# Working with Sets

## Incorporating constraints in learning machines through model architecture

Jonathon Hare

University of Southampton

# Learning to Pool

Yan Zhang and Jonathon Hare and Adam Prügel-Bennett (2020) FSPool: Learning Set Representations with Featurewise Sort Pooling. International Conference on Learning Representations.

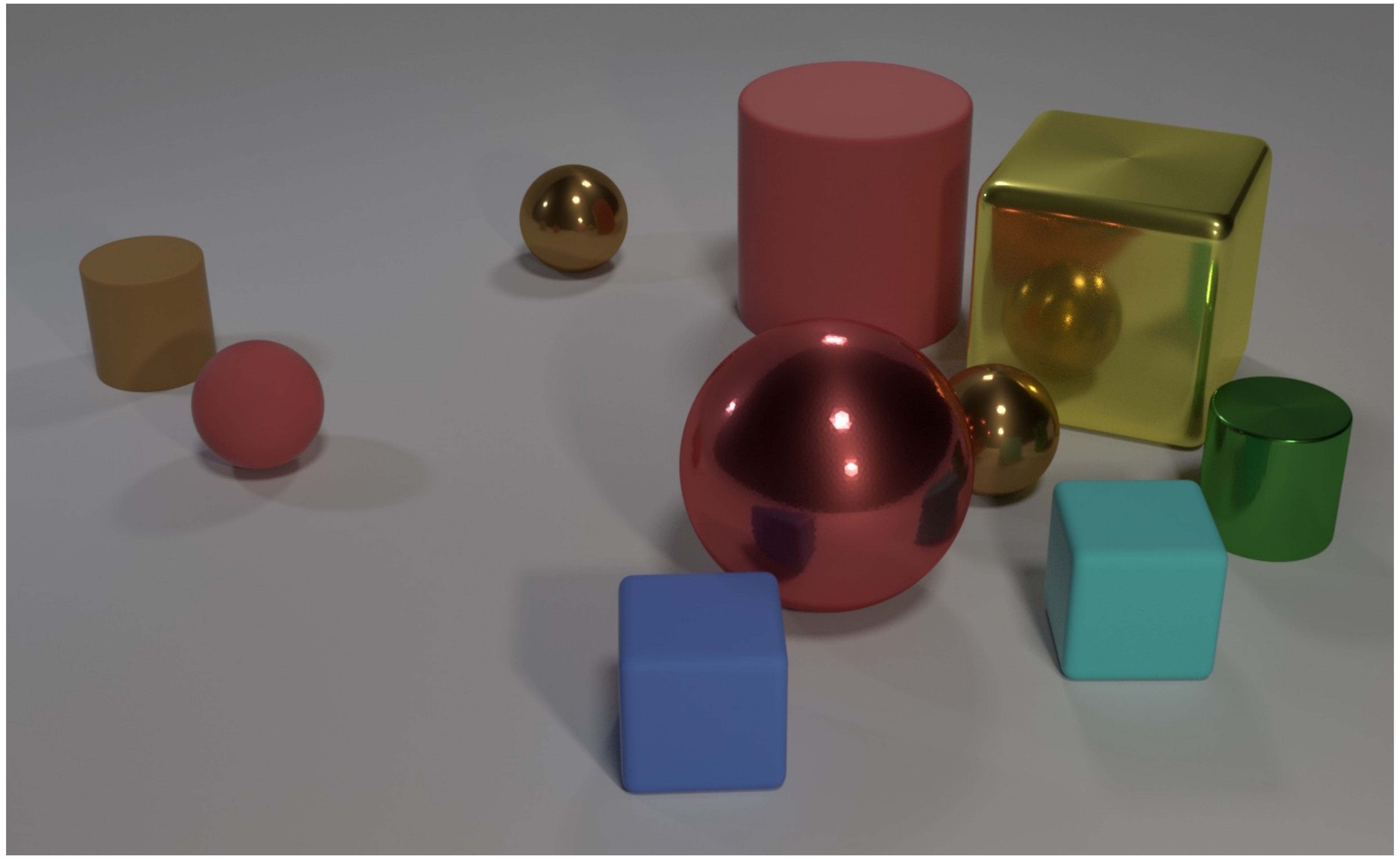
Yan Zhang, Jonathon Hare, Adam Prügel-Bennett (2019) Learning Representations of Sets through Optimized Permutations. International Conference on Learning Representations

Yan Zhang, Jonathon Hare and Adam Prugel-Bennett (2019) FSPool: Learning set representations with featurewise sort pooling. Sets & Partitions: NeurIPS 2019 Workshop

# Motivation

**Learning how to pool sets of vectors**

- Consider a system for answering questions based on an image
  - e.g. You want to ask how many red objects are in the scene to the right
- There are multiple objects in the scene and no canonical ordering (they are a set); if each object was represented by a vector how could we filter that set and reduce it to a single vector?



