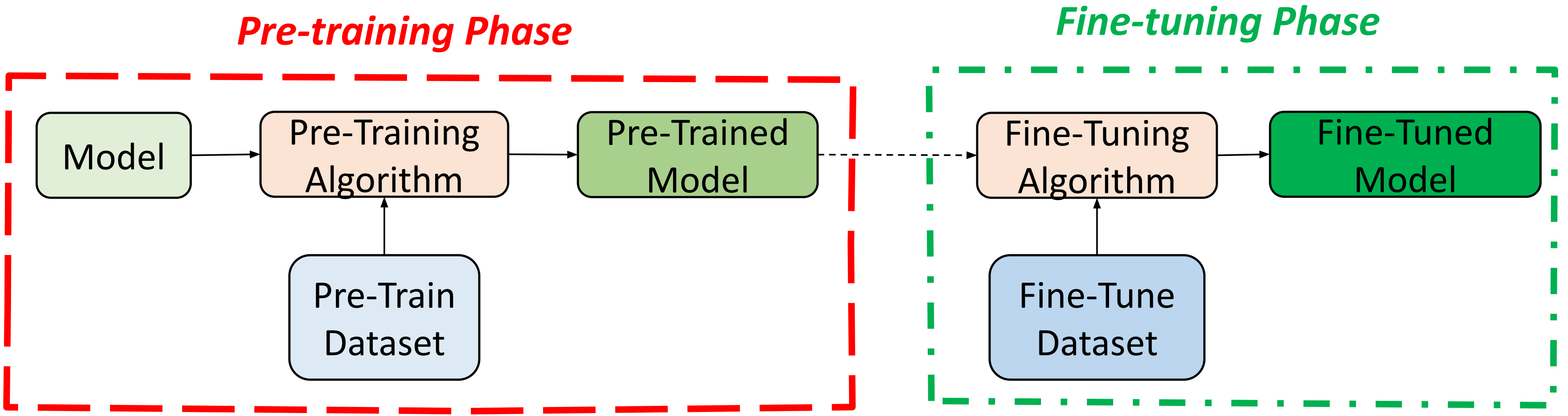
Meta-Learning to Improve Pre-Training

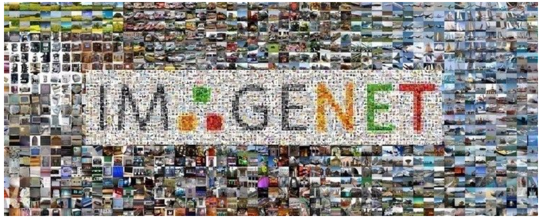
Aniruddh Raghu, Jonathan Lorraine, Simon Kornblith,
Matthew McDermott, David Duvenaud

Pre-training (PT) followed by Fine-tuning (FT):
An important neural network training paradigm...

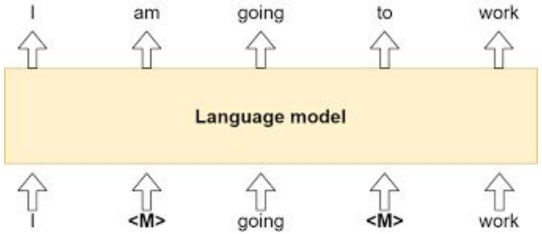


Pre-training (PT) followed by Fine-tuning (FT):
An important neural network training paradigm...

