



Reinforcement Learning; Uncertainty Quantification & Multitask Learning



[YouTube Playlist](#)

Maziar Raissi

Assistant Professor

Department of Applied Mathematics

University of Colorado Boulder

maziar.raissi@colorado.edu

Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning


[YouTube Video](#)

$$p(y|x, W^1, W^2) = \text{softmax}(W^2 \text{ReLU}(W^1 x))$$

$$x \in \mathbb{R}^{d_0}, W^1 \in \mathbb{R}^{d_1 \times d_0}, W^2 \in \mathbb{R}^{d_2 \times d_1}$$

$$p(W^1) = \mathcal{N}(0, I), p(W^2) = \mathcal{N}(0, I)$$

$$p(y^*|x^*, \mathcal{D}) = \int p(y^*|x^*, W^1, W^2) \underbrace{p(W^1, W^2|\mathcal{D})}_{\text{not tractable}} dW^1 dW^2$$

$\mathcal{D} \rightarrow$ dataset

$$KL[q(W^1, W^2)||p(W^1, W^2|\mathcal{D})] \leq KL[q(W^1, W^2)||p(W^1)p(W^2)] - \sum_{n=1}^N \int q(W^1, W^2) \log p(y_n|x_n, W^1, W^2)$$

$$q(W^1, W^2) = q(W^1)q(W^2) \rightarrow \text{Variational Distribution}$$

$$q(W^\ell) = p^\ell \mathcal{N}(M^\ell, \sigma^2 I) + (1 - p^\ell) \mathcal{N}(0, \sigma^2 I) \rightarrow \text{Gaussian mixture distribution}$$

$$KL[q(W^\ell)||p(W^\ell)] \approx d^{\ell-1} d^\ell (\sigma^2 - \log(\sigma^2) - 1) + \frac{p^\ell}{2} \|W^\ell\|_2^2 + \text{constant}$$

$$\sigma = 10^{-33} \implies \log(\sigma) = -76$$

$$W^\ell \approx \text{diag}(z^\ell) M^\ell, M^\ell \in \mathbb{R}^{d_\ell \times d_{\ell-1}}, z^\ell \sim \text{Bernoulli}(p^\ell)$$

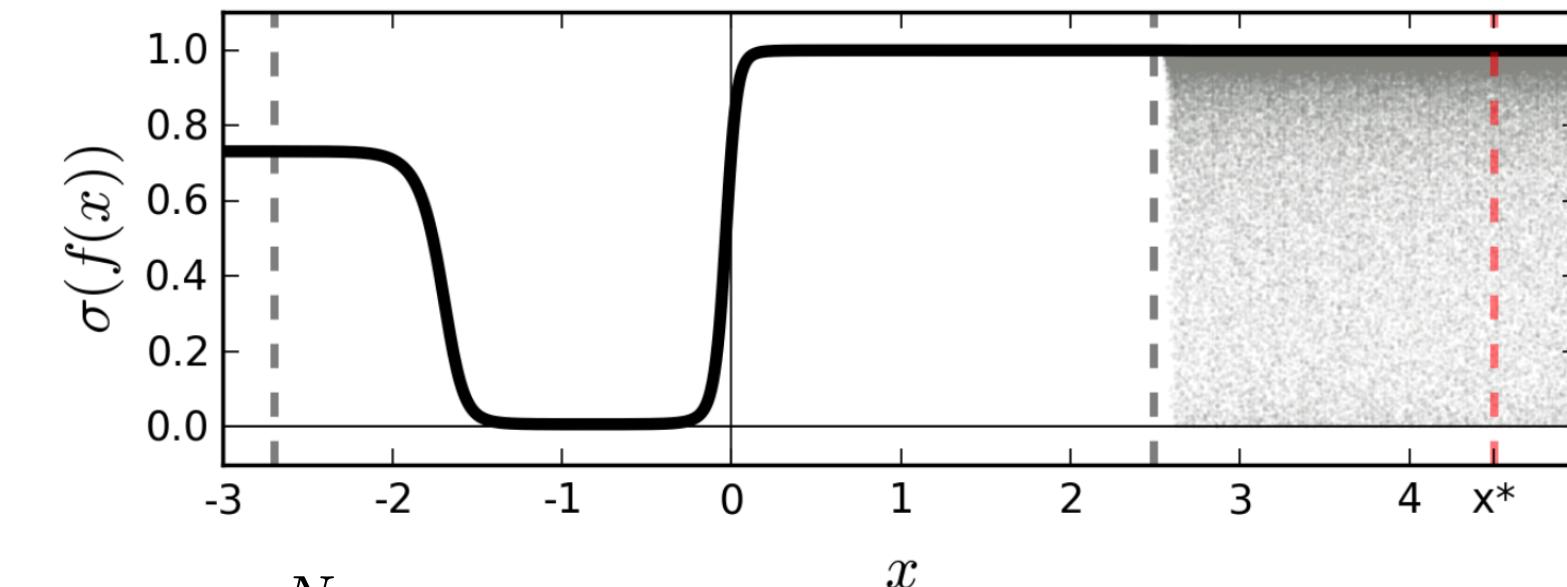
$$KL[q(W^1, W^2)||p(W^1, W^2|\mathcal{D})] \leq - \sum_{n=1}^N \log p(y_n|x_n, W_n^1, W_n^2) + \frac{p^1}{2} \|M_1\|_2^2 + \frac{p^2}{2} \|M_2\|_2^2$$