# Pooling sets of vectors

## Permutation invariance is hard to learn

- **Problem**: Want to learn a function to turn sets (of vectors) into a single vector.
  - Function clearly needs to be permutation invariant
  - MLPs really struggle with learning symmetries in this scenario - for a set of cardinality *n* there are *n!* possible combinations of input that should have the same output
    - The problem arises from discontinuities
  - We still want to learn the function rather than predefine - weighted average, min, max, …

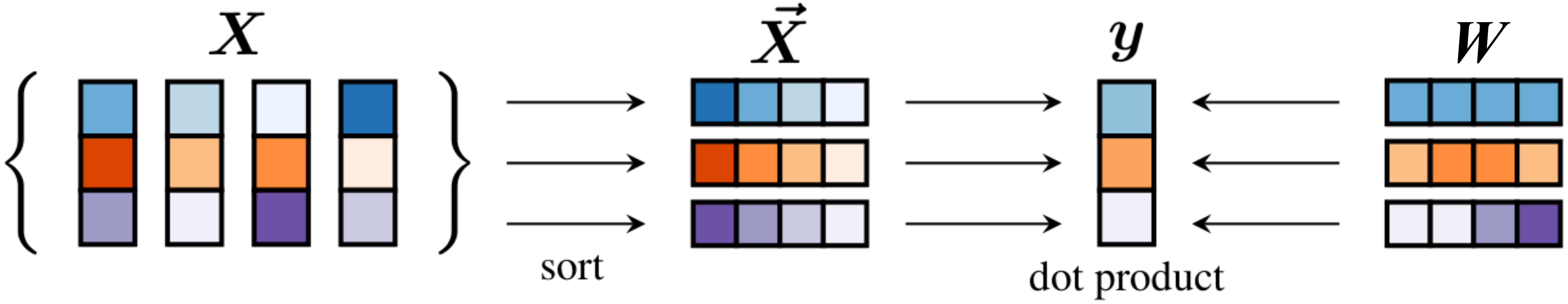
# Featurewise Sort Pooling

## FSPool

# Featurewise Sort Pooling

## FSPool

- **Solution**: sort vectors feature-wise (with a differentiable sort) & take dot-product with a set of learned weights
  - For each feature, across the set of vectors, the dot-product is computing a weighted average
    - Weights of [1,0,0…] selects *min*; [0,0…,1] is *max*; [1,…,1] is the *sum*

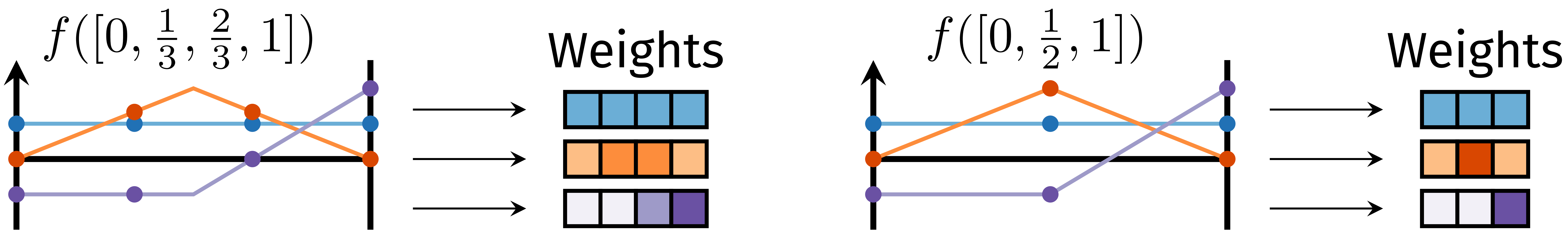


# Variable sized sets

## Piecewise linear approximations

- **Problem:** Previously described method requires a weight per feature per vector; what if the number of vectors can vary?

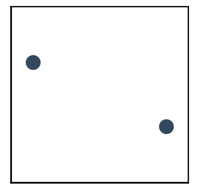
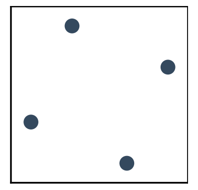
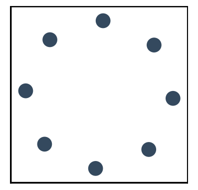
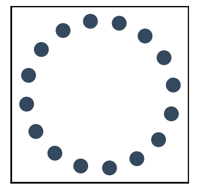
- **Solution:** Continuous relaxation - use a parametric piecewise-linear function (a calibrator function) for each feature to estimate the weight for each of the vectors



