Few-Shot Learning with Meta-Learning: Progress Made and Challenges Ahead

Hugo Larochelle
Google Brain

A RESEARCH AGENDA

- Deep learning successes have required a lot of labeled training data
  - collecting and labeling such data requires significant human labor
  - practically, is that really how we'll solve AI ?
  - scientifically, this means there is a gap with ability of humans to learn, which we should try to understand

- Alternative solution : exploit other sources of data that are imperfect but plentiful
  - unlabeled data (unsupervised learning)
  - multimodal data (multimodal learning)
  - multidomain data (transfer learning, domain adaptation)

A RESEARCH AGENDA

- Deep learning successes have required a lot of labeled training data
  - collecting and labeling such data requires significant human labor
  - practically, is that really how we'll solve AI ?
  - scientifically, this means there is a gap with ability of humans to learn, which we should try to understand

- Alternative solution : exploit other sources of data that are imperfect but plentiful
  - unlabeled data (unsupervised learning)
  - multimodal data (multimodal learning)
  - multidomain data (transfer learning, domain adaptation)


1
2
3
4
5
?
?
D_{train}
D_{test}

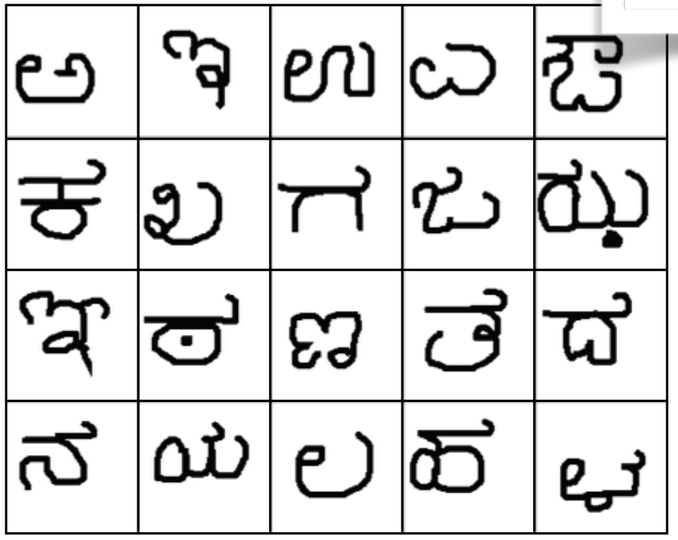
People are good at it

People are good at it

Human-level concept learning through probabilistic program induction

Brenden M. Lake,[1*] Ruslan Salakhutdinov,[2] Joshua B. Tenenbaum[3]

Machines are getting better at it



Teachable Machine

Secure | https://teachablemachine.withgoogle.com

LEARNING

INPUT

0 EXAMPLES CONFIDENCE

TRAIN GREEN

0 EXAMPLES CONFIDENCE

TRAIN PURPLE

0 EXAMPLES CONFIDENCE

TRAIN ORANGE

OUTPUT

GIF Sound Speech

Teachable Machine

Secure | https://teachablemachine.withgoogle.com

LEARNING

INPUT

0 EXAMPLES CONFIDENCE

TRAIN GREEN

0 EXAMPLES CONFIDENCE

TRAIN PURPLE

0 EXAMPLES CONFIDENCE

TRAIN ORANGE

OUTPUT

GIF Sound Speech

A RESEARCH AGENDA

- Let's attack directly the problem of **few-shot learning**
  - we want to design a learning algorithm A that outputs a good parameters $\boldsymbol{\theta}$ of a model M, when fed a small dataset $D_{train}=\{(\mathbf{x}_i, y_i)\}_{i=1}^{L}$

- Idea: let's *learn* that algorithm A, end-to-end
  - this is known as **meta-learning** or **learning to learn**

RELATED WORK: TRANSFER LEARNING

- Large image datasets (e.g. ImageNet) have been shown to allow training representations that transfer to other problems
  - DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition (2014)
    Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng and Trevor Darrell
  - CNN Features off-the-shelf: an Astounding Baseline for Recognition (2014)
    Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, Stefan Carlsson
  - …some have even reported some positive transfer on medical imaging datasets!

- In few-shot learning, we aim at transferring the complete training of the model on new datasets (not just transferring the features or initialization)
  - ideally there should be no human involved in producing a model for new datasets

RELATED WORK: ONE-SHOT LEARNING

- One-shot learning has been studied before
  - One-Shot learning of object categories (2006)
    Fei-Fei Li, Rob Fergus and Pietro Perona
  - Knowledge transfer in learning to recognize visual objects classes (2004)
    Fei-Fei Li
  - Object classification from a single example utilizing class relevance pseudo-metrics (2004)
    Michael Fink
  - Cross-generalization: learning novel classes from a single example by feature replacement (2005)
    Evgeniy Bart and Shimon Ullman
- These largely relied on hand-engineered features
  - with recent progress in end-to-end deep learning, we hope to learn a representation better suited for few-shot learning

META-LEARNING

- Learning algorithm A
  - *input*: training set $D_{train}=\{(\mathbf{x}_i,\mathbf{y}_i)\}$ (from a **single task**)
  - *output*: parameters $\boldsymbol{\theta}$ model M (the **learner**)
  - *objective*: good performance on test set $D_{test}=\{(\mathbf{x}'_i,\mathbf{y}'_i)\}$

- Meta-learning algorithm
  - *input*: meta-training set $\mathscr{D}_{meta-train}=\{(D^{(n)}_{train},D^{(n)}_{test})\}_{n=1}^{N}$ of *episodes* (from **multiple tasks**)
  - *output*: parameters $\boldsymbol{\Theta}$ algorithm A (the **meta-learner**)
  - *objective*: good performance on meta-test set $\mathscr{D}_{meta-test}=\{(D'^{(n)}_{train},D'^{(n)}_{test})\}_{n=1}^{N'}$

If you don’t evaluate on never-seen problems/datasets…

If you don’t evaluate on never-seen problems/datasets…

… it’s not meta-learning!

