DeepMind

Deep Learning with Low Resources

An algorithmic approach

Jonathan Schwarz
DeepMind
Gatsby Unit, UCL

schwarzjn@google.com / @schwarzjn_

Will make slides available after the talk

UCL

Outline

- **Small data: Meta-Learning**
    - **Overview**
    - Optimisation-based
    - Black-box
- Faster training and inference: Sparsity
    - Overview
    - Explicit Sparsity
    - Implicit Sparsity

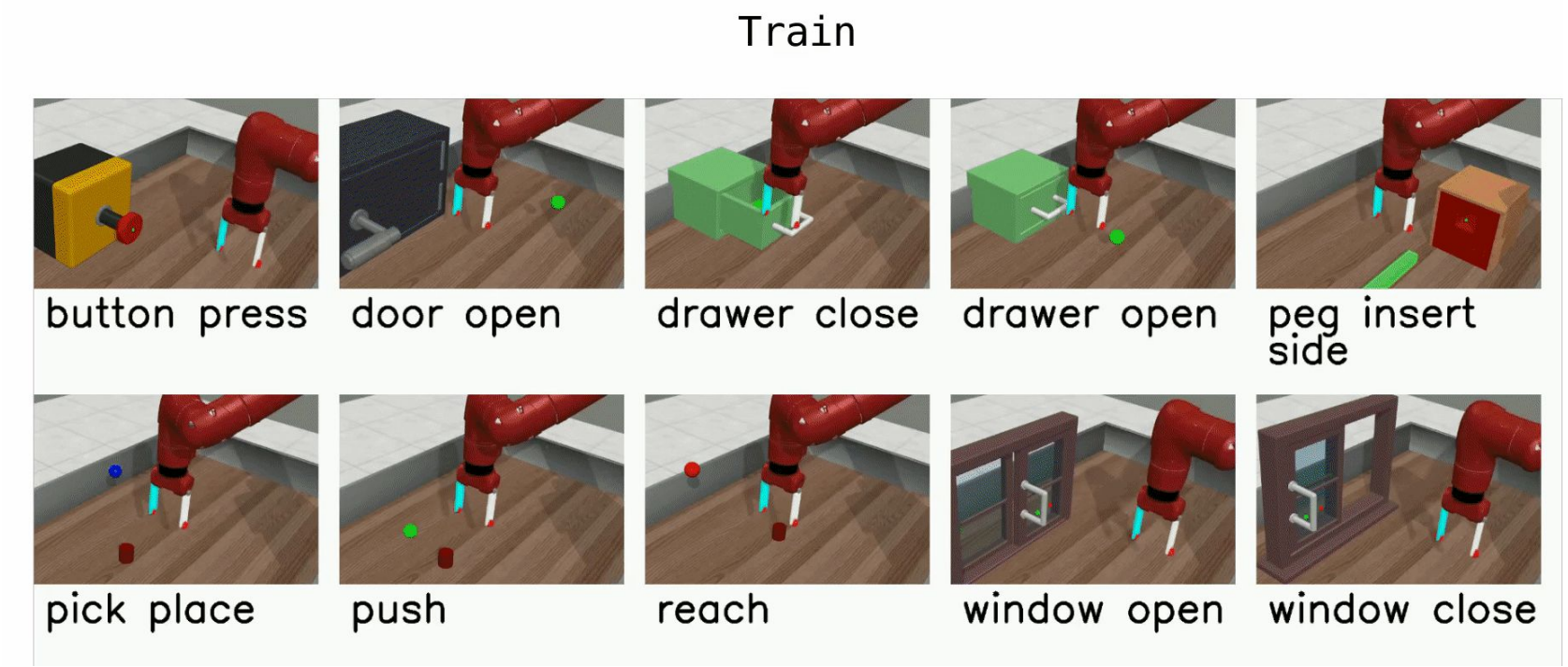
- Discussion & Looking forward

Small data problems - Few shot learning

[Andrei Rusu, ImageNet]

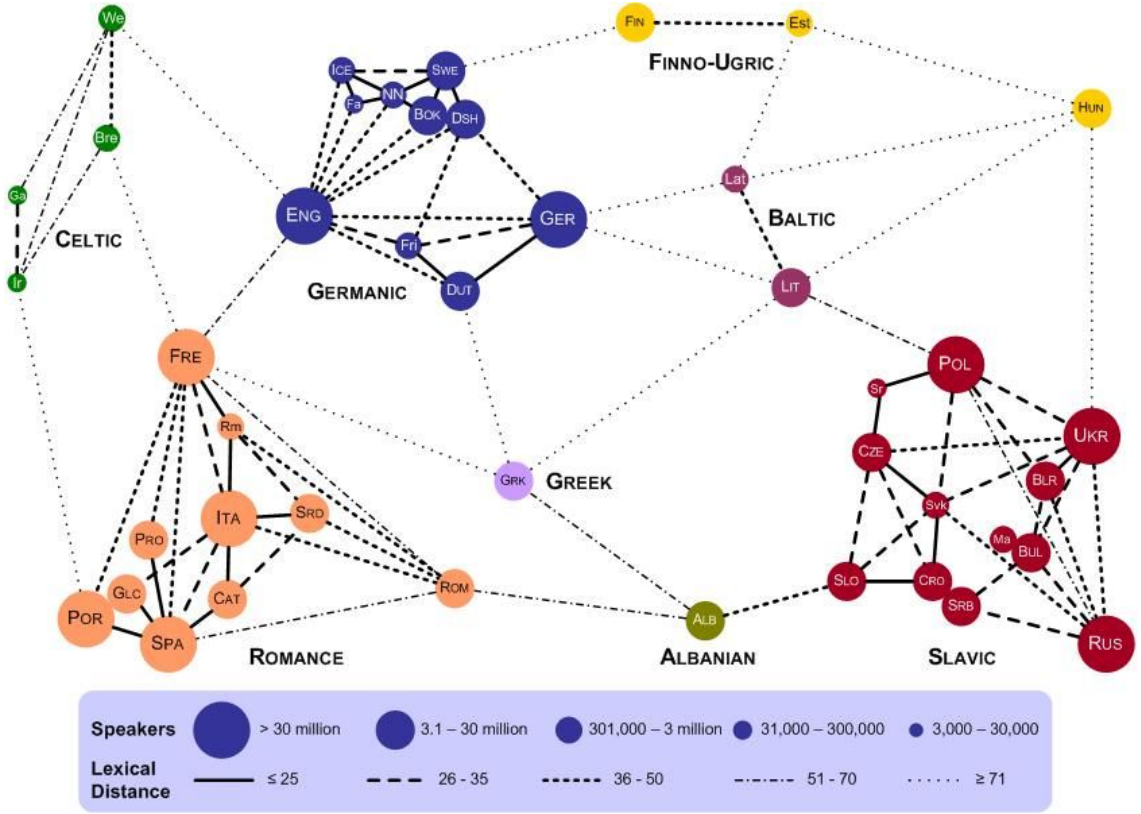
Small data problems - Recommender Systems

Small data problems - Robotics (e.g. demonstrations)

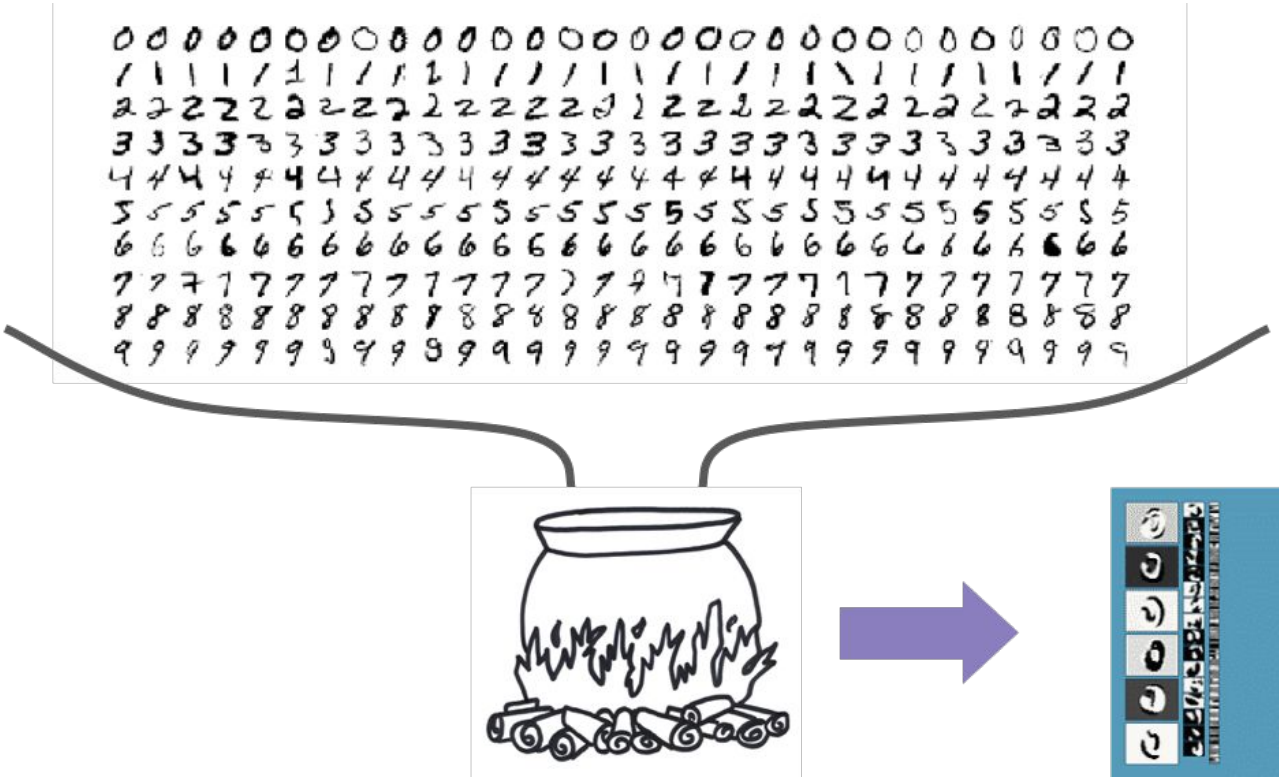


[MetaWorld]

Small data problems - Language

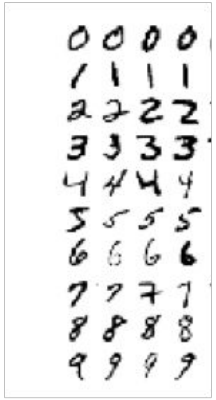


Machine Learning with big data



[Yann LeCun, MNIST]

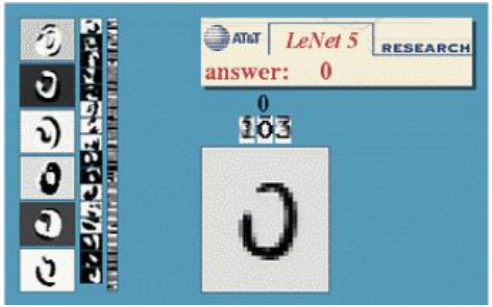
Machine Learning with small data



Inductive biases:

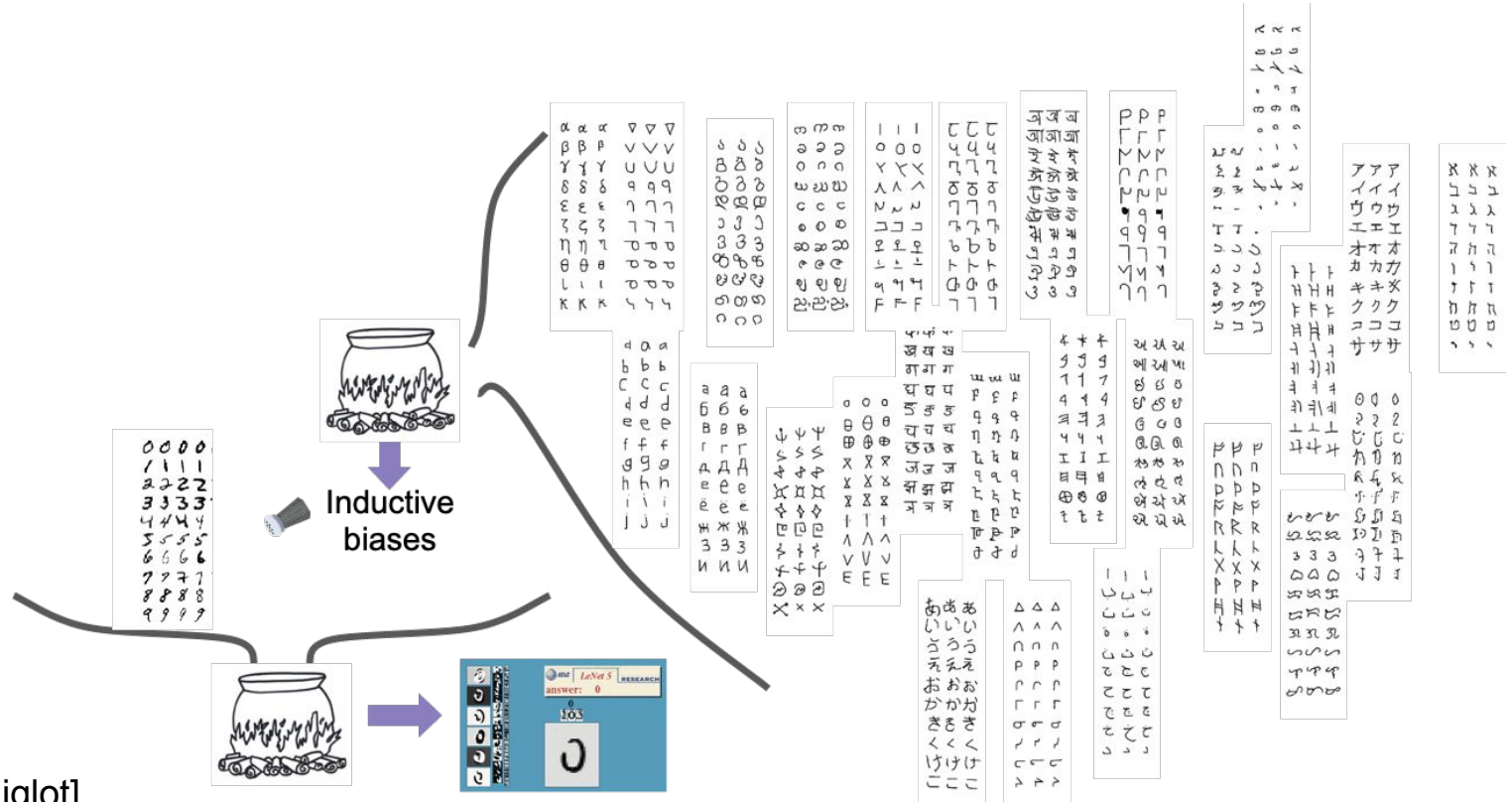
- Preprocessing/Feature engineering
- Data augmentation
- Regularisation
- Model architectures

→ **Time intensive!**



[Yann LeCun, MNIST]

