Deep Set Prediction Networks

Yan Zhang
University of Southampton
Southampton, UK
yz5n12@ecs.soton.ac.uk

Jonathon Hare
University of Southampton
Southampton, UK
jsh2@ecs.soton.ac.uk

Adam Prügel-Bennett
University of Southampton
Southampton, UK
apb@ecs.soton.ac.uk

Abstract

We study the problem of predicting a set from a feature vector with a deep neural network. Existing approaches ignore the set structure of the problem and suffer from discontinuity issues as a result. We propose a general model for predicting sets that properly respects the structure of sets and avoids this problem. With a single feature vector as input, we show that our model is able to auto-encode point sets, predict bounding boxes of the set of objects in an image, and predict the attributes of these objects in an image.

1 Introduction

The full version of this paper is available as [19]. Object detection – predicting the set of objects in an image – is an example of a *set prediction* problem. The main difficulty in predicting sets comes from the ability to permute the elements in a set freely, which means that there are $n!$ equally good solutions for a set of size n. MLPs and RNNs – the currently prevailing approach for set prediction – do not take this structure into account, which leads to discontinuities and hinders training [2].

How can we build a model that properly respects the set structure of the problem so that we can predict sets without running into discontinuity issues? In this paper, we aim to address this question.

1. We propose a model that can predict a set from a feature vector while properly taking the structure of sets into account (section 3). We explain what properties we make use of that enables this. Our model uses backpropagation through a set encoder to decode a set and works for variable-size sets. The model is applicable to a wide variety of set prediction tasks since it only requires a feature vector as input.
2. We evaluate our model on several set prediction datasets (section 4): auto-encoding a set version of MNIST, bounding box prediction on CLEVR, and object attribute prediction on CLEVR. Our model is a completely different approach to usual anchor-based object detectors because we pose the task as a set prediction problem, which does not need complicated post-processing techniques such as non-maximum suppression.

2 Background: Responsibility problem

A widely-used approach is to simply ignore the set structure of the problem. A feature vector can be mapped to a set $\hat{\boldsymbol{Y}}$ by using an MLP that takes a vector as input and directly produces $\hat{\boldsymbol{Y}}$ with $d \times n$ outputs, where d is the dimensionality of each feature vector in the set and n is the set size. Since the order of elements in $\hat{\boldsymbol{Y}}$ does not matter, it appears reasonable to always produce them in a certain order based on the weights of the MLP.

However, [2] point out that this results in a discontinuity issue: there are points where a small change in set space requires a large change in the neural network outputs. The model needs to "decide" which

Sets & Partitions Workshop at 33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

Algorithm 1 Forward pass of set prediction algorithm

$\boldsymbol{z} = F(x)$ ▷ encode input with a model
$\hat{\boldsymbol{Y}}^{(0)} \leftarrow$ init ▷ initialise set
for t $\leftarrow 1, T$ **do**
  $l \leftarrow L_{\text{repr}}(\hat{\boldsymbol{Y}}^{(t-1)}, \boldsymbol{z})$ ▷ compute representation loss
  $\hat{\boldsymbol{Y}}^{(t)} \leftarrow \hat{\boldsymbol{Y}}^{(t-1)} - \eta \frac{\partial l}{\partial \hat{\boldsymbol{Y}}^{(t-1)}}$ ▷ gradient descent step on the set
end for
predict $\hat{\boldsymbol{Y}}^{(T)}$
$\mathcal{L} = \frac{1}{T} \sum_{t=0}^{T} L_{\text{set}}(\hat{\boldsymbol{Y}}^{(t)}, \boldsymbol{Y}) + \lambda L_{\text{repr}}(\boldsymbol{Y}, \boldsymbol{z})$ ▷ compute loss of outer optimisation

of its outputs is responsible for producing which element, and this responsibility must be resolved discontinuously.

The main problem is that the output of an MLP or RNN is inherently *ordered*, which forces the model to give the set an order. Instead, it is preferable if the outputs of the model themselves are freely interchangeable – in the same way the elements of the set are interchangeable – to not impose an order on the outputs. We are thus looking for a model that has inherently unordered outputs so that it models the unordered nature of sets correctly.

3 Deep Set Prediction Networks

Our main idea is based on the observation that the gradient of a set encoder with respect to the input set is permutation-equivariant (see proof in Appendix A): *to decode a feature vector into a set, we can use gradient descent to find a set that* encodes *to that feature vector*. Since each update of the set using the gradient is permutation-equivariant, we always properly treat it as a set and avoid the responsibility problem. This gives rise to a nested optimisation: an inner loop that changes a set to encode more similarly to the input feature vector, and an outer loop that changes the weights of the encoder to minimise a loss over a dataset.

