Group Meeting - October 30, 2020

Paper review & Research progress

Truong Son Hy *

*Department of Computer Science
The University of Chicago

Ryerson Physical Lab

Homer (Iliad)

Any moment might be our last. Everything is more beautiful because we're doomed. You will never be lovelier than you are now. We will never be here again.

Papers

1. **Natural Graph Networks**, https://arxiv.org/abs/2007.08349
   This is CCN 1D(!?)
2. **SE(3)-Transformers: 3D Roto-Translation Equivariant Attention Networks**, https://arxiv.org/abs/2006.10503
   This is a special case of Cormorant with attention(!?)
3. **PointFlow: 3D Point Cloud Generation with Continuous Normalizing Flows** (ICCV 2019),
   https://arxiv.org/pdf/1906.12320.pdf
   Some ideas can be applied to graph/molecule generation.

Paper 2

SE(3)-Transformers: 3D Roto-Translation Equivariant Attention Networks

Fabian B. Fuchs, Daniel E. Worrall, Volker Fischer, Max Welling

https://arxiv.org/abs/2006.10503

Proposals

$SE(3)$-Transformer: a variant of the self-attention module for 3D point clouds, which is **equivariant** under continuous 3D roto-translations.

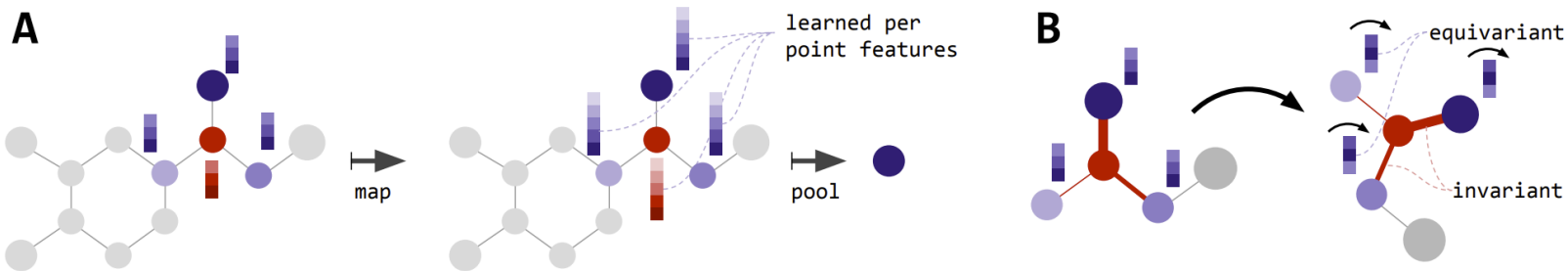


Figure 1: **A**) Each layer of the SE(3)-Transformer maps from a point cloud to a point cloud while guaranteeing euqivariance. For classification, this is followed by an invariant pooling layer and an MLP. **B**) In each layer, for each node, attention is performed. Here, the red node attends to its neighbours. Attention weights (indicated by line thickness) are invariant w.r.t. rotation of the input.

Background – The Attention Mechanism (1)

The attention mechanism:

$$\text{Attn}(\boldsymbol{q}_i, \{\boldsymbol{k}_j\}, \{\boldsymbol{v}_j\}) = \sum_{j=1}^{n} \alpha_{ij} \boldsymbol{v}_j$$

where:

$$\alpha_{ij} = \frac{\exp(\boldsymbol{q}_i^T \boldsymbol{k}_j)}{\sum_{j'=1}^{n} \exp(\boldsymbol{q}_i^T \boldsymbol{k}_{j'})}$$

- A set of query vectors: $\boldsymbol{q}_i \in \mathbb{R}^p$, $i = 1, .., m$.
- A set of key vectors: $\boldsymbol{k}_j \in \mathbb{R}^p$, $j = 1, .., n$.
- A set of value vectors: $\boldsymbol{v}_j \in \mathbb{R}^r$, $j = 1, .., n$.

Background – The Attention Mechanism (2)

Self-attention: Query, key, and value vectors are embeddings of the input features

$$\boldsymbol{q} = h_Q(\boldsymbol{f})$$

$$\boldsymbol{k} = h_K(\boldsymbol{f})$$

$$\boldsymbol{v} = h_V(\boldsymbol{f})$$

where $\{h_Q, h_K, h_V\}$ are, in the most general case, neural networks; and n coordinate vectors $\boldsymbol{x}_i \in \mathbb{R}^3$ with optional per-point features $\boldsymbol{f}_i \in \mathbb{R}^d$.

Note

- A key property of self-attention is **permutation equivariance**.
- **Intuition:** Is self-attention just a smarter way of doing **soft - thresholding**?

Background – Graph Neural Networks

Problem of attention

- Attention scales quadratically with point cloud size.
- $\rightarrow$ It is useful to introduce neighborhoods: instead of each point attending to **all** other points, it only attends to its **nearest neighbors**.