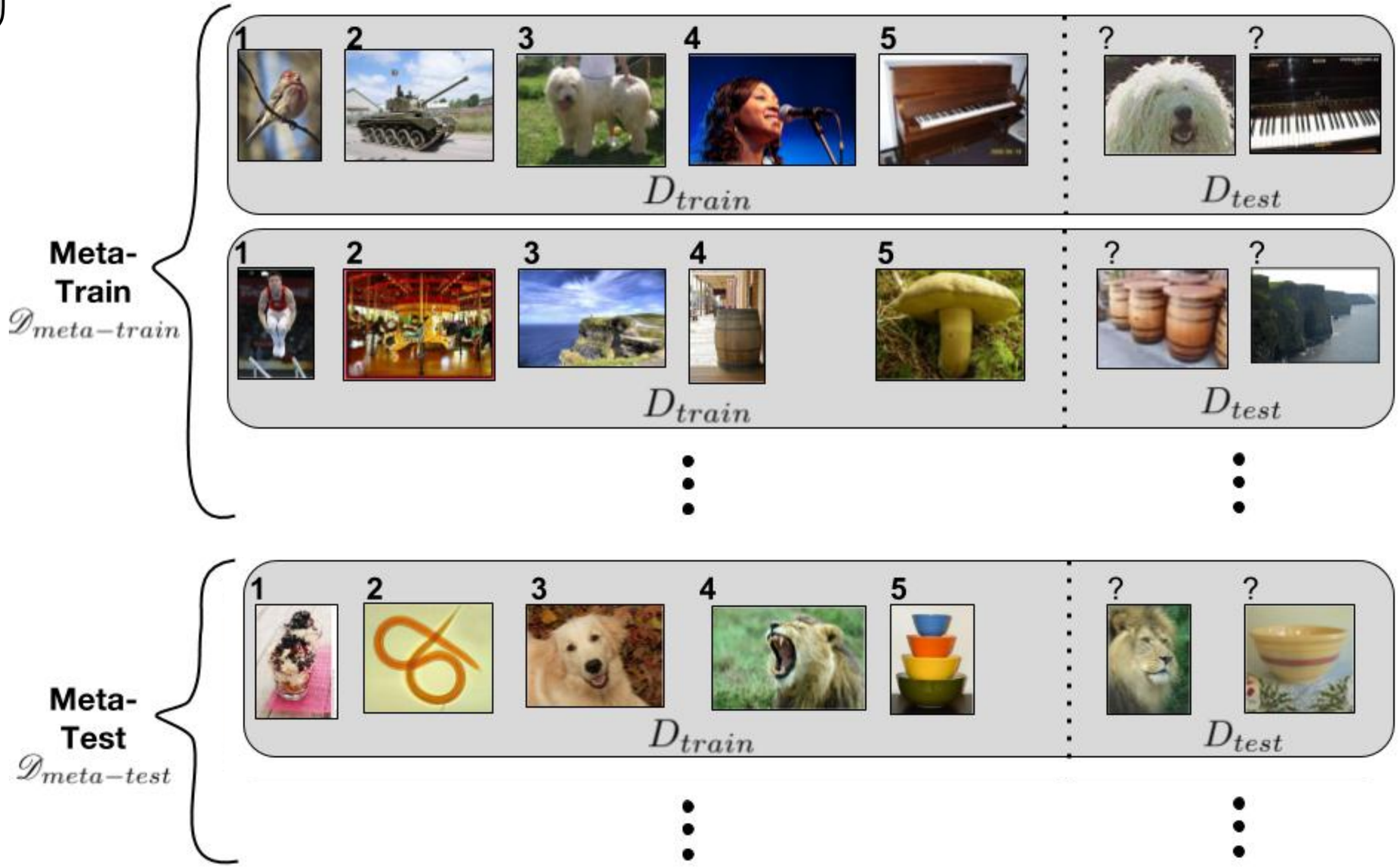
META-LEARNING

META-LEARNING

$D_{train} \mid D_{test}$

=

episode

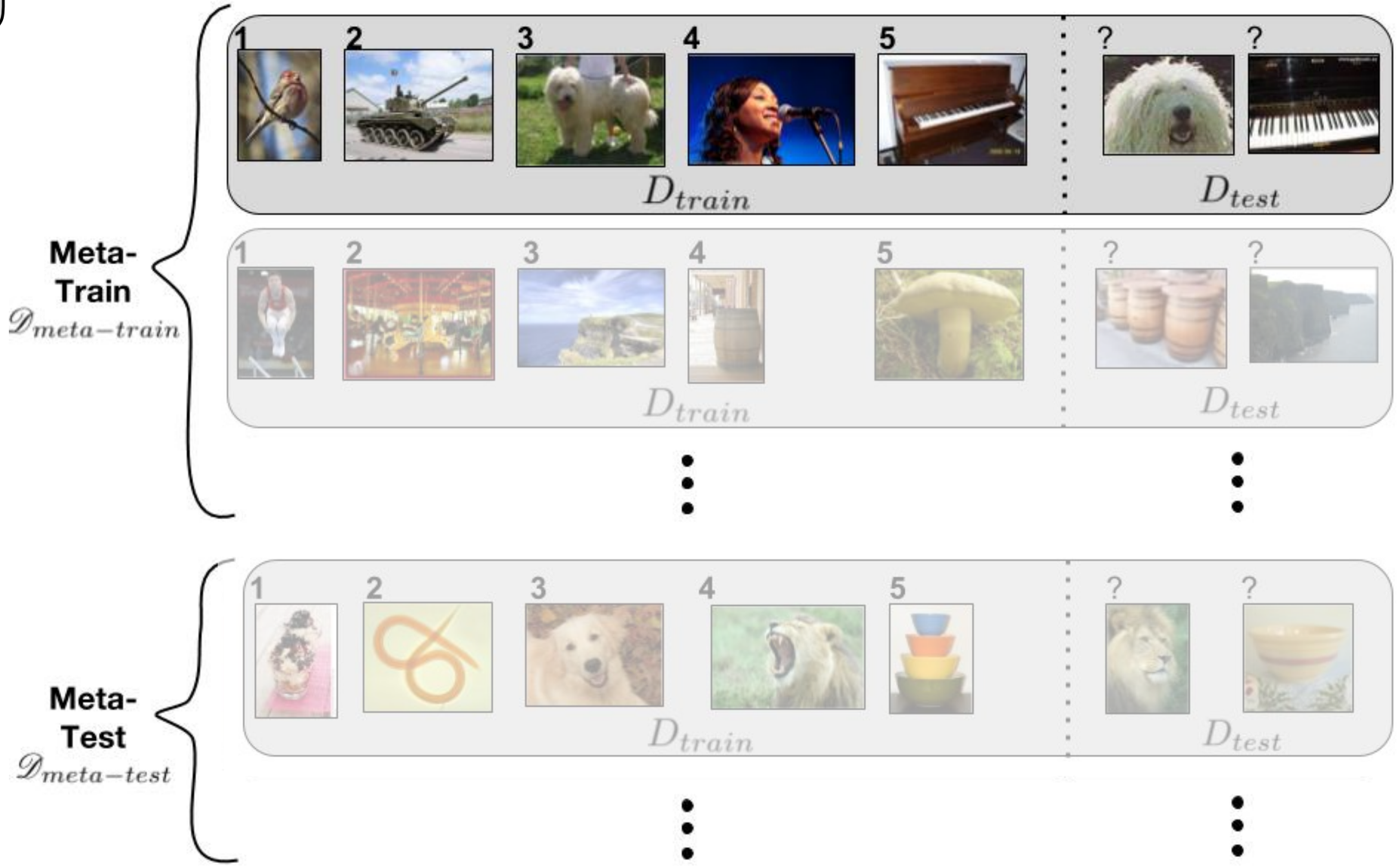


META-LEARNING

$D_{train} \mid D_{test}$

=

episode



META-LEARNING

D_{train} | D_{test}

=

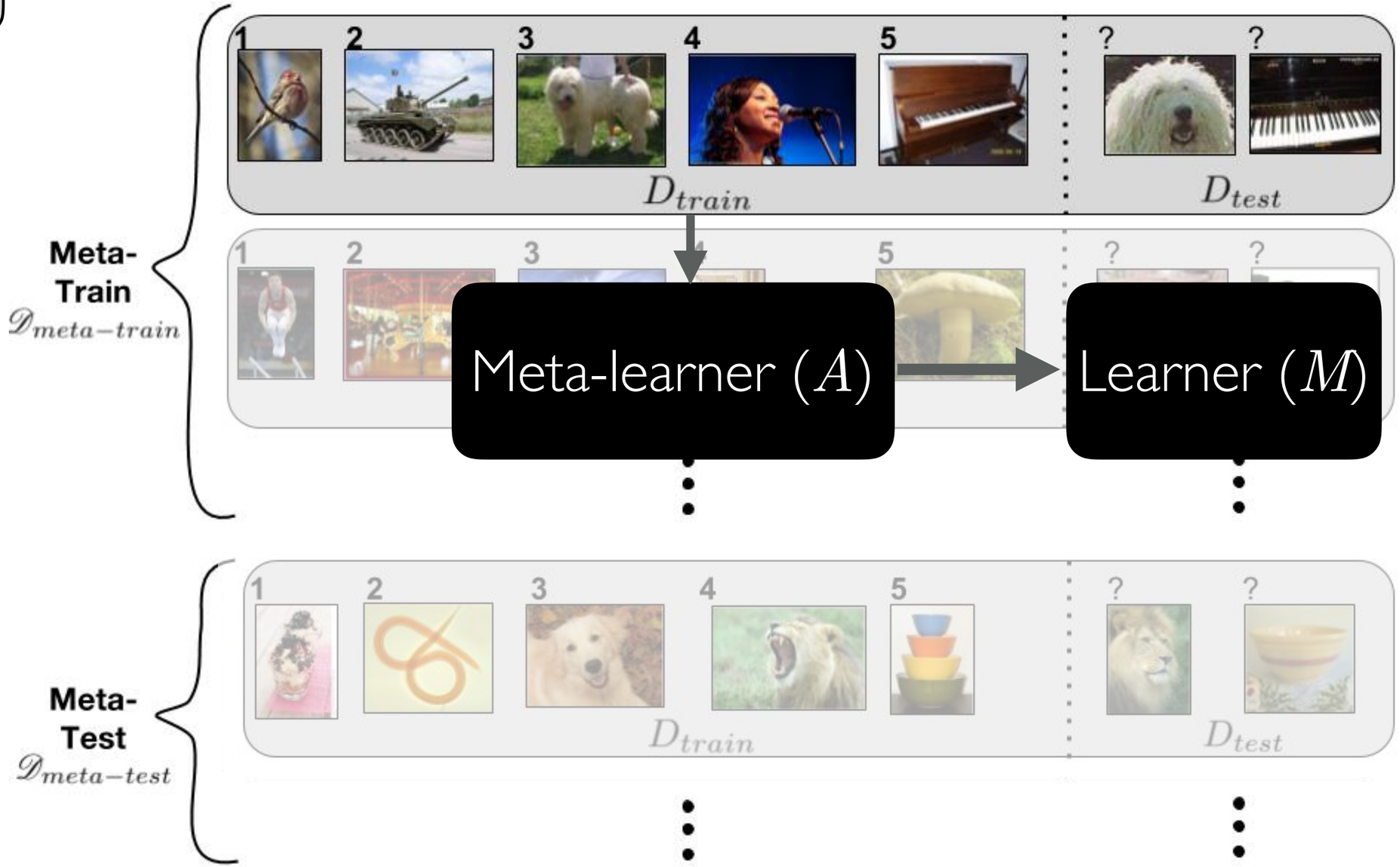
episode

META-LEARNING

$D_{train} \mid D_{test}$

||

episode

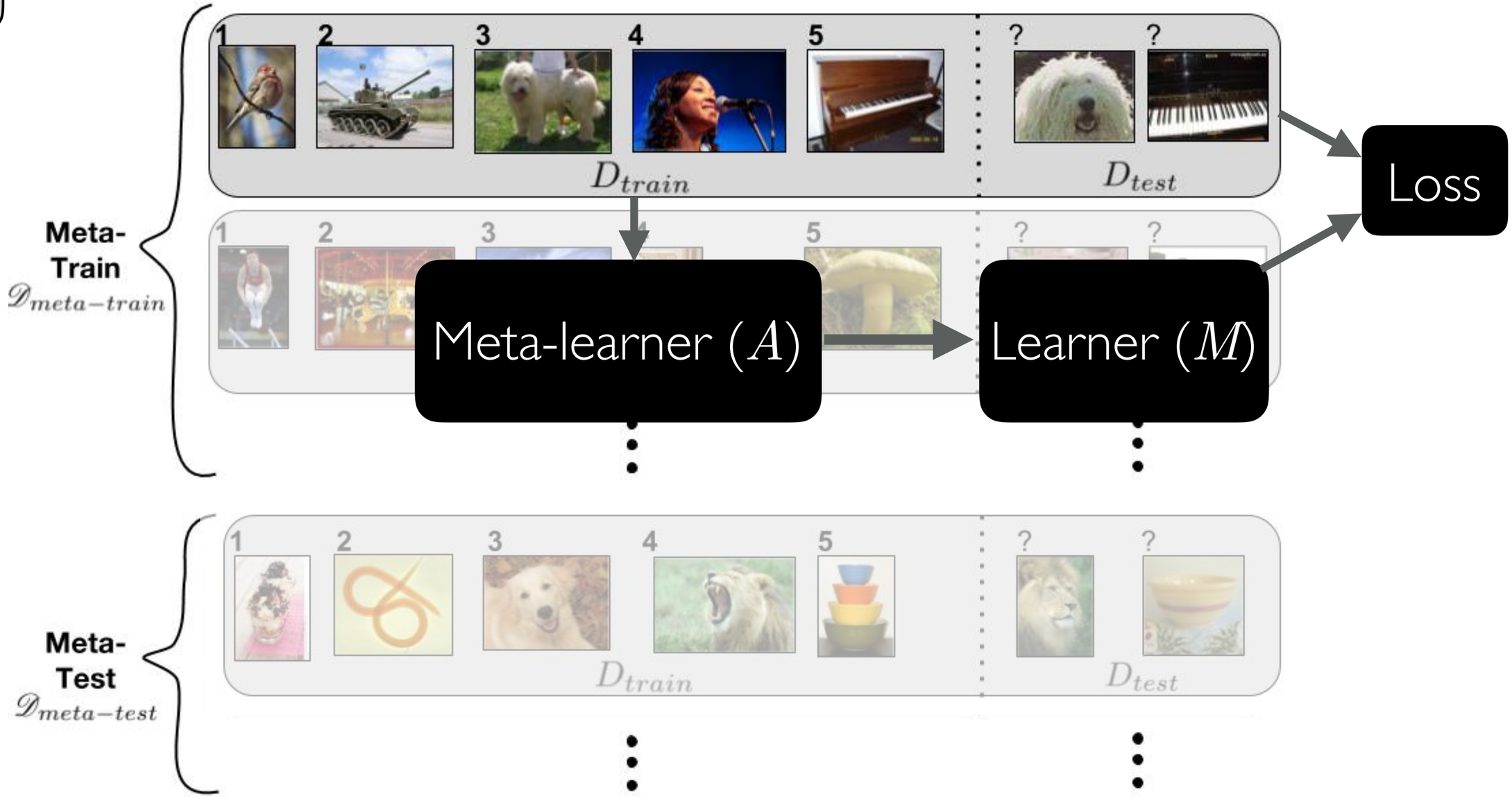


META-LEARNING

D_{train} | D_{test}

||

episode

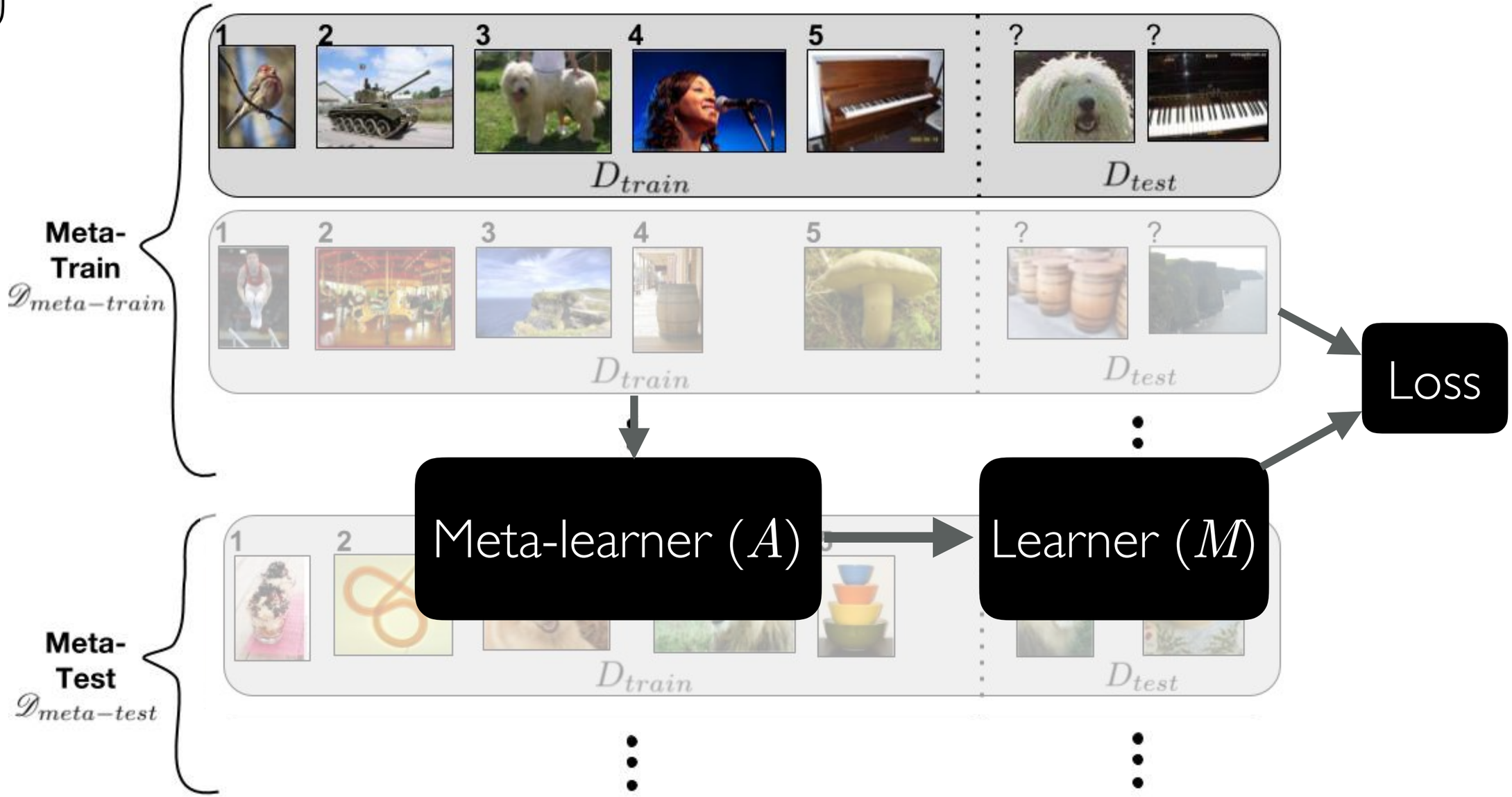


META-LEARNING

$D_{train} \mid D_{test}$

=

episode

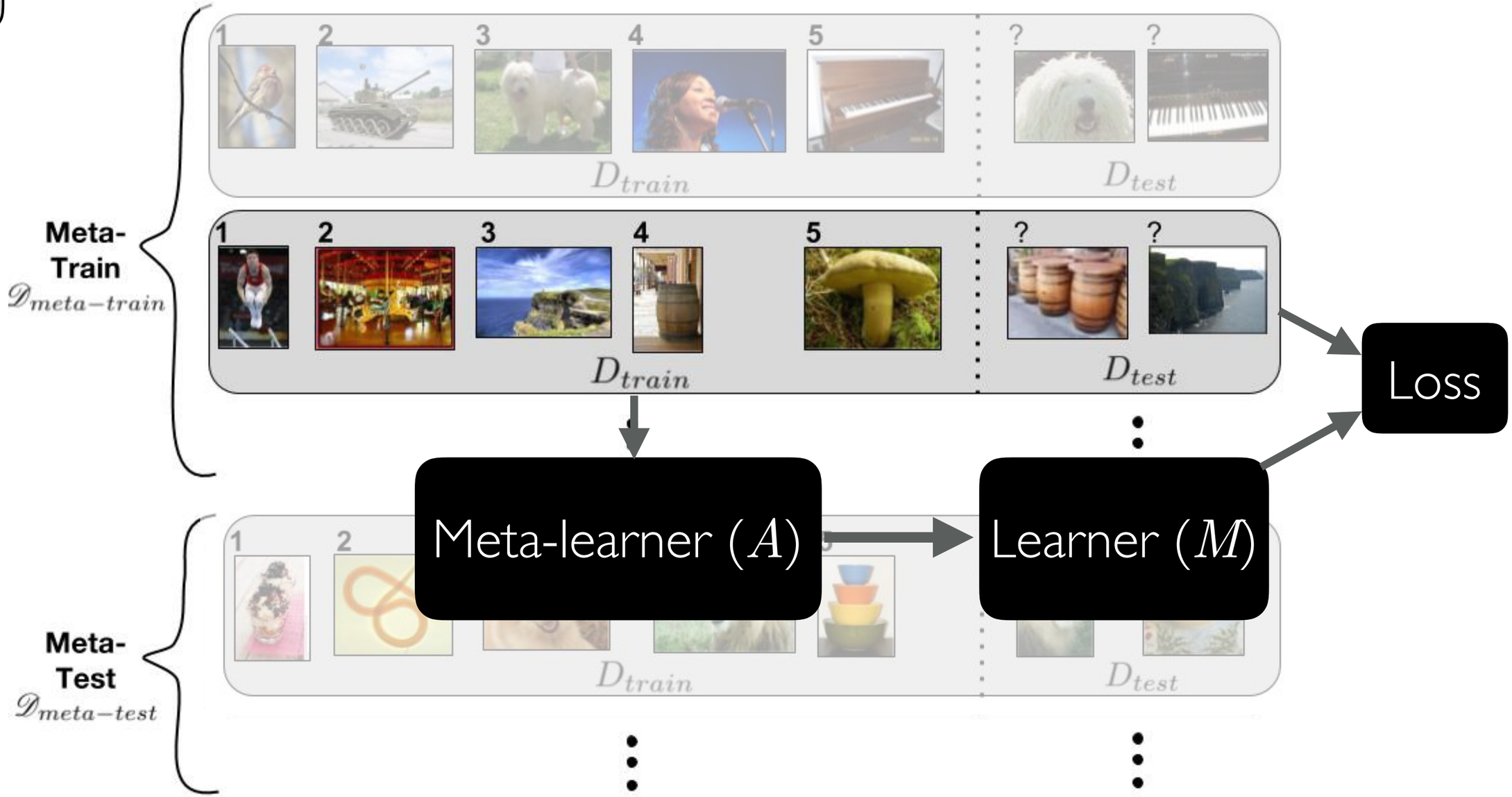


META-LEARNING

D_{train} | D_{test}

||

episode



META-LEARNING NOMENCLATURE

Training set

Test set

Meta-training set

Meta-test set

META-LEARNING NOMENCLATURE

Episode { Training set / Test set

Meta-training set

Meta-test set

META-LEARNING NOMENCLATURE

Episode

Training set ⟺ Support set

Test set ⟺ Query set

Meta-training set

Meta-test set

META-LEARNING NOMENCLATURE

Episode

Training set ⟺ Support set

Test set ⟺ Query set

Meta-training set ⟺ Training set

Meta-test set ⟺ Test set

LEARNING PROBLEM STATEMENT

- Assuming a probabilistic model M over labels, the cost per episode can become

$$C(\Theta; D_{train}, D_{test}) = \frac{1}{|D_{test}|} \sum_{\substack{(\mathbf{x}'_i, y'_i) \\ \in D_{test}}} -\log p_\Theta(y'_i | \mathbf{x}'_i, D_{train})$$

- Depending on the choice of meta-learner, $p_\Theta(y'|\mathbf{x}', D_{train})$ will take a different form

CHOOSING A META-LEARNER

- How to parametrize learning algorithms?

- Two approaches to defining a meta-learner
  - Take inspiration from a known learning algorithm
    - kNN/kernel machine: Matching networks (Vinyals et al. 2016)
    - Gaussian classifier: Prototypical Networks (Snell et al. 2017)
    - Gradient Descent: Meta-Learner LSTM (Ravi & Larochelle, 2017) , MAML (Finn et al. 2017)
  - Derive it from a black box neural network