Learning to learn / Meta-Learning

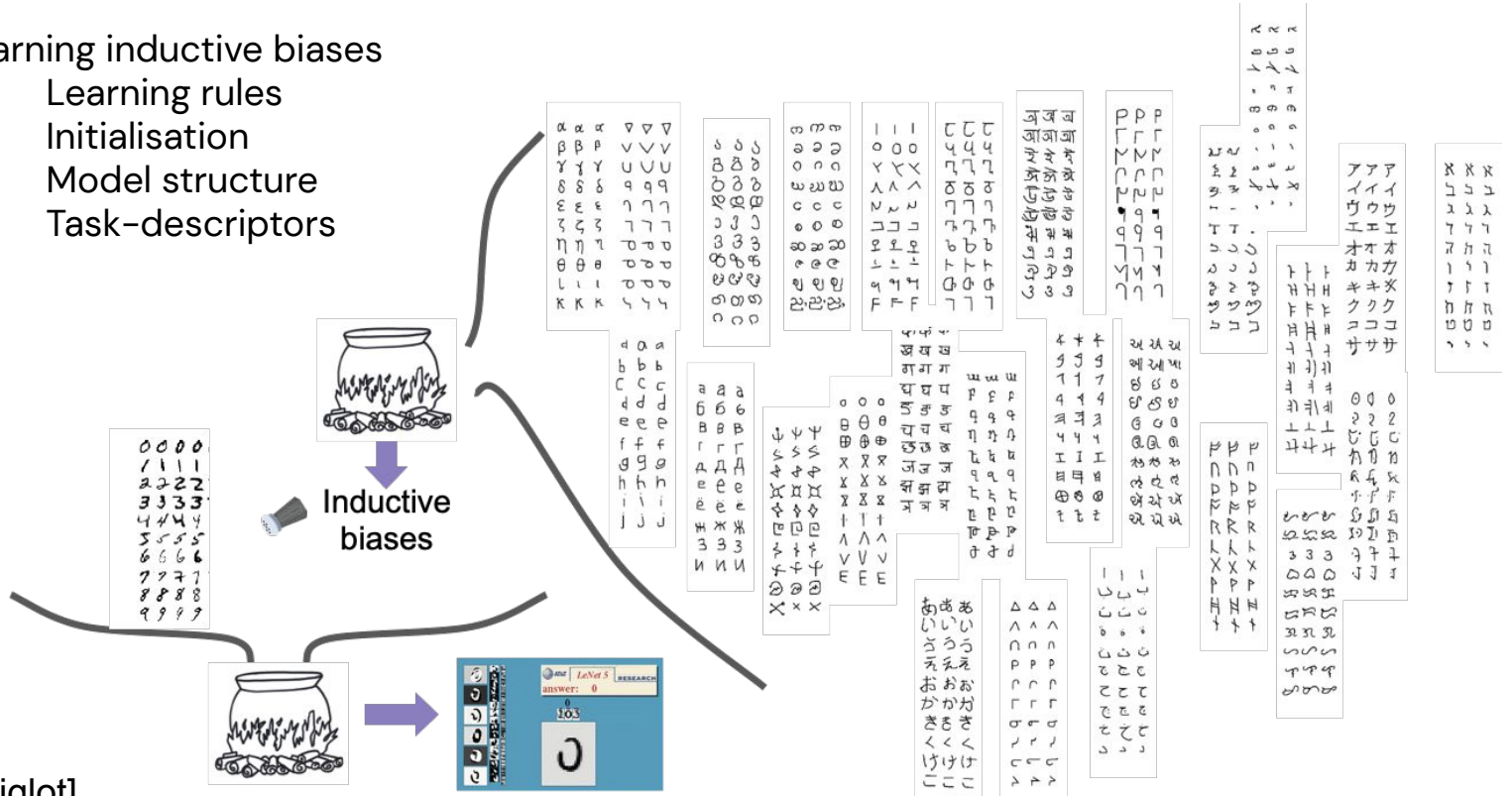


[Omniglot]

Learning to learn / Meta-Learning

Learning inductive biases

- Learning rules
- Initialisation
- Model structure
- Task-descriptors



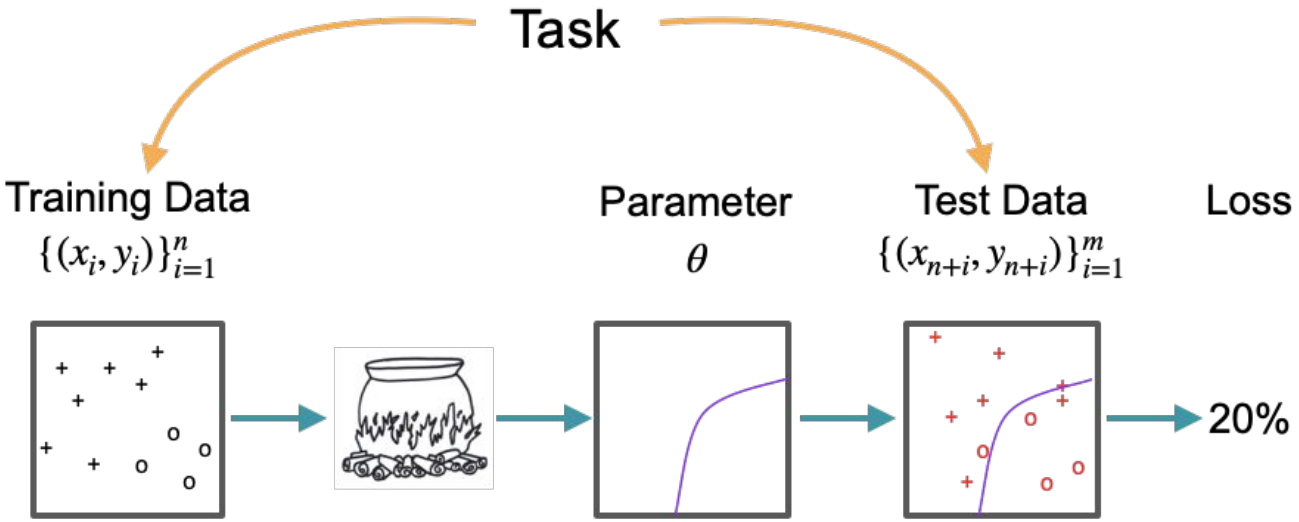
[Omniglot]

Outline

- **Small data: Meta-Learning**
    - **Overview**
    - Optimisation-based
    - Black-box
- Faster training and inference: Sparsity
    - Overview
    - Explicit Sparsity
    - Implicit Sparsity

- Discussion & Looking forward

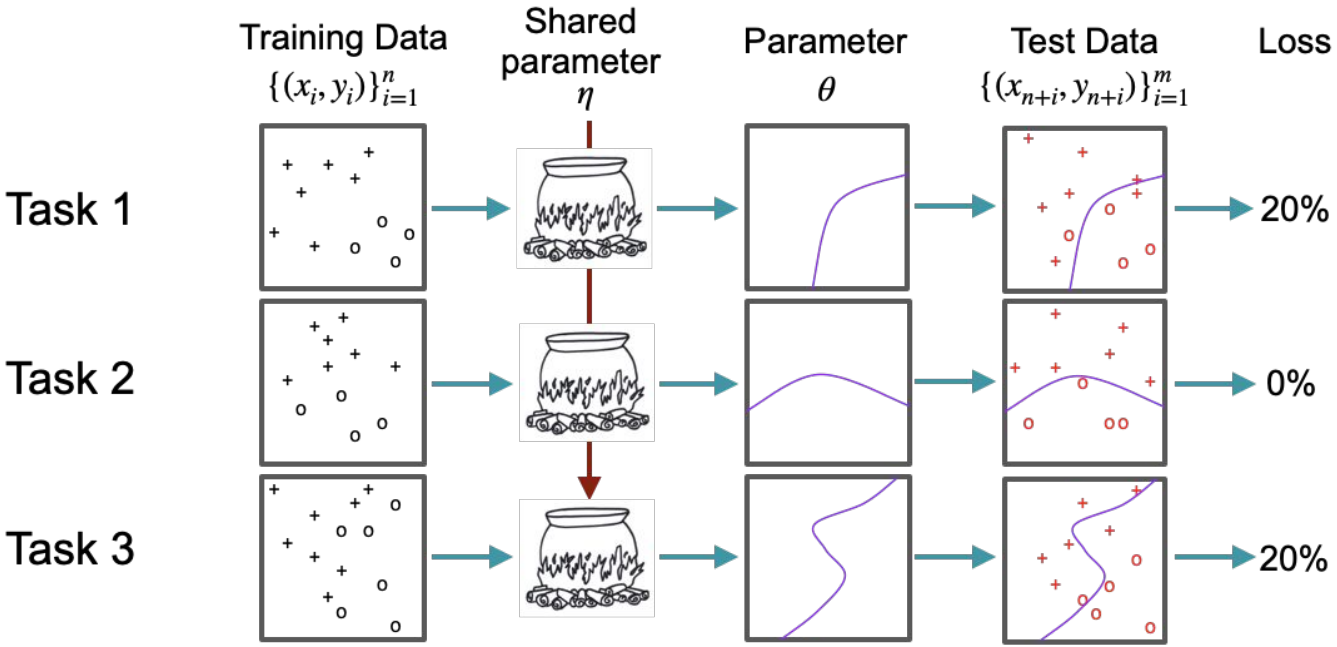
Single Task Learning



Empirical Risk Minimisation

$$\arg\min_{\theta} \sum_{i=1}^{n} \text{Loss}(f_{\eta,\theta}(x_i), y_i) + \text{Regulariser}(\theta)$$

Multi Task Learning



$$\arg\min_{\eta,\{\theta_j\}} \sum_{j=1}^{T} \sum_{i=1}^{n} \text{Loss}(f_{\eta,\theta_j}(x_{ji}), y_{ji}) + \text{Regulariser}(\theta_j)$$

Meta-Learning

- Is it possible to learn to generalise well from small data?
  - Learn:

$$\theta_j = \arg\min_{\theta_j} \sum_{i=1}^{n} \text{Loss}(f_{\eta,\theta_j}(x_{ji}), y_{ji}) =: \text{Learner}(\eta, \text{TrainData}_j)$$

  - Test performance

$$\sum_{i=n+1}^{n+m} \text{Loss}(f_{\eta,\theta_j}(x_{ji}), y_{ji}) =: \mathscr{L}(\eta, \theta_j, \text{TestData}_j)$$

Meta-Learning

- Is it possible to learn to generalise well from small data?
  - Learn:

$$\theta_j = \arg\min_{\theta_j} \sum_{i=1}^{n} \text{Loss}(f_{\eta,\theta_j}(x_{ji}), y_{ji}) =: \text{Learner}(\eta, \text{TrainData}_j)$$

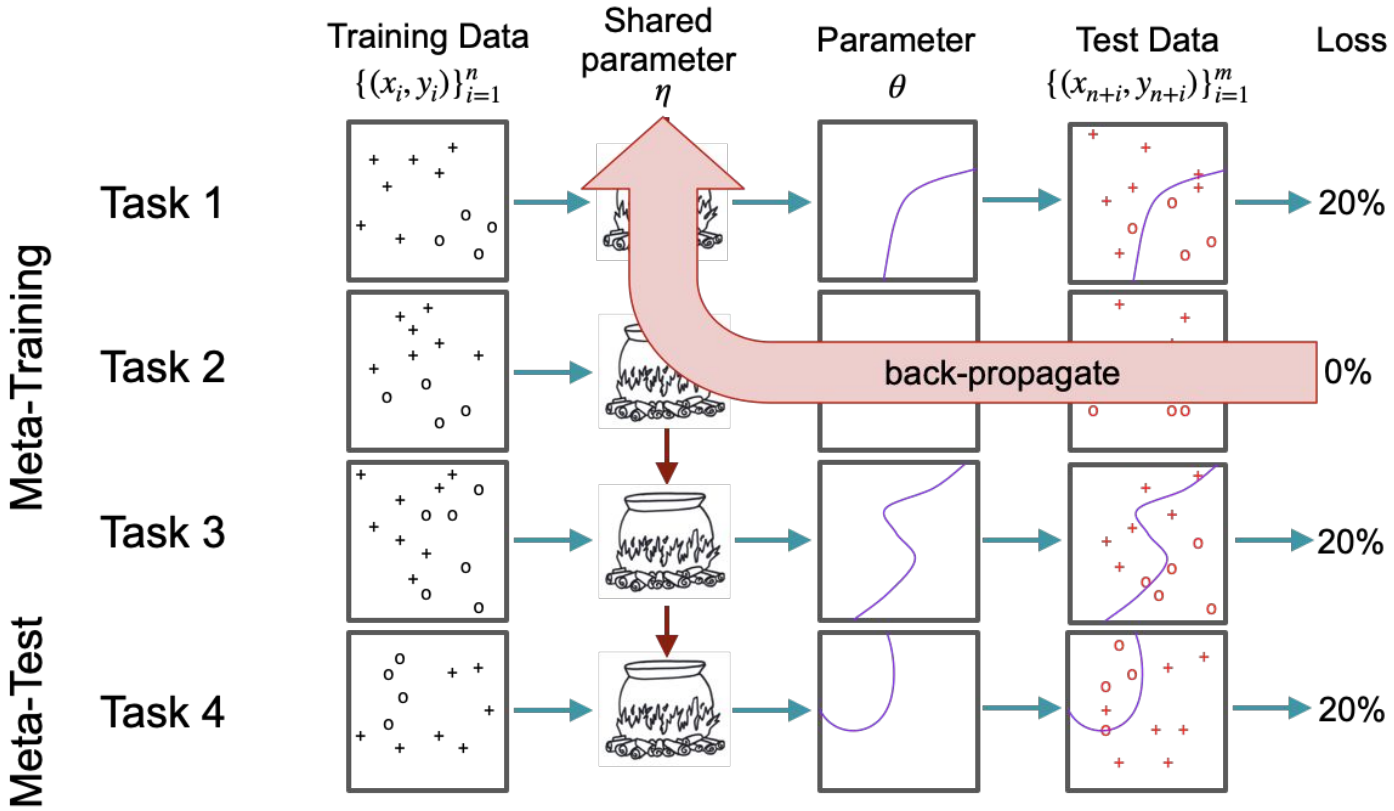
  - Test performance

$$\sum_{i=n+1}^{n+m} \text{Loss}(f_{\eta,\theta_j}(x_{ji}), y_{ji}) =: \mathscr{L}(\eta, \theta_j, \text{TestData}_j)$$

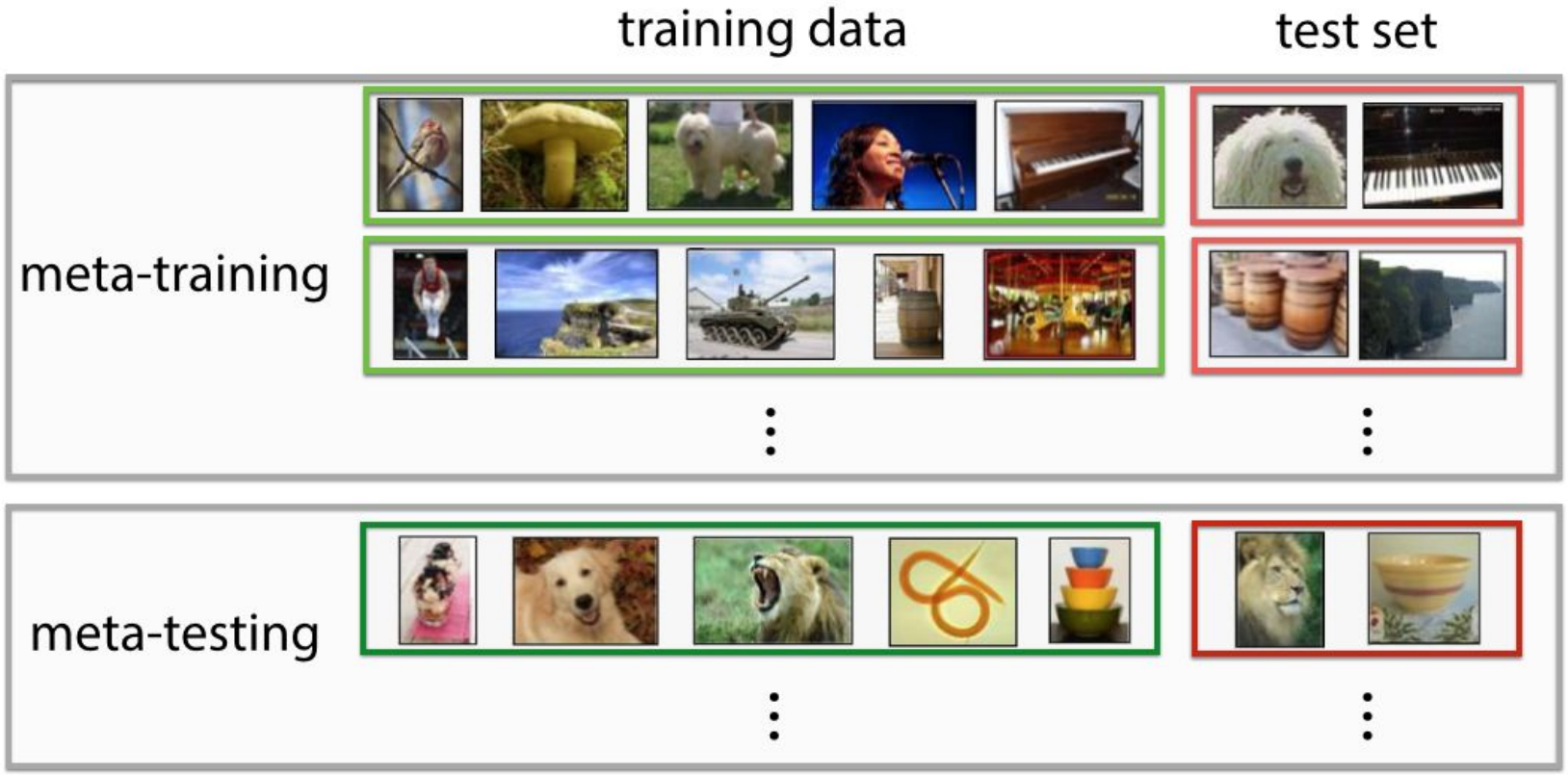
  - Optimise shared parameters for test performance

$$\arg\min_{\eta} \sum_{j=1}^{T} \mathscr{L}(\eta, \theta_j, \text{TestData}_j) = \arg\min_{\eta} \sum_{j=1}^{T} \sum_{i=n+1}^{n+m} \text{Loss}(f_{\eta,\text{Learner}(\eta,\text{TrainData}_j)}(x_{ji}), y_{ji})$$

Meta Learning



Training/Test?