Examples include:

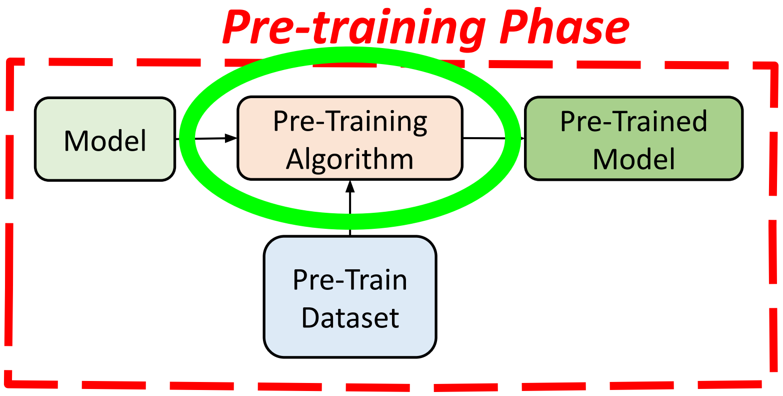


- Supervised PT on ImageNet
- (Masked) language modelling for NLP
- Self-supervised PT: SimCLR, BYOL, …

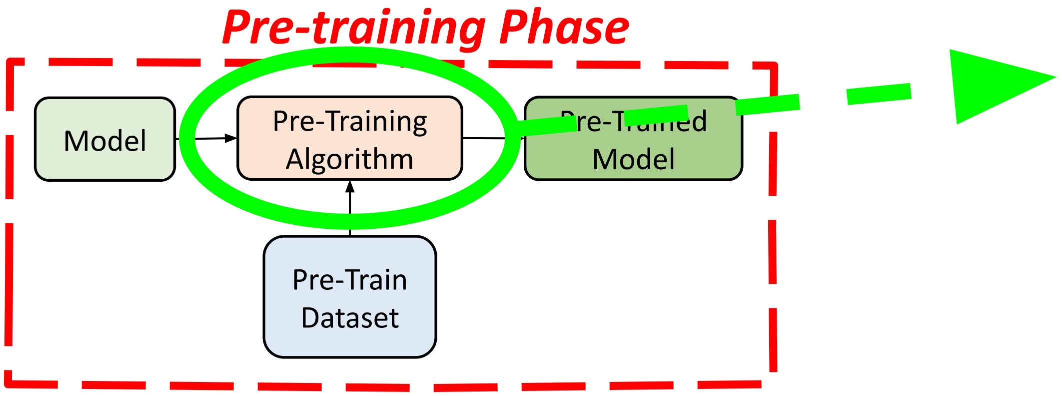


… but introduces many complex design choices (meta-parameters), especially during pre-training (PT)

… but introduces many complex design choices (meta-parameters), especially during pre-training (PT)



… but introduces many complex design choices (meta-parameters), especially during pre-training (PT)



Meta-parameters may include:

- Task weights for multitask PT
- Sampling strategies
- Augmentation strategies for self-supervised PT
- Noise models for PT
- …

Optimizing these meta-parameters is hard!

Optimizing these meta-parameters is hard!

- Random/Grid Search, BayesOpt
  - Slow, may not scale to high-dimensional meta-parameters

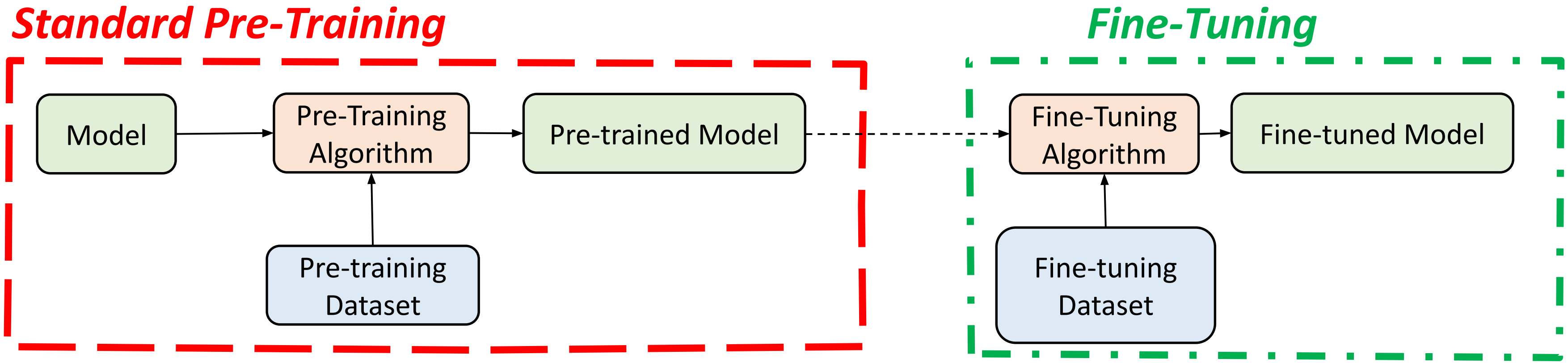
Optimizing these meta-parameters is hard!