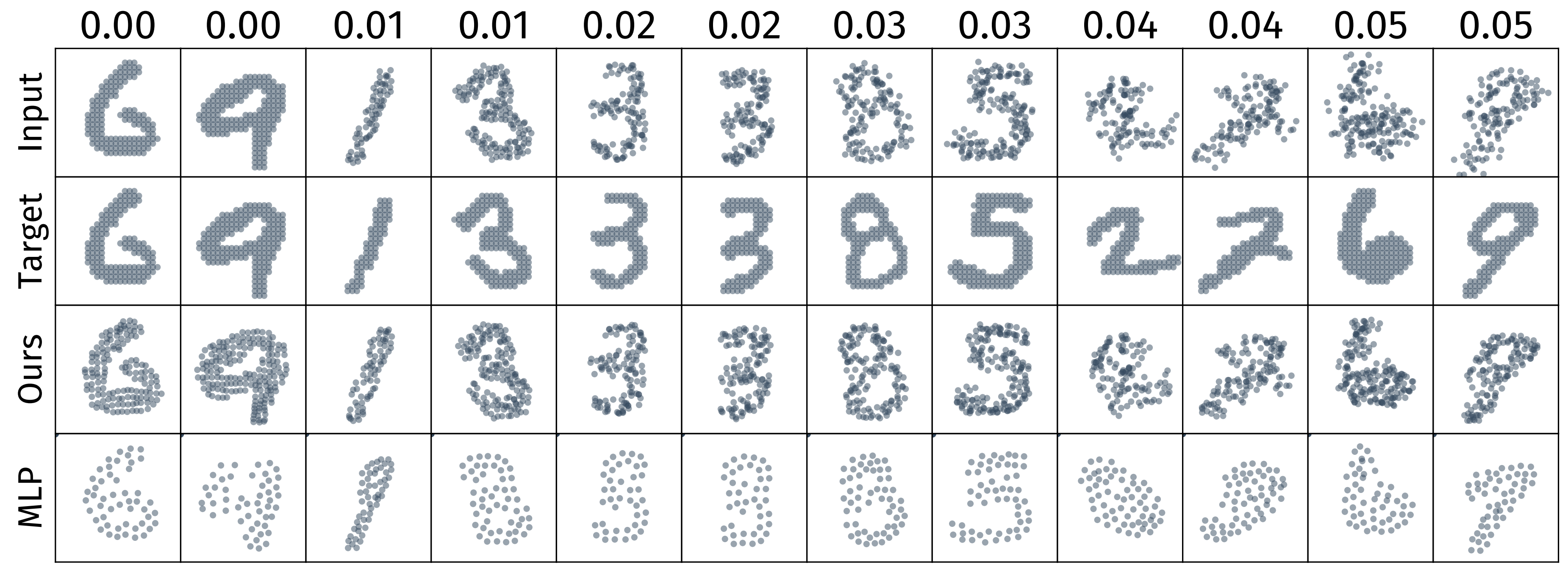
# Experimental results

## Auto-encoding sets

Fixed size

| **Rotating polygons** Eval: Chamfer loss |  |  |  |  |
|---|---|---|---|---|
| **FSPool & FSUnpool** | **0.001** | **0.001** | **0.001** | **0.000** |
| MLP + Chamfer loss | 1.189 | 1.771 | 0.274 | 1.272 |
| MLP + Hungarian loss | 1.517 | 0.400 | 0.251 | 1.266 |
| Random | 72.848 | 19.866 | 5.112 | 1.271 |

**Denoising auto-encoder for different noise levels**




Variable size

## Encoding sets

| **MNIST set classification** | 1 epoch of training | | | 10 epochs of training | | |
|---|---|---|---|---|---|---|
| Weights from auto-encoder: | Frozen | Unfrozen | Random init | Frozen | Unfrozen | Random init |
| **FSPool** | **82.2%**$_{\pm 2.1}$ | **86.9%**$_{\pm 1.3}$ | **84.7%**$_{\pm 1.9}$ | **84.3%**$_{\pm 1.8}$ | **91.5%**$_{\pm 0.5}$ | **91.9%**$_{\pm 0.5}$ |
| Sum | 76.6%$_{\pm 1.3}$ | 68.7%$_{\pm 3.5}$ | 30.3%$_{\pm 5.6}$ | 79.0%$_{\pm 1.0}$ | 77.7%$_{\pm 2.3}$ | 72.7%$_{\pm 3.4}$ |
| Mean | 25.7%$_{\pm 3.6}$ | 32.2%$_{\pm 10.5}$ | 30.1%$_{\pm 1.6}$ | 36.8%$_{\pm 5.0}$ | 75.0%$_{\pm 2.7}$ | 73.0%$_{\pm 1.7}$ |
| Max | 73.6%$_{\pm 1.3}$ | 73.0%$_{\pm 3.5}$ | 56.1%$_{\pm 5.6}$ | 77.3%$_{\pm 0.9}$ | 80.4%$_{\pm 1.8}$ | 76.9%$_{\pm 1.3}$ |

