x x' x''

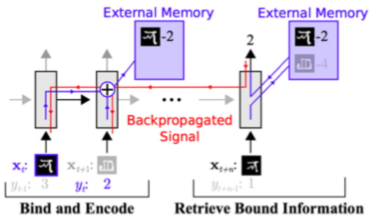
train

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 keys $k_t = \phi(x_t)^T$:
$M_t = M_{t-1} + w^w k_t$; $w^w =$ write weights
Retrieved memory $r_t = w^r M_t$ used to predict $\hat{y}_t$ using a feedforward layer.
(read weights $w^r = Softmax(M_t \cdot k_t)$)
usage weights: $w^u_t = \gamma u^u_{t-1} + w^r_t + w^w_t$
used to compute w^w as follows:



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

update W

$k + y$

$= M$

D^{train}

$ff \rightarrow y$

$f_W(D_{Train}, x)$

$x \in D^{test}$

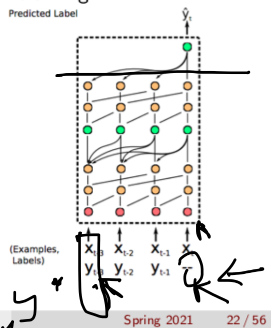
$loss(D^{test})$

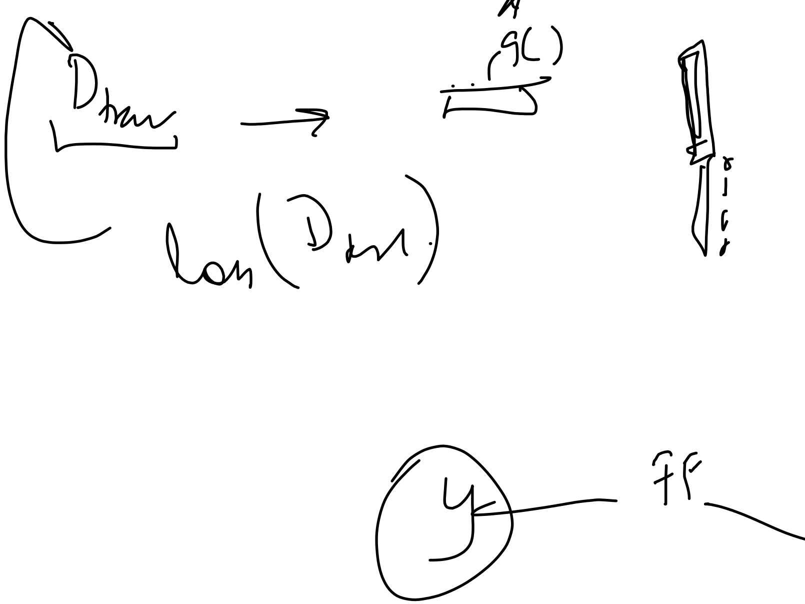
Model-based Meta-learning (cont) 3

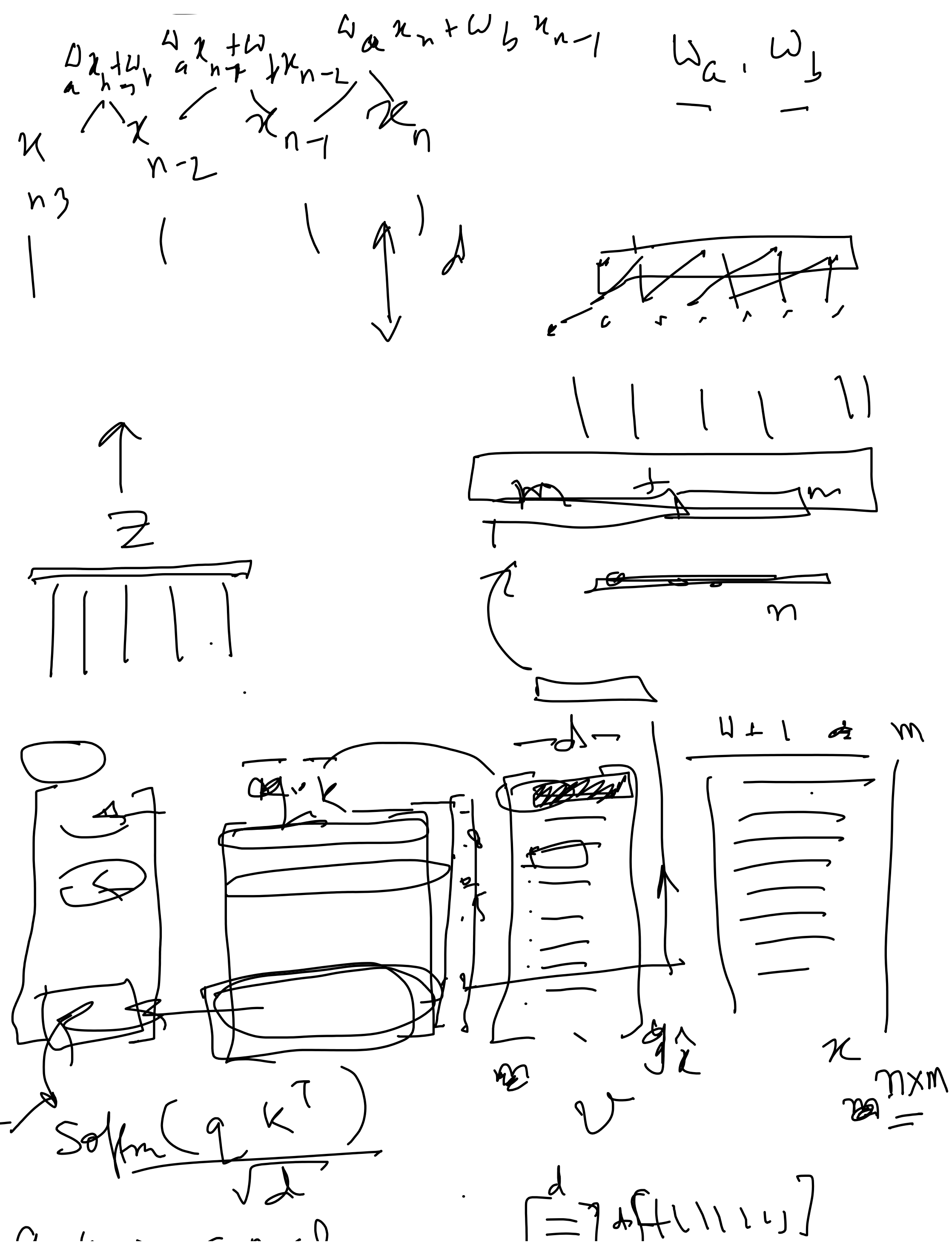
Reference - *A Simple Neural Attentive Meta-learner*

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 $y_{\mathtt{test}}$ 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:
$u = \mathrm{1Dconv}(z, 2^i)$; $a = \tanh(u) * \sigma(u)$; $C(z) = (z, a)$
$k = W_k z + b_k$, $q = W_q z + b_q$, $v = W_v z + b_v$
$p = Softmax(\frac{qk^T}{\sqrt{d}})$; $A(z) = (z, vp^T)$ $z_{in} \in \mathbb{R}^{d\times t}$
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.





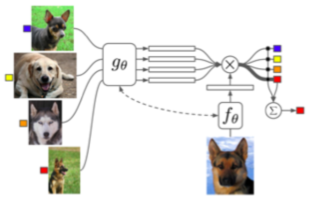


Metric-based Meta-learning

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.