Non-local neural networks [Wang et al.] (CVPR 2017):

$$\boldsymbol{y}_i = \frac{1}{\mathcal{C}(\{\boldsymbol{f}_j \in \mathcal{N}_i\})} \sum_{j \in \mathcal{N}_i} w(\boldsymbol{f}_i, \boldsymbol{f}_j) h(\boldsymbol{f}_j)$$

- w and h are neural networks.
- $\mathcal{C}$ normalises the sum as a function of all features in the neighborhood $\mathcal{N}_i$.
- **Opinion:** This is just attention per neighborhood.

Background – Equivariance (1)

Equivariance

Given a set of transformations $T_g : \mathcal{V} \to \mathcal{V}$ for $g \in G$, where G is an abstract group, a function $\phi : \mathcal{V} \to \mathcal{Y}$ is called **equivariant** if for every g there exists a transformation $S_g : \mathcal{Y} \to \mathcal{Y}$ such that:

$$S_g[\phi(v)] = \phi(T_g[v])$$

Group representation

A group representation $\rho : G \to GL(N)$ to the set of $N \times N$ invertible matrices $GL(N)$. Group homomorphism: ρ satisfies the following properties:

$$\rho(g_1 g_2) = \rho(g_1)\rho(g_2)$$

for all $g_1, g_2 \in G$.

Background – Equivariance (2)

For 3D rotations $G = SO(3)$:

$$\rho(g) = \boldsymbol{Q}^T \left[\bigoplus_{\ell} \boldsymbol{D}_\ell(g) \right] \boldsymbol{Q}$$

- Each $\boldsymbol{D}_\ell$, for $\ell = 0, 1, 2, ..$ is a $(2\ell + 1) \times (2\ell + 1)$ Wigner-D matrix.
- $\bigoplus$ is the **direct sum** – concatentation of matrices along the diagonal.
- $\boldsymbol{Q}$ is an orthogonal $N \times N$ change-of-basis matrix.
- The Wigner-D matrices are **irreducible representations** of $SO(3)$.
  D = Darstellung (in German, means representation).

Background – Tensor Field Networks (1)

The type-ℓ output of the TFN layer at position $\boldsymbol{x}_i$:

$$\mathbf{f}_{\text{out},i}^{\ell} = \sum_{k\geq 0} \underbrace{\int \mathbf{W}^{\ell k}(\mathbf{x}' - \mathbf{x}_i)\mathbf{f}_{\text{in}}^{k}(\mathbf{x}')\,\mathrm{d}\mathbf{x}'}_{k\to\ell \text{ convolution}} = \sum_{k\geq 0}\sum_{j=1}^{n} \underbrace{\mathbf{W}^{\ell k}(\mathbf{x}_j - \mathbf{x}_i)\mathbf{f}_{\text{in},j}^{k}}_{\text{node } j \to \text{node } i \text{ message}}, \tag{7}$$

$$\mathbf{W}^{\ell k}(\mathbf{x}) = \sum_{J=|k-\ell|}^{k+\ell} \varphi_J^{\ell k}(\|\mathbf{x}\|)\mathbf{W}_J^{\ell k}(\mathbf{x}), \qquad \text{where } \mathbf{W}_J^{\ell k}(\mathbf{x}) = \sum_{m=-J}^{J} Y_{Jm}(\mathbf{x}/\|\mathbf{x}\|)\mathbf{Q}_{Jm}^{\ell k}. \tag{8}$$

- A learnable function $\varphi_J^{\ell k} : \mathbb{R}_{\geq 0} \to \mathbb{R}$ of the radius $||\boldsymbol{x}||$.
- $\boldsymbol{Q}$: Clebsch-Gordan matrices.
- Y: spherical harmonic.

Background – Tensor Field Networks (2)

If we include the **self-interaction**, the TFN layer can be written as:

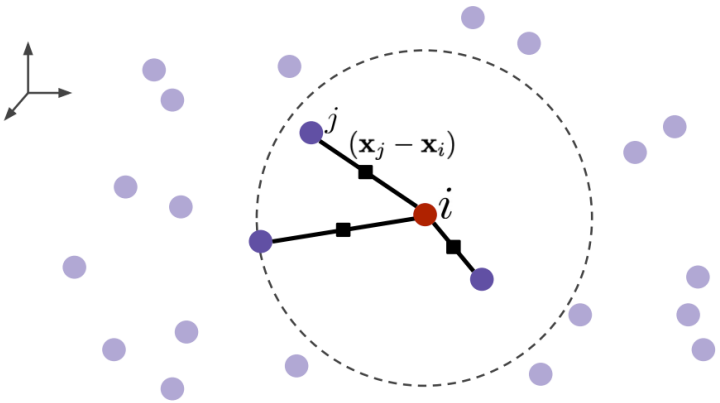
$$\mathbf{f}_{\text{out},i}^{\ell} = \underbrace{w^{\ell\ell}\mathbf{f}_{\text{in},i}^{\ell}}_{\text{self-interaction}} + \sum_{k\geq 0}\sum_{j\neq i}^{n} \mathbf{W}^{\ell k}(\mathbf{x}_j - \mathbf{x}_i)\mathbf{f}_{\text{in},j}^{k}, \tag{9}$$

Note

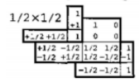
1. I am confused by this description.
2. The authors seemed to try to write it in the language of message passing.
3. In my opinion, it is indeed a special case of Cormorant. I think they should describe the Clebsh-Gordon decomposition.