Variable size

| **CLEVR** | | Epochs to reach accuracy | | | Time for |
|---|---|---|---|---|---|
| | Accuracy | 98.00% | 98.50% | 99.00% | 350 epochs |
| **FSPool** | **99.27%**$_{\pm 0.18}$ | **141**$_{\pm 5}$ | **166**$_{\pm 16}$ | **209**$_{\pm 33}$ | 8.8 h |
| RN | 98.98%$_{\pm 0.25}$ | 144$_{\pm 6}$ | 189$_{\pm 29}$ | *268$_{\pm 46}$ | 15.5 h |
| Janossy | 97.00%$_{\pm 0.54}$ | – | – | – | 11.5 h |
| Sum | 99.05%$_{\pm 0.17}$ | 146$_{\pm 13}$ | 191$_{\pm 40}$ | 281$_{\pm 56}$ | 8.0 h |
| Mean | 98.96%$_{\pm 0.27}$ | 169$_{\pm 6}$ | 225$_{\pm 31}$ | 273$_{\pm 33}$ | 8.0 h |
| Max | 96.99%$_{\pm 0.26}$ | – | – | – | 8.0 h |

- Replace sum or max with **FSPool** for consistent improvements.
- Also improves Relation Networks and a top graph classifier.
- **FSUnpool** avoids the responsibility problem, but only in auto-encoders.

# Generating Sets

Yan Zhang, Jonathon Hare and Adam Prugel-Bennett (2019) Deep Set Prediction Networks. In Advances in Neural Information Processing Systems 32. vol. 32, Neural Information Processing Systems.

Yan Zhang and Jonathon Hare and Adam Prügel-Bennett (2020) FSPool: Learning Set Representations with Featurewise Sort Pooling. International Conference on Learning Representations.

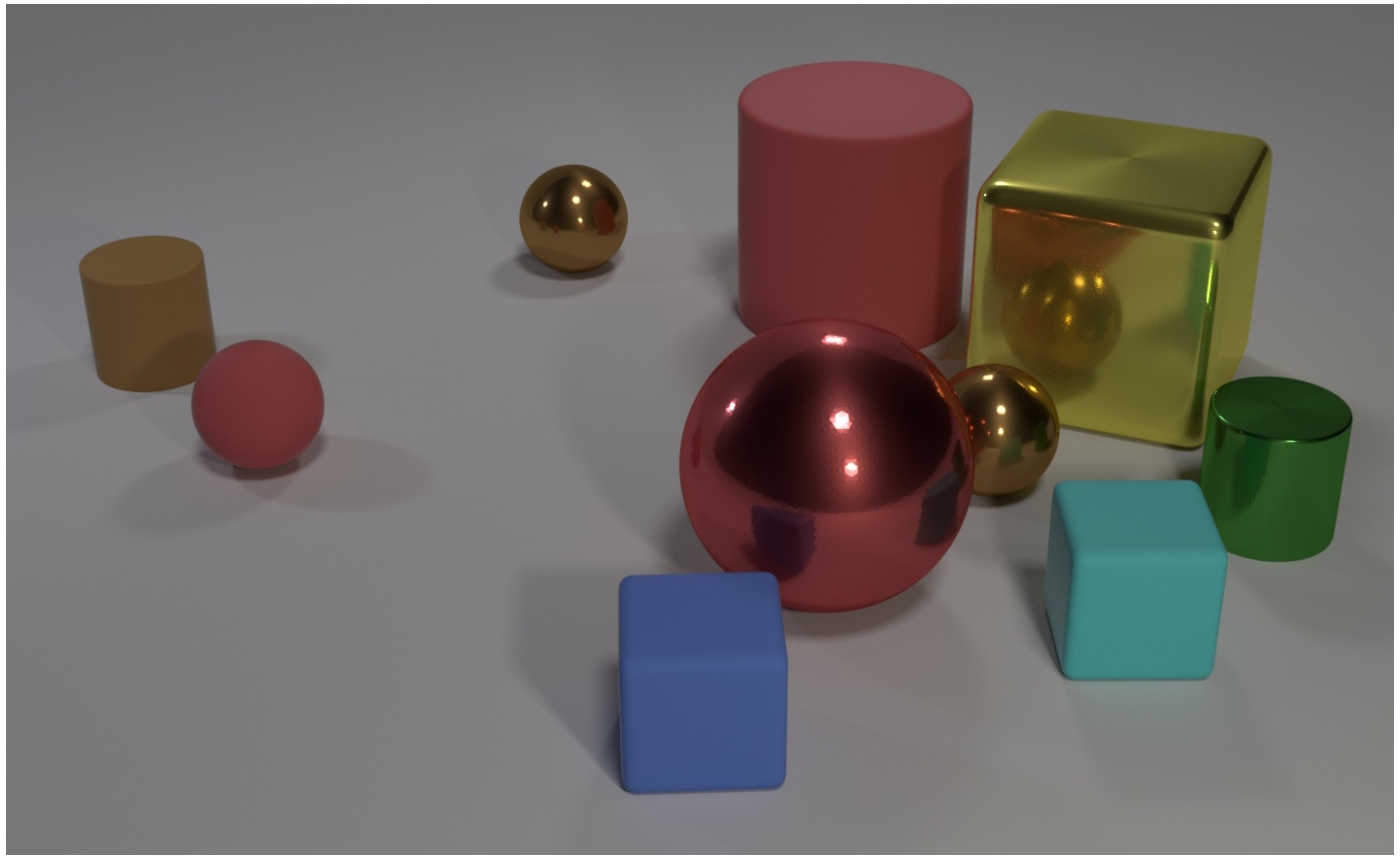
Yan Zhang, Jonathon Hare and Adam Prugel-Bennett (2019) FSPool: Learning set representations with featurewise sort pooling. Sets & Partitions: NeurIPS 2019 Workshop

