Meta-Learning Symmetries By Reparameterization

Paper by: Allan Zhou, Tom Knowles, Chelsea Finn

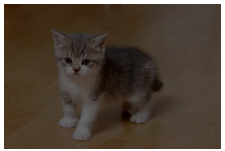
Slides by: Paul Vicol

Traditional Paths to Equivariant Models

- Equivariance to certain transformations can be *useful for generalization*
  - Note: *equivariance includes invariance as a special case*


Rotation


Reflection

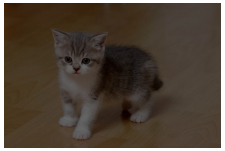

Brightness


Hue

Traditional Paths to Equivariant Models

- Equivariance to certain transformations can be *useful for generalization*
  - Note: *equivariance includes invariance as a special case*


Reflection


Brightness

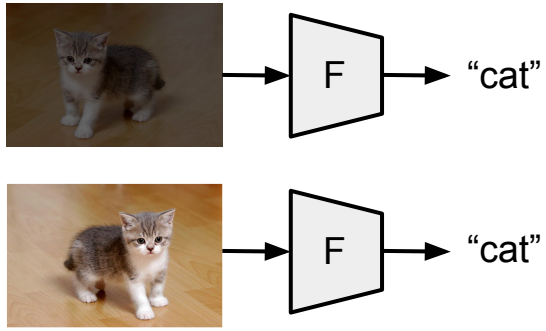

Hue

Manual Architecture Design

- CNNs are *designed to be equivariant to translation*
- Work from Max Welling's group extends this to other symmetry groups (Spherical CNNs, etc.)

Data Augmentation

- "Train in" desired invariances



Idea: Meta-Learn Equivariances

- The meta-learning approach
  - *Learn equivariances from data* without needing to design custom task-specific architectures
  - Learn to *exploit symmetries shared by a collection of tasks*, using gradient-based meta-learning to *learn parameter sharing patterns* (which enforce equivariance) separately from actual parameter values

Definitions

- *Equivariance*
  - A function is equivariant to a transformation if *transforming the input to the function is equivalent to transforming the output*
  - Consider a network layer $\phi : \mathbb{R}^n \rightarrow \mathbb{R}^m$
  - Assume we have two representations of a group G where $\pi_1(g)$ transforms the input vectors and $\pi_2(g)$ transforms the output vectors. The layer ϕ is G-equivariant w.r.t these transformations if:

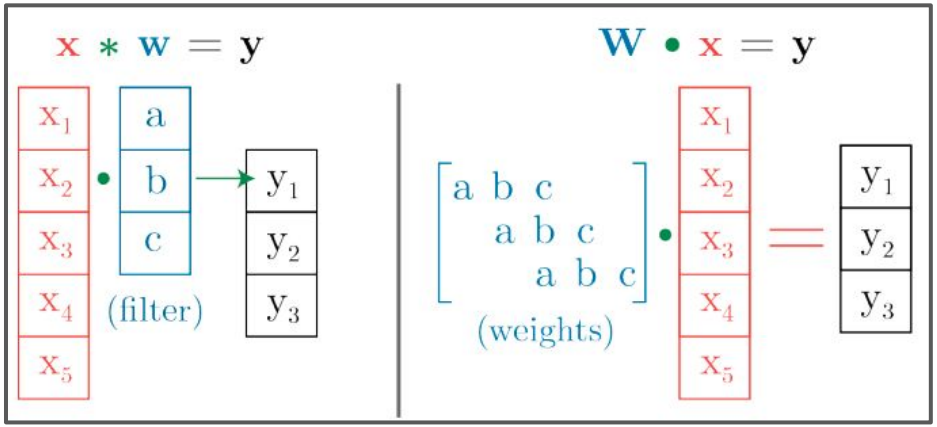
$$\phi(\pi_1(g)\mathbf{v}) = \pi_2(g)\phi(\mathbf{v}) \qquad \text{for all} \quad g \in G, v \in \mathbb{R}^n$$

- *Invariance is a special case of equivariance*
  - If $\pi_2 = \mathrm{id} = I$ then we have:

$$\underbrace{\phi(\pi_1(g)\mathbf{v})}_{\text{Layer output on the transformed input}} = \pi_2(g)\phi(\mathbf{v}) = \underbrace{\phi(\mathbf{v})}_{\text{Layer output on the original input}}$$