[Chelsea Finn]

Meta-Learning Pseudocode

- For each iteration:
  - Pick a (minibatch of) training tasks (here j)
  - Base Learner

$$\theta_j = \text{Learner}(\eta, \text{TrainData}_j) = \arg\min_{\theta_j} \sum_{i=1}^{n} L(f_{\eta,\theta_j}(x_{ji}), y_{ji})$$

  - Test performance

$$\mathscr{L}(\eta, \theta_j, \text{TestData}_j) = \sum_{i=n+1}^{n+m} \text{Loss}(f_{\eta,\theta_j}(x_{ji}), y_{ji})$$

Meta-Learning Pseudocode

- For each iteration:
  - Pick a (minibatch of) training tasks (here j)
  - Base Learner

$$\theta_j = \text{Learner}(\eta, \text{TrainData}_j) = \arg\min_{\theta_j} \sum_{i=1}^{n} L(f_{\eta,\theta_j}(x_{ji}), y_{ji})$$

  - Test performance

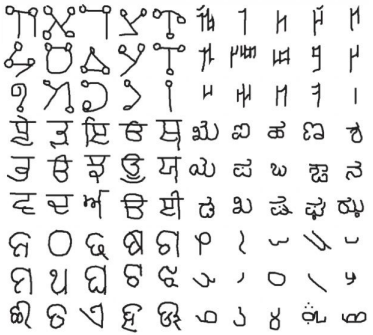
$$\mathscr{L}(\eta, \theta_j, \text{TestData}_j) = \sum_{i=n+1}^{n+m} \text{Loss}(f_{\eta,\theta_j}(x_{ji}), y_{ji})$$

  - Meta-Learner: Optimise shared parameters for test performance

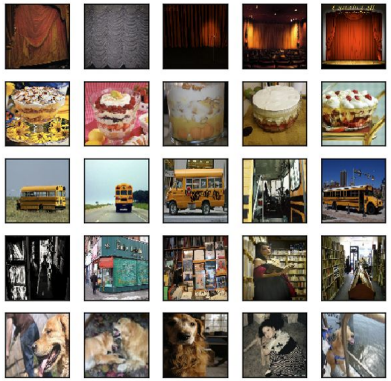
$$\eta \leftarrow \eta - \epsilon \frac{d}{d\eta} \sum_{i=n+1}^{n+m} \text{Loss}(f_{\eta,\text{Learner}(\eta,\text{TrainData}_j)}(x_{ji}), y_{ji})$$

Meta-Learning Datasets

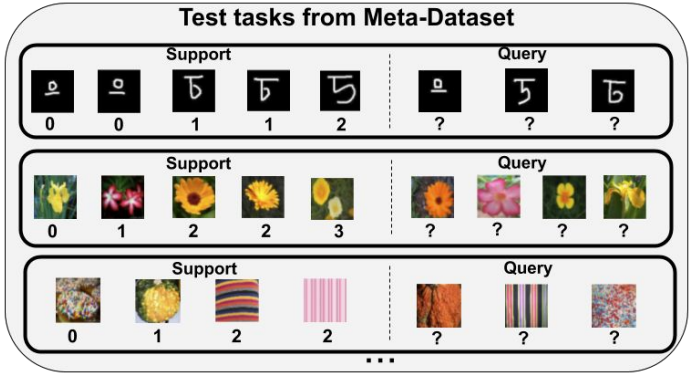
Omniglot



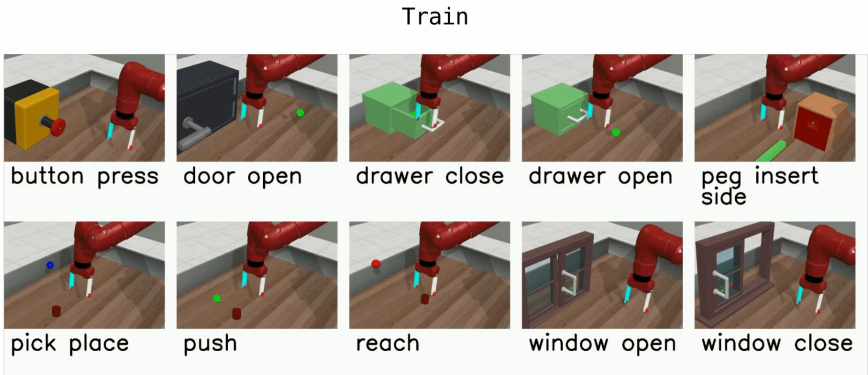
mini-Imagenet



Meta-Dataset

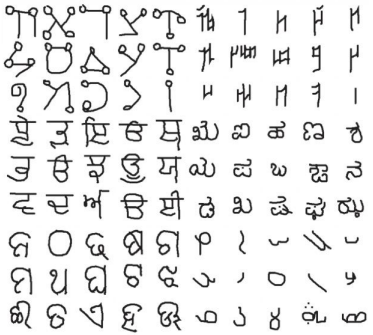


MetaWorld

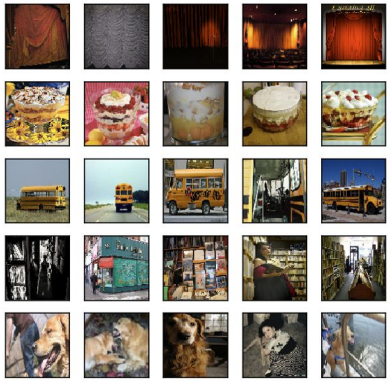


Meta-Learning Datasets

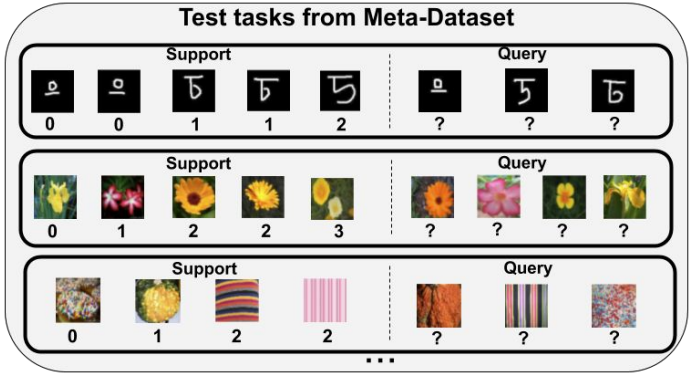
Omniglot



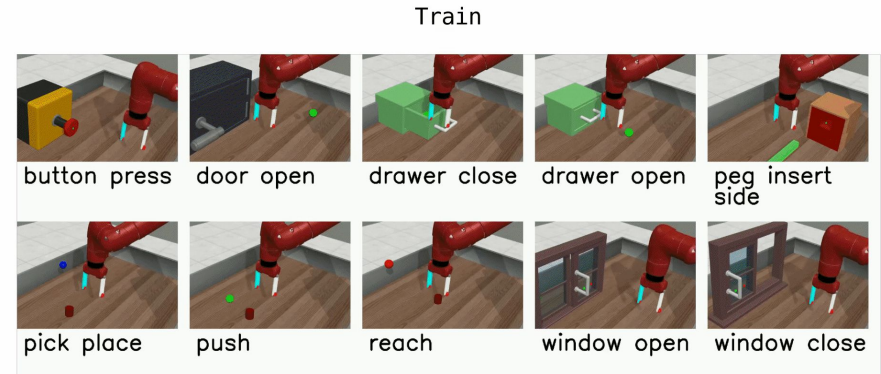
mini-Imagenet



Meta-Dataset



MetaWorld



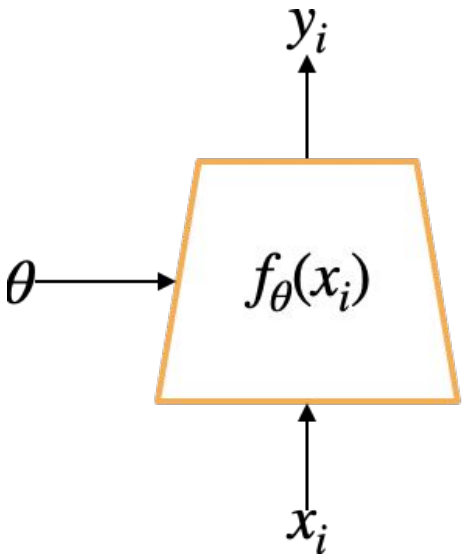
Design of such datasets still an empirical question

Outline

- **Small data: Meta-Learning**
    - Overview
    - **Optimisation-based**
    - Black-box
- Faster training and inference: Sparsity
    - Overview
    - Explicit Sparsity
    - Implicit Sparsity

- Discussion & Looking forward

Optimisation-based Meta-Learning

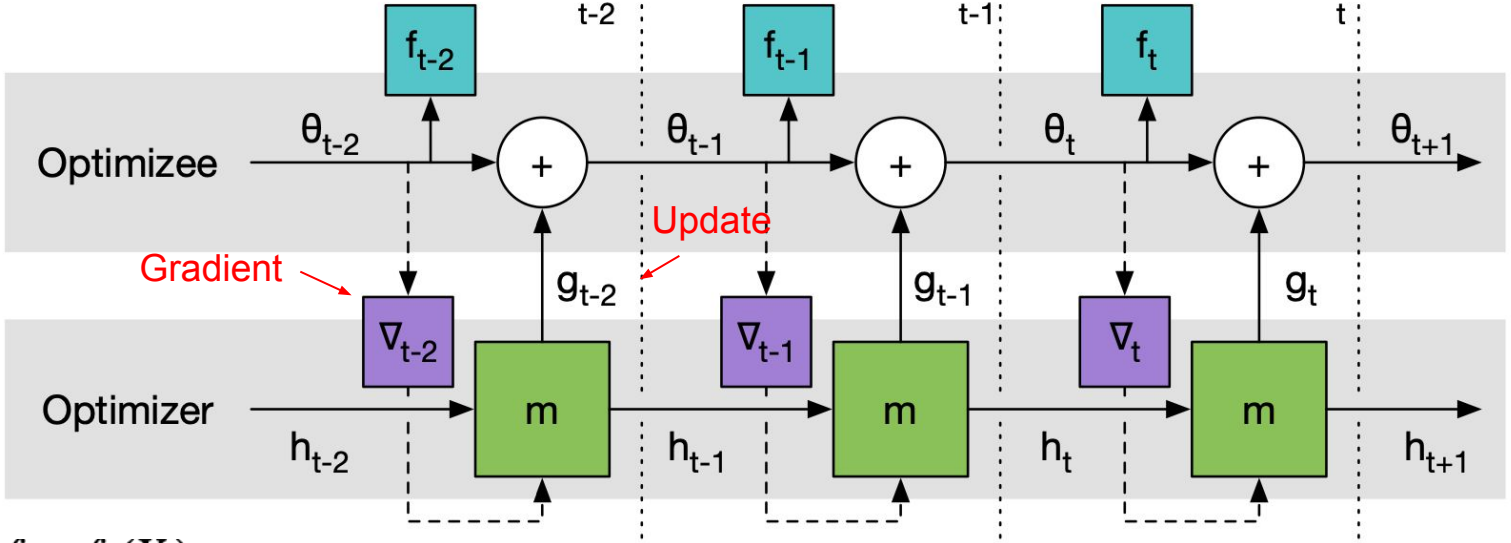


- Optimisation-based base learner (for each task j)

$$\theta_0$$
$$\theta_1 \leftarrow \theta_0 + \text{Update}_\eta(\nabla_{\theta_0} L(f_{\theta_0}(X_1), Y_1))$$
$$\theta_2 \leftarrow \theta_1 + \text{Update}_\eta(\nabla_{\theta_1} L(f_{\theta_1}(X_2), Y_2))$$
$$\vdots$$

- Meta-Learn parameters θ_0, η

Learning to Learn by Gradient Descent by Gradient Descent, LSTM Meta-Learner



$$f_t = f_{\theta_t}(X_t)$$

$$\nabla_t = \frac{d}{d\theta_t} L(f_t, Y_t)$$

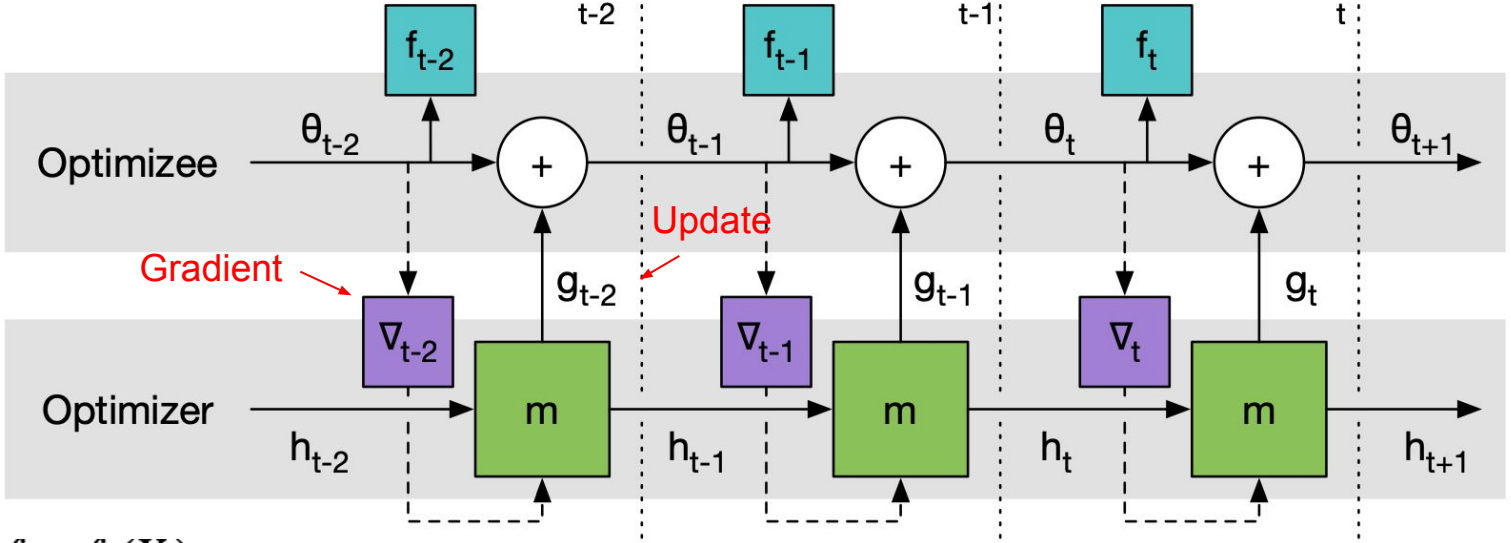
X_t : training inputs in iter t

Y_t : training targets in iter t

$$\theta_{t+1} = \theta_t + g_t$$

[Andrychowicz NeurIPS 2016, Ravi & Larochelle ICLR 2017]

Learning to Learn by Gradient Descent by Gradient Descent, LSTM Meta-Learner



$$f_t = f_{\theta_t}(X_t)$$

$$\nabla_t = \frac{d}{d\theta_t} L(f_t, Y_t)$$

X_t : training inputs in iter t

Y_t : training targets in iter t

ADAM

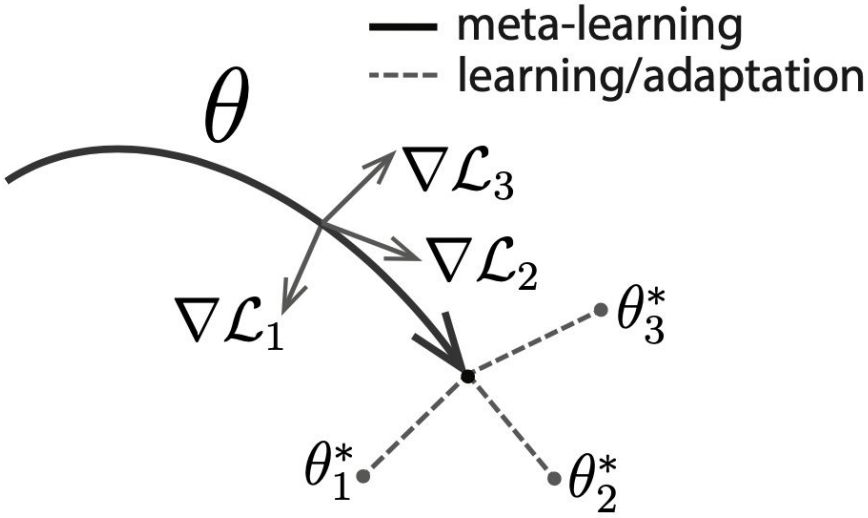
$$\mu_t = \beta_1 \mu_{t-1} + (1-\beta_1)\nabla_t$$

$$V_t = \beta_2 V_{t-1} + (1-\beta_2)\nabla_t^2$$

$$g_t = -\frac{\epsilon}{\sqrt{V_t} + \delta}\mu_t$$

[Andrychowicz NeurIPS 2016, Ravi & Larochelle ICLR 2017]

Model-Agnostic Meta-Learning (MAML)



