CS 4803 / 7643: Deep Learning

Topics:

- (Continue) Low-label ML Formulations

Zsolt Kira
Georgia Tech

Administrative

- Projects!
  - Poster details out on piazza
  - Note: No late days for anything project related!
  - Also note: Keep track of your GCP usage and costs! Set limits on spending

Meta-Learning for Few-Shot Recognition

- **Key idea**: We want to learn from a *few* examples (called the **support set**) to make predictions on **query** set for **novel** classes
  - Assume: We have larger labeled dataset for a different set of categories (**base classes**)

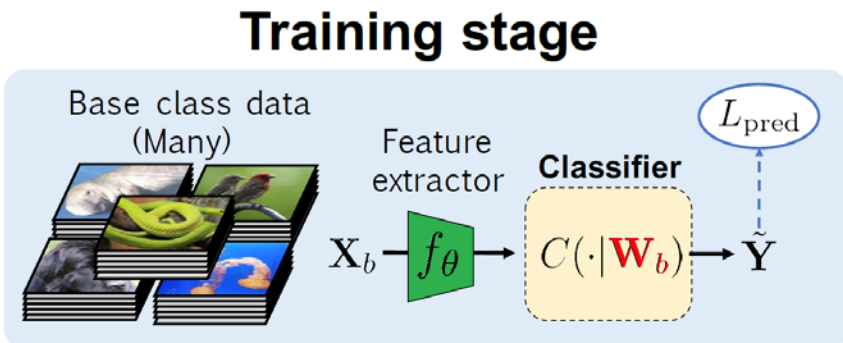
- How do we test this?
  - N-way k-shot test
  - k: Number of examples in **support set**
  - N: Number of "confusers" that we have to choose target class among

Target

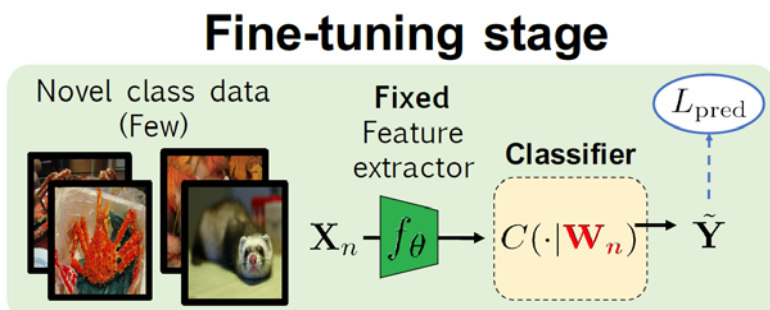
Query Set

Normal Approach

- **Do what we always do: Fine-tuning**
  - Train classifier on base classes



  - Freeze features
  - Learn classifier weights for new classes using few amounts of labeled data (during "*inference*" time!)



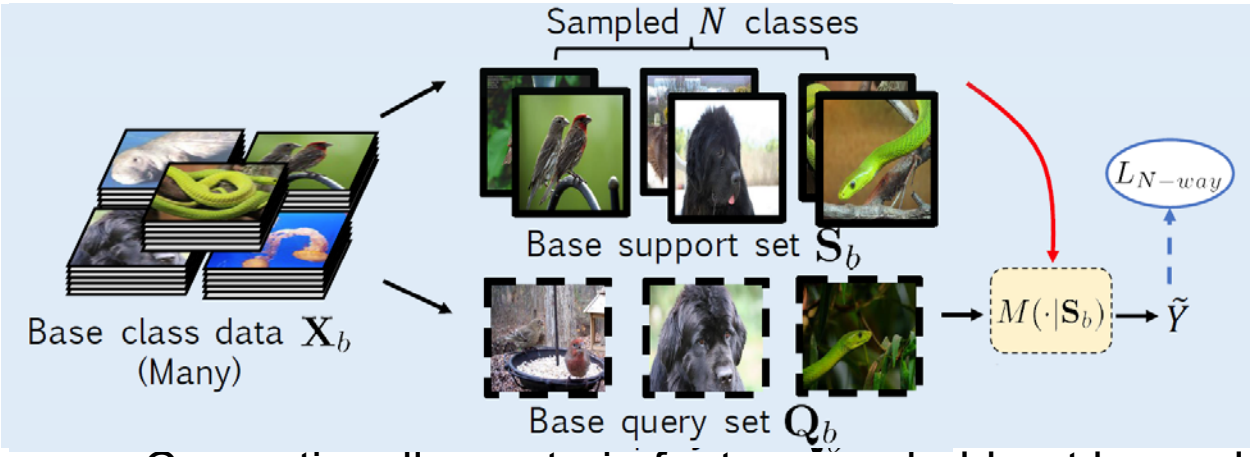
A Closer Look at Few-shot Classification, Wei-Yu Chen, Yen-Cheng Liu, Zsolt Kira, Yu-Chiang Frank Wang, Jia-Bin Huang