SE(3)-Transformer (1)

Step 1: Get nearest neighbours and relative positions

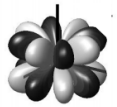


Step 2: Get SO(3)-equivariant weight matrices



Clebsch-Gordon Coeff.

Radial Neural Network



Spherical Harmonics

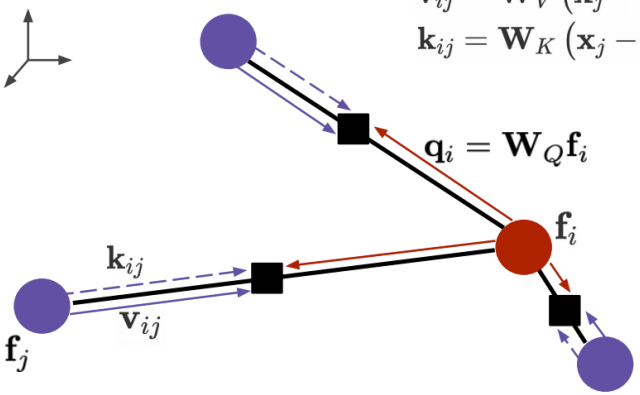
$$\mathbf{Q}_{Jm}^{\ell k} \qquad \varphi_J^{\ell k}(||x||) \qquad Y_{Jm}\left(\frac{x}{||x||}\right)$$

Matrix W consists of blocks mapping between degrees

$$\mathbf{W}(x) = \mathbf{W}\left(\left\{\mathbf{Q}_{Jm}^{\ell k}, \varphi_J^{\ell k}(||x||), Y_{Jm}\left(\frac{x}{||x||}\right)\right\}_{J,m,\ell,k}\right)$$

Step 3: Propagate queries, keys, and values to edges

$$\mathbf{v}_{ij} = \mathbf{W}_V\left(\mathbf{x}_j - \mathbf{x}_i\right)\mathbf{f}_j$$
$$\mathbf{k}_{ij} = \mathbf{W}_K\left(\mathbf{x}_j - \mathbf{x}_i\right)\mathbf{f}_j$$



Step 4: Compute attention and aggregate

$$\alpha_{ij} = \frac{\exp\left(\mathbf{q}_i^\top \mathbf{k}_{ij}\right)}{\sum_{j'} \exp\left(\mathbf{q}_i^\top \mathbf{k}_{ij'}\right)}$$

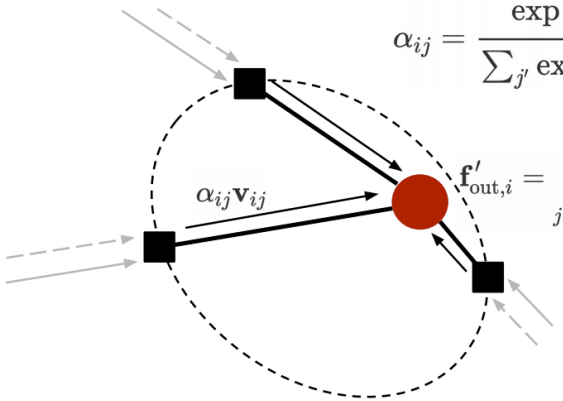


Figure 2: Updating the node features using our equivariant attention mechanism in four steps. A more detailed description, especially of step 2, is provided in the Appendix. Steps 3 and 4 visualise a graph network perspective: features are passed from nodes to edges to compute keys, queries and values, which depend both on features and relative positions in a rotation-equivariant manner.

SE(3)-Transformer (2)

Attention is performed on a per-neighborhood basis as follows:

$$\mathbf{f}_{\text{out},i}^{\ell} = \underbrace{\mathbf{W}_V^{\ell\ell}\mathbf{f}_{\text{in},i}^{\ell}}_{\text{③ self-interaction}} + \sum_{k\geq 0}\sum_{j\in\mathcal{N}_i\setminus i} \underbrace{\alpha_{ij}}_{\text{① attention}} \underbrace{\mathbf{W}_V^{\ell k}(\mathbf{x}_j - \mathbf{x}_i)\mathbf{f}_{\text{in},j}^{k}}_{\text{② value message}}. \tag{10}$$

The attention mechanism consists of a normalized inner product between a query vector $\boldsymbol{q}_i$ at node i and a set of key vectors $\{\boldsymbol{k}_{ij}\}_{j\in\mathcal{N}_i}$ along each edge ij in the neighborhood $\mathcal{N}_i$:

$$\alpha_{ij} = \frac{\exp(\mathbf{q}_i^\top \mathbf{k}_{ij})}{\sum_{j'\in\mathcal{N}_i\setminus i}\exp(\mathbf{q}_i^\top \mathbf{k}_{ij'})}, \quad \mathbf{q}_i = \bigoplus_{\ell\geq 0}\sum_{k\geq 0}\mathbf{W}_Q^{\ell k}\mathbf{f}_{\text{in},i}^{k}, \quad \mathbf{k}_{ij} = \bigoplus_{\ell\geq 0}\sum_{k\geq 0}\mathbf{W}_K^{\ell k}(\mathbf{x}_j - \mathbf{x}_i)\mathbf{f}_{\text{in},j}^{k}. \tag{11}$$

Note: This is just Cormorant with attention(!?)