$$W_n^\ell \approx \text{diag}(z_n^\ell) M^\ell \rightarrow \text{Monte Carlo}$$

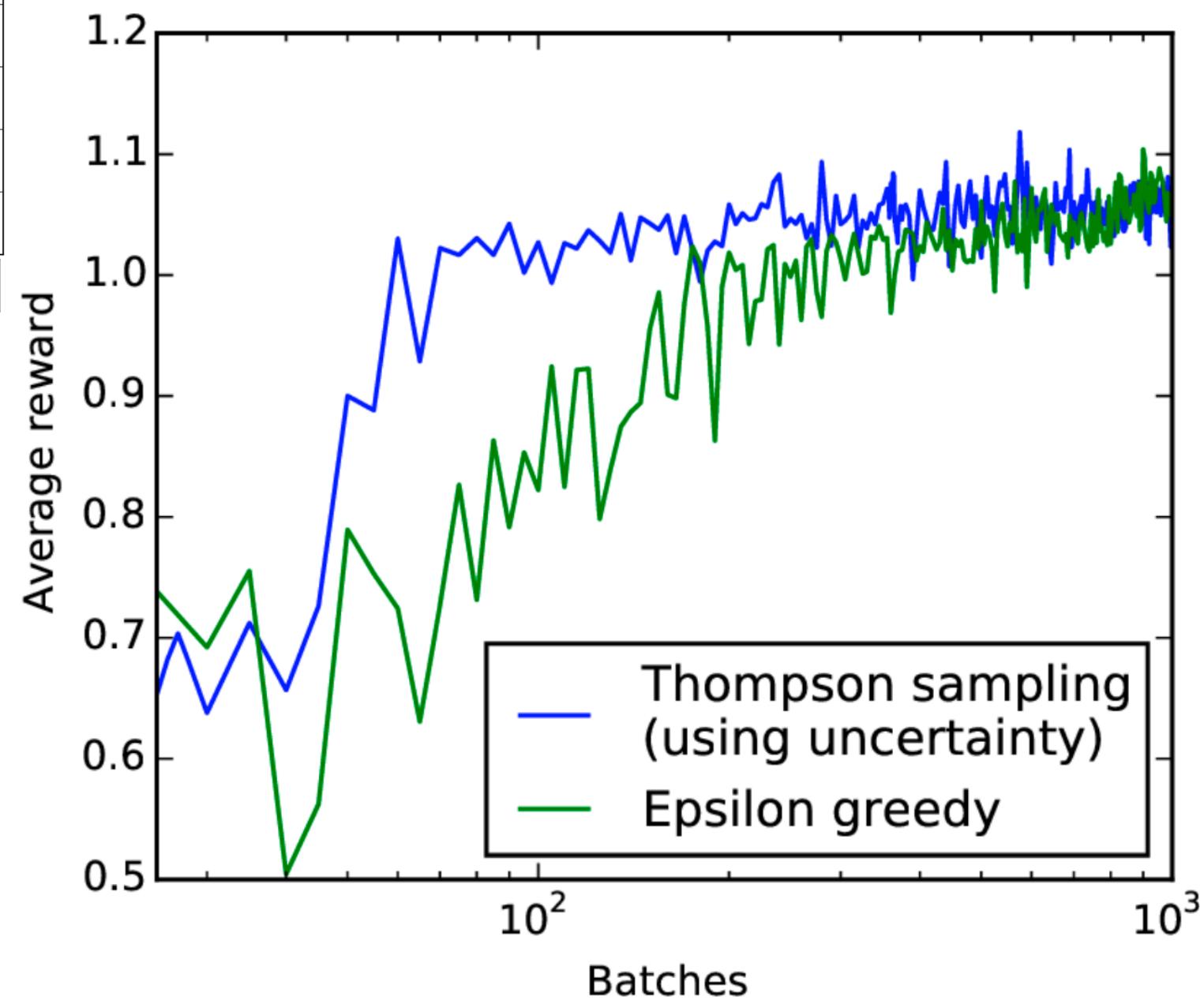
