecasting, optimal replenishment, pre-picking in distribution centers across *very* large number of SKUs.
4. **Industrial Operations (IOT):** Optimal control of industrial equipment, plant operations; preventive maintenance (predicting RUL, detect anomalies). Adapting to equipment from different manufacturers, conditions, usage profiles etc.

Applications of Meta-learning (cont)

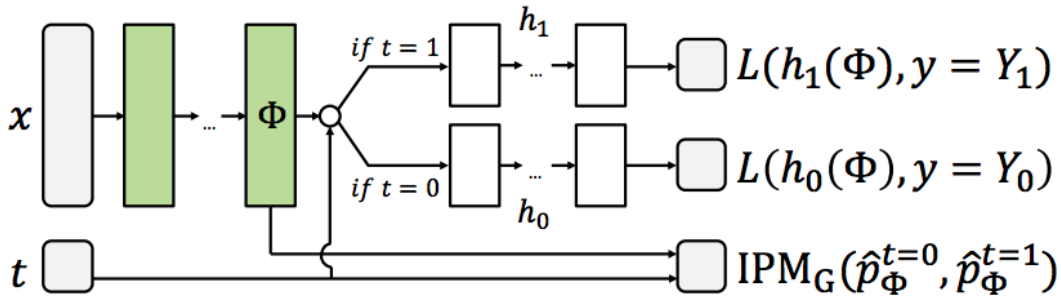
5. **Healthcare:** Drug/protocol efficacy and optimal choice, from operational data (causality). Predicting incidents in ICU. Medical imaging. Adapting to different populations, equipment, etc.
6. **Finance:** Optimizing returns while controlling risk, adapting to changes in market conditions. Adapting global models to specific financial instruments.
7. **Software Development:** Program synthesis from examples, natural language. Adaptation to different use-cases, programming/natural languages.
8. **Cybersecurity:** Detecting malware/intrusions. Adapting to new attacks - most attacks are variations of previous ones.

5. Causality, Disentangled Representations and Causal Reinforcement Learning

Causality and Meta-learning

References - *Learning Representations for Counterfactual Inference* and *MetaCI: Meta-learning for Causal Inference ...*

Treatment is imbalanced due to confounding features, i.e. $p(t|x) \neq p(\bar{t}|x)$: So we learn a representation $\phi(x)$ in which $p(t|\phi(x))$ and $p(\bar{t}|\phi(x))$ are close. Goal is to predict average treatment effect $ATE = \mathbb{E}|Y_1(x) - Y_0(x)|$. Note: one of these is 'counterfactual' and always unknown in real data for given x, t!



Batch of n_1 treatments and n_0 non-treatments, minimize loss

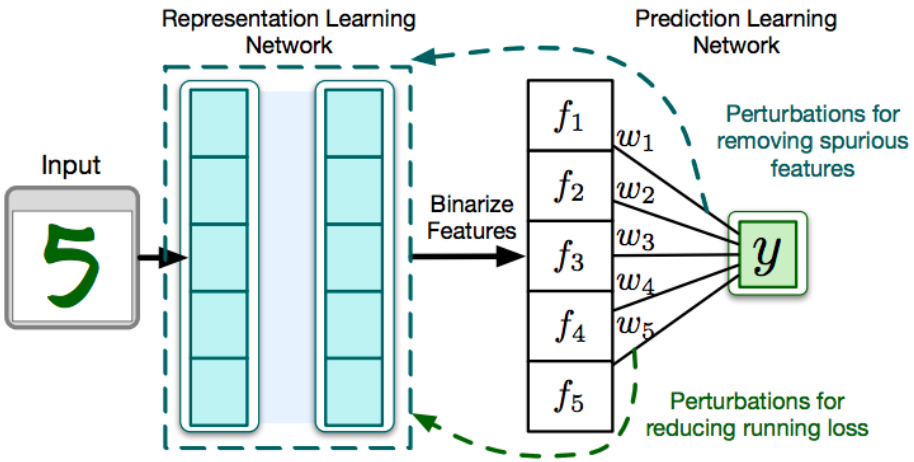
$$\mathcal{L} = \frac{1}{n_1} L(h_1(\phi) + \frac{1}{n_0} L(h_0(\phi)) + \|f(\phi)\|_2; \quad f_W = \frac{1}{n_0} \sum_{i=0}^{n_0-1} \phi_W(x_i) - \frac{1}{n_1} \sum_{i=n_0}^{n_1} \phi_W(x_i)$$

MetCI applies MAML-like meta-learning across tasks differing in $p(t|x)$ and $p(y|x,t)$, so as to estimate ATE *faster* on new tasks.

