Unsupervised extraction of interpretable graph representations from multiple-object scenes

Duo Wang [1] **Mateja Jamnik** [1] **Pietro Lio** [1]

Abstract

In this work we present Discrete Attend Infer Repeat (Discrete-AIR), a model for extracting interpretable object-level representation in an unsupervised way. Discrete-AIR is a Recurrent Auto-Encoder with structured latent distributions containing discrete categorical distributions, continuous attribute distributions, and factorised spatial attention. We show that for efficient inference in the case of Multi-MNIST (Eslami et al., 2016) and Multi-Sprites (Matthey et al., 2017) datasets, the Discrete-AIR model needs just one categorical latent variable, one attribute variable (for Multi-MNIST only), together with spatial attention variables. We perform analysis to show that the learnt categorical distributions achieves 87.0% and 94.5% category correspondence rate for Multi-MNIST and for Multi-Sprites. We also discuss ways of constructing interpretable initial edge embedding from the object representations.

1. Introduction and related works

Many real-world tasks, such as inferring physical relationships between objects in an image and visual-spatial reasoning, requires identifying and learning a useful representation of elements in the scene and representations of relations between each elements. The elements in the scene are essentially nodes in a graph and the relationships between elements are edges. Whilst there are many proposed methods (Kipf & Welling, 2016; Veličković et al., 2017; Battaglia et al., 2018) in processing such graphs, there are few methods for extracting graph representations from raw input images. We propose Discrete-AIR, a model that extract fully interpretable object-level representations from input images without any supervision signal, which subsequently allows initializing edge representation with interpretable embeddings.

Our model builds upon Attend-Infer-Repeat (AIR) model by Eslami et al (Eslami et al., 2016), a recurrent-VAE developed to decompose a scene into multiple objects with each represented by latent code $z = (z_{what}, z_{where}, z_{pres})$. While this latent code disentangles spatial information z_{where} and object presence z_{pres}, the object representation z_{what} is an entangled real-valued vector in the end-to-end trainable setting and thus difficult to interpret. Discrete-AIR, an end-to-end trainable autoencoder which structures latent representation z_{what} into z_{cat} representing category of objects and z_{attr} representing attributes of objects. With this disentangled node-level representations, we can readily compute initial edge embeddings in the form of distances between different parts of latent variables.

Related to this work are other approaches which decompose scenes into different categories. Neural Expectation Maximization (NEM) by Greff et al (Greff et al., 2017) implemented Expectation-Maximization algorithm with an end-to-end trainable neural network. NEM is able to perceptually group pixels of an image into different clusters. However, it does not learn a generative model that allows controllable generation like using z_{cat} and z_{where} in Discrete-AIR, and does not learn disentangled latent variables in the generative model setting. Ganin et al (Ganin et al., 2018) train a neural network to synthesize programs that can be fed into a graphics engine to generate scenes. While it learns an inference model for the generative model, a graphics engine that can provide learning gradients is pre-defined and not learnt. In contrast, Discrete-AIR jointly learns an inference model and a generative model from scratch.

We test Discrete-AIR model on two multi-object datasets, namely Multi-MNIST dataset as used in the original AIR model (Eslami et al., 2016) and a multi-object dataset in similar style as the dSprites dataset (Matthey et al., 2017). We show that unsupervised training of Discrete-AIR model is able to effectively capture the categories of objects in the scene. We also propose a method of constructing meaningful edge embeddings between node representations.

[1]Department of Computer Science and Technology, University of Cambridge, Cambridge, UK. Correspondence to: Duo Wang <duo.wang@cl.cam.ac.uk>.

Proceedings of the 36^{th} International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019.

2. Discrete Attend Infer Repeat

Attend-Infer-Repeat (AIR) model, introduced by Eslami et al (Eslami et al., 2016), is a recurrent version of Variational Auto-Encoder (VAE) (Kingma & Welling, 2013) that decomposes a scene into multiple objects represented by latent code $z^i = (z^i_{what}, z^i_{where}, z^i_{pres})$ at each recurrent time step i. For details of AIR model we refer readers to Appendix A. While the AIR model can encode objects in a scene into latent code z_{what}, the representation is still entangled and therefore not interpretable. In Discrete-AIR, we introduce structure into the latent distribution to encourage disentanglement. We break z_{what} into z_{cat} and z_{attr}. z_{cat} is discrete latent variable that captures the category of the object, while z_{attr} is a combination of continuous and discrete latent variables that captures attributes of the object. We do not use any objective function to encourage z_{cat} to capture category and z_{attr} to capture attributes. Rather, we allow the model to automatically learn the best way of using these discrete and latent variables through the process of likelihood maximisation.

In Discrete-AIR, we treat binary discrete variables as scalar of 0/1 values and multi-class categorical discrete variables as one-hot vectors. As sampling from a discrete distribution is non-differentiable, we model discrete latent variables with Gumbel Softmax (Maddison et al., 2016; Jang et al., 2016), a continuous approximation to the discrete distribution from which we can sample approximately one-hot discrete vectors. We refer readers to Appendix B for detailed discussion of Gumbel Softmax.

2.1. Generative model

The probabilistic generative model is shown in Figure 1. From z_{cat}, a template $T_{z_{cat}}$ of this object category is generated. This template is then modified by attributes z_{attr} into an image of object o that is subsequently drawn onto the canvas using spatial write attention. z_{cat}, z_{attr}, z_{where} and z_{pres}, jointly as $\boldsymbol{z}_t$, are estimated from the inference model for each time step t of inference.

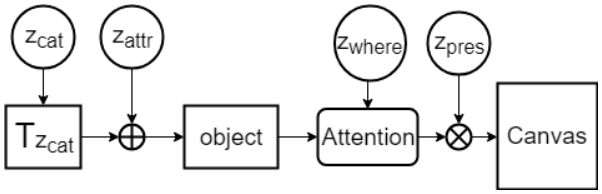


Figure 1. Generative Model of Discrete-AIR.