Yan Zhang, Jonathon Hare and Adam Prugel-Bennett (2019) Deep Set Prediction Networks. Sets & Partitions: NeurIPS 2019 Workshop

# Motivation

**Learning how to pool sets of vectors**

- Many problems involve predicting a set of outputs for a given input
  - predicting all the objects in images (types, positions, etc)
  - molecule generation
  - …



# Predicting sets

## Learning unordered things with an ordered function is hard

- **Problem**: turn a vector (or more generally tensor) into a set of vectors
  - **Applications**: predicting objects in images, molecule generation, …
  - But, MLPs have ordered outputs and sets are by definition unordered

# Predicting sets

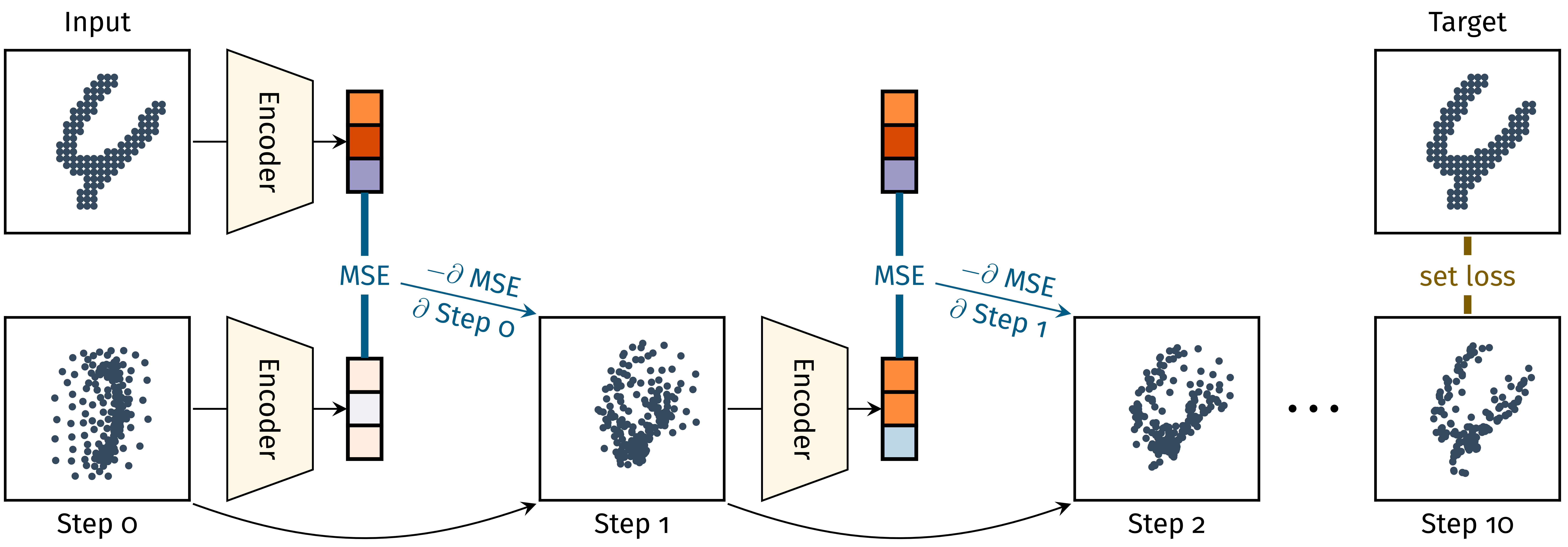
## Learning unordered things with an ordered function is hard

