

# **Graph Neural Networks and Bayesian Learning for molecular property prediction**

**Seongok Ryu, Ph.D, Research Scientist in Galux**

# Let me introduce myself

## Seongok Ryu

- Education
  - (2009. 02 ~ 2014. 02) B.S.: KAIST, Chemistry and Physics,
  - (2014. 02 ~ 2020. 02) Ph.D: KAIST, Chemistry, Advisor: Woo Youn Kim  
Thesis: Machine learning systems for drug discovery
- Career
  - (2020. 02 ~ 2021. 03) AITRICS, Drug Discovery, Research Scientist & Team Leader
  - (2021. 04 ~ ) Galux, Small molecule drug team, Research Scientist
- Research Interest
  - Applying machine learning for molecule design, QSAR modeling, reaction planning and others
  - Computational chemistry: molecular docking, molecular dynamics
  - All about *in silico* drug discovery



Let me introduce our company, GALUX



## 카카오브레인, AI 신약 개발사 '갤럭스'에 50억원 투자

| 단백질 모델링 기술-AI 초거대 모델 역량 융합해 신약 개발 산업 혁신 목표

인터넷 | 입력 : 2021/12/16 14:01

- 서울대 화학과 석차옥 교수님과 대학원생들이 공동창업
- 2004년부터 단백질 구조예측, molecule docking, protein design 등 신약개발 소프트웨어를 개발
- CASP, CAPRI 등과 같은 국제 대회에서 우수한 성적을 거두어왔음
- We are hiring! AI, 계산화학, 합성, 단백질 합성, 생물학 실험 등 다양한 분야에서 채용이 진행중입니다.

## 강의 자료, 예시 코드

- [github.com/seongokryu/LAIDD\\_GNN](https://github.com/seongokryu/LAIDD_GNN)
- [github.com/seongokryu/my-study-materials](https://github.com/seongokryu/my-study-materials)
- [github.com/AITRICS/mol\\_reliable\\_gnn](https://github.com/AITRICS/mol_reliable_gnn)

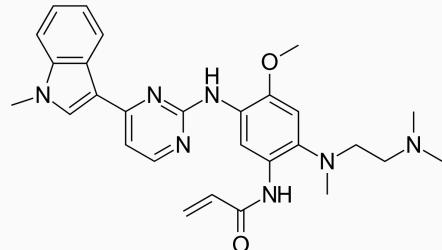
# Installation

- 강의에서 필요한 package
  - Anaconda: 파이썬 실행을 위한 가상환경 및 패키지 관리  
→ Anaconda 홈페이지 참고
  - PyTorch: Deep Learning 구현을 위한 Python Library  
→ PyTorch 홈페이지 Installation 참고
  - Deep Graph Library (DGL): Graph Neural Network 구현을 위한 Python Library  
→ DGL 홈페이지 참고  
→ conda install 명령어: **conda install -c dgltteam dgl-cuda11.1**
  - RDKit: Molecule Data를 다루기 위한 Python Library  
→ conda install 명령어: **conda install -c rdkit rdkit**
  - Therapeutic Data Commons: Molecule Data를 다운받고 Dataset으로 변환을 위한 Python Library  
→ PIP install 명령어: pip install PyTDC

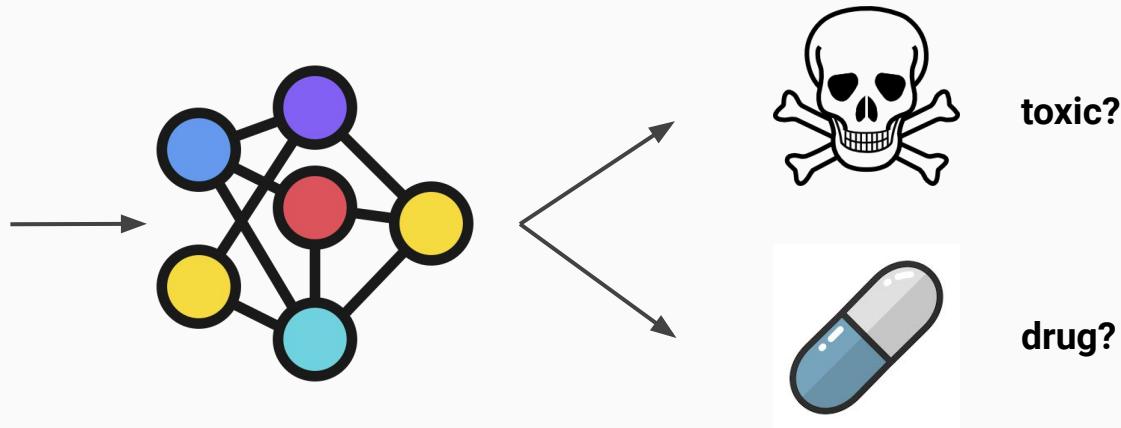
# Motivation

# ML applications in drug discovery

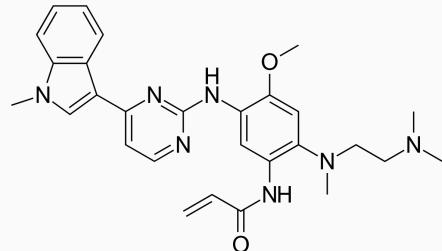
- Prediction model



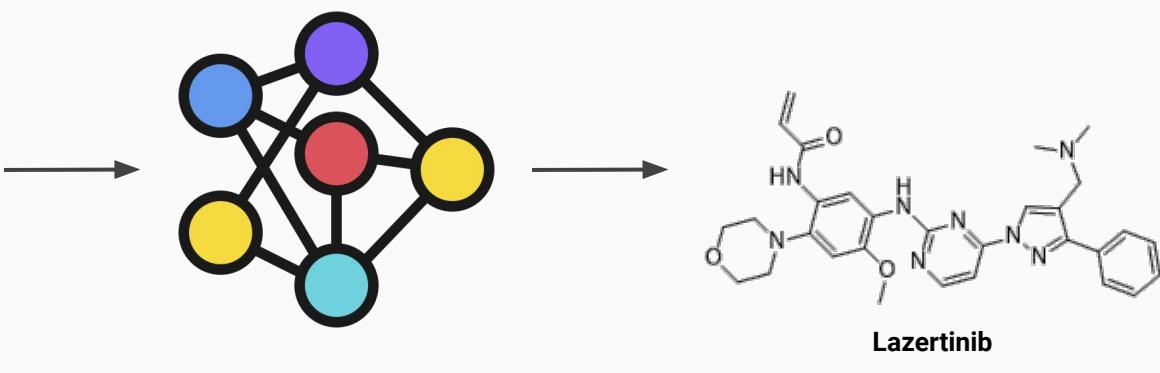
Osimertinib



- Generative model



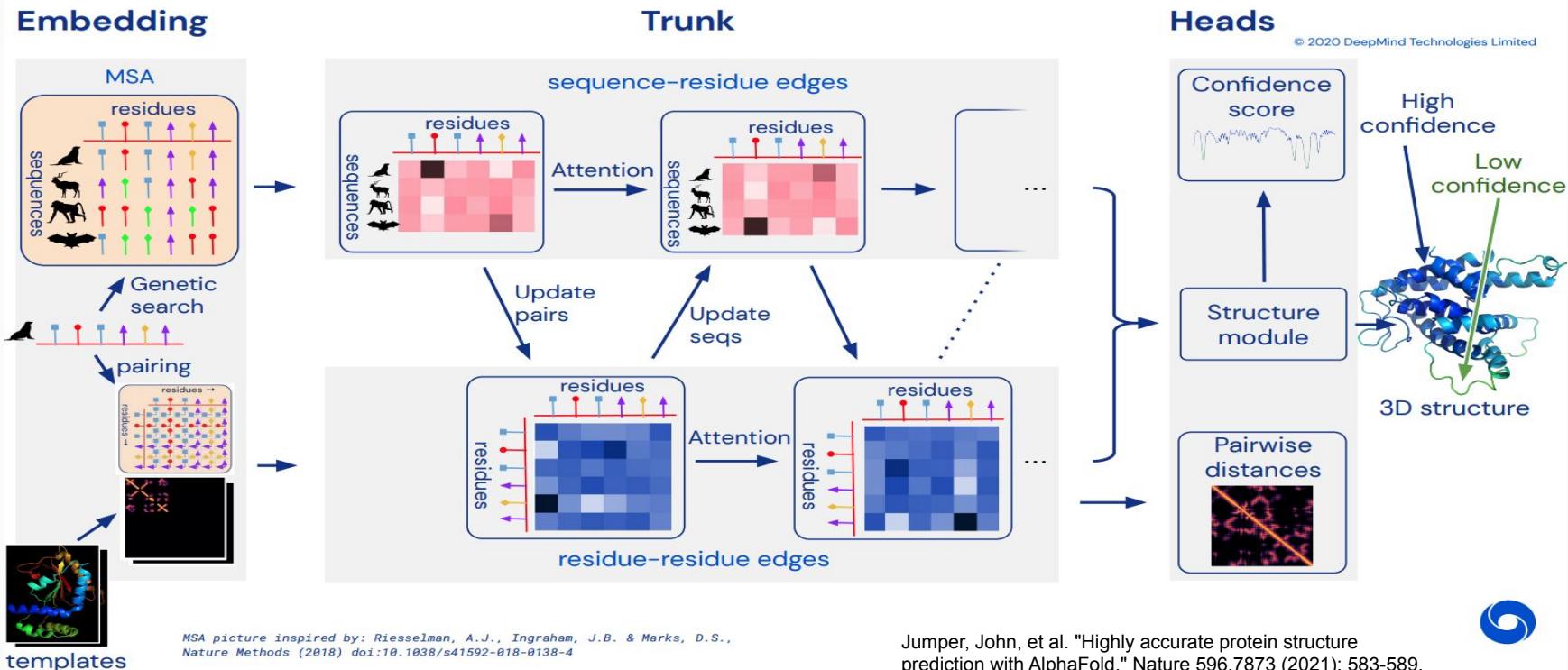
Osimertinib



Lazertinib

# ML applications in drug discovery

- **AlphaFold2:** using self-attention (Transformer) to update sequence and residue features

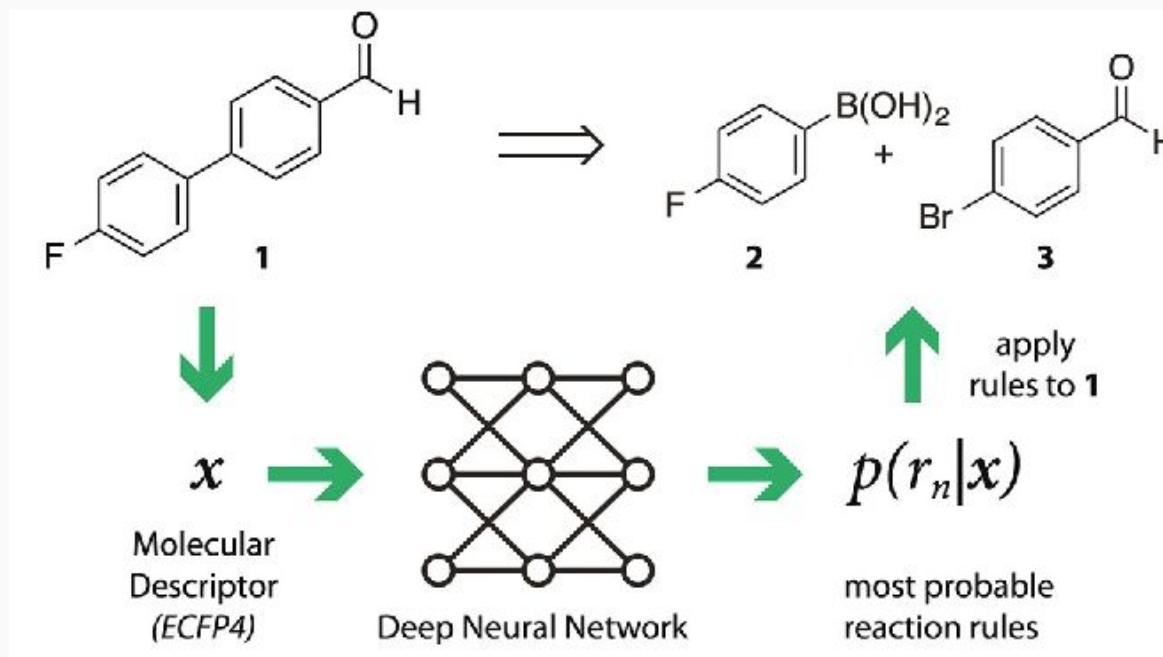


Jumper, John, et al. "Highly accurate protein structure prediction with AlphaFold." *Nature* 596.7873 (2021): 583-589.



# ML applications in drug discovery

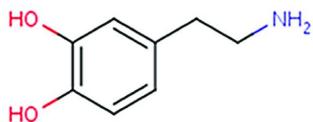
- **Retrosynthesis:** to infer “how to synthesize a given molecule” in most efficient/possible way



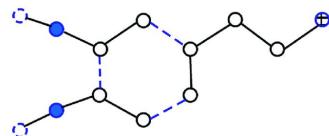
Segler, Marwin HS, and Mark P. Waller. "Neural-symbolic machine learning for retrosynthesis and reaction prediction." Chemistry—A European Journal 23.25 (2017): 5966-5971.

# Molecules can be represented with “Graph”

Dopamine



Molecular graph



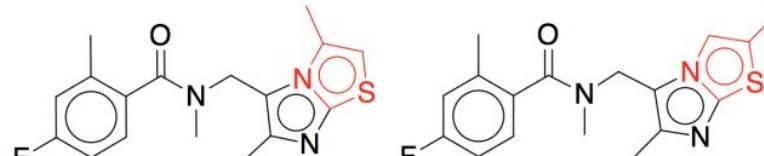
Nodes	Atomic Symbol	Valency
○	C	4
○	H	1
●	O	2
⊕	N	3

Edges	Chemical bond
—	simple bond
----	double bond

Jin, Wengong, Regina Barzilay, and Tommi Jaakkola.

"Junction tree variational autoencoder for molecular graph generation."

International conference on machine learning. PMLR, 2018.



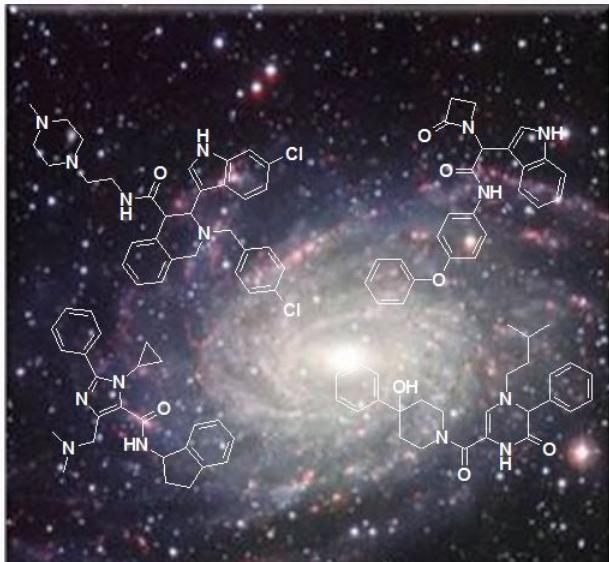
Cc1cn2c(CN(C)C(=O)c3ccc(F)cc3C)c(C)nc2s1  
Cc1cc(F)ccc1C(=O)N(C)Cc1c(C)nc2scc(C)n12

Figure 1. Two almost identical molecules with markedly different canonical SMILES in RDKit. The edit distance between two strings is 22 (50.5% of the whole sequence).

- Molecules can be represented with SMILES, 1-dim string representation.
- However, SMILES representation has a limitation as shown in above.
- Thus, it may be better to use graph representation for an input of ML model.

# Data-hungry problem in molecular science

Size of chemical space (Enamine REAL):  $10^{10}$



>>



Tox21: dataset for toxicity prediction  
# of data points:  $< 10^4$

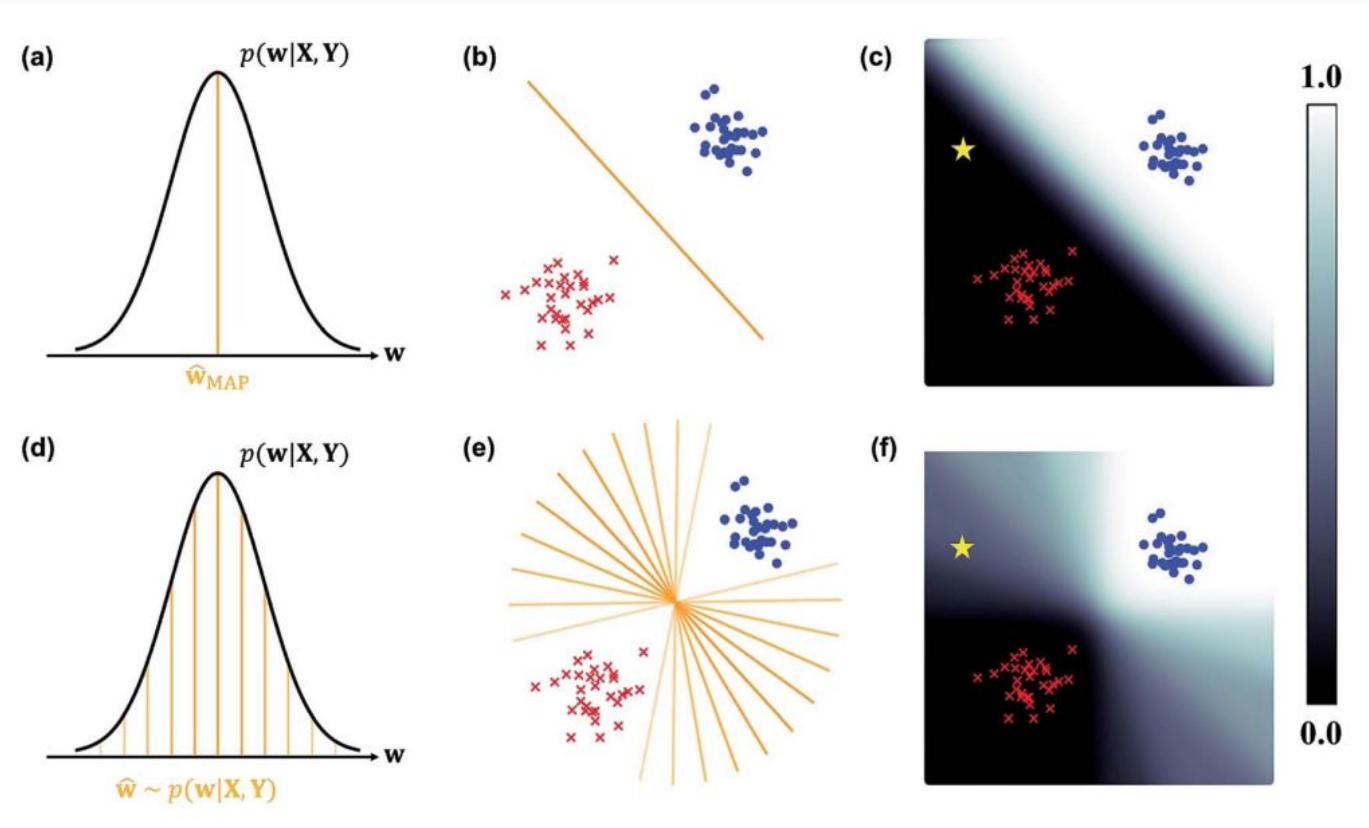


DUD-E: dataset for binding affinity prediction  
# of data points:  $O(10^3) \sim O(10^4)$

- Usually, dataset size is much smaller than the chemical space (or the size of virtual screening library).
- Also, noisy labels are obtained because of stochastic nature of biochemical assays.
- Thus, using such datasets make models prone to poor generalization (over-fitting problem)

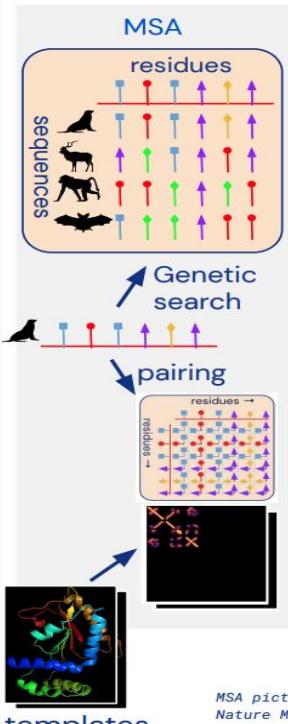
# Data-hungry problem in molecular science

- Bayesian approach enables us to quantify uncertainty, which can be used for confidence score.

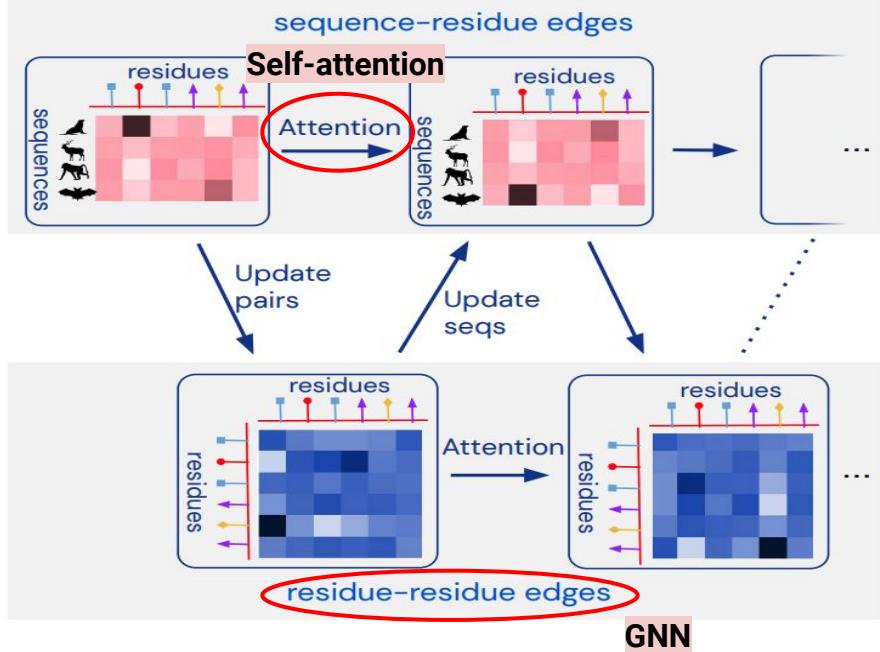


# How AlphaFold2 achieve state-of-the-art performance

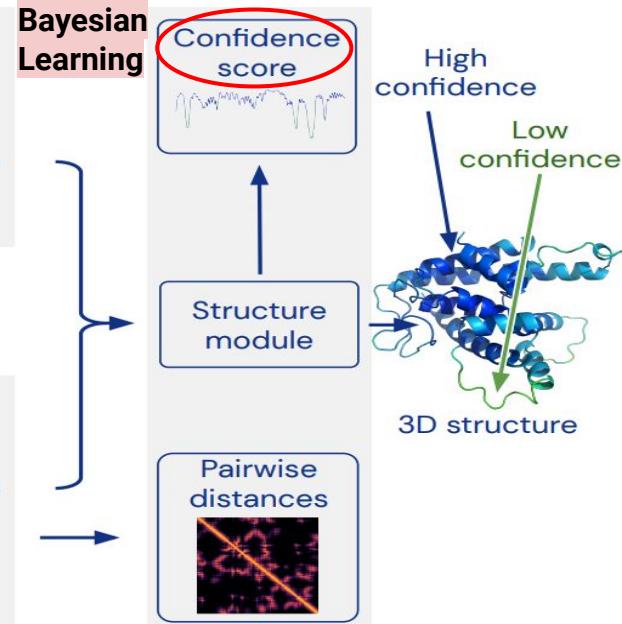
## Embedding



## Trunk



## Heads



MSA picture inspired by: Riesselman, A.J., Ingraham, J.B. & Marks, D.S., Nature Methods (2018) doi:10.1038/s41592-018-0138-4

Jumper, John, et al. "Highly accurate protein structure prediction with AlphaFold." Nature 596.7873 (2021): 583-589.