We replace the decoder function $f_{dec}(z^i_{what})$ from the original AIR model with a new function $f_{dec}(z^i_{cat}, z^i_{attr})$ parametrized by category variable z_{cat} and z_{attr}. There are various candidate functions for combining z_{cat} and z_{attr}. We have experimented with three different variations and found that additive function $f(f_t(z^i_{cat}) + f_a(z^i_{attr}))$ or multiplicative function $f(f_t(z^i_{cat}) \odot f_a(z^i_{attr}))$ works similarly well.

In the original AIR model, the spatial transformation operation specified by attention variable z_{where} only contains translation and scaling. Affine transformations such as rotation and shearing are accounted for in the latent variable in an entangled way. In Discrete-AIR, we explicitly introduce additional spatial transformer networks that account for rotation and skewing, thereby allowing z_{attr} to have a reduced number of variables. The spatial attention for the generative decoder is thus factorised as in Equation 1:

$$\boldsymbol{T}^d = \boldsymbol{T}^d_{st}\boldsymbol{T}^d_r\boldsymbol{T}^d_k = \tag{1}$$
$$\begin{bmatrix} s_x & 0 & t_x \\ 0 & s_y & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} cos(\omega) & sin(-\omega) & 0 \\ sin(\omega) & cos(\omega) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1+k_xk_y & k_x & 0 \\ k_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

where $\boldsymbol{T}^d_{st}$ is the combined transformation matrix of translation and scaling used in the original AIR model, $\boldsymbol{T}_r$ is the transformation matrix for rotation and $\boldsymbol{T}_k$ is the transformation matrix for skewing. In the matrix, s_x and s_y are for scaling, t_x and t_y are horizontal and vertical translations, ω is an angle of rotation, k_x and k_y are parameters for shearing in horizontal and vertical axis.

2.2. Inference

Figure 2 shows an overview of the Discrete-AIR architecture. At inference step t, a difference image between input image D and previous canvas C_{t-1} is fed together with previous latent code $\boldsymbol{z}_{t-1}$ into a Recurrent Neural Network (RNN), implemented as Long Short-Term Memory (LSTM) (Hochreiter & Schmidhuber, 1997) to generate parameters of the distribution for z_{where} and z_{pres}. Spatial Attention module then attends to parts of the image and applies transformation according to z_{where}. We enforce the encoding transformation $\boldsymbol{T}^e$ to be the inverse of decoding transformation $\boldsymbol{T}^d$, which means $\boldsymbol{T}^e\boldsymbol{T}^d = \boldsymbol{I}$. This constraint forces the model to match attended objects in the scene with the invariant template specified by z_{cat}. In practice, we compute $\boldsymbol{T}^e$ as the product of inverses of the transformation matrices composing $\boldsymbol{T}^d$: $\boldsymbol{T}^e = \boldsymbol{T}^{d^{-1}} = \boldsymbol{T}^{d^{-1}}_k\boldsymbol{T}^{d^{-1}}_r\boldsymbol{T}^{d^{-1}}_{st}$

The transformed image is then processed by an encoder to estimate parameters of distributions for z_{cat} and z_{attr}. z_{cat} are sampled from Gumbel Softmax as discussed in Appendix B. z_{attr} can be sampled from any distribution that is suitable for the paradigm of tasks. For tasks presented, continuous variables such as the colour intensity or part deformation of an object can be sampled from a multivariate Gaussian distribution using the Re-parameterisation trick (Kingma & Welling, 2013), which allows gradient to pass through the originally un-differentiable sampling function. The generative model described in Section 2.1 then

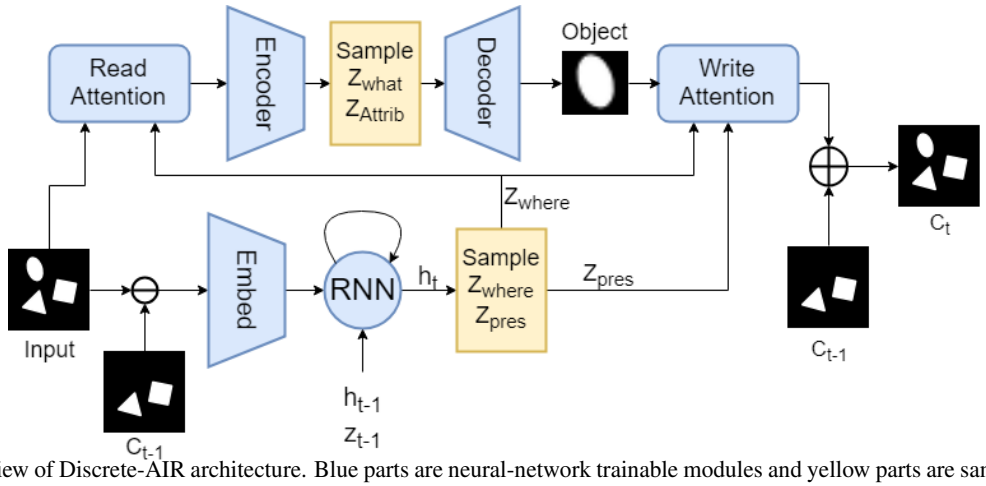


Figure 2. Overview of Discrete-AIR architecture. Blue parts are neural-network trainable modules and yellow parts are sampling processes.

samples z_{cat} and z_{attr} from the distributions in order to generate an object that will be written to canvas C_t using spatial attention module.