    - MANN (Santoro et al. 2016)
    - SNAIL (Mishra et al. 2018)

CHOOSING A META-LEARNER

- How to parametrize learning algorithms?

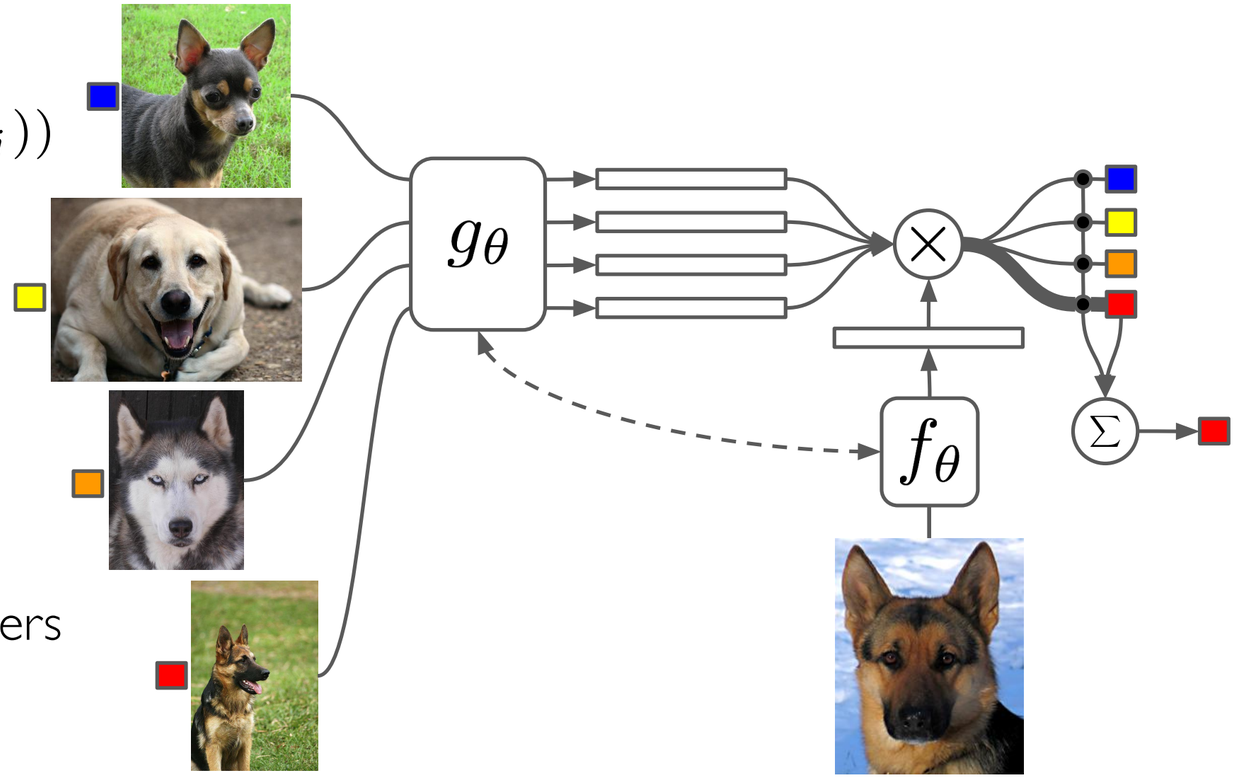
- Two approaches to defining a meta-learner
  - **Take inspiration from a known learning algorithm**
    - kNN/kernel machine: Matching networks (Vinyals et al. 2016)
    - Gaussian classifier: Prototypical Networks (Snell et al. 2017)
    - Gradient Descent: Meta-Learner LSTM (Ravi & Larochelle, 2017) , MAML (Finn et al. 2017)
  - Derive it from a black box neural network
    - MANN (Santoro et al. 2016)
    - SNAIL (Mishra et al. 2018)

MATCHING NETWORKS

- Training a "**pattern matcher**"

$$a(x', x_i) = e^{c(f(x'), g(x_i))} / \sum_{j=1}^{k} e^{c(f(x'), g(x_j))}$$

$$\hat{y} = \sum_{i=1}^{k} a(x', x_i) y_i$$



  - similarity function can be the cosine similarity
  - embedding functions f and g can have tied parameters or be separate and dependent on D_{train} (i.e. $f(x', D_{train})$ and $g(x_j, D_{train})$)
  - see description of Fully Conditional Embedding (FCE) functions in paper

- Matching networks for one shot learning (2016)
  Oriol Vinyals, Charles Blundell, Timothy P. Lillicrap, Koray Kavukcuoglu, and Daan Wierstra

PROTOTYPICAL NETWORKS

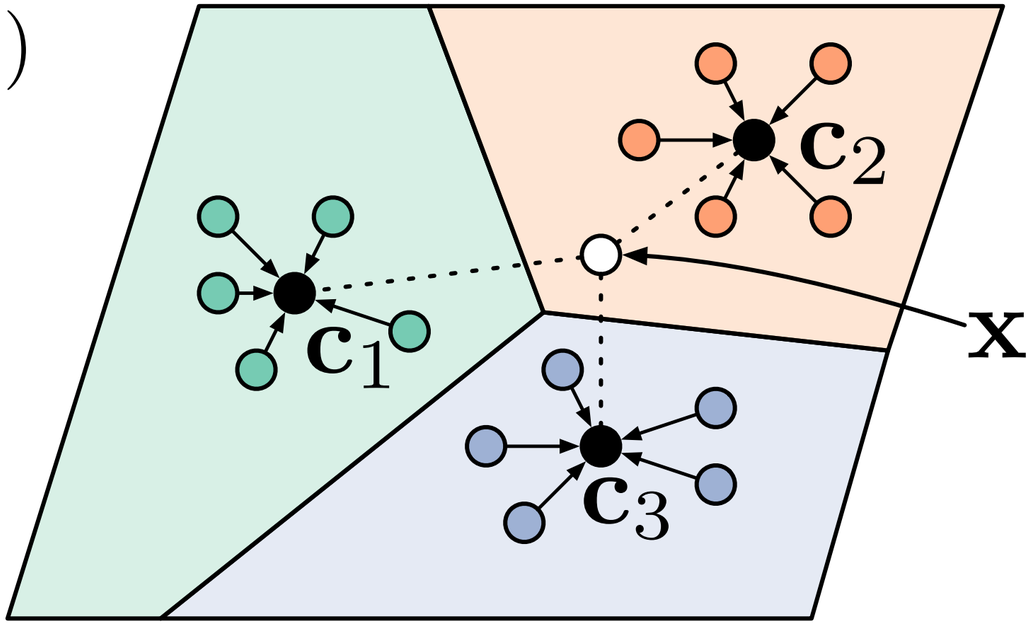
- Training a "**prototype extractor**"

$$p_{\boldsymbol{\phi}}(y' = k \mid \mathbf{x}') = \frac{\exp(-d(f_{\boldsymbol{\phi}}(\mathbf{x}'), \mathbf{c}_k))}{\sum_{k'} \exp(-d(f_{\boldsymbol{\phi}}(\mathbf{x}'), \mathbf{c}_{k'}))}$$

$$\mathbf{c}_k = \frac{1}{|S_k|} \sum_{(\mathbf{x}_i, y_i) \in S_k} f_{\boldsymbol{\phi}}(\mathbf{x}_i)$$

$$S_k = \{(\mathbf{x}_i, y_i) | y_i = k, (\mathbf{x}_i, y_i) \in D_{train}\}$$

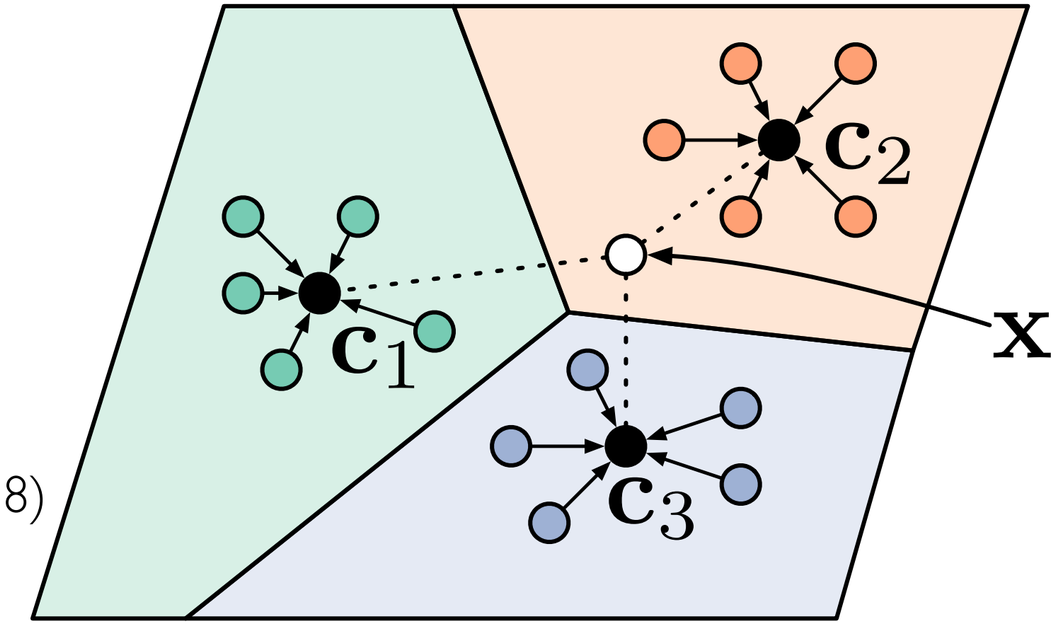
$$\phi \equiv \Theta$$



- Prototypical Networks for Few-shot Learning (2017)
  Jake Snell, Kevin Swersky and Richard Zemel