Our model is easiest to explain with an auto-encoder. In a set auto-encoder, the goal is to turn the input set $\boldsymbol{Y}$ into a small latent space $\boldsymbol{z} = g_{\text{enc}}(\boldsymbol{Y})$ with the encoder g_{enc} and turn it back into the predicted set $\hat{\boldsymbol{Y}} = g_{\text{dec}}(\boldsymbol{z})$ with the decoder g_{dec}. Using our main idea, we define a *representation loss* and the corresponding decoder as:

$$L_{\text{repr}}(\hat{\boldsymbol{Y}}, \boldsymbol{z}) = ||g_{\text{enc}}(\hat{\boldsymbol{Y}}) - \boldsymbol{z}||^2 \tag{1}$$

$$g_{\text{dec}}(\boldsymbol{z}) = \arg\min_{\hat{\boldsymbol{Y}}} L_{\text{repr}}(\hat{\boldsymbol{Y}}, \boldsymbol{z}) \tag{2}$$

In essence, L_{repr} compares $\hat{\boldsymbol{Y}}$ to $\boldsymbol{Y}$ in the latent space. To understand what the decoder does, first consider the simple, albeit not very useful case of the identity encoder $g_{\text{enc}}(\boldsymbol{Y}) = \boldsymbol{Y}$. Solving $g_{\text{dec}}(\boldsymbol{z})$ simply means setting $\hat{\boldsymbol{Y}} = \boldsymbol{Y}$, which perfectly reconstructs the input as desired.

When we instead choose g_{enc} to be a set encoder, the latent representation $\boldsymbol{z}$ is a permutation-invariant feature vector. If this representation is "good", $\hat{\boldsymbol{Y}}$ will only encode to similar latent variables as $\boldsymbol{Y}$ if the two sets themselves are similar. Thus, the minimisation in Equation 2 should still produce a set $\hat{\boldsymbol{Y}}$ that is the same (up to permutation) as $\boldsymbol{Y}$, except this has now been achieved with $\boldsymbol{z}$ as a bottleneck.

We then perform gradient descent to find an approximate solution. Starting from some initial set $\hat{\boldsymbol{Y}}^{(0)}$, gradient descent is performed for a fixed number of steps T with the update rule:

$$\hat{\boldsymbol{Y}}^{(t+1)} = \hat{\boldsymbol{Y}}^{(t)} - \eta \cdot \frac{\partial L_{\text{repr}}(\hat{\boldsymbol{Y}}^{(t)}, \boldsymbol{z})}{\partial \hat{\boldsymbol{Y}}^{(t)}} \tag{3}$$

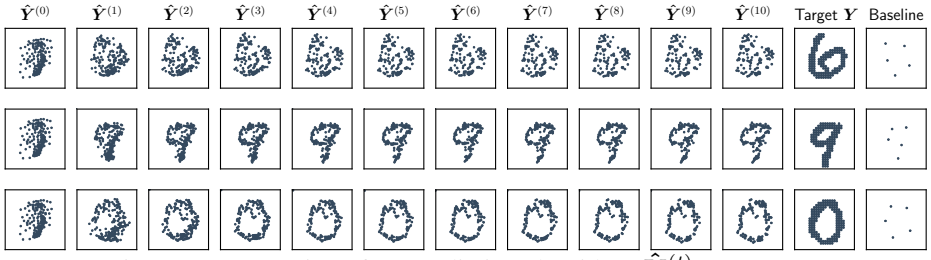


Figure 1: Progression of set prediction algorithm ($\hat{\boldsymbol{Y}}^{(t)}$) on MNIST.

with η as the learning rate and the prediction being the final state, $g_{\text{dec}}(\boldsymbol{z}) = \hat{\boldsymbol{Y}}^{(T)}$. This is the aforementioned inner optimisation loop. In practice, we let $\hat{\boldsymbol{Y}}^{(0)}$ be a learnable $\mathbb{R}^{d \times n}$ matrix which is part of the neural network parameters.

To obtain a good representation $\boldsymbol{z}$, we still have to train the weights of g_{enc}. For this, we compute the auto-encoder objective $L_{\text{set}}(\hat{\boldsymbol{Y}}^{(T)}, \boldsymbol{Y})$ – with L_{set} as the Chamfer or Hungarian loss – and differentiate with respect to the weights as usual, backpropagating through the steps of the inner optimisation. This is the aforementioned outer optimisation loop.

In summary, each forward pass of our auto-encoder first encodes the input set to a latent representation as normal. To decode this back into a set, gradient descent is performed on an initial guess with the aim to obtain a set that encodes to the same latent representation as the input. The same set encoder is used in encoding and decoding.

Notice how our algorithm only depends on $\boldsymbol{z}$ as input, not the input set. This means that we can use anything that produces a feature vector for $\boldsymbol{z}$, like an image encoder. This immediately gives us a model that is not limited to the auto-encoder setting, but can be used for general set prediction.

We describe how to modify the loss function for the supervised case in Appendix B. The minor changes necessary for variable-size sets are located in Appendix C.

4 Experiments

We now compare our set prediction network to a model that uses an MLP or RNN as set decoder, which matches approaches such as in [13] and [15]. Further details about the model architectures, training settings, and hyperparameters are given in Appendix E. The source code to reproduce all experiments is available at [redacted].

4.1 MNIST

We begin with the task of auto-encoding a set version of MNIST. A set is constructed from each image by including all the pixel coordinates (x and y, scaled to the interval $[0, 1]$) of pixels that have a value above the mean pixel value. We compare our model against a baseline where the decoder is replaced with an MLP.

Results In Figure 1, we show the progression of $\hat{\boldsymbol{Y}}$ throughout the minimisation with $\hat{\boldsymbol{Y}}^{(10)}$ as the final prediction, the ground-truth set, and the baseline prediction. Our model clearly predicts much better sets than the baseline model, which only learned to output very few points in the rough shape of the digit. Observe how every optimisation starts with the same set $\hat{\boldsymbol{Y}}^{(0)}$, but is transformed differently depending on the gradient of g_{enc}. Through this minimisation of L_{repr} by the inner optimisaton, the set is gradually changed into a shape that closely resembles the correct digit.