2.3. Learning

Similar to the original AIR model, we train Discrete-AIR model end-to-end by maximizing the lower bound on the marginal likelihood of data:

$$log\, p_\theta(x) \leq \mathcal{L}(\theta, \phi) = \mathbb{E}_{q_\phi}\Big[\log \frac{p_\theta(x, z, n)}{q_\phi(z, n|x)}\Big] \quad (2)$$

In the original AIR model, one cannot further rearrange this equation due to undifferentiable discrete variable sampling process used. For Discrete-AIR, by using Gubmel-Softmax as a reparameterised sampling process, we can rearrange Equation 2 as $\mathcal{L} = \mathbb{E}_{q_\phi}\Big[\log p_\theta(x|z, n)\Big] - D_{KL}(q_\phi(z, n|x)||p(z, n))$ where $p_\theta(x|z, n)$ is data likelihood and D_{KL} is Kullback-Leibler (KL) divergence, same as the original VAE (Kingma & Welling, 2013). Computing $\frac{\partial \mathcal{L}}{\partial \theta}$, the loss derivative with respect to parameters of the generative model, is relatively straightforward as it is fully differentiable. With a sampled batch of latent codes $z = (z_{cat}, z_{attr}, z_{where}, z_{pres}) \sim q(\cdot|x)$, the partial derivative $\frac{\partial}{\partial \theta} p_\theta(x|z, n)$ can be directly computed.

When computing $\frac{\partial \mathcal{L}}{\partial \phi}$, we can use the re-parametrisation trick (Kingma & Welling, 2013) to re-parametrise the sampling of both, continuous and discrete latent variables as a deterministic function in the form $h(\omega^i, \epsilon^i)$. ω^i is the parameters of the distributions for z at time step i, and ϵ^i are random noise at time step i. In this way we can use the chain rule to compute the gradient with respect to ϕ as $\frac{\partial \mathcal{L}}{\partial \phi} = \frac{\partial \mathcal{L}}{\partial h} \times \frac{\partial h}{\partial \omega^i} \times \frac{\partial \omega^i}{\partial \phi}$.

For our experiments, we parametrise continuous variables as multivariate Gaussian distributions with a diagonal covariance matrix and discrete variables as use Gumbel-Softmax distribution, which is itself a re-parametrised differentiable sampling function. For more details, please refer to Appendix C.2.

2.4. Edge Embedding

The output of Discrete-AIR is a list of object-level embeddings $z = (z_{cat}, z_{attr}, z_{where}, z_{pres})$. We can build a graph representation by firstly instantiating nodes for embeddings with $z_{pres} = 1$ and subsequently instantiating embeddings of edges by computing distances between sub-variables of z. For details, please refer to Appendix C.3. For illustration, we embed z_{where} distance as $z^i_{where} - z^j_{where}$ with an edge direction indicator $d_{i,j}$. We use L2 distance for z_{attr} and equality function $\mathbf{1}(z^i = z^j)$ for z_{cat}.

3. Evaluation

We evaluate Discrete-AIR on two multi-object datasets, namely Multi-MNIST dataset as used in the original AIR model (Eslami et al., 2016) and a multi-object shape dataset comprising of simple shapes similar to dSprites dataset (Matthey et al., 2017). We perform experiments to show that Discrete-AIR, while retaining the original strength of the AIR model of discovering the number of objects in a scene, can additionally categorise each discovered object. In order to evaluate how accurately can Discrete-AIR categorise each object, we compute the correspondence rate between the best permutation of category assignments from Discrete-AIR model and the true labels of the dataset.

We briefly explain the metric used for evaluating category correspondence between Discrete-AIR assigned categories and the ground-truth categories. For a more detailed discussion, please see Appendix D. We define a function $f_p(C, p)$ where C is a set or array of sets, and p is an index permutation function to map elements in C. For the whole dataset, we have an array of predicted category set O and an array of true label set T. We define correspondence rate as $in(O, T)/size(T)$ where $in(O, T)$ gives the number of true

labels t in T that are correctly identified in O. $size(T)$ gives the total number of labels. We thus compute the best correspondence rate as $R_{corr} = \max_{p \in P} \frac{in(f_p(O,p),T)}{size(T)}$, where P is the set of all possible permutations of predicted categories. This score is ranging from 0 to 1, and the score of a random category assignment should have the expected score of $1/k$ where k is the number of categories.

3.1. Multi-Sprites

To evaluate Discrete-AIR, we have built a multi-object dataset in similar style as the dSprites dataset (Matthey et al., 2017). This dataset consists of 90000 images of pixel sizes 64×64. In each image there are 0 to 3 objects with shapes in the categories of square, triangle and ellipse. The objects' spatial locations, orientations and size are all sampled randomly from uniform distributions. Details about constructing this dataset can be found in Appendix B. Figure 3 (a) illustrates the application of Discrete-AIR on the Multi-Sprites dataset. The left and middle figures show samples of input data from the dataset with each object detected and categorised (with differently coloured bounding box) and reconstructed images by the Discrete-AIR model. The number at the top-left corner shows the estimated number of objects in the scene. The right graph illustration shows interpretable edge embedding between each object in the scene. For interpretable node embedding please refer to Multi-MNIST illustration in Figure 3 (b). For this dataset, we used a discrete variable of 3 categories as z_{cat} together with spatial attention variables z_{where}. We did not include z_{attr} for this dataset as the attributes of each object, including location, orientation and size, can all be controlled by z_{where}. We did not include shear transformation $\boldsymbol{T}_k^d$ in the spatial attention as the dataset generation process does not have a shear transformation. For more details about the architecture, please see Appendix A. For quantitative evaluation of Discrete-AIR we use two metrics, count accuracy of number of objects in the scene and categorical correspondence rate. We also compare Discrete-AIR with AIR for the first two metrics. Table 1 shows the performance for these three objectives. We report mean performance across 10 independent runs. Discrete-AIR has slightly better count accuracy than AIR, and is able to categorise objects with a mean category correspondence rate of 0.945. The best achieved correspondence rate is 0.967.