PROTOTYPICAL NETWORKS

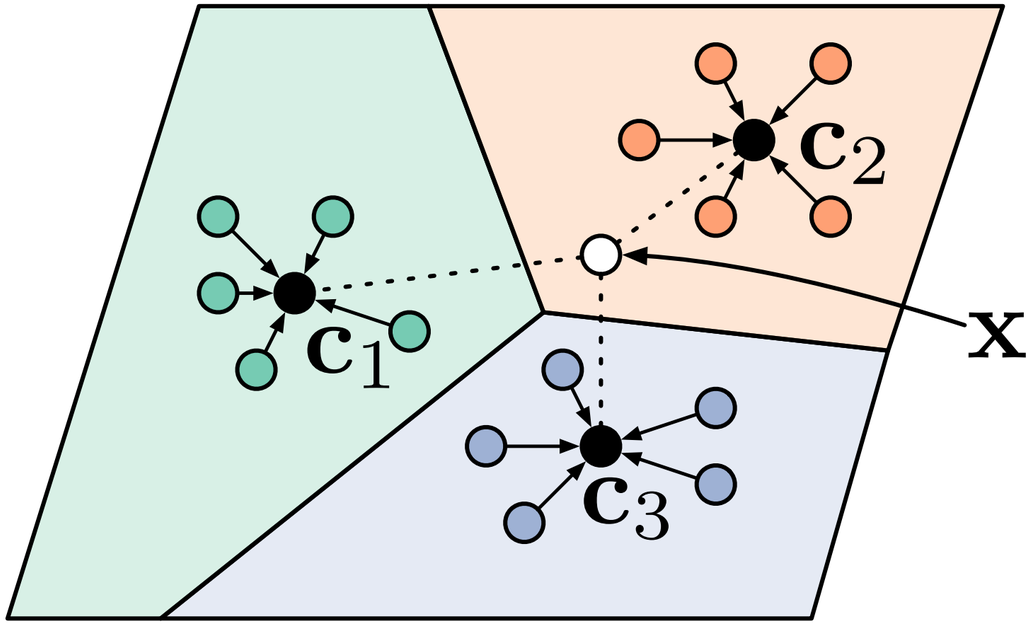
- Training a "**prototype extractor**"
  - distance function $d(\cdot, \cdot)$ can be anything (euclidean squared, negative cosine)
  - if distance is euclidean squared, equivalent to learning an embedding network $f_\phi(\cdot)$ such that a Gaussian classifier works well
  - idea of training a representation such that *classifier X* works is good in general, as long as can backprop through the training of the classifier
    - Meta-learning with differentiable closed-form solvers (2018) *Bertinetto, Henriques, Torr, Vedaldi*



- Prototypical Networks for Few-shot Learning (2017)
  Jake Snell, Kevin Swersky and Richard Zemel

PROTOTYPICAL NETWORKS

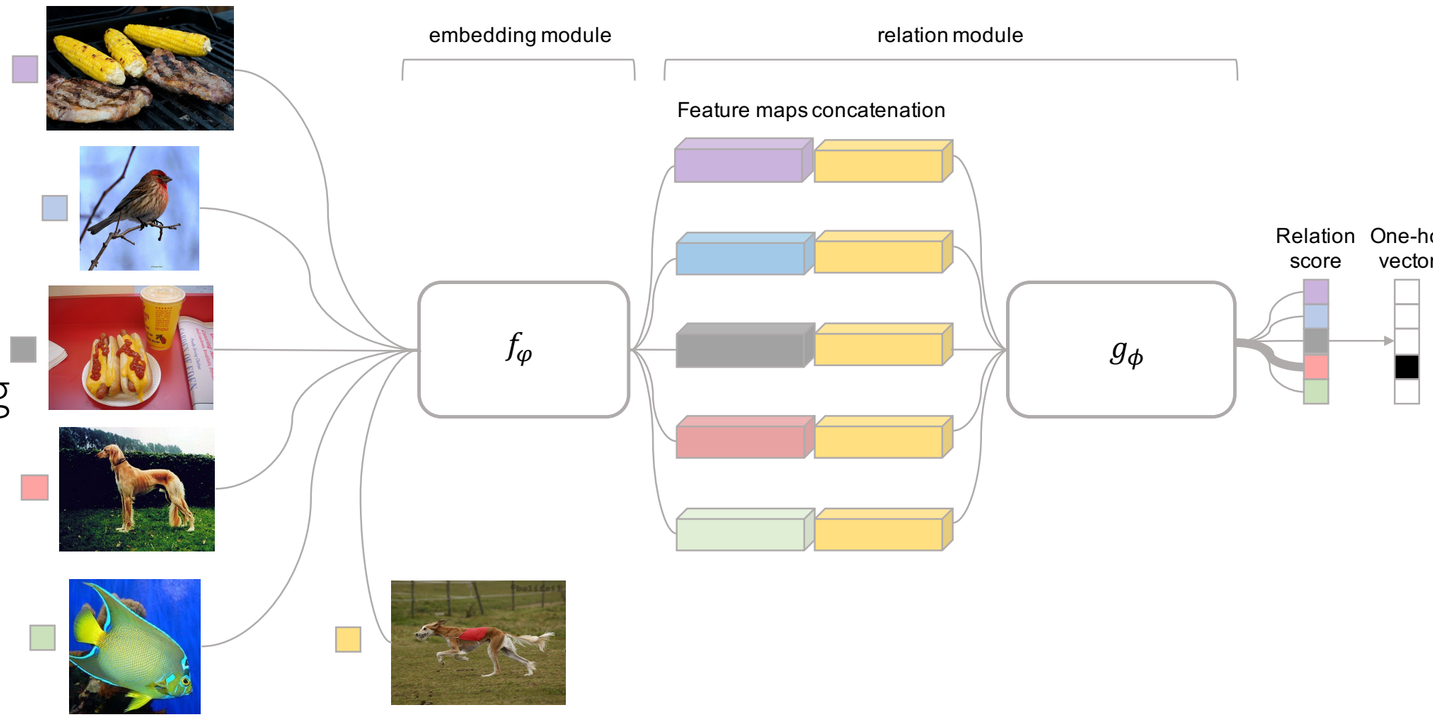
- Training a "**prototype extractor**"
  - Snell et al. find that using more classes in the meta-training episodes compared to meta-testing works better
  - generally, it is possible that better meta-learning can be achieved by generating meta-training episodes differently than meta-testing episodes
    - see leave-one-out training in Uncertainty in Multitask Transfer Learning (2018) *Lacoste, Oreshkin, Chung, Boquet, Rostamzadeh, Krueger*



- Prototypical Networks for Few-shot Learning (2017)
  Jake Snell, Kevin Swersky and Richard Zemel

PROTOTYPICAL NETWORKS

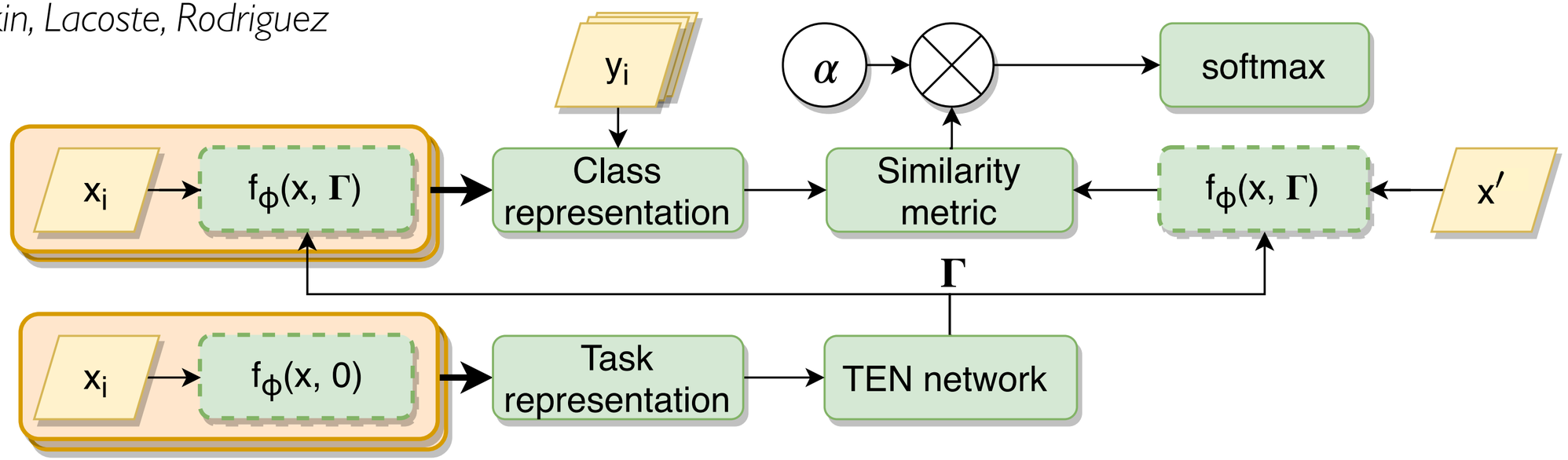
- Training a "**prototype extractor**"
  - Prototypical Networks (and Matching Networks) assume a fixed distance function
  - Why not *learn* the distance function!
  - **Relation Networks** add a *relation module* g_ϕ trained to output a relation score between the class and a test input
  - Embedding module f_φ outputs feature maps, to allow for some spatial reasoning
  - Input of relation module is the concatenation of prototypes with the embedding of the test input
  - Outputs are in [0,1] and trained by MSE to predict the binary one-hot vector



- Learning to Compare: Relation Network for Few-Shot Learning (2018)
  Flood Sung, Yongxin Yang, Li Zhang, Tao Xiang, Philip H.S. Torr, Timothy M. Hospedales

PROTOTYPICAL NETWORKS

- Training a "**prototype extractor**"
  - TADAM: Task dependent adaptive metric for improved few-shot learning (2018)
    Oreshkin, Lacoste, Rodriguez

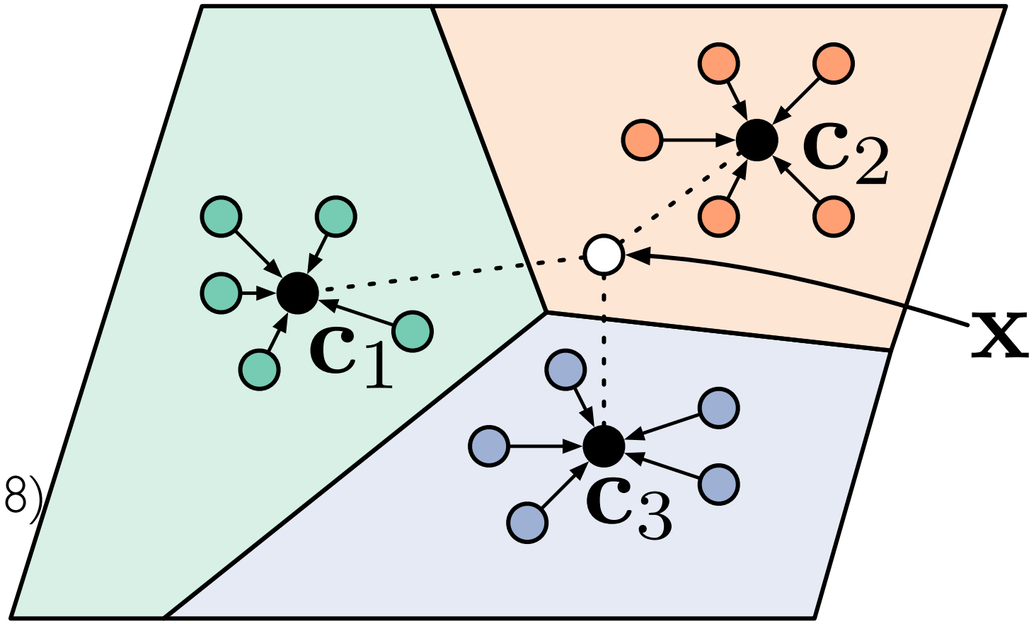


  - embedding function dependent on support set (i.e. task) improves things
  - training the embedding function on a regular classification task from the meta-training set helps

- Prototypical Networks for Few-shot Learning (2017)
  Jake Snell, Kevin Swersky and Richard Zemel

PROTOTYPICAL NETWORKS

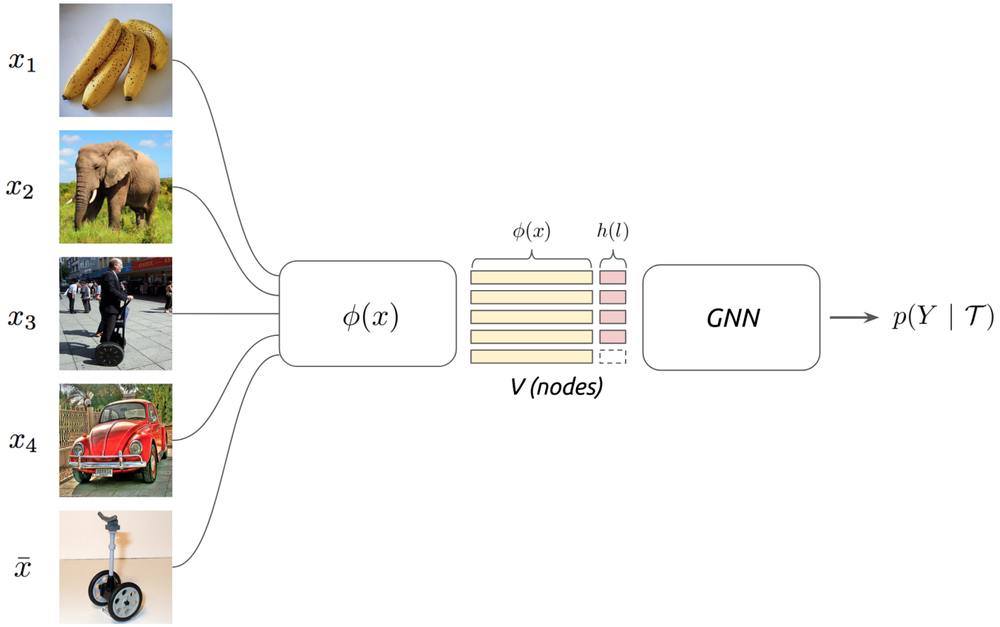
- Training a "**prototype extractor**"
  - prototype vectors are equivalent to output weights of a neural network
  - idea of a meta-learner directly outputting the parameters of a learner has also been explored, with some thought on the architecture of the meta-learning
    - Learning feed-forward one-shot learners (2016)
      Bertinetto, Henriques, Valmadre, Torr, Vedaldi
    - Dynamic Few-Shot Visual Learning without Forgetting (2018)
      Gidaris, Komodakis