4.2 Bounding box prediction

Next, we turn to the task of object detection on the CLEVR dataset [7]. The goal is to predict the set of bounding boxes for the objects in an image. We use ResNet-34 as image encoder to produce a feature vector, which is then decoded with our model. As baseline, we replace our decoder with an MLP or an RNN. These two baselines closely match the approaches in [13] and [15] respectively.

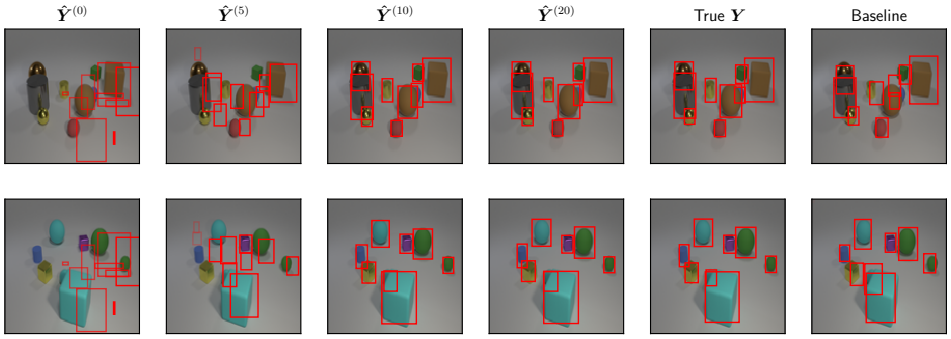


Figure 2: Progression of set prediction algorithm for bounding boxes in CLEVR. The baseline sometimes struggles with heavily-overlapping objects and often fails to centre the object in the boxes.

Table 1: Average Precision (AP) for different intersection-over-union thresholds for a predicted bounding box to be considered correct. Mean and standard deviation over 6 runs.

Model	AP_{50}	AP_{90}	AP_{95}	AP_{98}	AP_{99}
MLP Baseline	99.3±0.2	94.0±1.9	57.9±7.9	0.7±0.2	0.0±0.0
RNN Baseline	99.4±0.2	94.9±2.0	65.0±10.3	2.4±0.0	0.0±0.0
Ours (10 iters)	98.8±0.3	94.3±1.5	85.7±3.0	**34.5**±5.7	**2.9**±1.2
Ours (20 iters)	**99.8**±0.0	**98.7**±1.1	**86.2**±7.2	24.3±8.0	1.4±0.9
Ours (30 iters)	**99.8**±0.1	96.7±2.4	75.5±12.3	17.4±7.7	0.9±0.7

Table 2: Average Precision (AP) in % for different distance thresholds of a predicted set element to be considered correct. AP_∞ only requires all attributes to be correct, regardless of 3d position. Mean and standard deviation over 6 runs.

Model	AP_∞	AP_1	$AP_{0.5}$	$AP_{0.25}$	$AP_{0.125}$
MLP Baseline	3.6±0.5	1.5±0.4	0.8±0.3	0.2±0.1	0.0±0.0
RNN Baseline	4.0±1.9	1.8±1.2	0.9±0.5	0.2±0.1	0.0±0.0
Ours (10 iters)	72.8±2.3	59.2±2.8	39.0±4.4	12.4±2.5	1.3±0.4
Ours (20 iters)	84.0±4.5	80.0±4.9	**57.0**±12.1	**16.6**±9.0	**1.6**±0.9
Ours (30 iters)	**85.2**±4.8	**81.1**±5.2	47.4±17.6	10.8±9.0	0.6±0.7

Results Our results in Table 1 show that our model is consistently superior, especially for the very strict AP_{98} and AP_{99} metrics. We can also run our model with more inner optimisation steps than the 10 it was trained with. Up to AP_{95}, the results improve when doubling the number of steps, which shows that further minimisation of $L_{\text{repr}}(\hat{\boldsymbol{Y}}, \boldsymbol{z})$ is still beneficial, even if it is unseen during training.

4.3 State prediction

Lastly, we want to directly predict the full state of a scene from images on CLEVR. This is the set of objects with their position in the 3d scene (x, y, z coordinates), shape (sphere, cylinder, cube), colour (eight colours), size (small, large), and material (metal/shiny, rubber/matte) as features. For example, an object can be a "small cyan metal cube" at position (0.95, -2.83, 0.35). We use the same model and baselines as before, except each set element to be predicted now has 18 dimensions instead of 4.

Results We show our results in Table 2 and give sample outputs in Appendix F. The evaluation metric is the standard average precision as used in object detection, with the modification that a prediction is considered correct if there is a matching groundtruth object with exactly the same properties and within a given Euclidean distance of the 3d coordinates.

Our model outperforms the baselines by a large margin. This shows that our model is also suitable for modeling high-dimensional set elements. This time, the results for 20 iterations are all better than for 10 iterations, which suggests that using only 10 steps in training is not enough and that increasing this would give further benefits.

References

[1] Achlioptas, P., Diamanti, O., Mitliagkas, I., and Guibas, L. J. Learning representations and generative models for 3D point clouds. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, 2018.

