LayoutTransformer: Layout Generation and Completion with Self-attention Supplementary Material

Kamal Gupta[1]*, **Justin Lazarow**[2], **Alessandro Achille**[3],
Larry Davis[1,3], **Vijay Mahadevan**[3], **Abhinav Shrivastava**[1]

[1]University of Maryland, College Park [2]University of California, San Diego [3]Amazon AWS

Appendix

A. Architecture and training details

In all our $\mathbb{R}^2$ experiments, our base model consists of $d = 512$, $L = 6$, $n_{\text{head}} = 8$, precision $= 8$ and $d_{\text{ff}} = 2048$. We also use a dropout of 0.1 at the end of each feedforward layer for regularization. We fix the the maximum number of elements in each of the datasets to 128 which covers over 99.9% of the layouts in each of the COCO, Rico and PubLayNet datasets. We also used Adam optimizer [3] with initial learning rate of 10^{-4}. We train our model for 30 epochs for each dataset with early stopping based on maximum log likelihood on validation layouts. Our COCO Bounding Boxes model takes about 6 hours to train on a single NVIDIA GTX1080 GPU. Batching matters a lot to improve the training speed. We want to have evenly divided batches, with minimal padding. We sort the layouts by the number of elements and search over this sorted list to use find tight batches for training.

In all our $\mathbb{R}^3$ experiments, we change $d = 128$ and $d_{\text{ff}} = 512$, and learning rate to 10^{-5}.

B. Ablation studies

We evaluate the importance of different model components with negative log-likelihood on COCO layouts. The ablation studies show the following:

Small, medium and large elements: NLL of our model for COCO large, medium, and small boxes is 2.4, 2.5, and 1.8 respectively. We observe that even though discretizing box coordinates introduces approximation errors, it later allows our model to be agnostic to large vs small objects.

Varying precision: Increasing it allows us to generate finer layouts but at the expense of a model with more parameters. Also, as we increase the precision, NLL increases, suggesting that we might need to train the model with more data to get similar performance (Table 1).

*Corresponding author, email: kampta@umd.edu
Work started during an internship at Amazon.

Table 1: Effect ofprecision on NLL

n_{anchors}	# params	COCO	Rico	PubLayNet
32×32	19.2	2.28	1.07	1.10
8×8	19.1	1.69	0.98	0.88
16×16	19.2	1.97	1.03	0.95
64×64	19.3	2.67	1.26	1.28
128×128	19.6	3.12	1.44	1.46

Size of embedding: Increasing the size of the embedding d improves the NLL, but at the cost of increased number of parameters (Table 2).

Table 2: Effect of d on NLL

d	# params	COCO	Rico	PubLayNet
512	19.2	2.28	1.07	1.10
32	0.8	2.51	1.56	1.26
64	1.7	2.43	1.40	1.19
128	3.6	2.37	1.29	1.57
256	8.1	2.32	1.20	1.56

Model depth: Increasing the depth of the model L, does not significantly improve the results (Table 3). We fix the $L = 6$ in all our experiments.

Table 3: Effect of L on NLL

L	# params	COCO	Rico	PubLayNet
6	19.2	2.28	1.07	1.10
2	6.6	2.31	1.18	1.13
4	12.9	2.30	1.12	1.07
8	25.5	2.28	1.11	1.07