Model	Multi-Sprites		Multi-MNIST	
	count acc.	cat. corr.	count acc.	cat. corr.
D-AIR	0.985	0.945	0.984	0.87
AIR	0.981	N/A	0.985	N/A

Table 1. Quantitative evaluation of Discrete-AIR (D-AIR) and comparison with AIR model.

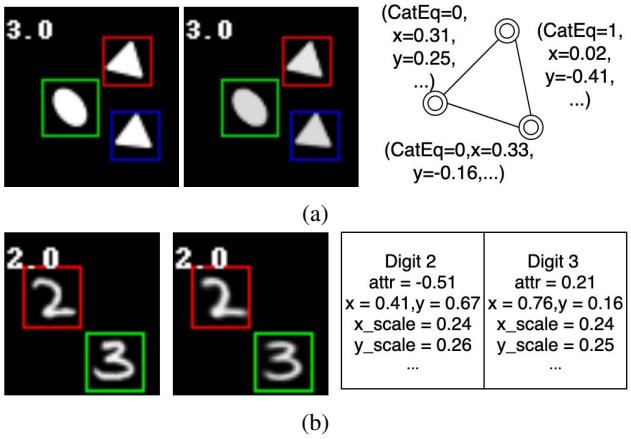


Figure 3. Input data, reconstruction and learned latent codes from Discrete-AIR model for (a) Multi-Sprites and (b) Multi-MNIST datasets. The coloured bounding boxes show each detected object. The number at the top-left corner shows the count of number of objects in the image. (a) shows interpretable edge embeddings.'CatEq' indicates equality in category. (b) shows node embeddings. 'attr' shows z_{attr} value.

3.2. Multi-MNIST

We also evaluated Discrete-AIR on the Multi-MNIST dataset used by the original AIR model (Eslami et al., 2016). The dataset consists of 60000 images of size 50×50. Each image contains 0 to 2 digits sampled randomly from MNIST dataset (LeCun et al., 1998) and placed at random spatial positions. The dataset is publicly available in 'observations' python package[1]. For this dataset, we choose a categorical variable with 10 categories as z_{cat} and 1 continuous variable with Normal distribution as z_{attr} as this gives best correspondence rate performance. We choose to combine transformation matrices T_r and T_k as one because this gives slightly better results. Figure 3 (b) shows sampled input data from the dataset and reconstruction by Discrete-AIR, and interpretable latent codes for each digit in the image. From this figure we can observe clearly that Discrete-AIR learns to match templates of category z_{cat} with modifiable attributes z_{attr} to input data. For example, in the reconstructed image of Figure 3 (b), we can see that Discrete-AIR picks a template of digit '3' and modifies attribute variables to fit to the style of '3' in the input image.

We also performed the same quantitative analysis from Multi-Sprites dataset, as shown in Table 1. While the count accuracy of Discrete-AIR and AIR model are very close, Discrete-AIR is able to categorise the digits in the image with a mean correspondence rate of 0.871. The best achieved correspondence rate is 0.913.

[1] https://github.com/edwardlib/observations

References

Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

Eslami, S. A., Heess, N., Weber, T., Tassa, Y., Szepesvari, D., Hinton, G. E., et al. Attend, infer, repeat: Fast scene understanding with generative models. In *Advances in Neural Information Processing Systems*, pp. 3225–3233, 2016.

Ganin, Y., Kulkarni, T., Babuschkin, I., Eslami, S., and Vinyals, O. Synthesizing programs for images using reinforced adversarial learning. *arXiv preprint arXiv:1804.01118*, 2018.

Greff, K., van Steenkiste, S., and Schmidhuber, J. Neural expectation maximization. In *Advances in Neural Information Processing Systems*, pp. 6691–6701, 2017.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Jaderberg, M., Simonyan, K., Zisserman, A., et al. Spatial transformer networks. In *Advances in neural information processing systems*, pp. 2017–2025, 2015.

Jang, E., Gu, S., and Poole, B. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Maddison, C. J., Mnih, A., and Teh, Y. W. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

Matthey, L., Higgins, I., Hassabis, D., and Lerchner, A. dsprites: Disentanglement testing sprites dataset. https://github.com/deepmind/dsprites-dataset/, 2017.

Mnih, A. and Gregor, K. Neural variational inference and learning in belief networks. In *International Conference on Machine Learning*, pp. 1791–1799, 2014.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

A. Details of Attend-Infer-Repeat Model

Attend-Infer-Repeat (AIR) model, introduced by Eslami et al (Eslami et al., 2016), is a recurrent version of Variational Auto-Encoder (VAE) (Kingma & Welling, 2013) that decomposes a scene into multiple objects represented by latent code $z^i = (z^i_{what}, z^i_{where}, z^i_{pres})$ at each recurrent time step i. Among them z^i_{pres} is a binary discrete variable encoding whether an object is inferred in current step i. If z^i_{pres} is 0, the inference will be stopped. The sequence of z^i_{pres} for all i can be concatenated into a vector of n ones and a final zero. n therefore is a variable representing the number of objects in the scene. z^i_{where} is a spatial attention parameter used to locate a target object in the image, and z^i_{what} is the latent code of the target object. In AIR an amortised variational approximation $q_\phi(z|x)$, as computed in equation 3, is used to approximate true posterior $p(z|x)$ by minimizing KL divergence $KL[q_\phi(z|x)||(z|x)]$. In AIR implementation, z_{what} and z_{where} are parametrised as Gaussian distributions with diagonal covariance $\mathcal{N}(\mu, \Sigma)$.

