Meta-Learning for Few-Shot Learning

Henry Gouk

Royal Academy of Engineering

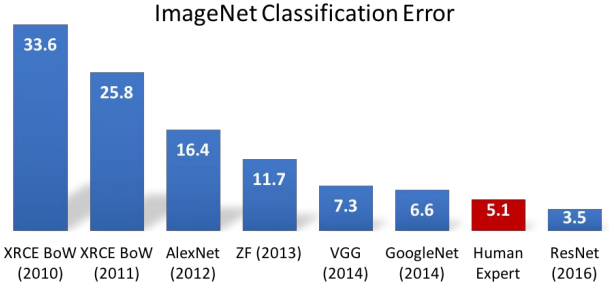
THE UNIVERSITY of EDINBURGH informatics

ECML PKDD 2022

Overview

- Transfer Learning
- Meta-Learning a Shared Feature Extractor
- Gradient-Based Meta-Learning
- Bayesian Meta-Learning

Deep Learning Success



DeepMind

"Human-level control through deep reinforcement learning"

letter

Deep Q-Learning

5

Mechanism for Deep Learning Success?

Caffe

theano

TensorFlow

PyTorch

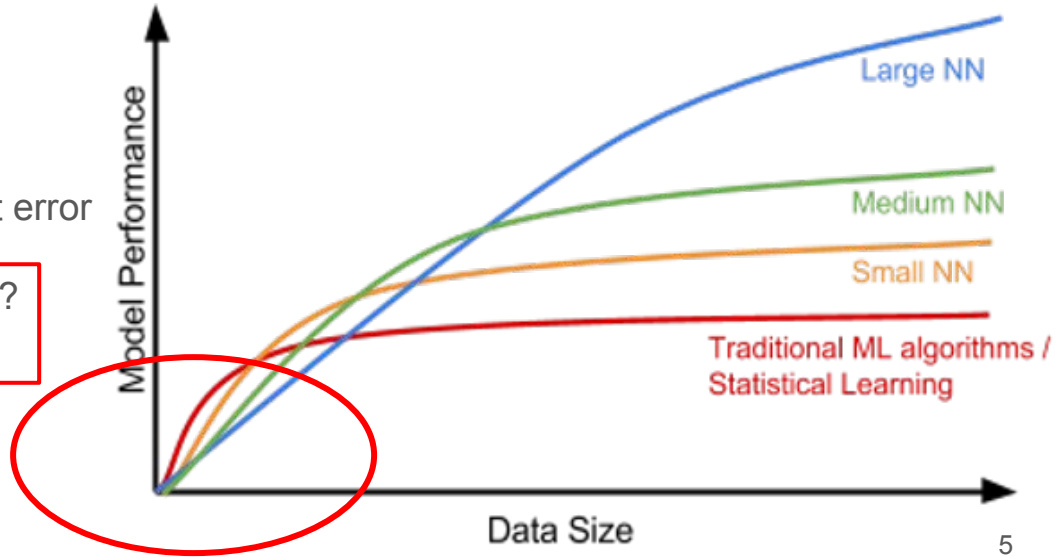
```
from torchvision import models
model = models.resnet18(pretrained=True)
```

```
pred = model(image)

loss = cross_entropy(pred, true_label)

loss.backward()

optimiser.step()
```

Mechanism for Deep Learning Success?

$$\text{Test Error} < \text{Training Error} + \frac{\text{Model Capacity}}{\text{Train Set Size}}$$

- Increase model capacity?
  - better train error, worse test error
- Increase train set size?
  - (maybe) worse train error, better test error
- Increase model capacity and train set size?
  - better train and test error!



How to Avoid Overfitting?

Classic Solutions

- Try several models with different capacities
- Evaluate validation set performance for each
- Pick the model with best validation performance