- One (or few) Gradient step Learner:

$$\theta_j = \eta - \epsilon \nabla_\eta L(f_\eta(X_j), Y_j)$$

- Meta-Learning shared initialisation

$$\eta \leftarrow \eta - \epsilon_0 \frac{d}{d\eta} \sum_j L(f_{\theta_j}(X_{jt}), Y_{jt})$$

- Compute Gradient through inner optimisation step (Second order)

[Finn et al, ICML 2017]

Model-Agnostic Meta-Learning (MAML)

$$\theta_j = \eta - \epsilon \nabla_\eta L(f_\eta(X_j), Y_j)$$

$$\frac{d}{d\eta} L(f_{\theta_j}(X_{jt}), Y_{jt}) = \nabla_{\theta_j} L(f_{\theta_j}(X_{jt}), Y_{jt}) \frac{d\theta_j}{d\eta}$$

$$= \nabla_{\theta_j} L(f_{\theta_j}(X_{jt}), Y_{jt}) \left(I - \epsilon \nabla_\eta^2 L(f_\eta(X_{jt}), Y_{jt}) \right)$$

- Difficulty in computing second term
- Vector-Hessian products can be automatically computed in linear time

[Pearlmutter NECO 1994]

Optimisation-based Meta-Learning

- Two-level optimisation problem:
  - Flexible and applicable to wide range of neural architectures (in theory)
  - Positive inductive bias

- Challenges
  - Sensitive to neural architecture
  - Can be expensive and unstable
  - Difficult to learn for long inner optimisation processes

Outline

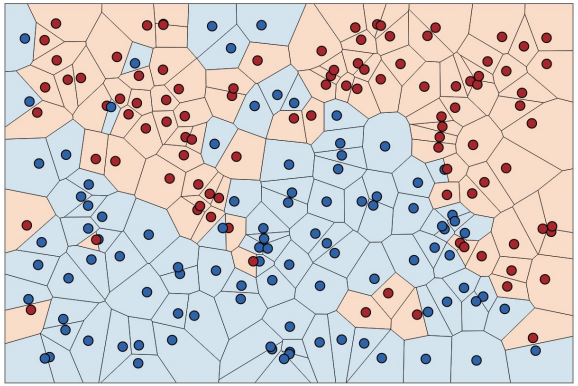
- **Meta-Learning**
  - Overview
  - Optimisation-based
  - **Black-box**
- Continual Learning
  - Overview
  - Regularisation-based
  - Rehearsal/Pseudo-rehearsal based methods

- Discussion & Looking forward

Black-box Meta-Learning

- So far, the base learner is an **optimisation-based** learner
- Not all learning algorithms are optimisation-based

- For example, in k-nearest neighbours, learner simply memorises training data and compares test inputs to make predictions



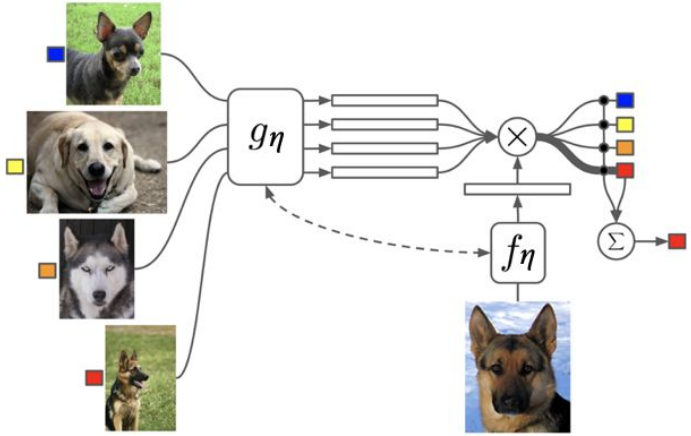
Matching Networks

- Embed each training input $x_i \rightarrow g_\eta(x_i)$
- Embed each test input $x_i \rightarrow f_\eta(x_i)$

- "Softened" 1-NN classifier

$$p(\cdot \mid x_{n+j}, \text{TrainData}) = \sum_{i=1}^{n} \frac{e^{g_\eta(x_i)^\top f_\eta(x_{n+j})}}{\sum_{l=1}^{n} e^{g_\eta(x_l)^\top f_\eta(x_{n+j})}} y_i$$

- Memory-based Meta-Learning
- Metric-based Meta-Learning



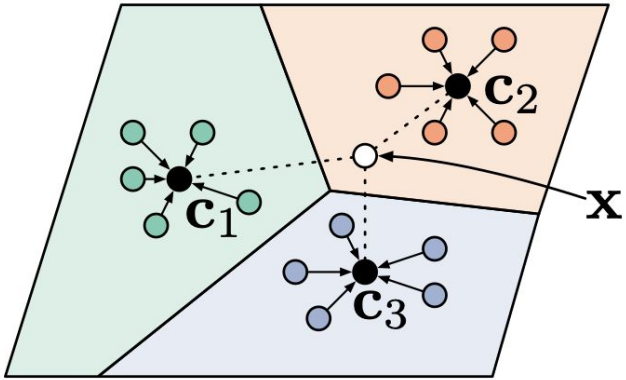
[Vinyals et. al, NeurIPS 2016]

Prototypical Networks

- Embed each input
- Form “prototypes”

$$x_i \rightarrow f_\eta(x_i)$$

$$c_k = \frac{\sum_{i:y_i=k} f_\eta(x_i)}{\sum_{i:y_i=k} 1}$$



- Predict

$$p(y_{n_j} = k \,|\, x_{n+j}, \text{TrainData}) = \frac{e^{-\|f_\eta(x_{n+j}) - c_k\|^2/\sigma^2}}{\sum_{l=1}^{K} e^{-\|f_\eta(x_{n+j}) - c_l\|^2/\sigma^2}}$$

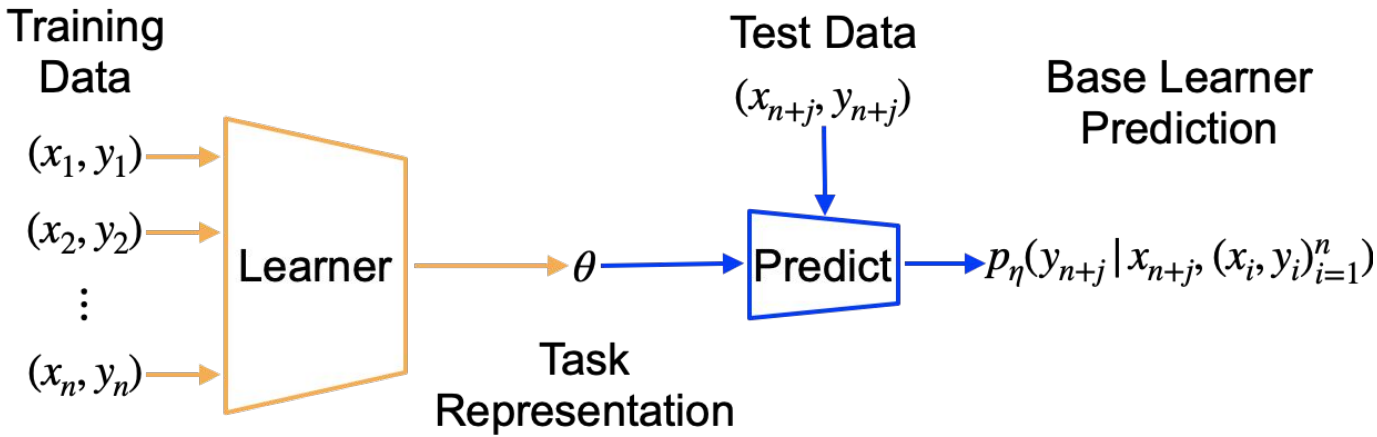
[Snell et al, NeurIPS 2017]

Black-box Meta-Learning

- Learn a differentiable function to map from training data to test predictions
  - Simple: Reduces Meta-Learning to Supervised Learning
  - Broad and flexible

- Challenges
  - Less able to generalise out of meta-training distribution

Task-inference



- Examples:
  - A vector describing preferences of a user in recommender systems
  - A physical description of the properties of a control system with unknown variation (different robot arms)

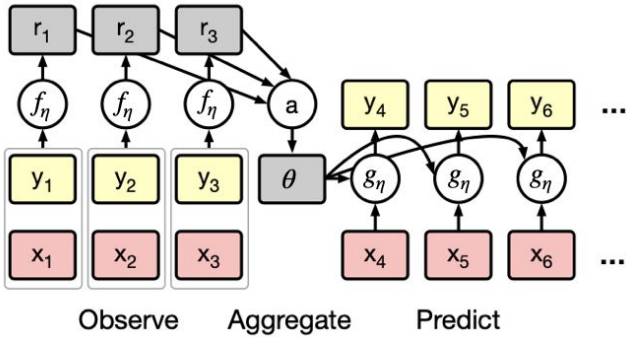
Permutation Invariance

- Assumed that data instances within each task are iid.
- Learner function should be invariant to permutation of the training data

$$\text{Learner}(\eta, \{(x_1, y_1), \ldots, (x_n, y_n)\}) = \text{Learner}(\eta, \{(x_{\pi(1)}, y_{\pi(1)}), \ldots, (x_{\pi(n)}, y_{\pi(n)})\})$$

- MAML, Prototypical Nets, Matching Nets are
- LSTM Meta-Learner are not
- Permutation invariance is an inductive bias

Conditional Neural Processes



- Embed input/output pairs

$$(x_i, y_i) \rightarrow f_\eta(x_i, y_i)$$

- Aggregate embeddings

$$\theta \leftarrow \frac{1}{N} \sum_{i=1}^{N} f_\eta(x_n, y_n)$$

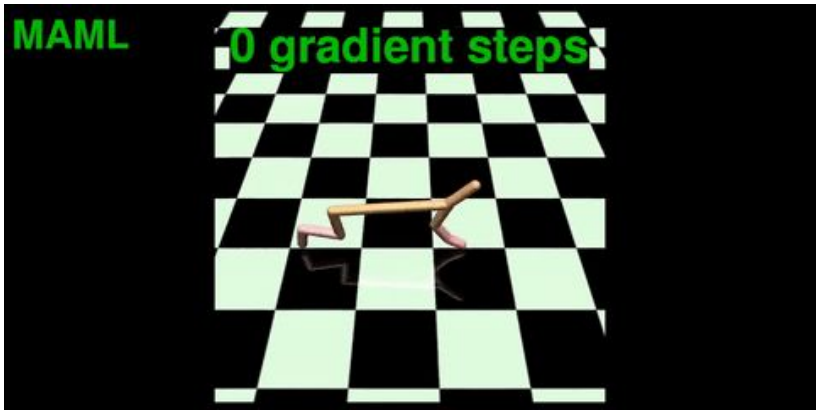
- Predict

$$p(y_t | x_t) = g_\theta(x_t, \theta)$$

- Permutation invariant by design
- Interpret θ as task representation

[Garnelo et al, ICML 2018]

What Meta-Learning can achieve - Reinforcement Learning

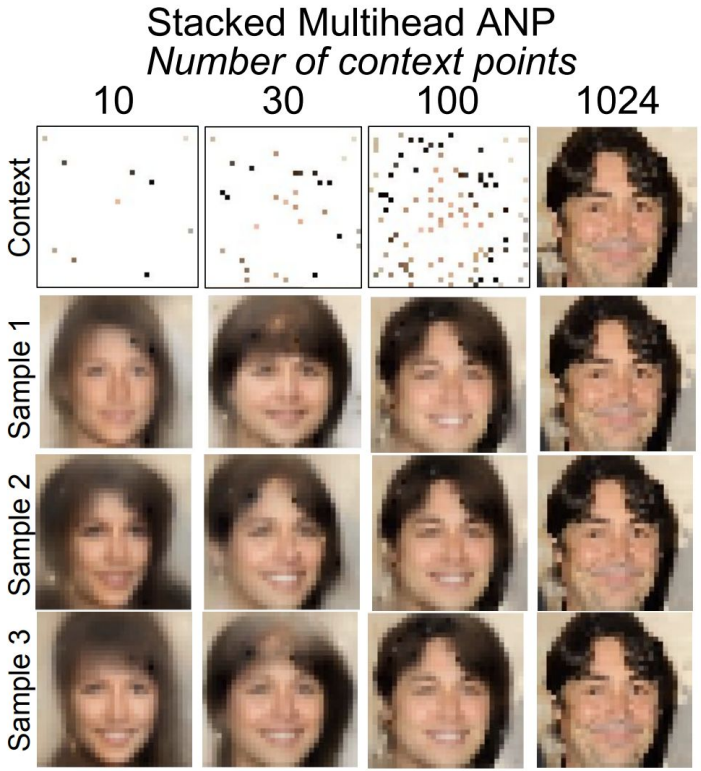


[Finn et al, ICML 2017]

What Meta-Learning can achieve - Reinforcement Learning

[Finn et al, ICML 2017]

What Meta-Learning can achieve - Image Inpainting



[Kim et al, ICRL 2018]

What Meta-Learning can achieve - Recommender Systems

MovieLens 100k

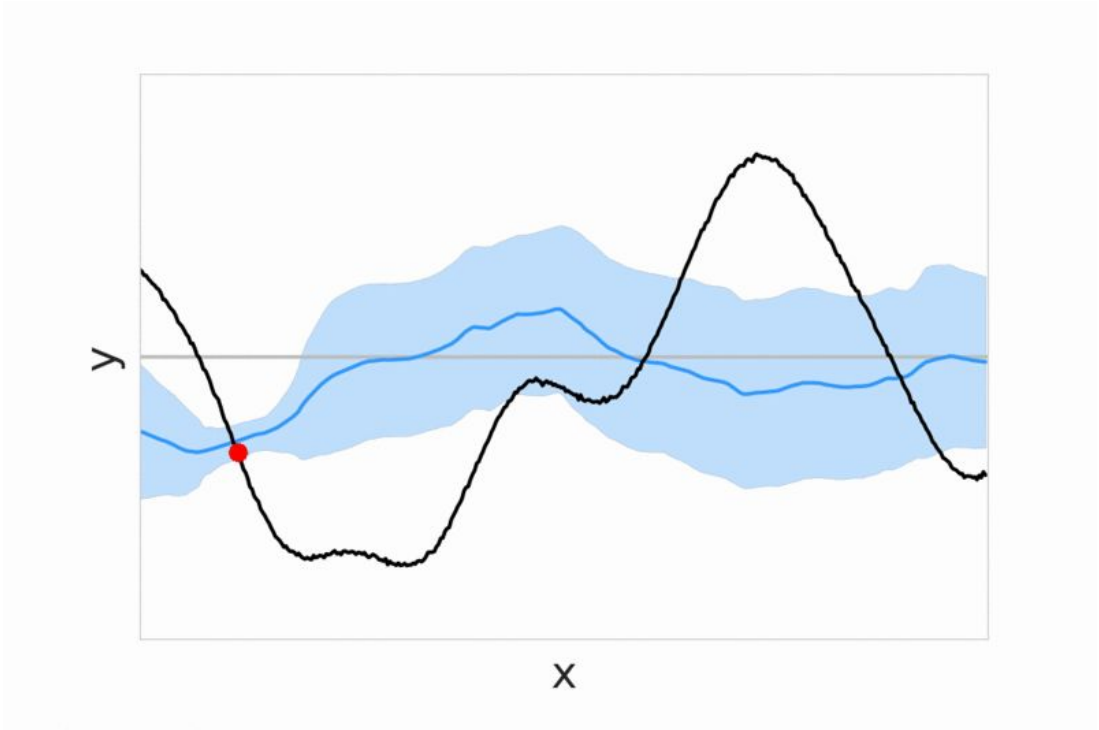
Model	20% of user data	50%	80%
SVD++ (SELF)	1.0517	1.0217	1.0124
Multitask MLP	0.9831	0.9679	0.9507
MAML	0.9593	0.9441	0.9295
NP (random)	$0.9359_{\pm 0.0017}$	$0.9215_{\pm 0.0055}$	$0.9151_{\pm 0.0085}$
NP (Info gain)	$0.9288_{\pm 0.0011}$	$0.8829_{\pm 0.0020}$	$0.8557_{\pm 0.0084}$

MovieLens 20m

Model	20% of user data	50%	80%
SVD++ (MIXED)	$0.9454_{\pm 0.0002}$	$0.9454_{\pm 0.0005}$	$0.9452_{\pm 0.0010}$
Multitask MLP	$0.8570_{\pm 0.0003}$	$0.8401_{\pm 0.0003}$	$0.8348_{\pm 0.0008}$
MAML	$0.8142_{\pm 0.0043}$	$0.7852_{\pm 0.0019}$	$0.7780_{\pm 0.0048}$
NP (random)	$0.7982_{\pm 0.0002}$	$0.7684_{\pm 0.0003}$	$0.7570_{\pm 0.0006}$
NP (Info gain)	$0.7926_{\pm 0.0003}$	$0.7362_{\pm 0.0005}$	$0.6859_{\pm 0.0006}$

[Schwarz et al, arxiv 2019]

What Meta-Learning can achieve - Learning Uncertainty



[Garnelo et al, ICML 2018]

DeepMind

Meta-Learning Q&A

UCL

DeepMind

10min Break

UCL

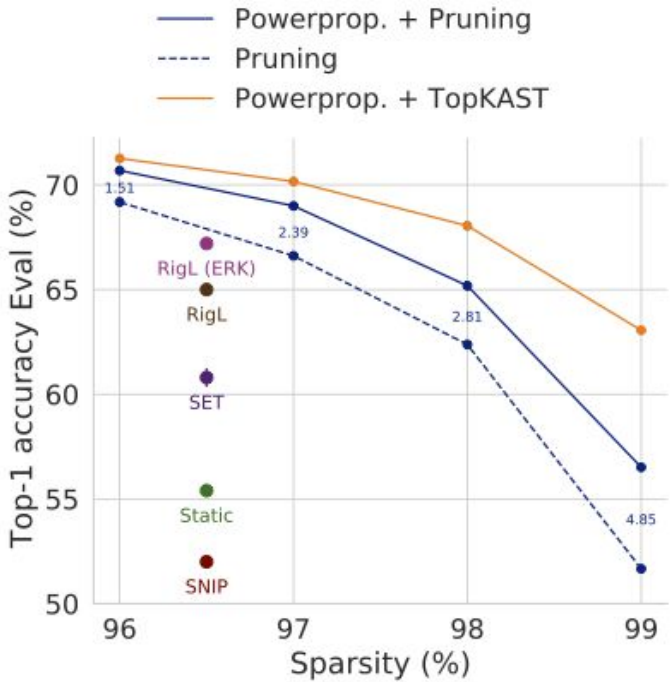
Outline

- Small data: Meta-Learning
    - Overview
    - Optimisation-based
    - Black-box
- **Faster training and inference: Sparsity**
    - **Overview**
    - Explicit Sparsity
    - Implicit Sparsity

- Discussion & Looking forward

Overview

Neural networks are incredibly powerful learning tools. But they are massively overparameterised:
We can train incredibly sparse networks at little performance decrease:



Big potential to cut training and inference cost!