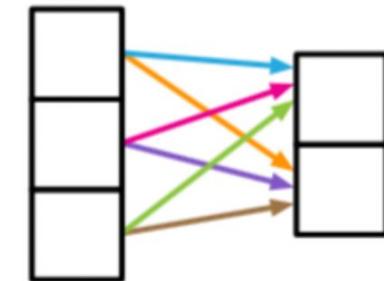
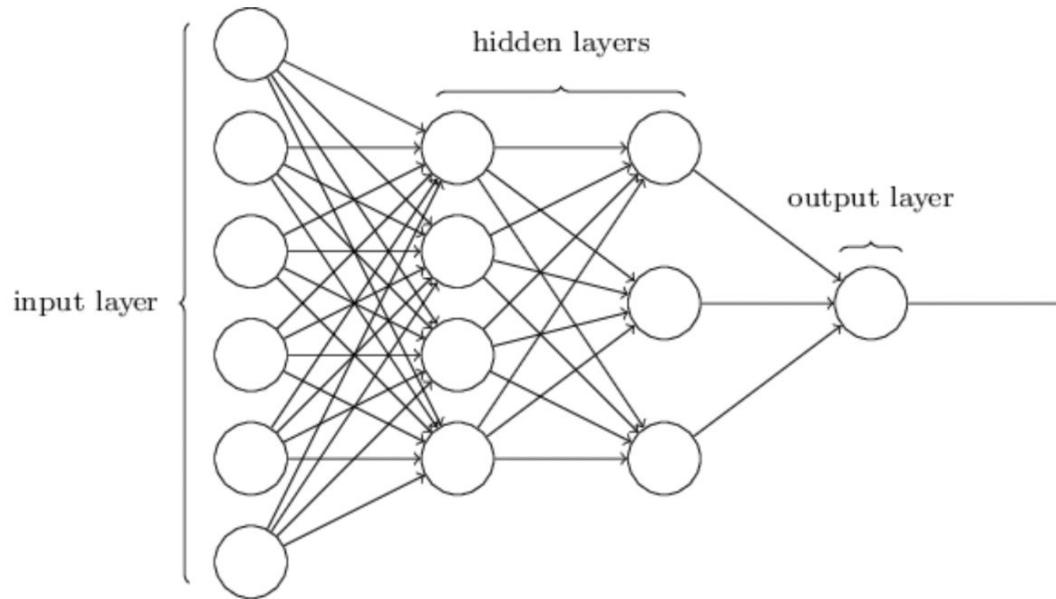
# In today's lecture

- **Graph Neural Networks**
  - Relational inductive bias in neural networks
  - Node-/edge-/graph- level feature updating
  - Graph Convolutional Network, Graph Isomorphism Network, ...
- **Self-attention for GNN**
  - Set transformer
  - Node-/graph- level attention
- **Bayesian Learning**
  - Why Bayesian Learning in molecular science?
  - Variational inference
  - Dropout as Bayesian approximation, a.k.a. MC-Dropout
  - Stochastic Weight Averaging (SWA)

# Graph Neural Networks

# Relational inductive bias

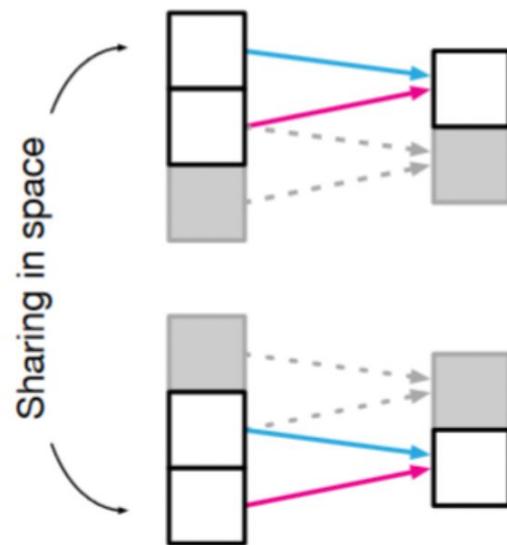
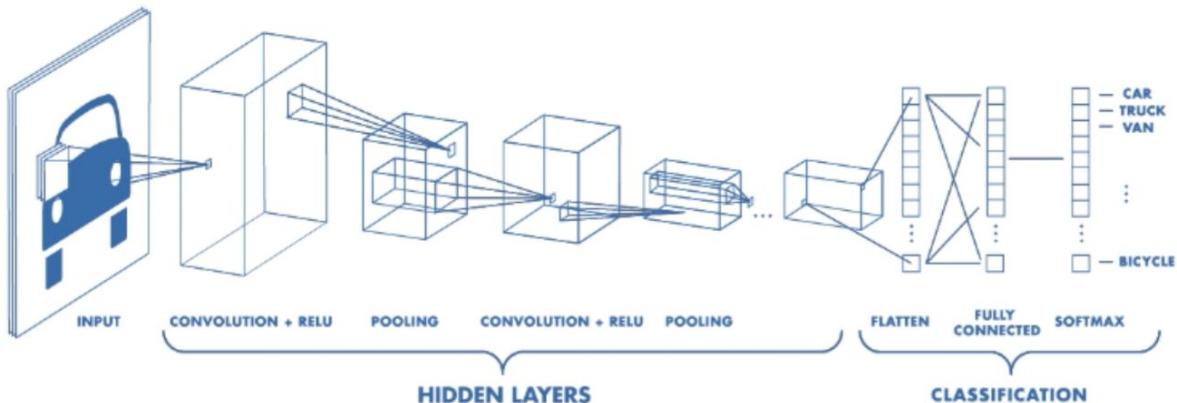
- **Weight sharing in neural networks**
  - Fully-connected neural network (also referred as multi-layer perceptron; MLP)



No weight sharing

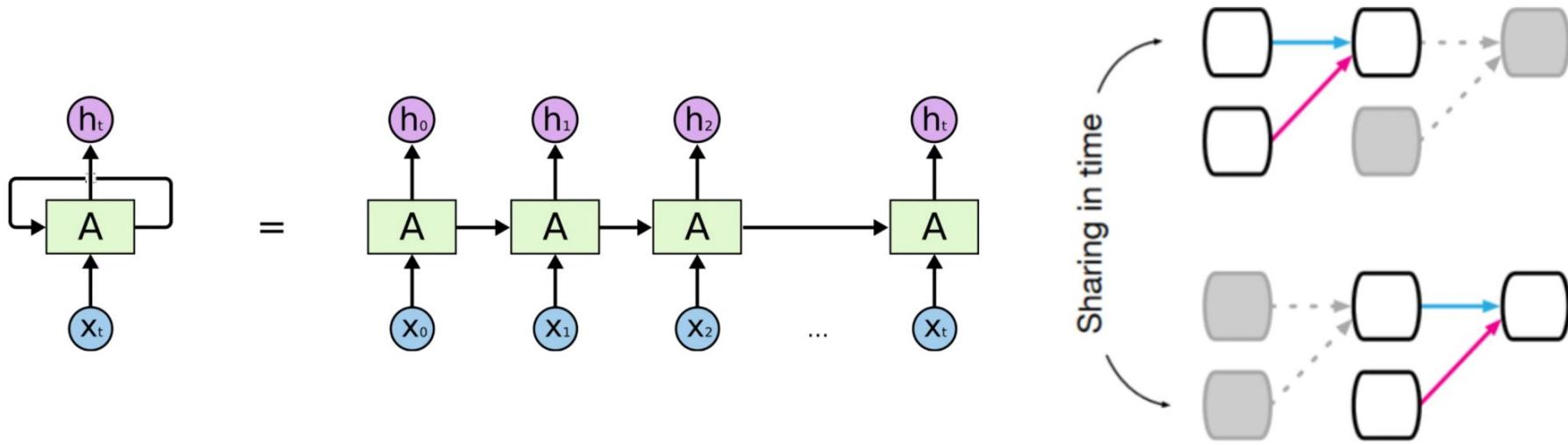
# Relational inductive bias

- Weight sharing in neural networks
  - Convolutional neural network (CNN)



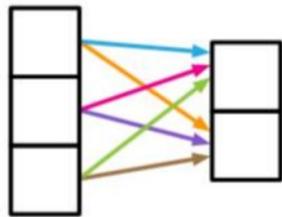
# Relational inductive bias

- Weight sharing in neural networks
  - Recurrent neural network (RNN)

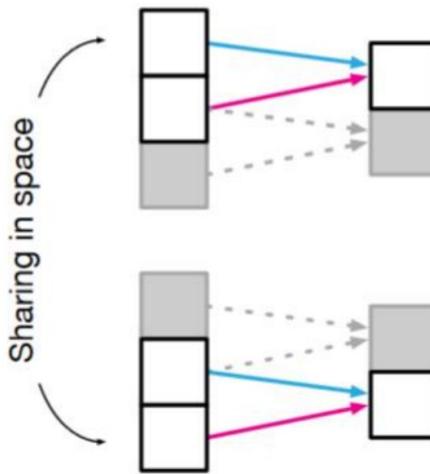


# Relational inductive bias

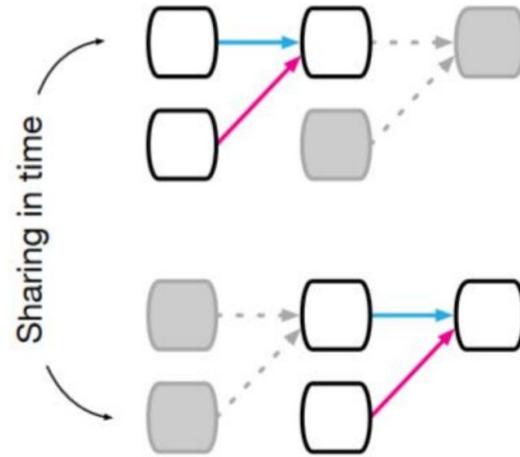
- Inductive bias is another name of weight sharing



(a) Fully connected



(b) Convolutional



(c) Recurrent

- Q) What is the inductive bias for each operation?
- Before to answer the question, what is inductive bias?

# Relational inductive bias

- **Interpretation of inductive bias**

- An inductive bias allows a learning algorithm to prioritize one solution (or interpretation) over another, independent of the observed data.
- In a Bayesian model, inductive biases are typically expressed through the choice and parameterization of the prior distribution.

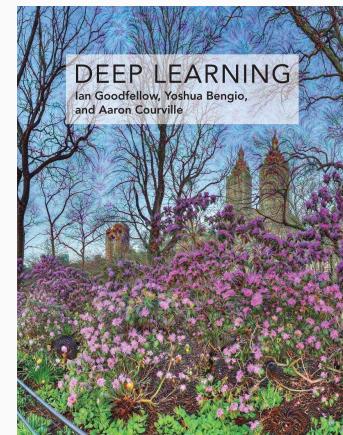
$$p(\omega|X, Y) = \frac{p(Y|X, \omega) \cdot \mathbf{p}(\omega)}{p(Y|X)}$$

- In other contexts, an inductive bias might be a regularization term added to avoid overfitting, or it might be encoded in the architecture of the algorithm it self.

# Relational inductive bias

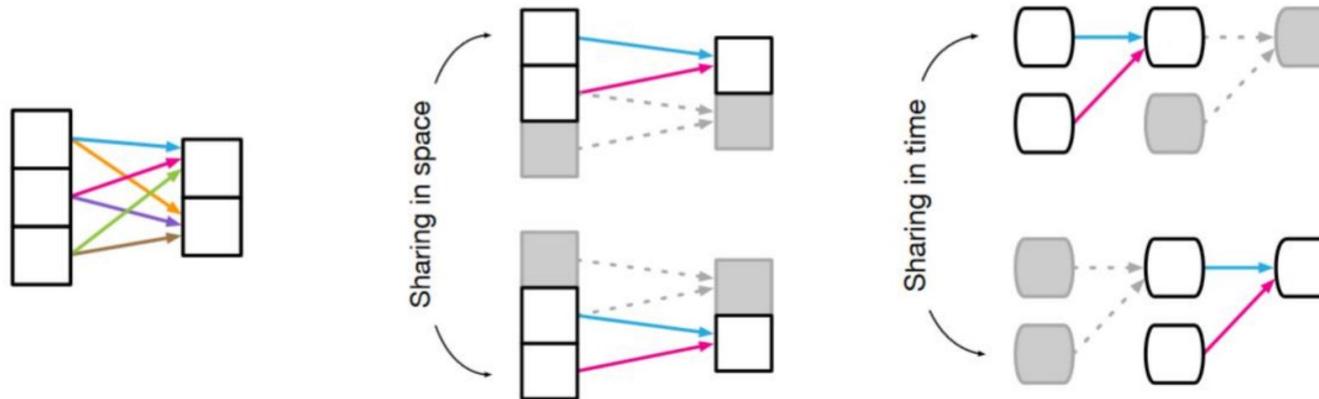
- **Interpretation of inductive bias**

- “Priors can be considered weak or strong depending on how concentrated the probability density in the prior is.
- A weak prior is a prior distribution with high entropy, such as a Gaussian distribution with high variance. Such a prior allows the data to move the parameters more or less freely.
- A strong prior has very low entropy, such as a Gaussian distribution with low variance. Such a prior plays a more active role in determining where the parameters end up.
- An infinitely strong prior places zero probability on some parameters and says that these parameter values are completely forbidden, regardless of how much support the data give to those values.”



# Relational inductive bias

- Different inductive biases in neural networks

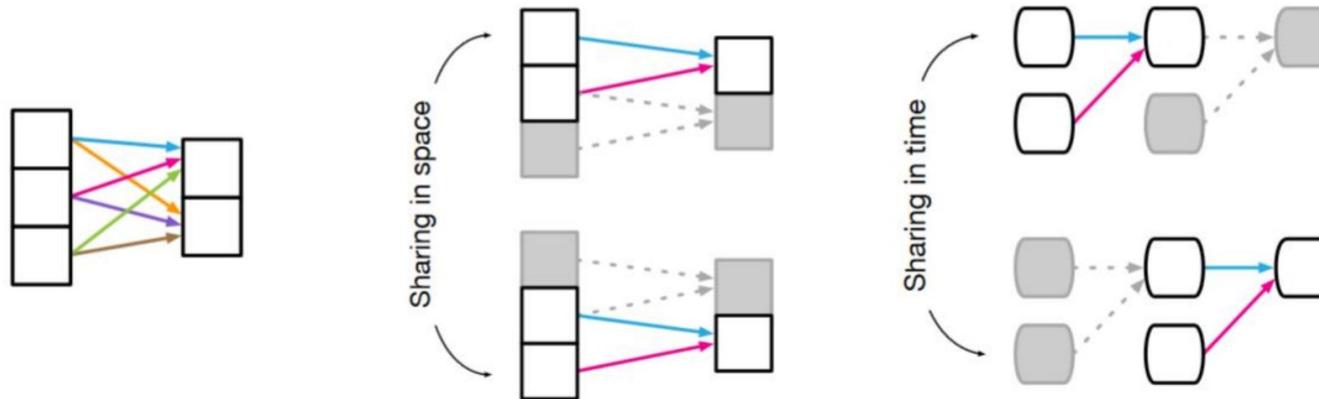


Component	Entities	Relations	Rel. inductive bias	Invariance
Fully connected	Units	All-to-all	Weak	-
Convolutional	Grid elements	Local	Locality	Spatial translation
Recurrent	Timesteps	Sequential	Sequentiality	Time translation

**Q) What are the entities, relations, relational inductive bias and invariance for GNN?**

# Relational inductive bias

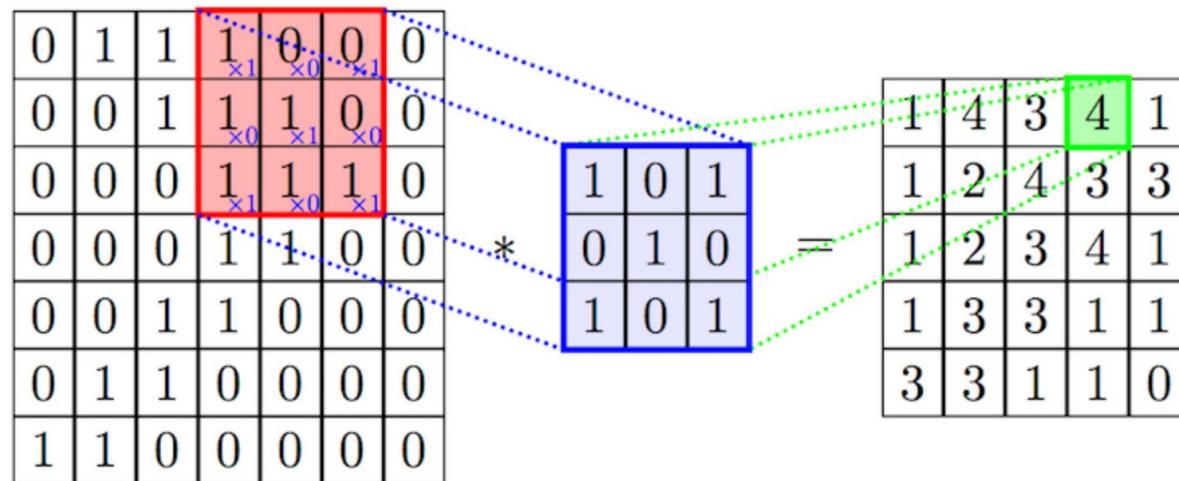
- Different inductive biases in neural networks



Component	Entities	Relations	Rel. inductive bias	Invariance
Fully connected	Units	All-to-all	Weak	-
Convolutional	Grid elements	Local	Locality	Spatial translation
Recurrent	Timesteps	Sequential	Sequentiality	Time translation
Graph network	Nodes	Edges	Arbitrary	Node, edge permutations

# Graph convolution

## Convolutional neural network

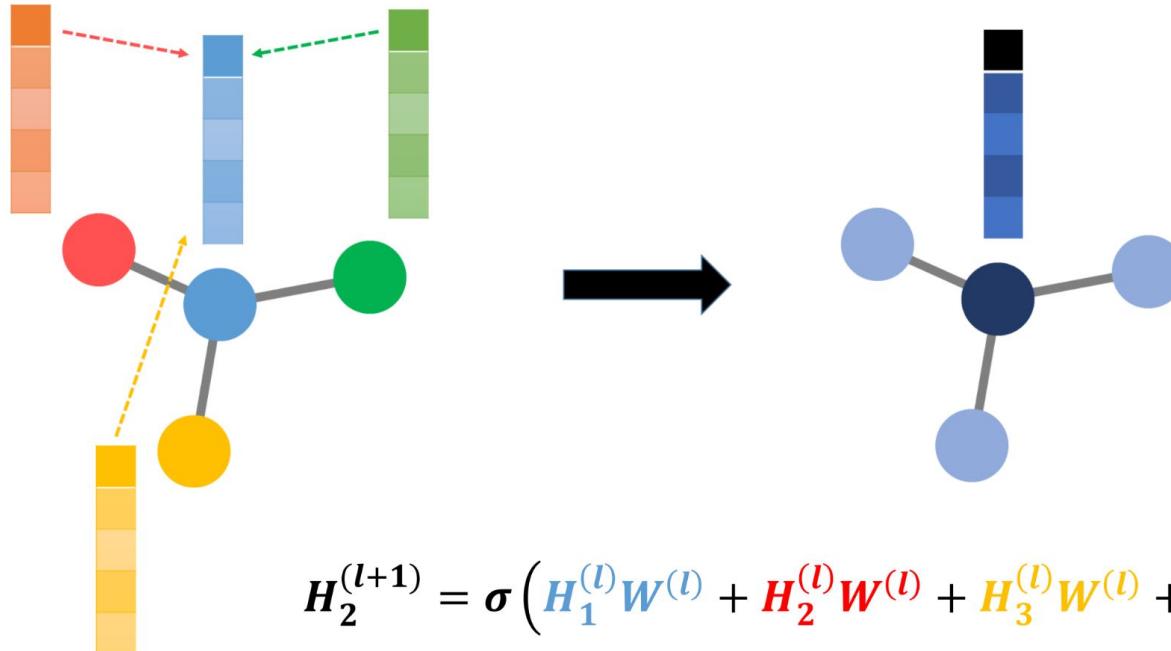


$$X_i^{(l+1)} = \sigma(\sum_{j \in [i-k, i+k]} W_j^{(l)} X_j^{(l)} + b^{(l)})$$

Learnable parameters are shared

# Graph convolution

## Graph convolutional network

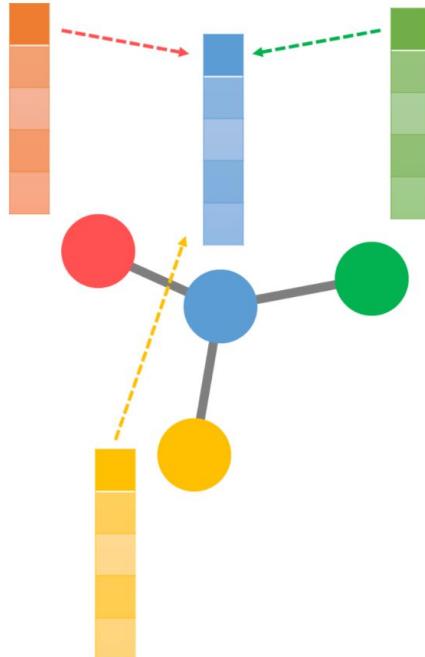


$$\mathbf{H}_2^{(l+1)} = \sigma \left( \mathbf{H}_1^{(l)} \mathbf{W}^{(l)} + \mathbf{H}_2^{(l)} \mathbf{W}^{(l)} + \mathbf{H}_3^{(l)} \mathbf{W}^{(l)} + \mathbf{H}_4^{(l)} \mathbf{W}^{(l)} \right)$$

$$\rightarrow \mathbf{H}_i^{(l+1)} = \sigma \left( \sum_{j \in N(i)} \mathbf{H}_j^{(l)} \mathbf{W}^{(l)} \right)$$