Online Meta-learning of Invariant (Causal) Features

References - *Learning Causal Features Online*

Correlation between a causal feature and target should not change as different parts of the source distribution $p(x)$ are experienced. Note: assumption is that $x = o(s)$ is observed from a 'true' set of features s, and $p(y|s)$ does not change over time. Further, learning is online: $w^t = w^{t-1} - \eta \nabla_{w^{t-1}} \mathcal{L}(f^T w^{t-1}, y_t)$.



Track mean u_i and variance v_i of w_i:

if $f_i = 1$**:**

$u_i^t = \alpha u_i^{t-1} + (1-\alpha) w_i$

$v_i^t = \beta v_i^{t-1} + (1-\beta)(w_i^t - u_i^t)(w_i^t - u_i^{t-1})$

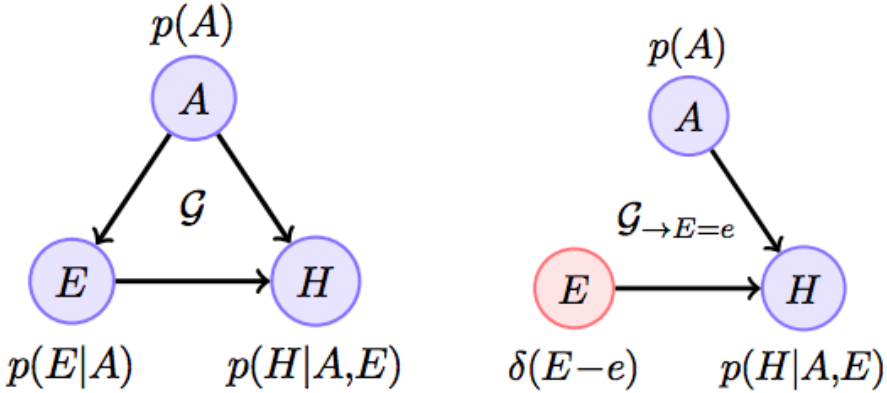
Mask f_i using $\sigma(g_i)$; $g_i^0 = 0$ &

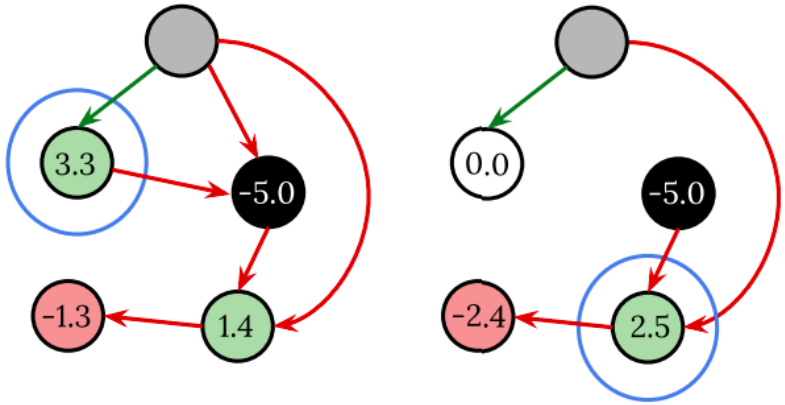
$$\Delta g_i = -\frac{1}{|v_i|^2}(v_i - \sum_{j=1}^{n} v_j) \text{ every } N \text{ steps.}$$

Weights in the representation network are randomly set to $0, 1, -1$ during learning; the change is kept if either overall v or running loss decreases.