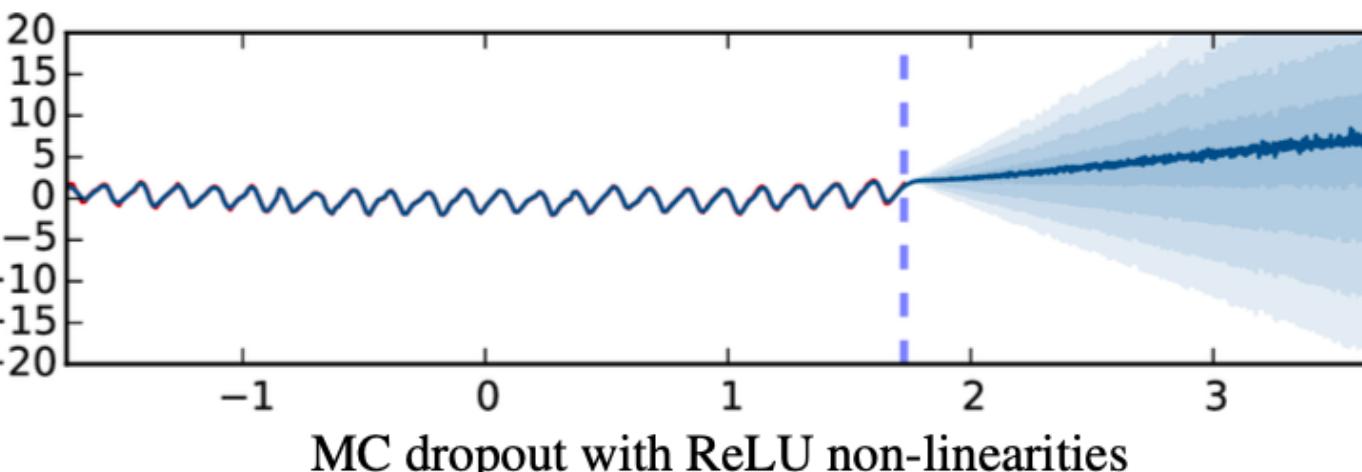
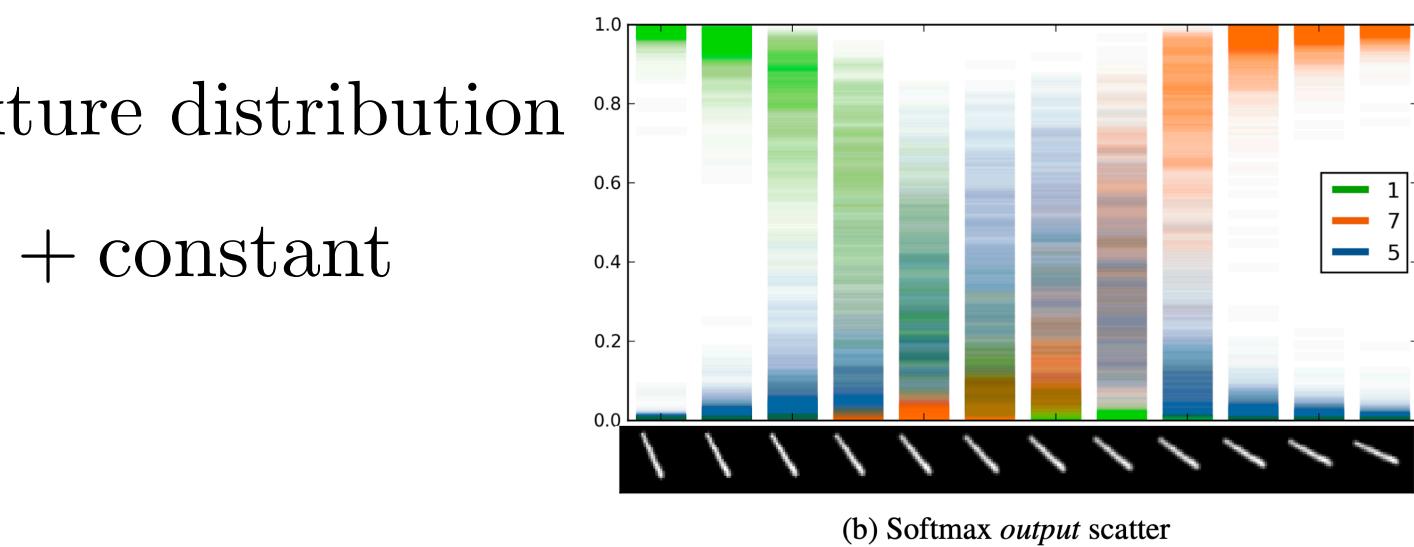
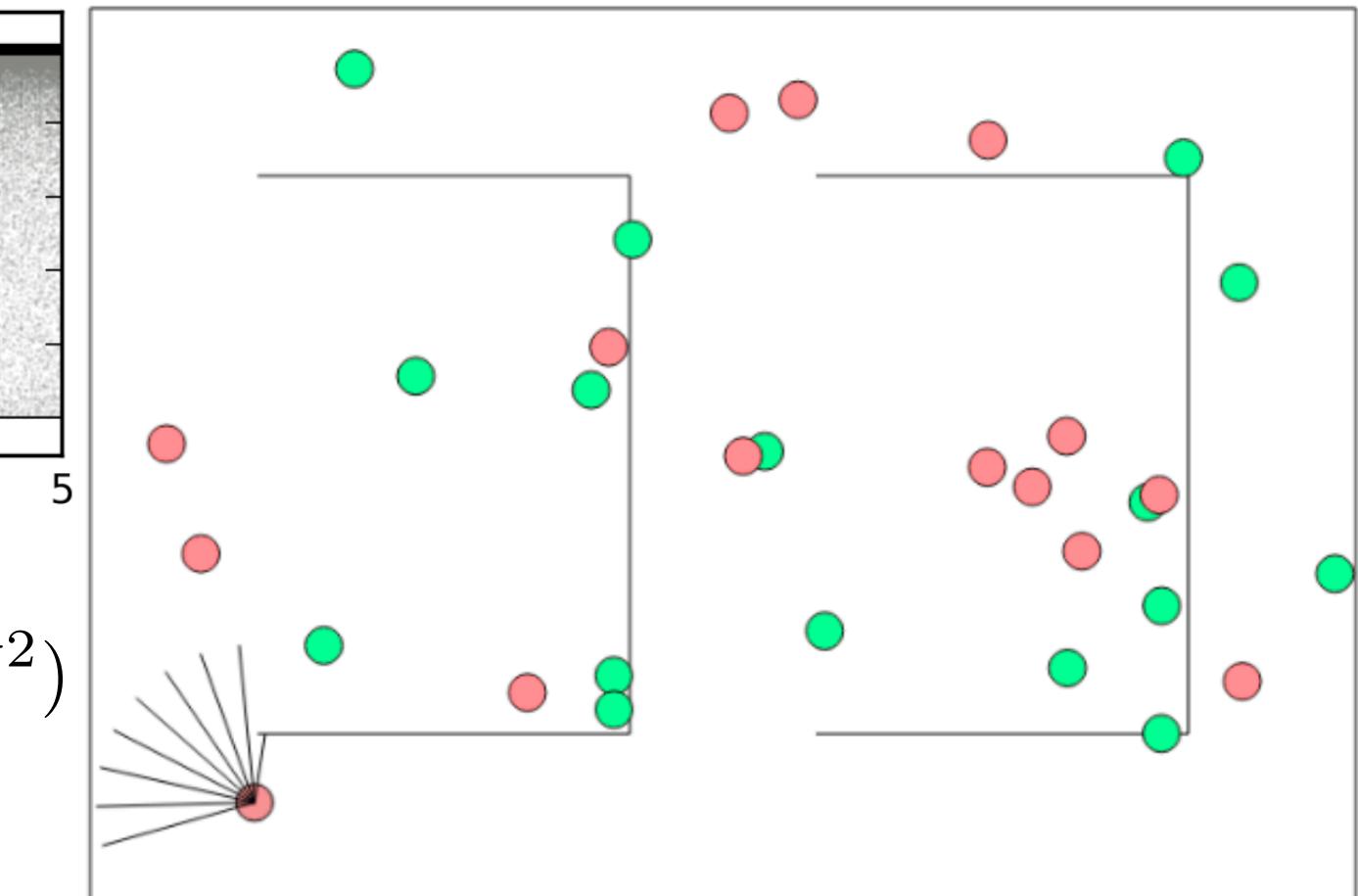
Same loss as the one used by dropout!