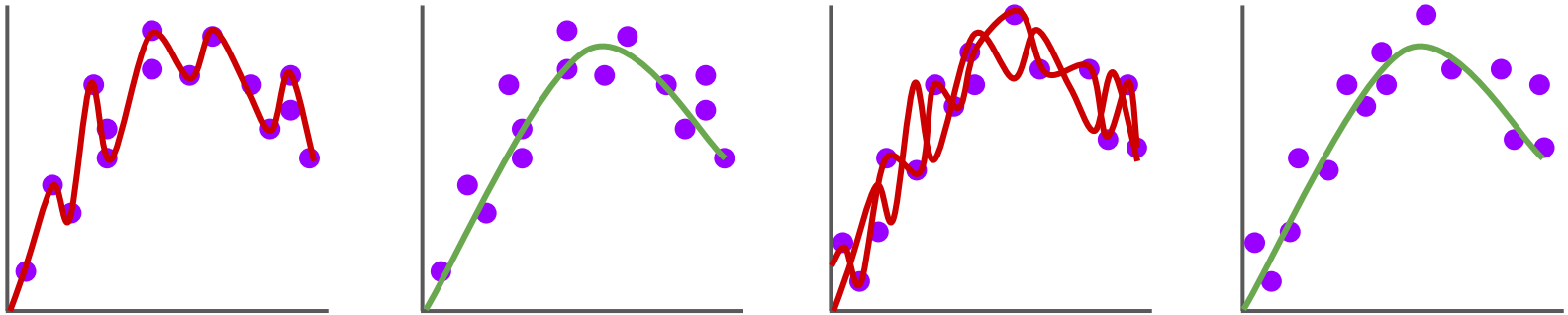
Deep learning Issues

- Too many parameters impact capacity
- In context of data scarcity: would pick a simple model that doesn’t provide deep learning level of performance

How to Avoid Overfitting?

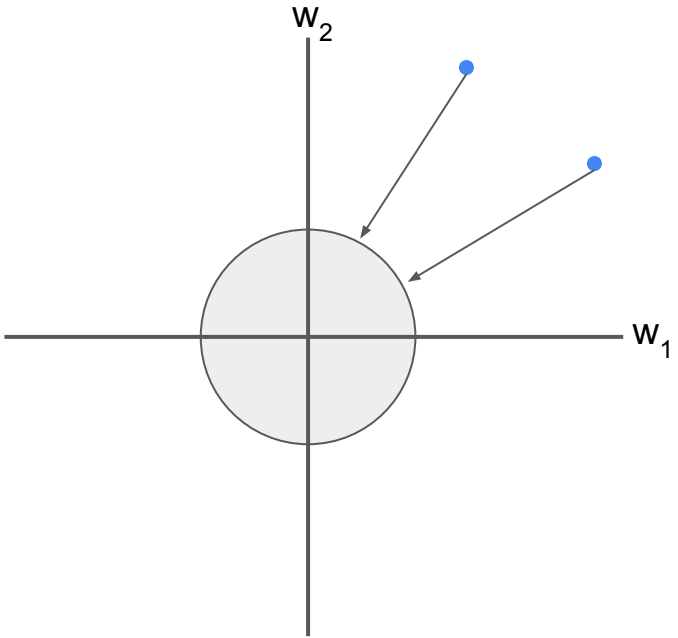
Ask yourself:

Would my pipeline give me a similar model if I collected a new training set?



How can we control modelling capacity?

Controlling Modelling Capacity: Weight Decay



Minimise Training Loss + $||w||^2$

Problem: might just make us underfit

Benefit of weight decay is often quite marginal in neural networks

How can we more intelligently allocate modelling capacity?

Overview

- **Transfer Learning**
- Meta-Learning a Shared Feature Extractor
- Gradient-Based Meta-Learning
- Bayesian Meta-Learning