# Graph convolution

## Graph convolutional network



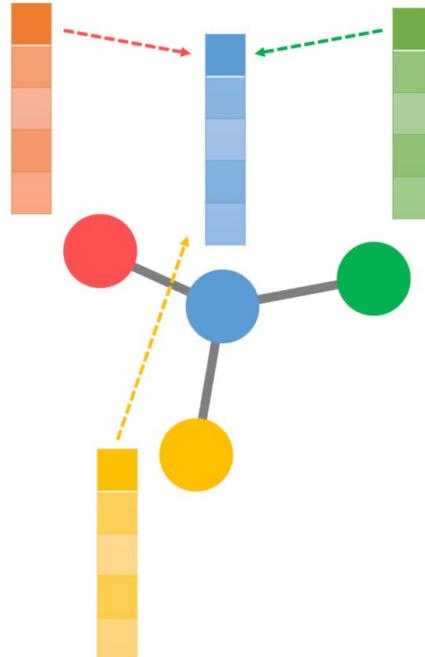
$$H^{(l+1)} = \sigma(\textcolor{red}{A} H^{(l)} \textcolor{blue}{W}^{(l)})$$

Learnable parameter is shared

**Question) What is the inductive bias for GCN?**

# Graph convolution

## Graph convolutional network



$$H^{(l+1)} = \sigma \left( \textcolor{red}{A} H^{(l)} \textcolor{blue}{W}^{(l)} \right)$$

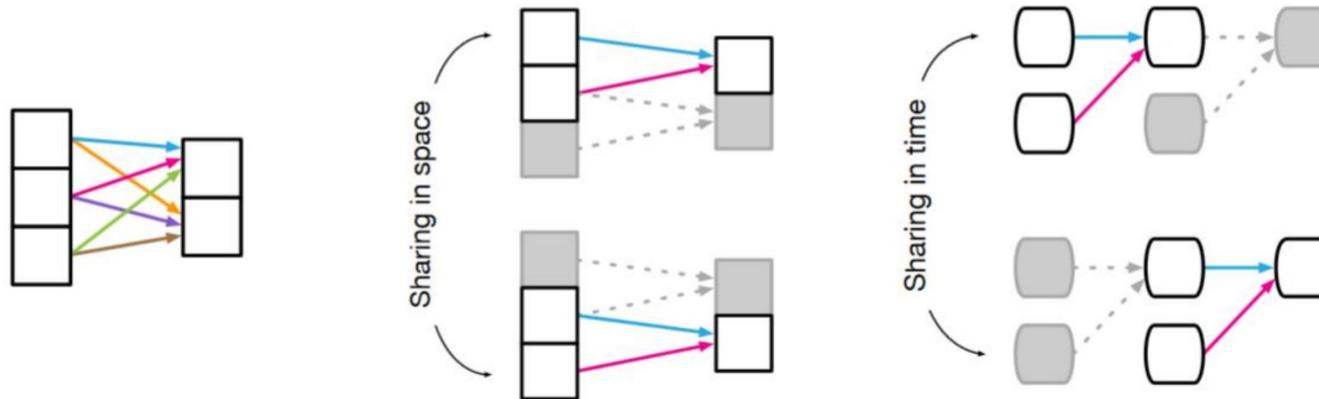
Learnable parameter is shared

**Question) What is the inductive bias for GCN?**

**Answer) Connectivity between nodes – the adjacency matrix**

# Relational inductive bias

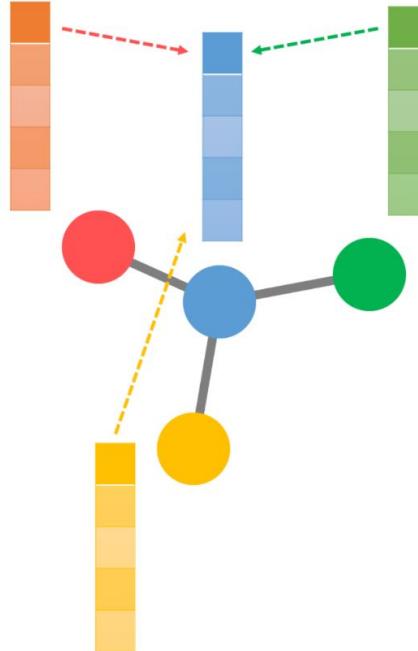
- Different inductive biases in neural networks



Component	Entities	Relations	Rel. inductive bias	Invariance
Fully connected	Units	All-to-all	Weak	-
Convolutional	Grid elements	Local	Locality	Spatial translation
Recurrent	Timesteps	Sequential	Sequentiality	Time translation
Graph network	Nodes	Edges	Arbitrary	Node, edge permutations

# Graph convolution

## Graph convolutional network



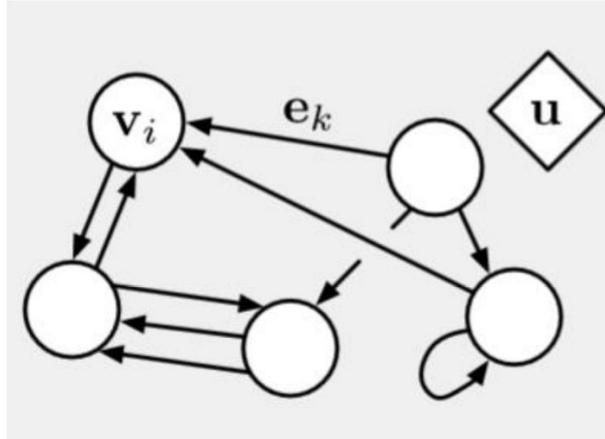
$$H^{(l+1)} = \sigma(AH^{(l)}W^{(l)})$$

Learnable parameter is shared

Sharing weights for all nodes in graph,  
but nodes are differently updated by reflecting individual node features,  $H_j^{(l)}$

# Graph neural network – Elements

## Graph neural networks



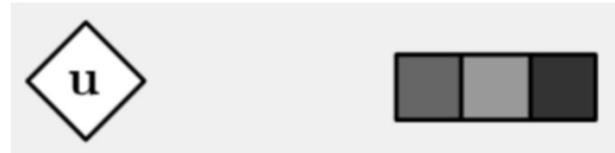
### Node's attribute



### Edge's attribute



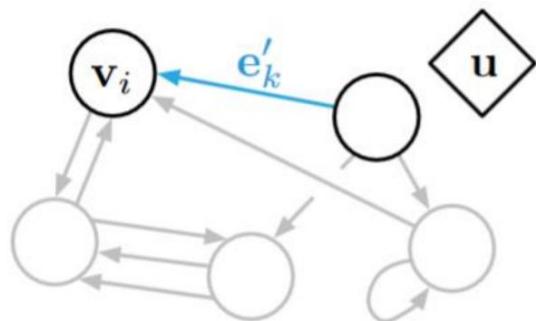
### Global attribute



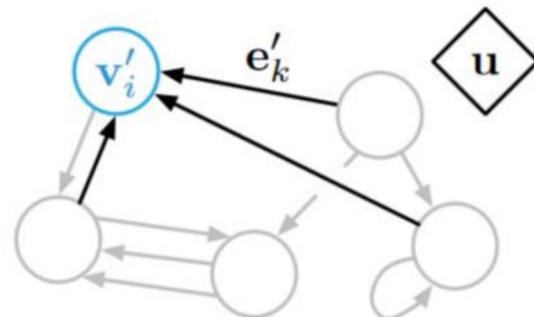
- ✓ Directed : one-way edges, from a “sender” node to a “receiver” node.
- ✓ Attribute : properties that can be encoded as a vector, set, or even another graph
- ✓ Attributed : edges and vertices have attributes associated with them

# Graph neural network – Elements

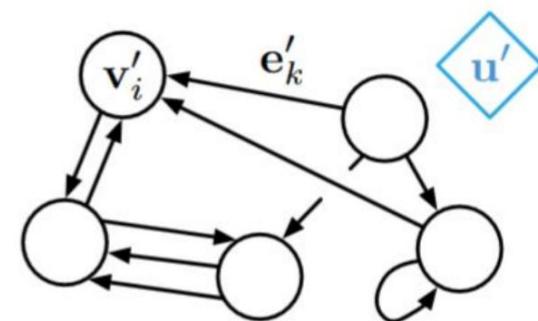
## GNN blocks



(a) Edge update



(b) Node update



(c) Global update

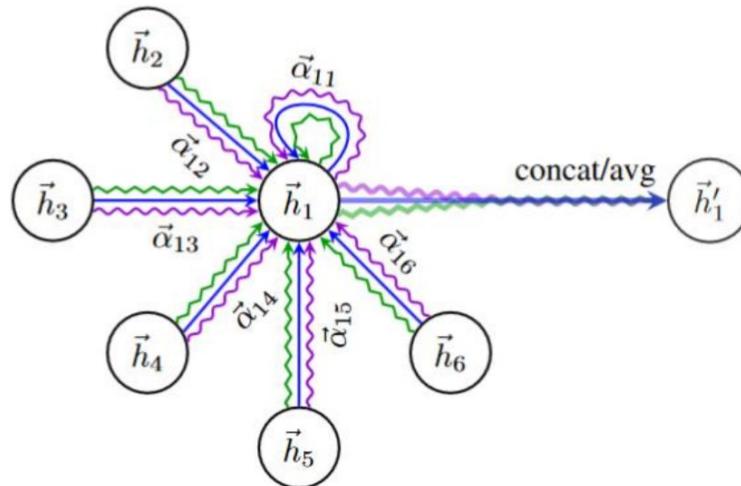
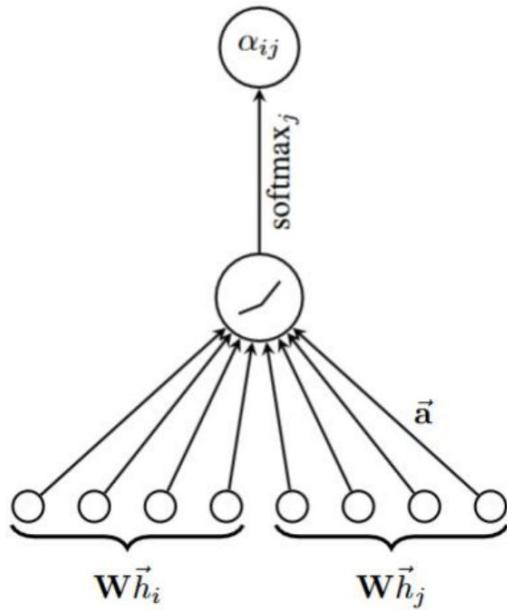
$$\mathbf{e}'_k = \text{NN}(\mathbf{v}_{s_k}, \mathbf{v}_{r_k}, \mathbf{e}_k, \mathbf{u})$$

$$\bar{\mathbf{e}}'_i = \sum_{k:r_k=i} \mathbf{e}'_k$$

$$\mathbf{v}'_i = \text{NN}(\bar{\mathbf{e}}'_i, \mathbf{v}_i, \mathbf{u})$$

# Graph neural network – Elements

## Case) Graph attention network



$$\mathbf{e}'_k = \text{NN}(\mathbf{v}_{r_k}, \mathbf{v}_{s_k})$$

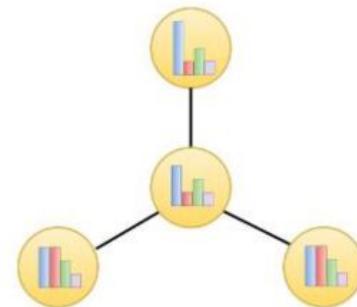
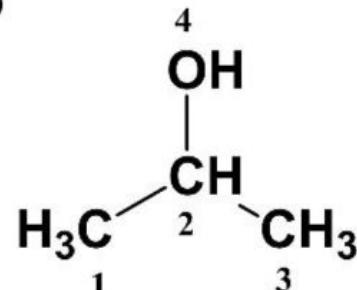
$$\mathbf{v}'_i = \sum_{k:r_k=i} \text{NN}(\mathbf{e}'_k, \mathbf{v}_{s_k})$$

# Graph neural network – overall architecture

## Preliminary) Molecular Graph

- Undirected graph,  $G(X, A)$  where  $X = H^{(0)} \in \mathbb{R}^{n \times d^{(0)}}$ ,  $A \in \mathbb{R}^{n \times n}$  and  $A_{ij} \in [0,1]$

(a)



(b)

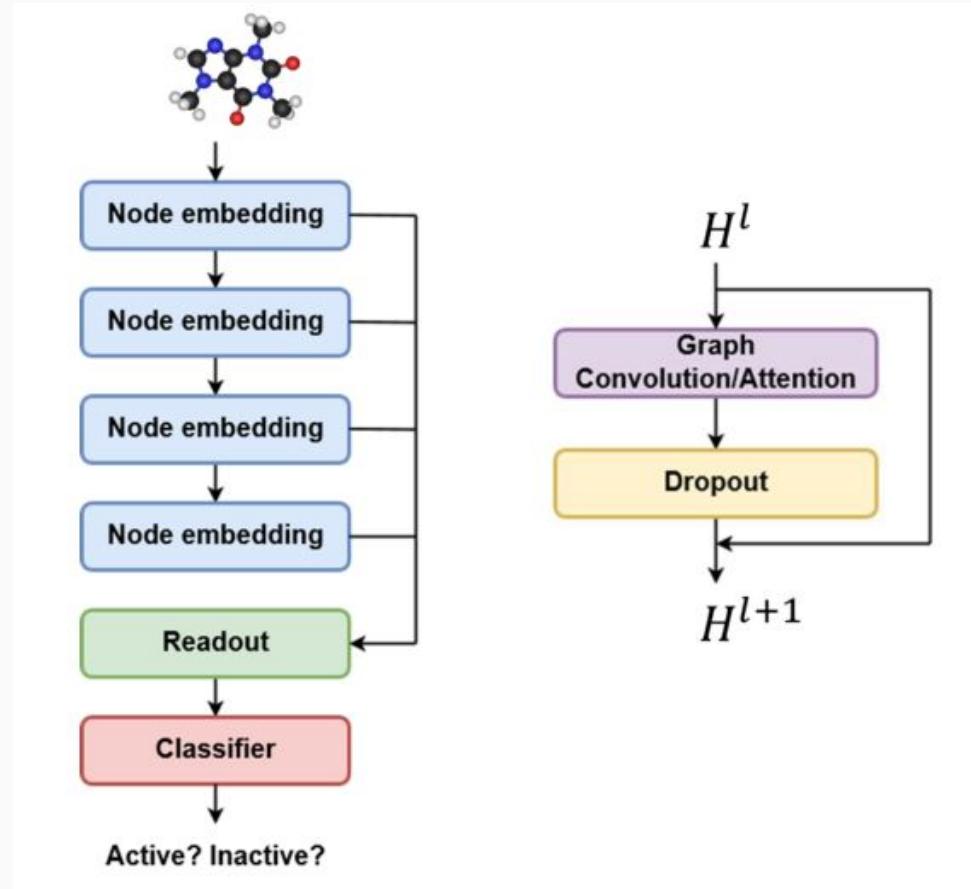
$$X_1 = \begin{bmatrix} 6 \\ 3 \\ 4 \\ 0 \end{bmatrix} \quad \dots \quad X_4 = \begin{bmatrix} 8 \\ 1 \\ 4 \\ 0 \end{bmatrix}$$

Atom type  
# of Hs.  
# Valence  
Aromaticity

$$A = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}$$

Input molecular graph of 2-propanol

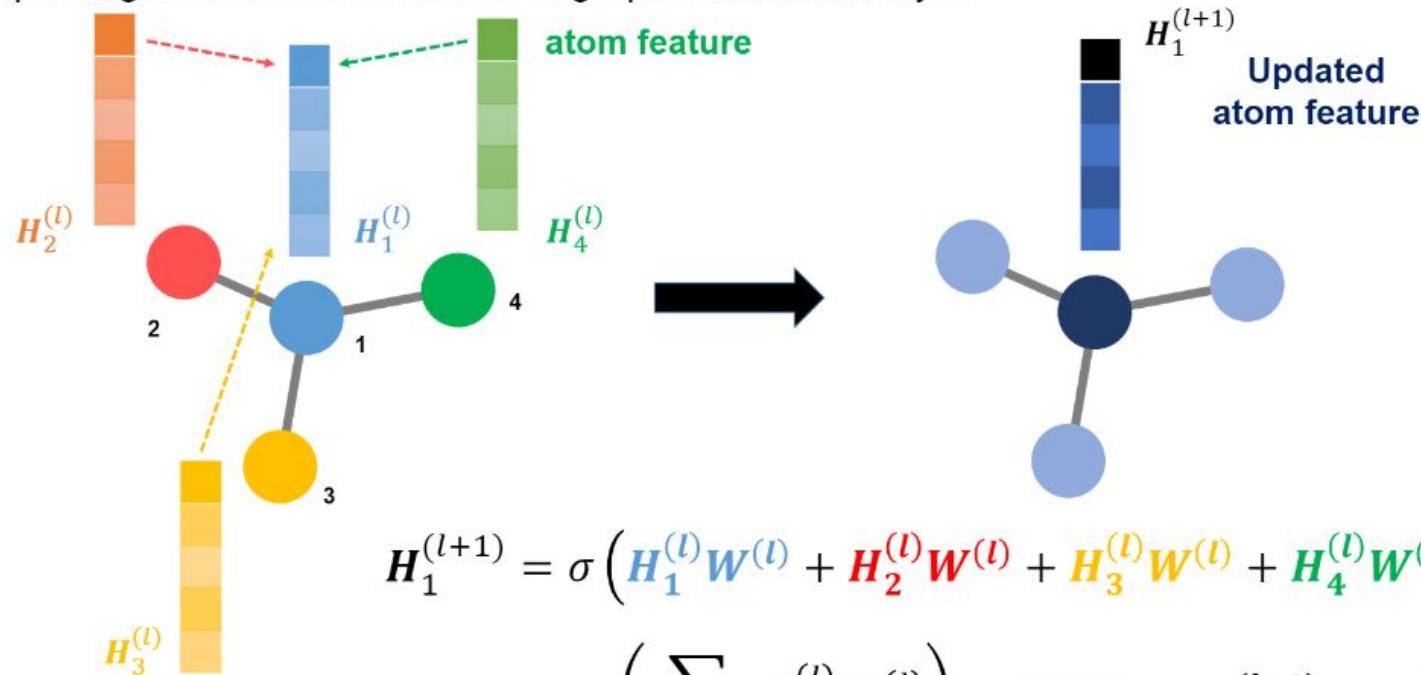
# Graph neural network – overall architecture



# Graph neural network – node embedding

## Preliminary) Graph convolutional networks

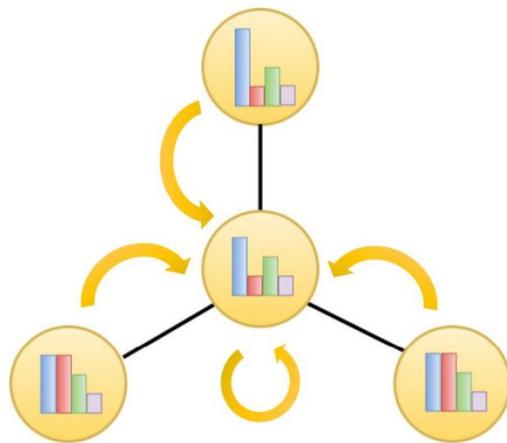
: updating node features at the  $l$ -th graph convolution layer



# Graph neural network – node embedding

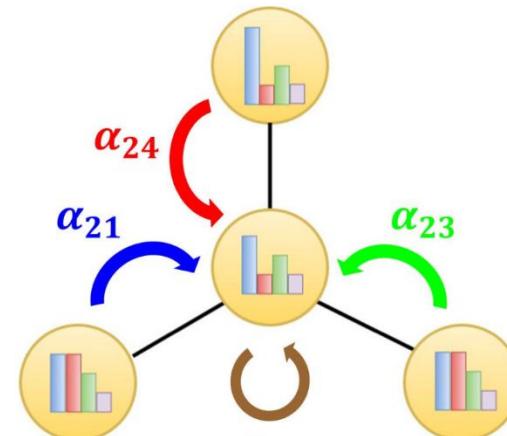
## Comparison between GCN and GAT

Vanilla GCN updates information of neighbor atoms **with same importance**.



$$\mathbf{H}^{(l+1)} = \sigma \left( \sum_{j \in N(i)} \mathbf{H}_j^{(l)} \mathbf{W}^{(l)} \right)$$

Attention mechanism enables GCN to update nodes **with different importance**.