$$q_\phi(z|x) = q(z^{n+1}_{pres}|z^{1:n}, x)\prod_{i=1}^{n} q_\phi(z^i, z^i_{pres} = 1|x, z^{1:i-1}) \tag{3}$$

In the generative model of AIR, the number of objects n can be sampled from a prior such as geometric prior, and then form the sequence of z^i_{pres}. Next, z^i_{what} and z^i_{where} are sampled from $N(0, \boldsymbol{I})$. An object o^i is generated by processing z^i_{what} through a decoder. o^i is then written to the canvas, gated by z^i_{pres} and with scaling and translation specified by z^i_{where} using Spatial Transformer (Jaderberg et al., 2015), a powerful spatial attention module. The generative model can be summarised in equation 4, where f_{dec} is the decoder, ST is the spatial transformer and $\odot$ is element-wise product.

$$p_\theta(x|z) = \mathcal{N}(x|y, \sigma_x I) \tag{4}$$

$$y = \sum_{i=1}^{n} ST(f_{dec}(z^i_{what}), z^i_{where}) \odot z^i_{pres} \tag{5}$$

Inference and generative models of AIR are jointly optimized by maximizing the lower bound $\mathcal{L}(q_\phi, p_\theta) = E_{q_\phi}[\log \frac{p_\theta(x,z,n)}{q_\phi(z,n|x)}]$. While sampling operation of z is not differentiable (which is a requirement for gradient-based training), there are various ways to circumvent this. For the continuous latent codes, re-parametrization trick for VAE (Kingma & Welling, 2013) is applied, which lets parameters estimated from the inference model to deterministically modify a sampled distribution, thereby allowing back-propagation through the deterministic function. For discrete latent codes, AIR uses NVIL likelihood ratio estimator introduced by Mnih et al (Mnih & Gregor, 2014) to produce an unbiased estimate of the gradient for discrete latent variables.

B. Details of Gumbel Softmax

As sampling from a discrete distribution is non-differentiable, we model discrete latent variables with Gumbel Softmax (Maddison et al., 2016; Jang et al., 2016), a continuous approximation to the discrete distribution from which we can sample approximately one-hot discrete vector y where:

$$y_i = \frac{exp(\frac{log\, a_i + g_i}{\tau})}{\sum_{j=1}^{k} exp(\frac{log\, a_i + g_i}{\tau})} \quad ; i = 1, \ldots, k \tag{6}$$

a_i are a parametrization of the distribution, g_i are Gumbel noise sampled from the Gumbel distribution $Gumbel(0, 1)$, and τ is temperature parameters controlling smoothness of the distribution. As $\tau \to 0$, the distribution converges to a discrete distribution. For binary discrete variables such as z_{pres}, we use Gumbel Sigmoid, which is essentially Gumbel softmax with softmax function replaced with Sigmoid function:

$$y = \frac{exp(\frac{log\, a + g}{\tau})}{1 + exp(\frac{log\, a + g}{\tau})} \tag{7}$$

In contrast to the NVIL estimator (Mnih & Gregor, 2014) used in the original AIR model, we found that Gumbel softmax/Sigmoid is more stable during training, experiencing no model collapse during all the training experiments.

C. Details of Architecture and Training

We train Discrete-AIR with the ELBO objective as presented in equation 2. We use Adam optimiser (Kingma & Ba, 2014) to optimise the model with batch size of 64 and learning rate of 0.0001. For Gumbel Softmax, we also applied temperature annealing (Jang et al., 2016) of tau to start with a smoother distribution first and gradually approximate to discrete distribution. For more details about training, please see Appendix A in supplementary material.

C.1. Architecture

We use PyTorch package[2] for Python to build the neural network model. We describe below the details of each module in discrete-AIR.

Embedding: Embedding module embeds the difference image between input and previous canvas C^{t-1} into a feature vector which are input to the RNN module. For Multi-

[2] https://pytorch.org/

Sprites we use a Convolutional Neural Network as embedding module. The Conv-Net contains three convolutional layers with number of filters $16, 24, 32$. All layers have a kernel size of 5. Batch Normalization is used for all three layers. After each Convolutional layer a max-pooling layer of pool size 2×2 is applied. For Multi-MNIST dataset we did not use an embedding layer, same as original AIR model, but feed the difference image directly to RNN.

RNN: RNN module takes embedded difference image, together with latent codes of previous time step z^{t_1} as input, and generate spatial attention variables z^t_{where} and presence variable z^t_{pres} for the current step. The RNN is implemented as Long Short-Term Memory (LSTM) (Hochreiter & Schmidhuber, 1997). The LSTM module has 256 recurrent units. We perform gradient norm clipping to stabilize the training of RNN.

Read Attention: Read attention module takes input image and spatial attention parameters z_{where}. We use two consecutive Spatial Transformers (Jaderberg et al., 2015), one for scaling and translation and the other for rotation and shearing combined. For Multi-Sprites dataset we did not include shearing.

Encoder: Encoder takes the spatially attended image patch and encodes into a set of latent codes. This is implemented as a CNN of 3 layers with number of filters as $48, 64, 96$ and kernel size 5. Batch Normalization is used for all three layers. We did not use max-pooling layer but instead set the stride of each convolutional layer to be 2.

Decoder: Decoder decodes latent codes into an reconstructed image patch. We used additive function to combine z_{cat} and z_{attr} as discussed in the main text. The detailed architecture is shown in table 2. For Multi-Sprites dataset, no attribute variable is used.

z_{cat}	z_{attr}
Fully Connected (128 units)	
Fully Connected (1024 units)	
Resize 1024 to $64 \times 4 \times 4$	
Transposed Conv (64 to 48) Batch-Norm	
Transposed Conv (48 to 32) Batch-Norm	
Transposed Conv (32 to 1)	

Table 2. Decoder Architecture

Write Attention: Write Attention Module takes generated objects and z_{where} and write the object onto the canvas with two Spatial transformers with inverse transformation matrices of those in Read Attention Module.