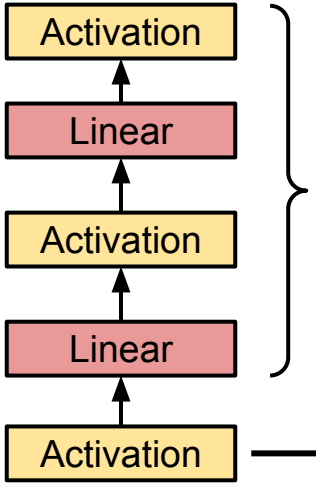
Parameter Sharing Patterns → Equivariance

- A convolutional layer can be viewed as a linear layer with a *specific parameter-sharing structure*:



- Equivariant layers for other transformations, like rotation and reflection, correspond to *different parameter sharing patterns*

Parameter Sharing Patterns → Equivariance



Function *composition preserves equivariance*, so if we achieve equivariance in each individual layer, then the whole network will be equivariant

Elementwise nonlinearities (ReLU, sigmoid, tanh) are already equivariant to any permutation of the input and output indices, which includes translation, reflection, and rotation

Reparameterization

- They propose a *representation to encode possible equivariances*
- A fully-connected layer $\phi : \mathbb{R}^n \to \mathbb{R}^m$ with weight matrix $W \in \mathbb{R}^{m \times n}$ is given by $\phi(x) = Wx$
- They factorize W as the product of a symmetry matrix $\mathbf{U}$ and a vector $\mathbf{V}$ of filter params

$$\text{vec}(W) = Uv, \quad v \in \mathbb{R}^k, U \in \mathbb{R}^{mn \times k}$$

Encodes the pattern by which the weights W will "share" the filter parameters v

Shared filter parameters

- This separates the problem of learning the sharing pattern from the problem of learning the filter params
- Then $\text{vec}(W) \in \mathbb{R}^{mn}$ is reshaped into the weight matrix $W \in \mathbb{R}^{m \times n}$

Meta-Learning Equivariances

Meta-Training Time

- For a task $\mathcal{T}_i \sim p(\mathcal{T})$ the *inner loop* fixes $\mathbf{U}$ and only updates $\mathbf{V}$ using the task *training data*:

$$\mathbf{v}' \leftarrow \mathbf{v} - \alpha \nabla_{\mathbf{v}} \mathcal{L}(\mathbf{U}, \mathbf{v}, \mathcal{D}_i^{\text{train}})$$

- The *outer loop* updates $\mathbf{U}$ by computing the loss on the task's validation data using $\mathbf{V}'$ and the task's validation data:

$$\mathbf{U} \leftarrow \mathbf{U} - \eta \nabla_{\mathbf{U}} \mathcal{L}(\mathbf{U}, \mathbf{v}', \mathcal{D}_i^{\text{val}})$$

- (We can also still meta-learn the initialization of the filter parameters $\mathbf{V}$ as in MAML)

Meta-Learning Equivariances

Meta-Training Time

- For a task $\mathcal{T}_i \sim p(\mathcal{T})$ the *inner loop* fixes $\mathbf{U}$ and only updates $\mathbf{V}$ using the task *training data*:

$$\mathbf{v}' \leftarrow \mathbf{v} - \alpha \nabla_{\mathbf{v}} \mathcal{L}(\mathbf{U}, \mathbf{v}, \mathcal{D}_i^{\text{train}})$$

- The *outer loop* updates $\mathbf{U}$ by computing the loss on the task's validation data using $\mathbf{V}'$ and the task's validation data:

$$\mathbf{U} \leftarrow \mathbf{U} - \eta \nabla_{\mathbf{U}} \mathcal{L}(\mathbf{U}, \mathbf{v}', \mathcal{D}_i^{\text{val}})$$

- (We can also still meta-learn the initialization of the filter parameters $\mathbf{V}$ as in MAML)

Meta-Test Time

- For a test task, $\mathbf{U}$ is frozen and only the filter params $\mathbf{V}$ are updated
  - This enforces meta-learned parameter sharing and improves generalization by reducing the number of task-specific inner-loop parameters

A Hybrid Approach

- Some equivariances are *useful but expensive to meta-learn, like standard convolutions*
- But there may still be *symmetries in the data that we wish to discover automatically*

- **Hybrid approach:** some baked-in invariances + some learned ones
  - Can directly reparameterize a standard convolution layer by reshaping vec(W) *into a convolution filter bank rather than a weight matrix*
  - Bakes in translation equivariance, and allows for learning rotation equivariance from data