Experiments (1)

Table 1: Predicting future locations and velocities in an electron-proton simulation.

		Linear	DeepSet [40]	Tensor Field [25]	Set Transformer [14]	**SE(3)-Transformer**
Position	MSE x	0.0691	0.0639	0.0151	0.0139	**0.0076**
	std	-	0.0086	0.0011	0.0004	0.0002
	Δ_{EQ}	-	0.038	$1.9 \cdot 10^{-7}$	0.167	$3.2 \cdot 10^{-7}$
Velocity	MSE v	0.261	0.246	0.125	0.101	**0.075**
	std	-	0.017	0.002	0.004	0.001
	Δ_{EQ}	-	1.11	$5.0 \cdot 10^{-7}$	0.37	$6.3 \cdot 10^{-7}$

Toy example:

- Five particles each carry either a positive or a negative charge, and exert repulsive or attractive forces on each other.
- The input to the network is the position of a particle in a specific time step, its velocity, and its charge.
- The task is to predict the relative location and **velocity** (?) 500 time steps into the future.

Experiments (2)

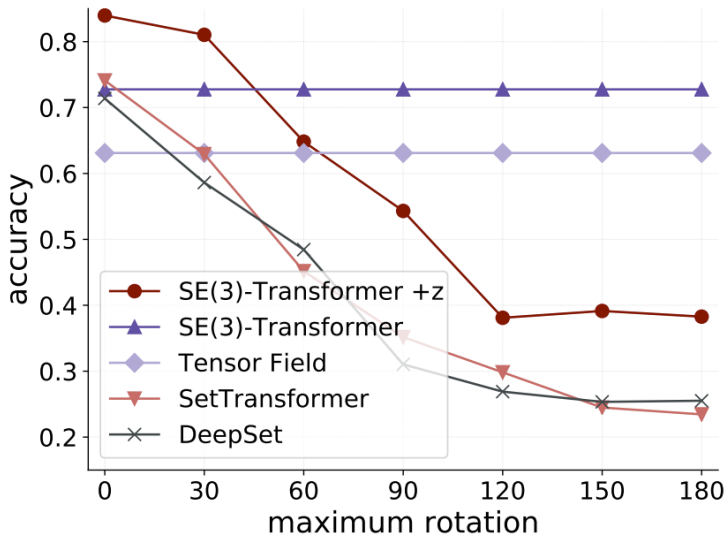
Table 2: Classification accuracy on the 'object only' category of the ScanObjectNN dataset[4]. The performance of the SE(3)-Transformer is averaged over 5 runs (standard deviation 0.7%).

	DeepSet	3DmFV	Set Transformer	PointNet	SpiderCNN	Tensor Field +z	PointNet++	**SE(3)-Transf.+z**	PointCNN	DGCNN	PointGLR
No. Points	1024	1024	1024	1024	1024	128	1024	**128**	1024	1024	1024
Accuracy	71.4%	73.8%	74.1%	79.2%	79.5%	81.0%	84.3%	**85.0%**	85.5%	86.2%	87.2%

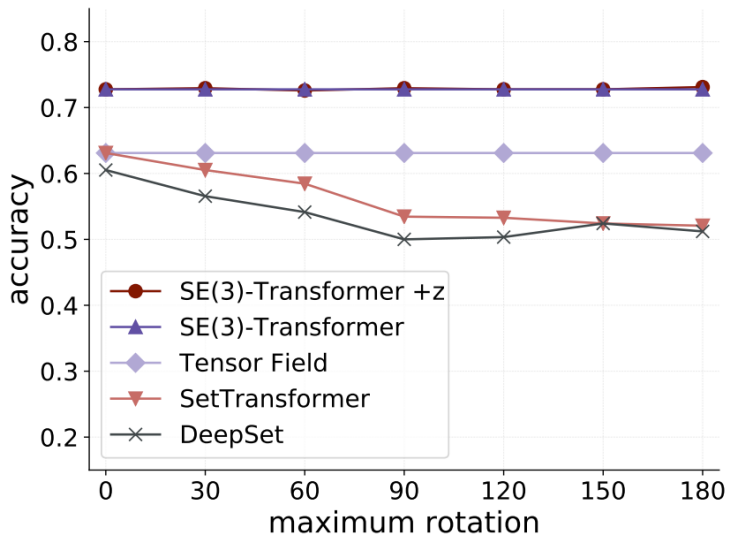
ScanObjectNN:

- Real-world object classification.
- Point clouds of 2,902 objects across 15 different categories.

Experiments (3)



(a) Training *without* data augmentation.



(b) Training *with* data augmentation.