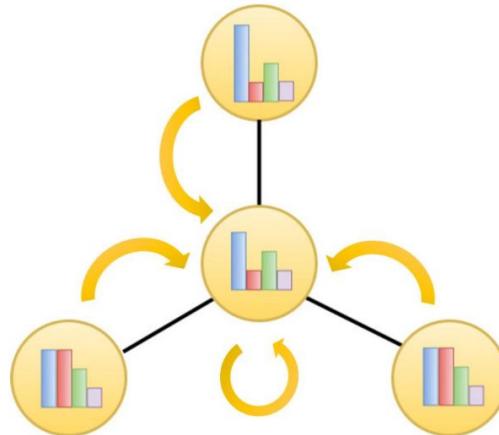


$$\mathbf{H}^{(l+1)} = \sigma \left( \sum_{j \in N(i)} \alpha_{ij} \mathbf{H}_j^{(l)} \mathbf{W}^{(l)} \right)$$

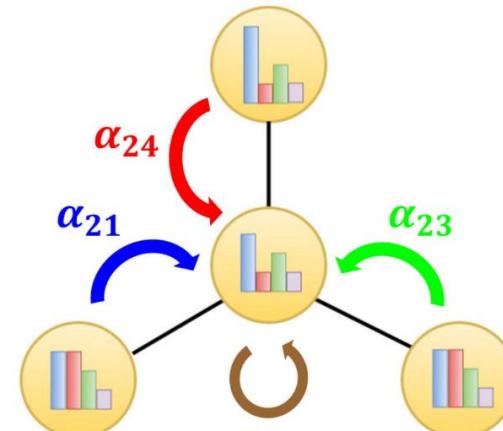
# Graph neural network – node embedding

## Comparison between GCN and GAT

The **attention** is nothing but **edge attribute** which find a relation between node attributes



$$H^{(l+1)} = \sigma \left( \sum_{j \in N(i)} H_j^{(l)} W^{(l)} \right)$$



$$H^{(l+1)} = \sigma \left( \sum_{j \in N(i)} \alpha_{ij} H_j^{(l)} W^{(l)} \right)$$

## Graph neural network – readout

**Readout : aggregating node features to map input graphs to graph features**

$$z^{(l)} = \text{Readout} \left( \left\{ H_i^{(l)} \right\} \right)$$

- Average pooling

$$z^{(l)} = \frac{1}{N} \sum_{i=1}^N H_i^{(l)} W_g^{(l)}, \quad W_g^{(l)} \in \mathbb{R}^{d^{(l)} \times d_g^{(l)}}$$

- Sum pooling

$$z^{(l)} = \sum_{i=1}^N H_i^{(l)} W_g^{(l)}, \quad W_g^{(l)} \in \mathbb{R}^{d^{(l)} \times d_g^{(l)}}$$

## Graph neural network – readout

**Readout : aggregating node features to map input graphs to graph features**

$$z^{(l)} = \text{Readout} \left( \left\{ H_i^{(l)} \right\} \right)$$

- Average pooling cannot give correct summary statistics of node features

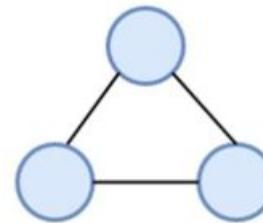
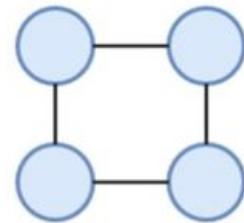
$$z_1 = \frac{1}{3}(h + h + h) = h$$

$$z_2 = \frac{1}{4}(h + h + h + h) = h$$

- Sum pooling gives correct summary statistics of node features

$$z_1 = (h + h + h) = 3h$$

$$z_2 = (h + h + h + h) = 4h$$

 $\mathcal{G}_1$  $\mathcal{G}_2$

## Graph neural network – readout

**Readout : aggregating node features to map input graphs to graph features**

$$z^{(l)} = \text{Readout} \left( \left\{ H_i^{(l)} \right\} \right)$$

- Attention pooling can summarize node features with different (learnable) weights

$$z^{(l)} = \sum_{i=1}^N \alpha_i^{(l)} H_i^{(l)} W_g^{(l)}, \quad \alpha_i^{(l)} = N \times \text{softmax} \left( \frac{1}{\sqrt{d_g^{(l)}}} \right) (\mathbf{1}) \left( H_i^{(l)} W_g^{(l)} \right)^T, \quad W_g^{(l)} \in \mathbb{R}^{d^{(l)} \times d_g^{(l)}}$$

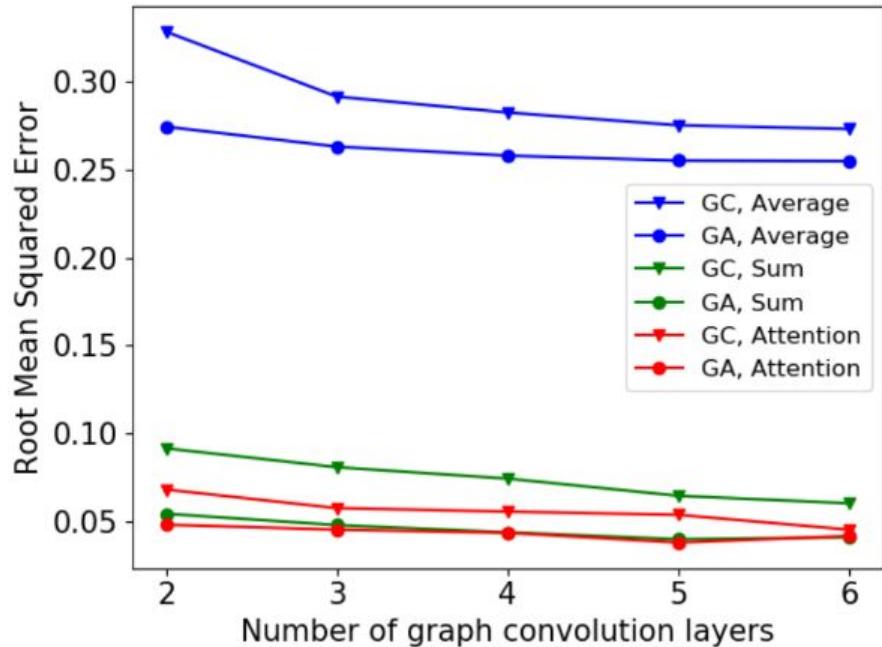
- Concatenation can summarize hierarchical sub-graph structures

$$z_G = \text{Concat}([z^{(1)}, \dots, z^{(L)}])$$

# Graph neural network – simple toy experiments

## Result 1) Self-attention for node embedding and readout improves prediction performances

- Toy example: logP prediction
- Augmenting the self-attention for node-embedding and readout improves prediction performances.
- Average pooling significantly underperforms sum and attention pooling.
- Attention pooling outperforms the other readouts.



- ✓ # training : 80,000
- ✓ # test : 20,000

# Graph neural network – simple toy experiments

## Result 2) Other prediction tasks

✓ # training : 80,000  
✓ # test : 20,000

		LogP		TPSA		SAS	
		Concat	Last	Concat	Last	Concat	Last
GC	Mean	0.283	0.280	6.01	5.76	0.104	0.108
	Sum	0.074	0.083	0.52	0.62	0.068	0.066
	Attention	0.055	0.077	0.42	0.50	0.060	0.063
GA	Mean	0.258	0.261	5.88	0.56	0.091	0.098
	Sum	0.044	0.054	0.53	0.55	0.057	0.057
	Attention	<b>0.043</b>	0.055	<b>0.52</b>	0.60	<b>0.053</b>	0.054

- Augmenting the self-attention for node-embedding and readout improves prediction performances.
- Concatenation of all graph features at  $l = 1, \dots, 4$  layers outperforms using only the last graph feature.

# Graph neural network – simple toy experiments

## Result 3) Binary classification tasks

# of samples (5-fold random splitting with 80:20 ratio)

- ✓ BACE: 1,513
- ✓ BBBP: 2,050
- ✓ HIV: 41,127

	Architecture	Accuracy ( $\uparrow$ )	AUROC ( $\uparrow$ )	F1-score ( $\uparrow$ )	ECE ( $\downarrow$ )	OCE ( $\downarrow$ )
BACE	GCN+Sum	0.809	0.878	0.780	0.102	0.086
	GCN+Attn	<b>0.822</b>	<b>0.897</b>	<b>0.802</b>	<b>0.091</b>	<b>0.065</b>
	GAT+Sum	0.793	0.879	0.764	0.166	0.139
	GAT+Attn	0.799	0.880	0.781	0.174	0.202
BBBP	GCN+Sum	0.890	0.915	0.929	<b>0.066</b>	<b>0.082</b>
	GCN+Attn	<b>0.892</b>	<b>0.919</b>	<b>0.931</b>	0.087	0.235
	GAT+Sum	0.864	0.902	0.913	0.120	0.211
	GAT+Attn	0.871	0.898	0.917	0.115	0.169
HIV	GCN+Sum	0.970	0.805	0.392	<b>0.008</b>	0.008
	GCN+Attn	<b>0.971</b>	<b>0.816</b>	<b>0.438</b>	0.010	<b>0.007</b>
	GAT+Sum	0.965	0.797	0.425	0.030	0.021
	GAT+Attn	0.970	0.812	0.410	0.010	0.008

- Using attention in node update involves over-fitting
- More parameterization discourage reliability (see more details in later)

# Understanding GNN – Graph Isomorphism Network

- Xu, Keyulu, et al. "How powerful are graph neural networks?." arXiv preprint arXiv:1810.00826 (2018).

Published as a conference paper at ICLR 2019

---

## HOW POWERFUL ARE GRAPH NEURAL NETWORKS?

**Keyulu Xu** \*†

MIT

keyulu@mit.edu

**Weihua Hu** \*‡

Stanford University

weihuahu@stanford.edu

**Jure Leskovec**

Stanford University

jure@cs.stanford.edu

**Stefanie Jegelka**

MIT

stefje@mit.edu

# Understanding GNN – Graph Isomorphism Network

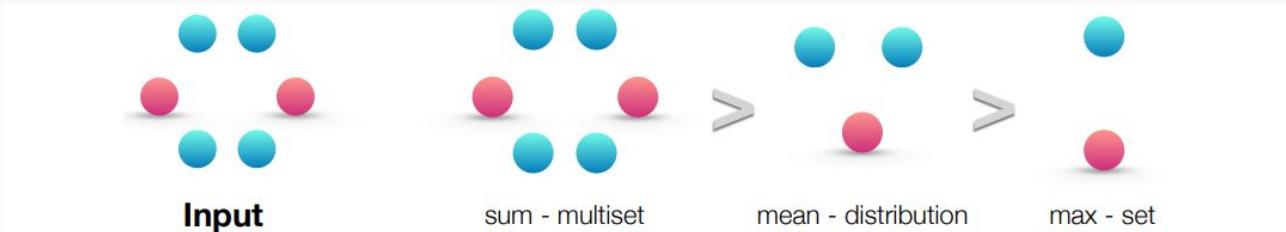


Figure 2: **Ranking by expressive power for sum, mean and max aggregators over a multiset.** Left panel shows the input multiset, i.e., the network neighborhood to be aggregated. The next three panels illustrate the aspects of the multiset a given aggregator is able to capture: sum captures the full multiset, mean captures the proportion/distribution of elements of a given type, and the max aggregator ignores multiplicities (reduces the multiset to a simple set).

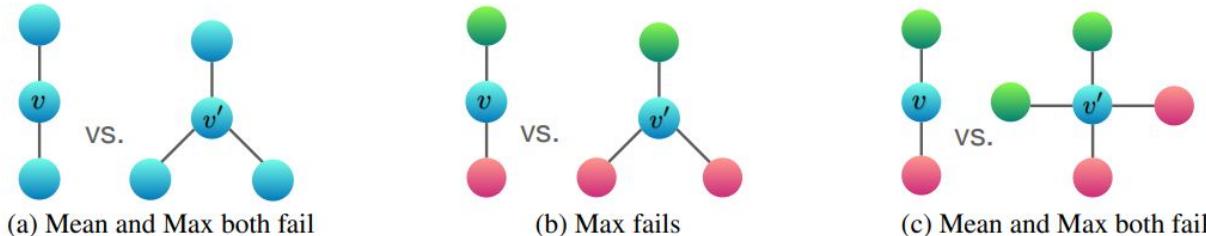


Figure 3: **Examples of graph structures that mean and max aggregators fail to distinguish.** Between the two graphs, nodes  $v$  and  $v'$  get the same embedding even though their corresponding graph structures differ. Figure 2 gives reasoning about how different aggregators “compress” different multisets and thus fail to distinguish them.

## Understanding GNN – Graph Isomorphism Network

**Corollary 6.** Assume  $\mathcal{X}$  is countable. There exists a function  $f : \mathcal{X} \rightarrow \mathbb{R}^n$  so that for infinitely many choices of  $\epsilon$ , including all irrational numbers,  $h(c, X) = (1 + \epsilon) \cdot f(c) + \sum_{x \in X} f(x)$  is unique for each pair  $(c, X)$ , where  $c \in \mathcal{X}$  and  $X \subset \mathcal{X}$  is a multiset of bounded size. Moreover, any function  $g$  over such pairs can be decomposed as  $g(c, X) = \varphi((1 + \epsilon) \cdot f(c) + \sum_{x \in X} f(x))$  for some function  $\varphi$ .

Universal function approximator : MLP

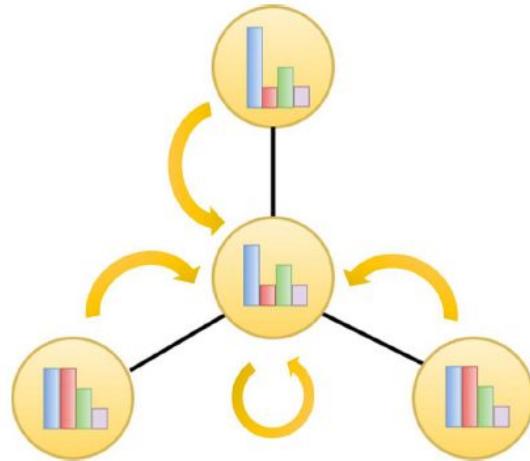
$$h_v^{(k)} = \text{MLP}^{(k)} \left( \left(1 + \epsilon^{(k)}\right) \cdot h_v^{(k-1)} + \sum_{u \in \mathcal{N}(v)} h_u^{(k-1)} \right).$$

- GCN: Linear layer and non-linear activation (e.g. ReLU)  
→ Less expressive power than GIN

# **Self-attention for GNN**

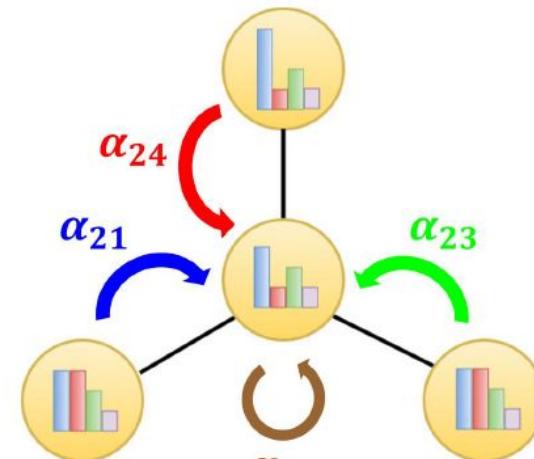
# Graph Convolution vs Graph Attention

Vanilla GCN updates information of neighbor atoms **with same importance**.



$$H^{(l+1)} = \sigma \left( \sum_{j \in N(i)} H_j^{(l)} W^{(l)} \right)$$

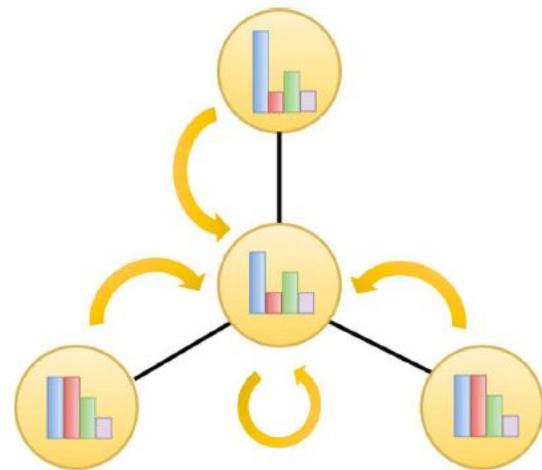
Attention mechanism enables GCN to update nodes **with different importance**.



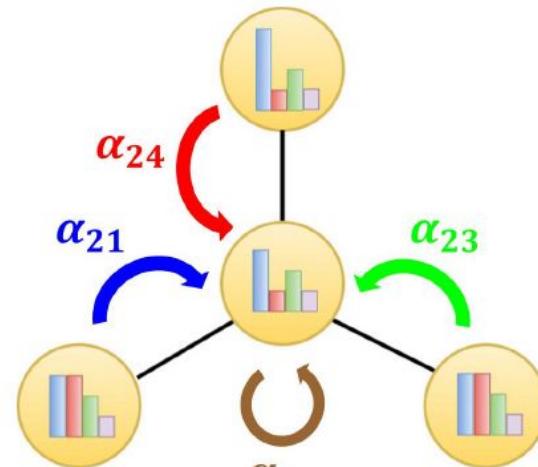
$$H^{(l+1)} = \sigma \left( \sum_{j \in N(i)} \alpha_{ij} H_j^{(l)} W^{(l)} \right)$$

# Graph Convolution vs Graph Attention

The **attention** is nothing but **edge attribute** which find a relation between node attributes



$$H^{(l+1)} = \sigma \left( \sum_{j \in N(i)} H_j^{(l)} W^{(l)} \right)$$

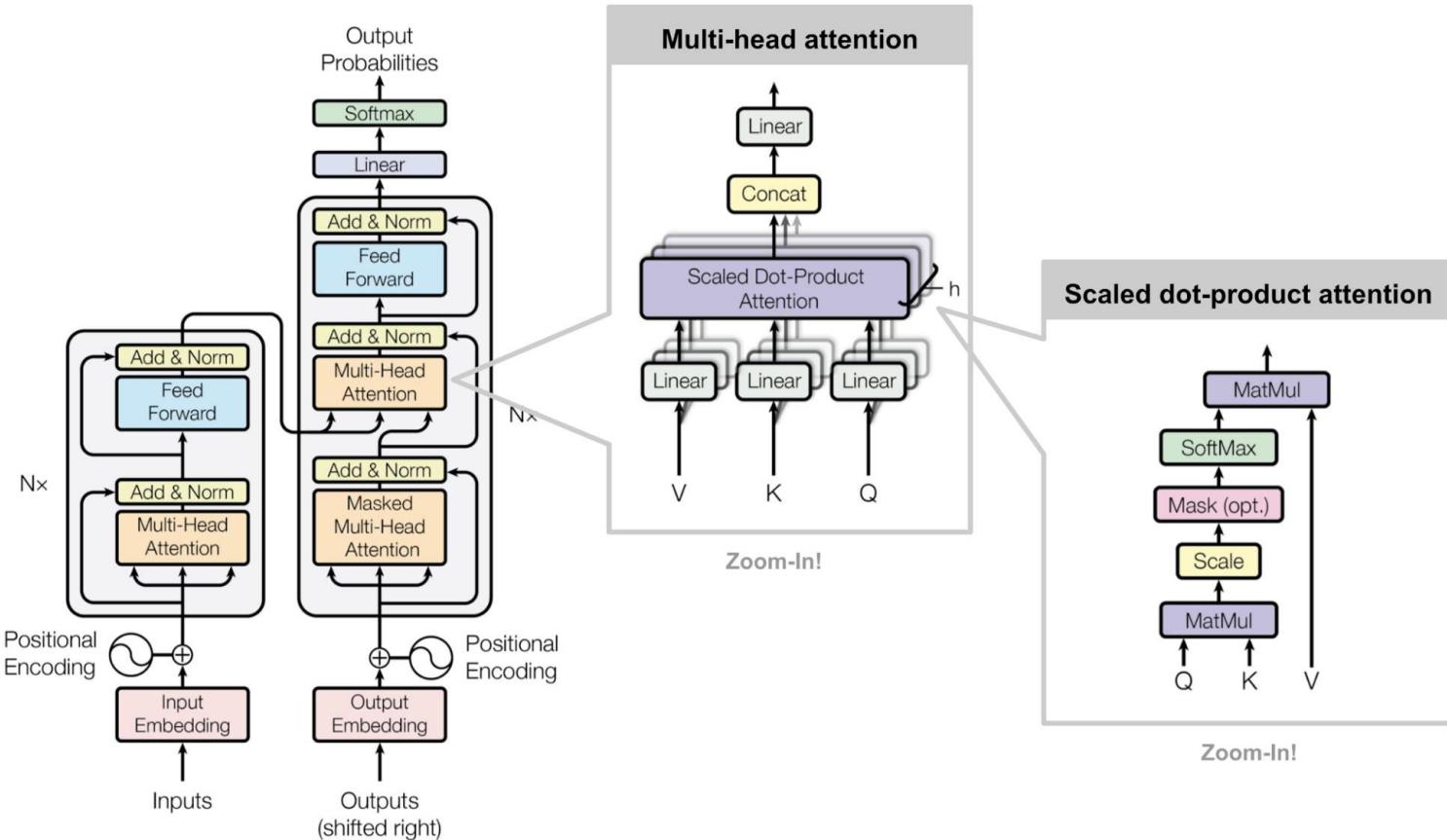


$$H^{(l+1)} = \sigma \left( \sum_{j \in N(i)} \alpha_{ij} H_j^{(l)} W^{(l)} \right)$$

## Attention score

Name	Alignment score function	Citation
Additive	$\text{score}(\mathbf{s}_t, \mathbf{h}_i) = \mathbf{v}_a^T \tanh(\mathbf{W}_a[\mathbf{s}_t; \mathbf{h}_i])$	Bahdanau 2015
Location-Base	$\alpha_{t,i} = \text{softmax}(\mathbf{W}_a \mathbf{s}_t)$ Note : This simplifies the softmax alignment max to only depend on the target position.	Luong 2015
General	$\text{score}(\mathbf{s}_t, \mathbf{h}_i) = \mathbf{s}_t^T \mathbf{W}_a \mathbf{h}_i$	Luong 2015
Dot-Product	$\text{score}(\mathbf{s}_t, \mathbf{h}_i) = \mathbf{s}_t^T \mathbf{h}_i$	Luong 2015
Scaled Dot-Product	$\text{score}(\mathbf{s}_t, \mathbf{h}_i) = \frac{\mathbf{s}_t^T \mathbf{h}_i}{\sqrt{n}}$ where $n$ is the dimension of source hidden state	Vaswani 2017
Self-Attention	Relating different positions of the same input sequence. Theoretically the self-attention can adopt any score functions above, just replace	Cheng 2016
Global/Soft	Attending to the entire input state space	Xu 2015
Local/Hard	Attending to the part of input state space; i.e. a patch of the input image.	Xu 2015 Luong 2015

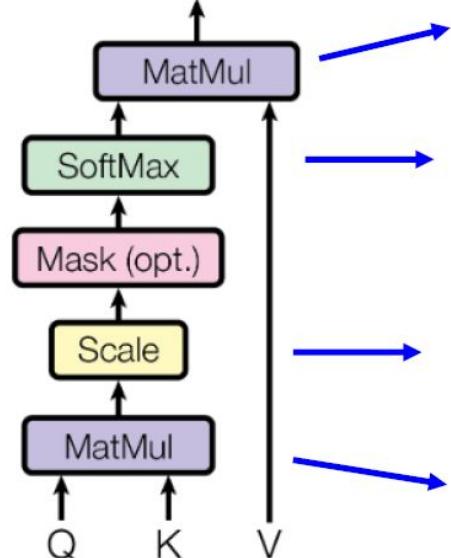
# Self Attention - Transformer



# Self Attention - Transformer

## Self-attention

### Scaled Dot-Product Attention



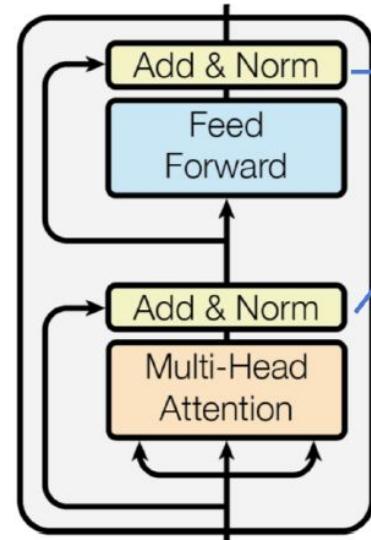
$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V} \in \mathbb{R}^{d_{model} \times d_v}$$

$$\text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)$$

$\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}$  : assume that the components of  $q$  and  $k$  are independent random variables with mean 0 and variance 1. Then their dot product,  $q \cdot k = \sum_{i=1}^{d_k} q_i k_i$ , has mean 0 and variance  $d_k$ .

$\mathbf{Q}\mathbf{K}^T \in \mathbb{R}^{d_{model} \times d_{model}}$  : dot-product attention is much faster and more space-efficient in practice.