+

Learned equivariances from data

=

Experiments

Synthetic Tasks

- Synthetic meta-learning problems *designed to have certain symmetries* including translation, rotation, reflection, etc.
- They use meta-learning w/ general architectures not designed around these symmetries to see whether each method can automatically meta-learn the equivariances

Learning Invariances from Data Augmentation

- Using MSR to encode invariances into the parameter sharing structure, to transfer to the test-set tasks more easily
- Evaluated on standard few-shot learning tasks: Omniglot, MiniImageNet

Learning Invariances from Augmented Data

- Data augmentation can yield invariant models
- In meta-learning settings, we *need augmented data for each task*
  - May not be able to augment data in the *training set of a meta-test task (e.g., real-world robot setting)*
- **Idea:** use MSR to learn invariances from data augmentation at training time
  - Encode these invariances into the network itself through $\mathbf{U}$
  - *Preserve learned invariances on new meta-test tasks* without needing additional data augmentation

Algorithm 2: Augmentation Meta-Training

input: $\{\mathcal{T}_i\}_{i=1}^{N}$: Meta-training tasks
input: META-TRAIN: Any meta-learner
input: AUGMENT: Data augmenter
forall $\mathcal{T}_i \in \{\mathcal{T}_i\}_{i=1}^{N}$ **do**
  $\{\mathcal{D}_i^{tr}, \mathcal{D}_i^{val}\} \leftarrow \mathcal{T}_i$; `// task data split`
  $\hat{\mathcal{D}}_i^{val} \leftarrow \text{AUGMENT}(\mathcal{D}^{val})$;
  $\hat{\mathcal{T}}_i \leftarrow \{\mathcal{D}^{tr}, \hat{\mathcal{D}}_i^{val}\}$
$\text{META-TRAIN}\left(\{\hat{\mathcal{T}}_i\}_{i=1}^{N}\right)$

Learning Invariances from Augmented Data

Method	Aug-Omniglot 5 way 1-shot	Aug-Omniglot 5 way 5-shot	Aug-Omniglot 20 way 1-shot	Aug-Omniglot 20 way 5-shot	Aug-MiniImagenet 5 way 1-shot	Aug-MiniImagenet 5 way 5-shot
MAML	87.3 ± 0.5	93.6 ± 0.3	67.0 ± 0.4	79.9 ± 0.3	42.5 ± 1.1	61.5 ± 1.0
MAML (Big)	89.3 ± 0.4	94.8 ± 0.3	69.6 ± 0.4	83.2 ± 0.3	37.2 ± 1.1	63.2 ± 1.0
ANIL	86.4 ± 0.5	93.2 ± 0.3	67.5 ± 3.5	79.8 ± 0.3	43.0 ± 1.1	62.3 ± 1.0
ProtoNets	92.9 ± 0.4	$\mathbf{97.4 \pm 0.2}$	$\mathbf{85.1 \pm 0.3}$	$\mathbf{94.3 \pm 0.2}$	34.6 ± 0.5	54.5 ± 0.6
MSR (Ours)	$\mathbf{95.3 \pm 0.3}$	$\mathbf{97.7 \pm 0.2}$	84.3 ± 0.2	92.6 ± 0.2	$\mathbf{45.5 \pm 1.1}$	$\mathbf{65.2 \pm 1.0}$

Table 3: Meta-test accuracies on Aug-Omniglot and Aug-MiniImagenet few-shot classification, which requires generalizing to augmented validation data from un-augmented training data. MSR performs comparably to or better than other methods under this augmented regime. Results shown with 95% CIs.

- Data augmentation consists of a combination of random rotations, flips, and resizes
- All methods (MAML, ANIL, PN, MSR) *can learn invariant features*
  - MSR seems to do better “because it enforces learned invariance through its symmetry matrices”

Summary

- MSR is a method for *meta-learning equivariance-inducing parameter sharing patterns* in each layer of a network
- \+ Sharing patterns reduce the number of task-specific parameters that must be learned in the inner loop, and improve generalization
- \+ Can meta-learn invariances using data augmentation, *into the network structure*
- \- Does not make use of parameter sharing to reduce actual computation (still needs large, probably sparse, matrices)
- \- Requires a distribution of tasks with shared symmetries
    - Unclear how to discover symmetries quickly for a single task

Q/A