ResNet 50 - Dense: ~76.8% Top-1 accuracy

The Lottery Ticket Hypothesis

So why is this possible? The LTH:

> *"Dense, randomly-initialized, feed-forward networks contain subnetworks (winning tickets) that - when trained in isolation - reach test accuracy comparable to the original network in a similar number of iterations. The winning tickets we find have won the initialization lottery: their connections have initial weights that make training particularly effective."*

[Frankle & Carbin, ICLR 2019]

The Lottery Ticket Hypothesis

So why is this possible? The LTH:

> *"Dense, randomly-initialized, feed-forward networks contain subnetworks (winning tickets) that - when trained in isolation - reach test accuracy comparable to the original network in a similar number of iterations. The winning tickets we find have won the initialization lottery: their connections have initial weights that make training particularly effective."*

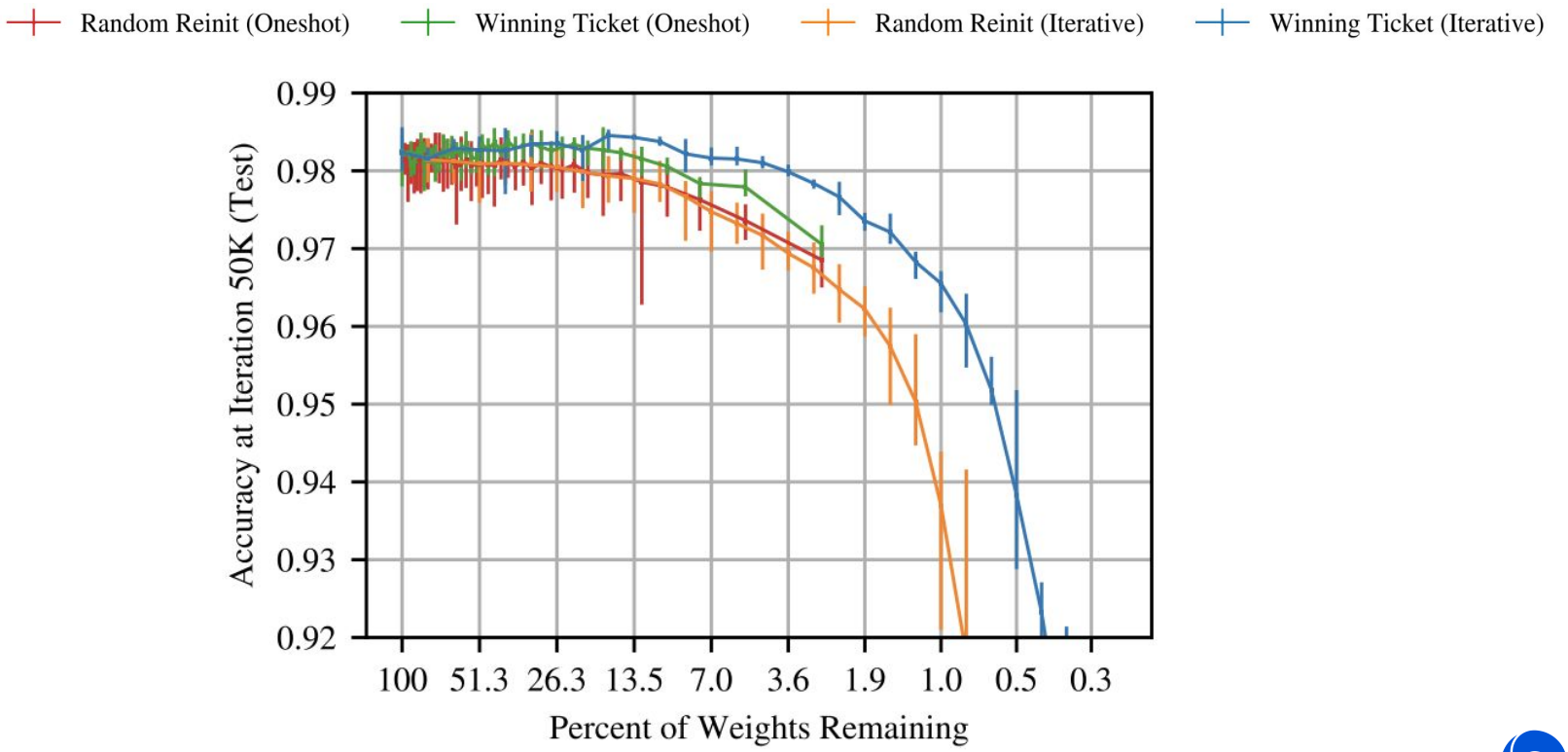
Finding winning tickets

1. Randomly initialize a neural network $f(x; \theta_0)$ (where $\theta_0 \sim \mathcal{D}_\theta$).
2. Train the network for j iterations, arriving at parameters θ_j.
3. Prune $p\%$ of the parameters in θ_j, creating a mask m.
4. Reset the remaining parameters to their values in θ_0, creating the winning ticket $f(x; m \odot \theta_0)$

[Frankle & Carbin, ICLR 2019]

The Lottery Ticket Hypothesis

Reinit: Same structure, different initialisation



[Frankle & Carbin, ICLR 2019]

LeNet MLP (300-100-10) on MNIST

The Lottery Ticket Hypothesis

So why is this possible? The LTH:

> *"Dense, randomly-initialized, feed-forward networks contain subnetworks (winning tickets) that - when trained in isolation - reach test accuracy comparable to the original network in a similar number of iterations. The winning tickets we find have won the initialization lottery: their connections have initial weights that make training particularly effective."*

Finding winning tickets

1. Randomly initialize a neural network $f(x; \theta_0)$ (where $\theta_0 \sim \mathcal{D}_\theta$).
2. Train the network for j iterations, arriving at parameters θ_j.
3. Prune $p\%$ of the parameters in θ_j, creating a mask m.
4. Reset the remaining parameters to their values in θ_0, creating the winning ticket $f(x; m \odot \theta_0)$

[Frankle & Carbin, ICLR 2019]

Outline

- Small data: Meta-Learning
    - Overview
    - Optimisation-based
    - Black-box
- **Faster training and inference: Sparsity**
    - Overview
    - **Explicit Sparsity**
    - Implicit Sparsity

- Discussion & Looking forward

Explicit Sparsity

How do we train networks that are *explicitly sparse*?

- Explicit sparsity = Solution encoded by reduced capacity, resulting from an external (to the learning process) force used to impose frugality (e.g.. masking, regularisation, etc.)

Explicit Sparsity

How do we train networks that are *explicitly sparse*?

- Explicit sparsity = Solution encoded by reduced capacity, resulting from an external (to the learning process) force used to impose frugality (e.g.. masking, regularisation, etc.)

We already saw two ways of doing this:

- *One-shot pruning:* Train a dense network to convergence, p% of weights are pruned, and the surviving weights remain unchanged.
- *Iterative pruning*: Repeatedly train and prune over n rounds; each round prunes p^(1/n) % of the weights that survive the previous round.

Explicit Sparsity

How do we train networks that are *explicitly sparse*?

- Explicit sparsity = Solution encoded by reduced capacity, resulting from an external (to the learning process) force used to impose frugality (e.g.. masking, regularisation, etc.)

We already saw two ways of doing this:

- *One-shot pruning:* Train a dense network to convergence, p% of weights are pruned, and the surviving weights remain unchanged.
- *Iterative pruning*: Repeatedly train and prune over n rounds; each round prunes p^(1/n) % of the weights that survive the previous round. **Tends to work significantly better.**

Explicit Sparsity

How do we train networks that are *explicitly sparse*?

- Explicit sparsity = Solution encoded by reduced capacity, resulting from an external (to the learning process) force used to impose frugality (e.g.. masking, regularisation, etc.)

We already saw two ways of doing this:

- *One-shot pruning:* Train a dense network to convergence, p% of weights are pruned, and the surviving weights remain unchanged.
- *Iterative pruning*: Repeatedly train and prune over n rounds; each round prunes p^(1/n) % of the weights that survive the previous round. **Tends to work significantly better.**

Which parameters to remove? Magnitude-based Pruning

$$f(x,\theta) : f(x,\theta_s) \approx f(\theta,x) + g^T(\theta_s - \theta) + \frac{1}{2}(\theta_s - \theta)^T H(\theta_s - \theta) + \ldots$$

[Magnitude based pruning: Frankle & Carbin, 1995, Iterative pruning: Ström 1997]

Explicit Sparsity

How do we train networks that are *explicitly sparse*?

- Explicit sparsity = Solution encoded by reduced capacity, resulting from an external (to the learning process) force used to impose frugality (e.g.. masking, regularisation, etc.)

We already saw two ways of doing this:

- *One-shot pruning:* Train a dense network to convergence, p% of weights are pruned, and the surviving weights remain unchanged.
- *Iterative pruning*: Repeatedly train and prune over n rounds; each round prunes p^(1/n) % of the weights that survive the previous round. **Tends to work significantly better.**

Which parameters to remove? Magnitude-based Pruning

$$f(x,\theta) : f(x,\theta_s) \approx f(\theta,x) + g^T(\theta_s - \theta) + \frac{1}{2}(\theta_s - \theta)^T H(\theta_s - \theta) + \ldots$$

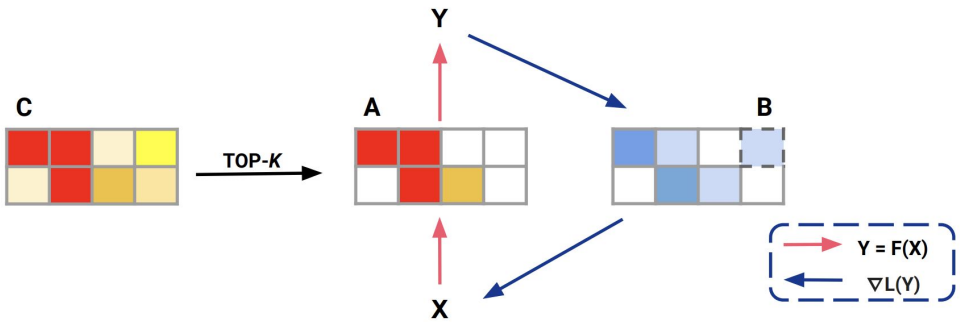
- Higher-order derivatives useful but computationally intractable to do so for very large modern networks. However, as error scales with $(\theta_s - \theta)$, minimise that norm instead

[Magnitude based pruning: Frankle & Carbin, 1995, Iterative pruning: Ström 1997]

TopKAST: Top-K Always Sparse Training

One-shot and iterative pruning sparsify a dense network. Can we stay sparse throughout training?

- Sparse forward mask: $A^t = \{i | \theta_i^t \in \texttt{TopK}(\theta^t, D)\}$ $\quad \alpha_i^t = \begin{cases} \theta_i^t & \texttt{if } i \in A^t \\ 0 & \texttt{otherwise} \end{cases}$



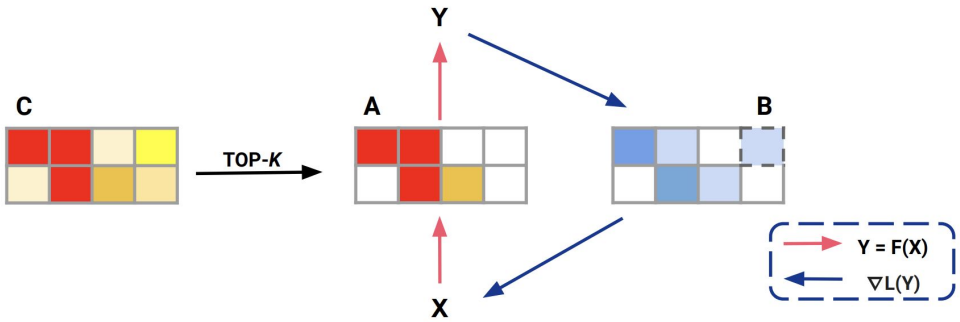
[Jayakumar et al., NeurIPS 2020]

TopKAST: Top-K Always Sparse Training

One-shot and iterative pruning sparsify a dense network. Can we stay sparse throughout training?

- Sparse forward mask: $A^t = \{i | \theta_i^t \in \texttt{TopK}(\theta^t, D)\}$ $\alpha_i^t = \begin{cases} \theta_i^t & \texttt{if } i \in A^t \\ 0 & \texttt{otherwise} \end{cases}$

- Sparse backward mask: $B^t = \{i | \theta_i^t \in \texttt{TopK}(\theta^t, D + M)\}$ $\Delta_{\theta_i^t} = \begin{cases} -\eta \nabla_{\alpha^t} L(y, x, \alpha^t)_i & \texttt{if } i \in B \\ 0 & \texttt{otherwise} \end{cases}$



[Jayakumar et al., NeurIPS 2020]

TopKAST: Exploration

- *Exploratory stage:* At each iteration we select a different active set A, and its corresponding α, and perform one update step on θ using gradients obtained from the loss on f(α, x) and the regularizer.

- *Refinement stage:* The active set A effectively becomes fixed, as we settle on a stable pattern of non-zero weights which then undergo fine-tuning to their optimal values

[Jayakumar et al., NeurIPS 2020]

TopKAST: Exploration

- *Exploratory stage:* At each iteration we select a different active set A, and its corresponding α, and perform one update step on θ using gradients obtained from the loss on f(α, x) and the regularizer.