Figure 4: ScanObjectNN: x-axis shows data augmentation on the test set. The x-value corresponds to the maximum rotation around a random axis in the x-y-plane. If both training and test set are not rotated ($x = 0$ in **a**), breaking the symmetry of the SE(3)-Transformer by providing the z-component of the coordinates as an additional, scalar input improves the performance significantly. Interestingly, the model learns to ignore the additional, symmetry-breaking input when the training set presents a rotation-invariant problem (strongly overlapping dark red circles and dark purple triangles in **b**).

Experiments (4)

Table 3: QM9 Mean Absolute Error. Top: Non-equivariant models. Bottom: Equivariant models

TASK UNITS	α bohr^3	$\Delta\varepsilon$ meV	$\varepsilon_{\text{HOMO}}$ meV	$\varepsilon_{\text{LUMO}}$ meV	μ D	C_ν cal/mol K
WaveScatt [9]	.160	118	85	76	.340	.049
NMP [8]	.092	69	43	38	.030	.040
SchNet [22]	.235	63	41	34	.033	.033
Cormorant [1]	.085	61	34	38	.038	.026
LieConv(T3) [6]	.084	49	30	25	.032	.038
TFN [25]	.223	58	40	38	.064	.101
Us	.148	53	36	33	.053	.057

Paper 3

PointFlow: 3D Point Cloud Generation with Continuous Normalizing Flows (ICCV 2019)

Guandao Yang, Xun Huang, Zekun Hao, Ming-Yu Liu, Serge Belongie, Bharath Hariharan

`https://arxiv.org/pdf/1906.12320.pdf`

PointFlow (1) – Motivation

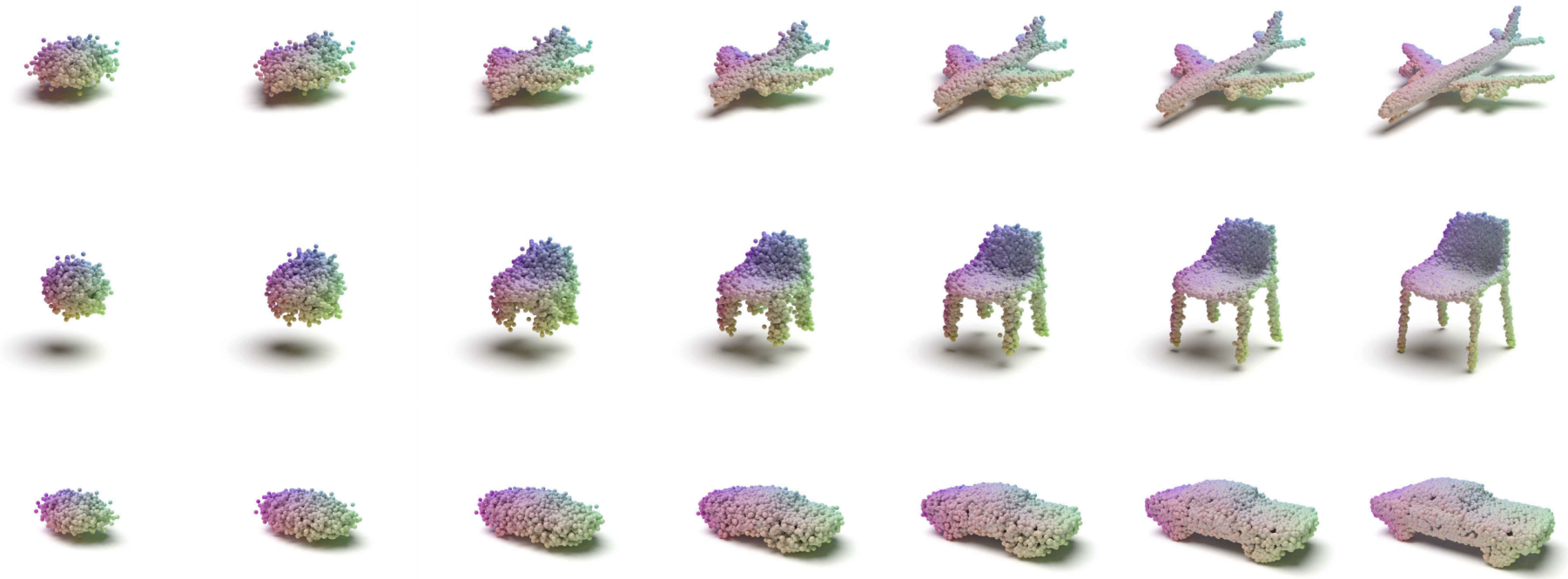


Figure 1: Our model transforms points sampled from a simple prior to realistic point clouds through continuous normalizing flows. The videos of the transformations can be viewed on our project website: https://www.guandaoyang.com/PointFlow/.