Ordering of the elements: Adding position encoding,

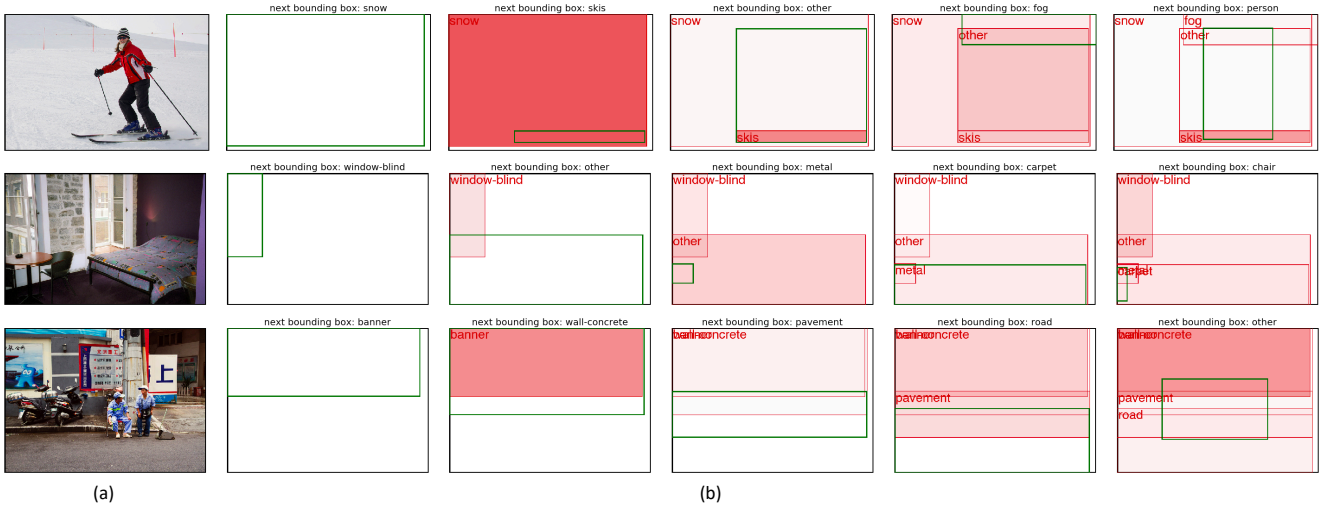


Figure 1: Visualizing attention. (a) Image source for the layout (b) In each row, the model is predicting one element at a time (shown in a green bounding box). While predicting that element, the model pays the most attention to previously predicted bounding boxes (in red). For example, in the first row, "snow" gets the highest attention score while predicting "skis". Similarly in the last column, "skis" get the highest attention while predicting "person".

Table 4: Effect of other hyperparameters on NLL

Order	Split-XY	Loss	# params	COCO	Rico	PubLayNet
raster	Yes	NLL	19.2	2.28	1.07	1.10
random			19.2	2.68	1.76	1.46
	No		21.2	3.74	2.12	1.87
		LS	19.2	1.96	0.88	0.88

makes the self-attention layer dependent to the ordering of elements. In order to make it depend less on the ordering of input elements, we take randomly permute the sequence. This also enables our model to be able to complete any partial layout. Since output is predicted sequentially, our model is not invariant to the order of output sequence also. In our experiments, we observed that predicting the elements in a simple raster scan order of their position improves the model performance both visually and in terms of negative log-likelihood. This is intuitive as filling the elements in a pre-defined order is an easier problem. We leave the task of optimal ordering of layout elements to generate layouts for future research. (Table 4).

Discretization strategy: Instead of the factorizing location in x-coordinates and y-coordinates, we tried predicting them at once (refer to the Split-xy column of Table 4). This increases the vocabulary size of the model (since we use $H \times H$ possible locations instead of H alone) and in turn the number of hyper-parameters with decline in model performance. An upside of this approach is that generating new layouts takes less time as we have to make half as many predictions for each element of the layout (Table 4).

Loss: We tried two different losses, label smoothing [7] and NLL. Although optimizing using NLL gives better validation performance in terms of NLL (as is expected), we do not find much difference in the qualitative performance when using either loss function. (Table 4)

C. Baselines

In this section we give more details on the various baselines that we implemented or modified from the author's original code. In all the ShapeNet baselines, we used the models provided by the authors or the original numbers provided in the paper. In all the two-dimensional baselines, we implemented the baseline from scratch in the case when code was not available (LayoutVAE) and use the author's own implementation when the code was available (LayoutGAN, ObjGAN, sg2im). Since we train our model in all cases, we ran a grid search over different hyperparameters and report the best numbers in the paper. We found that the GAN based methods were harder to converge and had higher variation in the outcomes as compared to non-adversarial approaches.

LayoutVAE. LayoutVAE [2] uses a similar representation for layout and consists of two separate autoregressive VAE models. Starting from a label set, which consists of categories of elements that will be present in a generated layout, their CountVAE generates counts of each of the elements of the label set. After that BoundingBoxVAE, generates the location and size of each occurrence of the bounding box. Note that the model assumes assumption of label set, and

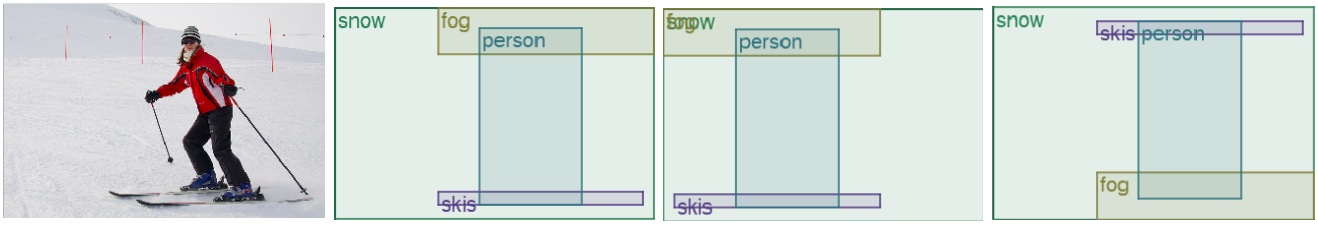


Figure 2: We observe the impact of operations such as left right flip, and up down flip on log likelihood of the layout. We observe that unlikely layouts (such as fog at the bottom of image have higher NLL than the layouts from data.