# Reversing an invariant encoder

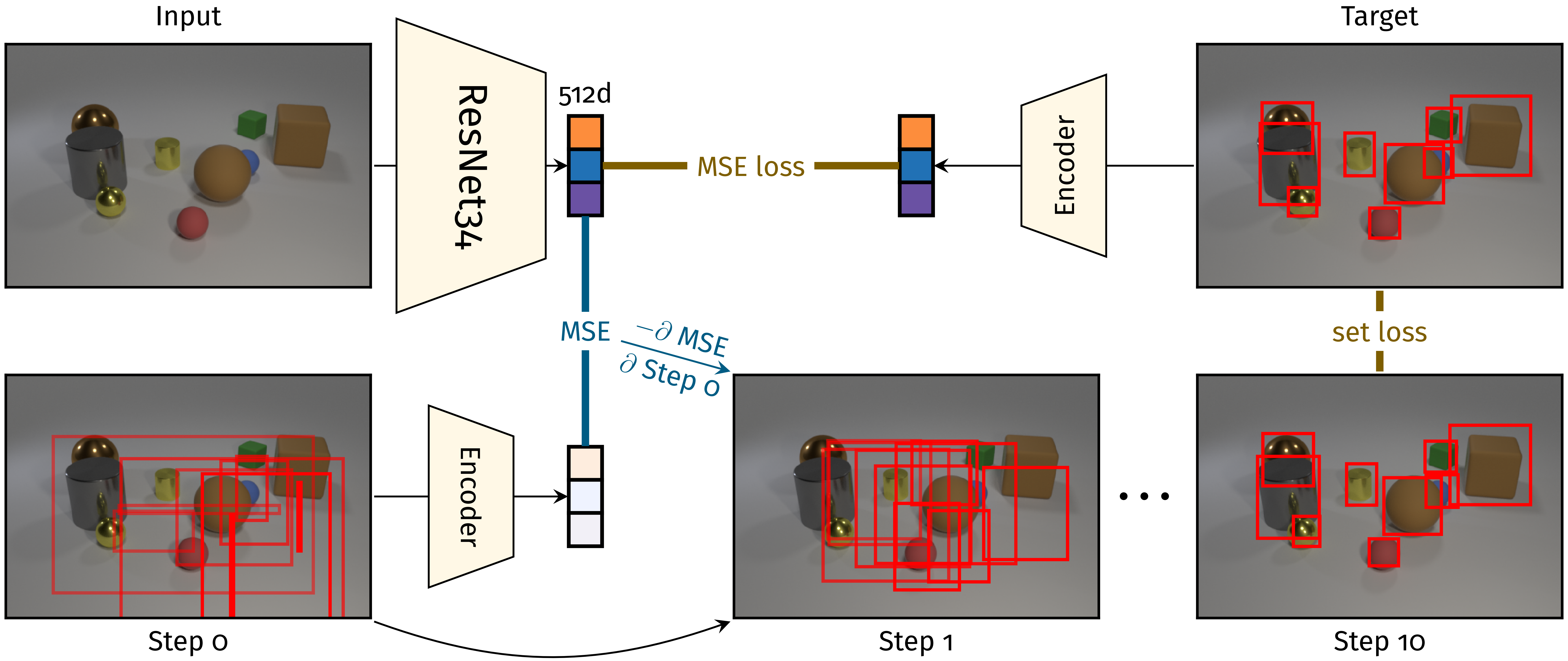
## Deep Set Prediction Networks

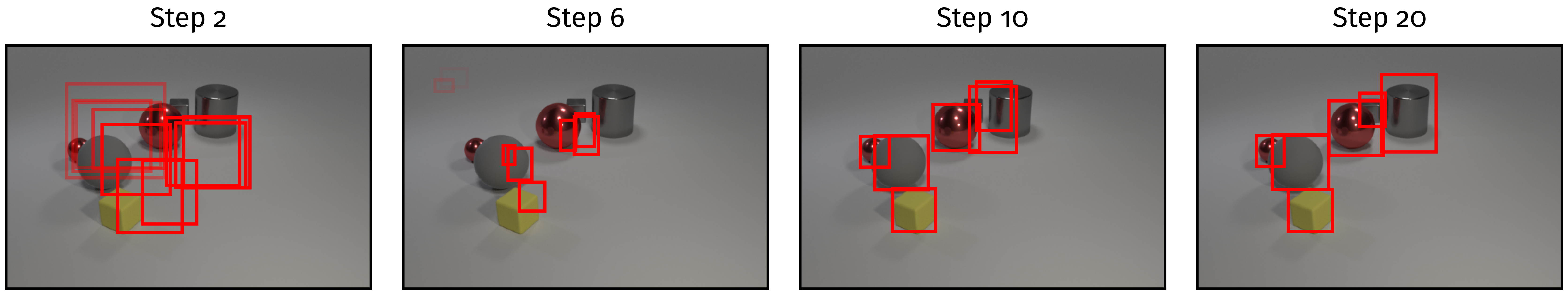
- **Solution**: need to define a function (or procedure) that is *unordered*
  - **Observation**: gradient of a permutation-invariant encoder (set to vector) with respect to the input are permutation equivariant
    - i.e. gradients $\frac{\delta \text{loss}}{\delta \text{set}}$ do not depend on order!
  - **Implication**: to decode a feature vector into a set, we can use gradient descent to find a set that encodes to that feature vector
    - We can define a procedure that iteratively follows gradients in the forward pass