PointFlow (2) – Continuous normalizing flow (CNF)

Let f_1, f_2, .., f_n denote a series of **invertible** transformations we want to apply to a latent variable y with a distribution $P(y)$:

$$x = f_n \circ f_{n-1} \circ \cdots \circ f_1(y)$$

is the output variable. The probability density of the output variable is given by the change of variables formula:

$$\log P(x) = \log P(y) - \sum_{k=1}^{n} \log \left| \det \frac{\partial f_k}{\partial y_{k-1}} \right|$$

where y can be computed from x using the **inverse flow**:

$$y = f_1^{-1} \circ \cdots \circ f_n^{-1}(x)$$

PointFlow (3) – Idea

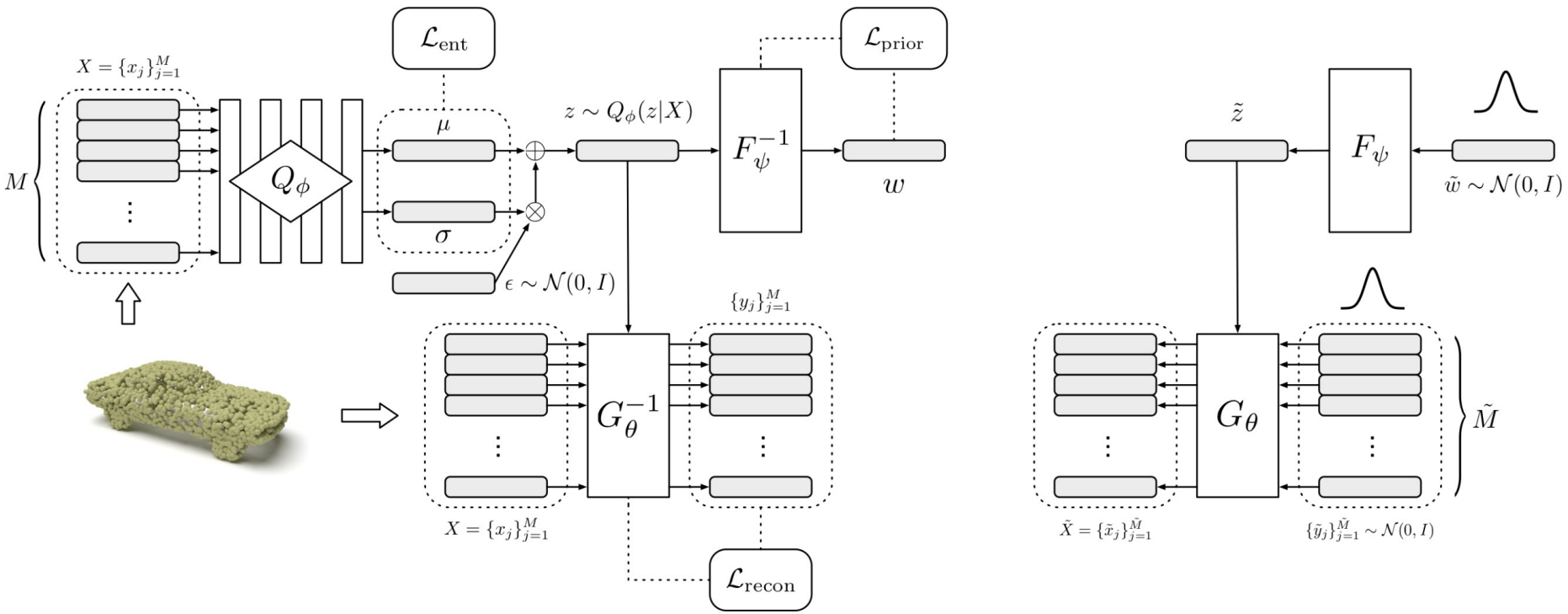
I think we can summarize the idea as:

1. **Flow-based prior over shapes:** First, we want to generate the shape representation by sampling from a simple Gaussian prior and then transform it through a **learnable** CNF.
2. **Flow-based point generation from shape representation:** We extend the CNF to a conditional version. Basically, we propose the model $P_\theta(x|z)$ conditioned on the shape representation z. Each point x is transformed independently through the dynamics of CNF.

Note

I think we can apply this idea into our graph generation.

PointFlow (4) – Overview



(a) Training (Auto-encoding) (a) Test (Sampling)

Figure 2: Model architecture. (a) At training time, the encoder Q_ϕ infers a posterior over shape representations given an input point cloud X, and samples a shape representation z from it. We then compute the probability of z in the prior distribution ($\mathcal{L}_{\text{prior}}$) through a inverse CNF F_ψ^{-1}, and compute the reconstruction likelihood of X ($\mathcal{L}_{\text{recon}}$) through another inverse CNF G_θ^{-1} conditioned on z. The model is trained end-to-end to maximize the evidence lower bound (ELBO), which is the sum of $\mathcal{L}_{\text{prior}}$, $\mathcal{L}_{\text{recon}}$, and $\mathcal{L}_{\text{ent}}$ (the entropy of the posterior $Q_\phi(z|X)$). (b) At test time, we sample a shape representation $\tilde{z}$ by sampling $\tilde{w}$ from a Gaussian prior and transforming it with F_ψ. To sample points from the shape represented by $\tilde{z}$, we first sample points from the 3-D Gaussian prior and then move them according to the CNF parameterized by $\tilde{z}$.