hence, while reporting we actually make the task easier for LayoutVAE by providing label-sets of layouts in the validation dataset.

ObjGAN. ObjGAN [5] provides an object-attention based GAN framework for text to image synthesis. An intermediate step in their image synthesis approach is to generate a bounding box layout given a sentence using a BiLSTM (trained independently). We adopt this step of the ObjGAN framework to our problem setup. Instead of sentences we provide categories of all layout elements as input to the ObjGAN and synthesize all the elements' bounding boxes. This in turn is similar to providing label set as input (similar to the case of LayoutVAE).

sg2im. Image generation from scene graph [1] attempts to generate complex scenes given scene graph of the image by first generating a layout of the scene using graph convolutions and then using the layout to generate complete scene using GANs. The system is trained in an end-to-end fashion. Since sg2im requires a scene graph input, following the approach of [1], we create a scene graph from the input and reproduce the input layout using the scene graph.

LayoutGAN. LayoutGAN [4] represents each layout with a fixed number of bounding boxes. Starting with bounding box coordinates sampled from a Gaussian distribution, its GAN based framework assigns new coordinates to each bounding box to resemble the layouts from given data. Optionally, it uses non-maximum suppression (NMS) to remove duplicates. The problem setup in LayoutGAN is similar to the proposed approach and they do not condition the generated layout on anything. Note that the authors didn't provide the code for the case of bounding boxes (but only for toy datasets used in the paper). In our implementation, we weren't able to reproduce similar results as the authors reported in the paper for documents.

D. Visualizing attention

The self-attention based approach proposed enables us to visualize which existing elements are being attending to while the model is generating a new element. This is demonstrated in Figure 1. We note that While predicting an element, the model pays the most attention to previously predicted bounding boxes (in red). For example, in the first row, "snow" gets the highest attention score while predicting "skis". Similarly in the last column, "skis" get the highest attention while predicting "person".

E. Layout Verification

Since in our method it is straightforward to compute likelihood of a layout, we can use our method to test if a given layout is likely or not. Figure 2 shows the NLL given by our model by doing left-right and top-down inversion of layouts in COCO (following [4]). In case of COCO, if we flip a layout left-right, we observe that layout remains likely, however flipping the layout upside decreases the likelihood (or increases the NLL of the layout). This is intuitive since it is unlikely to see fog in the bottom of an image, while skis on top of a person.

F. More semantics in learned category embeddings

Table 6 captures the most frequent bigrams and trigrams (categories that co-occur) in real and synthesized layouts. Table 5 shows word2vec [6] style analogies being captured by embeddings learned by our model. Note that the model was trained to generate layouts and we did not specify any additional objective function for analogical reasoning task.

Table 5: **Analogies**. We demonstrate linguistic nuances being captured by our category embeddings by attempting to solve word2vec [6] style analogies.

Analogy	Nearest neighbors
snowboard:snow::surfboard:?	waterdrops, sea, sand
car:road::train:?	railroad, platform, gravel
sky-other:clouds::playingfield:?	net, cage, wall-panel
mouse:keyboard::spoon:?	knife, fork, oven
fruit:table::flower:?	potted plant, mirror-stuff

Table 6: **Bigrams and trigrams**. We consider the most frequent pairs and triplets of (distinct) categories in real *vs.*generated layouts.

Real	Ours	Real	Ours
other person	other person	person other person	other person clothes
person other	person clothes	other person clothes	person clothes tie
person clothes	clothes tie	person handbag person	tree grass other
clothes person	grass other	person clothes person	grass other person
chair person	other dining table	person chair person	wall-concrete other person
person chair	tree grass	chair person chair	grass other cow
sky-other tree	wall-concrete other	person other clothes	tree other person
car person	person other	person backpack person	person clothes person
person handbag	sky-other tree	person car person	other dining table table
handbag person	clothes person	person skis person	person other person

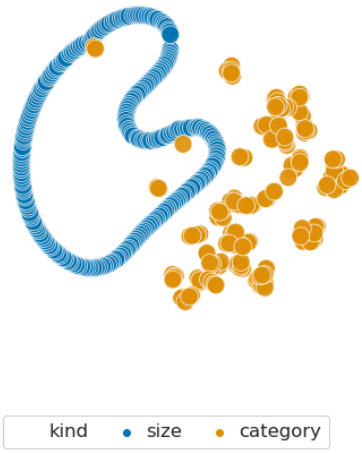


Figure 3: TSNE plot for dimension embedding (256 of them) and category embedding for COCO.