$$p(y^*|x^*, \mathcal{D}) \approx \int p(y^*|x^*, W^1, W^2) q(W^1) q(W^2) dW^1 dW^2$$

Use Monte Carlo to obtain the mean and variance of the predictions!



$$\sum_{n=1}^N \int q(W^1, W^2) \log p(y_n|x_n, W^1, W^2)$$





Boulder

Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks



[YouTube Playlist](#)

Goal: Train a model on a variety of learning tasks such that it can solve new learning tasks using only a small number of training examples

Tasks as training examples!

$f \rightarrow$ model

$x \mapsto a$ outputs
observations

$\mathcal{T} = \{ \mathcal{L}(x_1, a_1, \dots, x_H, a_H), q(x_1), q(x_{t+1}|x_t, a_t), H \}$

episode length
transition distribution
distribution over initial observations

$H = 1$ in *i.i.d* supervised problems

$p(\mathcal{T}) \rightarrow$ distribution over tasks

$$\min_{\theta} \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta'_i}) = \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta - \alpha \nabla_{\theta} \mathcal{L}_{\mathcal{T}_i}(f_{\theta})})$$

Optimize the model parameters such that one or a small number of gradient steps on a new task will produce maximally effective behavior on that task

$$\mathcal{L}_{\mathcal{T}_i}(f_{\phi}) = \sum_{\mathbf{x}^{(j)}, \mathbf{y}^{(j)} \sim \mathcal{T}_i} \|f_{\phi}(\mathbf{x}^{(j)}) - \mathbf{y}^{(j)}\|_2^2, \rightarrow \text{Regression}$$

