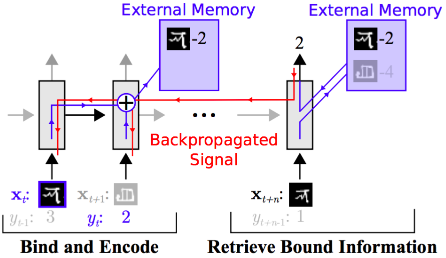
Model-based Meta-learning (cont) 2

Reference - *Meta-Learning with Memory-Augmented Neural Networks*

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 $\mathrm{k}^{r/w} = \phi(z)^T$: ($z$ is 'cell state')
$M_t = M_{t-1} + \mathrm{w}^w \mathrm{k}_t^w$; $\mathrm{w}^w =$ 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:



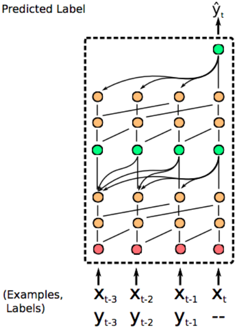
'least used' $w^{lu} = 1$ for t smallest elements of w^u, 0 otherwise
$\mathrm{w}_t^w = \delta w_{t-1}^r + (1-\delta)\mathrm{w}_{t-1}^{lu}$; prior to writing the least used row of M is zeroed.
Read/write to 'memory' - we can view this as a 'neural Turing machine'.

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.



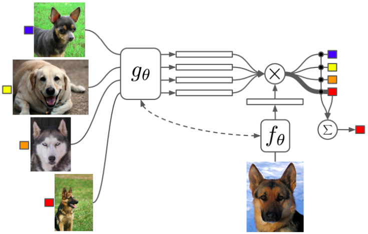
Metric-based Meta-learning

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 $f(\hat{(}x), g(S))$.

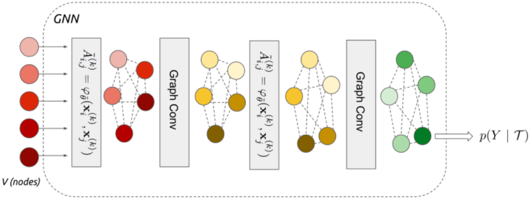


Metric-based Meta-learning (cont)

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.