- Random/Grid Search, BayesOpt
  - Slow, may not scale to high-dimensional meta-parameters

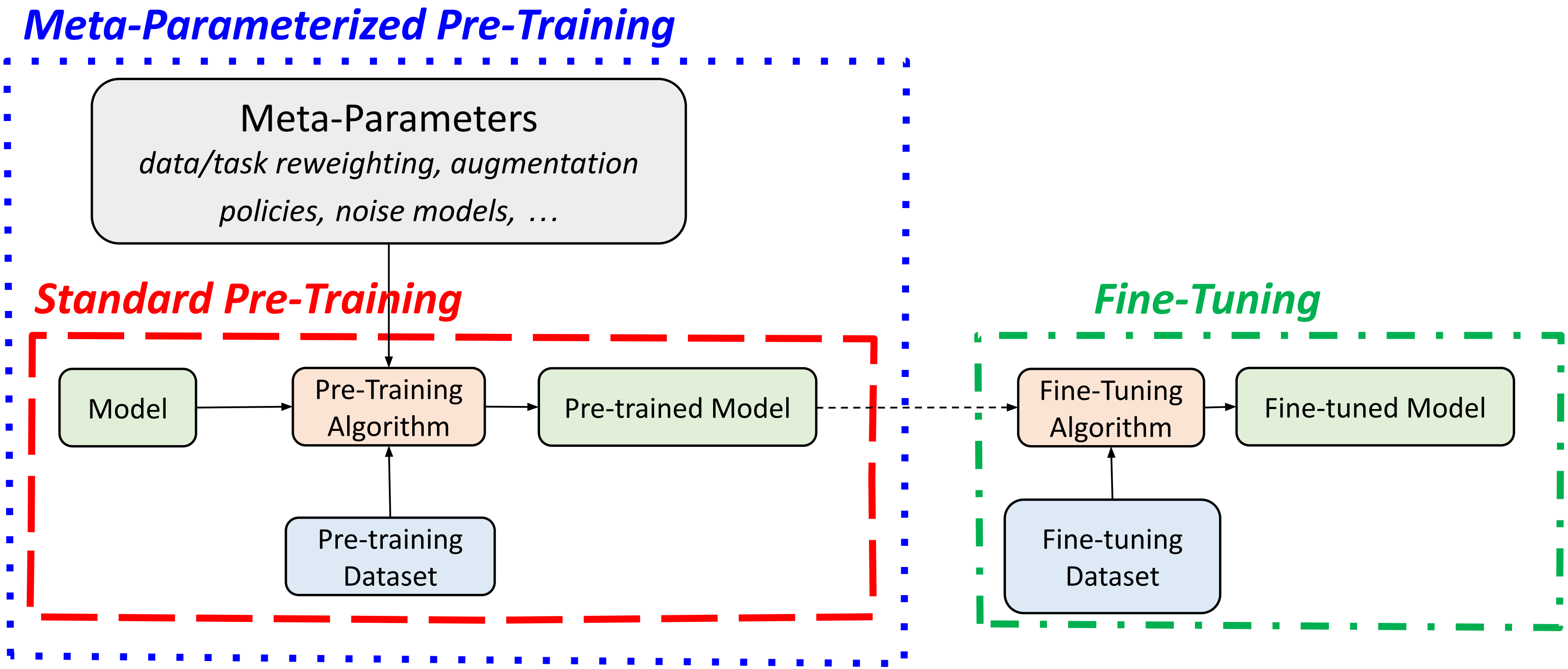
- Gradient based
  - How to efficiently obtain gradients through two optimization stages (PT and FT)?