# Autoencoding sets

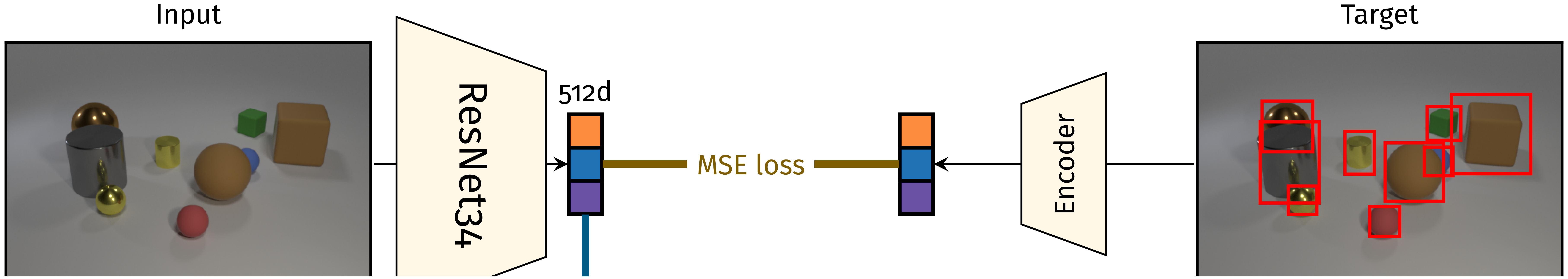


# Object detection

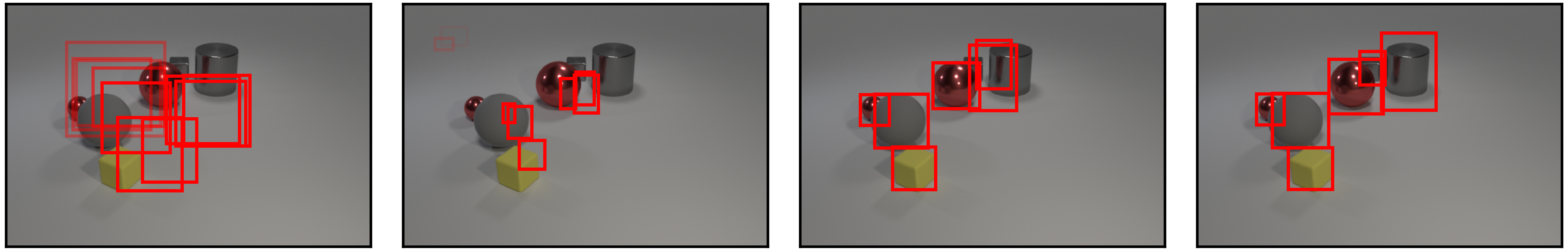


# Object detection

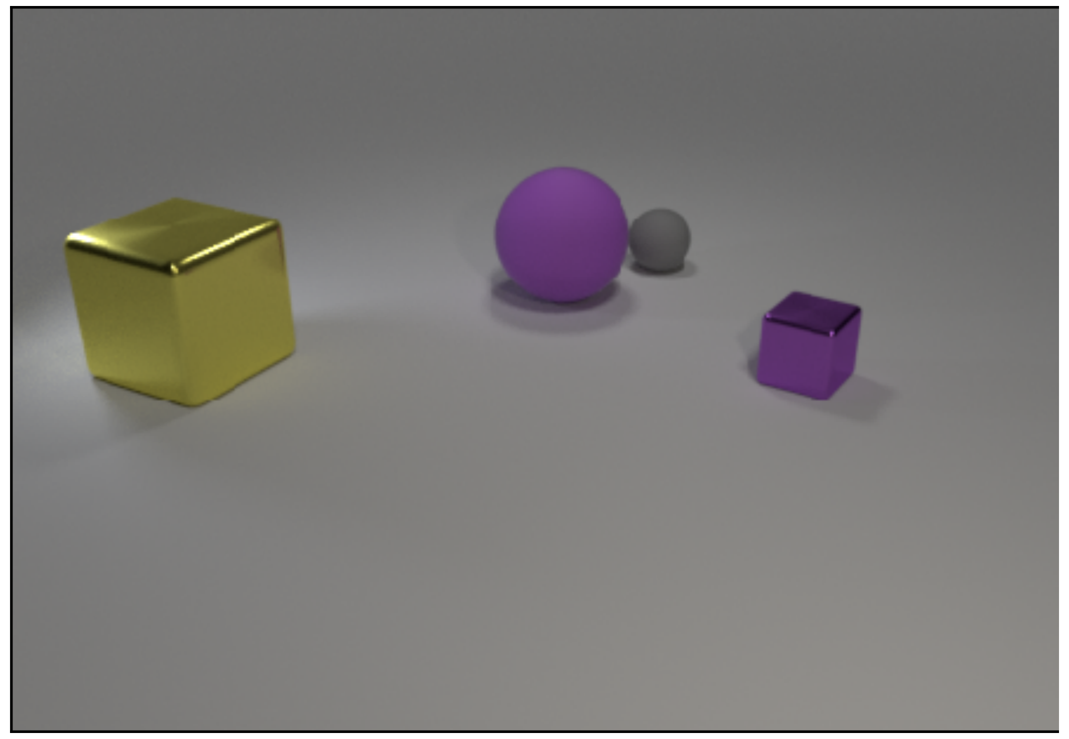



| Bounding box prediction | $AP_{50}$ | $AP_{90}$ | $AP_{95}$ | $AP_{98}$ | $AP_{99}$ |
|---|---|---|---|---|---|
| MLP baseline | $99.3_{\pm 0.2}$ | $94.0_{\pm 1.9}$ | $57.9_{\pm 7.9}$ | $0.7_{\pm 0.2}$ | $0.0_{\pm 0.0}$ |
| RNN baseline | $99.4_{\pm 0.2}$ | $94.9_{\pm 2.0}$ | $65.0_{\pm 10.3}$ | $2.4_{\pm 0.0}$ | $0.0_{\pm 0.0}$ |
| **Ours** (train 10 steps, eval 10 steps) | $98.8_{\pm 0.3}$ | $94.3_{\pm 1.5}$ | $85.7_{\pm 3.0}$ | $\mathbf{34.5}_{\pm 5.7}$ | $\mathbf{2.9}_{\pm 1.2}$ |
| **Ours** (train 10 steps, eval 20 steps) | $\mathbf{99.8}_{\pm 0.0}$ | $\mathbf{98.7}_{\pm 1.1}$ | $\mathbf{86.2}_{\pm 7.2}$ | $24.3_{\pm 8.0}$ | $1.4_{\pm 0.9}$ |
| **Ours** (train 10 steps, eval 30 steps) | $\mathbf{99.8}_{\pm 0.1}$ | $96.7_{\pm 2.4}$ | $75.5_{\pm 12.3}$ | $17.4_{\pm 7.7}$ | $0.9_{\pm 0.7}$ |



# Object and attribute prediction

| Object attribute prediction | $AP_{\infty}$ | $AP_{1}$ | $AP_{0.5}$ | $AP_{0.25}$ | $AP_{0.125}$ |
|---|---|---|---|---|---|
| MLP baseline | $3.6_{\pm 0.5}$ | $1.5_{\pm 0.4}$ | $0.8_{\pm 0.3}$ | $0.2_{\pm 0.1}$ | $0.0_{\pm 0.0}$ |
| RNN baseline | $4.0_{\pm 1.9}$ | $1.8_{\pm 1.2}$ | $0.9_{\pm 0.5}$ | $0.2_{\pm 0.1}$ | $0.0_{\pm 0.0}$ |
