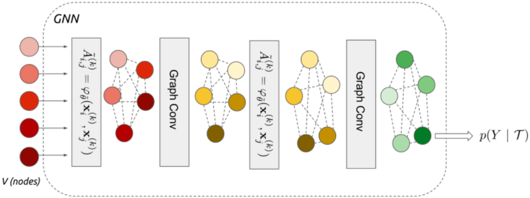
Metric-based Meta-learning (cont)

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; $W^k, V^k \in \mathbb{R}^{d^{k+1} \times d^k}$:
$$\mathrm{x}_i^{k+1} = \sum_{j \neq i} \varphi_{ij}^k(W^k \mathrm{x}_j, V^k \mathrm{x}_i)$$

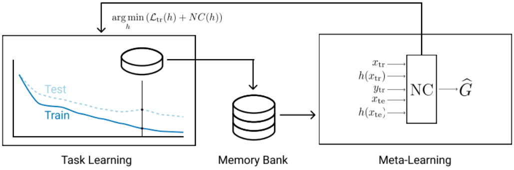


Finally $p(y|\mathcal{T}) = Softmax(\mathrm{x}^N)$ for all nodes, test and training. Network is trained using available training and test labels across many tasks - each task's data D_{Train}, D_{Test} is input as a graph, with possibly different number of examples. This is a mix of metric-based (due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

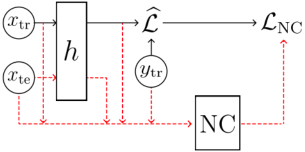


$\mathcal{L}_{NC} \propto \mathcal{L}_{Test} - \mathcal{L}_{Train} - NC(h)$

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

$$NC(X_{tr}, X_{te}, Y_{tr}, h(X_{tr}), h(X_{te})) = \frac{1}{m'} \sum_{i=1}^{m'} g(A);$$

where $A = \frac{Softmax(QK^T)}{\sqrt{d}} V$



NC-regularized loss: $\mathcal{L}_{reg} = \mathcal{L}_{Test} + \lambda \mathcal{L}_{NC}$; 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.