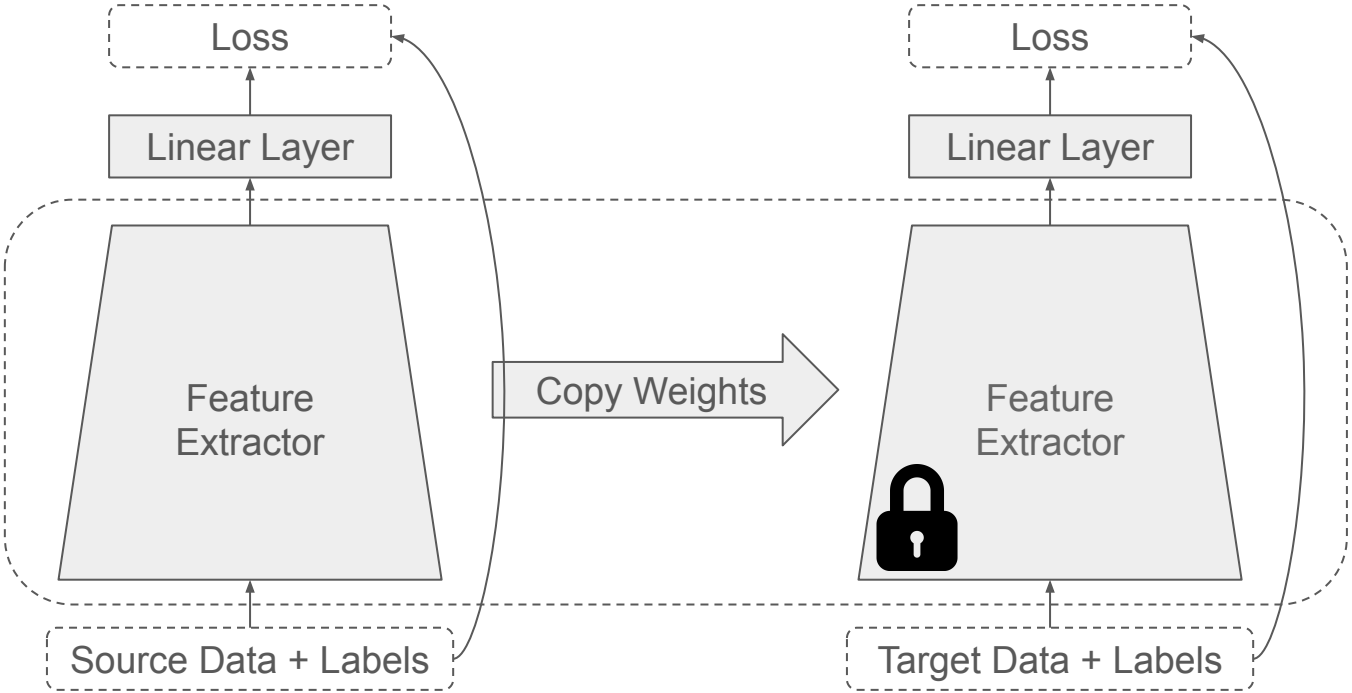
Transfer Learning

"*The application of skills, knowledge, and/or attitudes that were learned in one situation to another learning situation*" (Perkins, 1992)

Note

- Fine-tuning ≠ Transfer Learning
- Fine-tuning ⊂ Transfer Learning

Transfer Learning: Linear Readout



Tuning the model

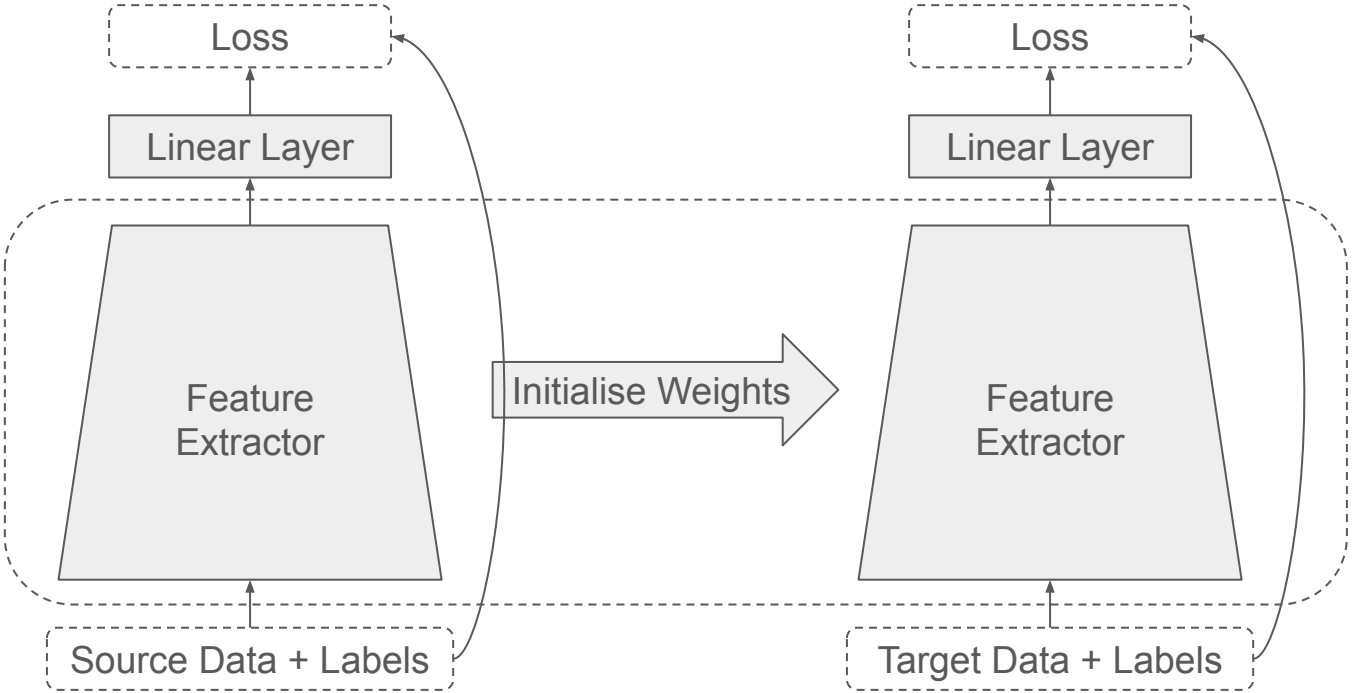
Use standard methods to optimise linear model