This work: An efficient and scalable algorithm to optimize these meta-parameters

Standard Pre-Training (PT)



Meta-Parameterized Pre-Training (PT)



More formally...

Define meta-parameters: ϕ

More formally...

Define meta-parameters: ϕ

These parameterize a PT algorithm: $\mathrm{Alg}_{\mathrm{PT}}$

More formally...

Define meta-parameters: ϕ

These parameterize a PT algorithm: $\mathrm{Alg}_{\mathrm{PT}}$

Used when PT a neural network: $f(x;\theta)$

More formally...

Define meta-parameters: ϕ

These parameterize a PT algorithm: $\mathrm{Alg}_{\mathrm{PT}}$

Used when PT a neural network: $f(x;\theta)$

Result: Pre-trained parameters: θ^*_{PT}

More formally...

Following PT, the network is fine-tuned on a FT dataset using $\mathrm{Alg}_{\mathrm{FT}}$

More formally...

Following PT, the network is fine-tuned on a FT dataset using $\mathrm{Alg}_{\mathrm{FT}}$

Result: Fine-tuned parameters: θ^*_{FT}

More formally...

Following PT, the network is fine-tuned on a FT dataset using $\mathrm{Alg}_{\mathrm{FT}}$

Result: Fine-tuned parameters: θ^*_{FT}

Inducing some loss on a FT validation dataset: L_{FT}

More formally...

Following PT, the network is fine-tuned on a FT dataset using $\mathrm{Alg}_{\mathrm{FT}}$

Result: Fine-tuned parameters: θ^*_{FT}

Inducing some loss on a FT validation dataset: L_{FT}

Optimization problem: How do we adjust ϕ to minimize the FT loss L_{FT}?

Gradient-based learning of meta-parameters

Optimization problem: How do we adjust ϕ to minimize the FT loss L_{FT}?

Gradient-based learning of meta-parameters

Optimization problem: How do we adjust ϕ to minimize the FT loss L_{FT}?

We want the gradient: $\dfrac{\partial L_{\mathrm{FT}}}{\partial \phi}$

Gradient-based learning of meta-parameters

Optimization problem: How do we adjust ϕ to minimize the FT loss L_{FT}?

We want the gradient: $\dfrac{\partial L_{\mathrm{FT}}}{\partial \phi}$

By chain rule, the gradient of FT loss wrt ϕ is a product of sets of terms arising from pre-training and fine-tuning.

Gradient-based learning of meta-parameters

Optimization problem: How do we adjust ϕ to minimize the FT loss L_{FT}?

We want the gradient: $\dfrac{\partial L_{\mathrm{FT}}}{\partial \phi}$

By chain rule, the gradient of FT loss wrt ϕ is a product of sets of terms arising from pre-training and fine-tuning.

Informally: $$\frac{\partial L_{\mathrm{FT}}}{\partial \phi} = \frac{\partial L_{\mathrm{FT}}}{\partial \theta^*_{\mathrm{FT}}} \times \frac{\partial \mathrm{Alg}_{\mathrm{FT}}}{\partial \theta^*_{\mathrm{PT}}} \times \frac{\partial \mathrm{Alg}_{\mathrm{PT}}}{\partial \phi}$$

Gradient-based learning of meta-parameters

Optimization problem: How do we adjust ϕ to minimize the FT loss L_{FT}?

We want the gradient: $\dfrac{\partial L_{\mathrm{FT}}}{\partial \phi}$

By chain rule, the gradient of FT loss wrt ϕ is a product of sets of terms arising from pre-training and fine-tuning.