Cons of Normal Approach

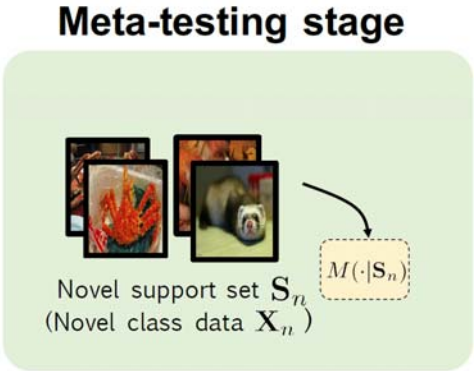
- The training we do on the base classes does not factor the task into account
- No notion that we will be performing a bunch of N-way tests
- Idea: simulate what we will see during test time

Meta-Training Approach

- Set up a set of smaller tasks *during training* which simulates what we will be doing during testing

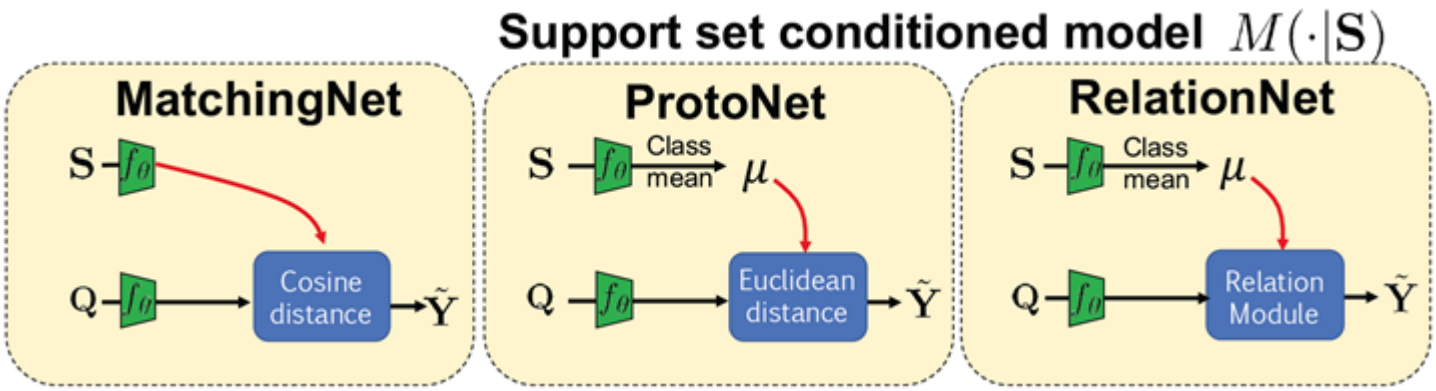


  - Can optionally pre-train features on held-out base classes (not typical)
- Testing stage is now the same, but with new classes



Meta-Learning Approaches

- Learning a model conditioned on support set $M(\cdot|\mathbf{S})$



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)

Meta-Learner

- How to parametrize learning algorithms?

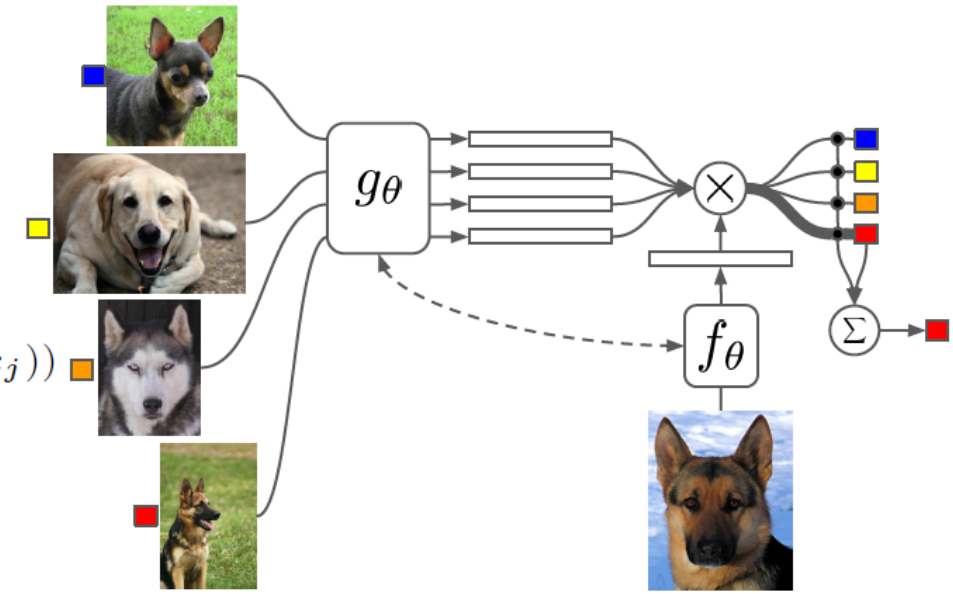
- Two approaches to defining a meta-learner
  - **Take inspiration from a known learning algorithm**
    - kNN/kernel machine: Matching networks (Vinyals et al. 2016)
    - Gaussian classifier: Prototypical Networks (Snell et al. 2017)
    - Gradient Descent: Meta-Learner LSTM (Ravi & Larochelle, 2017) , 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**"