Conventional validation/cross validation is typically sufficient

Often work wells for other types of “heads” as well

Bonus: data augmentation probably still beneficial

Transfer Learning: Fine-Tuning



Back to the mess of deep learning design choices:

- Which optimiser to use?
- How to tune optimiser parameters?
- Should we still freeze some layers?
- Should we do early stopping?

Transfer Learning: Fine-Tuning Considerations

How are we allocating modelling capacity?

- Trying to keep weights near informative initialisation
- Contrast with weight decay: keeping weights near uninformative initialisation

Fine-tuning “tricks”:

- Use a small learning rate
- Do early stopping
- Freeze some layers
  - Early layers if task shift
  - Later layers if input distribution shift

Why not add an explicit regulariser like weight decay?

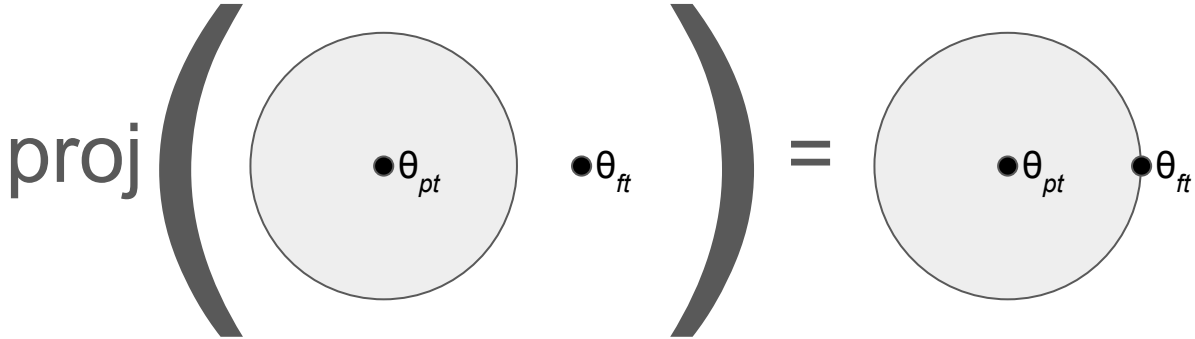
Transfer Learning: Advanced Fine-Tuning

Penalty Term

$$\min_{\theta_{ft}} \mathcal{L}(f_{\theta_{ft}}(\vec{x}), y) + \lambda d(\theta_{ft}, \theta_{pt})$$

"Explicit Inductive Bias for Transfer Learning with Convolutional Networks", [Li, ICML 2018]

Projection Function



"Distance-Based Regularisation of Deep Networks for Fine-Tuning", [Gouk, ICLR 2021]

Could also:

- Penalise deviations in activations
- AutoML for which layers to freeze/unfreeze

Transfer Learning: Advanced Fine-Tuning

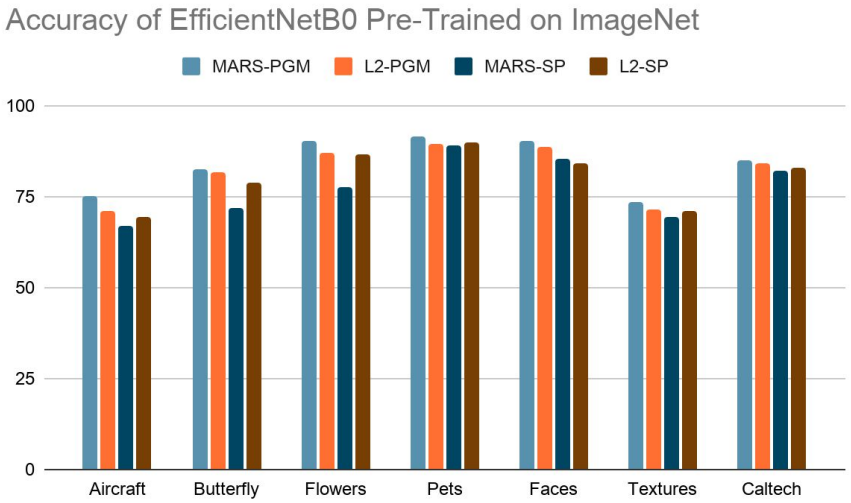
How to measure distance in weight space?