Learning Causality via Meta-RL

References - *Causal Reasoning from Meta-reinforcement Learning*





Meta-RL is used to learn causal structure by training on many random causal graphs of a fixed size N. The agent first interacts with the graph for $N-1$ steps and then answers a quiz: e.g., "given an (unobserved) intervention, e.g., $do(x_j) = 5$) which observed variable x_i would be have the largest average value?", and the agent's reward r is x_i itself. Root node is unobserved.

During the interaction phase: interventional agents can select x_i to intervene on, e.g., $do(x_i) = -5$, whereas observational agents cannot.

Counterfactual quiz: "given an (unobserved) counterfactual intervention on an un-intervened x_j, e.g., $do(x_j) = 5$) *for the last observation*, which observed variable x_i would *have had* the largest *value*?

Results: interventional agents beat observational agents, and obsv. oracles.

Disentangled Causal Representations

References - *Learning Causal State Representations ...* and *Learning Finite-state Representations of Recurrent Policy Networks*

Discrete causal states of a (continuous-state) MDP are partitions of histories that are causally equivalent, i.e., predict the same future. Learn continuous-space RNNs for next-state and reward prediction (not shown separately):

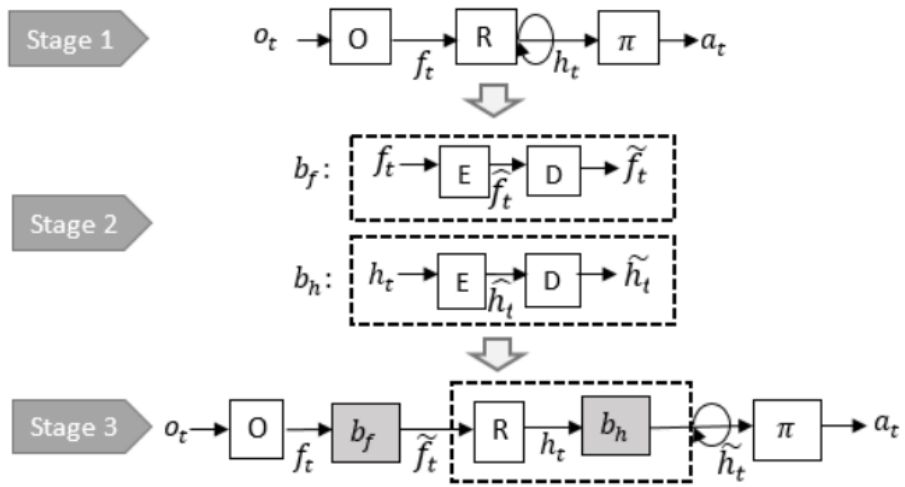
$f_t = \phi(o_t); h_t = \delta(f_t, h_{t-1}); a_t = \pi(h_t)$

Auto-encode f and h into $\{-1, 0, 1\}^k$:

$b(x) = D(Q(E(x)); \; Q(x) \in \{-1, 0, 1\}^k$

not differentiable: use identity as gradient in back-propagation

Insert into trained RNN and fine-tune. Now the RL problem can be solved directly, or using another RNN with these bottlenecks inserted, e.g., via Q-learning, Reinforce, etc. Experiments show that inserting the bottlenecks improves learning speed.



QBN: Quantized Bottlenck Network

Further, in another paper *Invariant Causal Prediction for Block MDPs* similar causal states learned across *different* MDPs help to generalize to unseen ones

PAC Bayes Theory: Generalization Bounds

PAC-Bayes Model:
Risk of hypothesis $h(x) = \hat{y}$, $R(h) = \mathbb{E}_{x,y\sim\mathcal{D}}\mathcal{L}(h(x), y)$
Empirical risk: $R_S(h) = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(h(x_i), y_i), S = \{x_i, y_i\}$
Gibbs classifier G_Q: sample h from distribution Q:
Gibbs risks: $R(G_Q) = \mathbb{E}_Q R(h)$ and $R_S(G_Q) = R_S(h)$

Generalization bounds in PAC-Bayes model:

$$\Pr\left(R(G_Q) \leq R_S(G_Q) + \sqrt{\frac{\mathrm{KL}(Q||P) + \log\frac{m}{\delta}}{2(m-1)}}\right) \geq 1 - \delta$$

where P is the *prior* distribution of h "before learning". {Proof-sketch: slide 51}

Note that Q is a function of P and S, i.e., Q is 'learned' starting from P using the samples S; we bound the true Gibbs risk R in terms of the empirical Gibbs risk R_S, for h sampled from Q post learning.