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

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



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

Prototypical Networks

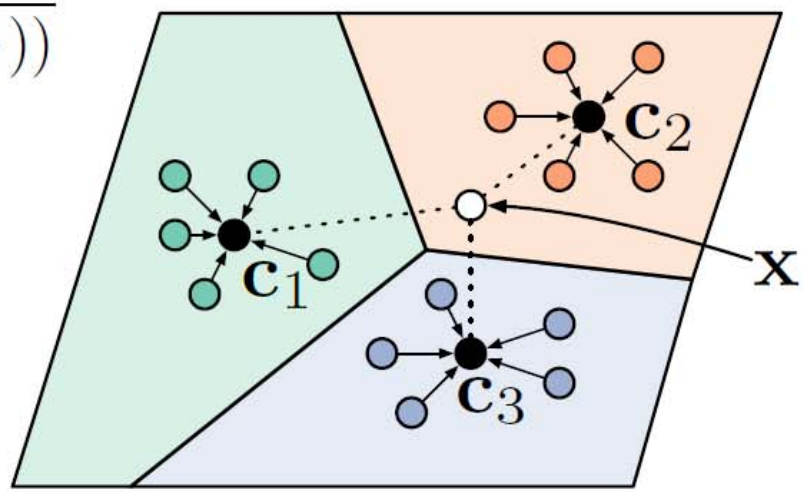
- Training a "**prototype extractor**"

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

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

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

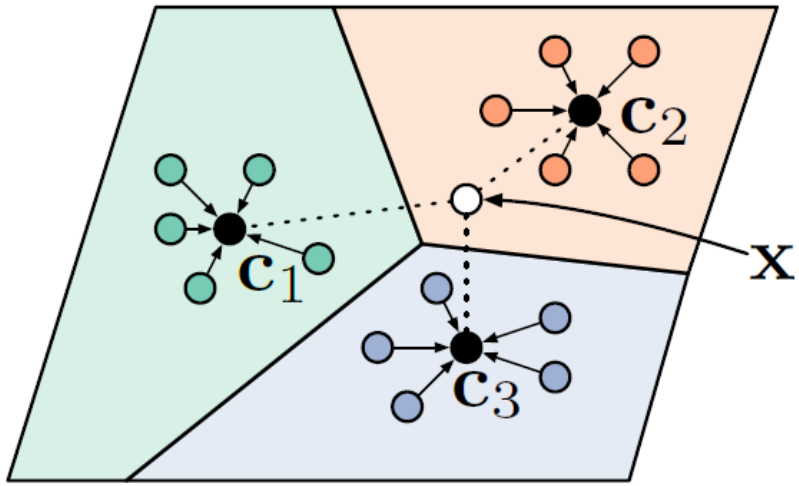
$$\phi \equiv \Theta$$



- 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
  - prototype vectors are equivalent to output weights of a neural network
  - Snell et al. find that using more classes in the meta-training episodes compared to meta-testing works better



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

More Sophisticated Meta-Learning Approaches

- Learn gradient descent:
  - Parameter initialization and update rules

- Learn just an initialization and use normal gradient descent (MAML)

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:

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

- 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:

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

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

- 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 ← 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:

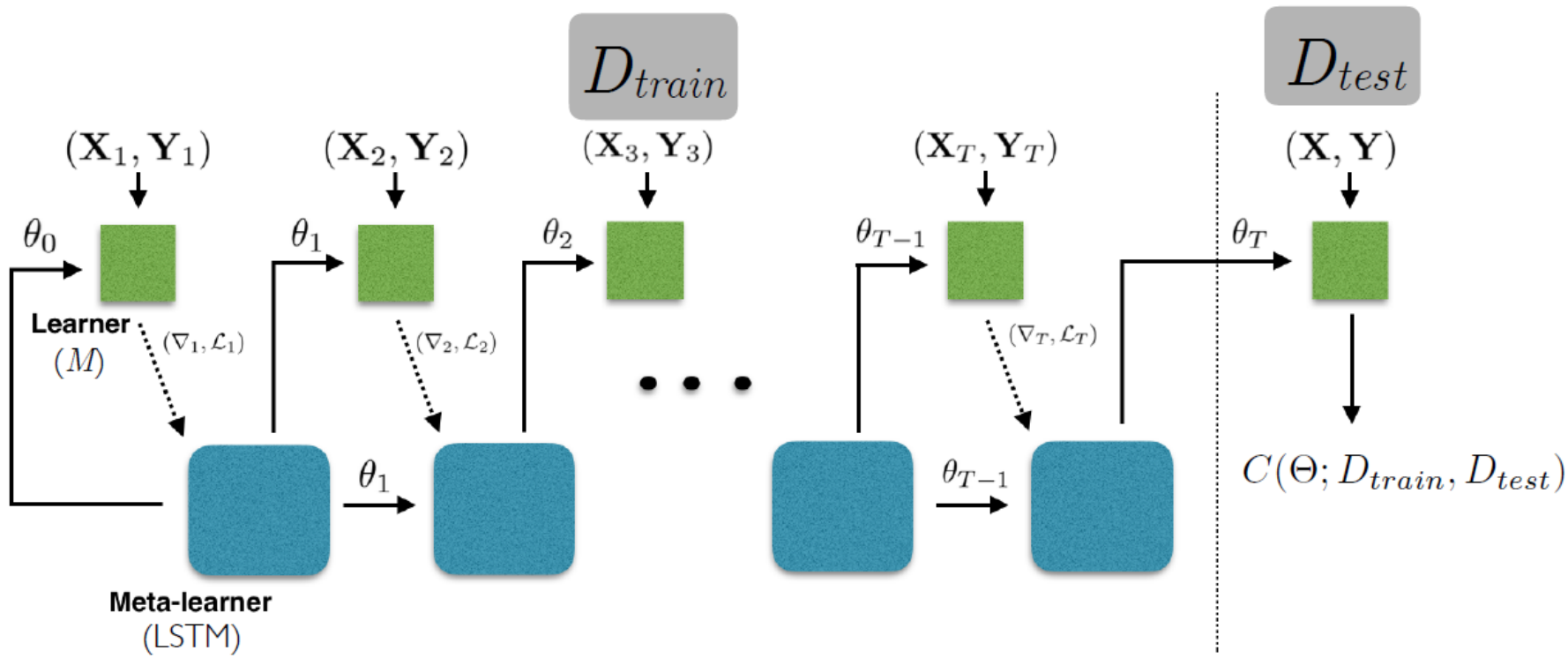
$$i_t = \sigma\left(\mathbf{W}_I \cdot \left[\nabla_{\theta_{t-1}} \mathcal{L}_t, \mathcal{L}_t, \theta_{t-1}, i_{t-1}\right] + \mathbf{b}_I\right)$$ ← *adaptive learning rate*