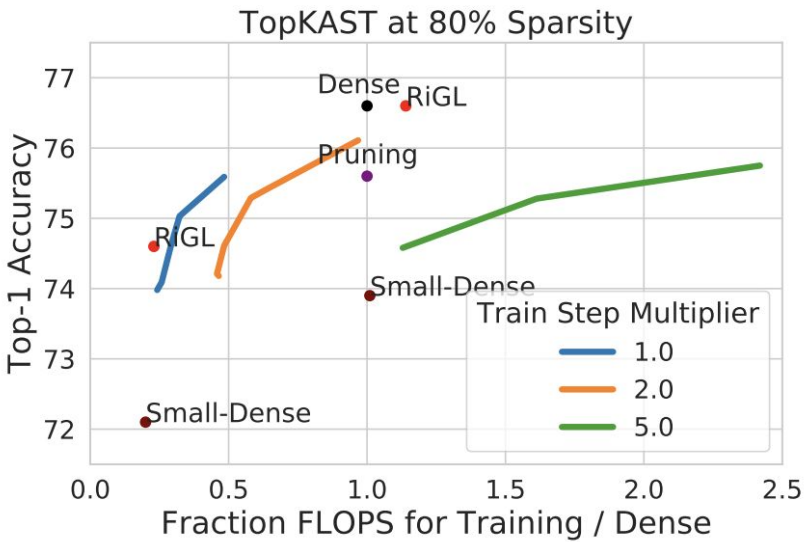
- *Refinement stage:* The active set A effectively becomes fixed, as we settle on a stable pattern of non-zero weights which then undergo fine-tuning to their optimal values

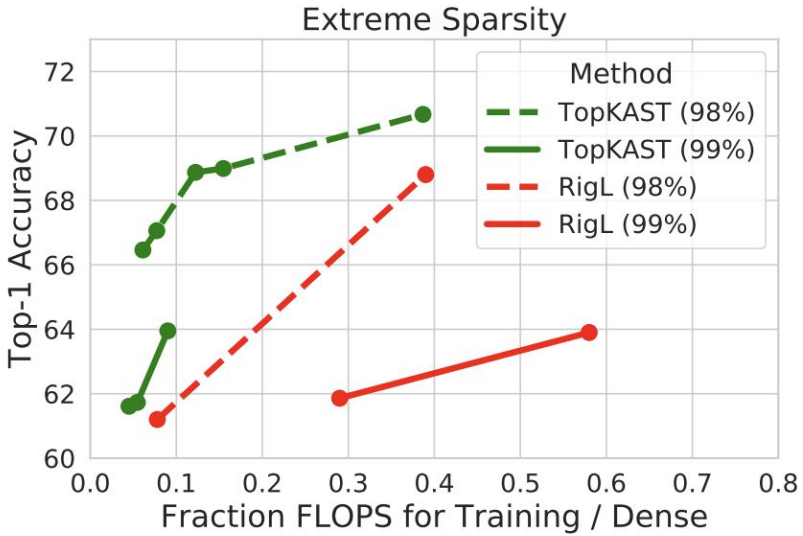
Could this lead to a rich-get-richer scheme? Exploration loss:

$$Loss_R(\alpha_i^t) = \begin{cases} |\theta_i^t| & \texttt{if } i \in A^t \\ \frac{|\theta_i^t|}{D} & \texttt{if } i \in B^t \setminus A^t \\ 0 & else \end{cases}$$

[Jayakumar et al., NeurIPS 2020]

TopKAST: FLOPS trade-off





[Jayakumar et al., NeurIPS 2020]

Training/Inference FLOPs savings

Method	Top-1 Accuracy	FLOPs (Train)	FLOPs (Test)	Top-1 Accuracy	FLOPs (Train)	FLOPs (Test)
Dense	76.8±0.09	1x (3.2e18)	1x (8.2e9)			
	S=0.95			S=0.965		
Static	59.5+-0.11	0.23x	0.08x	55.4+-0.06	0.13x	0.07x
Snip	57.8+-0.40	0.23x	0.08x	52.0+-0.20	0.13x	0.07x
SET	64.4+-0.77	0.23x	0.08x	60.8+-0.45	0.13x	0.07x
RigL	67.5+-0.10	0.23x	0.08x	65.0+-0.28	0.13x	0.07x
$\text{RigL}_{5\times}$	73.1+-0.12	1.14x	0.08x	71.1+-0.20	0.66x	0.07x
Static (ERK)	72.1±0.04	0.42x	0.42x	67.7±0.12	0.24x	0.24x
RigL (ERK)	69.7+-0.17	0.42x	0.12x	67.2+-0.06	0.25x	0.11x
$\text{RigL}_{5\times}$ (ERK)	74.5+-0.09	2.09x	0.12x	72.7+-0.02	1.23x	0.11x
SNFS (ERK)	70.0+-0.04	0.61x	0.12x	67.1+-0.72	0.50x	0.11x
Pruning* (Gale)	70.6	0.56x	0.08x	n/a	0.51x	0.07x
$\text{Pruning}_{1.5\times}$ (Gale)	72.7	0.84x	0.08x	69.26	0.76x	0.07x

[Evci et al., ICML 2020]

Training/Inference FLOPs savings

Method	Top-1 Accuracy	FLOPs (Train)	FLOPs (Test)	Top-1 Accuracy	FLOPs (Train)	FLOPs (Test)
Dense	76.8±0.09	1x (3.2e18)	1x (8.2e9)			
	S=0.95			S=0.965		
Static	59.5+-0.11	0.23x	0.08x	55.4+-0.06	0.13x	0.07x
Snip	57.8+-0.40	0.23x	0.08x	52.0+-0.20	0.13x	0.07x
SET	64.4+-0.77	0.23x	0.08x	60.8+-0.45	0.13x	0.07x
RigL	67.5+-0.10	0.23x	0.08x	65.0+-0.28	0.13x	0.07x
$\text{RigL}_{5\times}$	73.1+-0.12	1.14x	0.08x	71.1+-0.20	0.66x	0.07x
Static (ERK)	72.1±0.04	0.42x	0.42x	67.7±0.12	0.24x	0.24x
RigL (ERK)	69.7+-0.17	0.42x	0.12x	67.2+-0.06	0.25x	0.11x
$\text{RigL}_{5\times}$ (ERK)	74.5+-0.09	2.09x	0.12x	72.7+-0.02	1.23x	0.11x
SNFS (ERK)	70.0+-0.04	0.61x	0.12x	67.1+-0.72	0.50x	0.11x
Pruning* (Gale)	70.6	0.56x	0.08x	n/a	0.51x	0.07x
$\text{Pruning}_{1.5\times}$ (Gale)	72.7	0.84x	0.08x	69.26	0.76x	0.07x

[Evci et al., ICML 2020]

Outline

- Small data: Meta-Learning
    - Overview
    - Optimisation-based
    - Black-box
- **Faster training and inference: Sparsity**
    - Overview
    - Explicit Sparsity
    - **Implicit Sparsity**

- Discussion & Looking forward

Powerpropagation: Key idea

How do we train networks that are *inherently sparse*?

Powerpropagation: Key idea

How do we train networks that are *inherently sparse*?

- Inherent sparsity = Solution encoded by reduced capacity, resulting from the learning process itself, without any explicit force to impose frugality (i.e. no masking, regularisation, etc.)

Powerpropagation: Key idea

How do we train networks that are *inherently sparse*?

- Inherent sparsity = Solution encoded by reduced capacity, resulting from the learning process itself, without any explicit force to impose frugality (i.e. no masking, regularisation, etc.)

- Our idea: Create a “rich get richer” dynamic. Leave low-magnitude parameters largely unaffected by gradient updates.

  → This can be achieved by ensuring the parameter magnitude appears in the gradient update

Powerpropagation: Key idea

How do we train networks that are *inherently sparse*?

- Inherent sparsity = Solution encoded by reduced capacity, resulting from the learning process itself, without any explicit force to impose frugality (i.e. no masking, regularisation, etc.)

- Our idea: Create a “rich get richer” dynamic. Leave low-magnitude parameters largely unaffected by gradient updates.

  → This can be achieved by ensuring the parameter magnitude appears in the gradient update

- **Powerpropagation in a nutshell:** Raise the weights of your network to the a-th power (preserving the sign). This will result in the magnitude of the weight appearing in the gradient (chain rule), thus encouraging the “rich get richer” dynamics

Powerpropagation: Key idea

How do we train networks that are *inherently sparse*?

- Inherent sparsity = Solution encoded by reduced capacity, resulting from the learning process itself, without any explicit force to impose frugality (i.e. no masking, regularisation, etc.)

- Our idea: Create a “rich get richer” dynamic. Leave low-magnitude parameters largely unaffected by gradient updates.

  → This can be achieved by ensuring the parameter magnitude appears in the gradient update

- **Powerpropagation in a nutshell:** Raise the weights of your network to the a-th power (preserving the sign). This will result in the magnitude of the weight appearing in the gradient (chain rule), thus encouraging the “rich get richer” dynamics

- Motivation from (Vaškevičius et al, NeurIPS 2019: Implicit regularization for optimal sparse recovery). Sparsity for matrix factorisation: $w = v \odot v - u \odot u$

Powerpropagation

- The Powerpropagation reparameterisation: $\Psi(\phi_i) = \phi_i |\phi_i|^{\alpha-1} = \theta_i$

Powerpropagation

- The Powerpropagation reparameterisation: $\Psi(\phi_i) = \phi_i |\phi_i|^{\alpha-1} = \theta_i$

$$\frac{\partial \mathcal{L}(\cdot, \Psi(\phi))}{\partial \phi} = \frac{\partial \mathcal{L}}{\partial \Psi(\phi)} \frac{\partial \Psi(\phi)}{\partial \phi} = \frac{\partial \mathcal{L}}{\partial \Psi(\phi)} \operatorname{diag}(\alpha |\phi|^{\circ \alpha - 1}).$$

Standard Update

Powerprop. scaling

Powerpropagation

- The Powerpropagation reparameterisation: $\Psi(\phi_i) = \phi_i |\phi_i|^{\alpha-1} = \theta_i$

$$\frac{\partial \mathcal{L}(\cdot, \Psi(\phi))}{\partial \phi} = \frac{\partial \mathcal{L}}{\partial \Psi(\phi)} \frac{\partial \Psi(\phi)}{\partial \phi} = \frac{\partial \mathcal{L}}{\partial \Psi(\phi)} \operatorname{diag}(\alpha |\phi|^{\circ \alpha - 1}).$$

Standard Update

Powerprop. scaling

Properties:

- a=1 recovers the baseline, i.e. standard gradient descent
- Zero is a critical point of any weight for a>1.
- In addition, zero is surrounded by a plateau: Weights are less likely to change sign

Powerpropagation: Details

- How to ensure a good initialisation?
  - Set $\phi_i \leftarrow \text{sign}(\theta_i) \cdot \sqrt[\alpha]{|\theta_i|}$, $\theta_i \sim p(\theta)$
  - Preserves identical results at first forward pass

Powerpropagation: Details

- How to ensure a good initialisation?
  - Set $\phi_i \leftarrow \text{sign}(\theta_i) \cdot \sqrt[\alpha]{|\theta_i|}$, $\theta_i \sim p(\theta)$
  - Preserves identical results at first forward pass

- Modern optimisers often by a decaying average over past squared gradients (RMSProp, Adadelta, Adam, ...): $\Delta\theta_t = -\frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$
  - This will be approx. proportional to the powerpropagation scaling term, reducing the desired “rich get richer” effect.

Powerpropagation: Details

- How to ensure a good initialisation?
  - Set $\phi_i \leftarrow \text{sign}(\theta_i) \cdot \sqrt[\alpha]{|\theta_i|}$, $\theta_i \sim p(\theta)$
  - Preserves identical results at first forward pass

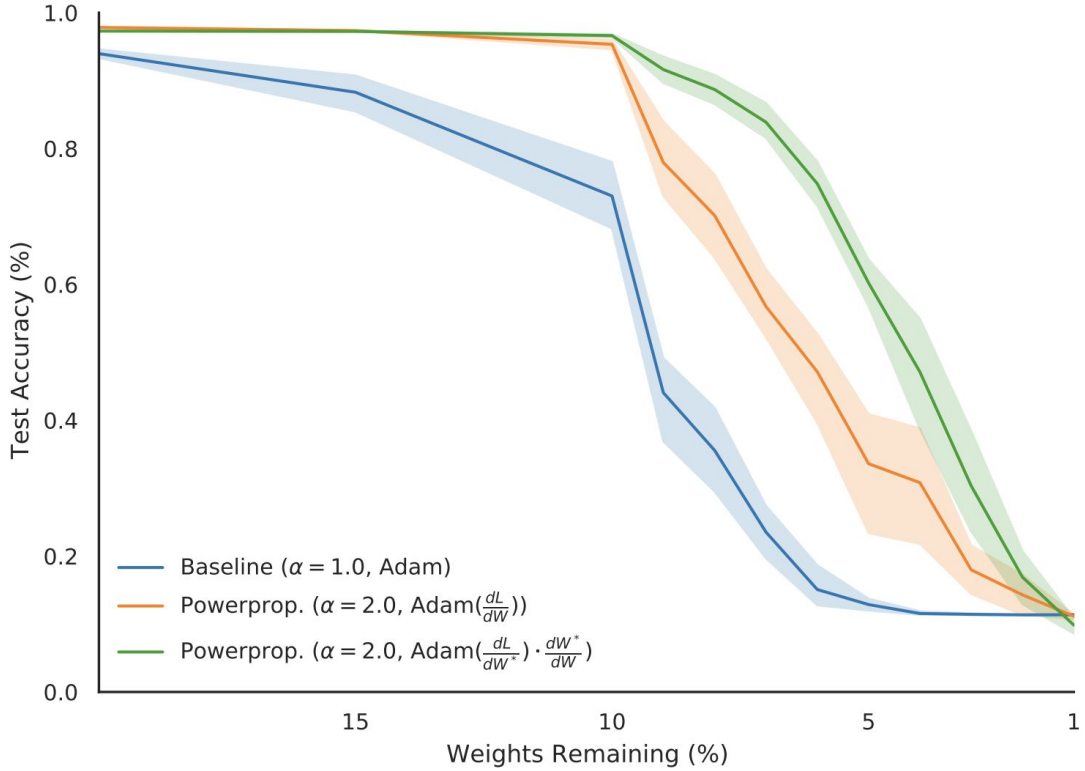
- Modern optimisers often by a decaying average over past squared gradients (RMSProp, Adadelta, Adam, ...): $\Delta\theta_t = -\frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$
  - This will be approx. proportional to the powerpropagation scaling term, reducing the desired "rich get richer" effect.
  - Assume weights don't change: Each gradient is scaled by the same value: $\text{diag}(\alpha|\phi|^{\circ 2(\alpha-1)})$
  - This factors out in decaying average summation: Optimiser will undo the scaling

Powerpropagation: Details

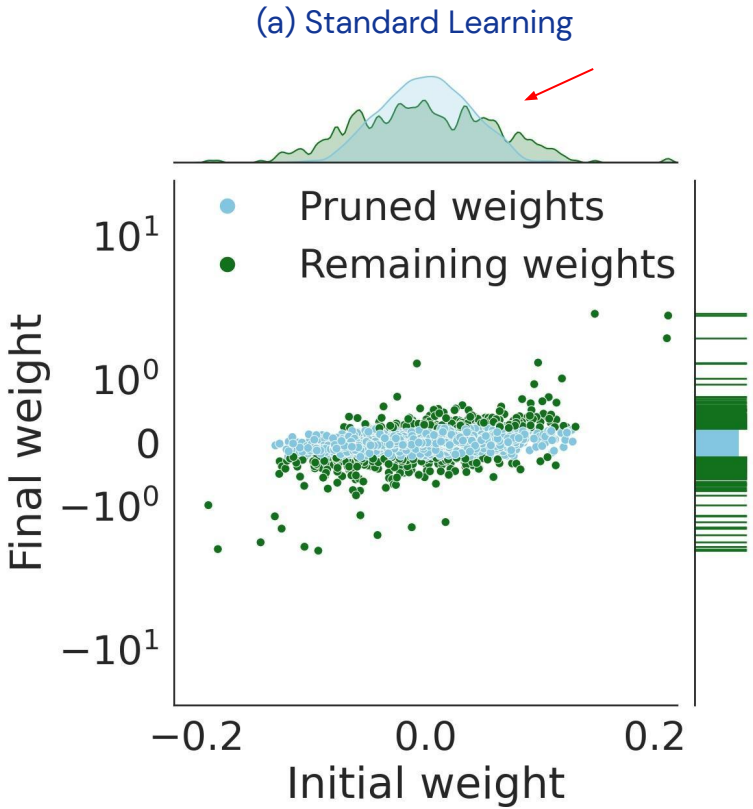
- How to ensure a good initialisation?
  - Set $\phi_i \leftarrow \text{sign}(\theta_i) \cdot \sqrt[\alpha]{|\theta_i|}$, $\theta_i \sim p(\theta)$
  - Preserves identical results at first forward pass

- Modern optimisers often by a decaying average over past squared gradients (RMSProp, Adadelta, Adam, ...): $\Delta\theta_t = -\frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$
  - This will be approx. proportional to the powerpropagation scaling term, reducing the desired "rich get richer" effect.
  - **Solution:** Pretend that the exponentiated parameters are the de-facto parameters of the model. Compute update wrt. to this virtual target, multiply by partial derivative. This is a correct update (Follows typical proof steps taken in the target propagation literature) $\Delta\phi = optim\left(\frac{\partial \mathcal{L}}{\partial \Psi(\phi)}\right) \text{diag}(\alpha|\phi|^{\circ\alpha-1})$