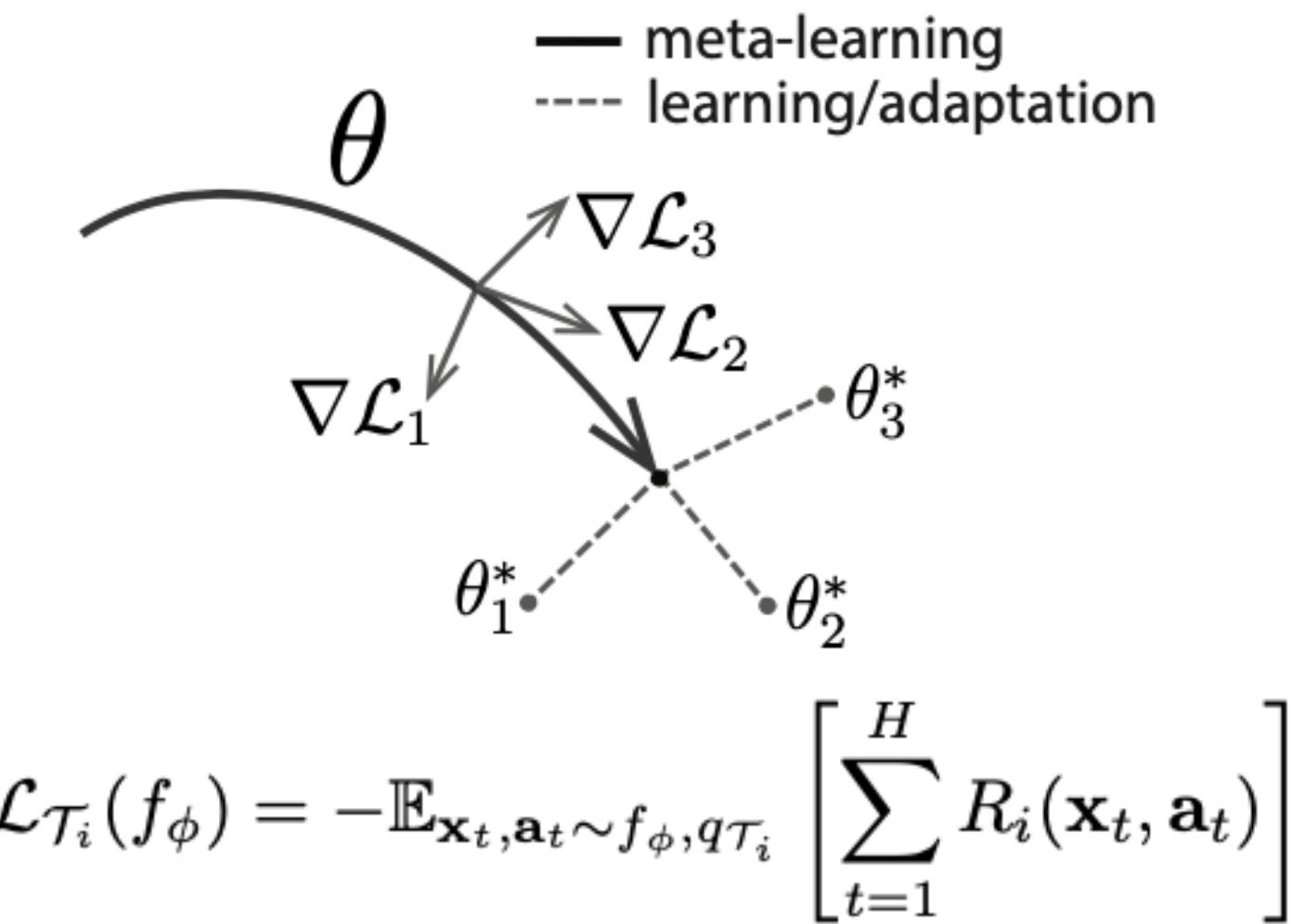
$$\begin{aligned} \mathcal{L}_{\mathcal{T}_i}(f_{\phi}) &= \sum_{\mathbf{x}^{(j)}, \mathbf{y}^{(j)} \sim \mathcal{T}_i} \mathbf{y}^{(j)} \log f_{\phi}(\mathbf{x}^{(j)}) \\ &\quad + (1 - \mathbf{y}^{(j)}) \log(1 - f_{\phi}(\mathbf{x}^{(j)})) \end{aligned} \rightarrow \text{Classification}$$

Algorithm 1 Model-Agnostic Meta-Learning

Require: $p(\mathcal{T})$: distribution over tasks

Require: α, β : step size hyperparameters

- 1: randomly initialize θ
 - 2: **while** not done **do**
 - 3: Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
 - 4: **for all** \mathcal{T}_i **do**
 - 5: Evaluate $\nabla_{\theta} \mathcal{L}_{\mathcal{T}_i}(f_{\theta})$ with respect to K examples
 - 6: Compute adapted parameters with gradient descent: $\theta'_i = \theta - \alpha \nabla_{\theta} \mathcal{L}_{\mathcal{T}_i}(f_{\theta})$
 - 7: **end for**
 - 8: Update $\theta \leftarrow \theta - \beta \nabla_{\theta} \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta'_i})$
 - 9: **end while**
-