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

Algorithm 1 Train Meta-Learner

Input: Meta-training set $\mathscr{D}_{meta-train}$, Learner M with parameters θ, Meta-Learner R with parameters Θ.

$\Theta_0 \leftarrow$ random initialization

for $d = 1, n$ **do**

  $D_{train}, D_{test} \leftarrow$ random dataset from $\mathscr{D}_{meta-train}$

  $\theta_0 \leftarrow c_0$ ▷ Intialize learner parameters

  for $t = 1, T$ **do**

    $\mathbf{X}_t, \mathbf{Y}_t \leftarrow$ random batch from D_{train}

    $\mathcal{L}_t \leftarrow \mathcal{L}(M(\mathbf{X}_t; \theta_{t-1}), \mathbf{Y}_t)$ ▷ Get loss of learner on train batch

    $c_t \leftarrow R((\nabla_{\theta_{t-1}} \mathcal{L}_t, \mathcal{L}_t); \Theta_{d-1})$ ▷ Get output of meta-learner using Equation 2

    $\theta_t \leftarrow c_t$ ▷ Update learner parameters

  end for

  $\mathbf{X}, \mathbf{Y} \leftarrow D_{test}$

  $\mathcal{L}_{test} \leftarrow \mathcal{L}(M(\mathbf{X}; \theta_T), \mathbf{Y})$ ▷ Get loss of learner on test batch

  Update Θ_d using $\nabla_{\Theta_{d-1}} \mathcal{L}_{test}$ ▷ Update meta-learner parameters

end for

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)

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

  - 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 (MAML)

- Training a "**gradient descent procedure**" applied on some learner M
  - MAML proposes not to bother with training an LSTM for the gradient descent updates and constant step-size updates

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

Slide Credit: Hugo Larochelle

Model-Agnostic Meta-Learning (MAML)

Algorithm 1 Model-Agnostic Meta-Learning

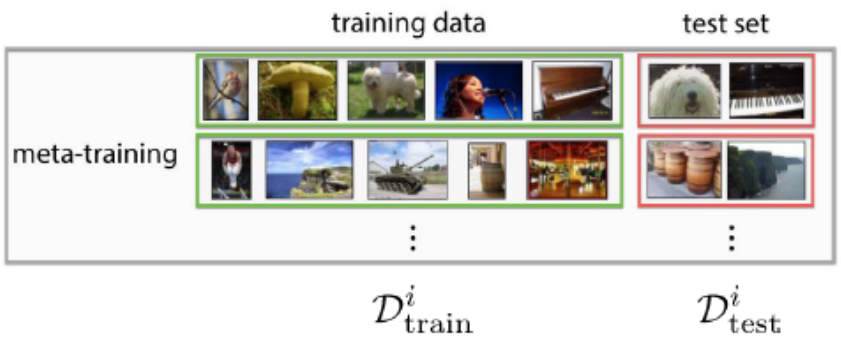
Require: $p(\mathcal{T})$: distribution over tasks
Require: α, β: step size hyperparameters