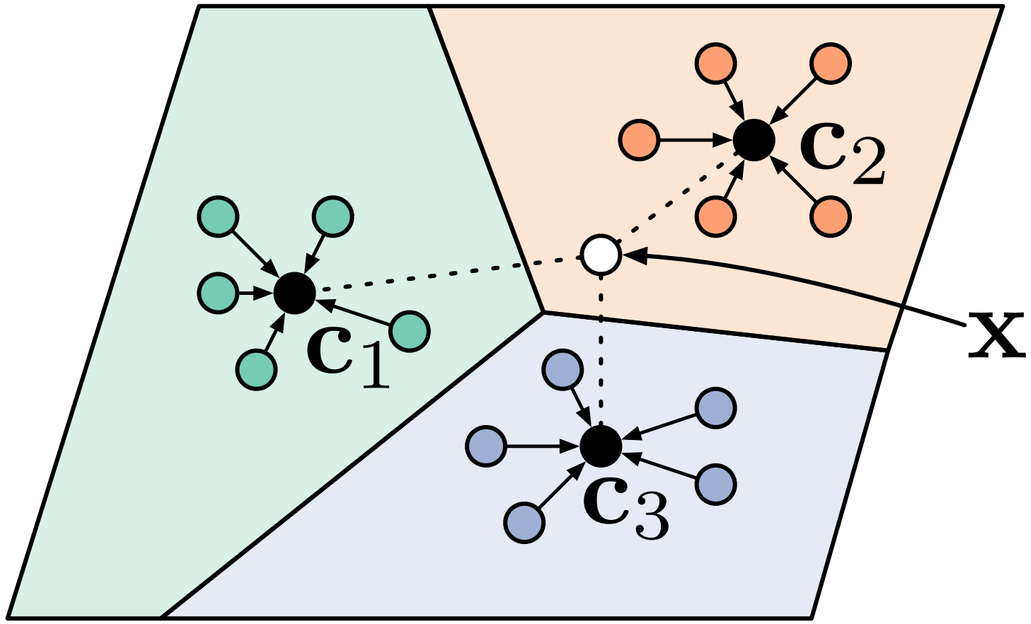


- Prototypical Networks for Few-shot Learning (2017)
  Jake Snell, Kevin Swersky and Richard Zemel

PROTOTYPICAL NETWORKS

- Training a "**prototype extractor**"
  - can be seen as a special case of using a graph neural network
    - Few-Shot Learning with Graph Neural Networks (2018)
      Garcia, Bruna





- Prototypical Networks for Few-shot Learning (2017)
  Jake Snell, Kevin Swersky and Richard Zemel

META-LEARNER LSTM

- Training a "**gradient descent procedure**" applied on some learner M
  - gradient descent starts from some initial parameters θ_0 and then performs the following updates:

$$\theta_t = \theta_{t-1} - \alpha_t \nabla_{\theta_{t-1}} \mathcal{L}_t$$

  - this is quite similar to LSTM cell state updates:

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$

    - state c_t is model M's parameter space θ_t
    - state update $\tilde{c}_t$ is the negative gradient $-\nabla_{\theta_{t-1}} \mathcal{L}_t$
    - f_t and i_t are LSTM gates:

$$i_t = \sigma\left(\mathbf{W}_I \cdot \left[\nabla_{\theta_{t-1}} \mathcal{L}_t, \mathcal{L}_t, \theta_{t-1}, i_{t-1}\right] + \mathbf{b}_I\right)$$

$$f_t = \sigma\left(\mathbf{W}_F \cdot \left[\nabla_{\theta_{t-1}} \mathcal{L}_t, \mathcal{L}_t, \theta_{t-1}, f_{t-1}\right] + \mathbf{b}_F\right)$$

- Optimization as a Model for Few-Shot Learning (2017)
  Sachin Ravi and Hugo Larochelle

META-LEARNER LSTM

- Training a "**gradient descent procedure**" applied on some learner M
  - gradient descent starts from some initial parameters θ_0 and then performs the following updates:

$$\theta_t = \theta_{t-1} - \alpha_t \nabla_{\theta_{t-1}} \mathcal{L}_t$$

  - this is quite similar to LSTM cell state updates:

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$

    - state c_t is model M's parameter space θ_t
    - state update $\tilde{c}_t$ is the negative gradient $-\nabla_{\theta_{t-1}} \mathcal{L}_t$
    - f_t and i_t are LSTM gates:

$$i_t = \sigma\left(\mathbf{W}_I \cdot \left[\nabla_{\theta_{t-1}} \mathcal{L}_t, \mathcal{L}_t, \theta_{t-1}, i_{t-1}\right] + \mathbf{b}_I\right)$$

← *adaptive learning rate*

$$f_t = \sigma\left(\mathbf{W}_F \cdot \left[\nabla_{\theta_{t-1}} \mathcal{L}_t, \mathcal{L}_t, \theta_{t-1}, f_{t-1}\right] + \mathbf{b}_F\right)$$

- Optimization as a Model for Few-Shot Learning (2017)
  Sachin Ravi and Hugo Larochelle

META-LEARNER LSTM

- Training a "**gradient descent procedure**" applied on some learner M
  - gradient descent starts from some initial parameters θ_0 and then performs the following updates:

$$\theta_t = \theta_{t-1} - \alpha_t \nabla_{\theta_{t-1}} \mathcal{L}_t$$

  - this is quite similar to LSTM cell state updates:

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$

    - state c_t is model M's parameter space θ_t
    - state update $\tilde{c}_t$ is the negative gradient $-\nabla_{\theta_{t-1}} \mathcal{L}_t$
    - f_t and i_t are LSTM gates:

$$i_t = \sigma\left(\mathbf{W}_I \cdot \left[\nabla_{\theta_{t-1}} \mathcal{L}_t, \mathcal{L}_t, \theta_{t-1}, i_{t-1}\right] + \mathbf{b}_I\right)$$ ← *adaptive learning rate*

$$f_t = \sigma\left(\mathbf{W}_F \cdot \left[\nabla_{\theta_{t-1}} \mathcal{L}_t, \mathcal{L}_t, \theta_{t-1}, f_{t-1}\right] + \mathbf{b}_F\right)$$ ← *adaptive weight decay*

- Optimization as a Model for Few-Shot Learning (2017)
  Sachin Ravi and Hugo Larochelle

META-LEARNER LSTM

- Training a "**gradient descent procedure**" applied on some learner M
  - gradient descent starts from some initial parameters θ_0 and then performs the following updates:

$$\theta_t = \theta_{t-1} - \alpha_t \nabla_{\theta_{t-1}} \mathcal{L}_t$$

  - this is quite similar to LSTM cell state updates:

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$

    - state c_t is model M's parameter space θ_t ← c_0 becomes a *learned initialization*
    - state update $\tilde{c}_t$ is the negative gradient $-\nabla_{\theta_{t-1}} \mathcal{L}_t$
    - f_t and i_t are LSTM gates:

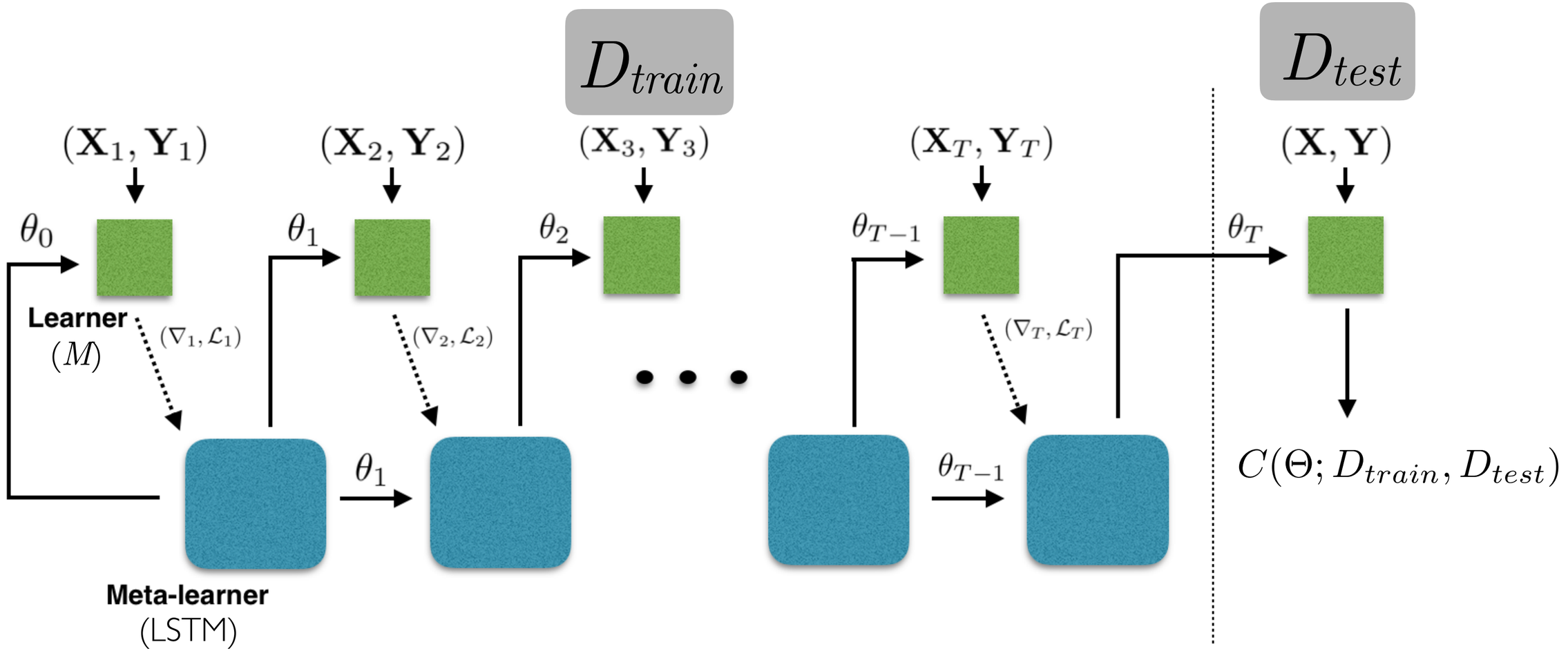
$$i_t = \sigma\left(\mathbf{W}_I \cdot \left[\nabla_{\theta_{t-1}} \mathcal{L}_t, \mathcal{L}_t, \theta_{t-1}, i_{t-1}\right] + \mathbf{b}_I\right)$$ ← *adaptive learning rate*

$$f_t = \sigma\left(\mathbf{W}_F \cdot \left[\nabla_{\theta_{t-1}} \mathcal{L}_t, \mathcal{L}_t, \theta_{t-1}, f_{t-1}\right] + \mathbf{b}_F\right)$$ ← *adaptive weight decay*

- Optimization as a Model for Few-Shot Learning (2017)
  Sachin Ravi and Hugo Larochelle

META-LEARNER LSTM

- Training a "**gradient descent procedure**" applied on some learner M



- Optimization as a Model for Few-Shot Learning (2017)
  Sachin Ravi and Hugo Larochelle

META-LEARNER LSTM

- Training a "**gradient descent procedure**" applied on some learner M
  - LSTM parameters are shared across M's parameters (i.e. treated like a large minibatch)
  - can ignore (stop) gradients through the inputs of the LSTM
  - gradient (and loss) inputs to the Meta-LSTM preprocessed as proposed by Andrychowicz et al. (2016)

$$\nabla^k \rightarrow \begin{cases} \left(\frac{\log(|\nabla|)}{p}, \text{sgn}(\nabla)\right) & \text{if } |\nabla| \geq e^{-p} \\ (-1, e^p \nabla) & \text{otherwise} \end{cases}$$

  - we are careful to avoid "leakage" from batchnorm statistics between meta-train / meta-test sets (sometimes referred to as the "transductive setting")

- Optimization as a Model for Few-Shot Learning (2017)
  Sachin Ravi and Hugo Larochelle

MODEL-AGNOSTIC META-LEARNING

- Training a "**gradient descent procedure**" applied on some learner M
  - MAML proposes not to bother with the LSTM for the gradient descent updates and use constant step-sizes
  - in its most common form (1 step of gradient descent), MAML corresponds to

$$p_\Theta(y'|x', D_{train}) = \mathrm{NNet}(y'|x', \theta - \alpha \nabla_\theta \mathcal{L}(\theta, D_{train}))$$

where we are only learning θ and

- $\mathrm{NNet}(y|\mathbf{x}, \theta')$ is the softmax output for label y of a neural net with parameters θ'
- $\mathcal{L}(\theta, D_{train}) = \frac{1}{D_{train}} \sum_{\substack{(\mathbf{x}_i, y_i) \\ \in D_{train}}} -\log \mathrm{NNet}(y_i|\mathbf{x}_i, \theta)$

- Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks (2017)
  Chelsea Finn, Pieter Abbeel and Sergey Levine