Algorithm 2 MAML for Few-Shot Supervised Learning

Require: $p(\mathcal{T})$: distribution over tasks

Require: α, β : step size hyperparameters

- 1: randomly initialize θ
 - 2: **while** not done **do**
 - 3: Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
 - 4: **for all** \mathcal{T}_i **do**
 - 5: Sample K datapoints $\mathcal{D} = \{\mathbf{x}^{(j)}, \mathbf{y}^{(j)}\}$ from \mathcal{T}_i
 - 6: Evaluate $\nabla_{\theta} \mathcal{L}_{\mathcal{T}_i}(f_{\theta})$ using \mathcal{D} and $\mathcal{L}_{\mathcal{T}_i}$ in Equation (2) or (3)
 - 7: Compute adapted parameters with gradient descent: $\theta'_i = \theta - \alpha \nabla_{\theta} \mathcal{L}_{\mathcal{T}_i}(f_{\theta})$
 - 8: Sample datapoints $\mathcal{D}'_i = \{\mathbf{x}^{(j)}, \mathbf{y}^{(j)}\}$ from \mathcal{T}_i for the meta-update
 - 9: **end for**
 - 10: Update $\theta \leftarrow \theta - \beta \nabla_{\theta} \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta'_i})$ using each \mathcal{D}'_i and $\mathcal{L}_{\mathcal{T}_i}$ in Equation 2 or 3
 - 11: **end while**
-

Algorithm 3 MAML for Reinforcement Learning

Require: $p(\mathcal{T})$: distribution over tasks

Require: α, β : step size hyperparameters

- 1: randomly initialize θ
 - 2: **while** not done **do**
 - 3: Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
 - 4: **for all** \mathcal{T}_i **do**
 - 5: Sample K trajectories $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{a}_1, \dots, \mathbf{x}_H)\}$ using f_{θ} in \mathcal{T}_i
 - 6: Evaluate $\nabla_{\theta} \mathcal{L}_{\mathcal{T}_i}(f_{\theta})$ using \mathcal{D} and $\mathcal{L}_{\mathcal{T}_i}$ in Equation 4
 - 7: Compute adapted parameters with gradient descent: $\theta'_i = \theta - \alpha \nabla_{\theta} \mathcal{L}_{\mathcal{T}_i}(f_{\theta})$
 - 8: Sample trajectories $\mathcal{D}'_i = \{(\mathbf{x}_1, \mathbf{a}_1, \dots, \mathbf{x}_H)\}$ using $f_{\theta'_i}$ in \mathcal{T}_i
 - 9: **end for**
 - 10: Update $\theta \leftarrow \theta - \beta \nabla_{\theta} \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta'_i})$ using each \mathcal{D}'_i and $\mathcal{L}_{\mathcal{T}_i}$ in Equation 4
 - 11: **end while**
-



Boulder

Overcoming catastrophic forgetting in neural networks

Continual learning: Learning multiple tasks sequentially

$\theta \rightarrow$ weight and biases of a neural network

$\theta_B^* \rightarrow$ solution to task B

$\theta_A^* \rightarrow$ solution to task A

Over-parametrization makes it likely that θ_B^* is close to θ_A^*

Elastic Weight Consolidation (EWC)

$$\mathcal{L}(\theta) = \mathcal{L}_B(\theta) + \frac{1}{2}\lambda \sum_i F_i (\theta_i - \theta_{A,i}^*)^2 \rightarrow \text{loss function}$$

Constraining “important” parameters to stay close to their old values!

$p(\theta|\mathcal{D}) \rightarrow$ posterior

$$\log p(\theta|\mathcal{D}) = \underbrace{\log p(\mathcal{D}|\theta)}_{-\mathcal{L}(\theta)} + \log p(\theta) - \log p(\mathcal{D})$$

$$\mathcal{D} = \mathcal{D}_A \cup \mathcal{D}_B$$

$\mathcal{D}_A \rightarrow$ data on task A

$\mathcal{D}_B \rightarrow$ data on task B

$$\log p(\theta|\mathcal{D}) = \underbrace{\log p(\mathcal{D}_B|\theta)}_{-\mathcal{L}_B(\theta)} + \underbrace{\log p(\theta|\mathcal{D}_A)}_{\text{interactable}} - \log p(\mathcal{D}_B)$$

Laplace Approximation

$$f(\theta) := \log p(\theta|\mathcal{D}_A) = \underbrace{\log p(\mathcal{D}_A|\theta)}_{-\mathcal{L}_A(\theta)} + \log p(\theta) - \log p(\mathcal{D}_A)$$

$$f(\theta) \approx f(\theta_A^*) + \underbrace{\nabla_\theta f(\theta_A^*)}_{\text{const.}} (\theta - \theta_A^*) + \frac{1}{2} (\theta - \theta_A^*)^T \underbrace{\nabla_\theta^2 f(\theta_A^*)}_{\text{Hessian}} (\theta - \theta_A^*)$$