# Self Attention - Transformer



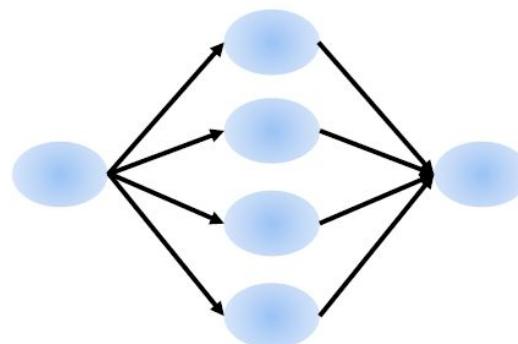
$\text{LayerNorm}(x + \text{SubLayer}(x))$

$$\text{FFN}(x) = \text{ReLU}(x\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2$$

$$\mathbf{x} \in \mathbb{R}^{d_{\text{model}} \times d_v}$$

$$\mathbf{W}_1 \in \mathbb{R}^{d_v \times m * d_v}$$

$$\mathbf{W}_2 \in \mathbb{R}^{m * d_v \times d_v}$$



Published as a conference paper at ICLR 2018

---

## GRAPH ATTENTION NETWORKS

**Petar Veličković\***

Department of Computer Science and Technology  
University of Cambridge  
[petar.velickovic@cst.cam.ac.uk](mailto:petar.velickovic@cst.cam.ac.uk)

**Guillem Cucurull\***

Centre de Visió per Computador, UAB  
[gcucurull@gmail.com](mailto:gcucurull@gmail.com)

**Arantxa Casanova\***

Centre de Visió per Computador, UAB  
[ar.casanova.8@gmail.com](mailto:ar.casanova.8@gmail.com)

**Adriana Romero**

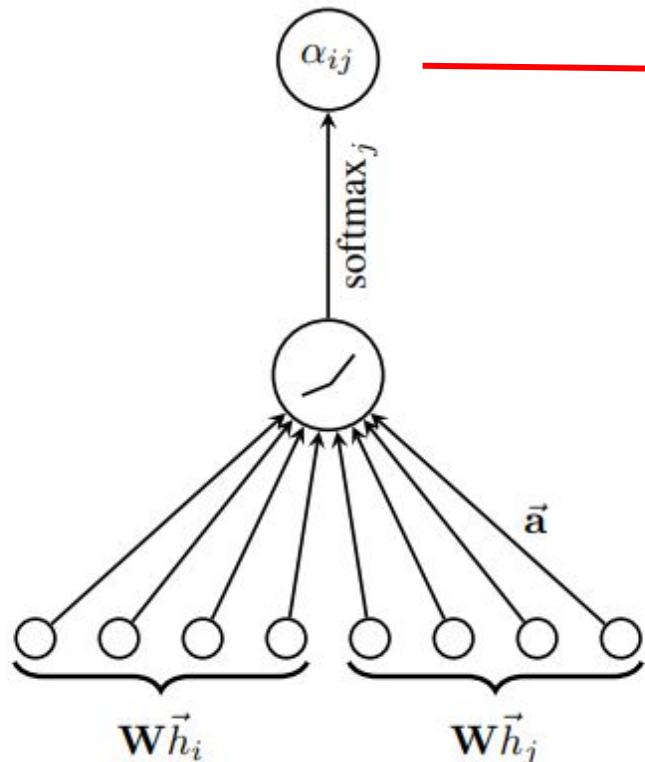
Montréal Institute for Learning Algorithms  
[adriana.romero.soriano@umontreal.ca](mailto:adriana.romero.soriano@umontreal.ca)

**Pietro Liò**

Department of Computer Science and Technology  
University of Cambridge  
[pietro.lio@cst.cam.ac.uk](mailto:pietro.lio@cst.cam.ac.uk)

**Yoshua Bengio**

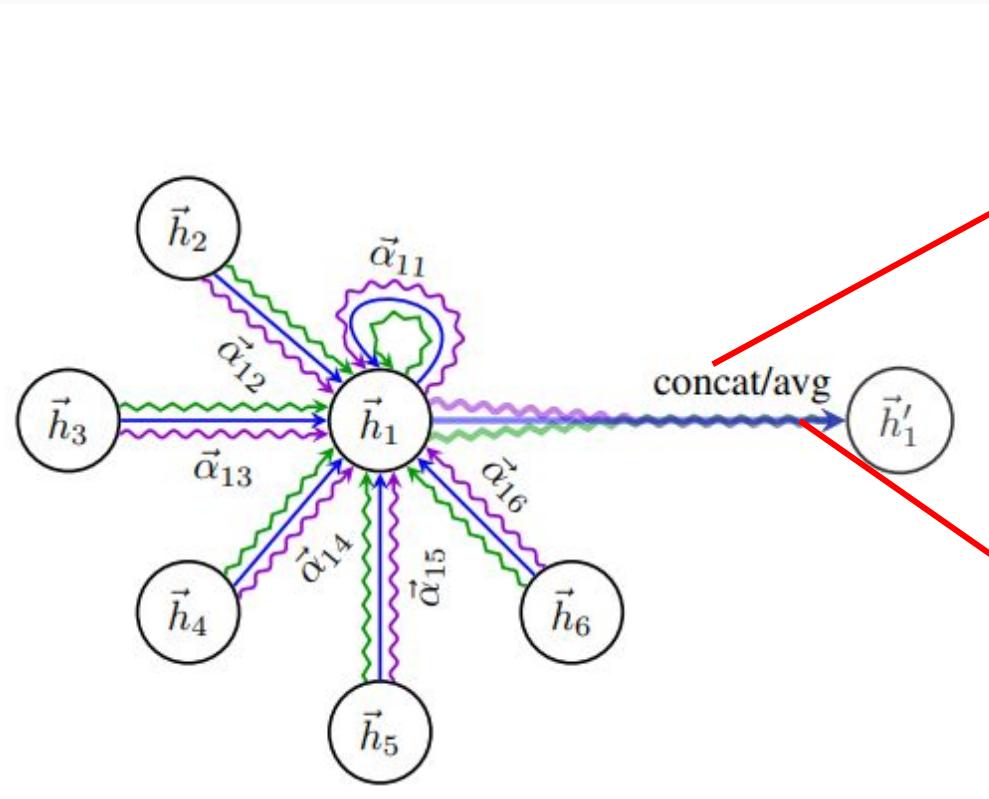
Montréal Institute for Learning Algorithms  
[yoshua.umontreal@gmail.com](mailto:yoshua.umontreal@gmail.com)



$$\alpha_{ij} = \text{softmax}_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}_i} \exp(e_{ik})}.$$

$$e_{ij} = a(\vec{W}\vec{h}_i, \vec{W}\vec{h}_j)$$

$$\alpha_{ij} = \frac{\exp \left( \text{LeakyReLU} \left( \vec{a}^T [\vec{W}\vec{h}_i \| \vec{W}\vec{h}_j] \right) \right)}{\sum_{k \in \mathcal{N}_i} \exp \left( \text{LeakyReLU} \left( \vec{a}^T [\vec{W}\vec{h}_i \| \vec{W}\vec{h}_k] \right) \right)}$$



$$\vec{h}'_i = \left\| \sigma \left( \sum_{j \in \mathcal{N}_i} \alpha_{ij}^k \mathbf{W}^k \vec{h}_j \right) \right\|$$

$$\vec{h}'_i = \sigma \left( \frac{1}{K} \sum_{k=1}^K \sum_{j \in \mathcal{N}_i} \alpha_{ij}^k \mathbf{W}^k \vec{h}_j \right)$$

- Query, Key, Value ?
  - Query: source node
  - Key: neighbor node
  - Value: neighbor node
  - Attention coefficient: importance of neighbor node w.r.t. a given source node
- Attention scoring method ?
  - Original GAT paper: Concat & Linear method
    - This method is not cost-efficient.
  - Transformer: Self-attention (scaled-dot product)

# Transformer Conv in PyTorch geometric

[https://pytorch-geometric.readthedocs.io/en/latest/modules/nn.html#torch\\_geometric.nn.conv.TransformerConv](https://pytorch-geometric.readthedocs.io/en/latest/modules/nn.html#torch_geometric.nn.conv.TransformerConv)

CLASS TransformerConv ( `in_channels: Union[int, Tuple[int, int]]`, `out_channels: int`, `heads: int = 1`, `concat: bool = True`, `beta: bool = False`, `dropout: float = 0.0`, `edge_dim: Optional[int] = None`, `bias: bool = True`, `root_weight: bool = True`, `**kwargs` )     [\[source\]](#)



## Without edge-feature

$$\mathbf{x}'_i = \mathbf{W}_1 \mathbf{x}_i + \sum_{j \in \mathcal{N}(i)} \alpha_{i,j} \mathbf{W}_2 \mathbf{x}_j,$$

where the attention coefficients  $\alpha_{i,j}$  are computed via multi-head dot product attention:

$$\alpha_{i,j} = \text{softmax} \left( \frac{(\mathbf{W}_3 \mathbf{x}_i)^\top (\mathbf{W}_4 \mathbf{x}_j)}{\sqrt{d}} \right)$$

## With edge-feature

$$\mathbf{x}'_i = \mathbf{W}_1 \mathbf{x}_i + \sum_{j \in \mathcal{N}(i)} \alpha_{i,j} (\mathbf{W}_2 \mathbf{x}_j + \mathbf{W}_6 \mathbf{e}_{ij}),$$

where the attention coefficients  $\alpha_{i,j}$  are now computed via:

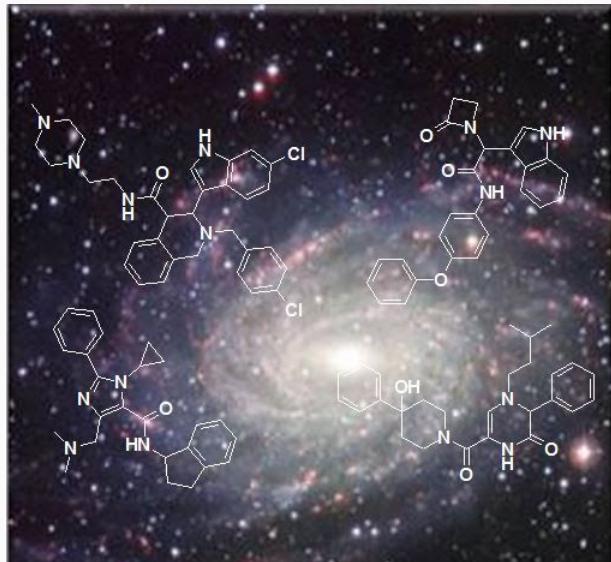
$$\alpha_{i,j} = \text{softmax} \left( \frac{(\mathbf{W}_3 \mathbf{x}_i)^\top (\mathbf{W}_4 \mathbf{x}_j + \mathbf{W}_6 \mathbf{e}_{ij})}{\sqrt{d}} \right)$$

- 구현해볼 GNN with attention model 은 위의 “With edge-feature” case

# Bayesian Learning

# Data-hungry problem in molecular science

Size of chemical space (Enamine REAL):  $10^{10}$



>>



Tox21: dataset for toxicity prediction  
# of data points:  $< 10^4$



DUD-E: dataset for binding affinity prediction  
# of data points:  $O(10^3) \sim O(10^4)$

- Usually, dataset size is much smaller than the chemical space (or the size of virtual screening library).
- Also, noisy labels are obtained because of stochastic nature of biochemical assays.
- Thus, using such datasets make models prone to poor generalization (over-fitting problem)

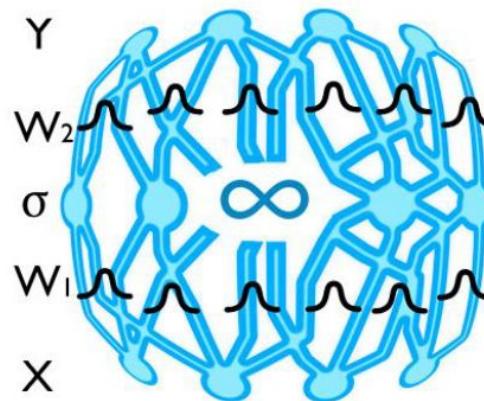
# Statistical Inference and Bayesian Inference

## Bayesian's eye

: how to do inference about **hypotheses (uncertain quantities)** from **data (measured quantities)**.

$$p(\text{hypothesis}|\text{data}) = \frac{p(\text{data}|\text{hypothesis}) \times p(\text{hypothesis})}{P(\text{data})}$$

<https://www.youtube.com/watch?v=FD8I2vPU5FY&t=2s>



Bayesian let all quantities, except data, as uncertain quantities.

[http://www.cs.ox.ac.uk/people/yarin.gal/website/blog\\_3d801aa532c1ce.html](http://www.cs.ox.ac.uk/people/yarin.gal/website/blog_3d801aa532c1ce.html)

# Statistical Inference and Bayesian Inference

Training a certain neural network model is equivalent to obtaining a posterior  $p(\omega|X, Y)$ .

$$p(\omega|X, Y) = \frac{\text{likelihood} \quad \text{prior}}{\text{evidence (or marginal likelihood)}}$$
$$= \frac{p(Y|X, \omega)p(\omega)}{p(Y|X)}$$

In general, the model is obtained by solving optimization problem.

$$\mathcal{L}(\omega) = \log p(Y|X, \omega) + \log p(\omega)$$

If we assume that the likelihood is Gaussian distribution  $p(Y|X, \omega) \propto \exp\left(\frac{(Y - f^\omega(X))^2}{2\sigma^2}\right)$  and also the prior is Gaussian distribution  $p(\omega) \propto \exp\left(\frac{\|\omega\|^2}{2l^2}\right)$ , then the minimization objective is

$$\mathcal{L}(\omega) = (Y - f^\omega(X))^2 + \frac{\|\omega\|^2}{2l^2} + \text{const.}$$

L2-norm (MSE)                    L2-regularization  
 $l$ : prior length scale

# Statistical Inference and Bayesian Inference

**Frequentist inference:** model parameters and predicted output is **deterministic**.

- Frequentist model estimation:  $\hat{\omega} = \underset{\omega \in \Omega}{\operatorname{argmax}} \mathcal{L}(\omega)$
- Maximum-a-posteriori (MAP) estimation:  $\hat{\omega} = \underset{\omega \in \Omega}{\operatorname{argmax}} p(Y|X, \omega) p(\omega)$
- Maximum-likelihood estimation (MLE) :  $\hat{\omega} = \underset{\omega \in \Omega}{\operatorname{argmax}} p(Y|X, \omega)$
- Frequentist inference:  $y^* = f^{\hat{\omega}}(x^*)$

Assuming prior has a uniform distribution

**Bayesian inference:** model parameters and predicted output **are probabilistic (have distributions)**.  
In other words, it allows to model ‘uncertainty’ over parameters.

$$p(y^*|x^*, X, Y) = \int p(y^*|x^*, \omega)p(\omega|X, Y) d\omega$$

In this reason, we can estimate the uncertainty of our prediction by measuring the variance of predictive distribution  $p(y^*|x^*, X, Y)$ .

# Statistical Inference and Bayesian Inference

Inference from maximum-a-posteriori (MAP) model estimation

Choosing the posterior  
which can explain the given data distribution the best.

$$p(y^*|x^*) = f^{\hat{w}}(x^*) \quad \leftarrow \quad \hat{w} = \operatorname{argmax}_{w \in \mathcal{W}} p(w|X, Y)$$

Then, our inference will be a single deterministic value.

Inference from Bayesian model estimation

Summation over all possible model posteriors

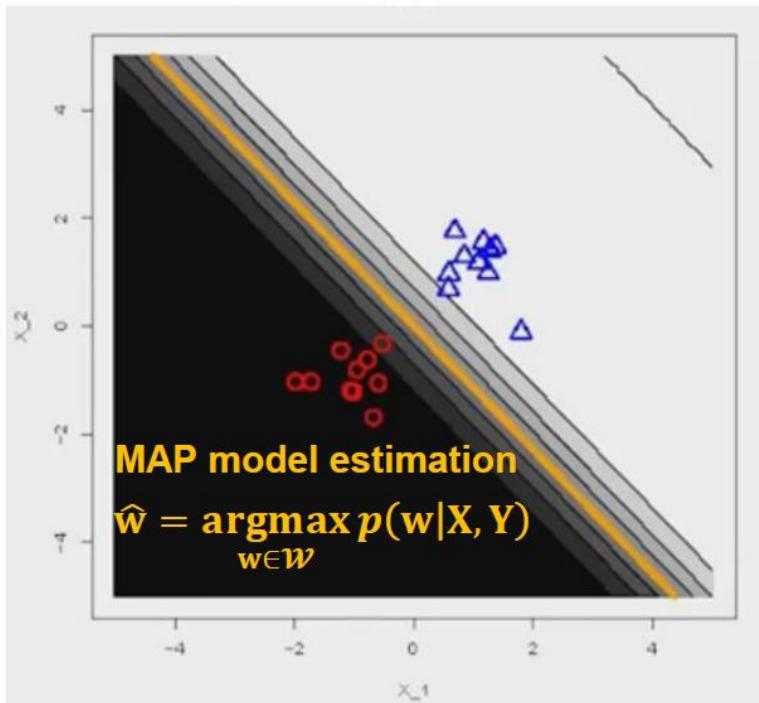
$$p(y^*|x^*, X, Y) = \int_{w \in \mathcal{W}} p(y^*|x^*, w) p(w|X, Y) dw$$

Then, our inference will have a distribution instead of a single deterministic value.

→ The uncertainty on model parameter give the uncertainty on our inference

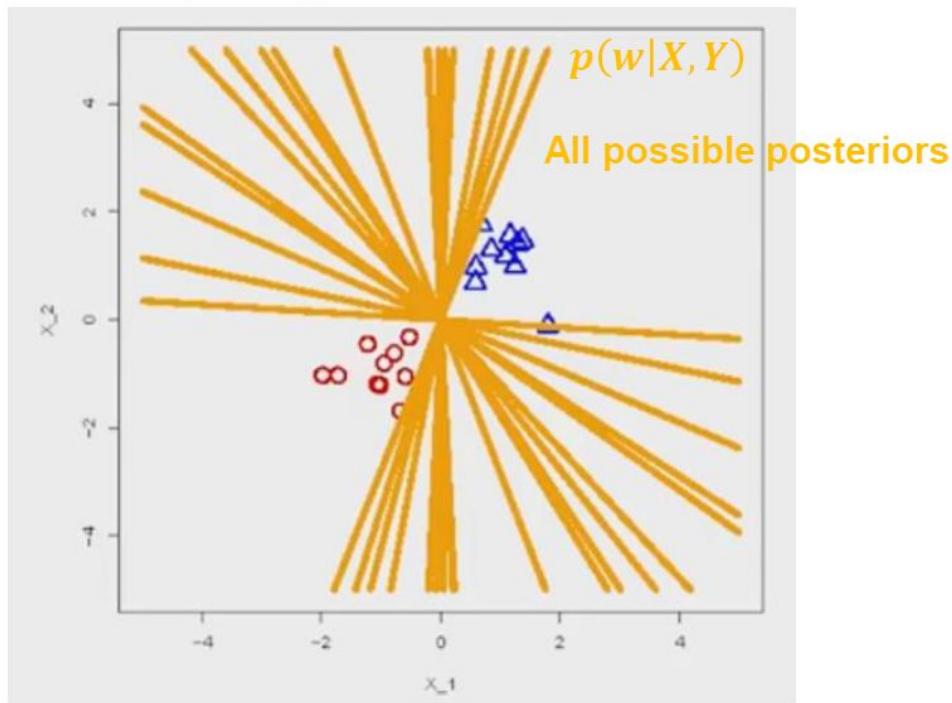
# Statistical Inference and Bayesian Inference

MAP predictive distribution



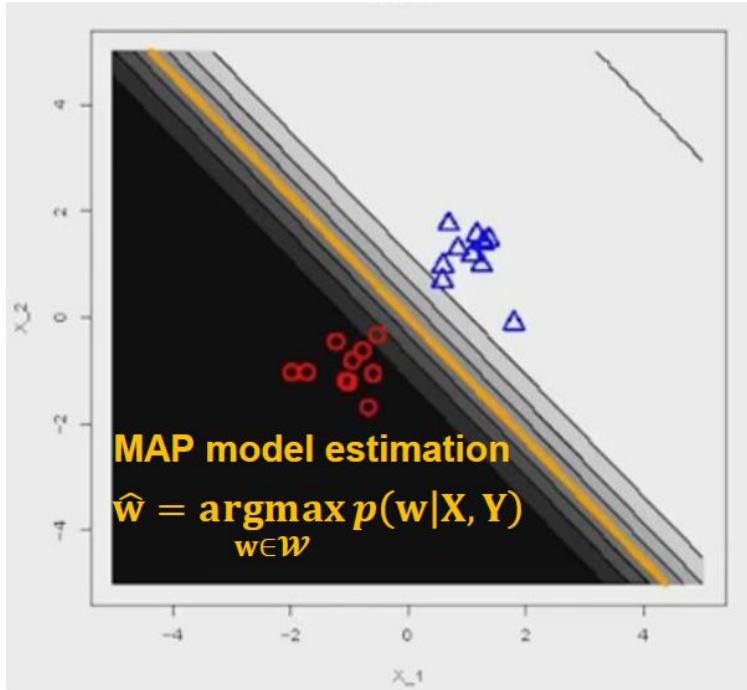
$$p(y^*|x^*) = f^{\hat{w}}(x^*)$$

Bayesian predictive distribution



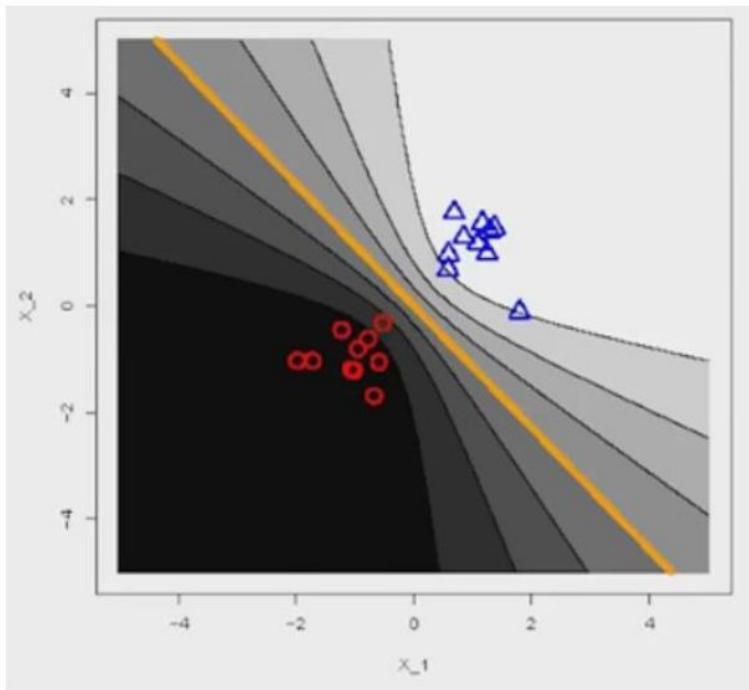
# Statistical Inference and Bayesian Inference

MAP predictive distribution



$$p(y^* | x^*) = f^{\hat{\mathbf{w}}}(x^*)$$

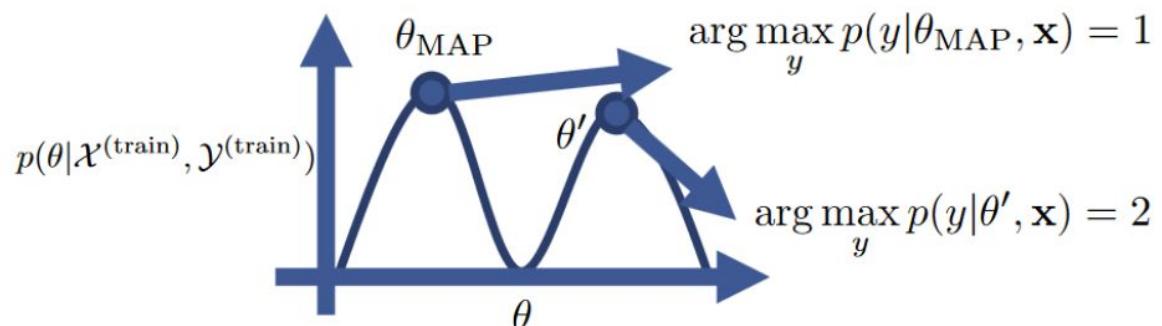
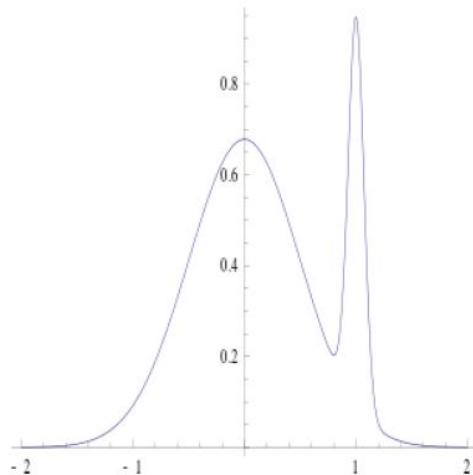
Bayesian predictive distribution



$$p(y^* | x^*, \mathbf{X}, \mathbf{Y}) = \int_{\mathbf{w} \in \mathcal{W}} p(y^* | x^*, \mathbf{w}) p(\mathbf{w} | \mathbf{X}, \mathbf{Y}) d\mathbf{w}$$

# Statistical Inference and Bayesian Inference

## Limitations of Frequentist's model estimation:



- ✓ MAP estimation only considers a single point estimate:  
It may fail when the posterior have a multi-modal distribution in which the highest mode is uncharacteristic of the majority of the distribution.