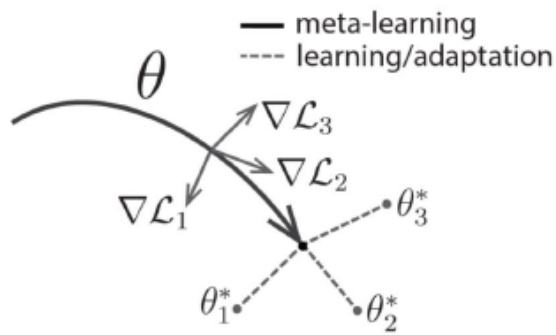
```
randomly initialize θ
while not done do
    Sample batch of tasks T_i ~ p(T)
    for all T_i do
        Evaluate ∇_θ L_{T_i}(f_θ) with respect to K examples
        Compute adapted parameters with gradient descent: θ'_i = θ − α ∇_θ L_{T_i}(f_θ)
    end for
    Update θ ← θ − β ∇_θ Σ_{T_i ~ p(T)} L_{T_i}(f_{θ'_i})
end while
```

Note: the meta-update is using different set of data.

Model-Agnostic Meta-Learning (MAML)

a general recipe:





$$\theta \leftarrow \theta - \beta \sum_i \nabla_\theta \underbrace{\mathcal{L}(\theta - \alpha \nabla_\theta \mathcal{L}(\theta, \mathcal{D}^i_{\text{train}}), \mathcal{D}^i_{\text{test}})}_{\text{“meta-loss” for task } i}$$

* in general, can take more than one gradient step here

** we often use 4 – 10 steps

Chelsea Finn

Finn et al., “Model-Agnostic Meta-Learning”

Model-Agnostic Meta-Learning (MAML)

supervised learning: $f(x) \rightarrow y$

supervised meta-learning: $f(\mathcal{D}_{\text{train}}, x) \rightarrow y$

model-agnostic meta-learning: $f_{\text{MAML}}(\mathcal{D}_{\text{train}}, x) \rightarrow y$

$$f_{\text{MAML}}(\mathcal{D}_{\text{train}}, x) = f_{\theta'}(x)$$

$$\theta' = \theta - \alpha \sum_{(x,y) \in \mathcal{D}_{\text{train}}} \nabla_\theta \mathcal{L}(f_\theta(x), y)$$

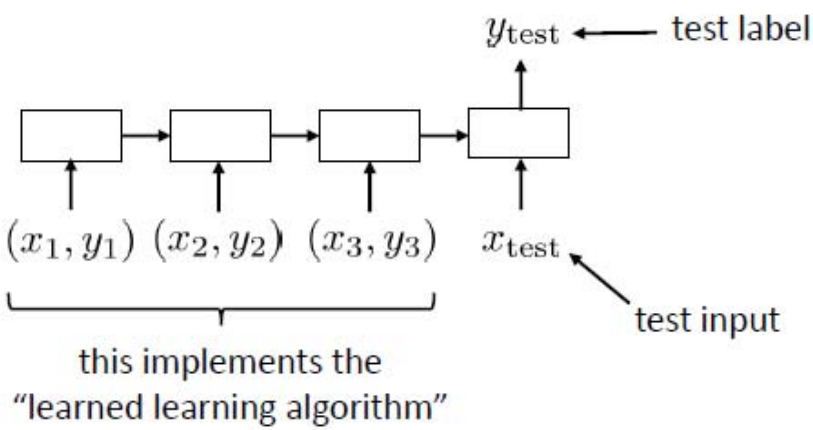
Just another computation graph…

Can implement with any autodiff package (e.g., TensorFlow)

Slide Credit: Sergey Levine

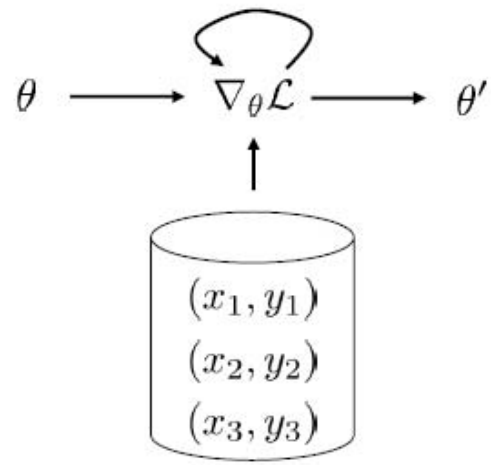
Comparison

RNN-based meta-learning



- Does it converge?
  - Kind of?
- What does it converge to?
  - Who knows…
- What to do if it's not good enough?
  - Nothing…

MAML



- Does it converge?
  - Yes (it's gradient descent…)
- What does it converge to?
  - A local optimum (it's gradient descent…)
- What to do if it's not good enough?
  - Keep taking gradient steps (it's gradient descent…)

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)