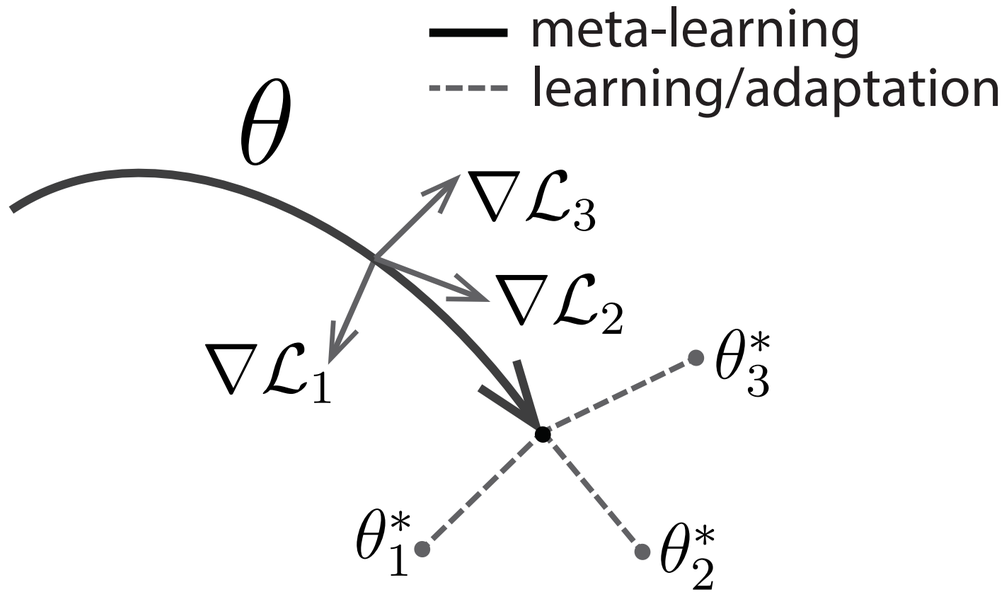
MODEL-AGNOSTIC META-LEARNING

- Training a "**gradient descent procedure**" applied on some learner M
  - better results are also reported by the so-called *bias transformation* architecture

    One-Shot Visual Imitation Learning via Meta-Learning (2017)
    Finn, Yu, Zhang, Abbeel, Levine
    - concatenates to one of the layers a trainable parameter vector, for instance to the input layer $[\mathbf{x}_i, \theta_b]$
    - decouples the updates of the bias and weights of that layer
  - with bias transformation, can show that single gradient descent update MAML is a universal approximator over functions mapping D_{train} and $\mathbf{x}$ to any label y, for a sufficiently deep ReLU network and certain losses
    - Meta-Learning and Universality: Deep Representations and Gradient Descent can Approximate any Learning Algorithm (2018)
      Finn and Levine
- Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks (2017)
  Chelsea Finn, Pieter Abbeel and Sergey Levine

MODEL-AGNOSTIC META-LEARNING

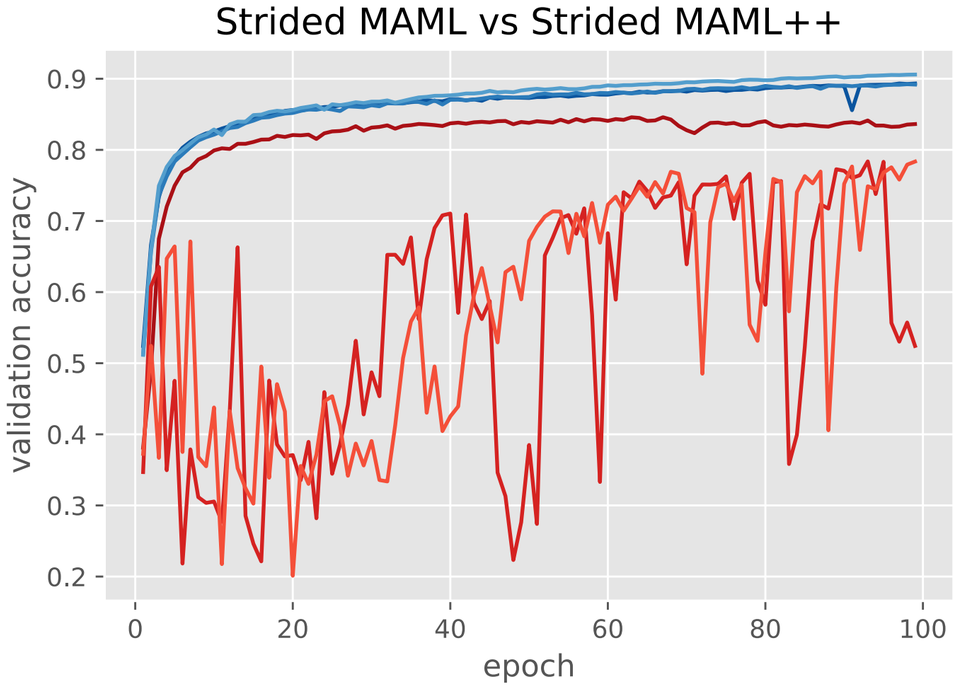
- Training a "**gradient descent procedure**" applied on some learner M
  - MAML is also related to hierarchical Bayesian inference
    - Recasting Gradient-Based Meta-Learning as Hierarchical Bayes (2018)
      Grant, Finn, Levine, Darrell, Griffiths



- Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks (2017)
  Chelsea Finn, Pieter Abbeel and Sergey Levine

MODEL-AGNOSTIC META-LEARNING

- Training a "**gradient descent procedure**" applied on some learner M
  - MAML training can be instable, but many "tricks" can help (MAML++)
    - optimize sum of losses on test-set (query set) at each step
    - switch on full-order gradients later during training
    - use a trainable learning rate per layer and per time ste
    - use running and per-step statistics for batch norm
    - use per-step batch norm parameters
  - See this paper for more details:
    - How to train your MAML (2018),
      Antreas Antoniou, Harrison Edwards, Amos Storkey



- Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks (2017)
  Chelsea Finn, Pieter Abbeel and Sergey Levine

CHOOSING A META-LEARNER

- How to parametrize learning algorithms?

- Two approaches to defining a meta-learner
  - Take inspiration from a known learning algorithm
    - kNN/kernel machine: Matching networks (Vinyals et al. 2016)
    - Gaussian classifier: Prototypical Networks (Snell et al. 2017)
    - Gradient Descent: Meta-Learner LSTM (Ravi & Larochelle, 2017) , MAML (Finn et al. 2017)
  - **Derive it from a black box neural network**
    - MANN (Santoro et al. 2016)
    - SNAIL (Mishra et al. 2018)

LEARNING PROBLEM STATEMENT

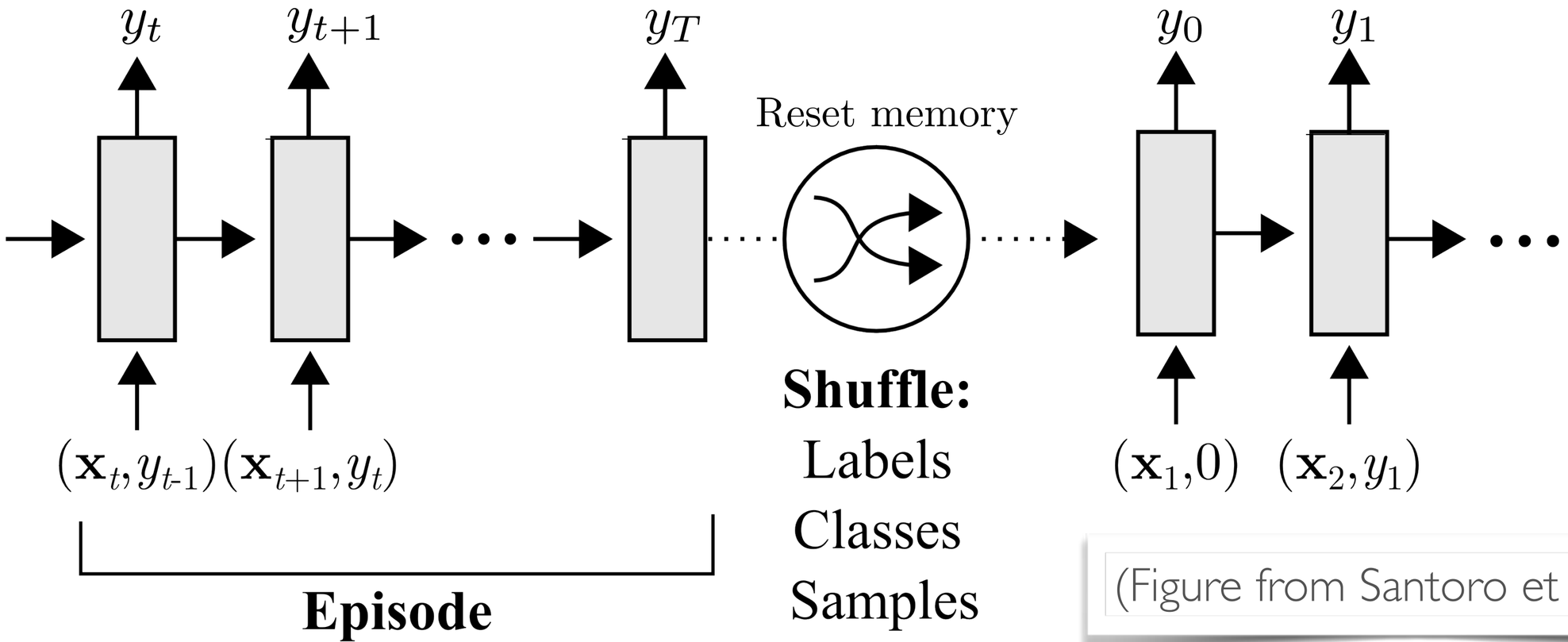
- Assuming a probabilistic model M over labels, the cost per episode can become

$$C(\Theta; D_{train}, D_{test}) = \frac{1}{|D_{test}|} \sum_{\substack{(\mathbf{x}'_i, y'_i) \\ \in D_{test}}} -\log p_\Theta(y'_i | \mathbf{x}'_i, D_{train})$$

- Depending on the choice of meta-learner, $p_\Theta(y'|\mathbf{x}', D_{train})$ will take a different form

BLACK-BOX META-LEARNER

- Frame meta-learning as sequence labeling with correct labels as delayed inputs

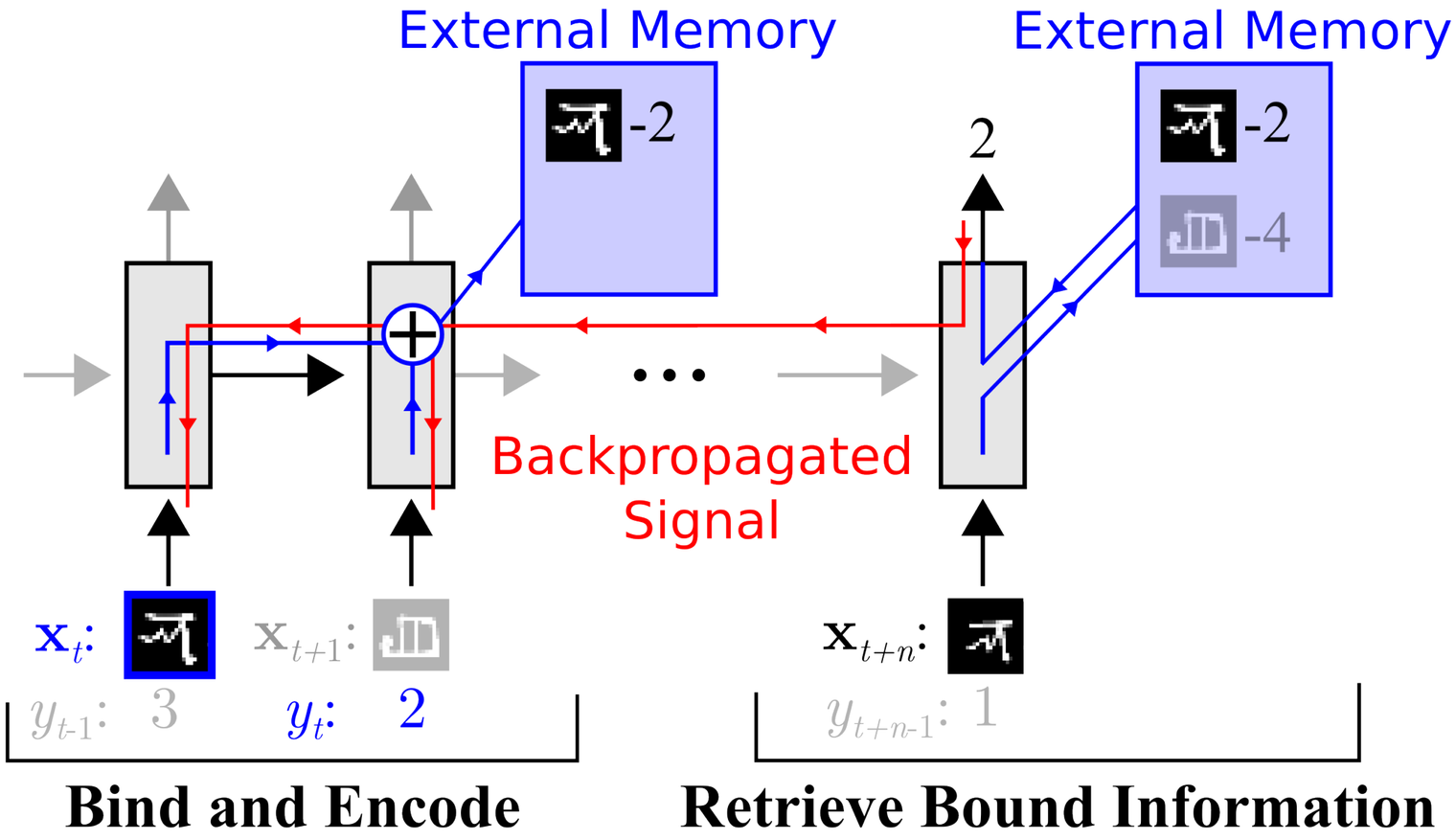


(Figure from Santoro et al. 2016)

- Learning to learn using gradient descent (2001)
  Sepp Hochreiter, A. Steven Younger, and Peter R. Conwell

MEMORY-AUGMENTED NEURAL NETWORK

- Training a **neural Turing machine** to learn a learning algorithm



- One-shot learning with memory-augmented neural networks (2016)
  Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy P. Lillicrap