[2] Anonymous. FSPool: Learning set representations with featurewise sort pooling. 2019.

[3] Balles, L. and Fischbacher, T. Holographic and other point set distances for machine learning, 2019. URL `https://openreview.net/forum?id=rJlpUiAcYX`.

[4] Cao, N. D. and Kipf, T. MolGAN: An implicit generative model for small molecular graphs. In *ICML Deep Generative Models Workshop*, 2018.

[5] Fan, H., Su, H., and Guibas, L. J. A point set generation network for 3D object reconstruction from a single image. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[6] Greff, K., Kaufmann, R. L., Kabra, R., Watters, N., Burgess, C., Zoran, D., Matthey, L., Botvinick, M., and Lerchner, A. Multi-object representation learning with iterative variational inference. arXiv:1903.00450, 2019.

[7] Johnson, J., Hariharan, B., van der Maaten, L., Fei-Fei, L., Lawrence Zitnick, C., and Girshick, R. CLEVR: A diagnostic dataset for compositional language and elementary visual reasoning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[8] Lazarow, J., Jin, L., and Tu, Z. Introspective neural networks for generative modeling. In *The IEEE International Conference on Computer Vision (ICCV)*, pp. 2774–2783, 2017.

[9] Lee, K., Xu, W., Fan, F., and Tu, Z. Wasserstein introspective neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[10] Li, C.-L., Zaheer, M., Zhang, Y., Poczos, B., and Salakhutdinov, R. Point cloud GAN. arXiv:1810.05795, 2018.

[11] Li, Y., Vinyals, O., Dyer, C., Pascanu, R., and Battaglia, P. Learning deep generative models of graphs. arXiv:1803.03324, 2018.

[12] Mordatch, I. Concept learning with energy-based models. arXiv:1811.02486, 2018.

[13] Rezatofighi, S. H., Kaskman, R., Motlagh, F. T., Shi, Q., Cremers, D., Leal-Taixé, L., and Reid, I. Deep perm-set net: Learn to predict sets with unknown permutation and cardinality using deep neural networks. arXiv:1805.00613, 2018.

[14] Simonovsky, M. and Komodakis, N. GraphVAE: Towards generation of small graphs using variational autoencoders. In *International Conference on Artificial Neural Networks (ICANN)*, 2018.

[15] Stewart, R. and Andriluka, M. End-to-end people detection in crowded scenes. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[16] Vinyals, O., Bengio, S., and Kudlur, M. Order Matters: Sequence to sequence for sets. In *International Conference on Learning Representations (ICLR)*, 2015.

[17] Yang, Y., Feng, C., Shen, Y., and Tian, D. FoldingNet: Point cloud auto-encoder via deep grid deformation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[18] You, J., Ying, R., Ren, X., Hamilton, W., and Leskovec, J. GraphRNN: Generating realistic graphs with deep auto-regressive models. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, 2018.

[19] Zhang, Y., Hare, J., and Prügel-Bennett, A. Deep Set Prediction Networks. In *Advances in Neural Information Processing Systems*, 2019. URL `https://arxiv.org/abs/1906.06565`.

A Proof of permutation-equivariance

Definition 1. *A function $f : \mathbb{R}^{n \times c} \to \mathbb{R}^d$ is permutation-invariant iff it satisfies:*

$$f(\boldsymbol{X}) = f(\boldsymbol{P}\boldsymbol{X}) \tag{4}$$

for all permutation matrices $\boldsymbol{P}$.

Definition 2. *A function $g : \mathbb{R}^{n \times c} \to \mathbb{R}^{n \times d}$ is permutation-equivariant iff it satisfies:*

$$\boldsymbol{P}g(\boldsymbol{X}) = g(\boldsymbol{P}\boldsymbol{X}) \tag{5}$$

for all permutation matrices $\boldsymbol{P}$.

Theorem 1. *The gradient of a permutation-invariant function $f : \mathbb{R}^{n \times c} \to \mathbb{R}^d$ with respect to its input is permutation-equivariant:*

$$\boldsymbol{P}\frac{\partial f(\boldsymbol{X})}{\partial \boldsymbol{X}} = \frac{\partial f(\boldsymbol{P}\boldsymbol{X})}{\partial \boldsymbol{P}\boldsymbol{X}} \tag{6}$$

Proof. Using Definition 1, the chain rule, and the orthogonality of $\boldsymbol{P}$:

$$\boldsymbol{P}\frac{\partial f(\boldsymbol{X})}{\partial \boldsymbol{X}} = \boldsymbol{P}\frac{\partial f(\boldsymbol{P}\boldsymbol{X})}{\partial \boldsymbol{X}} \tag{7}$$

$$= \boldsymbol{P}\frac{\partial \boldsymbol{P}\boldsymbol{X}}{\partial \boldsymbol{X}}\frac{\partial f(\boldsymbol{P}\boldsymbol{X})}{\partial \boldsymbol{P}\boldsymbol{X}} \tag{8}$$

$$= \boldsymbol{P}\boldsymbol{P}^T\frac{\partial f(\boldsymbol{P}\boldsymbol{X})}{\partial \boldsymbol{P}\boldsymbol{X}} \tag{9}$$

$$= \frac{\partial f(\boldsymbol{P}\boldsymbol{X})}{\partial \boldsymbol{P}\boldsymbol{X}} \tag{10}$$

□

B Predicting sets from a feature vector

In our auto-encoder, we used an encoder to produce both the latent representation as well as to decode the set. This is no longer possible in the general set prediction setup, since the target representation $\boldsymbol{z}$ can come from a separate model (for example an image encoder F encoding an image $\boldsymbol{x}$), so there is no longer a set encoder in the model.