$$d_{mars}(W, V) = \max_i \sum_j |W_{ij} - V_{ij}|$$

Capacity $\propto$ Distance, θ_{pt}

$$d_{frob}(W, V) = \sqrt{\sum_{ij} (W_{ij} - V_{ij})^2}$$

Capacity $\propto$ Distance, θ_{pt}, no. units

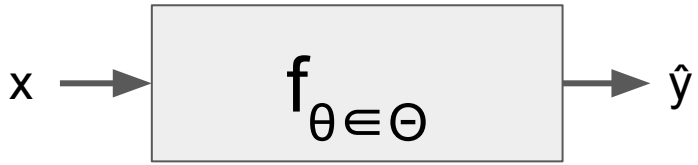


-PGM denotes projection method
-SP denotes penalty method

Overview

- Transfer Learning
- **Meta-Learning a Shared Feature Extractor**
- Gradient-Based Meta-Learning
- Bayesian Meta-Learning

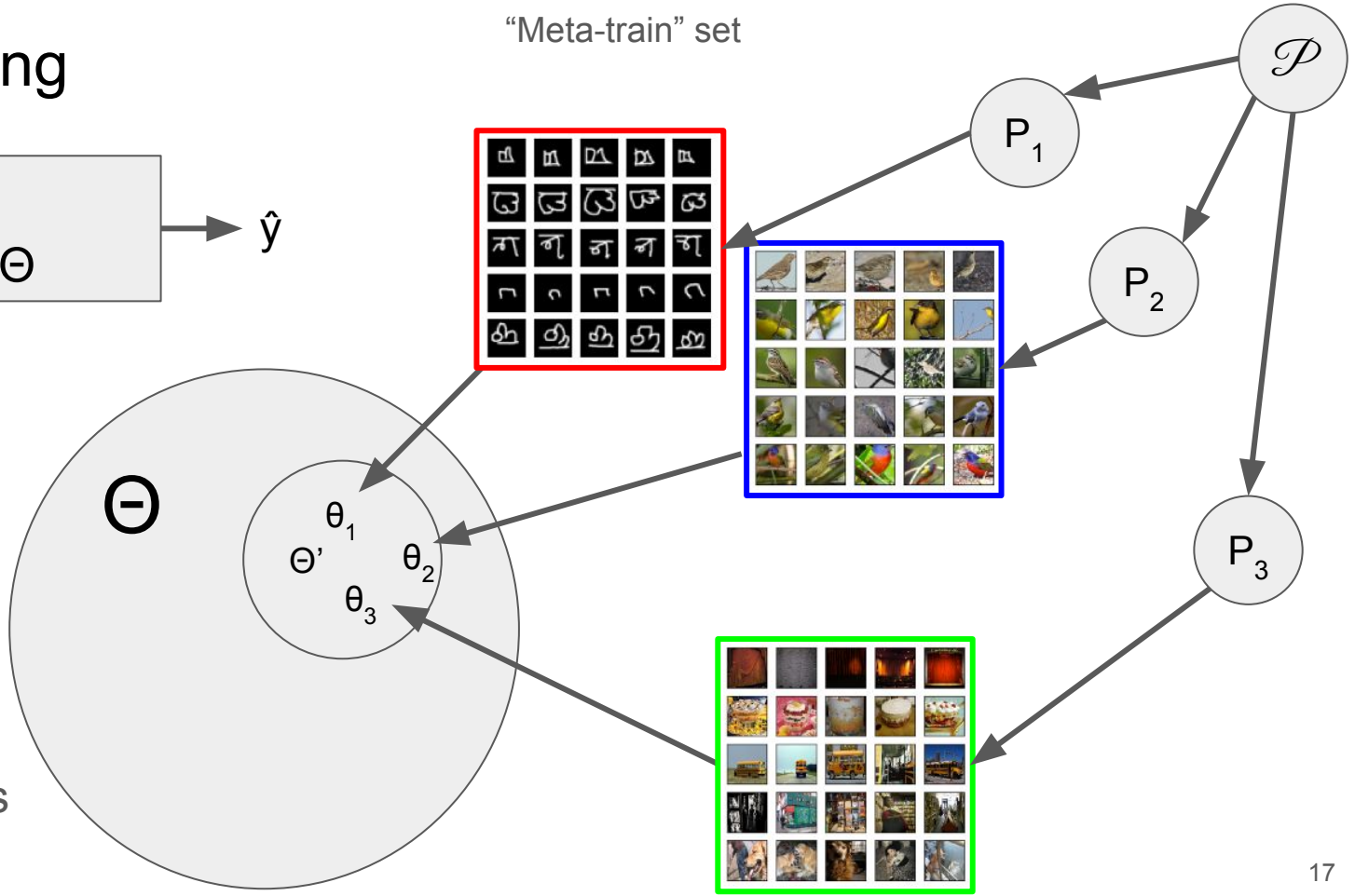
Meta-Learning



Capacity ∝ |Θ|

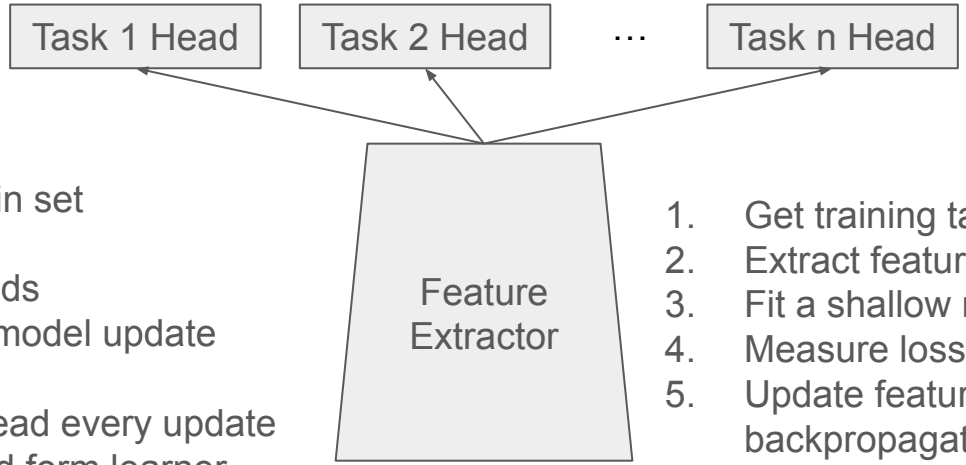
Only use relevant subset of Θ

Meta-learning finds this subset



Metric-Based Meta-Learning

Basic idea: shared feature extractor (=meta-knowledge), different head for each task



Get tasks from meta-train set

Problem: too many heads
Solution: one task per model update

Problem 2: train new head every update
Solution 2: use a closed form learner

1. Get training task from meta-train
2. Extract features for these instances
3. Fit a shallow model on the features
4. Measure loss of classifier
5. Update feature extractor with backpropagation + SGD
6. Go to 1

Nearest centroid classifier: use mean feature vector for each class
"Prototypical Networks for Few-Shot Learning", [Snell, NeurIPS 2017]

Support Vector Machine: using a specialised convex solver
"Meta-Learning with Differentiable Convex Optimization", [Lee, CVPR 2019]

Overview

- Transfer Learning
- Meta-Learning a Shared Feature Extractor
- **Gradient-Based Meta-Learning**
- Bayesian Meta-Learning

Gradient-Based Meta-Learning

Meta-knowledge

$$\phi^* = \underset{\phi}{\operatorname{argmin}} \sum_i \mathcal{M}(D_i^{val}, \theta_i^*(\phi), \phi)$$ Outer problem

$$\text{s.t. } \theta_i^*(\phi) = \underset{\theta}{\operatorname{argmin}}\, \mathcal{L}(D_i^{tr}, \theta, \phi)$$ Inner problem

Intuitively: find meta-knowledge that gives best performance on held out data

How to solve with gradient-based optimisation?