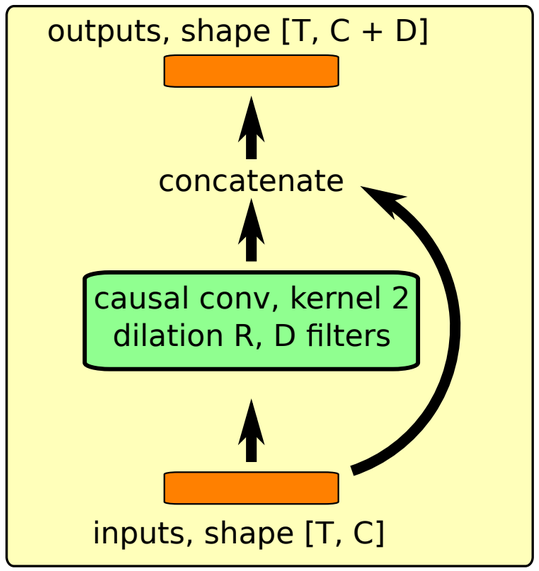
MEMORY-AUGMENTED NEURAL NETWORK

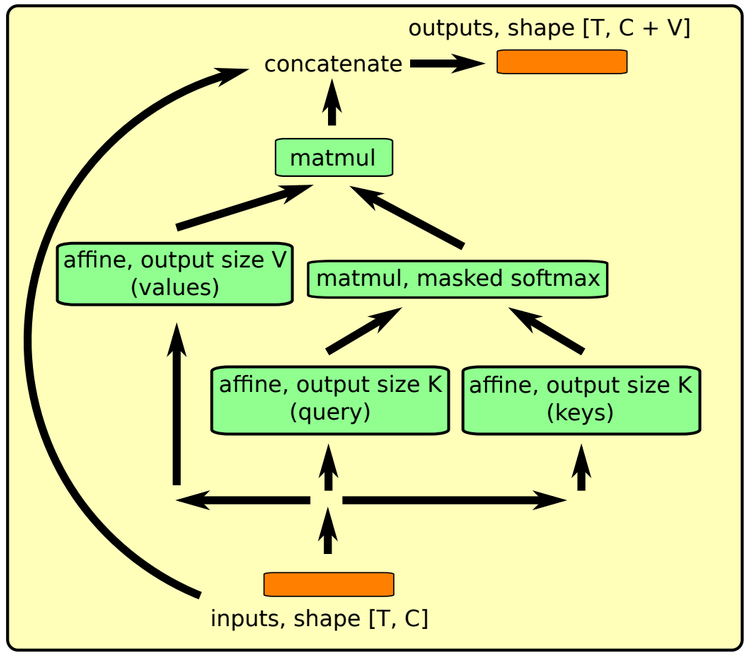
- Training a **neural Turing machine** to learn a learning algorithm
  - required adapting the original neural Turing machine, using a fairly non-trivial Least Recently Used Access (LRUA) writing mechanism
  - only shown to work on Omniglot dataset (and not on the harder split used by Lake et al. 2015)

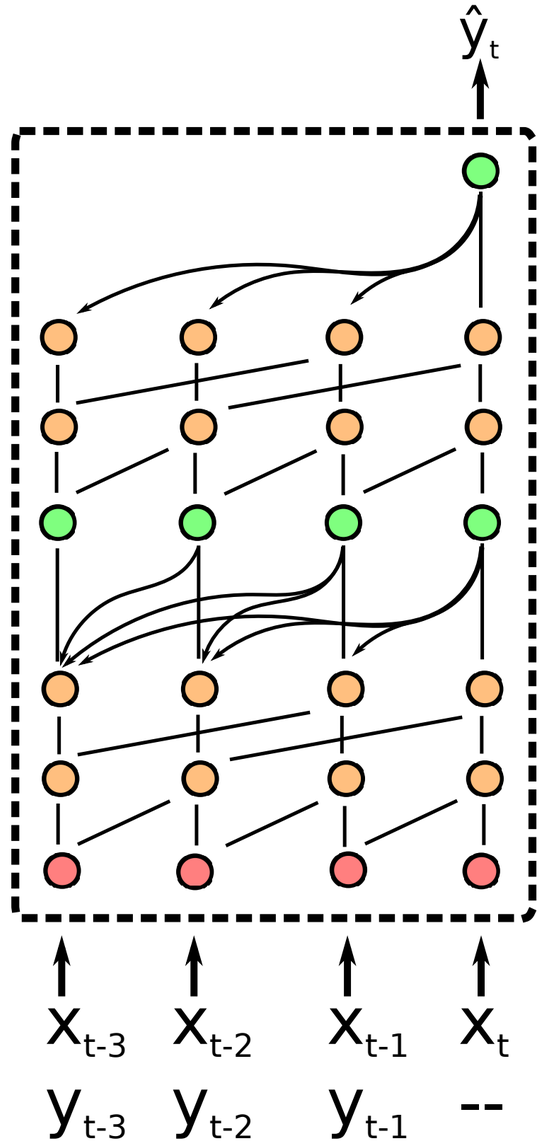
- One-shot learning with memory-augmented neural networks (2016)
  Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy P. Lillicrap

SIMPLE NEURAL ATTENTIVE LEARNER

- Training a **convolutional/attentional network** to learn
  - alternates between dilated convolutional layers and attentional layers
  - when inputs are images, an convolutional embedding network is used to map to a vector space



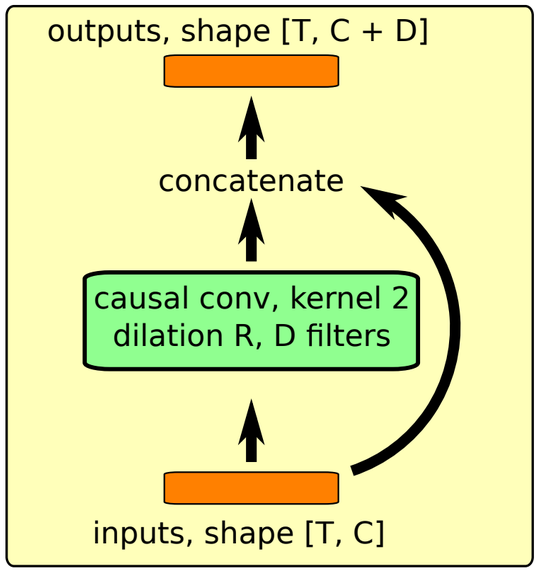


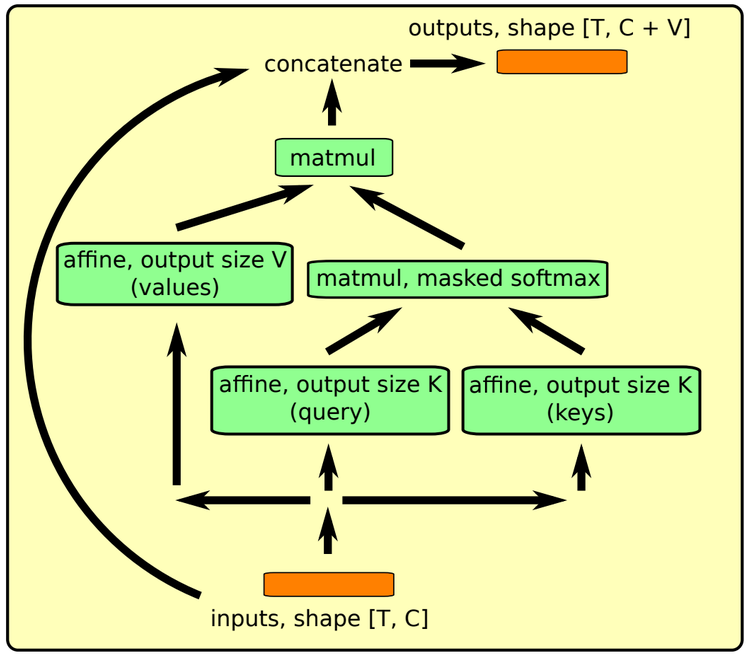


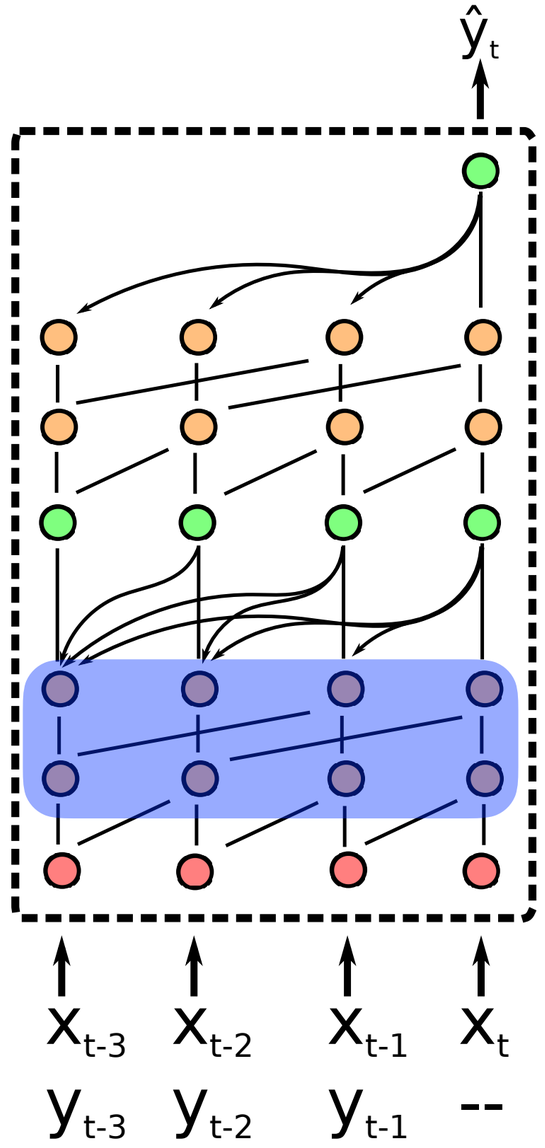
- A Simple Neural Attentive Meta-Learner (2018)
  Nikhil Mishra, Mostafa Rohaninejad, Xi Chen and Pieter Abbeel

SIMPLE NEURAL ATTENTIVE LEARNER

- Training a **convolutional/attentional network** to learn
  - alternates between dilated convolutional layers and attentional layers
  - when inputs are images, an convolutional embedding network is used to map to a vector space



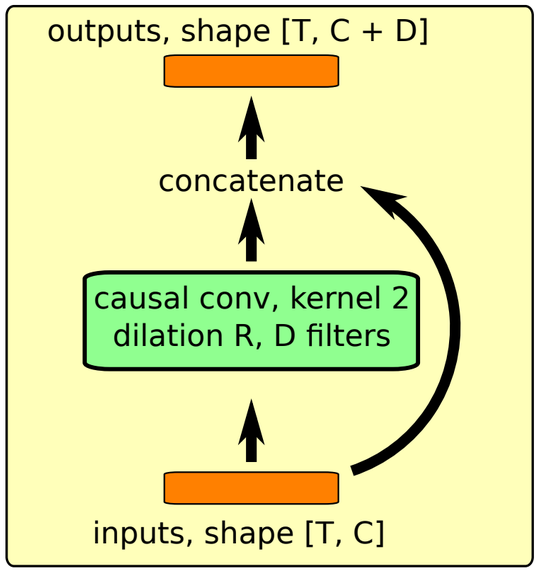


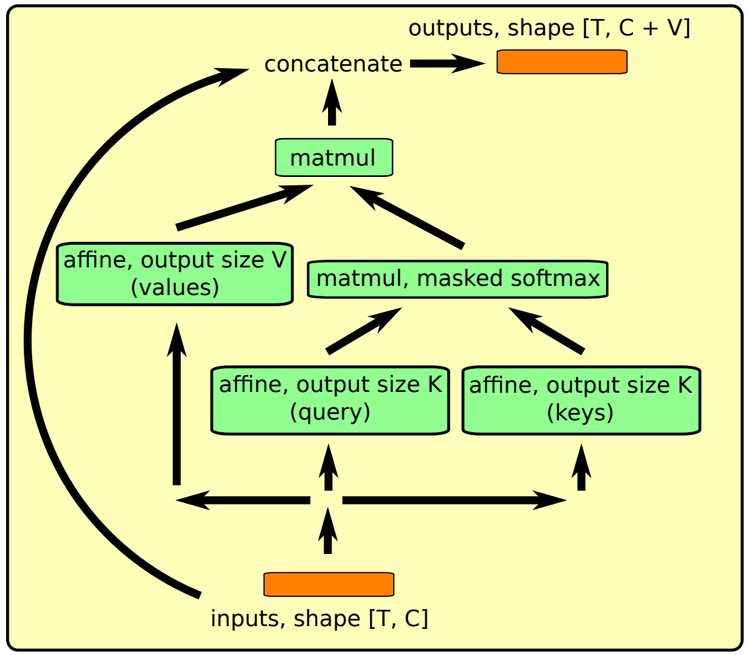


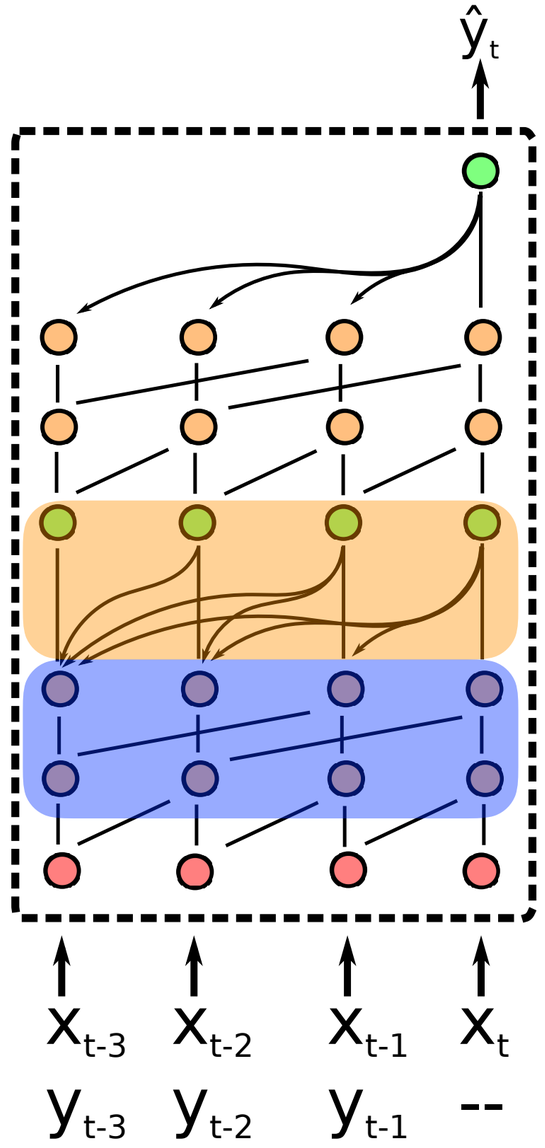
- A Simple Neural Attentive Meta-Learner (2018)
  Nikhil Mishra, Mostafa Rohaninejad, Xi Chen and Pieter Abbeel