When naïvely using $\boldsymbol{z} = F(\boldsymbol{x})$ as input to our decoder, our decoding process is unable to predict sets correctly from it. Because the set encoder is no longer shared in our set decoder, there is no guarantee that optimising $g_{\text{enc}}(\hat{\boldsymbol{Y}})$ to match $\boldsymbol{z}$ converges towards $\boldsymbol{Y}$ (or a permutation thereof). To fix this, we simply add a term to the loss of the outer optimisation that encourages $g_{\text{enc}}(\boldsymbol{Y}) \approx \boldsymbol{z}$ again. In other words, the target set should have a very low representation loss itself. This gives us an additional L_{repr} term in the loss function of the outer optimisation for supervised learning:

$$\mathcal{L} = L_{\text{set}}(\hat{\boldsymbol{Y}}, \boldsymbol{Y}) + \lambda L_{\text{repr}}(\boldsymbol{Y}, \boldsymbol{z}) \tag{11}$$

with L_{set} again being either L_{cha} or L_{hun}. With this, minimising $L_{\text{repr}}(\hat{\boldsymbol{Y}}, \boldsymbol{z})$ in the inner optimisation will converge towards $\boldsymbol{Y}$. The additional term is not necessary in the pure auto-encoder because $\boldsymbol{z} = g_{\text{enc}}(\boldsymbol{Y})$, so $L_{\text{repr}}(\boldsymbol{Y}, \boldsymbol{z})$ is always 0 already.

Practical tricks For the outer optimisation, we can compute the set loss for not only $\hat{\boldsymbol{Y}}^{(T)}$, but all $\hat{\boldsymbol{Y}}^{(t)}$. That is, we use $\frac{1}{T}\sum_t L_{\text{set}}(\hat{\boldsymbol{Y}}^{(t)}, \boldsymbol{Y})$ as loss. This encourages $\hat{\boldsymbol{Y}}$ to converge to $\boldsymbol{Y}$ quickly and not diverge with more steps, which significantly increases the robustness of our algorithm.

We sometimes observed divergent training behaviour when the outer learning rate is set inappropriately. By replacing the instances of $||\cdot||^2$ in L_{set} and L_{repr} with the Huber loss (squared error for differences below 1 and absolute error above 1) – as is commonly done in object detection models – training became less sensitive to hyperparameter choices.

C Extension to variable-size sets

To extend this from fixed- to variable-size sets, we make a few modifications to this algorithm. First, we pad all sets to a fixed maximum size to allow for efficient batch computation. We then concatenate an additional mask feature m_i to each set element $\hat{\boldsymbol{y}}_i$ that indicates whether it is a regular element ($m_i = 1$) or padding ($m_i = 0$). With this modification to $\hat{\boldsymbol{Y}}$, we can optimise the masks in the same way as the set elements are optimised. To ensure that masks stay in the valid range between 0 and 1, we simply clamp values above 1 to 1 and values below 0 to 0 after each gradient descent step. This performed better than using a sigmoid in our initial experiments, possibly because it allows exact 0s and 1s to be recovered.

D Related work

The main approach we compare our method to is the simple method of using an MLP decoder to predict sets. This has been used for predicting point clouds [1; 5], bounding boxes [13; 3], and graphs (sets of nodes and edges) [4; 14]. These predict an ordered representation (list) and treat it as if it is unordered (set). As we previously discussed, this approach runs into the responsibility problem. Some works on predicting 3d point clouds make domain-specific assumptions such as independence of points within a set [10] or grid-like structures [17].

An alternative approach is to use an RNN decoder to generate this list [11; 15; 16]. The problem can be made easier if it can be turned from a set into a sequence problem by giving a canonical order to the elements in the set through domain knowledge [16]. For example, [18] generate the nodes of a graph by ordering the set of nodes based on the node traversal order of a breadth-first search.

The closest work to ours is by Mordatch [12]. They also iteratively minimise a function (their energy function) in each forward pass of the neural network and differentiate through the iteration to learn the weights. They have only demonstrated that this works for modifying small sets of 2d elements in relatively simple ways, so it is unclear whether their approach scales to the harder problems such as object detection that we tackle in this paper. In particular, minimising L_{repr} in our model has the easy-to-understand consequence of making the predicted set more similar to the target set, while it is less clear what minimising their learned energy function $E(\hat{\boldsymbol{Y}}, \boldsymbol{z})$ does.

Anonymous [2] construct an auto-encoder that pools a set into a feature vector where information from the encoder is shared with their decoder. This is done to make their decoder permutation-equivariant, which they use to avoid the responsibility problem. However, this strictly limits their decoder to usage in auto-encoders because it requires an encoder to be present during inference.

Greff et al. [6] construct an auto-encoder for images with a set-structured latent space. They are able to find latent sets of variables to describe an image composed of a set of objects with some task-specific assumptions. While interesting from a representation learning perspective, our model is more immediately useful in practice because it works for general supervised learning tasks.