Meta-learning Theory using PAC-Bayes and its Application

Reference - *Meta-Learning by Adjusting Priors Based on Extended PAC-Bayes Theory*

'Non-vacuous' (i.e., practically useful, beat MAML) *bounds* for meta-learning. Each task $\tau_i \sim \tau$ uses $S_j \sim D_j$ with m_j samples; and samples prior $P_i \sim \mathcal{P}$: a 'meta-prior'. Meta-learner computes $\mathcal{Q}$: meta-posterior, from n tasks.

$$\Pr\left(\left(\mathop{\mathbb{E}}_{P\sim\mathcal{Q}} R(G_Q) \leq \sum_{j=1}^{n} \mathop{\mathbb{E}}_{P\sim\mathcal{Q}} \left[\left[R_{S_j}(G_{Q_j}) + \sqrt{\frac{\mathrm{KL}(\mathcal{Q}||\mathcal{P}) + \mathrm{KL}(Q_j||P) + \log\frac{2nm_j}{\delta}}{2(m-1)}}\right.\right.\right.$$
$$\left.\left.\left. + \sqrt{\frac{\mathrm{KL}(\mathcal{Q}||\mathcal{P}) + \log\frac{2n}{\delta}}{2(n-1)}}\right]\right) \geq 1-\delta \quad (1)$$

{Proof-sketch slide 52}

Essentially, meta-learning learns to sample 'good' priors $P \sim \mathcal{Q}$ to learn from. Bound is better if (i) $\mathcal{Q} \approx \mathcal{P}$ and (ii) $Q_i \approx P$ ($\sim$ similar tasks).

Appendix

Proof-sketch for PAC-Bayes bound

Let $r_s(h) \equiv f(\Delta)$ where $\Delta = \|R_S(h) - R(h)\|^2$, and $f = e^{c\Delta}$ $(c = 2(m-1))$.

$$\log \mathbb{E}_P r_S(h) \geq \log \mathbb{E}_Q \frac{P(h)}{Q(h)} r_s(h) \geq \mathbb{E}_Q \log r_S(h) - \mathbb{E}_Q \log \frac{Q(h)}{P(h)}$$
$$= \mathbb{E}_Q \log r_S(h) - \mathtt{KL}(Q||P)$$

By Markov[53] $\Pr\left(r_S(h) \leq \frac{1}{\delta}\mathbb{E}_P r_S(h)\right) \geq 1-\delta$; so using the above in this:
$\Pr\left(\mathbb{E}_Q \log r_S(h) \leq \mathtt{KL}(Q||P) + \log\left[\frac{1}{\delta}\mathbb{E}_P r_S(h)\right]\right) \geq 1-\delta$; using $f = e^{c\Delta}$:
$\Pr\left(\mathbb{E}_Q \Delta \leq \frac{1}{c}\left[\mathtt{KL}(Q||P) + \log\left[\frac{1}{\delta}\mathbb{E}_P e^{c\Delta}\right]\right]\right) \geq 1-\delta$; now taking $\mathbb{E}_S$:
$\Pr\left(\|R_S(G_Q) - R(G_Q)\|^2 \leq \frac{1}{c}\left[\mathtt{KL}(Q||P) + \log\left[\frac{1}{\delta}\mathbb{E}_P \mathbb{E}_S e^{c\Delta}\right]\right]\right) \geq 1-\delta$
Hoeffding[53]: $\Pr(R_S(h) - R(h) \geq \epsilon) \leq e^{-2m\epsilon^2} \overset{54}{\Longrightarrow} \mathbb{E}_S[e^{2(m-1)\Delta^2}] \leq m$

$$\Pr\left(R(G_Q) - R_S(G_Q) \leq \sqrt{\frac{\mathtt{KL}(Q||P) + \log\frac{m}{\delta}}{2(m-1)}}\right) \geq 1-\delta \qquad \square$$