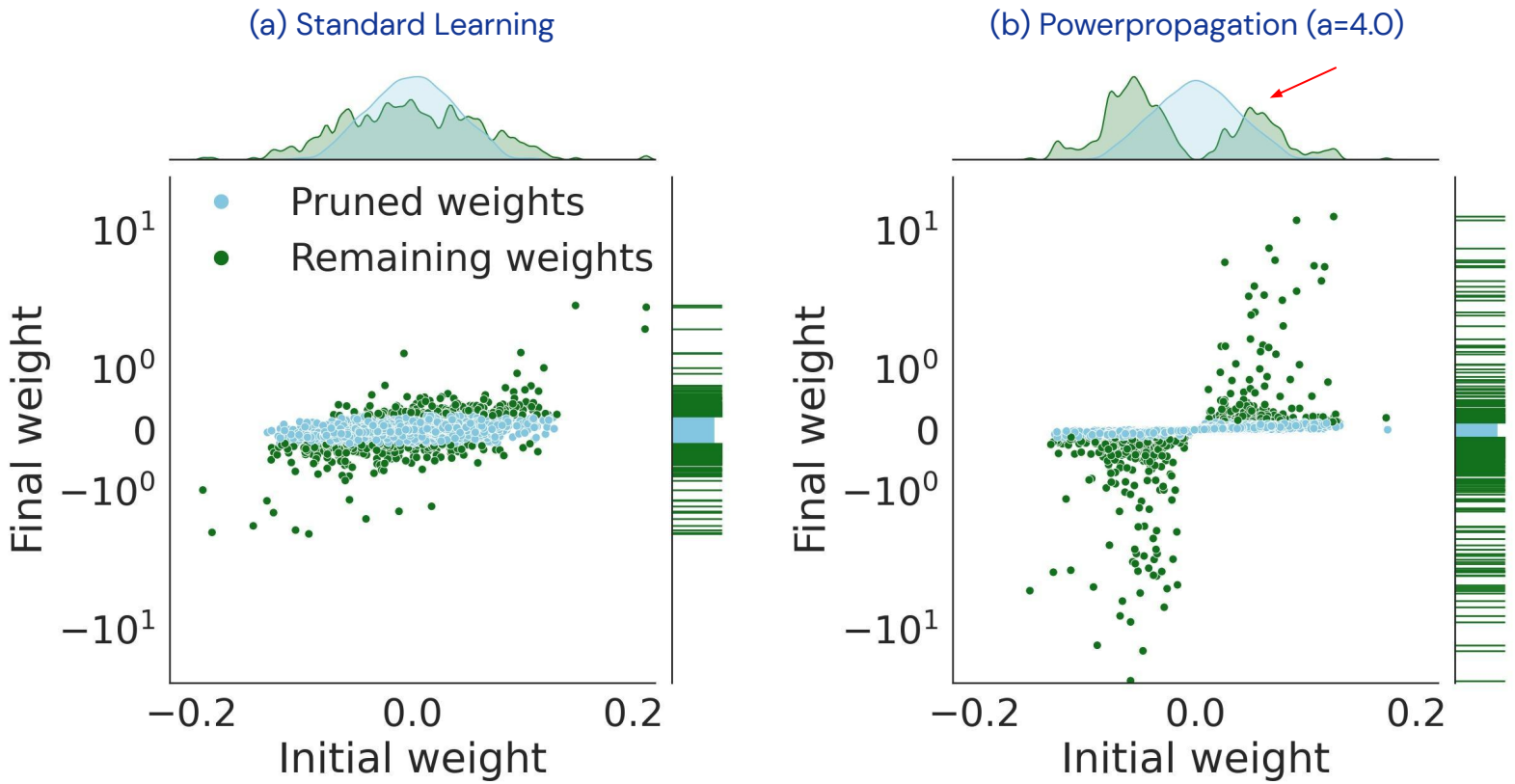
Powerpropagation with modern optimisers



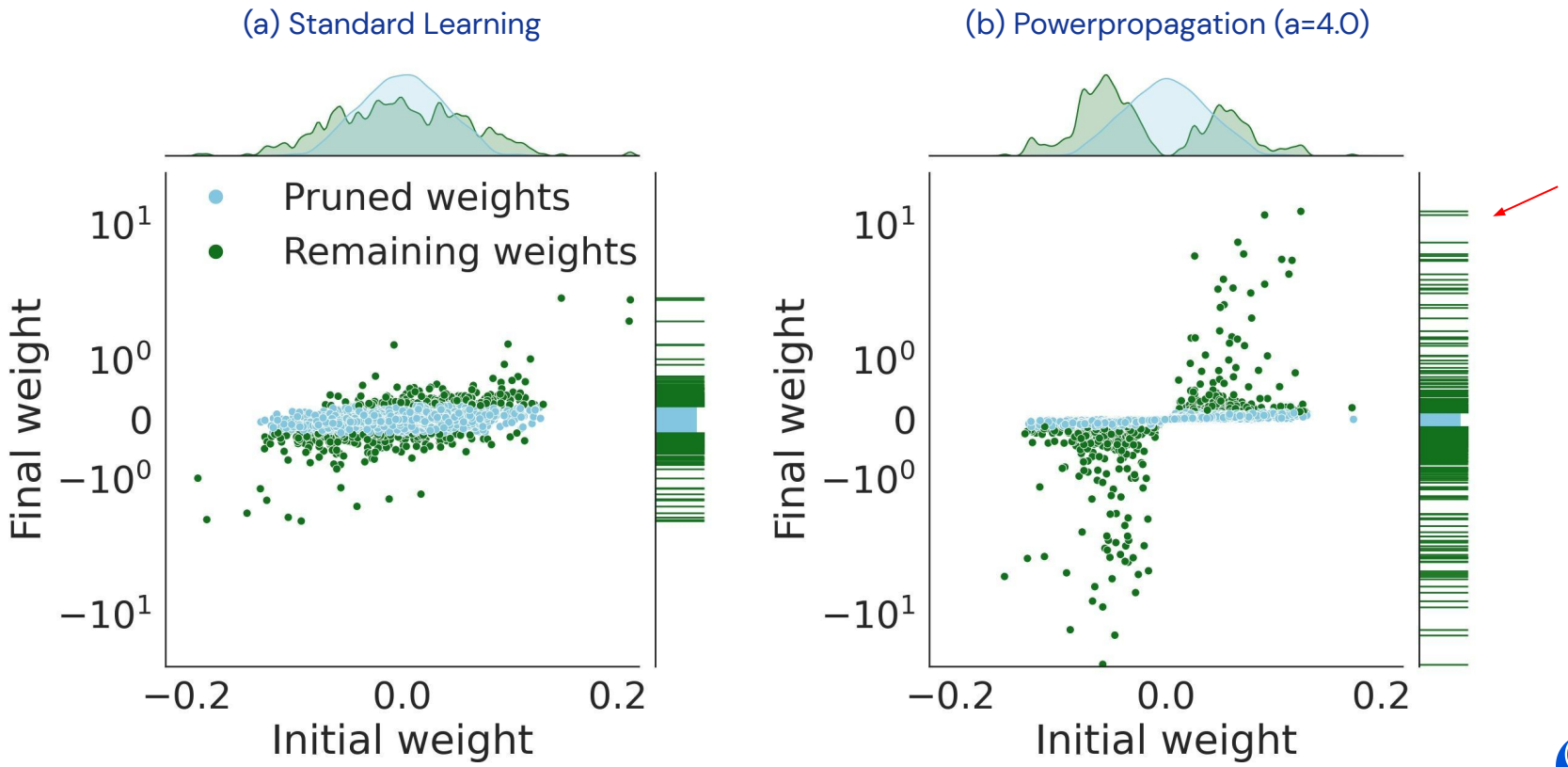
Effect on weight distribution



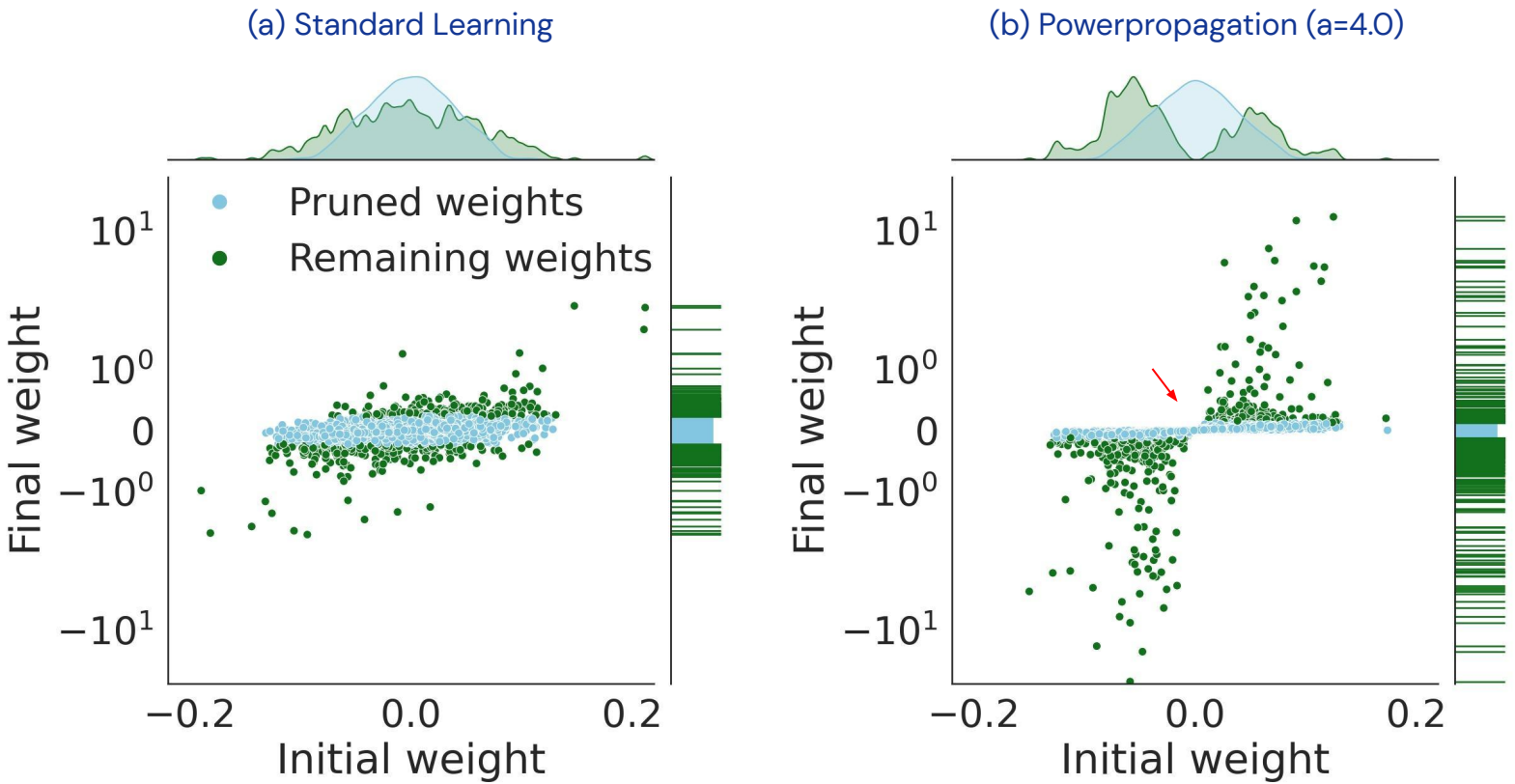
Effect on weight distribution



Effect on weight distribution

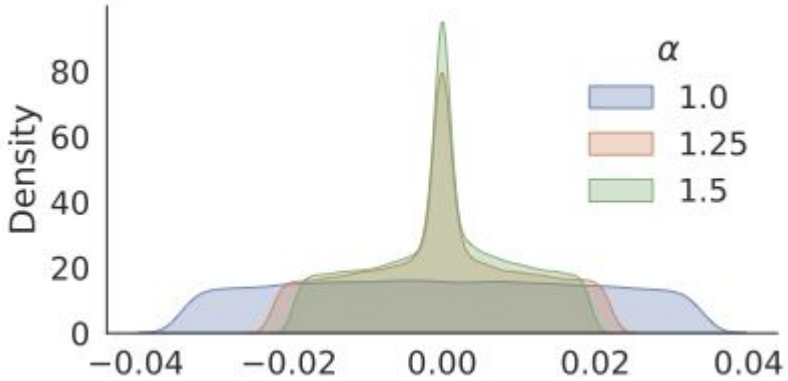


Effect on weight distribution



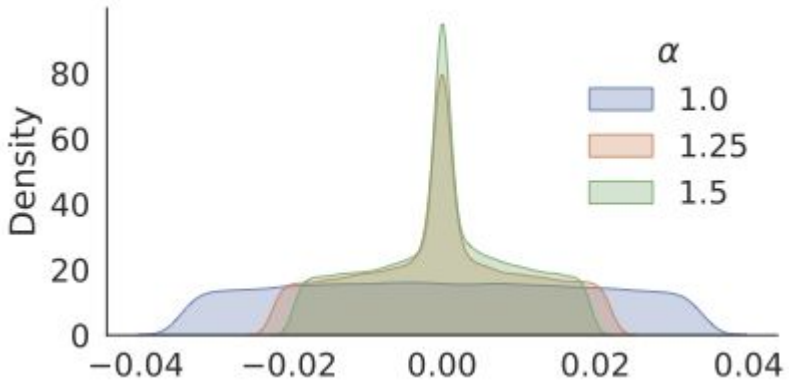
Effect on weight distribution

(a) 100k smallest weights

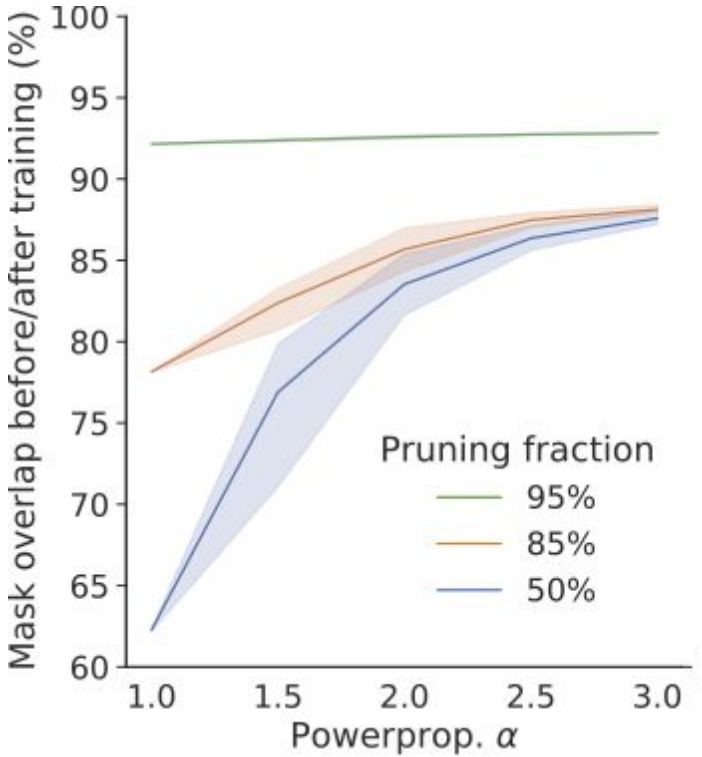


Effect on weight distribution

(a) 100k smallest weights



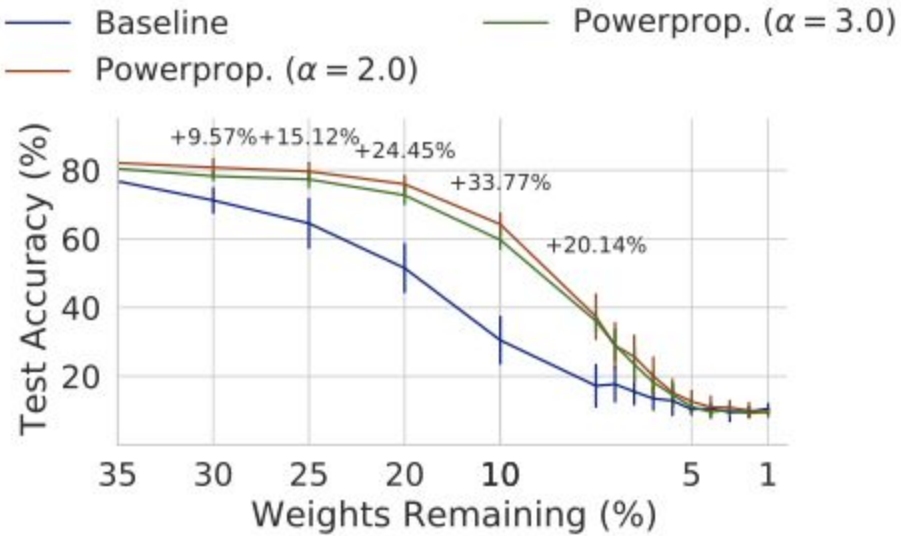
(b) Mask overlap



Inherent sparsity: One-shot pruning

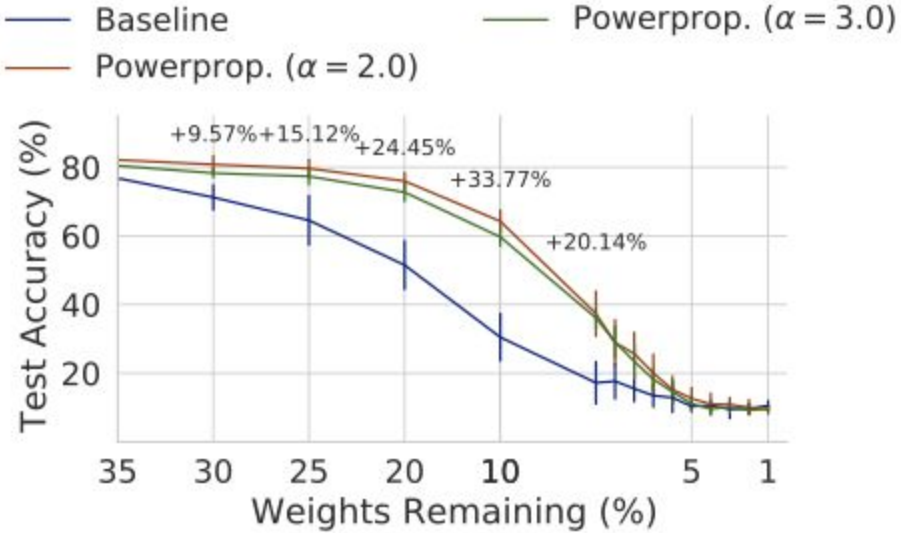
Inherent sparsity: One-shot pruning (AlexNet)