Our inspiration for using backpropagation through an encoder as a decoder comes from the line of introspective neural networks [8; 9] for image modeling. An important difference is that in these works, their two optimisation loops (generating predictions/samples and learning the network weights) are performed in sequence, while ours are nested. The nesting allows our outer optimisation to differentiate through the inner optimisation. Note that [6] and [12] also differentiate through an optimisation, which suggests that this type of optimisation is of general use when working with sets.

E Details

In our algorithm, η was chosen in initial experiments and we did not tune it beyond that. We did this by increasing η until the output set visibly changed between inner optimisation steps when the set encoder is randomly initialised. This makes it so that changing the set encoder weights has a noticeable effect rather than being stuck with $\hat{\boldsymbol{Y}}^{(T)} \approx \hat{\boldsymbol{Y}}^{(0)}$.

$T = 10$ was chosen because it seemed to be enough to converge to good solutions on MNIST. We simply kept this for the supervised experiments on CLEVR.

In the supervised experiments, we would often observe large spikes in training that cause the model diverge when $\lambda = 1$. By changing around various parameters, we found that reducing λ eliminated most of this issue and also made training converge to better solutions. Much smaller values than 0.1 converged to worse solutions. This is likely because the issue of not having the $L_{\text{repr}}(\boldsymbol{Y}, \boldsymbol{z})$ term in the outer loss in the first place ($\lambda = 0$) is present again.

For all experiments, we used Adam with the default momentum values and batch size 32. The only hyperparameter we tuned in the experiments is the learning rate. Every individual experiment is run on a single 1080 Ti GPU.

We tried an alternative version of the inner optimisation that includes a momentum term. This gave us slightly better results, but we did not use this for any experiments in the paper to keep our method as simple as possible.

E.1 MNIST

For MNIST, we train our model and the baseline model for 100 epochs to make sure that they have converged. Both models have a 3-layer MLP with ReLU activations and 256 neurons in the first two layers and 64 in the third. For simplicity, sets are padded to a fixed size for FSPool. FSPool has 20 pieces in its piecewise linear function. We tried learning rates in $\{1.0, 0.1, 0.03, 0.01, 0.003, 0.001, 0.0003, 0.0001, 0.00001\}$ and chose 0.01. For the baseline, none of the other learning rates performed significantly better than the one we chose.

The baseline is trained slightly differently to our model. The baseline does not output mask values natively, so we have to train it with the mask values in the training target. In other words, it is trained to predict x coordinate, y coordinate, and the mask for each point. Better results (but still worse than our model) with fewer predicted padding points can be achieved by not including this mask value in the target, but this baseline model no longer predicts variable-sized sets. Our model is trained without mask values in the target and is able to learn the mask values purely through L_{repr} in the inner optimisation.

It is possible to explicitly include the sum of masks as a feature in the representation $\boldsymbol{z}$ for our model. This improves our results on MNIST – likely due to an explicit signal for the model to predict the correct set size – but again, we do not use this to keep our model as simple as possible.

E.2 CLEVR

We train our model and the baseline model for 100 epochs on the training set of CLEVR and evaluate on the validation set, since no ground-truth scene information is available for the test set. The set encoder is a 2-layer Relation Network with ReLU activation between the two layers, wherein the sum pooling is replaced with FSPool. The two layers have 512 neurons each. Because we use the Hungarian loss instead of the Chamfer loss here, including the mask feature in the target set does not worsen results, so we include the mask target for both the baseline and our model for consistency. To tune the learning rate, we started with the learning rate found for MNIST and decreased it similarly-sized steps until the training accuracy after 100 epochs worsened. We settled on 0.0003 as learning rate for both the bounding box and the state prediction task. All other hyperparameters are kept the same as for MNIST. The ResNet-18 that encodes the image is not pre-trained.

F Additional outputs

Figure 3: Progression of set prediction algorithm on MNIST.