Explicit Gradients
Unroll inner problem optimiser

Implicit Gradients
Agnostic to inner problem solver

Model Agnostic Meta-Learning

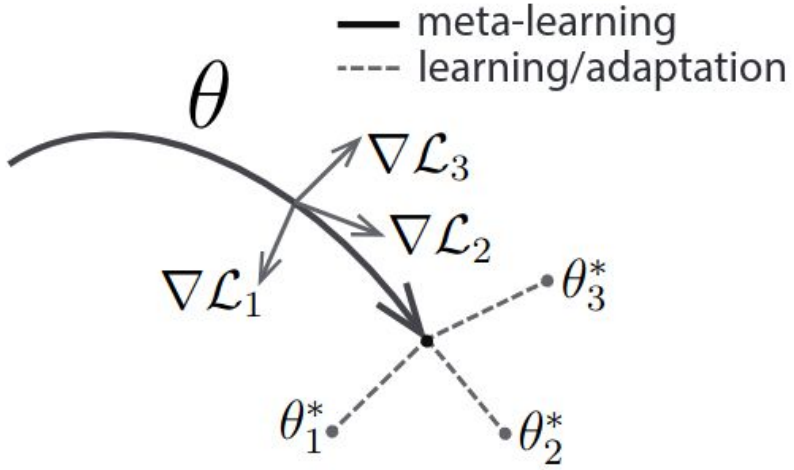
Meta-knowledge: initial weights

Key Idea: approximate inner problem

$$\theta_i^*(\phi) \approx \phi - \alpha \nabla \mathcal{L}(D_i^{tr}, \phi)$$

Substitute into the outer objective

$$\phi^* = \underset{\phi}{\operatorname{argmin}} \sum_i \mathcal{M}(D_i^{val}, \phi - \alpha \nabla \mathcal{L}(D_i^{tr}, \phi))$$



Need to evaluate Hessian

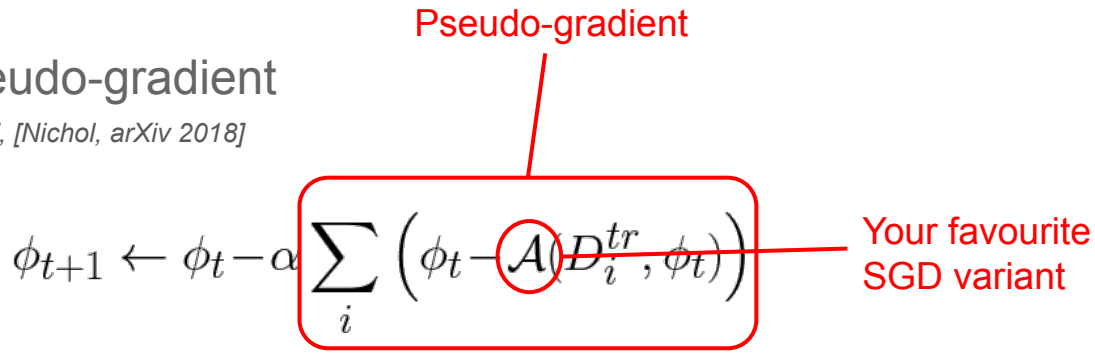
FO-MAML discards Hessian term

"Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks", [Finn, ICML 2017]

Multiple Inner Steps

Reptile: compute a pseudo-gradient

"On First Order Meta-Learning Algorithms", [Nichol, arXiv 2018]



Plug pseudo-gradient into any gradient-based optimiser

Implicit Gradient Methods

- It's possible to differentiate argmin (sometimes)
  - Continuity and convexity requirements not well understood in ML context
  - No need to approximate inner problem!
- Seems to require some expensive Hessian-vector products

Overview

- Transfer Learning
- Meta-Learning a Shared Feature Extractor
- Gradient-Based Meta-Learning
- **Bayesian Meta-Learning**