# Statistical Inference and Bayesian Inference

## Limitations of Bayesian's model estimation:

**Bayesian inference:** model parameters and predicted output are probabilistic (have distributions)

$$p(\boldsymbol{\omega}|\mathbf{X}, \mathbf{Y}) = \frac{p(\mathbf{Y}|\mathbf{X}, \boldsymbol{\omega})p(\boldsymbol{\omega})}{p(\mathbf{Y}|\mathbf{X})}$$

likelihood                  prior  
                                  evidence (or marginal likelihood)

$$p(\mathbf{y}^*|\mathbf{x}^*, \mathbf{X}, \mathbf{Y}) = \int p(\mathbf{y}^*|\mathbf{x}^*, \boldsymbol{\omega})p(\boldsymbol{\omega}|\mathbf{X}, \mathbf{Y}) d\boldsymbol{\omega}$$

✓ Choice of prior

: Is assuming prior as Gaussian distribution correct? How can we choose a good prior?

✓ Posterior is usually intractable

: good approximation is needed, but it often hurts the quality of approximated posterior.

# Variational Inference

Reparameterization trick using variational parameter  $\theta$

$$p(\omega|X, Y) \approx q_\theta(\omega)$$

Kullback-Leibler divergence

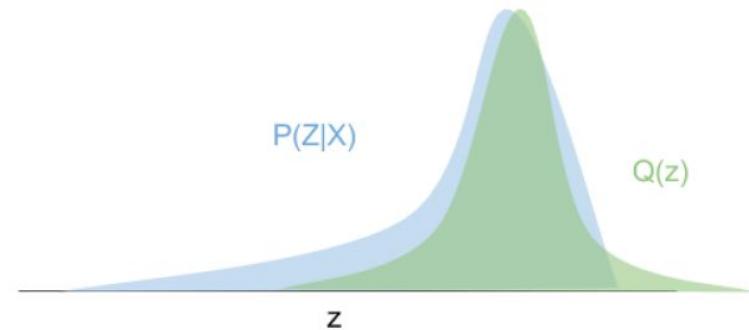
: metric to make two distributions similar

$$\text{KL}(q_\theta(\omega) \| p(\omega|X, Y)) = \int_{\Omega} q_\theta(\omega) \log \frac{q_\theta(\omega)}{p(\omega|X, Y)} d\omega$$

Still intractable because the posterior exists in the KL term.

$$p(\omega|X, Y) = \frac{p(X, Y|\omega)p(\omega)}{p(X, Y)} = p(Y|X, \omega)p(\omega) \frac{p(X)}{p(X, Y)}$$

$$\mathcal{L}_{VI}(\omega) = - \int_{\Omega} q_\theta(\omega) \log p(Y|X, \omega) d\omega + \text{KL}(q_\theta(\omega) \| p(\omega))$$



Minimization objective is equivalent to minimizing the evidence lower-bound (ELBO).

## Variational Inference

Therefore, our minimization objective is

$$\begin{aligned}\text{KL}(q_\theta(\boldsymbol{\omega}) \| p(\boldsymbol{\omega} | \mathbf{X}, \mathbf{Y})) &= - \int_{\Omega} q_\theta(\boldsymbol{\omega}) \log p(\mathbf{y}_i | \mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x}_i)) d\boldsymbol{\omega} + \text{KL}(q_\theta(\boldsymbol{\omega}) \| p(\boldsymbol{\omega})) \\ &= - \sum_{i=1}^N \int_{\Omega} q_\theta(\boldsymbol{\omega}) \log p(\mathbf{y}_i | \mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x}_i)) d\boldsymbol{\omega} + \text{KL}(q_\theta(\boldsymbol{\omega}) \| p(\boldsymbol{\omega}))\end{aligned}$$

Instead of performing computations over the entire dataset, we may use data sub-sampling, also referred to as mini-batch optimization.

$$\begin{aligned}\text{KL}(q_\theta(\boldsymbol{\omega}) \| p(\boldsymbol{\omega} | \mathbf{X}, \mathbf{Y})) &= -\frac{N}{M} \sum_{i \in S}^M \int_{\Omega} q_\theta(\boldsymbol{\omega}) \log p(\mathbf{y}_i | \mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x}_i)) d\boldsymbol{\omega} + \text{KL}(q_\theta(\boldsymbol{\omega}) \| p(\boldsymbol{\omega})) \\ &\rightarrow \frac{1}{M} \sum_{i \in S}^M \int_{\Omega} q_\theta(\boldsymbol{\omega}) \log p(\mathbf{y}_i | \mathbf{f}^{\boldsymbol{\omega}}(\mathbf{x}_i)) d\boldsymbol{\omega} + \frac{1}{N} \text{KL}(q_\theta(\boldsymbol{\omega}) \| p(\boldsymbol{\omega}))\end{aligned}$$

## Variational Inference

For practical applications, the integration over whole parameter space can be replaced to summation of subsampled parameters with a Monte Carlo (MC) estimator.

$$\begin{aligned}\mathcal{L}_{VI}(\theta) &= -\frac{1}{M} \sum_{i \in S} \int_{\Omega} \log p(\mathbf{y}_i | \mathbf{f}^{\omega}(\mathbf{x}_i)) q_{\theta}(\omega) d\omega + \frac{1}{N} \text{KL}(q_{\theta}(\omega) \| p(\omega)) \\ &= -\frac{1}{M} \sum_{i \in S} \int_{\Omega} \log p(\mathbf{y}_i | \mathbf{f}^{g(\theta, \epsilon)}(\mathbf{x}_i)) p(\epsilon) d\epsilon + \frac{1}{N} \text{KL}(q_{\theta}(\omega) \| p(\omega)) \\ \hat{\mathcal{L}}_{MC}(\theta) &= -\frac{1}{M} \sum_{i \in S} \log p(\mathbf{y}_i | \mathbf{f}^{g(\theta, \epsilon)}(\mathbf{x}_i)) + \frac{1}{N} \text{KL}(q_{\theta}(\omega) \| p(\omega))\end{aligned}$$

We can then estimate the predictive distribution with MC integration as well.

$$\begin{aligned}\tilde{q}_{\theta}(\mathbf{y}^* | \mathbf{x}^*) &\coloneqq \frac{1}{T} \sum_{t=1}^T p(\mathbf{y}^* | \mathbf{x}^*, \hat{\omega}_t) \xrightarrow{T \rightarrow \infty} \int p(\hat{\mathbf{y}}^* | \hat{\mathbf{x}}^*, \omega) q_{\theta}(\omega) d\omega \\ &\approx \int p(\hat{\mathbf{y}}^* | \hat{\mathbf{x}}^*, \omega) p(\omega | \mathbf{X}, \mathbf{Y}) d\omega \\ &= p(\hat{\mathbf{y}}^* | \hat{\mathbf{x}}^*, \mathbf{X}, \mathbf{Y})\end{aligned}$$

# Bayesian Uncertainty

## Bayesian inference and uncertainty

We can estimate **the uncertainty** on **our inference** as variance of predictive distribution

$$Var[p(y^*|x^*, X, Y)] \leftarrow p(y^*|x^*, X, Y) = \int_{w \in W} p(y^*|x^*, w)p(w|X, Y)dw$$



*“Knowing what we don’t know” is as important as “knowing itself”.*

# Bayesian Uncertainty

Homoskedastic regression assumes constant noise  $\sigma$  for every input point  $x$ .

On the other hand, we draw model weights from the approximate posterior  $\hat{\omega} \sim q(\omega)$  to obtain a model output, this time composed of both predictive mean as well as predictive variance:

$$[\hat{y}_i, \hat{\sigma}_i^2] = f^{\hat{\omega}}(x_i)$$

**Heteroskedastic regression**, on the other hand, assumes that observation noise can vary with input  $x$ .

We fix a Gaussian likelihood to model our aleatoric uncertainty. This induces a minimization objective given labeled output points  $x$ :

$$\mathcal{L}_{NN}(\theta) = \frac{1}{N} \sum_{i=1}^N \frac{1}{2\sigma_i^2} \|\mathbf{y}_i - \hat{\mathbf{y}}_i\|^2 + \frac{1}{2} \log \sigma_i^2$$

To summarize, the predictive uncertainty can be approximated using:

$$Var(y) \approx \frac{1}{T} \sum_{t=1}^T \hat{y}_t^2 - \left( \frac{1}{T} \sum_{t=1}^T \hat{y}_t \right)^2 + \frac{1}{T} \sum_{t=1}^T \hat{\sigma}_t^2$$

**Epistemic uncertainty**

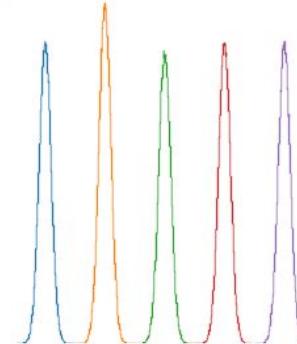
**Aleatoric uncertainty**

# Bayesian Uncertainty

Low epistemic, Low aleatoric

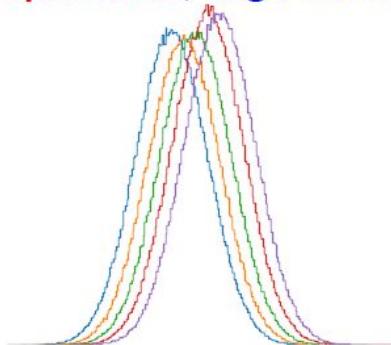


High epistemic, Low aleatoric

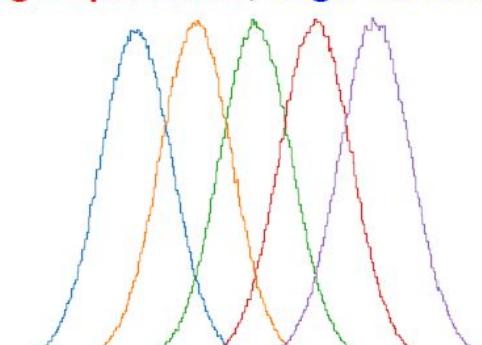


$$Var(y) \approx \frac{1}{T} \sum_{t=1}^T \hat{y}_t^2 - \left( \frac{1}{T} \sum_{t=1}^T \hat{y}_t \right)^2 + \frac{1}{T} \sum_{t=1}^T \hat{\sigma}_t^2$$

Low epistemic, High aleatoric



High epistemic, High aleatoric



## Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning

ICML 2015

**Yarin Gal**  
**Zoubin Ghahramani**  
University of Cambridge

YG279@CAM.AC.UK  
ZG201@CAM.AC.UK

```
import math
import numpy as np

logit_list = []
for _ in range(num_samplings):
    logit = model(x, training=True)
    logit_list.append(logit)

# Regression case
predictive_mean = np.mean(logit_list, axis=0)
predictive_unc = np.std(logit_list, axis=0)

# Classification case
predictive_mean = np.mean(logit_list, axis=0)
predictive_unc = math.sqrt(predictive_mean*(1.-predictive_mean))
```

- Inference phase에서 Dropout을 키고 T번 output을 sampling (MC-sampling)
- 오른쪽의 식과 같이 regression/classification에 대해서 predictive mean/uncertainty를 계산

# Bayesian Inference in practice: Stochastic Weight Averaging

---

## Averaging Weights Leads to Wider Optima and Better Generalization

---

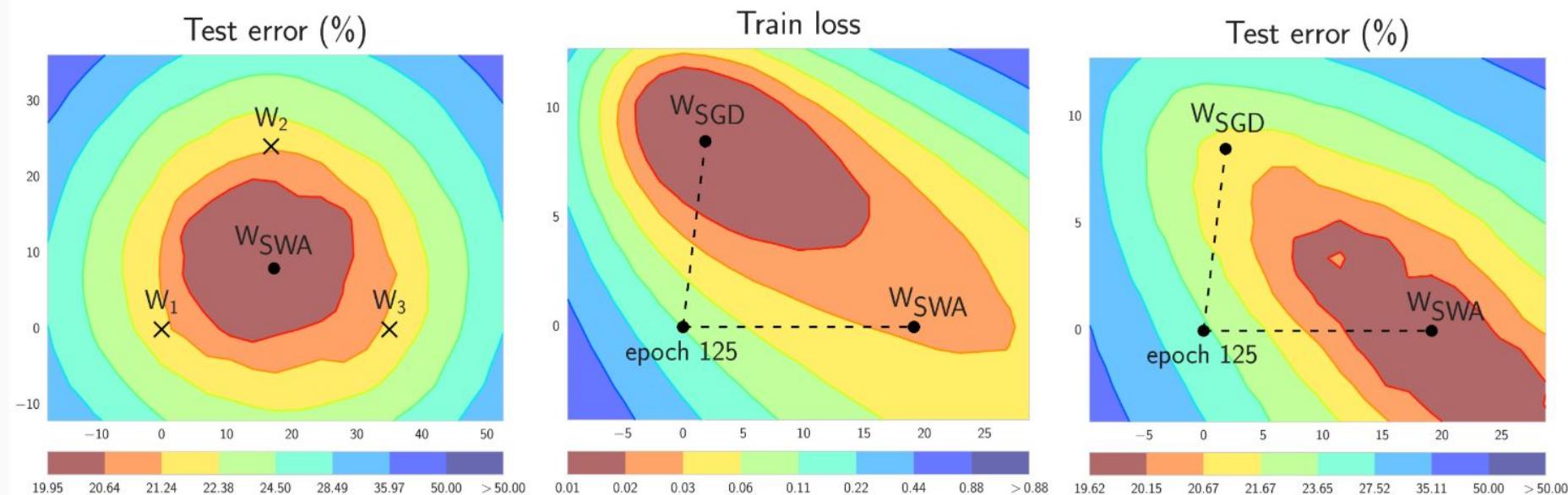
**Pavel Izmailov<sup>\*1</sup>**   **Dmitrii Podoprikhin<sup>\*2,3</sup>**   **Timur Garipov<sup>\*4,5</sup>**   **Dmitry Vetrov<sup>2,3</sup>**   **Andrew Gordon Wilson<sup>1</sup>**  
<sup>1</sup>Cornell University, <sup>2</sup>Higher School of Economics, <sup>3</sup>Samsung-HSE Laboratory,  
<sup>4</sup>Samsung AI Center in Moscow, <sup>5</sup>Lomonosov Moscow State University

April 29, 2019

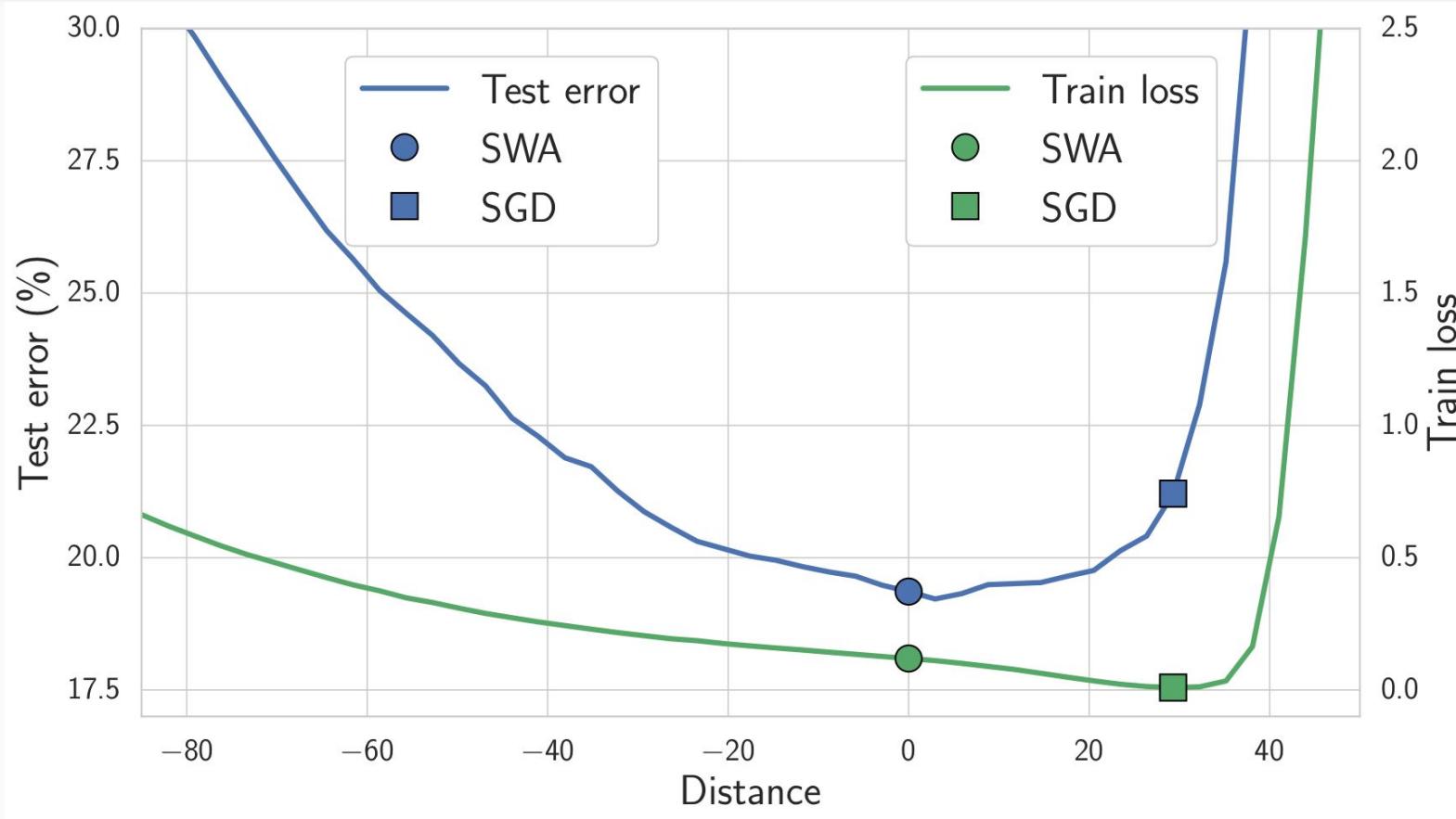
# Stochastic Weight Averaging in PyTorch

Official PyTorch Blog: <https://pytorch.org/blog/stochastic-weight-averaging-in-pytorch/>

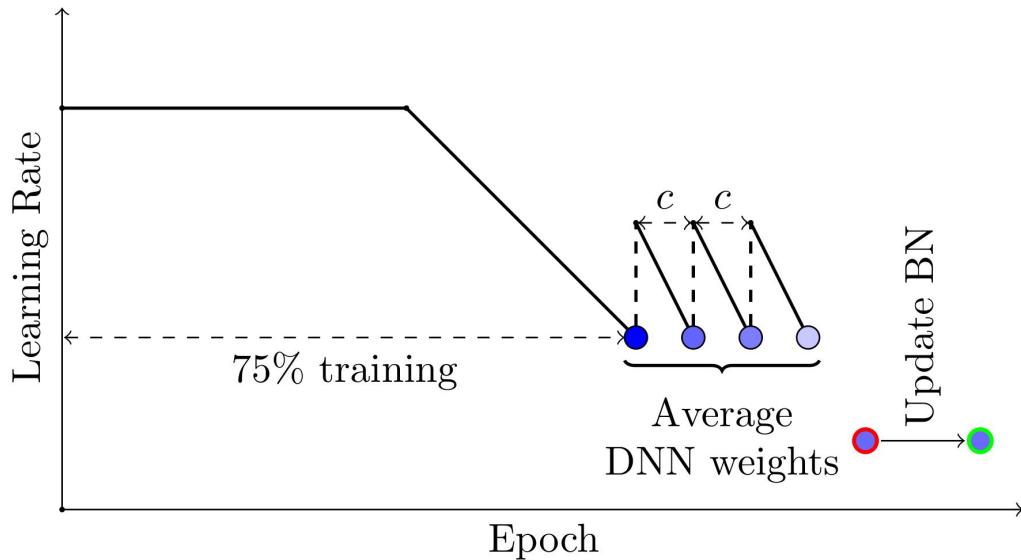
# Bayesian Inference in practice: Stochastic Weight Averaging



# Bayesian Inference in practice: Stochastic Weight Averaging



# Bayesian Inference in practice: Stochastic Weight Averaging



```
opt = torchcontrib.optim.SWA(base_opt)
for i in range(100):
    opt.zero_grad()
    loss_fn(model(input), target).backward()
    opt.step()
    if i > 10 and i % 5 == 0:
        opt.update_swa()
opt.swap_swa_sgd()
```

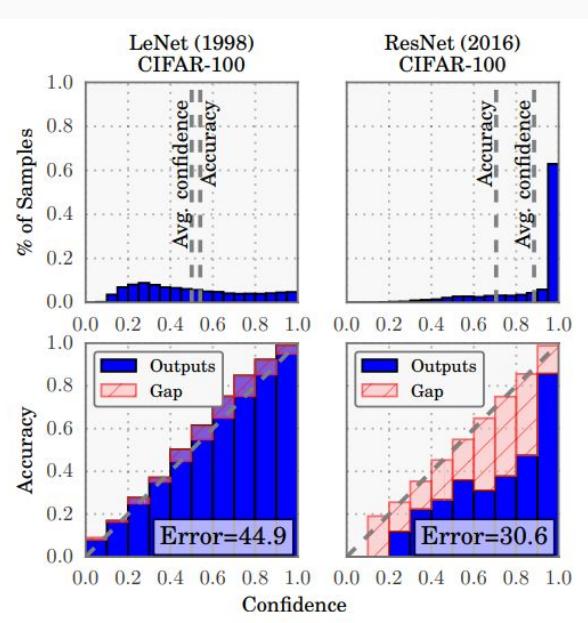
- We will practice with the updated version of SWA implementation.