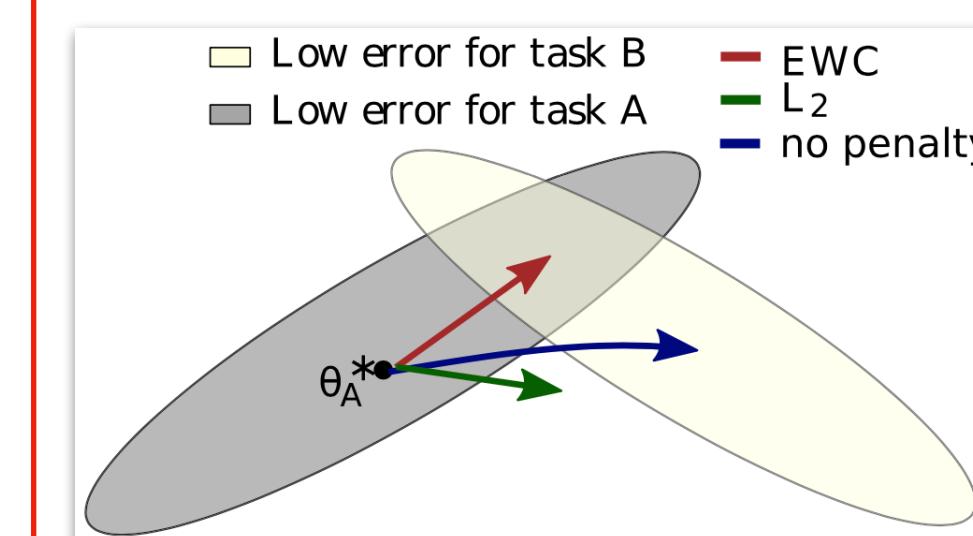
$$\nabla_\theta^2 f(\theta_A^*) = - \underbrace{\nabla_\theta^2 \mathcal{L}_A(\theta_A^*)}_{\lambda F} + \underbrace{\nabla_\theta^2 \log p(\theta)}_{\text{ignore}}$$

$F \rightarrow$ empirical Fisher information matrix

$$p(\theta|\mathcal{D}_A) \approx \mathcal{N}(\theta_A^*, \text{diag}(\lambda F)^{-1})$$

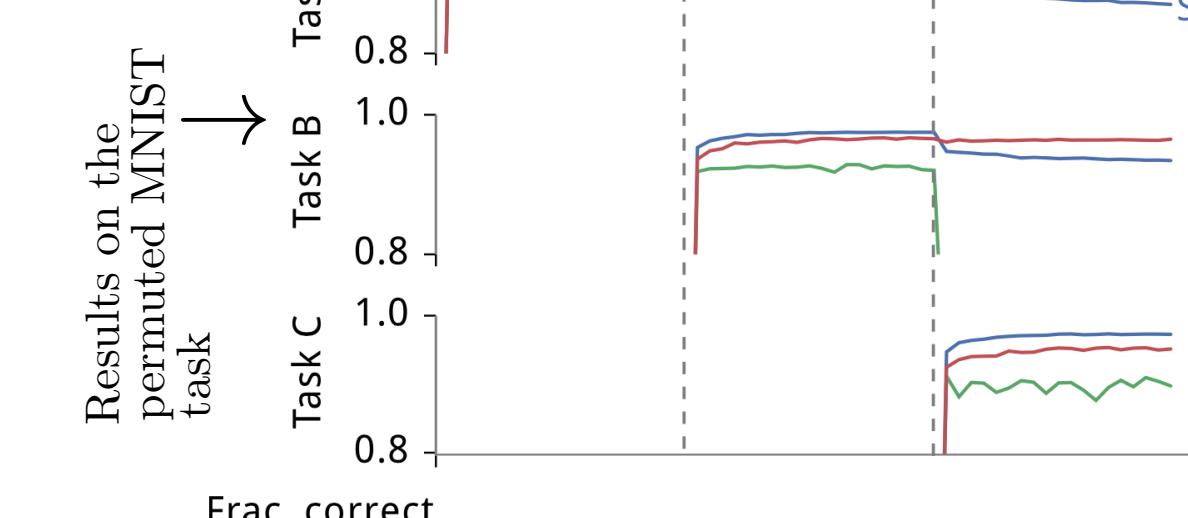
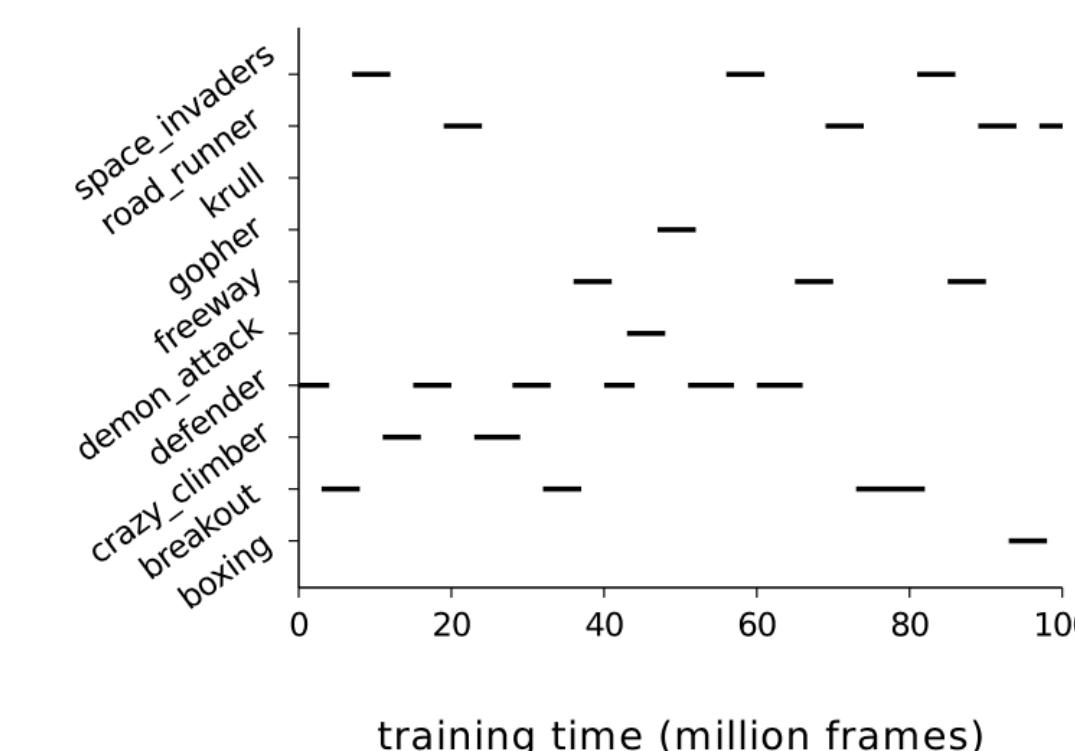
Three key properties of F :