PointFlow (5) – Evaluation metrics

Evaluation metrics of similary between 2 point clouds X and Y:

1. **Chamfer distance (CD):** looking for the nearest match

$$CD(X, Y) = \sum_{x \in X} \min_{y \in Y} ||x - y||_2^2 + \sum_{y \in Y} \min_{x \in X} ||x - y||_2^2$$

2. **Earth mover's distance (EMD):** optimal transport

$$EMD(X, Y) = \min_{\phi: X \to Y} \sum_{x \in X} ||x - \phi(x)||_2^2$$

PointFlow (6) – Evaluation metrics

Let S_g be the set of generated point clouds and S_r be the set of reference point clouds with $|S_r| = |S_g|$. Let P_g and P_r be the marginal distributions of points in the S_g and S_r, approximated by discretizing the space into 28^3 voxels and assigning each point to one of them.

- **Jensen-Shannon Divergence (JSD):**

$$JSD(P_g, P_r) = \frac{1}{2}\mathcal{D}_{KL}(P_r||M) + \frac{1}{2}\mathcal{D}_{KL}(P_g||M)$$

where $M = (P_r + P_g)/2$.
Problem of JSD is if the model just outputs the average shape, it can get a perfect score.

PointFlow (7) – Evaluation metrics

- **Coverage (COV):** measures the fraction of point clouds in the reference set that are matched to at lest one point cloud in the generated set.

$$COV(S_g, S_r) = \frac{|\{\arg\min_{Y \in S_r} D(X, Y) | X \in S_g\}|}{|S_r|}$$

It can detect **mode collapse** when the score is as low as $1/|S_r|$. But it does not evaluate the quality of generated point clouds.

PointFlow (8) – Evaluation metrics

- **Minimum matching distance (MMD):**

$$MMD(S_g, S_r) = \frac{1}{|S_r|} \sum_{Y \in S_r} \min_{X \in S_g} D(X, Y)$$

MMD is insensitive to low-quality point clouds in S_g, since they are unlikely to be matched to real point clouds in S_r.

PointFlow (9)

- **1-nearest neighbor accuracy (1-NNA):** Let $S_{-X} = S_r \cup S_g - \{X\}$ and N_X be the nearest neighbor X in S_{-X}.

$$1 - NNA(S_g, S_r) = \frac{\sum_{X \in S_g} \mathbb{I}[N_X \in S_g] + \sum_{Y \in S_r} \mathbb{I}[N_Y \in S_r]}{|S_g| + |S_r|}$$

  where $\mathbb{I}[\cdot]$ is the indicator function.

Open question

I think we should add/improve metrics for graph/molecule generation:

1. Compare based on **graph similarity** $\rightarrow$ the base distance is some graph kernel.
2. Compare based on **molecular properties**.

PointFlow (10)

Table 1: Generation results. ↑: the higher the better, ↓: the lower the better. The best scores are highlighted in bold. Scores of the real shapes that are worse than some of the generated shapes are marked in gray. MMD-CD scores are multiplied by 10^3; MMD-EMD scores are multiplied by 10^2; JSDs are multiplied by 10^2.

Category	Model	# Parameters (M)		JSD (↓)	MMD (↓)		COV (%, ↑)		1-NNA (%, ↓)	
		Full	Gen		CD	EMD	CD	EMD	CD	EMD
Airplane	r-GAN	7.22	6.91	7.44	0.261	5.47	42.72	18.02	93.58	99.51
	l-GAN (CD)	1.97	1.71	4.62	0.239	4.27	43.21	21.23	86.30	97.28
	l-GAN (EMD)	1.97	1.71	**3.61**	0.269	3.29	**47.90**	**50.62**	87.65	85.68
	PC-GAN	9.14	1.52	4.63	0.287	3.57	36.46	40.94	94.35	92.32
	PointFlow (ours)	**1.61**	**1.06**	4.92	**0.217**	**3.24**	46.91	48.40	**75.68**	**75.06**
	Training set	-	-	6.61	0.226	3.08	42.72	49.14	70.62	67.53
Chair	r-GAN	7.22	6.91	11.5	2.57	12.8	33.99	9.97	71.75	99.47
	l-GAN (CD)	1.97	1.71	4.59	2.46	8.91	41.39	25.68	64.43	85.27
	l-GAN (EMD)	1.97	1.71	2.27	2.61	**7.85**	40.79	41.69	64.73	65.56
	PC-GAN	9.14	1.52	3.90	2.75	8.20	36.50	38.98	76.03	78.37
	PointFlow (ours)	**1.61**	**1.06**	**1.74**	**2.42**	7.87	**46.83**	**46.98**	**60.88**	**59.89**
	Training set	-	-	1.50	1.92	7.38	57.25	55.44	59.67	58.46
Car	r-GAN	7.22	6.91	12.8	1.27	8.74	15.06	9.38	97.87	99.86
	l-GAN (CD)	1.97	1.71	4.43	1.55	6.25	38.64	18.47	63.07	88.07
	l-GAN (EMD)	1.97	1.71	2.21	1.48	5.43	39.20	39.77	69.74	68.32
	PC-GAN	9.14	1.52	5.85	1.12	5.83	23.56	30.29	92.19	90.87
	PointFlow (ours)	**1.61**	**1.06**	**0.87**	**0.91**	**5.22**	**44.03**	**46.59**	**60.65**	**62.36**
	Training set	-	-	0.86	1.03	5.33	48.30	51.42	57.39	53.27