Need to differentiate through two optimization problems!

Informally: $$\frac{\partial L_{\mathrm{FT}}}{\partial \phi} = \frac{\partial L_{\mathrm{FT}}}{\partial \theta^*_{\mathrm{FT}}} \times \boxed{\frac{\partial \mathrm{Alg}_{\mathrm{FT}}}{\partial \theta^*_{\mathrm{PT}}} \times \frac{\partial \mathrm{Alg}_{\mathrm{PT}}}{\partial \phi}}$$

Gradient-based learning of meta-parameters

Informally:

$$\frac{\partial L_{\mathrm{FT}}}{\partial \phi} = \frac{\partial L_{\mathrm{FT}}}{\partial \theta^*_{\mathrm{FT}}} \times \frac{\partial \mathrm{Alg}_{\mathrm{FT}}}{\partial \theta^*_{\mathrm{PT}}} \times \frac{\partial \mathrm{Alg}_{\mathrm{PT}}}{\partial \phi}$$

Key insight: Approximate gradient by combining backpropagation through training and implicit differentiation

Gradient-based learning of meta-parameters

Informally:

$$\frac{\partial L_{\mathrm{FT}}}{\partial \phi} = \boxed{\frac{\partial L_{\mathrm{FT}}}{\partial \theta^*_{\mathrm{FT}}}} \times \frac{\partial \mathrm{Alg}_{\mathrm{FT}}}{\partial \theta^*_{\mathrm{PT}}} \times \frac{\partial \mathrm{Alg}_{\mathrm{PT}}}{\partial \phi}$$

Easy to compute:
Use direct backprop

Key insight: Approximate gradient by combining backpropagation through training and implicit differentiation

Gradient-based learning of meta-parameters

Informally:

$$\frac{\partial L_{\mathrm{FT}}}{\partial \phi} = \boxed{\frac{\partial L_{\mathrm{FT}}}{\partial \theta^*_{\mathrm{FT}}}} \times \boxed{\frac{\partial \mathrm{Alg}_{\mathrm{FT}}}{\partial \theta^*_{\mathrm{PT}}}} \times \frac{\partial \mathrm{Alg}_{\mathrm{PT}}}{\partial \phi}$$

Easy to compute:
Use direct backprop

Harder to compute: Use
Truncated Backprop
through training

Key insight: Approximate gradient by combining backpropagation through training and implicit differentiation

Gradient-based learning of meta-parameters

Informally:

$$\frac{\partial L_{\mathrm{FT}}}{\partial \phi} = \boxed{\frac{\partial L_{\mathrm{FT}}}{\partial \theta^*_{\mathrm{FT}}}} \times \boxed{\frac{\partial \mathrm{Alg}_{\mathrm{FT}}}{\partial \theta^*_{\mathrm{PT}}}} \times \boxed{\frac{\partial \mathrm{Alg}_{\mathrm{PT}}}{\partial \phi}}$$

Easy to compute: Use direct backprop

Harder to compute: Use Truncated Backprop through training

Harder to compute: Use Implicit differentiation

Key insight: Approximate gradient by combining backpropagation through training and implicit differentiation

$$\frac{\partial L_{\mathrm{FT}}}{\partial \phi} = \underbrace{\overbrace{\boxed{\frac{\partial L_{\mathrm{FT}}}{\partial \theta^{*}_{\mathrm{FT}}}}}^{\text{FT Loss Gradient}}}_{\substack{\text{Easy to compute:} \\ \text{Use direct backprop}}} \times \underbrace{\overbrace{\boxed{\frac{\partial \mathrm{Alg}_{\mathrm{FT}}}{\partial \theta^{*}_{\mathrm{PT}}}}}^{\text{FT Best Response Jacobian}}}_{\substack{\text{Harder to compute:} \\ \text{Use (Truncated) Backprop} \\ \text{through training}}} \times \underbrace{\overbrace{\boxed{\frac{\partial \mathrm{Alg}_{\mathrm{PT}}}{\partial \phi}}}^{\text{PT Best Response Jacobian}}}_{\substack{\text{Harder to compute:} \\ \text{Use Implicit differentiation}}}$$