# How to evaluate reliability of models? -- Expected Calibration Error

## On Calibration of Modern Neural Networks

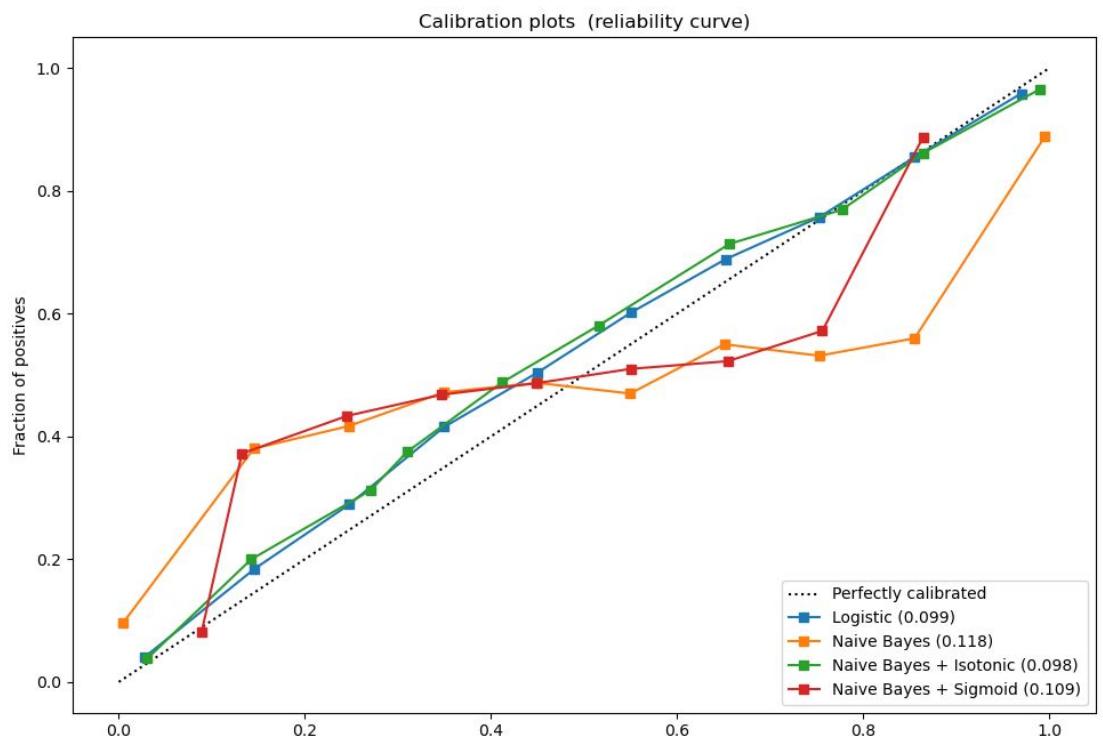
ICML 2017

Chuan Guo \*<sup>1</sup> Geoff Pleiss \*<sup>1</sup> Yu Sun \*<sup>1</sup> Kilian Q. Weinberger<sup>1</sup>



- 초기 CNN 구조인 LeNet (1998)에 비해 ResNet (2016)은 Accuracy가 많이 개선됨. (Error: 44.9% vs 30.6% for CIFAR-100)
- 하지만 모델 parameter 갯수가 크게 증가하였고,
- 특정 class에 속할 확률을 over-estimation하는 경향이 있음  
→ Over-confident prediction problem
- “어떻게 모델의 over-confidence 여부, 즉 reliability (신뢰도)를 평가할 수 있을까?”

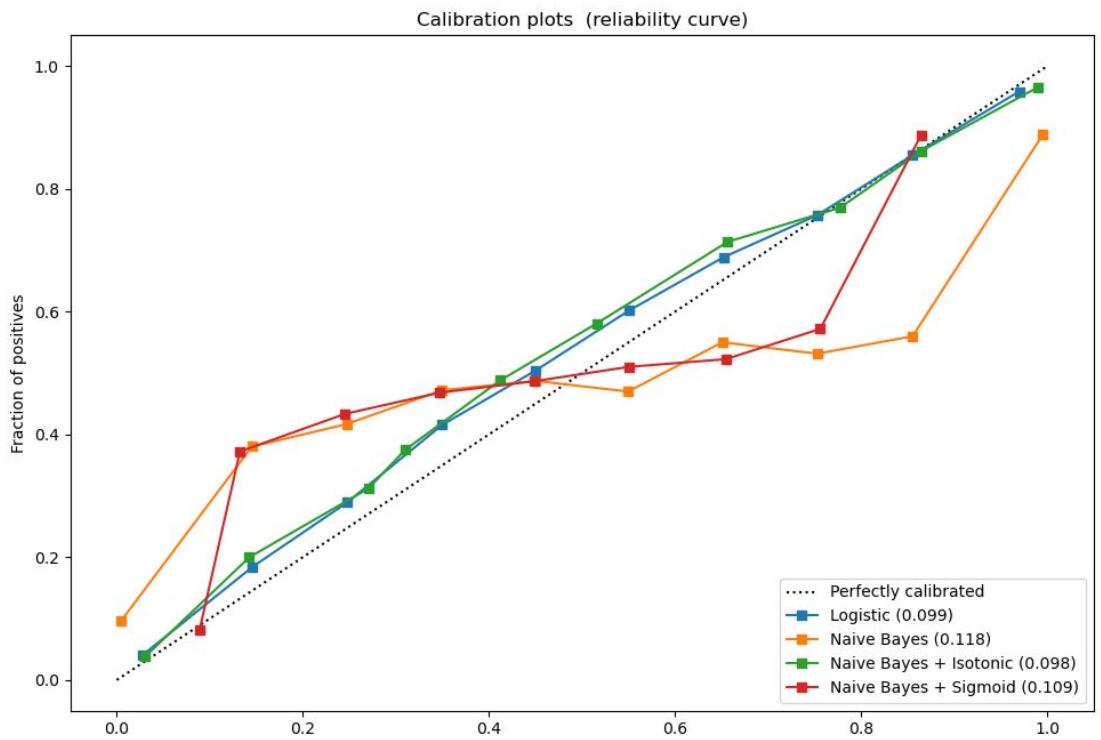
# How to evaluate reliability of models? -- Expected Calibration Error



- 점선: perfect calibration line
- Output 값이 0.70이면 class=1 일 확률이 70%, 0.90이면 class=1 일 확률이 90%
- 색선: model들의 prediction result
- 점선과 색선의 deviation 차이  
→ Expected Calibration Error (ECE)

[https://scikit-learn.org/stable/auto\\_examples/calibration/plot\\_calibration\\_curve.html](https://scikit-learn.org/stable/auto_examples/calibration/plot_calibration_curve.html)

# How to evaluate reliability of models? -- Expected Calibration Error



Perfect calibration case)

$$\mathbb{P}(\hat{Y} = Y \mid \hat{P} = p) = p, \quad \forall p \in [0, 1]$$

Perfect calibration과 실제 model results의 차이 → calibration error

$$\mathbb{E}_{\hat{P}} \left[ \left| \mathbb{P}(\hat{Y} = Y \mid \hat{P} = p) - p \right| \right]$$

실제 histogram 방식으로 구하는 경우

$$ECE = \sum_{m=1}^M \frac{|B_m|}{n} \left| \text{acc}(B_m) - \text{conf}(B_m) \right|,$$

## How to evaluate reliability of models? -- Expected Calibration Error

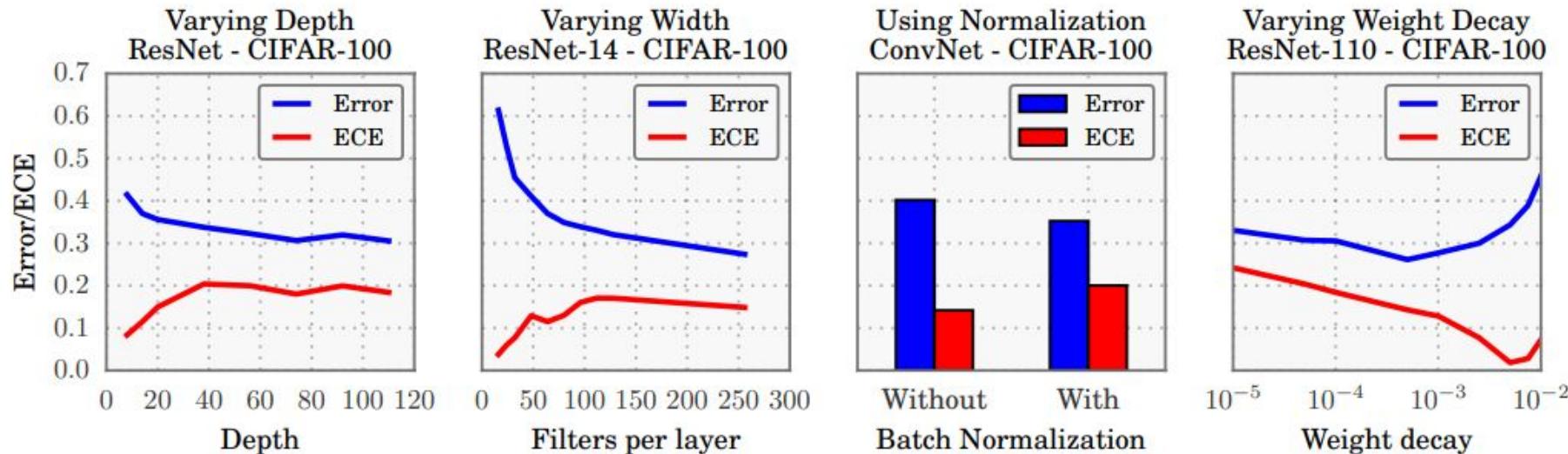


Figure 2. The effect of network depth (far left), width (middle left), Batch Normalization (middle right), and weight decay (far right) on miscalibration, as measured by ECE (lower is better).

- 모델을 개선시키는 방법들 -- layer depth/width 늘리기, Batch normalization, weight-decay -- 을 사용할 경우 accuracy는 좋아지지만 reliability는 떨어지는 현상 (ECE는 증가하는 현상) 이 종종 관측됨.

# How to evaluate reliability of models? -- Expected Calibration Error

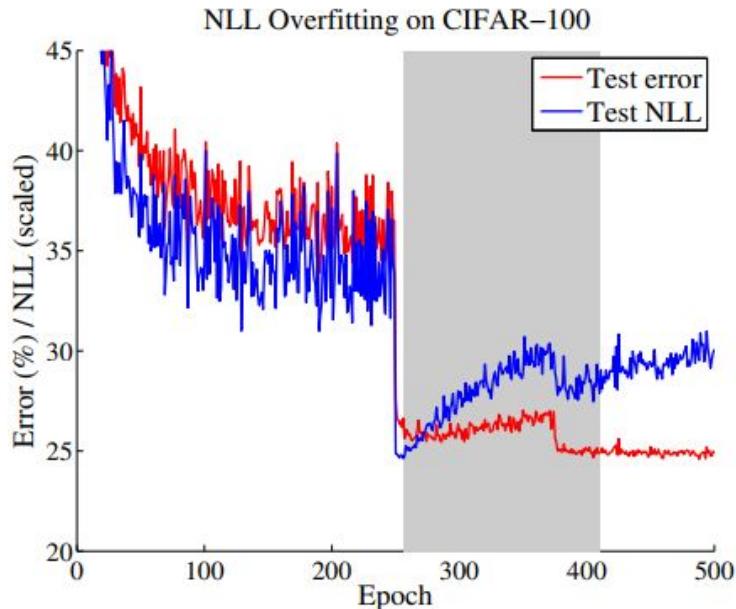


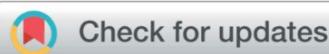
Figure 3. Test error and NLL of a 110-layer ResNet with stochastic depth on CIFAR-100 during training. NLL is scaled by a constant to fit in the figure. Learning rate drops by 10x at epochs 250 and 375. The shaded area marks between epochs at which the best validation *loss* and best validation *error* are produced.

- 모델을 계속 트레이닝하다보면 (training epoch이 지나다보면)
  - Test error는 어느순간부터 더 떨어지지 않음.
  - 하지만 negative log-likelihood (NLL)은 증가함.
  - 이런 결과는 **reliability**를 떨어트리는 결과로 이어짐  
→ SWA는 이런 NLL over-fitting을 방지함.

# **Application of GNNs and Bayesian learning in molecular property prediction**

### EDGE ARTICLE

[View Article Online](#)  
[View Journal](#) | [View Issue](#)



Cite this: *Chem. Sci.*, 2019, 10, 8438

All publication charges for this article have been paid for by the Royal Society of Chemistry

## A Bayesian graph convolutional network for reliable prediction of molecular properties with uncertainty quantification†

Seongok Ryu,<sup>a</sup> Yongchan Kwon <sup>b</sup> and Woo Youn Kim \*ac

# Applications in molecular property predictions

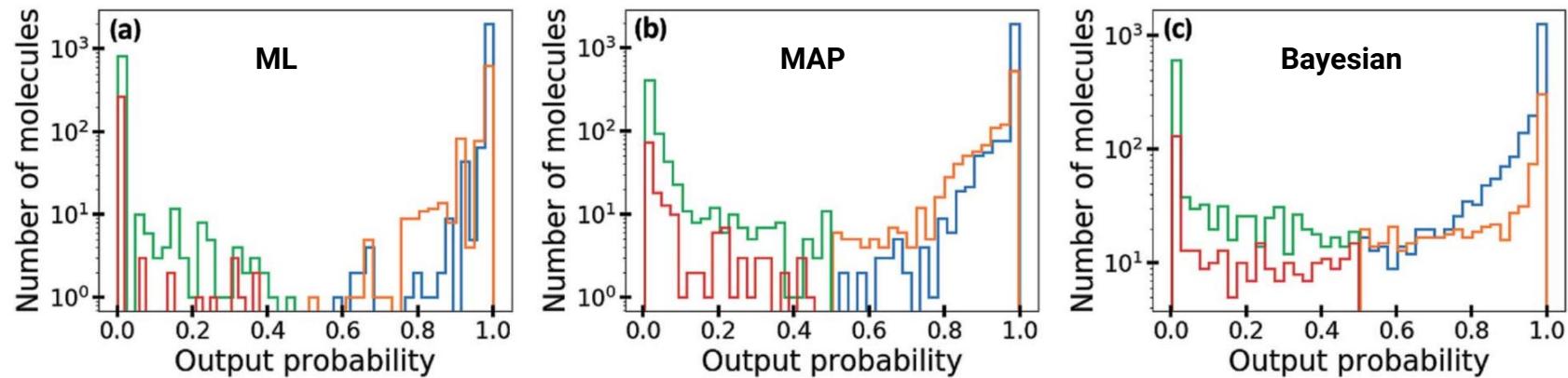


Fig. 5 Distributions of output probability obtained by (a) the ML, (b) the MAP and (c) the Bayesian GCNs. The total distribution is divided into true positive (blue), false positive (orange), true negative (green) and false negative (red) groups. Note that the y-axis is represented with a log scale.

- Good classification model에 기대하는 바:
  - 0.0으로 가까워질수록 **True negative** 는 많고, **False negative** 는 적고
  - 1.0으로 가까워질수록 **True positive** 는 많고, **False positive** 는 적고
- Bayesian model 은 그런 경향성을 보이나, ML/MAP model은 그렇지 못함
  - 따라서 1.0 (0.0)에 가까운 값을 취사선택하면, **True positive (negative)** 를 선택할 확률이 더 높아짐
  - Virtual screening 을 위한 classification model 개발에 Bayesian learning 을 응용하는 것이 적합함.

# Applications in molecular property predictions

- Clean Energy Project (CEP) dataset 의 Photo-conversion efficiency (PCE) 데이터에는 0.0 으로 label 된 데이터가 있었음
- 해당 데이터의 일부를 이용하여 Bayesian regression model을 training
- Test set에 inference 하여보니 0.0으로 label 된 데이터는 aleatoric uncertainty가 매우 커고, 해당 데이터가 mis-label 된 것을 의심
- Simulation을 통해 PCE 값을 얻는 과정을 살펴보니, 잘못된 근사법으로 인해 얻어진 데이터임을 확인하였음.

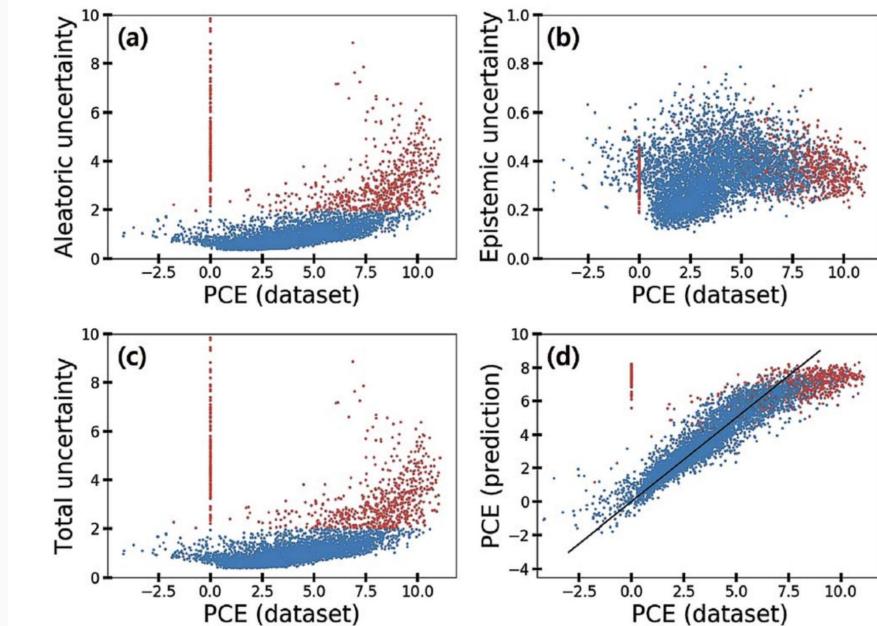


Fig. 7 (a) Aleatoric, (b) epistemic, and (c) total uncertainties and (d) predicted PCE value against the PCE value in the dataset. The samples colored in red show a total uncertainty greater than two.

## Comprehensive Study on Molecular Supervised Learning with Graph Neural Networks

Doyeong Hwang, Soojung Yang, Yongchan Kwon, Kyung Hoon Lee, Grace Lee, Hanseok Jo, Seyeol Yoon, and Seongok Ryu\*



Cite This: <https://dx.doi.org/10.1021/acs.jcim.0c00416>



Read Online

# Applications in molecular property predictions

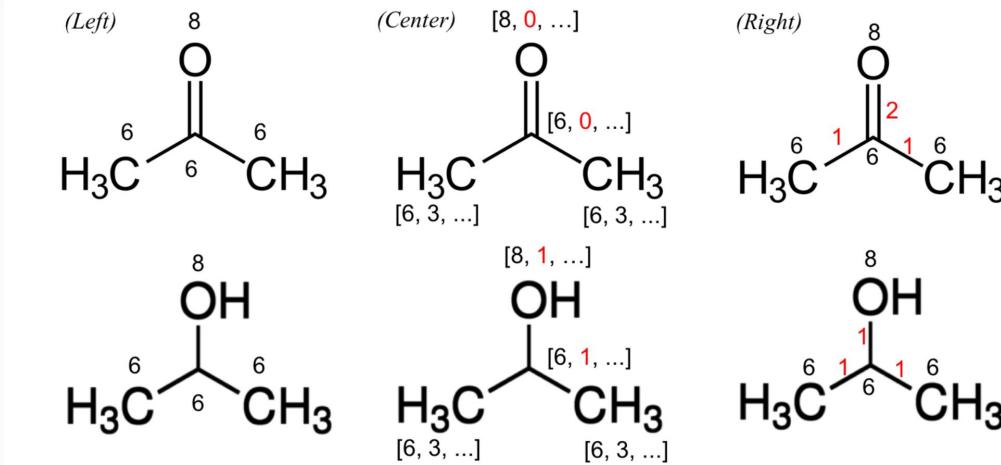


Figure 1: Two different molecules, propanone (*top*) and 2-propanol (*bottom*), featurized by three approaches. (*Left*) Using atomic types for node features and connectivity for edge features. This input digitization cannot distinguish the molecules. (*Middle*) Using atomic types and number of attached hydrogens (and additional meta-information; colored in red) for node features and connectivity for edge features. (*Right*) Using atomic types for node features and bond types (colored in red) for edge features. The last two digitization methods can distinguish the molecules.

- 2-propanone vs 2-propanol (ketone vs alcohol – which show completely different chemistry)
- # of hydrogen, bond order 등을 meta-information 으로 사용하지 않고, atomic number 만 사용하면 위 두 분자는 구분되지 않음.

## Applications in molecular property predictions

Table 1: Mean absolute error of pLogP prediction results of the models using input expression utilizing the atomic meta-information or not. We report mean and standard deviation of results from eight different experiments with scaffold-splitting of the datasets. We note that GatedGCN and GGNN utilize bond features and the other GNNs do not.

	Using meta-info	No meta-info
GCN	0.3431 (0.0136)	1.0387 (0.0118)
GraphSAGE	0.2786 (0.0010)	0.9937 (0.0062)
GIN	0.3134 (0.0093)	1.0301 (0.0118)
GAT	0.3268 (0.0057)	1.0398 (0.0163)
GatedGCN	0.2549 (0.0081)	0.5747 (0.0179)
GGNN	<b>0.2532 (0.0053)</b>	<b>0.5087 (0.0059)</b>

- 따라서 # of hydrogen, bond order와 같은 meta-information을 사용하는 것이 pLogP toy experiment에서 더 좋은 성능을 보임.  
→ Molecular graph를 표현하는 방법은 다양하니, 어떻게 featurize할지를 잘 고민하는 것이 중요하다!

## Applications in molecular property predictions

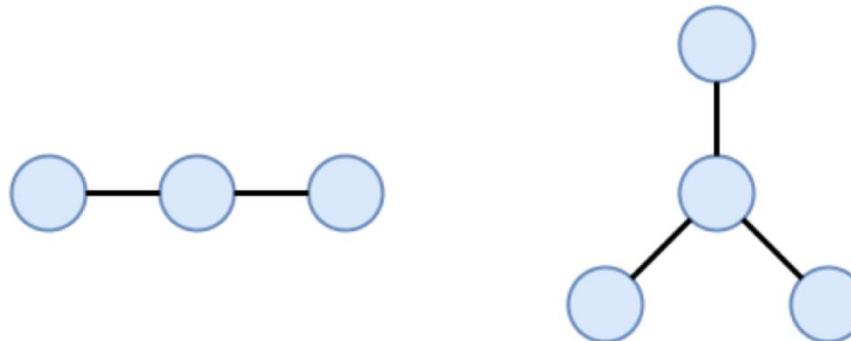


Figure 2: Two graphs each of which consists of identical nodes, but with three and four nodes, respectively.

- Understanding isomorphism test
- Node-feature updating 에서 node feature 들을 aggregation 할 때 sum, mean 중 어떤 연산을 사용해야 할까?
- Readout (graph-feature updating) 에서 node feature 들을 aggregation 할 때, sum mean 중 어떤 연산을 사용해야 할까?

## Applications in molecular property predictions

Table 2: Mean absolute error of pLogP predictions of the GNNs whose node-updating layers aggregate messages with sum and mean aggregations. We report mean and standard deviation of results from eight different experiments with scaffold-splitting of the datasets.