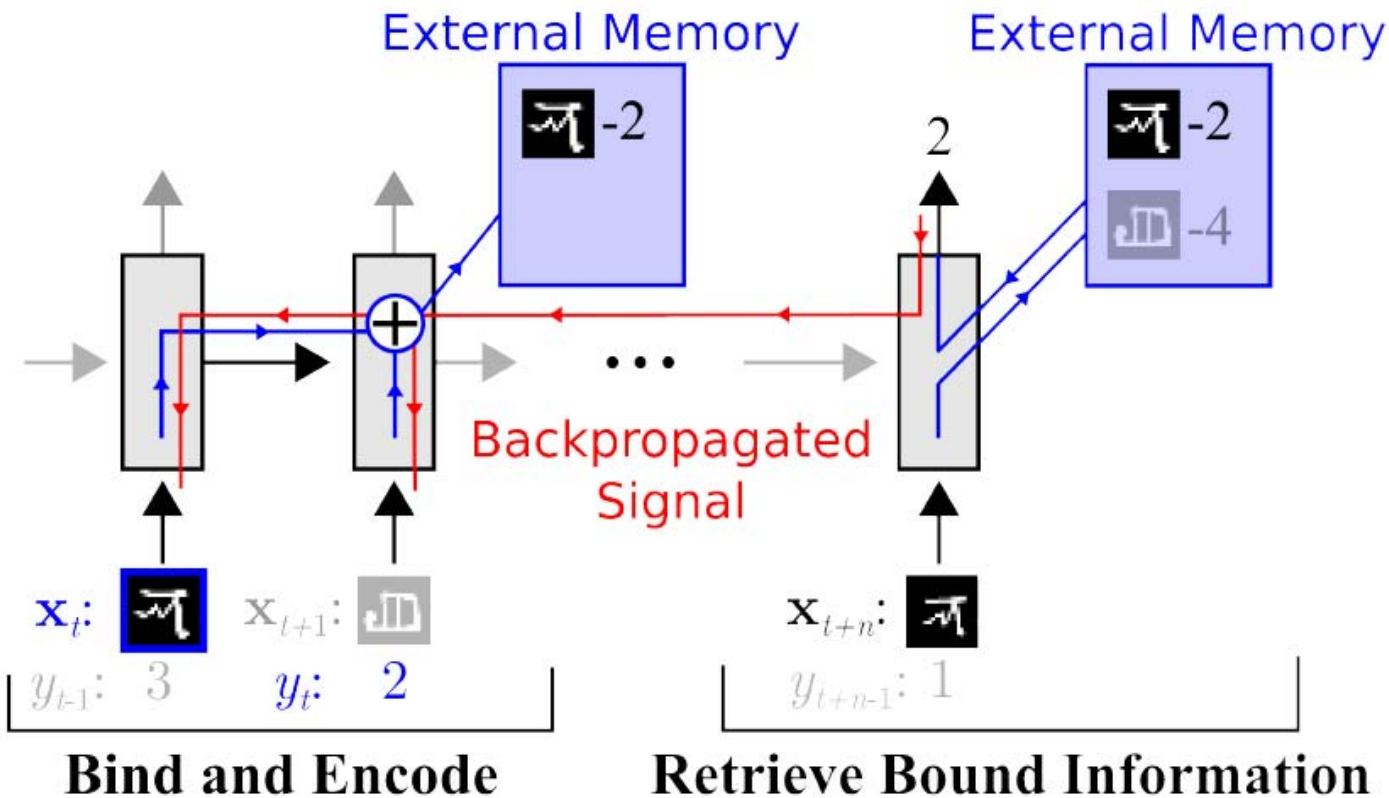
Experiments

- 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%

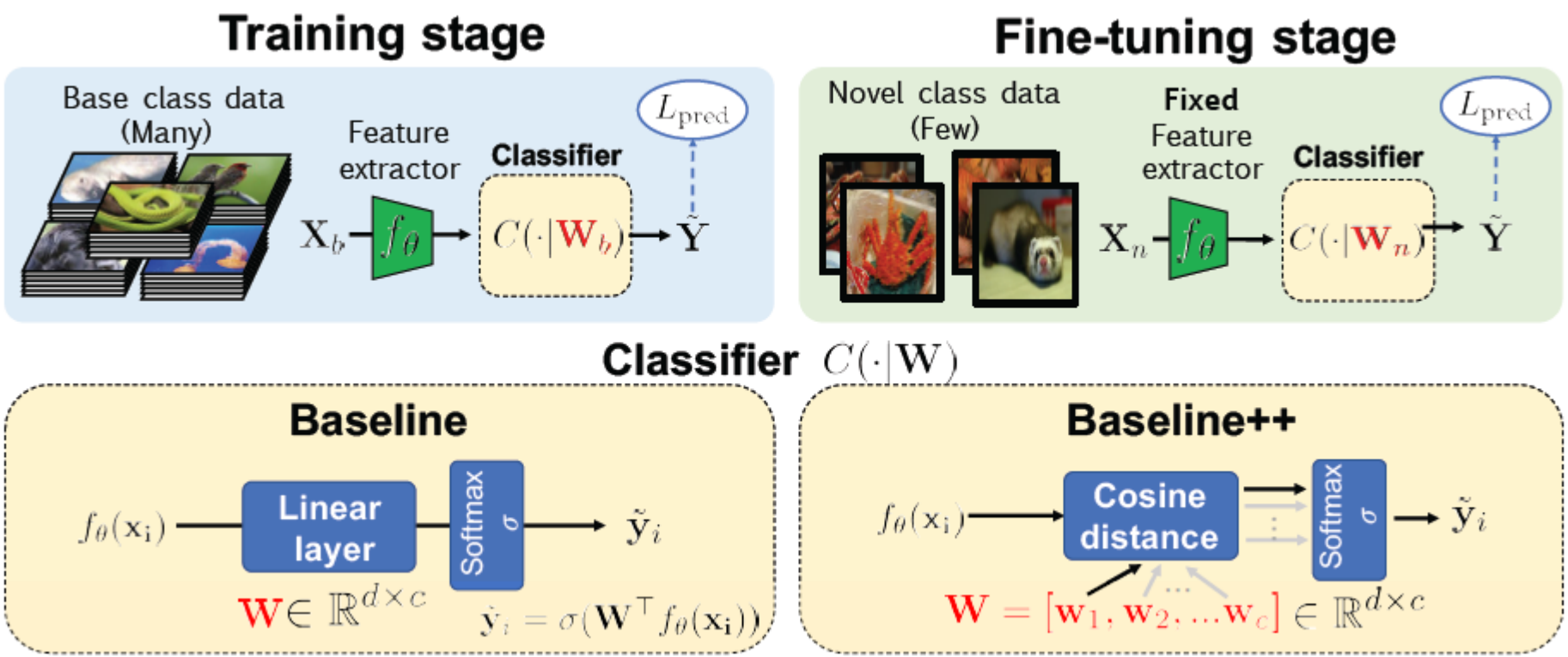
Memory-Augmented Neural Network

- 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

But beware



A Closer Look at Few-shot Classification,

Wei-Yu Chen, Yen-Cheng Liu, Zsolt Kira, Yu-Chiang Frank Wang, Jia-Bin Huang

Table 2: **Few-shot classification results for both the *mini*-ImageNet and *CUB* datasets.** The Baseline++ consistently improves the Baseline model by a large margin and is competitive with the state-of-the-art meta-learning methods. All experiments are from 5-way classification with a Conv-4 backbone and data augmentation.

	CUB		***mini*-ImageNet**	
Method	**1-shot**	**5-shot**	**1-shot**	**5-shot**
Baseline	47.12 ± 0.74	64.16 ± 0.71	42.11 ± 0.71	62.53 ± 0.69
Baseline++	60.53 ± 0.83	79.34 ± 0.61	48.24 ± 0.75	66.43 ± 0.63
MatchingNet Vinyals et al. (2016)	61.16 ± 0.89	72.86 ± 0.70	48.14 ± 0.78	63.48 ± 0.66
ProtoNet Snell et al. (2017)	51.31 ± 0.91	70.77 ± 0.69	44.42 ± 0.84	64.24 ± 0.72