Canvas function: We used an additive canvas function $C^t = C^{t-1} + O^t \otimes z^t_{pres}$ where O^t, the generated object is gated by the presence variable z_{pres}.

C.2. Latent variables

For the KL-divergence term. We use Gaussian prior for all continuous variables. While KL divergence between two Gumbel-Softmax distribution are not available in closed form, we approximate with a Monte-Carlo estimation of KL divergence with a categorical prior for z_{cat}, similar as (Jang et al., 2016). For z_{pres} we used a geometric prior and compute Monte-Carlo estimation of KL divergence (Maddison et al., 2016).

C.3. Edge Embedding

Here we describe how to initialize edge embedding with distances computed between nodes for different parts of the latent variable z. For Spatial variables z_{where} we can compute Euclidean L-2 distances with $D = (z^i_{where} - z^j_{where})^2$. For object attributes z_{attr} one can use any distance metrics in the continuous space. For z_{cat}, which is categorical variable representing categories of objects, different distance metrics might be useful for different tasks. For example the indicator function $\mathbf{1}(z^i_{cat}, z^i_{cat})$ maybe useful for logic reasoning. A customized distance matrix between each category pair may be useful for object query tasks such as visual question answering.

C.4. Training

Optimizer: ADAM
Learning Rate: 0.0001
Batch Size: 64
Temperature Annealing Scheme: $\tau = \max(0.5, e^{-rt})$ where t is number of training iterations and r is anneal rate. We set r to be 0.005 for both experiments.
Training epochs: We trained Multi-Sprites for 300 epochs and Multi-MNIST for 420 epochs. This is determined by reconstruction loss not improving for 10 consecutive epochs.

D. Metrics for category correspondence

To explain the metric we used, we first define a few notations. For each input image x_i, Discrete-AIR generates a corresponding category latent code z^i_{cat} and presence variable z^i_{pres}. From this we can form a set of predicted object categories $O^i = \{o^i_1, \ldots, o^i_n\}$ for n predicted objects where o^i_k is the k^{th} object category. For each image we also have a set of true labels of existing objects $T^i = \{t^i_1, ..t^i_m\}$. Due to non-identifiability problem of unsupervised learning where a simple permutation of best cluster assignment will give the same optimal result, the category assignments produced by Discrete-AIR do not necessarily correspond to the labels. For example, an image patch of digit 1 could fall into category 4. We thus permute the category assignments and use the permutation that corresponds best with the true label as the category assignment. For example, for predicted

category set $\{1, 4, 2, 2, 3\}$ and true label set $\{4, 0, 1, 1, 5\}$, we can use the following permutation of category for predicted category set $(1 \rightarrow 4, 4 \rightarrow 0, 2 \rightarrow 1, 3 \rightarrow 5)$ to achieve best correspondence. To put it more formally, we define a function $f_p(C, p)$ where C is a set or array of sets, and p is an index permutation function to map elements in C. For the whole dataset, we have an array of predicted category set O and an array of true label set T. We define correspondence rate as $in(O, T)/size(T)$ where $in(O, T)$ gives the number of true labels t in T that are correctly identified in O. $size(T)$ gives the total number of labels. We thus compute the best correspondence rate as $R_{corr} = \max_{p \in P} \frac{in(f_p(O,p),T)}{size(T)}$

where P is the set of all possible permutations of predicted categories. This score is ranging from 0 to 1, and the score of a random category assignment should have expected score of $1/k$ where k is the number of categories.

E. More examples of learnt latent code

F. Building Multi-Sprites Dataset

We built the Multi-Sprites dataset using the following pseudo code:

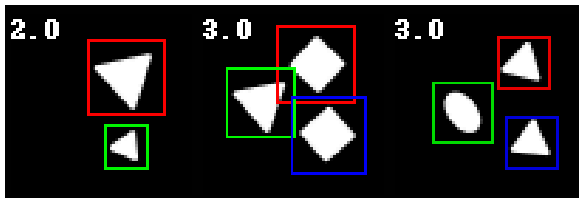
Algorithm 1 Bubble Sort

Input: num to generate N, num max objects M
for $i = 1$ **to** N **do**
  n_{obj} = RandomInt(M)
  List of objects L initalized
  for $j = 1$ **to** n_{obj} **do**
    $Category$ = RandomInt(num of classes)
    x = RandomUniform(x_{min},x_{max})
    y = RandomUniform(y_{min},y_{max})
    $angle$ = RandomUniform(-π,π)
    O = generate($Category, x, y, angle$)
    while Overlap(O,L) ¿ threshold **do**
      Repeat generation
    end while
    Append O in L
  end for
  Image = paint(L)
end for

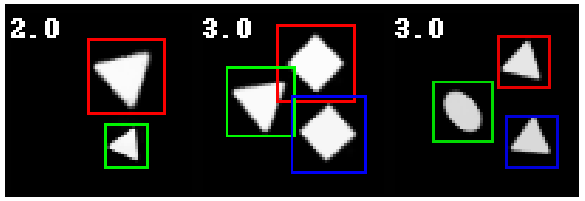
G. Analysis of failures

For Multi-Sprites dataset, the model perform less well when objects are too small. This is because small objects only occupies tens of pixels space and are thus heavily sub-sampled in the rasterization process. This causes the difference between different shapes to be indistinguishable.

For Multi-MNIST dataset, the model occasionally learn to use one category for two similar-looking digits by encoding the difference in the attribute variable, thereby causing considerable drop in correspondence rate. Figure 6 illustrates the case between digit 4 and 9 and digit 3 and 8.