Back to slide 48

Proof-sketch for Meta-learning bound

For any task let $\rho(h) \equiv Q(S,P), P \sim \mathcal{Q}$ and $\pi(h) \equiv P \sim \mathcal{P}$: $\mathtt{KL}(\rho||\pi) \equiv \mathbb{E}_\rho \log \frac{\rho}{\pi} = \mathbb{E}_{P\sim\mathcal{Q}}\mathbb{E}_{h\sim Q(S,P)} \log \frac{\mathcal{Q}(P)Q(S,P)(h)}{\mathcal{P}(P)P(h)} = \mathtt{KL}(\mathcal{Q}||\mathcal{P}) + \mathbb{E}_{P\sim\mathcal{Q}}\mathtt{KL}(Q||P)$
plugging this into the bound for task j and taking $\mathbb{E}_{P\sim\mathcal{Q}}$:

$$\mathtt{Pr}\left(\mathbb{E}_{P\sim\mathcal{Q}}[R(G_{Q_j}) - R_{S_j}(G_{Q_j})] \leq \sqrt{\frac{\mathtt{KL}(\mathcal{Q}||\mathcal{P}) + \mathbb{E}_{P\sim\mathcal{Q}}\mathtt{KL}(Q_j||P) + \log\frac{m_j}{\delta_j}}{2(m_j-1)}}\right) \geq 1-\delta$$

Now view meta-learning as learning $Q(S,P)$ from 'data' S_j:

$$\mathtt{Pr}\left(\mathop{\mathbb{E}}_{P\sim\mathcal{P}}[R(G_{Q(S,P)}) - \frac{1}{n}\sum_{j=1}^{n} R_{S_j}(G_{Q(S_j,P)})] \leq \sqrt{\frac{\mathtt{KL}(\mathcal{Q}||\mathcal{P}) + \log\frac{n}{\delta_0}}{2(n-1)}}\right) \geq 1-\delta_0$$

combining above and using $\delta_i = \frac{\delta_0}{2i}$: Pr(risk of at least one task exceeds its bound) $< \sum_i \frac{\delta_0}{2i} < \delta_0$. So Pr(no tasks exceeds its bound) $\geq 1 - \delta_0$ $\square$

Back to slide 49

Some Probability Bounds

Markov Inequality:

$$\mathbb{E}[x] = \int_0^\infty x p(x) dx > \int_{\frac{\mathbb{E}[x]}{\delta}}^\infty x p(x) dx > \frac{\mathbb{E}[x]}{\delta} \int_{\frac{\mathbb{E}[x]}{\delta}}^\infty p(x) dx = \frac{\mathbb{E}[x]}{\delta} P\left(x > \frac{\mathbb{E}[x]}{\delta}\right)$$

So

$$P\left(x \leq \frac{\mathbb{E}[x]}{\delta}\right) \geq 1 - \delta$$

Hoeffding Inequality: (Proof)

Sum S_n of n independent random variables $x_i \in \{a_i, b_i\}$ satisfies:

$$P(S_n - \mathbb{E}[S_n] \geq \epsilon) \leq e^{\frac{-2\epsilon^2}{\sum_{i=1}^n (b_i - a_i)^2}}$$

Back to slide 48

Lemma 1

Let $\Delta = R_S(h) - R(h)$.

By Hoeffding $P(\Delta \geq \epsilon) = \int_{\epsilon}^{\infty} f(\Delta) d\Delta \leq e^{-2m\epsilon^2}$.

Choose $f(\Delta)$ to maximize; so $\int_0^{\epsilon} f(\Delta) d\Delta = 1 - e^{-2m\epsilon^2}$, $f(\Delta) = 4m\Delta e^{-2n\Delta^2}$

Now $\mathbb{E}[e^{2(m-1)\Delta^2}] = \int_0^{\infty} e^{2(m-1)x^2} f(x) dx \leq \int_0^{\infty} e^{2(m-1)x^2}\ 4m\ x e^{-2mx^2}$

$$= 4m \int_0^{\infty} x e^{-2x^2} = -m|_0^{\infty} e^{-2x^2} = m$$

Back to slide 52