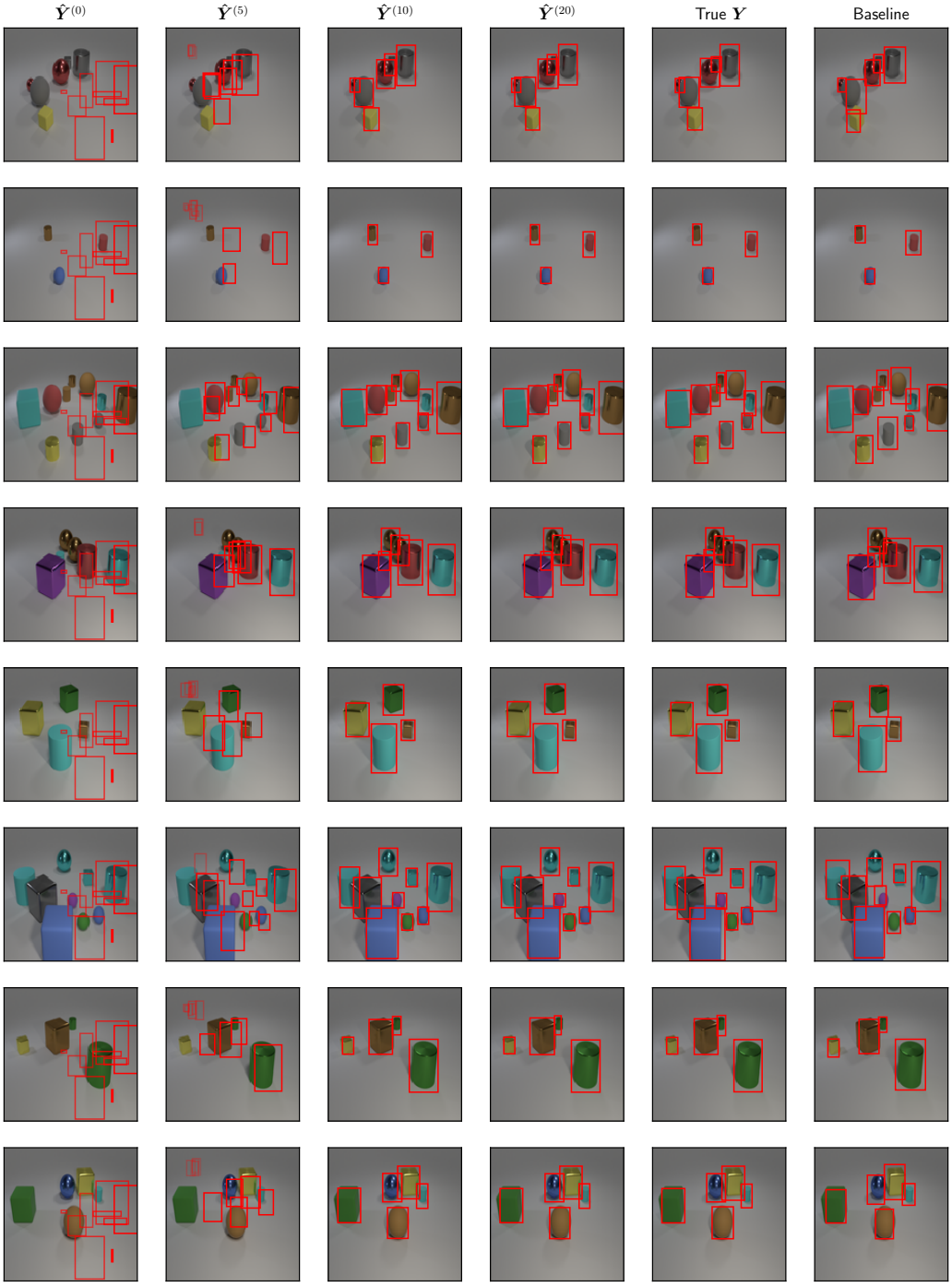
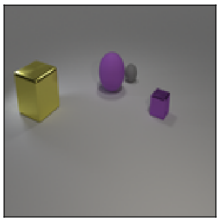
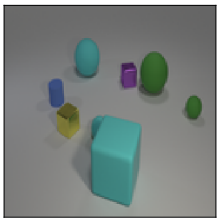


Figure 4: Progression of set prediction algorithm on CLEVR bounding boxes.

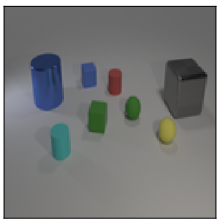
Table 3: Progression of set prediction algorithm on CLEVR state prediction. Red text denotes a wrong attribute. Objects are sorted by x coordinate, so they are sometimes misaligned with wrongly-coloured red text (see third example: red entries in $\hat{\boldsymbol{Y}}^{(20)}$ and bottom two red entries in baseline).



$\hat{\boldsymbol{Y}}^{(5)}$	$\hat{\boldsymbol{Y}}^{(10)}$	$\hat{\boldsymbol{Y}}^{(20)}$	True $\boldsymbol{Y}$	Baseline
(-0.14, 1.16, 3.57)	(-2.33, -2.41, 0.73)	(-2.33, -2.42, 0.78)	(-2.42, -2.40, 0.70)	(-1.65, -2.85, 0.69)
large purple rubber sphere	large yellow metal cube	large yellow metal cube	large yellow metal cube	large yellow metal cube
(0.01, 0.12, 3.42)	(-1.20, 1.27, 0.67)	(-1.21, 1.20, 0.65)	(-1.18, 1.25, 0.70)	(-0.95, 1.08, 0.68)
large gray metal cube	large purple rubber sphere	large purple rubber sphere	large purple rubber sphere	large green rubber sphere
(0.67, 0.65, 3.38)	(-0.96, 2.54, 0.36)	(-0.96, 2.59, 0.36)	(-1.02, 2.61, 0.35)	(-0.40, 2.14, 0.35)
small purple metal cube	small gray rubber sphere	small gray rubber sphere	small gray rubber sphere	small red rubber sphere
(0.67, 1.14, 2.96)	(1.61, 1.57, 0.36)	(1.58, 1.62, 0.38)	(1.74, 1.53, 0.35)	(1.68, 1.77, 0.35)
small purple rubber sphere	small yellow metal cube	small purple metal cube	small purple metal cube	small brown metal cube