(a) Data



(b) Reconstruction

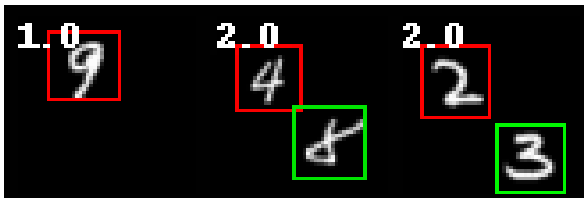
Triangle x = 0.71,y = 0.62 size = 0.31 angle = 0.66	Square x = 0.70,y = 0.69 size = 0.27 angle = 0.72	Triangle x = 0.73,y = 0.65 size = 0.23 angle = 2.31
Triangle x = 0.70,y = 0.23 size = 0.18 angle = -1.81	Triangle x = 0.32,y = 0.49 size = 0.29 angle = 0.71	Ellipse x = 0.41,y = 0.40 size = 0.25 angle = -0.34
	Square x = 0.73,y = 0.27 size = 0.27 angle = 0.74	Triangle x = 0.72,y = 0.24 size = 0.23 angle = 0.18

(c) Latent codes

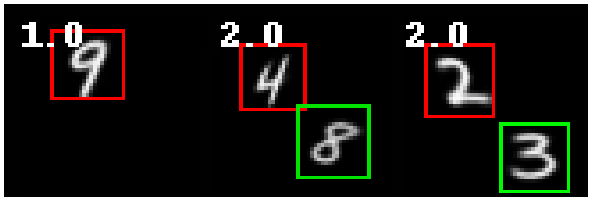
Figure 4. Input data from Multi-Sprites dataset and reconstruction from Discrete-AIR model. The coloured bounding boxes show each detected object. The number at the top-left corner shows the count of number of objects in the image. Latent codes representing the scene, including object categories, sizes, spatial locations and orientation are also presented.

H. Varying attribute variables

Figure 7 shows a sampled generated image. Two digits are generated in subsequent images with attribute variable increasing from top to bottom. In the first sequence we generate digits '5' and '2' while in the second sequence we generate digits '3' and '9'. We can observe that the learnt attribute variable z_{attr} encodes attributes that cannot be encoded by affine transformation spatial variable z_{where}. For example, increasing z_{attr} increases the size of the hook space in digit '5', the hook space in digit '2', and the hook curve in digit '9'.



(a) Data

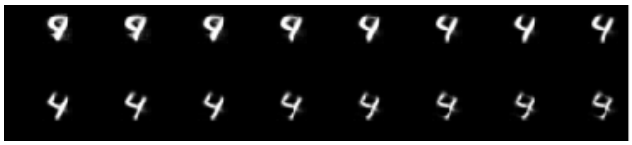


(b) Reconstruction

Digit 9 attr = 0.12 x = 0.38,y = 0.74 x_scale = 0.24 y_scale = 0.25 ...	Digit 4 attr = -0.11 x = 0.42,y = 0.69 x_scale = 0.20 y_scale = 0.21 ...	Digit 2 attr = -0.51 x = 0.41,y = 0.67 x_scale = 0.24 y_scale = 0.26 ...
	Digit 8 attr = 0.45 x = 0.72,y = 0.24 x_scale = 0.25 y_scale = 0.26 ...	Digit 3 attr = 0.21 x = 0.76,y = 0.16 x_scale = 0.24 y_scale = 0.25 ...

(c) Latent codes

Figure 5. Input data from Multi-MNIST dataset and reconstruction from Discrete-AIR model. The coloured bounding boxes show each detected object. The number at the top-left corner shows the count of number of objects in the image. Latent codes representing the scene, including digit categories, attribute variable value, sizes and spatial locations are also presented.



(a) 4 and 9

(b) 3 and 8

Figure 6. Illustration of Discrete-AIR model occasionally squeeze two digits into one category.

I. Some more reconstructions

Figure 8 shows additional samples of step-by-step reconstruction for Multi-MNIST dataset. Figure 9 shows this for Multi-Sprites dataset.

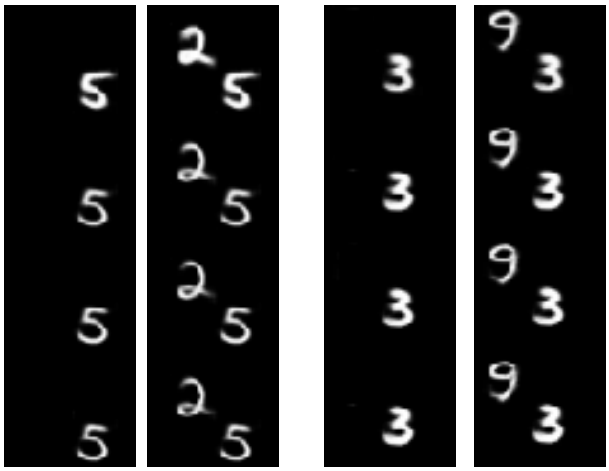
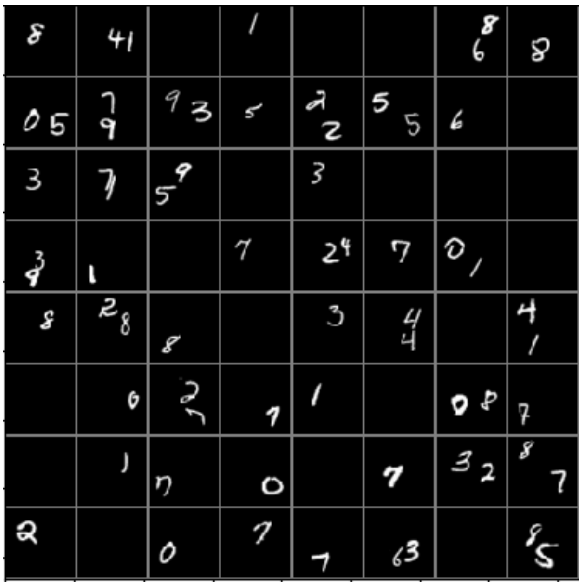
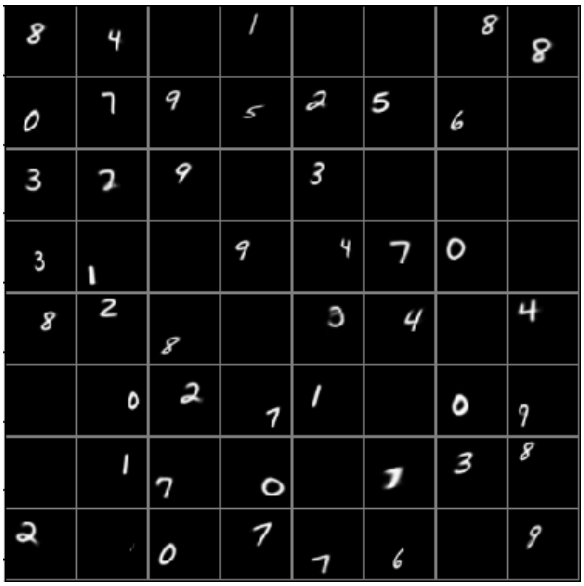