MAML Finn et al. (2017)	55.92 ± 0.95	72.09 ± 0.76	46.47 ± 0.82	62.71 ± 0.71
RelationNet Sung et al. (2018)	62.45 ± 0.98	76.11 ± 0.69	49.31 ± 0.85	66.60 ± 0.69

Distribution Shift

- What if there is a distribution shift (cross-domain)?

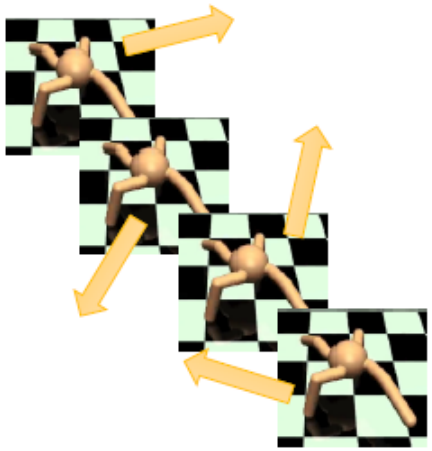
- **Lesson:** Methods that are successful *within-domain* might be worse *across domains*!

	mini-ImageNet →CUB
Baseline	**65.57±0.70**
Baseline++	62.04±0.76
MatchingNet	53.07±0.74
ProtoNet	62.02±0.70
MAML	51.34±0.72
RelationNet	57.71±0.73

Table 3: **5-shot accuracy under the cross-domain scenario with a ResNet-18 backbone.** Baseline outperforms all other methods under this scenario.

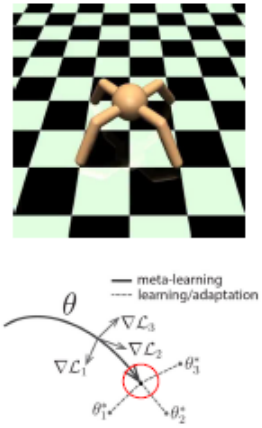
Distribution Shift

Let's Talk about Meta-Overfitting

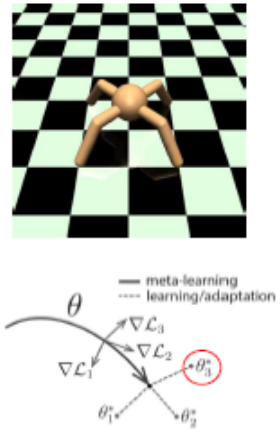


- Meta learning requires task distributions
- When there are too few meta-training tasks, we can *meta-overfit*
- Specifying task distributions is hard, especially for meta-RL!
- Can we propose tasks *automatically*?