| **Ours** (train 10 steps, eval 10 steps) | $72.8_{\pm 2.3}$ | $59.2_{\pm 2.8}$ | $39.0_{\pm 4.4}$ | $12.4_{\pm 2.5}$ | $1.3_{\pm 0.4}$ |
| **Ours** (train 10 steps, eval 20 steps) | $84.0_{\pm 4.5}$ | $80.0_{\pm 4.9}$ | $\mathbf{57.0}_{\pm 12.1}$ | $\mathbf{16.6}_{\pm 9.0}$ | $\mathbf{1.6}_{\pm 0.9}$ |
| **Ours** (train 10 steps, eval 30 steps) | $\mathbf{85.2}_{\pm 4.8}$ | $\mathbf{81.1}_{\pm 5.2}$ | $47.4_{\pm 17.6}$ | $10.8_{\pm 9.0}$ | $0.6_{\pm 0.7}$ |

| Input | Step 5 | Step 10 | Step 20 | Target |
|---|---|---|---|---|
|  | x, y, z = (-0.14, 1.16, 3.57) large purple rubber sphere | x, y, z = (-2.33, -2.41, 0.73) large yellow metal cube | x, y, z = (-2.33, -2.42, 0.78) large yellow metal cube | x, y, z = (-2.42, -2.40, 0.70) large yellow metal cube |
| | x, y, z = (0.01, 0.12, 3.42) large gray metal cube | x, y, z = (-1.20, 1.27, 0.67) large purple rubber sphere | x, y, z = (-1.21, 1.20, 0.65) large purple rubber sphere | x, y, z = (-1.18, 1.25, 0.70) large purple rubber sphere |
| | x, y, z = (0.67, 0.65, 3.38) small purple metal cube | x, y, z = (-0.96, 2.54, 0.36) small gray rubber sphere | x, y, z = (-0.96, 2.59, 0.36) small gray rubber sphere | x, y, z = (-1.02, 2.61, 0.35) small gray rubber sphere |
| | x, y, z = (0.67, 1.14, 2.96) small purple rubber sphere | x, y, z = (1.61, 1.57, 0.36) small yellow metal cube | x, y, z = (1.58, 1.62, 0.38) small purple metal cube | x, y, z = (1.74, 1.53, 0.35) small purple metal cube |