	Sum aggregation	Mean aggregation
GCN	<b>0.3431 (0.0136)</b>	0.3524 (0.0121)
GraphSAGE	<b>0.2786 (0.0010)</b>	0.2943 (0.0055)
GIN	<b>0.3134 (0.0093)</b>	0.3194 (0.0167)

- Understanding isomorphism test
- Node-feature updating 에서 node feature 를 aggregation 할 때 sum, mean 중 어떤 연산을 사용해야 할까?  
→ Sum aggregation이 더 좋은 성능 (더 낮은 RMS error) 을 보임

## Applications in molecular property predictions

Table 3: Mean absolute error of pLogP prediction results of the models using the sum, mean, and max readouts. We report mean and standard deviation of results from eight different experiments with scaffold-splitting of the datasets.

	Sum	Mean	Max
GCN	<b>0.3431 (0.0136)</b>	0.3696 (0.0110)	0.4209 (0.0125)
GraphSAGE	<b>0.2786 (0.0100)</b>	0.2818 (0.0088)	0.3565 (0.0061)
GIN	<b>0.3134 (0.0093)</b>	0.3407 (0.0101)	0.3791 (0.0077)
GAT	<b>0.3268 (0.0057)</b>	0.3672 (0.0093)	0.3863 (0.0064)
GatedGCN	<b>0.2549 (0.0081)</b>	0.2883 (0.0089)	0.2791 (0.0086)
GGNN	<b>0.2532 (0.0053)</b>	0.2920 (0.0076)	0.2833 (0.0036)

- Understanding isomorphism test
- Readout (graph-feature updating) 에서 node feature 들을 aggregation 할 때, sum mean 중 어떤 연산을 사용해야 할까?
  - Sum aggregation이 더 좋은 성능 (더 낮은 RMS error) 을 보임

# Applications in molecular property predictions

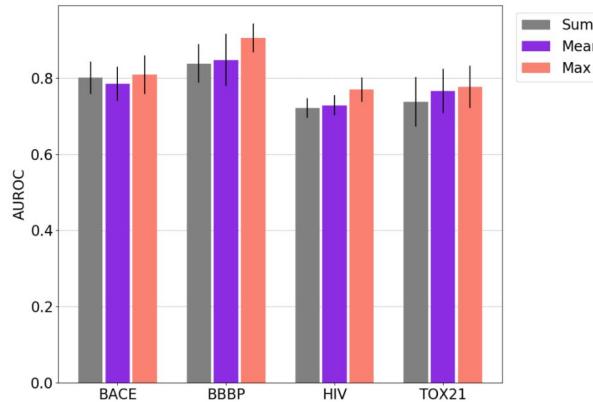


Figure 5: Prediction performance (AUROC) of the GIN models with sum, mean, and max readouts on the four classification tasks; BACE, BBBP, HIV and Tox21 predictions. We report mean and standard deviation of results from eight different experiments with scaffold-splitting of the datasets.

- 하지만 (binary) classification model에서는 max readout 0l sum, mean 보다 더 좋은 성능 (더 높은 AUROC) 을 보임
- Why?
  - Classification 에서는 common pattern (common substructure) 를 찾는 식으로 모델이 학습되는 것으로 추측됨.
  - Max readout 은 common pattern 을 찾는 것에 가장 뛰어난 성능을 보임.

# Applications in molecular property predictions

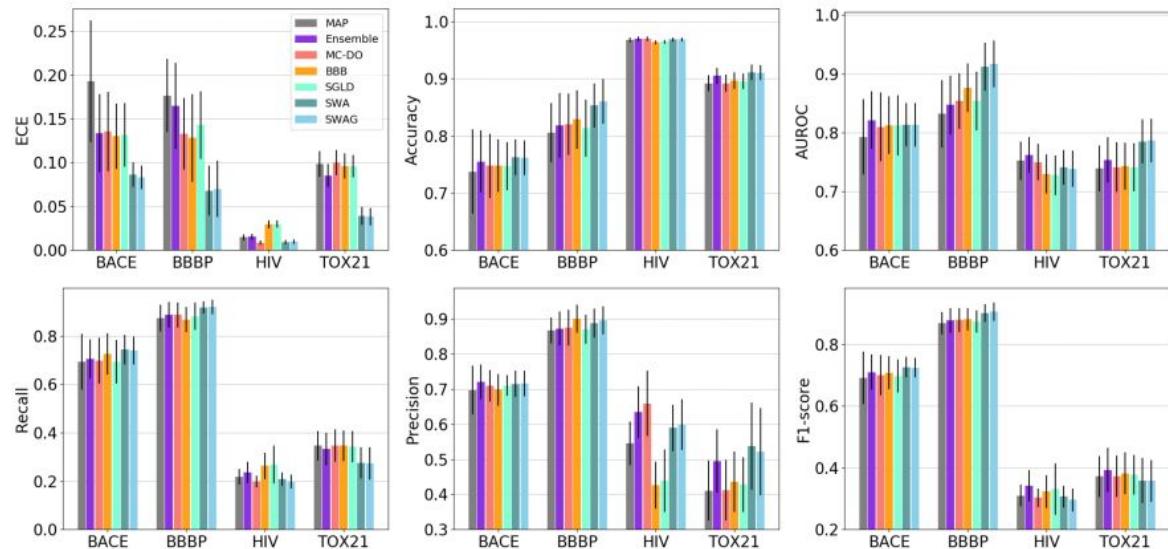


Figure 8: The prediction reliability (ECE; ↓) and performance (accuracy, AUROC, precision, recall and F1-score; ↑) of the GIN model for the BACE, BBBP, HIV, and Tox21 prediction tasks. We report mean and standard deviation of results from eight different experiments with scaffold-splitting of the datasets.

- 다양한 approximate Bayesian learning 방법론들을 구현하고 benchmark test
- SWA, SWAG 가 전반적으로 가장 우수한 성능을 보임.

# Applications in molecular property predictions

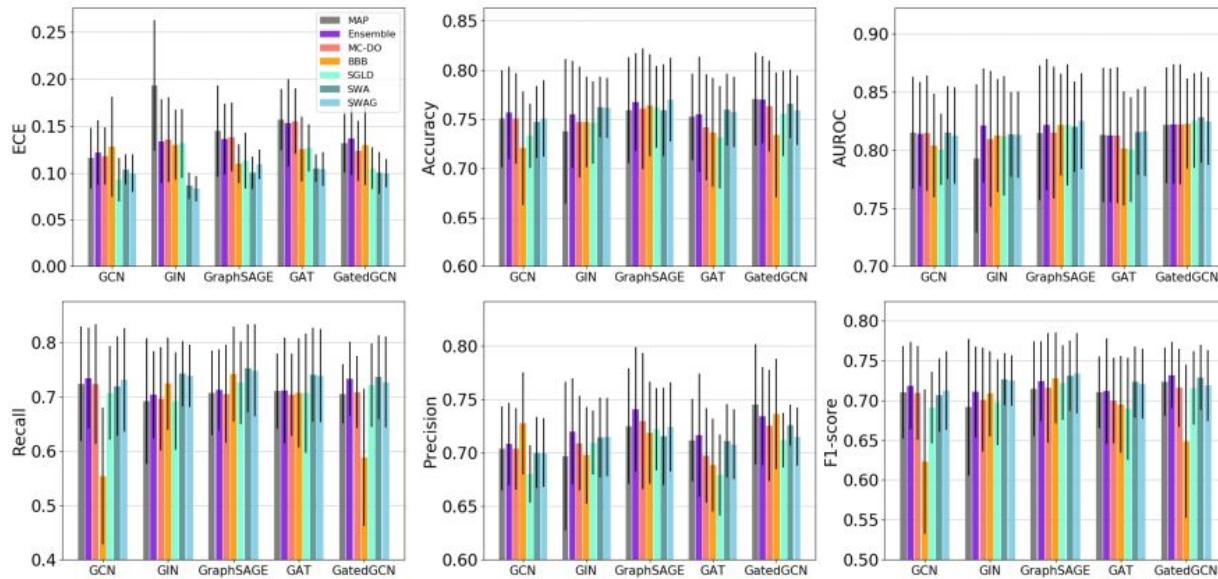


Figure 9: The prediction reliability (ECE; ↓) and performance (accuracy, AUROC, precision, recall and F1-score; ↑) of the five different GNN models on the BACE prediction task. We report mean and standard deviation of results from eight different experiments with scaffold-splitting.

- 다양한 approximate Bayesian learning 방법론들을 구현하고 benchmark test
- SWA, SWAG 가 전반적으로 가장 우수한 성능을 보임.

# Applications in molecular property predictions

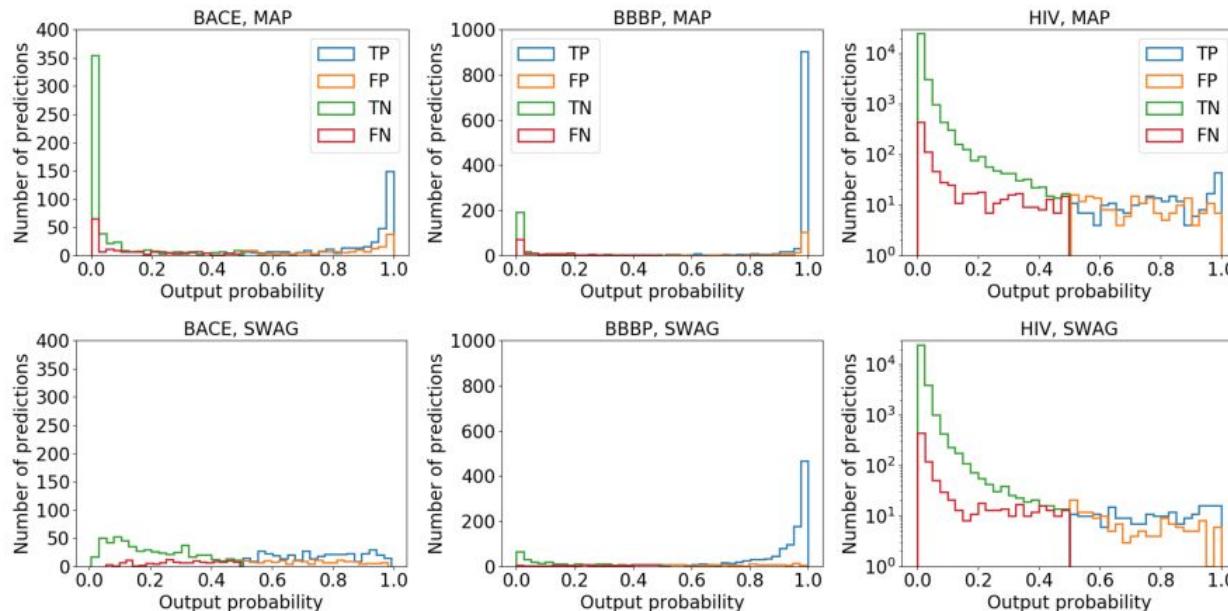


Figure 10: The histograms of true positive (TP), false positive (FP), true negative (TN), and false negative (FN) predictions from the GIN trained with MAP (top) and SWAG (bottom).

- BACE, BBBP, HIV dataset들에 대해서, MAP model estimation 보다 SWAG를 이용한 Bayesian learning이 virtual screening에 적합한 예측 경향성 결과를 보여줌.

# Accurate, reliable and interpretable solubility prediction of druglike molecules with advanced machine learning techniques

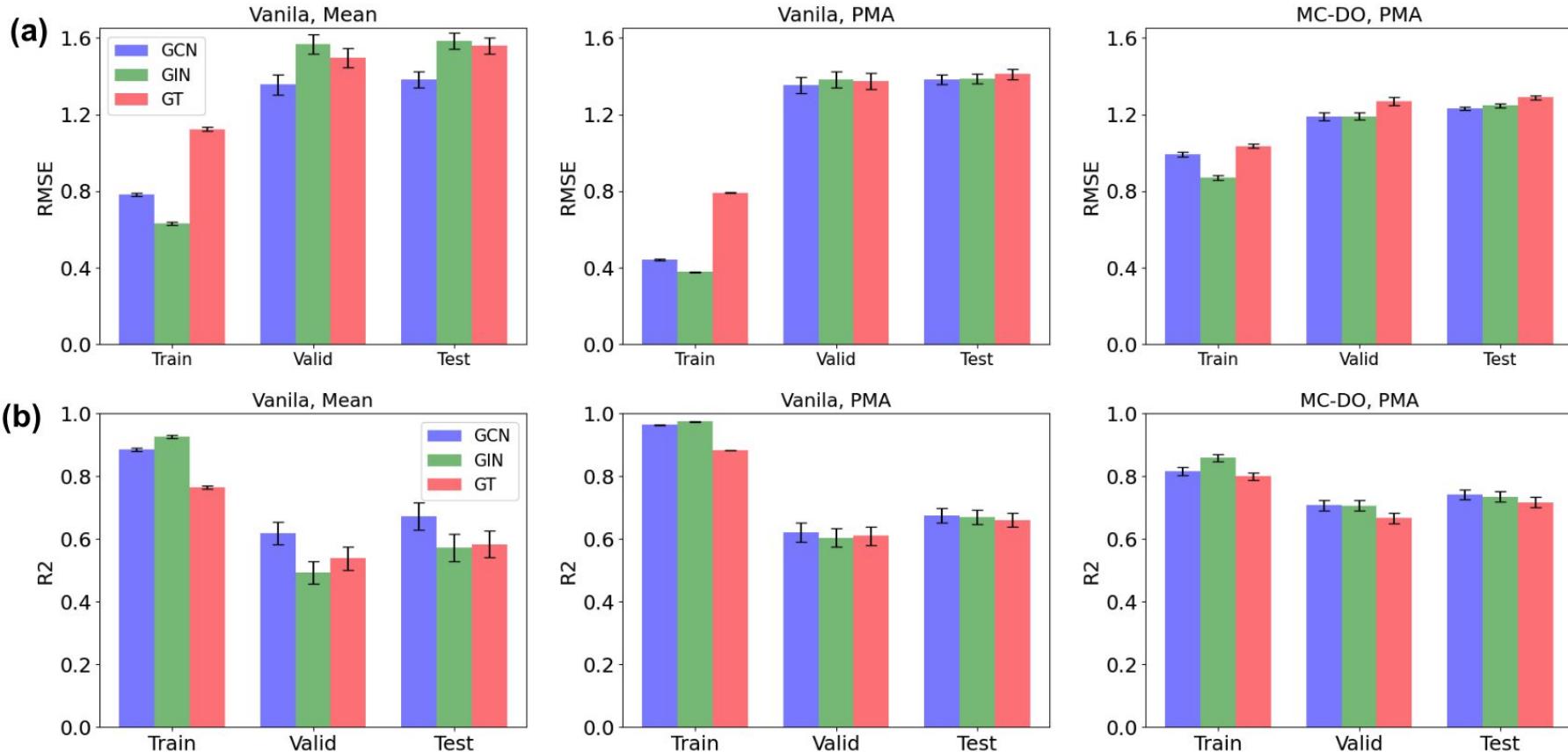
Seongok Ryu,<sup>†</sup> Sumin Lee,<sup>‡</sup> and Cha-ok Seok<sup>\*,†,‡</sup>

<sup>†</sup>*GALUX inc., Seoul, Republic of Korea*

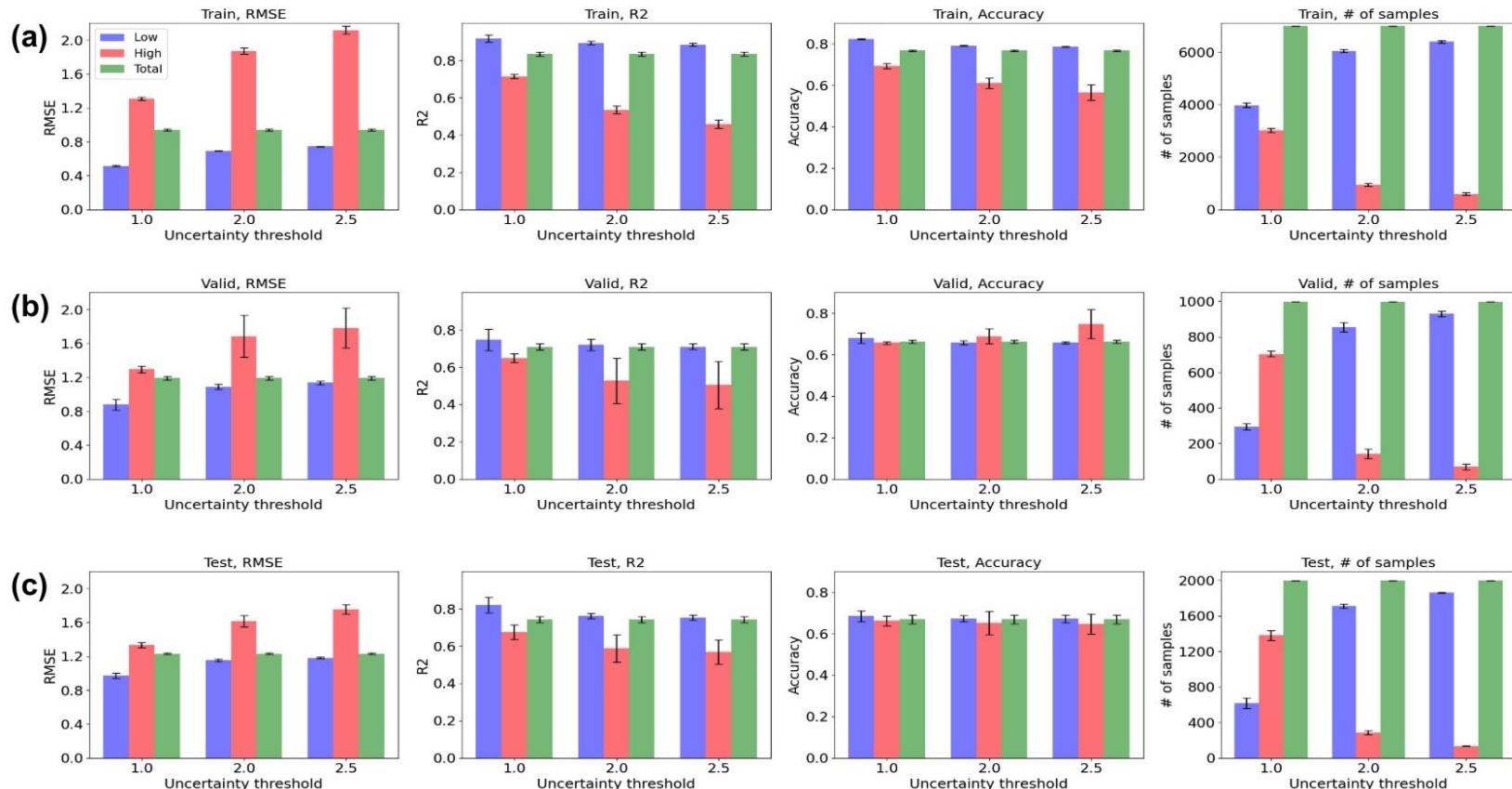
<sup>‡</sup>*Seoul National University, Seoul, Republic of Korea*

- Unpublished results, in preparation
- The title and contents can be revised

# Applications in molecular property predictions

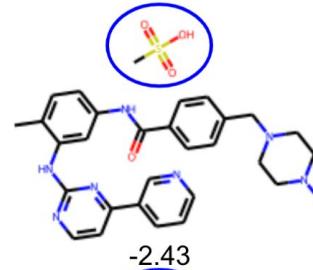
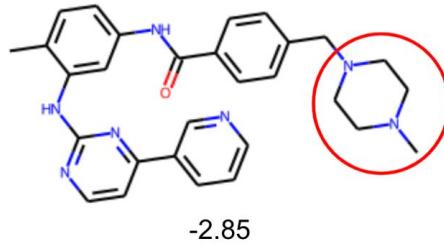
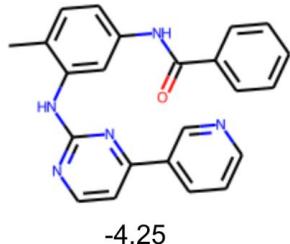


# Applications in molecular property predictions

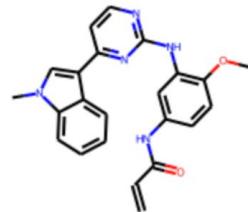


# Applications in molecular property predictions

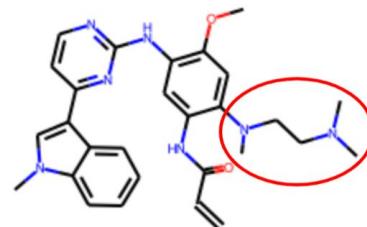
Imatinib



Osimertinib



-4.21

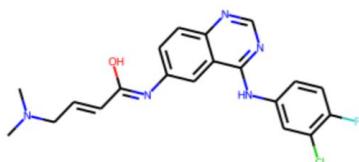


-3.58

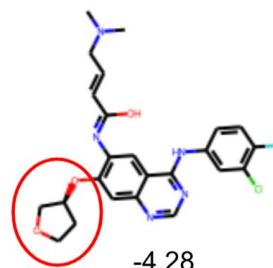


-2.96

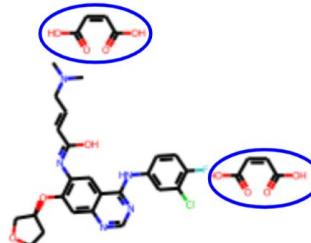
Afatinib



-5.3



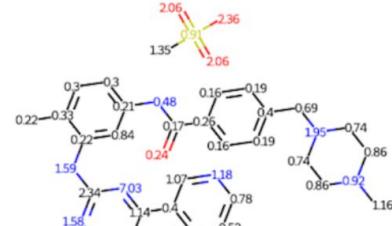
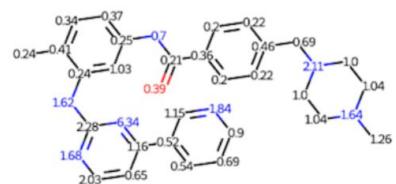
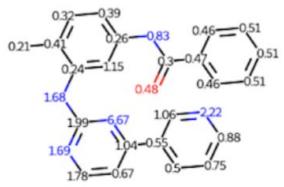
-4.28



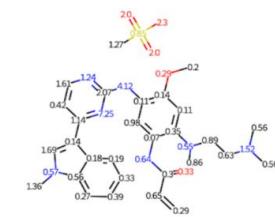
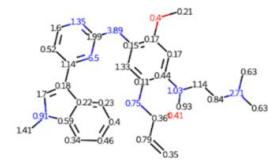
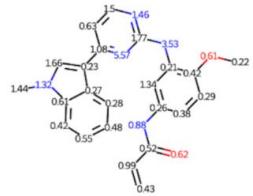
-3.56

# Applications in molecular property predictions

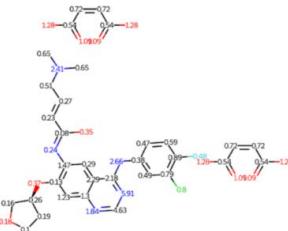
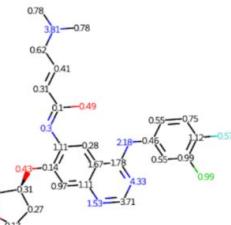
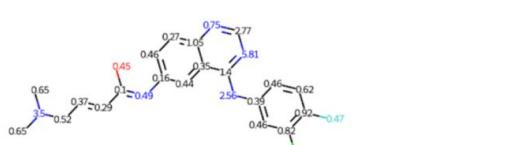
Imatinib



Osimertinib



Afatinib



**Thanks for your attention!**