An Online Meta-Learning Algorithm

In practice, learn meta-parameters and network parameters online for efficiency

An Online Meta-Learning Algorithm

In practice, learn meta-parameters and network parameters online for efficiency

For T meta-optimization steps:

An Online Meta-Learning Algorithm

In practice, learn meta-parameters and network parameters online for efficiency

For *T* meta-optimization steps:

- Do *P* steps of pre- training, obtain: θ^*_{PT}

An Online Meta-Learning Algorithm

In practice, learn meta-parameters and network parameters online for efficiency

For *T* meta-optimization steps:

- Do *P* steps of pre- training, obtain: θ^*_{PT}
- Do *K* steps of fine-tuning: θ^*_{FT}

An Online Meta-Learning Algorithm

In practice, learn meta-parameters and network parameters online for efficiency

For *T* meta-optimization steps:

- Do *P* steps of pre- training, obtain: θ^*_{PT}
- Do *K* steps of fine-tuning: θ^*_{FT}
- Compute FT validation set loss: L_{FT}

An Online Meta-Learning Algorithm

In practice, learn meta-parameters and network parameters online for efficiency

For *T* meta-optimization steps:

- Do *P* steps of pre- training, obtain: θ^*_{PT}
- Do *K* steps of fine-tuning: θ^*_{FT}
- Compute FT validation set loss: L_{FT}
- Compute meta-gradients using previous approximation: $\frac{\partial L_{\mathrm{FT}}}{\partial \phi}$

An Online Meta-Learning Algorithm

In practice, learn meta-parameters and network parameters online for efficiency

For *T* meta-optimization steps:

- Do *P* steps of pre- training, obtain: θ^*_{PT}
- Do *K* steps of fine-tuning: θ^*_{FT}
- Compute FT validation set loss: L_{FT}
- Compute meta-gradients using previous approximation: $\frac{\partial L_{\mathrm{FT}}}{\partial \phi}$
- Perform meta-parameter update: $\phi \leftarrow \phi - \eta \frac{\partial L_{\mathrm{FT}}}{\partial \phi}$

Experimental evaluation

Synthetic Experiments:
Can our algorithm learn optimal meta-parameters?

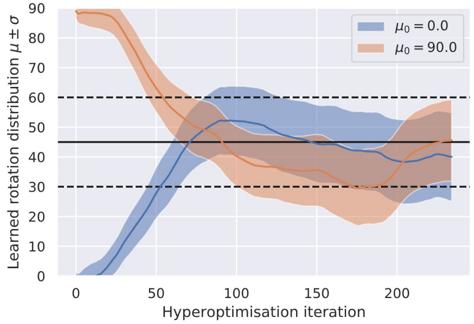
Synthetic Experiments:
Can our algorithm learn optimal meta-parameters?

Yes! In two synthetic MNIST domains, the algorithm finds optimal meta-parameters.

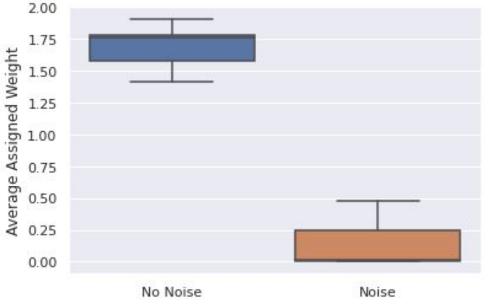
Synthetic Experiments:
Can our algorithm learn optimal meta-parameters?

Yes! In two synthetic MNIST domains, the algorithm finds optimal meta-parameters.