G. Coordinate Embedding

Just like in Fig. **??**, we project the embedding learned by our model on COCO in a 2-d space using TSNE. In the absence of explicit constraints on the learned embedding, the model learns to cluster together all the coordinate embedding in a distinct space, in a ring-like manner. Fig. 3 shows all the embeddings together in a single TSNE plot.

H. Nearest neighbors

To see if our model is memorizing the training dataset, we compute nearest neighbors of generated layouts using chamfer distance on top-left and bottom-right bounding box coordinates of layout elements. Figure 4 shows the nearest neighbors of some of the generated layouts from the training dataset. We note that nearest neighbor search for layouts is an active area of research.

I. More examples for Layout to Image

Layouts for natural scenes are cluttered and hard to qualitatively evaluate even for a trained user. Here we share some more sample layouts generated from two methods used in the paper. Figure 5 shows some extra sample layouts and corresponding images generated using Layout2Im tool. Existing layout to image methods don't work as well as free-form image generation methods but are arguably more beneficial in downstream applications. We hope that improving layout generation will aid the research community to develop better scene generation tools both in terms of diversity and quality.

J. Dataset Statistics

In this section, we share statistics of different elements and their categories in our dataset. In particular, we share the total number of occurrences of an element in the trai ning dataset (in descending order) and the total number of distinct layouts an element was present in throughout the training data. Tables 7, 7 show the statistics for Rico wireframes, and table 8 show the statistics for PubLayNet documents.

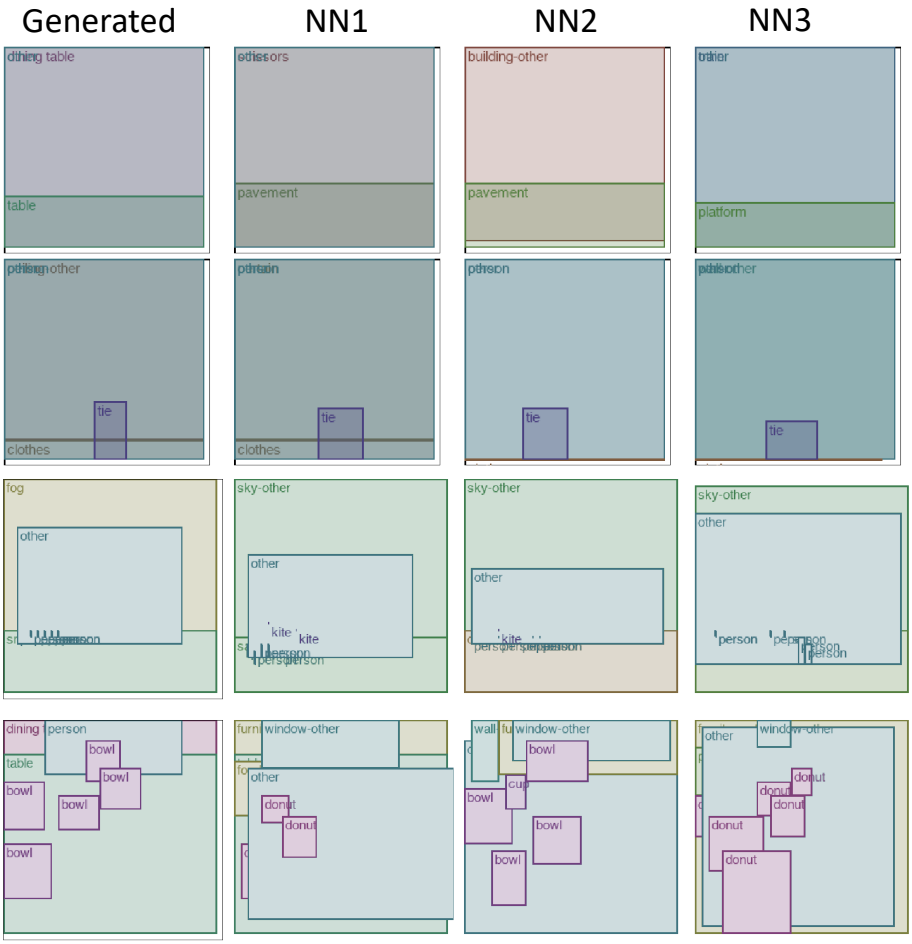
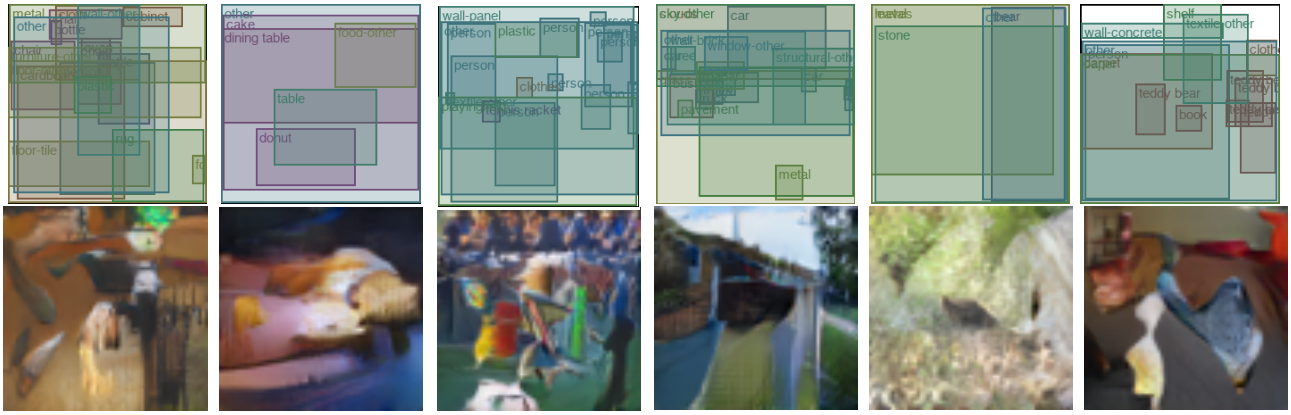