(a) CIFAR-10

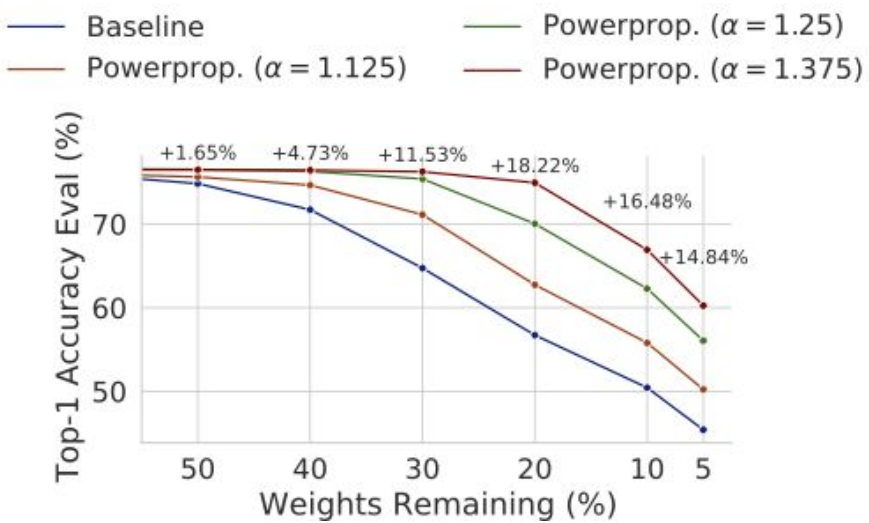


Inherent sparsity: One-shot pruning (ResNet50)

(a) CIFAR-10

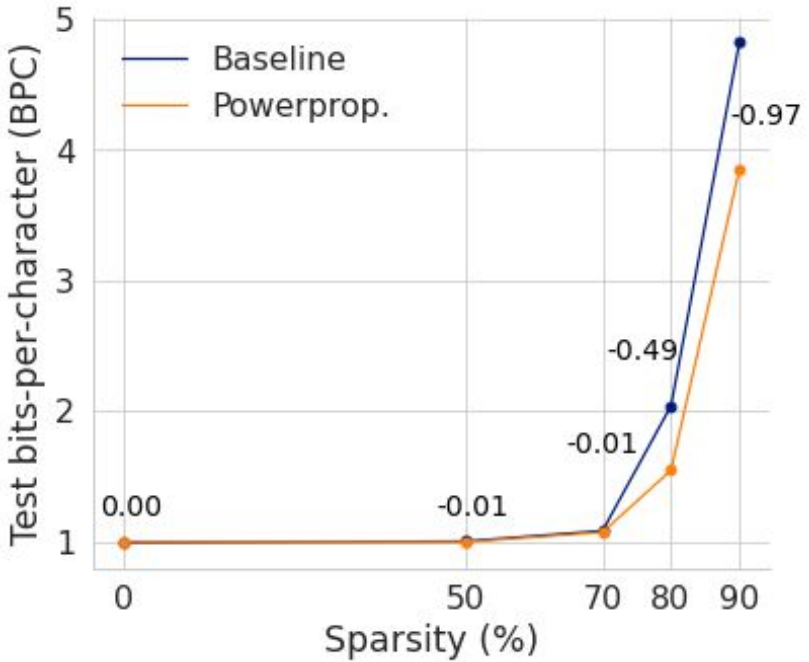


(b) ImageNet



Inherent sparsity: One-shot pruning (TransformerXL)

(a) enwik8



ImageNet w/ ResNet50 (Following RigL [Utku et al., ICML 2020])

Method	0%		
Dense	76.8 ± 0.09		
	80%	**90%**	**95%**
Static	70.6 ± 0.06	65.8 ± 0.04	59.5 ± 0.11
DSR	73.3	71.6	
SNFS	74.2	72.3	
SNIP	72.0 ± 0.10	67.2 ± 0.12	57.8 ± 0.40
RigL	74.6 ± 0.06	72.0 ± 0.05	67.5 ± 0.10
Iterative pruning	75.6	73.9	70.6

ImageNet w/ ResNet50 (Dense -> Sparse)

Method	0%		
Dense	76.8 ± 0.09		
	80%	**90%**	**95%**
Static	70.6 ± 0.06	65.8 ± 0.04	59.5 ± 0.11
DSR	73.3	71.6	
SNFS	74.2	72.3	
SNIP	72.0 ± 0.10	67.2 ± 0.12	57.8 ± 0.40
RigL	74.6 ± 0.06	72.0 ± 0.05	67.5 ± 0.10
Iterative pruning	75.6	73.9	70.6
Iterative pruning (ours)	75.3 ± 0.07	73.7 ± 0.14	70.6 ± 0.05
Powerprop. + Iter. Pruning	**75.7** ± 0.05	**74.4** ± 0.02	**72.1** ± 0.00

ImageNet w/ ResNet50 (Sparse -> Sparse)

Method	0%		
Dense	76.8 ± 0.09		
	80%	**90%**	**95%**
Static	70.6 ± 0.06	65.8 ± 0.04	59.5 ± 0.11
DSR	73.3	71.6	
SNFS	74.2	72.3	
SNIP	72.0 ± 0.10	67.2 ± 0.12	57.8 ± 0.40
RigL	74.6 ± 0.06	72.0 ± 0.05	67.5 ± 0.10
Iterative pruning	75.6	73.9	70.6
Iterative pruning (ours)	75.3 ± 0.07	73.7 ± 0.14	70.6 ± 0.05
Powerprop. + Iter. Pruning	**75.7** ± 0.05	**74.4** ± 0.02	**72.1** ± 0.00
TopKAST† [14]	75.47 ± 0.03	74.65 ± 0.03	**72.73** ± 0.10
Powerprop. + TopKAST†	**75.75** ± 0.05	**74.74** ± 0.04	**72.89** ± 0.10

@50% Backward sparsity

ImageNet w/ ResNet50 (Sparse -> Sparse)

Method	0%		
Dense	76.8 ± 0.09		
	80%	**90%**	**95%**
Static	70.6 ± 0.06	65.8 ± 0.04	59.5 ± 0.11
DSR	73.3	71.6	
SNFS	74.2	72.3	
SNIP	72.0 ± 0.10	67.2 ± 0.12	57.8 ± 0.40
RigL	74.6 ± 0.06	72.0 ± 0.05	67.5 ± 0.10
Iterative pruning	75.6	73.9	70.6
Iterative pruning (ours)	75.3 ± 0.07	73.7 ± 0.14	70.6 ± 0.05
Powerprop. + Iter. Pruning	**75.7** ± 0.05	**74.4** ± 0.02	**72.1** ± 0.00
TopKAST† [14]	75.47 ± 0.03	74.65 ± 0.03	**72.73** ± 0.10
Powerprop. + TopKAST†	**75.75** ± 0.05	**74.74** ± 0.04	**72.89** ± 0.10
TopKAST* [14]	76.08 ± 0.02	75.13 ± 0.03	73.19 ± 0.02
Powerprop. + TopKAST*	**76.24** ± 0.07	**75.23** ± 0.02	**73.25** ± 0.02

@0% Backward sparsity

ImageNet w/ ResNet50 (Erdos-Renyi Kernel)

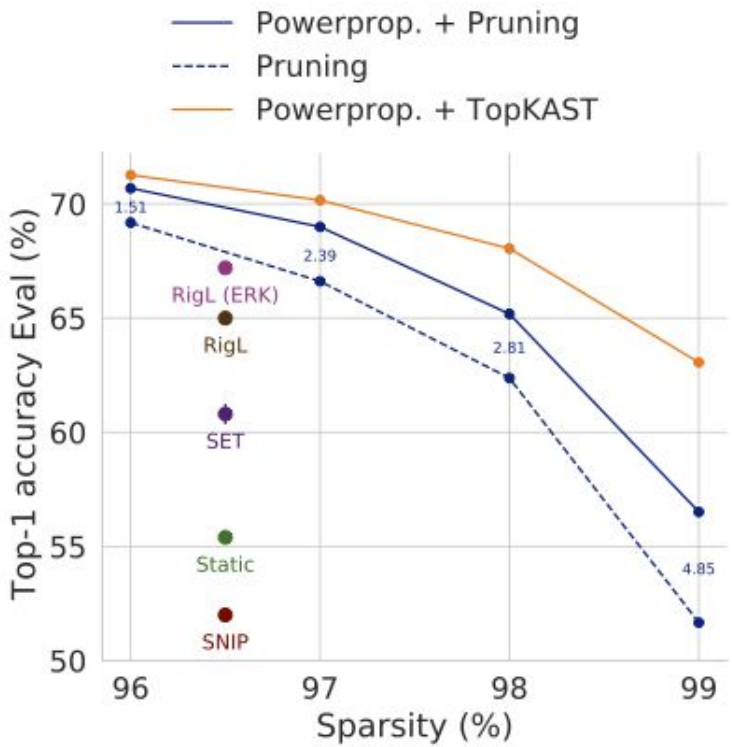
Method	0%				
Dense	76.8 ± 0.09				
	80%	**90%**	**95%**	**80% (ERK)**	**90% (ERK)**
Static	70.6 ± 0.06	65.8 ± 0.04	59.5 ± 0.11	72.1 ± 0.04	67.7 ± 0.12
DSR	73.3	71.6			
SNFS	74.2	72.3		75.2 ± 0.11	72.9 ± 0.06
SNIP	72.0 ± 0.10	67.2 ± 0.12	57.8 ± 0.40		
RigL	74.6 ± 0.06	72.0 ± 0.05	67.5 ± 0.10	75.1 ± 0.05	73.0 ± 0.04
Iterative pruning	75.6	73.9	70.6		
Iterative pruning (ours)	75.3 ± 0.07	73.7 ± 0.14	70.6 ± 0.05		
Powerprop. + Iter. Pruning	**75.7** ± 0.05	**74.4** ± 0.02	**72.1** ± 0.00		
TopKAST† [14]	75.47 ± 0.03	74.65 ± 0.03	**72.73** ± 0.10	75.71 ± 0.06	74.79 ± 0.05
Powerprop. + TopKAST†	**75.75** ± 0.05	**74.74** ± 0.04	**72.89** ± 0.10	**75.84 ± 0.01**	**74.98 ± 0.09**
TopKAST* [14]	76.08 ± 0.02	75.13 ± 0.03	73.19 ± 0.02	76.42 ± 0.03	75.51 ± 0.05
Powerprop. + TopKAST*	**76.24** ± 0.07	**75.23** ± 0.02	**73.25** ± 0.02	**76.76 ± 0.08**	**75.74 ± 0.08**

ImageNet w/ ResNet50: Extended Training & Extreme Sparsity

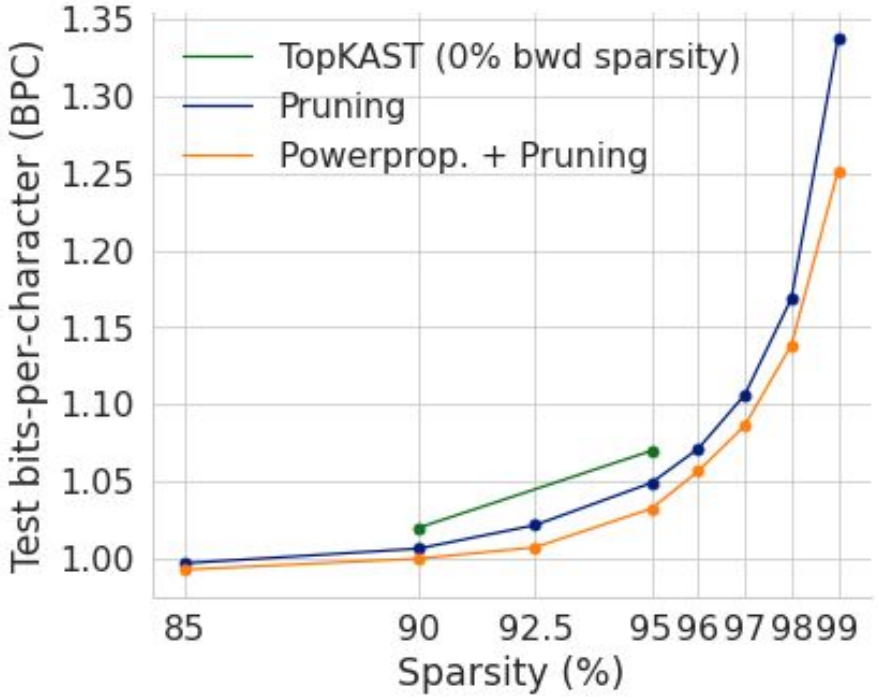
Method	80% Sparsity	90% Sparsity
Iter. Pruning $_{(1.5\times)}$	76.5 [0.84x]	75.2 [0.76x]
RigL $_{(5\times)}$	$76.6 \pm_{0.06}$ [1.14x]	$75.7 \pm_{0.06}$ [0.52x]
ERK		
RigL $_{(5\times)}$	$77.1 \pm_{0.06}$ [2.09x]	$76.4 \pm_{0.05}$ [1.23x]
Powerprop. + TopKAST* $_{(2\times)}$	$77.51 \pm_{0.03}$ [1.21x]	$76.94 \pm_{0.10}$ [0.97x]
Powerprop. + TopKAST* $_{(3\times)}$	$\mathbf{77.64} \pm_{0.05}$ [1.81x]	$\mathbf{77.16} \pm_{0.19}$ [1.46x]

ImageNet w/ ResNet50: Extended Training & Extreme Sparsity

Method	80% Sparsity	90% Sparsity
Iter. Pruning $_{(1.5\times)}$	76.5 [0.84x]	75.2 [0.76x]
RigL $_{(5\times)}$	76.6$\pm_{0.06}$ [1.14x]	75.7$\pm_{0.06}$ [0.52x]
ERK		
RigL $_{(5\times)}$	77.1$\pm_{0.06}$ [2.09x]	76.4$\pm_{0.05}$ [1.23x]
Powerprop. + TopKAST* $_{(2\times)}$	77.51$\pm_{0.03}$ [1.21x]	76.94$\pm_{0.10}$ [0.97x]
Powerprop. + TopKAST* $_{(3\times)}$	**77.64**$\pm_{0.05}$ [1.81x]	**77.16**$\pm_{0.19}$ [1.46x]



enwik8 w/ TransformerXL: Iterative Pruning



DeepMind

Sparsity Q&A

UCL

Discussion & Looking forward

- It can be fruitful to think of existing data and see if we can re-interpret as a distribution of tasks (e.g. RecSys)

Discussion & Looking forward

- It can be fruitful to think of existing data and see if we can re-interpret as a distribution of tasks (e.g. RecSys)
- In practice, task design continues to be a mainly empirical (but very important problem)

Discussion & Looking forward

- It can be fruitful to think of existing data and see if we can re-interpret as a distribution of tasks (e.g. RecSys)
- In practice, task design continues to be a mainly empirical (but very important problem)
- It can be difficult to understand when a test task is outside of the space of tasks seen at training time

Discussion & Looking forward

- It can be fruitful to think of existing data and see if we can re-interpret as a distribution of tasks (e.g. RecSys)
- In practice, task design continues to be a mainly empirical (but very important problem)
- It can be difficult to understand when a test task is outside of the space of tasks seen at training time
- There are specific architectures optimised for e.g. mobile devices (e.g. MobileNet [Howard et al., CVPR 2017])

Discussion & Looking forward

- It can be fruitful to think of existing data and see if we can re-interpret as a distribution of tasks (e.g. RecSys)
- In practice, task design continues to be a mainly empirical (but very important problem)
- It can be difficult to understand when a test task is outside of the space of tasks seen at training time
- There are specific architectures optimised for e.g. mobile devices (e.g. MobileNet [Howard et al., CVPR 2017])
- Very active fields with dozens/hundreds of papers each year

DeepMind

Thank you!

Special thanks to:

Yee Whye Teh