PointFlow (11)

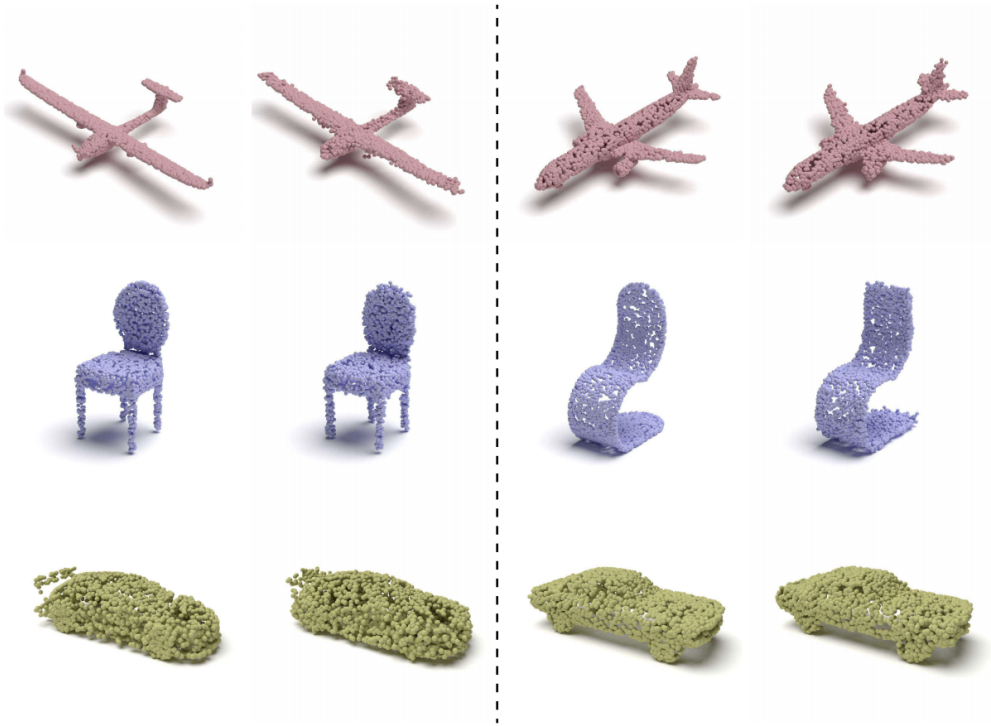


Figure 4: Examples of point clouds reconstructed from inputs. From top to bottom: airplane, chair, and car. On each side of the figure we show the input point cloud on the left and the reconstructed point cloud on the right.

PointFlow (12)

Table 2: Unsupervised feature learning. Models are first trained on ShapeNet to learn shape representations, which are then evaluated on ModelNet40 (MN40) and ModelNet10 (MN10) by comparing the accuracy of off-the-shelf SVMs trained using the learned representations.

Method	MN40 (%)	MN10 (%)
SPH [24]	68.2	79.8
LFD [4]	75.5	79.9
T-L Network [14]	74.4	-
VConv-DAE [42]	75.5	80.5
3D-GAN [48]	83.3	91.0
l-GAN (EMD) [1]	84.0	**95.4**
l-GAN (CD) [1]	84.5	**95.4**
PointGrow [44]	85.7	-
MRTNet-VAE [13]	86.4	-
FoldingNet [51]	**88.4**	94.4
l-GAN (CD) [1] †	**87.0**	92.8
l-GAN (EMD) [1] †	86.7	92.2
PointFlow (ours)	86.8	**93.7**

† We run the official code of l-GAN on our pre-processed dataset using the same encoder architecture as our model.

Note

We can **unsupervisedly** learn graph/molecule representation by VAE/AE and evaluate the learned representation by a down-stream task.

PointFlow (13)

Table 3: Auto-encoding performance evaluated by CD and EMD. AtlasNet is trained with CD and l-GAN is trained on CD or EMD. Our method is not trained on CD or EMD. CD scores are multiplied by 10^4; EMD scores are multiplied by 10^2.

Model	# Parameters (M)	CD	EMD
l-GAN (CD) [1]	1.77	7.12	7.95
l-GAN (EMD) [1]	1.77	8.85	5.26
AtlasNet [17]	44.9	**5.13**	5.97
PointFlow (ours)	**1.30**	7.54	**5.18**

PointFlow (14) – Idea for our research

Idea from PointFlow (needs some modification) for our task in graph/molecule generation:

1. Flow-based prior over the **global** graph/molecule shape.
2. Flow-based point generation from the **global** representation:
   - Each atom/node/vertex is associated with a point in the point cloud.
   - The adjacency is determined by the distance between 2 points given a threshold.