$\hat{y} = \sum_{i=1}^{k} a(\hat{x}, x_i) y_i$ 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 $f(\hat{(x)}, g(S))$.

1, r, r ∈ nx*

~~W~~ b

nxm

xW + b

nxm ↑ $\boxed{mxd}$

MAML - Opt based

CNP - Model based

MAN - Metric based

#HW2 compare all three — shots, ways

speed of training.
accuracy.

meta-train sample, tried.

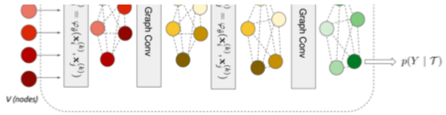
Metric-based Meta-learning (cont)

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
$G(\{x \in \mathbb{R}^d\} = \{(x, y)\}, \varphi)$
$\varphi^k(x_i, x_j) = MLP_{\theta_k}(|x_i - x_j|)$

$\varphi(x_i, x_j) = MLP_{\theta^k}(|x_i - x_j|)$
GC: aggregate from neighbors & concatenate; $W^k, V^k \in \mathbb{R}^{d^{k+1} \times d^k}$:
$x_i^{k+1} = (\sum_{j \neq i} \varphi_{ij}^k W^k x_j, V^k x_i)$



Finally $p(y|\mathcal{T}) = Softmax(x^N)$ 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.

Store
=
mch_lrn
(fix + tang
sem)

PAC Bayes Theory: Generalization Bounds PAC-Bayes Model: Risk of hypothesis h(x) = 9, R(h) = y) Empirical risk: Rs(h) = y), S = {Xi, y} Gibbs classifier GQ: sample h from distribution Q: Gibbs risks: R(GQ) = EQR(h) and Rs(GQ) = Rs(h) Generalization bounds in PAC-Bayes model: KL(QIIP) + log -T Pr R(GQ) 21-5 2(m— 1) where P is the prior distribution of h "before learning" . 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 Rs, for h sampled from Q post learning. spin. ml