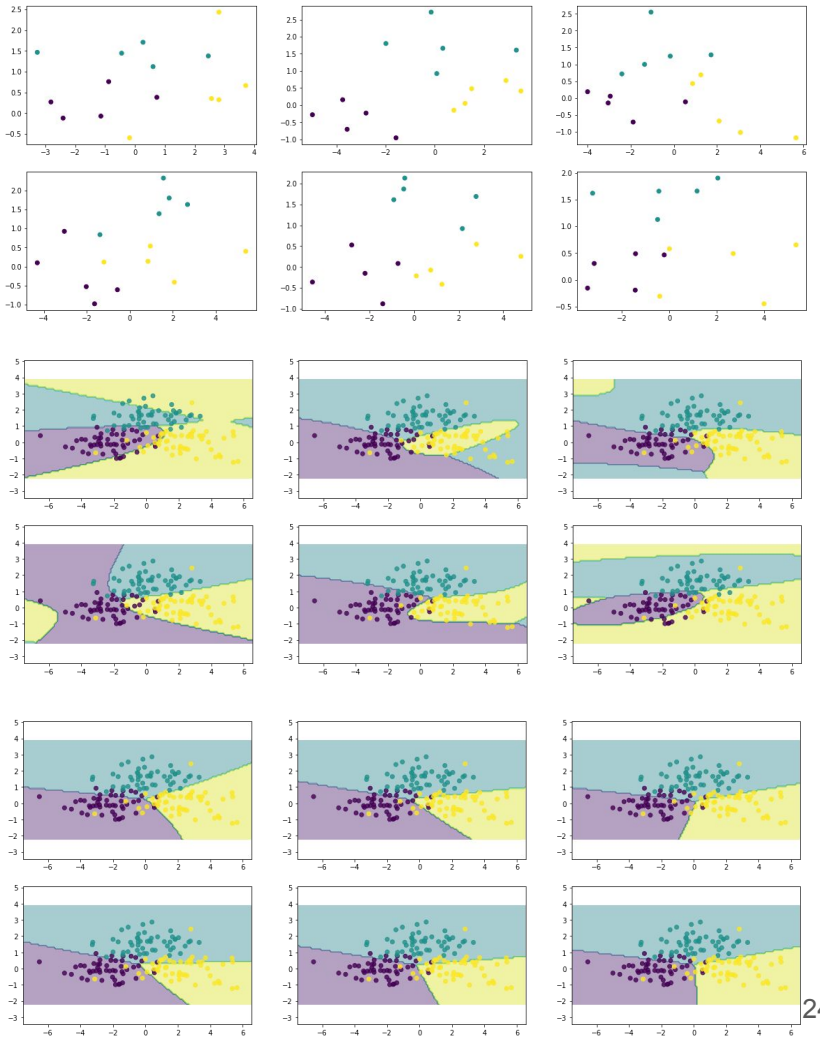
Bayesian Meta-Learning

Motivation: Bayesian probabilistic modelling enables incorporating prior knowledge—can we learn this prior?

Simple probabilistic classifier: fit a Gaussian to each class with maximum likelihood (QDA)

Problem: QDA does not work well with small training datasets—we want different training sets to give similar models!

Solution: Meta-learn a prior on related tasks, compute full posterior over parameters



"Bayesian Meta-Learning for Real-World Few-Shot Recognition", [Zhang, ICCV 2021]

Bayesian Meta-Learning

Model	Backbone	1-shot	5-shot
MAML [37]	Conv-4	$58.90 \pm 1.90\%$	$71.50 \pm 1.00\%$
RELATIONNET [37]	Conv-4	$55.50 \pm 1.00\%$	$69.30 \pm 0.80\%$
PROTONET [37]	Conv-4	$55.50 \pm 0.70\%$	$72.02 \pm 0.60\%$
R2D2 [3]	Conv-4	$62.30 \pm 0.20\%$	$77.40 \pm 0.10\%$
SIMPLESHOT$^+$ [61]	Conv-4	$59.35 \pm 0.89\%$	$74.76 \pm 0.72\%$
METAQDA	Conv-4	$\mathbf{60.52 \pm 0.88\%}$	$\mathbf{77.33 \pm 0.73\%}$
PROTONET [37]	ResNet-12	$72.20 \pm 0.70\%$	$83.50 \pm 0.50\%$
METAOPT [30]	ResNet-12*	$72.00 \pm 0.70\%$	$84.20 \pm 0.50\%$
UNRAVELLING [14]	ResNet-12*	$72.30 \pm 0.40\%$	$86.30 \pm 0.20\%$
BASELINE++ [4, 37]	ResNet-18	$59.67 \pm 0.90\%$	$71.40 \pm 0.69\%$
S2M2 [37]	ResNet-18	$63.66 \pm 0.17\%$	$76.07 \pm 0.19\%$
METAOPTNET [30]	WRN	$72.00 \pm 0.70\%$	$84.20 \pm 0.50\%$
BASELINE++ [37, 4]	WRN	$67.50 \pm 0.64\%$	$80.08 \pm 0.32\%$
S2M2 [37]	WRN	$74.81 \pm 0.19\%$	$87.47 \pm 0.13\%$
METAQDA	WRN	$\mathbf{75.83 \pm 0.88\%}$	$\mathbf{88.79 \pm 0.75\%}$

Model	Backbone	ECE+TS		ECE	
		1-shot	5-shot	1-shot	5-shot
LIN.CLASSIF.	Conv-4	3.56	2.88	8.54	7.48
SIMPLESHOT	Conv-4	3.82	3.35	33.45	45.81
QDA	Conv-4	8.25	4.37	43.54	26.78
MQDA-MAP	Conv-4	2.75	0.89	8.03	5.27
MQDA-FB	Conv-4	**2.33**	**0.45**	**4.32**	**2.92**
S2M2+LIN.CLASSIF	WRN	4.93	2.31	33.23	36.84
SIMPLESHOT	WRN	4.05	1.80	39.56	55.68
QDA	WRN	4.52	1.78	35.95	18.53
MQDA-MAP	WRN	3.94	0.94	31.17	17.37
MQDA-FB	WRN	**2.71**	**0.74**	**30.68**	**15.86**

Improved calibration: expected calibration error measures quality of model uncertainty

Note: can use pre-trained feature extractor and still meta-learn prior

Fin