$\hat{\boldsymbol{Y}}^{(5)}$	$\hat{\boldsymbol{Y}}^{(10)}$	$\hat{\boldsymbol{Y}}^{(20)}$	True $\boldsymbol{Y}$	Baseline
(-0.29, 1.14, 3.73)	(-2.78, 0.86, 0.72)	(-2.62, 0.83, 0.68)	(-2.88, 0.78, 0.70)	(-2.42, 0.63, 0.71)
small purple metal cube	large cyan rubber sphere	large cyan rubber sphere	large cyan rubber sphere	large purple rubber sphere
(-0.11, -0.37, 3.65)	(-2.17, -1.59, 0.38)	(-2.12, -1.58, 0.49)	(-2.14, -1.63, 0.35)	(-2.40, -2.07, 0.35)
small brown metal cube	small blue rubber cylinder	small blue rubber cylinder	small blue rubber cylinder	small green rubber cylinder
(0.08, 0.56, 3.84)	(-0.45, 2.19, 0.40)	(-0.60, 2.23, 0.29)	(-0.78, 1.97, 0.35)	(-0.74, 2.46, 0.33)
large cyan rubber cube	small purple metal cube	small purple metal cube	small purple metal cube	small cyan metal cube
(0.69, -0.43, 3.55)	(-0.14, -2.15, 0.38)	(-0.30, -1.99, 0.32)	(-0.38, -2.06, 0.35)	(0.30, -1.86, 0.34)
small brown rubber sphere	small yellow metal cube	small yellow metal cube	small yellow metal cube	small gray rubber sphere
(1.12, 0.21, 3.83)	(0.53, 2.56, 0.70)	(0.27, 2.46, 0.72)	(0.42, 2.56, 0.70)	(0.69, -2.10, 0.36)
large cyan rubber cube	large green rubber sphere	large green rubber sphere	large green rubber sphere	small red metal cube
(1.23, -0.25, 3.58)	(0.93, -1.41, 0.35)	(0.86, -1.31, 0.27)	(0.81, -1.30, 0.35)	(1.12, 2.28, 0.70)
small cyan rubber sphere	small cyan rubber sphere	small cyan rubber sphere	small cyan rubber sphere	large cyan rubber sphere
(1.73, 1.04, 3.57)	(2.50, -2.08, 0.76)	(2.64, -2.05, 0.76)	(2.56, -1.94, 0.70)	(2.55, -2.26, 0.73)
small cyan rubber sphere	large cyan rubber cube	large cyan rubber cube	large cyan rubber cube	large yellow rubber cube
(2.06, 1.94, 3.81)	(2.61, 2.59, 0.33)	(2.75, 2.73, 0.35)	(2.74, 2.64, 0.35)	(2.99, 2.59, 0.35)
large brown rubber sphere	small green rubber sphere	small green rubber sphere	small green rubber sphere	small purple rubber sphere



$\hat{\boldsymbol{Y}}^{(5)}$	$\hat{\boldsymbol{Y}}^{(10)}$	$\hat{\boldsymbol{Y}}^{(20)}$	True $\boldsymbol{Y}$	Baseline
(0.22, 0.12, 3.47)	(-2.76, -1.42, 0.68)	(-2.68, -1.64, 0.77)	(-2.62, -1.76, 0.70)	(-2.47, -1.73, 0.70)
small brown rubber cube	large blue metal cylinder	large blue metal cylinder	large blue metal cylinder	large cyan metal cylinder
(0.41, 0.11, 3.77)	(-1.56, -0.61, 0.35)	(-2.43, 0.03, 0.34)	(-2.29, 0.49, 0.35)	(-2.42, 0.09, 0.36)
large gray metal cube	small blue rubber cylinder	small blue rubber cube	small blue rubber cube	small blue rubber cylinder
(0.50, 0.44, 3.61)	(-1.08, 0.23, 0.33)	(-1.00, 1.18, 0.33)	(-0.93, 1.15, 0.35)	(-1.24, 1.16, 0.36)
small gray rubber cube	small green rubber cube	small red rubber cylinder	small red rubber cylinder	small red rubber cube
(0.83, 0.53, 3.45)	(-0.07, 0.97, 0.36)	(-0.01, -1.00, 0.46)	(0.28, -2.84, 0.35)	(0.39, 0.20, 0.33)
small cyan rubber sphere	small green rubber cylinder	small green rubber cube	small cyan rubber cylinder	small red rubber sphere
(0.86, 0.85, 3.50)	(0.28, -2.44, 0.49)	(0.21, -2.88, 0.40)	(0.29, -0.98, 0.35)	(0.56, -3.11, 0.35)
small gray rubber sphere	small cyan rubber cylinder	small cyan rubber cylinder	small green rubber cube	small yellow rubber cylinder
(1.86, 2.34, 3.80)	(1.36, -0.63, 0.38)	(0.99, 0.17, 0.37)	(0.92, 0.54, 0.35)	(0.90, 0.64, 0.35)
large gray metal cube	small green rubber sphere	small green rubber sphere	small green rubber sphere	small green rubber sphere
(1.97, 0.55, 3.61)	(2.01, 3.07, 0.65)	(1.97, 2.89, 0.39)	(2.04, 2.78, 0.70)	(2.39, 0.27, 0.36)
small green rubber sphere	large gray metal cube	large gray metal cube	large gray metal cube	small yellow rubber sphere
	(2.69, 0.63, 0.34)	(2.87, 0.51, 0.25)	(2.70, 0.67, 0.35)	(2.44, 2.55, 0.68)
	small yellow rubber sphere	small yellow rubber sphere	small yellow rubber sphere	large gray metal cube