after MAML training



after 1 gradient step

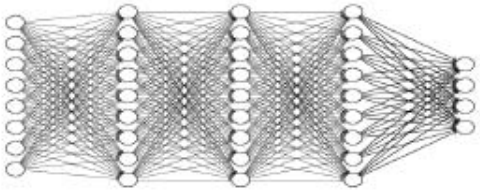


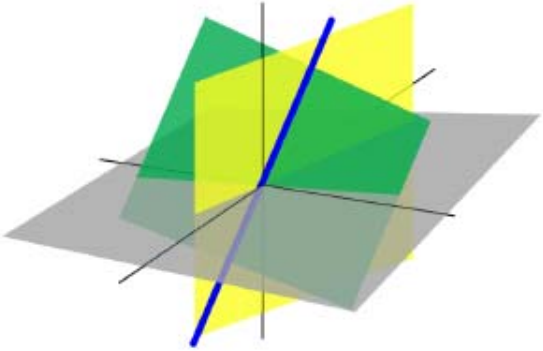
Random Task Proposals

- Use randomly initialize discriminators for reward functions

$$R(s, z) = \log p_D(z|s)$$

D → randomly initialized network

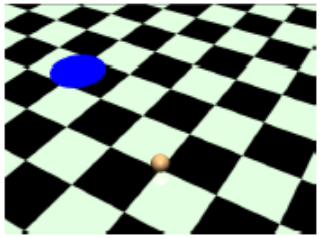


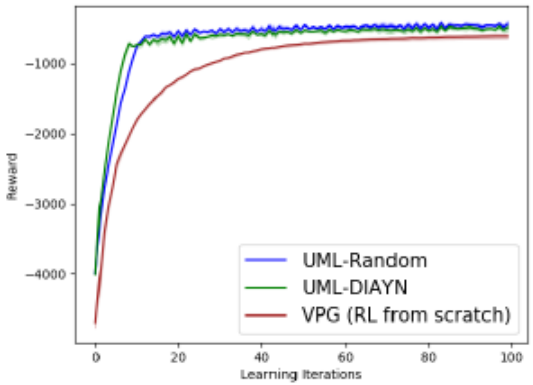


- Important: Random functions over state space, **not** random policies

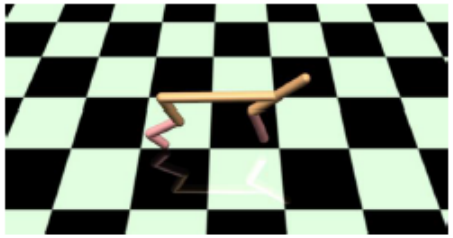
Does it Work?

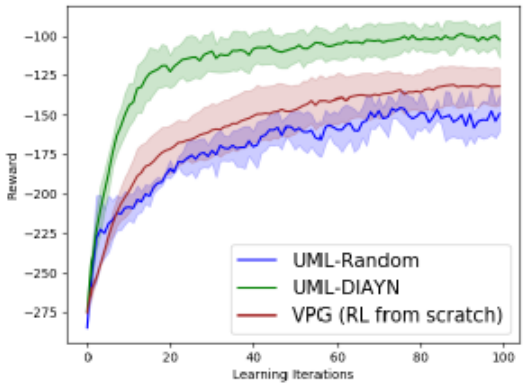
2D Navigation



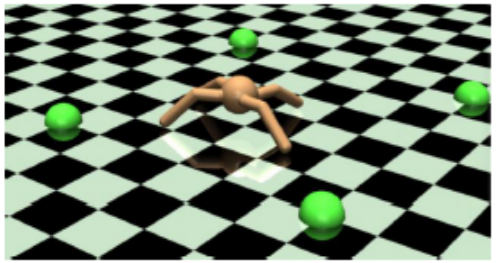


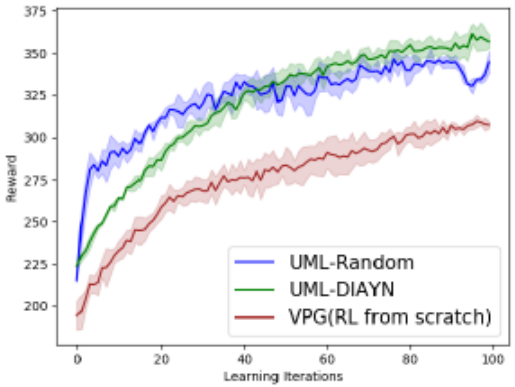
Cheetah





Ant





Meta-test performance with rewards

Gupta, Eysenbach, Finn, Levine. Unsupervised Meta-Learning for Reinforcement Learning.

Discussions

- What is the right 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) ?
- Move from static to interactive learning
  - how should this impact how we generate episodes ?
  - meta-active learning ? (few successes so far)