- PT augmentations:
  learns the optimal augmentation distribution



- PT example importance weighting with noisy labels:
  downweights noisy label examples



Real-world experimental evaluation

Study two settings:

1) Optimizing task weights for multi-task PT with GNNs

2) Optimizing data augmentation pipeline for self-supervised learning with electrocardiogram signals

Focus on 1) in this talk

Multitask Pre-training with GNNs

- **Task:** Given input Protein-Protein Interaction graph, predict the presence of several biological functions [1]

[1] Strategies for Pre-training Graph Neural Networks, Hu et al., ICLR 2020

Multitask Pre-training with GNNs

- **Task:** Given input Protein-Protein Interaction graph, predict the presence of several biological functions [1]

- **PT:** Input graphs (*x*) together with 5000 binary attributes (*y*)
  FT: Input graphs (*x*) together with 40 (distinct) binary attributes (*y*)

[1] Strategies for Pre-training Graph Neural Networks, Hu et al., ICLR 2020

Multitask Pre-training with GNNs

- **Task:** Given input Protein-Protein Interaction graph, predict the presence of several biological functions [1]

- **PT:** Input graphs (*x*) together with 5000 binary attributes (*y*)
  FT: Input graphs (*x*) together with 40 (distinct) binary attributes (*y*)

- **Meta-parameters:** 5000-dimensional vector representing a weight for each task in PT, used to weight PT loss.

[1] Strategies for Pre-training Graph Neural Networks, Hu et al., ICLR 2020

Multitask Pre-training with GNNs

- **Baselines:**
  - No PT
  - Supervised PT: weights set to 1
  - CoTrain: PT jointly on all labels
  - CoTrain + PCGrad [1]: Like above, but apply PCGrad to reduce conflicting gradient updates among tasks

[1] Gradient Surgery for Multi-Task Learning, Yu et al., NeurIPS 2020

Multitask Pre-training with GNNs

- **Results:**

Method	Average AUC across tasks
No PT	66.6 ± 0.7
Graph Supervised PT	74.7 ± 0.1
CoTrain	70.2 ± 0.3
CoTrain + PCGrad	69.4 ± 0.2
Meta-Parameterized PT	$\mathbf{78.6 \pm 0.1}$

Meta-Parameterized PT improves significantly on baselines!

Multitask Pre-training with GNNs

- **Results:**

Method	Average AUC across tasks
No PT	66.6 ± 0.7
Graph Supervised PT	74.7 ± 0.1
CoTrain	70.2 ± 0.3
CoTrain + PCGrad	69.4 ± 0.2
Meta-Parameterized PT	$\mathbf{78.6} \pm \mathbf{0.1}$

Meta-Parameterized PT improves significantly on baselines!

- Additional experimental settings and baselines in paper
  - Testing generalization to new tasks
  - Further studies on sample efficiency
  - Other meta-parameter learning baselines

Conclusion

- **Pre-training followed by Fine-tuning:**

Conclusion

- **Pre-training followed by Fine-tuning:**

    + A powerful and successful neural network training paradigm

Conclusion

- **Pre-training followed by Fine-tuning:**

  + A powerful and successful neural network training paradigm

  - Comes with many design choices (meta-parameters) during pre-training

Conclusion

- **Pre-training followed by Fine-tuning:**

  + A powerful and successful neural network training paradigm

  - Comes with many design choices (meta-parameters) during pre-training

- **This work:**
  - Formalize the problem of optimizing these meta-parameters
  - Propose a scalable meta-learning algorithm to do so

Conclusion

- **Pre-training followed by Fine-tuning:**

  \+ A powerful and successful neural network training paradigm

  \- Comes with many design choices (meta-parameters) during pre-training

- **This work:**
  - Formalize the problem of optimizing these meta-parameters
  - Propose a scalable meta-learning algorithm to do so

- **Experimental evaluation**: our algorithm can effectively optimize these meta-parameters, and improves performance in two-real world domains