SIMPLE NEURAL ATTENTIVE LEARNER

- Training a **convolutional/attentional network** to learn
  - alternates between dilated convolutional layers and attentional layers
  - when inputs are images, an convolutional embedding network is used to map to a vector space



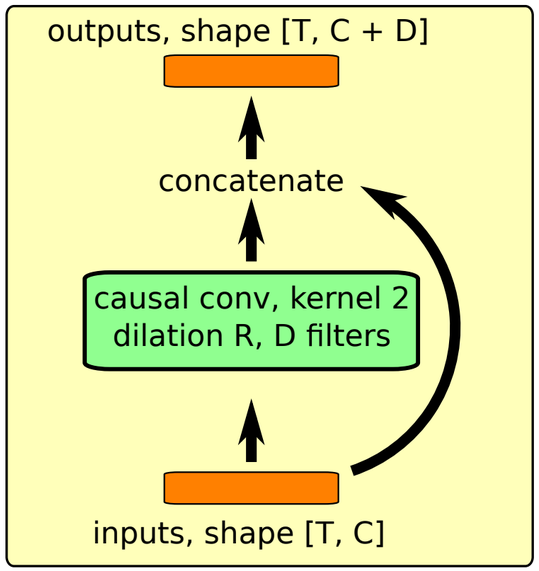


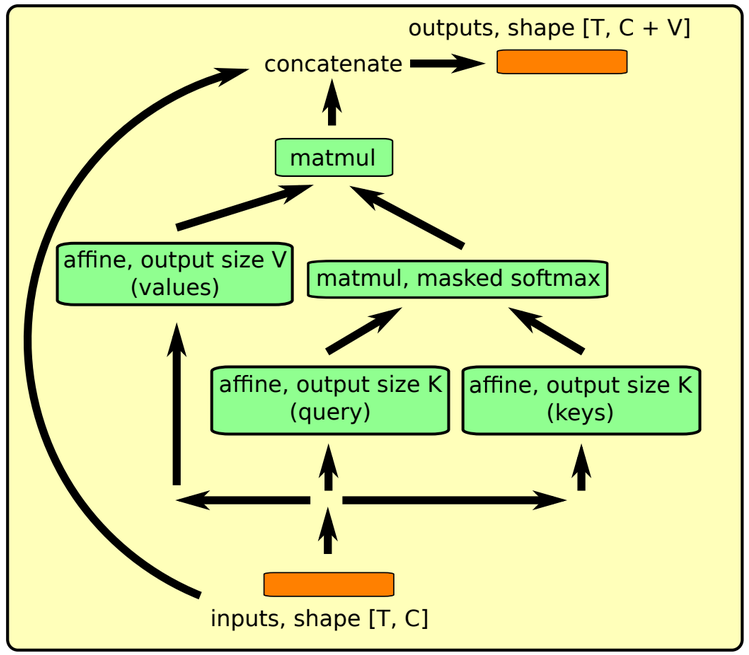


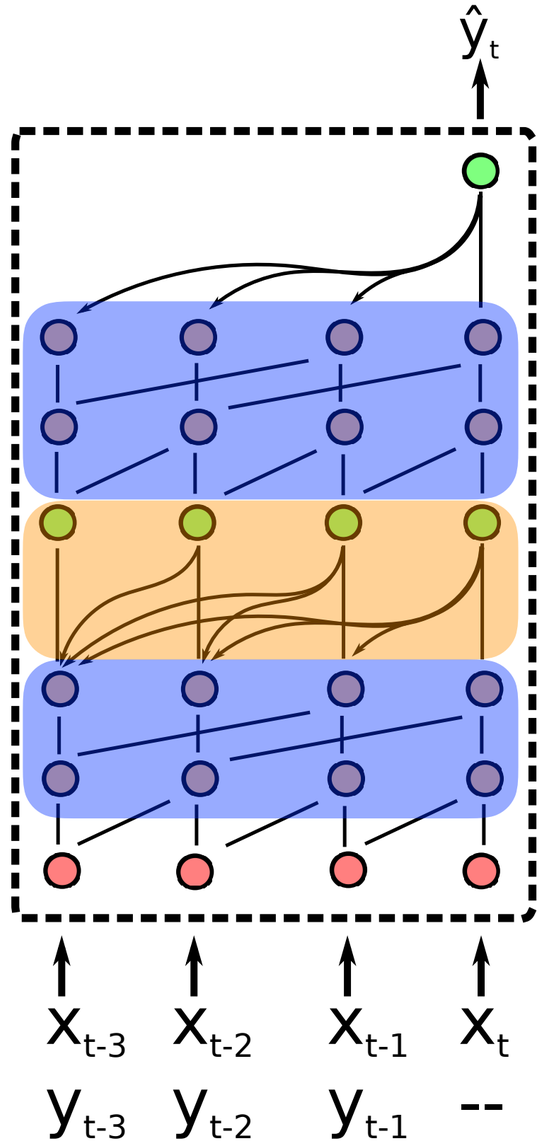
- A Simple Neural Attentive Meta-Learner (2018)
  Nikhil Mishra, Mostafa Rohaninejad, Xi Chen and Pieter Abbeel

SIMPLE NEURAL ATTENTIVE LEARNER

- Training a **convolutional/attentional network** to learn
  - alternates between dilated convolutional layers and attentional layers
  - when inputs are images, an convolutional embedding network is used to map to a vector space



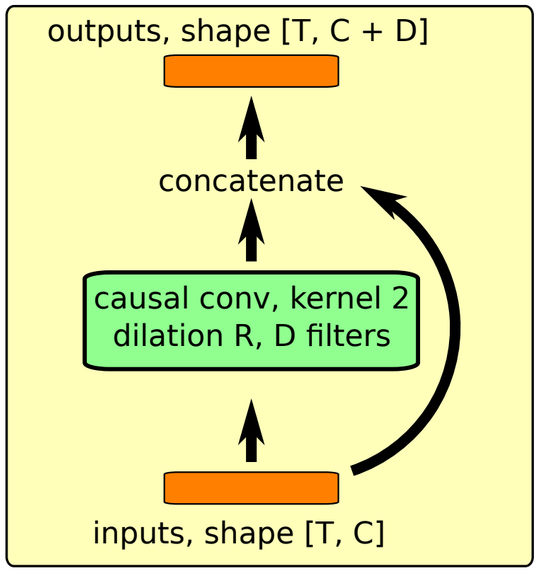


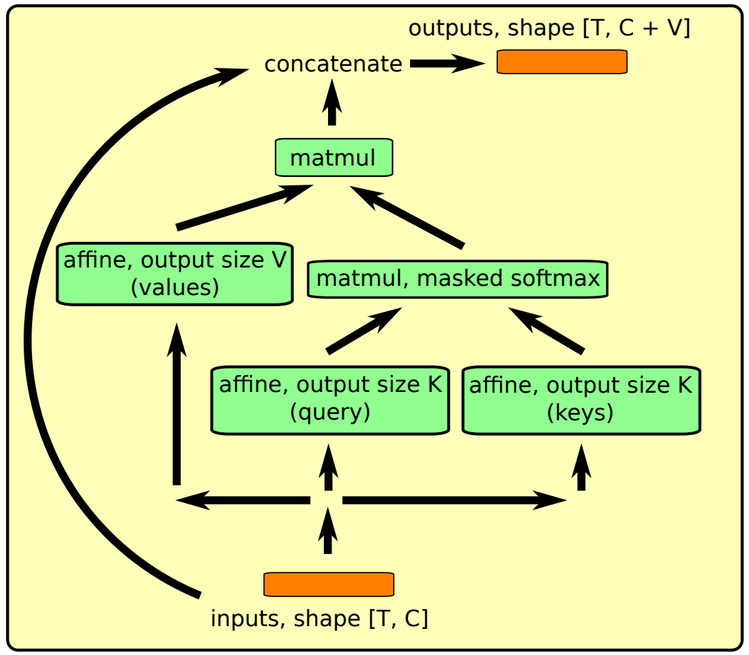


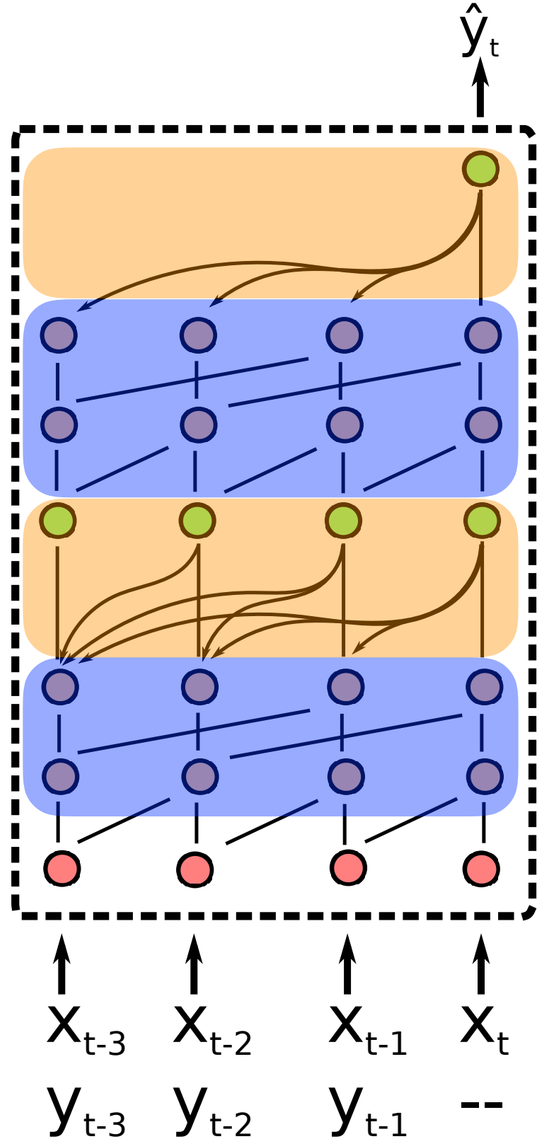
- A Simple Neural Attentive Meta-Learner (2018)
  Nikhil Mishra, Mostafa Rohaninejad, Xi Chen and Pieter Abbeel

SIMPLE NEURAL ATTENTIVE LEARNER

- Training a **convolutional/attentional network** to learn
  - alternates between dilated convolutional layers and attentional layers
  - when inputs are images, an convolutional embedding network is used to map to a vector space







- A Simple Neural Attentive Meta-Learner (2018)
  Nikhil Mishra, Mostafa Rohaninejad, Xi Chen and Pieter Abbeel

EXPERIMENT

- Mini-ImageNet (split used in Ravi & Larochelle, 2017)
  - random subset of 100 classes (64 training, 16 validation, 20 testing)
  - random sets D_{train} are generated by randomly picking 5 classes from class subset

Model	5-class	
	1-shot	5-shot
Baseline-finetune	$28.86 \pm 0.54\%$	$49.79 \pm 0.79\%$
Baseline-nearest-neighbor	$41.08 \pm 0.70\%$	$51.04 \pm 0.65\%$
Matching Network	$\mathbf{43.40 \pm 0.78\%}$	$51.09 \pm 0.71\%$
Matching Network FCE	**43.56% ± 0.84%**	$55.31\% \pm 0.73\%$
Meta-Learner LSTM (OURS)	**43.44% ± 0.77%**	**60.60% ± 0.71%**

EXPERIMENT

- Mini-ImageNet (split used in Ravi & Larochelle, 2017)
  - random subset of 100 classes (64 training, 16 validation, 20 testing)
  - random sets D_{train} are generated by randomly picking 5 classes from class subset

Model	5-class	
	1-shot	5-shot
Prototypical Nets (Snell et al.)	49.42% ± 0.78%	**68.20% ± 0.66%**
MAML (Finn et al.)	48.70% ± 1.84%	63.10% ± 0.92%
SNAIL (Mishra et al.)	**55.71% ± 0.99%**	**68.88% ± 0.98%**
Matching Network FCE	43.56% ± 0.84%	55.31% ± 0.73%
Meta-Learner LSTM (OURS)	43.44% ± 0.77%	60.60% ± 0.71%

REMAINING CHALLENGES

- Coming up with a realistic definition of distributions over problems
  - varying number of classes / examples per class (meta-training vs. meta-testing) ?
  - semantic differences between meta-training vs. meta-testing classes ?
  - overlap in meta-training vs. meta-testing classes (see recent "low-shot" literature) ?

- Going beyond supervised classification
  - semi-supervised ?
  - structured output ?

MERCI !