Figure 4: Nearest neighbors from training data. Column 1 shows samples generated by model. Column 2, 3, 4 show the 3 closest neighbors from training dataset. We use chamfer distance on bounding box coordinates to obtain the nearest neighbors from the dataset.



(a) LayoutVAE layouts (top) and images generated with Layout2Im (bottom)

(b) Our layouts (top) and images generated with Layout2Im (bottom)

Figure 5: Some sample layouts and corresponding images

Table 7: Category statistics for Rico

Category	# occurrences	# layouts
Text	387457	50322
Image	179956	38958
Icon	160817	43380
Text Button	118480	33908
List Item	72255	9620
Input	18514	8532
Card	12873	3775
Web View	10782	5808
Radio Button	4890	1258
Drawer	4138	4136
Checkbox	3734	1126
Advertisement	3695	3365
Modal	3248	3248
Pager Indicator	2041	1528
Slider	1619	954
On/Off Switch	1260	683
Button Bar	577	577
Toolbar	444	395
Number Stepper	369	147
Multi-Tab	284	275
Date Picker	230	217
Map View	186	94
Video	168	144
Bottom Navigation	75	27

Table 8: Category statistics for PubLayNet

Category	# occurrences	# layouts
text	2343356	334548
title	627125	255731
figure	109292	91968
table	102514	86460
list	80759	53049

References

[1] Justin Johnson, Agrim Gupta, and Li Fei-Fei. Image generation from scene graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1219–1228, 2018. 3

[2] Akash Abdu Jyothi, Thibaut Durand, Jiawei He, Leonid Sigal, and Greg Mori. Layoutvae: Stochastic scene layout generation from a label set. *arXiv preprint arXiv:1907.10719*, 2019. 2

[3] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 1

[4] Jianan Li, Jimei Yang, Aaron Hertzmann, Jianming Zhang, and Tingfa Xu. Layoutgan: Generating graphic layouts with wireframe discriminators. *arXiv preprint arXiv:1901.06767*, 2019. 3

[5] Wenbo Li, Pengchuan Zhang, Lei Zhang, Qiuyuan Huang, Xiaodong He, Siwei Lyu, and Jianfeng Gao. Object-driven text-to-image synthesis via adversarial training. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 12174–12182, 2019. 3

[6] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013. 3

[7] Rafael Müller, Simon Kornblith, and Geoffrey E. Hinton. When does label smoothing help? *CoRR*, abs/1906.02629, 2019. 2