Figure 7. Generation of images by Discrete-AIR.



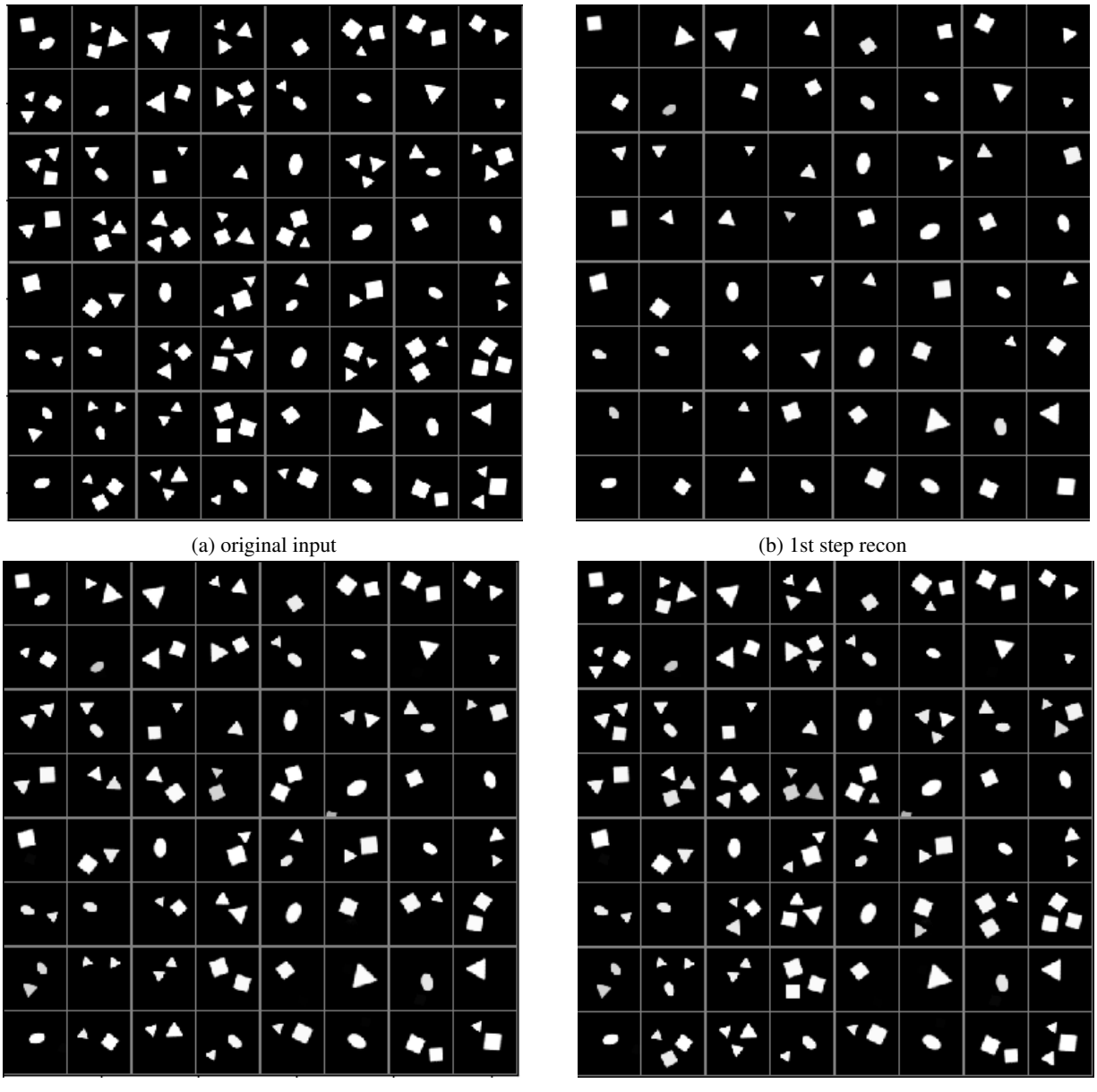
(a) original input



(b) 1st step recon

(c) 2nd step recon

Figure 8. Illustration of reconstruction for Multi-MNIST.



(a) original input (b) 1st step recon

(c) 2nd step recon (d) 3rd step recon

Figure 9. Illustration of reconstruction for Multi-Sprites.