- It is equivalent to the second derivative of the loss near a minimum.
- It can be computed from first-order derivatives alone and is thus easy to calculate even for large models.
- It is guaranteed to be positive semidefinite.

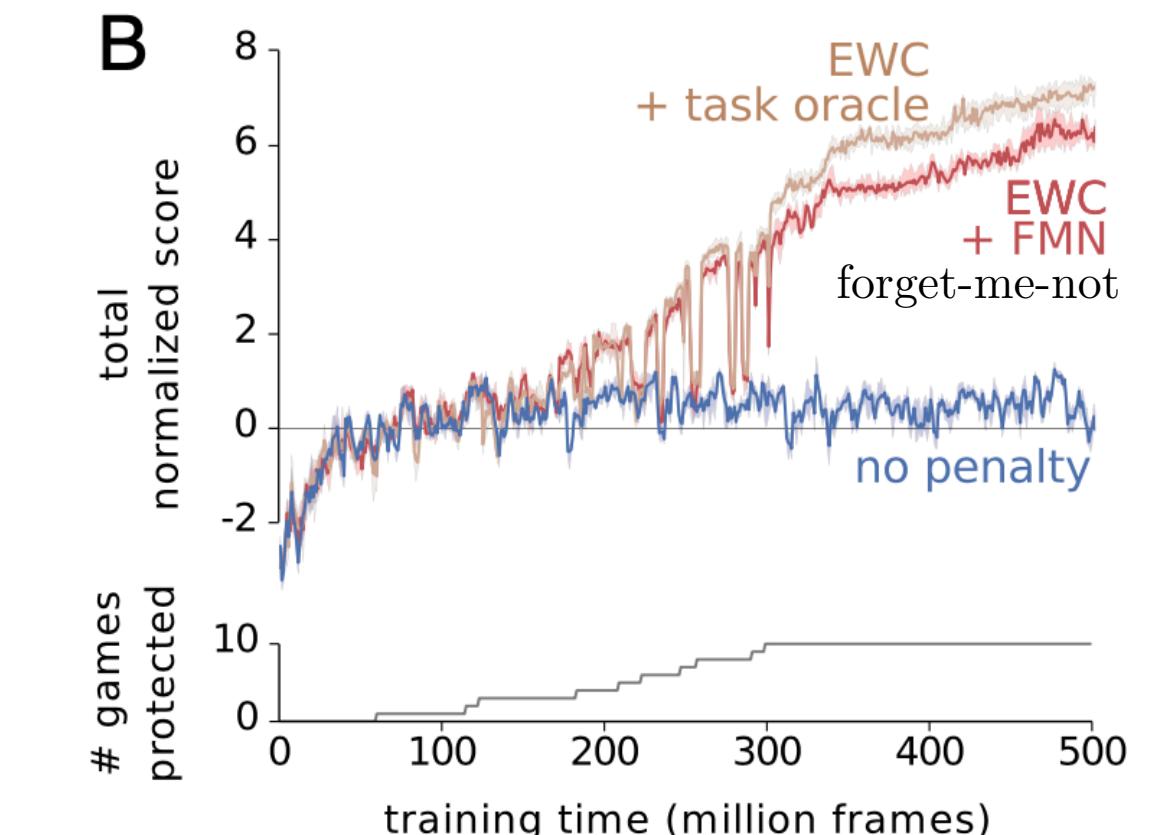


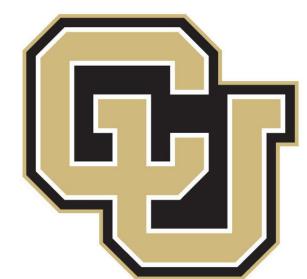
A

Results on Atari task



B





Boulder



Questions?

[YouTube Playlist](#)
